Chapter 2

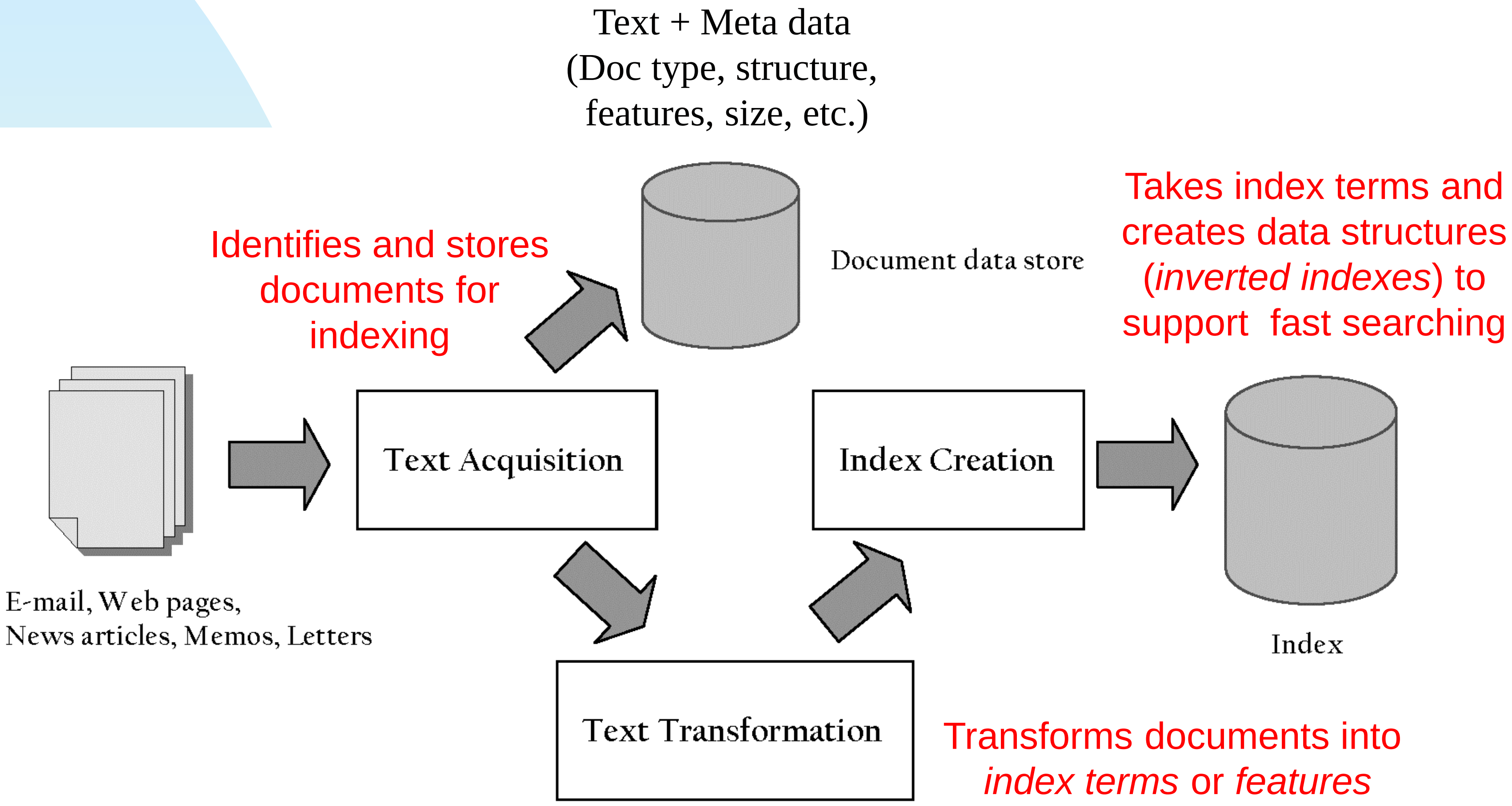
Architecture of a Search Engine

Search Engine Architecture

- A software architecture consists of *software* components, the *interfaces* provided by those components and the *relationships* between them
  - Describes a system at a particular level of abstraction
- Architecture of a search engine determined by *two* requirements
  - Effectiveness (*quality* of results)
  - Efficiency (*response time* and throughput)

Indexing Process

- One of the two major functions of search engine components



Text + Meta data
(Doc type, structure,
features, size, etc.)

