universität koblenz

weiter:denken

Web Information Retrieval

Boolean Retrieval & Phrase Queries

(SOSE 2023)

Frank Hopfgartner, Stefania Zourlidou
Institute for Web Science and Technologies

Credit for these slides

These slides have been adapted from

- Web IR (Zeyd Boukhers-WeST, SOSE 2020)

Recapitulation

- 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?

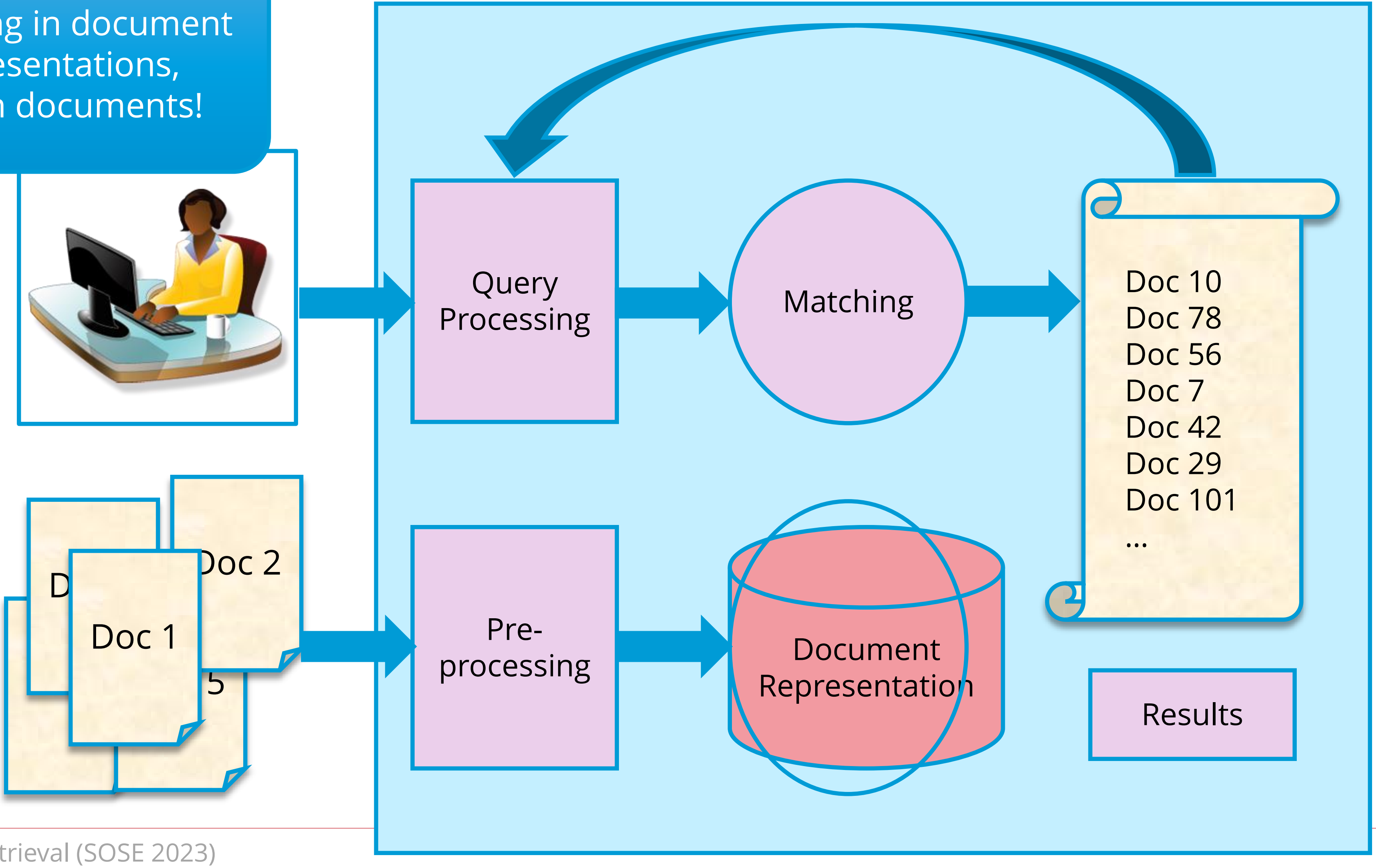
Objectives of this lecture

- What is a *Boolean model*?
  - Examples of Boolean models
- What is a *T-D matrix?*
- *What is a retrieval function?*
- *What is* an *inverted index*?
- What is a *phrase query*?
- What are *wildcard searches*?

1. Boolean Model

Information Retrieval ⇔ Fulltext search

Searching in document representations,
NOT in documents!



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)

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)

d_2 d_4

d_4

d_1 d_2 d_3 d_4

d_2 d_5

d_2 d_3 d_4

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 $\{0{,}1\}^M$

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

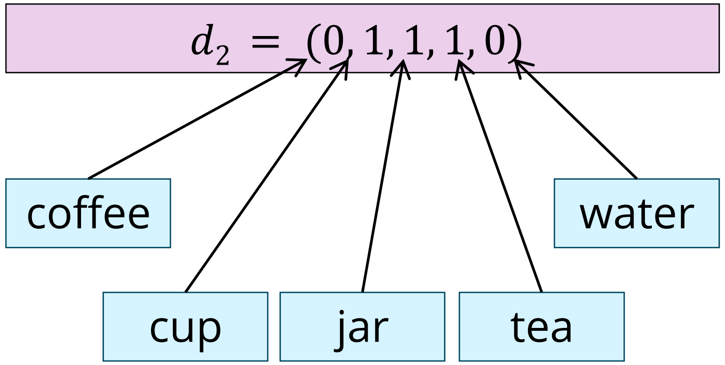
➢ where

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

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

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

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

$$\left[\begin{array}{ccccccc} & & d_1 & d_2 & d_3 & d_4 & d_5 \\ \text{coffee} & t_1 & 1 & 0 & 1 & 1 & 0 \\ \text{cup} & t_2 & 0 & 1 & 1 & 1 & 0 \\ \text{jar} & t_3 & 0 & 1 & 1 & 1 & 1 \\ \text{tea} & t_4 & 0 & 1 & 0 & 1 & 0 \\ \text{water} & t_5 & 0 & 0 & 0 & 0 & 1 \end{array}\right]$$

