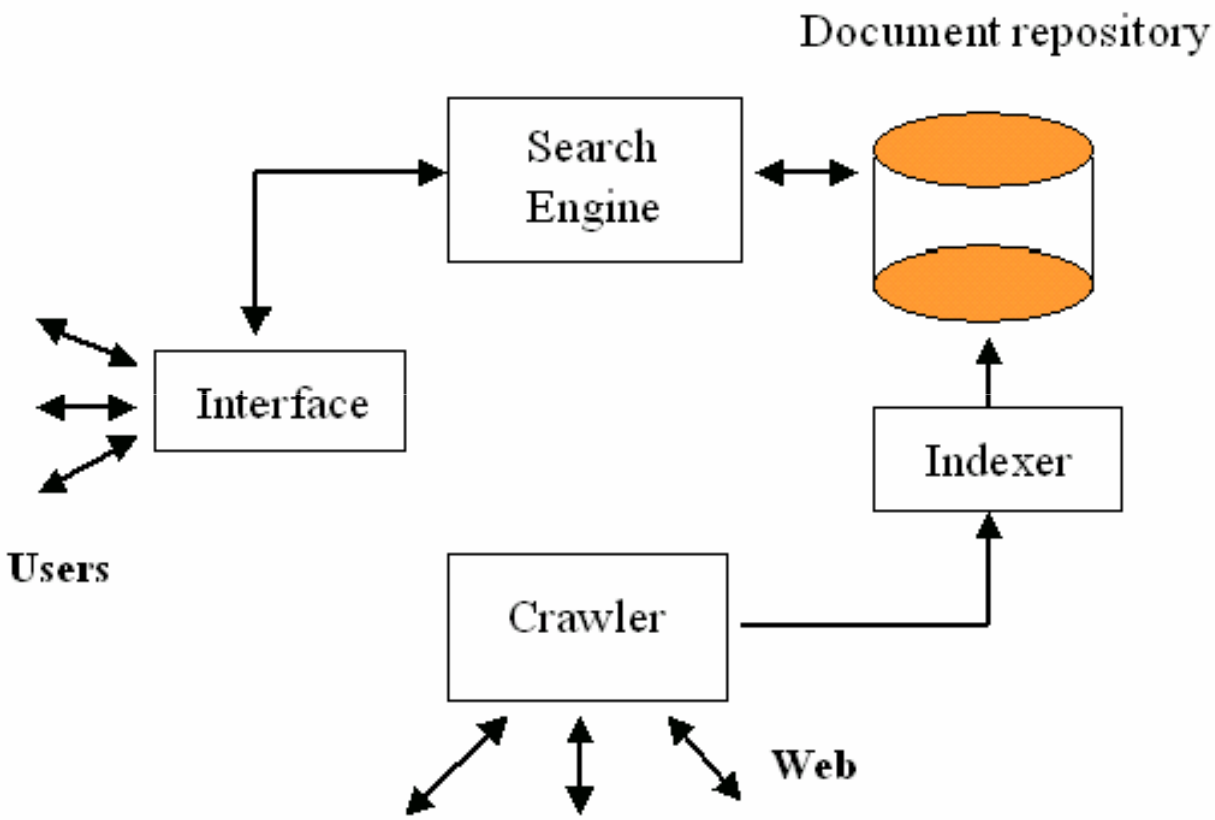
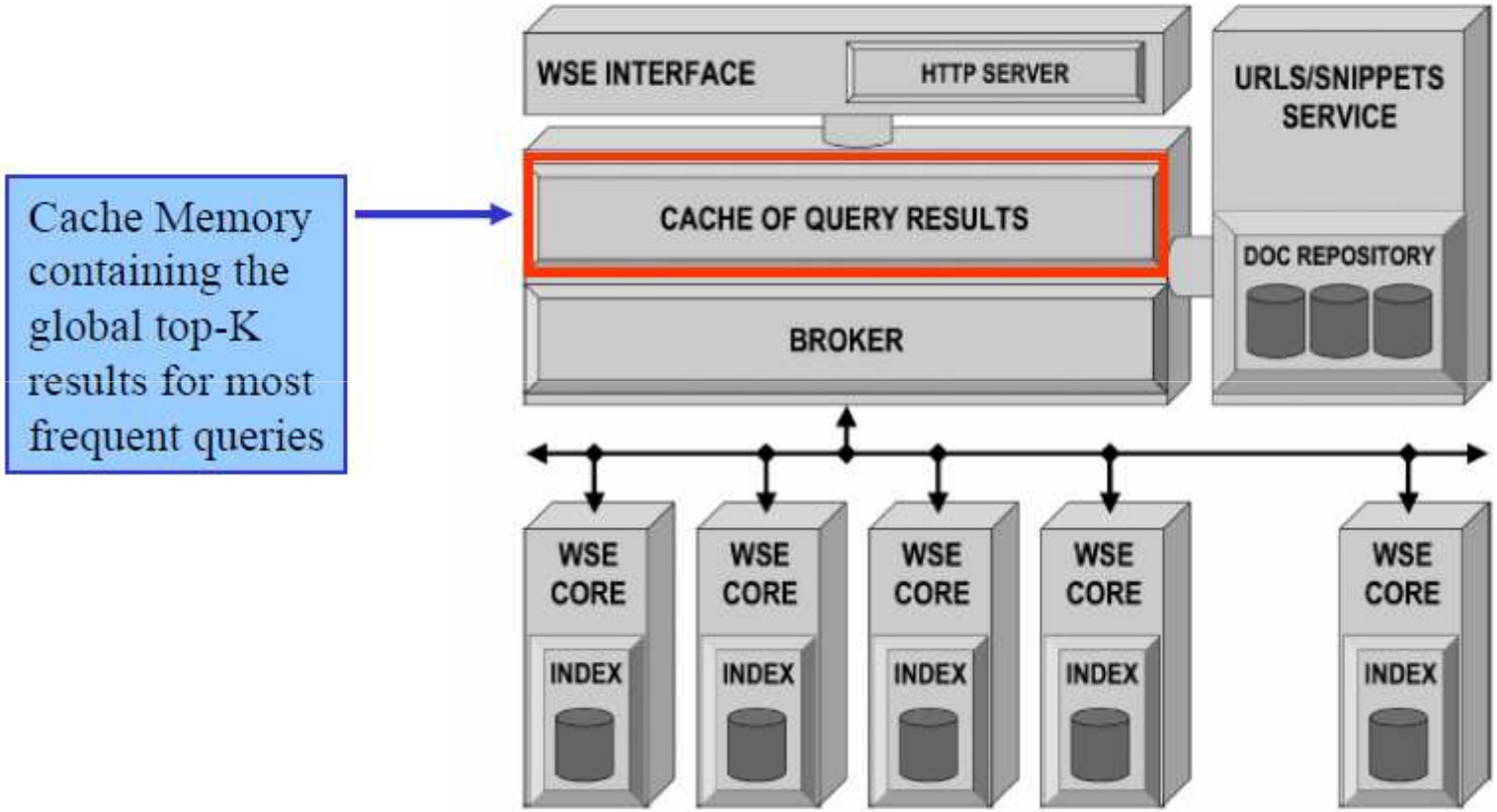
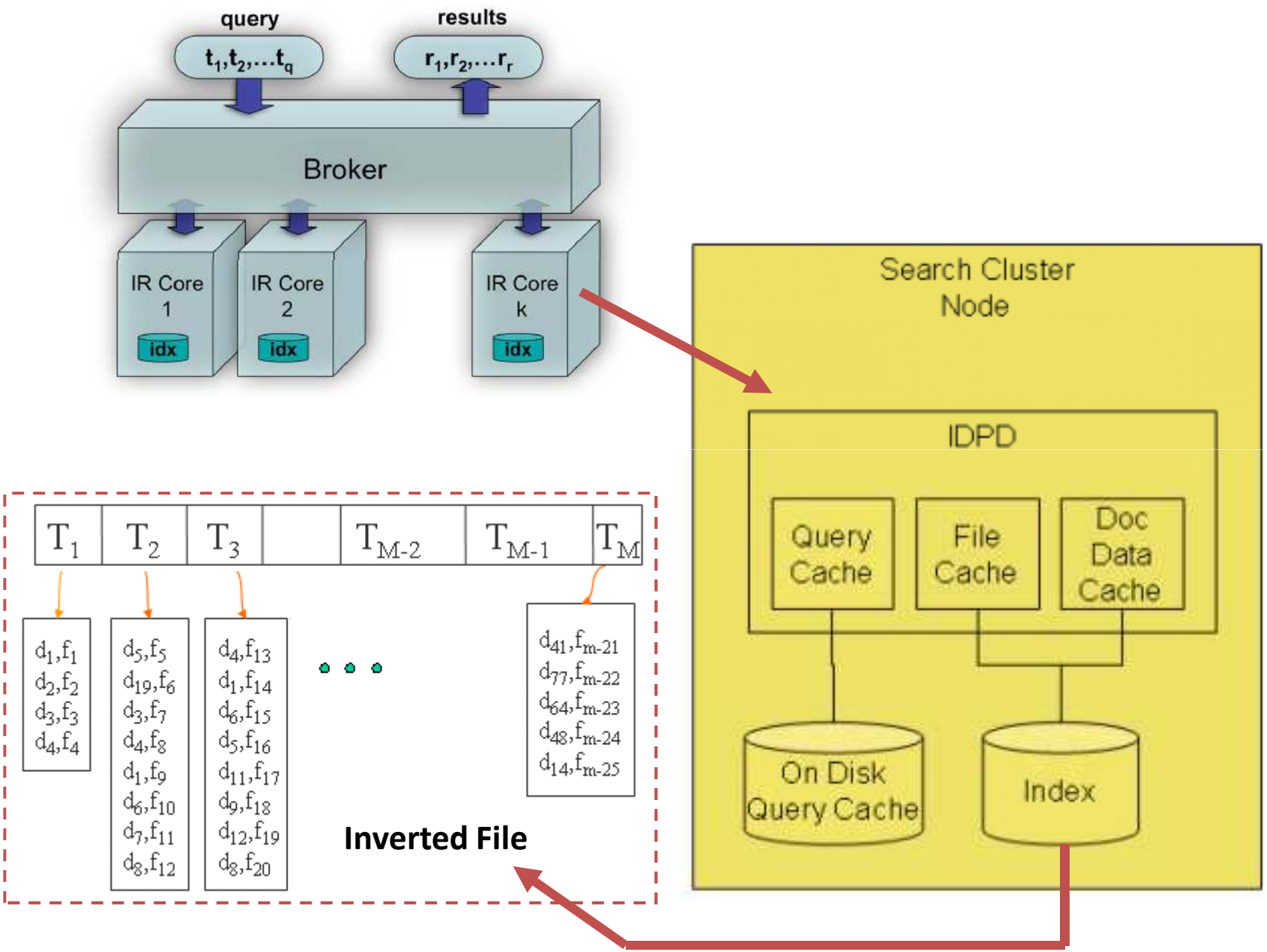
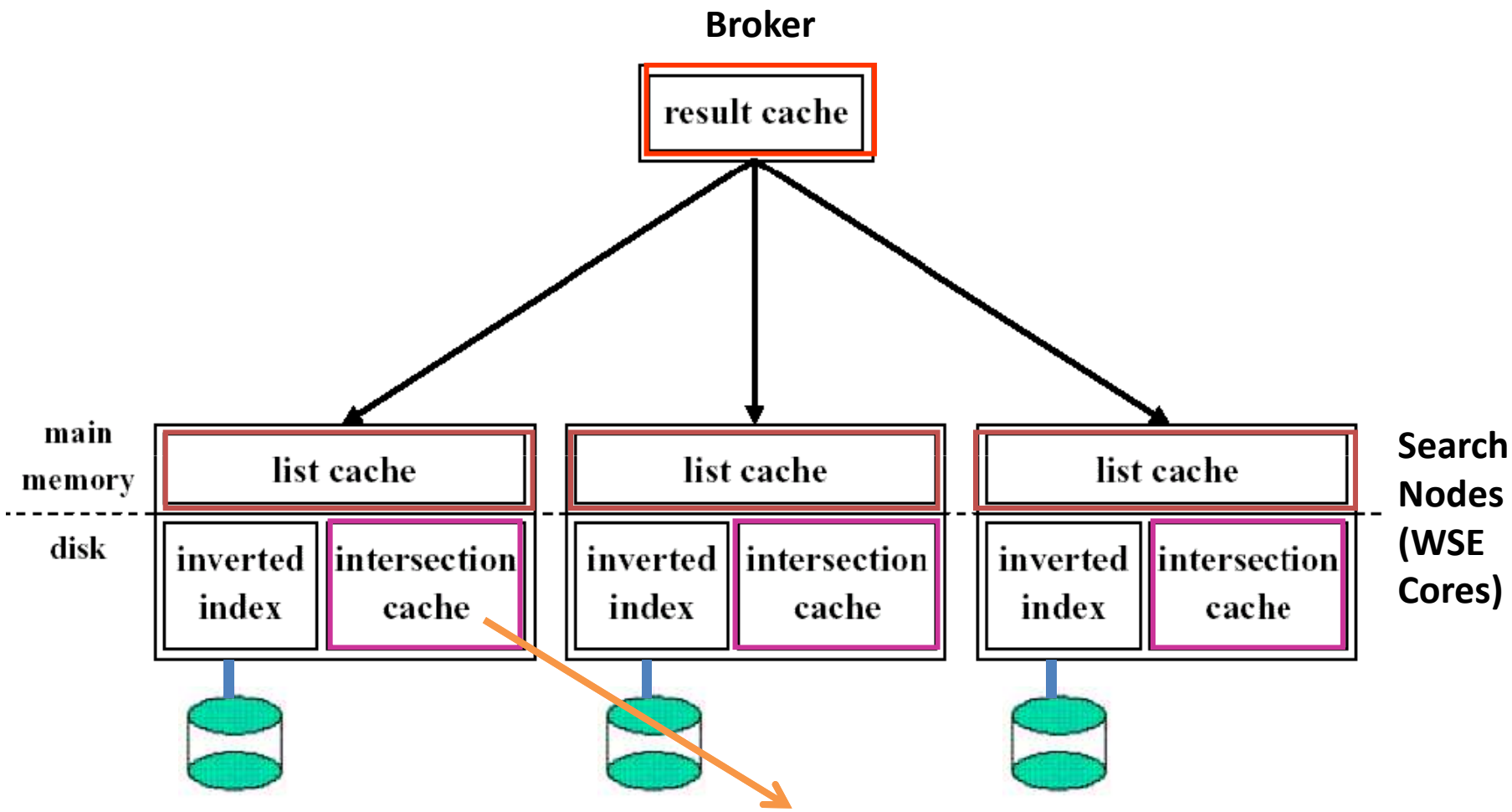
Jerarquía de Caches sobre Motores de Búsqueda Web

