ProxySQL - GTID Consistent Reads

Adaptive query routing based on GTID tracking

Introduction

Rene Cannao

- Founder of ProxySQL
- MySQL DBA

Introduction

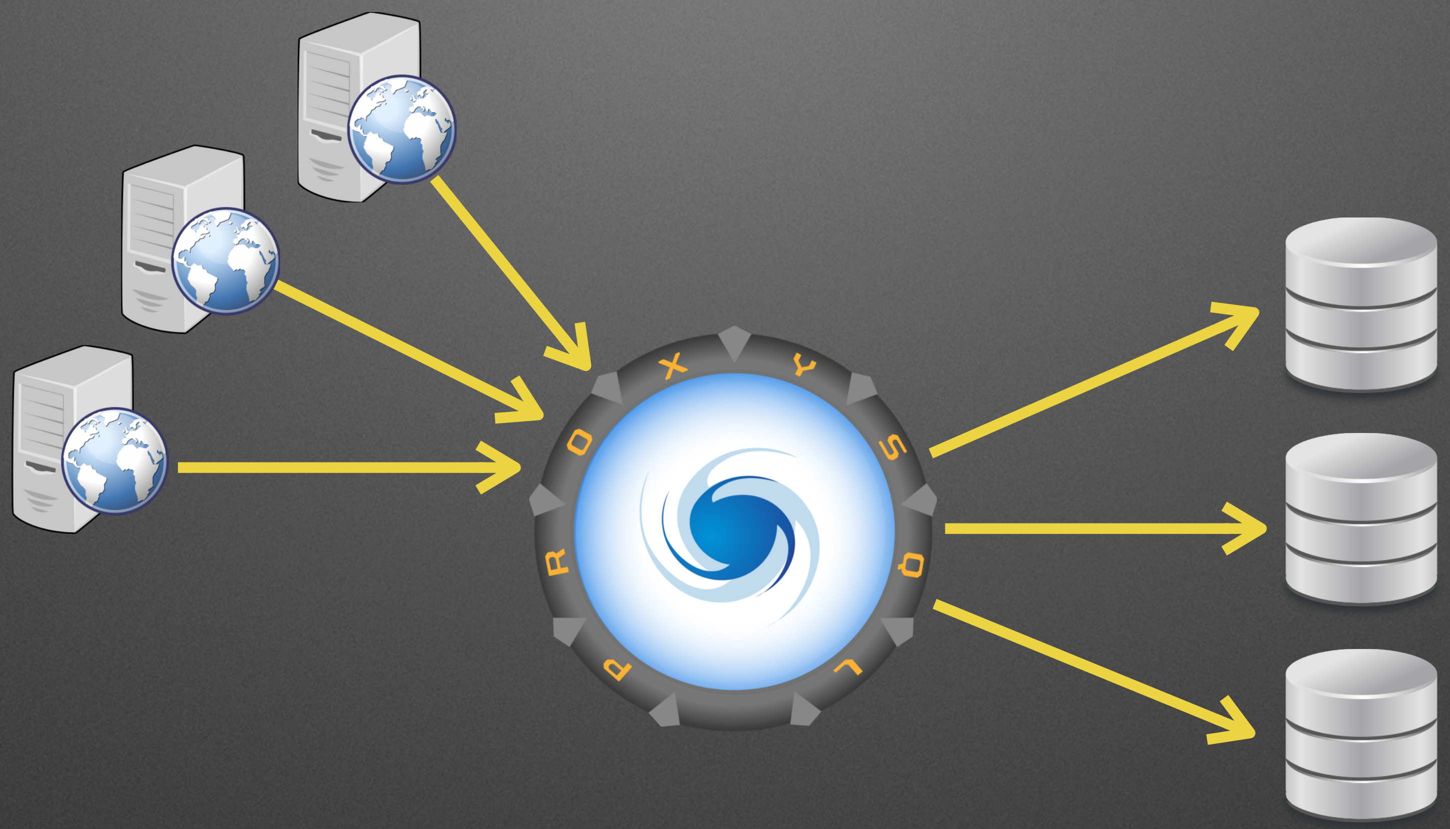
Nick Vyzas

- ProxySQL Committer
- MySQL DBA

What is ProxySQL?