Document data store

E-mail, Web pages,
News articles, Memos, Letters

Text Acquisition

Text Transformation

Index Creation

Index

Identifies and stores documents for indexing

Takes index terms and creates data structures (*inverted indexes*) to support fast searching

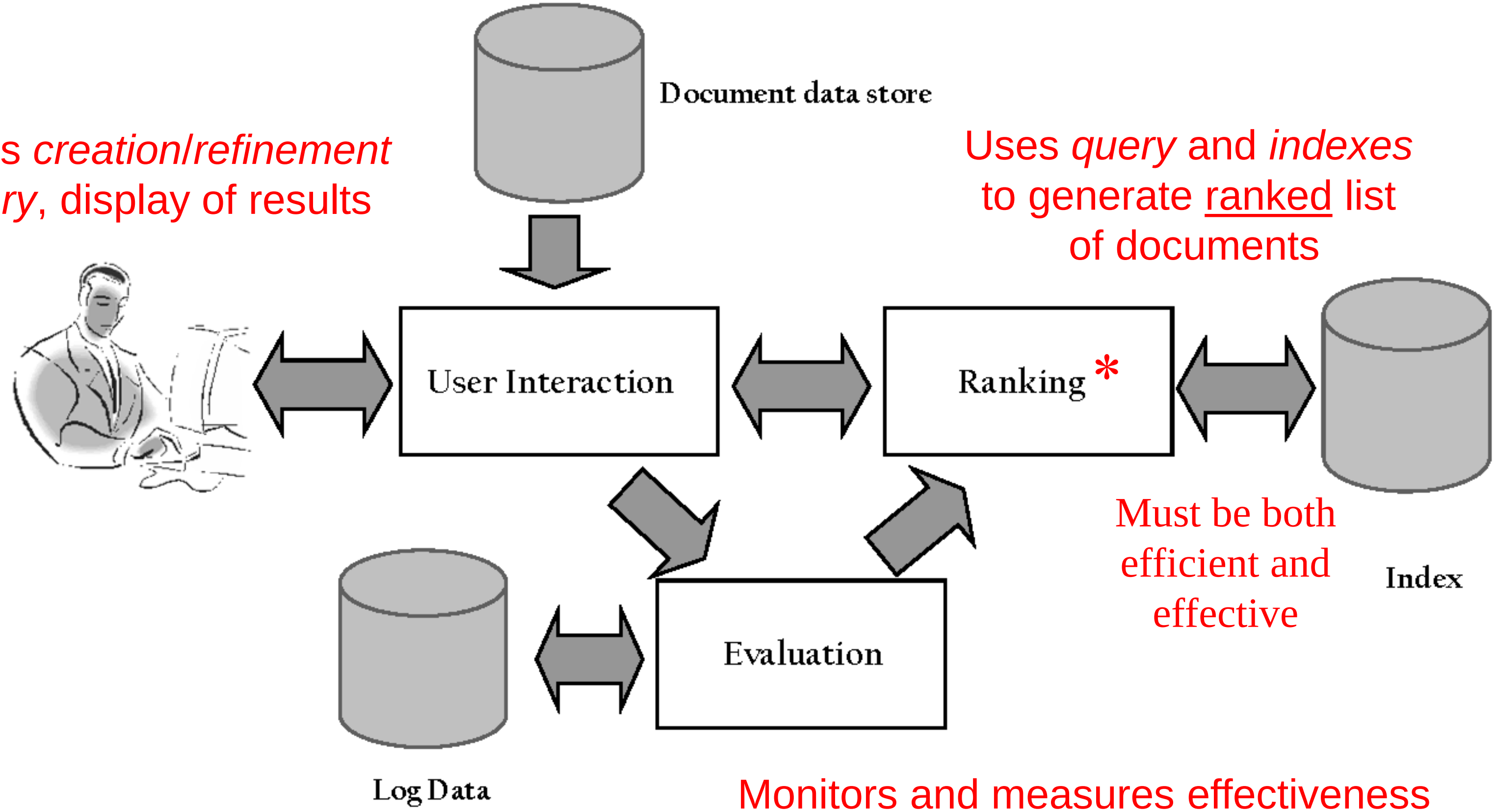
Transforms documents into *index terms* or *features*

Query Process

- Another major function of search engine components

Supports *creation/refinement* of *query*, display of results

Uses *query* and *indexes* to generate ranked list of documents



Must be both efficient and effective

Monitors and measures effectiveness and efficiency (primarily offline) using log data

