universität koblenz
weiter:denken

Web Retrieval
Internal Data Storage / Underlying Models (I)

Frank Hopfgartner
Institute for Web Science and Technologies

Recap

universität koblenz
weiter:denken

- What is Information Retrieval (IR)?
- What makes Web Information Retrieval (WIR) special?
- What are the essential components of IR?
- What is relevance?
- How to evaluate an IR system?
- How to make a corpus for an IR system?
- What is the difference between the various evaluation metrics?

Intended Learning Outcomes

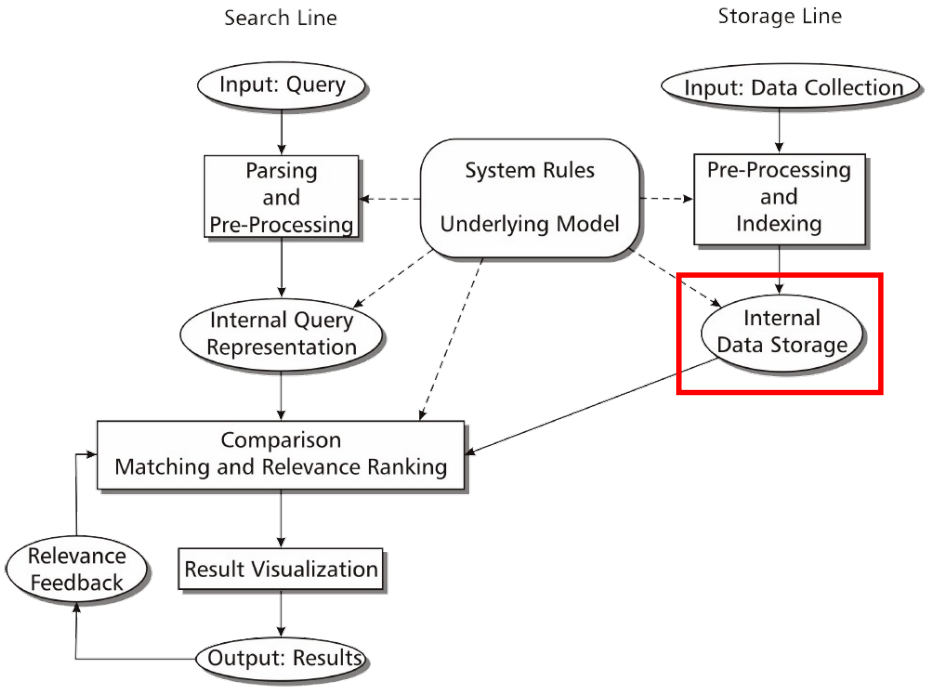
universität koblenz
weiter:denken

At the end of this lecture, you will be able to

- describe the Boolean model
- compose a T-D matrix
- outline the advantages of an an inverted index
- explain how to enable phrase queries and wildcard searches

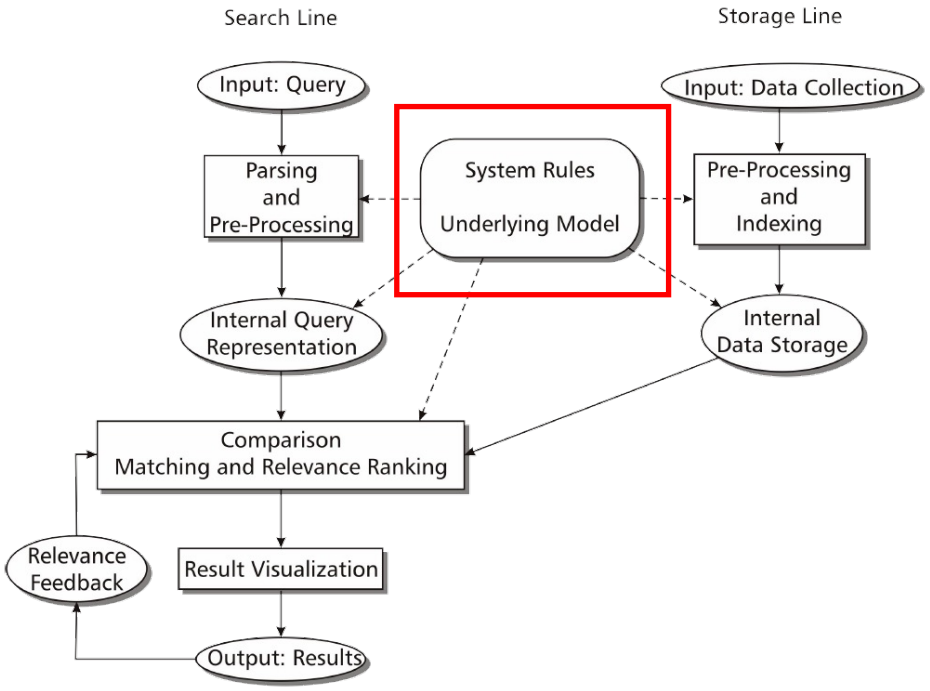
Internal data storage

- Store the data so that
  - its content can be described accurately
  - efficient access is guaranteed
  - storage space is kept to a minimum
- Data storage types:
  - Inverted index
  - Suffix trees
  - …



Underlying Model

- Fundamental component
- Framework for the representation of
  - queries
  - objects and
  - their relations
- Models
  - Boolean
  - Vector Space
  - Probabilistic, ...



Outline

universität koblenz
weiter:denken

- Pre-Processing
- Underlying Model: Boolean Model
- Internal Data Storage: Inverted index
- Input queries:
  - Boolean retrieval
  - Phrase queries
  - Wildcard searches
- Summary

1. Pre-Processing

What is a document?

universität koblenz
weiter:denken

- Mostly consisting of **words** from a surprisingly small dictionary
  - Increased by misspellings
- each of which independently convey some meaning.
- whose **meaning** is changed as words are concatenated into units of dialogue called sentences.
- for which there is a grammar of allowable combinations to which sentences conform.
- which are in turn are concatenated to make **prose** which makes up **documents**.
- which can reach a large enough size they may be **structurally organized**.
- and typically this is a **hierarchy**, to ease user navigation through the document.
  - chapters, sections, sub-sections, paragraphs, sentences, clauses, phrases, words, morphemes, letters…
- And don’t forget about **linked text** systems.

