# Chapter 4

# Text Search in a Nutshell

Sergej Sizov

Web Retrieval

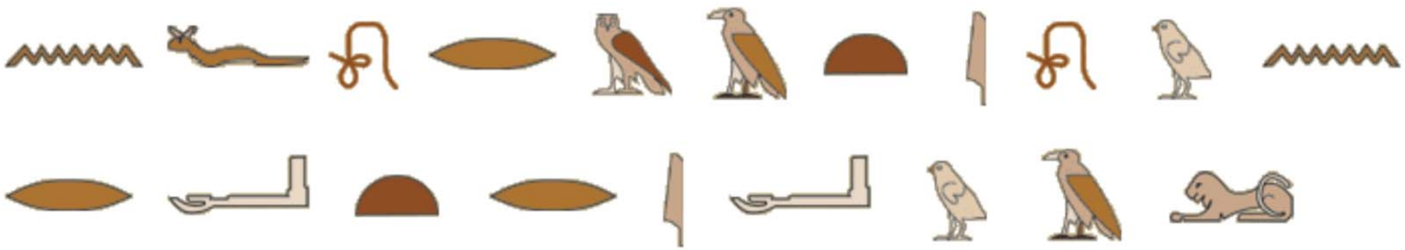
Summer term 2010

UNIVERSITÄT KOBLENZ · LANDAU

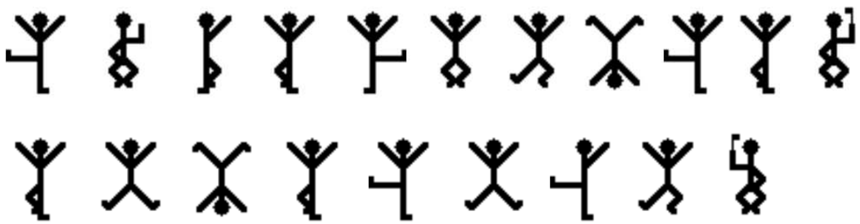
# Motivation

## Information Retrieval

a)



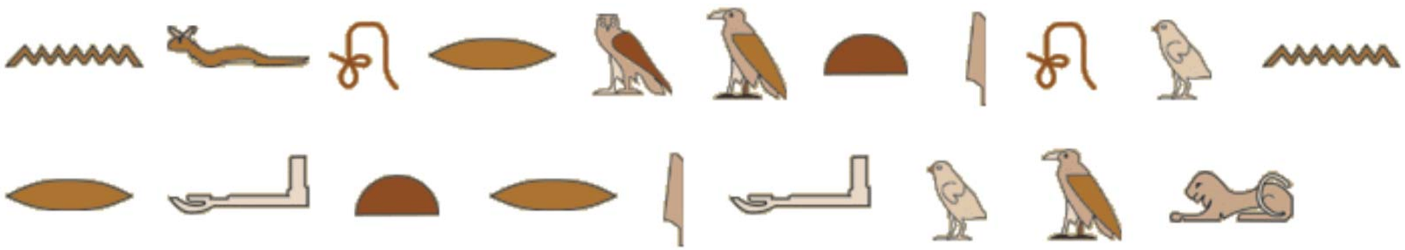
b)

