2025 Winter RocksDB Study
1st week

Hojin Shin, Guangxun Zhao

http://sslab.dankook.ac.kr/, https://sslab.dankook.ac.kr/~choijm

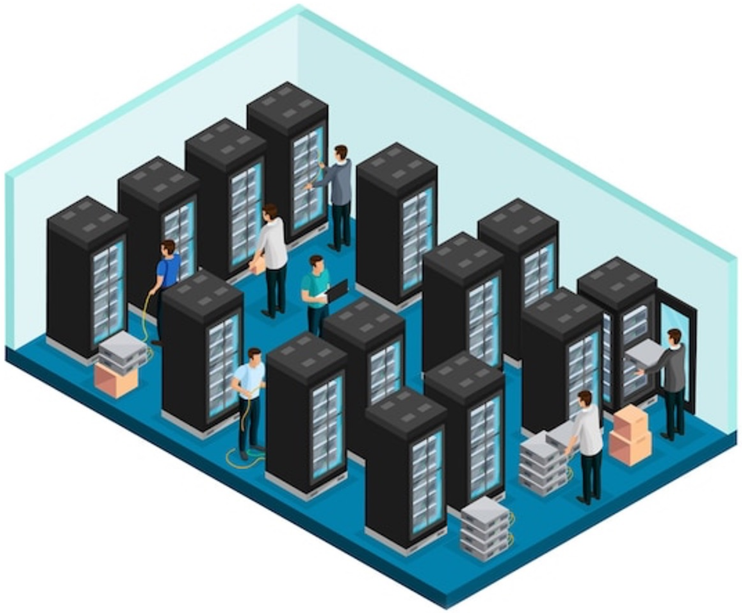
Presentation by Hojin Shin

hojin03s@dankook.ac.kr

Contents

Chapter 1

What is data?

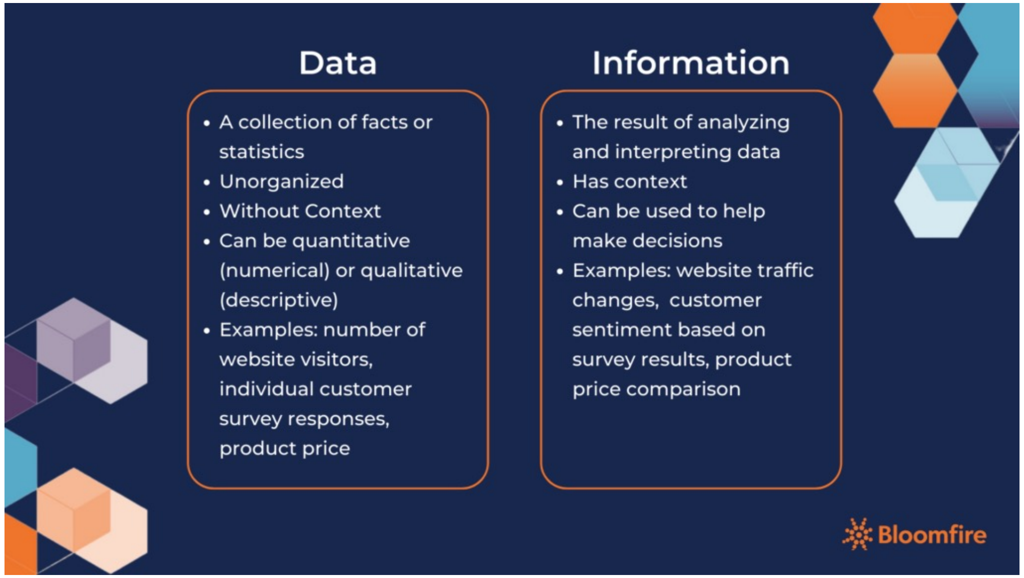


What is data?

- Data
  - ✓1) Units of **information**, often numeric, that are collected through observation
  - ✓2) Fact on which a theory is based
  - ✓3) Data in the form of letters, numbers, sounds, pictures that a computer can process

What is data?

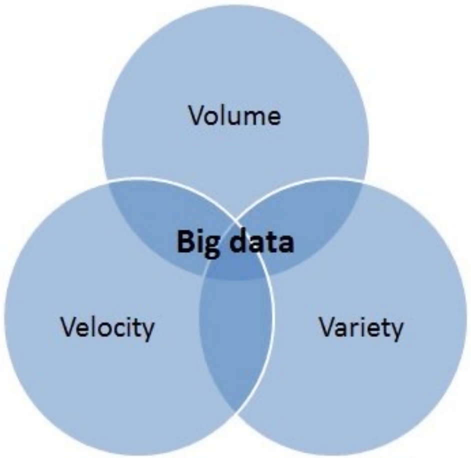
- Information
  - ✓ Information is obtained by processing data
  - ✓ A form in which data is processed according to its meaning and purpose for specific decision-making



source: https://bloomfire.com/blog/data-vs-information/

What is data?

