LevelDB Cache Thesis Draft

Made by subin hong

E-Mail: zed6740@dankook.ac.kr

1. 배경
1. LSM-tree

2. LevelDB

3. LevelDB에서의 Cache

2. 실험
1. LevelDB Benchmark experiment

1-1. LSM-tree

1-2. LevelDB

1-3. LevelDB에서의 Cache

2-1. LevelDB Benchmark experiment

LSM-tree

스키마도 없고 관계도 없는 NoSQL 데이터 베이스의 종류 중 하나인 RokcsDB, LevelDB

계속해서 증가하는 소스에서 엄청난 양의 데이터를 수집하면서 NoSQL 데이터베이스 뒤에는 LSM-tree(Log-Structured Merge-Tree) 구조가 존재한다.

LSM-tree는 빠른 쓰기에 최적화 되어있다.

1. 배경/이론

- 1. LSM-tree가 무엇인지
- 2. LevelDB에 대해서 설명
- 3. LevelDB에서의 Cache에 대해서 설명
  - LRU
  - Sharding
  - Table Cache
    - Code Flow, Structure
  - Block Cache
    - Code Flow, Structure

LSM-tree

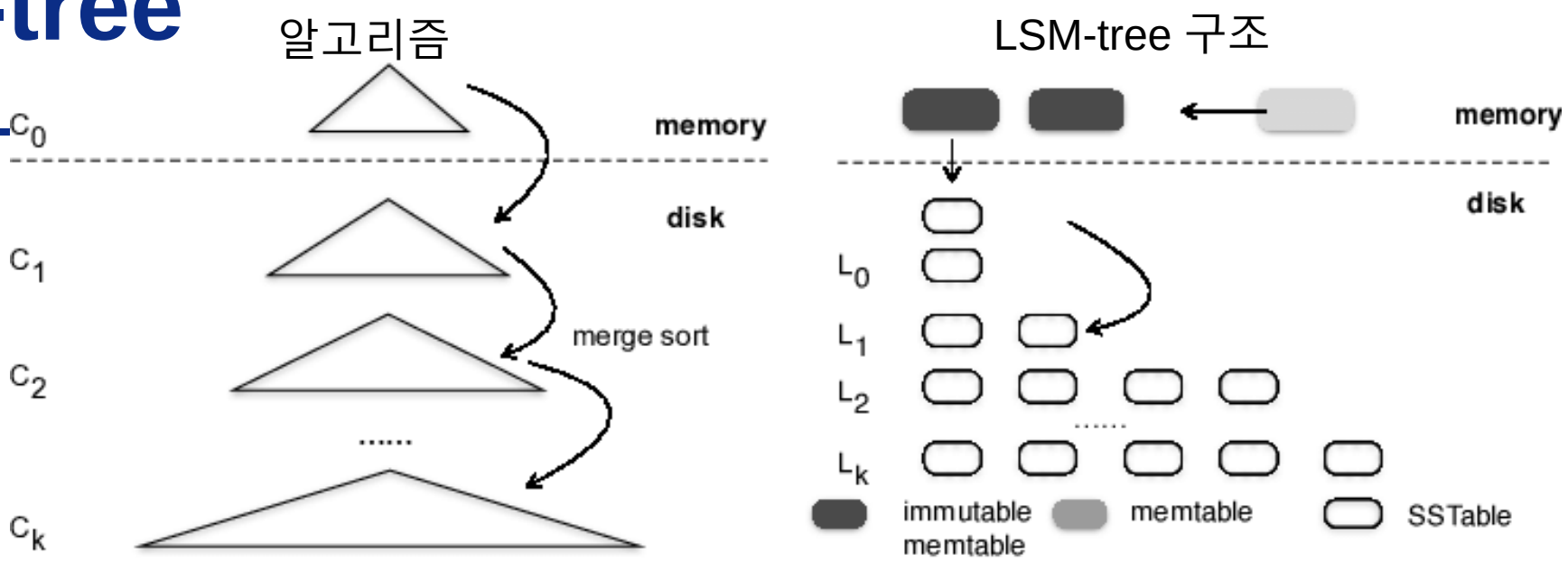
스키마도 없고 관계도 없는 NoSQL 데이터 베이스의 종류 중 하나인 LevelDB

