Rome | March 22 - 23, 2019

{codemotion}

Everything You Always Wanted to Know About Search

(But Were Afraid to Ask)

Antonio Mallia

About me

me@antoniomallia.it

@antonio_mallia

amallia

in/antoniomallia

www.antoniomallia.it

New York City

I am currently pursuing a Ph.D. with a focus on advancing **efficiency in Information Retrieval for large-scale systems**. I am part of the Web Exploration and Search Technology Lab at **New York University** where I explore new techniques and architectures for web search and related problems.

Agenda

- Learn about Information Retrieval (IR)
- How a search engine works
- How crawling works
- Indexing of documents
- Query processing
- Ranking
- Search Quality Evaluation
- Image search
- Alerts

Not All About Search

Information retrieval (IR) is finding material (usually documents) of an unstructured nature (usually text) that satisfies an information need from within large collections.

Nowadays, Information Retrieval goes beyond building just search engines!

Question Answering

Google

when is codemotion rome?

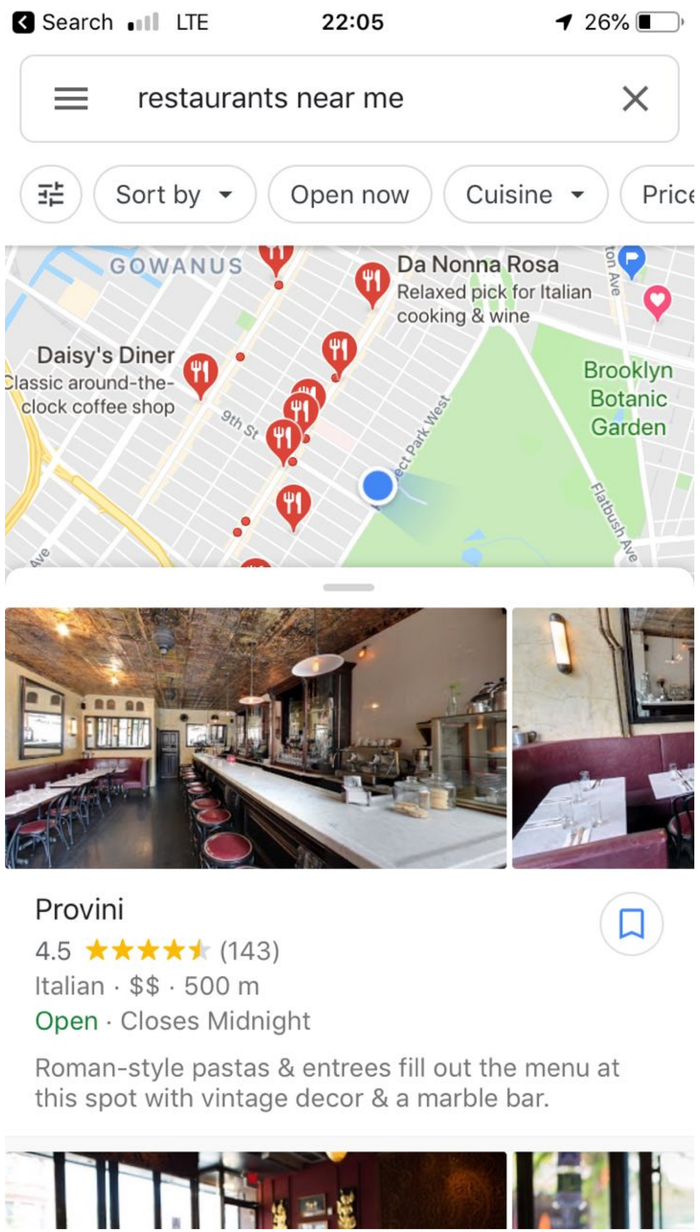
All News Images Maps Shopping More Settings Tools

About 90,800 results (0.54 seconds)

2019 Codemotion - Rome / Dates

Fri, Mar 22, 2019 – Sat, Mar 23, 2019

Geo and Trajectory Search



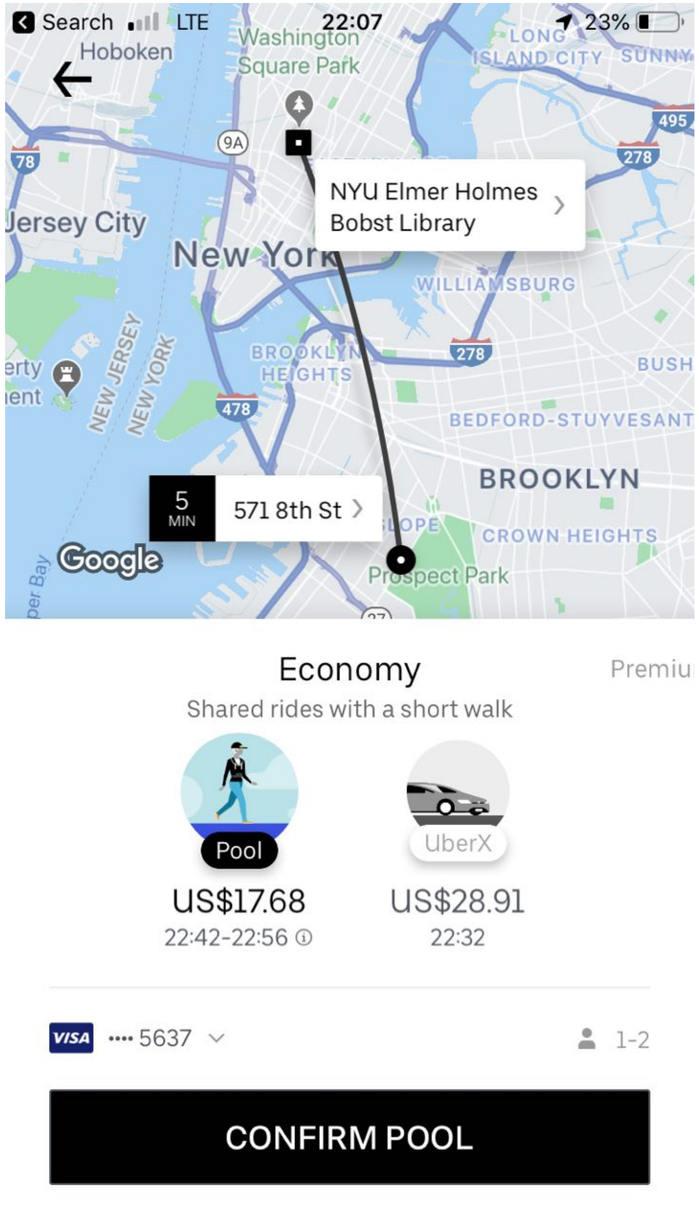
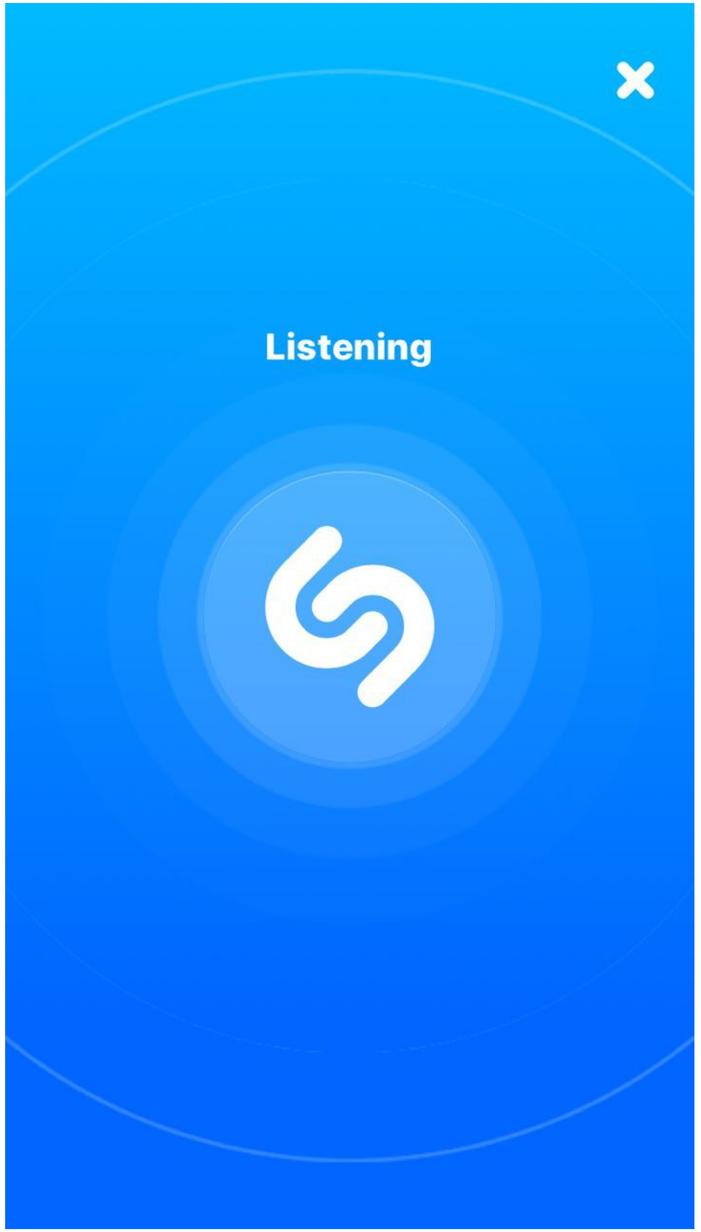
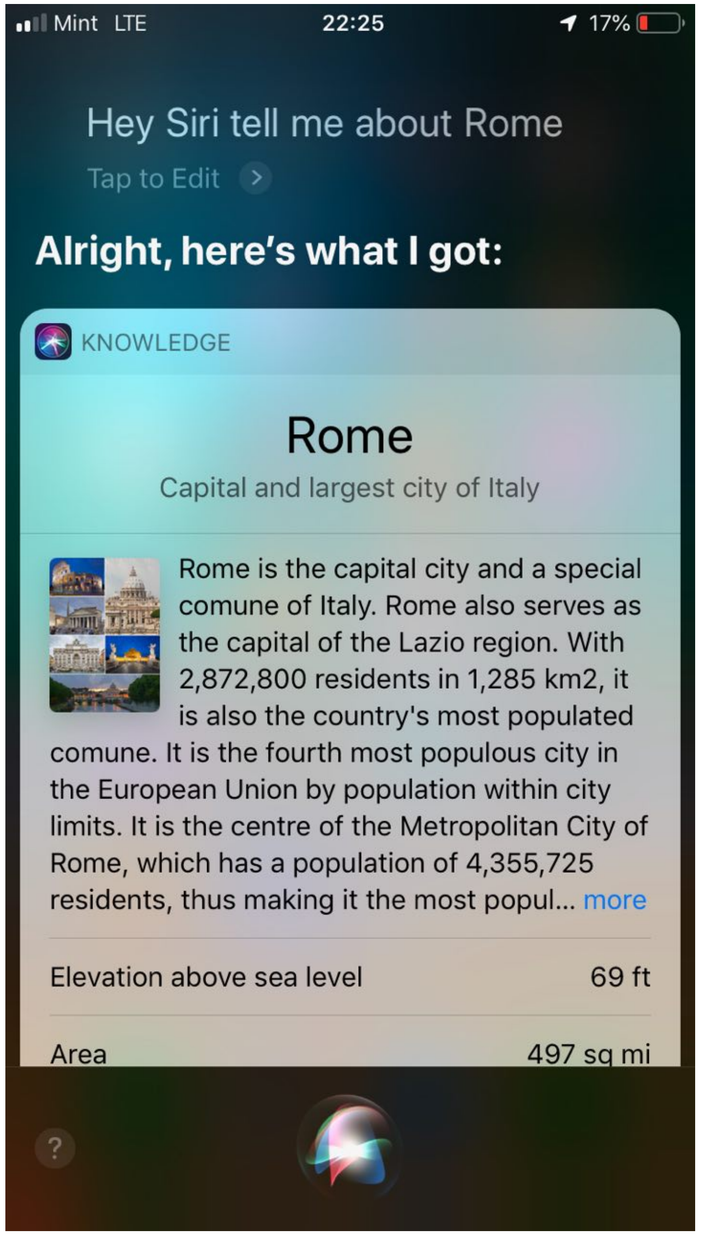
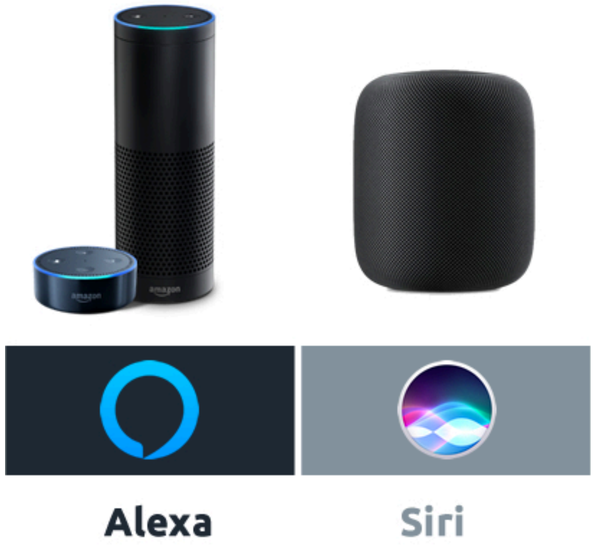


