Optimizing Space Amplification in RocksDB, CIDR, 2017

연세대학교 컴퓨터과학과 김휘군

과제명: IoT 환경을 위한 고성능 플래시 메모리 스토리지 기반 인메모리 분산 DBMS 연구개발

과제번호: 2017-0-00477

과학기술정보통신부
Ministry of Science and ICT

연세대학교
YONSEI UNIVERSITY

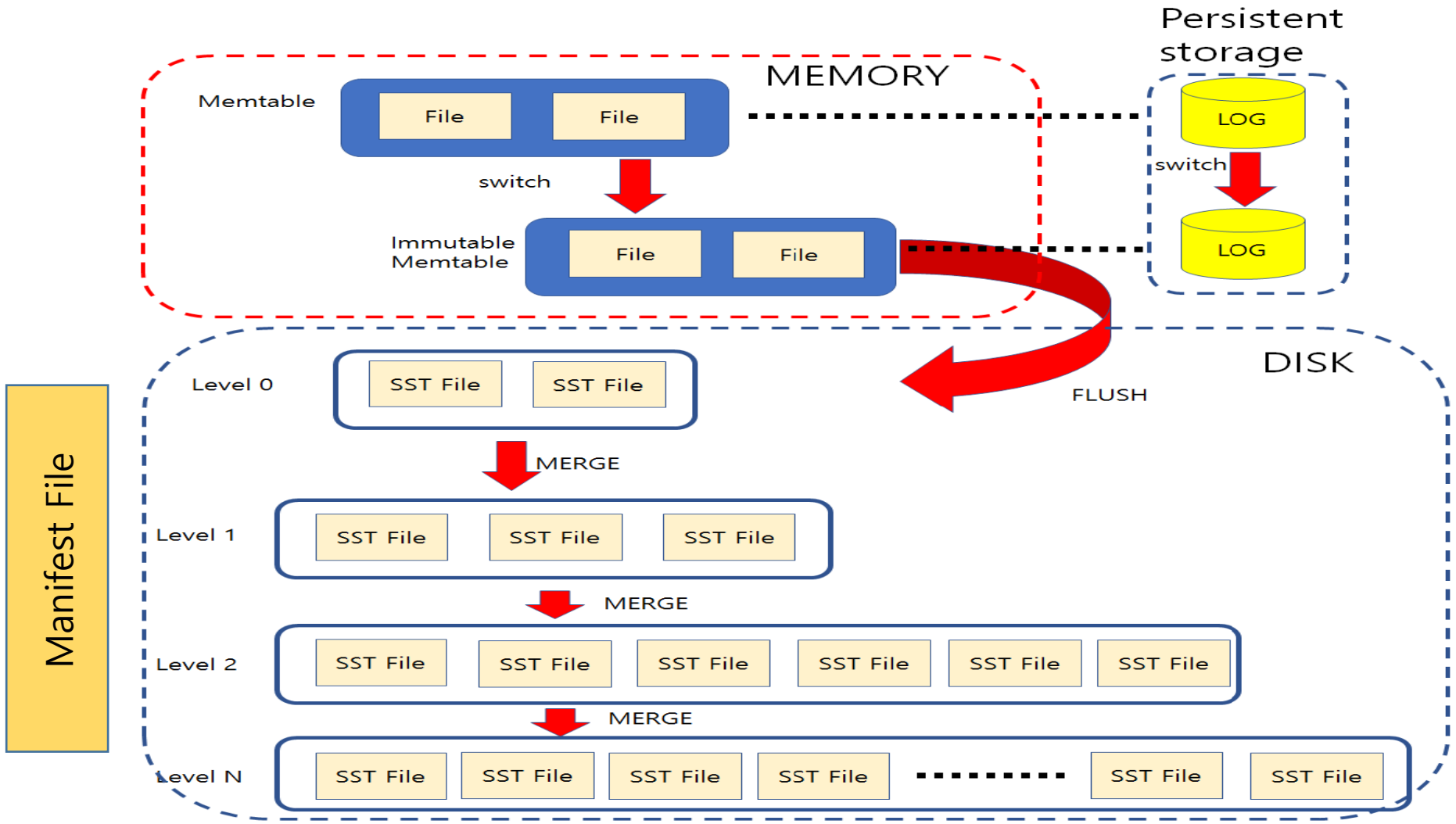
IITP 정보통신기술진흥센터
Institute for Information & communications Technology Promotion

