# Information Retrieval and Web Search

**Salvatore Orlando**

***Bing Liu. "Web Data Mining: Exploring Hyperlinks, Contents", and Usage Data. Springer-Verlag, 2006***

***Christopher D. Manning, Prabhakar Raghavan and Hinrich Schütze, Introduction to Information Retrieval, Cambridge University Press. 2008 (http://nlp.stanford.edu/IR-book/information-retrieval-book.html)***

# Introduction

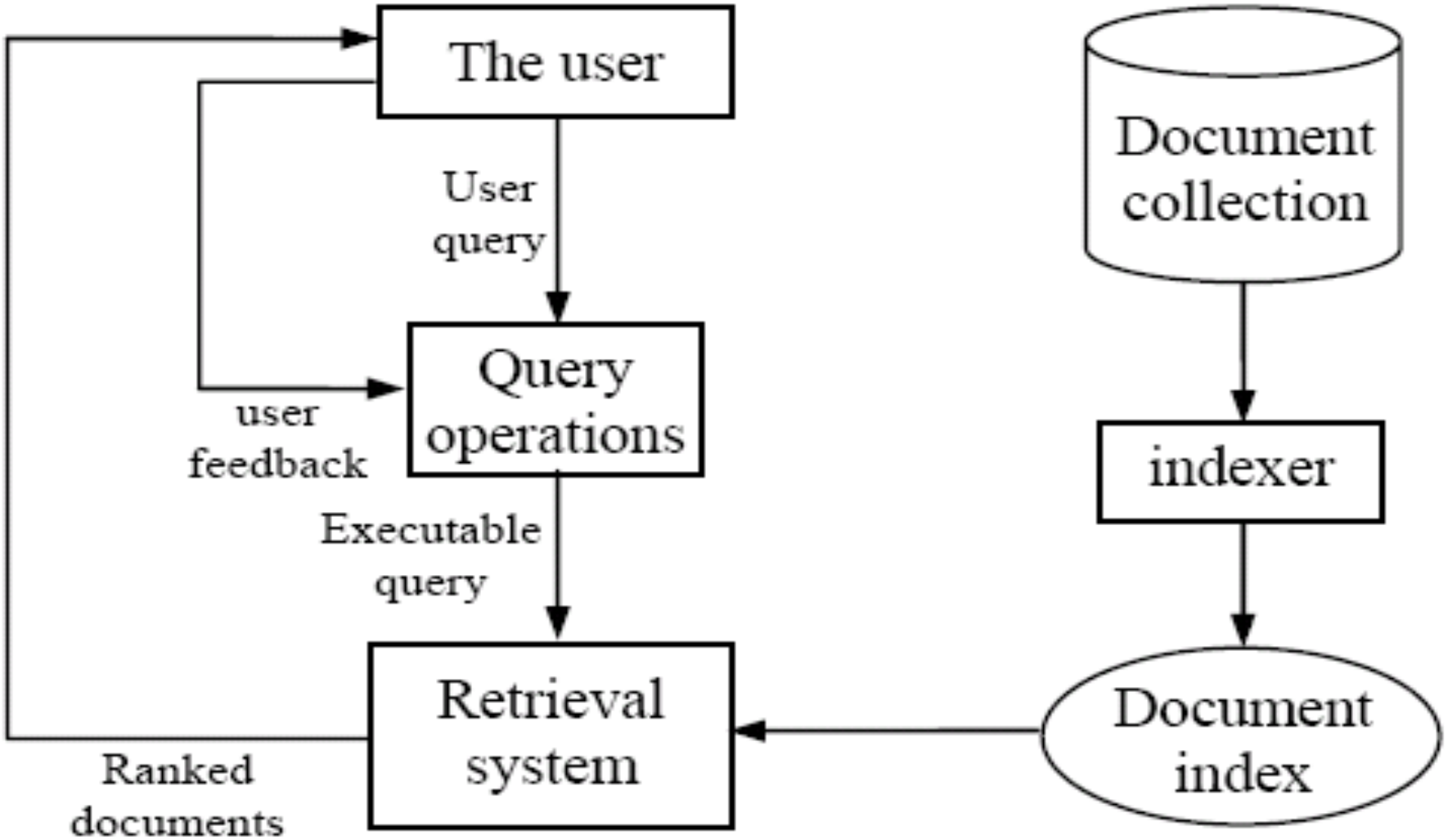
- Text mining refers to data mining using text documents as data.
- Most text mining tasks use Information Retrieval (IR) methods to pre-process text documents.
- These methods are quite different from traditional data pre-processing methods used for relational tables.
- Web search also has its root in IR.

# Information Retrieval (IR)

- **IR helps users find information that matches their information needs expressed as queries**
- **Historically, IR is about document retrieval, emphasizing *document* as the *basic unit*.**
  - **Finding documents relevant to user queries**

- **Technically, IR studies the acquisition, organization, storage, retrieval, and distribution of information.**

# IR architecture



# IR queries

- Keyword queries
- Boolean queries (using AND, OR, NOT)
- Phrase queries
- Proximity queries
- Full document queries
- Natural language questions

# Information retrieval models

- An IR model governs how a document and a query are represented and how the relevance of a document to a user query is defined

- Main models:
  - Boolean model
  - Vector space model
  - Statistical language model
  - etc

# Boolean model

- **Each document or query is treated as a "bag" of words or terms**
  - **Word sequences are not considered**

- **Given a collection of documents *D*, let**
  $V = \{t_1, t_2, ..., t_{|V|}\}$
  **be the set of distinctive words/terms in the collection.** $V$ **is called the vocabulary**

- **A weight** $w_{ij} > 0$ **is associated with each term** $t_i$ **of a document** $d_j \in D$**.**
- **For a term that does not appear in document** $d_j$**,** $w_{ij} = 0$
  $d_j = (w_{1j}, w_{2j}, ..., w_{|V|j})$

# Boolean model (contd)

- **Query terms are combined logically using the Boolean operators AND, OR, and NOT.**
  - **E.g., ((*data* AND *mining*) AND (NOT *text*))**
- **Weights $w_{ij} = 0/1$ (absence/presence) are associated with each term $t_i$ of a document $d_j \in D$**

- **Retrieval**
  - **Given a Boolean query, the system retrieves every document that makes the query logically true**
  - **Exact match**

- **The retrieval results are usually quite poor because term frequency is not considered.**

# Vector space model

- Documents are still treated as a “bag” of words or terms.
- Each document is still represented as a vector.
- However, the term weights are no longer 0 or 1.
- Each term weight is computed on the basis of some variations of TF or TF-IDF scheme.

- Term Frequency (TF) Scheme: The weight of a term $t_i$ in document $\mathrm{d}_j$ is the number of times that $t_i$ appears in $d_j$, denoted by $f_{ij}$. Normalization may also be applied.

# TF-IDF term weighting scheme

- **The most well known weighting scheme**
  - **TF: still term frequency**
  - **IDF: inverse document frequency.**

    $N$: **total number of docs**

    $df_i$: **the number of docs where** $t_i$ **appears**

$$tf_{ij} = \frac{f_{ij}}{\max\{f_{1j}, f_{2j}, \ldots, f_{|V|j}\}}$$

$$idf_i = \log \frac{N}{df_i}$$

- **The final TF-IDF term weight is:**

$$w_{ij} = tf_{ij} \times idf_i$$

# Retrieval in vector space model

- Query $q$ is represented in the same way or slightly differently.
- **Relevance of $d_i$ to $q$**: Compare the similarity of query $q$ and document $d_i$, i.e. the similarity between the two associated vectors.

