RocksDB Brownbag: Write Paths

Siying Dong

9/11/2020

What does RocksDB do in write path?

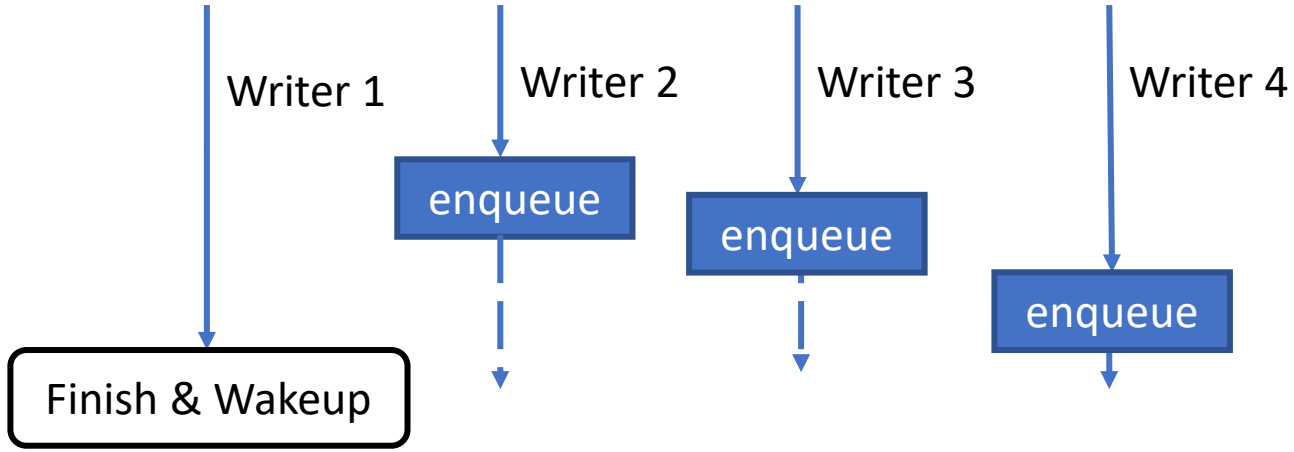
- Write to WAL
- Write to Memtable

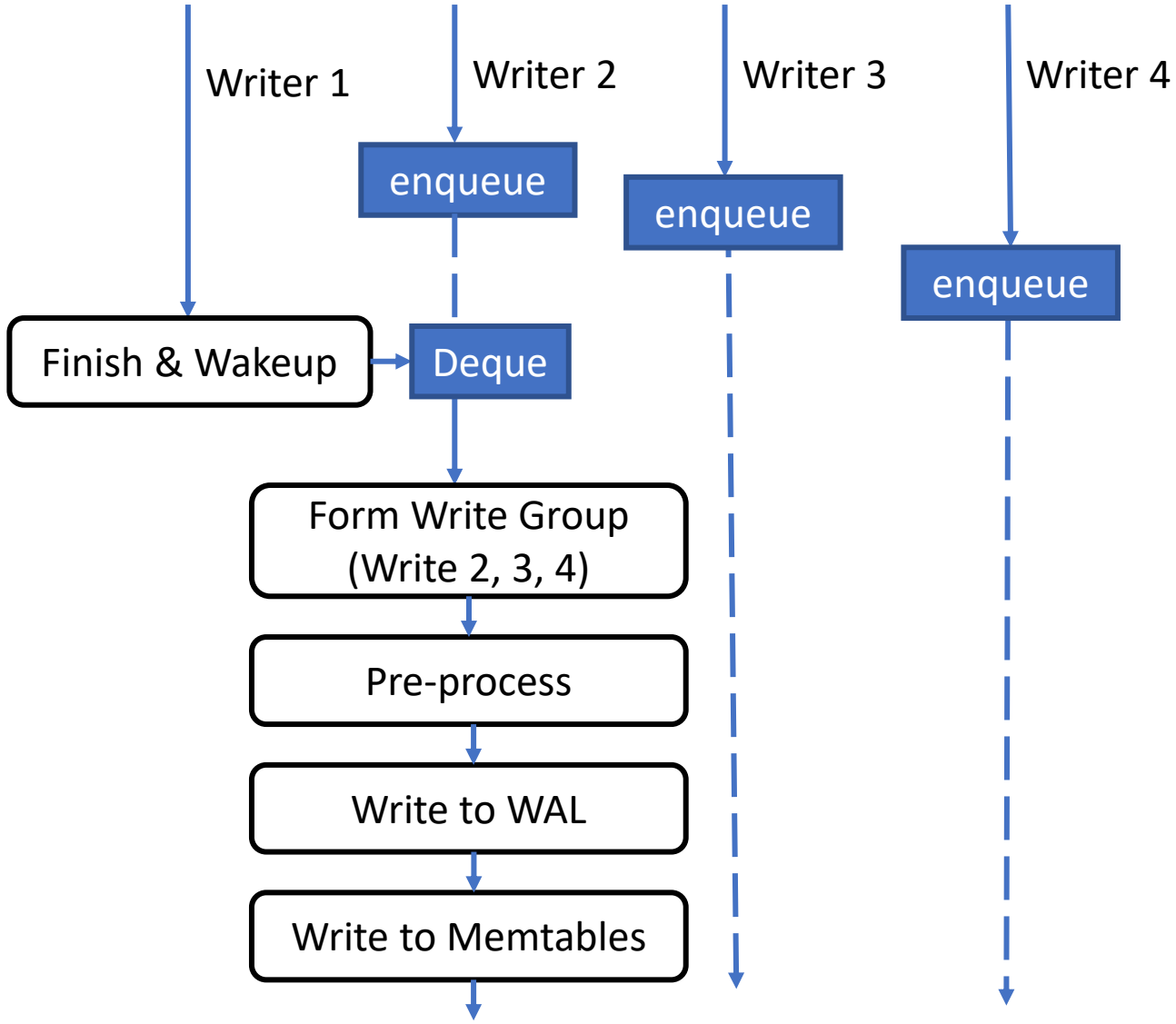
Why is write path complex?

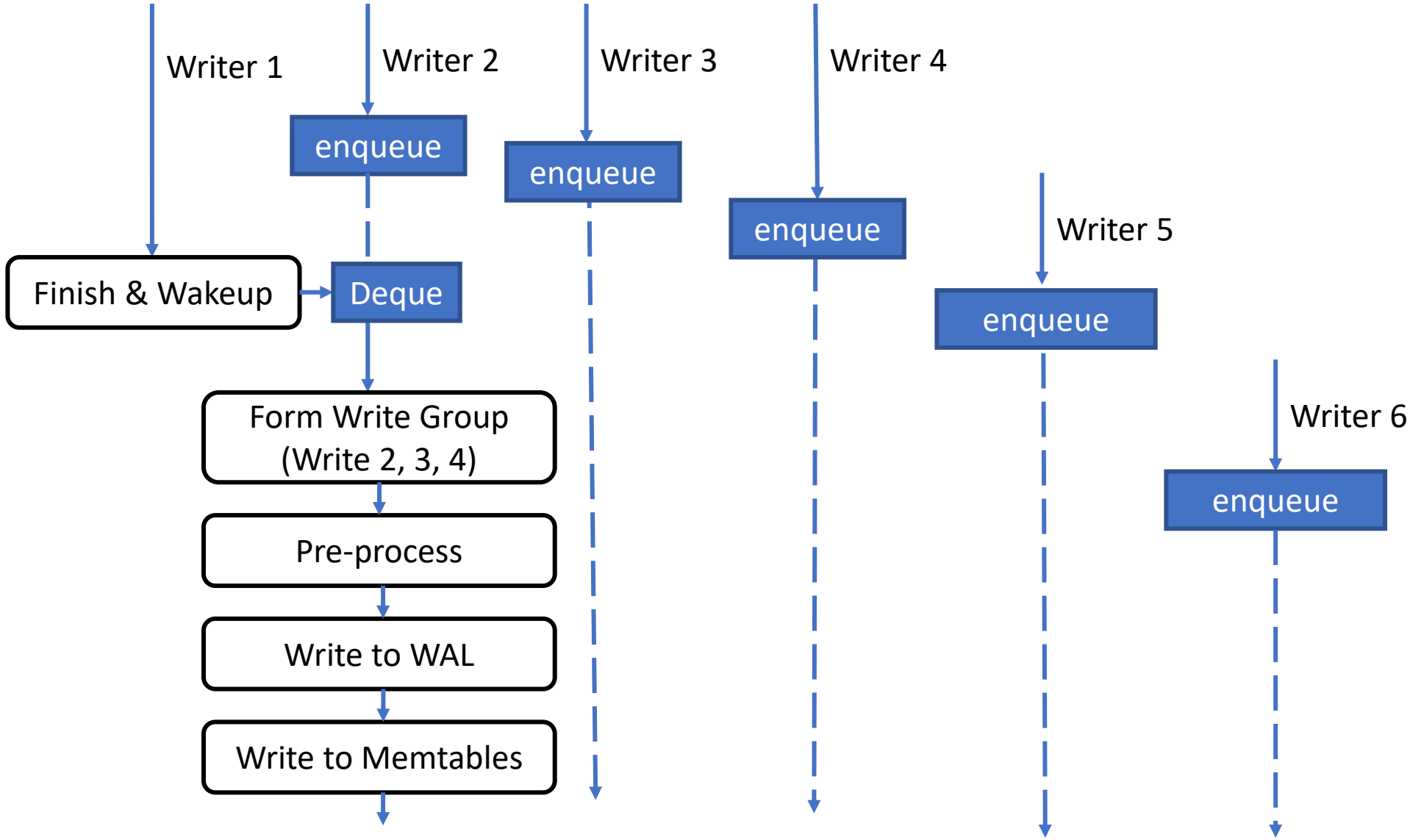
- Writing to WAL and memtable are not fully parallelizable
- Batching write (group commit)
- Pipelining
- Two-Phase-Commit (2PC)
- Unordered write

