NTNU
Kunnskap for en bedre verden

SSDs, LSM-trees and RocksDB

Håvard Dybvik

Svein Erik Bratsberg, IDI/NTNU

Contents

HDDs

- Rotating, magnetic disks
- Have been developed since 1956
- Storing vast amounts of data at low costs
- Access time does not improve much with new disks
- 5-10 millisecs today, the first (1956) had 600 millisecs.
- Throughput (typical every day, desktop disk):

  160 MB/s (write)

  180 MB/s (read)
- Needs to be careful with layout of file system

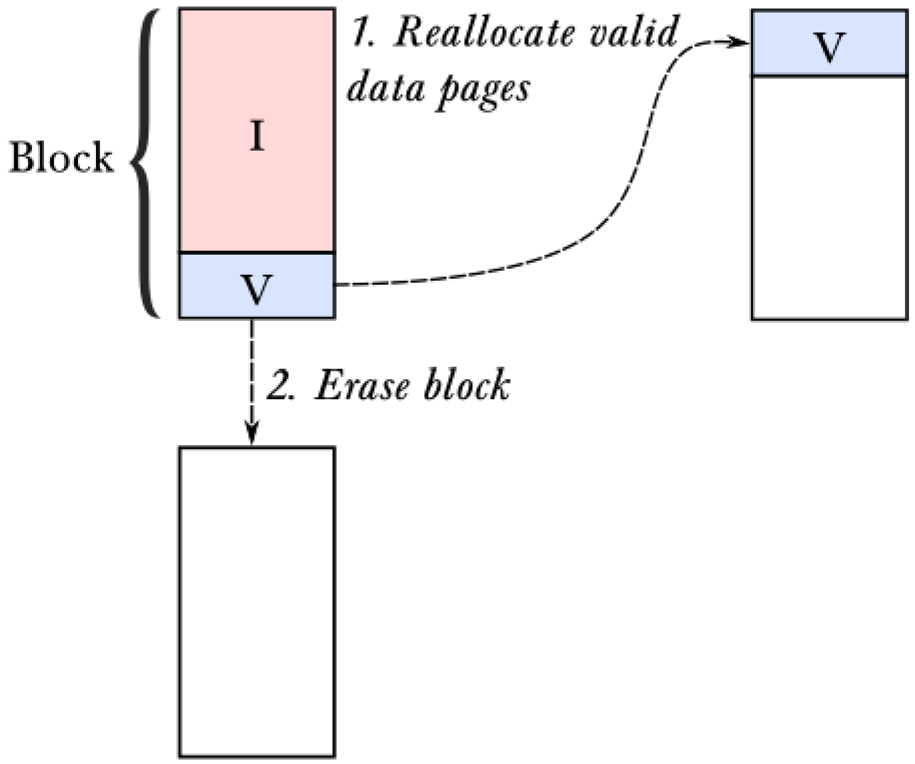
SSDs (1)

- Solid state drives – purely electronic devices
- Faster access times, lower latency, lower power consumption, completely silent operation, uniform random access speed
- NAND: Block: 64 or 128 *pages* of 2 Kbyte or 4 Kbyte Block: 128 KB to 512 KB
- Reads and writes of *pages*
- Erase *complete blocks*
- Erase-before-write
- Throughput: Four times of HDD (at least), 550 MB/s

