PageRank

- Alternative link-analysis method used by Google (Brin & Page, 1998).
- Does not attempt to capture the distinction between hubs and authorities.
- Ranks pages just by authority.
- Applied to the entire web rather than a local neighborhood of pages surrounding the results of a query.

Initial PageRank Idea

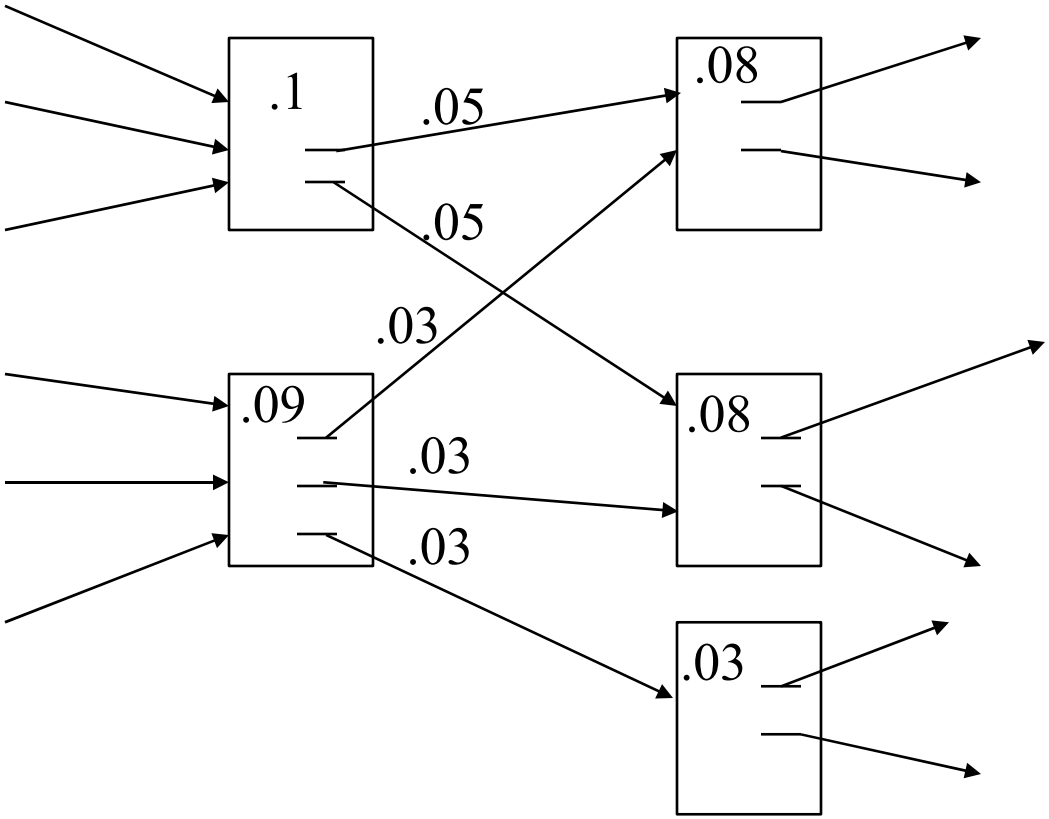
- Just measuring in-degree (citation count) doesn't account for the authority of the source of a link.
- Initial page rank equation for page *p*:

$$R(p) = c \sum_{q:q \to p} \frac{R(q)}{N_q}$$

  - N_q is the total number of out-links from page *q*.
  - A page, *q*, "gives" an equal fraction of its authority to all the pages it points to (e.g. *p*).
  - *c* is a normalizing constant set so that the rank of all pages always sums to 1.

Initial PageRank Idea (cont.)

- Can view it as a process of PageRank "flowing" from pages to the pages they cite.



Initial Algorithm

- Iterate rank-flowing process until convergence:

  Let S be the total set of pages.

