# Efficiency & Scalability in IR

**Nicola Tonellotto**
ISTI-CNR, Pisa
nicola.tonellotto@isti.cnr.it

# Credits

Slides contributed by:

- Raffaele Perego (ISTI-CNR)
- Salvatore Orlando (Università Ca' Foscari Venezia)
- Iadh Ounis, Craig Macdonald (University of Glasgow)

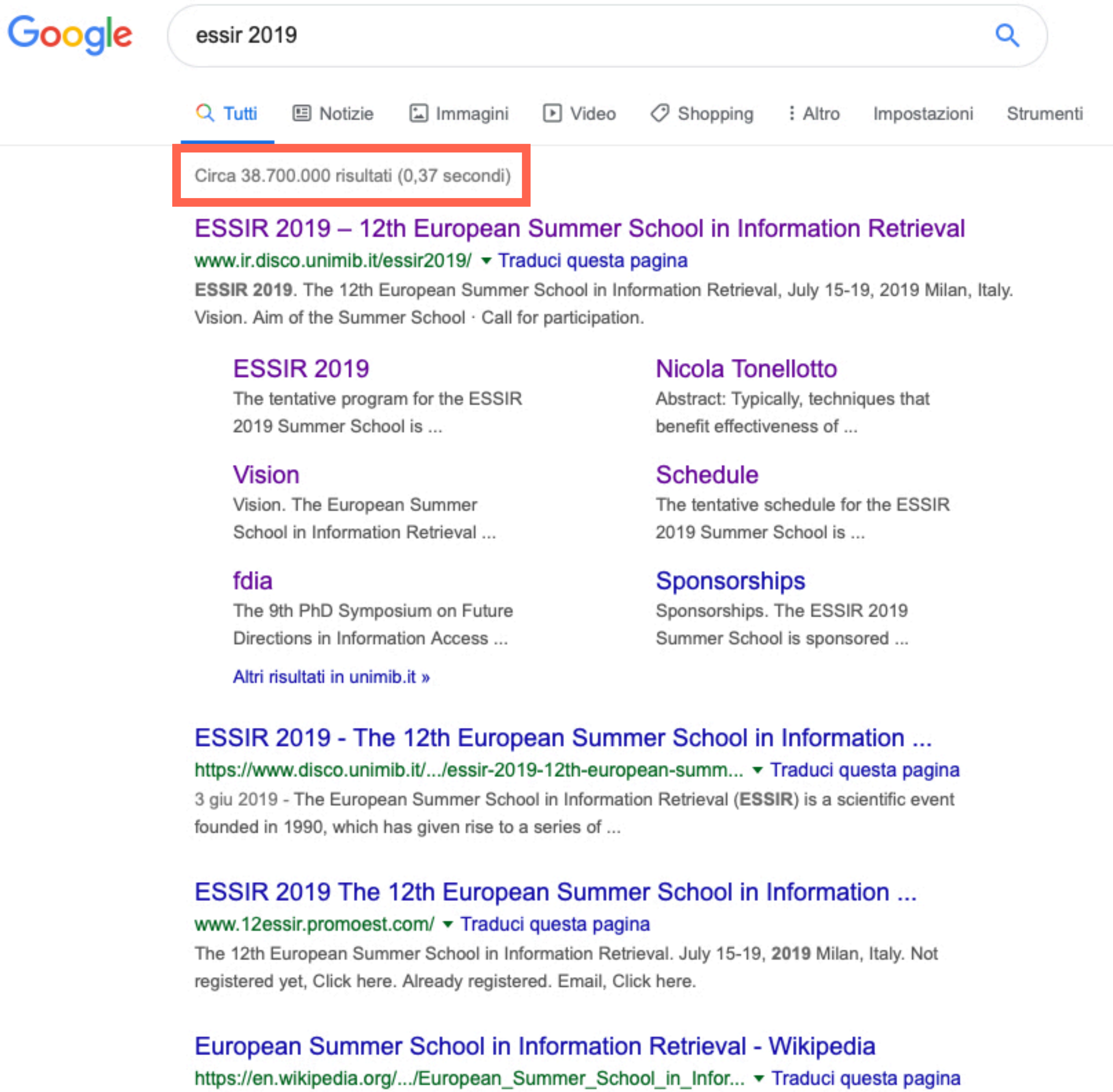
Consiglio Nazionale delle Ricerche

ISTITUTO DI SCIENZA E TECNOLOGIE DELL'INFORMAZIONE "A. FAEDO"

Big Data Grapes

EU H2020 research project grant agreement N°780751

# The scale of Web search challenge



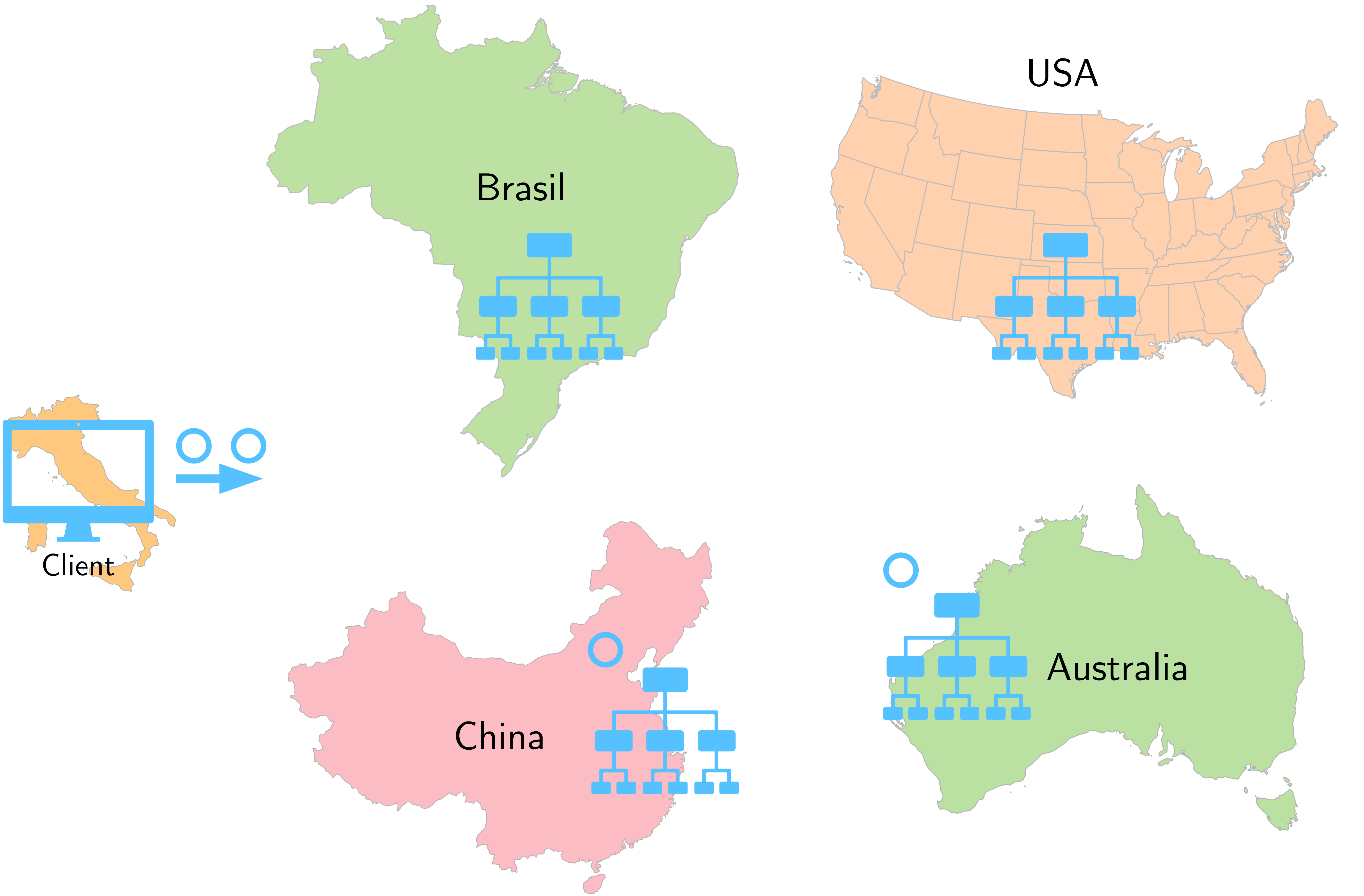
# How many documents? In how long?

- Reports suggest that Google considers a total of 30 trillion pages in the indexes of its search engine
  - And it identified relevant results from these 30 trillion in 0.63 seconds
  - Clearly this a **big data** problem!
- To answer a user's query, a search engine doesn't read through all of those pages: the **index data structures** help it to **efficiently** find pages that **effectively** match the query and will help the user
  - **Effective**: users want relevant search results
  - **Efficient**: users aren't prepared to wait a long time for search results

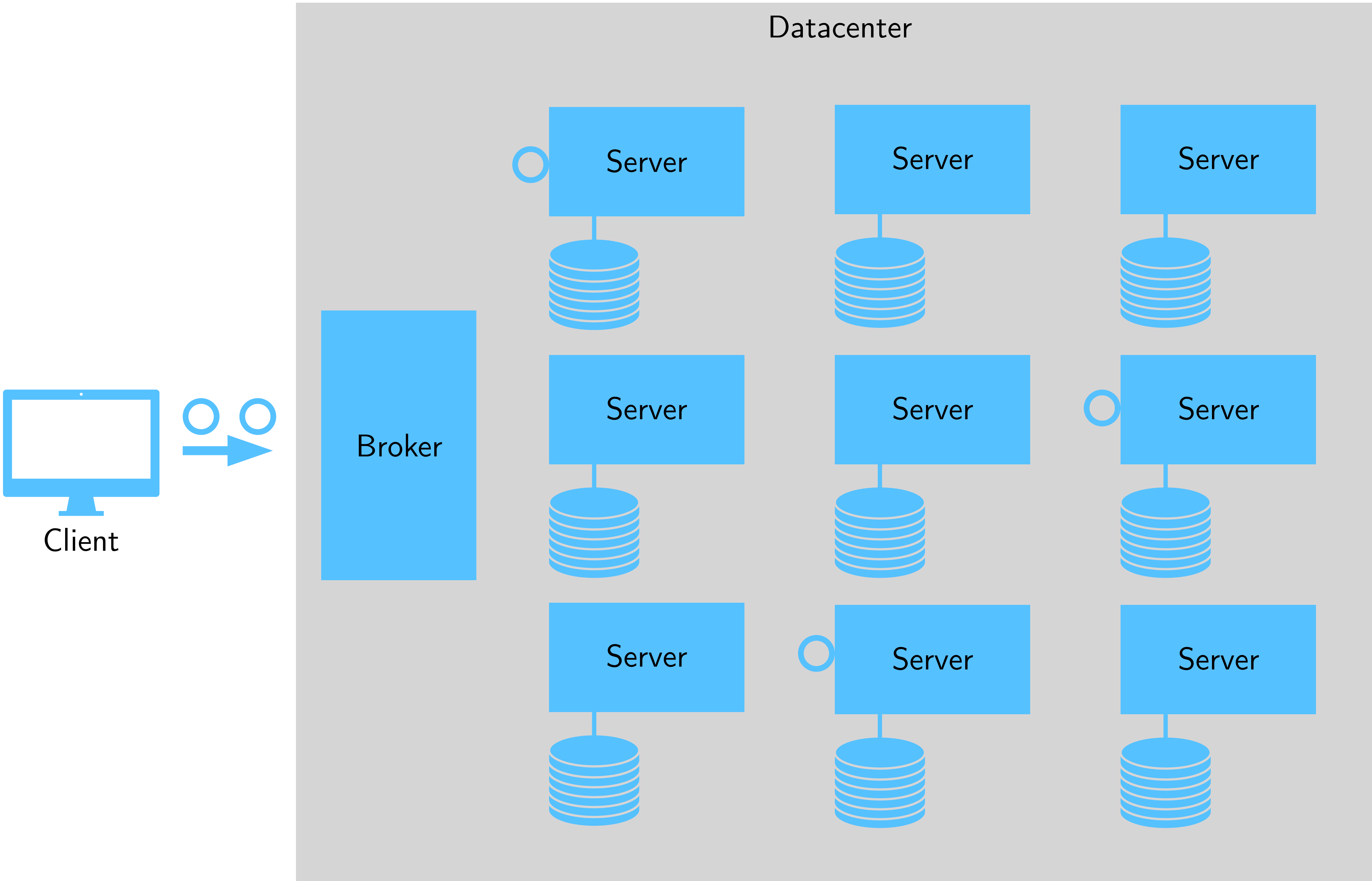
# How to measure efficiency?

- Space occupancy
  - Total space in MB/GB
  - Average space in bpi (bits per integer)
- Processed postings
  - Number of processed items
  - Platform and implementation independent
- Response time
  - User perspective
  - Mean, median
  - Tail latency, 95th and 99th percentile
- Throughput
  - System perspective
  - QPS (queries per second)

# Search as a Distributed Problem



# Search as a Distributed Problem

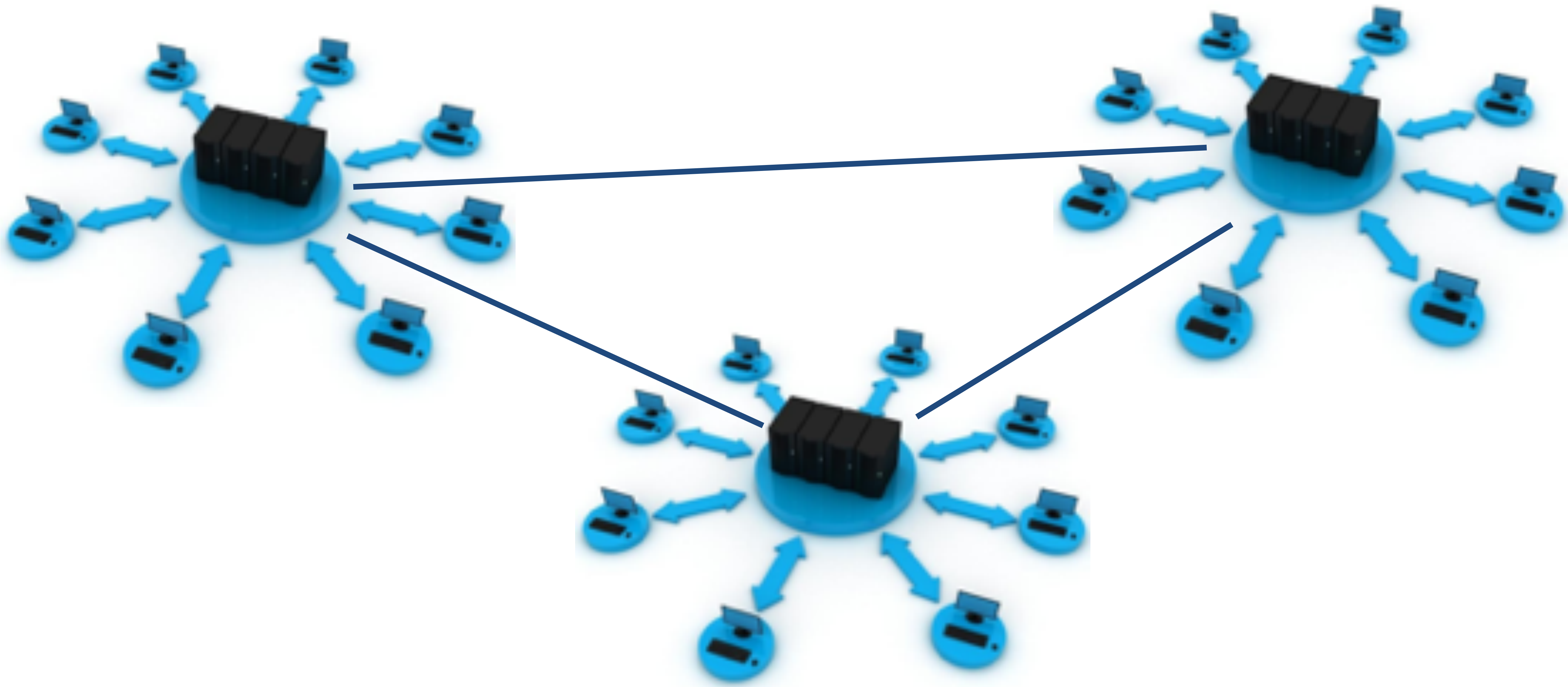


# Search engines are growing

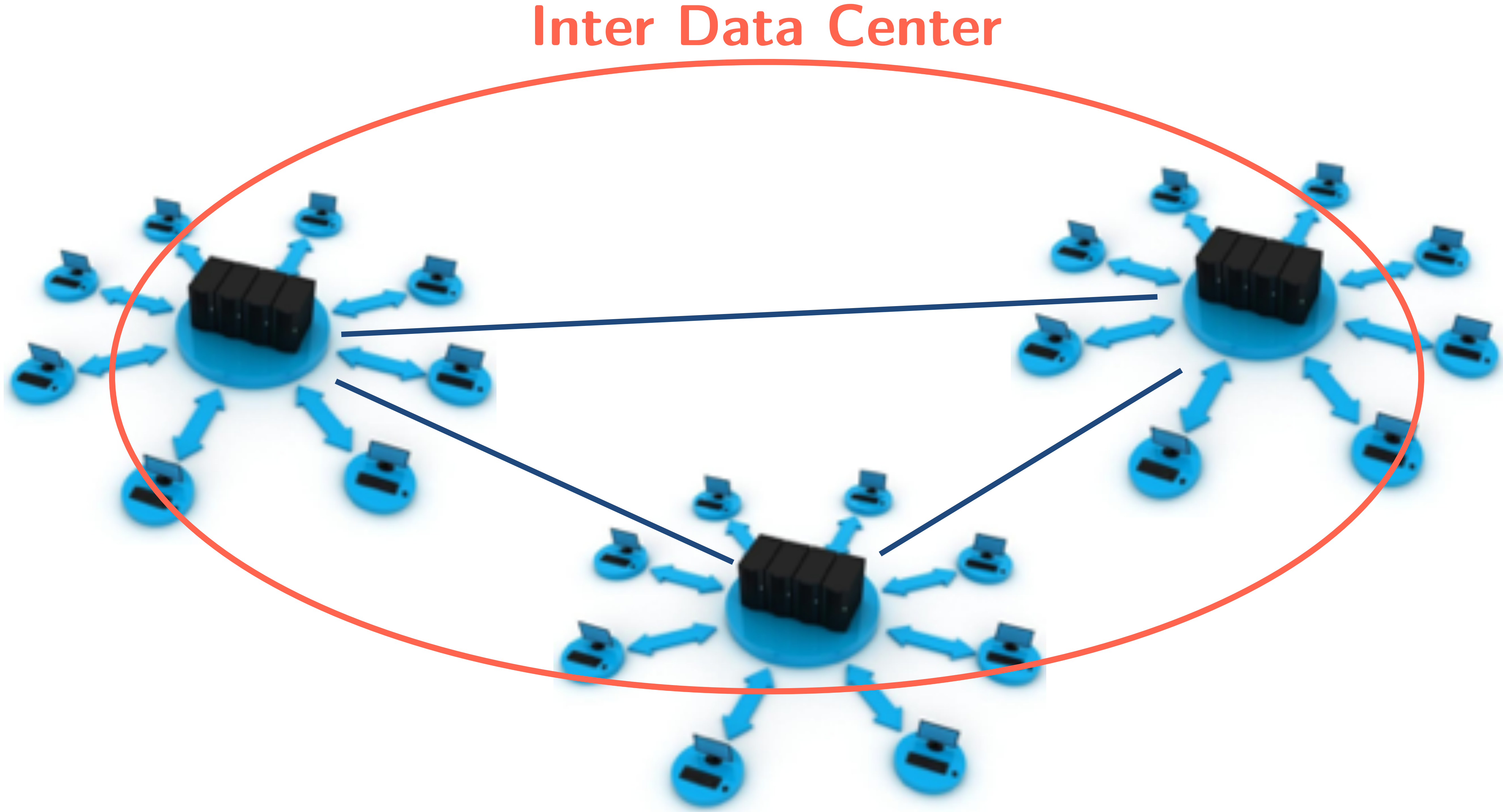
- More queries $\Rightarrow$ More servers $\Rightarrow$ $$$
- More documents $\Rightarrow$ More servers $\Rightarrow$ $$$
- More users $\Rightarrow$ More advertising $\Rightarrow$ $$$

- Reduce $$$ $\Rightarrow$ Reduce resources $\Rightarrow$ Improve **efficiency**
- Increase $$$ $\Rightarrow$ Improve results $\Rightarrow$ Improve **effectiveness**

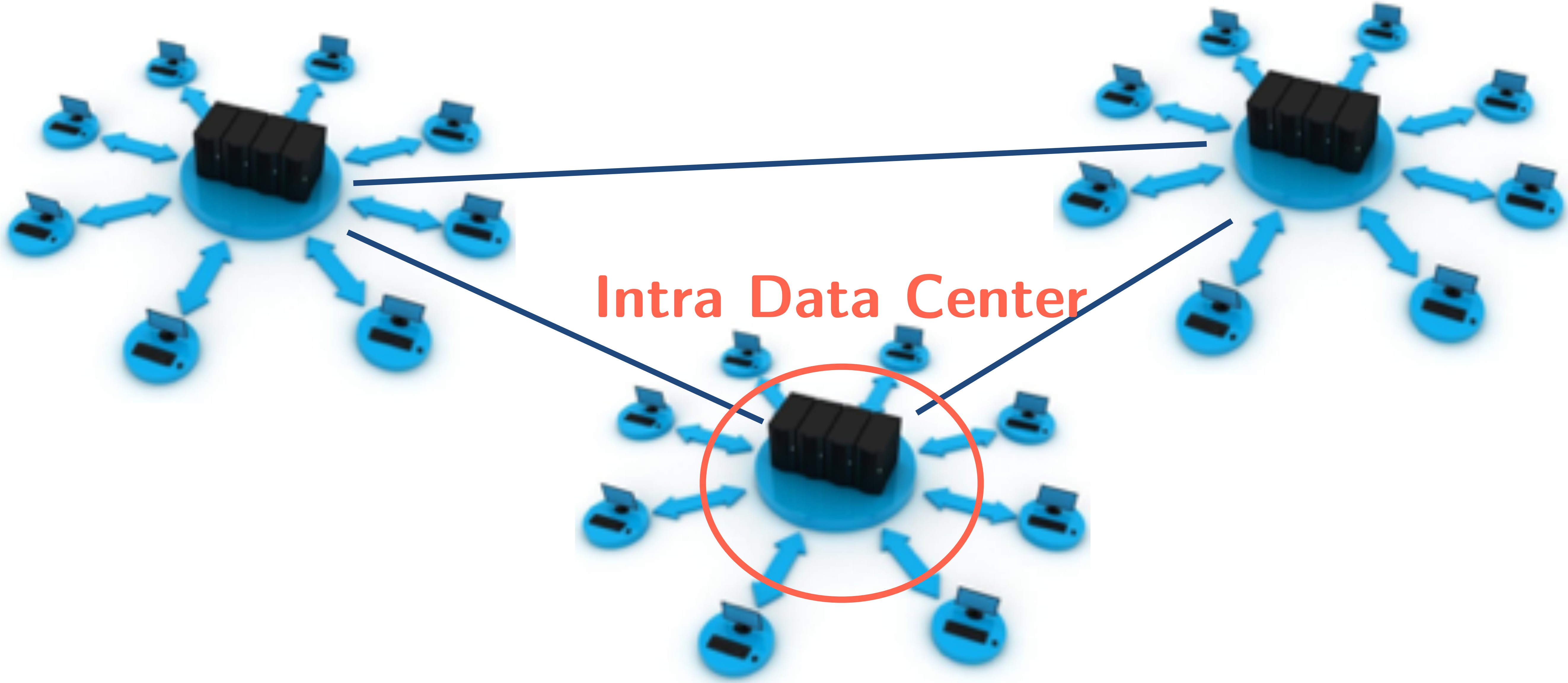
# Where is our focus?