Queries

- Set of all possible queries: Q
- Recursive definition of queries
  - Each term is a query
    - $\forall t \in V : t \in Q$
  - Combination of queries are queries
    - $\forall q_1, q_2 \in Q : q_1 \text{ AND } q_2 \in Q$
    - $\forall q_1, q_2 \in Q : q_1 \text{ OR } q_2 \in Q$
    - $\forall q \in Q : \text{NOT } q \in Q$
  - For clarity, add parenthesis
    - $\forall q \in Q : (q) \in Q$

Retrieval function ρ

- Function

$$\rho : D \times Q \rightarrow \{0,1\}$$

  - Values:
    - $\rho(d,q) = 1$: document d is relevant to query q
    - $\rho(d,q) = 0$: document d is not relevant to query q

- Recursive definition
  - For single term queries
    - $\rho\left(d_i, t_j\right) = \Delta_i^{[j]}$
  - For compositions
    - $\rho(d_i, q_1 \mathrm{AND} q_2) = \min\left(\rho(d_i, q_1), \rho(d_i, q_2)\right)$
    - $\rho(d_i, q_1 \mathrm{OR} q_2) = \max\left(\rho(d_i, q_1), \rho(d_i, q_2)\right)$
    - $\rho(d_i, \mathrm{NOT} q) = 1 - \rho(d_i, q)$

Alternative View using T-D Matrix

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

$$\left[\begin{array}{ccccccc} & & d_1 & d_2 & d_3 & d_4 & d_5 \\ \text{coffee} & t_1 & 1 & 0 & 1 & 1 & 0 \\ \text{cup} & t_2 & 0 & 1 & 1 & 1 & 0 \\ \text{jar} & t_3 & 0 & 1 & 1 & 1 & 1 \\ \text{tea} & t_4 & 0 & 1 & 0 & 1 & 0 \\ \text{water} & t_5 & 0 & 0 & 0 & 0 & 1 \end{array}\right]$$

- Example:
  - coffee AND tea

This view will be exploited for realization

$$\begin{array}{ccccccc} & & d_1 & d_2 & d_3 & d_4 & d_5 \\ \text{coffee} & t_1 & 1 & 0 & 1 & 1 & 0 \\ & & & & \text{AND} & & \\ \text{tea} & t_4 & 0 & 1 & 0 & 1 & 0 \\ \hline \text{RESULT} & & 0 & 0 & 0 & 1 & 0 \end{array}$$

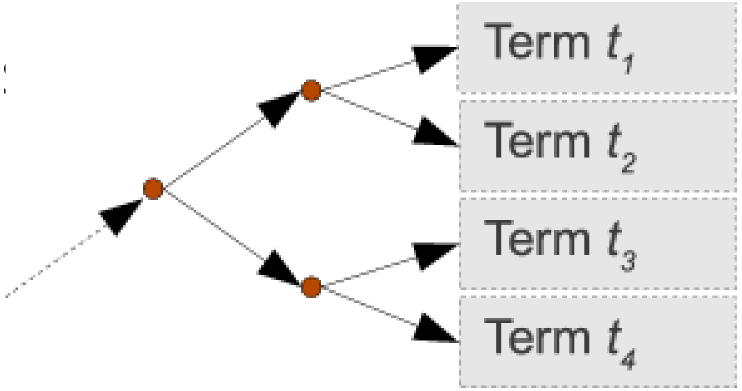
2. Inverted Index & Boolean Retrieval

Inverted Index

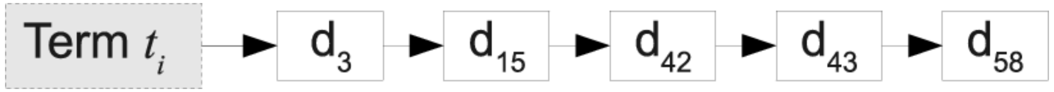
- Central data-structure in IR
- Requirements from T-D-matrix view
  - Lookup row vectors (term-vector)
  - Apply bit operations (AND, OR, NOT)
- Additionally
  - T-D-Matrix is typically very large
    - 1,000,000 documents
    - 100,000 terms
    - each entry 1 bit
    - entire Matrix: 12.5 GB
  - T-D-Matrix is typically very sparse
    - 1,000 terms per document: 99% of entries are 0
  - Compress matrix : store only entries with value 1

