Gestión Eficiente de Transacciones en Máquinas de Búsqueda para la Web

Carolina Bonacic Castro

Directores: Manuel Prieto Matías, Carlos García Sánchez
Mauricio Marín.

Agenda

- Contexto general del problema
- Servicio de índice
- Planteamiento del problema
- Solución propuesta
- Comparación con estrategias alternativas.
- Conclusiones

Contexto General del Problema

Rendimiento eficiente

Estrategias: indexación, caching y procesamiento paralelo de consultas

Un thread una consulta

Clusters de procesadores de memoria distribuida

Contexto General del Problema

Las máquinas de búsqueda han comenzado a permitir la actualización de sus contenidos de manera online.

Problema de sincronización de threads lectores y escritores que acceden concurrentemente a las estructura de datos en memoria compartida

Contexto General del Problema

- **Objetivos**: diseñar, analizar, implementar y evaluar estrategias síncronas de procesamiento de transacciones y compararlas con sus contrapartes asíncronas.

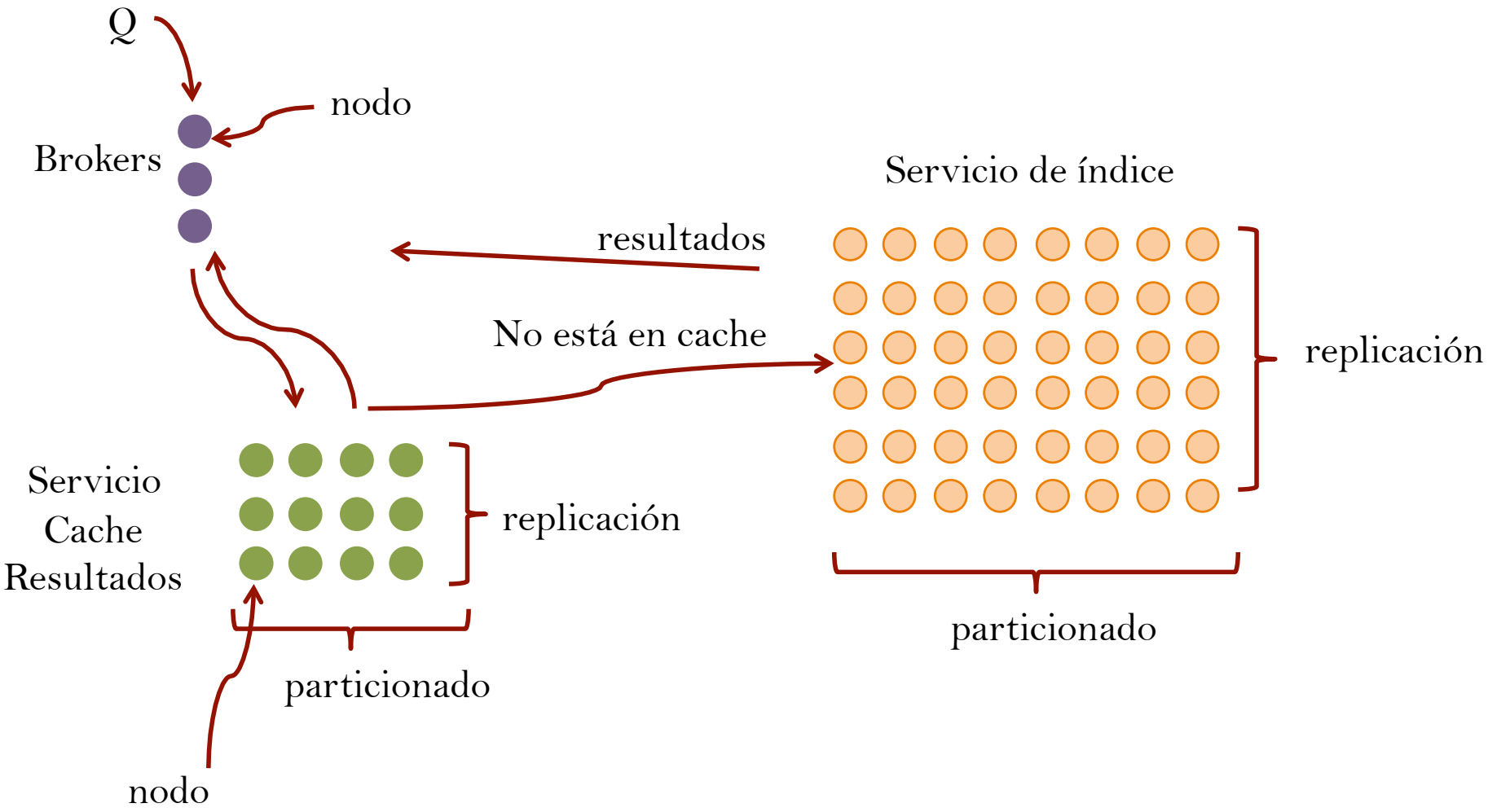
- **Hipótesis:** paralelismo síncrono permite alcanzar mejor rendimiento.

- **Métricas de interés**:

  Throughput: cantidad de transacciones resueltas por unidad de tiempo.

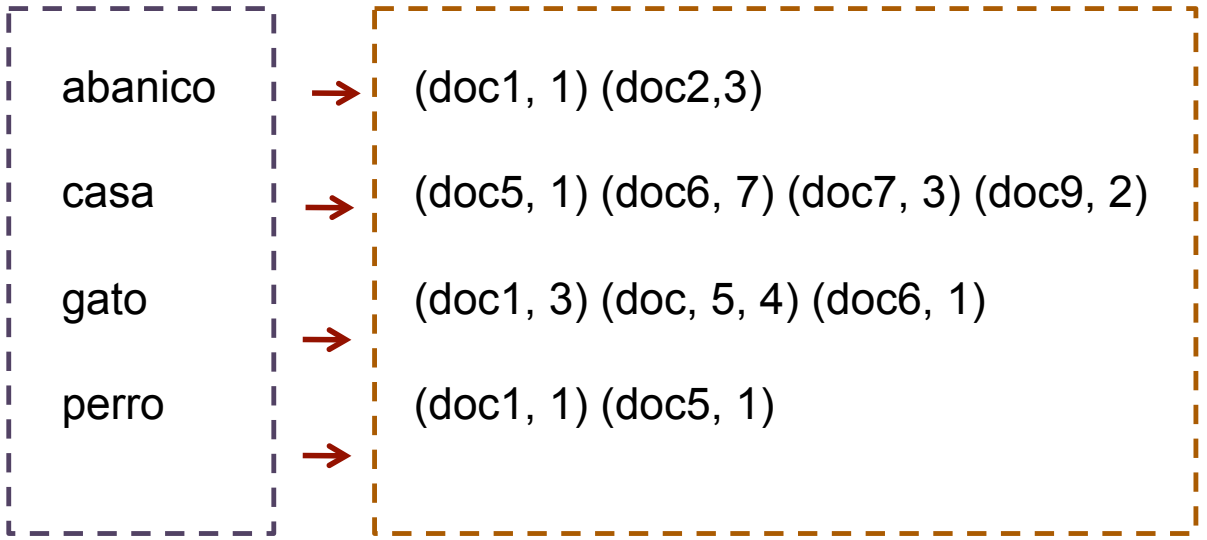
  Calidad de Servicio: garantizar un tiempo de respuesta **individual** menor a una cota superior.

Servicio de Índice.
Arquitectura de un Motor Web



Servicio de Índice.
Índice invertido (posting list)

Tabla de vocabulario



Lista de punteros a documentos

Servicio de Índice.
Procesamiento de una consulta

Listas Invertidas

(1)

casa → d0,f0 | d1,f1 | d2,f2 | d3,f3 | d4,f4 | d5,f5 | d6,f6 | d7,f7

árbol → d2,f2 | d3,f3 | d5,f5 | d6,f6 | d8,f8 | d9,f9

---

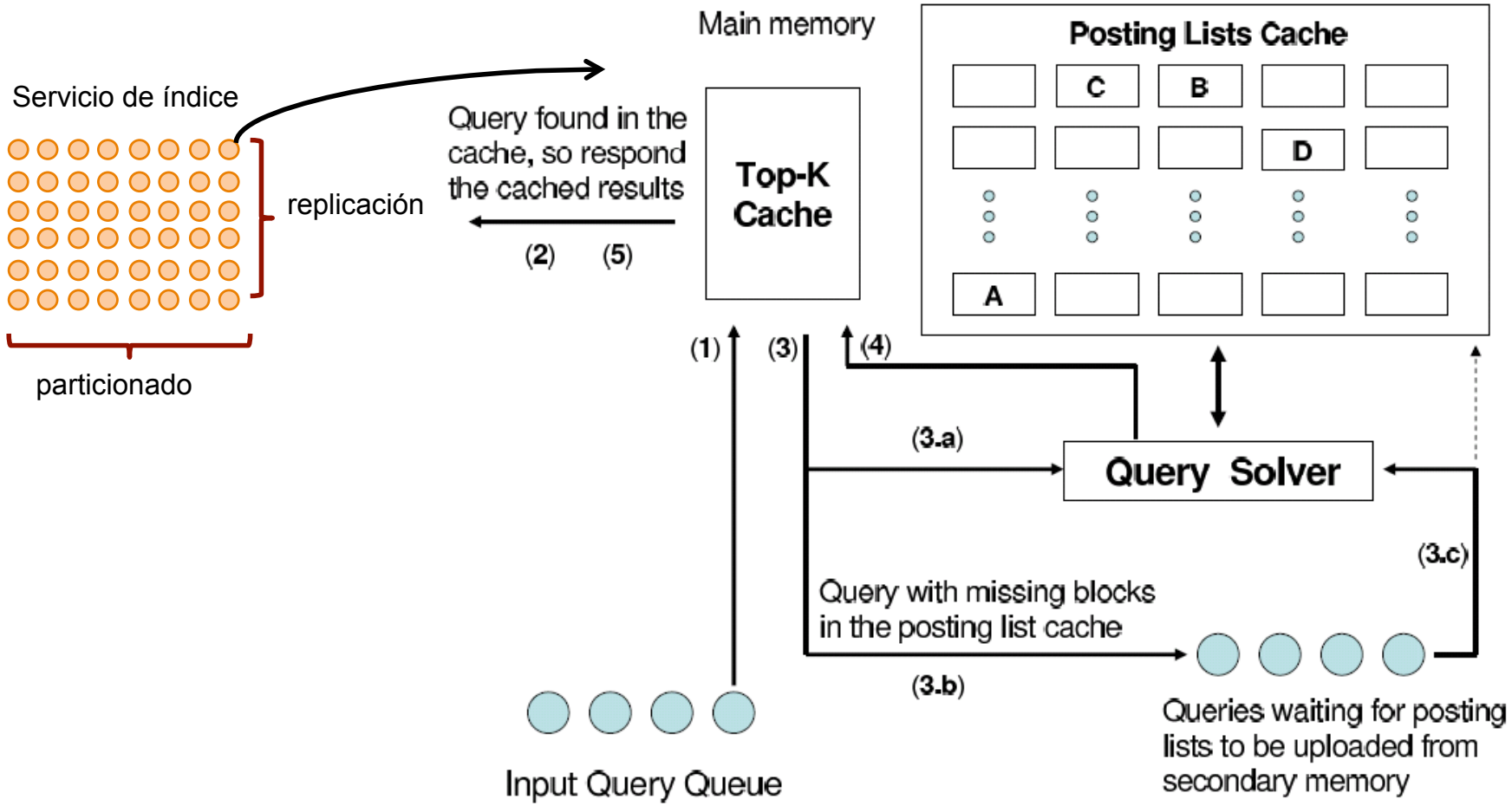
(2) Intersección(casa,árbol) → d2,f2 | d3,f3 | d5,f5 | d6,f6

---

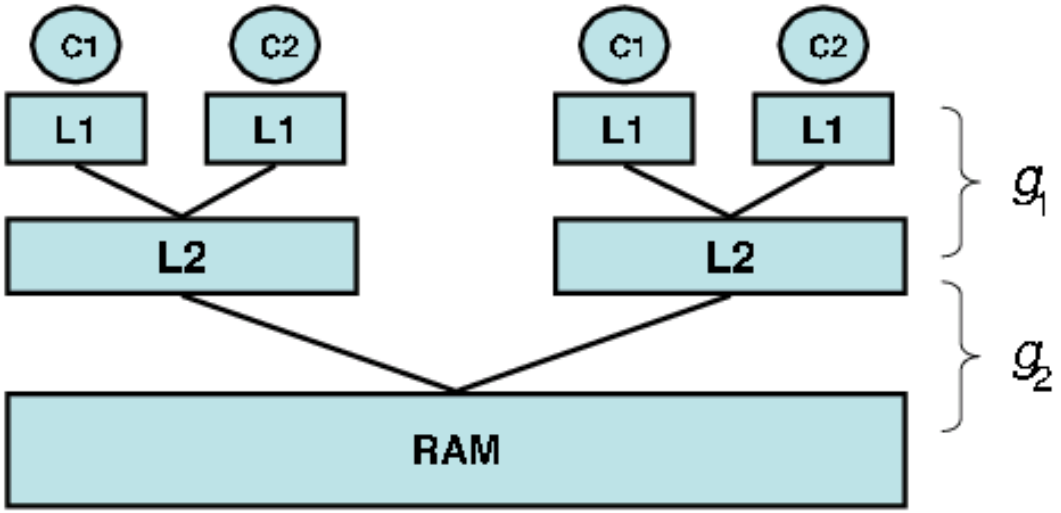
(3) Ranking(d2,f2 | d3,f3 | d5,f5 | d6,f6) → d6,f6 | d3,f3

Top-K

Un nodo del servicio de índice.
Camino seguido por una consulta



Los nodos del cluster generalmente son procesadores multi-core

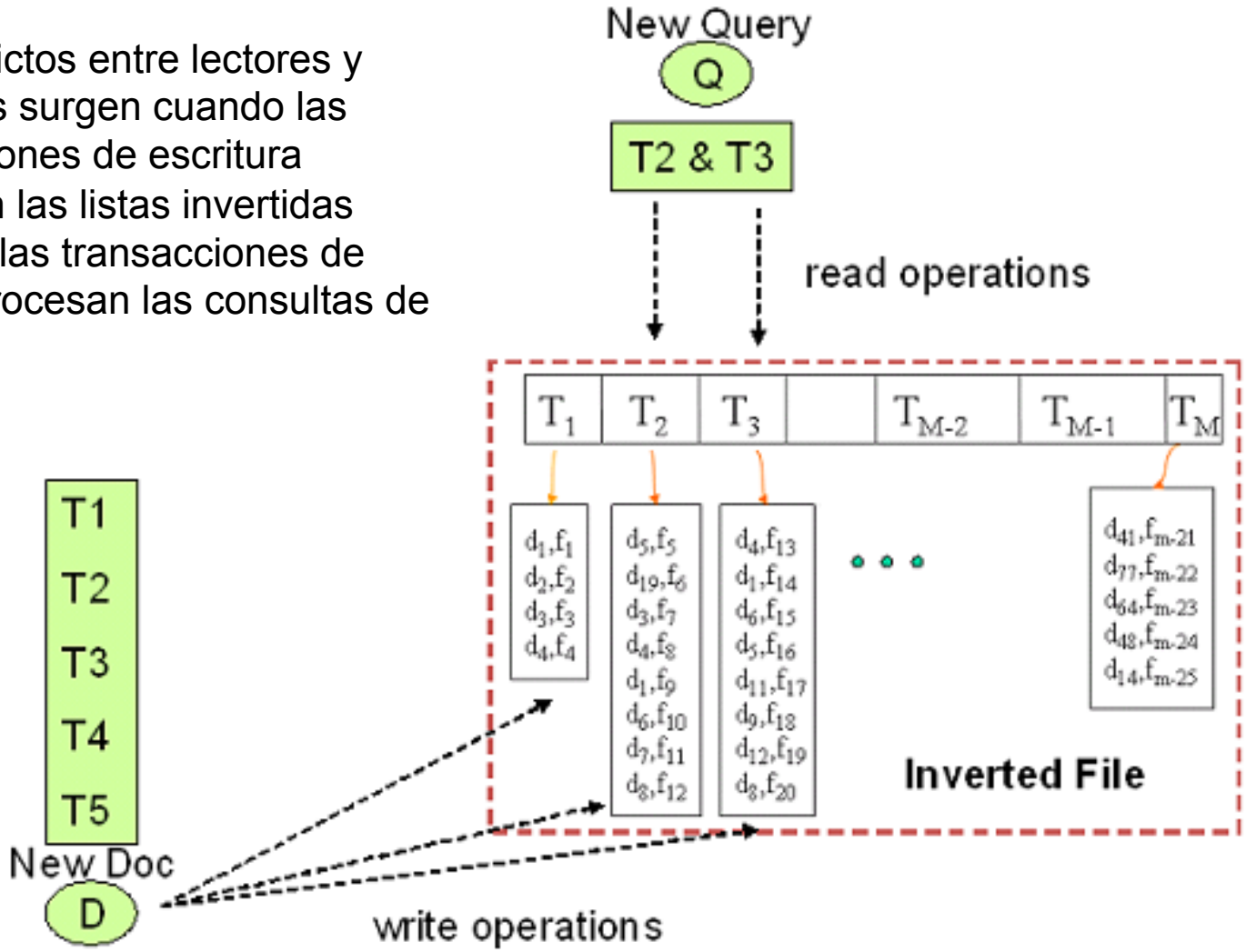


La gestión de threads debería optimizar el uso de los núcleos y la jerarquía de memoria del procesador y evitar conflictos de lectores y escritores durante el procesamiento de transacciones.