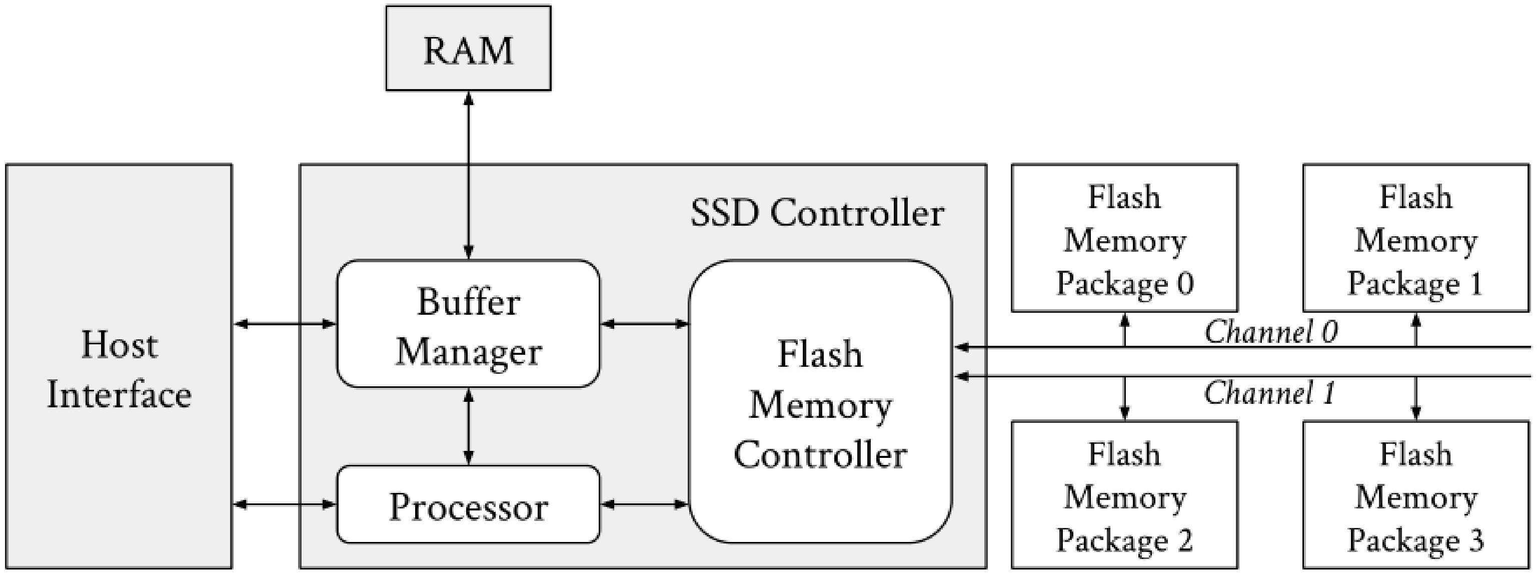
SSDs – wear leveling

- Limited number of writes possible (wearing)
- Flash Translation Layer – firmware layer implementing wear leveling: Standardized by Intel.
  - *No wear leveling*: Fixed mapping from logical addresses to physical addresses
  - *Dynamic wear leveling*: At updates, the old block is marked as invalid and the block is relocated.
  - *Static wear leveling*: Also moves static blocks periodically. All parts of the disk will be worn out eventually.
- Garbage collection is important in SSDs. Done in units of blocks.
- Host based FTL (computer) and array-based FTL (disk)