Inverted Index

- Data structure consisting of
  - Lookup terms (row vector
    - Search tree

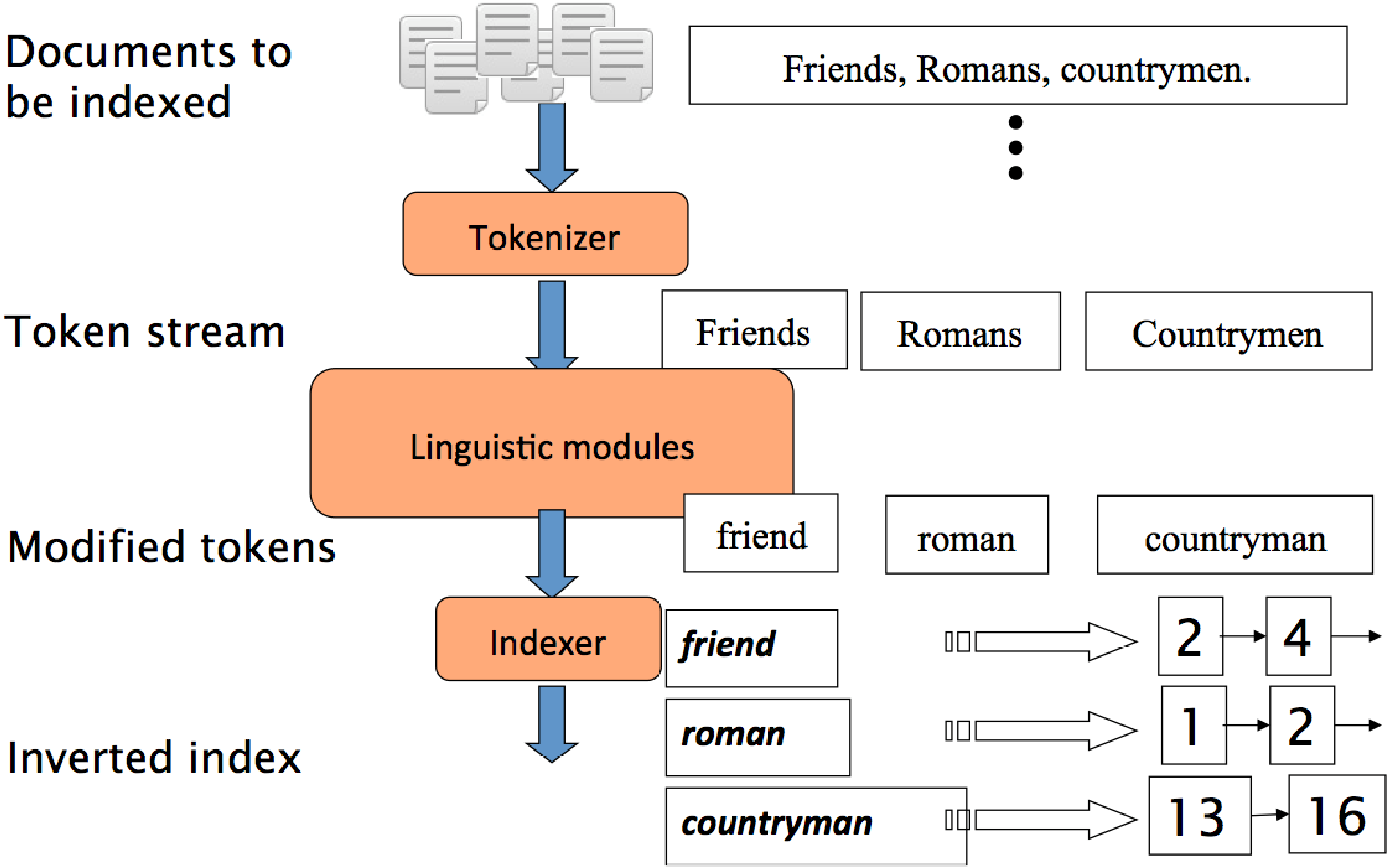


- Posting-List of non-zero entries in vector
  - Linked list of postings



- Posting: reference to a document

Inverted index construction



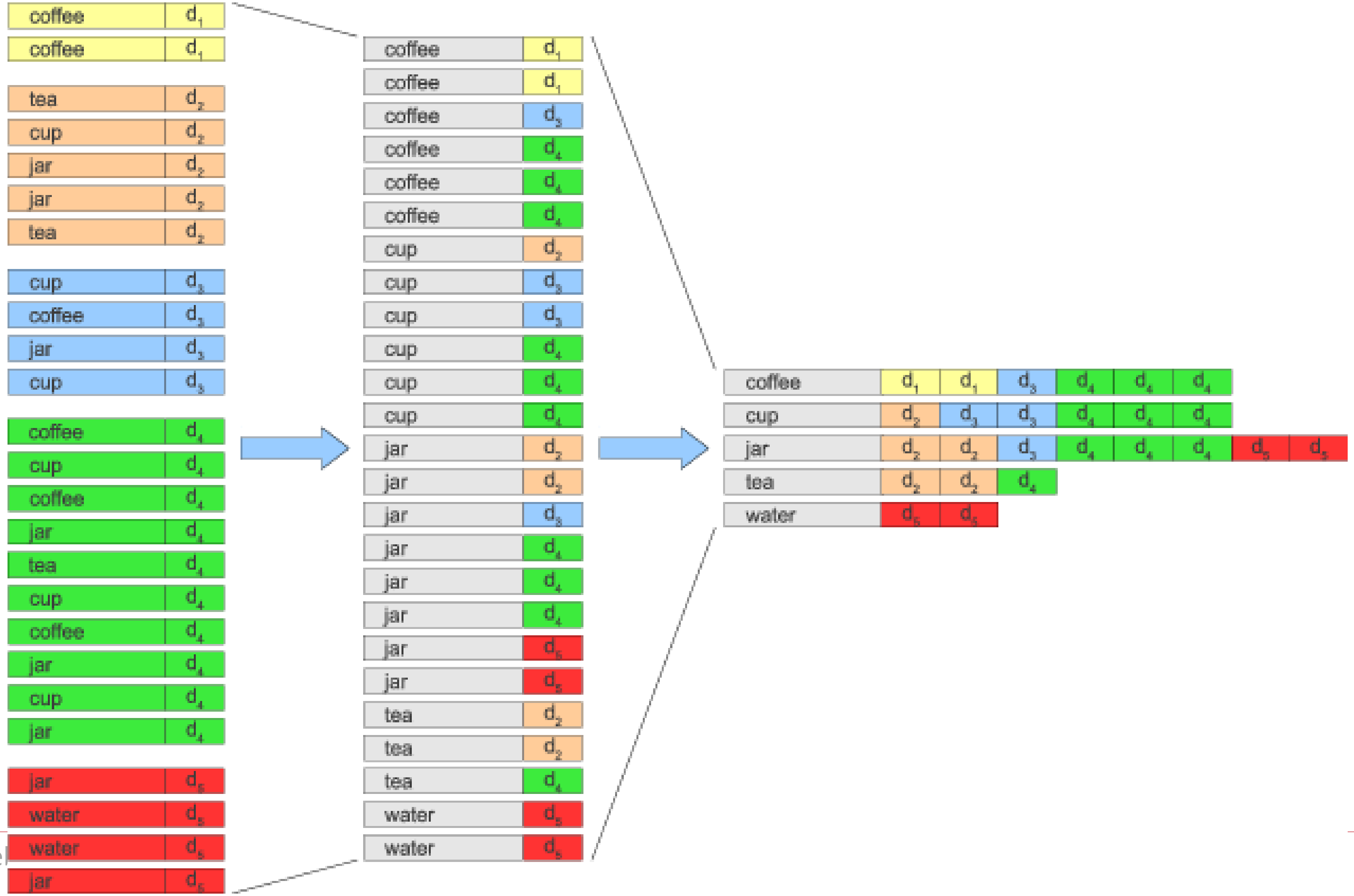
Document icons from free icon set: http://www.icojoy.com/articles/44/