최근 계속해서 증가하는 소스에서 엄청난 양의 데이터를 수집하면서 SQL로는 담을 수 없는 데이터들로 인해 NoSQL을 여러 곳에서 사용

이러한 NoSQL데이터베이스 뒤에는 LSM-tree(Log-Structured Merge-Tree) 구조가 존재

<LSM-tree는 빠른 쓰기에 최적>

LSM-tree



데이터를 읽어 들이는 것은 캐시에 의해 처리되며 이는 LevelDB의 Cache 에서 후술

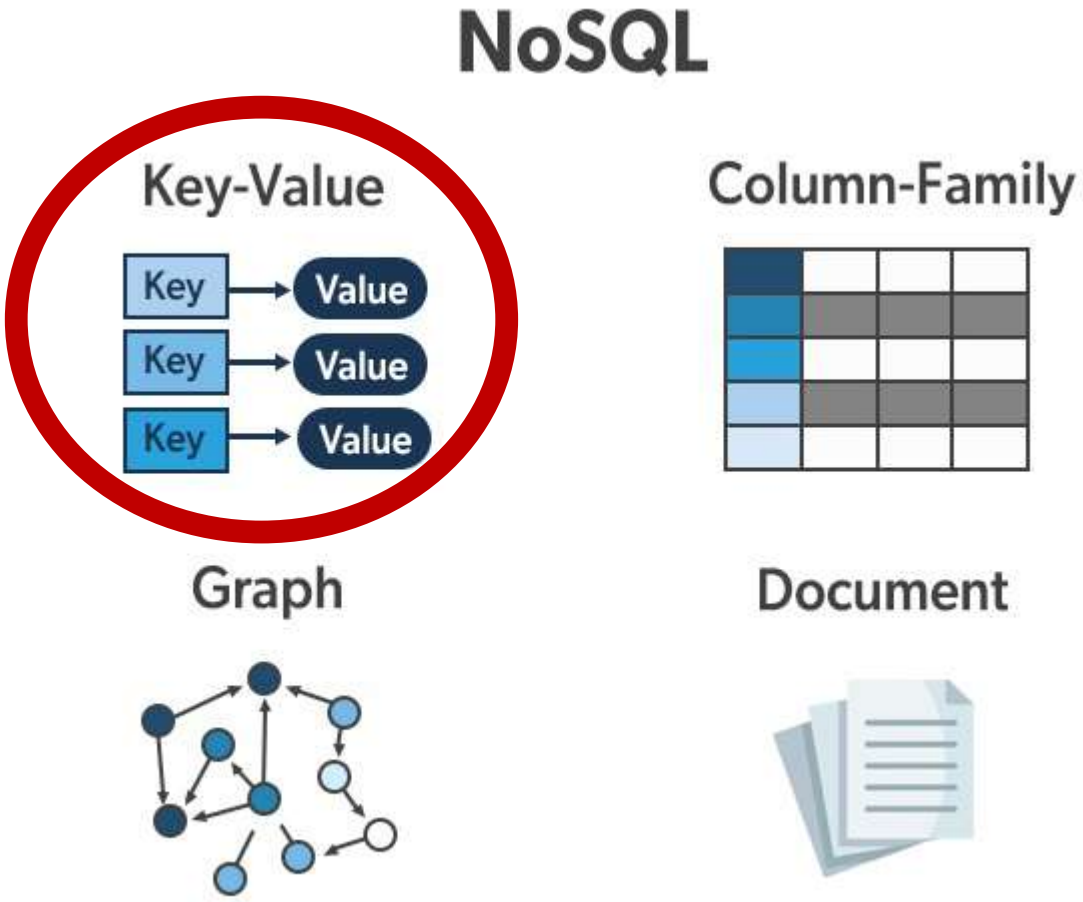
읽기는 캐시에 의해서 처리되기 때문에 파일 시스템 성능은 쓰기에 의존

순차적인 I/O를 지원하는 LSM-tree는 임의적 I/O보다 디스크 성능을 크게 향상

오른쪽의 그림을 보면 LSM-tree는 log(Log Structure) -> mem -> im mem -> sstable.. 식으로 순차 I/O를 구현

데이터가 한 레벨에서 가득 찰 때마다 해당 데이터를 아래의 레벨로 내리는데 이 때 아래 레벨의 데이터와 합쳐 merge sort 방식을 이용하기 때문에 ls M(Merge)-tree라고 불림

LevelDB



LevelDB는 Google에서 만든 빠른

key-value storage library

=> string keys로부터 string values값으로 정렬된 매핑을 제공

Key-value => NoSQL => LSM-tree

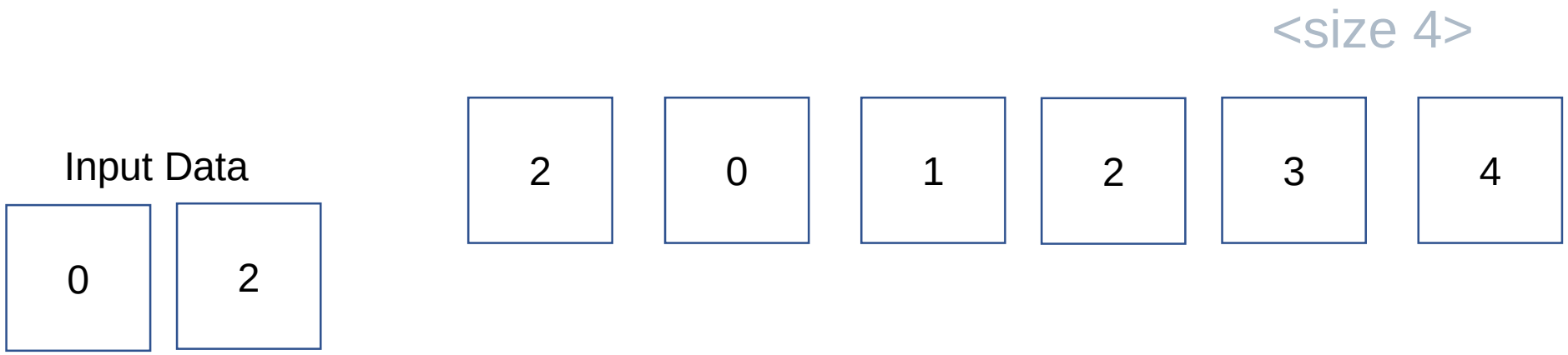
Look forward to Write performance improve

LevelDB에서의 Cache

What is LRU?

LRU(Least Recently Used)란 직역하면 최소로 최근에 사용

LRU 정책이란 페이지 교체 알고리즘으로 불리며 페이지에서 데이터를 제거할 때 가장 오랫동안 사용하지 않은 것을 제거하겠다는 알고리즘



LevelDB에서의 Cache

What is Cache?

캐시는 컴퓨팅 환경에서 일시적으로 무언가 데이터를 저장하는데 사용되는 하드웨어 또는 소프트웨어이며 최근 또는 자주 액세스하는 데이터의 성능을 향상시키는데 더 빠르게 사용되는 더 빠르고 더 비싼 소량의 메모리

LevelDB에서의 Cache

What is LRU Cache?

Cache의 용량이 한정되어 있는 경우가 대부분이기 때문에 Cache가 가득 찼을 때 LRU 정책을 Cache 내부에서 구현한 것을 LRU Cache라 함

이는 LevelDB에서 사용하는 Cache구조!

LevelDB에서의 Cache

Use Cache

get() : cache에서 항목을 가져오는 api

put() : cache 앞에 항목을 추가하고

Cache의 적정 용량이 넘칠경 우 가장 적게 사용된 항목을 제거하는 api

*LevelDB는 delete를 할 때 put에 인자로 전달함

LevelDB에서의 Cache

get() : Need Fast Lookups(Cache를 들여다 보는 함수)

빠른 조회가 가능한 자료구조?

put() : Need Fast Removals

Least Recently Used한 항목의 위치와 그 항목을 빠르게 제거하는 방법

원하는 항목을 빠르게 얻을 수 있는 자료구조?

LevelDB에서의 Cache

Why use Linked List?

