Information Retrieval

Basics

by

Giovanni Stilo, PhD.

giovanni.stilo@univaq.it

Outline

- Introduction
- Information Retrieval Archietecture
- Documents Representation
  - Document parsing
  - Tokenization
  - Stopwords/Normalization
  - POS Tagging
  - Stemming
  - Deep Analysis
- Documents Indexing
- Document Querying/Ranking

Information Retrieval

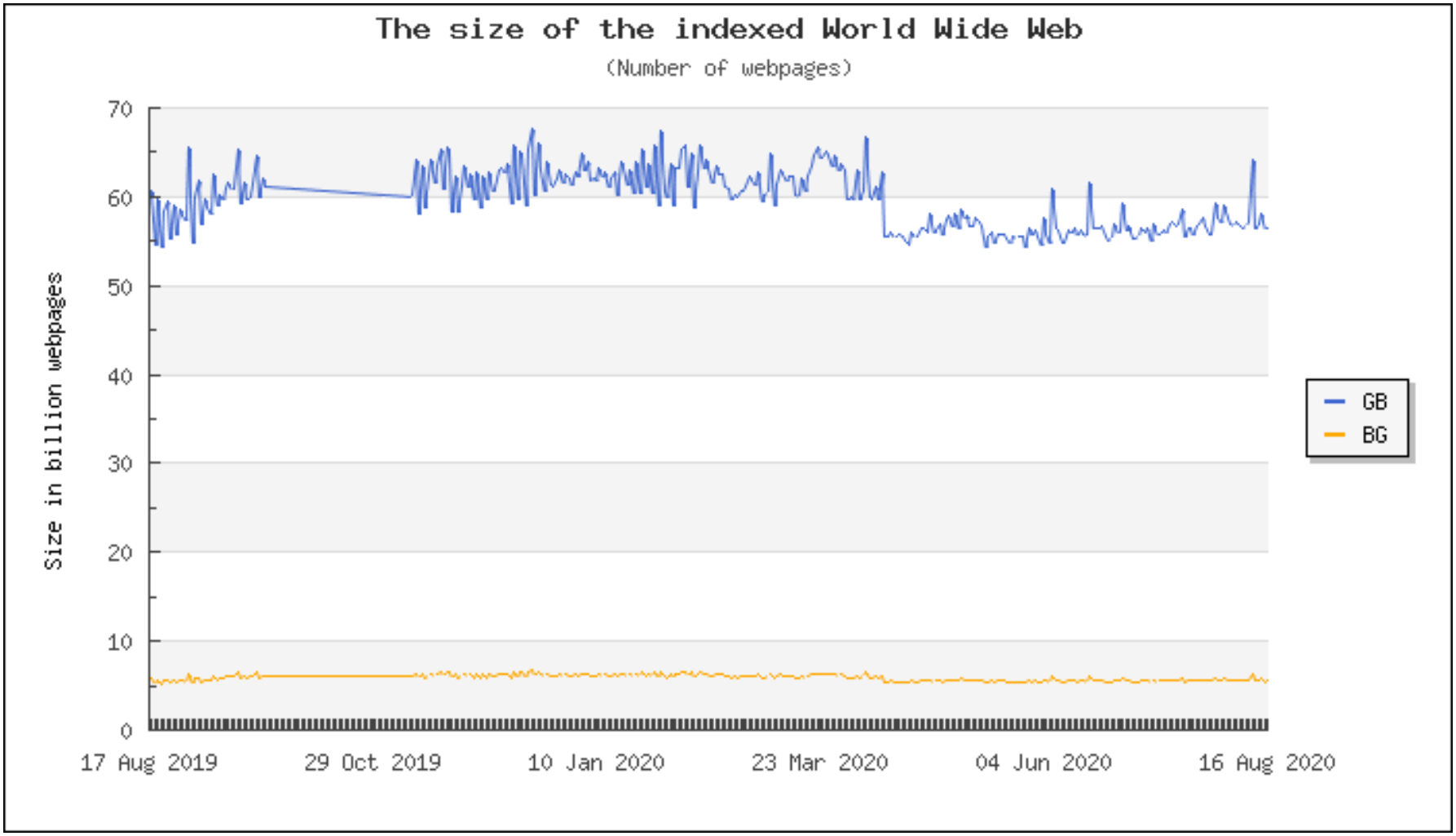
- Information Retrieval (IR) is finding material (usually documents) of an unstructured nature (usually text) that satisfies an information need from large collections (usually stored on computers).

- "*Usually*" text, but more and more: images, videos, data, services, audio..

- "*Usually*" unstructured (= no pre-defined model) but: Xml (and its dialects e.g. Voicexml..),RDF, html are "more structured" than plain txt or pdf

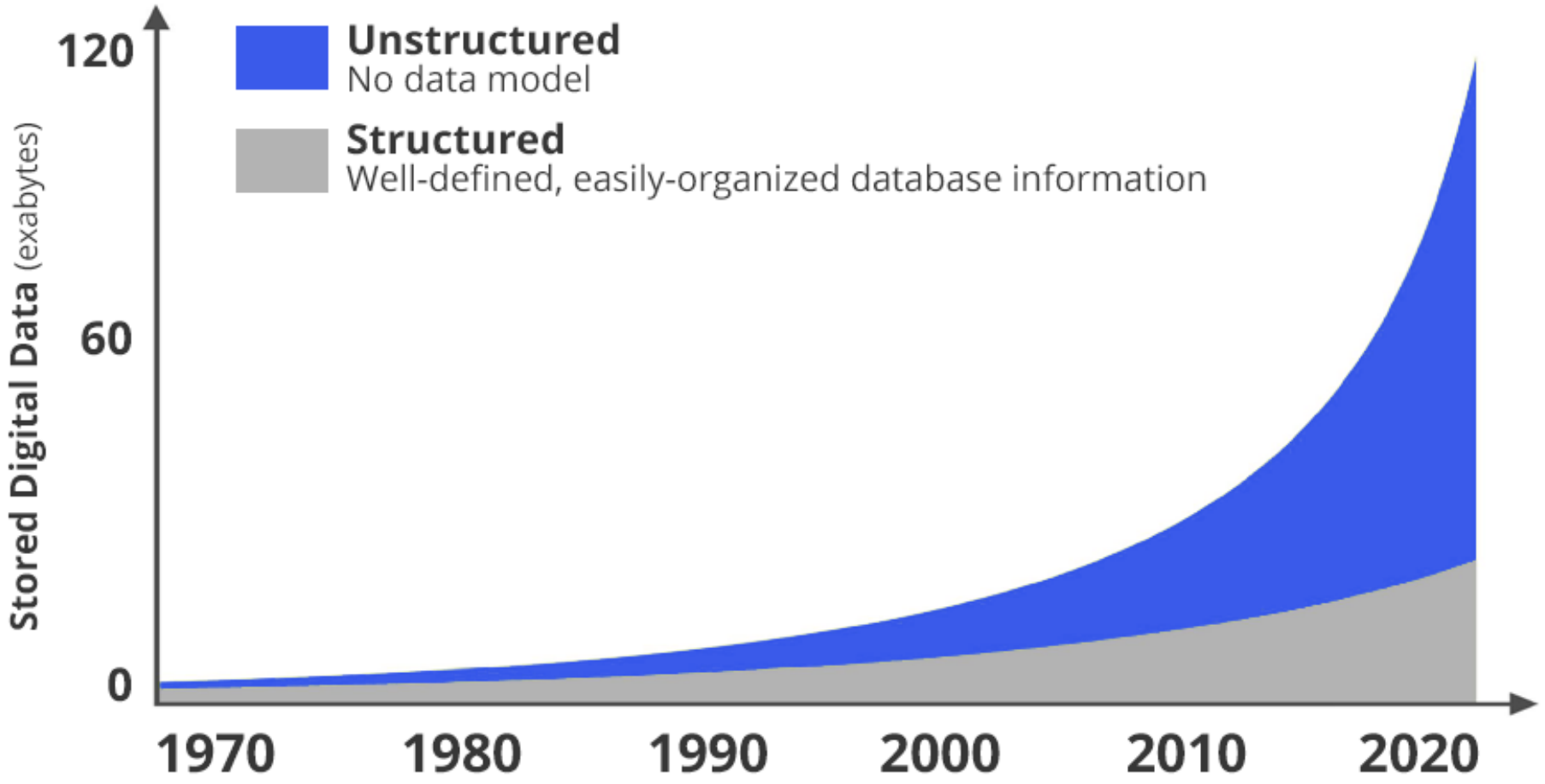
- "*Large*" collections: how large?? The Web! (The Indexed Web contains **at least 50 billion pages** .)

Indexed pages (19-20)

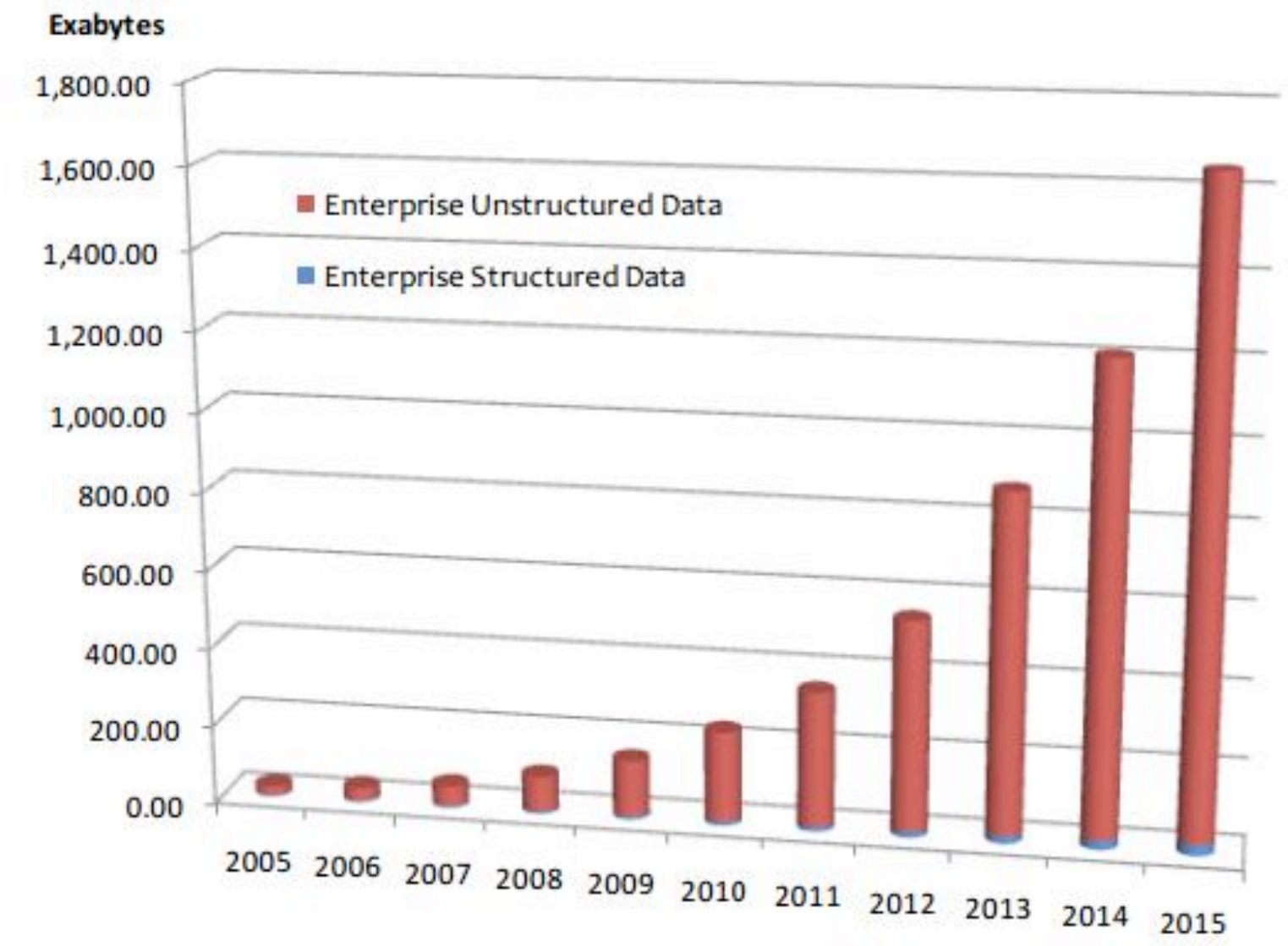


https://www.worldwidewebsize.com/

Unstructured vs. structured



Unstructured Data in Enterprise



Structured Data

- Structured data tends to refer to information in "tables"

Employee	Manager	Salary
Smith	Jones	50000
Chang	Smith	60000
Ivy	Smith	50000

Typically allows:

numerical **range** and **exact match**

For queries, e.g.:

Salary < 60000 AND Manager = Smith.

Unstructured data

- Typically refers to **free-form text**

- Allows:
  - Keyword queries including operators **Λ**, **V**:
    - (information Λ (retrieval V extraction))

  - Or More sophisticated "concept" queries, e.g.,
    - find all web pages dealing with *Social Mining*

Semi-structured data

- In fact almost no data is “unstructured”
- E.g., this slide has distinctly identified zones such as the *Title* and *Bullets*
- This structure allows for “semi-structured” search as:
  - *Title* contains “data” AND *Bullets* contain “search”
  - Only **old plain txt format** is truly unstructured (though even natural language does have a structure..)

IR Tasks – not only retrieval

- Clustering: Given a set of docs, group them into clusters based on their contents.
- Classification: Given a set of topics, plus a new doc *d*, decide which topic(s) *d* belongs to (eg spam vs nospam).
- Information Extraction: Find all snippets dealing with a given topic (e.g. *company merges*)
- Question Answering: deal with a wide range of question types including: facts, lists, definitions, How, Why, hypothetical, semantically constrained, and cross-lingual questions;
- Opinion Mining: Analyse/summarize sentiment in a text (e.g. TripAdvisor)
- All the above, applied to **images, video, audio, social networks;**

Terminology

Searching: Seeking for specific information within a body of information. The result of a search is a set of **hits** (e.g. the list of web pages matching a query).