- **Cosine similarity** (the cosine of the angle between the two vectors)

$$cosine(\mathbf{d}_j, \mathbf{q}) = \frac{\langle \mathbf{d}_j \bullet \mathbf{q} \rangle}{\| \mathbf{d}_j \| \times \| \mathbf{q} \|} = \frac{\sum_{i=1}^{|V|} w_{ij} \times w_{iq}}{\sqrt{\sum_{i=1}^{|V|} w_{ij}^2} \times \sqrt{\sum_{i=1}^{|V|} w_{iq}^2}}$$

- Cosine is also commonly used in text clustering

# An Example

- A document space is defined by three terms:
  - hardware, software, users
  - the vocabulary / lexicon
- A set of documents are defined as:
  - A1=(1, 0, 0), A2=(0, 1, 0), A3=(0, 0, 1)
  - A4=(1, 1, 0), A5=(1, 0, 1), A6=(0, 1, 1)
  - A7=(1, 1, 1) A8=(1, 0, 1). A9=(0, 1, 1)
- If the Query is “hardware, software”
  - i.e., (1, 1, 0)
- what documents should be retrieved?

# An Example (cont.)

- **In Boolean query matching:**
  - **AND: documents A4, A7**
  - **OR: documents A1, A2, A4, A5, A6, A7, A8, A9**
- **In similarity matching (cosine):**
  - **q=(1, 1, 0)**
  - **S(q, A1)=0.71, S(q, A2)=0.71, S(q, A3)=0**
  - **S(q, A4)=1, S(q, A5)=0.5, S(q, A6)=0.5**
  - **S(q, A7)=0.82, S(q, A8)=0.5, S(q, A9)=0.5**
  - **Document retrieved set (with ranking, where cosine>0):**
    - **{A4, A7, A1, A2, A5, A6, A8, A9}**

# Relevance feedback

- **Relevance feedback is one of the techniques for improving retrieval effectiveness. The steps:**
  - **the user first identifies some relevant ($D_r$) and irrelevant documents ($D_{ir}$) in the initial list of retrieved documents**
  - **goal: "expand" the query vector in order to maximize similarity with relevant documents, while minimizing similarity with irrelevant documents**
    - **query *q* expanded by extracting additional terms from the sample relevant ($D_r$) and irrelevant ($D_{ir}$) documents to produce $q_e$**

$$\mathbf{q}_e = \alpha\mathbf{q} + \frac{\beta}{|D_r|}\sum_{\mathbf{d}_r \in D_r}\mathbf{d}_r - \frac{\gamma}{|D_{ir}|}\sum_{\mathbf{d}_{ir} \in D_{ir}}\mathbf{d}_{ir}$$

  - **Perform a second round of retrieval.**
- **Rocchio method (*α*, *β* and *γ* are parameters)**

# Rocchio text classifier

- **Training set: relevant and irrelevant docs**
  - **you can train a classifier**

- **The Rocchio classification method, can be used to improve retrieval effectiveness too**
- **Rocchio classifier is constructed by producing a prototype vector $c_i$ for each class $i$ (*relevant* or *irrelevant* in this case) associated with document set $D_i$:**

$$\mathbf{c}_i = \frac{\alpha}{|D_i|}\sum_{\mathbf{d}\in D_i}\frac{\mathbf{d}}{\|\mathbf{d}\|} - \frac{\beta}{|D - D_i|}\sum_{\mathbf{d}\in D - D_i}\frac{\mathbf{d}}{\|\mathbf{d}\|}$$

- **In classification, cosine is used.**

# Text pre-processing

- **Word (term) extraction: easy**
- **Stopwords removal**
- **Stemming**
- **Frequency counts and computing TF-IDF term weights.**

# Stopwords removal

- Many of the most frequently used words in English are useless in IR and text mining – these words are called *stop words*
  - "the", "of", "and", "to", ….
  - Typically about 400 to 500 such words
  - For an application, an additional domain specific stopwords list may be constructed
- Why do we need to remove stopwords?
  - Reduce indexing (or data) file size
    - stopwords accounts 20-30% of total word counts.
  - Improve efficiency and effectiveness
    - stopwords are not useful for searching or text mining
    - they may also confuse the retrieval system
- Current Web Search Engines generally do not use stopword lists to perform "phrase search"

# Stemming

- **Techniques used to find out the root/stem of a word. e.g.,**

| | |
|---|---|
| user | engineering |
| users | engineered |
| used | engineer |
| using | |
| use | engineer |

← **stem**

**Usefulness:**

- **improving effectiveness of IR and text mining**
  - **Matching similar words**
  - **Mainly improve recall**
- **reducing indexing size**
  - **combing words with the same roots may reduce indexing size as much as 40-50%**
  - **Web Search Engine may need to index un-stemmed words too for "phrase search"**

# Basic stemming methods

**Using a set of rules. e.g., English rules**

- **remove ending**
  - **if a word ends with a consonant other than s, followed by an s, then delete s.**
  - **if a word ends in es, drop the s.**
  - **if a word ends in ing, delete the ing unless the remaining word consists only of one letter or of th.**
  - **If a word ends with ed, preceded by a consonant, delete the ed unless this leaves only a single letter.**
  - **……**
- **transform words**
  - **if a word ends with "ies", but not "eies" or "aies", then "ies ➔ y"**

# Evaluation: Precision and Recall

- **Given a query:**
  - **Are all retrieved documents relevant?**
  - **Have all the relevant documents been retrieved?**
- **Measures for system performance:**
  - **The first question is about the precision of the search**

$$\text{Precision} = \frac{\#(\text{relevant items retrieved})}{\#(\text{retrieved items})} = P(\text{relevant}|\text{retrieved})$$

  - **The second is about the completeness (recall) of the search.**

$$\text{Recall} = \frac{\#(\text{relevant items retrieved})}{\#(\text{relevant items})} = P(\text{retrieved}|\text{relevant})$$

# Precision-recall curve

**Example 2:** Following Example 1, we obtain the interpolated precisions at all 11 recall levels in the table of Fig. 6.4. The precision-recall curve is shown on the right.

| $i$ | $p(r_i)$ | $r_i$ |
|---|---|---|
| 0 | 100% | 0% |
| 1 | 100% | 10% |
| 2 | 100% | 20% |
| 3 | 100% | 30% |
| 4 | 80% | 40% |
| 5 | 80% | 50% |
| 6 | 71% | 60% |
| 7 | 70% | 70% |
| 8 | 70% | 80% |
| 9 | 62% | 90% |
| 10 | 62% | 100% |

