Chapter 2

Text Search in a Nutshell

Sergej Sizov

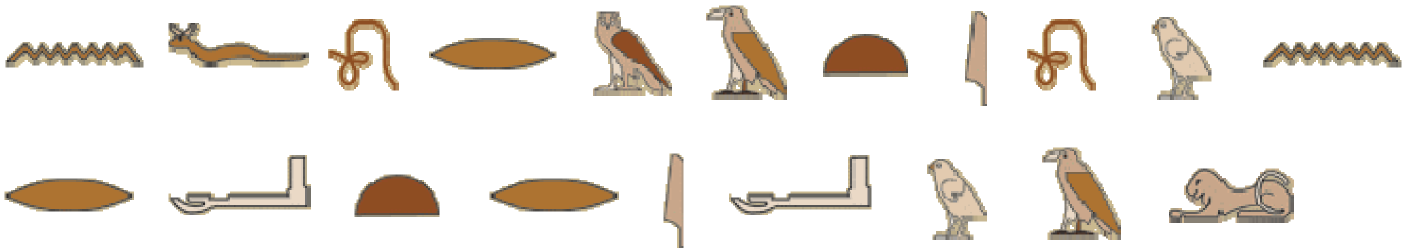
Information Retrieval

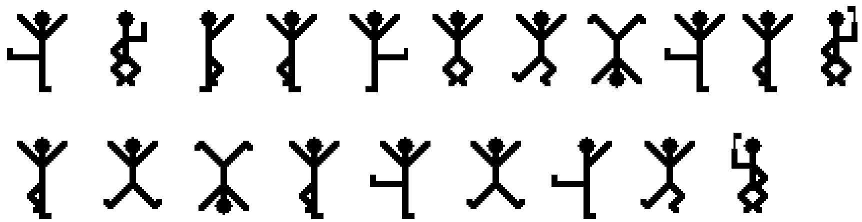
Summer term 2009

UNIVERSITÄT KOBLENZ · LANDAU

Motivation

Information Retrieval

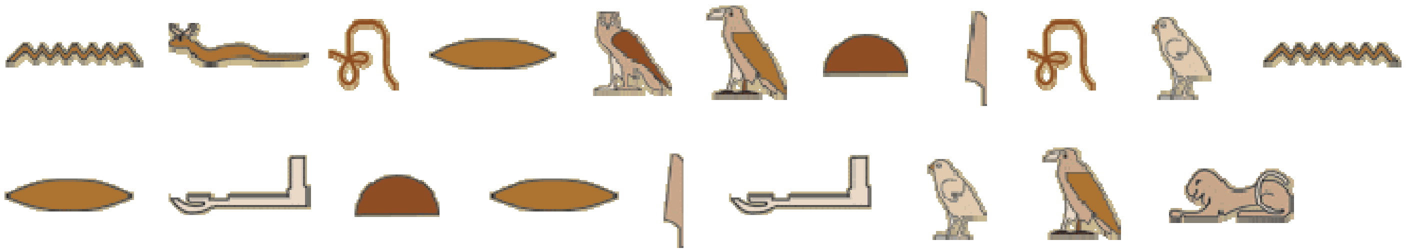
a) 

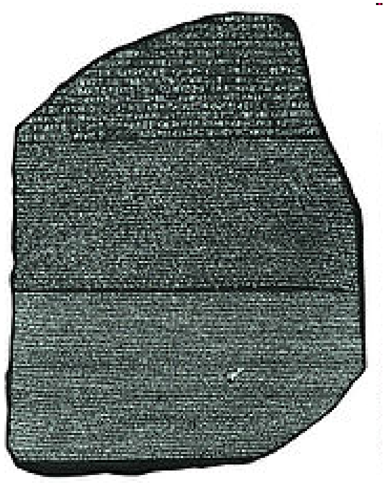
b) 

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

82 101 116 114 105 101 118 97 108

Motivation (2)

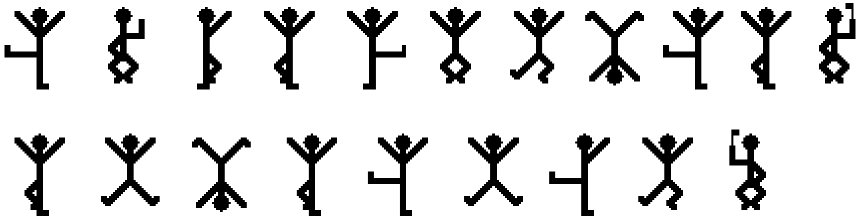




Stein von Rosetta

Jean-François Champollion

Motivation (3)



