

# Consistent Reads Using ProxySQL and GTID

---

Santa Clara, California | April 23th – 25th, 2018



PERCONA  
LIVE

# Disclaimer

- I am
  - not René Cannaò
  - @lefred – MySQL Community Manager / Oracle
  - the one who provided a hint for this
  - not the one writing these slides (credit to René and Nik Vyzas)

sysown / proxysql

Code Issues 372 Pull requests 22 Projects 0 Wiki Insights

Watch 187 Star 1,844 Fork 303

Feature Request: Consistent reads #1208

lefred opened this issue on Oct 16, 2017 · 2 comments

**Open** lefred commented on Oct 16, 2017

It would be awesome that ProxySQL could be used to perform consistent reads (casual reads) on demand when using replication architecture (asynchronous replica or group replication). The MySQL protocol at every commits returns the GTID of the transaction in the OK packet. It would be nice to keep track of this GTID (per connection) and allow consistent reads when using another host to perform a read then the one where the write happened. ProxySQL could be used to rewrite the query adding WAIT\_FOR\_EXECUTED\_GTID\_SET or just adding a query with this and ignore it. This could be done globally (so wait for the highest GTID set in the history kept in proxySQL) or per session where we only wait for the last write belonging to that session. Enabling it could be done globally or using a hint that will trigger the rule in ProxySQL. (see <http://lefred.be/content/mysql-group-replication-read-your-own-write-across-the-group/>)

lefred changed the title from **Consistent reads** to **Feature Request: Consistent reads** on Nov 7, 2017

Assignees  
No one assigned

Labels  
None yet

Projects  
None yet

Milestone  
No milestone

Notifications

Unsubscribe

You're receiving notifications because you authored the thread.

# What is ProxySQL?

A brief introduction to ProxySQL

# What is ProxySQL?

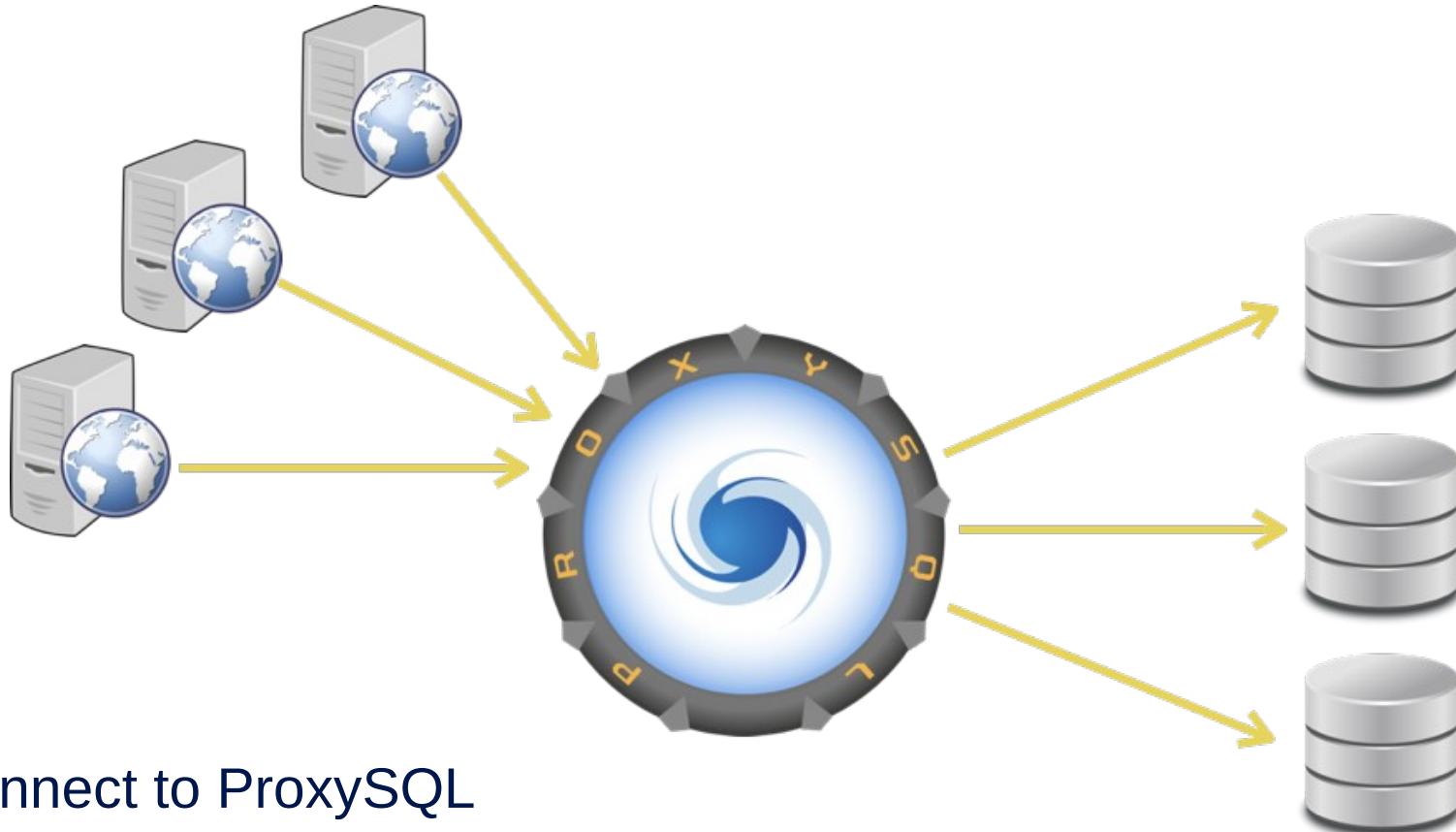
- The winner of a MySQL Community Award 2018 !!



# What is ProxySQL?

- A "Layer 7" database proxy
- MySQL / ClickHouse protocol aware
- High Performance
- High Availability
- Feature Rich

# ProxySQL Overview



- Clients connect to ProxySQL
- Requests are evaluated
- Actions are performed (e.g. RW Split / Sharding / etc.)

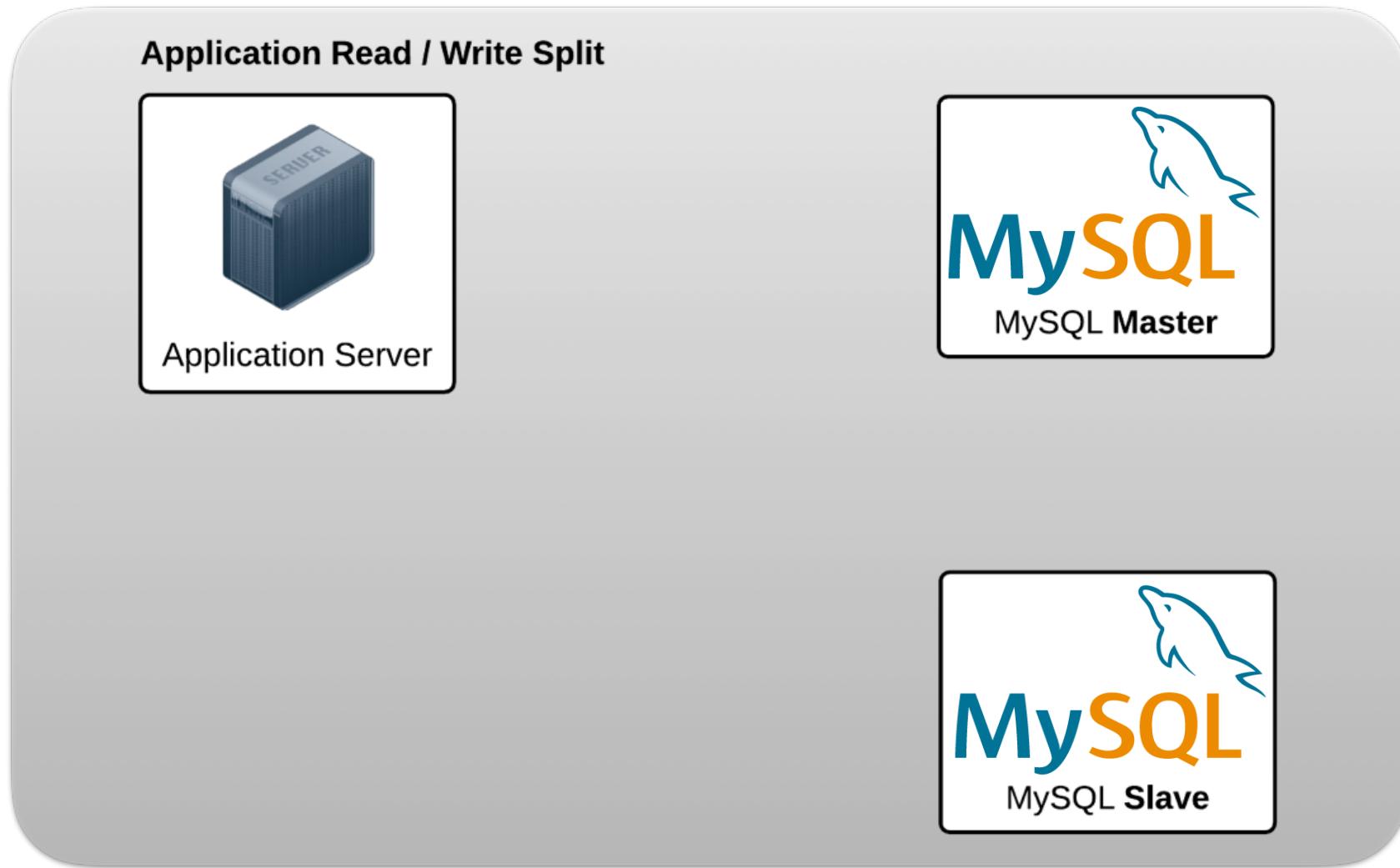
# Master - Slave Replication Pain Points

- Asynchronous replication
  - Replication lag is the **major challenge**
- Semi-synchronous replication
  - Completion time for a transaction depends on availability of slave(s)
  - The time taken to complete the transaction can still cause stale data
  - To avoid stale data applications / client connections must be aware if there is replication delay