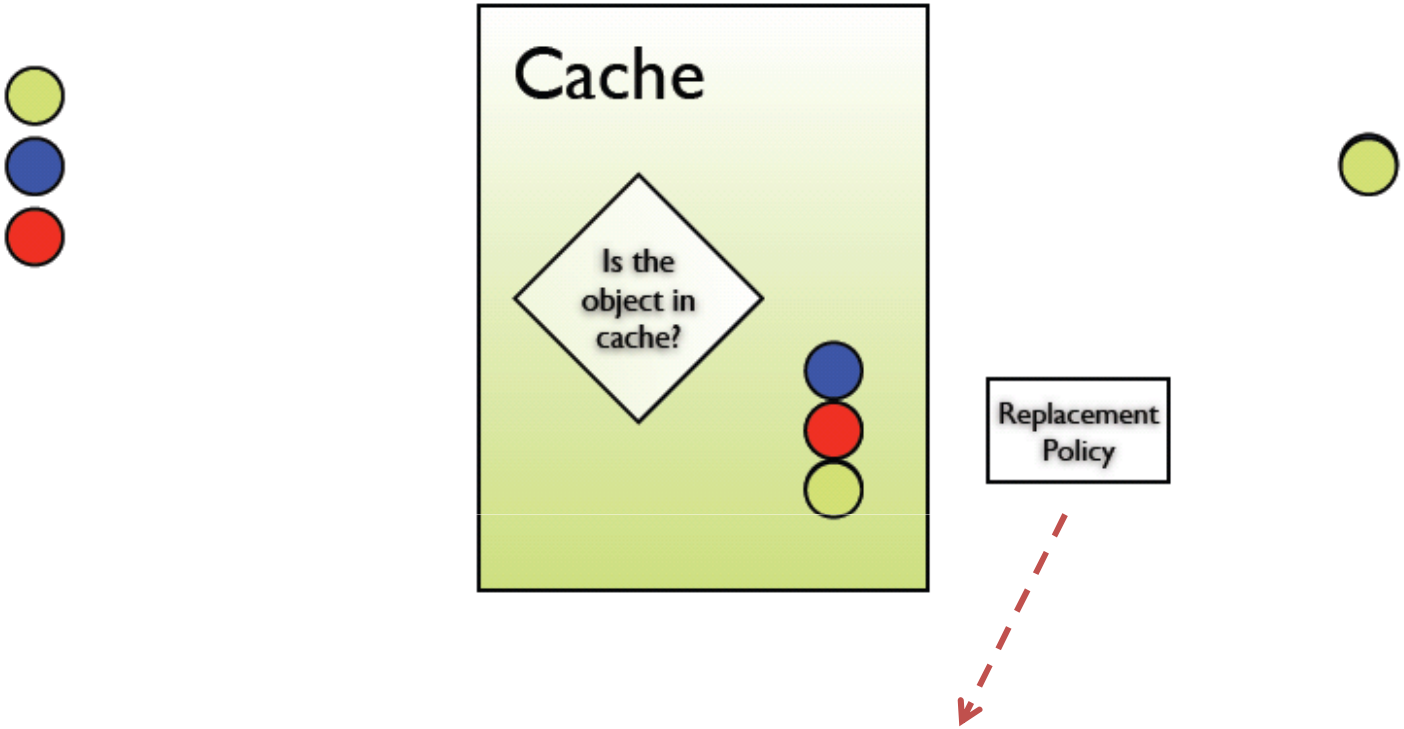

Document repository
Search Engine
Interface
Users
Indexer
Crawler
Web


Cache Memory containing the global top-K results for most frequent queries
WSE INTERFACE
HTTP SERVER
URLS/SNIPPETS SERVICE
CACHE OF QUERY RESULTS
DOC REPOSITORY
BROKER
WSE CORE
INDEX
WSE CORE
INDEX
WSE CORE
INDEX
WSE CORE
INDEX
WSE CORE
INDEX


query
$t_1, t_2, \ldots t_q$
results
$r_1, r_2, \ldots r_r$
Broker
IR Core 1
idx
IR Core 2
idx
IR Core k
idx
Search Cluster Node
IDPD
Query Cache
File Cache
Doc Data Cache
On Disk Query Cache
Index
T_1 T_2 T_3 T_{M-2} T_{M-1} T_M
d_1,f_1 d_2,f_2 d_3,f_3 d_4,f_4
d_5,f_5 d_{19},f_6 d_3,f_7 d_4,f_8 d_1,f_9 d_6,f_{10} d_7,f_{11} d_8,f_{12}
d_4,f_{13} d_1,f_{14} d_6,f_{15} d_5,f_{16} d_{11},f_{17} d_9,f_{18} d_{12},f_{19} d_8,f_{20}
d_{41},f_{m-21} d_{77},f_{m-22} d_{64},f_{m-23} d_{48},f_{m-24} d_{14},f_{m-25}
Inverted File



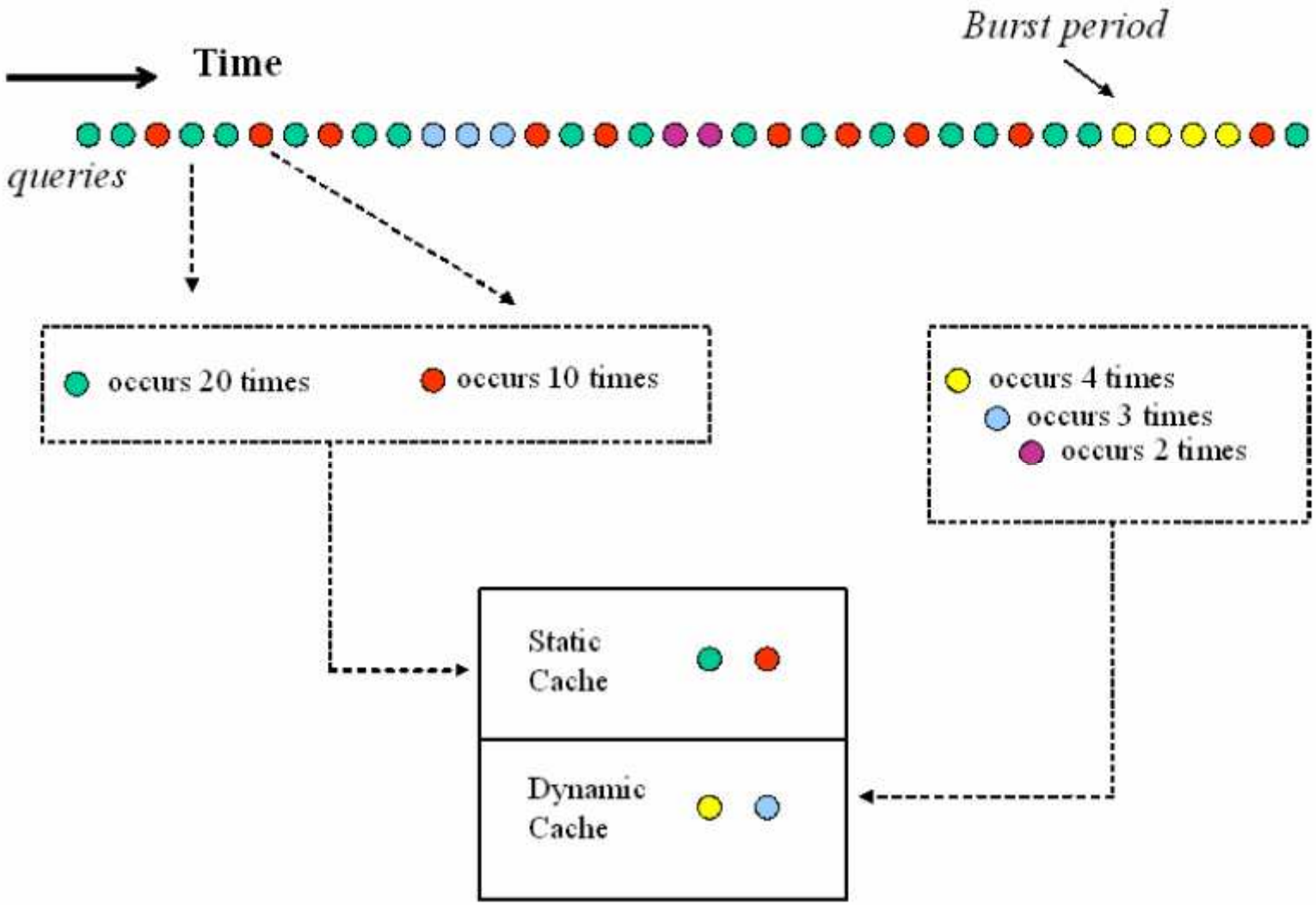
Projection Lists: Ia→b and Ib→a that share the same document IDs in the intersection set, but keep data from a and b respectively that are used to score the documents.



