Implementación de LSM-Trees en PostgreSQL

Patricio Iribarne Catella
Gonzalo L. Petraglia

Tutoría: Dr. Mariano Beiró

FIUBA

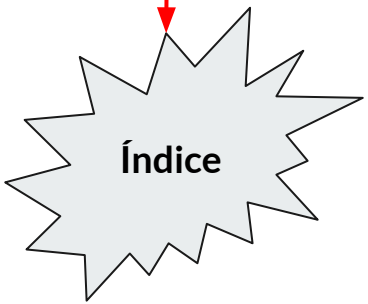
Agenda

- Problema general
- LSM-Trees
- PostgreSQL
- LSM-Trees en PostgreSQL
- Benchmarks y Resultados

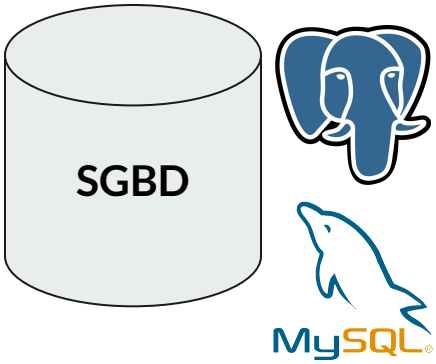
Problema general

Problema general

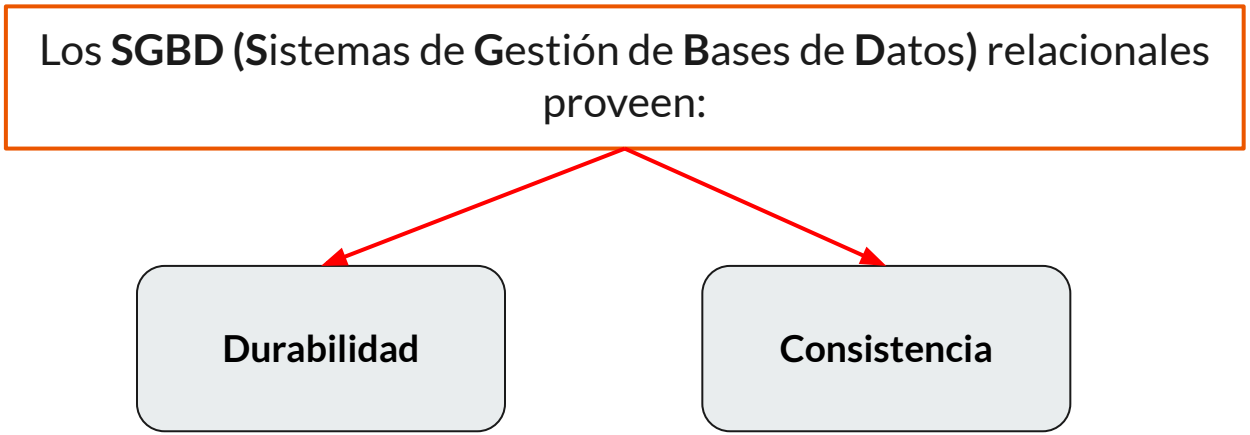
Información que queremos **organizar** para realizar búsquedas **eficientemente**



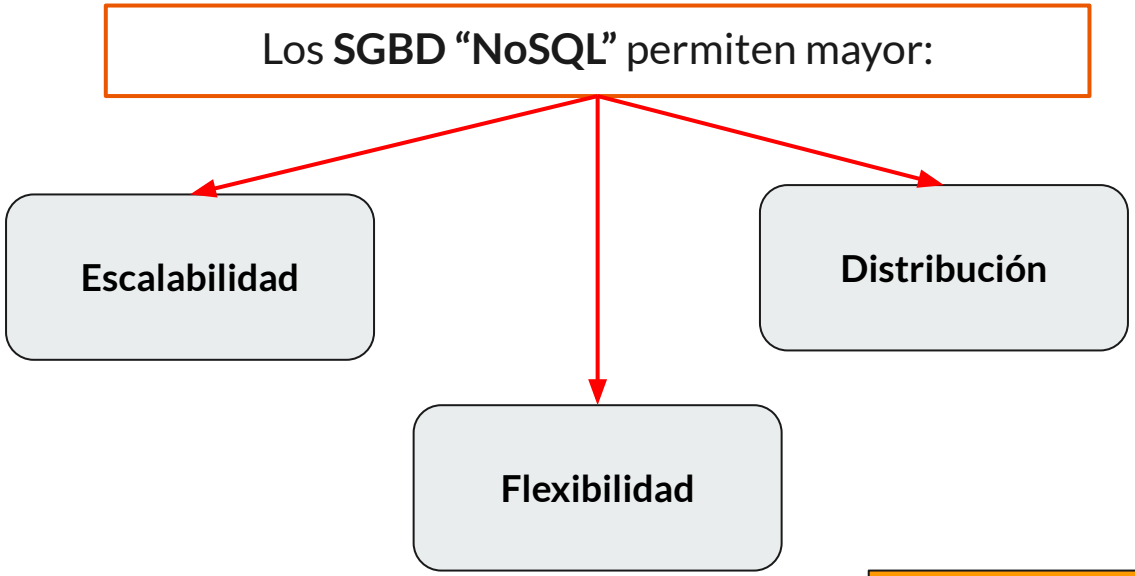
Problema general



Problema general



Problema general



Relajan restricciones de **consistencia** y **durabilidad**

LSM-Trees

LSM-Trees

La estructura protagónica es el **LSM-Tree: L**og-**S**tructured **M**erge-Tree.

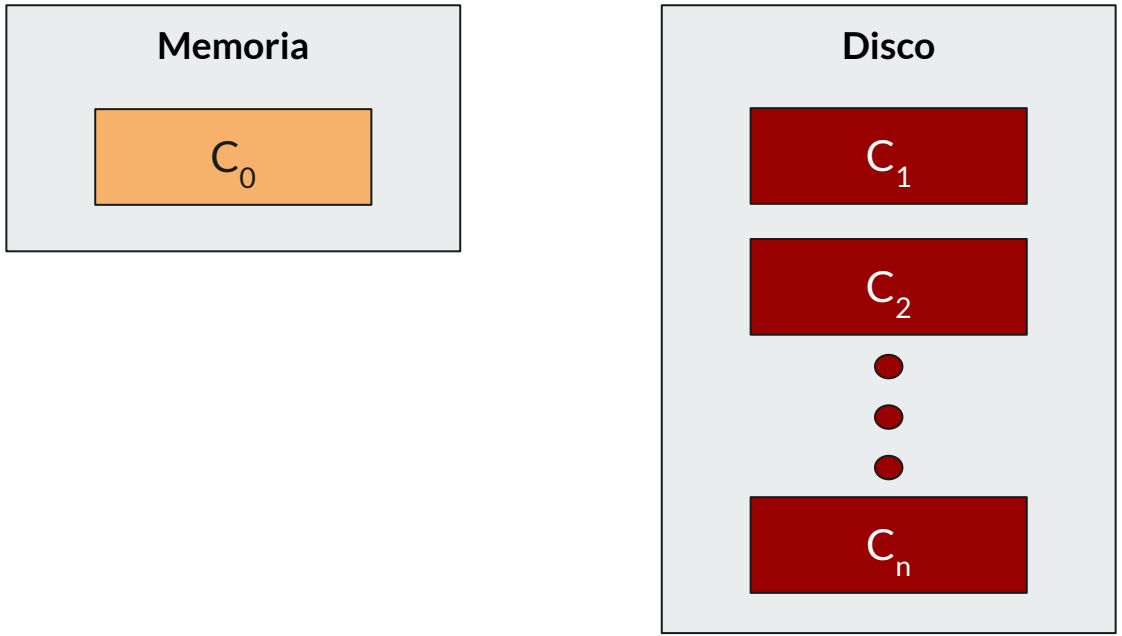
Presentada por primera vez en el *paper* de Patrick O'Neil [1]

