SWE3033:
Database Project

# Space Amplification in DBMS

Bo-Hyun Lee
lia323@skku.edu

# Contents

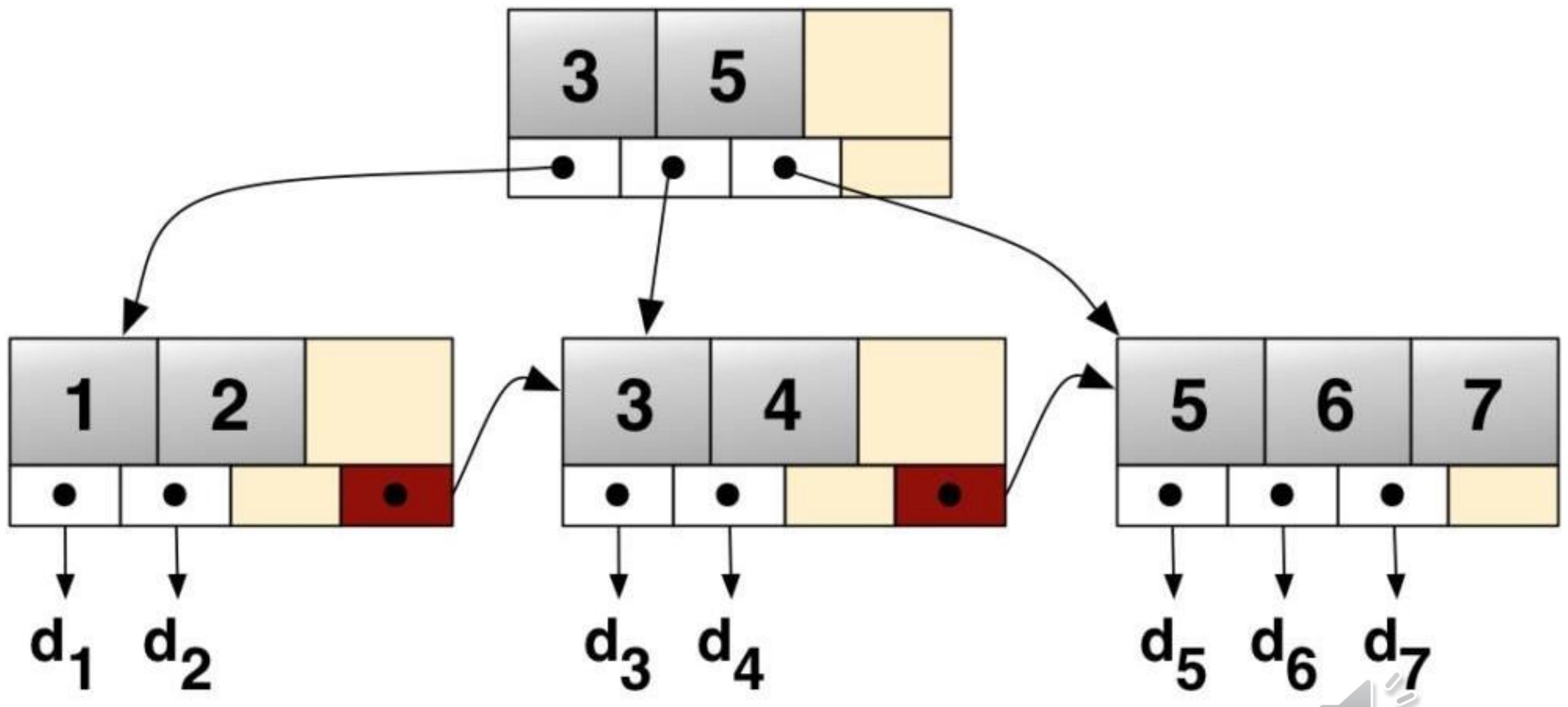
- Space utilization in MySQL/InnoDB and its implications
- Two types of Compactions and Space utilization in RocksDB

# MySQL/InnoDB Space Management

# MySQL/InnoDB on Flash Storage

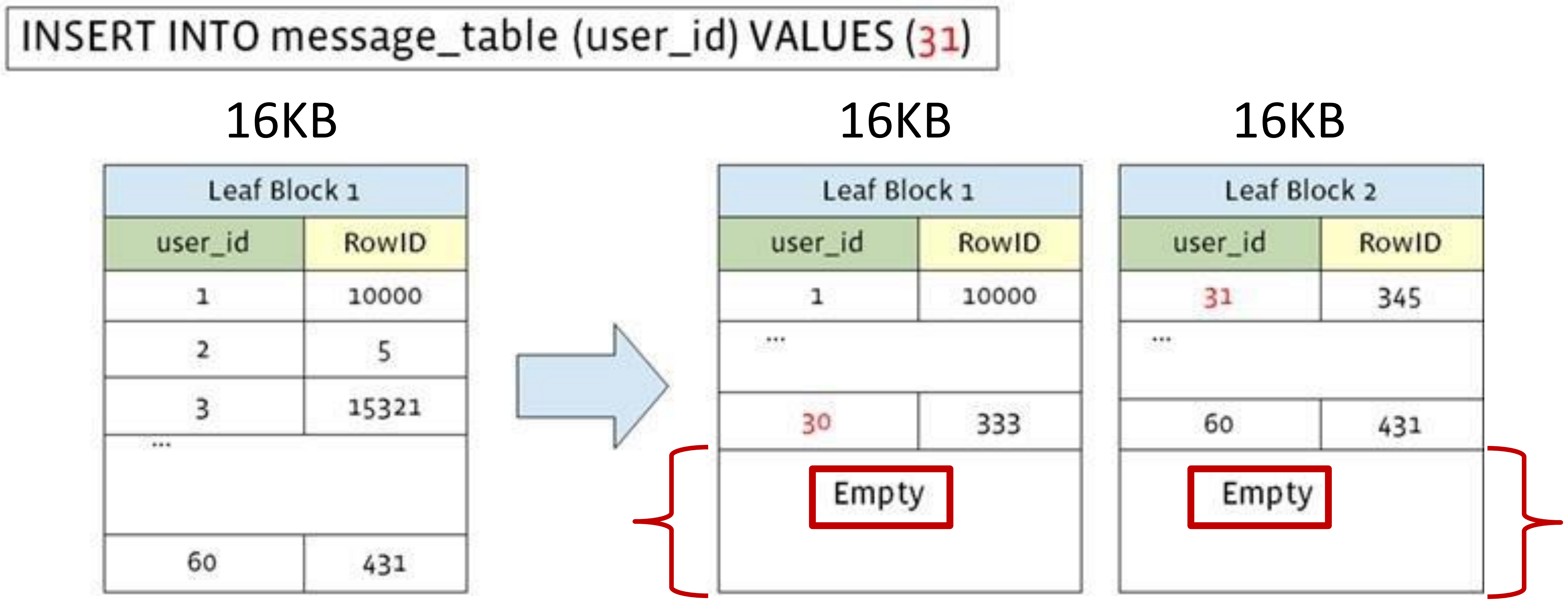
- The standard index data structure of DBMS: **B+Tree**
  - Fixed-size page
  - In-place update → incurs random I/O

- MySQL on SSDs:
  - Great for performance and reliability
  - Some **inefficiencies** in **space amplification**
  - Space grows by node split