LFU = Least Frequently Used

LRU = Least Recently Used

SDC = LFU (80%) + LRU (20%)


Burst period
Time
queries
occurs 20 times
occurs 10 times
occurs 4 times
occurs 3 times
occurs 2 times
Static Cache
Dynamic Cache

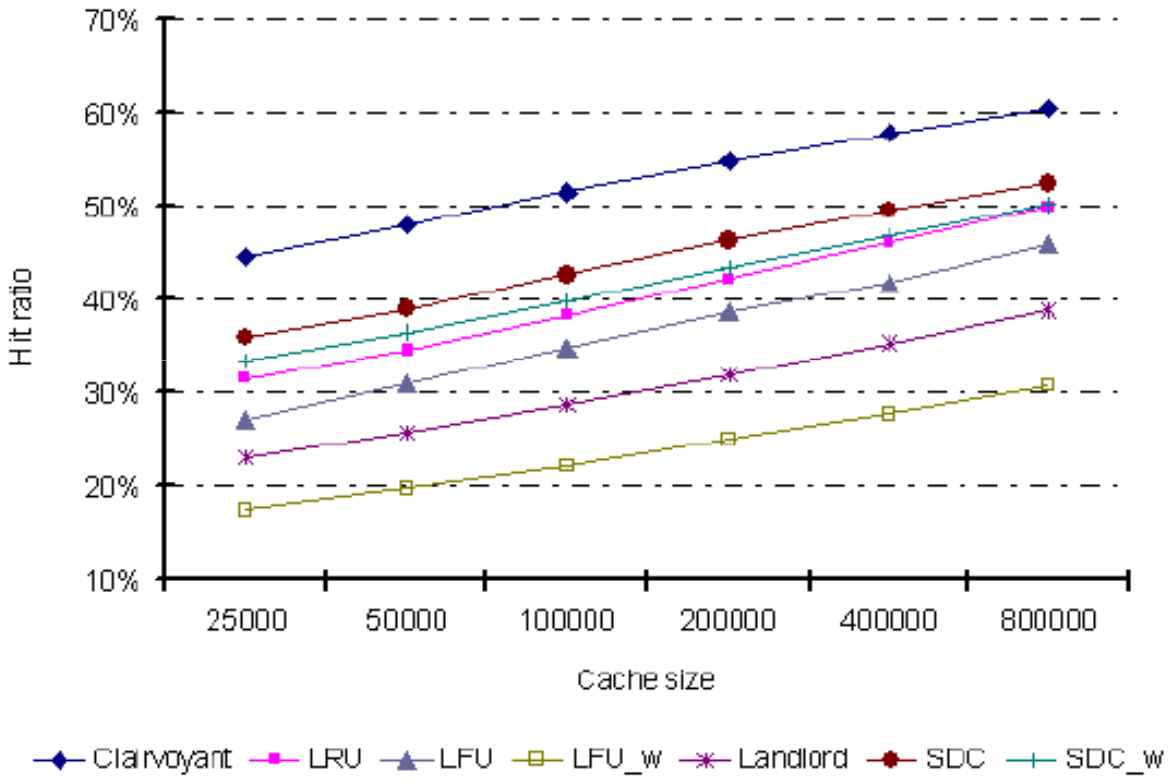
RCACHE

SDC = LFU (80%) + LRU (20%)

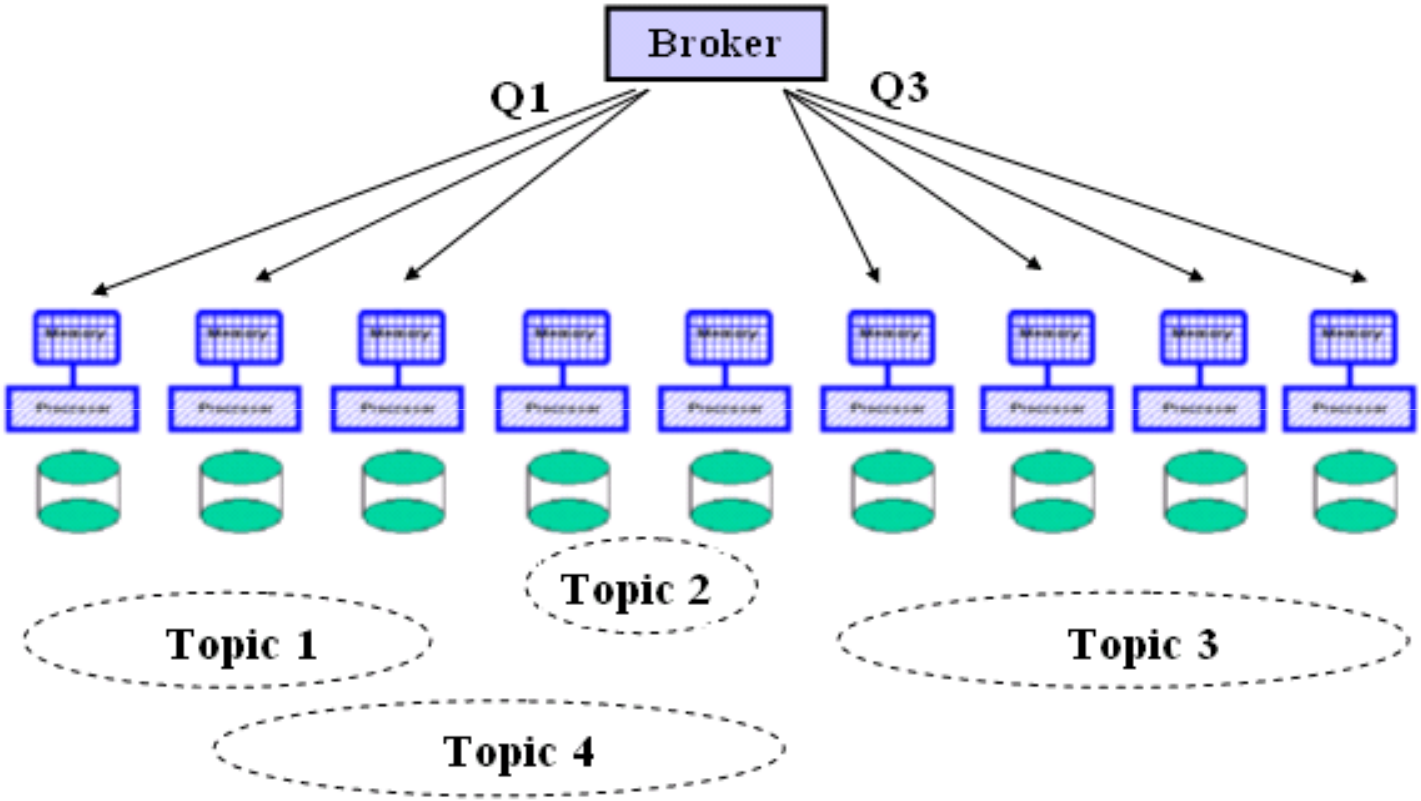
LFU-RCache we use cost =

$$\text{frequency} \cdot \log(L) \cdot n$$

where L is the sum of the lengths of the posting lists associated with the query terms and n the number of processors used to get the global top-R results for the query.

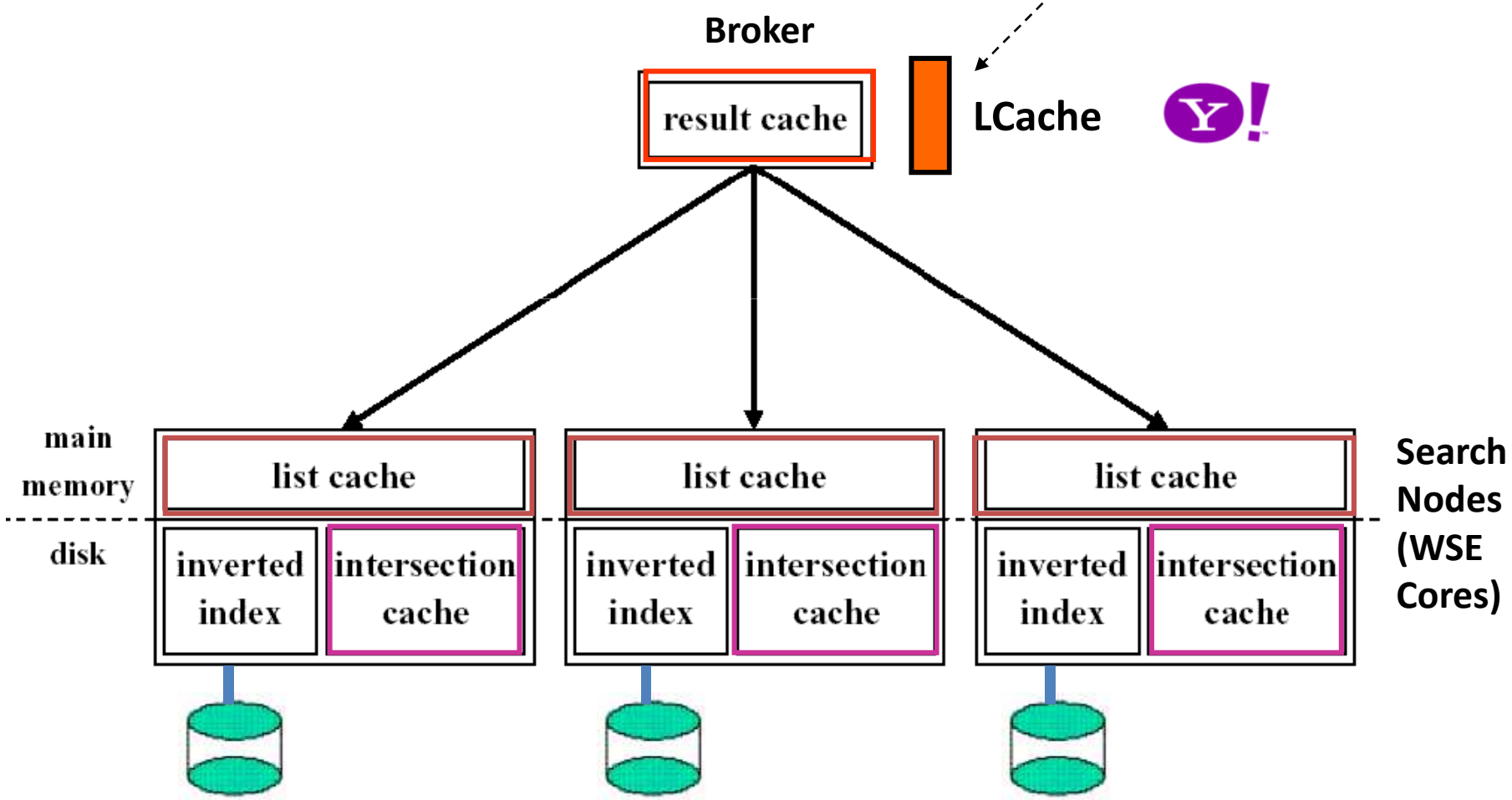

70%
60%
50%
40%
30%
20%
10%
Hit ratio
25000
50000
100000
200000
400000
800000
Cache size
Clairvoyant
LRU
LFU
LFU_w
Landlord
SDC
SDC_w