# Where is our focus?

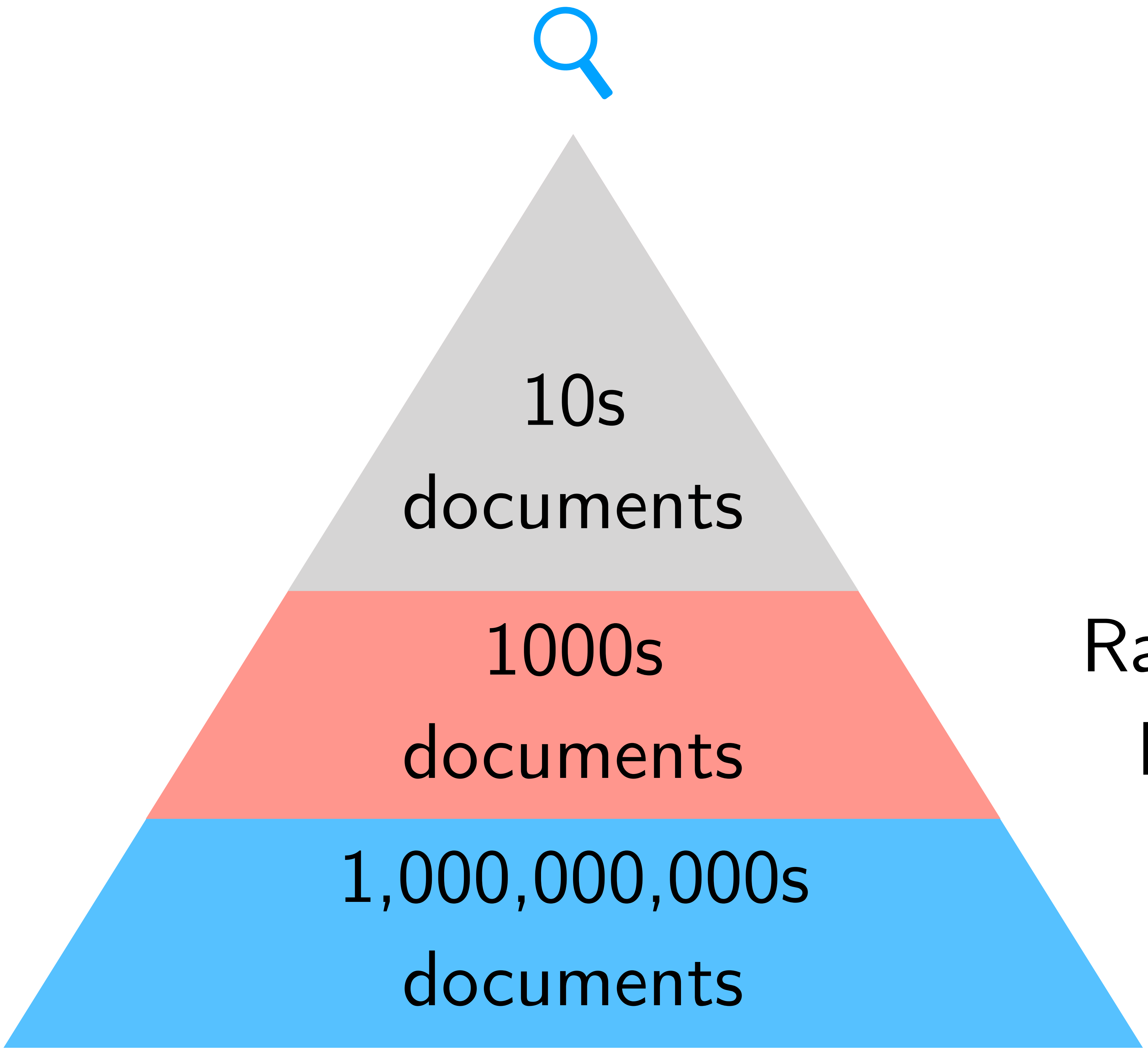


# Where is our focus?



# Where is our focus?

# Roadmap




Learning to rank

Ranked retrieval &
Dynamic pruning

Index layout &
Boolean retrieval

Pedersen, J. "Query Understanding at Bing"., SIGIR 2010

# Format of an index

- Typically, an **index** contains 3 data structures:
  - A **lexicon**: to store unique terms and their statistics

| term | df | cf | • |
|---|---|---|---|

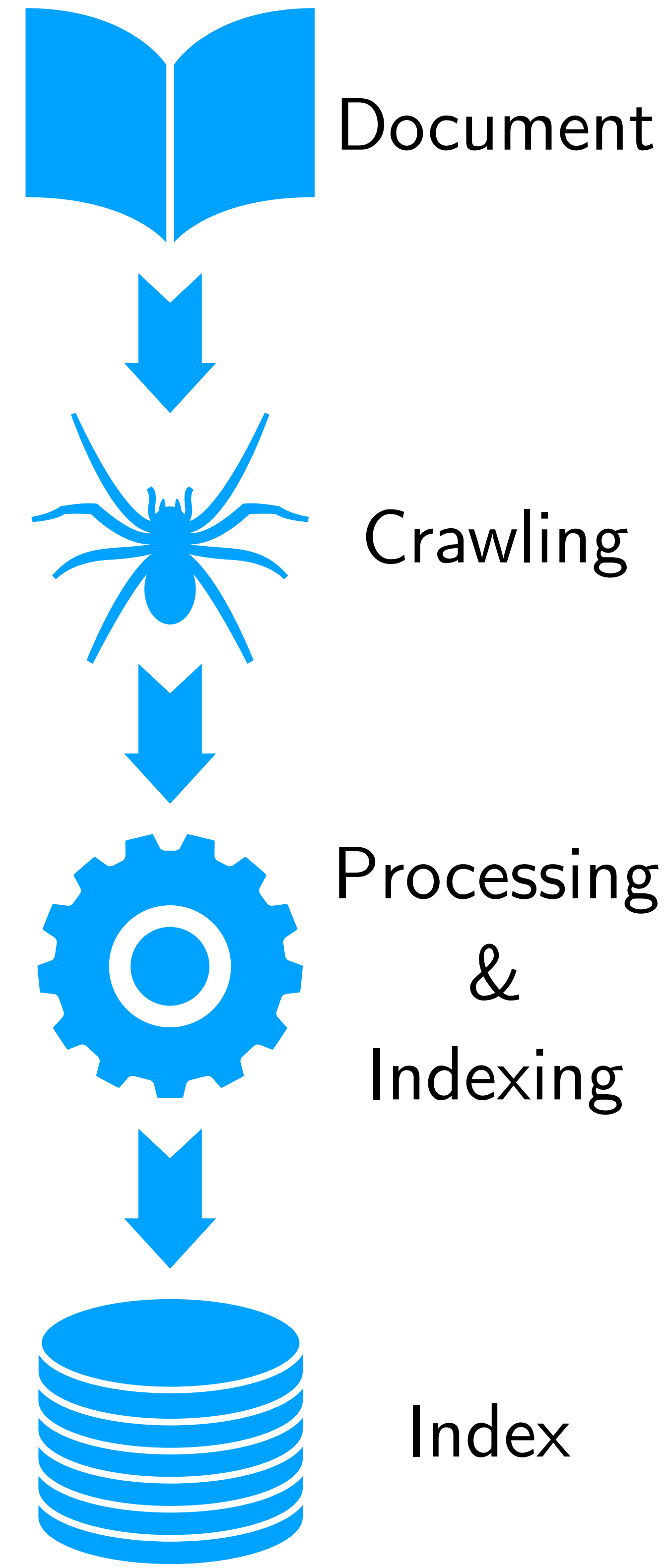
  - A **document index**: to store documents and their statistics

| docid | len | PR | URL |
|---|---|---|---|

  - An **inverted index**: to store term-document statistics (in posting lists)

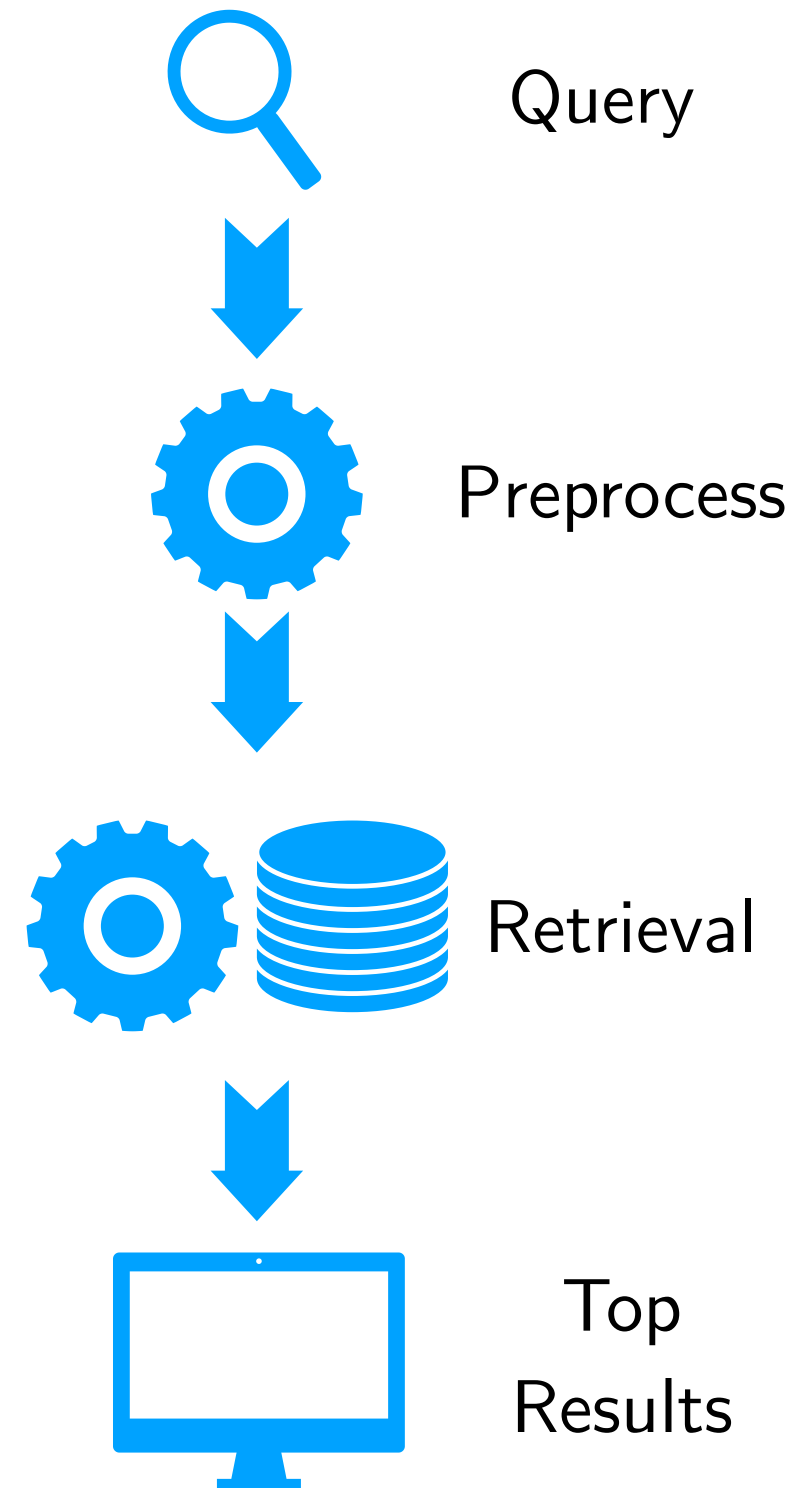
| id | tf | id | tf | id | tf | id | tf |
|---|---|---|---|---|---|---|---|

posting



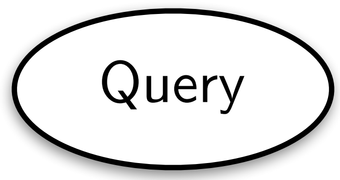
# Query processing

- It is important to make retrieval **as fast as possible**
  - Research by MS Bing indicates that even slightly slower retrieval (0.2 – 0.4 sec) can lead to a **dramatic drop** in the perceived quality of the results
- Why is document scoring expensive?
  - **Trillions of pages**
- The cost of a search depends on
  - The **query length** (number of terms)
  - The **posting list length** (for each query term)



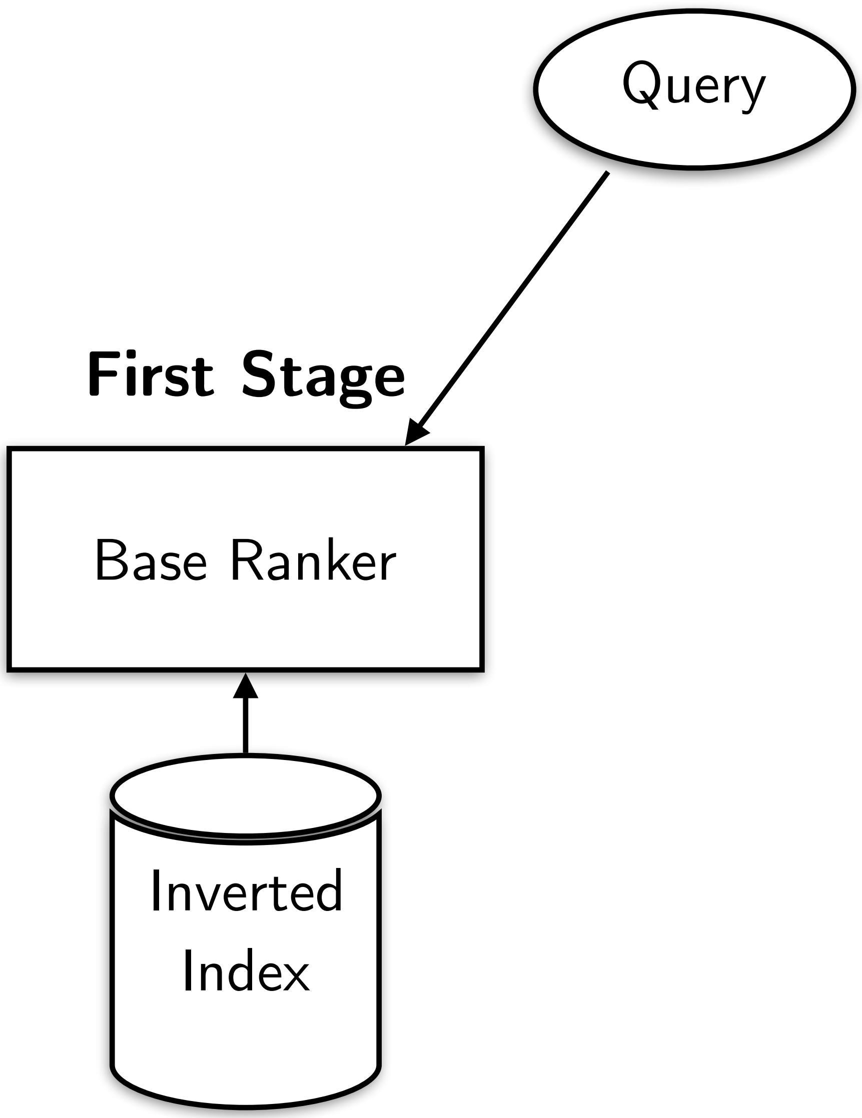

Arapakis, I., X. Bai, and B. B. Cambazoglu. "Impact of Response Latency on User Behavior in Web Search", SIGIR 2014

# Cascading



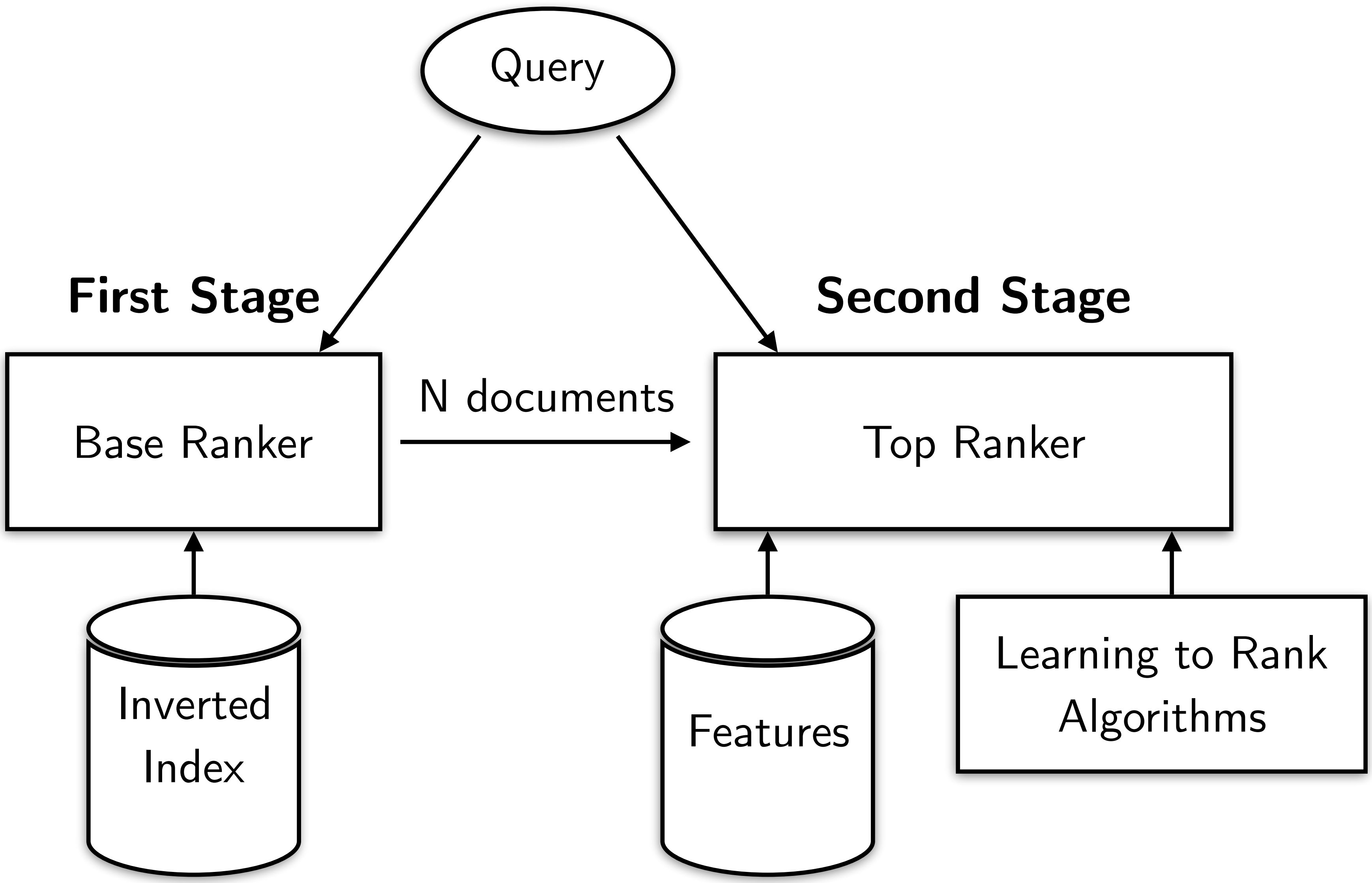

Tonellotto, N., C. Macdonald, and I. Ounis. "Efficient and Effective Retrieval Using Selective Pruning", WSDM 2013