- Bigdata
  - ✓ A large amount of **structured data** that exceed existing DB management tools
  - ✓ Set of **unstructured data** that is not in the form of data
  - ✓ Features: 3V ➔ 10V

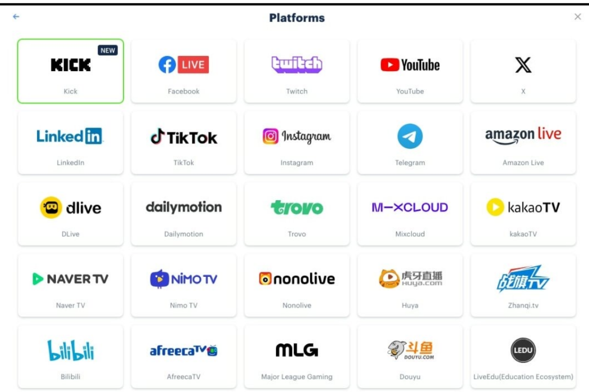


source: https://www.optalitix.com/insights/what-are-the-3-vs-of-big-data

source: https://www.researchgate.net/figure/Vs-of-big-data-characteristics_fig1_354879212

What is data?

- Bigdata features
  - ✓ Volume
    - Big Data refers to an **enormous** amount of data that is difficult to handle using traditional data management systems
  - ✓ Velocity
    - Refers to how **fast** data is generated, transmitted, and needs to be processed
  - ✓ Variety
    - Data comes in **various forms**, including structured, unstructured, and semi-structured formats



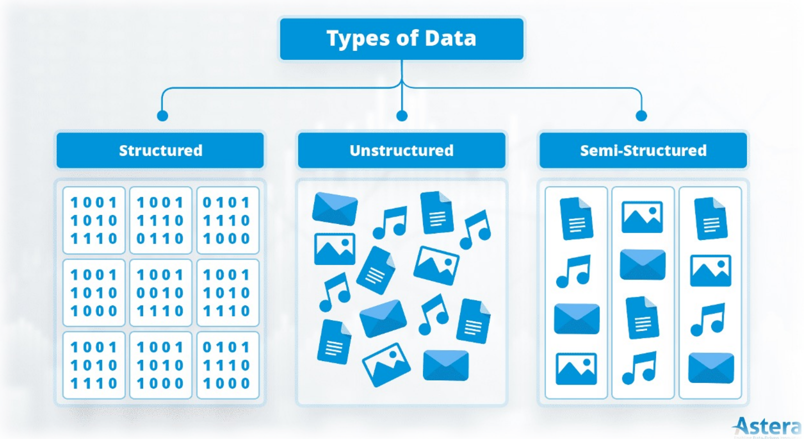
Variety

Volume: Live Streaming Platform

Velocity: Social Network Service

What is data?

- Types of data
  - ✓ Structured Data
    - Data organized and processed into a form suitable for immediate statistical analysis
    - Data stored in **fixed** fields
  - ✓ Unstructured Data
    - One piece of data, **not a set of data**, is objectified as collected data
    - Difficult to understand the meaning of a value because there is no set rule
  - ✓ Semi-structured Data
    - File type, metadata (schema of structured data inside data)



source: https://www.astera.com/type/blog/unstructured-data-challenges/

What is data?

- Types of Database
  - ✓ SQL (Structured Query Language)
    - Interact with a particular type of database
    - Can store, modify, delete and retrieve data from RDBMS
    - Features: Strict Schema, Relation
  - ✓ NoSQL (Not only SQL)
    - Adjust the stored data at any time and add new “fields”
    - **Key-value**, document, wide-column, graph
    - Features: No schema, No relation

What is data?

- Key-Value Store (a.k.a Key-Value DB)
  - ✓ A de-facto standard DB for unstructured data
  - ✓ Google, Facebook, Amazon, Microsoft, MongoDB, Yahoo, Hbase, LinkedIn, Oracle, Baidu, Basho, In Memory DB (Memcached, Redis), …

What is data?

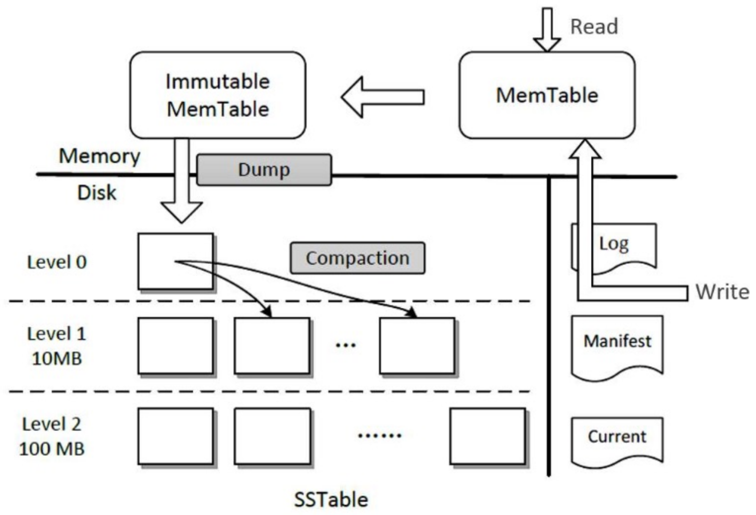
- Key-Value Store: some examples
  - ✓ LevelDB
    - By Google, 2011, a subset of Bigtable (Column-oriented DB, OSDI, 2006)
    - Leveled compaction, Open-source
  - ✓ RocksDB
    - By Facebook (Meta), 2012, a fork of LevelDB
    - Various algorithms (e.g. Tiered compaction, Blob file), High performance, Diverse applications
  - ✓ Hbase
    - By Apache, 2008, motivated by Google's Bigtable
    - A distributed data storage system for the Hadoop ecosystem
  - ✓ Cassandra
    - By Apache, 2008, motivated by Amazon's Dynamo
    - A decentralized architecture, each node is powered by LSM-tree based DB