Throughput can be further improved by preventing the broker from sending each query to all of the search nodes

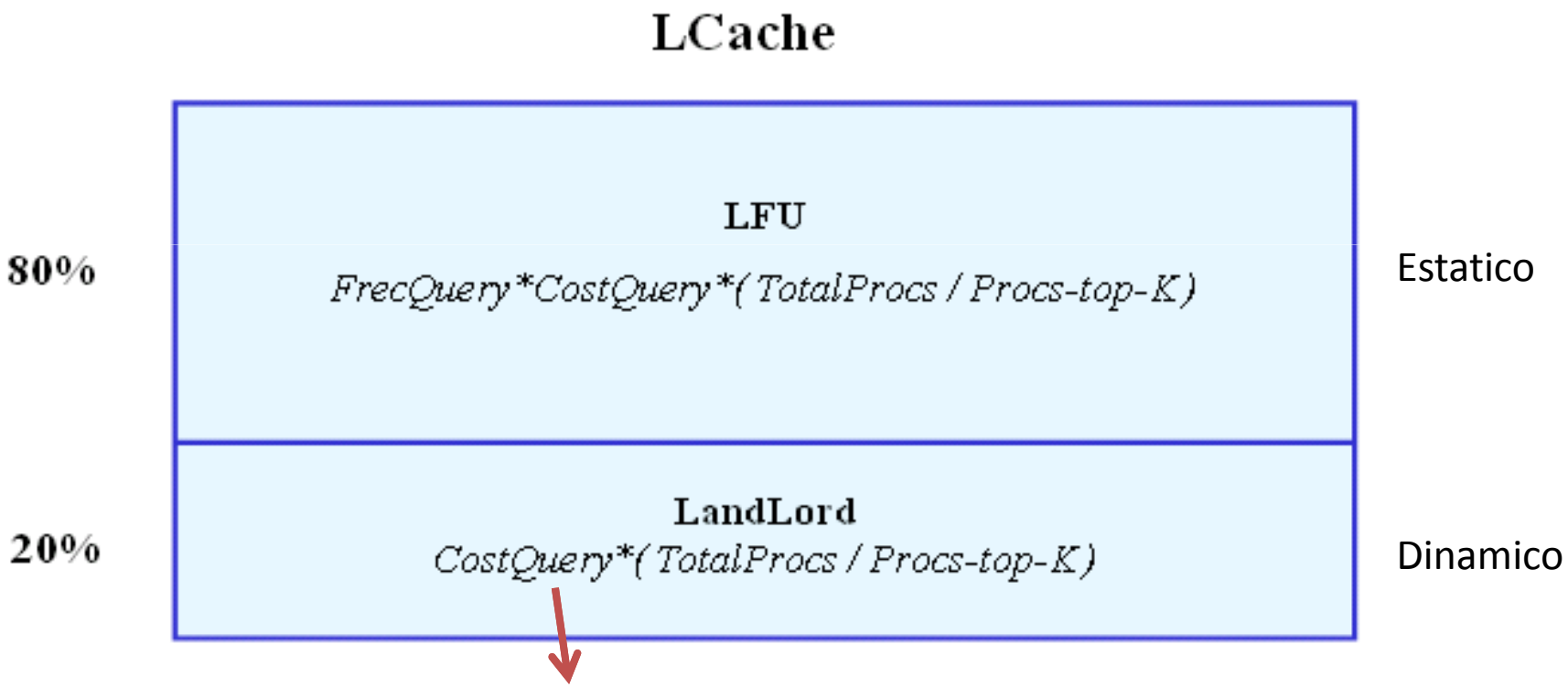


The total number of processors able to produce results among the global top-K ones can be reduced by performing document clustering.

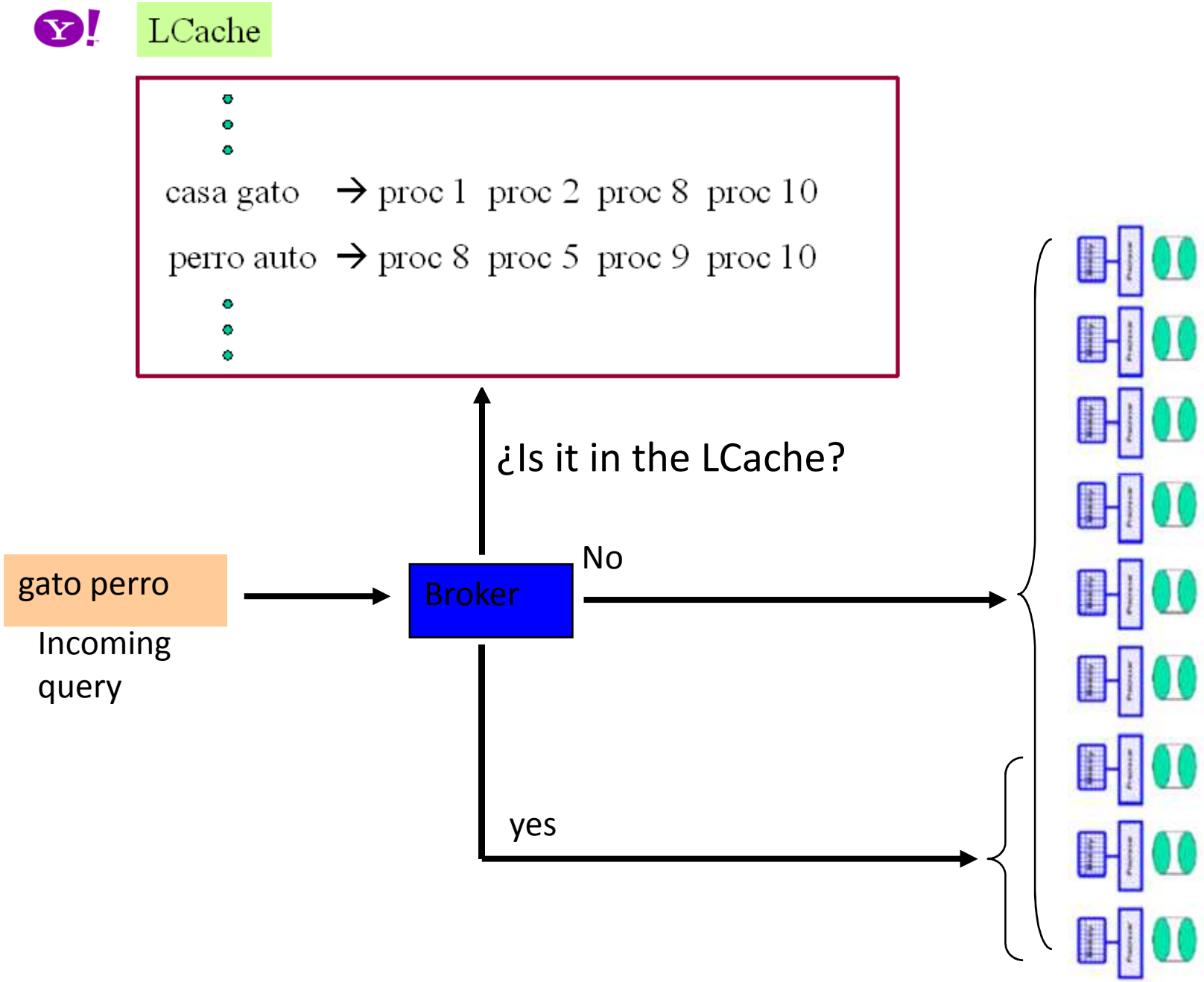
Proposal: keep a tiny cache indicating processor Ids producing the top-K results for frequent queries.