  Initialize: $\forall p \in S: R(p) = 1/|S|$

  Until ranks do not change (much) *(convergence)*

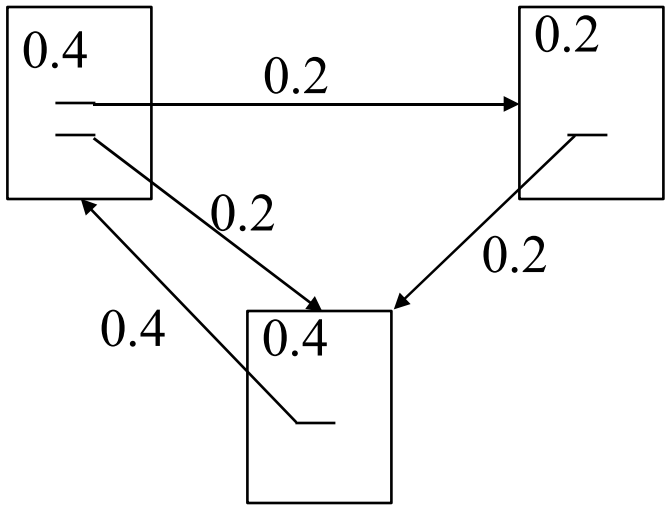
  For $\forall p \in S$ compute:

$$R'(p) = \sum_{q:q \to p} \frac{R(q)}{N_q}$$

$$c = 1 / \sum_{p \in S} R'(p)$$

$$\forall p \in S: R(p) = cR'(p) \quad \textit{(normalize)}$$

Sample Stable Fixpoint

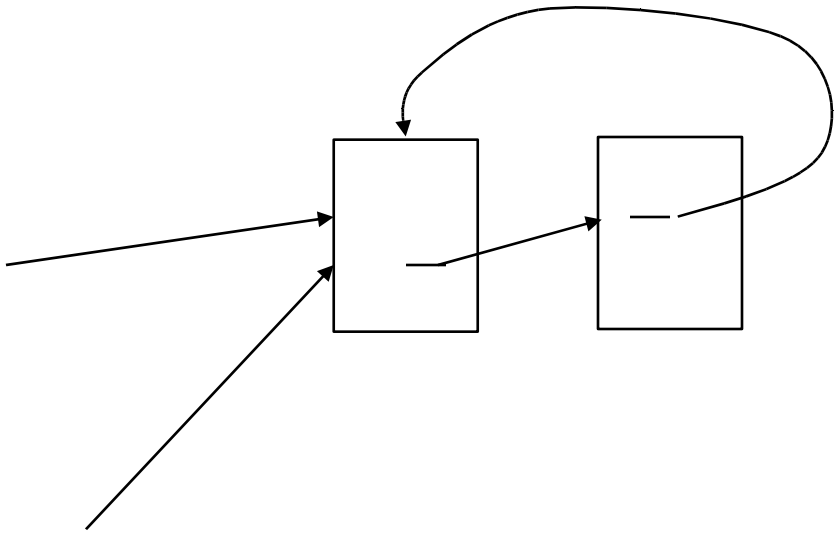


Linear Algebra Version

- Treat **R** as a vector over web pages.
- Let **A** be a 2-d matrix over pages where
  - $\mathbf{A}_{vu} = 1/N_u$ if there is a link from v to u else $\mathbf{A}_{vu} = 0$
- Then $\mathbf{R} = c\mathbf{AR}$
- **R** converges to the principal eigenvector of **A**.

Problem with Initial Idea

- A group of pages that only point to themselves but are pointed to by other pages act as a "rank sink" and absorb all the rank in the system.



Rank flows into cycle and can't get out

Rank Source

- Introduce a "rank source" E that continually replenishes the rank of each page, p, by a fixed amount $E(p)$.

$$R(p) = c\left(\sum_{q : q \to p} \frac{R(q)}{N_q} + E(p) \right)$$

PageRank Algorithm

Let S be the total set of pages.

Let $\forall p \in S$: $E(p) = q/|S|$ (for some $0<q<1$, e.g. 0.15)

Initialize $\forall p \in S$: $R(p) = 1/|S|$

Until ranks do not change (much) (*convergence*)

For each p S:

$$R'(p) = \sum_{q:q \to p} \frac{R(q)}{N_q} + E(p)$$

$$c = 1 / \sum_{p \in S} R'(p)$$

For each $p \in S$: $R(p) = cR'(p)$ (*normalize*)

Linear Algebra Version

- $\mathbf{R} = c(\mathbf{AR} + \mathbf{E})$
- Since $\|\mathbf{R}\|_1 = 1$: $\mathbf{R} = c(\mathbf{A} + \mathbf{E}\ \mathbf{1})\mathbf{R}$
  - Where **1** is the vector consisting of all 1's.
- So **R** is an eigenvector of $(\mathbf{A} + \mathbf{E} \times \mathbf{1})$

Random Surfer Model

- PageRank can be seen as modeling a "random surfer" that starts on a random page and then at each point:
  - With probability $E(p)$ randomly jumps to page p.
  - Otherwise, randomly follows a link on the current page.
- $R(p)$ models the probability that this random surfer will be on page p at any given time.
- "E jumps" are needed to prevent the random surfer from getting "trapped" in web sinks with no outgoing links.