Servicio de Índice.

Los conflictos entre transacciones de lectura y escritura deben ser resueltos de manera on-line.

Los conflictos entre lectores y escritores surgen cuando las transacciones de escritura modifican las listas invertidas mientras las transacciones de lectura procesan las consultas de usuarios.



Servicio de Índice (multi-threading)

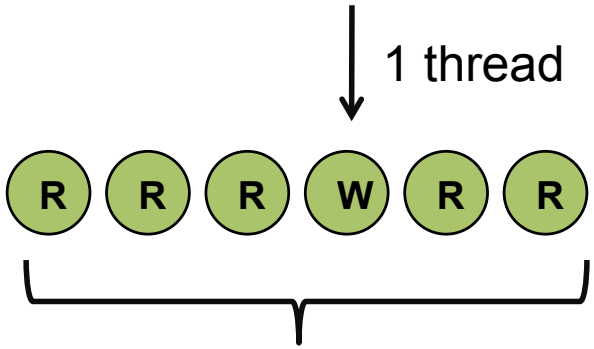
Dos propiedades deseables para la sincronización de transacciones: Atomicidad y seriabilidad

- **Atomicidad:** durante el procesamiento de una consulta, es deseable no afectar el resultado del ranking de documentos evitando que escritores modifiquen las listas invertidas involucradas.

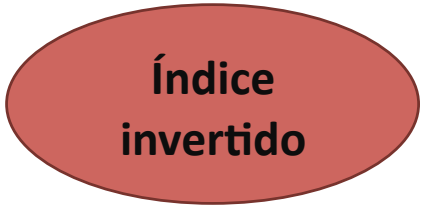
- **Serialidad**: que el resultado de la ejecución concurrente de varias transacciones sea equivalente a haberlas ejecutado de manera secuencial, una por una, respetando el orden de llegada al nodo.

Soluciones factibles desde la literatura.
Solución 1: lock read/write a nivel de índice

- Async. Modo de los motores Web.
- Un thread por transacción **read** y un thread por transacción **write**.



cola de lockRead y lockWrite

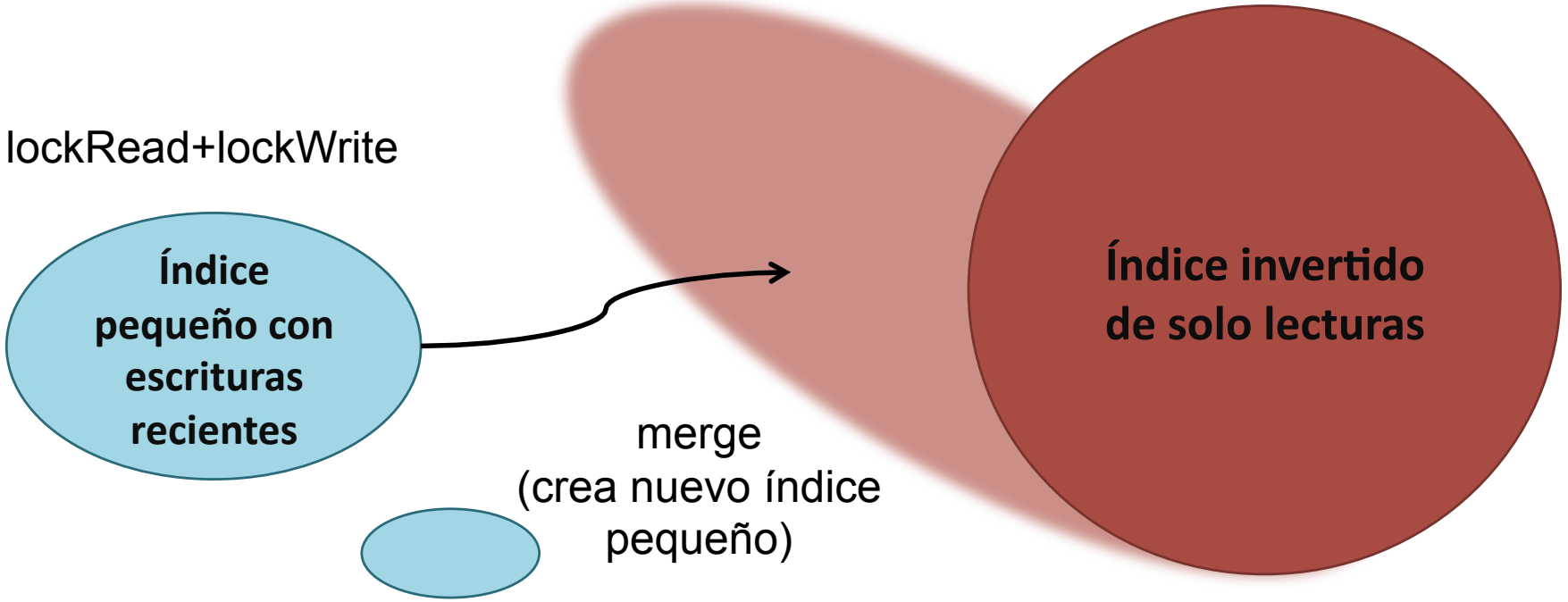


Se permiten muchos lectores dentro del índice, pero un solo escritor por vez

Soluciones factibles desde la literatura

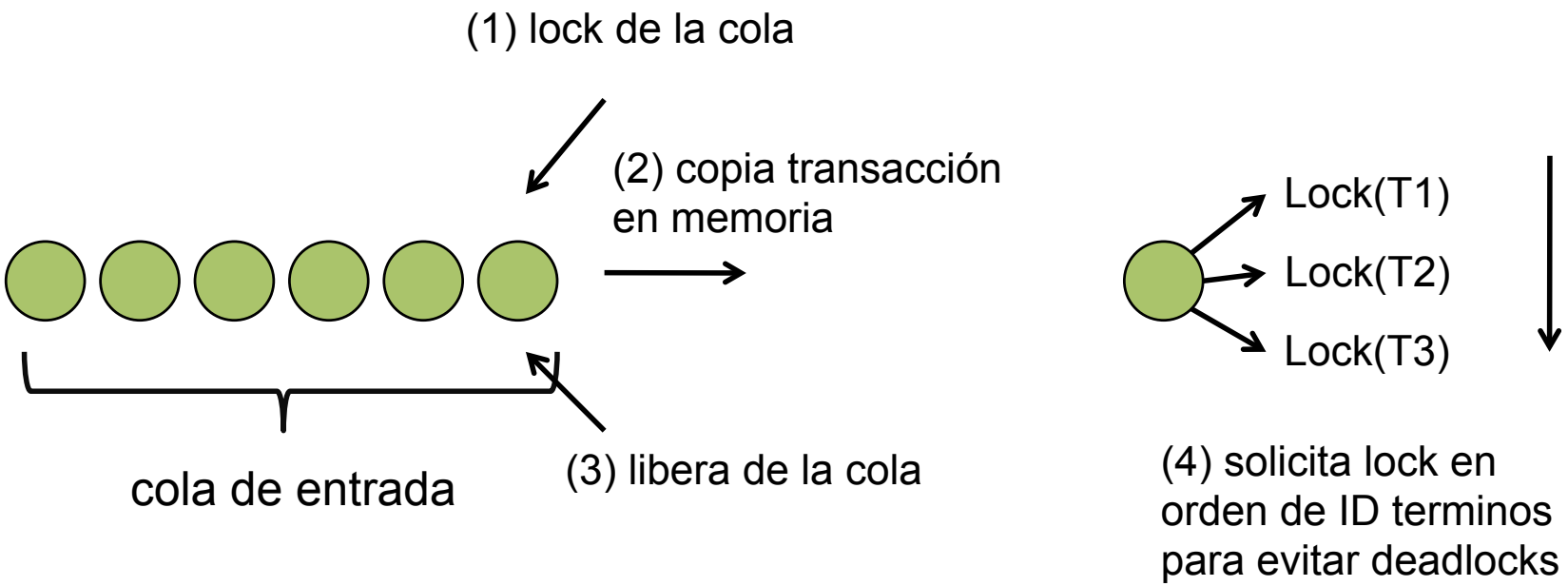
Solución 2: Mejora a solución 1 pero con más consumo de memoria

- Supone **inserciones** al **final** de cada **lista** para soportar intersecciones.
- Se utilizan en algunos motores de búsqueda verticales.
- Reduce la contención, porque limita las escrituras a una sección pequeña del índice



Soluciones factibles desde la literatura
Solución 3: Lock a nivel de términos o listas invertidas

- Protocolo 2 fases (base de datos), evitando deadlocks al pedir lock en el orden dado por los identificadores de término.



Solución Propuesta

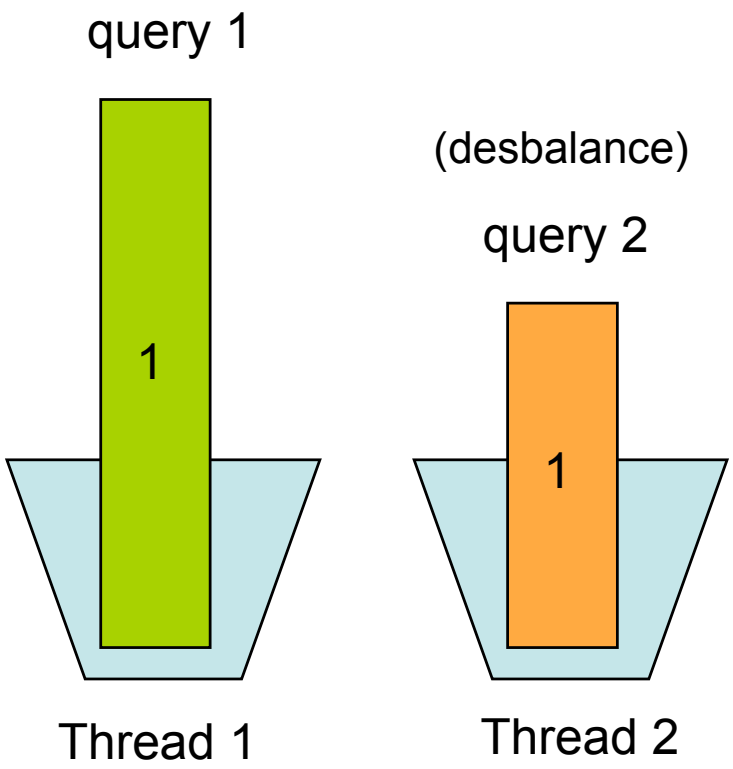
- Asignar un thread principal al nodo de búsqueda que sigue de manera **secuencial** el camino para resolver una consulta, y en cada paso recurre a tantos threads como cores tenga el procesador para paralelizar las operaciones de mayor costo. (ranking, actualización del índice y administración de cache).

- Se utiliza paralelismo síncrono para organizar de manera eficiente las operaciones realizadas por los threads.

Sync versus Async

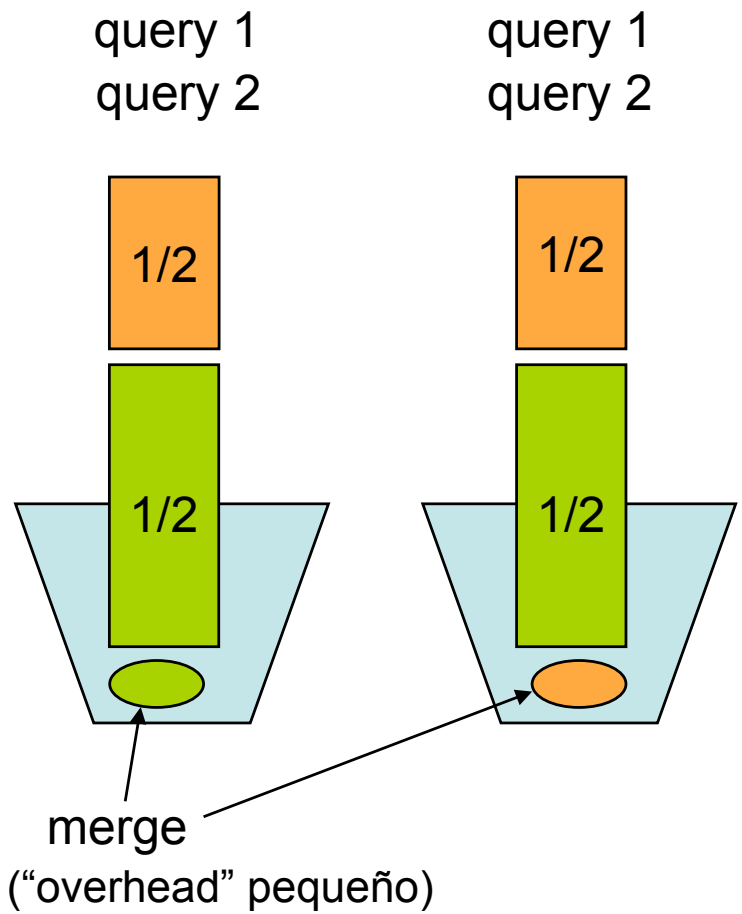
El principio de eficiencia detrás de la estrategia propuesta

Async

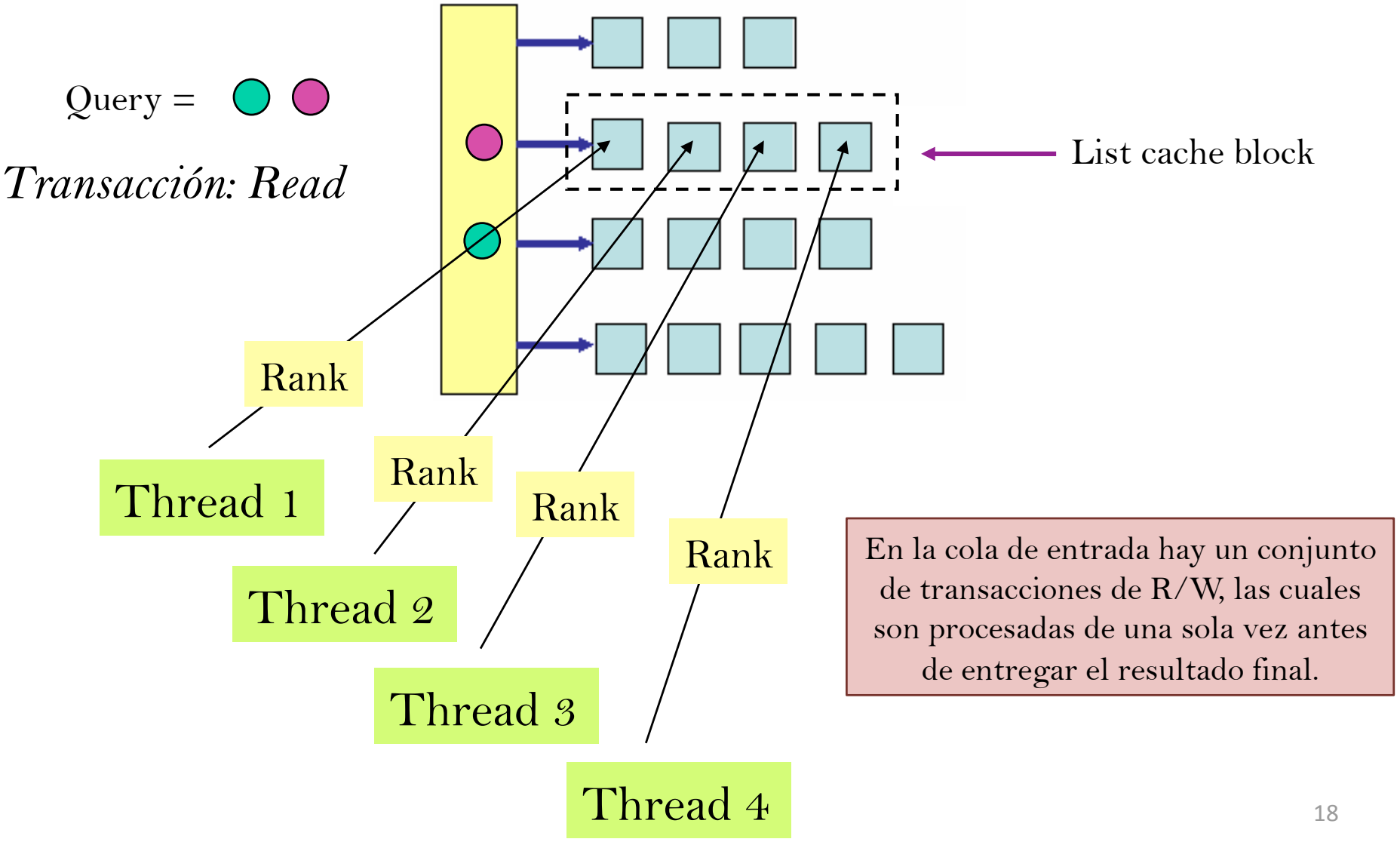


La acumulación de desbalances puede provocar una reducción del throughput.

