Galera Replication Demystified

how does it work ?

about.me/lefred

Who am I ?

Frédéric Descamps

- @lefred

Frédéric Descamps

- @lefred
- Working for Percona since 2011

Frédéric Descamps

- @lefred
- Working for Percona since 2011
- Senior Architect

Frédéric Descamps

- @lefred
- Working for Percona since 2011
- Senior Architect
- Managing MySQL since 3.23

Frédéric Descamps

- @lefred
- Working for Percona since 2011
- Senior Architect
- Managing MySQL since 3.23
- devops believer

Frédéric Descamps

- @lefred
- Working for Percona since 2011
- Senior Architect
- Managing MySQL since 3.23
- devops believer
- and I installed my first Galera Cluster in February 2010 ;-)

Galera Replication

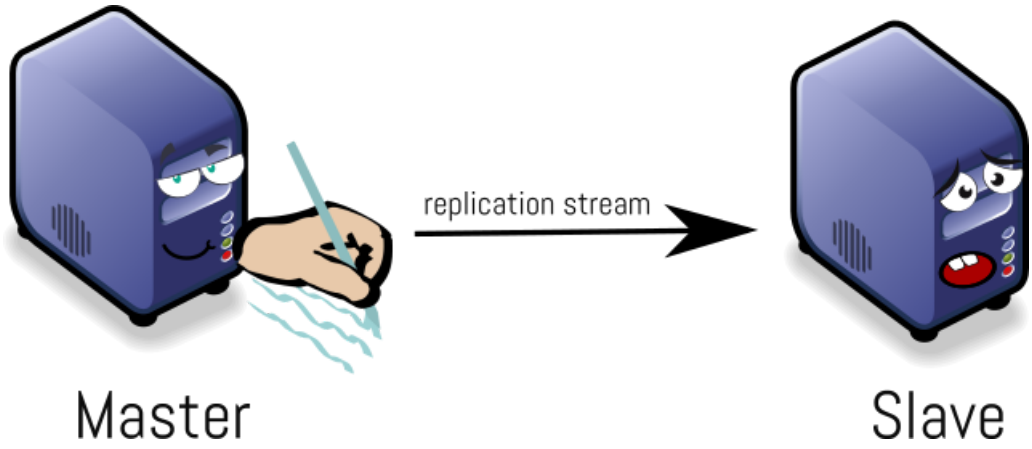
Cluster

Galera Replication - Cluster

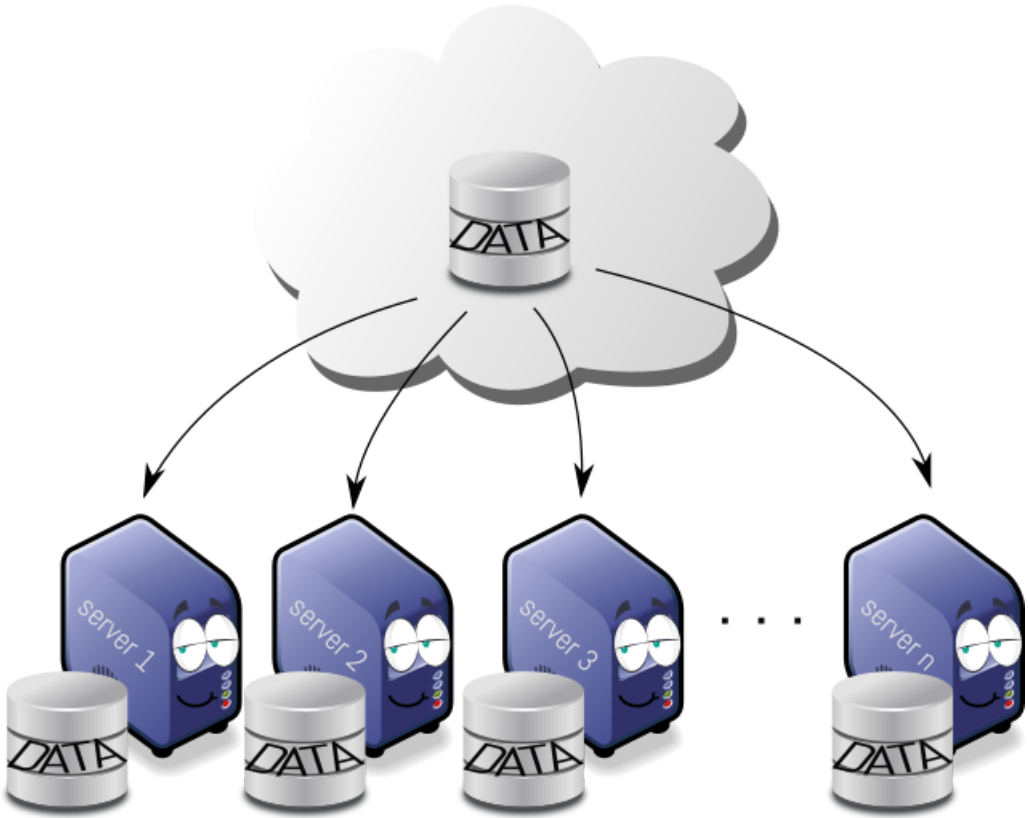
What is it ?

What does it handle ?

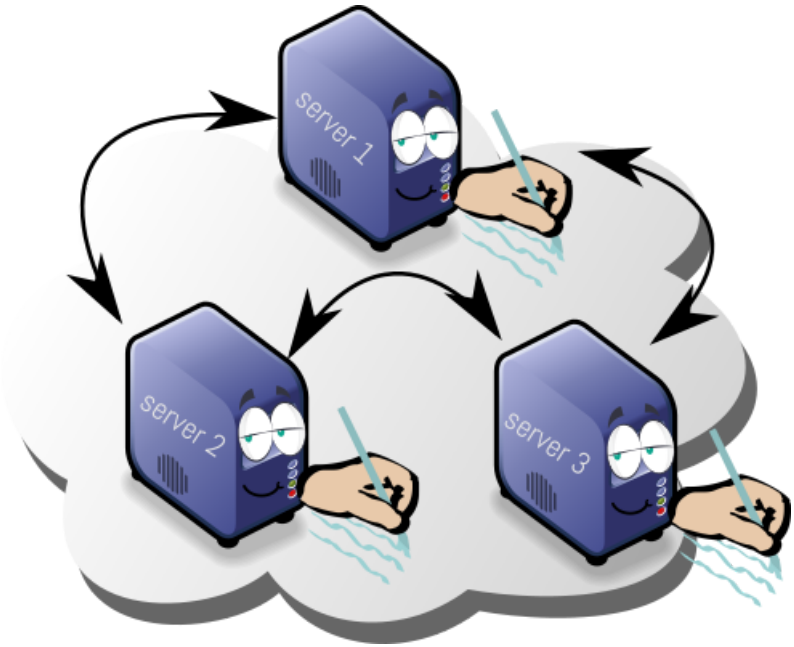
Standard asyncrhonous replication is **server-centric**, one server streams data to another one. All the nodes have a specific role.



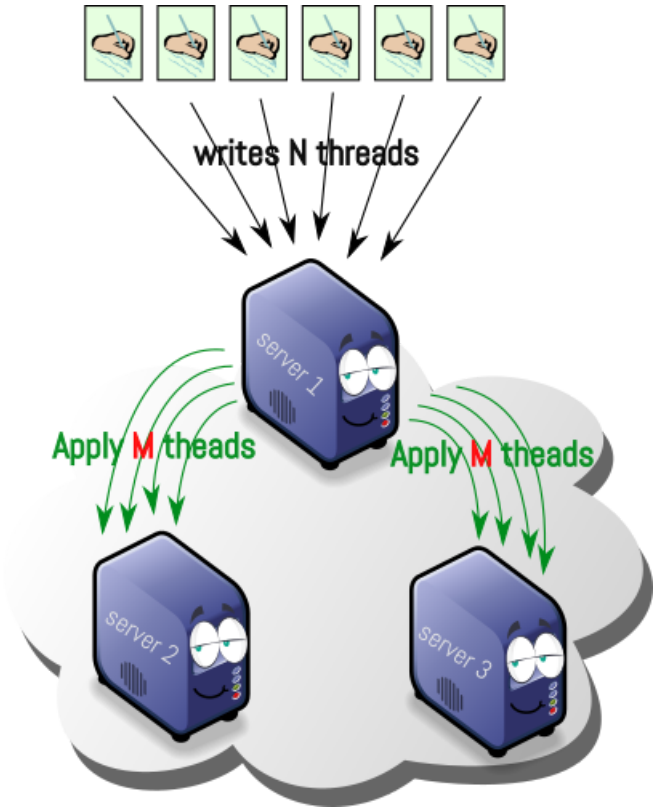
In Galera, the dataset is **synchronized** between one or more servers:
data-centric



You can write to any node in your cluster No need to worry about eventual out-of-sync



Write events/transactions are sent in parallel



Cluster Membership

- determined **by the cluster**
- *wsrep_cluster_address* is just a pointer
- any node is permitted to join that
  - knows the cluster name
  - can find a single active cluster node

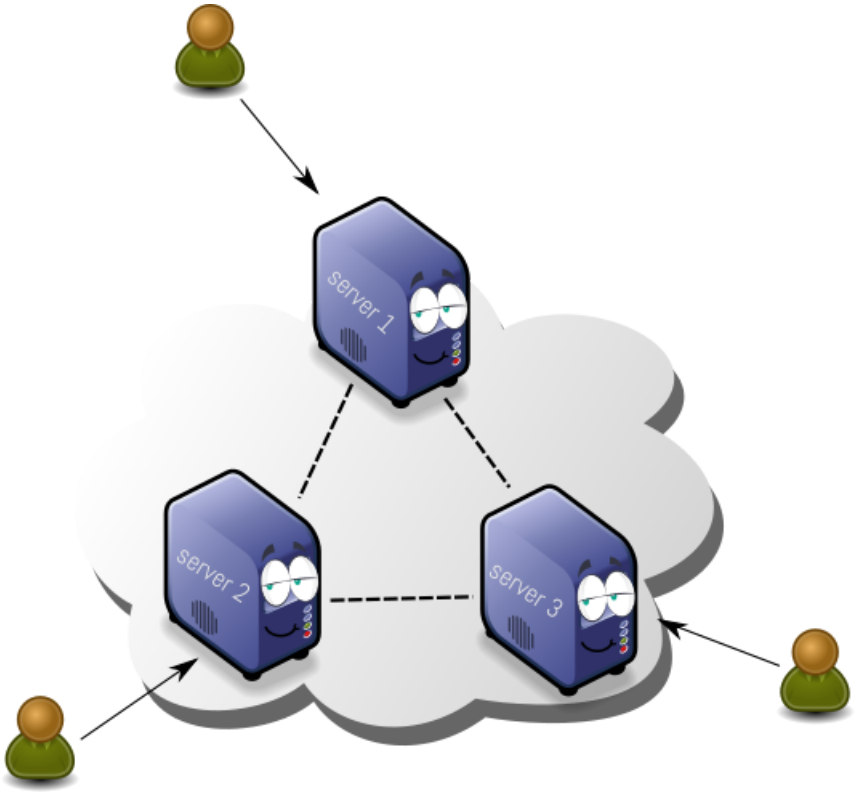
Cluster Membership

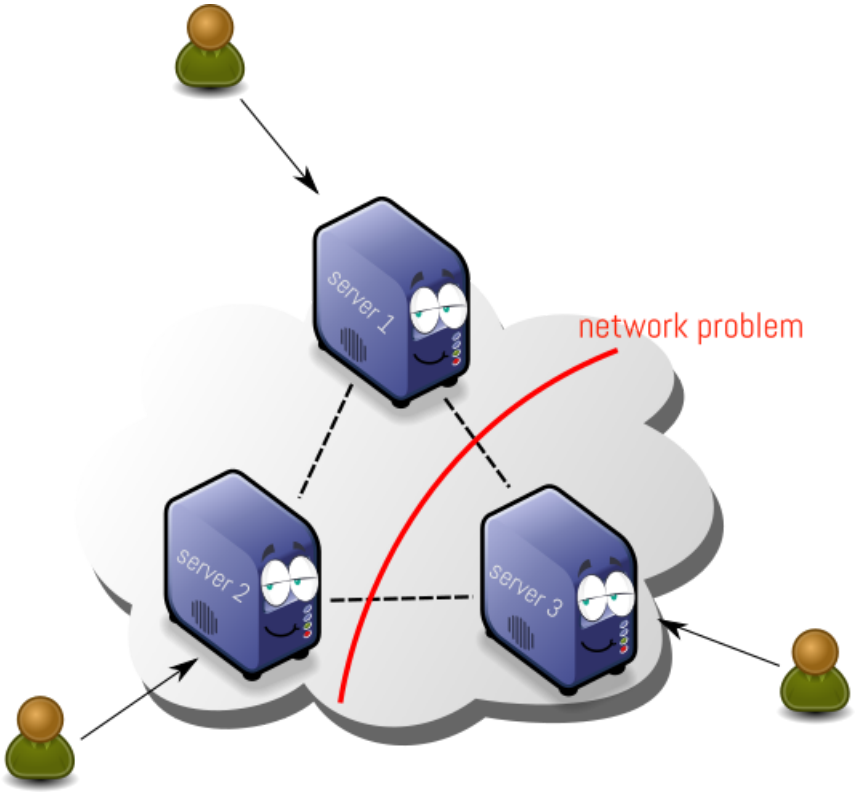
- determined **by the cluster**
- *wsrep_cluster_address* is just a pointer
- any node is permitted to join that
  - knows the cluster name
  - can find a single active cluster node

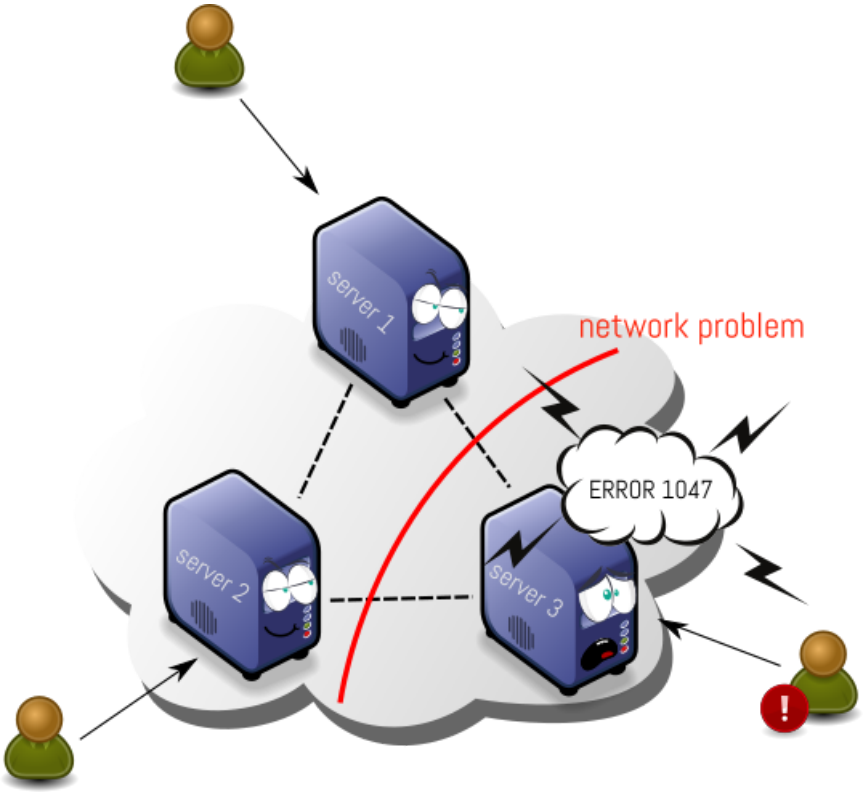
```
wsrep_cluster_address = gcomm://192.168.1.1,192.168.1.2,192.168.1.3
```

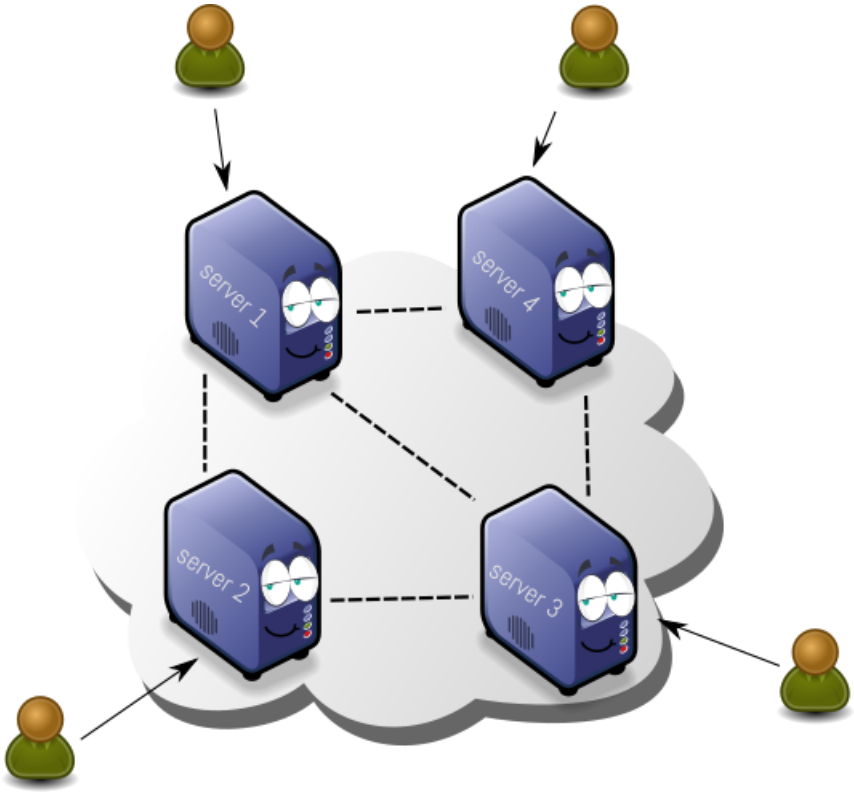
The cluster manages **quorum**

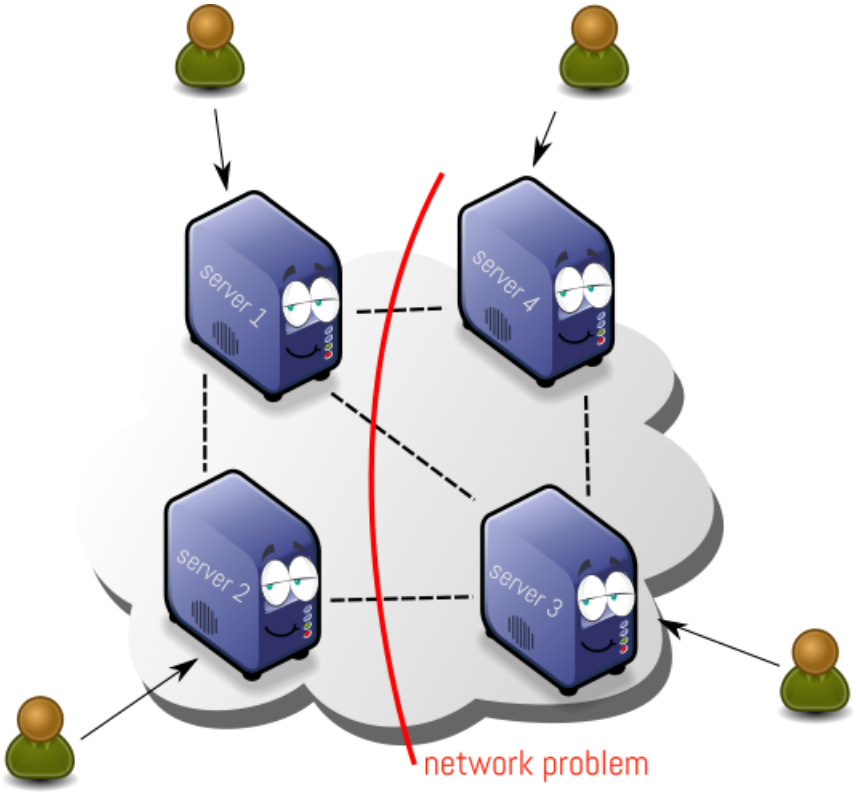
and has split-brain protection.

