LevelDB-Study

Team_Cache Benchmark Experiment

Made by Subin Hong, Seungwon Ha

E-Mail: zed6740@dankook.ac.kr, 12gktmddnjs@naver.com

DANKOOK UNIVERSITY

Dankook University
System Software Laboratory

Contents

- ## Cache Definition

- What is Cache

- ## LRU Cache Structure in LevelDB

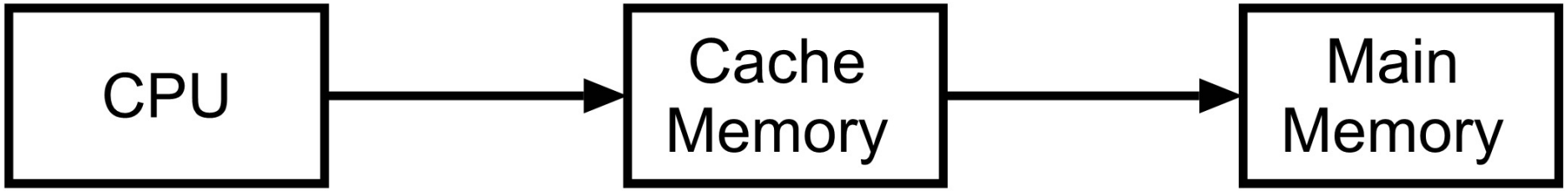
- What is LRU, analysis LRU structure in leveldb

- ## Benchmarks Experiment

- Hypothesis, Design, Run Experiment, Result and Discussion

Cache Definition

- What is Cache



Cache Memory Diagram

Cache Definition

- What is Cache

Cache	vs	Main Memory
+ Fast access		- Slow access
- High cost		+ Cheap cost
- Small capacity		+ Big capacity

LRU Cache Structure in LevelDB

-What is LRU

LRU(Least Recently Used)

Access	Hit/Miss	Evict	Cache state
0	Miss		0

LRU Cache Structure in LevelDB

-What is LRU

LRU(Least Recently Used)

Access	Hit/Miss	Evict	Cache state
0	Miss		0
1	Miss		0,1

LRU Cache Structure in LevelDB

-What is LRU

LRU(Least Recently Used)

Access	Hit/Miss	Evict	Cache state
0	Miss		0
1	Miss		0,1
2	Miss		0,1,2

LRU Cache Structure in LevelDB

-What is LRU

LRU(Least Recently Used)

Access	Hit/Miss	Evict	Cache state
0	Miss		0
1	Miss		0,1
2	Miss		0,1,2
0	Hit		1,2,0

LRU Cache Structure in LevelDB

-What is LRU

LRU(Least Recently Used)

Access	Hit/Miss	Evict	Cache state
0	Miss		0
1	Miss		0,1
2	Miss		0,1,2
0	Hit		1,2,0
1	Hit		2,0,1

LRU Cache Structure in LevelDB

-What is LRU

LRU(Least Recently Used)

Access	Hit/Miss	Evict	Cache state
0	Miss		0
1	Miss		0,1
2	Miss		0,1,2
0	Hit		1,2,0
1	Hit		2,0,1
3	Miss	2	0,1,3

Least Recently Used…. (2)