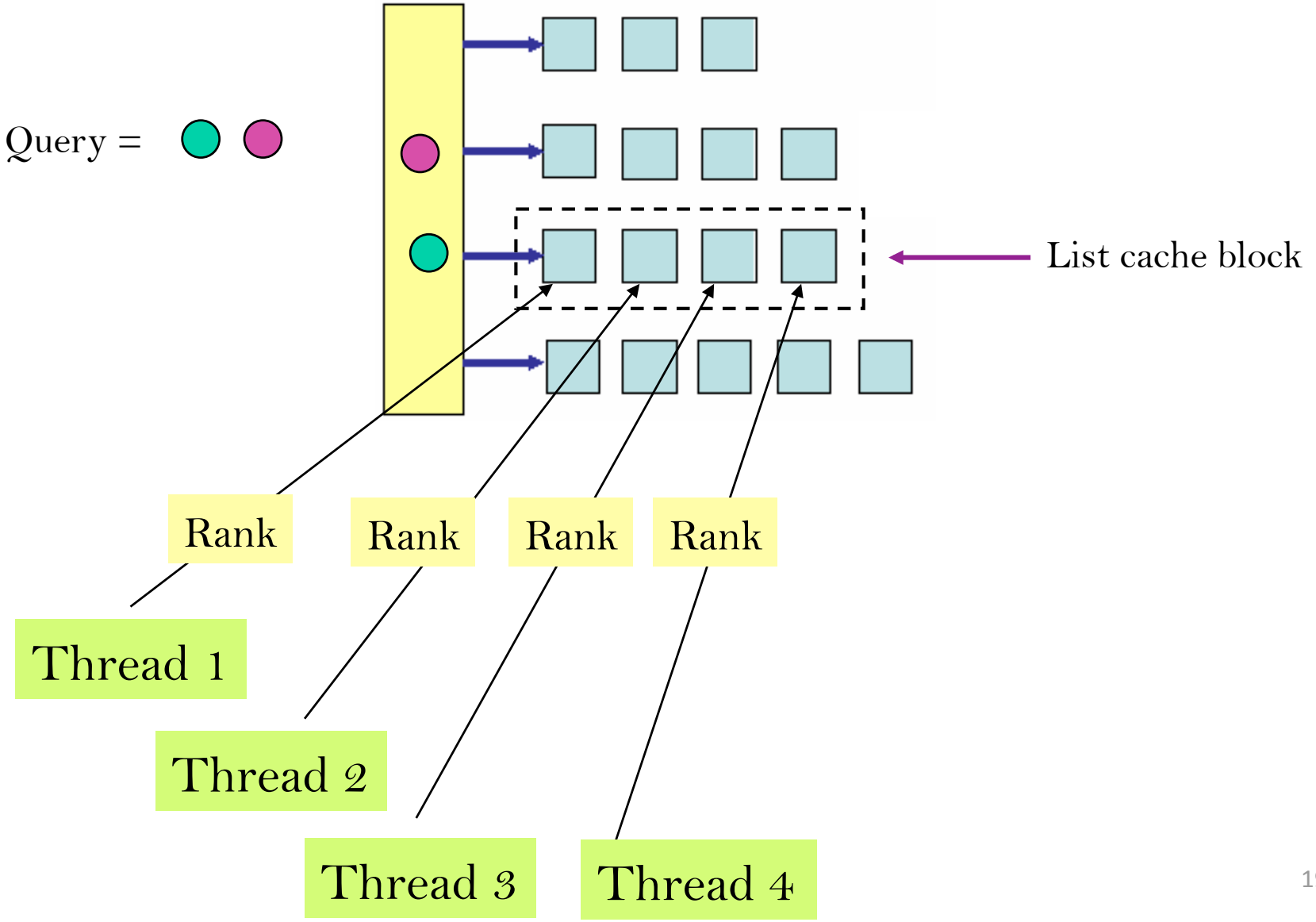
Sync



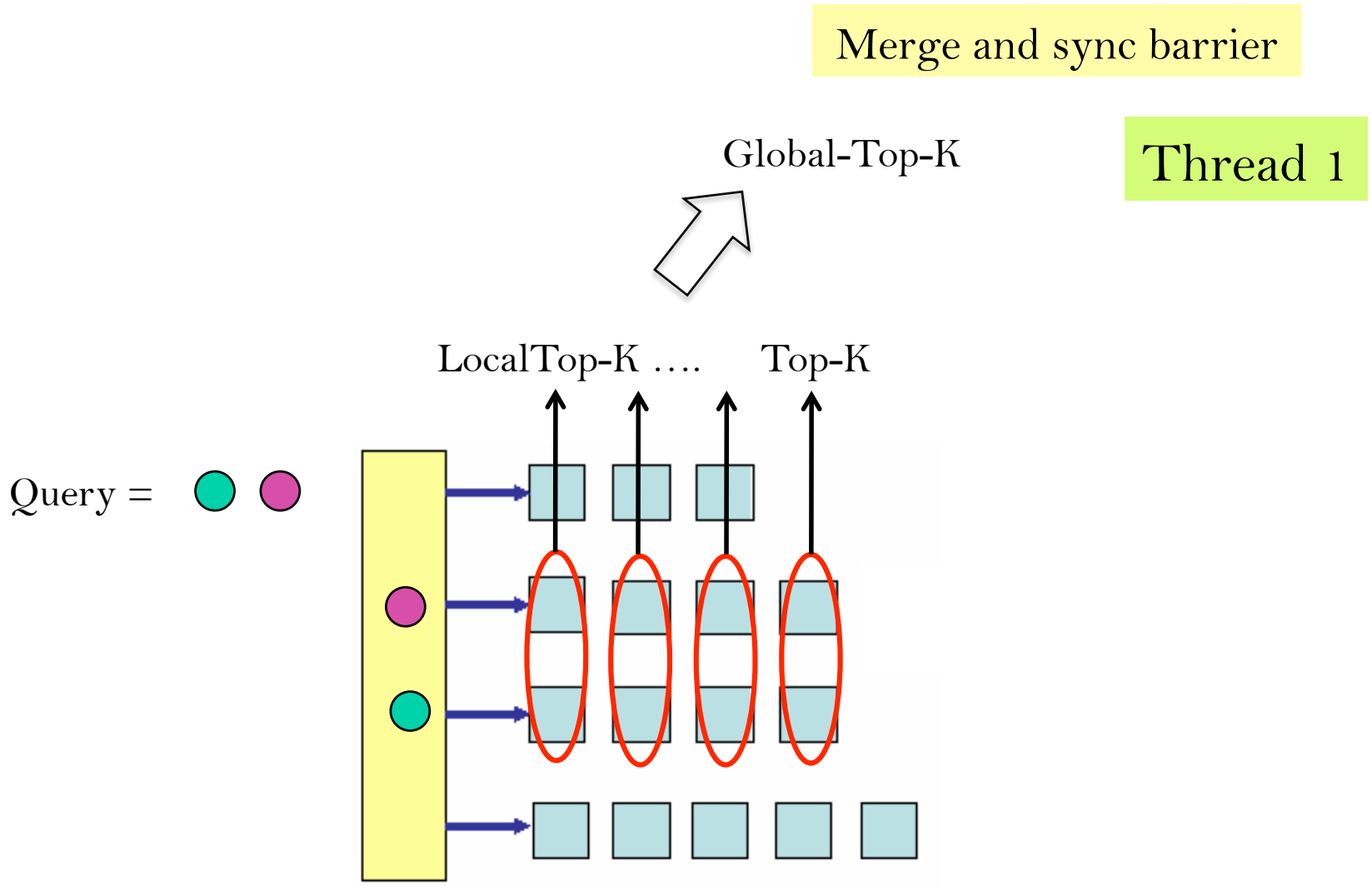
Solución Propuesta.
Algoritmo Bulk Processing (BP)



Solución Propuesta.
Algoritmo Bulk Processing (BP)

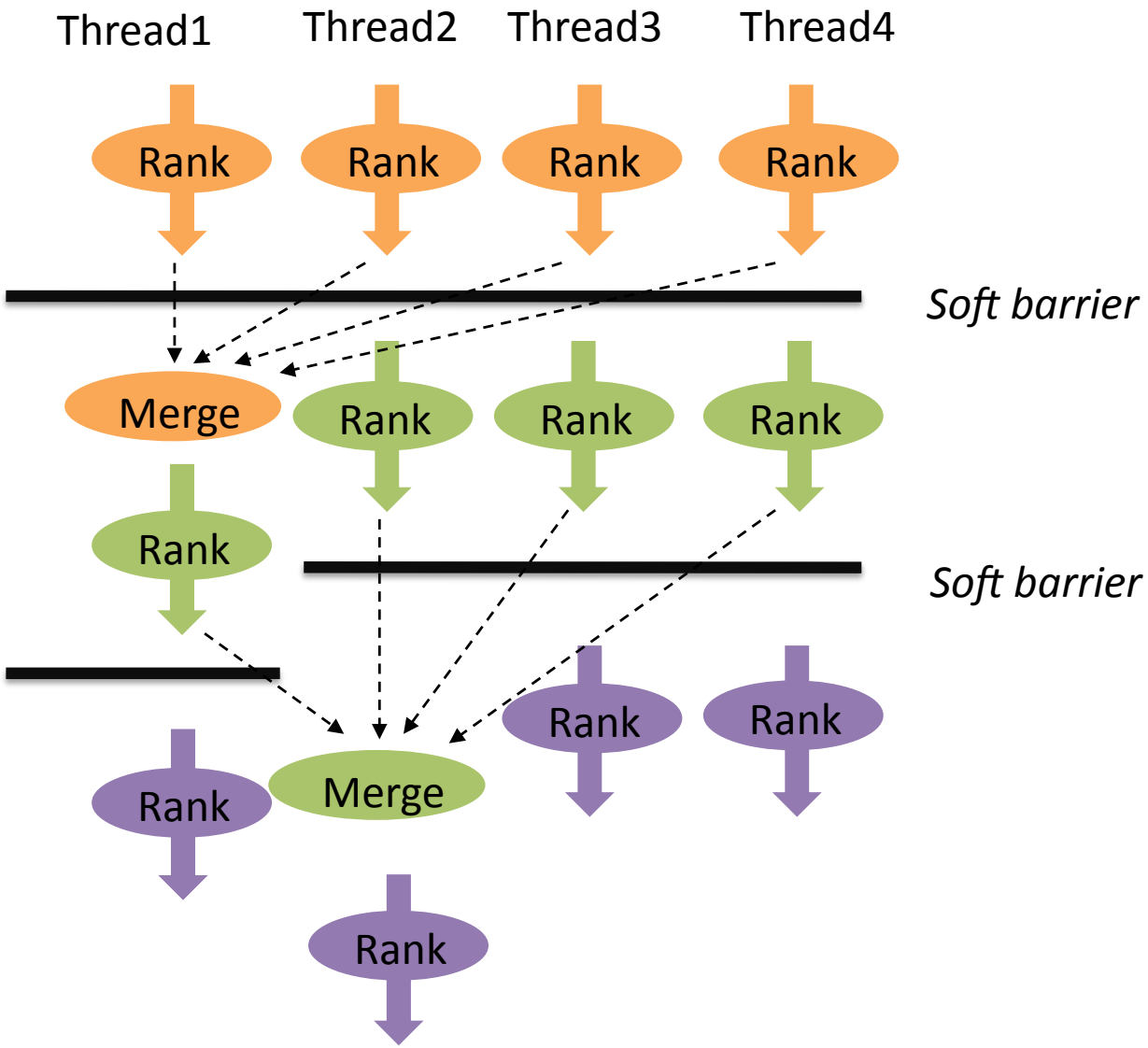


Solución Propuesta.
Algoritmo Bulk Processing (BP)

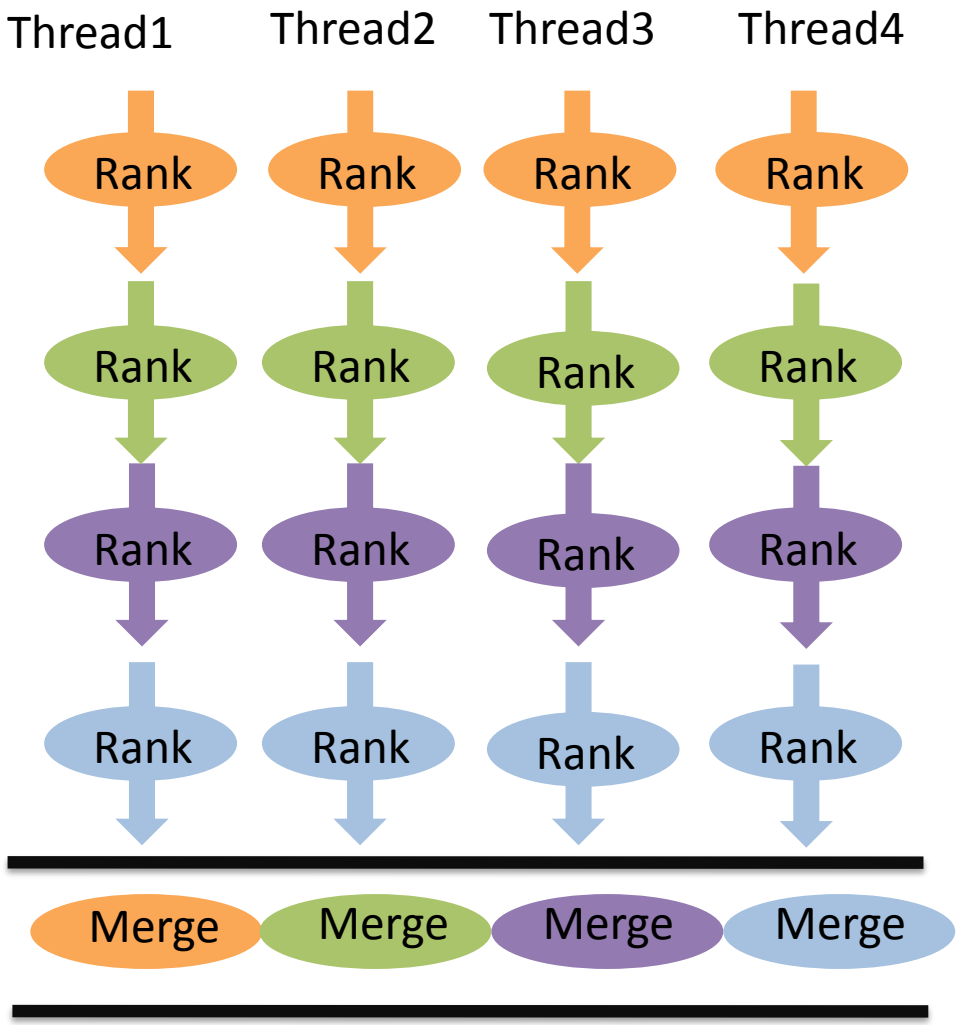


Solución Propuesta.