Sherlock Holmes **Arthur Conan Doyle**

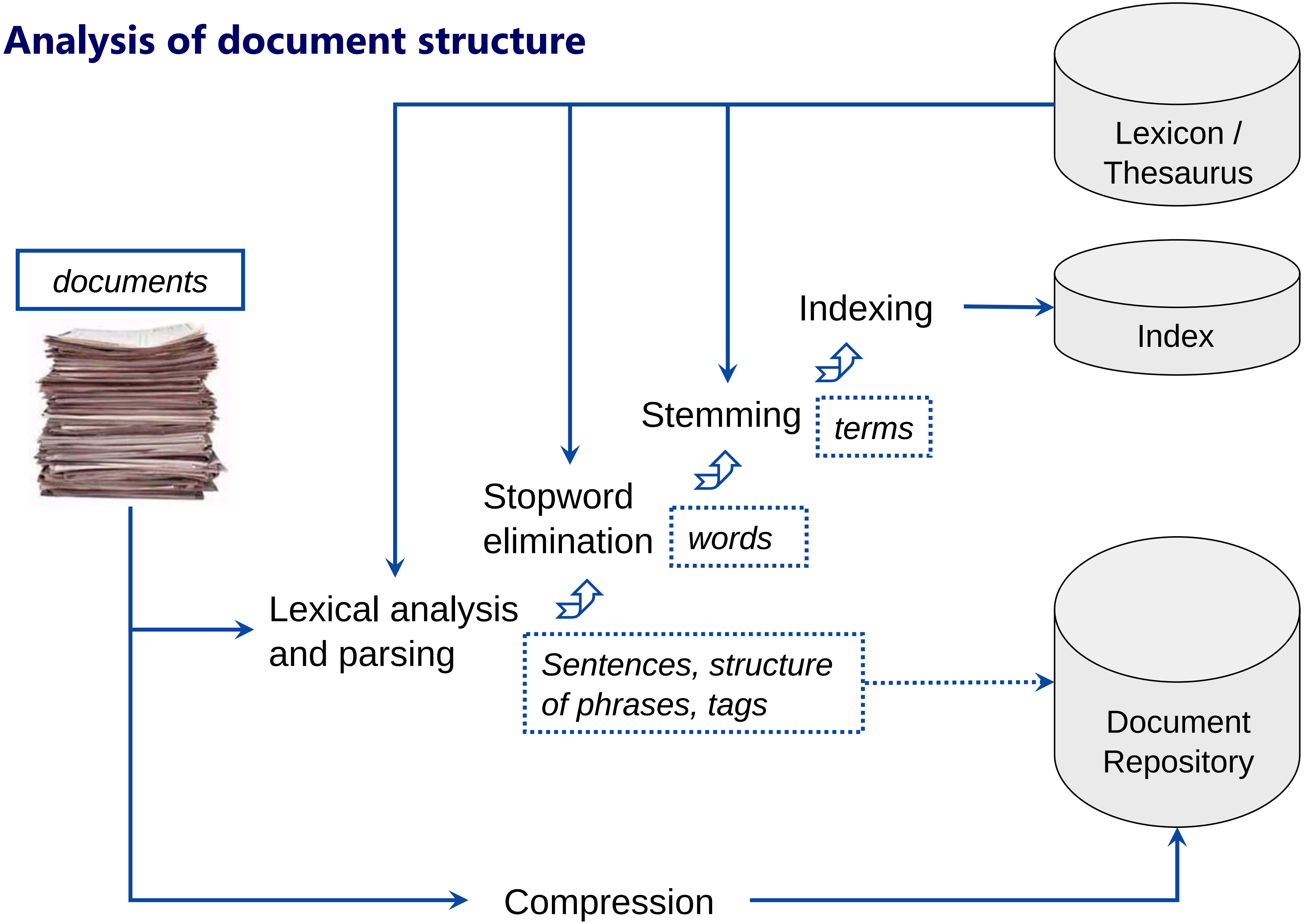
Motivation (4)

73 110 102 111 114 109 97 116 105 111 110

82 101 116 114 105 101 118 97 108

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.

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:

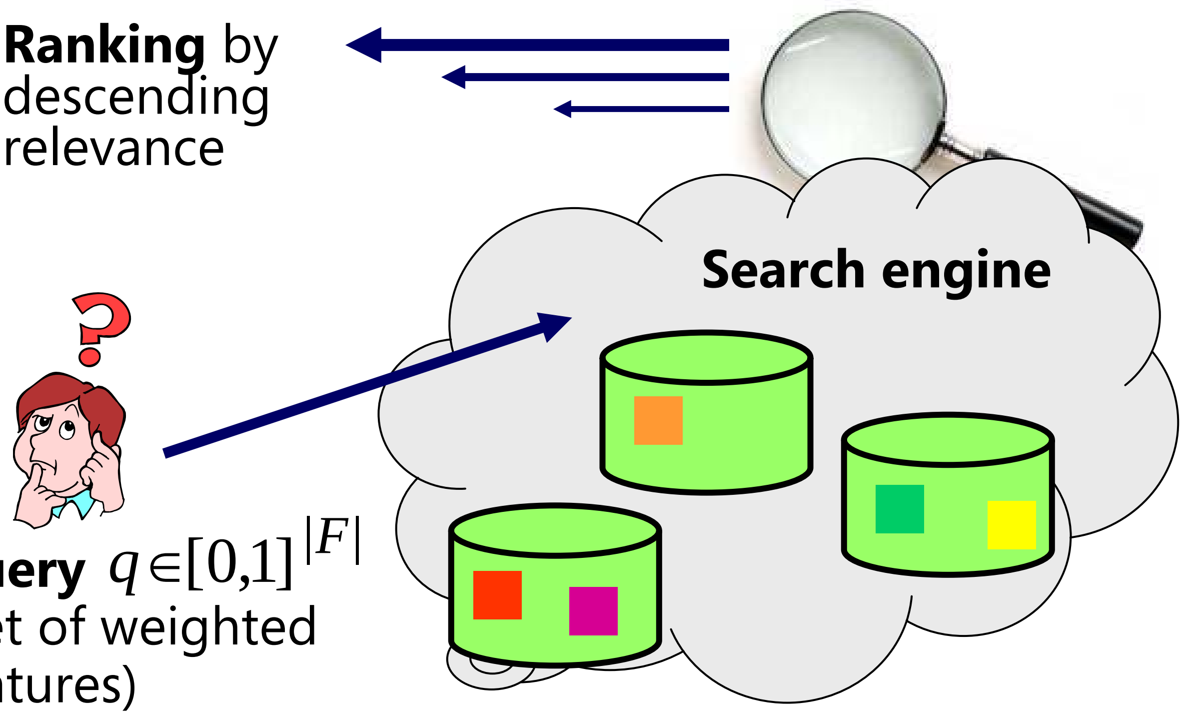
- Feature Space: words in documents are reduced to terms.
- Document model: each document is represented as vector in $[0,1]^{|F|}$

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

Ranking by descending relevance



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

Similarity metric:

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

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

e.g., using:

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

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

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: $w_{ij} = \frac{1}{k}$

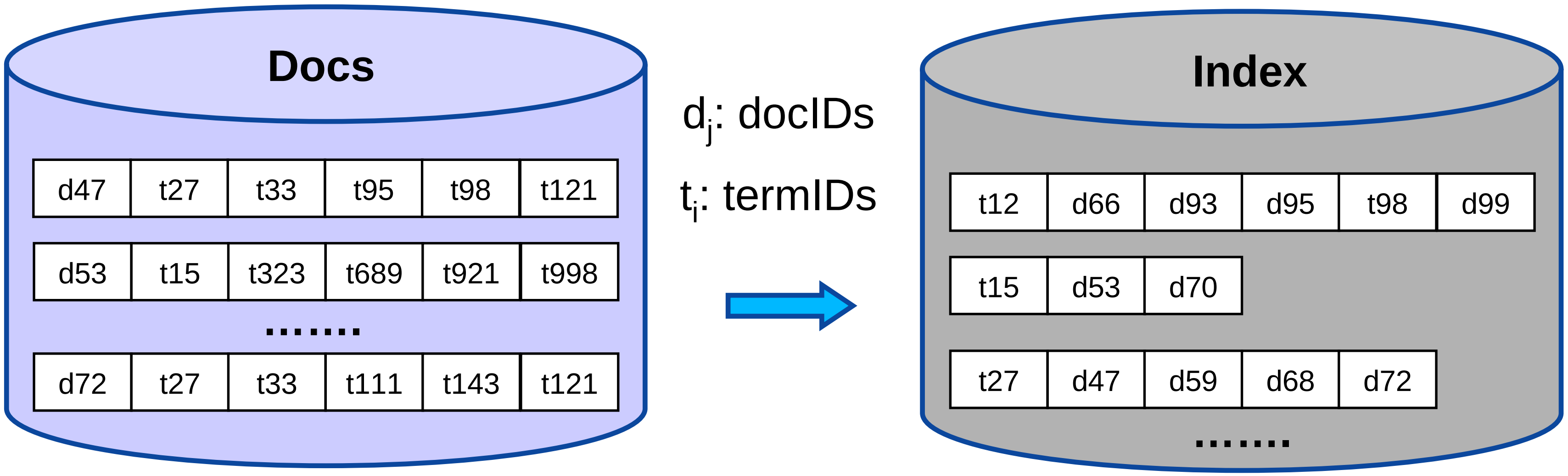
  (when conjunctive query q contains k terms and t_i is in k^{th} position)

Text Indexing and Query Execution

Konzeptionell:

invertierte Dateien (invertierte Listen) mit binärer Suche

nach Suchschlüsseln (Felder von Records, Strings in Texten)

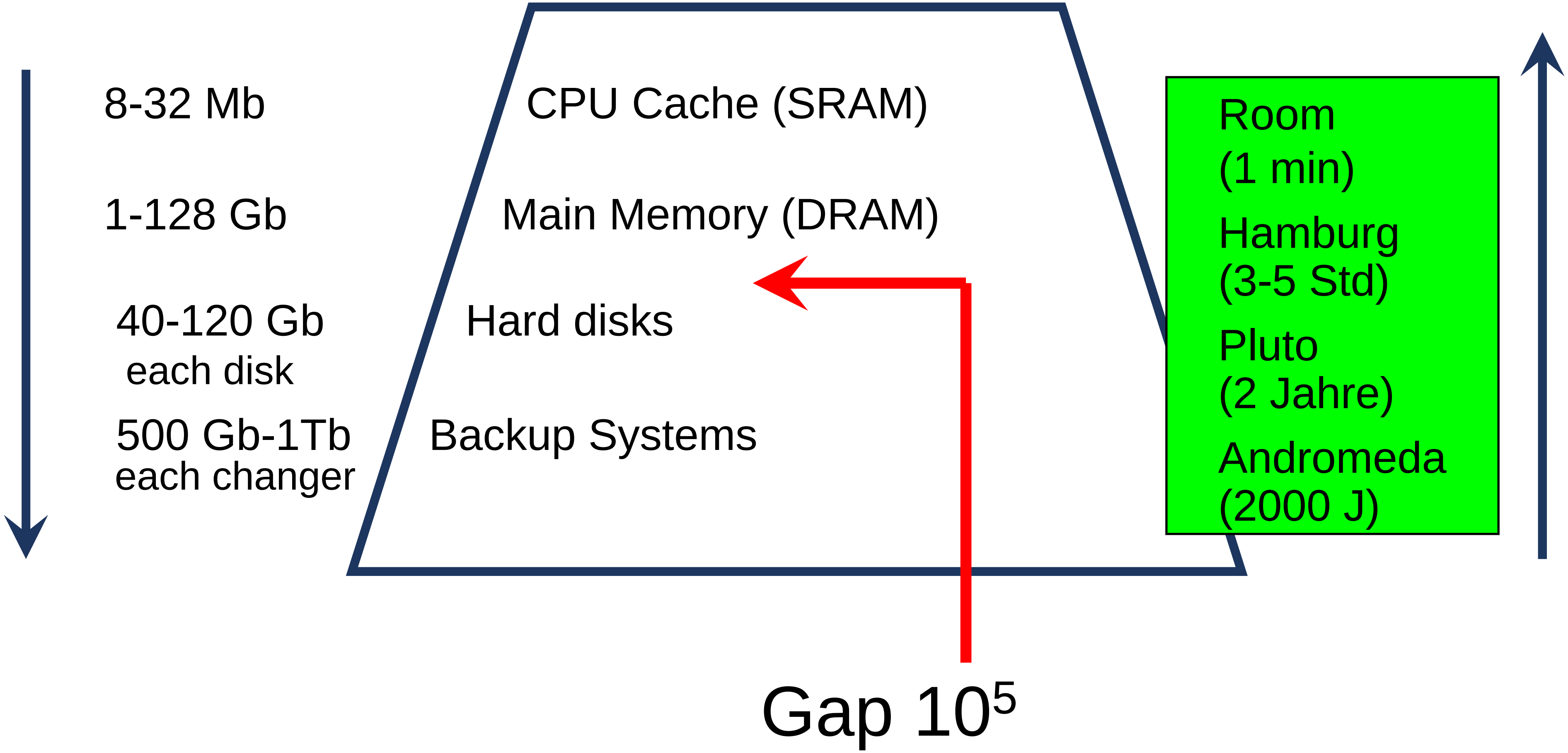


Problem:
Speicherungsorganisation in Plattenblöcken (pagination) und effiziente Implementierung der (binären) Suche für