목차

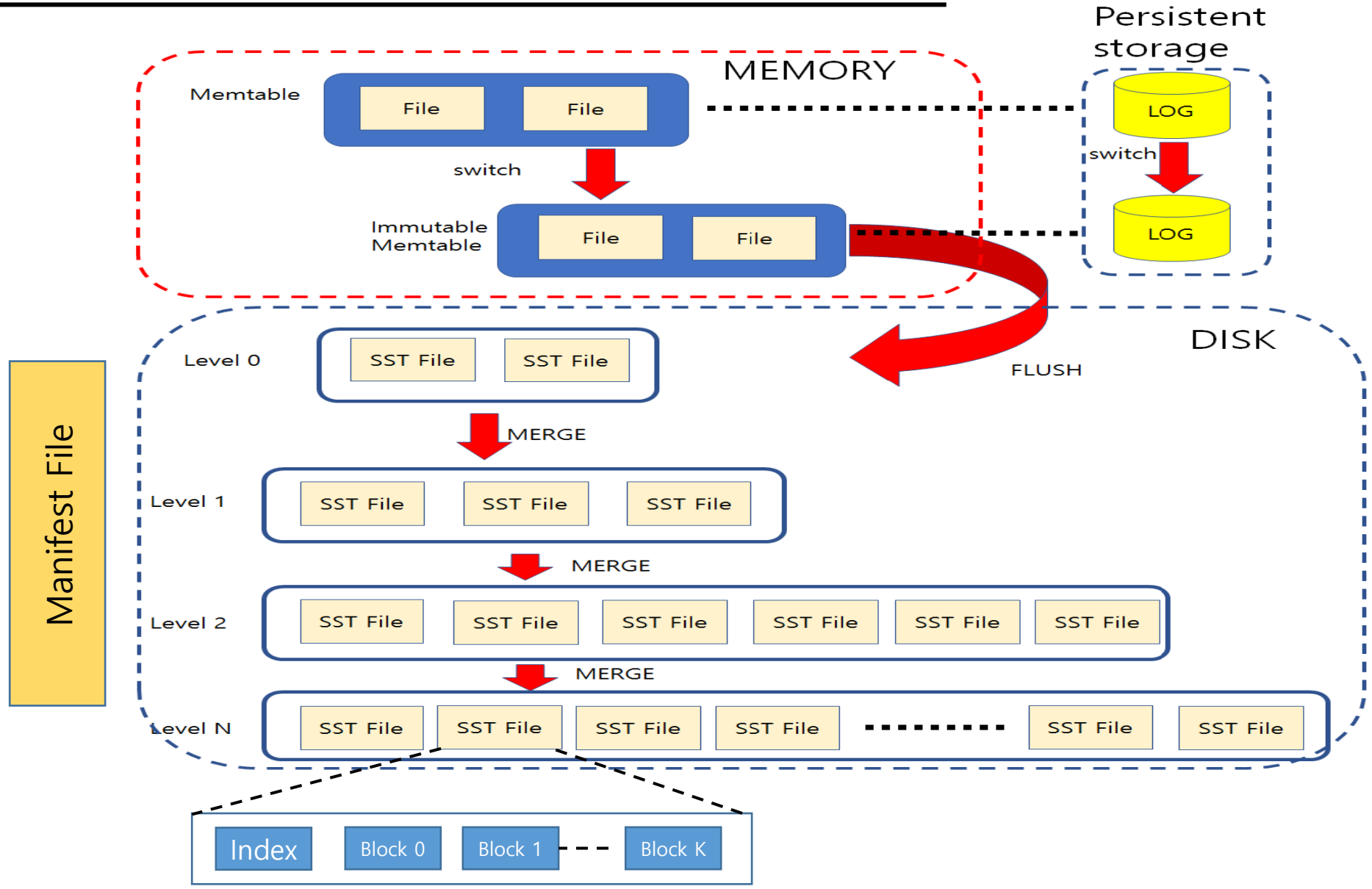
- Introduction
- Space Amplification
- Level Size Adaptation
- Tradeoffs
- Data Compression
- Tiered Compression
- Evaluation

