

Chapter 4

Text Search in a Nutshell

Sergej Sizov

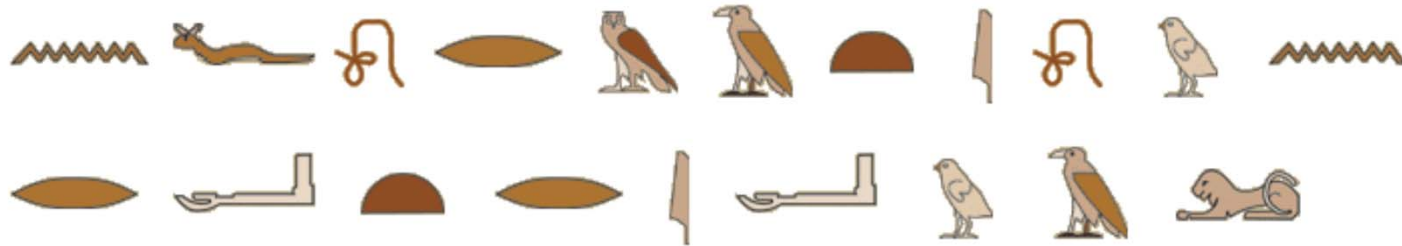
Web Retrieval

Summer term 2010

Motivation

Information Retrieval

a)



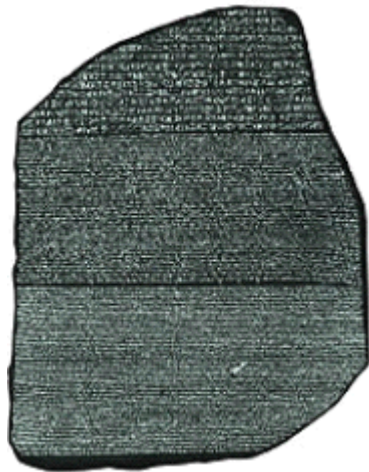
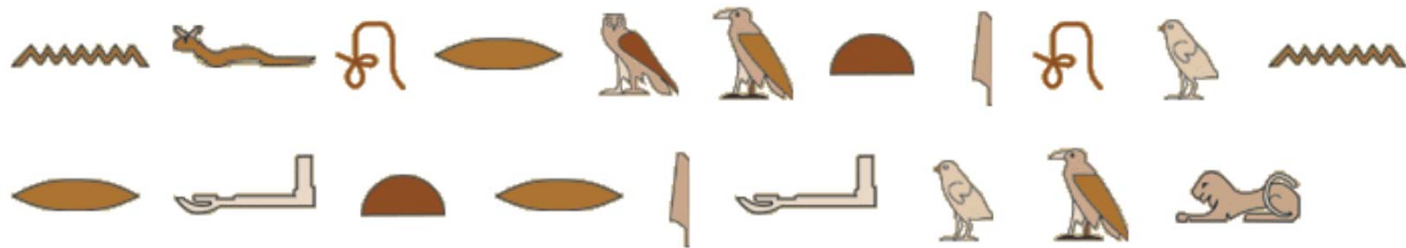
b)



c)

73 110 102 111 114 109 97 116 105 111 110
82 101 116 114 105 101 118 97 108

Motivation (2)

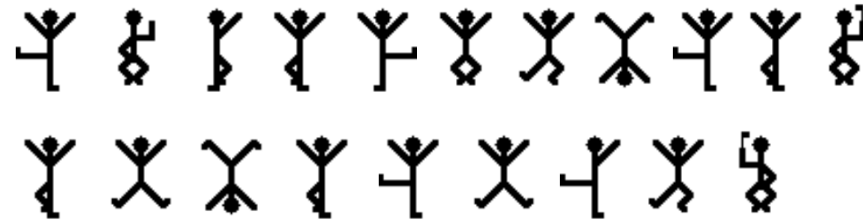


Stein von Rosetta

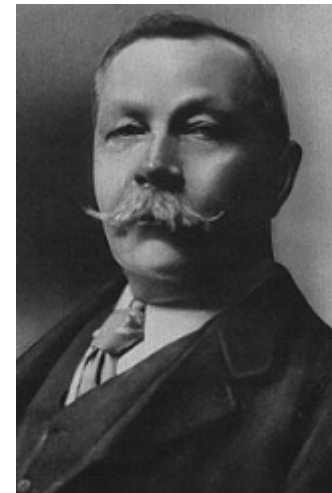


Jean-François Champollion

Motivation (3)