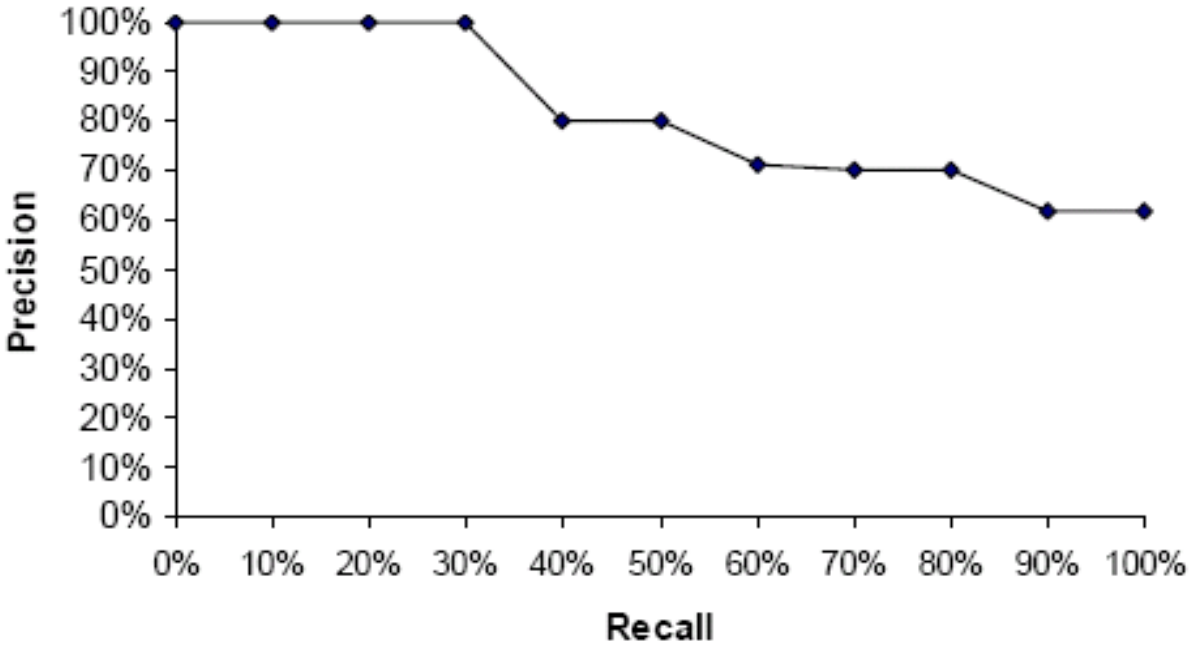



**Fig. 6.4.** The precision-recall curve

# Compare different retrieval algorithms

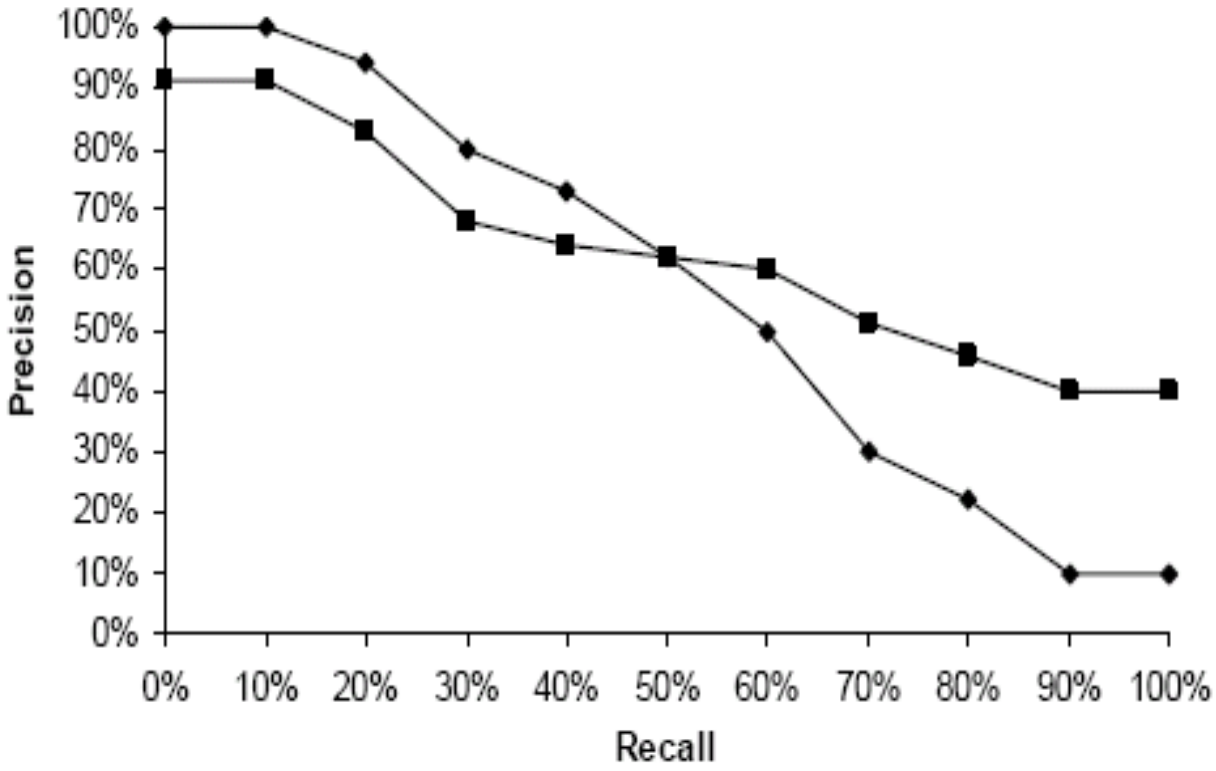



**Fig. 6.5.** Comparison of two retrieval algorithms based on their precision-recall curves

# Compare with multiple queries

- **Compute the average precision at each recall level**

$$\overline{p}(r_i) = \frac{1}{|Q|} \sum_{j=1}^{|Q|} p_j(r_i), \qquad (22)$$

where $Q$ is the set of all queries and $p_j(r_i)$ is the precision of query $j$ at the recall level $r_i$. Using the average precision at each recall level, we can also draw a precision-recall curve.

- **Draw precision recall curves**
- **Do not forget the F-score evaluation measure.**

# Rank precision

- **Compute the precision values at some selected rank positions.**
  - **Mainly used in Web search evaluation**

- **For a Web search engine, we can compute precisions for the top 5, 10, 15, 20, 25 and 30 returned pages**
  - **as the user seldom looks at more than 30 pages**
  - **P@5, P@10, P@15, P@20, P@25, P@30**

- ***Recall* is not very meaningful in Web search.**
  - **Why?**

# Inverted index

- The inverted index of a document collection is basically a data structure that
  - attaches each distinctive term with a list of all documents that contain the term.
- Thus, in retrieval, it takes constant time to
  - find the documents that contains a query term.
  - multiple query terms are also easy handled as we will see soon.

# An example

**Example 3:** We have three documents of $id_1$, $id_2$, and $id_3$:

$id_1$: Web mining is useful.
1 2 3 4

$id_2$: Usage mining applications.
1 2 3

$id_3$: Web structure mining studies the Web hyperlink structure.
1 2 3 4 5 6 7 8

