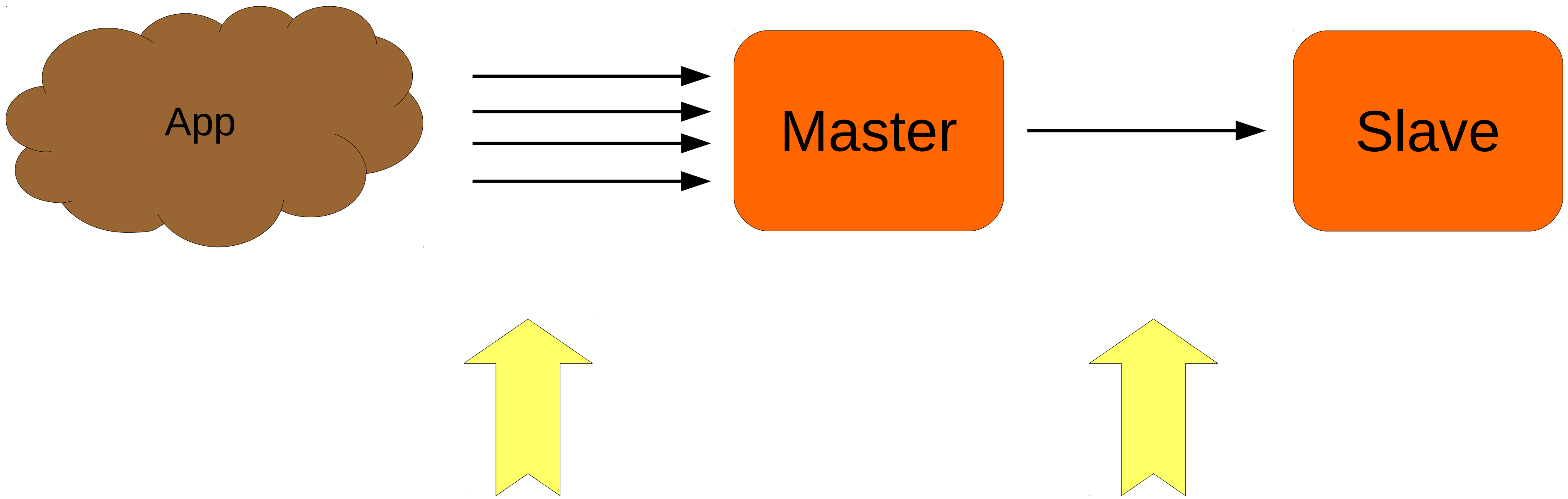
Multi-Threaded Replication in MySQL 5.6 and MySQL 5.7

Stéphane Combaudon
April 14th, 2015

Agenda

- Why multi-threaded replication?
- Performance benefits
- Positioning: GTID or not?
- GTID in a nutshell
- MySQL 5.7

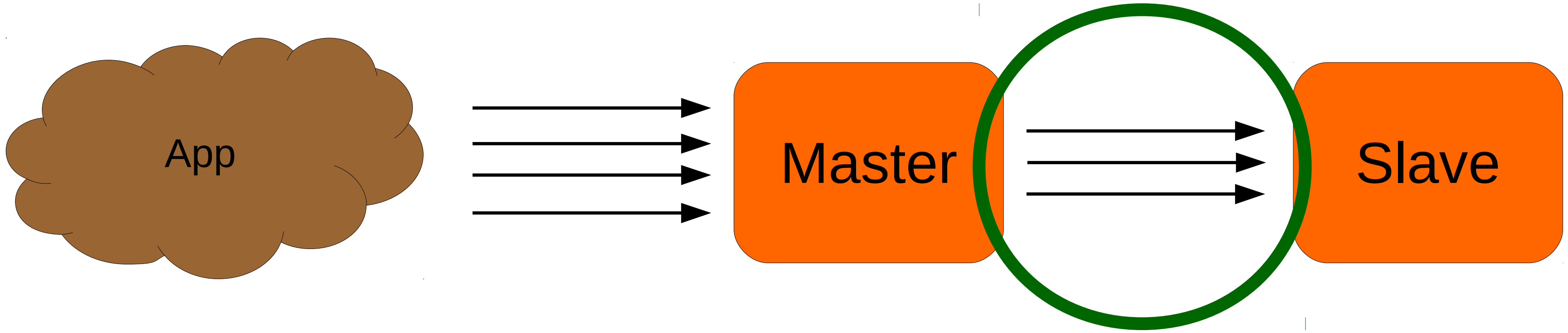
Good old replication



App can write in parallel
on the master...

… but writes are serialized
on slaves: quickly
becomes a bottleneck!

Multi-Threaded Slaves (MTS)



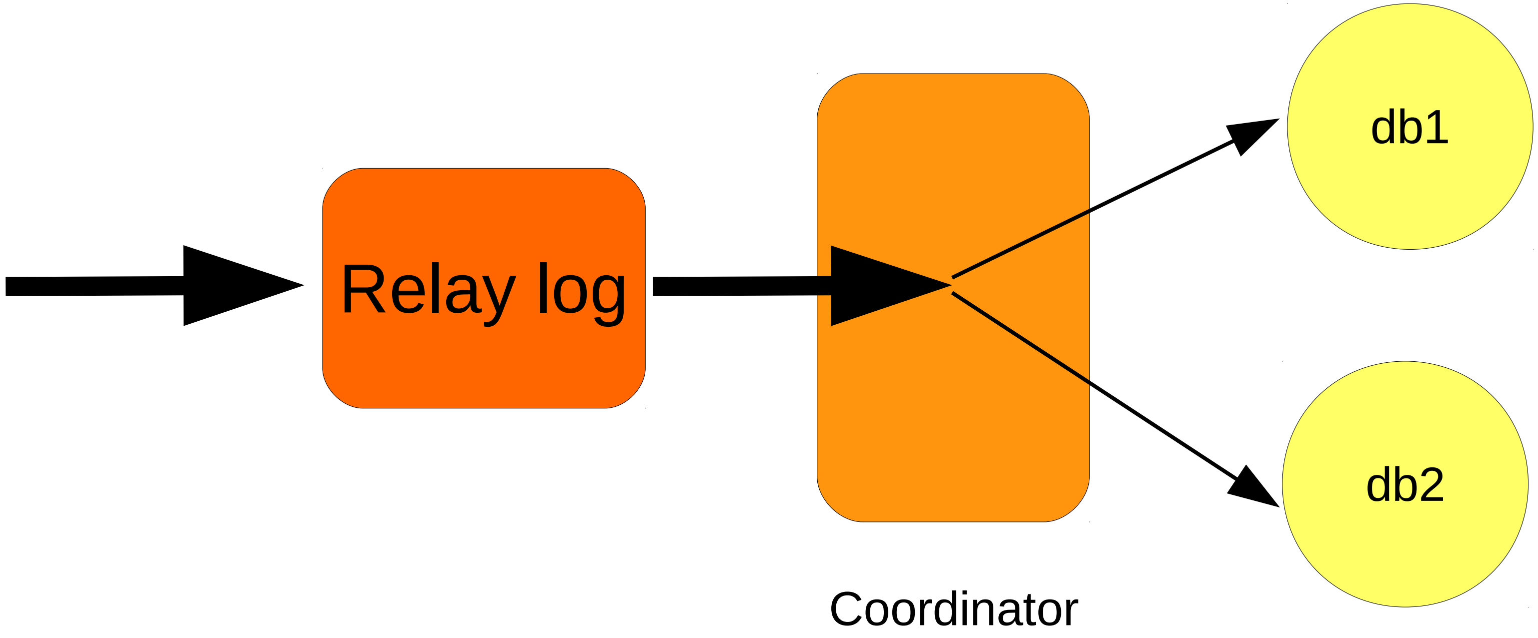
- Coordinator thread on slave dispatches work across several worker threads
  - Each worker thread commits trx in isolation