Sherlock Holmes



Arthur Conan Doyle

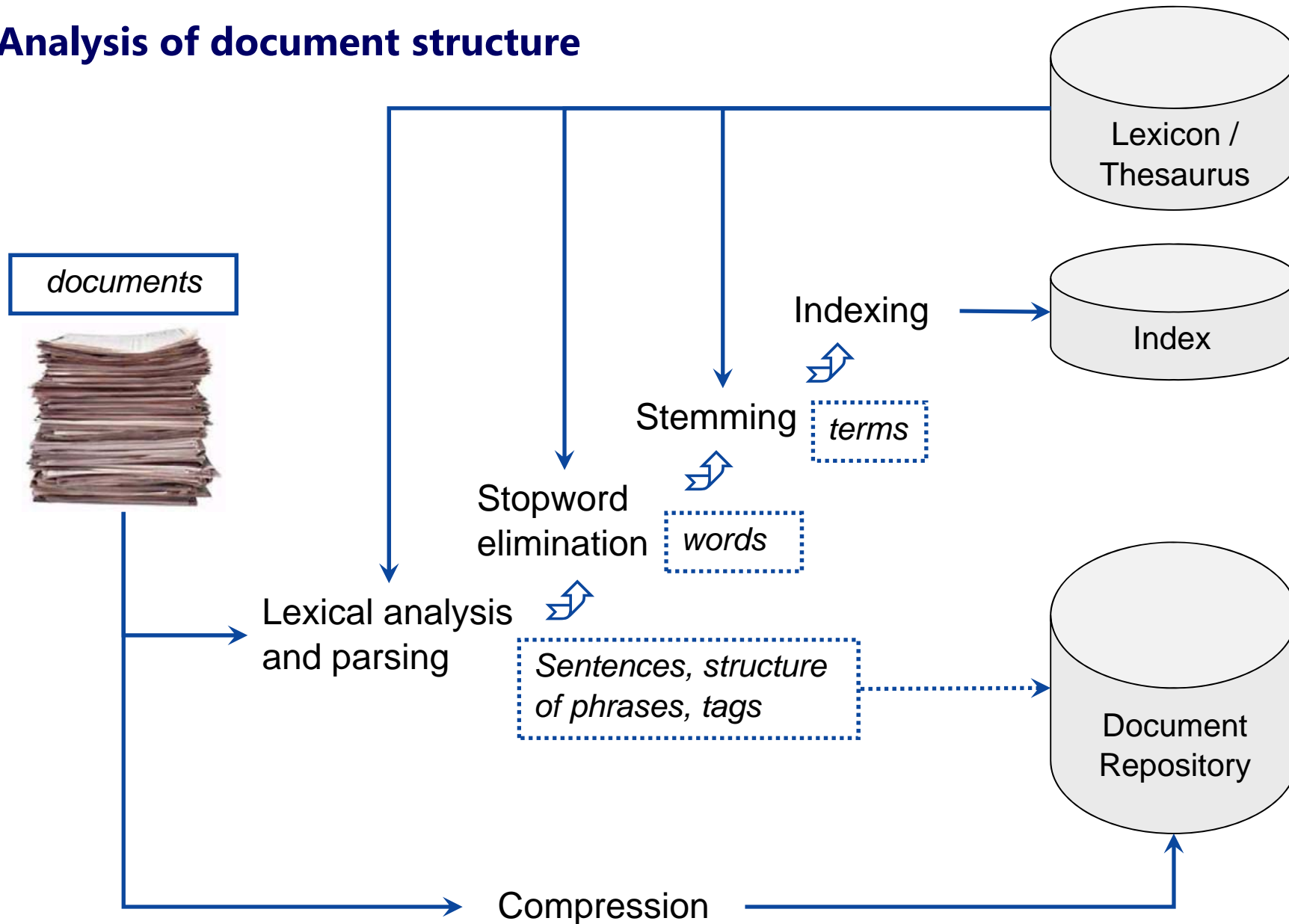
Motivation (4)

73 110 102 111 114 109 97 116 105 111 110
82 101 116 114 105 101 118 97 108



Text Processing: an overview

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.

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.

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)

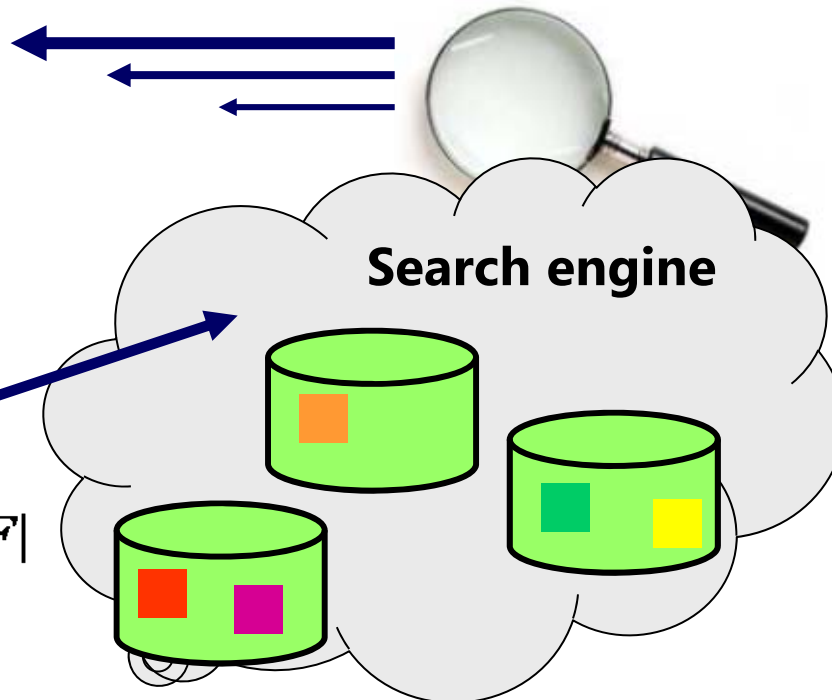
Example: WordNet, <http://www.cogsci.princeton.edu/~wn>