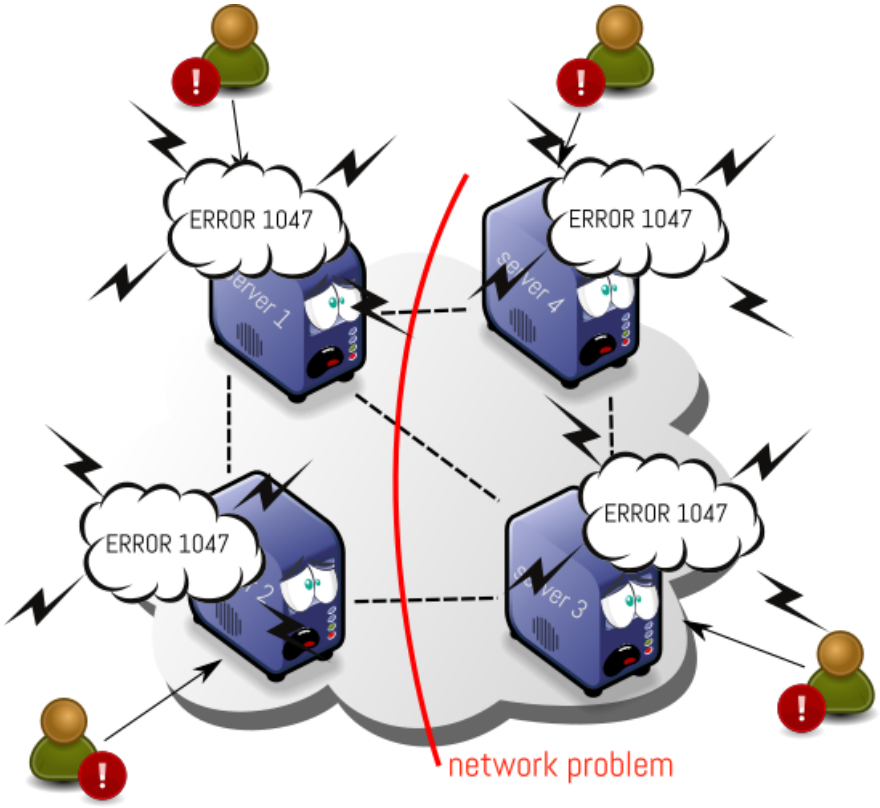

server 1
server 2
server 3


server 1
network problem
server 2
server 3


server 1
network problem
ERROR 1047
server 2
server 3
!


server 1
server 4
server 2
server 3


server 1
server 4
server 2
server 3
network problem


ERROR 1047
ERROR 1047
ERROR 1047
ERROR 1047
network problem

Replication

Replication

- Delivers the writeset to all nodes in the cluster

Replication

- Delivers the writeset to all nodes in the cluster
  - and all nodes acknowledge the writeset

Replication

- Delivers the writeset to all nodes in the cluster
  - and all nodes acknowledge the writeset

Replication

- Delivers the writeset to all nodes in the cluster
  - and all nodes acknowledge the writeset
- Cost is ~roundtrip latency to furthest node

Replication

- Delivers the writeset to all nodes in the cluster
  - and all nodes acknowledge the writeset
- Cost is ~roundtrip latency to furthest node
- Serialized by Group Communication

GTID

GTID

- Not the same as 5.6 Aynchronous GTID's

GTID

- Not the same as 5.6 Aynchronous GTID's
  - though they appear the same

GTID

- Not the same as 5.6 Aynchronous GTID's
  - though they appear the same
  - 939aac77-f7d1-11e3-bd5e-b211d6ab1ec6:1534285

GTID

- Not the same as 5.6 Aynchronous GTID's
  - though they appear the same
  - 939aac77-f7d1-11e3-bd5e-b211d6ab1ec6:1534285
- GTIDs ensure cluster members are consistent with each other

GTID

- Not the same as 5.6 Aynchronous GTID's
  - though they appear the same
  - 939aac77-f7d1-11e3-bd5e-b211d6ab1ec6:1534285
- GTIDs ensure cluster members are consistent with each other
  - nodes joining a cluster have their GTIDs checked

GTID

- Not the same as 5.6 Aynchronous GTID's
  - though they appear the same
  - 939aac77-f7d1-11e3-bd5e-b211d6ab1ec6:1534285
- GTIDs ensure cluster members are consistent with each other
  - nodes joining a cluster have their GTIDs checked
- GTIDs can be used to compare downed nodes to each other

GTID

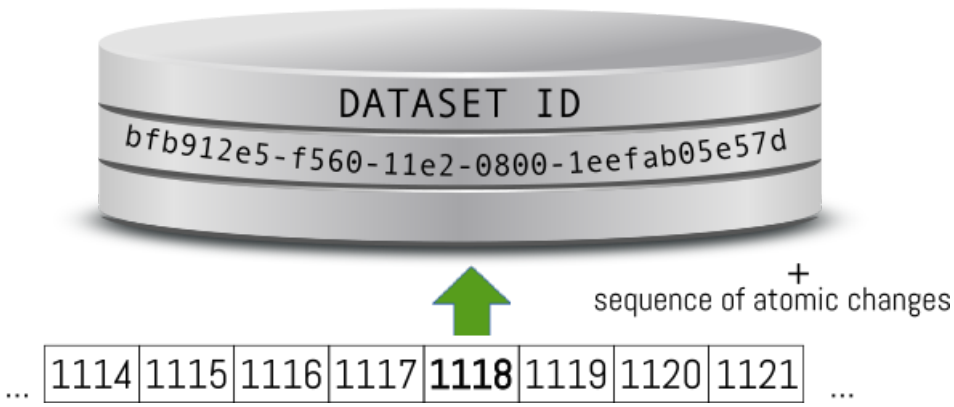
The highest GTID is the most recently written

Generally the best practice it to bootstrap the node with the most recent data

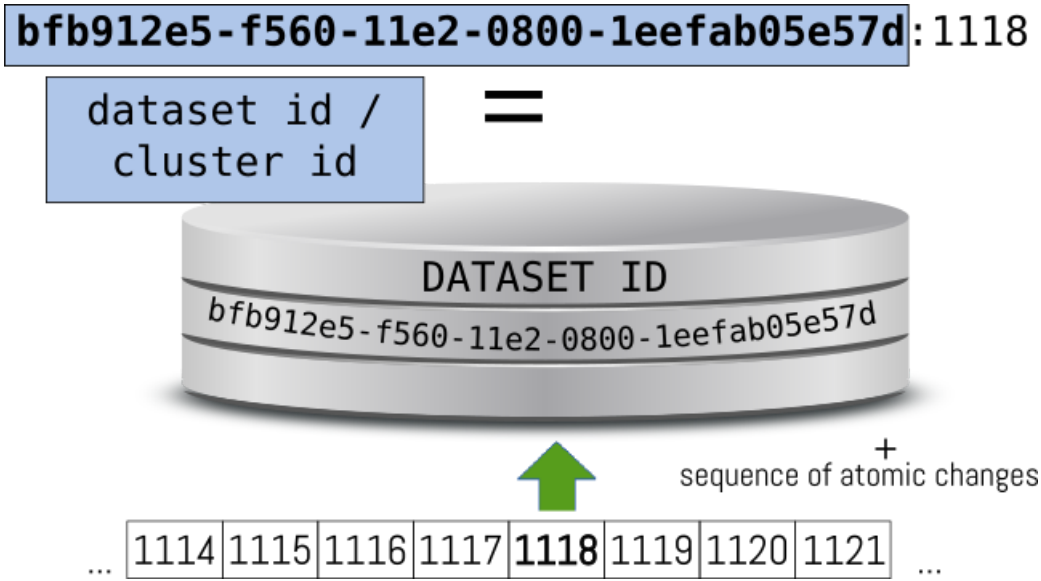
GTID

bfb912e5-f560-11e2-0800-1eefab05e57d:1118

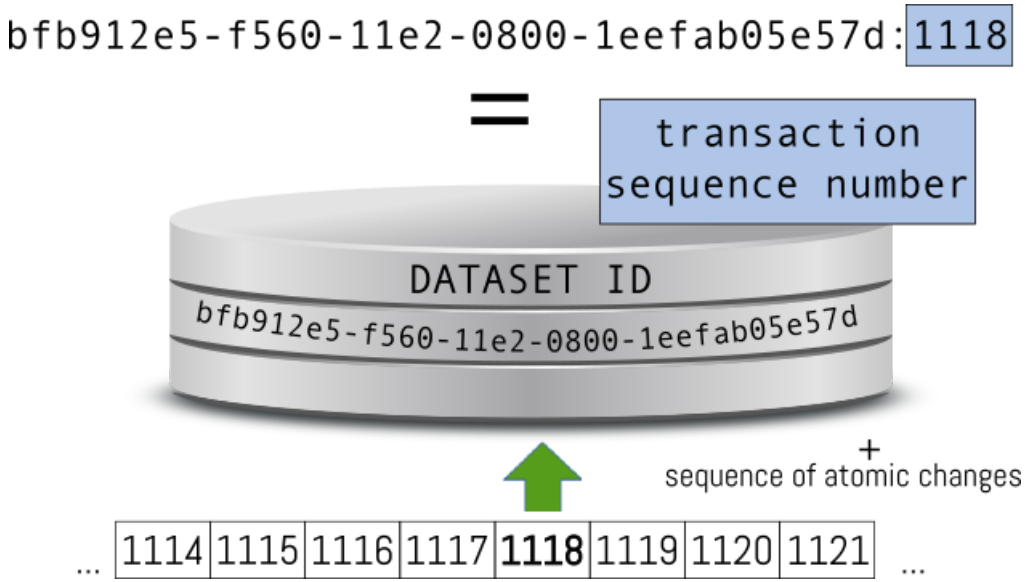
=



GTID


bfb912e5-f560-11e2-0800-1eefab05e57d:1118
dataset id / cluster id
=
DATASET ID
bfb912e5-f560-11e2-0800-1eefab05e57d
+
sequence of atomic changes
... 1114 1115 1116 1117 1118 1119 1120 1121 ...

GTID


bfb912e5-f560-11e2-0800-1eefab05e57d:1118
=
transaction sequence number
DATASET ID
bfb912e5-f560-11e2-0800-1eefab05e57d
+
sequence of atomic changes
... 1114 1115 1116 1117 1118 1119 1120 1121 ...

Global Transaction IDs

- initial dataset
  - bfb912e5-f560-11e2-0800-1eefab05e57d:**0**

Global Transaction IDs

- initial dataset
  - bfb912e5-f560-11e2-0800-1eefab05e57d:**0**
- first change/transaction/writeset
  - bfb912e5-f560-11e2-0800-1eefab05e57d:**1**

Global Transaction IDs

- initial dataset
  - bfb912e5-f560-11e2-0800-1eefab05e57d:**0**
- first change/transaction/writeset
  - bfb912e5-f560-11e2-0800-1eefab05e57d:**1**
- undefined GTID
  - 00000000-0000-0000-0000-000000000000:**-1**

Global Transaction IDs : Galera vs MySQL 5.6

Galera GTID :

```
bfb912e5-f560-11e2-0800-1eefab05e57d:1118
```

MySQL 5.6 GTID :


```
3e11fa47-71ca-11e2-9e33-c80aa9429562:23
```

Global Transaction IDs : Galera vs MySQL 5.6

Galera GTID :

`bfb912e5-f560-11e2-0800-1eefab05e57d:1118`

dataset / cluster ID

MySQL 5.6 GTID :

`3e11fa47-71ca-11e2-9e33-c80aa9429562:23`

server ID

Global Transaction IDs : Galera vs MySQL 5.6

Galera GTID :

`bfb912e5-f560-11e2-0800-1eefab05e57d`:`1118`

dataset / cluster ID

data change inside the cluster

MySQL 5.6 GTID :

`3e11fa47-71ca-11e2-9e33-c80aa9429562`:`23`

server ID

trx processed by the server

Global Transaction IDs : Galera vs MySQL 5.6

In MySQL 5.6

```
9a511b7b-7059-11e2-9a24-08002762b8af:32
9a511b7b-7059-11e2-9a24-08002762b8af:33
9a511b7b-7059-11e2-9a24-08002762b8af:34
[ new master promoted ]
3e11fa47-71ca-11e2-9e33-c80aa9429562:1
3e11fa47-71ca-11e2-9e33-c80aa9429562:2
3e11fa47-71ca-11e2-9e33-c80aa9429562:3
```

Global Transaction IDs : Galera vs MySQL 5.6

In Galera

```
bfb912e5-f560-11e2-0800-1eefab05e57da:1118
bfb912e5-f560-11e2-0800-1eefab05e57da:1119
bfb912e5-f560-11e2-0800-1eefab05e57da:1120
```

Global Transaction IDs : Galera vs MySQL 5.6

In Galera

```
bfb912e5-f560-11e2-0800-1eefab05e57da:1118
bfb912e5-f560-11e2-0800-1eefab05e57da:1119
bfb912e5-f560-11e2-0800-1eefab05e57da:1120
[ new master promoted ]
```

Global Transaction IDs : Galera vs MySQL 5.6

In Galera

```
bfb912e5-f560-11e2-0800-1eefab05e57da:1118
bfb912e5-f560-11e2-0800-1eefab05e57da:1119
bfb912e5-f560-11e2-0800-1eefab05e57da:1120
[ new master promoted ]
bfb912e5-f560-11e2-0800-1eefab05e57da:1121
bfb912e5-f560-11e2-0800-1eefab05e57da:1122
bfb912e5-f560-11e2-0800-1eefab05e57da:1123
```

GTID Assignment

UUID

The **UUID** section, 128-bit, is generated during bootstrapping to identify the cluster.

Generated by mixing the timer value and pseudo-random numbers (depending primarily from the timer and PID), but, currently, there is no use of NIC's MAC-address although in theory it may be done that way.

More info: https://gist.github.com/lefred/88a2cec88d03854d9934

GTID Assignment (3)

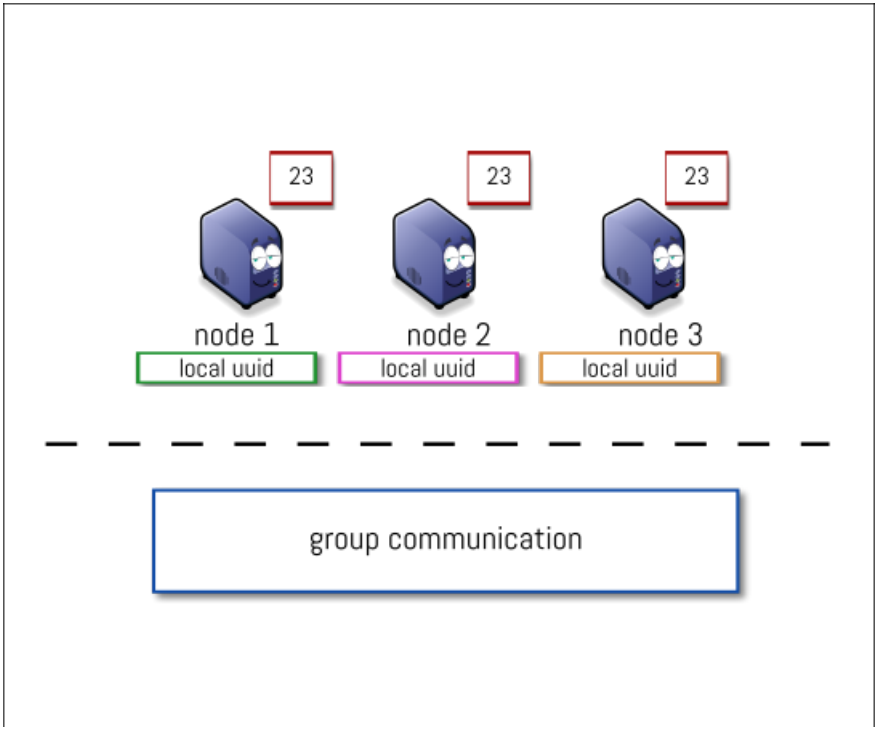
SEQNO

The **seqno**, 64-bit, is incremented only when the transaction passes certification and is ready for commit.

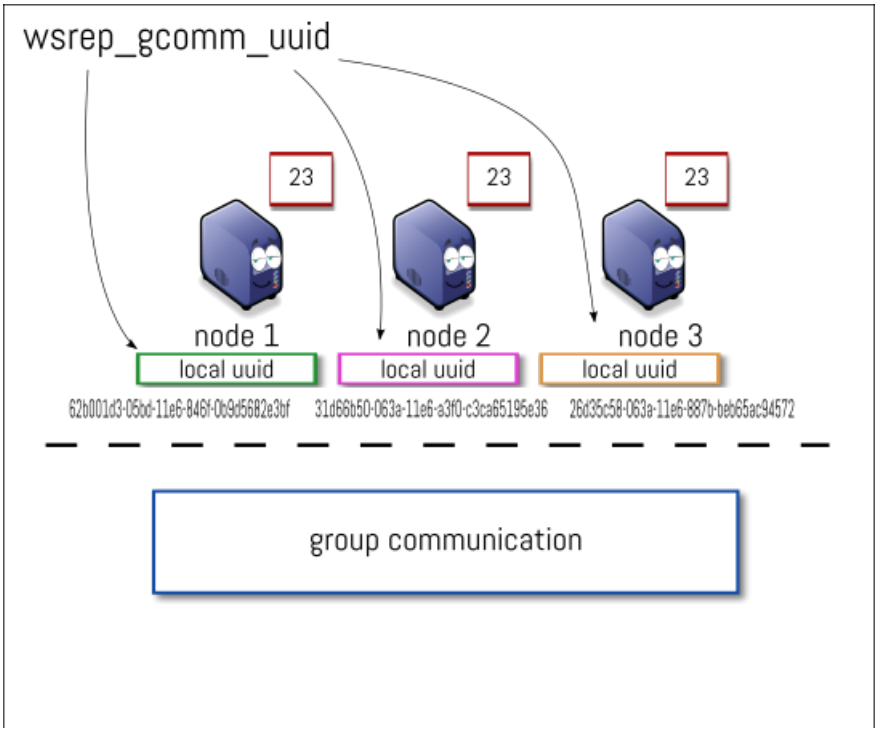
For the curious, the algorithm used to derive sequence number is Totem Single-ring Ordering protocol.

Before that, there is already communication between the nodes, *group communication* is used to define a group-channel id which is a locally maintained counter by each node in sync with the group.