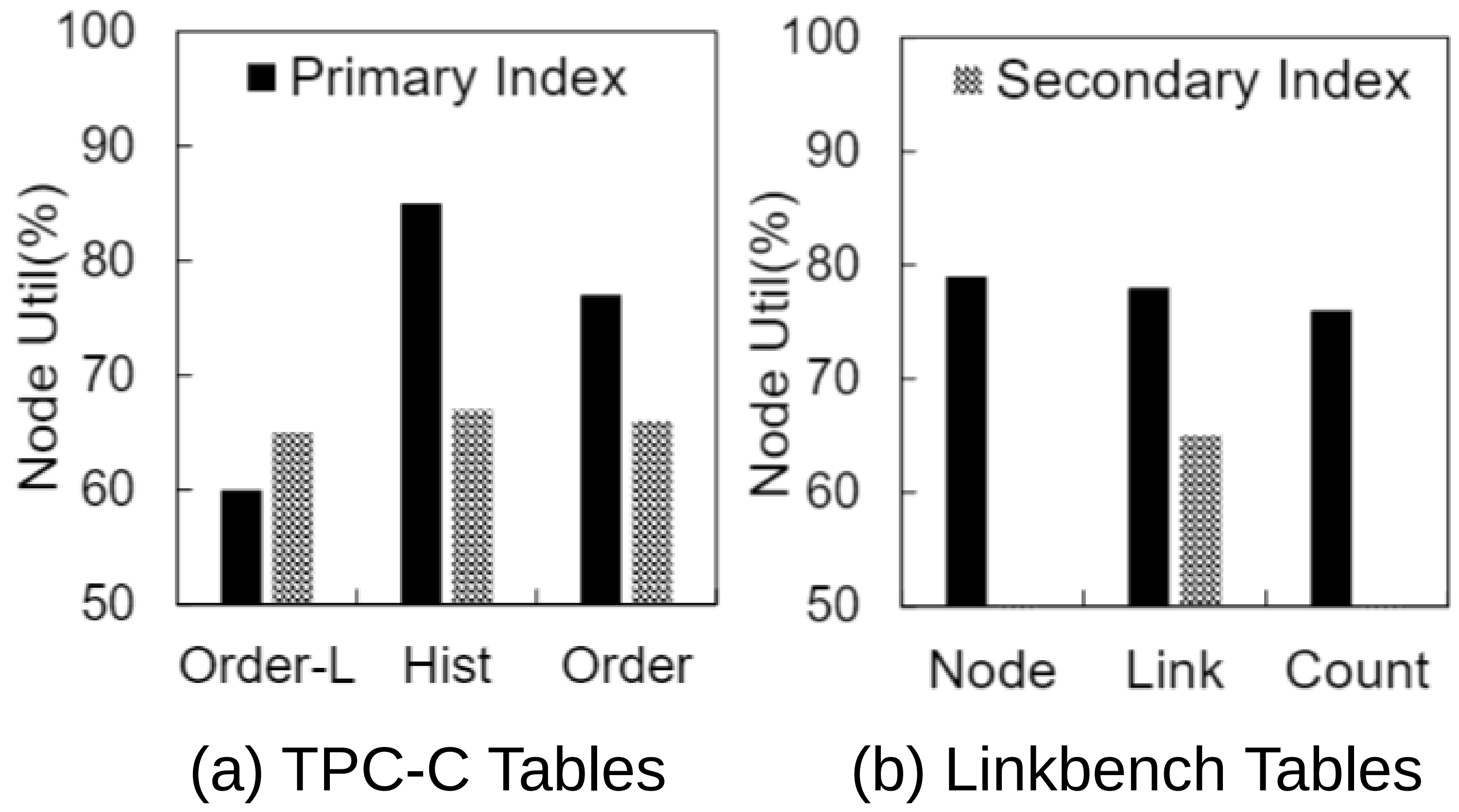


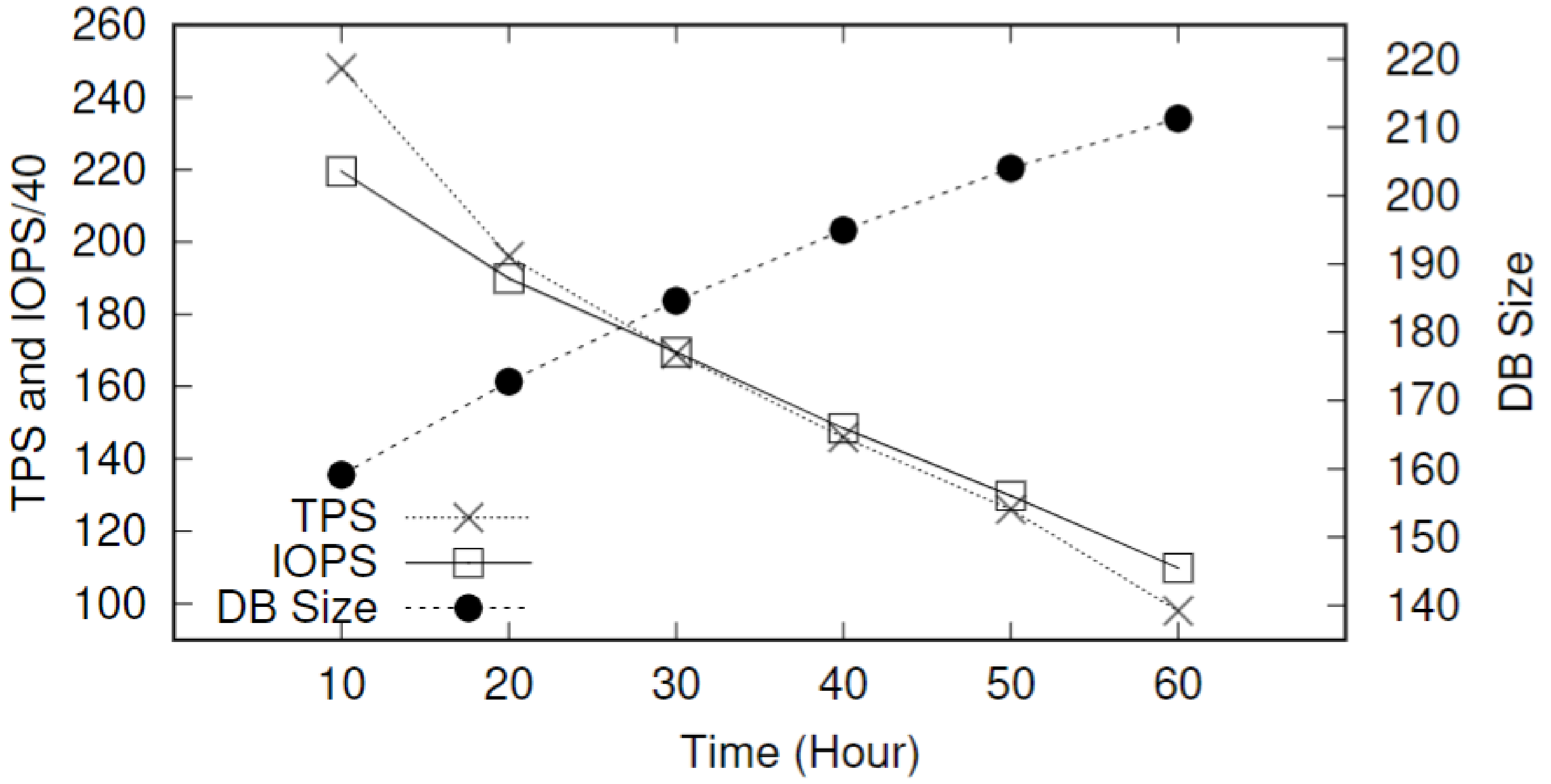
# MySQL/InnoDB Space Utilization

- $\color{red}{\mathbf{\textit{Space Amplification}}} = \dfrac{\textbf{\textit{The size of the database}}}{\textbf{\textit{The size of the data in the database}}}$
- **Low space utilization**
  - The avg. fill factor of B-Tree is typically below ***70%*** due to B+tree node split mechanism
  - Acceptable on cheap disks but *wasteful* on <u>expensive SSDs</u>