Problems with text

universität koblenz
weiter:denken

- Can be polysemous (many meanings)
  - e.g. BAR
- ‘SMITH’ is NOT ‘Smith’ is NOT ‘smith’.
  - or
- ‘plane’ is NOT ‘aircraft’
  - or
- “cooking” is NOT “cooked” is NOT “cookery”

However

- We still take words and index them one by one into the search engine
- Multiple meanings are often solved by the other words in the query or in the document

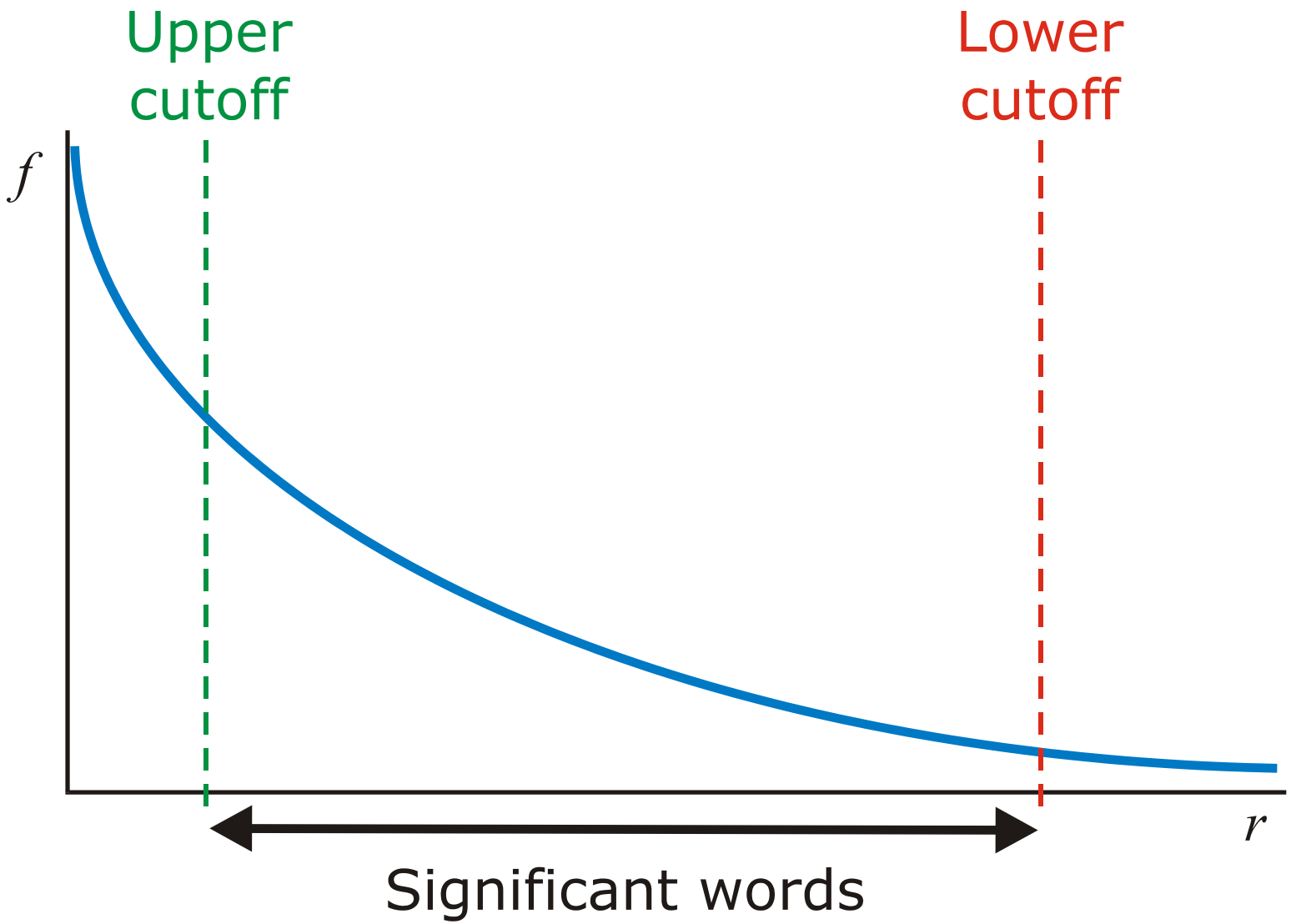
Document indexing

universität koblenz
weiter:denken

A process to convert a document into a fast searchable internal representation

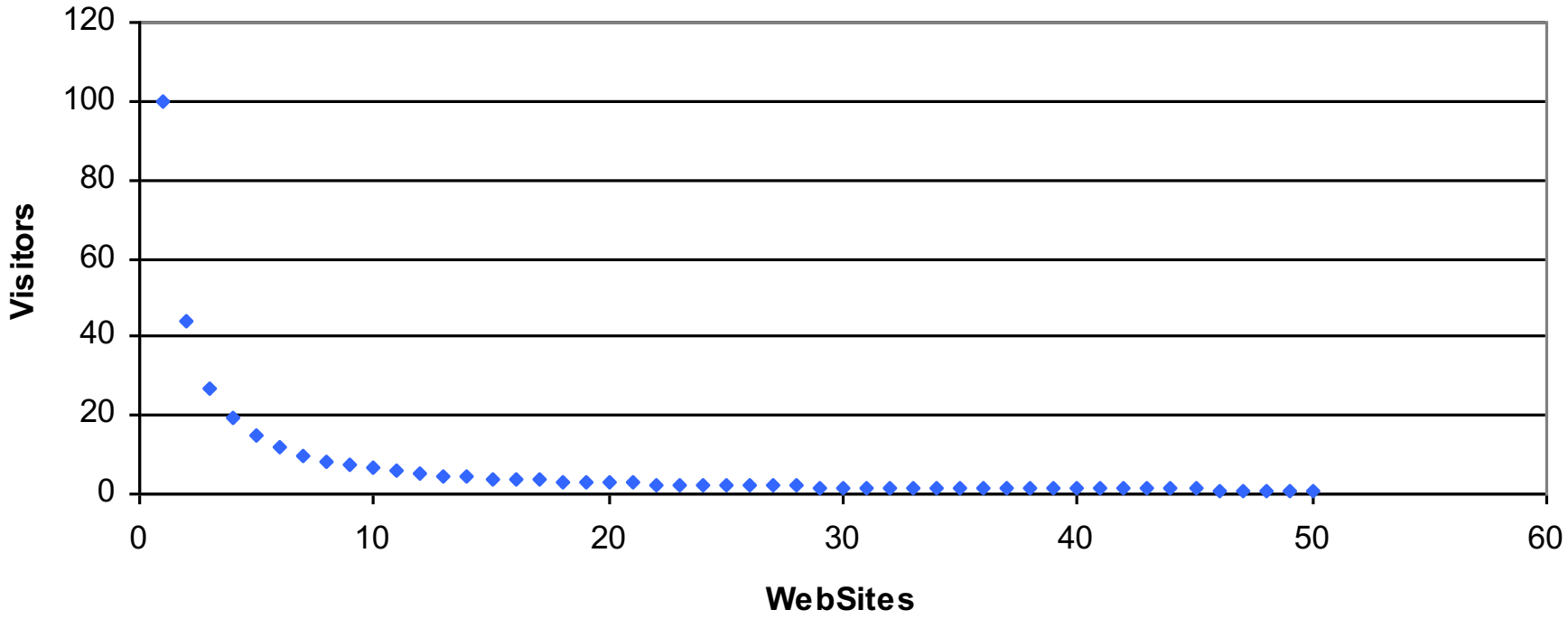
1. Document Tokenisation
   break into words… remove formatting (HTML)
2. Term Normalisation
   Make the same case (all lower case)
3. Stopword Removal
4. Term Weighting
5. Indexing of terms and term weights

Distribution of Word frequencies



A common distribution

universität koblenz
weiter:denken



This occurs everywhere

universität koblenz
weiter:denken

This type of distribution is all around us in the natural world

- Height of trees
- Distribution of wealth
- Height of buildings
- Size of lakes
- Size of melons
- Number of employees in companies
- Number of links into a web page
- Distribution of word usage in language
- Size of ???