Introduction

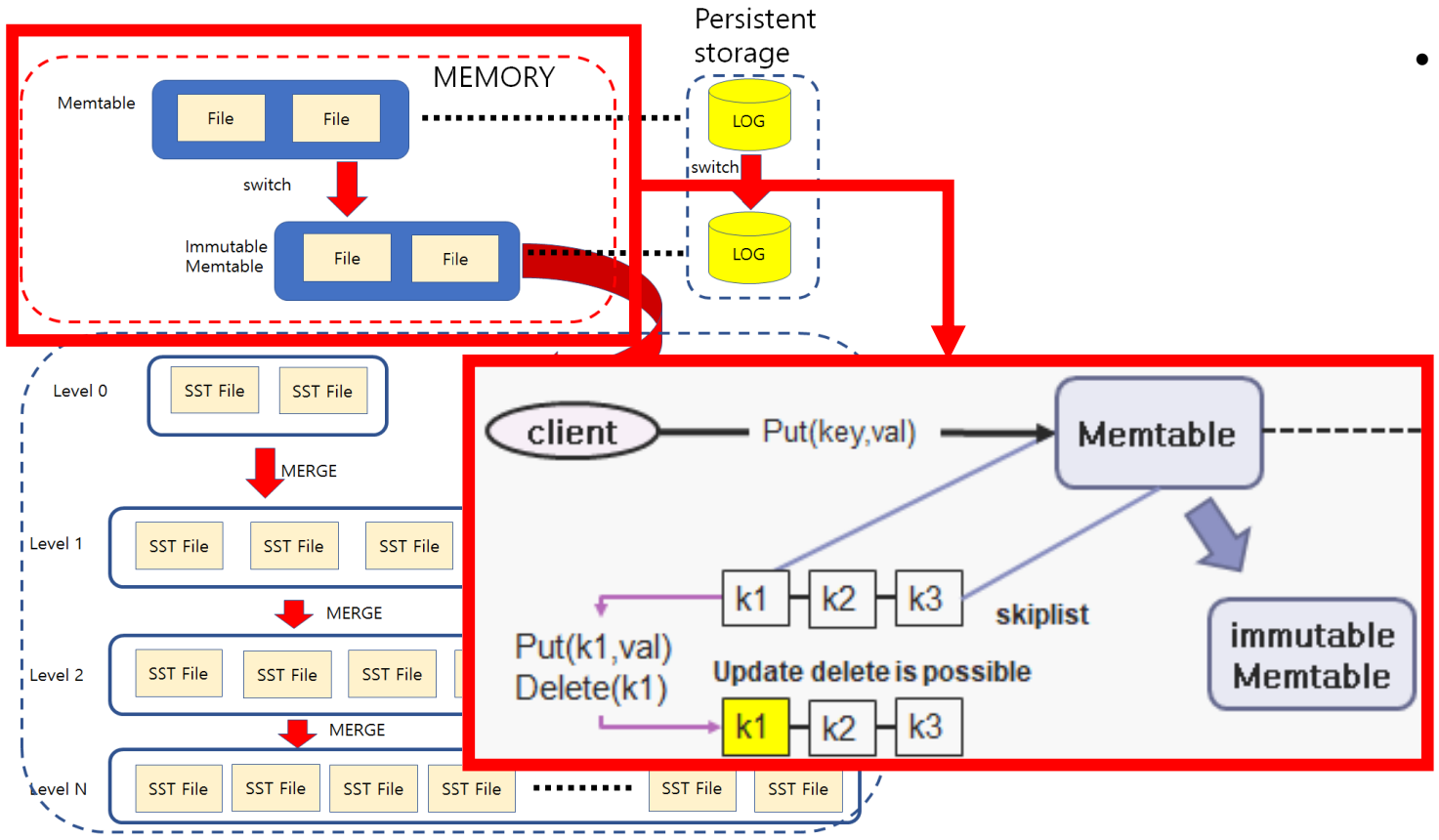
RocksDB Architecture



RocksDB Architecture

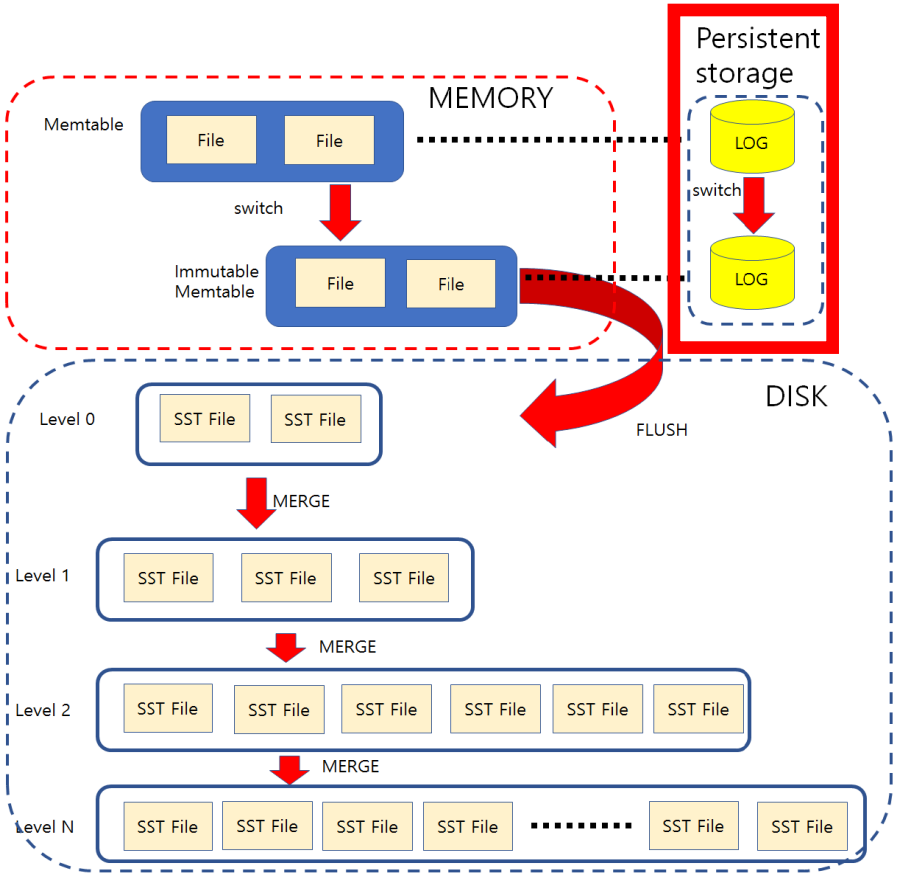


RocksDB Architecture



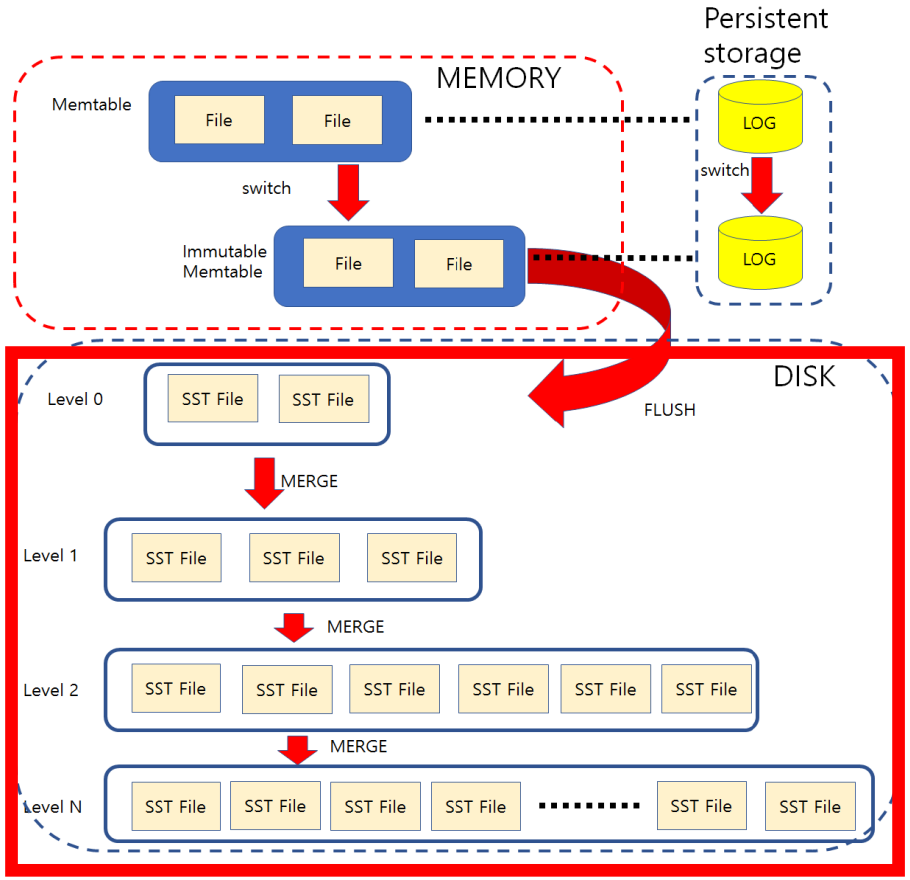
- Memtable
  - In-Memory 데이터 구조
  - 데이터가 기록될 때 데이터가 최초로 저장되는 공간
  - Memtable에 데이터가 일정 용량까지 기록되면 변경이 불가능한 Immutable Memtable로 변경
  - 변경된 Immutable Memtable은 SSTable로 변경되어 Disk로 Flush됨

