MySQL Best practices for Replication and High Availability

Upper New York Oracle User Group

Presenter: Joseph Layous

Date: November 13, 2015

Speaker Introduction – Joseph Layous

- Working with MySQL since 2004
- Senior Consultant, Rolta AdvizeX
- Areas of Expertise
  - MySQL
  - MS SQL Server
  - Application Development

- Email Addresses:
  - JLAYOUS@ADVIZEX.COM
  - JOSEPH.LAYOUS@ROLTA.COM

Who's Using MySQL

Components of Replication

- **Master**
  - Changes data
  - Logs changes (events) into a file (the binary log)
- **Slave**
  - Retrieves events from the master
  - Replays the events on the slaves databases
- **Binary Log**
  - Records every change on the master (in either format: row or statement)
  - Split into transactional groups

The Path of Replication

- Asynchronous Replication
  - Transactions committed immediately
  - Events are propagated after the commit operation is acknowledged
  - Faster but vulnerable to lost updates on server crashes and inconsistency
  - Built into the server
- Semi-synchronous Replication
  - Master commits transaction but waits for N slaves to acknowledge having received and stored the correspondent event before replying to the client

Replication Commands

- **change master to...**
  Which server? Port? Credentials? Log file & position?
- **change master to**

  master_host='Master', master_port=3306,

  master_user='repl', master_password='pass',

  master_log_file='mysql_bin.000001',

  master_log_pos=4563744;

Basic Setup Walk-Through

- Replication User

On Master:

```
grant replication slave on *.* to ‘repl’
identified by ‘pass’;
```

ROLTA ADVIZEX

Basic Setup Walk-Through

- Point to the correct binlog position
- On master:

```
show master status;
```

...once both databases are equal

- On Replica:

```
change master to ...
```

Basic Setup Walk-Through

- Start it up
- On replica:

```
start slave;
```

- Check it
- On replica:

```
show slave status\G
```

Troubleshooting Replication

- show slave status\G
- Manual troubleshooting
- Use in scripts for alarming

```
LAG=$(mysql -s -e'show slave status\G'|grep
  'Seconds_Behind_Master'|cut -f2 -d:| tr -d ' ')

IO_UP=$(mysql -s -e'show slave status\G'|grep 'Slave_IO_Running'|cut
  -f2 -d:| tr -d ' ')
```

Replication Fault: Lag

- Long-running transactions
- High concurrency on master becomes single session on replica
- **`show slave status\G`**
- Easy to alarm on

Replication Fault: Stopping from Error

- *Any* error stops SQL Thread!
- Duplicate Key
- Unknown Function
- **`slave_exec_mode=idempotent`**
  ...suppresses duplicate-key and no-key-found errors
- **`show slave status\G`**

ROLTA ADVIZEX

Replication Faults: Can’t Connect to Master

- Fault shows in slave status as soon as you **`start slave`**.
- Did you create replication slave user?
- Correct user/password?
- Correct hostname/port?
- Master is running and reachable?

Replication Fault: Can’t Find Binlog

- Clearly stated in the slave status
- Can happen if replica has been stopped (or is lagged) and the binlogs are removed from the master
- Execute from archive

OR

- Rebuild replica and restart from a known good binlog position

Replication Faults: Configuration

- replication-do-table
- replication-do-schema
- log_slave_updates

Replication Faults: server_id

- Must be unique!
- If equal: “Seconds behind” jumps everywhere.

