# Disclaimer

- Some of these slides have been prepared by B. Barla Cambazoglu, Director of Applied Science, NTENT Inc.
- Some of these slides have been prepared by Craig Macdonald (University of Glasgow) and Nicola Tonellotto (ISTI-CNR)
- Used with permission, please do not redistribute in any form.
- Please refer to:

Morgan & Claypool Publishers

Scalability Challenges in Web Search Engines

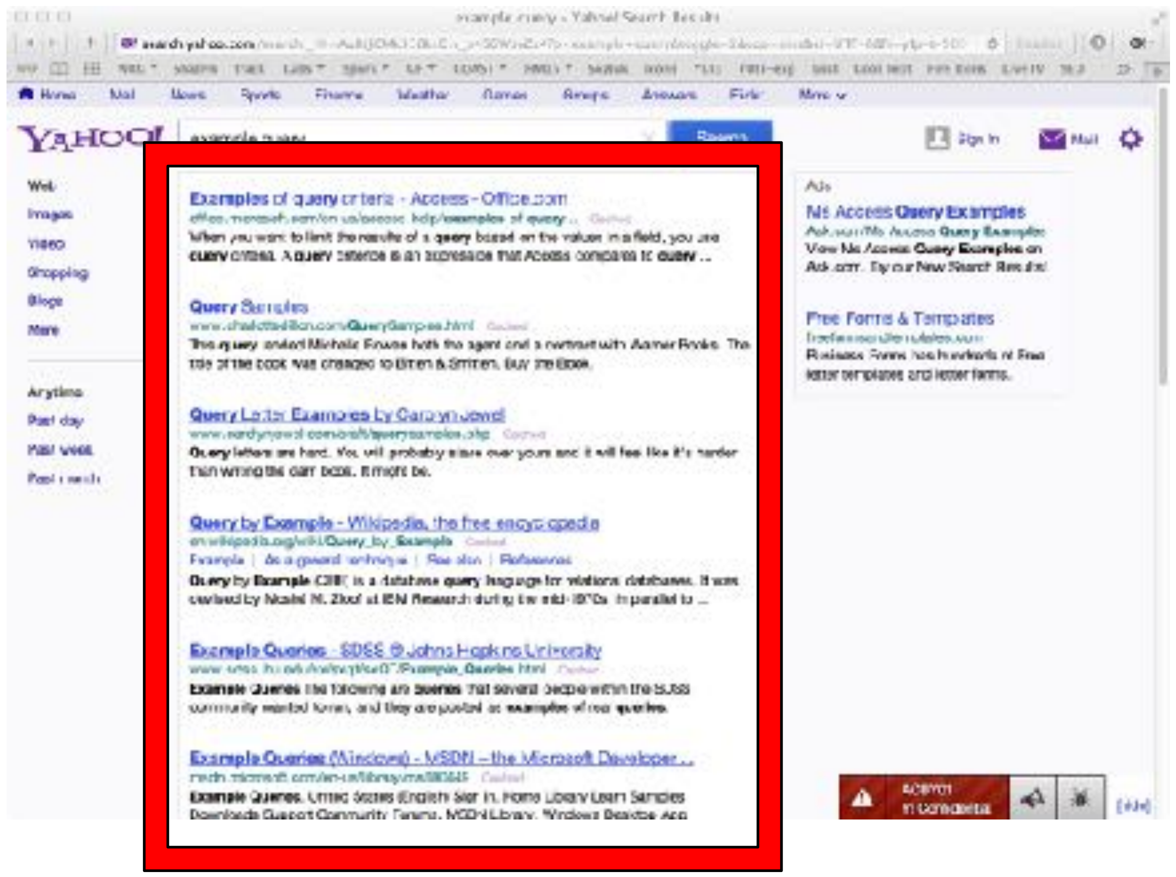
B. Barla Cambazoglu
Ricardo Baeza-Yates

Synthesis Lectures on Information Concepts, Retrieval, and Services

# Query Processing

- Query processing is the problem of generating the best-matching answers (typically, top 10 documents) to a given user query, spending the least amount of time.

- Our focus: creating 10 blue links as an answer to a user query.



Examples of query criteria - Access - Office.com
office.microsoft.com/en-us/access-help/examples-of-query... Cached
When you want to limit the results of a query based on the values in a field, you use query criteria. A query criterion is an expression that Access compares to query ...

Query Samples
www.charlottedillon.com/QuerySamples.html Cached
This query landed Michelle Rowan both the agent and a contract with Warner Books. The title of the book was changed to Bitten & Smitten. Buy the Book.

Query Letter Examples by Carolyn Jewel
www.carolynjewel.com/craft/querysamples.php Cached
Query letters are hard. You will probably slave over yours and it will feel like it's harder than writing the darn book. It might be.

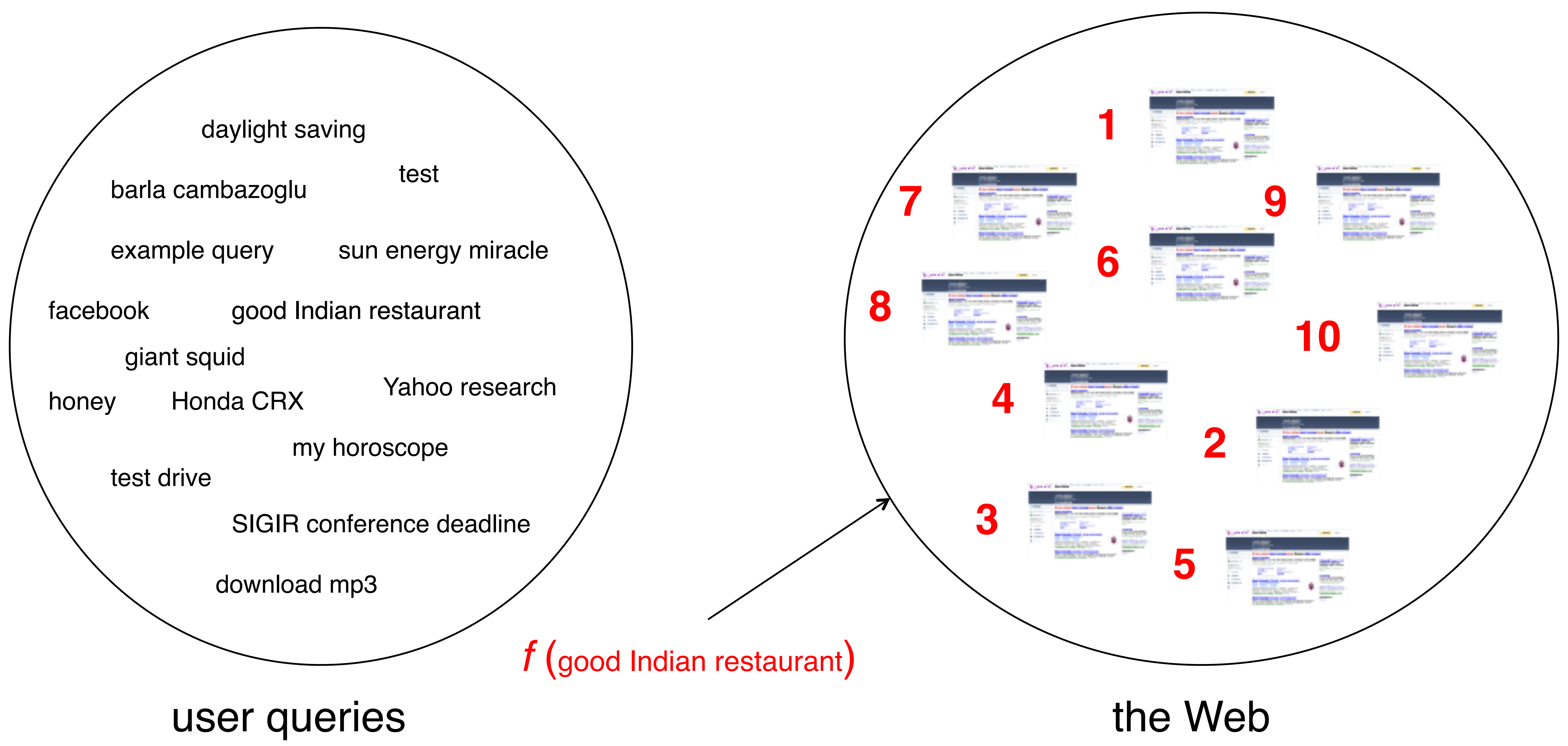
Query by Example - Wikipedia, the free encyclopedia
en.wikipedia.org/wiki/Query_by_Example Cached
Example | As a general technique | See also | References
Query by Example (QBE) is a database query language for relational databases. It was devised by Moshé M. Zloof at IBM Research during the mid-1970s, in parallel to ...

Example Queries - SDSS @ Johns Hopkins University
www.sdss.jhu.edu/[illegible]/Example_Queries.html Cached
Example Queries The following are queries that several people within the SDSS community wanted to run, and they are posted as examples of real queries.