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

Architecture Overview

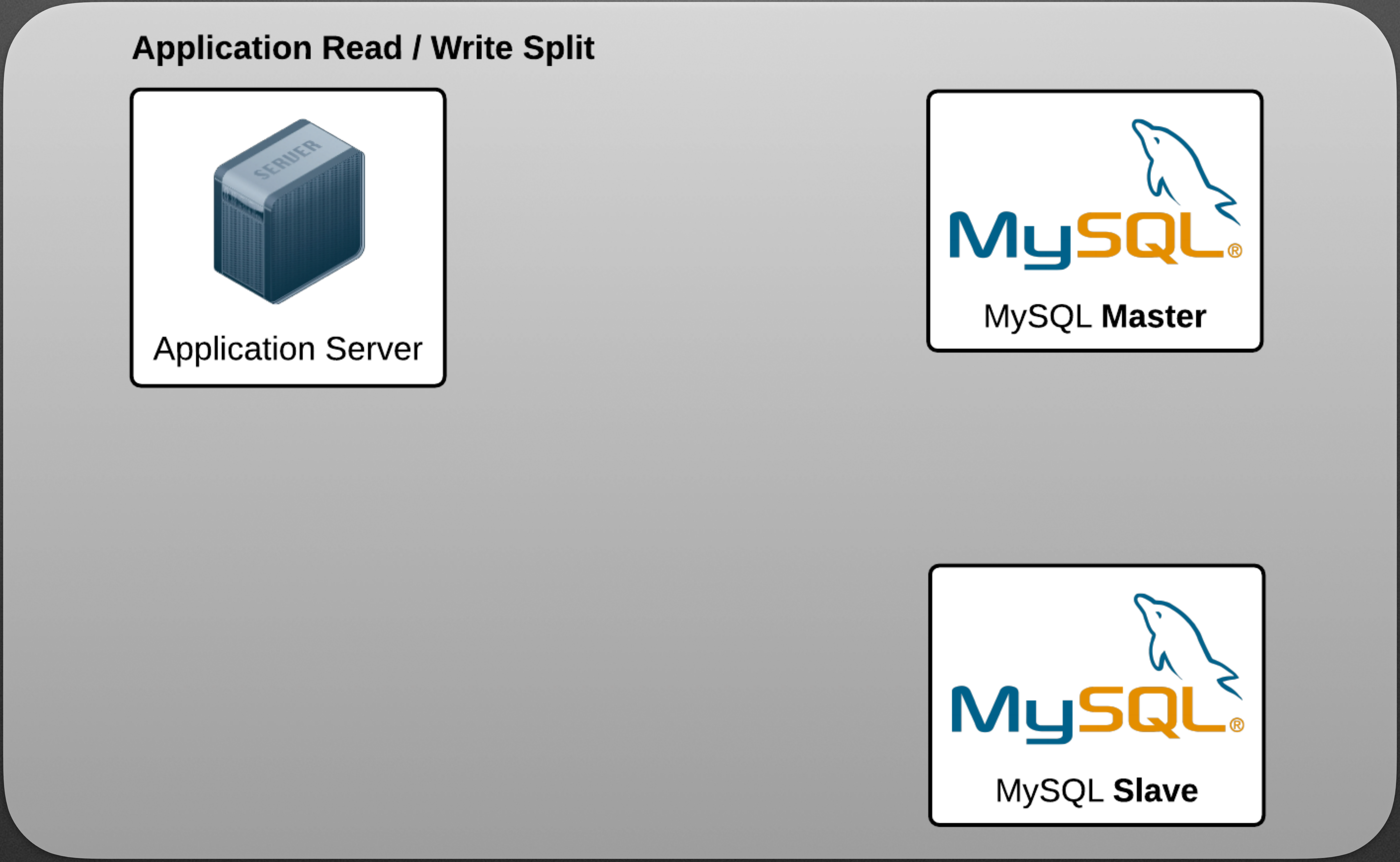


- Clients connect to ProxySQL
- Requests are evaluated
- Actions are performed

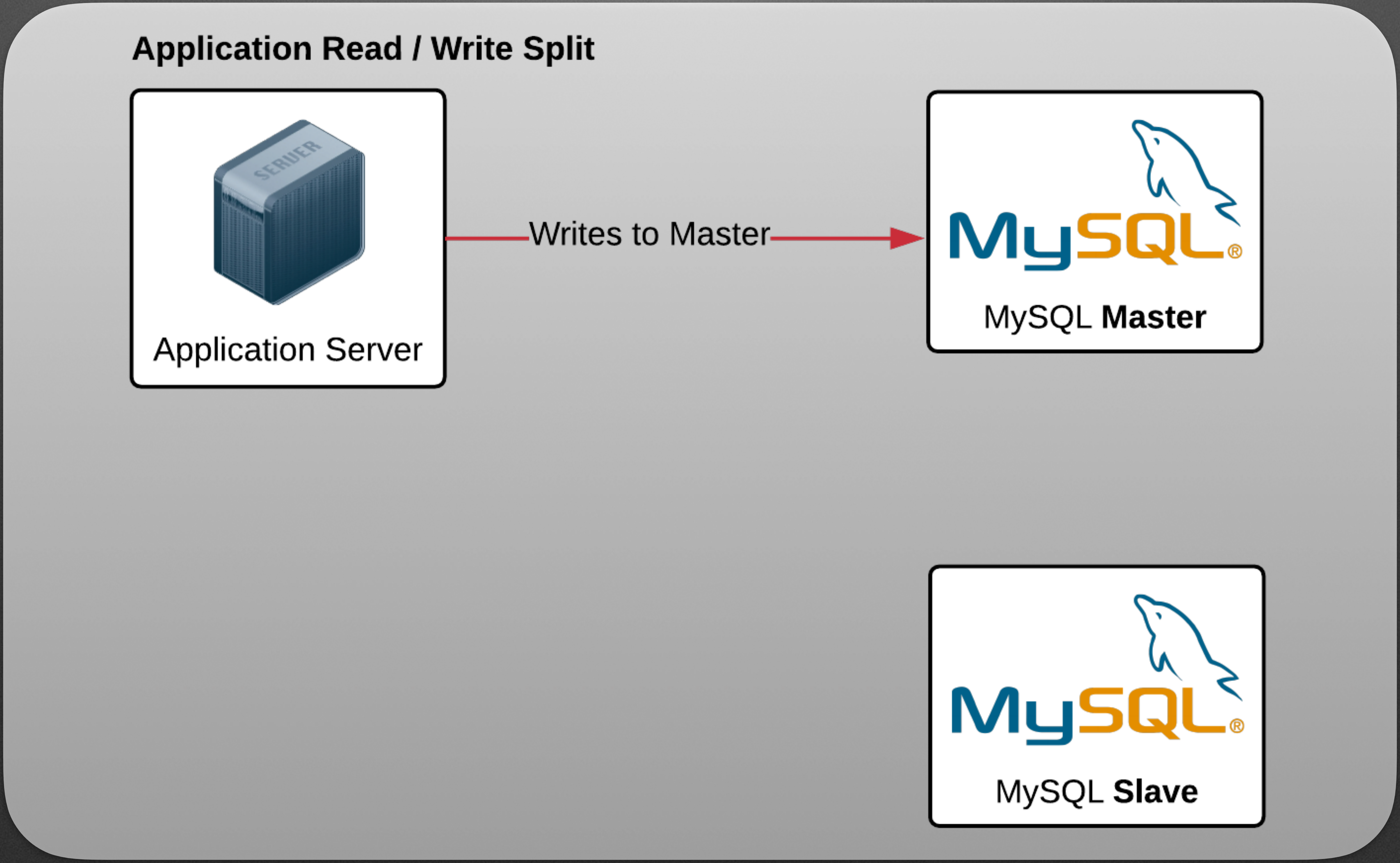
Master - Slave Replication

- Asynchronous replication
  - Replication lag is the <u>major challenge</u>
- 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

