RocksDB

Embedded Key-Value Store for Flash and Faster Storage

Team1 黃啟軒、李昆憶、曾冠博

Overview

- Introduction to RocksDB
- Which part of source code do we trace?
- What do we learn from tracing source code?
- Discussion

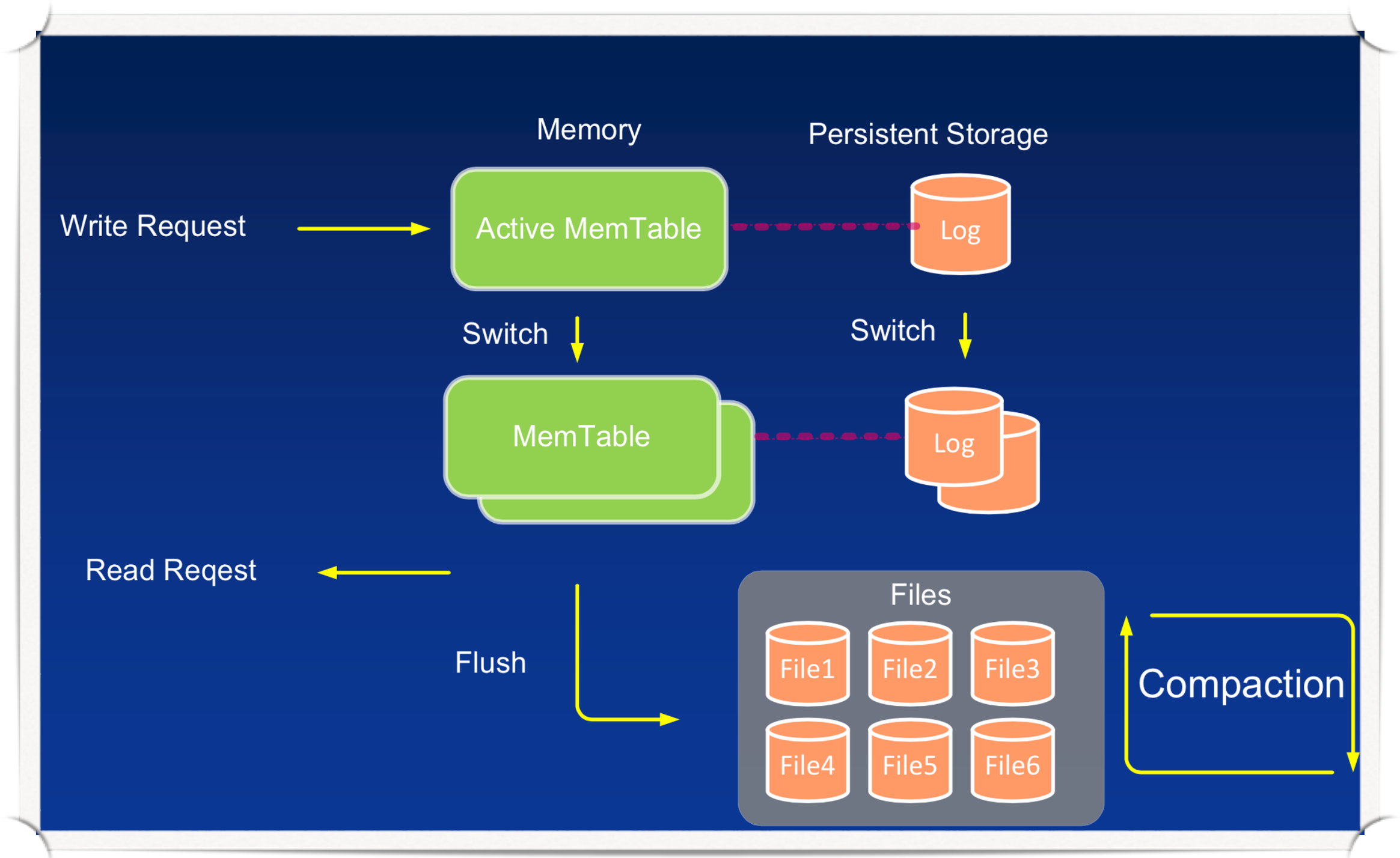
What is RocksDB

- Embedded Key-Value Store
- Open-Source, builds on LevelDB code base, written in C++
- high read/write rates, high random-read workloads, high update
- Optimized for fast storage