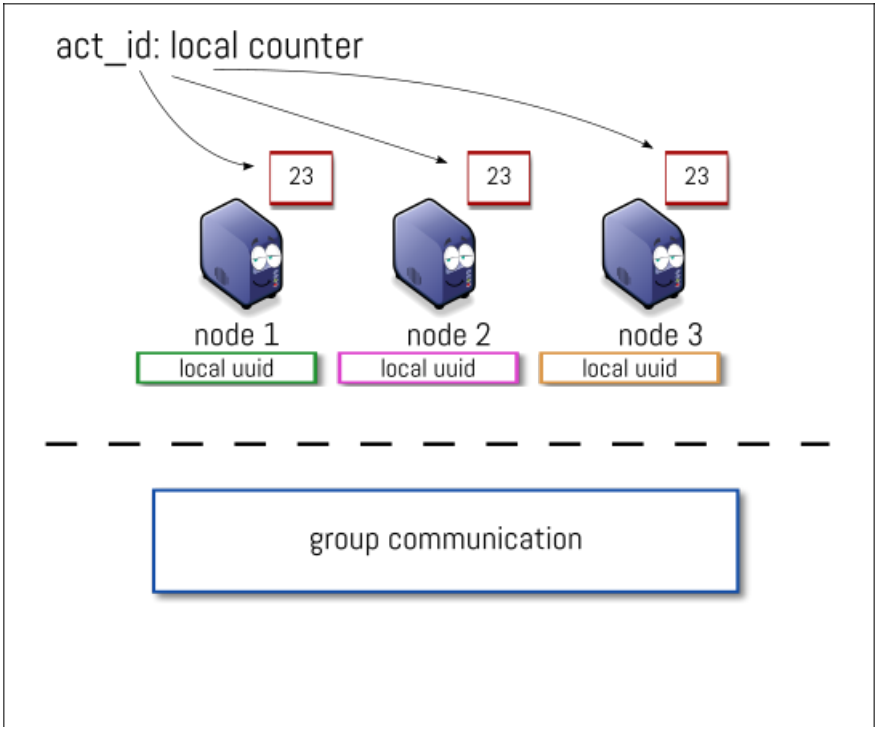
Serialization of writesets



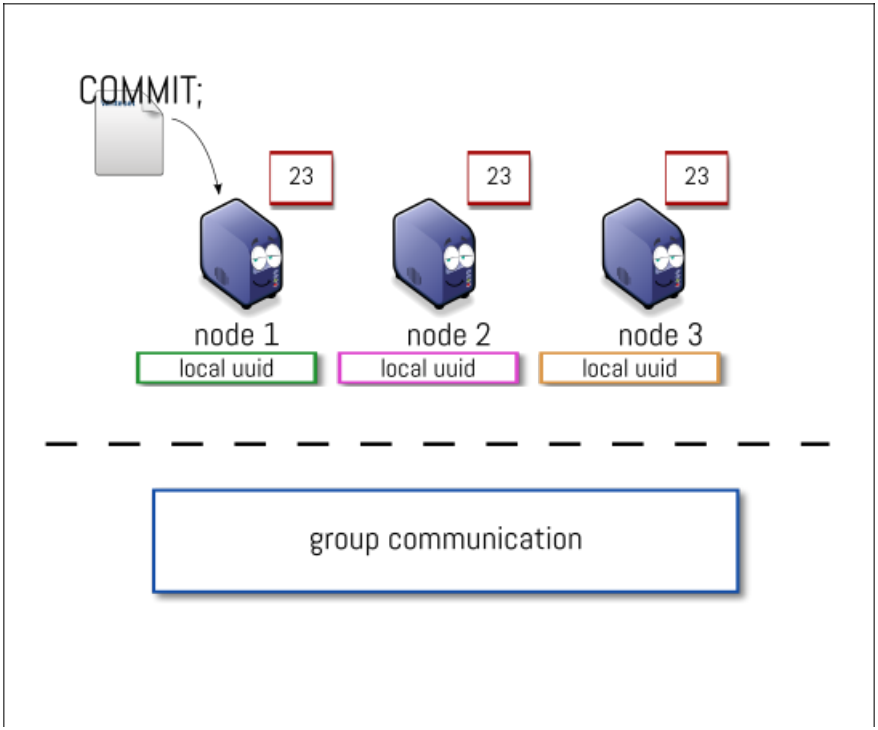
Serialization of writesets



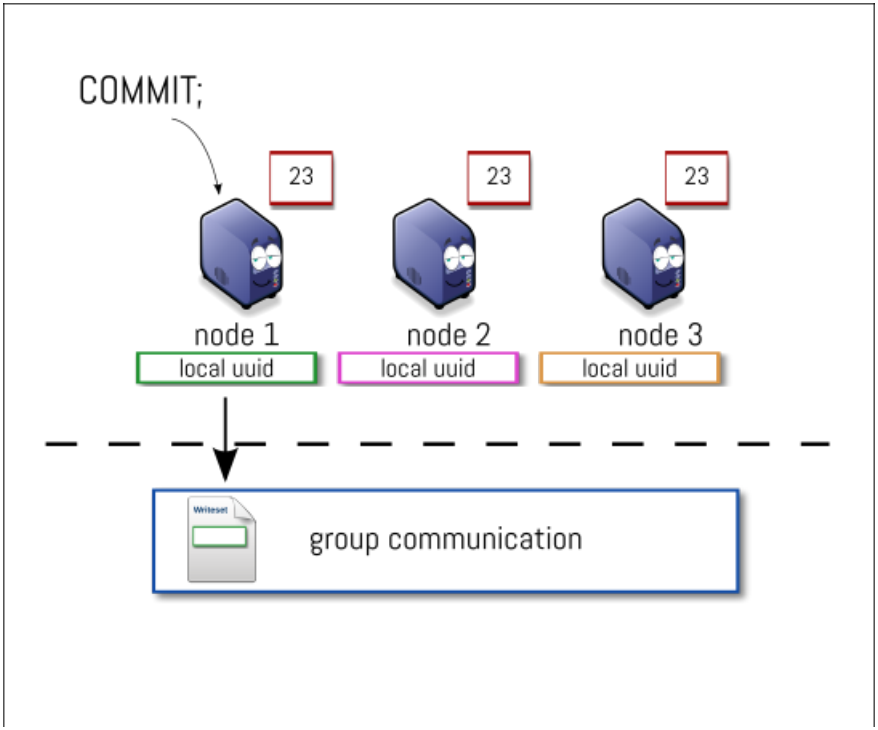
Serialization of writesets



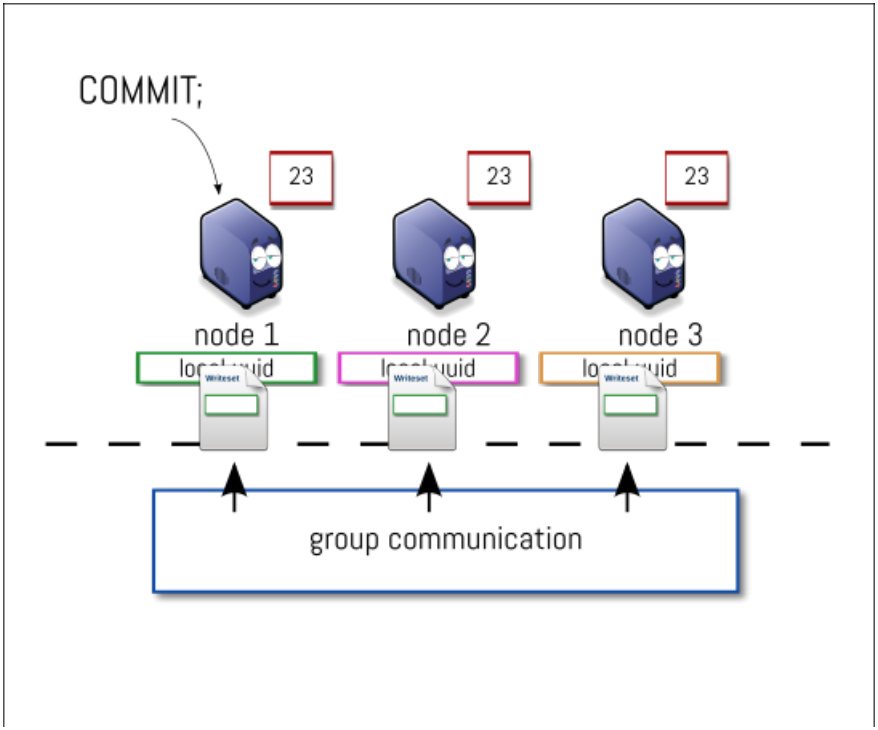
Serialization of writesets



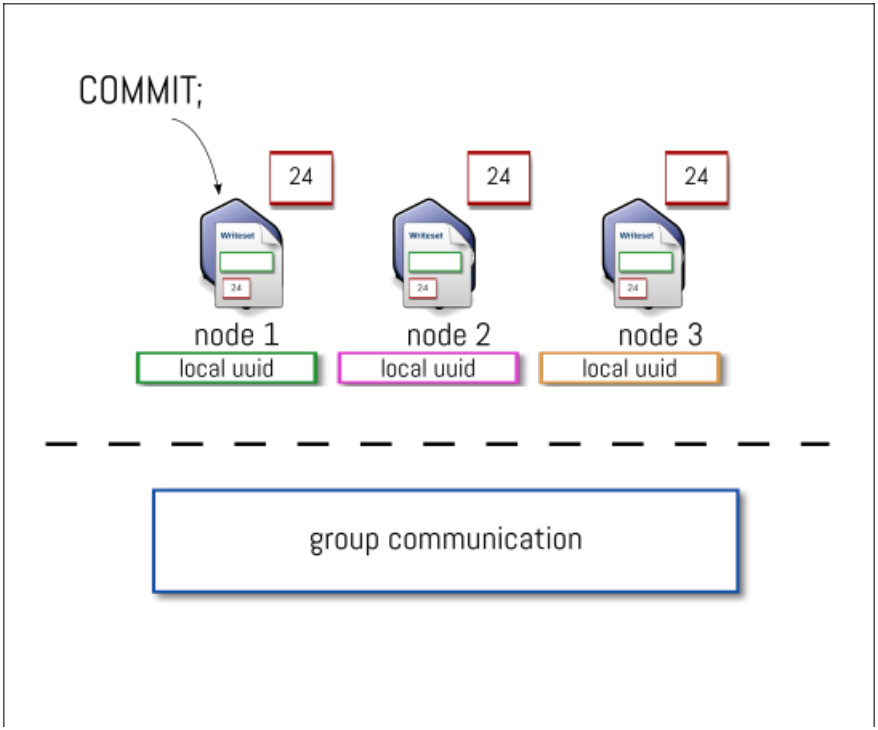
Serialization of writesets



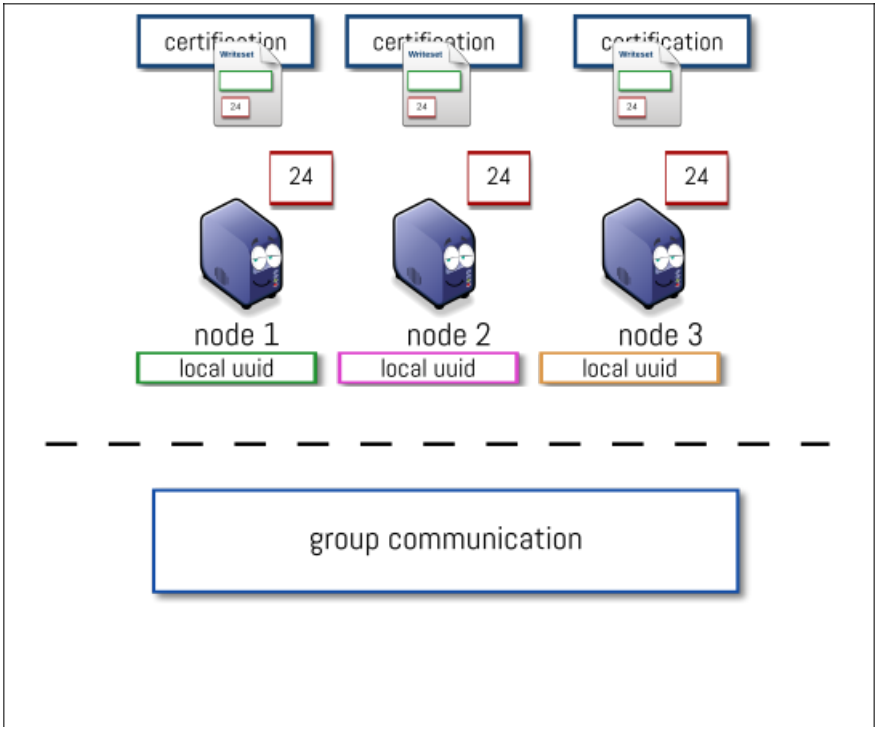
Serialization of writesets



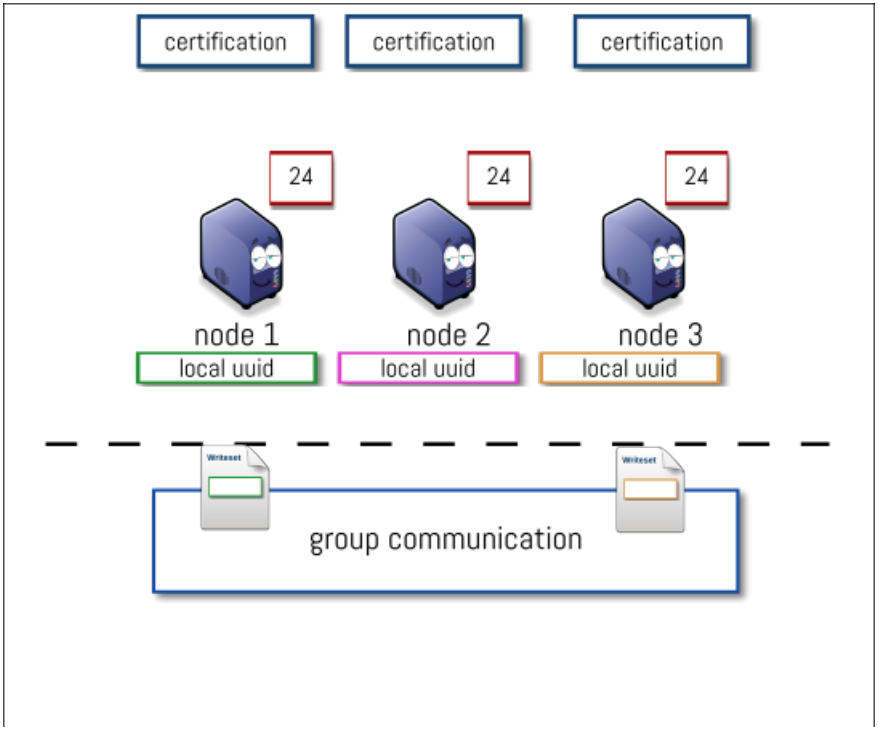
Serialization of writesets



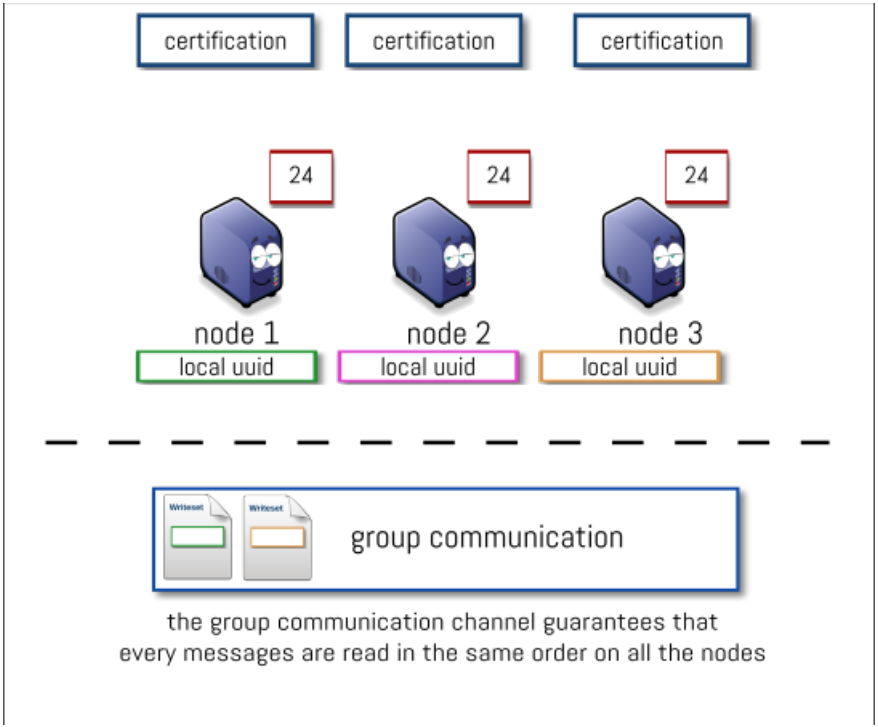
Serialization of writesets



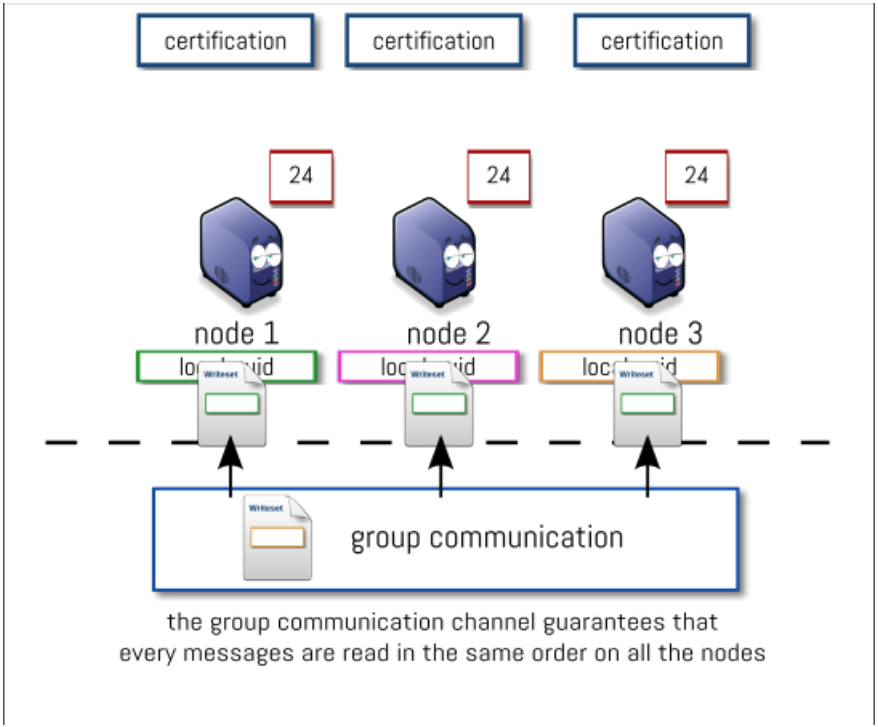
Serialization of writesets



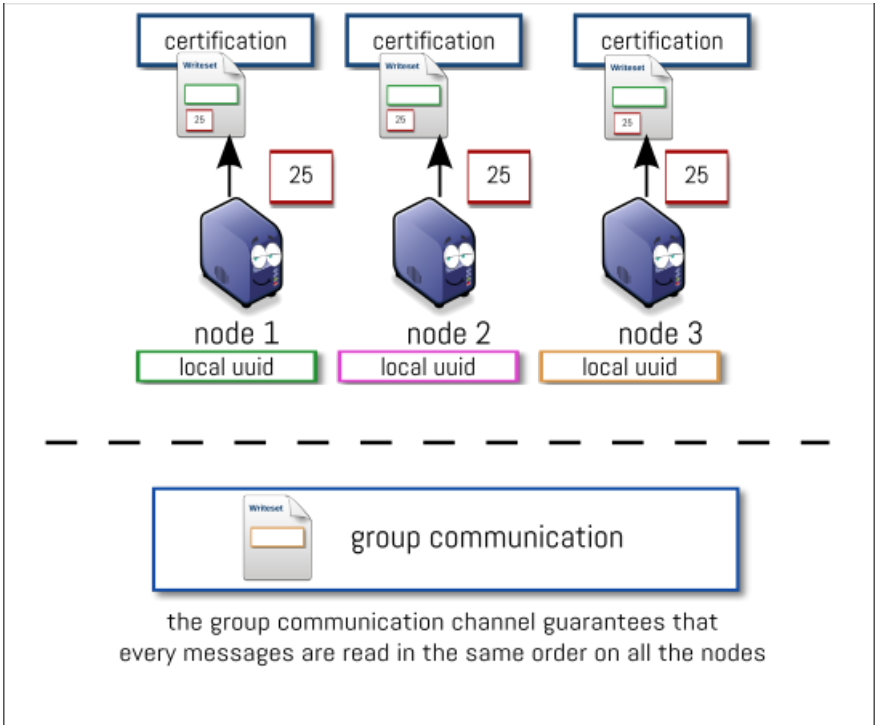
Serialization of writesets



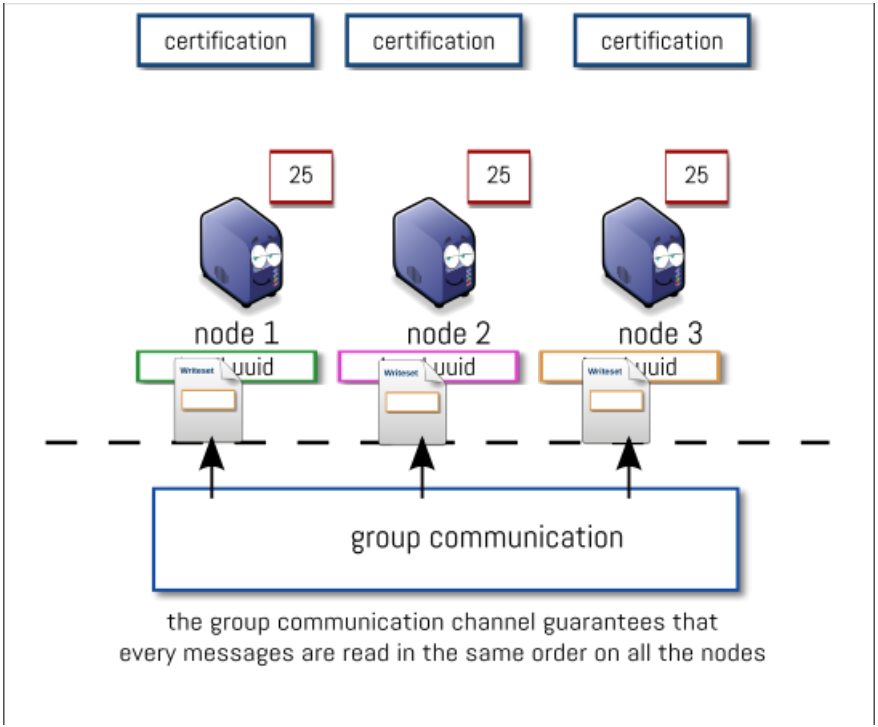
Serialization of writesets



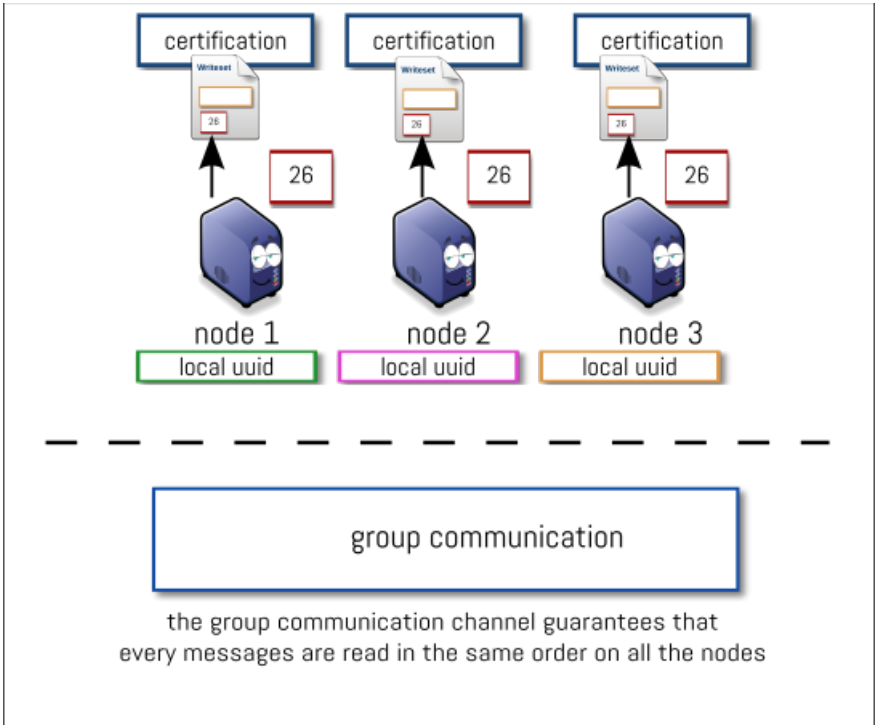
Serialization of writesets



Serialization of writesets



Serialization of writesets



Roles

We have 4 distinct roles in Galera:

- 2 for replication
- 2 for state transfer

Replication Roles

- Within the cluster, all nodes are equal

Replication Roles

- Within the cluster, all nodes are equal
- '**master/donor** node'

Replication Roles

- Within the cluster, all nodes are equal
- '**master/donor** node'
  - The node a given transaction was written and committed on.

Replication Roles

- Within the cluster, all nodes are equal
- '**master/donor** node'
  - The node a given transaction was written and committed on.
- '**slave/joiner** node'

Replication Roles

- Within the cluster, all nodes are equal
- '**master/donor** node'
  - The node a given transaction was written and committed on.
- '**slave/joiner** node'
  - The node that received the given transaction via Galera replication.

