

ORACLE®

MySQL Group Replication in a nutshell

MySQL InnoDB Cluster: hands-on tutorial

Percona Live Amsterdam - October 2016

Frédéric Descamps - MySQL Community Manager - Oracle

Kenny Gryp - MySQL Practice Manager - Percona

Safe Harbor Statement

The following is intended to outline our general product direction. It is intended for information purpose only, and may not be incorporated into any contract. It is not a commitment to deliver any material, code, or functionality, and should not be relied upon in making purchasing decisions. The development, release and timing of any features or functionality described for Oracle's product remains at the sole discretion of Oracle.

about.me/lefred

Who are we ?





Frédéric Descamps

- @lefred
- MySQL Evangelist
- Managing MySQL since 3.23
- devops believer



Kenny Gryp

- @gryp
- MySQL Practice Manager



get more at the conference

MySQL Group Replication



Copyright @ 2016 Oracle and/or its affiliates. All rights reserved.



Other session

- MySQL Replication: Latest Developments
 - *Luis Soares*
 - Tuesday 4 October 2016
 - 3:10pm to 4:00pm - Zürich 1



Agenda

- Prepare your workstation



Agenda

- Prepare your workstation
- Group Replication concepts



Agenda

- Prepare your workstation
- Group Replication concepts
- Migration from Master-Slave to GR



Agenda

- Prepare your workstation
- Group Replication concepts
- Migration from Master-Slave to GR
- How to monitor ?



Agenda

- Prepare your workstation
- Group Replication concepts
- Migration from Master-Slave to GR
- How to monitor ?
- Application interaction



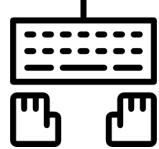
VirtualBox

Setup your workstation



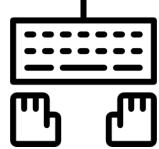
Copyright @ 2016 Oracle and/or its affiliates. All rights reserved.





Setup your workstation

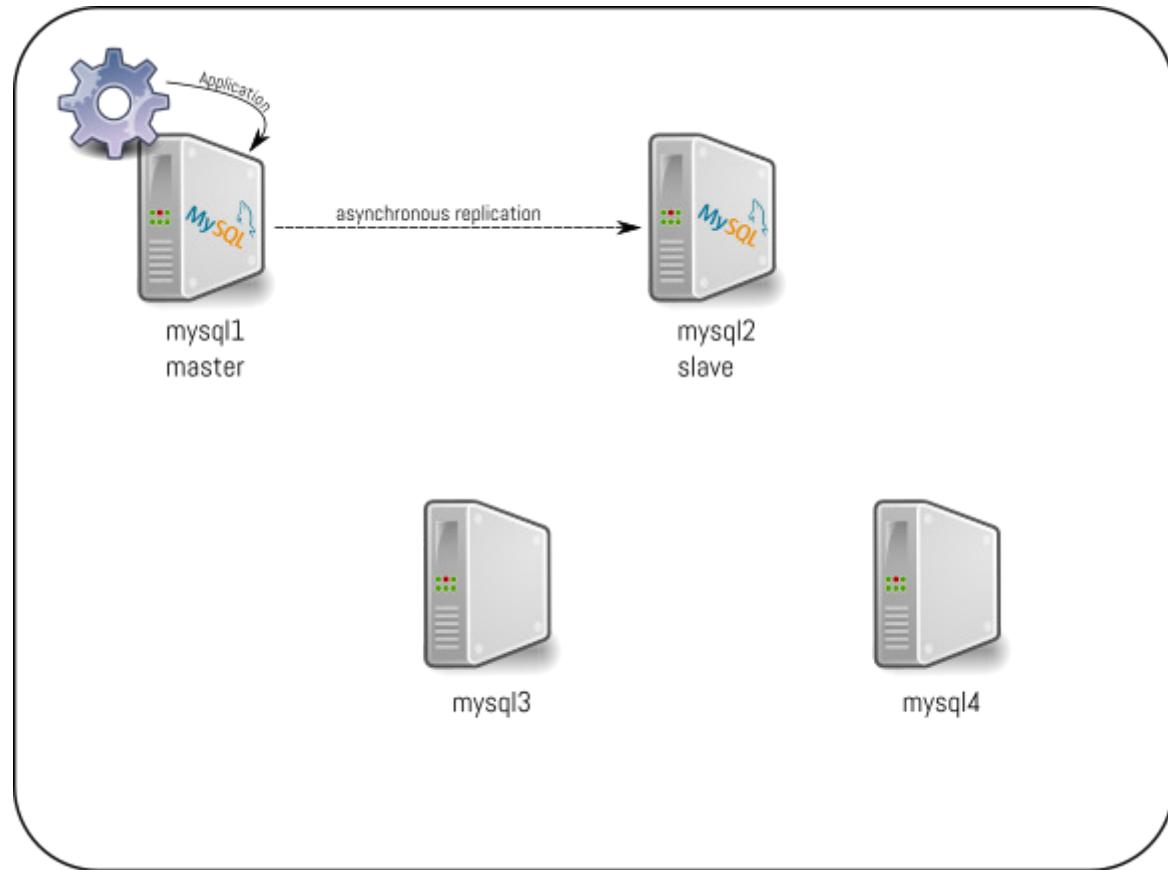
- Install VirtualBox 5
- On the USB key, there is a file called PLAM16_GR.ova, please copy it on your laptop and doubleclick on it
- Start all virtual machines (mysql1, mysql2, mysql3 & mysql4)
- Install putty if you are using Windows

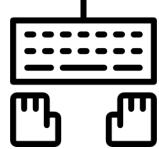


Setup your workstation

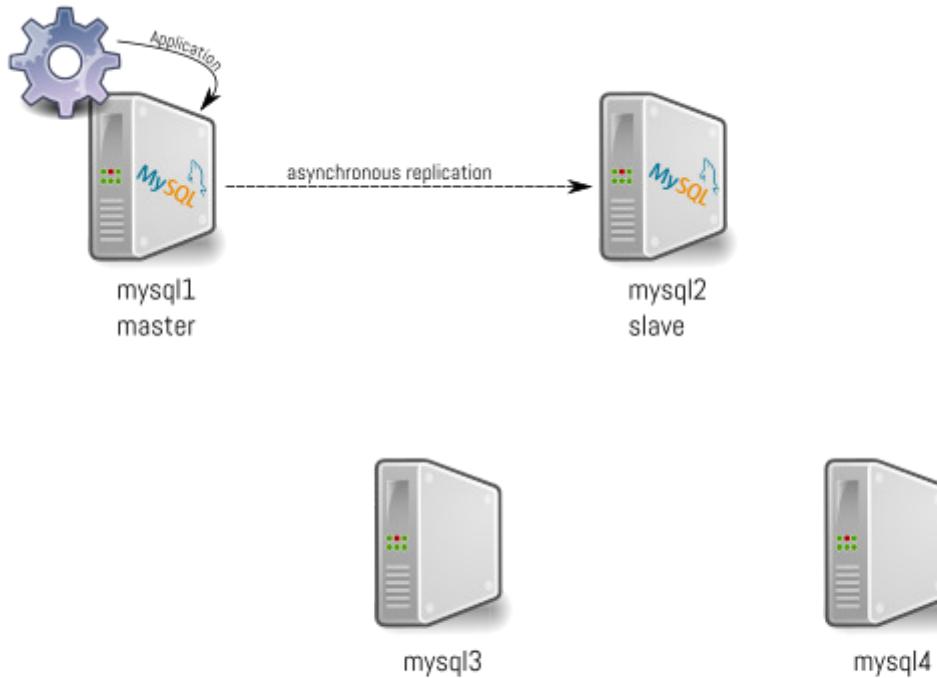
- Install VirtualBox 5
- On the USB key, there is a file called PLAM16_GR.ova, please copy it on your laptop and doubleclick on it
- Start all virtual machines (mysql1, mysql2, mysql3 & mysql4)
- Install putty if you are using Windows
- Try to connect to all VM's from your terminal or putty (*root password is X*):
 - ssh -p 8821 root@127.0.0.1 to mysql1
 - ssh -p 8822 root@127.0.0.1 to mysql2
 - ssh -p 8823 root@127.0.0.1 to mysql3
 - ssh -p 8824 root@127.0.0.1 to mysql4

LAB1: Current situation





LAB1: Current situation



- launch `run_app.sh` on `mysql1` into a `screen` session
- verify that `mysql2` is a running slave

Summary

	ROLE	SSH PORT	INTERNAL IP
mysql1	master	8821	192.168.56.11
mysql2	slave	8822	192.168.56.12
mysql3	n/a	8823	192.168.56.13
mysql4	n/a	8824	192.168.56.14

the magic explained

Group Replication Concept



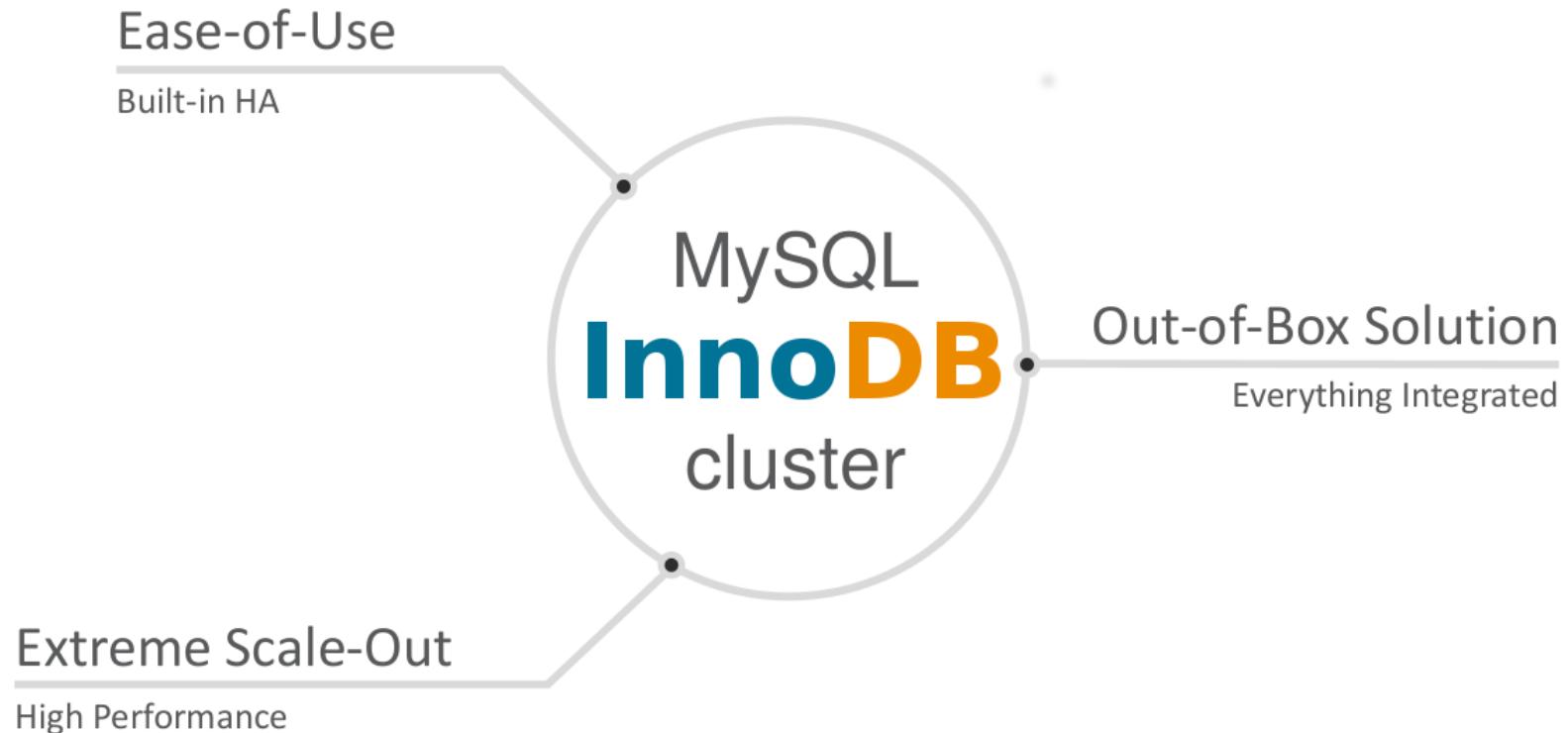
Group Replication : what is it ?

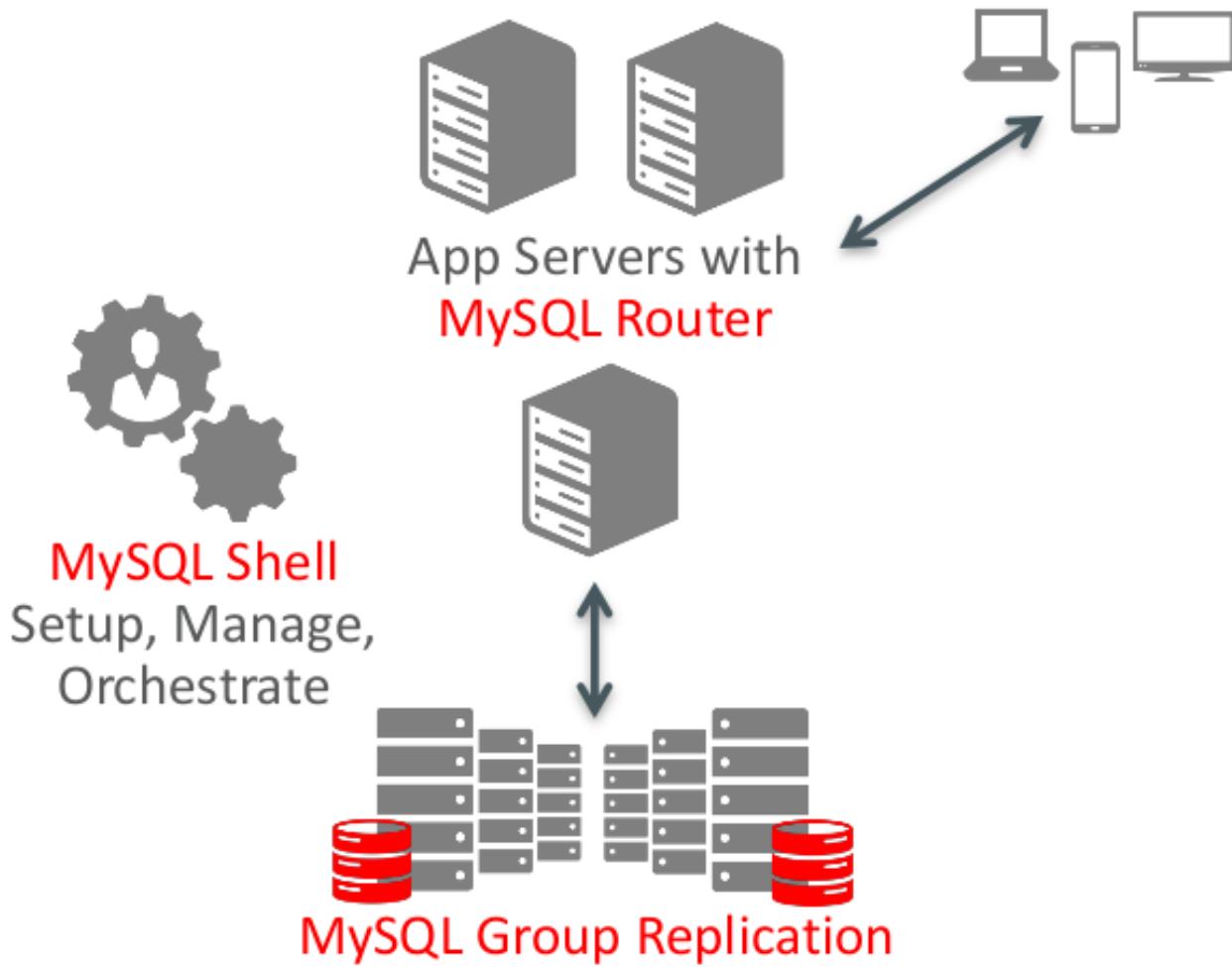
Group Replication : what is it ?

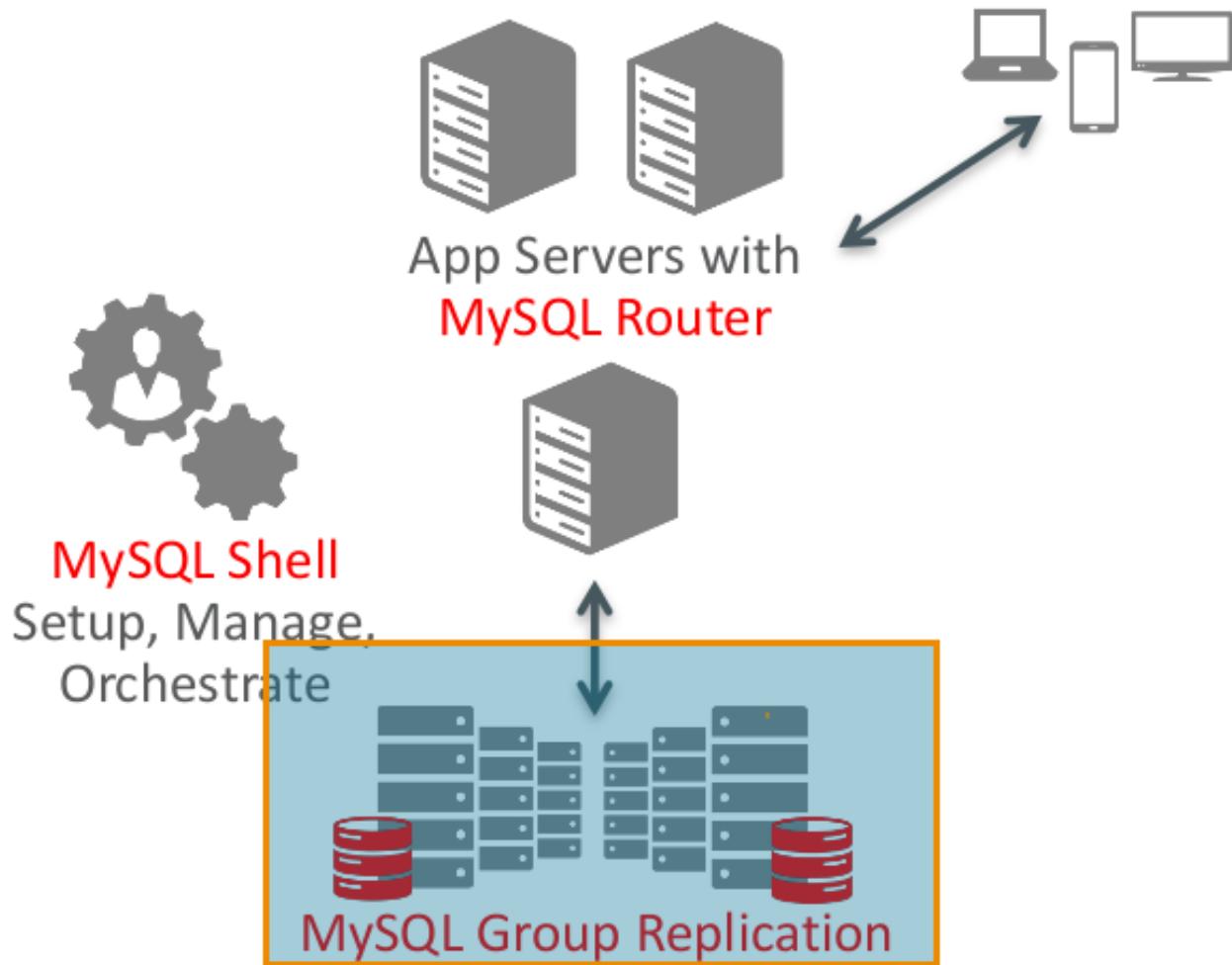
MySQL Group Replication is one of the major components of MySQL InnoDB Cluster

Group Replication : what is it ?

MySQL Group Replication is one of the major components of MySQL InnoDB Cluster







Group replication is a plugin !

Group replication is a plugin !

- GR is a plugin for MySQL, made by MySQL and packaged with MySQL

Group replication is a plugin !

- GR is a plugin for MySQL, made by MySQL and packaged with MySQL
- GR is an implementation of Replicated Database State Machine theory
 - MySQL Group Communication System (GCS) is based on Paxos Mencius

Group replication is a plugin !

- GR is a plugin for MySQL, made by MySQL and packaged with MySQL
- GR is an implementation of Replicated Database State Machine theory
 - MySQL Group Communication System (GCS) is based on Paxos Mencius
- Provides virtually synchronous replication for MySQL 5.7+

Group replication is a plugin !

- GR is a plugin for MySQL, made by MySQL and packaged with MySQL
- GR is an implementation of Replicated Database State Machine theory
 - MySQL Group Communication System (GCS) is based on Paxos Mencius
- Provides virtually synchronous replication for MySQL 5.7+
- Supported on all MySQL platforms !!
 - Linux, Windows, Solaris, OSX, FreeBSD

“Multi-master update anywhere replication plugin for MySQL with built-in conflict detection and resolution, automatic distributed recovery, and group membership.”

Group Replication : how does it work ?

Group Replication : how does it work ?

- A node (member of the group) sends the changes (binlog events) made by a transaction to the group through the GCS.

Group Replication : how does it work ?

- A node (member of the group) sends the changes (binlog events) made by a transaction to the group through the GCS.
- All members consume the writeset and certify it, no need to wait for all members, ack by the majority is enough.

Group Replication : how does it work ?

- A node (member of the group) sends the changes (binlog events) made by a transaction to the group through the GCS.
- All members consume the writeset and certify it, no need to wait for all members, ack by the majority is enough.
- If passed it is applied, if failed, transaction is rolled back on originating server and discarded at other servers.

OK... but how does it work ?!

OK... but how does it work ?!

It's just magic !

OK... but how does it work ?!

It's just magic !

... no, the writeset replication is **synchronous** and then certification and applying of the changes happen locally on each node and is asynchronous.

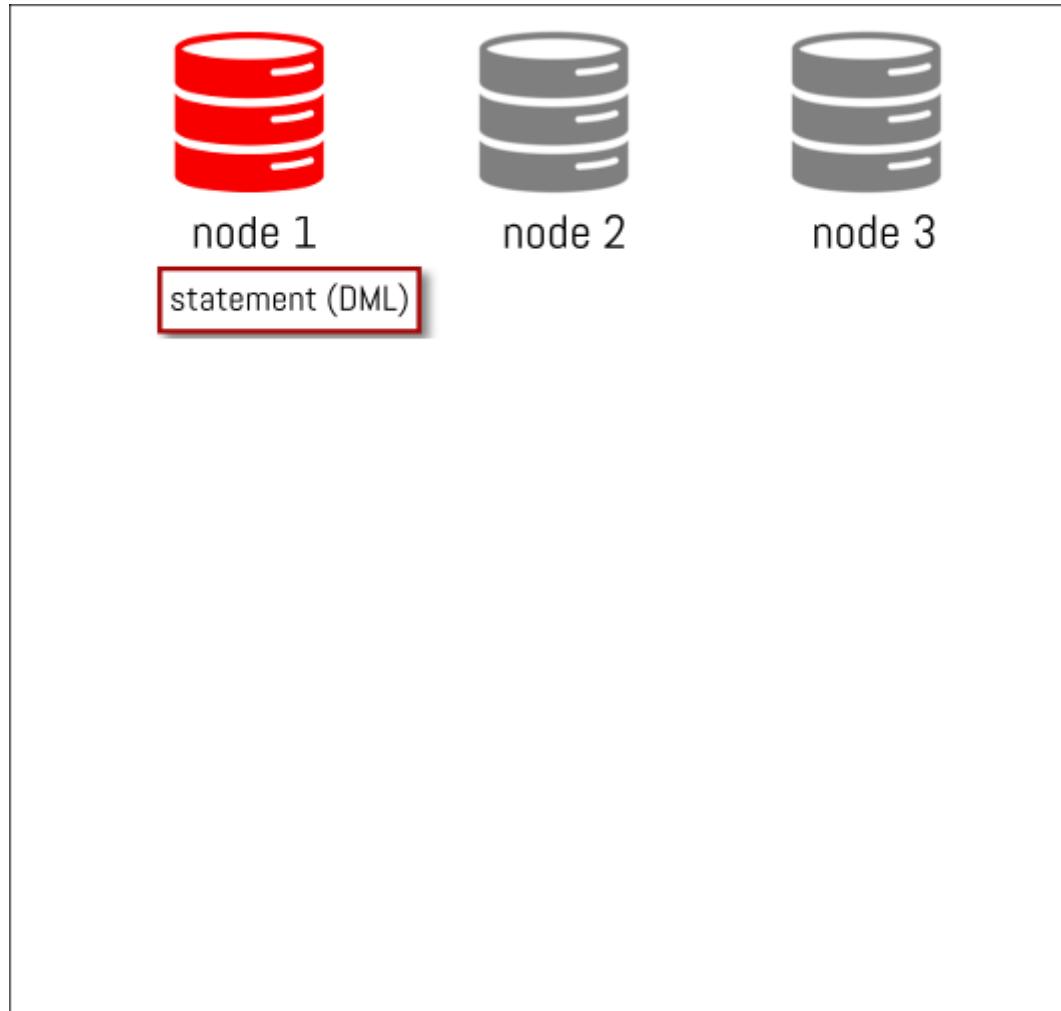
OK... but how does it work ?!

It's just magic !

... no, the writeset replication is **synchronous** and then certification and applying of the changes happen locally on each node and is asynchronous.

not that easy to understand... right ? Let's illustrate this...

MySQL Group Replication (autocommit)

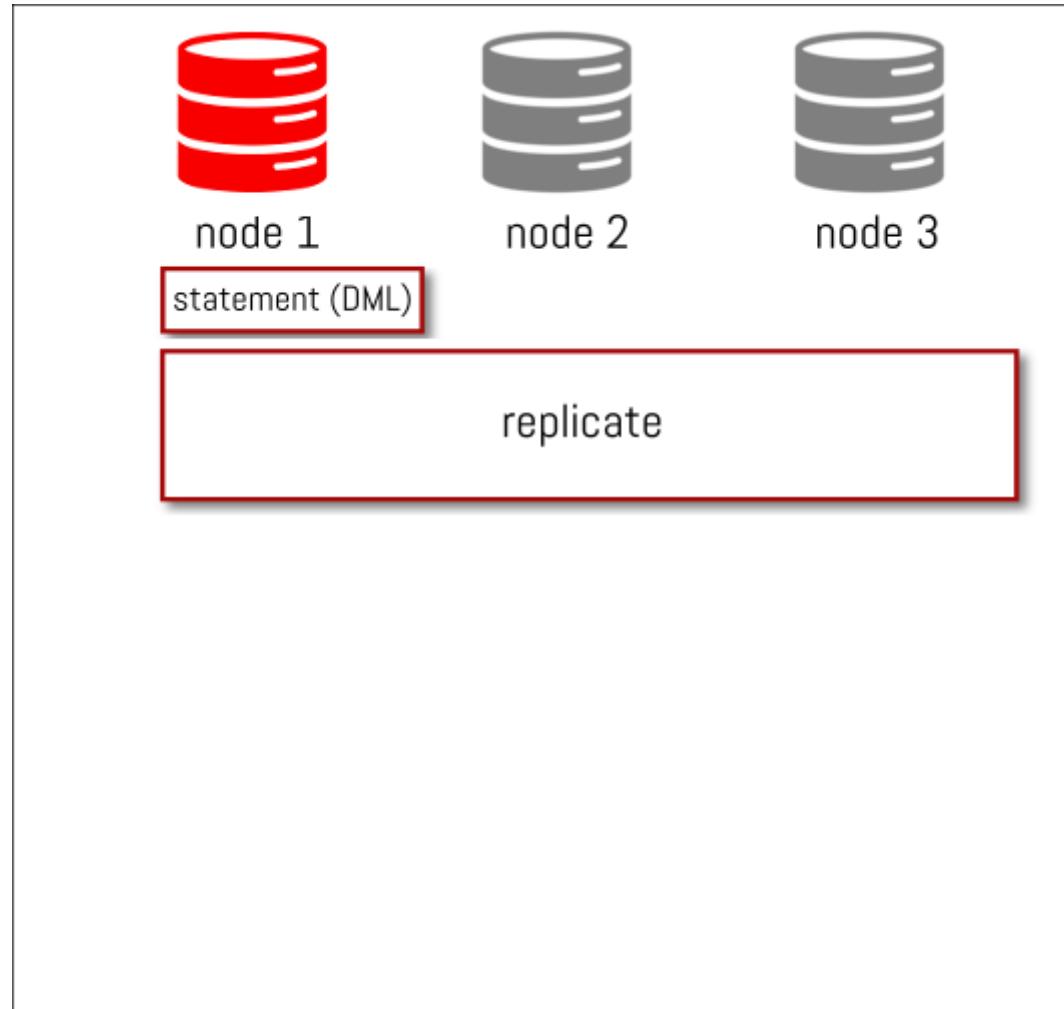


ORACLE®

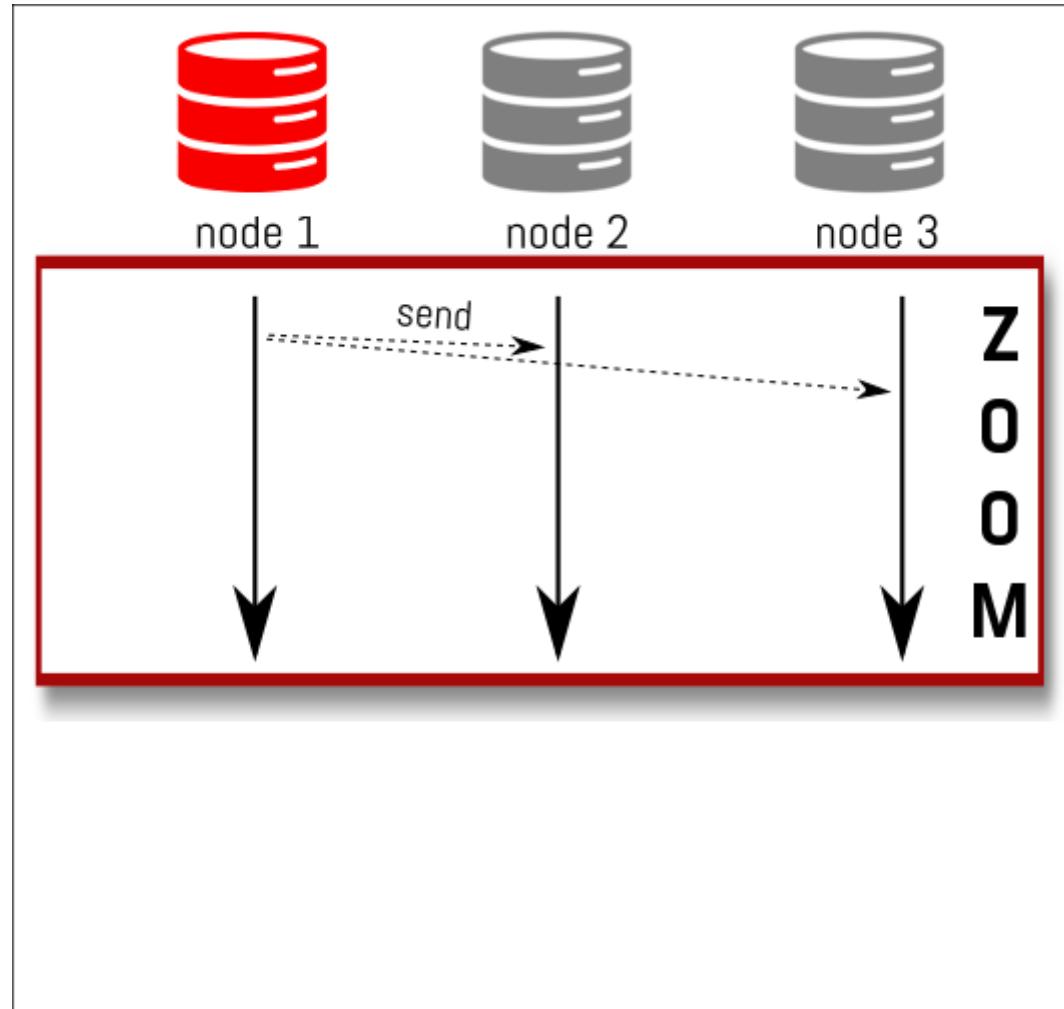
Copyright @ 2016 Oracle and/or its affiliates. All rights reserved.