RocksDB API

- Keys and values are arbitrary byte arrays
- Data are stored **sorted** by key
- Update Operations: Put / Delete / Merge
- Queries: Get / Iterator

RocksDB Architecture



https://github.com/facebook/rocksdb/raw/gh-pages/talks/2014-08-05-Flash-Memory-Summit-Siying-RocksDB.pdf

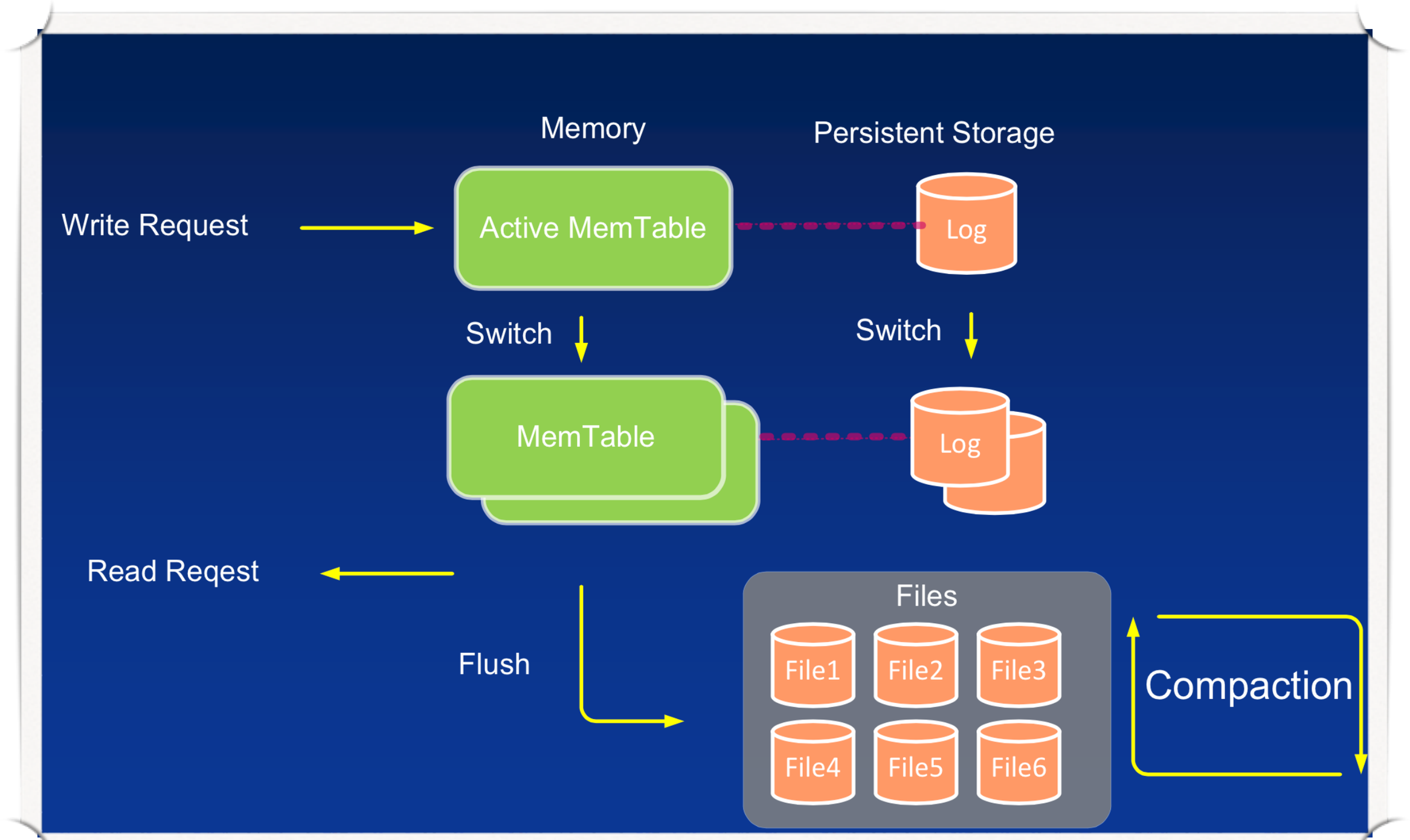
Why is RocksDB Friendly to Flash Devices?

