Key-Value Store: Database for Unstructured Bigdata

September 2, 2021
Jongmoo Choi
Dankook University
http://embedded.dankook.ac.kr/~choijm

Content

- What is Key-Value Store?
  - ✓ What is Bigdata?
  - ✓ Structured vs. Unstructured data
  - ✓ Instances and Recent Studies about Key-Value Store
- RocksDB Basic
- Advance Topic 1: Compaction
- Advance Topic 2: Lookup
- Advance Topic 3: WAL
- Advance Topic 4: New approach
- RocksDB Practice
- Conclusion

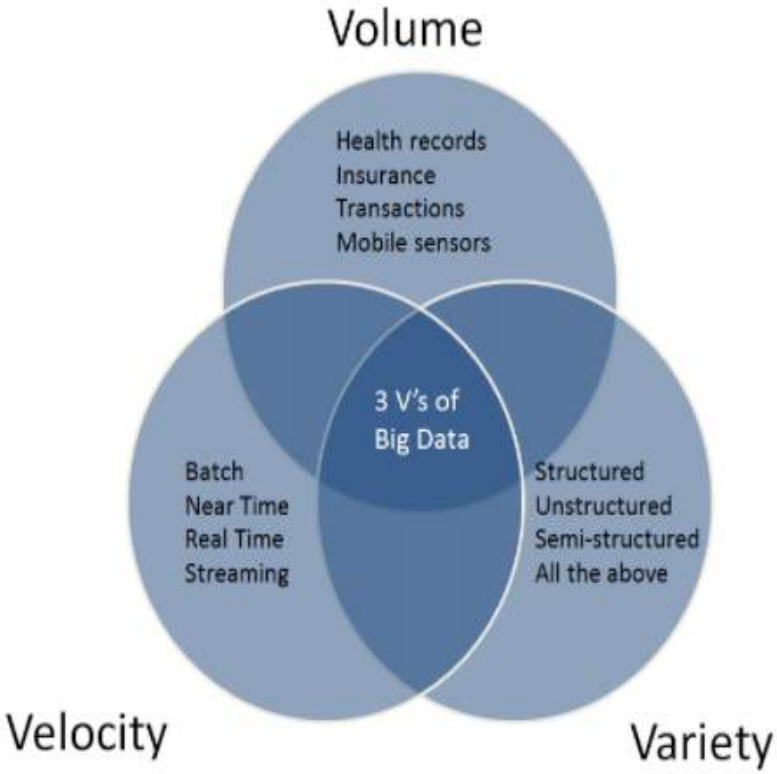
What is Key-Value Store? (1/12)

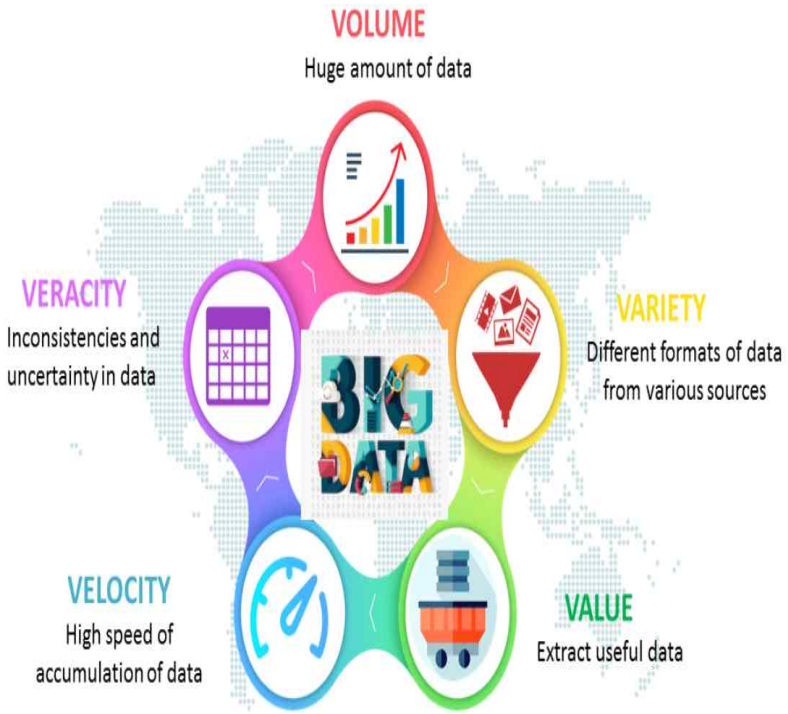
- Lecture Objective
  - ✓ Understand characteristics of Bigdata.
  - ✓ Can explain differences between structured and unstructured data
  - ✓ Know about NoSQL, Database for unstructured data
  - ✓ Explore real instances of Key-Value Store (a kind of NoSQL)
  - ✓ Investigate research issues of Key-Value Store

(Source: www.dreamstime.com)

What is Key-Value Store? (2/12)

- What is Bigdata?
  - ✓ Three features: Volume, Velocity, Variety
    - These days, being extended to Five or Eight features

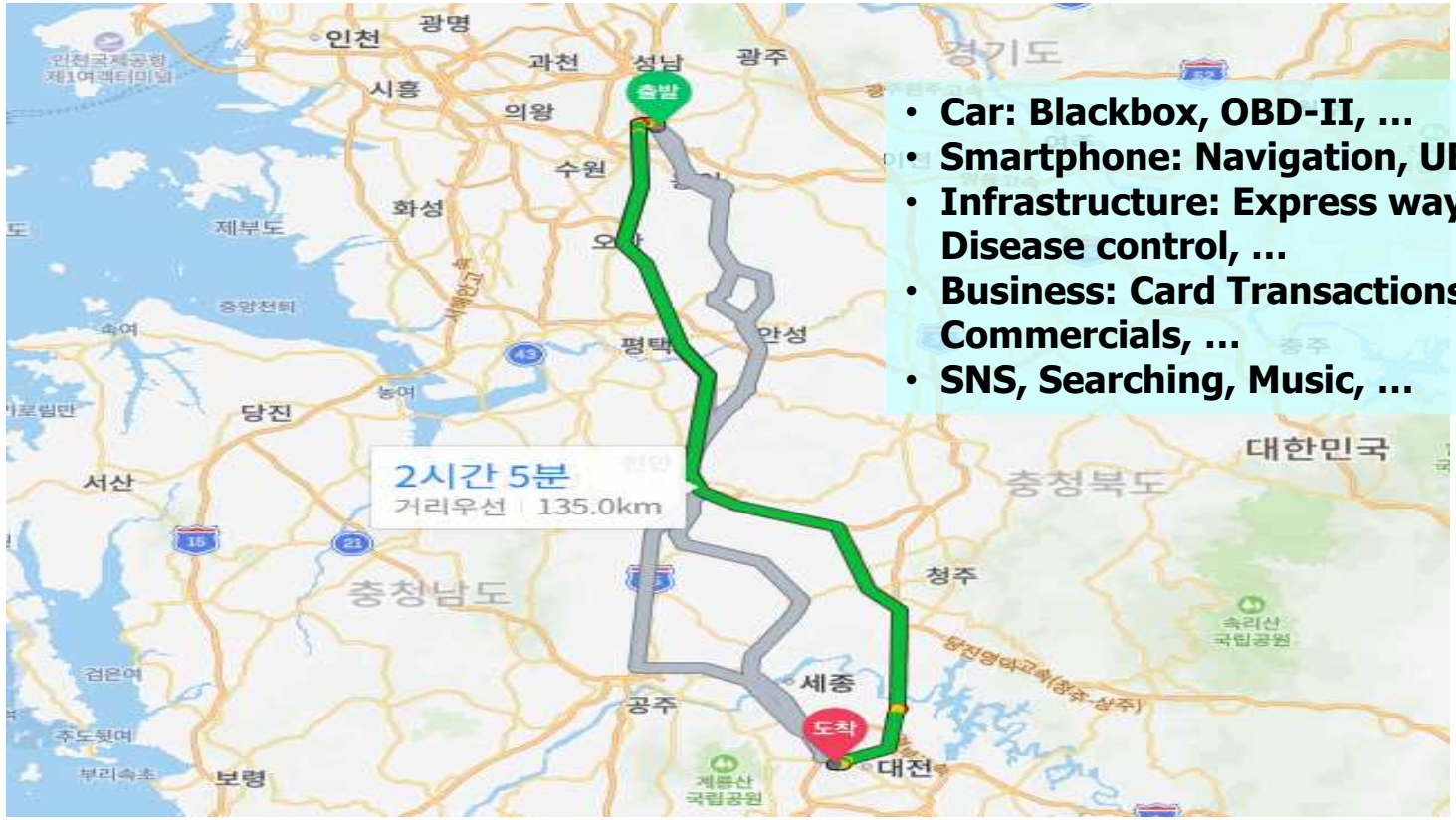




(Source: https://bigdataldn.com/intelligence/big-data-the-3-vs-explained/, https://laptrinhx.com/what-is-big-data-a-beginner-s-guide-to-the-world-of-big-data-1936020691/)

What is Key-Value Store? (3/12)

- Volume and Velocity Perspective
  - ✓ Data gathered while driving

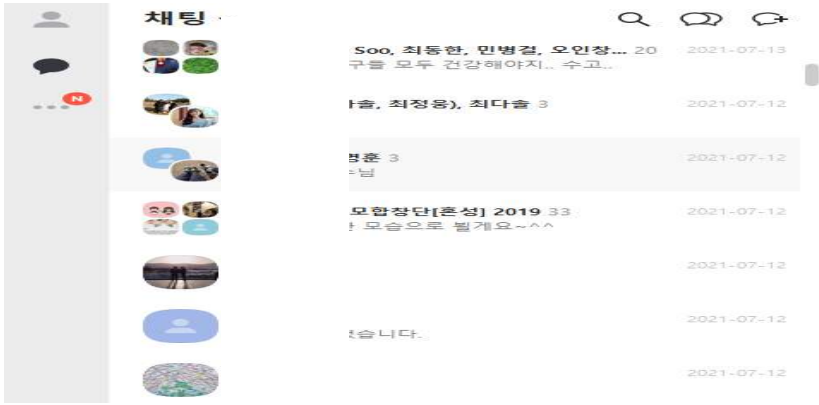


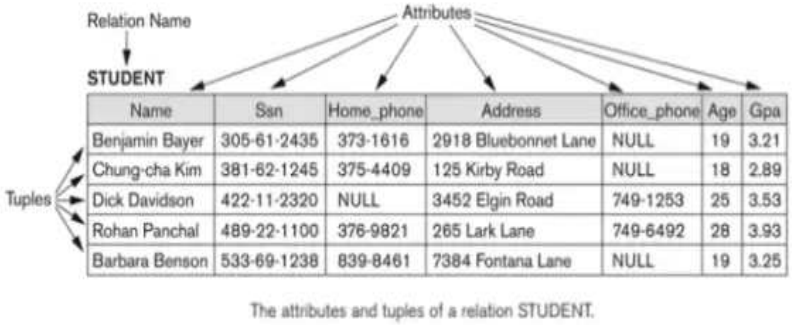
- **Car: Blackbox, OBD-II, …**
- **Smartphone: Navigation, UBI, …**
- **Infrastructure: Express way coop., Disease control, …**
- **Business: Card Transactions, LBS, Commercials, …**
- **SNS, Searching, Music, …**

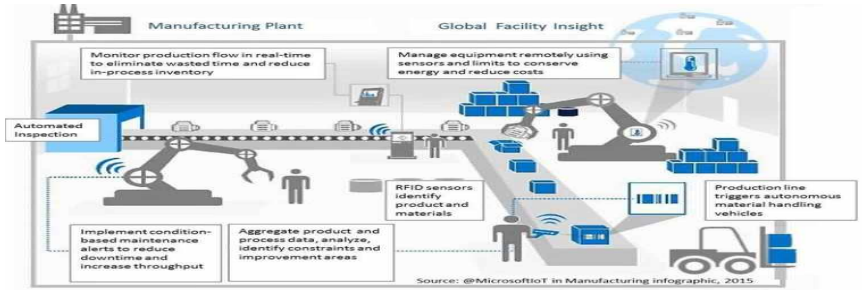
  - ✓ 25GB per hour (from IEEE AutoSafety, 2020)

What is Key-Value Store? (4/12)

- How about Variety?
  - ✓ Conventional data: well defined format (schema)
    - Value, Field (Column, Attribute), Record (Row, Tuple), Table, Relation
  - ✓ Bigdata: various and diverse (schema free)
    - Various fields/frequencies in each record, Diverse format, …



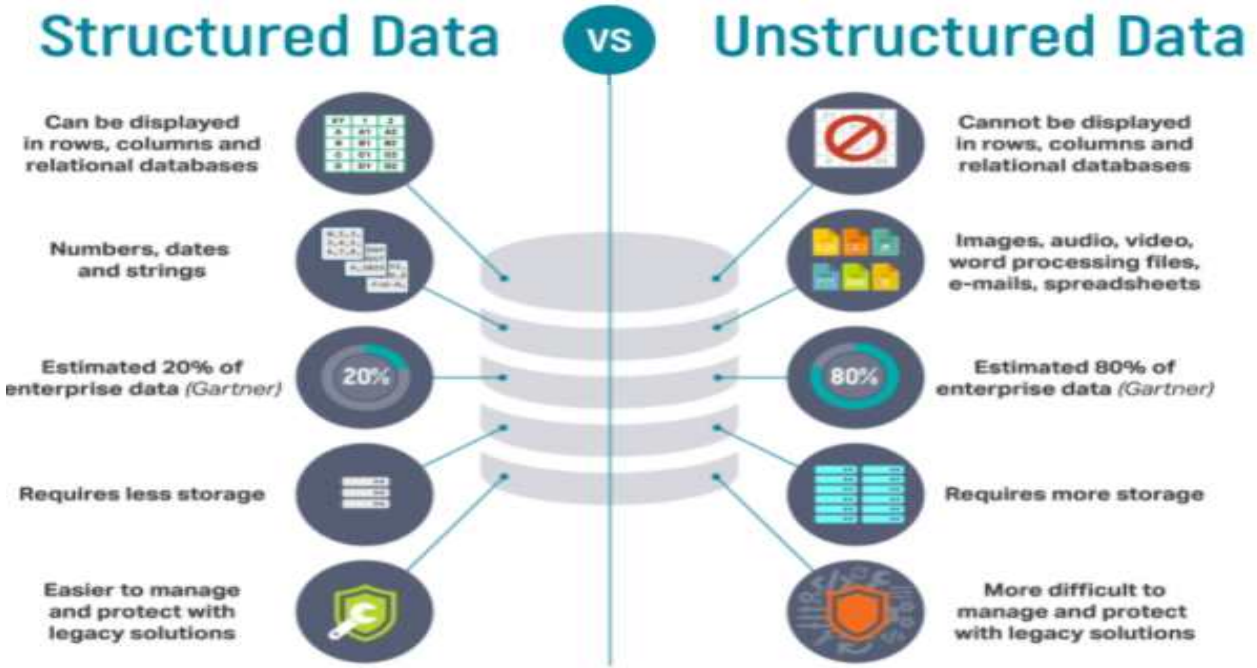


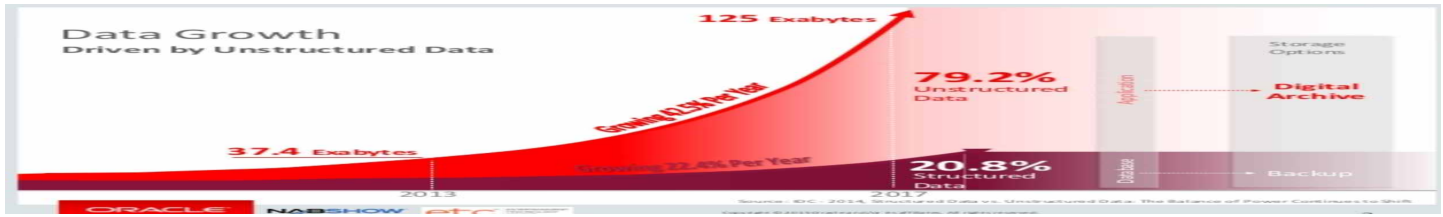


(Source: www.slideshare.net/b15ku7/chapter-2-relational-data-modelpart1/ and www.researchgate.net/figure/Shows-an-example-of-product-and-data-flow-in-a-Smart-Factory-Product-carries-RFID-tag_fig1_327884684)

What is Key-Value Store? (5/12)

- Types of Data
  - ✓ Structured vs. Unstructured (vs. Semi-structured)

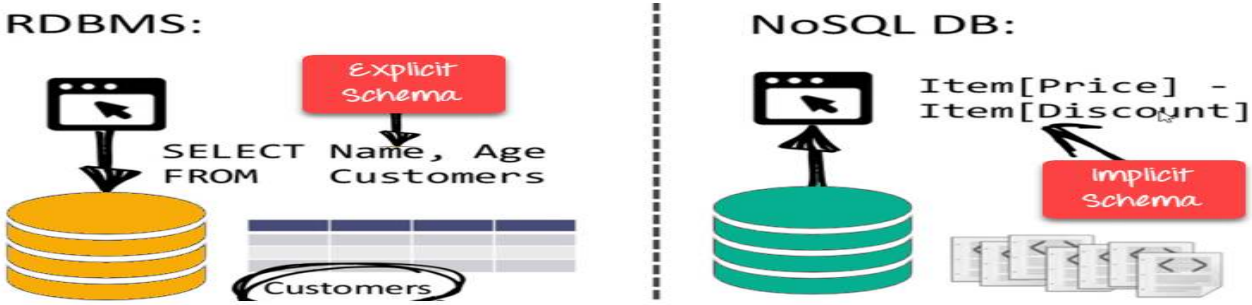




(Source: https://lawtomated.com/structured-data-vs-unstructured-data-what-are-they-and-why-care/ and https://www.linkedin.com/pulse/deconstructing-internet-things-oil-gas-keith-moore/)

What is Key-Value Store? (6/12)

- Types of Database
  - ✓ RDB(Relational DB) for structured vs. Non RDB for unstructured
    - Also known as SQL DB vs. NoSQL DB (Note: Not-Only SQL!)



(Source: https://www.guru99.com/nosql-tutorial.html/)

  - ✓ Example of RDB
    - Infer information via the relation of tables

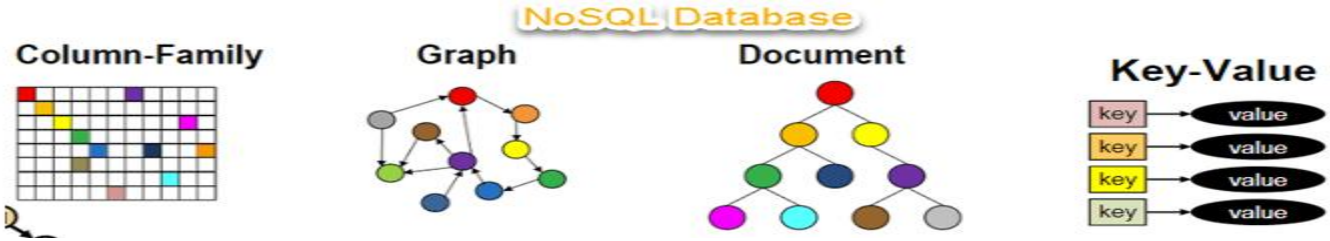
PubID	Publisher	PubAddress
03-4472822	Random House	123 4th Street, New York
04-7733903	Wiley and Sons	45 Lincoln Blvd, Chicago
03-4859223	O'Reilly Press	77 Boston Ave, Cambridge
03-3920886	City Lights Books	99 Market, San Francisco

AuthorID	AuthorName	AuthorBDay
345-28-2938	Haile Selassie	14-Aug-92
392-48-9965	Joe Blow	14-Mar-15
454-22-4012	Sally Hemmings	12-Sept-70
663-59-1254	Hannah Arendt	12-Mar-06

ISBN	AuthorID	PubID	Date	Title
1-34532-482-1	345-28-2938	03-4472822	1990	Cold Fusion for Dummies
1-38482-995-1	392-48-9965	04-7733903	1985	Macrame and Straw Tying
2-35921-499-4	454-22-4012	03-4859223	1952	Fluid Dynamics of Aquaducts
1-38278-293-4	663-59-1254	03-3920886	1967	Beads, Baskets & Revolution

What is Key-Value Store? (7/12)

- Types of Database (cont')
  - ✓ Classification of NoSQL DB



(Source: www.guru99.com/nosql-tutorial.html/)

- Key-Value DB (Key Value Store)
  - Manage key and value pair ➔ simple
  - Examples: LevelDB, RocksDB, Redis, ..
- Document DB
  - Extend KV store, KVs are stored in an organized format (like XML)
  - Examples: MongoDB, Couchbase, …
- Column-family (wide column) DB
  - Column-oriented KV management (sparse)
  - Examples: Bigtable, HBase, Cassandra, …
- Graph DB
  - KV pairs and their relationships
  - Examples: Neo4j, JanusGraph, …


```
{
  "id": "webShop635874323254478467",
  "transactionDateUTC": "2016-01-03T15:38:45.44884722",
  "lines": [
    {
      "articleNo": "12345678",
      "articleName": "Bookcase White Rodney 233x100",
      "quantity": 1,
      "itemPrice": 44.45
    }
  ],
  "invoiceAddress": {
    "addressType": "DELIVERY",
    "addressLine1": "The big City Street 2233",
    "addressLine2": "Dept Invoice",
    "city": "Malmö",
    "zipCode": 11111
  },
  "customer": {
    "firstName": "Peter",
    "lastName": "InTheAzureSky",
    "email": "peter@example.com",
    "mobilePhone": "+4642000000"
  }
}
```

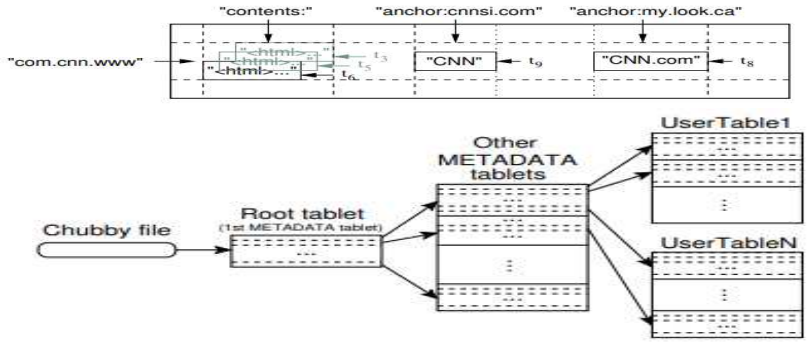

(Source: peter.intheazuresky.com/)

What is Key-Value Store? (8/12)

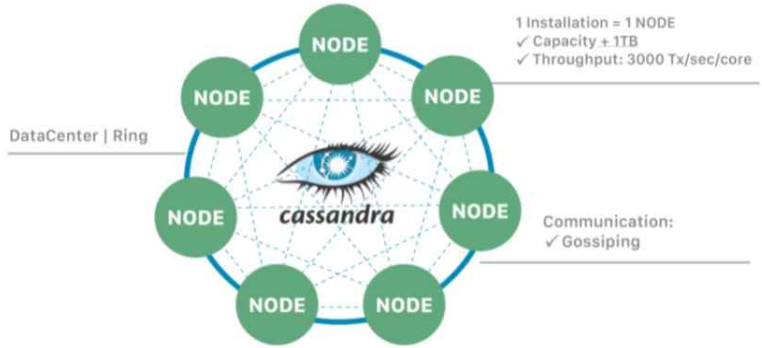
- Key-Value Store (a.k.a. Key-Value DB)
  - ✓ A de-facto standard DB for unstructured data
  - ✓ Google, Facebook, Amazon, Microsoft, MongoDB, Yahoo, Hbase, LinkedIn, Oracle, Baidu, Basho, In Memory DB (Memcached), …

What is Key-Value Store? (9/12)

- Key-Value Stores: some examples
  - ✓ LevelDB
    - By Google, 2011, a subset of Bigtable (Column-oriented DB, OSDI, 2006)
    - Level compaction, Open-source
  - ✓ RocksDB
    - By Facebook, 2012, a fork of LevelDB
    - Various algorithms (e.g. Tier compaction), 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 (Facebook), 2008, motivated by Amazon's Dynamo
    - A decentralized architecture, each node is powered by LST-tree based DB



(Source: Bigtable, index by row key, column key, timestamp)



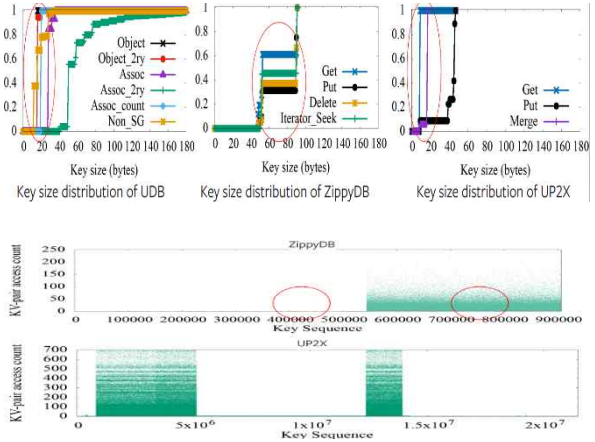
(Source: cassandra.apache.org/_/cassandra-basics.html/)

What is Key-Value Store? (10/12)

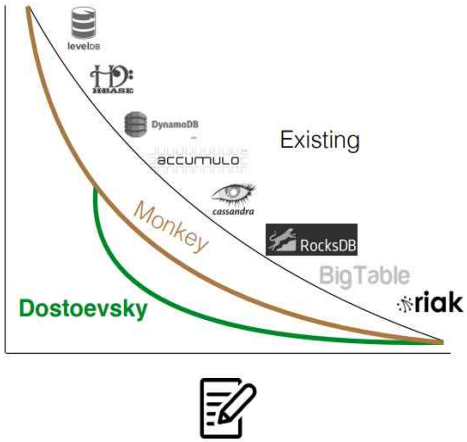
- Key-Value Store: Hot research topics not only academia but also industry
  - ✓ S. Dong et al., “Evolution of Development Priorities in Key-value Stores Serving Large-scale Applications: The RocksDB Experience”, FAST’21.
  - ✓ L. Lu et al., “WiscKey: Separating Keys from Values in SSD-conscious Storage”, FAST’16.
  - ✓ Y. Dai et al., “From WiscKey to Bourbon: A Learned Index for Log-Structured Merge Trees”, OSDI’20.
  - ✓ P. Raju et al., “PebblesDB: Building Key-Value Stores using Fragmented Log-Structured Merge Trees”, SOSP’17.
  - ✓ O. Balmau et al., “SILK: Preventing Latency Spikes in Log-Structured Merge Key-Value Stores”, ATC’19.
  - ✓ W. Zhong et al., “REMIX: Efficient Range Query for LSM-trees”, FAST’21.
  - ✓ J. Im et al., “PinK: High-speed In-storage Key-value Store with Bounded Tails”, ATC’20.
  - ✓ T. Yao et al., “MatrixKV: Reducing Write Stalls and Write Amplification in LSM-tree Based KV Stores with a Matrix Container in NVM”, ATC’20.
  - ✓ H. Chen et al, “SpanDB: A Fast, Cost-Effective LSM-tree Based KV Store on Hybrid Storage”, FAST’21
  - ✓ G. DeCandia et al., “Dynamo: Amazon’s Highly Available Key-value Store”, SOSP’07.
  - ✓ L. Bindschaedler et al., “Hailstorm: Disaggregated Compute and Storage for Distributed LSM-based Databases”, ASPLOS’20.
  - ✓ N. Dayan et al., "Dostoevsky: Better Space-Time Trade-Offs for LSM-Tree Based Key-Value Stores via Adaptive Removal of Superfluous Merging, SIGMOD’18.
  - ✓ Y. Matsunobu et al., "MyRocks: LSM-Tree Database Storage Engine Serving Facebook's Social Graph“, VLDB’20
  - ✓ …

What is Key-Value Store? (11/12)

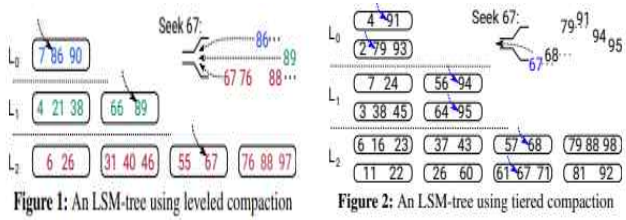
- Interesting recent studies regarding Key-Value Store

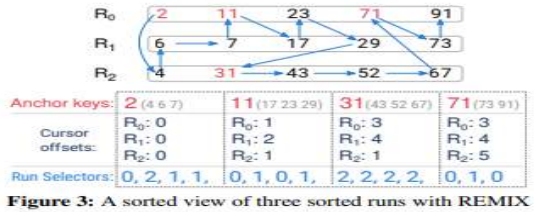


(RocksDB analysis, FAST'20)

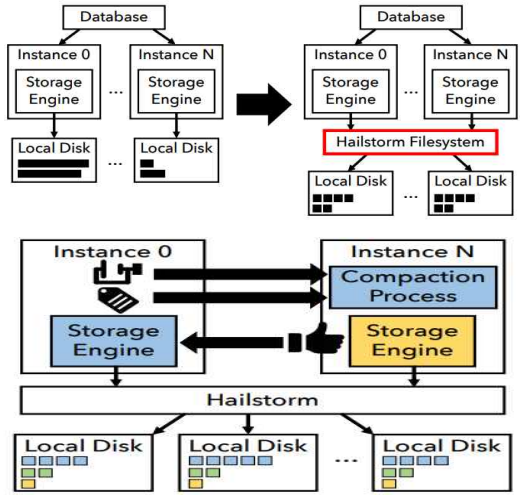


(Dostoevsky, SIGMOD'18)

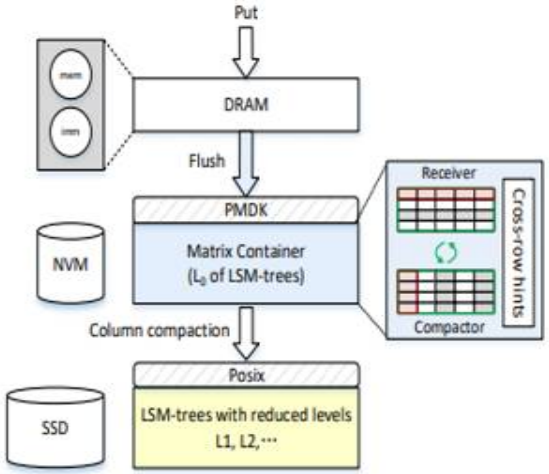




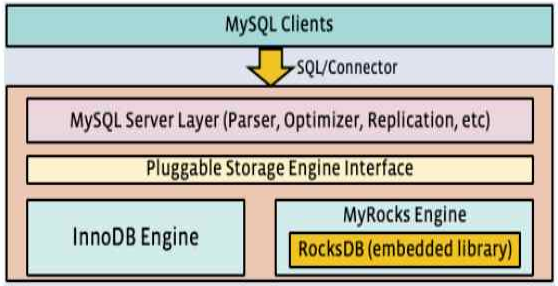
(REMIX, FAST'21)



(Hailstorm, ASPLOS'20)



(MatrixKV, ATC'20)



