LevelDB-Study

Team_Cache Analysis

Made by Subin Hong, Seungwon Ha

E-Mail: zed6740@dankook.ac.kr, 12gktmddnjs@naver.com

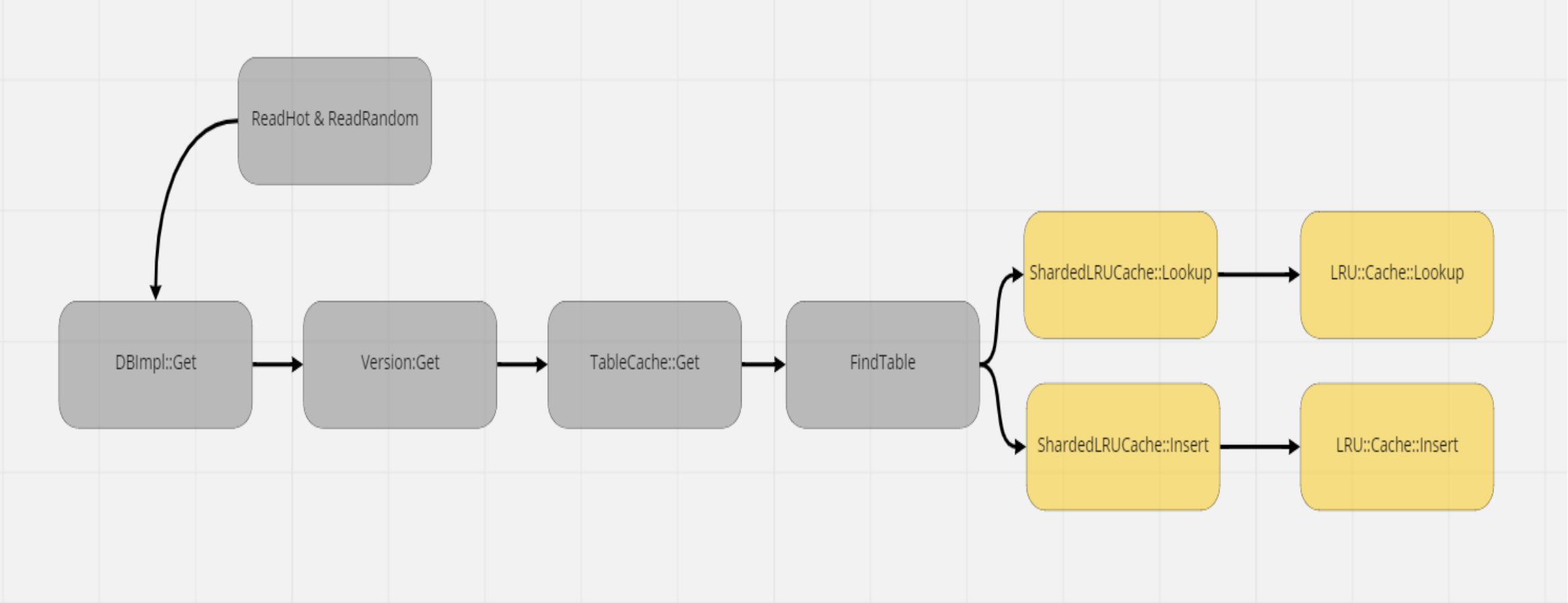
Contents

1. Cache flow analysis

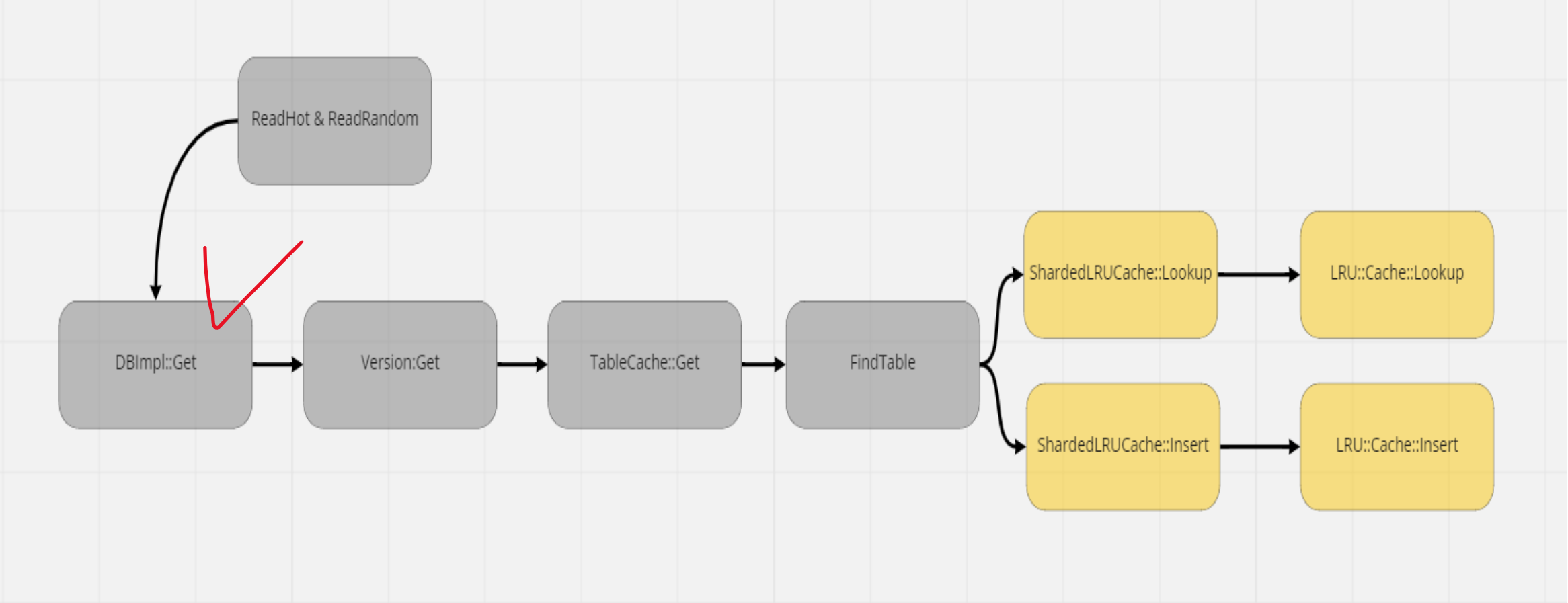
Cache flow analysis

- Specific code analysis

Cache flow analysis



Cache flow analysis