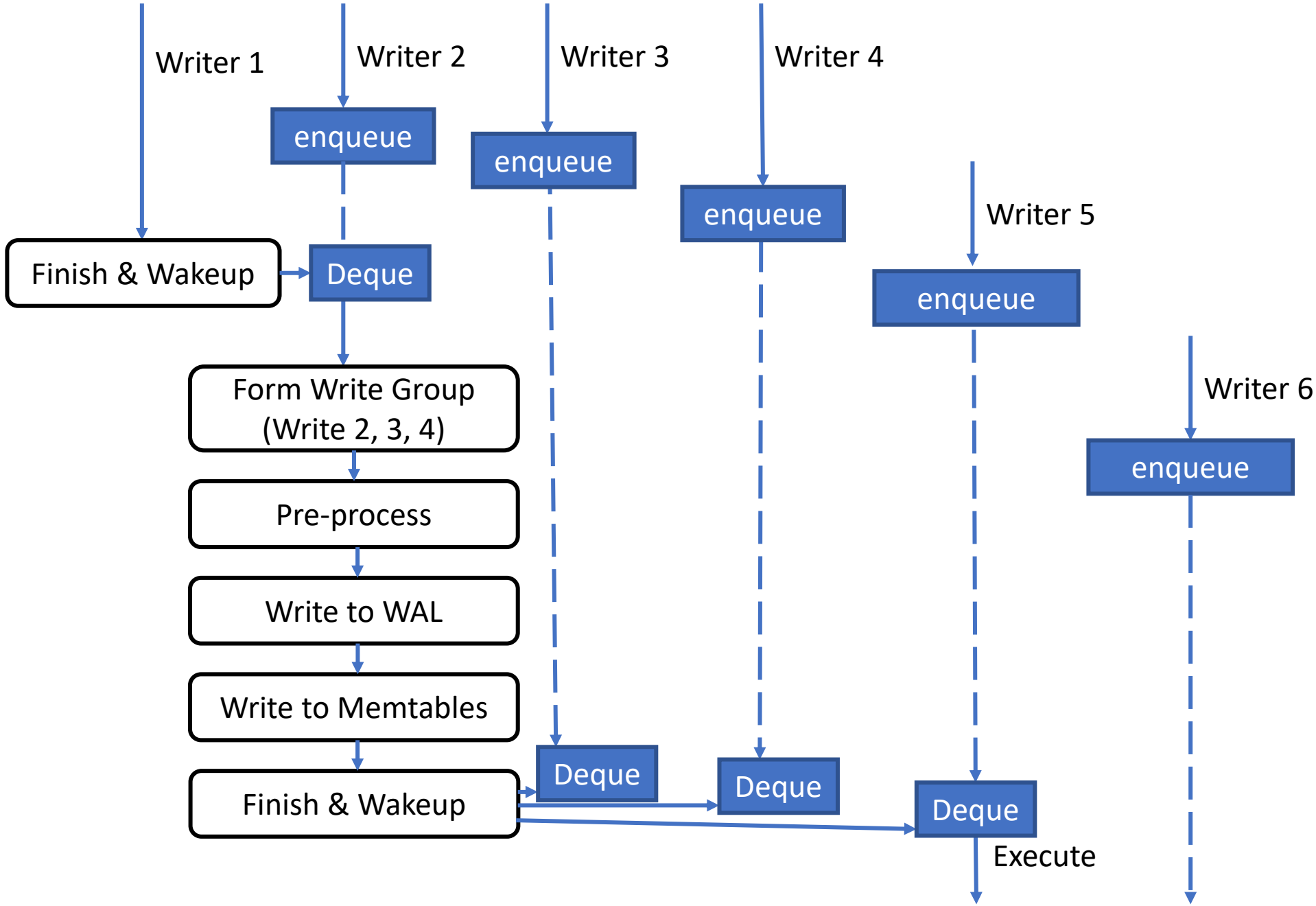
Where is it?

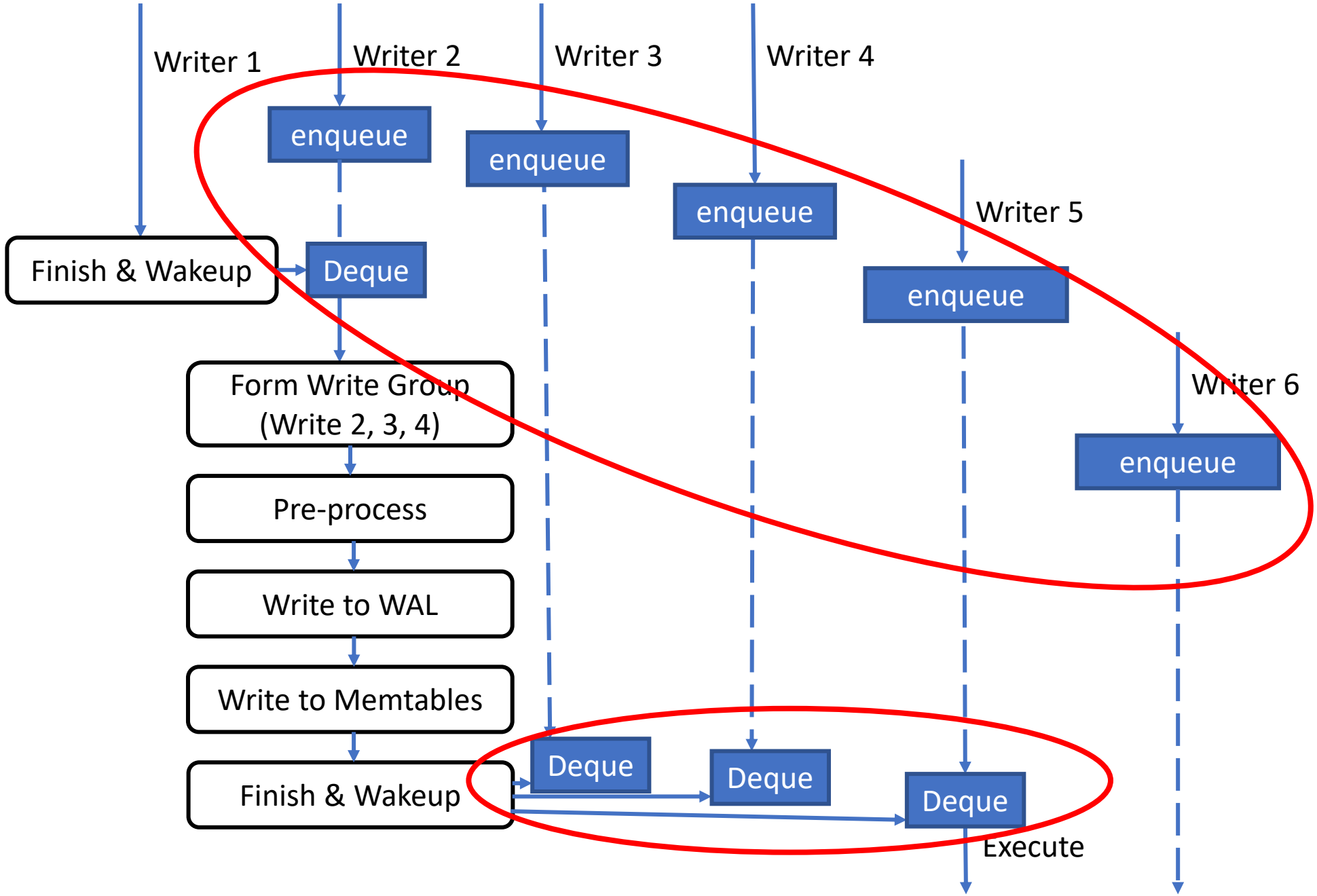
Function `DBImpl::WriteImpl()`


Writer 1
Writer 2
Writer 3
Writer 4
enqueue
enqueue
enqueue
Finish & Wakeup

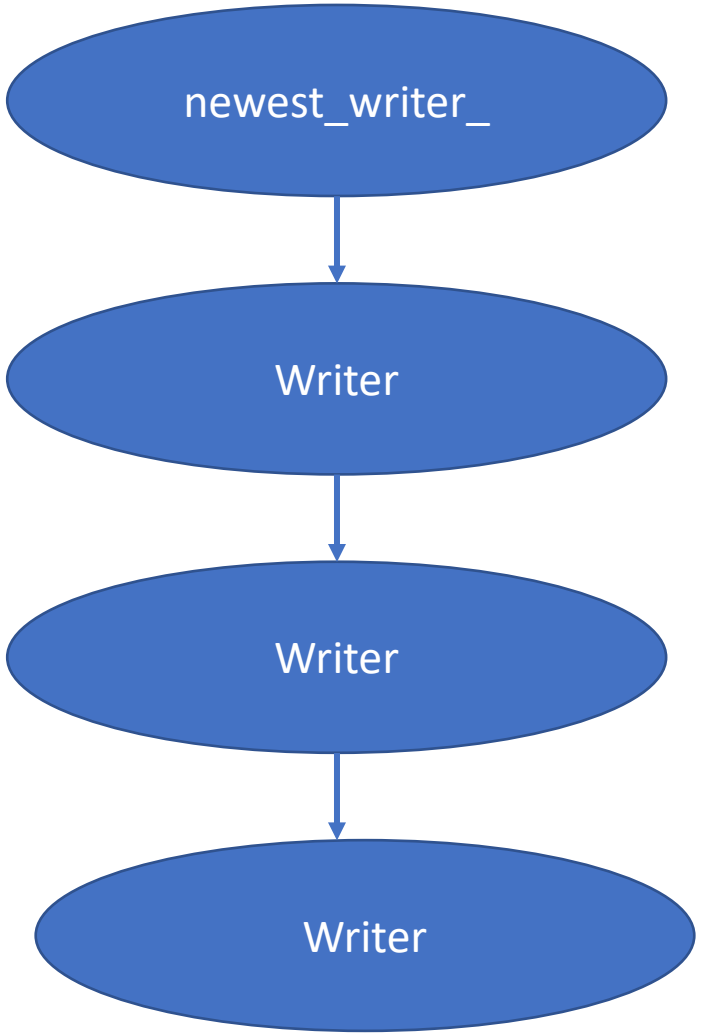

Writer 1
Writer 2
Writer 3
Writer 4
enqueue
enqueue
enqueue
Finish & Wakeup
Deque
Form Write Group
(Write 2, 3, 4)
Pre-process
Write to WAL
Write to Memtables


