Week5 Homework

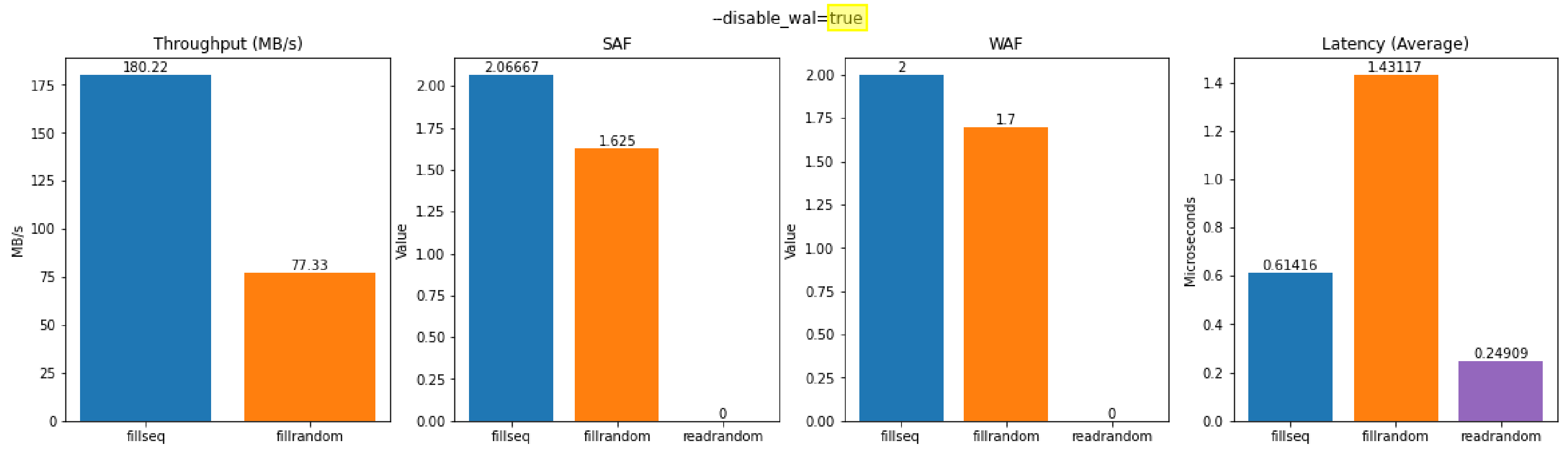
Made by Suhwan Shin, Isu Kim

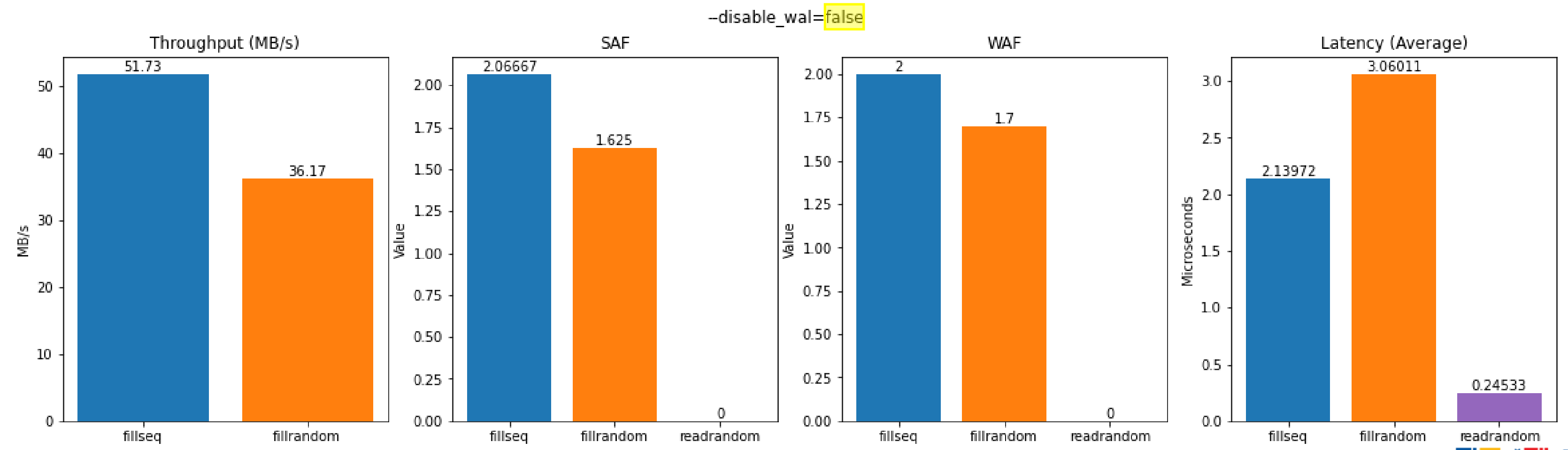
E-Mail: tlstnghks77@dankook.ac.kr

Content

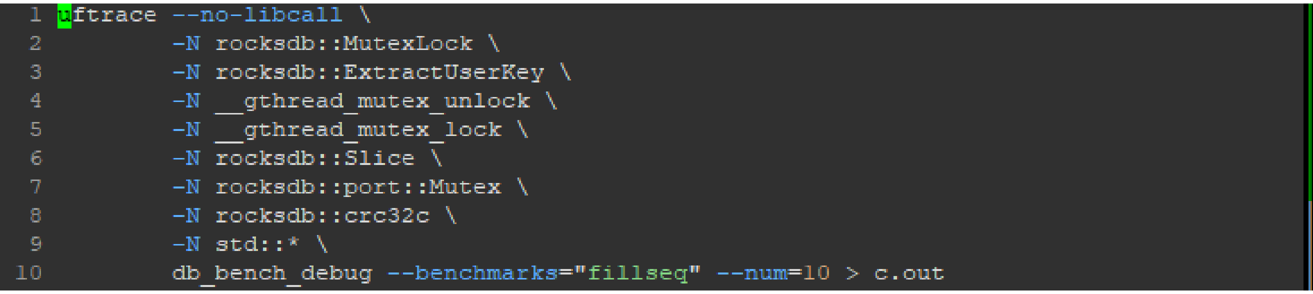
1. Preview - RocksDB --disable_wal

- Average of 10 db_bench results

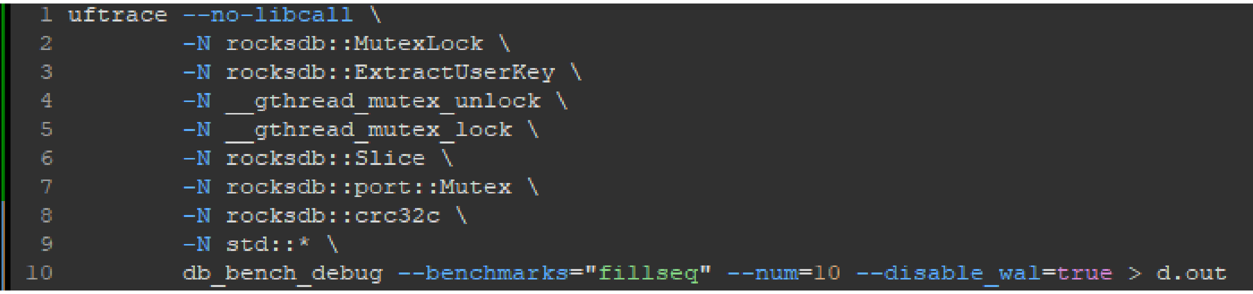




2. Discussion - Uftrace outputs



```
uftrace --no-libcall \
        -N rocksdb::MutexLock \
        -N rocksdb::ExtractUserKey \
        -N __gthread_mutex_unlock \
        -N __gthread_mutex_lock \
        -N rocksdb::Slice \
        -N rocksdb::port::Mutex \
        -N rocksdb::crc32c \
        -N std::* \
        db_bench_debug --benchmarks="fillseq" --num=10 > c.out
```



```
uftrace --no-libcall \
        -N rocksdb::MutexLock \
        -N rocksdb::ExtractUserKey \
        -N __gthread_mutex_unlock \
        -N __gthread_mutex_lock \
        -N rocksdb::Slice \
        -N rocksdb::port::Mutex \
        -N rocksdb::crc32c \
        -N std::* \
        db_bench_debug --benchmarks="fillseq" --num=10 --disable_wal=true > d.out
```

--disable_wal=false (default)

- rocksdb::DBImpl::CreateWAL
- rocksdb::DBImpl::CalculateWALWriteHint()
- rocksdb::WalManager::PurgeObsoleteWALFiles()
- rocksdb::DBImpl::TEST_WALBufferIsEmpty()
- rocksdb::WalManager::PurgeObsoleteWALFiles()
- rocksdb::DBImpl::WriteToWAL()

--disable_wal=true (default)

- rocksdb::DBImpl::CreateWAL
- rocksdb::DBImpl::CalculateWALWriteHint()
- rocksdb::WalManager::PurgeObsoleteWALFiles()
- rocksdb::DBImpl::TEST_WALBufferIsEmpty()
- rocksdb::WalManager::PurgeObsoleteWALFiles()