What does this tell us about text?

universität koblenz
weiter:denken

We can make two observations relating to term importance, that:

- Terms below the lower bound are considered too infrequent to help search.
  - They may be removed, but in practise this does not happen
- Terms above the upper bound are considered to occur too frequently to be of benefit
  - They are often removed from the internal document representation.
    - These words are referred to as stopwords

Stopwords for English

universität koblenz
weiter:denken

a about above across after again against all almost alone along also although always am among an and another any anybody anyone anything anywhere apart are around as aside at away be because been before behind being below besides between beyond both but by can cannot could deep did do does doing done down downwards during each either else enough etc even ever every everybody everyone except far few for forth from get gets got had hardly has have having her here herself him himself his how however if in indeed instead into inward is it its itself just kept many maybe might mine more most mostly much must myself near neither next no nobody none nor not nothing nowhere of off often on only onto or other others ought our ours out outside over own per please plus quite rather really said seem self selves several shall she should since so some somebody somewhat still such than that the their theirs them themselves then there therefore these they this thorough thoroughly those through thus to together too toward towards under until up upon very was well were what whatever when whenever where whether which while who whom whose will with within without would yet young your yourself

Stopword removal

universität koblenz
weiter:denken

- We want to remove words that occur too often from the indexing process / search engine.
  - Can be done automatically using a predefined stopword list
- Benefits?
  - Smaller index (30-50% smaller)
- Problems?
  - But removing stopwords does cause difficulties in dealing with some valid queries… an example: “to be or not to be”…
  - So search engines do not do this on the WWW

Summary

universität koblenz
weiter:denken

May I have information on the computational complexity of nearest neighbour problems in graph theory.

Document Tokenisation & Term Normalisation ↓

may i have information on the computational complexity of nearest neighbour problems in graph theory

Stopword Removal ↓

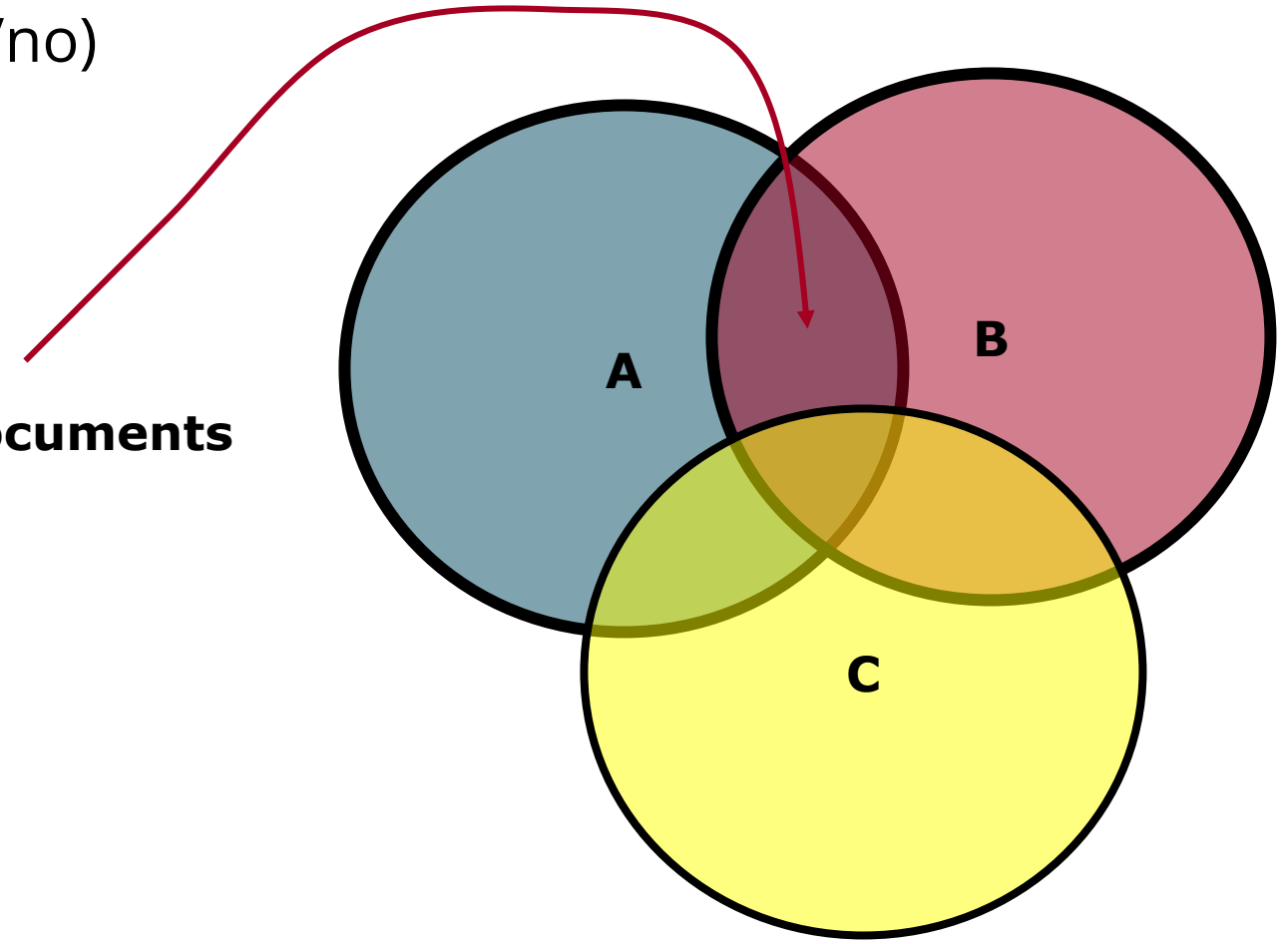
information computational complexity nearest neighbour problems graph theory