Writer 1
Writer 2
Writer 3
Writer 4
Writer 5
Writer 6
enqueue
enqueue
enqueue
enqueue
enqueue
Finish & Wakeup
Deque
Form Write Group
(Write 2, 3, 4)
Pre-process
Write to WAL
Write to Memtables

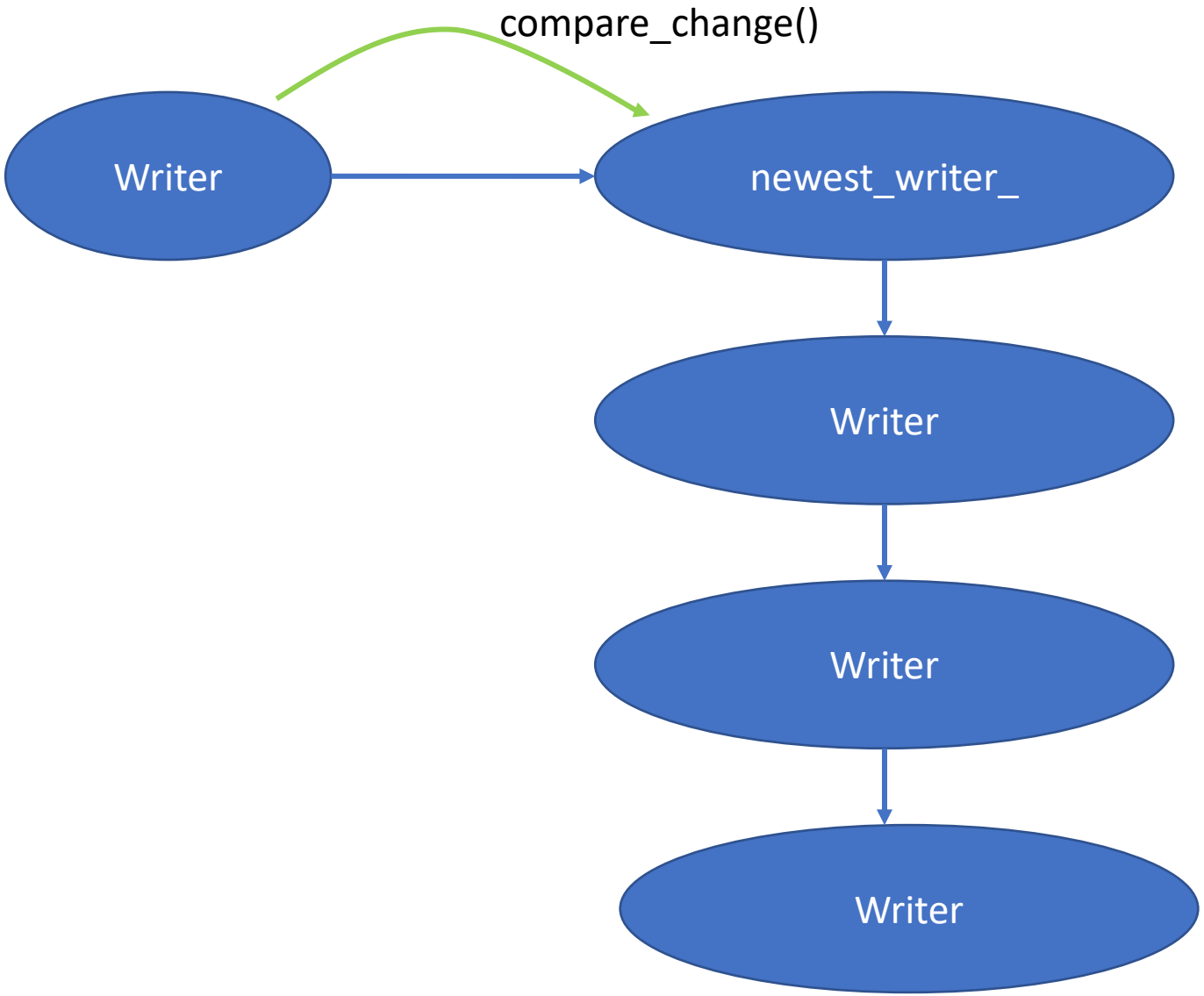

Writer 1
Writer 2
Writer 3
Writer 4
Writer 5
Writer 6
enqueue
enqueue
enqueue
enqueue
enqueue
Finish & Wakeup
Deque
Form Write Group
(Write 2, 3, 4)
Pre-process
Write to WAL
Write to Memtables
Finish & Wakeup
Deque
Deque
Deque
Execute


Writer 1
Writer 2
Writer 3
Writer 4
Writer 5
Writer 6
enqueue
enqueue
enqueue
enqueue
enqueue
Finish & Wakeup
Deque
Form Write Group
(Write 2, 3, 4)
Pre-process
Write to WAL
Write to Memtables
Finish & Wakeup
Deque
Deque
Deque
Execute