Basic Principles:

- Feature Space: words in documents are reduced to terms.
- Document model: each document is represented as vector in $[0,1]^{|\mathcal{F}|}$ whereby d_{ij} is the weight of the j -th term in d_i .
- Queries: queries are vectors q_i in $[0,1]^{|\mathcal{F}|}$
- Relevance: relevance of results is based on similarity function for vector space $[0,1]^{|\mathcal{F}|}$
- 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

Ranking by descending relevance



Similarity metric:

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

Query $q \in [0,1]^{|F|}$
(Set of weighted features)

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

e.g., using:

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

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

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

$$idf_i = \log \frac{N}{df_i}$$

- ◆ common combination:
(*tf***idf* formula)

$$w_{ij} = \frac{tf_{ij}}{\max_k tf_{kj}} \log \frac{N}{df_i}$$

$$d_{ij} = \frac{w_{ij}}{\sqrt{\sum_k w_{kj}^2}}$$

Term Weighting in Queries

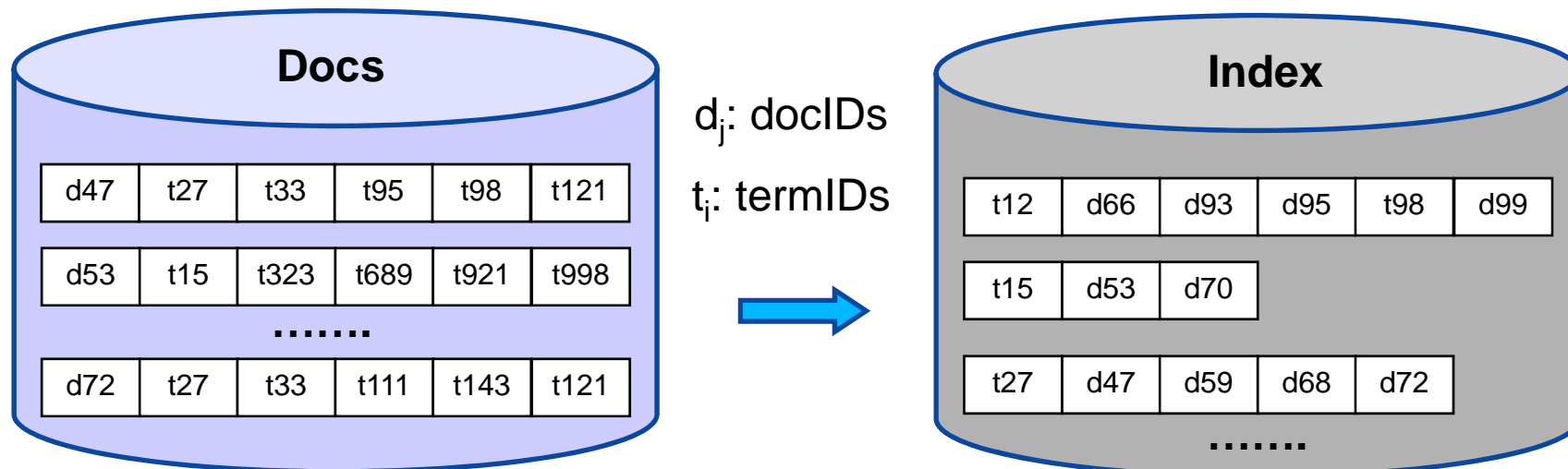
Depending of query interface and user category, simple or advanced term weightings may be used

