ProxySQL

Mirroring

ProxySQL LLC

We provide services to help **build**, **support** and **improve** the performance & reliability of your Cloud-Based and On-Premise MySQL infrastructure.

ProxySQL LLC

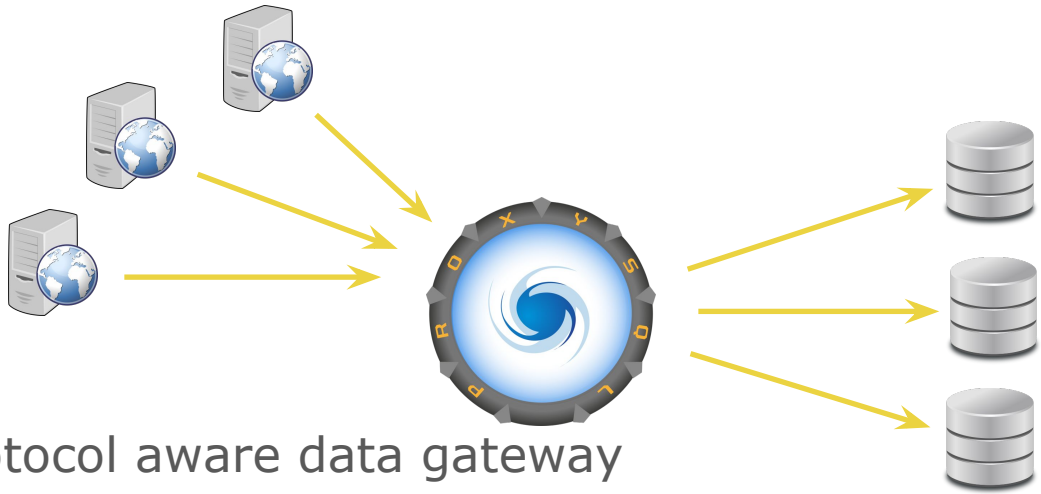
- ProxySQL Development
- ProxySQL Support Services
- ProxySQL, MySQL, DevOps & Outsourcing
- Consulting Services

About me

Rene Cannao

- Author of ProxySQL
- Founder of ProxySQL LLC
- MySQL DBA

What is ProxySQL?



MySQL protocol aware data gateway

- Clients connect to ProxySQL
- Requests are **evaluated**
- Various **actions** are performed

ProxySQL features

- MySQL, MariaDB, Percona and ClickHouse backends
- Supports millions of users and tens of thousands of database servers
- High Availability and infinite scalability
- Dynamic runtime reconfiguration
- Seamless planned and unplanned failover
- MySQL Session Load balancing

ProxySQL features

- Connection pooling and multiplexing
- Read caching outside of the database server
- Complex query routing and read/write split
- Query throttling, firewalling and mirroring
- On-the-fly query rewrite
- Advanced data masking

ProxySQL features

- Real time statistics and monitoring
- Scheduling of external scripts
- Support for PXC and Group Replication
- Native ProxySQL Clustering
- Launching multiple instances on same port

Whats new in ProxySQL 2.0?

- SSL support for frontend & SSL v1.2
- Native support for Galera Replication
- Integration with Amazon RDS Aurora (more to come)
- Causal reads by tracking GTIDs across backend servers
- Support for LDAP authentication

Mirroring 101

What mirroring is not…

- It is not replication
- It does not guarantee the order of execution
- By no means is data consistency guaranteed

Internal Design

- Each client creates a MySQL Session
- Rules define what needs to be mirrored
- When mirrored is enabled, a MySQL Session with the right user/schema/query is added in a queue. The mirrored session has no client

Internal Design

- MySQL threads execute the queries in the queue
- The resultset/ok/error are never returned to the client
- Statistics are collected
- Errors are logged