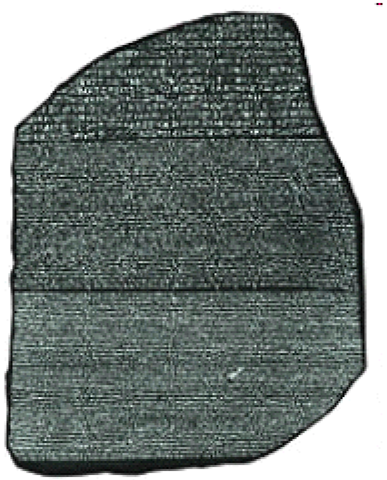


c) 73 110 102 111 114 109 97 116 105 111 110

82 101 116 114 105 101 118 97 108

# Motivation (2)

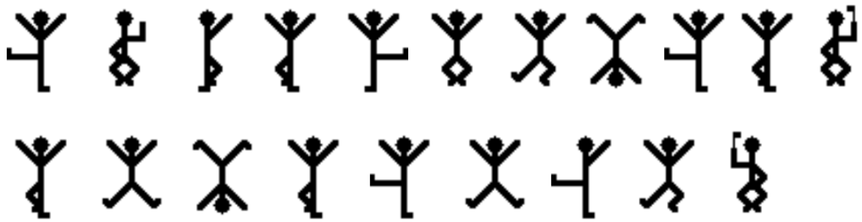




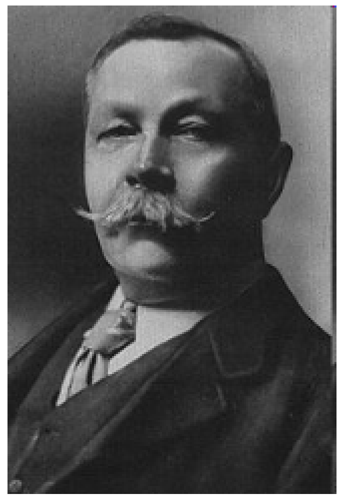
**Stein von Rosetta**

**Jean-François Champollion**

# Motivation (3)



**Sherlock Holmes**



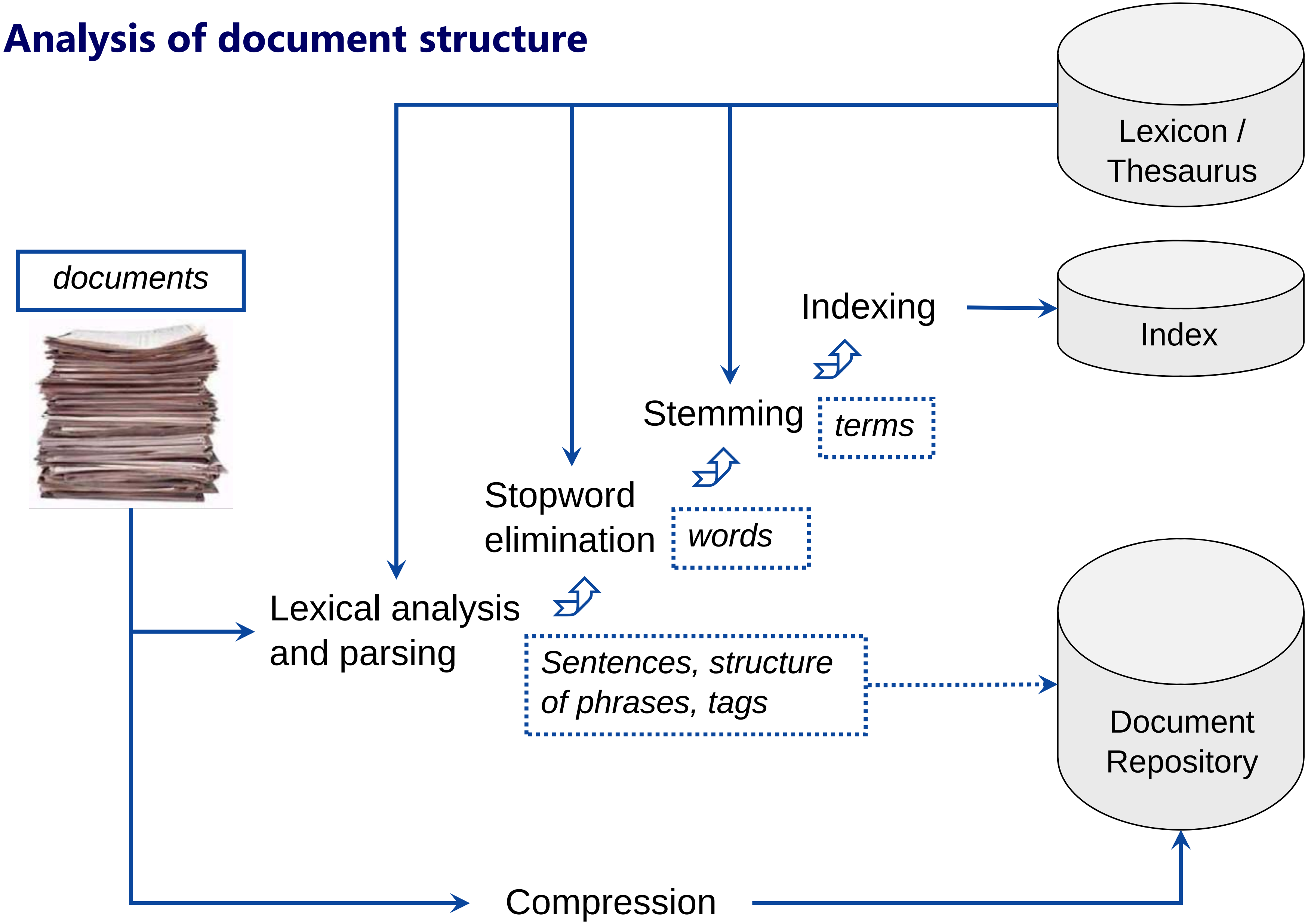
**Arthur Conan Doyle**

# Motivation (4)

73 110 102 111 114 109 97 116 105 111 110

82 101 116 114 105 101 118 97 108

## Analysis of document structure



# Stopword Elimination

Lookups in stopword lists

(potentially using domain-specific dictionary - lexikon/thesaurus)

e.g. „definition“ or „theorem“ for math documents

Common language-specific stopwords (prepositions, conjunctions, pronouns, „overloaded“ verbs etc. – several hundreds of stopwords):

a, also, an, and, as, at, be, but, by,
can, could, do, for, from, go,
have, he, her, here, his, how,
I, if, in, into, it, its,
my, of, on, or, our, say, she,
that, the, their, there, therefore, they,
this, these, those, through, to, until,
we, what, when, where, which, while, who, with, would,
you, your, ....

# Morphologic Reduction (Lemmatization)

- Grammatical base form:

Nominative for nouns, infinitive for verbs, plural to singular, passive to active, etc.

Examples:

• „students“ to „student“, „going“ to „go“