Browsing: Unstructured exploration of a body of information (e.g. a web browser traverses and retrieves info on the WWW).

Crawling: Moving from one item to another (in a Network) following links, such as citations, references, etc.

Scraping: pulling/extracting content from pages

Terminology

- **Query:** A string of text, describing the information that the user is seeking.
  Each word of the query is called a **search term** or **keyword**.
  - A query can be a single search term, a string of terms, a phrase in natural language, or a stylized expression using special symbols.
- **Full text searching:** Methods that compare the query with **every word in the text**, without distinguishing the grammatical function (meaning, position) of the various words.
- **Fielded searching:** Methods that search on specific bibliographic or **structural fields**, such as author or heading.

Examples of Search Systems

Find file on a computer system (e.g. *Spotlight* for Macintosh).

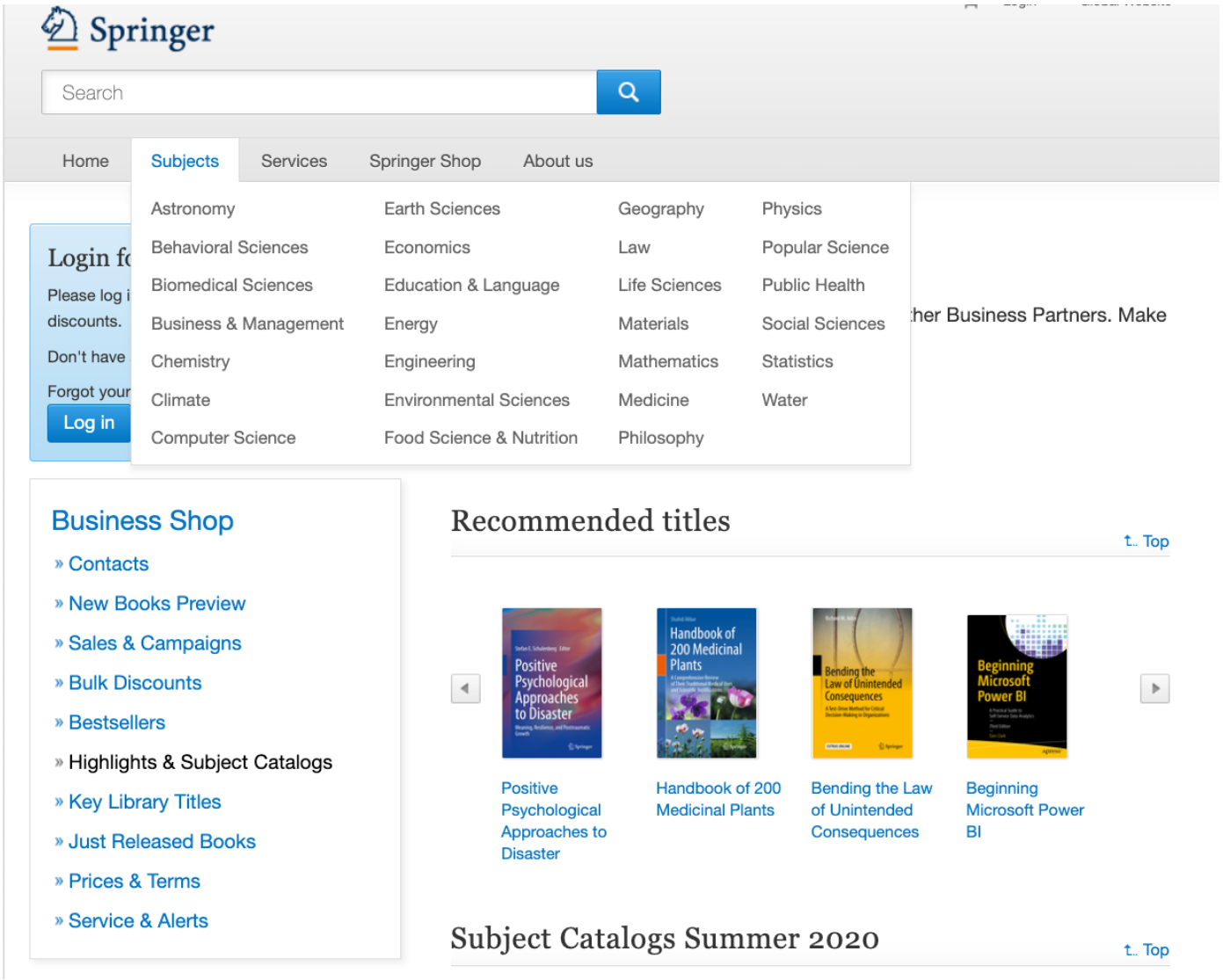
Library catalog for searching bibliographic records about books and other objects (e.g. *Library of Congress catalog*).

Abstracting and indexing system to find research information about specific topics (e.g. *Medline* for medical information).

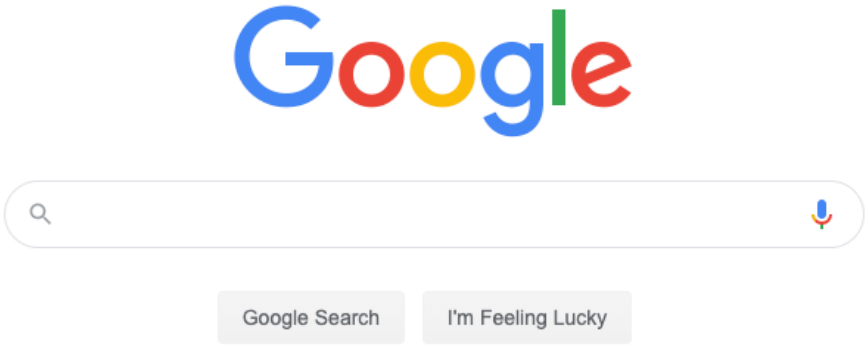
Web search service to find web pages (e.g. *Google*).