- ◆ 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

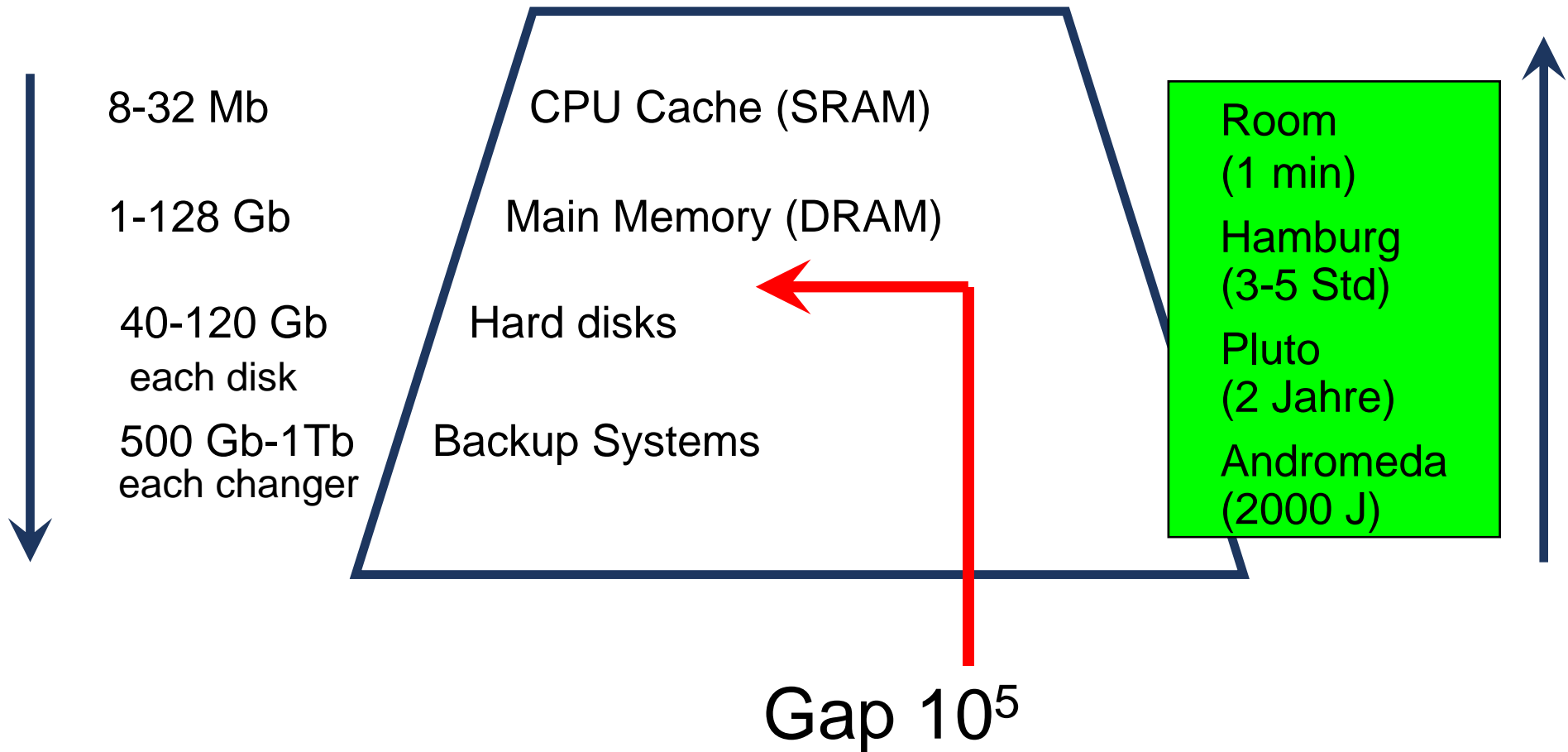
Exact-Match-Suche: search (key) returns ids