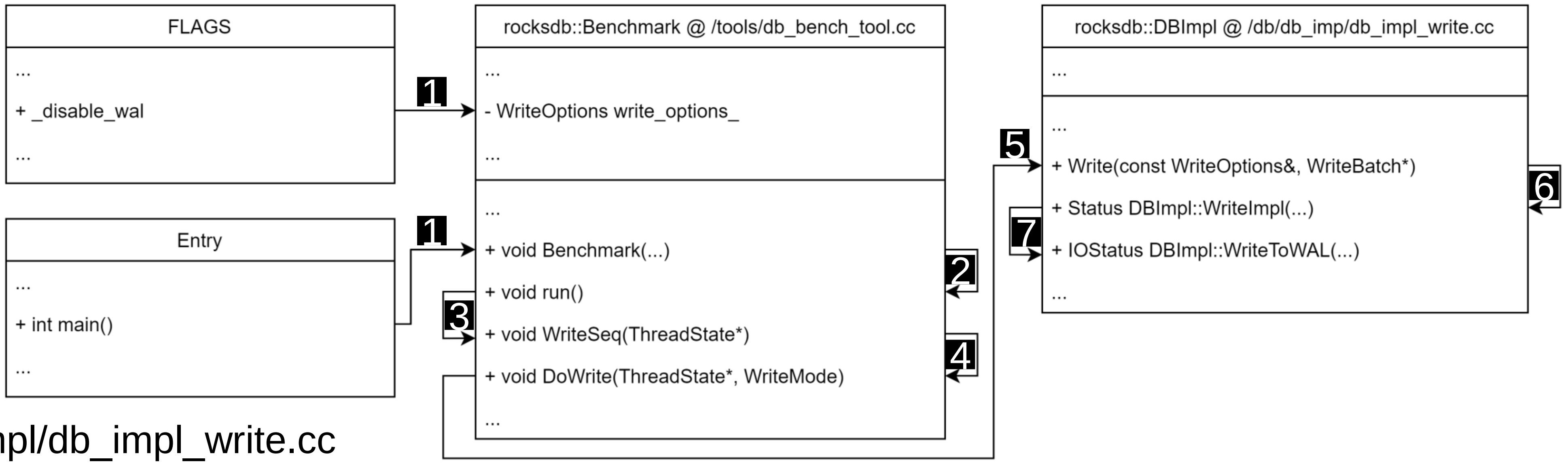
~~- rocksdb::DBImpl::WriteToWAL()~~

--disable_wal=true **does not** use rocksdb::DBImpl::WriteToWAL()

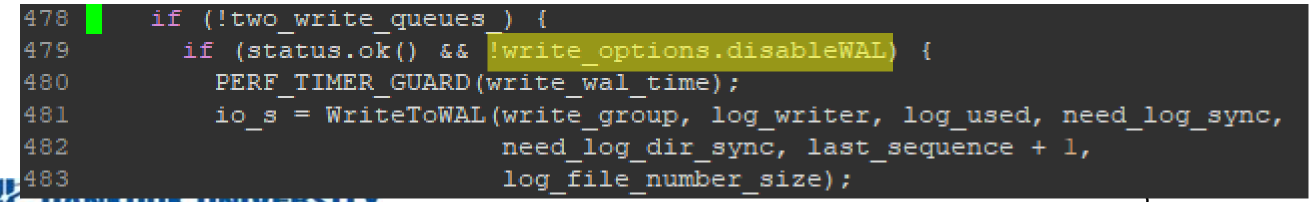
How? and Why?

3. Internal operations - fillseq

- This is not a full proper UML class diagram
- What happens if --disable_wal is enabled?


FLAGS
...
+ _disable_wal
...
Entry
...
+ int main()
...
rocksdb::Benchmark @ /tools/db_bench_tool.cc
...
- WriteOptions write_options_
...
...
+ void Benchmark(...)
+ void run()
+ void WriteSeq(ThreadState*)
+ void DoWrite(ThreadState*, WriteMode)
...
rocksdb::DBImpl @ /db/db_imp/db_impl_write.cc
...
...
+ Write(const WriteOptions&, WriteBatch*)
+ Status DBImpl::WriteImpl(...)
+ IOStatus DBImpl::WriteToWAL(...)
...
1
1
2
3
4
5
6
7

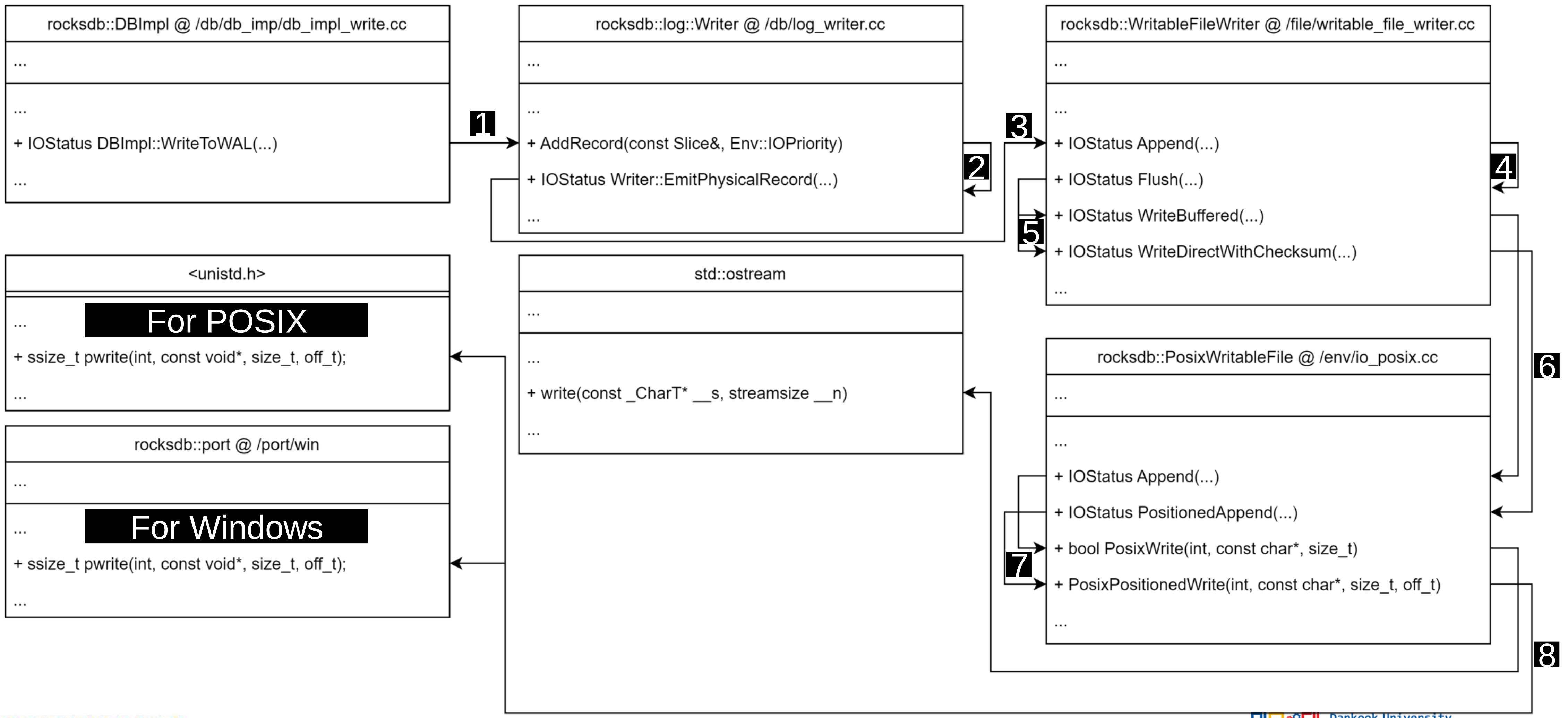
/db/db_impl/db_impl_write.cc



```
  if (!two_write_queues_) {
    if (status.ok() && !write_options.disableWAL) {
      PERF_TIMER_GUARD(write_wal_time);
      io_s = WriteToWAL(write_group, log_writer, log_used, need_log_sync,
                        need_log_dir_sync, last_sequence + 1,
                        log_file_number_size);
```