Users/Messages find messages/status/users in a social network

Library/Book Catalogue

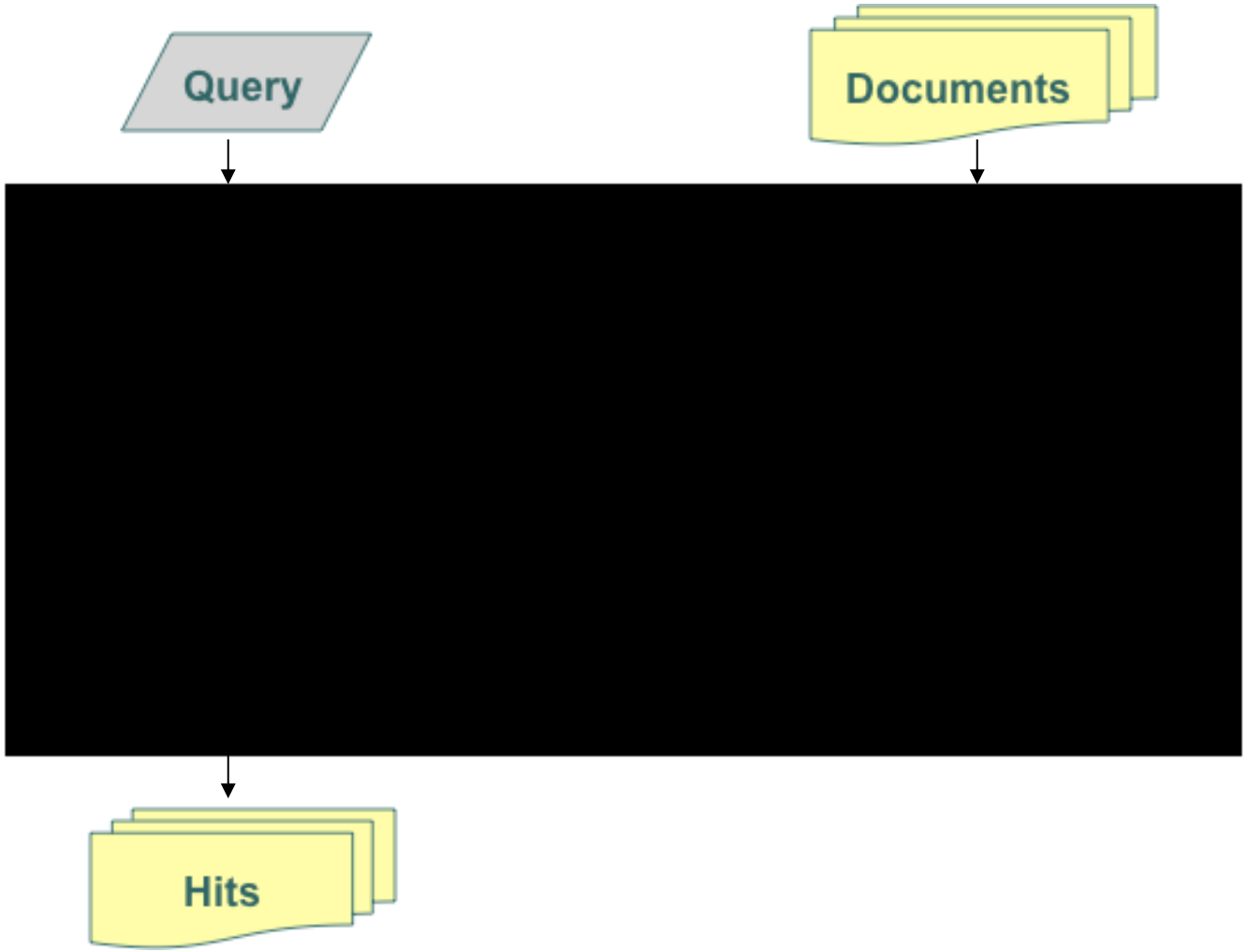


Web Search

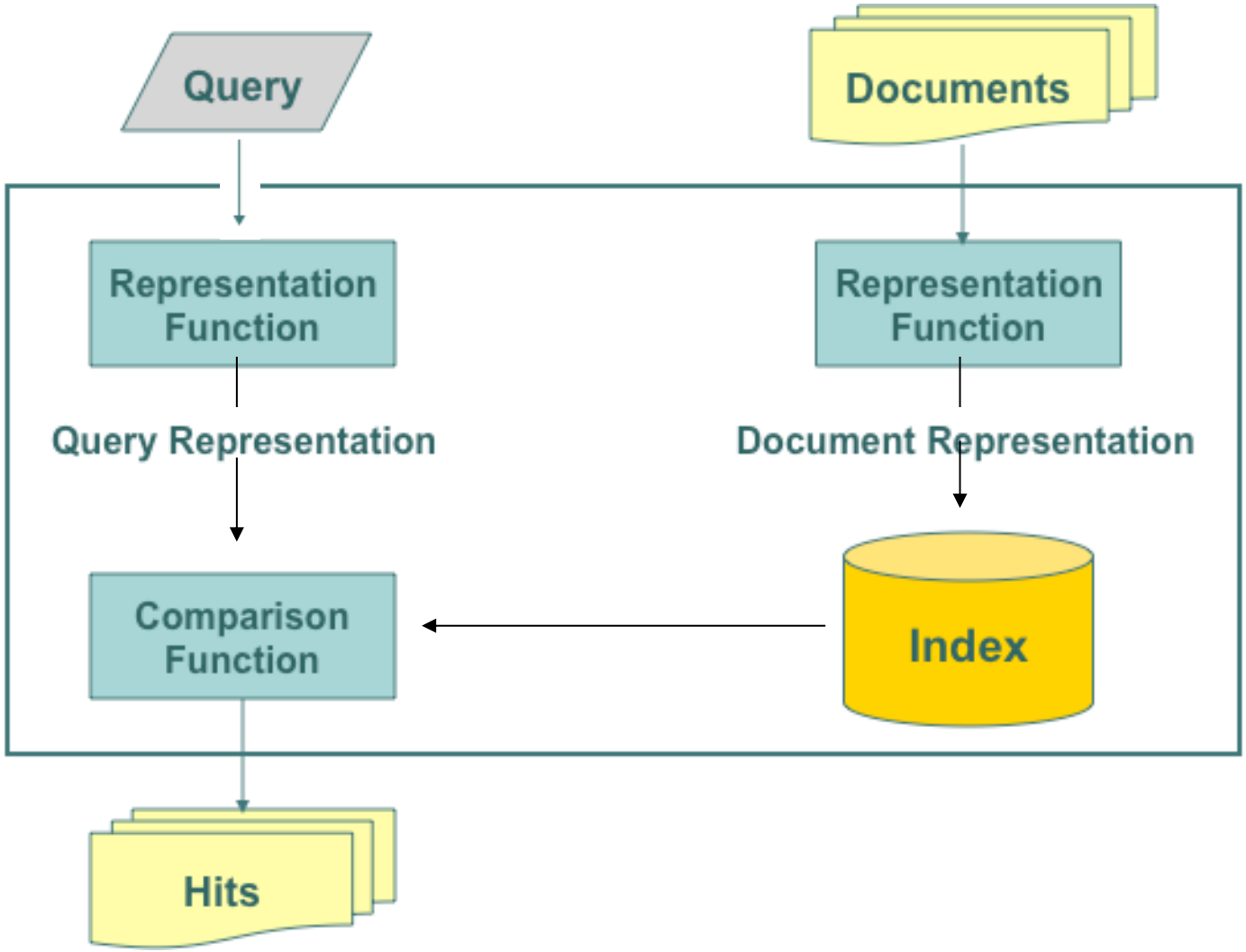


Architecture of an IR system

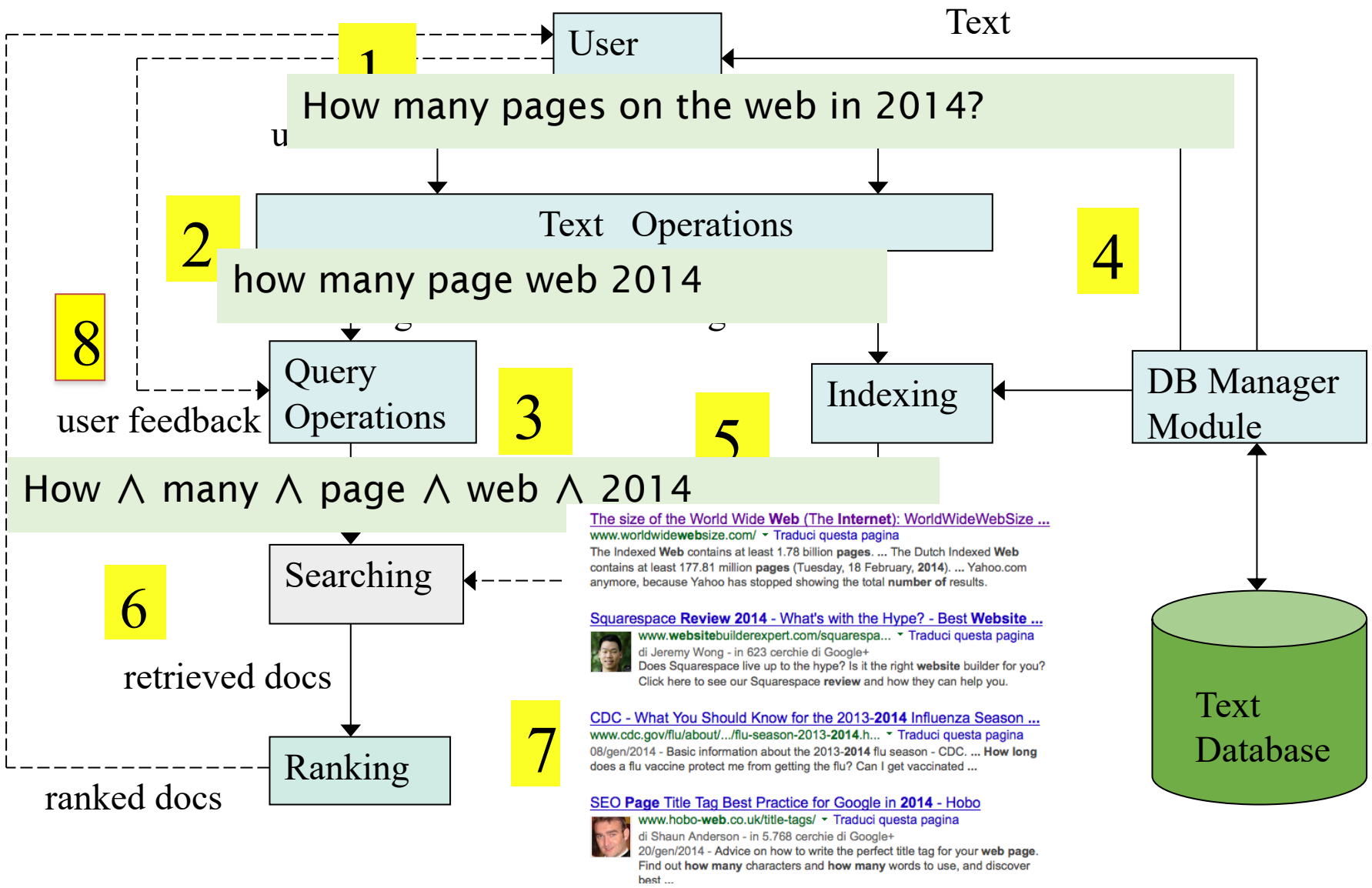
IR Black Box



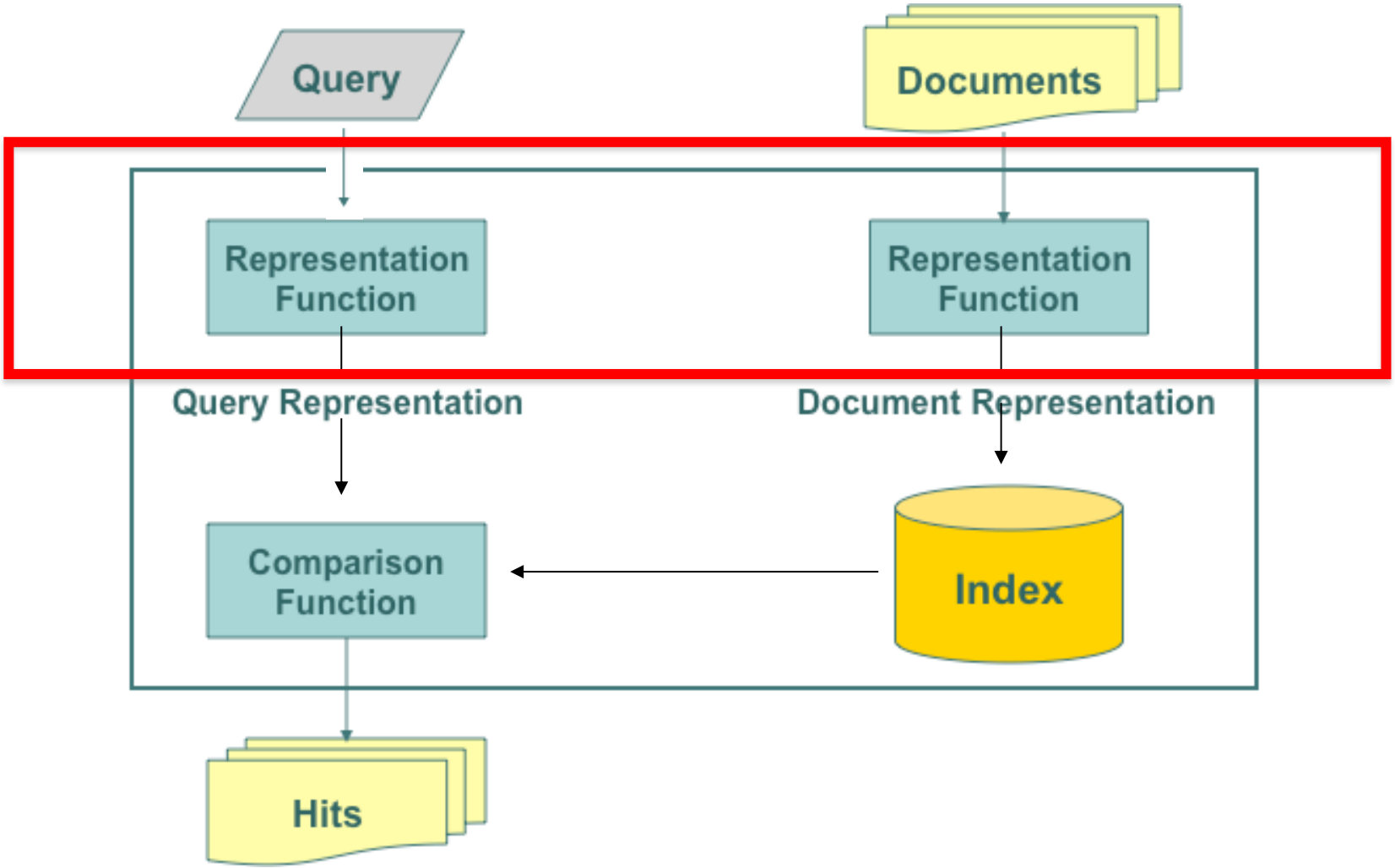
Inside The IR Black Box



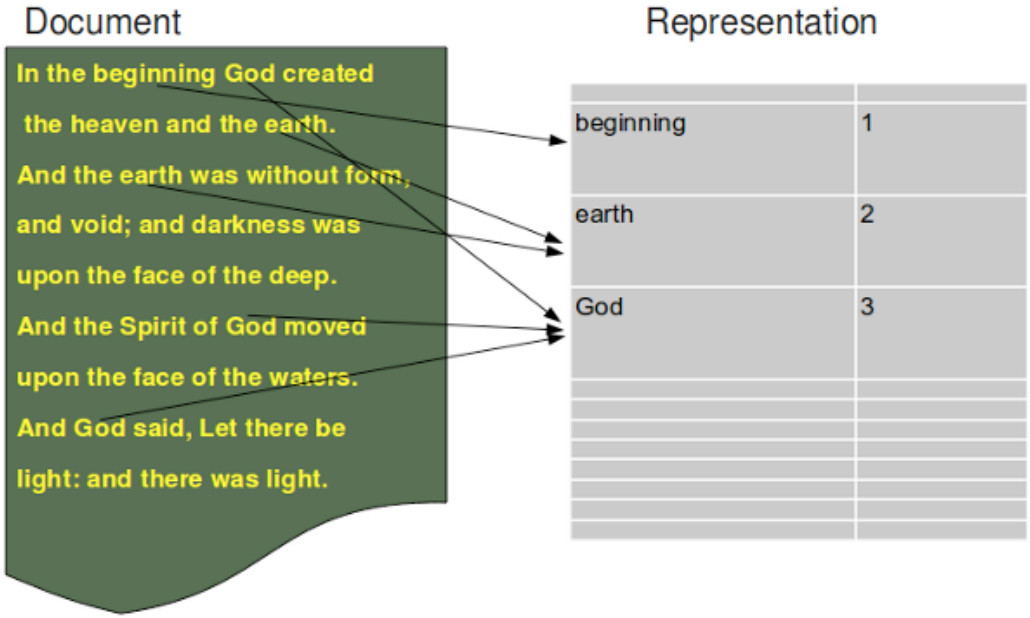
Workflow


User
Text
1
How many pages on the web in 2014?
u
2
Text Operations
how many page web 2014
4
8
Query Operations
Indexing
DB Manager Module
user feedback
3
5
How ∧ many ∧ page ∧ web ∧ 2014
Searching
6
retrieved docs
Ranking
ranked docs
7
Text Database
The size of the World Wide Web (The Internet): WorldWideWebSize ...
www.worldwidewebsize.com/ Traduci questa pagina
The Indexed Web contains at least 1.78 billion pages. ... The Dutch Indexed Web contains at least 177.81 million pages (Tuesday, 18 February, 2014). ... Yahoo.com anymore, because Yahoo has stopped showing the total number of results.
Squarespace Review 2014 - What's with the Hype? - Best Website ...
www.websitebuilderexpert.com/squarespa... Traduci questa pagina
di Jeremy Wong - in 623 cerchie di Google+
Does Squarespace live up to the hype? Is it the right website builder for you? Click here to see our Squarespace review and how they can help you.
CDC - What You Should Know for the 2013-2014 Influenza Season ...
www.cdc.gov/flu/about/.../flu-season-2013-2014.h... Traduci questa pagina
08/gen/2014 - Basic information about the 2013-2014 flu season - CDC. ... How long does a flu vaccine protect me from getting the flu? Can I get vaccinated ...
SEO Page Title Tag Best Practice for Google in 2014 - Hobo
www.hobo-web.co.uk/title-tags/ Traduci questa pagina
di Shaun Anderson - in 5.768 cerchie di Google+
20/gen/2014 - Advice on how to write the perfect title tag for your web page. Find out how many characters and how many words to use, and discover best ...