Prerequisites (5.6)

- Transactions are assumed independent only if they are executed in separate databases
  - And if there is no cross db transaction
- Using a single db? MTS 5.6 is not for you!
- Using N dbs? Use N parallel worker threads
- Use `slave_parallel_workers = N`
  - Worker threads are visible with `SHOW PROCESSLIST`

Visual Explanation

- Slave has 2 dbs and 2 worker threads



Agenda

- Why multi-threaded replication?
- Performance benefits
- Positioning: GTID or not?
- GTID in a nutshell
- MySQL 5.7

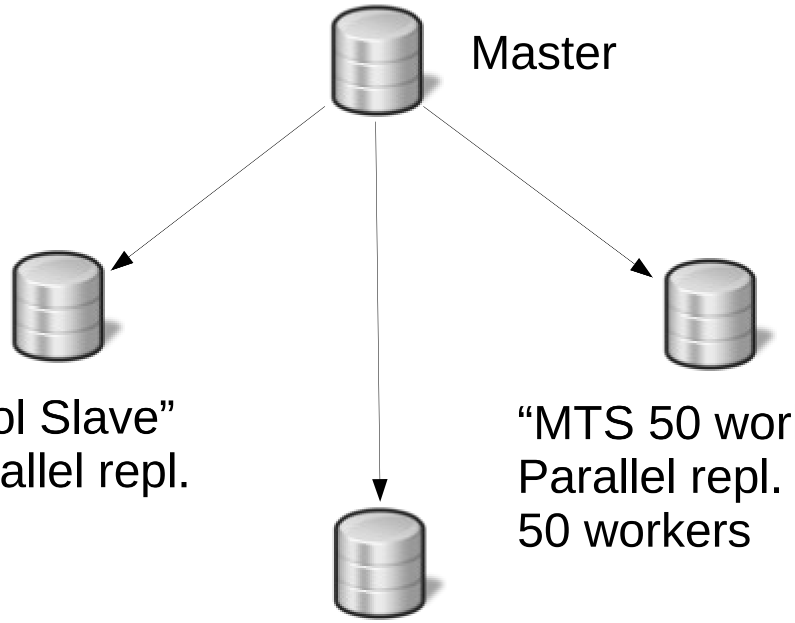
2 micro-benchmarks

- Goal: does MTS help reduce replication lag?
- Sysbench writes to 2 databases
- 3 slaves
  - 1 single-threaded slave
  - 1 MTS with 2 parallel workers
  - 1 MTS with 50 parallel workers
- 2 scenarios
  - 50% writes to each db
  - 80% writes to db1

Scenario #1: 50%/50% writes

- 4x m3.xlarge instances

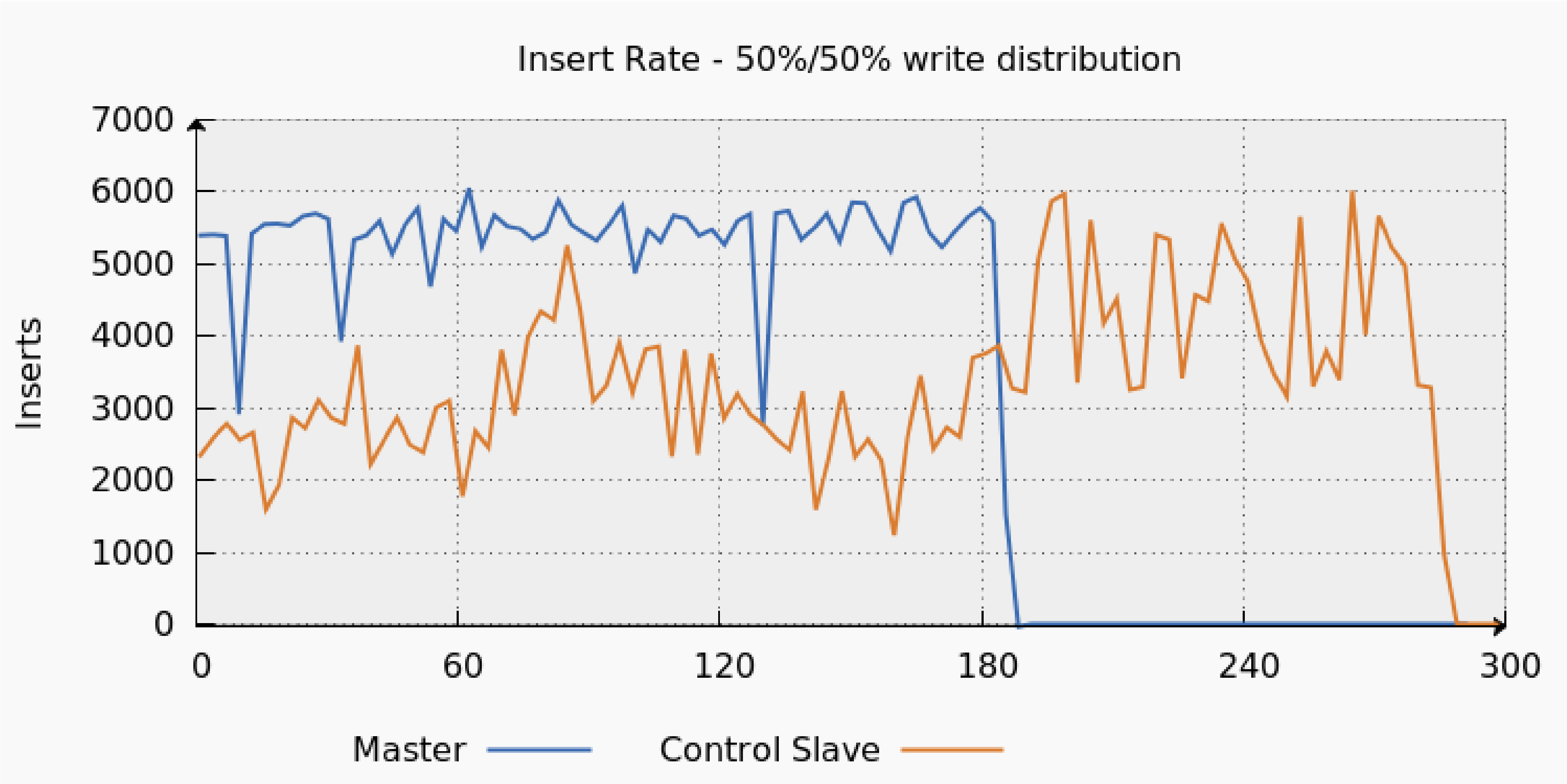
```
innodb_buffer_pool_size = 10G
innodb_log_file_size = 512M
innodb_flush_log_at_trx_commit
-> 1 (master)
-> 2 (slaves)

GTID-replication enabled
```