Initial stages of text processing

- 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



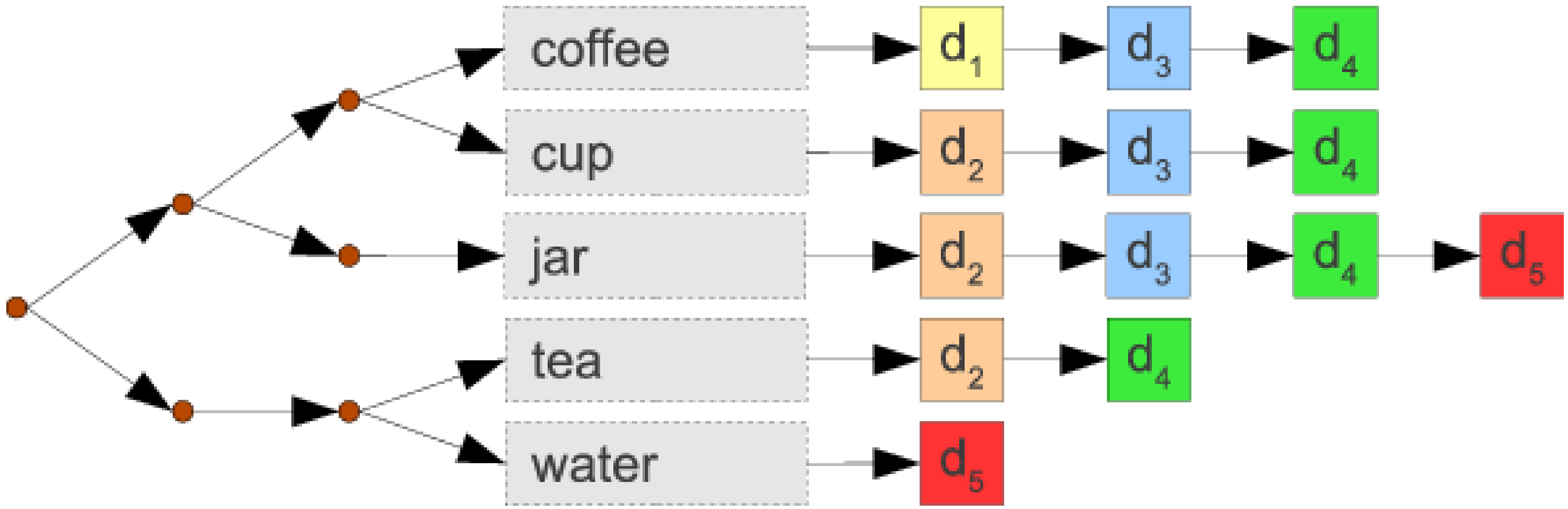
coffee	d_1
coffee	d_1
tea	d_2
cup	d_2
jar	d_2
jar	d_2
tea	d_2
cup	d_3
coffee	d_3
jar	d_3
cup	d_3
coffee	d_4
cup	d_4
coffee	d_4
jar	d_4
tea	d_4
cup	d_4
coffee	d_4
jar	d_4
cup	d_4
jar	d_4
jar	d_5
water	d_5
water	d_5
jar	d_5

coffee	d_1
coffee	d_1
coffee	d_3
coffee	d_4
coffee	d_4
coffee	d_4
cup	d_2
cup	d_3
cup	d_3
cup	d_4
cup	d_4
cup	d_4
jar	d_2
jar	d_2
jar	d_3
jar	d_4
jar	d_4
jar	d_4
jar	d_5
jar	d_5
tea	d_2
tea	d_2
tea	d_4
water	d_5
water	d_5

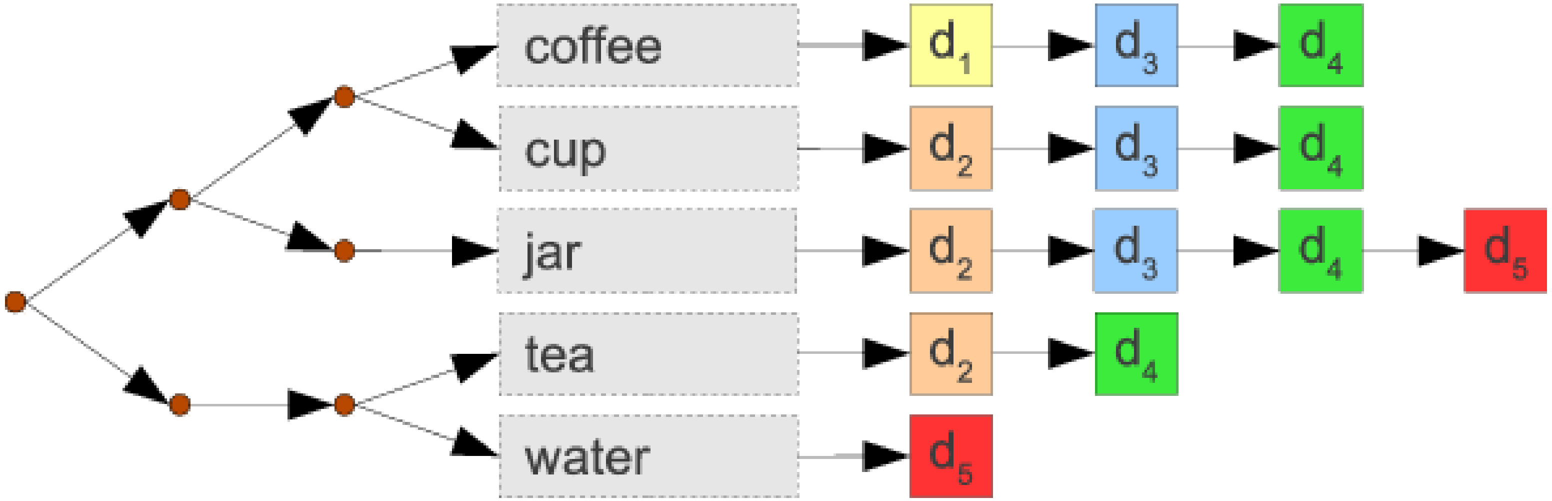
coffee	d_1	d_1	d_3	d_4	d_4	d_4		
cup	d_2	d_3	d_3	d_4	d_4	d_4		
jar	d_2	d_2	d_3	d_4	d_4	d_4	d_5	d_5
tea	d_2	d_2	d_4					
water	d_5	d_5						

Inverted Index

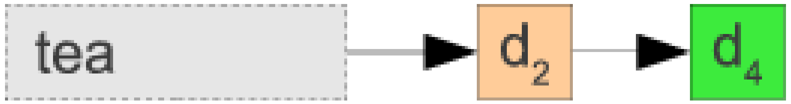
- Build search tree over vocabulary
- Compile a posting list for each term