LRU Cache Structure in LevelDB

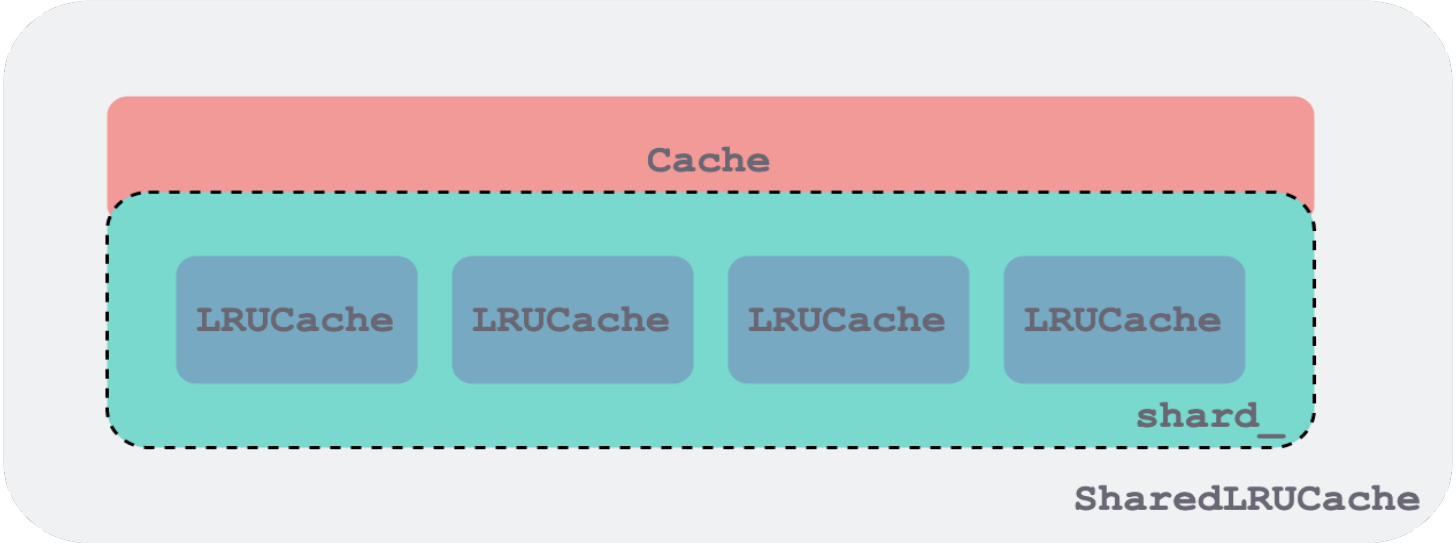
-What is LRU

LRU(Least Recently Used)

Access	Hit/Miss	Evict	Cache state
0	Miss		0
1	Miss		0,1
2	Miss		0,1,2
0	Hit		1,2,0
1	Hit		2,0,1
3	Miss	2	0,1,3
0	Hit		1,3,0

LRU Cache Structure in LevelDB

- sharding in LevelDB



LRU Cache Structure in LevelDB

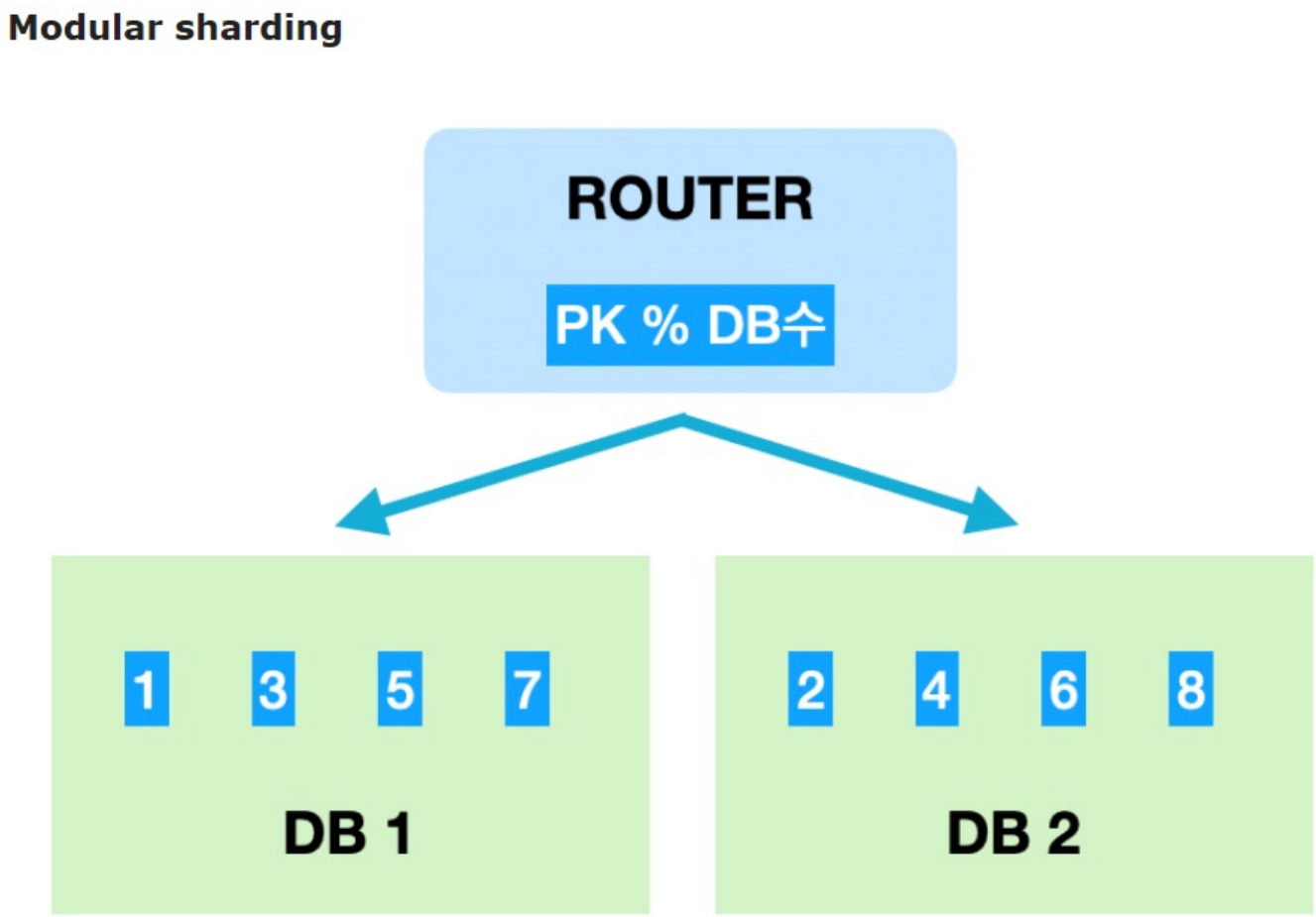
- sharding

What is sharding?