Two sysbench runs executed concurrently on the master:

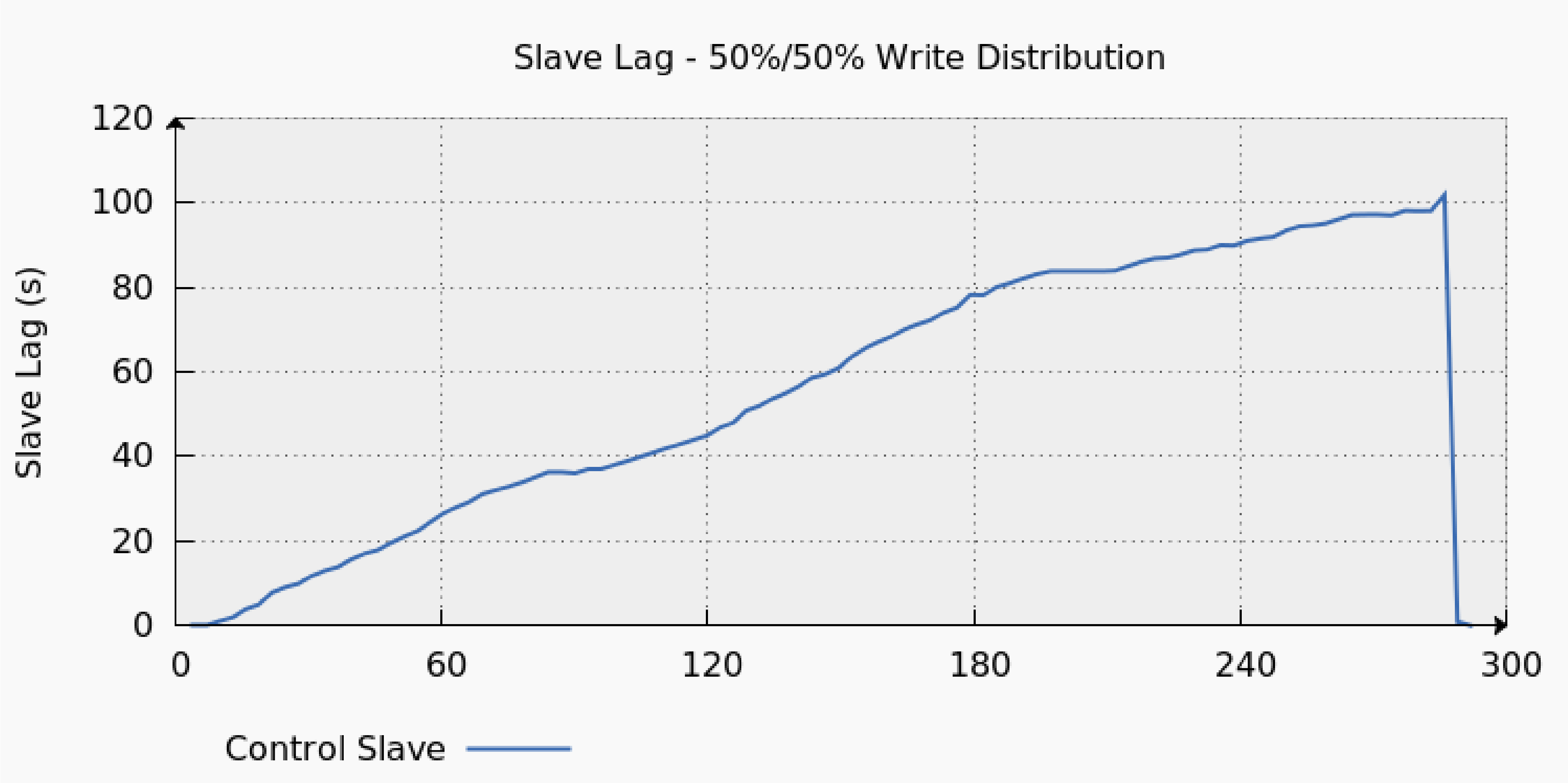
```
# sysbench --mysql-user=root --mysql-db=db1 --test=insert.lua --max-requests=100000
--num-threads=15 --oltp-tables-count=16 run

# sysbench --mysql-user=root --mysql-db=db2 --test=insert.lua --max-requests=100000
--num-threads=15 --oltp-tables-count=16 run
```

Master + Control Slave



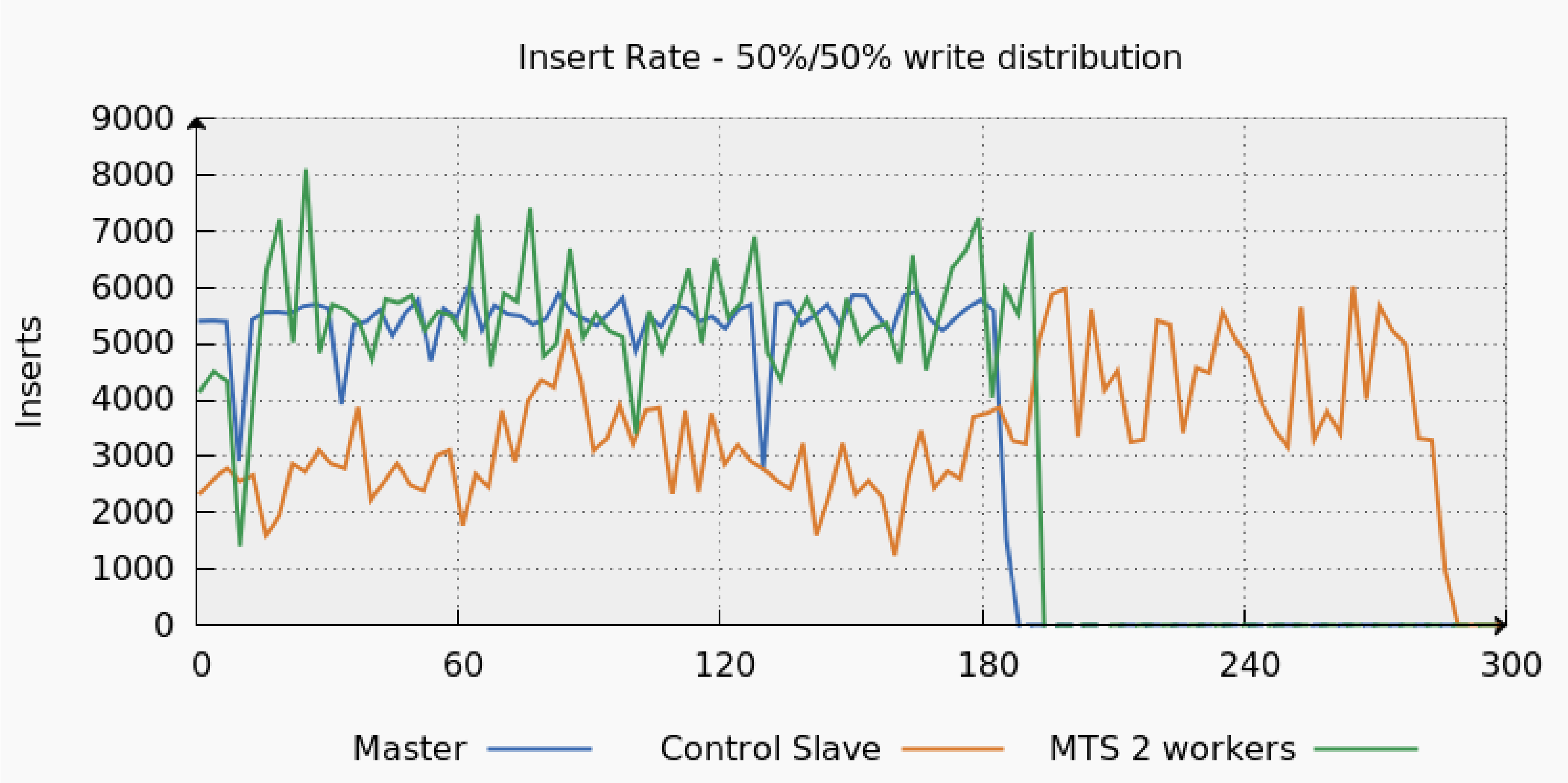
No surprise: the slave is not able to keep up