RocksDB Architecture



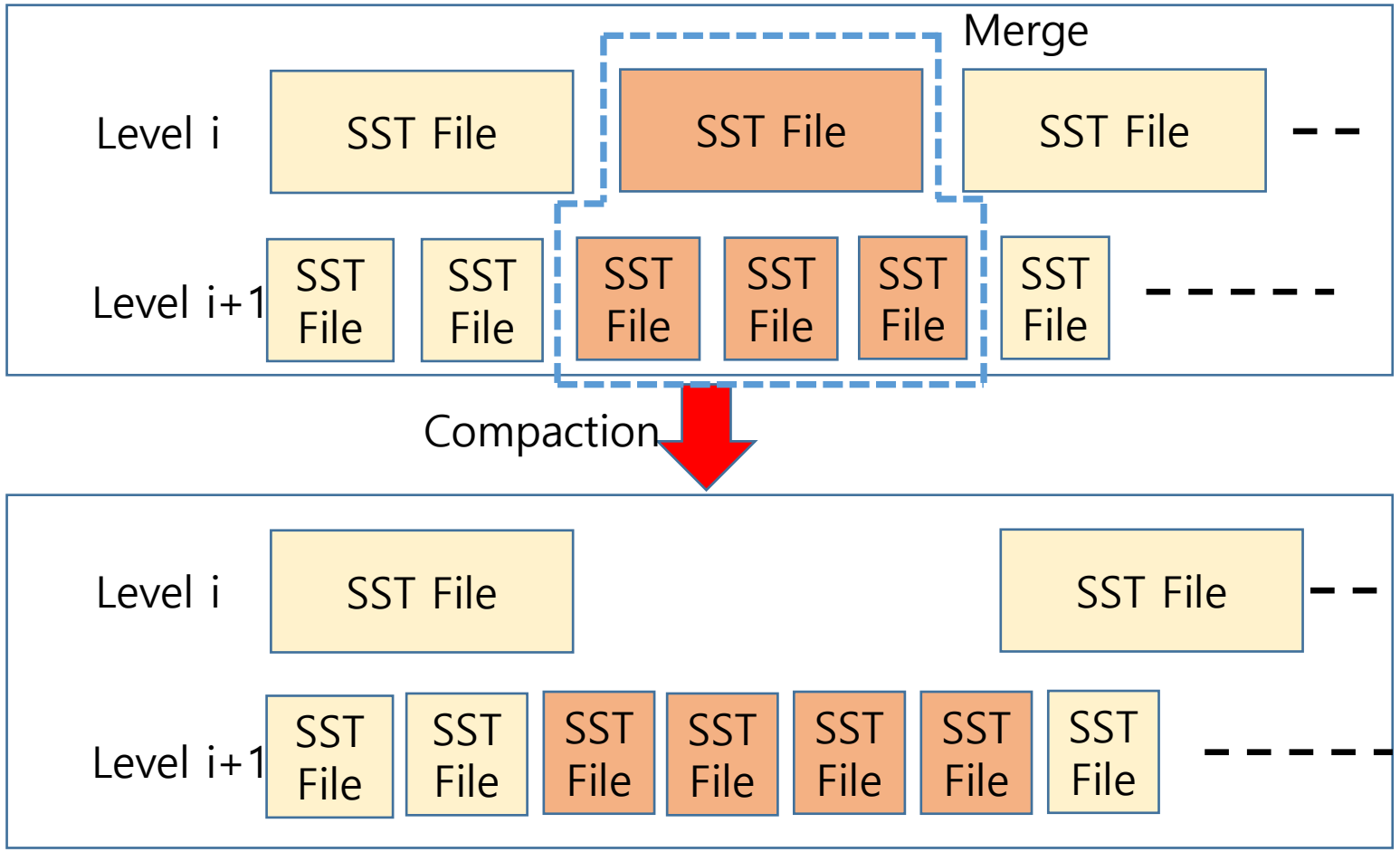
- Log
  - 데이터 유실을 방지하기 위하여 쓰기 및 갱신 연산에 대하여 Log를 기록
  - Memtable에 데이터를 기록하기 전에 Log를 기록하는 방식인 Write-Ahead-Logging 사용