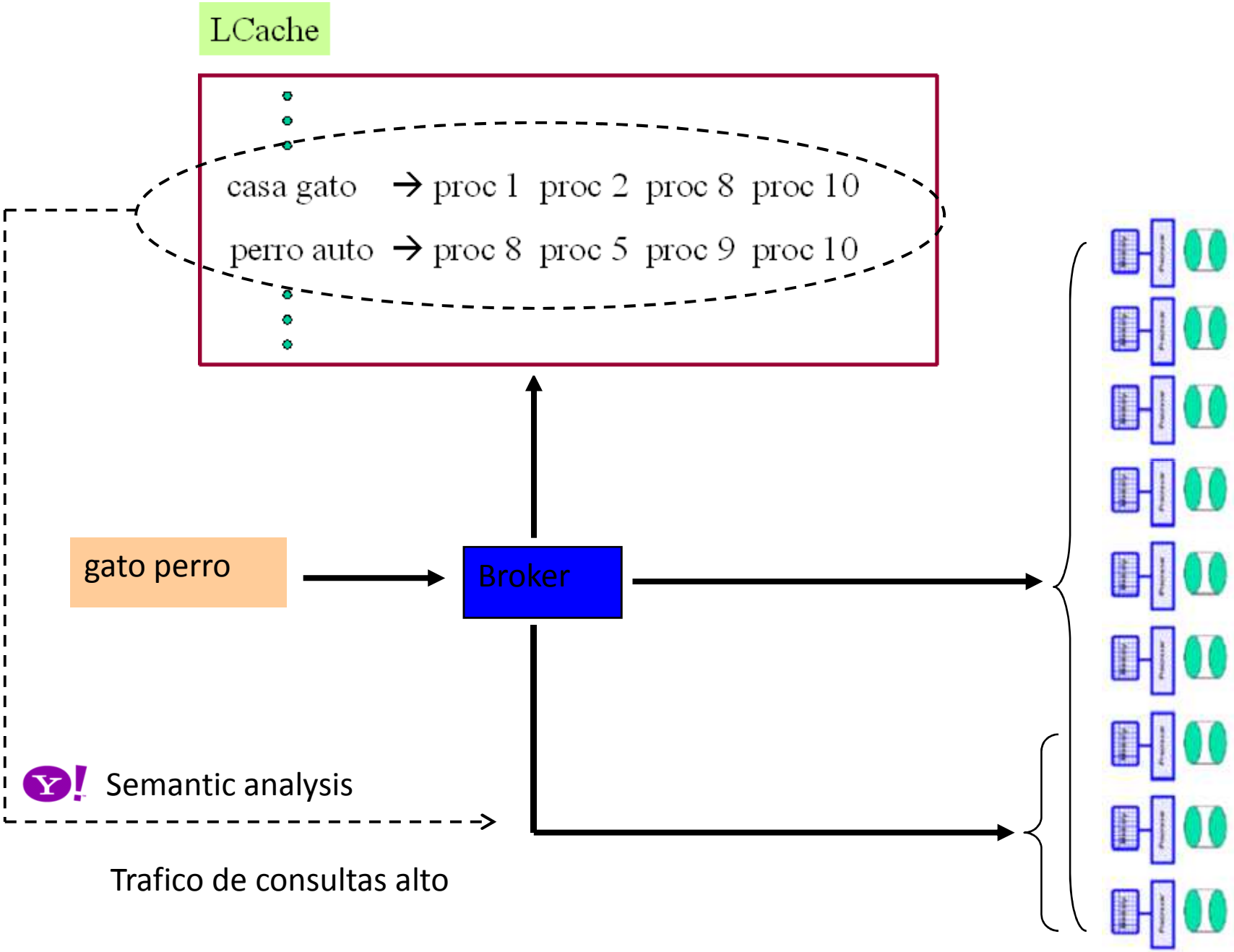


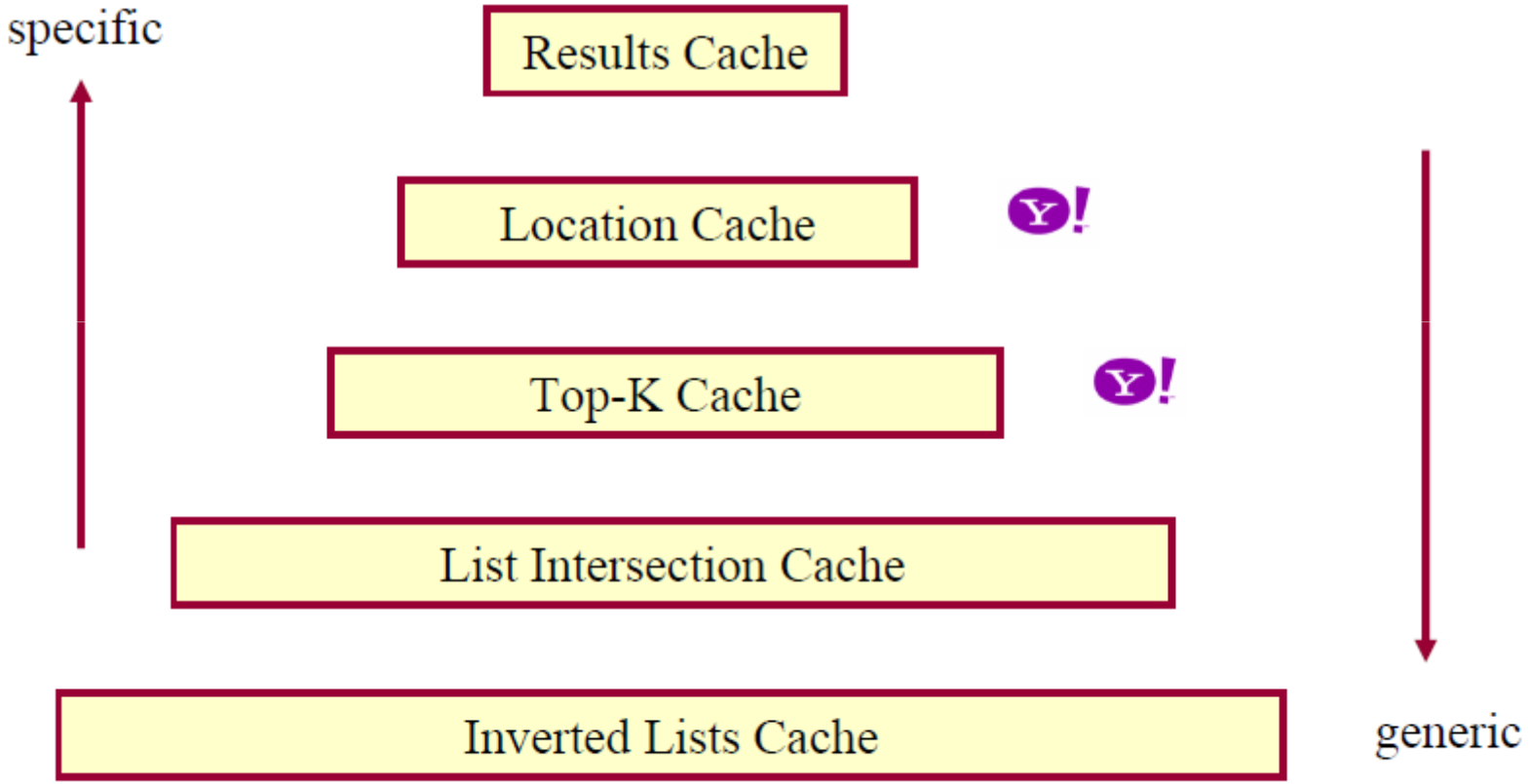
The cache policy grants more priority to queries which are costly to solve and whose top-K results come from a few processors.

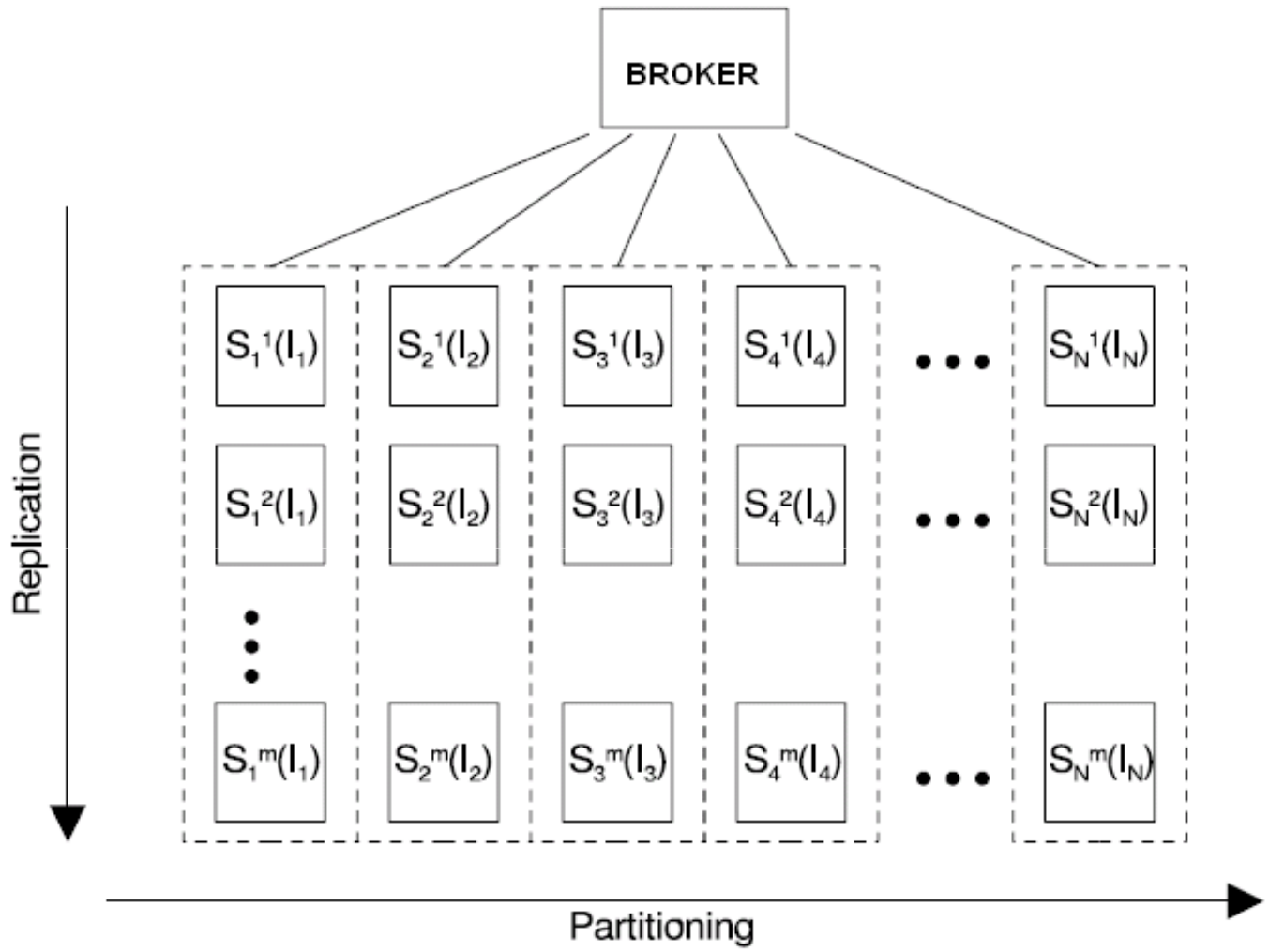


frequency · log(L) · (P/n) where L is the sum of the lengths of the posting lists.
Gives higher priority to the frequent queries that require fewer processors

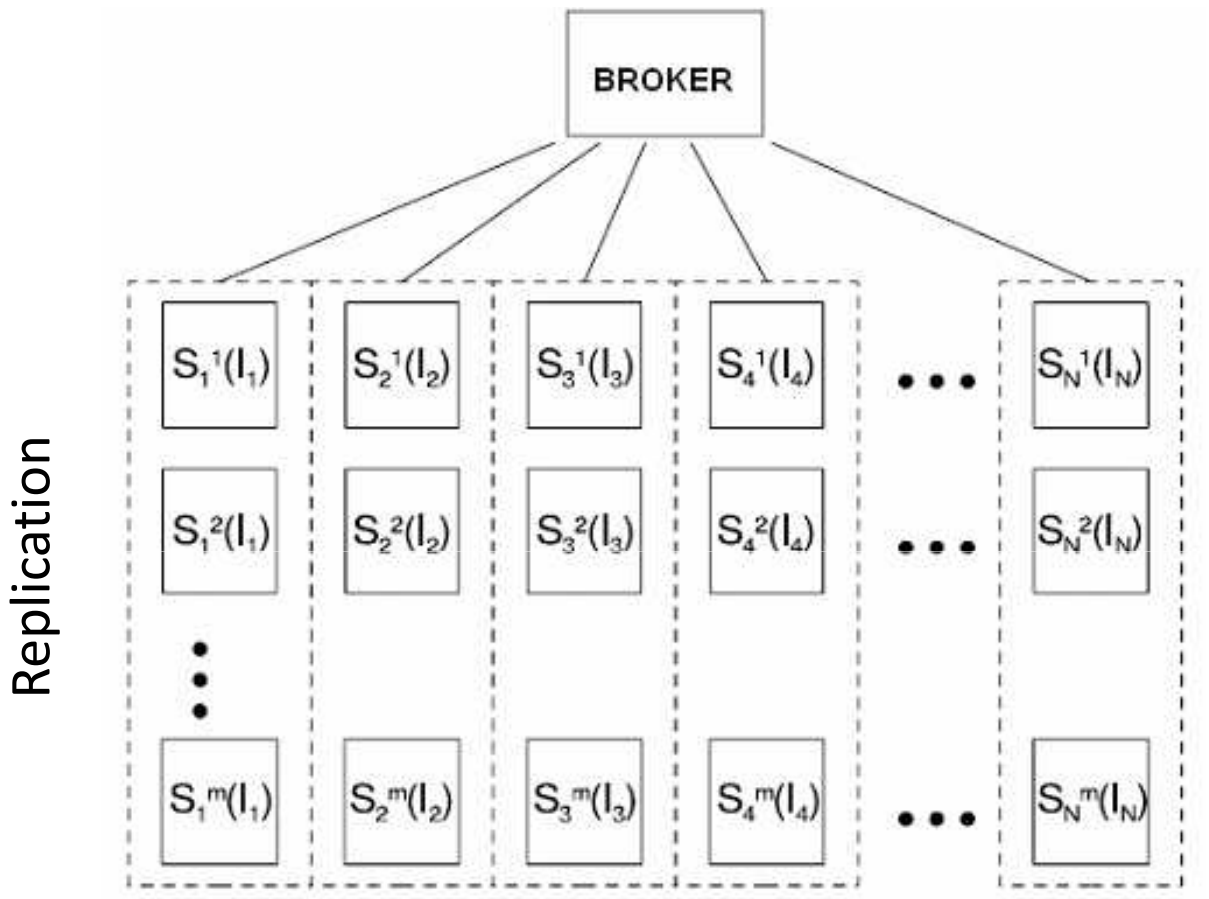

LCache
casa gato → proc 1 proc 2 proc 8 proc 10
perro auto → proc 8 proc 5 proc 9 proc 10
¿Is it in the LCache?
gato perro
Incoming query
Broker
No
yes


LCache
casa gato → proc 1 proc 2 proc 8 proc 10
perro auto → proc 8 proc 5 proc 9 proc 10
gato perro
Broker
Semantic analysis
Trafico de consultas alto


specific
Results Cache
Location Cache
Top-K Cache
List Intersection Cache
Inverted Lists Cache
generic