Image and Music Retrieval



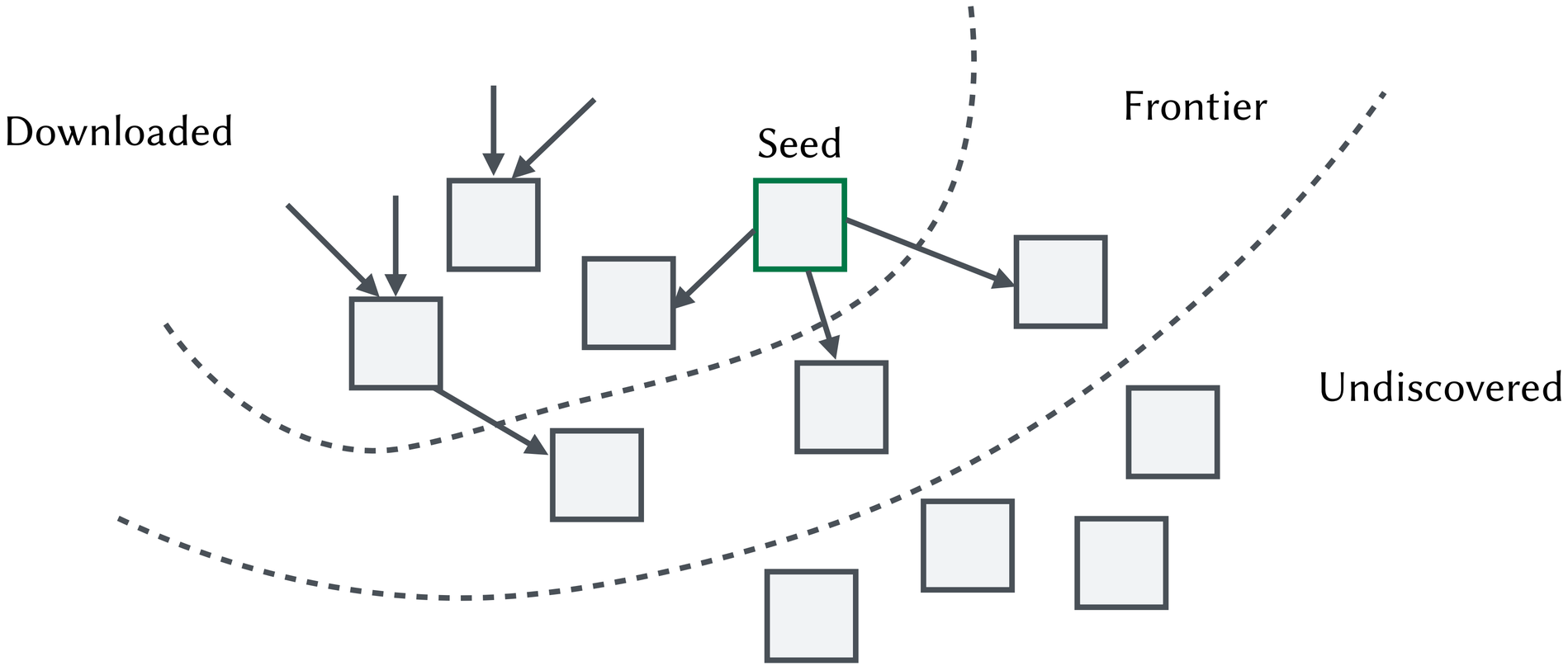
Conversational search





Crawling

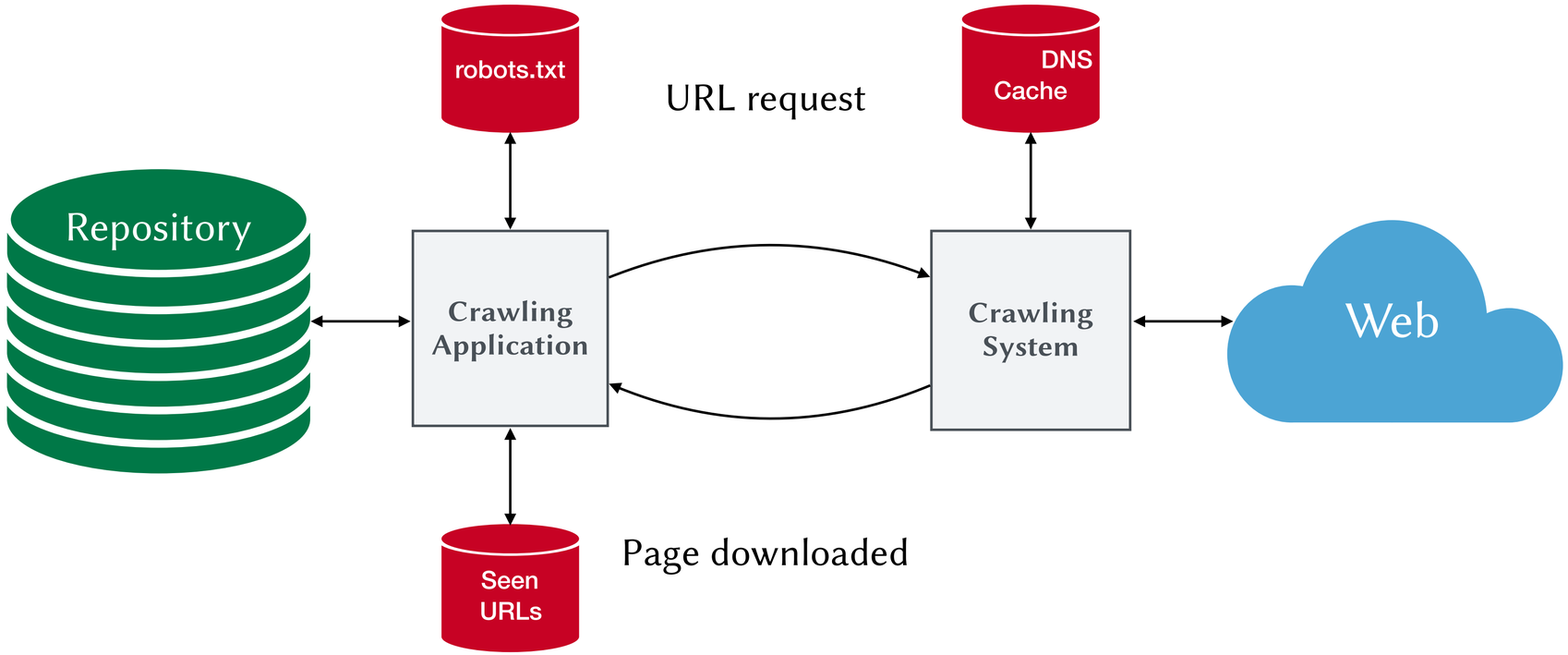
Crawling



Basic idea:

- start at a set of known URLs - explore the web in “concentric circles” around these URLs
- rely on the hyperlinks between web pages to discover new URLs

Crawling



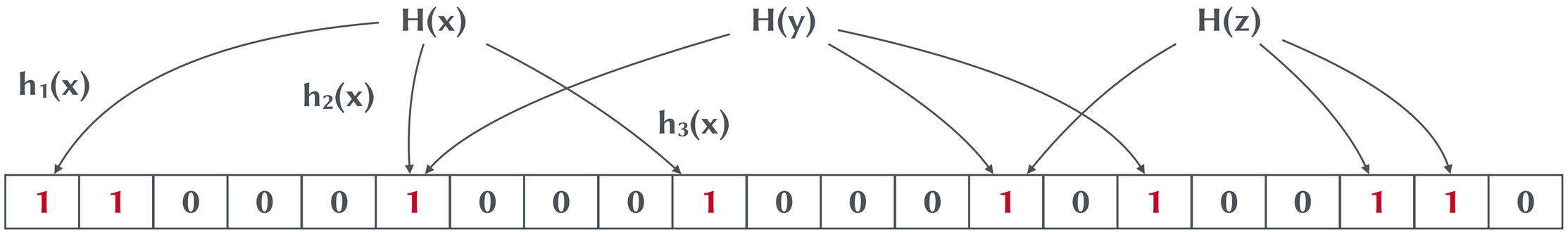
Crawling System: fetches pages, design to be fast and simple
Crawling Application: decides what to crawl next according to a crawling strategy

How to keep track of pages already crawled?
Bloom filters

Large engines crawl billions of pages per day, so they must be efficient!
Distributed crawling:

- dynamic: central coordinator dynamically assigns URLs to crawlers
- static: Web is statically partitioned and assigned to crawlers

Bloom Filter



A **Bloom filter** is a probabilistic data-structure with two operations:

- **Insert**(x) inserts an item into the BF
- **Exist**(x) checks if x was previously inserted in the BF

A Bloom filter can have false positives, but no false negatives.

Bloom filters achieve good trade-off between space and fpr.

Crawling

How to crawl?
- best pages first
- avoid duplicate
- robots.txt, minimize load
- spider traps

How often to crawl?
- freshness
- estimating change probabilities based on observation

How much to crawl?
- coverage. How much to cover?
- how much do competitors have
- some pages are more important than others

Open Source Crawler:
- StormCrawler - http://stormcrawler.net/
- Apache Nutch - https://nutch.apache.org/
- Heritrix - https://github.com/internetarchive/heritrix3
- BUbiNG - https://github.com/LAW-Unimi/BUbiNG
- wget

WARC - Web ARChive

Archive format which specifies a method for combining multiple digital resources into an aggregate archive file together with related information.

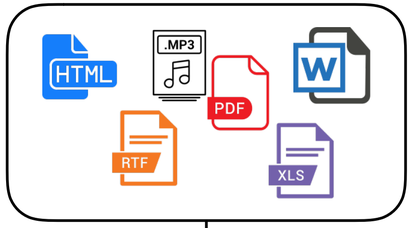
WARC is a standard way to store the raw crawl data.

```
WARC/1.0
WARC-Type: response
WARC-Date: 2013-12-04T16:47:32Z
WARC-Record-ID:
Content-Length: 73873
Content-Type: application/http; msgtype=response
WARC-Warcinfo-ID:
WARC-Concurrent-To:
WARC-IP-Address: 23.0.160.82
WARC-Target-URI: http://102jamzorlando.cbslocal.com/tag/nba/page/2/
WARC-Payload-Digest: sha1:FXV2BZKHT6SQ4RZWNMIMP7KMFUNZMZFB
WARC-Block-Digest: sha1:GMYFZYSACNBEGHVP3YFQNOSTV5LPXNAU

HTTP/1.0 200 OK
Server: nginx
Content-Type: text/html; charset=UTF-8
Vary: Accept-Encoding
Vary: Cookie
X-hacker: If you're reading this, you should visit automattic.com/jobs and apply to join the fun, mention this header.
Content-Encoding: gzip
Date: Wed, 04 Dec 2013 16:47:32 GMT
Content-Length: 18953
Connection: close


...HTML Content...
```

You can read more at https://iipc.github.io/warc-specifications/

Indexing

Parsing



What format is it in?
What language is it in?

Parser Identification

- A **classification** problem.
- Language is needed by further steps.

Split text into tokens

Tokenization

- **Normalization of terms**: deleting hyphens and punctuation, i.e. **U.S.A.** and **USA**
- **Case folding**: reduce all letters to lower case i.e. *Rome* vs. *rome*

Remove common words

Stopping

- **Stopwords:** common words, i.e. the, a, and, to
- Stopwords carry little **semantic content**

Reduce terms to their roots

Stemming

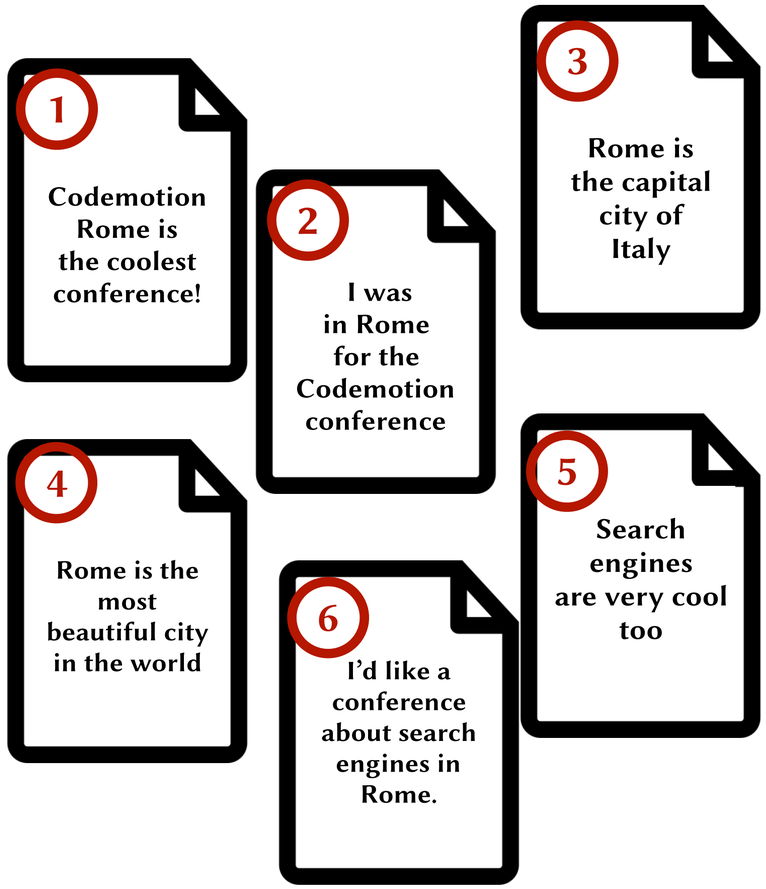
- Produce **morphological variants of a root**/base word, i.e. cool, cooler, coolest
- uses lists of related words (**dictionary-based**) or **algorithm** to determine related words

Synonyms and homonyms

Thesauri

- Rewrite to form **equivalence-class terms**
- Hand-construct equivalence classes, i.e. car = automobile

