Bauhaus-Universität Weimar

Big Data Big talk

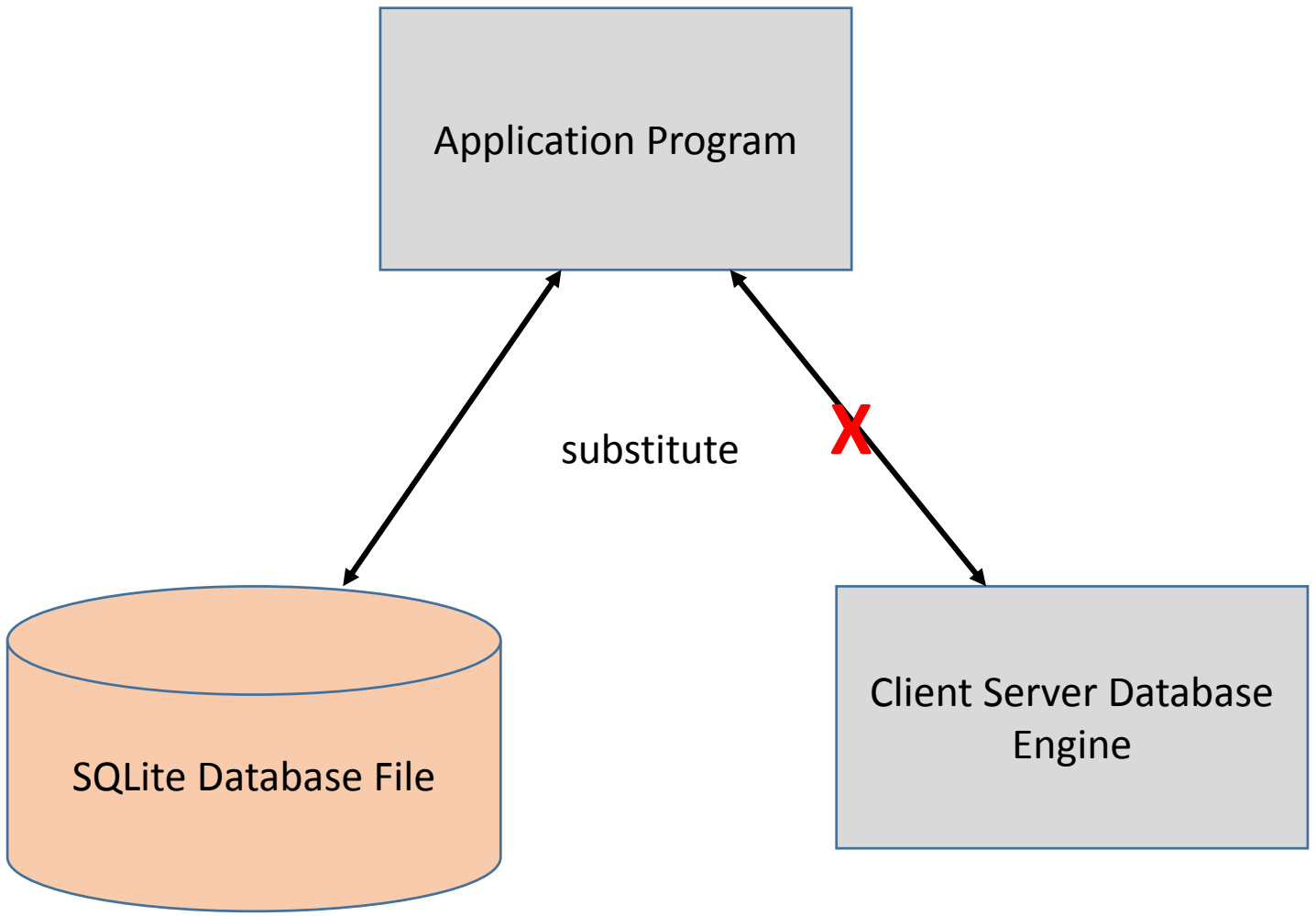
Sqlite

Abiodun Ola

Outline

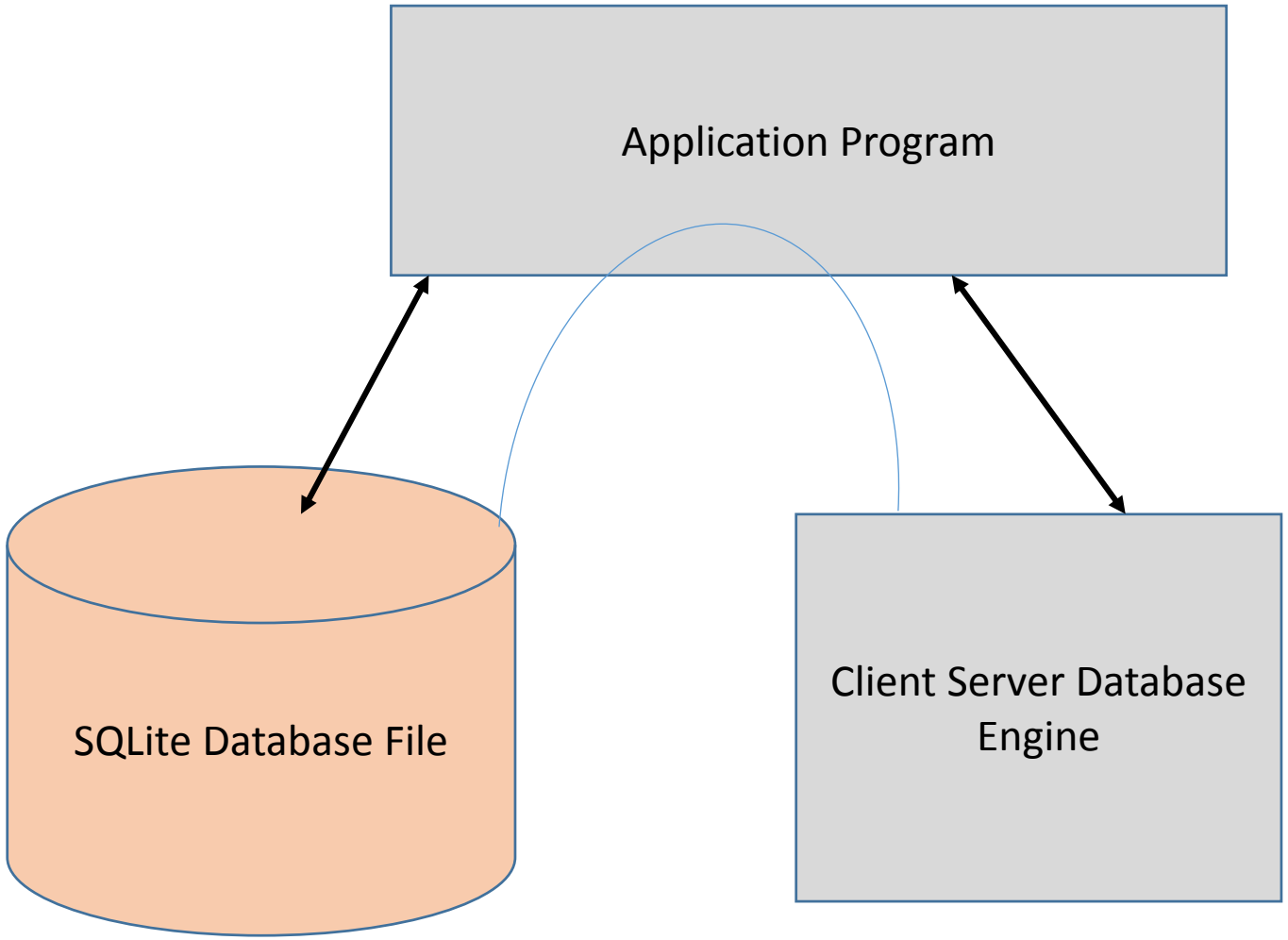
- What is Sqlite
- Introducing Sqlite 4 and LSM tree
- What is LSM data structure
- Comparing Sqlite3 and 4
- Perfomance fact of sqlite4 LSM Api