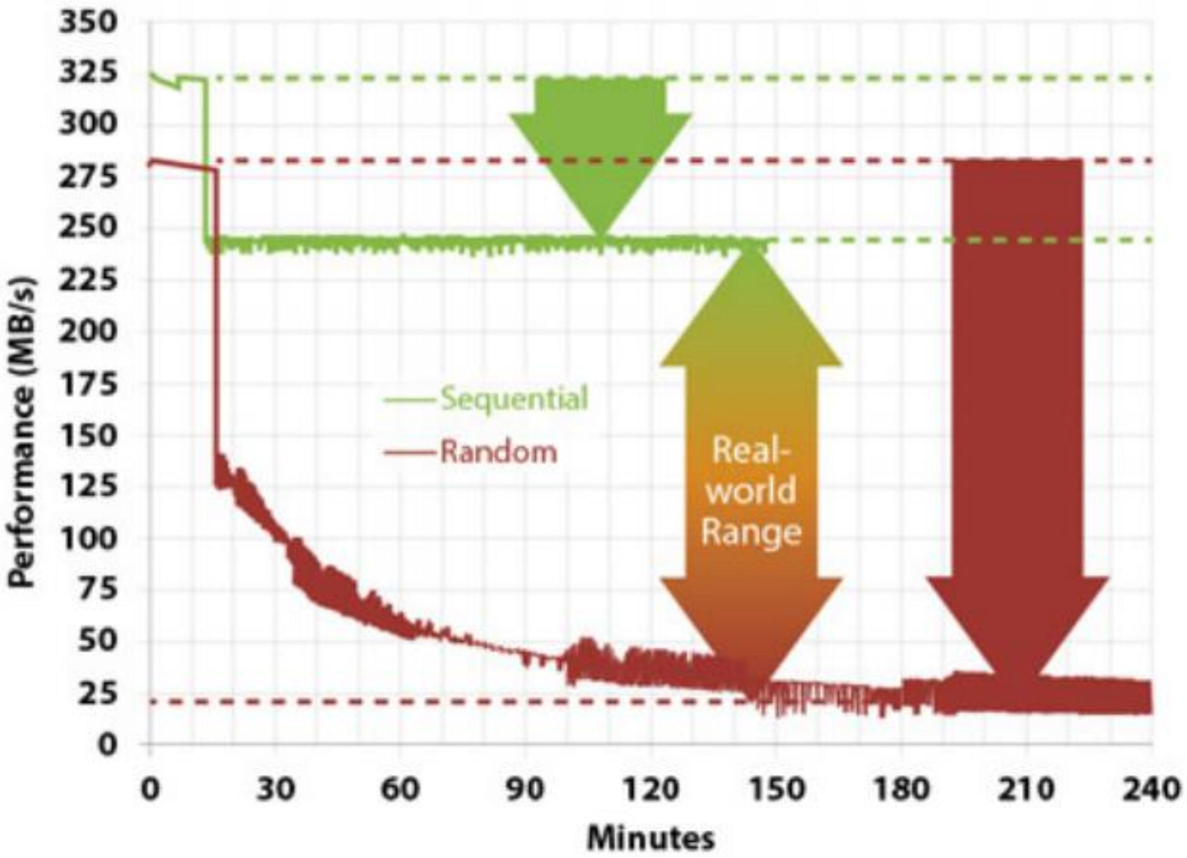
SSDs – garbage collection



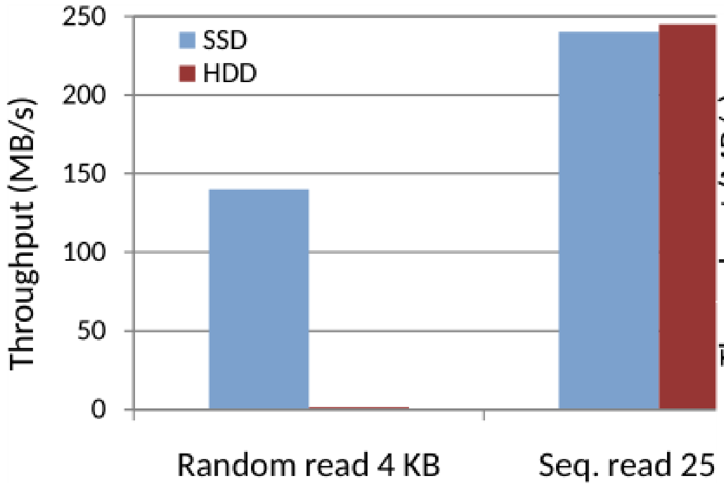
SSDs – parallel I/O



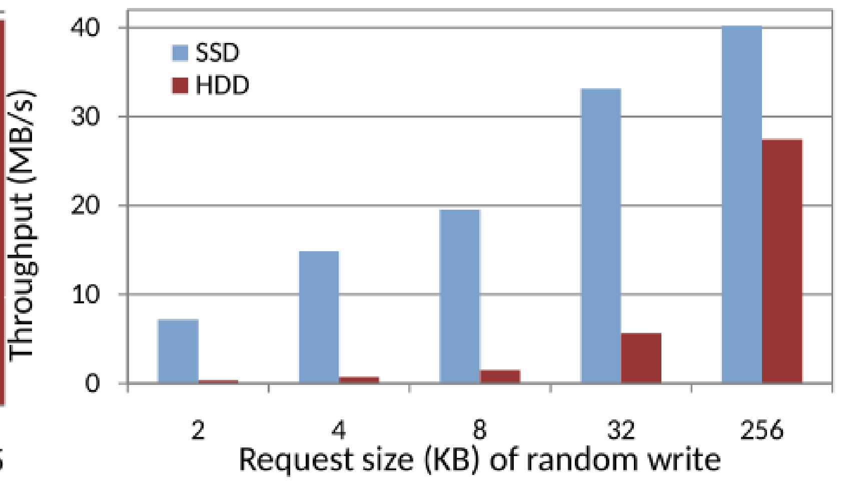
Sequential writes (1)



Random writes, use big chunks when writing



(a) Reads.



(b) Random writes.

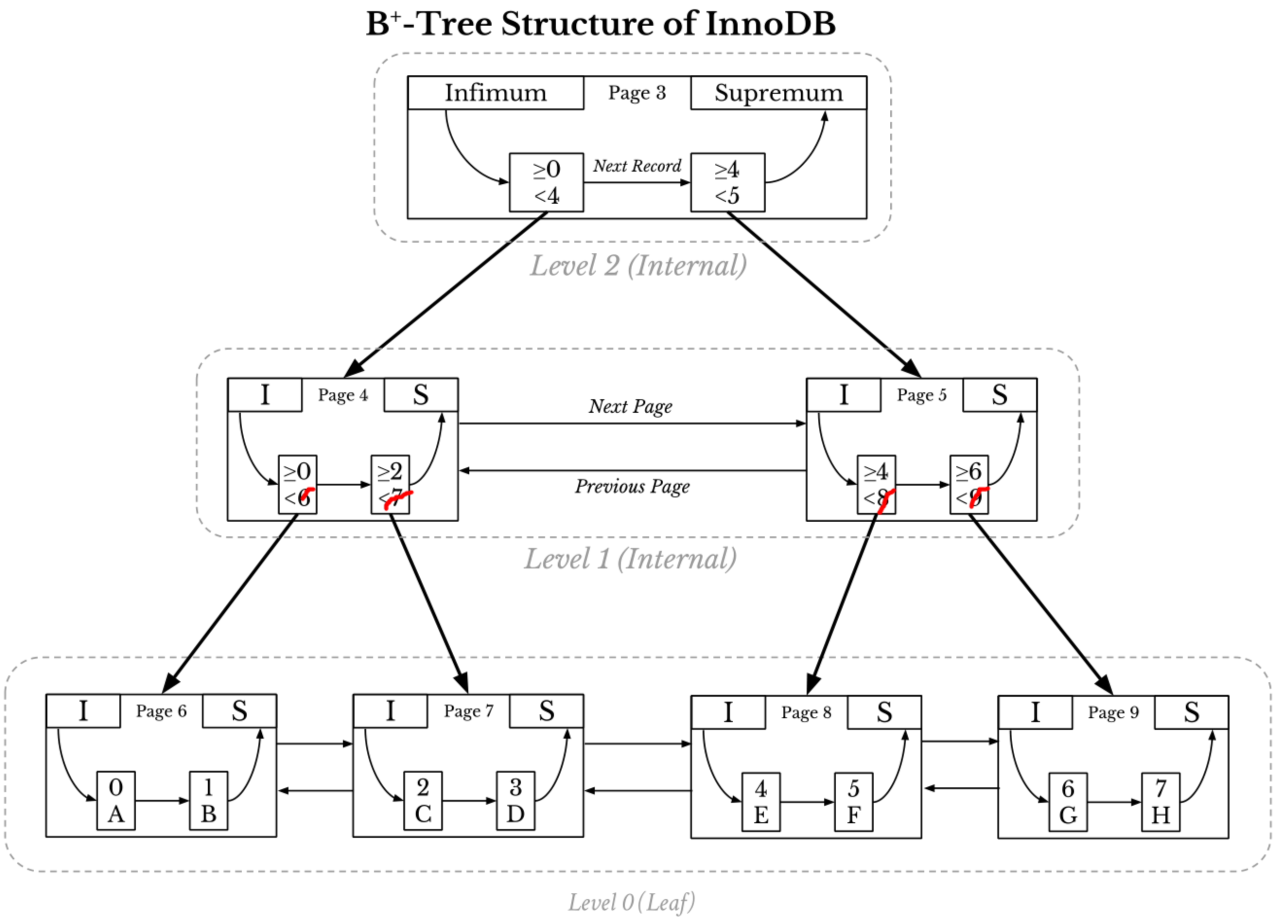
Sequential writes (3)

- One of the essential keys to great write performance is to organize data in such a way that each write request distributes the overhead of the write over multiple inserts batched together – large blocks are also good
- *Over-provisioning*: When the GC has too much to do, writes happen more often than erase operations -- The SSD has a reserved area where the garbage collector could put writes at high peaks periods. 7 – 28 % of disk.