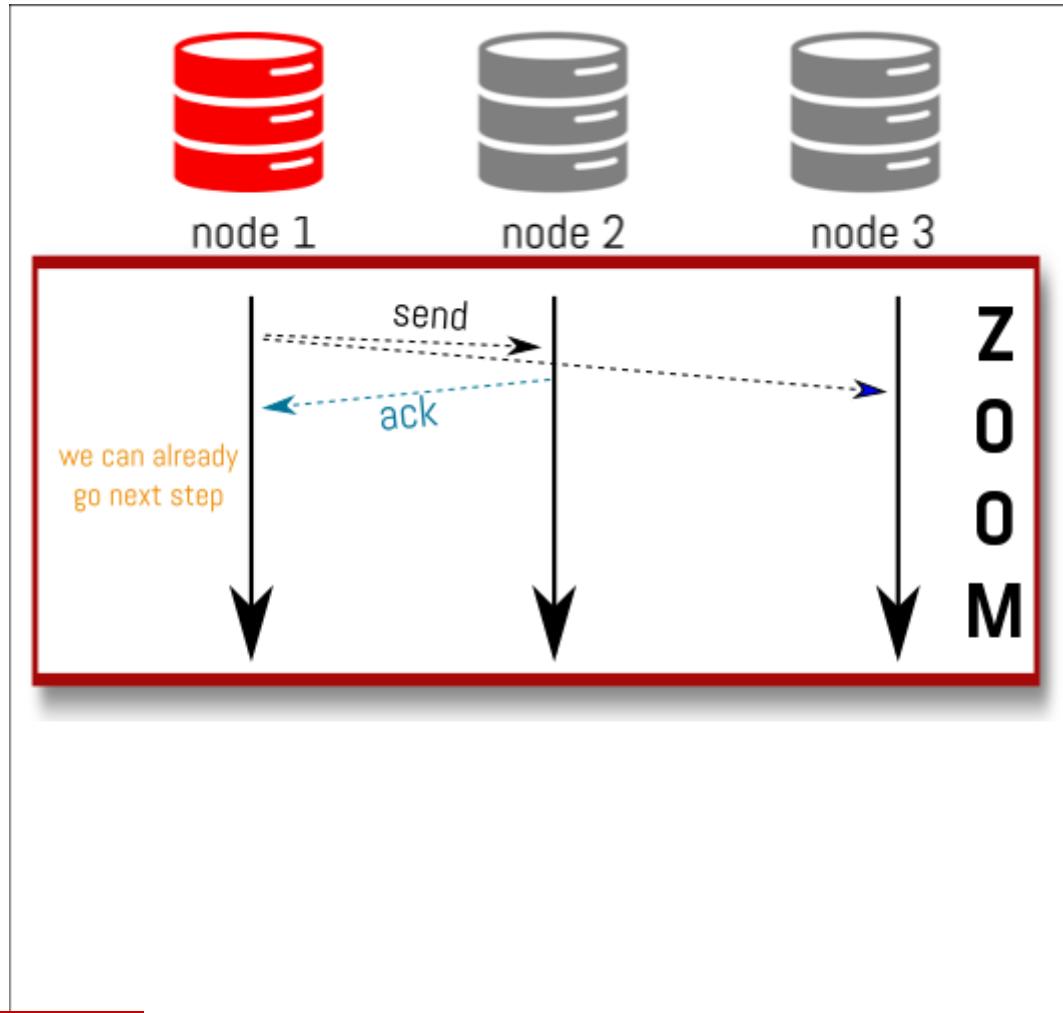
MySQL Group Replication (autocommit)



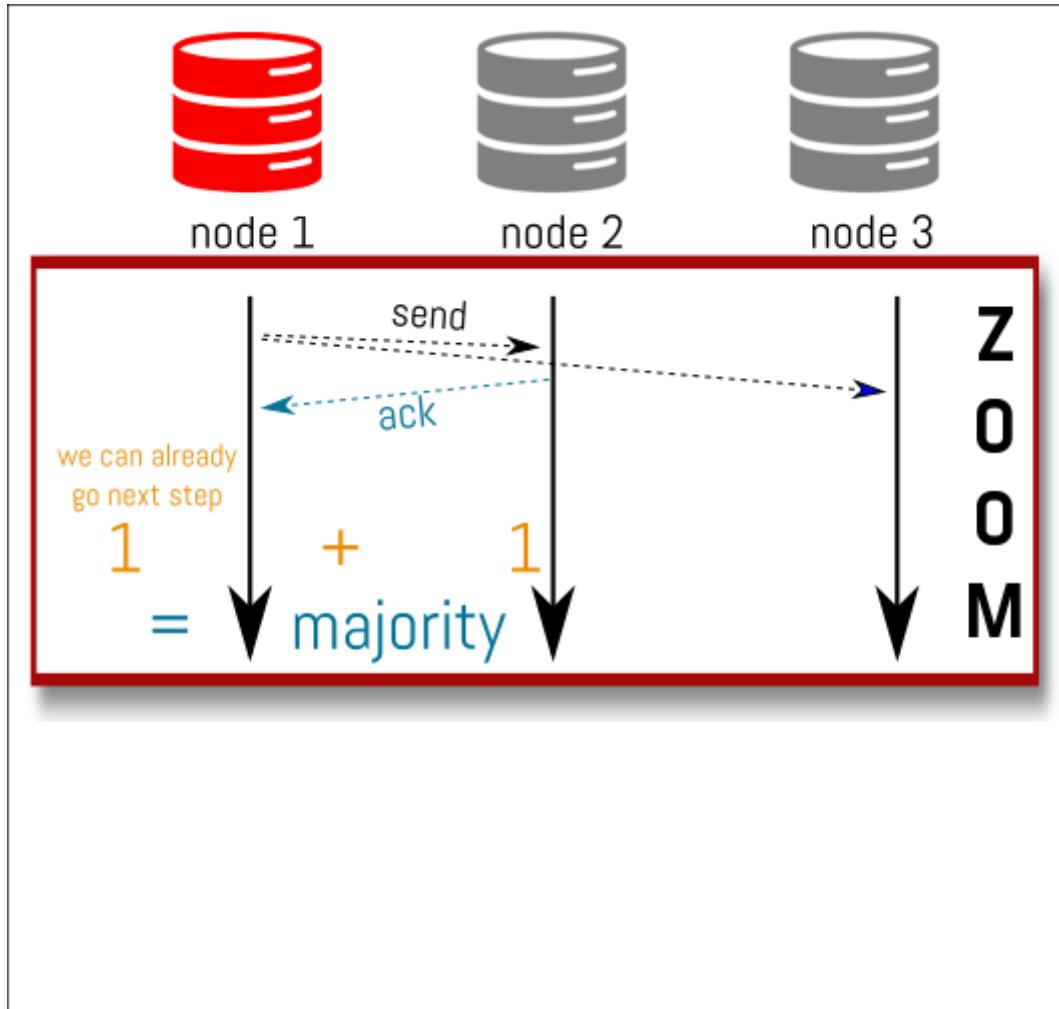
MySQL Group Replication (autocommit)



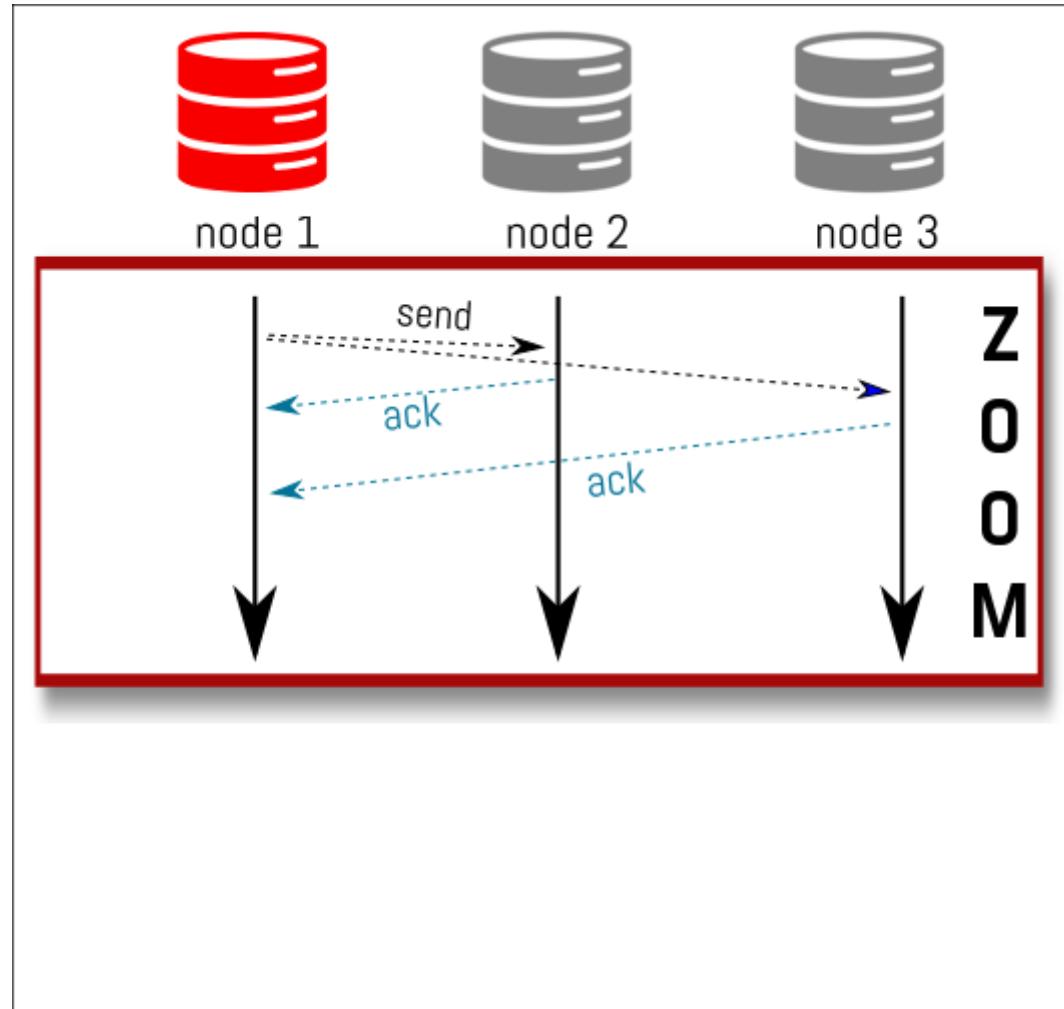
MySQL Group Replication (autocommit)



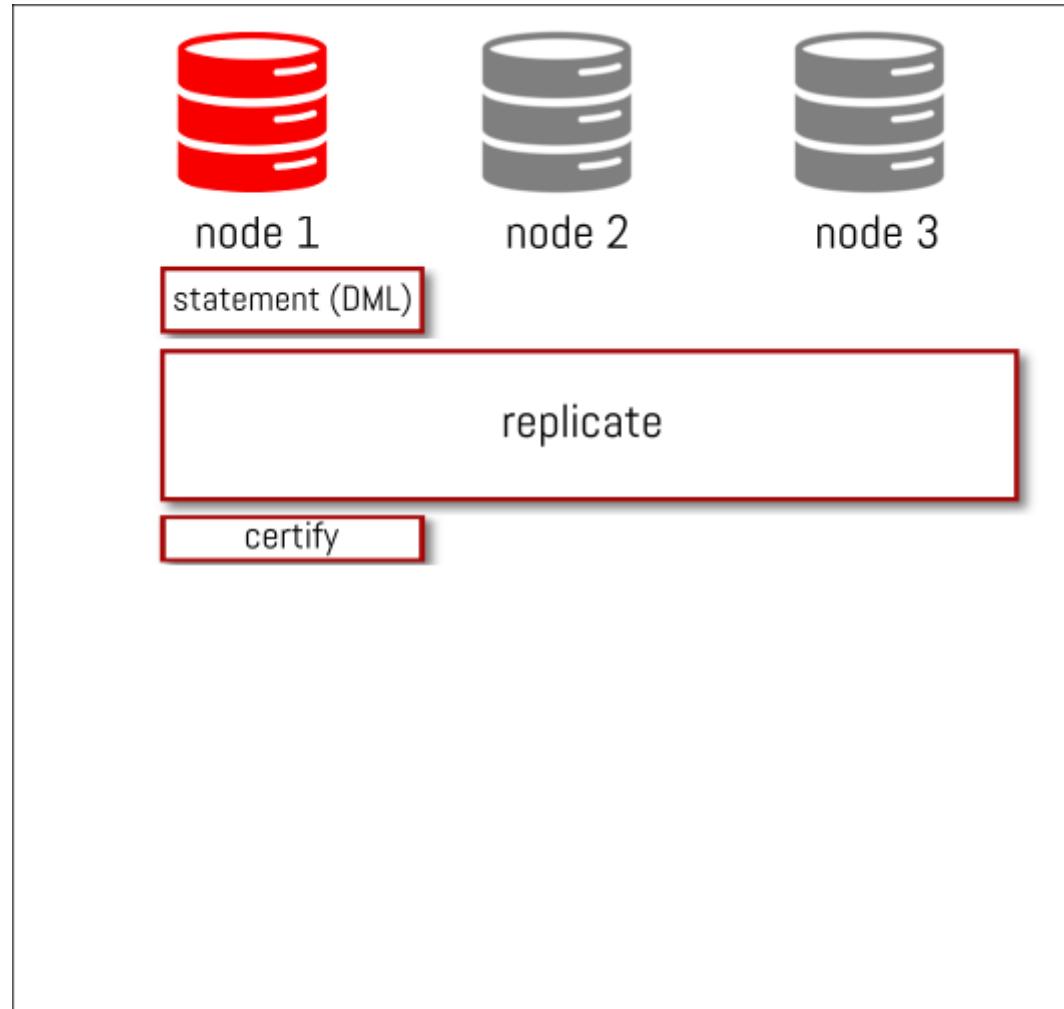
MySQL Group Replication (autocommit)



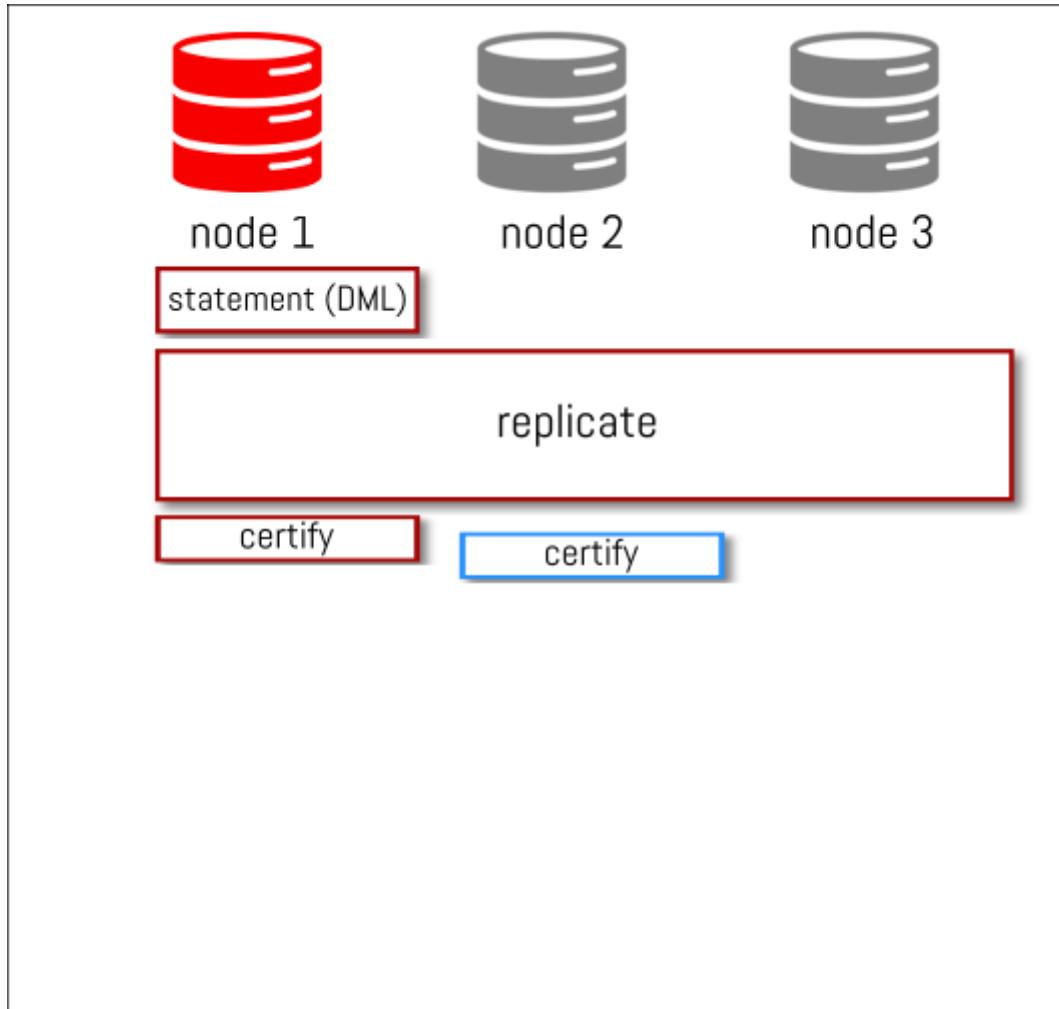
MySQL Group Replication (autocommit)



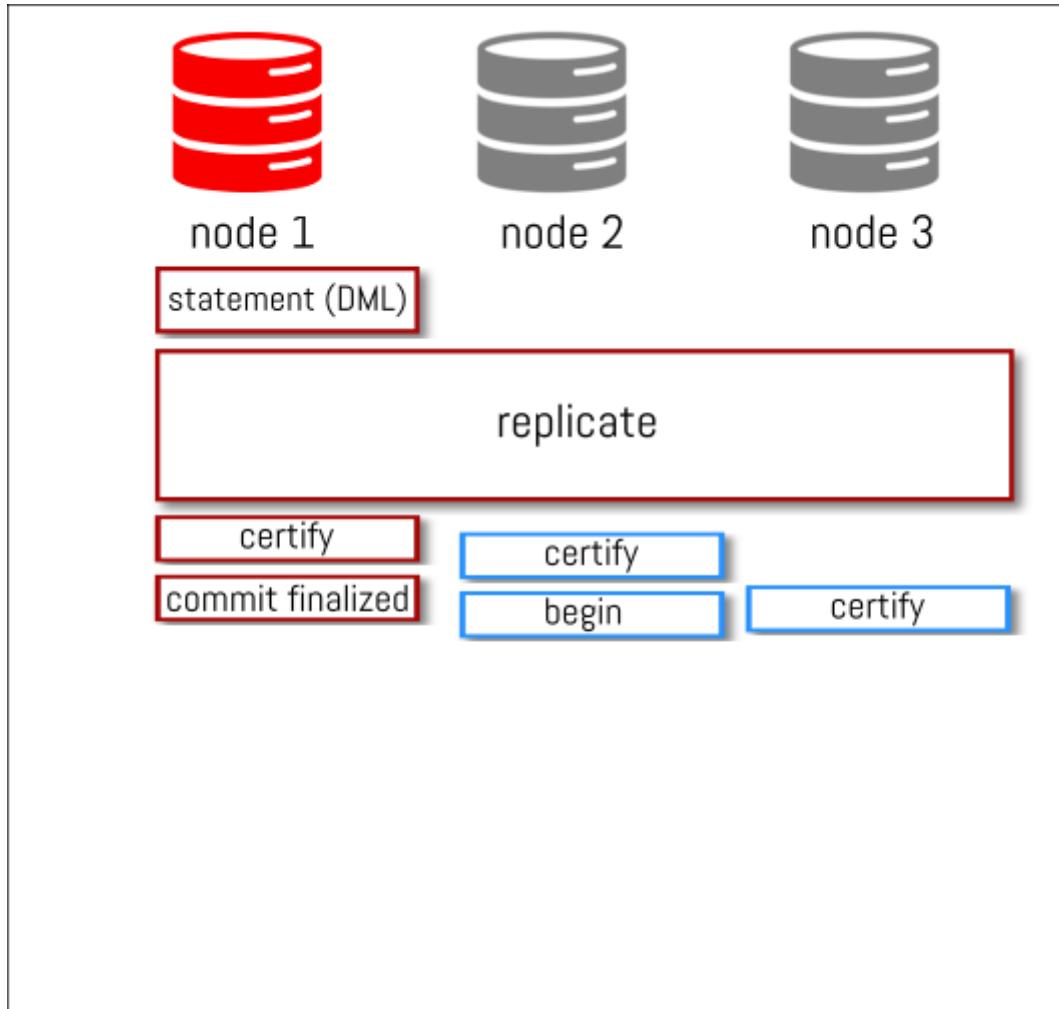
MySQL Group Replication (autocommit)



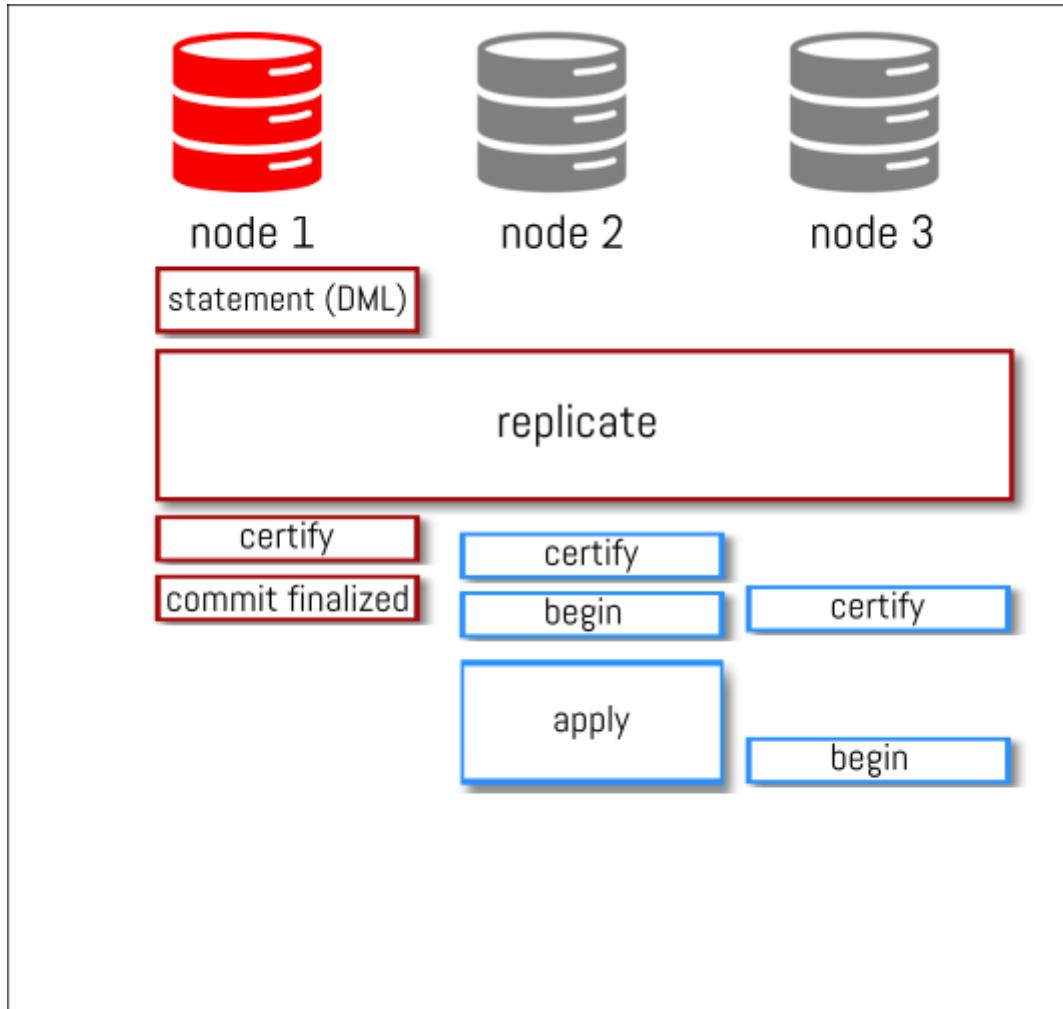
MySQL Group Replication (autocommit)



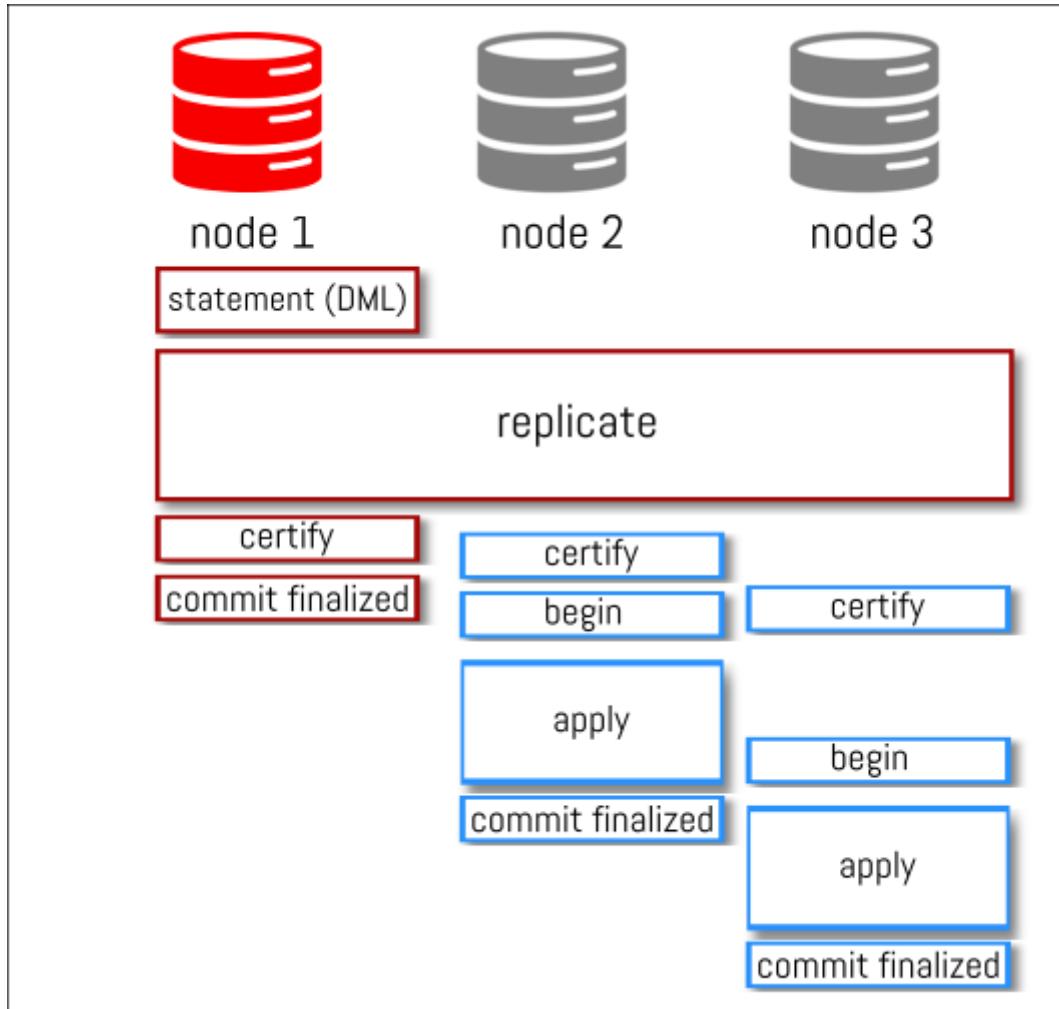
MySQL Group Replication (autocommit)



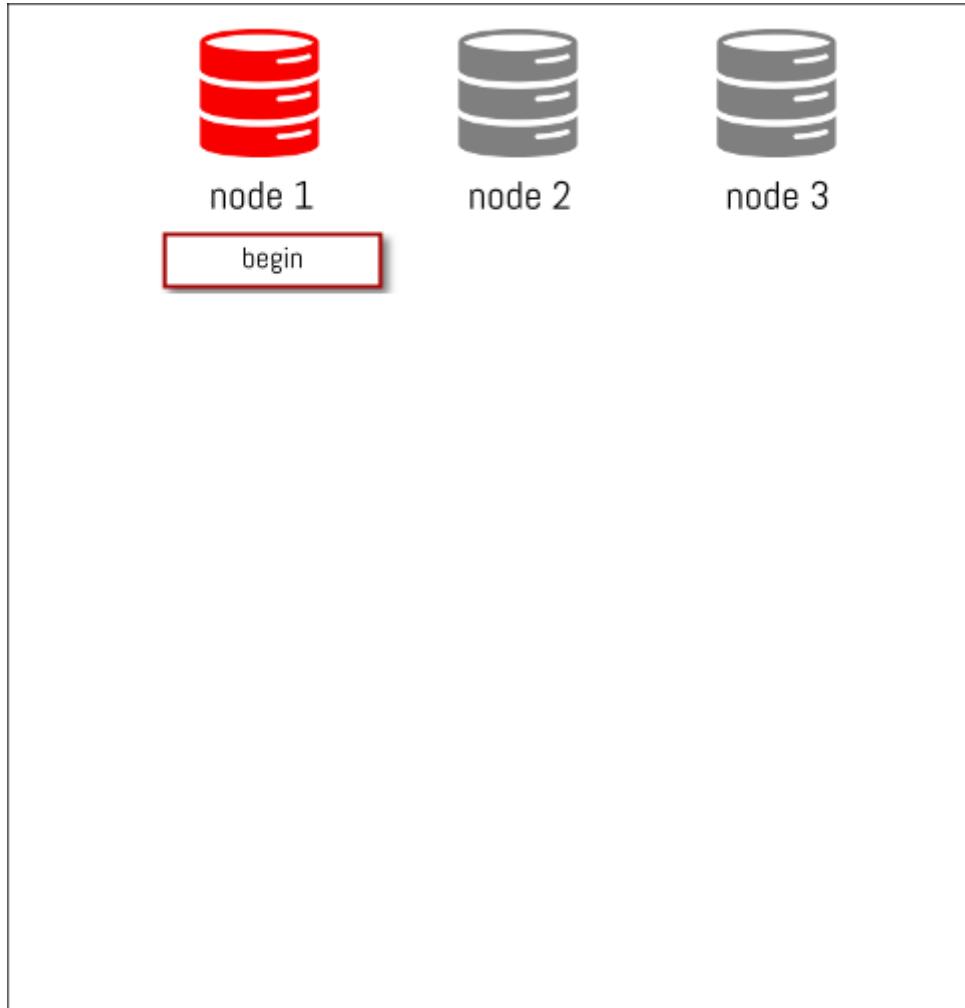
MySQL Group Replication (autocommit)