Queue details

- mysql-mirror_max_concurrency : 16
- mysql-mirror_max_queue_length : 32000

- Mirror_queue_length

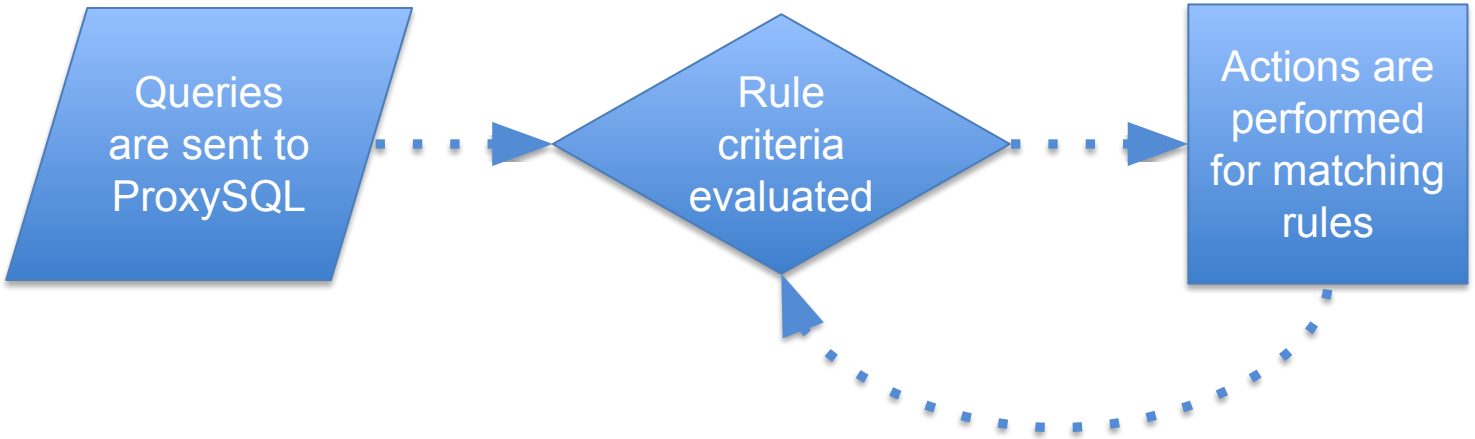
MySQL Query Rules

Define the query routing policies and processing behavior of ProxySQL, this is a **key area** of the application where the following are configured:

- Query caching
- Query routing / sharding
- Query firewalling
- **Mirroring** and more…

Working with query rules

- Incoming queries are evaluated against query rule criteria
- Actions are processed on queries matching query rules



MySQL Query Rules

Matching criteria

- username
- schemaname
- flagIN/OUT
- client_addr
- proxy_addr
- proxy_port
- digest
- match_digest
- match_pattern
- negate_match_pattern

Action fields:

- flagOUT
- replace_pattern
- destination_hostgroup
- cache_ttl
- timeout
- retries
- delay
- **mirror_flagOUT**
- **mirror_hostgroup**
- OK_msg
- error_msg
- log
- multiplex
- apply

Working with query rules

Query rules are configured in the “mysql_query_rules” table for MySQL.

- Each query rule must have a unique integer based rule_id
- Rules are processed in the order of the defined rule_id one after the other
- Using the fields “flagIN”, “flagOUT” and “apply” we can create chains of rules i.e. to create a workflow of rules

Working with query rules

For example:

```
- rule_id: 1,  flagOUT: 3, apply: 0
- rule_id: 5,  flagIN: 2,  apply: 1
- rule_id: 10, flagIN: 3,  apply: 1
```

If the criteria matches **rule_id: 1** , flagOUT=3 will be applied, then **rule_id: 10** will also be evaluated. If the criteria matches **rule_id: 5**, this rule will be applied and no further rules will be processed.

Mirroring Query Rule Criteria

Mirroring is defined using the following "mysql_query_rules" table columns:

- **mirror_flagOUT (similar to flagOUT)**
- **mirror_hostgroup**

If either are set for a matching query, real time query mirroring is automatically enabled.