Writes/reads measured (old exercise)

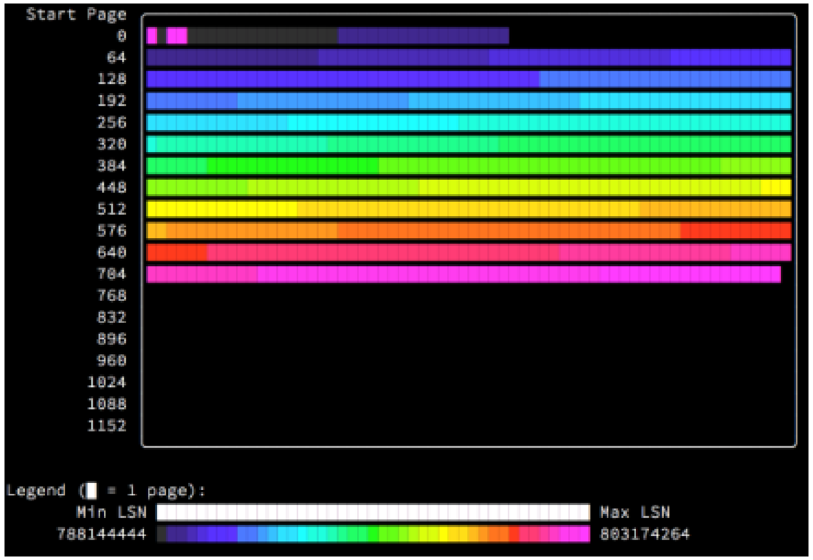
	HDD/w	HDD/r	SSD/w	SSD/r
1GB	1976 MB/s	3835 MB/s		
2GB	2101 MB/s	3954 MB/s		
4GB	1766 MB/s	202 MB/s		
8GB	386 MB/s	157 MB/s		
16GB	221 MB/s	162 MB/s		
32GB	184 MB/s	163 MB/s	354 MB/s	463 MB/s

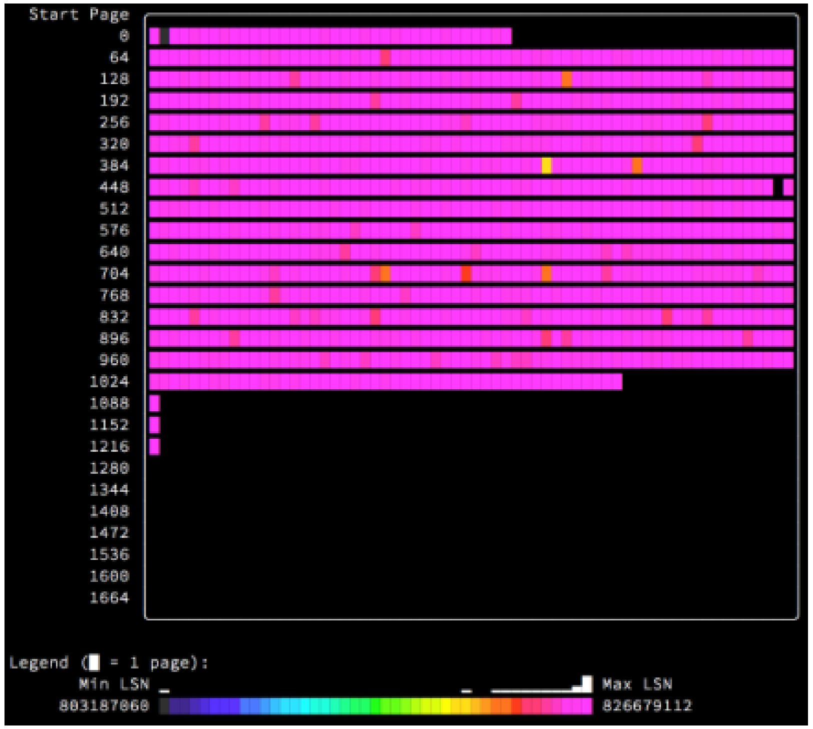
B+-trees (1)



B+-Tree Structure of InnoDB
Infimum
Page 3
Supremum
≥0 <4
Next Record
≥4 <5
Level 2 (Internal)
I
Page 4
S
≥0 <6
≥2 <7
Next Page
Previous Page
I
Page 5
S
≥4 <8
≥6 <9
Level 1 (Internal)
Page 6
0 A
1 B
Page 7
2 C
3 D
Page 8
4 E
5 F
Page 9
6 G
7 H
Level 0 (Leaf)