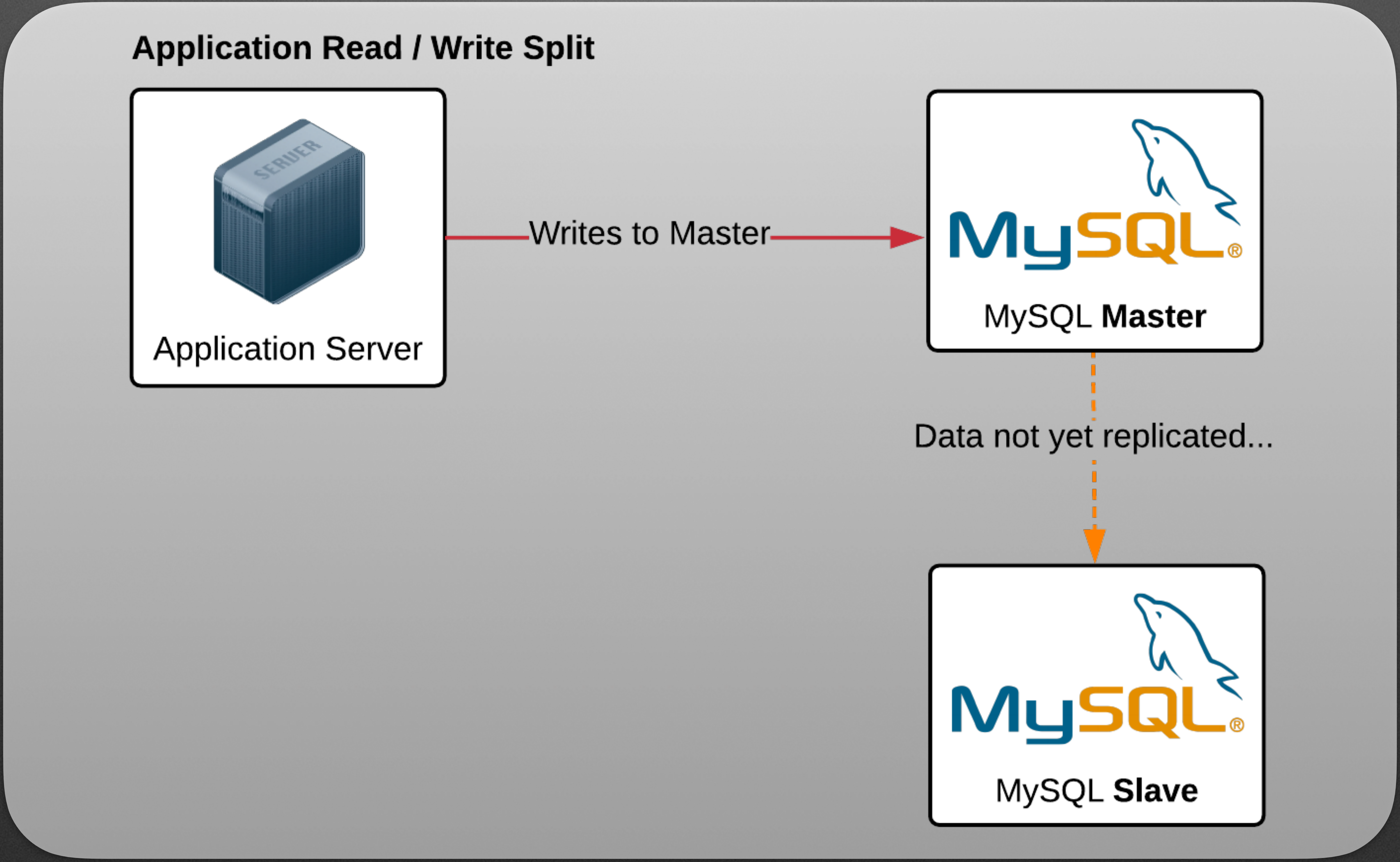
Application Read / Write Split



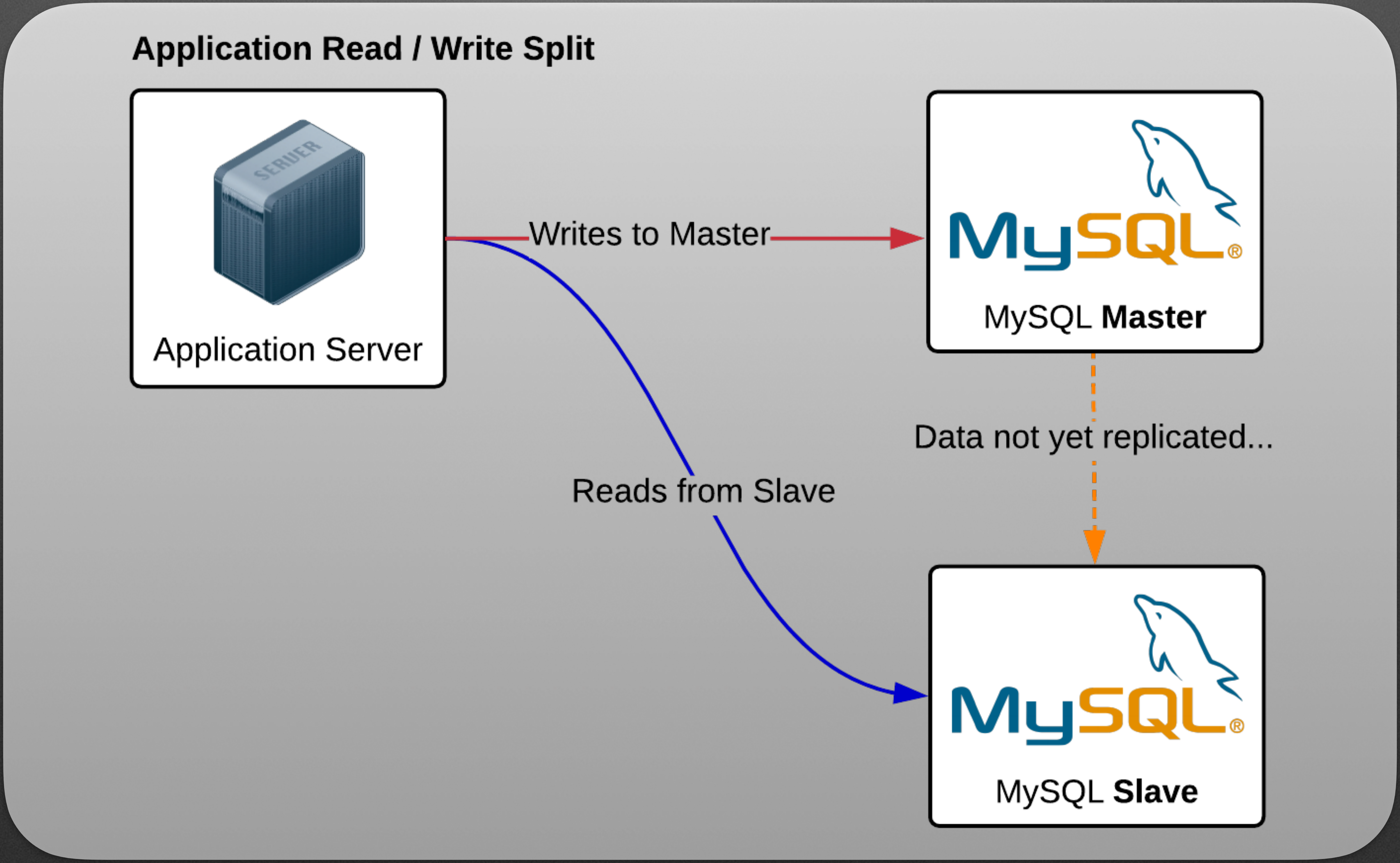
Application Read / Write Split



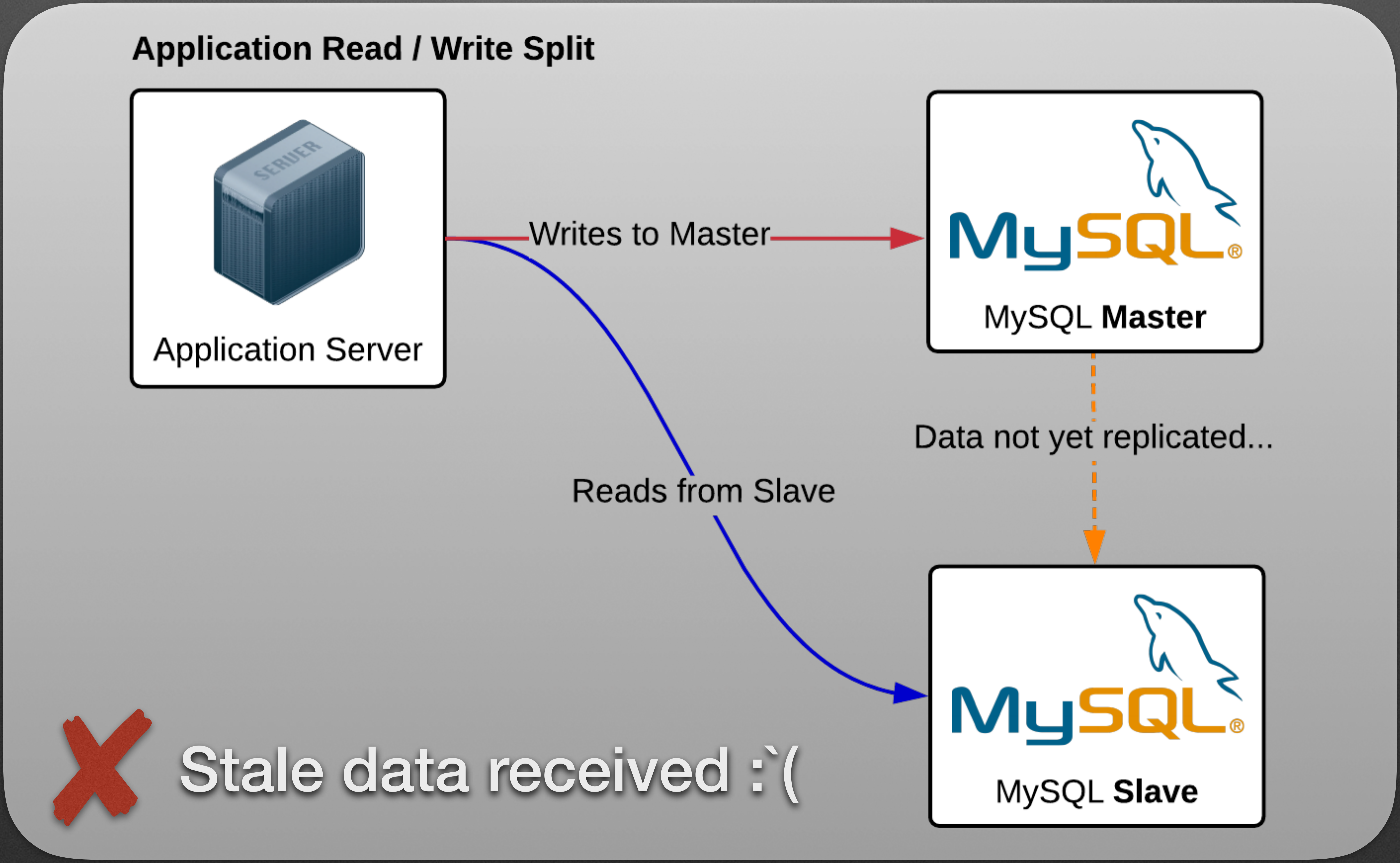
Application Read / Write Split



Application Read / Write Split

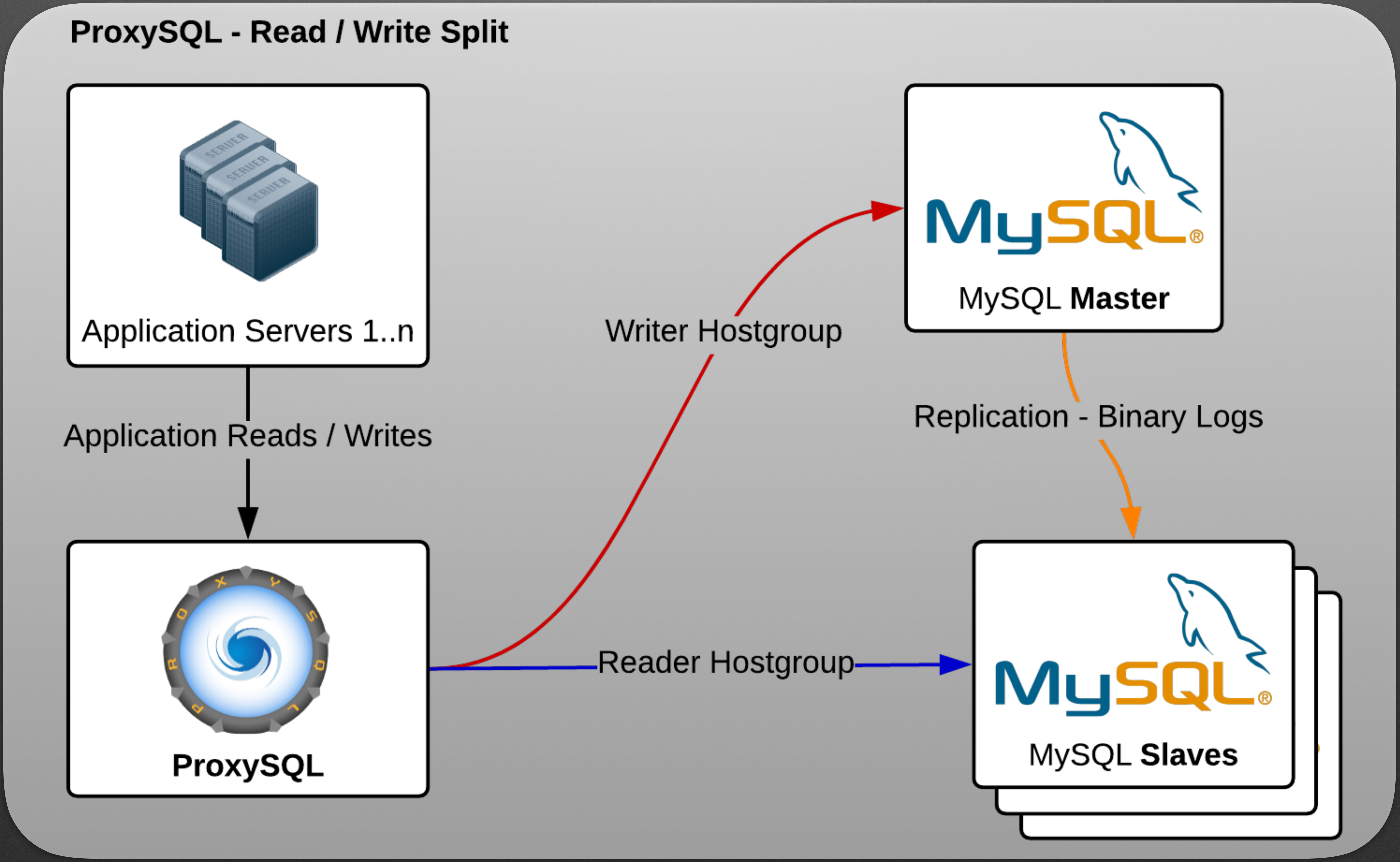


Application Read / Write Split



Stale data received :`(

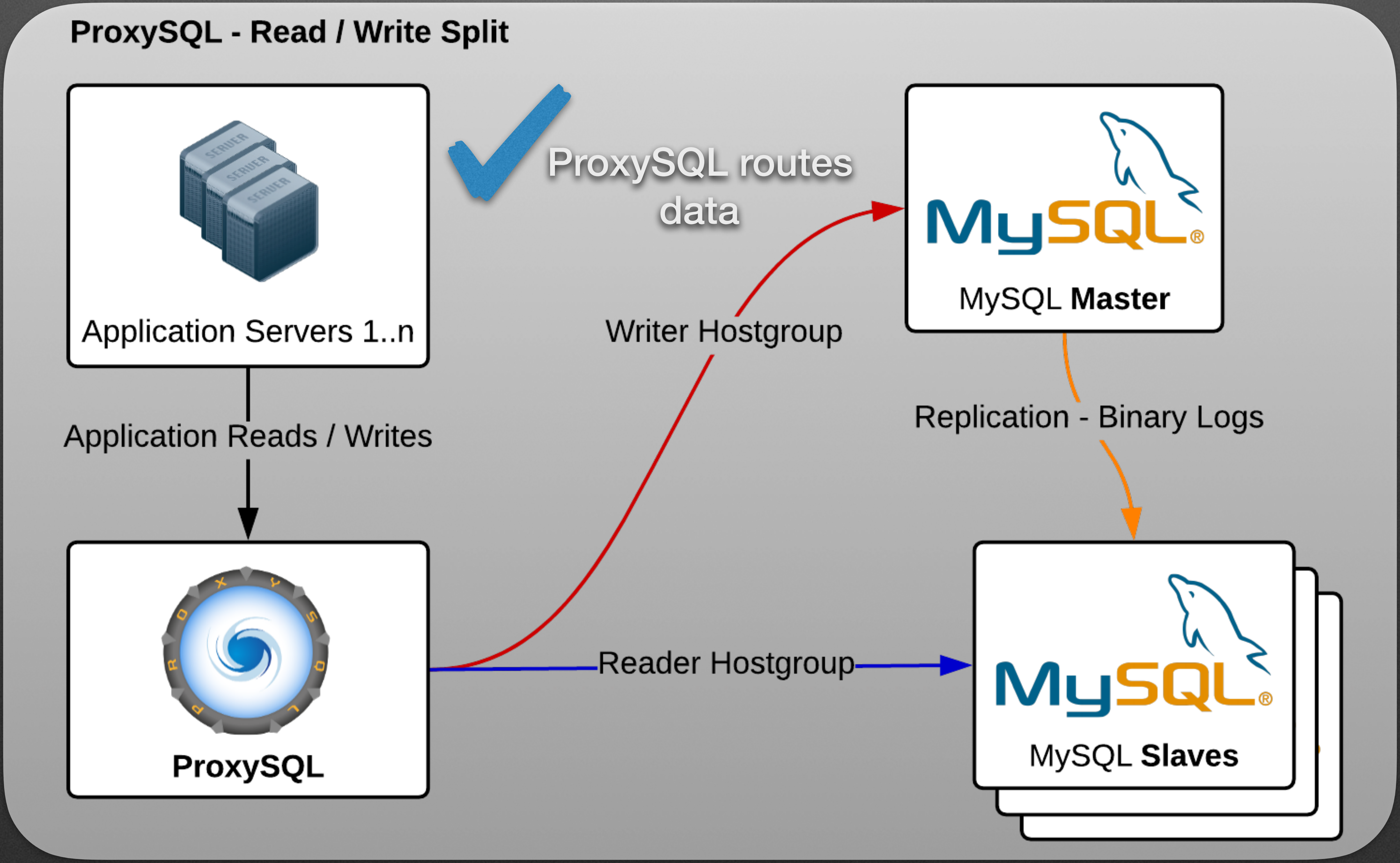
ProxySQL Read / Write Split



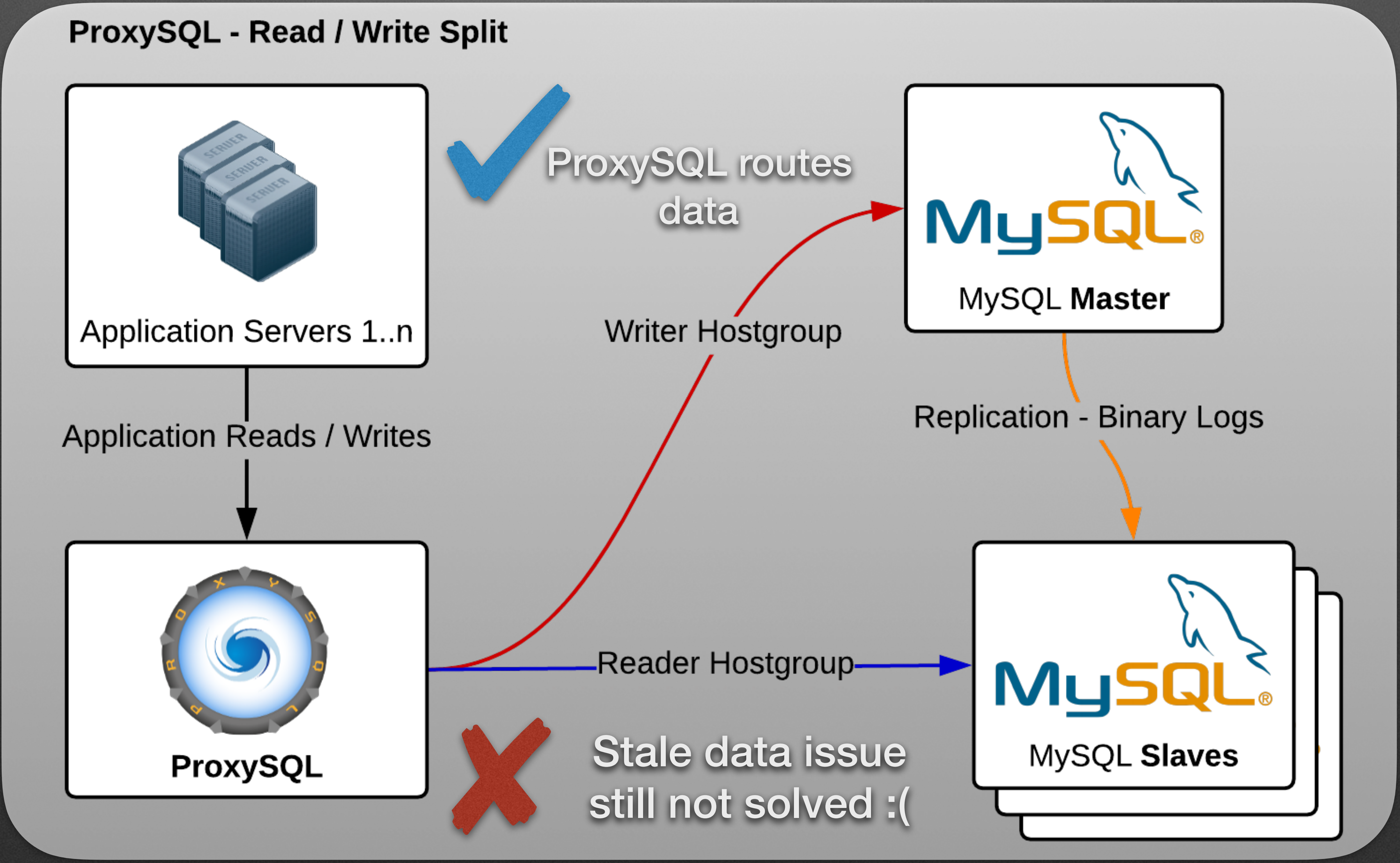