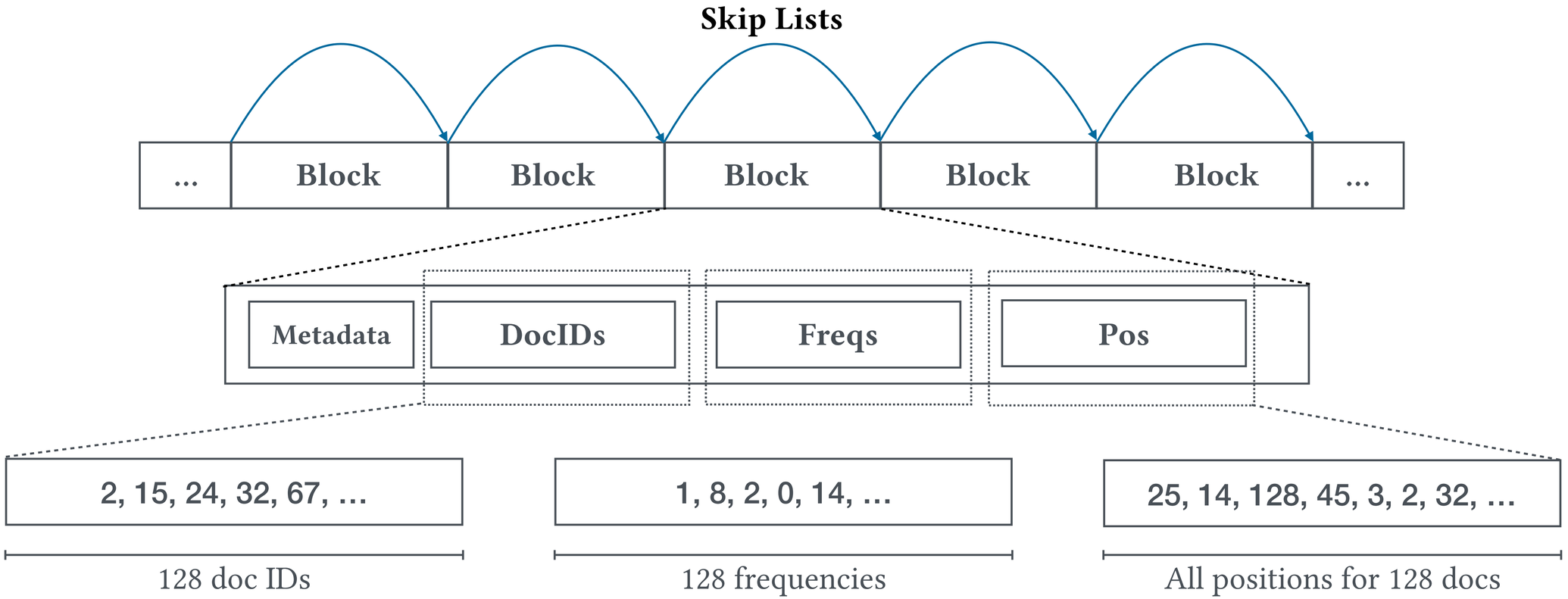
Indexing



T_1	codemotion	→	[1, 2]
T_2	cool	→	[1, 5]
T_3	conference	→	[1, 2, 6]
T_4	rome	→	[1, 2, 3, 4, 6]
T_5	capital	→	[3]
T_6	city	→	[3, 4]
T_7	italy	→	[3]
T_8	beautiful	→	[4]
T_9	world	→	[4]
T_{10}	search	→	[5, 6]
T_{11}	engine	→	[5, 6]

Terms ={codemotion, cool, conference, rome, capital, city, italy, beautiful, world, search, engine}

Inverted Index Layout



The list of the documents for a given term is called a **posting list**.

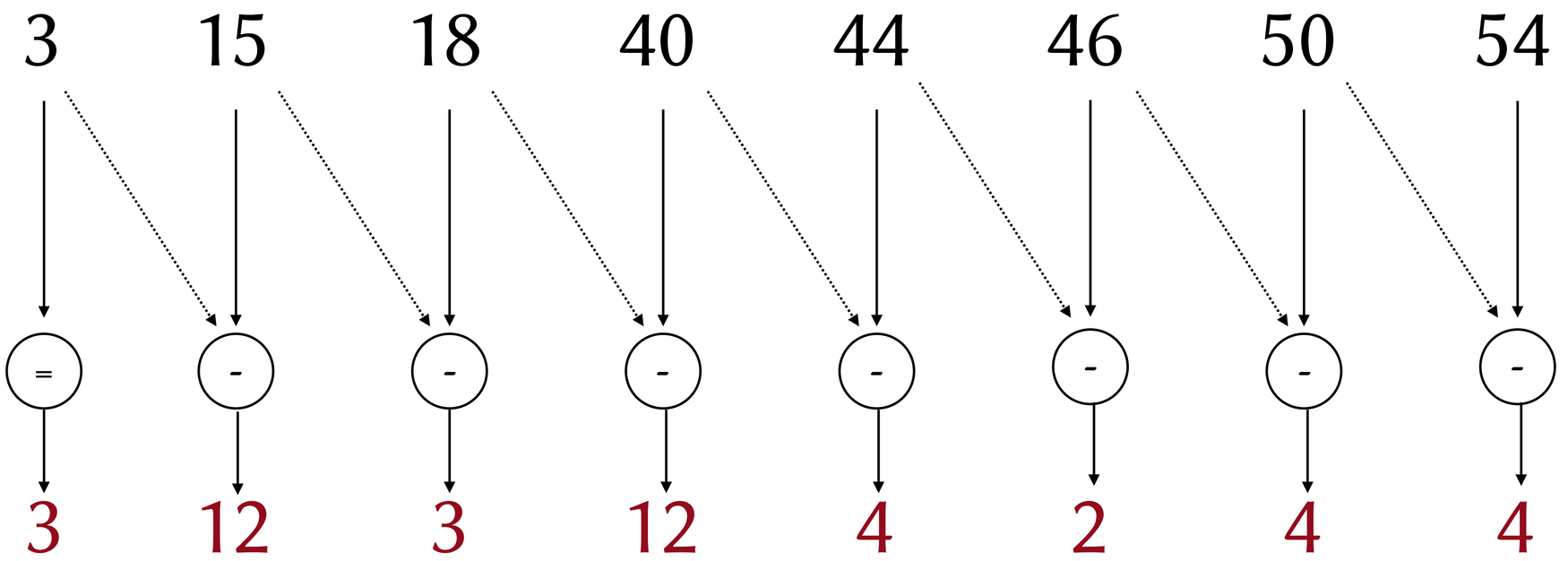
- Posting lists are usually divided in blocks of fixed size
- Each block can be individually encoded and decoded
- This allows to jump forward without uncompressing all entries

Index Maintenance

Three operations need to be supported:

- **Insert**: keep a buffer and re-build when big enough
- **Delete**: tombstone and remove when re-building
- **Update**: update the frequencies and remove if needed when re-building

Delta-Gap Encoding



Posting lists are sequences of integers

Document IDs can be large numbers, but differences are small

Documents IDs are typically stored in increasing order

Variable Byte

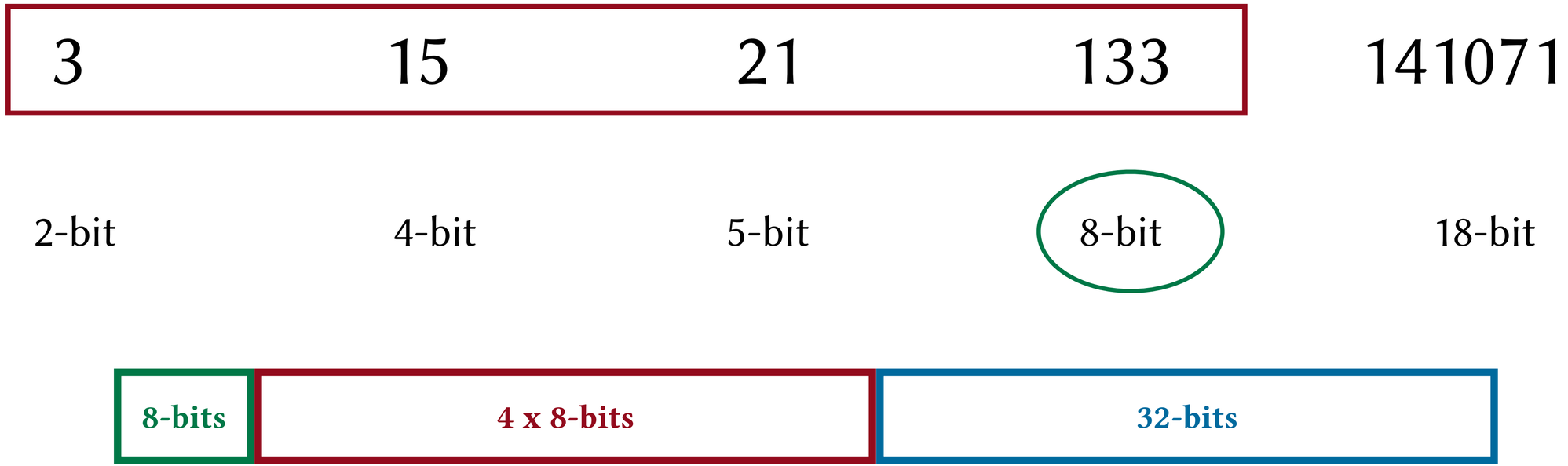
3	15	21	133	141071
8-bit	8-bit	8-bit	16-bit	24-bit
0|0000011	0|0001111	0|0010101	1|0000001	1|0001000
			0|0000101	1|1001110
				0|0001111

Small values need fewer bits to be represented than larger values. The majority of the values are small.

The last 7 bits of a byte are the **payload**, the first bit of the byte is a **continuation bit.**

Continuation bit is 1 for the last byte of the encoded value and to 0 otherwise. Keep reading until you find a continuation bit equal to 0.