La misma consta de subestructuras que residen tanto en **memoria** como en **disco.**

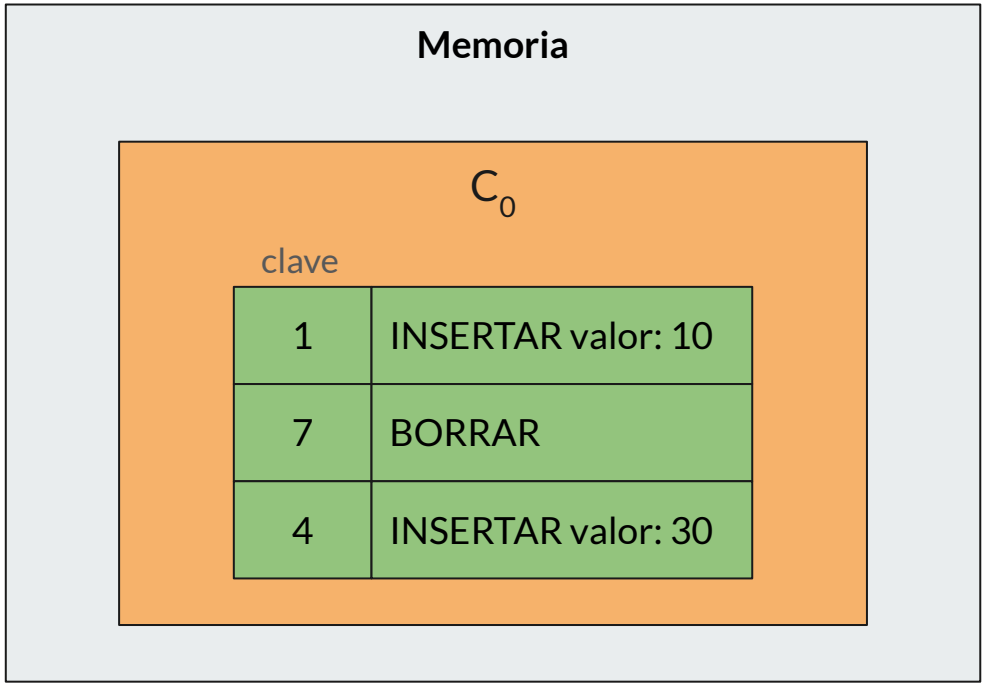
Busca priorizar la **velocidad** de **escritura** por sobre la de lectura.

1 O'Neil, P., Cheng, E., Gawlick, D., & O'Neil, E. (1996). The log-structured merge-tree (LSM-tree). *Acta Informatica*, *33*, 351-385.

LSM-Trees



LSM-Trees



- Usualmente se utiliza una estructura de **árbol.**
- Está completamente en memoria.
- Cada inserción realiza una modificación para un mismo valor de ***clave***.
- Tiene un **tamaño máximo** definido.

LSM-Trees

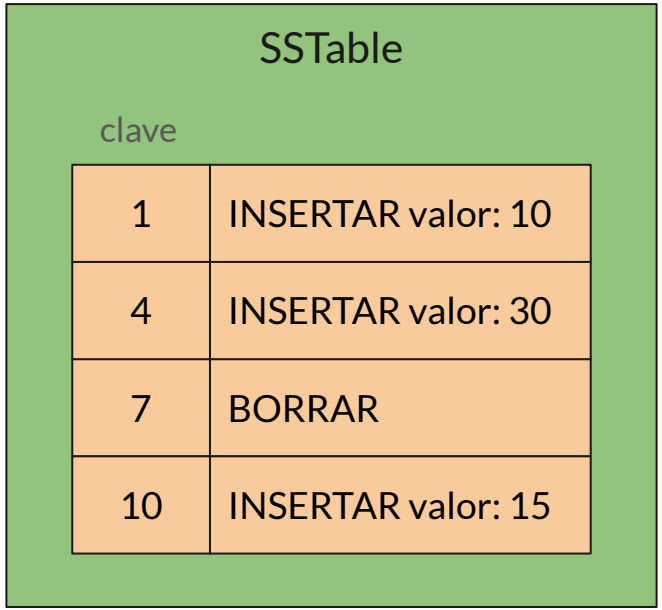
Ventajas:

- Acelerar escrituras → se resuelve sin tener que escribir en disco.
- Aprovechar el principio de localidad temporal → evita que tuplas insertadas recientemente se tengan que leer o actualizar en el disco.

LSM-Trees

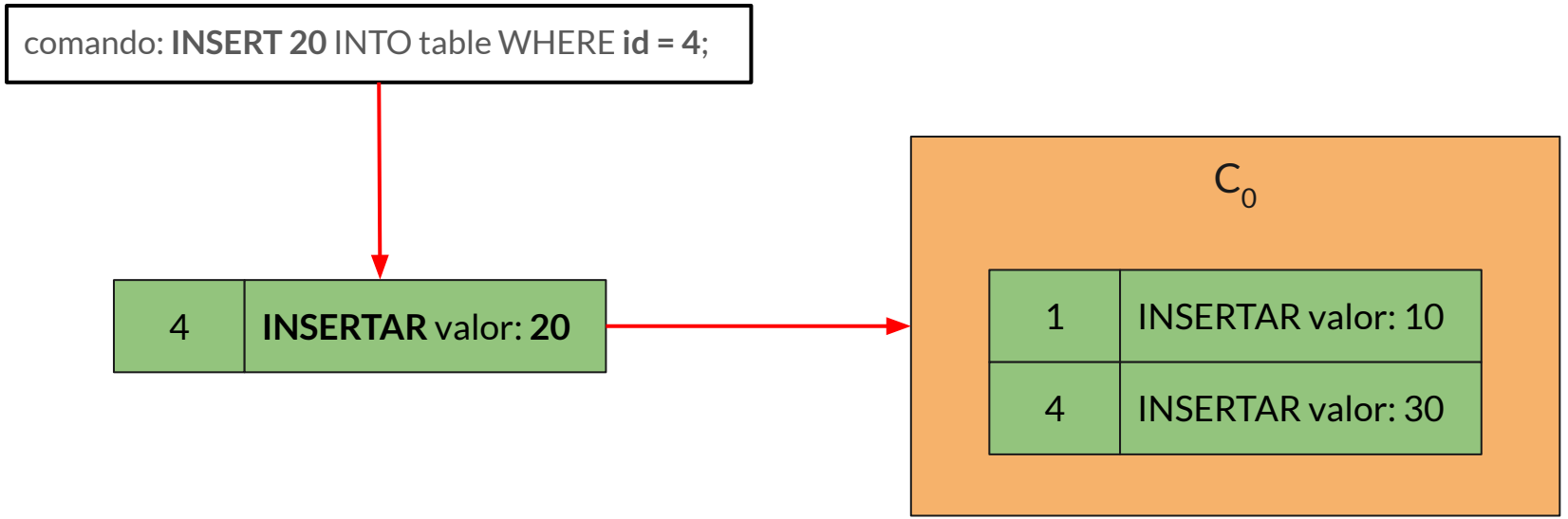
- Cada nivel consta de 1 o más *SSTables*.
- Los niveles bajos, contienen tuplas más recientes.
- Cuando se *llena* un nivel, se **compacta** y se envía al siguiente.