Kontext dependent and phrase dependent

• „went“ to „go“,

• „have been“ to „be“

- Linguistical base form

Tracking of flexion (e.g. declination), composition, substantization, etc.

Examples:

• „nonfood“ to „food“

• „founds“ to „find“

• „Schweinkram“, „Schweinshaxe“ und „Schweinebraten“ to „Schwein“ etc.

## Stemming

**Ideas**:

- use of dictionaries
- recognition through analysis of the linguistical strukture
- affix elimination: removal of prefixes and suffixes using (heuristic) rules

**Example**:

stresses → stress, stressing → stress, symbols → symbol

using rules  sses → ss, ing → e, s → e, etc.

Note: the usefulness of stemming in IR is not undisputable

**Example**:

Bill is operating a company.

On his computer he runs the … operating system.

## Thesaurus

For each **concept** (word sense) we store:

- the set of synonyms or instances (*words*)
- the set of generalizations and specializations (hypernyms, hyponyms)
- „part-of" and „contains" relationships (meronyms, holonyms)
- concept-example relationships (e.g. fairytale and cinderella)
- the set of antonyms

For each **word** we store:

- the set of associated *concepts* (e.g. with some statistics)

(for disambiguation of polysems or homonyms)

Example: WordNet, http://www.cogsci.princeton.edu/~wn

# Vector Space Model

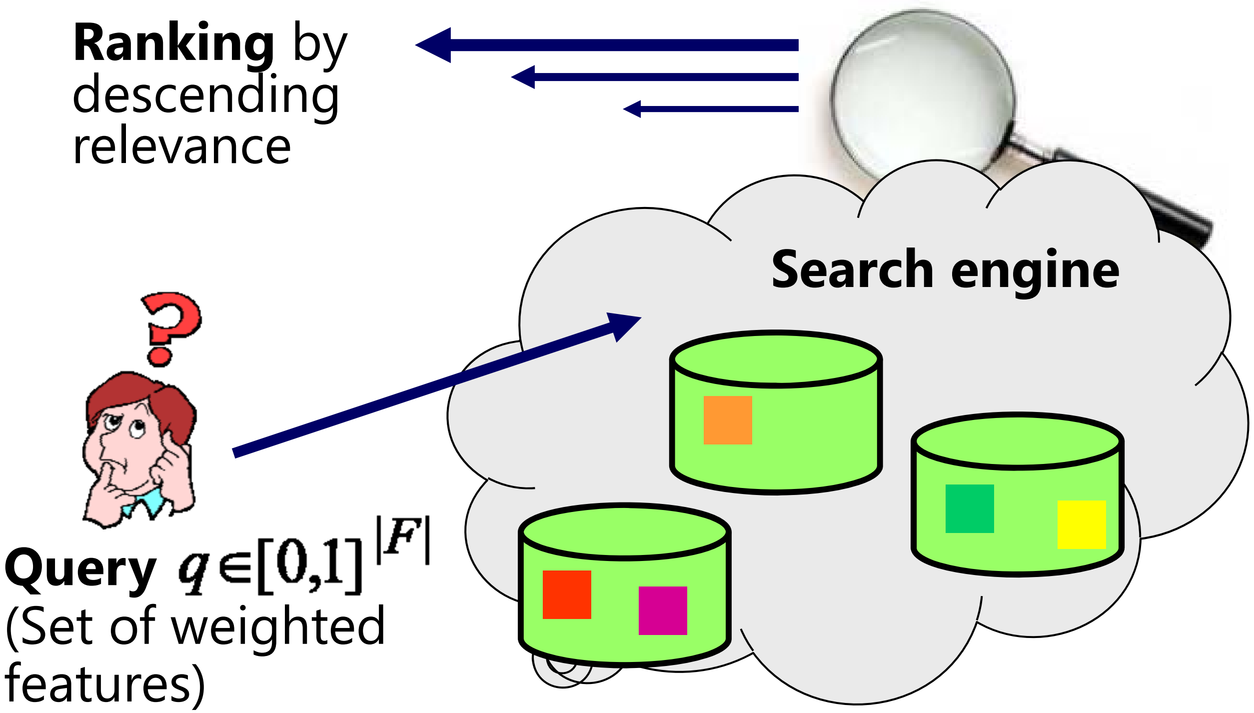
## Basic Principles:

- Feature Space: words in documents are reduced to terms.
- Document model: each document is represented as vector in $[0,1]^{|F|}$

whereby $d_{ij}$ is the weight of the $j$-th term in $d_i$ .

- Queries: queries are vectors $q_i$ in $[0,1]^{|F|}$
- Relevance: relevance of results is based on similarity function for vector space $[0,1]^{|F|}$
- Indexing: for each term there is a list of Doc-IDs (e.g. URLs) with associated weights, implemented as „inverted file" (search tree or hash table)
- Query execution: query is decomposed into several index-lookups for particular query terms in order to determine the ranked list of candidates

# Vector Space Model Relevance Ranking




**Similarity metric:**

$$sim(d_i, q) := \frac{\sum_{j=1}^{|F|} d_{ij}\, q_j}{\sqrt{\sum_{j=1}^{|F|} d_{ij}^2 \sum_{j=1}^{|F|} q_j^2}}$$

Documents are **feature vectors** $d_i \in [0,1]^{|F|}$

e.g., using:

$$d_{ij} := w_{ij} / \sqrt{\Sigma_k w_{ik}^2}$$

$$w_{ij} := \frac{freq(f_j, d_i)}{\max_k freq(f_k, d_i)} \log \frac{\#docs}{\#docs\ with\ f_i}$$

tf*idf formula

## Term Weighting

We consider following characteristics for *N* documents and *M* terms:

- $tf_{ij}$ : term frequency - frequency of term $t_i$ in document $d_j$
- $df_i$ : document frequency - number of documents that contain $t_i$
- $idf_i$ : inverse document frequency = $N / df_i$
- $cf_i$ : corpus frequency – frequency of $t_i$ in the corpus (e.g. separate counting of title terms, body terms, etc.)

Basic idea:

- The weight $w_{ij}$ of term $t_i$ in document $d_j$ should increase monotonically with $tf_{ij}$ and $idf_i$

First idea:

- use some tf-idf combination, e.g. $w_{ij} = f_{ij} * idf_i$ (**tf-idf formula**)
- $w_{ij}$ can be normalized: $$d_{ij} = \frac{w_{ij}}{\sqrt{\sum_k w_{kj}^2}}$$

# Variations of Term Weighting

Empirical results show that tf and idf values usually must be dampened or normalized

- normalized *tf* values $$tf_{ij} = \frac{tf_{ij}}{max_k \ tf_{kj}}$$
- *tf* weighting mit dampening $$tf_{ij} = 1 + log \ tf_{ij}$$
- *idf* weighting mit dampening $$idf_i = log \frac{N}{df_i}$$
- commont combination: (tf*idf formula)

$$\boxed{\begin{aligned} w_{ij} &= \frac{tf_{ij}}{max_k \ tf_{ij}} log \frac{N}{df_i} \\ d_{ij} &= \frac{w_{ij}}{\sqrt{\sum_k w_{kj}^2}} \end{aligned}}$$

# Term Weighting in Queries

Depending of query interface and user category, simple or advanced term weightings may be used

- simple weighting: $w_{ij} \in \{0, 1\}$
- advanced weighting: $w_{ij} = \left(0.5 + \frac{0.5 \cdot tf_{ij}}{max_k \; tf_{ij}}\right) \cdot log \frac{N}{df_i}$
- term ranking: $w_{ij} = \frac{1}{k}$

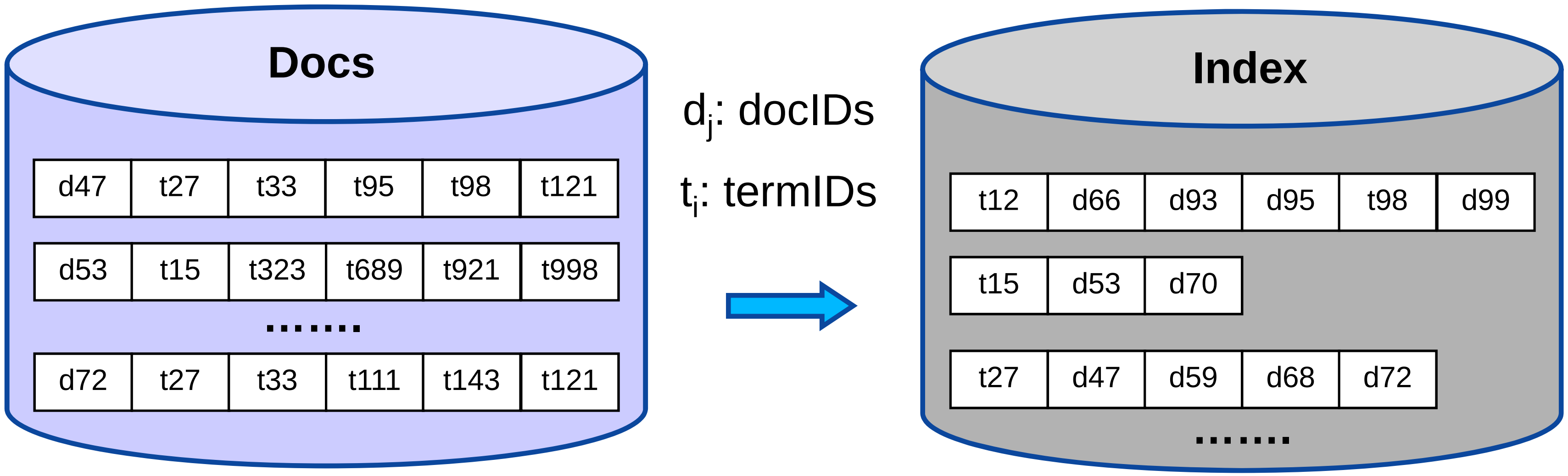
(when conjunctive query *q* contains *k* terms and $t_i$ is in $k^{th}$ position)