Details: Text Acquisition

- **Crawler**
  - Identifies and acquires documents for search engines
  - Many types – Web, enterprise, desktop
  - Web crawlers follow *links* to find documents
    - Must efficiently find huge numbers of web pages (**coverage**) and keep them up-to-date (**freshness**)
    - Single site crawlers for *site search*
    - *Topical* or *focused* crawlers for specific search
  - *Document* crawlers for enterprise and desktop search
    - Follow links and scan directories

Text Acquisition

- **Feeds**
  - Real-time streams of documents
    - e.g., Web feeds for news, blogs, video, radio, TV
  - RSS (Rich Site Summary) is a commonly-used web feed format (which has been standardized)
- **Conversion**
  - Convert variety of documents into a consistent text plus metadata format
    - e.g., HTML, Word, PDF, etc. → XML
  - Convert text encoding for different languages
    - Using a Unicode standard like UTF-8

Text Transformation

- **Parser**
  - Processing the sequence of text *tokens* (i.e., words) in the document to recognize *structural* elements
    - e.g., titles, links, headings, etc.
  - Tokenizer recognizes “words” in the text (and queries) for comparison, a *non-trivial* process.
    - Must consider issues like capitalization, hyphens, apostrophes, non-alpha characters, separators, etc.
  - *Markup languages* such as HTML and XML often used to specify structure
    - *Tags* used to specify document *elements, e*.g., <h2>Overview</h2>
    - Document parser uses *syntax* of markup language (or other formatting) to identify structure

Text Transformation

- Stopping
  - Remove *common (function)* words, e.g., “and”, “or”, “the”, “in”
  - Some impact on *efficiency* & *effectiveness* (reduce the size of indexes)
  - A problem for some queries, e.g., “to be or not to be”
- Stemming
  - Group words derived from a *common stem*, e.g., “compute”, “computer”, “computers”, “computing”
  - Often *effective* (in terms of *matching*); not for all queries
  - Benefits vary for different languages (Arabic vs. Chinese)
- Information Extraction
  - Identify classes of index terms, e.g., *named entity recognizers*, identify classes such as people, locations, companies & dates, using part-of-speech tagging

Index Creation

- Document Statistics (collected during the indexing process)
  - Gathers *word counts* and *positions* of words and other features (e.g., *length* of documents as number of tokens)
  - Used in *ranking* algorithm (IR model dependent)
  - Stored in *lookup tables* for fast retrieval
- Weighting (during the query process)
  - Computes *weights* (the relative importance) of index terms
  - Used in ranking algorithm (IR model dependent)
  - e.g., *TF-IDF* weight
    - Combination of *term frequency* (*TF*) in document and *inverse document frequency* (*IDF*) in the collection

Index Creation

- *Inversion* of word list, converting doc-term to term-doc

Word	Doc#
pap	1
report	1
novel	1
technique	1
literat	1
result	1
technique	1
index	1
:	:
report	2
charact	2
human	2
being	2
ab	2
:	:

Sort →

Word	Doc#
ab	2
being	2
charact	2
human	2
index	1
literat	1
novel	1
pap	1
report	1
report	2
result	1
technique	1
technique	1
: :	: :