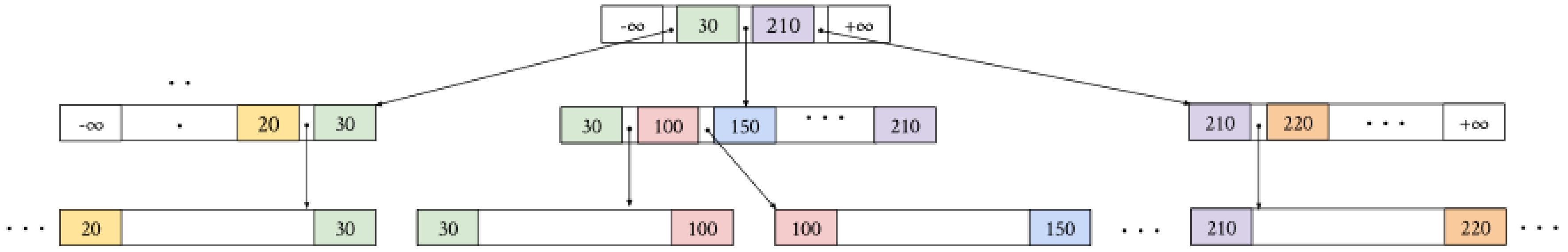
B+-trees (2) Freshness of PageLSNs



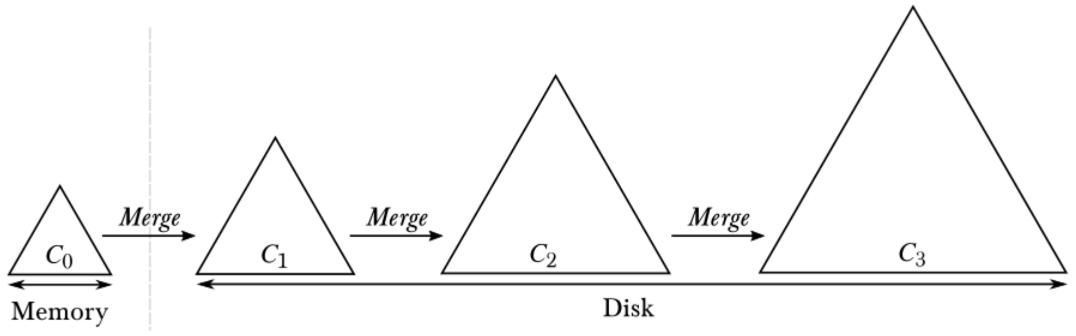


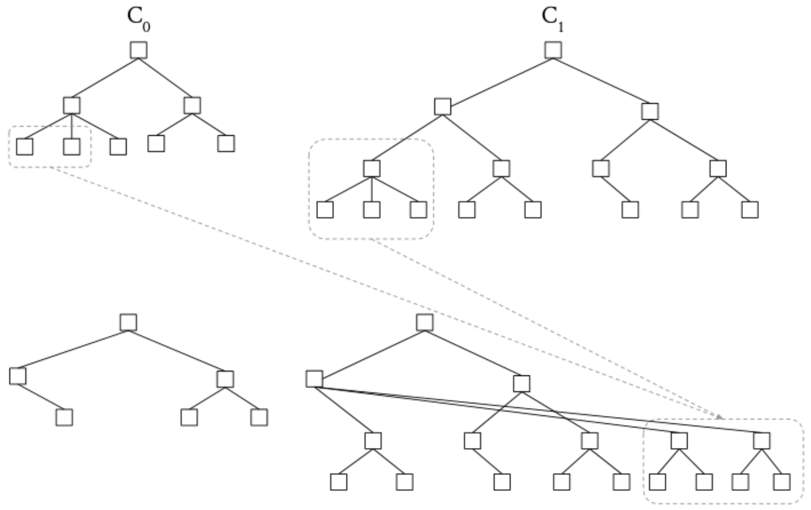
Variants of B+-trees

- *FB-tree*: Variable sized blocks simulating FTL’s GC algorithm. (disk-aware algorithm)
- *Copy-on-write B-tree*: Copy a block to a new location at updates. E.g. LMDB. BTRFS (file system on Linux).
- *Write-optimized B-tree*: No sideway pointers, index records contain lowKey and highKey of the leaf blocks. Prevents multiple writes due to pointers when moving a block.



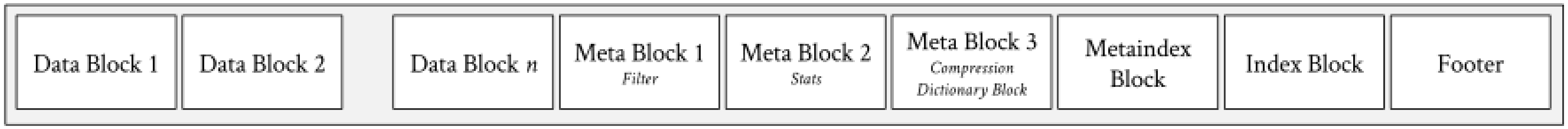
LSM-trees (O'Neil & O'Neil, 1996)