Benefits of ProxySQL's Read / Write Split

- 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

ProxySQL Read / Write Split



ProxySQL Read / Write Split


ProxySQL - Read / Write Split
Application Servers 1..n
Application Reads / Writes
ProxySQL
ProxySQL routes data
Writer Hostgroup
Reader Hostgroup
MySQL Master
Replication - Binary Logs
MySQL Slaves
Stale data issue still not solved :(

Challenges of R/W Split

- Susceptible to service stale data due to replication lag
- Replication lag is 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

Traditional binlog replication

- Traditional replication requires master & slave binary log file / position to be 100% synchronized
- 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 must 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 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

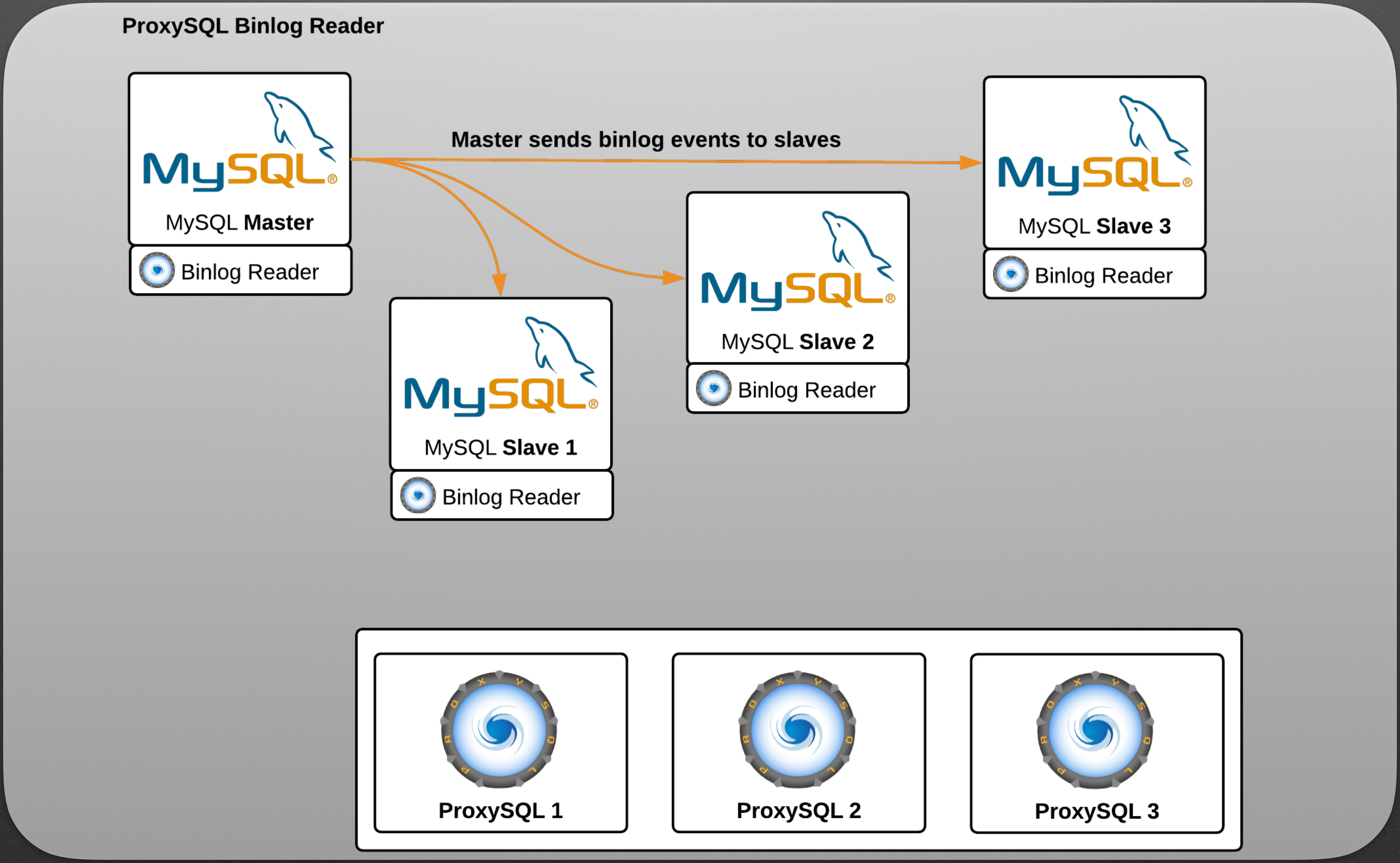
- The "push method" is far more efficient and results in **less requests** and **lower latency**