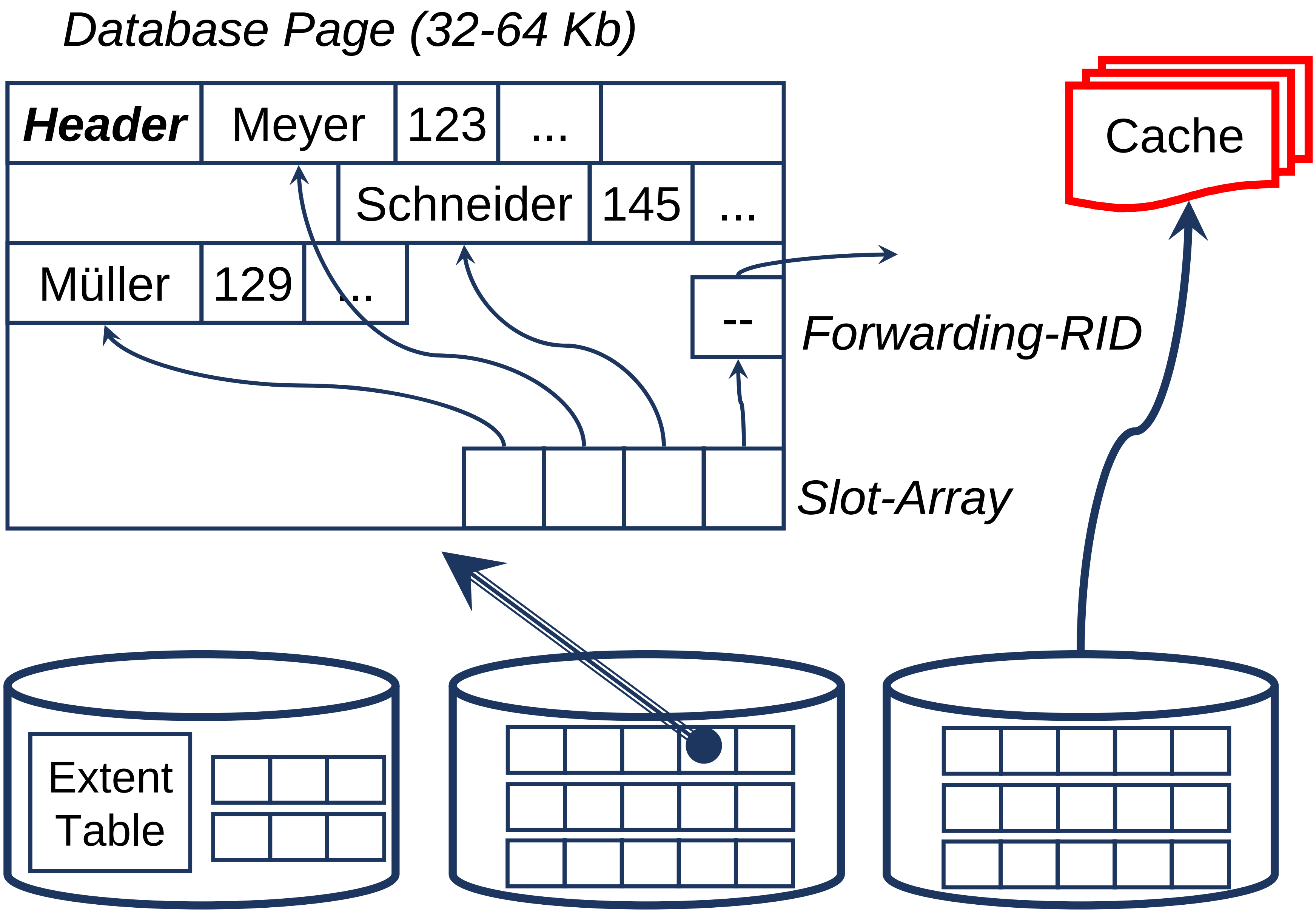
- Exact-Match-Suche: search (key) returns ids
- Bereichssuche: search (lowkey, highkey) returns ids
- Präfixstringsuche: search (prefix) returns ids