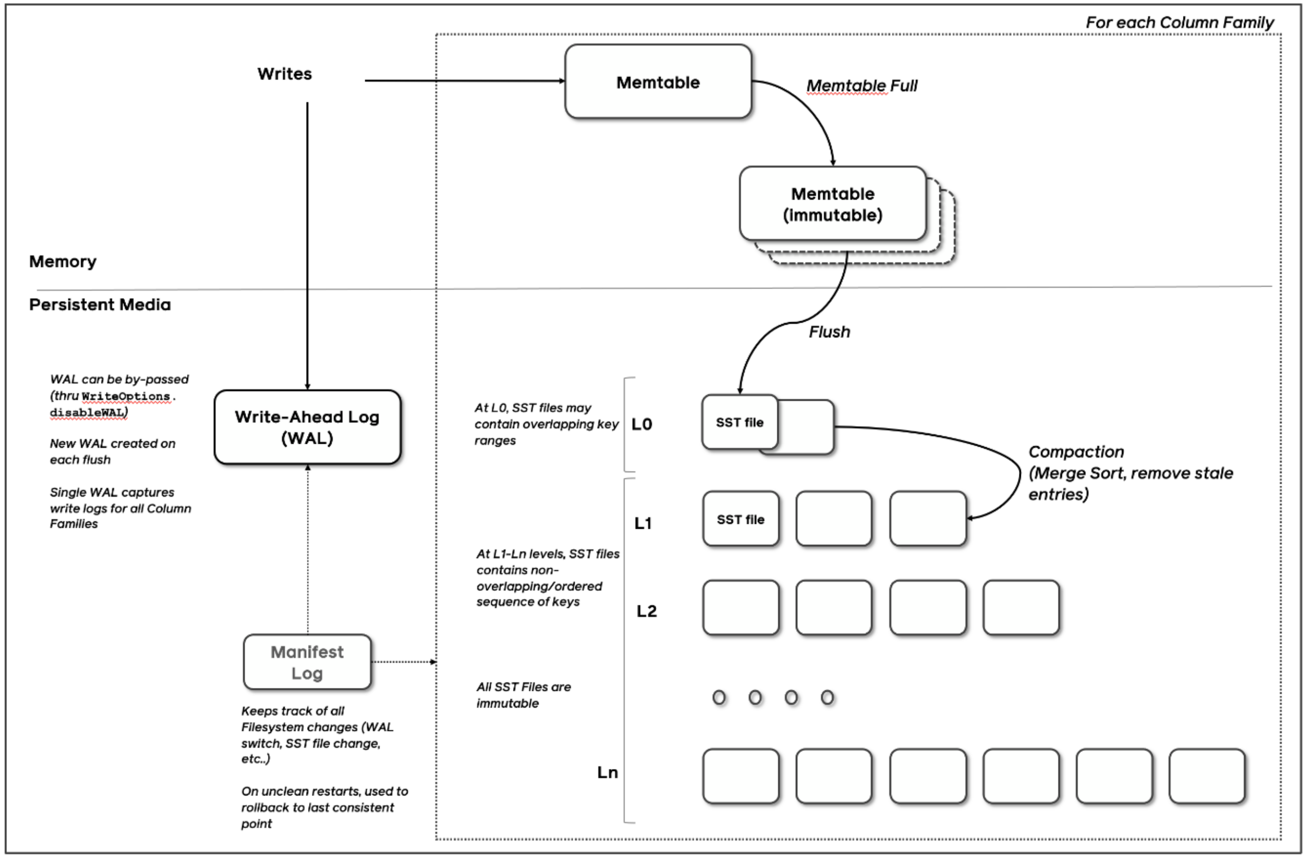
RocksDB

- Builds on Google’s LevelDB and Apache HBase
- Multi-threaded compaction
- Multi-threaded insertion into MemTable
- Extensive control over Bloom filters
- Multicore and SSD support
- 10 x improved write performance compared to LevelDB due to multi-threaded compactions



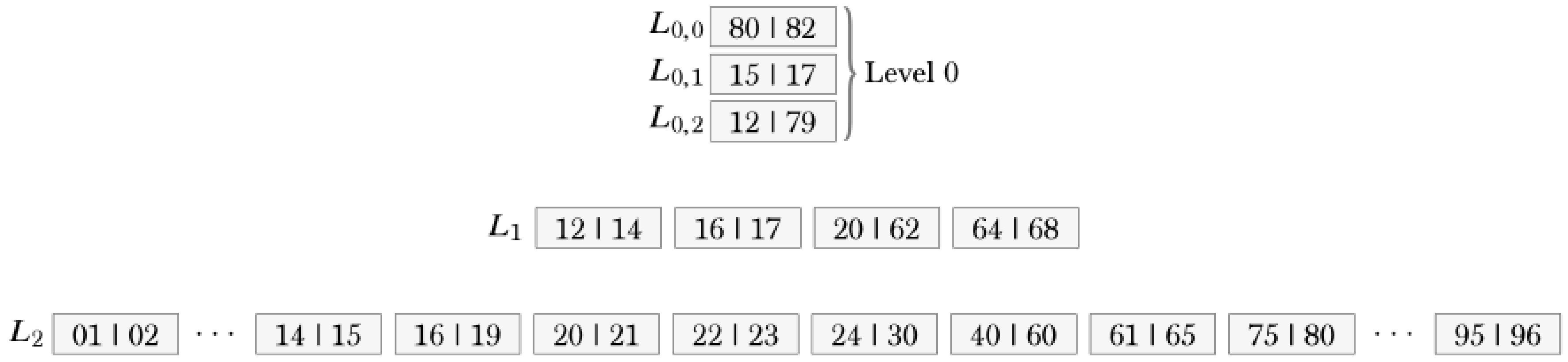
RocksDB (2)