  - Especially important in large scale deployments

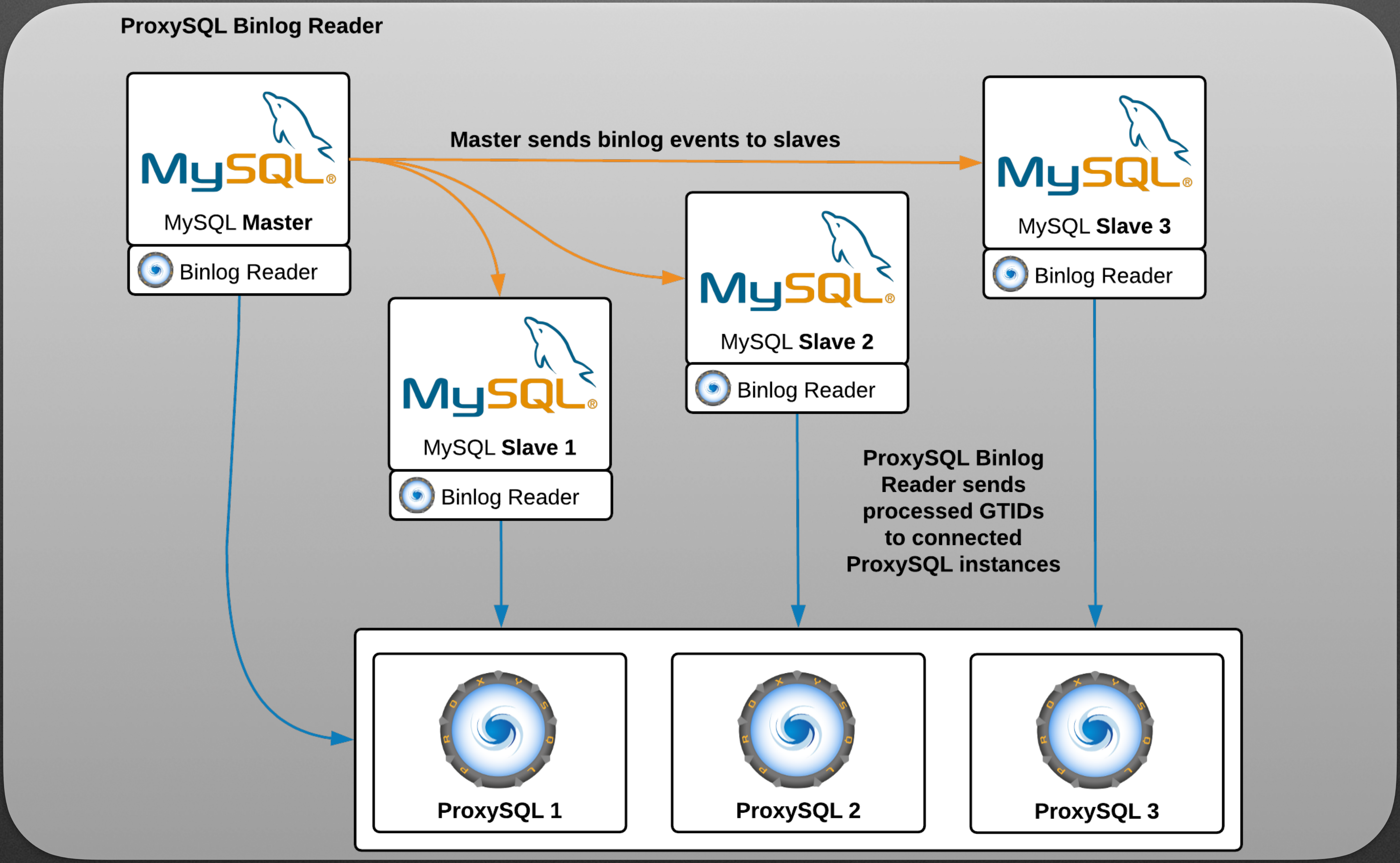
ProxySQL Binlog Reader

- 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
- Features an auto-restart mechanism in case of failure and a client side reconnect