사실상 LevelDB의 LRU Cache에서는 Hash Table과 double Liked List를 사용 중

double Liked List는 현재 있는 노드가 다음과 이전 노드에 접근이 가능한 자료구조이기 때문에

이전 노드와 다음 노드의 next, prev를 조정하여 자기 자신 노드를 빠른 시간안에 제거할 수 있기 때문에 double linked list 사용

LevelDB에서의 Cache

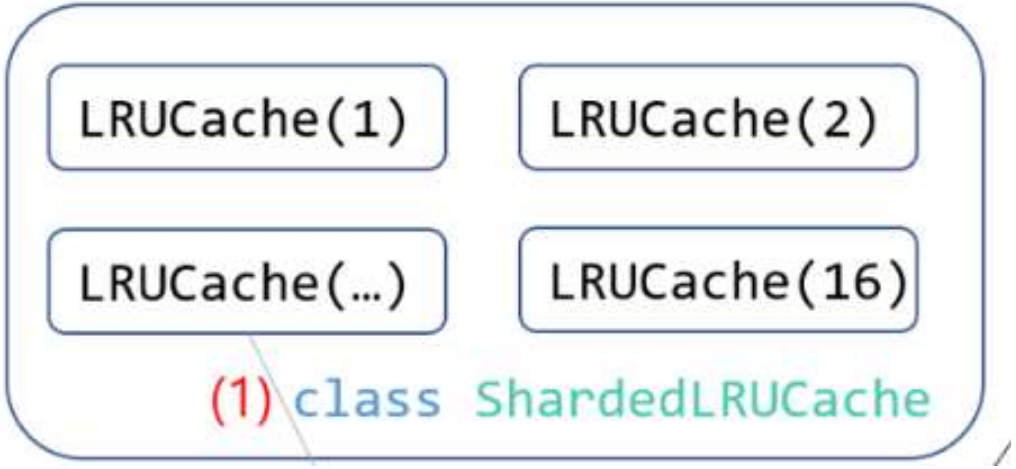
What is Sharding?

Shard : 조각, 파편

Sharding : 데이터를 조각으로 나누어 관리하는 일종의 로드 밸런싱 기술

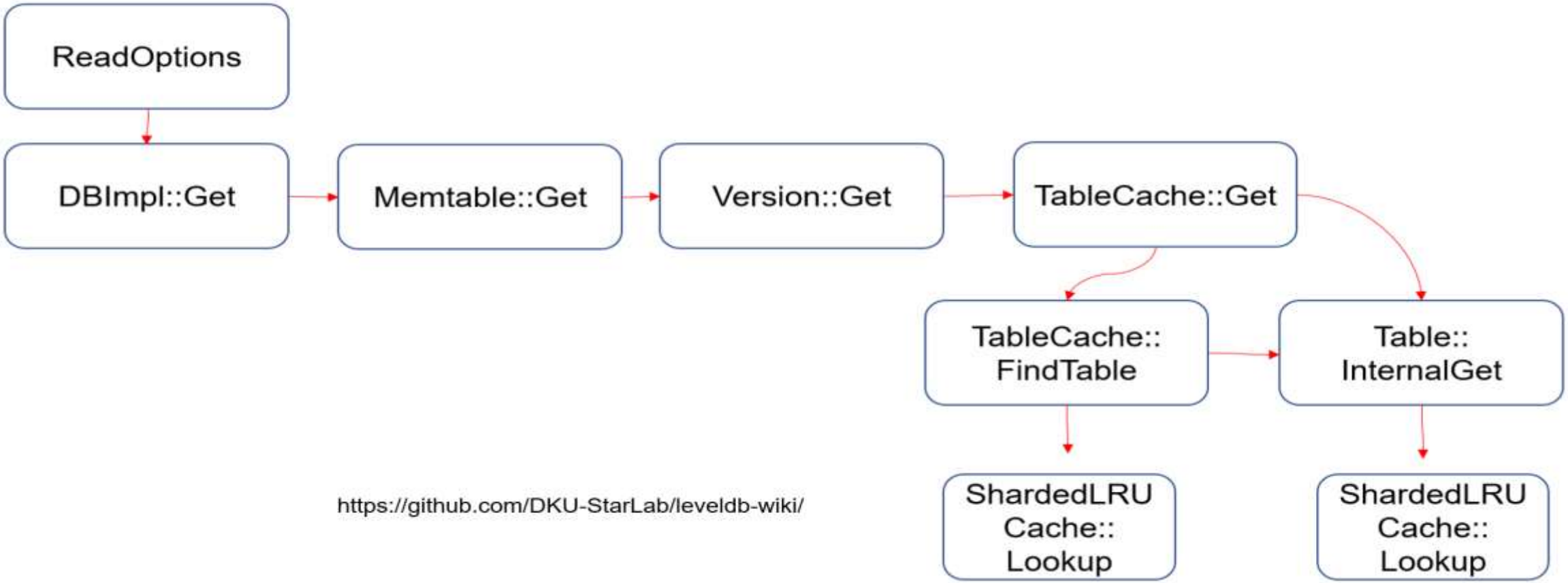
LevelDB에서 사용하는 LRU Cache는 Shard 기법을 통해 ShardedLRUCache으로 관리됨

⇒ ShardedLRUCache에서 LRU Cache들을 호출할 때 각각에 대해서 Lock을 수행, 이는 동일한 LRU Cache 개체에 대해서 여러 스레드들이 접근하는 것(경합)을 줄여 다중 스레드 조회 속도를 높임



LevelDB에서의 Cache

```
leveldb::Status s = db -> Get(leveldb::ReadOptions(),key,&value);
```



