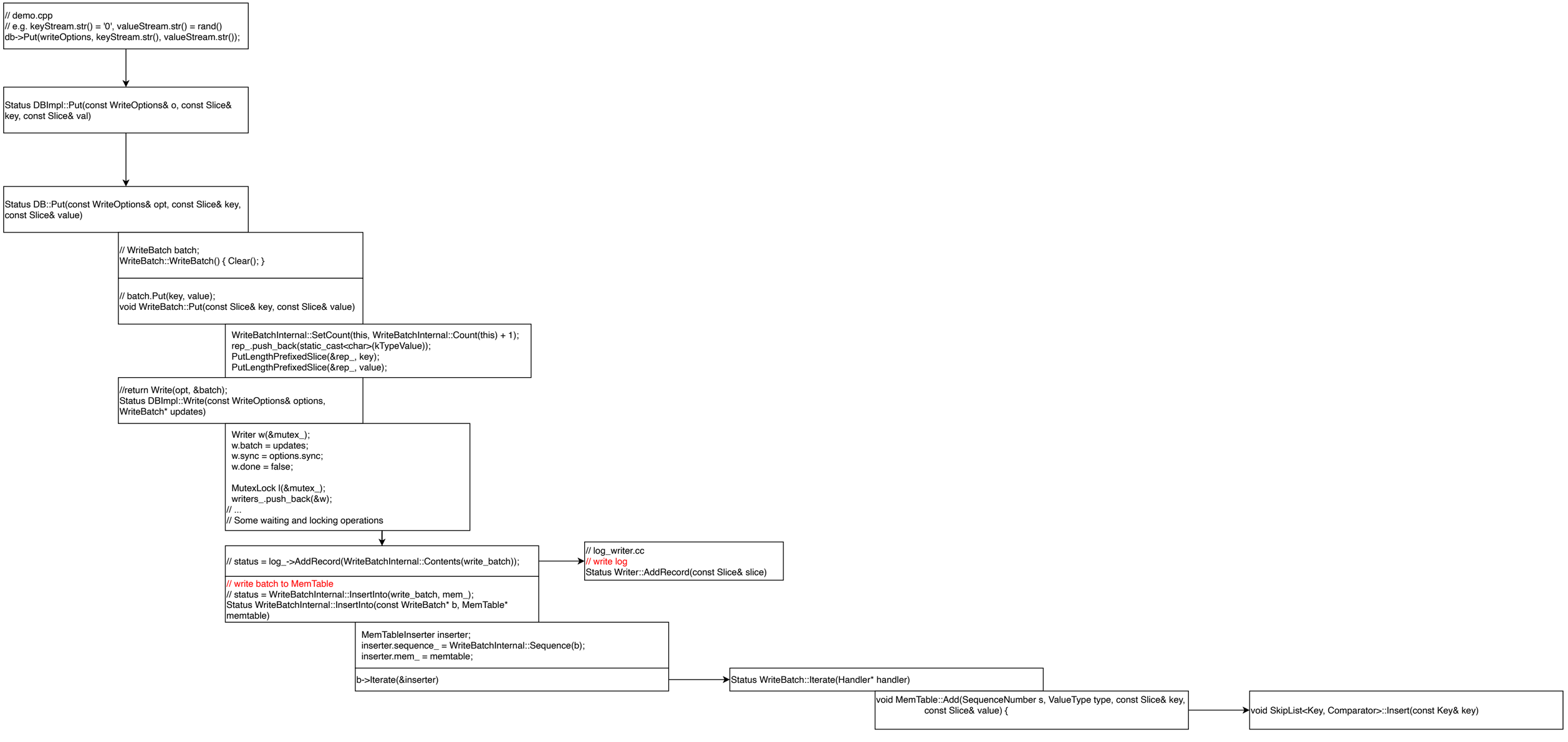
// demo.cpp
// e.g. keyStream.str() = '0', valueStream.str() = rand()
db->Put(writeOptions, keyStream.str(), valueStream.str());
Status DBImpl::Put(const WriteOptions& o, const Slice&
key, const Slice& val)
Status DB::Put(const WriteOptions& opt, const Slice& key,
const Slice& value)
// WriteBatch batch;
WriteBatch::WriteBatch() { Clear(); }
// batch.Put(key, value);
void WriteBatch::Put(const Slice& key, const Slice& value)
WriteBatchInternal::SetCount(this, WriteBatchInternal::Count(this) + 1);
rep_.push_back(static_cast<char>(kTypeValue));
PutLengthPrefixedSlice(&rep_, key);
PutLengthPrefixedSlice(&rep_, value);
//return Write(opt, &batch);
Status DBImpl::Write(const WriteOptions& options,
WriteBatch* updates)
Writer w(&mutex_);
w.batch = updates;
w.sync = options.sync;
w.done = false;
MutexLock l(&mutex_);
writers_.push_back(&w);
// ...
// Some waiting and locking operations
// status = log_->AddRecord(WriteBatchInternal::Contents(write_batch));
// log_writer.cc
// write log
Status Writer::AddRecord(const Slice& slice)
// write batch to MemTable
// status = WriteBatchInternal::InsertInto(write_batch, mem_);
Status WriteBatchInternal::InsertInto(const WriteBatch* b, MemTable*
memtable)
MemTableInserter inserter;
inserter.sequence_ = WriteBatchInternal::Sequence(b);
inserter.mem_ = memtable;
b->Iterate(&inserter)
Status WriteBatch::Iterate(Handler* handler)
void MemTable::Add(SequenceNumber s, ValueType type, const Slice& key,
const Slice& value) {
void SkipList<Key, Comparator>::Insert(const Key& key)