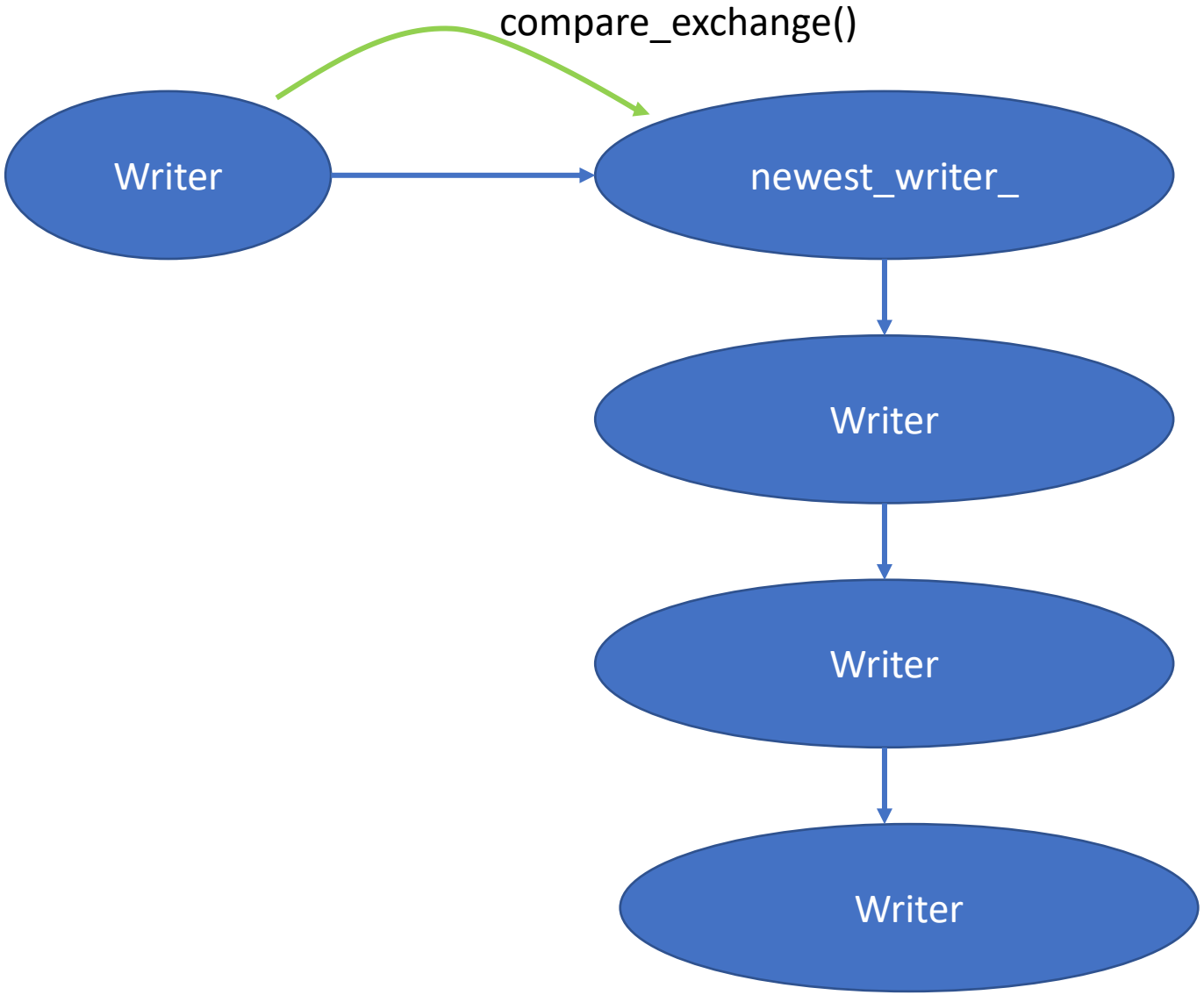
The write queue: *class* `WriteThread`



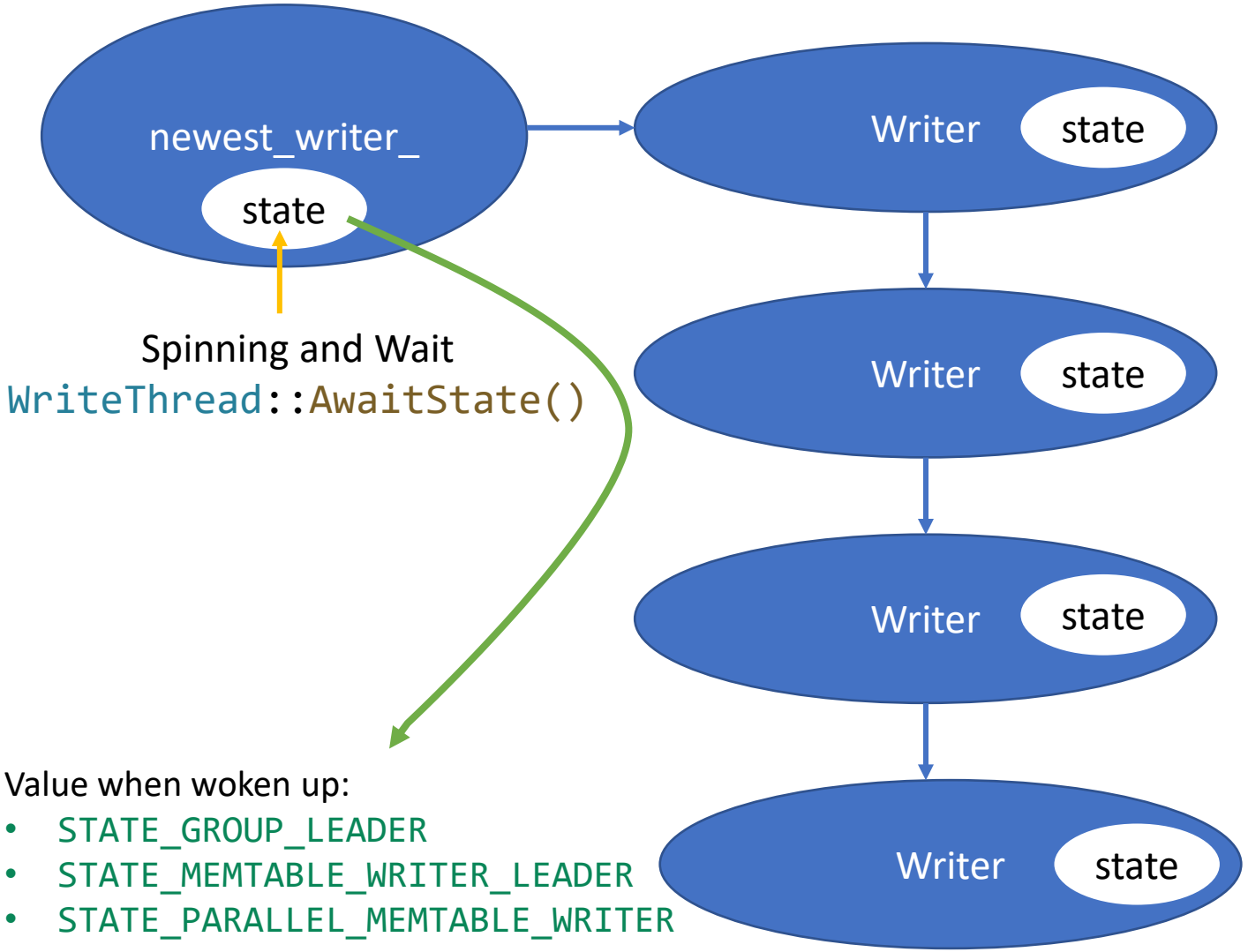
Enqueue: `WriteThread::JoinBatchGroup()`



Enqueue: `WriteThread::JoinBatchGroup()`



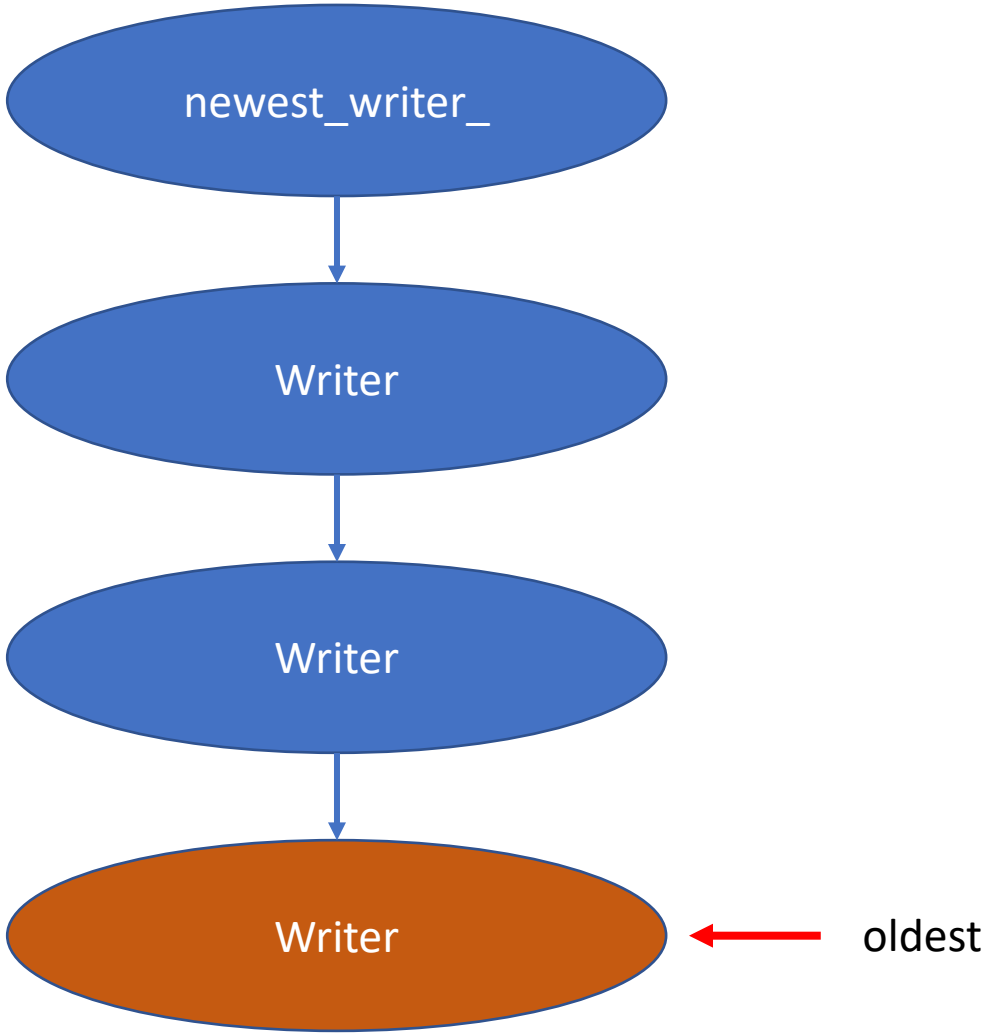
Enqueue: `WriteThread::JoinBatchGroup()`



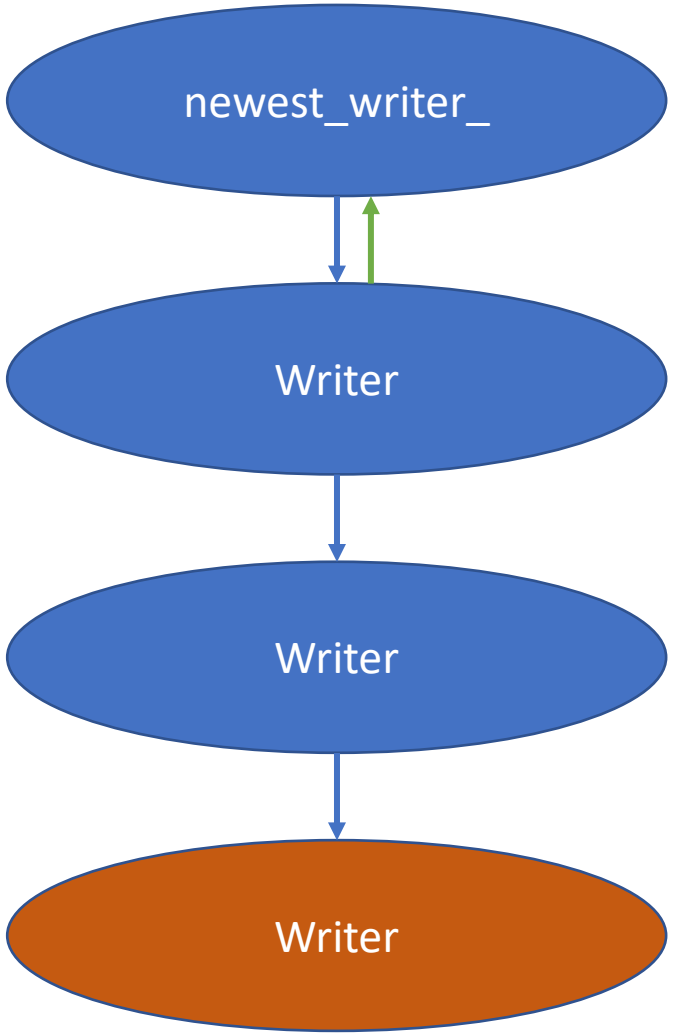
Value when woken up:

- `STATE_GROUP_LEADER`
- `STATE_MEMTABLE_WRITER_LEADER`
- `STATE_PARALLEL_MEMTABLE_WRITER`
- `STATE_COMPLETED`

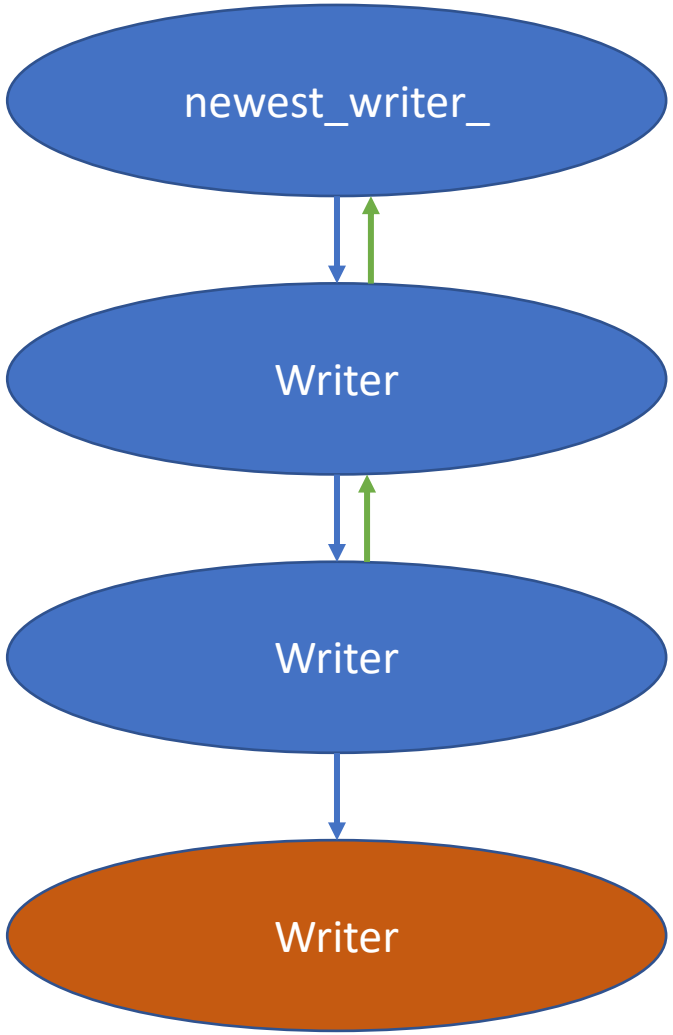
WriteThread::EnterAsBatchGroupLeader()



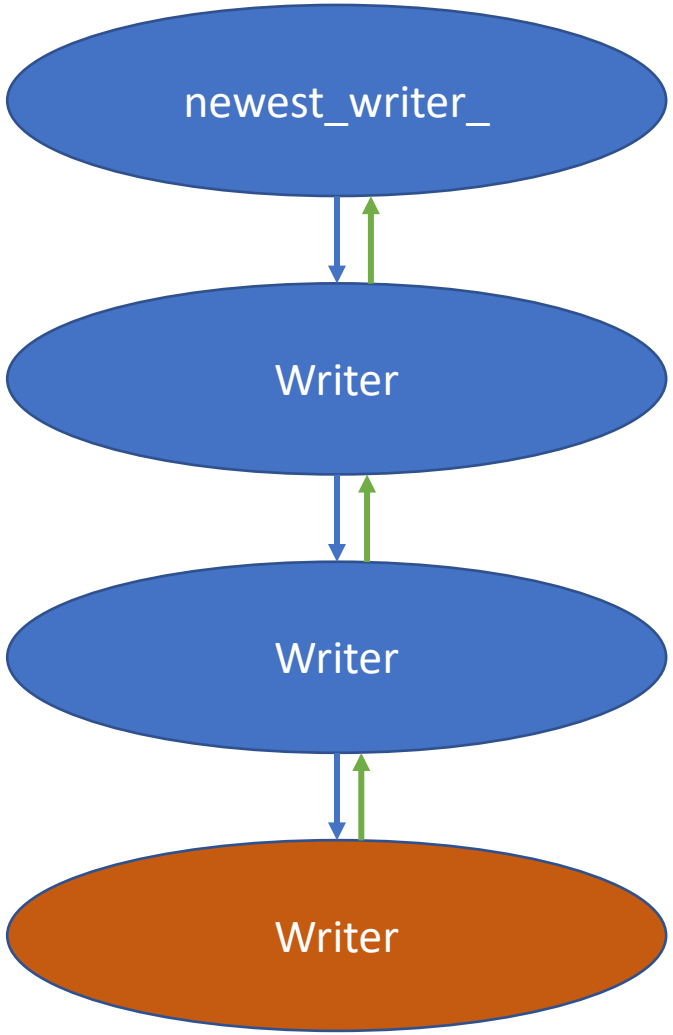
WriteThread::EnterAsBatchGroupLeader()



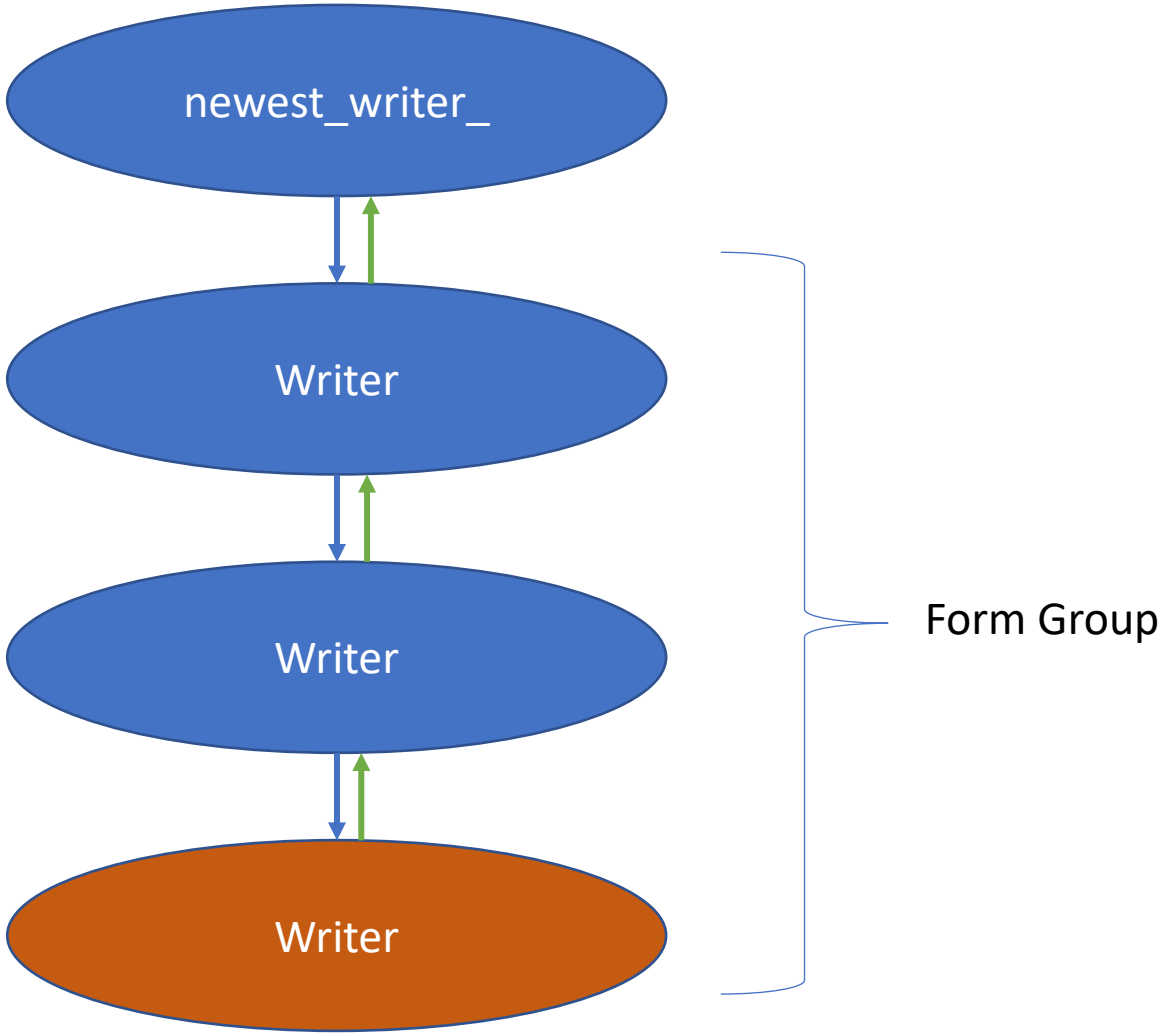
WriteThread::EnterAsBatchGroupLeader()