ProxySQL Binlog Reader



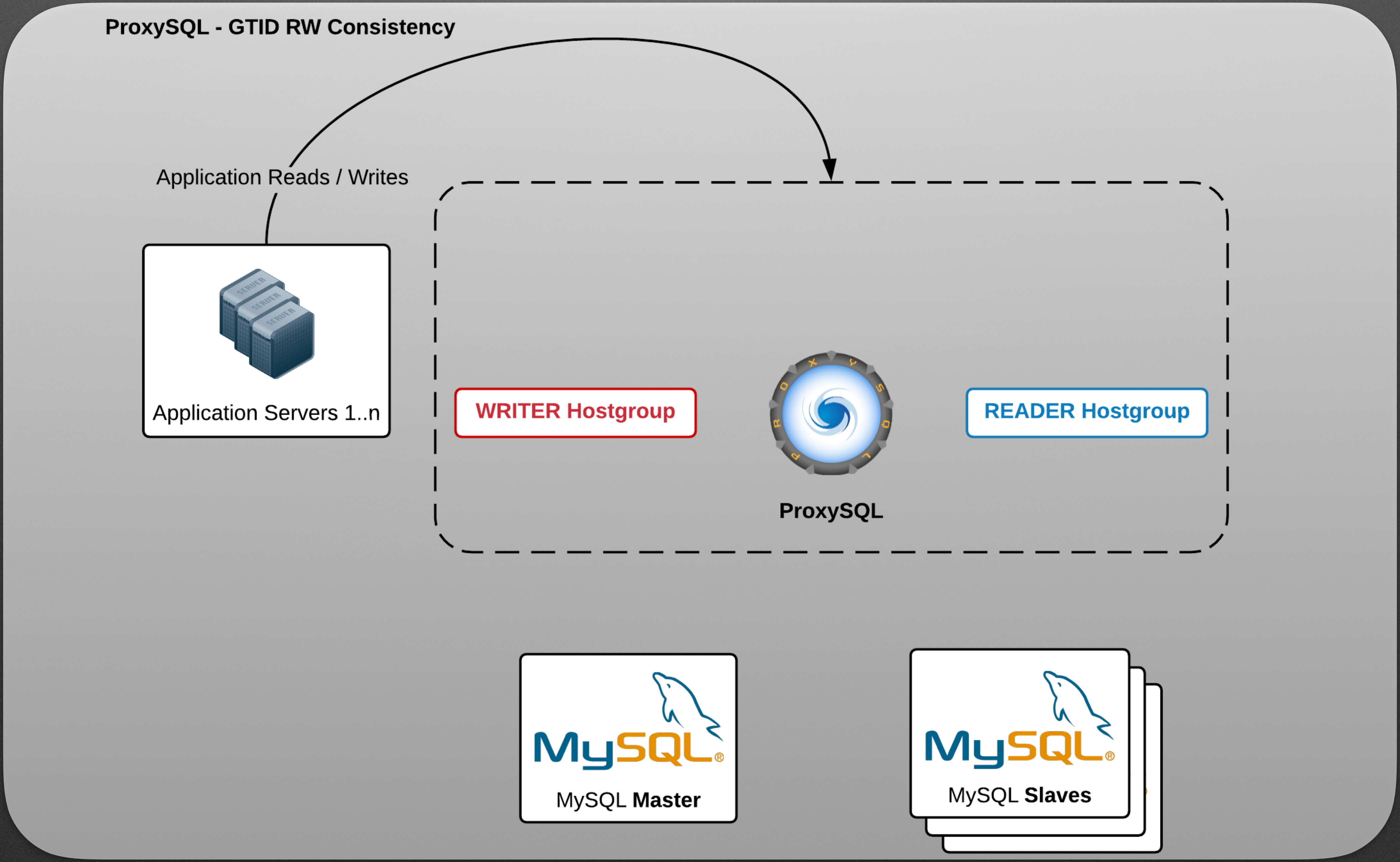
ProxySQL Binlog Reader



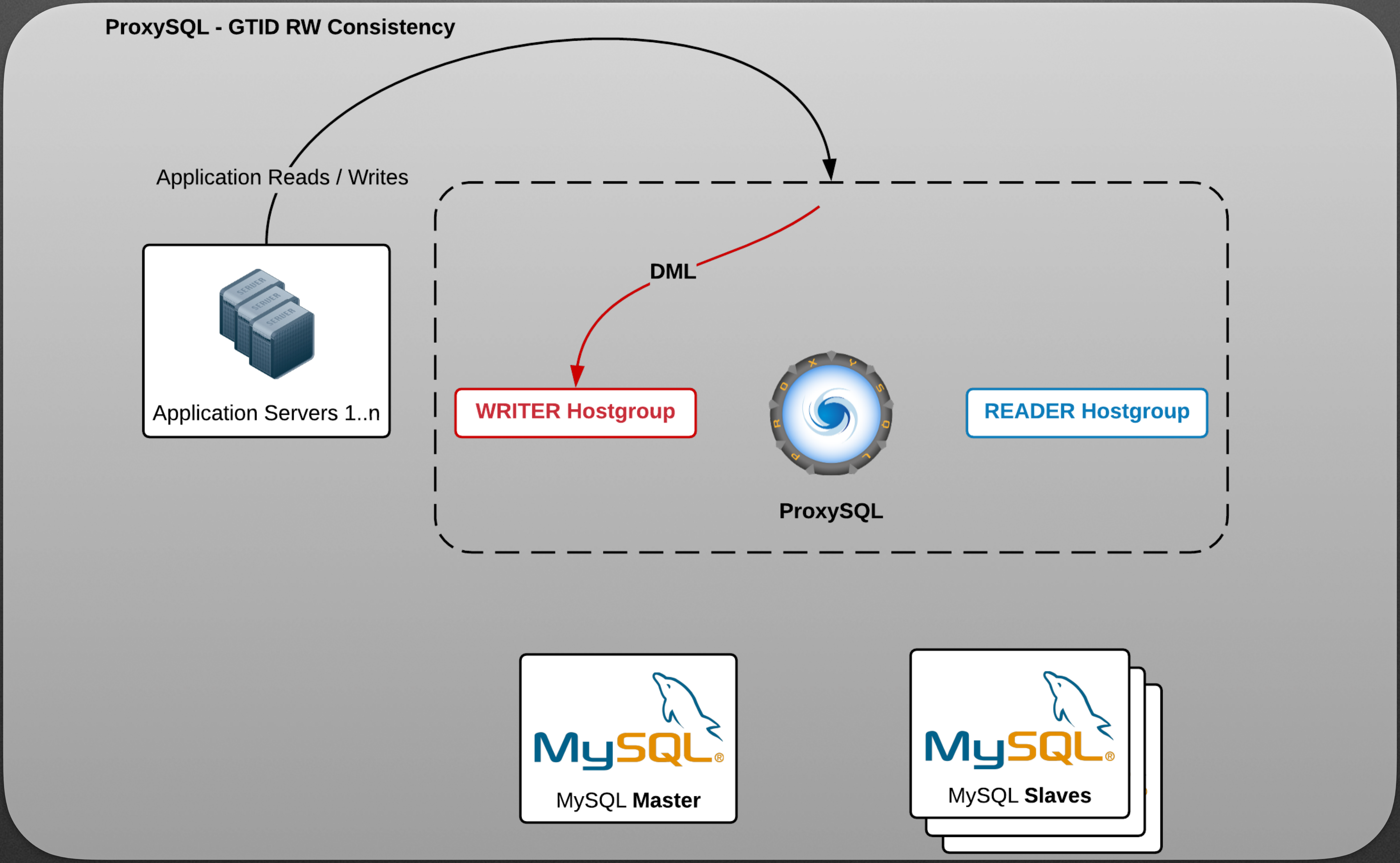
How does ProxySQL achieve GTID R/W Consistency?