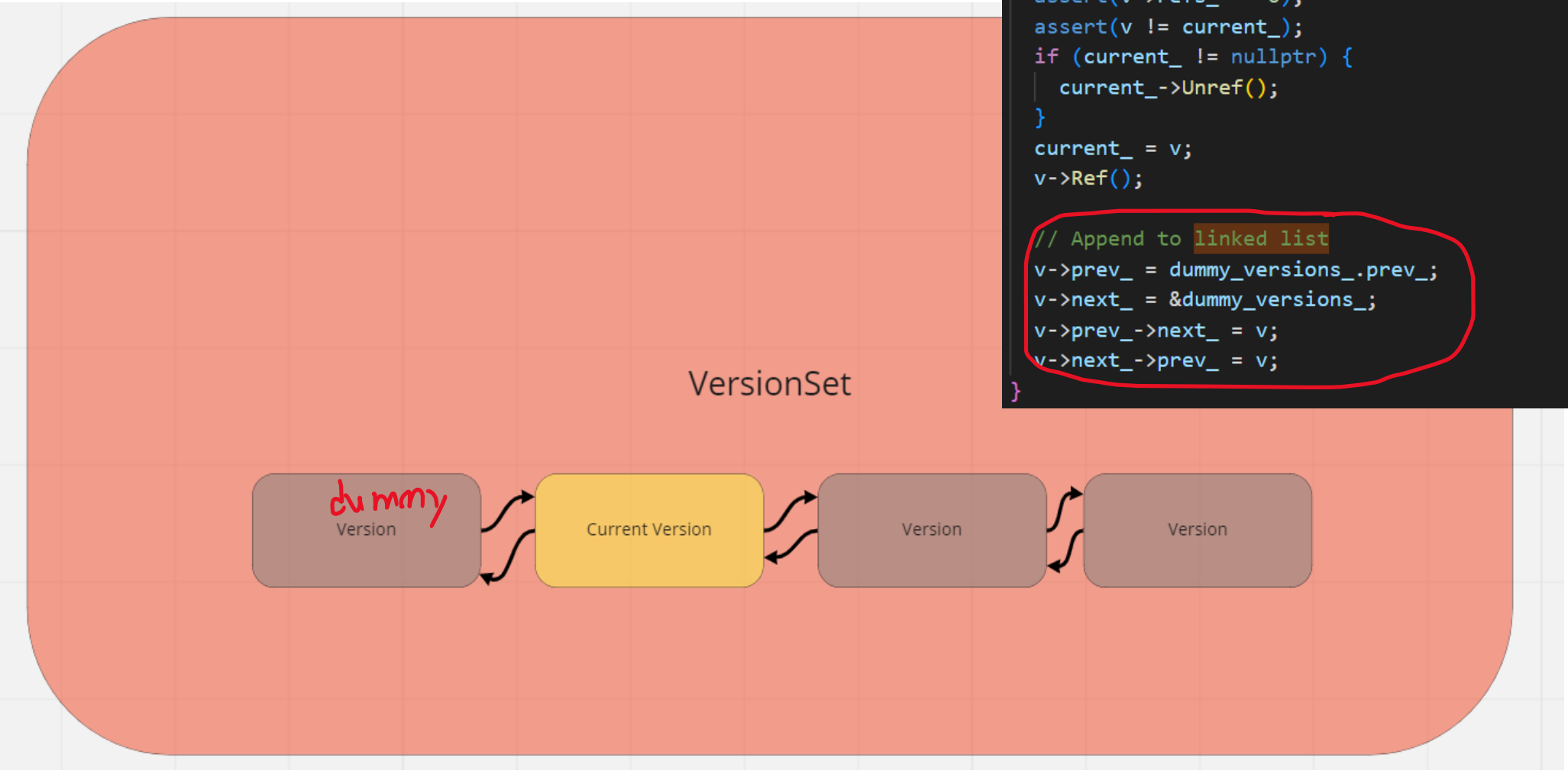
Cache flow analysis

```
Status DBImpl::Get(const ReadOptions& options, const Slice& key,
                   std::string* value) {
```

```
MemTable* mem = mem_;
MemTable* imm = imm_;
Version* current = versions_->current();
```

Cache flow analysis

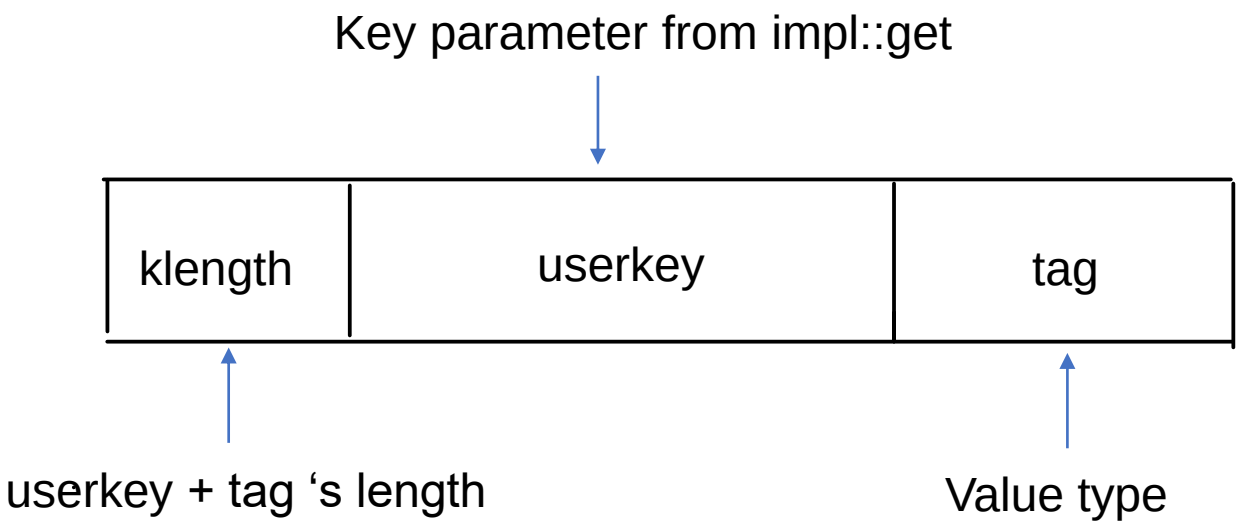
```
void VersionSet::AppendVersion(Version* v) {
  // Make "v" current
  assert(v->refs_ == 0);
  assert(v != current_);
  if (current_ != nullptr) {
    current_->Unref();
  }
  current_ = v;
  v->Ref();

  // Append to linked list
  v->prev_ = dummy_versions_.prev_;
  v->next_ = &dummy_versions_;
  v->prev_->next_ = v;
  v->next_->prev_ = v;
}
```



