Information Retrieval

CS6200

Search Engine Architecture

Jesse Anderton
College of Computer and Information Science
Northeastern University

Designing a Search Engine

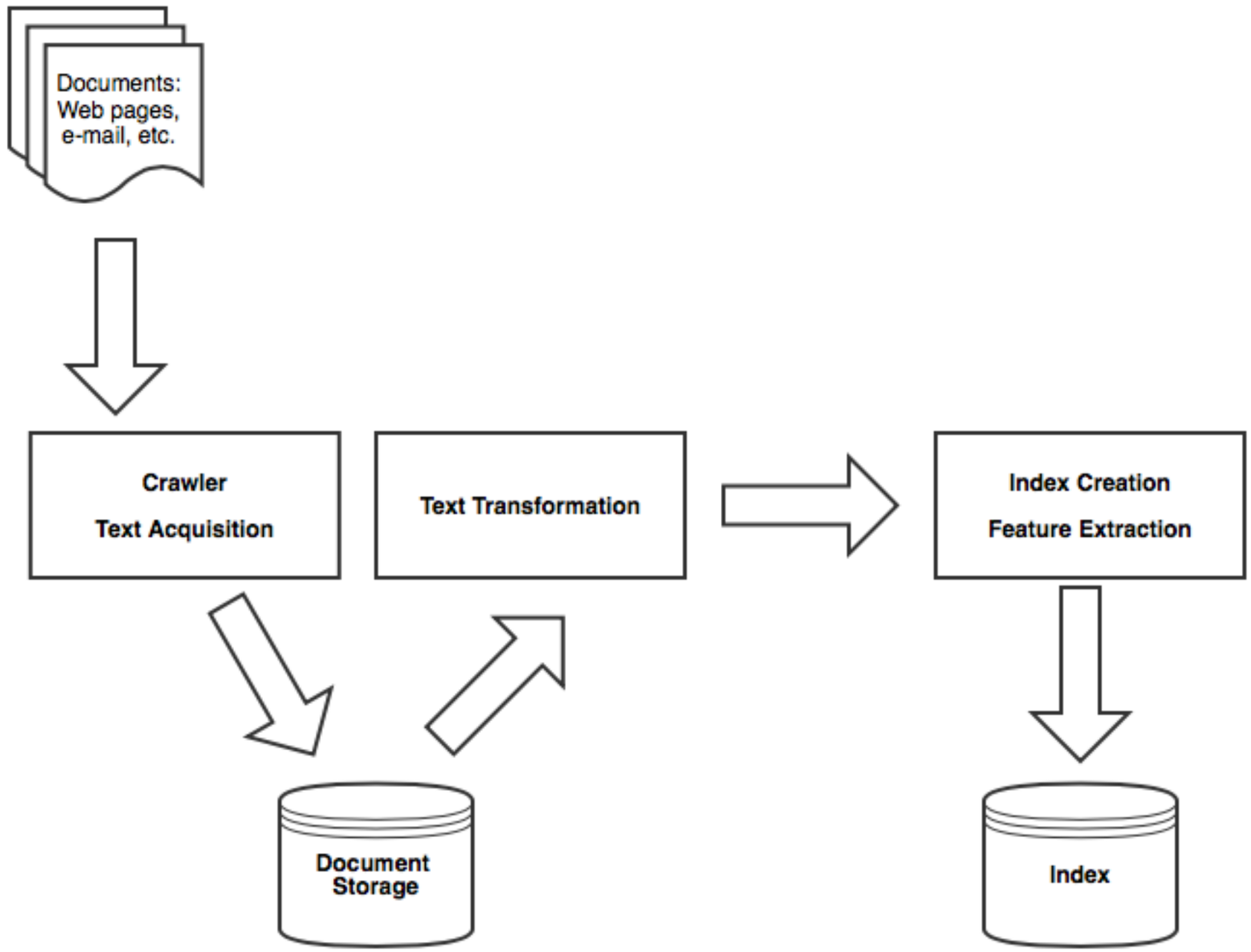
Search engine design balances two factors:

- Effectiveness – accuracy of results, presentation of results, absence of spam, good ad selection