# Cascading



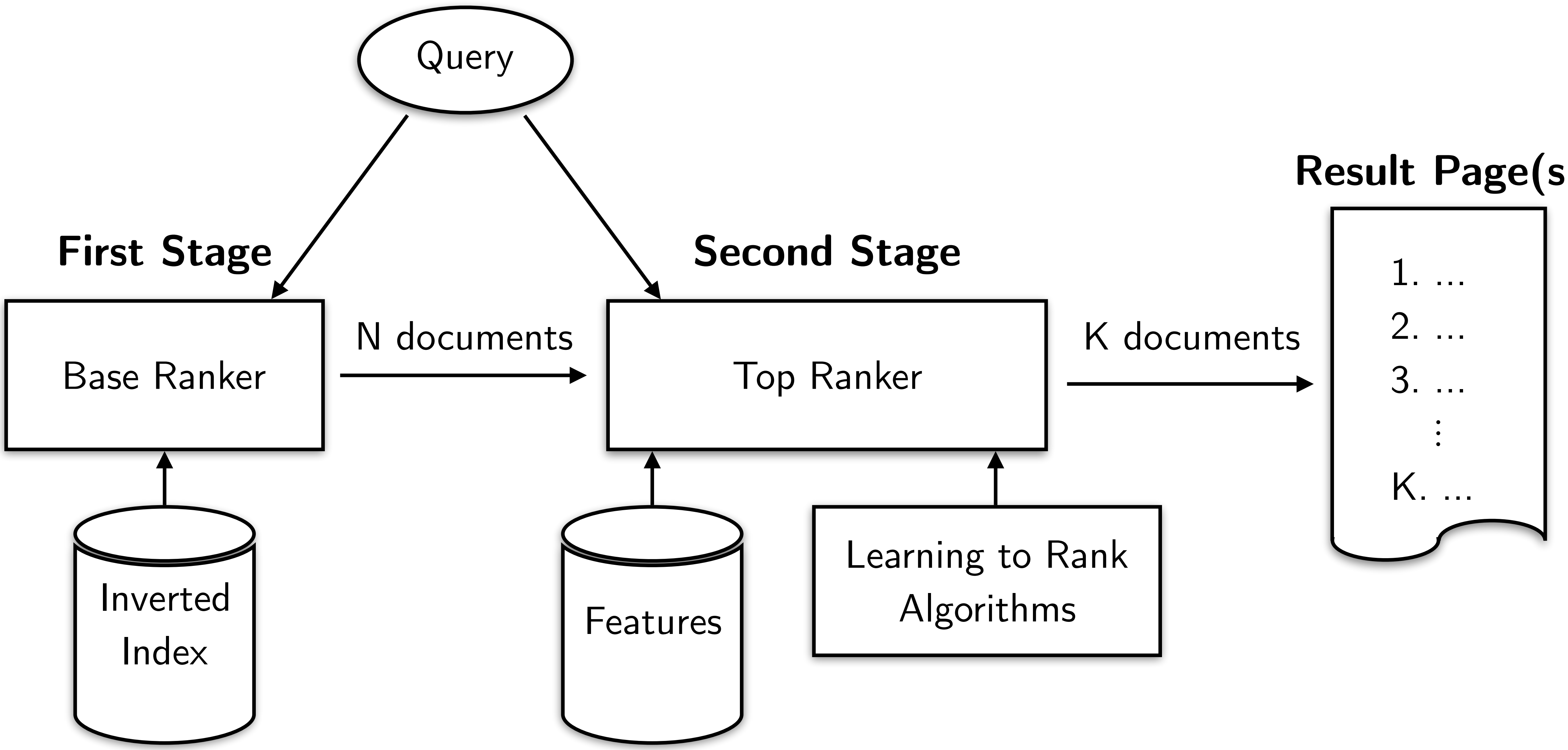

Tonellotto, N., C. Macdonald, and I. Ounis. "Efficient and Effective Retrieval Using Selective Pruning", WSDM 2013

# Cascading



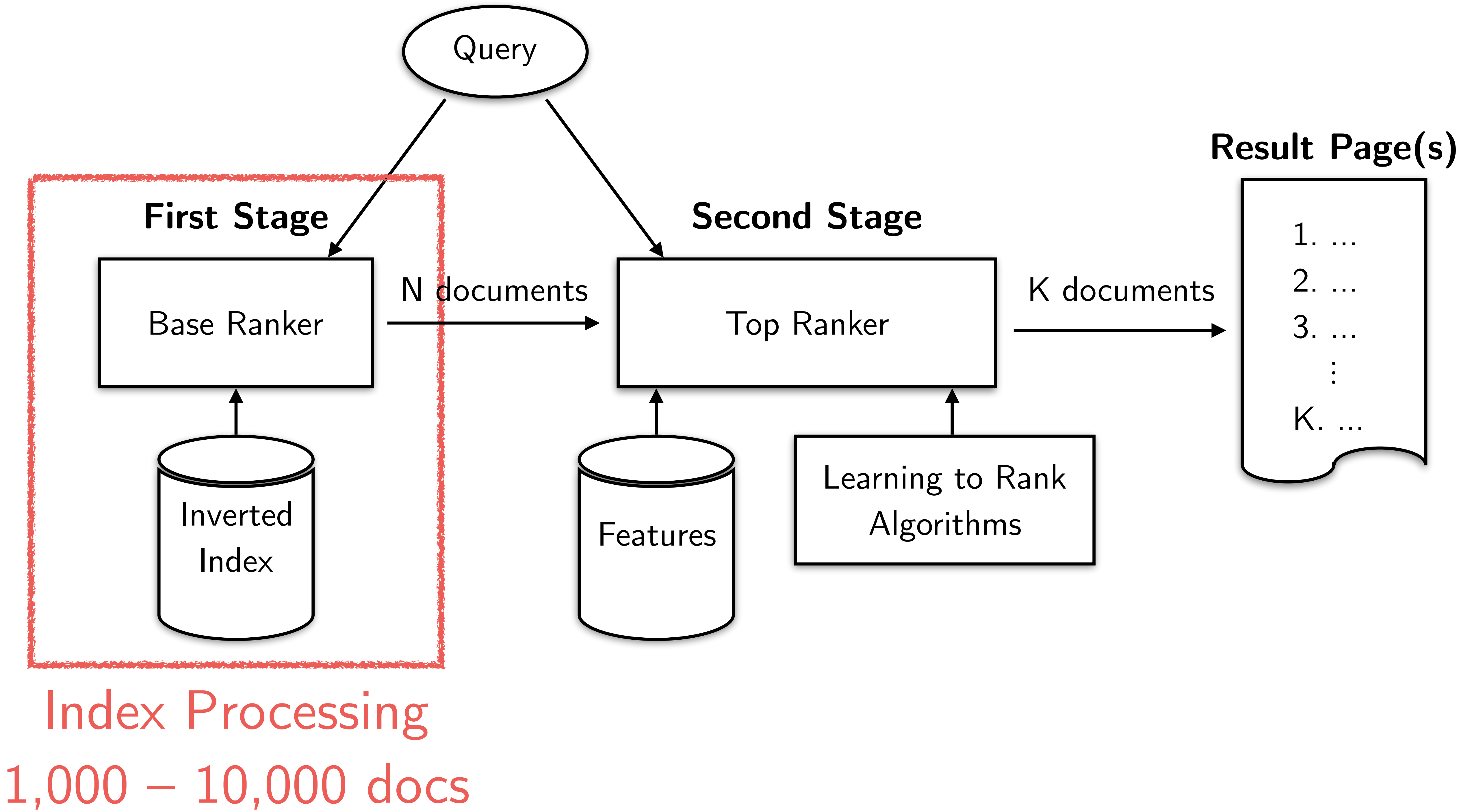

Tonellotto, N., C. Macdonald, and I. Ounis. "Efficient and Effective Retrieval Using Selective Pruning", WSDM 2013

# Cascading



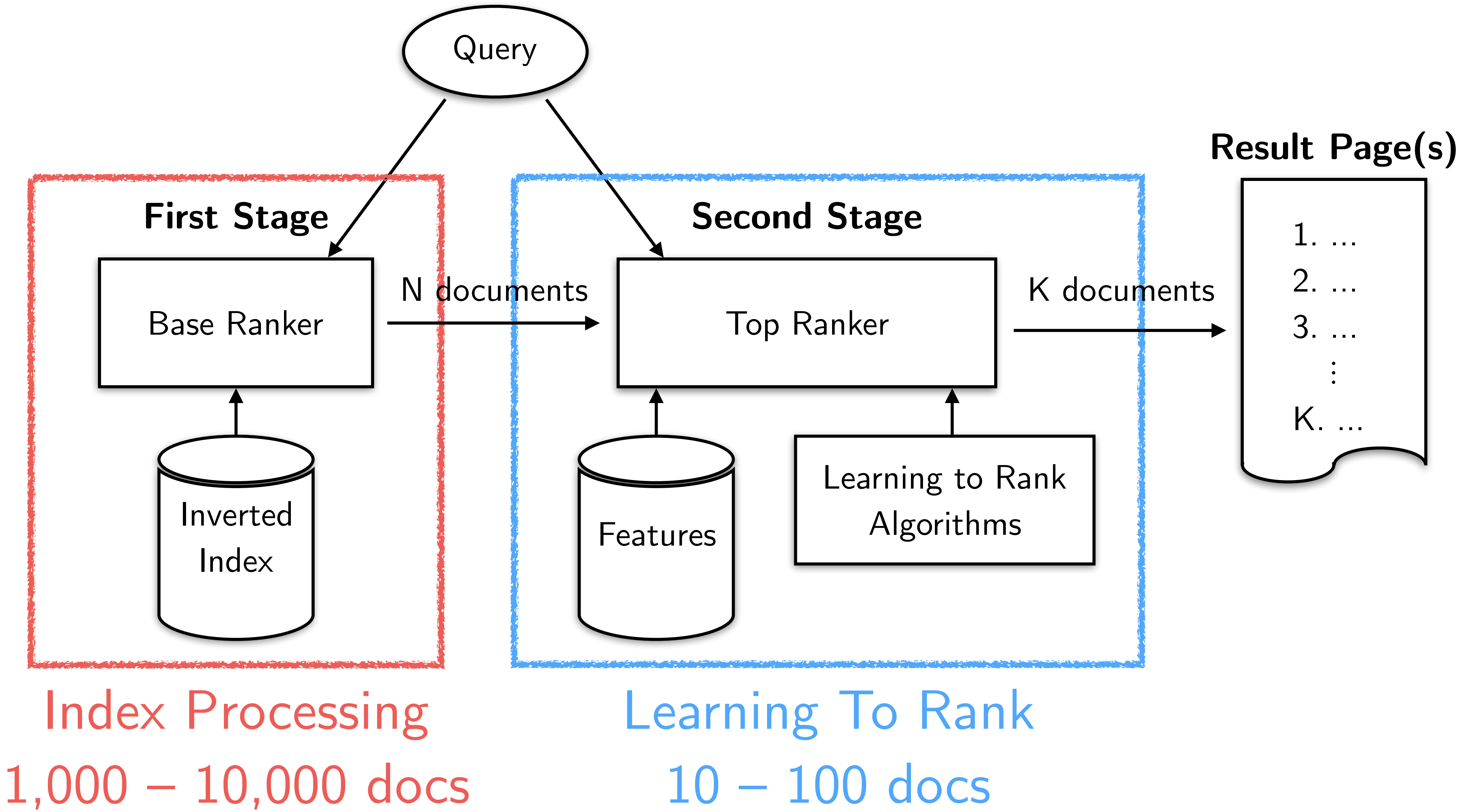

Tonellotto, N., C. Macdonald, and I. Ounis. "Efficient and Effective Retrieval Using Selective Pruning", WSDM 2013

# Cascading




Tonellotto, N., C. Macdonald, and I. Ounis. "Efficient and Effective Retrieval Using Selective Pruning", WSDM 2013

# Cascading



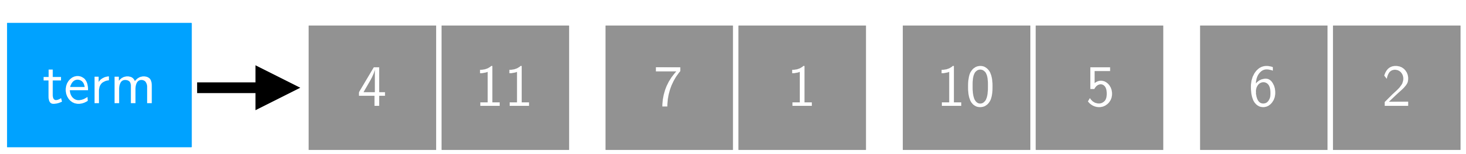

Tonellotto, N., C. Macdonald, and I. Ounis. "Efficient and Effective Retrieval Using Selective Pruning", WSDM 2013

# Outline

- Compression
- Query processing
- Dynamic pruning
- Impact-sorted indexes
- Learning to rank & cascading

# Index Compression

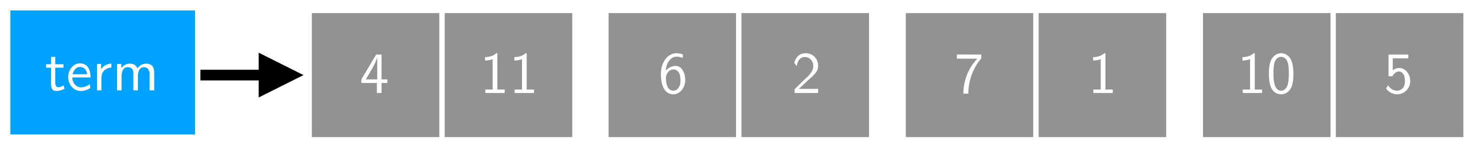
- It takes time to read the posting lists, in particular if they are stored on a (slow) hard disk
- Using **lossless compressed layouts** for the posting lists can **save space** (on disk or in memory) and reduce the amount of time spent doing IO
- But decompression can also be expensive, it can **cost time**
- **Space-time tradeoff**



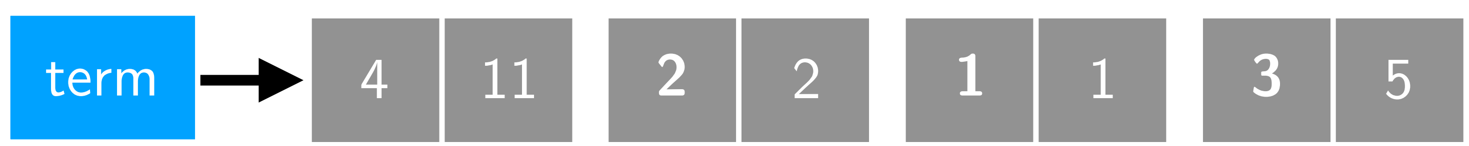

8 ints
32 bits per int
256 bits in total
Do we need all of them?

# Delta Gaps


- Docids can be stored in ascending order




- Taking the differences gives smaller numbers (a.k.a. dgaps)




- Each dgap could be represented using less bits
  - 32 bits are too many, we will get a bunch of 0s.

Zobel, J. and A. Moffat. 2006. "Inverted Files for Text Search Engines". ACM Comp. Surv. 38(2).

# Integer Compression

Treat each unsigned integer independently and encode it

- **Unary:** use as many 0s as the input value $d$ followed by a 1.

$$(5)_{10} \mapsto 000001$$

- **Gamma:** let $N = \lfloor \log_2 d \rfloor$ the highest power of 2 contained in $d$; write $N$ in unary and $d - 2^N$ in binary on $N$ bits.

$$(7)_{10} = (2^2 + 3)_{10} \mapsto 00111$$

- **Variable byte**: $(824)_{10} = (1100111000)_2 =$

$$= (0000110|0111000)_2 \mapsto 0000110\ 10111000$$

- Other compressors: delta, gamma, zeta, Huffman, Golomb, ...

# List-adaptive Compression

Documents are often **clustered** in the inverted index, e.g., by URL ordering; compression can be more effective if we consider **blocks of integers** and encode them

- **Frame of reference**: consider blocks of consecutive integers of fixed/variable size and compress them
- **Simple**: pack as many integers as possible in a memory word, i.e., 32 or 64 bits.
- **Elias-Fano**: suitable for monotonically increasing sequences of integers
- Other variants and implementations: BIC, QMX, PEF

Pibiri, G. E., R. Venturini. "Algorithms for the Inverted Index Compression", Arxiv 2019 (to appear)

# Posting List Iterators

- It is often convenient to see a posting list as an **iterator** over its postings.
- APIs:
  - `p.docid()` returns the docid of the current posting
  - `p.score()` returns the score of the current posting
  - `p.next()` moves **sequentially** the iterator to the next posting
  - `p.next(d)` advances the iterator forward to the next posting with a document identifier greater than or equal to `d` $\Rightarrow$ **skipping**

Broder, A., D. Carmel, M. Herscovici, A. Soffer, and J. Zien. "Efficient Query Evaluation using a Two-level Retrieval Process", CIKM 2003

# Query Processing

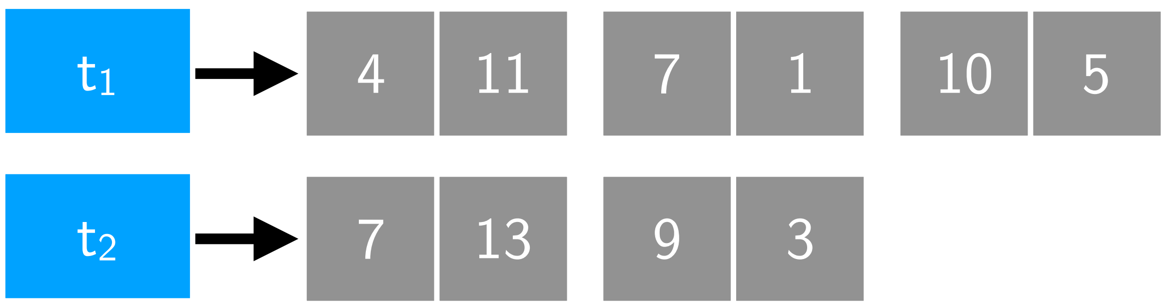
- **Conjunctive** (AND) vs. **Disjunctive** (OR)
- **Boolean retrieval**
  - Suitable for small/medium collections
  - Returns all documents matching the query
- **Ranked retrieval**
  - Suitable for Web-scale collections
  - Requires a similarity function between queries and documents

$$s_q(d) = \sum_{t \in q} s_t(q, d)$$

  - Returns only the top $K$ documents

# Term at a Time

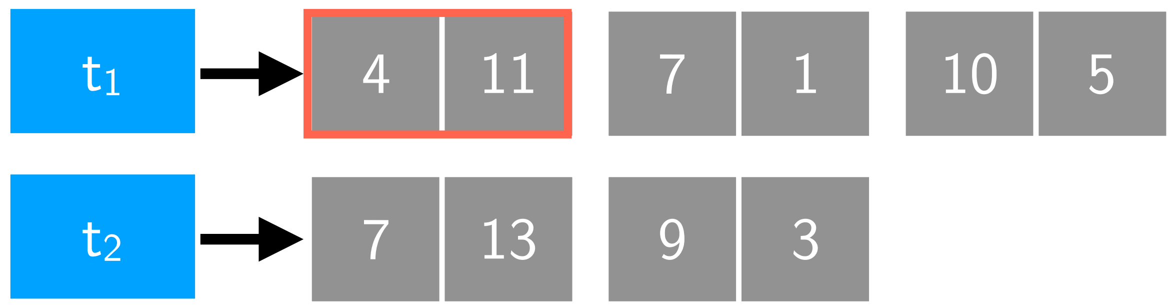
In ranked disjunctive retrieval, the posting lists can be traversed one query **term at a time** (TAAT)




Turtle, H. and J. Flood. 1995. "Query Evaluation: Strategies and Optimizations". Inf. Process. Manage. 31(6): 831–850.

# Term at a Time

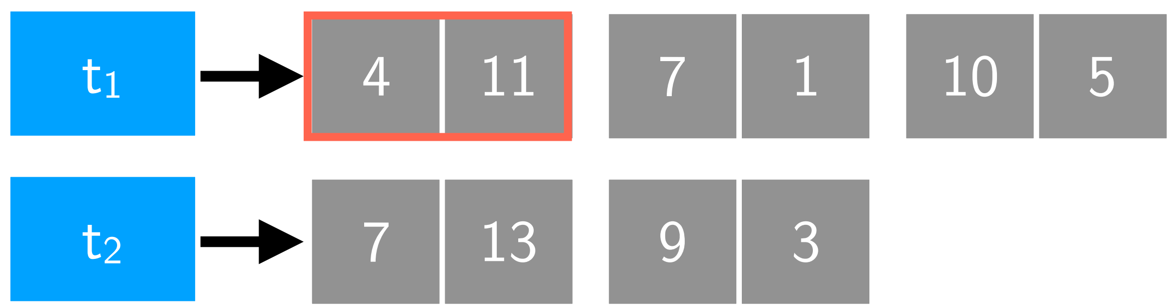
In ranked disjunctive retrieval, the posting lists can be traversed one query **term at a time** (TAAT)




Turtle, H. and J. Flood. 1995. "Query Evaluation: Strategies and Optimizations". Inf. Process. Manage. 31(6): 831–850.

# Term at a Time

In ranked disjunctive retrieval, the posting lists can be traversed one query **term at a time** (TAAT)

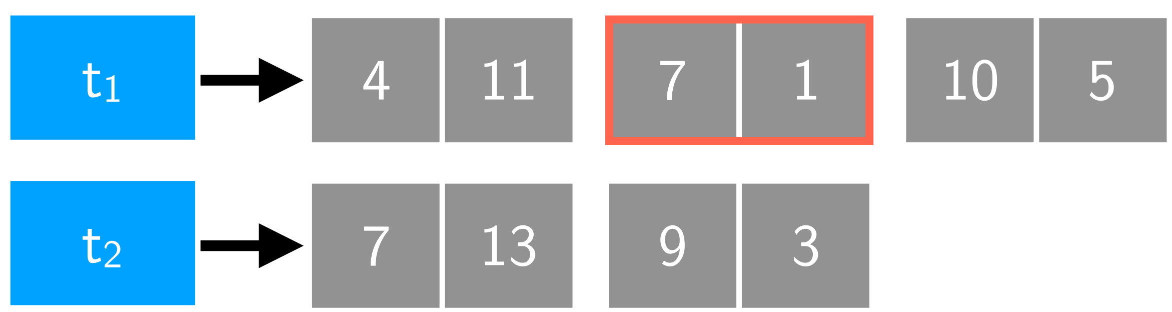



| rank | docid | score |
|---|---|---|
| 1 | 4 | 11 |
| 2 | | |
| 3 | | |
| 4 | | |

Turtle, H. and J. Flood. 1995. "Query Evaluation: Strategies and Optimizations". Inf. Process. Manage. 31(6): 831–850.