- SST File (sorted key ranges)
- Memtable
- Write Ahead Log


For each Column Family
Writes
Memtable
Memtable Full
Memtable (immutable)
Memory
Persistent Media
Flush
WAL can be by-passed (thru WriteOptions.disableWAL)
New WAL created on each flush
Single WAL captures write logs for all Column Families
Write-Ahead Log (WAL)
At L0, SST files may contain overlapping key ranges
L0
SST file
Compaction (Merge Sort, remove stale entries)
L1
SST file
At L1-Ln levels, SST files contains non-overlapping/ordered sequence of keys
L2
All SST Files are immutable
Ln
Manifest Log
Keeps track of all Filesystem changes (WAL switch, SST file change, etc..)
On unclean restarts, used to rollback to last consistent point

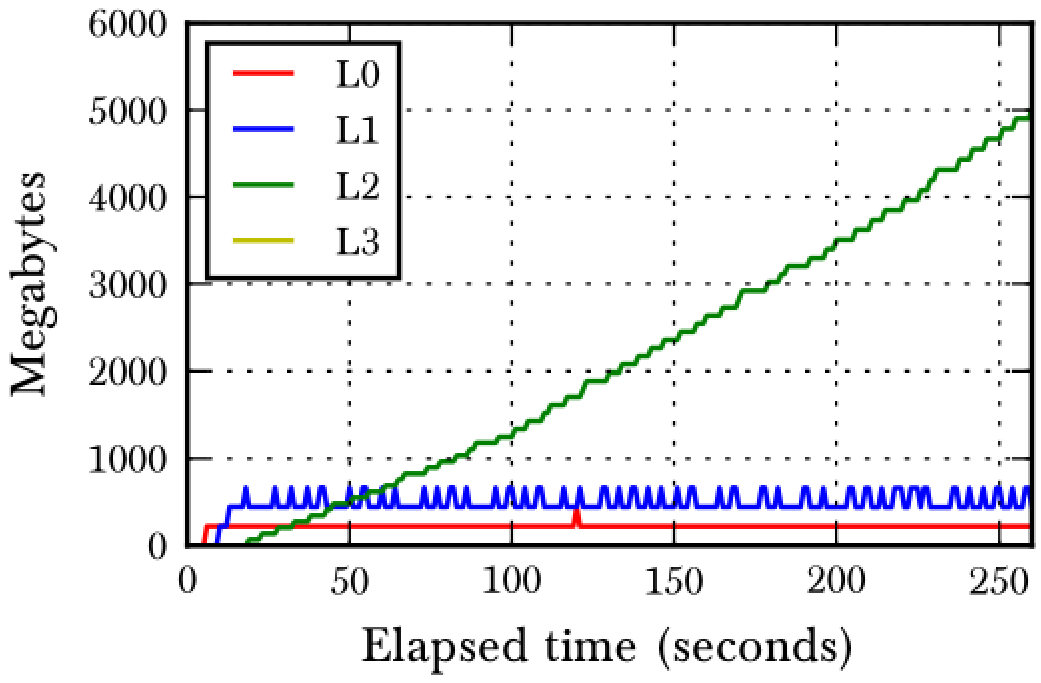
Leveled Compaction

- Original compaction style in LevelDB
- SSTable files stored in multiple levels
- Multiple overlapping SSTables at level 0 to get fast write of MemTables



$L_{0,0}$ 80 | 82
$L_{0,1}$ 15 | 17
$L_{0,2}$ 12 | 79
Level 0
L_1 12 | 14, 16 | 17, 20 | 62, 64 | 68
L_2 01 | 02 ··· 14 | 15, 16 | 19, 20 | 21, 22 | 23, 24 | 30, 40 | 60, 61 | 65, 75 | 80 ··· 95 | 96

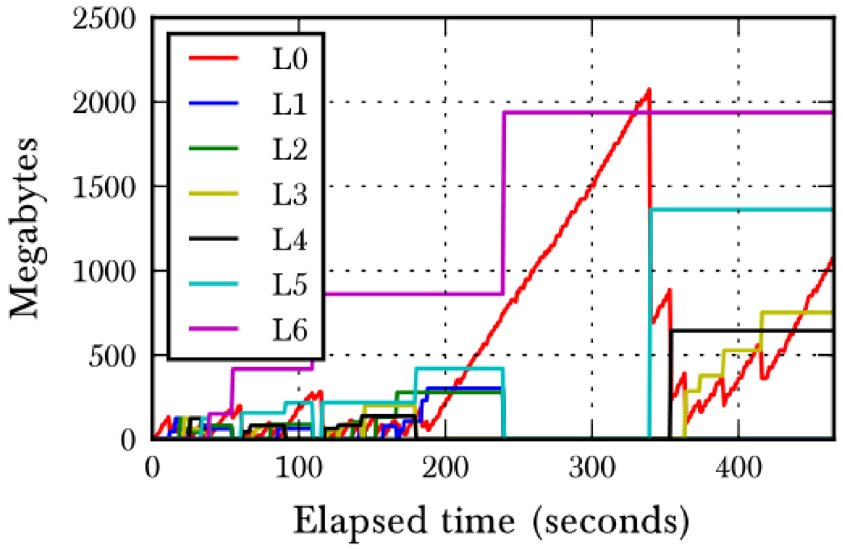
Leveled Compaction (2)



(a) Leveled compaction.

Universal Compaction

- SSTables overlap in key range, but not in time
- Merging SSTables to cover bigger time ranges
- Universal typically results in lower write-amplification but higher space- and read-amplification than Level Style Compaction.



(b) Universal compaction.