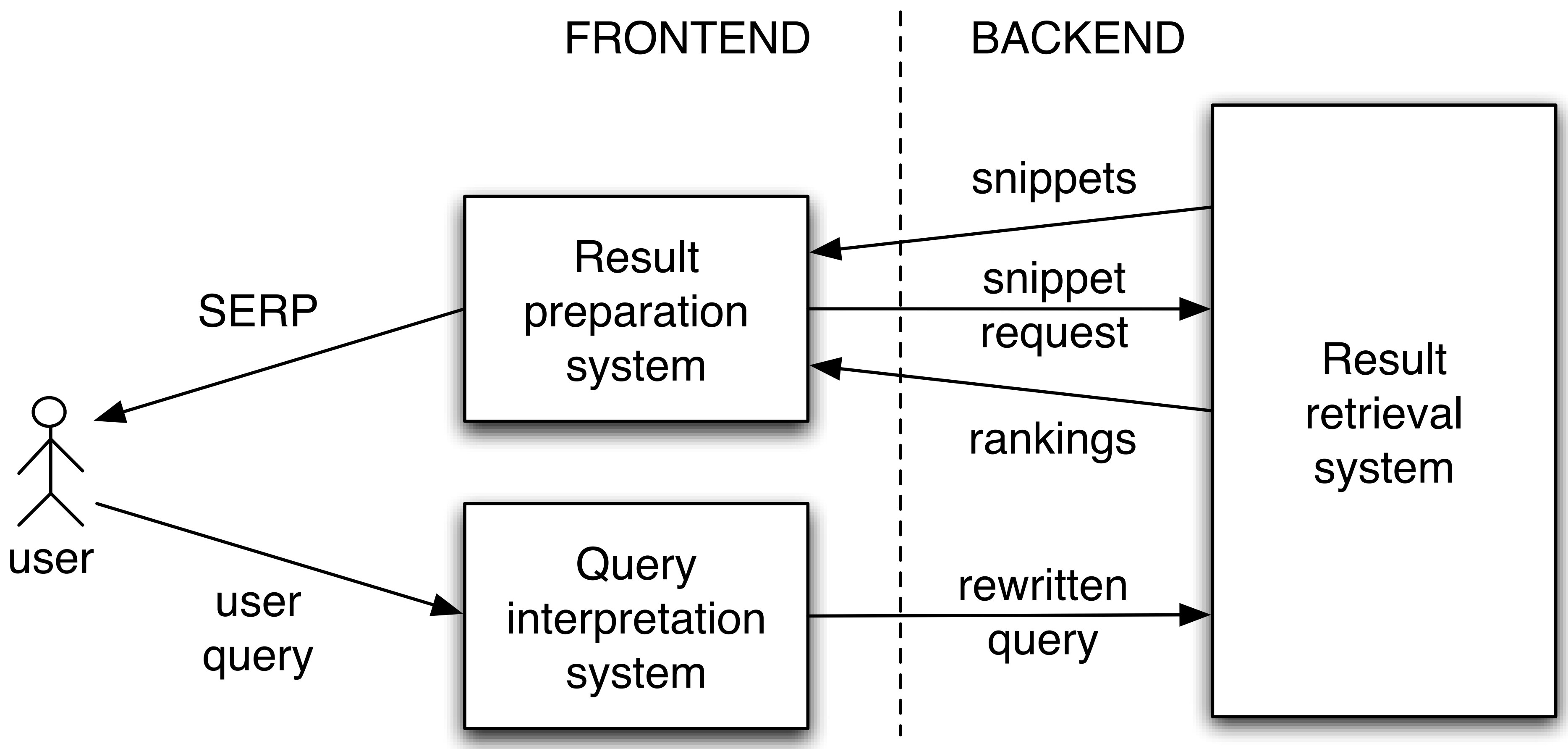
Example Queries (Windows) - MSDN – the Microsoft Developer ...
msdn.microsoft.com/en-us/library/[illegible] Cached
Example Queries. United States (English) Sign In. Home Library Learn Samples Downloads Support Community Forums. MSDN Library. Windows Desktop App

# Web Search

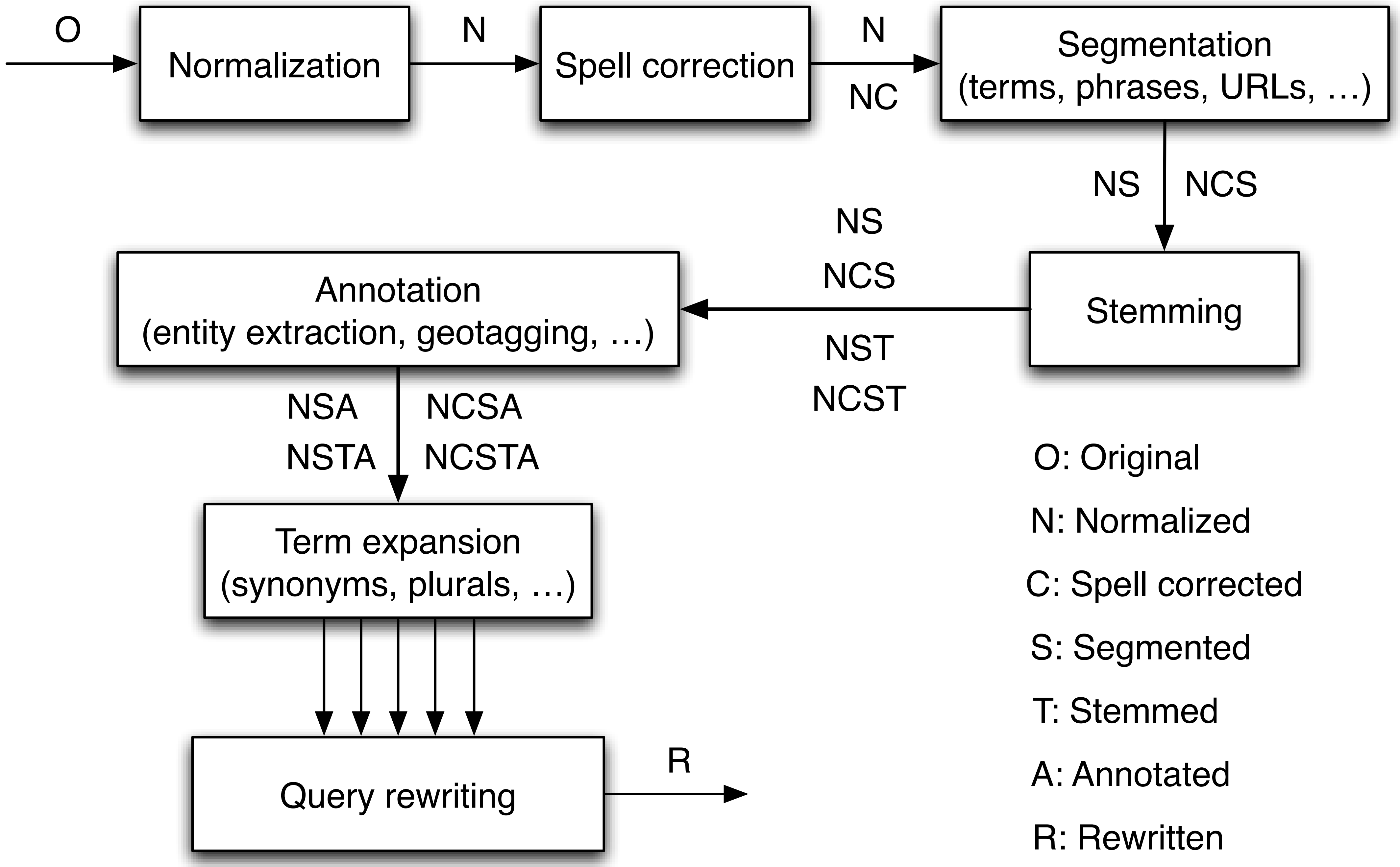
- Web search is a sorting problem!



# Query Processing



# Query Interpretation System


O
Normalization
N
Spell correction
N
NC
Segmentation (terms, phrases, URLs, …)
NS
NCS
Stemming
NS
NCS
NST
NCST
Annotation (entity extraction, geotagging, …)
NSA
NSTA
NCSA
NCSTA
Term expansion (synonyms, plurals, …)
Query rewriting
R

O: Original

N: Normalized

C: Spell corrected

S: Segmented

T: Stemmed

A: Annotated

R: Rewritten

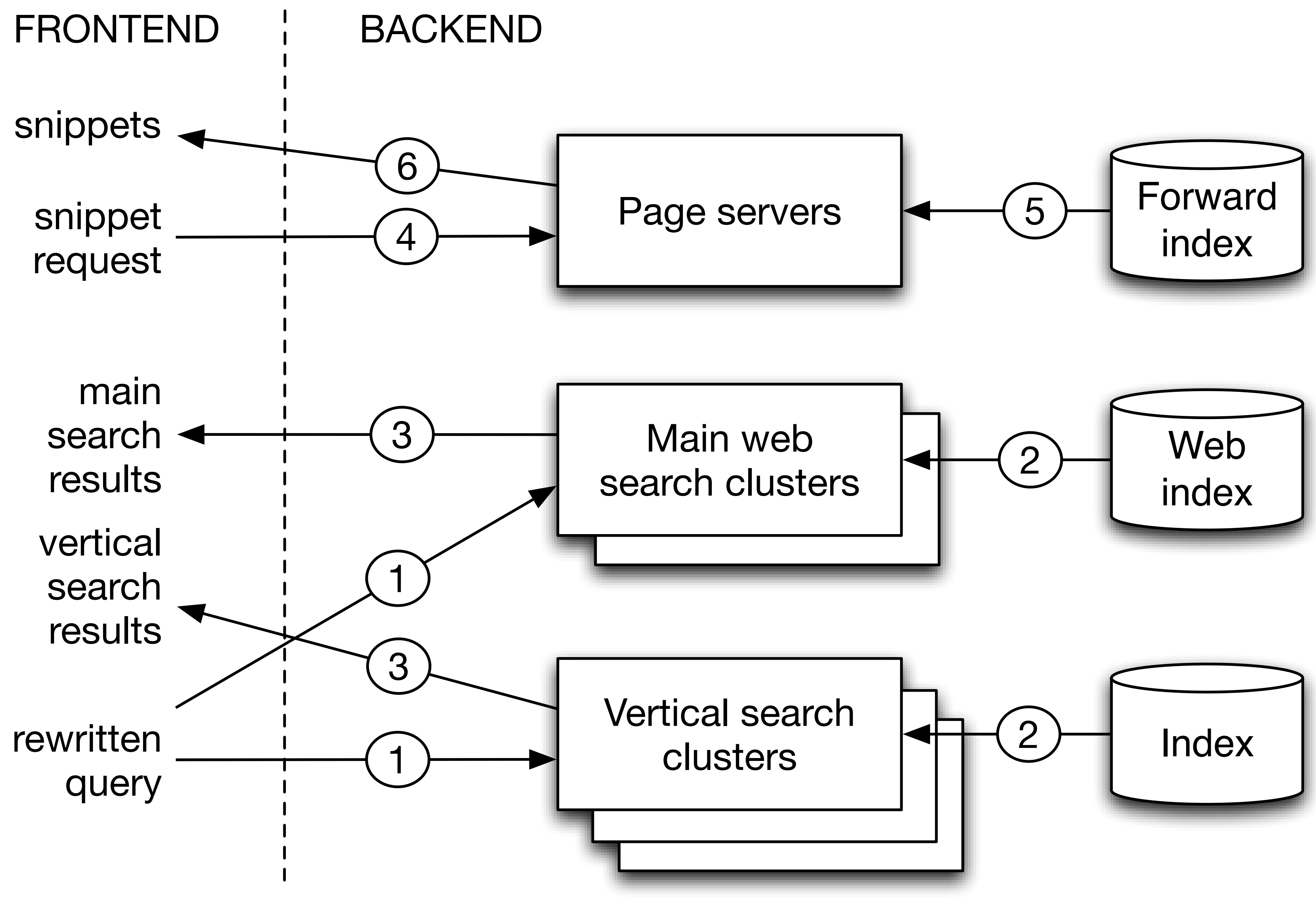
# Query Rewriting

- The original user query: “*amusement arcades in New York”.*
- Internal system query:

```
AND(OR(PHRASE(amusement arcade),
          PHRASE(video arcade)
       )
       LOCATION(new york)
   )
```

