RocksDB Festival

RF5_Team_LayOut

Supported by IITP, StarLab.

August 02, 2021

Minguk Choi, Jungwon Lee, Guangxun shin
koreachoi96@gmail.com, gardenlee960828@gmail.com, guangxun0621@naver.com

Docks

RocksDB Festival

- 1. What is BlobDB?
- 2. BlobDB structure
  - ✓ Write/Read
  - ✓ Blob file/Index format
  - ✓ File size
- 3. BlobDB feature
  - ✓ 1. Feature parity
  - ✓ 2. Consistency
  - ✓ 3. Compaction
  - ✓ 4. Write amplification
  - ✓ 5. Garbage collection
- 4. Next week

Before Start…

History / BlobDB

Revisions

- Updated BlobDB (markdown)
  Itamasi committed 18 days ago
- Updated BlobDB (markdown)
  Itamasi committed on 27 May
- Updated Blob DB (markdown)
  Itamasi committed on 7 Apr 2020

RocksDB features, and required users to adopt a custom API. In 2020, we decided to rearchitect BlobDB from the ground up, taking the lessons learned from WiscKey and the original BlobDB but also drawing inspiration and incorporating ideas from other similar systems. Our goals were to eliminate the above

- BlobDB is
  - ✓ updated at May 27
  - ✓ Similar to Wisckey(2016), but difference exists
  - ✓ Insufficient explanation, studied with codes and comments
- We are not sure…

1. What is BlobDB?

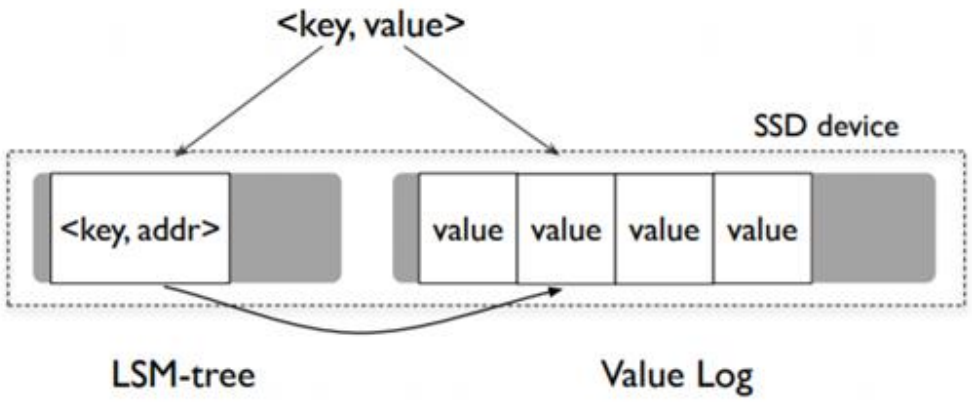
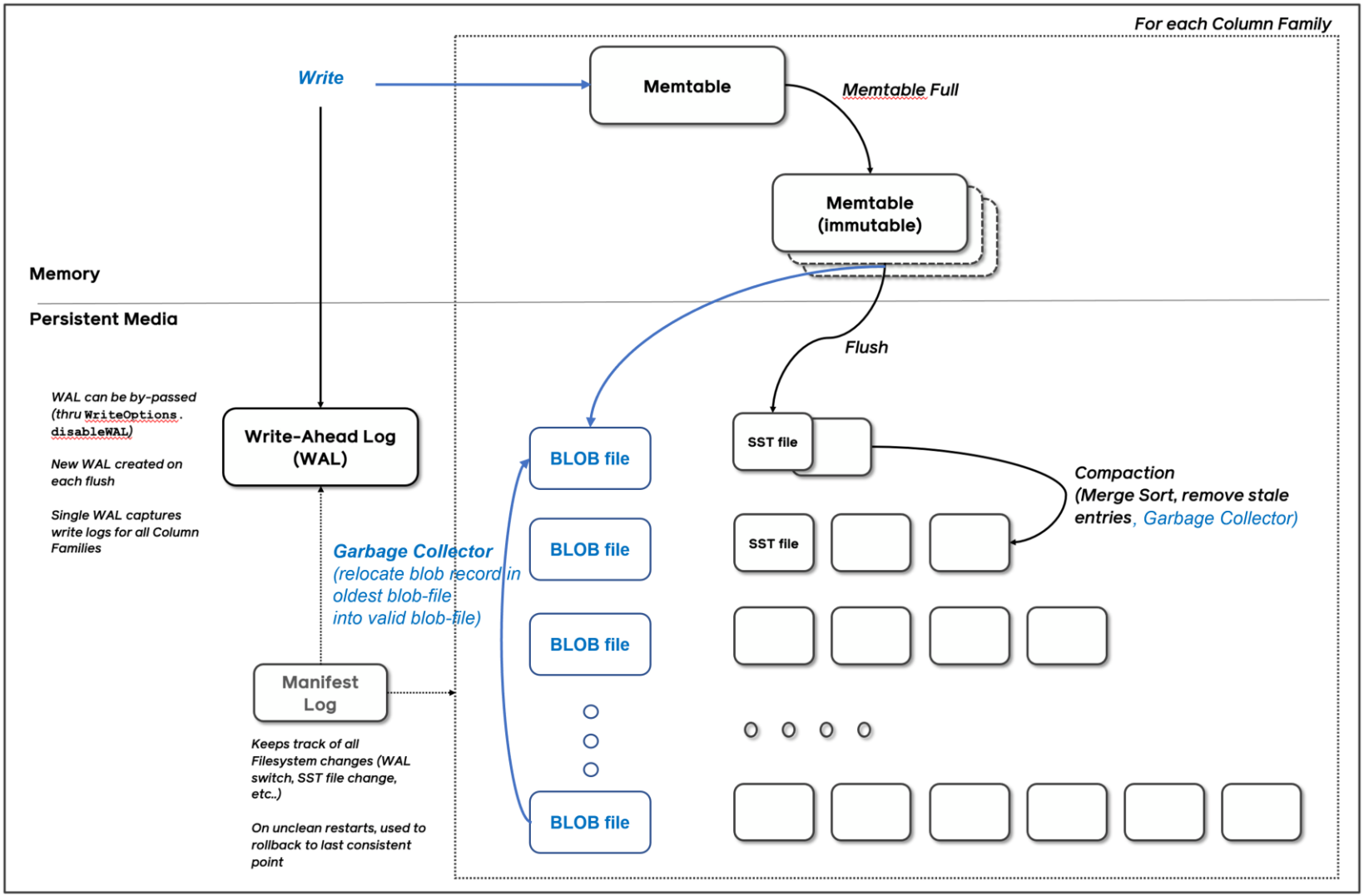