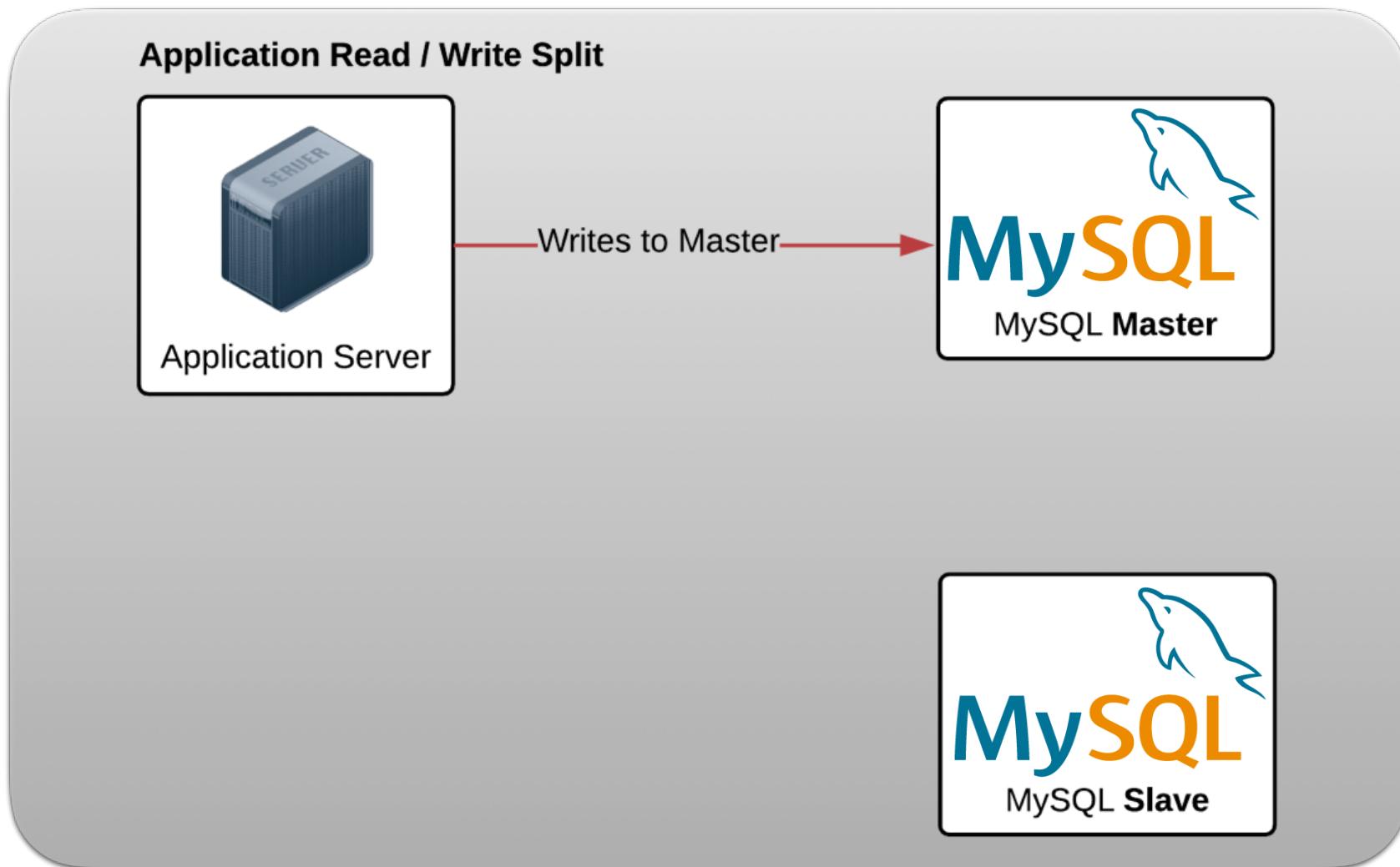
# RW Split and MySQL Replication

A few slides about read / write load balancing across MySQL masters and slaves

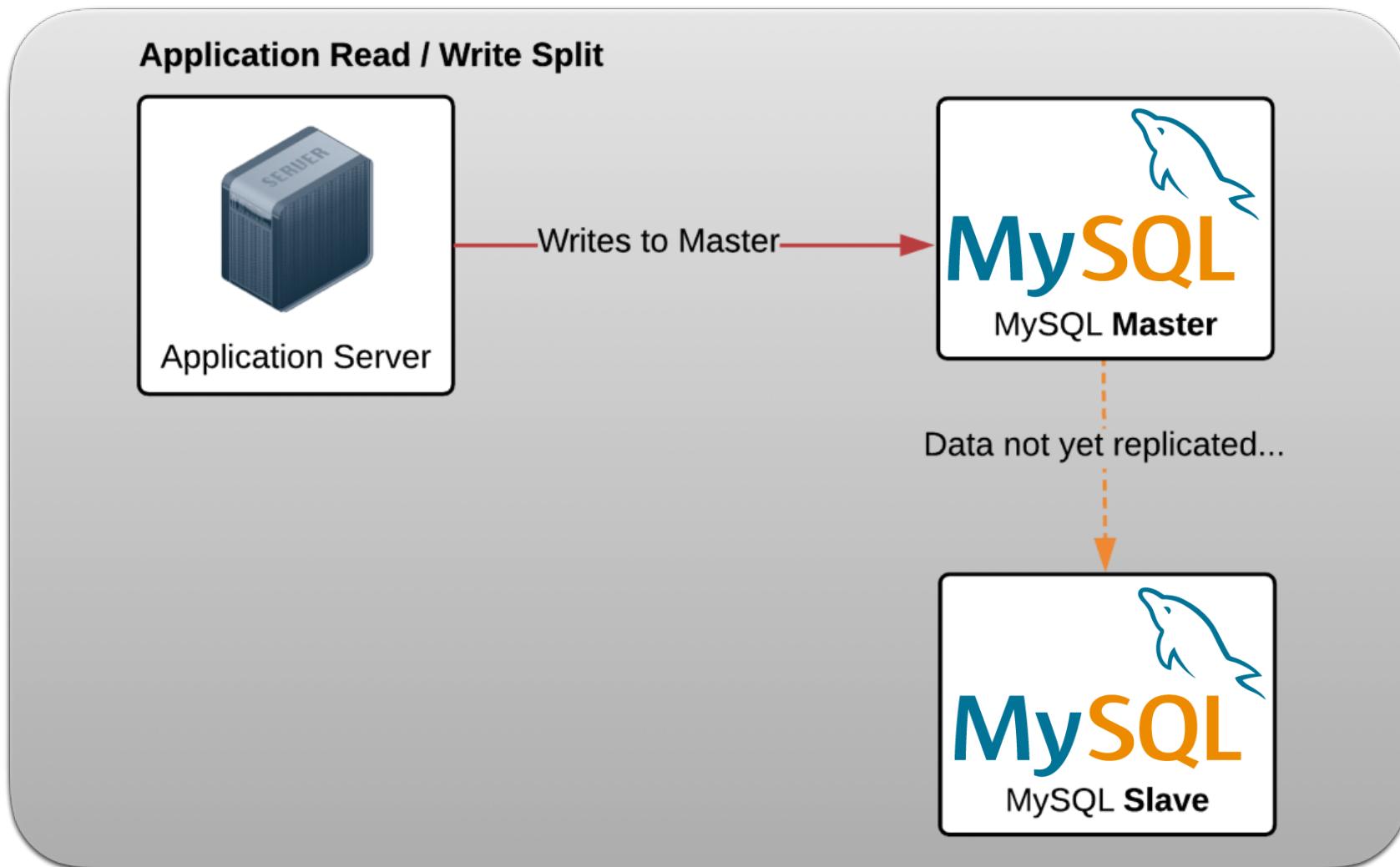