Bereichssuche: search (lowkey, highkey) returns ids

Präfixstringsuche: search (prefix) returns ids

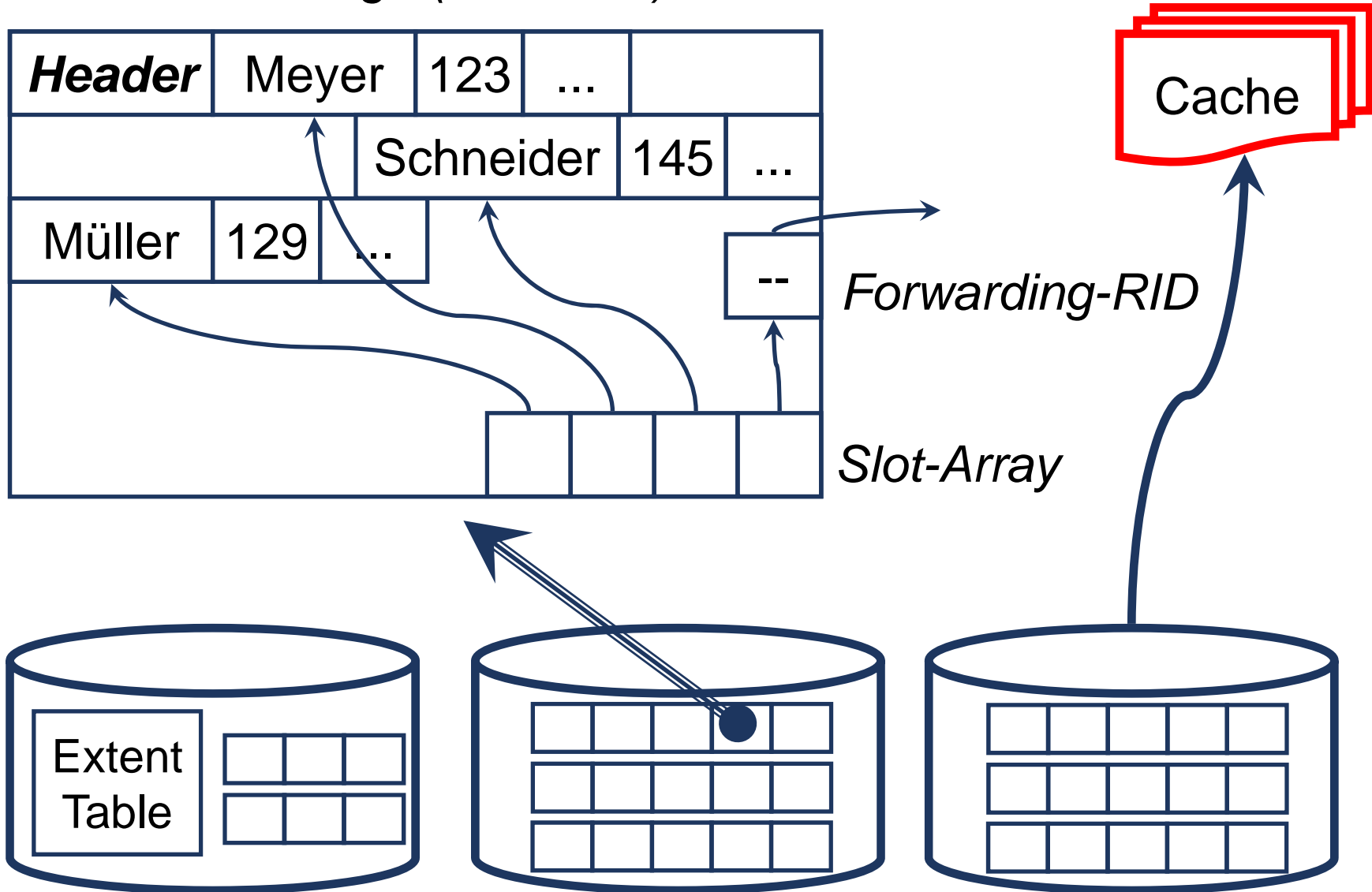
bei dynamischen Updates

Properties of Media Types

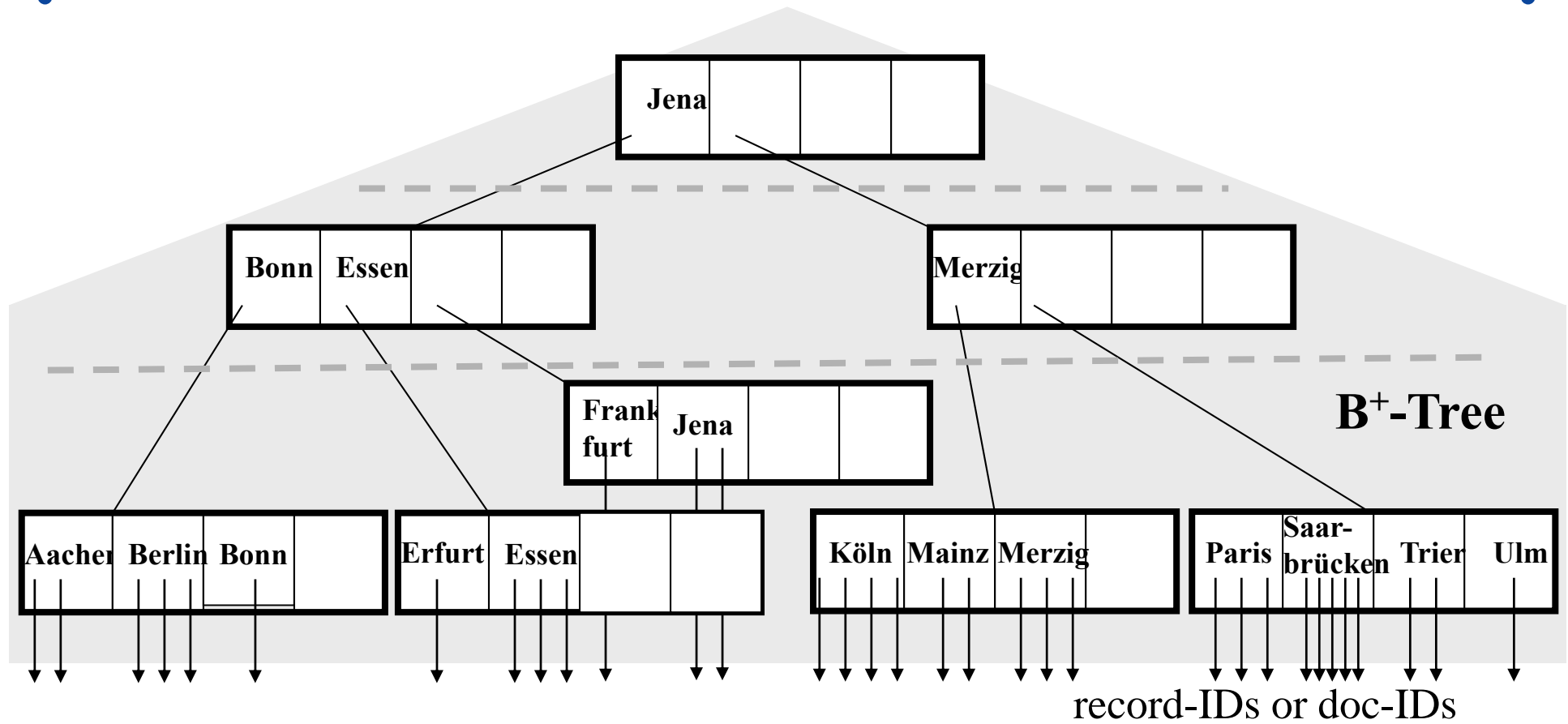


Access: Physical Data Organization

Database Page (32-64 Kb)