Engine	Space	CPU seconds/s for writes	CPU seconds/s for reads + writes	Bytes written per second
InnoDB	2187.4GB	0.89	1.83	13.34MB
MyRocks	824.4GB	0.55	1.65	3.42MB

(MyRocks, VLDB'20)

What is Key-Value Store? (12/12)

- Take-away Lessons
  - ✓ Understand characteristics of Bigdata
    - Volume, Velocity, Variety ➔ 80% unstructured data
  - ✓ NoSQL: Database for unstructured data
    - Key-Value store, Document DB, Column-family DB, Graph DB
  - ✓ Key-Value Store examples
    - LevelDB, RocksDB, Bigtable, HBase, Dynamo, …
    - Wiskey, PebblesDB, SILK, MatrixDB, Hailstorm, Pink, …
  - ✓ Popularly studied from both industry and academy
    - Performance
      - Write: Compaction, Flush, Background threads, …
      - Read: Bloom filter, Range query, Parallel access, …
    - Consistency: WAL, Recovery, …
    - New storage: NVM + SSD, Optane, …
    - Distributed environment: partition, replication, …

(Source: www.dreamstime.com)

(Special thank you to DKU Embedded Members and IITP's SW StarLab (No.2021-0-01475)

Content

- What is Key-Value Store?
- RocksDB Basic
  - ✓ Definition and Use cases
  - ✓ Architecture: LSM-tree data structure
  - ✓ Interface and Internal Operations
- Advance Topic 1: Compaction
- Advance Topic 2: Lookup
- Advance Topic 3: WAL
- Advance Topic 4: New approach
- RocksDB Practice
- Conclusion

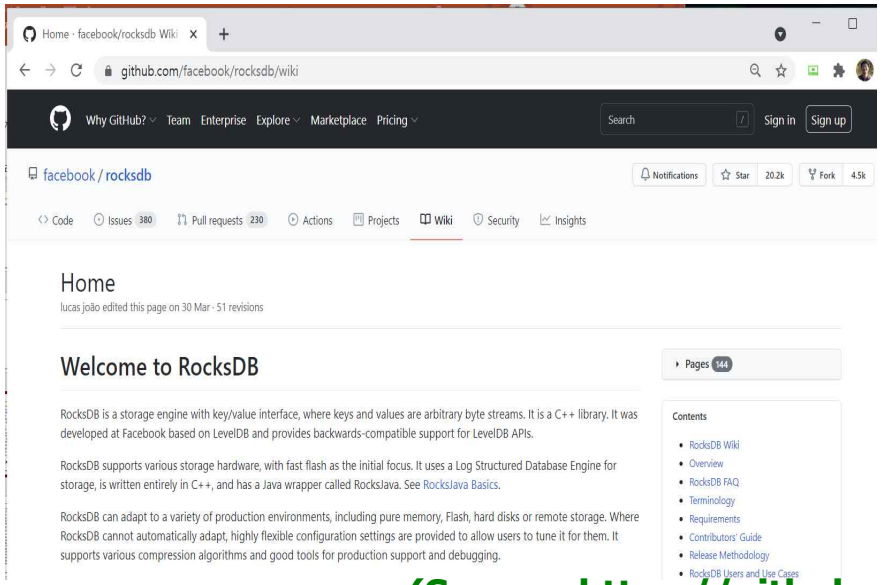
RocksDB Basic (1/15)

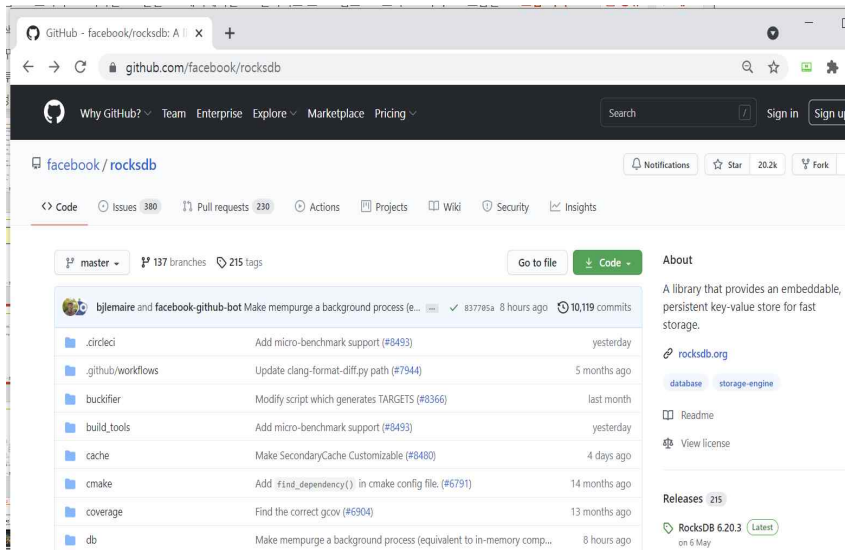
- Lecture Objective
  - ✓ Understand the features of RocksDB
  - ✓ Learn the core algorithm of RocksDB: LSM (Log-Structured Merge)-tree
  - ✓ Examine how the key-value interface works: put and get
  - ✓ Investigate how the internal operations works: flush, compaction and bloom filter
  - ✓ Explore RocksDB development history

(Source: www.dreamstime.com)

RocksDB Basic (2/15)

- What is RocksDB? (from RocksDB wiki)
  - ✓ 1) Famous KV Store of Facebook, 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 (e.g. compaction, filtering, format), configurations (e.g. size, BG thread), tools and debugging facilities

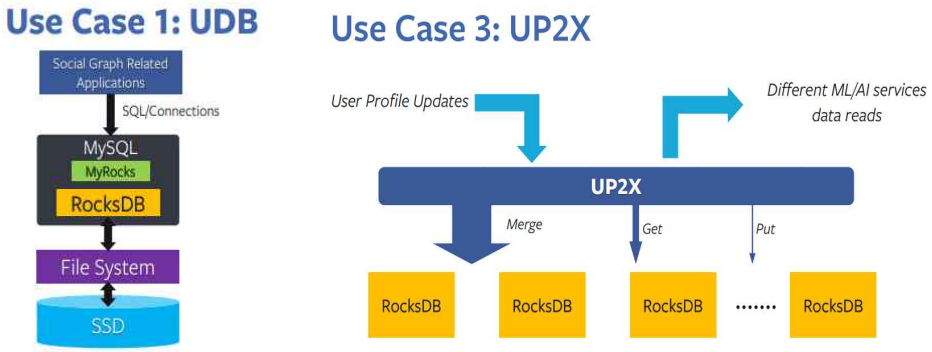


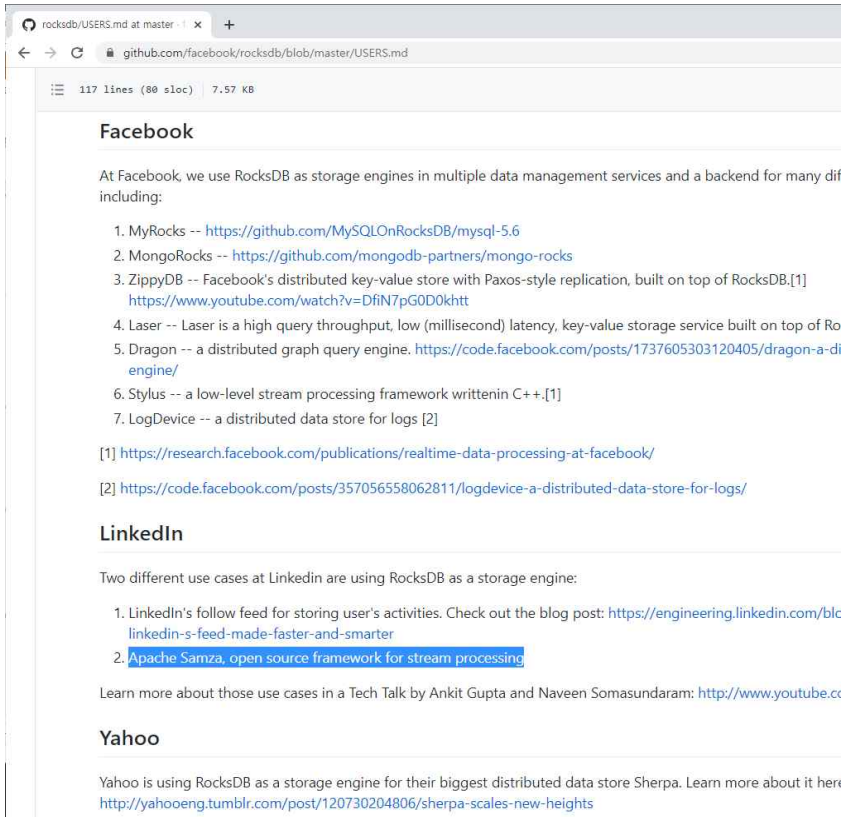


(Source: https://github.com/facebook/rocksdb/wiki)

RocksDB Basic (3/15)

- Use cases
  - ✓ 30 applications at Facebook alone (UDB for SQL, ZippyDB for distributed system, UP2X for Logging and ML)
  - ✓ Storage engine for other services: LinkedIn, Yahoo, Kafka, Netflix, …





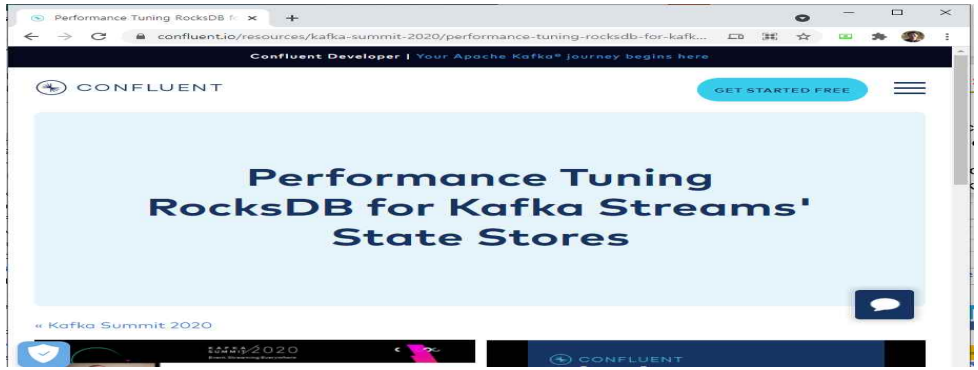
(Source: Facebook RocksDB paper in FAST'20 and https://github.com/facebook/rocksdb/blob/master/USERS.md)

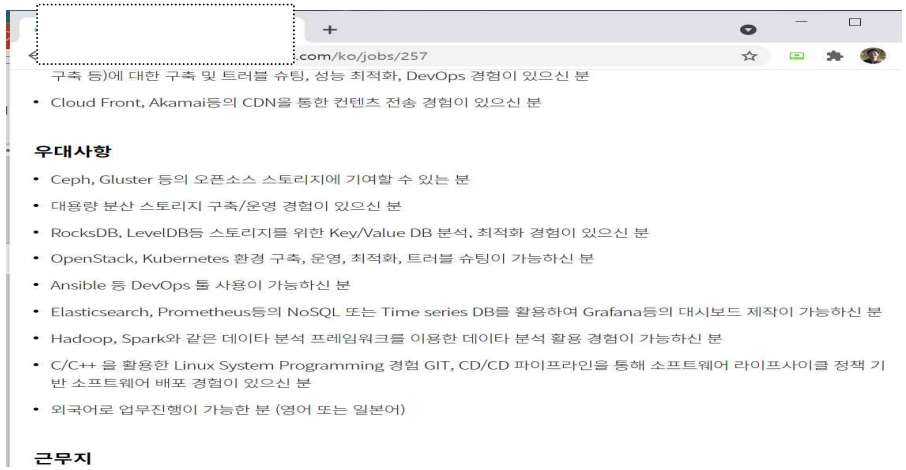
RocksDB Basic (4/15)

- Use cases (cont')
  - ✓ Types: DB, Streaming, Logging, Indexing, Caching, …
  - ✓ Core infrastructure for Bigdata service: need engineers

	Read/Write	Read Types	Special Characteristics
Databases	Mixed	Get + Iterator	Transactions and backups
Stream Processing	Write-Heavy	Get or Iterator	Time window and checkpoints
Logging / Queues	Write-Heavy	Iterator	Support HDD too
Index Services	Read-Heavy	Iterator	Bulk loading
Cache	Write-Heavy	Get	Can drop data

Table 1: RocksDB use cases and their workload characteristics

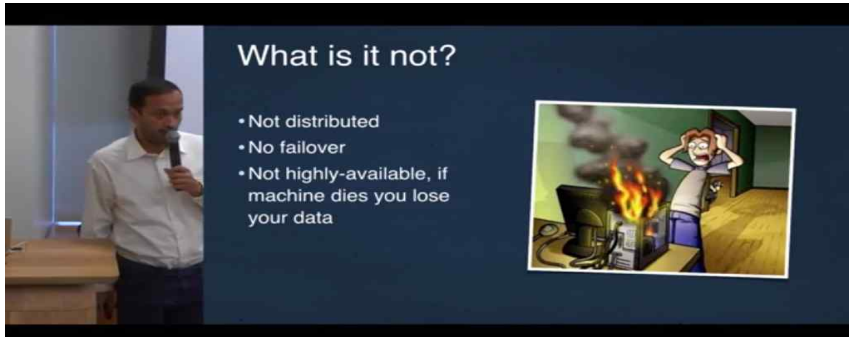


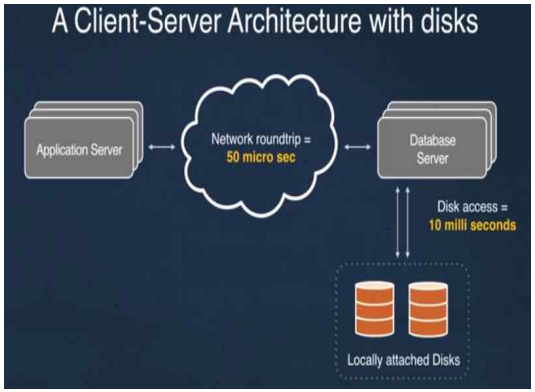


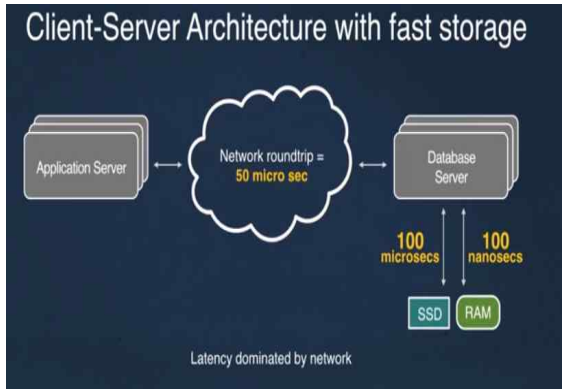
(Source: Facebook RocksDB papers FAST'21, www.confluent.io/resources/kafka-summit-2020 and Google job search)

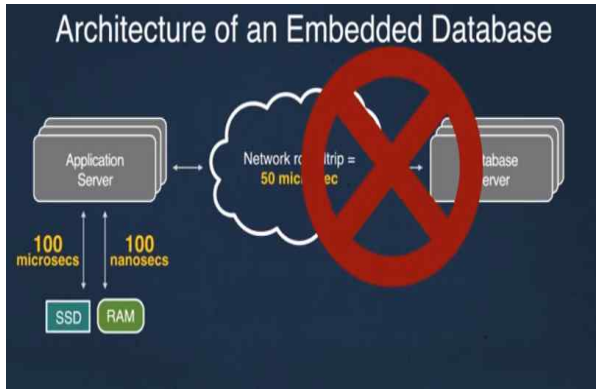
RocksDB Basic (5/15)

- Why Embedded?
  - ✓ Traditional: separate DB servers from Applications
  - ✓ Fast storage: network overhead becomes heavy
  - ✓ Decide to place DB as an embedded in an application





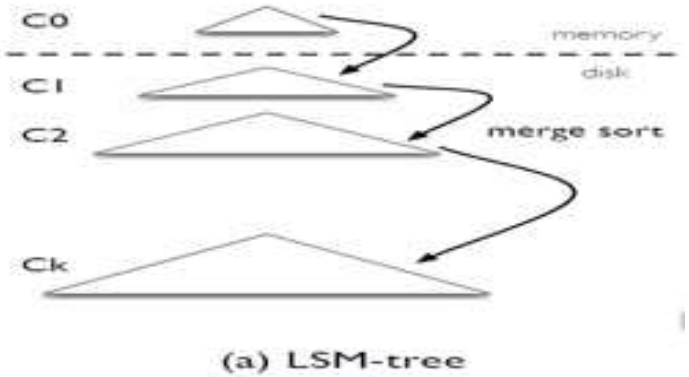




(Source: Dhruba Borthakur, www.youtube.com/watch?v=hHsxqkcZy7c)

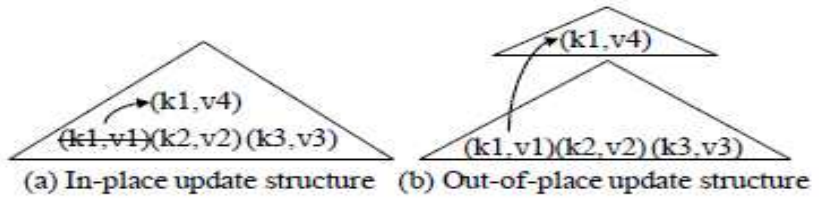
RocksDB Basic (6/15)

- What is (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
    - 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



(a) LSM-tree

(Source: Wisckey paper, FAST'16)



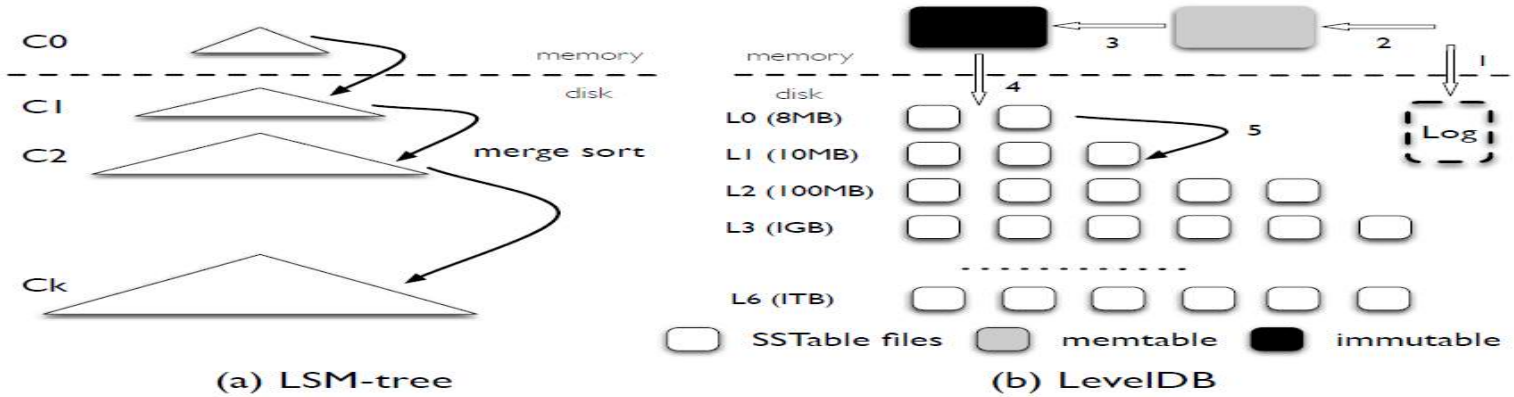
(a) In-place update structure (b) Out-of-place update structure

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'19)

RocksDB Basic (7/15)

- What is (Log Structured Merge)-tree?
  - ✓ 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)
    - Log (WAL) for durability
      - A set of records where each record consists of CRC, size, type and payload



(Source: Wisckey paper, FAST'16)

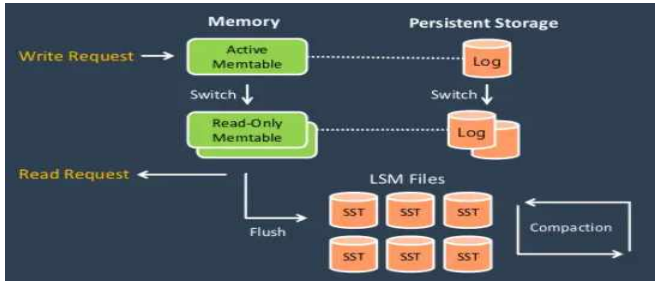
RocksDB Basic (8/15)

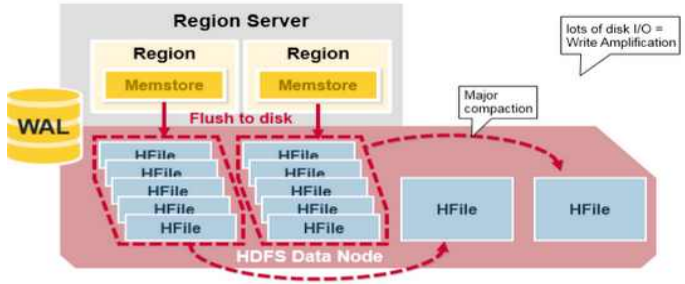
- Key-Value Interface
  - ✓ Key/Value: Arbitrary byte streams
  - ✓ User visible interfaces
    - put, get, range scan(iterator), delete, single delete, …

```
std::string value;
rocksdb::Status s = db->Get(rocksdb::ReadOptions(), key1, &value);
if (s.ok()) s = db->Put(rocksdb::WriteOptions(), key2, value);
if (s.ok()) s = db->Delete(rocksdb::WriteOptions(), key1);
```

(Source: https://github.com/facebook/rocksdb/wiki)

  - ✓ Internal operations
    - Flush: when a Memtable is full, it becomes immutable and written into L1, becoming a new SSTable.
    - Compaction: When Li is full, compaction is triggered and some SSTables are merged into Li+1

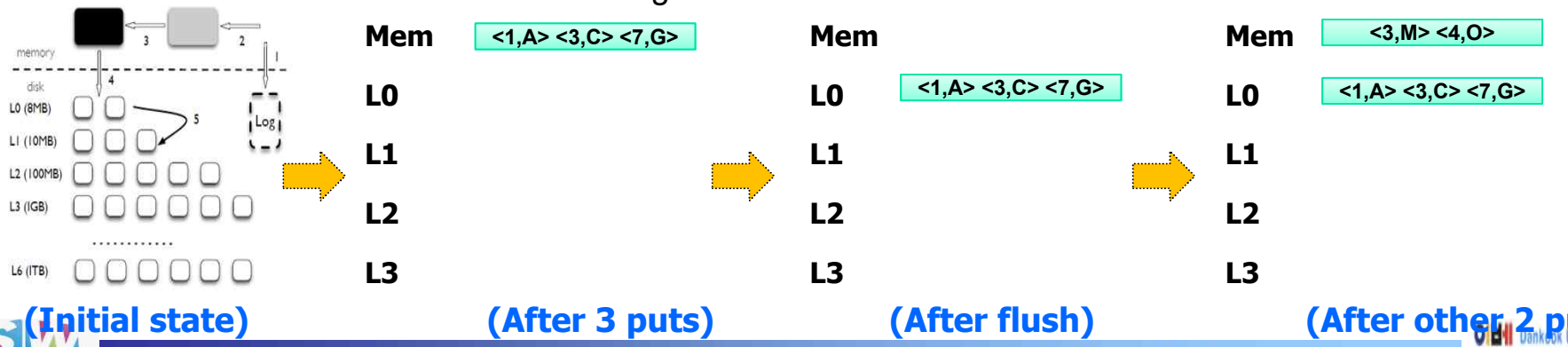




(Source: Dhruba Borthakur and https://nabillera.tistory.com/)

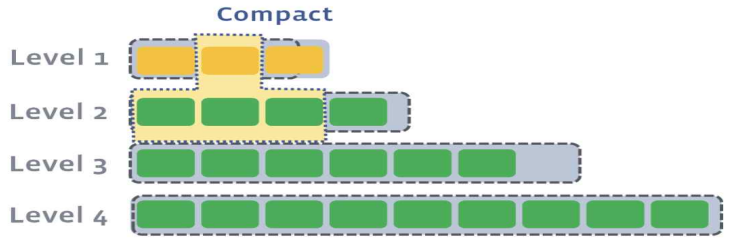
RocksDB Basic (9/15)

- Key-Value Interface
  - ✓ Example 1: put/get and flush
    - Assumption
      - 3 KVs in a Memtable and SSTables
      - L_0: 2 SSTables, $L_{i+1} = L_i * 2$ (default fanout 10)
    - Workload
      - put(“1”, “A”), put(“3”, “C”), put(“7”, “G”)
        - Memtable ➔ immutable, flushed (Background)
        - A New Memtable is allocated
      - put(“3”, “M”), put(“4”, “O”)
        - Note: Out-of-place update
      - get(1), get(3)
        - Recent data at higher level

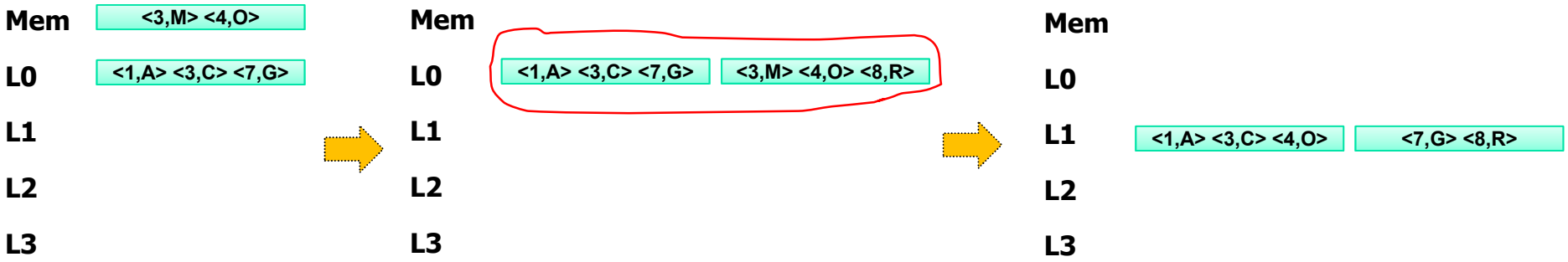


RocksDB Basic (10/15)

- Key-Value Interface
  - ✓ Example 2: compaction
    - Workload
      - put(“8”, “R”)
      - Memtable ➔immutable, flushed again
      - L0 beyond a threshold ➔ trigger compaction
    - Compaction Procedure
      - 1) select candidate (round robin, least overlapped, …)
      - 2) read overlapped SSTables from current and next level (L0 and L1)
      - 3) do merge sort (remove obsolete date)
      - 4) write new SSTables to the next level
      - Note ➔ L0: can be overlapped, L1~Lmax: no overlapped among SSTables

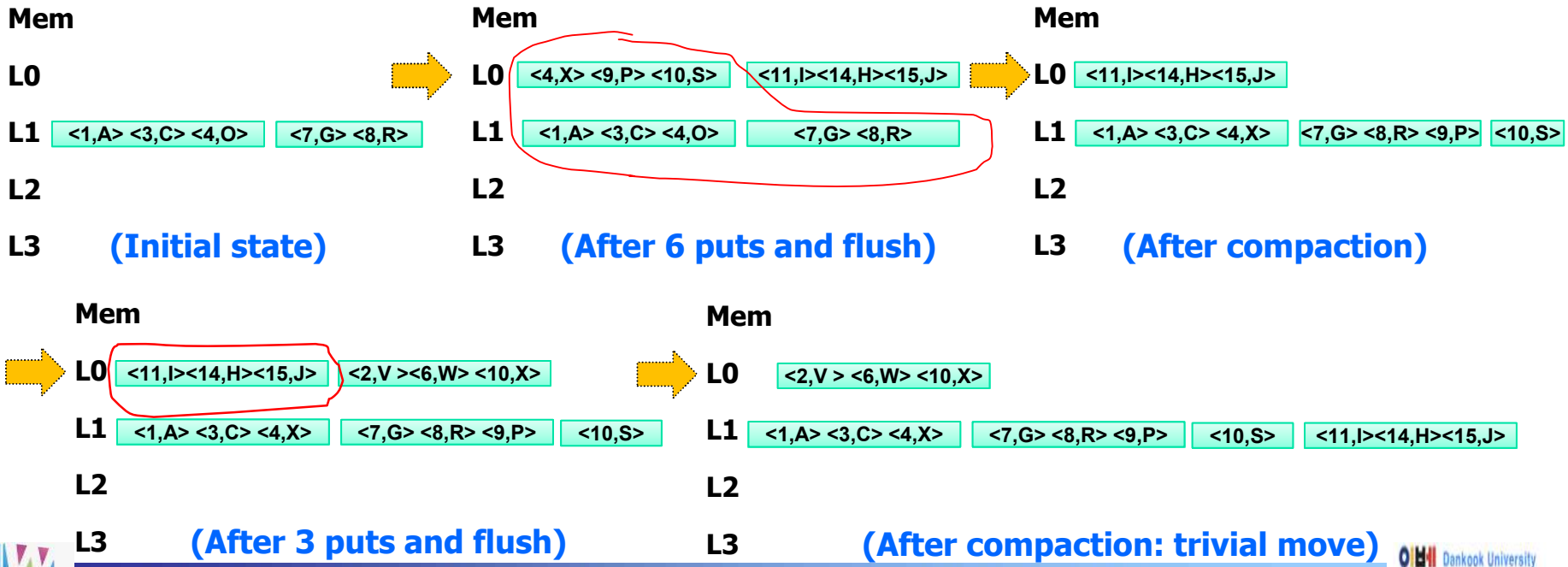


(Source: rocksdb/wiki)



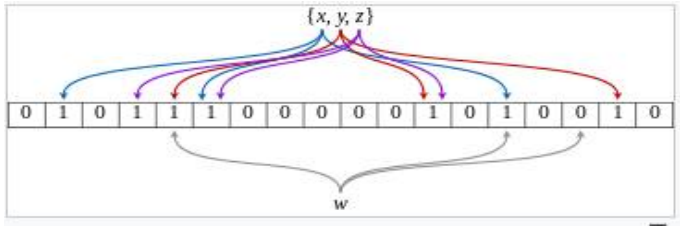
RocksDB Basic (11/15)

- Key-Value Interface
  - ✓ Example 2: compaction (cont')
    - Workload
      - put("10", "S"), put("9", "P"), put("4", "X"),
        - Memtable: managed by Skiplist
      - put("15", "J"), put("11", "I"), put("14", "H") ➔ compaction
      - put("2", "V"), put("6", "W"), put("10", "X") ➔ compaction
        - Candidate selection affect performance
        - Can trigger compaction recursively (background in actuality)

