
ORACLE®

MySQL Group Replication in a nutshell

the core of MySQL InnoDB Cluster

Oracle Open World

September 19th 2016

Frédéric Descamps

MySQL Community Manager

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 up 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.

MySQL Community Reception @ Oracle OpenWorld

Celebrate, Have Fun and Mingle with Oracle's MySQL Engineers & Your Peers

- Tuesday, September 20, 7.00 pm
- Jillian's at Metreon: 175 Fourth Street, San Francisco
  - At the corner of Howard and 4th st.; only 2-min walk from Moscone Center (same place as last year)

Join us!

about.me/lefred

Who am I ?

Frédéric Descamps

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

get more at the conference

MySQL Group Replication

Other sessions

- CON2907 - MySQL High Availability
  Matt Lord
- CON4669 - MySQL High Availability with Group Replication
  Nuno Gomes
- CON3373 - The State of the Art of MySQL Replication
  Luis Soares
- HOL2912 - Building a Highly Available MySQL Database Service with Group Replication
  Matt Lord

Agenda

- Group Replication concepts

Agenda

- Group Replication concepts
- Using new & cool tools

Agenda

- Group Replication concepts
- Using new & cool tools
- Application interaction

the magic explained

Group Replication Concept

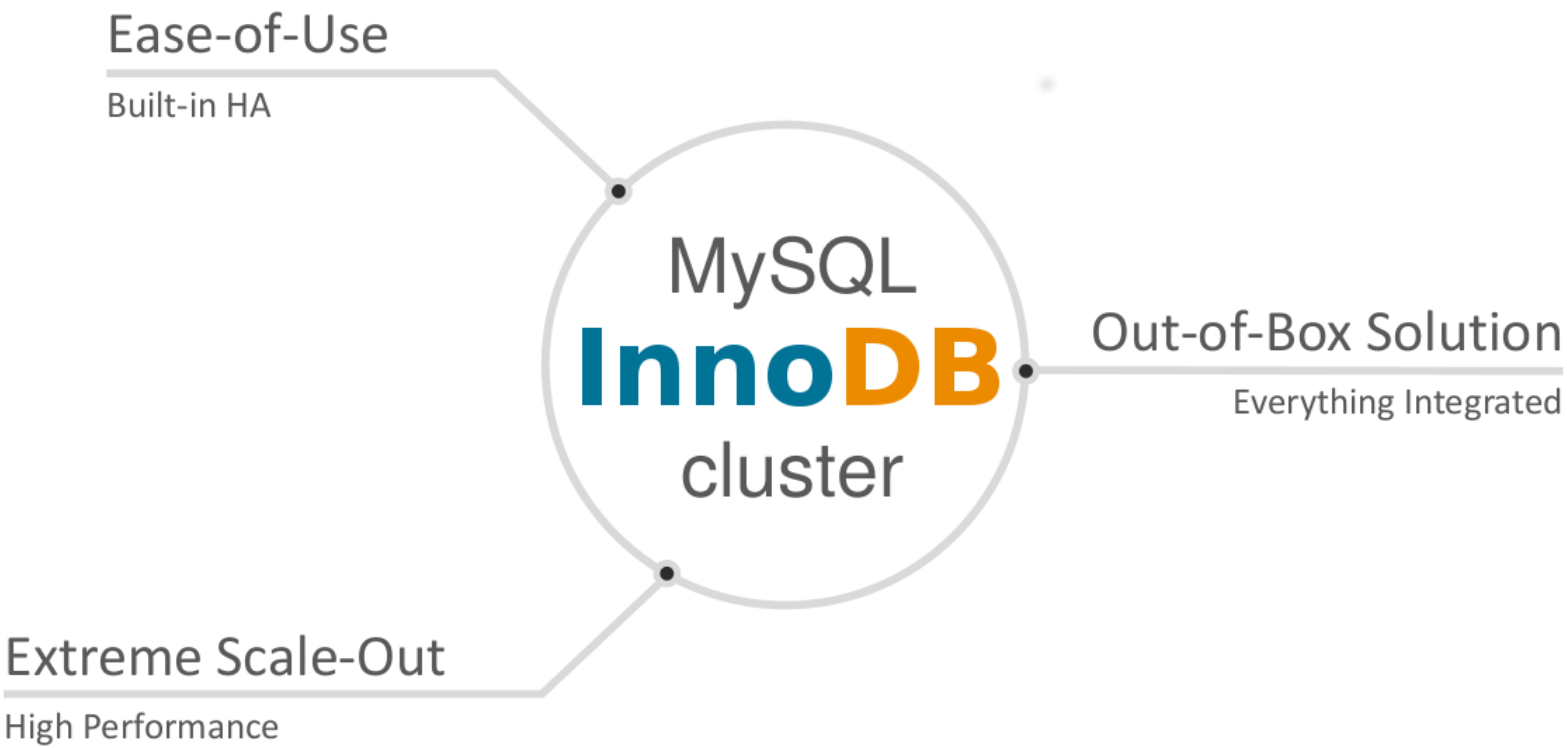
Group Replication : what is it ?

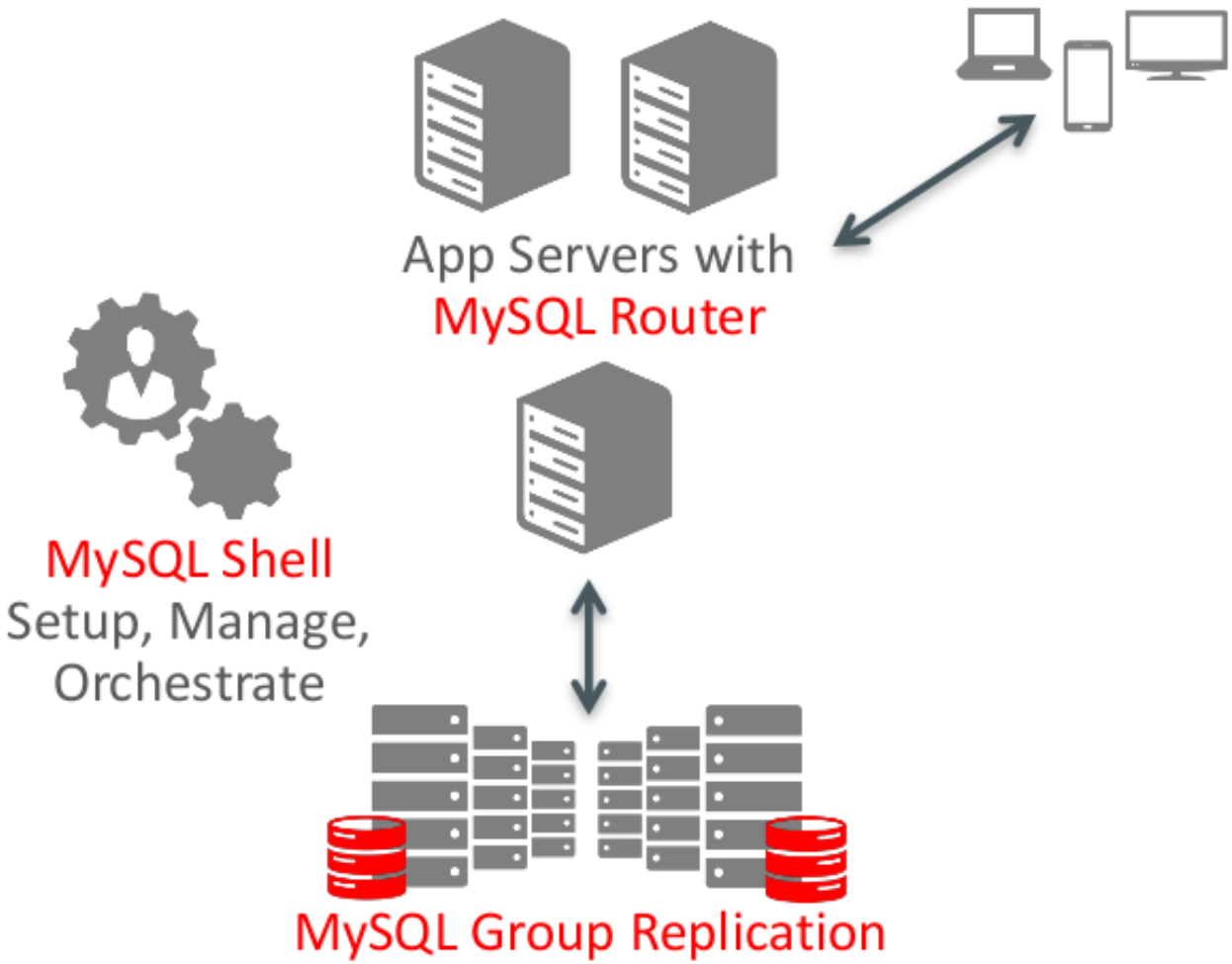
Group Replication : what is it ?

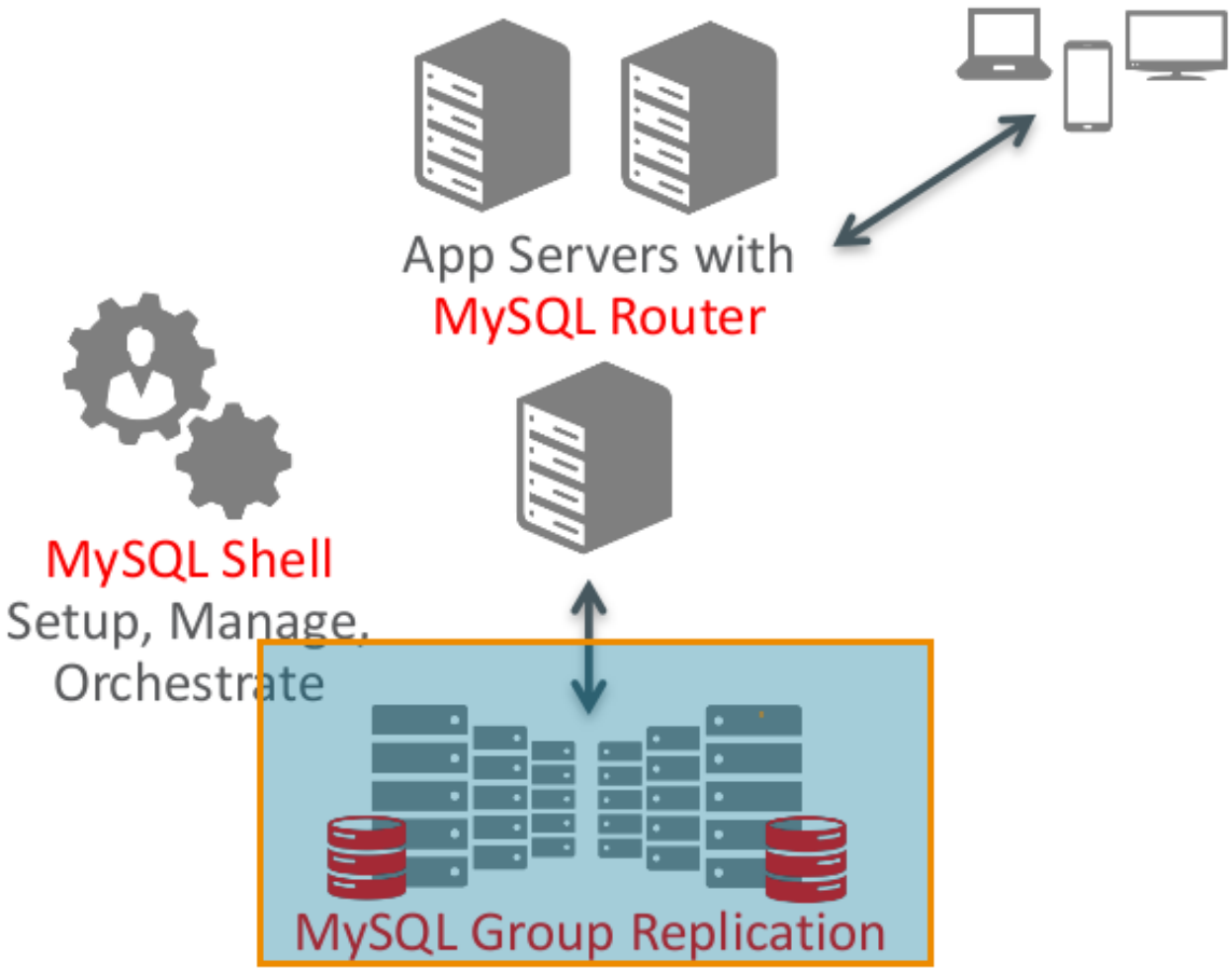
MySQL is one of the major components of MySQL InnoDB Cluster

Group Replication : what is it ?

MySQL 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

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
- 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
- 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) send 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) send the changes (binlog events) made by a transaction to the group through the GCS.
- All members consume the writeset and certify it

Group Replication : how does it work ?

- A node (member of the group) send the changes (binlog events) made by a transaction to the group through the GCS.
- All members consume the writeset and certify it
- 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, in fact the writesets replication is **synchronous** and then certification and apply of the changes are local to each nodes and asynchronous.

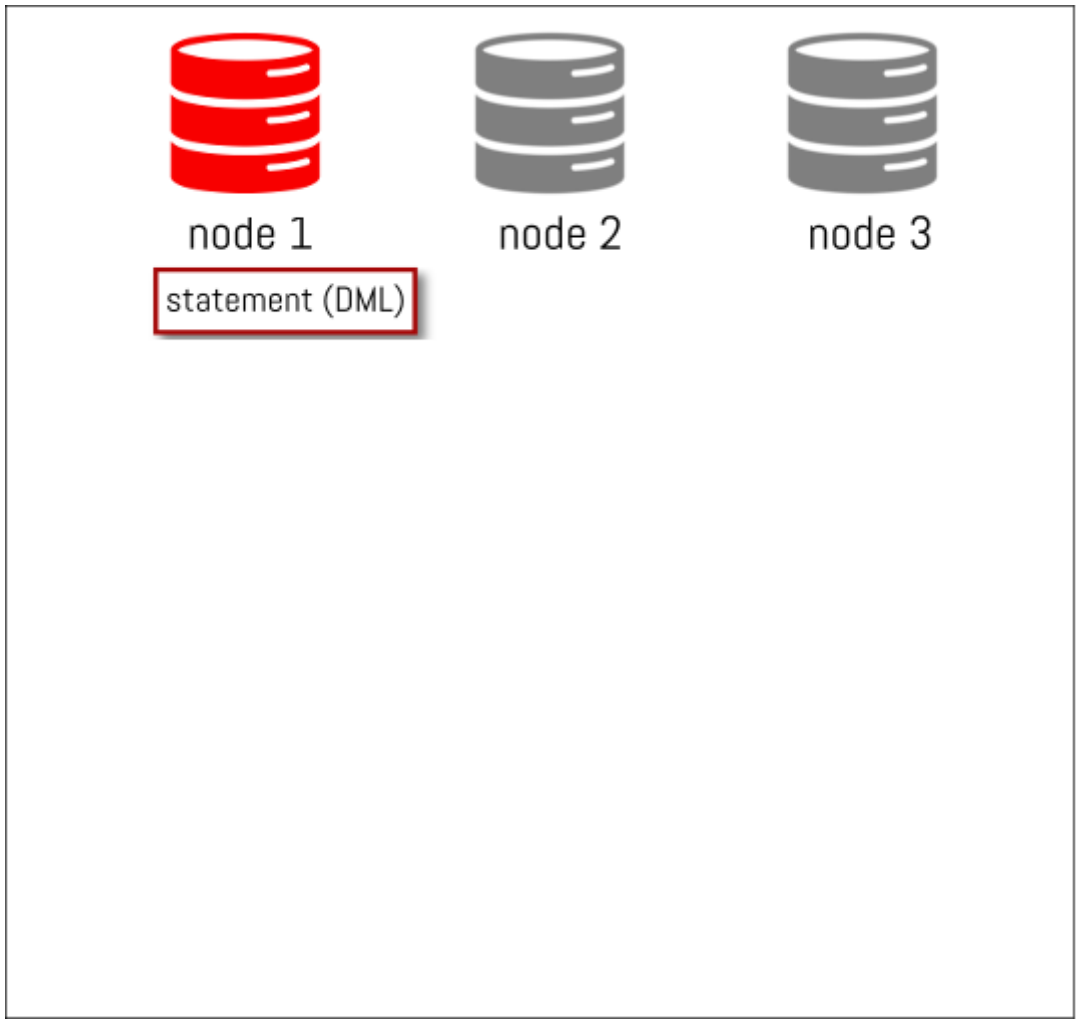
OK... but how does it work ?!