Bloom filter

- Simple data structure to quickly check if a key *may* exist in a dataset
- Bitmap of m bits and k unique hash functions
- Either used on each block or the complete SSTables

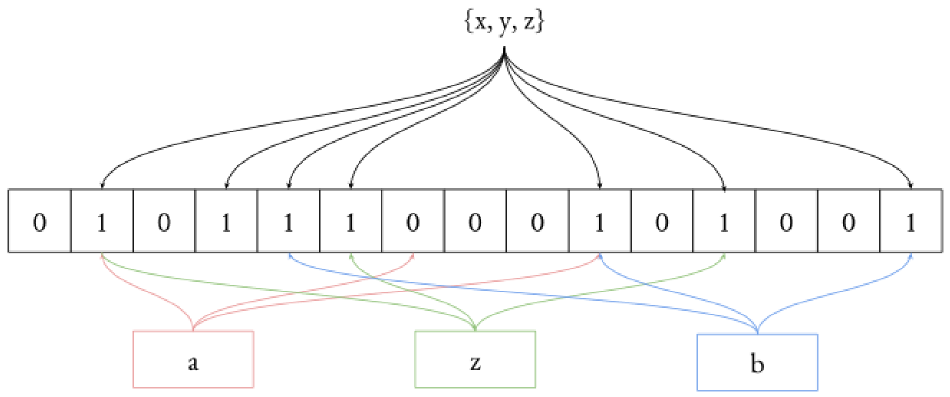


Figure 2.25: Example of a Bloom filter with the dataset x, y, z and $m = 15$ bits. There are $k = 3$ different hash functions.

MyRocks

- Facebook’s integration of RocksDB as a storage engine with MySQL
- Created due to serious space and write amplification of MySQL’s InnoDB
- MariaDB (MySQL competitor) is also integrating RocksDB as a storage engine

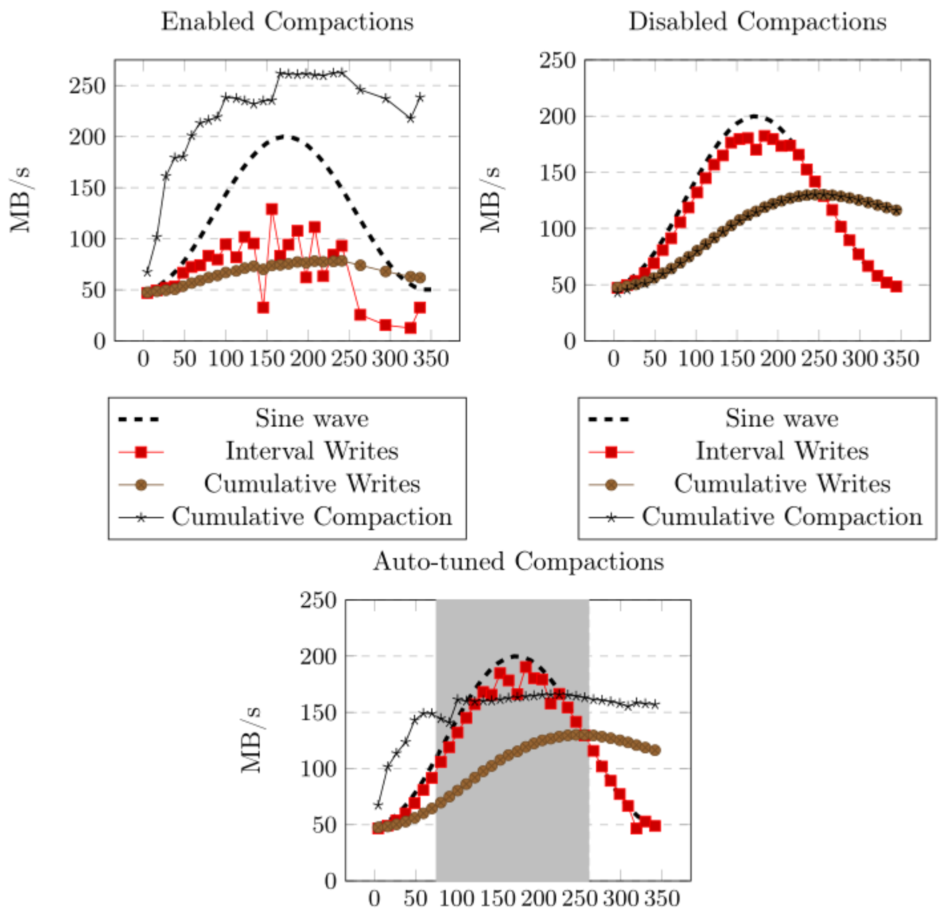
Write amplification

- #bytes written to database / #bytes written to api
- B-trees: write amp less with big records
- LSM-trees: less write amp due to big write chunks, but compaction require more writes
- LSM-trees: write amp independent on record size
- May be measured by insert throughput. RocksDB performs very well

Write stalls and write stops

- Write stalls: Slow the speed of inserts to cope with compaction
- Write stops: Stop inserts.
- RocksDB can configure parameters to control this:
  - Number of MemTables
  - Number of SST files at level 0
  - Number of bytes awaiting compactions
- Hans-Wilhelm Kirsch Warlo developed an auto-tuner for compactions in RocksDB during his master thesis: Turns off compactions during high insert loads (spring 2018).

Warlo's auto-tuner


Enabled Compactions
Disabled Compactions
Auto-tuned Compactions
MB/s
Sine wave
Interval Writes
Cumulative Writes
Cumulative Compaction