# Application Read / Write Split



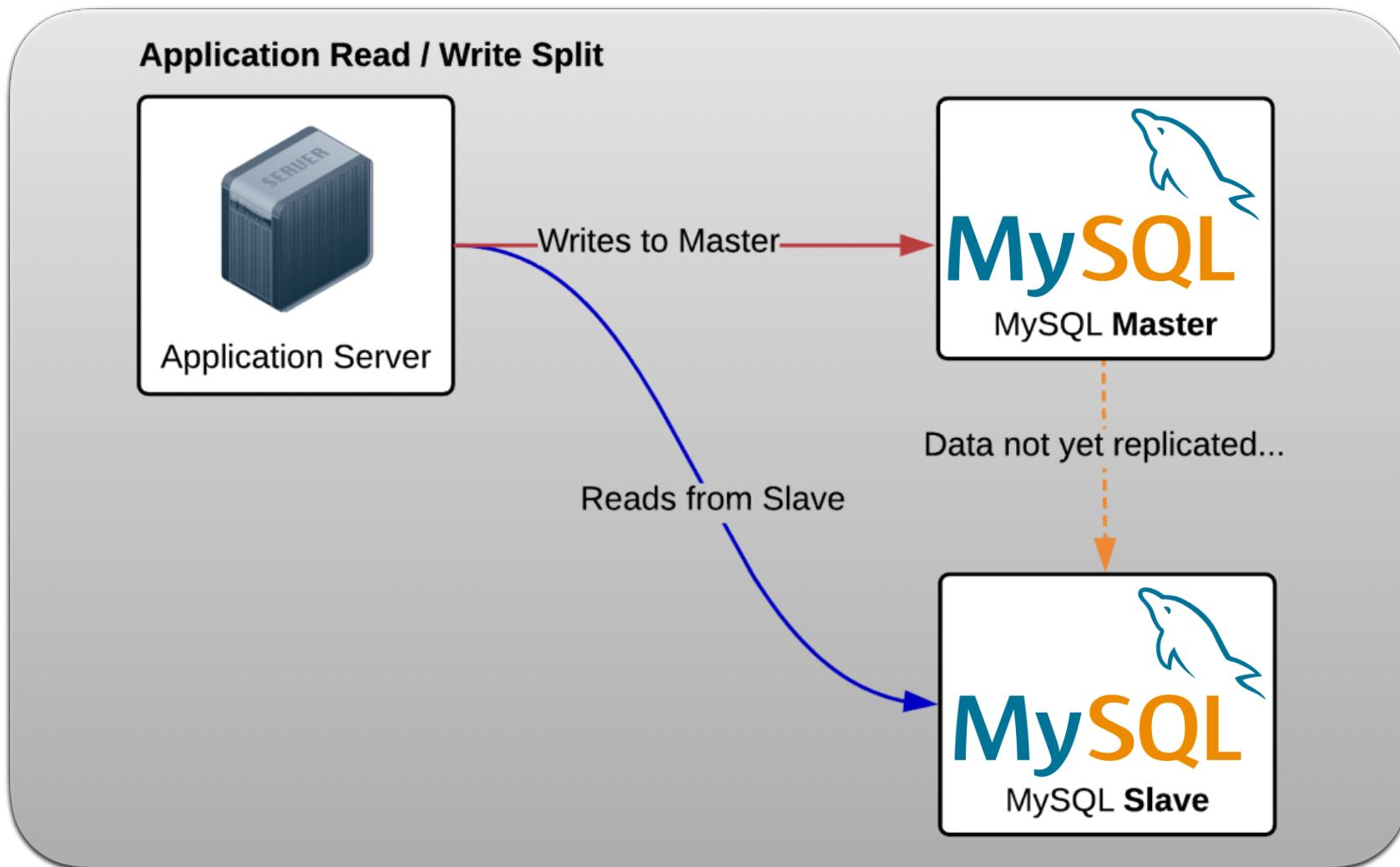
# Application Read / Write Split



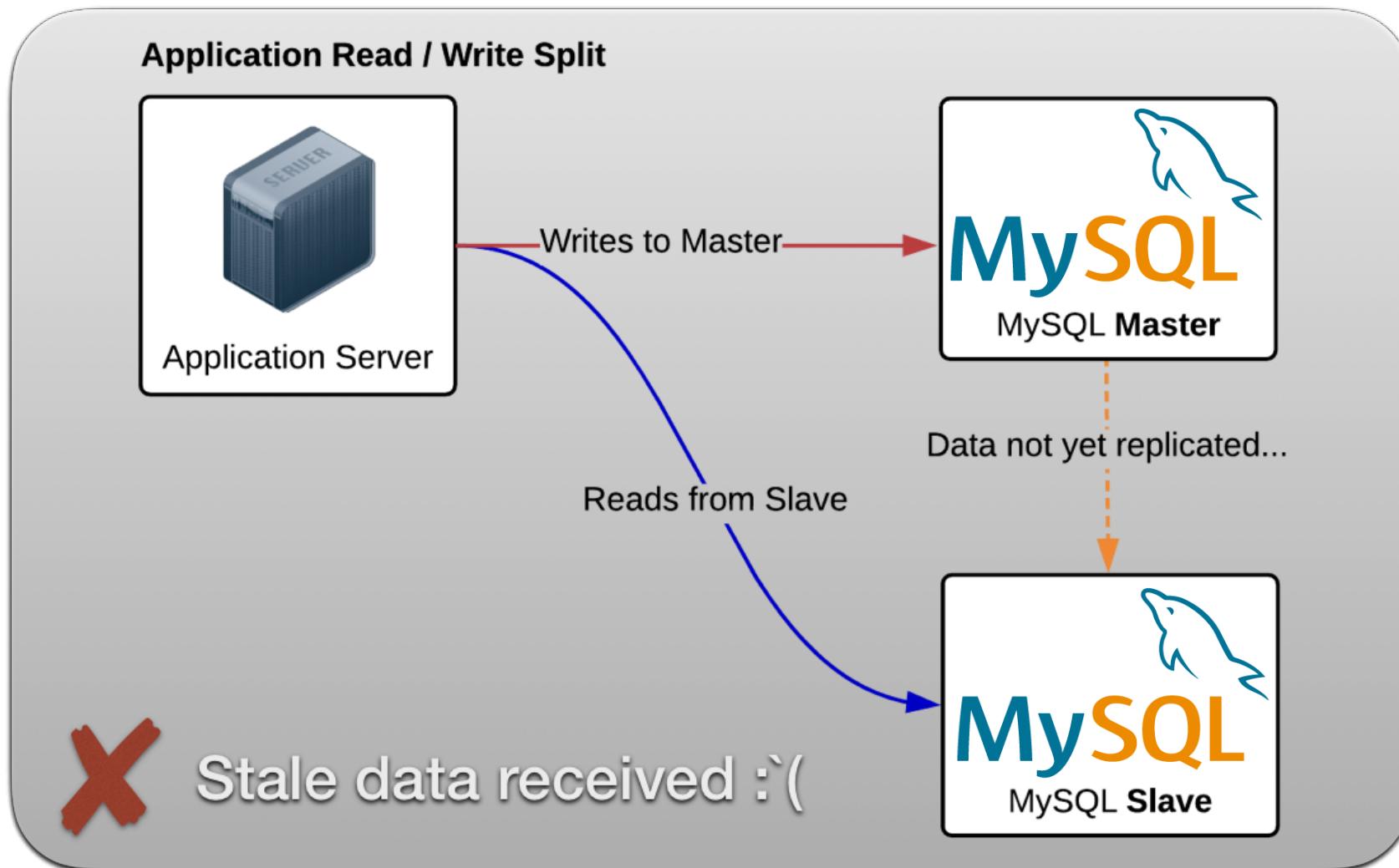
# Application Read / Write Split



# Application Read / Write Split

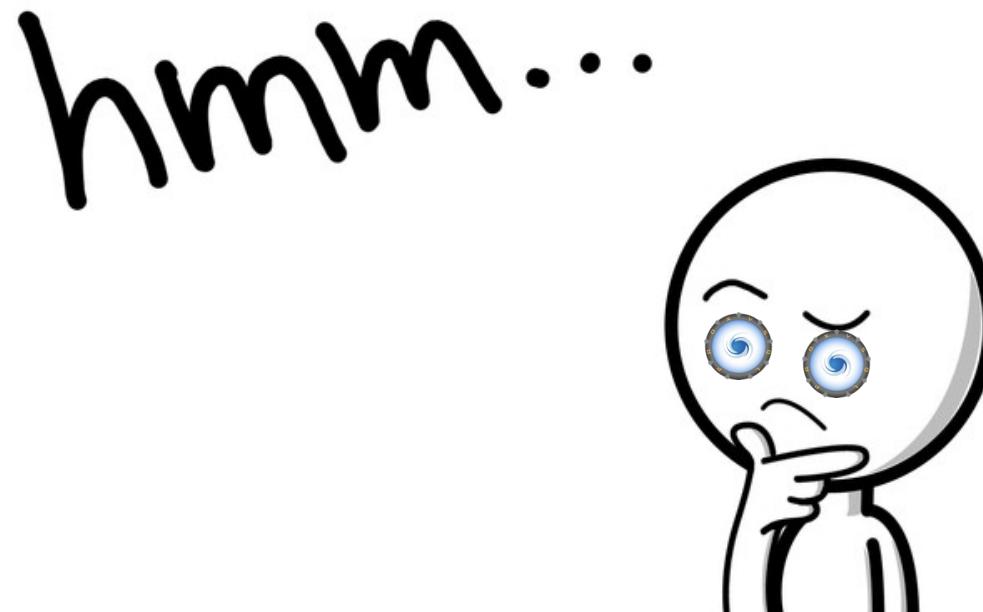