Speed of Convergence

- Early experiments on Google used 322 million links.
- PageRank algorithm converged (within small tolerance) in about 52 iterations.
- Number of iterations required for convergence is empirically O(log n) (where n is the number of links).
- Therefore calculation is quite efficient.

Simple Title Search with PageRank

- Use simple Boolean search to search web-page titles and rank the retrieved pages by their PageRank.
- Sample search for “university”:
  - Altavista returned a random set of pages with “university” in the title (seemed to prefer short URLs).
  - Primitive Google returned the home pages of top universities.

Google Ranking

- Complete Google ranking includes (based on university publications prior to commercialization).
  - Vector-space similarity component.
  - Keyword proximity component.
  - HTML-tag weight component (e.g. title preference).
  - PageRank component.
- Details of current commercial ranking functions are trade secrets.
- According to the web, reranking is done monthly

Personalized PageRank

- PageRank can be biased (personalized) by changing **E** to a non-uniform distribution.
- Restrict "random jumps" to a set of specified relevant pages.
- For example, let $E(p) = 0$ except for one's own home page, for which $E(p) = \Re$
- This results in a bias towards pages that are closer in the web graph to your own homepage.

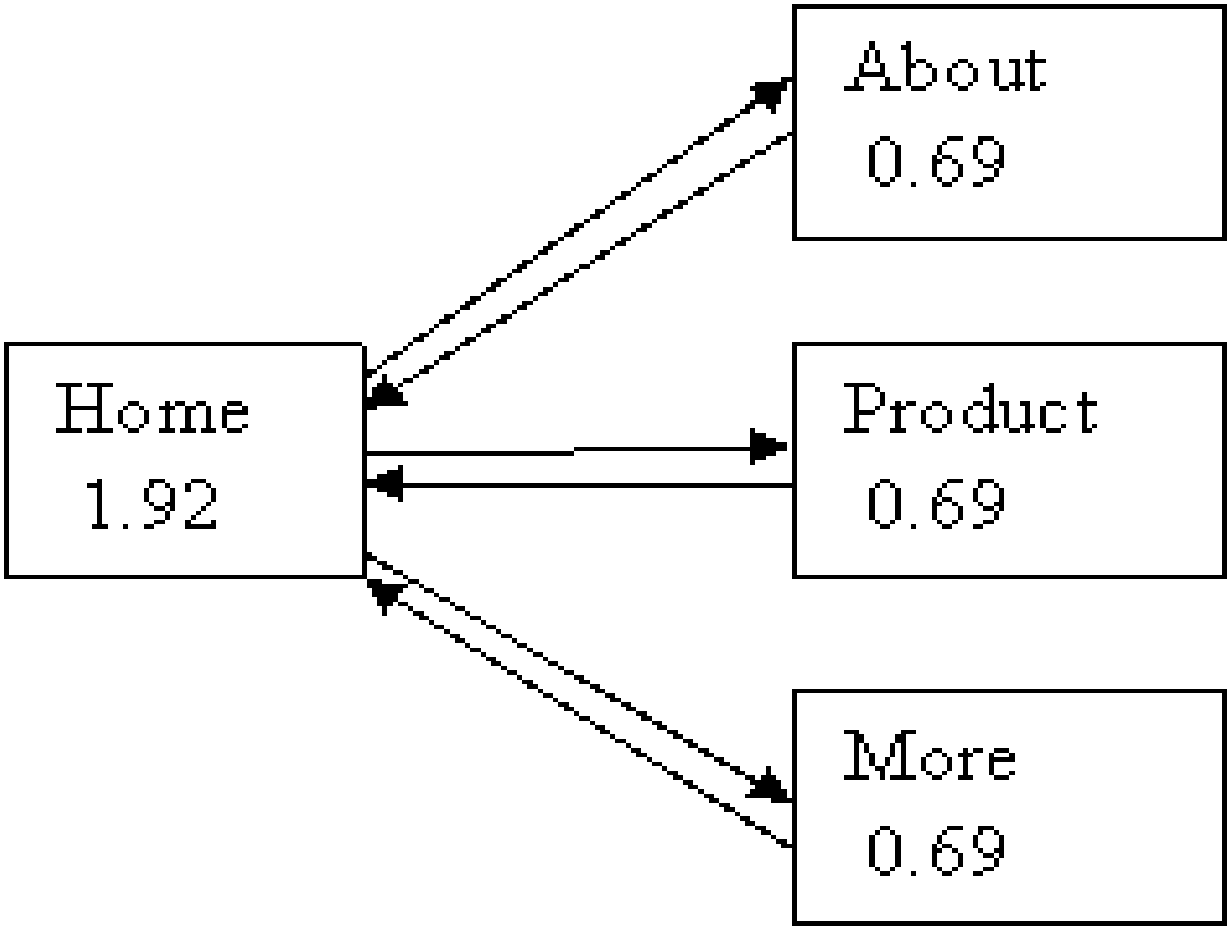
Google PageRank-Biased Spidering

- Use PageRank to direct (focus) a spider on "important" pages.
- Compute page-rank using the current set of crawled pages.
- Order the spider's search queue based on current estimated PageRank.

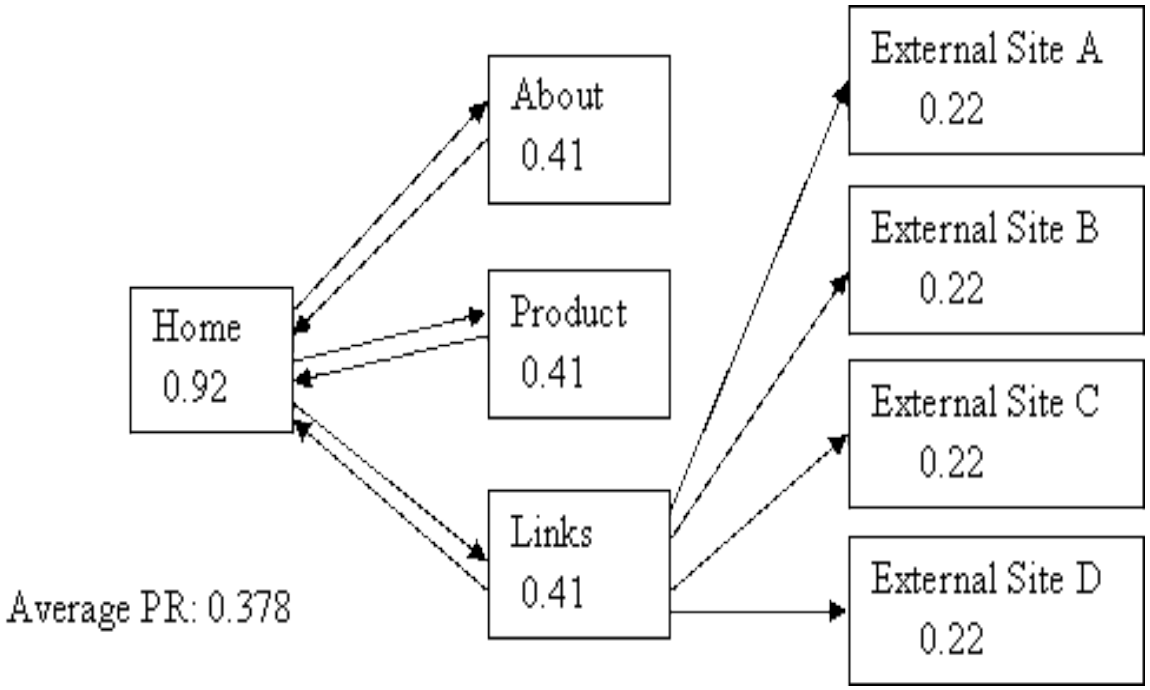
Link Analysis Conclusions

- Link analysis uses information about the structure of the web graph to aid search.
- It is one of the major innovations in web search.
- It is the primary reason for Google's success.
- BUT ..
  - link analysis is subject to abuse and rankings can be manipulated
  - it can lead to counterintuitive results