Replication Roles

- Within the cluster, all nodes are equal
- '**master/donor** node'
  - The node a given transaction was written and committed on.
- '**slave/joiner** node'
  - The node that received the given transaction via Galera replication.
- The terms master and slave in this context are only **relevant for a given transaction**

Replication Roles

- Within the cluster, all nodes are equal
- '**master/donor** node'
  - The node a given transaction was written and committed on.
- '**slave/joiner** node'
  - The node that received the given transaction via Galera replication.
- The terms master and slave in this context are only **relevant for a given transaction**
- Writeset: Galera's term for a transaction. One or more RBR row changes

State Transfer Roles

- New nodes joining an existing cluster get provisioned automatically

State Transfer Roles

- New nodes joining an existing cluster get provisioned automatically
  - **Joiner** = Mew node

State Transfer Roles

- New nodes joining an existing cluster get provisioned automatically
  - **Joiner** = Mew node
  - **Donor** = Node giving a copy of the datadir

State Transfer Roles

- New nodes joining an existing cluster get provisioned automatically
  - **Joiner** = Mew node
  - **Donor** = Node giving a copy of the datadir
- State Snapshot transfer

State Transfer Roles

- New nodes joining an existing cluster get provisioned automatically
  - **Joiner** = Mew node
  - **Donor** = Node giving a copy of the datadir
- State Snapshot transfer
  - Full backup of Donor to Joiner

State Transfer Roles

- New nodes joining an existing cluster get provisioned automatically
  - **Joiner** = Mew node
  - **Donor** = Node giving a copy of the datadir
- State Snapshot transfer
  - Full backup of Donor to Joiner
- Incremental Snapshot transfer

State Transfer Roles

- New nodes joining an existing cluster get provisioned automatically
  - **Joiner** = Mew node
  - **Donor** = Node giving a copy of the datadir
- State Snapshot transfer
  - Full backup of Donor to Joiner
- Incremental Snapshot transfer
  - Only changes since node left cluster

Writeset

- RBR payload (black box to Galera)

Writeset

- RBR payload (black box to Galera)
- Replication keys (generated by master/donor node)

Writeset

- RBR payload (black box to Galera)
- Replication keys (generated by master/donor node)
  - Primary keys

Writeset

- RBR payload (black box to Galera)
- Replication keys (generated by master/donor node)
  - Primary keys
  - Unique keys

Writeset

- RBR payload (black box to Galera)
- Replication keys (generated by master/donor node)
  - Primary keys
  - Unique keys
  - Foreign Keys

Writeset

- RBR payload (black box to Galera)
- Replication keys (generated by master/donor node)
  - Primary keys
  - Unique keys
  - Foreign Keys
  - Table names

Writeset

- RBR payload (black box to Galera)
- Replication keys (generated by master/donor node)
  - Primary keys
  - Unique keys
  - Foreign Keys
  - Table names
  - Schema names

Writeset

- RBR payload (black box to Galera)
- Replication keys (generated by master/donor node)
  - Primary keys
  - Unique keys
  - Foreign Keys
  - Table names
  - Schema names
- Keys are what make certification possible

Replication ?

It consists in 4 operations:

- Apply
- Replication
- Certification
- Commit

The order differs with the node's role

Replication Order on Master/Donor

1. Apply
2. Replication
3. Certification
4. Commit

Replication Order on Slave/Joiner

1. Replication (from master/donor)
2. Certification
3. Apply
4. Commit

Certification

- Can this writeset be applied?

Certification

- Can this writeset be applied?
  - Based on unapplied earlier transactions on master/donor

Certification

- Can this writeset be applied?
  - Based on unapplied earlier transactions on master/donor
  - Such conflicts must come from other nodes

Certification

- Can this writeset be applied?
  - Based on unapplied earlier transactions on master/donor
  - Such conflicts must come from other nodes
- Happens on every node

Certification

- Can this writeset be applied?
  - Based on unapplied earlier transactions on master/donor
  - Such conflicts must come from other nodes
- Happens on every node
- Should be deterministic on every node

Certification

- Can this writeset be applied?
  - Based on unapplied earlier transactions on master/donor
  - Such conflicts must come from other nodes
- Happens on every node
- Should be deterministic on every node
- Results are not reported to the cluster

Certification

- Can this writeset be applied?
  - Based on unapplied earlier transactions on master/donor
  - Such conflicts must come from other nodes
- Happens on every node
- Should be deterministic on every node
- Results are not reported to the cluster
  - Pass: enter apply queue (commit success on master/donor)

Certification

- Can this writeset be applied?
  - Based on unapplied earlier transactions on master/donor
  - Such conflicts must come from other nodes
- Happens on every node
- Should be deterministic on every node
- Results are not reported to the cluster
  - Pass: enter apply queue (commit success on master/donor)
  - Fail: drop transaction (or return deadlock on master/donor)

Certification

- Can this writeset be applied?
  - Based on unapplied earlier transactions on master/donor
  - Such conflicts must come from other nodes
- Happens on every node
- Should be deterministic on every node
- Results are not reported to the cluster
  - Pass: enter apply queue (commit success on master/donor)
  - Fail: drop transaction (or return deadlock on master/donor)
- Serialized by group communication sequence (and GTID will be synchronized following the same sequence)

Certification

- Can this writeset be applied?
  - Based on unapplied earlier transactions on master/donor
  - Such conflicts must come from other nodes
- Happens on every node
- Should be deterministic on every node
- Results are not reported to the cluster
  - Pass: enter apply queue (commit success on master/donor)
  - Fail: drop transaction (or return deadlock on master/donor)
- Serialized by group communication sequence (and GTID will be synchronized following the same sequence)
- Cost based on # of keys or # of rows

Apply

- Apply is done on slave nodes after certification

Apply

- Apply is done on slave nodes after certification
- Can be parallelized

Apply

- Apply is done on slave nodes after certification
- Can be parallelized
  - if wsrep_slave_threads > 1

Apply

- Apply is done on slave nodes after certification
- Can be parallelized
  - if wsrep_slave_threads > 1
  - if there are no other writesets with conflicting keys also being applied

Apply

- Apply is done on slave nodes after certification
- Can be parallelized
  - if wsrep_slave_threads > 1
  - if there are no other writesets with conflicting keys also being applied
- Cost: size of transaction

Apply

- Apply is done on slave nodes after certification
- Can be parallelized
  - if wsrep_slave_threads > 1
  - if there are no other writesets with conflicting keys also being applied
- Cost: size of transaction
- Generates brute force aborts on local node for conflicts

Commit

- Final local InnoDB commit

Commit

- Final local InnoDB commit
  - i.e., innodb_flush_log_at_trx_commit

Commit

- Final local InnoDB commit
  - i.e., innodb_flush_log_at_trx_commit
- GTID gets generated

Commit

- Final local InnoDB commit
  - i.e., innodb_flush_log_at_trx_commit
- GTID gets generated
- Done by applier threads on slaves/joiners

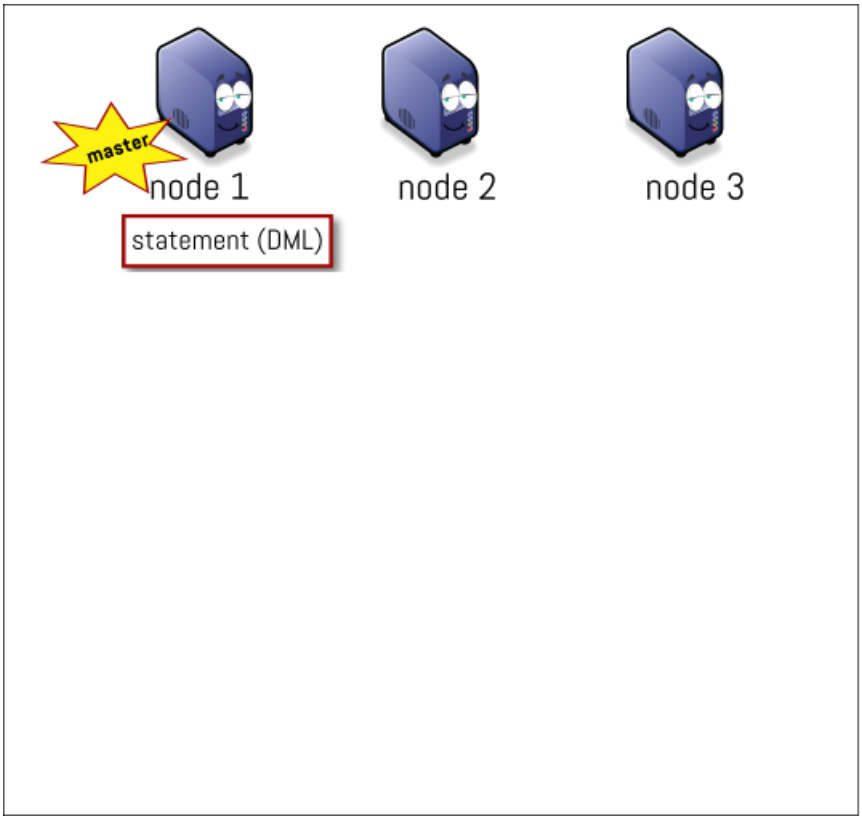
Commit

- Final local InnoDB commit
  - i.e., innodb_flush_log_at_trx_commit
- GTID gets generated
- Done by applier threads on slaves/joiners
- Done by client thread on master/donor

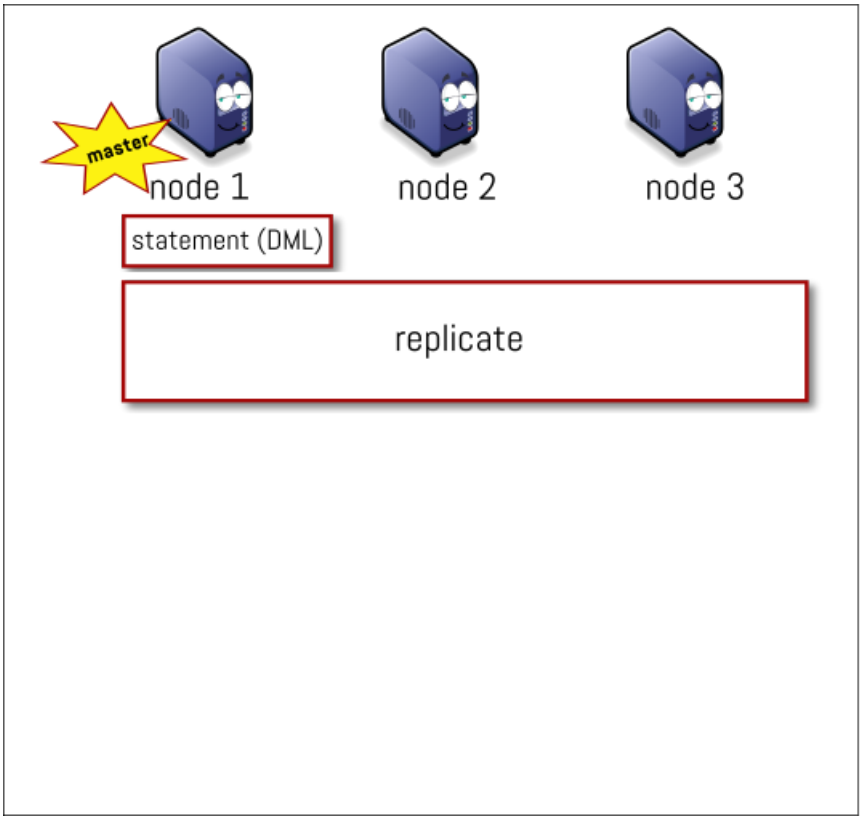
Commit

- Final local InnoDB commit
  - i.e., innodb_flush_log_at_trx_commit
- GTID gets generated
- Done by applier threads on slaves/joiners
- Done by client thread on master/donor
- innodb_flush_log_at_trx_commit=1 not required generally for PXC!

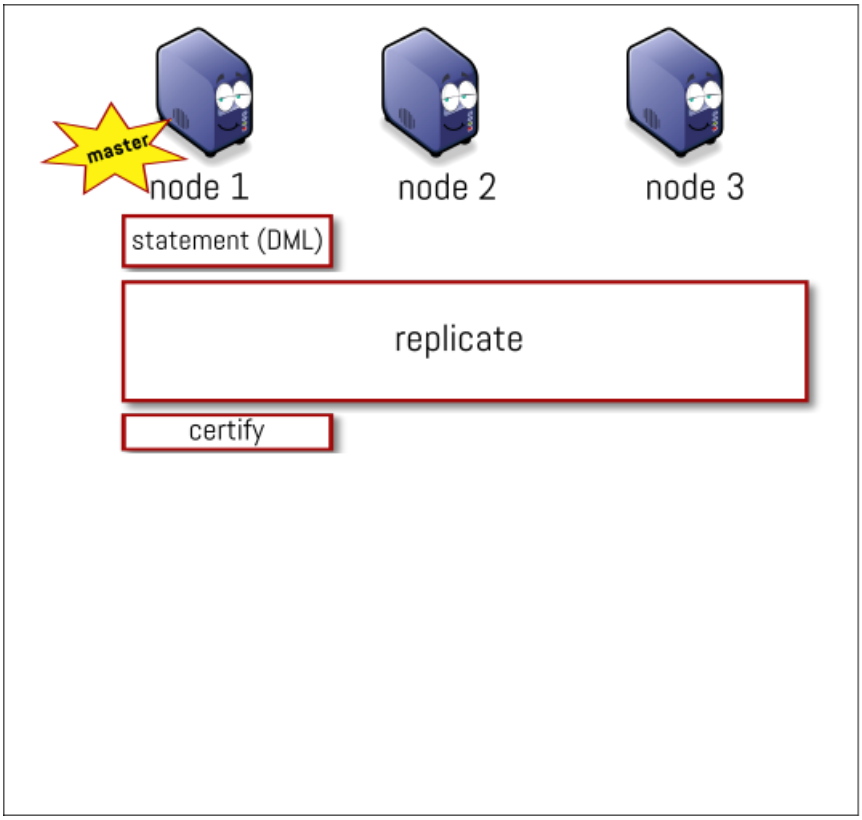
Galera Replication (autocommit)



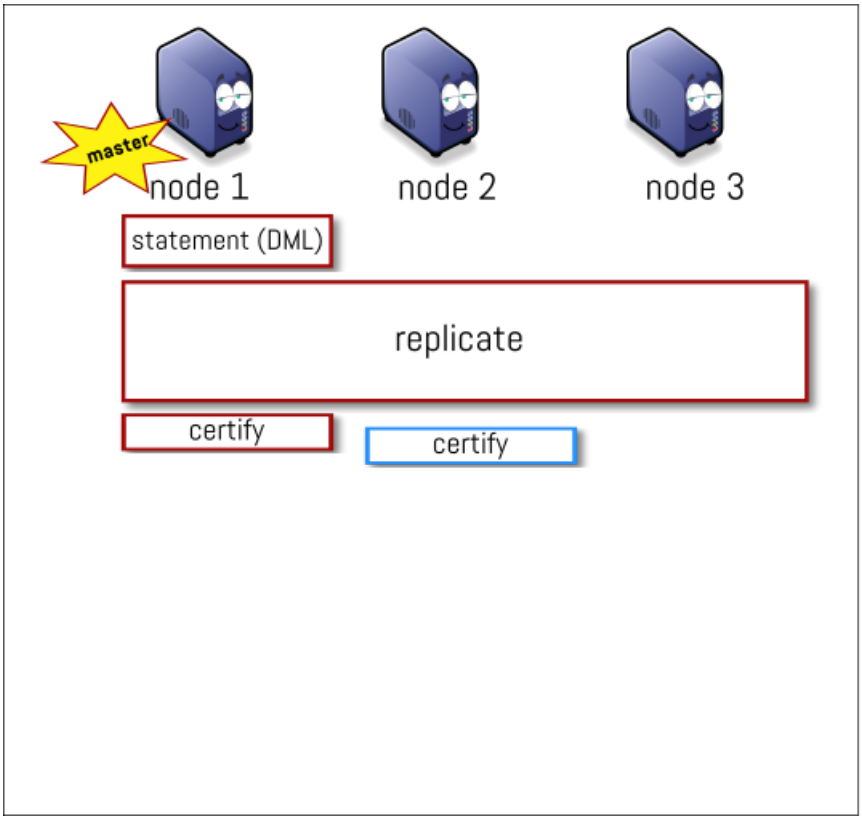
Galera Replication (autocommit)



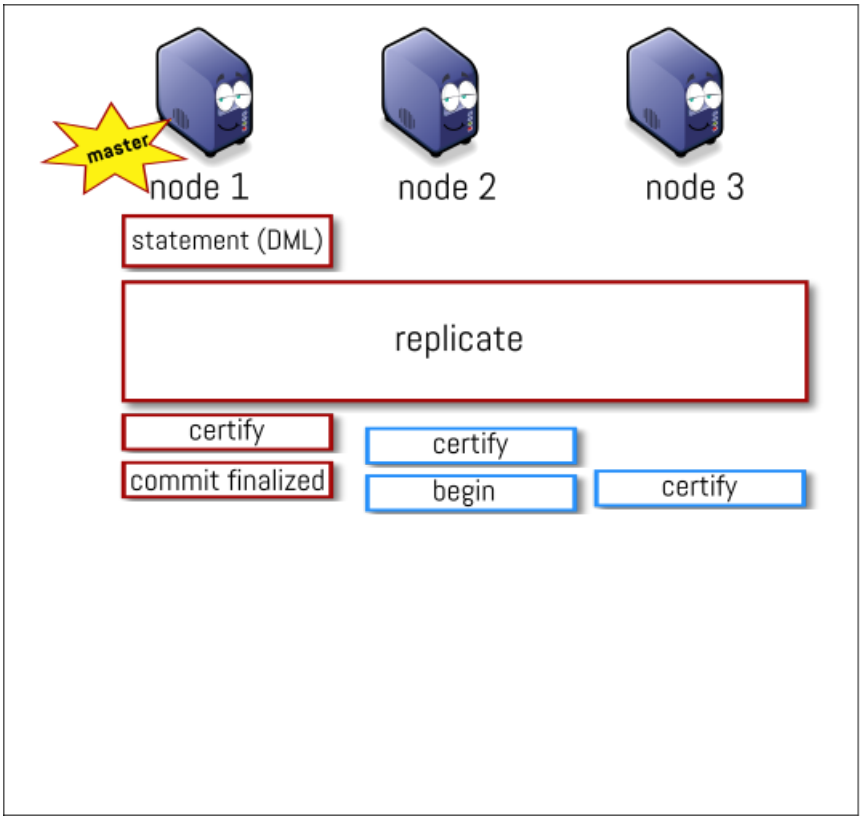
Galera Replication (autocommit)



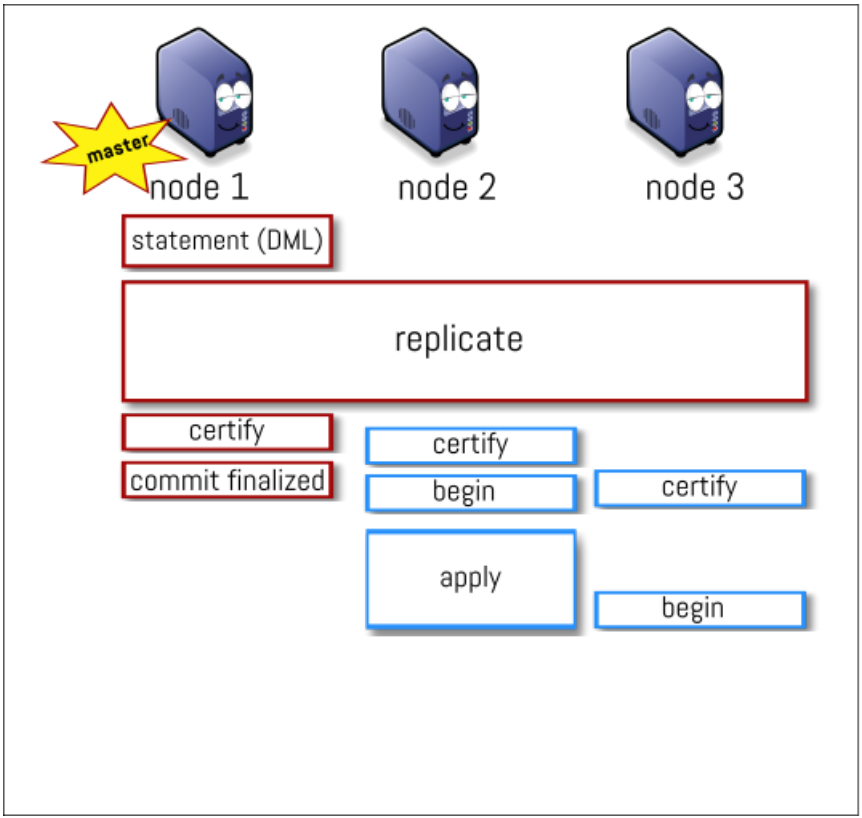
Galera Replication (autocommit)