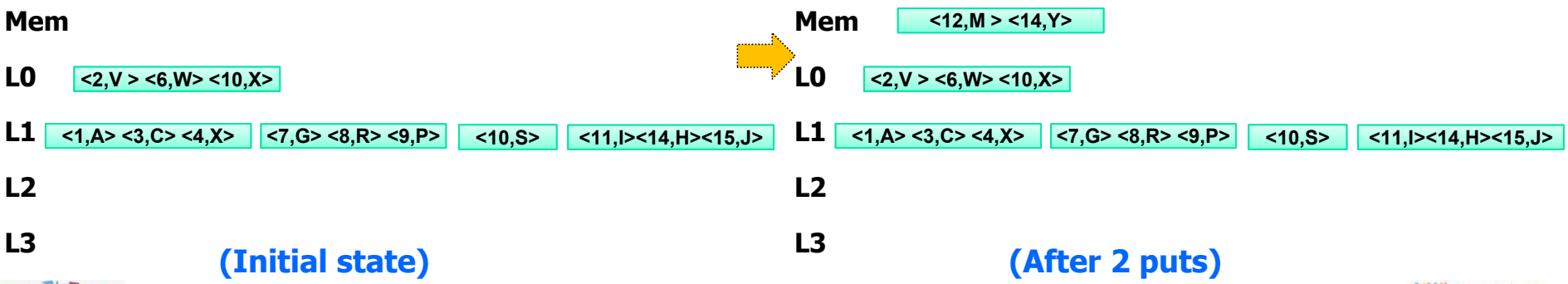


RocksDB Basic (12/15)

- Key-Value Interface (cont')
  - ✓ Example 3: get with Bloom filter
    - Workload
      - put("12", "M"), put("14", "Y")
      - get(12) ➔ find at Memtable
      - get(10) ➔ Both L0 and L1 (select higher level)
      - get(8) ➔ at L1 (but unnecessary read at L0)
    - Lookup procedure: Memtable first, then SSTables
      - Memtable: Skiplist
      - SSTable: range check from L0 to the next level until find the request key
      - May do unnecessary read (read amplification) ➔ Employ Bloom filter
        - test whether an element is a member of a set, No false negative

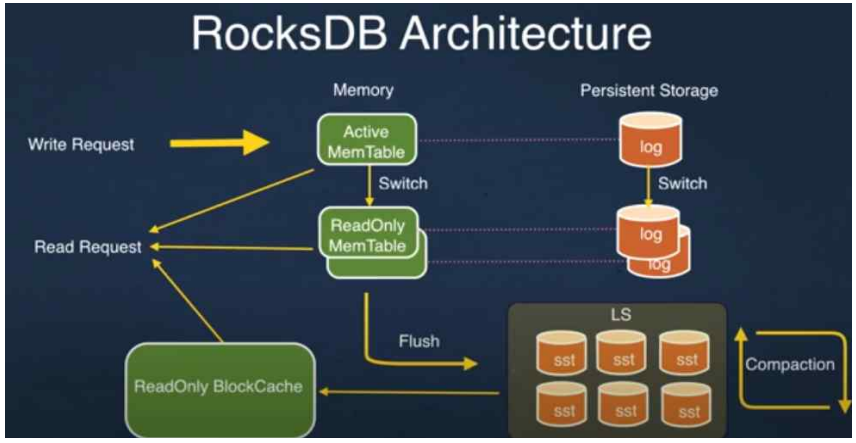


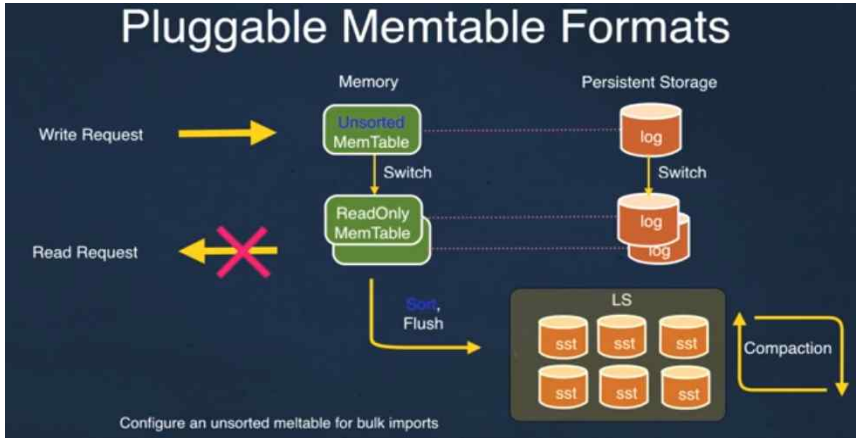
(Source: Wikipedia)



RocksDB Basic (13/15)

- Features for Performance enhancement
  - ✓ Various Compactions: Leveled, Universal (Tiered), FIFO, …
  - ✓ Multi-threaded & Background threads, Compaction filter (e.g. TTL)
  - ✓ Bloom filter, Block cache (LRU, Clock, ... ➔ pluggable)
  - ✓ Diverse Memtable/SSTable format
    - Memtable format: Skiplist, Unsorting (for bulk loading, vectored)
    - SST format: block based vs plain based (less computing, for RAM)
  - ✓ New functionalities
    - Column Family (CF), Merge, Transaction, Checksum, Prefix scan, Compression, …





(Source: Dhruba Borthakur, www.youtube.com/watch?v=hHsxqkcZy7c)

RocksDB Basic (14/15)

- History and Recent research
  - ✓ Development history: research shift
    - 1) Write amplification: Leveled ➔ Tiered compaction
    - 2) Space amplification: Compression, Compaction frequency (Especially important as shown in Table 2)
    - 3) CPU utilization: Prefix Bloom filter, Hash index
  - ✓ Recent focus
    - 1) Distributed system: resource allocation across multiple instances, replication, load balancing
    - 2) Data corruption: need to be detected earlier at every layer
    - 3) Compatibility: for incremental SW upgrade

	CPU	Space Util	Flash Endurance	Read Bandwidth
Stream Processing	11%	48%	16%	1.6%
Logging / Queues	46%	45%	7%	1.0%
Index Services	47%	61%	5%	10.0%
Cache	3%	78%	74%	3.5%

Table 2: System metrics for a typical use case from each application category.

A RocksDB Feature Timeline

	Performance	Configurability	Features
2012	• Multi-threaded compactions		• Compaction filters • Locking SSTables from deletion
2103	• Tiered compaction • Prefix Bloom filter • Bloom Filter for MemTables • Separate thread pool for MemTable flush	• Pluggable MemTable • Pluggable file format	• Merge Operator
2014	• FIFO compaction • Compaction rate limiter • Cache-friendly Bloom filters	• String-based config options • Dynamic config changes	• Backup engine • Support for multiple key spaces ("column family") • Physical checkpoints
2015	• Dynamic leveled compaction • File deletion rate limiting • Parallel Level 0 and 1 compaction	• Separate config file • Config compatibility checker	• Bulk loading for SSTable file integration • Optimistic and pessimistic transactions
2016	• Different compression for last level • Parallel MemTable inserts	• MemTable total size caps across instances • Compaction migration tools	• DeleteRange()
2017	• Separate thread pool for bottom-most compactions • Two-level file indices • Level 0 to level 0 compactions	• Single memory limit for both block cache and MemTable	
2018	• Dictionary compression • Hash index into data blocks		• Automatic recovery from out-of-space errors • Query trace and replay tools
2019	• Batched MultiGet() with parallel I/O	• Configure plug-in function using object registry	• Secondary instance
2020	• Multithreaded single file compression		• Entire file checksum • Automatically recover from retriable errors • Partial support for user-defined timestamps

(Source: Facebook RocksDB papers, FAST'21)

RocksDB Basic (15/15)

- Take-away Lessons
  - ✓ Understand the features of RocksDB
    - 1) Embedded, 2) LSM-tree, 3) Key-Value interface
  - ✓ LSM (Log-Structured Merge)-tree
    - 1) Log-structured ➔ Out-of-place update
      - Write-optimized, need compaction
    - 2) Merge
      - Non overlapped SSTables ➔ positive for read
    - 3) Tree
      - Maximum Obsolete data can be bounded
  - ✓ Issues
    - How to mitigate the compaction overhead? Advance topic 1
    - How to enhance read? Advance topic 2
    - How to support consistency, atomicity, integrity and durability? Advance topic 3
    - How to exploit new storage technology? Advance topic 4

(Source: www.dreamstime.com)

(Special thank you to DKU Embedded Members and IITP's SW StarLab (No.2021-0-01475)

Content

- What is Key-Value Store?
- RocksDB Basic
- Advance Topic 1: Compaction
  - ✓ Issue of compaction: Cause latency spike
  - ✓ Compaction styles: Leveled vs Tiered (Universal)
  - ✓ How to reduce compaction overhead?
- Advance Topic 2: Lookup
- Advance Topic 3: WAL
- Advance Topic 4: New approach
- RocksDB Practice
- Conclusion

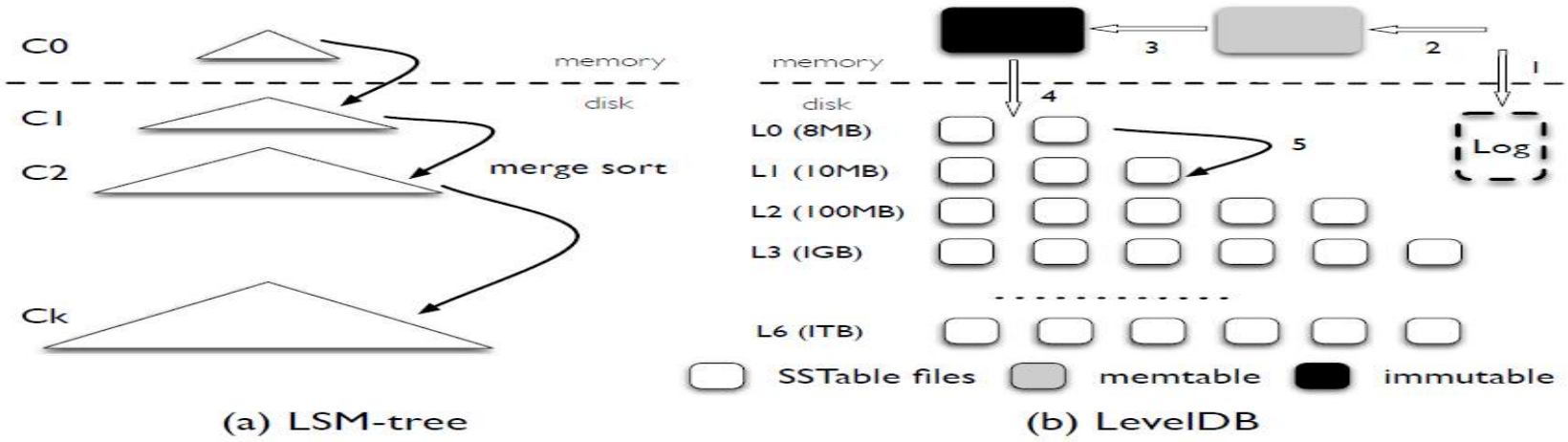
Advance Topic 1: Compaction (1/14)

- Lecture Objective
  - ✓ Understand the role of compaction
  - ✓ Learn about the amplification: write, space, and read
  - ✓ Examine how to reduce the compaction overhead: scheduling approach
  - ✓ Examine how to reduce the compaction overhead: algorithmic approach
  - ✓ Explore case studies

(Source: www.dreamstime.com)

Advance Topic 1: Compaction (2/14)

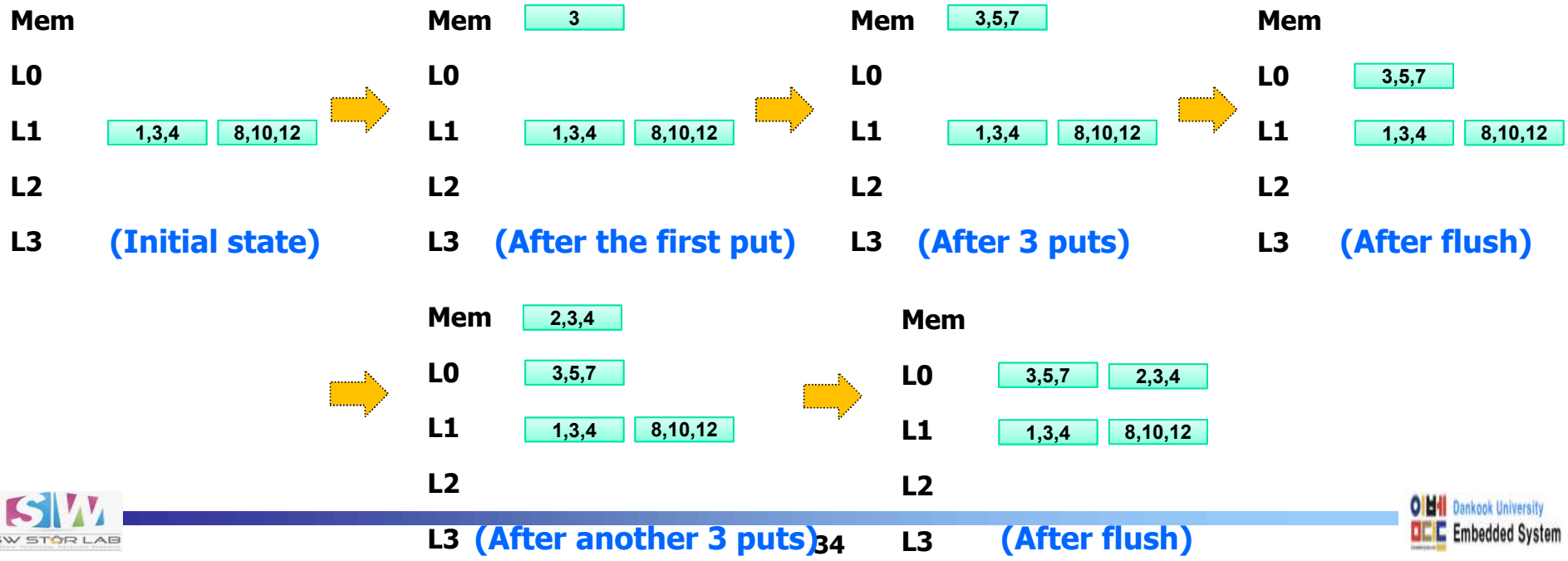
- What is the compaction?
  - ✓ RocksDB summary
    - Based on LSM (Log-Structured Merge)-tree
      - Layered: C0, C1, …, Ck (exponentially increasing)
    - Real implementation: Memtable, SSTable
      - Memtable in memory, SSTable in storage (multiple levels: L0, L1, …, Lk)
    - Interface: get, put, range scan, delete, …
    - put handling
      - 1. WAL, 2. Memtable, 3. immutable Memtable, 4. Flush, 5. Compaction



(Source: Wisckey paper, FAST'16)

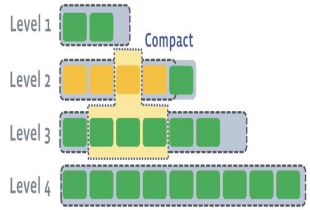
Advance Topic 1: Compaction (3/14)

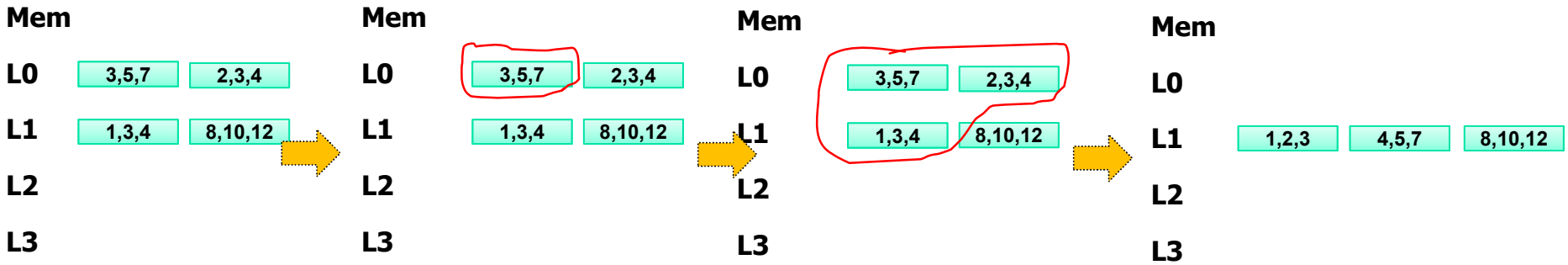
- What is the compaction?
  - ✓ Example
    - Assumption
      - 1) 3 KVs in a Memtable/SSTable, 2) L_0: 2 SSTables, $L_{i+1} = L_i * 2$
    - Workload
      - Initial state: 2 SSTables at L1 (show key only here)
      - 6 puts: keys =“3”, “7”, “5”, “2”, “3”, “4”
        - Memtable full ➔ trigger flush, L_i full ➔ trigger compaction



Advance Topic 1: Compaction (4/14)

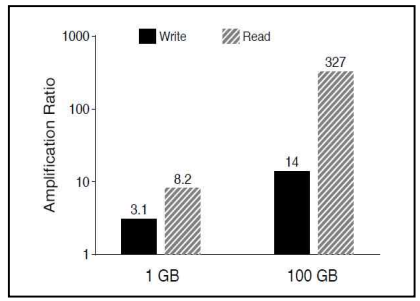
- What is the compaction?
  - ✓ Compaction procedure
    - 1. select candidate (FIFO, Least overlapped, …)
    - 2. read overlapped SSTables from current and next level
    - 3. do merge sort
    - 4. write new SSTables to the next level
  - ✓ Compaction effect
    - 1) Remove old data (reclaim), 2) Sort keys at L1, L2, … (fast lookup)
  - ✓ Compaction cost
    - Read/Write SSTables from/into Storage ➔ Heavy operation
    - Cause amplification



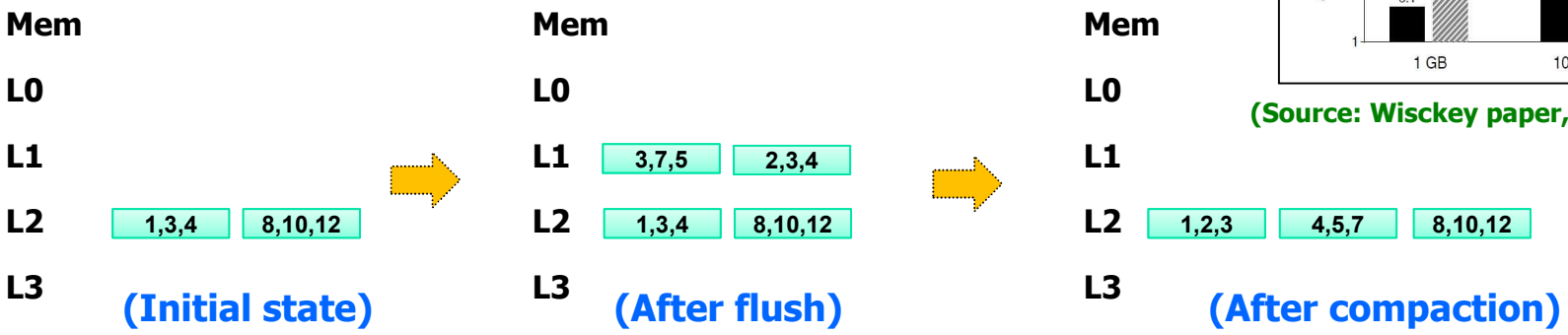


Advance Topic 1: Compaction (5/14)

- What is the amplification?
  - ✓ Amplification
    - An undesirable phenomenon where the actual amount of operations/space are more than intended
  - ✓ Three types
    - WAF (Write Amplification Factor) = actual writes/intended writes
    - SAF (Space Amplification Factor) = actual used space/required space
    - RAF (Read Amplification Factor) = actual reads/intended reads
  - ✓ Example
    - WAF at after compaction = 12/6
    - SAF at after compaction = 9/9, at after flush = 12/9

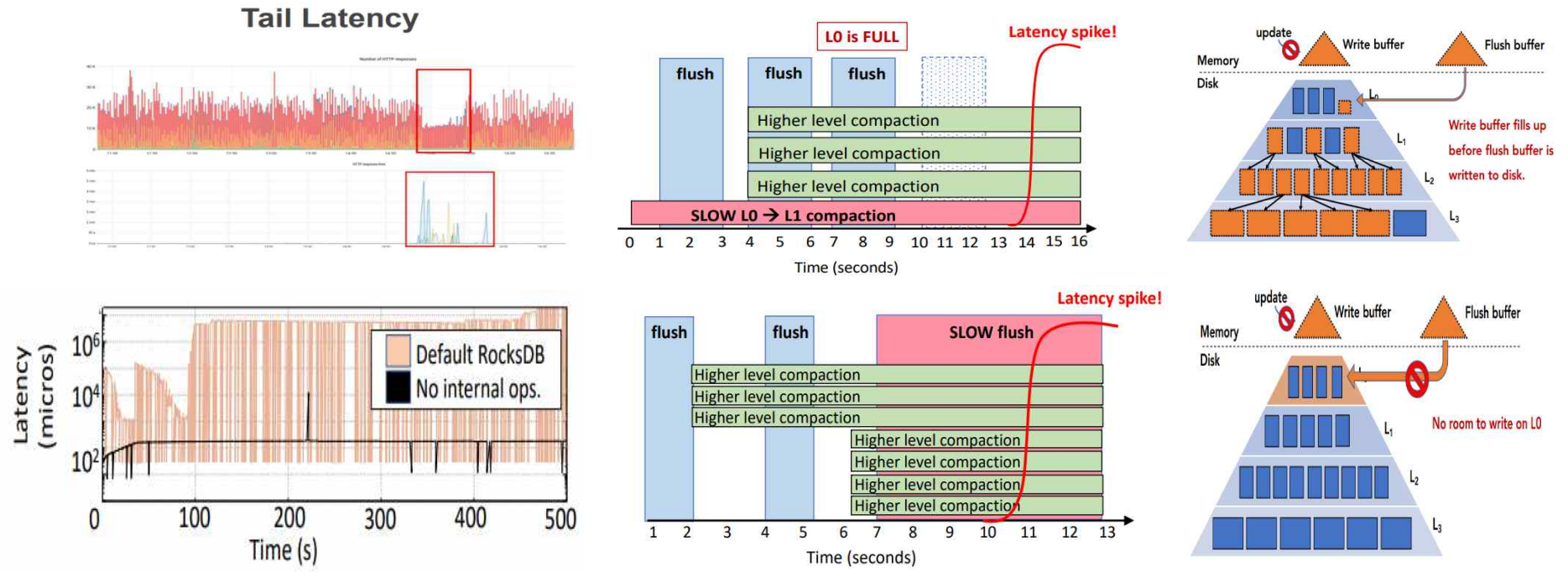


(Source: Wisckey paper, FAST'16)



Advance Topic 1: Compaction (6/14)

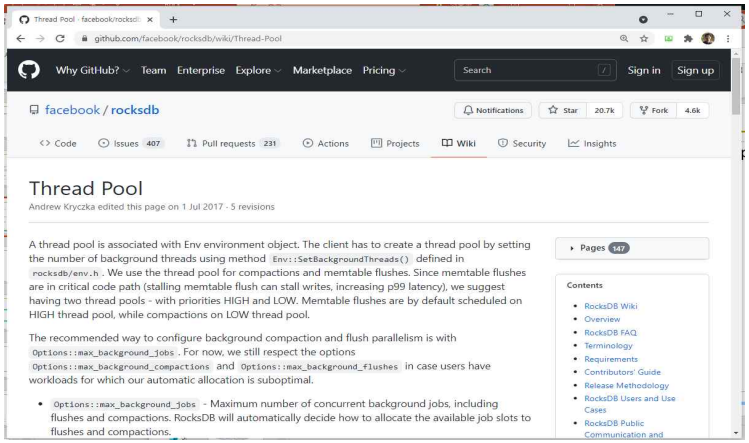
- Concerns about compaction
  - ✓ 1) Heavy operation (I/O, Sorting), 2) Amplification
  - ✓ 3) Delay user requests
    - Question from a developer (domestic company)
    - Unexpected latency Spike ➔ Work Graveyard (밤샘 근무)
    - Key-value DB: Compaction → Interfere Flush → Delay User Request

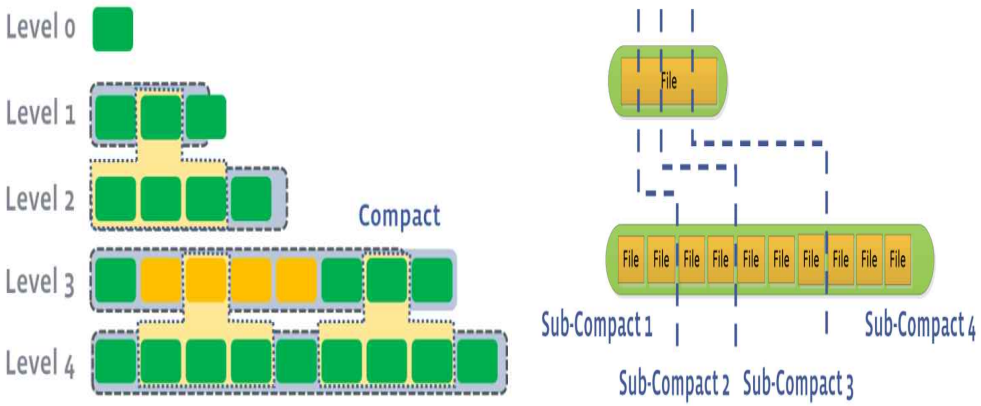


(Source: SILK papers, ATC'19)

Advance Topic 1: Compaction (7/14)

- How to reduce compaction overhead: scheduling approach
  - ✓ Thread Pool
    - max_background_jobs: # of threads for Flush and Compaction (default:2)
      - max_background_compactions and max_background_flushes, separately
  - ✓ Background and parallel compaction
    - Multiple compactions: controlled by the max_background_compactions
    - Sub compaction: To speed up a compaction job by partitioning it among multiple threads (only for L0 and L1)
  - ✓ Priority control
    - HIGH for flush threads, LOW for compaction threads

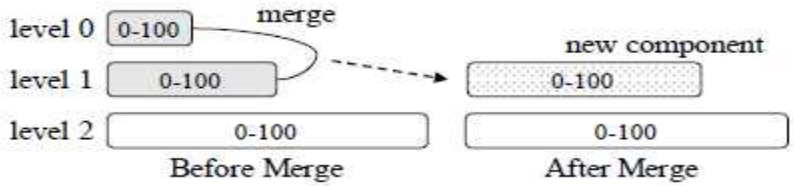




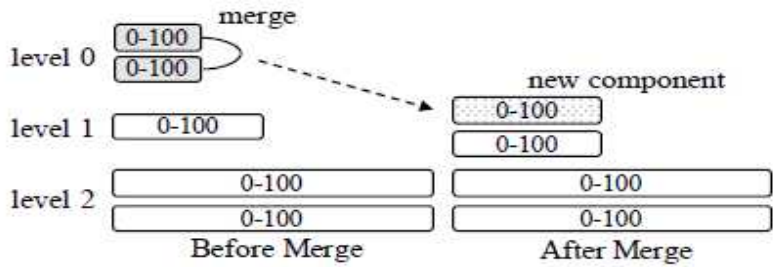
(Source: rocksdb/wiki and https://meeeejin.gitbooks.io/rocksdb-wiki-kr/content/leveled-compaction.html)

Advance Topic 1: Compaction (8/14)

- How to reduce compaction overhead: algorithmic approach
  - ✓ Leveled compaction
    - Level: consist of one run (component)
    - Merge: L_i and L_{i+1}
  - ✓ Tiered compaction (Universal compaction)
    - Level: consist of T runs (components)
      - Run: a set of SSTables generated during a time range
    - Merge: L_i only
  - ✓ Difference
    - Leveled: aggressively merge SSTables between levels
    - Tiered: waits for several sorted runs (lazy evaluation)
    - Tradeoff between WAF and SAF



(a) Leveling Merge Policy: one component per level



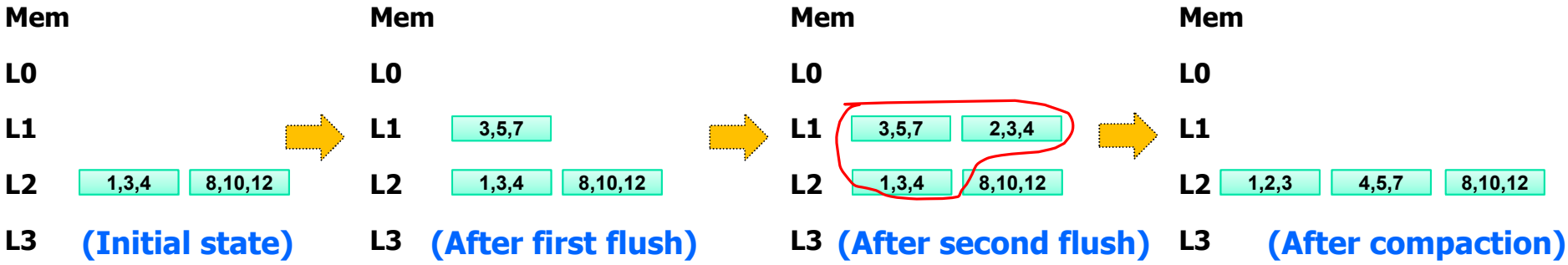
(b) Tiering Merge Policy: up to T components per level

(Source: LSM-based Storage Techniques: A Survey, VLDB Journal'20)

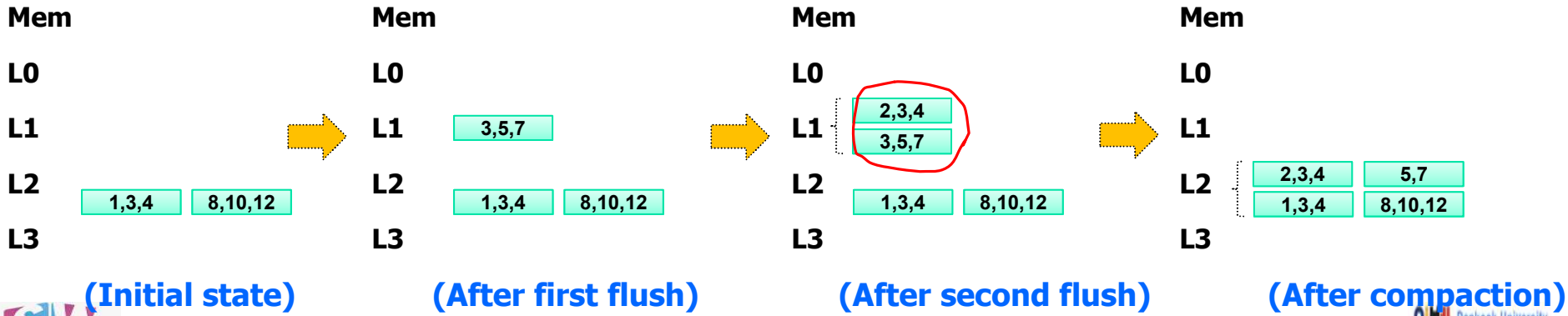
Advance Topic 1: Compaction (9/14)

- How to reduce compaction overhead: algorithmic approach
  - ✓ Example
    - Leveled compaction : WAF = 12/6, SAF = 9/9 (at after compaction)
    - Tiered compaction : WAF = 11/6, SAF = 11/9 (at after compaction)