Nested if becomes **if (false)**
Thus, not calling WriteToWAL()

3. Internal operations - fillseq


rocksdb::DBImpl @ /db/db_imp/db_impl_write.cc
...
...
+ IOStatus DBImpl::WriteToWAL(...)
...
rocksdb::log::Writer @ /db/log_writer.cc
...
...
+ AddRecord(const Slice&, Env::IOPriority)
+ IOStatus Writer::EmitPhysicalRecord(...)
...
rocksdb::WritableFileWriter @ /file/writable_file_writer.cc
...
...
+ IOStatus Append(...)
+ IOStatus Flush(...)
+ IOStatus WriteBuffered(...)
+ IOStatus WriteDirectWithChecksum(...)
...
<unistd.h>
...
For POSIX
+ ssize_t pwrite(int, const void*, size_t, off_t);
...
std::ostream
...
...
+ write(const _CharT* __s, streamsize __n)
...
rocksdb::port @ /port/win
...
...
For Windows
+ ssize_t pwrite(int, const void*, size_t, off_t);
...
rocksdb::PosixWritableFile @ /env/io_posix.cc
...
...
+ IOStatus Append(...)
+ IOStatus PositionedAppend(...)
+ bool PosixWrite(int, const char*, size_t)
+ PosixPositionedWrite(int, const char*, size_t, off_t)
...
1
2
3
4
5
6
7
8