https://github.com/DKU-StarLab/leveldb-wiki/

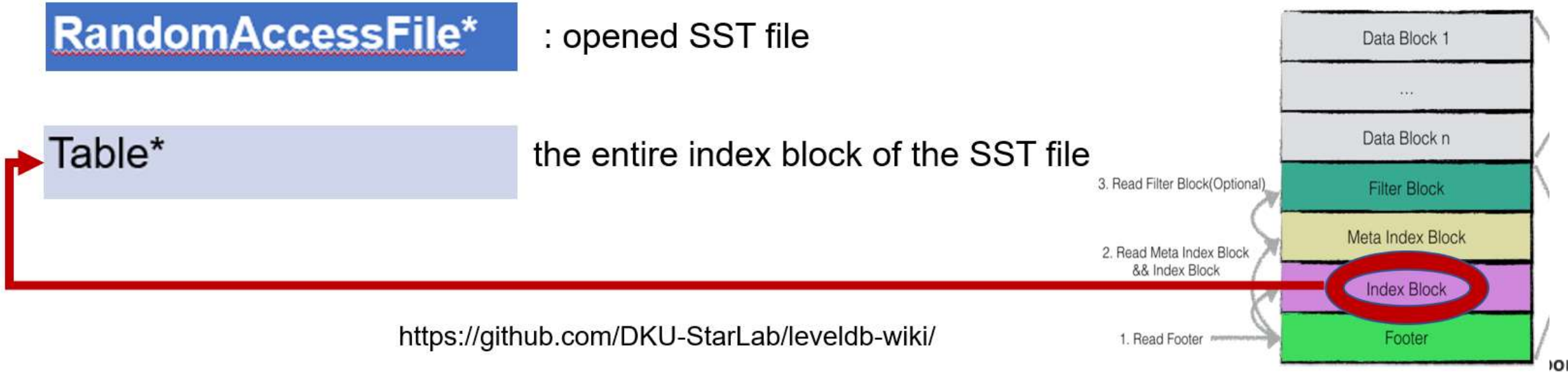
LevelDB에서의 Cache

key	value
file_number	RandomAccessFile*
	Table*

Table Cache Structure

RandomAccessFile* : opened SST file

Table* the entire index block of the SST file



https://github.com/DKU-StarLab/leveldb-wiki/

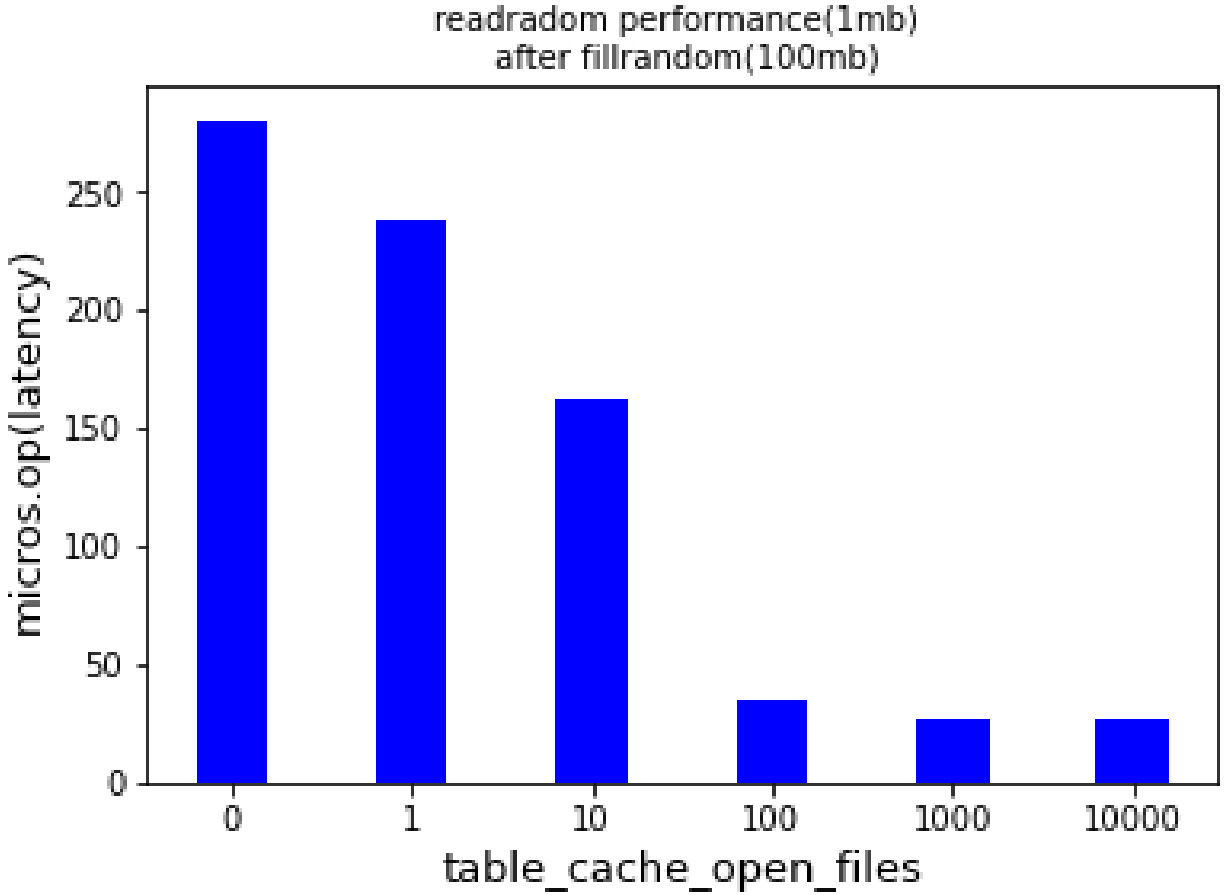
LevelDB에서의 Cache

key	value
cache_id + block_offset	**Block Data (Data Block)**
cache_id + block_offset	Block Data
cache_id + block_offset	Block Data