Cluster overview

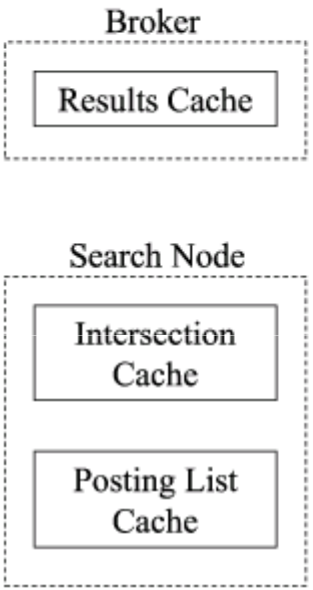


Term Partitioned Inverted File

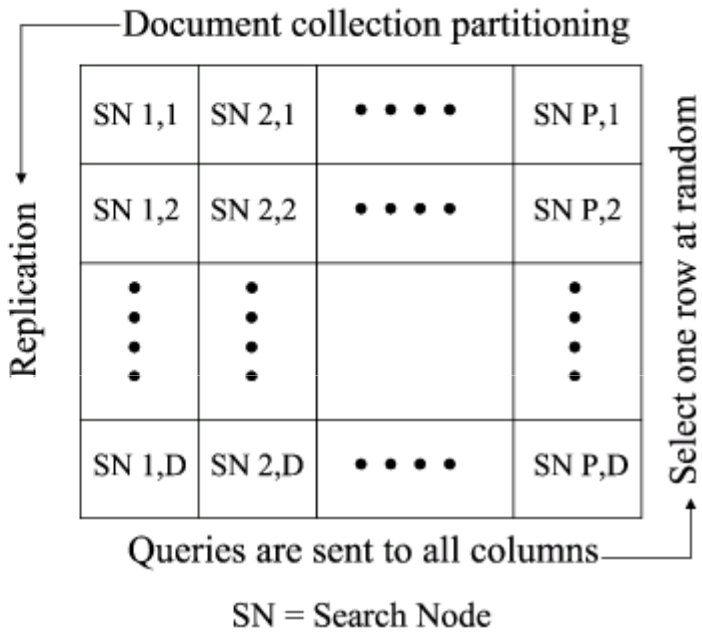
(main Memory)

Document Partitioned Inverted File

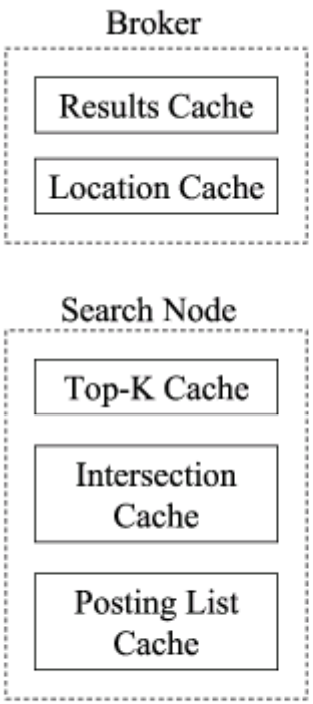
P columns (partitions)
D rows (replicas)



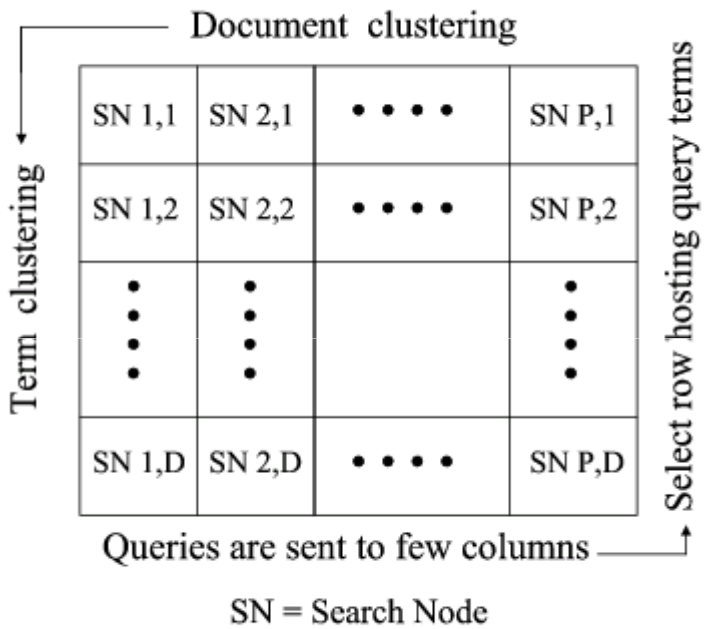
(a) Standard cache hierarchy



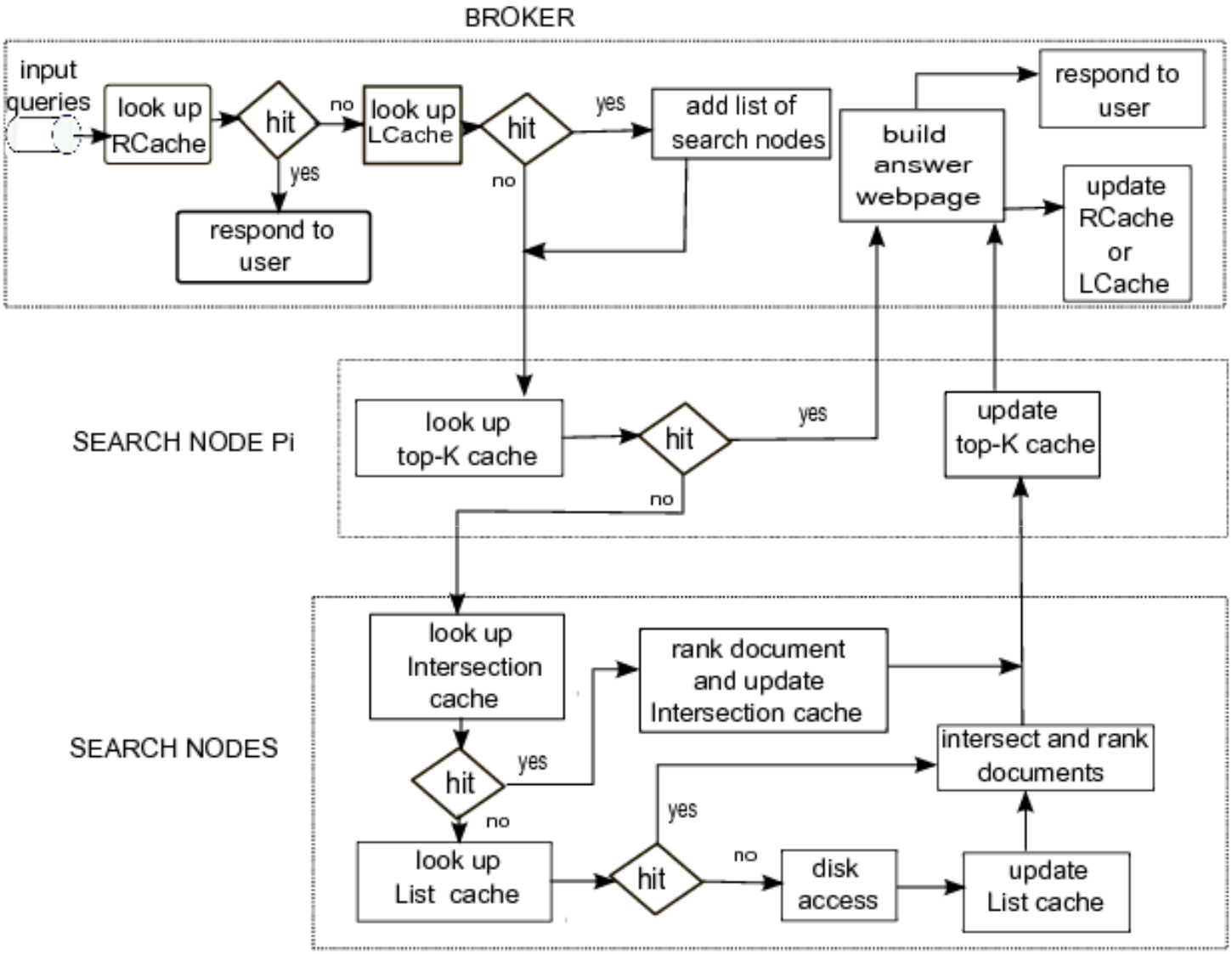
(b) Standard query routing on search nodes