# Term at a Time

In ranked disjunctive retrieval, the posting lists can be traversed one query **term at a time** (TAAT)

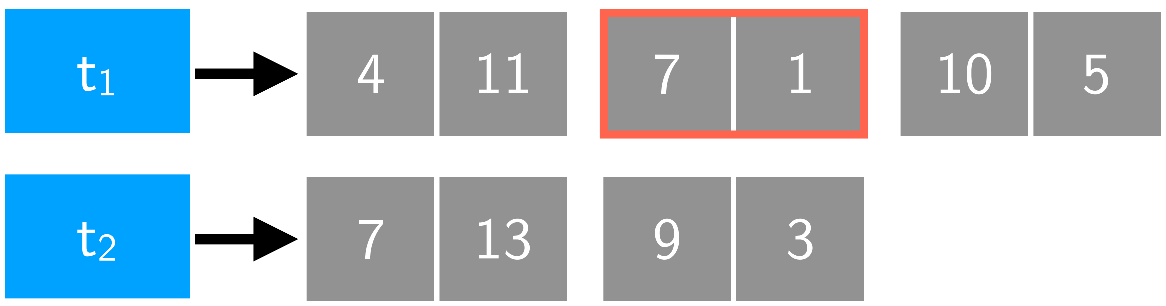



| rank | docid | score |
|---|---|---|
| 1 | 4 | 11 |
| 2 | | |
| 3 | | |
| 4 | | |

Turtle, H. and J. Flood. 1995. "Query Evaluation: Strategies and Optimizations". Inf. Process. Manage. 31(6): 831–850.

# Term at a Time

In ranked disjunctive retrieval, the posting lists can be traversed one query **term at a time** (TAAT)

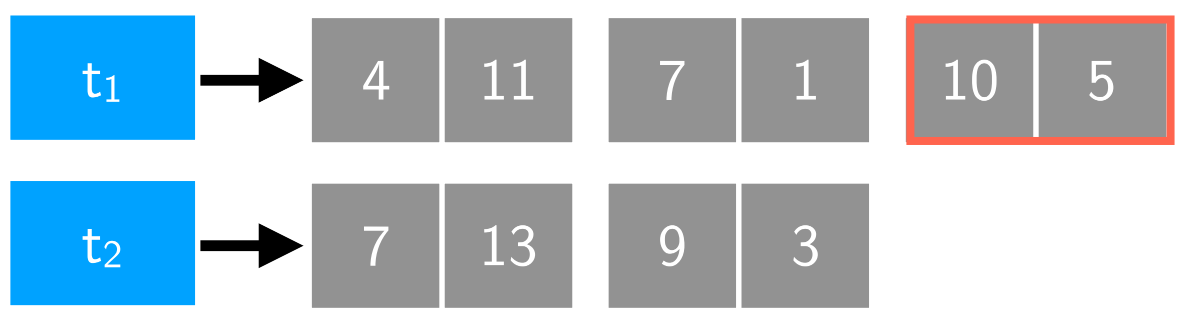



| rank | docid | score |
|---|---|---|
| 1 | 4 | 11 |
| 2 | 7 | 1 |
| 3 | | |
| 4 | | |

Turtle, H. and J. Flood. 1995. "Query Evaluation: Strategies and Optimizations". Inf. Process. Manage. 31(6): 831–850.

# Term at a Time

In ranked disjunctive retrieval, the posting lists can be traversed one query **term at a time** (TAAT)

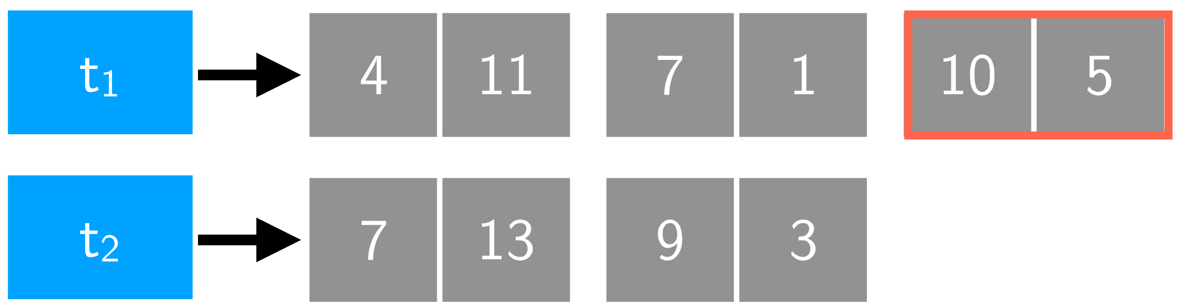



| rank | docid | score |
|---|---|---|
| 1 | 4 | 11 |
| 2 | 7 | 1 |
| 3 | | |
| 4 | | |

Turtle, H. and J. Flood. 1995. "Query Evaluation: Strategies and Optimizations". Inf. Process. Manage. 31(6): 831–850.

# Term at a Time

In ranked disjunctive retrieval, the posting lists can be traversed one query **term at a time** (TAAT)

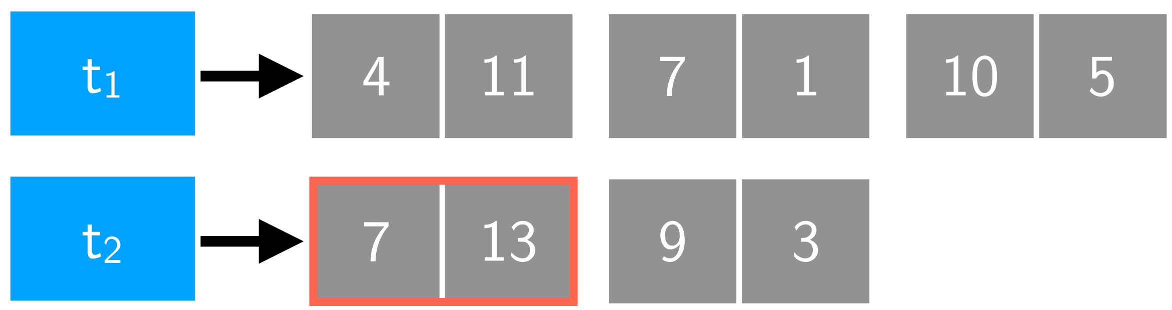



| rank | docid | score |
|---|---|---|
| 1 | 4 | 11 |
| 2 | 10 | 5 |
| 3 | 7 | 1 |
| 4 | | |

Turtle, H. and J. Flood. 1995. "Query Evaluation: Strategies and Optimizations". Inf. Process. Manage. 31(6): 831–850.

# Term at a Time

In ranked disjunctive retrieval, the posting lists can be traversed one query **term at a time** (TAAT)

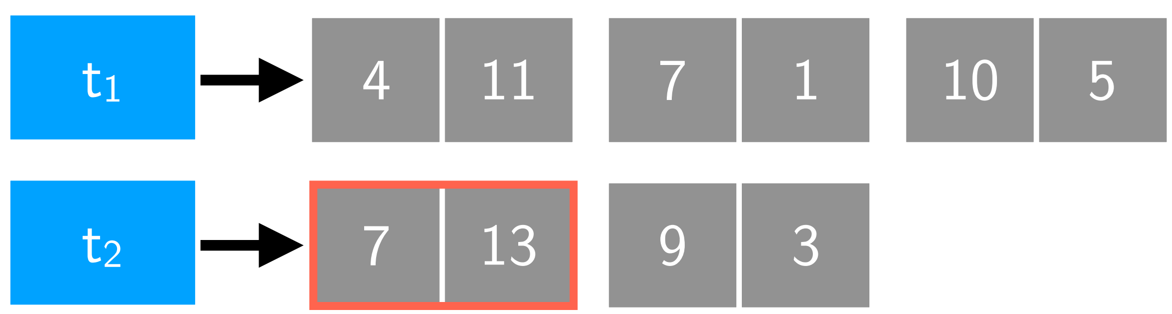



| rank | docid | score |
|---|---|---|
| 1 | 4 | 11 |
| 2 | 10 | 5 |
| 3 | 7 | 1 |
| 4 | | |

Turtle, H. and J. Flood. 1995. "Query Evaluation: Strategies and Optimizations". Inf. Process. Manage. 31(6): 831–850.

# Term at a Time

In ranked disjunctive retrieval, the posting lists can be traversed one query **term at a time** (TAAT)

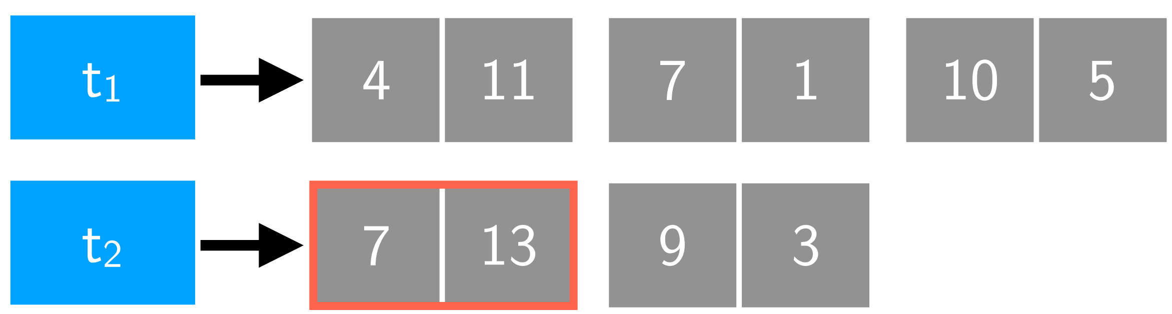



| rank | docid | score |
|---|---|---|
| 1 | 7 | 14 |
| 2 | 4 | 11 |
| 3 | 10 | 5 |
| 4 | | |

Turtle, H. and J. Flood. 1995. "Query Evaluation: Strategies and Optimizations". Inf. Process. Manage. 31(6): 831–850.

# Term at a Time

In ranked disjunctive retrieval, the posting lists can be traversed one query **term at a time** (TAAT)




| rank | docid | score |
|---|---|---|
| 1 | 7 | 14 |
| 2 | 4 | 11 |
| 3 | 10 | 5 |
| 4 | | |

## PROS

- Simple
- Cache-friendly

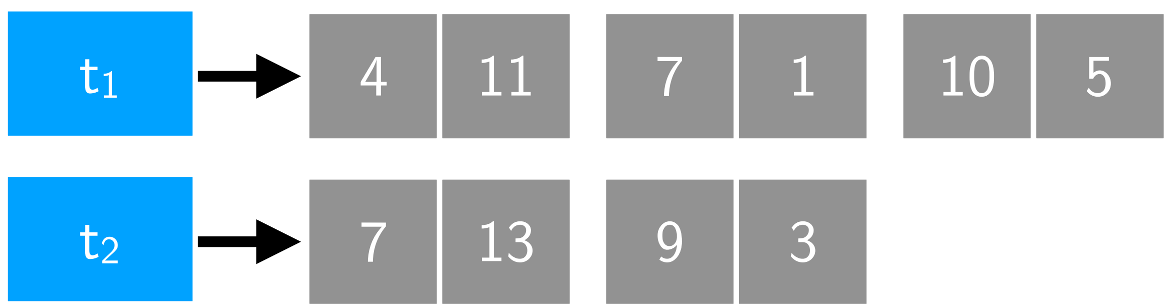
## CONS

- Requires lot of memory to contain partial scores for all documents
- Difficult to do boolean or phrasal queries

Turtle, H. and J. Flood. 1995. "Query Evaluation: Strategies and Optimizations". Inf. Process. Manage. 31(6): 831–850.

# Document at a Time

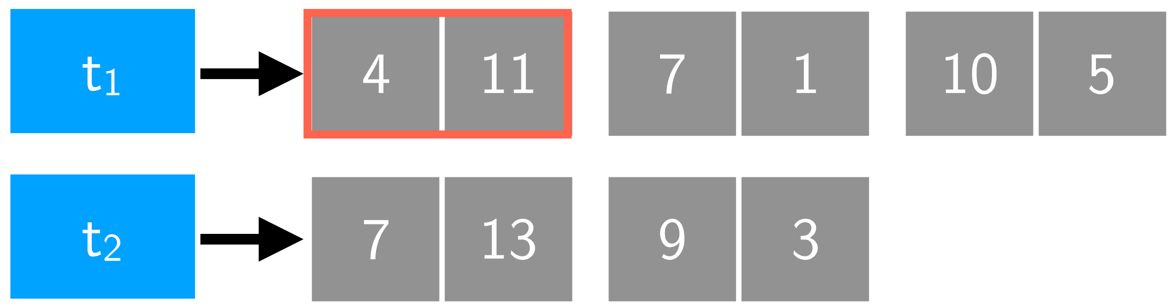
In ranked disjunctive retrieval, the posting lists can be traversed all query terms in parallel, to fully process a **document at a time** (DAAT)




Turtle, H. and J. Flood. 1995. "Query Evaluation: Strategies and Optimizations". Inf. Process. Manage. 31(6): 831–850.

# Document at a Time

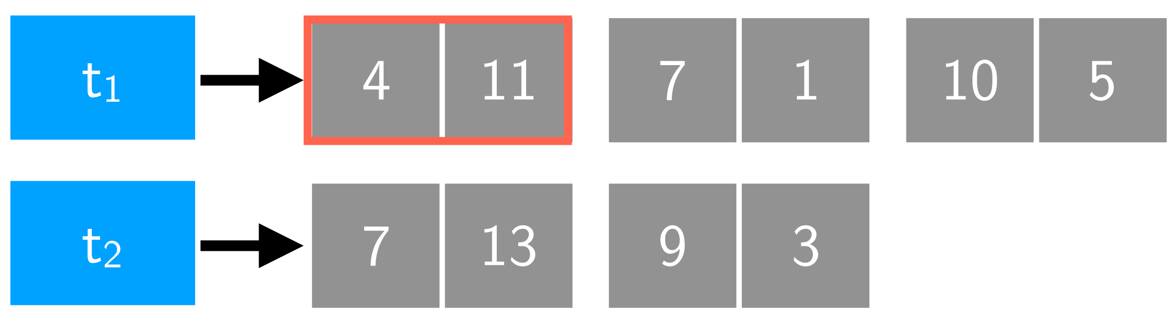
In ranked disjunctive retrieval, the posting lists can be traversed all query terms in parallel, to fully process a **document at a time** (DAAT)




Turtle, H. and J. Flood. 1995. "Query Evaluation: Strategies and Optimizations". Inf. Process. Manage. 31(6): 831–850.

# Document at a Time

In ranked disjunctive retrieval, the posting lists can be traversed all query terms in parallel, to fully process a **document at a time** (DAAT)

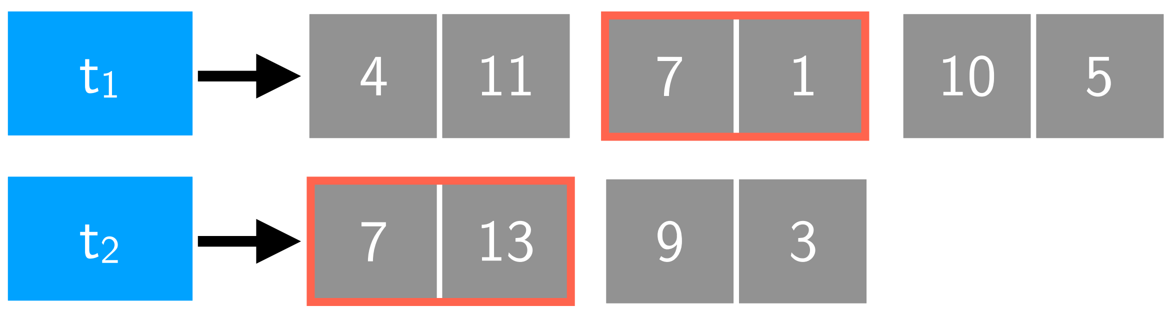



| rank | docid | score |
|---|---|---|
| 1 | 4 | 11 |
| 2 | | |
| 3 | | |
| 4 | | |

Turtle, H. and J. Flood. 1995. "Query Evaluation: Strategies and Optimizations". Inf. Process. Manage. 31(6): 831–850.

# Document at a Time

In ranked disjunctive retrieval, the posting lists can be traversed all query terms in parallel, to fully process a **document at a time** (DAAT)

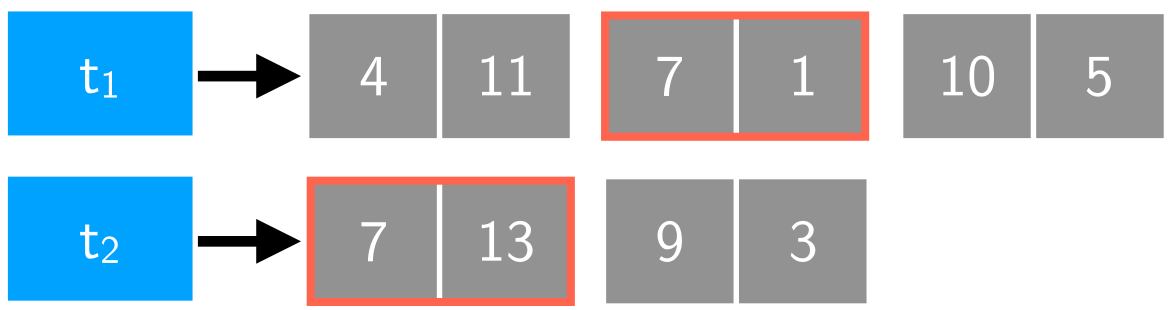



| rank | docid | score |
|---|---|---|
| 1 | 4 | 11 |
| 2 | | |
| 3 | | |
| 4 | | |

Turtle, H. and J. Flood. 1995. "Query Evaluation: Strategies and Optimizations". Inf. Process. Manage. 31(6): 831–850.

# Document at a Time

In ranked disjunctive retrieval, the posting lists can be traversed all query terms in parallel, to fully process a **document at a time** (DAAT)

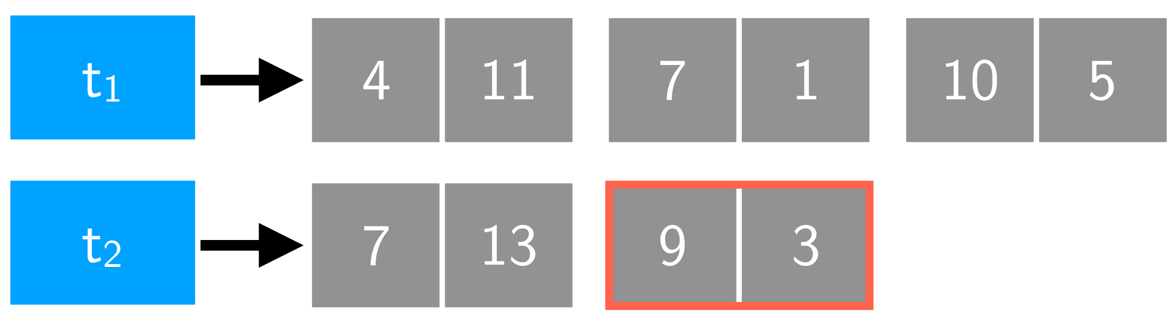



| rank | docid | score |
|---|---|---|
| 1 | 7 | 14 |
| 2 | 4 | 11 |
| 3 | | |
| 4 | | |

Turtle, H. and J. Flood. 1995. "Query Evaluation: Strategies and Optimizations". Inf. Process. Manage. 31(6): 831–850.

# Document at a Time

In ranked disjunctive retrieval, the posting lists can be traversed all query terms in parallel, to fully process a **document at a time** (DAAT)

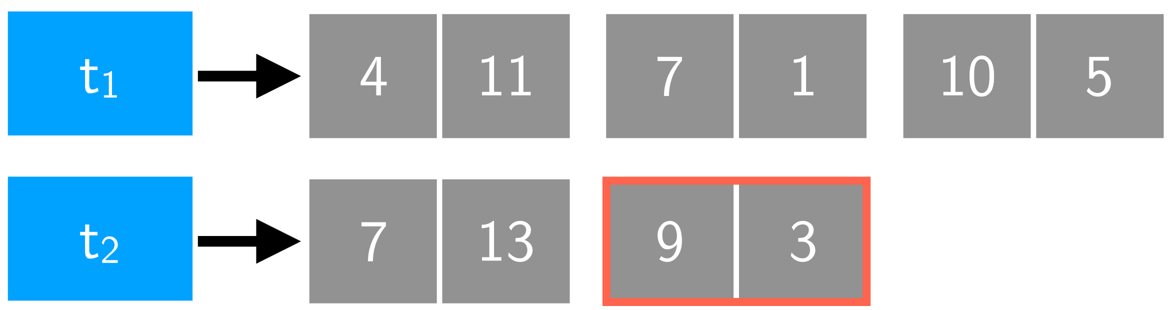



| rank | docid | score |
|---|---|---|
| 1 | 7 | 14 |
| 2 | 4 | 11 |
| 3 | | |
| 4 | | |

Turtle, H. and J. Flood. 1995. "Query Evaluation: Strategies and Optimizations". Inf. Process. Manage. 31(6): 831–850.

# Document at a Time

In ranked disjunctive retrieval, the posting lists can be traversed all query terms in parallel, to fully process a **document at a time** (DAAT)

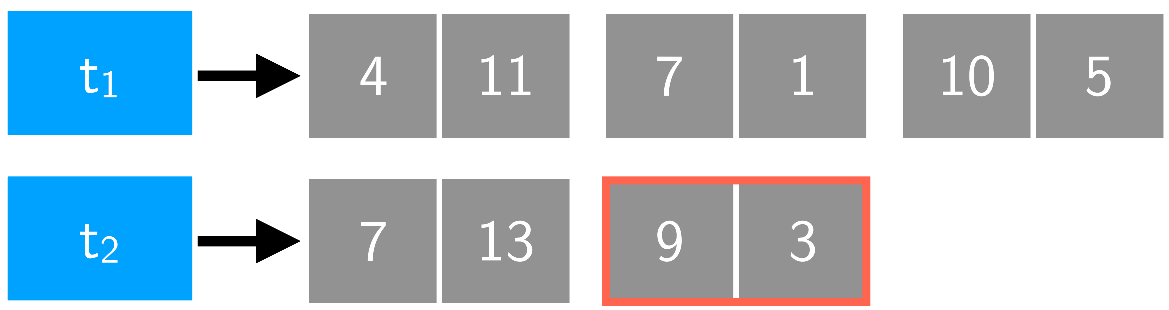



| rank | docid | score |
|---|---|---|
| 1 | 7 | 14 |
| 2 | 4 | 11 |
| 3 | 9 | 3 |
| 4 | | |

Turtle, H. and J. Flood. 1995. "Query Evaluation: Strategies and Optimizations". Inf. Process. Manage. 31(6): 831–850.