Inside The IR Black Box

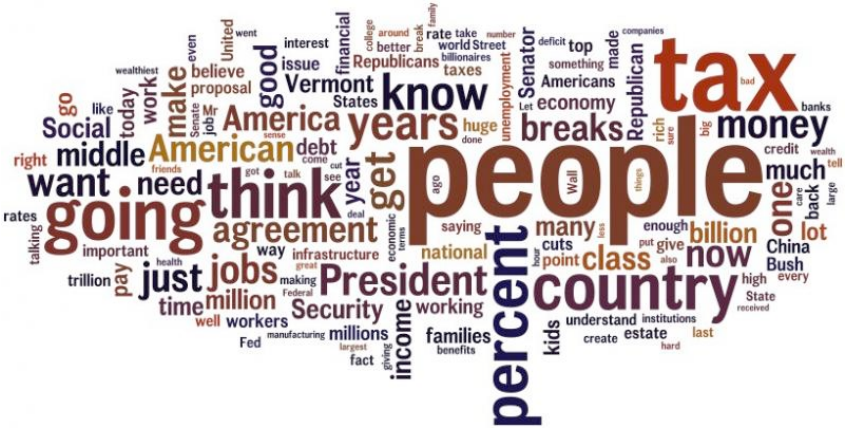


Representation

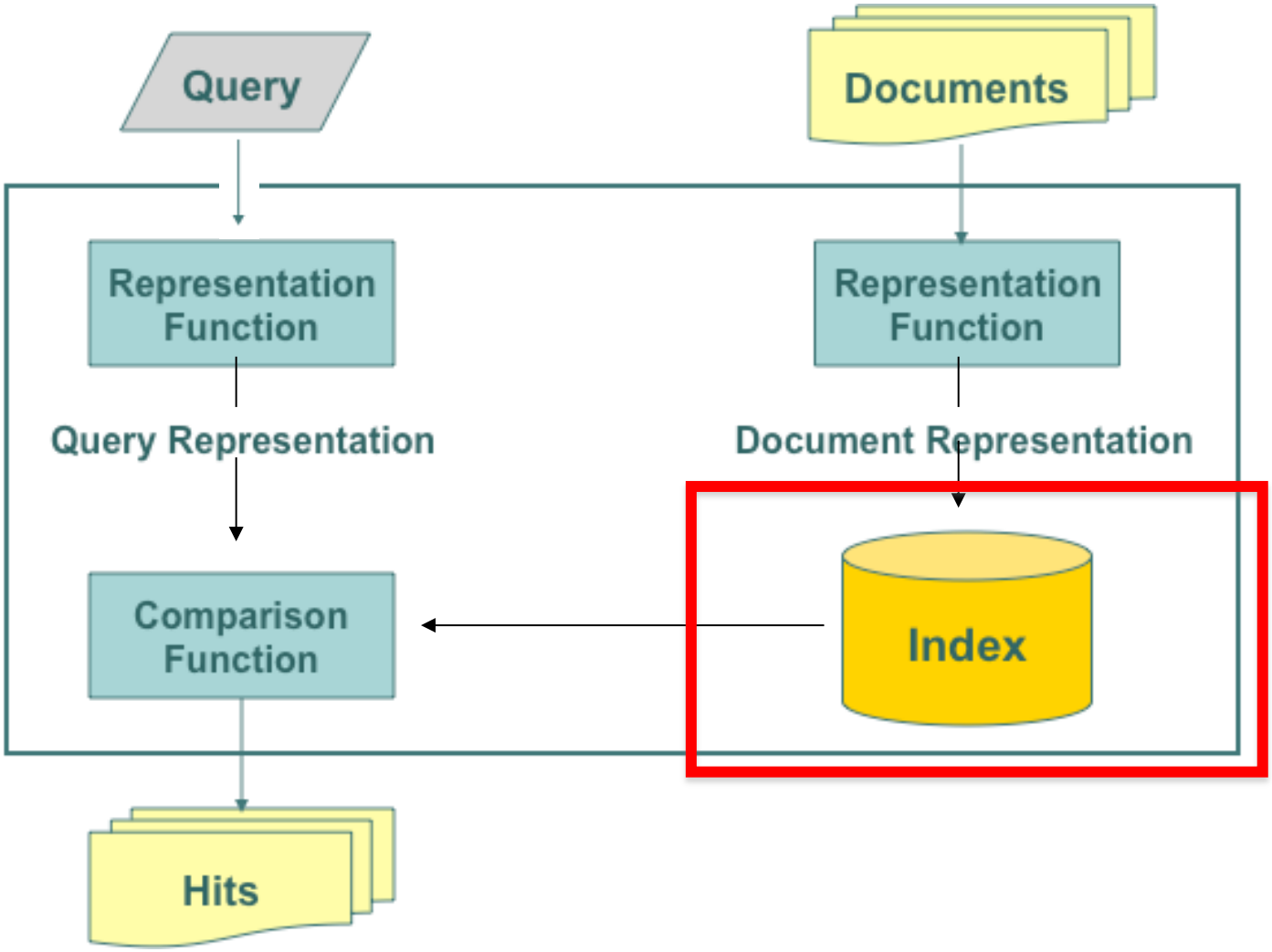


tables

clouds



Inside The IR Black Box



Indexing

We need a data structure that improves the speed of words / documents retrieval

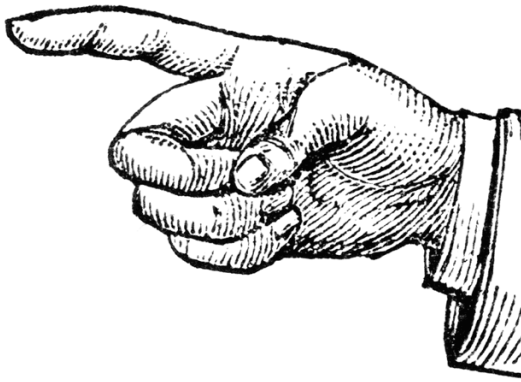
The corner of my mouth turned up in a wistful half-smile. "I used to think of you that way, you know. Like the sun. My personal sun. You balanced out the clouds nicely for me."

He sighed. "The clouds I can handle. But I can't fight with an eclipse."

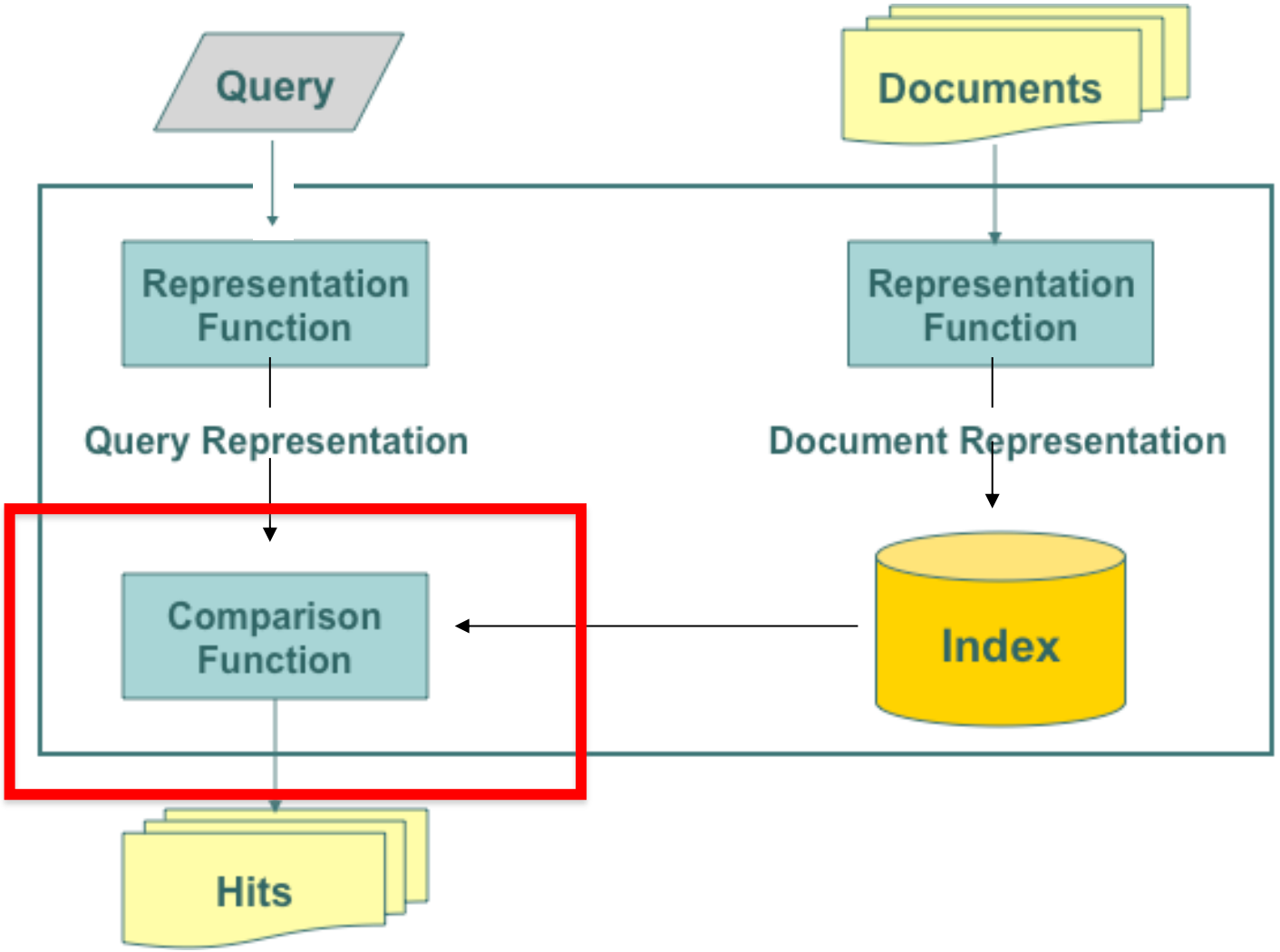
I touched his face, laying my hand against his cheek. He exhaled at my touch and closed his eyes. It was very quiet. For a minute I could hear the beating of his heart, slow and even.

"Tell me the worst part for you," he whispered.

Points at words in texts

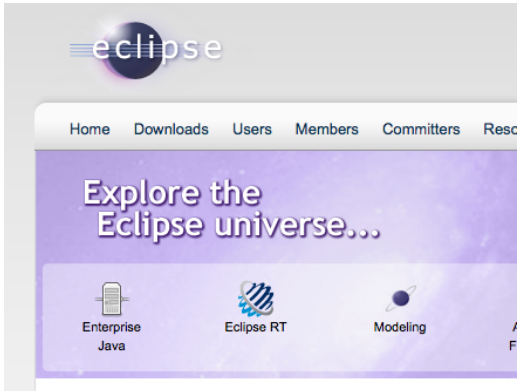


Inside The IR Black Box



Sorting & Ranking

Eclipse



Eclipse (I) (2010)

PG-13 124 min - Adventure | Dran
- 30 June 2010 (Italy)

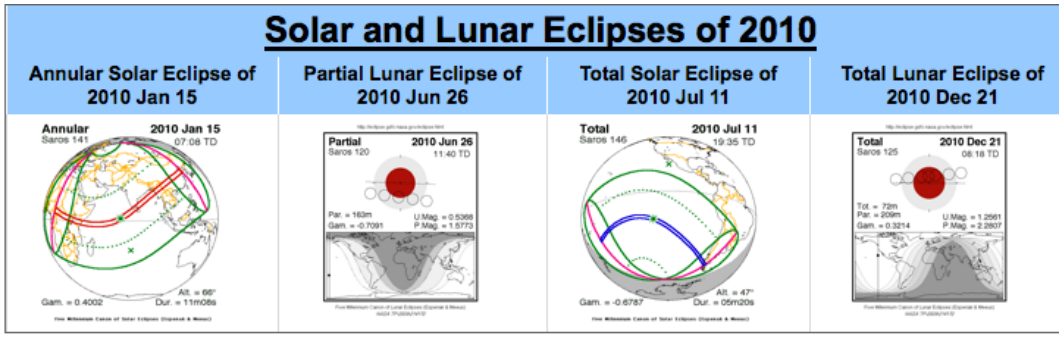
Users: (62,911 votes) 467 reviews | C
Metascore: 58/100 (based on 38 revie

As a string of mysterious killings high school graduation is fast ap choose between her love for vam friendship with werewolf Jacob.

Director: David Slade

Writers: Melissa Rosenberg (sc (novel)

Stars: Kristen Stewart, Robert



Sorting & ranking

When a **user** submits a **query** to a **search system**, the system returns a set of **hits (or result-set)**. With a large collection of documents, the set of hits maybe very large.

The value to the user **depends** on the **order** in which the hits are presented.

Three main methods:

- **Sorting** the hits, e.g., by date (more recent are better.. Is this true?)
- **Ranking** the hits by **similarity** between query and document
- **Ranking** the hits by the **importance** of the documents

Next (more details on)

- Representation
- Indexing
- Ranking

Document Representation

- **Objective**:
  given a document in whatever format (txt, html, pdf..)
  provide a formal, structured representation of the document (e.g. a vector whose attributes are words, or a graph, or..)

- **Several steps are needed from document downloading to the final selected representation**

- The most common representation model is “bag of words”

The bag-of-words model example

Tim Berners-Lee on November 18, 2005.

Background and early career [edit]

Sir Tim Berners-Lee's parents, both mathematicians, were employed together on the team that built the Ferranti Mark 1, one of the earliest computers. They taught their son to use mathematics everywhere, even at the dinner table. Berners-Lee attended Sheen Mount Primary School, before moving on to study his O-Levels and A-Levels at Emanuel School in Battersea, London where a computer centre is dedicated in his name.

He is an alumnus of The Queen's College, Oxford. While at Queen's, Berners-Lee built a computer with a soldering iron, TTL gates, an M6800 processor and an old television. During his time at university, he was caught hacking with a friend and was subsequently banned from using the university's computer. He graduated in 1976 with a degree in physics.

He met his first wife Jane while at Oxford University and they married soon after they started work in Poole, Dorset. After graduation, Berners-Lee was employed at Plessey Controls Limited in Poole as a

D_i = (..,..,…after,..attend,..both,..build,.before, ..center, college,…computer,.dinner,……..university,..work)

WORD ORDER DOES NOT MATTER!!!