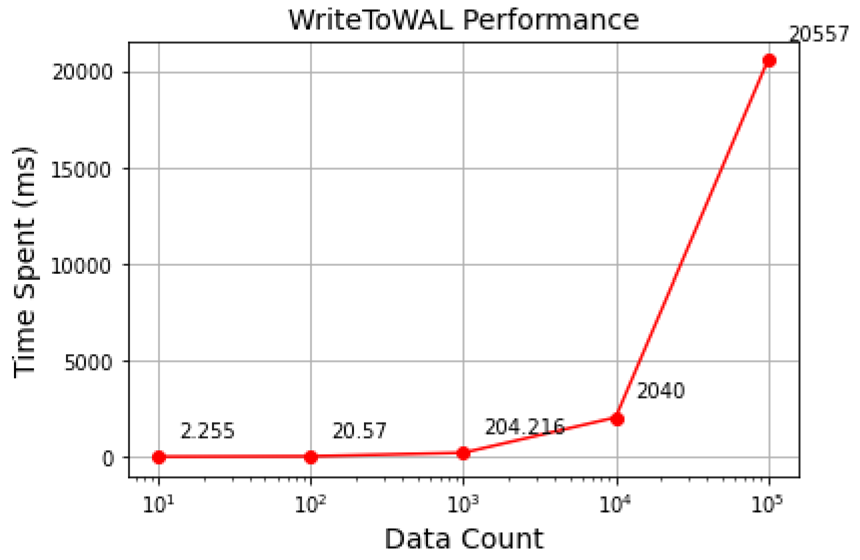
4. Performance - WriteToWAL

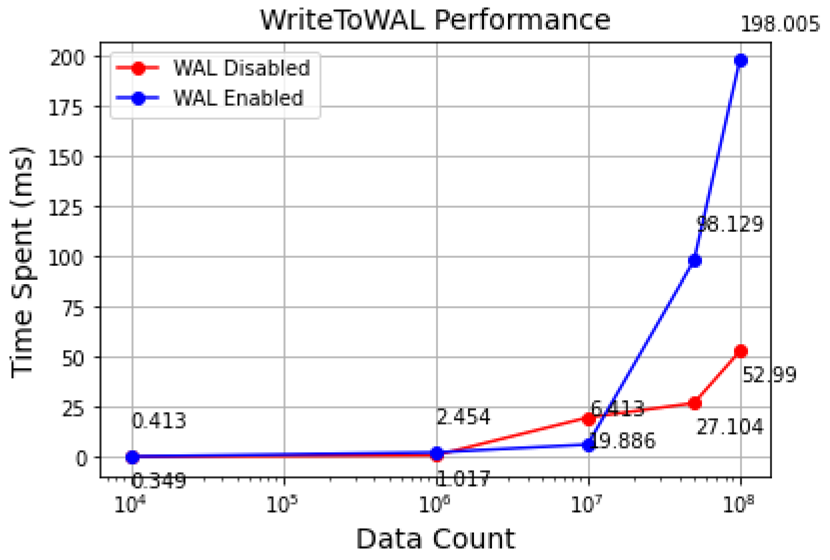
- Enabling WAL will use IO, thus in terms of throughput and latency, it is slower than disabling it.

- Future experiment ideas:
  - ✓ Use uftrace to analyze more about internal operations.
  - Disable WAL and shutdown abruptly.
  - Enable WAL and shutdown abruptly.
  - Use LevelDB for understanding WAL.

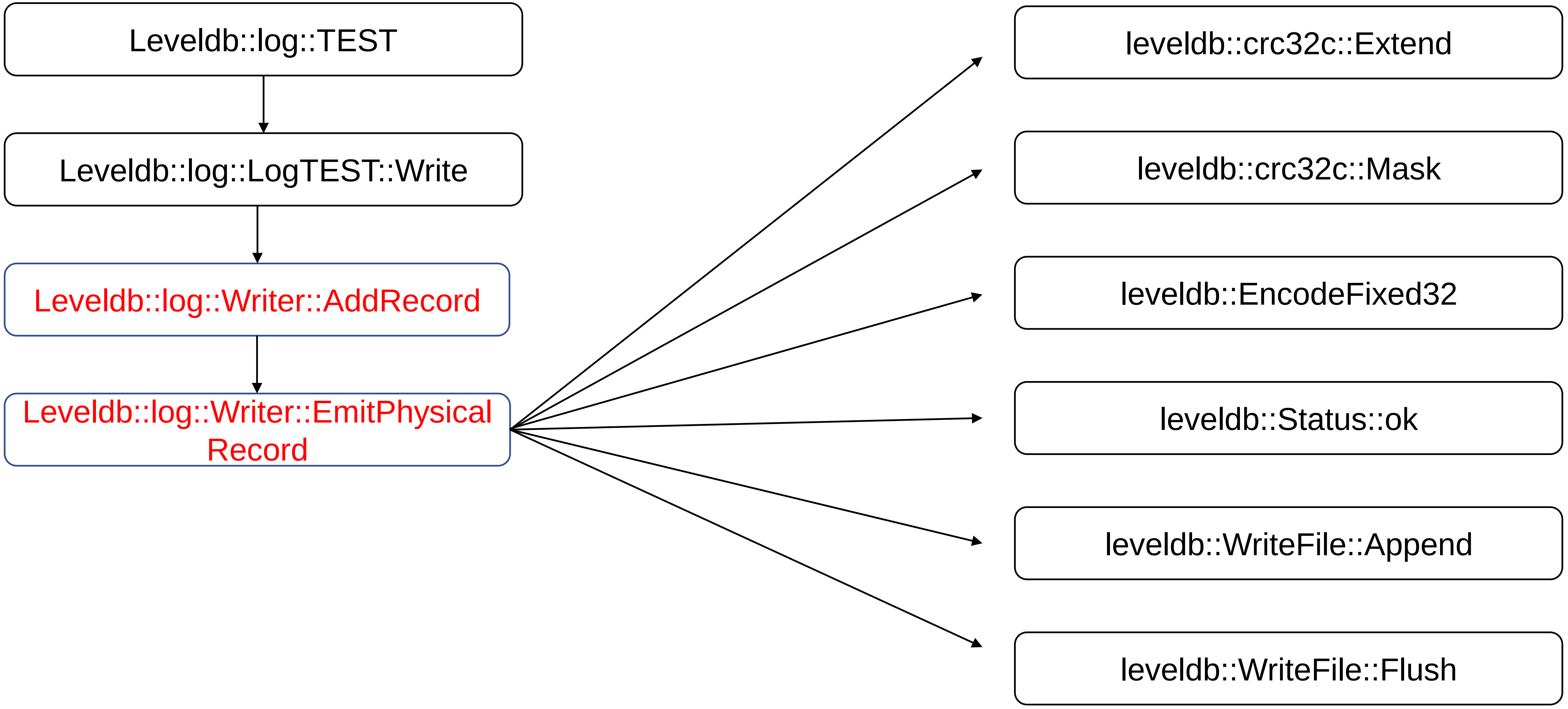
5. Conclusion & Future Study

- Minor differences with small entries
- However, huge performance gap with large data