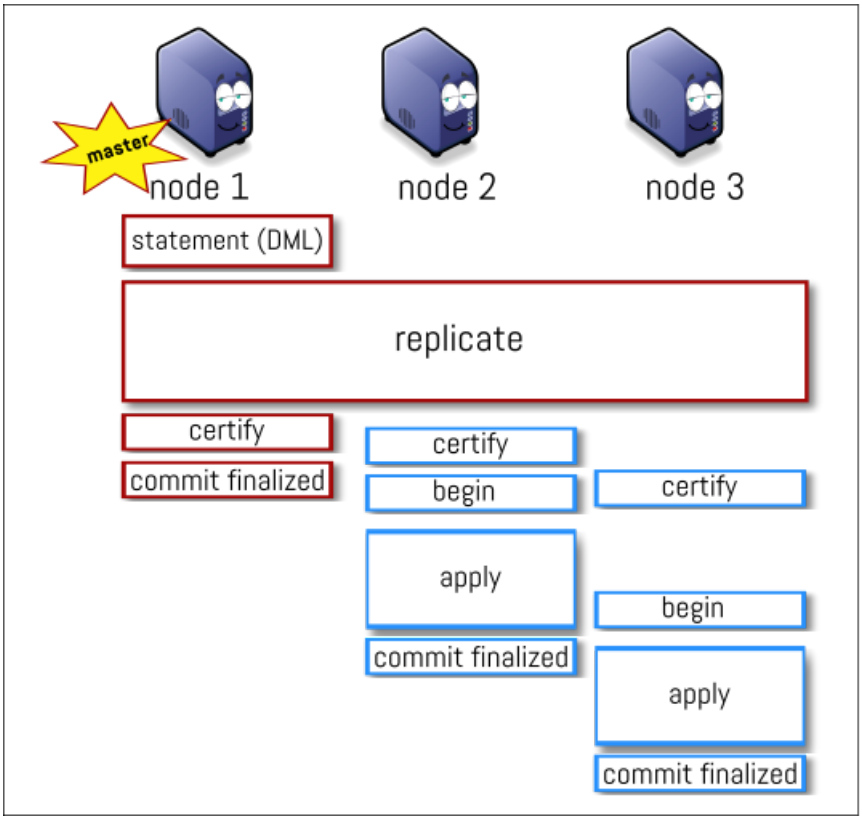
Galera Replication (autocommit)



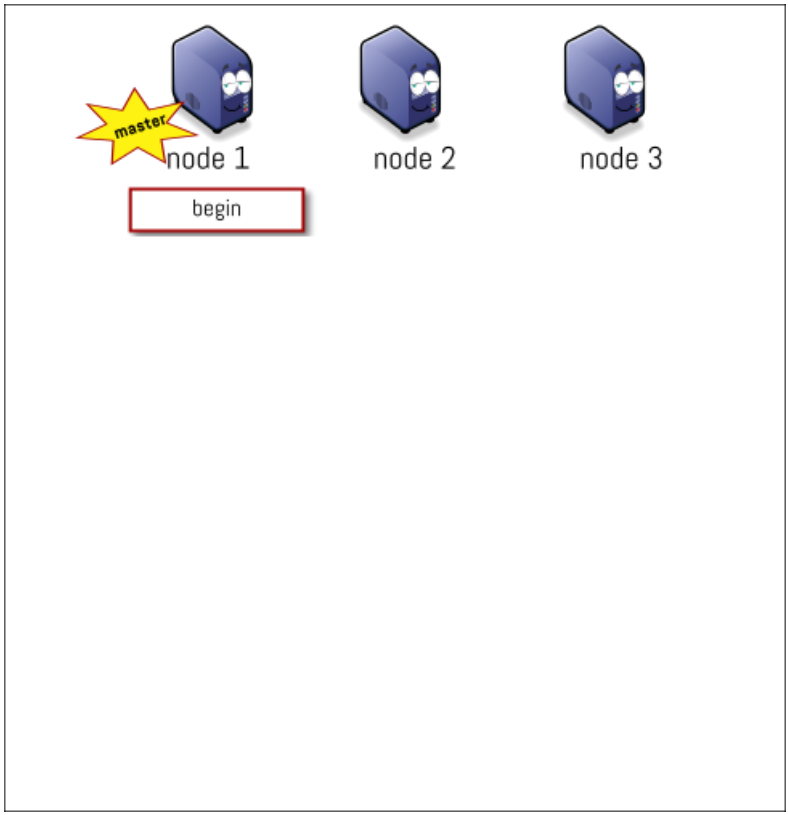
Galera Replication (autocommit)



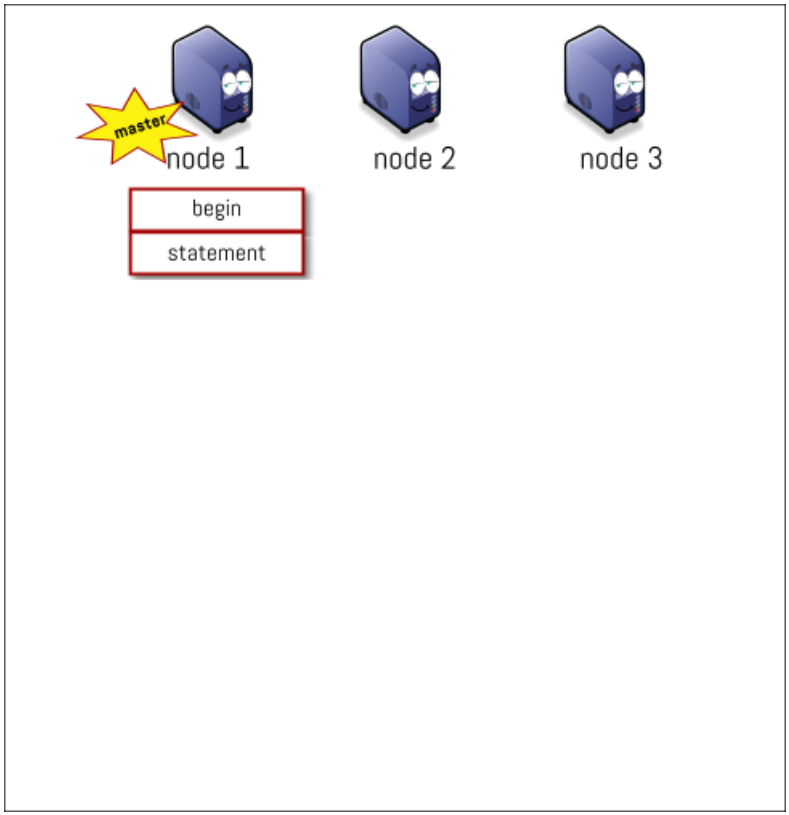
Galera Replication (autocommit)



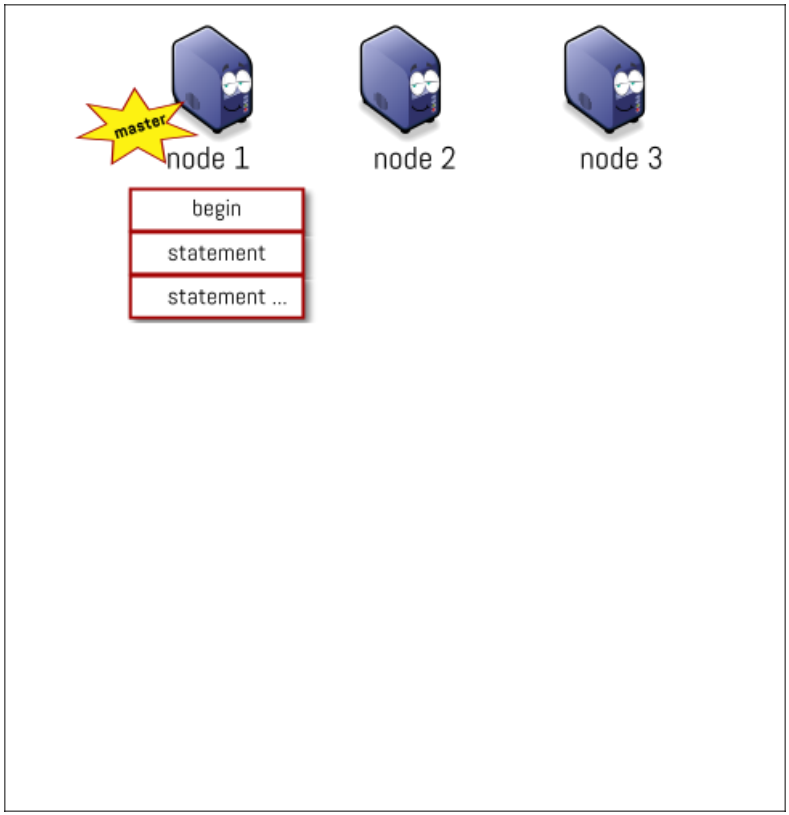
Galera Replication (full transaction)



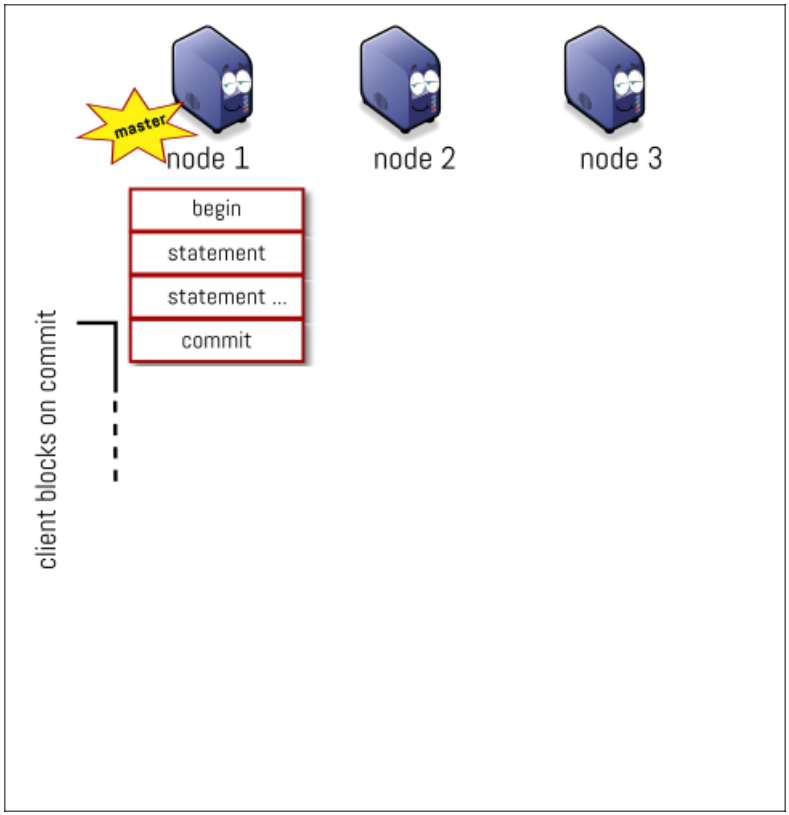
Galera Replication (full transaction)



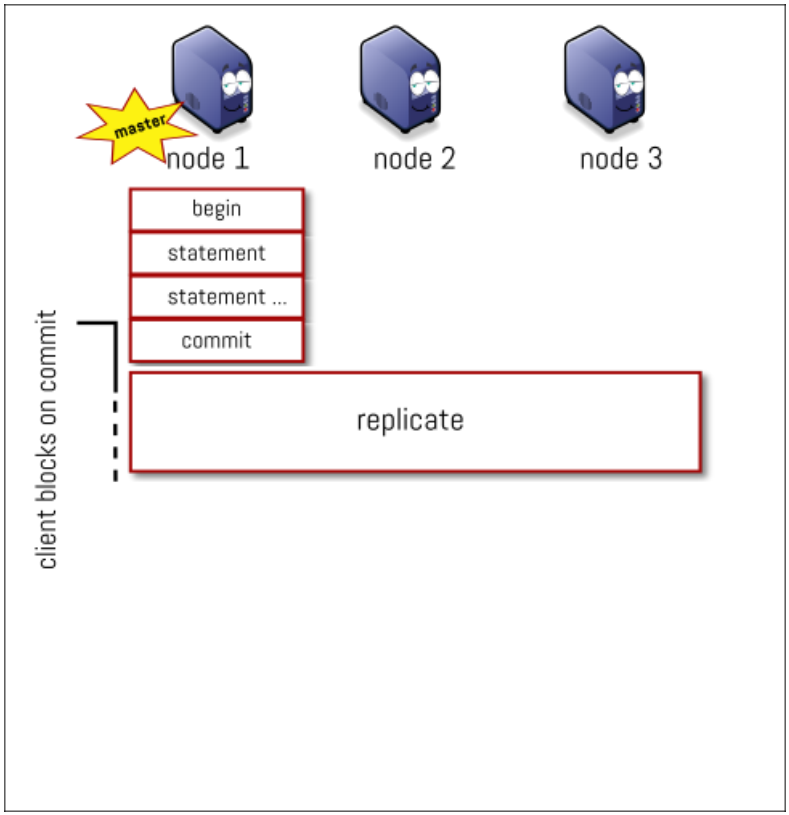
Galera Replication (full transaction)



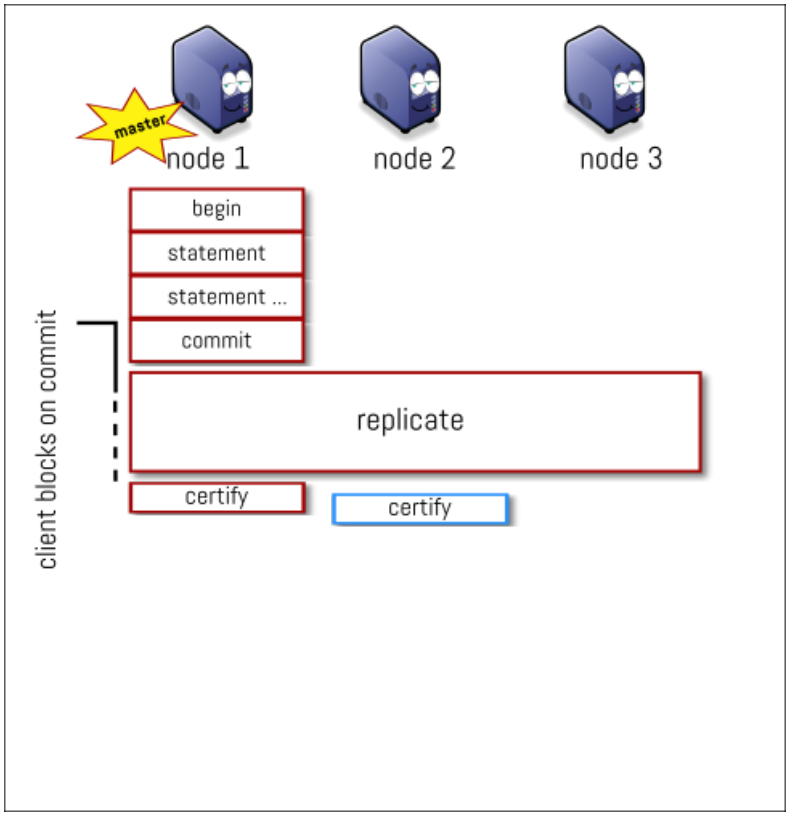
Galera Replication (full transaction)



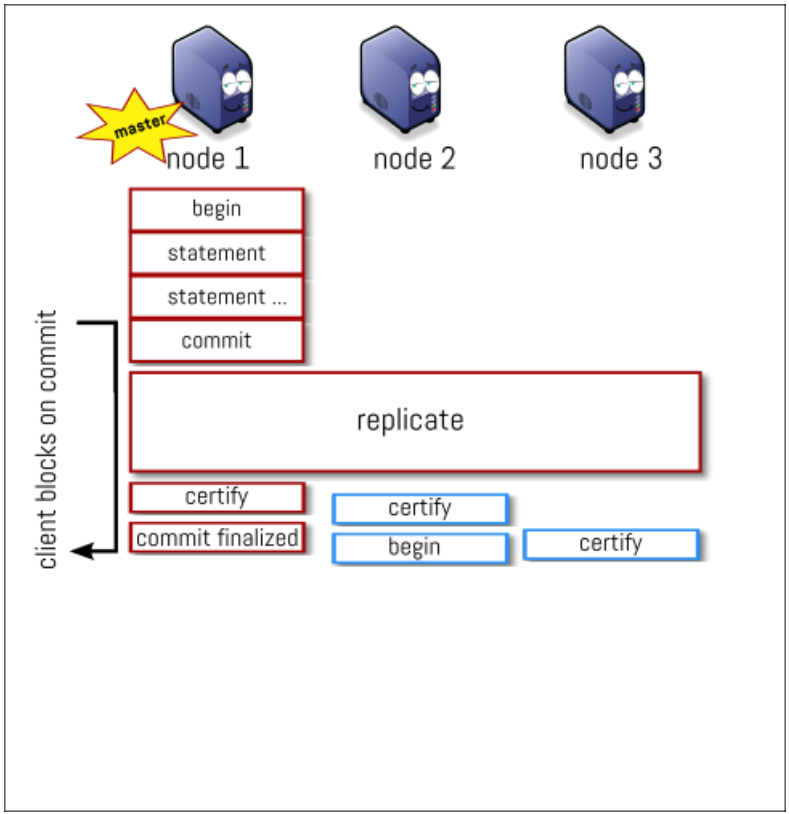
Galera Replication (full transaction)



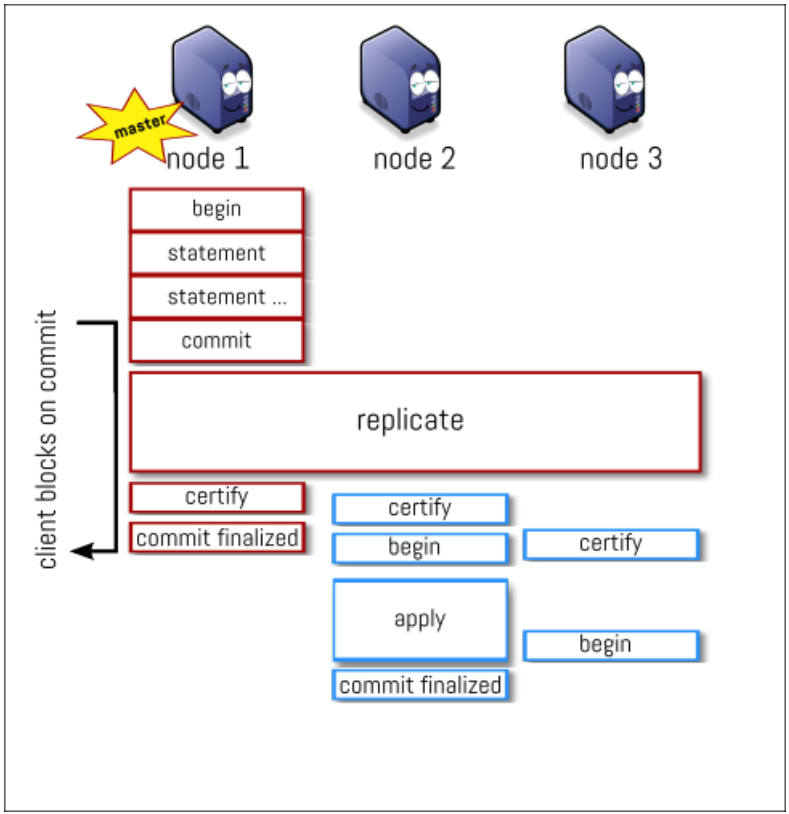
Galera Replication (full transaction)



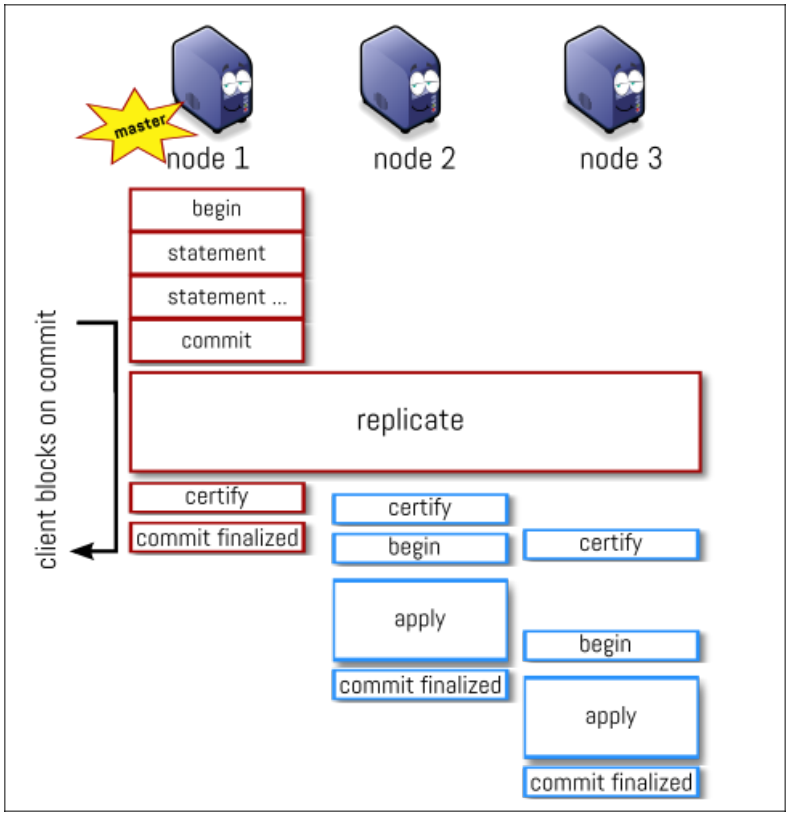
Galera Replication (full transaction)



Galera Replication (full transaction)

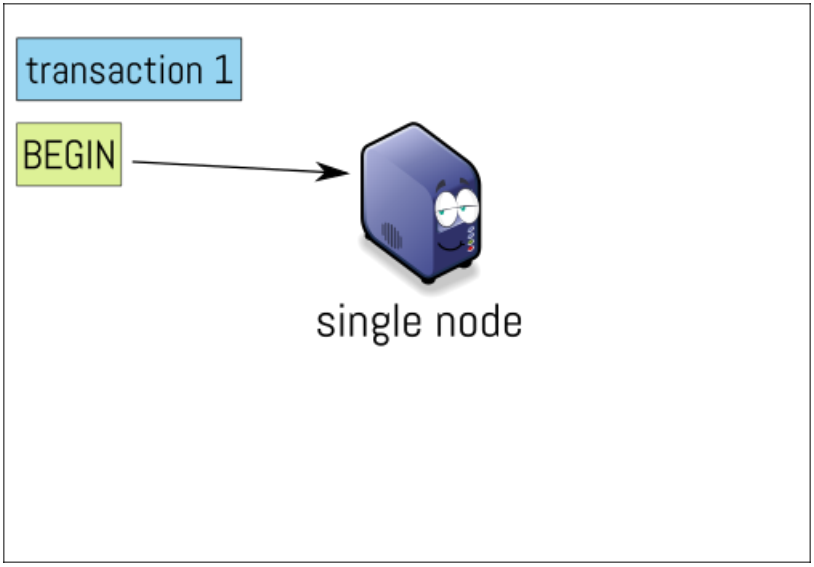


Galera Replication (full transaction)