It's just magic !

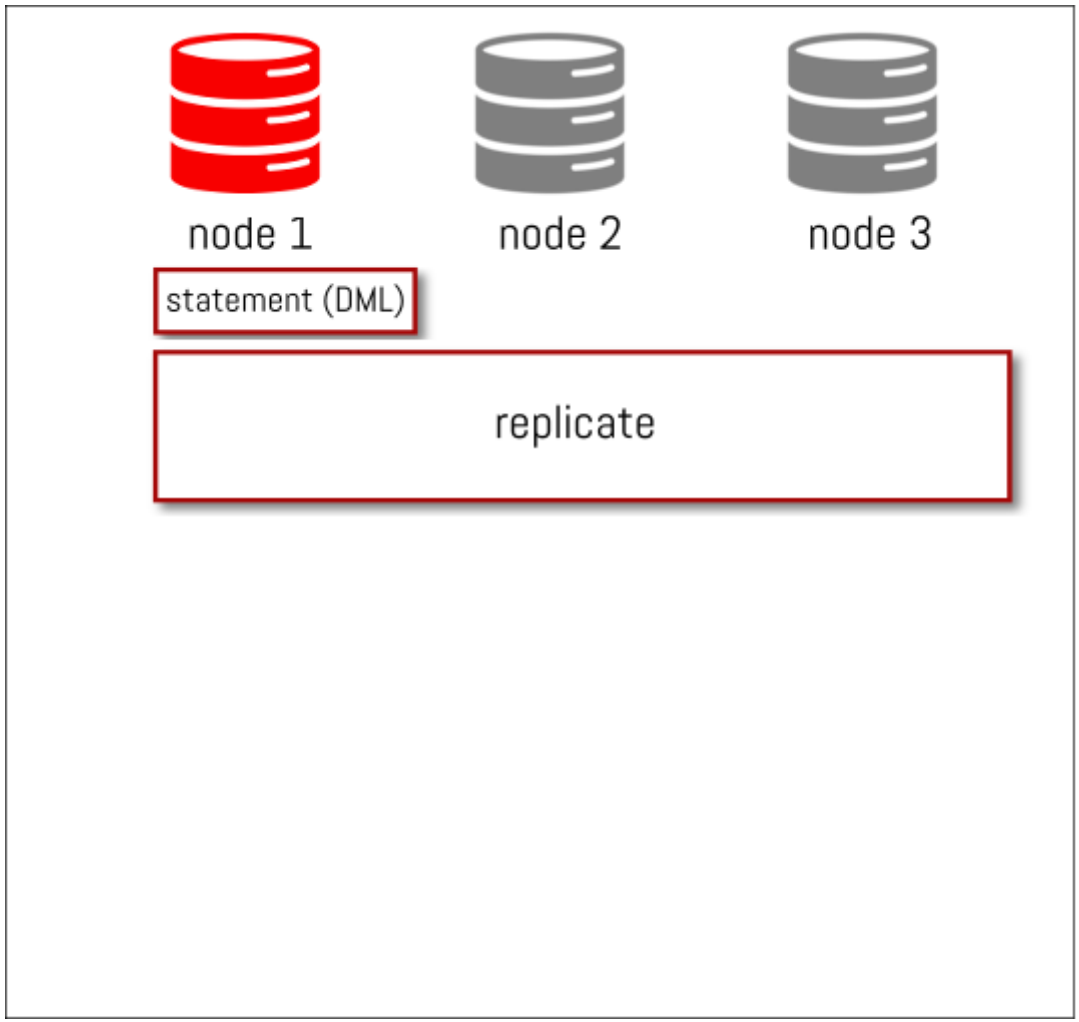
... no, in fact the writesets replication is **synchronous** and then certification and apply of the changes are local to each nodes and asynchronous.

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

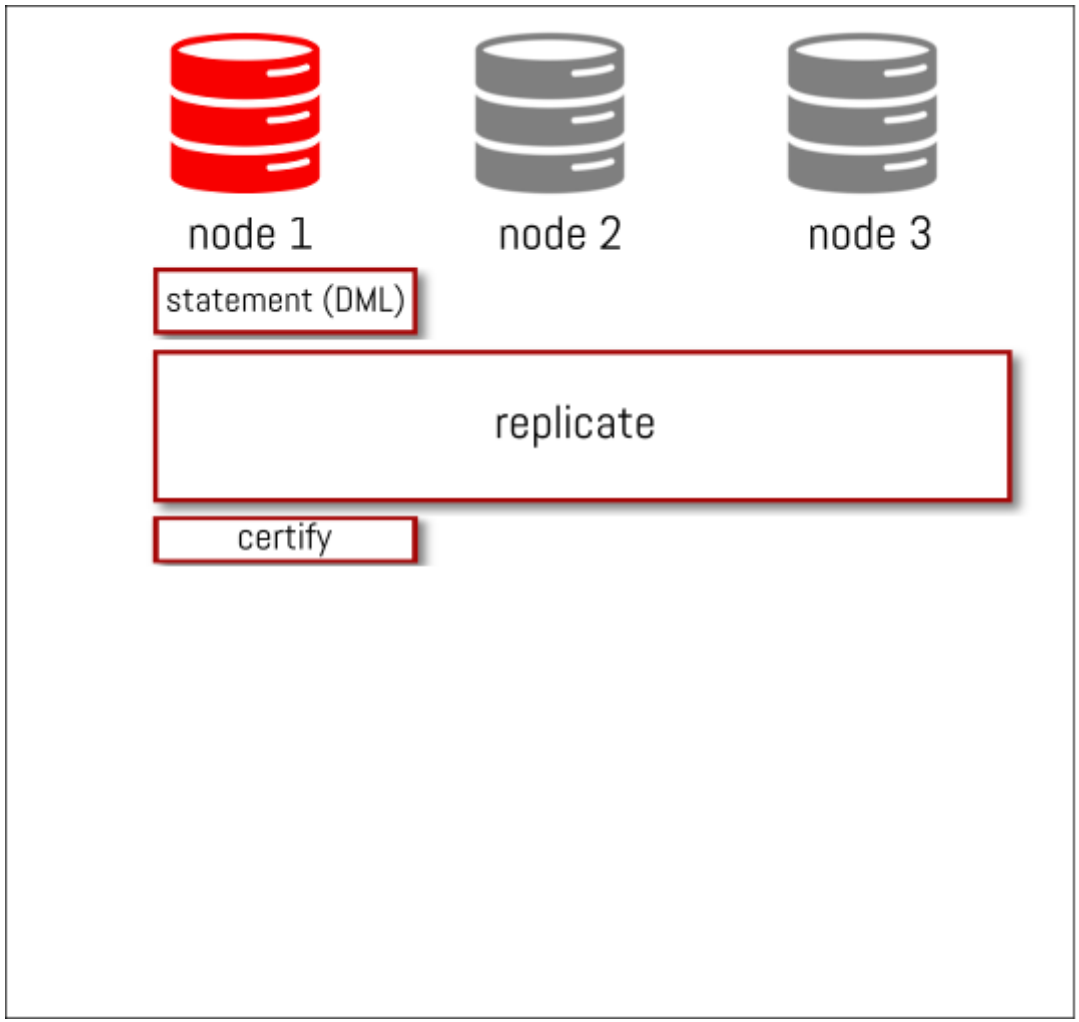
MySQL Group Replication (autocommit)



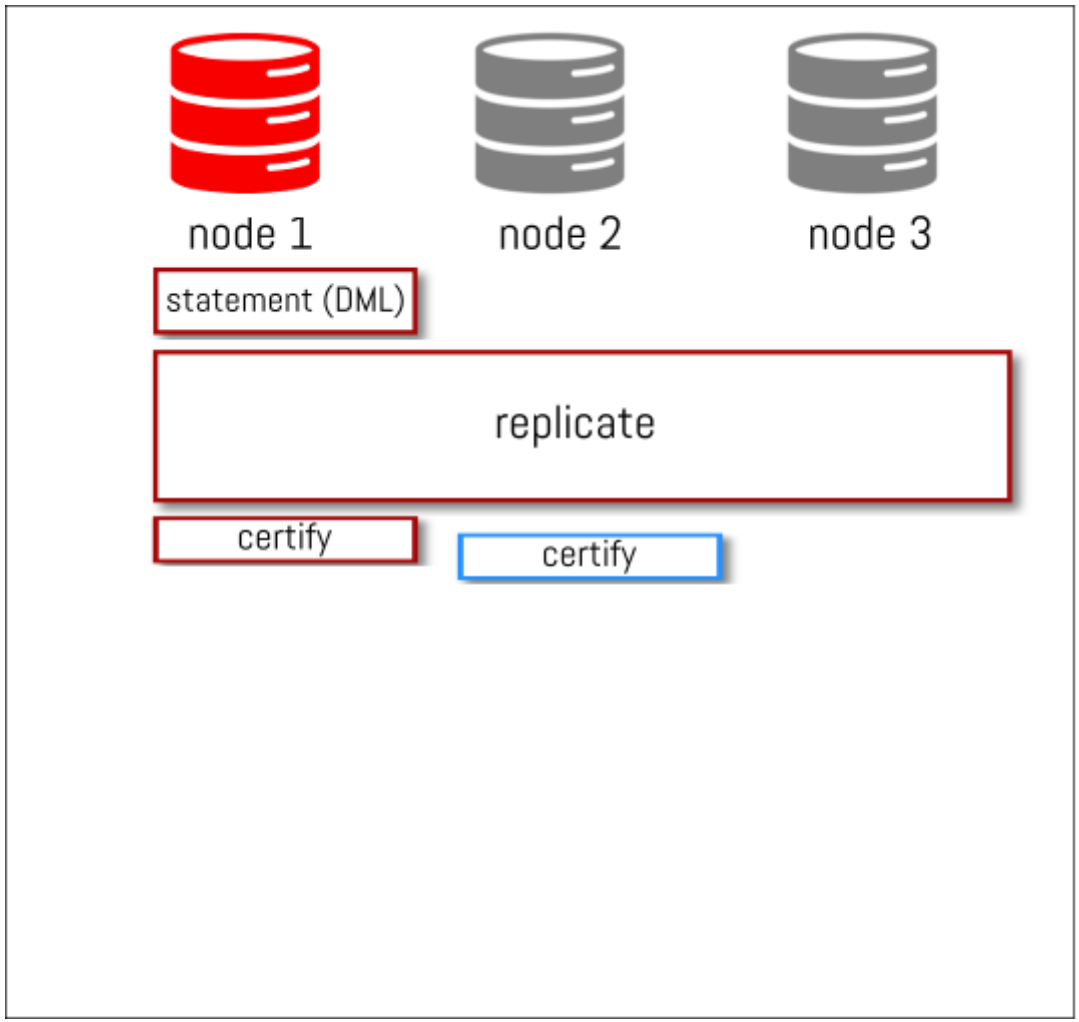
MySQL Group Replication (autocommit)



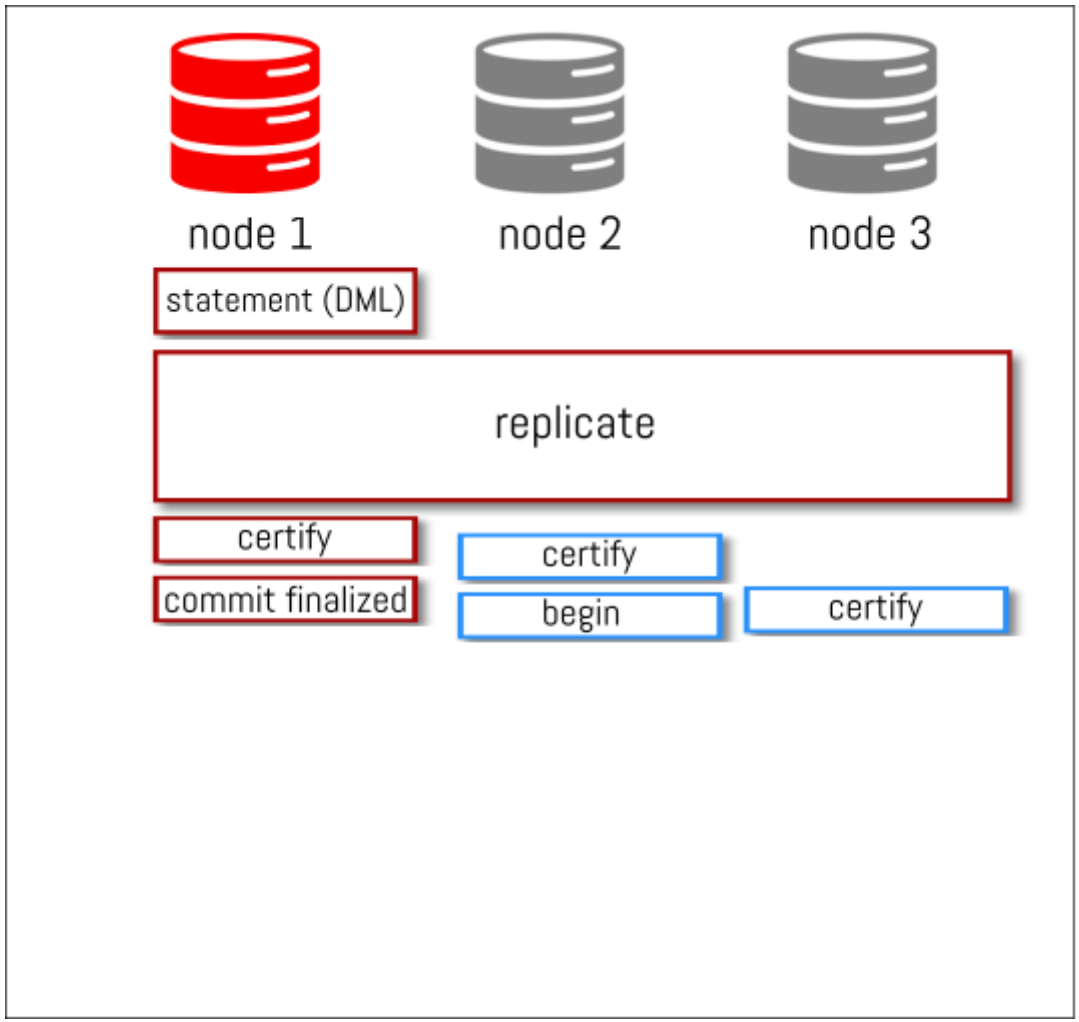
MySQL Group Replication (autocommit)