RocksDB Architecture



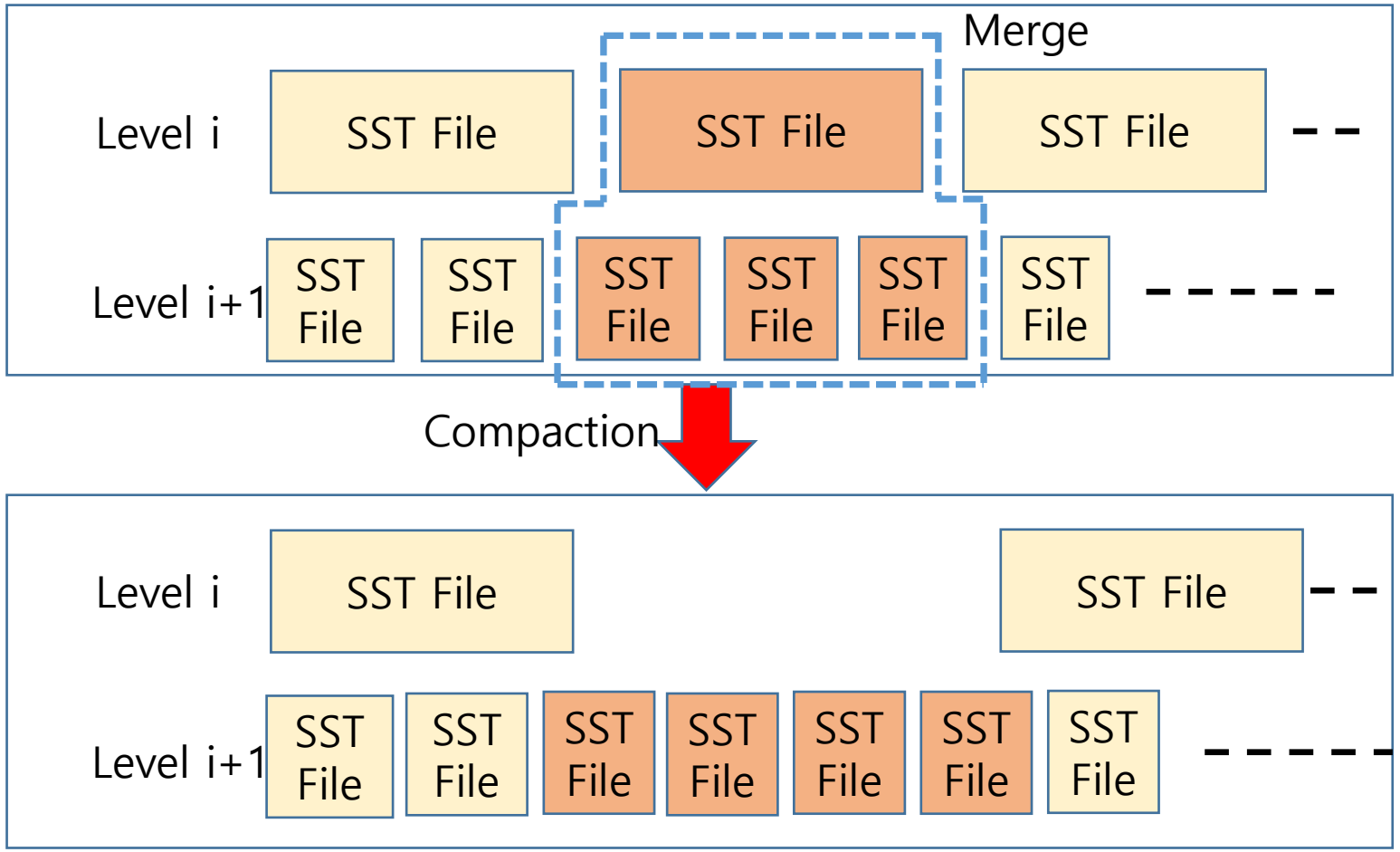
- SSTable
  - Immutable Metable의 데이터를 Key순으로 정렬하여 저장한 파일
  - 저장된 SSTable은 파일의 내용이 변하지 않음
  - Disk 영역의 레벨에 상주함
  - 상위 레벨에 상주하는 파일일수록 최신의 데이터가 위치함

Compaction



- Append 방식으로 데이터를 저장하는 LSM-Tree구조에서 데이터를 관리하는 방법
- 세가지 Compaction 방법이 존재
  - Leveled Compaction
  - Universal Compaction
  - FIFO Compaction

Compaction



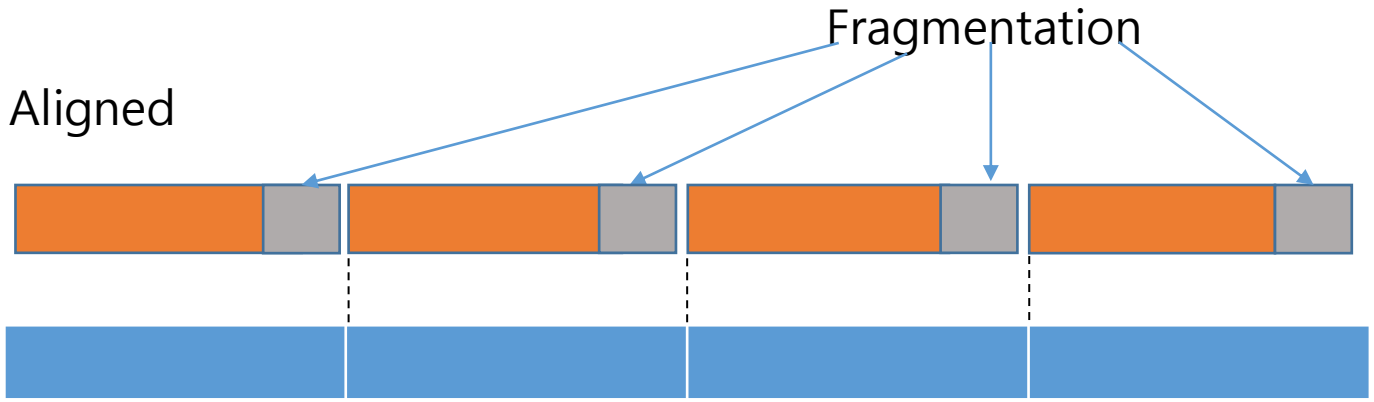
- Leveled Compaction 과정
  1. 상위 레벨의 크기가 임계점을 넘게 되면 하위레벨로 내려줄 SST파일을 선택
  2. 하위 레벨의 SST파일 중 선택된 상위 레벨의 SST파일의 Key 범위가 겹치는 파일들을 선택하여 병합 정렬 진행
  3. 병합 정렬을 통해 생성되는 SST파일들을 하위 레벨에 파일들을 기록
  4. 선택 되었던 상위 레벨의 SST파일은 Garbage Collector에 의해 삭제됨