# Document at a Time

In ranked disjunctive retrieval, the posting lists can be traversed all query terms in parallel, to fully process a **document at a time** (DAAT)




| rank | docid | score |
|---|---|---|
| 1 | 7 | 14 |
| 2 | 4 | 11 |
| 3 | 9 | 3 |
| 4 | | |

## PROS

- Smaller memory footprint than TAAT
- Support boolean and phrasal queries

## CONS

- Lesser cache-friendly than TAAT

**Most commercial search engines are reported to use a variant of DAAT**

Turtle, H. and J. Flood. 1995. "Query Evaluation: Strategies and Optimizations". Inf. Process. Manage. 31(6): 831–850.

# TAAT vs DAAT

- TAAT and DAAT have been the cornerstones of query evaluation in IR systems since 1970s.
- The plain implementations of these two strategies are seldom used anymore, since many optimisations have been proposed during the years
- Fontoura et al. (2011) report experiments on small (200k docs) and large indexes (3M docs), with short (4.26 terms) and long (57.76 terms) queries (times are in microseconds):

| Small index | | |
|---|---|---|
| | Short queries | Long queries |
| TAAT | **141.0** | **1,694.6** |
| DAAT | 193.0 | 4,554.6 |
| **Large index** | | |
| | Short queries | Long queries |
| TAAT | 3,777.6 | **18,913.0** |
| DAAT | **3,581.3** | 26,778.3 |

Fontoura, M., et al. "Evaluation Strategies for Top-k Queries over Memory-Resident Inverted Indexes". VLDB 2011

# Faster Query Processing

- **Dynamic pruning** strategies aim to make scoring faster by only scoring a subset of the documents
  - The core assumption of these approaches is that the user is only interested in the **top K results**, say $K = 20$.
  - During query scoring, it is possible to determine if a document cannot make the top K ranked results.
  - Hence, the scoring of such documents can be **terminated early**, or **skipped entirely**, without damaging retrieval effectiveness to rank $K$
  - This is called "**safe-to-rank K**"
- Dynamic pruning is based upon
  - **Early termination**
  - Comparing **upper bounds** on retrieval scores with **thresholds**

# Dynamic Pruning Strategies

- **MaxScore** (Turtle & Flood, 1995)
  - **Early termination**: does not compute scores for documents that won't be retrieved by comparing upper bounds with a score threshold
- **WAND** (Broder et al., 2003)
  - **Approximate evaluation**: does not consider documents with approximate scores (sum of upper bounds) lower than threshold
  - Therefore, it focuses on the combinations of terms needed (weak AND)
- **BlockMaxWand** (Ding & Suel, 2011)
  - State-of-the-art variant of WAND that uses benefits from the **block-layout** of posting lists

# Early Termination

A document evaluation is **early terminated** if all or some of its postings, relative to the terms of the query, are not fetched from the inverted index or not scored by the ranking function.

# Term and document upper bounds

- Similarity function between queries and documents:

$$s_q(d) = \sum_{t \in q} s_t(q, d)$$

- For each term $t$ in the vocabulary, we can compute a **term upper bound** (also known as **max score**) $\sigma_t(q)$ such that, for all documents d in the posting list of term $t$:

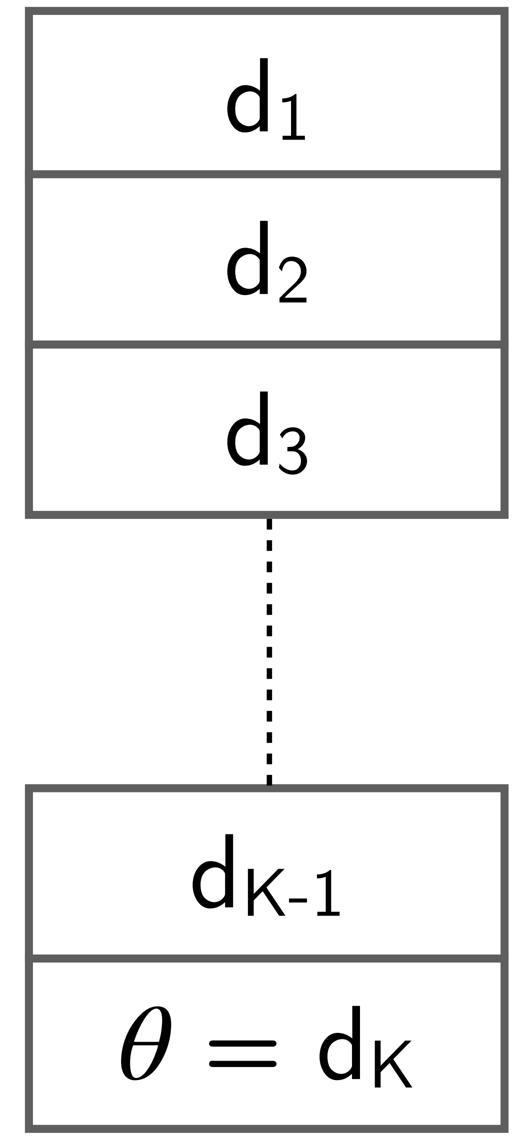
$$\sigma_t(q) \geq s_t(q, d)$$

- For a query $q$ and a document $d$, we can compute a **document upper bound** $\sigma_d(q)$ by summing up the query term upper bounds and/or actual scores:

$$\sigma_q(d) = \sum_{t \in q} \sigma_t(q, d) \qquad \text{and} \qquad \sigma_q(d) = \sum_{t \in \hat{q}} s_t(q, d) + \sum_{t \in q \setminus \hat{q}} \sigma_t(q, d)$$

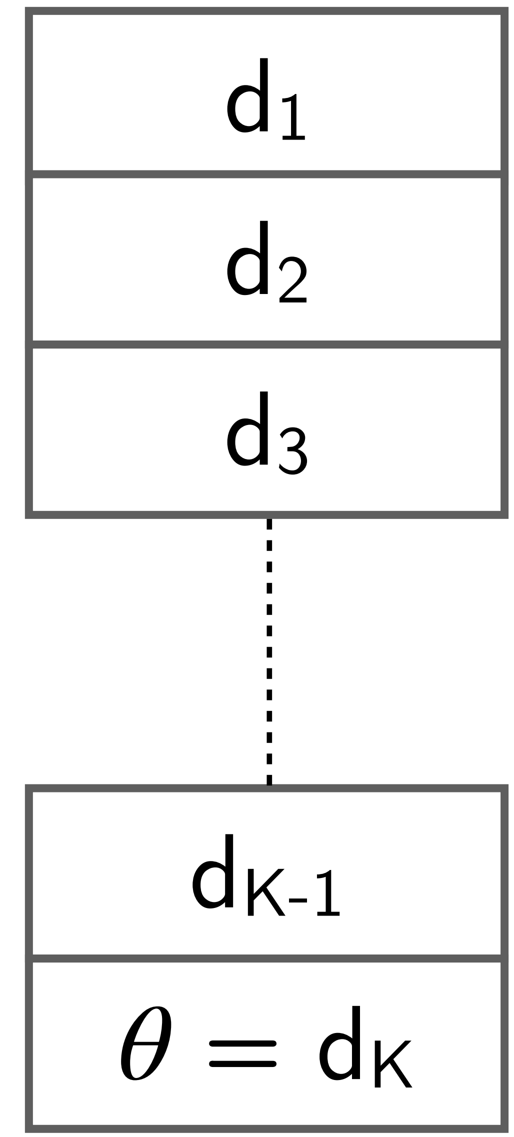
# Heap and Threshold

- During query processing, the top $K$ full or partial scores computed so far, together with the corresponding docids, are organised in a priority queue, or **min-heap**,
- The smallest value of these (partial) scores is called **threshold** $\theta$.
- If there are not at least $K$ scores, the threshold value is assumed to be 0.

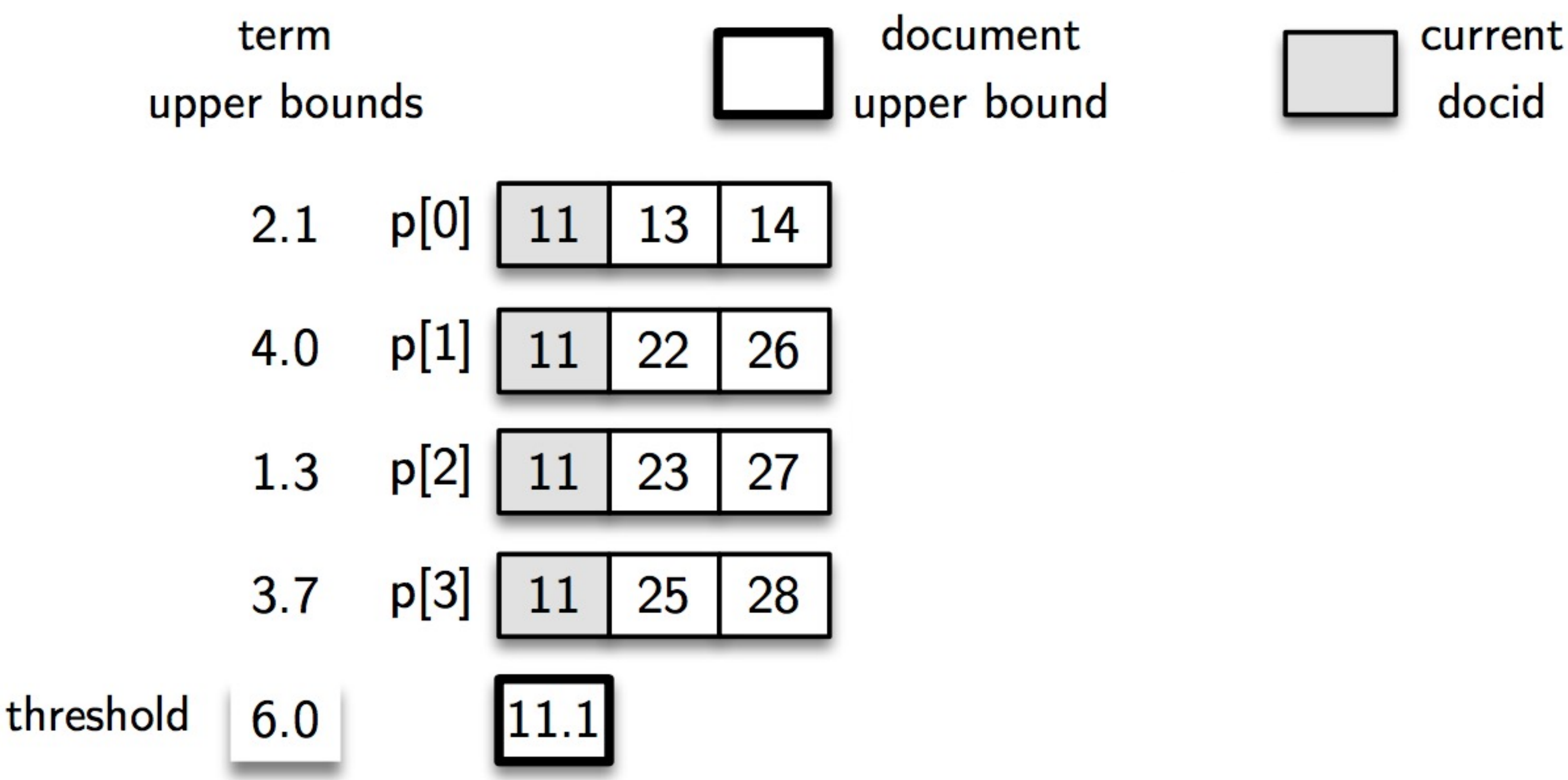


# Dynamic Pruning Condition

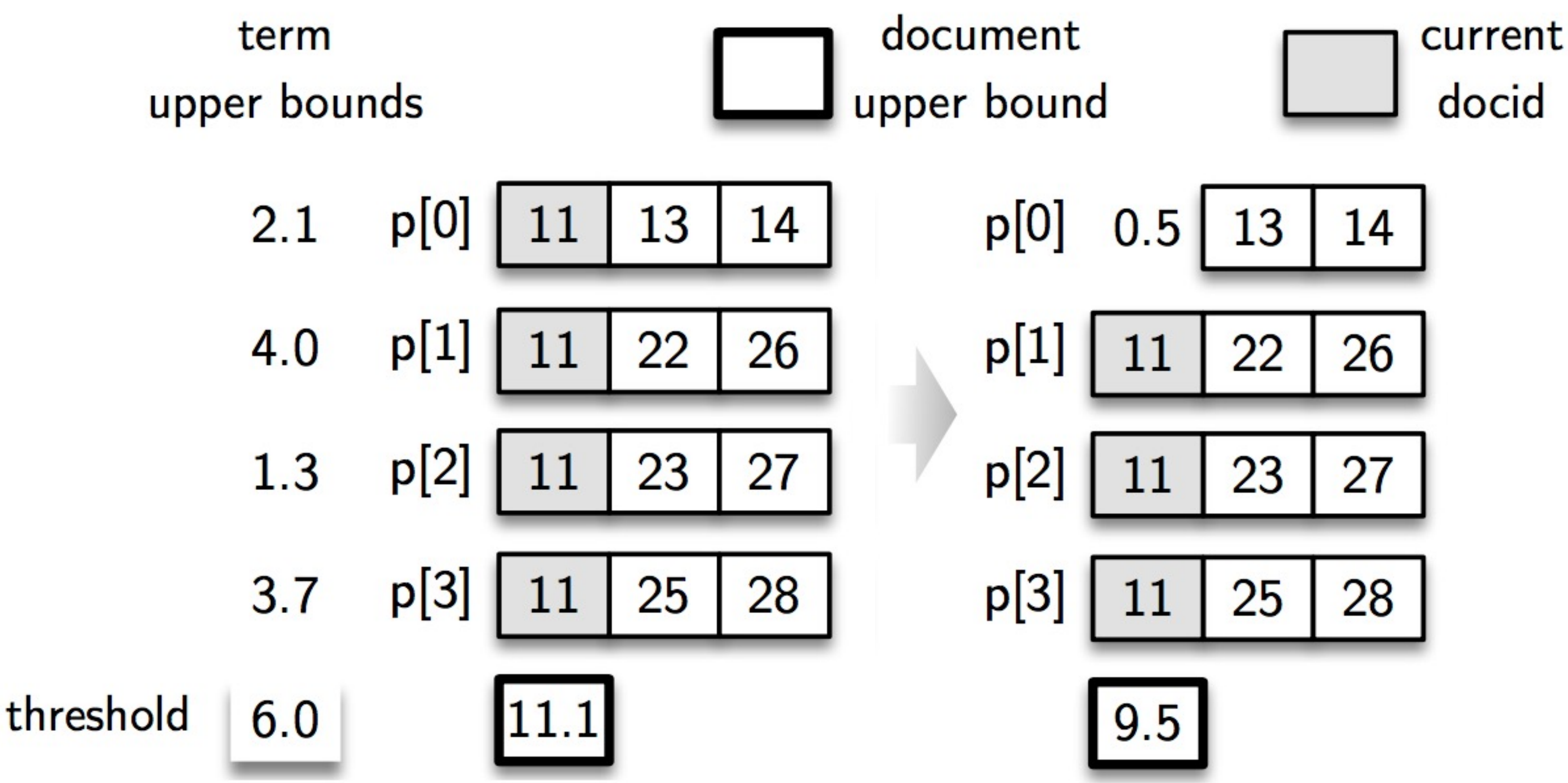
- **Property I**: The threshold value is not decreasing
- **Property II**: A document with a score smaller than the threshold will never be in final top $K$ documents
- **Pruning Condition**: for a query $q$ and a document $d$, if the document upper bound $\sigma_d(q)$, computed by using partial scores, if any, and term upper bounds, is lesser than or equal to the current threshold $\theta$, the document processing can be early terminated.



# Example

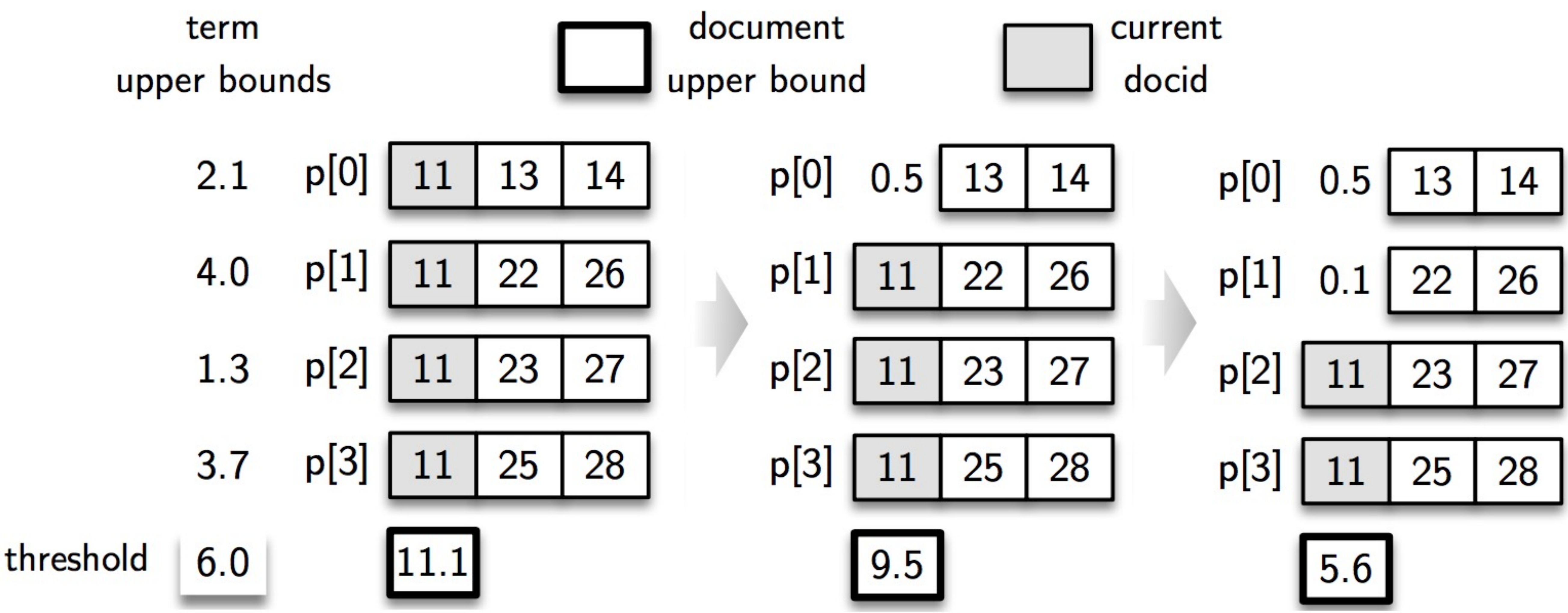

term
upper bounds
document
upper bound
current
docid
2.1
p[0]
11
13
14
4.0
p[1]
11
22
26
1.3
p[2]
11
23
27
3.7
p[3]
11
25
28
threshold
6.0
11.1

# Example



term upper bounds
document upper bound
current docid
2.1 p[0] 11 13 14
4.0 p[1] 11 22 26
1.3 p[2] 11 23 27
3.7 p[3] 11 25 28
threshold 6.0
11.1
p[0] 0.5 13 14
p[1] 11 22 26
p[2] 11 23 27
p[3] 11 25 28
9.5

# Example



term upper bounds
document upper bound
current docid
2.1 p[0] 11 13 14
4.0 p[1] 11 22 26
1.3 p[2] 11 23 27
3.7 p[3] 11 25 28
threshold 6.0
11.1
p[0] 0.5 13 14
p[1] 11 22 26
p[2] 11 23 27
p[3] 11 25 28
9.5
p[0] 0.5 13 14
p[1] 0.1 22 26
p[2] 11 23 27
p[3] 11 25 28
5.6

# Max Score Optimizations

- **MaxScore** strategy
  - **Early termination**: does not compute scores for documents that won't be returned
  - By comparing upper bounds with threshold
  - Based on **essential** and **non-essential** posting lists
  - Suitable for TAAT as well
- **WAND** strategy
  - **Approximate evaluation**: does not consider documents with approximate scores (sum of upper bounds) lower than threshold
  - Based on **pivot term** (posting list) and **pivot document**
- Both use **docid-sorted** posting lists
- Both exploit skipping


Turtle, H. and J. Flood. 1995. "Query Evaluation: Strategies and Optimizations". Inf. Process. Manage. 31(6): 831–850.

Broder, A., D. Carmel, M. Herscovici, A. Soffer, and J. Zien. "Efficient Query Evaluation using a Two-level Retrieval Process", CIKM 2003

# Performance

| | Small index | | | | Large index | | | |
|---|---|---|---|---|---|---|---|---|
| | Short queries | | Long queries | | Short queries | | Long queries | |
| DAAT | 0.19 | | 4.55 | | 3.58 | | 26.78 | |
| MaxScore | 0.17 | (1.12×) | 2.69 | (1.69×) | 1.58 | (2.27×) | 9.32 | (2.87×) |
| WAND | 0.21 | (0.90×) | 5.22 | (0.87×) | 1.90 | (1.88×) | 14.08 | (1.90×) |

Latency results (in ms) for DAAT, MaxScore and WAND (with speedups), for $K = 30$.