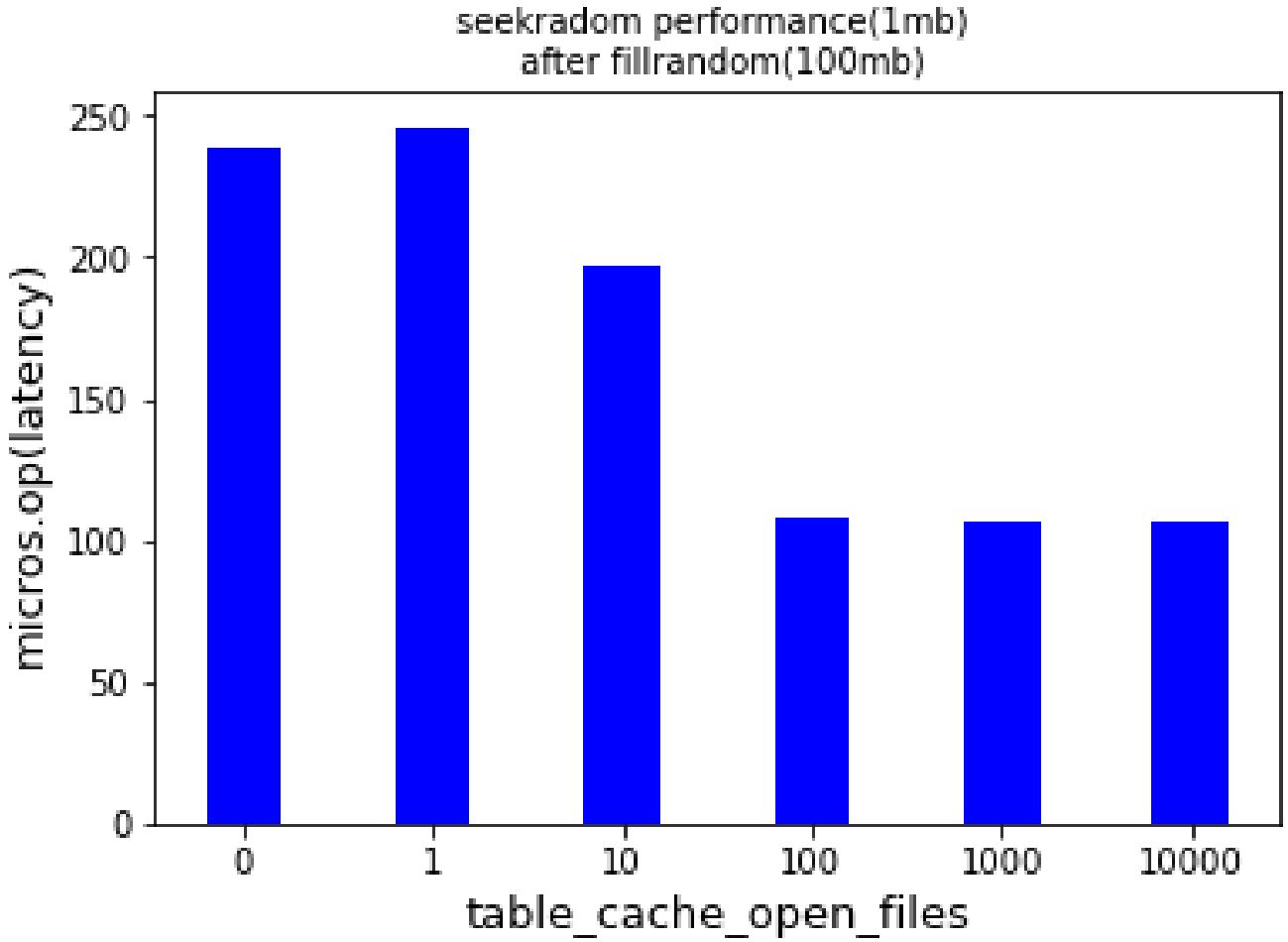
Block Cache Structure

https://github.com/DKU-StarLab/leveldb-wiki/

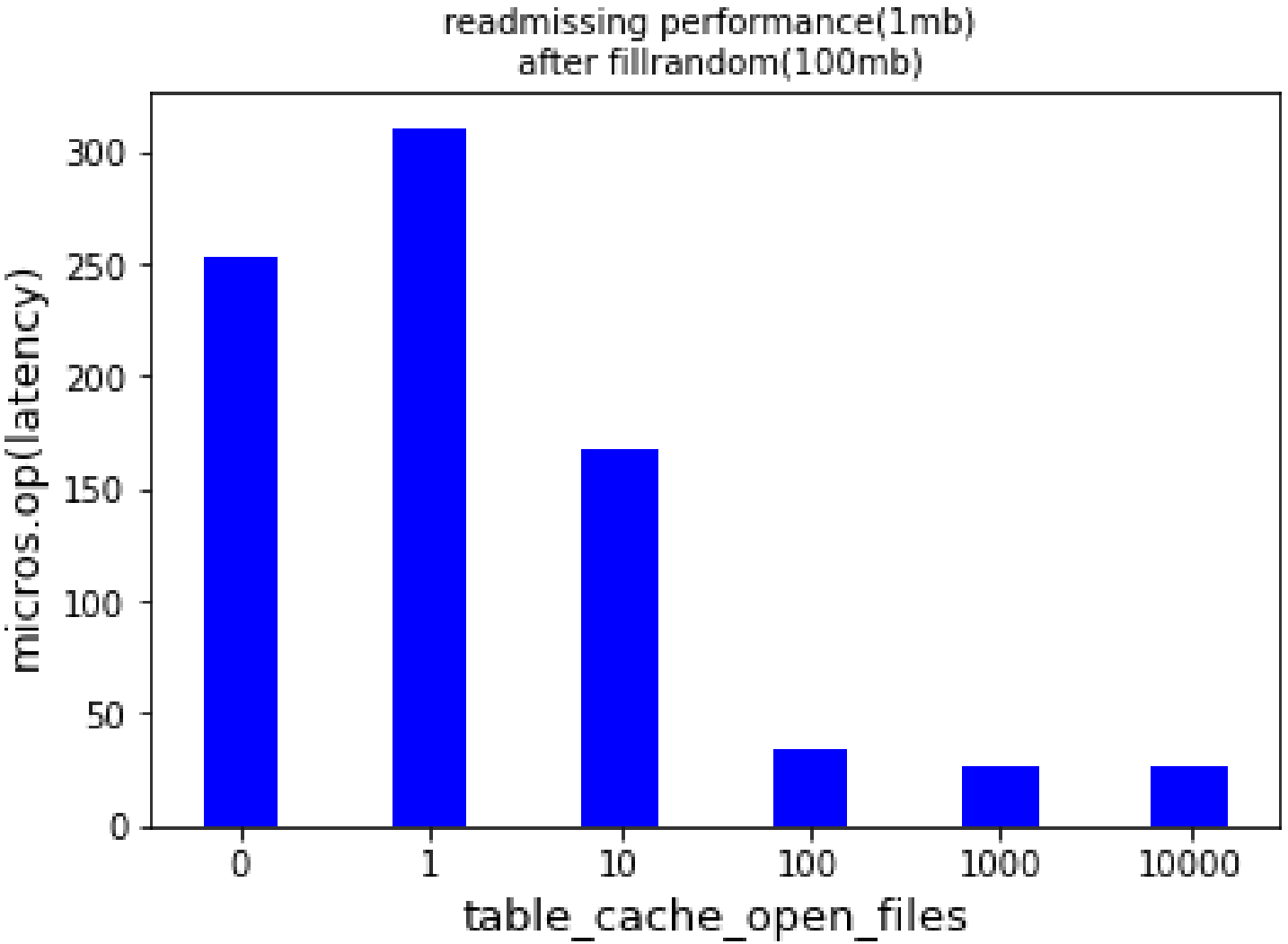
LevelDB Benchmark experiment