Implementing Boolean Retrieval

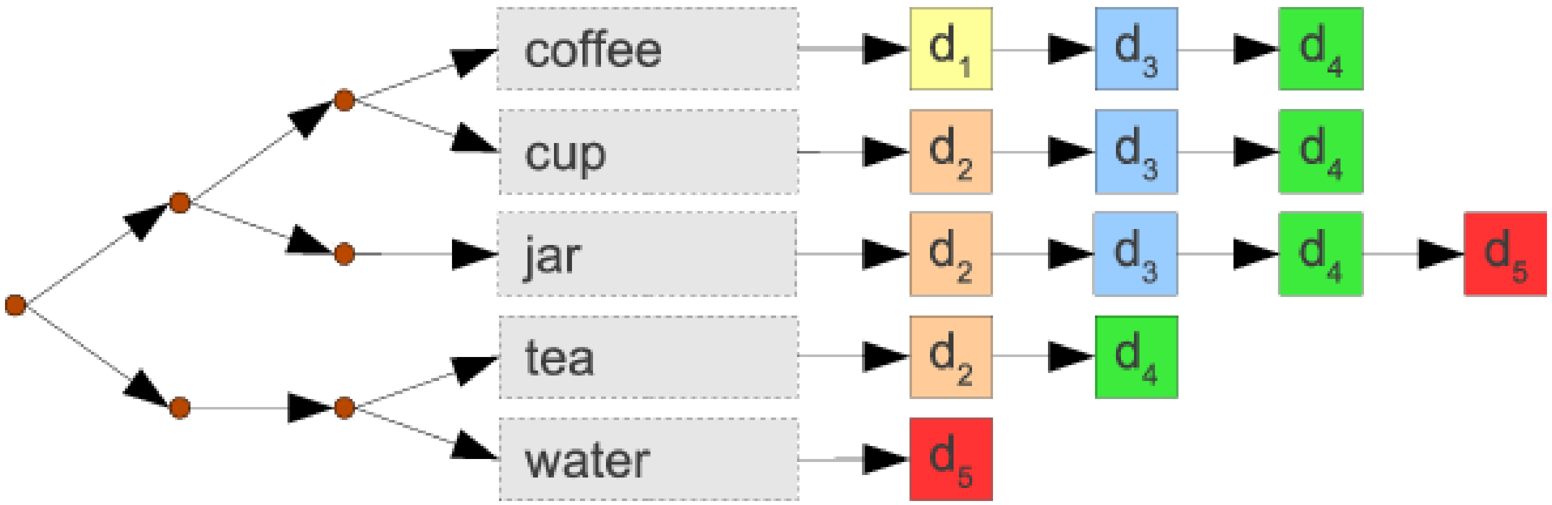


- Search for single term $q = t$
  - Look up posting list for t ... and done!
- Example: query „tea“