# Text Indexing and Query Execution

**Konzeptionell**:

invertierte Dateien (invertierte Listen) mit binärer Suche

nach Suchschlüsseln (Felder von Records, Strings in Texten)



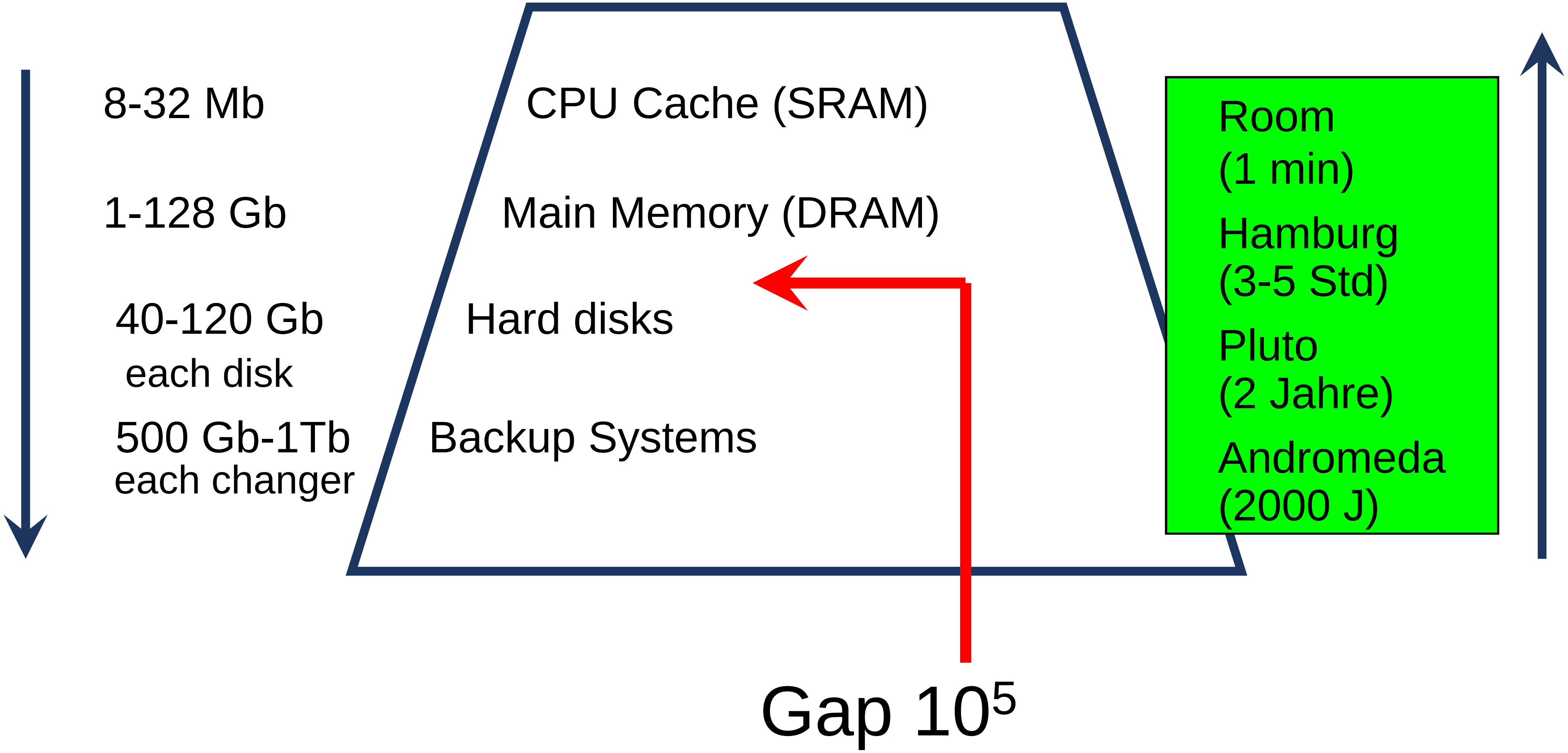

Problem:

Speicherungsorganisation in Plattenblöcken (pagination) und effiziente Implementierung der (binären) Suche für

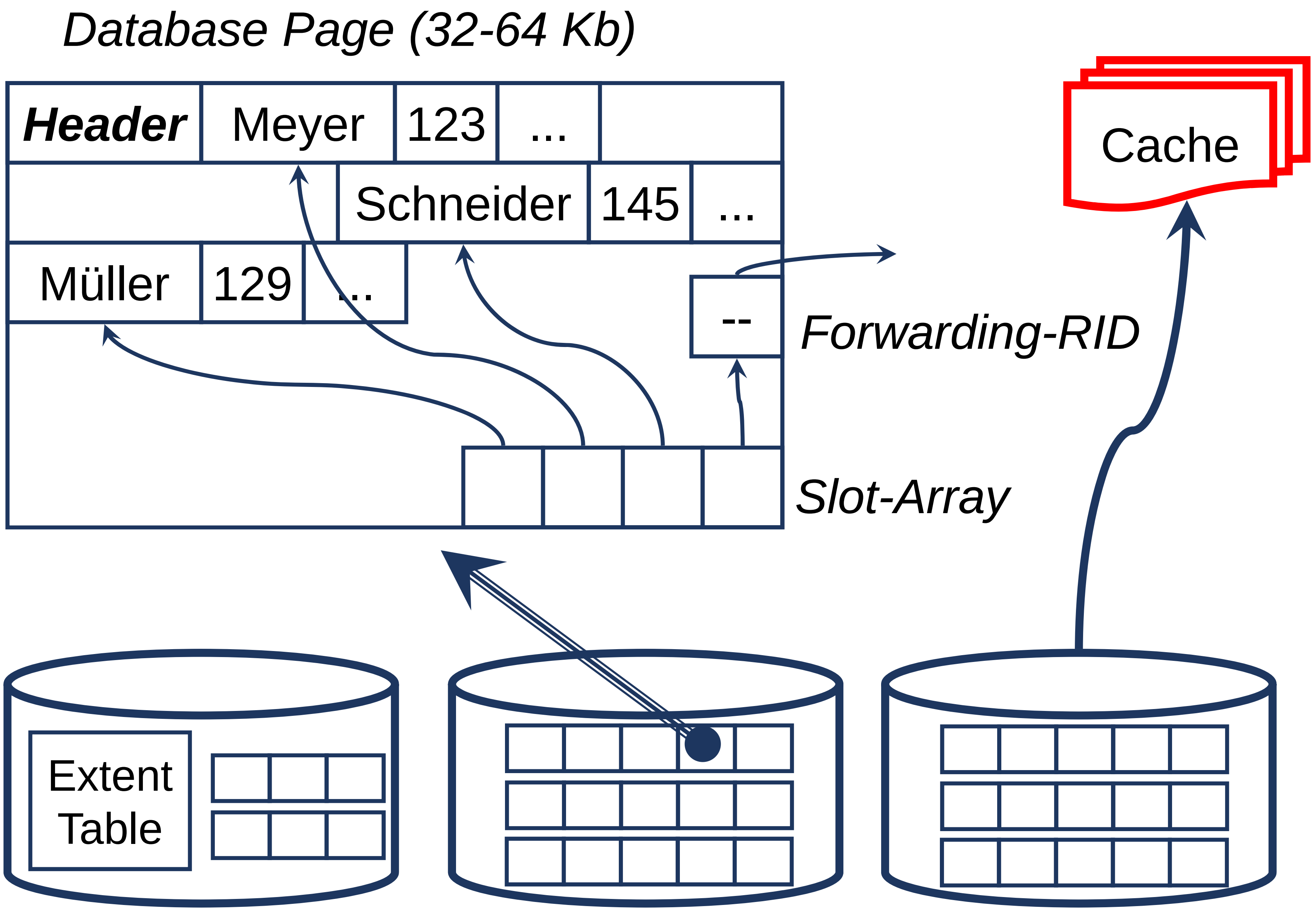
- Exact-Match-Suche: search (key) returns ids
- Bereichssuche: search (lowkey, highkey) returns ids
- Präfixstringsuche: search (prefix) returns ids