Mirroring Query Rule Criteria

- If a query is rewritten, the rewrite applies to mirroring and the query can be rewritten again
- if **mirror_flagOUT** or **mirror_hostgroup** are set while processing the source query, a new mysql session is created
- The new mysql session will get all the same properties of the original mysql session : same **credentials**, **schemaname**, **default hostgroup**

mirror_hostgroup

If **mirror_hostgroup** was set in the original session, the new session will change its default **hostgroup** to **mirror_hostgroup**

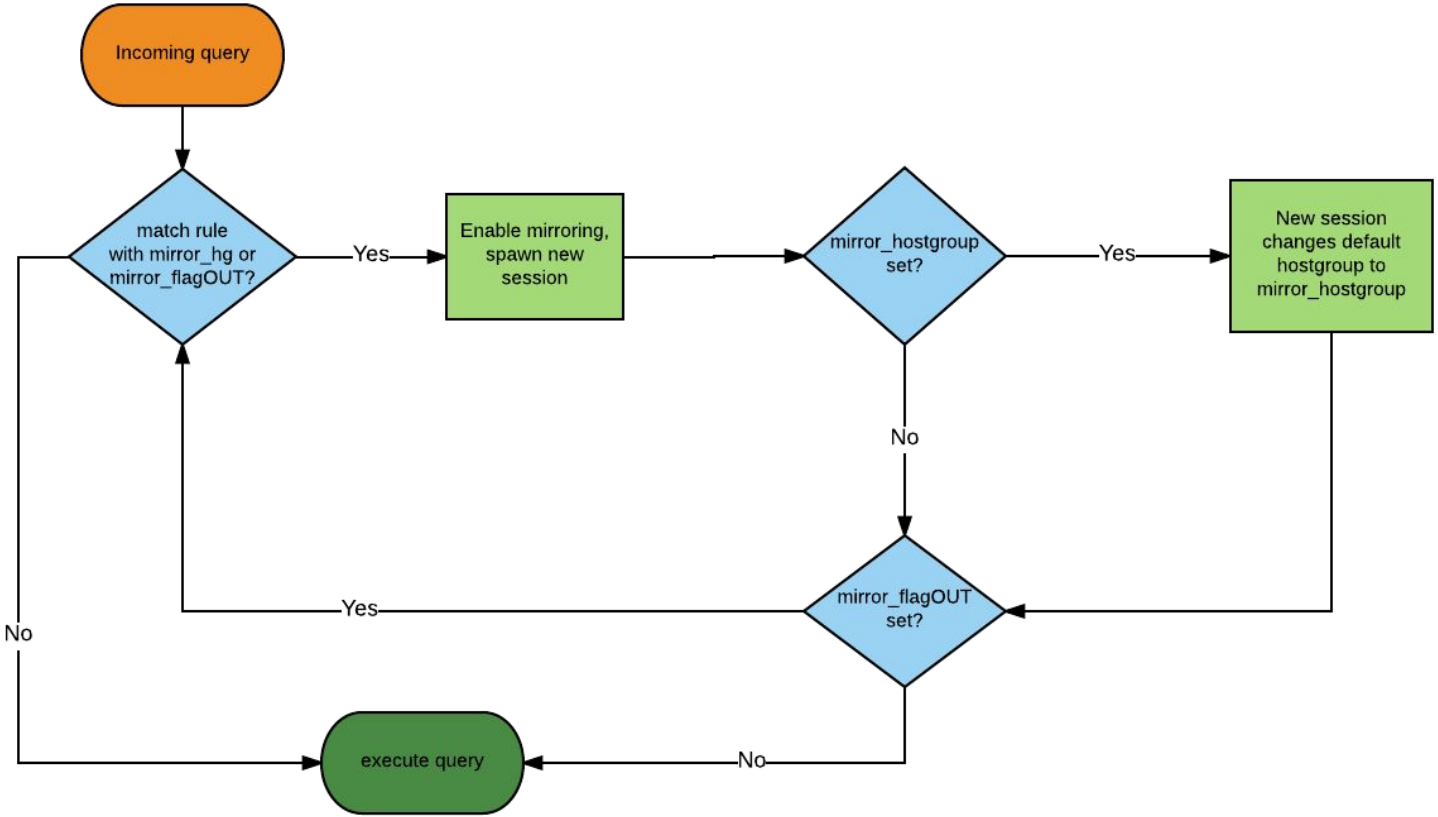
- The current query (either the original one or the rewritten one if a rewrite was performed) will be executed against a server in the define hostgroup