Space amplification

- InnoDB

- Page size
- Actual saved data size

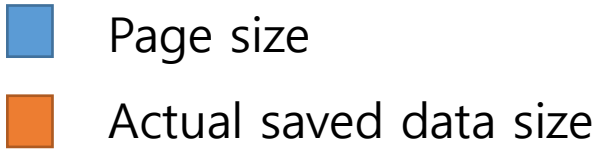
- Aligned



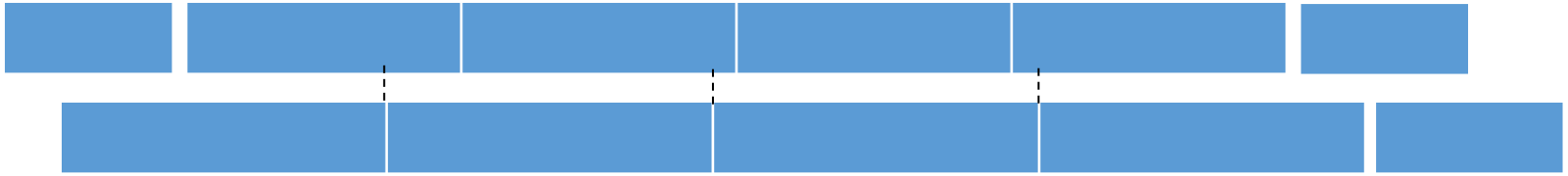
➡ Space amplification is worse than 1.5

Space amplification

- RocksDB



- Misaligned



→ Space amplification is worse than 1.11…

Space amplification

- RocksDB ‘s strategy to reduce space amplification
  1. Adapting level sizes

  2. Compression strategy
     - Key prefix encoding
     - Sequence ID GC
     - Data compression

Dynamic level size adaptation

The level size can be adjusted by parameter -> level size multiplier

- Worst case

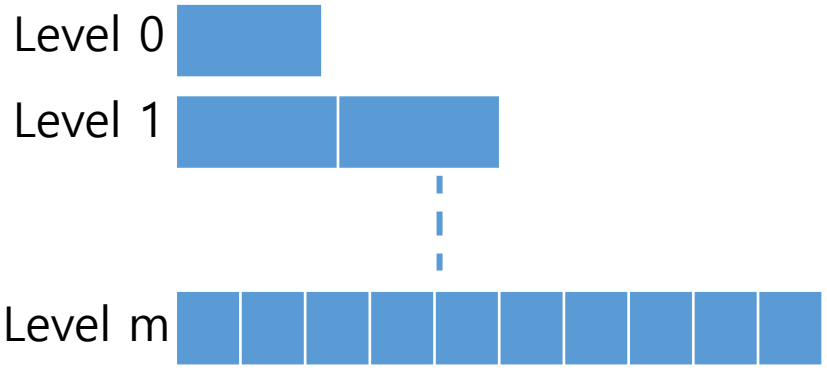
Dynamic level size adaptation