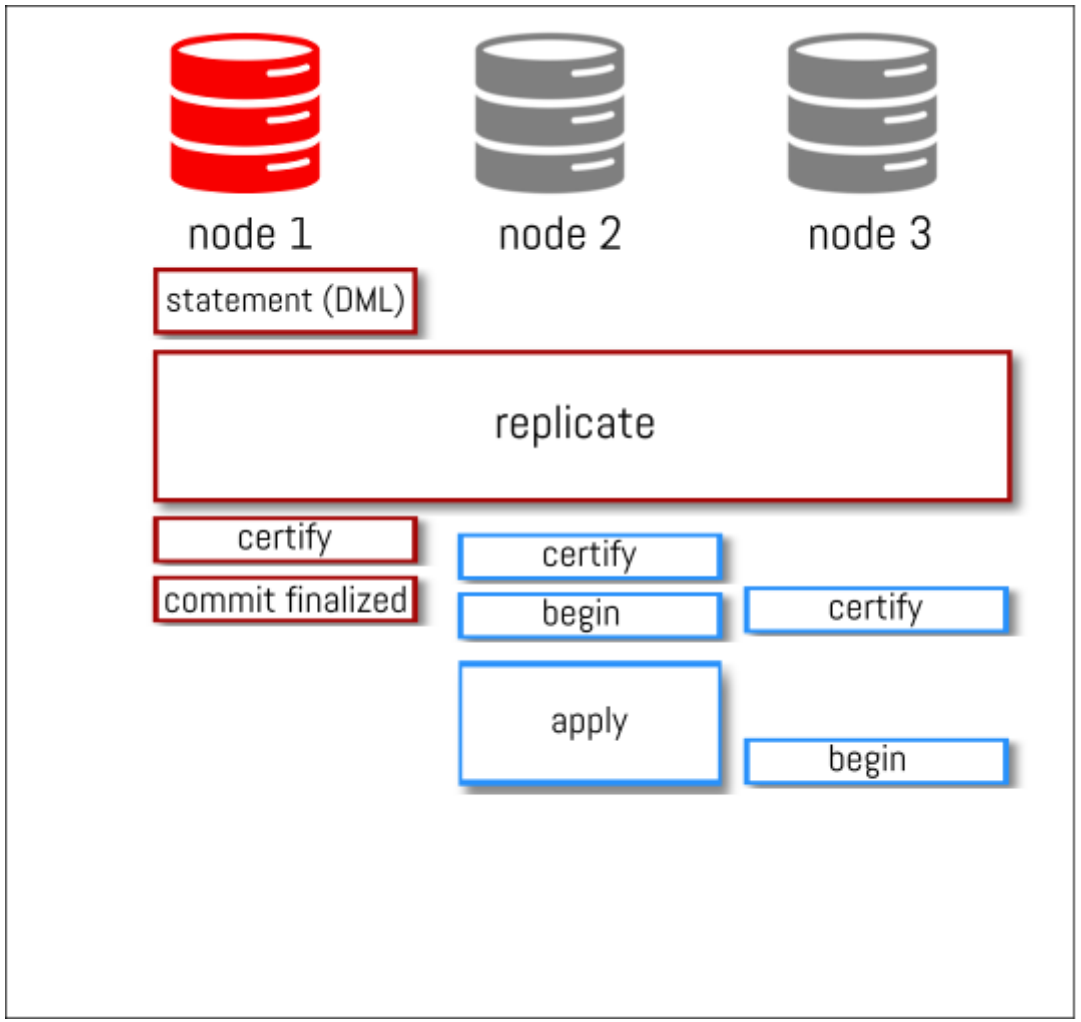
MySQL Group Replication (autocommit)



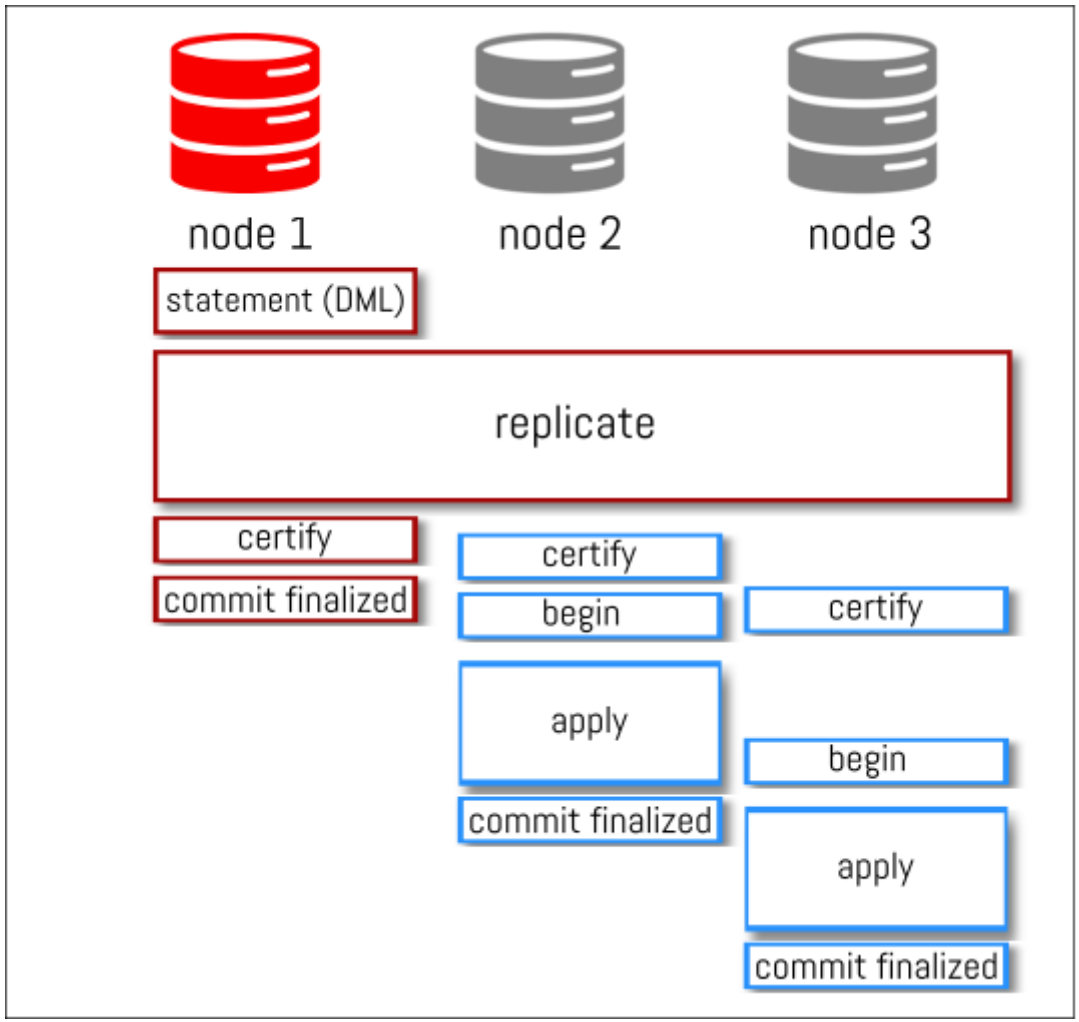
MySQL Group Replication (autocommit)



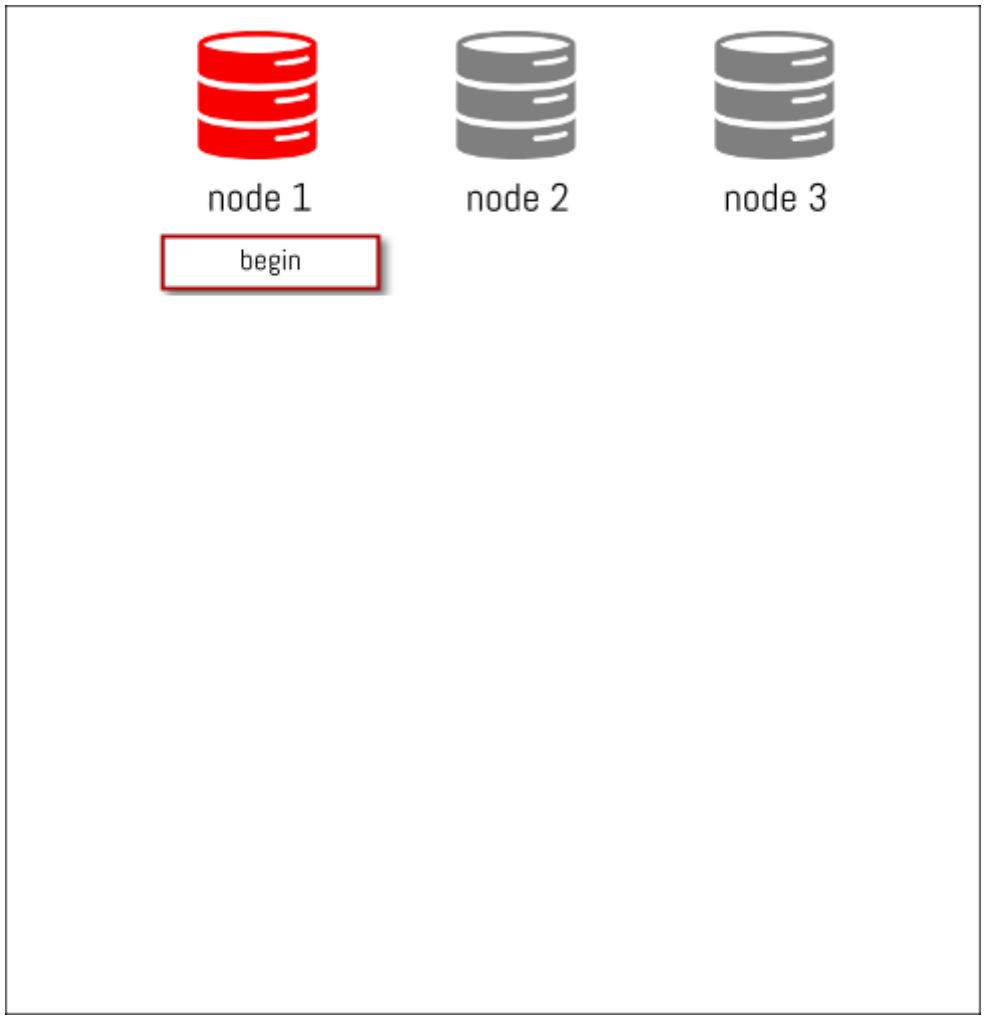
MySQL Group Replication (autocommit)



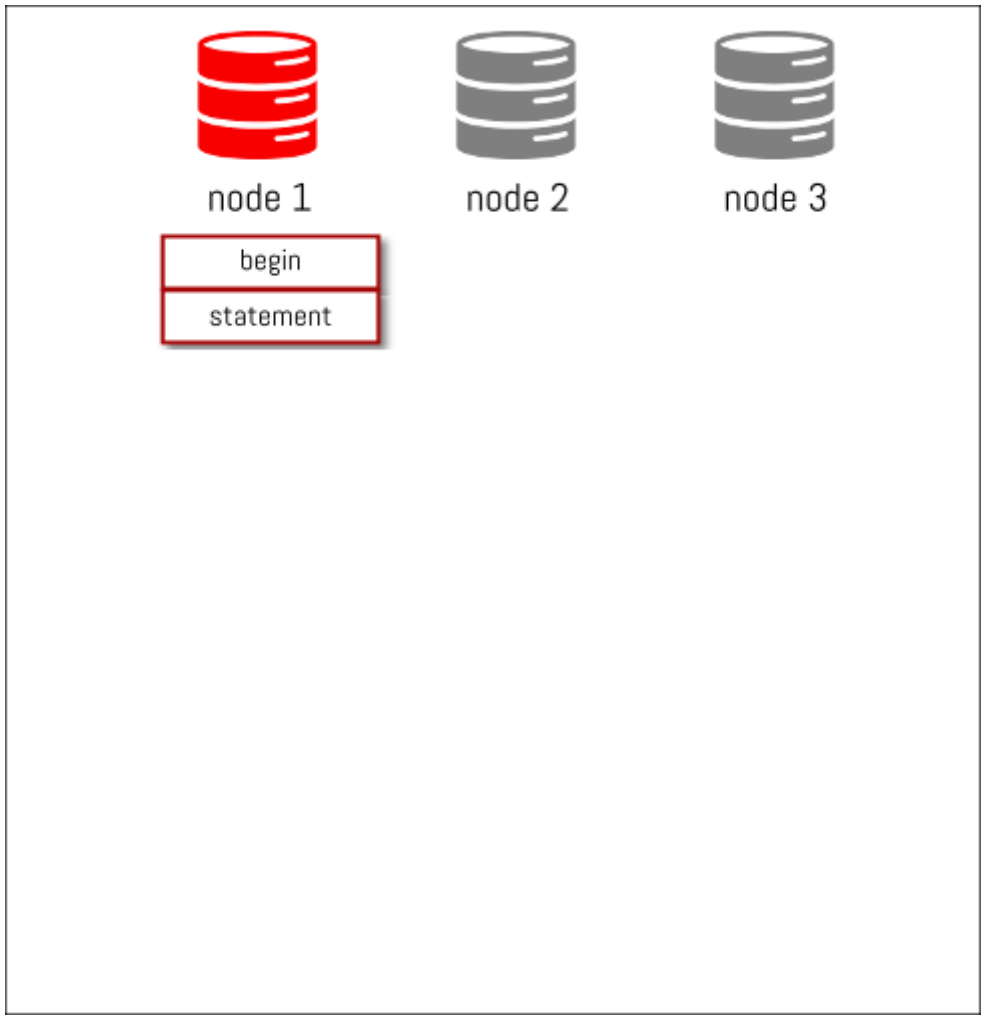
MySQL Group Replication (autocommit)