Figure 4: **WiscKey Data Layout on SSD.** *This figure*

- Key – Value seperation
  - ✓ proposed in Wisckey paper, 2016
- By storing
  - ✓ large values in blob file
  - ✓ pointers in LSM tree
- For
  - ✓ Copying values over compaction -> reduce Write amplification
  - ✓ Smaller LSM tree -> better reading, caching

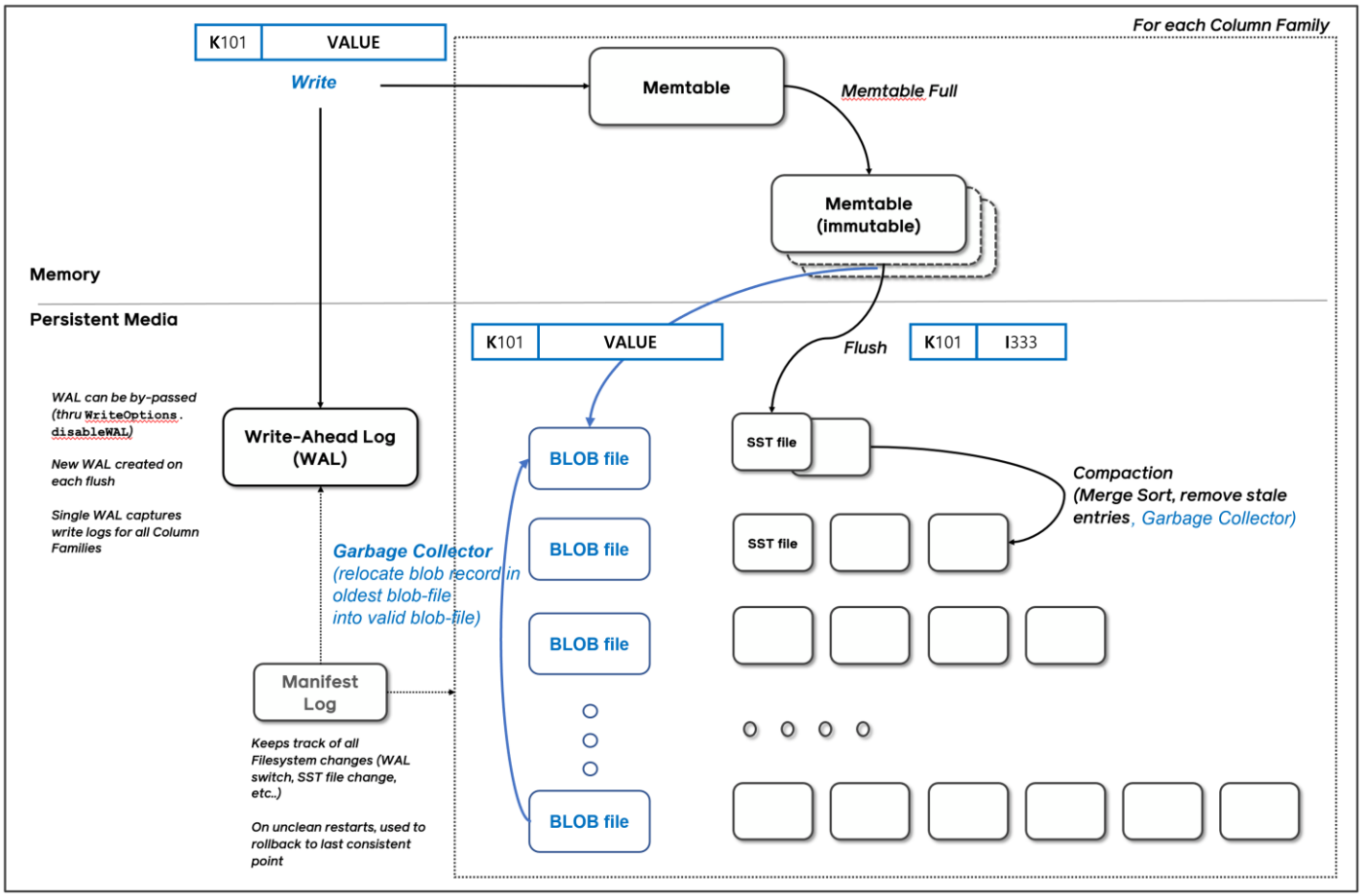
2. BlobDB Structure

- Basic structure



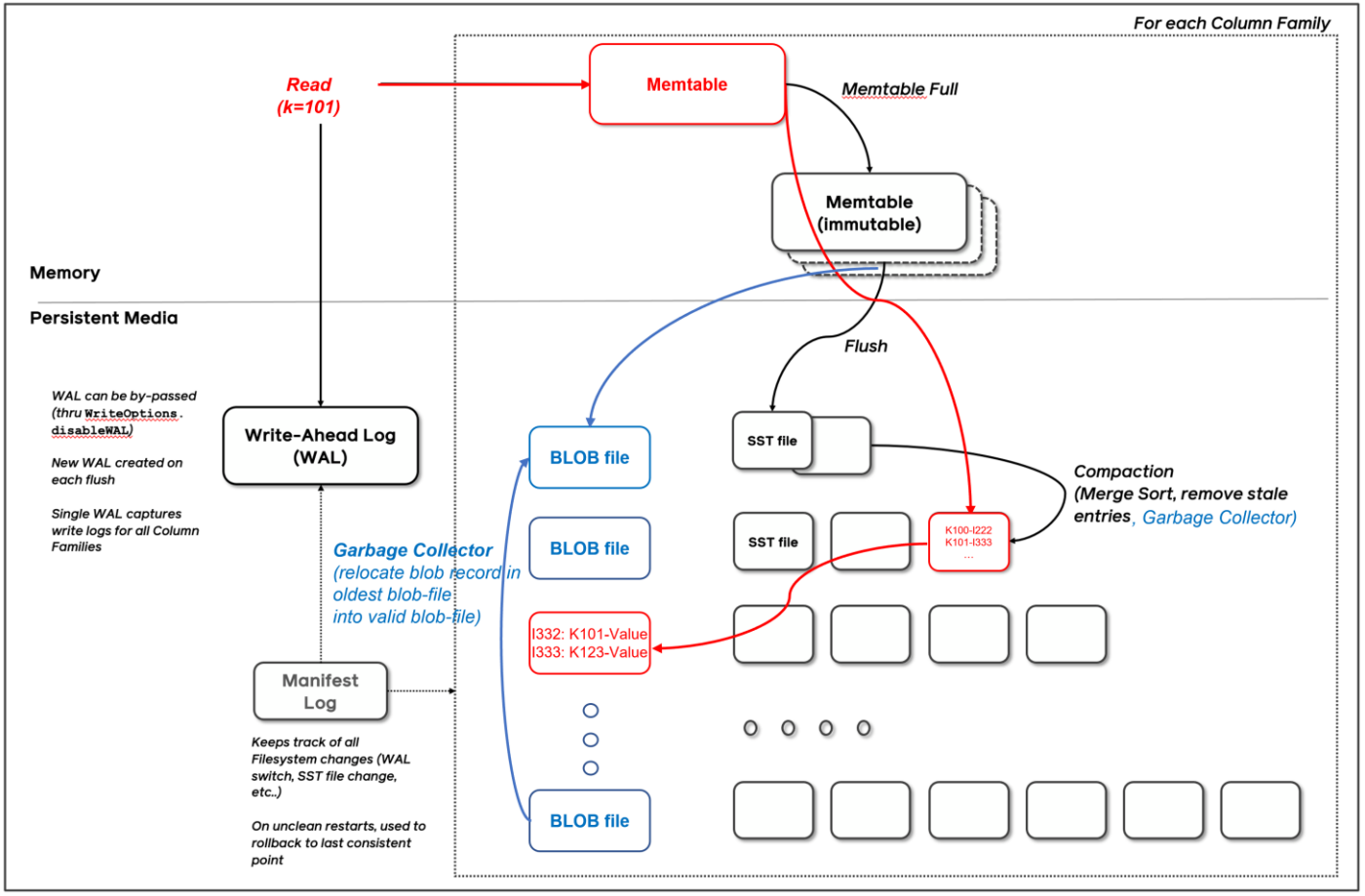
2. BlobDB Structure

- Write
  - ✓ key-index in SST file