LSM-Trees

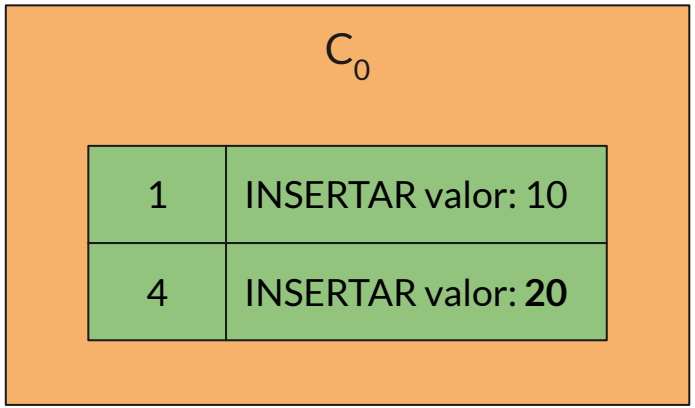


- **S**orted **S**trings **T**able
- Almacenadas en *disco*.
- Tuplas ordenadas por su *clave*.
- Son **inmutables**, es decir, nunca se modifican. Solo se *compactan* y se crean nuevas SSTables.
- Representan el estado de las tuplas en un momento del tiempo.

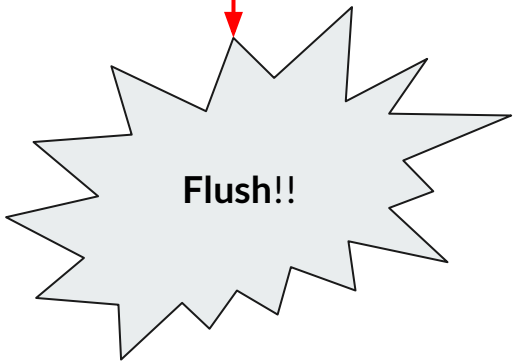
LSM-Trees (escritura)



LSM-Trees (escritura)

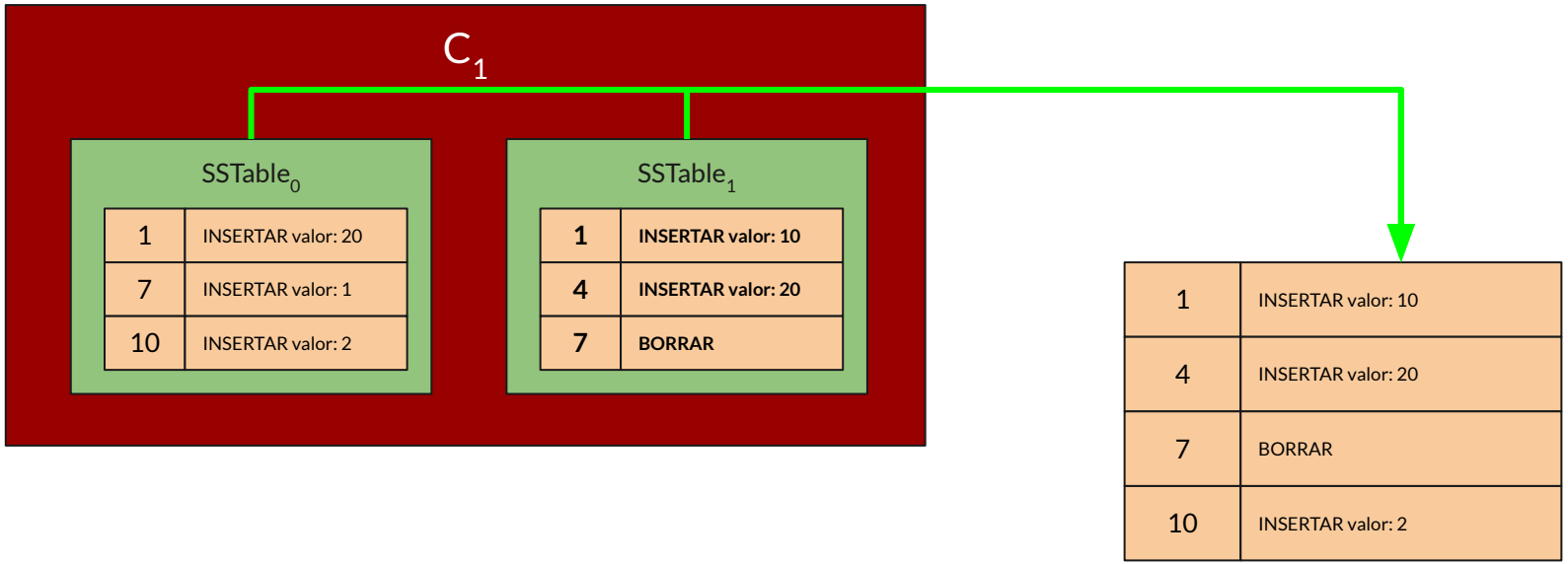


¿Qué ocurre si el **tamaño máximo** es alcanzado?