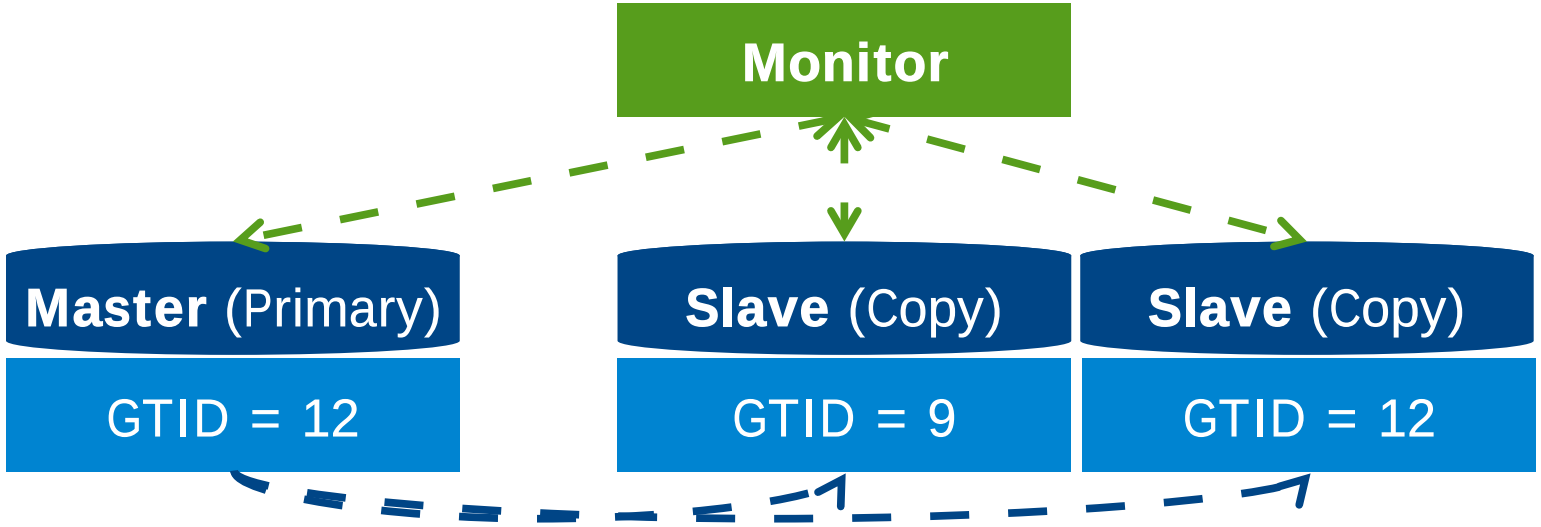
Now that we have replication setup

- Evaluate whether a simple Master → Slave setup is suitable for your business requirements
- Evaluate Recovery Time Objective
  - Maximum length of downtime before there is break in “business continuity”
- Evaluate Recovery Point Objective
  - Point in time to which data must be recovered when service is re-established

- **Most Importantly, Evaluate Single Point of Failure**

Things To Be Wary Of

- Master database process and/or master host monitoring
- How to identify a failover candidate
- How not to loose transactions, ever



Things To Consider Regarding HA

- SLA requirements to support business objectives
- Operational capabilities
- Service agility & time to market
- Budgetary constraints

Why Use High Availability MySQL

- Servers can crash (hardware, software or even power failure).
- Services should not.
- Dual masters, circular replication (conflict free partition workload on each).
- Seamless fail-over of affected partitions
- Scaling out with slaves.
- Ready to step up and replace a failed master (on dual masters watch out whether the slave is already ahead of second master – slave promotion instead)
- Active / Passive – binlog is shared by the two master's, on fail-over binlog positions match

Business Cases for HA MySQL

- High Availability: fail over
  - Servers can crash (hardware, software or even power failure).
  - Services should not.
- Dual masters, circular replication (conflict free partition workload on each).
  - Seamless fail-over of affected partitions
  - Scaling out with slaves.
    - Ready to step up and replace a failed master (on dual masters watch out whether the slave is already ahead of second master – slave promotion instead)
- Active / Passive – binlog is shared by the two master's, on fail-over binlog positions match