mirror_flagOUT

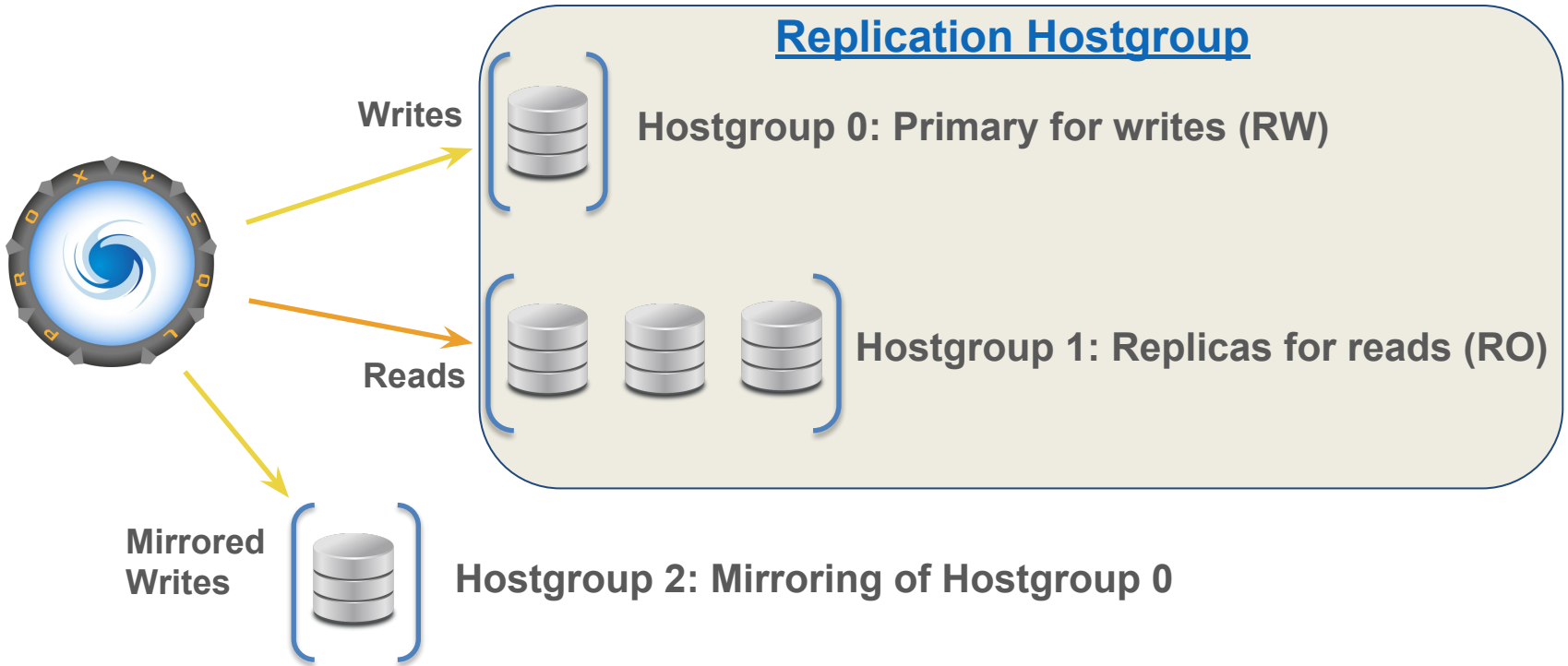
If **mirror_flagOUT** is set the new mysql session will try to match the query from the original session against **mysql_query_rules** starting from a value of **FlagIN=mirror_flagOUT**