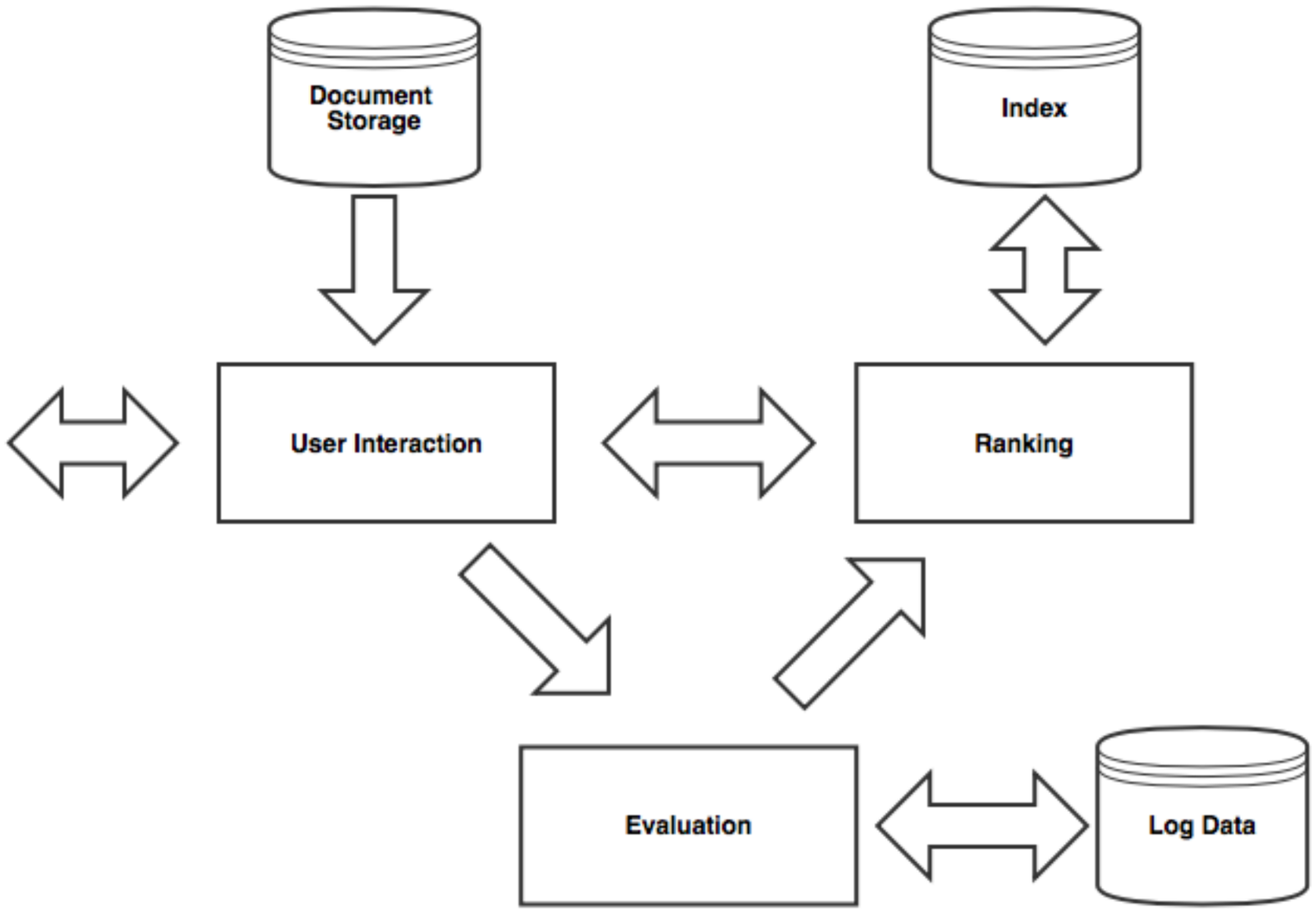
- Performance – response time, concurrency, disaster mitigation, security issues

These factors deeply impact the architecture of these systems. Often the engineering solutions feed back into research (NoSQL, Map Reduce, etc.).

Indexing Process



Query Process



Text Acquisition

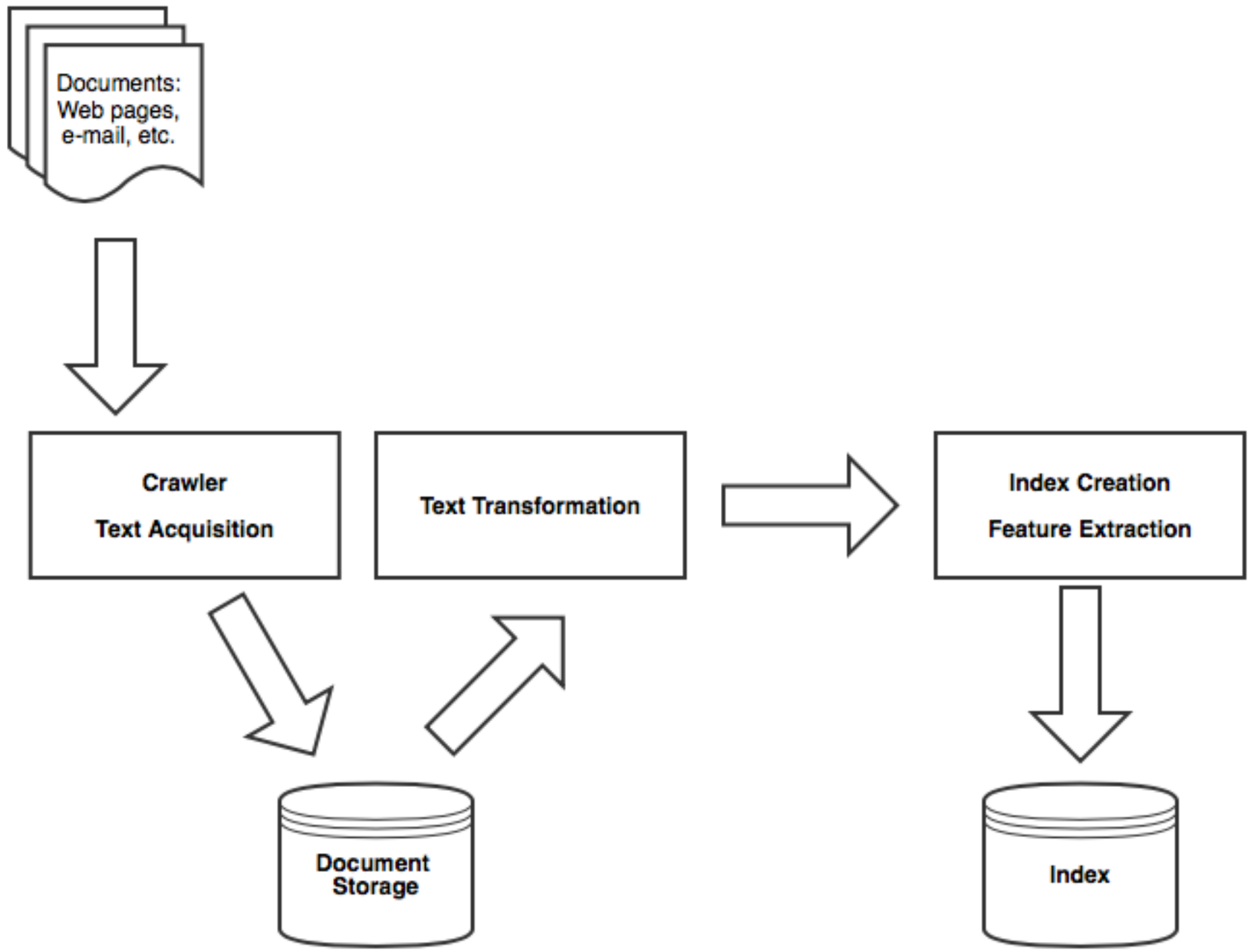
- Text acquisition is performed by a *crawler*
  - Selects and retrieves documents to be indexed, in part by following links on fetched documents
  - *Coverage*: Select a representative sample of all documents – often too many to get all of them
  - *Freshness*: Refresh updated and new documents
- There are many varieties of crawlers customized for general web search, single site indexing, corporate document repositories, e-mail repositories, server log files, personal computer filesystems, and on and on.