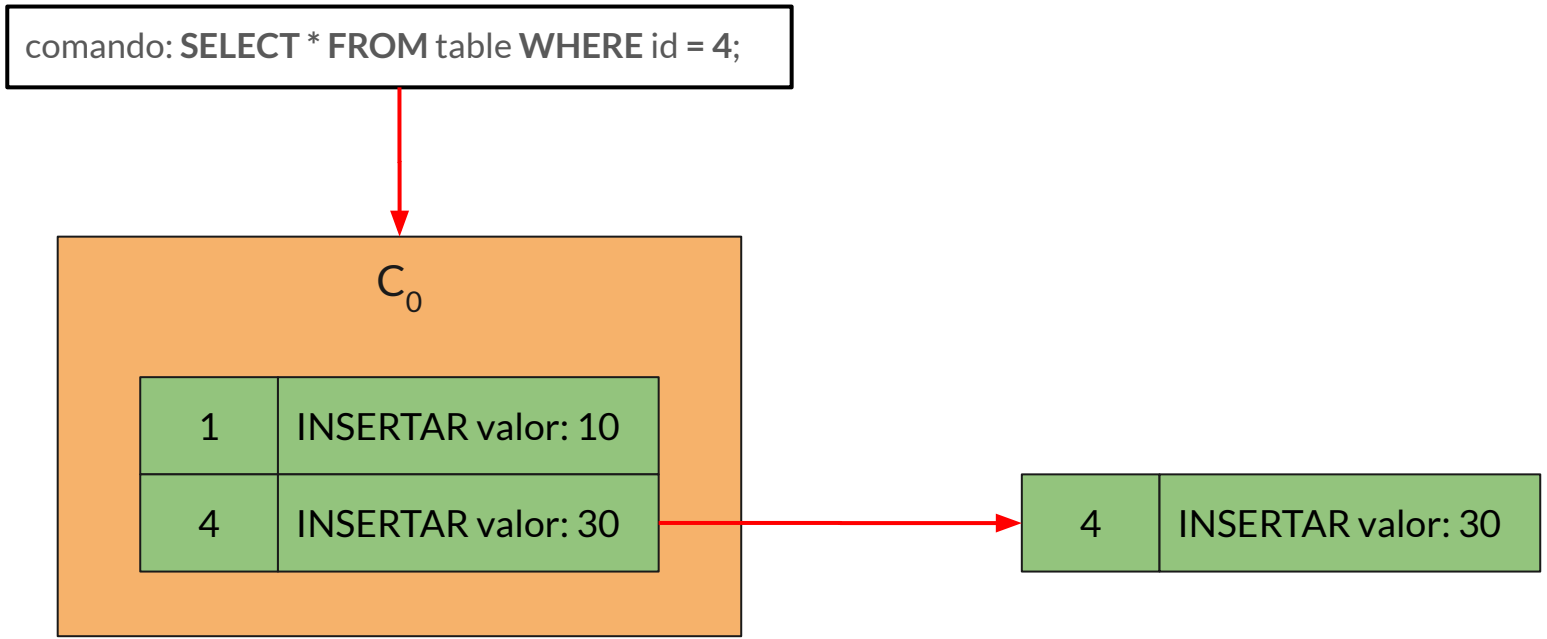


LSM-Trees (escritura → merging)

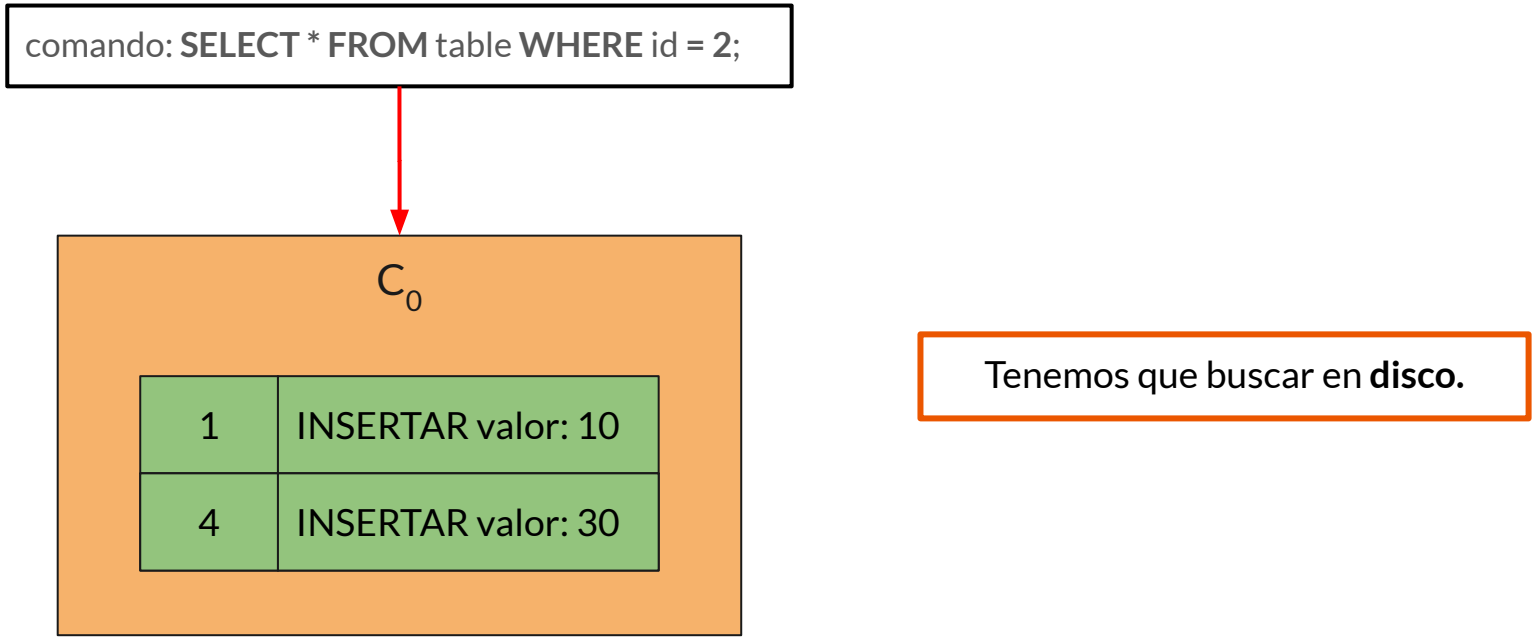
LSM-Trees (escritura → merging)



LSM-Trees (lectura → existe en Memoria)

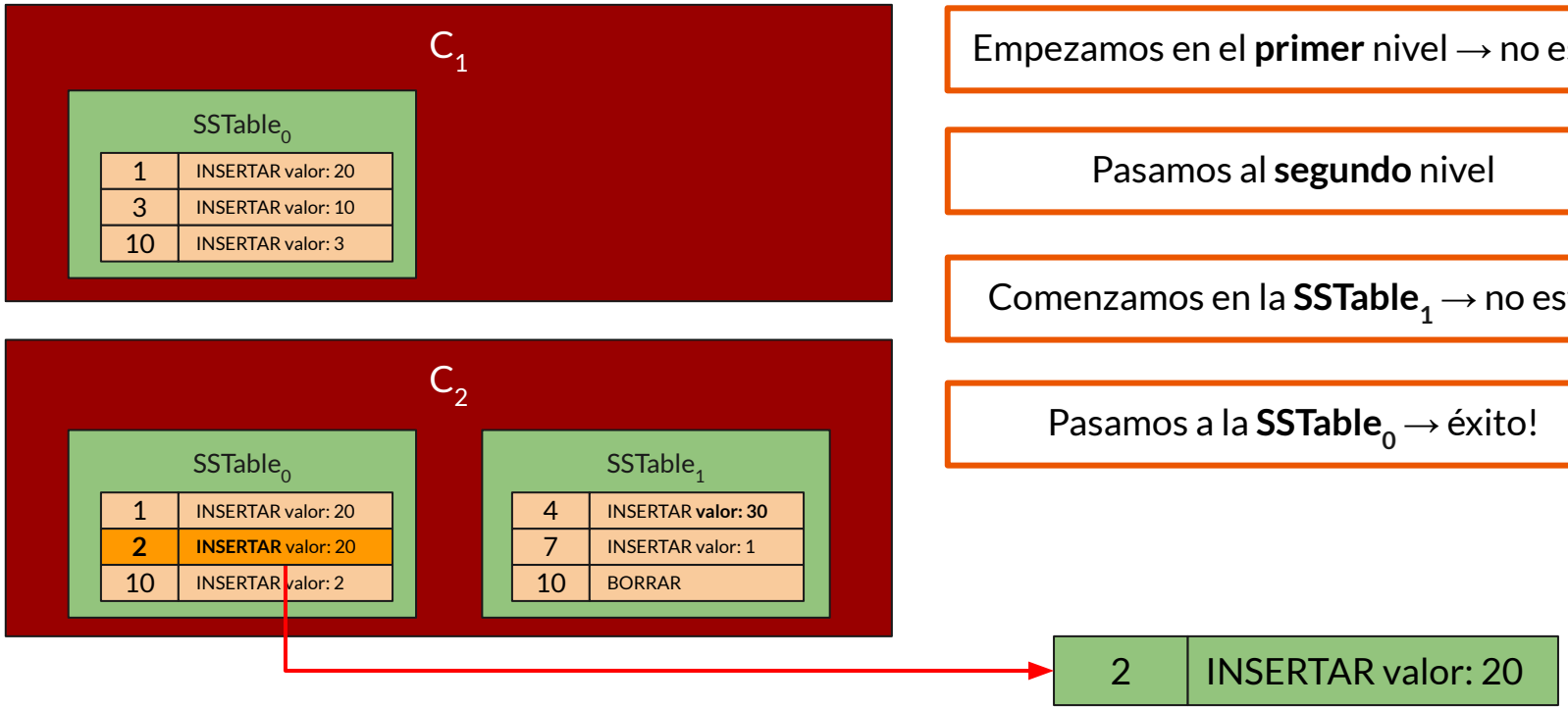


LSM-Trees (lectura → **no** existe en Memoria)



