SWE3033:
Database Project

RocksDB: An Introduction

Bo-Hyun Lee
lia323@skku.edu

RocksDB Introduction

- **A persistent key-value store for fast storage**
  - Exploiting the full potential of high read/write rates offered by *flash or RAM*
- A **log structured** database engine, written entirely in C++
- **Open-source**, based on LevelDB 1.5
- Focusing on **performance** and **scalability**
  - Optimized for Server workloads

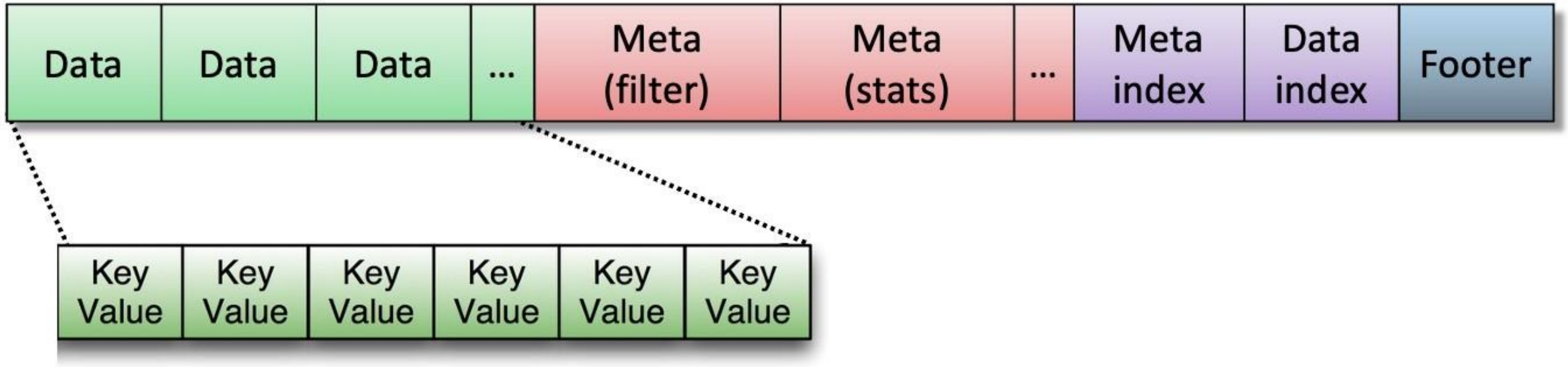
Three Basic Components of RocksDB

1. **Memtable**
   - In-memory data structure
   - A buffer, temporarily host the incoming writes

2. **Logfile**
   - Sequentially written file on storage

3. **SSTable (=SST file)**
   - Sorted sequence table on storage → to facilitate easy lookup of keys
   - A file which contains a set of arbitrary, sorted **key-value pairs** inside
   - Organized in levels
   - Immutable in its lifetime