Separate the data because of the capacity issue

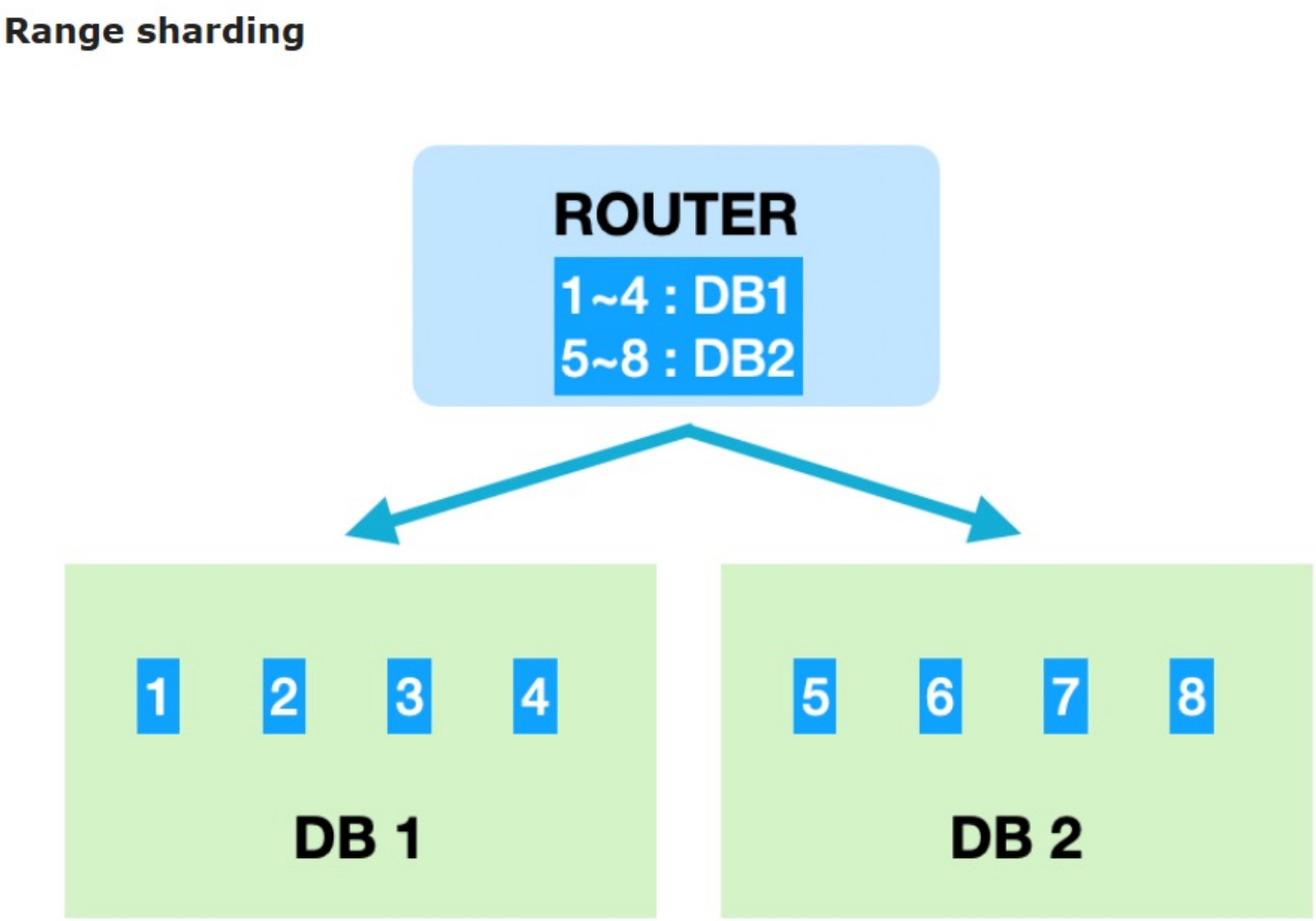
LRU Cache Structure in LevelDB

- sharding



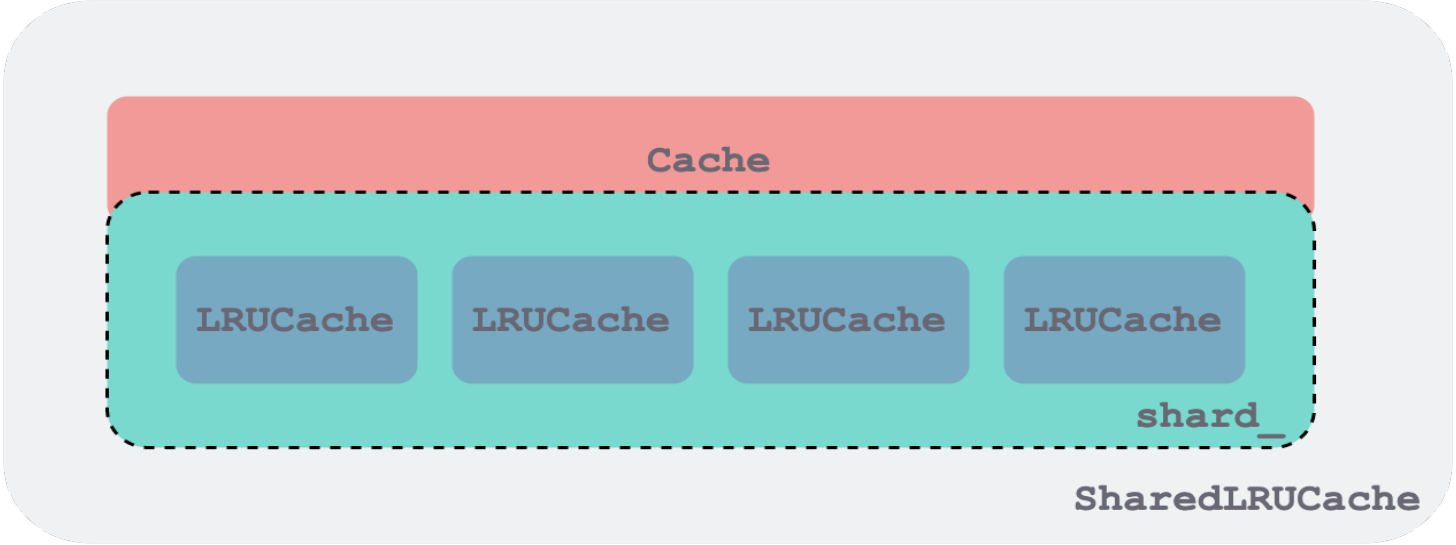
LRU Cache Structure in LevelDB

- sharding



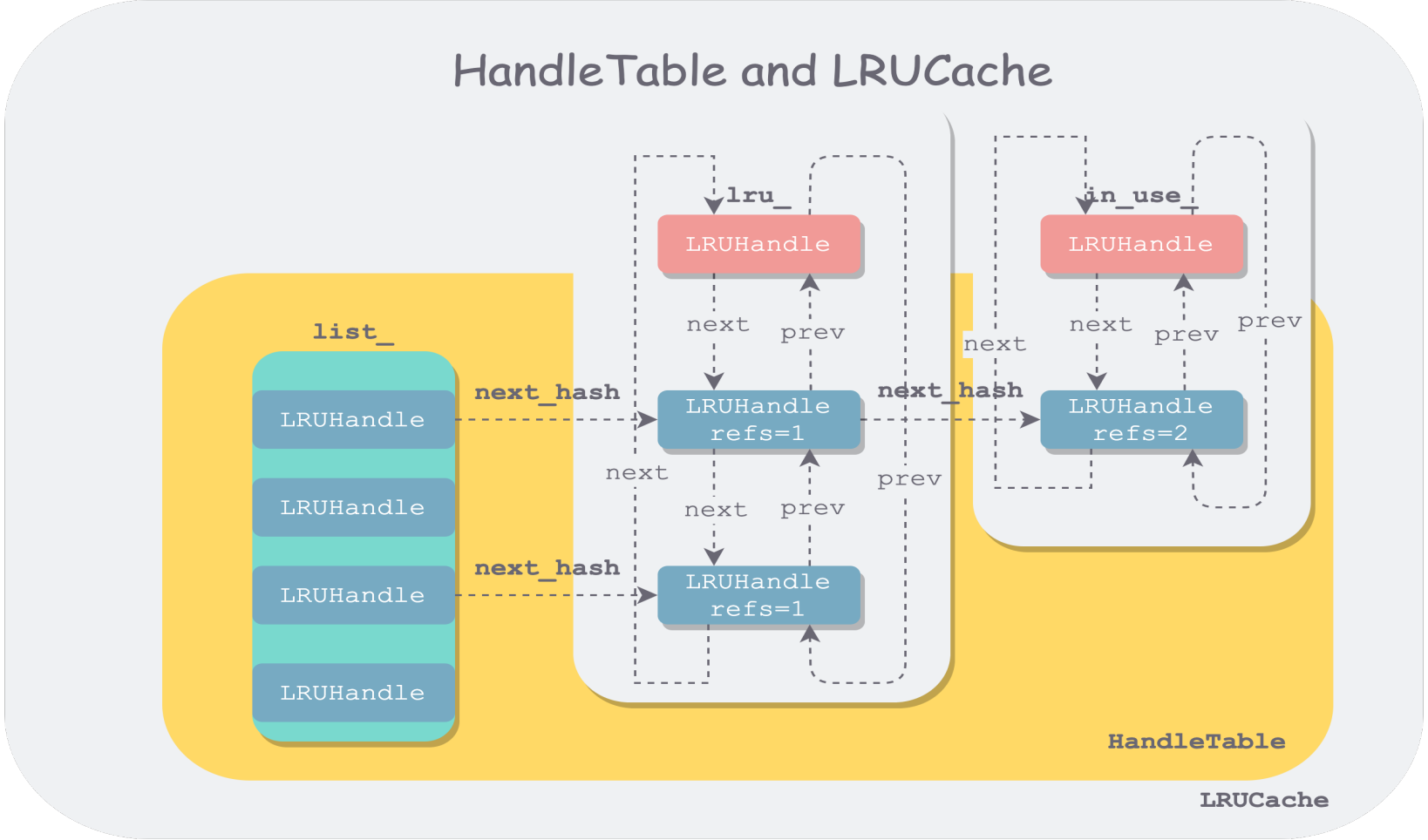
LRU Cache Structure in LevelDB

- sharding in LevelDB