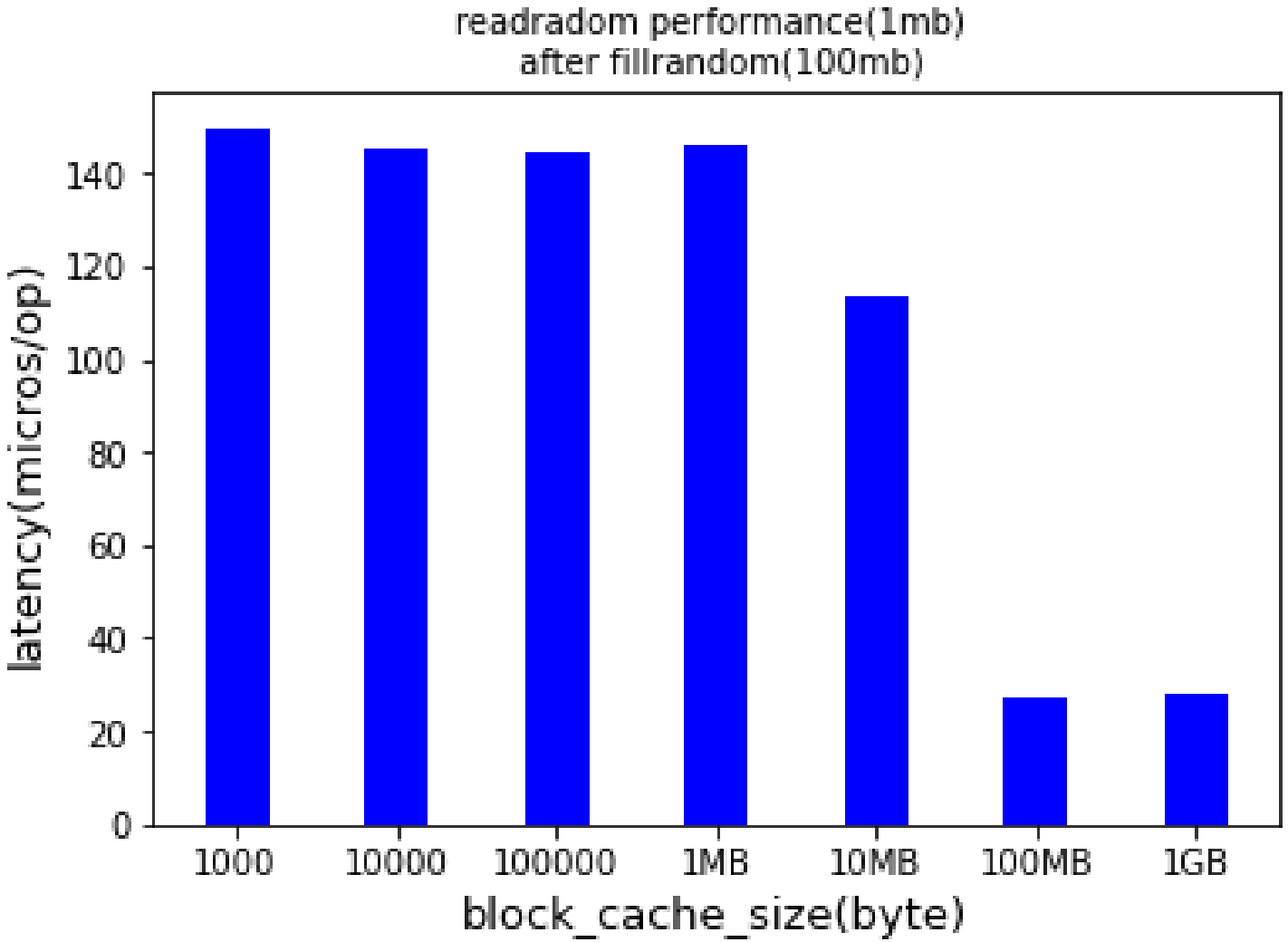
LevelDB Benchmark experiment



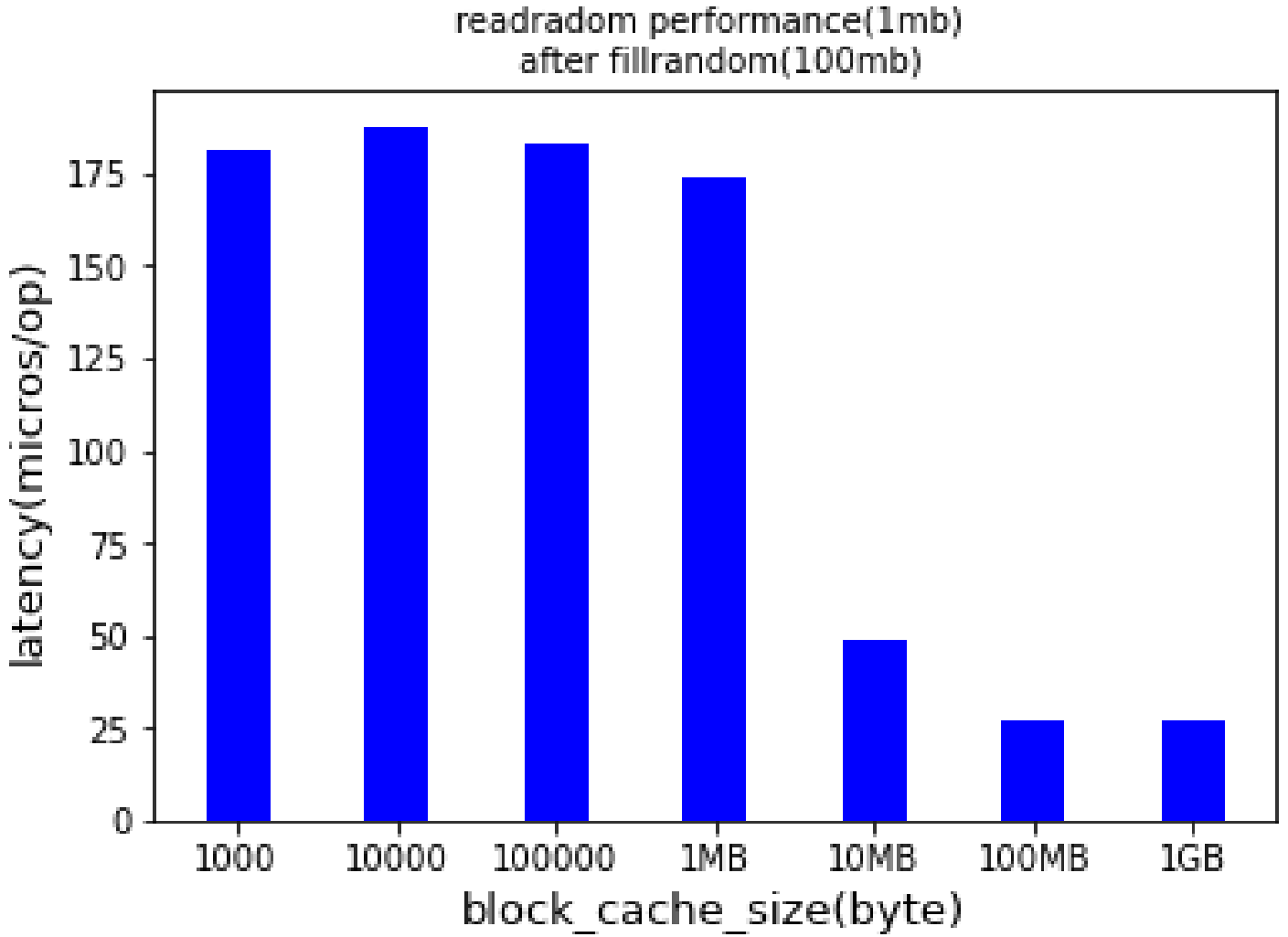
LevelDB Benchmark experiment