  - ✓ key-value in Blob file



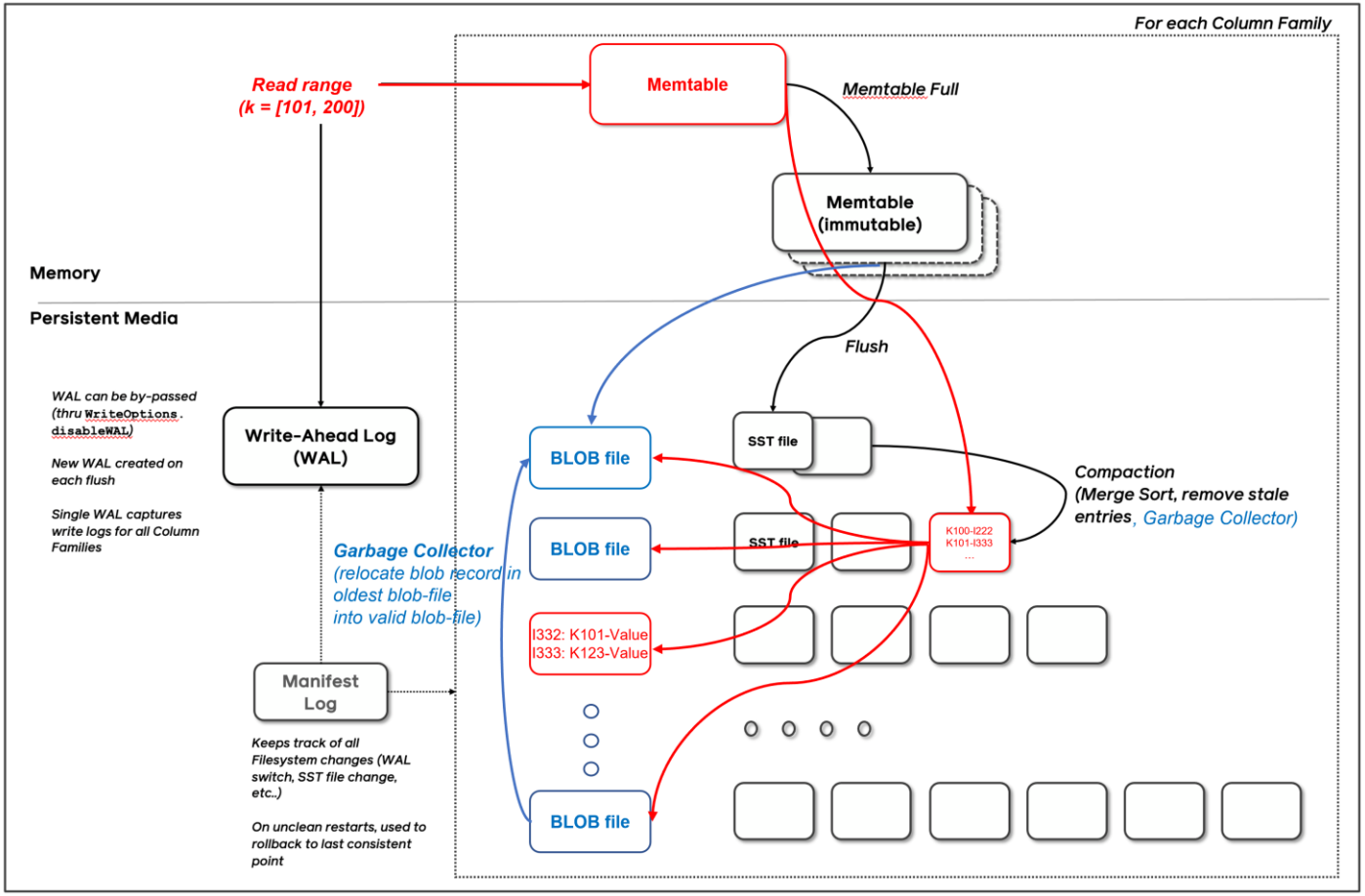
2. BlobDB Structure

- Read
  - ✓ Find Key in Memtable/SST file, Get Blob Index
  - ✓ Go to Blob file, Get value



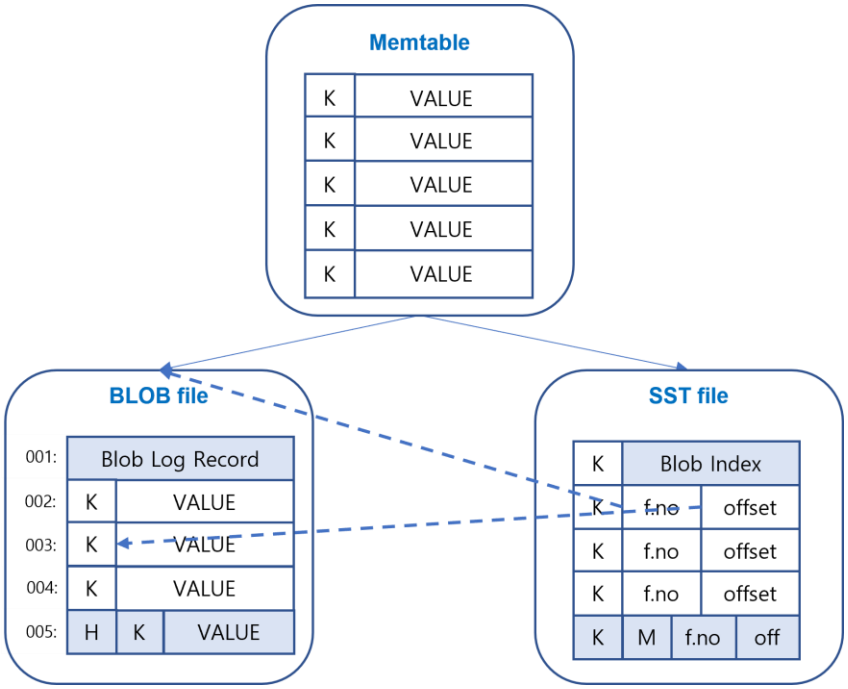
2. BlobDB Structure

- Read Range
  - ✓ Sequential read in SST file, but Random read in Blob file
  - ✓ Use Multi-threading, Internal parallelism of SSD



2. BlobDB Structure

- Blob Index/Record



✓ Blob Index (kBlobTTL)