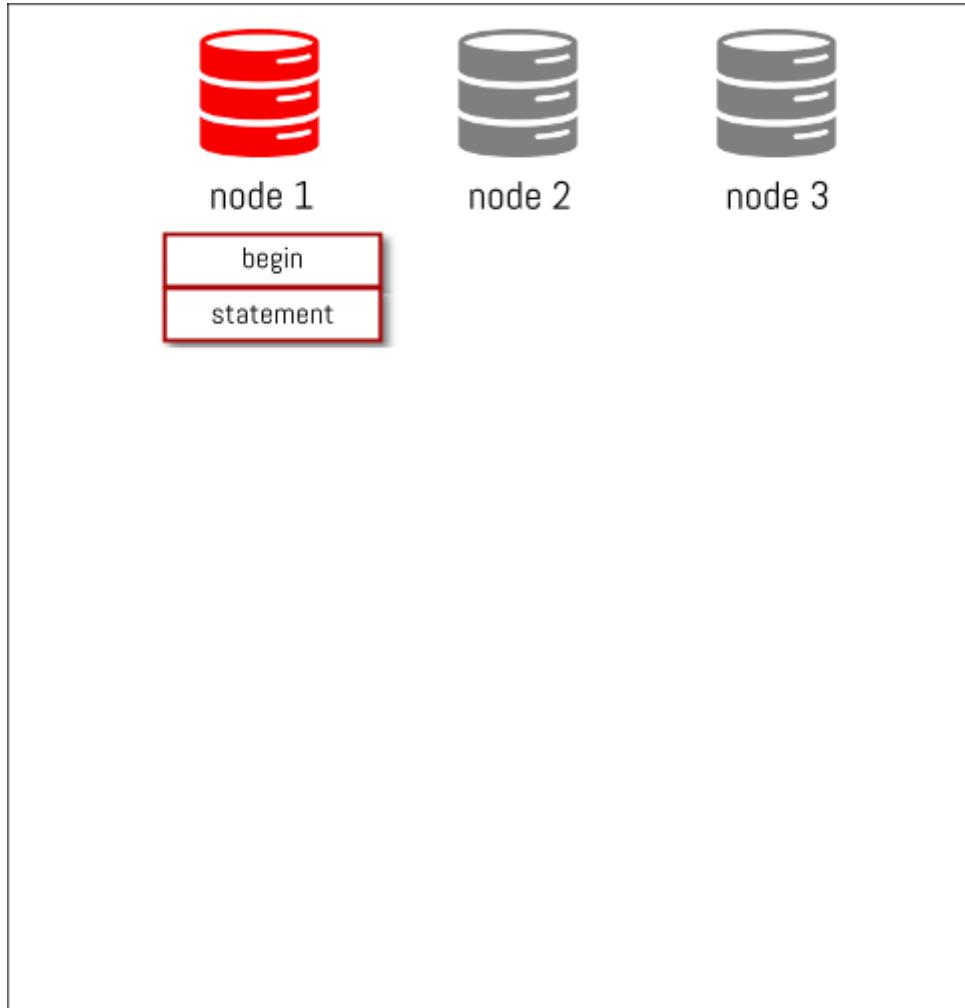
MySQL Group Replication (autocommit)



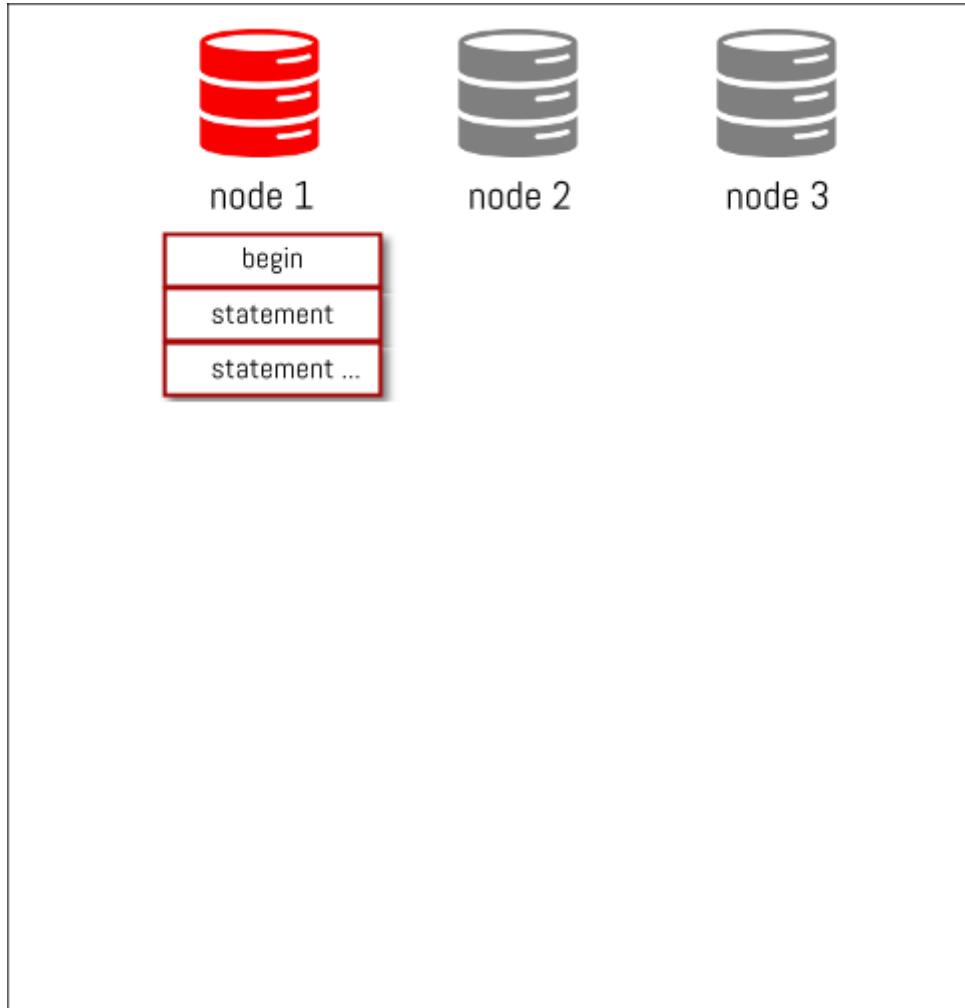
MySQL Group Replication (full transaction)



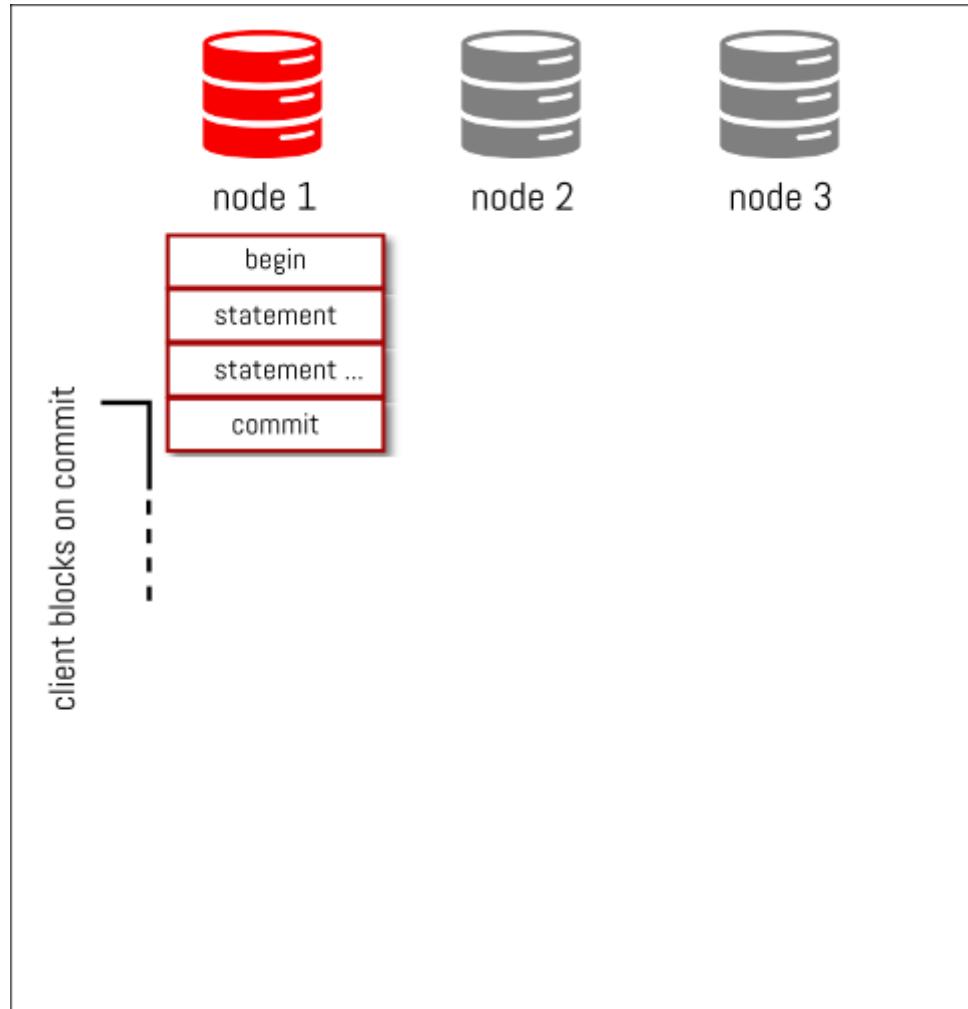
MySQL Group Replication (full transaction)



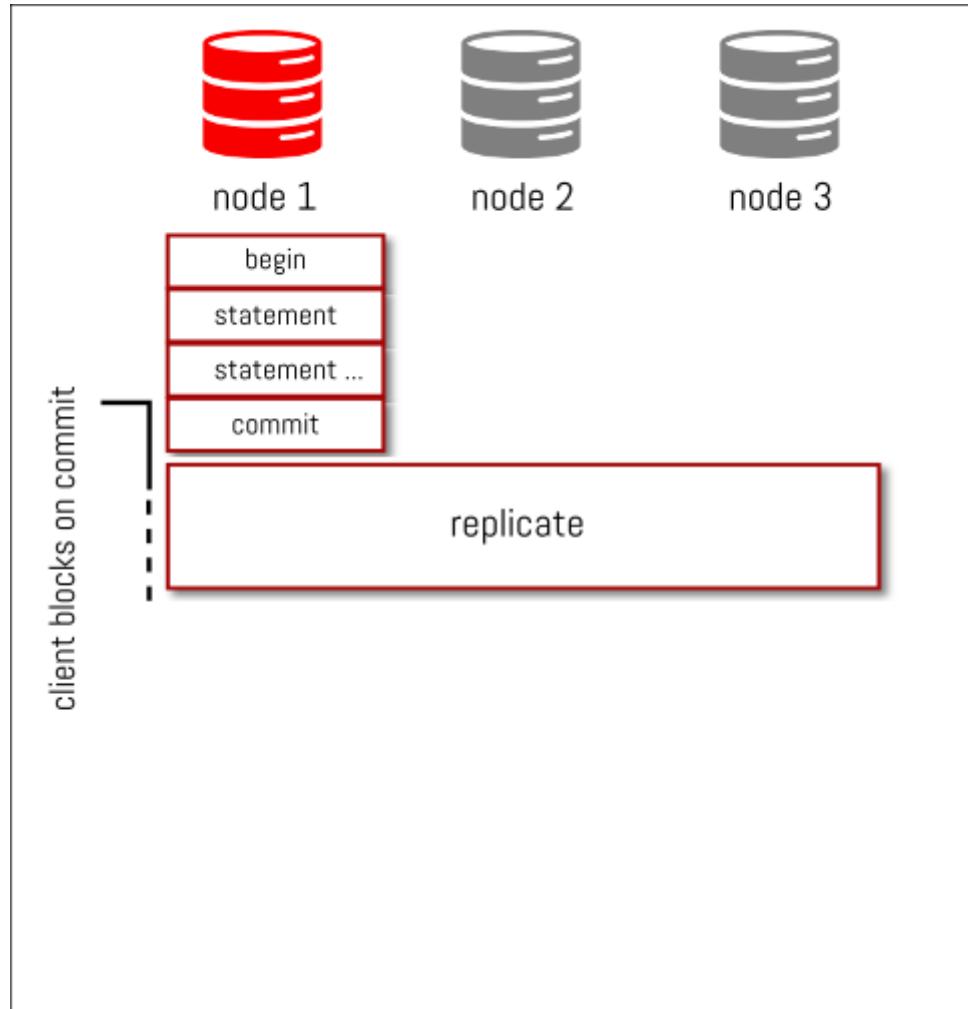
MySQL Group Replication (full transaction)



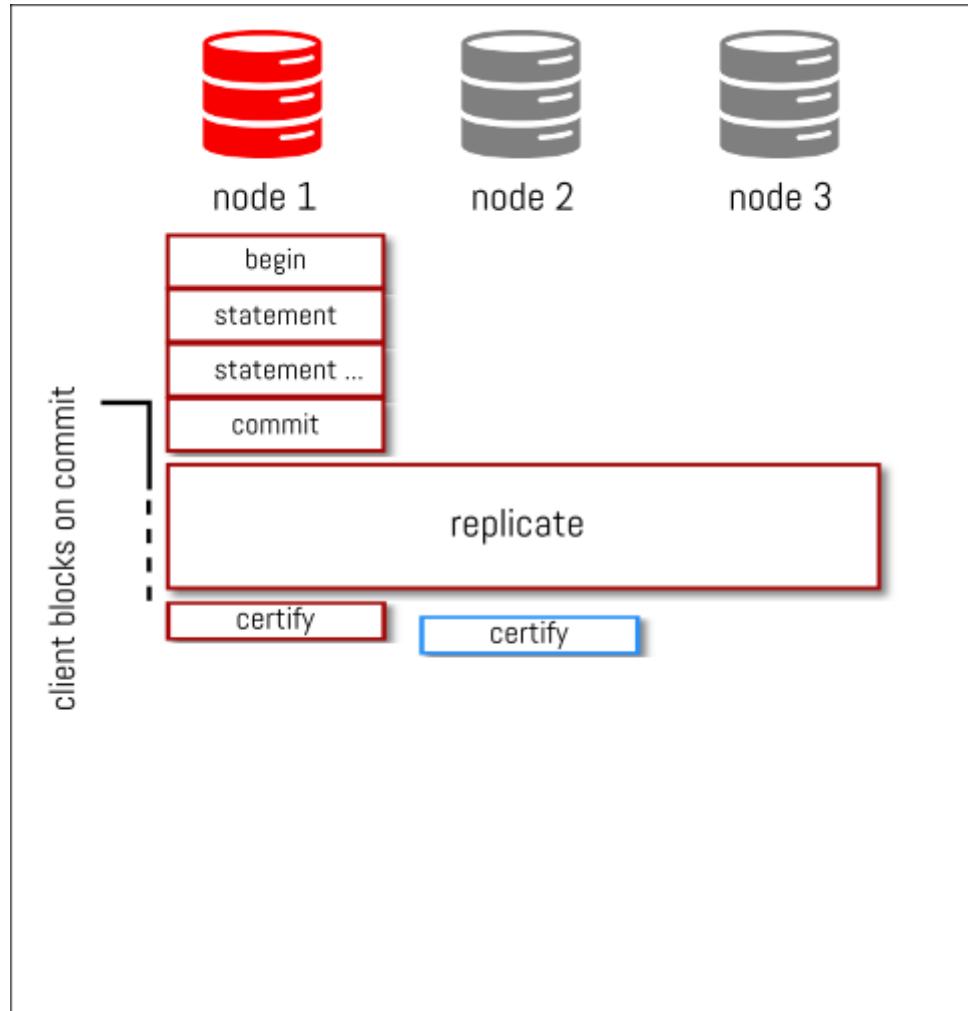
MySQL Group Replication (full transaction)



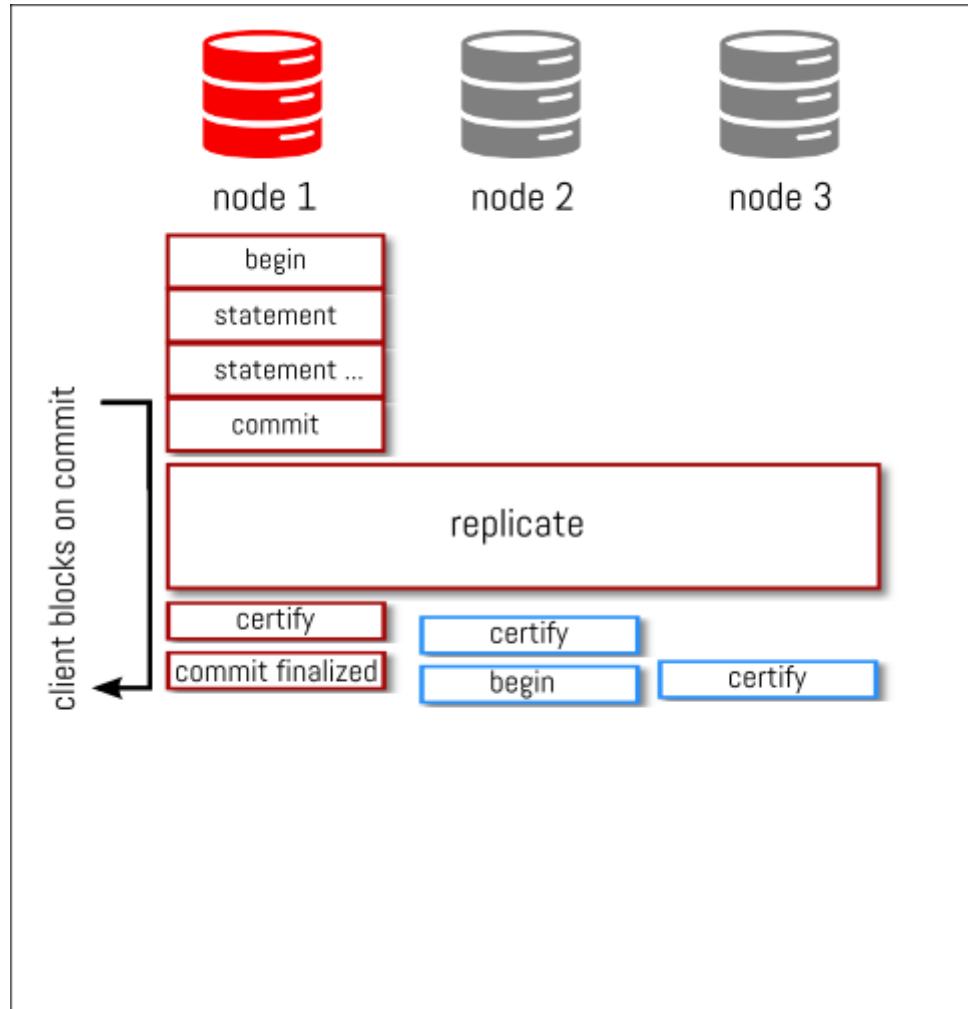
MySQL Group Replication (full transaction)



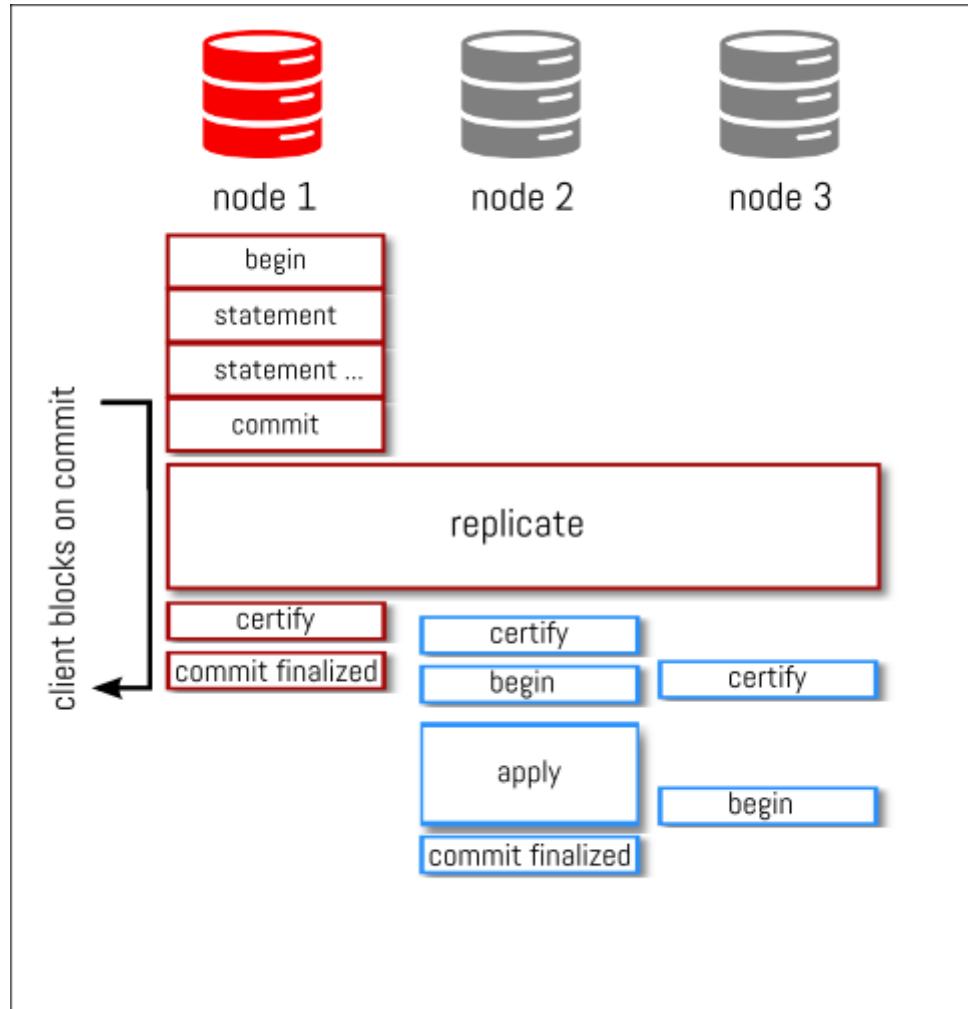
MySQL Group Replication (full transaction)



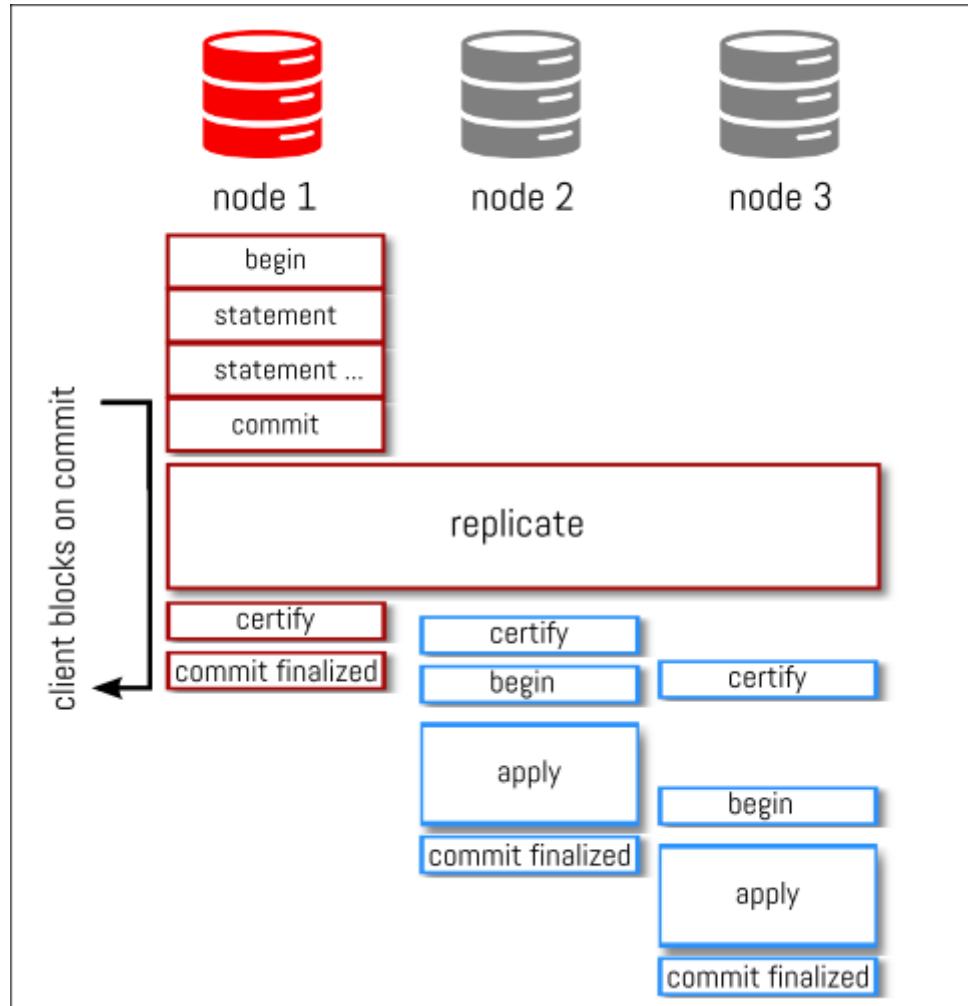
MySQL Group Replication (full transaction)



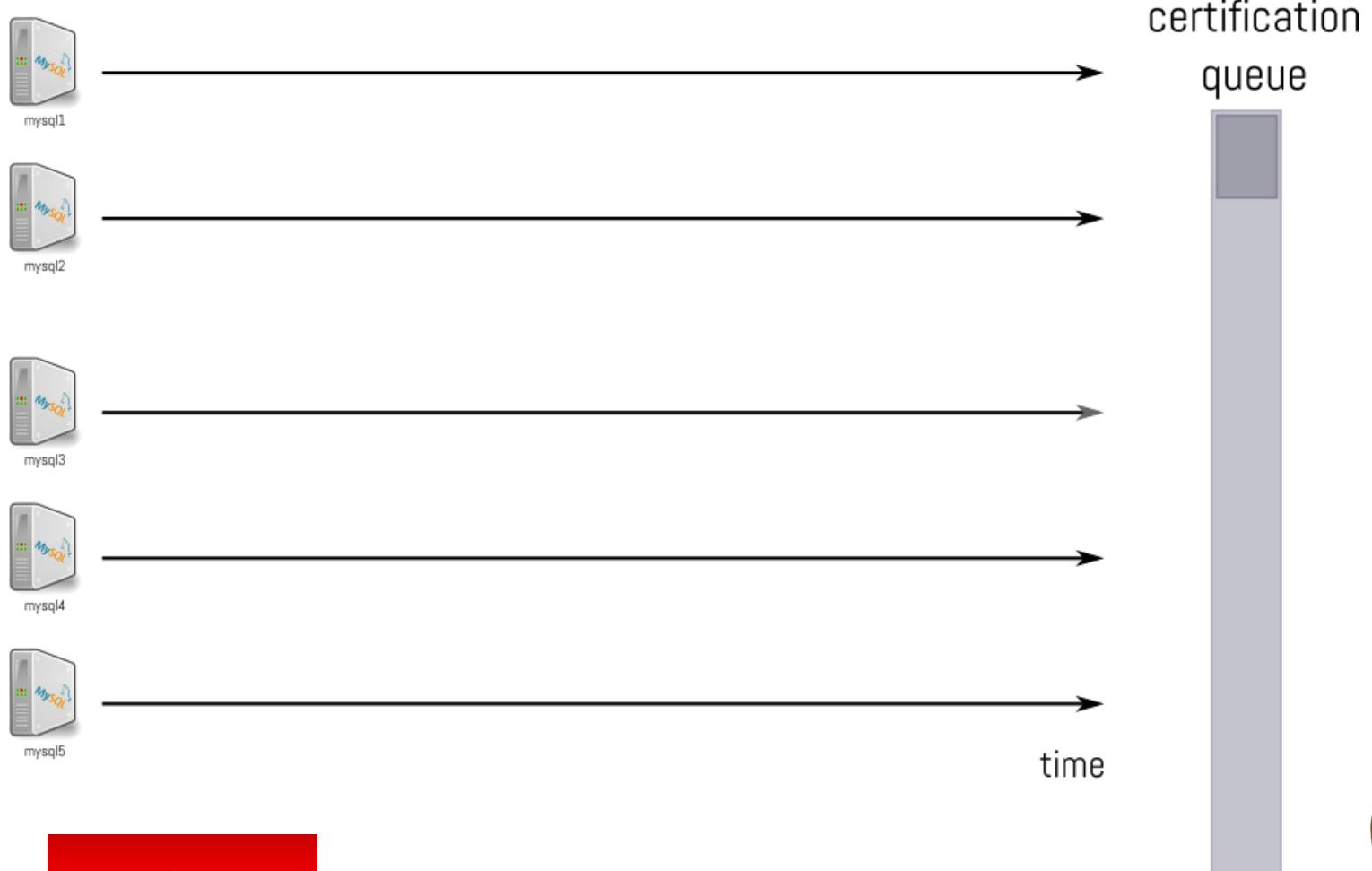
MySQL Group Replication (full transaction)