Cache flow analysis

```
LookupKey lkey(key, snapshot);
```

```
// We construct a char array of the form:
//    klength  varint32                <-- start_
//    userkey  char[klength]           <-- kstart_
//    tag      uint64
```



Cache flow analysis

```
mutex_.Unlock();
// First look in the memtable, then in the immutable memtabl
LookupKey lkey(key, snapshot);
if (mem->Get(lkey, value, &s)) {
  // Done
} else if (imm != nullptr && imm->Get(lkey, value, &s)) {
  // Done
} else {
  s = current->Get(options, lkey, value, &stats);
  have_stat_update = true;
}
mutex_.Lock();
```

```
1142    if (mem->Get(lkey, value, &s)) {
memtable.cc ~/leveldb_release/db - Definitions (2)
 98     assert(p + val_size == buf + encoded_len);
 99     table_.Insert(buf);
100   }
```

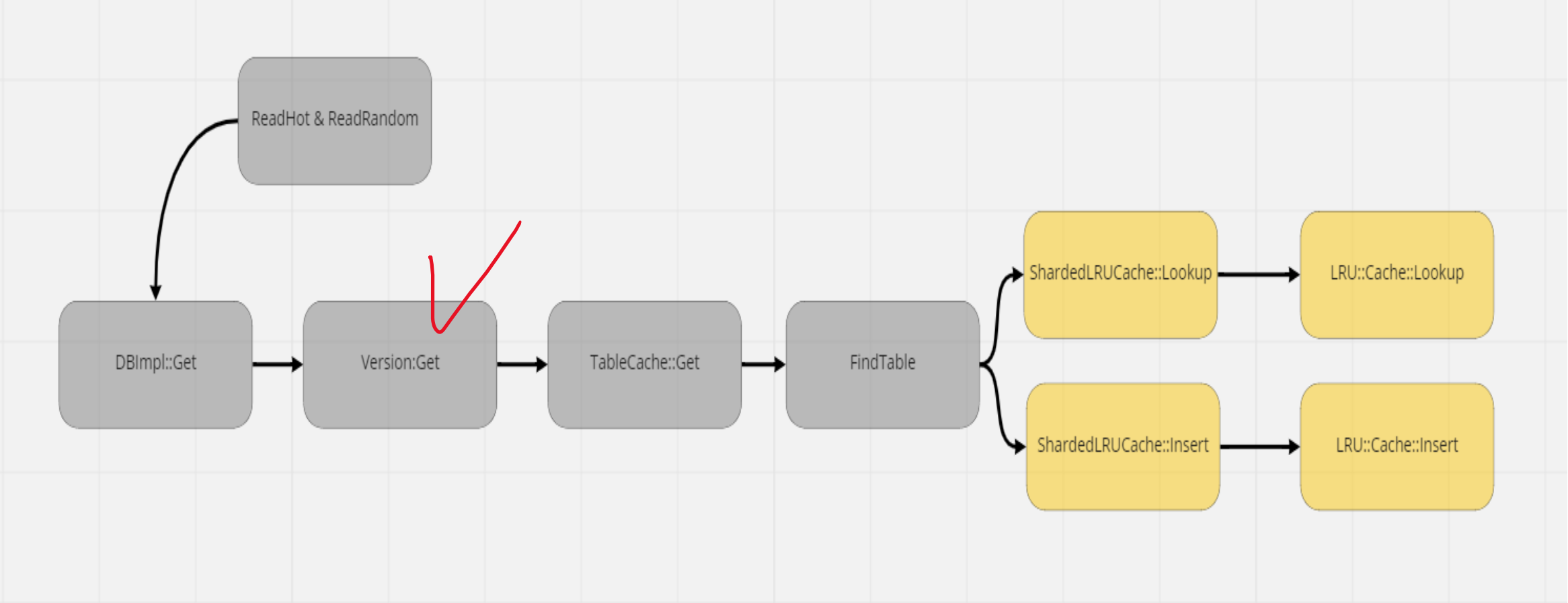
memtable.cc ~/lev
memtable.cc ~/lev
MemTable::Get(c