bei dynamischen Updates

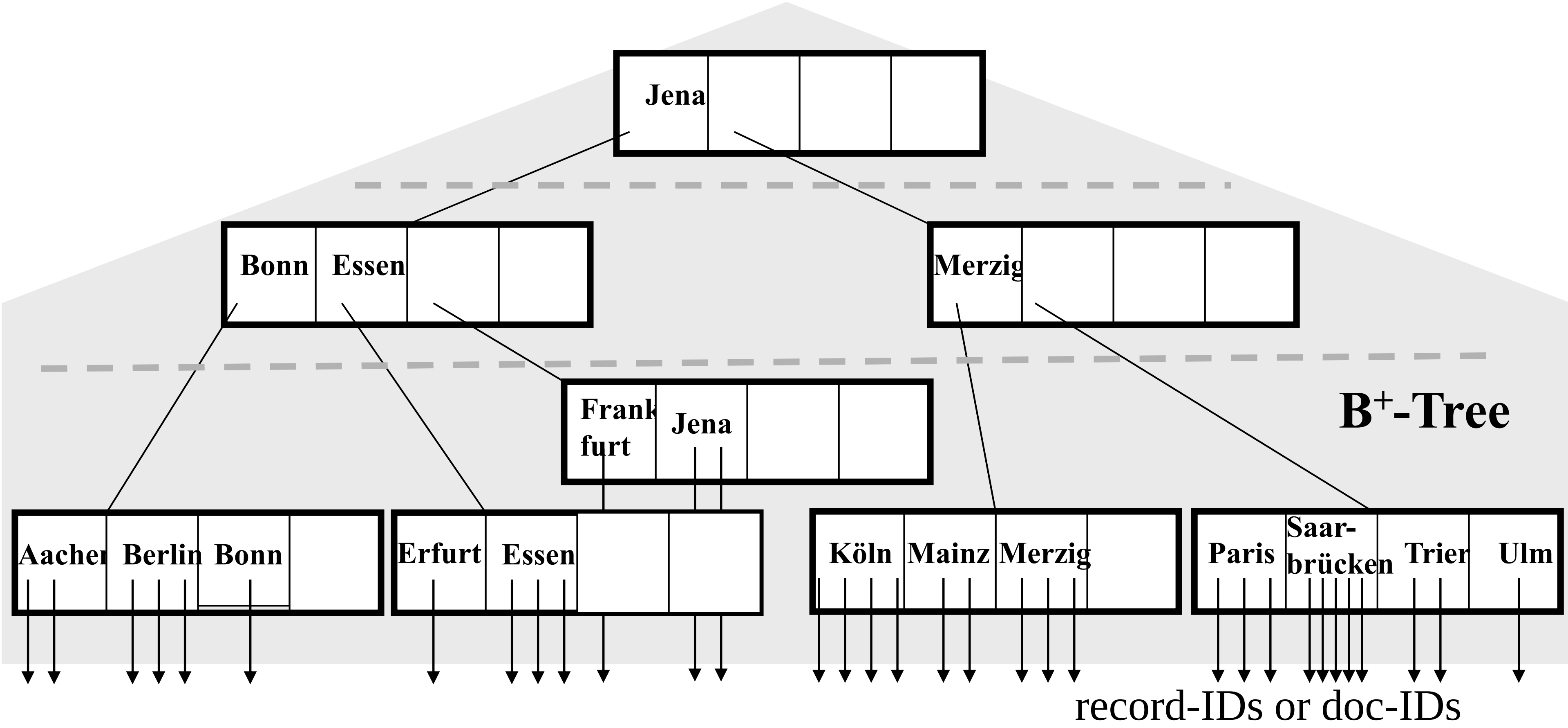
# Properties of Media Types



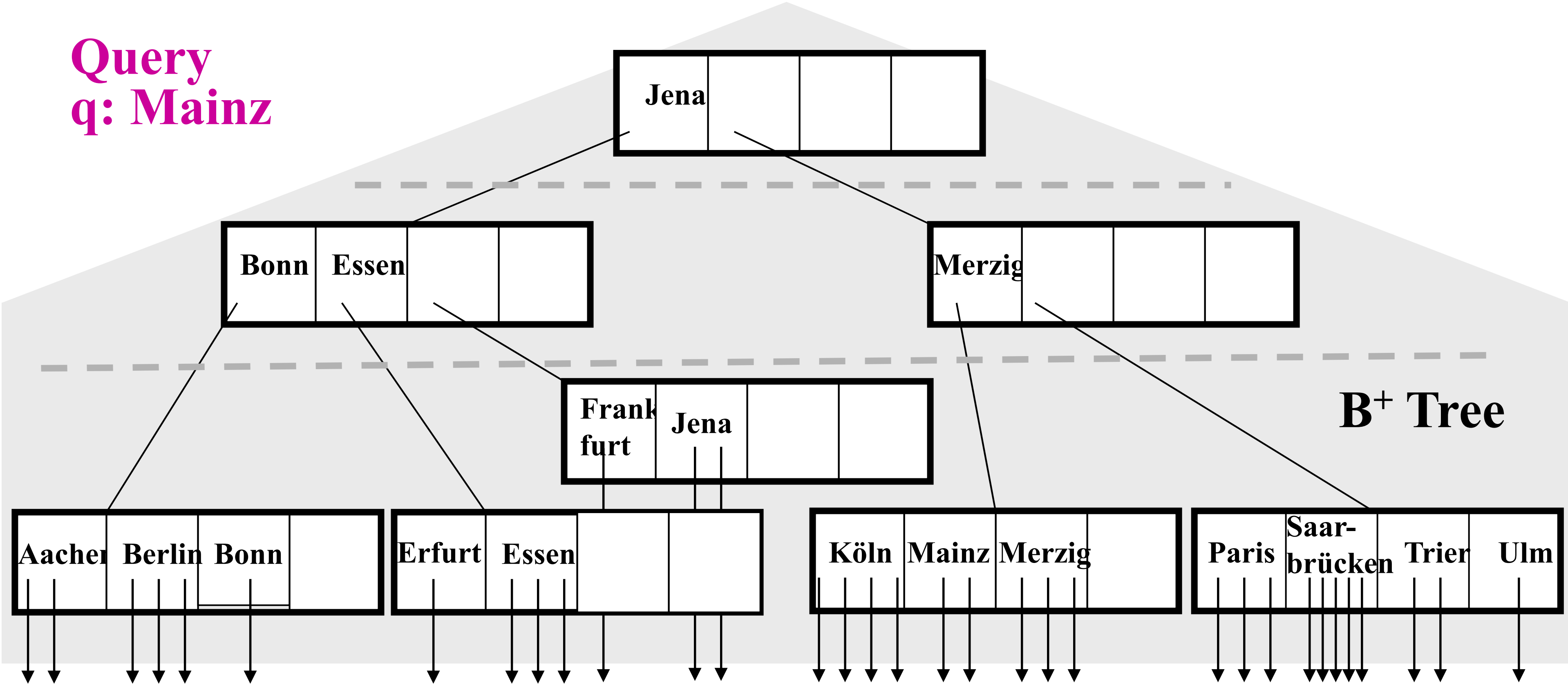
# Access: Physical Data Organization



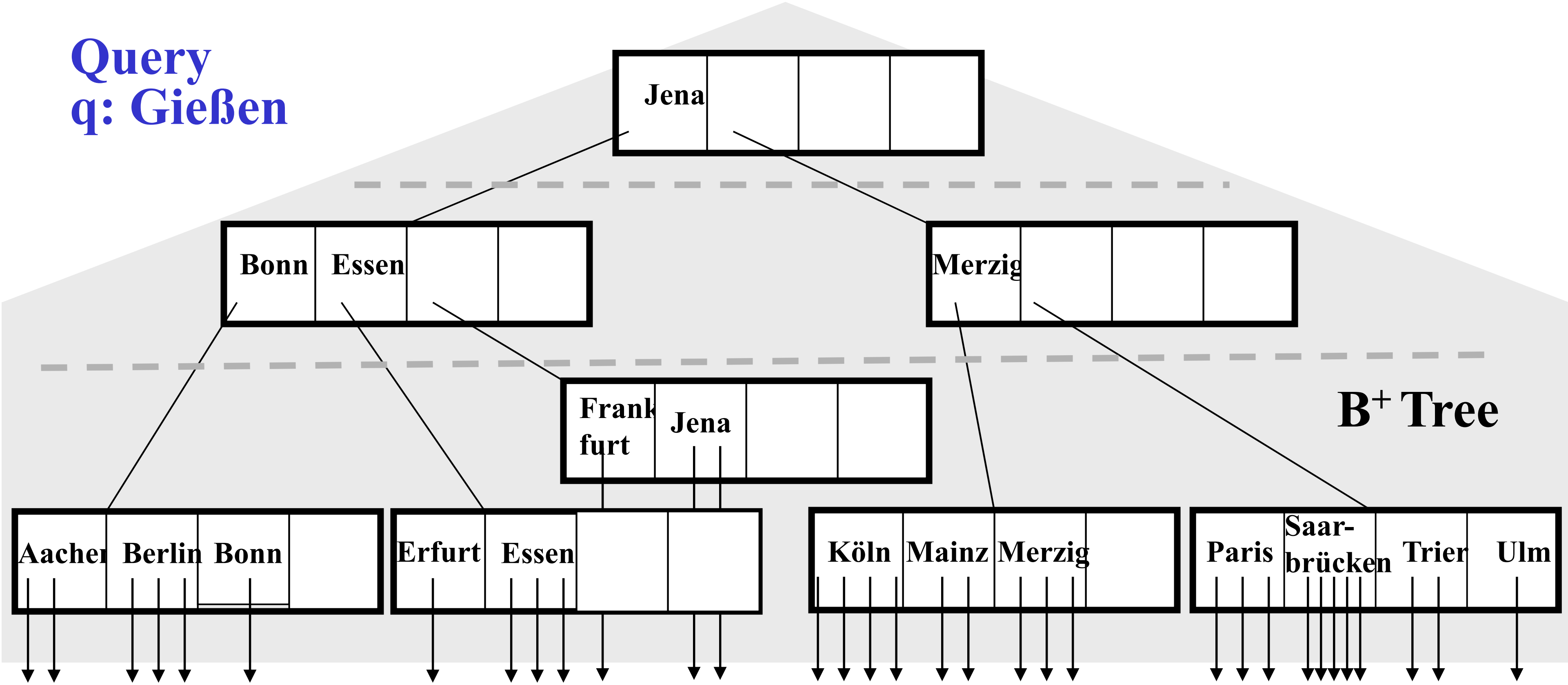
# $B^+$ Trees: Example



# $B^+$ Trees: Lookup (1)

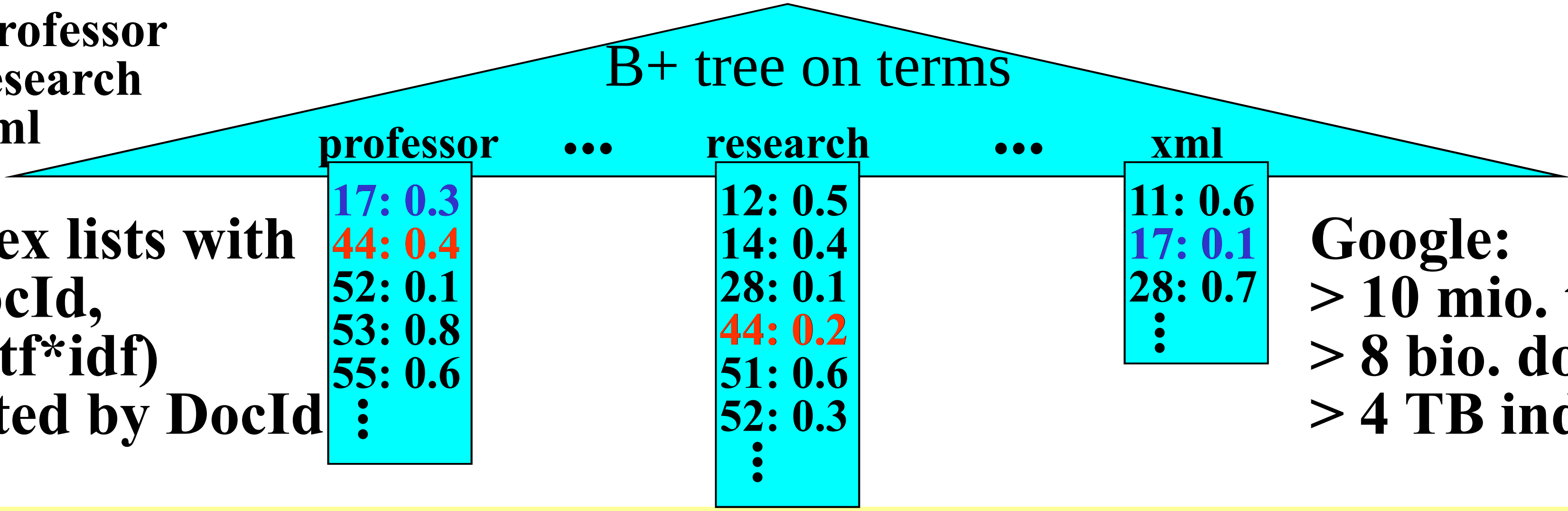


# B⁺ Trees: Lookup (2)



## DBS-Style Top-k Query Processing

q: professor
research
xml




index lists with (DocId, s = tf*idf) sorted by DocId

Google:
> 10 mio. terms
> 8 bio. docs
> 4 TB index

<u>**Given:**</u> query $q = t_1\ t_2 \ldots t_z$ with z (conjunctive) keywords
similarity scoring function score(q,d) for docs $d \in D$, e.g.: $\vec{q} \cdot \vec{d}$
with precomputed scores (index weights) $s_i(d)$ for which $q_i \neq 0$

<u>**Find:**</u> top k results w.r.t. score(q,d) =aggr{$s_i(d)$}(e.g.: $\Sigma_{i \in q}\ s_i(d)$)