# Application Read / Write Split

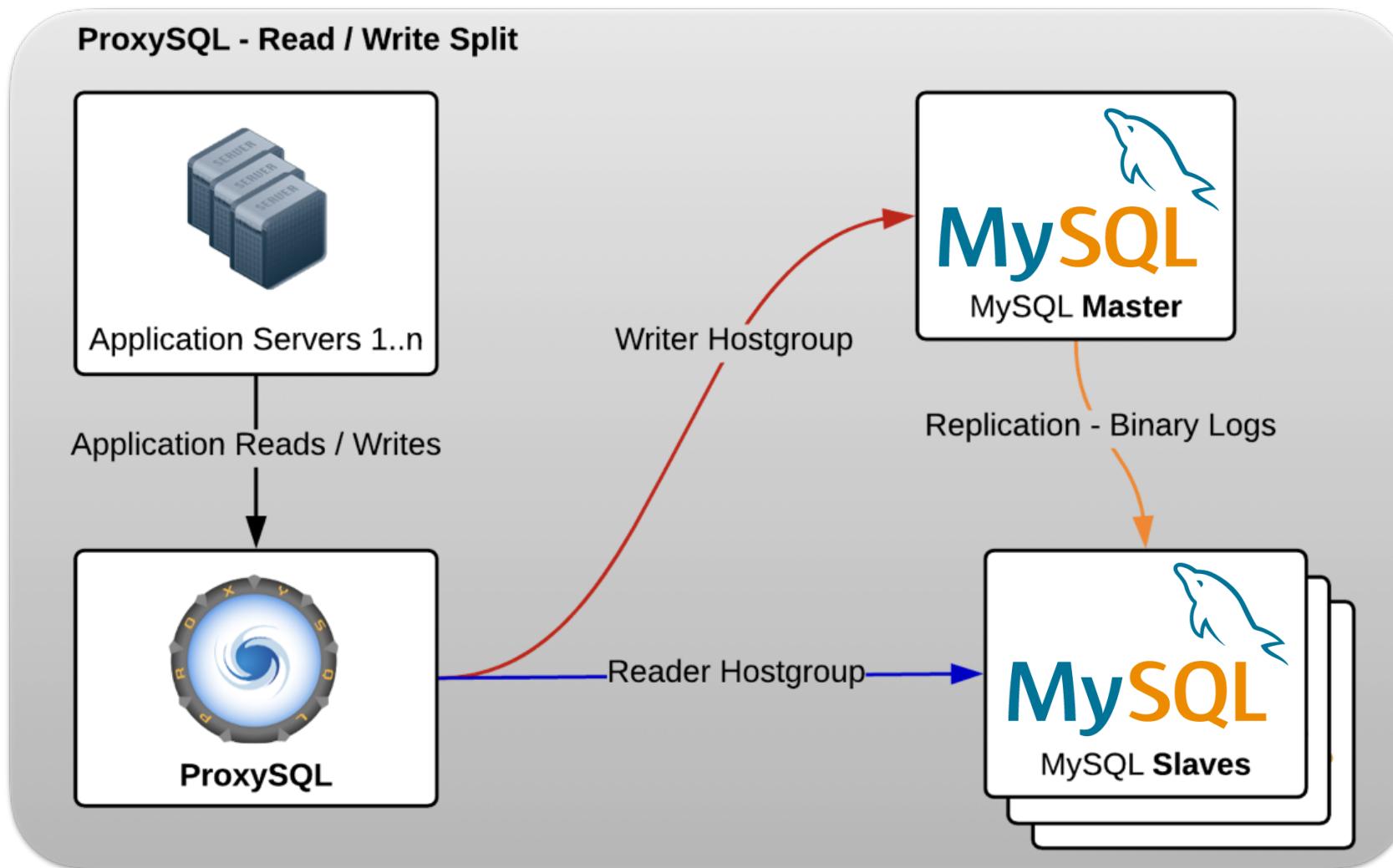


# ProxSQL Read / Write Split

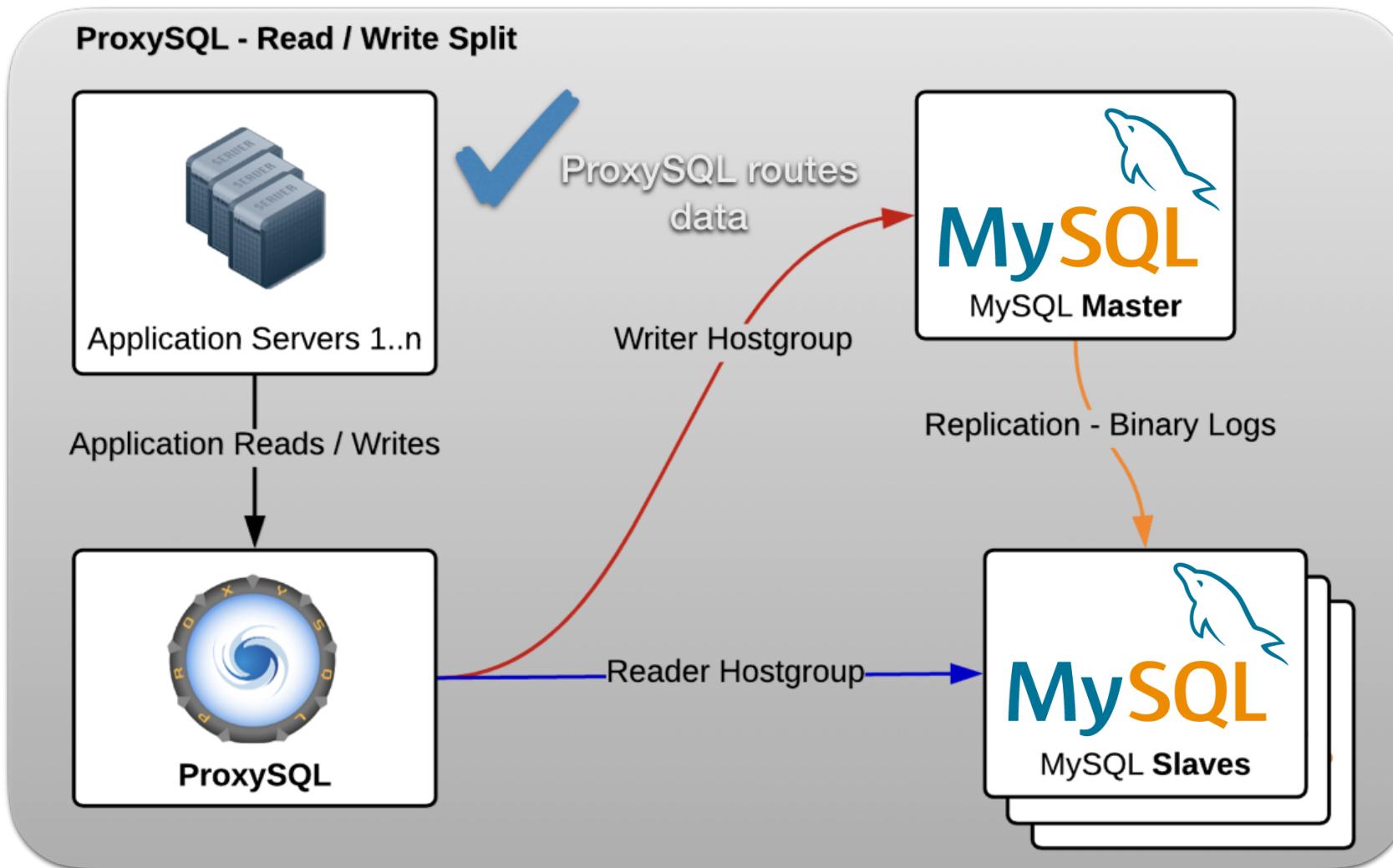
- How are these problems solved with ProxySQL...?