2. Underlying model: Boolean Model

Boolean Model

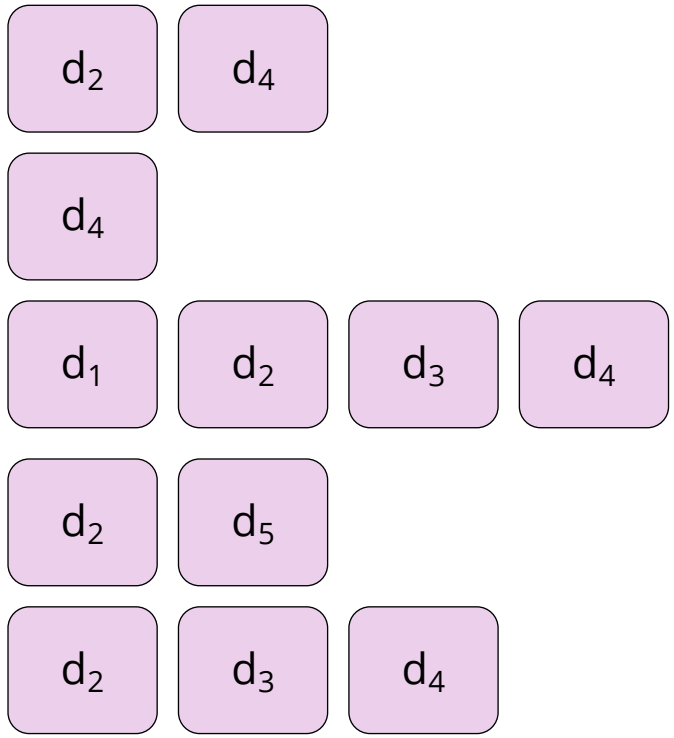
- Boolean
  - Retrieval based on boolean algebra
  - Binary concept of relevance (yes/no)
    - No ranking
  - Queries use boolean operators
- Examples
  - t_1
  - t_1 AND t_2
  - t_1 OR t_2
  - NOT t_1
  - (t_1 AND t_2) OR (t_3 AND t_4)

information AND retrieval NOT management
A AND B NOT C



Example: documents and queries

- Documents
  1. coffee, coffee
  2. tea, cup, jar, jar, tea
  3. cup, coffee, jar, cup
  4. coffee, cup, coffee, jar, tea, cup, coffee, jar, cup, jar
  5. jar, water, water, jar
- Queries
  - tea
  - tea AND coffee
  - tea OR coffee
  - NOT coffee
  - (tea AND cup) OR (coffee AND cup)



Theoretic Model

- Corpus: $D = \{d_1, d_2, \dots, d_N\}$
- Vocabulary: $V = \{t_1, t_2, \dots, t_M\}$
- Representation of documents
  - Of interest: is a given term present or not?
  - Document as vector in $\{0,1\}^M$

$$d_i = \left(\Delta_i^{[1]}, \Delta_i^{[2]} \dots, \Delta_i^{[M]}\right)^{\mathrm{T}}$$

  - ➢ where

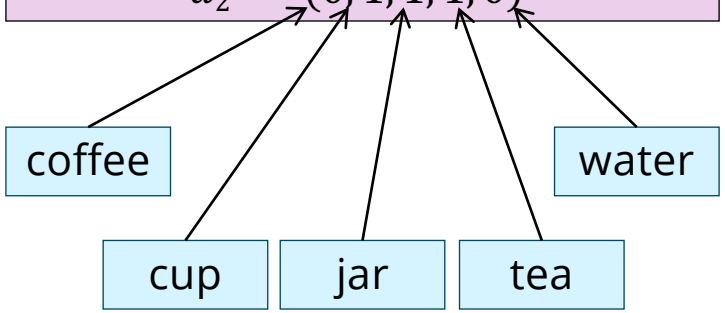
$$\Delta_i^{[j]} = \begin{cases} 0 : t_j \; not \; present \\ 1 : t_j \; present \end{cases}$$

$D = \{d_1, d_2, d_3, d_4, d_5\}$

$V = \{coffee, cup, jar, tea, water\}$

1. coffee, coffee
2. tea, cup, jar, jar, tea
3. cup, coffee, jar, cup
4. coffee, cup, coffee, jar, tea, cup, coffee, jar, cup, jar
5. jar, water, water, jar

$d_2 = (0, 1, 1, 1, 0)$



T-D-Matrix

- Term-document matrix $M \times N$:

$$C = \left(d_1^{[T]}, d_1^{[T]}, \ldots, d_1^{[T]}\right)$$

1. coffee, coffee
2. tea, cup, jar, jar, tea
3. cup, coffee, jar, cup
4. coffee, cup, coffee, jar, tea, cup, coffee, jar, cup, jar
5. jar, water, water, jar

$$\begin{bmatrix} & & d_1 & d_2 & d_3 & d_4 & d_5 \\ \text{coffee} & t_1 & 1 & 0 & 1 & 1 & 0 \\ \text{cup} & t_2 & 0 & 1 & 1 & 1 & 0 \\ \text{jar} & t_3 & 0 & 1 & 1 & 1 & 1 \\ \text{tea} & t_4 & 0 & 1 & 0 & 1 & 0 \\ \text{water} & t_5 & 0 & 0 & 0 & 0 & 1 \end{bmatrix}$$

Using the T-D Matrix to retrieve documents

- Single term query
  - Result: row in T-D-Matrix
- Combination
  - Bit operations on rows

		d_1	d_2	d_3	d_4	d_5
coffee	t_1	1	0	1	1	0
cup	t_2	0	1	1	1	0
jar	t_3	0	1	1	1	1
tea	t_4	0	1	0	1	0
water	t_5	0	0	0	0	1

- Example:
  - coffee AND tea

		d_1	d_2	d_3	d_4	d_5
coffee	t_1	1	0	1	1	0
		AND				
tea	t_4	0	1	0	1	0
RESULT		0	0	0	1	0

Limitations

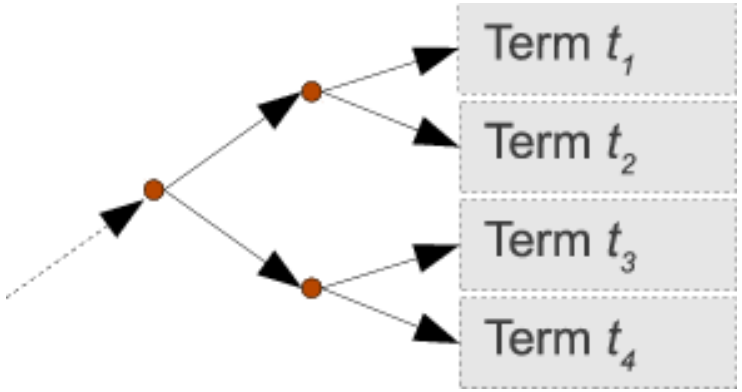
universität koblenz
weiter:denken

- T-D-Matrix is typically very large
  - 1,000,000 documents
  - 100,000 terms
  - each entry 1 bit
  - entire Matrix: 12.5 GB
- T-D-Matrix is typically very sparse
  - 1,000 terms per document: 99% of entries are 0
- Compress matrix : store only entries with value 1