- 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

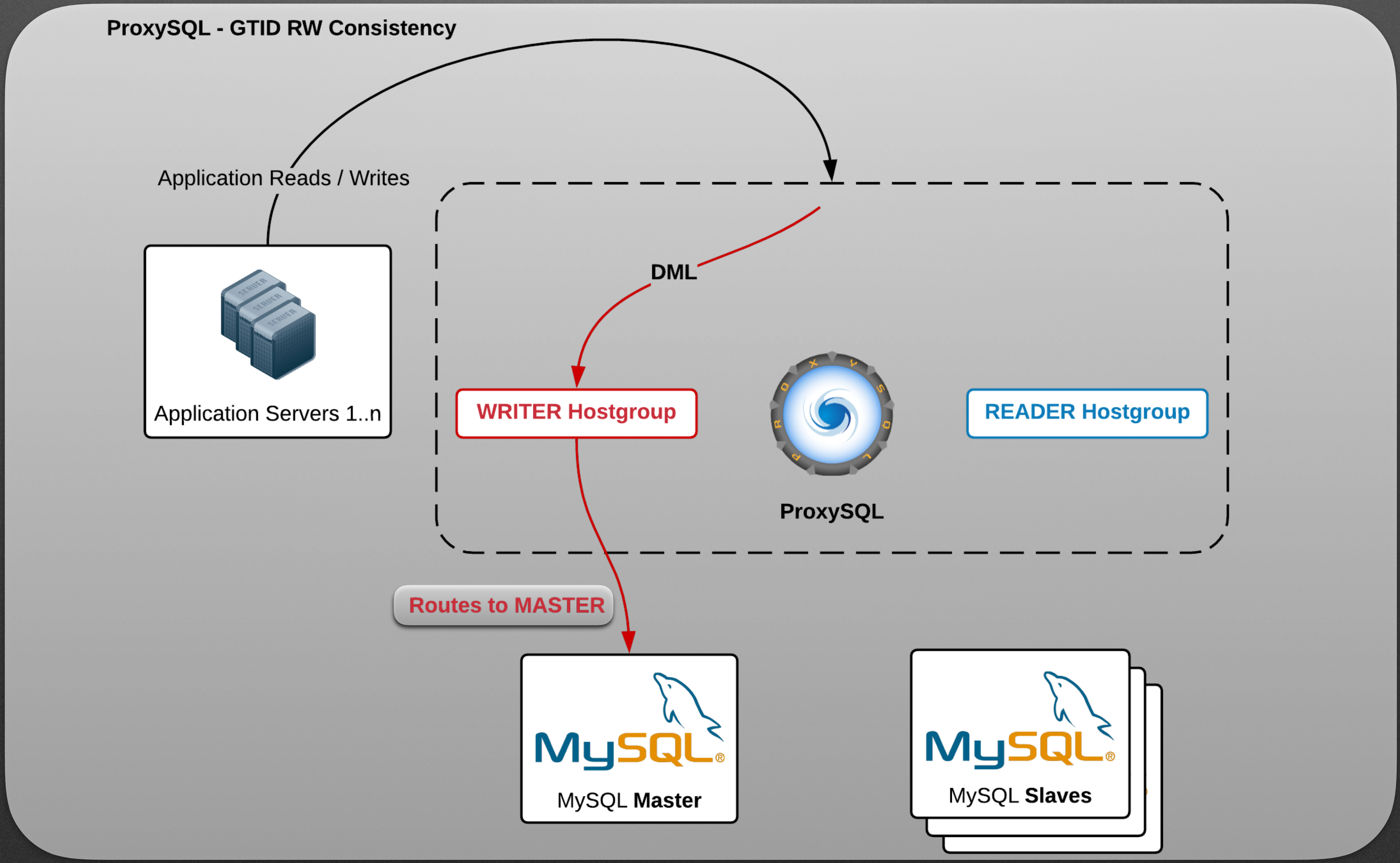
GTID R/W Consistency Flow



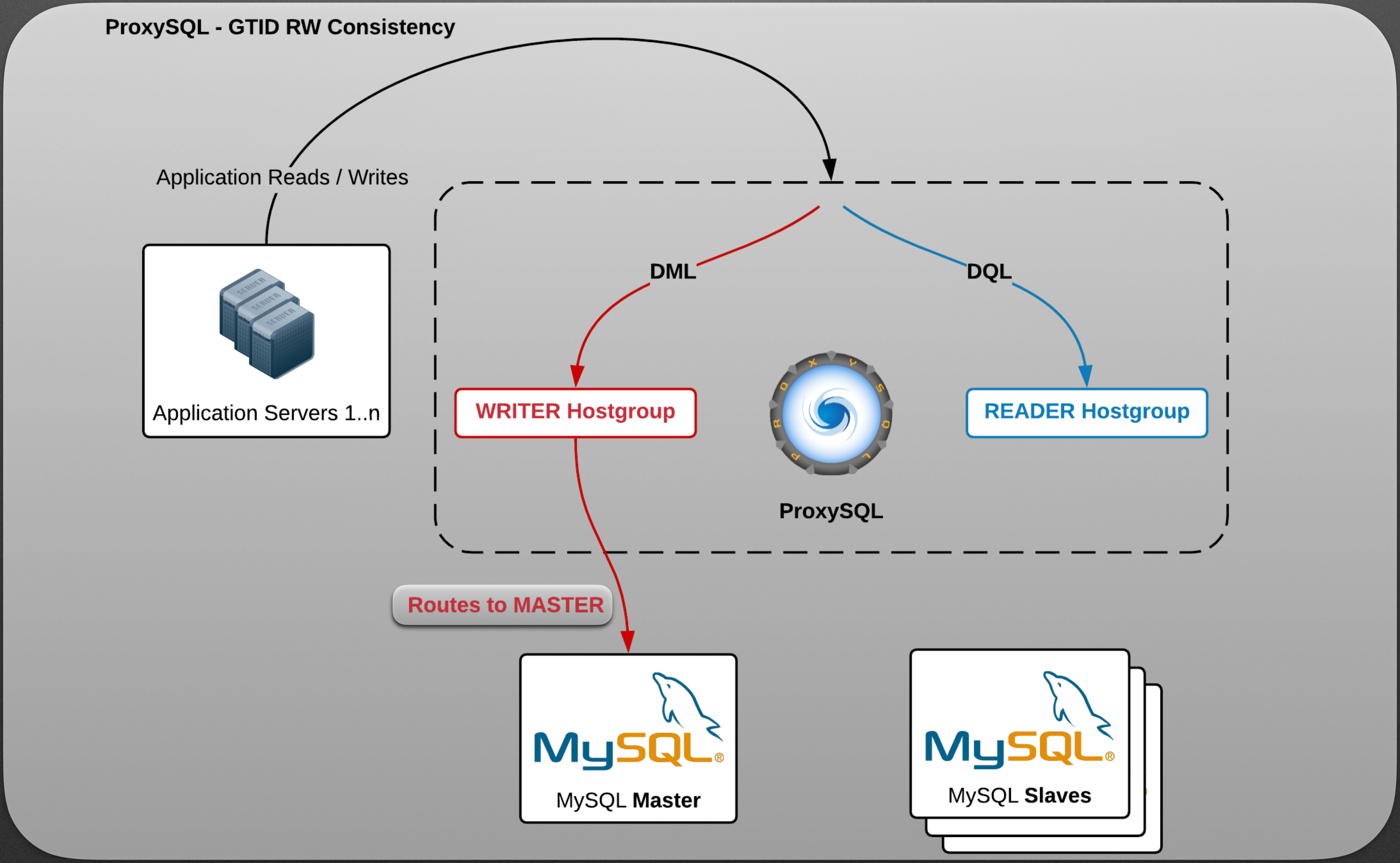
GTID R/W Consistency Flow



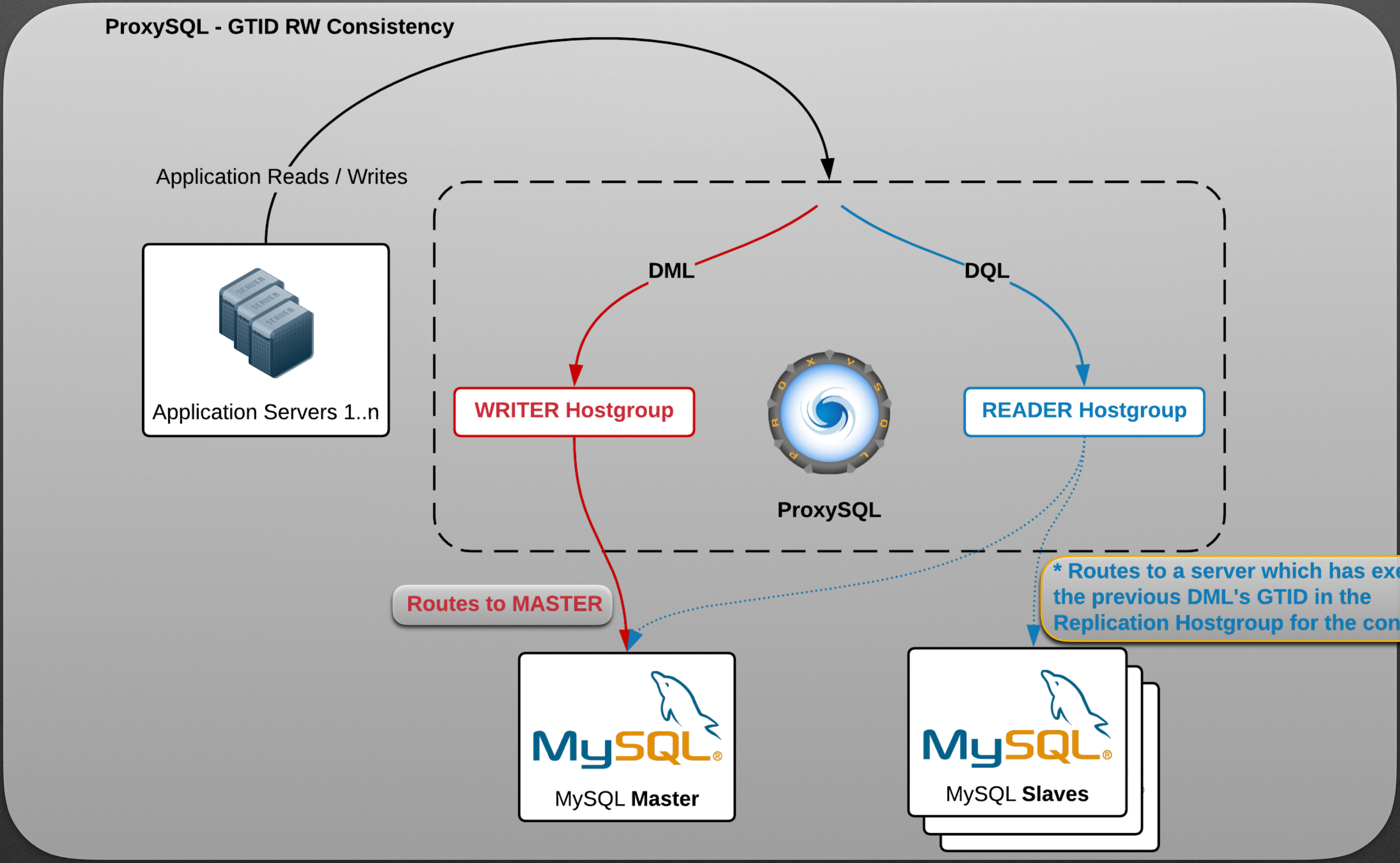
GTID R/W Consistency Flow



GTID R/W Consistency Flow



GTID R/W Consistency Flow


ProxySQL - GTID RW Consistency
Application Reads / Writes
Application Servers 1..n
DML
DQL
WRITER Hostgroup
READER Hostgroup
ProxySQL
Routes to MASTER
* Routes to a server which has executed the previous DML's GTID in the Replication Hostgroup for the connection
MySQL Master
MySQL Slaves

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**

What time is it?

Thank you!

- Please remember to report feature requests and bug reports: https://github.com/sysown/proxysql/
- Community support can be found on our forum: https://groups.google.com/forum/#!forum/proxysql
- Useful blog articles are available at our site: http://proxysql.com/blog
- Visit us at http://proxysql.com/support for subscription and support options