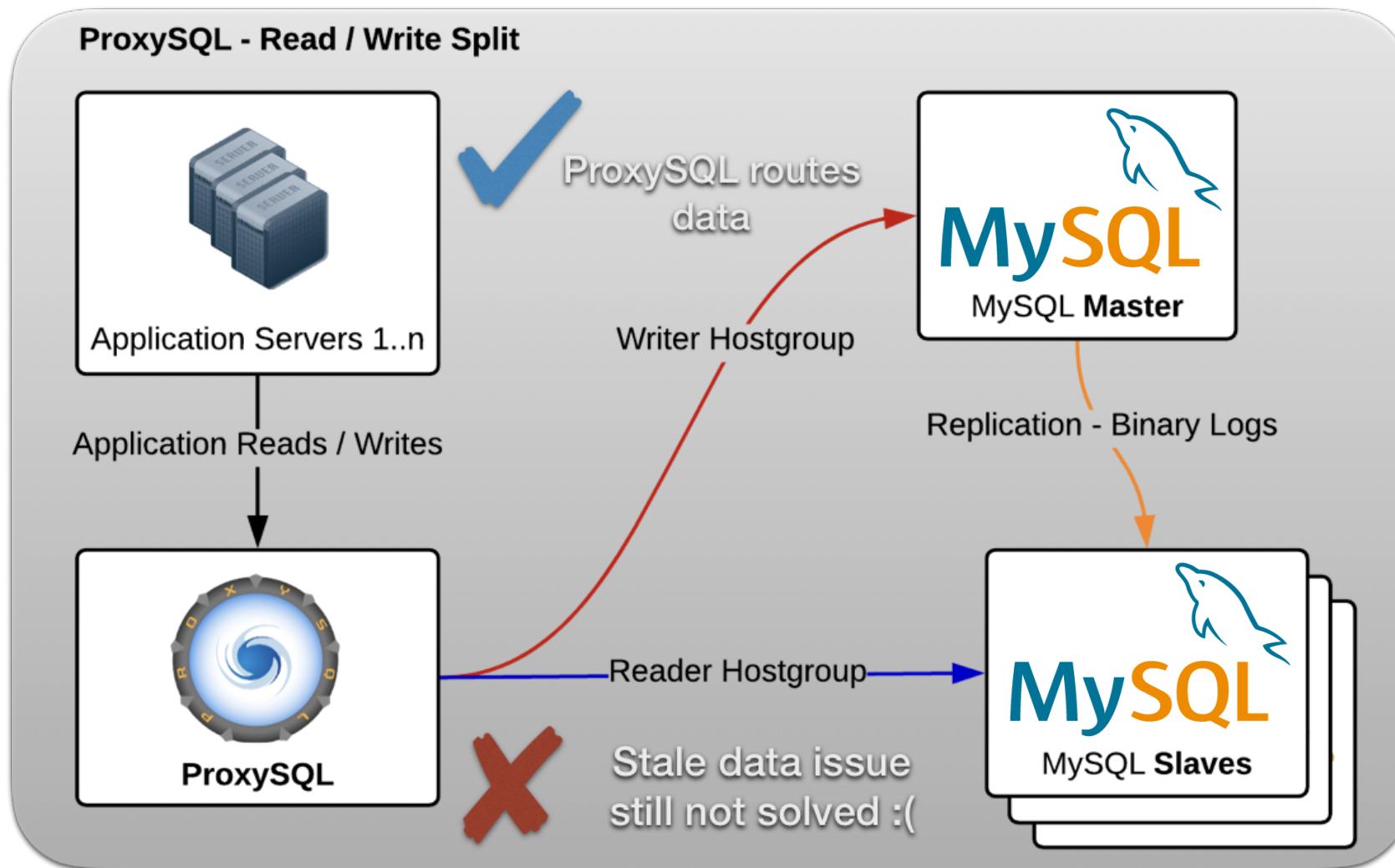
# ProxSQL Read / Write Split



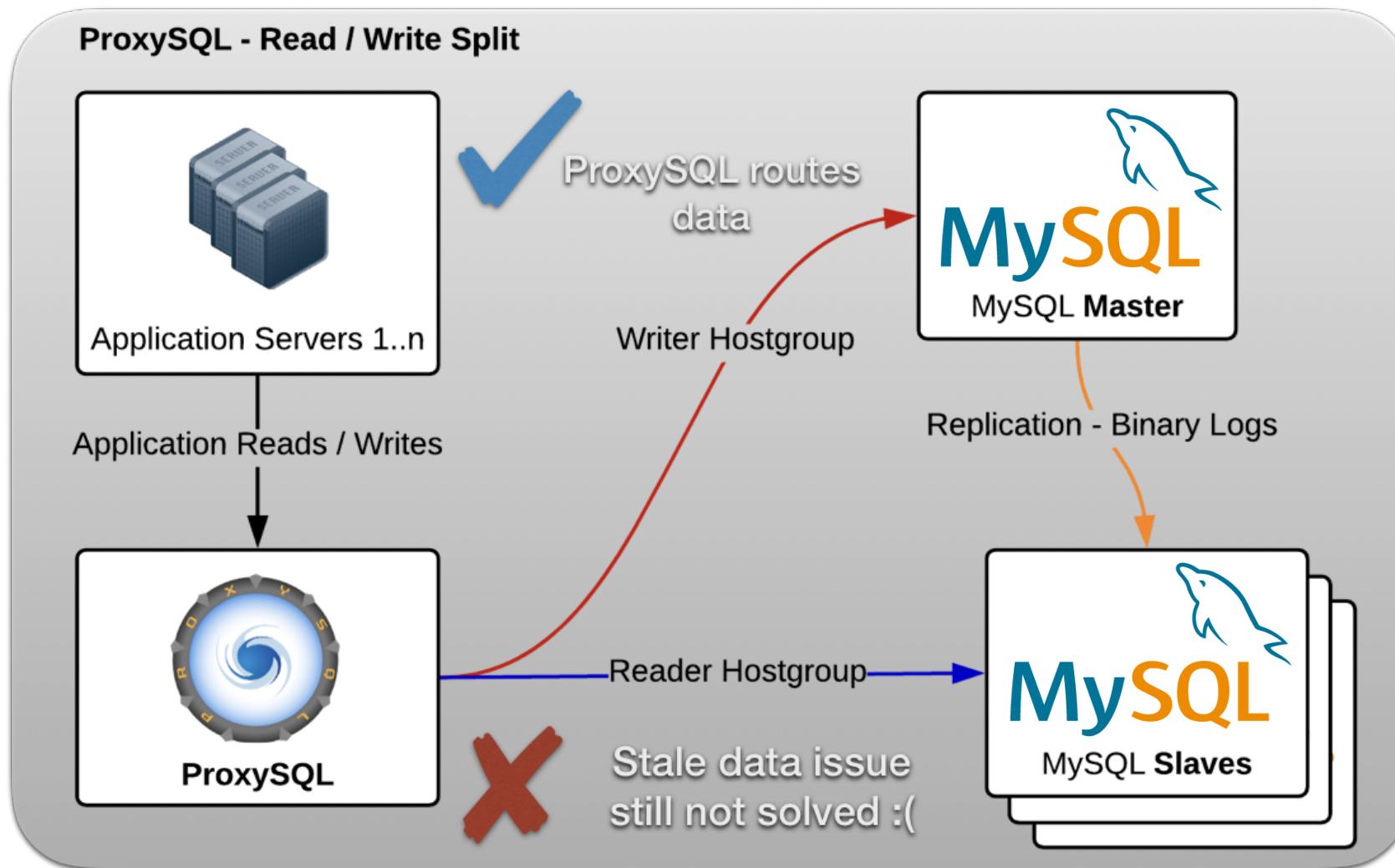
# ProxSQL Read / Write Split



# ProxSQL Read / Write Split



# ProxSQL Read / Write Split



# Benefits of ProxySQL's Read / Write Split

- Query rules defined in ProxySQL can dynamically route queries to READER or WRITER hostgroups
- Seamless for an application connecting and no application changes are required
- All traffic is served from a single listening port
- Slaves can be dynamically added / removed from a hostgroup to scale or perform maintenance

# Challenges of R/W Split

- Susceptible to serve stale data due to replication lag
- Replication lag can be monitored and the reads can be routed to the master if a threshold is breached
- Threshold is configurable in increments of 1 second
- Replication lag is determined by polling at regular intervals

# Replication in MySQL

A few slides about replication in MySQL historically as well as current features

# Traditional binlog replication

- Traditional replication requires master & slave binary log file / position to be 100% synchronised
- Binary log events must be processed sequentially
- Binary log events can be missed or re-executed if replication is started from the wrong binlog file / position
- During failover replication should be stopped at the same position on all slaves to ensure data consistency after promotion

# What is GTID?

- GTID is an acronym for "global transaction identifier"
- Unique identifier for every committed transaction
- GTID is unique across all servers in a master / slave cluster
- 1-to-1 mapping between all transactions and all GTIDs
- Represented as a colon separated pair of coordinates:

**GTID = source\_id:transaction\_id**

# Why is GTID important?

- GTID guarantees consistency by detecting missing transactions from the set of GTIDs executed on a slave
- Supports auto-positioning making failover simpler, safer and quicker as slaves can be repointed to masters at any level of the a replication hierarchy
- `SELECT WAIT_UNTIL_SQL_THREAD_AFTER_GTIDS()` was introduced in 5.6.9 obsoleting `WAIT_FOR_EXECUTED_GTID_SET()` from MySQL 5.6.5.
  - Allows "SELECT" to wait until all GTIDs in a specified set have executed
  - You need to have the GTID prior to executing
  - Better approach however queries may be delayed

# An important enhancement in MySQL 5.7

- In MySQL 5.7 & Percona Server 5.7 an important feature was added which allows sending the GTID for a transaction on the OK packet for a transaction
- Enabled explicitly by setting **--session-track-gtids** to one of the following values:
  - "OWN\_GTID": collect GTIDs generated for committed R/W transactions
  - "ALL\_GTIDS": collect ALL GTIDs in gtid\_executed when a R/W or R/O transaction commits

Note: This feature is NOT available in MariaDB

# Leveraging GTID in ProxySQL 2.0

New features / components introduced in ProxySQL 2.0 to leverage GTID

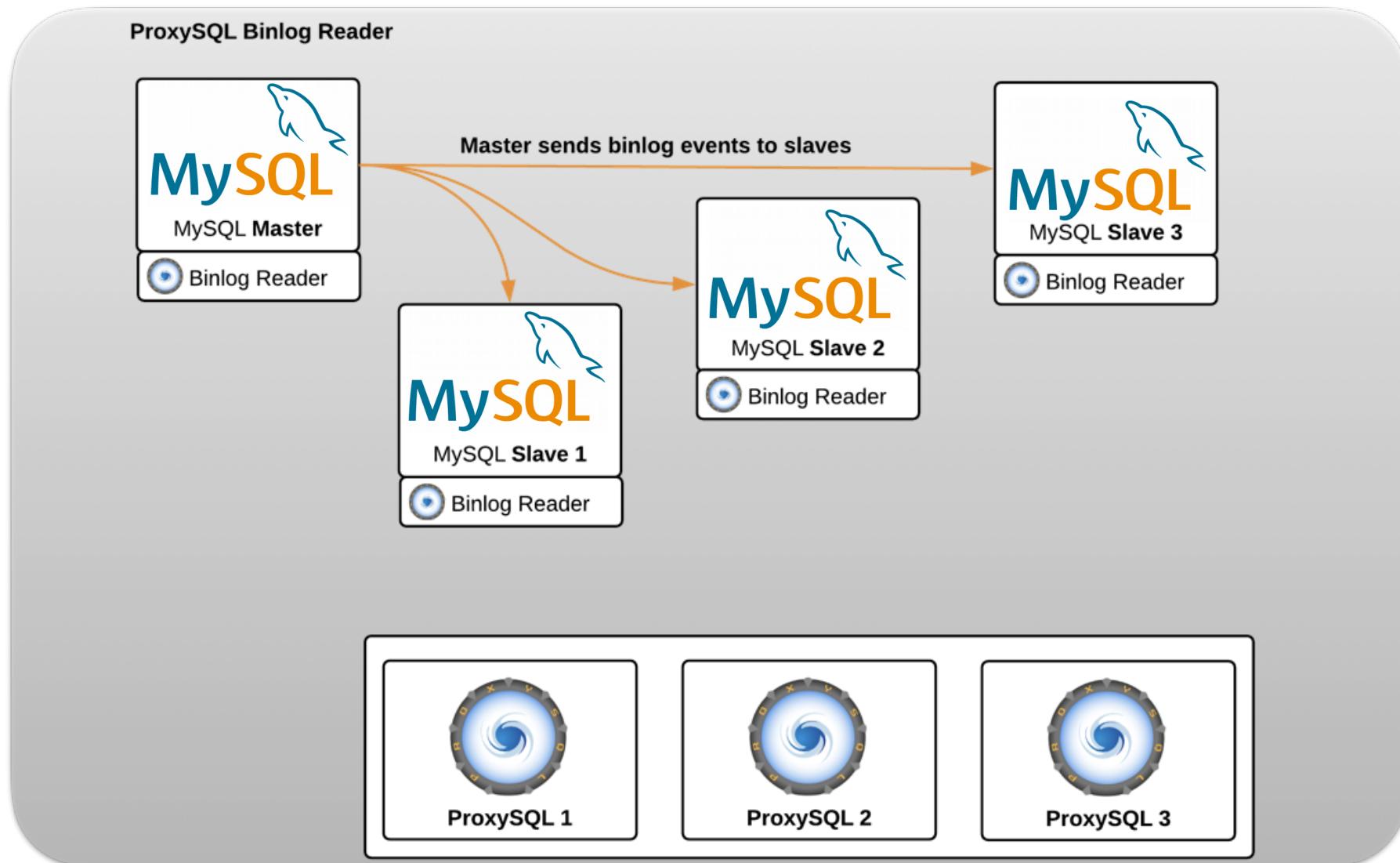
# GTID tracking in ProxySQL

- Since GTIDs can be tracked on client connections... why not track these in ProxySQL as well?
- Tracking the GTIDs executed on a MySQL server can be done in one of two ways:
  - pull method: ProxySQL can query each MySQL server to fetch the last executed GTID
  - push method: Parse the binlog events "**as a slave**" and send the GTIDs processed to ProxySQL
- The "push method" is far more efficient and results in **less requests** and **lower latency**
  - Especially important in large scale deployments