***Naive join&sort QP algorithm:***

top-k (
$\sigma$[term=$t_1$] (index) $\bowtie_{DocId}$
$\sigma$[term=$t_2$] (index) $\bowtie_{DocId}$
...  $\bowtie_{DocId}$
$\sigma$[term=$t_z$] (index)  order by s desc)

# Index List Processing by Merge Join

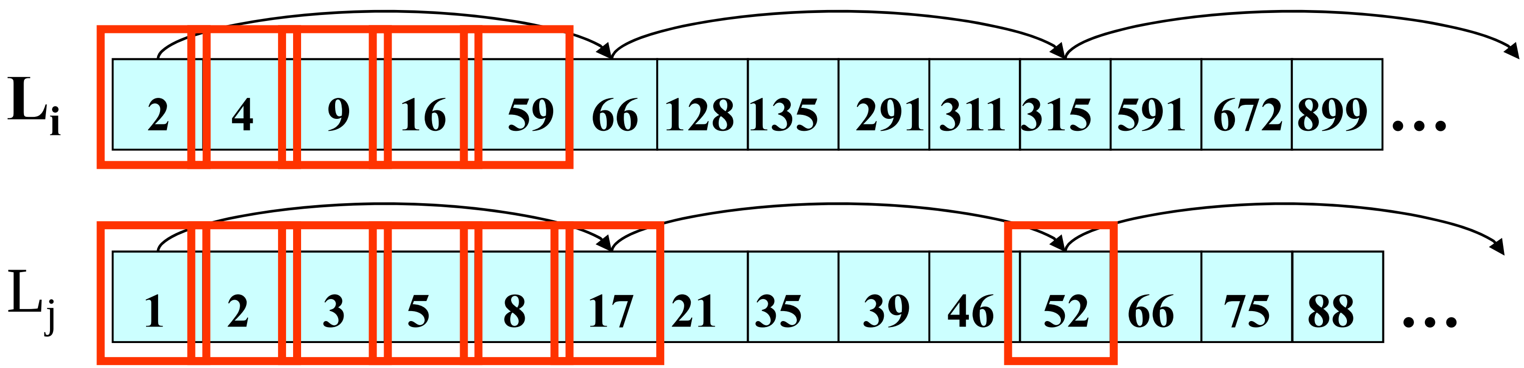
Keep L(i) in **ascending order of doc ids**

Compress L(i) by actually storing the gaps between successive doc ids
(or using some more sophisticated prefix-free code)

QP may start with those L(i) lists that are short and have high idf

Candidate results need to be looked up in other lists L(j)

To avoid having to uncompress the entire list L(j),
L(j) is encoded into groups of entries
with a skip pointer at the start of each group
→ sqrt(n) evenly spaced skip pointers for list of length n