Document Processing

- Once a document is fetched, it needs to be processed so downstream tools (like the indexer) get consistent input
  - Document formats may be converted to a single format (e.g. Word, HTML, XML, PDF –> XML)
  - Text encodings are standardized, perhaps to UTF-8

- Normalized documents are stored along with metadata
  - Metadata: the URL, time fetched, list of links on the page, anchor text associated with those links, etc. Useful signals for relevance.
  - The document store serves as fast input to the indexer and sometimes user interface (“see cached version”)
  - Relational databases tend to be avoided in favor of faster, simpler distributed storage systems (e.g. Big Table, NoSQL, …)

Indexing Process



Text Transformation

- Parsing the text
  - We use document format-specific parsers to extract relevant information: title, links, emphasized text, etc. Markup languages such as HTML help with this process. (e.g. anything in a <h1> or <h2> is probably important)
  - *Tokenizer*: A document is converted to a stream of *tokens*, e.g. individual words.
  - Word segmentation is a difficult NLP task, especially for noisy documents found on the web. What are the words in “P.T.Barnum?” Are they the same as in “PTBarnum?” How about “ptb-ar-num?”
  - Often heuristics are used, either because they are fast and “good enough,” or because the theoretical problem is unsolved

Text Transformation

- Transforming the text
  - *Stopping*: We remove too-common words, and words that serve a syntactic rather than semantic purpose (e.g. “a,” “the”)
    - This often helps with efficiency and effectiveness.
    - This hurts some queries. How can a user find information about the UK band “The The?”
  - *Stemming*: We transform words into a canonical form to group words having a common stem: “computer,” “computers,” “computing,” –> “comput”
    - This also usually helps, but sometimes hurts.
    - This trick is not equally useful (or possible) for all languages.