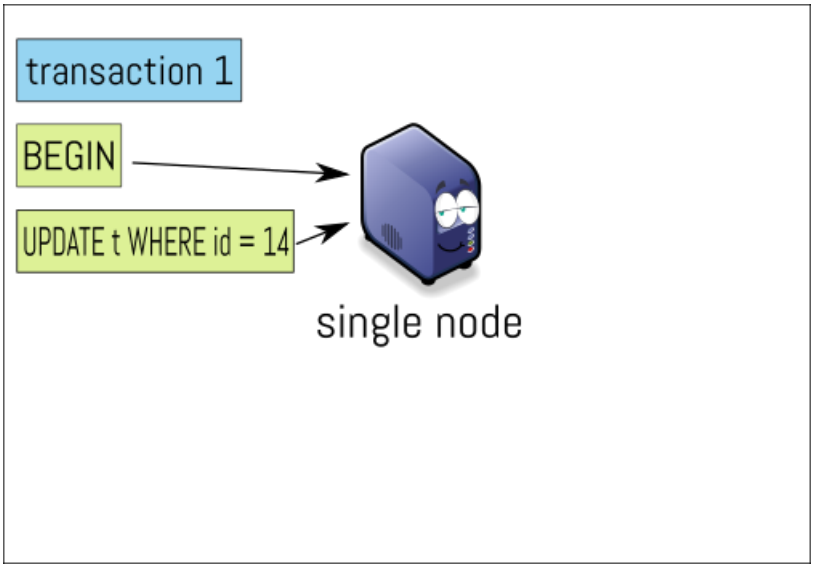
Optimistic Locking

Traditional locking



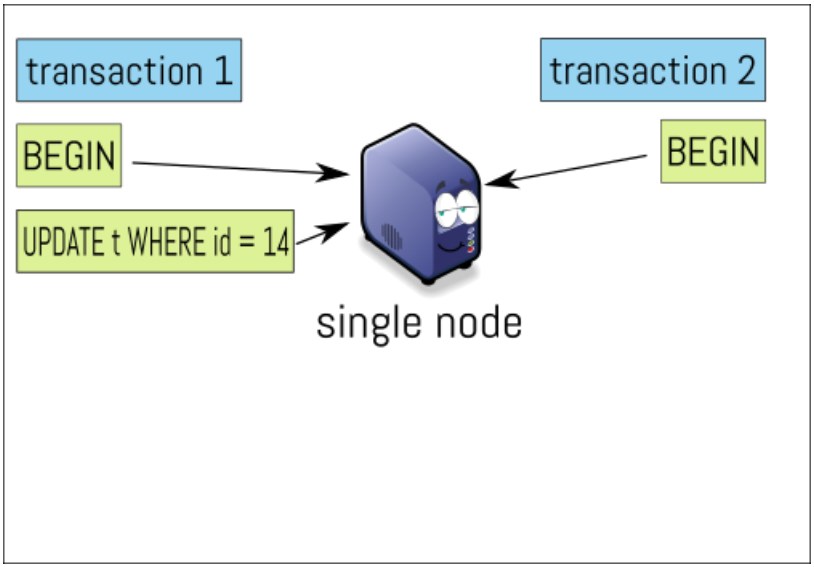
Optimistic Locking

Traditional locking



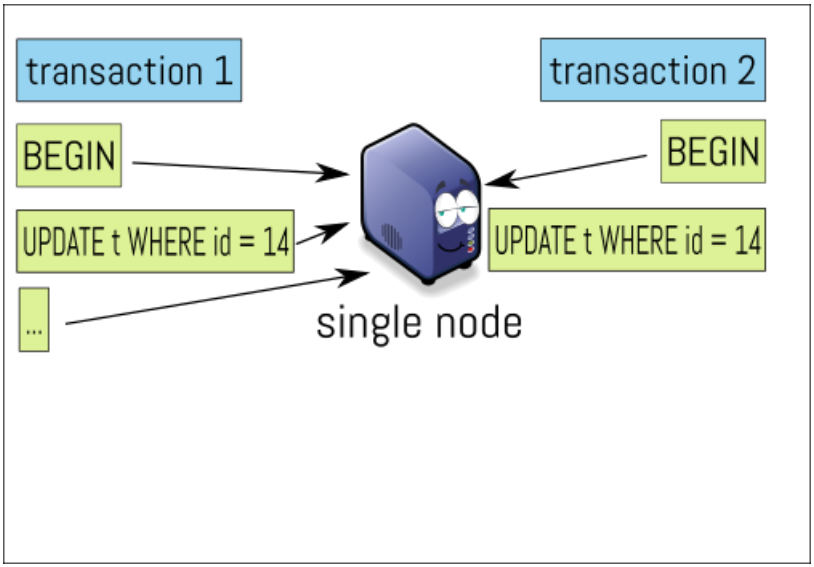
Optimistic Locking

Traditional locking



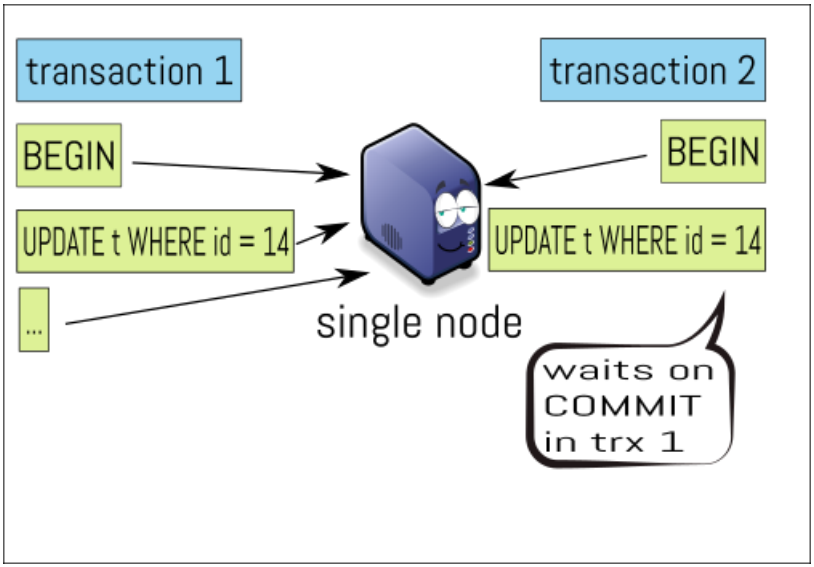
Optimistic Locking

Traditional locking



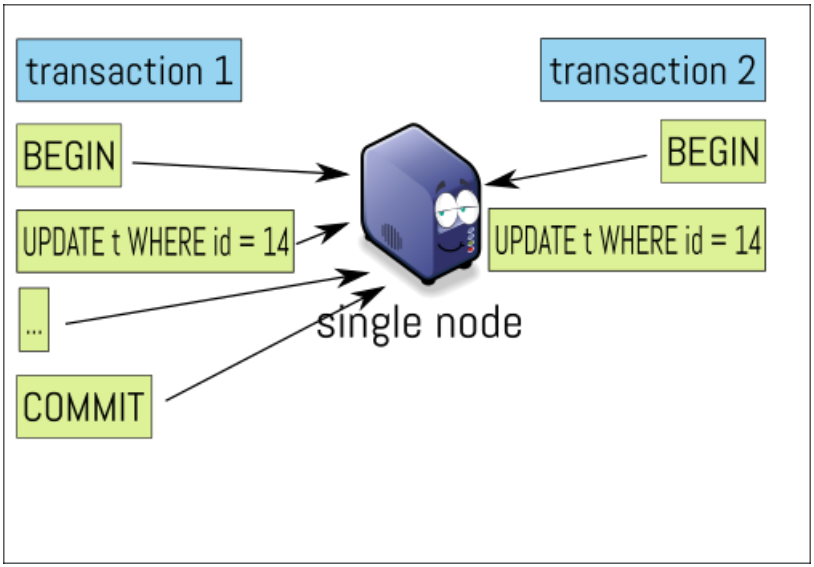
Optimistic Locking

Traditional locking



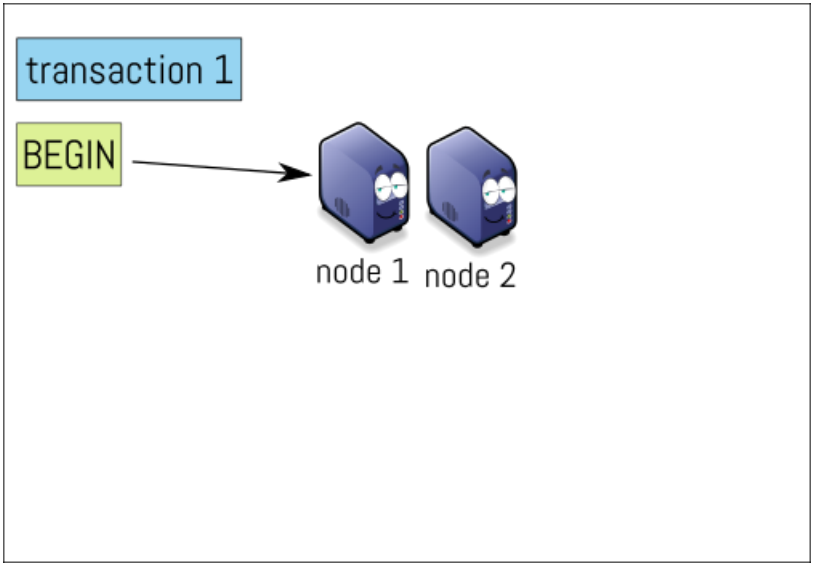
Optimistic Locking

Traditional locking



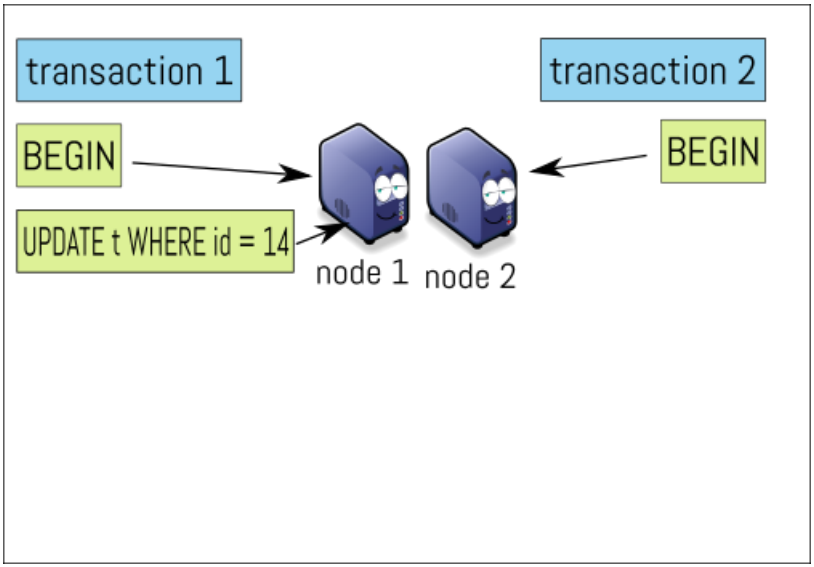
Optimistic Locking

Optimistic Locking



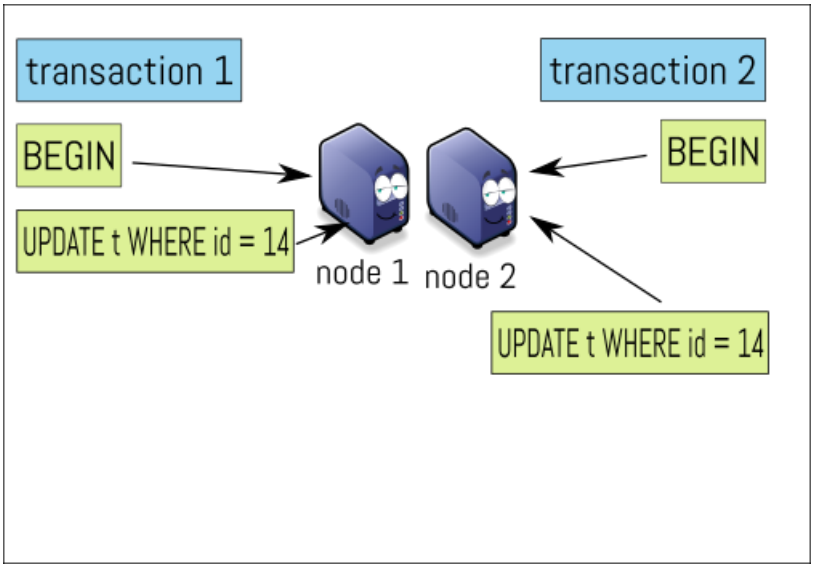
Optimistic Locking

Optimistic Locking



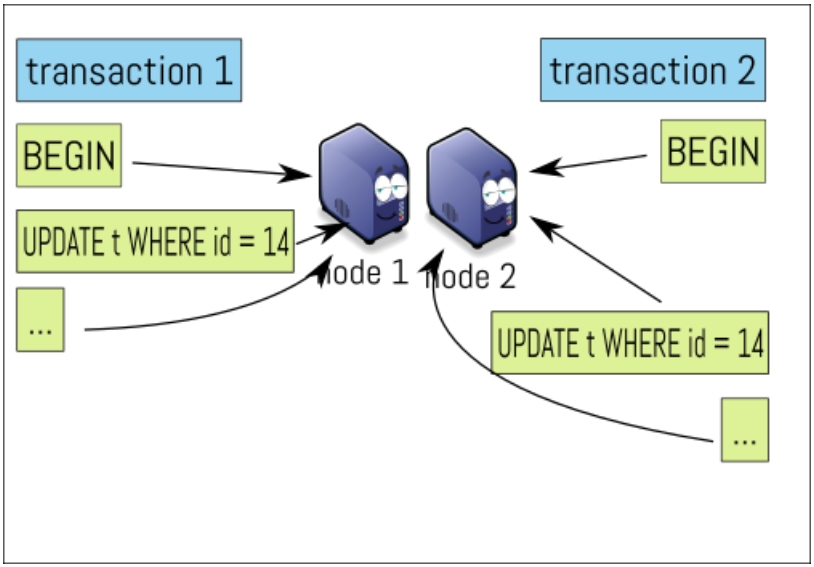
Optimistic Locking

Optimistic Locking



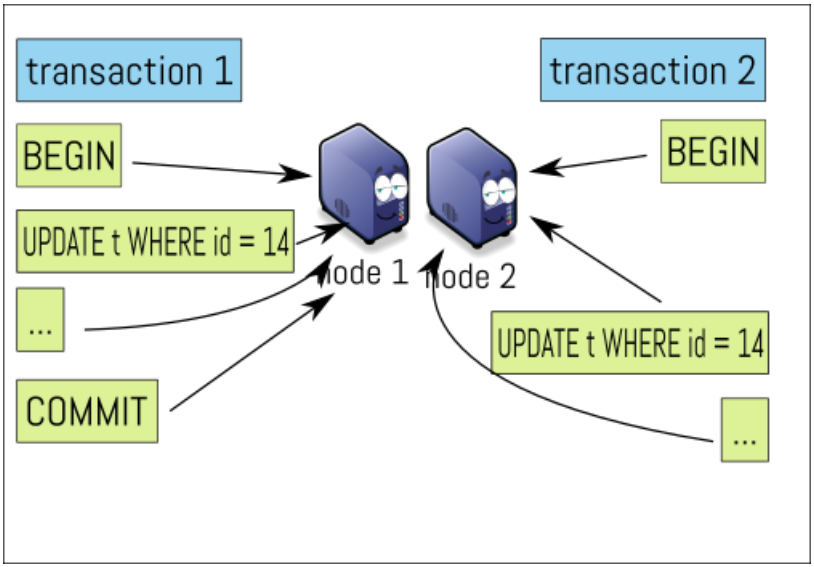
Optimistic Locking

Optimistic Locking



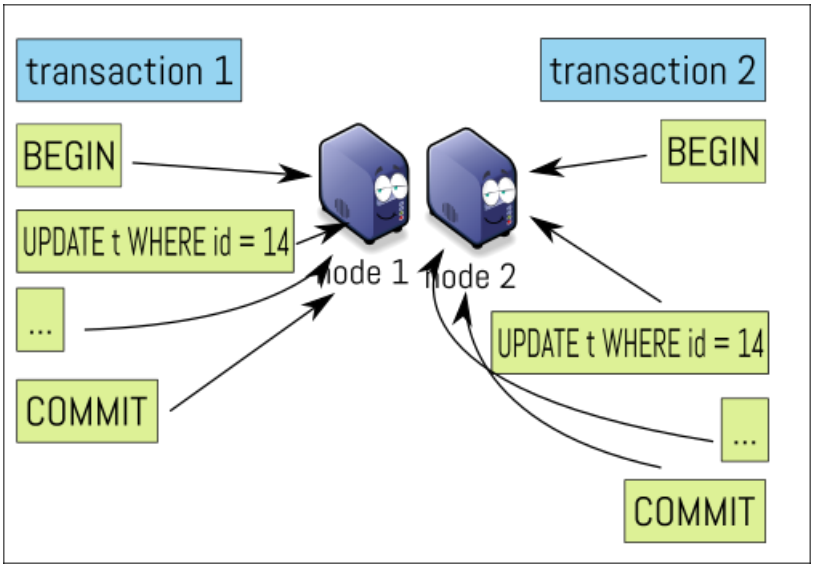
Optimistic Locking

Optimistic Locking



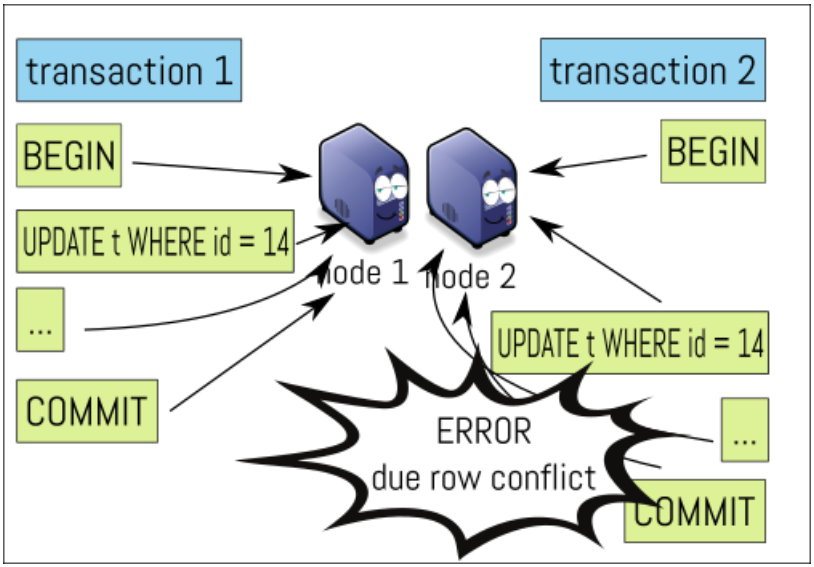
Optimistic Locking

Optimistic Locking

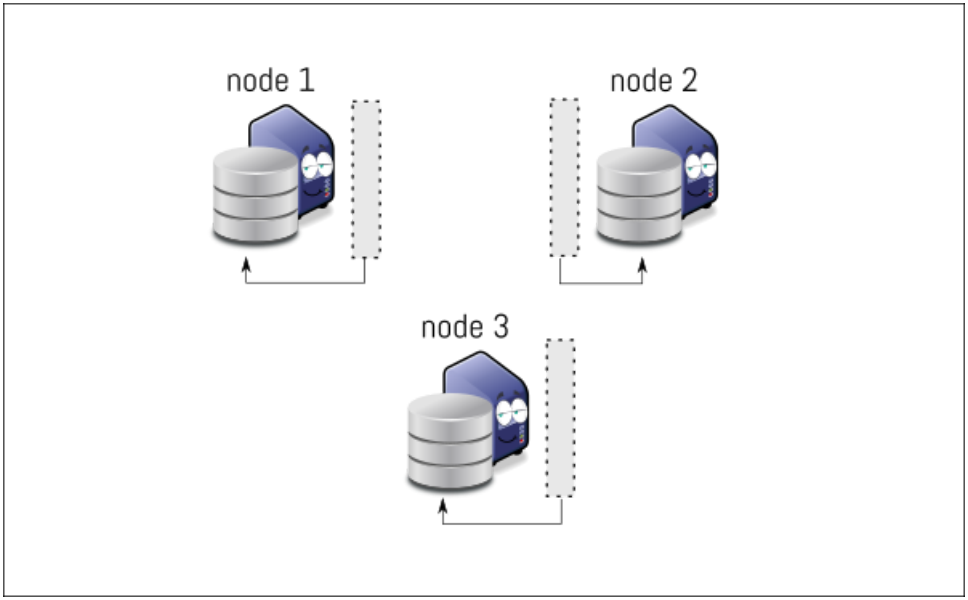


Optimistic Locking

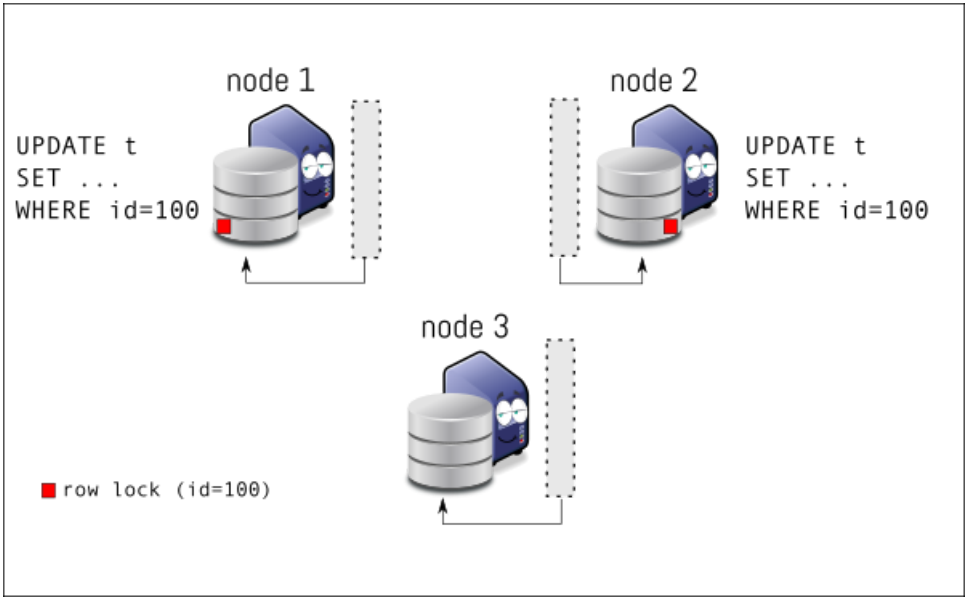
Optimistic Locking