Sincronización relajada (una transacción por paso)

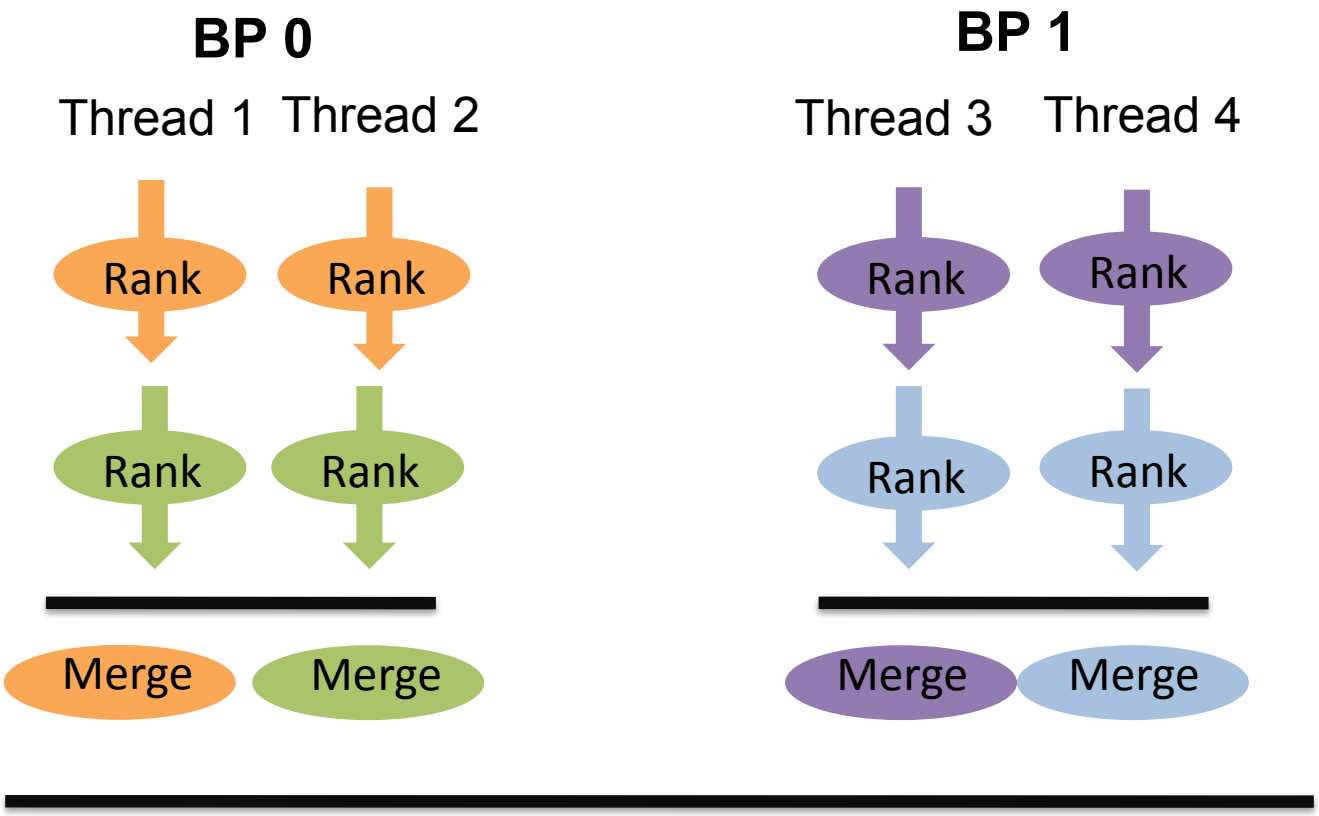


Solución Propuesta.
Agrupación dinámica en tráfico alto



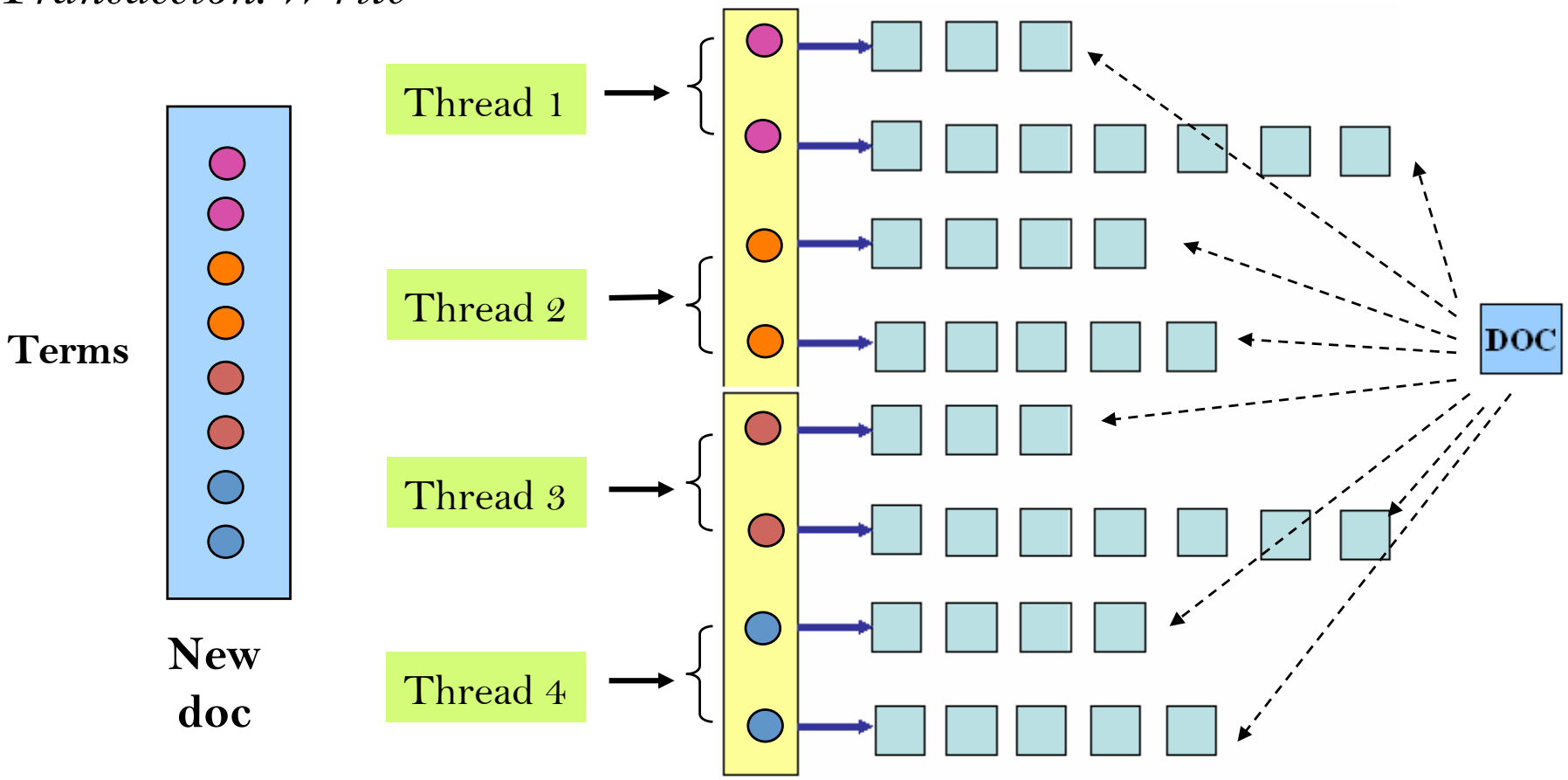
Solución Propuesta.

Varias instancias de BP para listas invertidas pequeñas.



Solución Propuesta.
Algoritmo Bulk Processing (BP)

Transacción: Write

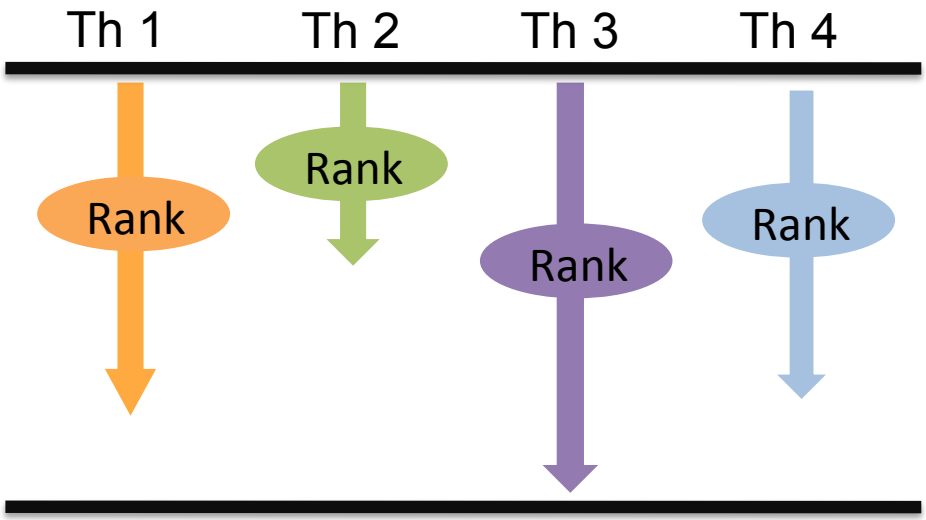


Comparación con otras estrategias alternativas adaptadas desde la literatura

- Concurrent Reads (CR)
- Term Level Parallelism (TLP)
  - Read/write lock protocol (TLP1)
  - Two phase exclusive lock protocol (TLP2)
- Relaxed Term Level Parallelism (RTLP)
- Relaxed Block Level Parallelism (RBLP)

Concurrent Reads (CR)

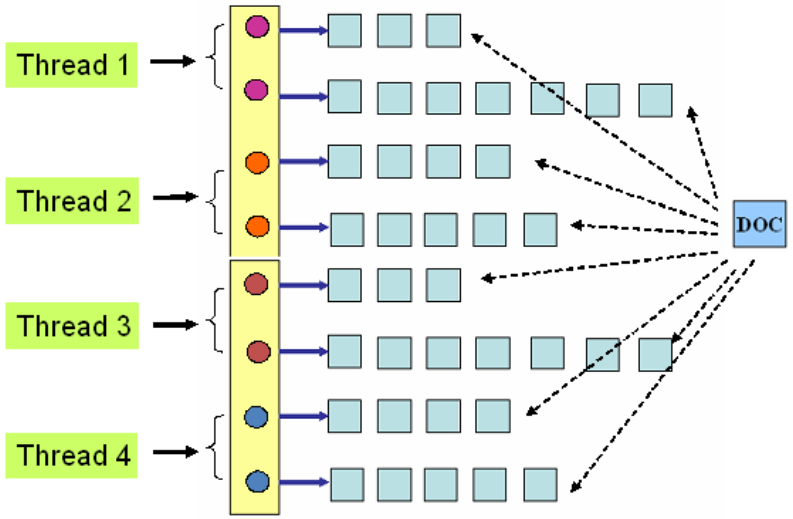
Cada thread resuelve una consulta



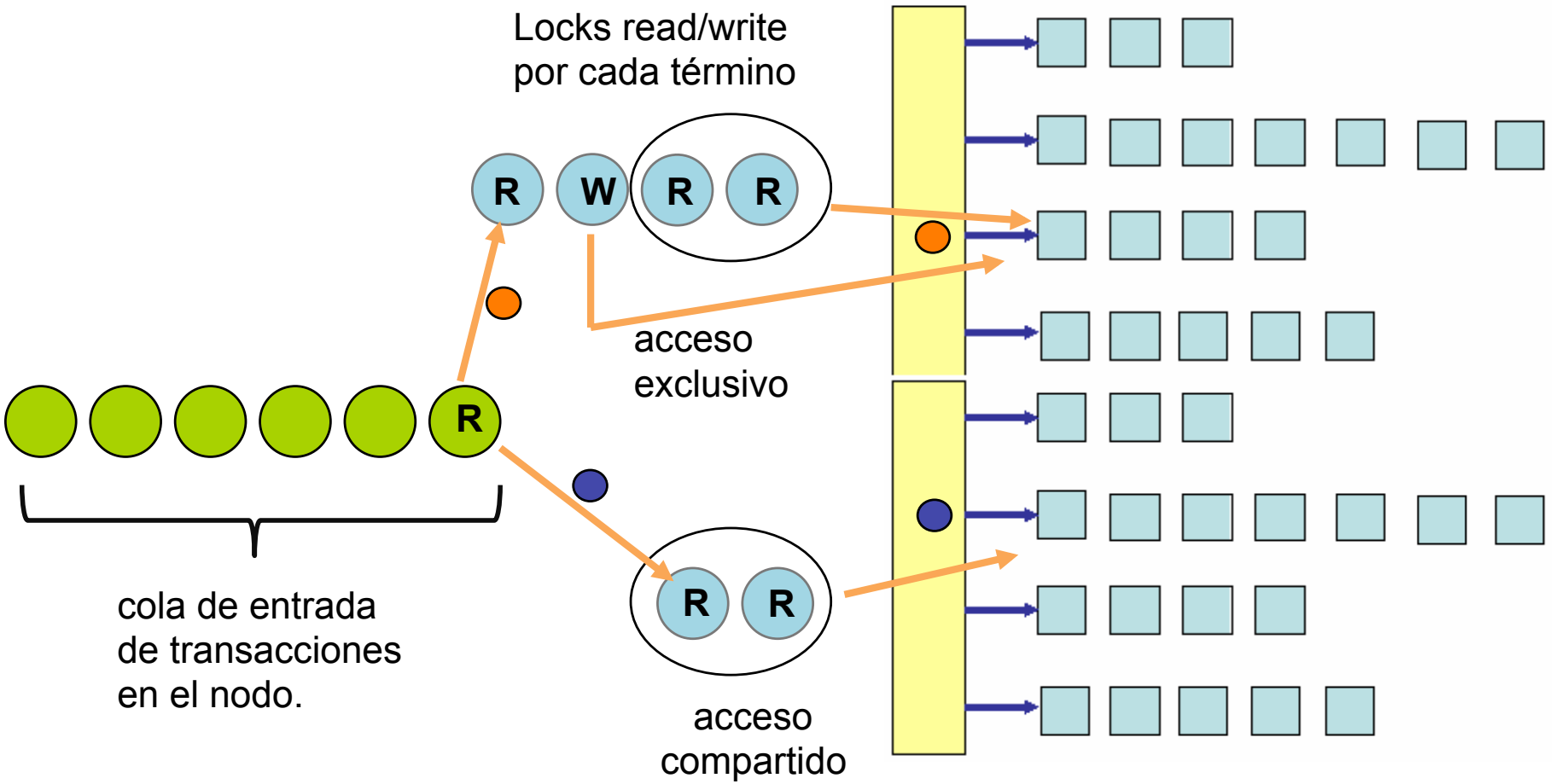
Se estudio JobStealing pero no dio buenos resultados

Barrera de sincronización

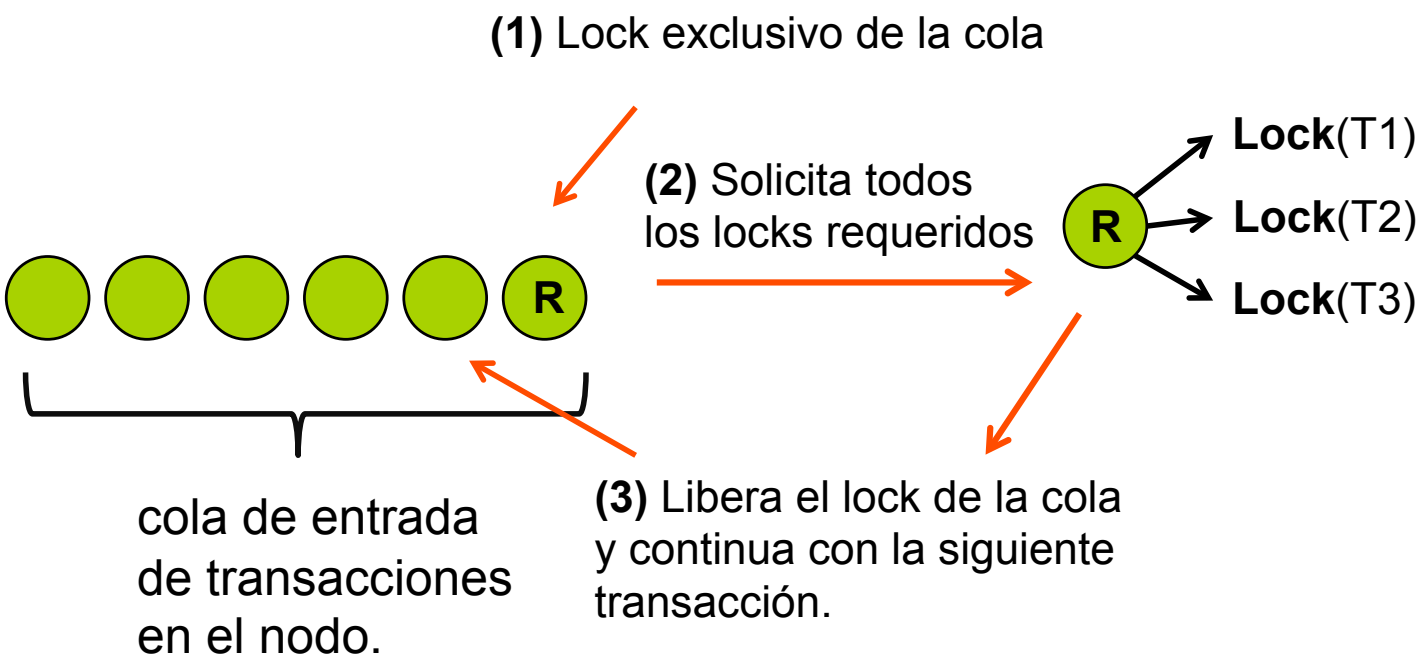
Todos los threads colaboran en la inserción