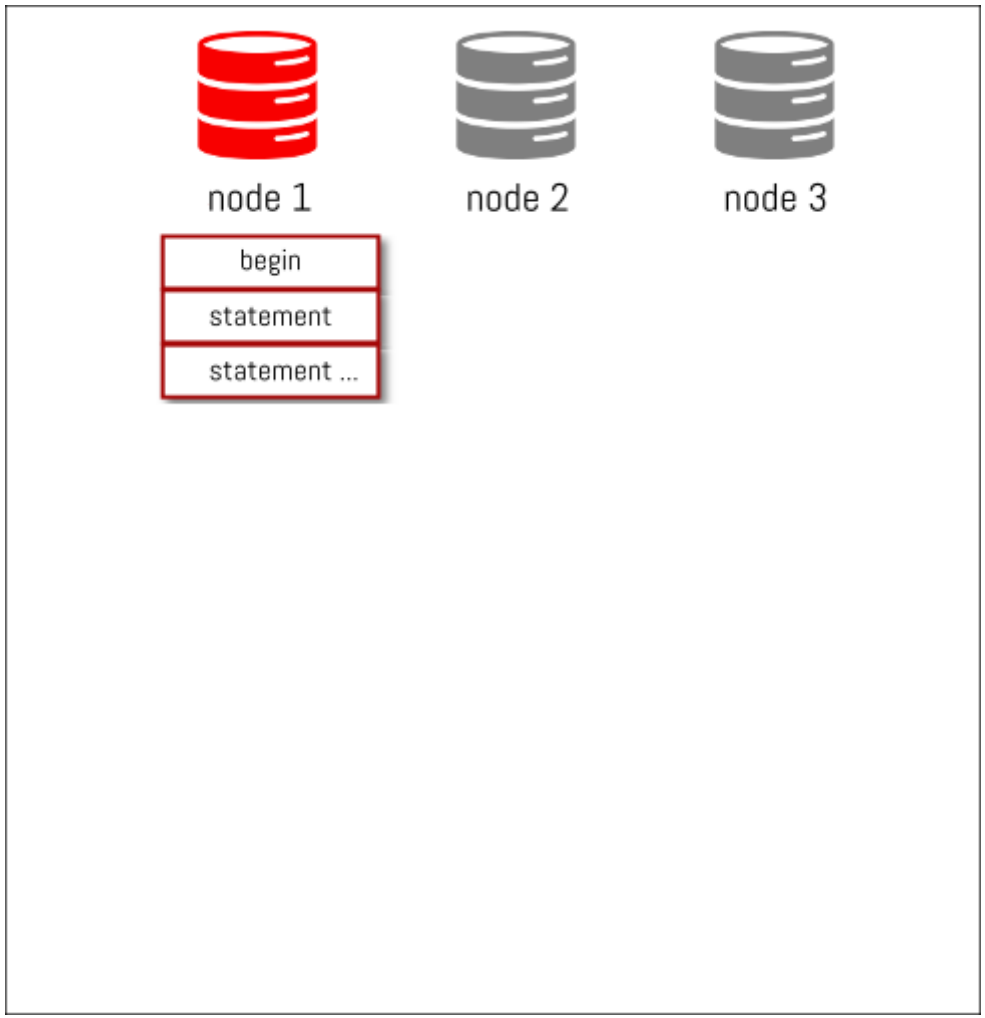
MySQL Group Replication (full transaction)



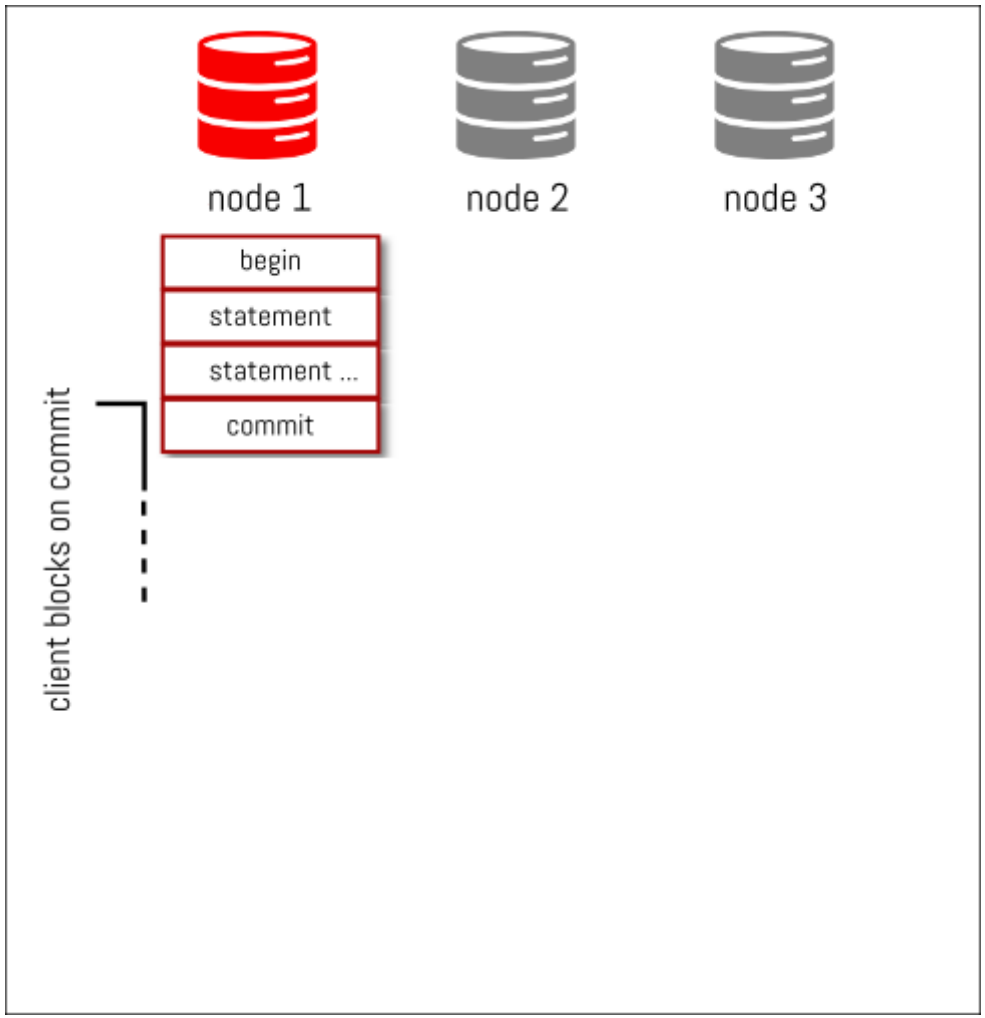
MySQL Group Replication (full transaction)



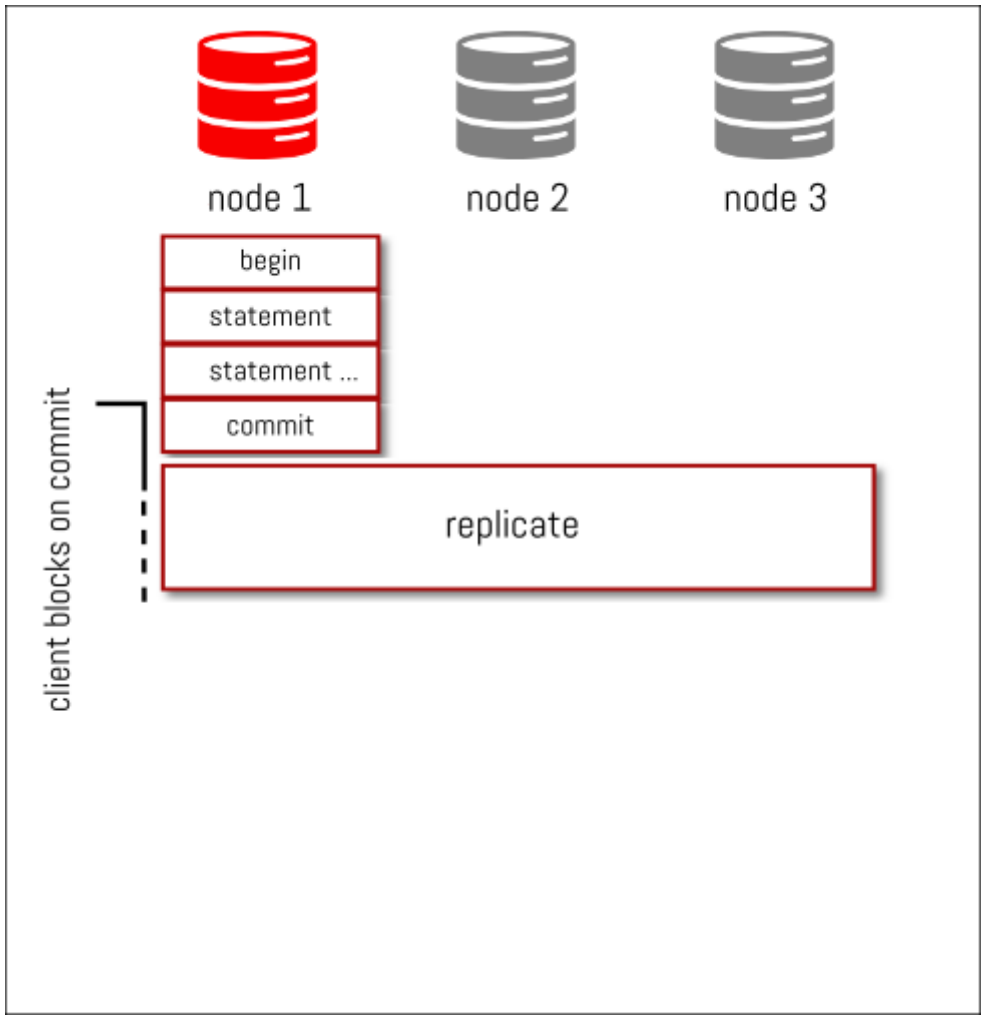
MySQL Group Replication (full transaction)



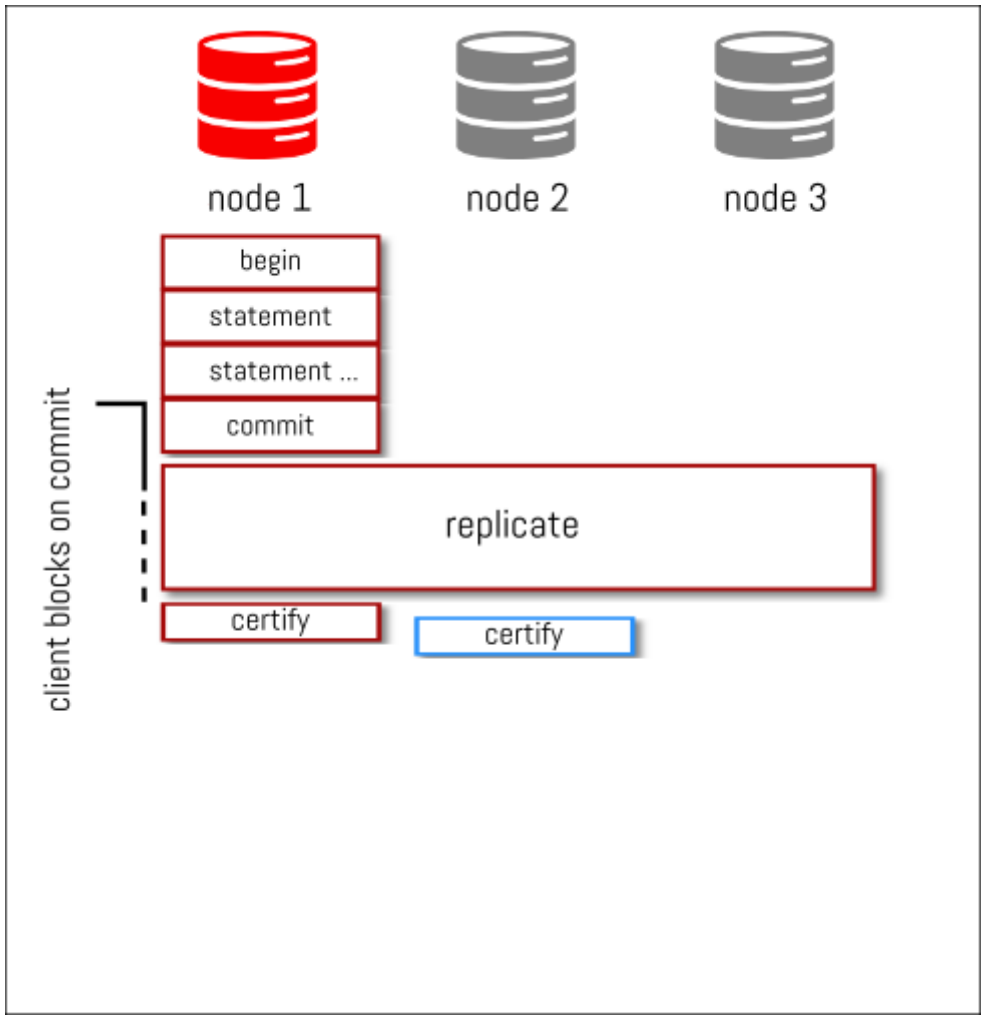
MySQL Group Replication (full transaction)



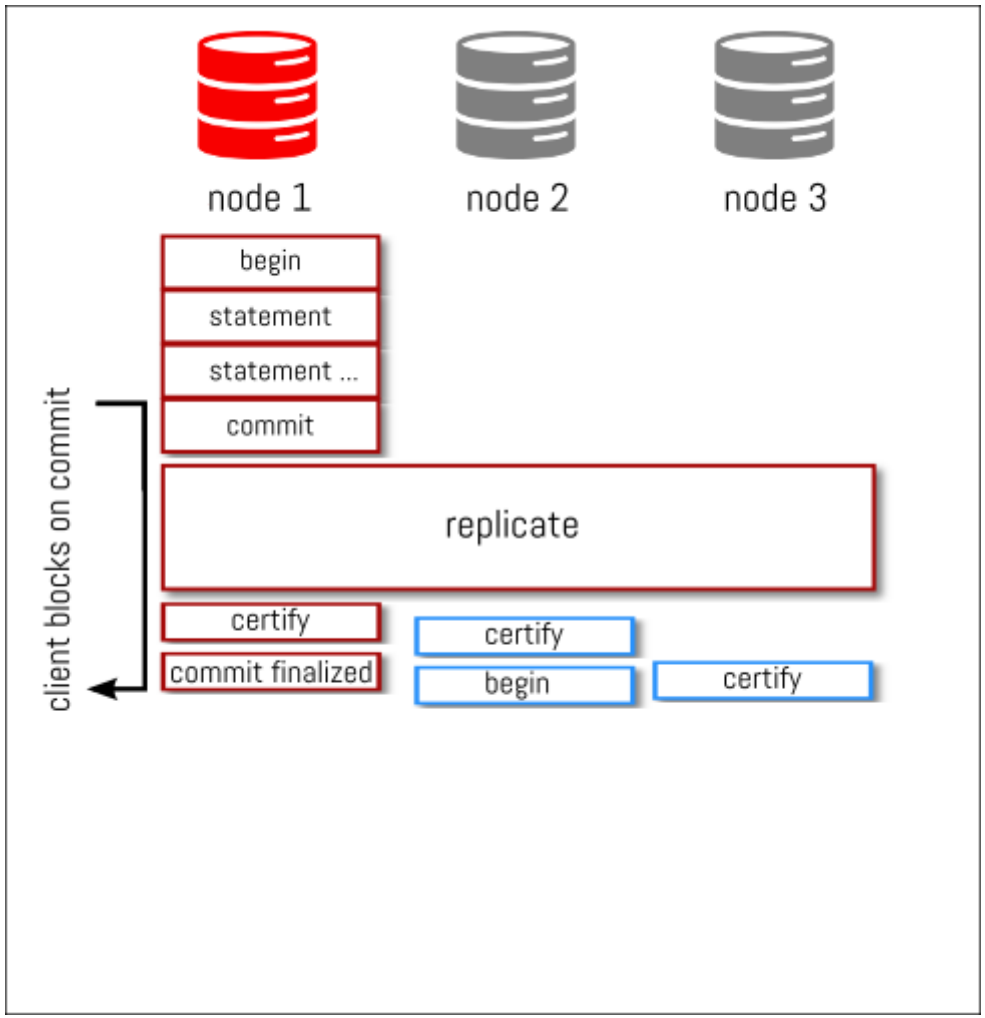
MySQL Group Replication (full transaction)