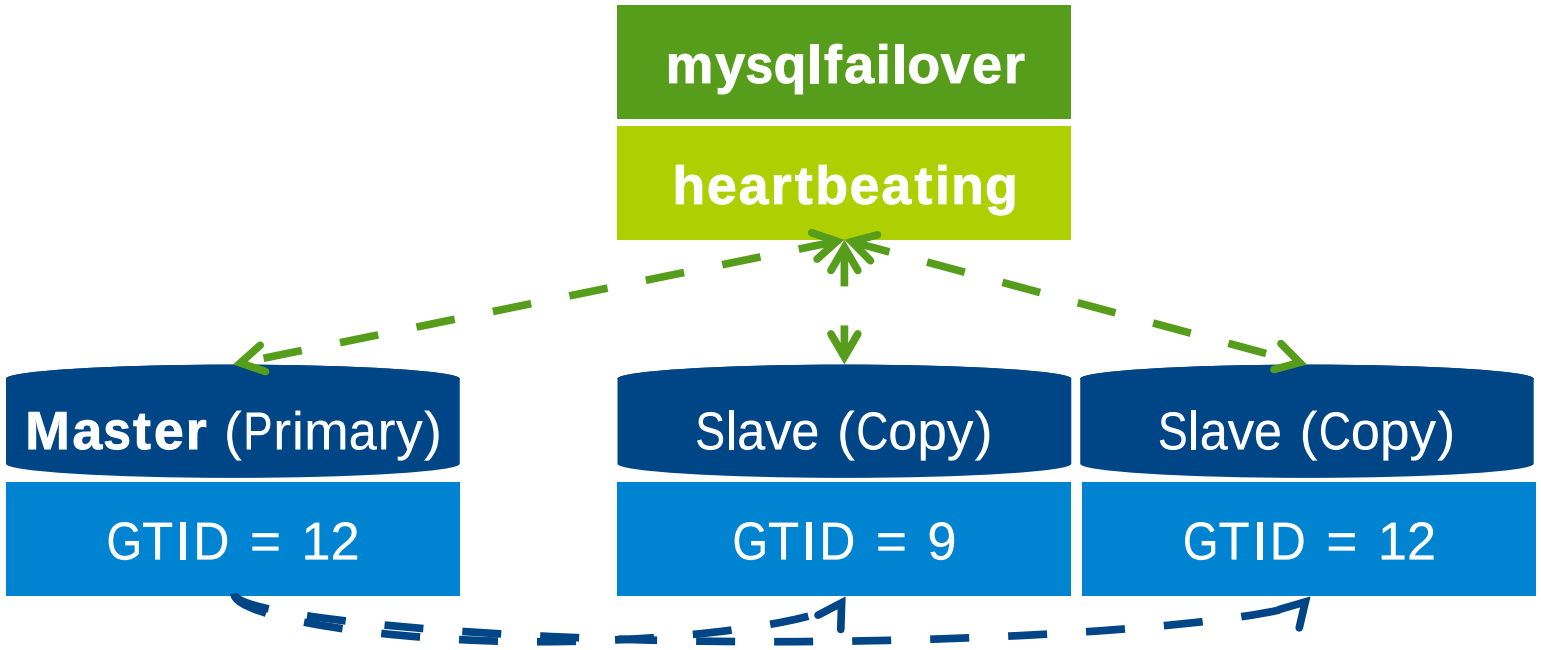
HA Load Balancing

- Requires monitoring tools (status and performance)
  - Servers crash every now and then!
- Requires intelligent query routing (reads vs writes)
  - Which queries go where?
- Session consistency is something to keep in mind!
- Enter MySQL Proxy, PHP mysqlnd, and..

	MySQL Replication	MySQL Fabric	Oracle VM Template	Oracle Clusterware	Solaris Cluster	Windows Cluster	DRBD	MySQL Cluster
App Auto-Failover	✗	✓	✓	✓	✓	✓	✓	✓
Data Layer Auto-Failover	✗	✓	✓	✓	✓	✓	✓	✓
Zero Data Loss	MySQL 5.7	MySQL 5.7	✓	✓	✓	✓	✓	✓
Platform Support	All	All	Linux	Linux	Solaris	Windows	Linux	All
Clustering Mode	Master + Slaves	Master + Slaves	Active/ Passive	Active/Passive	Active /Passive	Active/Passive	Active/ Passive	Multi-Master
Failover Time	N/A	Secs	Secs +	Secs +	Secs +	Secs +	Secs +	< 1 Sec
Scale-out	Reads	✓	✗	✗	✗	✗	✗	✓
Cross-shard operations	N/A	✗	N/A	N/A	N/A	N/A	N/A	✓
Transparent routing	✗	For HA	✓	✓	✓	✓	✓	✓
Shared Nothing	✓	✓	✗	✗	✗	✗	✓	✓
Single Vendor Support	✓	✓	✓	✓	✓	✗	✓	✓