Leveled compaction

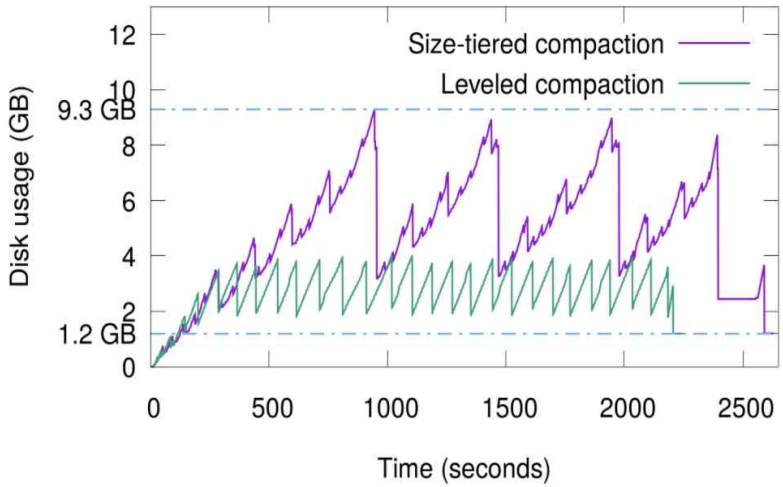


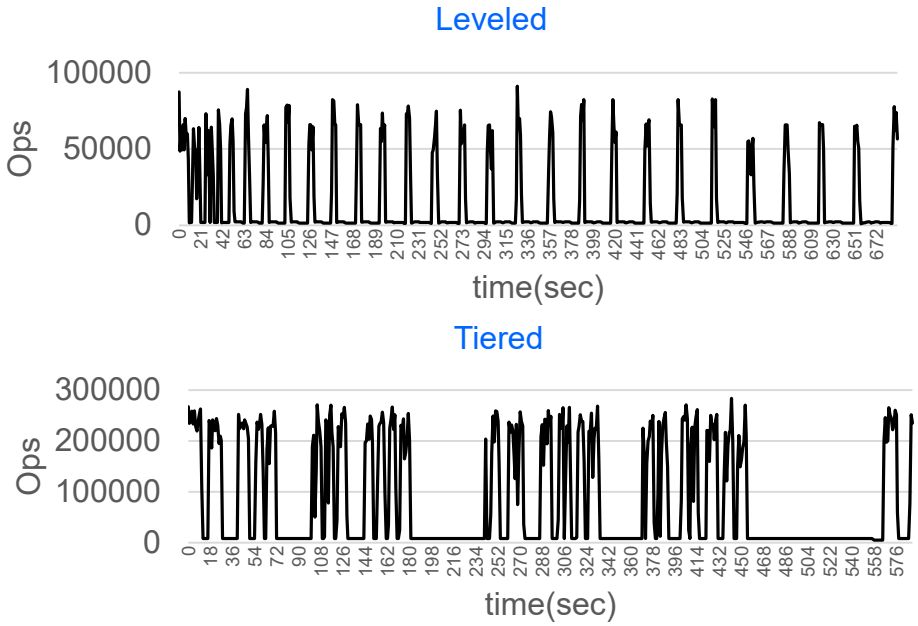
Tiered compaction



Advance Topic 1: Compaction (10/14)

- How to reduce compaction overhead: algorithmic approach
  - ✓ Comparison:
    - Leveled compaction : Good for SAF, Bad for WAF
    - Tiered compaction : Good for WAF, Bad for SAF
      - Guarantee write once at a level (c.f. it can be multiples in the Leveled compaction)
      - Might cause intensified compactions due to recursive triggering (lazy evaluation)

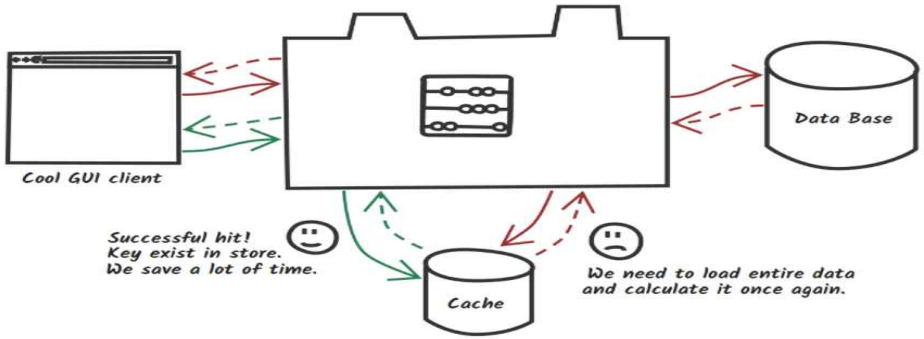


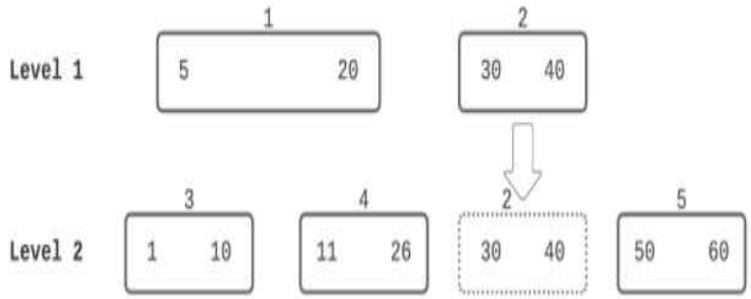


(Source: https://www.scylladb.com/2018/01/31/compaction-series-leveled-compaction/ and Our own experiment)

Advance Topic 1: Compaction (11/14)

- Other compaction features in RocksDB
  - ✓ FIFO compaction
    - L0 level only, delete SSTables based on TTL (Time-To-Live)
    - For Key-value cache (such as Redis, Memcached)
  - ✓ Candidate selection and trivial move
    - Select least overlapped SSTable
  - ✓ Hybrid scheme
    - Tiered + Leveled: tiered for the higher levels and leveled for others
      - Hot data ➔ WAF, Cold data ➔ SAF
    - Leveled-N: Leveled (merge L_i and L_{i+1}, but can be N-runs at a level)

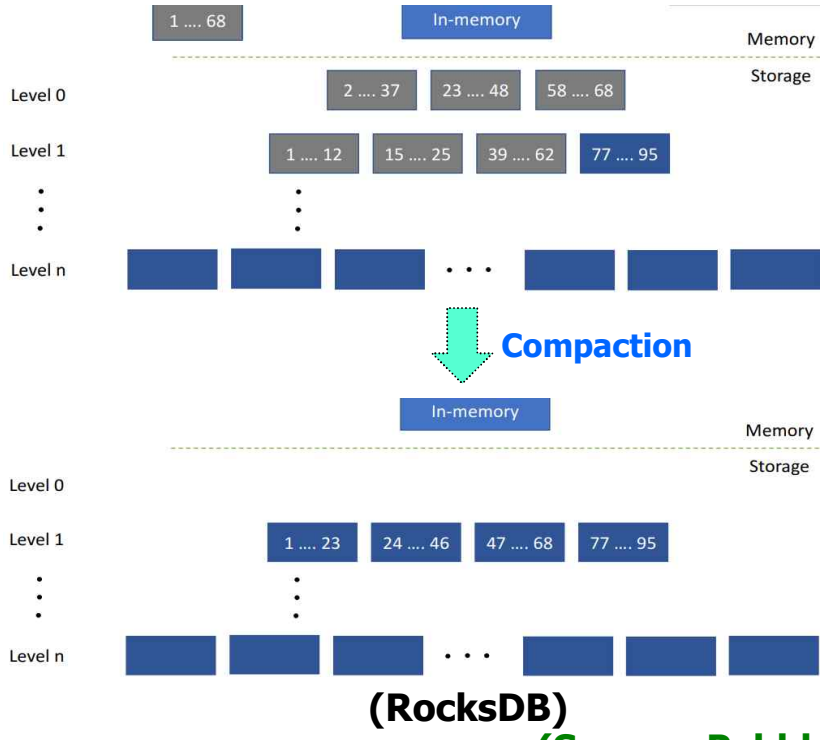




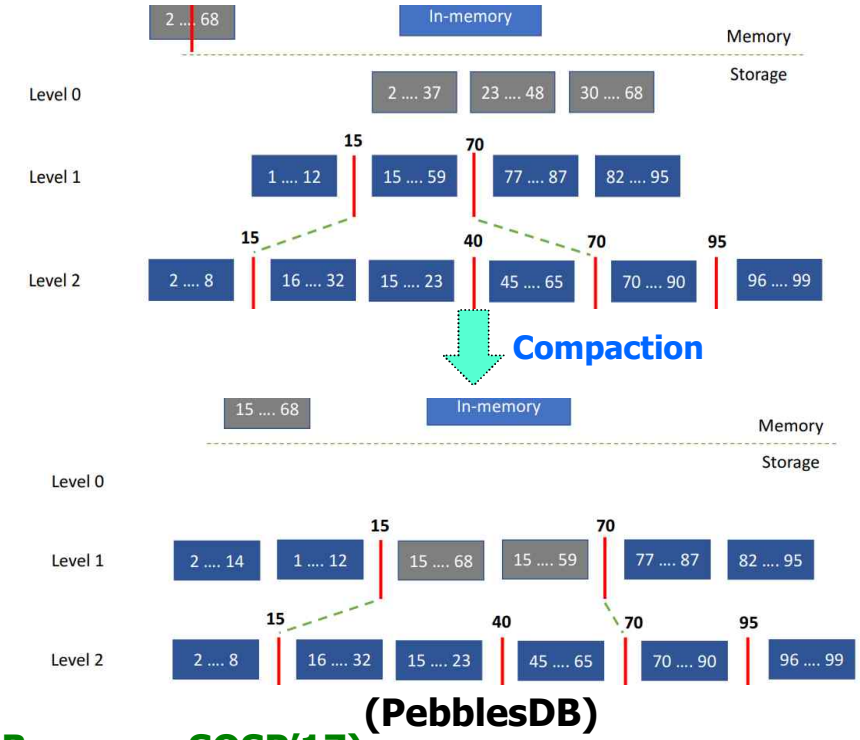
(Source: https://medium.com/@MatthewFTech/spring-boot-cache-with-redis-56026f7da83a/ and rocksdb/wiki)

Advance Topic 1: Compaction (12/14)

- Case study: PebblesDB, SOSP'17
  - ✓ Goal: Reduce compaction overhead
  - ✓ How to?
    - allow overlapping in a level with guard ➔ No need to read L_{i+1}
    - Write once per level vs. May deteriorate read latency



(RocksDB)

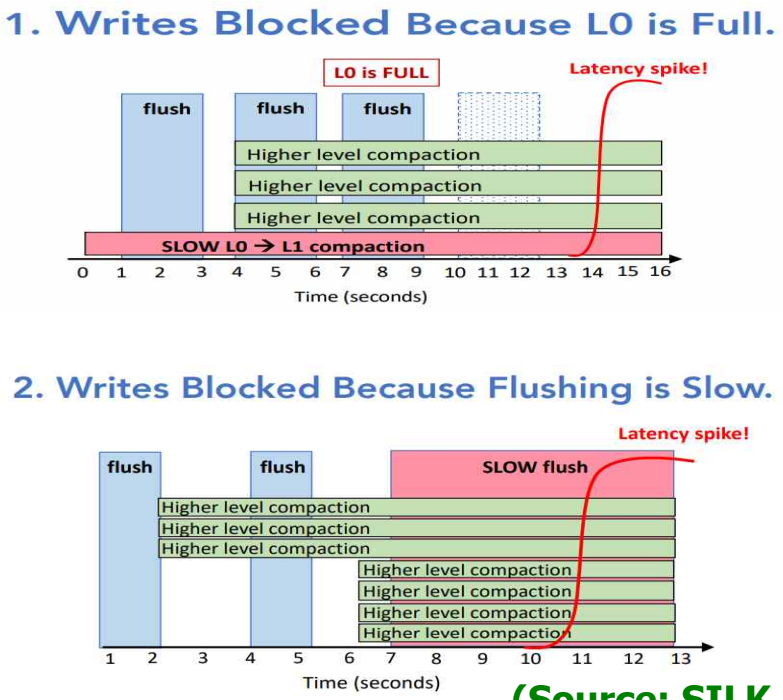


(PebblesDB)

(Source: PebblesDB papers, SOSP'17)

Advance Topic 1: Compaction (13/14)

- Case study: SILK, ATC'19
  - ✓ Goal: Hide compaction overhead
  - ✓ How to?
    - I/O scheduling: coordinate I/O bandwidth sharing to minimize interference between internal ops and client ops.
    - Motivate to design the scheduling approach in RocksDB (vice versa)



Lessons Learned	SILK Design
Make sure L0 is never full.	Prioritize internal operations at lower levels of the tree.
Ensure sufficient I/O for flush/ compactions on low levels.	Preempt higher level compactions if necessary.
Make sure other compactions do not fall behind too much.	Opportunistically allocate I/O for higher level compactions.

(Source: SILK papers, ATC'19)

Advance Topic 1: Compaction (14/14)

- Take-away Lessons
  - ✓ Role of compaction
    - To remove old data generated by updates in a Log-structured style
    - Heavy operation and cause latency spike
  - ✓ Three Compaction styles
    - Leveled, Tired, FIFO
    - Various influence in write, space and read amplification
  - ✓ Explore case studies
    - PebblesDB: algorithmic approach
    - SILK: scheduling approach
  - ✓ Future research
    - Hybrid: Leveled + Tiered
    - Make use of new storage (NVM, Fast SSD, …)

(Source: www.dreamstime.com)

(Special thank you to DKU Embedded Members and IITP's SW StarLab (No.2021-0-01475)

Content

- What is Key-Value Store?
- RocksDB Basic
- Advance Topic 1: Compaction
- Advance Topic 2: Lookup
  - ✓ Lookup in Memtable: Skiplist
  - ✓ Lookup in SSTable: Index block
  - ✓ How to enhance performance? Bloom filter
- Advance Topic 3: WAL
- Advance Topic 4: New approach
- RocksDB Practice
- Conclusion

Advance Topic 2: Lookup (1/12)

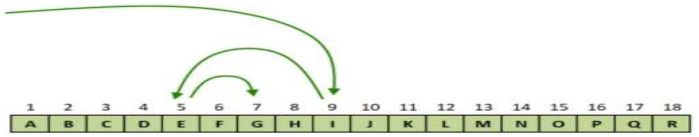
- Lecture Objective
  - ✓ Understand several data structures for lookup
  - ✓ Grasp the RocksDB approach for lookup
  - ✓ Learn about details of Memtable and SSTable
  - ✓ Investigate how Bloom filter can reduce the read amplification
  - ✓ Explore case studies

(Source: www.dreamstime.com)

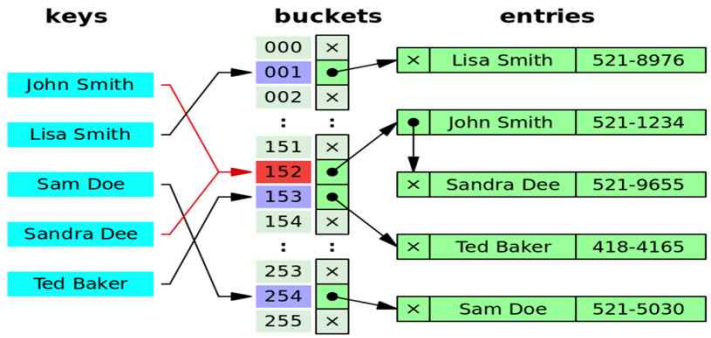
Advance Topic 2: Lookup (2/12)

- What is lookup (query)?
  - ✓ Two types
    - Point lookup (single query): get a value related to a given key
    - Range lookup (scan): get values related to a range of keys
    - Example: Find a number from an address, Calculate sum or average
  - ✓ How to: various data structures (e.g. array, list, sorted, hash, …)
    - 1) linear search: (*O(N)*), 2) binary search: (*O(Log N)*), 3) hash: (*O(1)*)
    - Tradeoffs: search speed, update overhead, scan overhead, …

key	value
H. Shin	010-1234-****
S. Kim	010-5678-****
J. Lee	010-1111-****
J. Ahn	010-2222-****
M. Oh	010-3333-****
G. Hong	010-4444-****
J. Choi	010-88**-****
⋮	
C. Kim	010-99**-****



(Binary search)

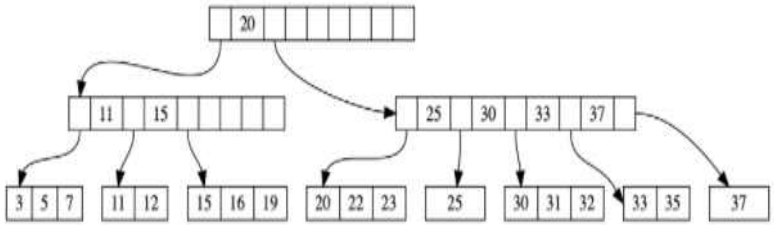


(Hash table)

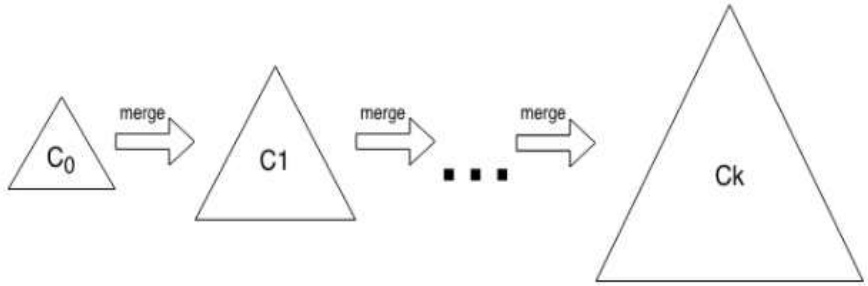
(Source: https://algorithms.tutorialhorizon.com/linear-search-vs-binary-search/ and mangkyu.tistory.com/102/)

Advance Topic 2: Lookup (3/12)

- How to materialize lookup in DB?
  - ✓ In traditional RDB (e.g. InnoDB)
    - Make use of B-tree (or B+-tree)
      - B-tree: generalized binary tree that has multiple children (n-ary tree)
      - B+-tree: all KV are stored in leaves, all leaves are linked for scan
    - Good for lookup, Bad for update due to tree reconstruction
  - ✓ RocksDB
    - Make use of LSM-tree, a write-optimized data structure
      - Employ B+-tree for lookup may deteriorate its original merit (possibly can be used as secondary index)
    - Utilize its own data structures for lookup purpose including Skiplist, Index block and Bloom filter



(B+-tree)

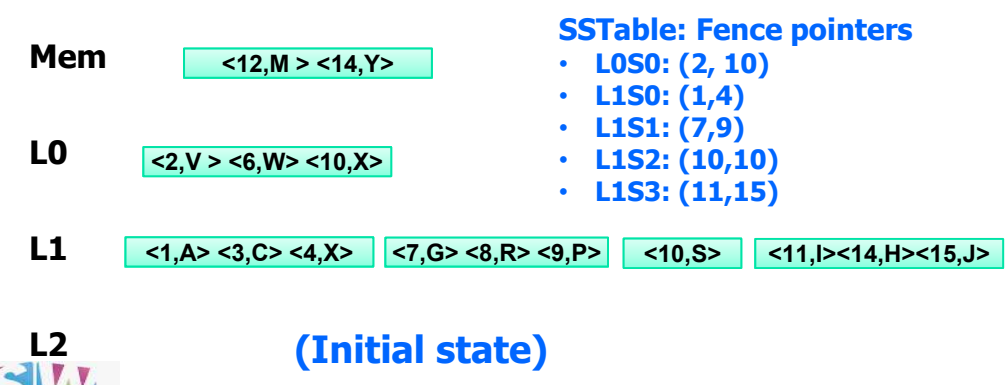


(LSM-tree)

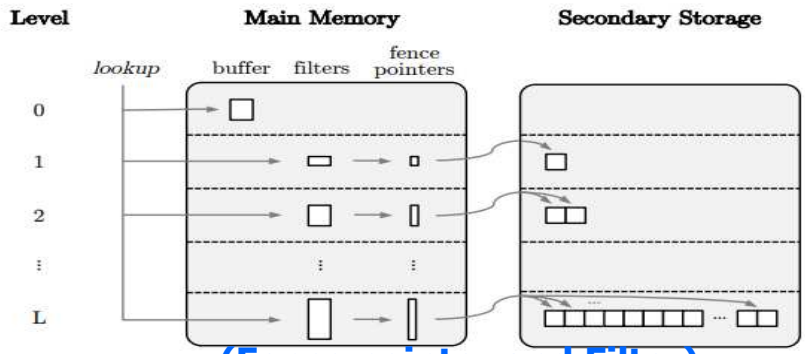
(Source: https://tikv.org/deep-dive/key-value-engine/b-tree-vs-lsm/)

Advance Topic 2: Lookup (4/12)

- RocksDB lookup: Overview
  - ✓ Lookup procedure: 1) Memtable, then 2) SSTables (from L_0 to L_{max})
  - ✓ Revisit the example discussed in the RocksDB basic section
    - get(12): find at Memtable using Skiplist
    - get(10): not in Memtable, need to find SSTables
      - Issue 1: many SSTables ➔ using Fence pointer in memory (min and max keys in SSTable)
      - Issue 2: Both L0 and L1 ➔ select higher level since it contains recent data
      - Issue 3: SSTable is large (default 64MB) ➔ using Index for fast retrieval
    - get(8): find in L1 only
      - Issue 4: Fence pointers suggest both L0 and L1 ➔ read L0 but there is no 8 (read amplification) ➔ using Bloom filter to reduce it



(Initial state)

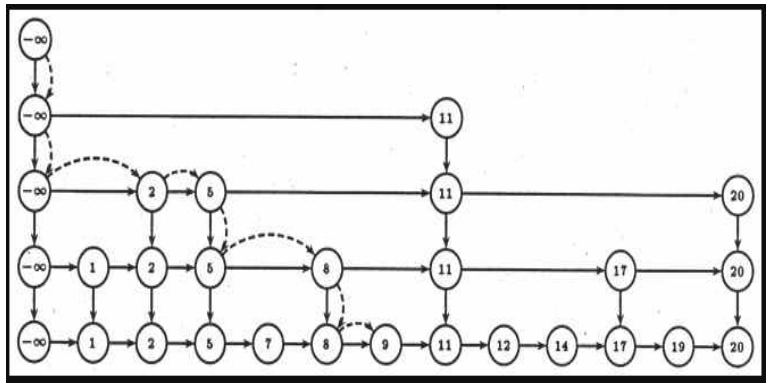


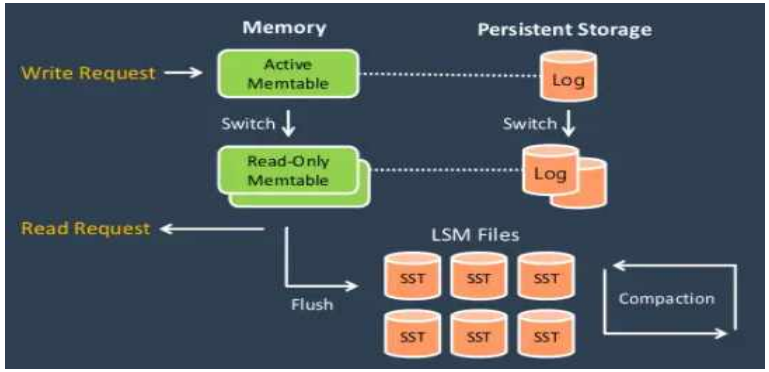
(Fence pointer and Filter)

(Source: http://daslab.seas.harvard.edu/classes/cs265/project.html/)

Advance Topic 2: Lookup (5/12)

- RocksDB detail: Memtable
  - ✓ KV pairs in memory, managed by Skiplist (or Hashlist)
    - Skiplist: a data structure with a set of sorted linked lists
      - All keys appears in the last list
      - Some keys also appear in the upper list (for fast search)
    - Good for both lookup and scan (get benefit both binary tree and list)
    - Useful in multithreaded system architectures
  - ✓ Size options
    - write_buffer_size (Memtable size): default 64MB (configurable)
    - max_write_buffer_number: default 2 (also configurable)





(Source: www.osa.fu-berlin.de/bioinformatics_msc/en/exemplary_tasks/informatics_algorithms/index.html and Dhruba Borthakur, www.youtube.com/watch?v=hHsxqkcZy7c)

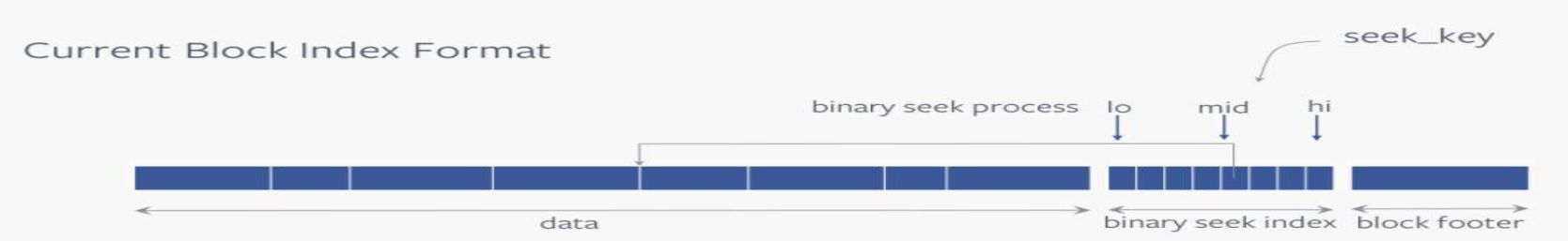
Advance Topic 2: Lookup (6/12)

- RocksDB detail: SSTable
  - ✓ KV pairs in storage, managed as a file
  - ✓ Issue: SSTable is large (default 64 MB)
    - Assume 1KB KV ➔ 64,000 pairs in a file
    - A file is divided into multiple disk blocks
  - ✓ Solution: well-defined SSTable format
    - 1) SSTable is divided into data blocks
    - 2) Each KV is searched using index (B-search)
    - 3) filter block
    - 4) other meta blocks (e.g. compression)
    - 5) metaindex: one entry for every meta blocks
    - 6) footer: pointers for metaindex & index blocks

File format

```
<beginning_of_file>
[data block 1]
[data block 2]
...
[data block N]
[meta block 1: filter block]
[meta block 2: index block]
[meta block 3: compression dictionary block]
[meta block 4: range deletion block]
[meta block 5: stats block]
...
[meta block K: future extended block]  (we m
[metaindex block]
[Footer]                               (fixe
<end_of_file>
```

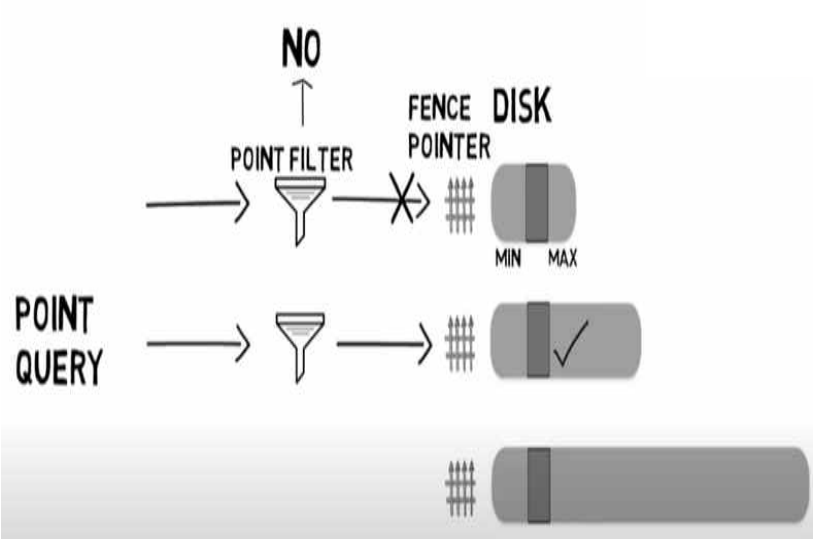
(Source: github.com/facebook/rocksdb/wiki/Rocksdb-BlockBasedTable-Format)

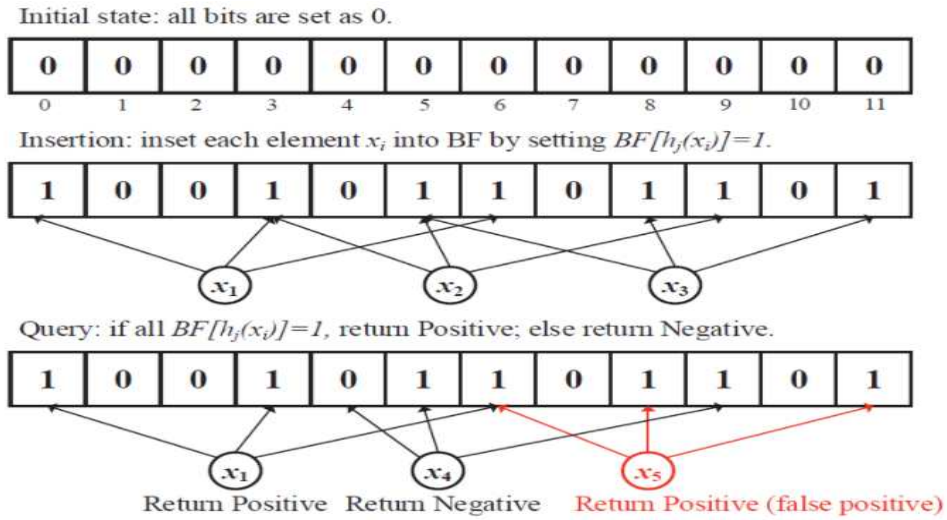


(Source: https://rocksdb.org/blog/2018/08/23/data-block-hash-index.html)

Advance Topic 2: Lookup (7/12)

- RocksDB detail: Bloom filter
  - ✓ Used to reduce the read amplification (unnecessary read)
  - ✓ Bloom filter: a data structure for identifying membership
    - Based on bits and multiple hashes
    - Good property: No false negative
    - Issue: can yield false positive ➔ tradeoffs between bits and rate (1% false positive rate with 9.9 bits per key, from RocksDB wiki)
  - ✓ Not only SSTable (per SSTable or per block) but also Memtable