Embedded in application or hardware

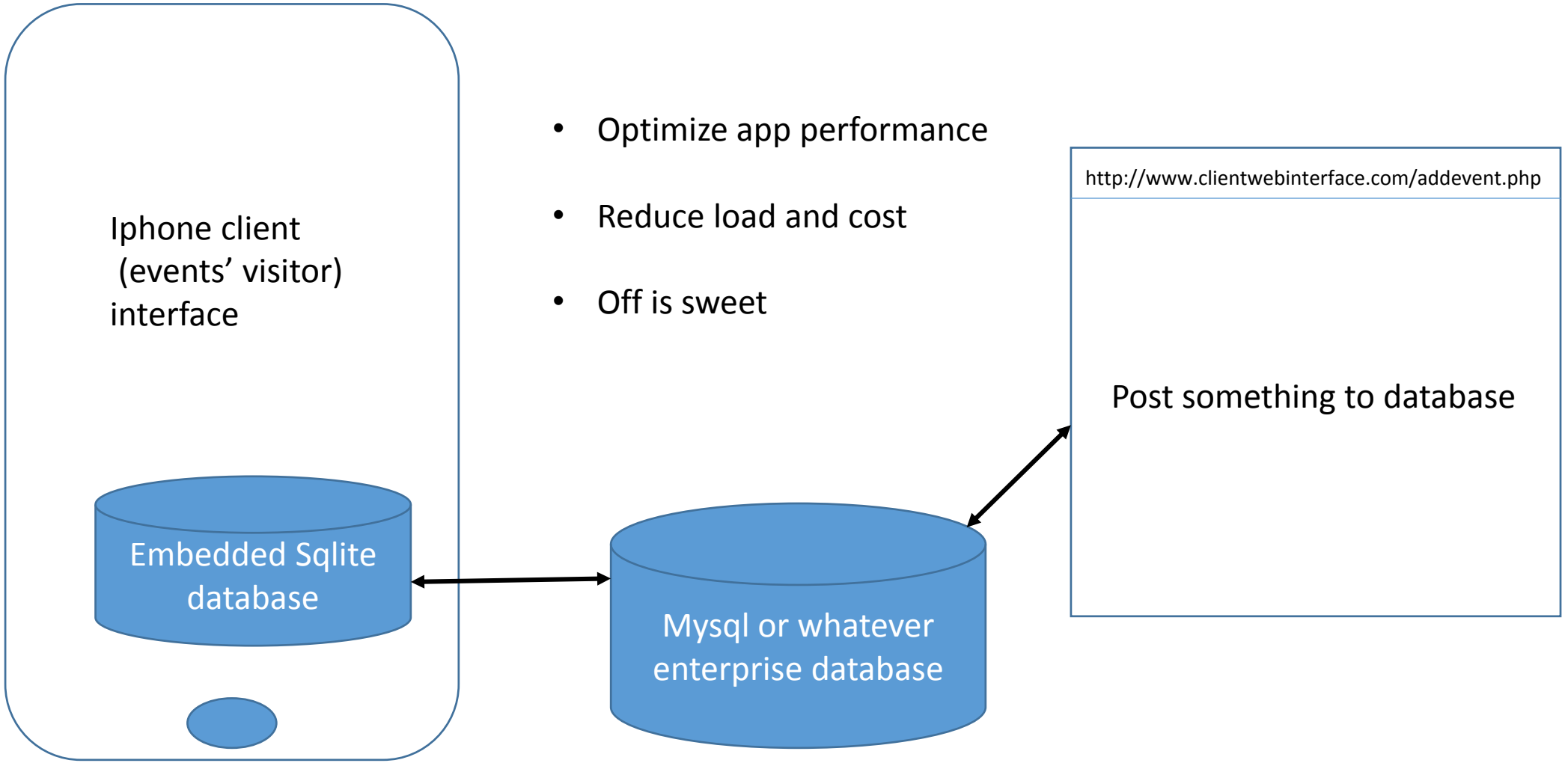


- No configuration it just work
- No scalibity bcos of the size
- Doesnt work on network file system

Why embedded?
Local database cache



MY usage



- Optimize app performance
- Reduce load and cost
- Off is sweet

Sql simpliefied

- Download it n use it in almost all available programming language
- No password, No configuration, no space allocation
- Common sql command

  INSERT DELETE CREATE

Am I alone ...who else find sqlite cool ?

Iphone, Ipad, safari, apple mail .. Ma
functio
MaC

SQLite comes bundled with the Python programming language since Python 2.5.

Chat

TM

Microsoft AIRBUS

Google

TOSHIBA

…ok, Great but we want a nosql database

What is Sqlite4 ?

Self contained

Compact

Just like sqlite3

Zero- administration

ACID

3 vs 4

- SQLite4 is an alternative, not a replacement, for SQLite3

Sqlite 4	Sqlite 3
stores all content, from all tables and all indices, in a single keyspace.	required a separate keyspace for each table and each index.
it requires the storage engine to sort keys is lexicographical order	uses a very complex comparison function to determine the record storage order.

Storage Engines

- SQLite4 uses a key/value storage engine
- The storage engine is pluggable; it can be changed out at runtime
- SQLite4 comes with two built-in storage engines. A log-structured merge-tree (LSM) storage engine is used for persistent on-disk databases and an in-memory binary tree storage engine is used for TEMP databases.

Adding a new storage engine

```
int storageEngineFactor( sqlite4_env *pEnv, /* The run-time environment */
sqlite4_kv_store **ppResult, /* OUT: storage engine object written here */
const char *zName, /* Name of the database file */
 unsigned flags /* Flags */ );
```

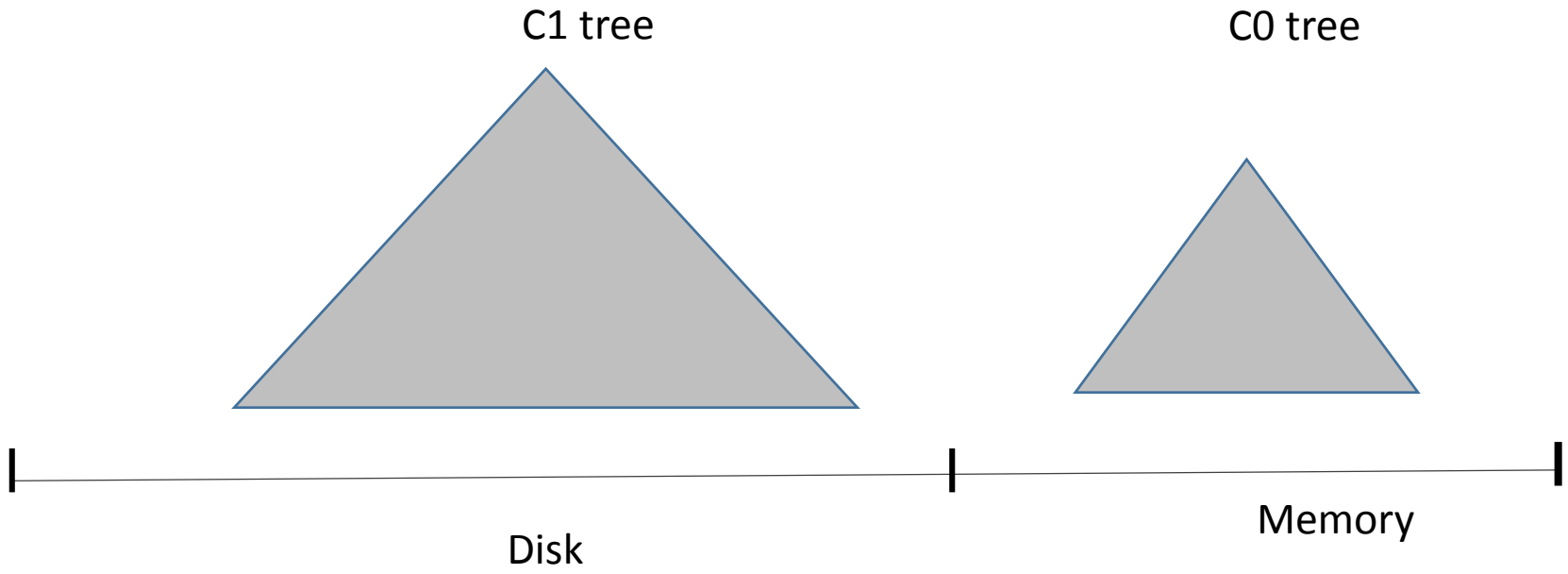
http://sqlite.org/src4/doc/trunk/www/storage.wiki

Log- structure Merge tree (LSM tree)