Chapter 2

LevelDB, RocksDB Basics

LevelDB, RocksDB basics

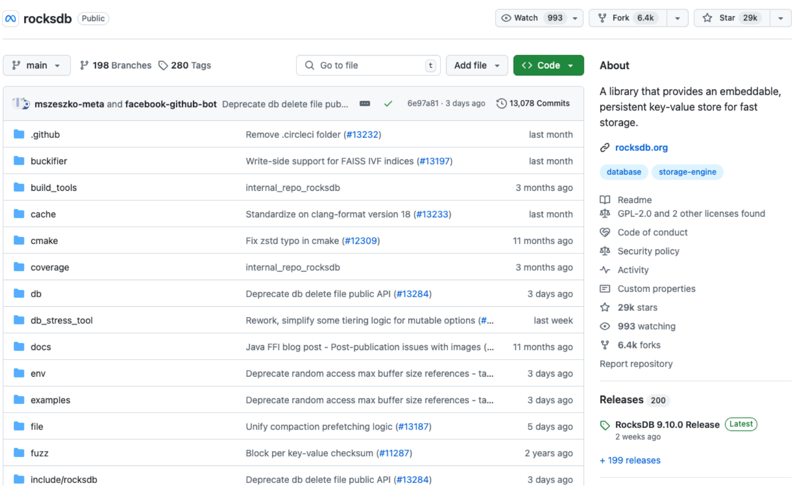
- What is LevelDB
  - ✓ Google's opensource project
  - ✓ Developed in the programming language C++
  - ✓ Data is stored after sorting by key
  - ✓ Operation: Put(K, V), Get(K), Delete(K)
  - ✓ Multiple operations can be created and processed in one batch
  - ✓ Limitation
    - Sing processing: only one process can access DB
    - Not support SQL query

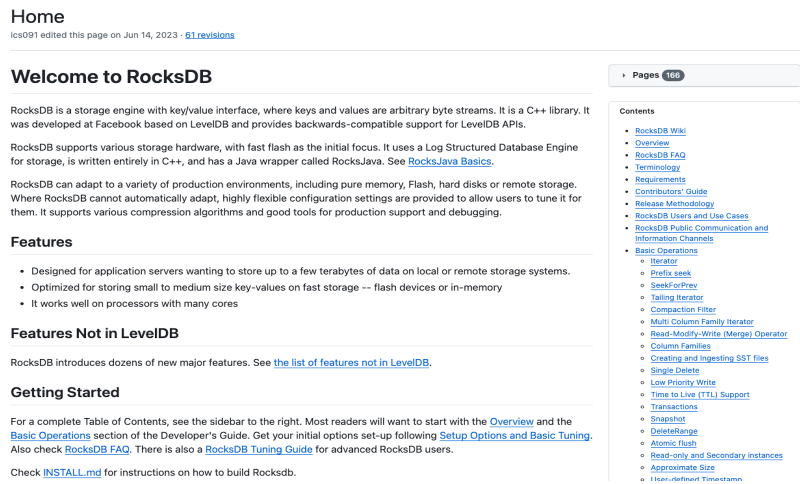


LevelDB Architecture

LevelDB, RocksDB basics

- What is RocksDB
  - ✓1) Famous KV Store of Facebook (Meta), derived from LevelDB
  - ✓2) A persistent storage engine that supports key/value interface
  - ✓3) LSM (Log Structured Merge)-Tree based (for SSD)
  - ✓4) Embedded (C++ library) and open source
  - ✓5) Support various algorithms, configurations, tools and debugging facilities

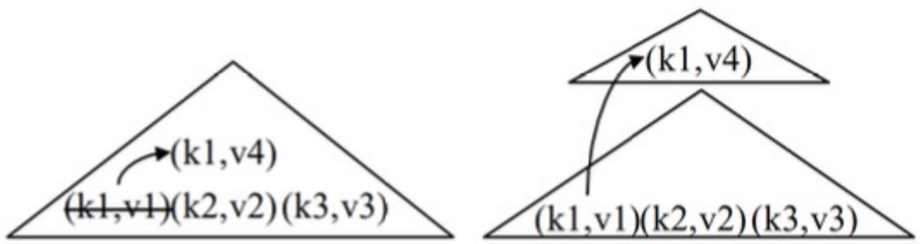




source: rocksdb github and rocksdb wiki

LevelDB, RocksDB basics

- What is LSM (Log Structured Merge)-tree
  - ✓ By Patrick O'Neil, The Log-Structured Merge Tree, 1996
  - ✓ Write Optimized data structure
  - ✓ Log-structure: In place update ➔ out-of-place update
    - In place update: good for **read**, bad for write (due to random writes)
    - Out-of-place update: good for **write**, possible bad for read (due to multiple locations), need reclaiming mechanism