Tenemos que buscar en **disco.**

LSM-Trees (lectura → **no** existe en Memoria)



Empezamos en el **primer** nivel → no está

Pasamos al **segundo** nivel

Comenzamos en la **$SSTable_1$** → no está

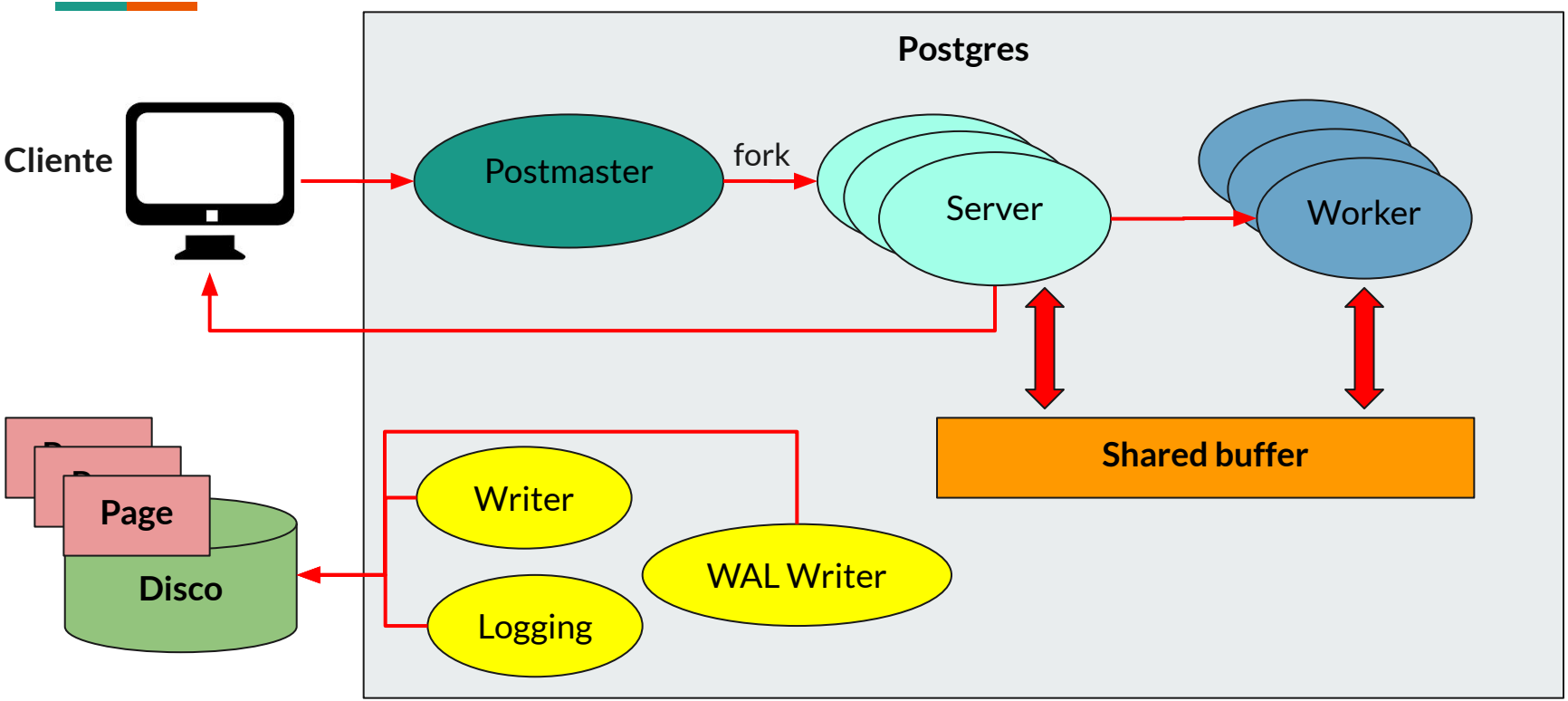
Pasamos a la **$SSTable_0$** → éxito!

PostgreSQL

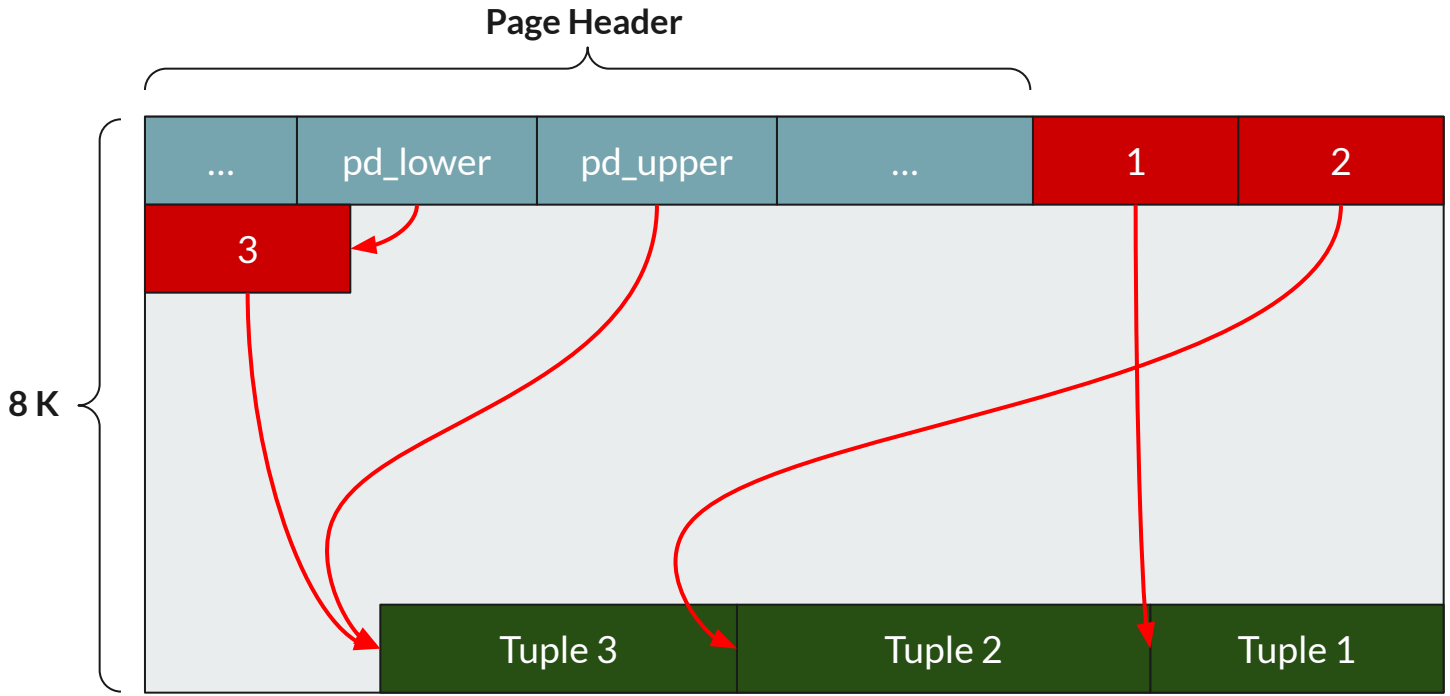
PostgreSQL

- Base de datos relacional
- 35 años de desarrollo activo (open source)
- Creado en la universidad de *Berkeley*
- Soporta restricciones **ACID**
- **Durabilidad** y **consistencia**
- **Extensibilidad** por medio de *módulos*.
- Diferentes *tipos* de **índices**.