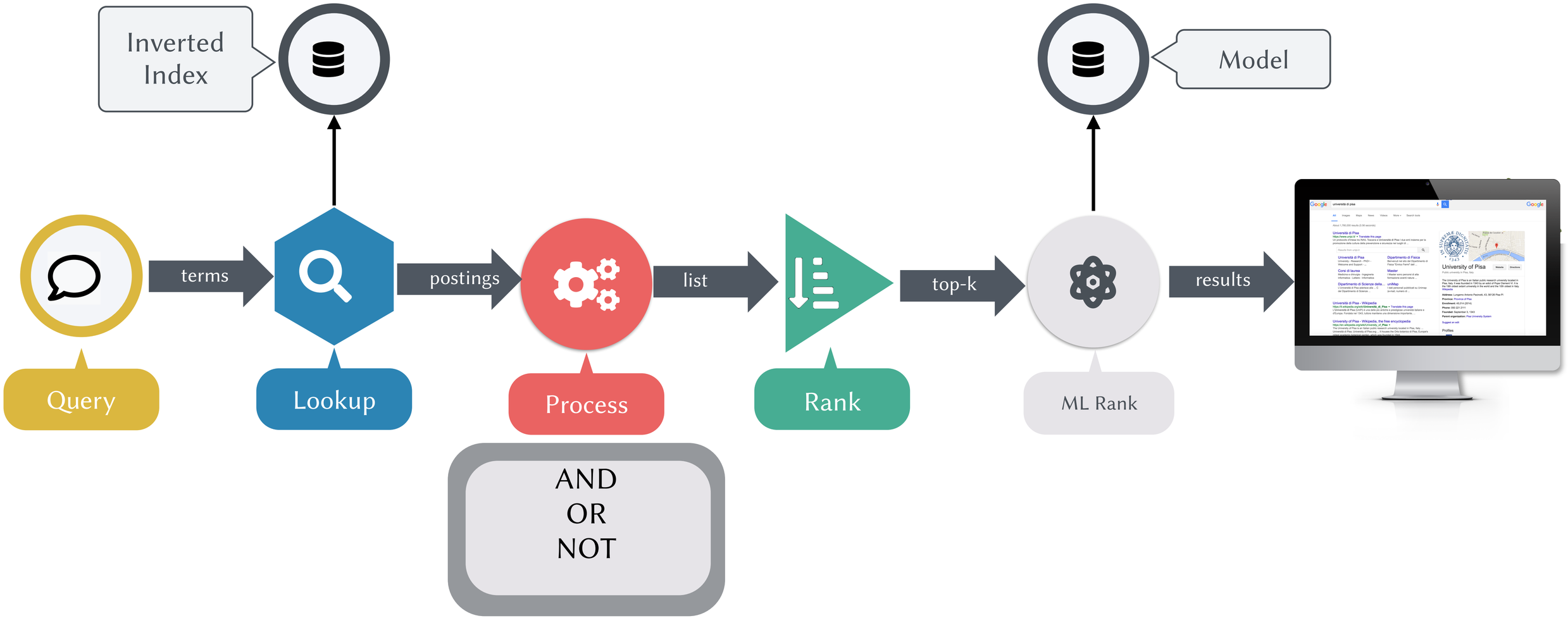
Patched Frame-of-reference (PFor)



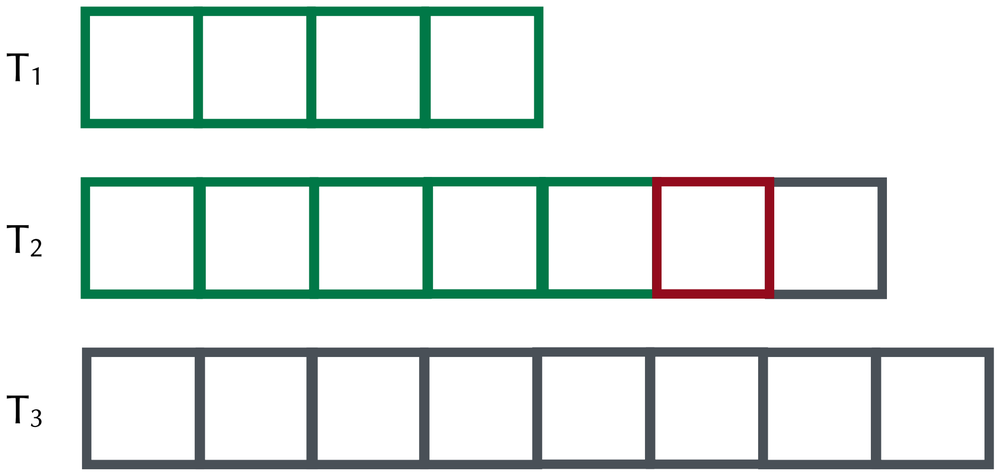
1. Find **B**, which indicates how many bits are needed to encode “most” of the elements in the list
2. Encode B using 8 bits (always enough to encode any value between 1 and 64)
3. Store the values that can be written using B bits
4. Store the **exceptions** at the end of the list using fix-length encoding

Query Processing

Query processing

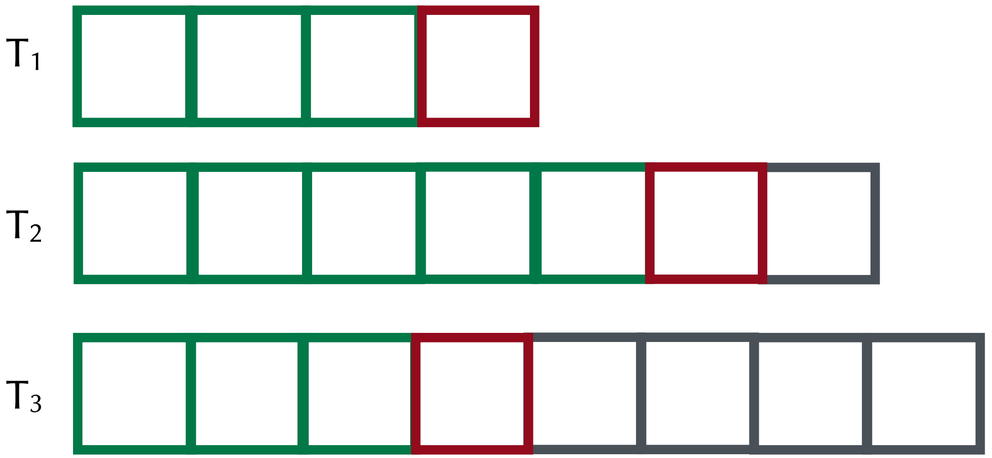


Term-at-a-time (TAAT) vs Document-at-a-time (DAAT)



Term-at-a-time

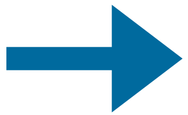
- Process one inverted list at a time, from shortest to longest list
- Accumulator is needed



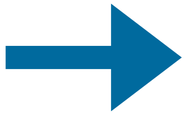
Document-at-a-time

- all inverted lists in the query are traversed simultaneously from left to right
- DAAT uses no extra space

Boolean Retrieval



$T_2 \cap T_3 \cap T_4 = [1, 5] \cap [1, 2, 6] \cap [1, 2, 3, 4, 6] = [1]$

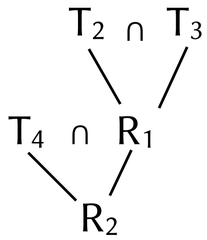


Query rewriting: different orders evaluate to the same result, but could not be equally efficient

Operators:

- AND
- OR
- NOT

T_2	cool	→	[1, 5]
T_3	conference	→	[1, 2, 6]
T_4	rome	→	[1, 2, 3, 4, 6]



Scoring Functions

Scoring function assigns **score** to each document with respect to a given query.

It allows to return only the **top-k** documents with highest scores, which enables several optimizations.

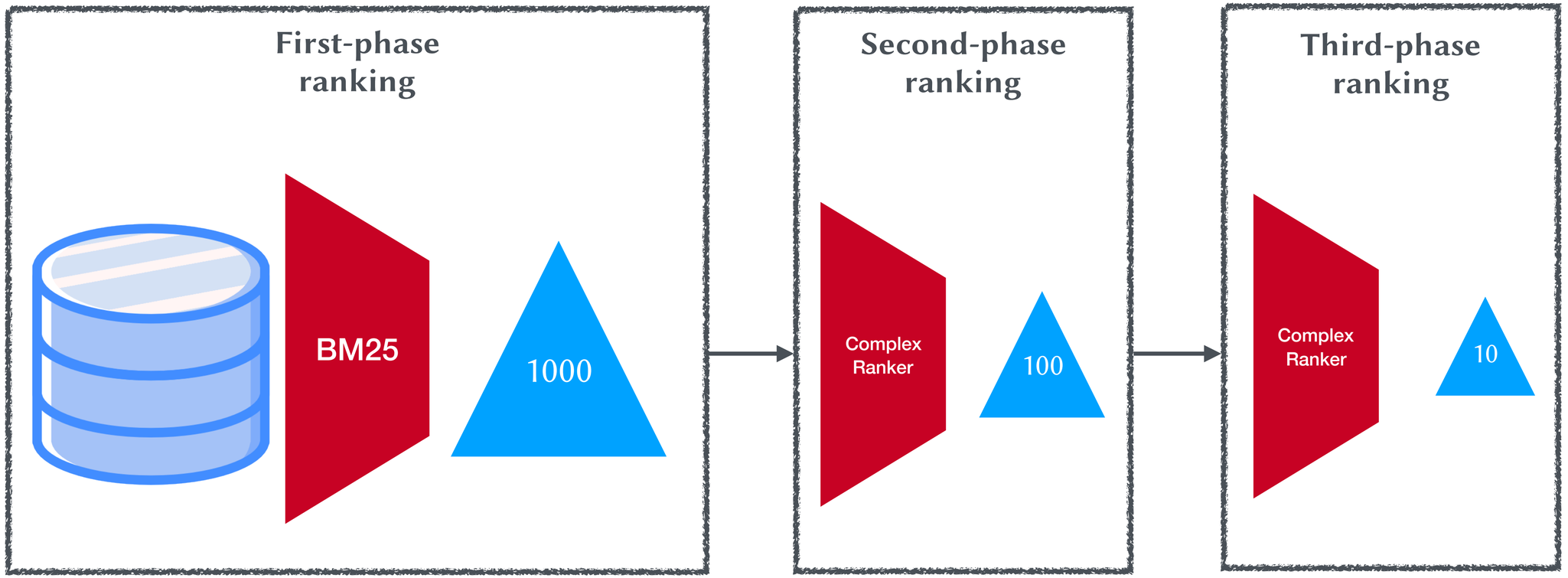
Popular functions:

- Cosine similarity
- Tf-idf
- BM25
- Language models

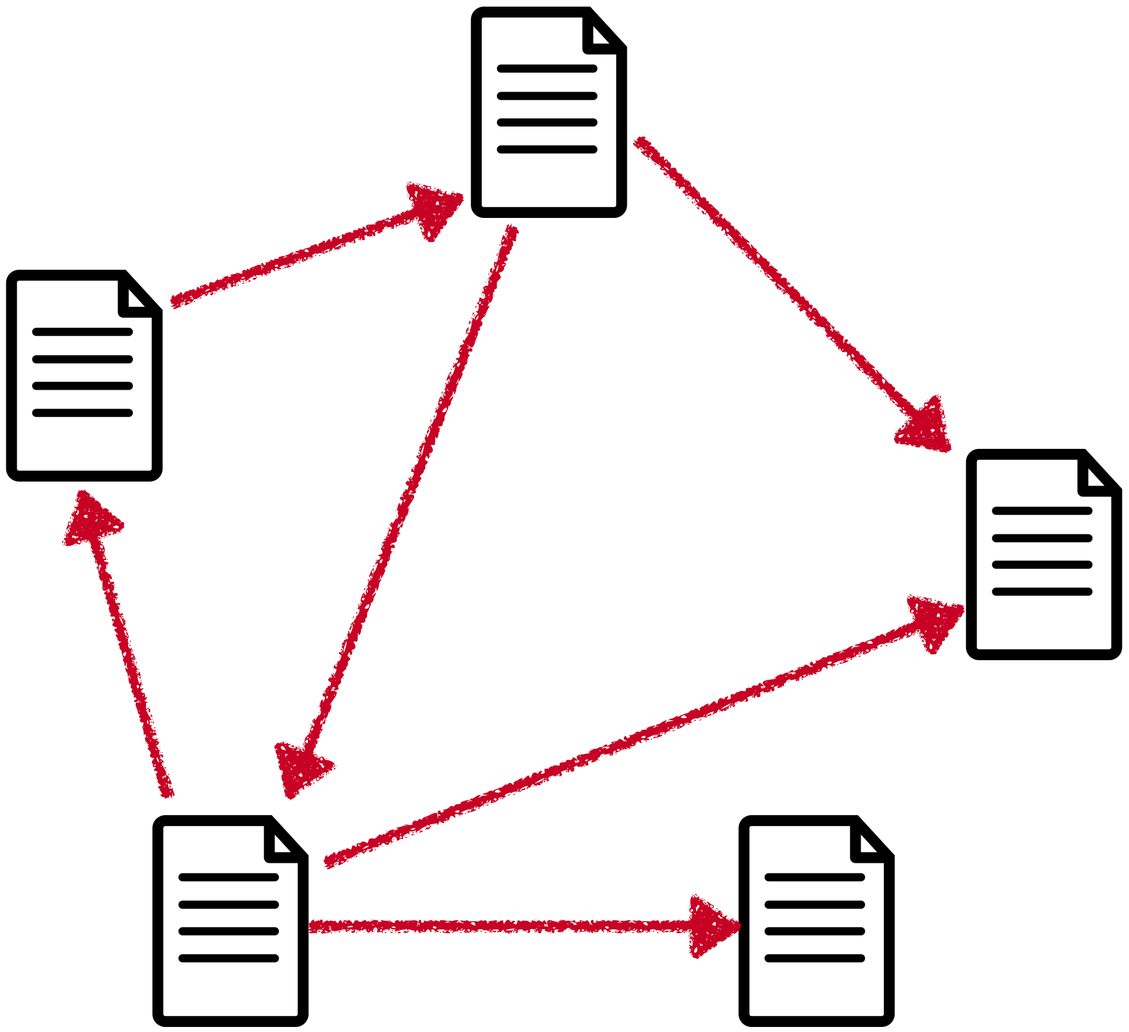
Ranking

Cascade Ranking

- Complex Ranking Functions are expensive
- Machine Learning models require hundreds of features
- Perform re-ranking on subset of documents
- Build a multi-level architecture, from simpler to more complex



The Web as a graph



Link counts as simple measures of popularity:

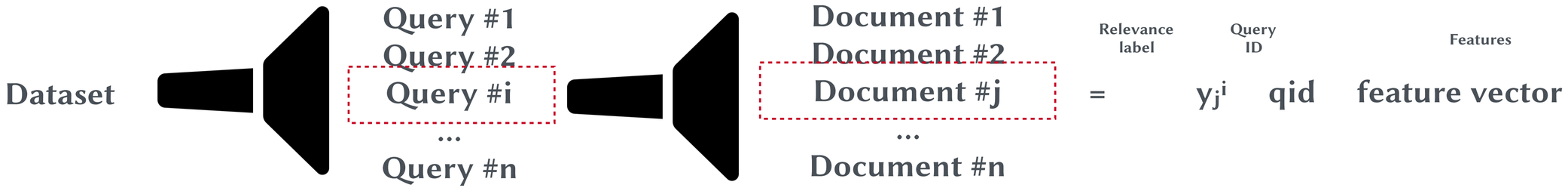
- **Undirected popularity:** each page gets a score given by the number of in-links plus the number of out-links.
- **Directed popularity**: score of a page equal to the number of its in-links.

More complex techniques include **Pagerank**, as a form of actual voting. Designed by Page & Brin as part of a research project that started in 1995 and ended in 1998 with the creation of Google.

Learning to Rank

Learning to Rank is the problem of ranking objects by using machine learning techniques.

Most popular algorithms are **LambdaMART**, **Gradient Boosting**, **RankSVM.**



Open Source implementations:

- **LightGBM** - https://github.com/Microsoft/LightGBM
- **RankLib** - https://sourceforge.net/p/lemur/wiki/RankLib/
- **XGBoost** - https://github.com/dmlc/xgboost
- **jforest** - https://github.com/yasserg/jforests
- **SVM^{rank}** - http://www.cs.cornell.edu/people/tj/svm_light/svm_rank.html

Evaluation

Search Quality Evaluation

How do you tell if users are happy?

An assessment of either **Relevant** or **Non-relevant** for each query and each document.
Also called **relevance judgments**.

- Human judgments, can be expensive and slow
- Click models

Open source libraries:

- **RRE** - https://github.com/SeaseLtd/rated-ranking-evaluator
- **trec_eval** - https://github.com/usnistgov/trec_eval
- **pytrec_eval** - https://github.com/cvangysel/pytrec_eval

Search Quality Evaluation

Evaluation measures:

- **Precision**: the fraction of retrieved documents that are relevant.
- **Recall**: the fraction of relevant documents that are retrieved.
- **NDCG**: Normalized Discounted Cumulative Gain; normalized sum of the graded relevance proportionally penalized with the position of the result.
- **Reciprocal Rank**: it is the multiplicative inverse of the rank of the first "correct" answer.
- **Mean Reciprocal Rank**: the average of the reciprocal ranks computed at query level.
- **Mean Average Precision**: the mean of the average precisions computed at query level.
- **F-Measure**: it measures the effectiveness of retrieval with respect to a user who attaches β times as much importance to recall as precision.
- **Average Precision**: the area under the precision-recall curve

Spam

Large amount of pages are generated or manipulated to achieve high rankings in search engines

- **Textual Methods:** change content of page so that pages appears highly relevant
- **Link-Based Methods:** create artificial in-coming links to that a page has very high Pagerank

How to contrast spam?

- use machine learning classifiers
- cooperative spam fighting systems
- design of more complex ranking functions

Additional application

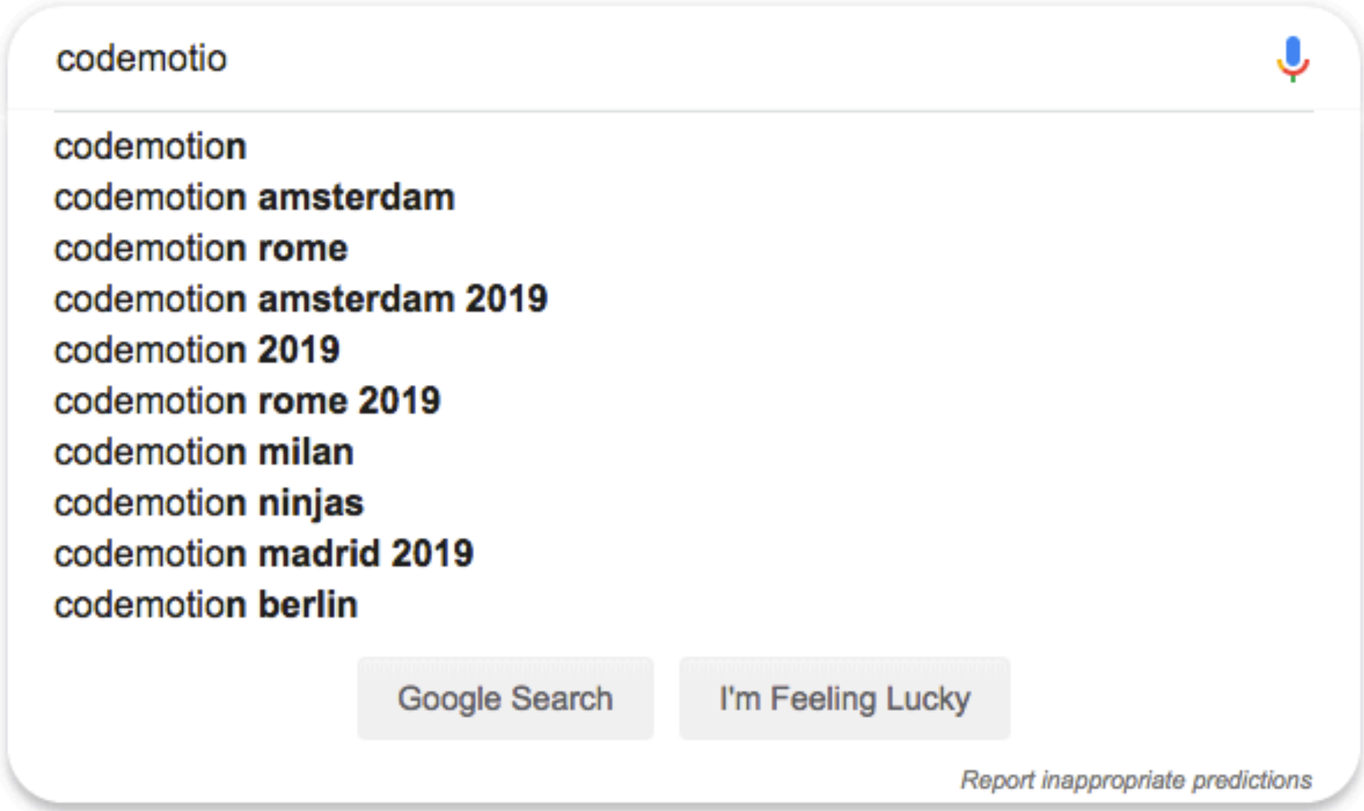
Autocomplete

When you start typing something, you get 10 results.