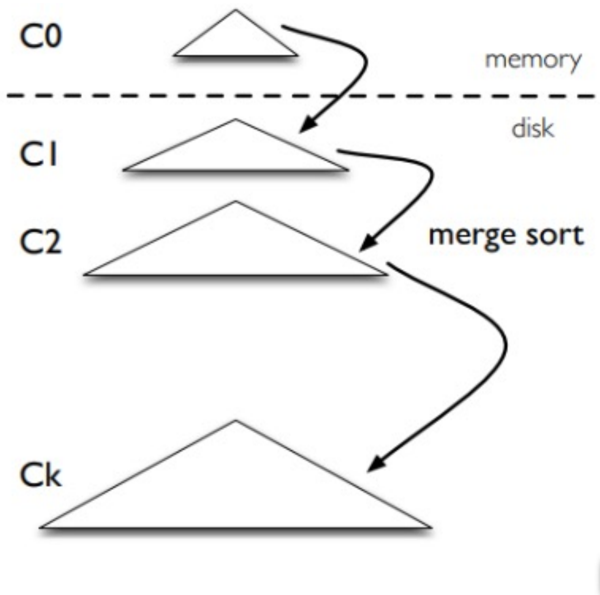
(a) In-place update structure (b) Out-of-place update structure

Fig. 1: Examples of in-place and out-of-place update structures: each entry contains a key (denoted as "k") and a value (denoted as "v")

source: LSM-based Storage Techniques, VLDB Journal 2019

LevelDB, RocksDB basics

- What is LSM (Log Structured Merge)-tree
  - ✓ Merge
    - Do merge sort from levels to a next level for deleting old data
    - All data in sorted order
  - ✓ Tree
    - Larger at lower levels like a tree
    - C0 is in main memory while C1~Ck in storage
    - C: tree's level



(a) LSM-tree

source: Wisckey, FAST 2016

LevelDB, RocksDB basics

- Real implementation in RocksDB (and LevelDB)
  - ✓ Memtable for C0
    - Further separate into mutable and immutable
    - Managed by the skiplist data structure (or hash)
  - ✓ A set of SSTables for C1~Ck (multiple levels, configurable)
    - Default fanout ratio = 10, |Li+1| / |Li|
    - SSTable internals: data block, index block (logically B+-tree)
    - Properties: 1) recent at higher, 2) L0 can be overlapped, while others not
    - Two core internal operations: **flush** and **compaction**
  - ✓ Log (WAL) for durability

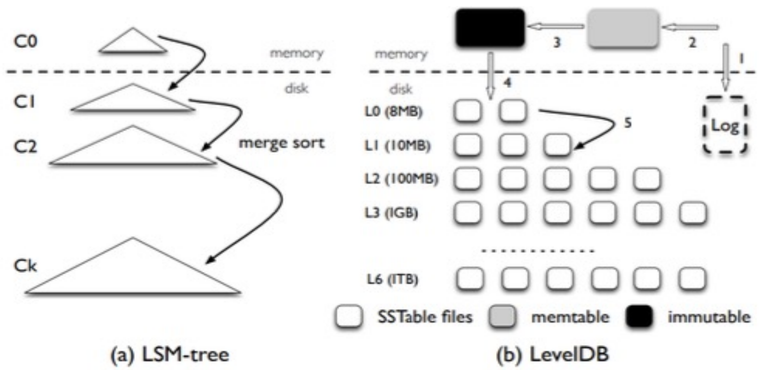


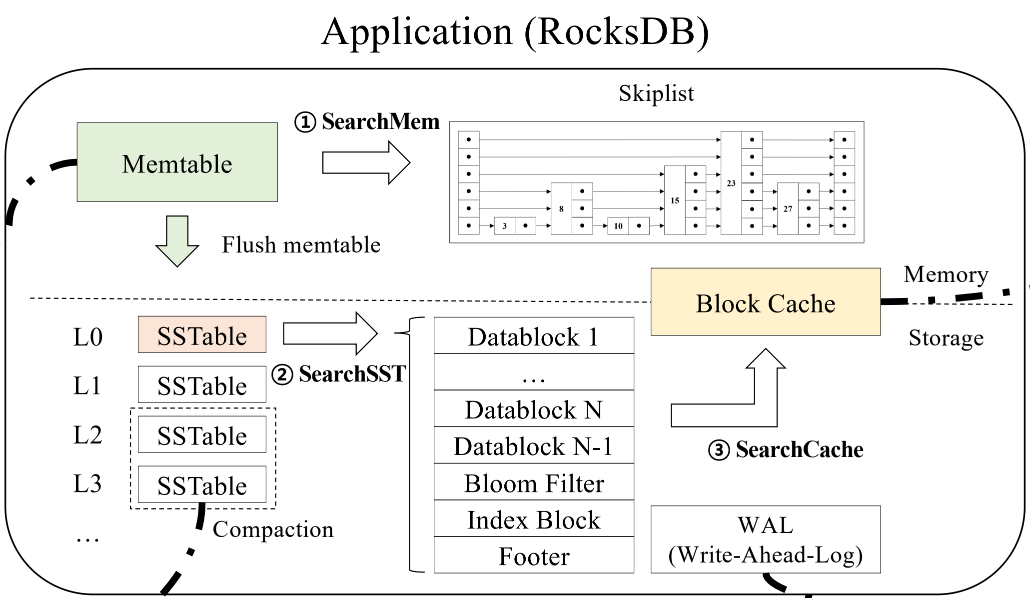
Figure 1: **LSM-tree and LevelDB Architecture.**

source: Wisckey, FAST 2016

LevelDB, RocksDB basics

- RocksDB Architecture Overview

- ❑ Write Buffer
- ❑ Use Skip List Data Structure
- ❑ Write/read must through here
- ❑ Default size (64MB)
- ❑ If full, become immutable