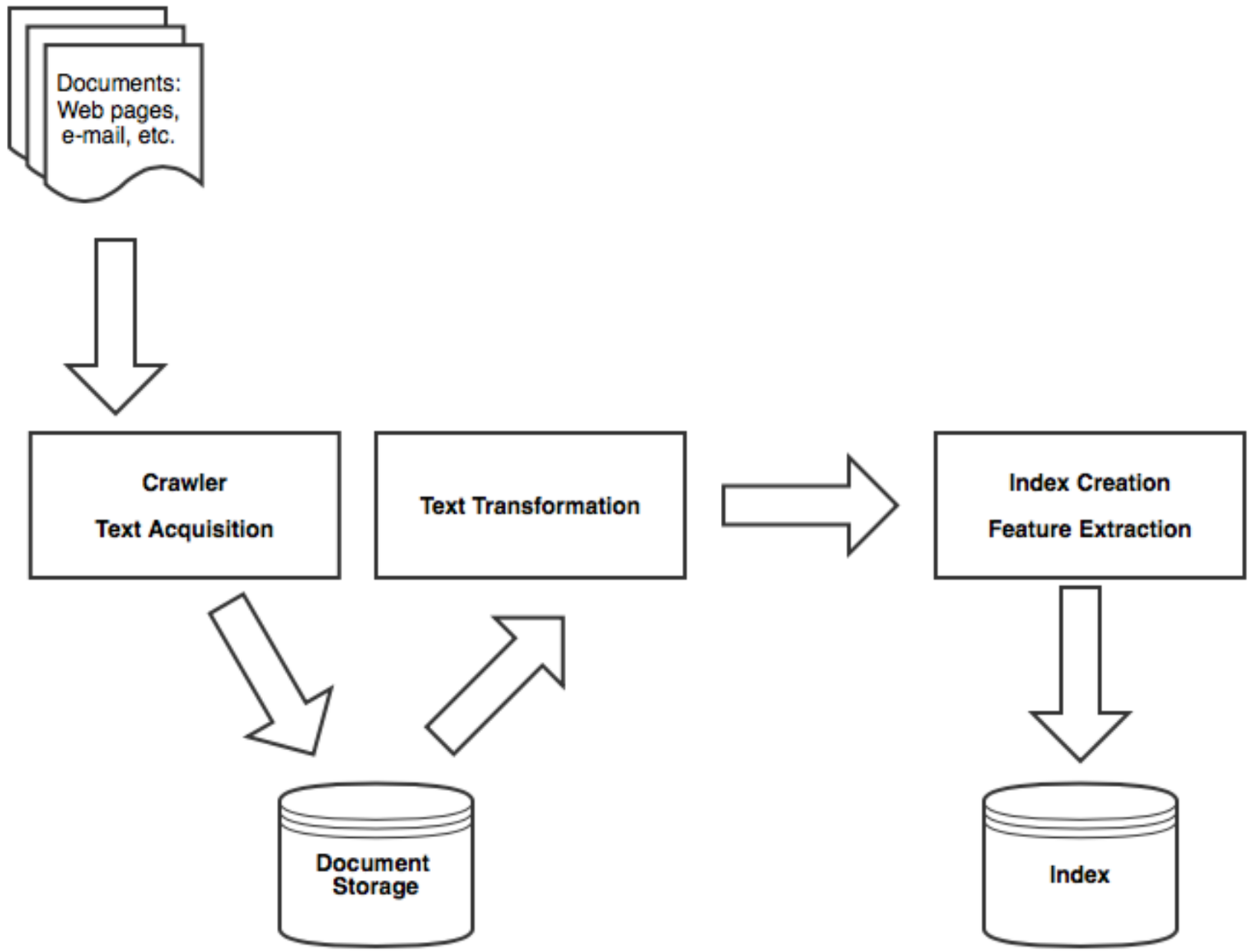
Text Transformation

- Analyzing the text
  - *Link Analysis*: The links to and from a document provide important relevance information, as does the anchor text of those links.
    - e.g. PageRank, which we'll cover later
    - More useful for web indexing than other document collections
  - *Information Extraction*: Some words are particularly informative, and classifiers have been built to recognize them
    - *Named entity recognition* identifies people, places, companies
    - Addresses, dates, job postings, etc. often get special handling

Text Transformation

- Analyzing the text
  - *Classification*: Identifies class-related metadata which impacts relevance
    - Topics, reading levels, sentiment, genre, authoritativeness, spamminess, etc.
    - These may or may not matter, depending on what you're doing
    - These classifiers can make a big difference for query relevance, and are sometimes closely-guarded secrets

Indexing Process



Index Creation

- Storing Document Statistics
  - The counts and positions of document terms are stored
    - *Forward Index*: Key is the document, value is a list of terms and term positions. Easiest for the crawler to build.
    - *Inverted Index*: Key is a term, value is a list of documents and term positions. Provides faster processing at query time.
  - Term weights are calculated and stored with the terms. The weight estimates the term's importance to the document.
    - The weights are used by ranking algorithms
    - e.g. *TF-IDF* ranks documents by the *Term Frequency* of the query term within the document times the *Inverse Document Frequency* of the term across all documents. Higher scores means you have more query terms which are not found in many documents.