- Applied modifications
  1. stop word “*in*” is removed
  2. term “*arcades*” is converted into its singular form “*arcade*”
  3. “*amusement arcade*” is detected as a phrase and expanded to “*video arcade*”
  4. “*New York*” is detected as a location and converted to lower case

# Results Generation



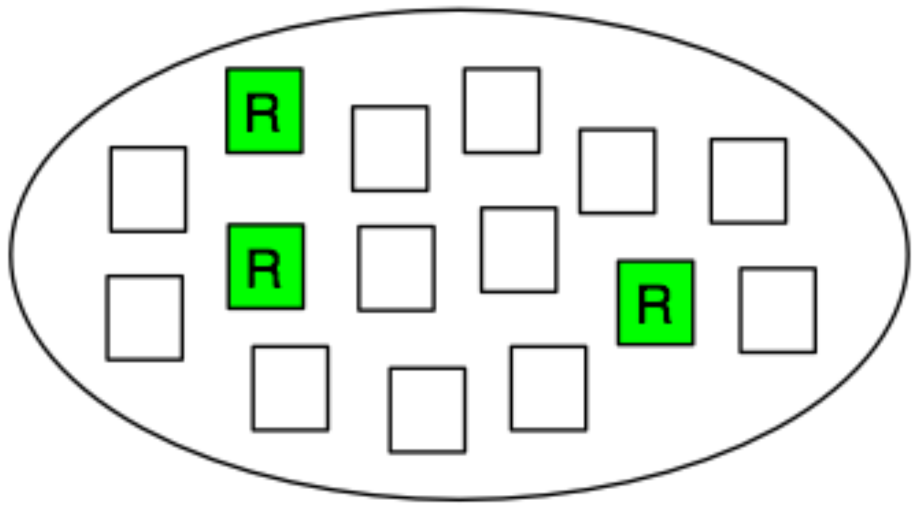
## Success Measures

- Quality measures
  - result quality: degree to which returned answers meet user’s information need.
- Performance measures
  - latency: response time delay experienced by the user
  - peak throughput: number of queries that can be processed per unit of time without any degradation in other metrics

# Measuring Results Quality

- Sources of feedback
  - editorial
  - user clicks

- Common measures
  - recall
  - precision
  - DCG
  - NDCG



| | Ranking 1 | Ranking 2 | Optimal |
|---|---|---|---|
| 1. | 🟩 R | | 🟩 R |
| 2. | | 🟩 R | 🟩 R |
| 3. | | | 🟩 R |
| 4. | | | |
| Recall: | 1/3 | 1/3 | 1 |
| Precision: | 1/4 | 1/4 | 3/4 |
| DCG: | 1 | 0.63 | 1+0.63+0.5=2.13 |
| NDCG: | 1/2.13 | 0.63/2.13 | |

# Query Evaluation

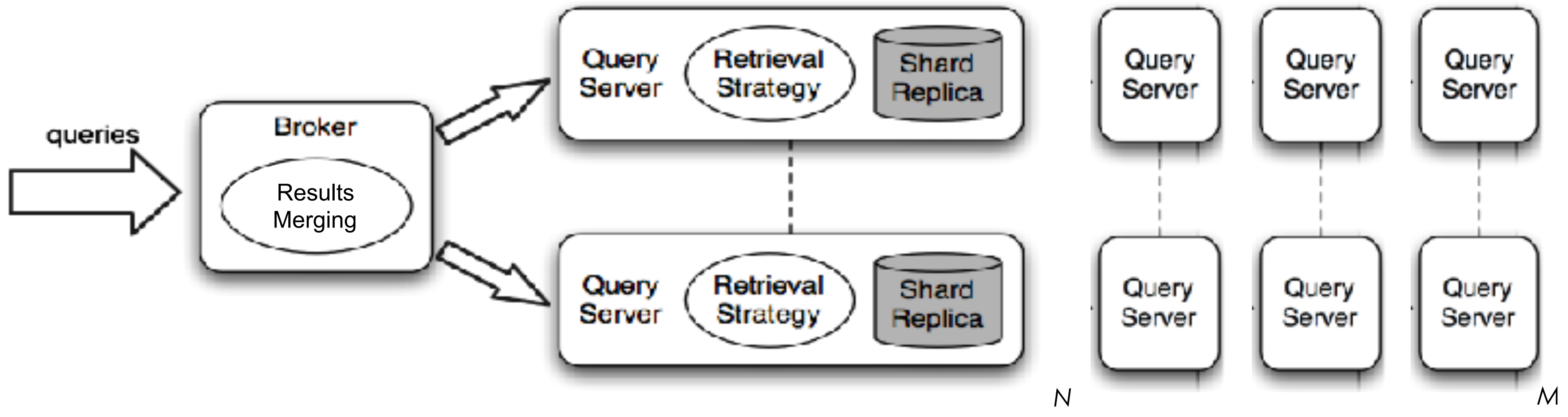
- Conjunctive (AND) vs. Disjunctive (OR)
- Boolean retrieval
  - Suitable for small/medium collections
  - Returns all documents matching the query
- Ranked retrieval
  - Suitable for Web-scale collections
  - Requires a similarity function between queries and documents

$$\textsc{Score}_q(d) = \sum_{t \in q} s_t(q, d)$$

  - Returns only the top k document

## Query Processing Infrastructures

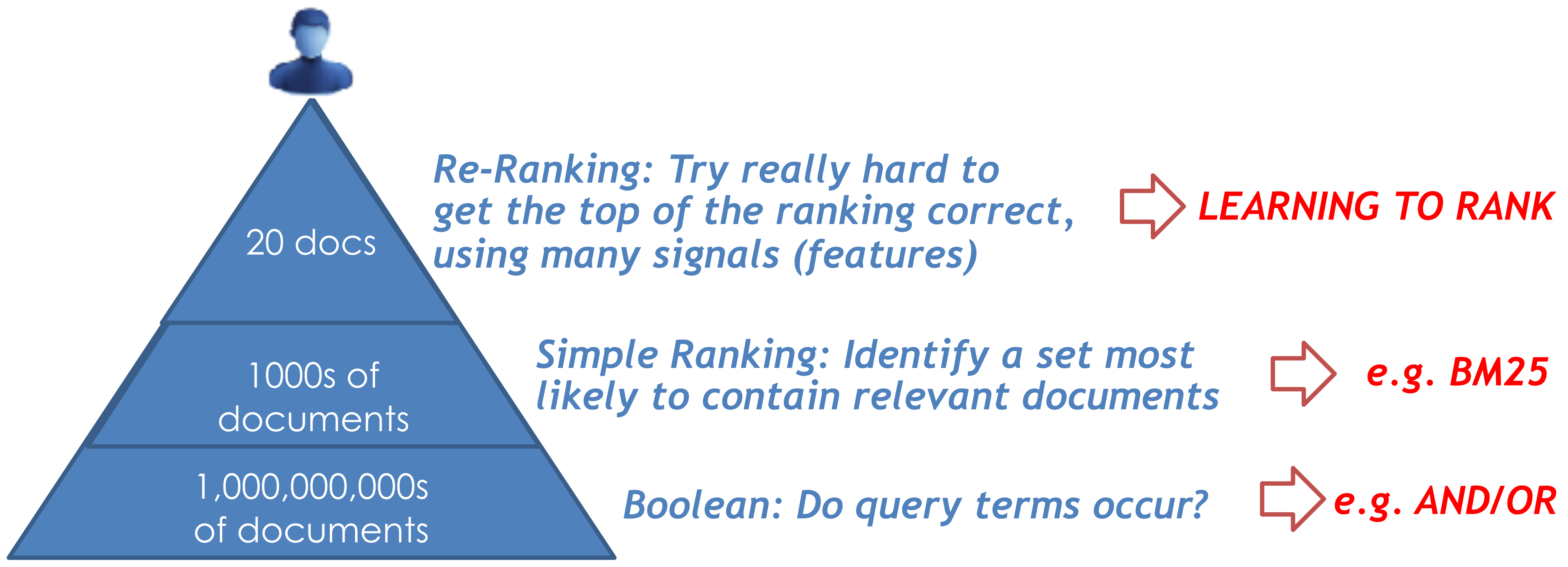
- To achieve efficiency at Big Data scale, search engines use many servers:



- *N* & *M* can be very big:
  – Microsoft's Bing search engine has "*hundreds of thousands of query servers*"

# Ranking Cascades

- Typically, in web-scale search, the ranking process can be conceptually seen as a series of cascades [1]
  - Rank some documents
  - Pass top-ranked onto next cascade for refined re-ranking


20 docs
1000s of documents
1,000,000,000s of documents
Re-Ranking: Try really hard to get the top of the ranking correct, using many signals (features)
LEARNING TO RANK
Simple Ranking: Identify a set most likely to contain relevant documents
e.g. BM25
Boolean: Do query terms occur?
e.g. AND/OR