# MySQL on Flash Storage: Space Amplification

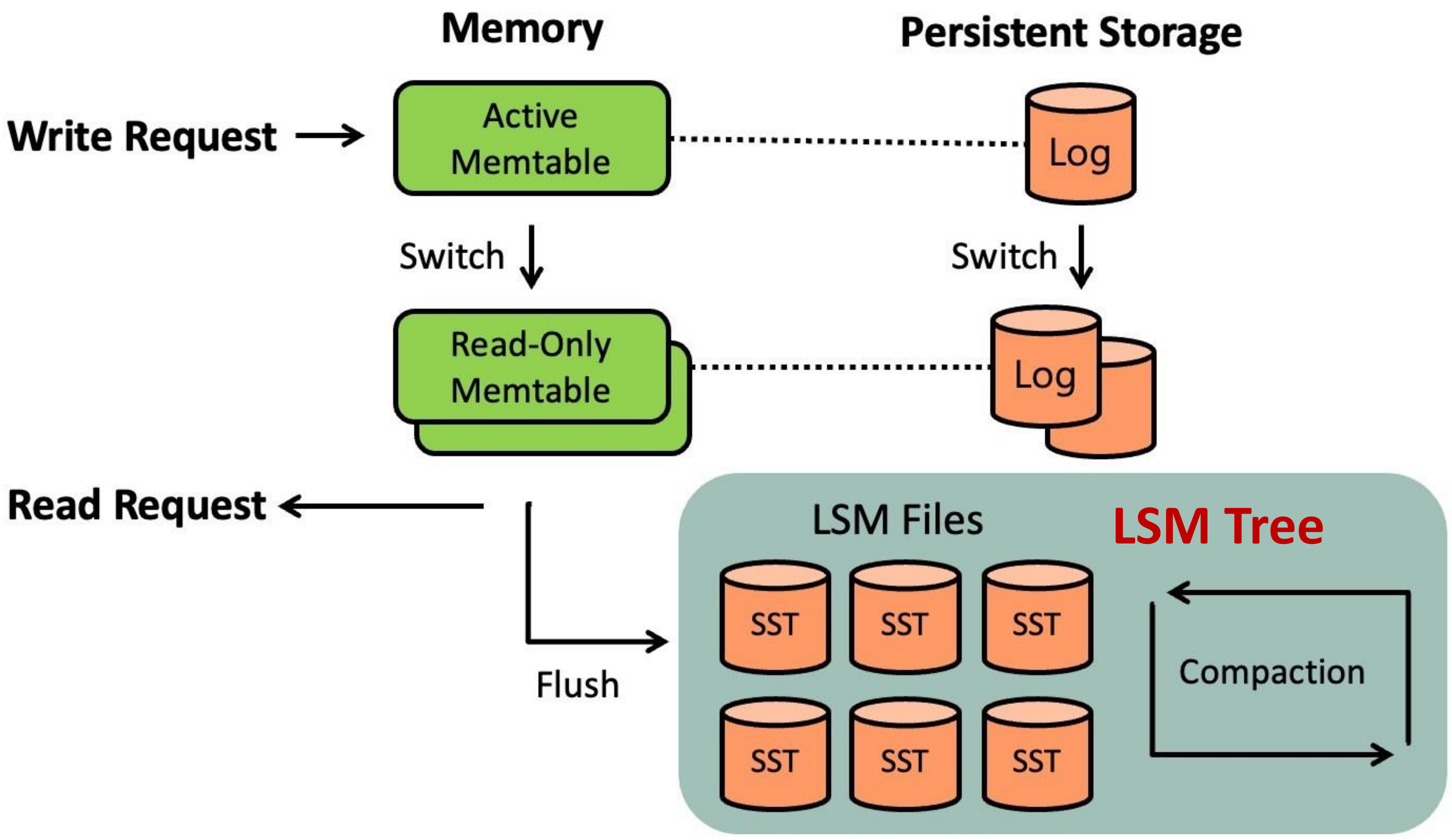


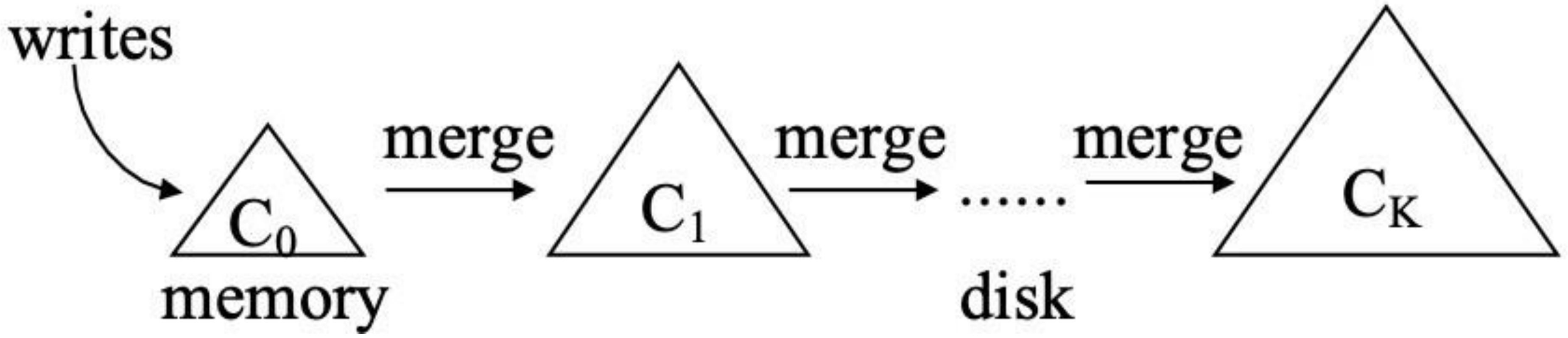

- Space amplification in real-world workloads
  - E.g., space amplification of Order-Line Primary Key B+tree is 1.67
- If DB size grows close to the SSD capacity…
  - Transaction throughput drops
  - IOPS exacerbates proportional to TPS
- It is crucial to efficiently use storage space in terms of **cost** as well as **performance**!

# RocksDB Space Management

# RocksDB Architecture



# 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 *memtable*
  - Duplicate or invalidated key-value data are processed by **compaction**