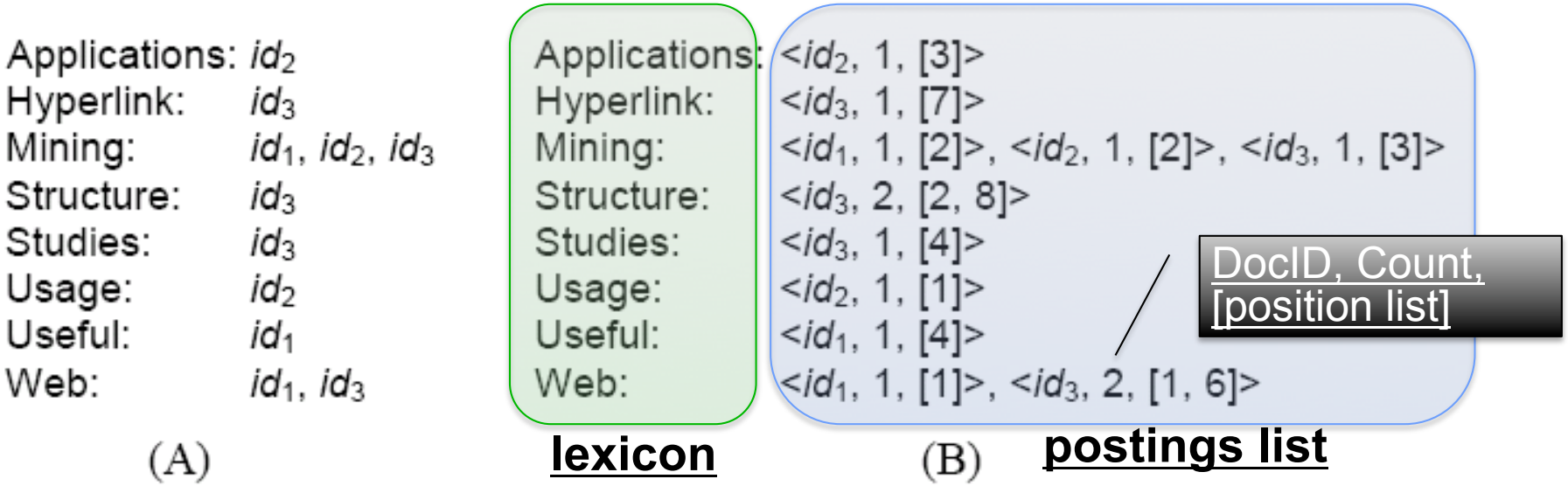



**Fig. 6.7.** Two inverted indices: a simple version and a more complex version

# Index construction

- **Easy! See the example,**

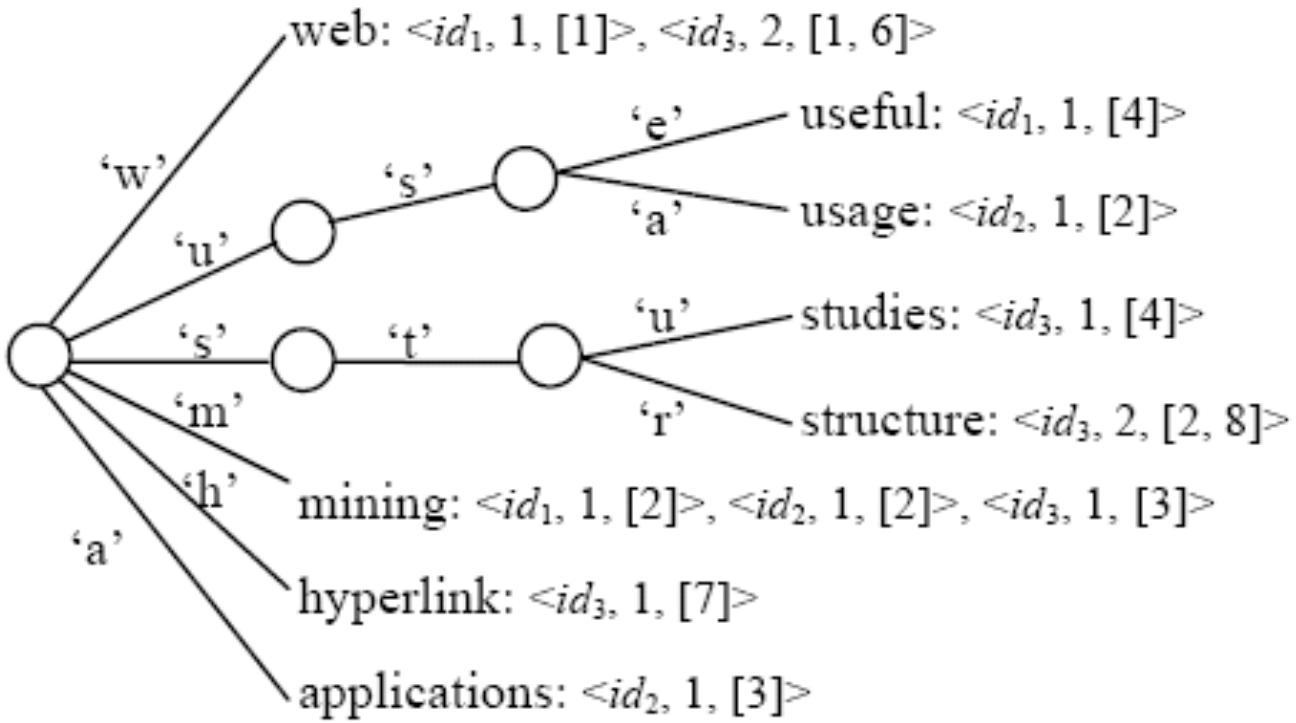
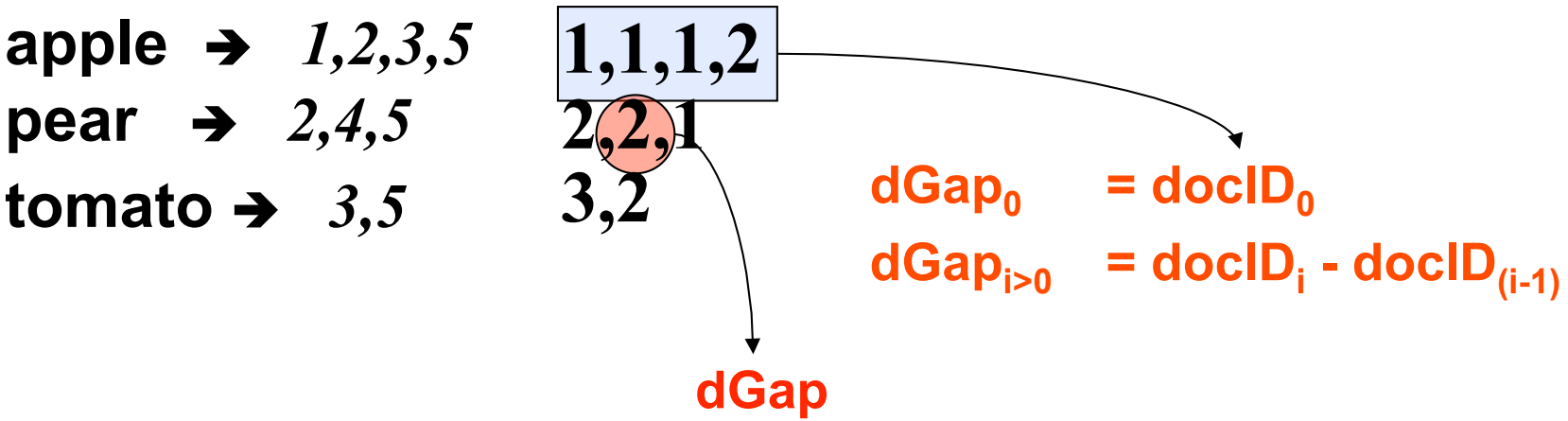



**Fig. 6.8.** The vocabulary trie and the inverted lists

# Index compression

- Postings lists are ordered with respect to docID
  - Compression – instead of docIDs we can compress smaller *gaps between doc*IDs, thus reducing space requirements for the index
- Use a variable number of bit/byte for gap representation
  - the gaps have a smaller magnitude than docIDs



# Index compression

- **Example of compression using Variable Byte econding**
  - **$824_{10}$= 110 $0111000_2$**
  - **$5_{10}$= $101_2$**
  - **$214577_{10}$= 1101 0001100 $0110001_2$**

| | | | |
|---|---|---|---|
| docIDs | 824 | 829 | 215406 |
| gaps | | 5 | 214577 |
| VB code | 00000110 10111000 | 10000101 | 00001101 00001100 10110001 |

► **Table 5.4** Variable byte (VB) encoding. Gaps are encoded using an integral number of bytes. The first bit, the continuation bit, of each byte indicates whether the code ends with this byte (1) or not (0).