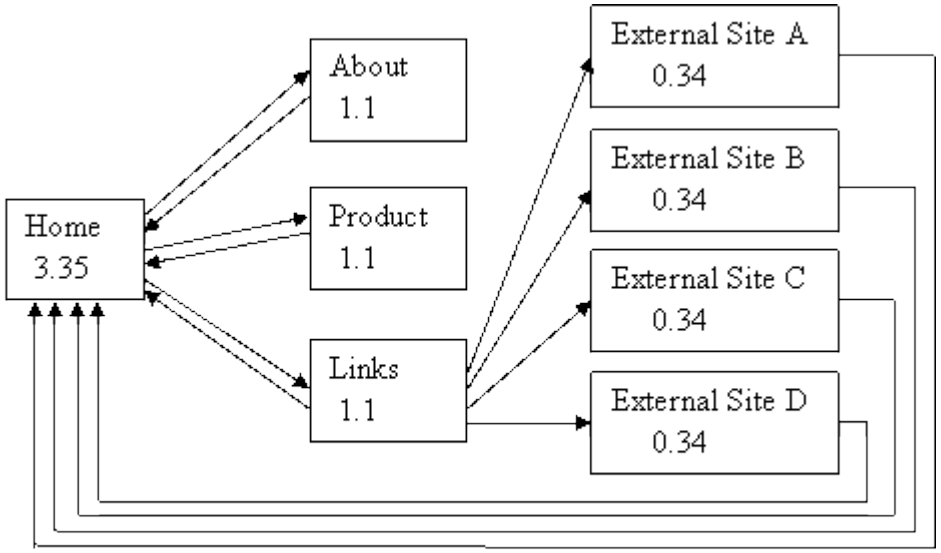
Simple hierarchical site



Hierarchy with outbound links

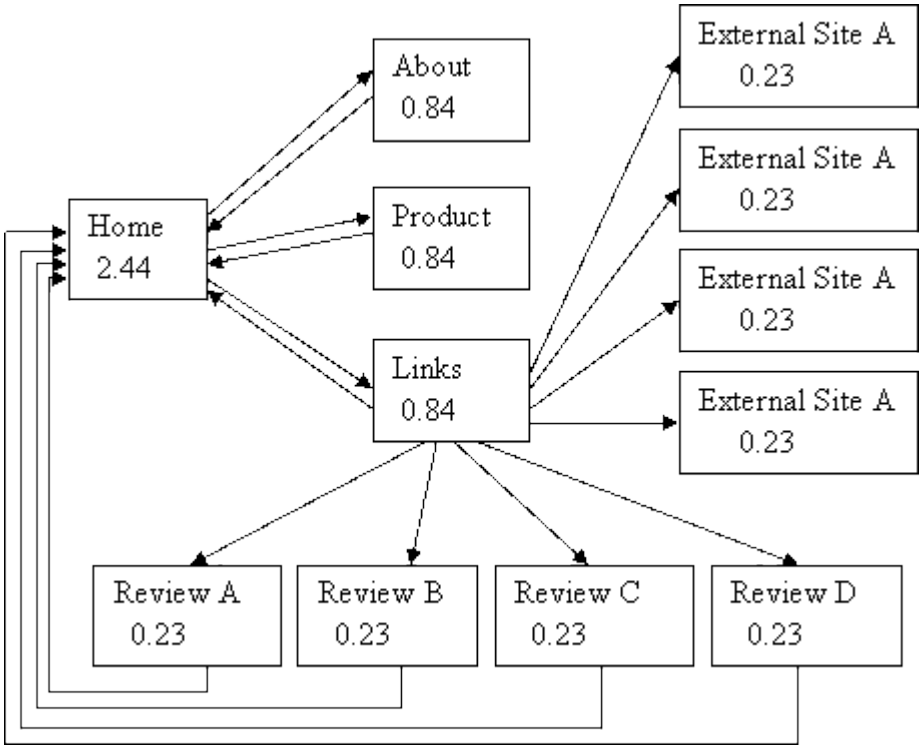


Hierarchy with outbound and links back

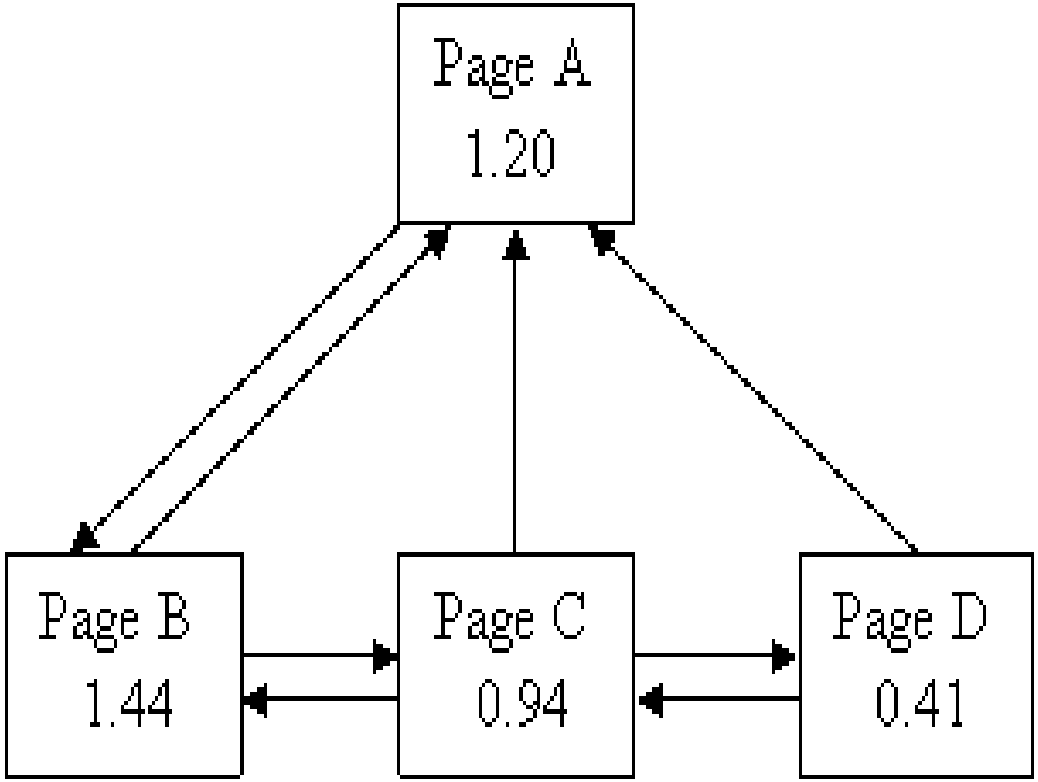


Average PR: 1.000

Hierarchy with links back within site



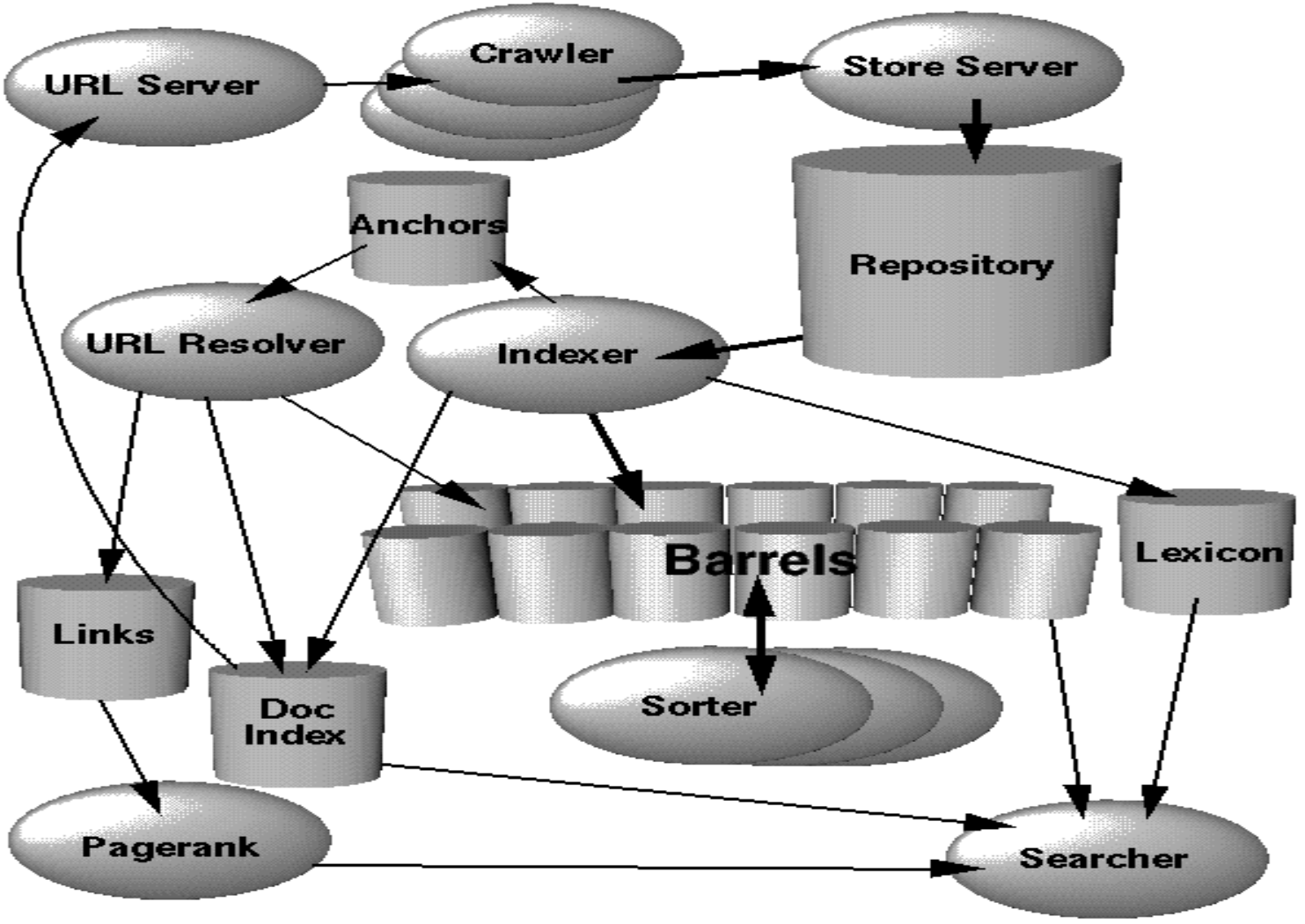
Header to a long page with foward & back pointers



Google's actual ranking method?

See http://google.com/technology/pigeonrank.html

Google


URL Server
Crawler
Store Server
Anchors
Repository
URL Resolver
Indexer
Barrels
Lexicon
Links
Doc Index
Sorter
Pagerank
Searcher

Google Crawling (as of 1998)

- URL server
  - holds lists of URLs to be crawled
- Crawlers
  - 300 open connections
  - 4 crawlers --> 100 web pages per second --> 600K per second
    - each has its own in memory DNS cache