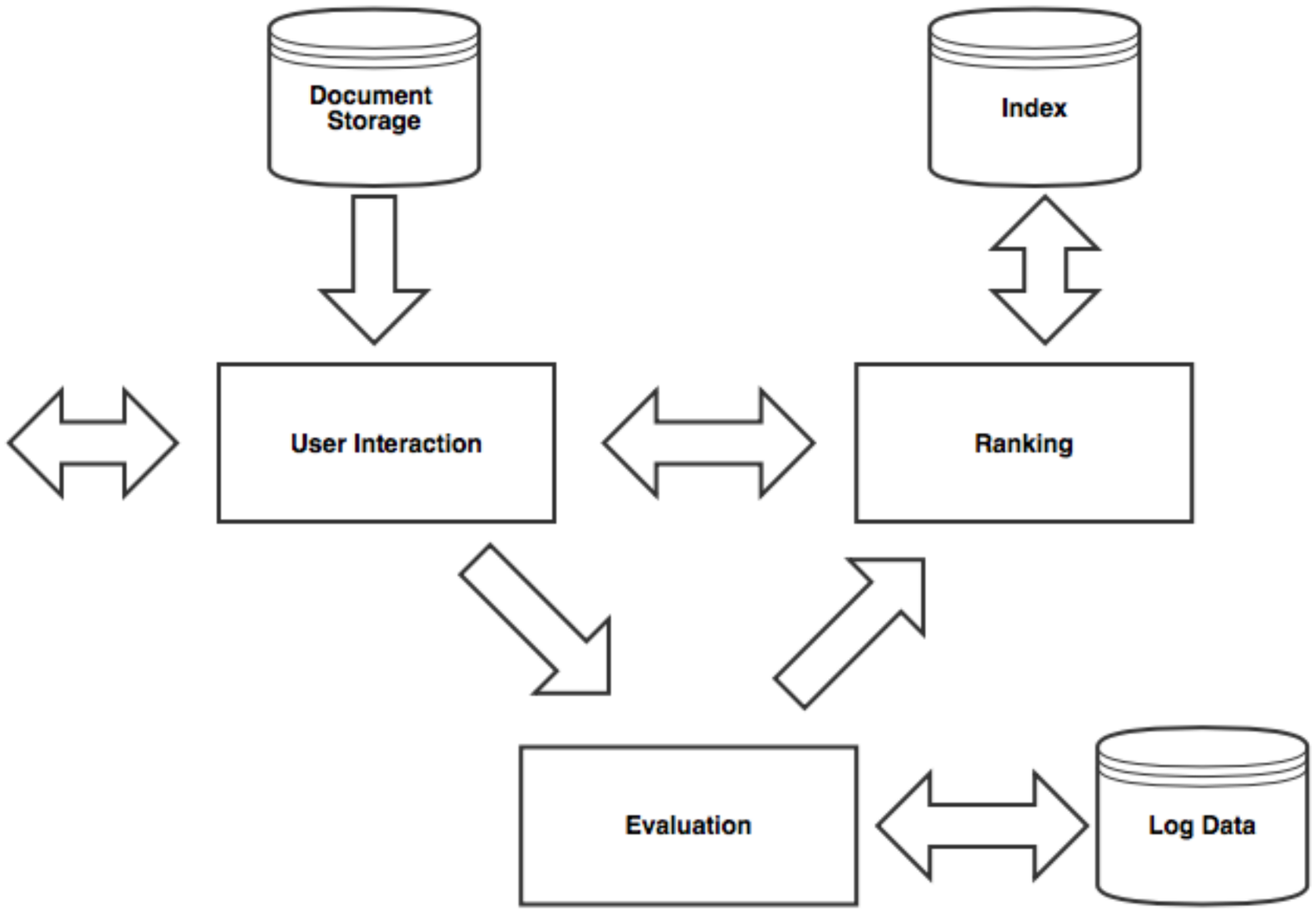
Index Creation

- Index Inversion
  - Essential for fast query performance
  - Converts document-level term positions to collection-level positions
  - Difficult for very large document collections
  - The index format must be carefully designed for fast reading, efficient (compressed) storage, many concurrent reads and writes, and data redundancy
  - If you have indexed private data, encryption and credential management may play an important role

Index Creation

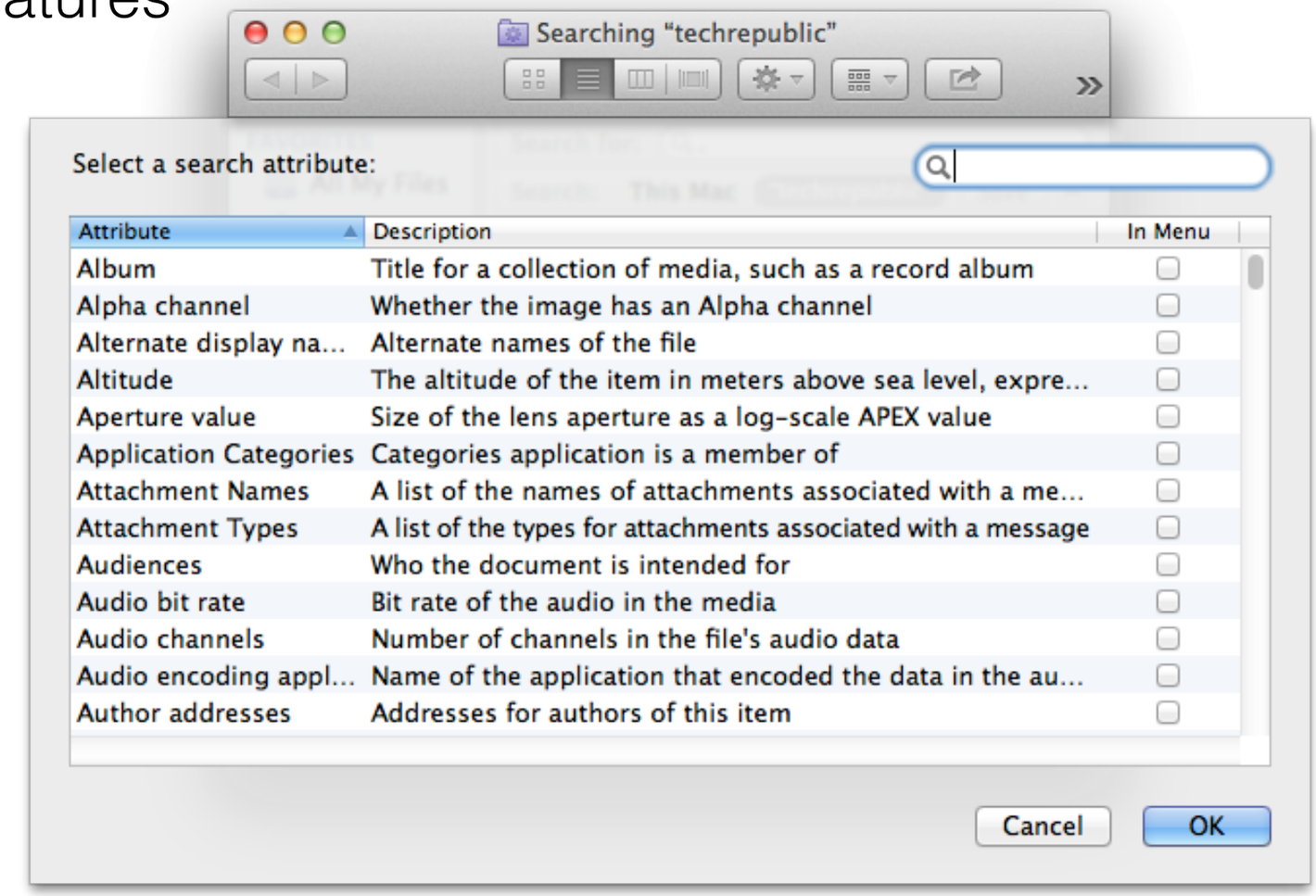
- Index Distribution
  - The index data must be replicated across many computers, and often many different sites
  - Essential for rapid processing of massive query numbers
  - Many variations on this process: distributing documents across sites, or distributing terms, or replicating the entire data set…
  - You often need to decide which queries to send to which sites, based on where the index is (or the *freshest* index)
  - P2P file sharing and *Distributed IR* involve searching across multiple sites