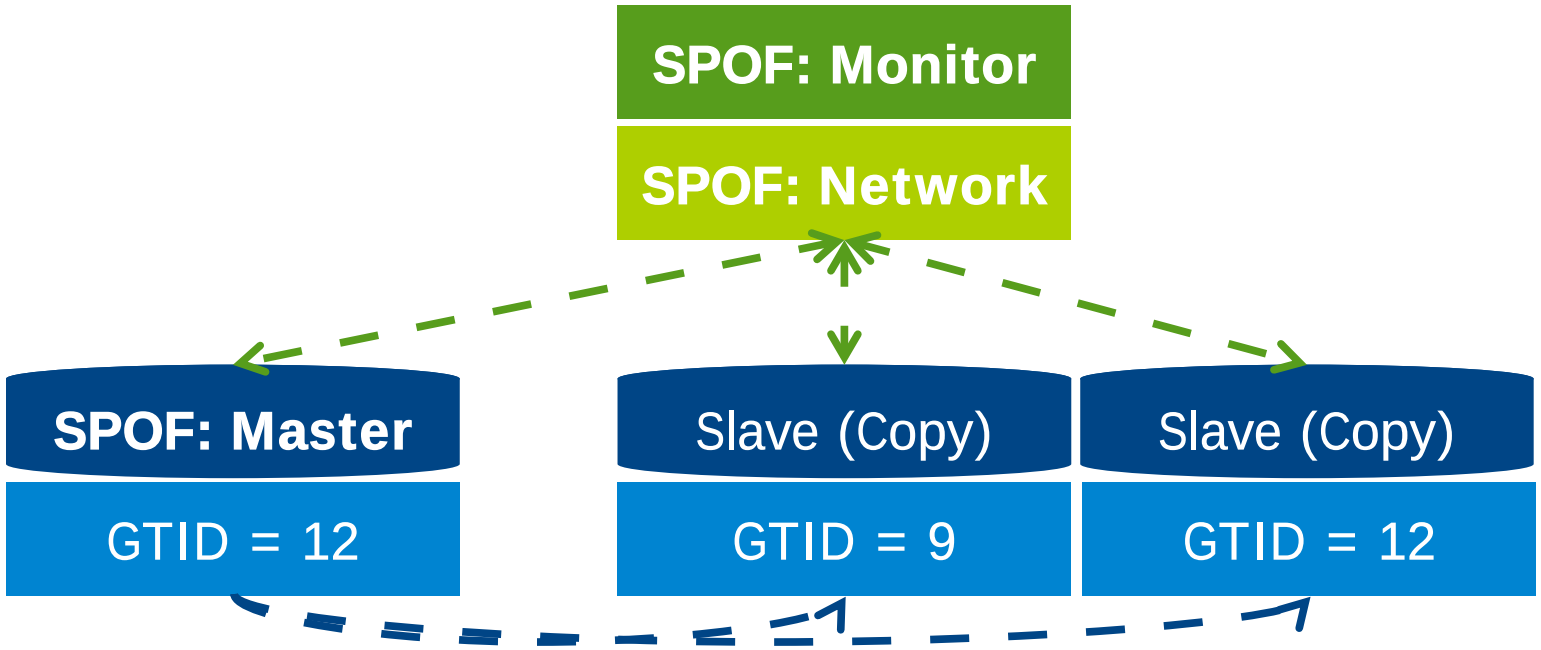
Your new best friend, the mysqlfailover Utility

- Introduced as MySQL 5.6 Utility
  - Health monitoring, Failover
  - Aims for 99.9% HA – 8 hours downtime per year