bei dynamischen Updates

Properties of Media Types



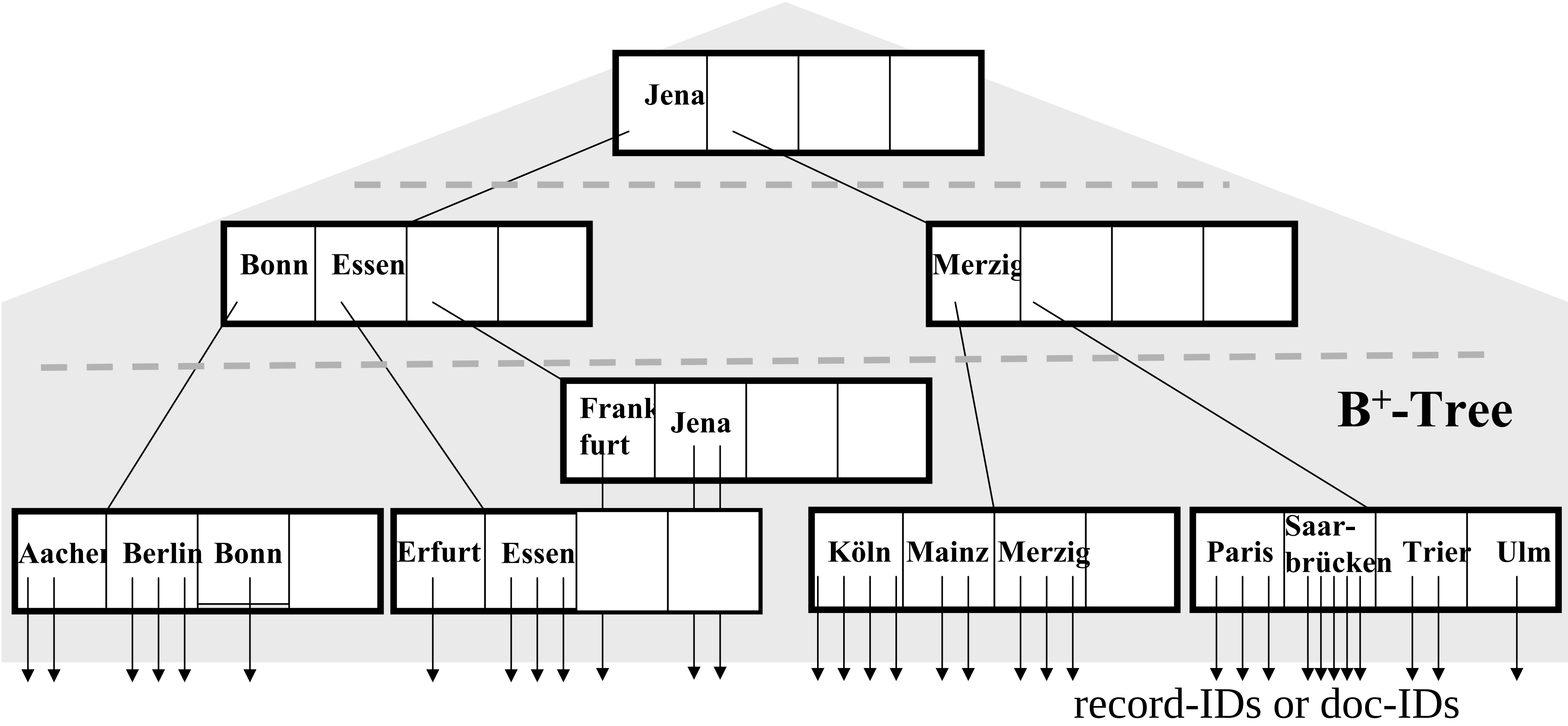
Access: Physical Data Organization



B^+ Trees: Seitenstrukturierte Mehrwegbäume

- Hohler Mehrwegebaum mit hohem Fanout ($\Rightarrow$ kleiner Tiefe)
- Knoten = Seite auf Platte
- Knoteninhalt:
  - (Sohnzeiger, Schlüssel)-Paare in inneren Knoten
  - Schlüssel (mit weiteren Daten) in Blättern
- perfekt balanciert: alle Blätter haben dieselbe Distanz zur Wurzel
- Sucheffizienz $O(\log_k n/C)$ Seitenzugriffe (Platten-I/Os)
  bei n Schlüsseln, Seitenkapazität C und Fanout k
  pro Baumniveau: bestimme kleinsten Schlüssel $\geq q$ und
  suche weiter im Teilbaum links von q
- Kosten einer Einfüge- oder Löschoperation $O(\log_k n/C)$
- mittlere Speicherplatzauslastung bei zufälligem Einfügen: $\ln 2 \approx 0.69$

B+ Trees: Example



B^+ Tree: Definition

Ein Mehrwegbaum heißt B*-Baum (eng.: B^+ Tree) der Ordnung (m, m*), wenn gilt:

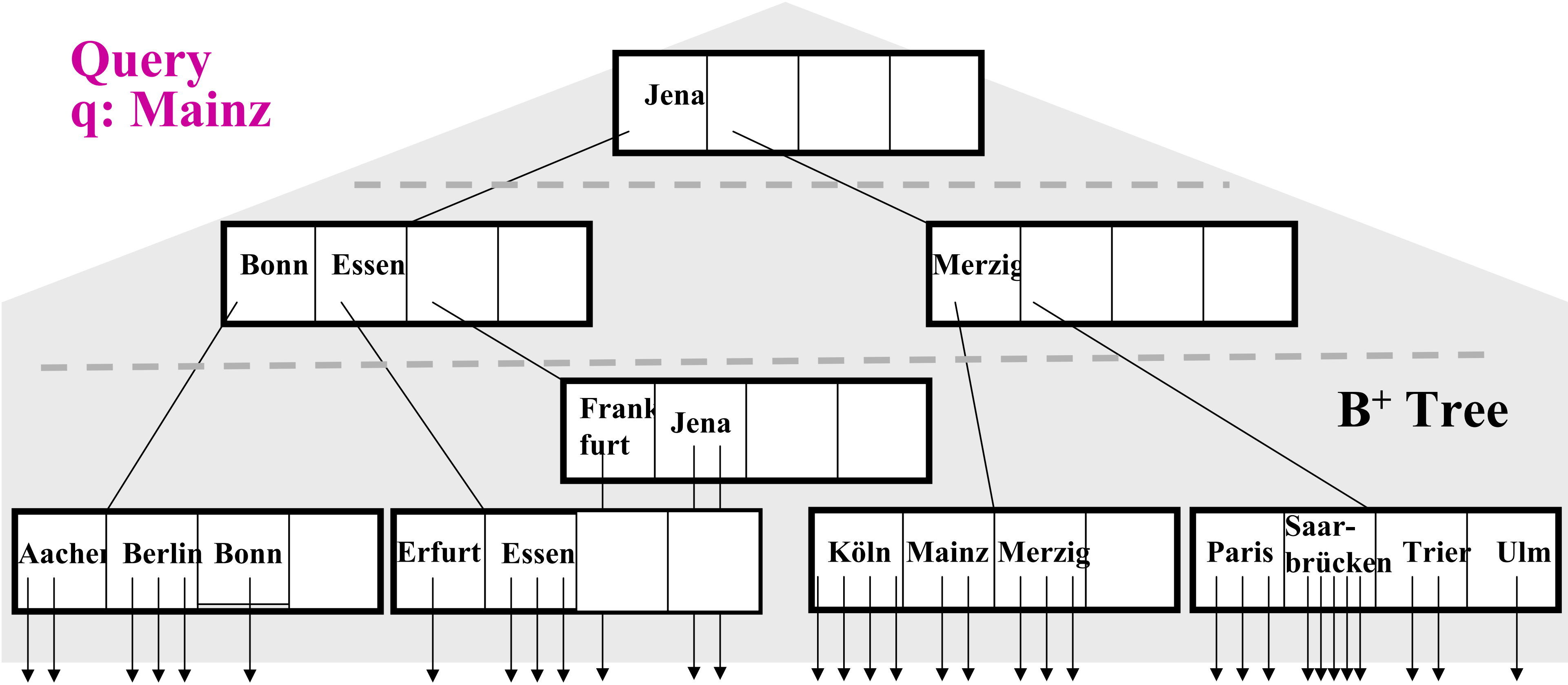
- Jeder Nichtblattknoten außer der Wurzel enthält mindestens $m \geq 1$ und höchstens 2m Schlüssel (Wegweiser).
- Ein Nichtblattknoten mit k Schlüssel x1, ..., xk hat genau k+1 Söhne t1, ..., t(k+1), so daß
  - für alle Schlüssel s im Teilbaum ti ($2 \leq i \leq k$) gilt $x(i-1) < s \leq xi$ und
  - für alle Schlüssel s im Teilbaum t1 gilt $s \leq x1$ und
  - für alle Schlüssel im Teilbaum t(k+1) gilt $xk < s$.
- Alle Blätter haben dasselbe Niveau (Distanz von der Wurzel)
- Jedes Blatt enthält mindestens $m^* \geq 1$ und höchstens 2m* Schlüssel.

Achtung: Implementierungen verwenden **variabel lange Schlüssel** und eine Knotenkapazität in Bytes statt Konstanten 2m und 2m*

Sonderfall m=m*=1: **2-3-Bäume** als Hauptspeicherdatenstruktur

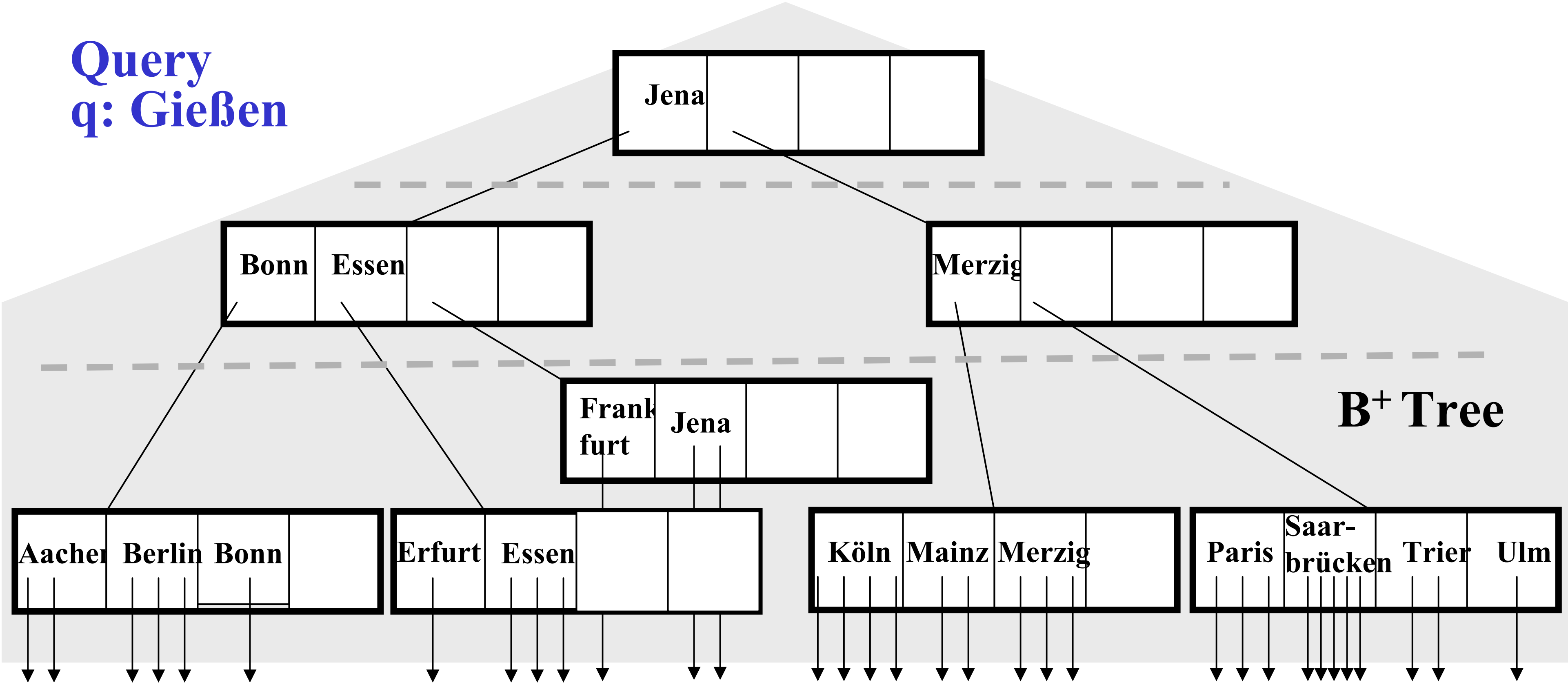
B⁺ Trees: Lookup (1)