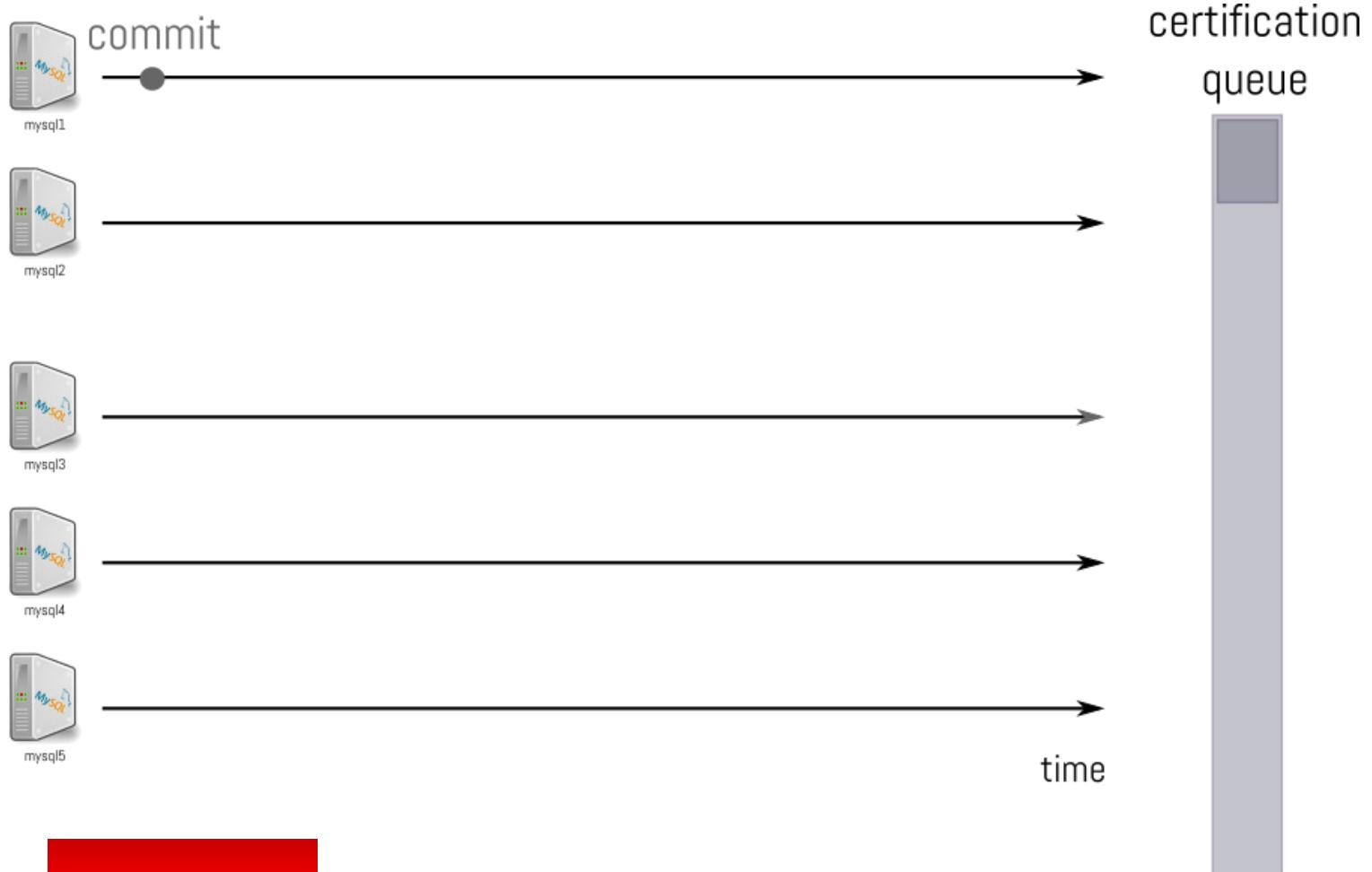
MySQL Group Replication (full transaction)



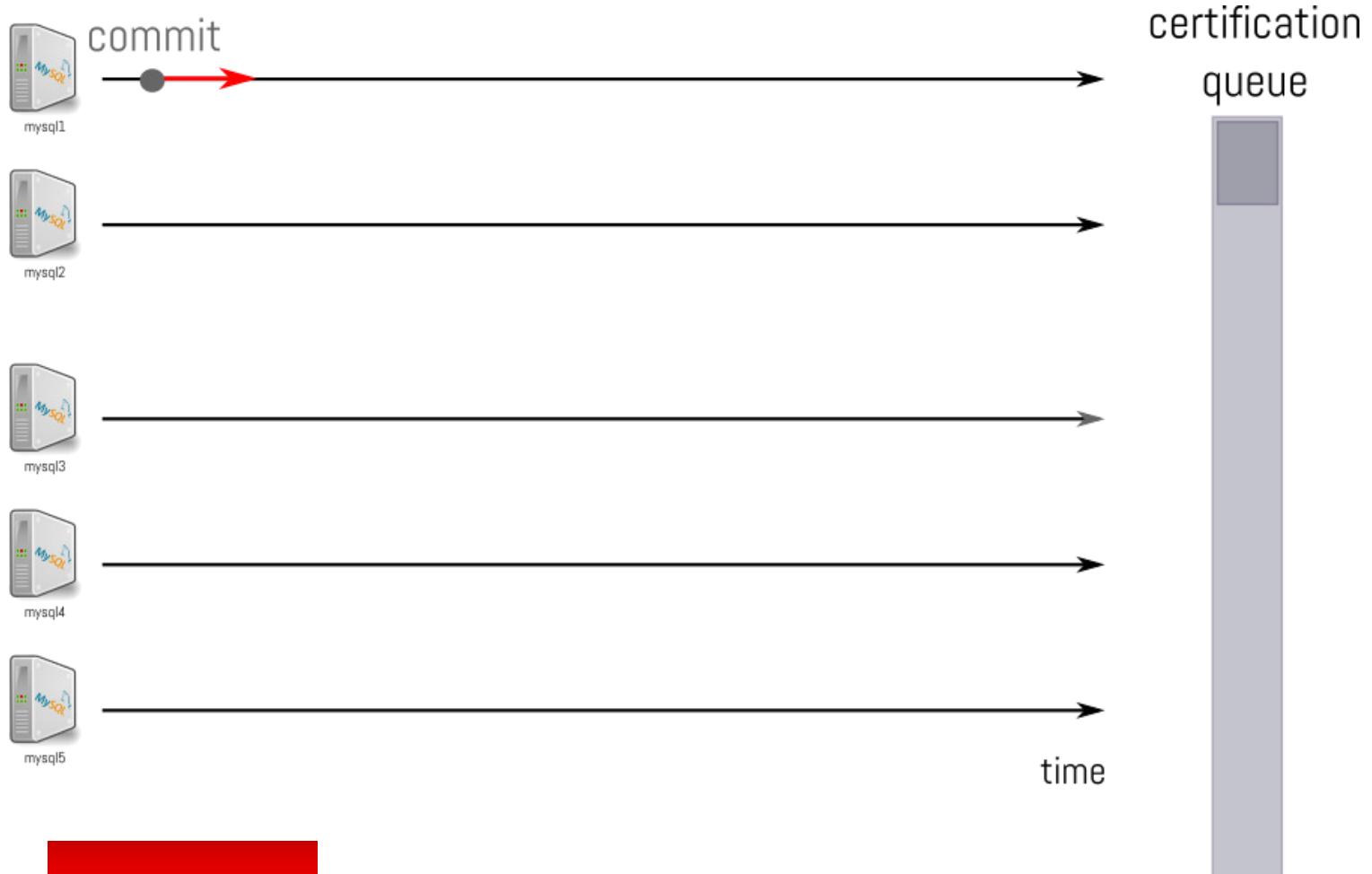
Group Replication : Total Order Delivery - GTID



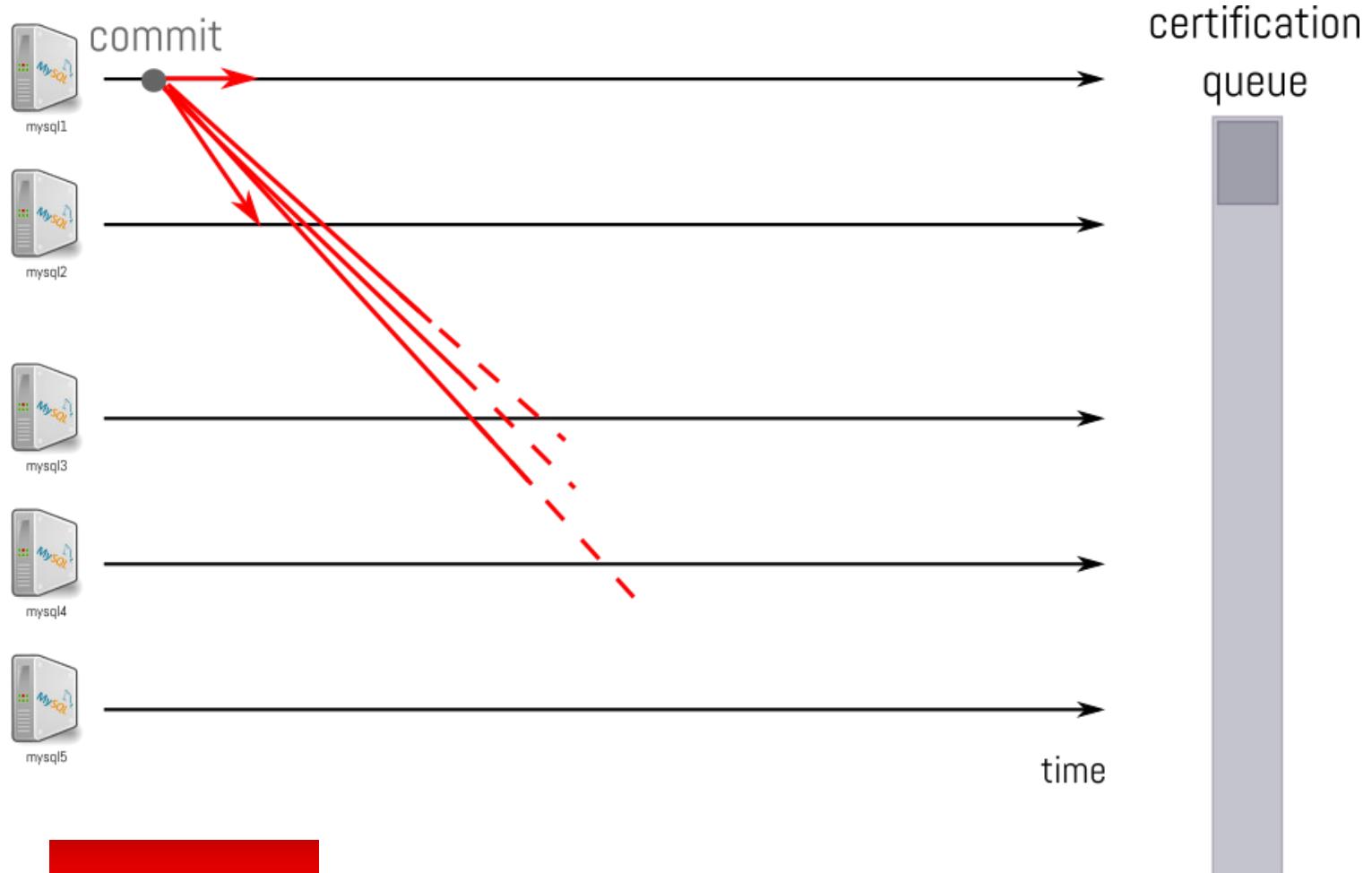
Group Replication : Total Order Delivery - GTID



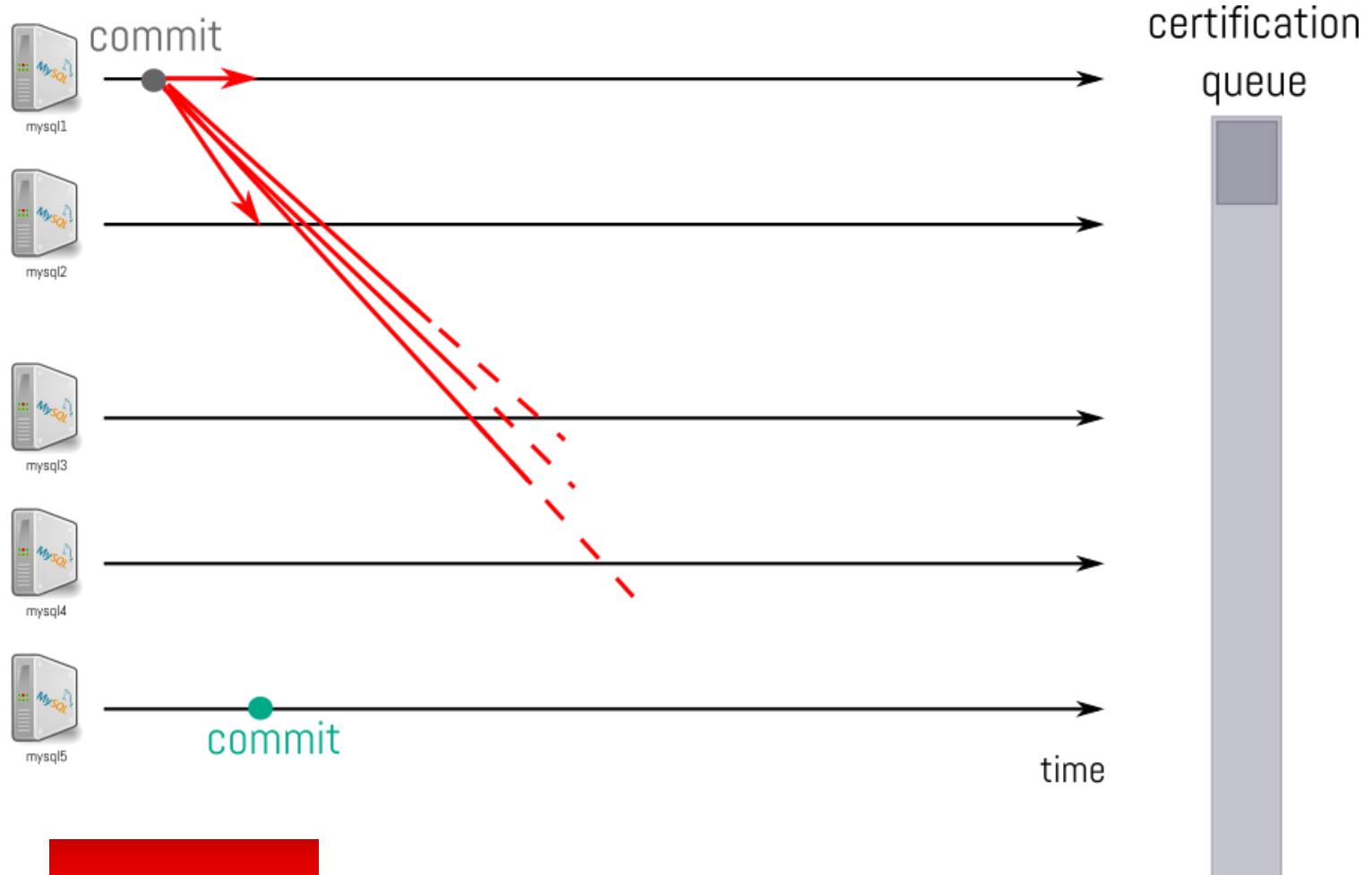
Group Replication : Total Order Delivery - GTID



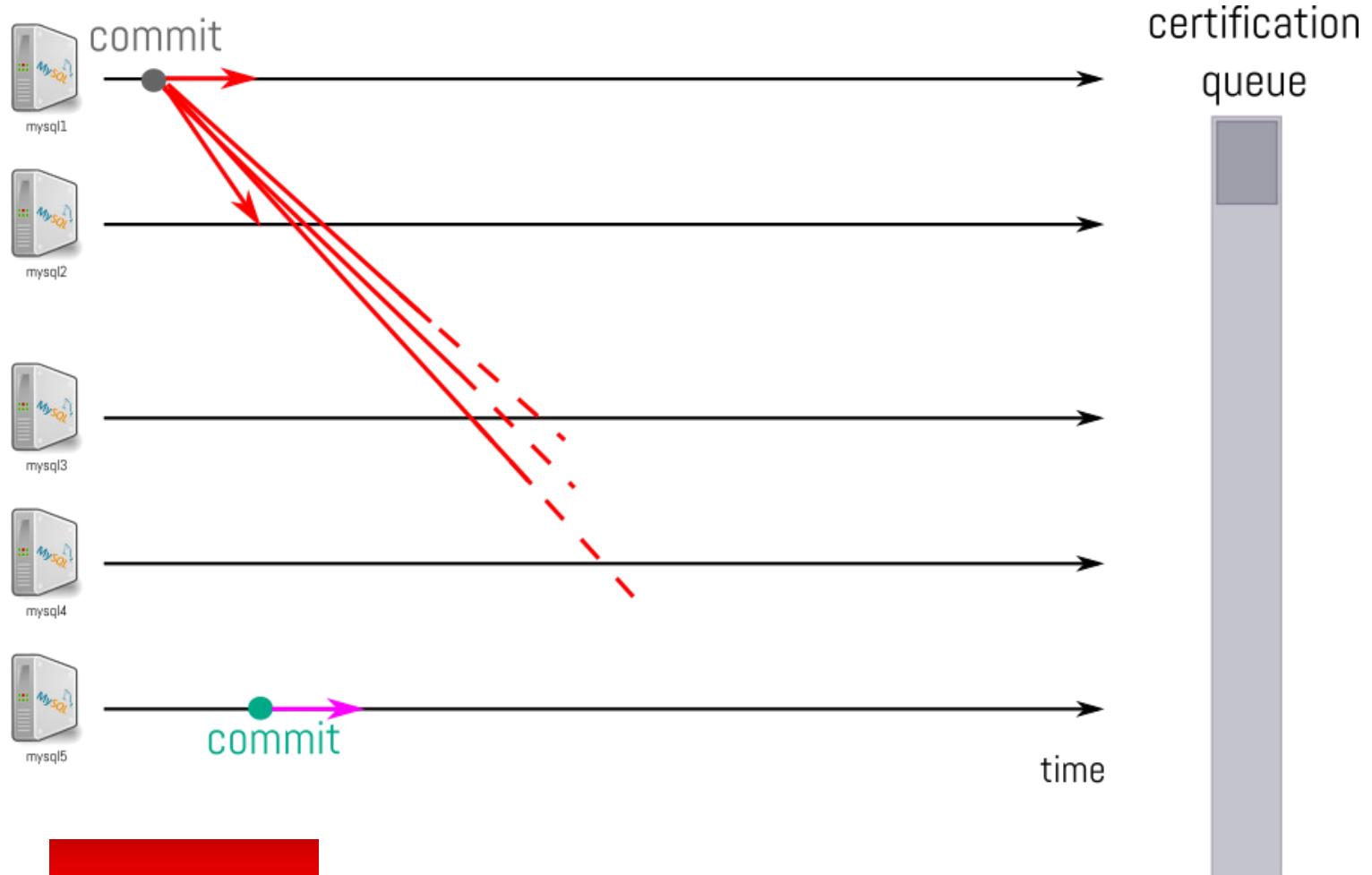
Group Replication : Total Order Delivery - GTID



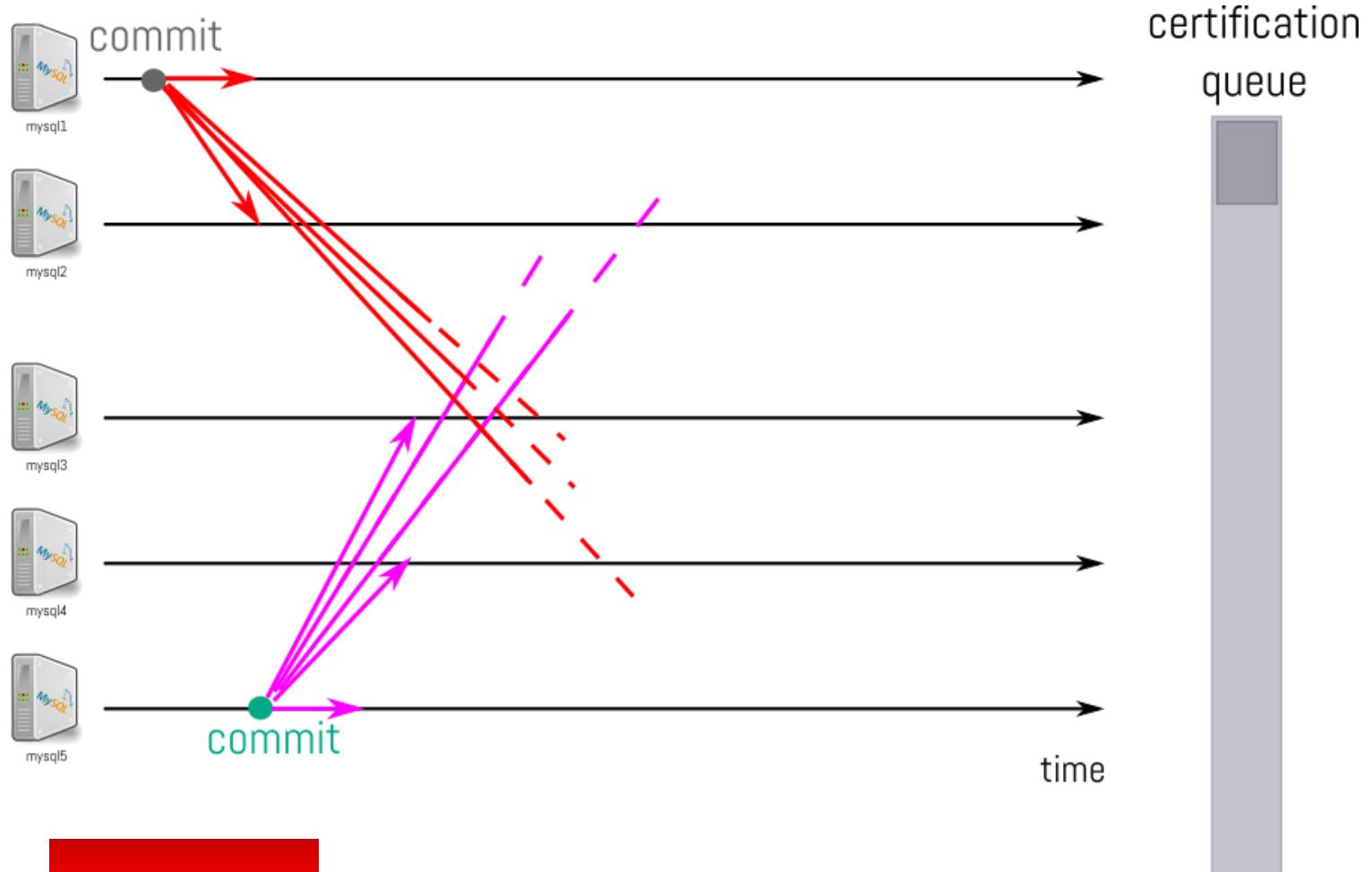
Group Replication : Total Order Delivery - GTID



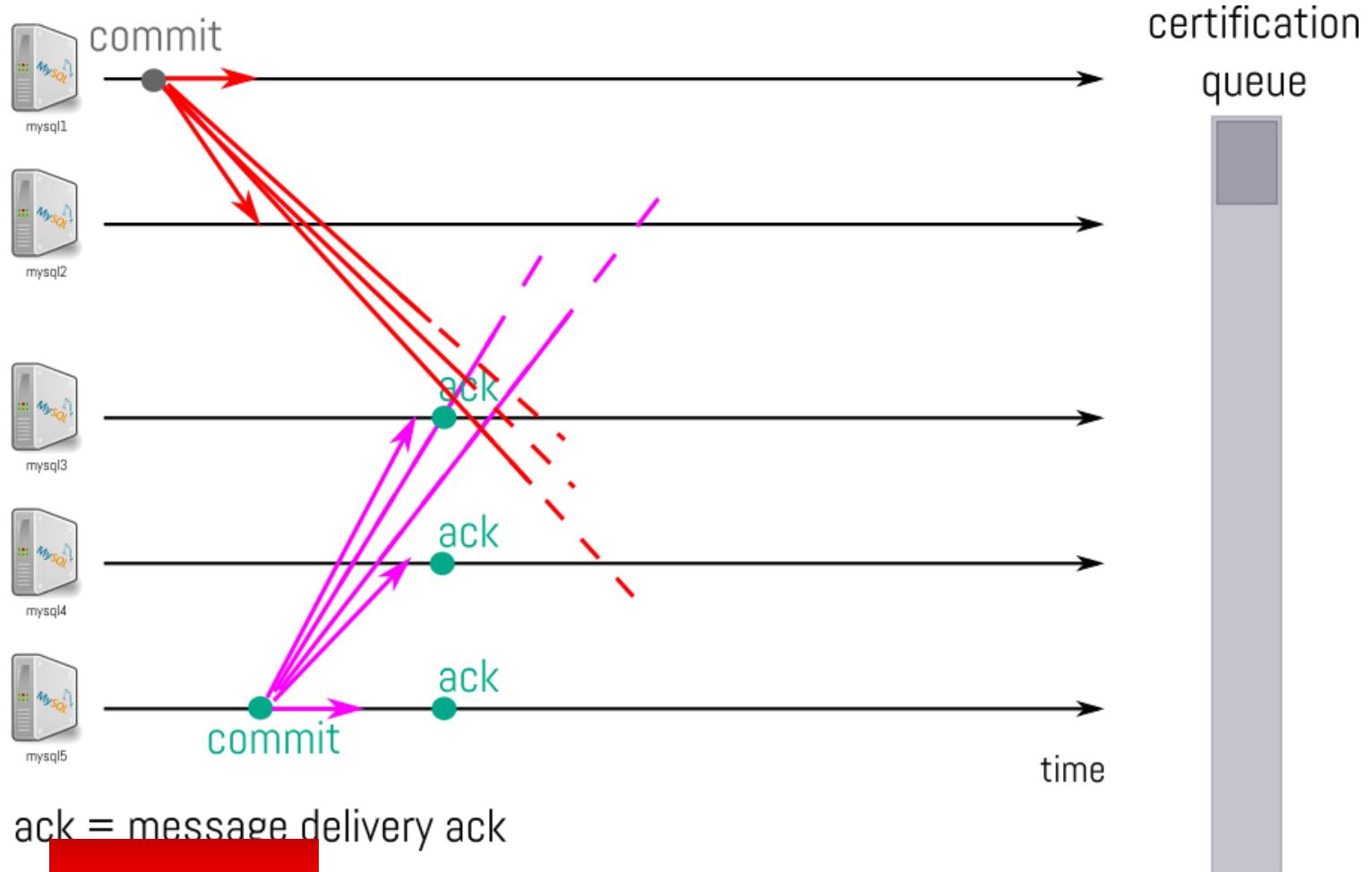
Group Replication : Total Order Delivery - GTID



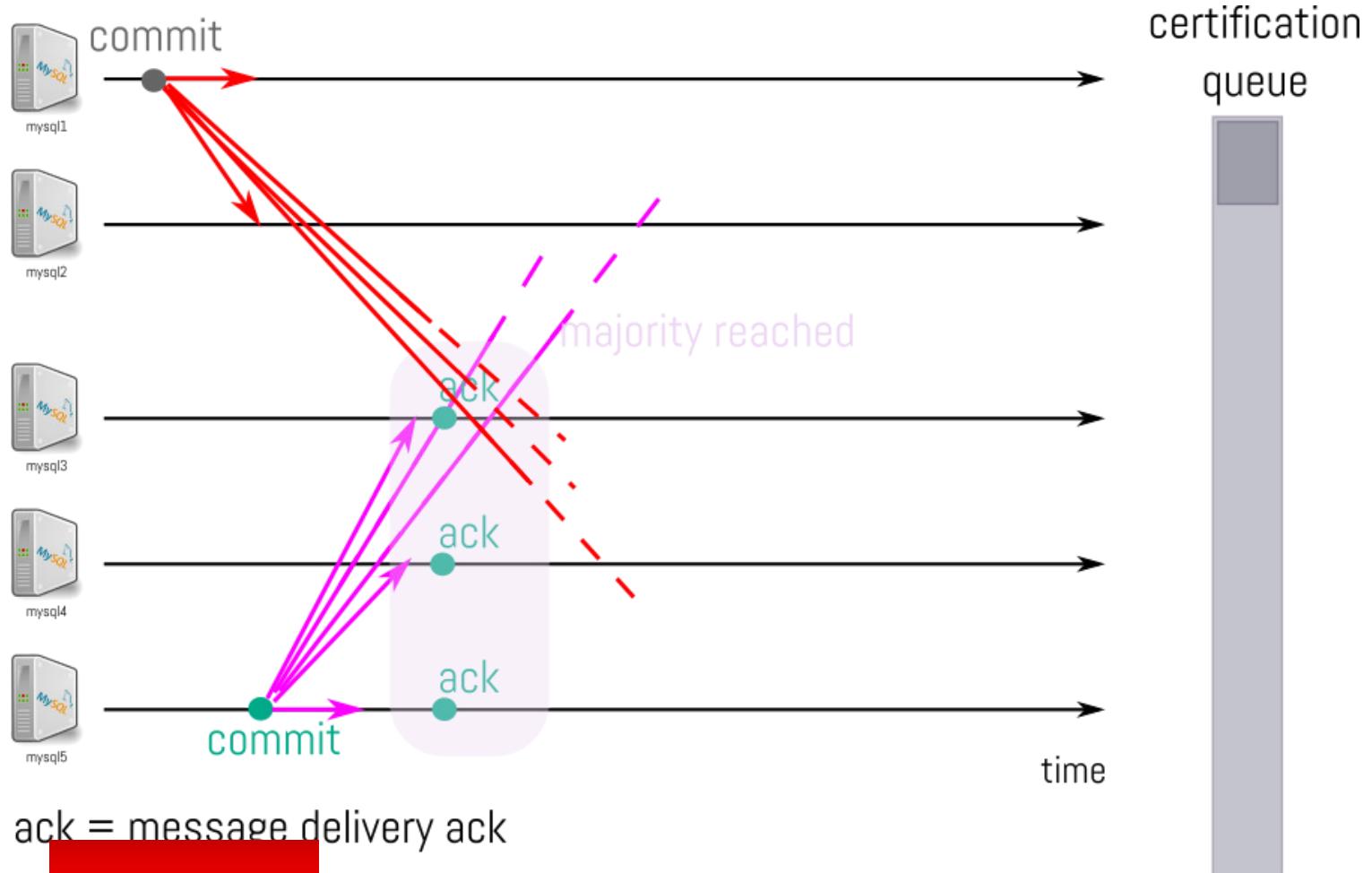
Group Replication : Total Order Delivery - GTID