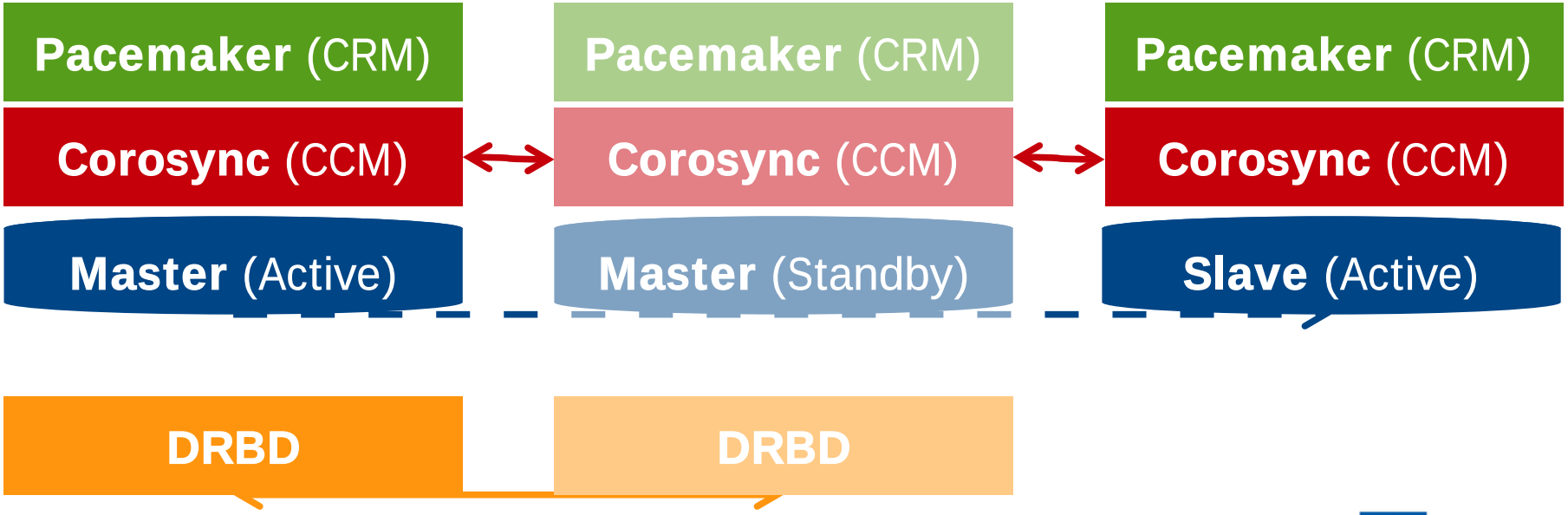
The results of setting this up?

- We effectively doubled the Single Point of Failure
- Common design because of its simplicity