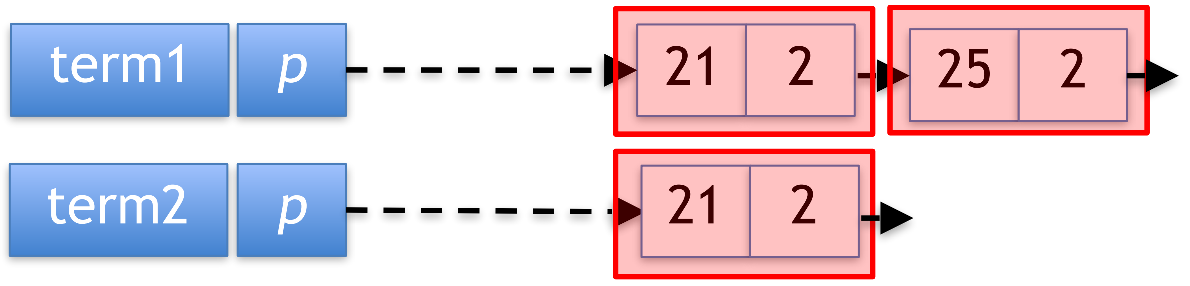
*[1] J Pederson. Query understanding at Bing. SIGIR 2010 Industry Day.*

## Query Evaluation

- Normal strategies make a pass on the postings lists for each query term
  - This can be done Term-at-a-Time (TAAT) – one query term at a time
  - Or Document-at-a-time (DAAT) – all query terms in parallel

- We will explain these, before showing how we can improve them

# Term-at-a-Time (TAAT)



| rank | docid | score |
|---|---|---|
| 1 | *21* | ~~2~~ 4 |
| 2 | *25* | *2* |
| … | | |

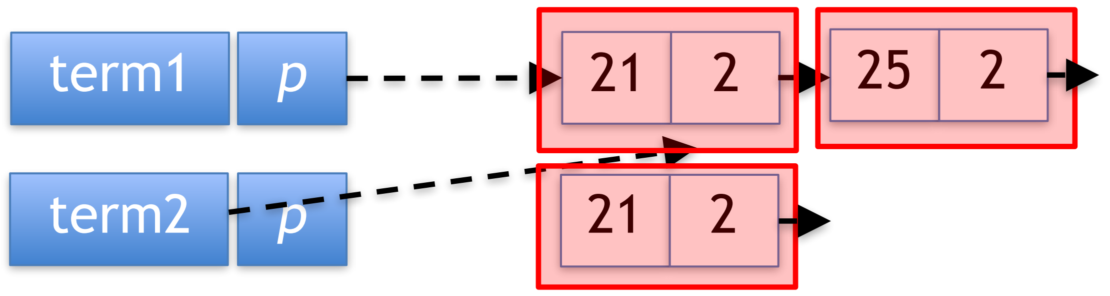
## Advantages:

- Simple

## Disadvantages:

- Requires lots of memory to contain partial scores for all documents
- Difficult to do Boolean or phrase queries, as we don't have a document's postings for all query terms at once

# Document-at-a-Time (DAAT)



| rank | docid | score |
|---|---|---|
| 1 | *21* | *4* |
| 2 | *25* | *2* |
| … | | |

**Advantages:**

- Reduced memory compared to TAAT (and hence faster)
- Supports Boolean query operators, phrases, etc.

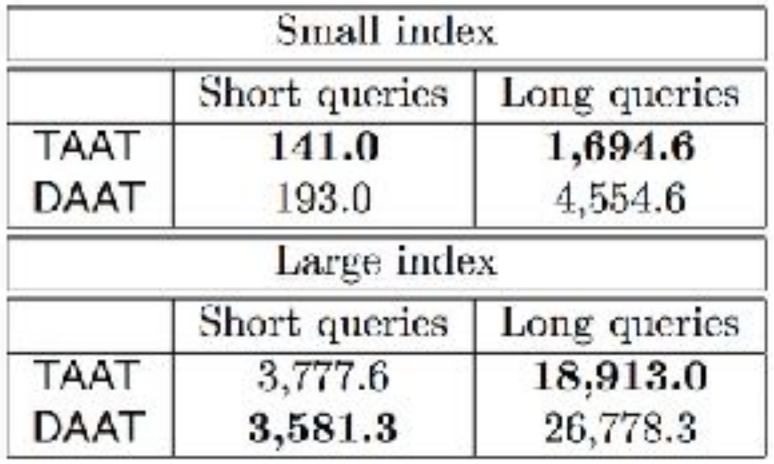
**Disadvantages:**

- Slightly more complex to implement

**Most commercial search engines are reported to use DAAT**

# TAAT vs DAAT

- TAAT and DAAT have been the cornerstores of query evaluation in IR systems since 1970s.
- The plain implementations of these two strategies are seldom used anymore, since many optimizations have been proposed during the years
- Several known systems in production today, from large scale search engines as Google and Yahoo!, to open source text indexing packages as Lucene and Lemur, use some optimized variation of these strategies
- [Turtle and Flood,1995] were the first to argue that DAAT could beat TAAT in practical environments.
  – For large corpora, DAAT implementations require a smaller run-time memory footprint
- [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 |

# Dynamic Pruning

- 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

- We call this "*safe-to-rank K*"

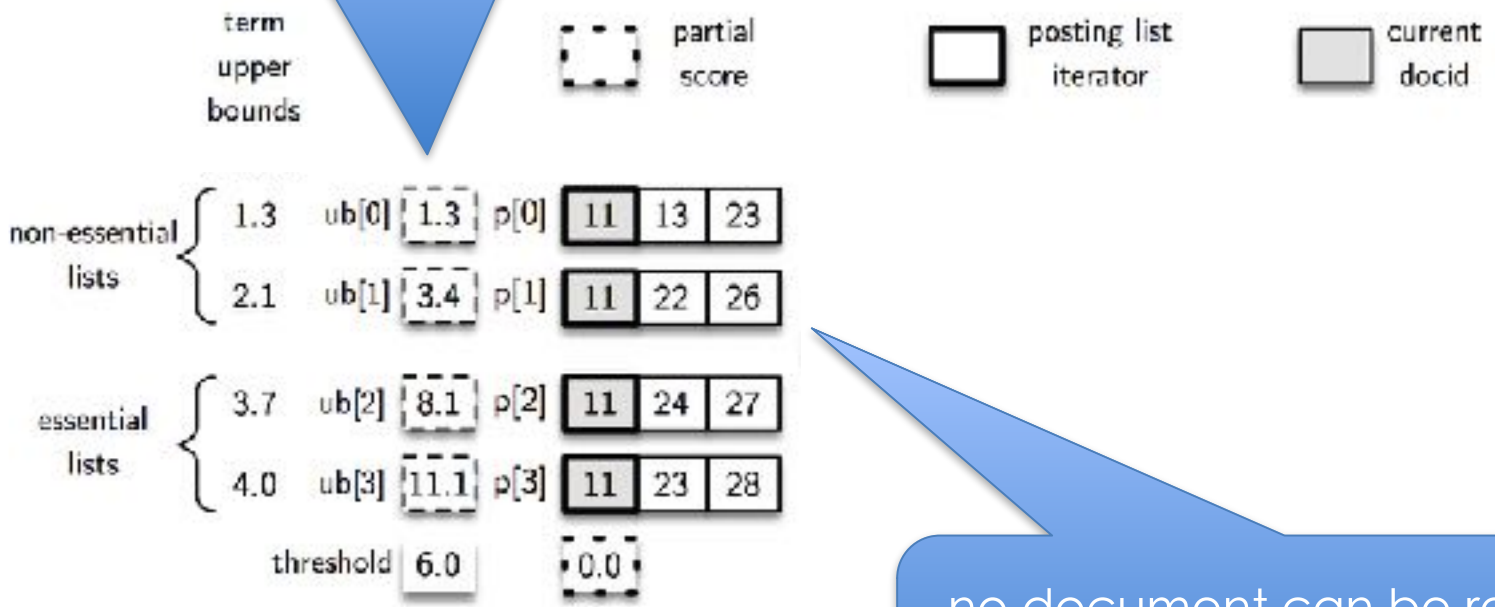
- Dynamic pruning is based upon
  - *Early termination*
  - *Comparing upper bounds on retrieval scores with thresholds*

# Techniques

- MaxScore
  - **Early termination**: does not compute scores for documents that won't be retrieved by comparing **upper bounds** with a score **threshold**
- WAND
  - **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 (w**AND**)
- BlockMaxWand
  - SOTA variant of WAND that uses benefits from the block-layout of posting lists