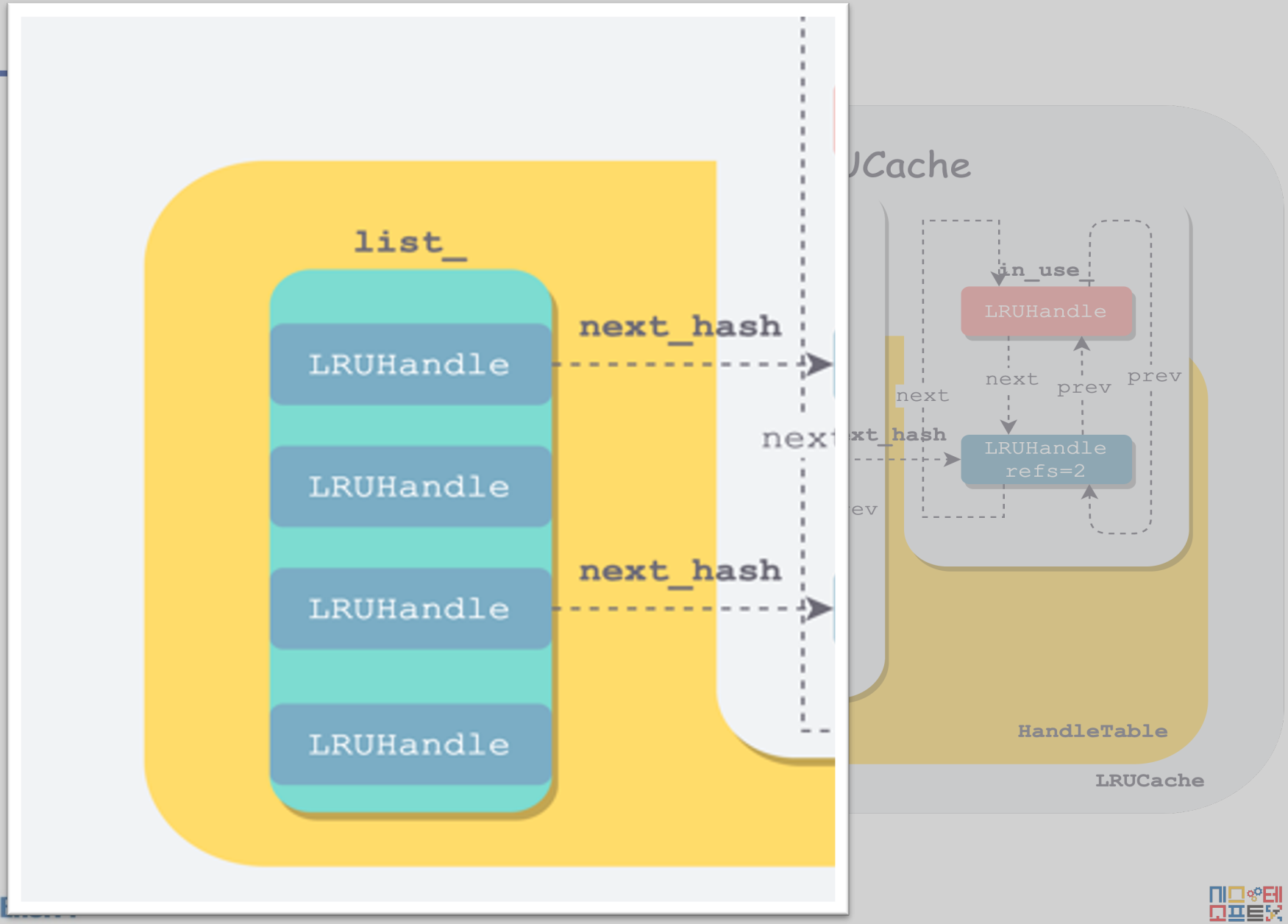
LRU Cache Structure in LevelDB

- analysis LRU structure in leveldb



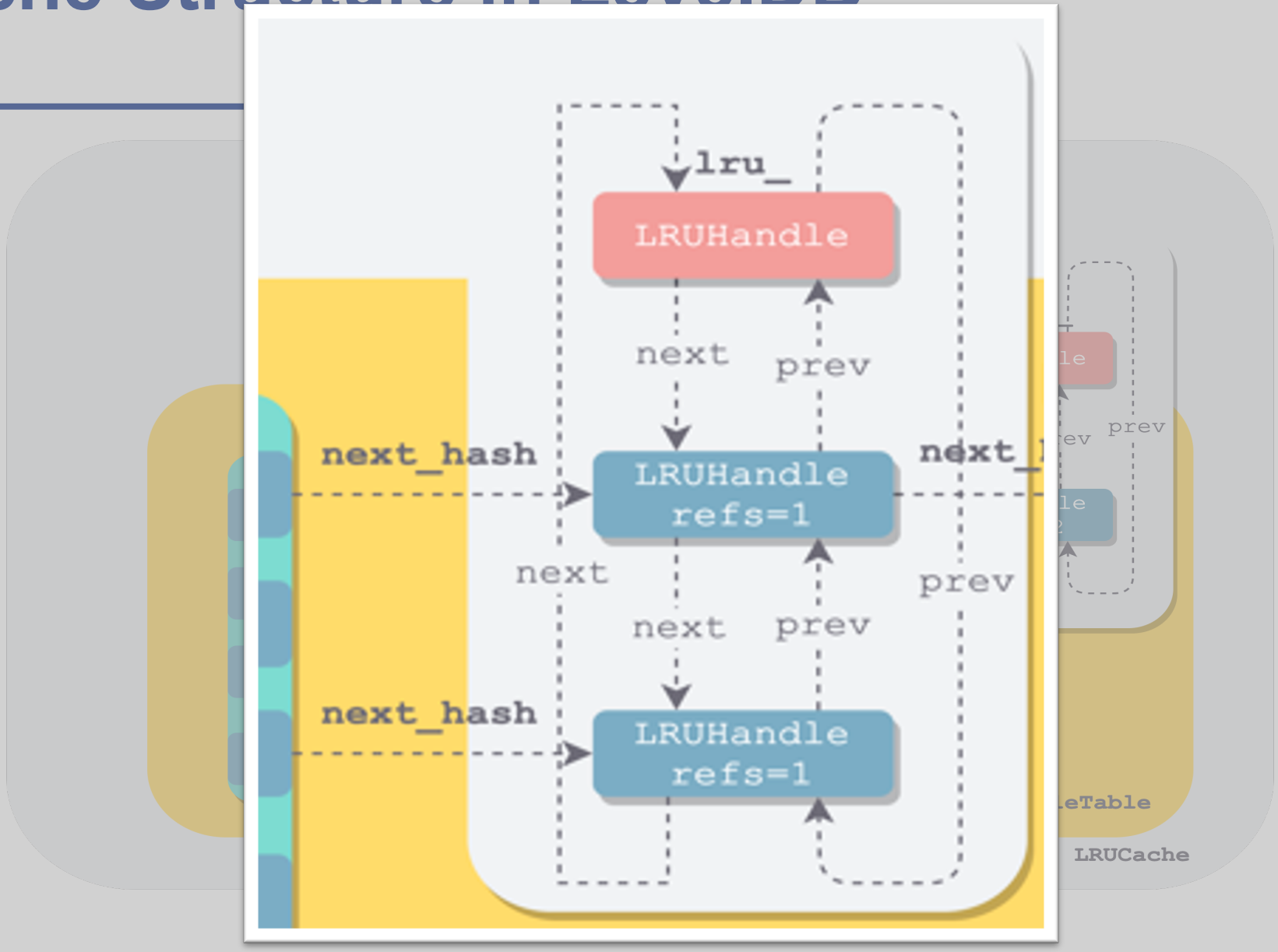
LRU Cache Structure in LevelDB

- hash table



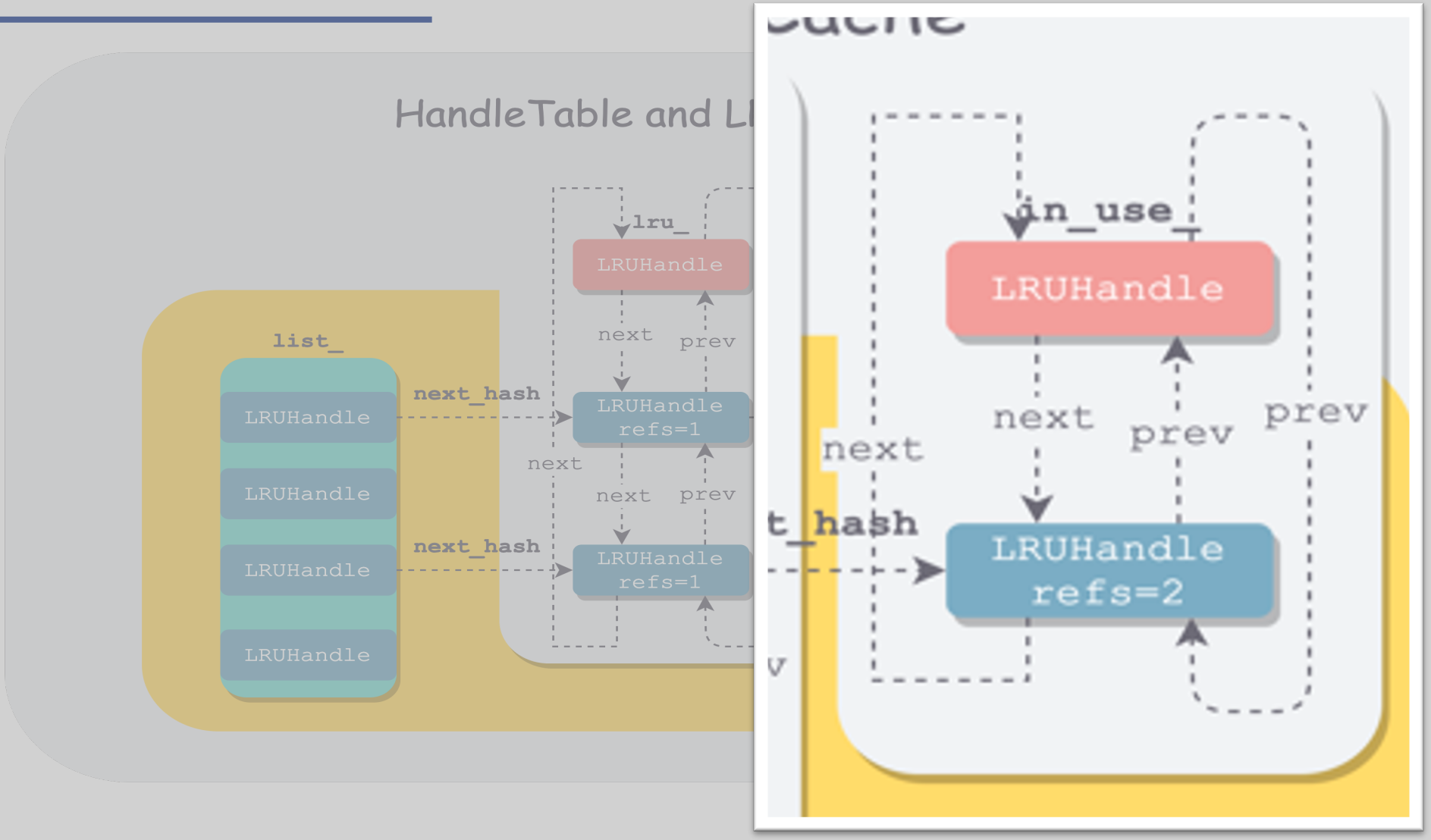
LRU Cache Structure in LevelDB

- hash table



LRU Cache Structure in LevelDB

- hash table



Benchmarks Experiment

- Hypothesis

Cache increases read performance by reducing disk I/O.

If so, it is obvious that increasing the size of cache_size will help improve latency performance.

Check latency when cache_size, block_size(options) change!

Benchmarks Experiment

- Design

In the case of fillrandom, the latency results are too random, so the experiment is conducted through fillseq.

- Check benchmarks

- readhot, readrandom, seekrandom

- Cache_size

- [-1,5000,100000]

- Block_size

- [4,8,32,128,256,512…]

Benchmarks Experiment

- Run Experiment

- Enviroment : --num=10000000(rawsize=1GB) => to check high latency change!

You can create a Block cache of your chosen the size for caching uncompressed data.

We recommend that this should be about 1/3 of your total memory budget. The remaining free memory can be left for the OS (Operating System) page cache. Leaving a large chunk of memory for OS page cache has the benefit of avoiding tight memory budgeting (see also: Memory Usage in RocksDB).