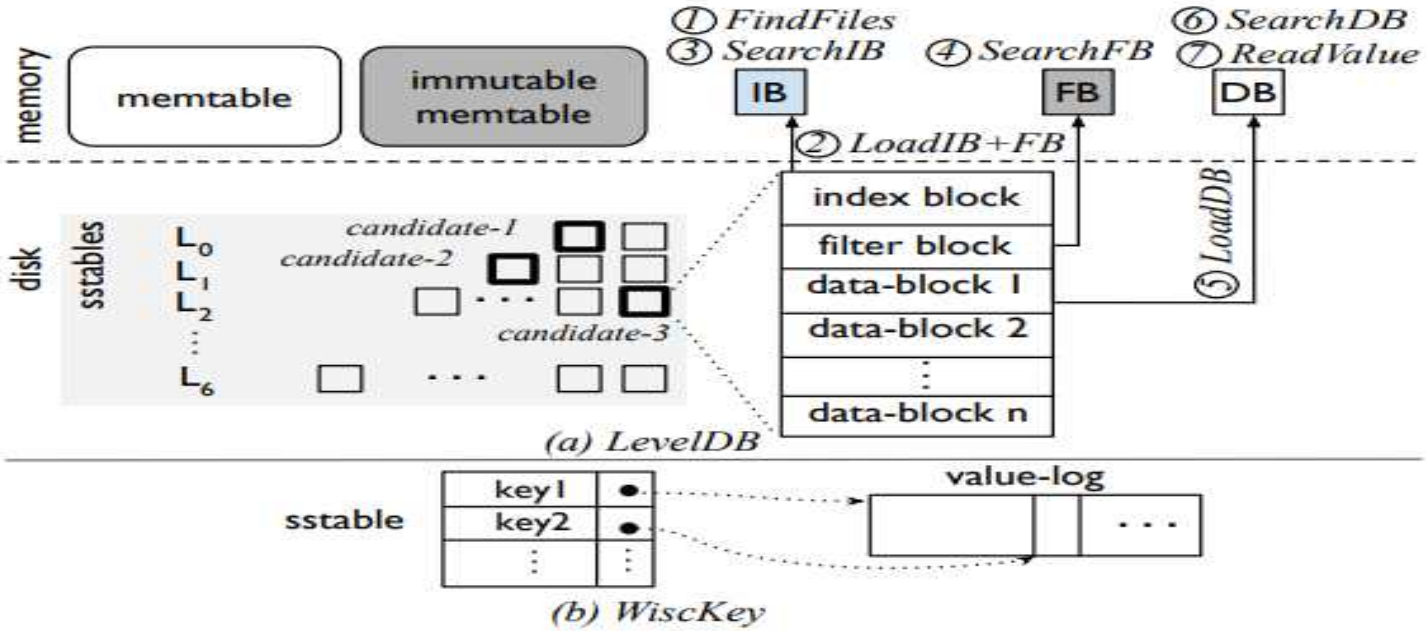




(Source: Rosetta paper, SIGMOD'20 and https://devopedia.org/bloom-filter)

Advance Topic 2: Lookup (8/12)

- RocksDB detail: SSTable and Bloom filter together
  - ✓ 1) Find candidates using fence pointers
  - ✓ 2) Load IB (Index Block) and FB (Filter Block)
  - ✓ 3, 4) Search IB and FB
  - ✓ 5) Load DB (Data Block)
  - ✓ 6, 7) Search DB and read value



(Source: Bourbon paper, OSDI'20)

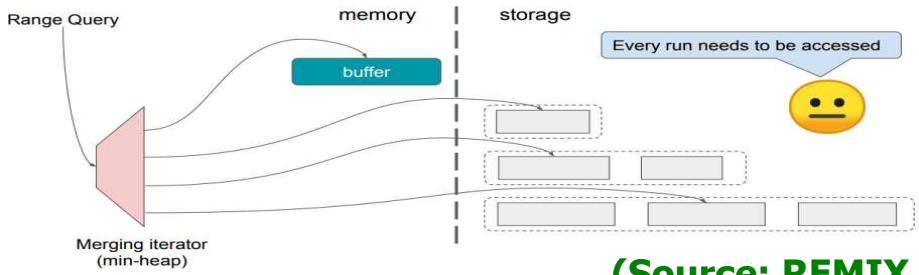
Advance Topic 2: Lookup (9/12)

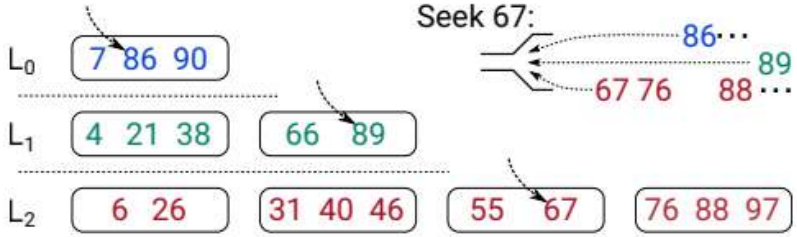
- RocksDB detail: Range query
  - ✓ Interface

(Source: rocksdb/wiki)

```
rocksdb::Iterator* it = db->NewIterator(rocksdb::ReadOptions());
for (it->Seek(start);
     it->Valid() && it->key().ToString() < limit;
     it->Next()) {
  ...
}
assert(it->status().ok()); // Check for any errors found during the scan
```

  - ✓ Internals
    - Range query: need to access all levels (c.f. point lookup: stop if found)
    - Using min-heap: example of scan(67, 95)
      - 1) find a key larger than seek for every levels (cursor)
      - 2) construct the min-heap using cursors (67, 89, 86)
      - 3) get the smallest key (67), 4) if it is larger than limit, stop, 5) print it, 6) find the next cursor (76), 7) insert it into the min-heap
      - Repeat 3) ~ 7) steps ➔ "67, 76, 86, 88, 89, 90, stop"
    - Prefix bloom filter is used to reduce unnecessary read during scan

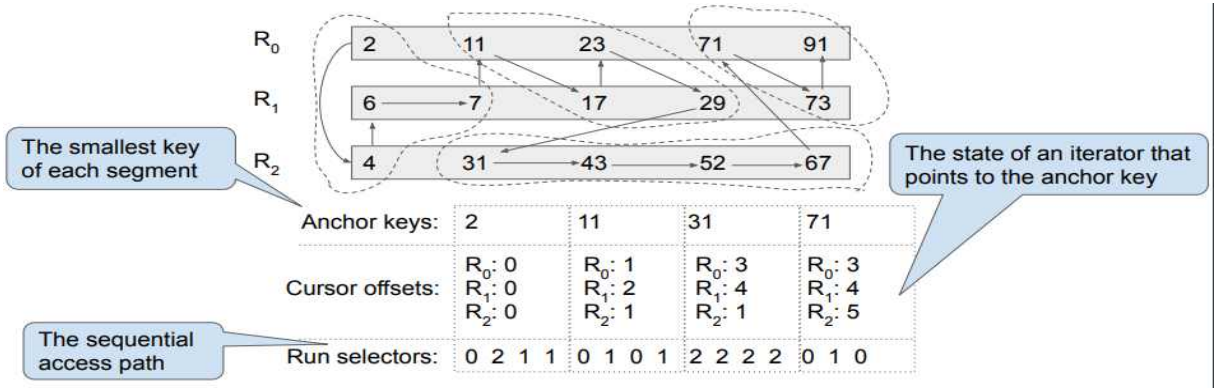




(Source: REMIX paper, FAST'21)

Advance Topic 2: Lookup (10/12)

- Case study: REMIX, FAST'21
  - ✓ Goal: Reduce range query overhead
  - ✓ How to?
    - Construct a new data structure that has a global sorted view among runs
      - 1) Anchor keys: the smallest key in each segment
      - 2) Cursor offsets: for identifying runs that has keys in each segment
      - 3) Run selector: for logical sorted view ➔ divide into segments
    - Comparison: example of scan (15, 50)
      - Traditional: min-heap with "23, 17, 31", do min-heap operations repeatedly
      - REMIX: anchor 11, using Run and Cursor for next key (e.g. R0/C1, R1/C2, R0/C2, R1/C3, …) ➔ no min-heap operations



(Source: REMIX papers, FAST'21)

Advance Topic 2: Lookup (11/12)

- Case study: Bourbon, OSDI'20
  - ✓ Goal: Reduce lookup overhead
  - ✓ How to? make use of machine learning (Learned index)
    - Idea: Index can be viewed as Learning
      - B-tree: 1) new key ➔ tree reconstruction, 2) lookup: key ➔ position
      - Model: 1) learning from "key -> position" data, 2) lookup: inference from key to expected position (not deterministic, probabilistic)
    - Bourbon: employ Learned index into Key-value store
      - Model lookup instead of Index block access
      - Suitable for Read intensive workloads

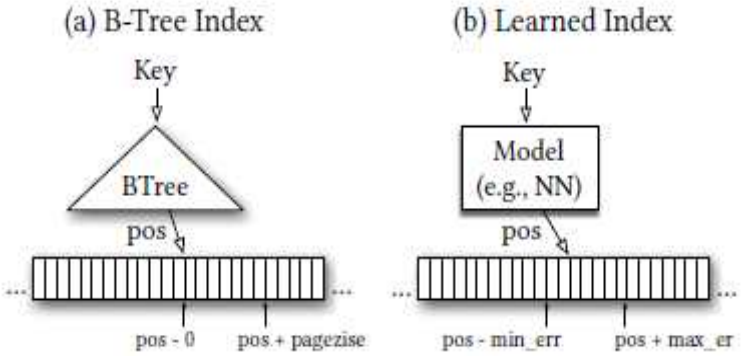
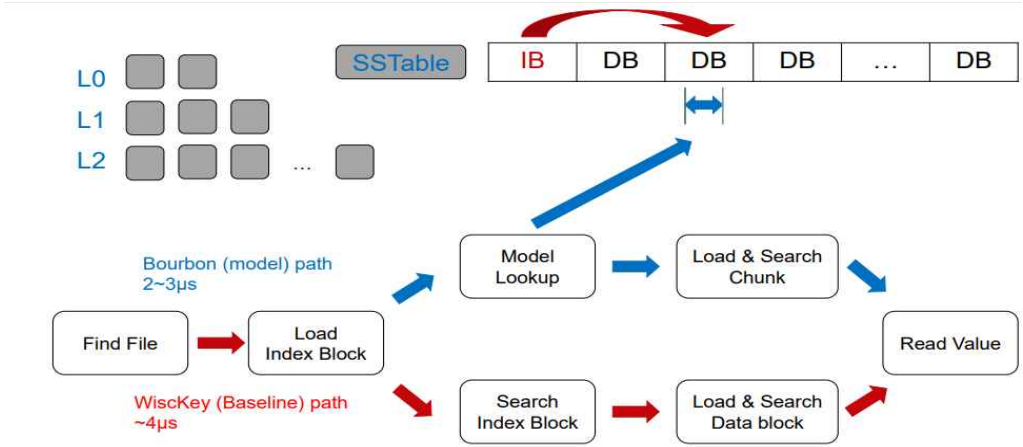


Figure 1: Why B-Trees are models



(Source: Learned Index, SIGMOD'18 and Bourbon papers, OSDI'21)

Advance Topic 2: Lookup (12/12)

- Take-away Lessons
  - ✓ Basic data structures for lookup
    - Linked-List, Binary tree, Hash, B+-tree, …
  - ✓ RocksDB approach for lookup
    - Memtable: Skiplist for both fast lookup and scan
    - SSTable: Index block and Bloom Filter
    - Range scan: min heap (priority queue)
  - ✓ Investigate how Bloom filter can reduce the read amplification
    - Tradeoff between Space/CPU overhead and false positive ratio
    - Active used for other purposes (e.g. Prefix Bloom Filter)
  - ✓ Explore case studies
    - REMIX: Build extra index for range query
    - Bourbon: Machine learning in Key-Value Store

(Source: www.dreamstime.com)

(Special thank you to DKU Embedded Members and IITP's SW StarLab (No.2021-0-01475)

Content

- What is Key-Value Store?
- RocksDB basic
- Advance Topic 1: Compaction
- Advance Topic 2: Lookup
- Advance Topic 3: WAL
  - ✓ Durability: WAL and Recovery
  - ✓ Consistency: atomicity and Transaction
  - ✓ Integrity: Checksum
- Advance Topic 4: New approach
- RocksDB Practice
- Conclusion

Advance Topic 3: WAL (1/11)

- Lecture Objective
  - ✓ Understand the requirements for DB Reliability
  - ✓ Find out the role of WAL (Write Ahead Log)
  - ✓ Learn about how to support transaction
  - ✓ Explore how to keep integrity
  - ✓ Examine the performance effect of WAL

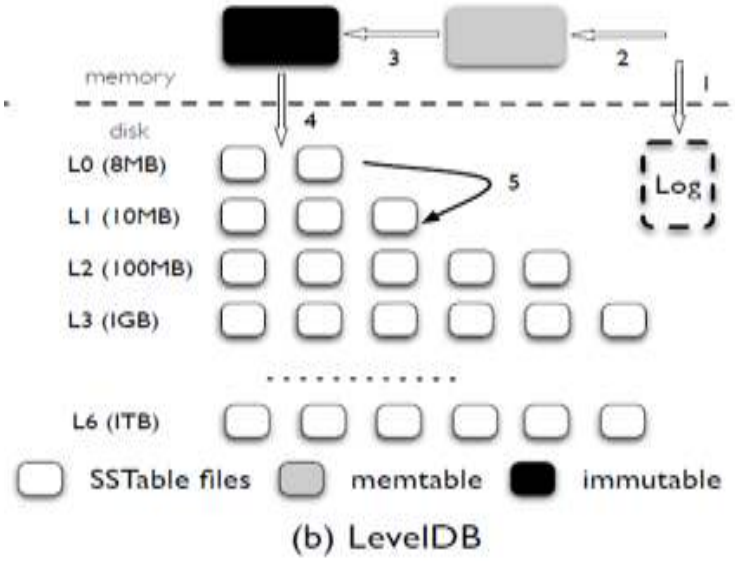
(Source: www.dreamstime.com)

Advance Topic 3: WAL (2/11)

- Performance and Reliability
  - ✓ Techniques such as Compaction and Bloom filter that we have learnt so far ➔ for performance enhancement
  - ✓ DB demands reliability (often degrade performance and vice versa)
- Requirements for high reliability
  - ✓ Durability: make data persistent
  - ✓ Atomicity: all or nothing
  - ✓ Consistency: stay in a valid state or agreement among parts
  - ✓ Integrity: assurance of data accuracy
  - ✓ Recovery: go back to a valid state

Advance Topic 3: WAL (3/11)

- Reliability matters
  - ✓ Put KV: first Memtable, and SSTable via flush
    - By the way, main memory is volatile ➔ What if power failure occurs?
    - Lost user data is a big risk in a DB company ➔ require durability
  - ✓ RocksDB approach
    - WAL (Write Ahead Log, also called as journal or intent log) ➔ in order to support durability and easy recovery



(b) LevelDB

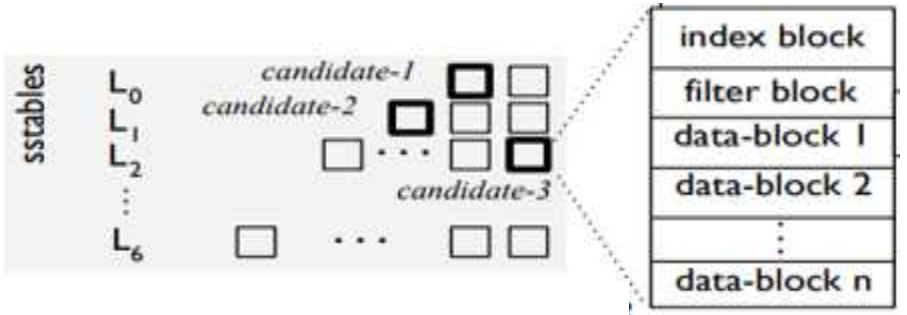
```
-rw-r--r-- 1 root root  37922501  7월  1 15:44 000214.sst
-rw-r--r-- 1 root root  37920200  7월  1 15:44 000216.sst
-rw-r--r-- 1 root root  37910828  7월  1 15:44 000219.sst
-rw-r--r-- 1 root root  37906740  7월  1 15:44 000221.sst
-rw-r--r-- 1 root root  37905482  7월  1 15:44 000224.sst
-rw-r--r-- 1 root root  37909294  7월  1 15:44 000227.sst
-rw-r--r-- 1 root root  66217640  7월  1 15:44 000228.log
-rw-r--r-- 1 root root  37892964  7월  1 15:44 000229.sst
-rw-r--r-- 1 root root  18621323  7월  1 15:44 000231.log
-rw-r--r-- 1 root root        16  7월  1 15:43 CURRENT
-rw-r--r-- 1 root root        37  7월  1 15:43 IDENTITY
-rw-r--r-- 1 root root         0  7월  1 15:43 LOCK
-rw-r--r-- 1 root root    572956  7월  1 15:44 LOG
-rw-r--r-- 1 root root     14592  7월  1 15:44 MANIFEST-000004
-rw-r--r-- 1 root root      6180  7월  1 15:43 OPTIONS-000007
```

(Source: Wisckey paper, FAST'16 and our experiment)

Advance Topic 3: WAL (4/11)

- Reliability matters
  - ✓ A set of operations are connected
    - Assume that the put() performs while the delete() does not ➔ duplicated
    - Assume that the delete() performs while the put() does not (various causes including scheduling, caching, …) ➔ Data loss
    - How to overcome? require atomicity
  - ✓ A set of blocks are connected
    - Assume that data is written while index is not ➔ Space leak
    - Assume that index is written while data is not ➔ Garbage read
    - How to overcome? require consistency
  - ✓ RocksDB approach
    - WriteBatch and Transaction

```
std::string value;
rocksdb::Status s = db->Get(rocksdb::ReadOptions(), key1, &value);
if (s.ok()) s = db->Put(rocksdb::WriteOptions(), key2, value);
if (s.ok()) s = db->Delete(rocksdb::WriteOptions(), key1);
```



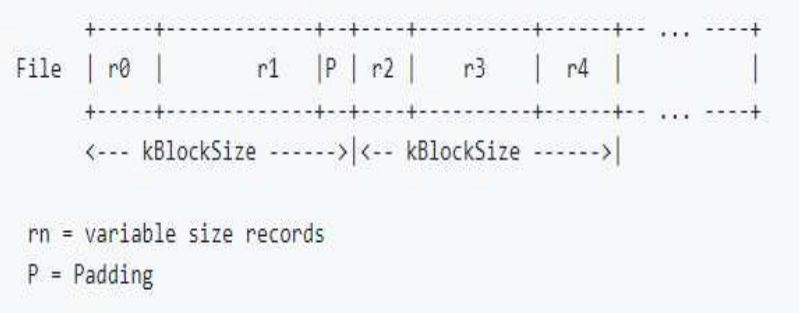
(Source: https://github.com/facebook/rocksdb/wiki/Basic-Operations and Bourbon paper, OSDI'20)

Advance Topic 3: WAL (5/11)

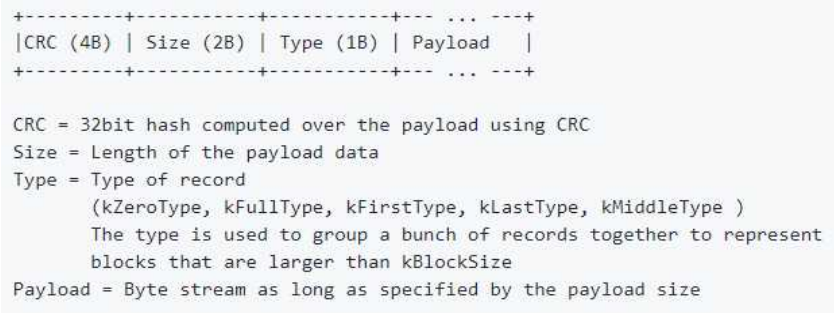
- RocksDB detail: WAL
  - ✓ What?
    - Logging every update

Put, key2, value	Delete, key1

  - ✓ How?
    - log file: consists of kBlocks (32KB default)
    - kBlock: a set of record (KV) and padding
      - Type: Full → KV in a record, First/Middle/Last → KV in multiple records
  - ✓ Lifecycle
    - A WAL file is created when a new DB is opened
    - A WAL is deleted (or archived if archival is enabled) when all data in the WAL have been persisted to SST files.



(Log file format)

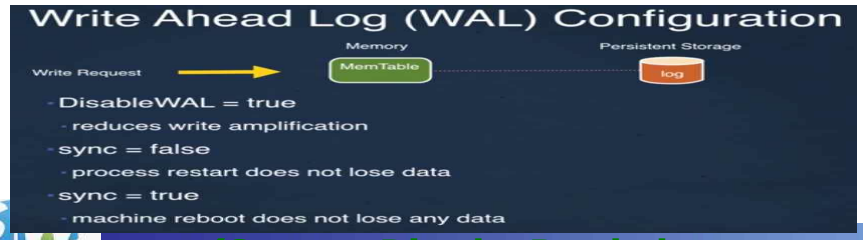


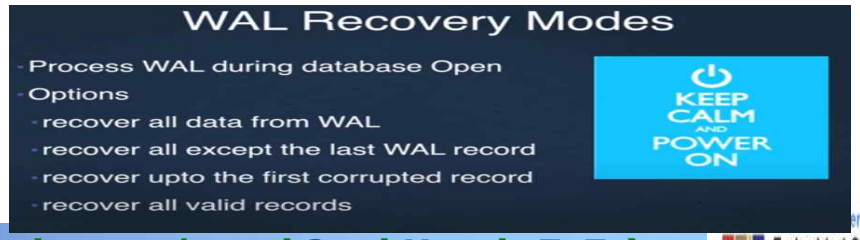
(Record format)

(Source: https://github.com/facebook/rocksdb/wiki/Write-Ahead-Log-File-Format)

Advance Topic 3: WAL (6/11)

- RocksDB detail: WAL
  - ✓ Configuration
    - DisableWAL, max_total_wal_size, wal_dir (for different storage), …
    - Sync mode
      - Non-sync mode (sync = false, default)
        - OS-level or manually sync (e.g. call SyncWAL() periodically)
        - Guarantees process crash consistency only
      - Sync mode (sync = true)
        - Do sync() before returning to user
        - Guarantees both process and system crash consistency, but heavy weight
  - ✓ Recovery
    - Redo WAL records
    - Policies: 1) try all (one corrupted record fails recovery), 2) try all except last WAL, 3) until first corrupted record (others from replica), 4) try your best and ignore corrupted records





(Source: Dhruba Borthakur, www.youtube.com/watch?v=hHsxqkcZy7c)

Advance Topic 3: WAL (7/11)

- RocksDB detail: Writebatch and Transaction
  - ✓ Writebatch
    - For atomicity (in addition for better performance)
  - ✓ Transaction
    - For atomicity, consistency (always valid state) and concurrency
    - Add transaction begin and commit into a log

Tr Begin	Put, key2, v2	Delete, key1	Tr Commit

    - Recovery
      - Committed ➔ redo, Not committed ➔ undo (rollback)
    - Pessimistic (default) vs Optimistic
      - Always locking for thread safe vs validating at commit time

```
#include "rocksdb/write_batch.h"
...
std::string value;
rocksdb::Status s = db->Get(rocksdb::ReadOptions(), key1, &value);
if (s.ok()) {
  rocksdb::WriteBatch batch;
  batch.Delete(key1);
  batch.Put(key2, value);
  s = db->Write(rocksdb::WriteOptions(), &batch);
}
```

(Writebatch)

```
TransactionDB* txn_db;
Status s = TransactionDB::Open(options, path, &txn_db);

Transaction* txn = txn_db->BeginTransaction(write_options, txn_options);
s = txn->Put("key", "value");
s = txn->Delete("key2");
s = txn->Merge("key3", "value");
s = txn->Commit();
delete txn;
```

(Transaction)

(Source: https://github.com/facebook/rocksdb/wiki/Basic-Operations and /transactions)

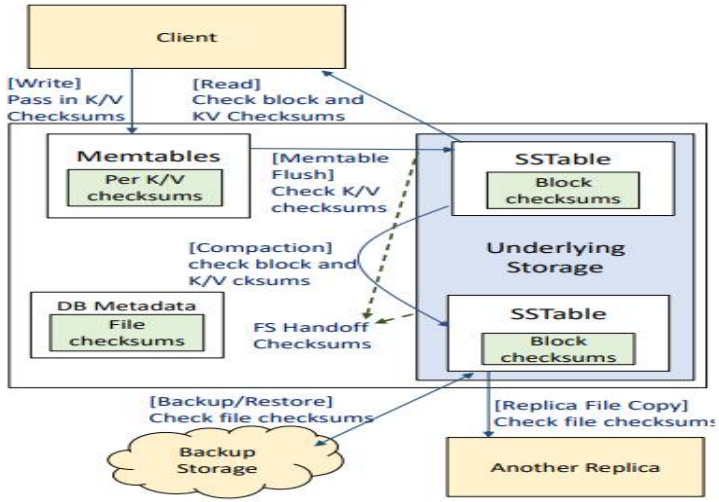
Advance Topic 3: WAL (8/11)

- RocksDB detail: Checksum
  - ✓ Why?
    - Atomic and consistent, but what if bit flip occurs (e.g. due to Flash memory error such as bad block, retention error and disturbance)
    - Require integrity
  - ✓ Checksum: a well-known method for keep Integrity
    - A small-sized block of data derived from main data (e.g. using parity and CRC (Cyclic Redundance Checks))
  - ✓ RocksDB approach
    - Four layers
      - Application, Block, File, Handoff

```
+---------+-----------+-----------+--- ... ---+
|CRC (4B) | Size (2B) | Type (1B) | Payload   |
+---------+-----------+-----------+--- ... ---+

CRC = 32bit hash computed over the payload using CRC
Size = Length of the payload data
Type = Type of record
       (kZeroType, kFullType, kFirstType, kLastType, kMiddleType )
       The type is used to group a bunch of records together to represent
       blocks that are larger than kBlockSize
Payload = Byte stream as long as specified by the payload size
```

(Record format)

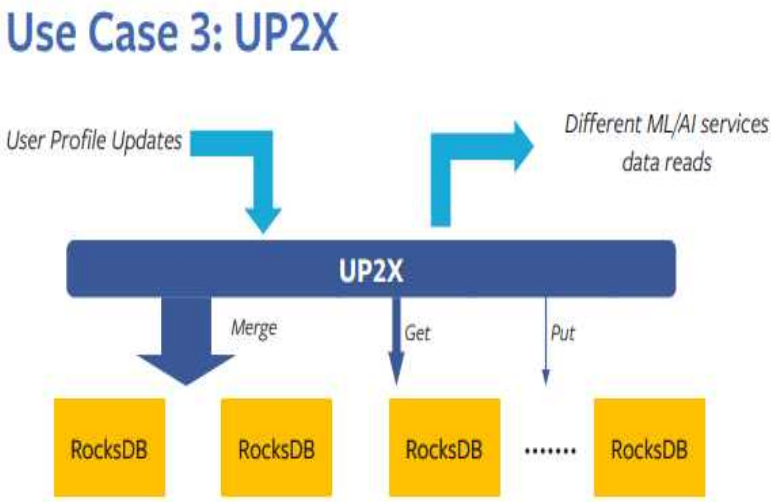


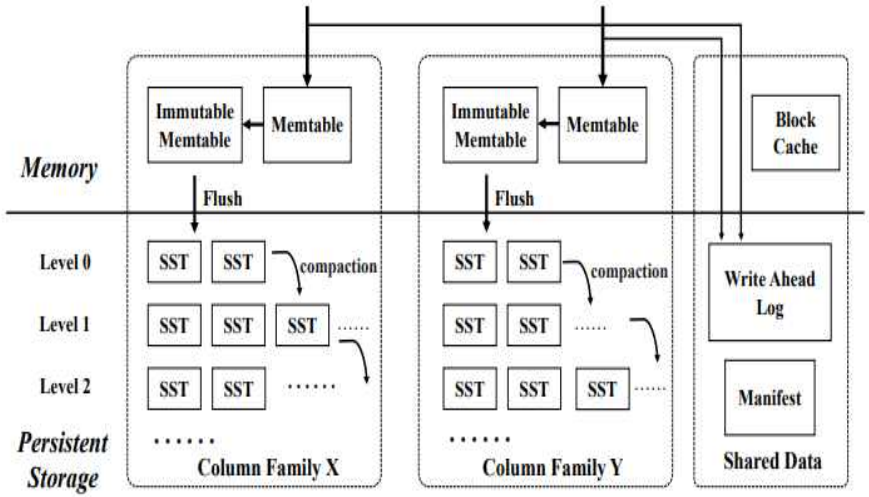
(Multi-layer checksum)

(Source: github.com/facebook/rocksdb/wiki/Write-Ahead-Log-File-Format & RocksDB Experience papers, FAST'21)

Advance Topic 3: WAL (9/11)

- Other features
  - ✓ Merge operator
    - A common pattern: update an existing value ➔ need read-modify-write
    - How about exploit logging and compaction ➔ write only ➔ combine later (e.g. at the compaction time)
  - ✓ Column family
    - A logical partition of DB (e.g. name → phone, name → photo)
    - Support atomic write across column families (share WAL)

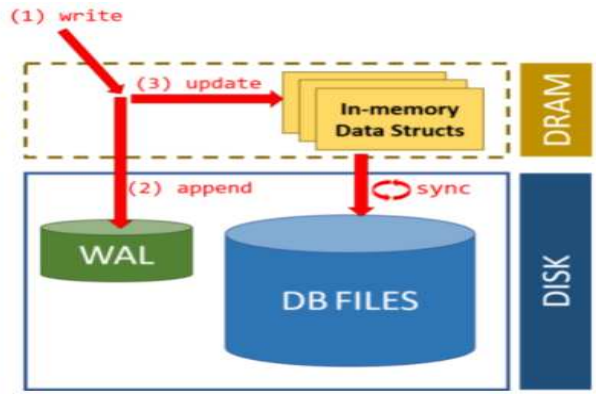




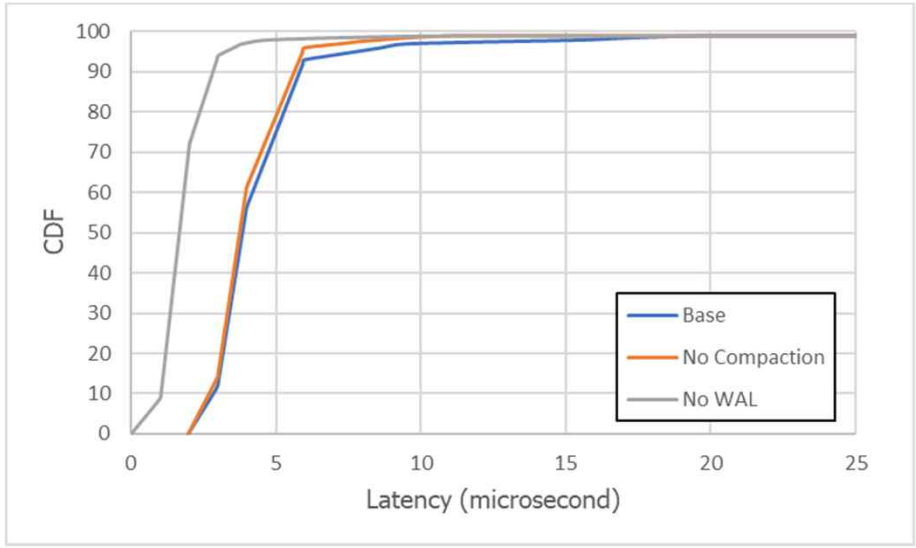
(Source: RocksDB Characterizing papers, FAST'20)

Advance Topic 3: WAL (10/11)