- ❑ Used for read only
- ❑ Freely Configurable size
- ❑ Do after MEM searching

- ❑ Sorted String Table (SST)
- ❑ Have various type of blocks
- ❑ Contains overlapping key ranges
- ❑ Need compaction
  (Space Amplification)

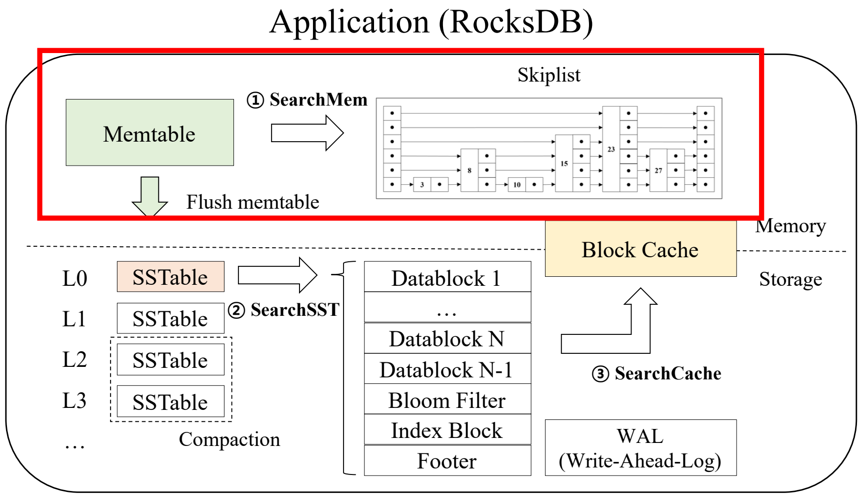
- ❑ Crash Recovery
- ❑ Do every request
- ❑ Do before MEM write

Chapter 3

Core Structure in RocksDB

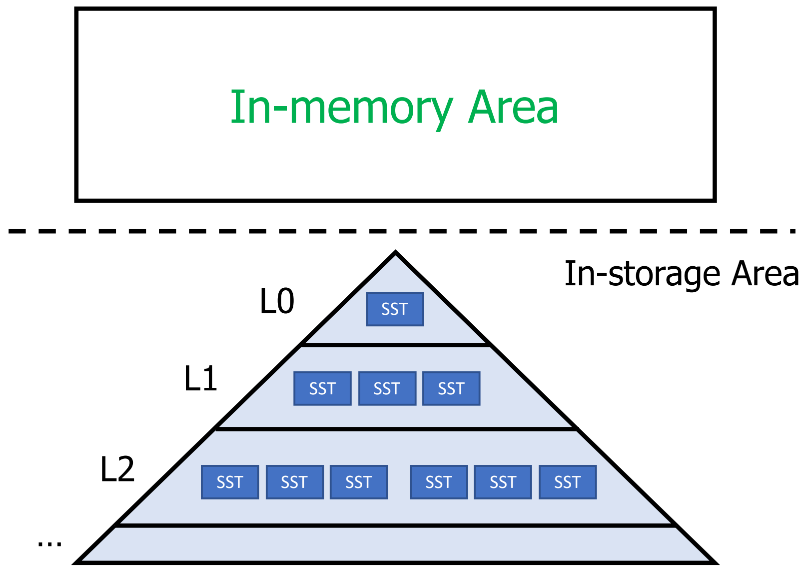
Core Structure in RocksDB

- RocksDB MemTable
  - ✓ An in-memory, write-optimized data structure
  - ✓ Functions
    - When data is written, it is first stored in MemTable
    - Typically implemented using SkipList or HashTable
    - Enables fast writes by performing operations in memory
  - ✓ Advantages
    - Minimizes disk I/O
    - Organizes data before it's flushed to SSTable
  - ✓ Workflow
    - Write (Put) ➔ Add to MemTable ➔ Flush to SSTable (default 64MB)