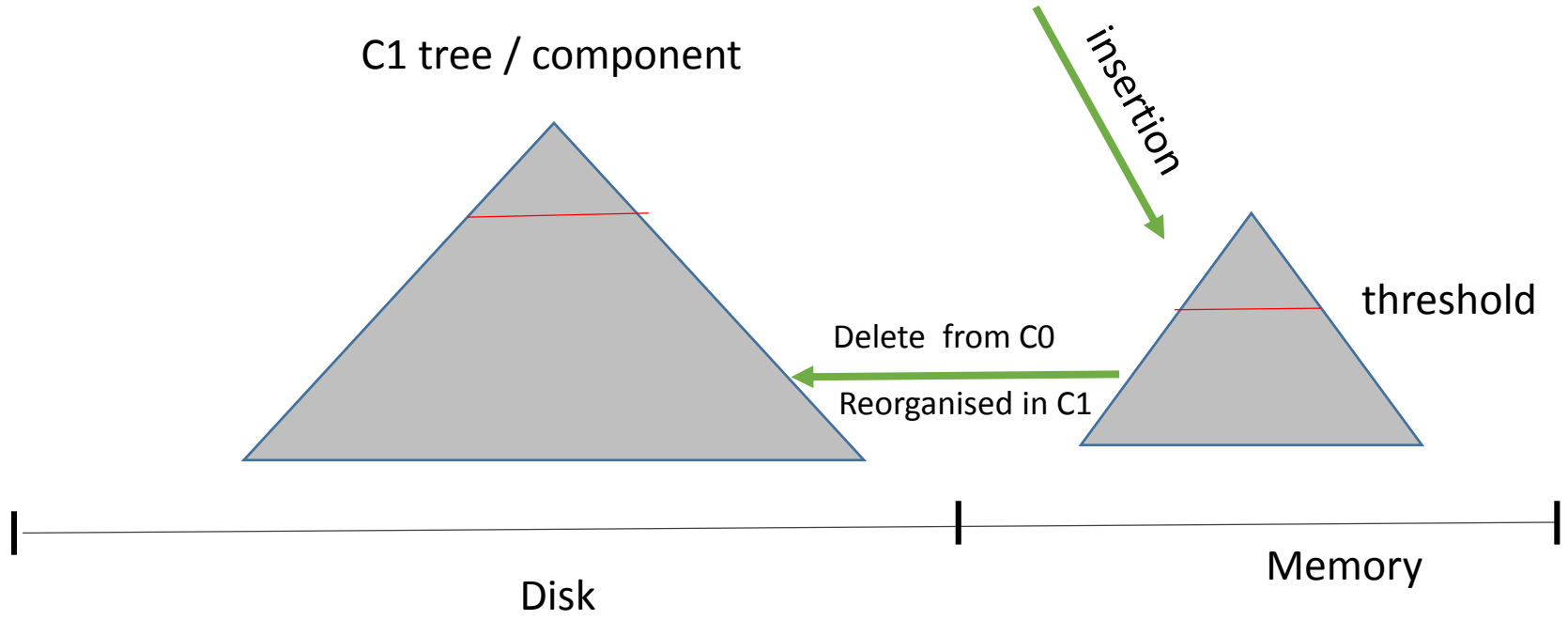
- Hierachical key/value index
- Built through incremental merging of similar sized subindex
- Efficient for read and write

LSM tree

- An LSM-tree is composed of two or more tree-like component data structures.
- In its simplest form .....



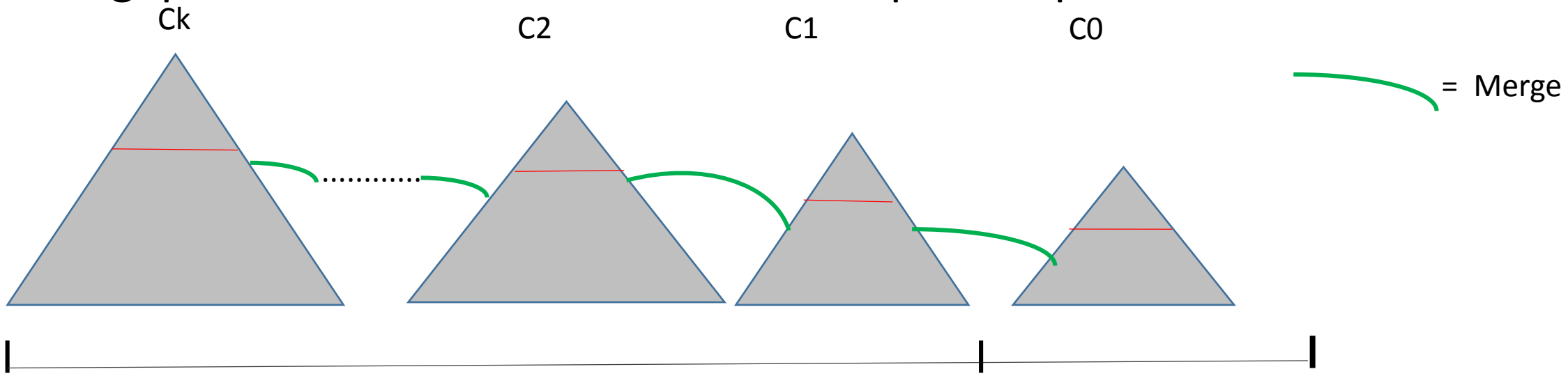
How it works


C1 tree / component
insertion
threshold
Delete from C0
Reorganised in C1
Disk
Memory

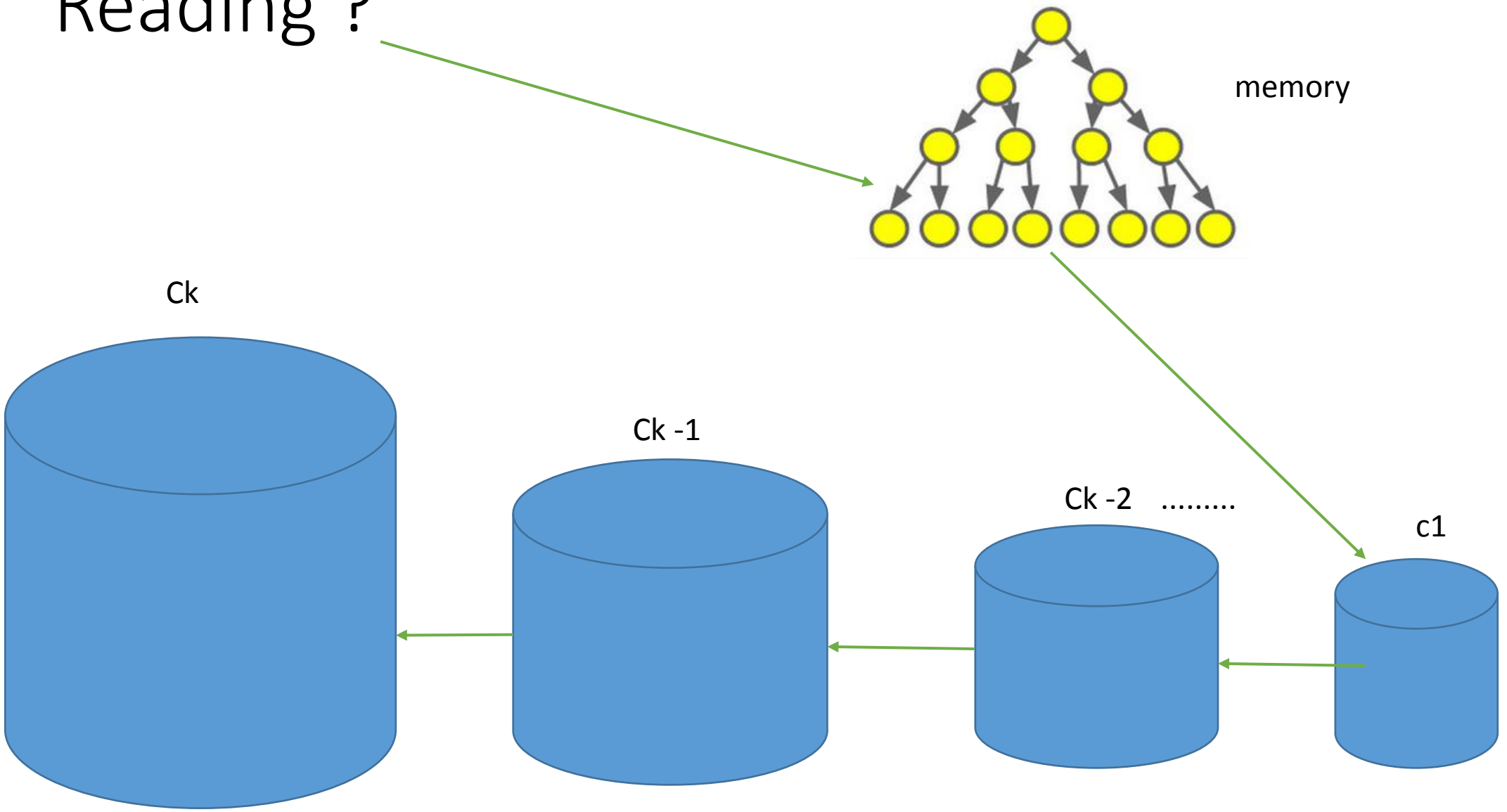
The ”M” is not silent

- The letter M in LSM stand for merge (Log –Structure- Merge tree)

- Under pressure from inserts, there are asynchronous rolling mergeprocesses in train between all component pairs



Reading ?