# Search using inverted index

**Given a query *q*, search has the following steps:**

- **Step 1 (Vocabulary search): find each term/word in *q* in the inverted index.**
- **Step 2 (Results merging): Merge results to find documents that contain all or some of the words/terms in *q***
  - ***AND/OR* of postings lists**
- **Step 3 (Rank score computation): To rank the resulting documents/pages, by using**
  - **content-based ranking (e.g. TF-IDF)**
  - **link-based ranking ⇐ Web Search Engine**

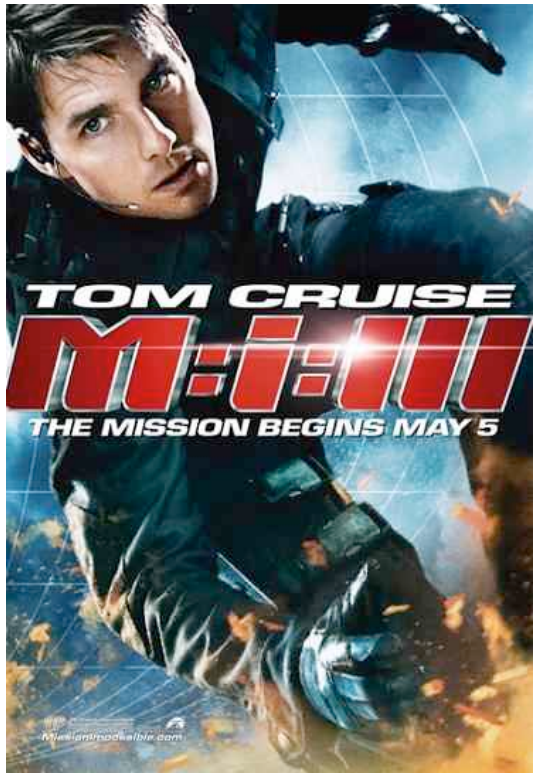
# Web Search Engine (WSE) as a huge IR system

- A Web crawler (spider, robot) crawls the Web to collect all the pages.
- WSE servers establish a huge inverted indexing database and other indexing databases
- At query (search) time, WSEs conduct different types of vector query matching.

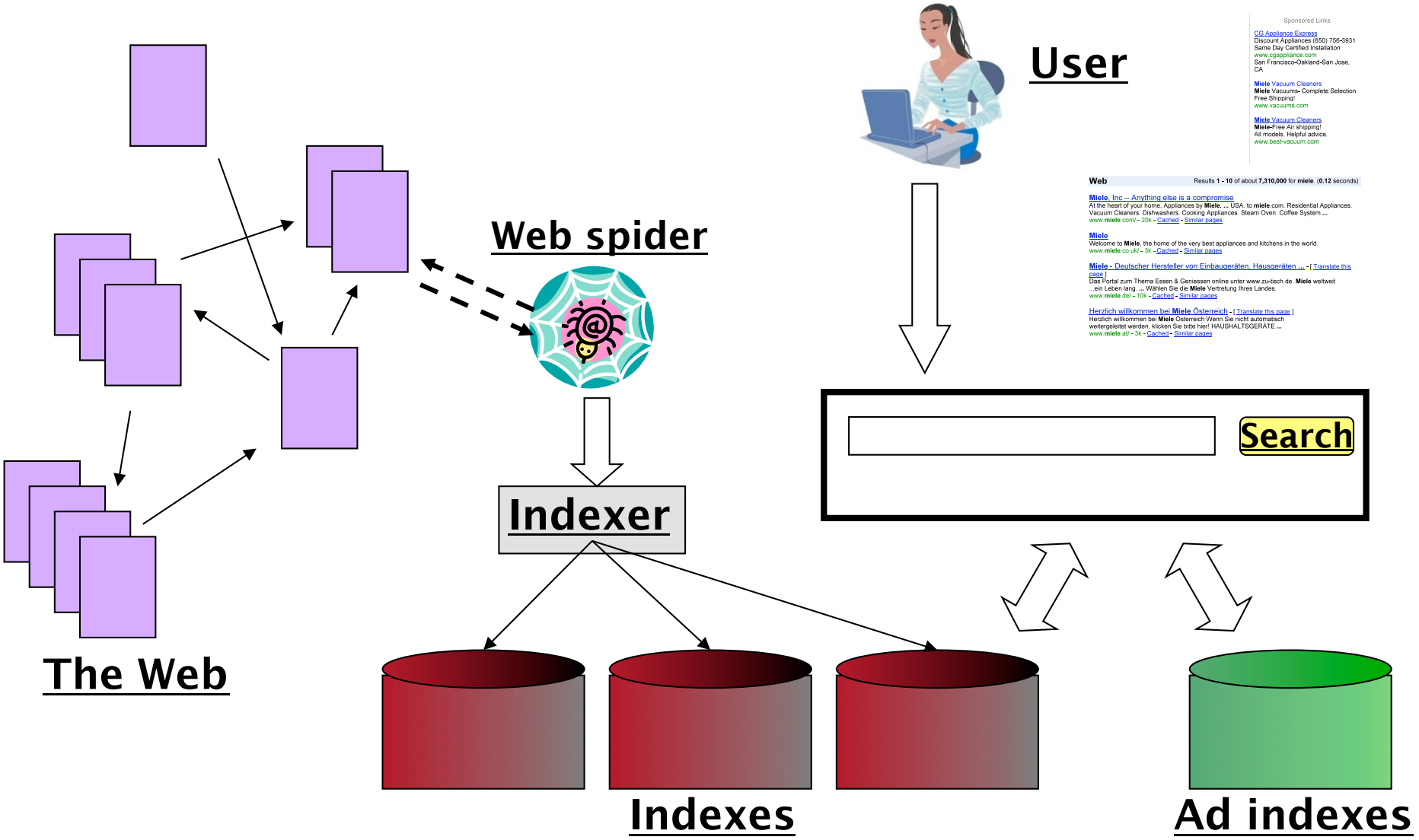
# Mission impossible ?

- WSE
  - Crawl and index billions of pages
  - Answer hundreds of millions of queries per day
  - In less than 1 sec. per query
- Users
  - Want to submit short queries (on avg. 2.5 terms), often with orthographic errors
  - Expect to receive the most relevant results of the Web
  - In a blink of eye

- In terms of 1990 IR, almost unimaginable



# Web Search as a huge IR system



# Different search engines

- The real differences among different search engines are
  - their index weighting schemes
    - Including context where terms appear, e.g., title, body, emphasized words, etc.
  - their query processing methods (e.g., query classification, expansion, etc)
  - their ranking algorithms
  - few of these are published by any of the search engine companies. They are tightly guarded secrets.

# Web Search Engines: what do the users search?


- **The 250 most frequent terms in the famous AOL query log!**

# Query analysis to evaluate user needs

- **Informational** – want to learn about something (~40% / 65%)

  `Low hemoglobin`

- **Navigational** – want to go to that page (~25% / 15%)