Bag of Words Model

- This is the most common way of representing documents in information retrieval;

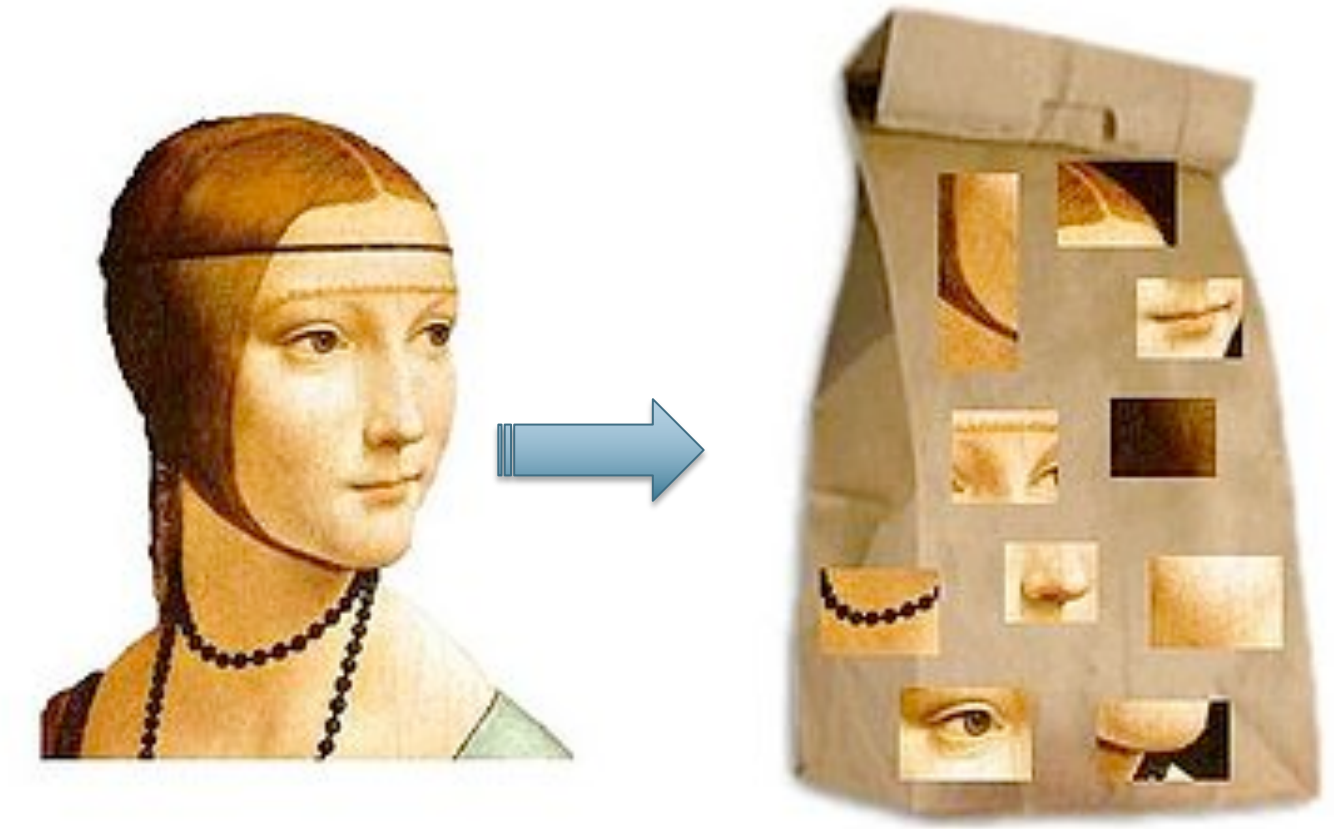
- Variants of this model include:

  - **How to weight a word** within a document (boolean, tf*idf, etc.)
    - Boolean: 1 is the word i is in doc j, 0 else
    - Tf*idf and others: the weight is a function of the word **frequency** in the document, and of the frequency of documents whith that word

  - **What is a "word":**
    - single, inflected word ("going"),
    - lemmatised word (going, go, gone→go)
    - Multi-word, proper nouns, numbers, dates ("board of directors", "John Wyne", "April, 2010"
    - Meaning: (plan,project,design→PLAN#03)

Bag of Words (images)

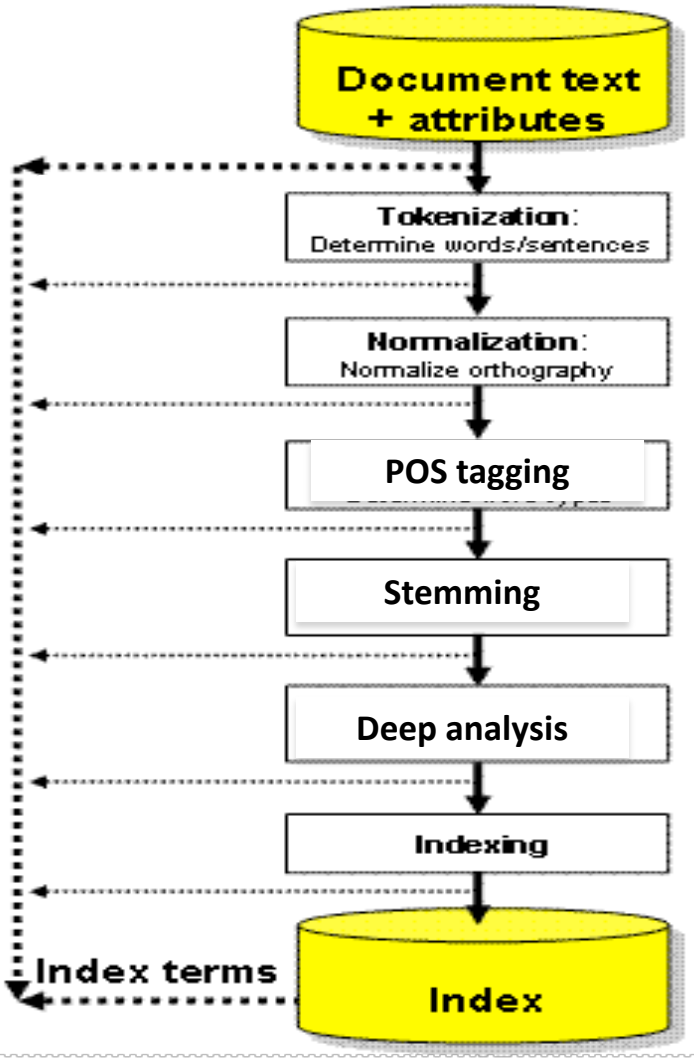
Bag of Words model is also used for images ("words" are now image features)



Phases in document processing

1. Document parsing
2. Tokenization
3. Stopwords/Normalation
4. *POS Tagging*
5. Stemming
6. *Deep Analysis*
7. Indexing

Notice that intermediate steps can be skipped



Document Parsing

Document parsing implies scanning a document and transforming it into a "bag of words" but: which words?

- We need to deal with format and language of each document:
  - What format is it in?
  - pdf/word/excel/html?
  - What language is it in?
  - What character set is in use (latin, greek, chinese..)?

But these tasks are often done heuristically …

Doc parsing: Format/language

- Documents being indexed can include docs from many different languages
  - A single index may have to contain terms of several languages.
- Sometimes a document or its components can contain multiple languages/formats(multilinguality)
  - ex : French email with a German pdf attachment.
- What is a “unit” document?
  - A single file? A zipped group of files
  - An email/message?
  - An email with 5 attachments?
  - A single web page of a full web site?
  - Phrases
  - Users
  - Audio File

Tokenization

- Input: "***Friends, Romans and Countrymen***"
- Output: Tokens
  - ***Friends***
  - ***Romans***
  - ***Countrymen***
- A token is an instance of a sequence of characters
- Each such token is now a candidate for an index entry, after further processing
  - Described below
- But what are valid tokens to emit?

Tokenization

- Issues in tokenization:
  - ***Finland's capital*** $\rightarrow$
    Finland? Finlands? Finland's?
  - ***Hewlett-Packard*** $\rightarrow$ ***Hewlett*** and ***Packard*** as two tokens?
    - ***state-of-the-art***: break up hyphenated sequence.
    - ***co-education***
    - ***lowercase***, ***lower-case***, ***lower case*** ?
  - ***San Francisco***: one token or two?
    - How do you decide it is one token?
    - ***cheap San Francisco-Los Angeles fares***

Tokenization : Numbers

- ***3/12/91***
- ***Mar. 12, 1991***
- ***12/3/91***
- ***55 B.C.***
- ***B-52***
- ***(800) 234-2333***
- ***1Z9999W99845399981 (package tracking numbers)***
  - Often have embedded spaces (ex. IBAN/SWIFT)
  - Older IR systems may not index numbers
    - Since their presence greatly expands the size of the vocabulary
  - Will **often index separately as document "meta-data"**
    - Creation date, format, etc.

Tokenization: language issues

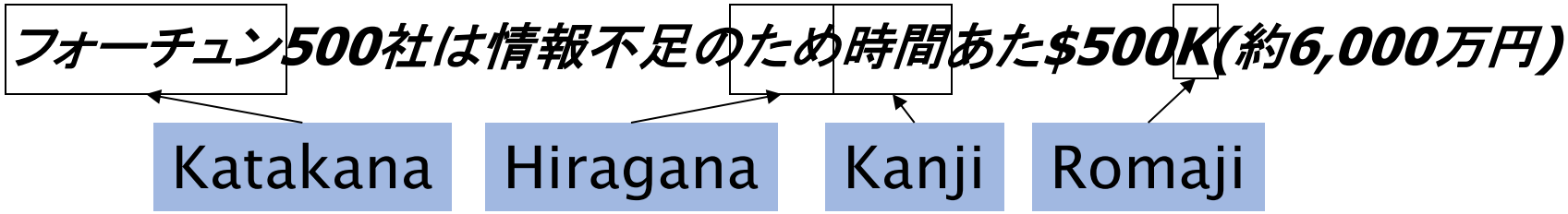
- French & Italian apostrophes
  - ***L'ensemble*** → one token or two?
    - ***L*** ? ***L'*** ? ***Le*** ?
    - We may want ***l' ensemble*** to match with ***un ensemble***

- German noun compounds are not segmented
  - ***Lebensversicherungsgesellschaftsangestellter***
  - 'life insurance company employee'
  - German retrieval systems benefit greatly from a **compound splitter** module

Tokenization: language issues

- Chinese and Japanese have **no spaces between words**:
  - 莎拉波娃现在居住在美国东南部的佛罗里达。
  - Not always guaranteed a unique tokenization

- Further complicated in Japanese, with multiple alphabets intermingled
  - Dates/amounts in multiple formats



Tokenization: language issues

- Arabic (or Hebrew) is basically written **right to left**, but with certain items like numbers written left to right
- Words are separated, but letter forms within a word form complex ligatures

  استقلت الجزائر في سنة 1962 بعد 132 عاما من الاحتلال الفرنسي.

- ← → ← → ← start
- *'Algeria achieved its independence in 1962 after 132 years of French occupation.'*
- Bidirectionality is not a problem if text is coded in **Unicode**.

Stop words

- With a stop list, you exclude from the dictionary entirely **the commonest words**. Intuition:
  - They have little semantic content: *the, a, and, to, be*
  - There are a lot of them: ~30% of postings for top 30 words
  - Stop word **elimination used to be standard in IR systems.**

- But the trend is away from doing this:
  - Good compression techniques means the space for including stopwords in a system is very small
  - Good query optimization techniques mean you **pay little** at query time for including stop words.
  - You need them for:
    - Phrase queries: “King of Denmark”
    - Various song/books titles, etc.: “Let it be”, “To be or not to be”
    - “Relational” queries: “flights to London”vrs “flight from London”

Normalization to terms

- We need to "normalize" words in indexed text as well as query words into the same form
  - We want that ***U.S.A.*** match ***USA***

- Result is terms: a term is a (normalized) word type, which is a **single entry** in our IR system dictionary

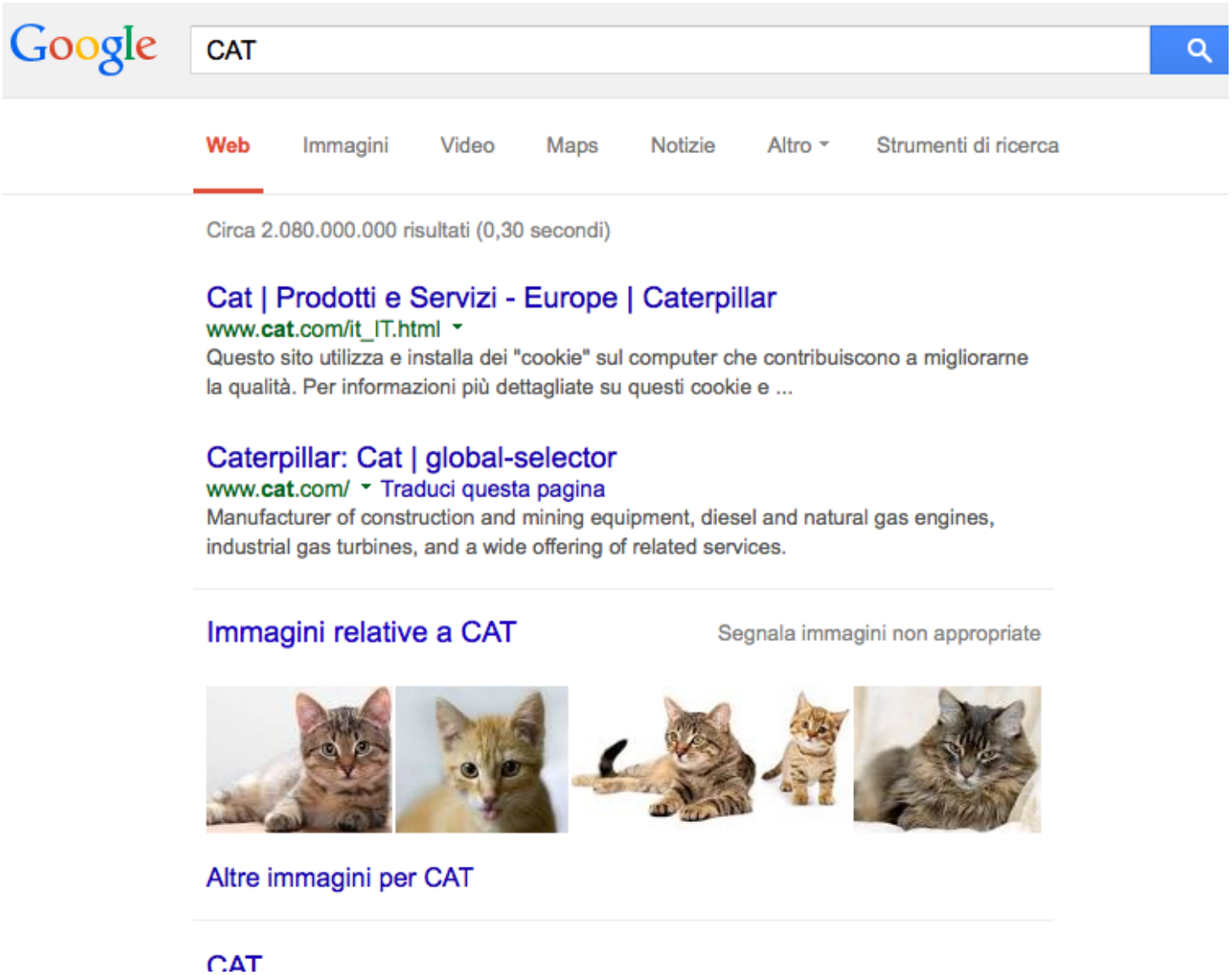
- We most commonly implicitly define **equivalence classes** of terms by, e.g.,
  - deleting periods to form a term
    - ***U.S.A., USA*** → ***USA***
  - deleting hyphens to form a term
    - ***anti-discriminatory, antidiscriminatory*** → ***antidiscriminatory***
  - Synonyms (this is rather more complex..)
    - car , automobile