WriteThread::EnterAsBatchGroupLeader()

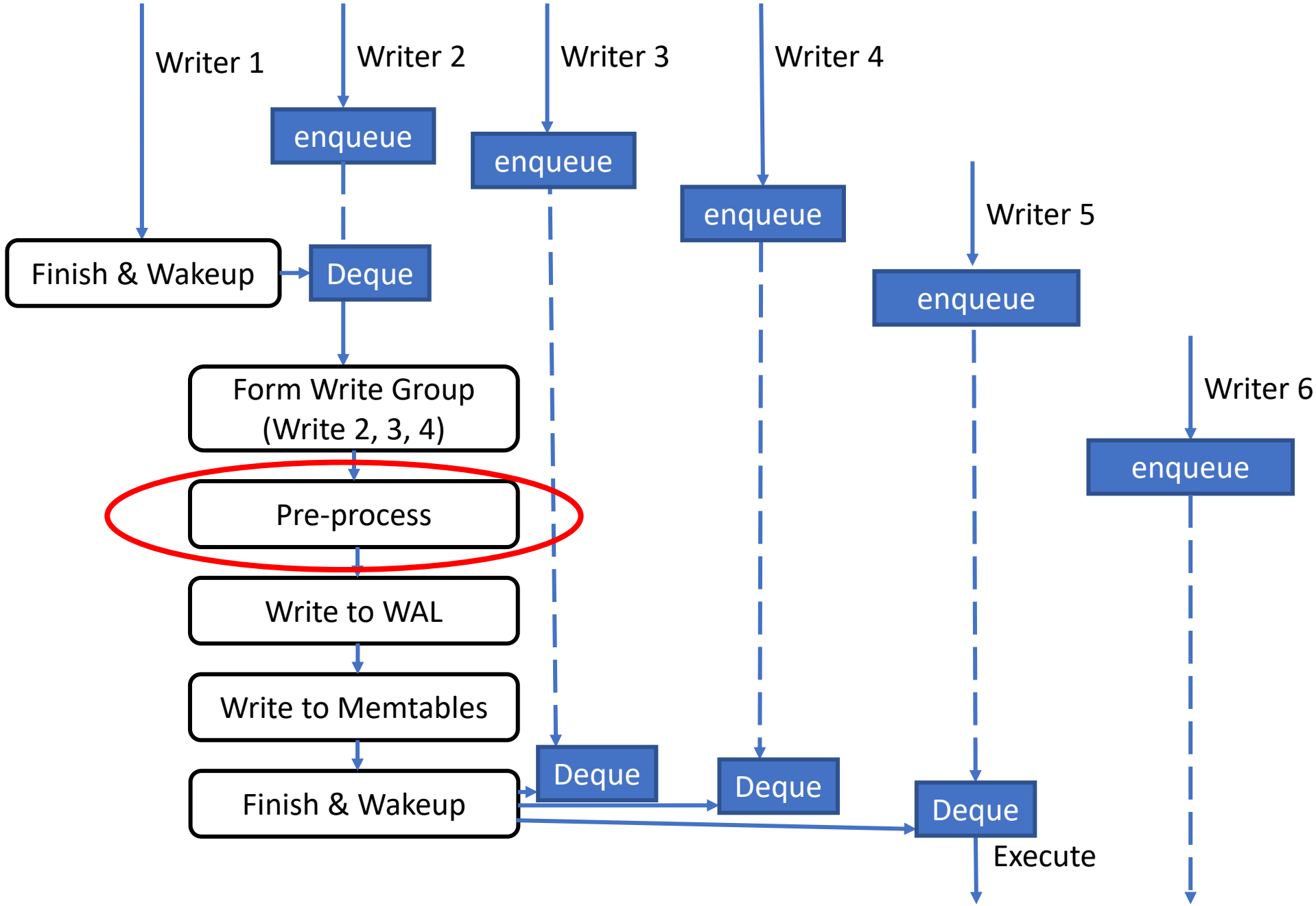


WriteThread::EnterAsBatchGroupLeader()



Terminal Conditions:

1. Max group size
2. Merge Operands

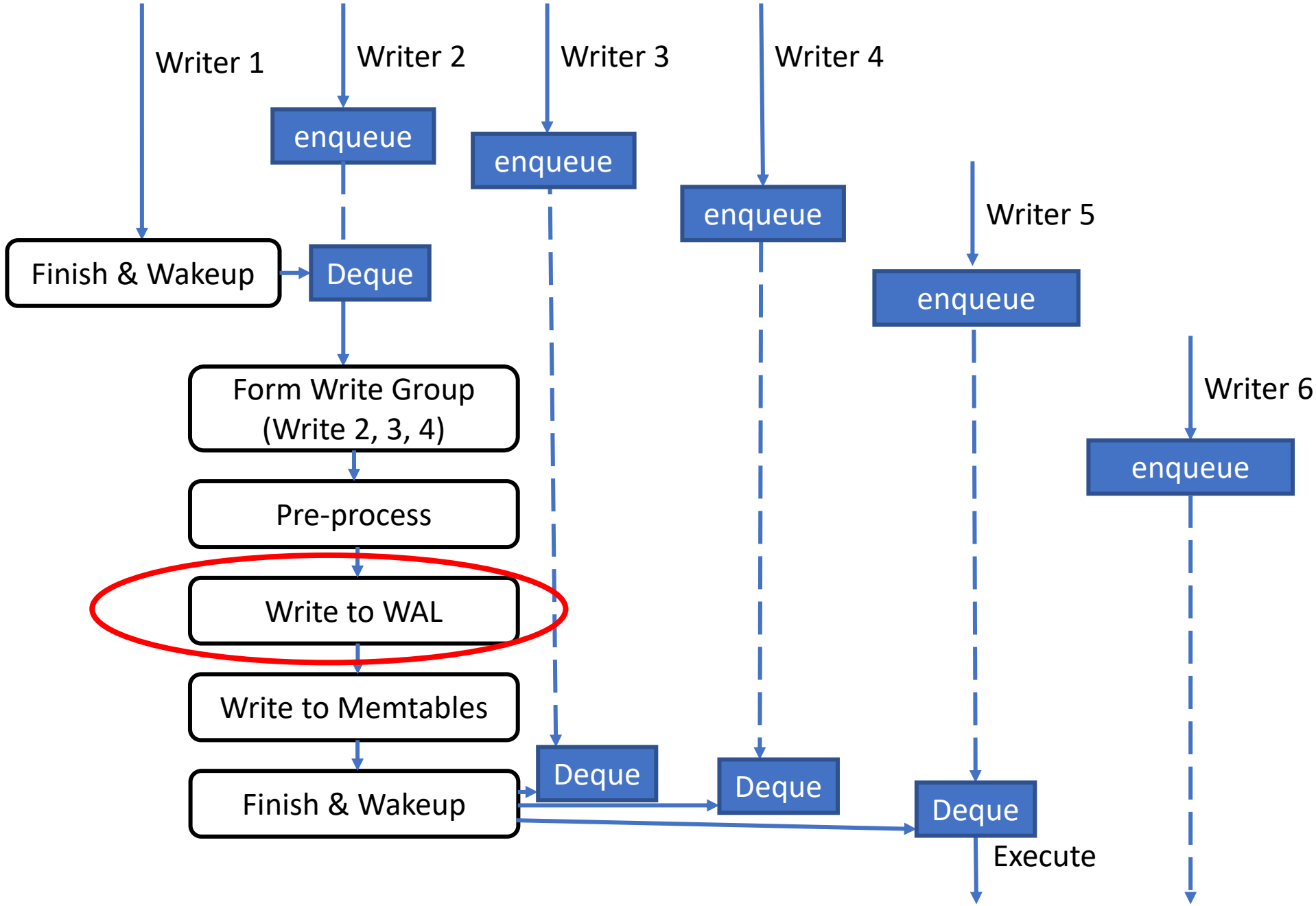

Writer 1
Writer 2
Writer 3
Writer 4
Writer 5
Writer 6
enqueue
enqueue
enqueue
enqueue
enqueue
Finish & Wakeup
Deque
Form Write Group
(Write 2, 3, 4)
Pre-process
Write to WAL
Write to Memtables
Finish & Wakeup
Deque
Deque
Deque
Execute

Pre-process: `DBImpl::PreprocessWrite()`

- Switch memtable when needed: `DBImpl::SwitchMemtable()`
  - `total_log_size_`
  - `write_buffer_manager_->ShouldFlush()`
  - Memtable full: `!flush_scheduler_.Empty()`
- Trim memtable history: `trim_history_scheduler_.Empty()`
- Throttling: checking `write_controller_`