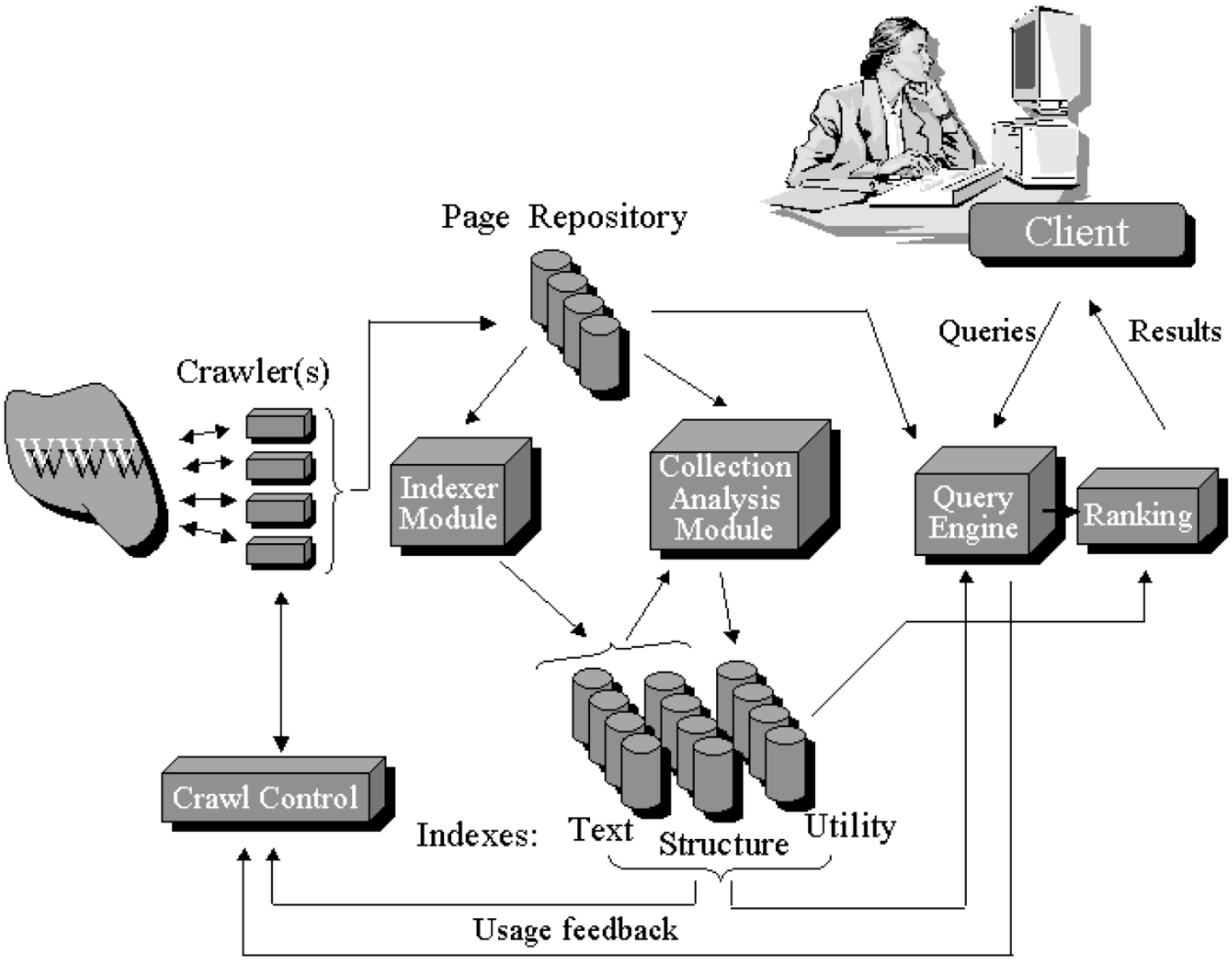
  `United Airlines`

- **Transactional** – want to do something (web-mediated) (~35% / 20%)
  - Access a service — `Seattle weather`
  - Downloads — `Mars surface images`
  - Shop — `Canon S410`
- **Gray areas**
  - Find a good hub — `Car rental Brasil`
  - Exploratory search "see what's there"

*A. Z. Broder,* ***"A taxonomy of web search",*** *SIGIR Forum, vol. 36, no. 2, pp. 3–10, 2002.*

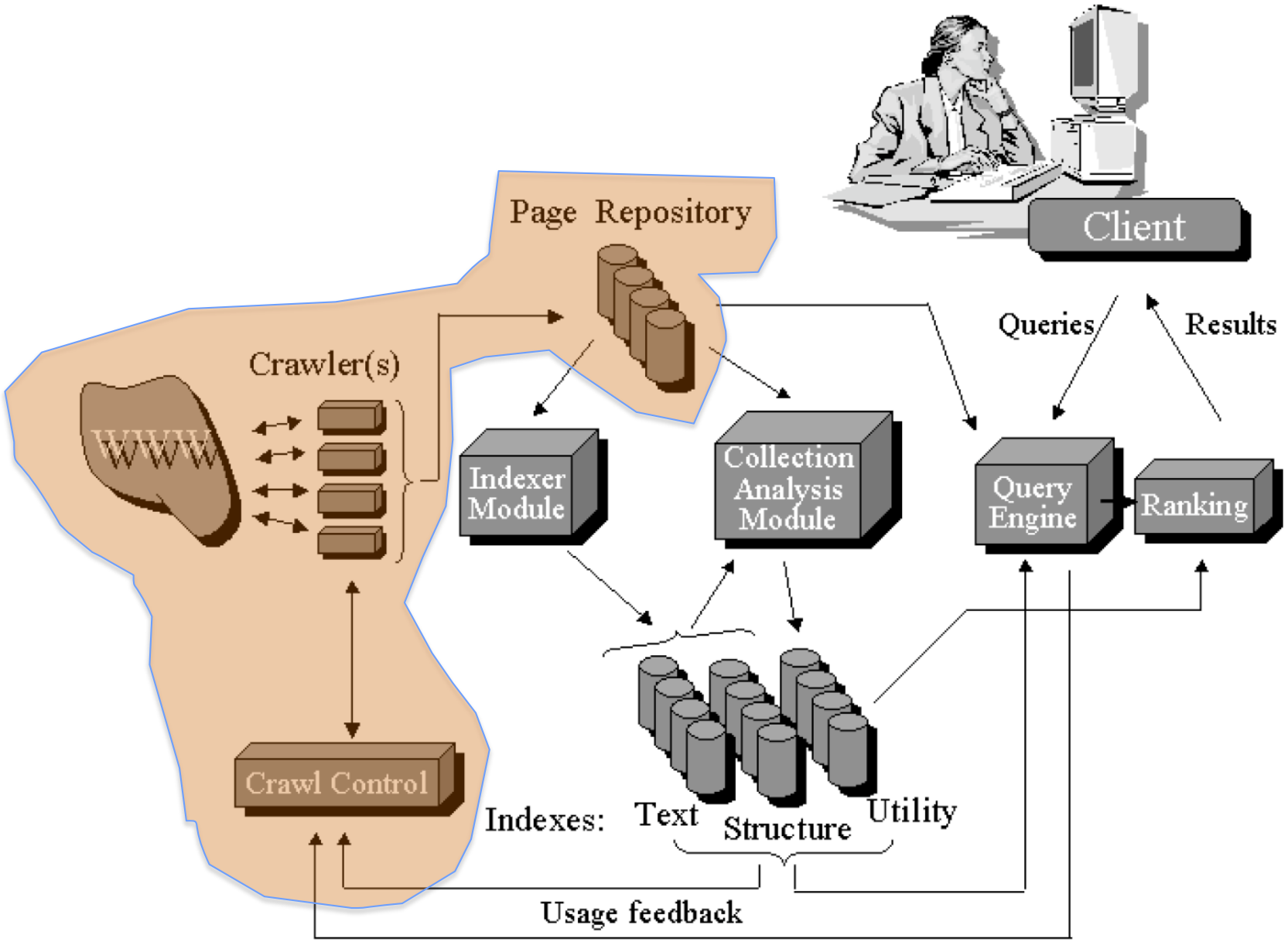
# Web Search Engines

# Anatomy of a modern Web Search Engine




A. Arasu et al., **"Searching the Web"**, ACM Transaction on Internet Technology, 1(1), 2001.

# Crawler


Client
Page Repository
Queries
Results
Crawler(s)
WWW
Indexer Module
Collection Analysis Module
Query Engine
Ranking
Crawl Control
Indexes:
Text
Structure
Utility
Usage feedback