Quick storage fact

- Level 0 is kept in main memory, and might be represented using a tree.
- The on-disk data is organized into sorted runs of data.
- Each run contains data sorted by the index key. A run can be represented on disk as a single file, or alternatively as a collection of files with nonoverlapping key ranges
- To perform a query on a particular key to get its associated value, one must search in the Level 0 tree, as well as each run.
- When an application writes to the database, the new data is written to the in-memory tree. Once the in-memory tree has grown large enough, its contents are written into the database file as a new sorted run. To reduce the number of sorted runs in the database file, chronologically adjacent sorted runs may be merged together into a single run, either automatically or on demand.
- HBase, LevelDB, SQLite4, RocksDB and Apache Cassandraa.

Lsm in Sqlite4

- LSM is an embedded database library for key-value data
- part of the SQLite4 library. #include lsm.h to access the api
- Both keys and values are specified and stored as byte arrays. Duplicate keys are not supported.
- Supports;
  - Writing a new key and value into the database.
  - Deleting an existing key from the database.
  - Deleting a range of keys from the database.
  - Querying the database for a specific key.
  - Iterating through a range of database keys (either forwards or backwards).

Lsm data storage

- Data are stored in 3 distinct data structures:
  - Shared memory region (in memory tree)
  - Log file (for Robustness)
  - Database file

Performance trade off

- LSM uses a different data structure that makes the following performance tradeoffs relative to a b-tree:
- A very large percentage of the disk sectors modified when writing to the database are contiguous. Additionally, in many cases the total number of sectors written to disk is reduced. This makes writing to an LSM database faster than the equivalent b-tree.
- LSM databases do not suffer from fragmentation to the same degree as b-trees. This means that the performance of large range queries does not degrade as the database is updated as it may with a b-tree.
- Under some circumstances searching an LSM database for a given key will involve examining more disk sectors than it would with a b-tree. In terms of disk sectors accessed when searching a database of size N, both b-trees and LSM provide O(log(N)) efficiency, but the base of the logarithm is generally larger for a b-tree than for LSM.
- In other words, writing to an LSM database should be very fast and scanning through large ranges of keys should also perform well, but searching the database for specific keys may be slightly slower than when using a b-tree based system. Additionally, avoiding random writes in favour of largely contiguous updates (as LSM does) can significantly reduce the wear on SSD or flash memory devices.

Writing

```
rc = lsm_insert(db, "a", 1, "one", 3);

rc = lsm_delete(db, "a", 1);

rc = lsm_delete_range(db, "c", 1, "f", 1);
```

returns LSM_OK (0) if successful, or an LSM error code

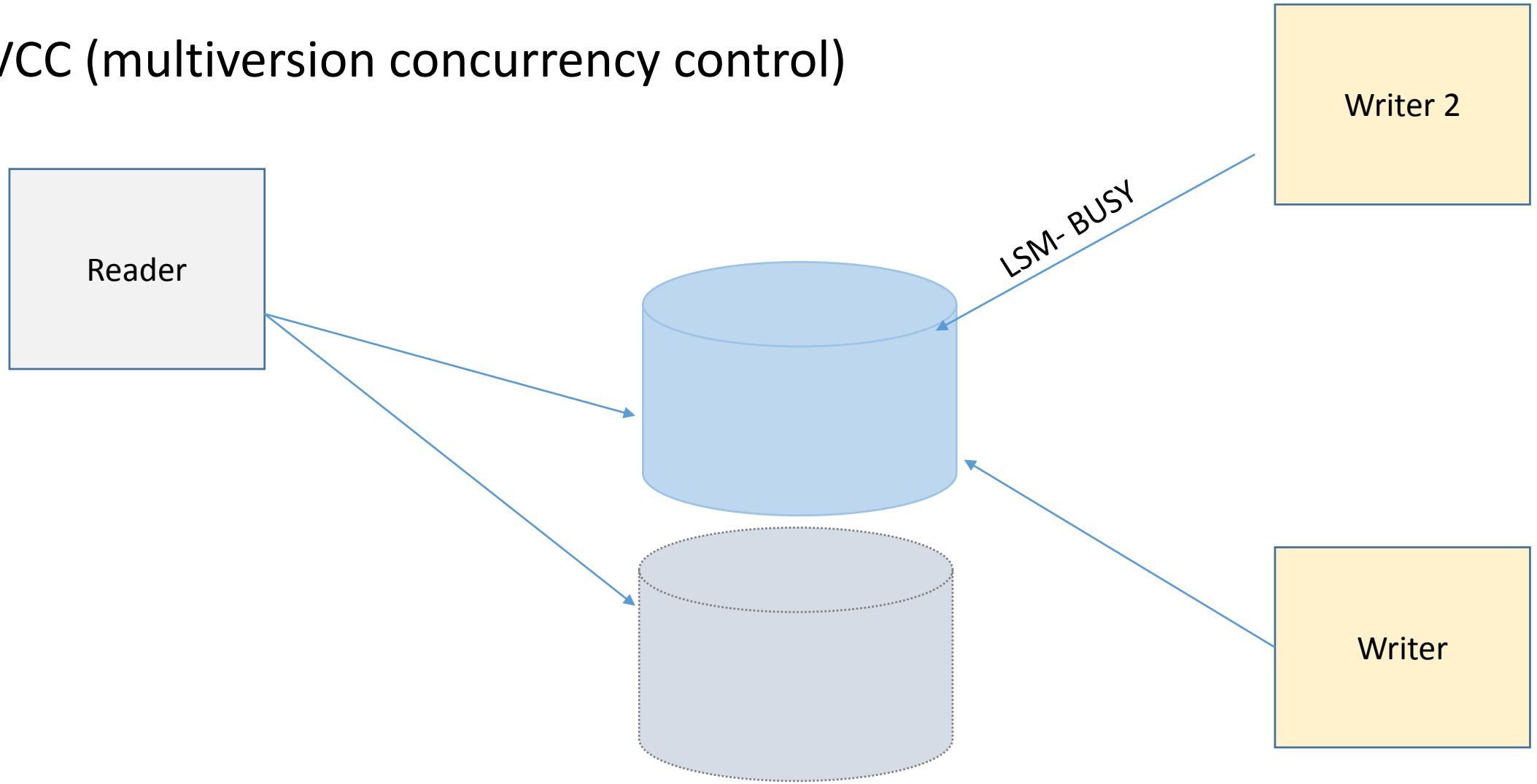
Read via cursor

```
lsm_csr *csr;

rc = lsm_csr_open(db, &csr);

lsm_csr_close(csr);
```