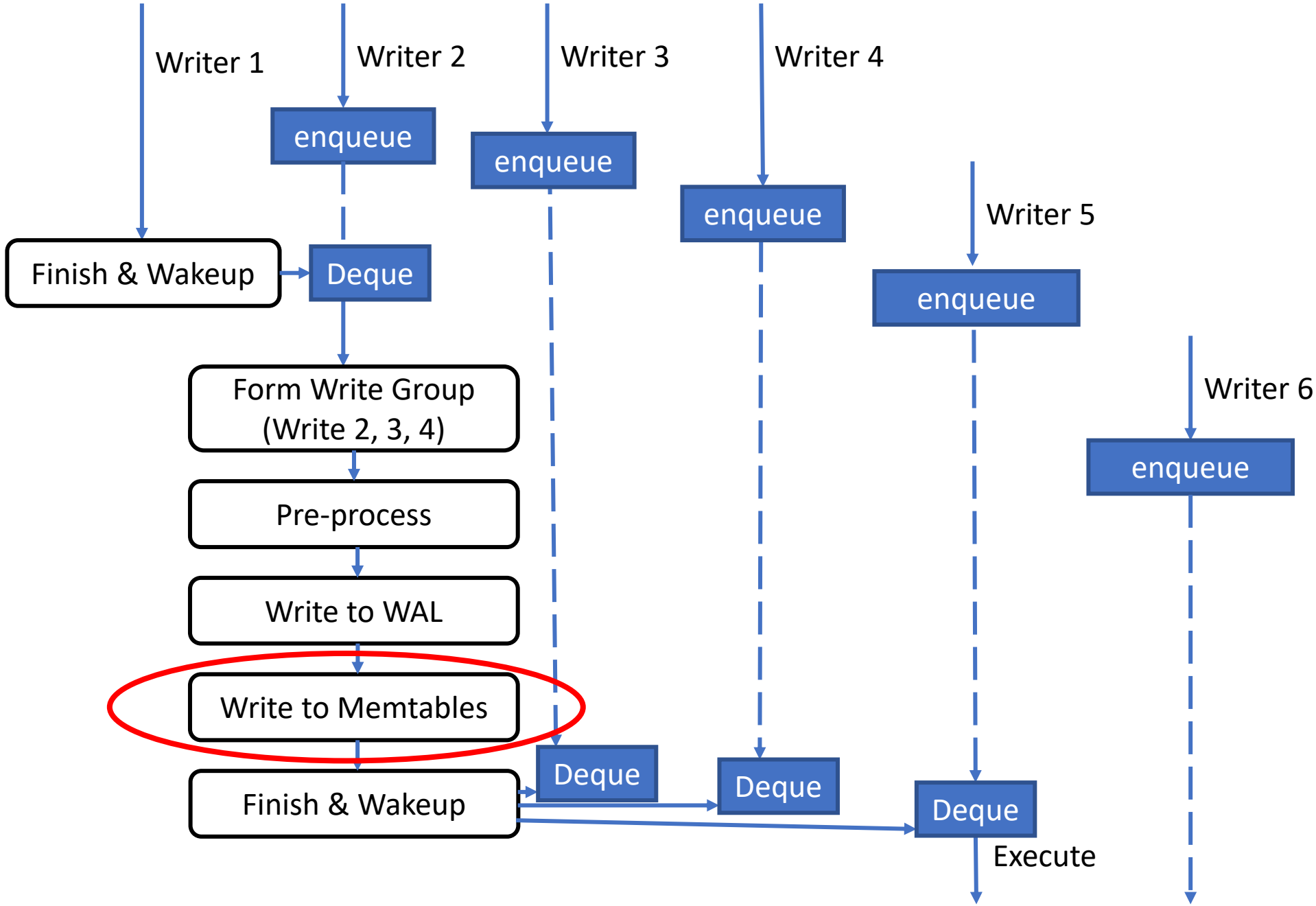
DBImpl::SwitchMemtable()

- Create new WAL file
- Create a new memtable
- Flush old WAL file buffer (to OS page cache)


Writer 1
Writer 2
Writer 3
Writer 4
Writer 5
Writer 6
enqueue
enqueue
enqueue
enqueue
enqueue
Finish & Wakeup
Deque
Form Write Group
(Write 2, 3, 4)
Pre-process
Write to WAL
Write to Memtables
Finish & Wakeup
Deque
Deque
Deque
Execute

Write to WAL

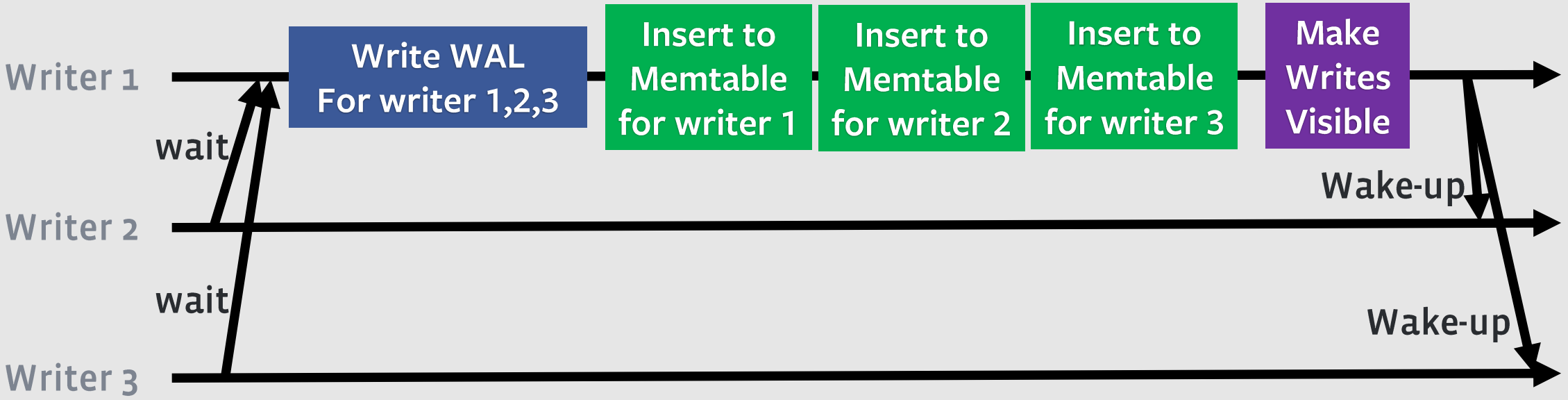
- Normal case: `DBImpl::WriteToWAL()`
- Unordered write: `DBImpl::WriteImplWALOnly()`
  (write thread queuing is done there)
- `two_write_queues_: ConcurrentWriteToWAL()`
  (synchronize through `log_write_mutex_`)


Writer 1
Writer 2
Writer 3
Writer 4
Writer 5
Writer 6
enqueue
enqueue
enqueue
enqueue
enqueue
Finish & Wakeup
Deque
Form Write Group
(Write 2, 3, 4)
Pre-process
Write to WAL
Write to Memtables
Finish & Wakeup
Deque
Deque
Deque
Execute

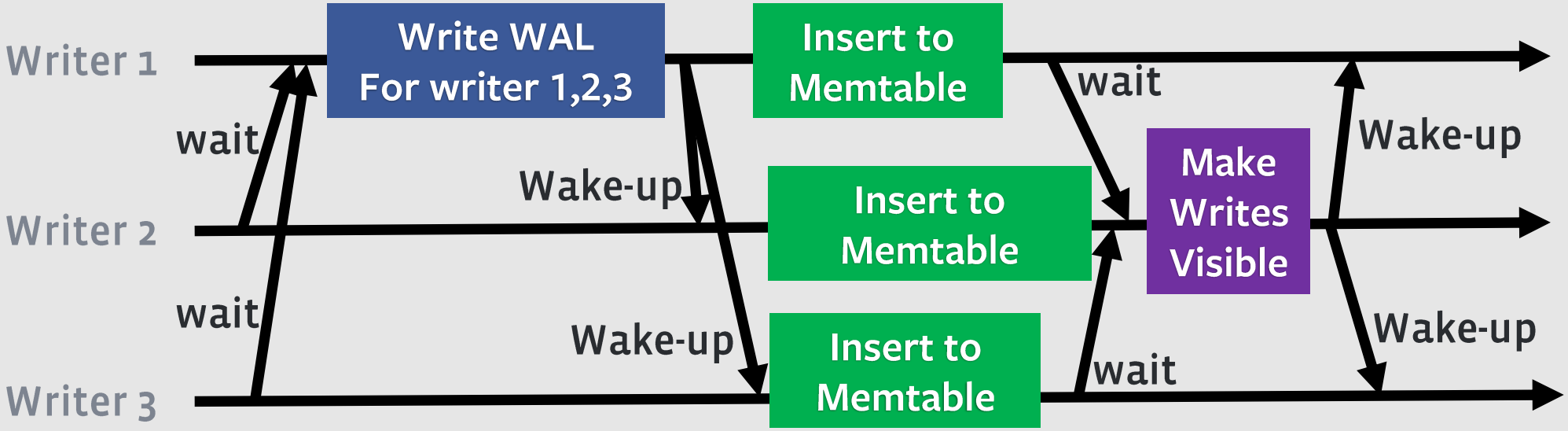
Memtable Writes: Three Modes

- Mode 1: group leader writes for all in the group
- Mode 2: concurrent memtable write
- Mode 3: unordered write: writes and return