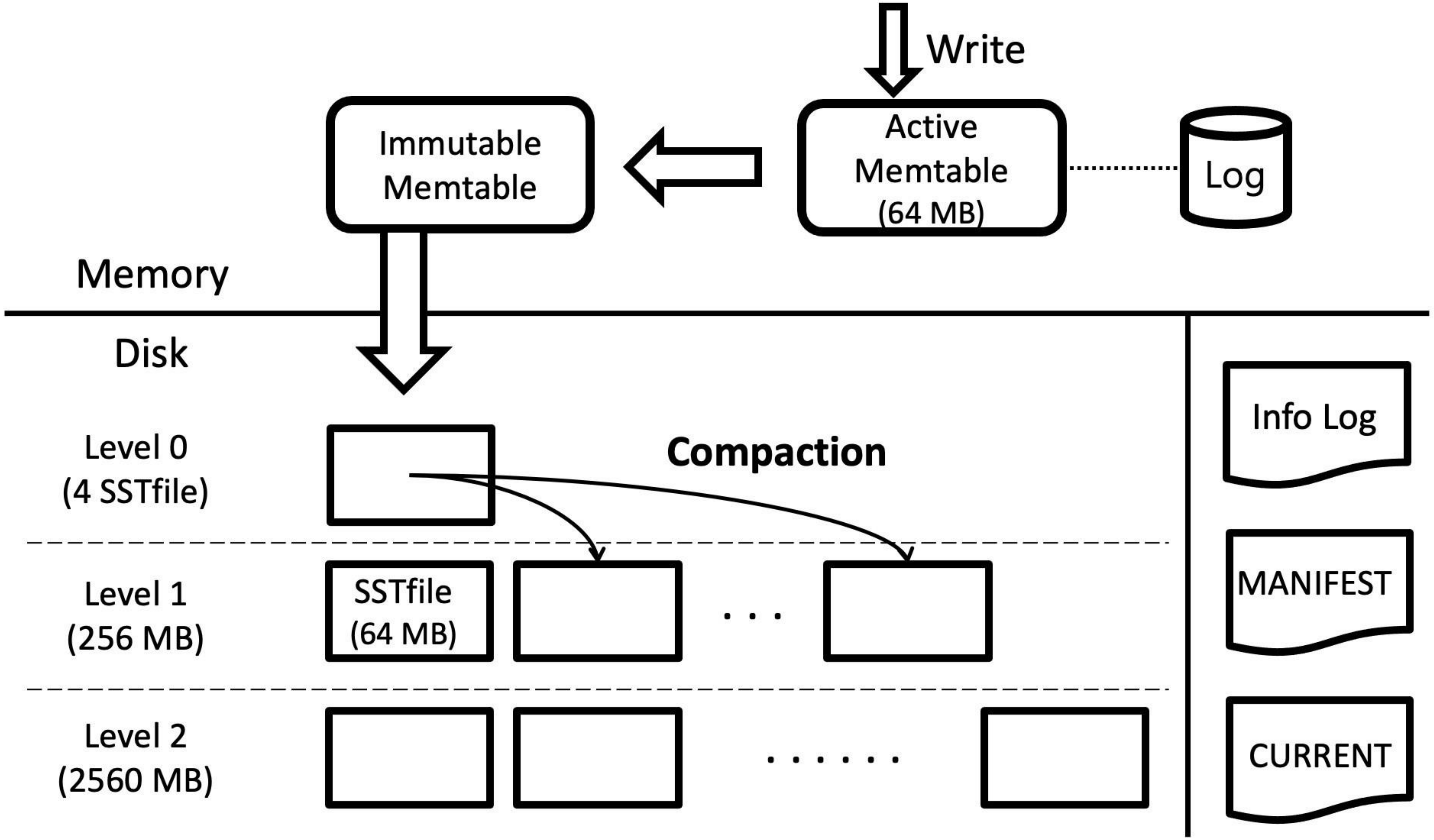
# RocksDB Compaction

- Compaction algorithms constrain the LSM tree shape

- What is *"compaction"*?
  - **Remove** multiple copies of the same key (**Duplicate** or **overwritten** keys due to out-of-place update)
  - Process **deletion** of keys
  - **Merge** SST files to a bigger SST file

- RocksDB supports two different styles of compaction:
  - Leveled compaction
  - Universal compaction