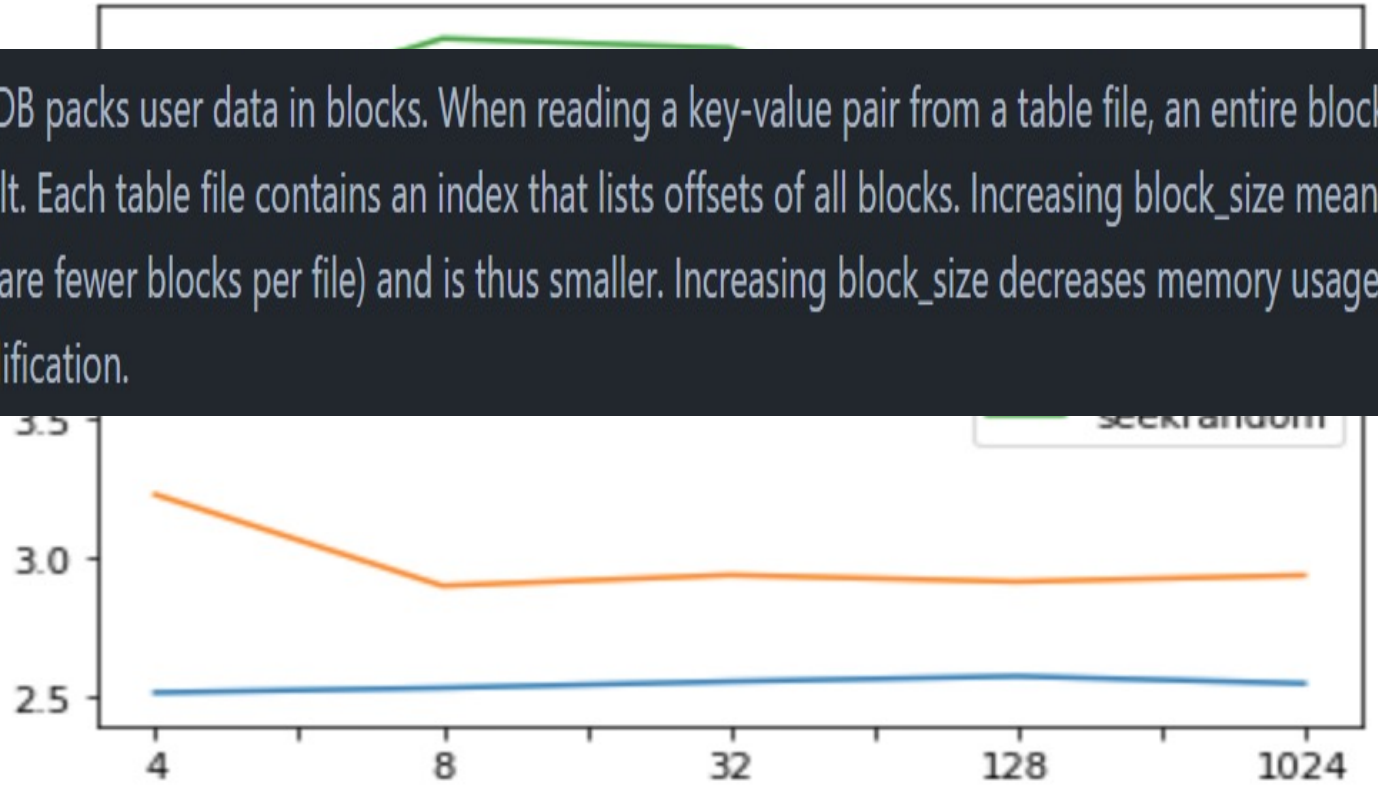
```
seekrandom   :       4.066 micros/op; (1)
seekrandom   :       3.838 micros/op; (2)
seekrandom   :       3.755 micros/op; (3)
seekrandom   :       4.115 micros/op; (4)
```

Benchmarks Experiment

- Run Experiment

- Enviroment :non

block_size -- RocksDB packs user data in blocks. When reading a key-value pair from a table file, an entire block is loaded into memory. Block size is 4KB by default. Each table file contains an index that lists offsets of all blocks. Increasing block_size means that the index contains fewer entries (since there are fewer blocks per file) and is thus smaller. Increasing block_size decreases memory usage and space amplification, but increases read amplification.



Block_size

Benchmarks Experiment

- Result and Discussion

- Cache_size improves read latency! But, the size should be appropriately adjusted according to the size of the file.

table_cache.cc
table_cache.h

- It is conceivable that block_size has nothing to do with read latency improve.

```
else if (sscanf(argv[i], "--cache_size=%d%c", &n, &junk) == 1) {
FLAGS_cache_size = n;
```

```
Benchmark()
    : cache_(FLAGS_cache_size >= 0 ? NewLRUCache(FLAGS_cache_size) : nullptr),
```

```
Cache* NewLRUCache(size_t capacity) { return new ShardedLRUCache(capacity); }
```

Reference

https://velog.io/@jelkov/%EC%BA%90%EC%8B%9C

https://chowdera.com/2022/187/202207061327595599.html

https://wiesen.github.io/post/leveldb-cache/

https://bbs.huaweicloud.com/blogs/251517

https://www.bookstack.cn/read/rocksdb-en/b3616cd1498e196f.md

https://www.bookstack.cn/read/rocksdb-en/5dd063f6b6ed224d.md#Block%20Cache%20Size

https://techblog.woowahan.com/2687/

https://velog.io/@gil0127/%EC%8B%B1%EA%B8%80%EC%8A%A4%EB%A0%88%EB%93%9CSingle-thread-vs-%EB%A9%80%ED%8B%B0%EC%8A%A4%EB%A0%88%EB%93%9C-Multi-thread-t5gv4udj