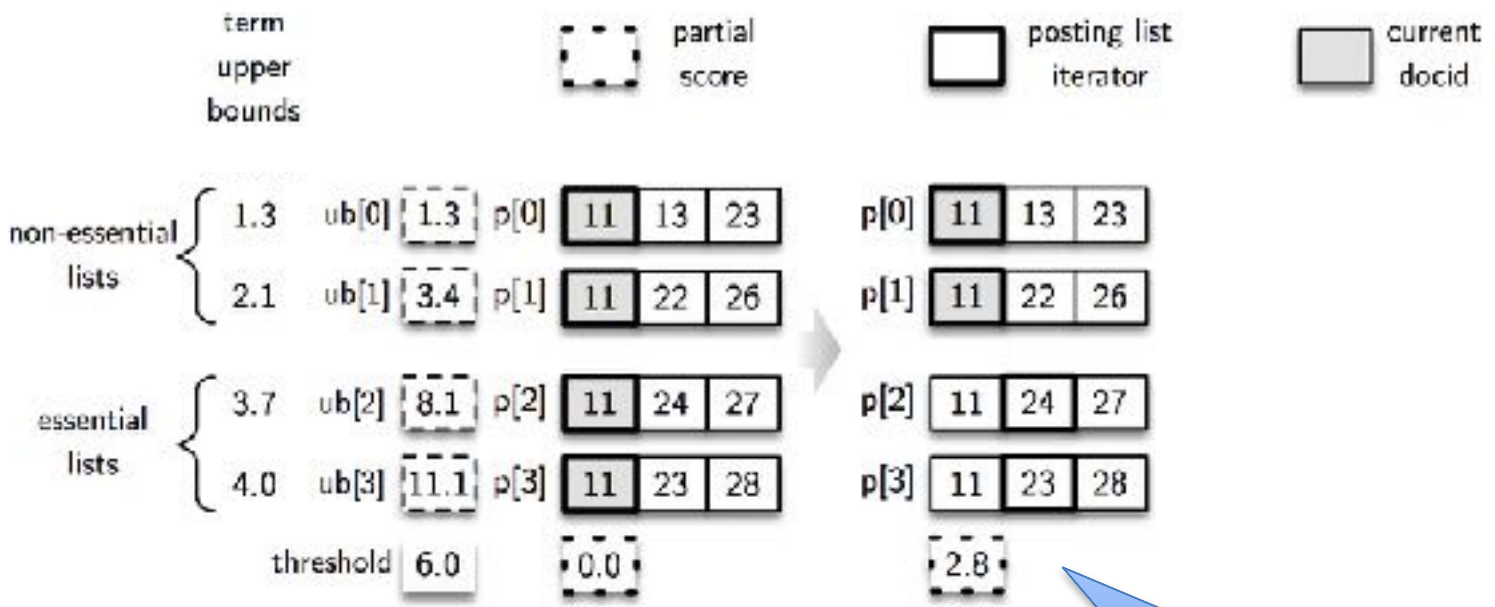
# MaxScore Example

Sorted in increasing order of term upper bound



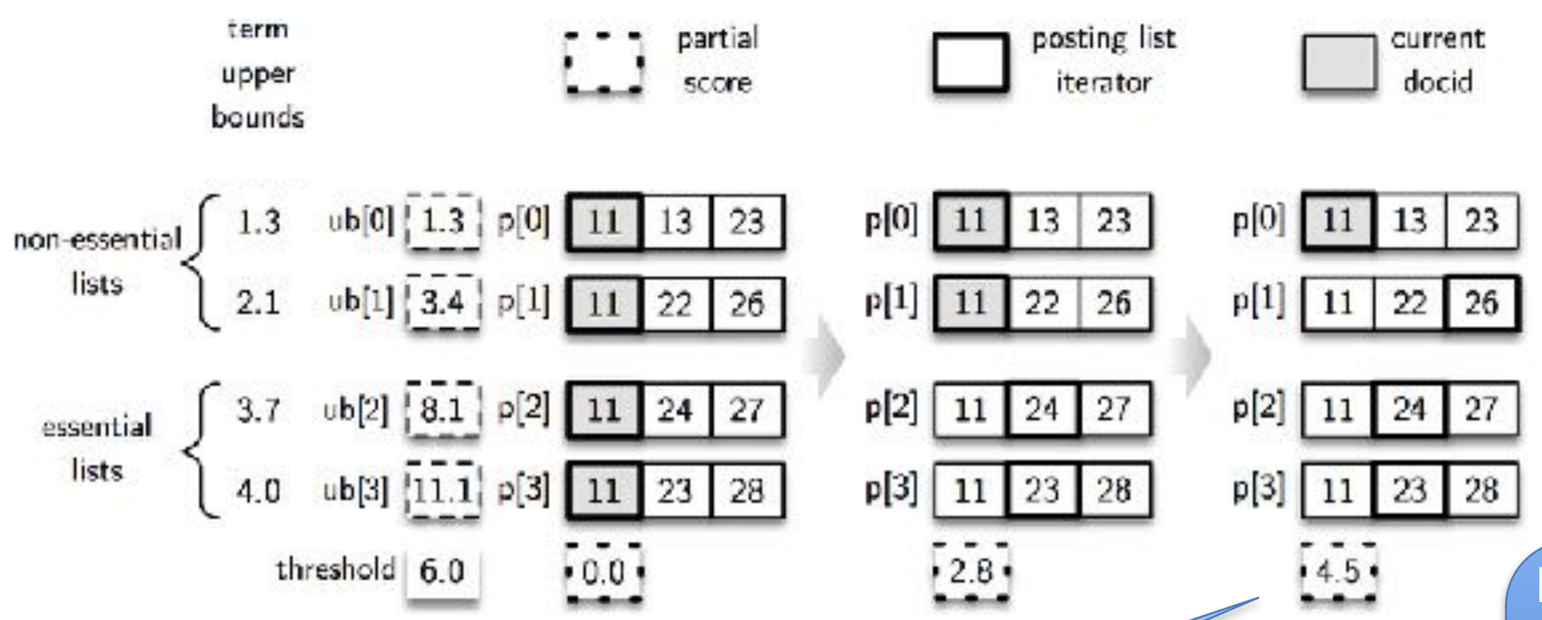
no document can be returned as a top k results if it appears in the non-essential lists only, i.e., 1.3 + 3.4 < 6.0

# MaxScore Example



Essential lists are processed in DAAT, and their score contributions computed

# MaxScore Example



As soon as partial score implies early termination, Move to next document.

Partial score is updated.

process the non essential lists by skipping to the candidate docid.

## Learning

- How to choose term weighting models?
  - 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
  - **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?**

# Learning to Rank

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

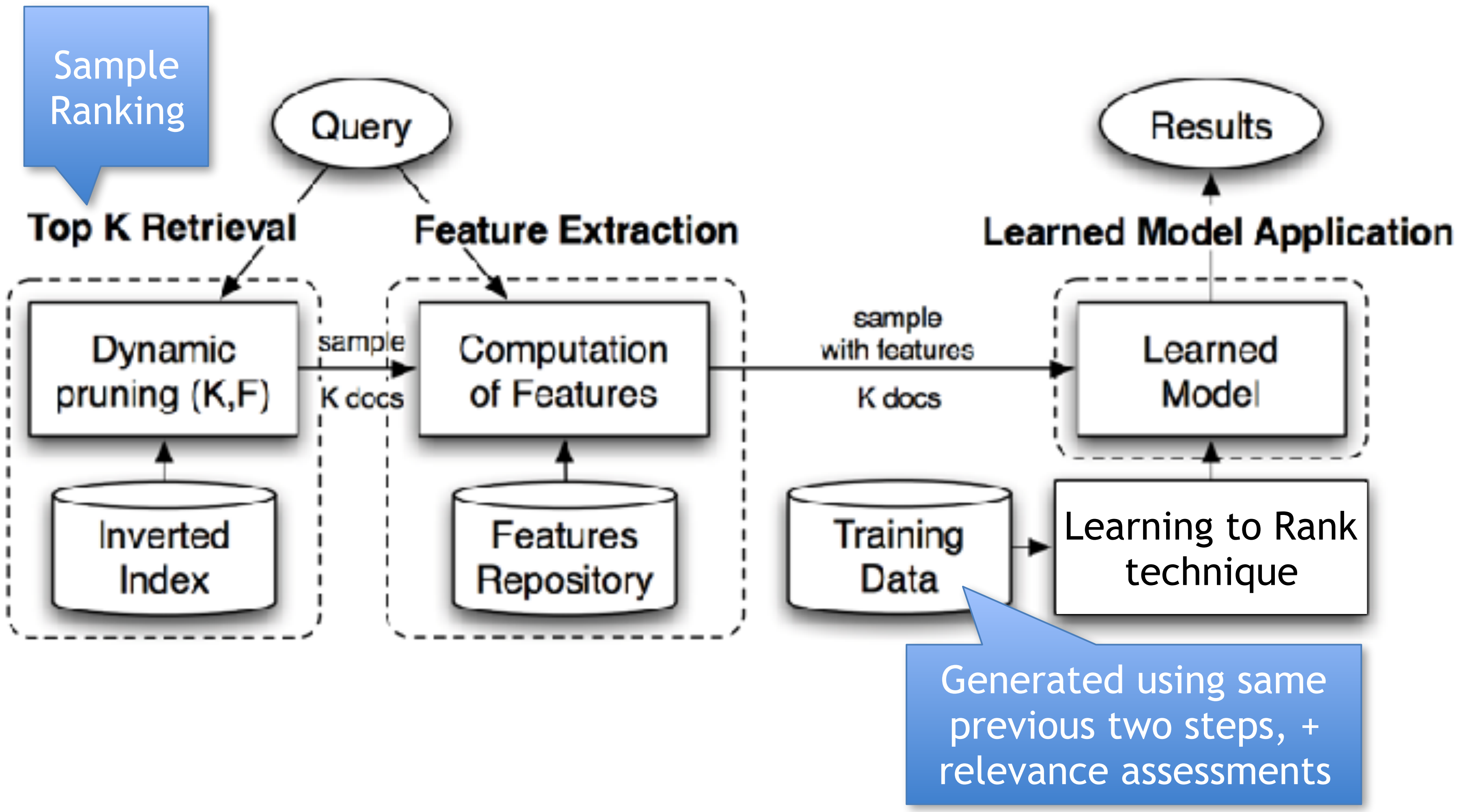
- Recently, learning to rank has been popularised by commercial search engines (e.g. Bing, Baidu, Yandex)
  - They require large training datasets, possibly instantiated from click-through data
  - **Click-through data** has facilitated the deployment of learning approaches