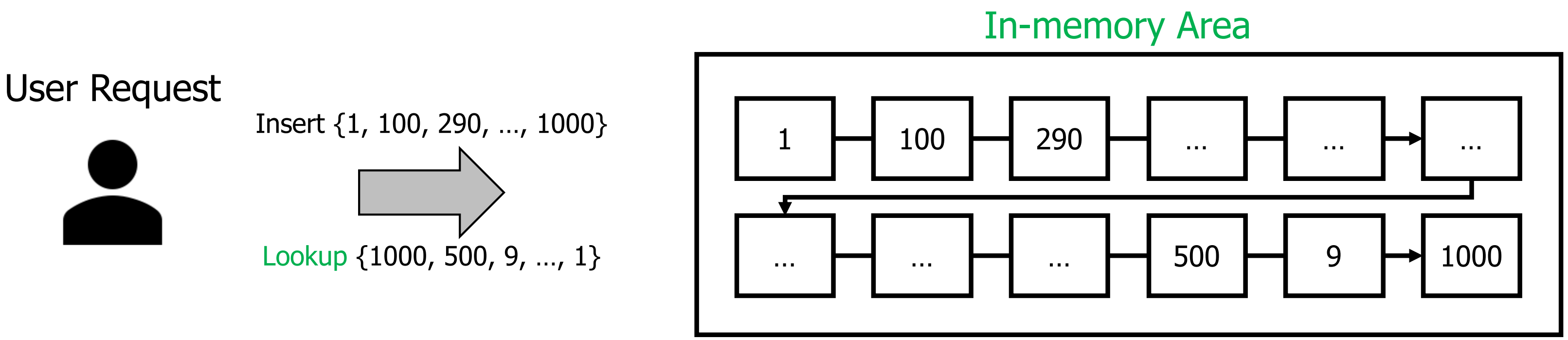
Core Structure in RocksDB

- RocksDB MemTable
  - ✓ Why **In-Memory Index** are necessary
  - ✓ Simple key-value store architecture
    - Two layer: In-memory / In-storage
  - ✓ If there is no indexing structure,
    we cannot retrieve data



Core Structure in RocksDB

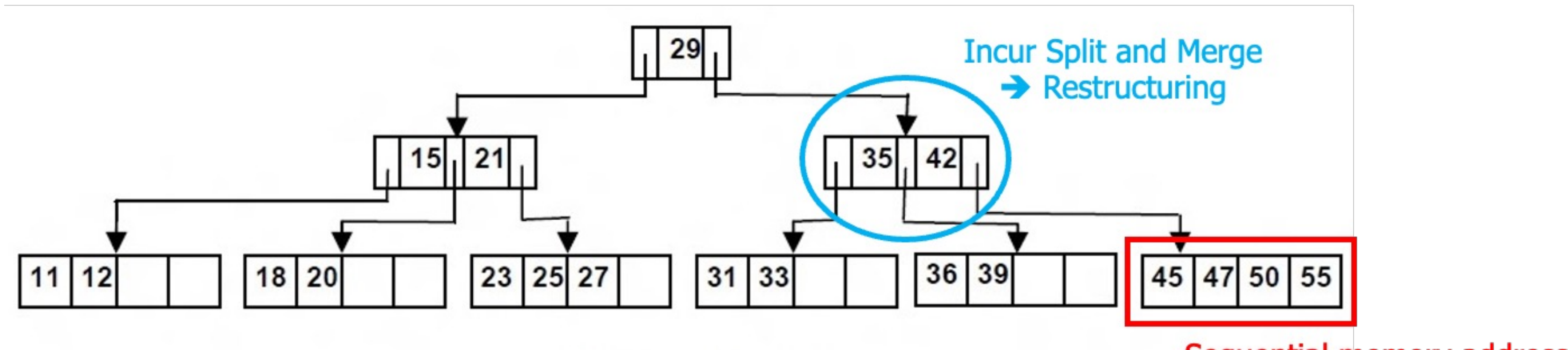
- RocksDB MemTable
  - ✓ Why **In-Memory Index** are necessary
  - ✓ Simple key-value store architecture
    - Assume that we use **List**



Inefficient in both Insert and Lookup: O(N)
➔ Need **In-memory indexing structure**

Core Structure in RocksDB

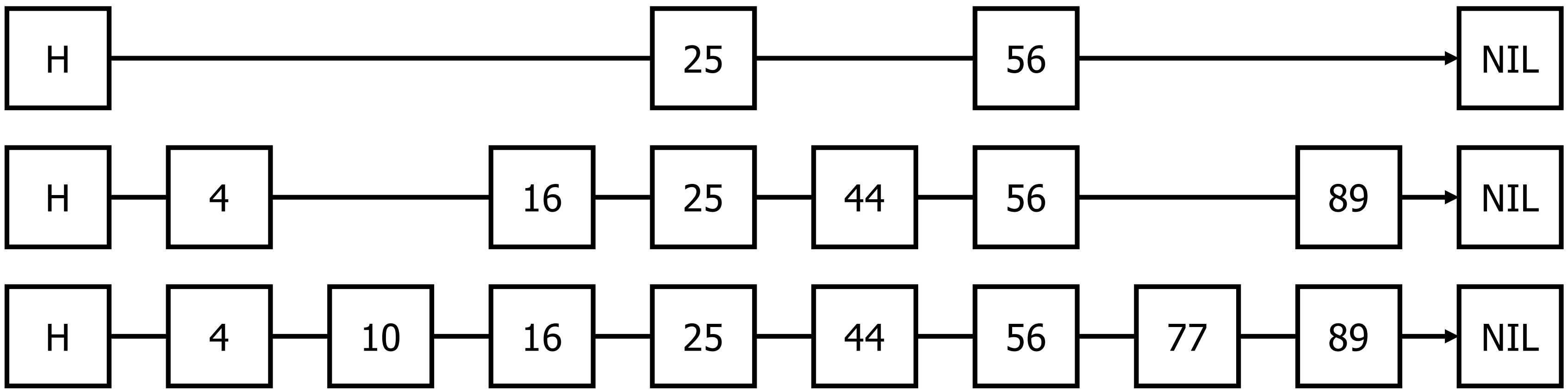
- RocksDB MemTable
  - ✓ Why use **SkipList** in RocksDB
  - ✓ There are many in-memory indexing structures
    - B+tree, skiplist, trie, tree and so on
    - B+tree/B-tree: often used in RDNMS, **O(logN)**