If not found, then look in the sstable file

```
  s = current->Get(options, lkey, value, &stats);
~/leveldb_release/db - Definitions (2)
}
tus Version::Get(const ReadOptions& options, const Loc
```

version_set.cc ~/l
version_set.cc ~/l
Version::Get(co

Cache flow analysis



Cache flow analysis

```
Iterator* Version::NewConcatenatingIterator(const ReadOptions& options,
                                            int level) const {
  return NewTwoLevelIterator(
      new LevelFileNumIterator(vset_->icmp_, &files_[level]), &GetFileIterator,
      vset_->table_cache_, options);
}
```

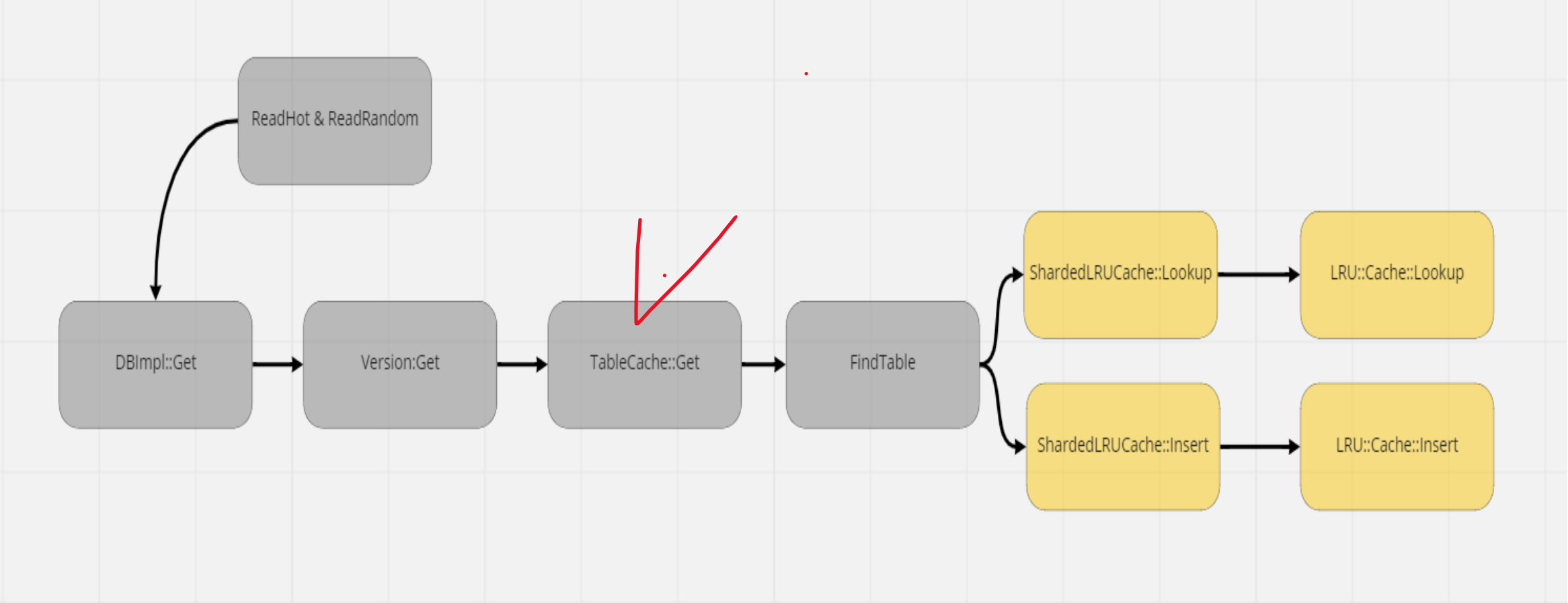
```
void Version::AddIterators(const ReadOptions& options,
                           std::vector<Iterator*>* iters) {
  // Merge all level zero files together since they may overlap
  for (size_t i = 0; i < files_[0].size(); i++) {
    iters->push_back(vset_->table_cache_->NewIterator(
        options, files_[0][i]->number, files_[0][i]->file_size));
  }
```