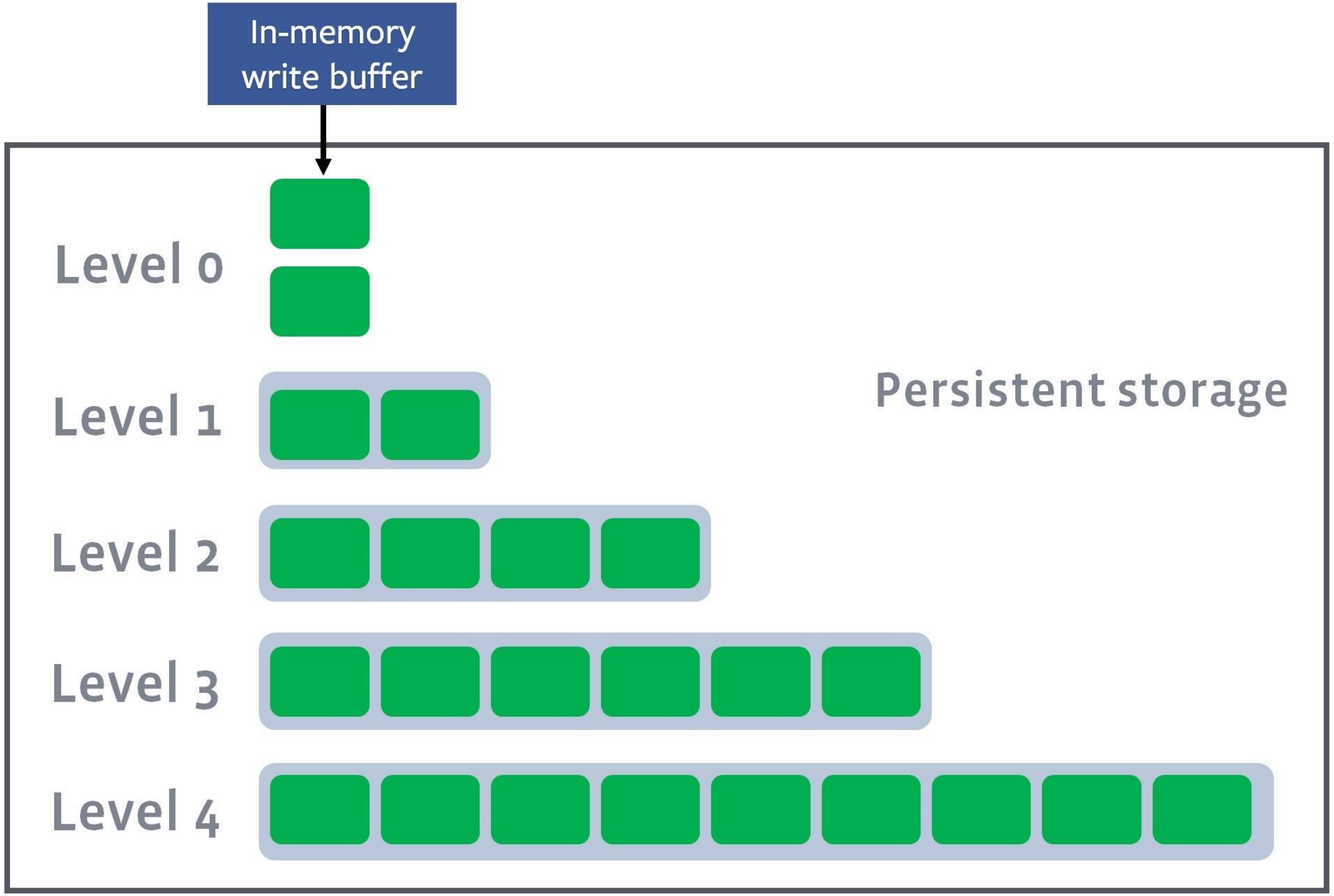
# Leveled Architecture of RocksDB



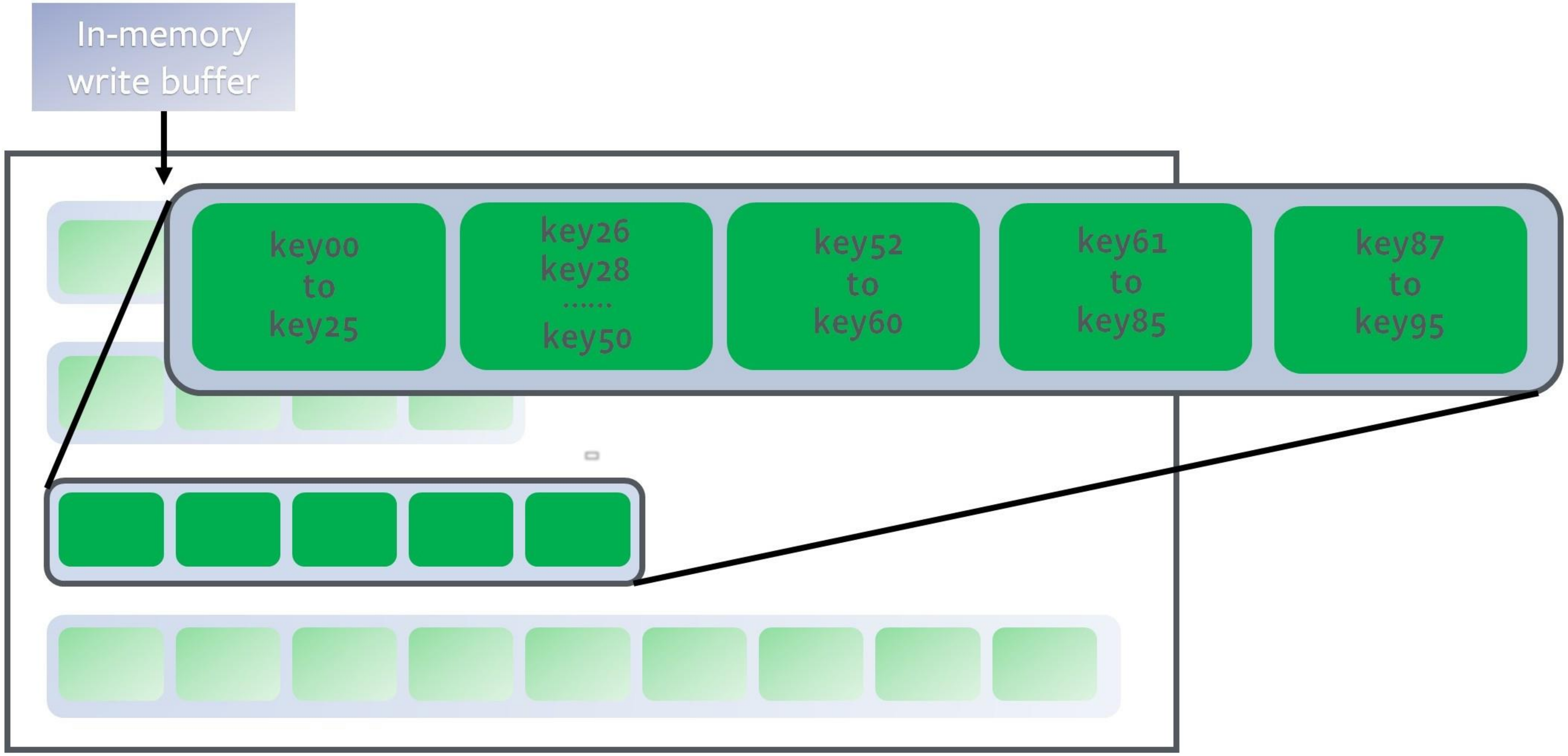
# Leveled Compaction

- RocksDB's **default** compaction style
- It stores data in *multiple levels* in the database
  - More **recent** data -> $L_0$
  - The **oldest** data -> $L_{\text{max}}$
- Each level is 10 times larger than the previous one by default (configurable)
- Files in $L_0$: **Overlapping** keys, sorted by **flush time**
- Files in $L_1 \sim L_{\text{max}}$: **Non-overlapping** keys, sorted by **key**

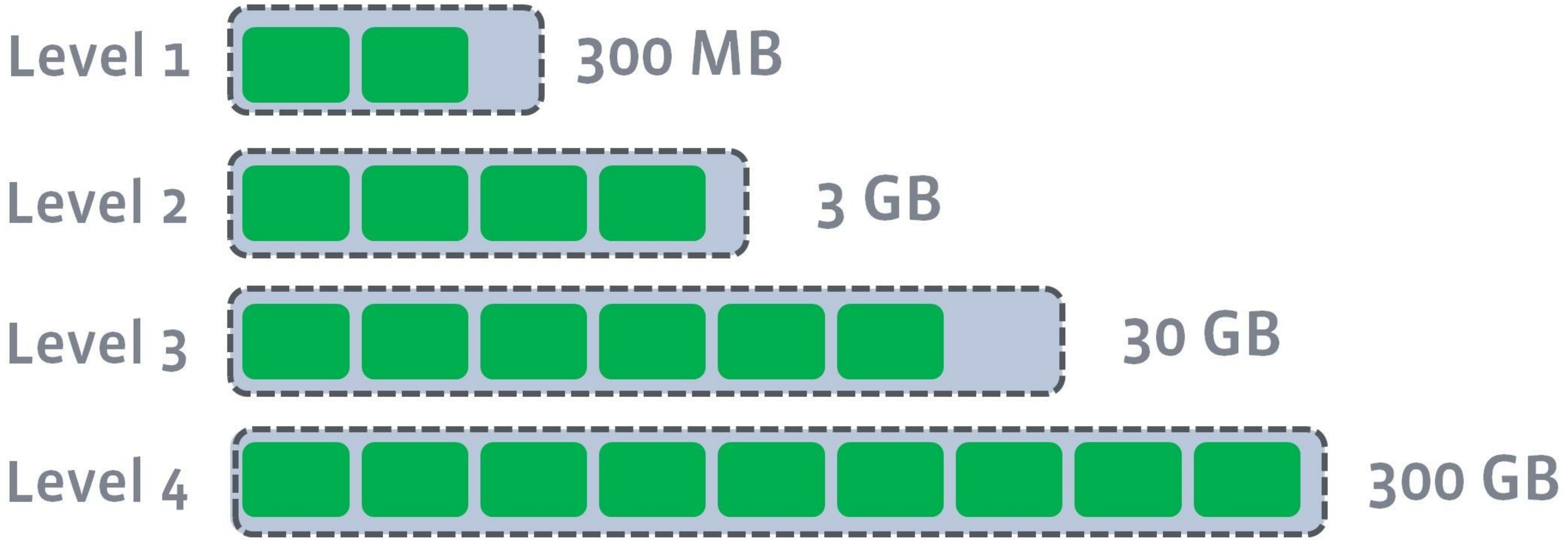
# Leveled Compaction: Structure of Files (1)



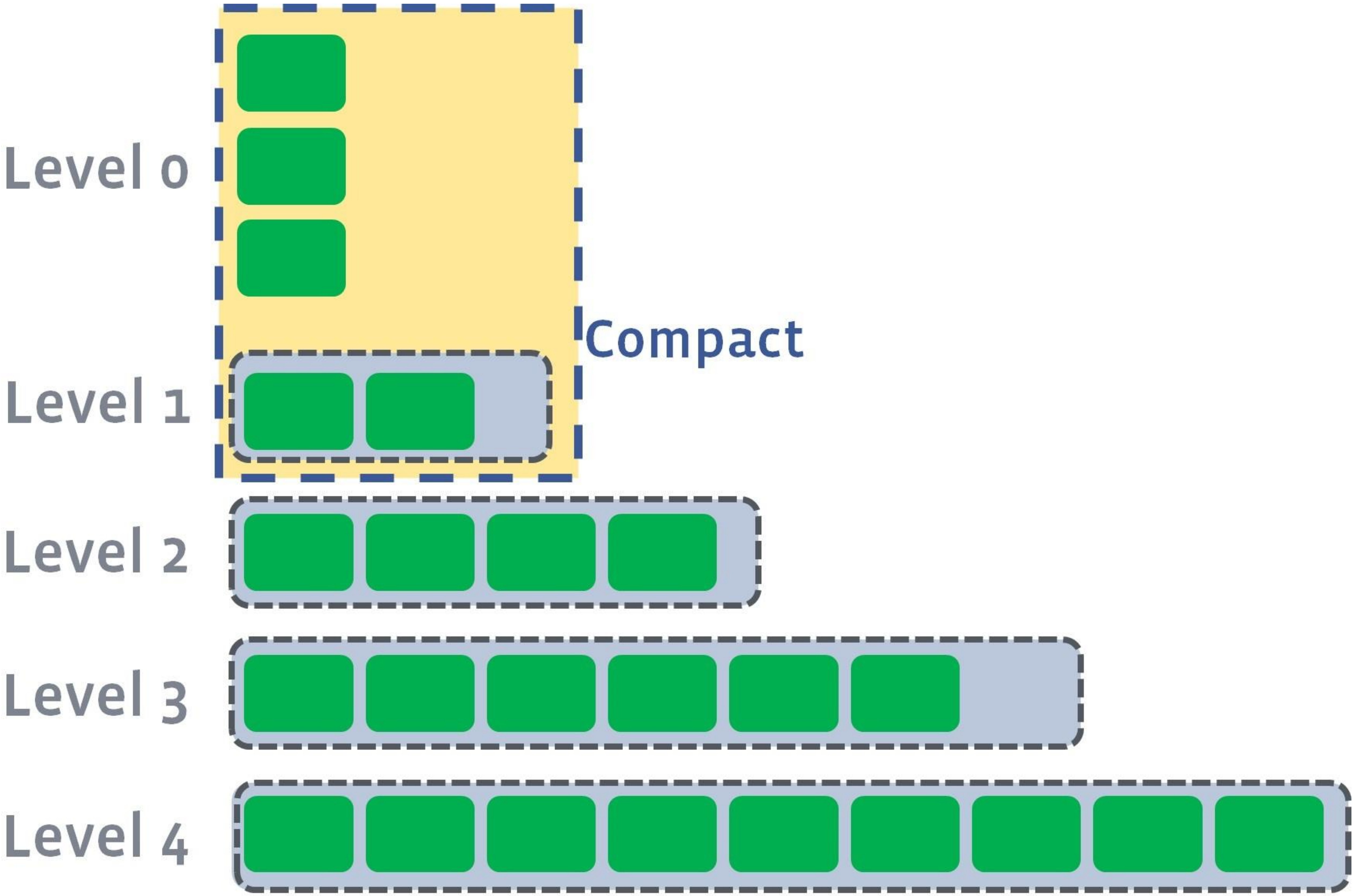
# Leveled Compaction: Structure of Files (2)