Query Process



User Interaction

- Query Input
  - A query language is defined, and a web site provides an interface for users to enter queries
    - Most queries are simple, but many search engines also have advanced language features
      - e.g. Boolean logic, date range or web site restrictions, searching custom index fields
      - Similar to SQL, but IR search tends to focus on *content* over *structure*

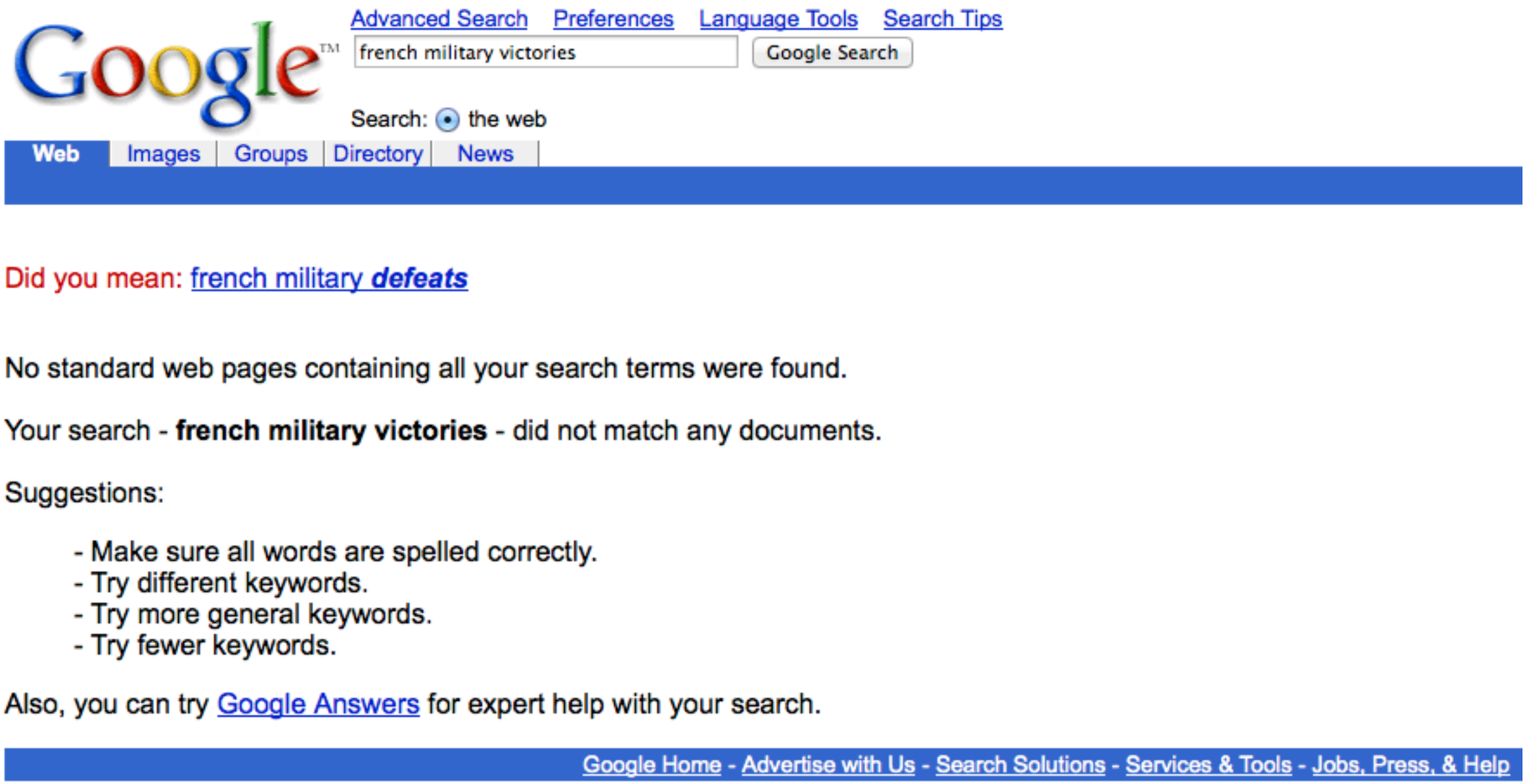


User Interaction

- Query Transformation
  - Improves the initial query, both before and after the initial search
  - *Spell checking* and *query suggestion* suggest improvements to the user, or run alternative queries in the background