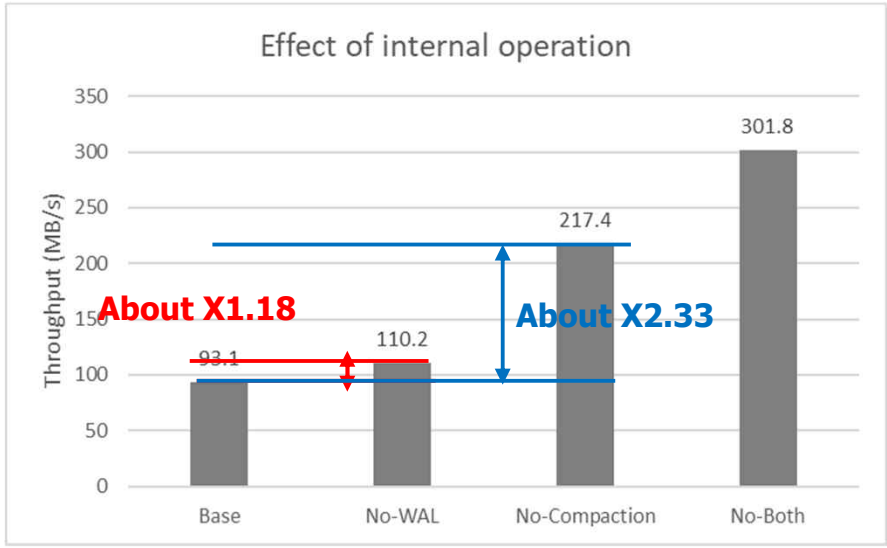
- Performance consideration
  - ✓ WAL incurs additional I/Os
    - Affect latency
  - ✓ How to mitigate?
    - Delayed write (No sync)
    - Fine-grained WAL or parallelism
    - Make use of NVM (Non-volatile memory)



(Source: software.intel.com, "optimizing-WAL-with-intel-optane-persistent-memory")



(latency)



(Throughput)

(Source: our own experiment)

Advance Topic 3: WAL (11/11)

- Take-away Lessons
  - ✓ Understand the requirements for DB Reliability
    - Durability, Atomicity, Consistency, Integrity and Recovery
  - ✓ RocksDB approach
    - WAL (Write Ahead Log): intent logging before actual actions
    - Writebatch: group a set of updates
    - Transaction: maintain valid states by undo or redo
    - Checksum: at various layers
  - ✓ Other features
    - Merge operator: read-modify-write ➔ log and compaction
    - Column family
    - Manifest: Initial point for restart or recovery
    - Checkpoint and snapshot

(Source: www.dreamstime.com)

(Special thank you to DKU Embedded Members and IITP's SW StarLab (No.2021-0-01475)

Content

- What is Key-Value Store?
- RocksDB basic
- Advance Topic 1: Compaction
- Advance Topic 2: Lookup
- Advance Topic 3: WAL
- Advance Topic 4: New approach
  - ✓ Layout: separate key and value
  - ✓ Make use of New storage and ISP
  - ✓ Disaggregate storage and computing
- RocksDB Practice
- Conclusion

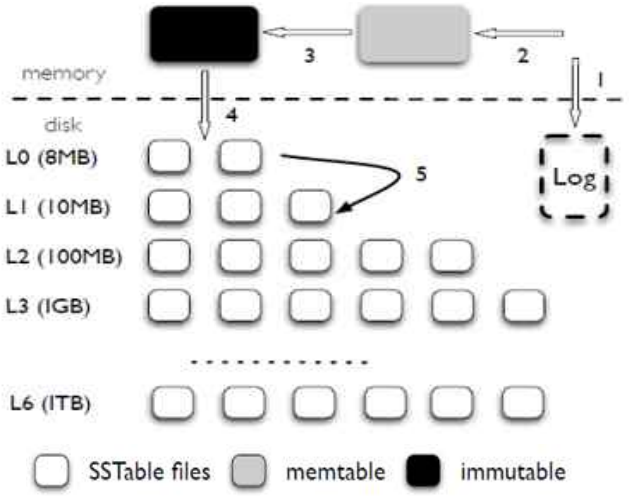
Advance Topic 4: New approach (1/12)

- Lecture Objective
  - ✓ Understand the merit of key value separation
  - ✓ Explore how to utilize new storage for key-value store
  - ✓ Learn about ISP (In-Storge Processing)
  - ✓ Find out how to design a distributed key-value store
  - ✓ Summarize RocksDB features

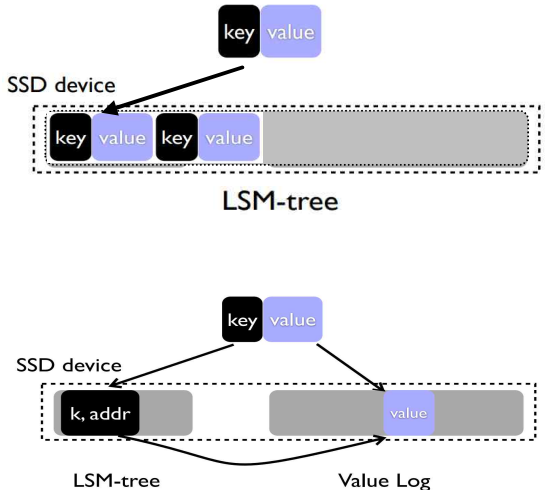
(Source: www.dreamstime.com)

Advance Topic 4: New approach (2/12)

- Case study: Wisckey, FAST'16
  - ✓ Goal: Reduce Compaction Overhead
  - ✓ How to?
    - Existing approach: scheduling, algorithmic, …
    - Proposal: key-value separation
      - Compaction trigger ➔ when the size of L_i is larger size than threshold
      - LSM-tree: key-value pair ➔ compaction uses key only ➔ How about managing key only in a LSM-tree ➔ smaller LSM-tree ➔ less compaction
      - Also, giving a positive impact on caching and searching



(LSM-tree basic)



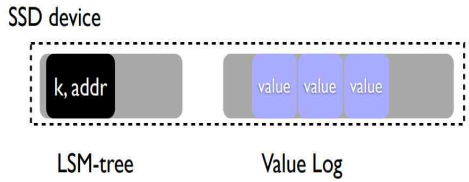
(Key-value separation)

load 100 GB database	LevelDB	WiscKey
limits of files	num of files	num of files
L0	9	7
L1 (5)	30	11
L2 (50)	365	127
L3 (500)	2184	460
L4 (5000)	15752	0
L5 (50000)	23733	0
L6 (500000)	0	0

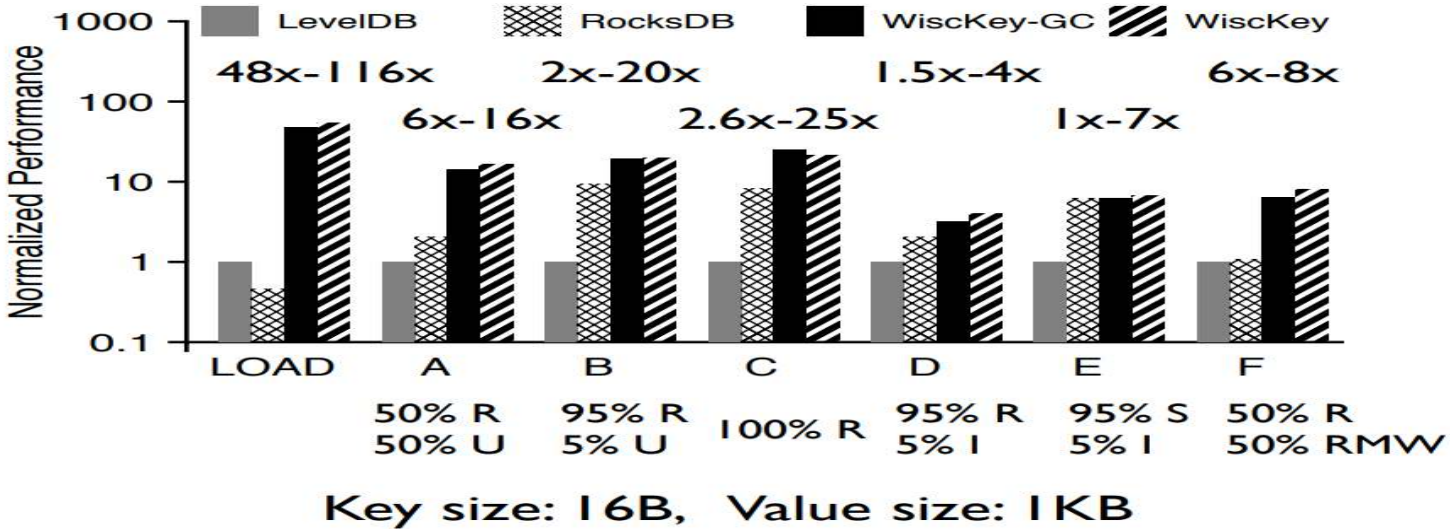
Small LSM-tree: less compaction, fewer levels to search, and better caching

Advance Topic 4: New approach (3/12)

- Case study: Wisckey, FAST'16 (cont')

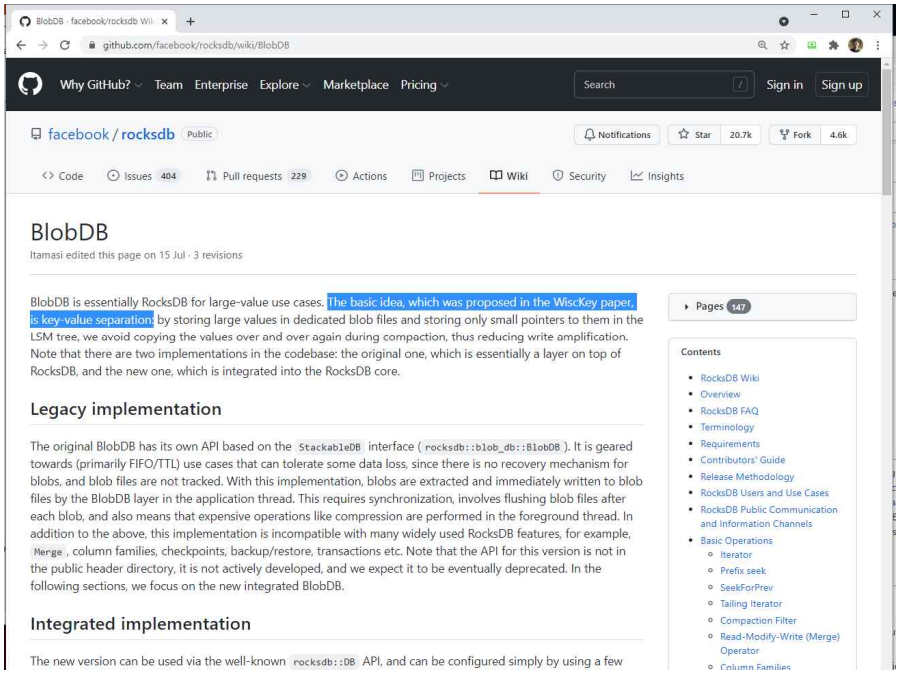


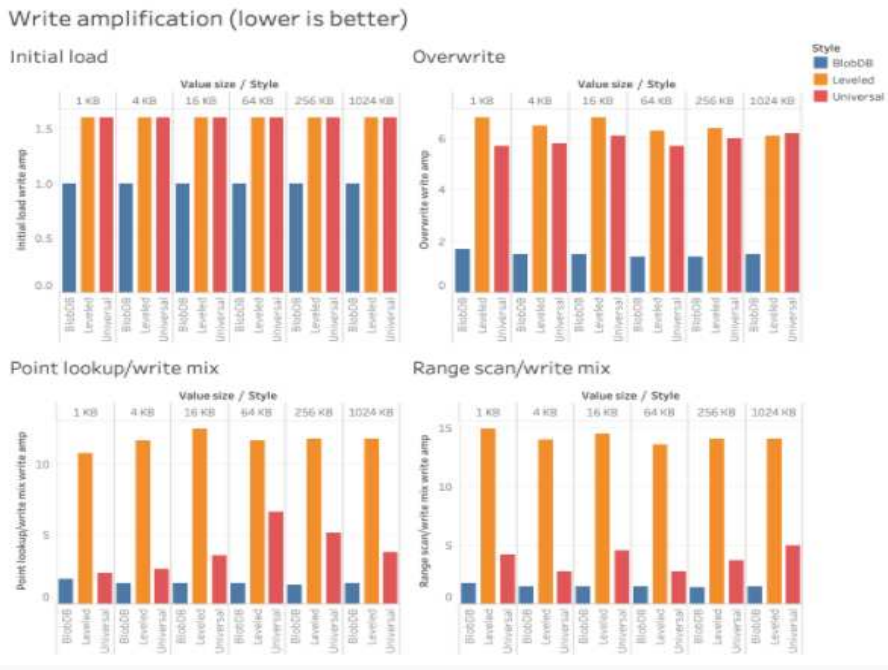
  - ✓ How to? (cont')
    - Value log implementation
      - 1) Queue with head (for new insert) and tail (for GC), 2) Lightweight GC
    - Range query
      - thread pool for concurrent processing (make use of SSD parallelism)
  - ✓ Lessons
    - Simple idea ➔ Substantial gain (adopted by many companies)



Advance Topic 4: New approach (4/12)

- Case study: Wisckey, FAST'16 (cont')
  - ✓ BlobDB
    - Wisckey implementation in RocksDB
    - Overall better performance
    - Initial stage: being actively added new features (e.g. merge operator, range scan optimization, …)

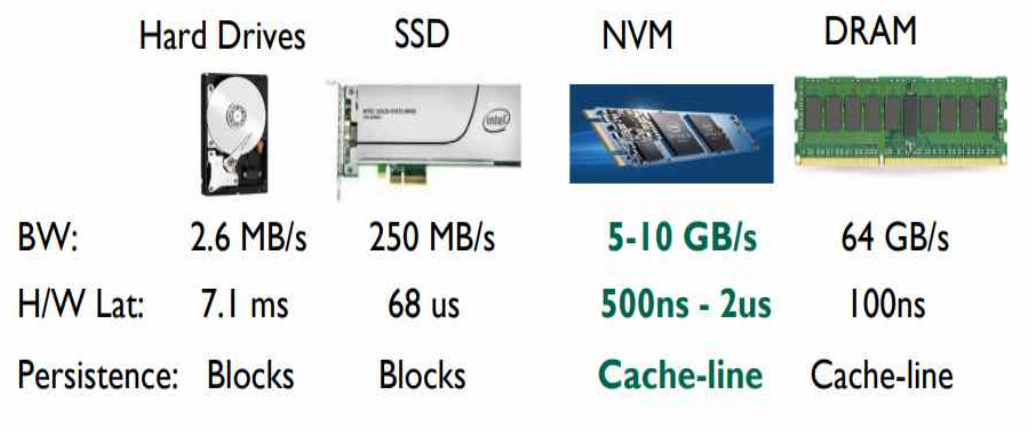




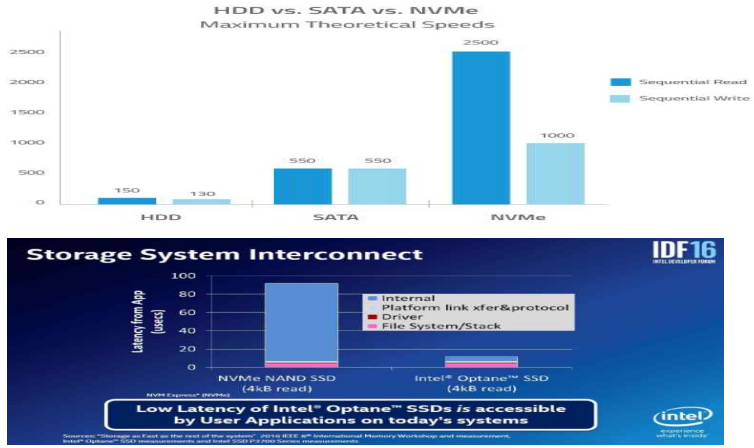
(Source: github.com/facebook/rocksdb/wiki/BlobDB, rocksdb.org/blog/2021/05/26/integrated-blob-db.html)

Advance Topic 4: New approach (5/12)

- Case study: MatrixKV, ATC'20
  - ✓ Goal: Make use of NVM (Non-Volatile Memory) or faster storage (e.g. NVMe SSD, Optane SSD)
    - NVM
      - Characteristics: both non-volatile and byte-addressable, fast
      - Types: PRAM (PCM), MRAM, RRAM, NVDIMM, 3D-Xpoint, Optane, …
    - SSD
      - SATA, NVMe, Optane SSD (SSD based on NVM) ➔ hybrid storage
    - Accelerate compaction via fast NVM/SSD



(NVM characteristics)

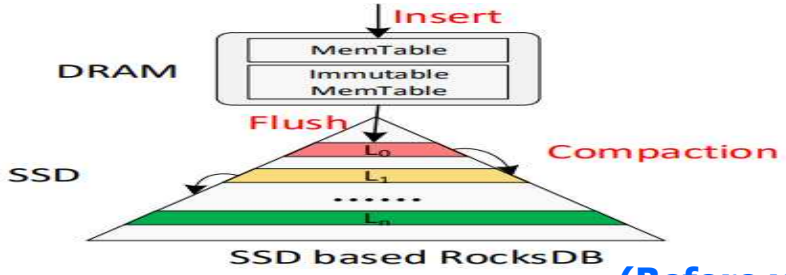


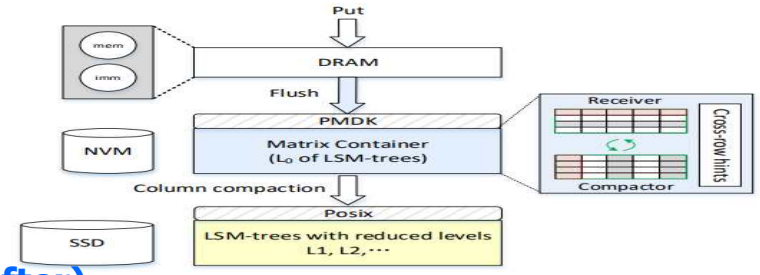
(SATA, NVMe, Optane SSD)

(Source: www.atpinc.com/de/blog/nvme-vs-sata-ssd-pcie-interface and intel document and NoveLSM paper)

Advance Topic 4: New approach (6/12)

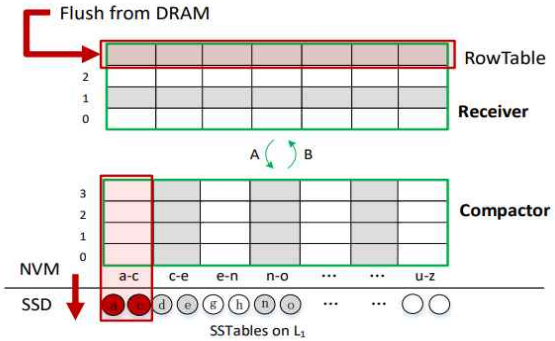
- Case study: MatrixKV, ATC'20
  - ✓ How to?
    - Place L0 in NVM (can extend to L1, L2, …), others in SSD



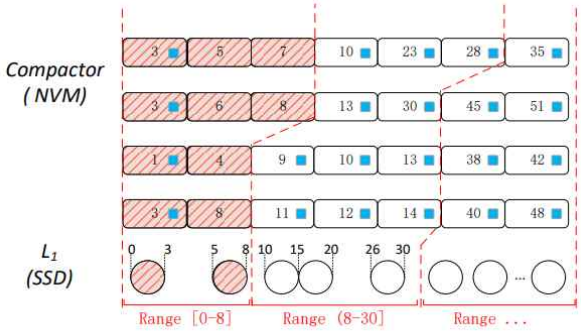


(Before vs. After)

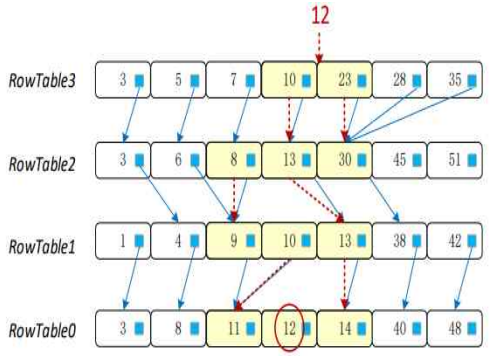
    - Design several novel techniques
      - Matrix form in NVM: 1) receiver (like tiered using rows), 2) compactor
      - Fine-grained compaction: based on non-overlapped compaction
      - Cross-row hint search: using forward pointers for fast search



(Matrix in NVM)



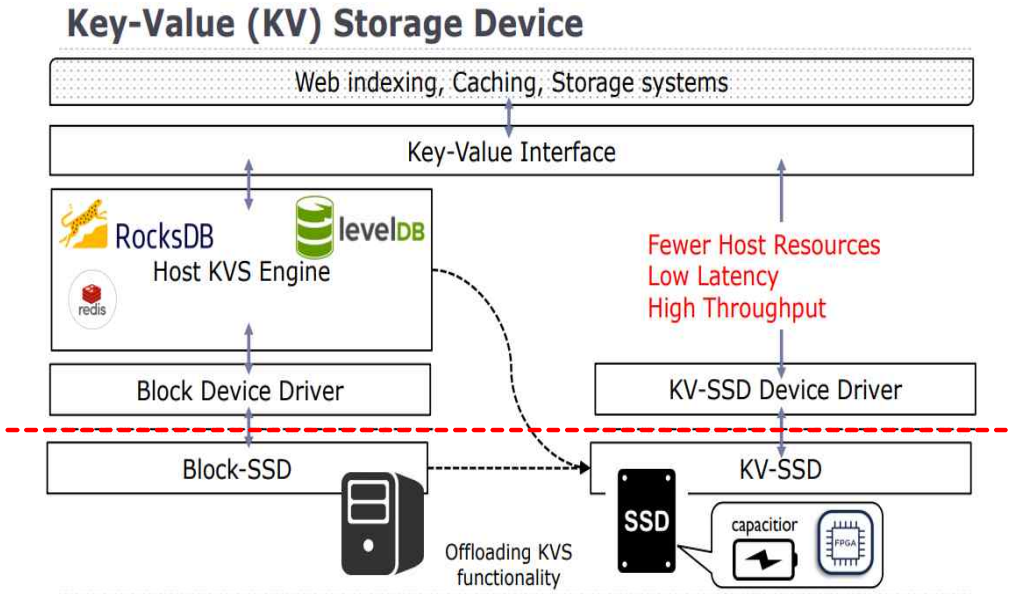
(Micro-grained compaction)



(Cross-row hint)

Advance Topic 4: New approach (7/12)

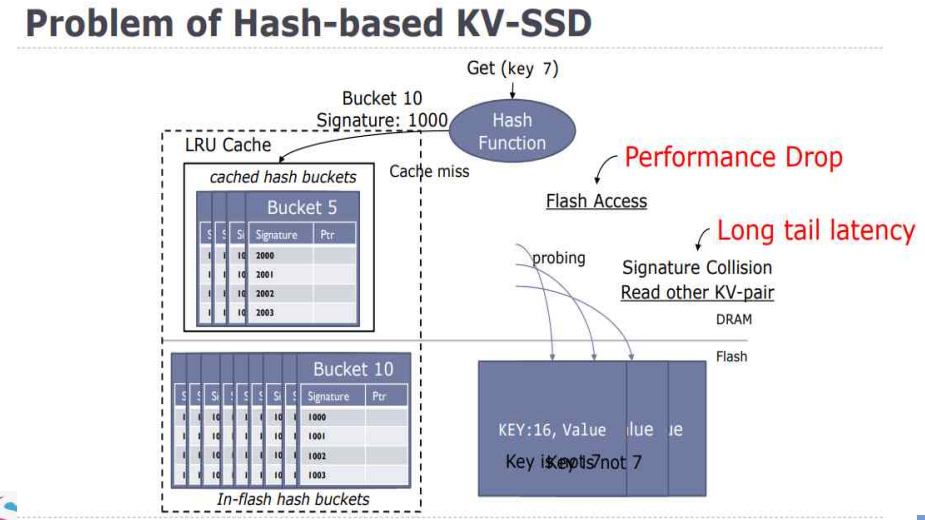
- Case study: Pink, ATC'20
  - ✓ Goal: Make use of ISP (In-Storage Processing)
    - SSD: equip with CPUs (multicores), large DRAM, HW accelerator, …
      - Why CPUs in SSD? For running SW called FTL (Flash Translation Layer) to handle unique Flash features such as erase-before-write and endurance
    - KVSSD
      - Conventional architecture: APP+KV Store in host, FTL in SSD, Block interface
      - KVSSD architecture: APP in host, KV Store + FTL in SSD, KV interface

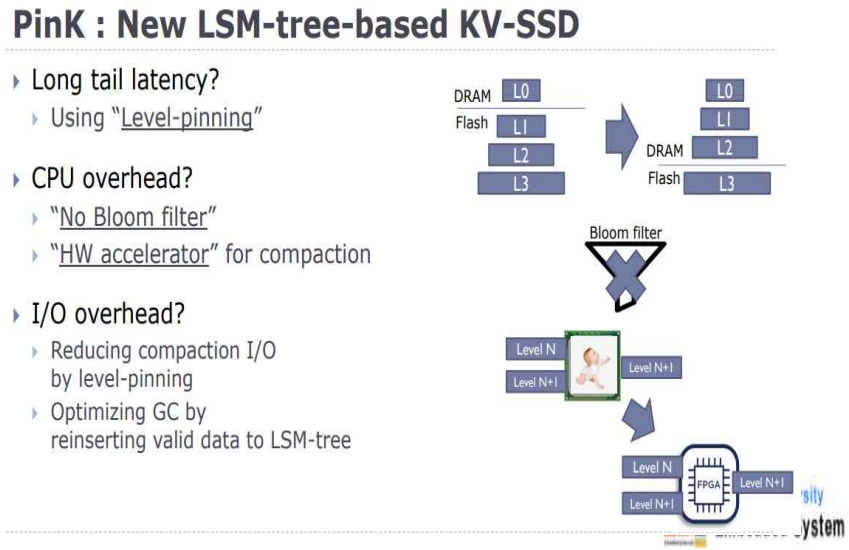


(Source: www.quietpc.com/multicore-pcie-ssd, m.blog.naver.com/ki630808/221968329122, and Samsung KVSSD paper)

Advance Topic 4: New approach (8/12)

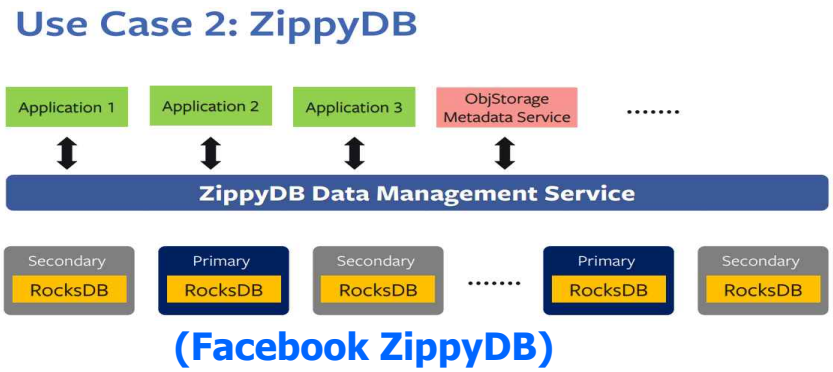
- Case study: Pink, ATC'20
  - ✓ How to?
    - KV store in SSD: Hash based vs. LSM-tree based ➔ Hash causes performance drop and long-tail latency due to hash collision and overflow ➔ LSM-tree based approach
    - LSM-tree issues
      - Long tail due to SSTable accesses ➔ Memory pinning (L0~L2)
      - Bloom filter overhead (due to embedded CPUs) ➔ No Bloom Filter (pinning makes it feasible)
      - Compaction overhead (lead to I/O overhead) ➔ HW accelerator



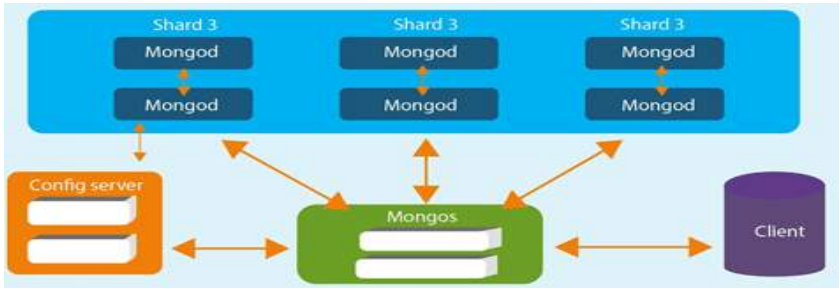


Advance Topic 4: New approach (9/12)

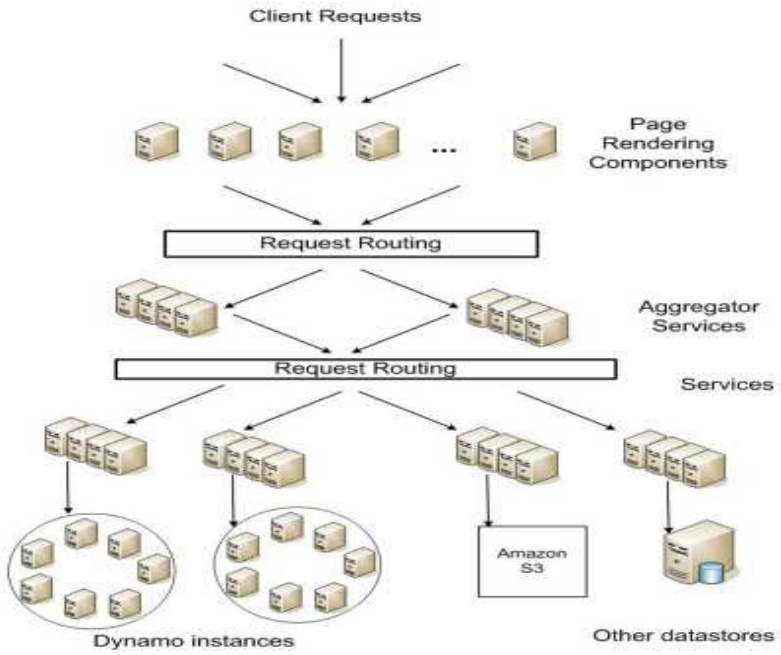
- Case study: Hailstorm, ASPLOS'20
  - ✓ Goal: Load balancing for Distributed Key-Value Store
    - Distributed key-value store: core infrastructure for data service companies to obtain availability and aggregated performance
    - Issue 1. Key distribution: mapping table, hash, circular, …
    - Issue 2. Load imbalance: main topic in this study



(Facebook ZippyDB)



(MongoDB)

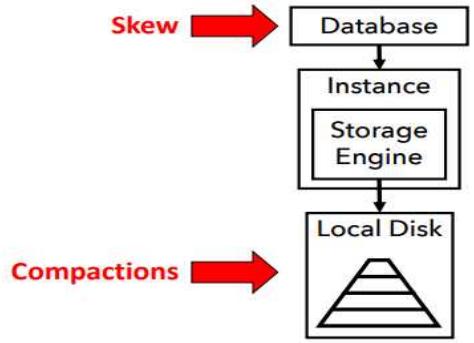


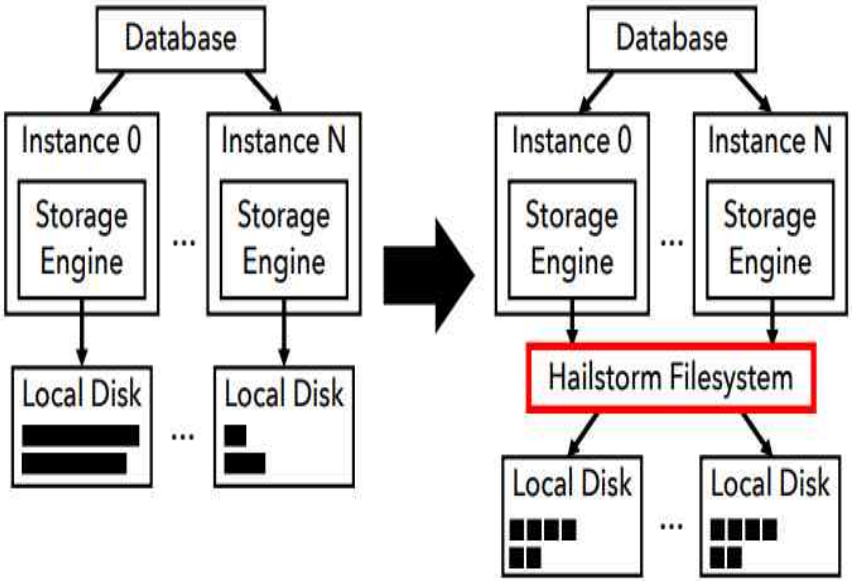
(DynamoDB)