type (char)	expiration (Variant 64)	file number (Variant 64)	Offset (variant 64)	Size (variant 64)	Compression (char)

- a pointer to the blob and metadata of the blob.
- 3 types: kInlinedTTL/kBlob/kBlobTTL
- points to blob value, not start of blob record
  - Used to Calculate the adjustment, if need to read record Header

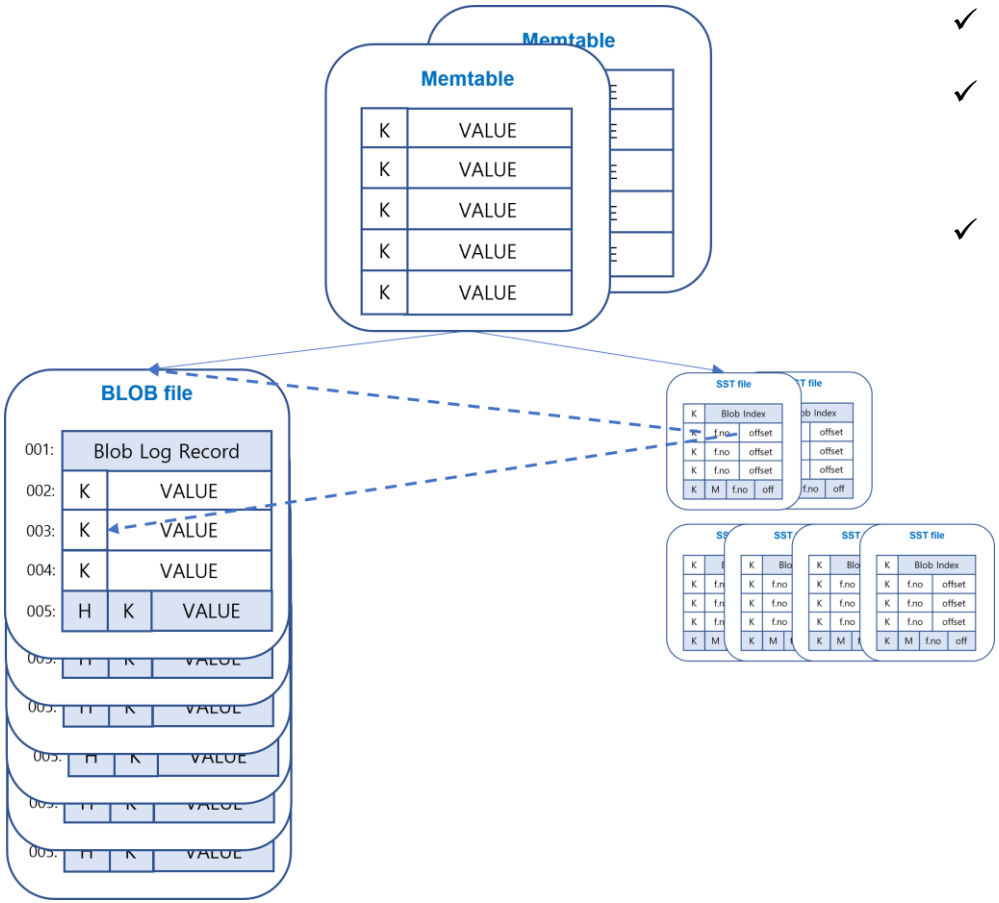
✓ Blob record

key length (Fixed 64)	value length (Fixed 64)	expiration (Fixed 64)	Header CRC (Fixed 32)	Blob CRC (Fixed 32)	Key (key len)	Value (Value len)