  - *Query expansion* adds terms related to the query terms (e.g. synonyms, related entities)
  - *Relevance feedback* runs an initial query, then uses the top-ranked documents to expand the query for a second run

User Interaction

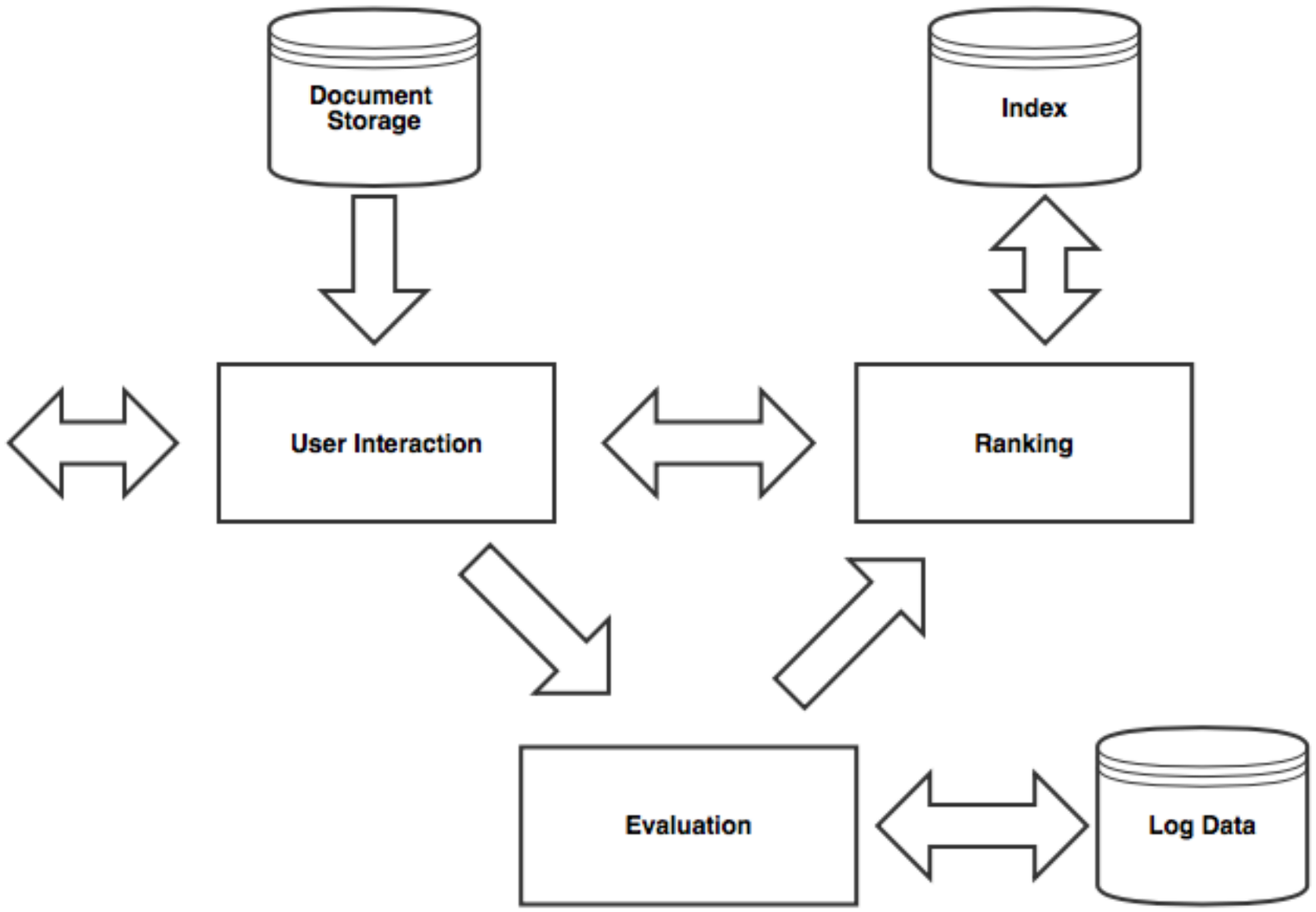
- Query suggestion (a prank)



User Interaction

- Results output
  - Displays the top-ranked results
  - Generates *snippets* to show how queries match documents
  - Highlights important words and passages
  - Retrieves query-relevant advertising
  - May *cluster* the results, or add additional content (e.g. Google One Box presents custom results when it's confident it knows what you're looking for)