Trie-based map from string to values.

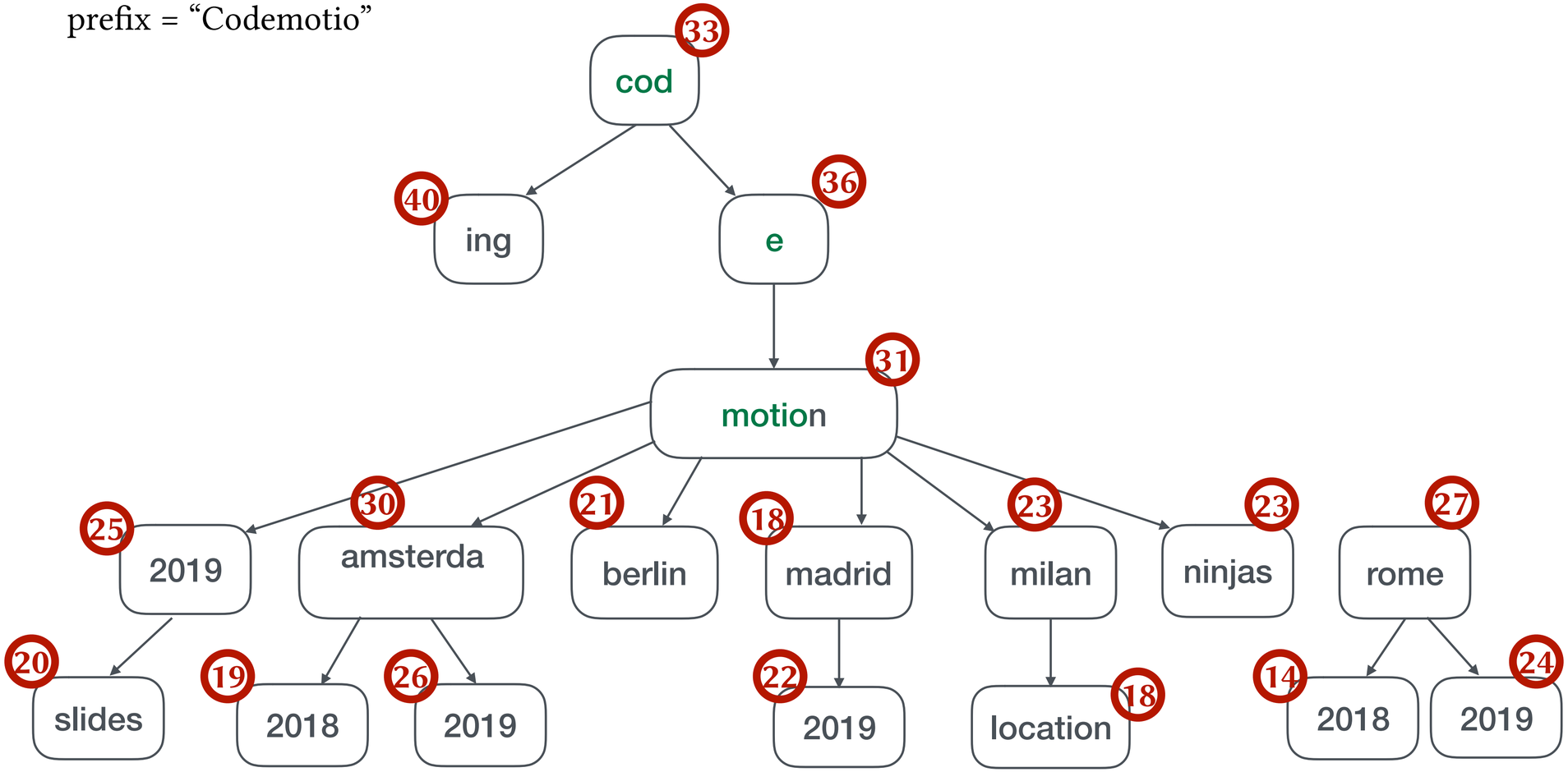
Values are importance scores

Trie can be space efficient since they share nodes, i.e. **Radix tree**

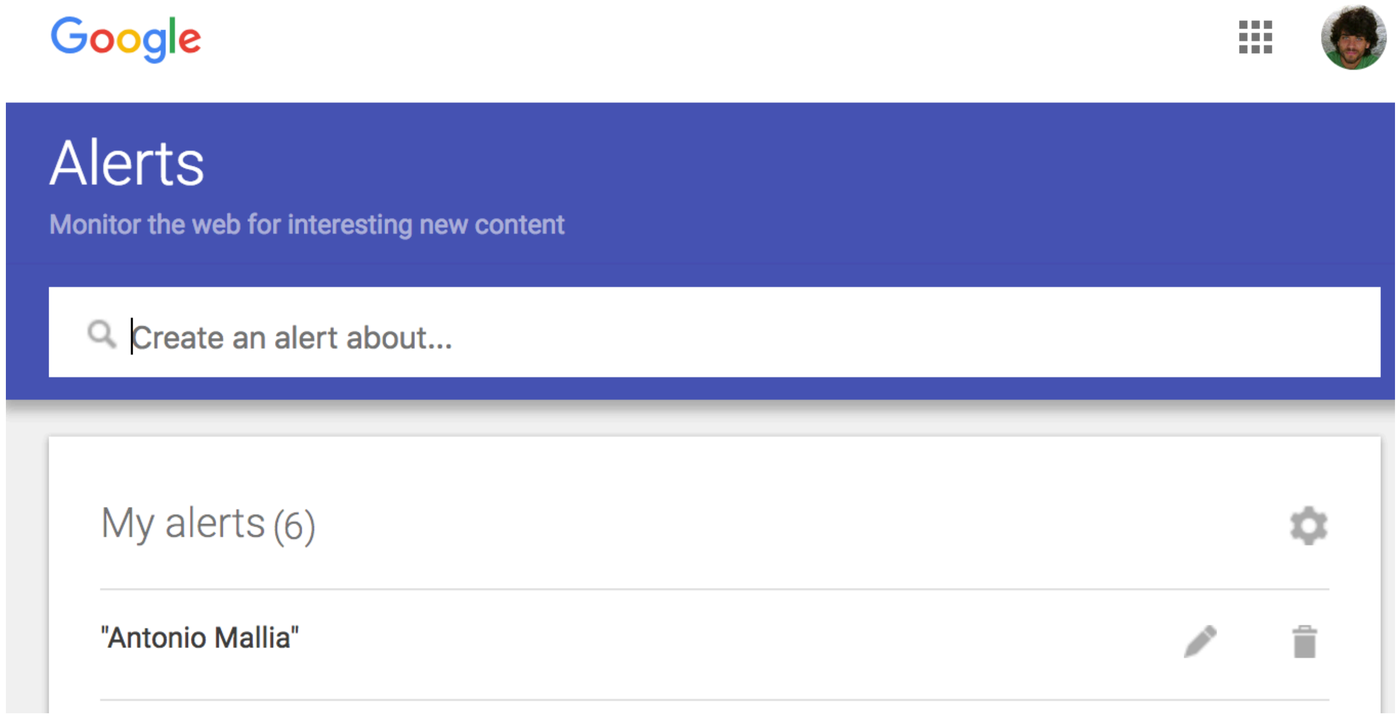


Autocomplete

prefix = “Codemotio”