Certification Failure

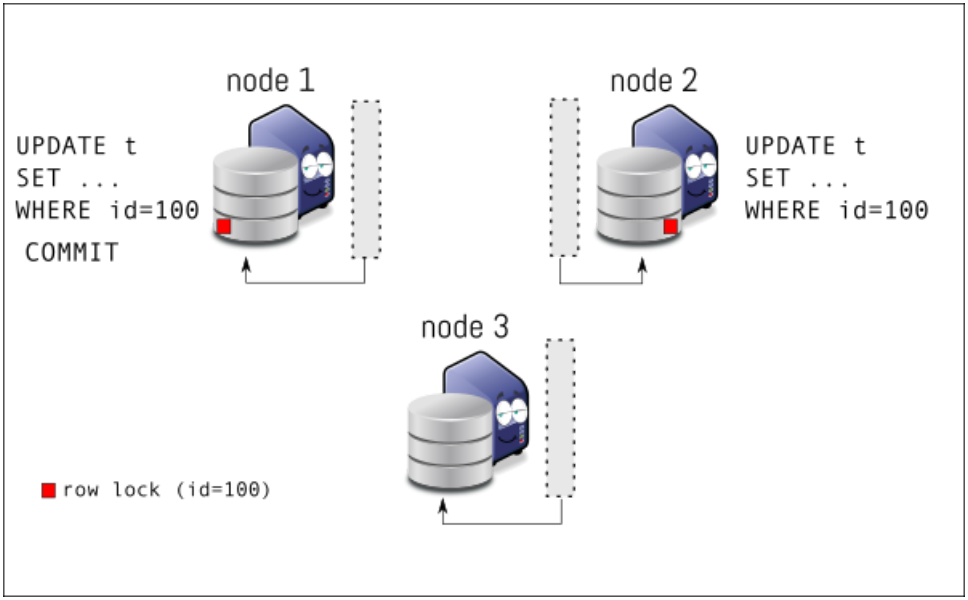


Certification Failure



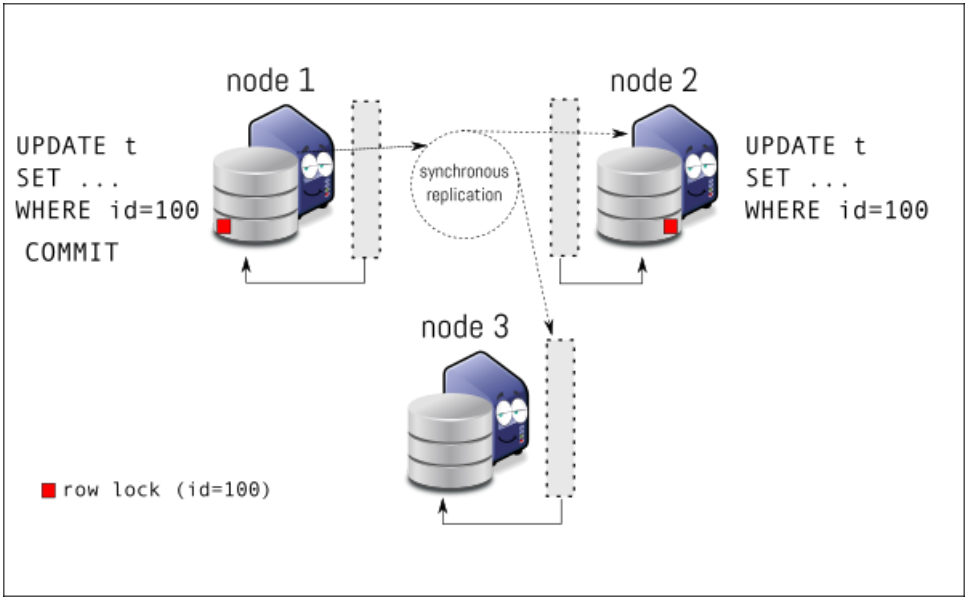
- Trx1 is open on **Node1**
- Trx2 is open on **Node2**

Certification Failure



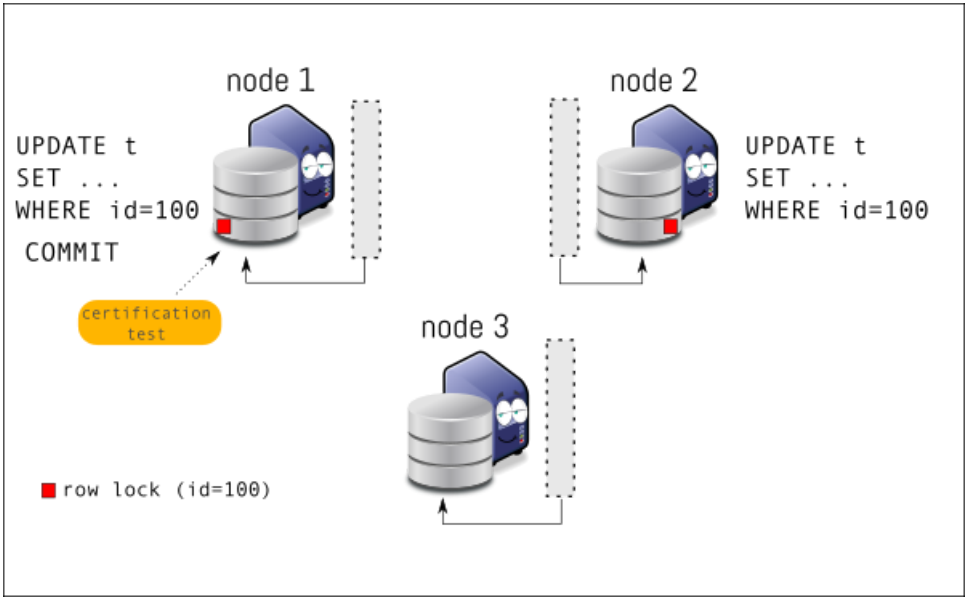
- Node1 gets COMMIT

Certification Failure



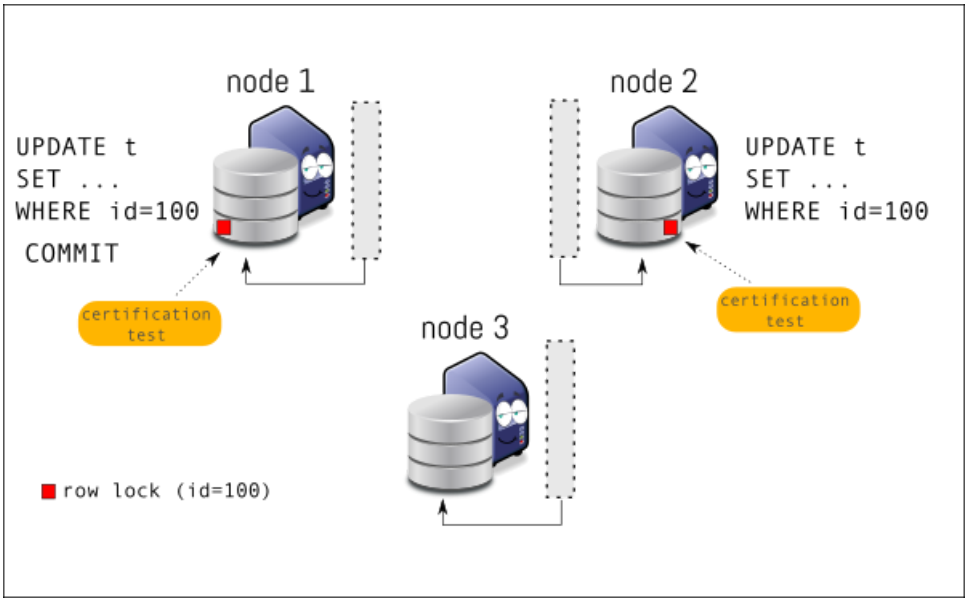
- Synchronous replication

Certification Failure



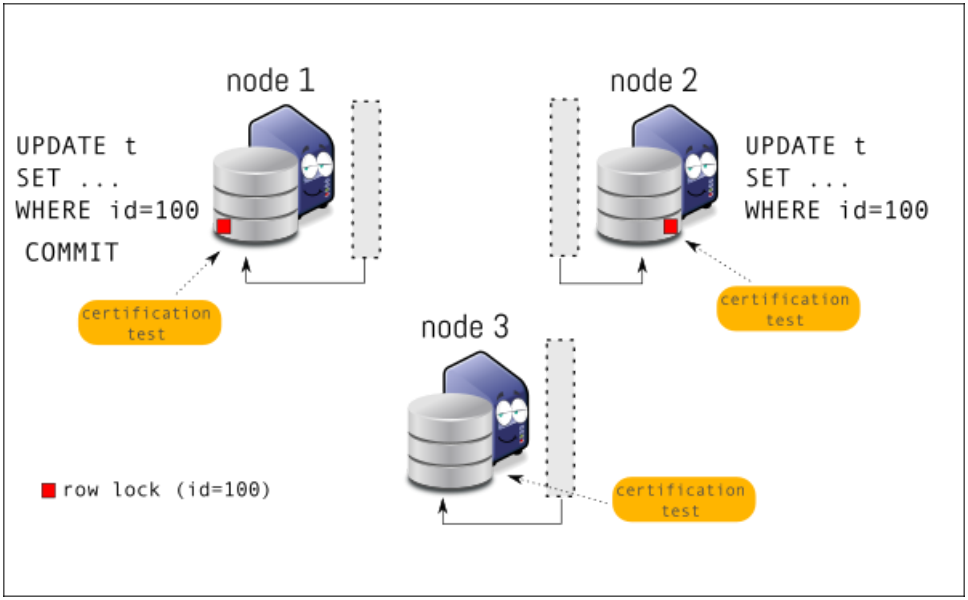
- Certification tests
  - run in isolation on each node

Certification Failure



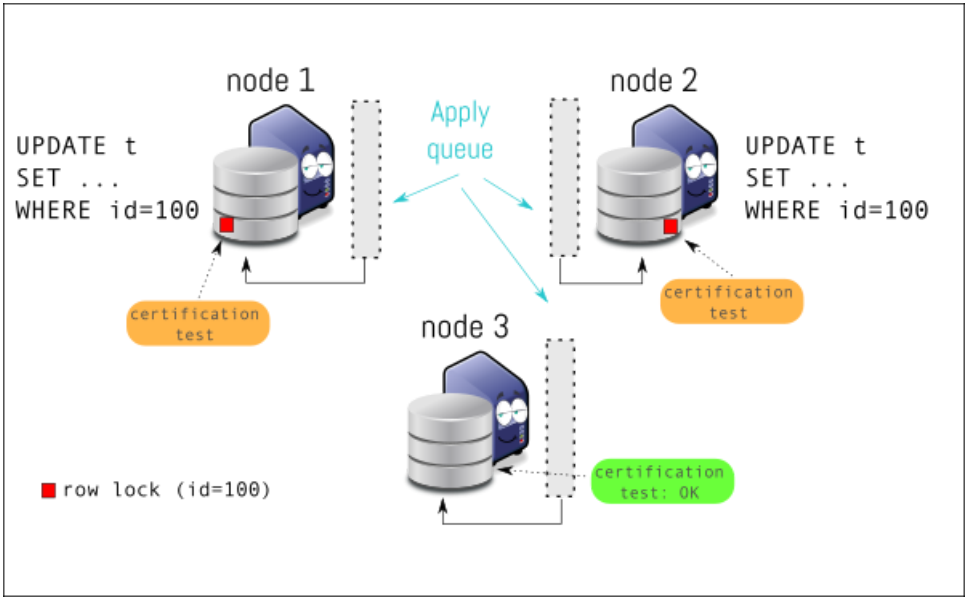
- Certification tests:
  - asynchronous

Certification Failure



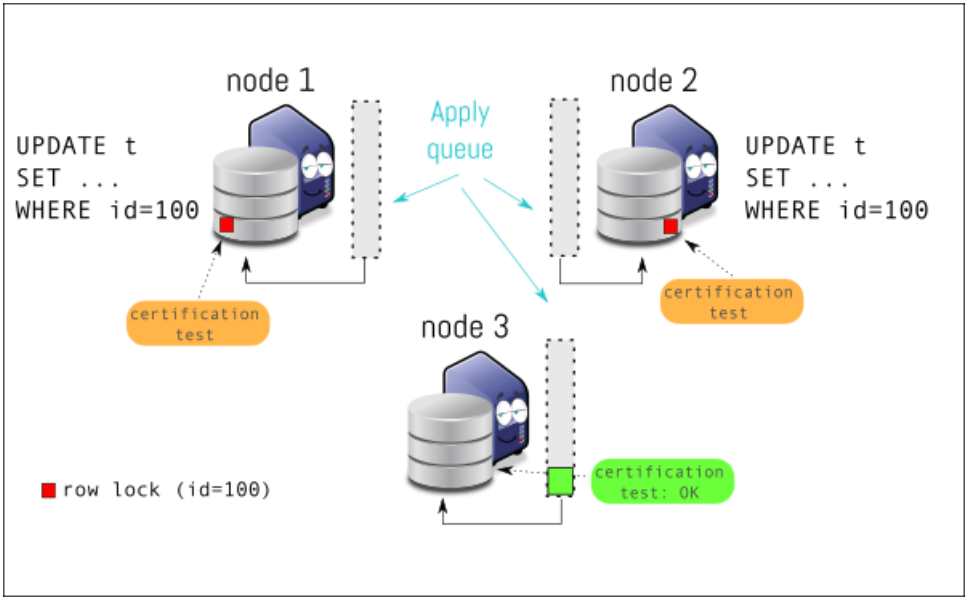
- Synchronous replication
  - deterministic

Certification Failure



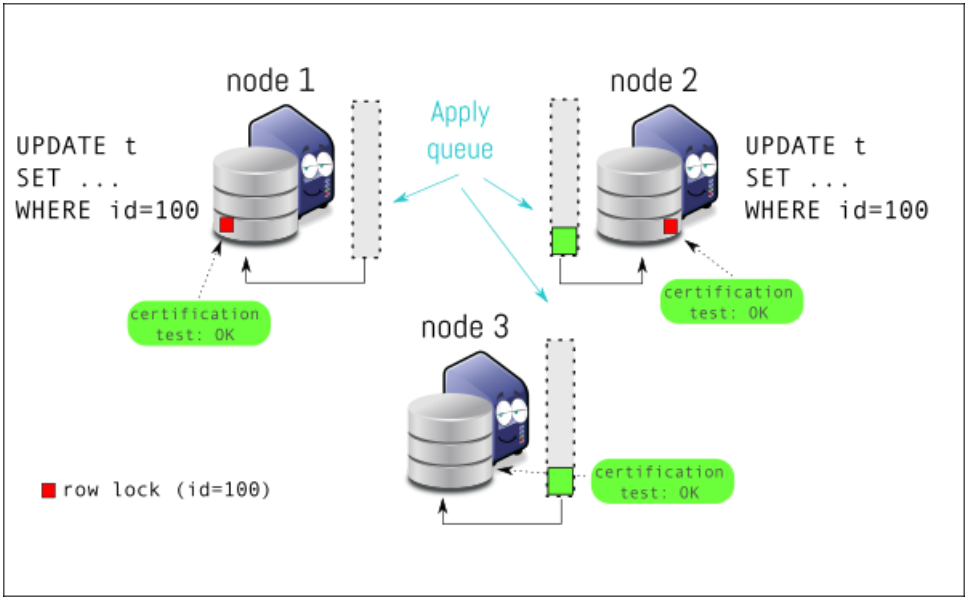
- Certification succeeds

Certification Failure

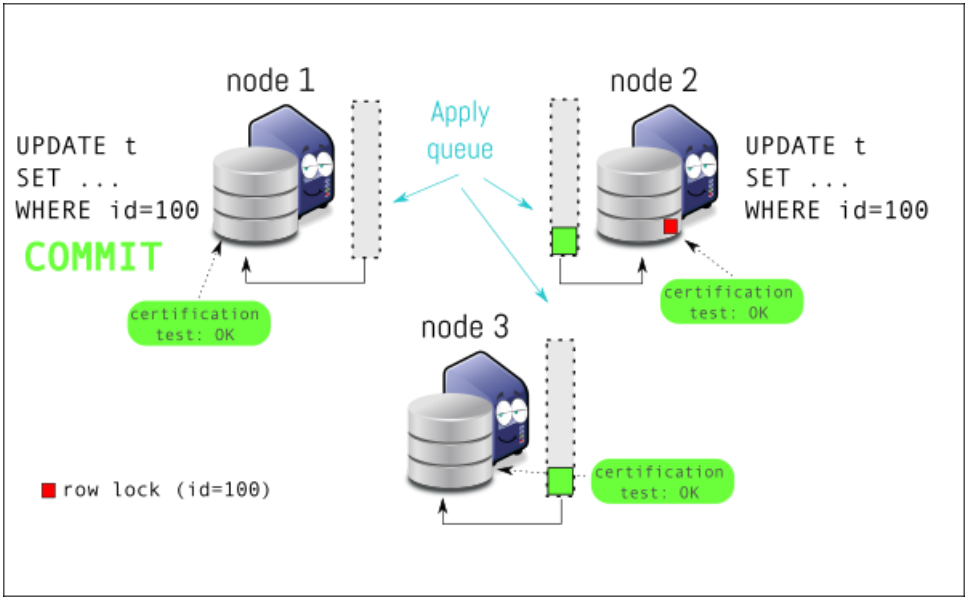


- Certified transaction goes to the apply queue

Certification Failure

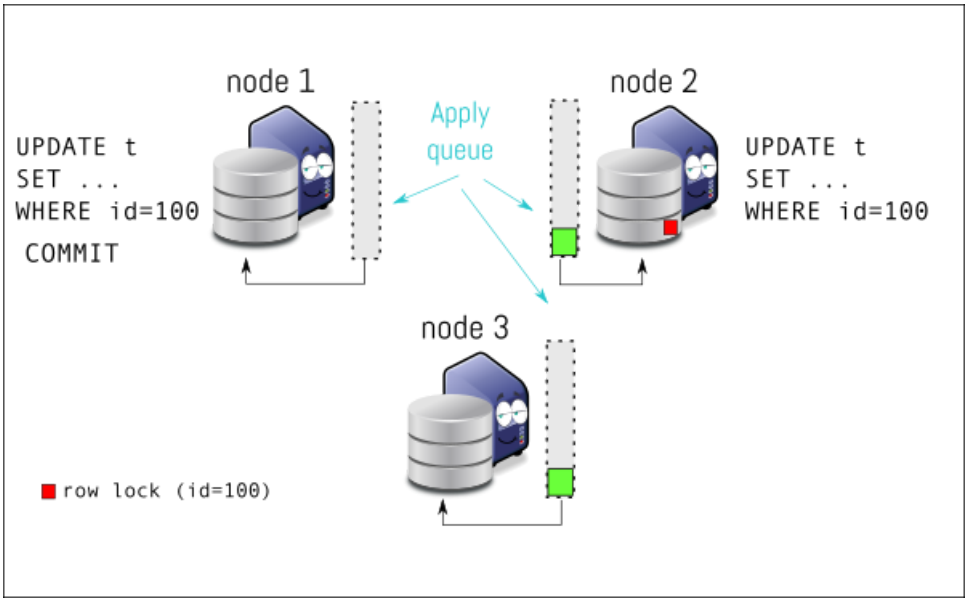


Certification Failure



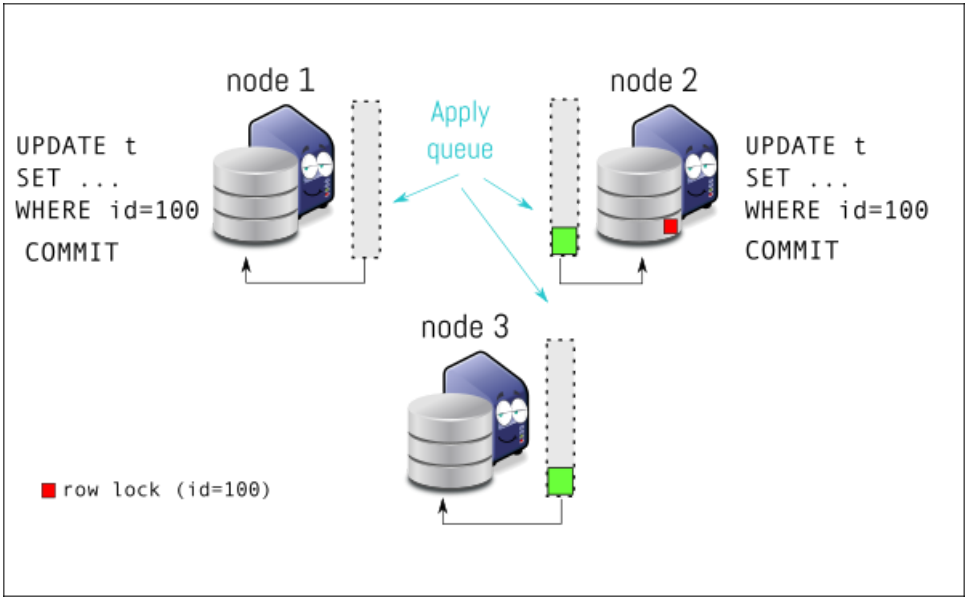
- On **Node1**, a successful cert test, means an actual **commit**