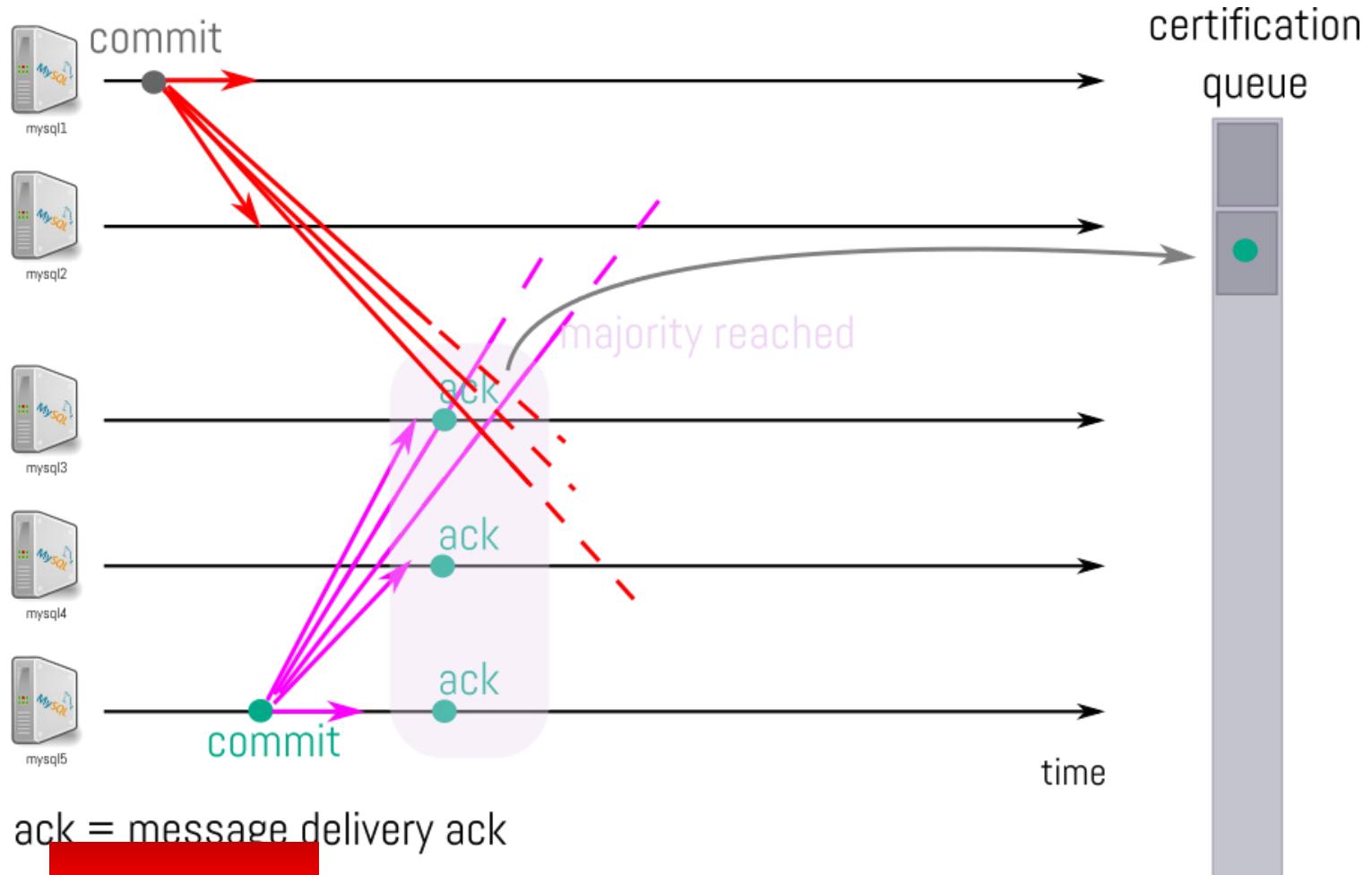
Group Replication : Total Order Delivery - GTID



Group Replication : Total Order Delivery - GTID

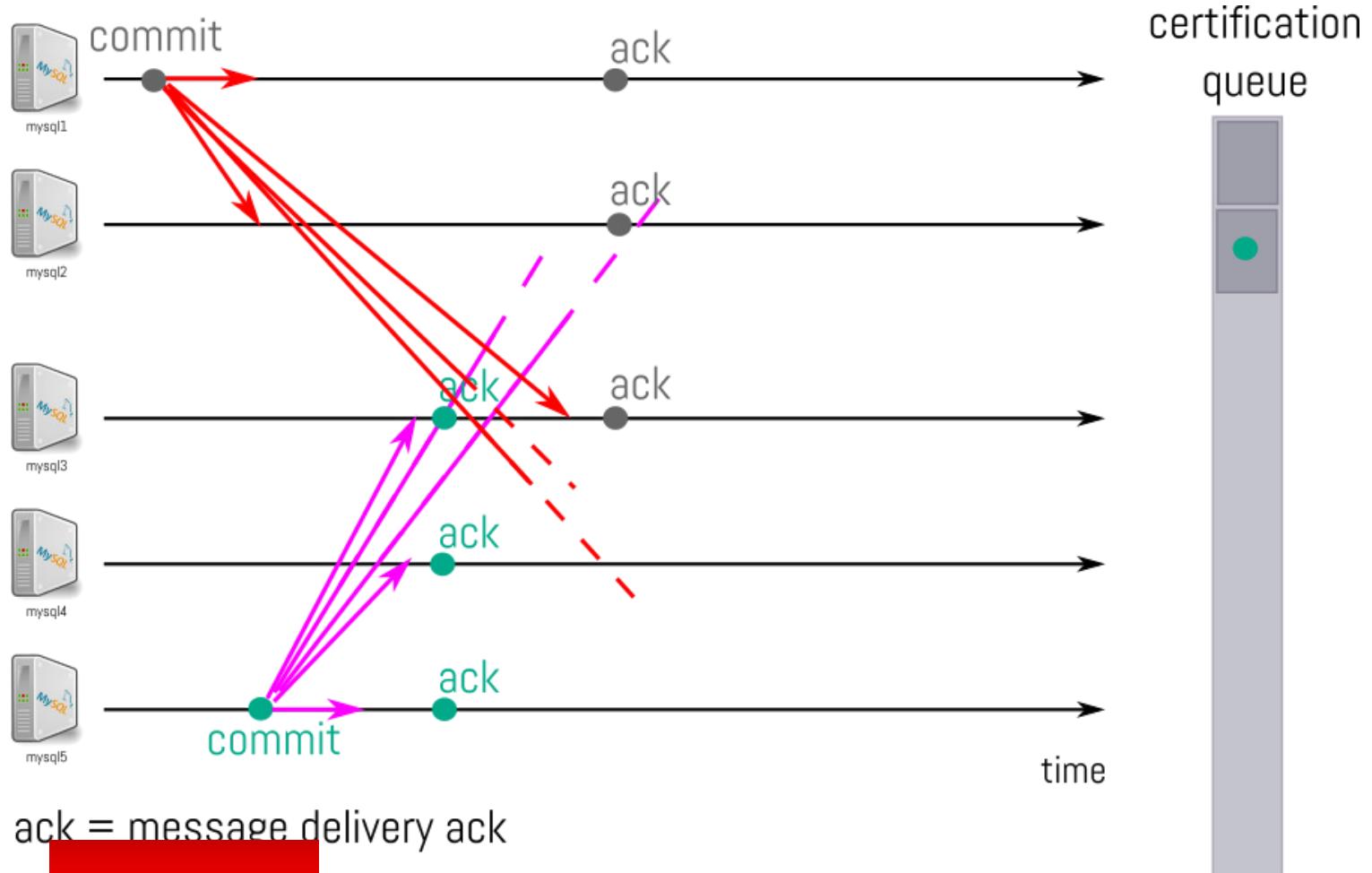


Group Replication : Total Order Delivery - GTID

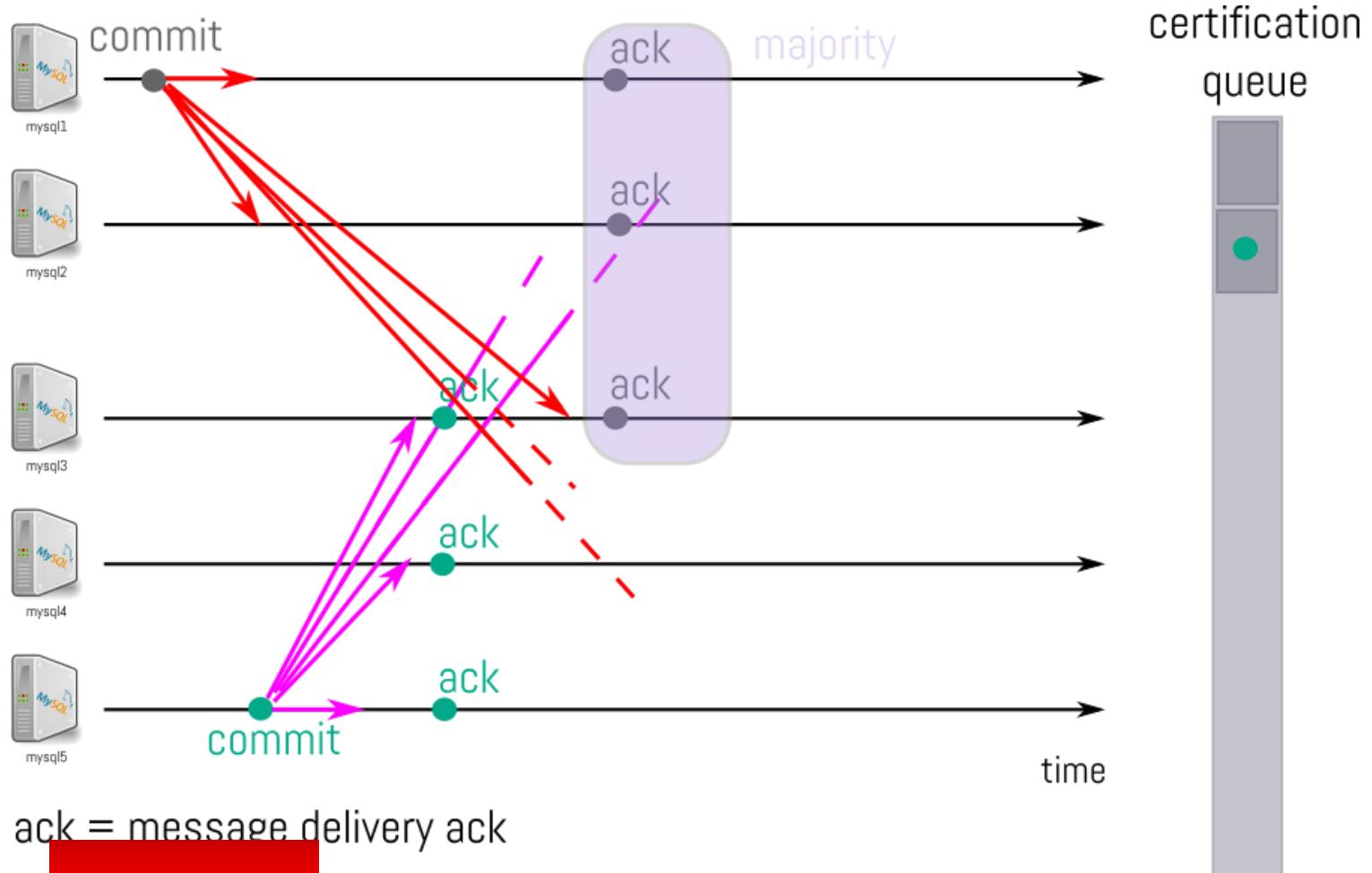


ORACLE®

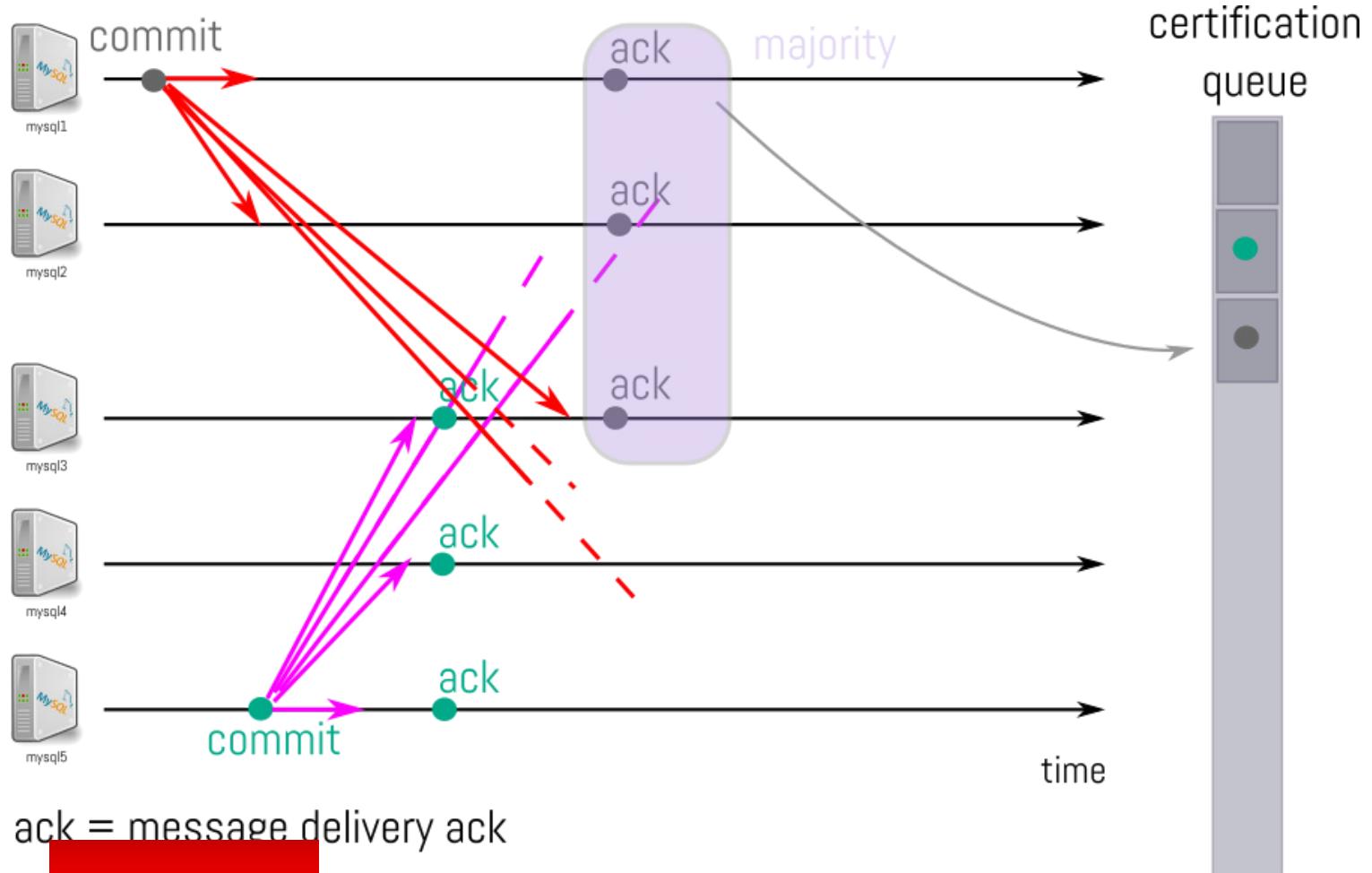
Group Replication : Total Order Delivery - GTID



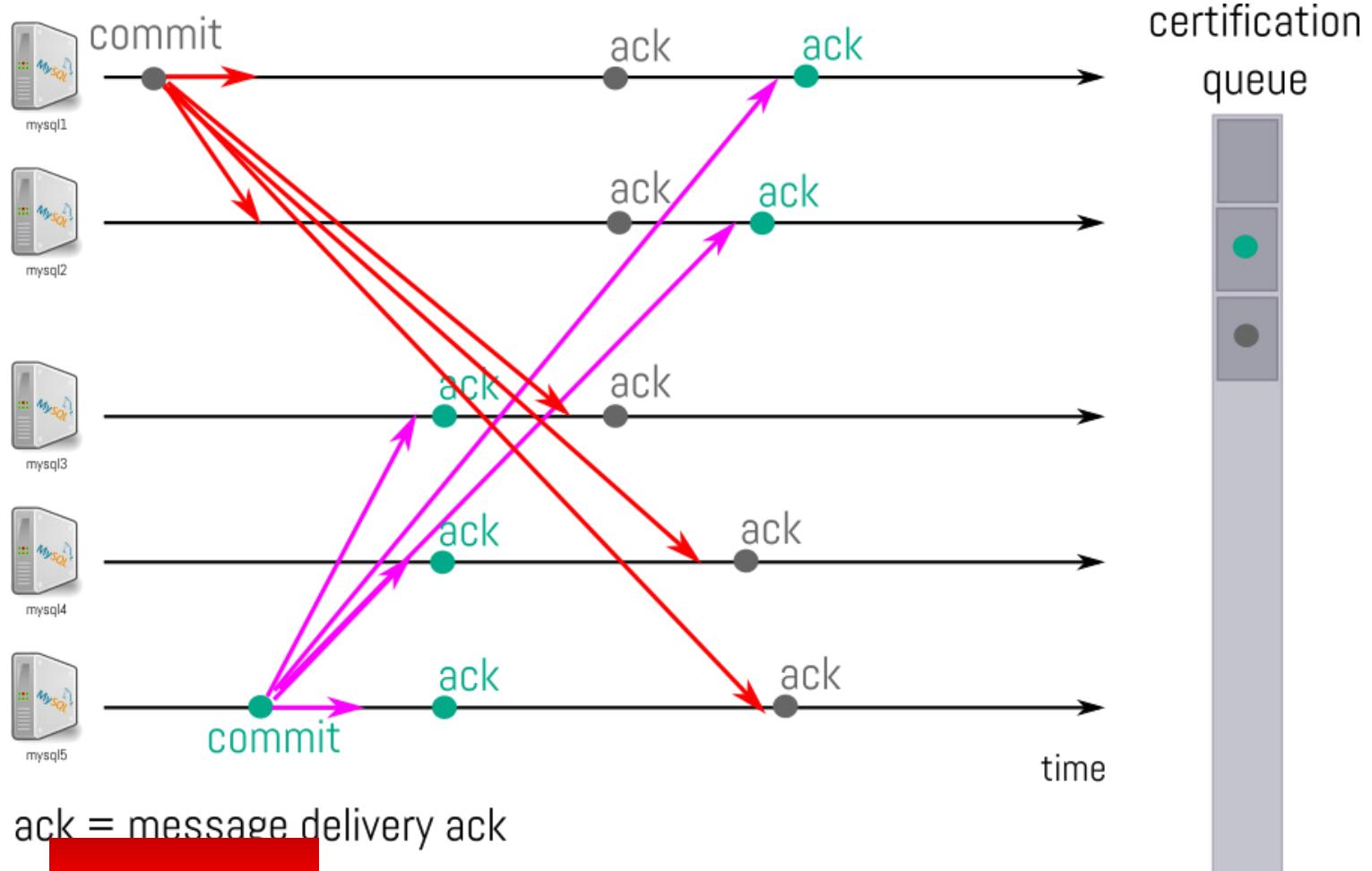
Group Replication : Total Order Delivery - GTID



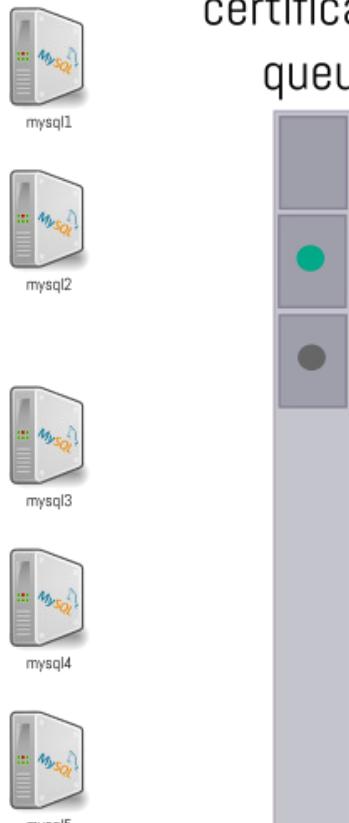
Group Replication : Total Order Delivery - GTID



Group Replication : Total Order Delivery - GTID



Group Replication : Total Order Delivery - GTID



certification

queue

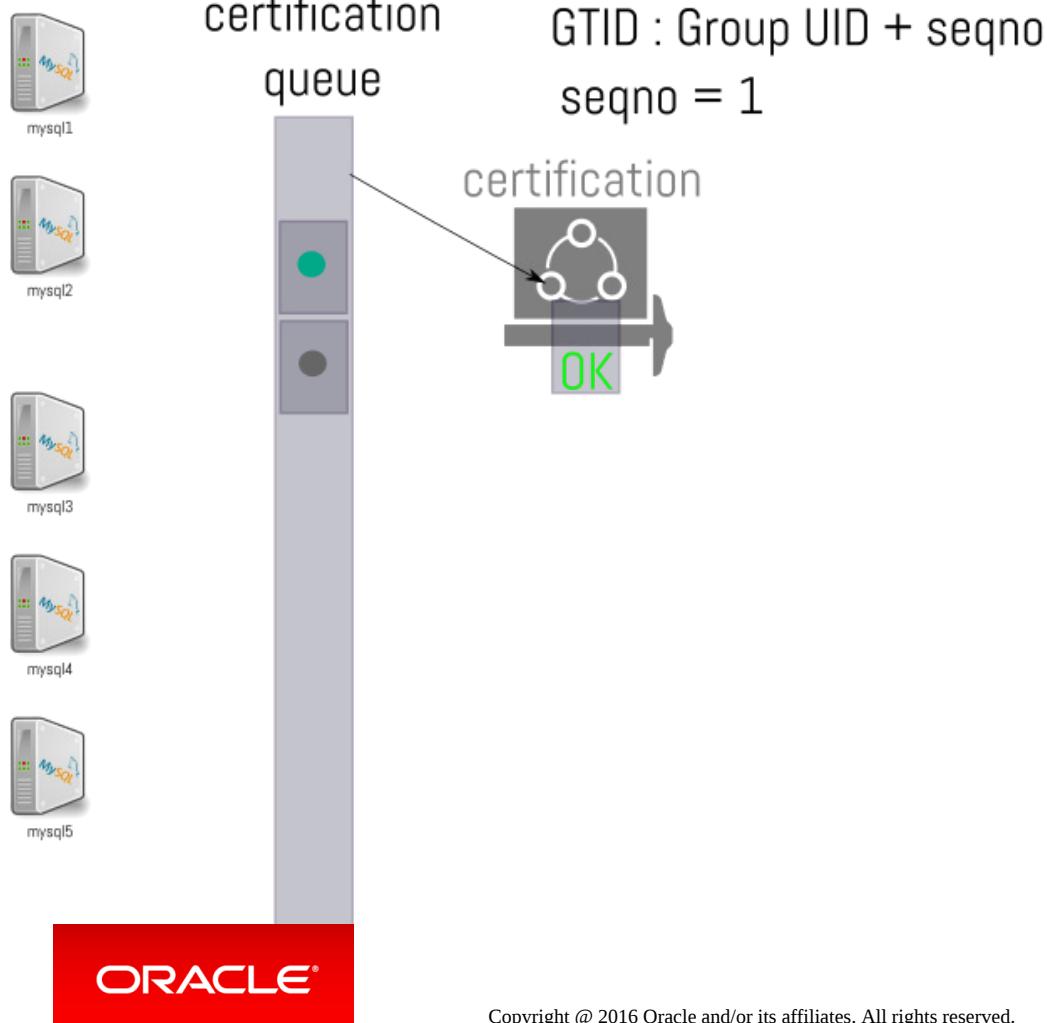
GTID : Group UID + seqno

seqno = 1

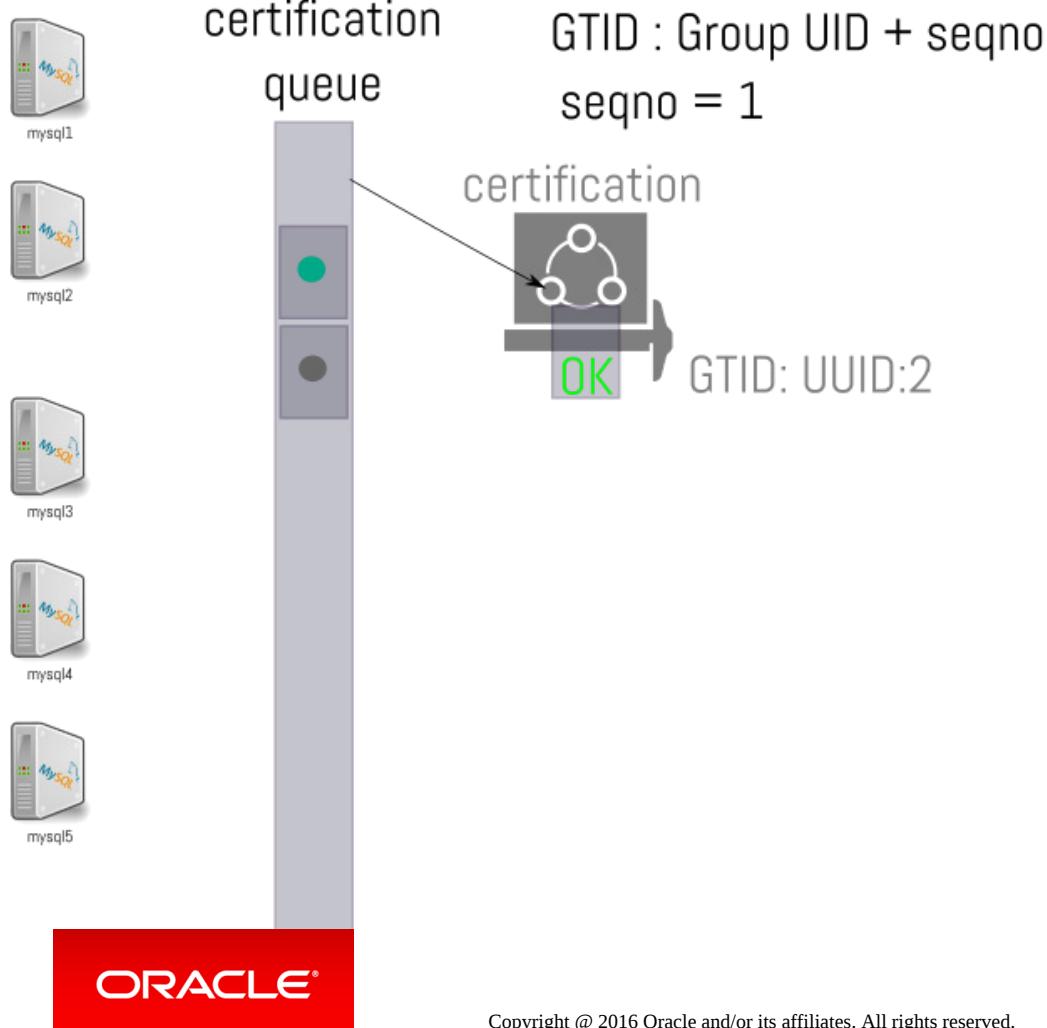
certification



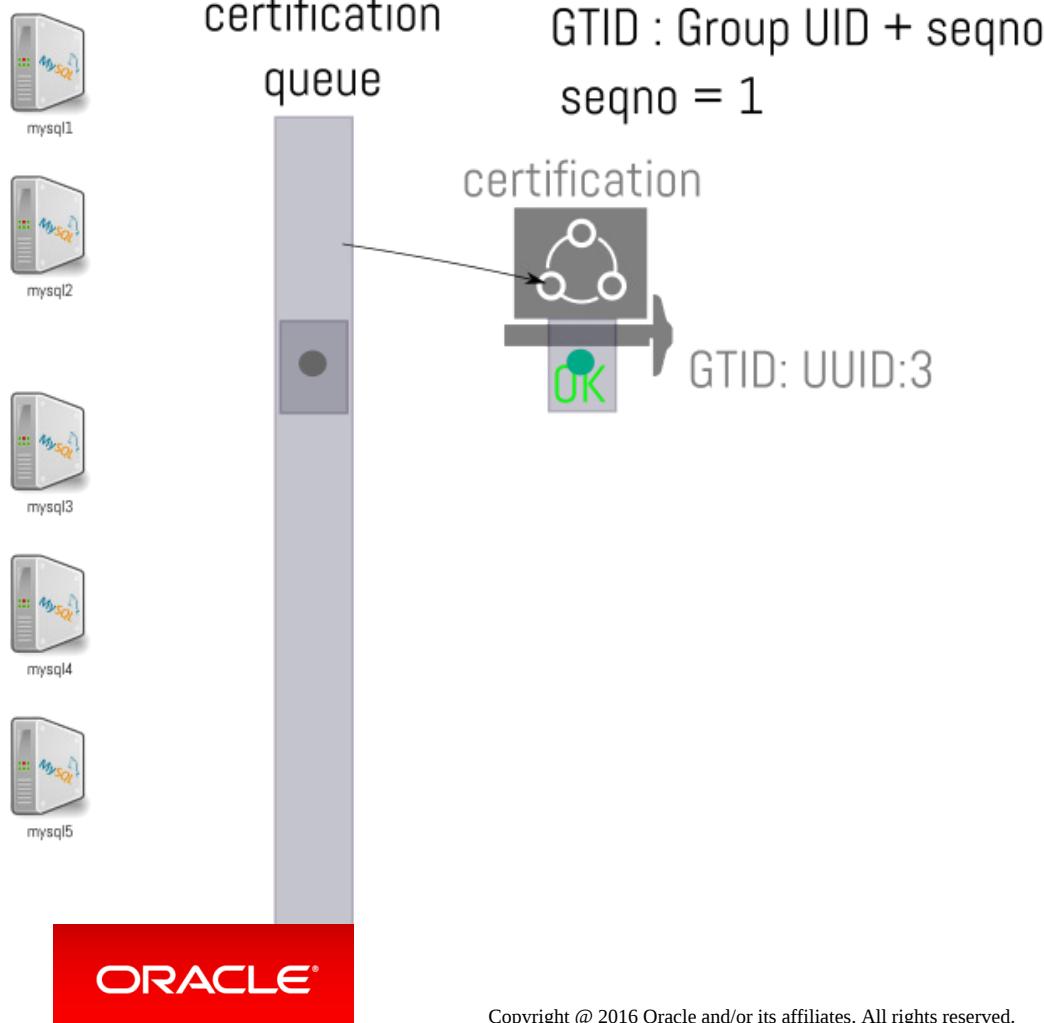
Group Replication : Total Order Delivery - GTID



Group Replication : Total Order Delivery - GTID



Group Replication : Total Order Delivery - GTID



Group Replication : Total Order Delivery - GTID



mysql1



mysql2



mysql3



mysql4

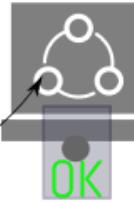


mysql5

certification
queue

GTID : Group UID + seqno
seqno = 1

certification



GTID: UUID:4

ORACLE®

Group Replication : Optimistic Locking

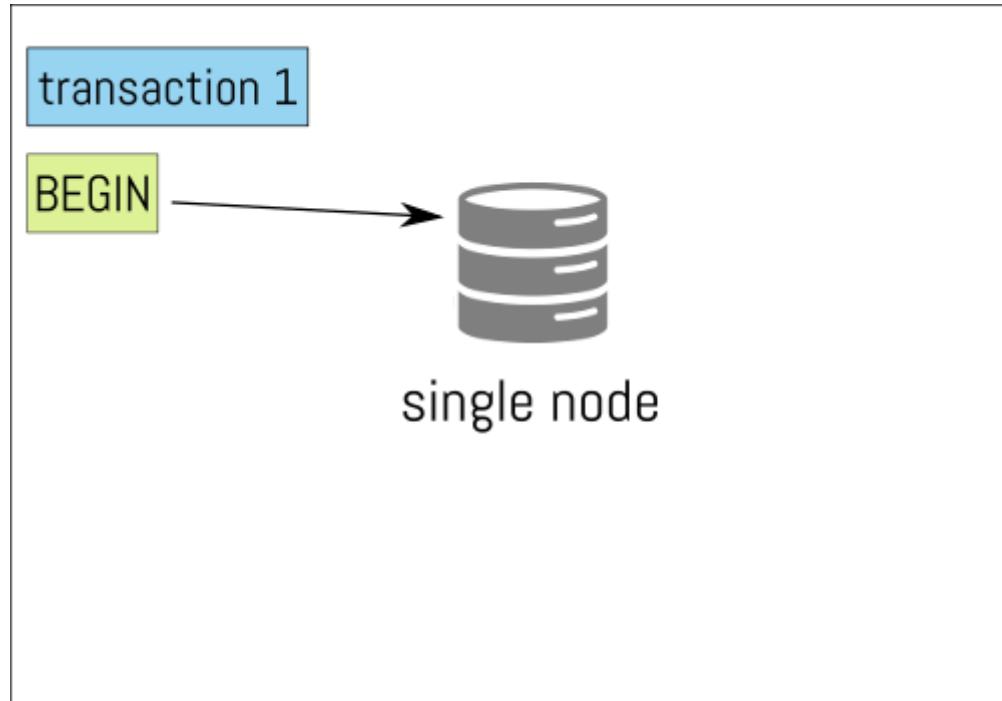
Group Replication uses optimistic locking

- during a transaction, **local (InnoDB) locking** happens
- **optimistically assumes** there will be no conflicts across nodes
(no communication between nodes necessary)
- cluster-wide conflict resolution happens only at COMMIT, during **certification**

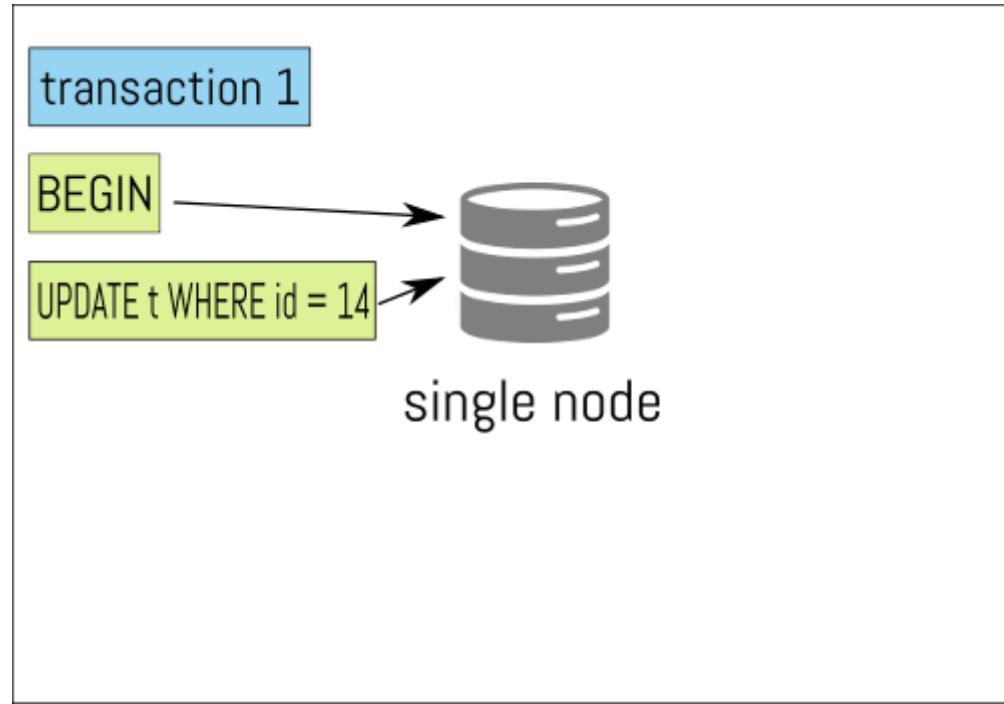
Let's first have a look at the traditional locking to compare.