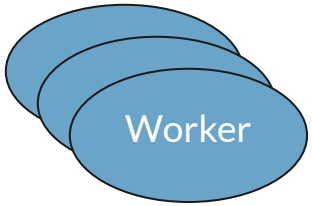
PostgreSQL (arquitectura)



PostgreSQL (páginas)



PostgreSQL (background workers)



- Existen de dos tipos: **Estáticos** y **Dinámicos**
- **Estáticos:** se configuran (*registran*) para lanzarse al comienzo cuando se inicia el motor.
- **Dinámicos:** se pueden *lanzar* en cualquier momento.
  - muy útiles para procesos *asíncronos* cuya terminación no desea ser *bloqueante*.
- Tienen acceso a cualquier tabla (*relación*) o índices, y operaciones dentro de Postgres.
- Admiten comunicación mediante *memoria compartida* y *colas de mensajes* para coordinarse entre sí.

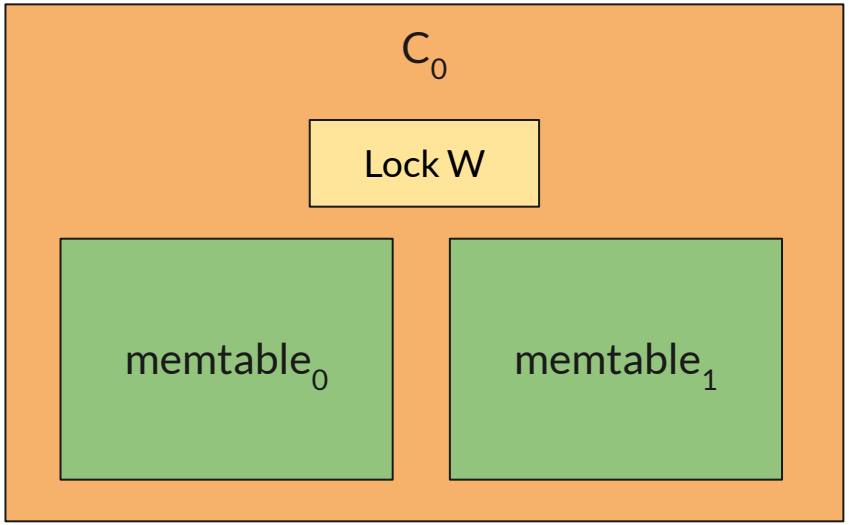
LSM-Trees en PostgreSQL

LSM-Trees en Postgres

Características principales:

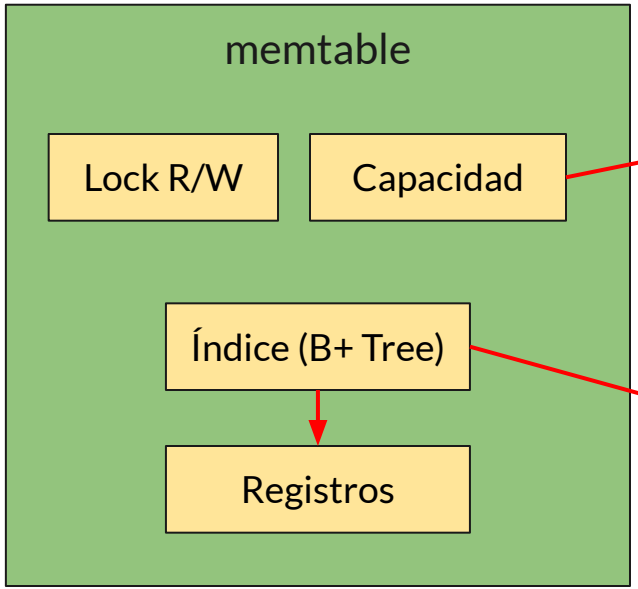
- Garantiza **durabilidad**
- Permite la **concurrencia**
- No puede garantizar **unicidad**
- Prioriza la **escritura sobre** la **lectura**
- Utiliza estructura y abstracciones internas de **Postgres**

LSM-Trees en Postgres (memtables)



- Consiste de dos *réplicas* de iguales características.
- Ambas manejan *búsquedas* e *inserciones*.
- **Ventaja:** permite realizar inserciones mientras se realiza el proceso de *flushing*.

LSM-Trees en Postgres (memtables)

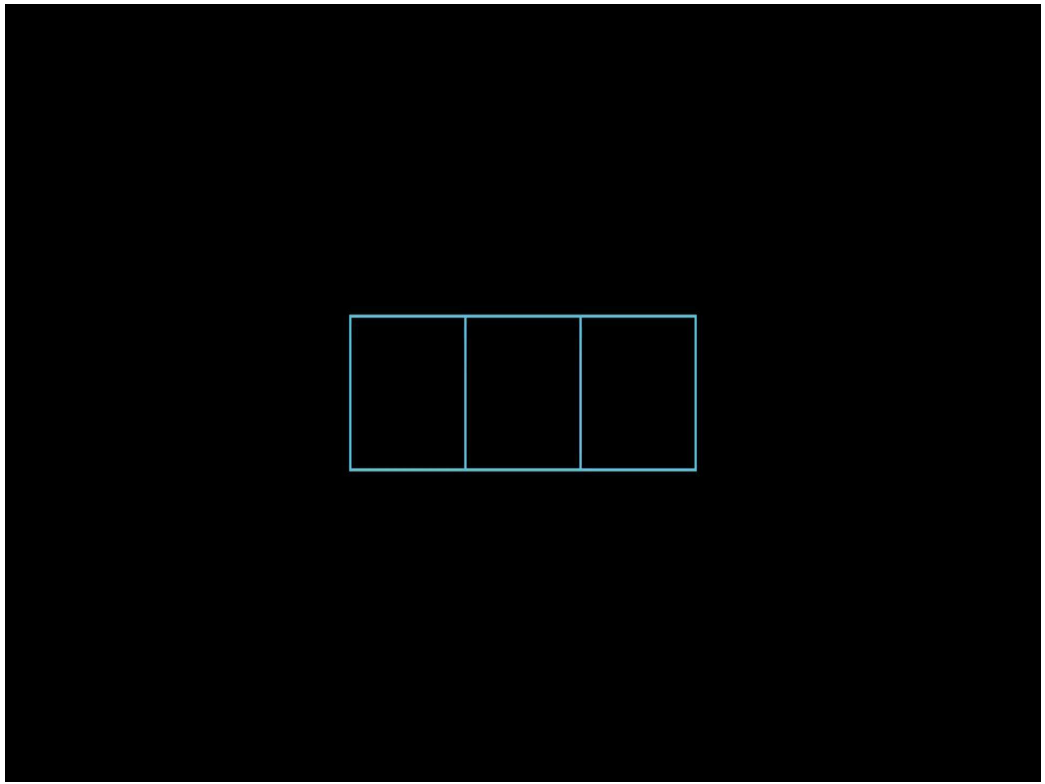


Cuando es alcanzada, se procede a realizar el proceso de **flushing** → ya no se pueden realizar *escrituras*.