Block Based Table

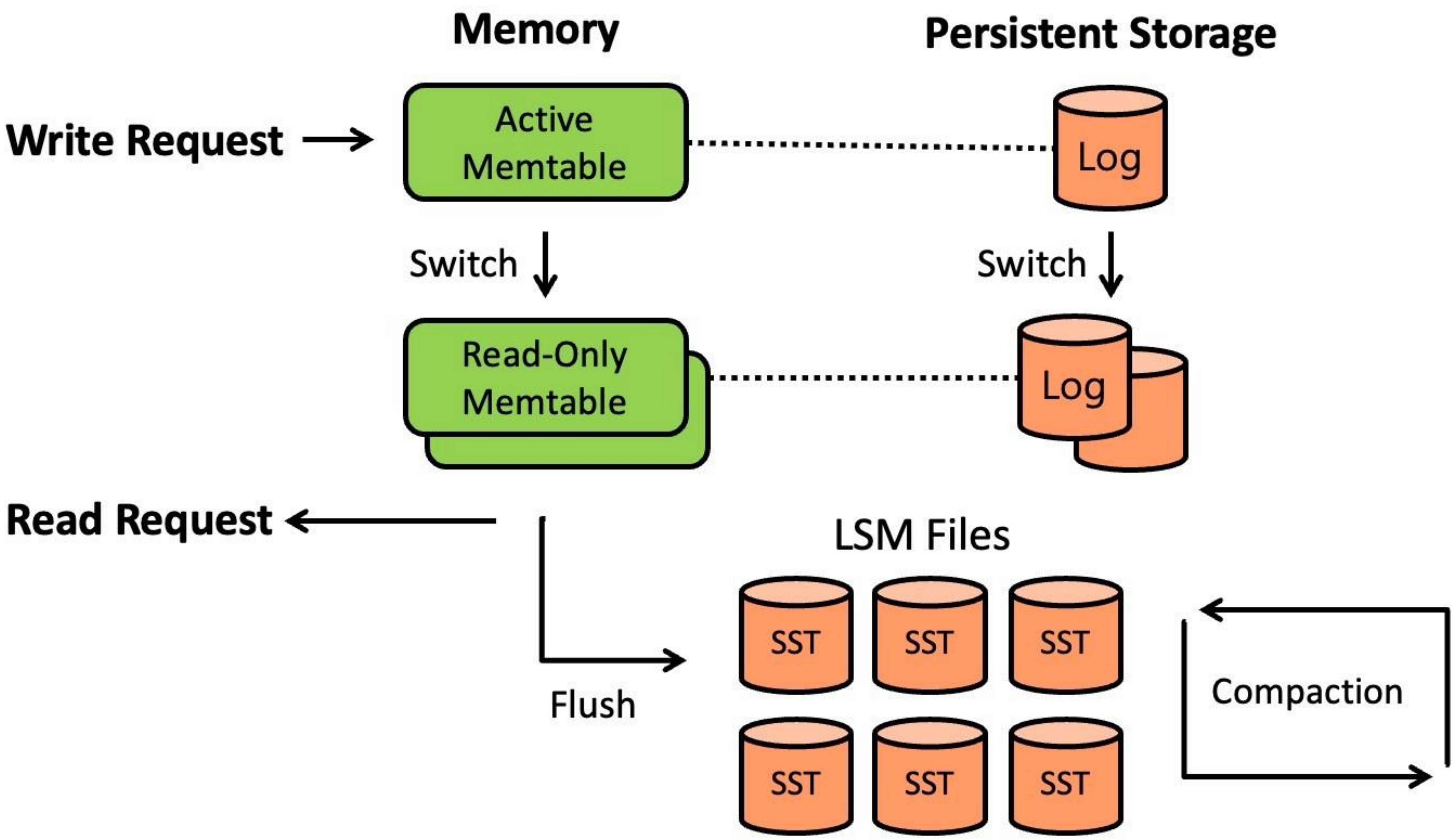


- The **default SST table format** in RocksDB
- The sequence of key/value pairs in the file are sorted in key order
  - They are partitioned into a sequence of data blocks