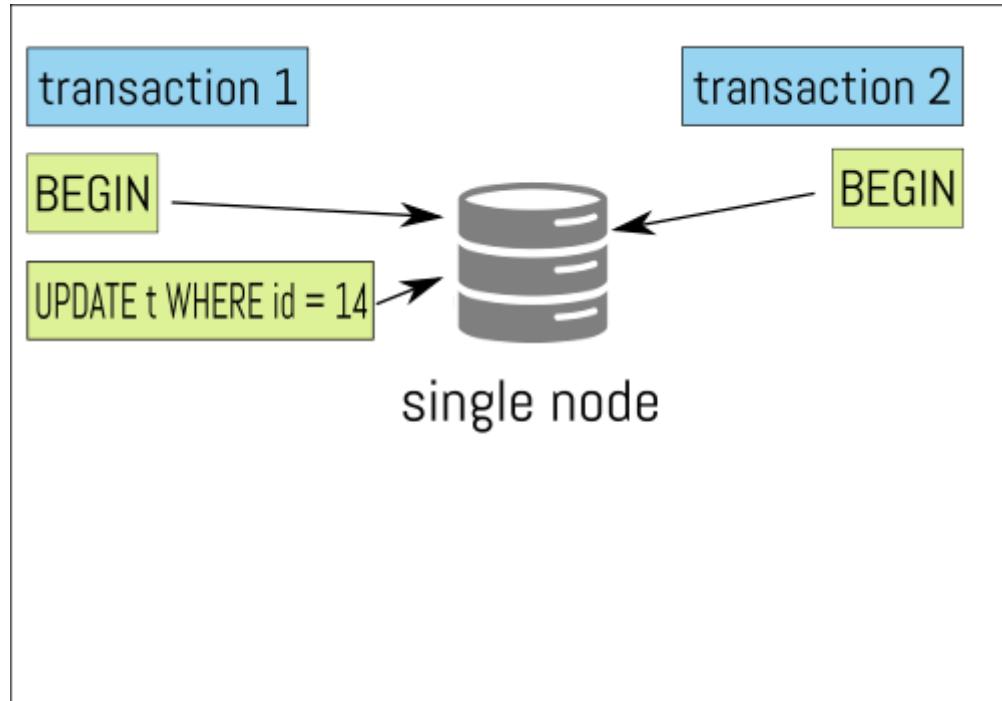
Traditional locking



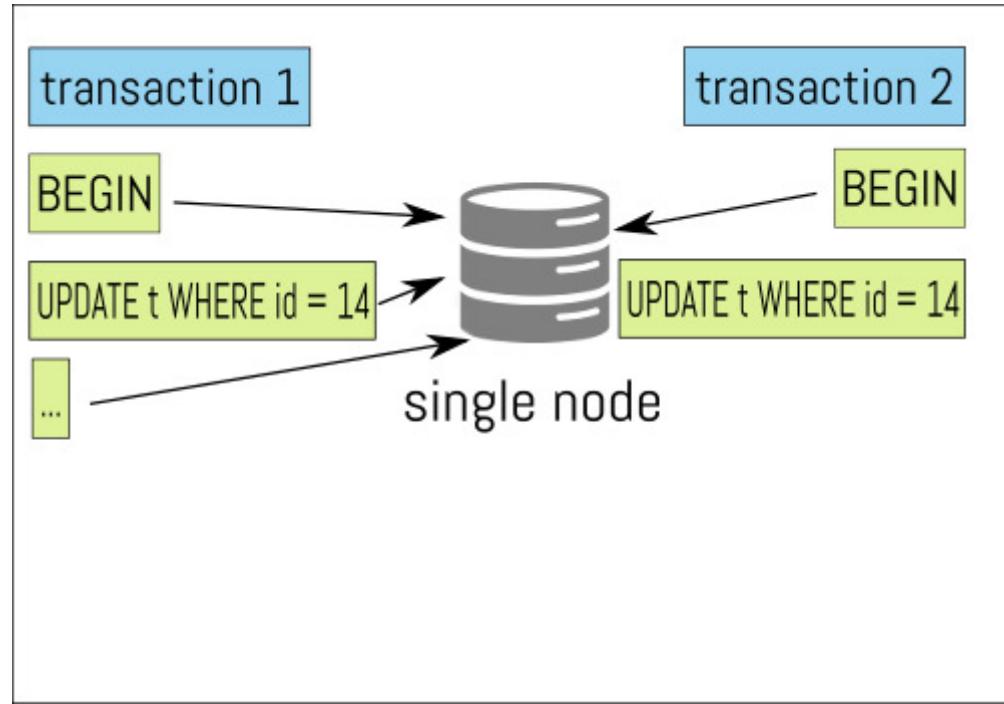
Traditional locking



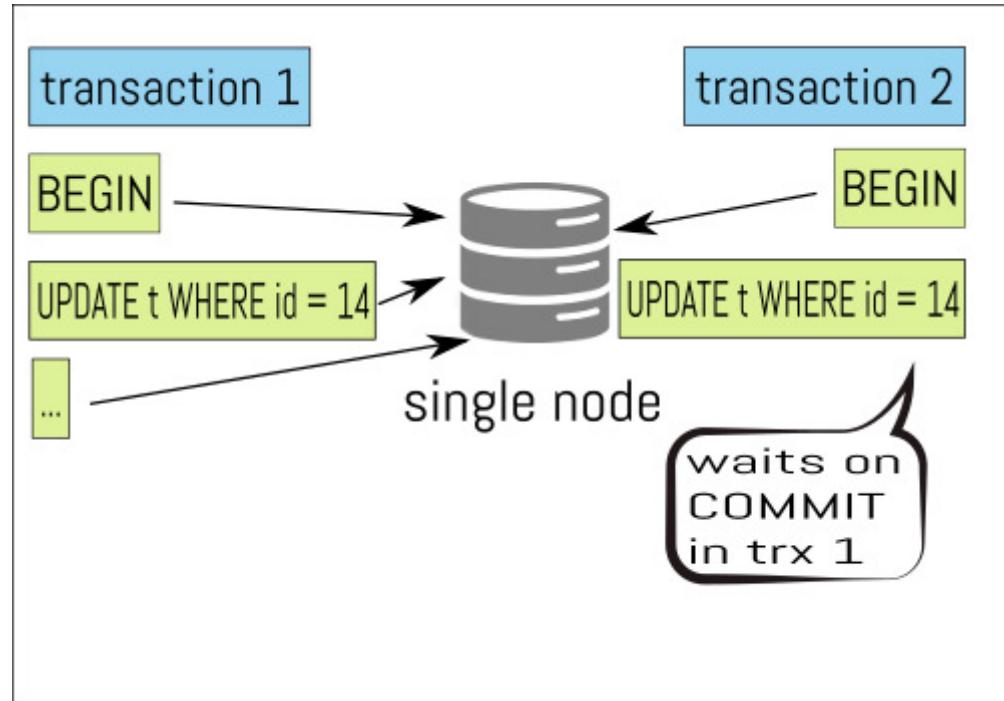
Traditional locking



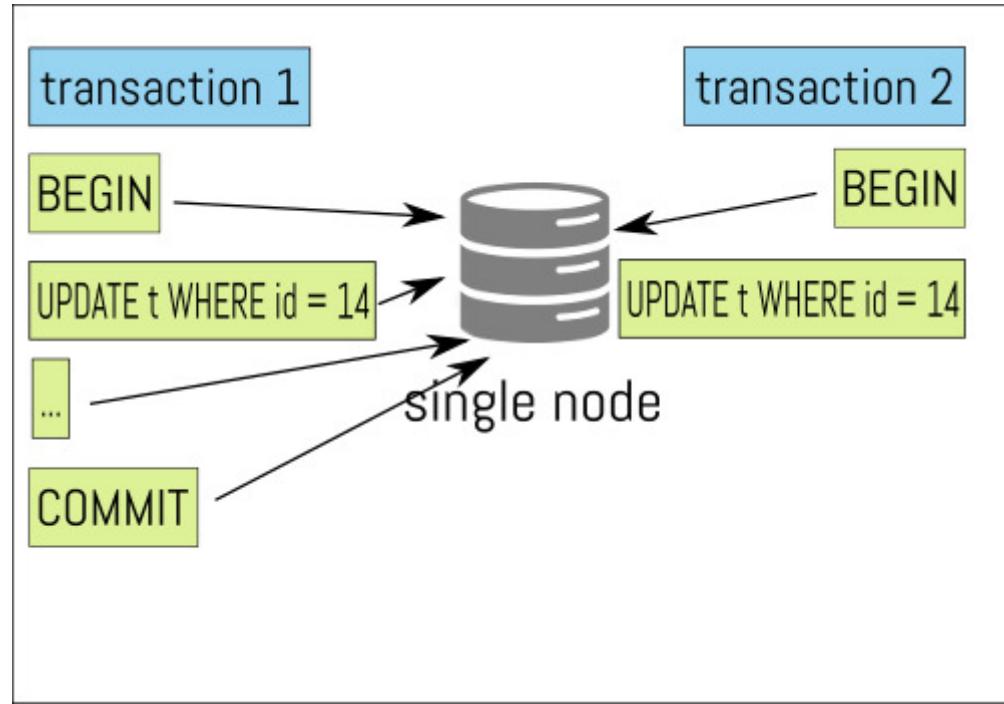
Traditional locking



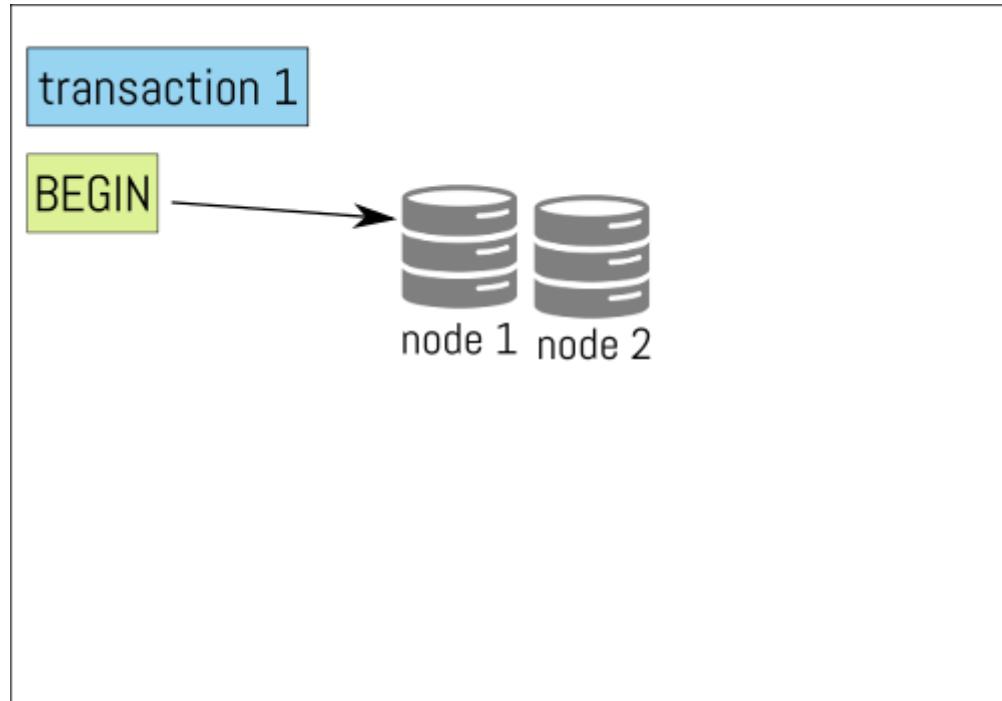
Traditional locking



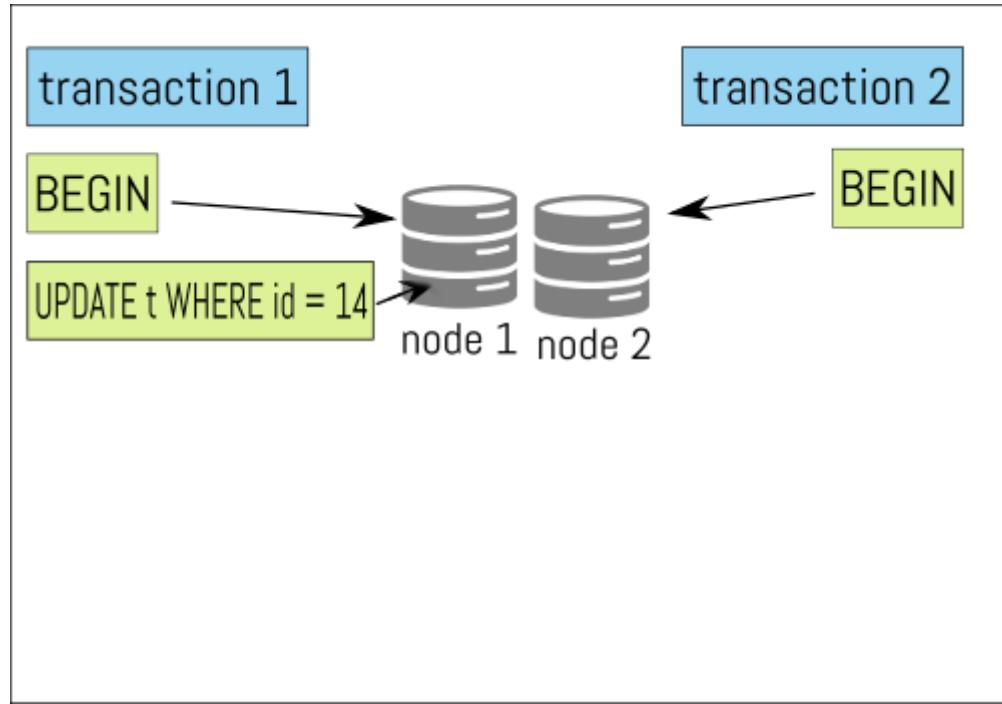
Traditional locking



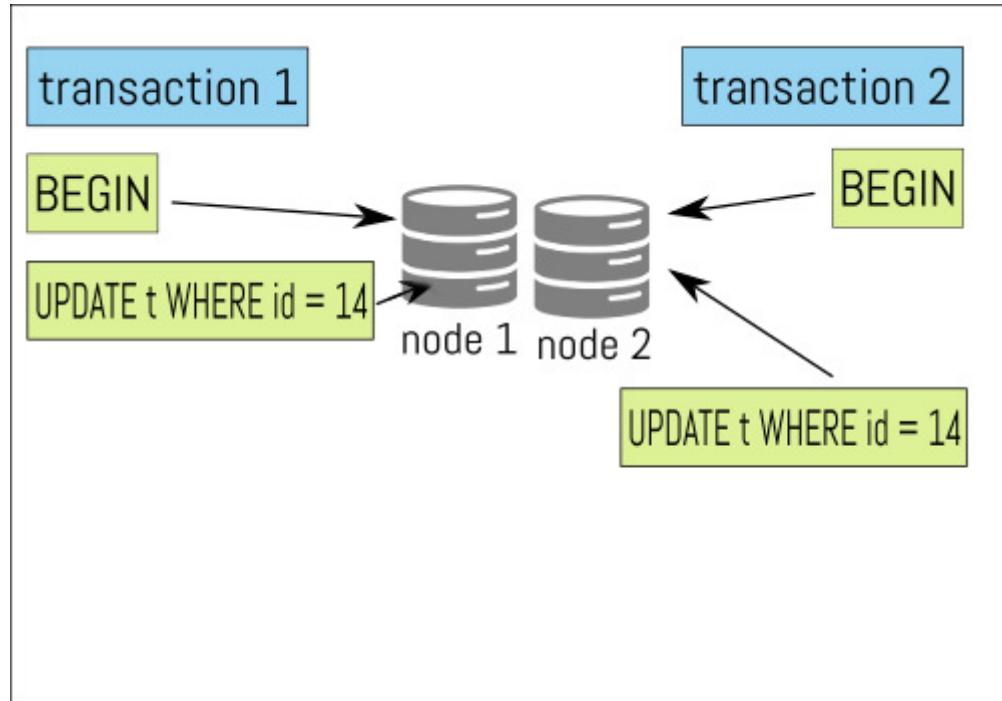
Optimistic Locking



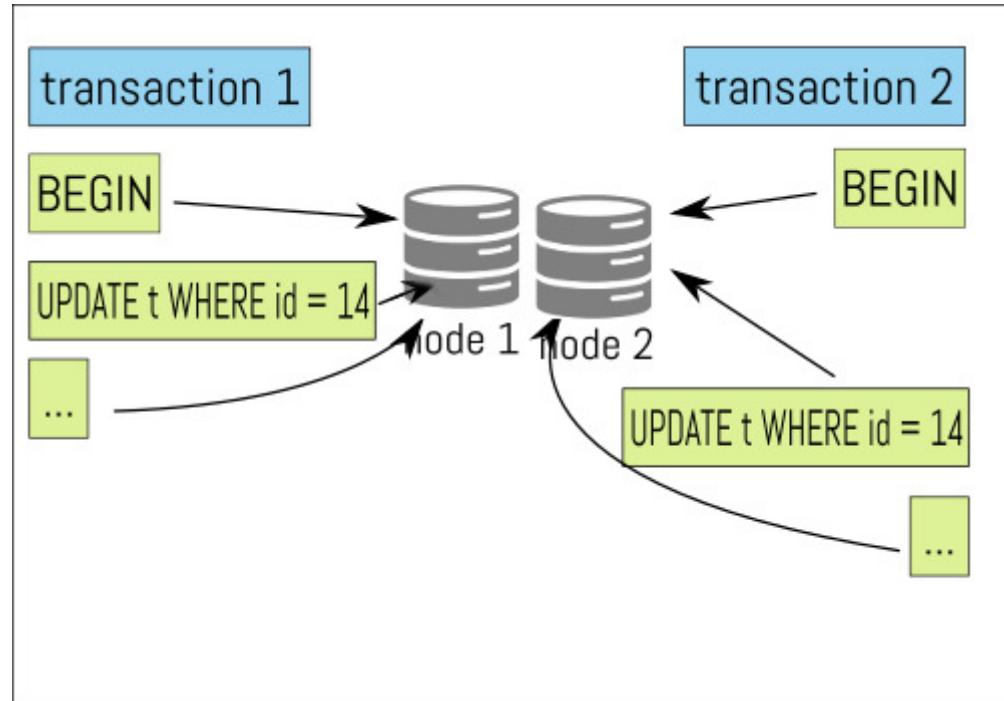
Optimistic Locking



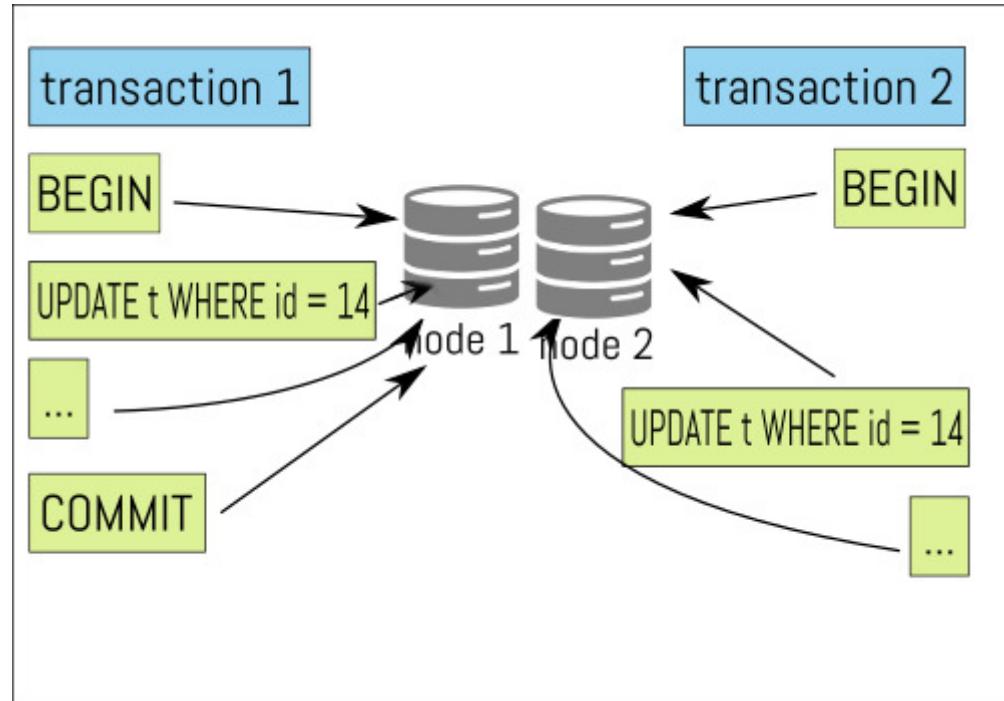
Optimistic Locking



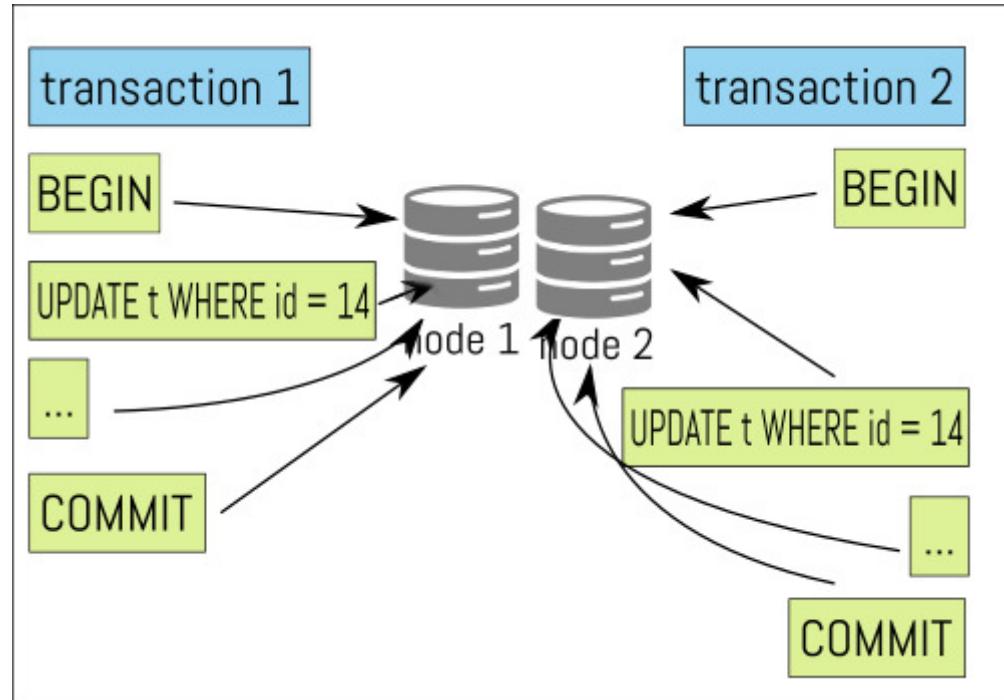
Optimistic Locking



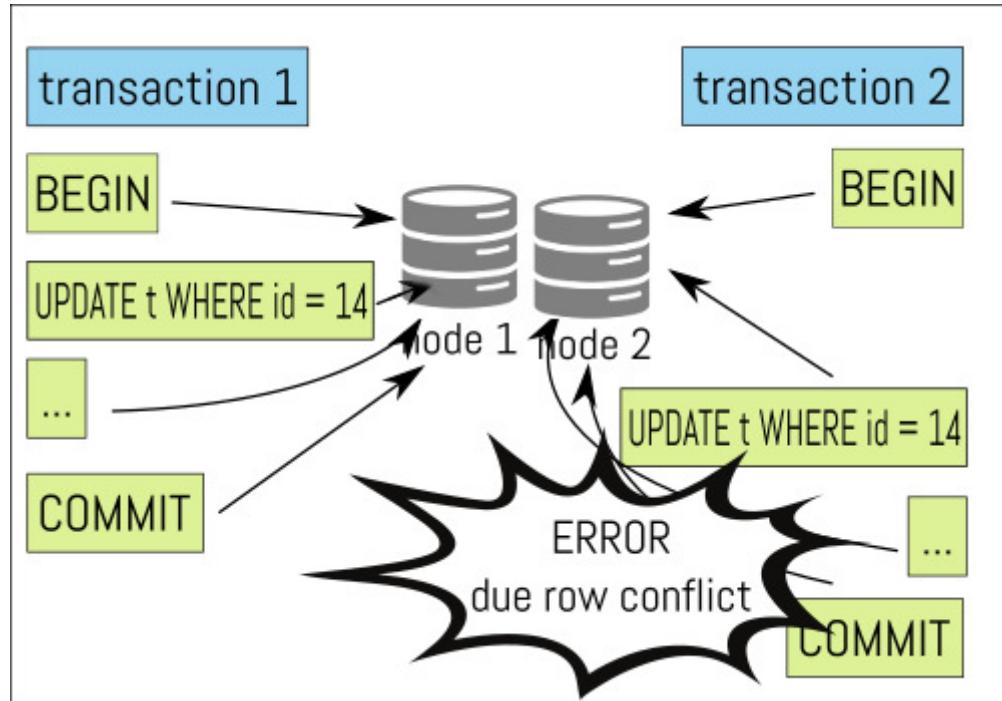
Optimistic Locking



Optimistic Locking



Optimistic Locking



The system returns error 149 as certification failed:

```
ERROR 1180 (HY000): Got error 149 during COMMIT
```

ORACLE®

Group Replication : requirements

- exclusively works with InnoDB tables only



Group Replication : requirements

- exclusively works with InnoDB tables only
- every tables must have a PK defined



Group Replication : requirements

- exclusively works with InnoDB tables only
- every tables must have a PK defined
- only IPV4 is supported



Group Replication : requirements

- exclusively works with InnoDB tables only
- every tables must have a PK defined
- only IPV4 is supported
- a good network with low latency is important



Group Replication : requirements

- exclusively works with InnoDB tables only
- every tables must have a PK defined
- only IPV4 is supported
- a good network with low latency is important
- maximum of 9 members per group



Group Replication : requirements

- exclusively works with InnoDB tables only
- every tables must have a PK defined
- only IPV4 is supported
- a good network with low latency is important
- maximum of 9 members per group
- log-bin must be enabled and only **ROW** format is supported



Group Replication : requirements (2)

- enable GTIDs

Group Replication : requirements (2)

- enable GTIDs
- replication meta-data must be stored on system tables

```
--master-info-repository=TABLE  
--relay-log-info-repository=TABLE
```

Group Replication : requirements (2)

- enable GTIDs
- replication meta-data must be stored on system tables

```
--master-info-repository=TABLE  
--relay-log-info-repository=TABLE
```

- writesets extraction must be enabled

```
--transaction-write-set-extraction=XXHASH64
```

Group Replication : requirements (2)

- enable GTIDs
- replication meta-data must be stored on system tables

```
--master-info-repository=TABLE  
--relay-log-info-repository=TABLE
```

- writesets extraction must be enabled

```
--transaction-write-set-extraction=XXHASH64
```

- log-slave-updates must be enabled

Group Replication : limitations

There are also some technical limitations:

Group Replication : limitations

There are also some technical limitations:

- binlog checksum is not supported

```
--binlog-checksum=NONE
```

Group Replication : limitations

There are also some technical limitations:

- binlog checksum is not supported

--binlog-checksum=NONE

- Savepoints are not supported

Group Replication : limitations

There are also some technical limitations:

- binlog checksum is not supported
--binlog-checksum=NONE
- Savepoints are not supported
- *SERIALIZABLE* is not supported as transaction isolation level

Group Replication : limitations

There are also some technical limitations:

- binlog checksum is not supported

--binlog-checksum=NONE

- Savepoints are not supported
- *SERIALIZABLE* is not supported as transaction isolation level
- an object cannot be changed concurrently at different servers by two operations, where one of them is a **DDL** and the other is either a **DML** or **DDL**.