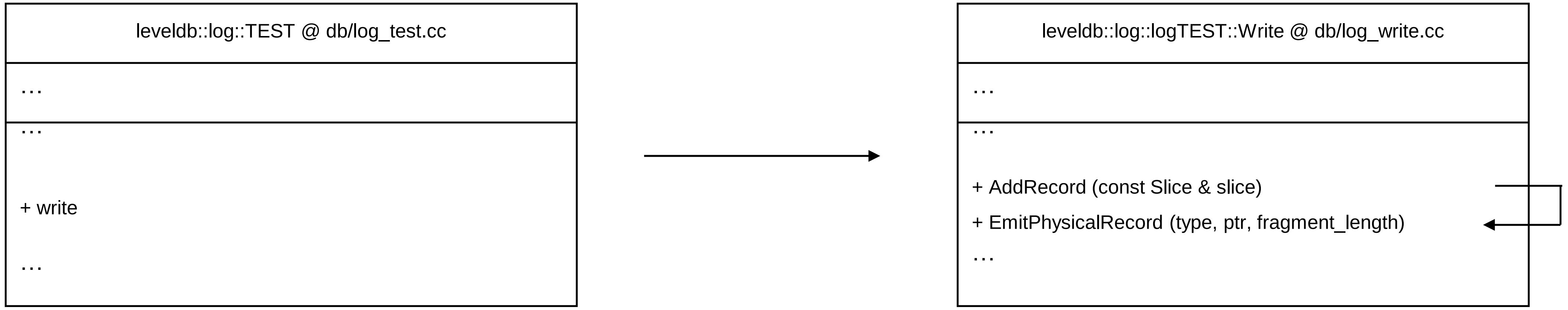




Leveldb/log/


Leveldb::log::TEST
Leveldb::log::LogTEST::Write
Leveldb::log::Writer::AddRecord
Leveldb::log::Writer::EmitPhysical Record
leveldb::crc32c::Extend
leveldb::crc32c::Mask
leveldb::EncodeFixed32
leveldb::Status::ok
leveldb::WriteFile::Append
leveldb::WriteFile::Flush

Leveldb/log/


leveldb::log::TEST @ db/log_test.cc
…
…
+ write
…
leveldb::log::logTEST::Write @ db/log_write.cc
…
…
+ AddRecord (const Slice & slice)
+ EmitPhysicalRecord (type, ptr, fragment_length)
…

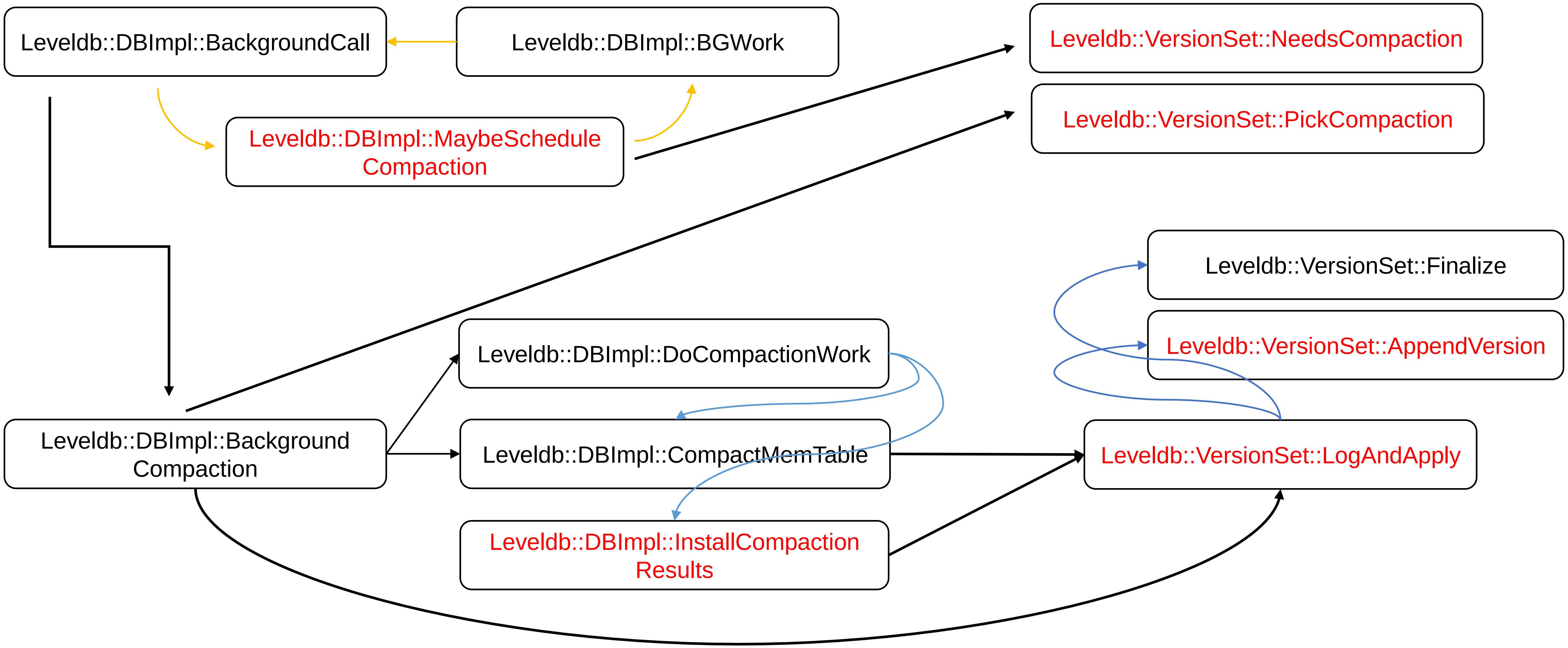
db/log_test.cc

```
void Write(const std::string& msg) {
  ASSERT_TRUE(!reading_) << "Write() after starting to read";
  writer_->AddRecord(Slice(msg));
}
```