3. Internal Data Storage: Inverted Index

Inverted Index

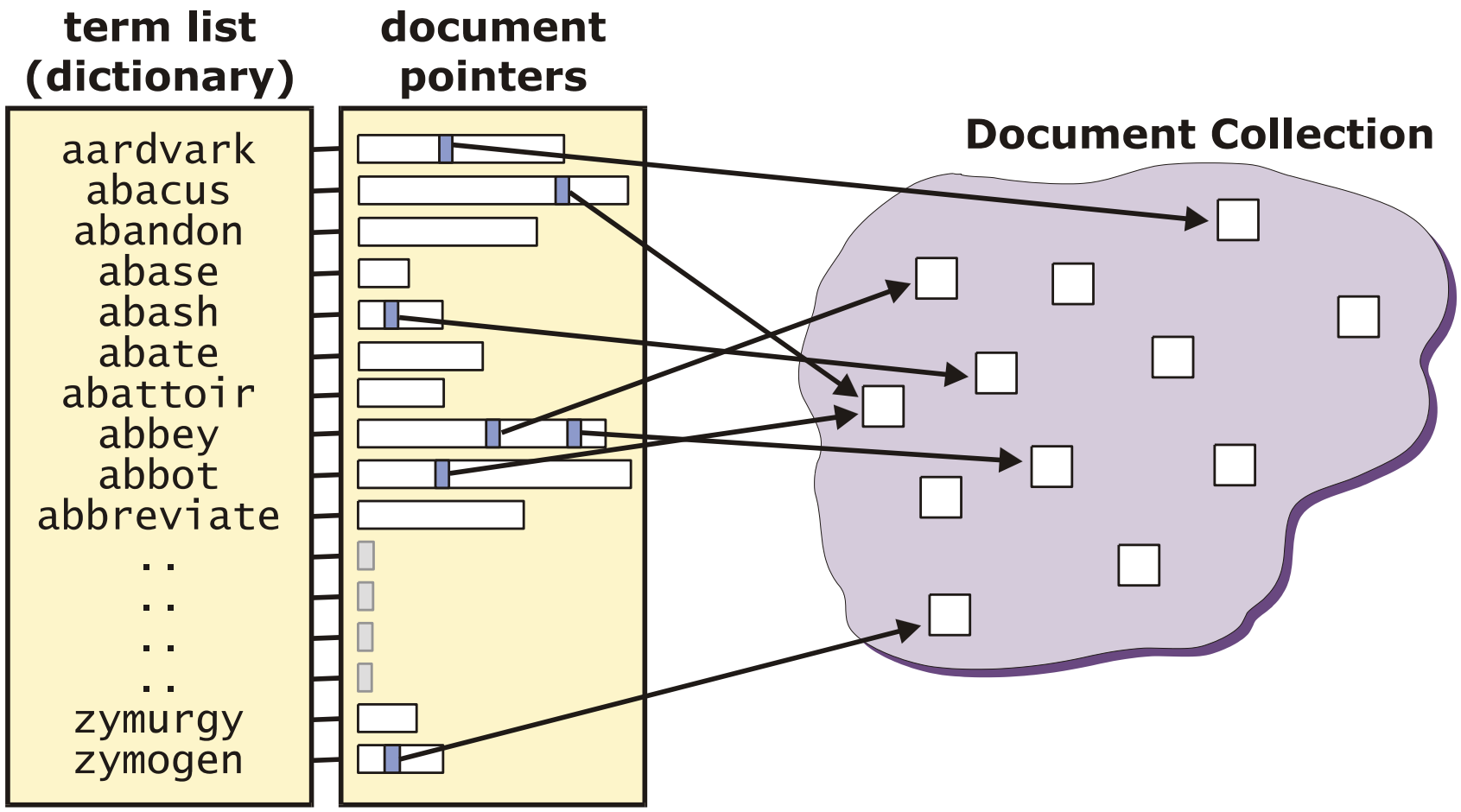
- Data structure consisting of
  - Lookup terms (row vectors)
    - ➢ Search tree



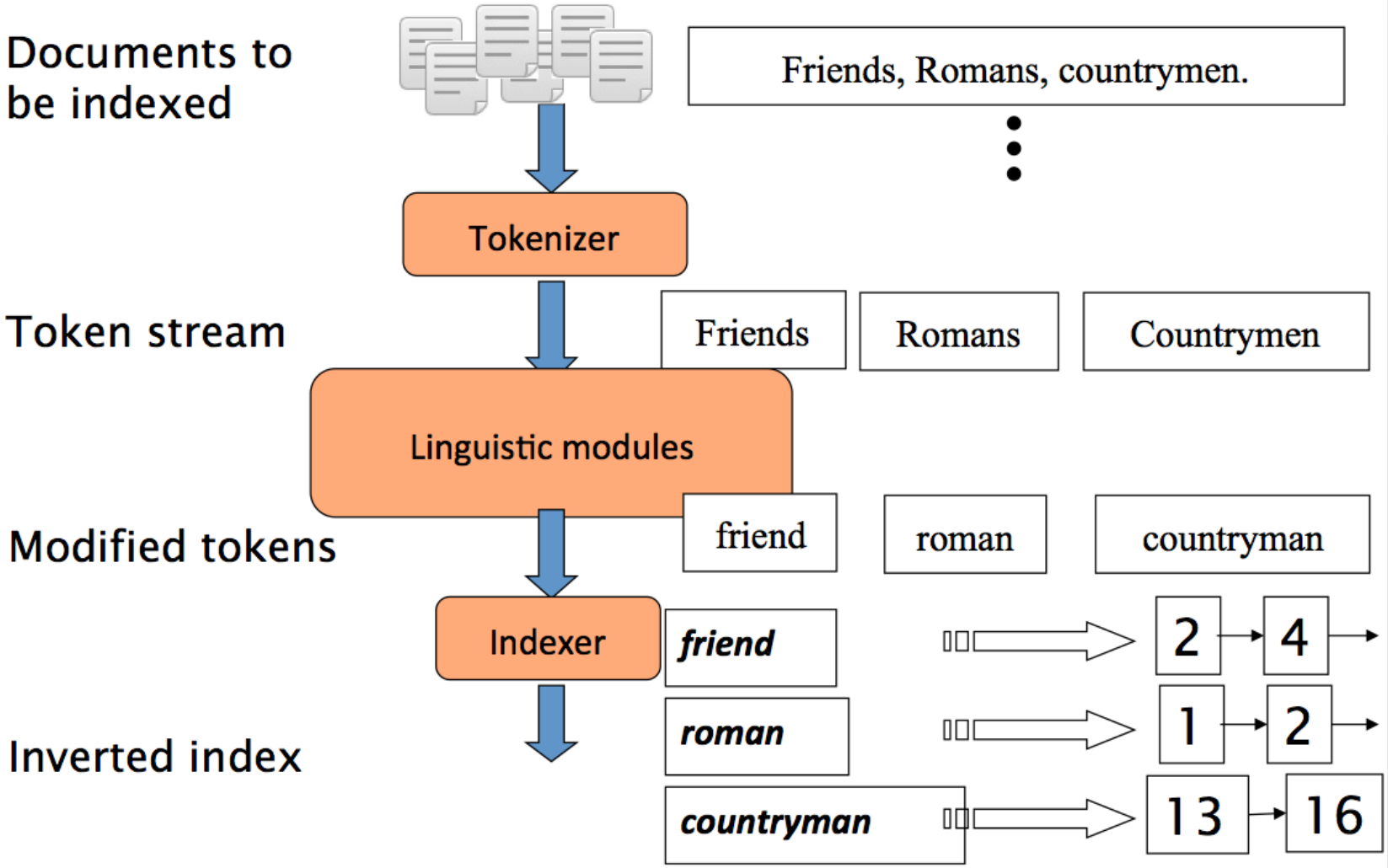
  - Posting-List of non-zero entries in vector
    - Linked list of postings