- This allows to further modify the query, like rewriting it, or changing again the hostgroup

Mirroring flow



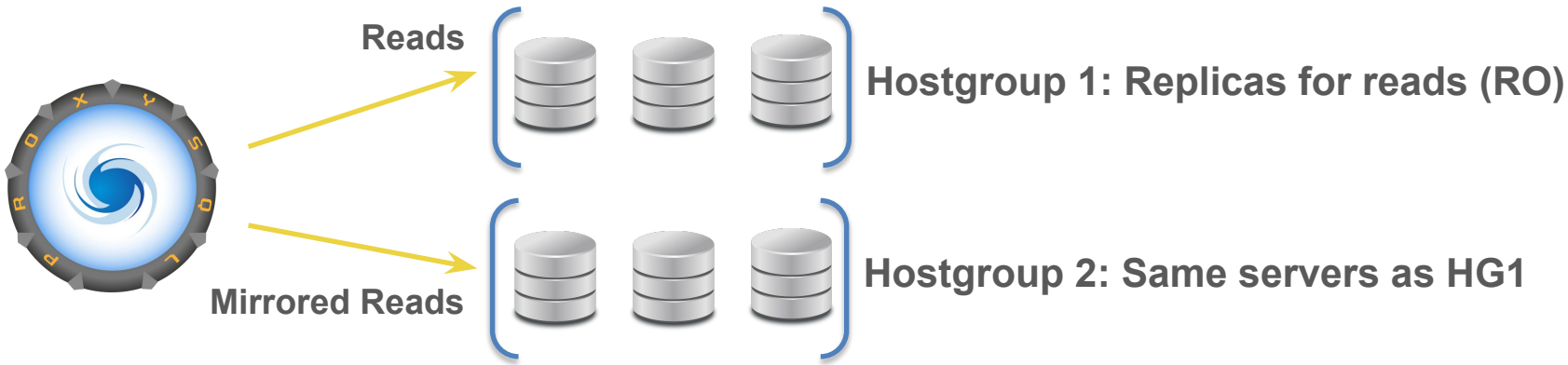
MySQL Query Rule Example



Mirroring Use Cases

Double Read Traffic on Backends

Mirroring read traffic against your backends (useful for capacity planning):