Is my workload ready for Group Replication ?

As the writesets (transactions) are replicated to all available nodes on commit, and as they are certified on every node, a very large writeset could increase the amount of certification errors.

Additionally, changing the same record on all the nodes (hotspot) concurrently will also cause problems.

And finally, the certification uses the primary key of the tables, a table without PK is also a problem.

Is my workload ready for Group Replication ?

Therefore, when using Group Replication, we should pay attention to these points:

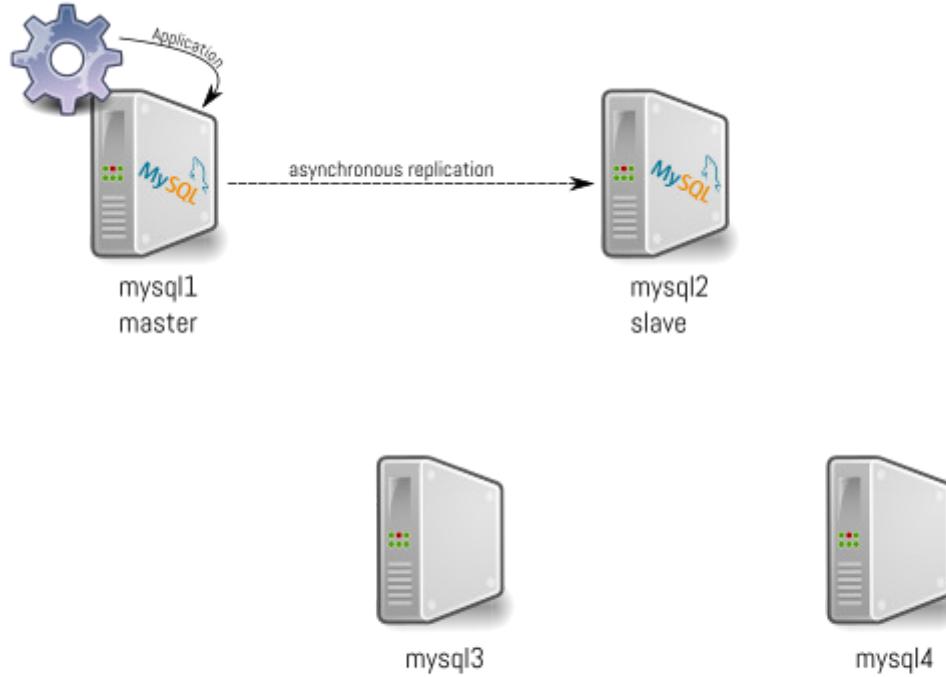
- PK is mandatory (and a good one is better)
- avoid large transactions
- avoid hotspot

ready?

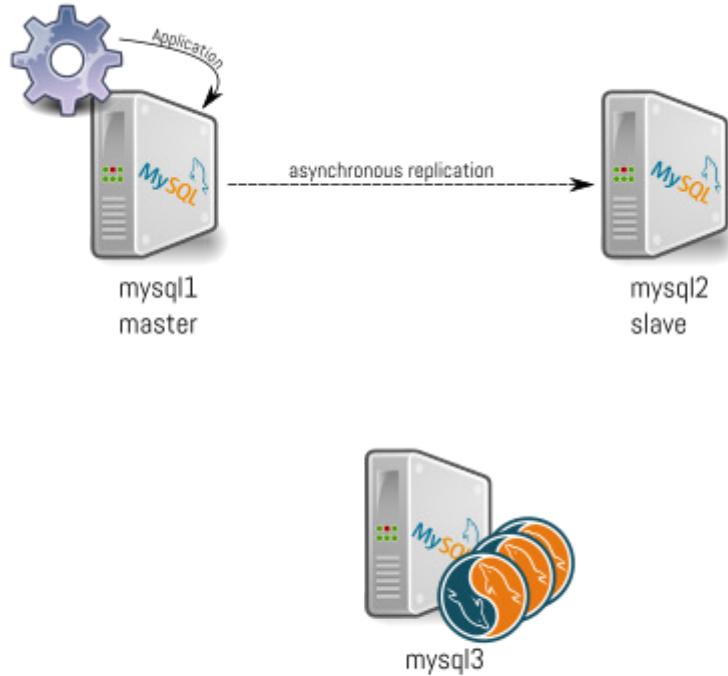
Migration from Master-Slave to GR



The plan

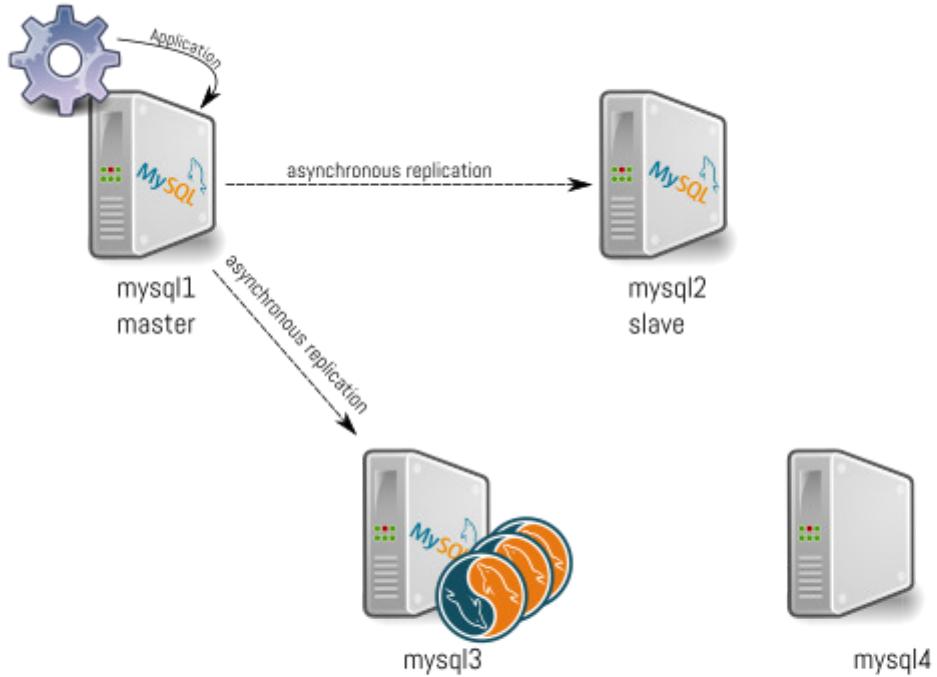


The plan



1) We install and setup **MySQL InnoDB Cluster** on one of the new servers

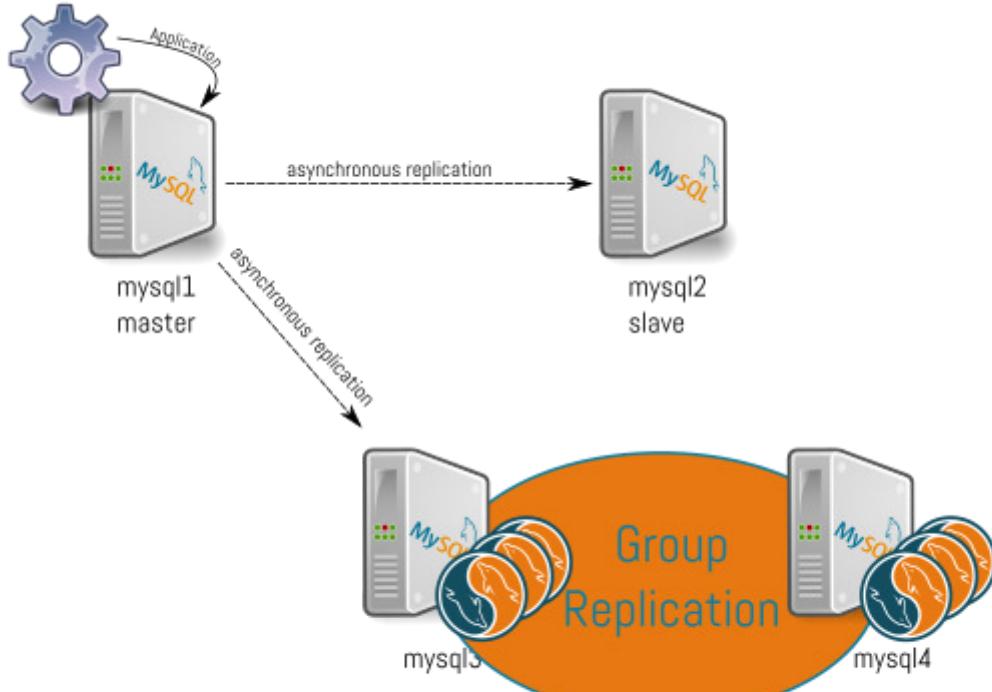
The plan



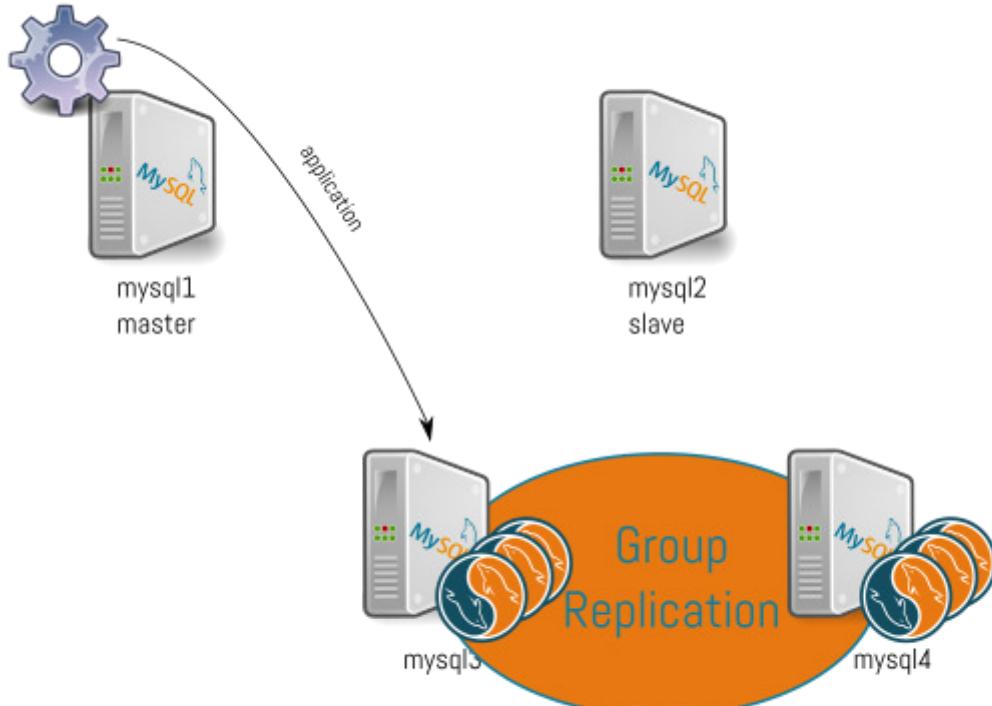
- 2) We restore a backup
- 3) setup asynchronous replication on the new server.

The plan

4) We add a new instance to our group

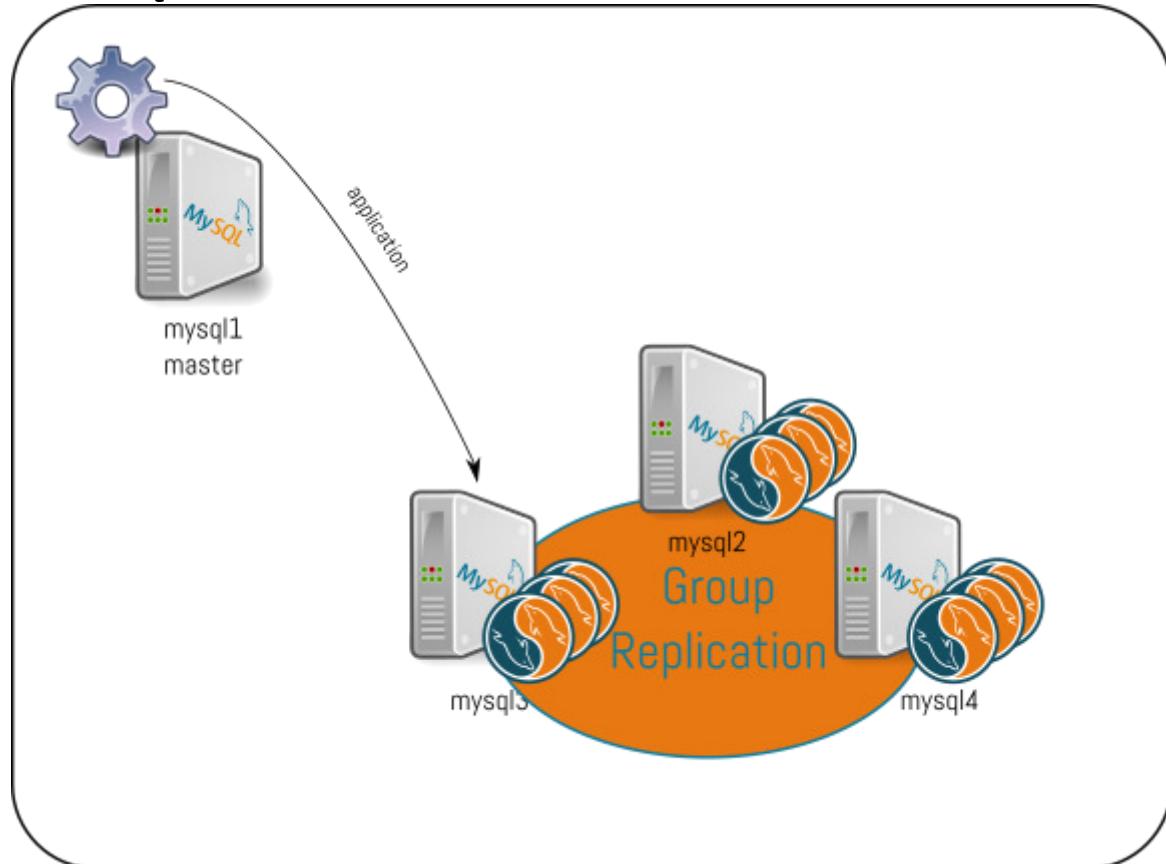


The plan



- 5) We point the application to one of our new nodes.
- 6) We wait and check that asynchronous replication is caught up
- 7) we stop those asynchronous slaves

The plan

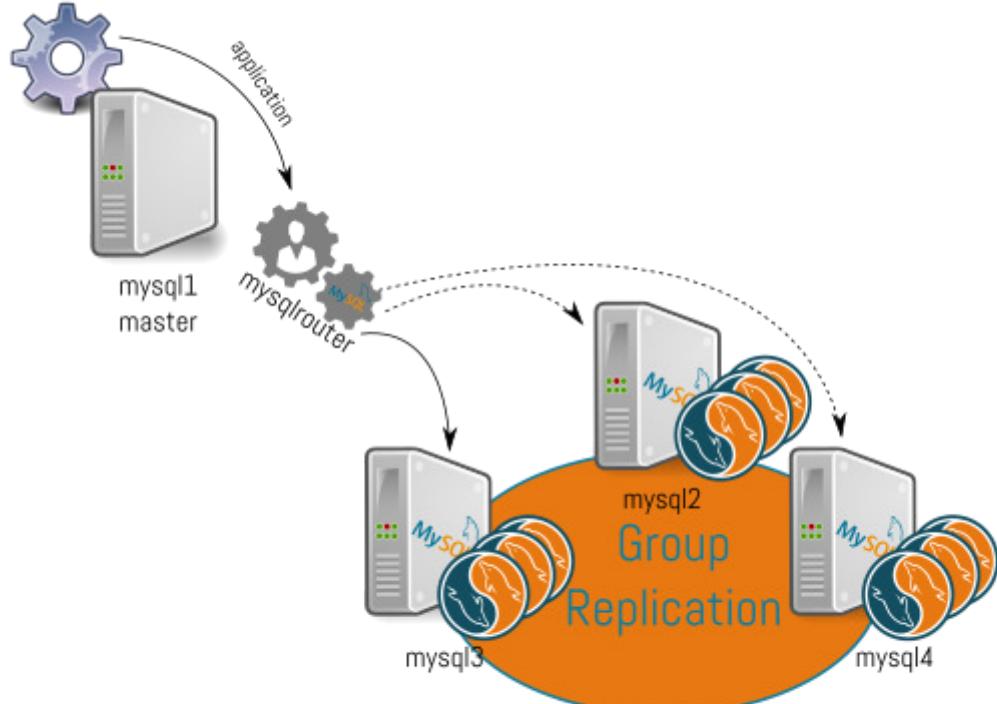


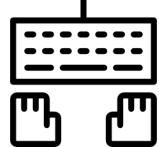
8) We attach the
mysql2 slave to the
group



The plan

9) Use MySQL Router for directing traffic





LAB2: Prepare mysql3

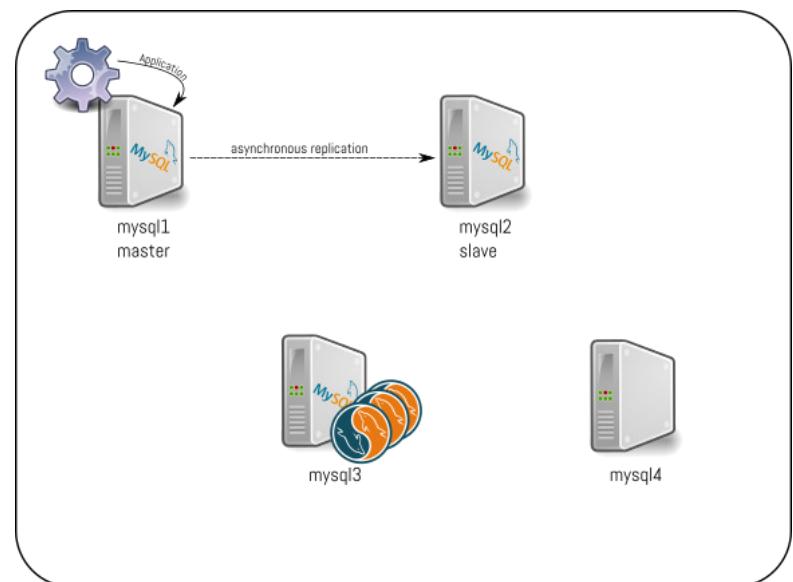
Asynchronous slave

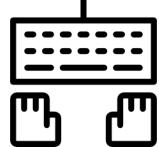
MySQL InnoDB Cluster from Labs is already installed on mysql3.

Let's take a backup on mysql1:

```
[mysql1 ~]# xtrabackup --backup \
--target-dir=/tmp/backup \
--user=root \
--password=X --host=127.0.0.1
```

```
[mysql1 ~]# xtrabackup --prepare \
--target-dir=/tmp/backup
```





LAB2: Prepare mysql3 (2)

Asynchronous slave

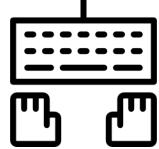
Copy the backup from mysql1 to mysql3:

```
[mysql1 ~]# scp -r /tmp/backup mysql3:/tmp
```

And restore it:

```
[mysql3 ~]# xtrabackup --copy-back --target-dir=/tmp/backup  
[mysql3 ~]# chown -R mysql. /var/lib/mysql
```





LAB3: mysql3 as asynchronous slave (2)

Asynchronous slave

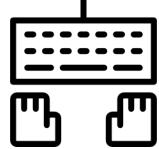
Configure `/etc/my.cnf`:

```
[mysqld]
...
server_id=3
enforce_gtid_consistency = on
gtid_mode = on
log_bin
log_slave_updates
```

ORACLE®

Copyright @ 2016 Oracle and/or its affiliates. All rights reserved.



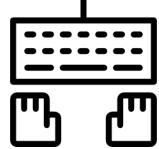


LAB2: Prepare mysql3 (3)

Asynchronous slave

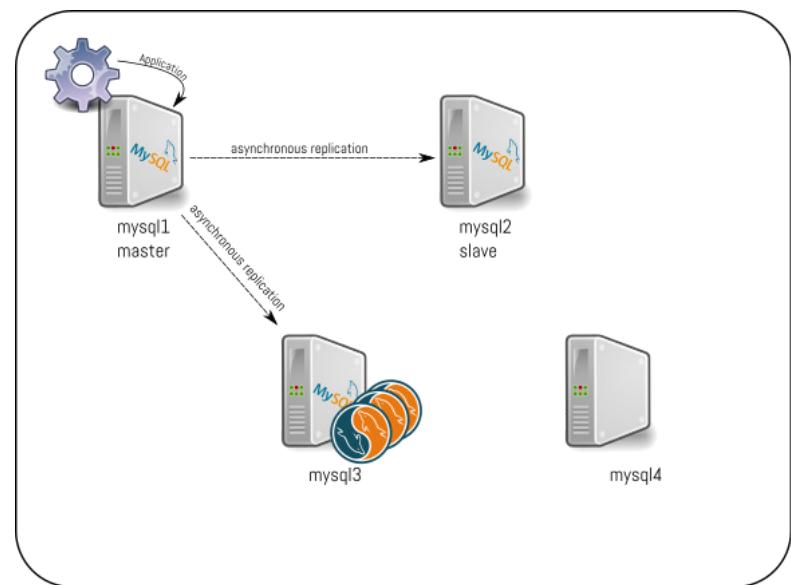
Let's start MySQL on mysql3:

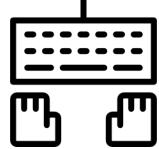
```
[mysql3 ~]# systemctl start mysqld
```



LAB3: mysql3 as asynchronous slave (1)

- find the GTIDs purged
- change MASTER
- set the purged GTIDs
- start replication





LAB3: mysql3 as asynchronous slave (2)

Find the latest purged GTIDs:

```
[mysql3 ~]# cat /tmp/backup/xtrabackup_binlog_info
mysql-bin.000002  167646328  b346474c-8601-11e6-9b39-08002718d305:1-771
```

Connect to mysql3 and setup replication:

```
mysql> CHANGE MASTER TO MASTER_HOST="mysql1",
      MASTER_USER="repl_async", MASTER_PASSWORD='Xslave',
      MASTER_AUTO_POSITION=1;

mysql> RESET MASTER;
mysql> SET global gtid_purged="VALUE FOUND PREVIOUSLY";

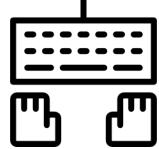
mysql> START SLAVE;
```

Check that you receive the application's traffic

Disclaimer !

The following commands to manage and use MySQL InnoDB Cluster are compatible with MySQL InnoDB Cluster 5.7.15 Preview from labs.

Functions may change in future releases.



LAB4: MySQL InnoDB Cluster

Create a single instance cluster

Time to use the new MySQL Shell !

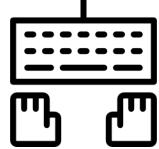
```
[mysql3 ~]# mysqlsh
```

Let's verify if our server is ready to become a member of a new cluster:

```
mysql-js> dba.validateInstance('root@mysql3:3306')
```



ORACLE®



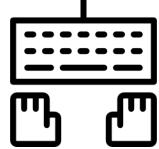
LAB4: configuration settings (2)

mysql3's my.cnf

Configure /etc/my.cnf:

```
[mysqld]
...
server_id=3
enforce_gtid_consistency = on
gtid_mode  = on
log_bin
log_slave_updates

binlog_checksum = none
master_info_repository = TABLE
relay_log_info_repository = TABLE
transaction_write_set_extraction = XXHASH64
```



LAB4: configuration settings (2)

mysql3's my.cnf

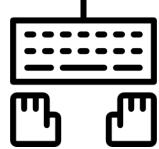
Configure /etc/my.cnf:

```
[mysqld]
...
server_id=3
enforce_gtid_consistency = on
gtid_mode  = on
log_bin
log_slave_updates

binlog_checksum = none
master_info_repository = TABLE
relay_log_info_repository = TABLE
transaction_write_set_extraction = XXHASH64
```

```
[mysql3 ~]# systemctl restart mysqld
```

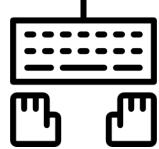
ORACLE®



LAB4: MySQL InnoDB Cluster (3)

Create a single instance cluster

```
[mysql ~]# mysqlsh
```



LAB4: MySQL InnoDB Cluster (3)

Create a single instance cluster

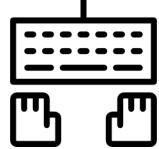
```
[mysql ~]# mysqlsh
```

```
mysql-js> dba.validateInstance('root@mysql3:3306')

mysql-js> \c root@mysql3:3306

mysql-js> cluster = dba.createCluster('plam')
```

The KEY is the only credential we need to remember to manage our cluster.



LAB4: MySQL InnoDB Cluster (3)

Create a single instance cluster

```
[mysql ~]# mysqlsh
```

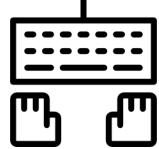
```
mysql-js> dba.validateInstance('root@mysql3:3306')

mysql-js> \c root@mysql3:3306

mysql-js> cluster = dba.createCluster('plam')
```

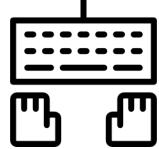
The KEY is the only credential we need to remember to manage our cluster.

```
mysql-js> cluster.status()
```



Cluster Status

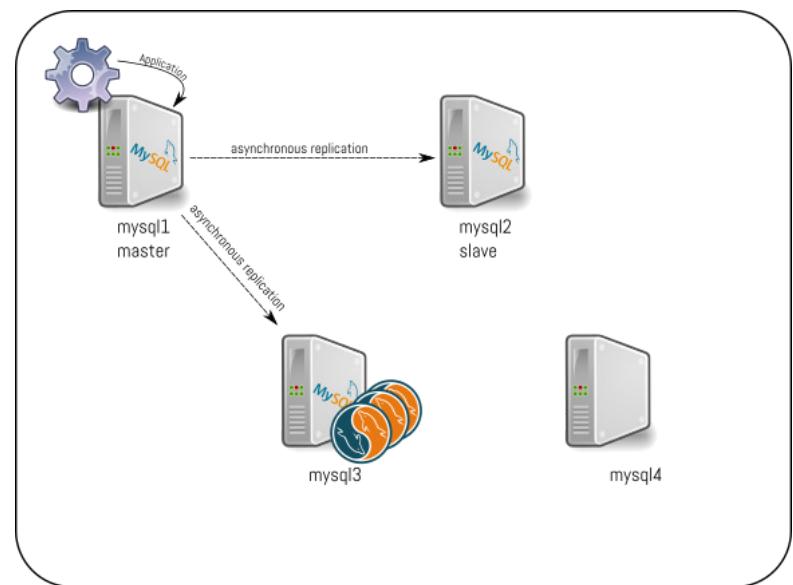
```
mysql-js> cluster.status()
{
  "clusterName": "plam",
  "defaultReplicaSet": {
    "status": "Cluster is NOT tolerant to any failures.",
    "topology": {
      "mysql3:3306": {
        "address": "mysql3:3306",
        "status": "ONLINE",
        "role": "HA",
        "mode": "R/W",
        "leaves": {}
      }
    }
  }
}
```

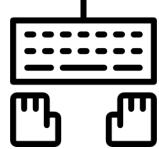


LAB5: add mysql4 to the cluster (1)

Add mysql4 to the Group:

- restore the backup
- set the purged GTIDs
- use MySQL shell





LAB5: add mysql4 to the cluster (2)

Copy the backup from mysql1 to mysql4:

```
[mysql1 ~]# scp -r /tmp/backup mysql4:/tmp
```

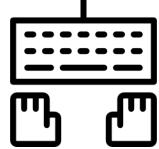
And restore it:

```
[mysql4 ~]# xtrabackup --copy-back --target-dir=/tmp/backup  
[mysql4 ~]# chown -R mysql. /var/lib/mysql
```

Start MySQL on mysql4:

```
[mysql4 ~]# systemctl start mysqld
```



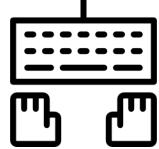


LAB5: MySQL shell to add an instance (3)

```
[mysql4 ~]# mysqlsh
```

Let's verify the config:

```
mysql> dba.validateInstance('root@mysql4:3306')
```

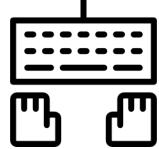


LAB5: MySQL shell to add an instance (4)

Configure /etc/my.cnf:

```
[mysqld]
...
server_id=4
enforce_gtid_consistency = on
gtid_mode    = on
log_bin
log_slave_updates

binlog_checksum = none
master_info_repository = TABLE
relay_log_info_repository = TABLE
transaction_write_set_extraction = XXHASH64
```



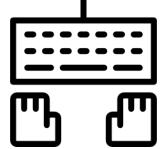
LAB5: MySQL shell to add an instance (4)

Configure /etc/my.cnf:

```
[mysqld]
...
server_id=4
enforce_gtid_consistency = on
gtid_mode = on
log_bin
log_slave_updates

binlog_checksum = none
master_info_repository = TABLE
relay_log_info_repository = TABLE
transaction_write_set_extraction = XXHASH64
```

```
[mysql4 ~]# systemctl restart mysqld
```



LAB5: MySQL InnoDB Cluster (4)

Group of 2 instances

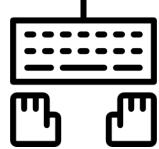
Find the latest purged GTIDs:

```
[mysql4 ~]# cat /tmp/backup/xtrabackup_binlog_info
mysql-bin.000002      167646328      b346474c-8601-11e6-9b39-08002718d305:1-77177
```

Connect to mysql4 and set GTID_PURGED

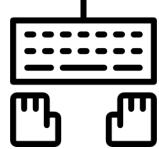
```
[mysql4 ~]# mysqlsh
```

```
mysql-js> \c root@mysql4:3306
mysql-js> \sql
mysql-sql> RESET MASTER;
mysql-sql> SET global gtid_purged="VALUE FOUND PREVIOUSLY";
```



LAB5: MySQL InnoDB Cluster (5)

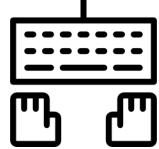
```
mysql> \js  
mysql> dba.validateInstance('root@mysql4:3306')  
mysql> \c root@mysql3:3306  
mysql> cluster = dba.getCluster('plam')  
mysql> cluster.addInstance("root@mysql4:3306")  
mysql> cluster.status()
```



Cluster Status

```
mysql-js> cluster.status()
{
  "clusterName": "plam",
  "defaultReplicaSet": {
    "status": "Cluster is NOT tolerant to any failures.",
    "topology": {
      "mysql3:3306": {
        "address": "mysql3:3306",
        "status": "ONLINE",
        "role": "HA",
        "mode": "R/W",
        "leaves": {
          "mysql4:3306": {
            "address": "mysql4:3306",
            "status": "RECOVERING",
            "role": "HA",
            "mode": "R/O",
            "leaves": {}
          }
        }
      }
    }
  }
}
```

ORACLE®

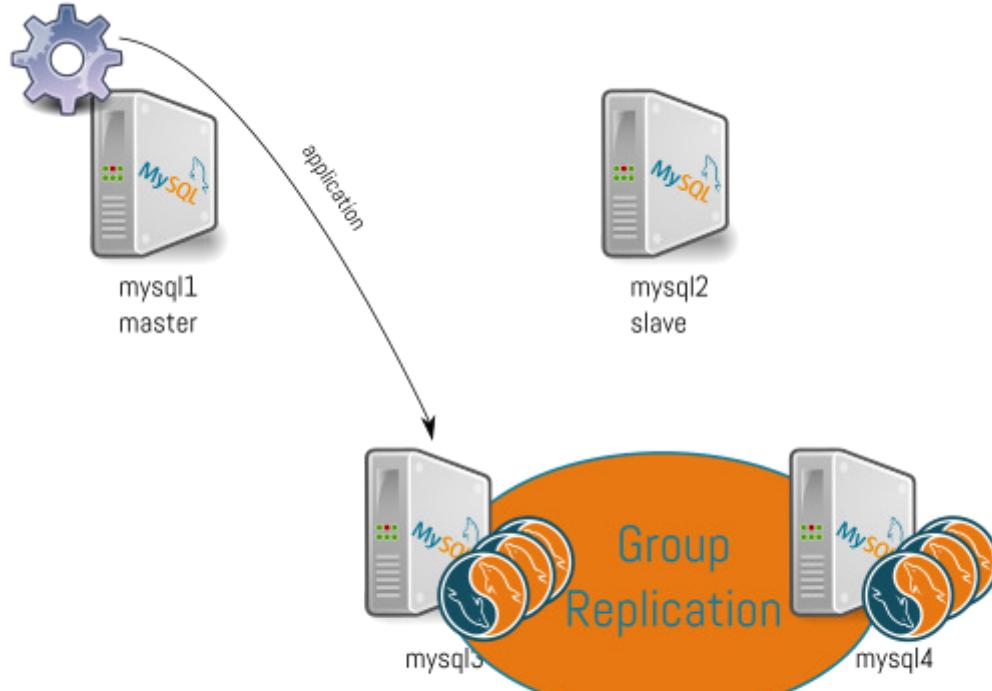


Recovering progress

On standard MySQL, monitor the group_replication_recovery channel to see the progress:

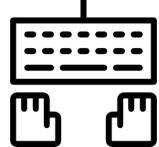
```
mysql> show slave status for channel 'group_replication_recovery'\G
***** 1. row *****
    Slave_IO_State: Waiting for master to send event
      Master_Host: mysql3
      Master_User: mysql_innodb_cluster_rpl_user
      ...
      Slave_IO_Running: Yes
      Slave_SQL_Running: Yes
      ...
      Retrieved_Gtid_Set: 6e7d7848-860f-11e6-92e4-08002718d305:1-6,
7c1f0c2d-860d-11e6-9df7-08002718d305:1-15,
b346474c-8601-11e6-9b39-08002718d305:1964-77177,
e8c524df-860d-11e6-9df7-08002718d305:1-2
      Executed_Gtid_Set: 7c1f0c2d-860d-11e6-9df7-08002718d305:1-7,
b346474c-8601-11e6-9b39-08002718d305:1-45408,
e8c524df-860d-11e6-9df7-08002718d305:1-2
      ...
```

Migrate the application



point the application
to the cluster

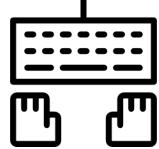




LAB6: Migrate the application

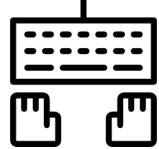
Now we need to point the application to mysql3, this is the only downtime !

```
[ 21257s] threads: 4, tps: 12.00, reads: 167.94, writes: 47.98, response time: ...
[ 21258s] threads: 4, tps: 6.00,  reads: 83.96, writes: 23.99, response time: ...
[ 21259s] threads: 4, tps: 7.00,  reads: 98.05, writes: 28.01, response time: ...
[ 31250s] threads: 4, tps: 8.00,  reads: 111.95, writes: 31.99, response time: ...
[ 31251s] threads: 4, tps: 11.00, reads: 154.01, writes: 44.00, response time: ...
[ 31252s] threads: 4, tps: 11.00, reads: 153.94, writes: 43.98, response time: ...
[ 31253s] threads: 4, tps: 10.01, reads: 140.07, writes: 40.02, response time: ...
^C
[mysql1 ~]# run_app.sh mysql3
```



LAB6: Migrate the application

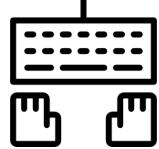
Make sure replication is running properly on mysql2 and mysql3.