Query
q: Mainz



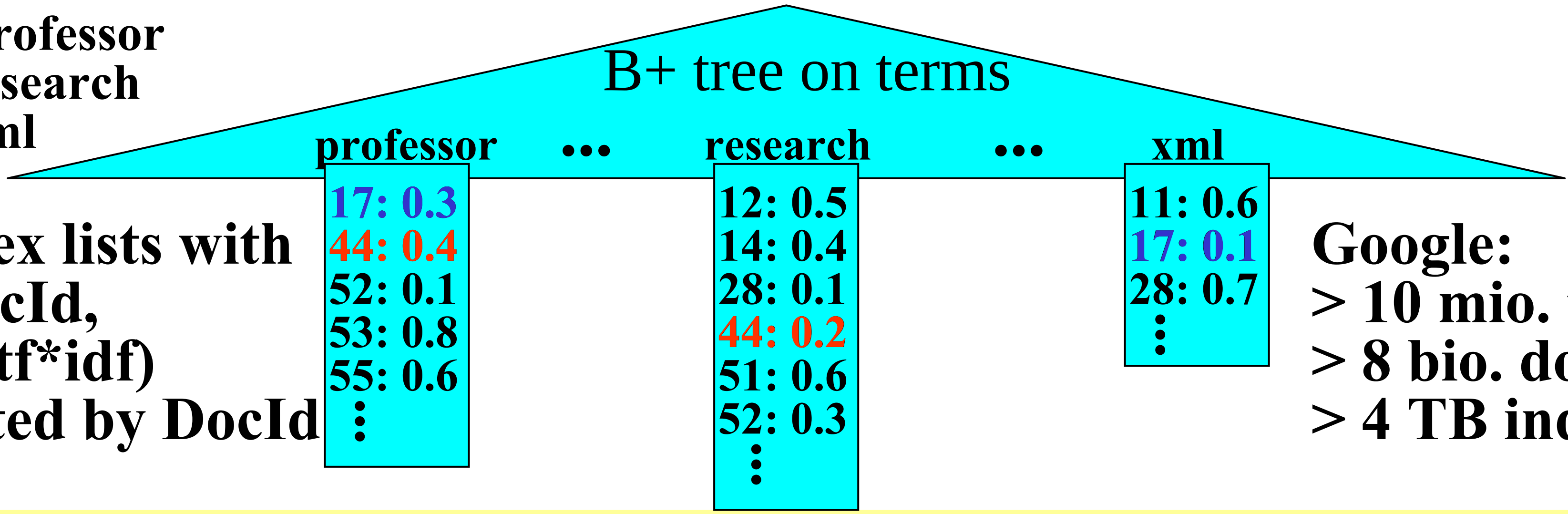
B+ Trees: Lookup (2)

Query
q: Gießen



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

Given: 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 $q_i \neq 0$

Find: 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:

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

Index List Processing by Merge Join

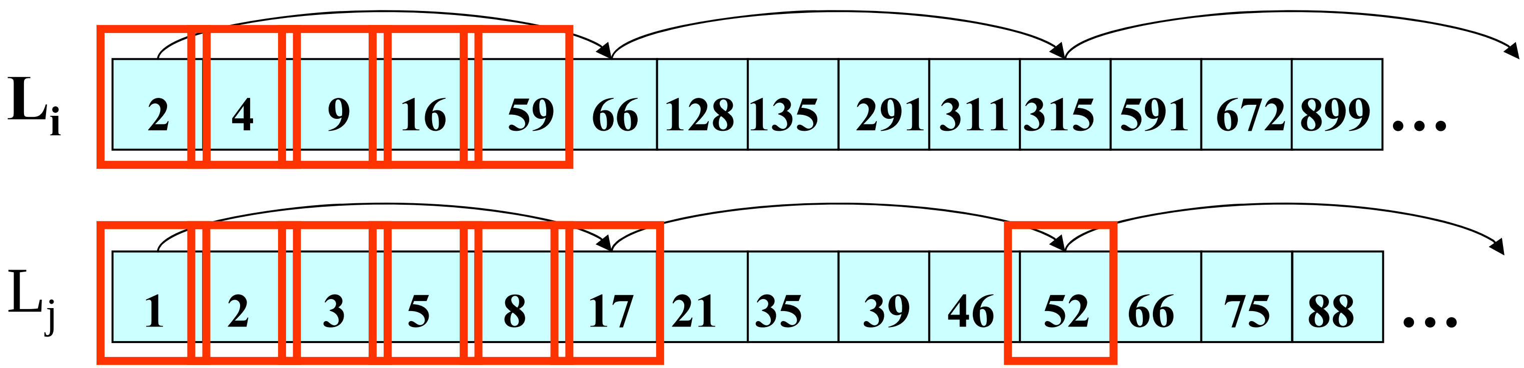
Keep L(i) in **ascending order of doc ids**