Cache flow analysis

```
Status Version::Get(const ReadOptions& options, const LookupKey& k,
                    std::string* value, GetStats* stats) {
```

```
state->s = state->vset->table_cache_->Get(*state->options, f->number,
                                          f->file_size, state->ikey,
                                          &state->saver, SaveValue);
```

Cache flow analysis



Cache flow analysis

```
Status TableCache::Get(const ReadOptions& options, uint64_t file_number,
                       uint64_t file_size, const Slice& k, void* arg,
                       void (*handle_result)(void*, const Slice&,
                                             const Slice&)) {
```

```
Cache::Handle* handle = nullptr;
Status s = FindTable(file_number, file_size, &handle);
if (s.ok()) {
  Table* t = reinterpret_cast<TableAndFile*>(cache_->Value(handle))->table;
  s = t->InternalGet(options, k, arg, handle_result);
  cache_->Release(handle);
}
return s;
```

Cache flow analysis

```
Status Table::InternalGet(const ReadOptions& options, const Slice& k, void* arg,
                          void (*handle_result)(void*, const Slice&,
                                                const Slice&)) {
```

```
Iterator* iiter = rep_->index_block->NewIterator(rep_->options.comparator);
```

```
if (filter != nullptr && handle.DecodeFrom(&handle_value).ok() &&
    !filter->KeyMayMatch(handle.offset(), k)) {
  // Not found
} else {
  Iterator* block_iter = BlockReader(this, options, iiter->value());
  block_iter->Seek(k);
  if (block_iter->Valid()) {
    (*handle_result)(arg, block_iter->key(), block_iter->value());
  }
  s = block_iter->status();
  delete block_iter;
```