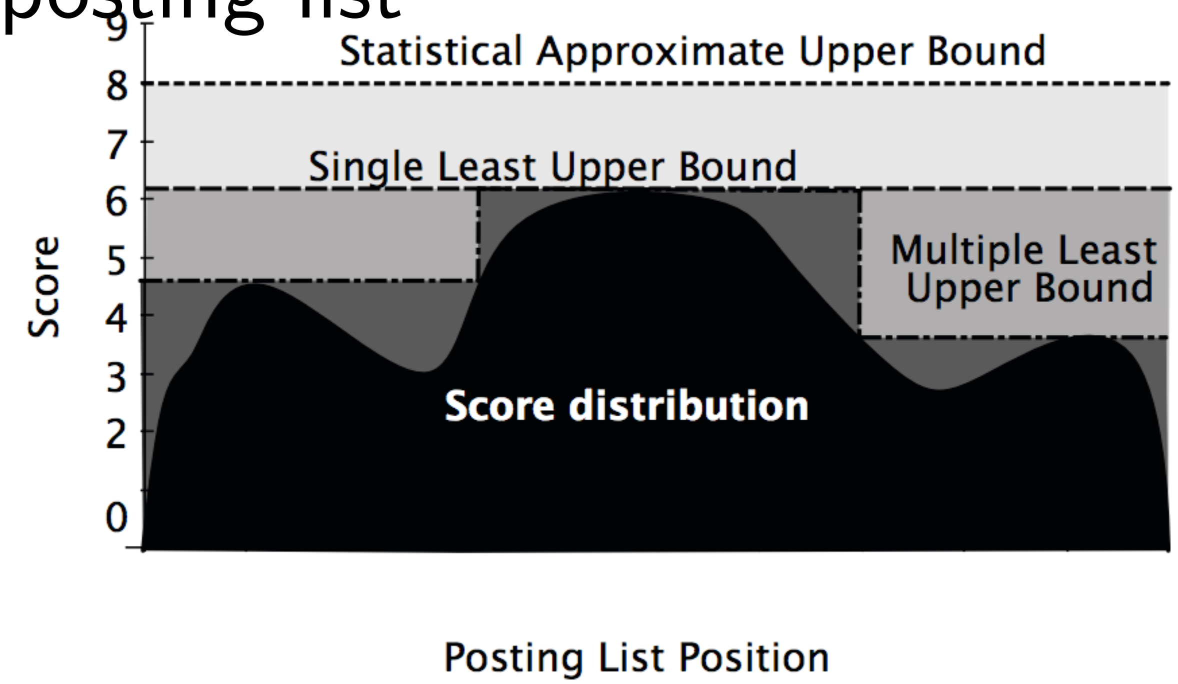
Adapted from [Fontoura et al., 2011].

| | Number of query terms | | | | | | | | | | Avg | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | 2 | | 3 | | 4 | | 5 | | 6+ | | | |
| DAAT | 60.6 | | 215.9 | | 439.1 | | 686.5 | | 1,270.5 | | 542.5 | |
| MaxScore | 12.7 | (4.77×) | 21.3 | (10.14×) | 27.1 | (16.20×) | 33.9 | (20.25×) | 55.0 | (23.10×) | 32.3 | (16.80×) |
| WAND | 14.2 | (4.27×) | 23.1 | (9.34×) | 27.3 | (16.08×) | 37.3 | (18.40×) | 73.8 | (17.22×) | 37.2 | (14.58×) |

Latency results (in ms) for DAAT, MaxScore and WAND (with speedups) for different query lengths, average query times on ClueWeb09, for $K = 10$. Adapted from [Mallia et al., 2017].

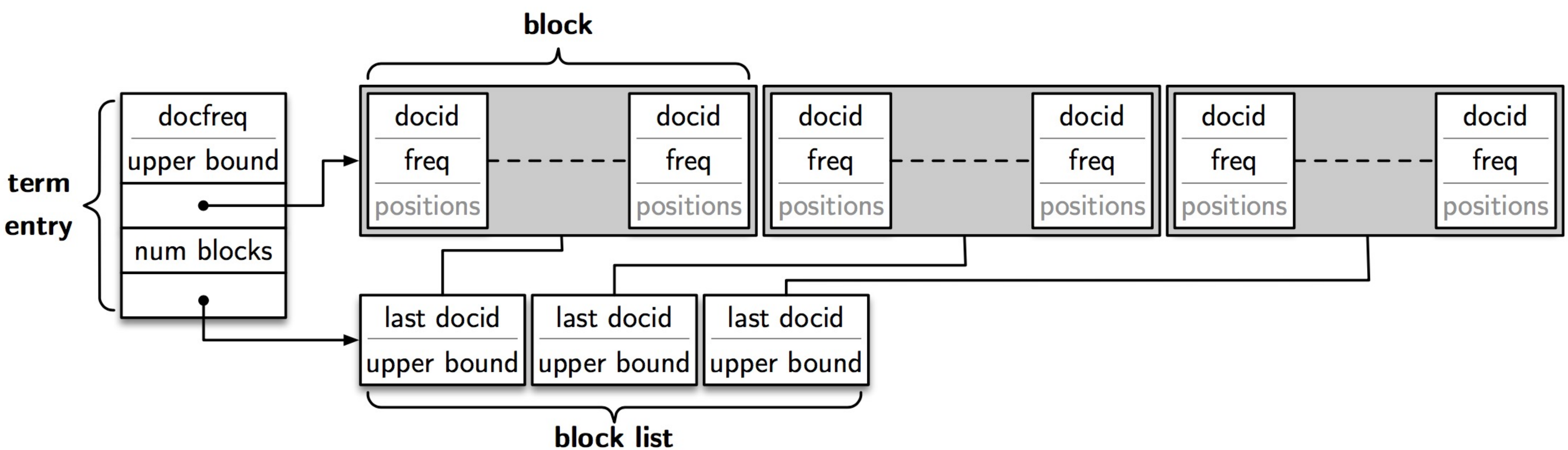
# Global vs Local Term Upper Bounds

- A (**global**) **term upper bound** is computed over the scores of all documents in a posting list




- Each posting list is sequentially divided in a block list, where each block contains a given number of consecutive postings, e.g., 128 postings per block.
- For each block, a block (**local**) **term upper bound** is computed, storing the maximum contribution of the postings in the block.

# Block Max Indexes




Ding, S. and T. Suel. "Faster Top-k Document Retrieval Using Block-max Indexes", SIGIR 2011

# Performance

| | Number of query terms | | | | | Avg |
|---|---|---|---|---|---|---|
| | 2 | 3 | 4 | 5 | 6+ | |
| DAAT | 60.0 | 159.2 | 261.4 | 376.0 | 646.4 | 225.7 |
| WAND | 23.0 (2.61×) | 42.5 (3.75×) | 89.9 (2.91×) | 141.2 (2.66×) | 251.6 (2.60×) | 77.6 (2.91×) |
| BMW | 4.1 (14.63×) | 11.5 (13.84×) | 33.6 (7.78×) | 54.5 (6.90×) | 114.2 (5.66×) | 27.9 (8.09×) |

Latency results (in ms) for DAAT, WAND and BMW (64 postings blocks) (with speedups) for different query lengths, average query times (Avg, in ms) on Gov2, for K = 10.

Ding, S. and T. Suel. "Faster Top-k Document Retrieval Using Block-max Indexes", SIGIR 2011

# Impact-sorted Indexes

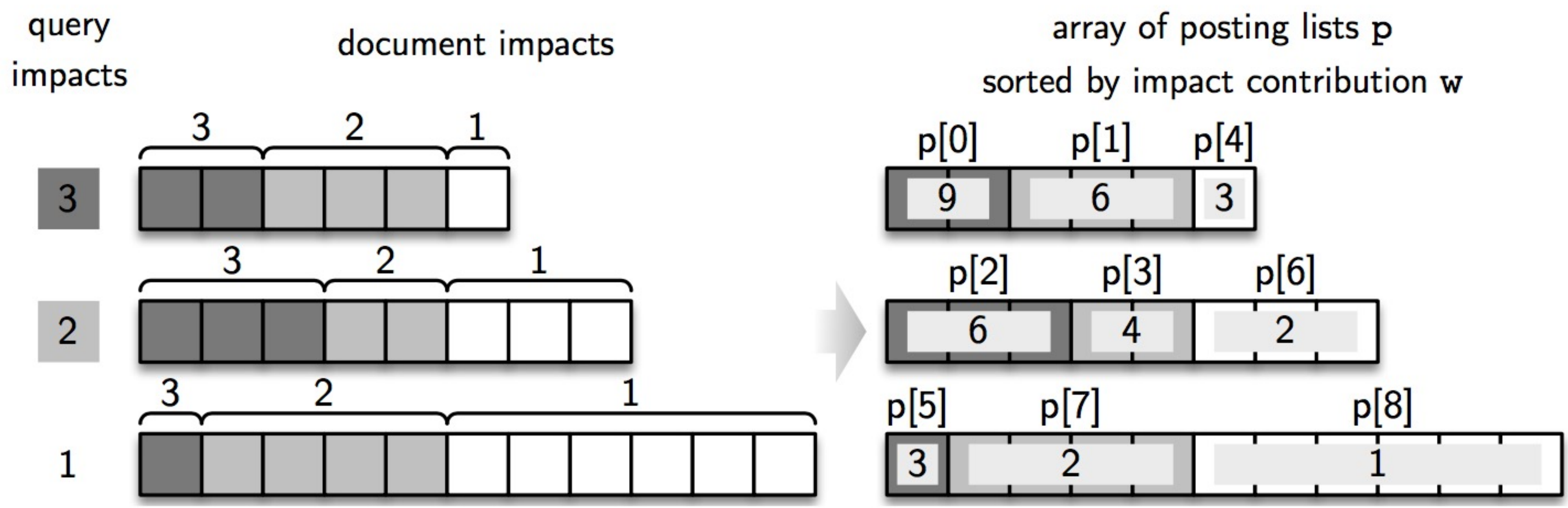
- An alternative way to arrange the posting lists is to sort them such that the **highest scoring documents appear early**.
- If an algorithm could find them at the beginning of posting lists, **dynamic pruning conditions can be enforced very early**.
- [Wong and Lee, 1993] proposed to sort posting list in **decreasing** order of **term-document frequency** value.
- [Anh et al., 2001] introduced the definition of **impact** of term $t$ in document $d$ for the quantity $w_t(d)/W_d$ (or **document impact**) and impact of term $t$ in query $q$ for $w_t(q)$ (or **query impact**):

$$s_q(d) = \sum_{t \in q} s_t(q, d) = \frac{1}{W_d} \sum_{t \in q} w_t(q) w_t(d)$$

- They proposed to facilitate effective query pruning by sorting the posting lists in **decreasing** order of **(document) impact**, i.e., to process queries on an impact-sorted index.

Wong, W. Y. P. and D. L. Lee. 1993. "Implementations of Partial Document Ranking Using Inverted Files". Inf. Process. Manage. 29(5): 647–669.

# Impact-based query processing




Anh, V. N., O. de Kretser, and A. Moffat. 2001. "Vector-space Ranking with Effective Early Termination", SIGIR 2001

# Anytime Ranking

- [Lin and Trotman, 2015] proposed a linear regression model to **translate a deadline time** on the query processing time **into the number of posting to process**.
- [Mackenzie et al., 2017] proposed to stop after processing a given percentage of the total postings in the query terms' posting lists, **on a per-query basis.**
- According to their experiments, the fixed threshold **may result in reduced effectiveness**, as the number of query terms increases, but conversely it gives a **very strict control over the tail latency** of the queries.

Lin, J. and A. Trotman. "Anytime Ranking for Impact-Ordered Indexes", ICTIR 2015
Mackenzie, J., F. Scholer, and J. S. Culpepper. "Early Termination Heuristics for Score-at-a-Time Index Traversal", ADCS 2017

# Why learning?

- How to **choose** term weighting models $s_t(q, d)$?
  - **Different** term weighting **models have different assumptions** about how relevant documents should be retrieved
- Also:
  - **Field**-based models: term occurrences in different fields matter differently
  - **Proximity** models: close co-occurrences matter more
  - **Neural** models: semantic similarity matters
  - **Priors**: documents with particular lengths or URL/inlink values matter more
  - **Query features**: Long queries, difficult queries, query type
- How to combine all these easily and appropriately?

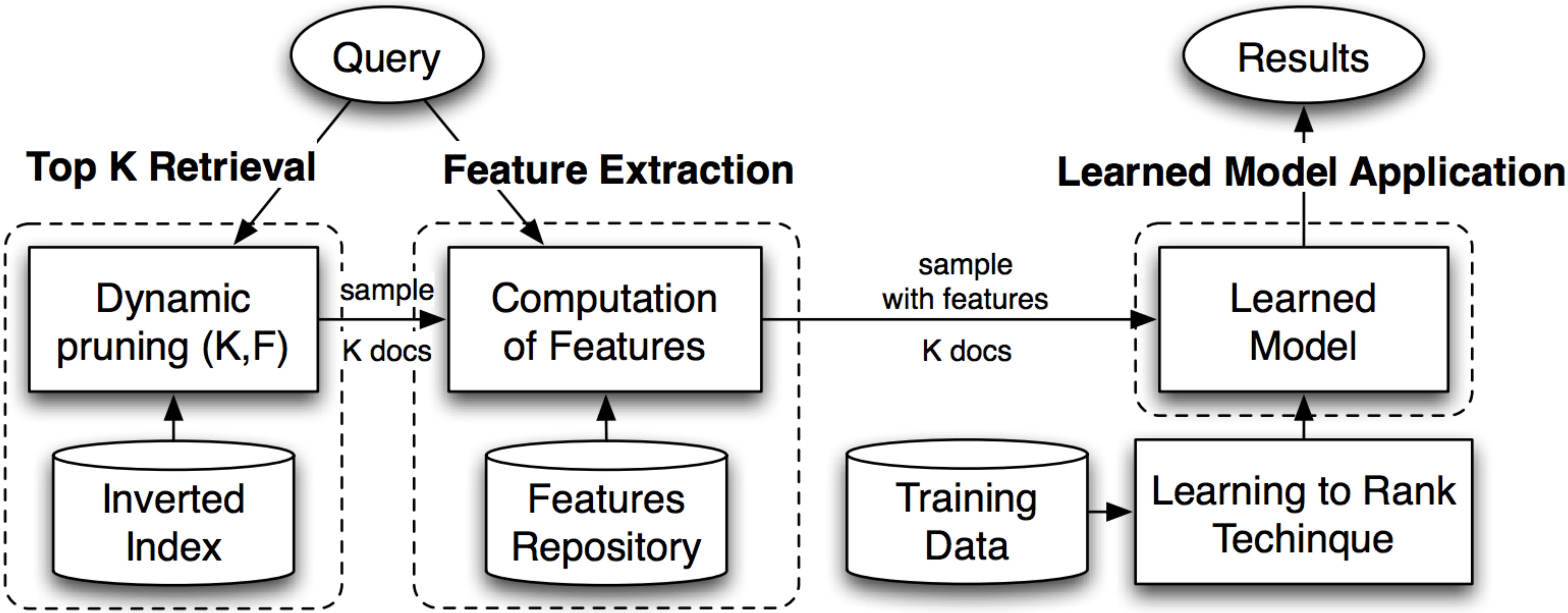
Macdonald, C., R. L. Santos, and I. Ounis. 2013a. "The Whens and Hows of Learning to Rank for Web Search". Inf. Retr. 16(5): 584–628.

# Learning to Rank

- Application of tailored machine learning techniques to automatically (select and) weight retrieval **features**
  - Based on training data with relevance assessments

- **Learning to rank** has been popularised by commercial Web search engines (e.g. Bing, Baidu, Yandex)
  - They require **large training datasets**
  - **Click-through data** has facilitated the deployment of learning approaches

# Schematically



# Learning To Rank

training set of queries with ideal document rankings (including irrelevant documents)

$q_a \rightarrow [(d_4, y_{a4}), (d_2, y_{a2}), (d_3, y_{a3}), (d_8, y_{a8}), (d_{41}, y_{a41}), \ldots]$

$q_b \rightarrow [(d_{99}, y_{b99}), (d_4, y_{b4}), (d_7, y_{b7}), (d_2, y_{b2}), (d_{11}, y_{b11}), \ldots]$

...

**Document $d_x$**
set of real-valued features
$[f_{x0}, f_{x1}, f_{x2}, f_{x3}, f_{x4}, f_{x5}, f_{x6}, \ldots]$

**Label $y_{xi}$**
five values,
from 0 (irrelevant)
to 4 (perfectly relevant)

# Learning To Rank

training set of queries with ideal document rankings (including irrelevant documents)

$q_a \rightarrow [(d_4, y_{a4}), (d_2, y_{a2}), (d_3, y_{a3}), (d_8, y_{a8}), (d_{41}, y_{a41}), \ldots]$

$q_b \rightarrow [(d_{99}, y_{b99}), (d_4, y_{b4}), (d_7, y_{b7}), (d_2, y_{b2}), (d_{11}, y_{b11}), \ldots]$