  - Posting: reference to a document

Example


term list
(dictionary)
document
pointers
Document Collection
aardvark
abacus
abandon
abase
abash
abate
abattoir
abbey
abbot
abbreviate
..
..
..
..
zymurgy
zymogen

Inverted index construction



Document icons from free icon set: http://www.icojoy.com/articles/44/

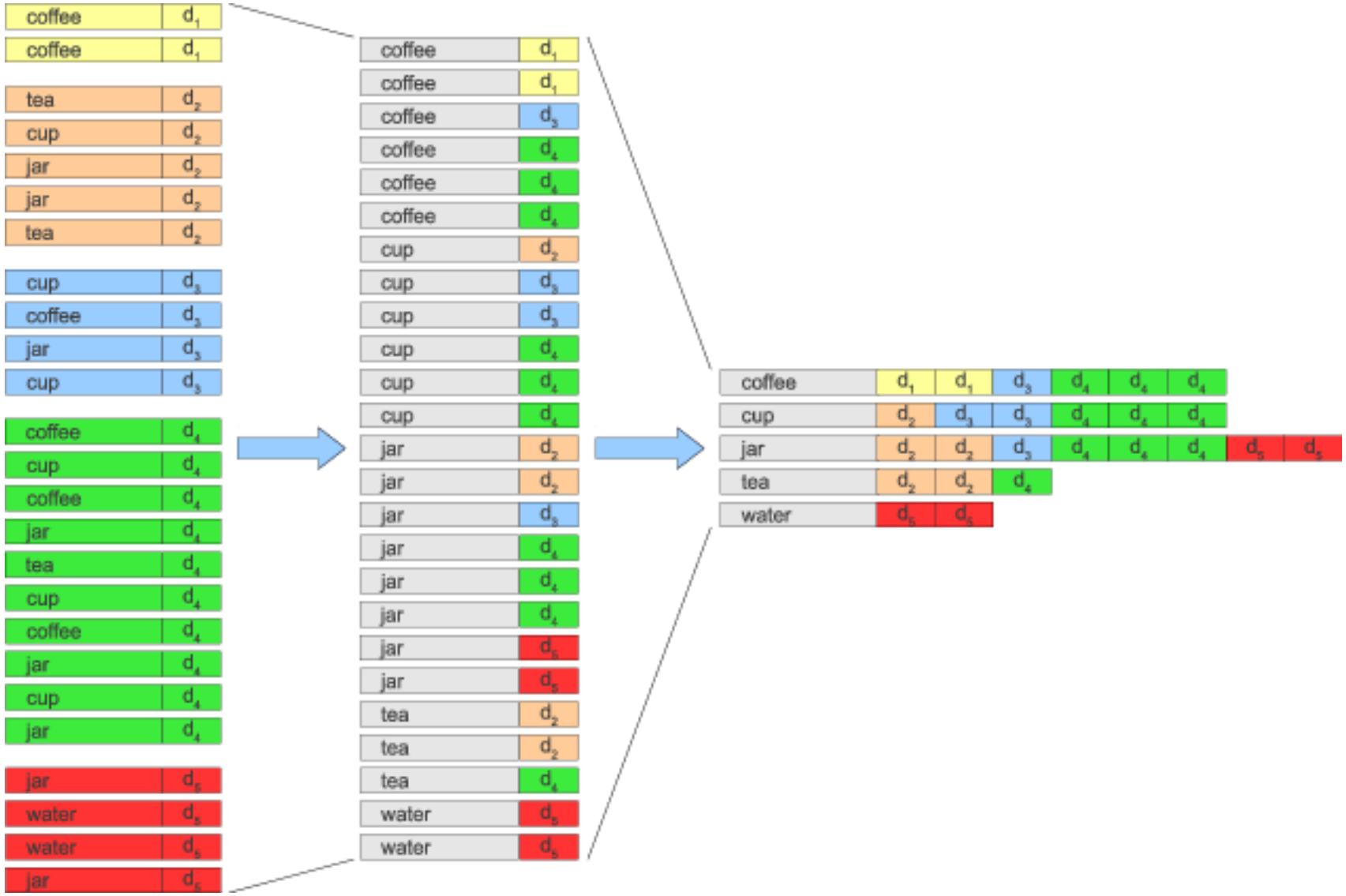
Initial stages of text processing

universität koblenz
weiter:denken

- Tokenization
  - Cut character sequence into word tokens
    - Deal with *"**John's**"*, ***a state-of-the-art solution***
- Normalization
  - Map text and query term to same form
    - You want ***U.S.A.*** and ***USA*** to match
- Stemming
  - We may wish different forms of a root to match
    - ***authorize***, ***authorization***
- Stop words
  - We may omit very common words (or not)
    - ***the, a, to, of***

Inverted Index Construction

- Sort terms with document references



Inverted Index

- Build search tree over vocabulary
- Compile a posting list for each term

4. Input Query

universität koblenz
weiter:denken

Boolean retrieval

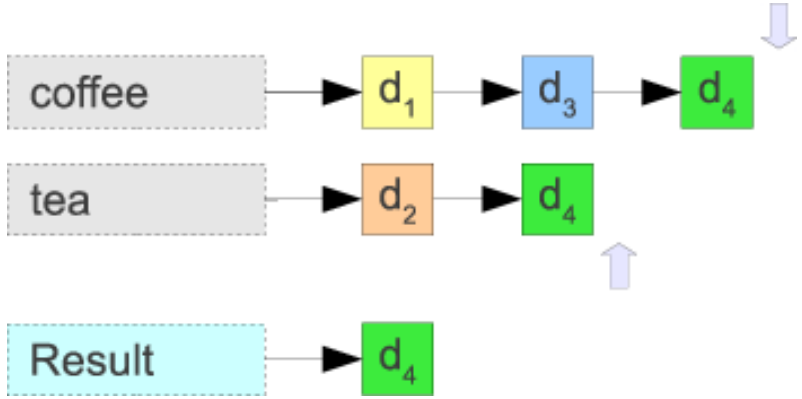
Implementing Boolean Retrieval

- Search for single term $q = t$
  - Look up posting list for t … and done!
- Example: query „tea“

Result of a query is a posting list

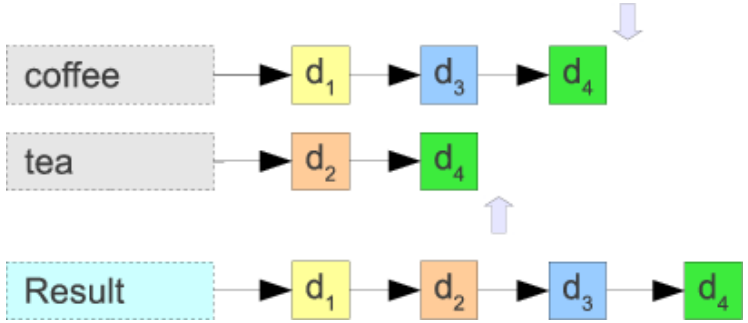
Implementing Boolean Retrieval

- Search for $q = q_1 \, AND \, q_2$
  - Intersect the result lists (posting lists)
- Example: query „coffee AND tea“