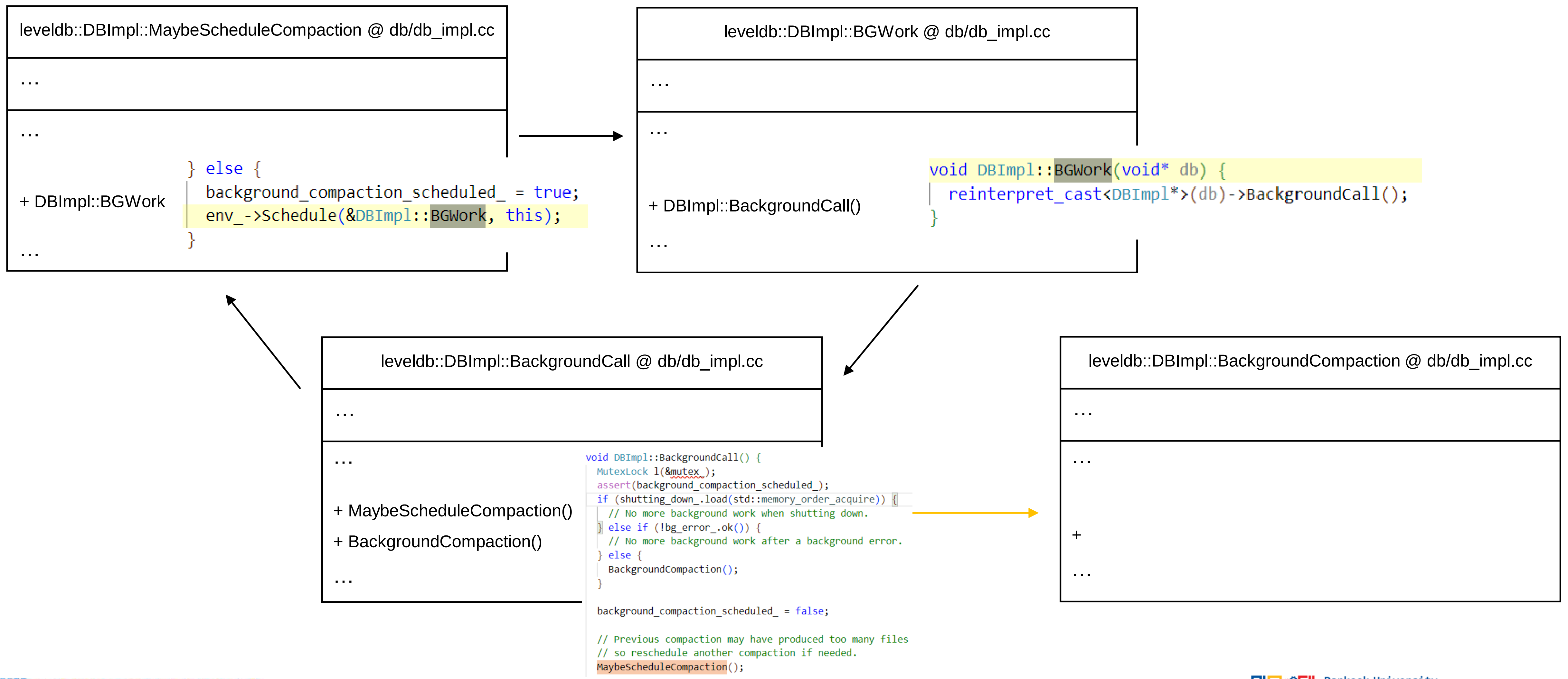
db/log_writer.cc

```
Status Writer::AddRecord(const Slice& slice) {
  const char* ptr = slice.data();
  size_t left = slice.size();
                          …
    s = EmitPhysicalRecord(type, ptr, fragment_length);
    ptr += fragment_length;
    left -= fragment_length;
    begin = false;
  } while (s.ok() && left > 0);
  return s;
}
```

Leveldb/DBImpl/


Leveldb::DBImpl::BackgroundCall
Leveldb::DBImpl::BGWork
Leveldb::VersionSet::NeedsCompaction
Leveldb::VersionSet::PickCompaction
Leveldb::DBImpl::MaybeSchedule Compaction
Leveldb::VersionSet::Finalize
Leveldb::VersionSet::AppendVersion
Leveldb::DBImpl::DoCompactionWork
Leveldb::DBImpl::Background Compaction
Leveldb::DBImpl::CompactMemTable
Leveldb::VersionSet::LogAndApply
Leveldb::DBImpl::InstallCompaction Results

Leveldb/DBImpl/MaybeSchedule-



leveldb::DBImpl::MaybeScheduleCompaction @ db/db_impl.cc

…

…

+ DBImpl::BGWork

…

```
} else {
  background_compaction_scheduled_ = true;
  env_->Schedule(&DBImpl::BGWork, this);
}
```

leveldb::DBImpl::BGWork @ db/db_impl.cc

…

…

+ DBImpl::BackgroundCall()

…

```
void DBImpl::BGWork(void* db) {
  reinterpret_cast<DBImpl*>(db)->BackgroundCall();
}
```

leveldb::DBImpl::BackgroundCall @ db/db_impl.cc

…

…

+ MaybeScheduleCompaction()