- Reason 1. Tunable between device wear-out and read latency
- Reason 2. Pluggable
- Reason 3. Optimized for fast storage
  - Lock-free reads
  - Optimize to reduce CPU usage

```cpp
#include "rocksdb/slice.h"
#include "rocksdb/options.h"

using namespace rocksdb;
using namespace std;

std::string kDBPath = "/tmp/rocksdb_simple_example";

int main() {
  DB* db;
  Options options;
  // Optimize RocksDB. This is the easiest way to get RocksDB to perform well
  options.IncreaseParallelism();
  options.OptimizeLevelStyleCompaction();
  // create the DB if it's not already present
  options.create_if_missing = true;

  // open DB
  Status s = DB::Open(options, kDBPath, &db);
  assert(s.ok());

  // Put key-value
  s = db->Put(WriteOptions(), "key", "value");
  assert(s.ok());
  std::string value;
  // get value
  s = db->Get(ReadOptions(), "key", &value);
  assert(s.ok());
  assert(value == "value");

  delete db;

  return 0;
}
```

1:simple_example.cc [cpp] line: 42 of 43, col: 1 (Bot)

RocksDB Architecture



名詞介紹

- **Slice**: 是一個簡單的資料結構包含長度和pointer指向外部的array，因為不用透過複製，所以比string型態好。

- **Snapshot**: A Snapshot API allows an application to create a point-in-time view of a database.

- **Column family**: A column family is a NoSQL object that contains columns of related data.

Open, Put, Get, Close

Open

- 創建新的 DB 並初始化
- ColumnFamilyDescriptor：Column 的名子
  - 在 Open 中會加入一個 member “default”
- ColumnFamilyHandle：取得 Column 的名子

```
3620 Status DB::Open(const Options& options, const std::string& dbname, DB** dbptr) {
3621   DBOptions db_options(options);
3622   ColumnFamilyOptions cf_options(options);
3623   std::vector<ColumnFamilyDescriptor> column_families;
3624   column_families.push_back(
3625       ColumnFamilyDescriptor(kDefaultColumnFamilyName, cf_options));
3626   std::vector<ColumnFamilyHandle*> handles;
3627   Status s = DB::Open(db_options, dbname, column_families, &handles, dbptr);
3628   if (s.ok()) {
3629     assert(handles.size() == 1);
3630     // i can delete the handle since DBImpl is always holding a reference to
3631     // default column family
3632     delete handles[0];
3633   }
3634   return s;
3635 }
```

Open

- 找尋儲存 DB 的空間，並建立 Logger 記錄
- 在創建 DB 所需空間時，使用 Mutex 獨佔資源
- 檢查 DB 的可用功能，如：SnapShot、MergeOperator

Put

- 新增 entry 到 DB 中
- Slice：有兩個 member，data 為所存的字串，size 為該字串的長度(可以指定其值)
  - key 的 size 為字串長度，val 的 size 則為長度 + 1
- 範例程式碼使用的 Put 函式會將 entry 新增到 defaultColumnFamily

```
649 void rocksdb_put(
650     rocksdb_t* db,
651     const rocksdb_writeoptions_t* options,
652     const char* key, size_t keylen,
653     const char* val, size_t vallen,
654     char** errptr) {
655   SaveError(errptr,
656             db->rep->Put(options->rep, Slice(key, keylen), Slice(val, vallen)));
657 }
```