# Features

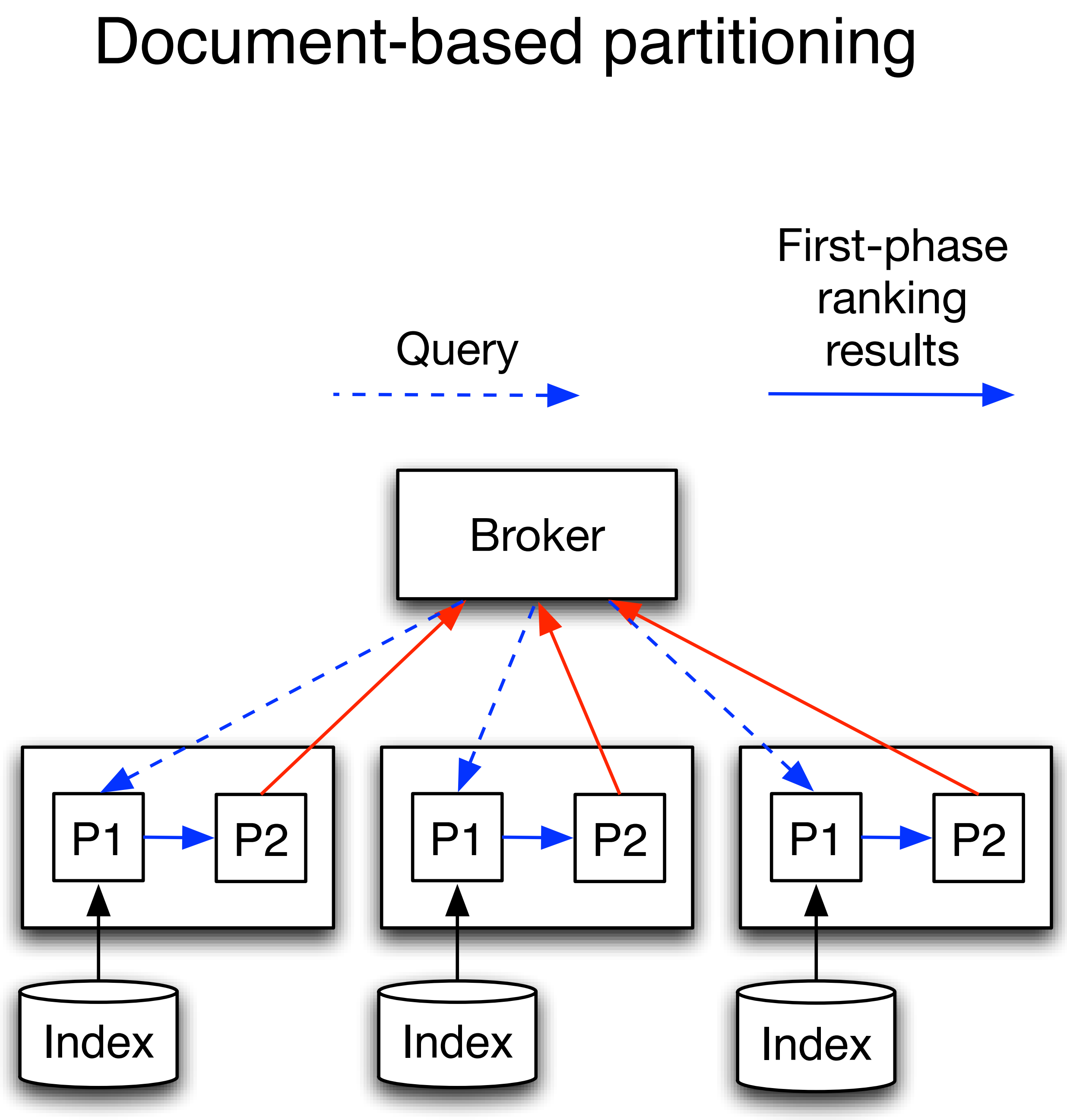
Typically, commercial search engines use hundreds of features for ranking documents, usually categorised as follows:

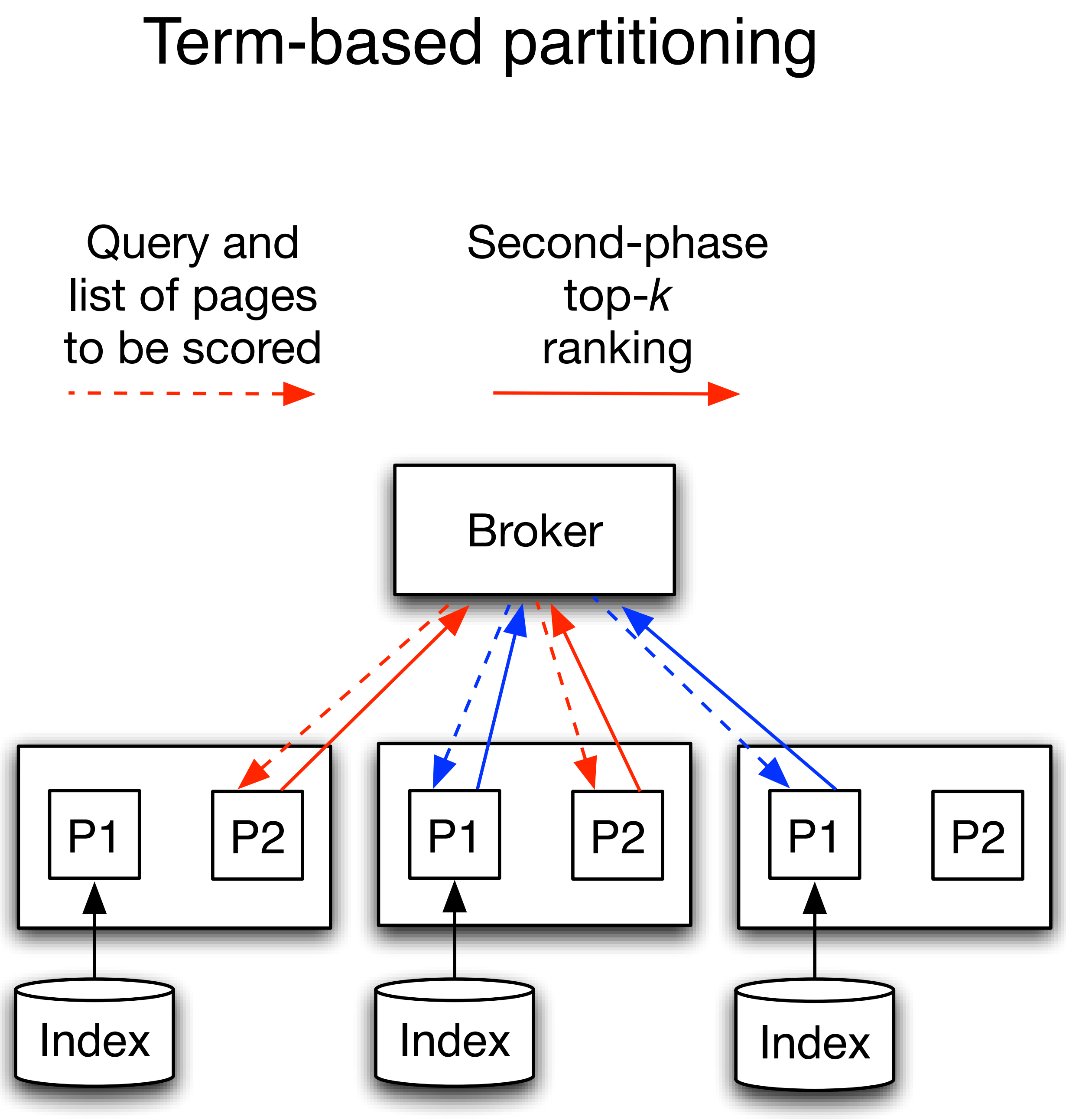
| Name | Varies depending on… | | Examples |
|---|---|---|---|
| | Query | Document | |
| Query Dependent Features | ✔ | ✔ | Weighting models, e.g. BM25, PL2<br>Proximity models, e.g. Markov Random Fields<br>Field-based weighting models, e.g. PL2F |
| Query Independent Features | ✗ | ✔ | PageRank, number of inlinks<br>Spamminess |
| Query Features | ✔ | ✗ | Query length<br>Presence of entities |

# Schematically



# Parallel Query Processing on a search cluster




Term-based partitioning
Query and list of pages to be scored
Second-phase top-k ranking
Broker
P1
P2
P1
P2
P1
P2
Index
Index
Index