Permite:

- búsquedas/inserciones eficientes
- obtener las claves ordenadas en *O(n)* para **flushing.**
- utilizar cache de CPU

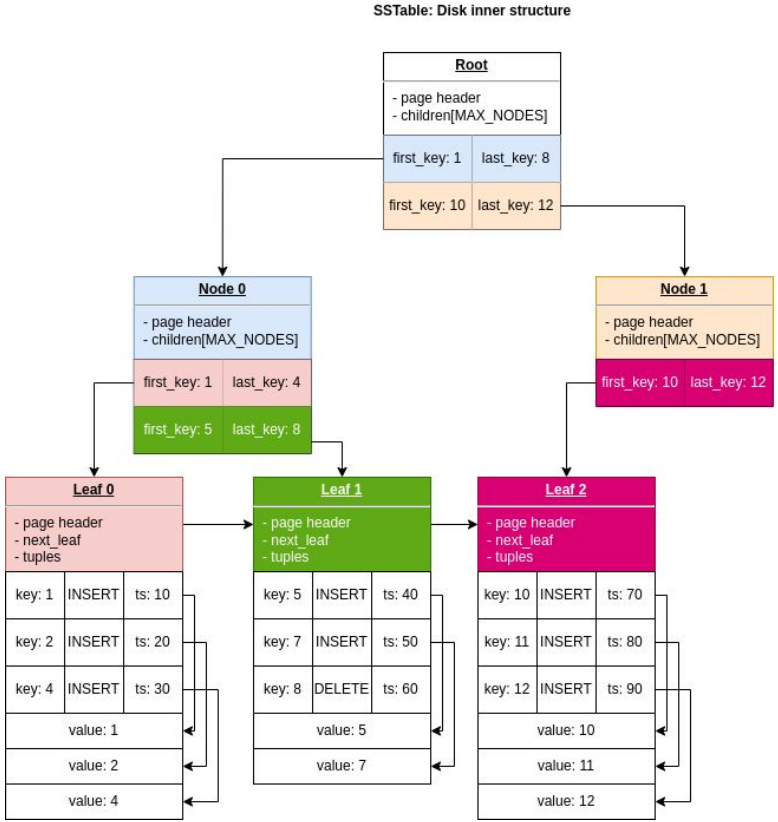
LSM-Trees en Postgres (B+Tree)



LSM-Trees en Postgres (SSTables)

- Cada **SSTable** consiste de un árbol B+ Tree inmutable
- Cada *nodo* es un *bloque* (página) de Postgres
- Posee tres niveles:
  - Raíz
  - Nivel intermedio
  - Hojas
- Todo *nodo* posee
  - una cantidad de *hijos* fija
  - un *enlace* al nodo *hermano*

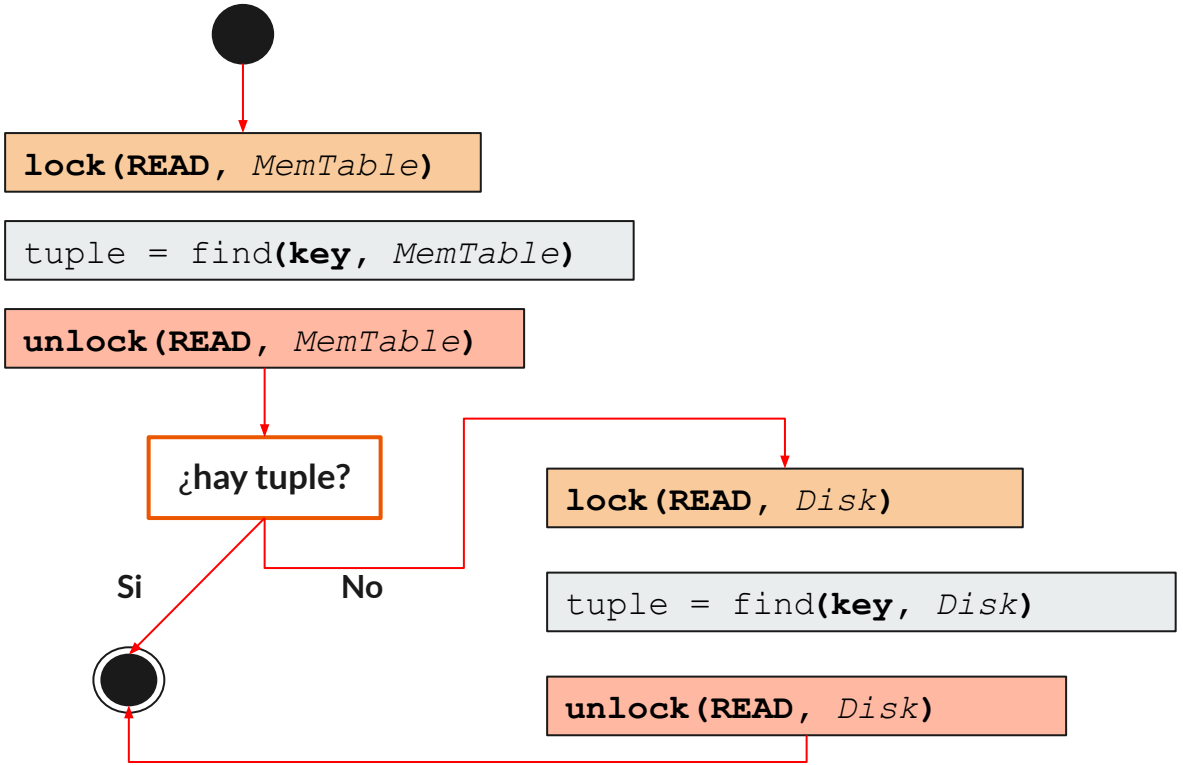
LSM-Trees en Postgres (SSTables)



LSM-Trees en Postgres (Merging)