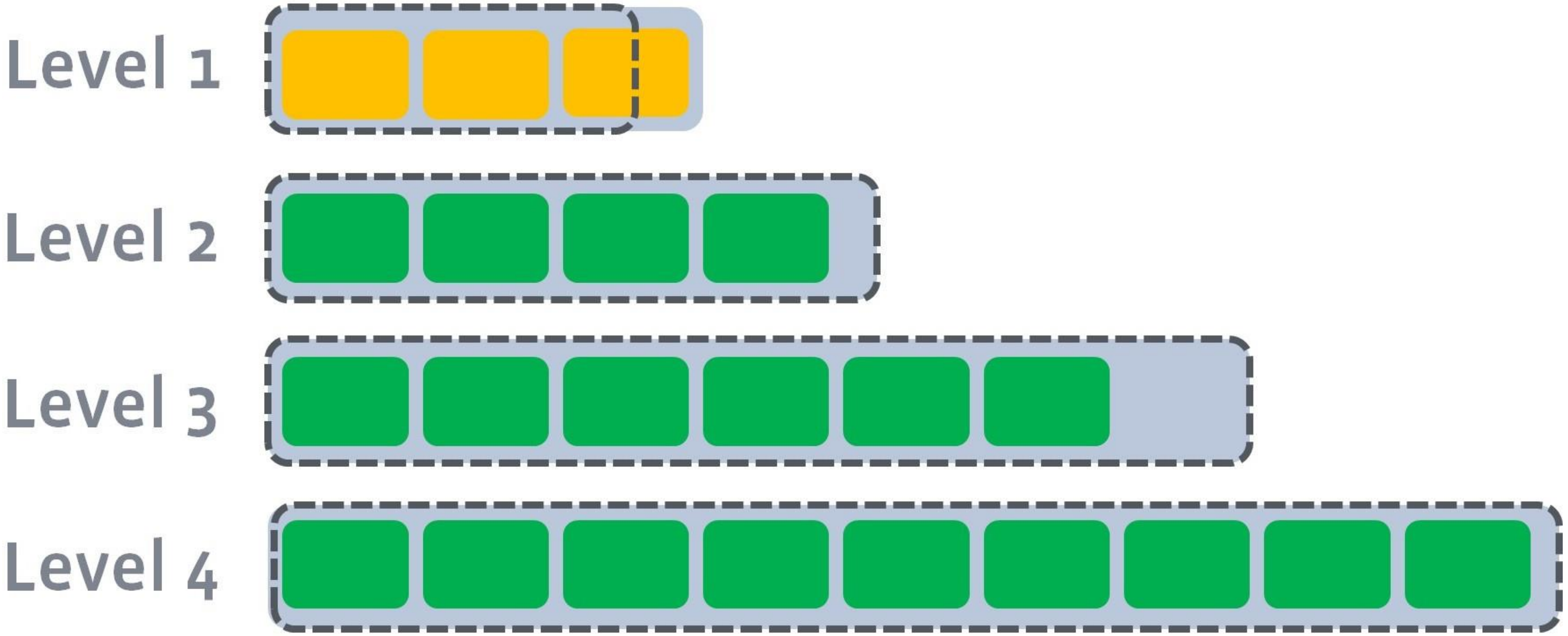
# Leveled Compaction: Structure of Files (3)



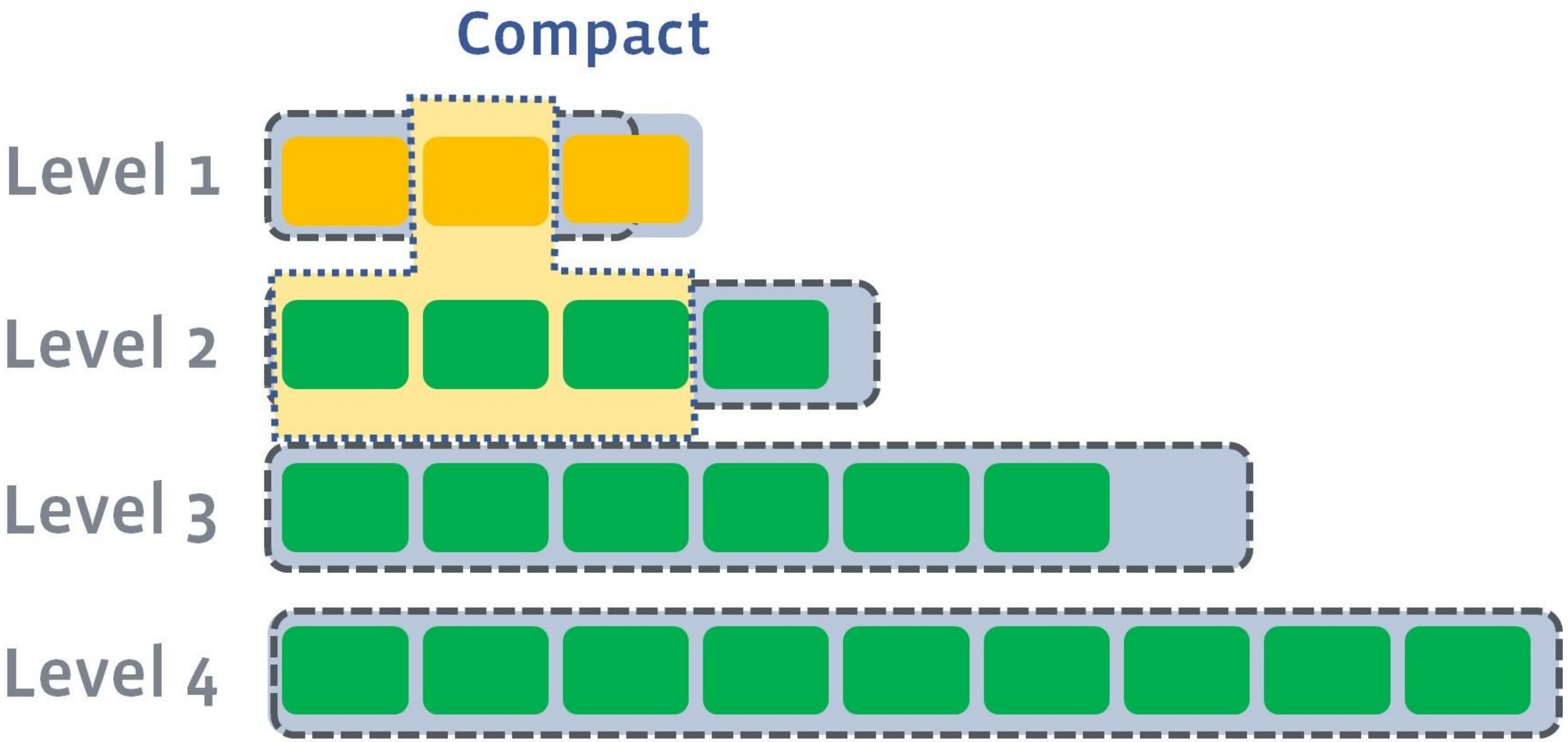
# Leveled Compaction: How It Works (1)