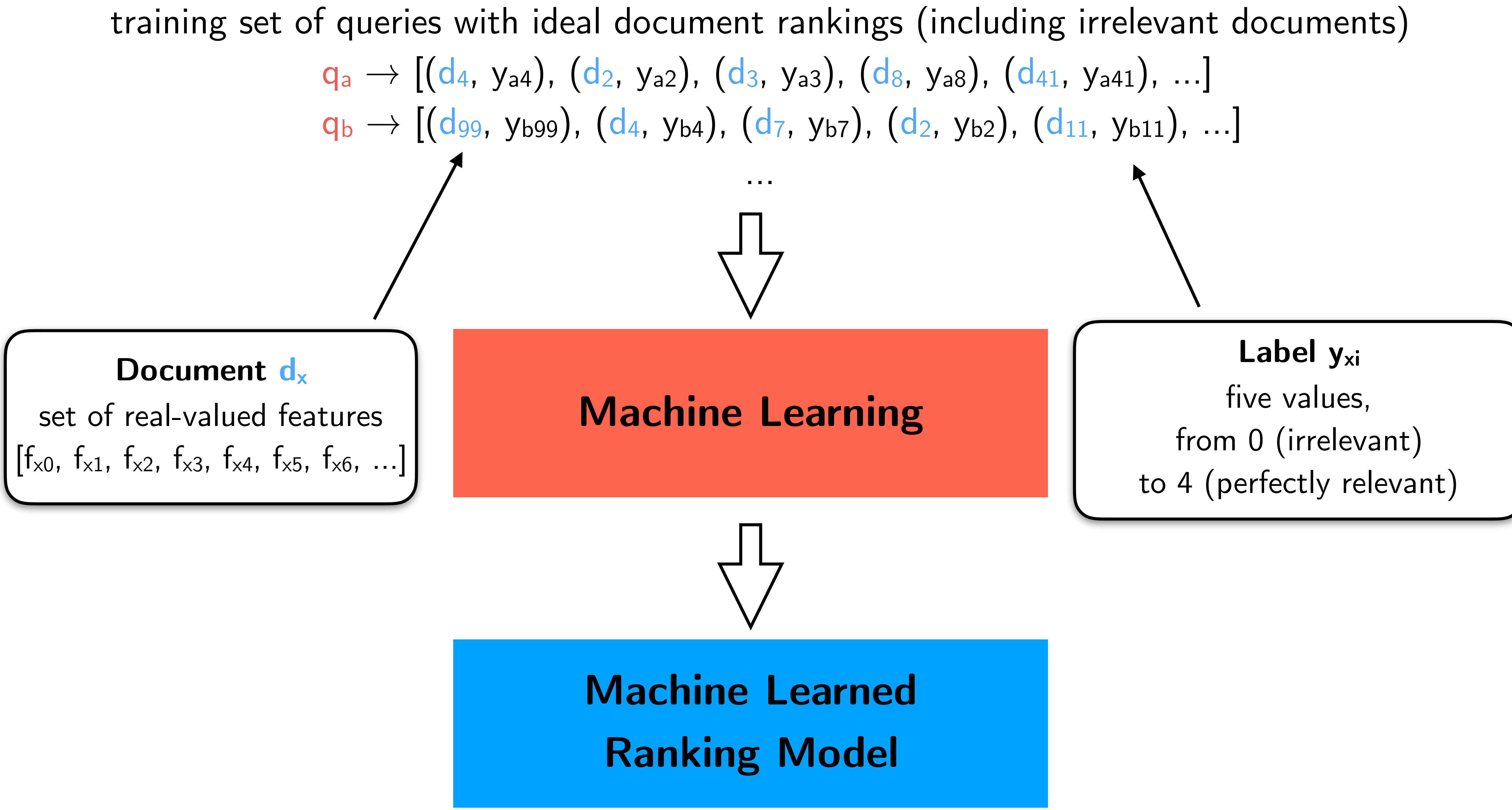
…

**Document $d_x$**
set of real-valued features
$[f_{x0}, f_{x1}, f_{x2}, f_{x3}, f_{x4}, f_{x5}, f_{x6}, \ldots]$

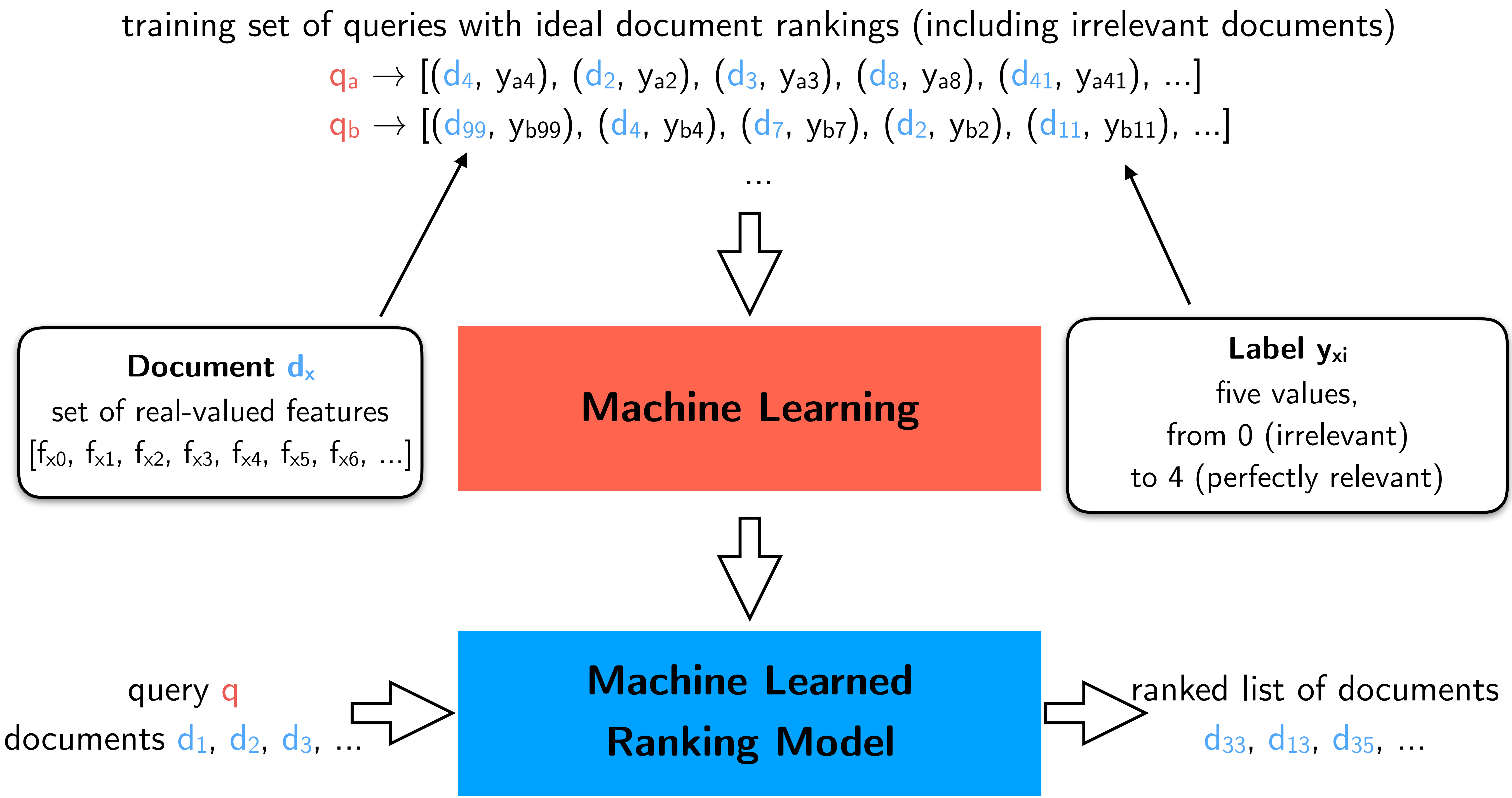
**Machine Learning**

**Label $y_{xi}$**
five values,
from 0 (irrelevant)
to 4 (perfectly relevant)

# Learning To Rank

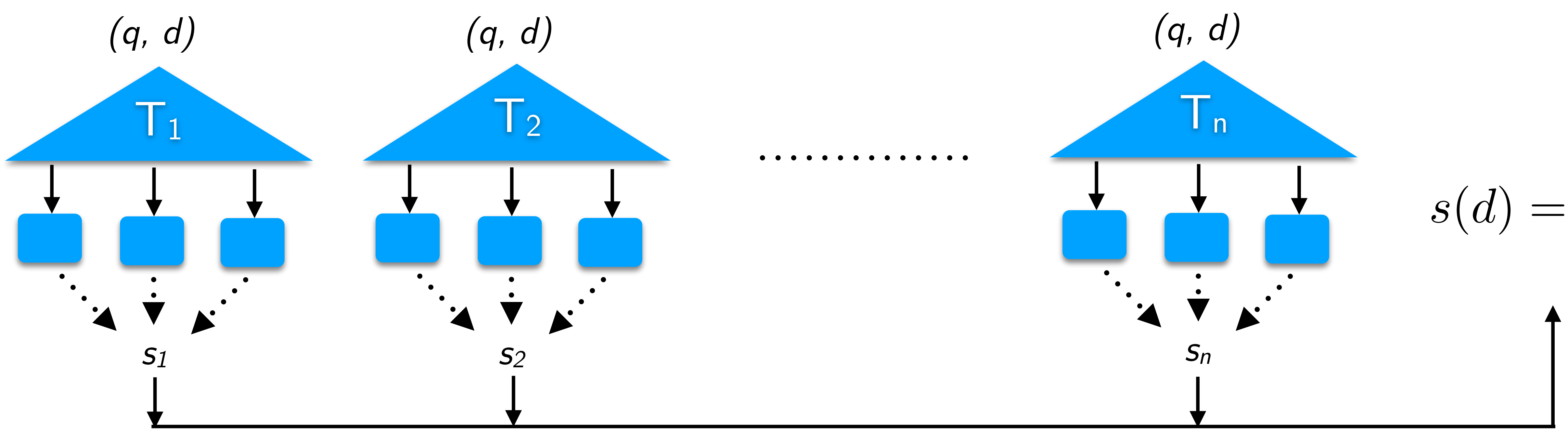


# Learning To Rank



# Learning To Rank with Large Tree Ensembles

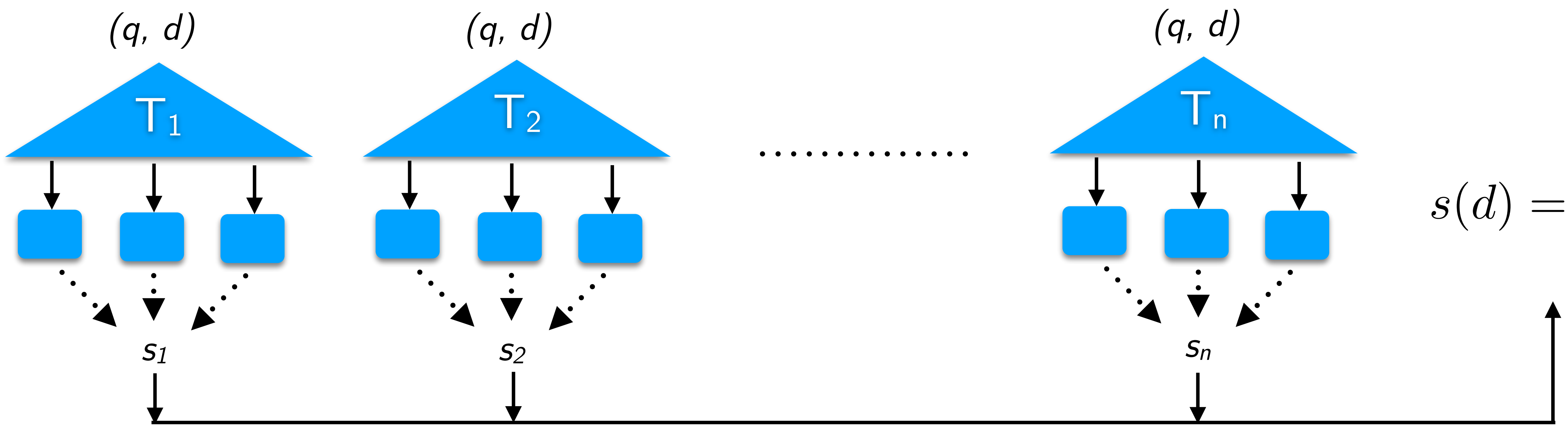
- Tree forests: GBRT, Lambda MART, Random Forest, Oblivious Trees, etc.
  - ensemble of weak learners, each contributing a partial score
  - at scoring time, all trees can be processed independently




$$s(d) = \sum_{i=1}^{n} s_i$$

# Learning To Rank with Large Tree Ensembles

- Tree forests: GBRT, Lambda MART, Random Forest, Oblivious Trees, etc.
  - ensemble of weak learners, each contributing a partial score
  - at scoring time, all trees can be processed independently



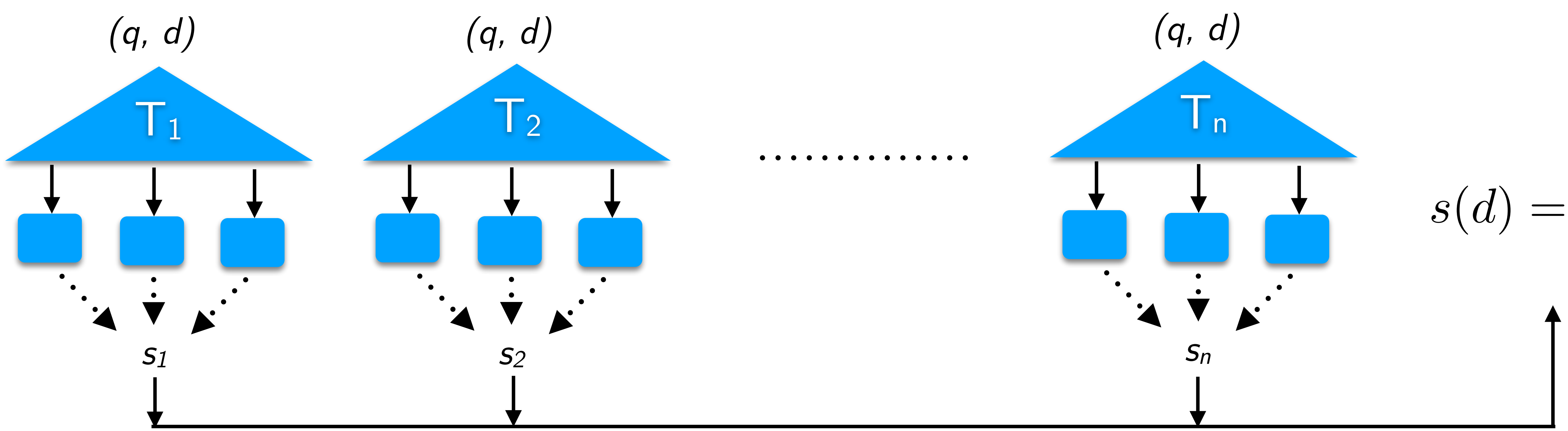

$$s(d) = \sum_{i=1}^{n} s_i$$

## Assuming:

- 3,000 trees
- an average tree depth of 10 nodes
- 1,000 documents scored per query
- a cluster with 1,000 search shards

# Learning To Rank with Large Tree Ensembles

- Tree forests: GBRT, Lambda MART, Random Forest, Oblivious Trees, etc.
  - ensemble of weak learners, each contributing a partial score
  - at scoring time, all trees can be processed independently




$$s(d) = \sum_{i=1}^{n} s_i$$

## Assuming:

- 3,000 trees
- an average tree depth of 10 nodes
- 1,000 documents scored per query
- a cluster with 1,000 search shards

## Expensive!

- 3,000 x 10 = 30 K tests per document
- 30 K x 1,000 = 30 M tests per query
- approx. 30 G tests for the entire search cluster

# Quickscorer Recipe

- Avoid processing one tree at a time
  - traverse one feature at a time
- Encode trees as bitvectors
  - reachable vs. unreachable leaves
  - traverse trees using logical bit-wise operations
  - bit-wise operations are commutative
- Adapt processing to the CPU characteristics.
  - avoid random memory lookups → improve cache usage
  - avoid conditional statements → reduce branch mispredictions

Dato, D., et al. 2016. "Fast Ranking with Additive Ensembles of Oblivious and Non-Oblivious Regression Trees". ACM TOIS 35(2): 15:1–15:31.

# Baselines

- Code Generators
  - If-Then: `if (x[0] ≤ 13.4) then` *go left* `else` *go right*
  - Cond-Op: `(x[0] ≤ 13.4) ?` *go left* `:` *go right*
    - High cache hit rate / High misprediction rate
- Code Optimizations
  - Struct+: pointer-based data structure
    - Low cache hit rate / High misprediction rate
  - VPred
    - Medium cache hit rate / Low misprediction rate
  - Oblivious
    - High cache hit rate / Low misprediction rate

# Performance

| Method | Λ | Number of trees/dataset | | | | | |
|---|---|---|---|---|---|---|---|
| | | 1,000 | | | 5,000 | | |
| | | MSN-1 | Y!S1 | Istella | MSN-1 | Y!S1 | Istella |
| QS | 8 | **2.2** (–) | **4.4** (–) | **2.8** (–) | **10.1** (–) | **14.5** (–) | **12.0** (–) |
| VPRED | | 7.8 (3.5x) | 8.4 (1.9x) | 7.6 (2.7x) | 39.8 (3.9x) | 41.1 (2.8x) | 39.3 (3.3x) |
| CONDOP | | 6.7 (3.0x) | 10.3 (2.3x) | 8.0 (2.9x) | 67.8 (6.7x) | 75.9 (5.2x) | 77.0 (6.4x) |
| IF-THEN | | 7.3 (3.3x) | 10.1 (2.3x) | 9.0 (3.2x) | 78.2 (7.7x) | 84.6 (5.8x) | 84.1 (7.0x) |
| STRUCT+ | | 21.0 (9.5x) | 23.1 (5.3x) | 24.9 (8.9x) | 98.7 (9.8x) | 119.5 (8.2x) | 117.9 (9.8x) |
| QS | 16 | **3.1** (–) | **6.4** (–) | **4.5** (–) | **15.9** (–) | **21.6** (–) | **17.9** (–) |
| VPRED | | 16.0 (5.2x) | 16.4 (2.6x) | 14.9 (3.3x) | 82.0 (5.2x) | 82.4 (3.8x) | 79.3 (4.4x) |
| CONDOP | | 14.1 (4.5x) | 17.2 (2.7x) | 16.1 (3.6x) | 100.0 (6.3x) | 110.0 (5.0x) | 155.0 (8.7x) |
| IF-THEN | | 18.1 (5.8x) | 20.7 (3.2x) | 19.6 (4.4x) | 128.0 (8.0x) | 128.8 (6.0x) | 135.5 (7.6x) |
| STRUCT+ | | 40.9 (13.2x) | 41.6 (6.5x) | 44.4 (9.9x) | 411.2 (25.9x) | 418.6 (19.4x) | 407.8 (22.8x) |
| QS | 32 | **5.2** (–) | **9.7** (–) | **6.8** (–) | **26.8** (–) | **34.5** (–) | **26.9** (–) |
| VPRED | | 31.8 (6.1x) | 31.5 (3.2x) | 28.1 (4.1x) | 164.5 (6.1x) | 16 1.6 (4.7x) | 157.7 (5.9x) |
| CONDOP | | 27.0 (5.2x) | 30.3 (3.1x) | 30.4 (4.5x) | NA (x) | NA (x) | NA (x) |
| IF-THEN | | 32.2 (6.2x) | 34.0 (3.5x) | 33.3 (4.9x) | 270.5 (10.1x) | 256.6 (7.4x) | 240.6 (8.9x) |
| STRUCT+ | | 69.4 (13.3x) | 66.5 (6.9x) | 67.8 (10.0x) | 861.0 (32.1x) | 833.2 (24.2x) | 807.9 (x) |
| QS | 64 | **9.4** (–) | **15.1** (–) | **11.2** (–) | **57.6** (–) | **70.2** (–) | **57.8** (–) |
| VPRED | | 62.2 (6.6x) | 57.3 (3.8x) | 54.3 (4.8x) | 347.2 (6.0x) | 333.6 (4.8x) | 326.8 (5.7x) |
| CONDOP | | 48.6 (5.2x) | 48.4 (3.2x) | 51.2 (4.6x) | NA (x) | NA (x) | NA (x) |
| IF-THEN | | 54.0 (5.8x) | 53.2 (3.5x) | 55.0 (4.9x) | 901.1 (15.6x) | 801.9 (11.4x) | 911.2 (15.8x) |
| STRUCT+ | | 132.3 (14.1x) | 109.5 (7.3x) | 112.6 (10.5x) | 1485.5 (25.8x) | 1498.2 (21.3x) | 1487.3 (25.7x) |

| Method | Λ | Number of trees/dataset | | | | | |
|---|---|---|---|---|---|---|---|
| | | 10,000 | | | 20,000 | | |
| | | MSN-1 | Y!S1 | Istella | MSN-1 | Y!S1 | Istella |
| QS | 64 | **158.2** (–) | **156.3** (–) | **146.7** (–) | **428.1** (–) | **335.0** (–) | **289.6** (–) |
| VPRED | | 733.2 (4.6x) | 704.7 (4.5x) | 696.3 (4.7x) | 1307.6 (3.0x) | 1412.9 (4.2x) | 1413.1 (4.9x) |
| CONDOP | | NA (x) | NA (x) | NA (x) | NA (x) | NA (x) | NA (x) |
| IF-THEN | | 2364.3 (14.9x) | 2350.5 (15.0x) | 2334.8 (15.9x) | 4397.1 (10.3x) | 4647.2 (13.9x) | 4678.8 (16.2x) |
| STRUCT+ | | 3014.8 (19.0x) | 2894.4 (18.5x) | 2942.8 (20.1x) | 6794.5 (15.9x) | 6923.9 (20.7x) | 7586.4 (25.7x) |

# Performance

| Method | Λ | Number of trees/dataset | | | | | |
|---|---|---|---|---|---|---|---|
| | | 1, 000 | | | 5, 000 | | |
| | | MSN-1 | Y!S1 | Istella | MSN-1 | Y!S1 | Istella |
| QS | 8 | **2.2** (–) | **4.4** (–) | **2.8** (–) | **10.1** (–) | **14.5** (–) | **12.0** (–) |
| VPred | | 7.8 (3.5x) | 8.4 (1.9x) | 7.6 (2.7x) | 39.8 (3.9x) | 41.1 (2.8x) | 39.3 (3.3x) |
| CondOp | | 6.7 (3.0x) | 10.3 (2.3x) | 8.0 (2.9x) | 67.8 (6.7x) | 75.9 (5.2x) | 77.0 (6.4x) |
| If-Then | | 7.3 (3.3x) | 10.1 (2.3x) | 9.0 (3.2x) | 78.2 (7.7x) | 84.6 (5.8x) | 84.1 (7.0x) |
| Struct+ | | 21.0 (9.5x) | 23.1 (5.3x) | 24.9 (8.9x) | 98.7 (9.8x) | 119.5 (8.2x) | 117.9 (9.8x) |
| QS | 16 | **3.1** (–) | **6.4** (–) | **4.5** (–) | **15.9** (–) | **21.6** (–) | **17.9** (–) |
| VPred | | 16.0 (5.2x) | 16.4 (2.6x) | 14.9 (3.3x) | 82.0 (5.2x) | 82.4 (3.8x) | 79.3 (4.4x) |
| CondOp | | 14.1 (4.5x) | 17.2 (2.7x) | 16.1 (3.6x) | 100.0 (6.3x) | 110.0 (5.0x) | 155.0 (8.7x) |
| If-Then | | 18.1 (5.8x) | 20.7 (3.2x) | 19.6 (4.4x) | 128.0 (8.0x) | 128.8 (6.0x) | 135.5 (7.6x) |
| Struct+ | | 40.9 (13.2x) | 41.6 (6.5x) | 44.4 (9.9x) | 411.2 (25.9x) | 418.6 (19.4x) | 407.8 (22.8x) |
| QS | 32 | **5.2** (–) | **9.7** (–) | **6.8** (–) | **26.8** (–) | **34.5** (–) | **26.9** (–) |
| VPred | | 31.8 (6.1x) | 31.5 (3.2x) | 28.1 (4.1x) | 164.5 (6.1x) | 16 1.6 (4.7x) | 157.7 (5.9x) |
| CondOp | | 27.0 (5.2x) | 30.3 (3.1x) | 30.4 (4.5x) | NA (x) | NA (x) | NA (x) |
| If-Then | | 32.2 (6.2x) | 34.0 (3.5x) | 33.3 (4.9x) | 270.5 (10.1x) | 256.6 (7.4x) | 240.6 (8.9x) |
| Struct+ | | 69.4 (13.3x) | 66.5 (6.9x) | 67.8 (10.0x) | 861.0 (32.1x) | 833.2 (24.2x) | 807.9 (x) |
| QS | 64 | **9.4** (–) | **15.1** (–) | **11.2** (–) | **57.6** (–) | **70.2** (–) | **57.8** (–) |
| VPred | | 62.2 (6.6x) | 57.3 (3.8x) | 54.3 (4.8x) | 347.2 (6.0x) | 333.6 (4.8x) | 326.8 (5.7x) |
| CondOp | | 48.6 (5.2x) | 48.4 (3.2x) | 51.2 (4.6x) | NA (x) | NA (x) | NA (x) |
| If-Then | | 54.0 (5.8x) | 53.2 (3.5x) | 55.0 (4.9x) | 901.1 (15.6x) | 801.9 (11.4x) | 911.2 (15.8x) |
| Struct+ | | 132.3 (14.1x) | 109.5 (7.3x) | 112.6 (10.5x) | 1485.5 (25.8x) | 1498.2 (21.3x) | 1487.3 (25.7x) |

| Method | Λ | Number of trees/dataset | | | | | |
|---|---|---|---|---|---|---|---|
| | | 10, 000 | | | 20, 000 | | |
| | | MSN-1 | Y!S1 | Istella | MSN-1 | Y!S1 | Istella |
| QS | 64 | **158.2** (–) | **156.3** (–) | **146.7** (–) | **428.1** (–) | **335.0** (–) | **289.6** (–) |
| VPred | | 733.2 (4.6x) | 704.7 (4.5x) | 696.3 (4.7x) | 1307.6 (3.0x) | 1412.9 (4.2x) | 1413.1 (4.9x) |
| CondOp | | NA (x) | NA (x) | NA (x) | NA (x) | NA (x) | NA (x) |
| If-Then | | 2364.3 (14.9x) | 2350.5 (15.0x) | 2334.8 (15.9x) | 4397.1 (10.3x) | 4647.2 (13.9x) | 4678.8 (16.2x) |
| Struct+ | | 3014.8 (19.0x) | 2894.4 (18.5x) | 2942.8 (20.1x) | 6794.5 (15.9x) | 6923.9 (20.7x) | 7586.4 (26.2x) |