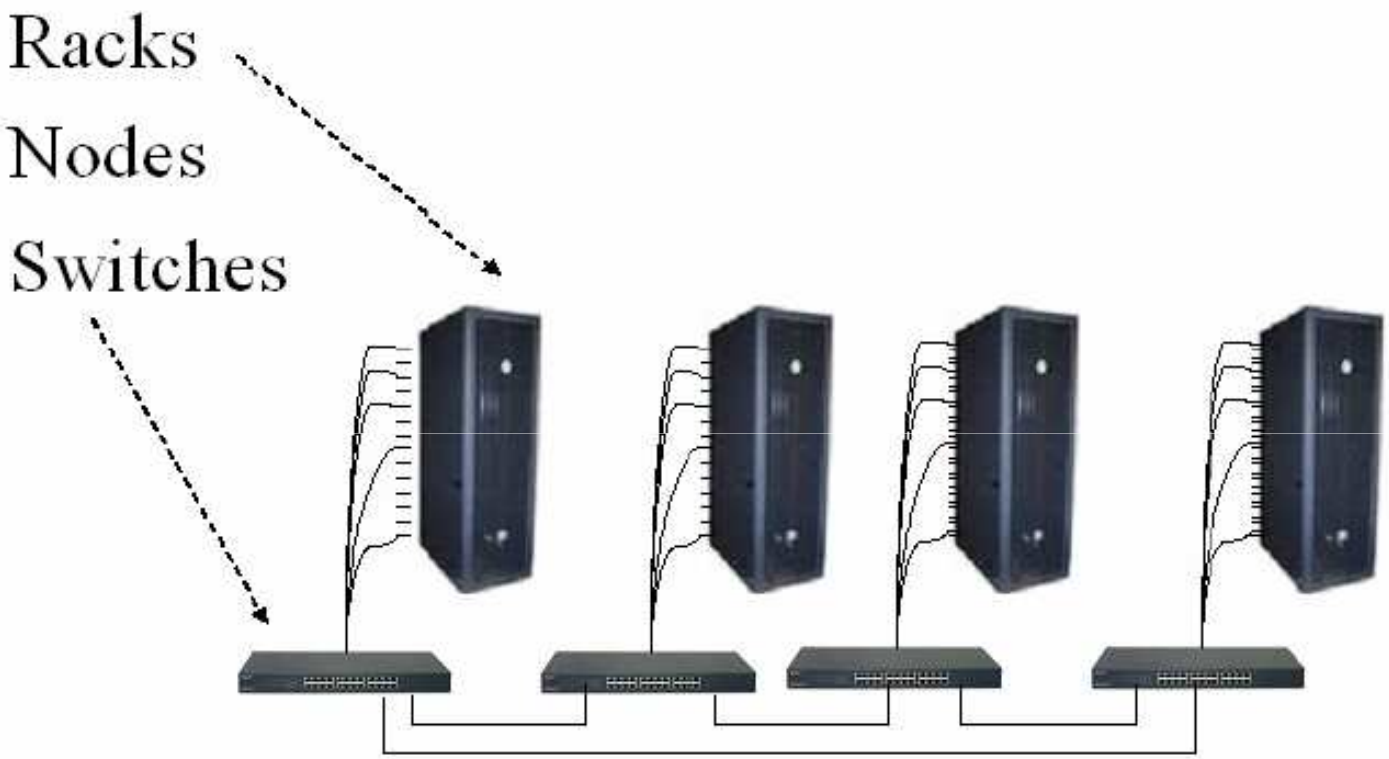
(a) Proposed cache hierarchy



(b) Proposed query routing on search nodes



Flow of queries across the proposed cache hierarchy

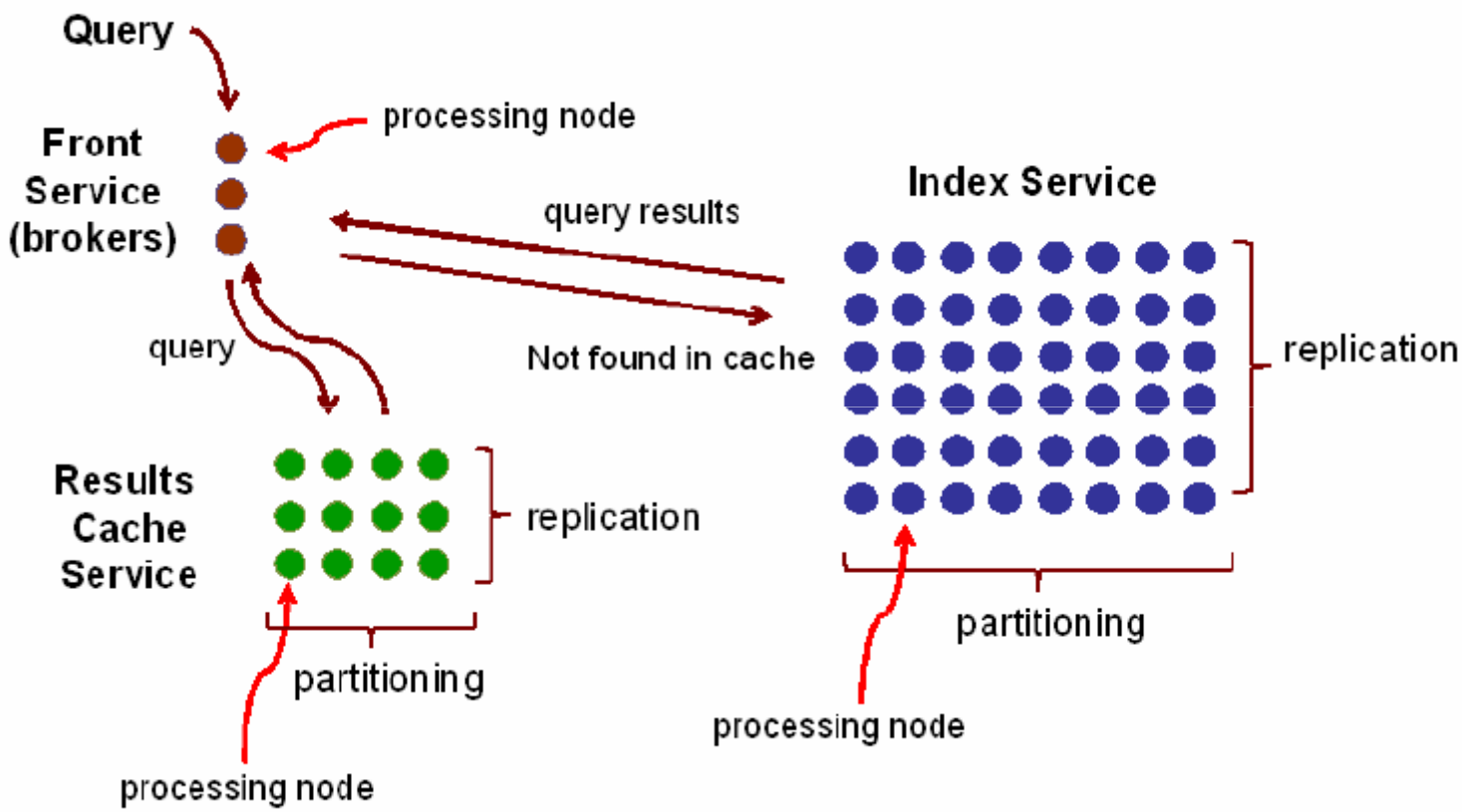

Racks
Nodes
Switches

40 processing nodes

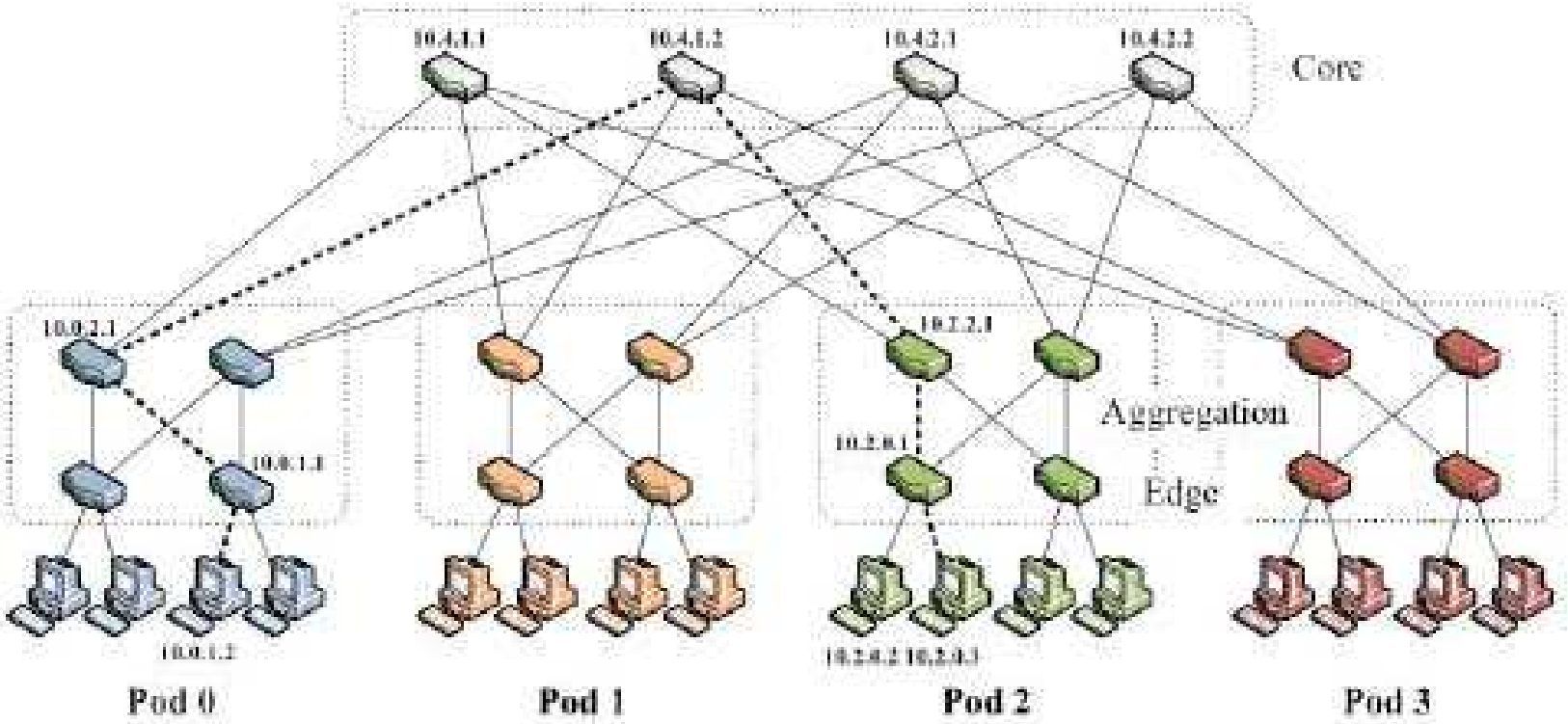
Services

- FS: Front server
- CS: Caching service
- IS: Index Service

Search engines are divided into a collection of services.
Each service is deployed in a set of processing nodes.



A key goal is optimizing throughput but making sure that query response time is below a given upper bound.



Simple fat-tree topology. Using the two-level routing tables packets from source 10.0.1.2 to destination 10.2.0.3 would take the dashed path.

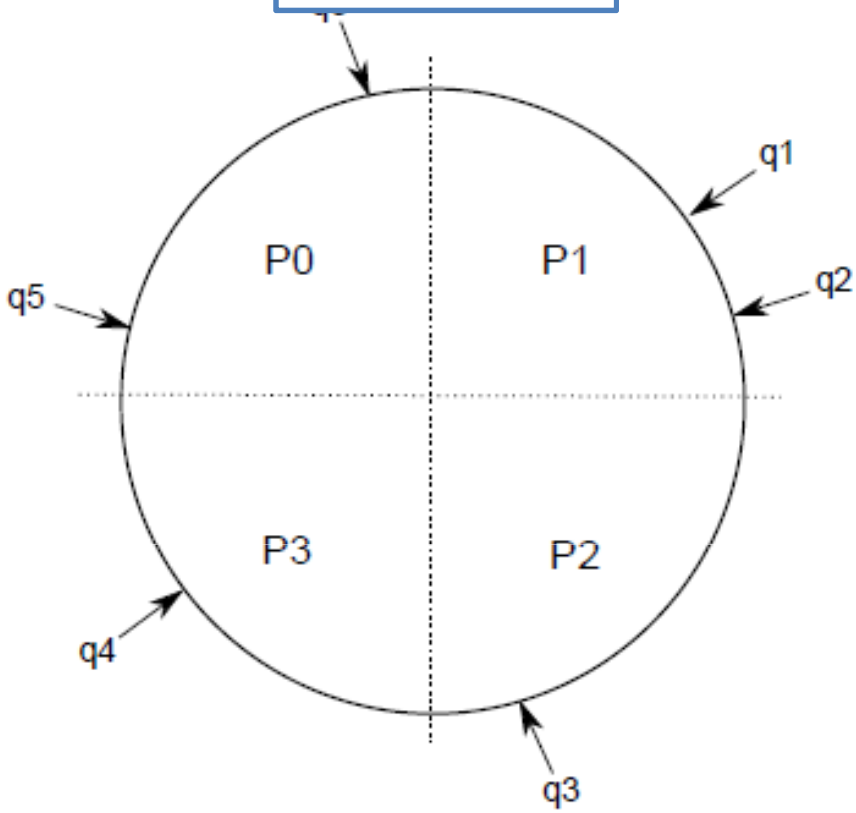
MEMCACHED

Free & open source, high-performance,
distributed memory object caching system

Simple Key/Value Store
The server does not care what your data looks like. Items are made up of a key, an expiration time, optional flags, and raw data.
It does not understand data structures; you must upload data that is pre-serialized.

http://memcached.org/

Baseline



(b) Consistent Hash

There are four operations performed by the CS nodes: (a) search for a query, (b) insert a query, (c) update a query priority and (d) delete a cache entry

Consistencia: enviar un mensaje a cada nodo replica de la particion.
NO ESCALABLE