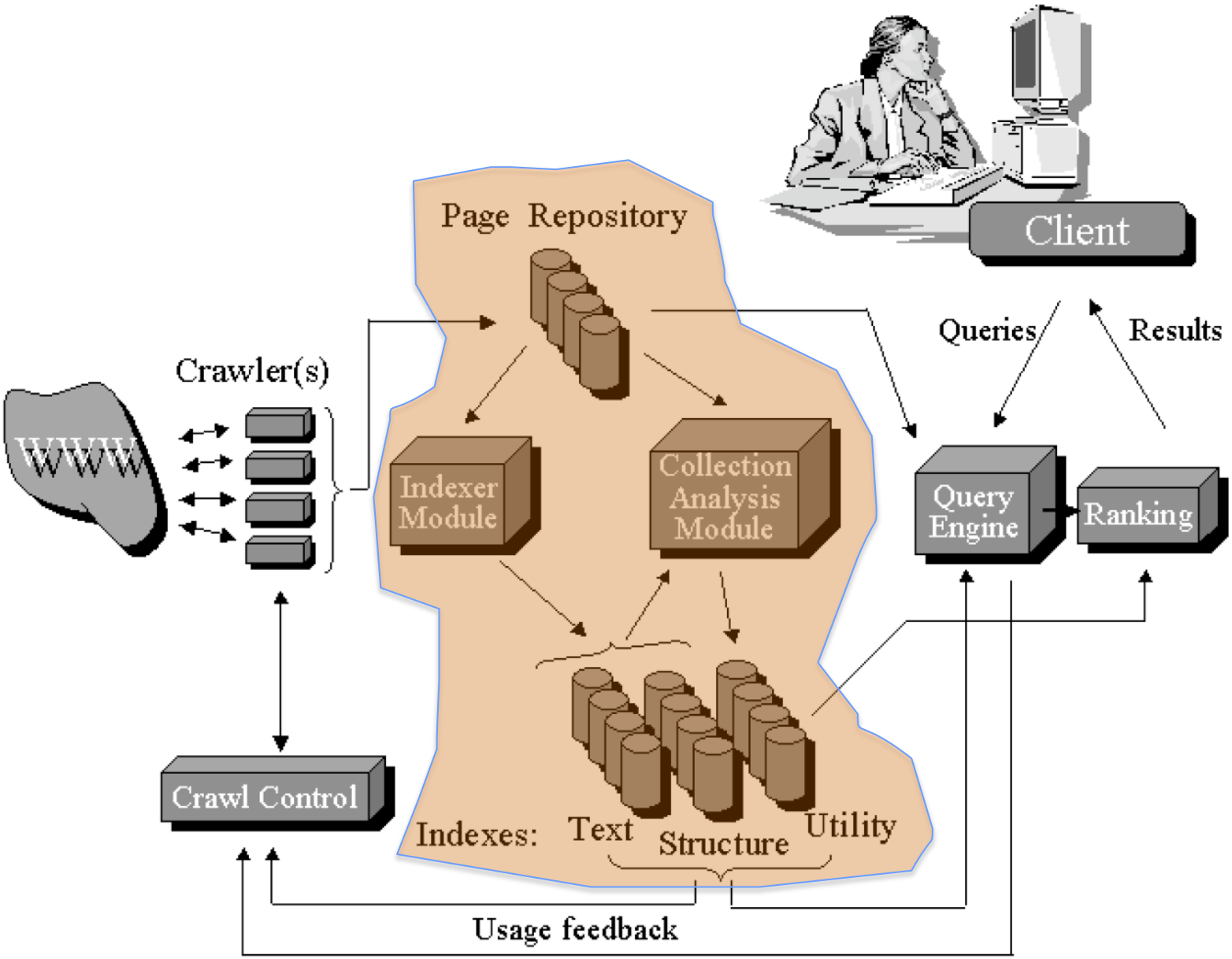
# Crawler

- It is a program that navigates the Web following the hyperlinks and stores them in a *page reporitory*

- Design Issues of the *Crawl module*:
  - What pages to download
  - When to refresh
  - Minimize load on web sites
  - How to parallelize the process

- Page selection: *Importance metric*
  - Given a page P, define how "good" that page is, on the basis of several metrics:
    - Interest driven: driven from a query, based on the similarity with page contents
    - Popularity driven: Back-link counts or PageRank
    - Location driven: Deepness of the page in a site
    - Usage driven: Click counts of the pages *(feedback)*
    - Combined

# Indexer and Page Repository



# Storage

- The page repository is a scalable storage system for web pages
- Allows the Crawler to store pages
- Allows the Indexer and Collection Analysis to retrieve them
- Similar to other data storage systems – DB or file systems
- Does not have to provide some of the other systems' features: transactions, logging, directory.

# Designing a Distributed Page Repository

- Repository designed to work over a cluster of interconnected nodes
- Page distribution across nodes
  - Uniform distribution – any page can be sent to any node
  - Hash distribution policy – hash page ID space into node ID space
- Physical organization within a node
- Update strategy
  - batch (Periodically executed)
  - steady (Run all the time)

# Indexer and collection analysis modules

- The Indexer module creates Two indexes:
  - Text (content) index : Uses “Traditional” indexing methods like Inverted Indexing.
  - Structure (links) index : Uses a directed graph of pages and links.
    Sometimes also creates an inverted graph, in order to answer queries that ask for all the pages that have hyperlinks pointing to a given page

- The collection analysis module uses the 2 basic indexes created by the indexer module in order to assemble “Utility Indexes”
  - e.g.: a site index.

# Indexer: Design Issues and Challenges

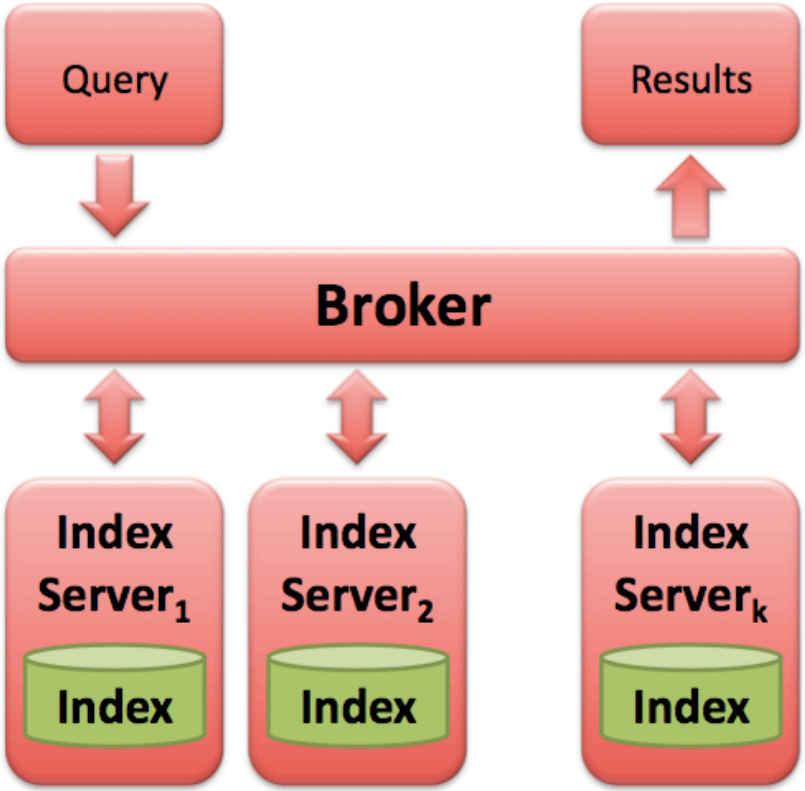
- **Index build must be :**
  - **Fast**
  - **Economic**