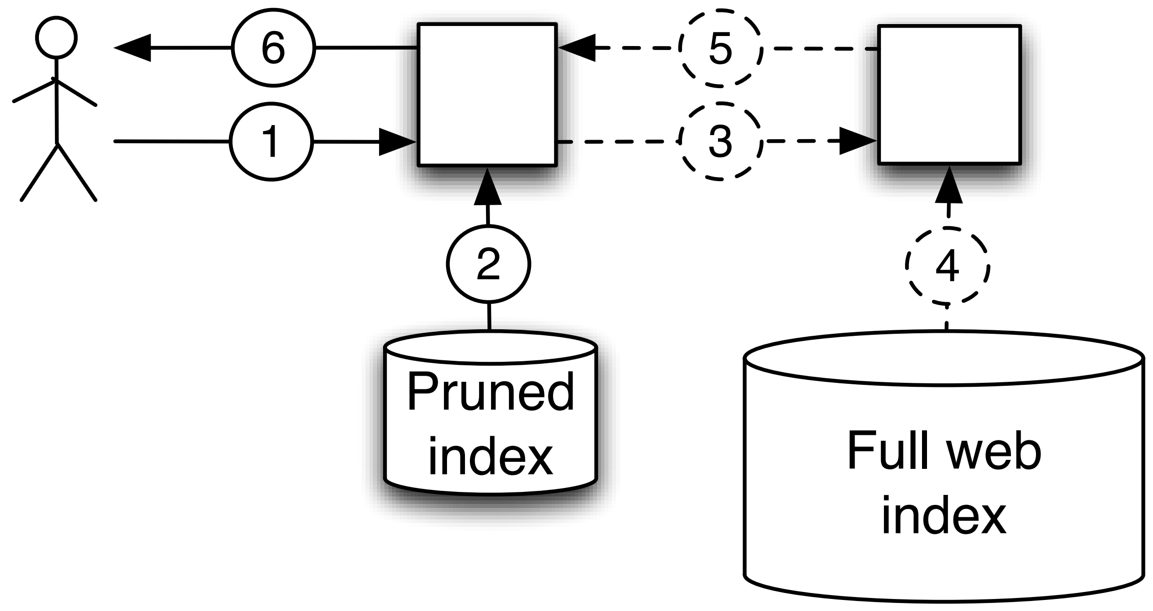
## Architectural Optimizations

- Static index pruning
- Index tiering
- Selective Search
- Caching

# Static Index Pruning

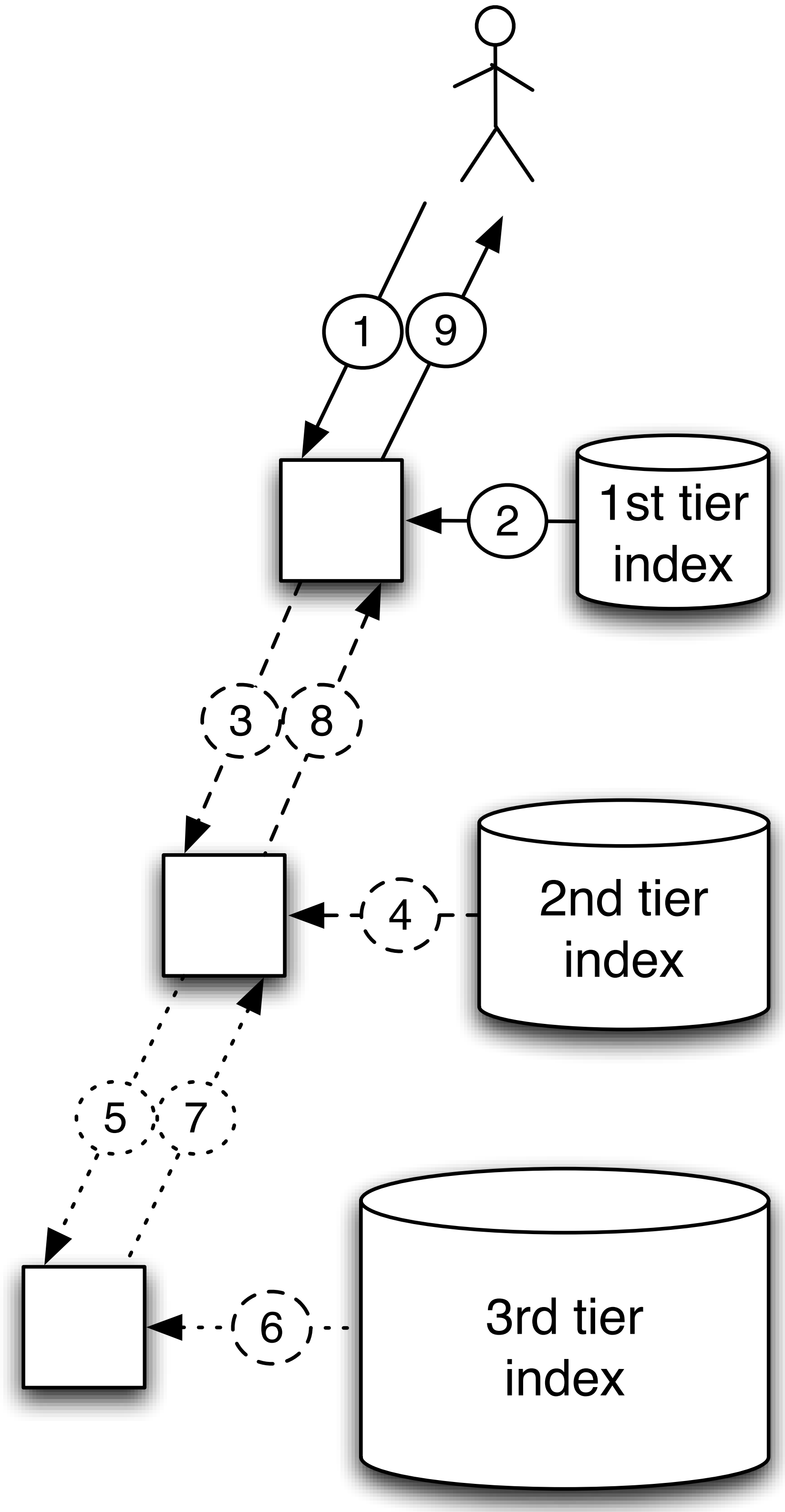
- Idea: to create a small version of the search index that can accurately answer most search queries

- Techniques
  - term-based pruning
  - doc-based pruning
- Result quality
  - guaranteed
  - not guaranteed



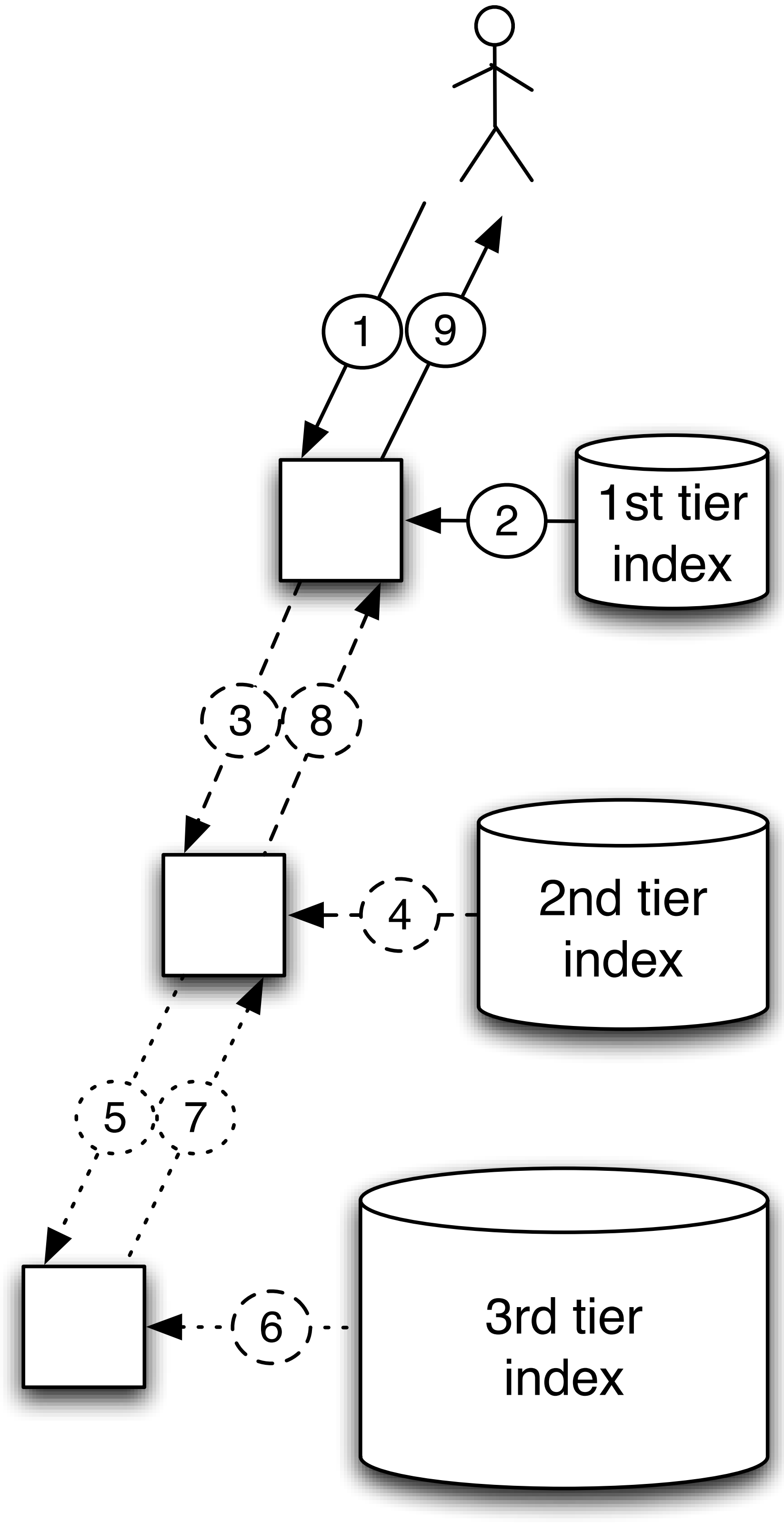
# Tiering

- A sequence of sub-indexes
  - former sub-indexes are small and keep more important documents
  - later sub-indexes are larger and keep less important documents
  - a query is processed selectively only on the first *n* tiers
- Two decisions need to be made
  - tiering (offline): how to place documents in different tiers
  - fall-through (online): at which tier to stop processing the query



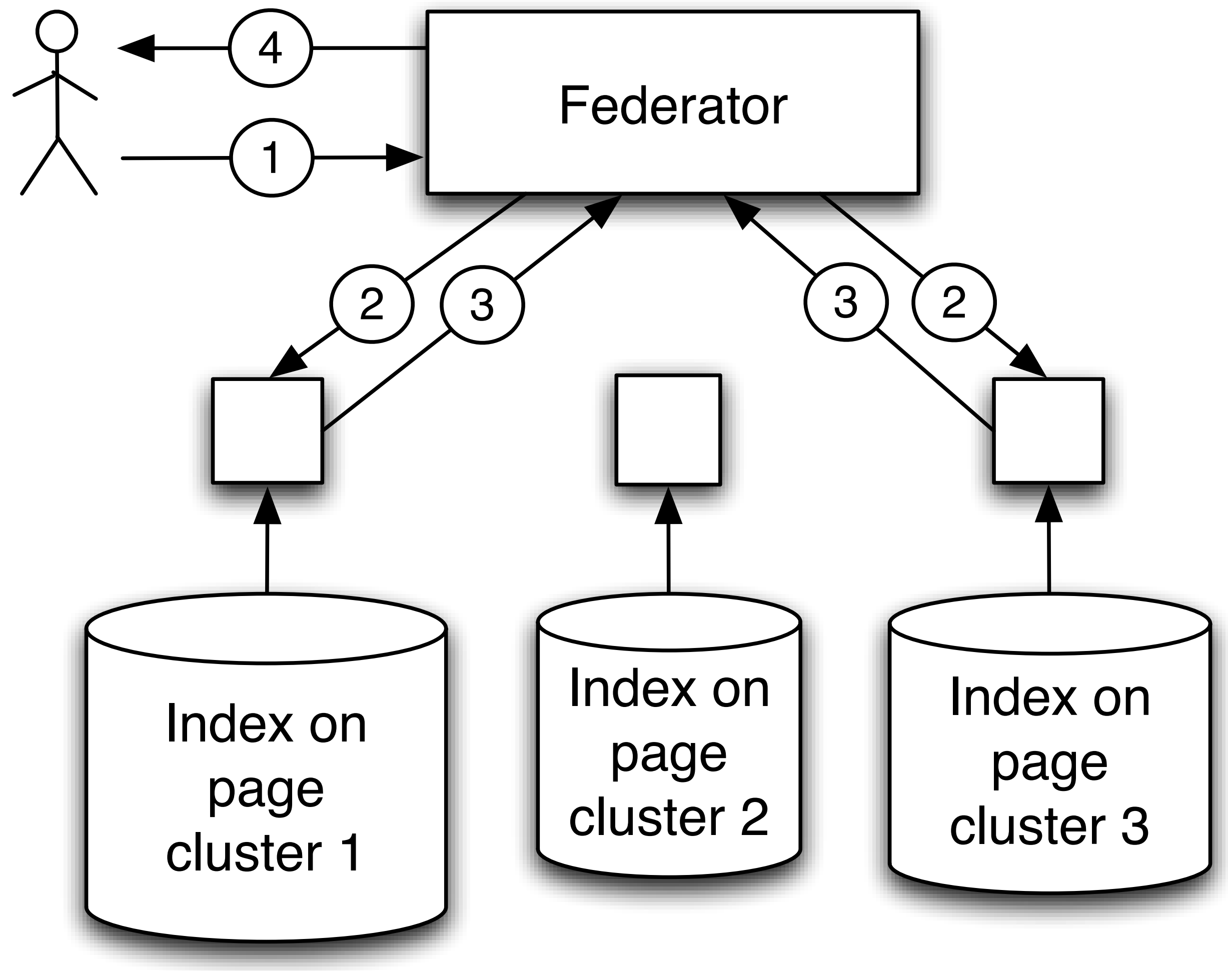
# Tiering

- Tiering strategy is based on some document importance metric
  - PageRank
  - click count
  - spam score
- Fall-through strategy
  - query the next index until there are enough results
  - query the next index until search result quality is good
  - predict the next tier's result quality by machine learning