B-Tree of order 5

Core Structure in RocksDB

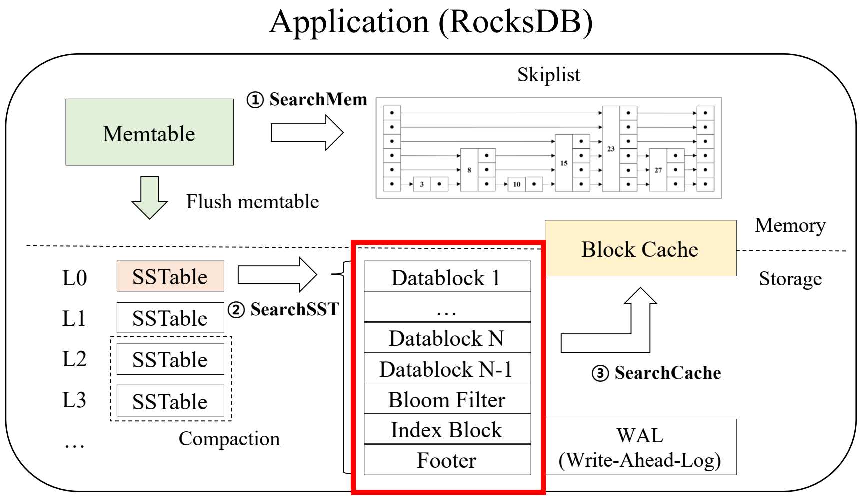
- RocksDB MemTable
  - ✓ Why use **SkipList** in RocksDB
  - ✓ SkipList
    - Maintanance cost is low ➔ Does no require complex adjustment
    - Data is sorted at the whole layer
      ➔ Can be efficiently **flushed** to disk and merged with other disk data structures



SkipList Structure

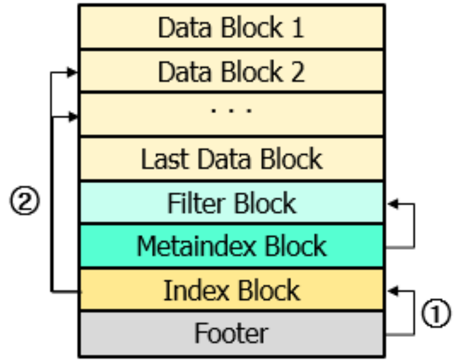
Core Structure in RocksDB

- RocksDB SSTable
  - ✓ An in-storage
  - ✓ Features
    - Immutable file consisting of sorted key-value pairs
    - Permanently stored on disk for durability
    - Supports efficient searches and merges
  - ✓ Structure
    - Datablocks: store key-value pairs
    - Metadata blocks: contains bloom filters and index info.
  - ✓ Advantages
    - Maintains sorted order for fast lookups
    - Optimizes I/O with indexes and bloom filters

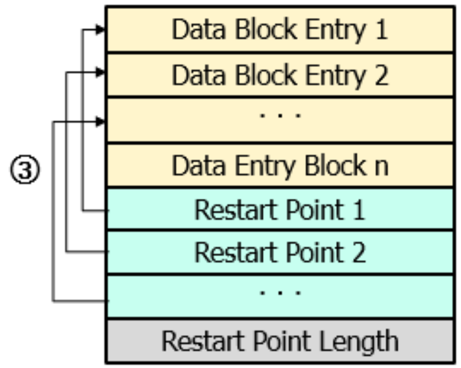


Core Structure in RocksDB

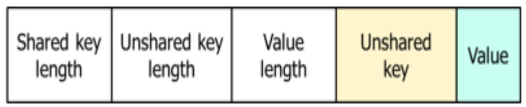
- RocksDB SSTable
  - ✓ SSTable structure
    - Data block: A collection of data entries storing keys and values
    - Filter block: Contains a bloom filter to check the existence of keys
    - Metaindex block: Stores metadata about the filter block
    - Index block: Stores metadata about the data block, including key offsets
    - Footer: Manages offsets for the metaindex block and index block
  - ✓ Data entry structure
    - Shared key length: The length of the key shared with the previous key
    - Unshared key length: The length of the key is not shared
    - Value length: The length of the value
  - ✓ Data lookup process
    - footer ➔ index block offset
    - Index block ➔ data block
    - Restart point ➔ data entry



(SSTable format)



(Data block format)



(Data block entry format)

Homework

Homework

- Today
  - ✓ Team member assignment

- Homework
  - ✓ Install RocksDB (Unitl next study)
    - RocksDB Github: https://github.com/facebook/rocksdb.git
    - Check whether db_bench is running or not
  - ✓ Deciding on a favorite subject (~ 2/4)

No	Topic	Benchmarks	Options	Result
1	WAL/Manifest	--disable_wal --wal_bytes_per_sync	fillseq/random	PPT
2	Memtable	--write_buffer_size --max_file_size	fillseq/random readrandom	PPT
3	Compaction	--base_background_compactions --compaction_style	fillseq/random readseq/random seekrandom	PPT
4	SSTable	--write_buffer_size --max_file_size --block_size	fillseq/random readseq/random seekrandom	PPT
5	Bloom Filter	--bloom_bits	readhot/random seekrandom	PPT
6	Cache	--cache_size --block_size	readhot/random seekrandom	PPT

Compilation

Important: If you plan to run RocksDB in production, don't compile using default `make` or `make all`. That will compile RocksDB in debug mode, which is much slower than release mode.