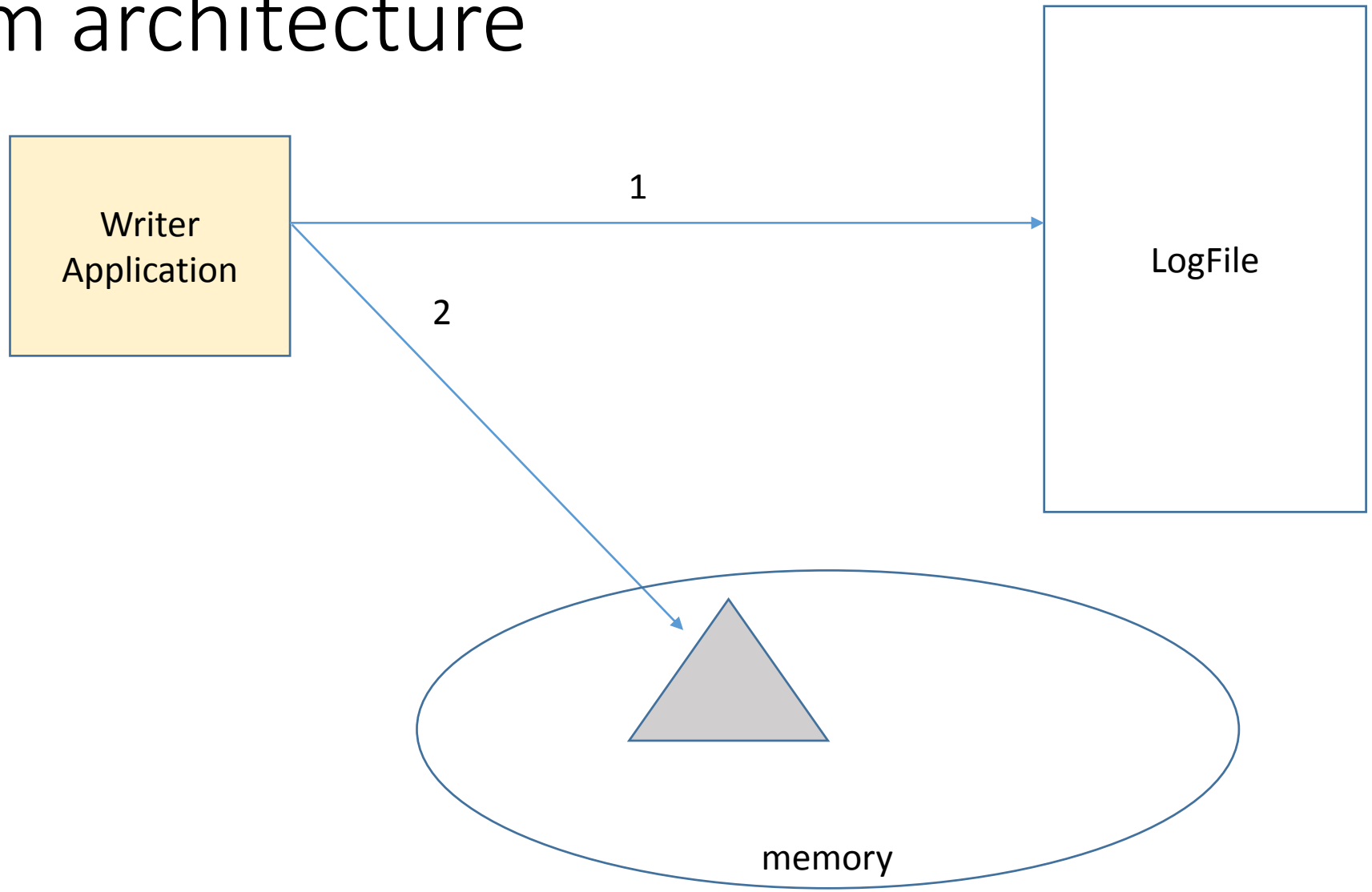
lsm_csr_seek() - move the cursor to point to a nominated database key.
lsm_csr_first() - move the cursor to point to the first entry in the database (the one with the smallest key).
lsm_csr_last() - move the cursor to point to the last entry in the database (the one with the largest key).
lsm_csr_next() - move the cursor to point to the next entry in the the database.
lsm_csr_prev() - move the cursor to point to the previous entry in the database.
Once a cursor has been positioned, it supports the following functions for retrieving the details of the current entry:
lsm_csr_valid() - determine whether or not the cursor currently points to a valid entry.
lsm_csr_key() - retrieve the key associated with the database entry the cursor points to.
lsm_csr_value() - retrieve the value associated with the database entry the cursor points to.
lsm_csr_cmp() - compare a key supplied by the application with the key associated with the entry the cursor points to.

Read Write conflict solve with MVCC

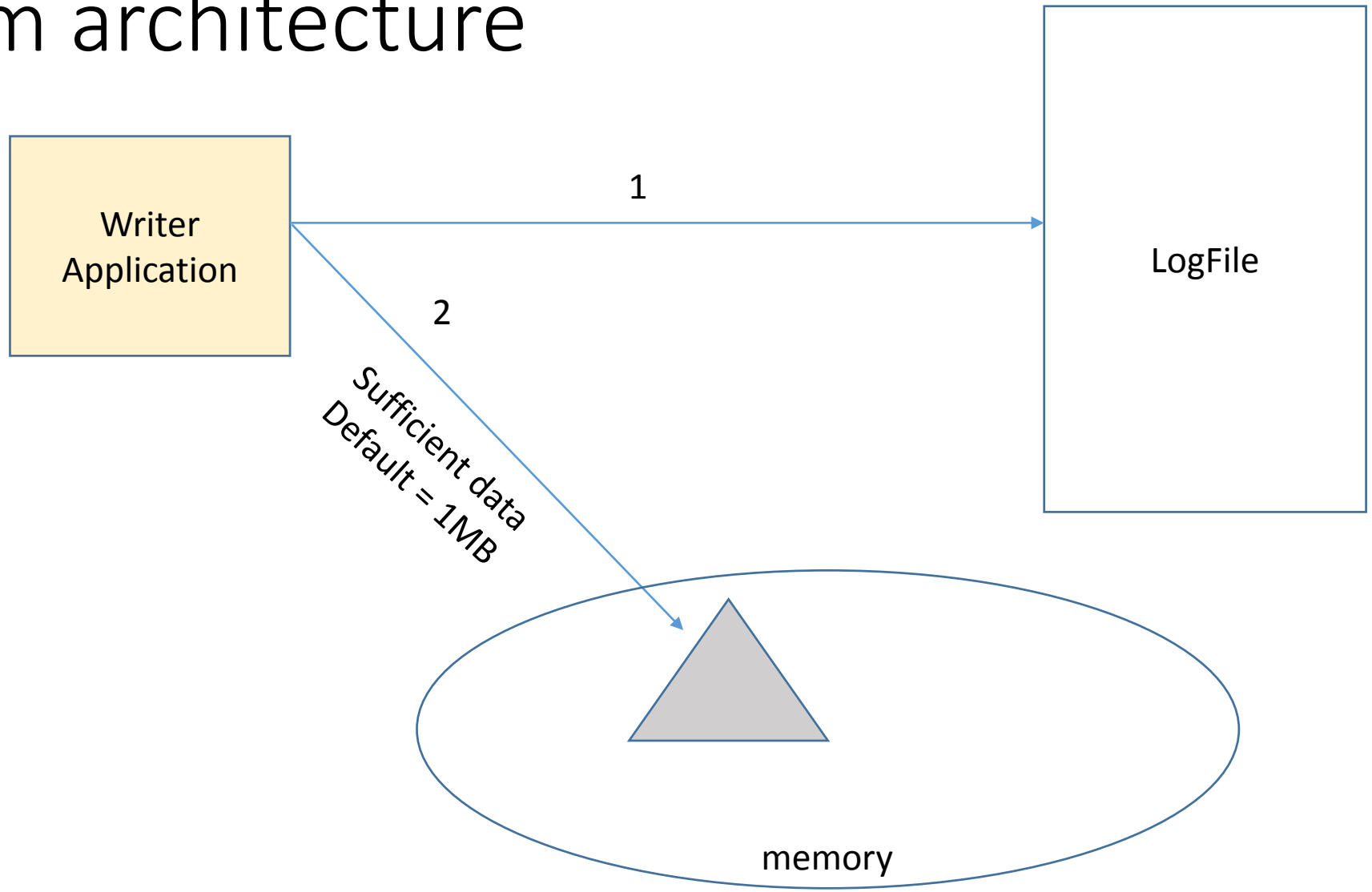
- MVCC (multiversion concurrency control)