Case folding

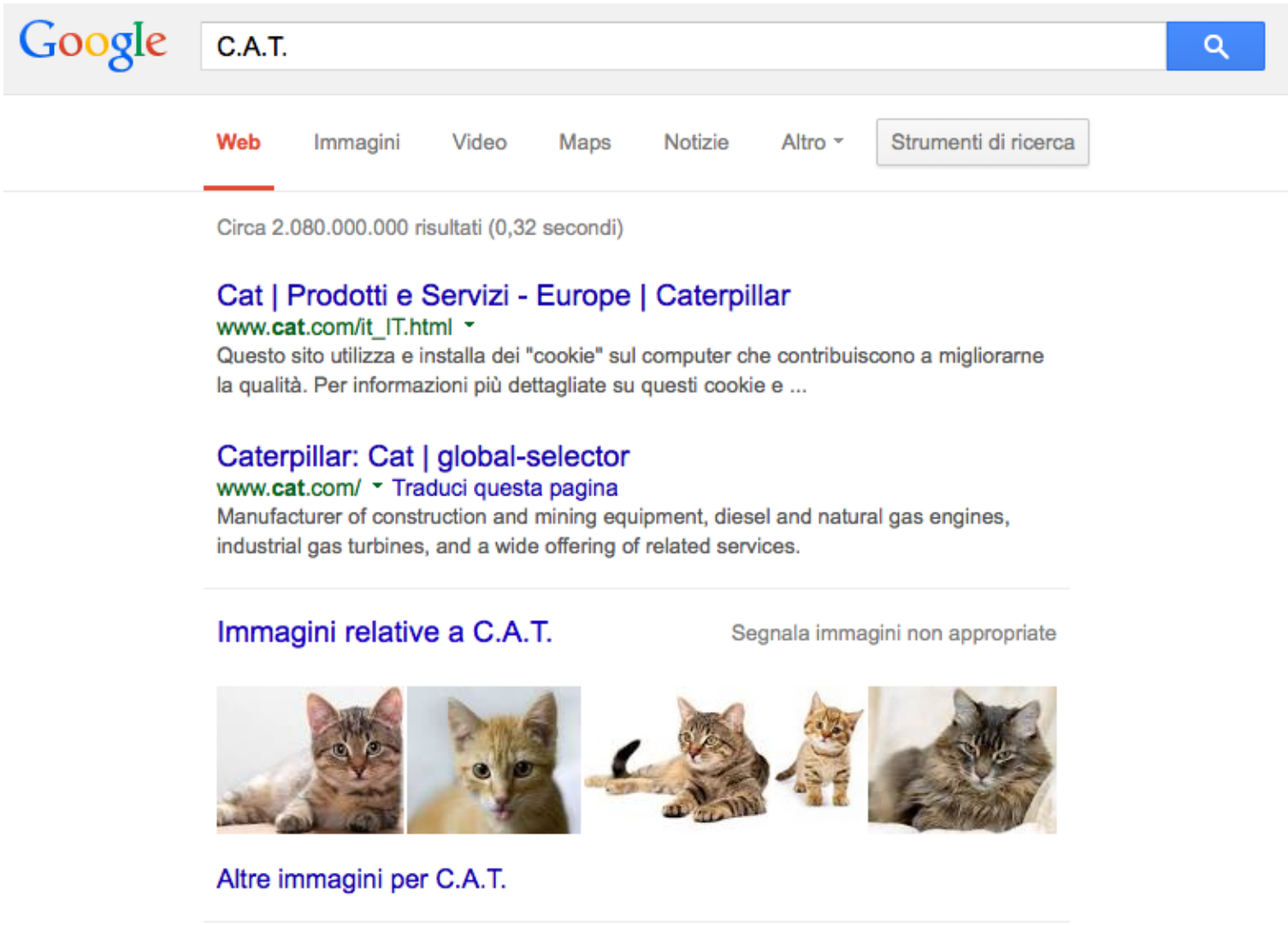
- Reduce all letters to lower case
  - exception: upper case in mid-sentence
    - e.g., ***General Motors***
    - ***Fed*** vs. ***fed***
    - ***MIT*** vs. mit
  - Often **best to lower case everything**, since users will use lowercase regardless of 'correct' capitalization…

- This may cause different senses to be merged.. Often the most relevant is simply the most frequent on the WEB, rather than the most intuitive

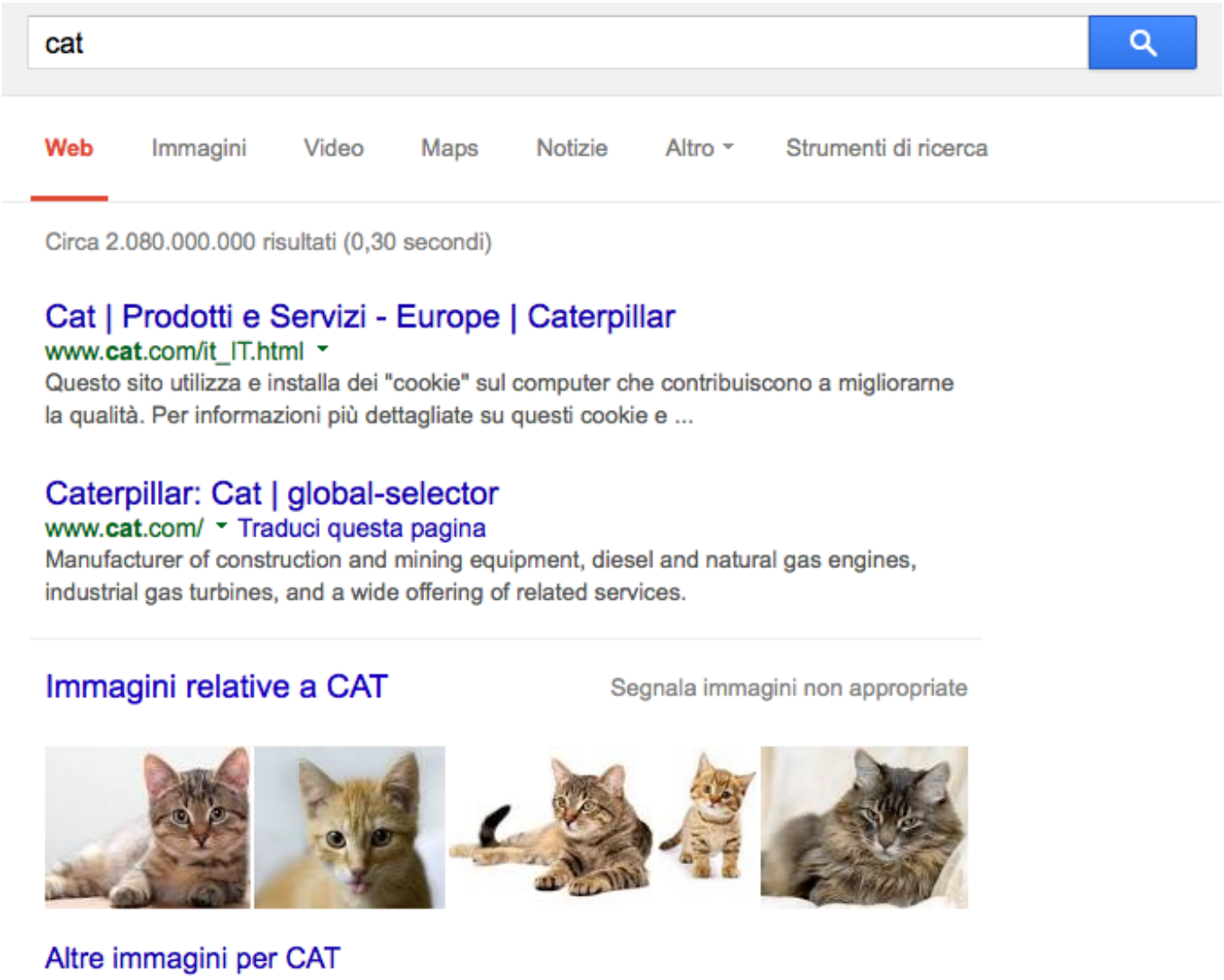
Normalization Example



Normalization Example



Normalization Example



Synonyms

- Do we handle synonyms and homonyms?
  - E.g., by hand-constructed equivalence classes
    - ***car*** = ***automobile*** ***color*** = ***colour***
  - We can rewrite to form equivalence-class terms
    - When the document contains ***automobile***, index it under ***car-automobile*** (and vice-versa)
  - Or we can expand a query
    - When the query contains ***automobile***, look under ***car*** as well

- What about spelling mistakes?
  - One approach is **Soundex**, a phonetic algorithm to encode homophones to the same representation so that they can be matched despite minor differences in spelling
  - Google → Googol

Stemming/Lemmatization

- Reduce inflectional/variant forms to base form
- E.g.,
  - *am, are, is* → *be*
  - *car, cars, car's, cars'* → *car*
- *the boy's cars are different colors* → *the boy car be different color*
  - Lemmatization implies doing "proper" reduction to dictionary form (the lemma).

Stemming

- Reduce terms to their "roots" before indexing
- "Stemming" suggest crude affix chopping
  - language dependent
  - e.g., ***automate(s), automatic, automation*** all reduced to ***automat***.

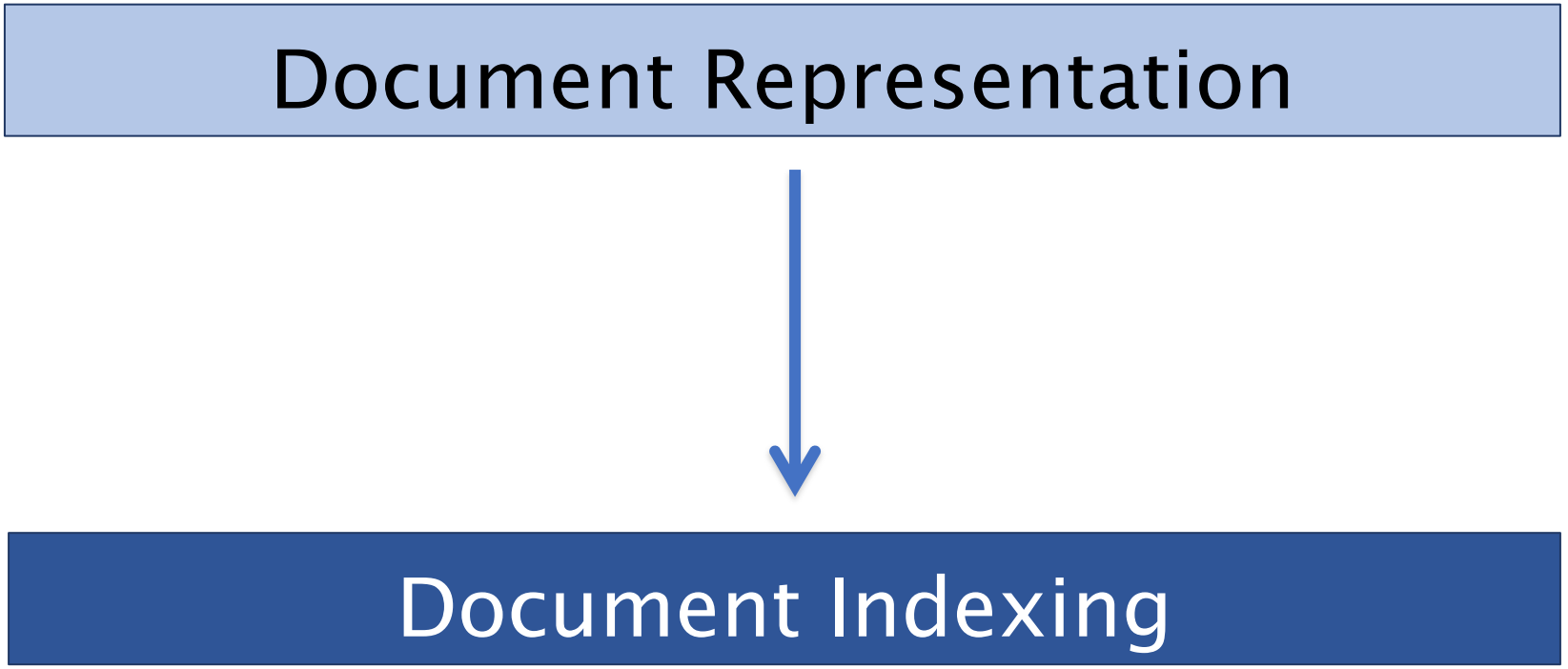
for example compressed and compression are both accepted as equivalent to compress.

for exampl compress and compress ar both accept as equival to compress

Deep Analysis

- Has to do with more detailed Natural Language Processing algorithms
- E.g. semantic disambiguation, phrase indexing (board of directors), named entities (President Obama= Barak Obama) etc.
- Standard search engines increasingly use deeper techniques (e.g. Google's **Knowledge Graph** http://www.google.com/insidesearch/features/search/knowledge.html)
- More (on deep NLP techniques) in NLP course (not here)!