The level size can be adjusted by parameter -> level size multiplier

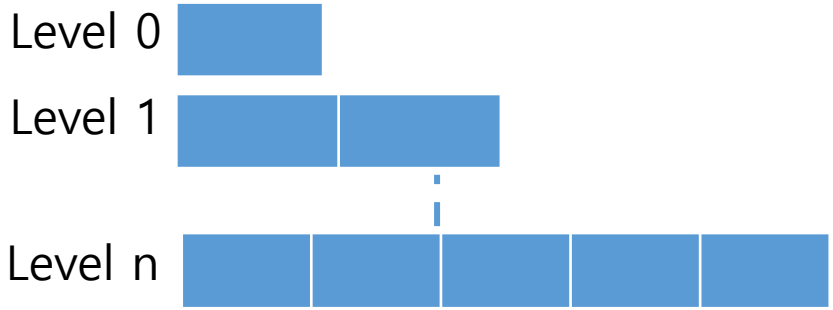
level size adaptation

* m > n

Smaller size multiplier



Larger size multiplier



Tradeoffs

- Level Size multiplier

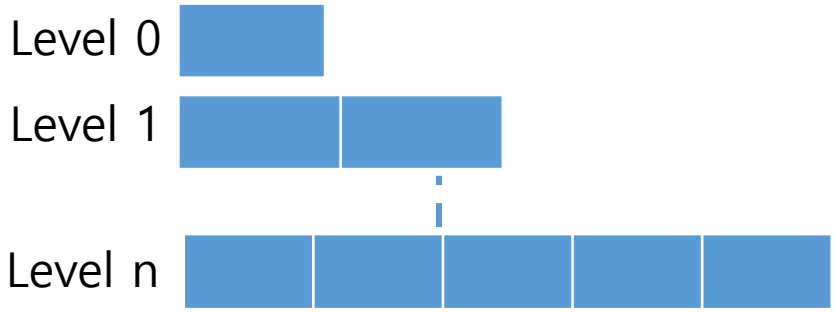
Smaller size multiplier

Write amplification ↓

Space, Read amplification ↑

Larger size multiplier

* m > n



Space, Read amplification ↓

Write amplification ↑

Tradeoffs

- Block size

Small block size

Compression ↓

Read amplification ↓

Large block size

Compression ↑

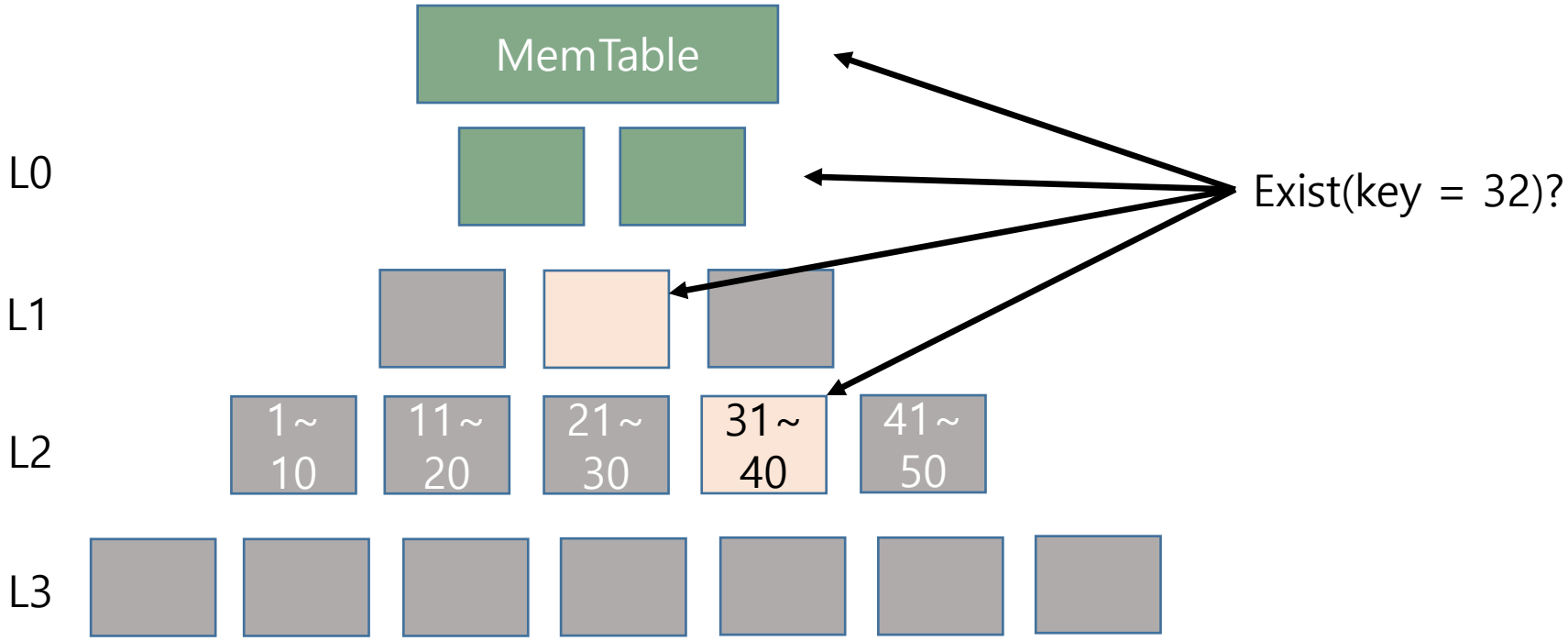
Read amplification ↑

Key prefix encoding

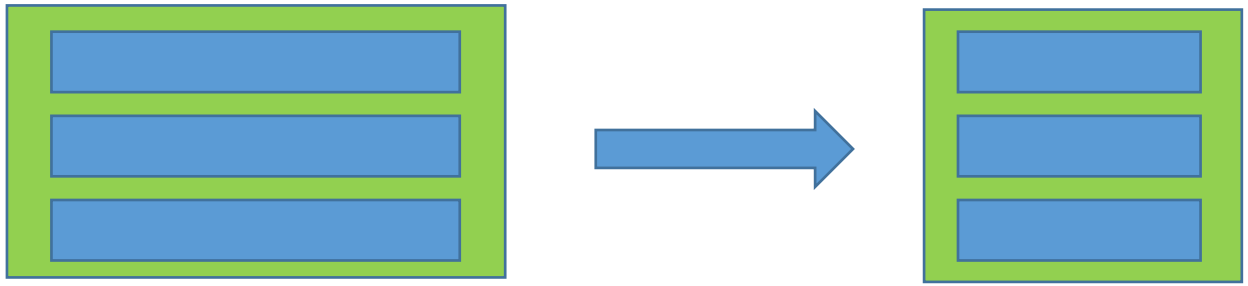
Key_len	Key	value
8	K0000001	1
8	K0000002	2
8	K0000003	3
8	K0000004	4