- Format: 32bytes header + key + value
- Header CRC
  - checksum of (key_len + val_len + expiration)
- Blob CRC
  - checksum of (key + value)

2. BlobDB Structure

- File size



- ✓ Blob file size ≈ Memtable size
- ✓ Key : Value = SST file size : Blob file size

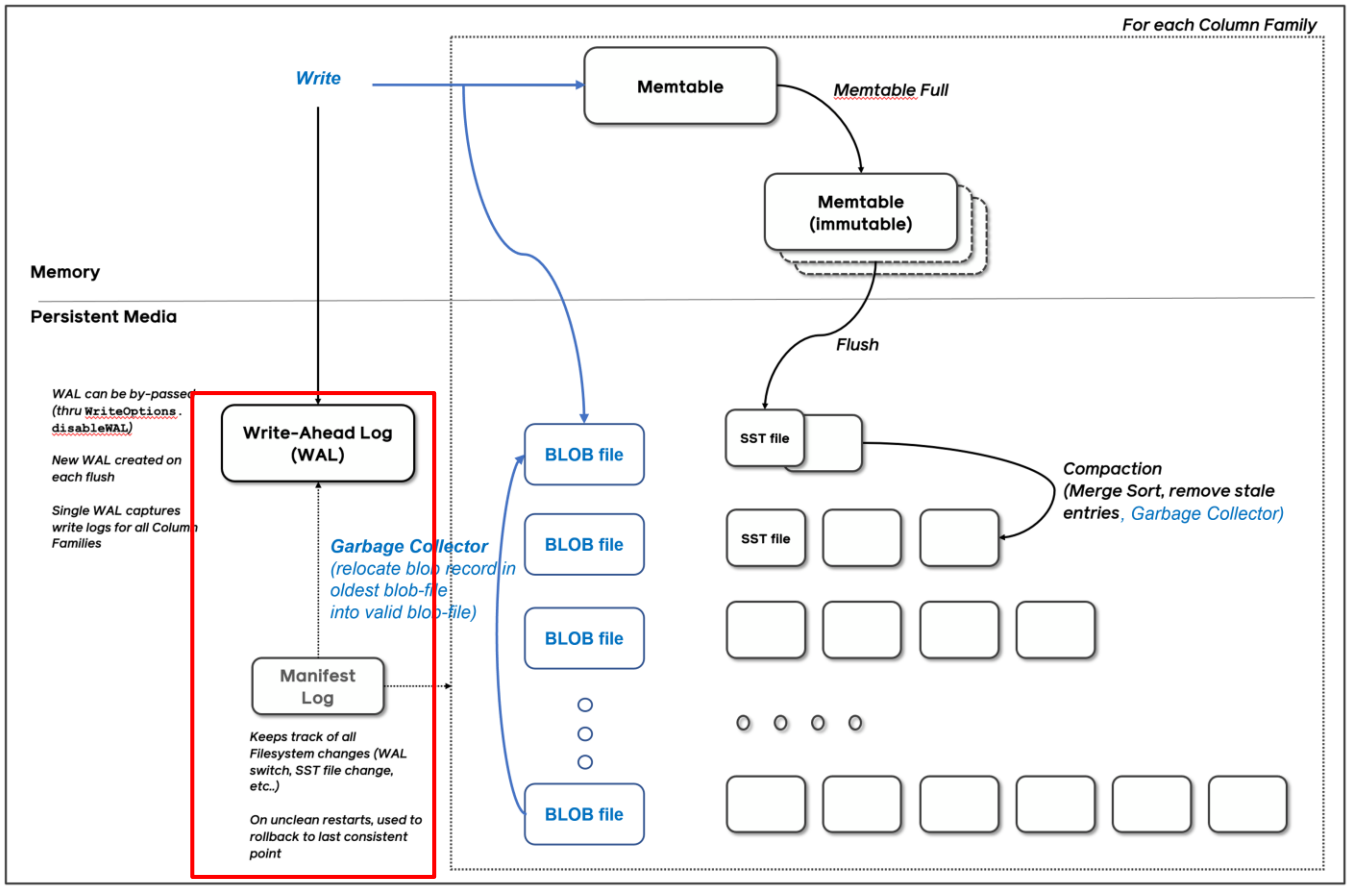
- ✓ Small SST file
  - Key 16B – Value 1KB
    - 100GB -> 2GB
  - low compaction overhead
  - better Caching/In-memory DB

3. BlobDB Feature

- Feature parity
  - ✓ Feature Parity
    - Way more features than original
    - Near feature parity with vanilla RocksDB
    - read/write APIs (~~merge~~), recovery, compression, atomic flush, column families, compaction filters, checkpoints, backup/restore, transactions, per-file checksums, and the SST file manager
  - ✓ Gap(missing)
    - Merge(GetMergeOperands): most important, will be updated soon
    - EventListener, GetLiveFilesMetaData, GetColumnFamilyMetaData
    - secondary instances
    - ingestion of blob files

3. BlobDB feature

- 2. Consistency
  - ✓ WAL, synchronous writes
  - ✓ Can track blob file in MANIFEST like SST File