Pipeline

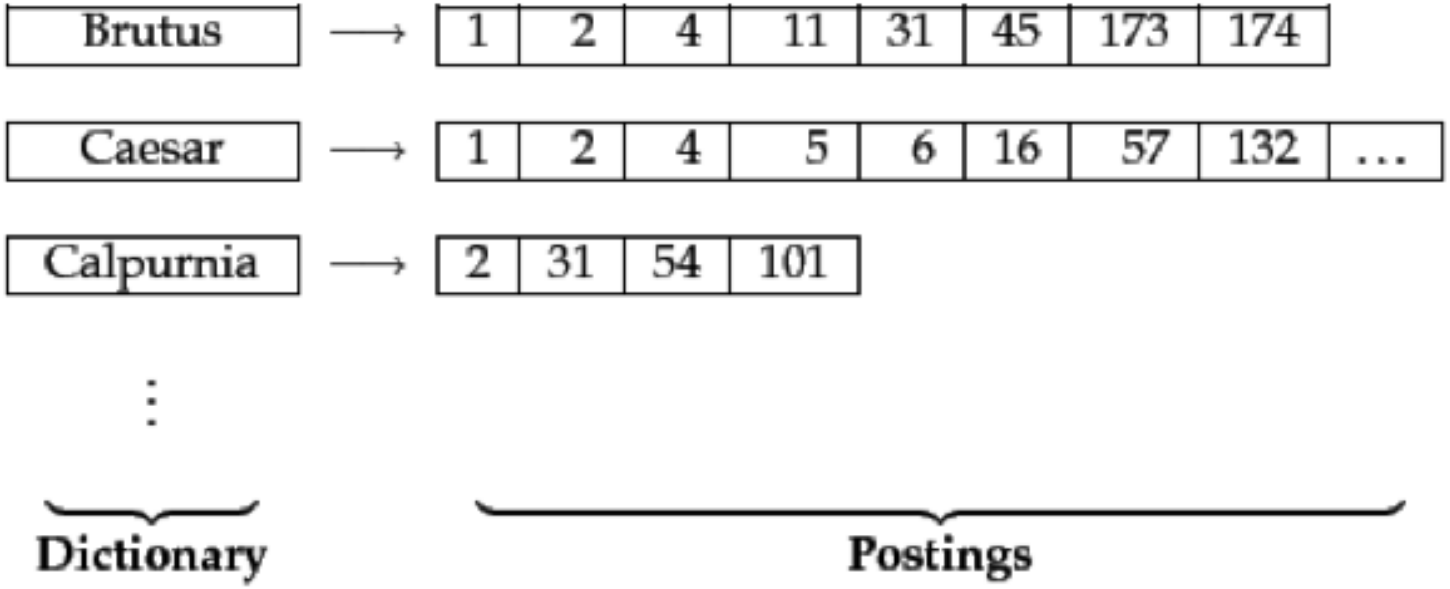


Why indexing

- The purpose of storing an index is to **optimize speed** and performance in finding **relevant documents** for a search query.
- **Without** an index, the search engine would scan **every** document in the corpus, which would **require** considerable time and computing power.
- For example, while an index of 10,000 documents can be **queried** within **milliseconds**, a sequential **scan** of every word in 10,000 large documents could take **hours**.

Inverted index

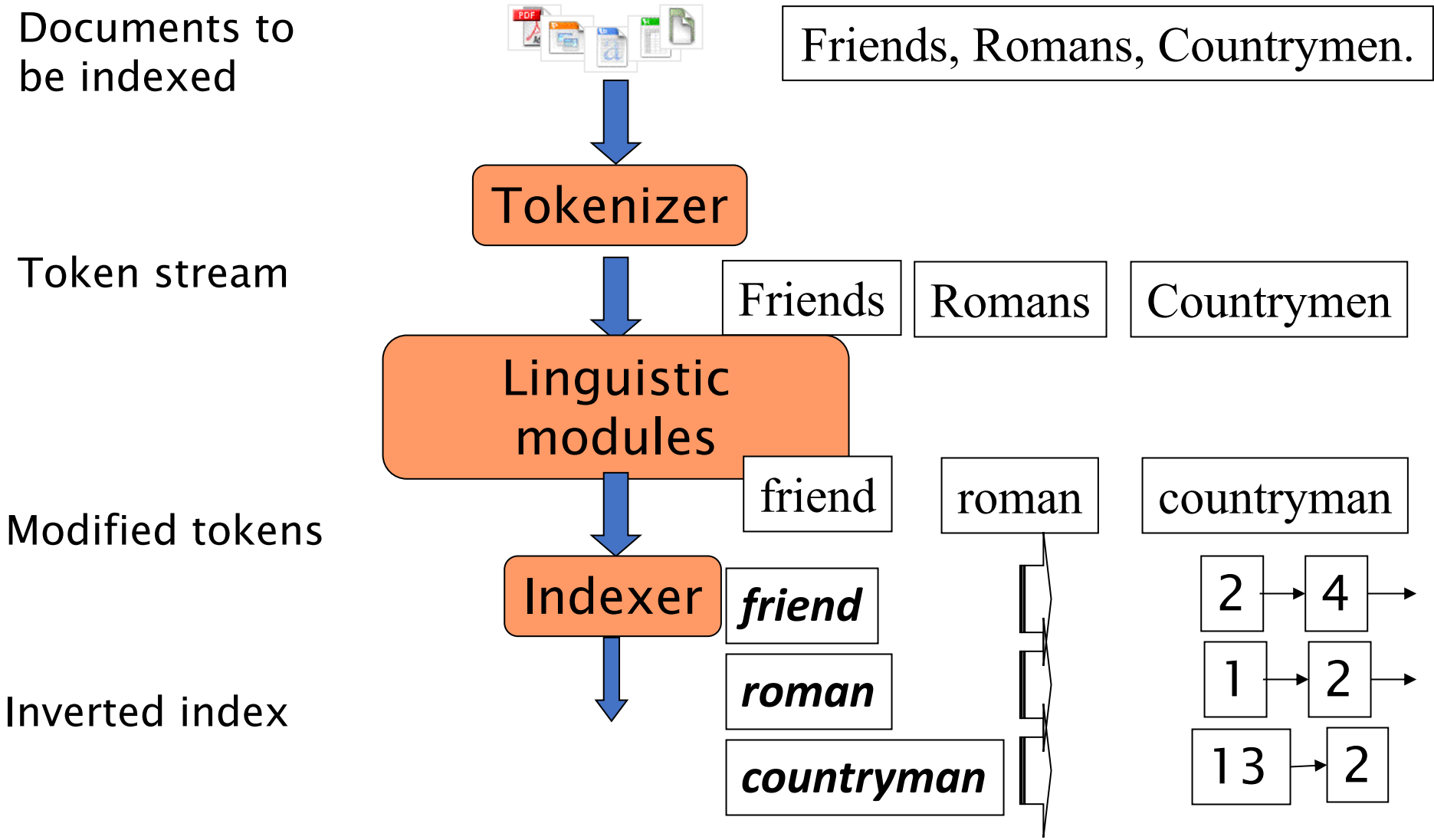
For each term, we have a list that records which documents the term occurs in.
The list is called **posting list.**



What happens if the word ***Caesar*** is added to document 14?

We need **variable-size postings** lists

Inverted index construction



Indexer steps: Token sequence

- Sequence of (Modified token, Document ID) pairs.

Doc 1

I did enact Julius **Caesar** I was killed i' the Capitol; Brutus killed me.

Doc 2

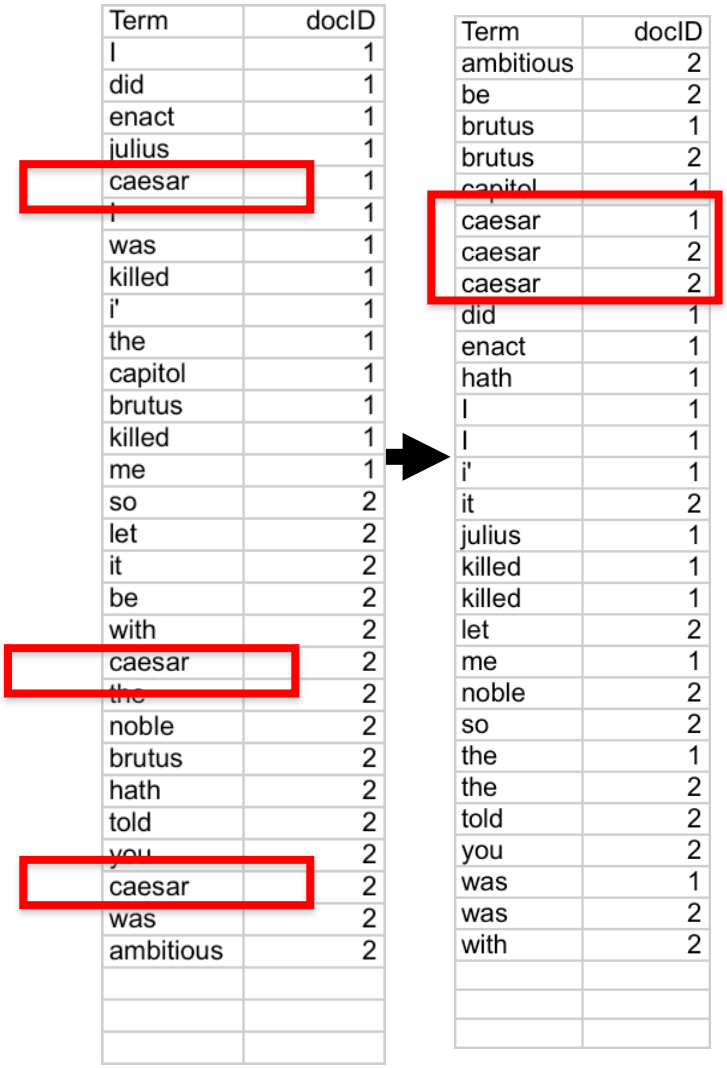
So let it be with **Caesar**. The noble Brutus hath told you **Caesar** was ambitious

Term	docID
I	1
did	1
enact	1
julius	1
caesar	1
I	1
was	1
killed	1
i'	1
the	1
capitol	1
brutus	1
killed	1
me	1
so	2
let	2
it	2
be	2
with	2
caesar	2
the	2
noble	2
brutus	2
hath	2
told	2
you	2
caesar	2
was	2
ambitious	2

Indexer steps: Sort

- Sort by terms
  - And then "docID"

Core indexing step

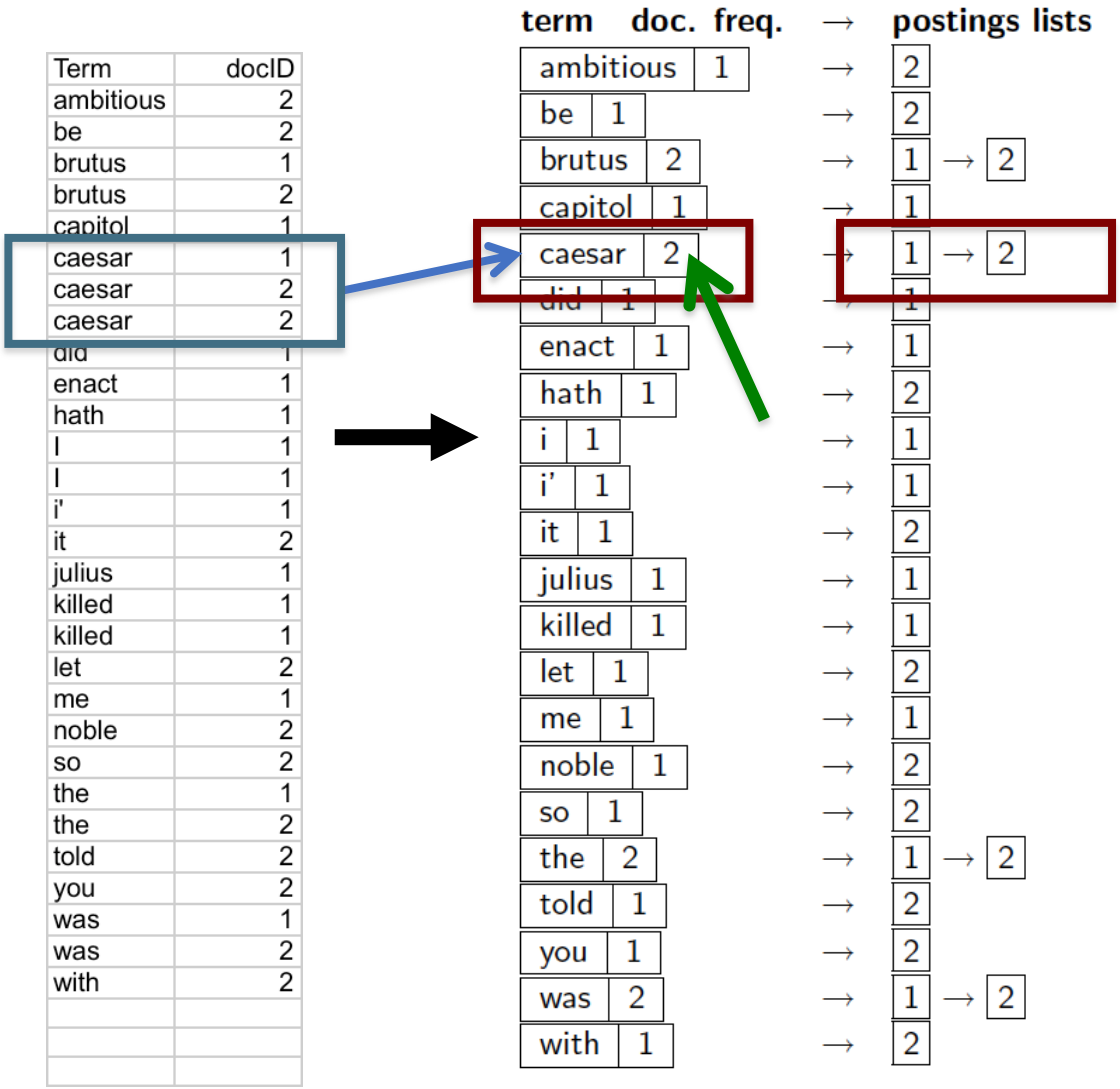


Term	docID
I	1
did	1
enact	1
julius	1
caesar	1
I	1
was	1
killed	1
i'	1
the	1
capitol	1
brutus	1
killed	1
me	1
so	2
let	2
it	2
be	2
with	2
caesar	2
the	2
noble	2
brutus	2
hath	2
told	2
you	2
caesar	2
was	2
ambitious	2

Term	docID
ambitious	2
be	2
brutus	1
brutus	2
capitol	1
caesar	1
caesar	2
caesar	2
did	1
enact	1
hath	1
I	1
I	1
i'	1
it	2
julius	1
killed	1
killed	1
let	2
me	1
noble	2
so	2
the	1
the	2
told	2
you	2
was	1
was	2
with	2

Indexer steps: Dictionary & Postings

- Multiple term entries in a single document are merged.
- Split into Dictionary and Postings
- Doc. **frequency information** is added.