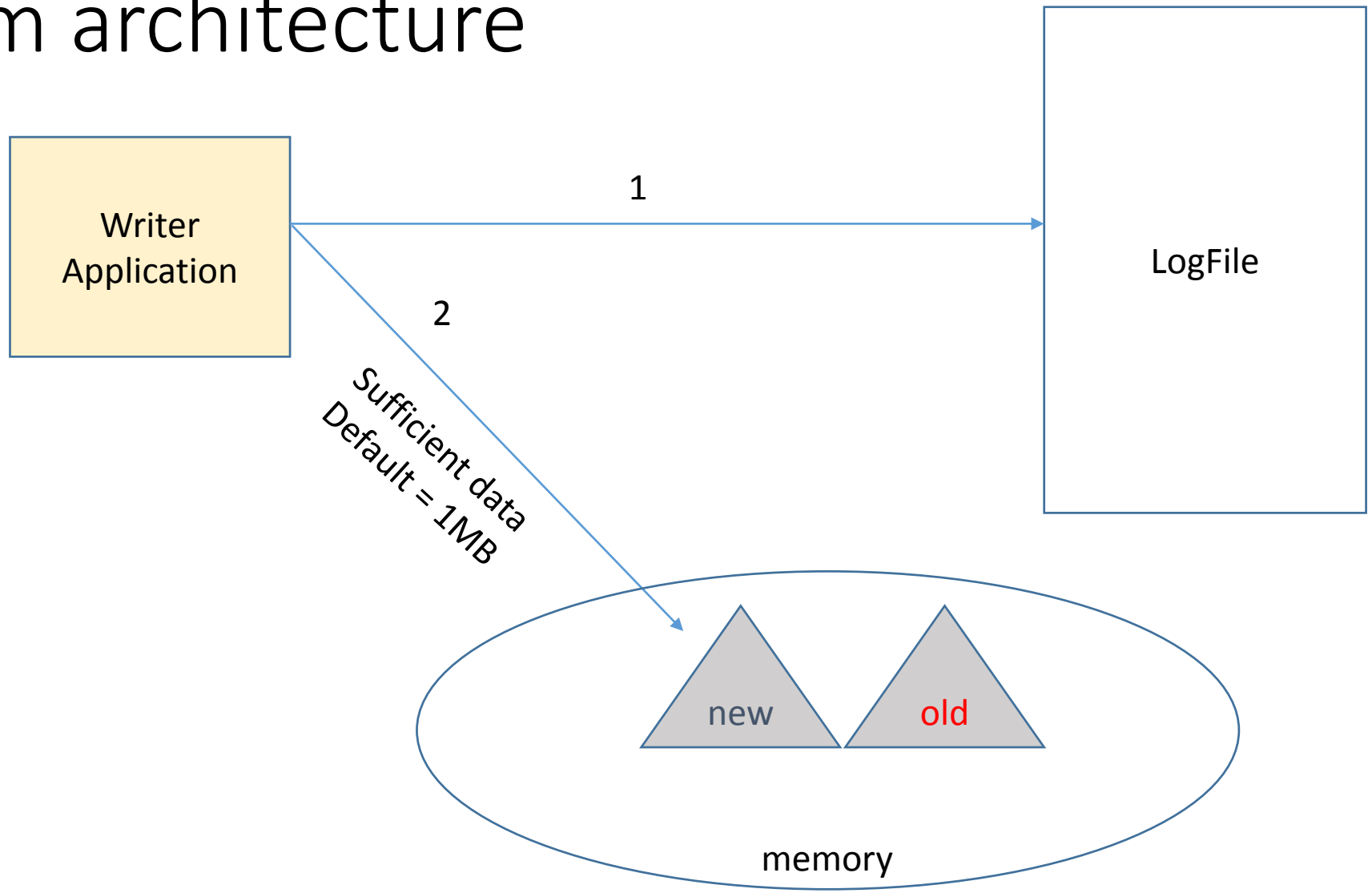
Lsm architecture



Lsm architecture

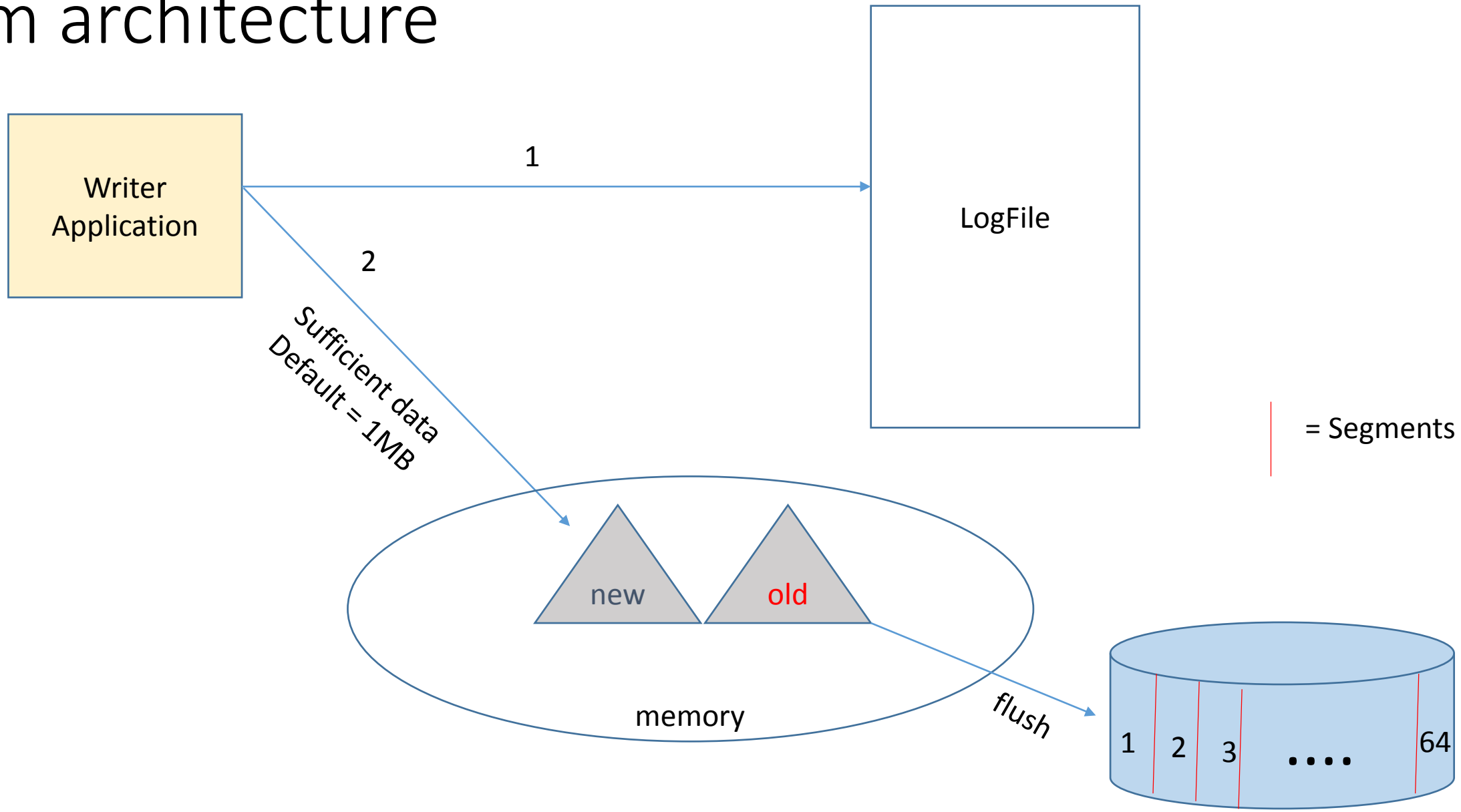


Lsm architecture



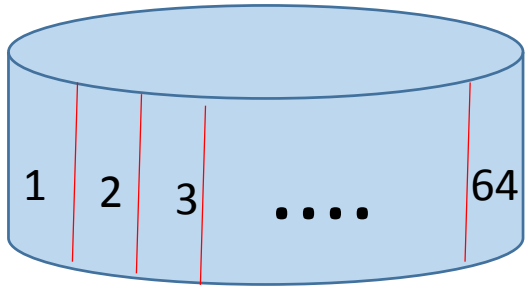
Old tree are immutable , there can be at most 1 of such in memory

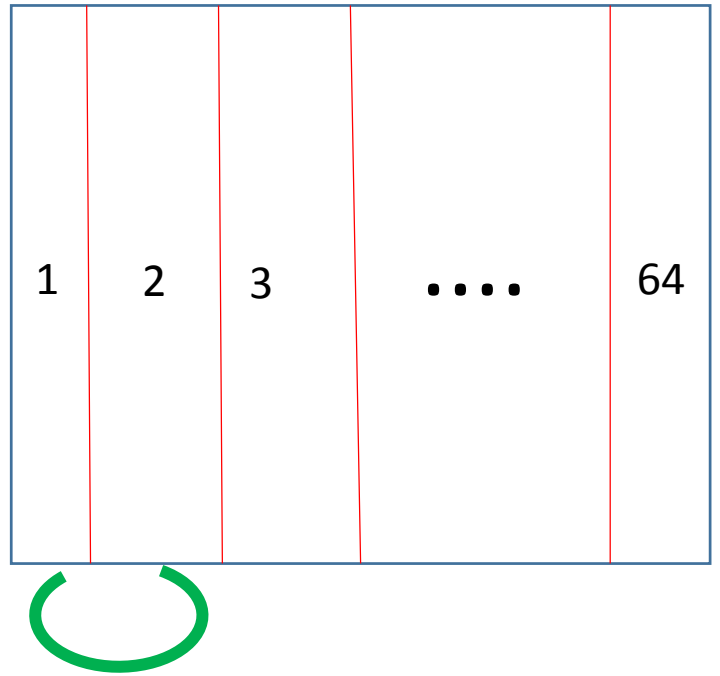
Lsm architecture



Segments merging

A database segment is an immutable b-tree structure stored within the database file