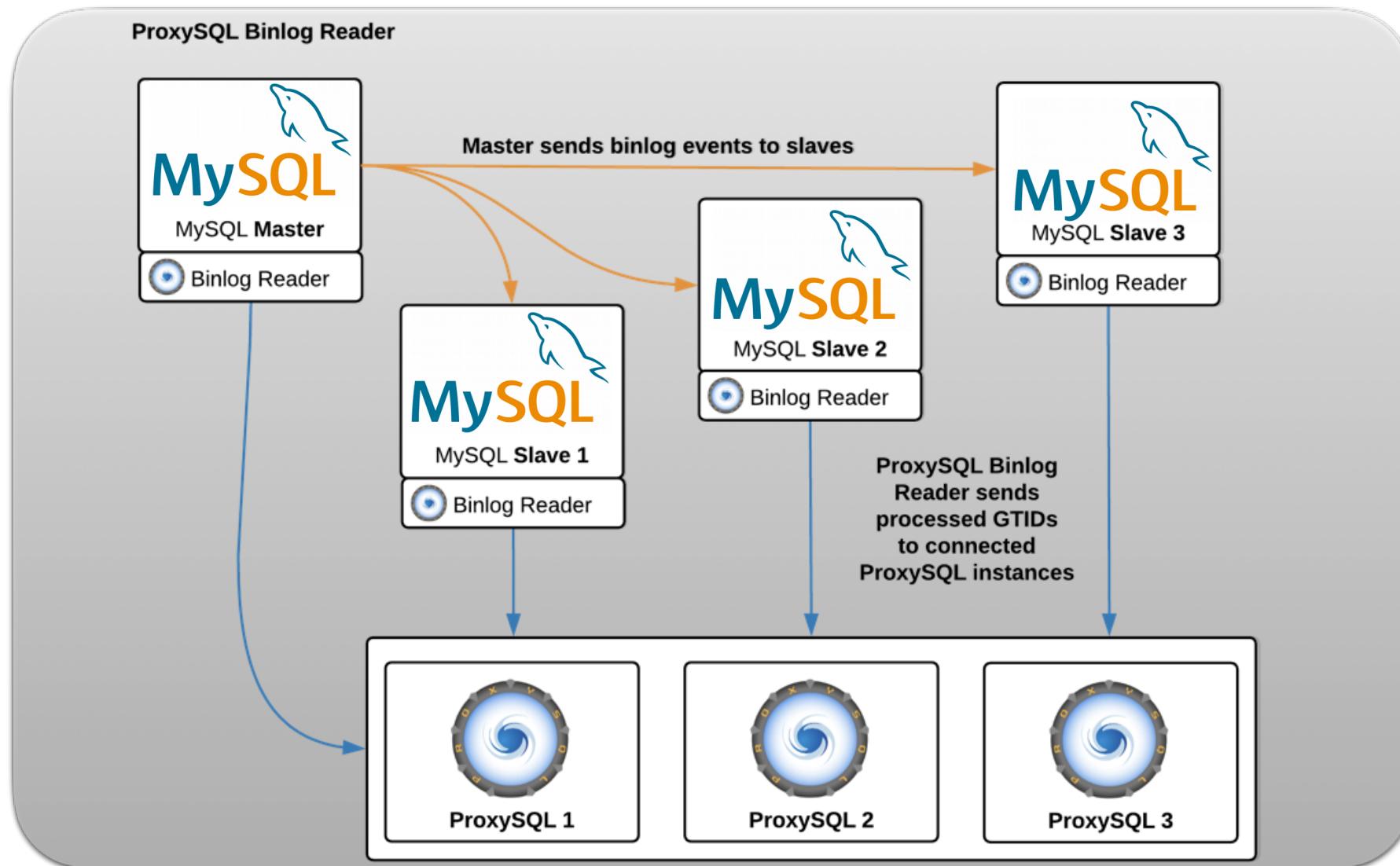
# ProxySQL Binlog Reader

- A lightweight process that runs on the MySQL server
- Primary task is to provide GTID information about a MySQL server to all connected ProxySQL instances
- Designed to be robust and efficient while keeping CPU and network I/O to an absolute minimum for supporting hundreds
- Features an auto-restart mechanism in case of failure and a client side reconnect

# ProxySQL Binlog Reader



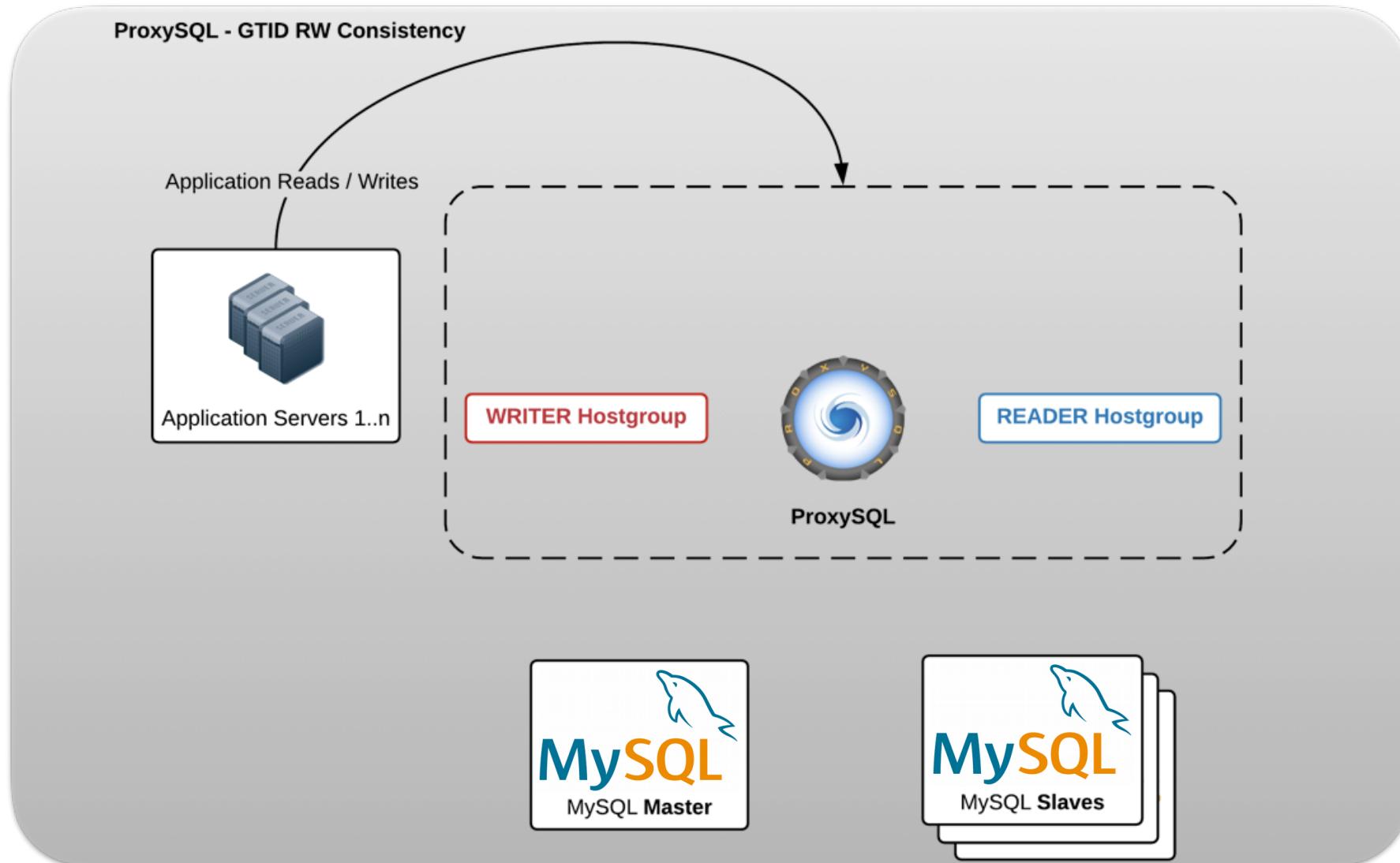
# ProxySQL Binlog Reader



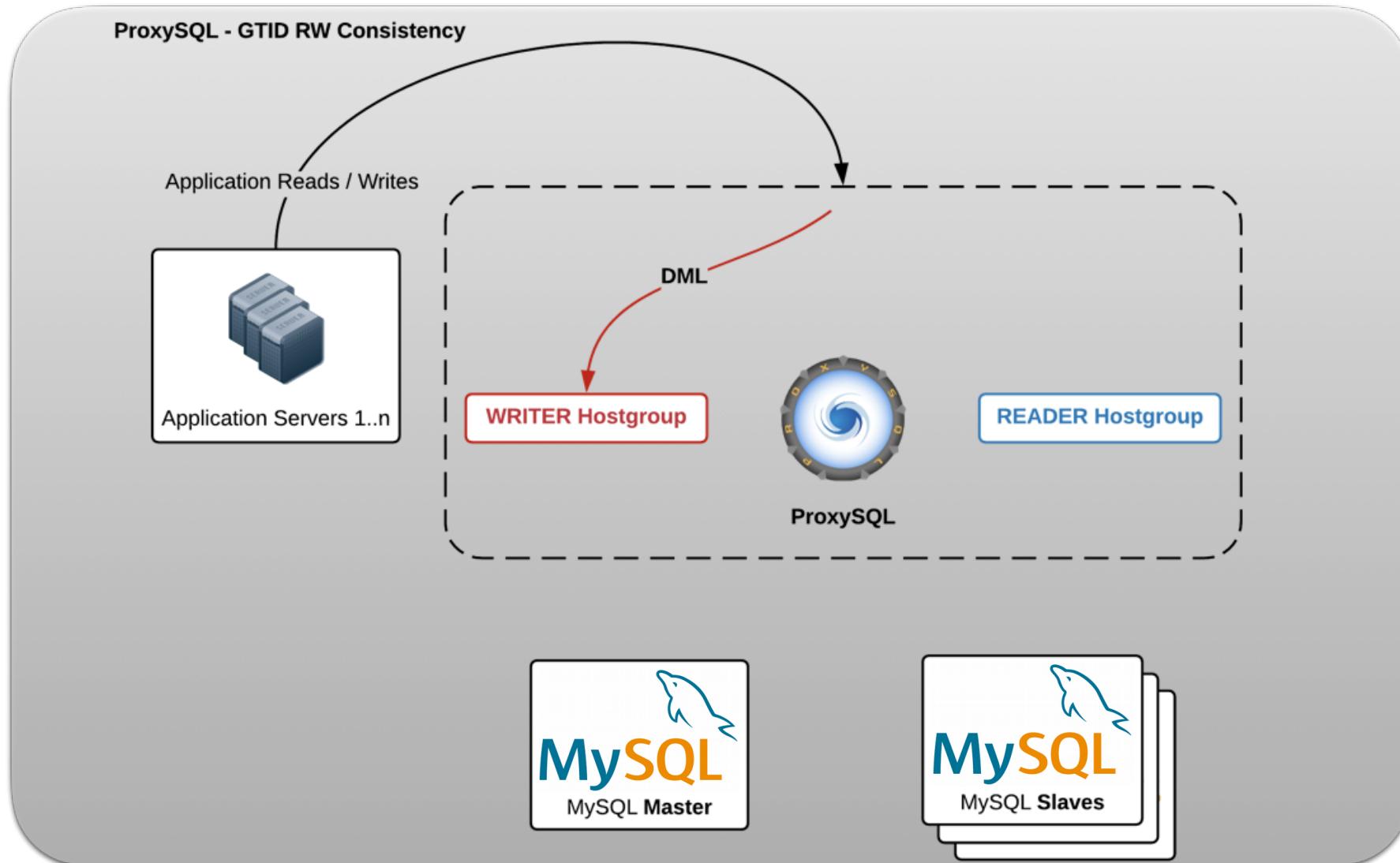
# How does ProxySQL achieve GTID R/W Consistency?