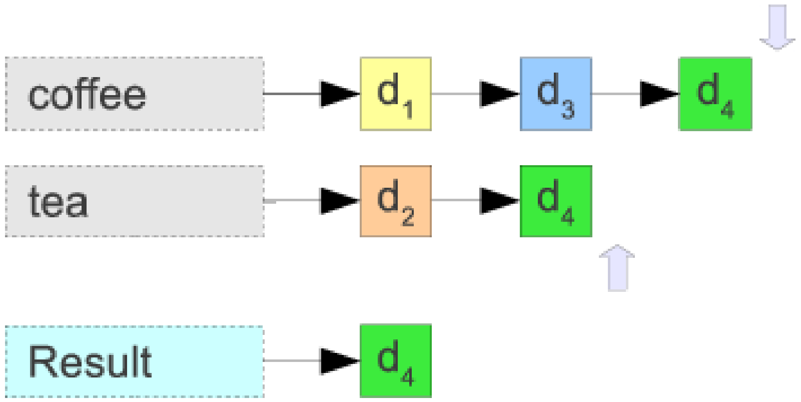


Result of a query is a posting list

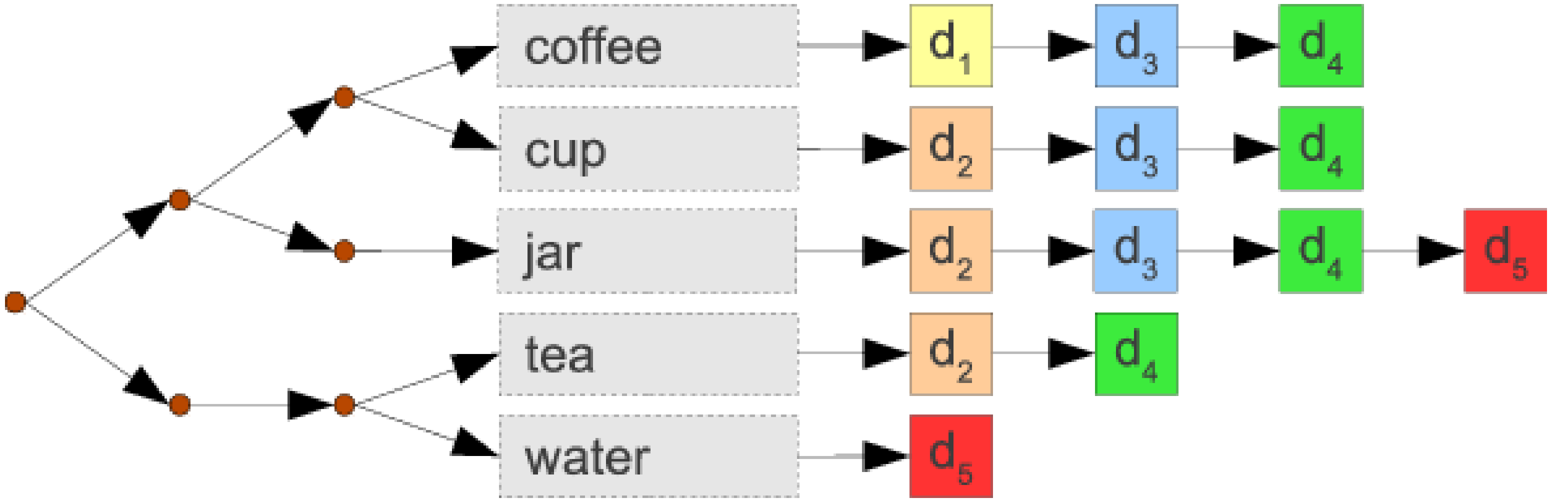
Implementing Boolean Retrieval



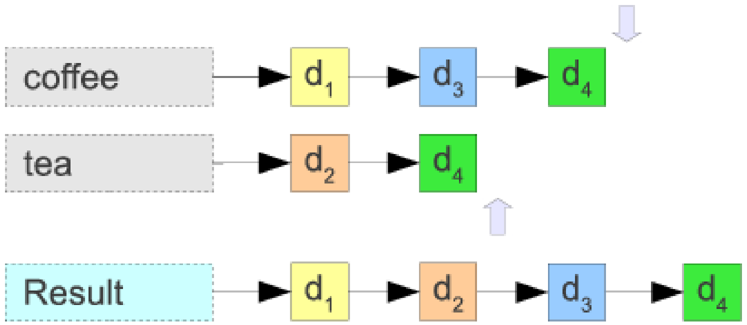
- Search for $q = q_1 \; AND \; q_2$
  - Intersect the result lists (posting lists)
- Example: query „coffee AND tea“



Implementing Boolean Retrieval

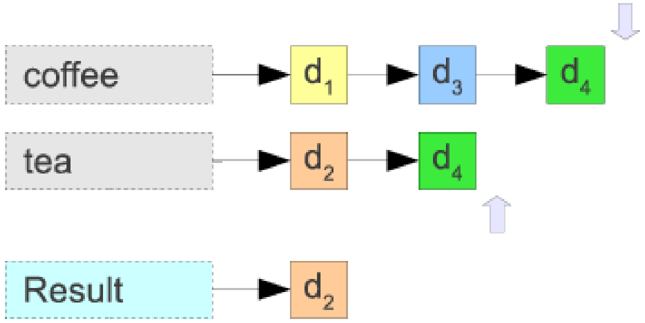


- Search for $q = q_1 \text{ OR } q_2$
  - Merge the result lists (posting lists)
- Example: query „coffee OR tea“