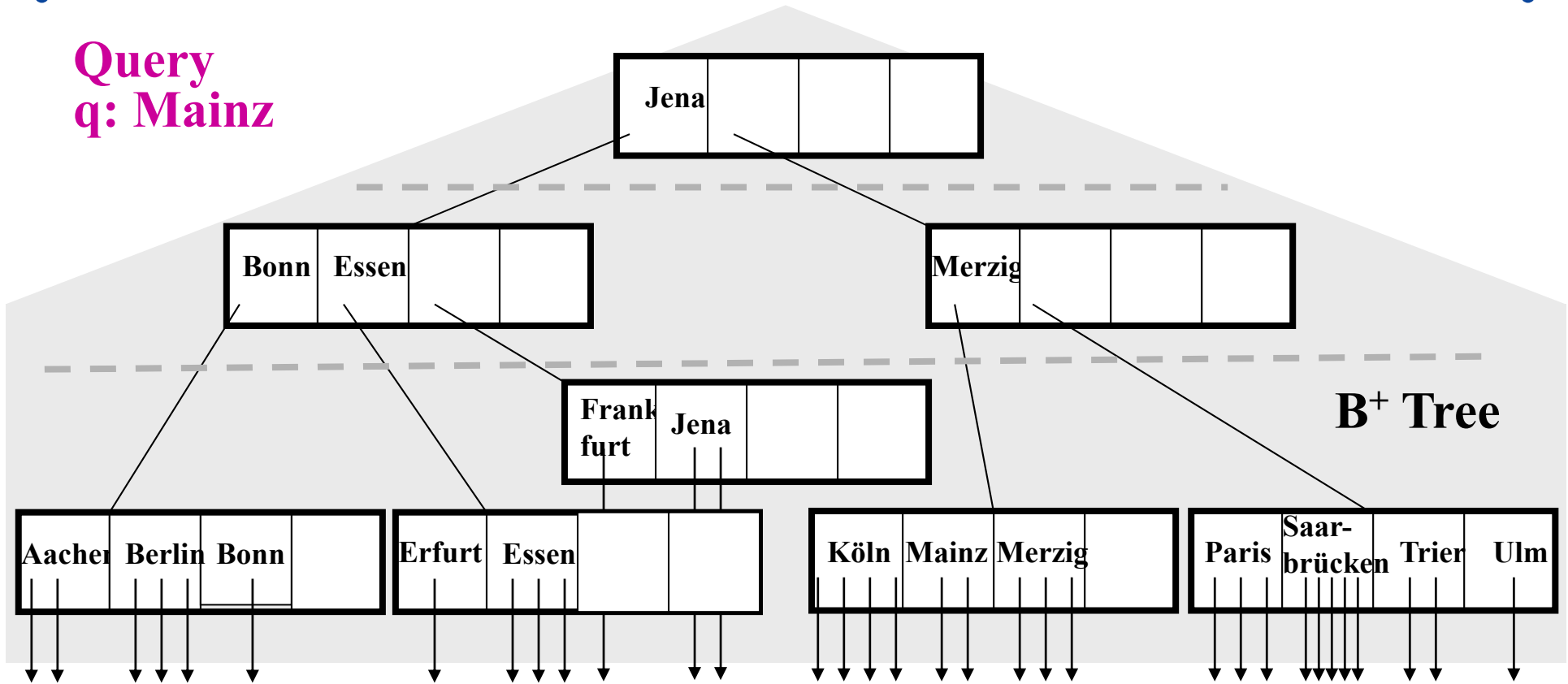
B⁺ Trees: Example



B+ Trees: Lookup (1)