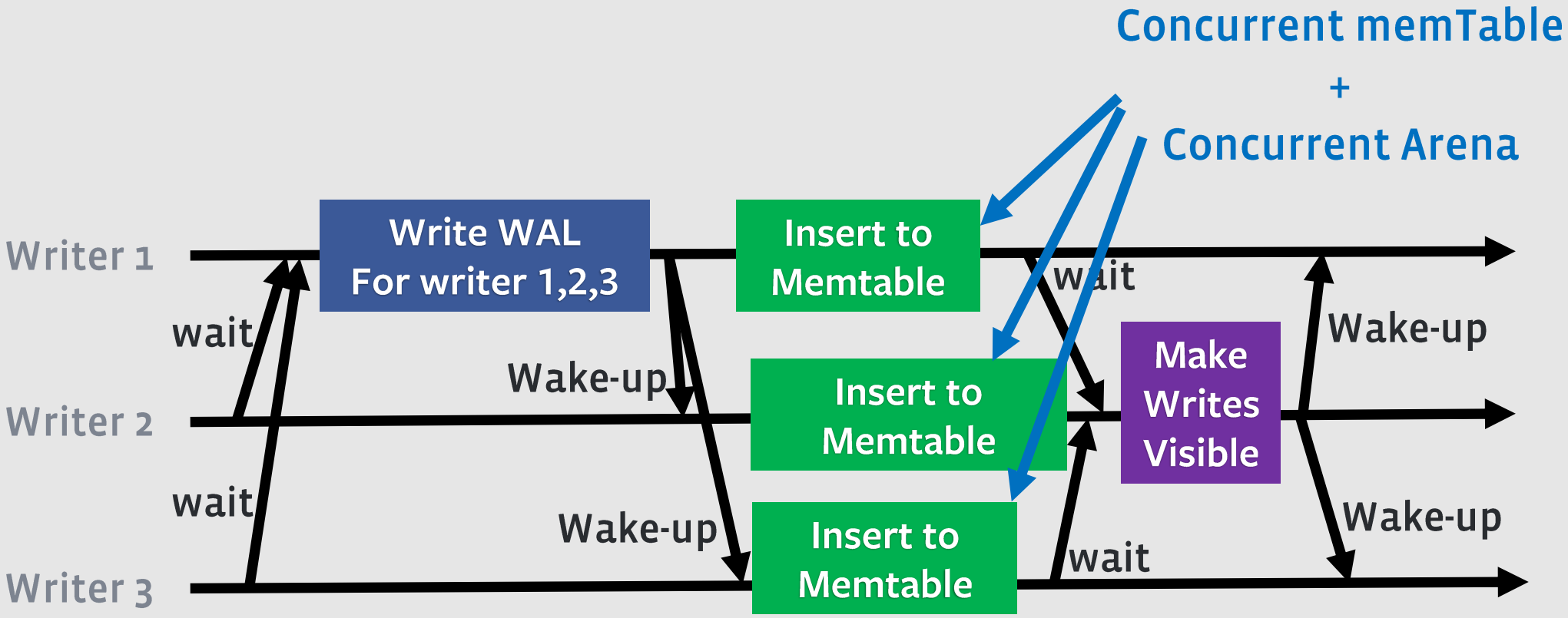
Mode 1: Group leader writes for all



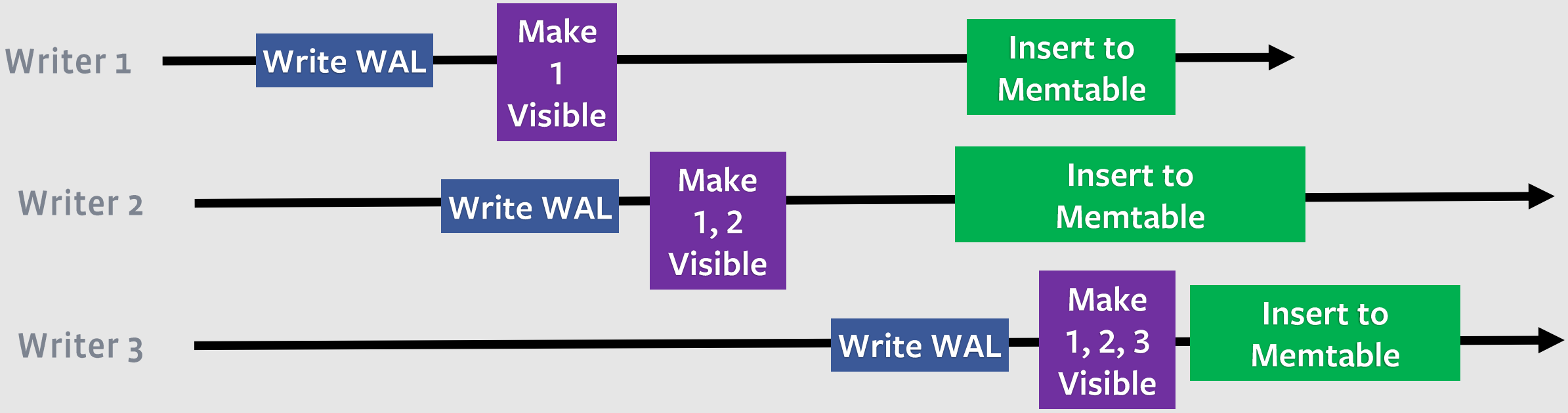
Mode 2: Concurrent memtable writes



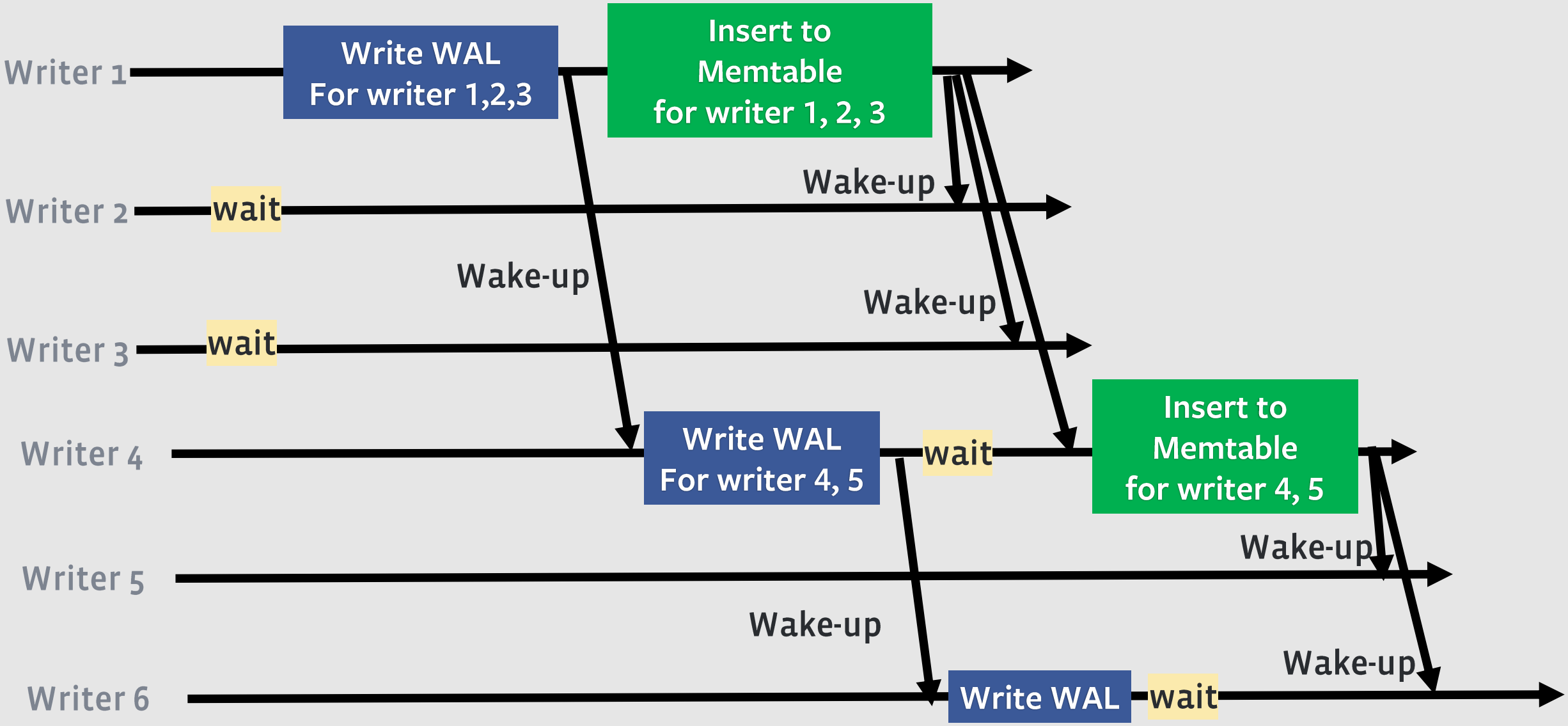
Mode 2: Concurrent memtable writes



Mode 3: Unordered Write



Pipelined Write (With Mode 1)



`WriteBatchInternal::InsertInto()`

- Go through every entry in the write batch:
  - Call `MemTable::Add()`
  - `MemTableInserter::CheckMemtableFull()`
    - `Check memtable full`
    - `Check memtable history trimming condition`

Two-Phase-Commit

	Write Committed	Write Prepared	Write UnPrepared

Before Prepare			1. Prepare Entry to WAL
			2. Write to memtable
			3. Add uncommitted seqNum to tracking data structure
			`WriteImpl(`
			`  disable_memtable = false,`
			`  add_prepared_callback)`

Prepare	Prepare Entry to WAL	1. Prepare Entry to WAL
		2. Write to memtable
		3. Add uncommitted seqNum to tracking data structure
		`WriteImpl(`
		`  disable_memtable = false,`
		`  add_prepared_callback)`	

Commit	Write to memtable
	Commit Entry to WAL
		Commit Entry to WAL:
		`WriteImpl(`
		`  disable_memtable = true,`
		`update_commit_map_with_aux_batch`
		`)`
			Commit Entry to WAL:
			`WriteImpl(`
			`  disable_memtable = true,`
			`update_commit_map_with_aux_batch)`

What's write_callback used for?

Optimistic Transaction's legacy conflict resolving mechanism.

`OptimisticTransaction::CommitWithSerialValidate()`

How should we deal with it?

Recap: Write Path