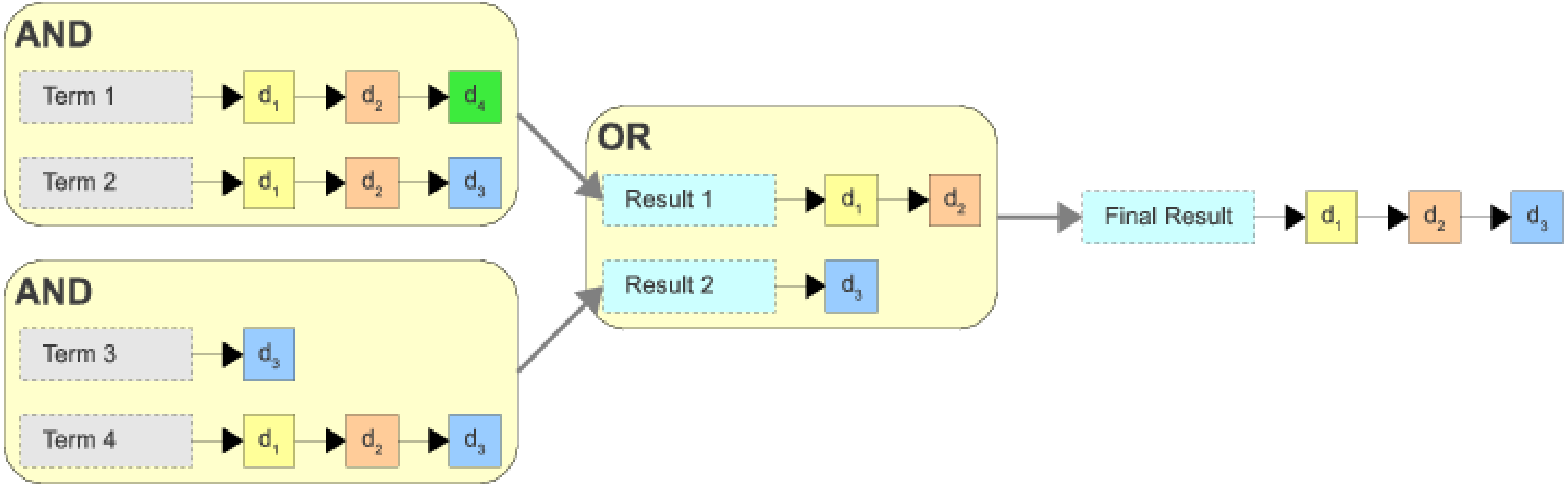
Implementing Boolean Retrieval

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



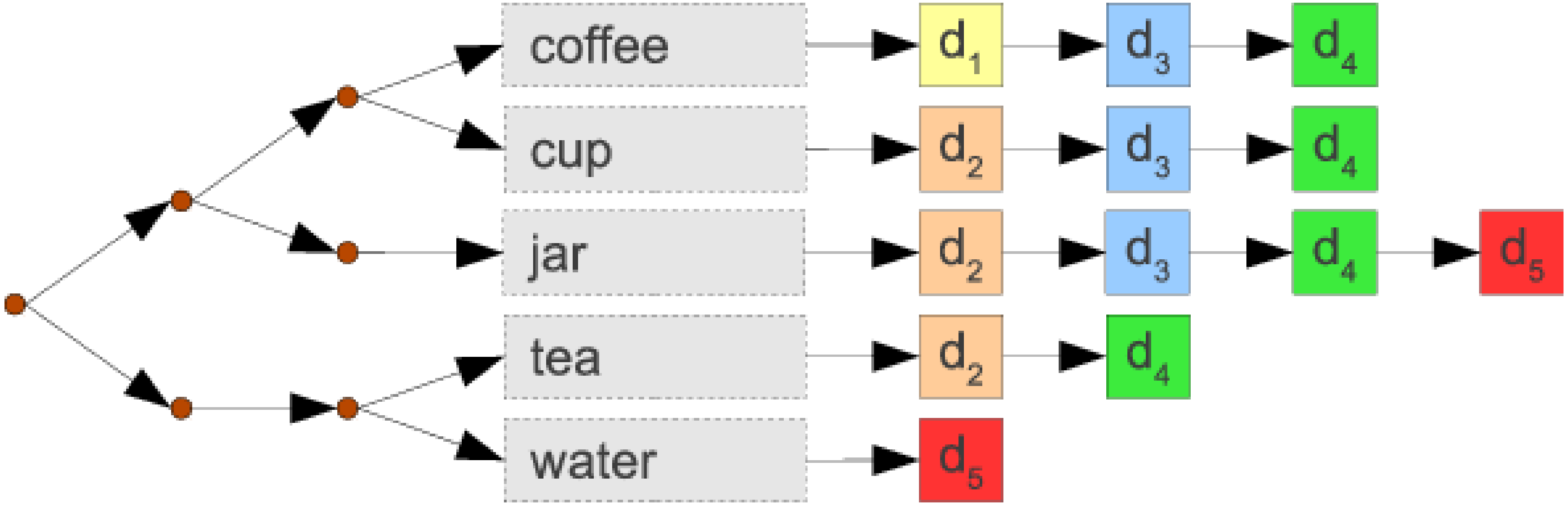
Implementing Boolean Retrieval

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



Optimization

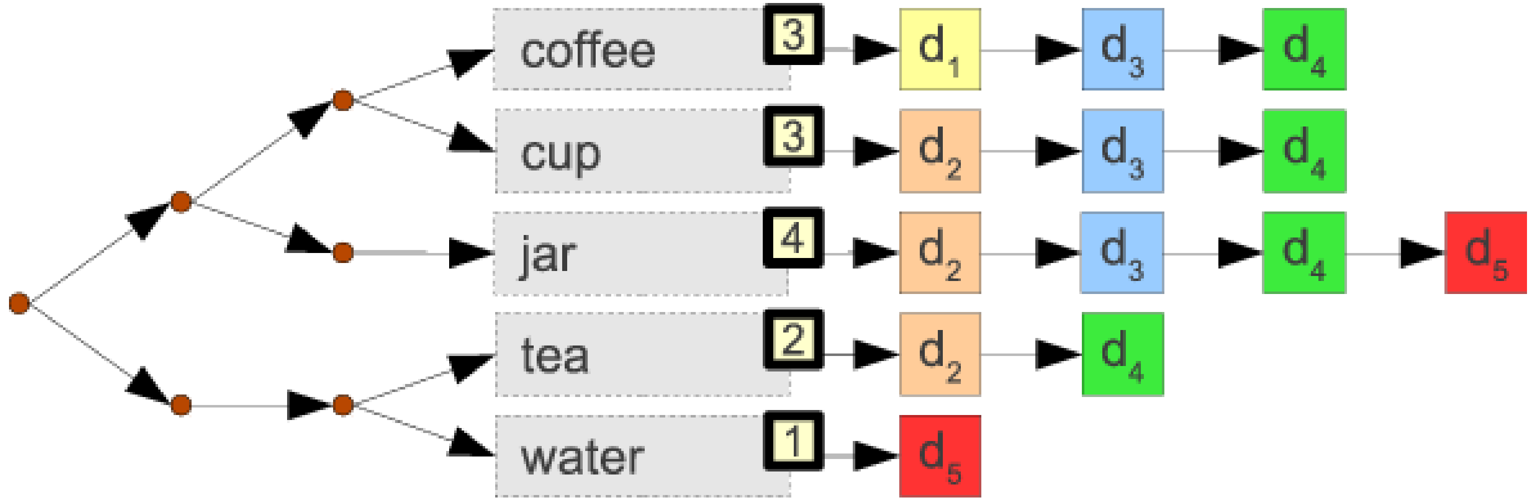
- Intersect short lists first
  - Faster to process
  - Result lists get shorter
  - Empty list serves as a stop criterion
- Example: query „coffee AND jar AND water“



Optimization

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

$$df(t_i)$$



3. Phrase Queries

Phrase queries

- 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 no longer sufficient to store only

<*term* : *docs*> entries

A first attempt: Biword indexes

- 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

- 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

- 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

- 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

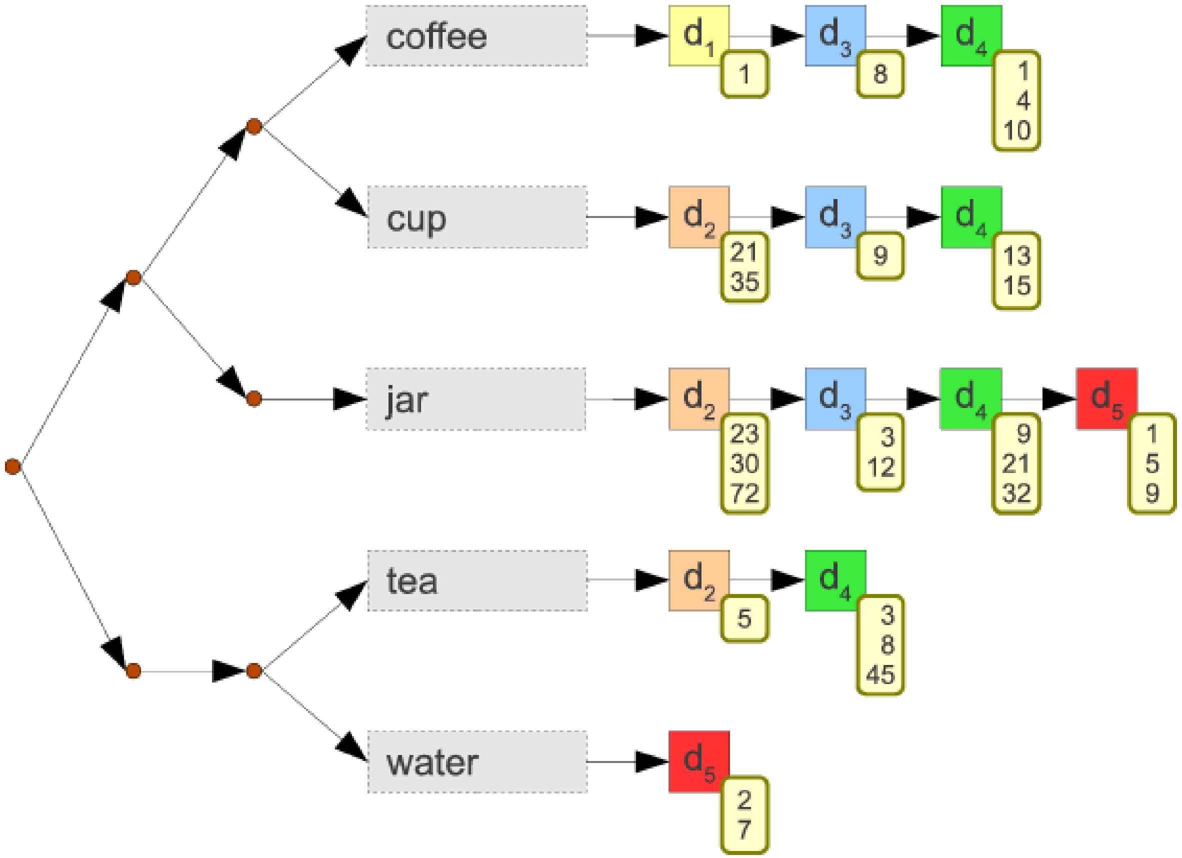
- Merge their *doc: position* lists to enumerate all positions with "***to be or not to be***".