Immutable segment can be merged to free up space

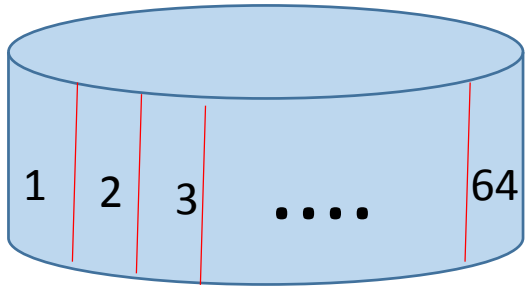
importance

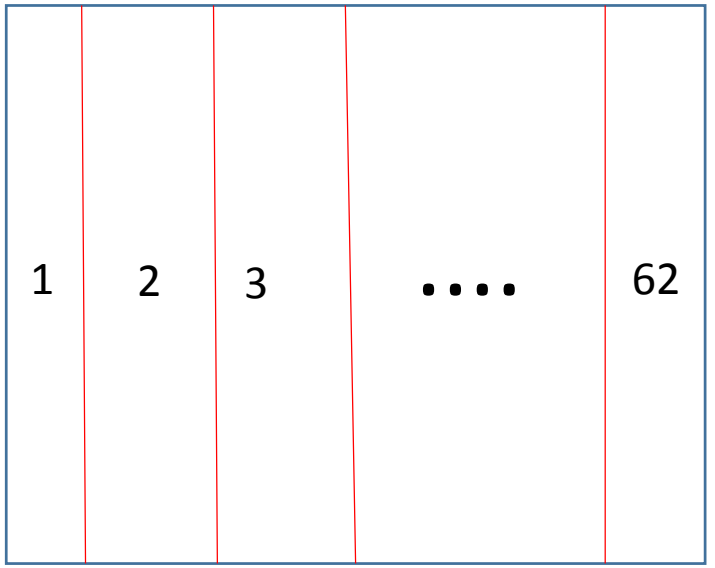
The speed of database read operations is largely determined by the number of segments in the database file

Checkpointing after “working”

After modificaton

i.e flushing or merging segment





updating the database file header and syncing the contents of the database file to disk.

Quickie on check-pointing

- without checkpoints the system will function, but both the log and database files will grow indefinitely as the database is modified

- Additionally, if a crash or power failure occurs, the next client to open the database file has to process all data written to the log file since the most recent checkpoint.

- If checkpoints are performed infrequently, this can be a time consuming exercise.

Data Durability

- LSM_SAFETY_OFF
- LSM_SAFETY_NORMAL
- LSM_SAFETY_FULL

The lock and the Key

Starting :

The Writer

The worker

The Checkpointer

The key and the Lock

- The Writer
  - required to modify the contents of the in-memory tree. Including marking an in-memory tree as "old" and starting a new live tree. It is also required to write to the log file
- The worker
  - required to write to segments within the database file. Either when merging two or more existing segments within the database, or when flushing an in-memory tree to disk to create a new segment.
- The Checkpoint
  - required to write to the database file header.

SQLITE4 LSM Benchmarks

The test uses a single client database connection. It begins with an empty database. The test runs for 200 iterations. Each iteration, the client:

- Inserts **50,000 new key-value pairs** into the database. Each key is a pseudo-randomly generated 12-byte blob. Each value is a pseudo-randomly generated 100 byte blob. Each insert operation is run in a separate implicit transaction.

- Runs 50,000 fetch queries to retrieve arbitrarily selected entries from the database (all searches are hits - the queried keys are selected from the set of keys that are present in the db).

Hardware

- Final db size = approx. 1.2gb
- 3.3GHz dual-core PC with 4GB of RAM
- file-system is ext4 on a 7200rpm HDD.

Measurement

- Insert and fetch duration (in operation per second)

- A result of 100,000 operation per second means it took 0.5 (30 secs) to insert or fecth 50,000 keys