LAB6: Migrate the application

Make sure replication is running properly on mysql2 and mysql3.

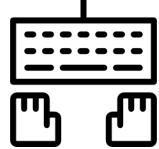
```
***** 1. row *****
...
    Master_Host: mysql1
    Slave_IO_Running: Yes
    Slave_SQL_Running: No
...
    Last_SQL_Error: Error in Xid_log_event: Commit could not be completed
    'Error on observer while running replication hook 'before_commit'.'
Replicate_Ignore_Server_Ids:
    Master_Server_Id: 1
    Master_UUID: 8fbcd944-8760-11e6-9b4e-08002718d305
...
    Last_SQL_Error_Timestamp: 160930 23:04:07
    Retrieved_Gtid_Set: 8fbcd944-8760-11e6-9b4e-08002718d305:360-3704
    Executed_Gtid_Set: 0b31b4f3-8762-11e6-bb35-08002718d305:1-7,
    30212757-8762-11e6-ad73-08002718d305:1-2,
    8fbcd944-8760-11e6-9b4e-08002718d305:1-2652
    Auto_Position: 1
```



LAB6: Migrate the application

Fix replication if necessary:

```
mysql> start slave;  
mysql> show slave status;
```



LAB6: Migrate the application

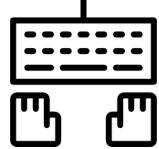
Fix replication if necessary:

```
mysql3> start slave;
mysql3> show slave status;
```

Stop asynchronous replication on mysql2 and mysql3:

```
mysql2> stop slave;
mysql3> stop slave;
```

Make sure gtid_executed range on mysql2 is lower or equal than on mysql3



LAB6: Migrate the application

Fix replication if necessary:

```
mysql3> start slave;
mysql3> show slave status;
```

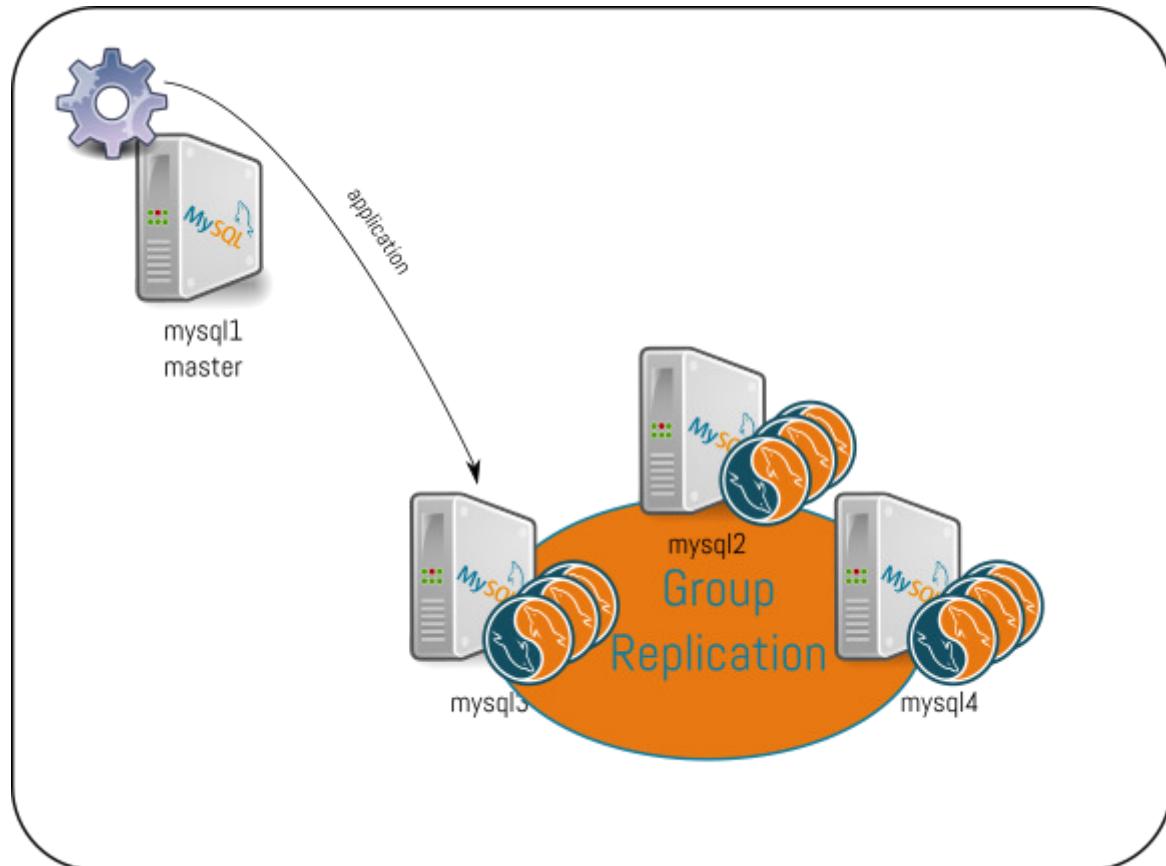
Stop asynchronous replication on mysql2 and mysql3:

```
mysql2> stop slave;
mysql3> stop slave;
```

Make sure gtid_executed range on mysql2 is lower or equal than on mysql3

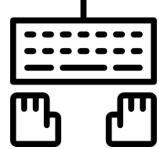
```
mysql2> set global super_read_only=off;# http://bugs.mysql.com/bug.php?id=83234
mysql2> reset slave all;
mysql3> reset slave all;
```

Add a third instance



previous slave
(mysql2) can now
be part of the cluster

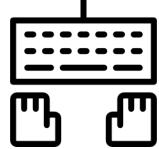




LAB7: Add mysql2 to the group

We first validate the instance using **MySQL** shell and we modify the configuration.



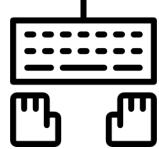


LAB7: Add mysql2 to the group

We first validate the instance using MySQL shell and we modify the configuration.

```
[mysqld]
...
super_read_only=0
server_id=2
binlog_checksum = none
enforce_gtid_consistency = on
gtid_mode = on
log_bin
log_slave_updates
master_info_repository = TABLE
relay_log_info_repository = TABLE
transaction_write_set_extraction = XXHASH64
```





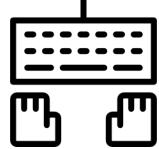
LAB7: Add mysql2 to the group

We first validate the instance using MySQL shell and we modify the configuration.

```
[mysqld]
...
super_read_only=0
server_id=2
binlog_checksum = none
enforce_gtid_consistency = on
gtid_mode = on
log_bin
log_slave_updates
master_info_repository = TABLE
relay_log_info_repository = TABLE
transaction_write_set_extraction = XXHASH64
```

```
[mysql ~]# systemctl restart mysqld
```

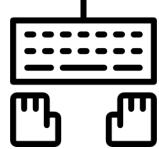




LAB7: Add mysql2 to the group (2)

Back in MySQL shell we add the new instance:

```
[mysql ~]# mysqlsh
```



LAB7: Add mysql2 to the group (2)

Back in MySQL shell we add the new instance:

```
[mysql ~]# mysqlsh
```

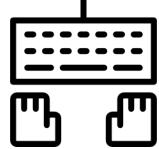
```
mysql-js> dba.validateInstance('root@mysql2:3306')

mysql-js> \c root@mysql3:3306

mysql-js> cluster = dba.getCluster('plam')

mysql-js> cluster.addInstance("root@mysql2:3306")

mysql-js> cluster.status()
```



LAB7: Add mysql2 to the group (3)

```
{  
  "clusterName": "plam",  
  "defaultReplicaSet": {  
    "status": "Cluster tolerant to up to ONE failure.",  
    "topology": {  
      "mysql3:3306": {  
        "address": "mysql3:3306",  
        "status": "ONLINE",  
        "role": "HA",  
        "mode": "R/W",  
        "leaves": {  
          "mysql4:3306": {  
            "address": "mysql4:3306",  
            "status": "ONLINE",  
            "role": "HA",  
            "mode": "R/O",  
            "leaves": {}  
          },  
          "mysql2:3306": {  
            "address": "mysql2:3306",  
            "status": "ONLINE",  
            "role": "HA",  
            "mode": "R/O",  
            "leaves": {}  
          }  
        }  
      }  
    }  
  }  
}
```

ORACLE®

WOOHOOOW

IT WORKS!

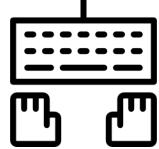
writing to a single server

Single Primary Mode



Default = Single Primary Mode

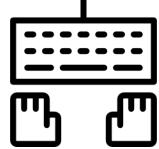
By default, MySQL InnoDB Cluster enables Single Primary Mode.



Default = Single Primary Mode

By default, MySQL InnoDB Cluster enables Single Primary Mode.

```
mysql> show global variables like 'group_replication_single_primary_mode';
+-----+-----+
| Variable_name          | Value   |
+-----+-----+
| group_replication_single_primary_mode | ON      |
+-----+-----+
```



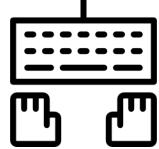
Default = Single Primary Mode

By default, **MySQL InnoDB Cluster** enables Single Primary Mode.

```
mysql> show global variables like 'group_replication_single_primary_mode';
+-----+-----+
| Variable_name          | Value   |
+-----+-----+
| group_replication_single_primary_mode | ON      |
+-----+-----+
```

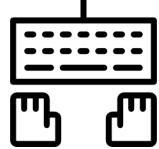
In Single Primary Mode, a single member acts as the writable master (**PRIMARY**) and the rest of the members act as hot-standbys (**SECONDARY**).

The group itself coordinates and configures itself automatically to determine which member will act as the **PRIMARY**, through a leader election mechanism.



Who's the Primary Master ?

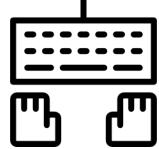
As the Primary Master is elected, all nodes part of the group knows which one was elected. This value is exposed in status variables:



Who's the Primary Master ?

As the Primary Master is elected, all nodes part of the group knows which one was elected. This value is exposed in status variables:

```
mysql> show status like 'group_replication_primary_member';
+-----+-----+
| Variable_name          | Value           |
+-----+-----+
| group_replication_primary_member | 28a4e51f-860e-11e6-bdc4-08002718d305 |
+-----+-----+
```



Who's the Primary Master ?

As the Primary Master is elected, all nodes part of the group knows which one was elected. This value is exposed in status variables:

```
mysql> show status like 'group_replication_primary_member';
+-----+-----+
| Variable_name          | Value           |
+-----+-----+
| group_replication_primary_member | 28a4e51f-860e-11e6-bdc4-08002718d305 |
+-----+-----+
```

```
mysql> select member_host as "primary master"
      from performance_schema.global_status
      join performance_schema.replication_group_members
    where variable_name = 'group_replication_primary_member'
      and member_id=variable_value;
+-----+
| primary master|
+-----+
| mysql3       |
+-----+
```

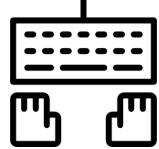
get more info

Monitoring



Copyright @ 2016 Oracle and/or its affiliates. All rights reserved.

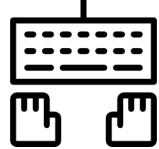




Performance Schema

Group Replication uses Performance_Schema to expose status

```
mysql3> SELECT * FROM performance_schema.replication_group_members\G
***** 1. row ****
CHANNEL_NAME: group_replication_applier
 MEMBER_ID: 00db47c7-3e23-11e6-afd4-08002774c31b
 MEMBER_HOST: mysql3.localdomain
 MEMBER_PORT: 3306
 MEMBER_STATE: ONLINE
```



Performance Schema

Group Replication uses Performance_Schema to expose status

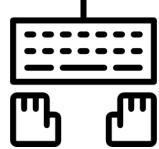
```
mysql3> SELECT * FROM performance_schema.replication_group_members\G
***** 1. row ****
CHANNEL_NAME: group_replication_applier
    MEMBER_ID: 00db47c7-3e23-11e6-af4-08002774c31b
    MEMBER_HOST: mysql3.localdomain
    MEMBER_PORT: 3306
    MEMBER_STATE: ONLINE
```

```
mysql3> SELECT * FROM performance_schema.replication_connection_status\G
***** 1. row ****
CHANNEL_NAME: group_replication_applier
    GROUP_NAME: afb80f36-2bff-11e6-84e0-0800277dd3bf
    SOURCE_UUID: afb80f36-2bff-11e6-84e0-0800277dd3bf
    THREAD_ID: NULL
    SERVICE_STATE: ON
COUNT_RECEIVED_HEARTBEATS: 0
LAST_HEARTBEAT_TIMESTAMP: 0000-00-00 00:00:00
RECEIVED_TRANSACTION_SET: afb80f36-2bff-11e6-84e0-0800277dd3bf:1-2
    LAST_ERROR_NUMBER: 0
MESSAGE:
LAST_TIMESTAMP: 0000-00-00 00:00:00
```

Member State

These are the different possible state for a node member:

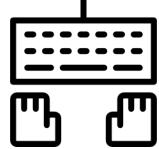
- ONLINE
- OFFLINE
- RECOVERING
- ERROR: when a node is leaving but the plugin was not instructed to stop
- UNREACHABLE



Status information & metrics

Members

```
mysql> SELECT * FROM performance_schema.replication_group_members\G
```



Status information & metrics

Members

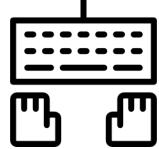
```
mysql> SELECT * FROM performance_schema.replication_group_members\G
```

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

```
CHANNEL_NAME: group_replication_applier
  MEMBER_ID: 00db47c7-3e23-11e6-af4-08002774c31b
  MEMBER_HOST: mysql3.localdomain
  MEMBER_PORT: 3306
  MEMBER_STATE: ONLINE
```

```
***** 2. row *****
```

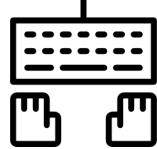
```
CHANNEL_NAME: group_replication_applier
  MEMBER_ID: e1544c9d-4451-11e6-9f5a-08002774c31b
  MEMBER_HOST: mysql4.localdomain.localdomain
  MEMBER_PORT: 3306
  MEMBER_STATE: ONLINE
```



Status information & metrics

Connections

```
mysql> SELECT * FROM performance_schema.replication_connection_status\G
```



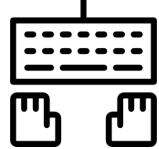
Status information & metrics

Connections

```
mysql> SELECT * FROM performance_schema.replication_connection_status\G
```

```
***** 1. row *****
CHANNEL_NAME: group_replication_applier
GROUP_NAME: afb80f36-2bff-11e6-84e0-0800277dd3bf
SOURCE_UUID: afb80f36-2bff-11e6-84e0-0800277dd3bf
THREAD_ID: NULL
SERVICE_STATE: ON
COUNT_RECEIVED_HEARTBEATS: 0
LAST_HEARTBEAT_TIMESTAMP: 0000-00-00 00:00:00
RECEIVED_TRANSACTION_SET: 5de4400b-3dd7-11e6-8a71-08002774c31b:1-814089,
                           afb80f36-2bff-11e6-84e0-0800277dd3bf:1-2834
LAST_ERROR_NUMBER: 0
LAST_ERROR_MESSAGE:
LAST_ERROR_TIMESTAMP: 0000-00-00 00:00:00
***** 2. row *****
CHANNEL_NAME: group_replication_recovery
GROUP_NAME:
SOURCE_UUID:
THREAD_ID: NULL
SERVICE_STATE: OFF
```

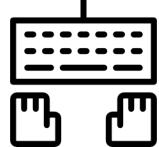
ORACLE®



Status information & metrics

Local node status

```
mysql> select * from performance_schema.replication_group_member_stats\G
```



Status information & metrics

Local node status

```
mysql> select * from performance_schema.replication_group_member_stats\G
```

```
***** 1. row *****
CHANNEL_NAME: group_replication_applier
VIEW_ID: 14679667214442885:4
MEMBER_ID: e1544c9d-4451-11e6-9f5a-08002774c31b
COUNT_TRANSACTIONS_IN_QUEUE: 0
COUNT_TRANSACTIONS_CHECKED: 5961
COUNT_CONFLICTS_DETECTED: 0
COUNT_TRANSACTIONS_ROWS_VALIDATING: 0
TRANSACTIONS_COMMITTED_ALL_MEMBERS: 5de4400b-3dd7-11e6-8a71-08002774c31b:1-81408
                                         afb80f36-2bff-11e6-84e0-0800277dd3bf:1-5718
LAST_CONFLICT_FREE_TRANSACTION: afb80f36-2bff-11e6-84e0-0800277dd3bf:5718
```



Performance_Schema

You can find GR information in the following **Performance_Schema** tables:

- replication_applier_configuration
- replication_applier_status
- replication_applier_status_by_worker
- replication_connection_configuration
- replication_connection_status
- replication_group_member_stats
- replication_group_members

Status during recovery

```
mysql> SHOW SLAVE STATUS FOR CHANNEL 'group_replication_recovery' \G
```

Status during recovery

```
mysql> SHOW SLAVE STATUS FOR CHANNEL 'group_replication_recovery' \G
```



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

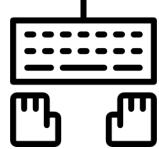
```
Slave_IO_State: 
Master_Host: <NULL>
Master_User: gr_repl
Master_Port: 0

...
Relay_Log_File: mysql4-relay-bin-group_replication_recovery.000001
...

Slave_IO_Running: No
Slave_SQL_Running: No

...
Executed_Gtid_Set: 5de4400b-3dd7-11e6-8a71-08002774c31b:1-814089,
afb80f36-2bff-11e6-84e0-0800277dd3bf:1-5718

...
Channel_Name: group_replication_recovery
```



Sys Schema

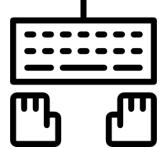
The easiest way to detect if a node is a member of the primary component (when there are partitioning of your nodes due to network issues for example) and therefore a valid candidate for routing queries to it, is to use the sys table.

Additional information for sys can be downloaded at

https://github.com/lefred/mysql_gr_routing_check/blob/master/addition_to_sys.sql

On the primary node:

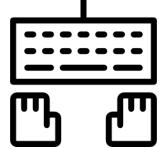
```
[mysql ~]# mysql < /tmp/gr_addition_to_sys.sql
```



Sys Schema

Is this node part of PRIMARY Partition:

```
mysql3> SELECT sys.gr_member_in_primary_partition();
+-----+
| sys.gr_node_in_primary_partition() |
+-----+
| YES
+-----+
```



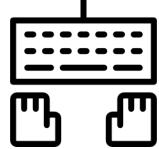
Sys Schema

Is this node part of PRIMARY Partition:

```
mysql3> SELECT sys.gr_member_in_primary_partition();
+-----+
| sys.gr_node_in_primary_partition() |
+-----+
| YES
+-----+
```

To use as healthcheck:

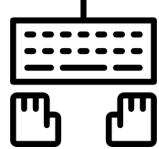
```
mysql3> SELECT * FROM sys.gr_member_routing_candidate_status;
+-----+-----+-----+-----+
| viable_candidate | read_only | transactions_behind | transactions_to_cert |
+-----+-----+-----+-----+
| YES            | YES      | 0              | 0
+-----+-----+-----+-----+
```



Sys Schema - Heath Check

On one of the non Primary nodes, run the following command:

```
mysql> flush tables with read lock;
```



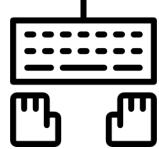
Sys Schema - Health Check

On one of the non Primary nodes, run the following command:

```
mysql> flush tables with read lock;
```

Now you can verify what the healthcheck exposes to you:

```
mysql> SELECT * FROM sys.gr_member_routing_candidate_status;
+-----+-----+-----+-----+
| viable_candidate | read_only | transactions_behind | transactions_to_cert |
+-----+-----+-----+-----+
| YES            | YES      | 950                | 0                  |
+-----+-----+-----+-----+
```



Sys Schema - Health Check

On one of the non Primary nodes, run the following command:

```
mysql> flush tables with read lock;
```

Now you can verify what the healthcheck exposes to you:

```
mysql> SELECT * FROM sys.gr_member_routing_candidate_status;
+-----+-----+-----+-----+
| viable_candidate | read_only | transactions_behind | transactions_to_cert |
+-----+-----+-----+-----+
| YES            | YES      | 950          | 0              |
+-----+-----+-----+-----+
```

```
mysql> UNLOCK tables;
```

application interaction

MySQL Router

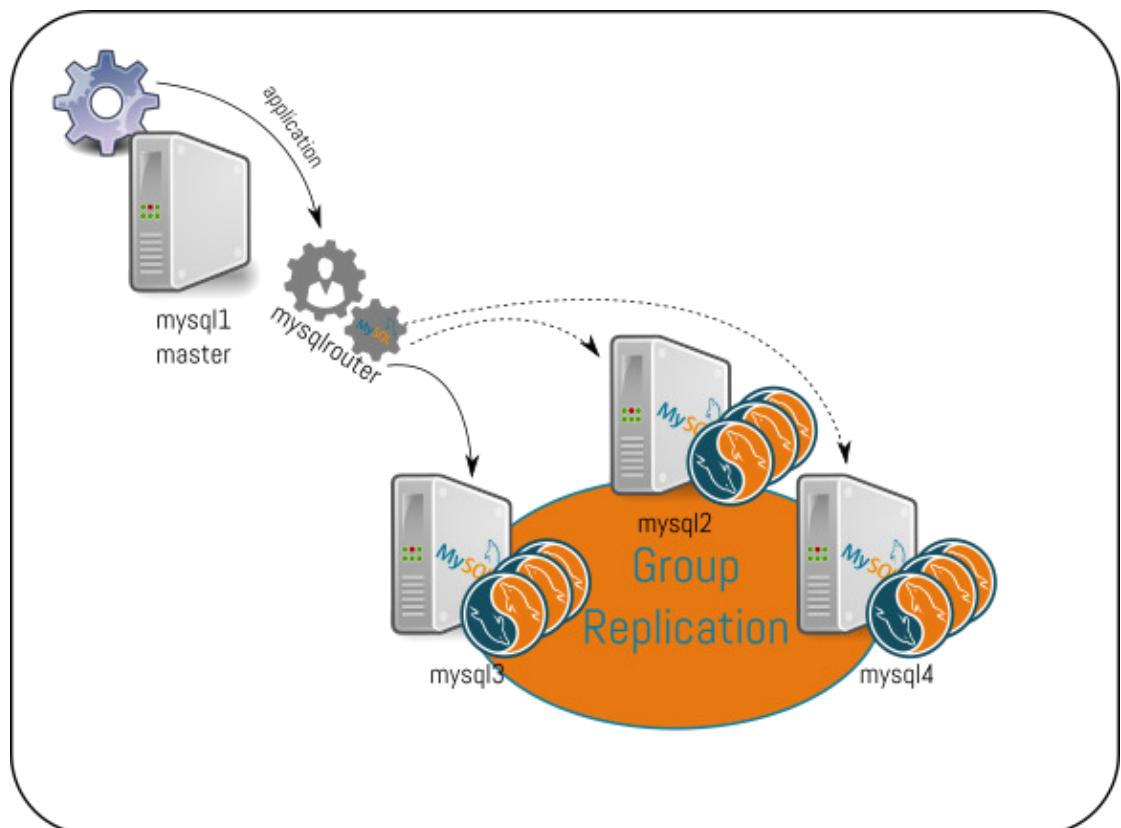


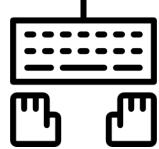
Copyright @ 2016 Oracle and/or its affiliates. All rights reserved.



MySQL Router

We will now use mysqlrouter between our application and the cluster.





MySQL Router (2)

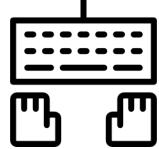
Configure MySQL Router (bootstrap it using an instance):

```
[mysql2 ~]# mysqlrouter --bootstrap mysql2:3306
Please enter the administrative MASTER key for the MySQL InnoDB cluster:
MySQL Router has now been configured for the InnoDB cluster 'plam'.
```

The following connection information can be used **to** connect **to** the cluster.

Classic MySQL protocol connections **to** cluster '**plam**':

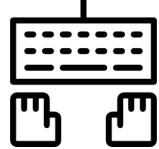
- **Read/Write** Connections: localhost:**6446**
- **Read/Only** Connections: localhost:**6447**



MySQL Router (3)

Now let's copy the configuration to mysql1 (our app server) and modify it to listen to port 3306:

```
[mysql2 ~]# scp /etc/mysqlrouter/mysqlrouter.conf mysql1:/etc/mysqlrouter/
```

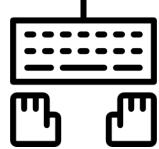


MySQL Router (3)

Now let's copy the configuration to mysql1 (our app server) and modify it to listen to port 3306:

```
[mysql2 ~]# scp /etc/mysqlrouter/mysqlrouter.conf mysql1:/etc/mysqlrouter/
```

```
[routing:default_rw]
-bind_port=6446
+bind_port=3306
+bind_address=mysql1
```



MySQL Router (3)

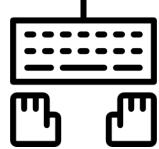
Now let's copy the configuration to mysql1 (our app server) and modify it to listen to port 3306:

```
[mysql2 ~]# scp /etc/mysqlrouter/mysqlrouter.conf mysql1:/etc/mysqlrouter/
```

```
[routing:default_rw]
-bind_port=6446
+bind_port=3306
+bind_address=mysql1
```

We can stop mysqld on mysql1 and start mysqlrouter into a screen session:

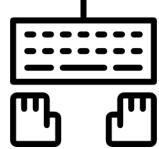
```
[mysql1 ~]# systemctl stop mysqld
[mysql1 ~]# mysqlrouter
```



MySQL Router (4)

Now we can point the application to the router:

```
[mysql1 ~]# run_app.sh mysql1
```



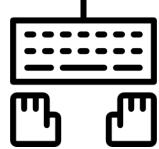
MySQL Router (4)

Now we can point the application to the router:

```
[mysql1 ~]# run_app.sh mysql1
```

Check app and kill mysqld on mysql3 (the Primary Master R/W node) !

```
[mysql3 ~]# kill -9 $(pidof mysqld)
```



MySQL Router (4)

Now we can point the application to the router:

```
[mysql1 ~]# run_app.sh mysql1
```

Check app and kill mysqld on mysql3 (the Primary Master R/W node) !

```
[mysql3 ~]# kill -9 $(pidof mysqld)
```

```
mysql> select member_host as "primary" from performance_schema.global_status
      join performance_schema.replication_group_members
      where variable_name = 'group_replication_primary_member'
      and member_id=variable_value;
+-----+
| primary |
+-----+
| mysql4  |
+-----+
```

Thank you !

Questions ?