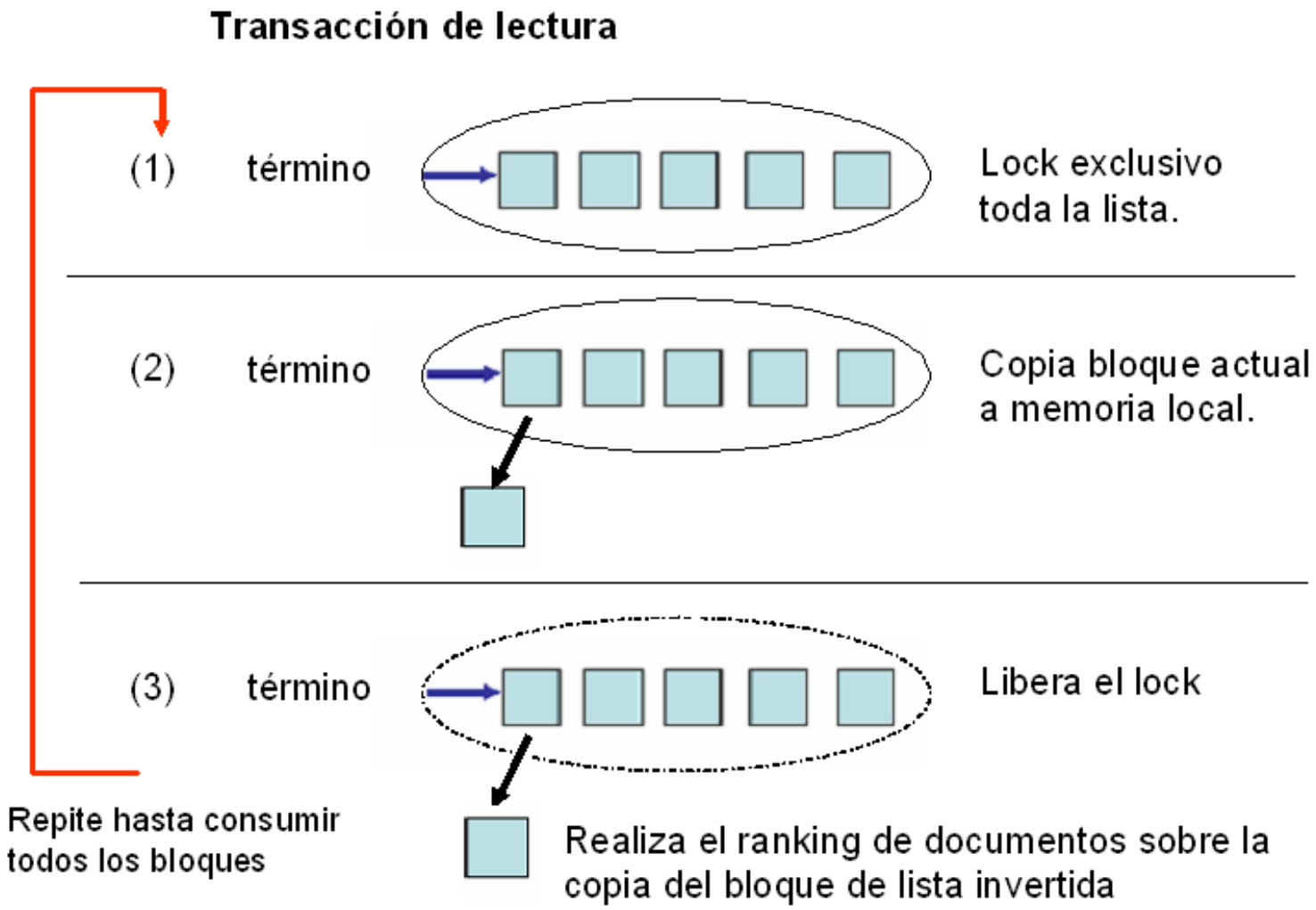
Read/write lock protocol (TLP1)



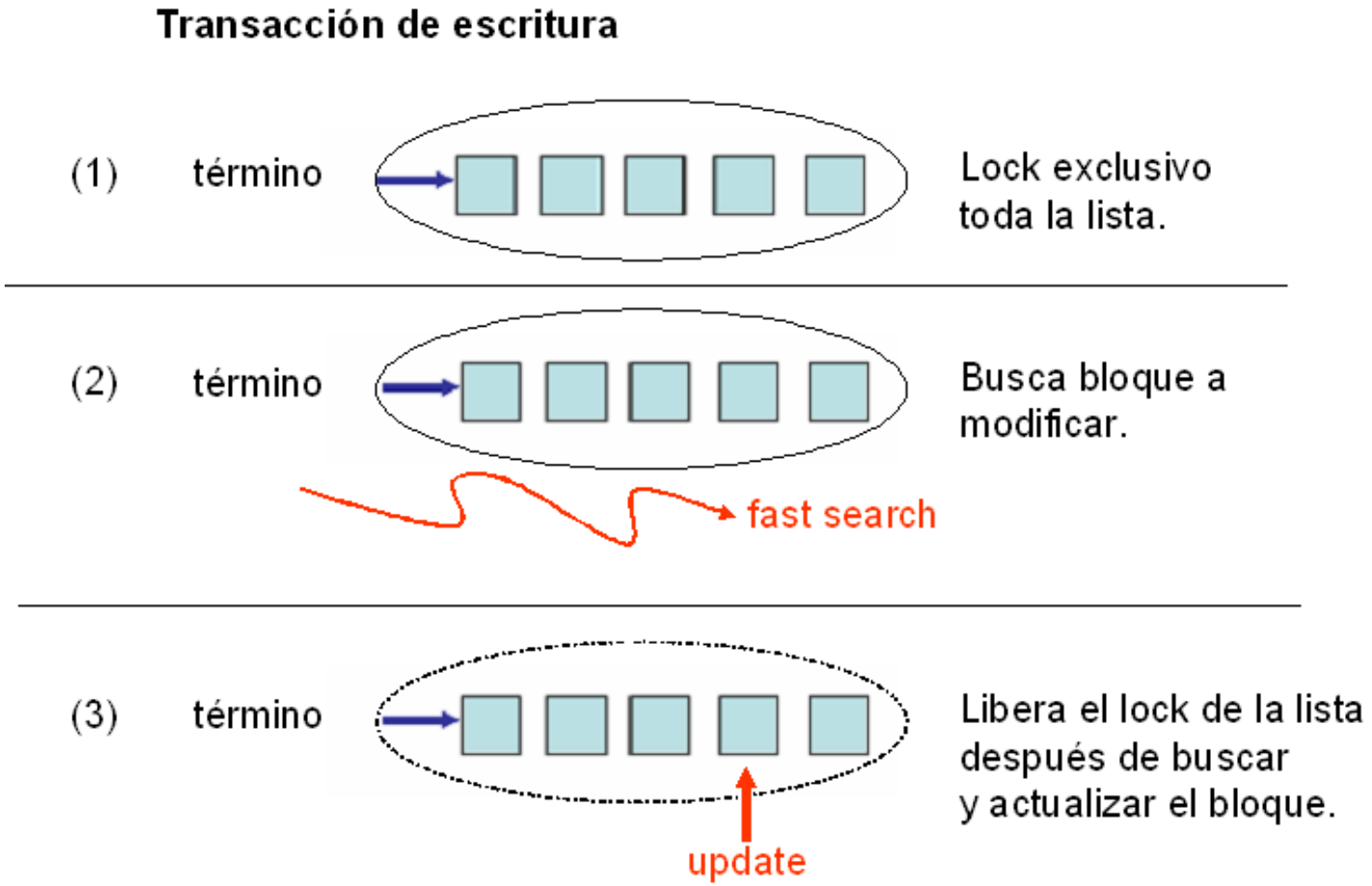
Two phase exclusive lock protocol (TLP2)



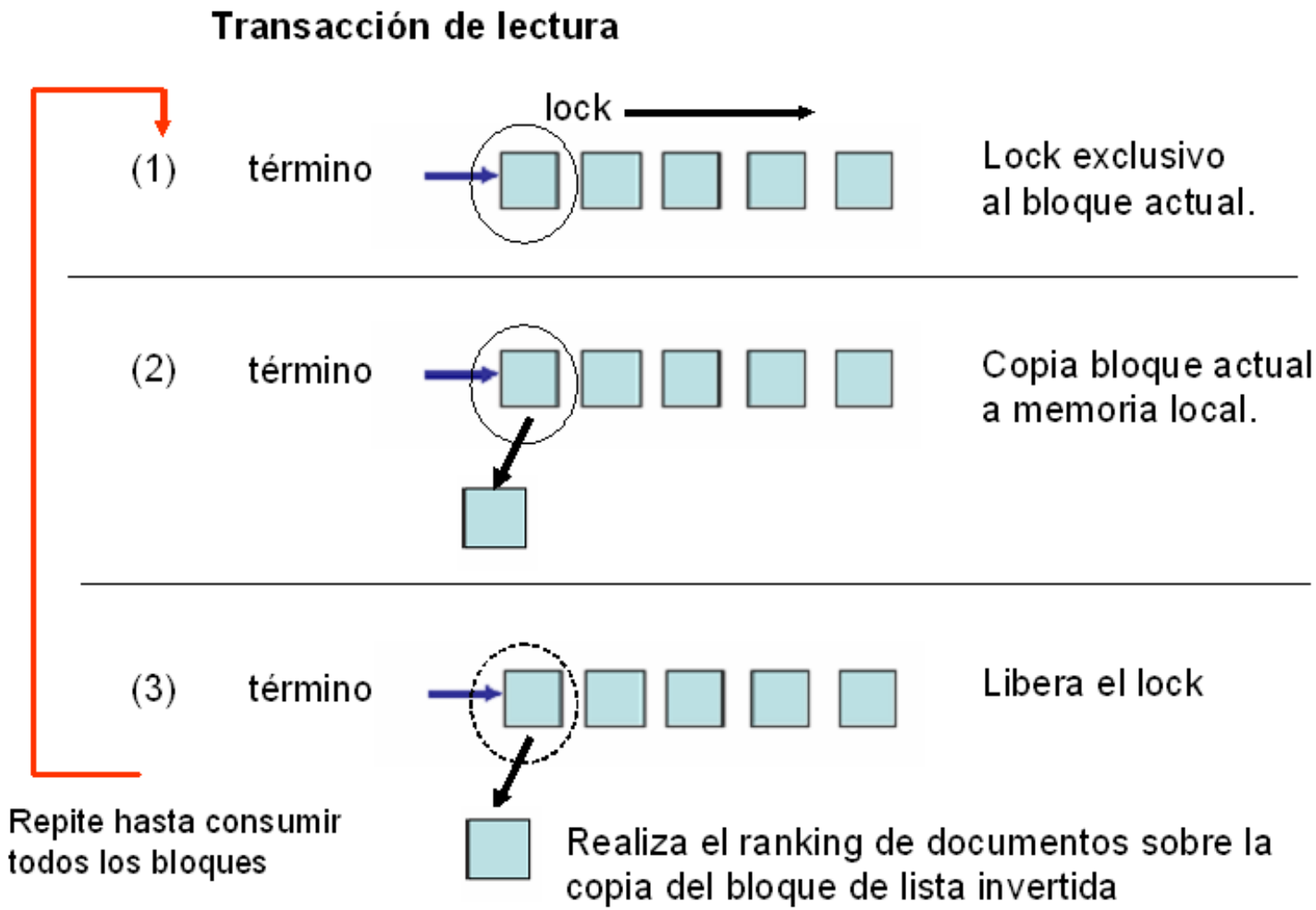
Relaxed Term Level Parallelism (RTLP)



Relaxed Term Level Parallelism (RTLP)

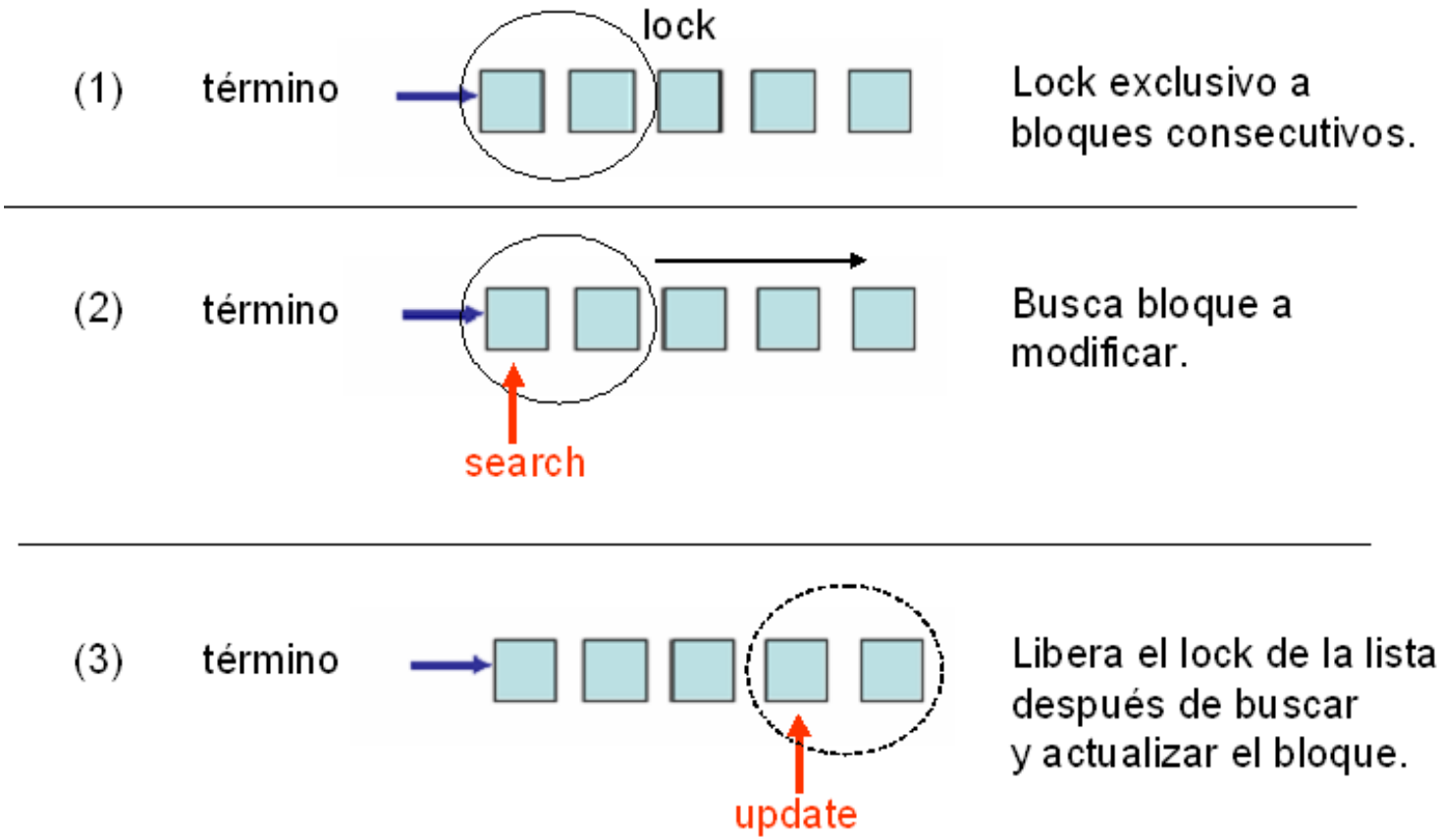


Relaxed Block Level Parallelism (RBLP)



Relaxed Block Level Parallelism (RBLP)

Transacción de escritura



Si el bloque se llena se crea uno nuevo con la mitad de los elementos de los bloques contiguos .

Evaluación de Estrategias
Entorno experimental

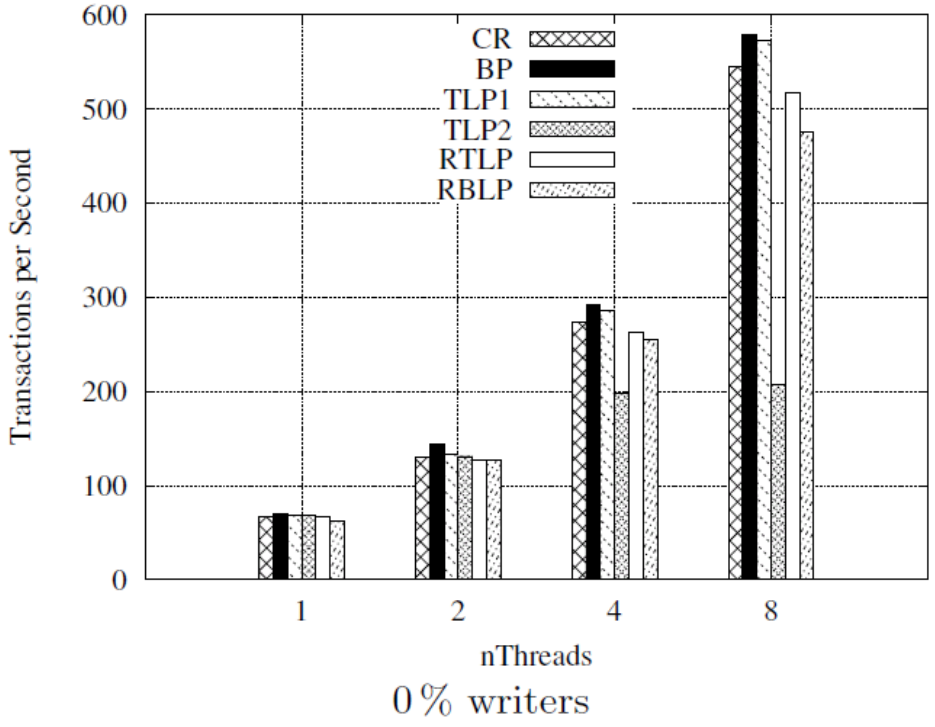
- El conjunto de listas requeridas se encuentra en la caché de listas, y BP operando en modo pesimista procesando una transacción por cada vez.