Replication Lag



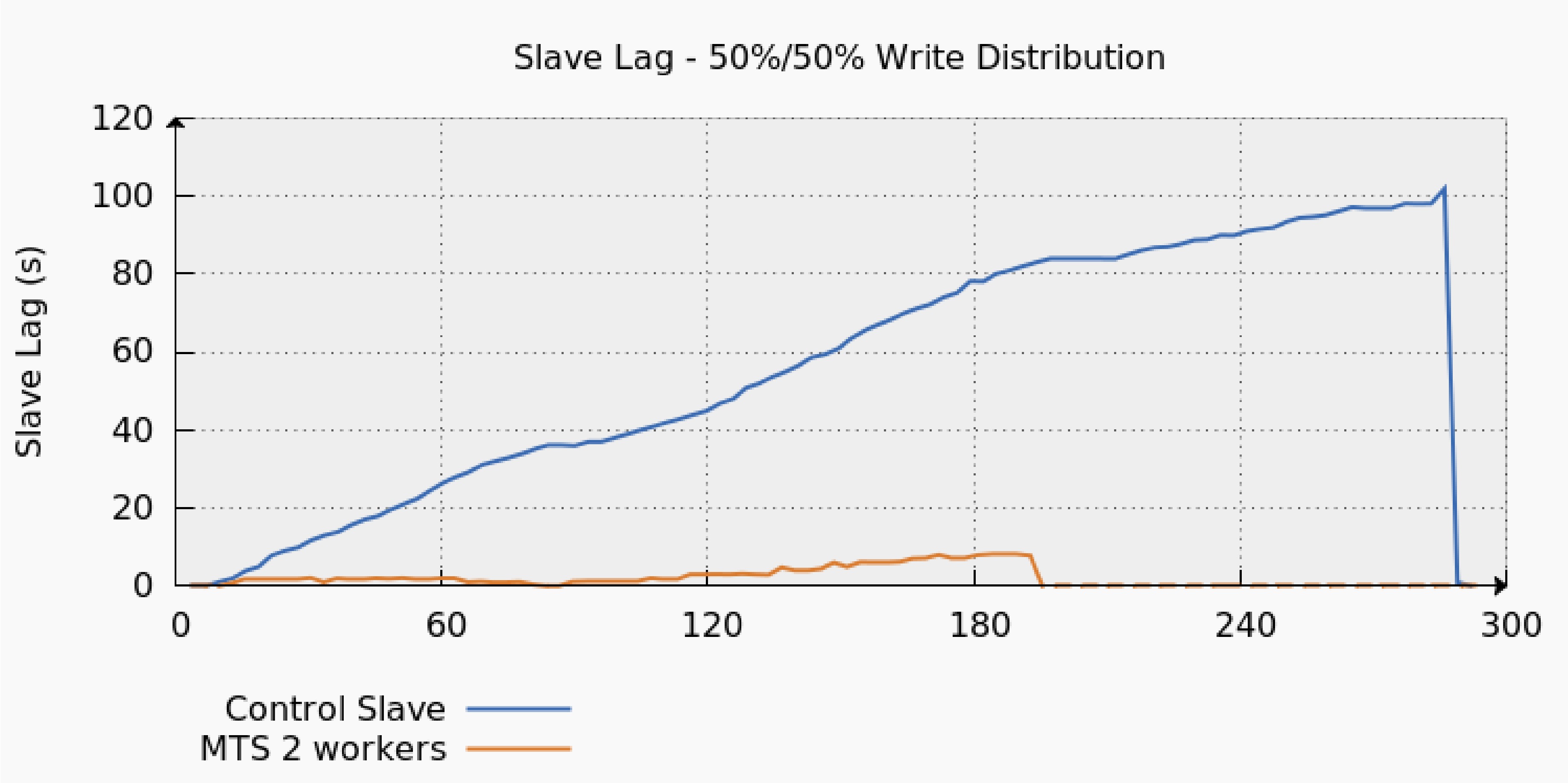
Very high replication lag as expected

Enter multi-threaded replication



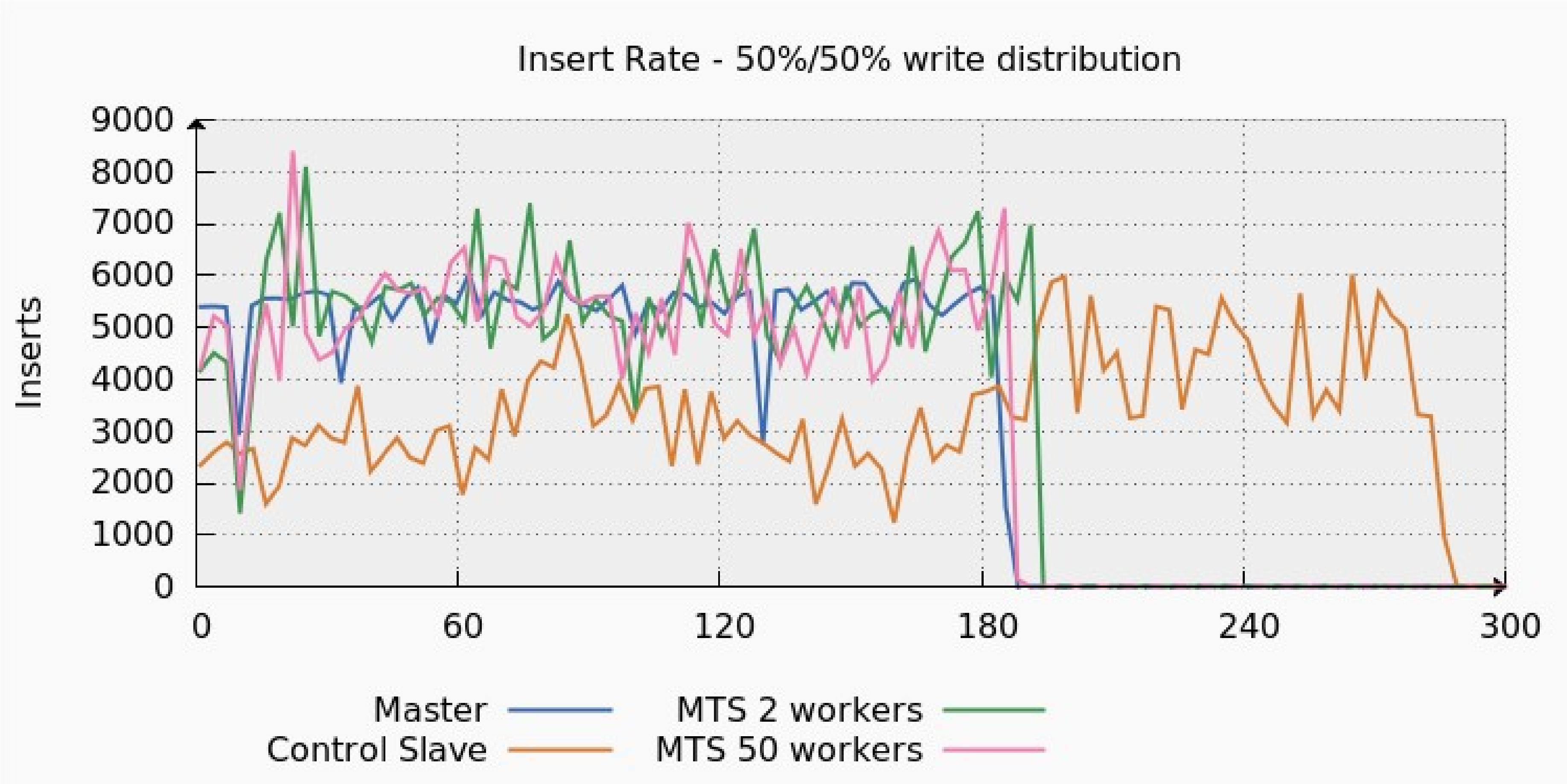
The multi-threaded slave is almost as efficient as the master!