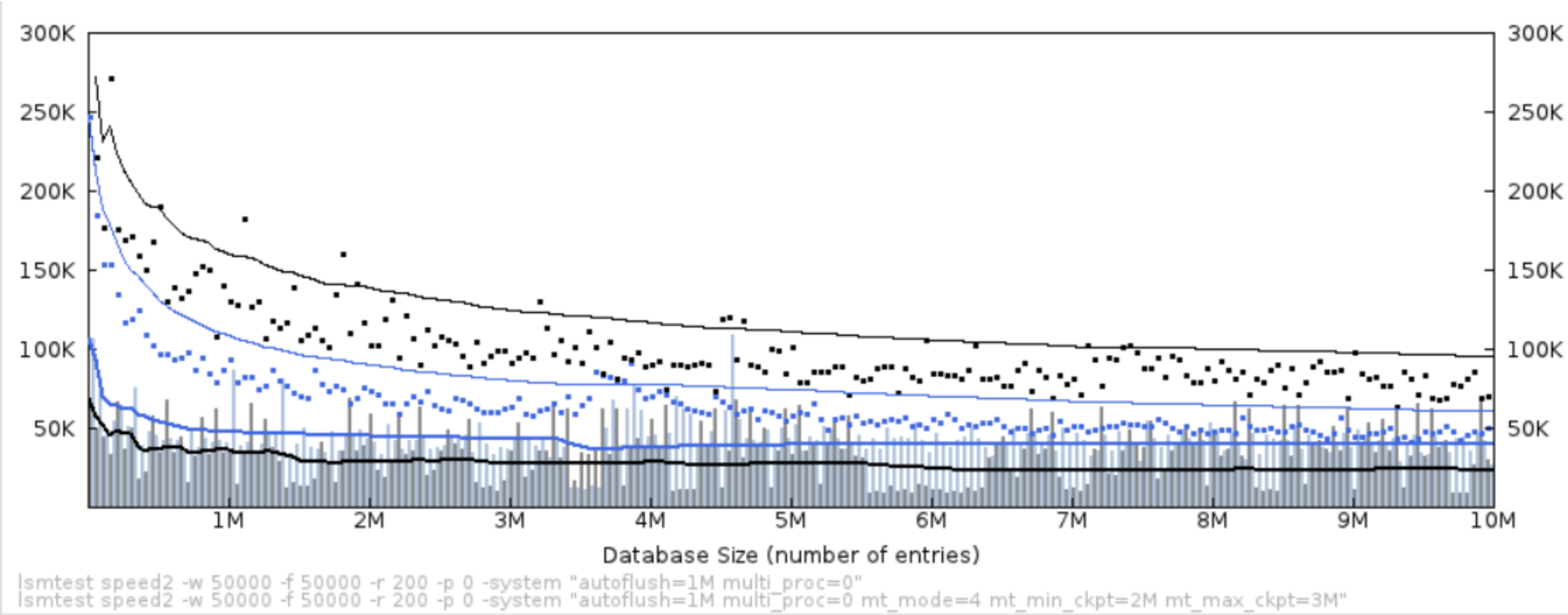
Prcedure and config

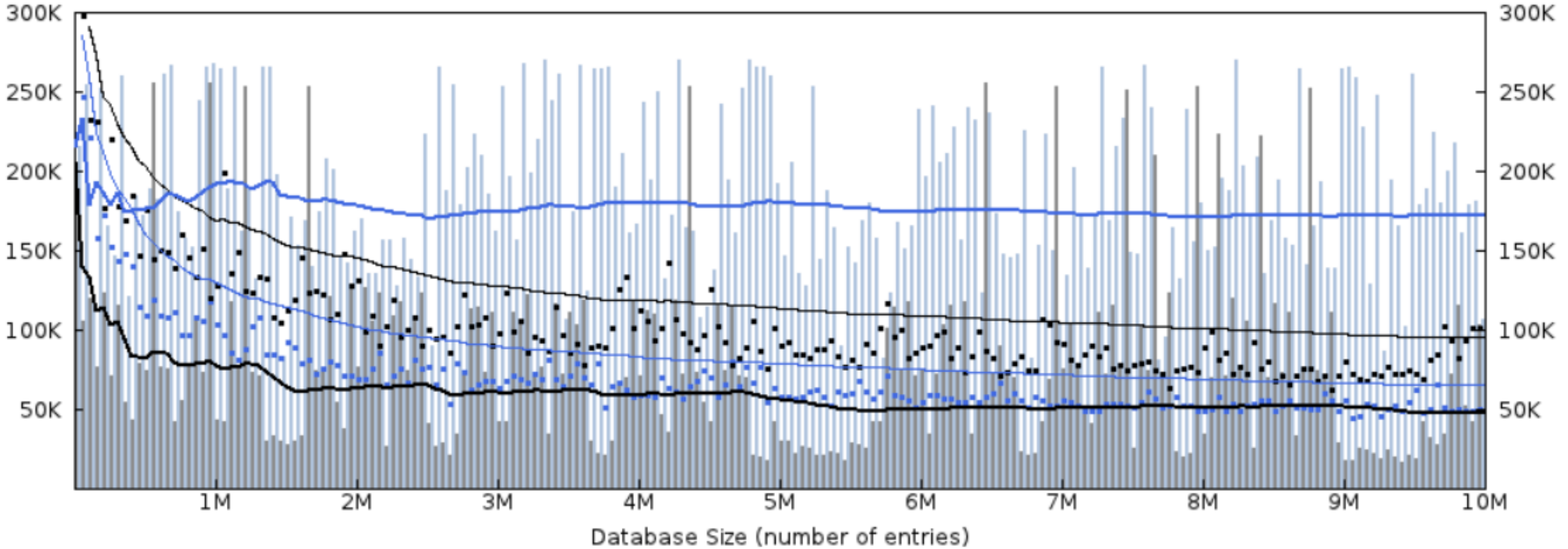
- **Small Write Buffers Test**
    - **Default** lsm ; single-threaded configuration against a multi-threaded deployment.
    - 1 MB of memory is used
    - Checkpoint after 2MB is written to database file
  - multi-threaded configuration
    - One background thread is dedicated to calling lsm_work()
    - the other to calling lsm_checkpoint()
    - The in-memory tree flushes to disk when it uses 1MB of memory.
        - If exceed 1.5MB ... Client thread is block until it has being flushed to disk
    - 2MB for the database file to be checkpoint
        - At 3MB the worker is blocked

Prcedure and config

- **Large Write Buffers Test**
    - Up to 4 MB of memory is used
    - Checkpoint after 8MB is written to database file
  - multi-threaded configuration
    - The in-memory tree flushes to disk when it uses 4MB of memory.
      - If exceed 6MB ... Client thread is block until it has being flushed to disk
    - still 2MB for the database file to be checkpoint
      - At 8MB the worker is blocked

- **Large Write Buffers Test With Pauses**
  - the client thread pauses (sleeps) for 2.5 seconds after performing the database inserts and queries.

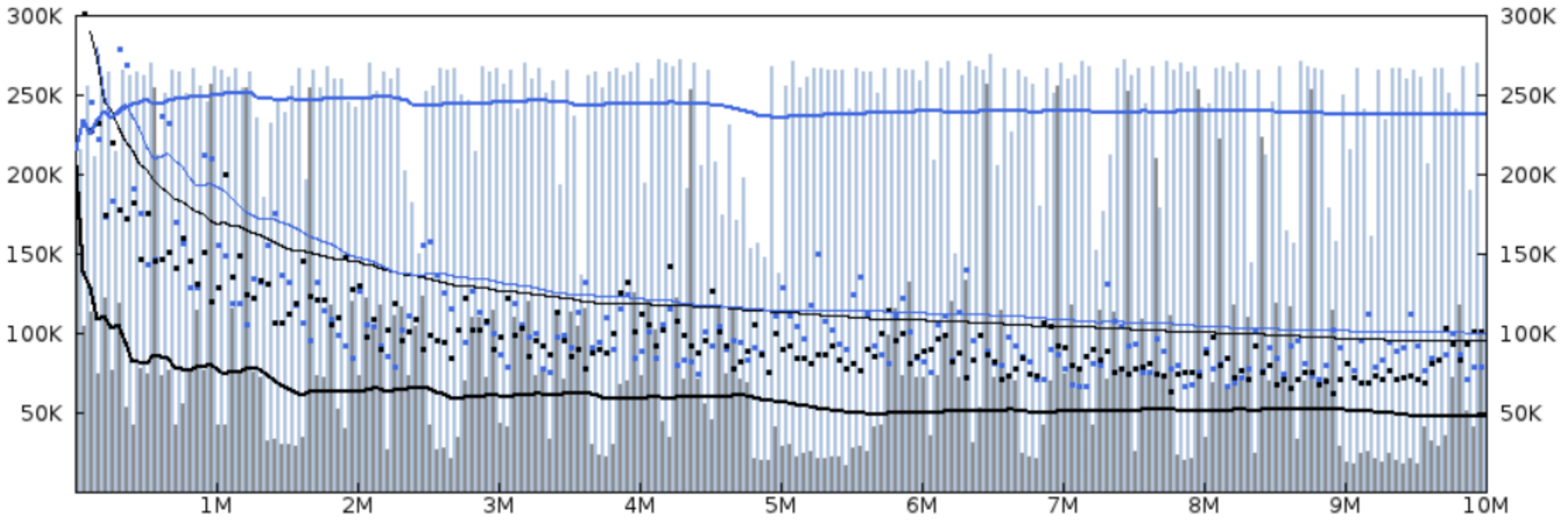

300K
250K
200K
150K
100K
50K
1M
2M
3M
4M
5M
6M
7M
8M
9M
10M
Database Size (number of entries)
lsmtest speed2 -w 50000 -f 50000 -r 200 -p 0 -system "autoflush=1M multi_proc=0"
lsmtest speed2 -w 50000 -f 50000 -r 200 -p 0 -system "autoflush=1M multi_proc=0 mt_mode=4 mt_min_ckpt=2M mt_max_ckpt=3M"
single-threaded fetches/sec
single-threaded writes/sec
single-threaded cumulative fetches/sec
single-threaded cumulative writes/sec
multi-threaded fetches/sec
multi-threaded writes/sec
multi-threaded cumulative fetches/sec
multi-threaded cumulative writes/sec



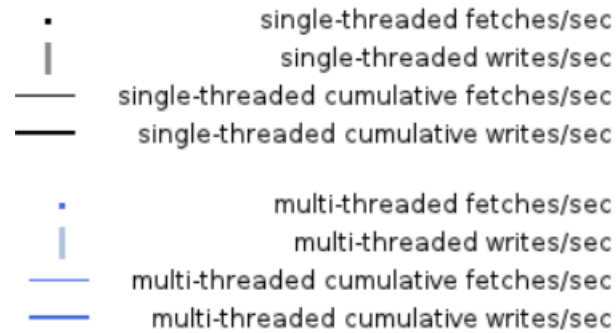
lsmtest speed2 -w 50000 -f 50000 -r 200 -p 0 -system "autoflush=4M multi_proc=0 autocheckpoint=8M"
lsmtest speed2 -w 50000 -f 50000 -r 200 -p 0 -system "autoflush=4M multi_proc=0 mt_mode=4"

single-threaded fetches/sec
single-threaded writes/sec
single-threaded cumulative fetches/sec
single-threaded cumulative writes/sec

multi-threaded fetches/sec
multi-threaded writes/sec
multi-threaded cumulative fetches/sec
multi-threaded cumulative writes/sec



lsmtest speed2 -w 50000 -f 50000 -r 200 -p 2500 -system "autoflush=4M multi_proc=0 autocheckpoint=8M"
lsmtest speed2 -w 50000 -f 50000 -r 200 -p 2500 -system "autoflush=4M multi_proc=0 mt_mode=4"



The END

THANK YOU

REFERENCES

- Patrick O'Neil1 et al 2006 : log structure merged tree
- http://en.wikipedia.org/wiki/Log-structured_merge-tree
- http://link.springer.com/content/pdf/10.1007/s002360050048.pdf
- http://sqlite.org/src4/doc/trunk/www/lsmusr.wiki
- http://sqlite.org/src4/doc/trunk/www/lsm.wiki
- http://sqlite.org/src4/doc/trunk/www/lsmperf.wiki
- http://sqlite.org/src4/doc/trunk/www/lsmapi.wiki