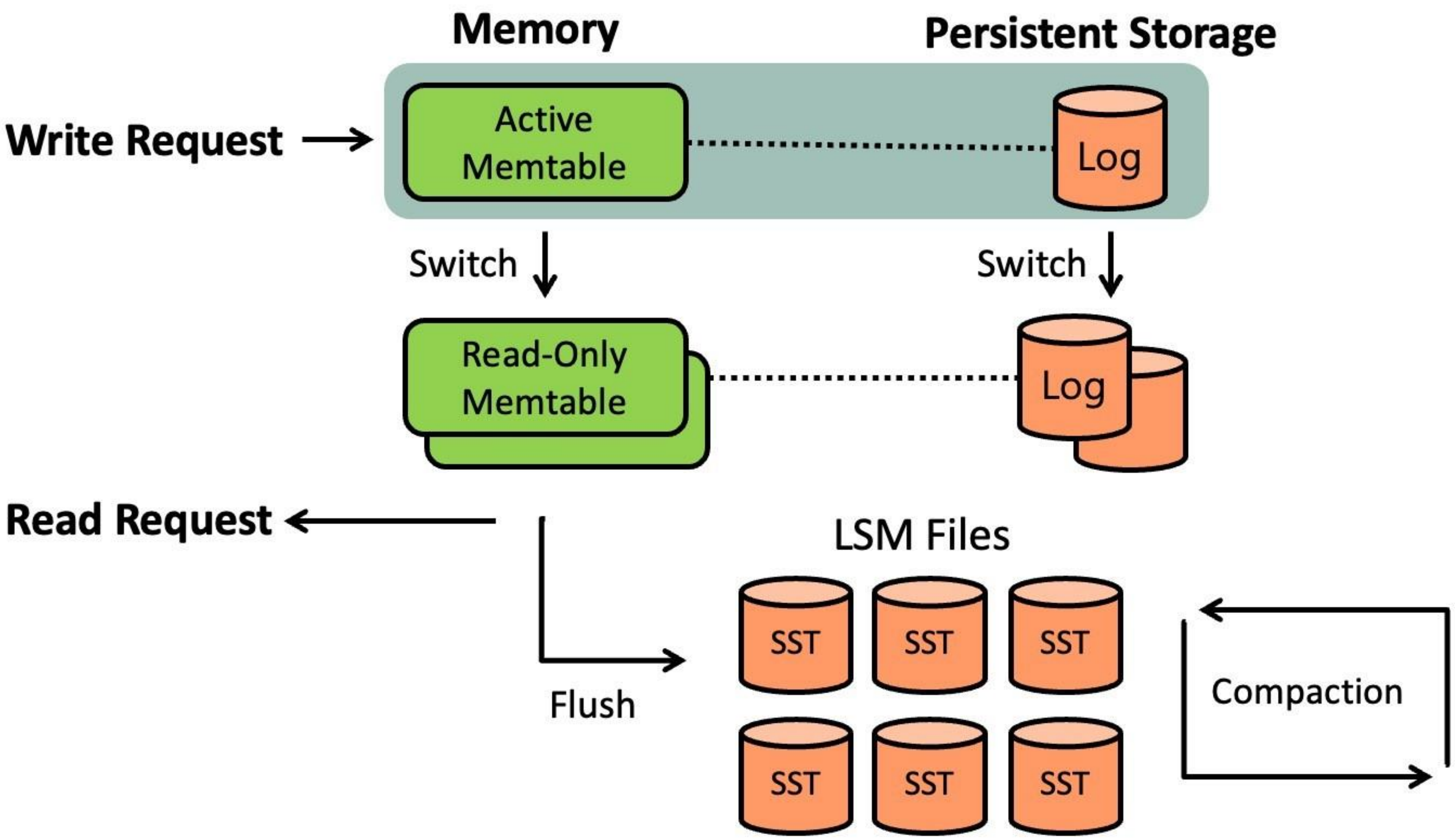
Memtable and SST File

- On-disk SSTable indexes are always loaded into memory
- All writes go directly to the memtable index
- **Reads** check the memtable first → then, the SSTable indexes
- Periodically, the memtable is flushed to disk as an SSTable
- Periodically, on-disk SSTables are merged
    - Update/delete records will overwrite/remove the older data