  **(unlike traditional index builds)**

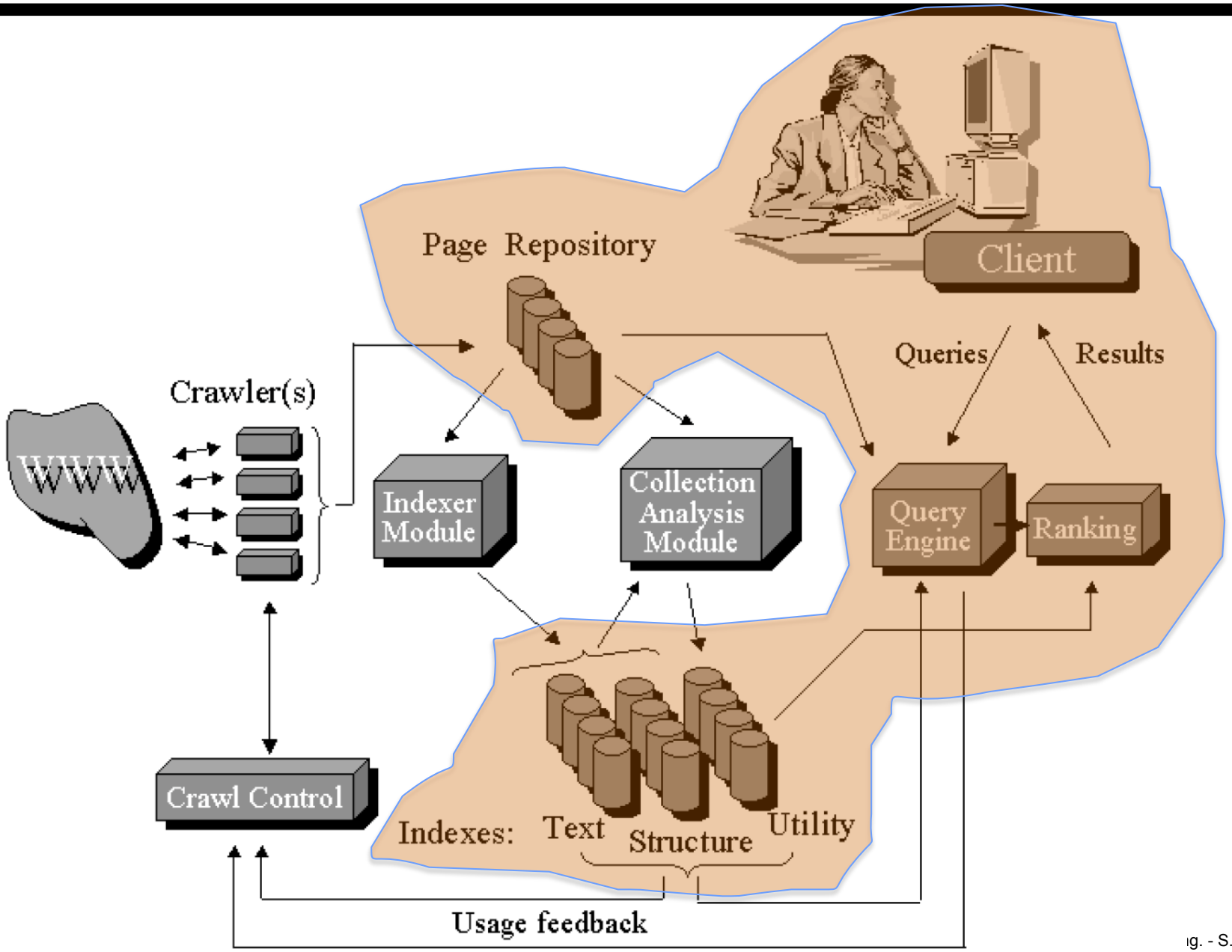
- **Incremental Indexing must be supported**
- **Personalization**
- **Storage : compression vs. speed**

# Index partitioning

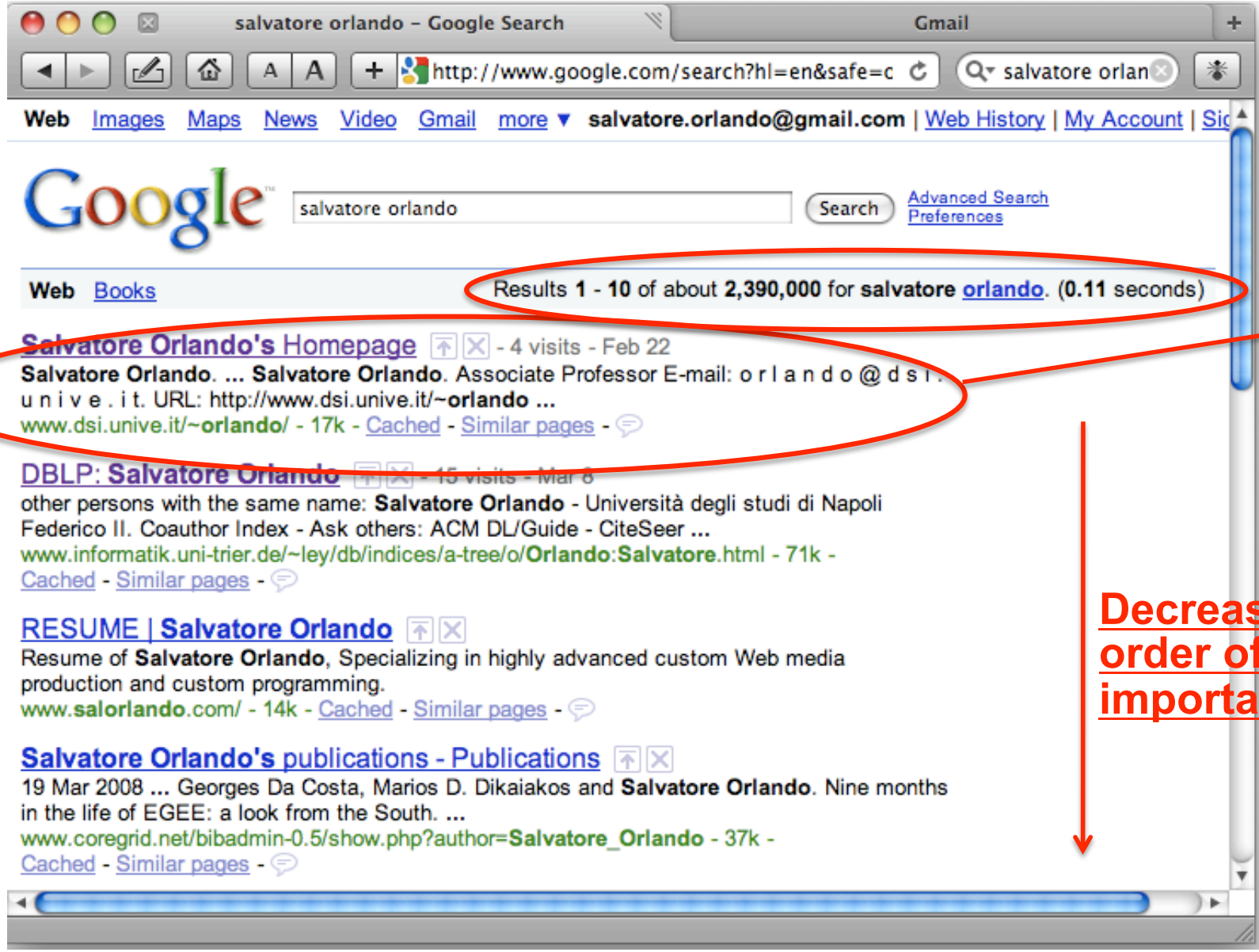
- Partitioning Inverted Files
  - Local inverted file
    - each node contains indexes of a disjoint partition of the document collection
    - query is broadcasted and answers are obtained by merging local results
  - Global inverted file
    - each node is responsible only for a subset of terms in the collection
    - query is selectively sent to the appropriate nodes only



# Query engine



# Query Engine



# Query engine

- The query engine module accepts queries from the multitude of users and return the results
  - Exploits the partitioned index to quickly find the relevant pages
  - Use Page Repository to prepare the page of the (10) results
    - snippet construction is query-based
  - Since the possible results are a huge number, the *ranking* module has to order the results according to their relevance
- Ranking
  - not only based on traditional IR content-based approaches
  - terms may be of poor quality or not relevant
  - insufficient self-description of user intent
  - spam

➔Link analysis, e.g. PageRank that exploits backlinks from "important" pages to raise the rank of pages

➔Exploit the position of the query terms in the pages

# Summary

- We only give a VERY brief introduction to IR. There are a large number of other topics, e.g.,
  - Statistical language model
  - Latent semantic indexing (LSI and SVD).
- Many other interesting topics are not covered, e.g.,
  - Web search
    - Index compression
    - Ranking: combining contents and hyperlinks (see the next block of slides)
  - Web page pre-processing
  - Combining multiple rankings and meta search
  - Web spamming
- Read the textbooks