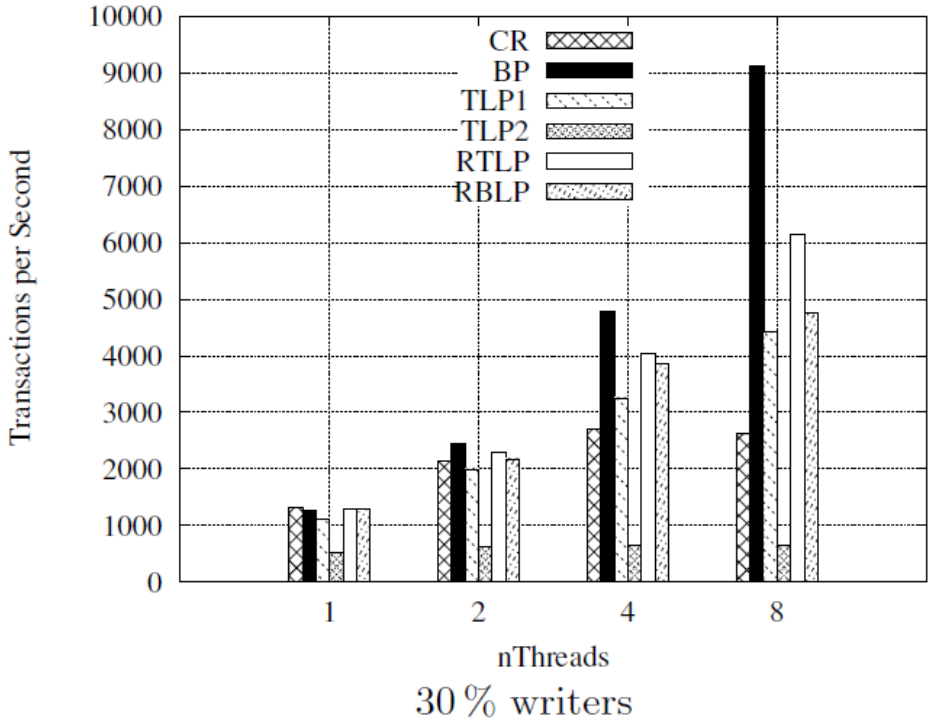
- La tasa de llegada de transacciones es lo suficientemente alta para mantener a los threads ocupados.

- Listas invertidas de la Web UK, log de consultas de Yahoo!, documentos cuyos términos tienen distintos grados de solapamiento con los términos de las consultas.

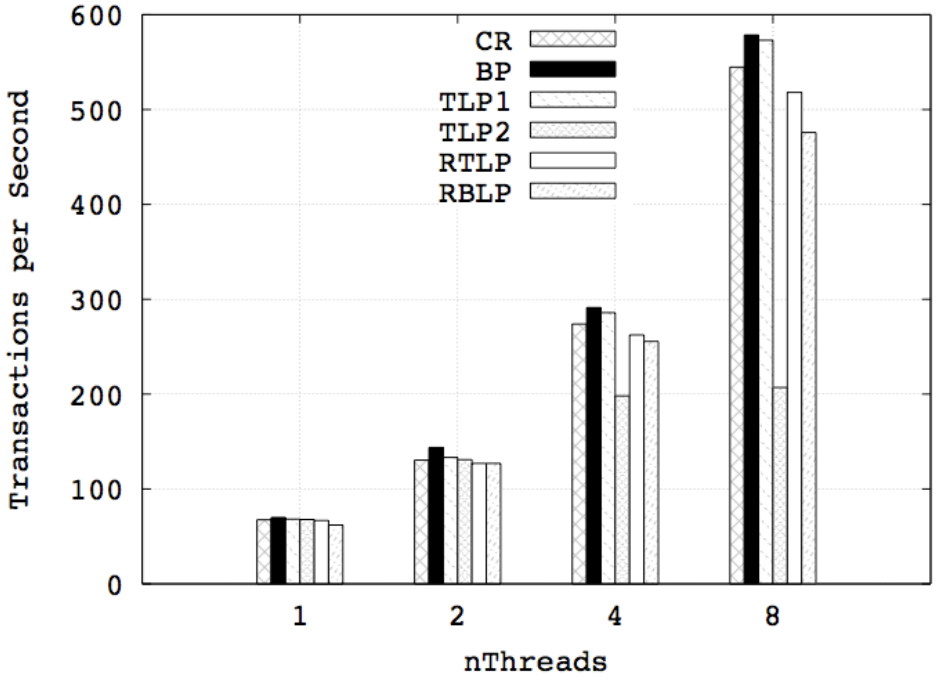
- Experimentos con los procesadores Intel Xeon e Intel i7, operando en modo exclusivo, es decir, sin otros procesos ejecutandose. La capacidad de la memoria principal es lo suficientemente grande como para mantener todas las estructuras de datos.

Rendimiento alcanzado por las estrategias para 0 y 30% de transacciones de escritura (procesador Intel Xeon)

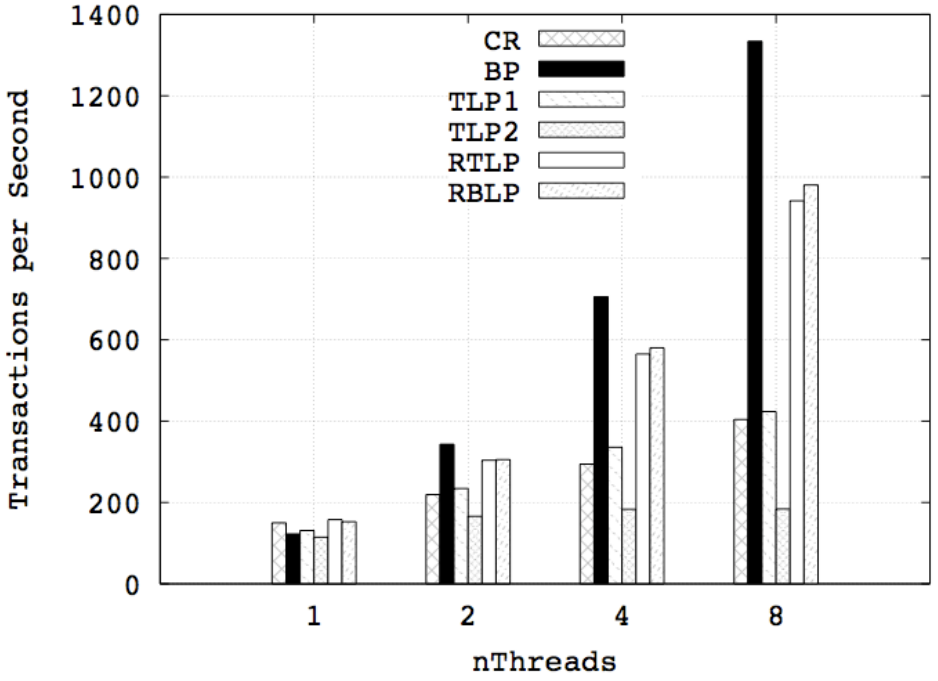




Rendimiento alcanzado por las estrategias, con alto grado de coincidencia entre términos de transacciones consecutivas

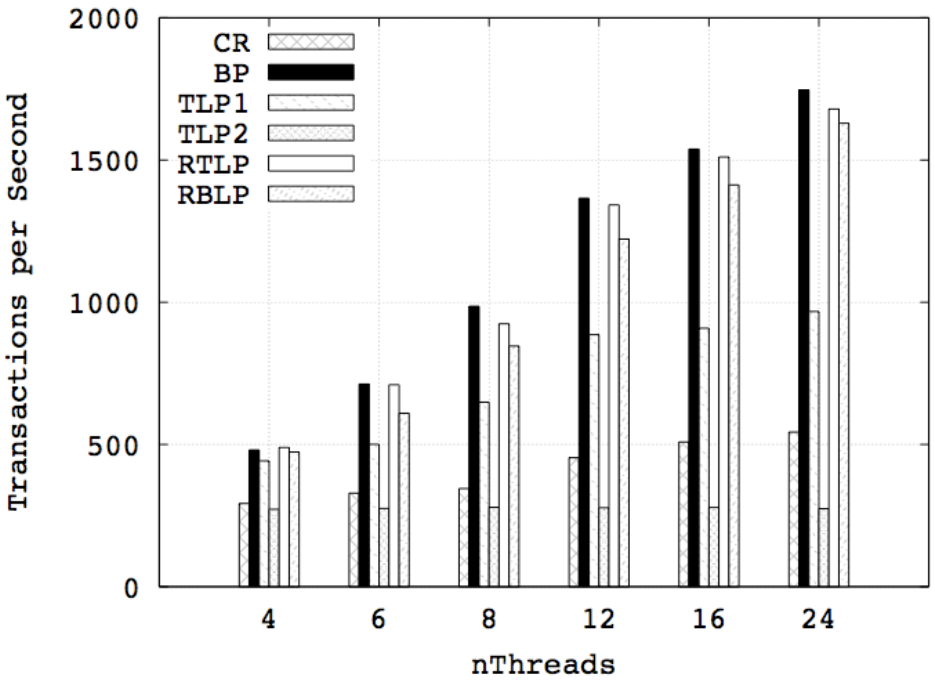


W=0%

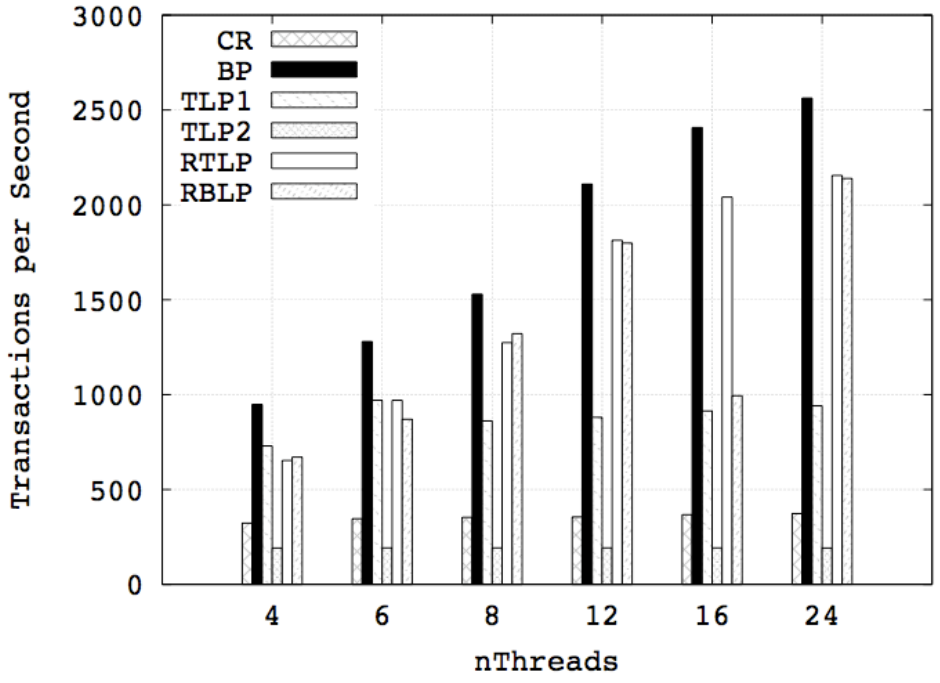


W=30%

Rendimiento alcanzado por las estrategias al aumentando el número de threads por core en otra arquitectura (2x Intel i7)



W=10%



W=40%

Conclusiones de los resultados obtenidos con implementaciones reales

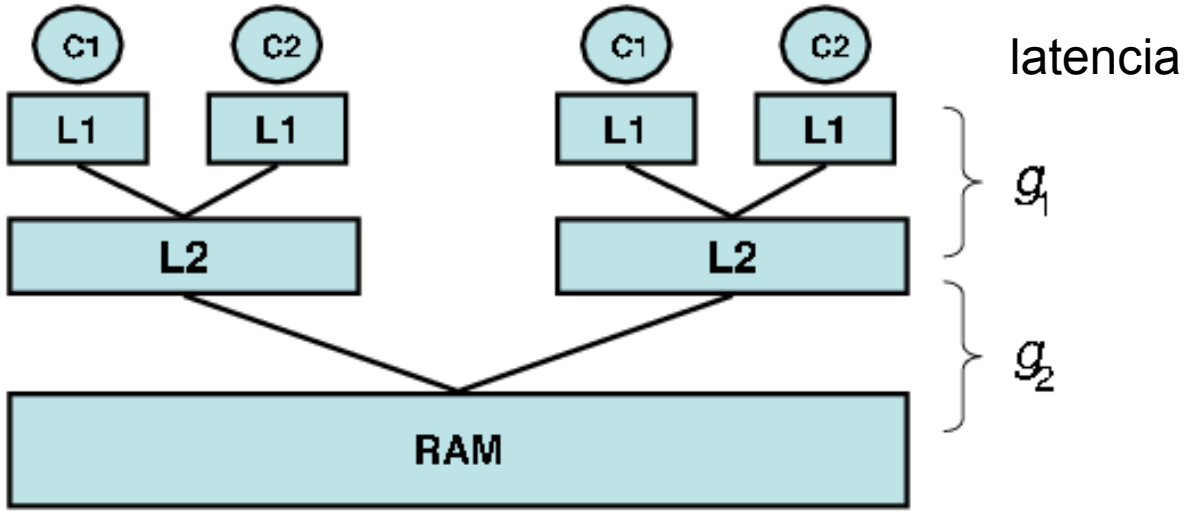
- BP escala de manera eficiente al aumentar el número de threads.
- Solo RTLP y RBLP logran un rendimiento competitivo pero a expensas de la pérdida de atomicidad y serialidad, y la necesidad de diseñar una solución que incluya la administración concurrente de los caches.
- El rendimiento de BP es similar a CR para el caso de solo lecturas.
- CR y TLP1 logran resultados satisfactorios solo para bajas transacciones de escritura.
- TLP2 no logra buen rendimiento debido a que las transacciones de lectura incluyen términos que coinciden ocasionalmente, causando demoras en la ejecución de threads que comparten términos.