+ BackgroundCompaction()

…

```
void DBImpl::BackgroundCall() {
  MutexLock l(&mutex_);
  assert(background_compaction_scheduled_);
  if (shutting_down_.load(std::memory_order_acquire)) {
    // No more background work when shutting down.
  } else if (!bg_error_.ok()) {
    // No more background work after a background error.
  } else {
    BackgroundCompaction();
  }

  background_compaction_scheduled_ = false;

  // Previous compaction may have produced too many files
  // so reschedule another compaction if needed.
  MaybeScheduleCompaction();
```

leveldb::DBImpl::BackgroundCompaction @ db/db_impl.cc

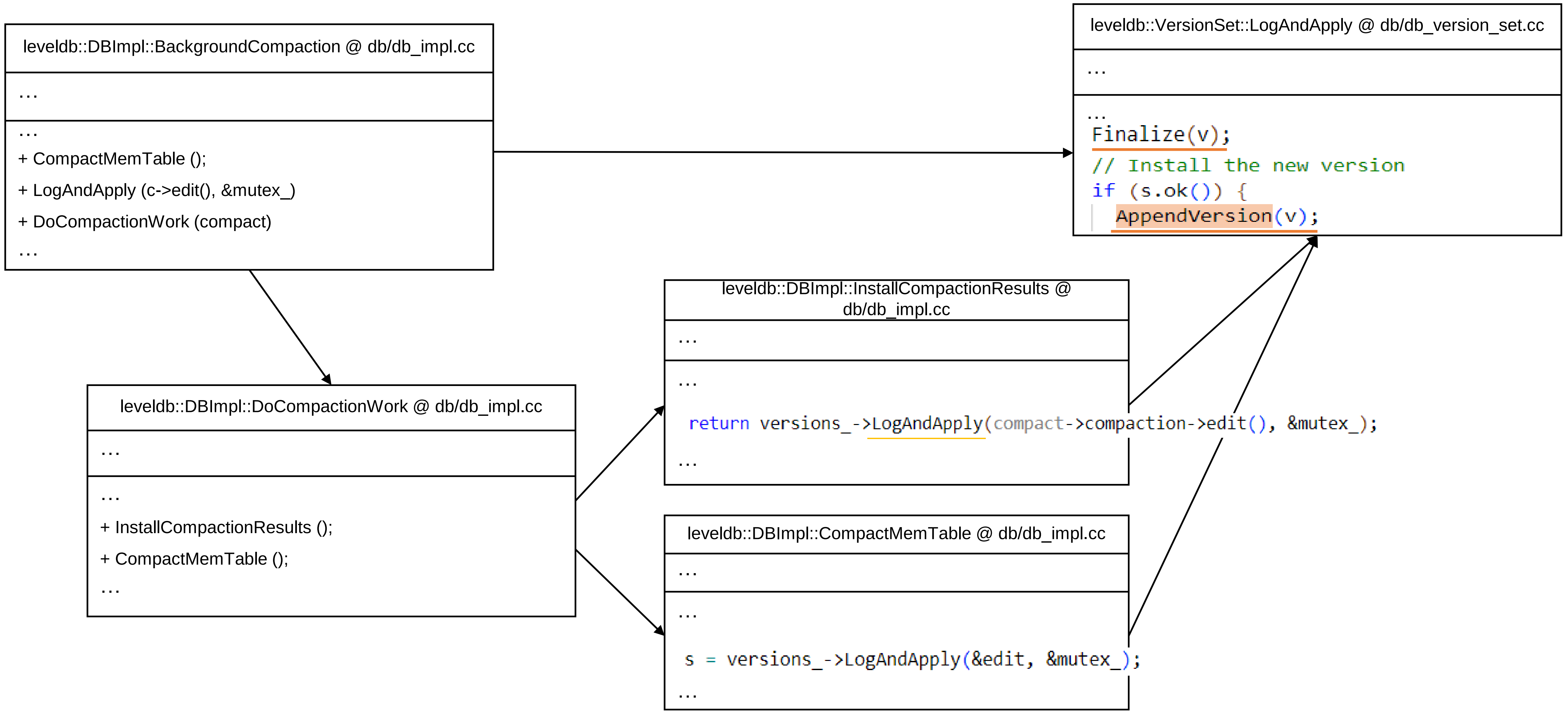
…

…

+

…

Leveldb/DBImpl/BackgroundCompaction



leveldb::DBImpl::BackgroundCompaction @ db/db_impl.cc

```
…
…
+ CompactMemTable ();
+ LogAndApply (c->edit(), &mutex_)
+ DoCompactionWork (compact)
…
```

leveldb::DBImpl::DoCompactionWork @ db/db_impl.cc

```
…
…
+ InstallCompactionResults ();
+ CompactMemTable ();
…
```

leveldb::DBImpl::InstallCompactionResults @ db/db_impl.cc

```
…
…
 return versions_->LogAndApply(compact->compaction->edit(), &mutex_);
…
```

leveldb::DBImpl::CompactMemTable @ db/db_impl.cc

```
…
…
 s = versions_->LogAndApply(&edit, &mutex_);
…
```

leveldb::VersionSet::LogAndApply @ db/db_version_set.cc

```
…
…
Finalize(v);
// Install the new version
if (s.ok()) {
  AppendVersion(v);
```

Question