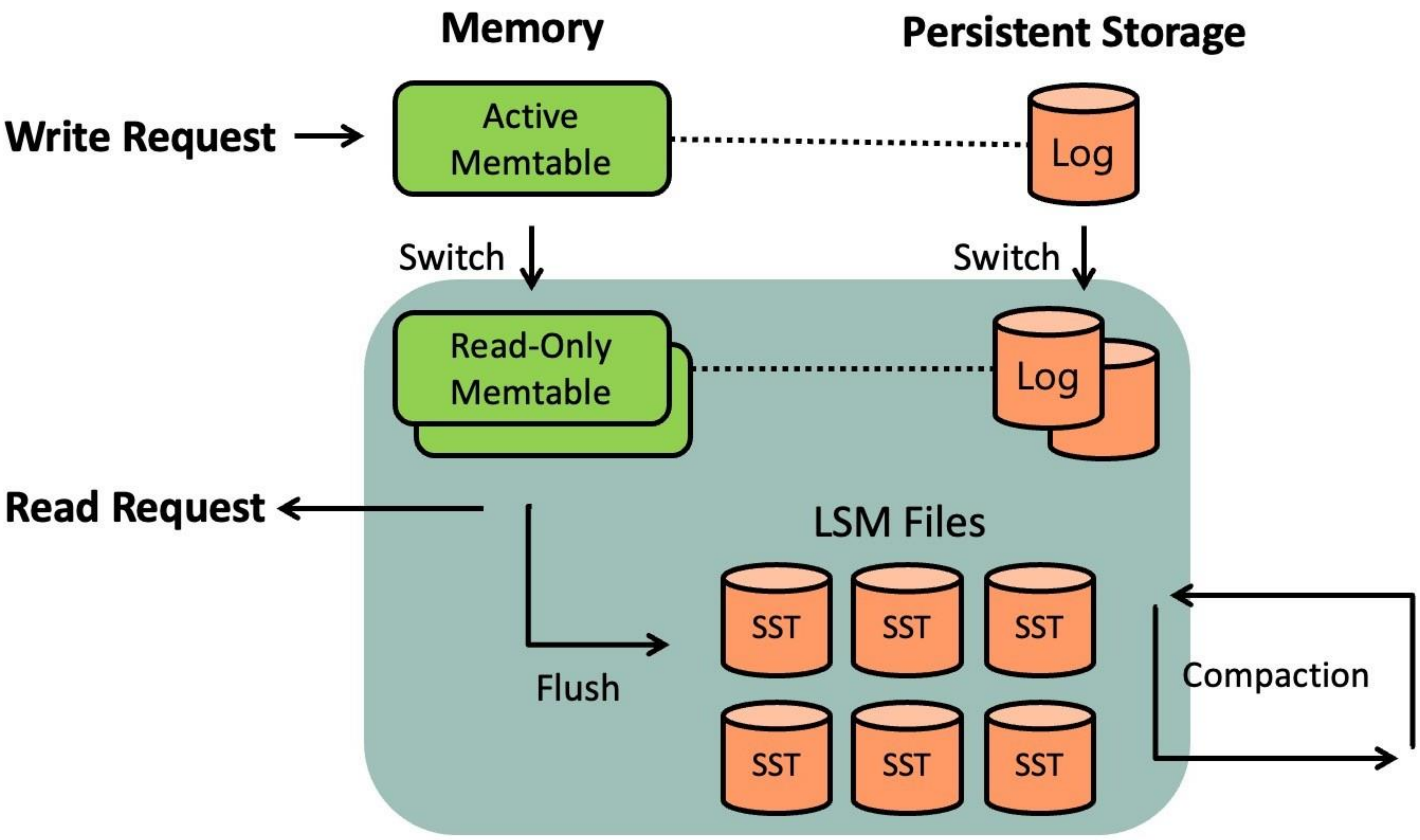
RocksDB Architecture



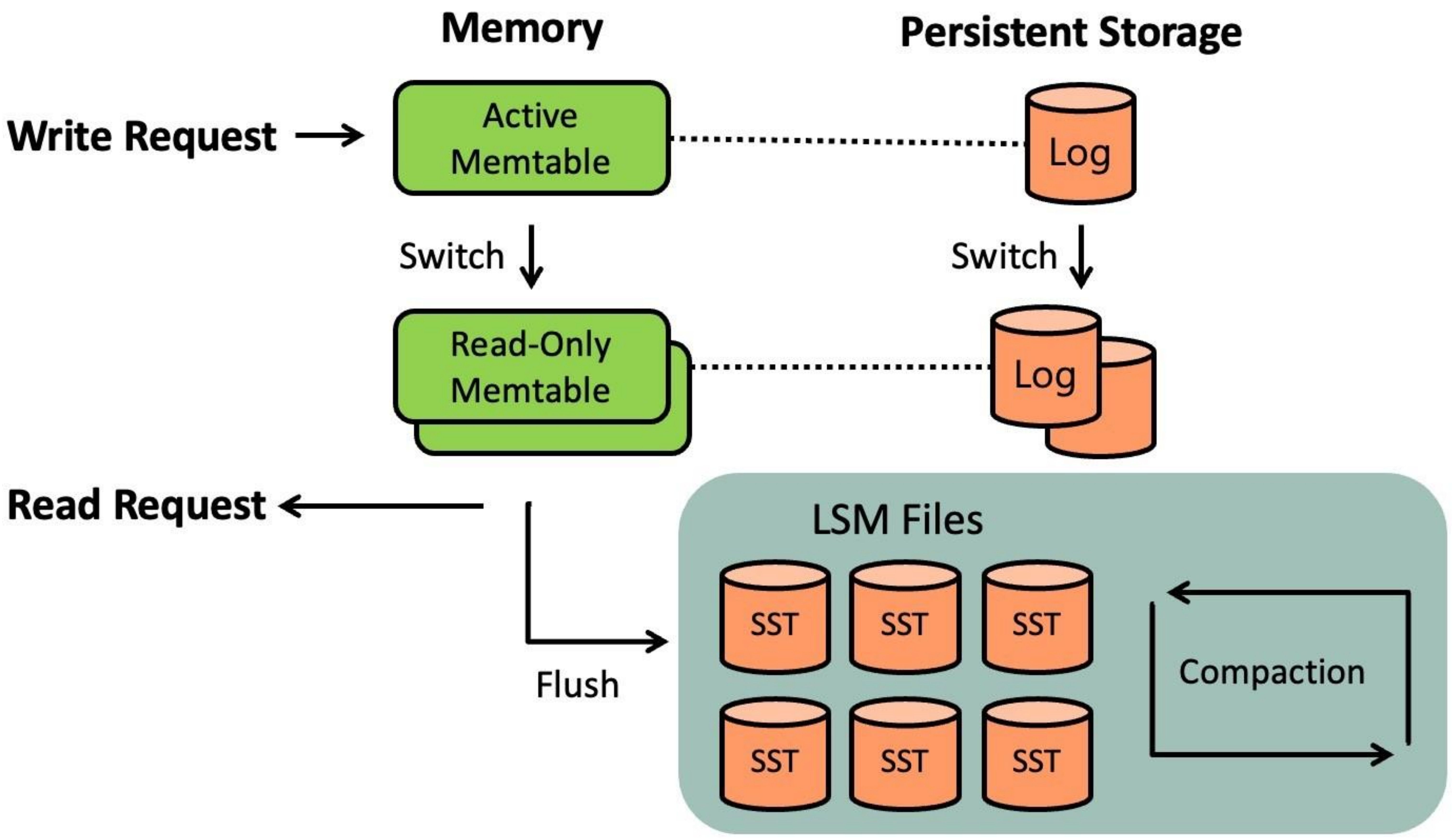
RocksDB Architecture



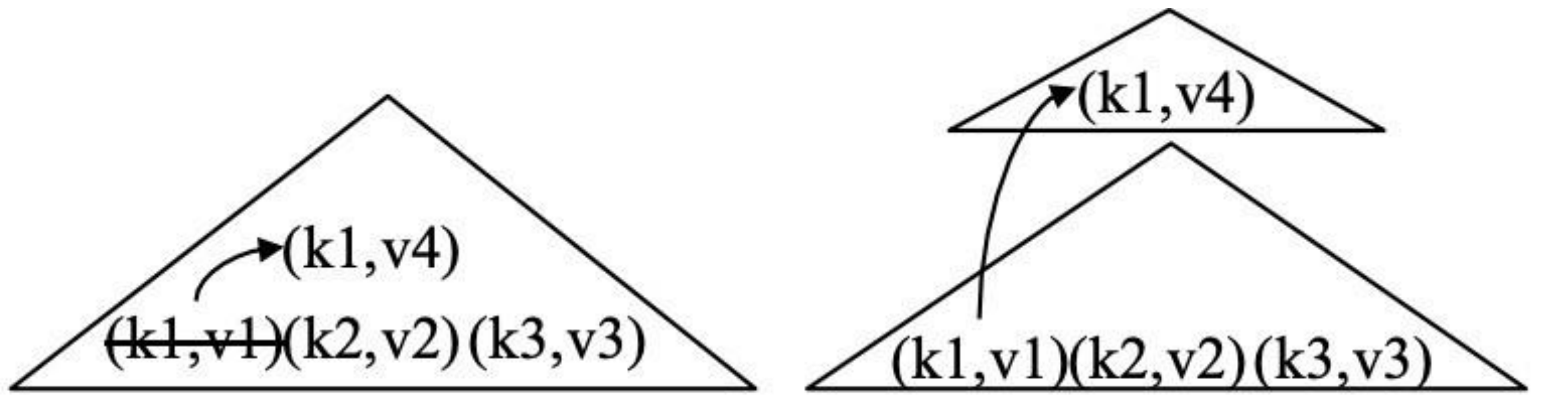
RocksDB Architecture



RocksDB Architecture



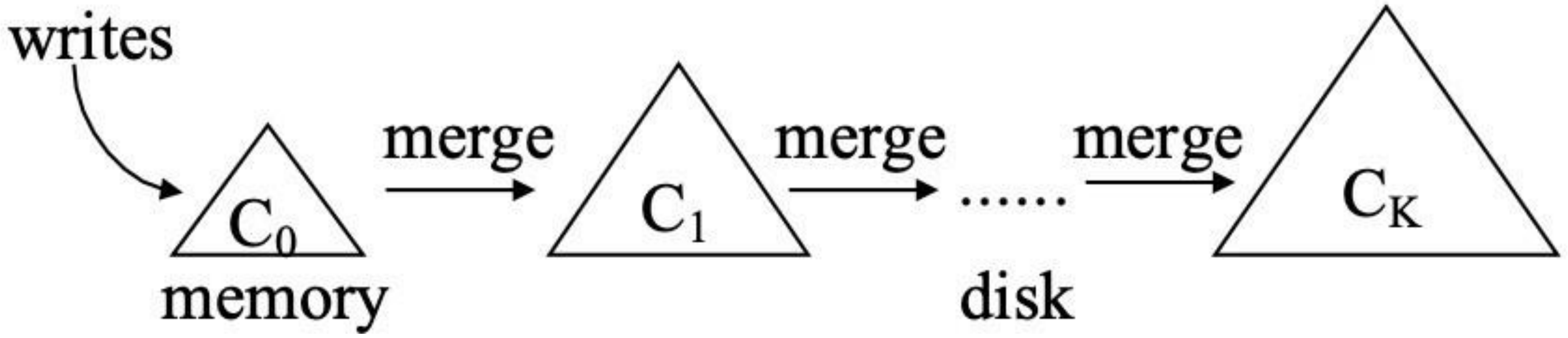
B+ Tree



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

- *B+Tree* directly overwrites old records to store new updates → **In-place update**
- This structure is **read-optimized** since only the most recent version of each record is stored
- However, this design sacrifices write performance as **updates incur random I/Os**

Log-Structured Merge Tree