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

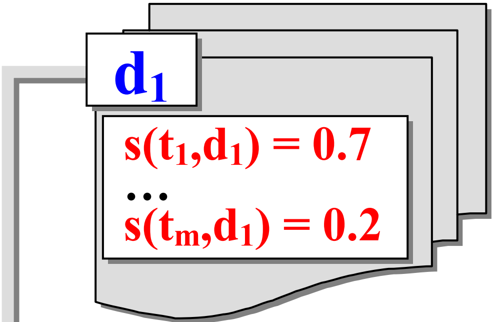


Efficient Top-k Search

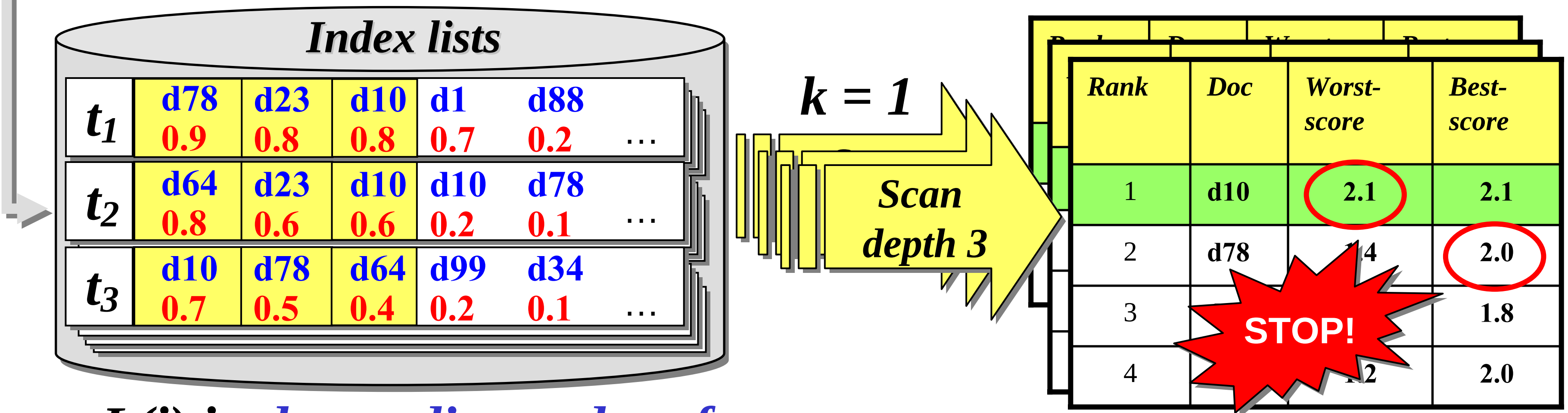
[Buckley85, Güntzer/Balke/Kießling 00, Fagin01]

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

Data items: $d_1, \ldots, d_n$



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



<u>TA with sorted access only (NRA)</u>:
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

Evaluation of Result Quality: Basic Measures

ideal measure is user satisfaction !
heuristically approximated by benchmarking measures
(on test corpora with query suite and relevance assessment by experts)

Capability to return **only** relevant documents:

$$\textbf{Precision (Präzision)} = \frac{\#\ relevant\ docs\ among\ top\ r}{r}$$

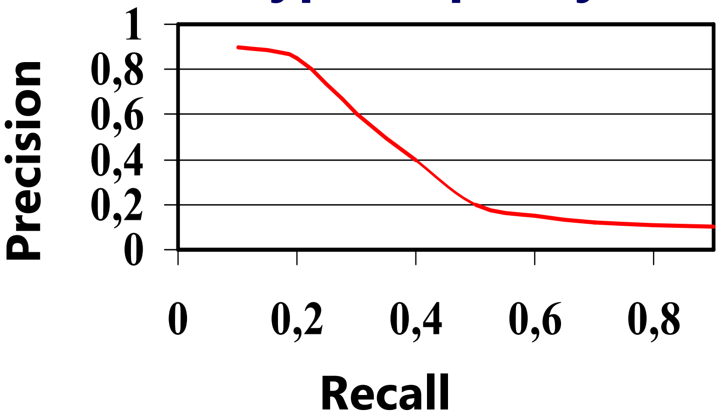
typically for
r = 10, 100, 1000

Capability to return **all** relevant documents:

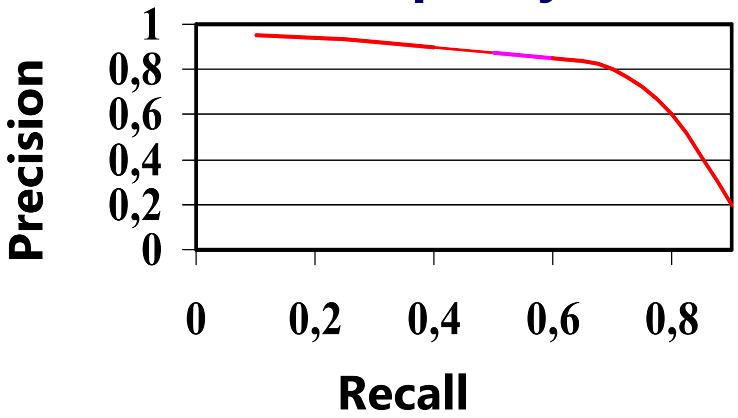
$$\textbf{Recall (Ausbeute)} = \frac{\#\ relevant\ docs\ among\ top\ r}{\#\ relevant\ docs}$$

typically for
r = corpus size

Typical quality



Ideal quality



Evaluation of Result Quality: Aggregated Measures

Combining precision and recall into **F measure**
(e.g. with α=0.5 harmonic mean **F_1**):

$$F = \frac{1}{\alpha \frac{1}{precision} + (1-\alpha)\frac{1}{recall}}$$

Precision-recall breakeven point of query q:
point on precision-recall curve p = f(r) with p = r

for a set of n queries q1, ..., qn   (e.g. TREC benchmark)

Macro evaluation
(user-oriented) =
of precision

$$\frac{1}{n}\sum_{i=1}^{n} precision(qi)$$

Micro evaluation
(system-oriented) =
of precision

$$\frac{\sum_{i=1}^{n} \#\ relevant\ \&\ found\ docs\ \ for\ qi}{\sum_{i=1}^{n} \#\ \ found\ docs\ \ for\ qi}$$

analogous
for recall
and F1

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

Problems with common IR-style evaluation on the Web

Collection is dynamic

- 10-20% urls change every month

Queries are time sensitive

- Topics are hot then they are not

Spam methods evolve

- Algorithms evaluated against last month's web may not work today

Need to keep the collection fresh

Need to keep the queries fresh

Search space is extremely large

Over 100 million unique queries a day

To measure a 5% improvement at 95% confidence level:

- .. one would need **2700** judged queries