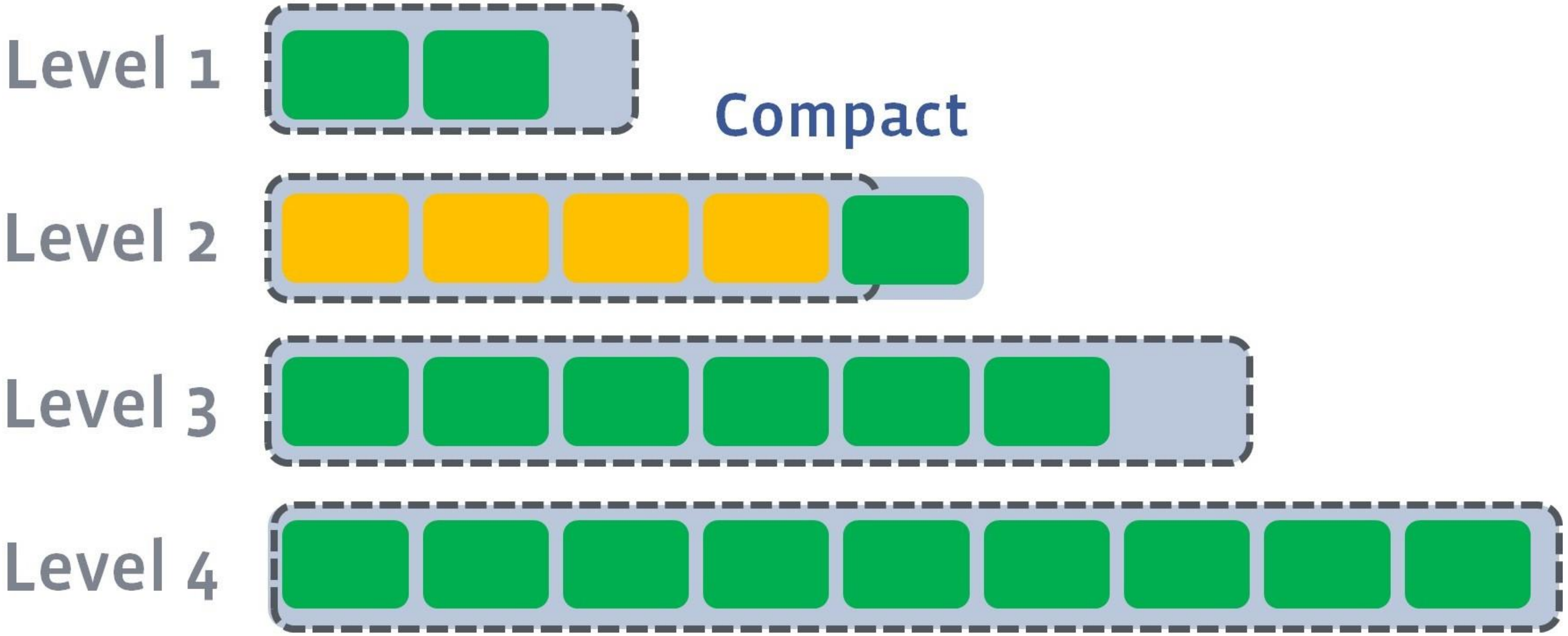
# Leveled Compaction: How It Works (2)



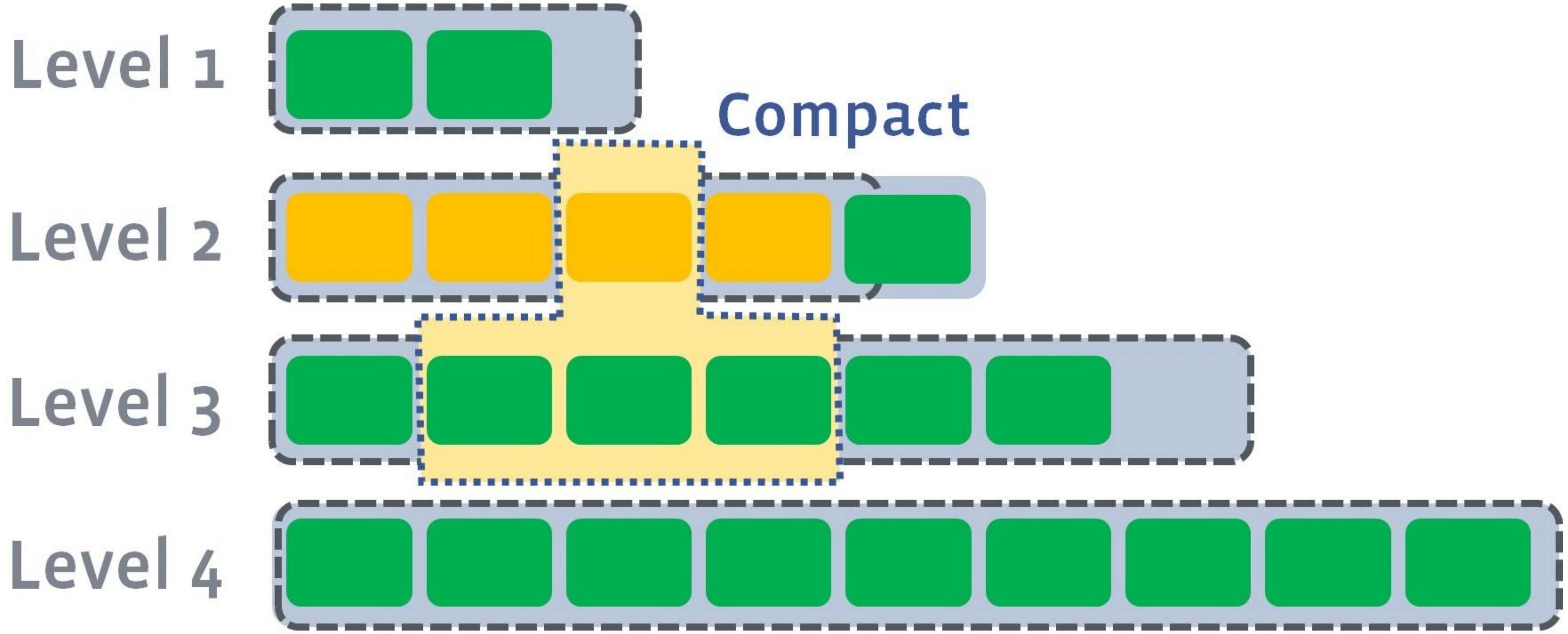
# Leveled Compaction: How It Works (3)



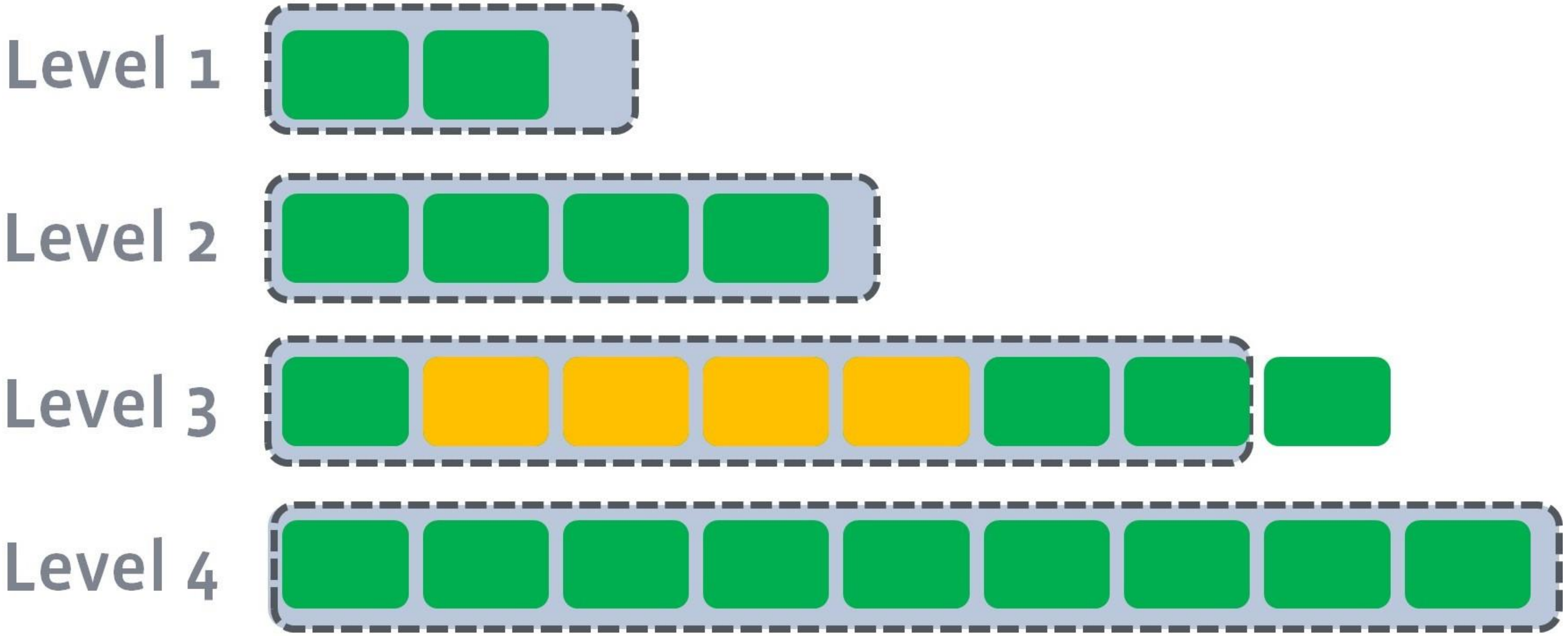
# Leveled Compaction: How It Works (4)