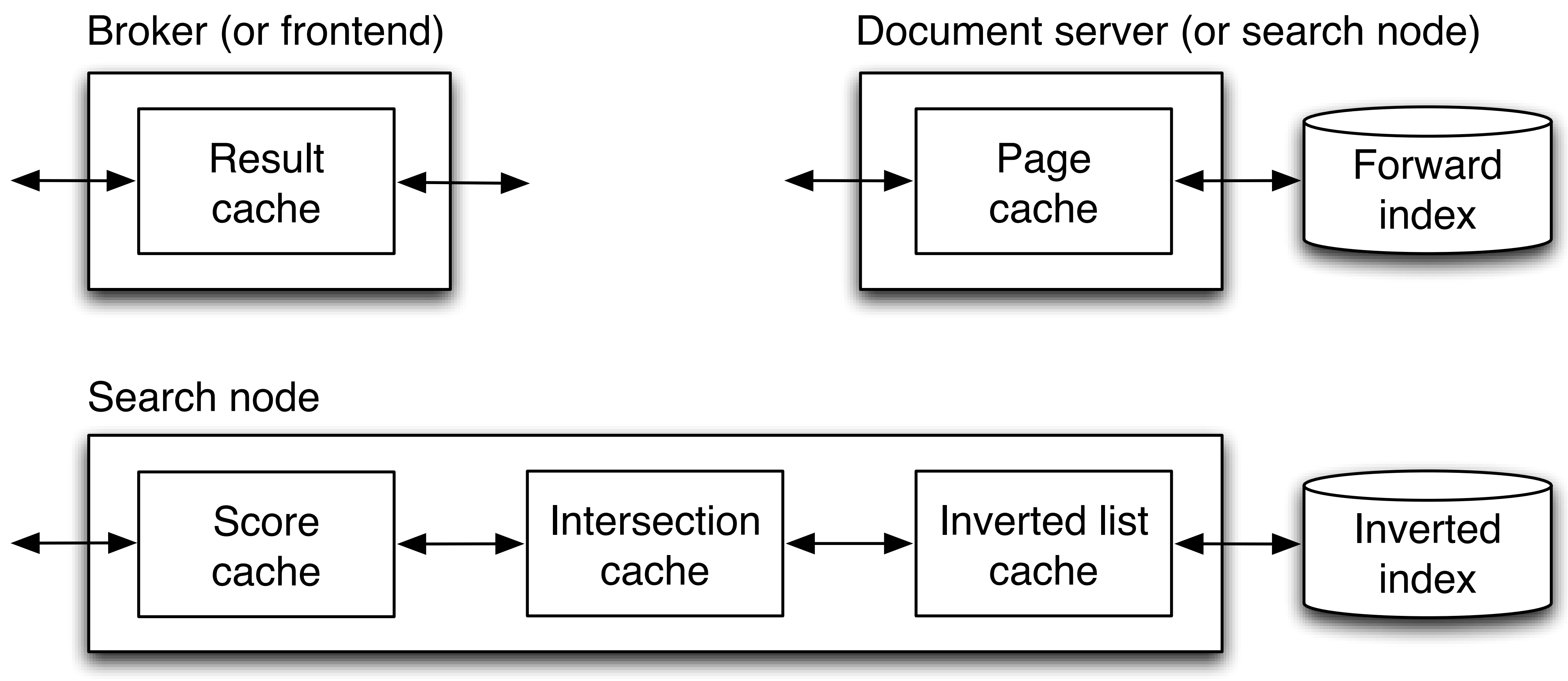
# Selective Search

- Documents are clustered and a separate index is built
  - similarity between documents
  - co-click likelihood
- A query is processed on the indexes associated with the most similar *n* clusters
- Reduces the workload, but suffers from the load imbalance problem
  - query topic distribution may be skewed
  - certain indexes have to be queried much more often



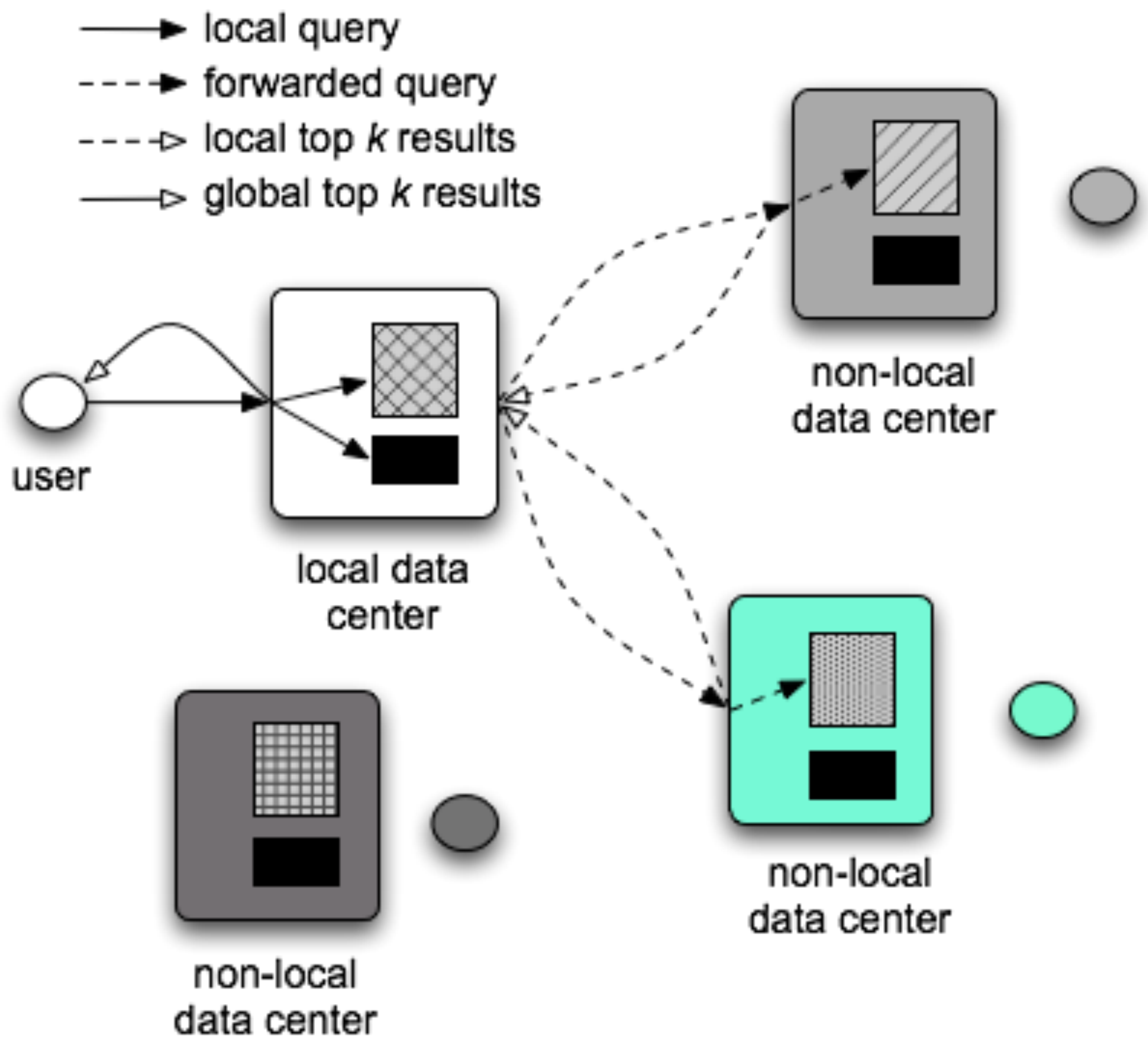
## Caching

Main caches in search engines: result cache, score cache, intersection cache, inverted list cache, page cache



# Partitioned Search Architectures

- Key points
  - multiple, regional data centers (sites)
  - user-to-center assignment
  - partitioned web index
  - partial document replication
- Enables
  - local web crawling
  - query processing with selective forwarding



## Open Source Search Engines

- ATIRE: Search engine (BSD license)
- DataparkSearch: Website search engine (GNU GPL)
- Elasticsearch: Search server based on Lucene (Apache license 2.0)
- Galago: Search engine toolkit (BSD license)
- Indri: Search engine (BSD-inspired license)
- **Lucene**: Search engine library (Apache license 2.0)
- **MG4J**: Distributed search engine (GNU LGPL)
- mnoGoSearch: Website search engine (GNU GPL)
- **Solr**: Distributed search engine based on Lucene (Apache license 2.0)
- Seeks: User-centric search engine (Affero GPLv3)
- Sphinx: Search engine (GNU GPLv2)
- **Terrier**: Search engine (Mozilla public license)
- Wumpus: Desktop search engine (GNU GPL)
- Zettair: Search engine (BSD license)

# Open Source Search Engines

- ATIRE: Search engine (BSD license)
- DataparkSearch: Website search engine (GNU GPL)
- Elasticsearch: Search server based on Lucene (Apache license 2.0)
- Galago: Search engine toolkit (BSD license)
- Indri: Search engine (BSD-inspired license)
- Lucene: Search engine library (Apache license 2.0)
- MG4J: Distributed search engine (GNU LGPL)
- mnoGoSearch: Website search engine (GNU GPL)
- Solr: Distributed search engine based on Lucene (Apache license 2.0)
- Seeks: User-centric search engine (Affero GPLv3)
- Sphinx: Search engine (GNU GPLv2)
- Terrier: Search engine (Mozilla public license)
- Wumpus: Desktop search engine (GNU GPL)
- Zettair: Search engine (BSD license)