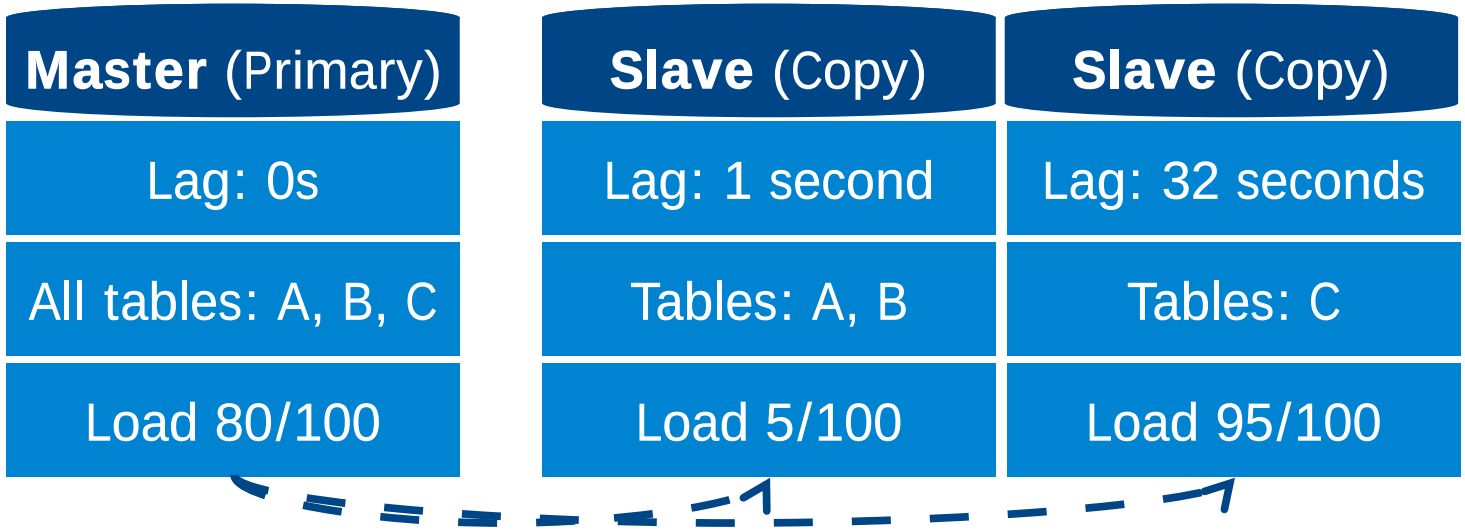
Standard HA Solutions

- Developed and pushed by major Linux vendors
- Pacemaker, Corosync/Heartbeat, DRBD
- Aims for 99.99% HA – 50 minutes downtime/year



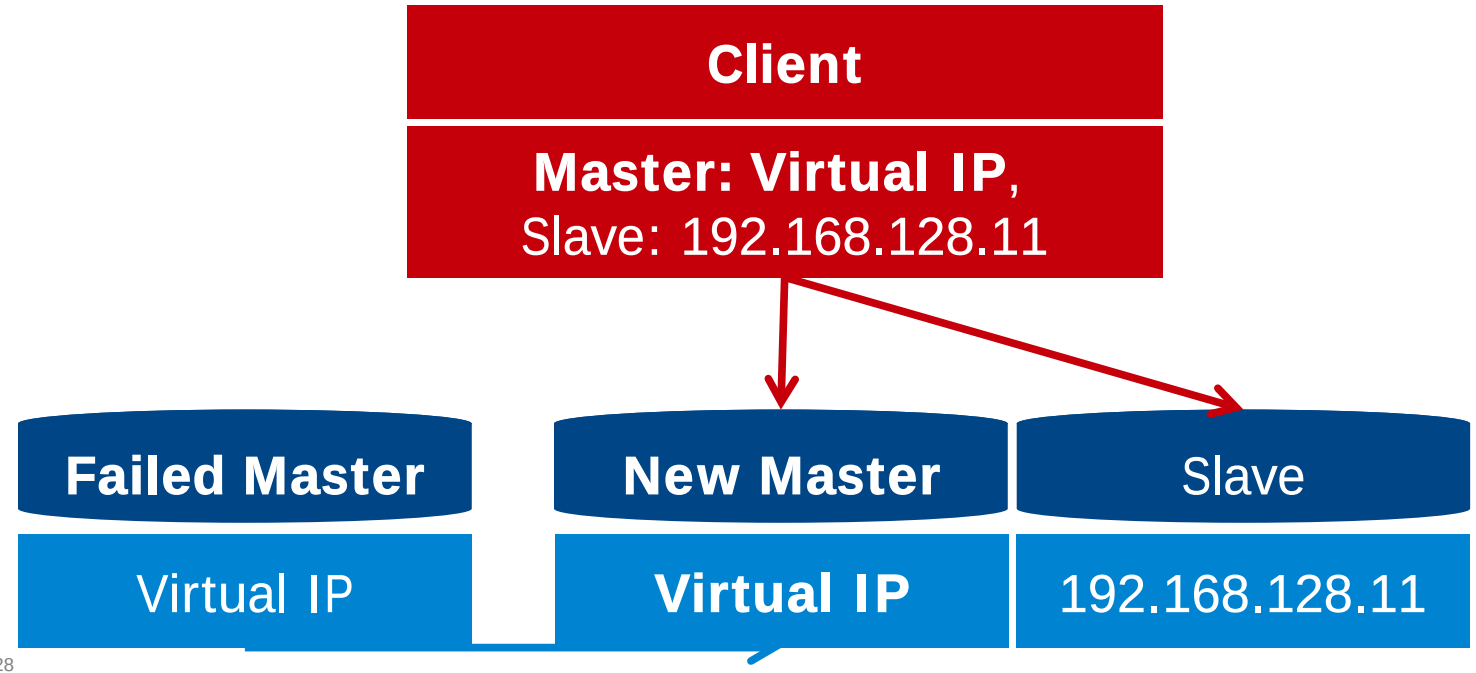
Issues with that setup

- Available servers, their roles, and possibly replication lag
- Partitioning and sharding hints, if needed
- Real-time server load