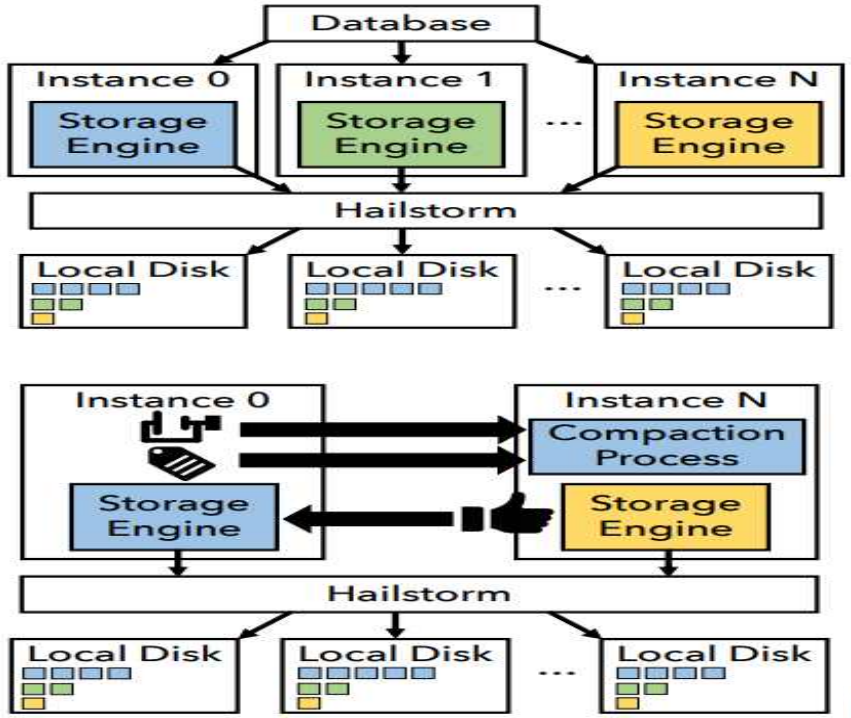
(Source: RocksDB paper, intellipaat.com/blog/what-is-mongodb/ and Dynamo paper)

Advance Topic 4: New approach (10/12)

- Case study: Hailstorm, ASPLOS'20
  - ✓ How to?
    - Causes of load imbalance
      - 1) Data skewness, 2) Compaction
    - Data skewness
      - Employ a middleware (file system in this study) for load balancing
    - Interference due to Compaction
      - Compaction offloading

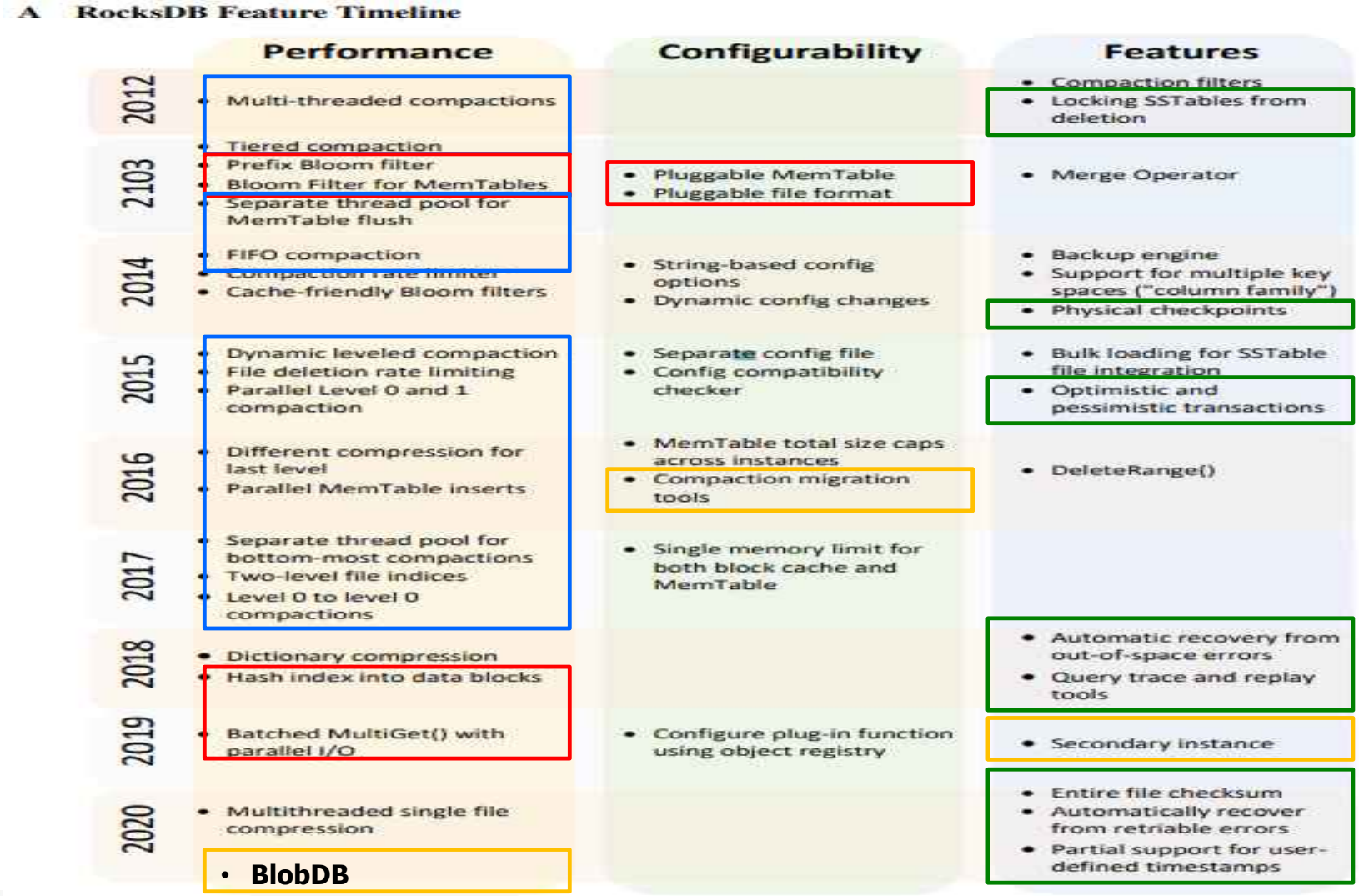






Advance Topic 4: New approach (11/12)

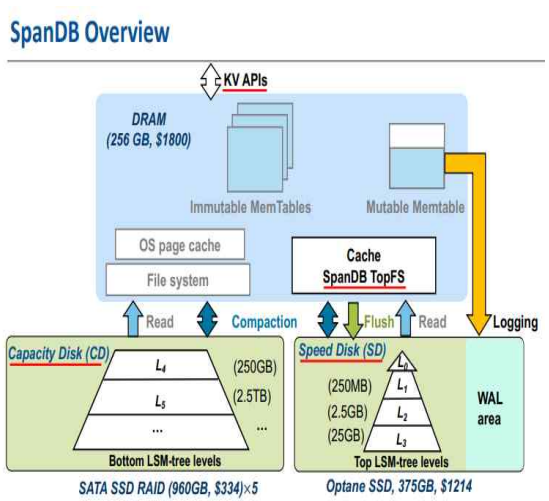
- RocksDB Features (what we have learned at the second lecture)



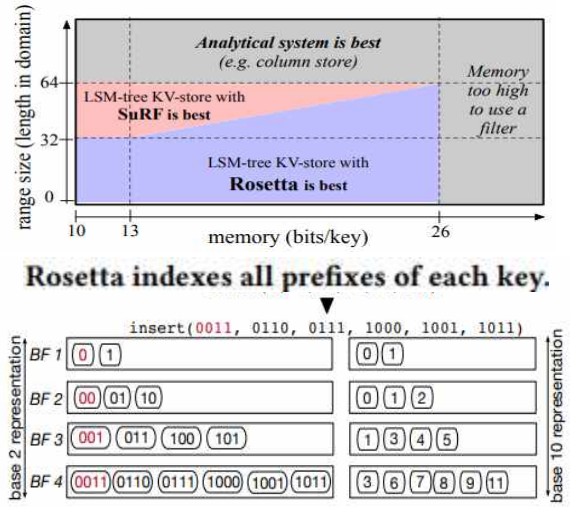
(Source: Facebook RocksDB papers, FAST'21)

Advance Topic 4: New approach (12/12)

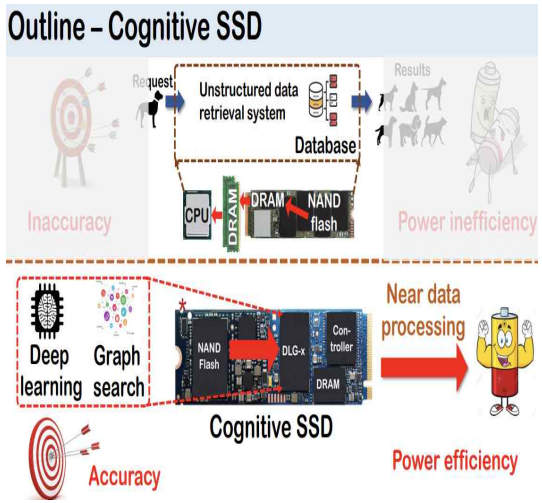
- Take-away Lessons
  - ✓ Outstanding researches are conducted
    - Key value separation for smaller LSM-tree
    - Utilize new storage (e.g. NVM, Optane SSD) and hybrid storage
    - Exploit computing capability in storage
    - Computing offload in a distributed key-value store
  - ✓ Still a hot research topic



(SpanDB, FAST'21)



(Rosetta, SIGMOD'20)



(Cognitive SSD, ATC'19)

(Special thank you to DKU Embedded Members and IITP's SW StarLab (No.2021-0-01475)

Content

- What is Key-Value Store?
- RocksDB basic
- Advance Topic 1: Compaction
- Advance Topic 2: Lookup
- Advance Topic 3: WAL
- Advance Topic 4: New approach
- RocksDB Practice
  - ✓ Install RocksDB in your computer
  - ✓ Run db_bench
  - ✓ How to analyze outputs?
- Conclusion

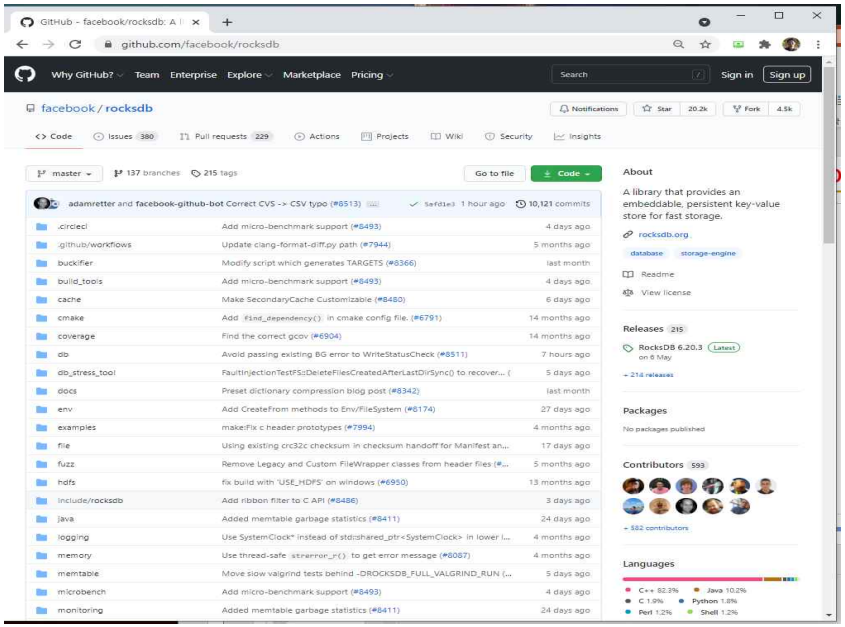
RocksDB Practice (1/12)

- Lecture Objective
  - ✓ Learn how to set up RocksDB in your computer
  - ✓ Understand how to run a benchmark on RocksDB
  - ✓ Investigate representative options for RocksDB
  - ✓ Examine how to analyze evaluation results
  - ✓ Explore the source code of RocksDB

(Source: www.dreamstime.com)

RocksDB Practice (2/12)

- Set up RocksDB in your computer
  - ✓ Preparation: Linux machine, ubuntu on Virtual box, WSL, Cloud
  - ✓ Download RocksDB from github
    - What is Github?
      - Git: distributed version control system
      - Github: open source hosting service based on git
    - How to? $git clone ➔ $ls rocksdb




```
root@shin96:/home/shin96/RocksDB_Festival# git clone https://github.com/facebook/rocksdb.git
Cloning into 'rocksdb'...
remote: Enumerating objects: 99699, done.
remote: Counting objects: 100% (589/589), done.
remote: Compressing objects: 100% (296/296), done.
remote: Total 99699 (delta 308), reused 465 (delta 272), pack-reused 99110
Receiving objects: 100% (99699/99699), 168.23 MiB | 22.59 MiB/s, done.
Resolving deltas: 100% (75591/75591), done.
Checking connectivity... done.
root@shin96:/home/shin96/RocksDB_Festival# ls
rocksdb
root@shin96:/home/shin96/RocksDB_Festival# cd rocksdb/
root@shin96:/home/shin96/RocksDB_Festival/rocksdb# ls
appveyor.yml                db                          HISTORY.md          memtable        test_util
AUTHORS                     db_stress_tool              include             monitoring      third-party
buckifier                   DEFAULT_OPTIONS_HISTORY.md  INSTALL.md          options         thirdparty.inc
build_tools                 defs.bzl                    issue_template.md   plugin          tools
cache                       docs                        java                PLUGINS.md      trace_replay
cmake                       DUMP_FORMAT.md              LANGUAGE-BINDINGS.md port           USERS.md
CMakeLists.txt              env                         LICENSE.Apache      README.md       util
CODE_OF_CONDUCT.md          examples                    LICENSE.leveldb     ROCKSDB_LITE.md utilities
CONTRIBUTING.md             file                        logging             src.mk          Vagrantfile
COPYING                     fuzz                        Makefile            table           WINDOWS_PORT.md
coverage                    hdfs                        memory              TARGETS
root@shin96:/home/shin96/RocksDB_Festival/rocksdb#
```

(Source: https://github.com/facebook/rocksdb)

RocksDB Practice (3/12)

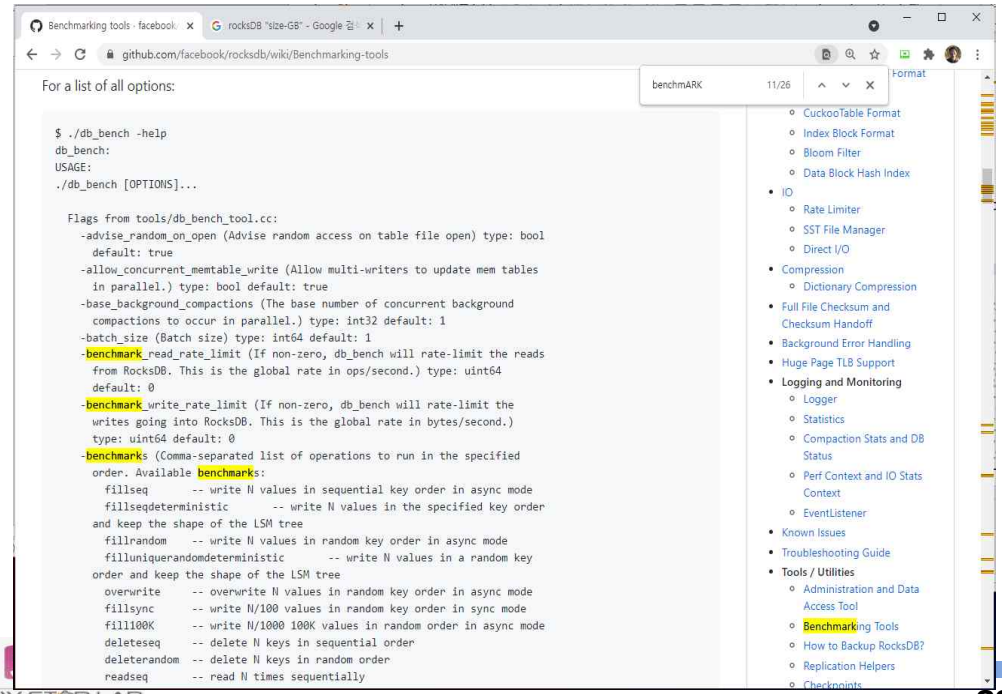
- Build and Run benchmark
  - ✓ Compile
    - Make: A build tool that controls the generation of executables via Makefile
    - How to? $make db_bench
      - It takes quite a long time (depend on your system)
  - ✓ Run benchmark
    - db_bench: a program for testing RocksDB performance
    - How to? $./db_bench

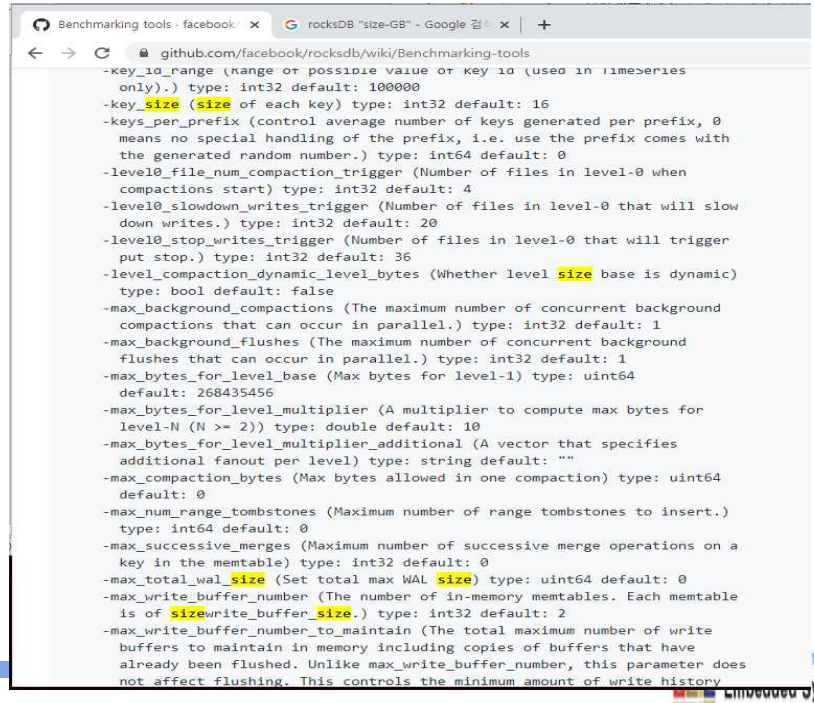
```
root@shin96:/home/shin96/RocksDB_Festival/rocksdb# make db_bench
$DEBUG_LEVEL is 1
Makefile:176: Warning: Compiling in debug mode. Don't use the resulting binary in production
$DEBUG_LEVEL is 1
Makefile:176: Warning: Compiling in debug mode. Don't use the resulting binary in production
  CC       tools/db_bench.o
  CC       tools/db_bench_tool.o
  CC       tools/simulated_hybrid_file_system.o
  CC       test_util/testutil.o
  CC       cache/cache.o
  CC       cache/cache_entry_roles.o
  CC       cache/clock_cache.o
  CC       cache/lru_cache.o
  CC       cache/sharded_cache.o
  CC       db/arena_wrapped_db_iter.o
  CC       db/blob/blob_fetcher.o
```

```
root@shin96:/home/shin96/RocksDB_Festival/rocksdb# ls
appveyor.yml        coverage                    file                LICENSE.Apache  plugin          thirdparty.inc
AUTHORS             db                          fuzz                LICENSE.leveldb PLUGINS.md      tools
buckifier           db_bench                    hdfs                logging         port            trace_replay
build_tools         db_stress_tool              HISTORY.md          make_config.mk  README.md       USERS.md
cache               DEFAULT_OPTIONS_HISTORY.md  include             Makefile        ROCKSDB_LITE.md util
cmake               defs.bzl                    INSTALL.md          memory          src.mk          utilities
CMakeLists.txt      docs                        issue_template.md   memtable        table           Vagrantfile
CODE_OF_CONDUCT.md  DUMP_FORMAT.md              java                mnt             TARGETS         WINDOWS_PORT.md
CONTRIBUTING.md     env                         LANGUAGE-BINDINGS.md monitoring     test_util
COPYING             examples                    librocksdb_debug.a  options         third-party
root@shin96:/home/shin96/RocksDB_Festival/rocksdb# ./db_bench --db=./mnt --benchmarks=fillseq --num=1000000
Initializing RocksDB Options from the specified file
Initializing RocksDB Options from command-line flags
RocksDB:    version 6.22
Date:       Fri Jul  2 22:02:50 2021
CPU:        4 * Intel(R) Core(TM) i5-6600 CPU @ 3.30GHz
CPUCache:   6144 KB
Keys:       16 bytes each (+ 0 bytes user-defined timestamp)
Values:     100 bytes each (50 bytes after compression)
Entries:    1000000
Prefix:    0 bytes
Keys per prefix:    0
RawSize:    110.6 MB (estimated)
FileSize:   62.9 MB (estimated)
Write rate: 0 bytes/second
Read rate: 0 ops/second
Compression: Snappy
Compression sampling rate: 0
Memtablerep: skip_list
Perf Level: 1
WARNING: Assertions are enabled; benchmarks unnecessarily slow
------------------------------------------------
Initializing RocksDB Options from the specified file
Initializing RocksDB Options from command-line flags
DB path: [./mnt]
fillseq      :       2.606 micros/op 383782 ops/sec;   42.5 MB/s
root@shin96:/home/shin96/RocksDB_Festival/rocksdb#
```

RocksDB Practice (4/12)

- Configurability
  - ✓ There are a lot of options (parameters for configuration)
    - Balance between customizability and self-adaptability
  - ✓ 1) Benchmark related options
    - Benchmark: a series of well-defined test to measure performance
    - Workload type: sequential, random, # of threads, …
    - Workload size: num (# of KV pairs), key_size, value_size, …





RocksDB Practice (5/12)

- Configurability (cont')
  - ✓ 2) Memtable related options
    - write_buffer_size: Size of a single Memtable (default: 64MB).
    - Max_write_buffer_number: # of Memtables (default: 2)
      - 1 for mutable and others for immutable that wait for being flushed
  - ✓ 3) SSTable related options
    - Level0_file_num_compaction_trigger (default 4): # of files to trigger
    - All non-0 levels: target_size, max_bytes_for_level_multiplier (default 10)
      - if (level_compaction_dynamic_level_bytes == false)
        - target_size(L_1) =max_bytes_for_level_base (default=256MB, 64*4 from L0)
        - target_size(L_{n+1}) = target_size(L_n) * max_bytes_for_level_multiplier
      - else
        - determine the last level size as the actual size
        - then, target_size(L_{n-1}) = target_size(L_n) / max_bytes_for_level_multiplier

(Source: https://github.com/facebook/rocksdb/wiki/Leveled-Compaction)

RocksDB Practice (6/12)

- Configurability (cont')
  - ✓ 4) Compaction related options
    - Compaction_style: kCompactionStyleLevel (default), Universal, FIFO, …
    - disable_auto_compactions: useful in the case of bursty case
    - max_background_jobs, max_background_compactions, level_compaction_dynamic_level_bytes, …
  - ✓ 5) Block cache related options
    - Block cache size, policy (LRU or Clock), allow_OS_buffer,, …
  - ✓ 6) Storage related options
    - Block_size (default 4KB): I/O unit
      - When reading a KV pair from a file, an entire block is loaded into memory
    - I/O rate limiter (e.g. rate_bytes_per_sec)
      - Throttle maximum write speed to give a chance to read
  - ✓ 7) Compression related options
  - ✓ 8) …

```
cf_options.level_compaction_dynamic_level_bytes = true;
options.max_background_compactions = 4;
options.max_background_flushes = 2;
options.bytes_per_sync = 1048576;
options.compaction_pri = kMinOverlappingRatio;
table_options.block_size = 16 * 1024;
table_options.cache_index_and_filter_blocks = true;
table_options.pin_l0_filter_and_index_blocks_in_cache = true;
table_options.format_version = <the latest version>;
```

(Source: https://github.com/facebook/rocksdb/wiki/Setup-Options-and-Basic-Tuning)

RocksDB Practice (7/12)

- Evaluation examples: from RocksDB wiki
  - ✓ Set up
    - AWS instance: 8 CPU, 32 GB Memory, 1 x 300 NVMe SSD
    - Linux kernel: 4.14, FS: XFS with discard enabled, RocksDB: 6.10
  - ✓ Tests (being added)
    - Test1 (bulkload): Bulk load of Keys in Random Order (write only)
    - Test2 (overwrite): Random write, started from Test 1
    - Test3 (readwhilewriting): randomly read keys and ongoing updates to existing keys, Based on Test 2
    - Test4 (randomread): random read performance
  - ✓ Result 1
    - 100M Keys, 2KB value, Default block size (4KB), bulkload time = 2018s

Test	ops/sec	mb/sec	Size-GB	L0_GB	Sum_GB	W-Amp	W-MB/s	usec/op	p50	p75	p9
bulkload	272448	537.3	0.1	85.3	85.3	1.0	242.6	3.7	0.7	1.1	
overwrite	22940	45.2	0.1	229.3	879.4	3.8	169.0	1394.9	212.9	350.7	76(
readwhilewriting	87093	154.2	0.1	5.4	162.6	30.1	31.0	367.4	369.2	491.9	22(
readrandom	95666	169.9	0.1	0.0	0.0	0	0	334.5	411.1	498.7	7-

(Source: https://github.com/facebook/rocksdb/wiki/Performance-Benchmarks)

RocksDB Practice (8/12)

- Evaluation examples: from Rocksdb wiki (cont')
  - ✓ Result 2
    - NUM_KEYS=9x10^8, NUM_THREADS=32, CACHE_SIZE=6442450944
    - Analyze while change an option (block size in this case)
      - With different block size: 8KB, 4KB, 16KB, 8KB with DIO, 8KB with RL
      - RL (Rate Limiting): a timed rate-limited operation was place before the reported operation ➔ give a chance to do background jobs ➔ creating more predictable results
    - Results are given per each test

Test Case 1 : benchmark.sh bulkload

8K: Complete bulkload in 4560 seconds 4K: Complete bulkload in 5215 seconds 16K: Complete bulkload in 3996 seconds DIO: Complete bulkload in 4547 seconds 8K RL: Complete bulkload in 4388 seconds

Block	ops/sec	mb/sec	Size-GB	L0_GB	Sum_GB	W-Amp	W-MB/s	usec/op	p50	p75	p99	p99.9
8K	924468	370.3	0.2	157.1	157.1	1.0	167.5	1.1	0.5	0.8	2	4
4K	853217	341.8	0.2	165.3	165.3	1.0	165.9	1.2	0.5	0.8	2	4
16K	1027567	411.6	0.1	149.0	149.0	1.0	181.6	1.0	0.5	0.8	2	3
DIO	921342	369.0	0.2	156.6	156.6	1.0	167.0	1.1	0.5	0.8	2	4
8K RL	989786	396.5	0.2	159.4	159.4	1.0	179.5	1.0	0.5	0.8	2	4

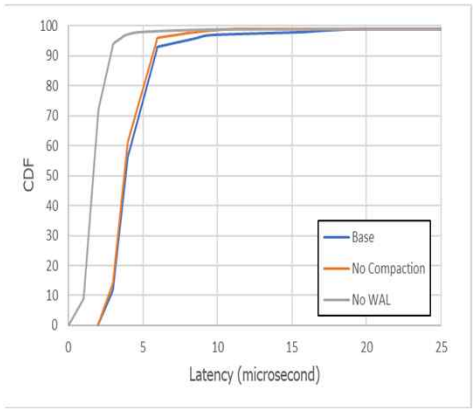
Test Case 2 : benchmark.sh overwrite

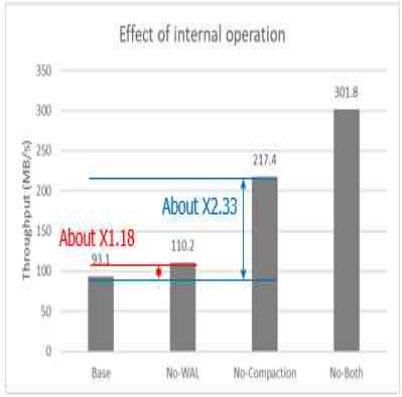
Block	ops/sec	mb/sec	Size-GB	L0_GB	Sum_GB	W-Amp	W-MB/s	usec/op
8K	85756	34.3	0.1	161.4	739.9	4.5	142.2	373.1
4K	79856	32.0	0.2	166.0	716.9	4.3	136.3	400.7
16K	93678	37.5	0.1	174.4	825.0	4.7	156.8	341.6
DIO	85655	34.3	0.1	163.9	734.9	4.4	140.7	373.6
8K RL	85542	34.3	0.1	161.2	757.8	4.7	143.6	748.1

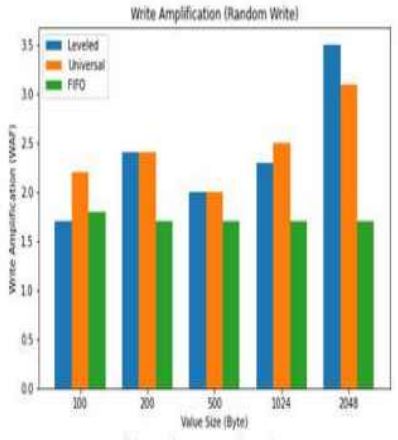
(Source: https://github.com/facebook/rocksdb/wiki/Performance-Benchmarks)

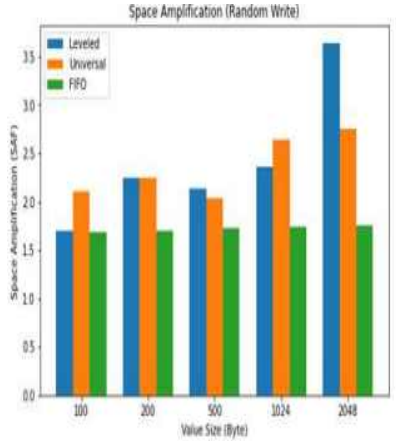
RocksDB Practice (9/12)