  - ***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

- 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

- 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

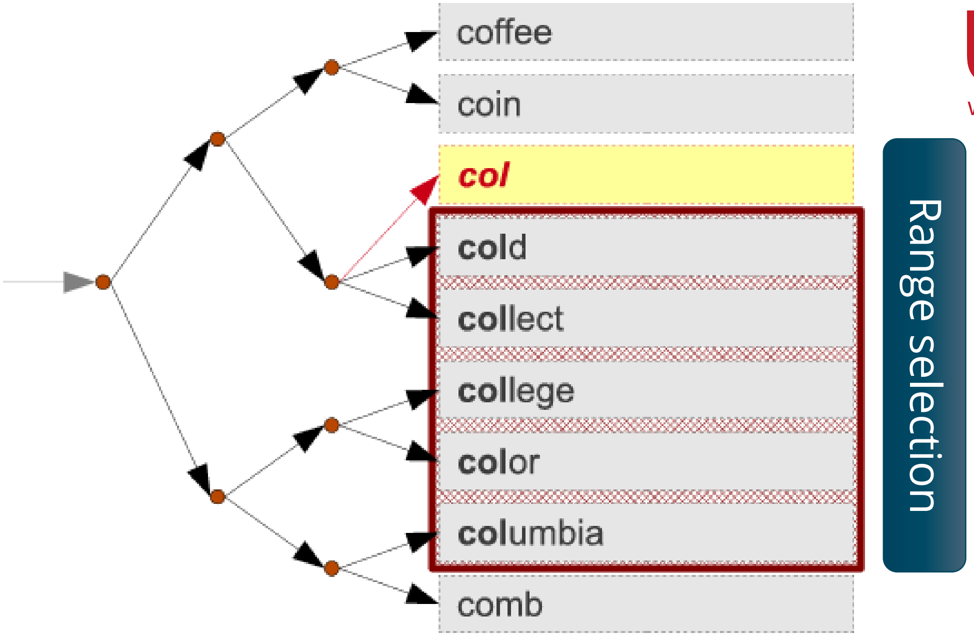
› 4. Wildcard Searches

Wildcard Search

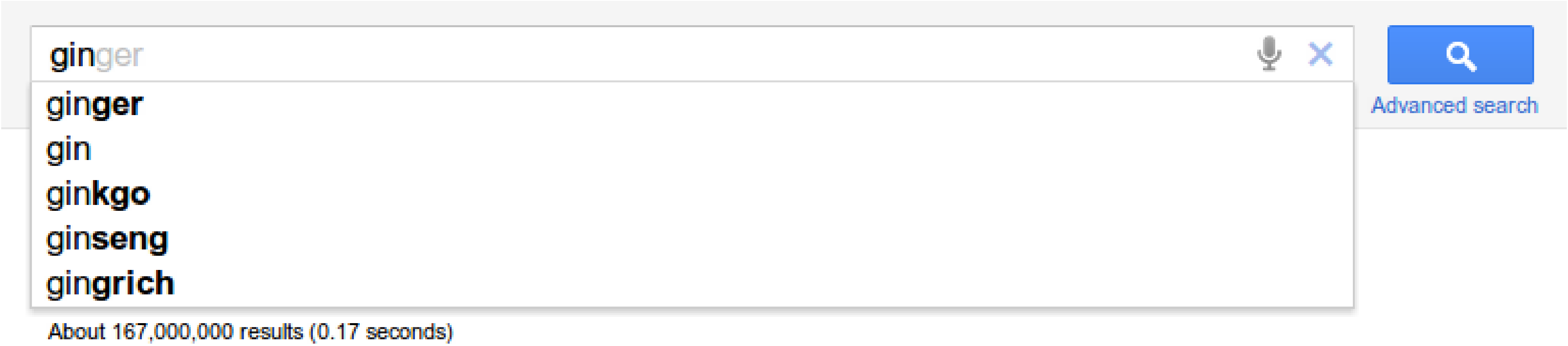
- Flexible search
  - Leave out fragments of terms, mark with wildcard
    - Question mark (?): single character
    - Asterisk (*): zero, one or more characters
- Examples:
  - na?ve → naive, naïve
  - universit* → university, universität, universitá
  - go* → go, goes, gone
  - g?n* → gun, gone, gin,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

Wildcard

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



- Further application: Autocomplete (without ranking)



ginger

ginger
gin
ginkgo
ginseng
gingrich

About 167,000,000 results (0.17 seconds)

› 5. Summary

Summary

- 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

References

[1] https://olat.vcrp.de/url/RepositoryEntry/2565867182

[2] https://videoakademie.ko-ld.de/Panopto/Pages/Sessions/List.aspx?folderID=07fcee3e-4b21-482c-90ab-ab9500ec2019

[3] http://west.uni-koblenz.de/studying/ws1920/machine-learning-and-data-mining

[4] https://videoakademie.ko-ld.de/Panopto/Pages/Viewer.aspx?id=545f21ba-a671-4ada-b137-ab9500f21941

[5] https://nlp.stanford.edu/IR-book/information-retrieval-book.html Christopher D. Manning, Prabhakar Raghavan and Hinrich Schütze, Introduction to Information Retrieval, Cambridge University Press. 2008.

[6] https://www.gartner.com/