Remove → Duplicates

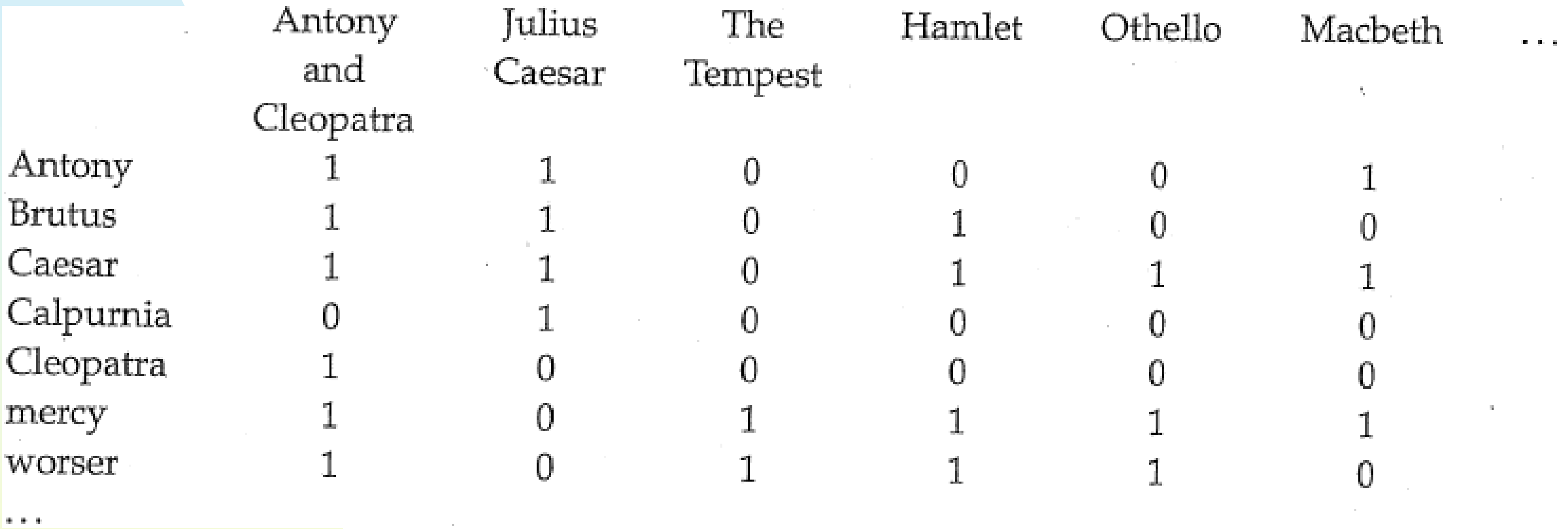
Word	Doc#	Freq
ab	2	1
being	2	1
charact	2	1
human	2	1
index	1	1
literat	1	1
novel	1	1
pap	1	1
report	1	1
	2	1
result	1	1
technique	1	2
: :	: :	

Term-Document Incidence Matrix

- Matrix element (*t*, *d*) = 1, if term *t* in document *d*; 0, otherwise
- Example.

Documents

Terms



	Antony and Cleopatra	Julius Caesar	The Tempest	Hamlet	Othello	Macbeth	...
Antony	1	1	0	0	0	1	
Brutus	1	1	0	1	0	0	
Caesar	1	1	0	1	1	1	
Calpurnia	0	1	0	0	0	0	
Cleopatra	1	0	0	0	0	0	
mercy	1	0	1	1	1	1	
worser	1	0	1	1	1	0	
...							

- Term-Term Correlation Matrix: *M* ο *M*^{T}, where *M* is a *term-document matrix*, *M*^{T} is the transpose of *M*, and 'ο' is the matrix composition operator

Index Creation

- Inversion
  - *Core* of indexing process
  - Converts document-term information to term-document for indexing
    - Difficult for very large numbers of documents to achieve high efficiency (for initial setup and subsequent updates)
    - *Multiple-level indexing* is desirable for very large number of indexes, e.g., B^+-tree indexing
  - Format of inverted file is designed for fast query processing
    - Must also handle *updates*, besides *creation*
    - *Compression* used for efficiency

User Interaction

- Query input
  - Provides *user interface* and *parser* for query language
  - Most web queries are very simple, such as keyword queries, other applications may use forms
  - Query language used to describe more complex queries and results of query transformation
    - Boolean queries
    - “Quotes” for phrase queries, indicating relationships among words
    - For keyword searches, *longer queries* yield *less results*
    - Similar to SQL language used in DB applications
    - IR query languages focus on content
  - Goal: yields good (better) results for a range of (specific) queries

User Interaction

- Query transformation
  - Performs text transformation on query text, e.g., stemming
  - Improves initial query, both *before* and *after* initial search
  - Spell checking/query suggestion, which provide alternatives (correcting spelling errors/specification) to the original query, is based on *query logs*
  - Modify the original query with additional terms
    - Query expansion: provides new, similar terms to a query based on *term occurrences* in documents or *query logs*
    - Relevance feedback: terms in previous retrieved *relevant documents*

User Interaction

- Results output
  - Constructs the display of *ranked* documents for a query
  - Generates *snippets* to show how queries match documents
  - *Highlights* important words and passages
  - May provide *clustering* and other visualization tools

Ranking

- Scoring
  - Calculates scores for documents using a ranking algorithm
  - Is a *core* component of search engine
  - Basic form of score is

$$\sum_{i=1}^{|V|} q_i \, d_i$$

    - where *V* is the *vocabulary* of the document collection
    - q_i & d_i are *query* and *document term weights*, respectively, e.g., TF/IDF or *term probability* for term *i*
  - Many variations of ranking algorithms and retrieval models
  - Must be calculated very rapidly to achieve *performance optimization*