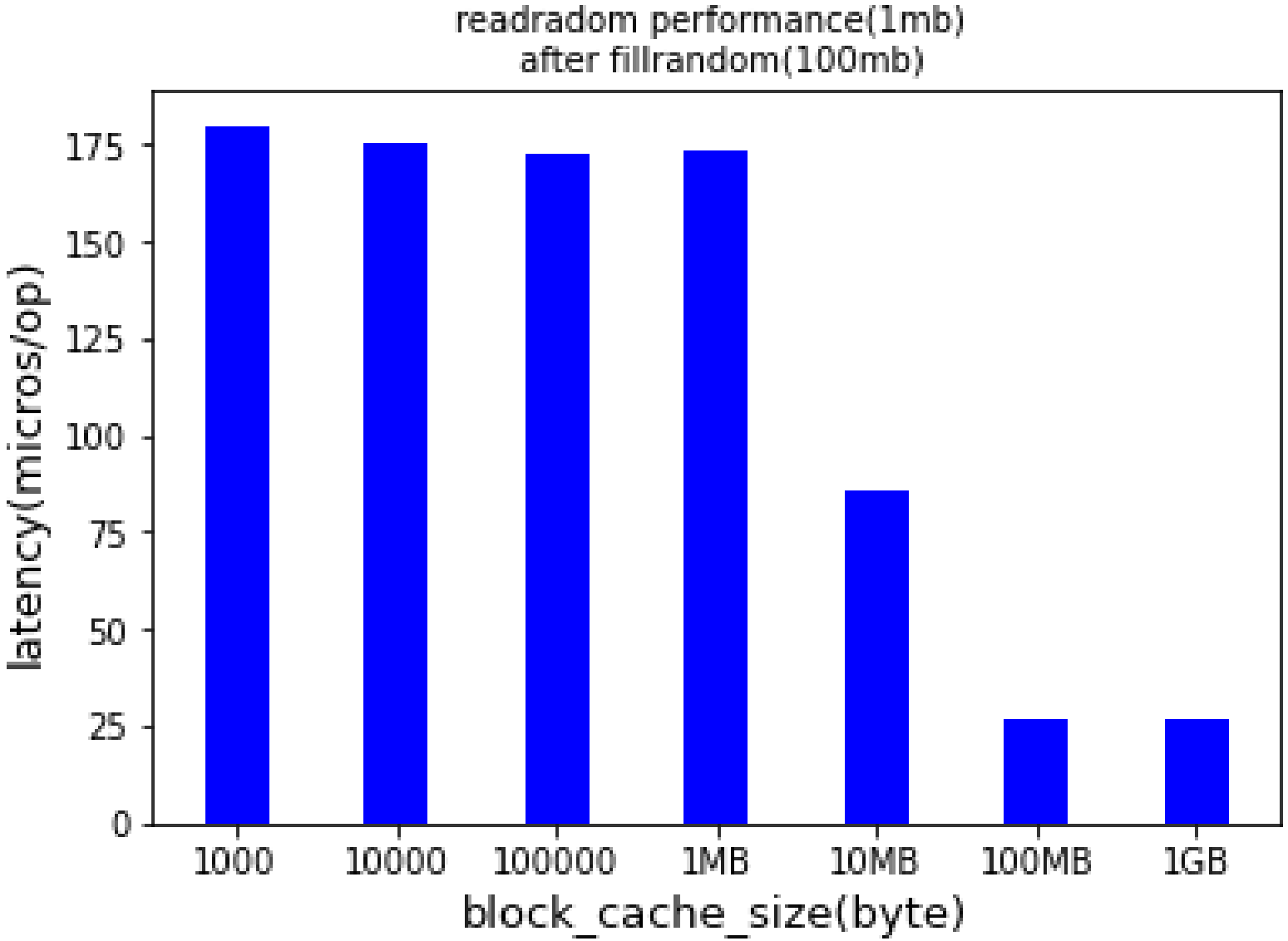
LevelDB Benchmark experiment



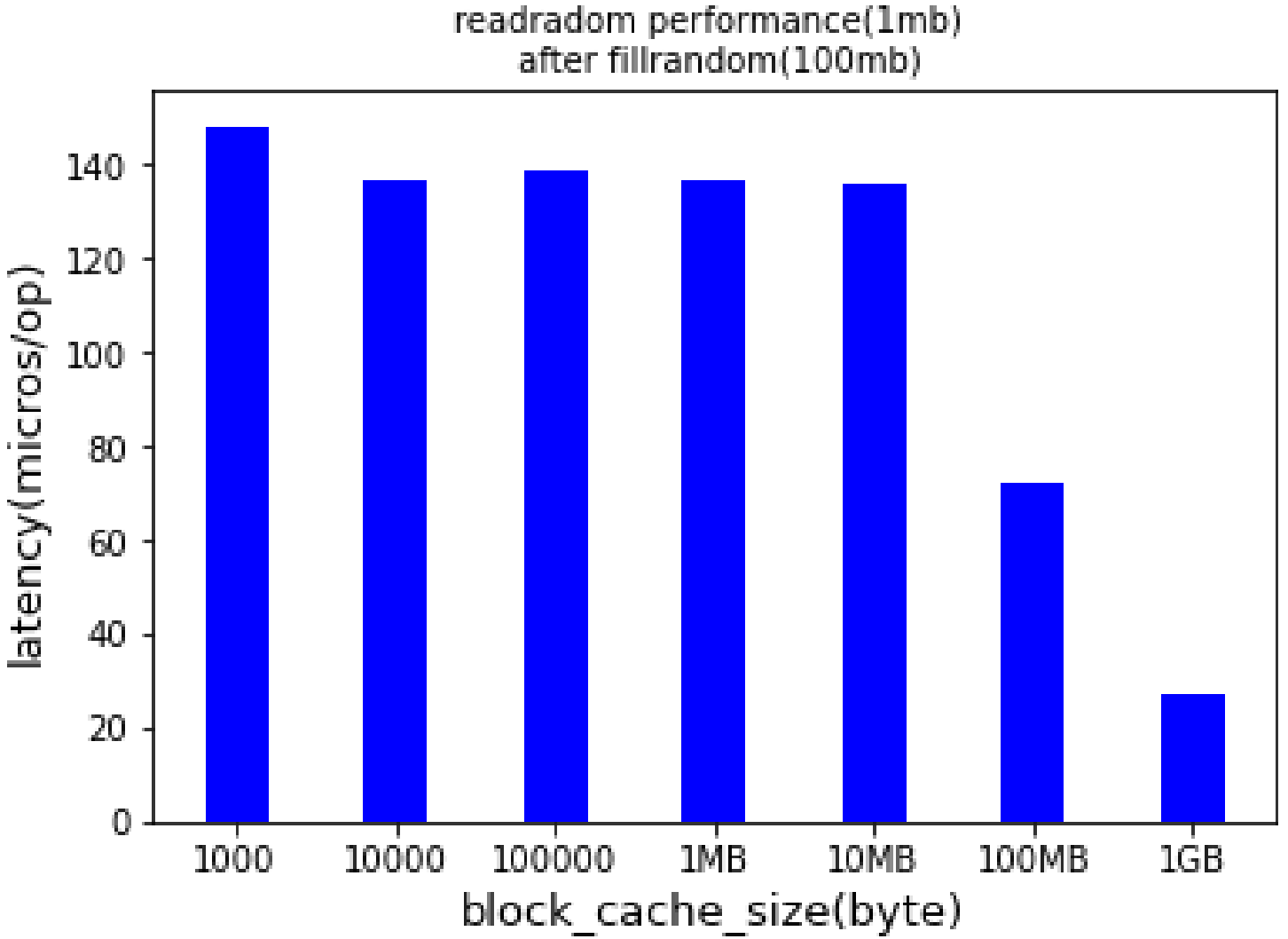
LevelDB Benchmark experiment



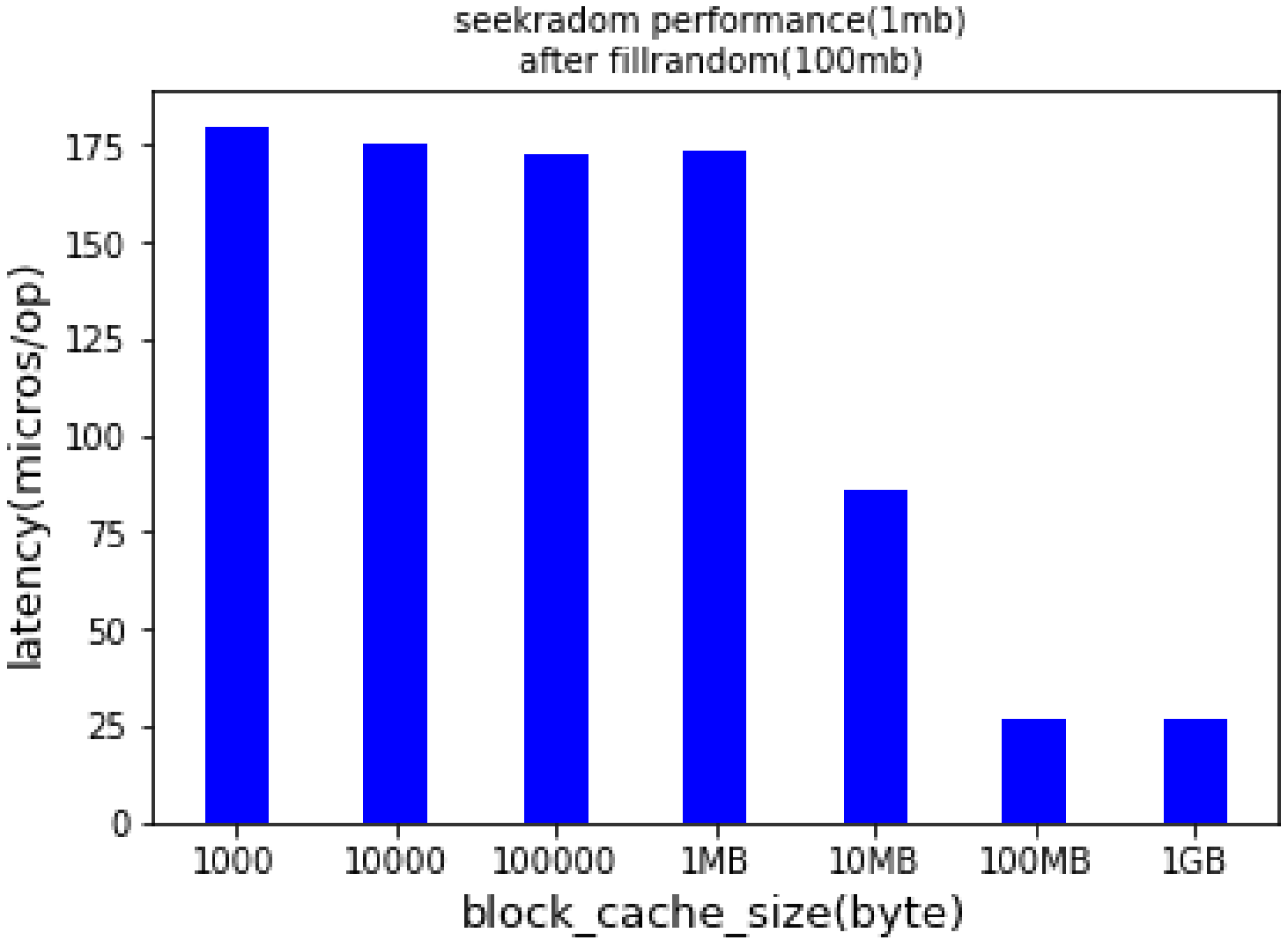
LevelDB Benchmark experiment



LevelDB Benchmark experiment



LevelDB Benchmark experiment



LevelDB Benchmark experiment

Table cache size = 4mb

Fill = 100mb , table 개수 => 25개..