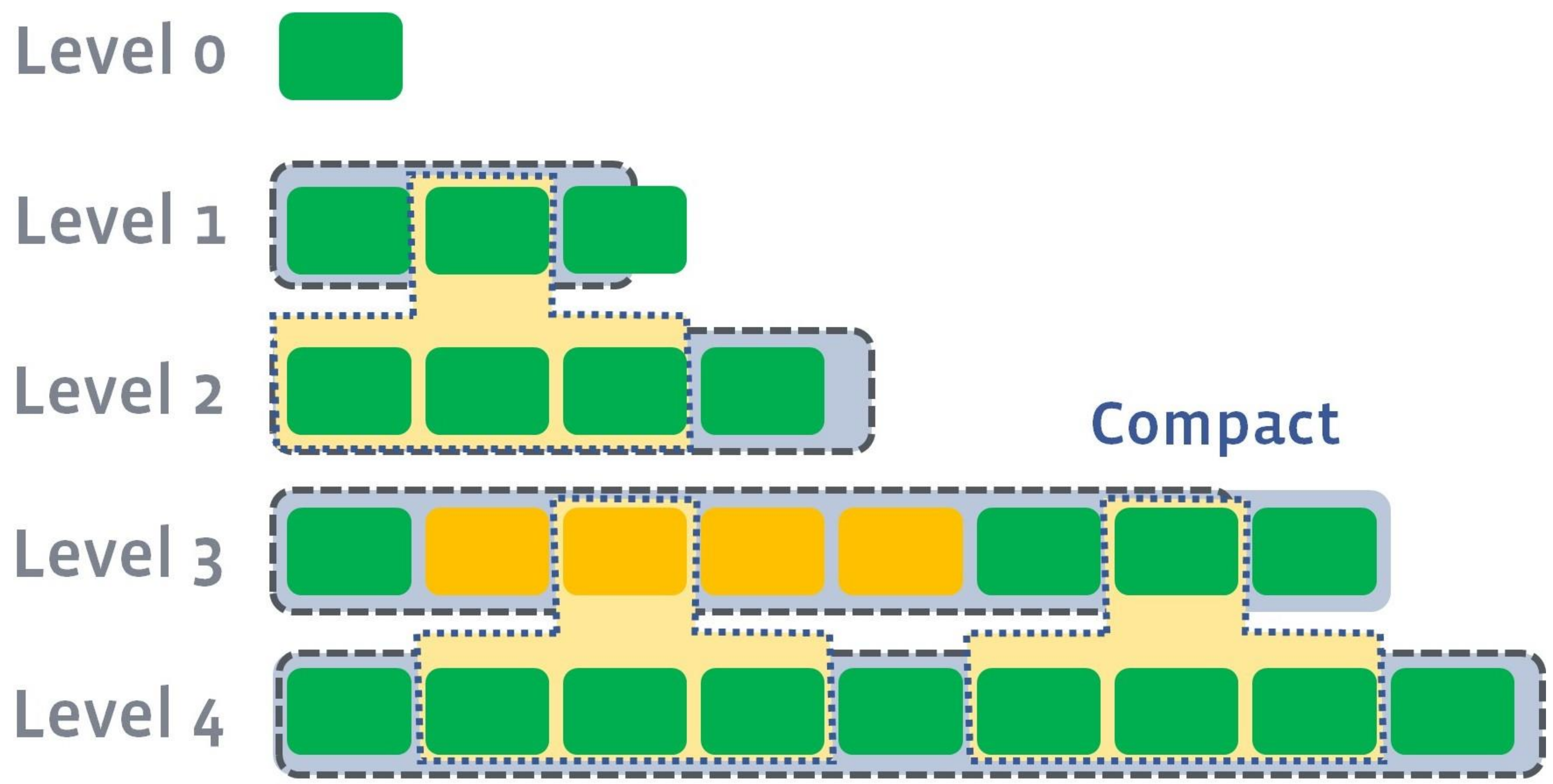
# Leveled Compaction: How It Works (5)



# Leveled Compaction: How It Works (6)



# Leveled Compaction: How It Works (7)



# Universal Compaction

```
Level 0: File0_0, File0_1, File0_2
Level 1: (empty)
Level 2: (empty)
Level 3: (empty)
Level 4: File4_0, File4_1, File4_2, File4_3
Level 5: File5_0, File5_1, File5_2, File5_3, File5_4, File5_5, File5_6, File5_7
```

- Sorted runs are laid out by updated time of the data in it and stored as either files in L0 or a whole "level"
  - Sorted run sorts data in chronological order
  - Each sorted run does not overlap each other in time

- Levels with a larger number contain older sorted run than levels of smaller numbers
  - In this example, there are five sorted runs: three files in level 0, level 4 and 5
  - Level 5 is the oldest sorted run, level 4 is newer, and the level 0 files are the newest

# Universal Compaction

```
Level 0: File0_0, File0_1, File0_2
Level 1: (empty)
Level 2: (empty)
Level 3: (empty)
Level 4: File4_0, File4_1, File4_2, File4_3
Level 5: File5_0, File5_1, File5_2, File5_3, File5_4, File5_5, File5_6, File5_7
```

**Before Compaction**

```
Level 0: File0_0
Level 1: (empty)
Level 2: (empty)
Level 3: (empty)
Level 4: File4_0', File4_1', File4_2', File4_3'
Level 5: File5_0, File5_1, File5_2, File5_3, File5_4, File5_5, File5_6, File5_7
```

**After Compaction**

- Compaction target: File0_1, File0_2 and Level 4
- Goal: place compaction outputs to the highest possible level, following the rule of older data on levels with larger numbers

# Universal Compaction

```
Level 0: File0_0, File0_1, File0_2
Level 1: (empty)
Level 2: (empty)
Level 3: (empty)
Level 4: File4_0, File4_1, File4_2, File4_3
Level 5: File5_0, File5_1, File5_2, File5_3, File5_4, File5_5, File5_6, File5_7
```

**Before Compaction**

```
Level 0: (empty)
Level 1: (empty)
Level 2: (empty)
Level 3: File3_0, File3_1, File3_2
Level 4: File4_0, File4_1, File4_2, File4_3
Level 5: File5_0, File5_1, File5_2, File5_3, File5_4, File5_5, File5_6, File5_7
```

**After Compaction**

- Compaction target: File0_1, File0_2 and File0_0
- Goal: place compaction outputs to the highest possible level, following the rule of older data on levels with larger numbers

# No Experiment for This Week

- No report for this week
- Study for the midterm exam :) Good Luck!

# References

1. Facebook, "RocksDB", http://rocksdb.org
2. 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.
3. Chen Luo, Michael J. Carey, "LSM-based Storage Techniques: A Survey". VLDB Journal, 2019
4. 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
5. Mijin An, "RocksDB detail", SlideShare, https://www.slideshare.net/meeeejin/rocksdb-detail
Facebook, "RocksDB Wiki", GitHub, https://github.com/facebook/rocksdb/wiki
6. Mark Callaghan, "Tiered or leveled compaction, why not both via adaptive compaction?", Small Datum, http://smalldatum.blogspot.com/2018/07/tiered-or-leveled-compaction-why-not.html
7. Mijin An, "RocksDB Compaction", SlideShare, https://www.slideshare.net/meeeejin/rocksdb-compaction
8. MySQL, "MySQL Community Downloads", https://dev.mysql.com/downloads/mysql/
9. Dong, Siying, Mark D. Callaghan, Leonidas Galanis, Dhruba Borthakur, T. Savor and Michael Strum. "Optimizing Space Amplification in RocksDB." CIDR (2017).
10. https://github.com/facebook/rocksdb/wiki/Universal-Compaction