Put

```
163   // Set the database entry for "key" to "value".
164   // If "key" already exists, it will be overwritten.
165   // Returns OK on success, and a non-OK status on error.
166   // Note: consider setting options.sync = true.
167   virtual Status Put(const WriteOptions& options,
168                      ColumnFamilyHandle* column_family, const Slice& key,
169                      const Slice& value) = 0;
170   virtual Status Put(const WriteOptions& options, const Slice& key,
171                      const Slice& value) {
172     return Put(options, DefaultColumnFamily(), key, value);
173   }
```

db.h

```
2864 // Convenience methods
2865 Status DBImpl::Put(const WriteOptions& o, ColumnFamilyHandle* column_family,
2866                    const Slice& key, const Slice& val) {
2867   return DB::Put(o, column_family, key, val);
2868 }
```

db_impl.cc

db_impl.cc

```
3582 // Default implementations of convenience methods that subclasses of DB
3583 // can call if they wish
3584 Status DB::Put(const WriteOptions& opt, ColumnFamilyHandle* column_family,
3585                 const Slice& key, const Slice& value) {
3586   // Pre-allocate size of write batch conservatively.
3587   // 8 bytes are taken by header, 4 bytes for count, 1 byte for type,
3588   // and we allocate 11 extra bytes for key length, as well as value length.
3589   WriteBatch batch(key.size() + value.size() + 24);
3590   batch.Put(column_family, key, value);
3591   return Write(opt, &batch);
3592 }
```

write_batch.cc

```
190 void WriteBatchInternal::Put(WriteBatch* b, uint32_t column_family_id,
191                               const Slice& key, const Slice& value) {
192   WriteBatchInternal::SetCount(b, WriteBatchInternal::Count(b) + 1);
193   if (column_family_id == 0) {
194     b->rep_.push_back(static_cast<char>(kTypeValue));
195   } else {
196     b->rep_.push_back(static_cast<char>(kTypeColumnFamilyValue));
197     PutVarint32(&b->rep_, column_family_id);
198   }
199   PutLengthPrefixedSlice(&b->rep_, key);
200   PutLengthPrefixedSlice(&b->rep_, value);
201 }
202
203 void WriteBatch::Put(ColumnFamilyHandle* column_family, const Slice& key,
204                      const Slice& value) {
205   WriteBatchInternal::Put(this, GetColumnFamilyID(column_family), key, value);
206 }
```

Put

- 資料儲存格式為一個字串
  - Sequence[64bits] + Count[32bits] + record[Count]
- Count：int：標記 record 的長度
- Record：
  - kTypeValue(0x1) + key.size + key.data + value.size + key.data
  - kTypeColumnFamilyValue(0x5) + column_family_id + key.size + key.data + value.size + value.data

```
pumpkin@pumpkin-u14:/tmp/rocksdb_simple_example$ xxd -g 1 -c 8 000003.log
0000000: 15 6b 85 d2 18 00 01 01  .k......
0000008: 00 00 00 00 00 00 00 01  ........
0000010: 00 00 00 01 03 6b 65 79  .....key
0000018: 06 76 61 6c 75 65 00     .value.
```

Get

- 以指定的 key 找尋 value
- 先保護 Timer，建立 Snapshot，將 Timer 設為 Snapshot 的時間，找尋完畢後，將 Timer 設回 process 的時間
- 找尋順序為 memtable -> immutable memtable

Get

- Seek：從表中找尋第一個 Node
- callback_func：程式中為 “SaveValue”，如果為所需的 key，就會存 value，並回傳 false
- Next：移動到下一個 Node
- Valid：檢查目前的 Node 是否為 null

```
44   virtual void Get(const LookupKey& k, void* callback_args,
45                    bool (*callback_func)(void* arg,
46                                          const char* entry)) override {
47     SkipListRep::Iterator iter(&skip_list_);
48     Slice dummy_slice;
49     for (iter.Seek(dummy_slice, k.memtable_key().data());
50          iter.Valid() && callback_func(callback_args, iter.key());
51          iter.Next()) {
52     }
53   }
```

Close

- 刪除 DB
- 將所開啟的 thread 刪除
- Release DB 所占用的記憶體
- 刪除 Logger

總結

Thank you!