Replication Lag



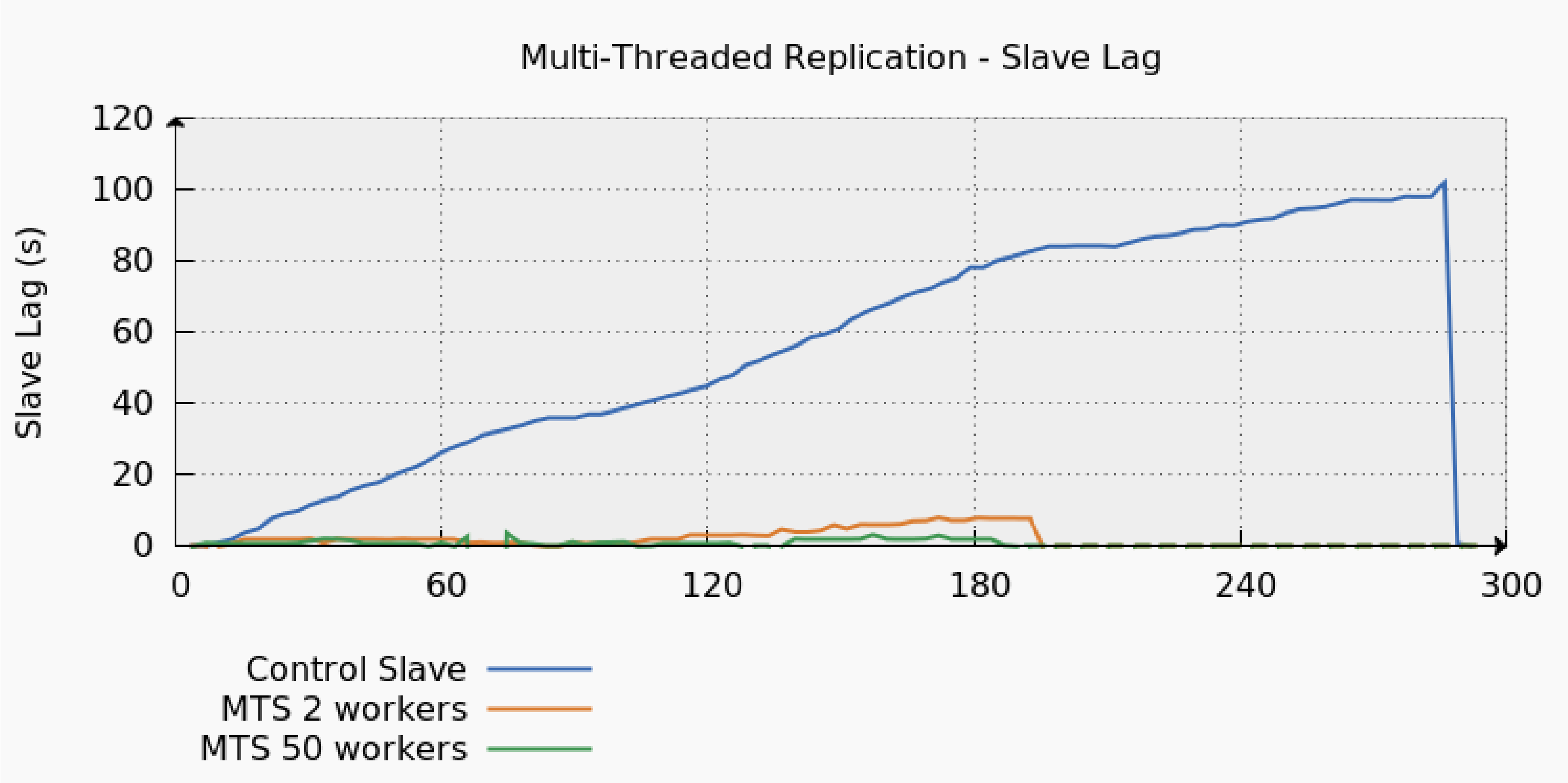
Almost no replication lag for MTS

What about 50 workers?