Key_len	Prefix_len	Key	value
8	0	K0000001	1
1	7	2	2
1	7	3	3
1	7	4	4

Bloom filter



Data Compression

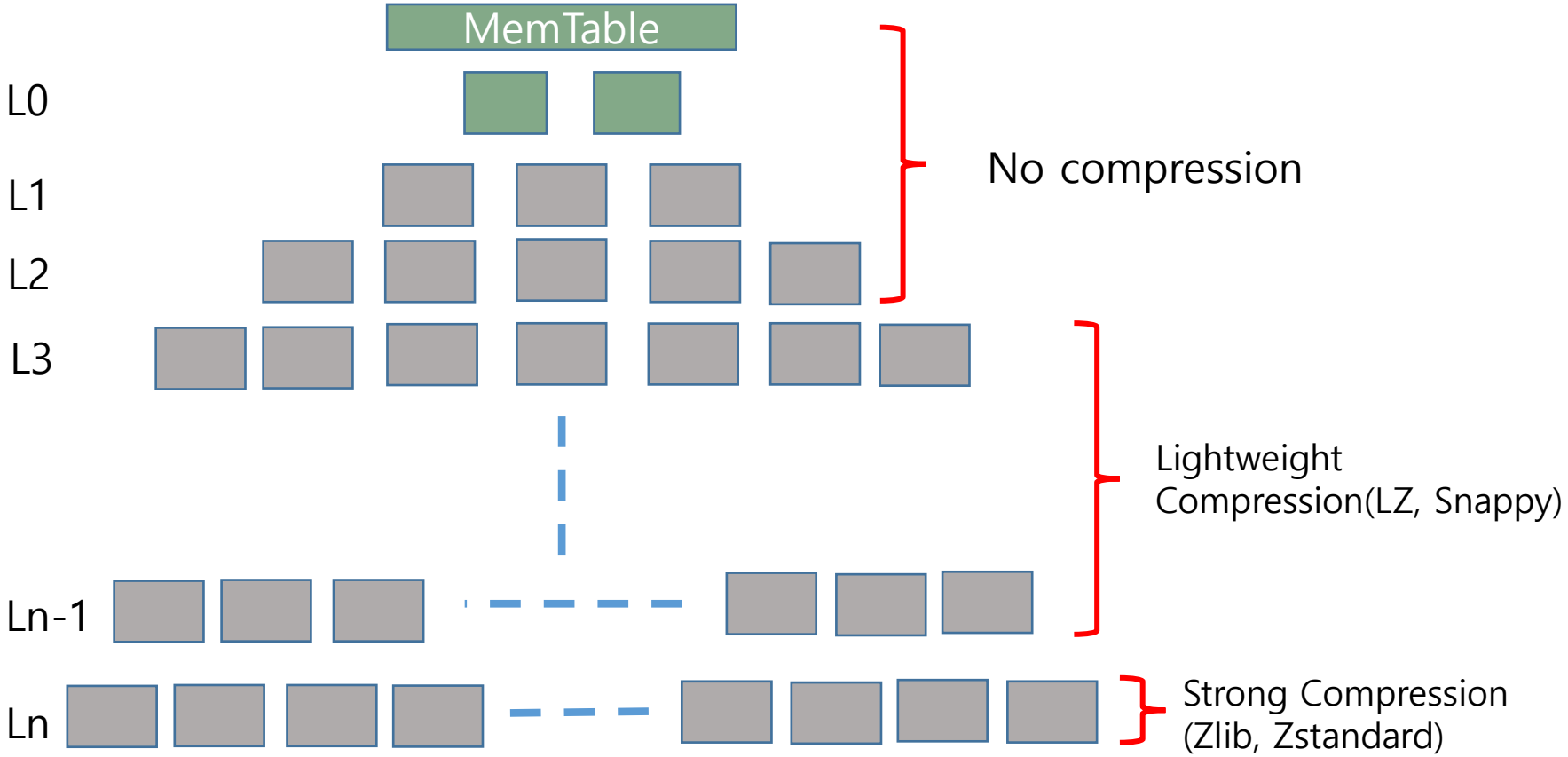


- LZ, Snappy, zlib, Zstandard

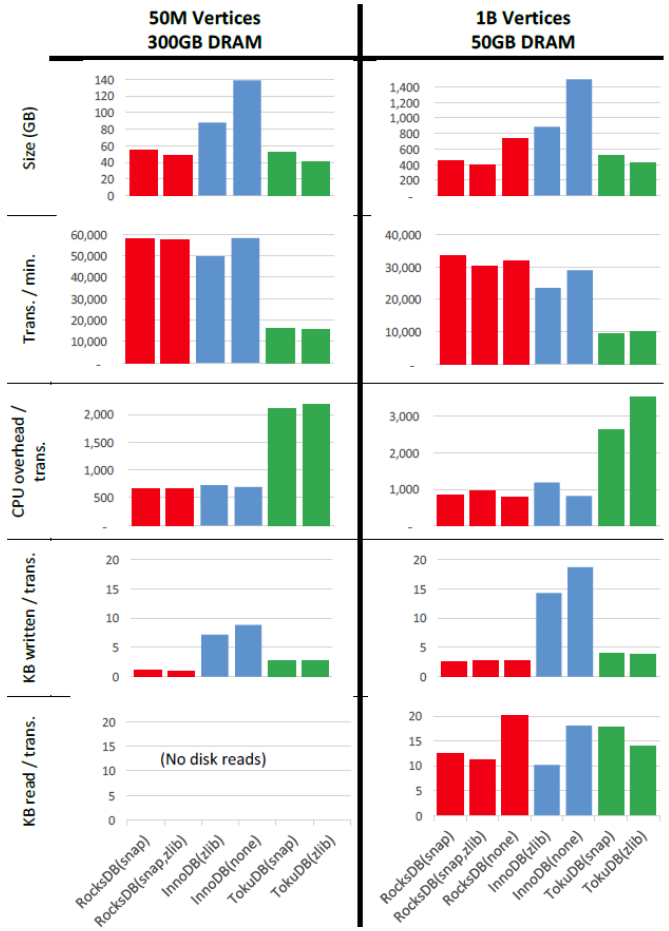
stronger

- Stronger Algorithm – 25%
- Weaker Algorithm – 40%

Tiered Compression



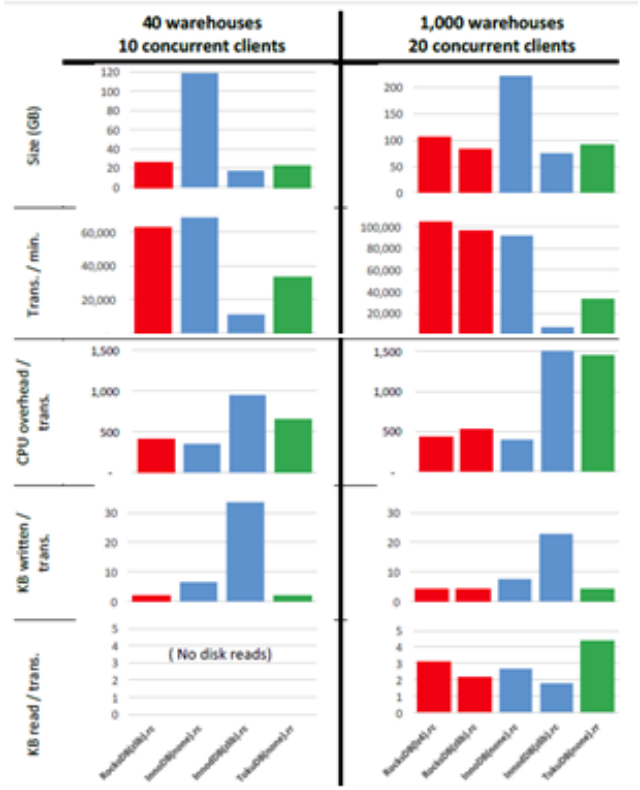
Evaluation - Linkbench



- RocksDB
- InnoDB
- TokuDB

- Space usage
  - RocksDB < InnoDB
- Throughput
  - RocksDB >= InnoDB
- CPU Overhead
  - RocksDB <= InnoDB
- Write volume
  - RocksDB <= InnoDB

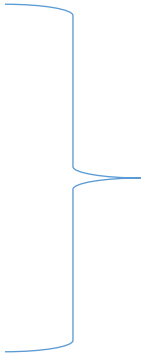
Evaluation – TPC-H



- RocksDB
- InnoDB
- TokuDB

Conclusion

- Technique
  - Level Size Adaptation
  - Bloom filter
  - Compression
    - Data compression
    - Tiered compression



Less storage space
High throughput