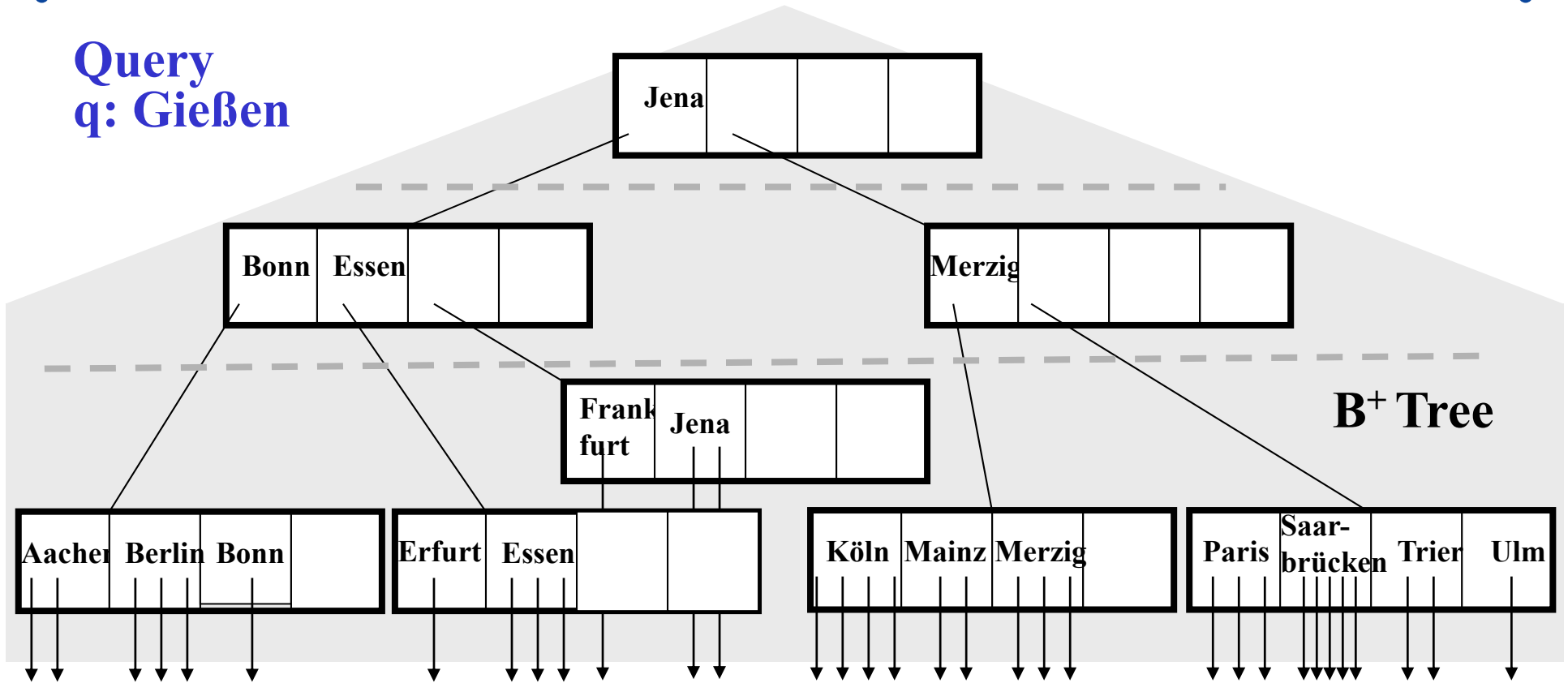
Query
q: Mainz



B+ Trees: Lookup (2)



Query
q: Gießen



DBS-Style Top-k Query Processing

q: professor
research
xml

B+ tree on terms

professor ... research ... xml

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

17: 0.3
44: 0.4
52: 0.1
53: 0.8
55: 0.6
⋮

12: 0.5
14: 0.4
28: 0.1
44: 0.2
51: 0.6
52: 0.3
⋮

11: 0.6
17: 0.1
28: 0.7
⋮

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

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

Find: top k results w.r.t. $\text{score}(q,d) = \text{aggr}\{s_i(d)\}$ (e.g.: $\sum_{i \in q} s_i(d)$)

Naive join&sort QP algorithm:

top-k (

$\sigma[\text{term}=t_1]$ (index)	×	DocId
$\sigma[\text{term}=t_2]$ (index)	×	DocId
...	×	DocId
$\sigma[\text{term}=t_z]$ (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

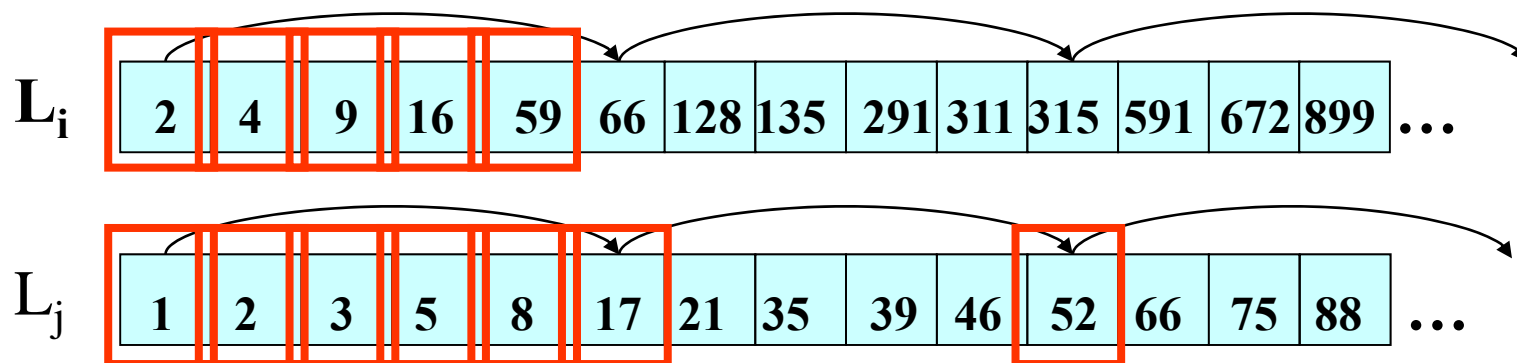
Candidate results need to be looked up in other lists $L(j)$