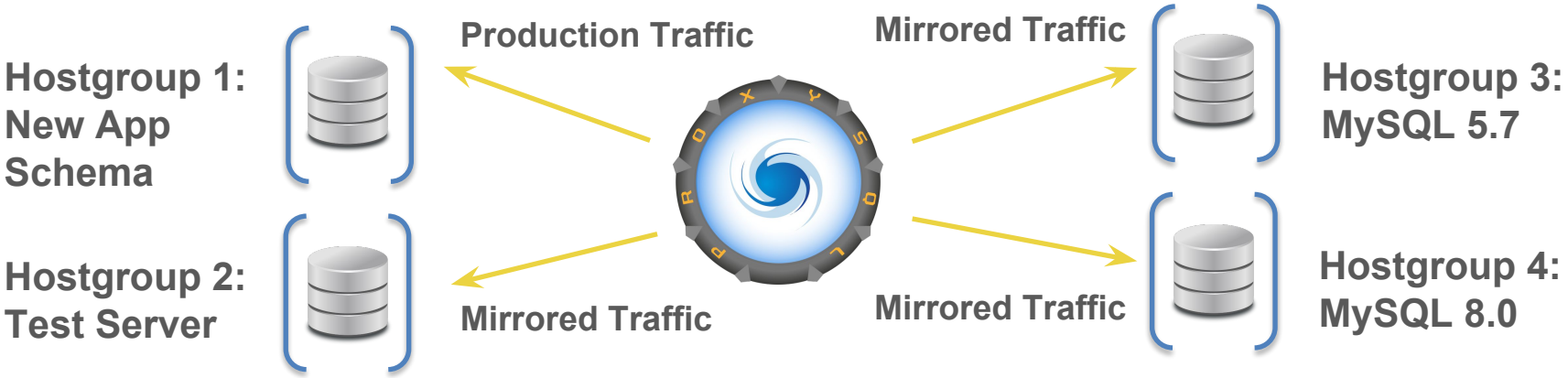


Mirror Reads from one HG to another

Useful to compare performance of two or more:

- different hardware setup
- different MySQL versions
- different MySQL configuration
- different schema design (different indexes)

Mirror Reads from one HG to another



Validate different version of MySQL

- Compare performance of two or more different MySQL versions
- Identify new keywords
- Identify any other error, for example related to sql_mode

Testing Query Rewrite

Before rewriting the original query:

- Create a mirror rule
- Rewrite the query in the mirror session
- Send the query to the same backends using the same or different HGs
- Check for errors

This use is meant to check the correctness (no syntax error) of query rewrite

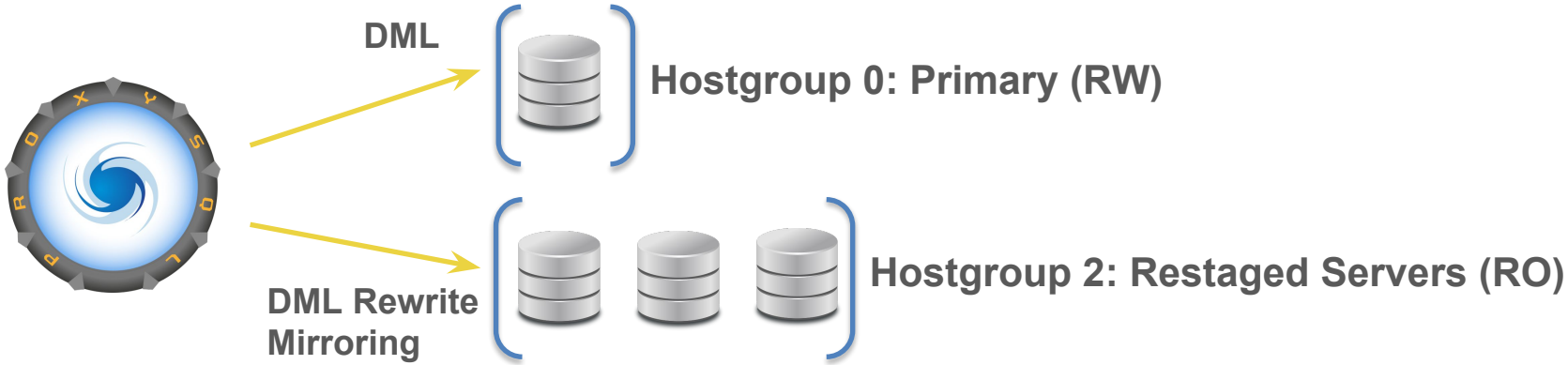
Query Rewrite and Performance

The rewritten query can have different performance.

- For example, adding FORCE INDEX
- Statistics are collected in real time, making possible to compare performance.