- LSM-Tree → **Out-of-place update**
  - **N-level merge trees**
  - Splitting **a logical tree** into **several physical pieces**
  - So that the *most-recently-updated* portion of data is in a tree in **memory**
  - **Transform random writes into sequential writes** using *logfile* and in-memory store (*memtable*)

Install RocksDB and Run DB_Bench

- This week, you will install RocksDB and run DB_Bench on your own system

- Refer to github link for week 6 experiment guide

- Please send an email to lia323@skku.edu if you have any question about this week’s lecture or experiment

Reference

- Facebook, “RocksDB”, http://rocksdb.org

- O'Neil, Patrick E., Edward Y. C. Cheng, Dieter Gawlick and Elizabeth J. O'Neil. “The log-structured merge-tree (LSM-tree).” Acta Informatica 33 (2009): 351-385.

- Chen Luo, Michael J. Carey, “LSM-based Storage Techniques: A Survey”. VLDB Journal, 2019

- Dhruba Borthakur and Hadbo Xu, “The Story of RocksDB”, SlideShare, https://www.slideshare.net/HiveData/tech-talk-rocksdb-slides-by-dhruba-borthakur-haobo-xu-of-facebook

- Mijin An, “RocksDB detail”, SlideShare, https://www.slideshare.net/meeeejin/rocksdb-detail

- Facebook, “RocksDB Wiki”, GitHub, https://github.com/facebook/rocksdb/wiki