Análisis de escalabilidad mediante simulación

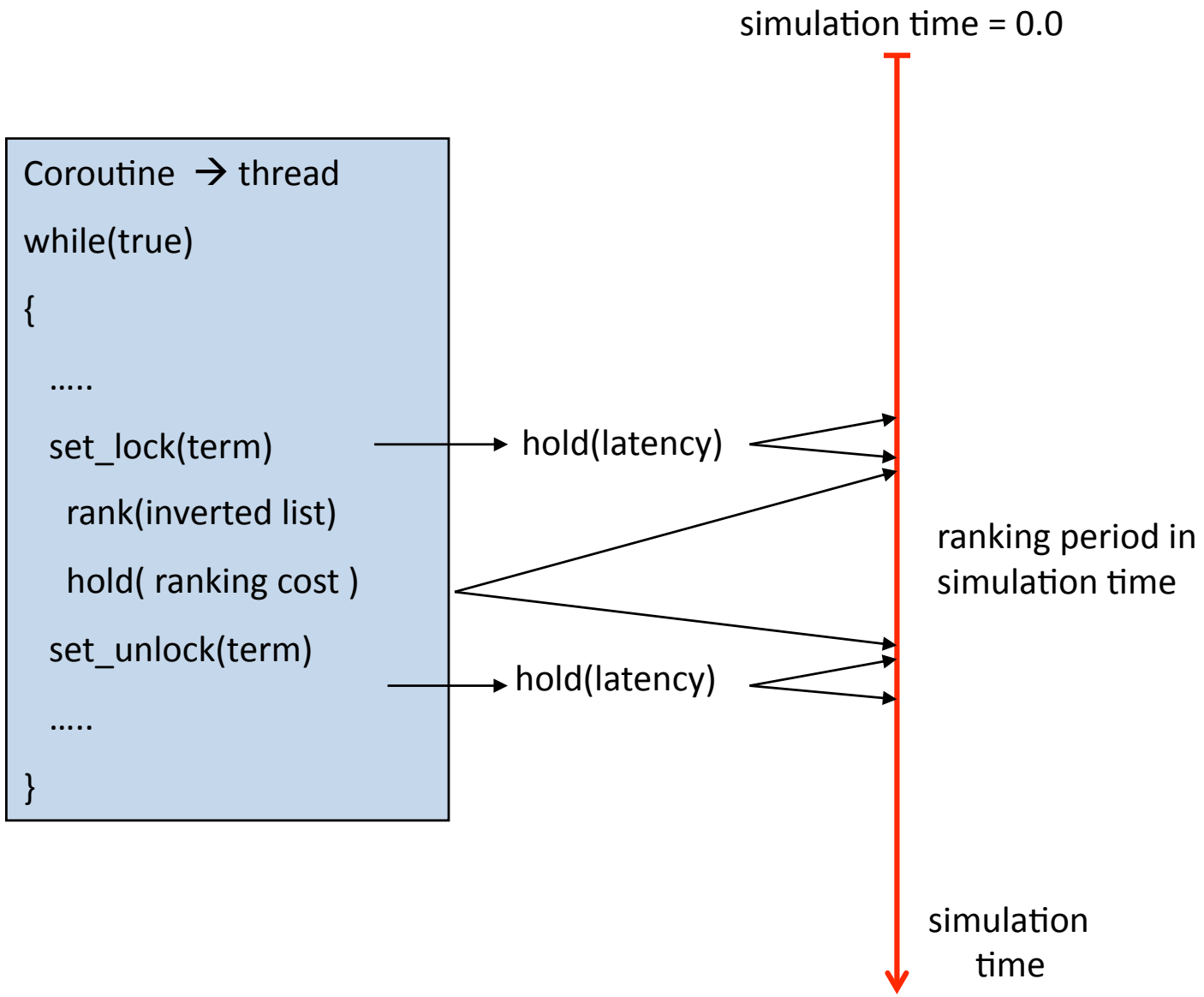
- Para entender **por qué** BP es más eficiente en **throughput** y **escalabilidad**, se implementó un modelo de simulación discreta de costo de las operaciones relevantes de las transacciones.

- Se despliegan los mismos threads utilizando co-rutinas, las cuales ejecutan las mismas operaciones de las estrategias reales, pero simulan el costo de ellas causando en el tiempo de simulación retardos equivalentes a los tiempos de ejecución que demandan las respectivas operaciones reales.

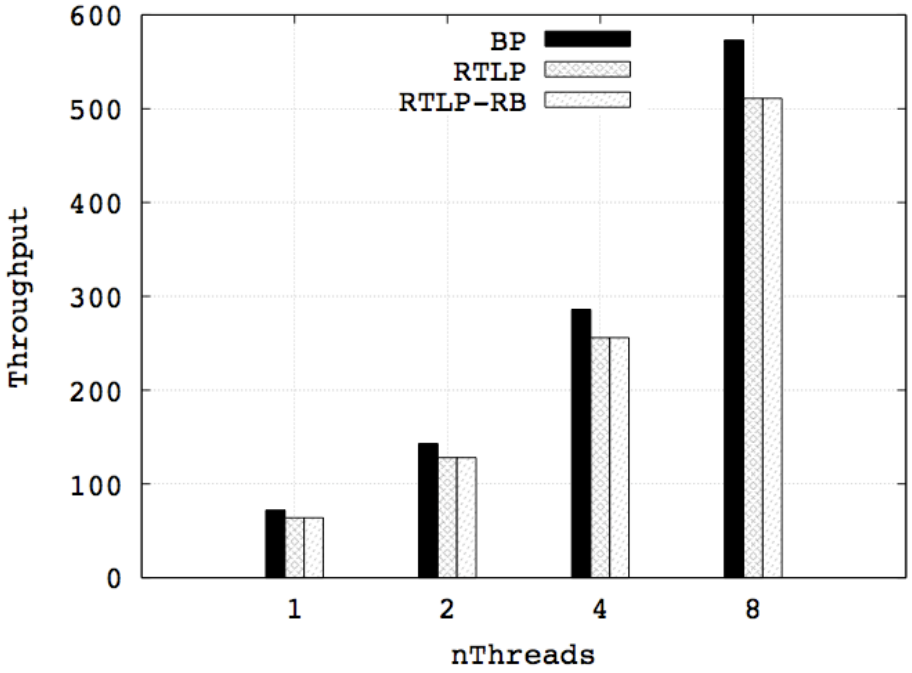
El modelo de simulación considera el costo de transferencias de bloques entre niveles de la jerarquía de memoria



El simulador utiliza co-rutinas para simular el costo de las operaciones relevantes de los threads en el tiempo de simulación.

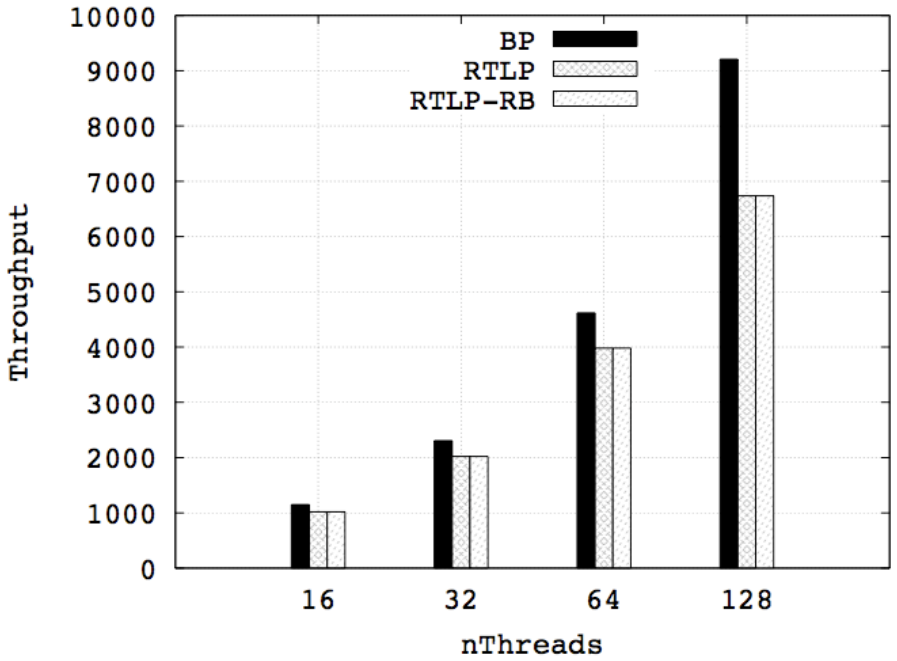


Rendimiento que el modelo de simulación predice para las estrategias BP, RTLP y RTLP con Rollbacks, en el procesador Intel Xeon (idem i7)



0% writes, 1 a 8 threads

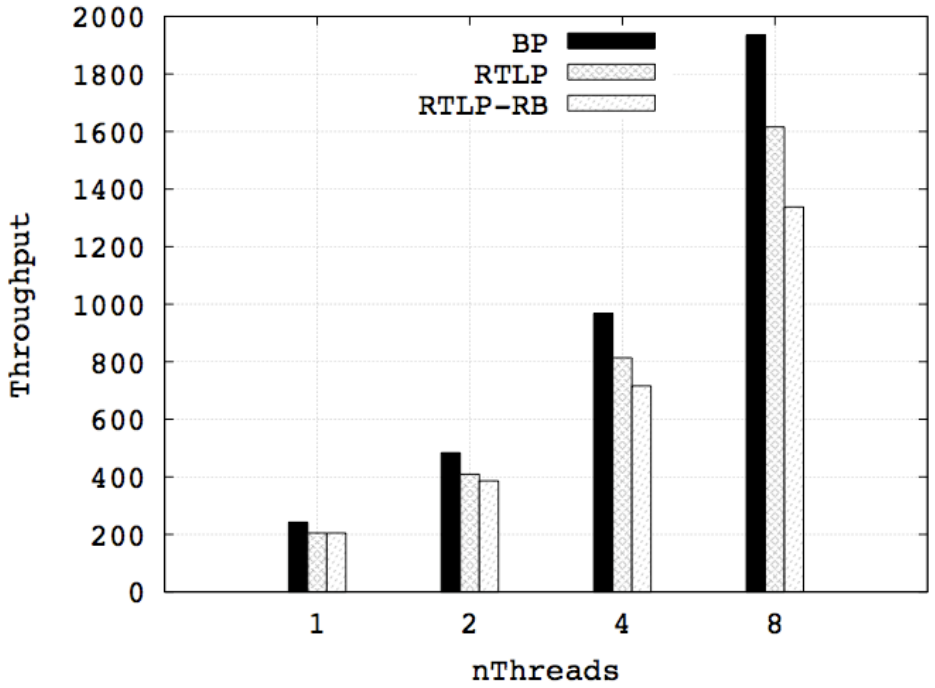
Similar al sistema real



0% writes, 16 a 128 threads

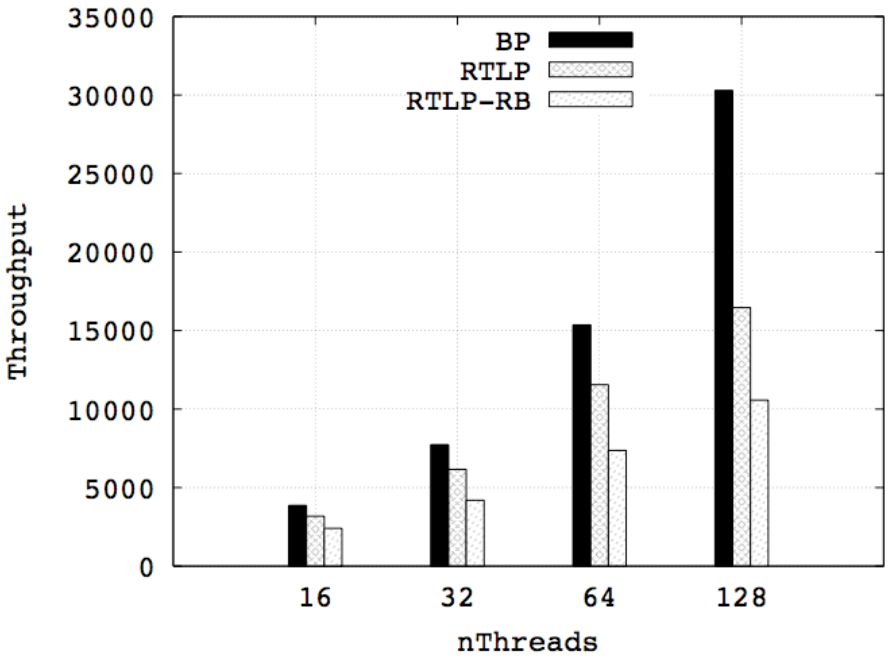
Escalabilidad a muchos threads

Rendimiento que el modelo de simulación predice para las estrategias BP, RTLP y RTLP con Rollbacks, en el procesador Intel Xeon (idem i7)



30% writes, 1 a 8 threads

Similar al sistema real



30% writes, 16 a 128 threads

Escalabilidad a muchos threads

Razones de la pérdida de rendimiento de RTLP

- P_w probabilidad de esperar por un lock
- LB balance de carga (razón del promedio de trabajo que hace un threads en un instante de tiempo, sobre el máximo de tiempo que trabajó un threads)
- RTLP-RB pierde eficiencia porque aumenta el número de rollbacks (cada rollback es una re-ejecución de la transacción)

	0%		30%	
NT	P_w	LB	P_w	LB
2	0	1	0	1
4	0.01	0.99	0.01	1
8	0.02	0.99	0.03	1
16	0.04	0.97	0.09	1
32	0.06	0.94	0.14	0.99
64	0.10	0.88	0.17	0.97
128	0.21	0.84	0.37	0.90

0% y 30% writes (RTLP)

Conclusiones

- Se ha estudiado en profundidad el problema de la gestión eficiente de threads en procesadores multi-core utilizados en máquinas de búsqueda para la Web.

- **La contribución principal de este trabajo es la propuesta de una estrategia llamada BP, la cual resuelve el problema de sincronización de transacciones de lectura y escritura utilizando paralelismo sincrónico por lotes (bulk-synchronous parallelism) a nivel de threads.**

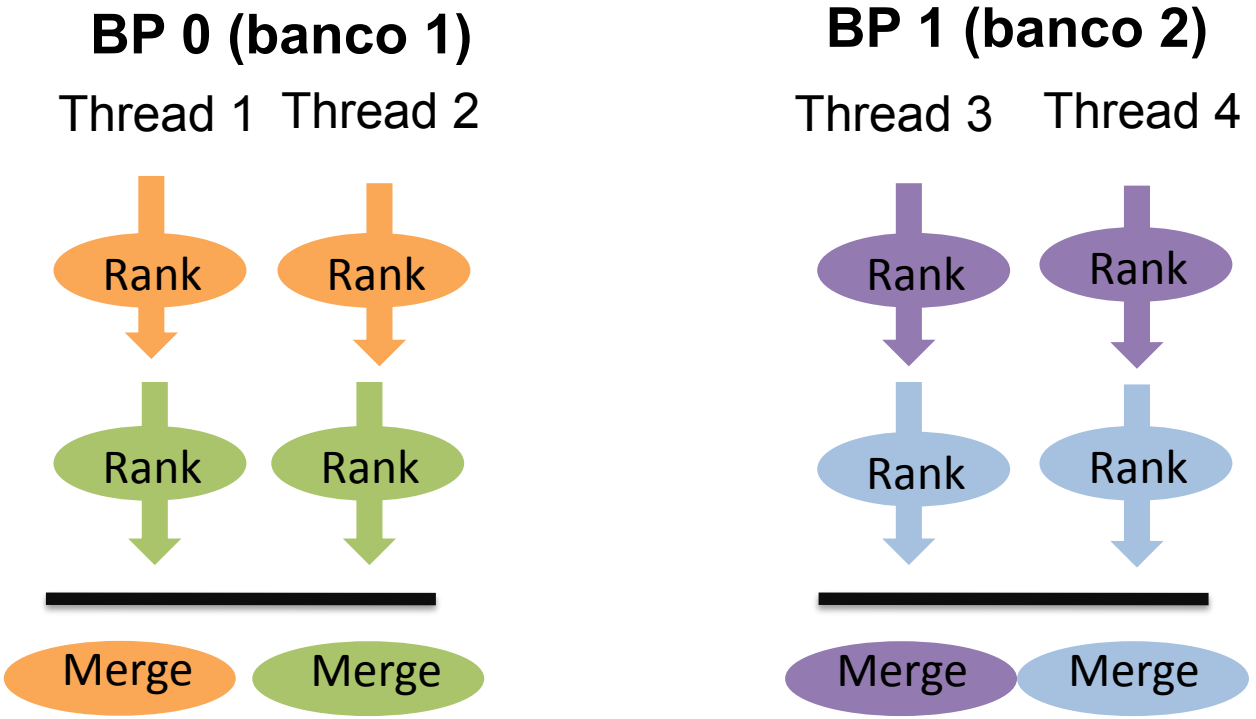
- BP es más eficiente tanto en throughput como en tiempo de respuesta, para transacciones individuales que para estrategias basadas en procesamiento asincrónico de transacciones.

- BP escala más eficientemente al aumentar el total de CPUs (cores) que las alternativas asincrónicas.

Trabajo futuro

Procesamiento de transacciones sobre procesadores con mecanismos de ahorro de energía

La idea es tener varias instancias de BP desplegadas en bancos de núcleos que pueden apagarse para ahorrar energía.



Trabajo futuro

Procesamiento de transacciones sobre procesadores con mecanismos de ahorro de energía

• Durante periodos de tráfico bajo, las transacciones son dirigidas a unas pocas instancias de BP y por lo tanto las otras instancias BP al no recibir tráfico ponen el banco de núcleos en modo de bajo consumo de energía.

• Dependiendo de la latencia asociada a hacer que un banco de núcleos vuelva a procesar transacciones, frente a una subida en tráfico de transacciones se pueden tener dos métodos de asignación de transacciones a las instancias de BP

- Simplemente comenzar enviar nuevas transacciones a las instancias de BP que están “dormidas”.

- Predecir con anticipación una subida brusca en el tráfico de transacciones y enviar transacciones sintéticas a las instancias de BP dormidas para ponerlas en funcionamiento lo antes posible.

Este trabajo ha dado lugar a las siguientes publicaciones

→"Building Efficient Multi-Threaded Search Nodes", In *19th ACM Conference on Information and Knowledge Management* (CIKM 2010), Toronto, Canada, Oct 26-30, ACM 2010.

→ "On-Line Multi-Threaded Processing of Web User-Clicks on Multi-Core Processors", In *9th International Meeting High Performance Computing for Computational Science* (VECPAR 2010), Berkeley, California, June 22-25, 2010, published in LNCS 6449, pp. 222-235, 2011 (Springer-Verlag).