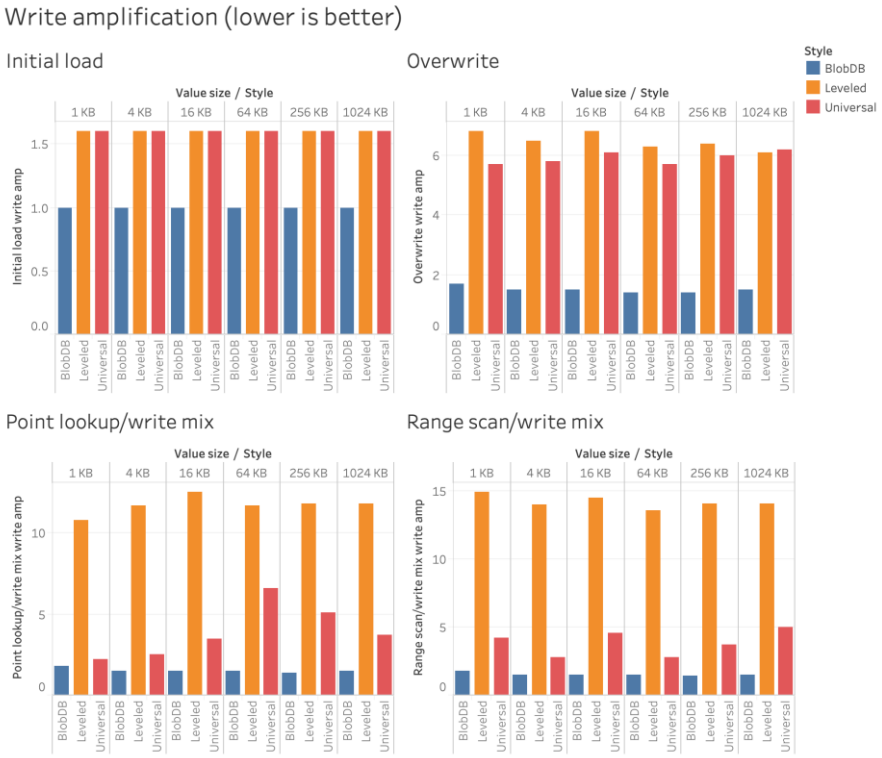
3. BlobDB Feature

- 3. Compaction
  - ✓ RocksDB
    - Universal Compaction: low write amp / high read amp
  - ✓ BlobDB: Leveled compaction
    - Write amplification: Leveled Compaction < Universal Compaction
    - Read performance: Leveled Compaction > Universal Compaction
  - ✓ Compaction filter: Key Optimiztion
    - Make Compaction decision about a key-value solely based on the key,
      -> it is unnecessary to read the value from the blob file
    - CompactionFilter::FilterBlobByKey

4. Amplification

3. Write Amplification

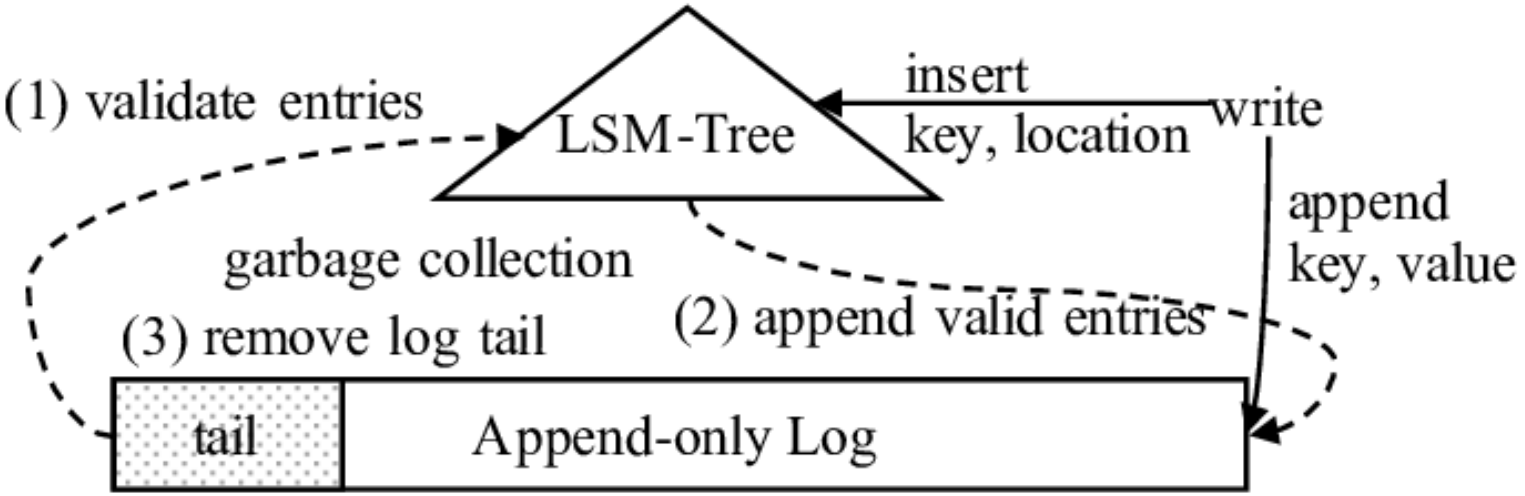
- ✓ Write Amplification = $\frac{\textit{total amount of data written by flushes and compactions}}{\textit{the amount of data written by flushes}}$
- ✓ Way better than vanilla RocksDB
  - by avoid copying the values over and over again during compaction



- Initial load
  – 36% lower than leveled/universal
- Overwrite
  - 75-78% lower than leveled
  - 70-77% lower than universal
- Point lookup/write mix
  - 83-88% lower than leveled
  - 18-77% lower than universal
- Range scan/write mix
  - 88-90% lower than leveled
  - 46-70% lower than universal

3. BlobDB feature

- 4. Garbage Collection
  - ✓ Why?
    - If a key pointing to a blob gets overwritten or deleted, blob becomes unreferenced garbage.
  - ✓ Wisckey