The Server / Client Collision

- Typical failover procedure
  - Master failure: switch virtual IP, no client deployment
  - Slave failure: deploy all clients
  - Role, status, lag, load, distribution, … : hacks, at best

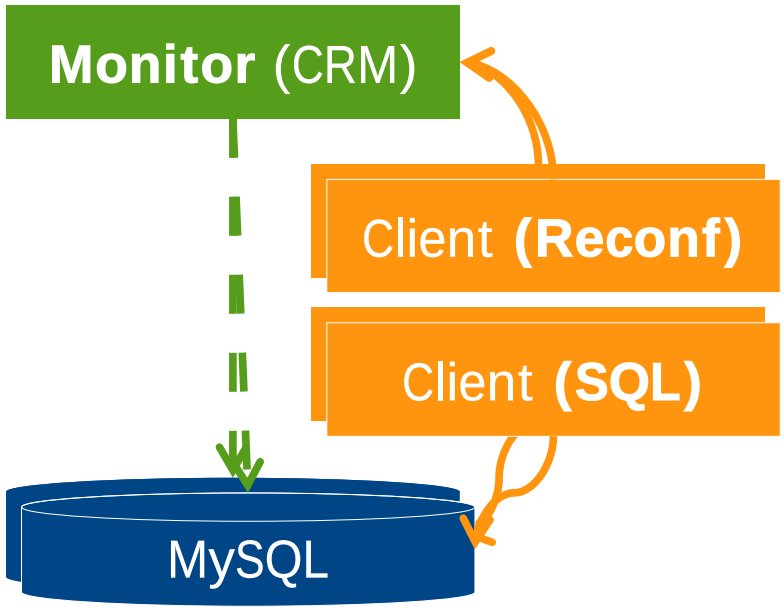


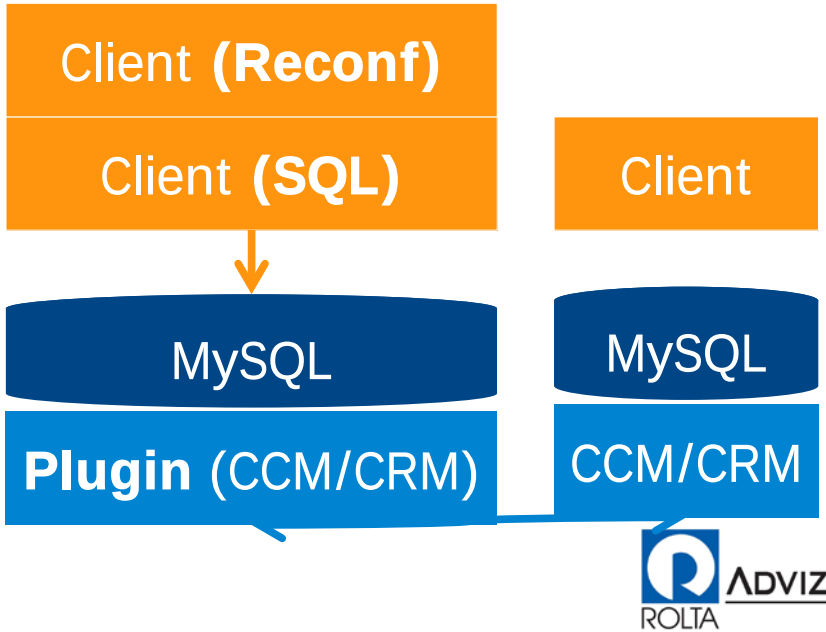
Compared to mysqlfailover other solutions offer

- Similar no SPOF design for the monitor
- Aims for: out-of-the-box experience, smaller installations
- Aims for: continuous, automatic client reconfiguration

Compared To The Larger Solutions

- No SPOF design for the monitor
- No central server that can get overloaded
- No new communication channels for client reconfiguration





Thank You !!!

Questions
&
Answers ??