No visible performance degradation with 50 workers

Replication Lag



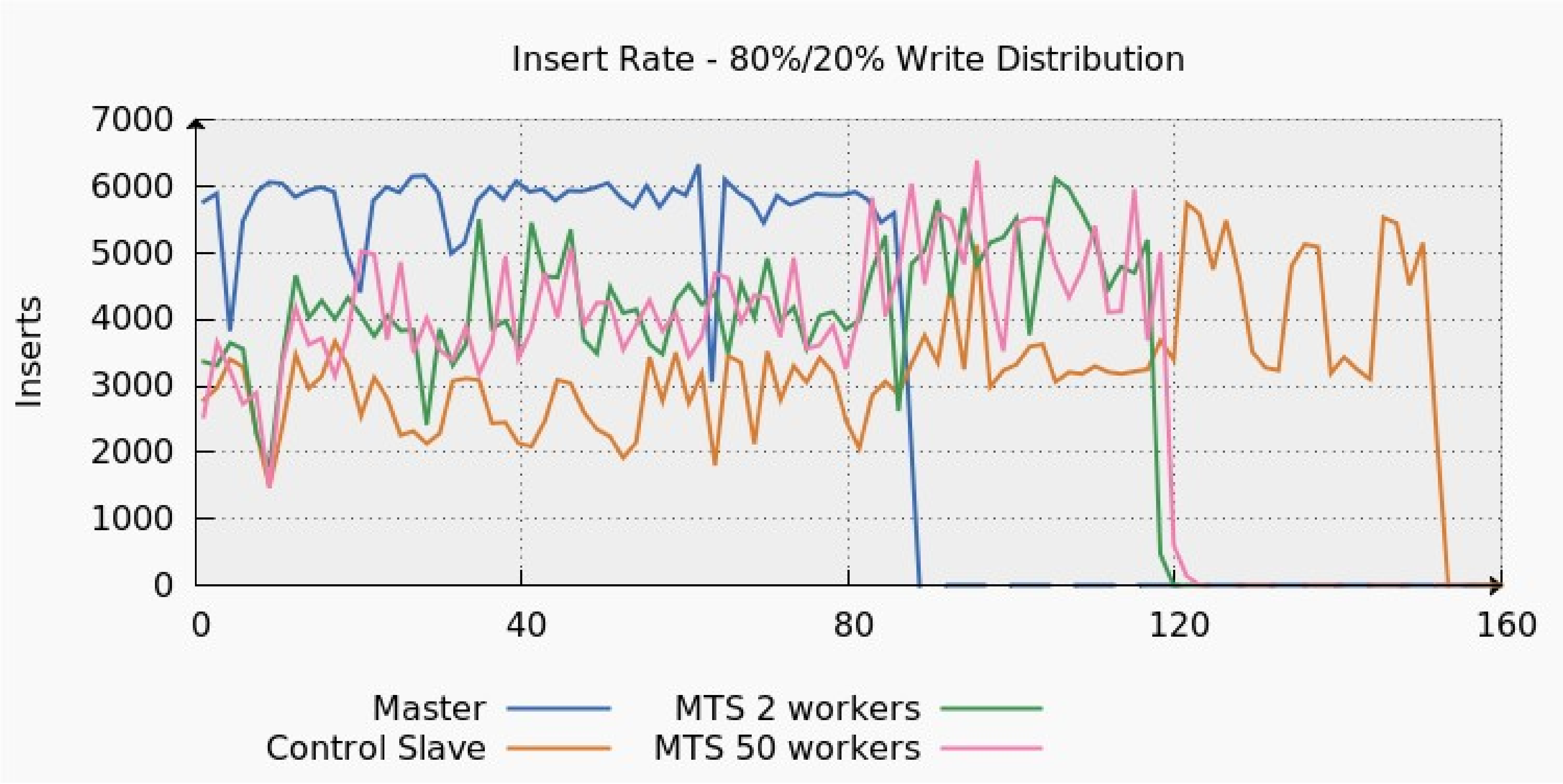
MTS 50 workers is even slightly better

Benchmark Scenario #2

- Same servers
- Same configuration
- Write distribution is not the same
  - db1 gets 80% of the writes

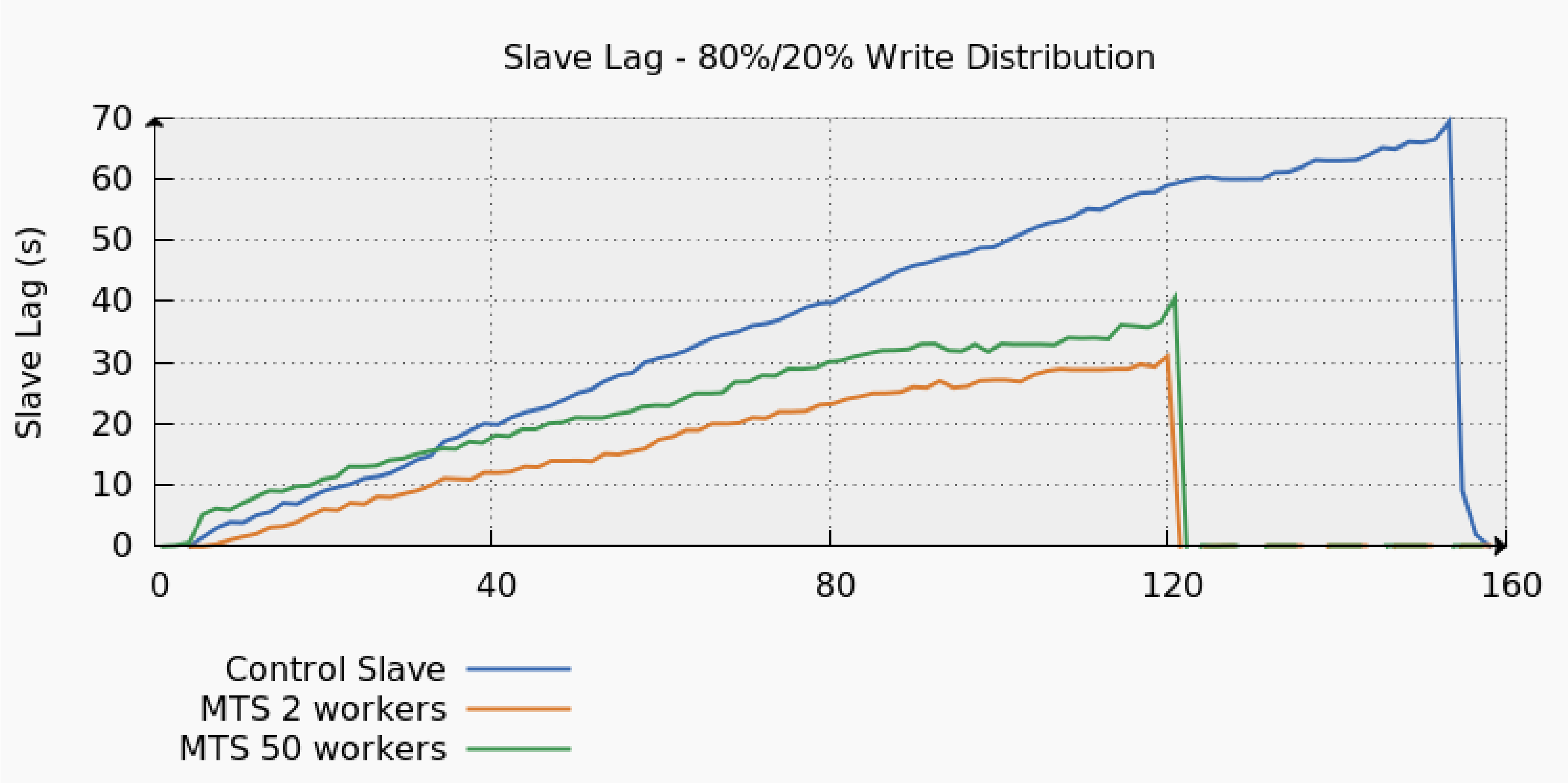
```
# sysbench --mysql-user=root --mysql-db=db1 --test=insert.lua --max-requests=400000 --num-threads=24 --oltp-tables-count=16 run

# sysbench --mysql-user=root --mysql-db=db2 --test=insert.lua --max-requests=100000 --num-threads=6 --oltp-tables-count=16 run
```

Insert Rate



This time MTS is not so efficient...

Replication Lag



… and replication lag shows up again!