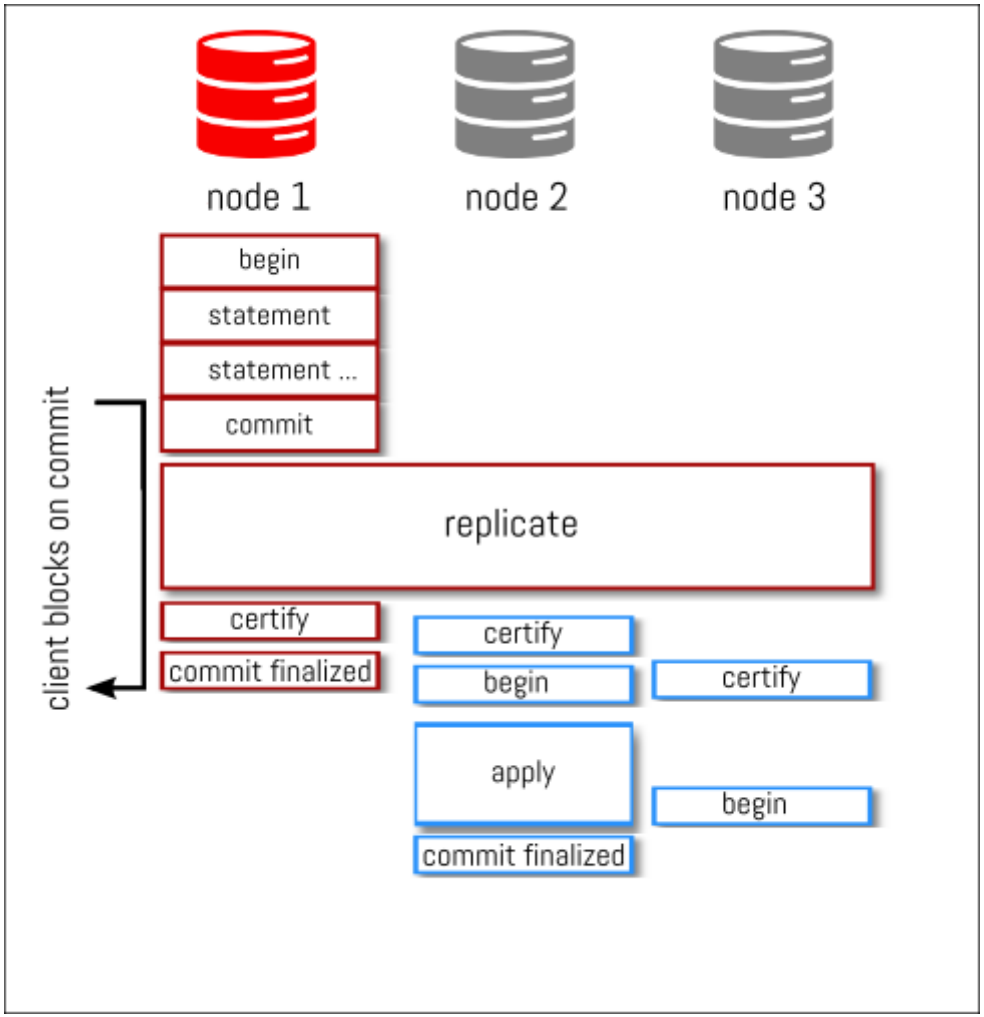
MySQL Group Replication (full transaction)



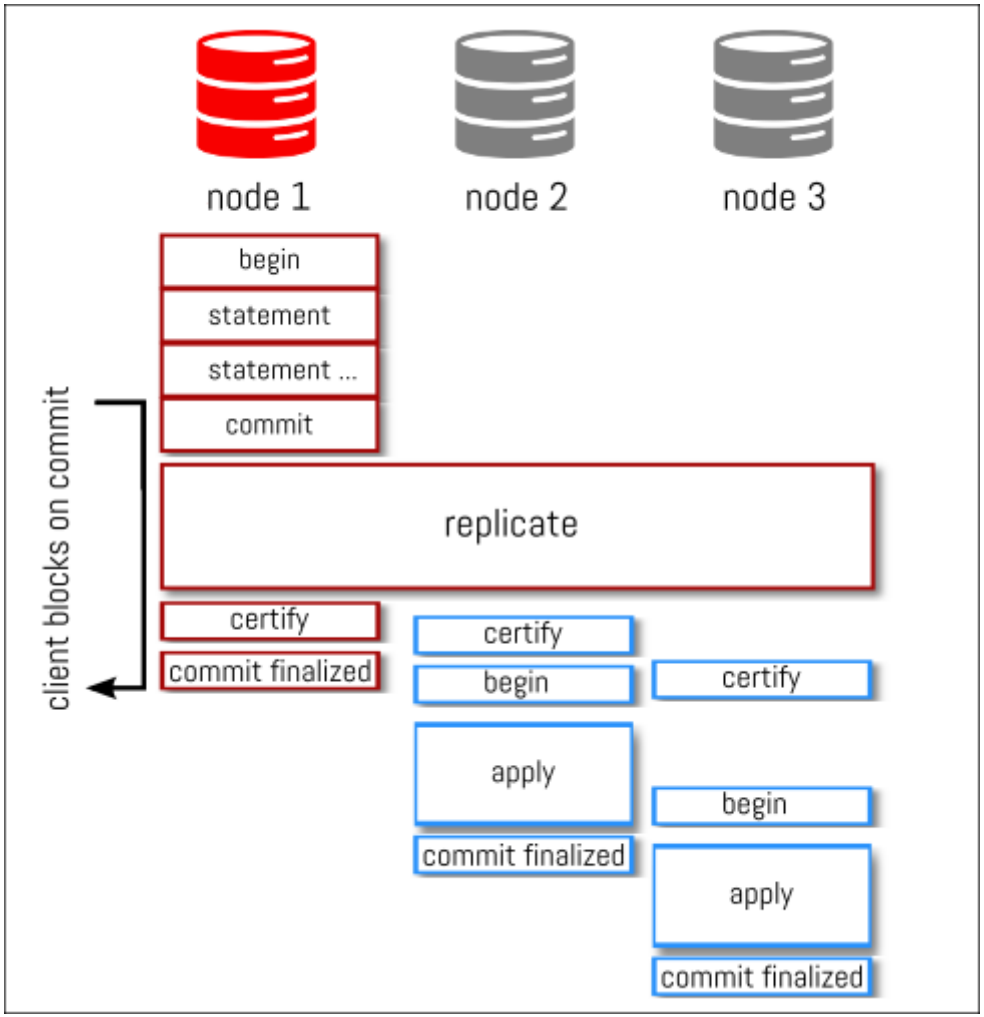
MySQL Group Replication (full transaction)



MySQL Group Replication (full transaction)



MySQL Group Replication (full transaction)

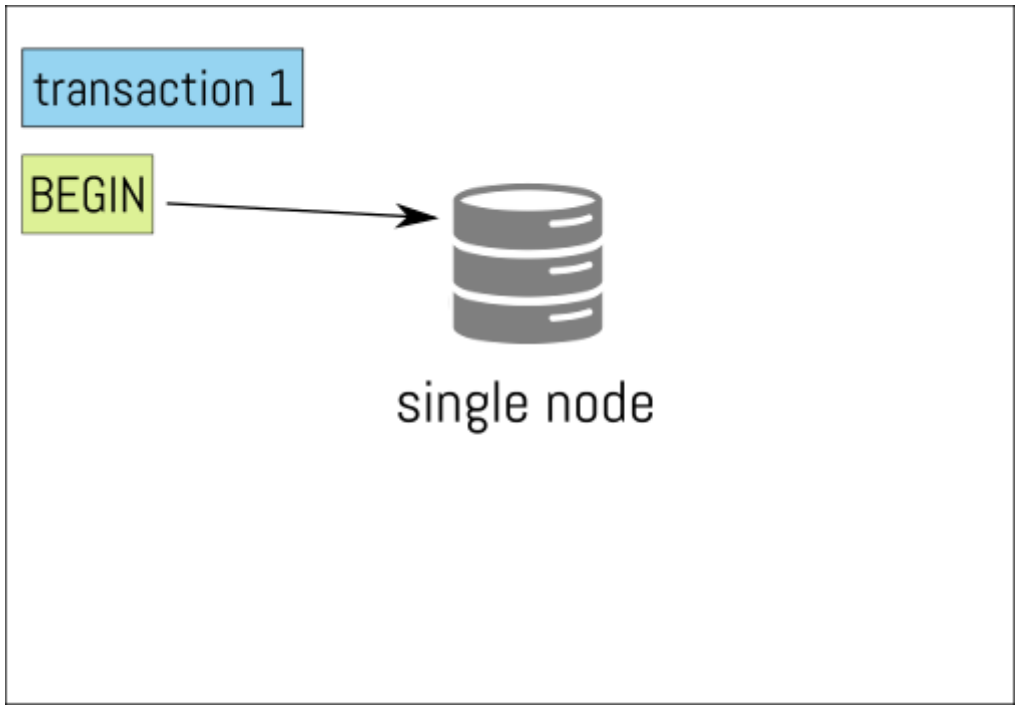


Group Replication : Optimistic Locking

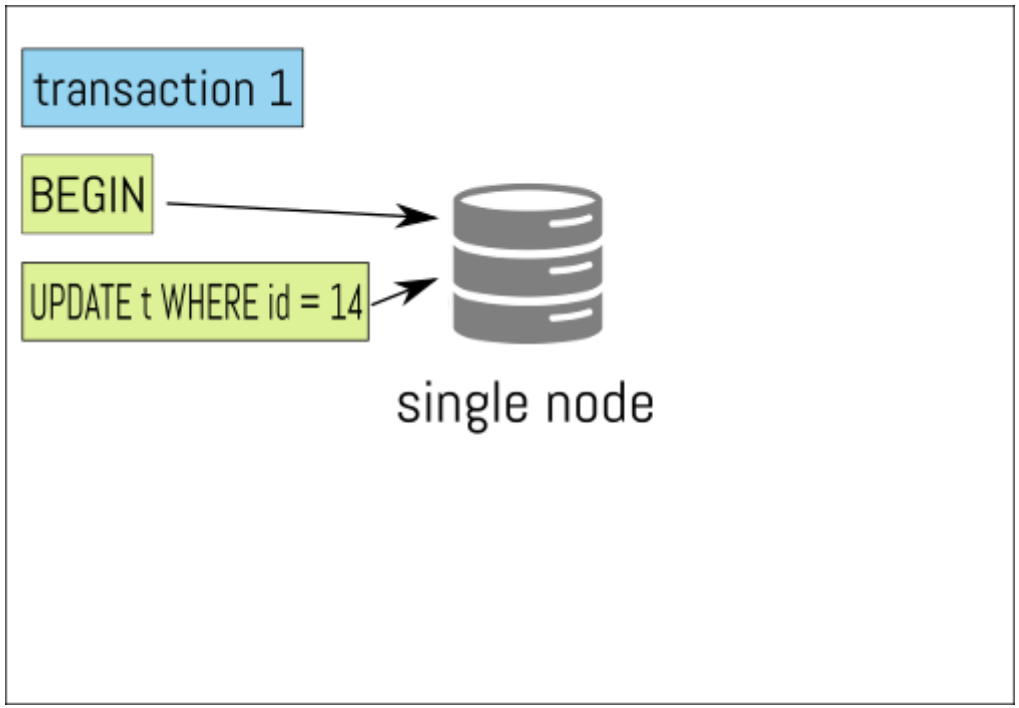
With GR we use what we call optimistic locking, it means we consider a resource free and this will be confirmed or not during certification.