- Store Server
  - compress, assign docID, put into repository
  - use gzip rather than bzip for decompression speed (also can send gzip out to many browsers so do not need to decompress at retrieval time

Indexing (1998)

- Repository
  - contains compressed docs 147G uncompressed, 53G compressed
- Indexer
  - parse doc into words
    - record for each word its location, font, capitalization
    - record word info in "barrels"
- Barrels
  - forward index (doc to words and their locations)
    - 43G
  - inverted index word to doc
    - 41G
- Lexicon – 14 million words – in memory

Retrieval and Ranking

- Retrieve using standard boolean inverted index stuff
- Combine
  - pagerank of document
  - counts of
    - proximity of multiple word query in document
    - number of words of query in document
  - words in anchor texts pointing to document

Architecture -- Now

- Massively parallel, Google File System
  - multiple server clusters
  - commodity servers ($1000 each)
    - they have so many that several fail every day
    - all info is distributed so that each cluster is tolerant of multiple server failures
- Cache refreshed every 1-7 days
  - this should mean no dead links
    - but I get a lot of them
- Map/Reduce Framework
  - for parallel programming structuring
- Global Work Queue
  - a pretty standard sort of batch queue

Google and Machine Learning

- Clustering
  - done ahead of time to allow queries to match documents that contain no words in the query.
- Ad Delivery
  - same idea as query – to deliver targeted ads when query on a word basis is all wrong.