- ProxySQL can be configured to enforce GTID consistency for reads on any hostgroup / replication hostgroup
- The hostgroup will ensure that any subsequent DQL:
  - Will be routed only to hosts which have executed the previous transaction's GTID for the connection
  - Since the MASTER host will be part of the hostgroup / READER replication hostgroup (with a lower weight) there is always a node available to serve the DQL statement

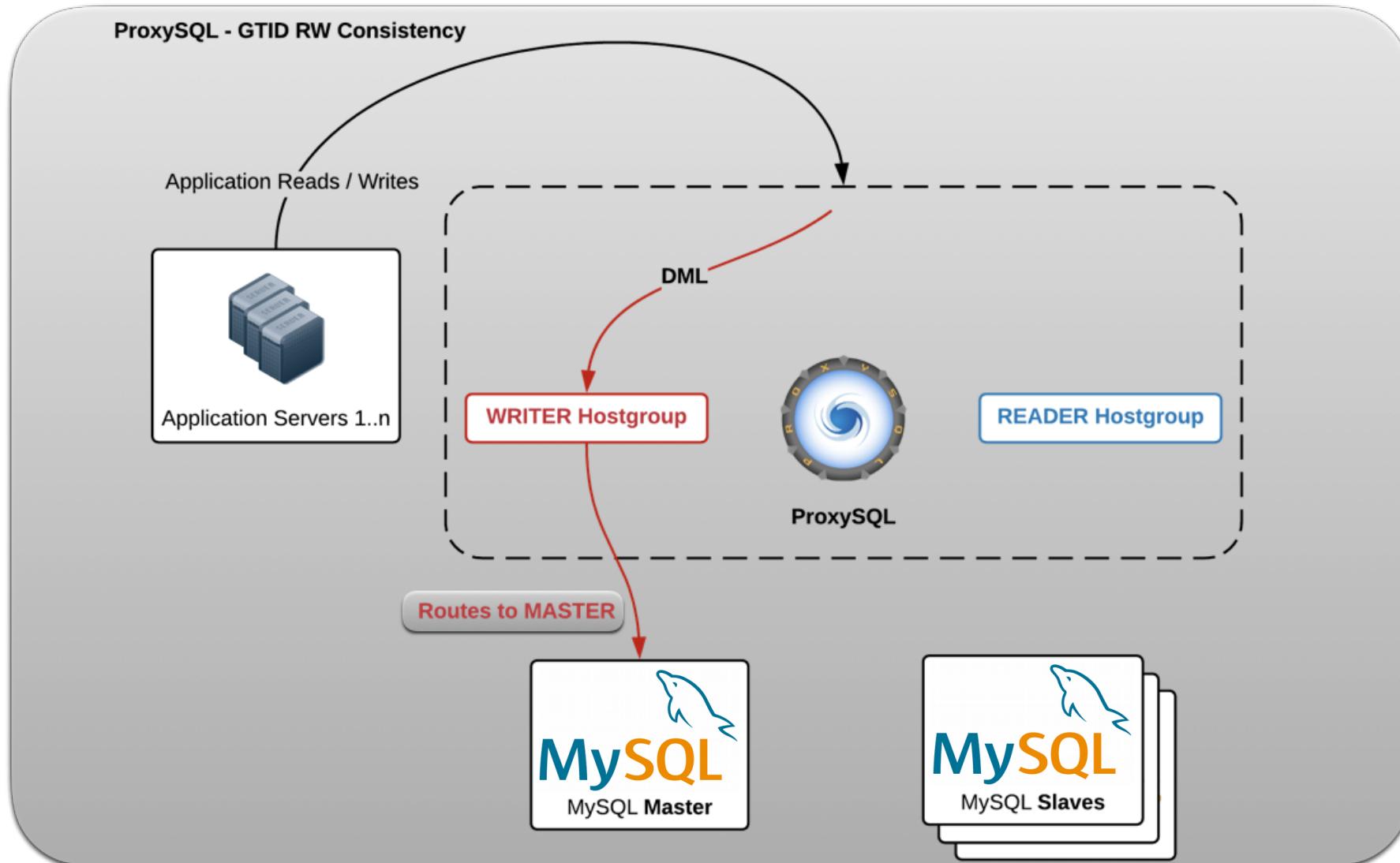
# ProxySQL Binlog Reader



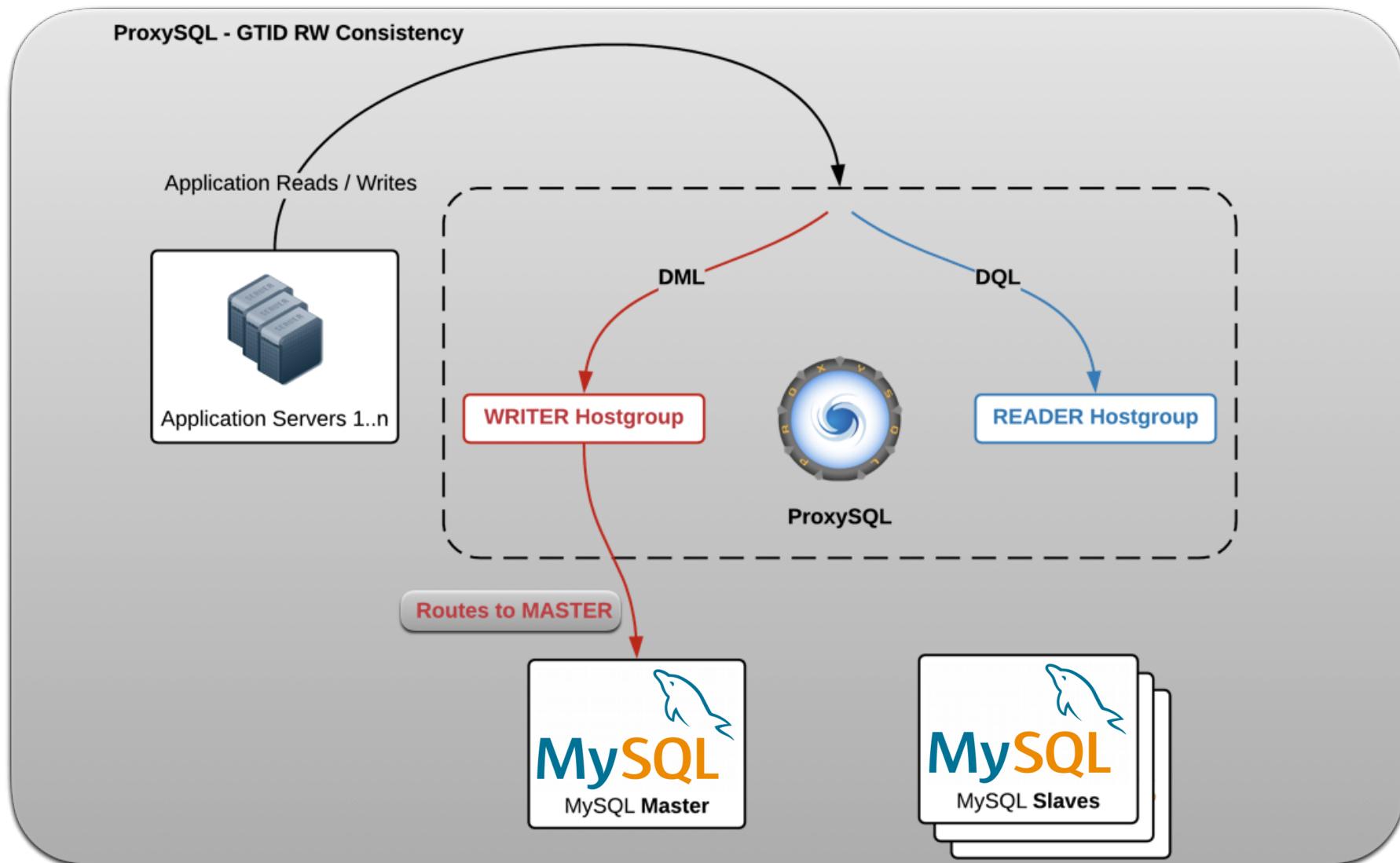
# ProxySQL Binlog Reader



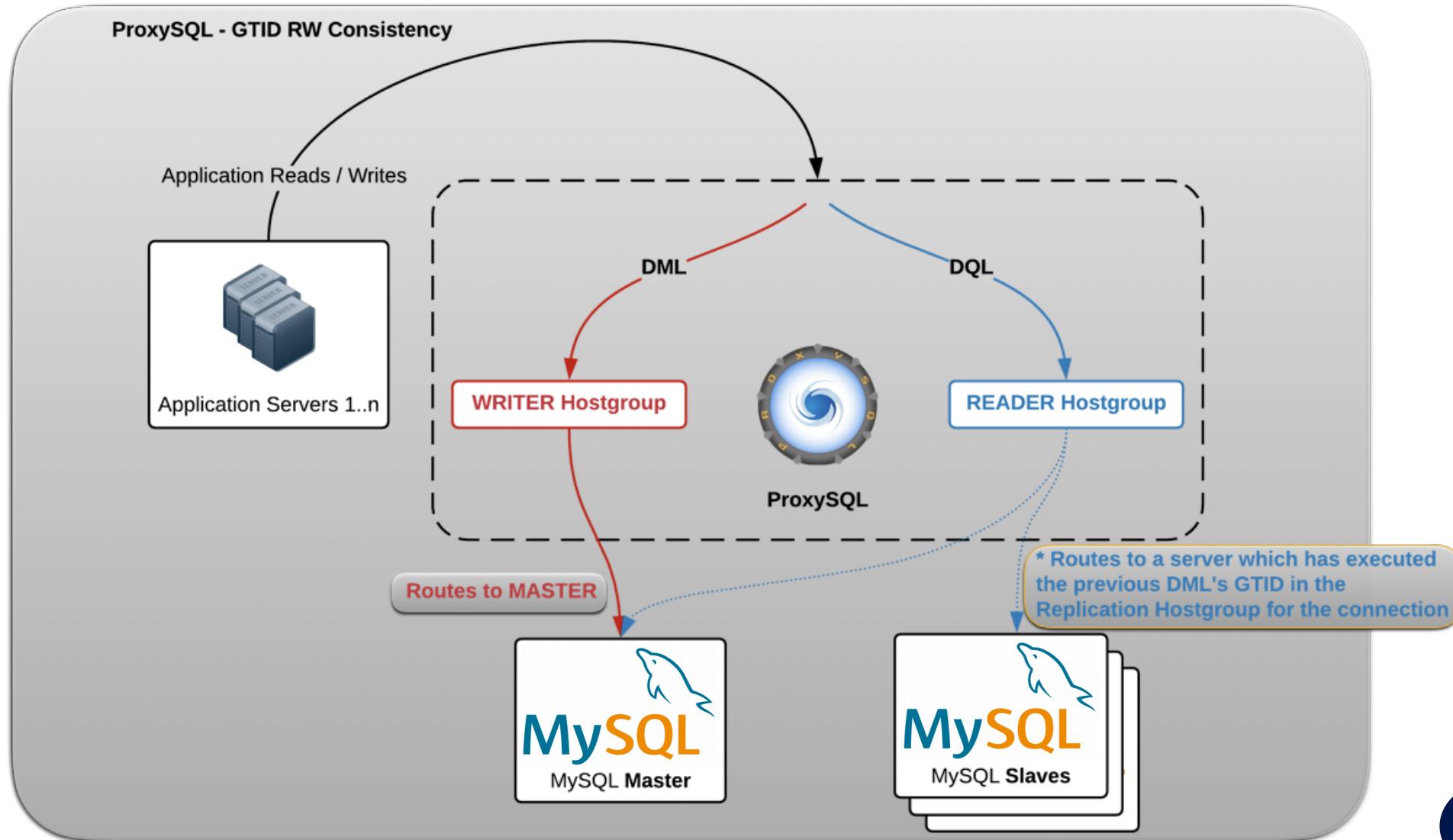
# ProxySQL Binlog Reader



# ProxySQL Binlog Reader



# ProxySQL Binlog Reader



# Supported Replication Models

- Master - Slave:
  - Asynchronous Replication
  - Semi-Synchronous Replication
- Multi - Master:
  - InnoDB Cluster / Group Replication
- Additional requirements:
  - **GTID** is required for all servers in the hostgroup which routes **GTID consistent queries**
  - The **binlog\_format** must be configured to **ROW**

# Live Demo

A demonstration of consistent reads with GTID and the components discussed in this presentation

# Demo Configuration

- 1x ProxySQL 2.0 instance
  - Proxysql1
- 3x MySQL 5.7 instances
  - Mysql1: Read / Write Master
  - Mysql2: Read Only Slave
  - Mysql3: Read Only Slave
- Python test script: “bin/gtid-tester”
  - Creates a “user” table and starts 4x threads (separate connections)
  - Each thread does 1000 iterations performing an INSERT followed by a COUNT(\*) on the table

# Demo MySQL Configuration

- Specific my.cnf variables of interest:
  - `binlog_format=ROW`
  - `gtid_mode=ON`
  - `enforce_gtid_consistency=true`
  - `session_track_gtids=OWN_GTID`

# Demo MySQL Configuration

Processes running in the MySQL Docker container

- MySQL (port 3306):
  - `mysqld`
- ProxySQL Binlog Reader (port 999):
  - `proxysql_binlog_reader \ -h 127.0.0.1 -u root -p xxxx -P 3306 \ -l 999 -L /var/log/binlog_reader.log`

# Demo ProxySQL Configuration

ProxySQL “mysql\_servers” configuration:

```
ProxySQL Admin> select hostgroup_id, hostname, port, gtid_port,  
status from runtime_mysql_servers;
```

hostgroup_id	hostname	port	gtid_port	status
0	mysql1	3306	999	ONLINE
1	mysql3	3306	999	ONLINE
1	mysql2	3306	999	ONLINE
1	mysql1	3306	999	ONLINE



# Demo ProxySQL Configuration

ProxySQL “mysql\_query\_rules” configuration (“gtid\_from\_hostgroup” defines which hostgroup determines the required GTID for a session):

```
ProxySQL Admin> select match_digest, destination_hostgroup,  
gtid_from_hostgroup from mysql_query_rules\G  
*****  
1. row *****  
match_digest: ^SELECT.*FOR UPDATE  
destination_hostgroup: 0  
gtid_from_hostgroup: NULL  
*****  
2. row *****  
match_digest: ^SELECT  
destination_hostgroup: 1  
gtid from hostgroup: 0
```

# Demo ProxySQL GTID Statistics

ProxySQL “stats\_mysql\_gtid\_executed” to view status of GTID tracking:

```
ProxySQL Admin> select * from stats_mysql_gtid_executed where  
hostname='mysql1'\G