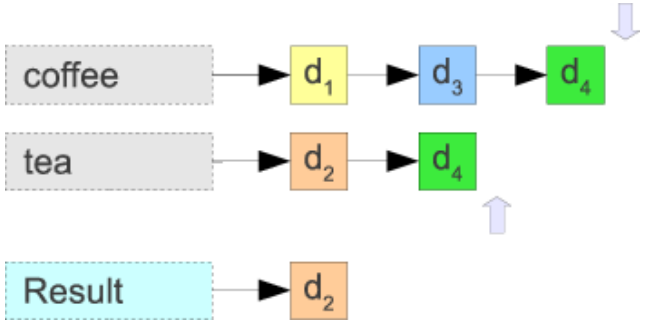
Implementing Boolean Retrieval

- Search for $q = q_1 \text{ OR } q_2$
  - Merge the result lists (posting lists)
- Example: query „coffee OR tea“



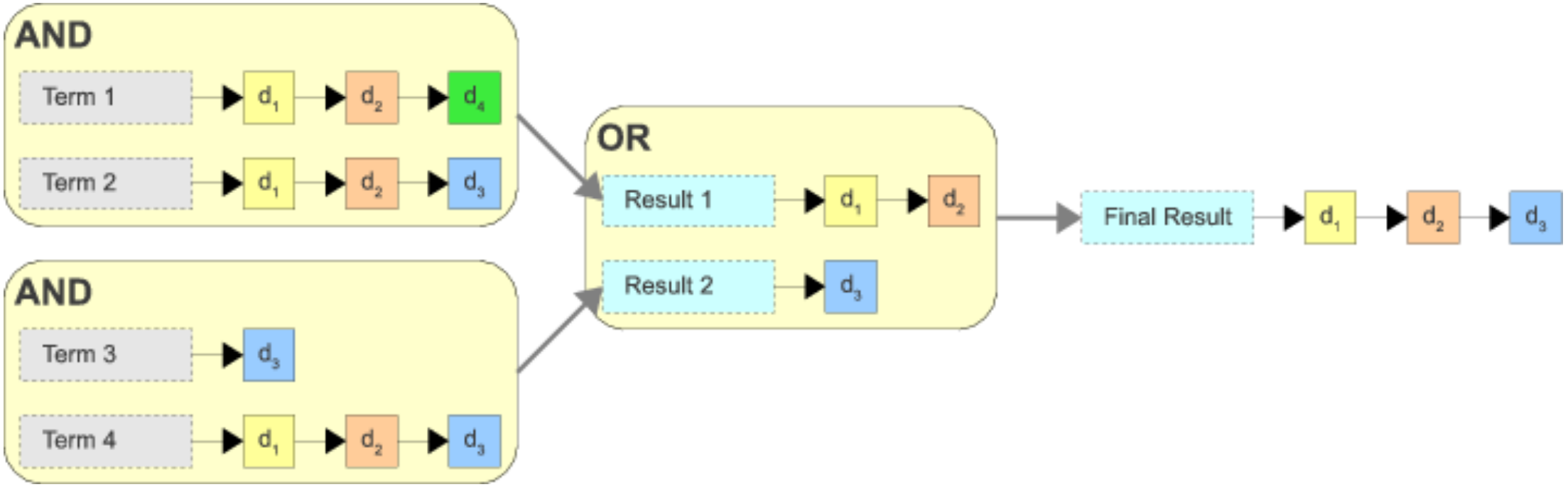
Implementing Boolean Retrieval

- Query: $\text{NOT } q$?
  - Technically simple
    - ➢ Invert posting list
- In practice
  - Too many entries
- BUT operator
  - Binary operator
  - Defined as: $q_1 \text{ BUT } q_2 = q_1 \text{ AND}(\text{NOT } q_2)$
- Example: „tea BUT coffee“



Implementing Boolean Retrieval

- More complex queries:
  - t_1 AND t_2 OR t_3 AND t_4
  - Recursive approach



Optimization

- Intersect short lists first
  - Faster to process
  - Result lists get shorter
  - Empty list serves as a stop criterion
- Example: query „coffee AND jar AND water“

Optimization

- Annotate terms with the length of posting list
  - List of length = number of documents containing term
  - Document frequency

$$df(t_i)$$

universität koblenz
weiter:denken

Phrase Queries

Phrase queries

universität koblenz
weiter:denken

- We want to be able to answer queries such as

"stanford university" – as a phrase

- Thus the sentence *"I went to university at Stanford"* is not a match
  - The concept of phrase queries has proven easily understood by users; one of the few "advanced search" ideas that works
  - Many more queries are *implicit phrase queries*
- For this, it is no longer sufficient to store only

<*term* : *docs*> entries

A first attempt: Biword indexes

universität koblenz
weiter:denken

- 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 direct

Longer phrase queries

universität koblenz
weiter:denken

- Longer phrases can be processed by breaking them down
- ***stanford university palo alto*** can be broken into the Boolean query on biwords:
  - ***stanford university*** *AND* ***university palo*** *AND* ***palo alto***
- Without the docs, we cannot verify that the docs matching the above Boolean query do contain the phrase
  - It can have false positives!

Issues regarding biword indexes

universität koblenz
weiter:denken

- False positives, as noted before
- Index blow-up 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