Term	docID
ambitious	2
be	2
brutus	1
brutus	2
capitol	1
caesar	1
caesar	2
caesar	2
did	1
enact	1
hath	1
I	1
I	1
i'	1
it	2
julius	1
killed	1
killed	1
let	2
me	1
noble	2
so	2
the	1
the	2
told	2
you	2
was	1
was	2
with	2

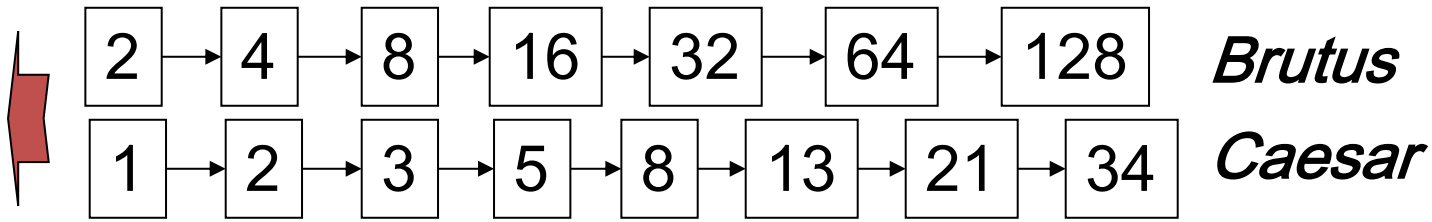
Query Processing

Query processing: AND

- Consider processing the query:

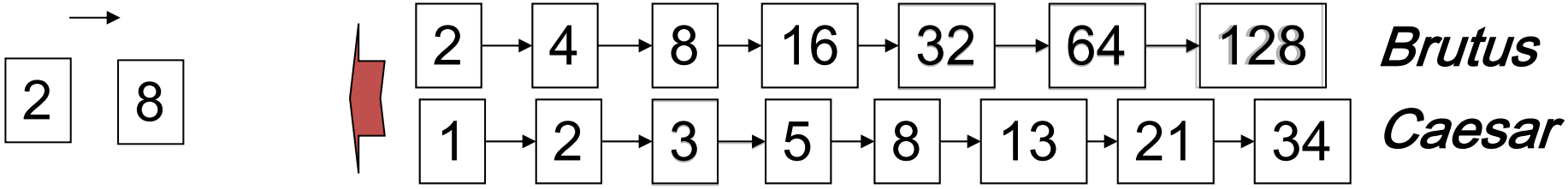
  Brutus *AND* ***Caesar***

  - Locate ***Brutus*** in the Dictionary;
    - Retrieve its postings (e.g. pointers to documents including Brutus).
  - Locate *Caesar* in the Dictionary;
    - Retrieve its postings.
  - "Merge" the two postings:



The “merge” operation

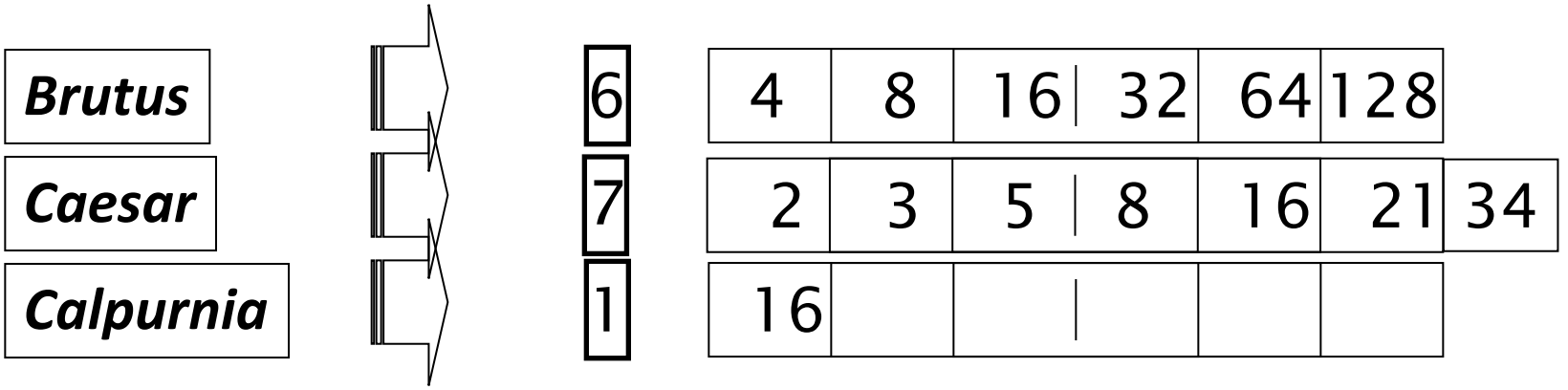
- Walk through the two postings simultaneously from right to left, in time linear in the total number of postings entries



If list lengths are *x* and *y*, merge takes O(*x*+*y*) operations.
Crucial: postings must be sorted by docID.

Optimization of index search

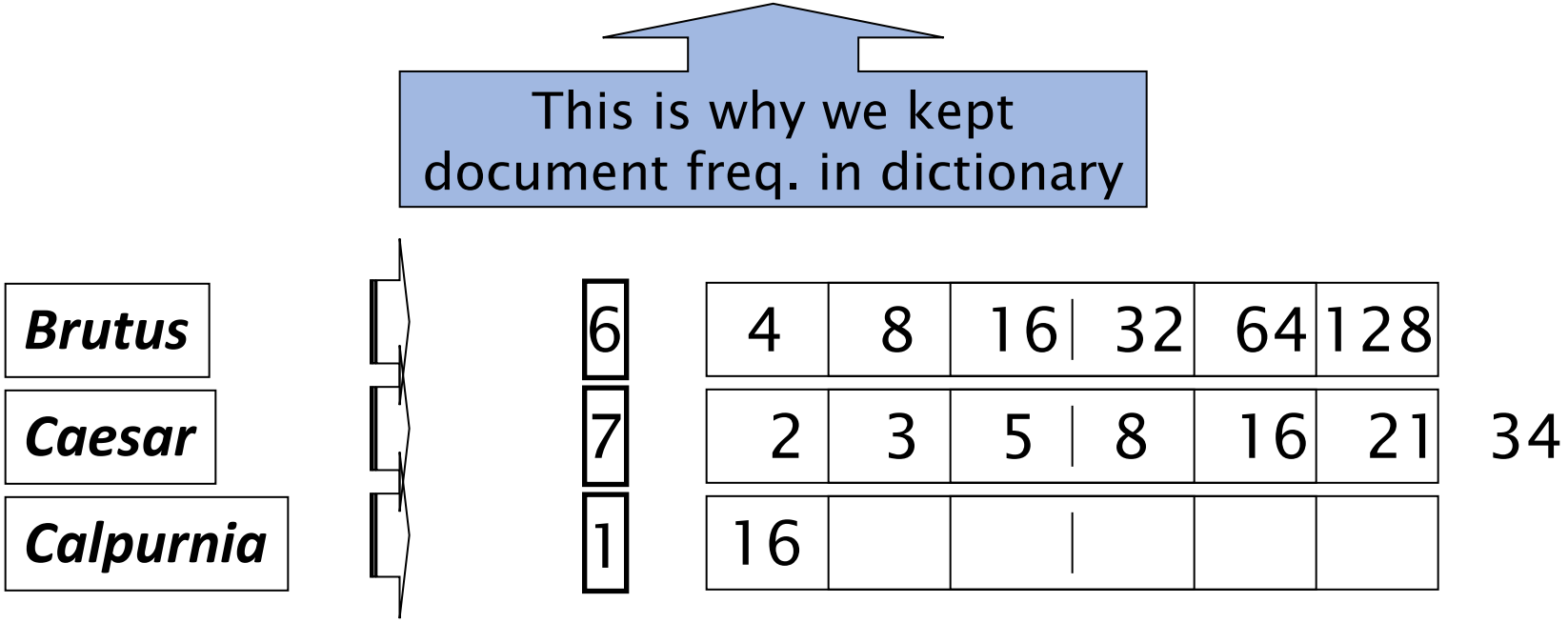
- What is the best order of words for query processing?
- Consider a query that is an *AND* of *n* terms.
- For each of the *n* terms, get its postings, then *AND* them together.



Query: ***Brutus*** *AND* ***Calpurnia*** *AND* ***Caesar***

Query optimization example

- Process words in order of increasing freq:
  - *start with smallest set, then keep cutting further*.



Execute the query as (***Calpurnia*** *AND* ***Brutus)*** *AND* ***Caesar***.

Phrase queries

- Want to be able to answer queries such as "**red brick house**"– as a phrase

- red AND brick AND house match phrases as:
  "*red house near the brick factory*"

  which is not what we are searching for
  - The concept of phrase queries has proven easily understood by users; one of the few "advanced search" ideas that works
  - **About 10% of web queries are phrase queries**.

- For this, it no longer **suffices** to store only

  <*term* : *docs*> entries

Bi-word indexes

- Index **every consecutive pair** of terms in the text as a phrase
- For example the text "Friends, Romans, Countrymen" would generate the biwords
  - ***friends romans***
  - ***romans countrymen***
- Each of these **biwords** is now a *dictionary term*
- Two-word phrase query-processing is now immediate.

Longer phrase queries

- Longer phrases are processed using bi-words:
- ***stanford university palo alto*** can be broken into the Boolean query on biwords:

stanford university *AND* ***university palo*** *AND* ***palo alto***

Drawbacks of the biword indexes

- Index **blowup** due to bigger dictionary
  - Infeasible for more than biwords, big even for them

- Biword indexes are not the standard solution (for all biwords) but can **be part of a compound strategy;**

Positional indexes

- Positional indexes are a more efficient alternative to biword indexes.
- In the postings, store, for each ***term*** the position(s) in which tokens of it appear:

<***term***, number of docs containing ***term***;
doc1: position1, position2 … ;
doc2: position1, position2 … ;
etc.>

to, 993427:
⟨ 1, 6: ⟨7, 18, 33, 72, 86, 231⟩;
2, 5: ⟨1, 17, 74, 222, 255⟩;
4, 5: ⟨8, 16, 190, 429, 433⟩;
5, 2: ⟨363, 367⟩;
7, 3: ⟨13, 23, 191⟩; . . . ⟩

be, 178239:
⟨ 1, 2: ⟨17, 25⟩;
4, 5: ⟨17, 191, 291, 430, 434⟩;
5, 3: ⟨14, 19, 101⟩; . . . ⟩

Positional index example

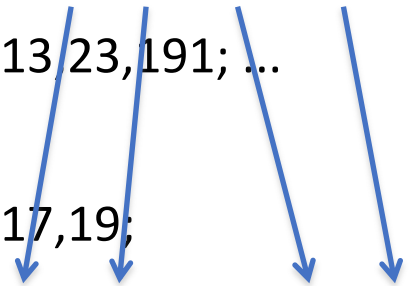
<*be*: 993427;
1: 7, 18, 33, 72, 86, 231;
2: 3, 149;
4: 17, 191, 291, 430, 434;
5: 363, 367, …>

Which of docs 1,2,4,5 could contain "***to be or not to be***"?

- For phrase queries, we use a merge algorithm recursively at the document level
- But we now need to deal with more than just equality

Processing a phrase query

- Extract inverted index entries for each distinct term: ***to, be, or, not.***
- Merge their *doc:position* lists to enumerate all positions with "***to be or not to be***".
  - ***to****:*
    - *2*:1,17,74,222,551;
    - *4*:8,16,190,429,433;
    - *7*:13,23,191; ...
  - ***be****:*
    - *1*:17,19;
    - *4*:17,191,291,430,434; *5*:14,19,101; ...



- Use N^X operator (e.g. N^1 if $pos(w_2) - pos(w_1) = 1$)

Proximity search

- We just saw how to use a positional index for **phrase searches**.
- We can also use it for **proximity search**.
- For example: *employment* /4 *place*
  Find all documents that contain EMPLOYMENT and PLACE within 4 words of each other.
- *"**Employment** agencies that **place** healthcare workers are seeing growth"* **is a hit**.
- *"**Employment** agencies that have learned to adapt now **place** healthcare workers"* is **not a hit**.

Positional index drawbacks

- Positional index expands postings storage ***substantially***
  - some rough rules of thumb are to expect a positional index to be 2 to 4 times as large as a non-positional index
- Positional index is now **standardly used** because of the power and usefulness of phrase and proximity queries

Combined scheme

- **Biword** indexes and **positional** indexes can be profitably **combined**.
- Many biwords (named entities) are extremely **frequent**:
  Michael Jackson, Barrack Obama etc.
- For these biwords, **increased speed** compared to positional postings intersection is substantial.
- Combination scheme:
  - Include frequent biwords as vocabulary terms in the index.
  - Do all other phrases by positional intersection.