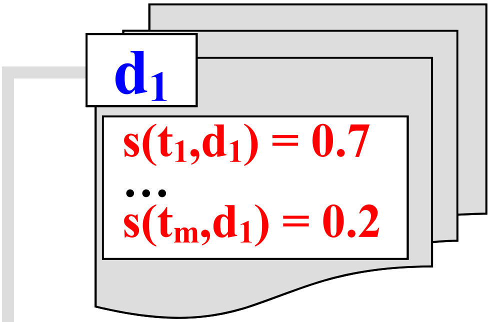


# Efficient Top-k Search

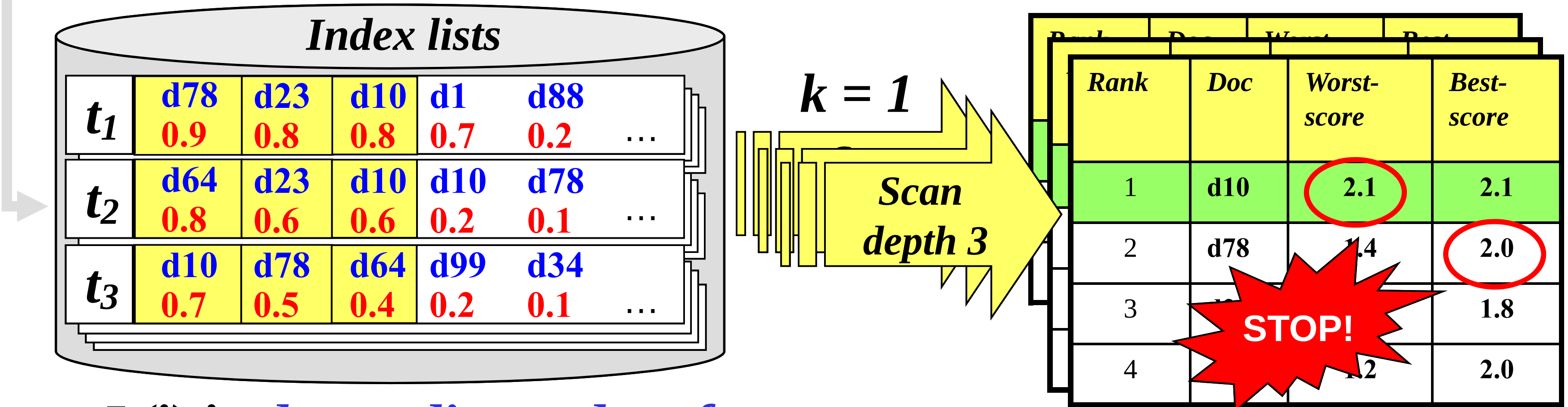
[Buckley85, Güntzer/Balke/Kießling 00, Fagin01]

*threshold algorithms: efficient & principled top-k query processing with monotonic score aggr.*

**Data items: $d_1, \dots, d_n$**




**Query: $q = (t_1, t_2, t_3)$**




**TA with sorted access only (NRA):**
**can index lists; consider d at $pos_i$ in $L_i$;**
**$E(d) := E(d) \cup \{i\}$; $high_i := s(t_i,d)$;**
**worstscore(d) := $aggr\{s(t_\nu,d) \mid \nu \in E(d)\}$;**
**bestscore(d) := aggr{worstscore(d), $aggr\{high_\nu \mid \nu \notin E(d)\}\}$;**
**if worstscore(d) > min-k then add d to top-k**
**min-k := min{worstscore(d') | d' $\in$ top-k};**
**else if bestscore(d) > min-k then**
**cand := cand $\cup$ {d}; s**
**threshold := max {bestscore(d') | d' $\in$ cand};**
**if threshold $\leq$ min-k then exit;**

***keep L(i) in descending order of scores***

# Threshold Algorithm (TA, Quick-Combine, MinPro)

(Fagin'01; Güntzer/Balke/Kießling; Nepal/Ramakrishna)

**scan all lists $L_i$ (i=1..m) in parallel:**
**consider dj at position $pos_i$ in Li;**
**$high_i := s_i(dj)$;**
**if dj $\notin$ top-k then {**
**look up $s_\nu(dj)$ in all lists $L_\nu$ with $\nu \neq i$; // random access**
**compute $s(dj) :=$ aggr $\{s_\nu(dj) \mid \nu=1..m\}$;**
**if s(dj) > min score among top-k then**
**add dj to top-k and remove min-score d from top-k; };**
**threshold := aggr $\{high_\nu \mid \nu=1..m\}$;**
**if min score among top-k $\geq$ threshold then exit;**

***but random accesses are expensive !***

m=3
aggr: sum
k=2

| | | |
|---|---|---|
| f: 0.5 | a: 0.55 | h: 0.35 |
| b: 0.4 | b: 0.2 | d: 0.35 |
| c: 0.35 | f: 0.2 | b: 0.2 |
| a: 0.3 | g: 0.2 | a: 0.1 |
| h: 0.1 | c: 0.1 | c: 0.05 |
| d: 0.1 | | f: 0.05 |

**top-k:**

~~f: 0.75~~
a: 0.95
b: 0.8

# No-Random-Access Algorithm

(NRA, Stream-Combine, TA-Sorted)

**scan index lists in parallel:**

**consider dj at position $pos_i$ in Li;**
**E(dj) := E(dj) ∪ {i}; $high_i$ := si(q,dj);**
**bestscore(dj) := aggr{x1, ..., xm)**
**with xi := si(q,dj) for i∈E(dj), $high_i$ for i ∉E(dj);**
**worstscore(dj) := aggr{x1, ..., xm)**
**with xi := si(q,dj) for i∈E(dj), 0 for i ∉E(dj);**
**top-k := k docs with largest worstscore;**
**threshold := bestscore{d | d not in top-k};**
**if min worstscore among top-k ≥ threshold then exit;**

m=3
aggr: sum
k=2

| | | |
|---|---|---|
| f: 0.5 | a: 0.55 | h: 0.35 |
| b: 0.4 | b: 0.2 | d: 0.35 |
| c: 0.35 | f: 0.2 | b: 0.2 |
| a: 0.3 | g: 0.2 | a: 0.1 |
| h: 0.1 | c: 0.1 | c: 0.05 |
| d: 0.1 | | f: 0.05 |

**top-k:**

a: 0.95
b: 0.8

**candidates:**

f: 0.7 + ? ≤ 0.7 + 0.1
h: 0.35 + ? ≤ 0.35 + 0.5
c: 0.35 + ? ≤ 0.35 + 0.3
d: 0.35 + ? ≤ 0.35 + 0.5
g: 0.2 + ? ≤ 0.2 + 0.4

**Approximation TA:**

A ***θ-approximation*** T' for top-k query q with $\theta > 1$
is a set T' of docs with:

- $|T'|=k$ and
- for each $d' \in T'$ and each $d'' \notin T'$: $\theta * score(q,d') \geq score(q,d'')$

Modified TA:

...

Stop when $\boldsymbol{min_k \geq aggr(high_1, ..., high_m) / \theta}$

## Pruning with Combined Authority/Similarity Scoring

(Long/Suel 2003)

Focus on **score(q,dj) = r(dj) + s(q,dj)**

with normalization r(·) ≤ a, s(·) ≤ b (and often a+b=1)

Keep index lists sorted in **descending order of „static" authority r(dj)**

***Conservative authority-based pruning:***

high(0) := max{r(pos(i)) | i=1..m}; high := high(0) + b;

high(i) := r(pos(i)) + b;

stop scanning i-th index list when high(i) < min score of top k

terminate algorithm when high < min score of top k

effective when total score of top-k results is dominated by r

***First-k' heuristics***:

scan all m index lists until k' ≥ k docs have been found

that appear in all lists;

the stopping condition is easy to check because of the sorting by r

## Text Retrieval: Limitations and Problems with Web Mining Apps

IR necessary but not sufficient for Web search !

- Doesn’t capture authority
  - An article on BBC as good as a copy on john-doe-news.com
- Doesn’t address web navigation
  - Query ibm seeks www.ibm.com
  - www.ibm.com may look less topical than a quarterly report