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

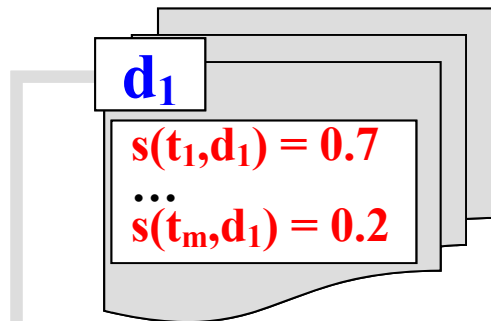


Efficient Top-k Search

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

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

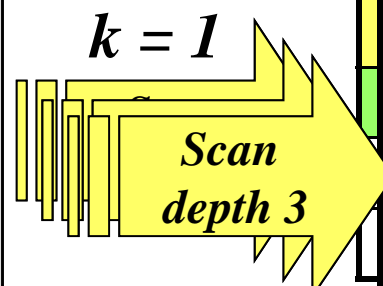
Data items: d_1, \dots, d_n



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

	Index lists					
t_1	d78 0.9	d23 0.8	d10 0.8	d1 0.7	d88 0.2	...
t_2	d64 0.8	d23 0.6	d10 0.6	d10 0.2	d78 0.1	...
t_3	d10 0.7	d78 0.5	d64 0.4	d99 0.2	d34 0.1	...

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



Rank	Doc	Worst-score	Best-score
1	d10	2.1	2.1
2	d78	1.4	2.0
3	d23	0.6	1.8
4	d99	0.2	2.0

STOP!

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  $d_j$  at position  $pos_i$  in  $L_i$ ;
   $high_i := s_i(d_j)$ ;
  if  $d_j \notin \text{top-k}$  then {
    look up  $s_v(d_j)$  in all lists  $L_v$  with  $v \neq i$ ; // random access
    compute  $s(d_j) := \text{aggr} \{s_v(d_j) \mid v=1..m\}$ ;
    if  $s(d_j) > \text{min score among top-k}$  then
      add  $d_j$  to top-k and remove min-score  $d$  from top-k; }
   $\text{threshold} := \text{aggr} \{high_v \mid v=1..m\}$ ;
  if  $\text{min score among top-k} \geq \text{threshold}$  then exit;
```

*but random accesses
are expensive !*

$m=3$
aggr: sum
 $k=2$

f: 0.5
b: 0.4
c: 0.35
a: 0.3
h: 0.1
d: 0.1

a: 0.55
b: 0.2
f: 0.2
g: 0.2
c: 0.1

h: 0.35
d: 0.35
b: 0.2
a: 0.1
c: 0.05
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 d_j at position pos_i in L_i ;

$E(d_j) := E(d_j) \cup \{i\}$; $high_i := si(q, d_j)$;

$bestscore(d_j) := aggr\{x_1, \dots, x_m\}$

with $x_i := si(q, d_j)$ for $i \in E(d_j)$, $high_i$ for $i \notin E(d_j)$;

$worstscore(d_j) := aggr\{x_1, \dots, x_m\}$

with $x_i := si(q, d_j)$ for $i \in E(d_j)$, 0 for $i \notin E(d_j)$;

$top-k := k$ docs with largest $worstscore$;

$threshold := bestscore\{d \mid d \text{ not in } top-k\}$;

if $\min worstscore \text{ among } top-k \geq threshold$ then exit;

$m=3$
aggr: sum
 $k=2$

f: 0.5
b: 0.4
c: 0.35
a: 0.3
h: 0.1
d: 0.1

a: 0.55
b: 0.2
f: 0.2
g: 0.2
c: 0.1

h: 0.35
d: 0.35
b: 0.2
a: 0.1
c: 0.05
f: 0.05

top-k:

a: 0.95

b: 0.8

candidates:

f: $0.7 + ? \leq 0.7 + 0.1$

h: $0.35 + ? \leq 0.35 + 0.5$

c: $0.35 + ? \leq 0.35 + 0.3$

d: $0.35 + ? \leq 0.35 + 0.5$

g: $0.2 + ? \leq 0.2 + 0.4$



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 $d'' \notin T'$: $\theta * \text{score}(q, d') \geq \text{score}(q, d'')$

Modified TA:

...

Stop when $\min_k \geq \text{aggr}(\text{high}_1, \dots, \text{high}_m) / \theta$

Pruning with Combined Authority/Similarity Scoring

(Long/Suel 2003)

Focus on $\text{score}(q,d_j) = r(d_j) + s(q,d_j)$

with normalization $r(\cdot) \leq a$, $s(\cdot) \leq b$ (and often $a+b=1$)

Keep index lists sorted in **descending order of „static“ authority $r(d_j)$**

Conservative authority-based pruning:

$\text{high}(0) := \max\{r(\text{pos}(i)) \mid i=1..m\}$; $\text{high} := \text{high}(0) + b$;

$\text{high}(i) := r(\text{pos}(i)) + b$;

stop scanning i -th index list when $\text{high}(i) < \text{min score of top } k$

terminate algorithm when $\text{high} < \text{min score of top } k$

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

First- k' heuristics:

scan all m index lists until $k' \geq k$ 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