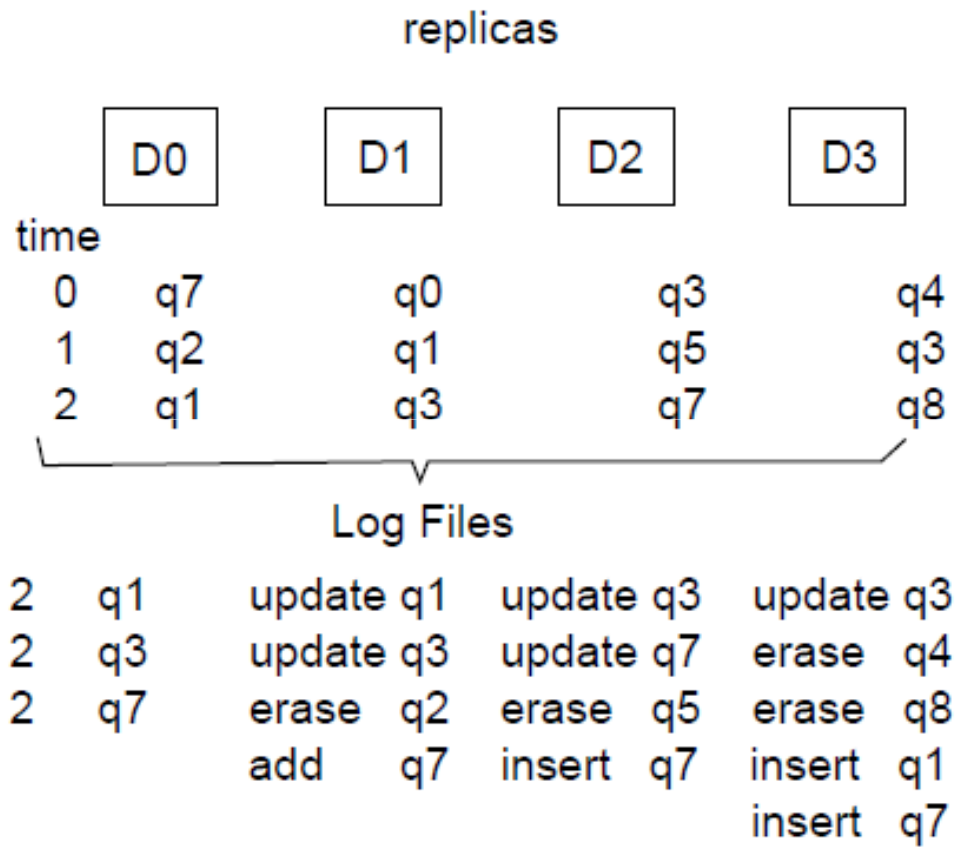


Fig. 2. Optimistic LRU cache strategy.

when a partition A is selected by the FS for a query q, we apply a second hash function over the query terms to select one replica from A

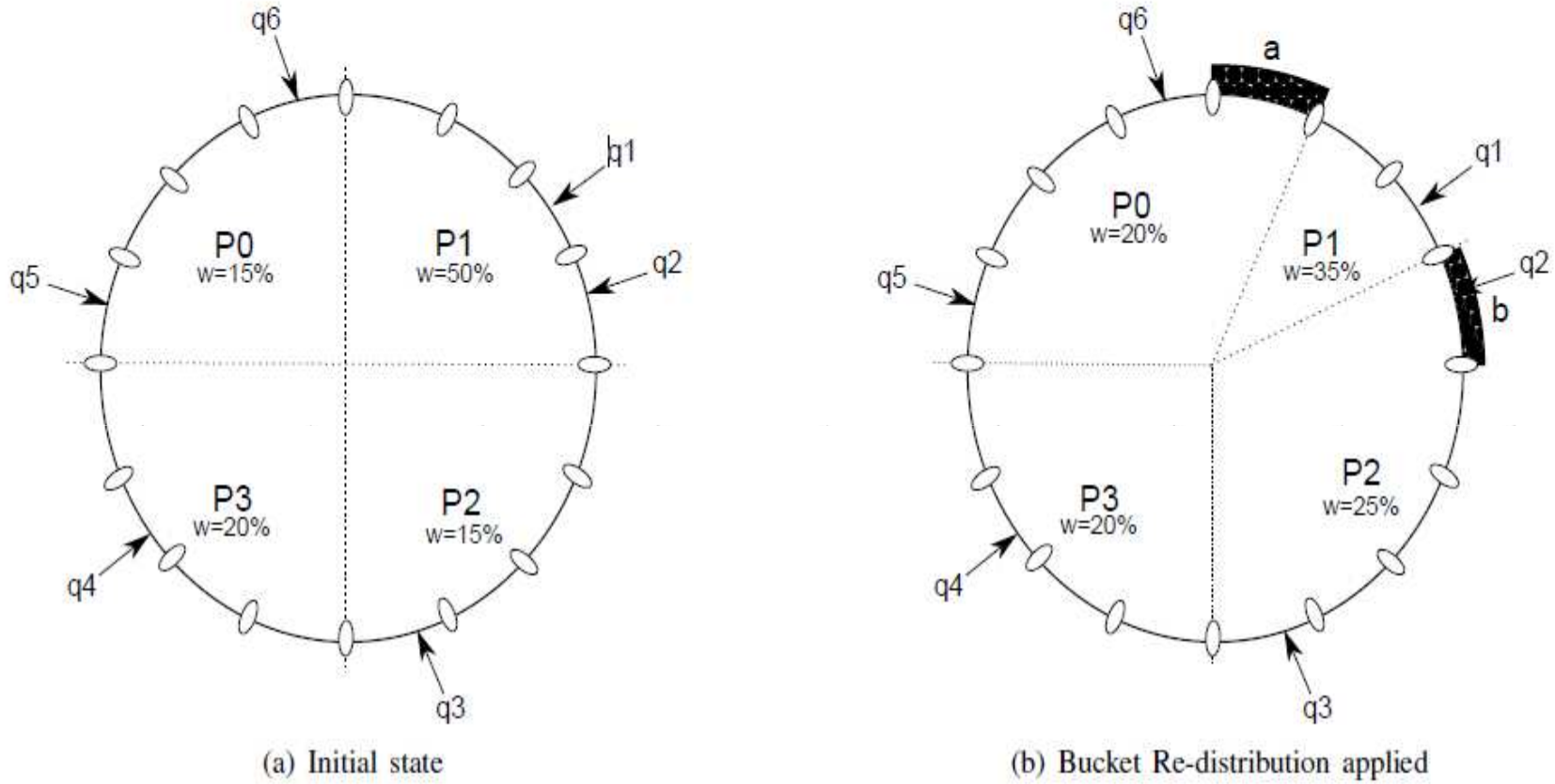


Fig. 3. Consistent Hash Ring

we balance the CS partitions workload by increasing/decreasing the range of each partition on the fly and according to their utilization

TOLERANCIA A FALLOS

RADIC is based on rollback-recovery techniques applying a pessimistic event-log approach. It is based on two kinds of components: *Protectors and Observers.*

Each CS node has a Protector and an Observer

1. Every X units of time all Observers send their checkpoint to the corresponding Protector.

2. If node m belonging to partition P fails, all requirements send to m are re-directed to its Protector allocated at the same partition P.

3. The Protector of m process its own queries and queries of m

Resultados Aproximados

Ranking del query cluster con BM25

Document cluster

Query cluster

PCAP	dc1	dc2	dc3	dc4	dc5	$r_q(qc_i)$
qc1		0.5	0.8	0.1		0.2
qc2	0.3		0.2		0.1	0.8
qc3	0.1	0.5	0.8			0

$$
\begin{aligned}
r_q(dc_1) &= 0 + 0.3 \times 0.8 + 0 = 0.24 \\
r_q(dc_2) &= 0.5 \times 0.2 + 0 + 0 = 0.10 \\
r_q(dc_3) &= 0.8 \times 0.2 + 0.2 \times 0.8 + 0 = 0.32 \\
r_q(dc_4) &= 0.1 \times 0.2 + 0 + 0 = 0.02 \\
r_q(dc_5) &= 0 + 0.1 \times 0.8 + 0 = 0.08
\end{aligned}
$$

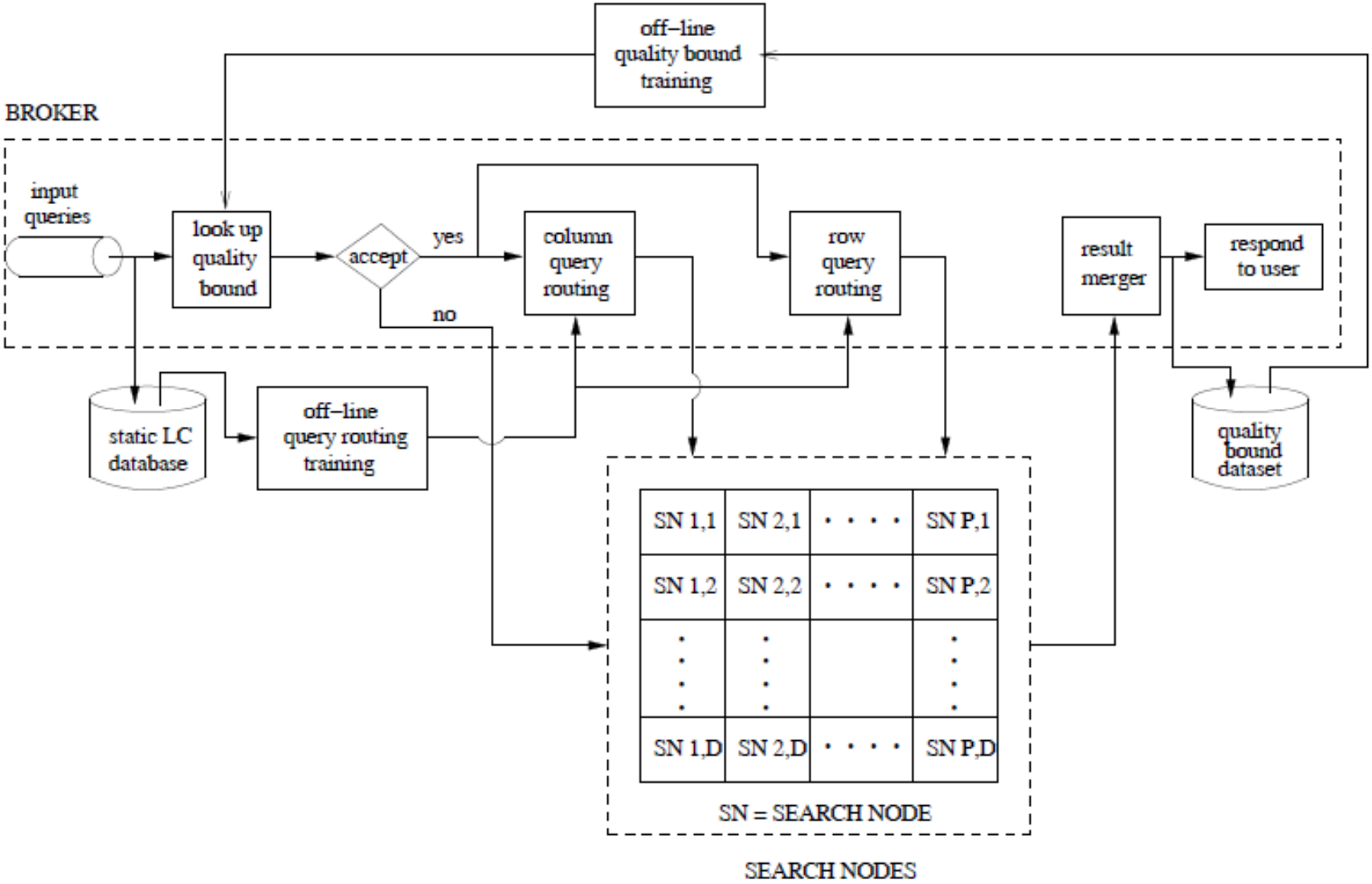


Figure 4: Flow of queries across the proposed query routing strategy.