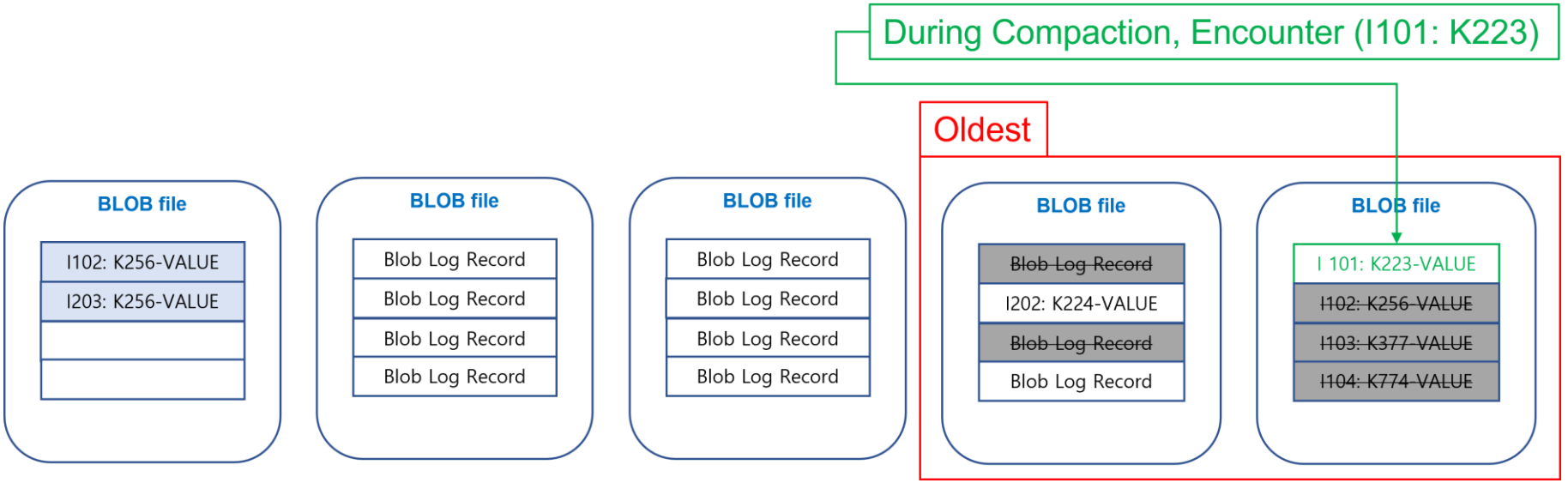


3. BlobDB feature

- 4. Garbage Collection
  - ✓ Why?
    - If a key pointing to a blob gets overwritten or deleted, blob becomes unreferenced garbage.
  - ✓ RocksDB
    - enable_blob_garbage_collection
      - relocate valid blobs from the oldest blob files as they are encountered during compaction.
    - Blob_garbage_collection_age_cutoff
      - Determine which blob files should be considered “old”
      - Trade off between write amplification and space amplification
      - Default 25%

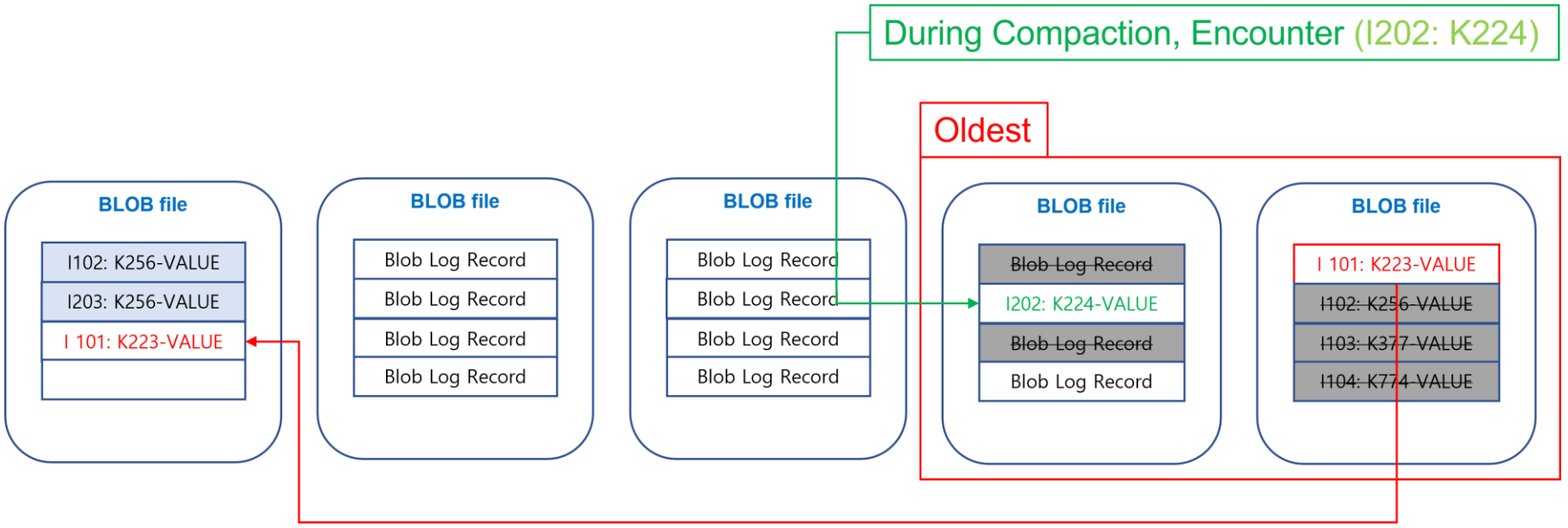
3. BlobDB feature

- 4. Garbage Collection
  - ✓ Mechanism
    - During compaction, if encounter valid blobs in oldest blob files
    - Relocate valid blobs from the oldest file to valid files