RocksDB's library should be able to compile without any dependency installed, although we recommend installing some compression libraries (see below). We do depend on newer gcc/clang with C++17 support (GCC >= 7, Clang >= 5).

There are few options when compiling RocksDB:

- [recommended] `make static_lib` will compile librocksdb.a, RocksDB static library. Compiles static library in release mode.
- `make shared_lib` will compile librocksdb.so, RocksDB shared library. Compiles shared library in release mode.
- `make check` will compile and run all the unit tests. `make check` will compile RocksDB in debug mode.
- `make all` will compile our static library, and all our tools and unit tests. Our tools depend on gflags 2.2.0 or newer. You will need to have gflags installed to run `make all`. This will compile RocksDB in debug mode. Don't use binaries compiled by `make all` in production.
- By default the binary we produce is optimized for the CPU you're compiling on (`-march=native` or the equivalent). To build a binary compatible with the most general architecture supported by your CPU and compiler, set `PORTABLE=1` for the build, but performance will suffer as many operations benefit from newer and wider instructions. In addition to `PORTABLE=0` (default) and `PORTABLE=1`, it can be set to an architecture name recognized by your compiler. For example, on 64-bit x86, a reasonable compromise is `PORTABLE=haswell` which supports many or most of the available optimizations while still being compatible with most processors made since roughly 2013.

Dependencies

- You can link RocksDB with following compression libraries:
  - zlib - a library for data compression.
  - bzip2 - a library for data compression.
  - lz4 - a library for extremely fast data compression.
  - snappy - a library for fast data compression.
  - zstandard - Fast real-time compression algorithm.
- All our tools depend on:
  - gflags - a library that handles command line flags processing. You can compile rocksdb library even if you don't have gflags installed.

Homework

- Comments
  - ✓ Paper list uploaded
    - Git: https://github.com/DKU-StarLab/1DanRock.git

Paper & Lecture List

Paper

- SkipList-based
  - William Pugh, "Skip lists: a probabilistic alternative to balanced trees", Communications of the ACM 1990
  - Zhongle Xie, et al. "Parallelizing Skip Lits for In-Memory Multi-Core Database Systems", ICDE 2017
  - Jeseong Yeon, et al. "JellyFish: A Fast Skip List with MVCC", Middleware '20
  - Tyler Crain, et al. "No Hot Spot Non-blocking Skip List", ICDCS 2013
  - Henry Daly, et al. "NUMASK: High-Performance Scalable Skip List for NUMA", DISC 2018
  - Yedam Na, et al. "ESL: A High-Performance Skiplist with Express Lane", MDPI 2023
  - Zhongle Xie, et al. "PI: a parallel in-memory skip list based index", CoRR 2016
  - Tadeusz Kobus, et al. "Jiffy: a lock-free skip list with batch updates and snapshots", PPoPP '22
  - Vitaly Aksenov, et al. "The splay-list: a distribution-adaptive concurrent skip-list", Distributed Computing 2023
- Key-value Separation
  - Dai, Yifan, et al. "From WiscKey to Bourbon: A Learned Index for Log-Structured Merge Trees." FAST 16
  - Chan, Helen HW, et al. "HashKV: Enabling Efficient Updates in KV Storage via Hashing." ATC 18
  - Li, Yongkun, et al. "Differentiated Key-Value storage management for balanced I/O performance." ATC 21
  - Tang, Chenlei, et al. "Fencekv: Enabling efficient range query for key-value separation." IEEE TPDS 22
- Storage/PM(Persistent Memory)-based
  - Kannan, Sudarsun, et al. "Redesigning LSMs for Nonvolatile Memory with NoveLSM." FAST 18
  - Yao, Ting, et al. "MatrixKV: Reducing Write Stalls and Write Amplification in LSM-tree Based KV Stores with Matrix Container in NVM." ATC 20
  - Ding, Chen, et al. "TriangleKV: Reducing write stalls and write amplification in LSM-tree based KV stores with triangle container in NVM." IEEE TPDS 22
  - Zhang, Wenhui, et al. "ChameleonDB: a key-value store for optane persistent memory." EuroSys 21
  - Fernando, Pradeep, et al. "Blizzard: Adding True Persistence to Main Memory Data Structures." arXiv 23

Homework

- Next week
  - ✓ RocksDB Operations
    - Put, Get, Flush, WAL, Compaction, Bloom filter, Cache
  - ✓ LevelDB with Key/Value Separation and Learned Index
  - ✓ How to analyze RocksDB
  - ✓ Introduction to Tools for Analysis
    - uftrace, linux perf tool, vscode, gdb

NoSQL Database
1st week

Hojin Shin, Guangxun Zhao

http://sslab.dankook.ac.kr/, https://sslab.dankook.ac.kr/~choijm

Thank You
Q & A ?

Presentation by Hojin Shin
hojin03s@dankook.ac.kr