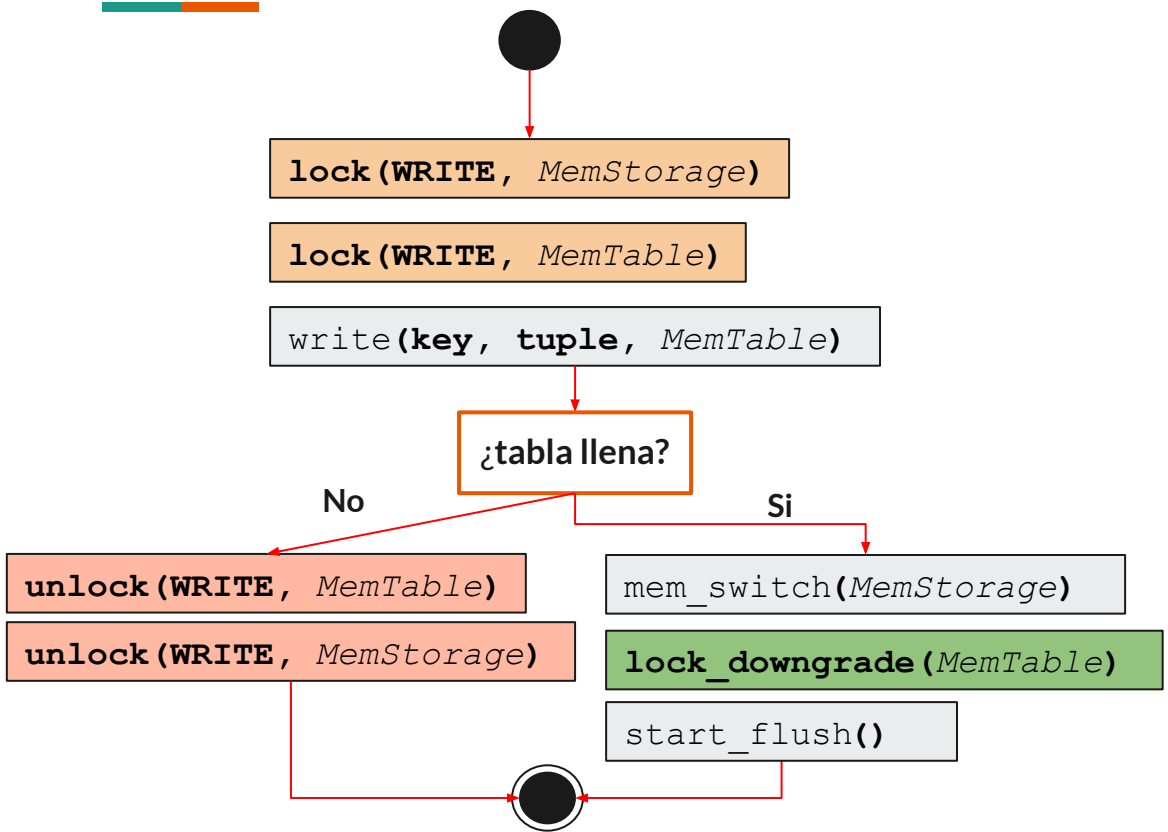
LSM-Trees en Postgres (Concurrencia)

Lectura



LSM-Trees en Postgres (Concurrencia)

Escritura



Flush worker (**background worker**)

- lock(WRITE, *Disk*)
- lock_upgrade(*MemTable*)
- reset(*MemTable*)
- unlock(WRITE, *MemTable*)
- flush(*MemTable*)
- unlock(WRITE, *MemStorage*)
- unlock(WRITE, *Disk*)

LSM-Trees en Postgres (Concurrencia)

El diseño del **lock** para la estructura en **memoria** también tuvo una consideración respecto a la **priorización de la escritura por sobre la lectura** (el proceso de *merge* de C_0 es englobado dentro de esta última categoría).

Es por esto que se eligió un **lock** que ante la *contención* (más de un proceso compitiendo por el lock) de **escritura** vs. **lectura**, el primero sea priorizado.

Benchmarks y Resultados

Benchmarks y Resultados

- **PGBench** como motor de pruebas
- Dos tipos de pruebas:
  - Escritura
  - Lectura

Benchmarks y Resultados (Lectura)

- Setup: Base de datos con 1.000.000 tuplas con claves aleatorias de 0 a 1.000.000
- **1.000.000 de tuplas buscadas** con una clave de 0 a 1.000.000

Benchmarks y Resultados (Escritura)

- Setup: Base de datos vacía
- **1.000.000** de tuplas insertadas
- Cada tupla es un número aleatorio de 0 a 100.000.000

Benchmarks y Resultados (Hardware)

Ejecutada en dos hardware distintos

	Hardware 1	Hardware 2
CPU	i7 11th Gen	i3 10th Gen
RAM	32 GB	8 GB
Tecnología de disco	SSD	HDD

Benchmarks y Resultados (Escritura - HDD)

Index Type	Input TPS	Output (real) TPS	Avg. Latency (ms)	Stddev Latency (ms)
LSM	500	499.3	33.26	43.20
B-Tree	500	499.0	27.53	36.55
LSM	700	700.5	65.10	87.04
B-Tree	700	700.00	133.5	814.94
LSM	900	836.9	72220	24630
B-Tree	900	900.2	38.81	47.38
LSM	1500	932.3	209200	115400
B-Tree	1500	1384	34110	15160

Benchmarks y Resultados (Lectura - HDD)

Index Type	Input TPS	Output (real) TPS	Avg. Latency (ms)	Stddev Latency (ms)
LSM	500	499.3	10.36	16.24
B-Tree	500	499.9	27.74	27.32
LSM	700	699.6	15.30	50.83
B-Tree	700	701.2	37.41	33.81
LSM	900	897.7	18.11	46.05
B-Tree	900	901.0	44.86	49.22
LSM	1500	1191	84260	50160
B-Tree	1500	1391	23780	12030

Benchmarks y Resultados (Análisis)

Cuando el server satura:

- Está constantemente resolviendo transacciones → La demora para resolver las transacciones es de 1/outputTPS, lo llamamos b
- Las transacciones se envían cada 1/inputTPS (que es menor que el tiempo de resolución), lo llamamos a
- Encolamiento de transacciones (n es la cantidad total de transacciones)
- **Las transacciones encoladas también suman a la latencia promedio**

Con un modelo basado en los puntos anteriores podemos predecir la latencia promedio como

$$avgLatency = \frac{(b-a)n}{2}$$

Benchmarks y Resultados (Escritura - SSD)

Index Type	Input TPS	Output (Real) TPS	Avg. Latency (ms)	Stddev Latency (ms)
LSM	700	699.9	3.155	10.99
B-Tree	700	700.4	14.15	40.83
LSM	1300	1301	2.991	11.15
B-Tree	1300	1300	10.5	33.01
LSM	3000	3009	3.869	13.43
B-Tree	3000	2998	3.297	15.81
LSM	7000	3489	71720	41610
B-Tree	7000	4741	33780	19630

Benchmarks y Resultados (Lectura - SSD)

Index Type	Input TPS	Output (Real) TPS	Avg. Latency (ms)	Stddev Latency (ms)
LSM	700	701.2	11.61	42.07
B-Tree	700	699.5	39.38	68.58
LSM	1300	1300	3.781	20.10
B-Tree	1300	1298	45.20	71.86
LSM	3000	2996	3.817	20.21
B-Tree	3000	3000	15.66	49.41
LSM	7000	6404	6032	3896
B-Tree	7000	7010	10.30	39.36

Benchmarks y Resultados (Resumen y análisis)

- Mejores métricas (menor latencia) del índice LSM siempre y cuando no esté saturado
- Satura más rápido (menor throughput)

El menor throughput se puede entender por una consumo de IOPS mucho mayor (30x) y la contradicción se puede entender por el trabajo asincrónico que se hace con los datos

Trabajo futuro

Trabajo futuro

- Implementar borrado
- Permitir tamaños configurables para las memtable
- Permitir mayor cantidad de datos por SSTable
- Utilizar otros benchmarks:
  - TPC
  - Mixtos

¡Gracias! ¿Preguntas?