Solution 2: Positional indexes

universität koblenz
weiter:denken

- 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.>

Example: positional index

<***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

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; …

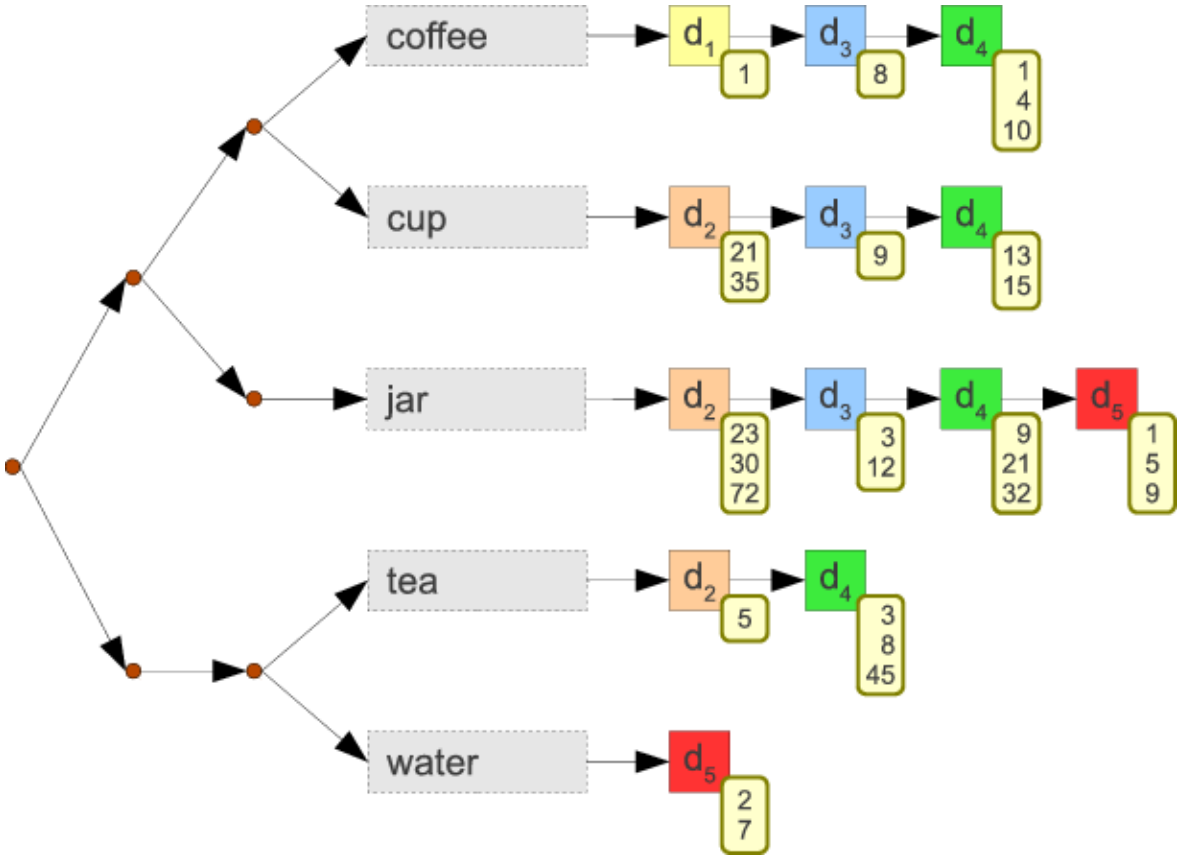
  - ***be***

    - *1*:17,19; *4*:17,191,291,430,434; *5*:14,19,101; …

- Same general method for proximity searches

Position Index

- Store position of term in postings



- Intersect position-lists with offset
- Also allows for NEAR operator

Proximity queries

universität koblenz
weiter:denken

- LIMIT! /3 STATUTE /3 FEDERAL /2 TORT
  - Here, */k* means "within *k* words of"
- Clearly, positional indexes can be used for such queries;
  - biword indexes cannot

Positional index size

universität koblenz
weiter:denken

- A positional index expands postings storage *substantially*
  - Even though indices can be compressed
- Nevertheless, a positional index is now standardly used because of the power and usefulness of phrase and proximity queries
  - … whether used explicitly or implicitly in a ranking retrieval system

universität koblenz
weiter:denken

Wildcard Searches

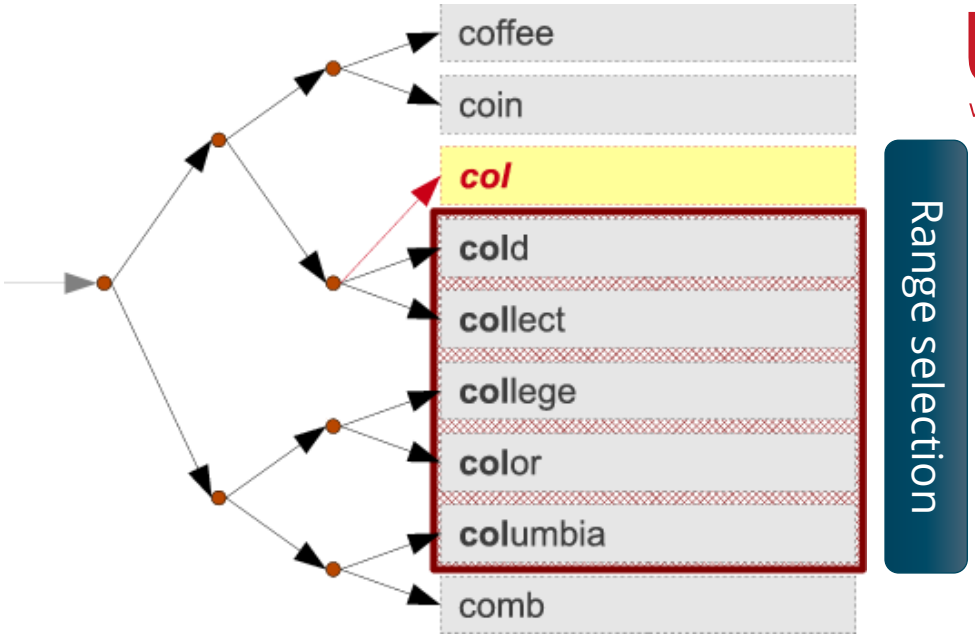
Wildcard Search

universität koblenz
weiter:denken

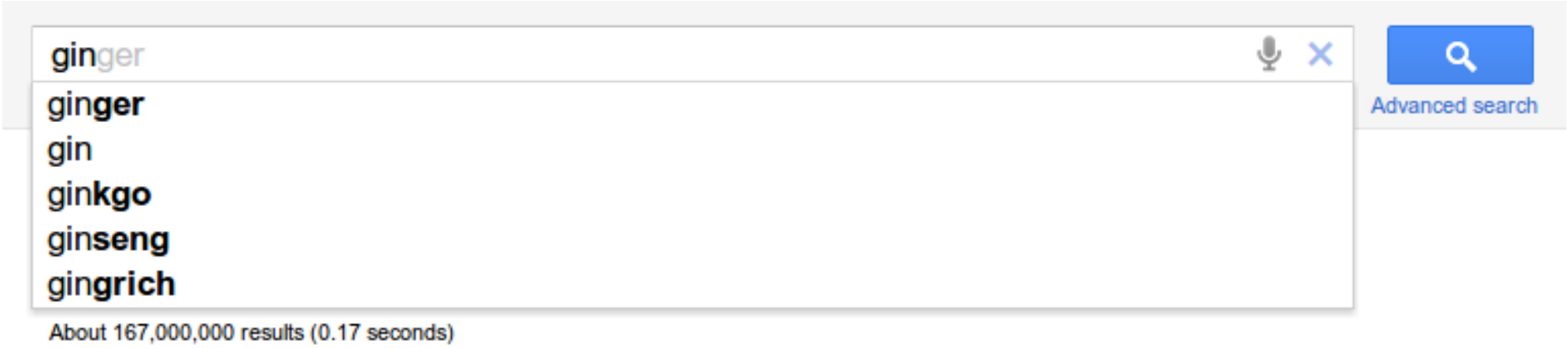
- Flexible search
  - Leave out fragments of terms, mark with wildcard
    - Question mark (?): single character
    - Asterisk (*): zero, one or more characters
- Examples:
  - na?ve → naive, naïve
  - universit* → university, universität, universitá
  - go* → go, goes, gone
  - g?n* → gun, gone, gin, ginger

Wildcard

- Simple case
  - Use sorting in inverted index
  - Example: „col*“



- Further application: Autocomplete (without ranking)



5. Summary

Summary

universität koblenz
weiter:denken

- At the end of this lecture, you are expected to understand the concepts of
  - Boolean model
  - T-D matrix
  - Retrieval function
  - Inverted index
  - Posting list
  - Optimization
  - Phrase queries
  - Positional index
  - Wildcard search