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

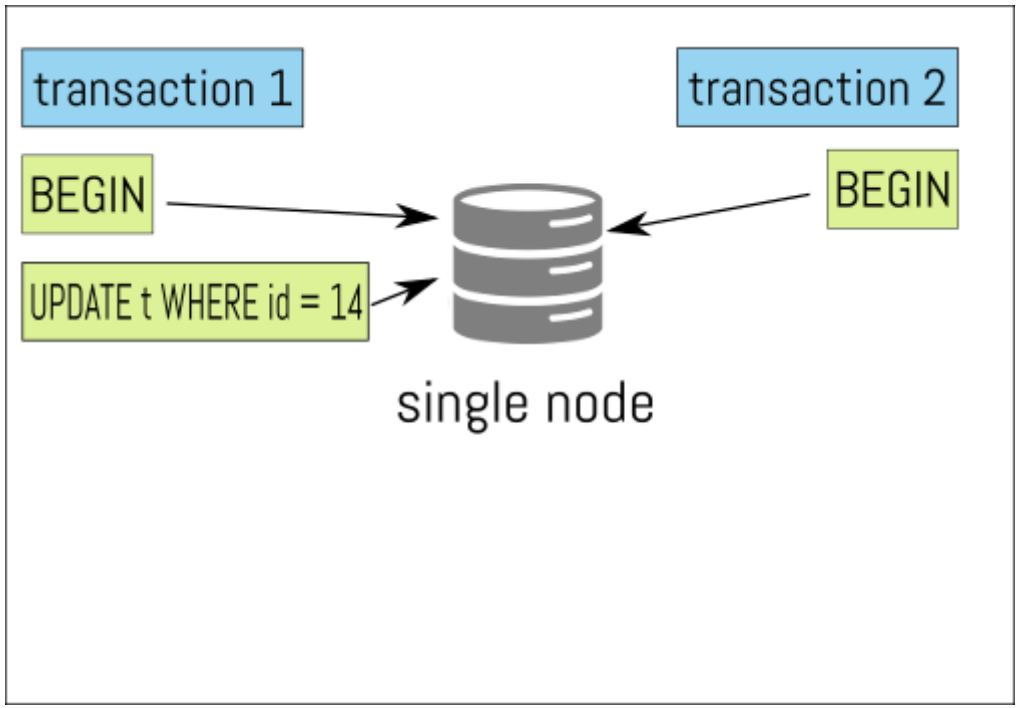
Traditional locking



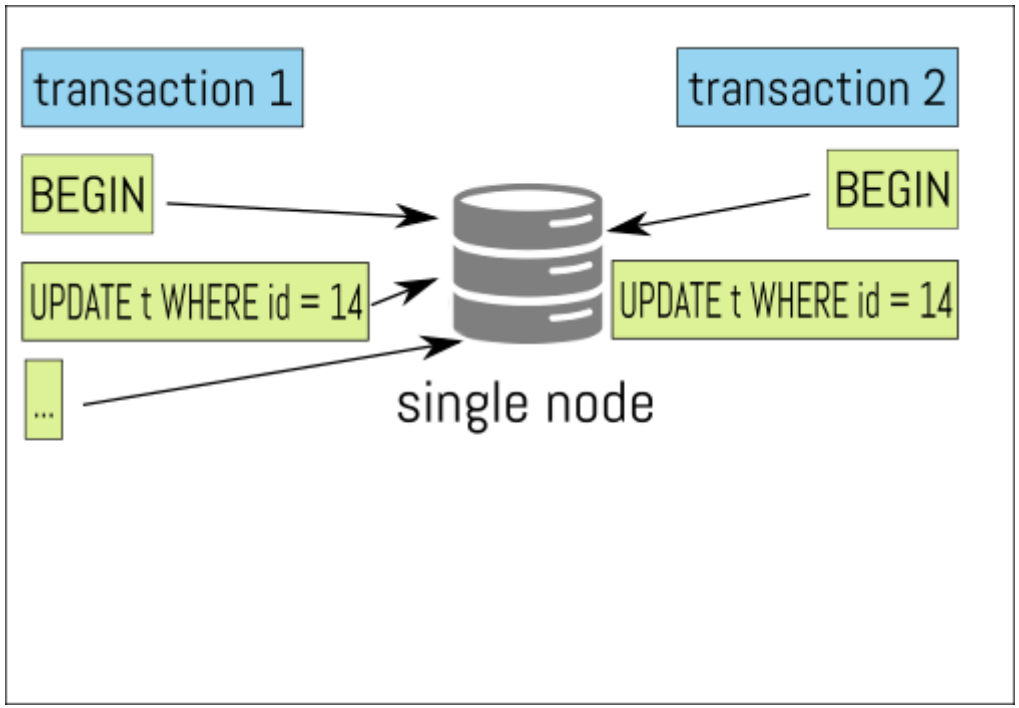
Traditional locking



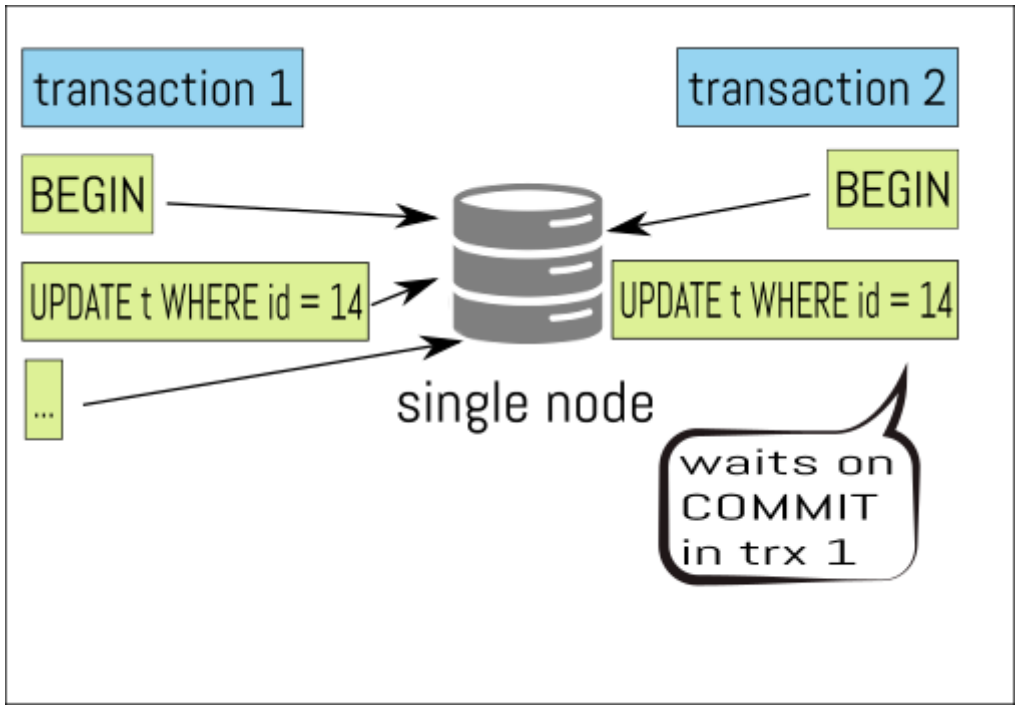
Traditional locking



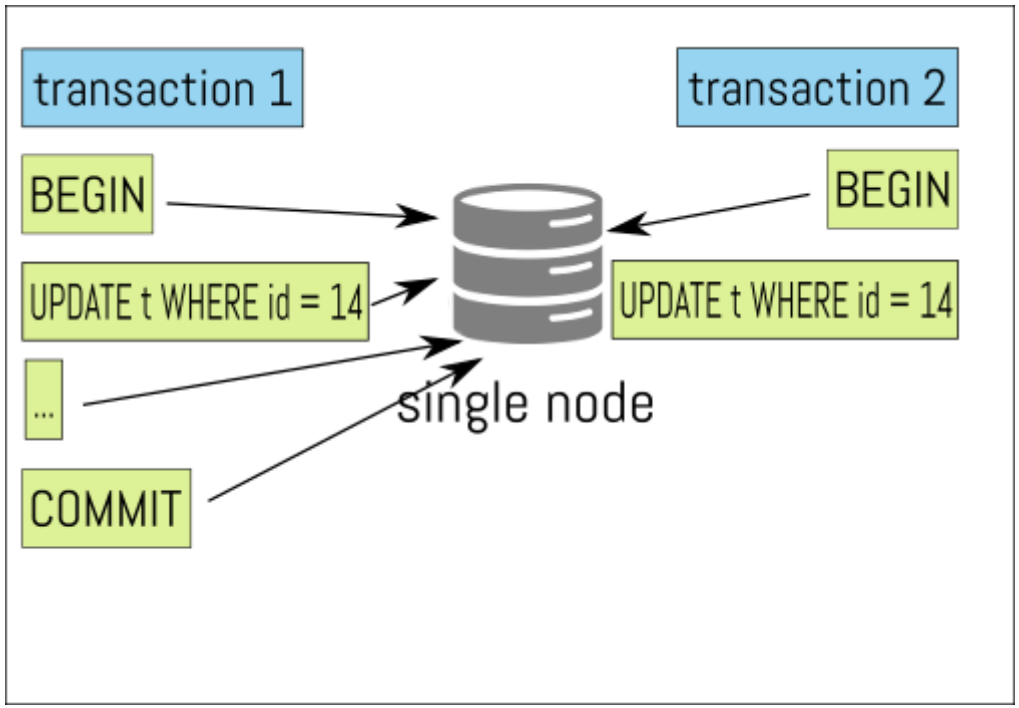
Traditional locking



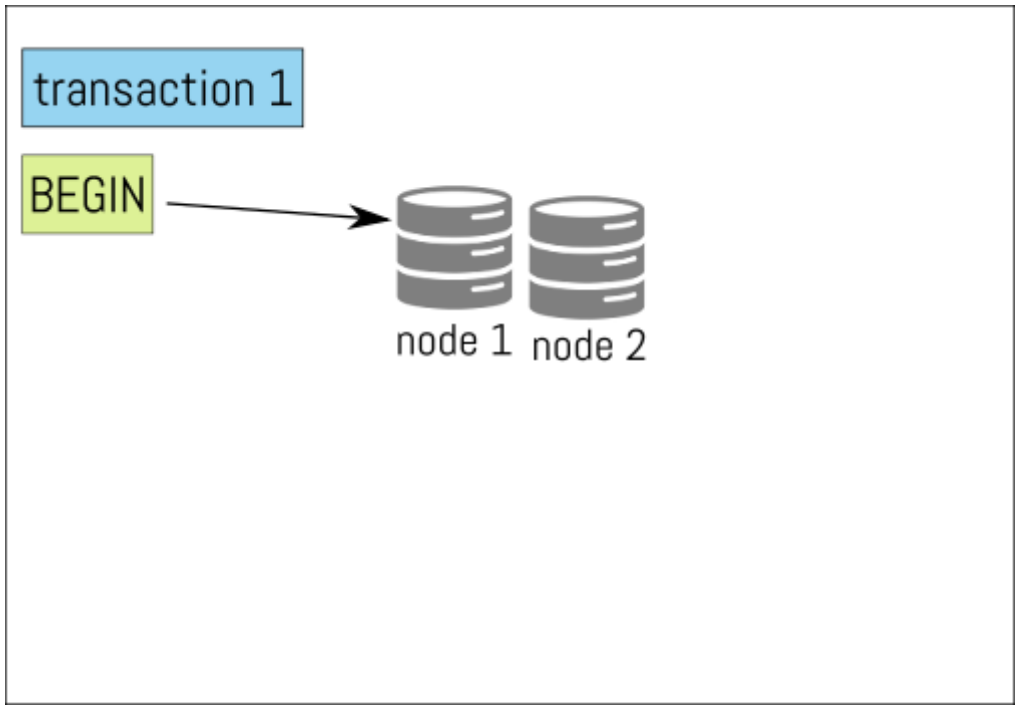
Traditional locking



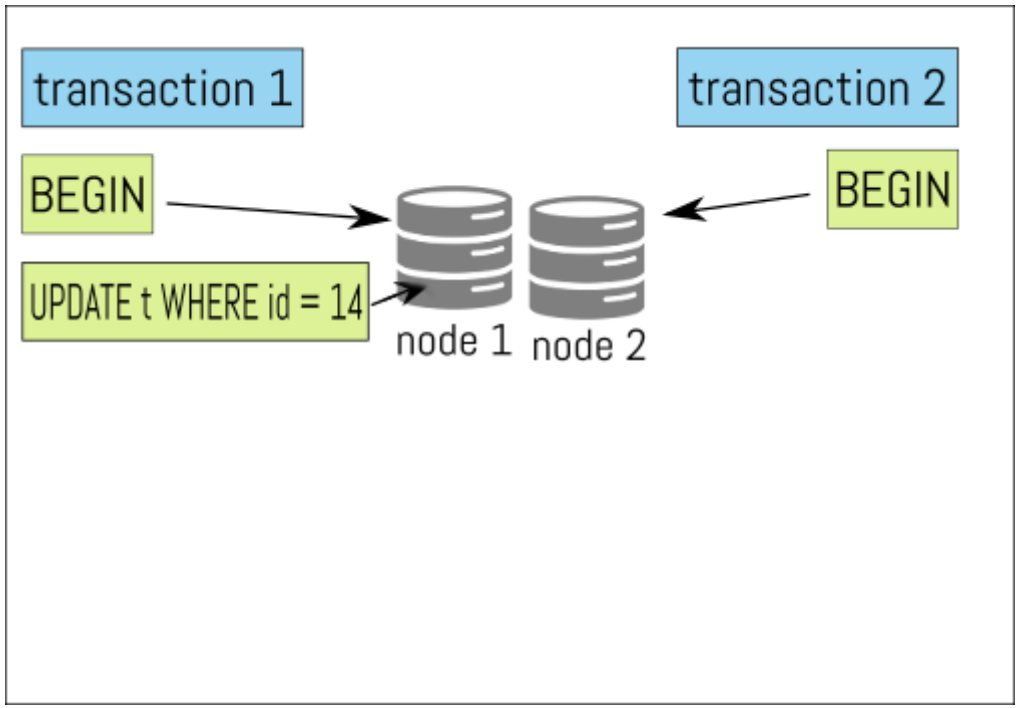
Traditional locking



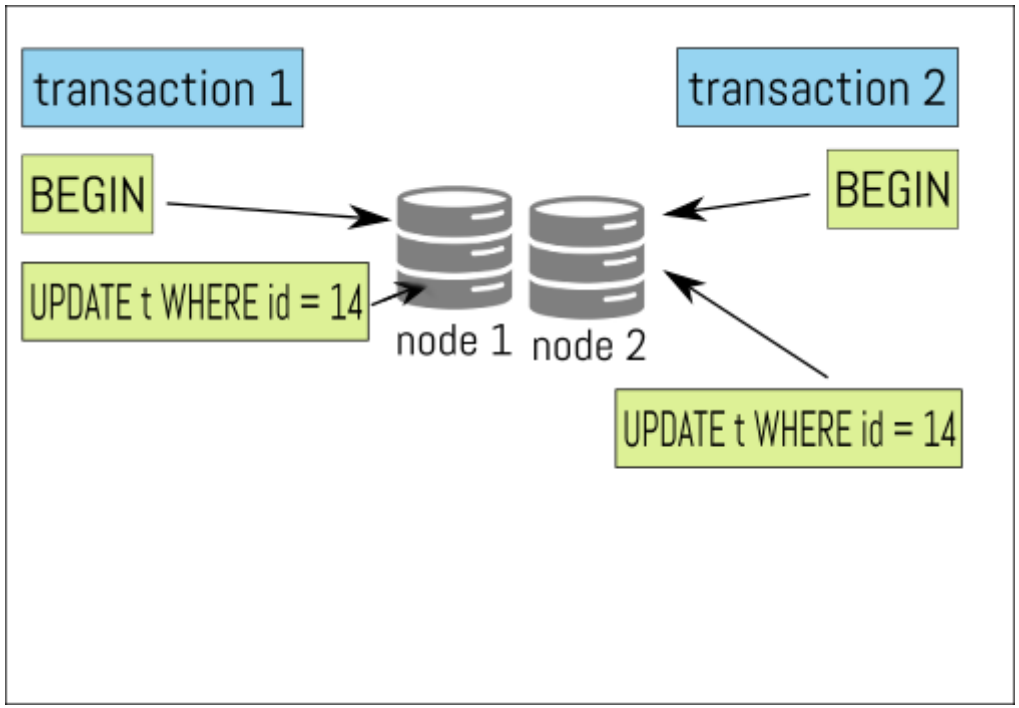
Optimistic Locking



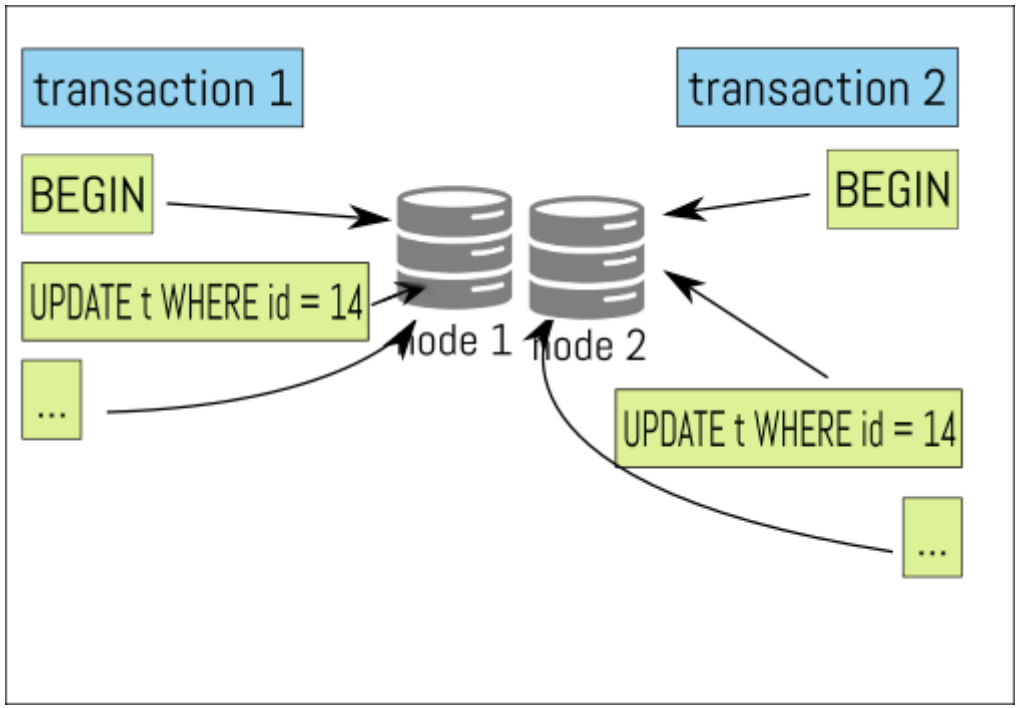
Optimistic Locking



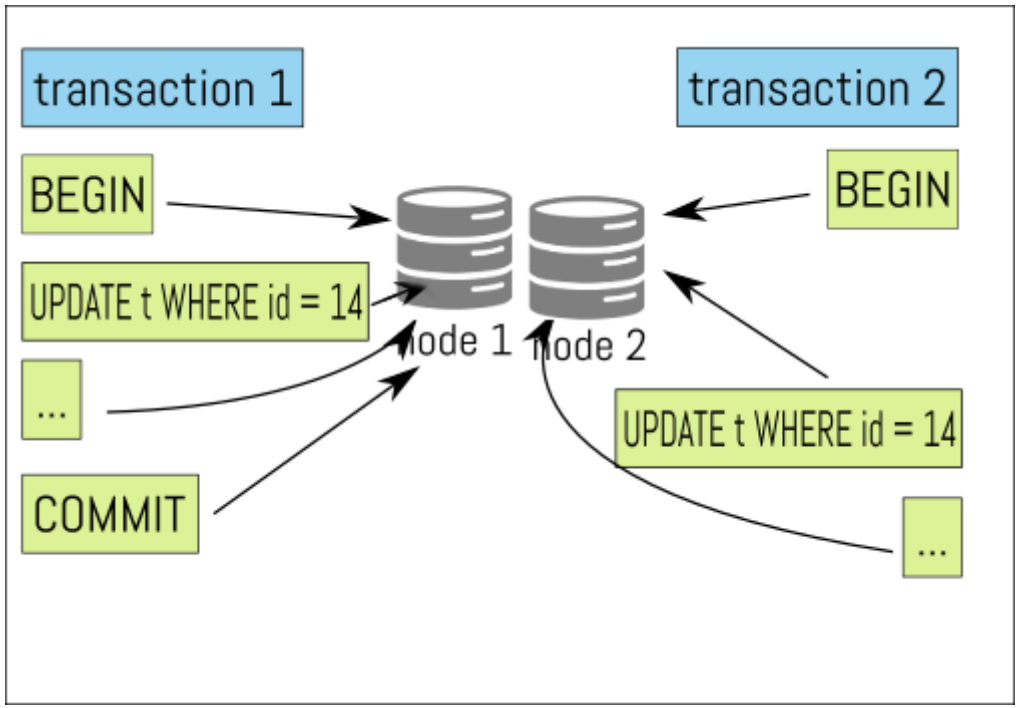
Optimistic Locking



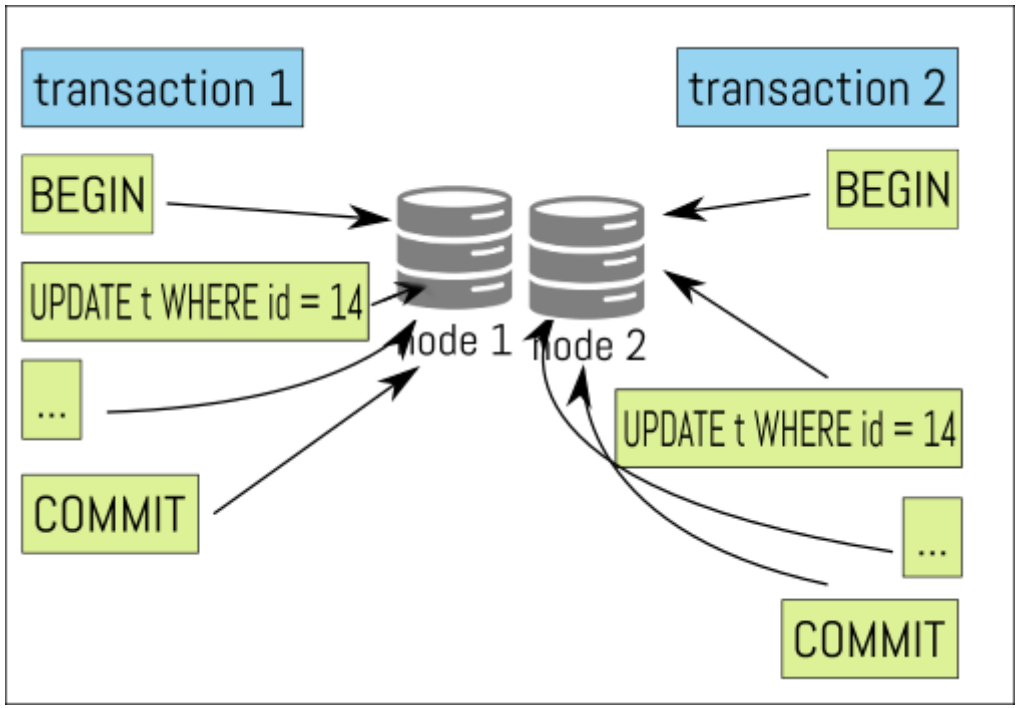
Optimistic Locking



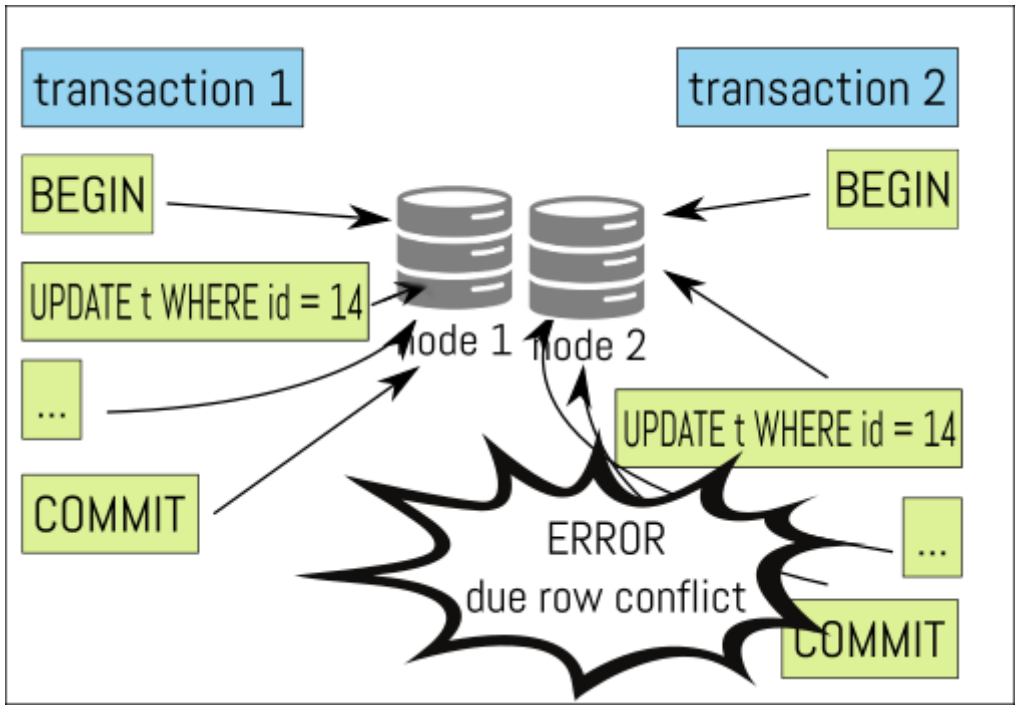
Optimistic Locking



Optimistic Locking



Optimistic Locking



The system returns error 149 as certification failed:

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

Group Replication : requirements

- exclusively use of InnoDB tables only

Group Replication : requirements

- exclusively use of InnoDB tables only
- every tables must have a PK defined

Group Replication : requirements

- exclusively use of InnoDB tables only
- every tables must have a PK defined
- only IPV4 is supported

Group Replication : requirements

- exclusively use of 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 use of 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 use of 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.

are we compatible ?

Pre-study

Is my workload ready for Group Replication ?

- Large transactions are more vulnerable because:

Is my workload ready for Group Replication ?

- Large transactions are more vulnerable because:
  - Large write set may increase the likelihood of certification conflicts

Is my workload ready for Group Replication ?

- Large transactions are more vulnerable because:
  - Large write set may increase the likelihood of certification conflicts
  - Transaction taking longer to execute may be more vulnerable to be aborted by a certified transaction.

Is my workload ready for Group Replication ?

- Large transactions are more vulnerable because:
  - Large write set may increase the likelihood of certification conflicts
  - Transaction taking longer to execute may be more vulnerable to be aborted by a certified transaction.
- Small transactions changing a very small set of data (hotspots):

Is my workload ready for Group Replication ?

- Large transactions are more vulnerable because:
  - Large write set may increase the likelihood of certification conflicts
  - Transaction taking longer to execute may be more vulnerable to be aborted by a certified transaction.
- Small transactions changing a very small set of data (hotspots):
  - Concurrently executing them at different sites will trade-off contention with roll backs due to the optimistic execution

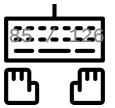
Therefore, when using Group Replication, we should pay attention to these points:

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

Scheme verification

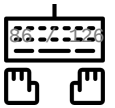
We will see now a list a queries that allows us to analyze our scheme. We will look for:

- non InnoDB tables
- tables without PK
- tables with a bad PK



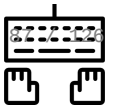
Non InnoDB tables

```
SELECT table_schema, table_name, engine, table_rows,
       (index_length+data_length)/1024/1024 AS sizeMB
FROM information_schema.tables
WHERE engine != 'innodb'
  AND table_schema NOT IN
    ('information_schema', 'mysql', 'performance_schema');
```



Tables Without Primary Key

```
select tables.table_schema, tables.table_name,
       tables.engine, tables.table_rows
from information_schema.tables
left join (
  select table_schema, table_name
  from information_schema.statistics
  group by table_schema, table_name, index_name
  having
  sum(
     case
      when non_unique = 0
           and nullable != 'YES' then 1
      else 0
    end
  ) = count(*)
) puks
on tables.table_schema = puks.table_schema
and tables.table_name = puks.table_name
where puks.table_name is null
and tables.table_type = 'BASE TABLE' and engine='InnoDB';
```



Tables with bad primarky keys

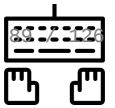
```
set SQL_MODE="";
select a.TABLE_SCHEMA,a.TABLE_NAME, b.ENGINE, a.COLUMN_NAME,
       a.DATA_TYPE, a.COLUMN_TYPE, a.COLUMN_KEY, b.TABLE_ROWS
from information_schema.COLUMNS as a
join information_schema.TABLES as b
  on b.TABLE_NAME=a.TABLE_NAME and b.TABLE_SCHEMA=a.TABLE_SCHEMA
where COLUMN_KEY='PRI' and ENGINE="InnoDB"