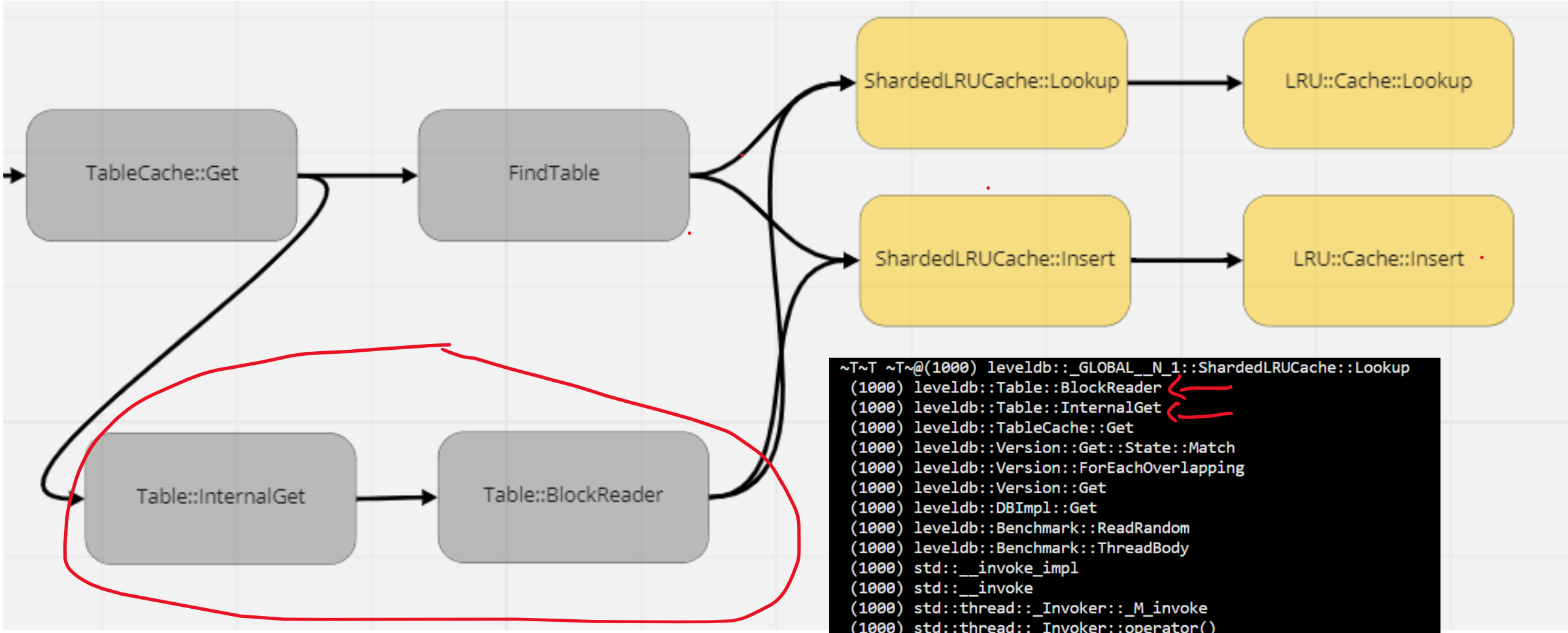
Cache flow analysis

```
Iterator* Table::BlockReader(void* arg, const ReadOptions& options,
                             const Slice& index_value) {
```

```
cache_handle = block_cache->Lookup(key);
```

```
block_cache->Insert(key, block, block->size(),
                    &DeleteCachedBlock);
```

Cache flow analysis




```
~T~T ~T~@(1000) leveldb::_GLOBAL__N_1::ShardedLRUCache::Lookup
 (1000) leveldb::Table::BlockReader
 (1000) leveldb::Table::InternalGet
 (1000) leveldb::TableCache::Get
 (1000) leveldb::Version::Get::State::Match
 (1000) leveldb::Version::ForEachOverlapping
 (1000) leveldb::Version::Get
 (1000) leveldb::DBImpl::Get
 (1000) leveldb::Benchmark::ReadRandom
 (1000) leveldb::Benchmark::ThreadBody
 (1000) std::__invoke_impl
 (1000) std::__invoke
 (1000) std::thread::_Invoker::_M_invoke
 (1000) std::thread::_Invoker::operator()
 (1000) std::thread::_State_impl::_M_run
```

Result

```
~T~\ ~T~@(1000) leveldb::_GLOBAL__N_1::ShardedLRUCache::Lookup
~T~B (1000) leveldb::TableCache::FindTable
~T~B (1000) leveldb::TableCache::Get
~T~B (1000) leveldb::Version::Get::State::Match
~T~B (1000) leveldb::Version::ForEachOverlapping
~T~B (1000) leveldb::Version::Get
~T~B (1000) leveldb::DBImpl::Get
~T~B (1000) leveldb::Benchmark::ReadRandom
```

```
~T~T ~T~@(1000) leveldb::_GLOBAL__N_1::ShardedLRUCache::Lookup
 (1000) leveldb::Table::BlockReader
 (1000) leveldb::Table::InternalGet
 (1000) leveldb::TableCache::Get
 (1000) leveldb::Version::Get::State::Match
 (1000) leveldb::Version::ForEachOverlapping
 (1000) leveldb::Version::Get
 (1000) leveldb::DBImpl::Get
 (1000) leveldb::Benchmark::ReadRandom
```

```
Status TableCache::FindTable(uint64_t file_number, uint64_t file_size,
                             Cache::Handle** handle) {
```

```
Iterator* Table::BlockReader(void* arg, const ReadOptions& options,
                             const Slice& index_value) {
```

```
  *handle = cache_->Lookup(key);
```

```
 cache_handle = block_cache->Lookup(key);
```