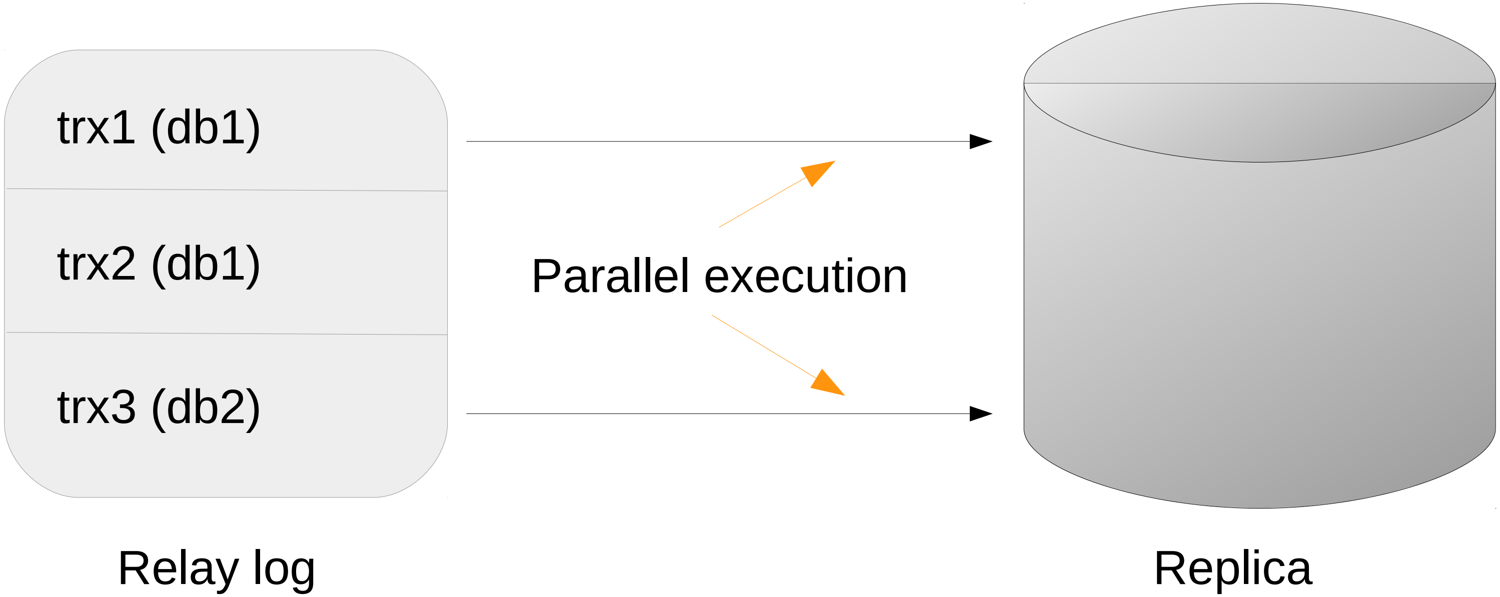
Agenda

- Why multi-threaded replication?
- Performance benefits
- Positioning: GTID or not?
- GTID in a nutshell
- MySQL 5.7

Execution Gaps & Checkpointing

- Worker threads can commit events in parallel
  - Means the events are no longer guaranteed to be consecutive (execution gaps)
- Execution gaps are tracked
- Checkpoints are performed from time to time
  - See `slave_checkpoint_period` and `slave_checkpoint_group` settings

More on execution gaps



- trx3 is executed *before* trx2
- Checkpoints will make sure trx2 is not forgotten

SHOW SLAVE STATUS w/o GTID

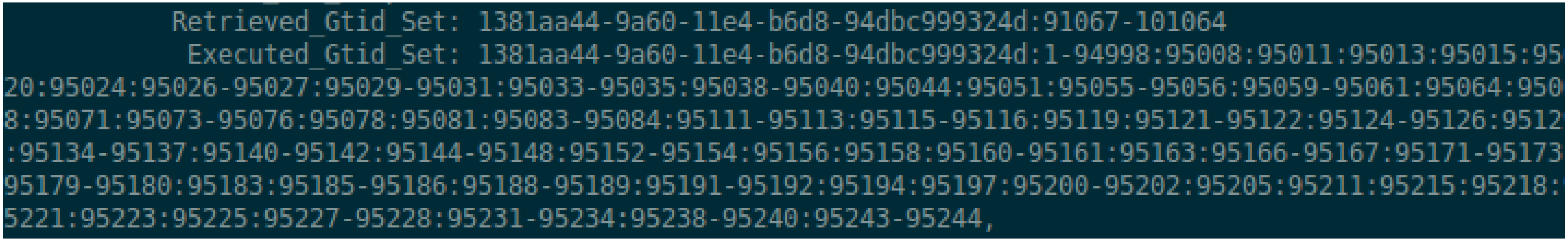
- `Exec_Master_Log_Pos` can no longer be trusted
  - Only shows the position at the latest checkpoint
- Is there a way to remove all execution gaps?
  - Yes: `STOP SLAVE` followed by `START SLAVE UNTIL SQL_AFTER_MTS_GAPS`
  - `STOP SLAVE` alone is not enough (see bug #74528)

Use GTID!

- As `Exec_Master_Log_Pos` is no longer reliable
  - `sql_slave_skip_counter` may not work
  - Be careful with the binlog position when taking a backup from a MTS

- Best option is to use GTIDs
  - `Executed_Gtid_Set` is reliable

SHOW SLAVE STATUS with GTID



```
           Retrieved_Gtid_Set: 1381aa44-9a60-11e4-b6d8-94dbc999324d:91067-101064
            Executed_Gtid_Set: 1381aa44-9a60-11e4-b6d8-94dbc999324d:1-94998:95008:95011:95013:95015:95
20:95024:95026-95027:95029-95031:95033-95035:95038-95040:95044:95051:95055-95056:95059-95061:95064:950
8:95071:95073-95076:95078:95081:95083-95084:95111-95113:95115-95116:95119:95121-95122:95124-95126:9512
:95134-95137:95140-95142:95144-95148:95152-95154:95156:95158:95160-95161:95163:95166-95167:95171-95173
95179-95180:95183:95185-95186:95188-95189:95191-95192:95194:95197:95200-95202:95205:95211:95215:95218:
5221:95223:95225:95227-95228:95231-95234:95238-95240:95243-95244,
```

- Looks ugly?
  - Blame the execution gaps!
  - At least it reflects the reality

Agenda

- Why multi-threaded replication?
- Performance benefits
- Positioning: GTID or not?
- GTID in a nutshell
- MySQL 5.7

GTID?