Query Process



Ranking

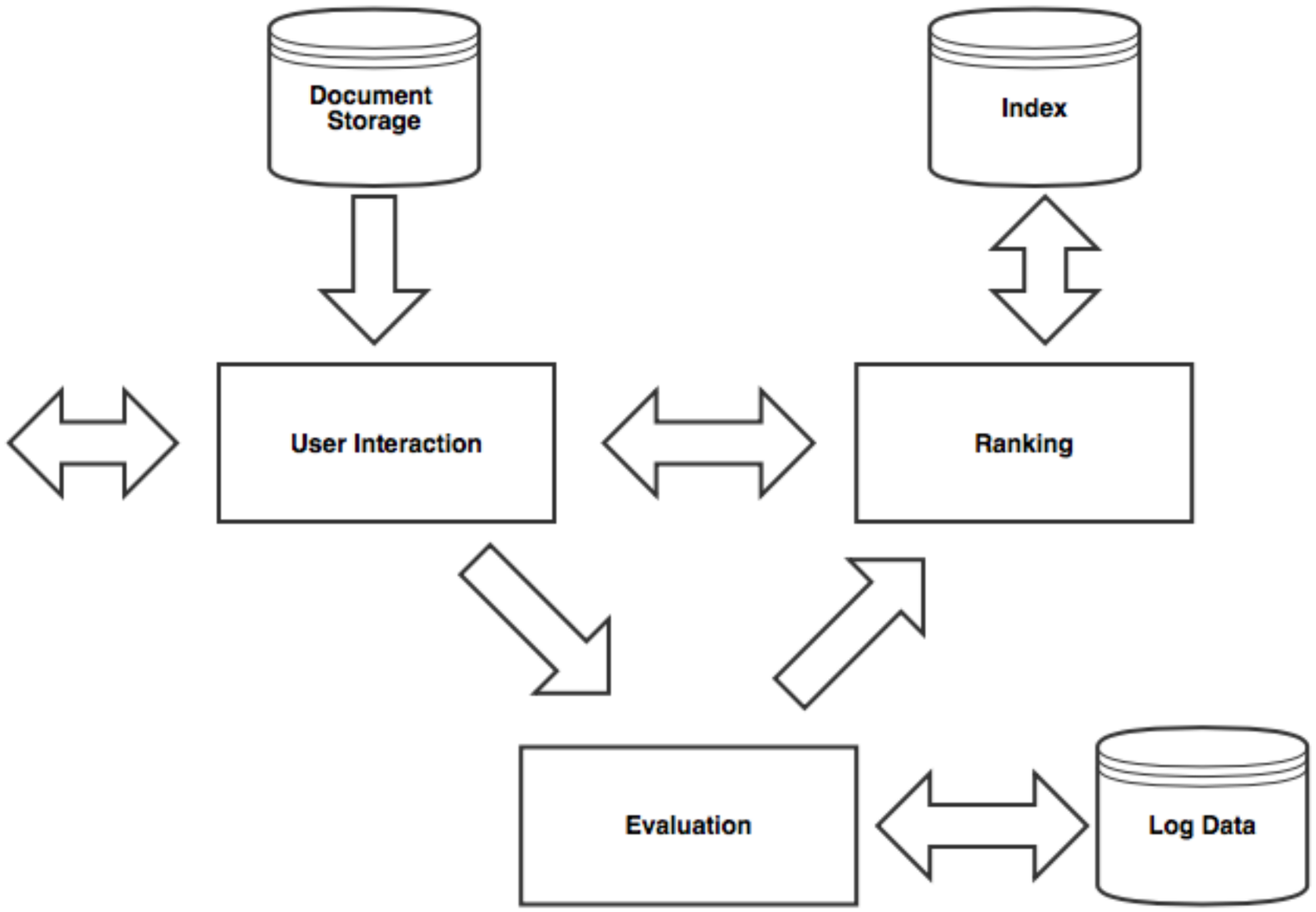
- Document scoring
  - A score is assigned to the most likely-relevant documents based on how well it matches the query.
  - Core component of a search engine, and often the most closely-guarded secret.
  - Many, many approaches and variations have been developed
  - The basic form is the dot product of query term weights and corresponding document weights:

$$\sum_i q_i d_i$$

Ranking

- Performance optimization
  - You generally have to run the ranking at query time, in much less than 1 second.
  - Efficient algorithm design is essential
    - *Term-at-a-time* vs. *document-at-a-time*
    - *Safe* vs. *unsafe* optimizations
- Distribution
  - Queries are processed in a distributed environment
  - A *query broker* distributes queries and assembles results
  - *Caching* is a form of distributed searching

Query Process



Evaluation

- Logging
  - Logging user interaction is an essential tool for measuring performance
  - *Query logs* and *clickthrough data* are used for query suggestion, spell checking, query caching, ranking, advertising search, …
- Ranking analysis
  - Given two different document rankings, which is better?
  - Some issues: *query relevance*, *topic coverage*, *subtopic diversity*
- Performance analysis
  - Measuring and tuning system efficiency

Further Reading

- Chapter 2 of *Search Engines* by Croft, Metzler, and Strohman

- Google – Inside Search: http://www.google.com/intl/en/insidesearch/

- The Anatomy of a Large-Scale Hypertextual Web Search Engine, Sergey Brin and Larry Page, 1998.