  and DATA_TYPE not like '%int'
  and DATA_TYPE not like 'enum%'
  and DATA_TYPE not like 'date%'
  and DATA_TYPE not like 'time%'
  and a.TABLE_SCHEMA not in ('mysql','sys')
group by table_schema, table_name;
```

Workload analysis

Now it's time to play with *Performance_Schema* to analyze our workload.

The goal is to identify:

- transactions with most statements (and most writes in particular)
- transactions with most rows affected
- largest statements by row affected
- queries updating most the same PK and therefore having to wait more)

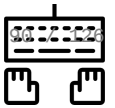


Setup Performance_Schema

We need to enable some consumers and instruments:

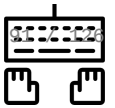
```
update performance_schema.setup_consumers
 set enabled = 'yes'
 where name like 'events_statement%' or name like 'events_transaction%';

update performance_schema.setup_instruments
 set enabled = 'yes', timed = 'yes'
 where name = 'transaction';
```



Find the transactions with most statements

```
select t.thread_id, t.event_id, count(*) statement_count,
       sum(s.rows_affected) rows_affected,
       length(replace(group_concat(
         case when s.event_name = "statement/sql/update" then 1
              when s.event_name = "statement/sql/insert" then 1
              when s.event_name = "statement/sql/delete" then 1
              else null end),',',''))
         as "# write statements"
from performance_schema.events_transactions_history_long t
join performance_schema.events_statements_history_long s
  on t.thread_id = s.thread_id and t.event_id = s.nesting_event_id
group by t.thread_id, t.event_id order by rows_affected desc limit 10;
```

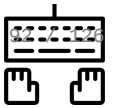


Find the transactions with most statements

It's possible to see what are all these statements in one transaction:

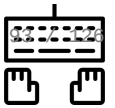
```
set group_concat_max_len = 1000000;

select t.thread_id, t.event_id, count(*) statement_count,
       sum(s.rows_affected) rows_affected,
       group_concat(sql_text order by s.event_id separator '\n') statements
from performance_schema.events_transactions_history_long t
join performance_schema.events_statements_history_long s
  on t.thread_id = s.thread_id and t.event_id = s.nesting_event_id
group by t.thread_id, t.event_id order by statement_count desc limit 1\G
```



Find the transactions with most rows affected

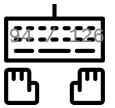
```
select t.thread_id, t.event_id, count(*) statement_count,
       sum(s.rows_affected) rows_affected,
       length(replace(group_concat(
       case
         when s.event_name = "statement/sql/update" then 1
         when s.event_name = "statement/sql/insert" then 1
         when s.event_name = "statement/sql/delete" then 1
         else null end),',','')) as "# write statements"
from performance_schema.events_transactions_history_long t
join performance_schema.events_statements_history_long s
  on t.thread_id = s.thread_id and t.event_id = s.nesting_event_id
group by t.thread_id, t.event_id order by rows_affected desc limit 10;
```



Find the transactions with most rows affected

It's again possible the see what are the statements:

```
select t.thread_id, t.event_id, count(*) statement_count,
       sum(s.rows_affected) rows_affected,
       group_concat(sql_text order by s.event_id separator '\n') statements
from performance_schema.events_transactions_history_long t
join performance_schema.events_statements_history_long s
  on t.thread_id = s.thread_id and t.event_id = s.nesting_event_id
group by t.thread_id, t.event_id order by rows_affected desc limit 1\G
```

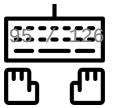


Find the transactions with most rows affected

It's again possible the see what are the statements:

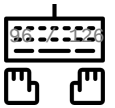
```
select t.thread_id, t.event_id, count(*) statement_count,
       sum(s.rows_affected) rows_affected,
       group_concat(sql_text order by s.event_id separator '\n') statements
from performance_schema.events_transactions_history_long t
join performance_schema.events_statements_history_long s
  on t.thread_id = s.thread_id and t.event_id = s.nesting_event_id
group by t.thread_id, t.event_id order by rows_affected desc limit 1\G
```

Don't forget to verify the **auto_commit** ones are they are not returned with the query above.



Find largest statements by row affected

```
select query, db, rows_affected, rows_affected_avg
from sys.statement_analysis
order by rows_affected_avg desc limit 10;
```



Find queries that write the most.