- Unique identifier of a transaction across all servers of a replication setup
- 2 parts
  - `source_id:transaction_id`
  - `3E11FA47-71CA-11E1-9E33-C80AA9429562:1`
- MySQL 5.6+

Main benefits

- Replication topology is easy to change
  - The options `master_log_file='mysql-bin.xxx', master_log_pos=yyy` are gone, just use `master_auto_position=1`!
- Failover is simplified
- Managing multi-tiered replication is easier

What it is NOT

- A high-availability solution
  - GTIDs do not provide replication monitoring
  - GTIDs do not provide failover
  - But they make HA so much easier

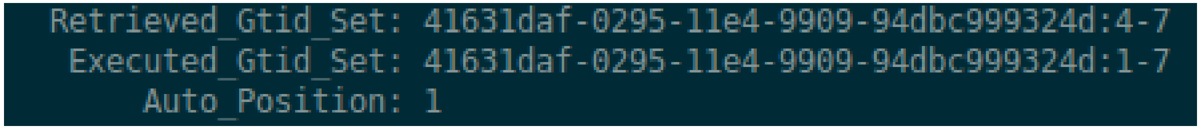
Caveats

- All servers must be restarted at the same time
  - Online GTID rollout in MySQL 5.7
  - Online GTID rollout in Percona Server 5.6.22-72.0+ (porting of the Facebook patch)
  - Booking.com has also developed another patch for online rollout

- `log_bin` + `log_slave_updates` adds some I/O overhead on slaves
  - In 5.7, binary logging is no longer needed for slaves

Checking replication status

- New columns for `SHOW SLAVE STATUS`



```
 Retrieved_Gtid_Set: 41631daf-0295-11e4-9909-94dbc999324d:4-7
  Executed_Gtid_Set: 41631daf-0295-11e4-9909-94dbc999324d:1-7
      Auto_Position: 1
```

- `Retrieved_Gtid_Set`: List of GTIDs received by the I/O thread, cleared after a server restart
- `Executed_Gtid_Set`: List of GTIDs executed by the SQL thread
- `Auto_position`: 1 if GTID-based replication is enabled

Replication protocol

- When slave connects to the master
  - Position-based replication
    - Master sends all transactions from the given offset
  - GTID-based replication
    - Slave sends the range of GTIDs it has executed
    - Master sends back all other transactions
    - Rule: a trx with a given GTID can only execute once
    - Good: allows auto-positioning
    - Bad: creates new challenges

Challenge #1: Skip a transaction

- `sql_skip_slave_counter = N` no longer works
  - Because of the new replication protocol, the transaction would automatically come back
  - Solution is to execute a fake trx with the GTID you want to skip

```
mysql> STOP SLAVE;
mysql> SET gtid_next = 'XXXX:NN';
mysql> BEGIN;COMMIT;      # Fake transaction!
mysql> SET gtid_next=automatic;
mysql> START SLAVE;
```

Challenge #2: Errant transactions

- A local trx on a slave generates its own GTID
- If slave is promoted, trx is sent to all servers
  - Again thanks to the new replication protocol

- That can bite on failover
  - Trx is not desired: well, now it is everywhere
  - Trx is no longer in the binlogs: the IO thread will exit with a 1236 error, replication is now broken

Detect/fix errant transactions

- Use `GTID_SUBSET() and GTID_SUBTRACT()` to identify an errant transaction

- Skip it on all other servers with an empty trx
  - Or inject the empty trx on the master if it is online

- If you need to run local transactions on slaves, prefer `SET sql_log_bin = 0`

Agenda

- Why multi-threaded replication?
- Performance benefits
- Positioning: GTID or not?
- GTID in a nutshell
- MySQL 5.7

More parallelization in 5.7

- Worker threads can parallel apply transactions on the same database
  - Use `slave_parallel_type = logical_clock`
  - Master must be running 5.7 for logical clock to work
- 5.6-style MTS is still available (`slave_parallel_type = database`)

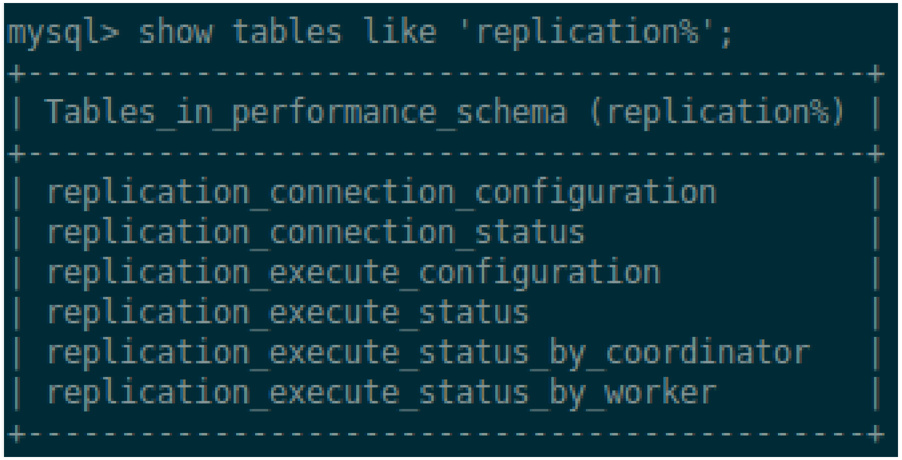
Logical clock

- On the master
  - Additional metadata is stored in the binlogs to identify transactions that can be applied in parallel
  - Takes advantage of binlog group commit
- On the slave
  - The coordinator thread is able to extract the metadata from the relay logs to dispatch the transactions across workers

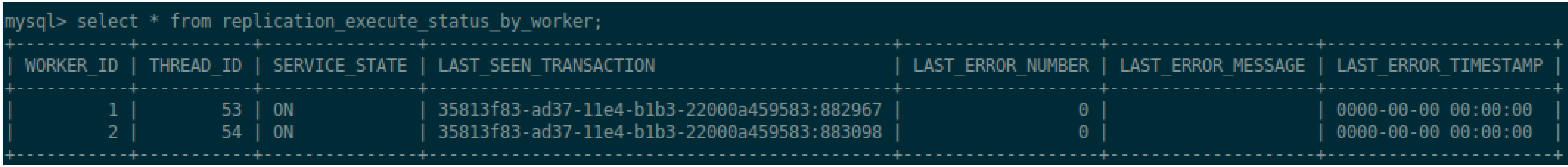
Better replication monitoring (5.7)

- `SHOW SLAVE STATUS` is okay for single-threaded replication, not so much for MTS
  - `Last_Error`: what if several threads have errors?
  - ...
- 5.7 is using performance_schema
  - Requires more complex SQL to get diagnostics
  - But flexible and extensible

performance_schema tables



```
mysql> show tables like 'replication%';
+---------------------------------------------+
| Tables_in_performance_schema (replication%) |
+---------------------------------------------+
| replication_connection_configuration        |
| replication_connection_status               |
| replication_execute_configuration           |
| replication_execute_status                  |
| replication_execute_status_by_coordinator   |
| replication_execute_status_by_worker        |
+---------------------------------------------+
```



```
mysql> select * from replication_execute_status_by_worker;
+-----------+-----------+---------------+---------------------------------------------+-------------------+--------------------+----------------------+
| WORKER_ID | THREAD_ID | SERVICE_STATE | LAST_SEEN_TRANSACTION                       | LAST_ERROR_NUMBER | LAST_ERROR_MESSAGE | LAST_ERROR_TIMESTAMP |
+-----------+-----------+---------------+---------------------------------------------+-------------------+--------------------+----------------------+
|         1 |        53 | ON            | 35813f83-ad37-11e4-b1b3-22000a459583:882967 |                 0 |                    | 0000-00-00 00:00:00  |
|         2 |        54 | ON            | 35813f83-ad37-11e4-b1b3-22000a459583:883098 |                 0 |                    | 0000-00-00 00:00:00  |
+-----------+-----------+---------------+---------------------------------------------+-------------------+--------------------+----------------------+
```

Q&A

Thanks for attending!

Feel free to drop me a line at:

stephane.combaudon@percona.com