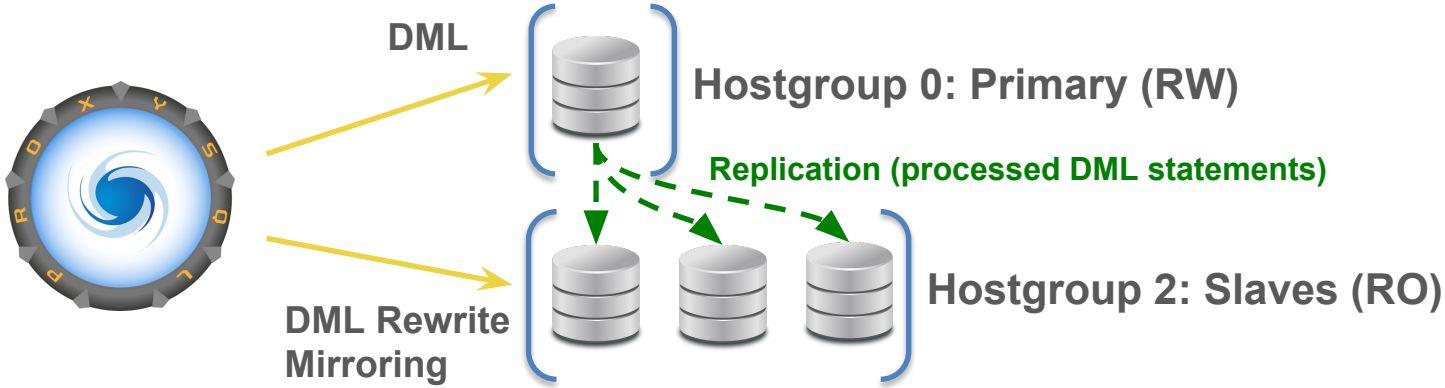
Warming up InnoDB's Buffer Pool

- Mirror read traffic (all, or part of it) from one HG to another HG
- Useful after a server was recently started and its buffer pool is cold



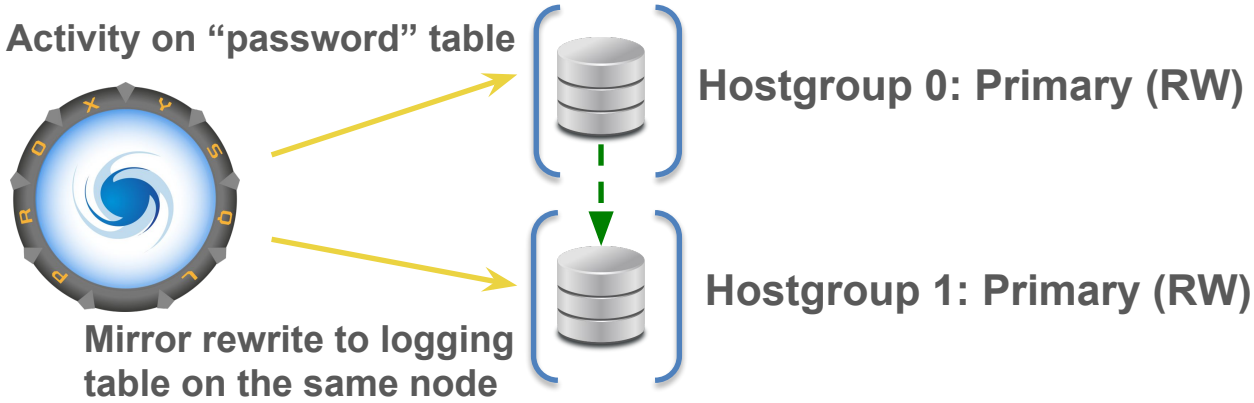
Replication Booster

- Create new mirror session for DML statements
- Rewrite the mirrored DML statements into SELECT statements
- Send the SELECT statements to replicas