→ "Exploiting Hybrid Parallelism in Web Search Engines", In *14th European International Conference on Parallel Processing* (Euro-Par 2008), Gran Canaria, Aug. 26-29, Spain, Lecture Notes in Computer Science 5168, pp. 414-423, Springer, 2008.

→ "Improving Search Engines Performance on Multithreading Processors", In *8th International Meeting on High Performance Computing for Computational Science* (VECPAR 2008), Revised Selected Papers, LNCS 5336, pp. 201-213, Toulouse, France, June 24-27, 2008.

→ "Comparative Study of Concurrency Control on Bulk-Synchronous Parallel Search Engines", In *International Conference on Parallel Computing 2007* (ParCo 2007), Aachen, Germany, Sep. 4-7, Advances in Parallel Computing Vol. 15, pp. 389-396, ISBN 978-1-58603-796-3, IO Press, 2007.

Gestión Eficiente de Transacciones en Máquinas de Búsqueda para la Web

Carolina Bonacic Castro

Directores: Manuel Prieto Matías, Carlos García Sánchez
Mauricio Marín.

Servicio de Índice (multi-threading)
Atomicidad y seriabilidad

Atomicidad

Thread lector -> lee $L(term_1)$

Thread escritor -> los términos de Doc_1 en las listas de $L(term_1)$ $L(term_2)$

Thread lector -> lee $L(term_2)$

Ranking final

Lista $L(term_1)$ **sin** Doc_1
Lista $L(term_2)$ **con** Doc_1

No debe ocurrir

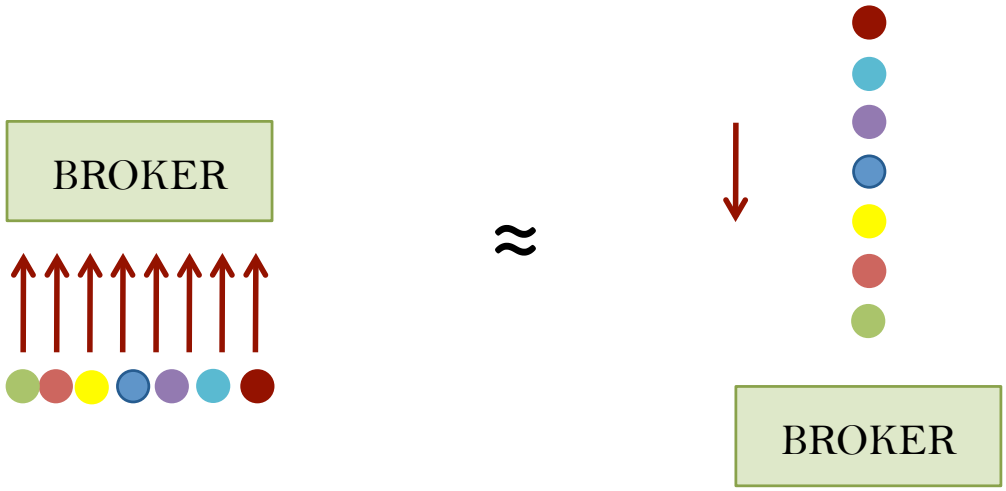
Si hay atomicidad los ranking posibles son:

Ranking **SIN** Doc_1 en todas las listas

Ranking **CON** Doc_1 en todas las listas

Servicio de Índice (multi-threading)
Atomicidad y seriabilidad

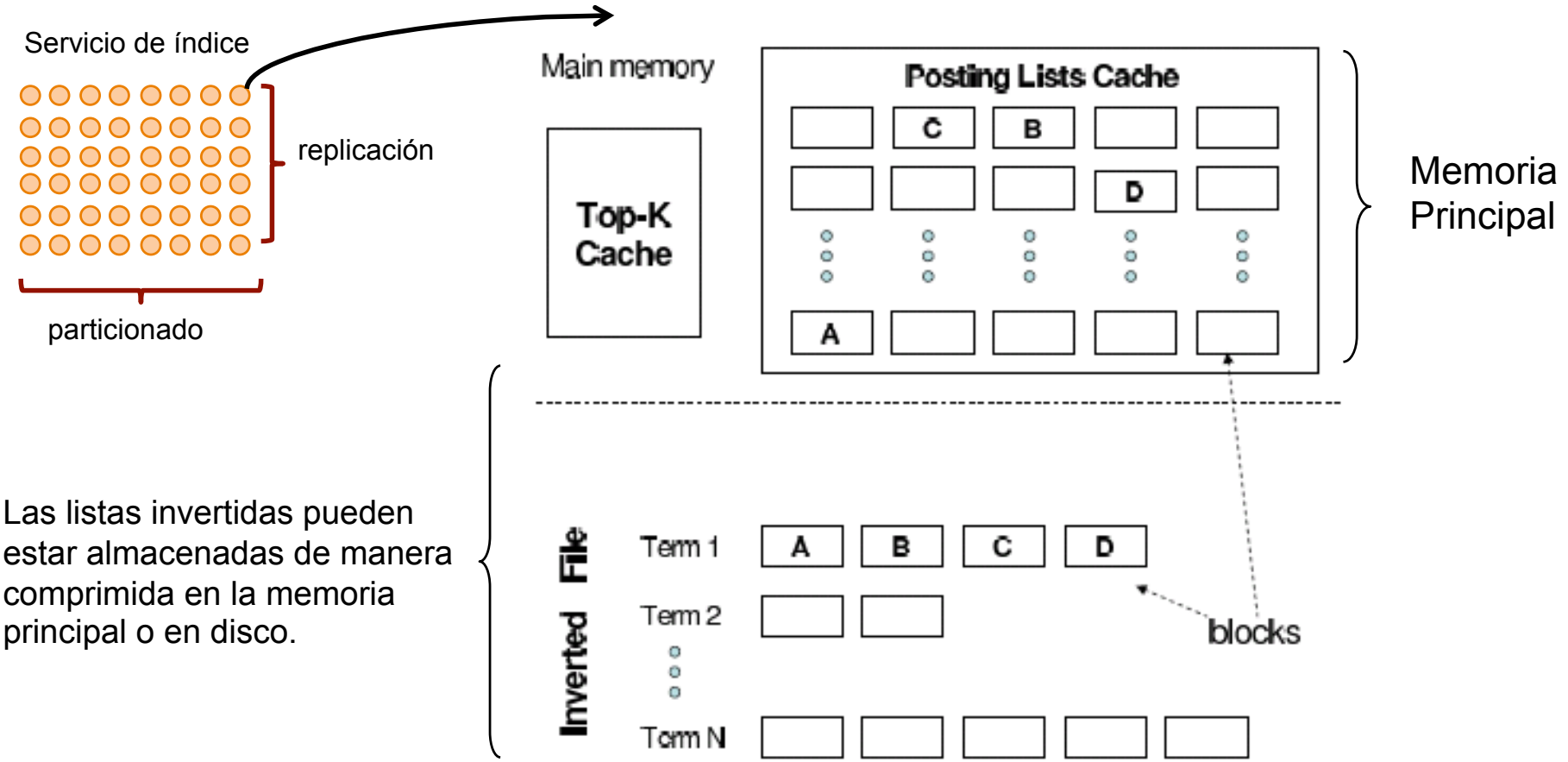
Serialidad



El resultado de la ejecución concurrente de varias transacciones sea equivalente a haberlas ejecutado de manera secuencial, una por una, respetando el orden de llegada al nodo.

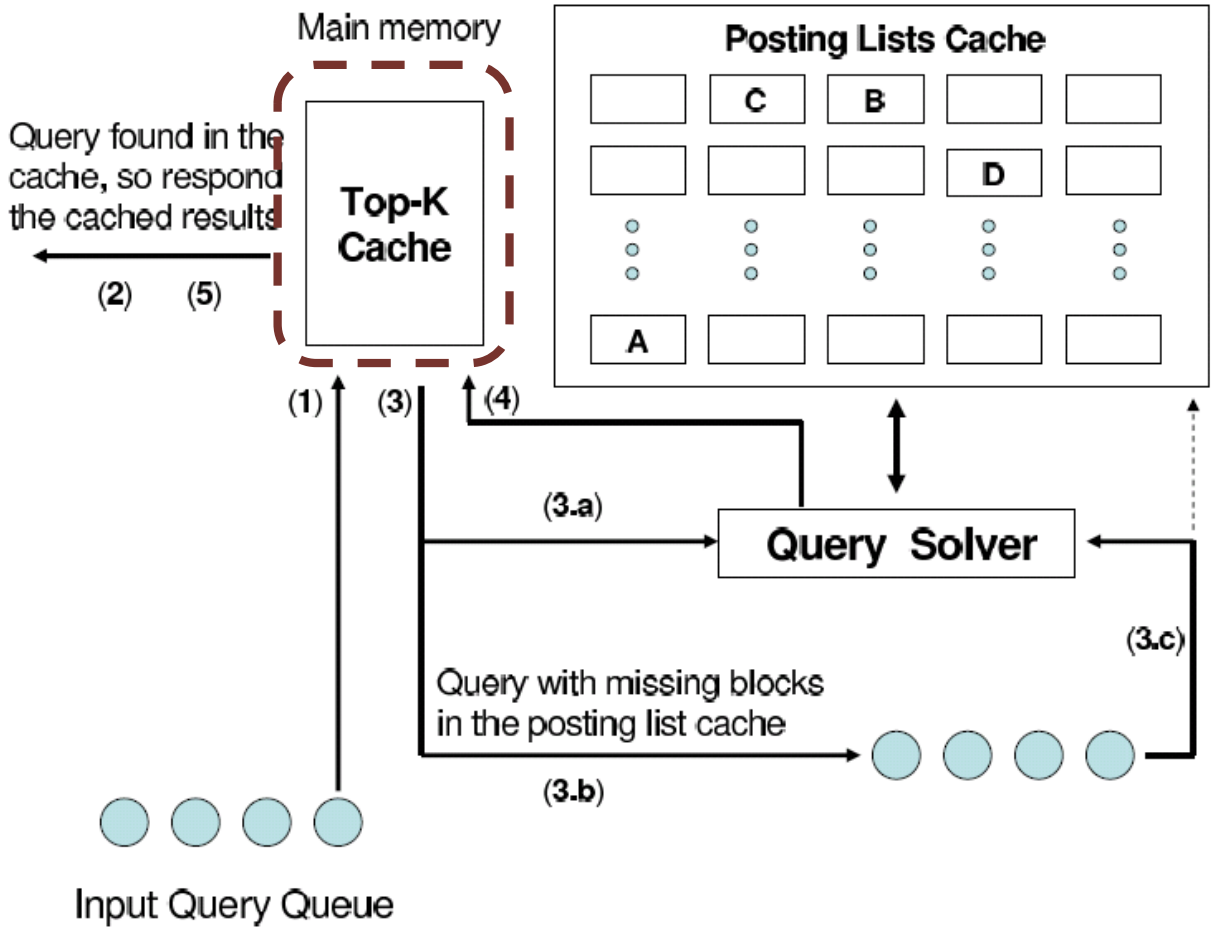
Planteamiento del Problema

Ejemplo de arquitectura de un nodo de búsqueda

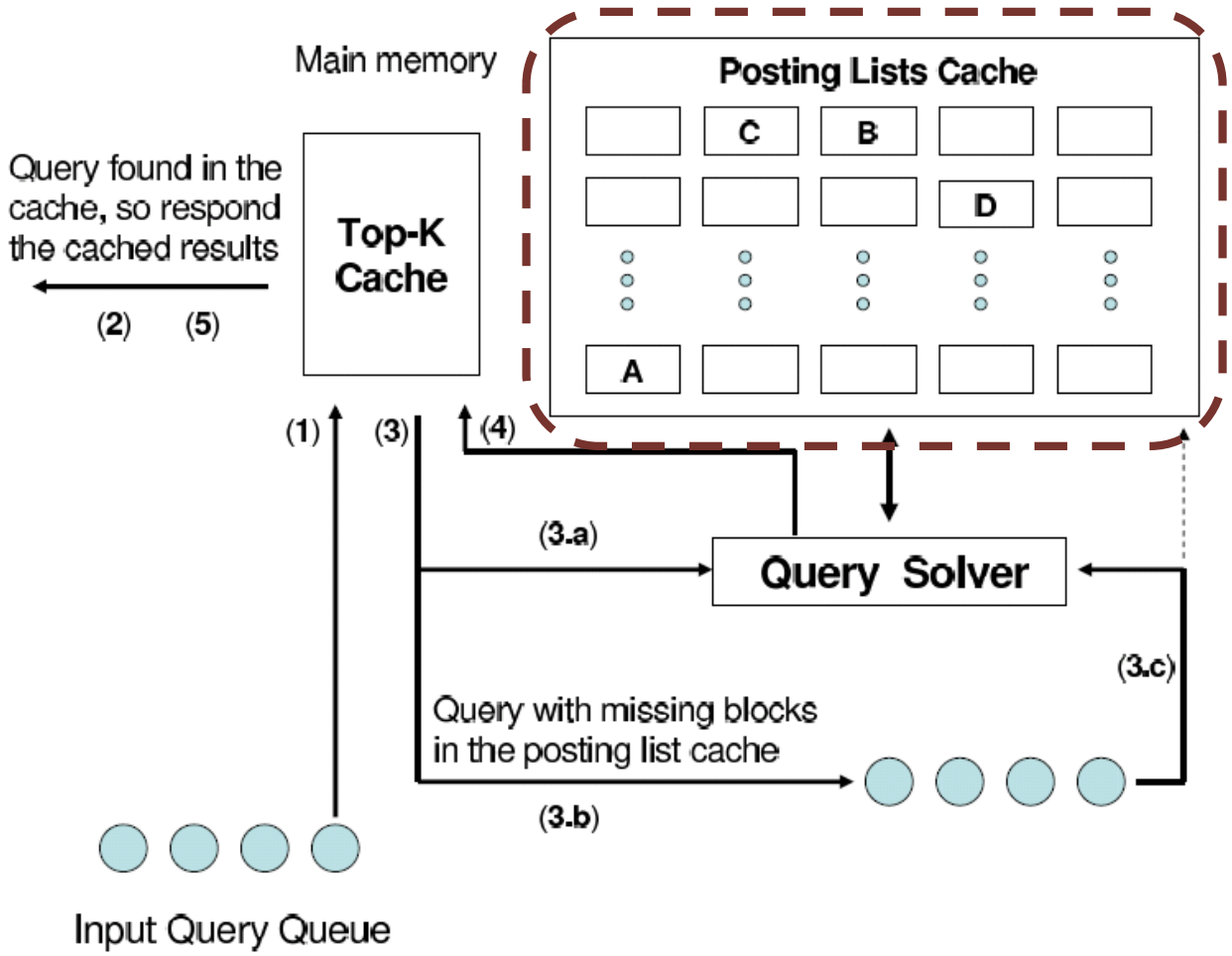


Las listas invertidas pueden estar almacenadas de manera comprimida en la memoria principal o en disco.

Planteamiento del Problema.
Top-K Cache



Planteamiento del Problema.
Posting list cache



Resumen de estrategias de procesamiento de transacciones

	Serial	Parallelism	Primitive
BP	Yes	PRT and PWT	Thread Barrier
CR (hybrid)	Yes	CRT and PWT	Thread Barrier
TLP1	Yes	CRT (term sharing) and CWT	R/W List Locking
TLP2	Yes	CRT (no sharing) and CWT	Exclusive List Locking
RTLP	No	non-atomic: CRT and CWT	Term Locking
RBLP	No	non-atomic: CRT and CWT	Block Locking

CRT = concurrent read transactions PRT = read transaction in parallel
CWT = concurrent write transactions PWT = write transaction in parallel

RTLP-RB

Para cada término del índice invertido se mantiene información sobre el timestamp del último lector o escritor que lo utilizó.

- T_i es una transacción que comienza en el instante $T_s(i)$
- Q es un término del índice invertido.
- $W_{TS}(Q)$: Máximo de los timestamps de las transacciones que han hecho write(Q).
- $R_{TS}(Q)$: Máximo de los timestamps de las transacciones que han hecho read(Q).

El protocolo es:

- Si la transacción T_i intenta ejecutar un **read**(Q)
  - Si $T_s(i) < W_{TS}(Q)$ no se ejecuta la operación y T_i se re-ejecuta.
  - Si $T_s(i) > W_{TS}(Q)$ se ejecuta la operación y $R_{TS}(Q) = max\{R_{TS}(Q), T_s(i)\})$.
- Si la transacción T_i intenta ejecutar un **write**(Q)
  - Si $T_s(i) < R_{TS}(Q)$ no se ejecuta la operación y T_i se re-ejecuta.
  - Si $T_s(i) < W_{TS}(Q)$ no se ejecuta la operación y T_i continúa.
  - Si no, sí se ejecuta la operación y $W_{TS}(Q) = T_s(i)$.

La cola de prioridad propuesta para gestionar el cache de listas explota la existencia de secuencias contiguas de entradas de cache conteniendo la misma prioridad.