```

```
***** 1. row *****
```

```
hostname: mysql1
```

```
port: 3306
```

```
gtid_executed: 85c17137-4258-11e8-8090-0242ac130002:1-65588
```

```
events: 65581
```

# Demo ProxySQL GTID Statistics

```
ProxySQL Admin> select hostname,gtid_executed from stats_mysql_gtid_executed  
order by hostname\G  
*****  
1. row *****  
hostname: mysql1  
gtid_executed: 85c17137-4258-11e8-8090-0242ac130002:1-146301  
*****  
2. row *****  
hostname: mysql2  
gtid_executed: 85c17137-4258-11e8-8090-0242ac130002:1-146300,8a093f5f-4258-11e8-8037-0242ac130004:1-5  
*****  
3. row *****  
hostname: mysql3  
gtid_executed: 85c17137-4258-11e8-8090-0242ac130002:1-146301,8a0ac961-4258-11e8-8003-0242ac130003:1-5
```

# Demo ProxySQL GTID Statistics

ProxySQL “stats\_mysql\_connection\_pool” shows how many queries were executed using GTID causal reads in the “Queries\_GTID\_sync” column:

```
ProxySQL Admin> select hostgroup, srv_host, queries, queries_gtid_sync  
from stats_mysql_connection_pool;
```

hostgroup	srv_host	Queries	Queries_GTID_sync
0	mysql1	603677	0
1	mysql1	480665	8270
1	mysql2	638147	8570
1	mysql3	631756	8387

# Sample ProxySQL Binlog Reader Logfile

```
$ cat /var/log/binlog_reader.log
```

Starting ProxySQL MySQL Binlog

Sucessfully started

**Angel process started ProxySQL MySQL Binlog process 242**

**2018-04-17 16:04:05 [INFO] Initializing client...**

**8a093f5f-4258-11e8-8037-0242ac130004:1-5,85c17137-4258-11e8-8090-  
0242ac130002:1-7**

**2018-04-17 16:04:05 [INFO] Reading binlogs...**

# Sample ProxySQL Binlog Reader Stream

SrcID is only sent when it changes, other the bare minimum  
TransactionID is sent (on client initialisation full GTID history available is  
sent), for example:

ST=85c17137-4258-11e8-8090-0242ac130002:1-209419

I1=85c17137425811e880900242ac130002:209420

I2=209421

I2=209422

I2=209423

I2=209424

I2=209425

# Thank You!

# Rate My Session

Schedule  
Timezone: Europe/Berlin +02:00

MON 3 TUE 4 WED 5

11:20

ClickHouse: High Performance  
Distributed  
11:20 - 12:10, Matterhorn 2

**TAP THE SESSION**

Introducing gh-ost: triggerless, painless, trusted online schema migrations  
11:20 - 12:10, Matterhorn 2

MongoDB query monitoring  
11:20 - 12:10, Matterhorn 3

MySQL Load Balancers - MaxScale, ProxySQL, HAProxy, MySQL Router  
Ganglia, ngine - a close up look  
11:20 - 12:10, Zurich 1

Securing your MySQL/MariaDB data  
11:20 - 12:10, Zurich 2

MySQL, and Ceph: A tale of two friends

Details

Introducing gh-ost: triggerless, painless, trusted online schema migrations

⌚ 11:20 → 12:10  
📍 Matterhorn 2

**Rate & Review**

**TAP TO RATE & REVIEW**

gh-ost is a MySQL tool which changes the paradigm of MySQL online schema changes, designed to overcome today's limitations and difficulties in online migrations.

**SPEAKERS**

Shlomi Noach  
Senior Infrastructure Engineer  
GitHub

Tom Kreuper  
Sr. Database Infrastructure Eng.  
GitHub

X Rate & Review

Tap a star to rate

Feedback (optional)

Anonymously

**SUBMIT**