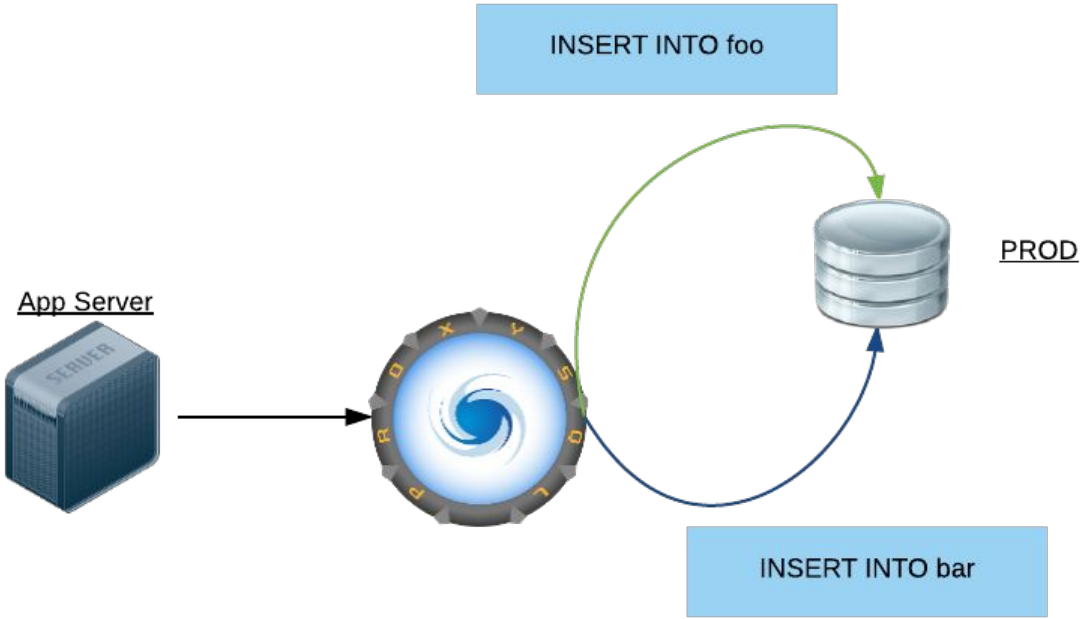


Auditing

- Create query rules for specific tables which rewrite statements to inserts
- Record activities into a logging table

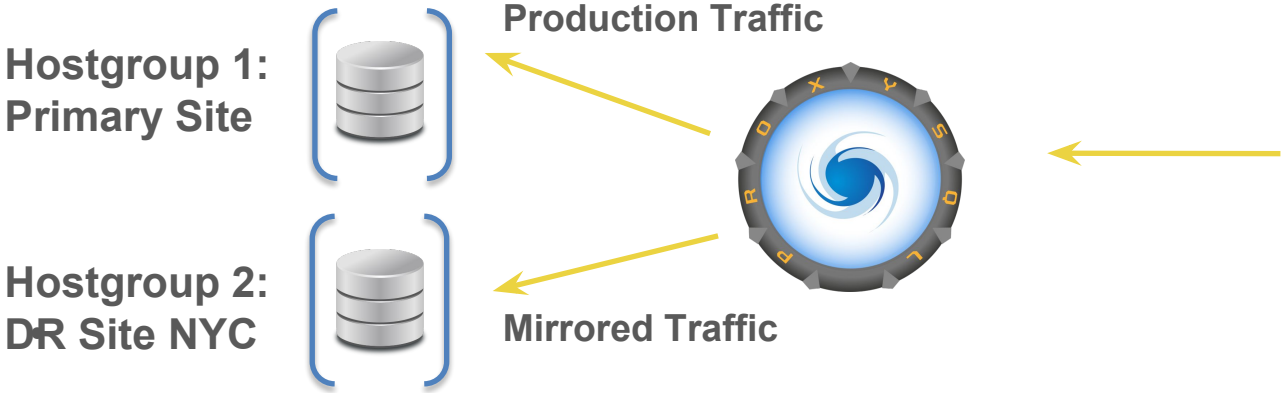


Auditing



Write Buffering

- Useful on WAN setup
- One local client performing a lot of INSERT with low latency
- Mirrored traffic will be queued
- Executed in parallel on a backend with big latency



Write Buffering

- Useful on WAN setup
- One local client performing a lot of INSERT with low latency
- Mirrored traffic will be queued
- Executed in parallel on a backend with big latency

Questions?