Alerts

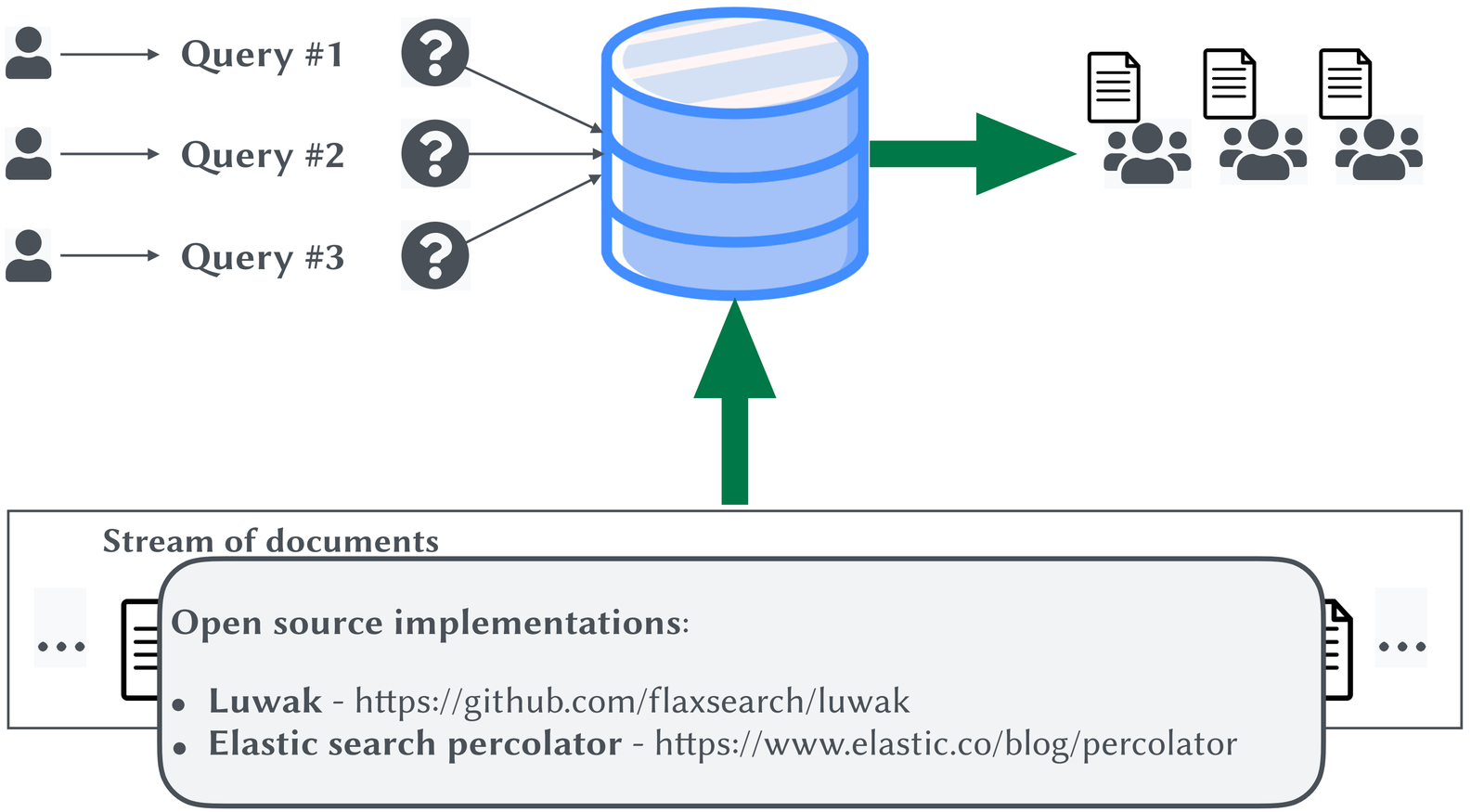


Register some search queries

Get a notification when new web pages matching your query are published

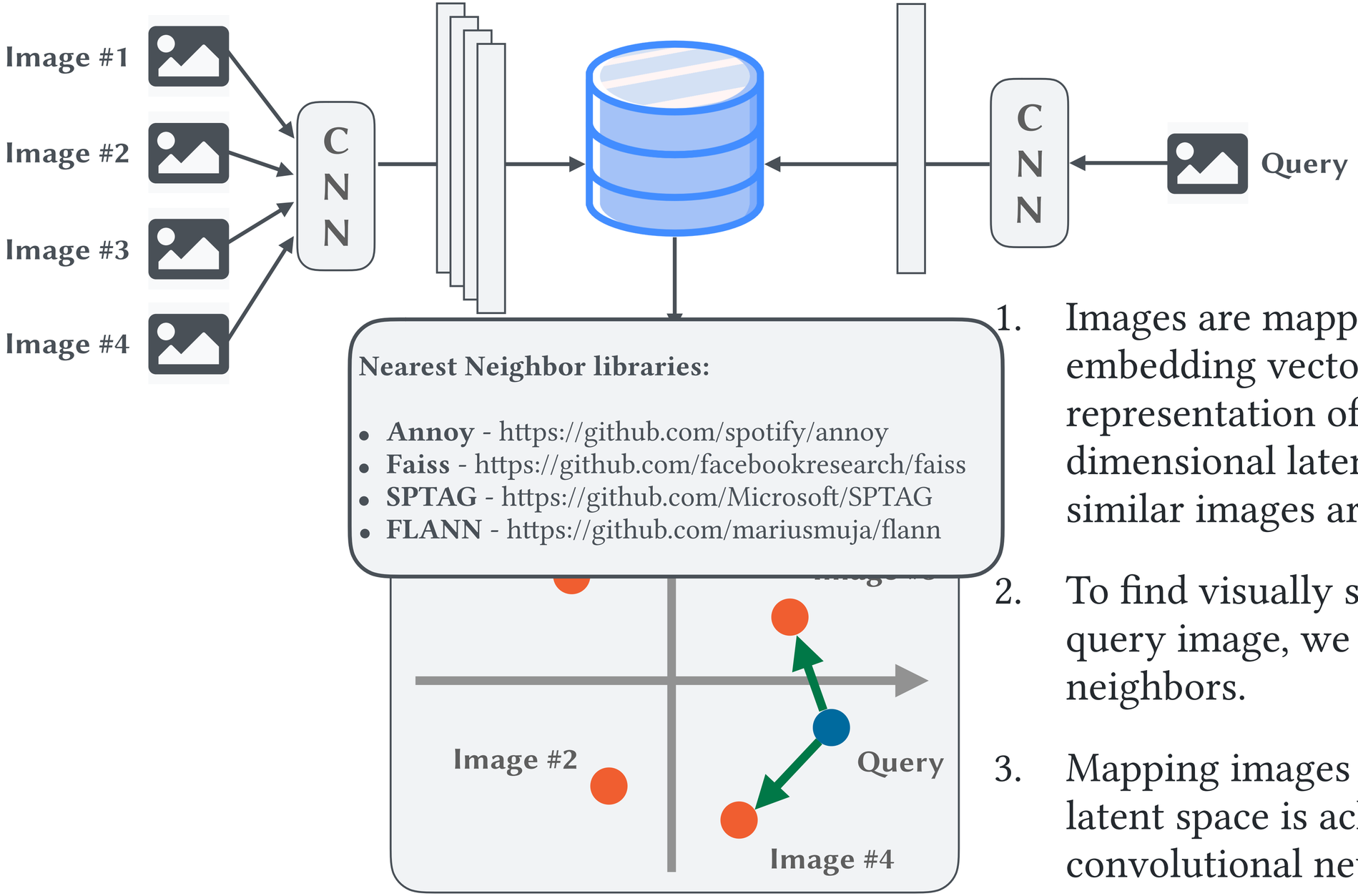
Also referred as **searching over streams** or **reverse query**

Alerts



1. Users' queries are indexed
2. Documents are turned into a disjunctive query
3. Process which queries match the documents

k-Nearest Neighbors



1. Images are mapped into an embedding vector, to get a representation of the input in a low-dimensional latent space where similar images are located nearby.
2. To find visually similar images for a query image, we simply find its neighbors.
3. Mapping images onto a meaningful latent space is achieved with a deep convolutional neural network

Open Source Search Engines

- **Apache Solr:** open source enterprise search platform, written in Java, from the Apache Lucene project - http://lucene.apache.org/solr
- **Elasticsearch:** a distributed, multitenant-capable full-text search engine based on the Lucene library - https://www.elastic.co
- **RediSearch**: FullText Search and Secondary Index module for Redis - https://github.com/RedisLabsModules/RediSearch
- **Vespa:** an engine for low-latency computation over large data sets - https://vespa.ai/
- **Riot:** Go Open Source, Distributed, Simple and efficient full text search engine - https://github.com/go-ego/riot
- **Belve**: full-text search and indexing for Go - http://blevesearch.com/
- **Tantivy**: a full-text search engine library inspired by Apache Lucene and written in Rust - https://github.com/tantivy-search/tantivy
- **Toshi**: a full-text search engine in rust - https://github.com/toshi-search/Toshi
- **Terrier:** highly flexible, efficient, and effective open source search engine, readily deployable on large-scale collections of documents - http://terrier.org/
- **MG4J**: highly customisable, high-performance, full-fledged search engine - http://mg4j.di.unimi.it/

PISA: Performant Indexes and Search for Academia

PISA: Performant Indexes and Search for Academia

Official Documentation

Description

PISA is a text search engine able to run on large-scale collections of documents. It allows researchers to experiment with state-of-the-art techniques, allowing an ideal environment for rapid development.

Some features of PISA are listed below:

- Written in C++ for performance;
- Indexing & Parsing capabilities;
- Many index compression methods implemented;
- Many query processing algorithms implemented;
- Implementation of document reordering;
- Free and open-source with permissive license;

https://github.com/pisa-engine/pisa/

Standard Datasets

Common Crawl Corpus: petabytes of data collected since 2011
http://commoncrawl.org/

The ClueWeb09 Dataset: one billion web pages in ten languages that were collected in January and February 2009
https://lemurproject.org/clueweb09.php/

The ClueWeb12 Dataset: English web pages, collected between February 10, 2012 and May 10, 2012 -
https://lemurproject.org/clueweb12/

GOV2 Test Collection: a crawl of .gov sites (early 2004)
http://ir.dcs.gla.ac.uk/test_collections/gov2-summary.htm

The New York Times Annotated Corpus
https://catalog.ldc.upenn.edu/LDC2008T19

Washington Post Corpus
https://trec.nist.gov/data/wapost/

TREC Collections
https://trec.nist.gov/data.html

Recommended Books

Introduction to Information Retrieval, by C. Manning, P. Raghavan, H. Schuetze. (Free online version of book)

Search Engines: Information Retrieval in Practice, by B. Croft, D. Metzler, and T. Strohman. Addison-Wesley 2009.

Information Retrieval: Implementing and Evaluating Search Engines, by S. Buettcher, C. Clarke, and G. Cormack. MIT Press, 2010.

Scalability Challenges in Web Search Engines, by B. Cambazoglu and R. Baeza-Yates, Morgan&Claypool, 2015.

Managing Gigabytes : Compressing and Indexing Documents and Images, by I. Witten, A. Moffat, and T. Bell. Morgan Kaufmann 1999.

Any questions?

Slides

https://www.antoniomallia.it/talks.html