# Performance

| Method | $\Lambda$ | Number of trees/dataset | | | | | |
|---|---|---|---|---|---|---|---|
| | | 1,000 | | | 5,000 | | |
| | | MSN-1 | Y!S1 | Istella | MSN-1 | Y!S1 | Istella |
| QS | 8 | **2.2** (–) | **4.4** (–) | **2.8** (–) | **10.1** (–) | **14.5** (–) | **12.0** (–) |
| VPred | | 7.8 (3.5x) | 8.4 (1.9x) | 7.6 (2.7x) | 39.8 (3.9x) | 41.1 (2.8x) | 39.3 (3.3x) |
| CondOp | | 6.7 (3.0x) | 10.3 (2.3x) | 8.0 (2.9x) | 67.8 (6.7x) | 75.9 (5.2x) | 77.0 (6.4x) |
| If-Then | | 7.3 (3.3x) | 10.1 (2.3x) | 9.0 (3.2x) | 78.2 (7.7x) | 84.6 (5.8x) | 84.1 (7.0x) |
| Struct+ | | 21.0 (9.5x) | 23.1 (5.3x) | 24.9 (8.9x) | 98.7 (9.8x) | 119.5 (8.2x) | 117.9 (9.8x) |
| QS | 16 | **3.1** (–) | **6.4** (–) | **4.5** (–) | **15.9** (–) | **21.6** (–) | **17.9** (–) |
| VPred | | 16.0 (5.2x) | 16.4 (2.6x) | 14.9 (3.3x) | 82.0 (5.2x) | 82.4 (3.8x) | 79.3 (4.4x) |
| CondOp | | 14.1 (4.5x) | 17.2 (2.7x) | 16.1 (3.6x) | 100.0 (6.3x) | 110.0 (5.0x) | 155.0 (8.7x) |
| If-Then | | 18.1 (5.8x) | 20.7 (3.2x) | 19.6 (4.4x) | 128.0 (8.0x) | 128.8 (6.0x) | 135.5 (7.6x) |
| Struct+ | | 40.9 (13.2x) | 41.6 (6.5x) | 44.4 (9.9x) | 411.2 (25.9x) | 418.6 (19.4x) | 407.8 (22.8x) |
| QS | 32 | **5.2** (–) | **9.7** (–) | **6.8** (–) | **26.8** (–) | **34.5** (–) | **26.9** (–) |
| VPred | | 31.8 (6.1x) | 31.5 (3.2x) | 28.1 (4.1x) | 164.5 (6.1x) | 16 1.6 (4.7x) | 157.7 (5.9x) |
| CondOp | | 27.0 (5.2x) | 30.3 (3.1x) | 30.4 (4.5x) | NA (x) | NA (x) | NA (x) |
| If-Then | | 32.2 (6.2x) | 34.0 (3.5x) | 33.3 (4.9x) | 270.5 (10.1x) | 256.6 (7.4x) | 240.6 (8.9x) |
| Struct+ | | 69.4 (13.3x) | 66.5 (6.9x) | 67.8 (10.0x) | 861.0 (32.1x) | 833.2 (24.2x) | 807.9 (x) |
| QS | 64 | **9.4** (–) | **15.1** (–) | **11.2** (–) | **57.6** (–) | **70.2** (–) | **57.8** (–) |
| VPred | | 62.2 (6.6x) | 57.3 (3.8x) | 54.3 (4.8x) | 347.2 (6.0x) | 333.6 (4.8x) | 326.8 (5.7x) |
| CondOp | | 48.6 (5.2x) | 48.4 (3.2x) | 51.2 (4.6x) | NA (x) | NA (x) | NA (x) |
| If-Then | | 54.0 (5.8x) | 53.2 (3.5x) | 55.0 (4.9x) | 901.1 (15.6x) | 801.9 (11.4x) | 911.2 (15.8x) |
| Struct+ | | 132.3 (14.1x) | 109.5 (7.3x) | 112.6 (10.5x) | 1485.5 (25.8x) | 1498.2 (21.3x) | 1487.3 (25.7x) |

| Method | $\Lambda$ | Number of trees/dataset | | | | | |
|---|---|---|---|---|---|---|---|
| | | 10,000 | | | 20,000 | | |
| | | MSN-1 | Y!S1 | Istella | MSN-1 | Y!S1 | Istella |
| QS | 64 | **158.2** (–) | **156.3** (–) | **146.7** (–) | **428.1** (–) | **335.0** (–) | **289.6** (–) |
| VPred | | 733.2 (4.6x) | 704.7 (4.5x) | 696.3 (4.7x) | 1307.6 (3.0x) | 1412.9 (4.2x) | 1413.1 (4.9x) |
| CondOp | | NA (x) | NA (x) | NA (x) | NA (x) | NA (x) | NA (x) |
| If-Then | | 2364.3 (14.9x) | 2350.5 (15.0x) | 2334.8 (15.9x) | 4397.1 (10.3x) | 4647.2 (13.9x) | 4678.8 (16.2x) |
| Struct+ | | 3014.8 (19.0x) | 2894.4 (18.5x) | 2942.8 (20.1x) | 6794.5 (15.9x) | 6923.9 (20.7x) | 7586.4 (26.2x) |

# Performance

| Method | Λ | Number of trees/dataset | | | | | |
|---|---|---|---|---|---|---|---|
| | | 1,000 | | | 5,000 | | |
| | | MSN-1 | Y!S1 | Istella | MSN-1 | Y!S1 | Istella |
| QS | 8 | **2.2** (–) | **4.4** (–) | **2.8** (–) | **10.1** (–) | **14.5** (–) | **12.0** (–) |
| VPRED | | 7.8 (3.5x) | 8.4 (1.9x) | 7.6 (2.7x) | 39.8 (3.9x) | 41.1 (2.8x) | 39.3 (3.3x) |
| CONDOP | | 6.7 (3.0x) | 10.3 (2.3x) | 8.0 (2.9x) | 67.8 (6.7x) | 75.9 (5.2x) | 77.0 (6.4x) |
| IF-THEN | | 7.3 (3.3x) | 10.1 (2.3x) | 9.0 (3.2x) | 78.2 (7.7x) | 84.6 (5.8x) | 84.1 (7.0x) |
| STRUCT+ | | 21.0 (9.5x) | 23.1 (5.3x) | 24.9 (8.9x) | 98.7 (9.8x) | 119.5 (8.2x) | 117.9 (9.8x) |
| QS | 16 | **3.1** (–) | **6.4** (–) | **4.5** (–) | **15.9** (–) | **21.6** (–) | **17.9** (–) |
| VPRED | | 16.0 (5.2x) | 16.4 (2.6x) | 14.9 (3.3x) | 82.0 (5.2x) | 82.4 (3.8x) | 79.3 (4.4x) |
| CONDOP | | 14.1 (4.5x) | 17.2 (2.7x) | 16.1 (3.6x) | 100.0 (6.3x) | 110.0 (5.0x) | 155.0 (8.7x) |
| IF-THEN | | 18.1 (5.8x) | 20.7 (3.2x) | 19.6 (4.4x) | 128.0 (8.0x) | 128.8 (6.0x) | 135.5 (7.6x) |
| STRUCT+ | | 40.9 (13.2x) | 41.6 (6.5x) | 44.4 (9.9x) | 411.2 (25.9x) | 418.6 (19.4x) | 407.8 (22.8x) |
| QS | 32 | **5.2** (–) | **9.7** (–) | **6.8** (–) | **26.8** (–) | **34.5** (–) | **26.9** (–) |
| VPRED | | 31.8 (6.1x) | 31.5 (3.2x) | 28.1 (4.1x) | 164.5 (6.1x) | 16 1.6 (4.7x) | 157.7 (5.9x) |
| CONDOP | | 27.0 (5.2x) | 30.3 (3.1x) | 30.4 (4.5x) | NA (x) | NA (x) | NA (x) |
| IF-THEN | | 32.2 (6.2x) | 34.0 (3.5x) | 33.3 (4.9x) | 270.5 (10.1x) | 256.6 (7.4x) | 240.6 (8.9x) |
| STRUCT+ | | 69.4 (13.3x) | 66.5 (6.9x) | 67.8 (10.0x) | 861.0 (32.1x) | 833.2 (24.2x) | 807.9 (x) |
| QS | 64 | **9.4** (–) | **15.1** (–) | **11.2** (–) | **57.6** (–) | **70.2** (–) | **57.8** (–) |
| VPRED | | 62.2 (6.6x) | 57.3 (3.8x) | 54.3 (4.8x) | 347.2 (6.0x) | 333.6 (4.8x) | 326.8 (5.7x) |
| CONDOP | | 48.6 (5.2x) | 48.4 (3.2x) | 51.2 (4.6x) | NA (x) | NA (x) | NA (x) |
| IF-THEN | | 54.0 (5.8x) | 53.2 (3.5x) | 55.0 (4.9x) | 901.1 (15.6x) | 801.9 (11.4x) | 911.2 (15.8x) |
| STRUCT+ | | 132.3 (14.1x) | 109.5 (7.3x) | 112.6 (10.5x) | 1485.5 (25.8x) | 1498.2 (21.3x) | 1487.3 (25.7x) |

| Method | Λ | Number of trees/dataset | | | | | |
|---|---|---|---|---|---|---|---|
| | | 10,000 | | | 20,000 | | |
| | | MSN-1 | Y!S1 | Istella | MSN-1 | Y!S1 | Istella |
| QS | 64 | **158.2** (–) | **156.3** (–) | **146.7** (–) | **428.1** (–) | **335.0** (–) | **289.6** (–) |
| VPRED | | 733.2 (4.6x) | 704.7 (4.5x) | 696.3 (4.7x) | 1307.6 (3.0x) | 1412.9 (4.2x) | 1413.1 (4.9x) |
| CONDOP | | NA (x) | NA (x) | NA (x) | NA (x) | NA (x) | NA (x) |
| IF-THEN | | 2364.3 (14.9x) | 2350.5 (15.0x) | 2334.8 (15.9x) | 4397.1 (10.3x) | 4647.2 (13.9x) | 4678.8 (16.2x) |
| STRUCT+ | | 3014.8 (19.0x) | 2894.4 (18.5x) | 2942.8 (20.1x) | 6794.5 (15.9x) | 6923.9 (20.7x) | 7586.4 (26.2x) |

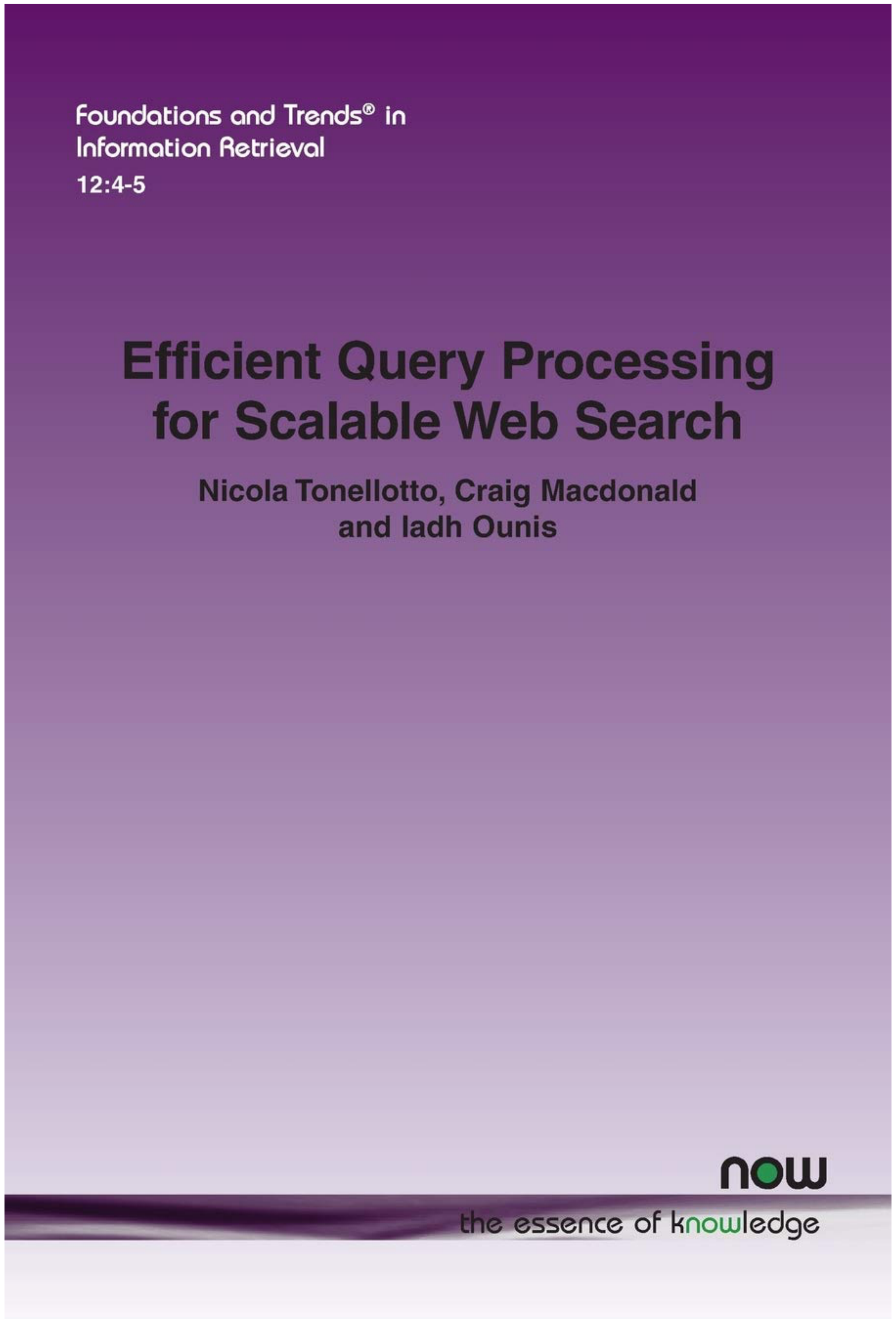
# Performance

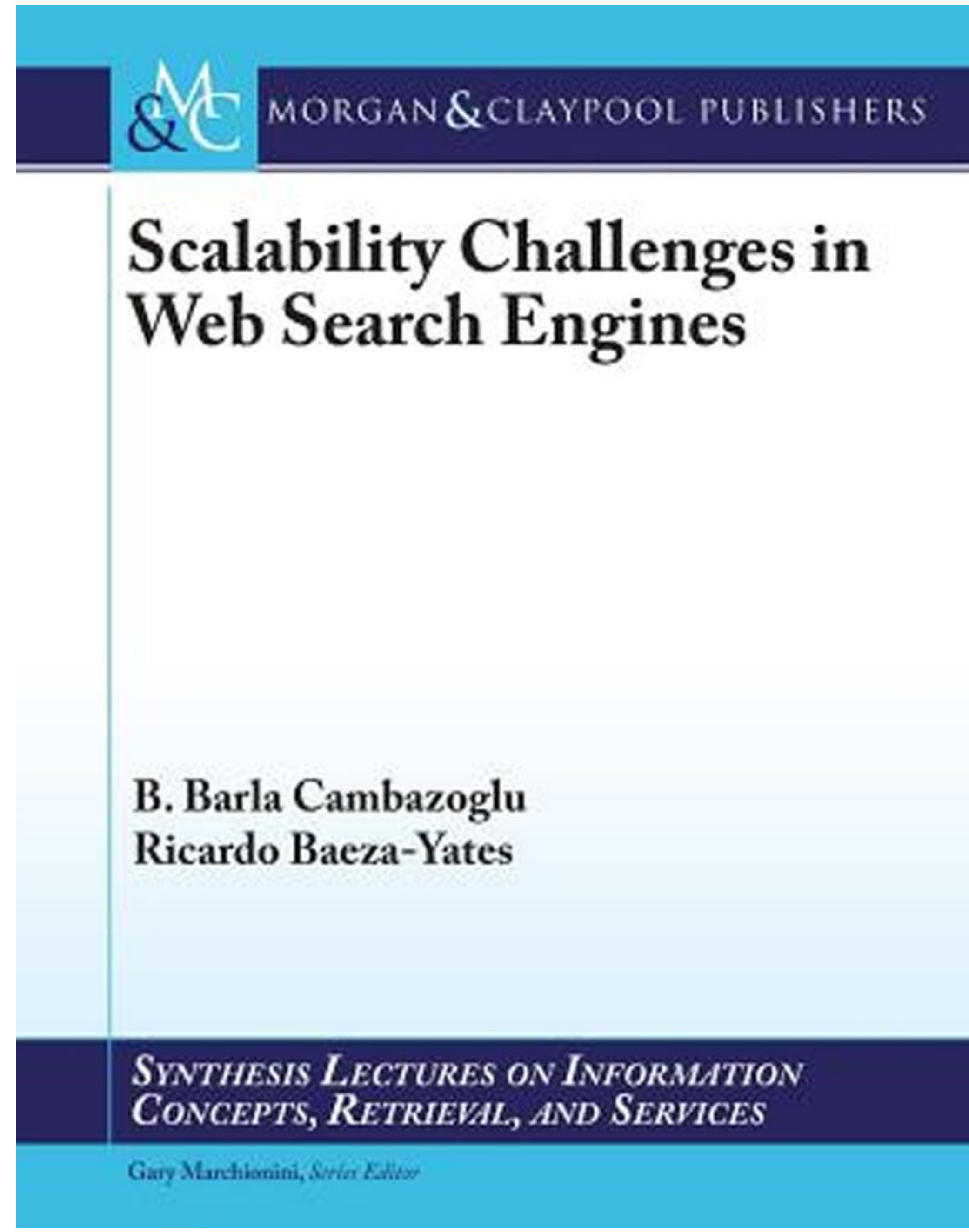
| Method | Λ | 1,000 MSN-1 | 1,000 Y!S1 | 1,000 Istella | 5,000 MSN-1 | 5,000 Y!S1 | 5,000 Istella |
|---|---|---|---|---|---|---|---|
| | | **Number of trees/dataset** | | | | | |
| QS | 8 | **2.2** (–) | **4.4** (–) | **2.8** (–) | **10.1** (–) | **14.5** (–) | **12.0** (–) |
| VPRED | | 7.8 (3.5x) | 8.4 (1.9x) | 7.6 (2.7x) | 39.8 (3.9x) | 41.1 (2.8x) | 39.3 (3.3x) |
| CONDOP | | 6.7 (3.0x) | 10.3 (2.3x) | 8.0 (2.9x) | 67.8 (6.7x) | 75.9 (5.2x) | 77.0 (6.4x) |
| IF-THEN | | 7.3 (3.3x) | 10.1 (2.3x) | 9.0 (3.2x) | 78.2 (7.7x) | 84.6 (5.8x) | 84.1 (7.0x) |
| STRUCT+ | | 21.0 (9.5x) | 23.1 (5.3x) | 24.9 (8.9x) | 98.7 (9.8x) | 119.5 (8.2x) | 117.9 (9.8x) |
| QS | 16 | **3.1** (–) | **6.4** (–) | **4.5** (–) | **15.9** (–) | **21.6** (–) | **17.9** (–) |
| VPRED | | 16.0 (5.2x) | 16.4 (2.6x) | 14.9 (3.3x) | 82.0 (5.2x) | 82.4 (3.8x) | 79.3 (4.4x) |
| CONDOP | | 14.1 (4.5x) | 17.2 (2.7x) | 16.1 (3.6x) | 100.0 (6.3x) | 110.0 (5.0x) | 155.0 (8.7x) |
| IF-THEN | | 18.1 (5.8x) | 20.7 (3.2x) | 19.6 (4.4x) | 128.0 (8.0x) | 128.8 (6.0x) | 135.5 (7.6x) |
| STRUCT+ | | 40.9 (13.2x) | 41.6 (6.5x) | 44.4 (9.9x) | 411.2 (25.9x) | 418.6 (19.4x) | 407.8 (22.8x) |
| QS | 32 | **5.2** (–) | **9.7** (–) | **6.8** (–) | **26.8** (–) | **34.5** (–) | **26.9** (–) |
| VPRED | | 31.8 (6.1x) | 31.5 (3.2x) | 28.1 (4.1x) | 164.5 (6.1x) | 161.6 (4.7x) | 157.7 (5.9x) |
| CONDOP | | 27.0 (5.2x) | 30.3 (3.1x) | 30.4 (4.5x) | NA (x) | NA (x) | NA (x) |
| IF-THEN | | 32.2 (6.2x) | 34.0 (3.5x) | 33.3 (4.9x) | 270.5 (10.1x) | 256.6 (7.4x) | 240.6 (8.9x) |
| STRUCT+ | | 69.4 (13.3x) | 66.5 (6.9x) | 67.8 (10.0x) | 861.0 (32.1x) | 833.2 (24.2x) | 807.9 (x) |
| QS | 64 | **9.4** (–) | **15.1** (–) | **11.2** (–) | **57.6** (–) | **70.2** (–) | **57.8** (–) |
| VPRED | | 62.2 (6.6x) | 57.3 (3.8x) | 54.3 (4.8x) | 347.2 (6.0x) | 333.6 (4.8x) | 326.8 (5.7x) |
| CONDOP | | 48.6 (5.2x) | 48.4 (3.2x) | 51.2 (4.6x) | NA (x) | NA (x) | NA (x) |
| IF-THEN | | 54.0 (5.8x) | 53.2 (3.5x) | 55.0 (4.9x) | 901.1 (15.6x) | 801.9 (11.4x) | 911.2 (15.8x) |
| STRUCT+ | | 132.3 (14.1x) | 109.5 (7.3x) | 112.6 (10.5x) | 1485.5 (25.8x) | 1498.2 (21.3x) | 1487.3 (25.7x) |

| Method | Λ | 10,000 MSN-1 | 10,000 Y!S1 | 10,000 Istella | 20,000 MSN-1 | 20,000 Y!S1 | 20,000 Istella |
|---|---|---|---|---|---|---|---|
| | | **Number of trees/dataset** | | | | | |
| QS | 64 | **158.2** (–) | **156.3** (–) | **146.7** (–) | **428.1** (–) | **335.0** (–) | **289.6** (–) |
| VPRED | | 733.2 (4.6x) | 704.7 (4.5x) | 696.3 (4.7x) | 1307.6 (3.0x) | 1412.9 (4.2x) | 1413.1 (4.9x) |
| CONDOP | | NA (x) | NA (x) | NA (x) | NA (x) | NA (x) | NA (x) |
| IF-THEN | | 2364.3 (14.9x) | 2350.5 (15.0x) | 2334.8 (15.9x) | 4397.1 (10.3x) | 4647.2 (13.9x) | 4678.8 (16.2x) |
| STRUCT+ | | 3014.8 (19.0x) | 2894.4 (18.5x) | 2942.8 (20.1x) | 6794.5 (15.9x) | 6923.9 (20.7x) | 7586.4 (26.2x) |

# Resources

- Terrier (http://www.terrier.org)
  - Java, highly flexible, efficient, and effective open source search engine
- Galago (http://sourceforge.net/projects/lemur)
  - Java, toolkit for experimenting with text search
- Indri (https://www.lemurproject.org/indri/)
  - C++, new search engine from the Lemur project
- Lucene (https://lucene.apache.org)
  - Java, indexing and search technology
- Anserini (https://github.com/castorini/anserini)
  - Java, open-source information retrieval toolkit built on Lucene
- ds2i (https://github.com/ot/ds2i)
  - C++, library of data structures to represent the integer sequences used in inverted indexes
- ATIRE (http://atire.org/)
  - C++, fast and accurate search engine
- JASS (https://codedocs.xyz/andrewtrotman/JASSv2/)
  - C++, experimental search engine


Foundations and Trends® in
Information Retrieval
12:4-5
Efficient Query Processing
for Scalable Web Search
Nicola Tonellotto, Craig Macdonald
and Iadh Ounis
now
the essence of knowledge


MORGAN & CLAYPOOL PUBLISHERS
Scalability Challenges in
Web Search Engines
B. Barla Cambazoglu
Ricardo Baeza-Yates
SYNTHESIS LECTURES ON INFORMATION
CONCEPTS, RETRIEVAL, AND SERVICES
Gary Marchionini, Series Editor

Thanks for your attention!