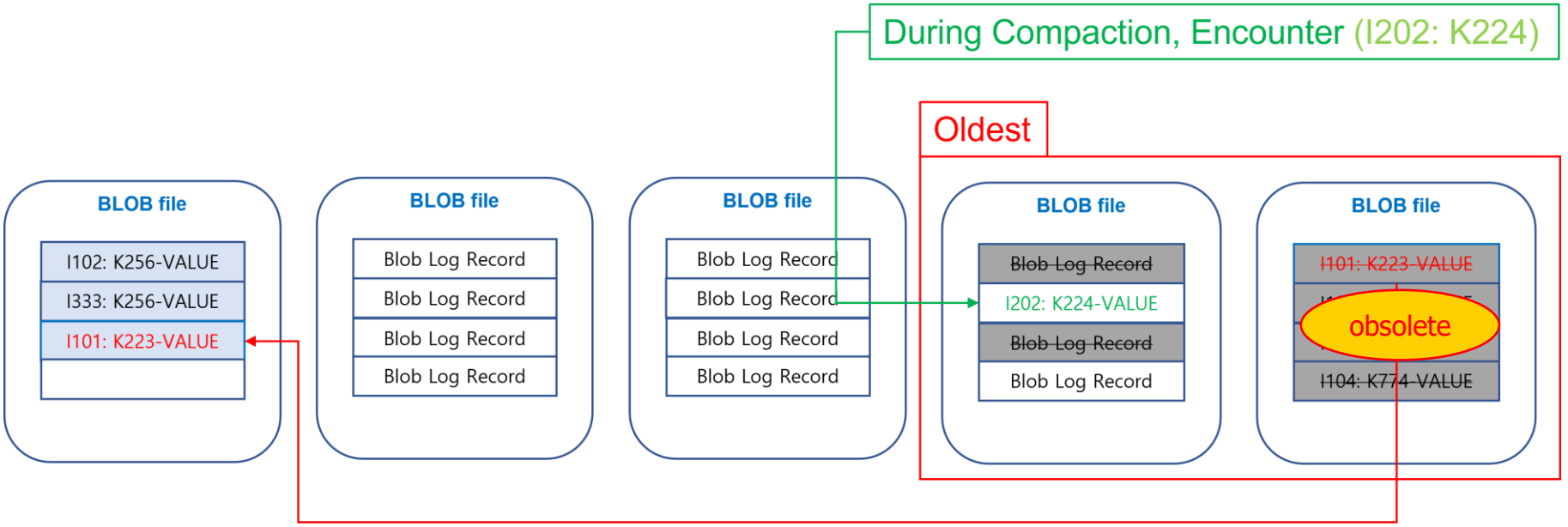
3. BlobDB feature

- 4. Garbage Collection
  - ✓ Mechanism
    - During compaction, if encounter valid blobs in oldest blob files
    - Relocate valid blobs from the oldest file to valid files



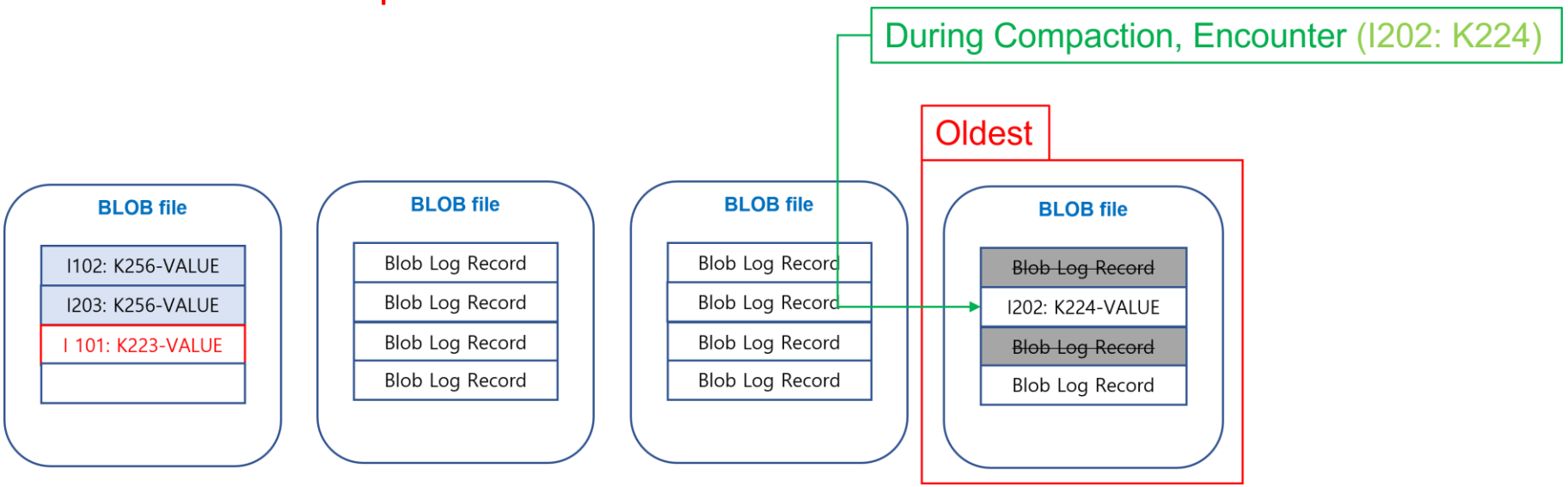
3. BlobDB feature

- 4. Garbage Collection
  - ✓ Mechanism
    - During compaction, if encounter valid blobs in oldest blob files
    - Relocate valid blobs from the oldest file to valid files



3. BlobDB feature

- 4. Garbage Collection
  - ✓ Mechanism
    - During compaction, if encounter valid blobs in oldest blob files
    - Relocate valid blobs from the oldest file to valid files
  - ✓ Need to be optimized



4. Future Work

- Performance analysis (Vanilla RocksDB vs BlobDB)
  - ✓ Read/Read-only/Read&Write/Write Workload
- Experiments on BlobDB options
  - ✓ min_blob_size / blob_file_size / blob_garbage_collection_age_cutoff
  - ✓ write_buffer_size / target_file_size_base / max_bytes_for_level_base
- Study BlobDB Structure/API deeper
  - ✓ Cache, iterator, garbage meter …
- Performance
  - ✓ garbage collection
  - ✓ dedicated cache for blobs
  - ✓ iterator and MultiGet
  - ✓ blob file format

Q & A