Looking for lots of updates/timer_wait to the same PKs.

```
select *
from performance_schema.events_statements_history_long
where rows_affected > 1 order by timer_wait desc limit 20\G
```

help me to configure my InnoDB Cluster

Using new cool tools

MySQL Shell

The MySQL Team extended MySQL Shell to use the new Admin API.

Now MySQL Shell is a single unified client for all adminstrative and operatons tasks.

- Multi-Language:

MySQL Shell

The MySQL Team extended MySQL Shell to use the new Admin API.

Now MySQL Shell is a single unified client for all adminstrative and operatons tasks.

- Multi-Language:
  - JavaScript

MySQL Shell

The MySQL Team extended MySQL Shell to use the new Admin API.

Now MySQL Shell is a single unified client for all adminstrative and operatons tasks.

- Multi-Language:
  - JavaScript
  - Python

MySQL Shell

The MySQL Team extended MySQL Shell to use the new Admin API.

Now MySQL Shell is a single unified client for all adminstrative and operatons tasks.

- Multi-Language:
  - JavaScript
  - Python
  - SQL

MySQL Shell

The MySQL Team extended MySQL Shell to use the new Admin API.

Now MySQL Shell is a single unified client for all adminstrative and operatons tasks.

- Multi-Language:
  - JavaScript
  - Python
  - SQL
- Supports both Document and Relational models

MySQL Shell

The MySQL Team extended MySQL Shell to use the new Admin API.

Now MySQL Shell is a single unified client for all adminstrative and operatons tasks.

- Multi-Language:
  - JavaScript
  - Python
  - SQL
- Supports both Document and Relational models
- Exposes full Development and **Admin API**

MySQL Shell: Admin API

- `mysql-js> dba.help()`

MySQL Shell: Admin API

- `mysql-js> dba.help()`
- the global variable *dba* is used to access the MySQL API

MySQL Shell: Admin API

- `mysql-js> dba.help()`
- the global variable *dba* is used to access the MySQL API
- perform DBA operations to manage MySQL InnoDB Cluster:

MySQL Shell: Admin API

- `mysql-js> dba.help()`
- the global variable *dba* is used to access the MySQL API
- perform DBA operations to manage MySQL InnoDB Cluster:
  - create cluster

MySQL Shell: Admin API

- `mysql-js> dba.help()`
- the global variable *dba* is used to access the MySQL API
- perform DBA operations to manage MySQL InnoDB Cluster:
  - create cluster
  - deploy MySQL instances (in sandbox only for the moment)

MySQL Shell: Admin API

- `mysql-js> dba.help()`
- the global variable *dba* is used to access the MySQL API
- perform DBA operations to manage MySQL InnoDB Cluster:
  - create cluster
  - deploy MySQL instances (in sandbox only for the moment)
  - get cluster info

MySQL Shell: Admin API

- `mysql-js> dba.help()`
- the global variable *dba* is used to access the MySQL API
- perform DBA operations to manage MySQL InnoDB Cluster:
  - create cluster
  - deploy MySQL instances (in sandbox only for the moment)
  - get cluster info
  - start/stop MySQL instances

MySQL Shell: Admin API

- `mysql-js> dba.help()`
- the global variable *dba* is used to access the MySQL API
- perform DBA operations to manage MySQL InnoDB Cluster:
  - create cluster
  - deploy MySQL instances (in sandbox only for the moment)
  - get cluster info
  - start/stop MySQL instances
  - validate MySQL instances

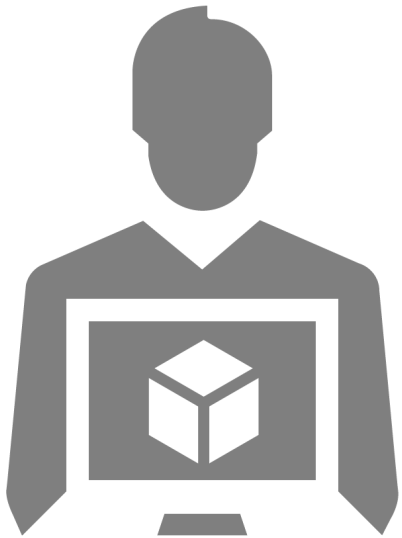
mysqlprovision

The mysqlprovision utility is designed to facilitate the management of MySQL Group Replication, allowing users to start a replica set (a new replication group), to add/remove members to/from an existing group. It can also check if an instance meets all the requirements to successfully create a new group or to be added to an existing group. This utility is also capable of modifying MySQL option files of existing instances in order for them to meet the requirements of Group Replication.

let's create our first MySQL InnoDB Cluster

Demo time !

MySQL InnoDB Cluster



demo

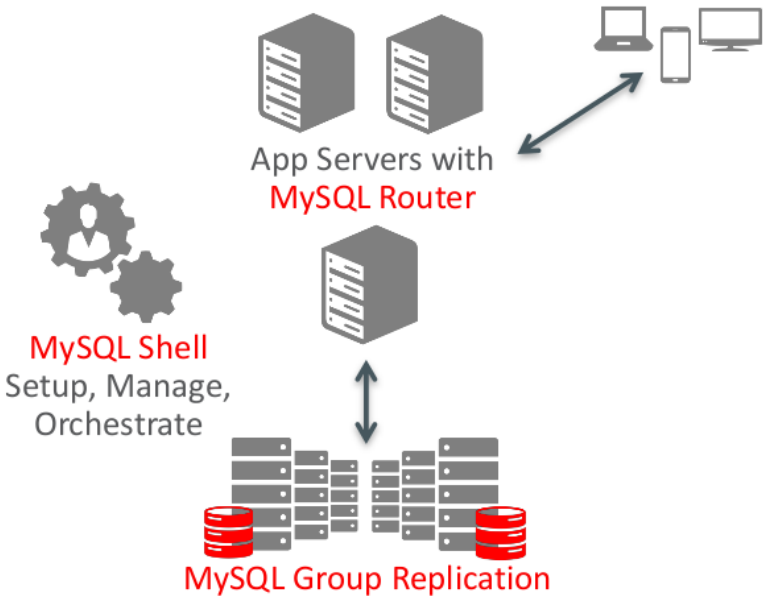
full HA ?

Application interaction

MySQL InnoDB Cluster

So MySQL InnoDB Cluster is composed by:

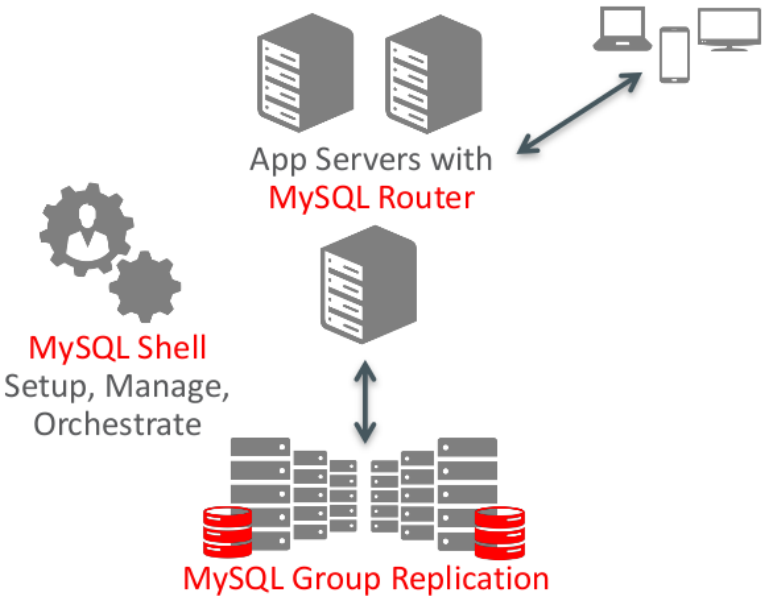
- MySQL Router
- MySQL Shell
- MySQL Group Replication
- MySQL Server



MySQL InnoDB Cluster

So MySQL InnoDB Cluster is composed by:

- MySQL Router
- MySQL Shell
- MySQL Group Replication
- MySQL Server



CON2907 - MySQL High Availability - *Matt Lord* - Wed 12:15

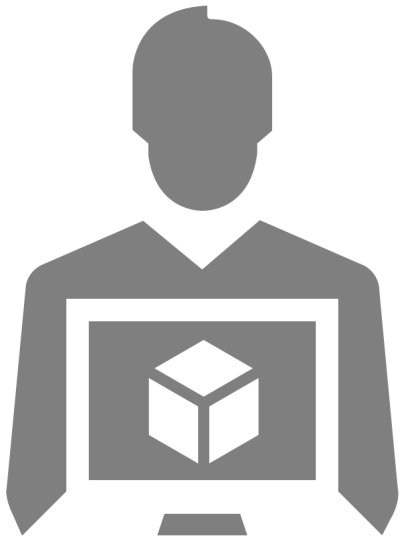
MySQL Router

- Transparent client connection routing (TCP level 4)
  - load balancing
  - connection failover
- Native support for InnoDB Clusters
  - understands Group Replication topology
  - utilizes metadata schema stored on each members
    - bootstrap and auto-config
    - supports multi-master and single primary modes

MySQL Router & Group Replication

Quick demo !

MySQL Router & Group Replication



demo

Any other alternatives ?

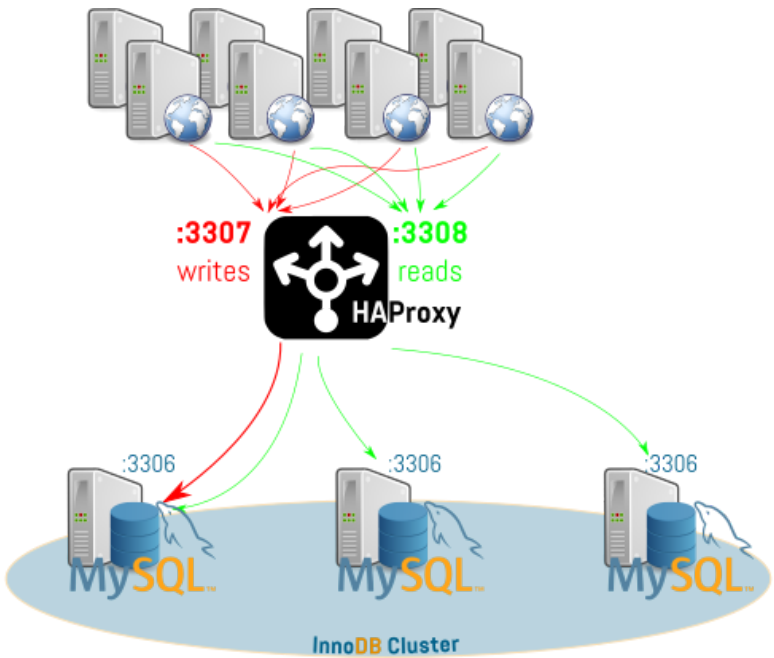
Any other alternatives ?

It's possible to use MySQL Group Replication with other proxies/load-balancers

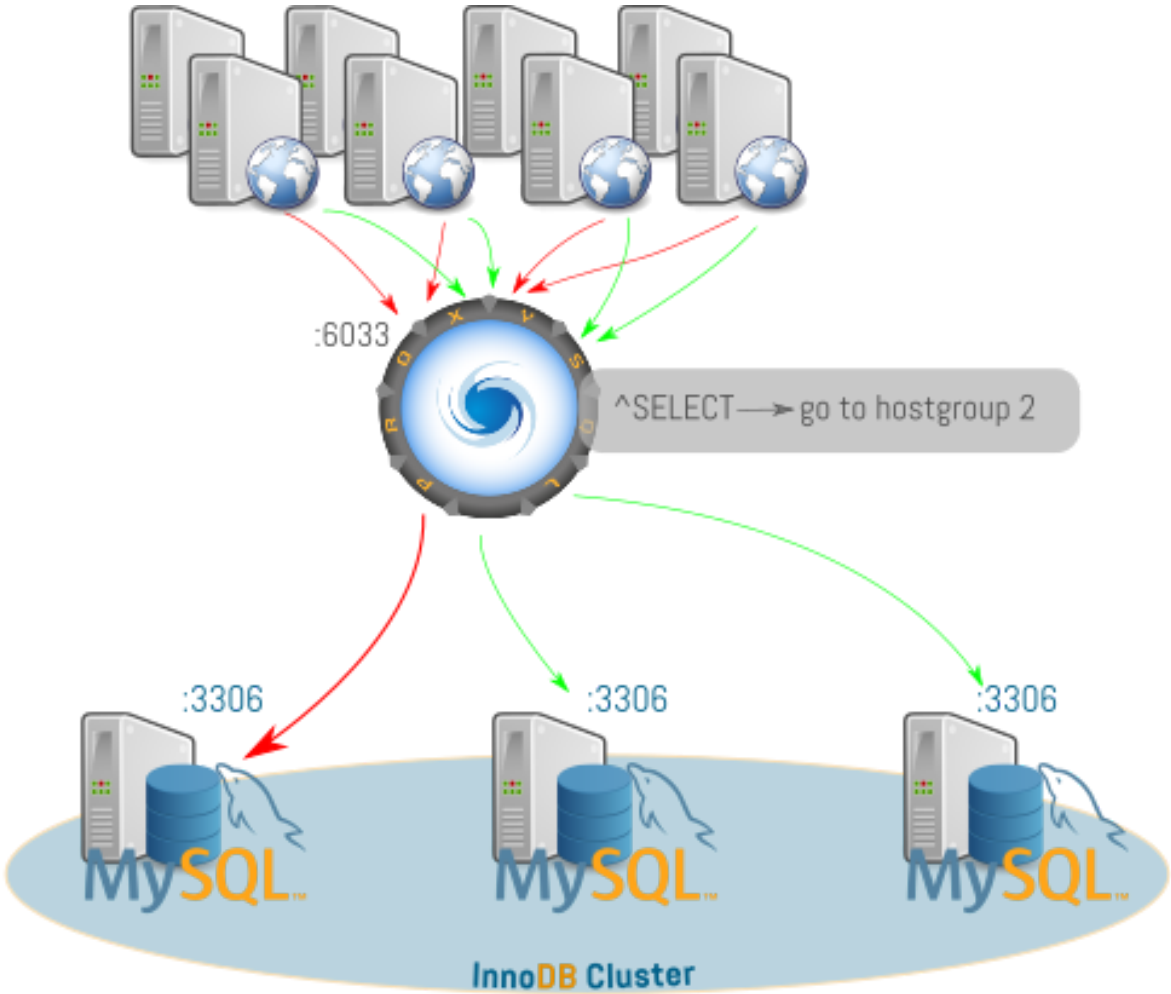
Any other alternatives ?

It's possible to use MySQL Group Replication with other proxies/load-balancers

- HAProxy : http://lefred.be/content/mysql-group-replication-as-ha-solution/



- ProxySQL : http://lefred.be/content/ha-with-mysql-group-replication-and-proxysql/



Thank you !

Questions ?

MySQL Community Reception @ Oracle OpenWorld

- Tuesday, September 20, 7.00 pm
- Jillian's at Metreon: 175 Fourth Street, San Francisco
  - At the corner of Howard and 4th st.; only 2-min walk from Moscone Center (same place as last year)

Join us!

Register: http://tinyurl.com/mysqloow16