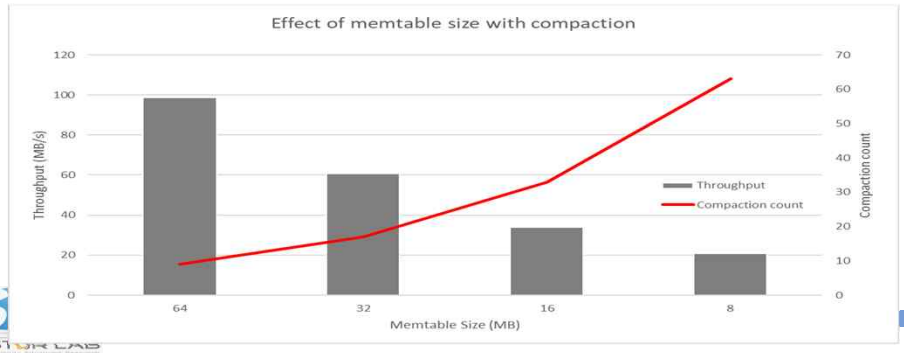
- Evaluation examples: from our experiments
  - ✓ Result 3
    - Effects of internal operations
    - Effect of Memtable size: smaller size incurs higher compaction
    - Effects of Compaction Policies (styles)
    - # of SSTables involved in each compaction

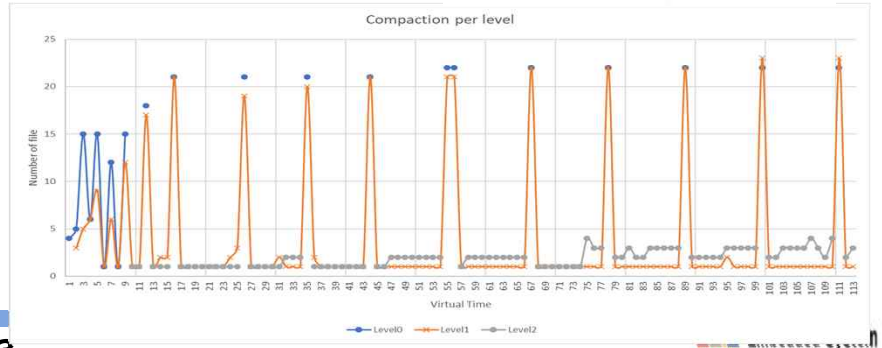












RocksDB Practice (10/12)

- How to modify RocksDB?
  - ✓ Browse source code
  - ✓ Modify a file (e.g. $vi tools/db_bench_tool.cc for Run())
  - ✓ Build and run the db_bench

```
root@linux-server-93:/home/choigunhee/hojin/RocksDB_Explorer# ls
appveyor.yml        db_stress_tool          issue_template.md  parsing_csv      third-party
AUTHORS             DEFAULT_OPTIONS_HISTORY.md  java           plugin           thirdparty.inc
buckifier           defs.bzl                LANGUAGE-BINDINGS.md  PLUGINS.md    tools
build_tools         docs                    librocksdb_debug.a  port            trace_replay
cache               DUMP_FORMAT.md          LICENSE.Apache     python_parser    USERS.md
cmake               env                     LICENSE.leveldb    README.md        util
CMakeLists.txt      examples                logging            result_txt       utilities
CODE_OF_CONDUCT.md  file                    make_config.mk     RocksDB_explorer_sh  Vagrantfile
CONTRIBUTING.md     fuzz                    Makefile           ROCKSDB_LITE.md  WINDOWS_PORT.md
COPYING             hdfs                    memory             src.mk
coverage            HISTORY.md              memtable           table
db                  include                 monitoring         TARGETS
db_bench            INSTALL.md              options            test_util
root@linux-server-93:/home/choigunhee/hojin/RocksDB_Explorer#
```

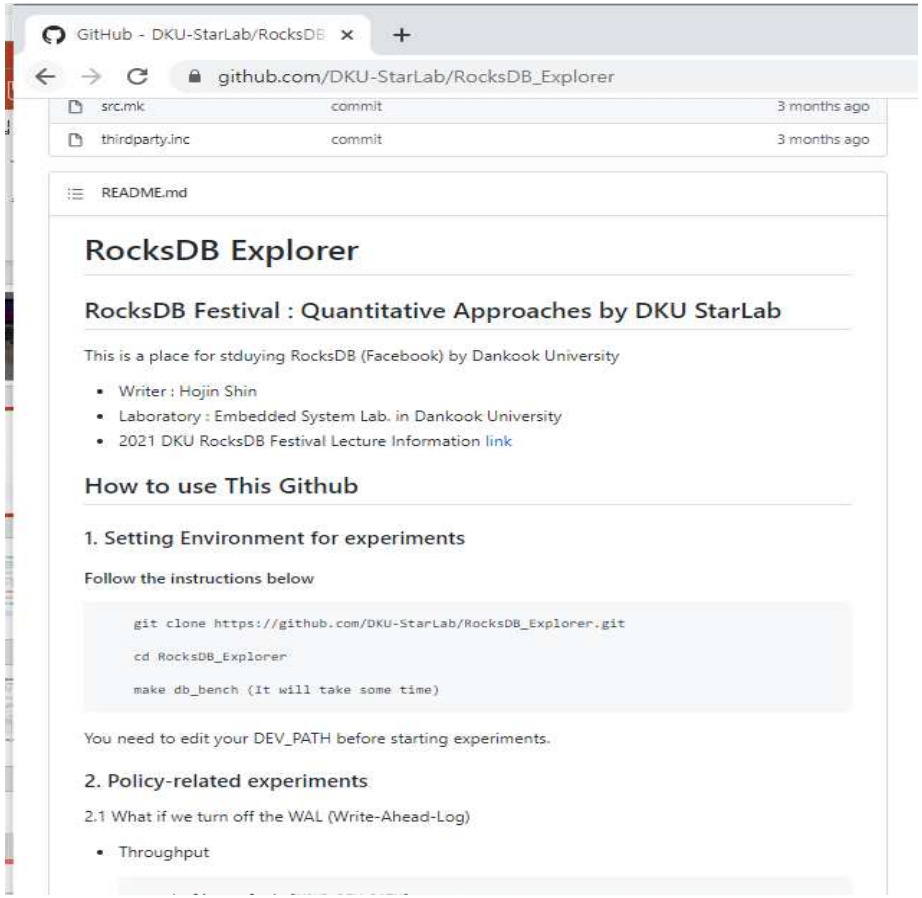
```
void Run() {
  if (!SanityCheck()) {
    ErrorExit();
  }
  Open(&open_options_);
  PrintHeader();
  fprintf(stdout, "Source Code Modifying!!!\n");
  std::stringstream benchmark_stream(FLAGS_benchmarks);
```

```
root@shin96:/home/shin96/RocksDB_Festival/rocksdb# make db_bench
$DEBUG_LEVEL is 1
Makefile:176: Warning: Compiling in debug mode. Don't use the resulting binary in production
$DEBUG_LEVEL is 1
Makefile:176: Warning: Compiling in debug mode. Don't use the resulting binary in production
  CC       tools/db_bench_tool.o
  CCLD     db_bench
root@shin96:/home/shin96/RocksDB_Festival/rocksdb#
```

```
root@shin96:/home/shin96/RocksDB_Festival/rocksdb# ./db_bench --db=
Initializing RocksDB Options from the specified file
Initializing RocksDB Options from command-line flags
RocksDB:    version 6.22
Date:       Fri Jul  2 22:06:32 2021
CPU:        4 * Intel(R) Core(TM) i5-6600 CPU @ 3.30GHz
CPUCache:   6144 KB
Keys:       16 bytes each (+ 0 bytes user-defined timestamp)
Values:     100 bytes each (50 bytes after compression)
Entries:    1000000
Prefix:    0 bytes
Keys per prefix:    0
RawSize:    110.6 MB (estimated)
FileSize:   62.9 MB (estimated)
Write rate: 0 bytes/second
Read rate: 0 ops/second
Compression: Snappy
Compression sampling rate: 0
Memtablerep: skip_list
Perf Level: 1
WARNING: Assertions are enabled; benchmarks unnecessarily slow
------------------------------------------------
Source Code Modifying!!!
Initializing RocksDB Options from the specified file
Initializing RocksDB Options from command-line flags
DB path: [./mnt]
fillseq      :       2.649 micros/op 377562 ops/sec;   41.8 MB/s
root@shin96:/home/shin96/RocksDB_Festival/rocksdb#
```

RocksDB Practice (11/12)

- RocksDB explorer
  - ✓ By Hojin Shin (at DKU Lab.)
    - Logging for source code analysis, Scripts for various experiments, …




```
root@linux-server-93:/home/choigunhee/hojin/RocksDB_Explorer/RocksDB_explorer_sh# ls
compaction_latency.sh      configuration_threads.sh  policy_wal.sh   workload_key_value_size.sh
configuration_memtable.sh  policy_compaction.sh      wal_latency.sh  workload_value_size_distribution.sh
```

(scripts)

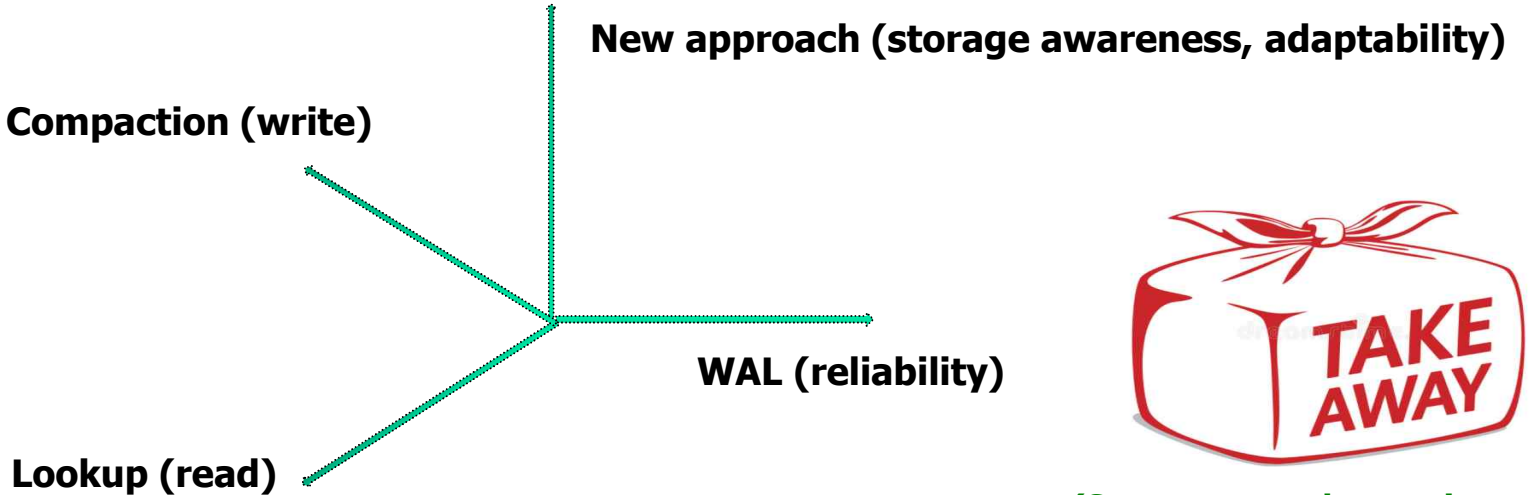
```
Keys:       16 bytes each (+ 0 bytes user-defined timestamp)
Values:     1024 bytes each (1024 bytes after compression)
Entries:    5000000
Prefix:    0 bytes
Keys per prefix:    0
RawSize:    4959.1 MB (estimated)
FileSize:   4959.1 MB (estimated)
Write rate: 0 bytes/second
Read rate: 0 ops/second
Compression: Snappy
Compression sampling rate: 0
Memtablerep: skip_list
Perf Level: 1
WARNING: Assertions are enabled; benchmarks unnecessarily slow
------------------------------------------------
Initializing RocksDB Options from the specified file
Initializing RocksDB Options from command-line flags
db_bench Write Flow - MaybeScheduleFlushOrCompaction() in db_impl_compaction_flush.cc
db_bench Write Flow - MaybeScheduleFlushOrCompaction() in db_impl_compaction_flush.cc
DB path: [../mnt/rocksdb_test/]
db_bench Write Flow - DoWrite() in db_bench_tool.cc
db_bench Write Flow - Write() in db_impl_write.cc
db_bench Write Flow - WriteImpl() in db_impl_write.cc
db_bench Write Flow - PipelinedWriteImpl() in db_impl_write.cc
db_bench Write Flow - PreprocessWrite() in db_impl_write.cc
db_bench Write Flow - Write() in db_impl_write.cc
db_bench Write Flow - WriteImpl() in db_impl_write.cc
```

(control flow analysis)

(Source: https://github.com/DKU-StarLab/RocksDB_Explorer)

RocksDB Practice (12/12)

- Take-away Lessons
  - ✓ Set up, run, analyze RocksDB is straightforward :-)
    - Open source (waiting for your contribution)
    - Provide a variety of options
    - Being actively developed by amazing experts
  - ✓ Let's do our own research!!



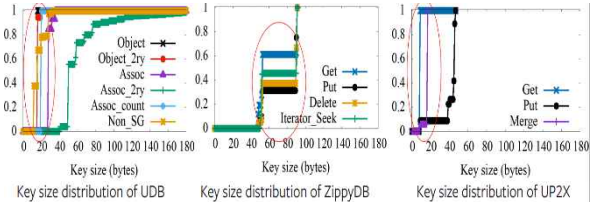
(Source: www.dreamstime.com)

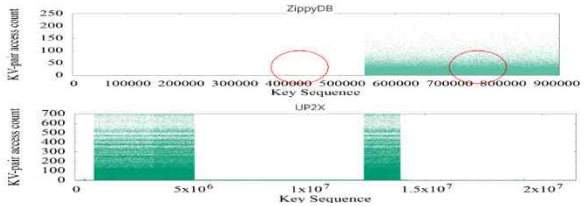
(Special thank you to DKU Embedded Members and IITP's SW StarLab (No.2021-0-01475)

Discussion

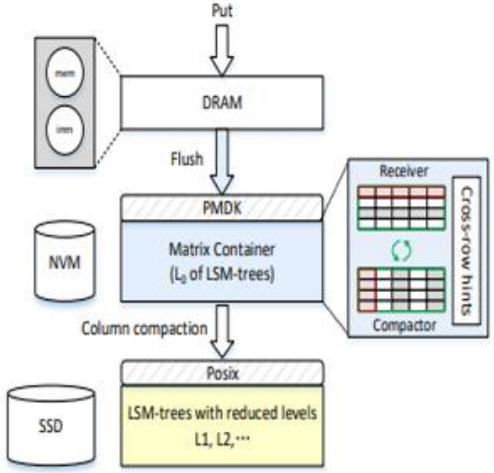
Appendix

- Interesting recent studies regarding Key-Value Store

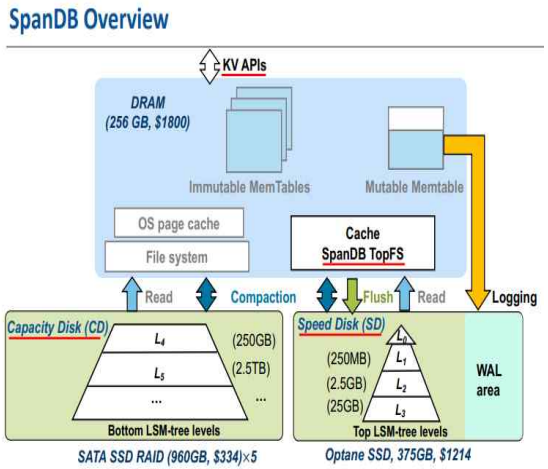




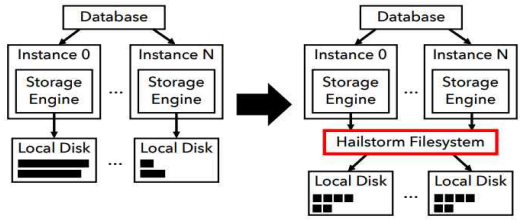
(RocksDB analysis, FAST'20)

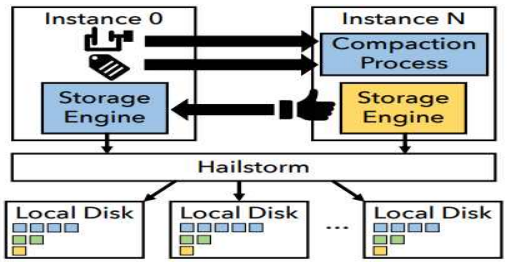


(MatrixKV, ATC'20)

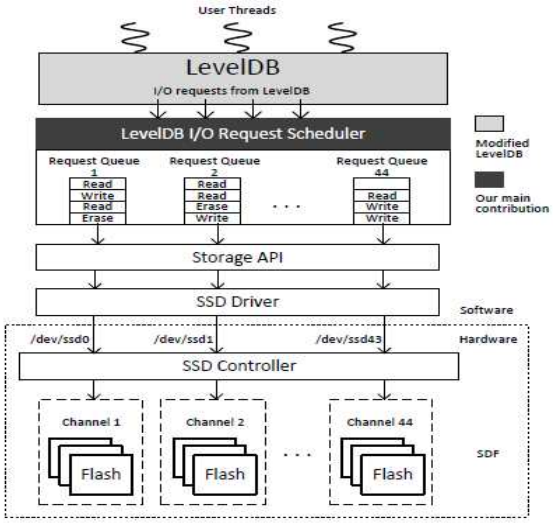


(SpanDB, FAST'21)

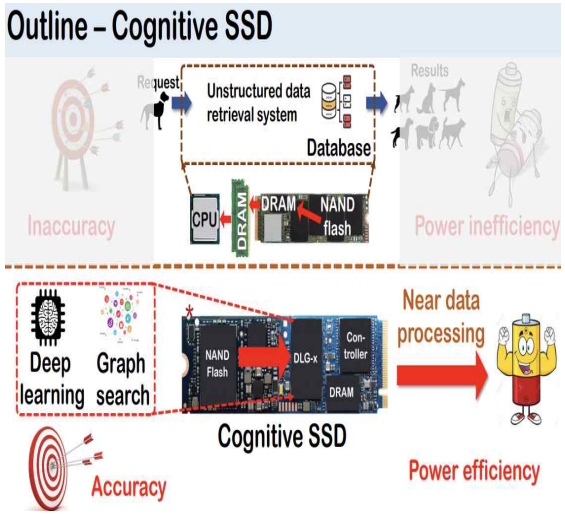




(Hailstorm, ASPLOS'20)



(SDF, ASPLOS'14)



(Cognitive SSD, ATC'19)

Appendix

- Interesting recent studies regarding Key-Value Store

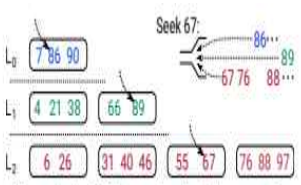


Figure 1: An LSM-tree using leveled compaction

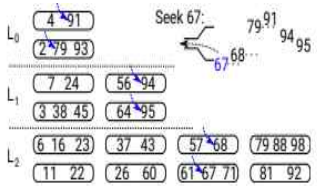


Figure 2: An LSM-tree using tiered compaction

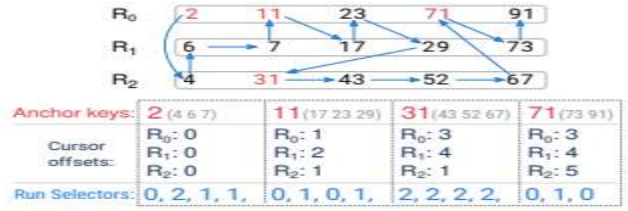
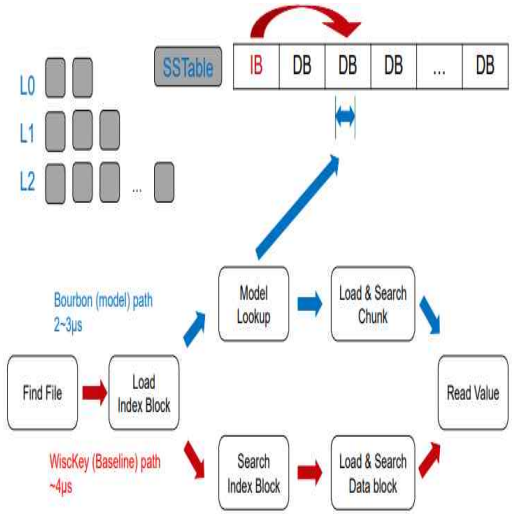
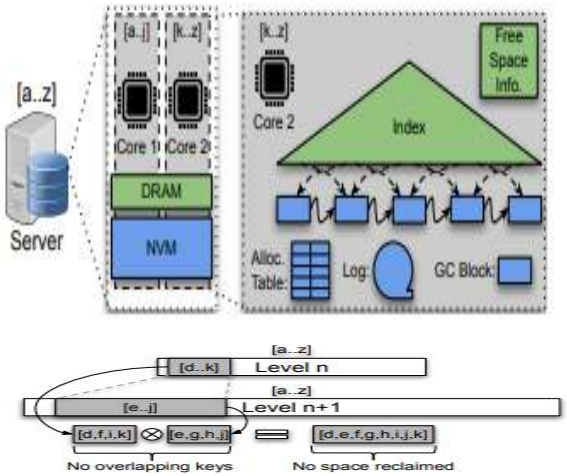


Figure 3: A sorted view of three sorted runs with REMIX

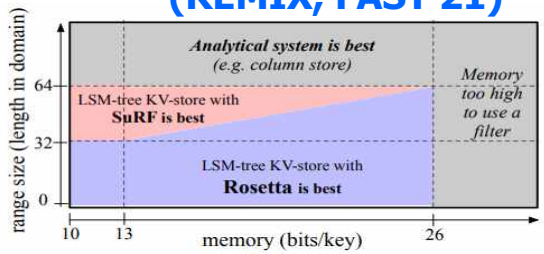
(REMIX, FAST'21)

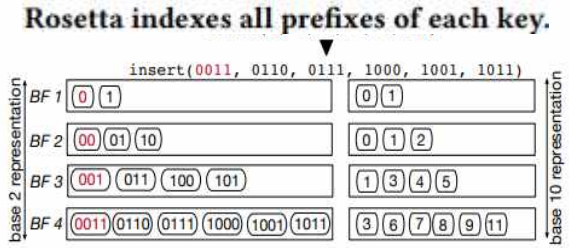


(Bourbon, OSDI'20)

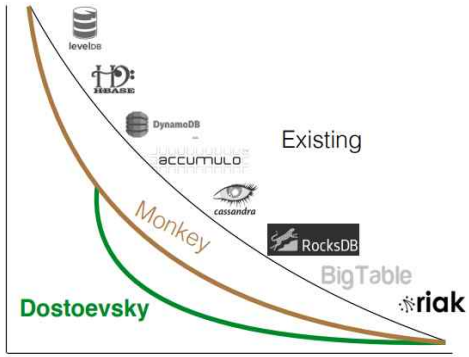


(RStore, VLDB'20)

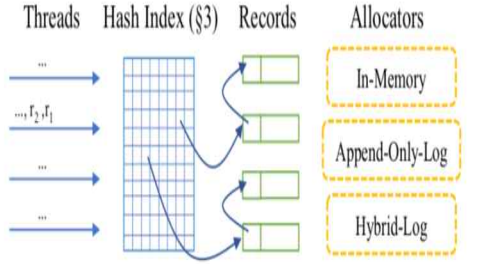




(Rosetta, SIGMOD'20)



(Dostoevsky, SIGMOD'18)



	Record Allocator	Concurrent & Latch-free	Larger-Than-Memory	In-Place Updates
§4	In-Memory	✓		✓
§5	Append-Only-Log	✓	✓	
§6	Hybrid-Log	✓	✓	✓

(Faster, SIGMOD'18)

Appendix

- YCSB (Yahoo! Cloud Serving Benchmark)
  - ✓ A de facto standard benchmark for key-value stores
  - ✓ 1) download, 2) build, 3) run workload
    - git clone https://github.com/brianfrankcooper/YCSB.git
    - ./bin/ycsb run rocksdb –s –P workloads/workloada –p rocksdb.dir=[PATH] –p operationcount=[num]

```
root@shin96:/home/shin96/RocksDB_Festival# git clone https://github.com/brianfrankcooper/YCSB.git
Cloning into 'YCSB'...
remote: Enumerating objects: 20648, done.
remote: Total 20648 (delta 0), reused 0 (delta 0), pack-reused 20648
Receiving objects: 100% (20648/20648), 31.68 MiB | 17.69 MiB/s, done.
Resolving deltas: 100% (8016/8016), done.
Checking connectivity... done.
root@shin96:/home/shin96/RocksDB_Festival# ls
rocksdb  YCSB
root@shin96:/home/shin96/RocksDB_Festival# cd YCSB/
root@shin96:/home/shin96/RocksDB_Festival/YCSB# ls
accumulo1.9       cassandra       distribution    googledatastore  LICENSE.txt  pom.xml       s3          zookeeper
aerospike         checkstyle.xml  doc             griddb           maprdb       postgrenosql  scylla
arangodb          cloudspanner    dynamodb        hbase1           maprjsondb   rados         seaweedfs
asynchbase        CONTRIBUTING.md elasticsearch   hbase2           memcached    README.md     solr7
azurecosmos       core            elasticsearch5  ignite           mongodb      redis         tablestore
azuretablestorage couchbase       foundationdb    infinispan       nosqldb      rest          tarantool
bin               couchbase2      geode           jdbc             NOTICE.txt   riak          voltdb
binding-parent    crail           googlebigtable  kudu             orientdb     rocksdb       workloads
root@shin96:/home/shin96/RocksDB_Festival/YCSB#
```

(git clone)

```
root@shin96:/home/shin96/RocksDB_Festival/YCSB# mvn -pl site.ycsb:rocksdb-binding -am clean package
[INFO] Scanning for projects...
[INFO] ------------------------------------------------------------------------
[INFO] Reactor Build Order:
[INFO]
[INFO] YCSB Root
[INFO] Core YCSB
[INFO] Per Datastore Binding descriptor
[INFO] YCSB Datastore Binding Parent
[INFO] RocksDB Java Binding
[INFO]
[INFO] ------------------------------------------------------------------------
[INFO] Building YCSB Root 0.18.0-SNAPSHOT
[INFO] ------------------------------------------------------------------------
[INFO]
[INFO] --- maven-clean-plugin:2.5:clean (default-clean) @ root ---
[INFO] Deleting /home/shin96/RocksDB_Festival/YCSB/target
[INFO]
[INFO] --- maven-enforcer-plugin:3.0.0-M1:enforce (enforce-maven) @ root ---
[INFO]
[INFO] --- maven-checkstyle-plugin:2.16:check (validate) @ root ---
[INFO]
[INFO] ------------------------------------------------------------------------
[INFO] Building Core YCSB 0.18.0-SNAPSHOT
[INFO] ------------------------------------------------------------------------
[INFO]
[INFO] --- maven-clean-plugin:2.5:clean (default-clean) @ core ---
[INFO] Deleting /home/shin96/RocksDB_Festival/YCSB/core/target
[INFO]
[INFO] --- maven-enforcer-plugin:3.0.0-M1:enforce (enforce-maven) @ core ---
[INFO]
[INFO] --- maven-checkstyle-plugin:2.16:check (validate) @ core ---
[INFO]
[INFO] --- maven-resources-plugin:2.6:resources (default-resources) @ core ---
[INFO] Using 'UTF-8' encoding to copy filtered resources.
[INFO] Copying 1 resource
[INFO]
[INFO] --- maven-compiler-plugin:3.7.0:compile (default-compile) @ core ---
[INFO] Changes detected - recompiling the module!
[INFO] Compiling 63 source files to /home/shin96/RocksDB_Festival/YCSB/core/target/classes
[INFO]
[INFO] --- maven-resources-plugin:2.6:testResources (default-testResources) @ core ---
[INFO] Using 'UTF-8' encoding to copy filtered resources.
[INFO] skip non existing resourceDirectory /home/shin96/RocksDB_Festival/YCSB/core/src/test/resources
[INFO]
```

(build with maven)

```
Command line: -db site.ycsb.db.rocksdb.RocksDBClient -s -P workloads/workloada -p rocksdb.dir=/tmp/rocksdb-data-ycsb -t
YCSB Client 0.18.0-SNAPSHOT

Loading workload...
Starting test.
SLF4J: Failed to load class "org.slf4j.impl.StaticLoggerBinder".
SLF4J: Defaulting to no-operation (NOP) logger implementation
SLF4J: See http://www.slf4j.org/codes.html#StaticLoggerBinder for further details.
2021-07-02 22:00:56:630 0 sec: 0 operations; est completion in 0 second
DBWrapper: report latency for each error is false and specific error codes to track for latency are: []
2021-07-02 22:00:56:845 0 sec: 1000 operations; 4366.81 current ops/sec; [READ: Count=521, Max=184, Min=3, Avg=13.95, 90
=28, 99=74, 99.9=140, 99.99=184] [CLEANUP: Count=1, Max=1123, Min=1123, Avg=1123, 90=1123, 99=1123, 99.9=1123, 99.99=112
3] [UPDATE: Count=479, Max=4065, Min=19, Avg=55.03, 90=86, 99=233, 99.9=4065, 99.99=4065]
[OVERALL], RunTime(ms), 229
[OVERALL], Throughput(ops/sec), 4366.812227074236
[TOTAL_GCS_PS_Scavenge], Count, 0
[TOTAL_GC_TIME_PS_Scavenge], Time(ms), 0
[TOTAL_GC_TIME_%_PS_Scavenge], Time(%), 0.0
[TOTAL_GCS_PS_MarkSweep], Count, 0
[TOTAL_GC_TIME_PS_MarkSweep], Time(ms), 0
[TOTAL_GC_TIME_%_PS_MarkSweep], Time(%), 0.0
[TOTAL_GCs], Count, 0
[TOTAL_GC_TIME], Time(ms), 0
[TOTAL_GC_TIME_%], Time(%), 0.0
[READ], Operations, 521
[READ], AverageLatency(us), 13.953934740882918
[READ], MinLatency(us), 3
[READ], MaxLatency(us), 184
[READ], 95thPercentileLatency(us), 41
[READ], 99thPercentileLatency(us), 74
[READ], Return=OK, 521
[CLEANUP], Operations, 1
[CLEANUP], AverageLatency(us), 1123.0
[CLEANUP], MinLatency(us), 1123
[CLEANUP], MaxLatency(us), 1123
[CLEANUP], 95thPercentileLatency(us), 1123
[CLEANUP], 99thPercentileLatency(us), 1123
[UPDATE], Operations, 479
[UPDATE], AverageLatency(us), 55.02713987473904
[UPDATE], MinLatency(us), 19
[UPDATE], MaxLatency(us), 4065
[UPDATE], 95thPercentileLatency(us), 153
[UPDATE], 99thPercentileLatency(us), 233
[UPDATE], Return=OK, 479
root@shin96:/home/shin96/RocksDB_Festival/YCSB#
```

(run workloada)

```
root@shin96:/home/shin96/RocksDB_Festival/YCSB/workloads# ls
tsworkloada  tsworkload_template  workloada  workloadb  workloadc  workloadd  workloade  workloadf  workload_template
root@shin96:/home/shin96/RocksDB_Festival/YCSB/workloads#
```

(type of workloads)