Certification Failure



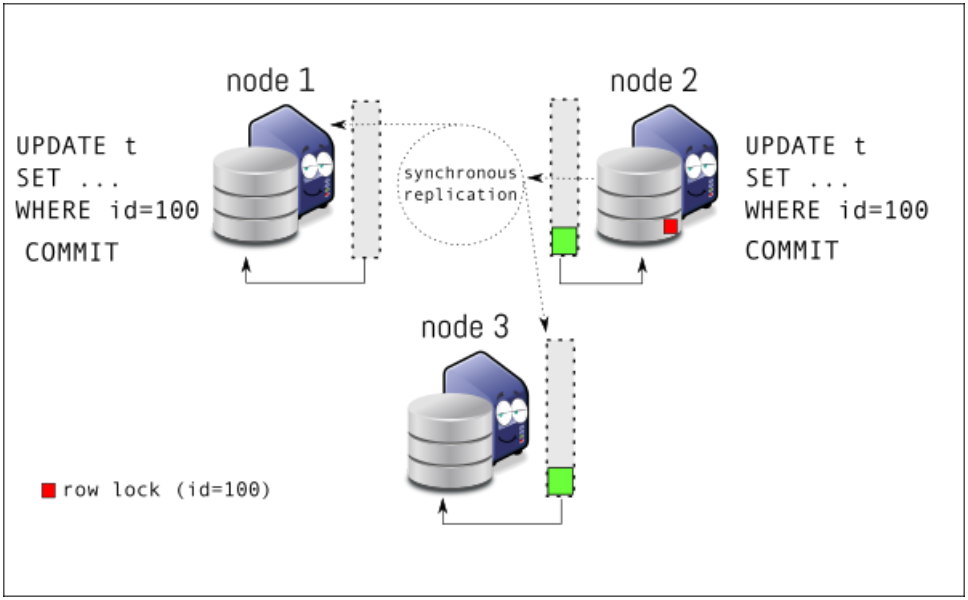
- and transactions in the apply queue (Node2 & Node3) are executed asynchronously

Certification Failure



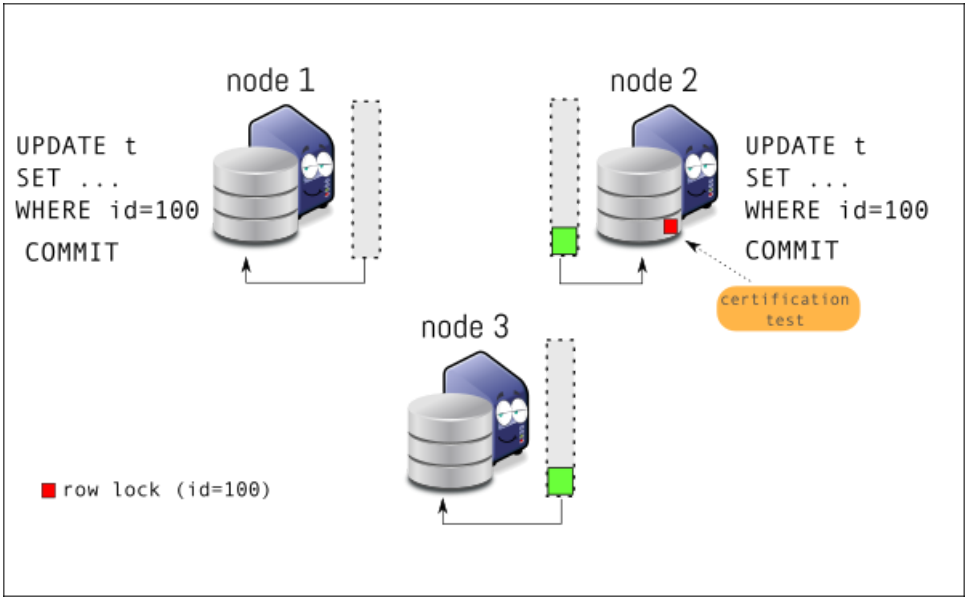
- On **node2** we commit the transaction

Certification Failure

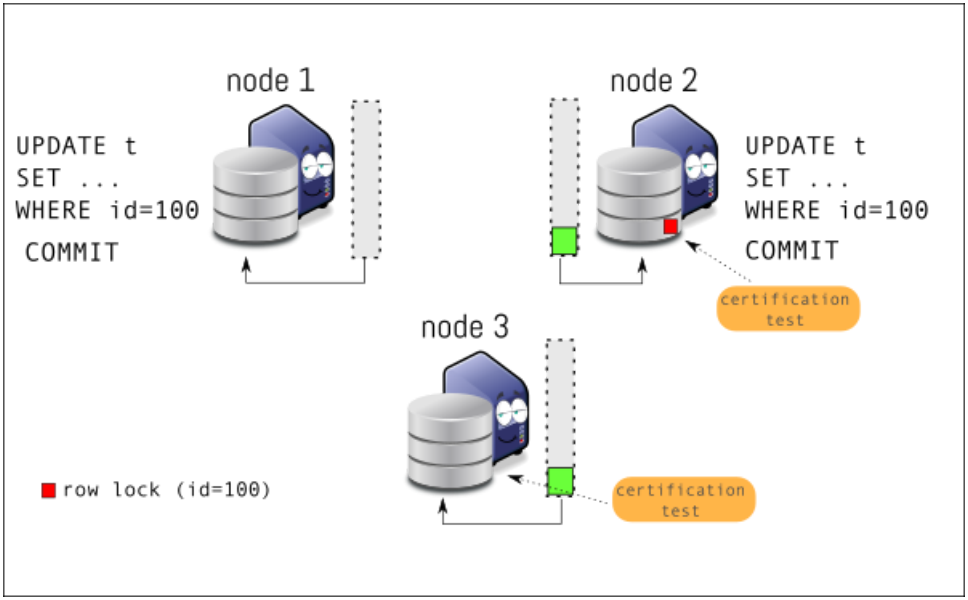


- Synchronous replication

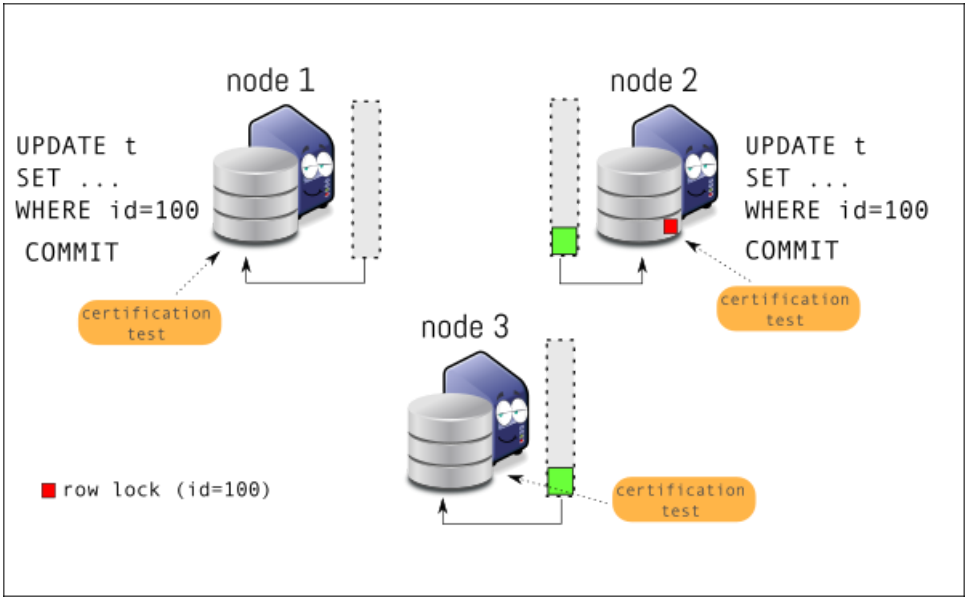
Certification Failure



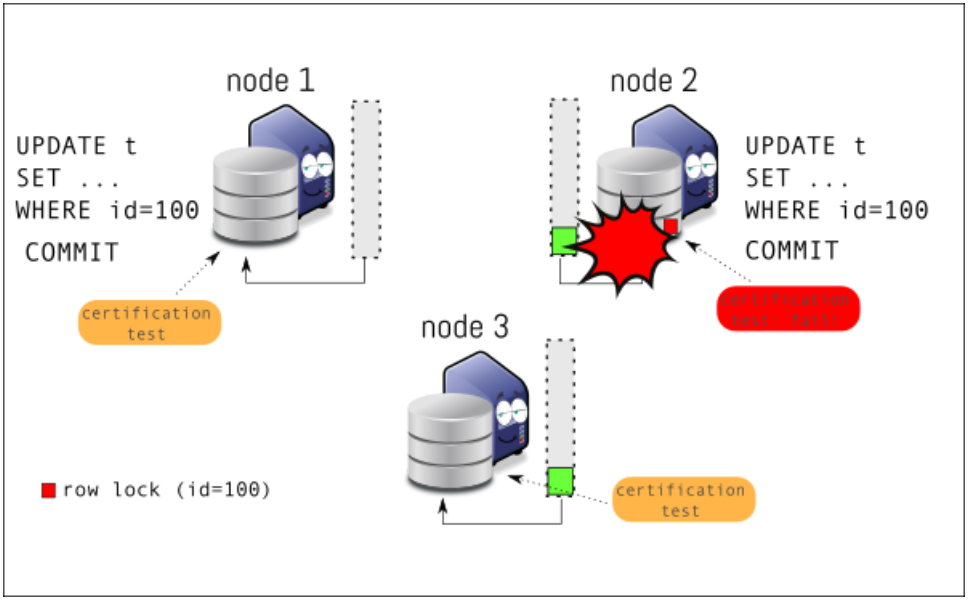
Certification Failure



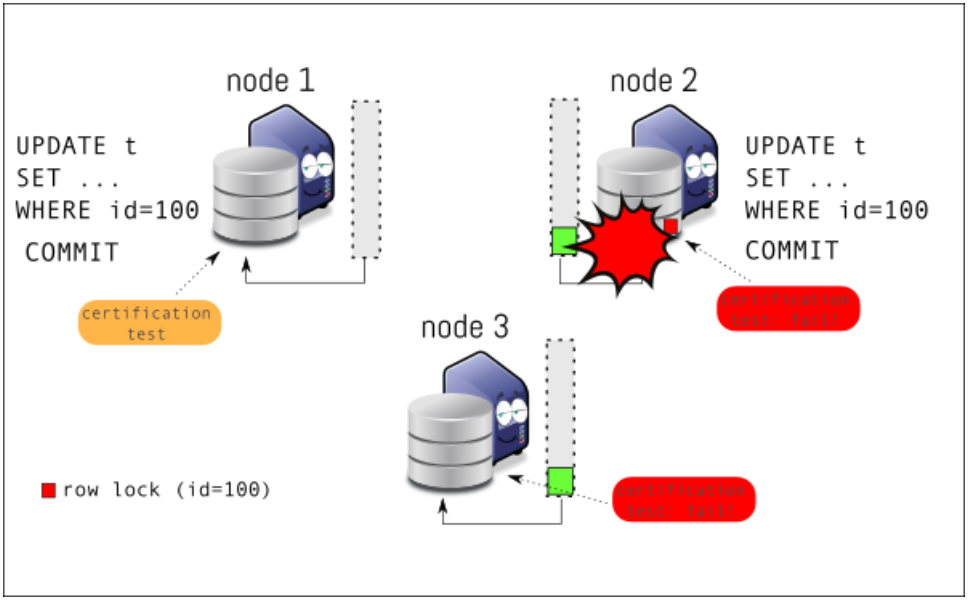
Certification Failure



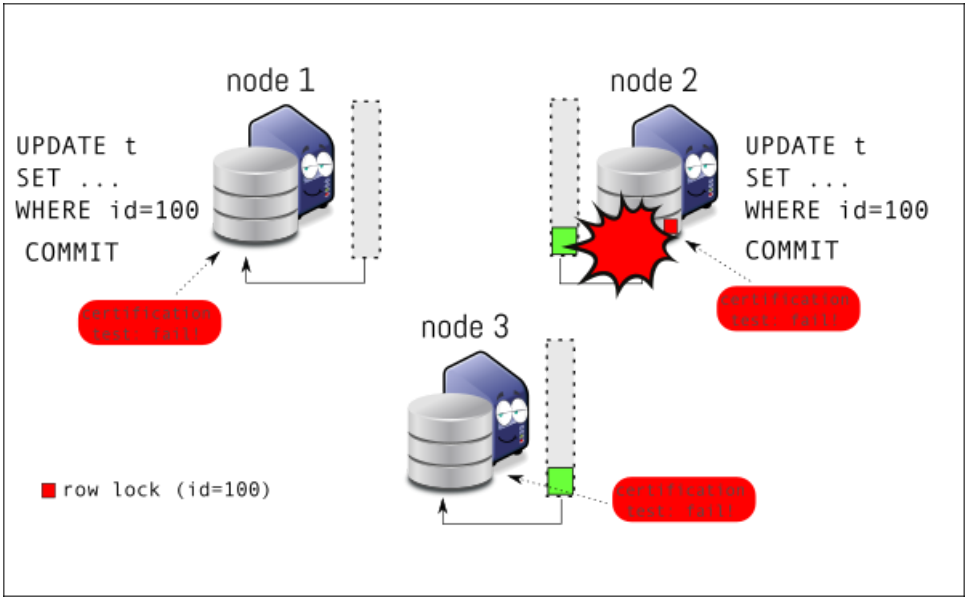
Certification Failure



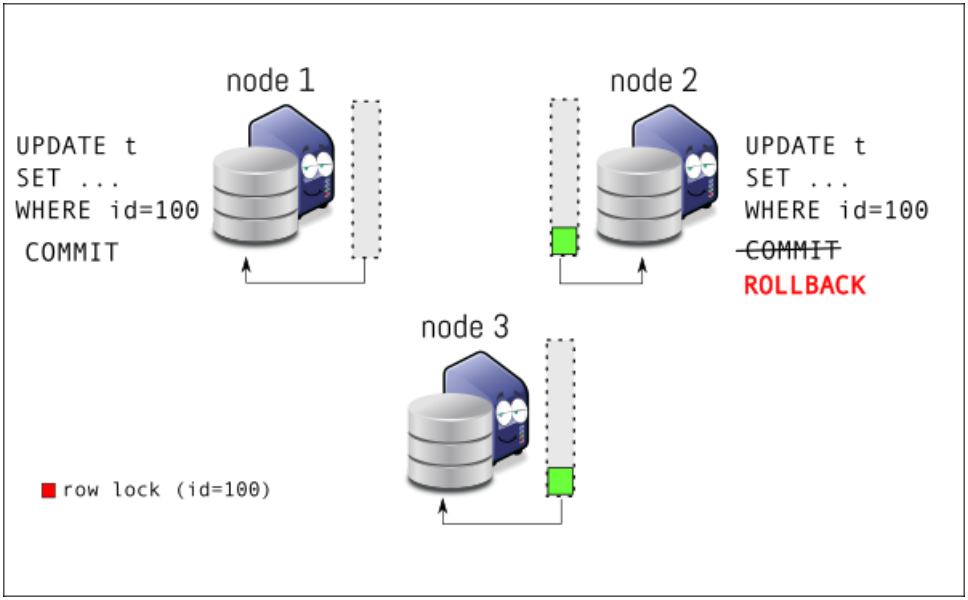
Certification Failure



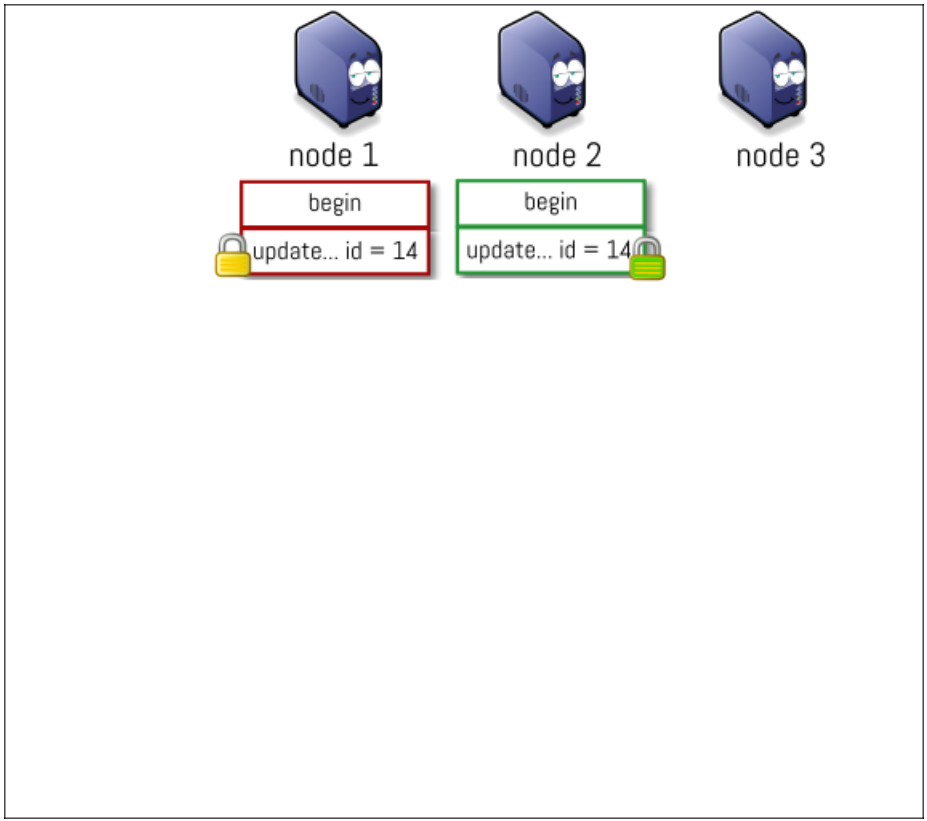
Certification Failure



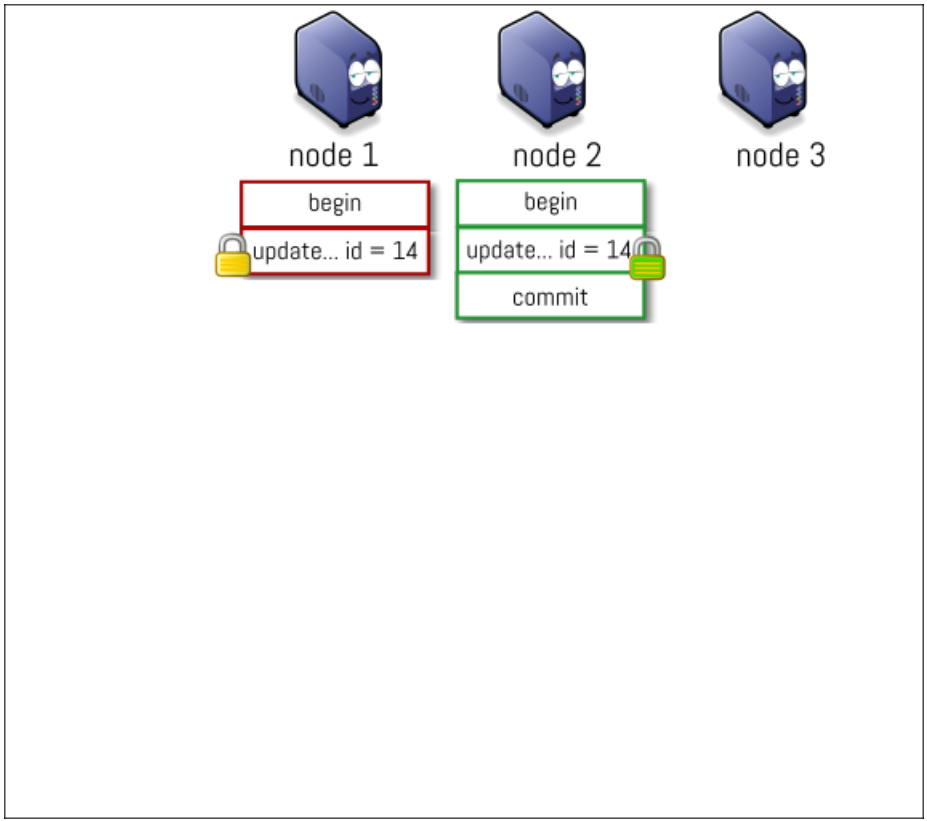
Certification Failure



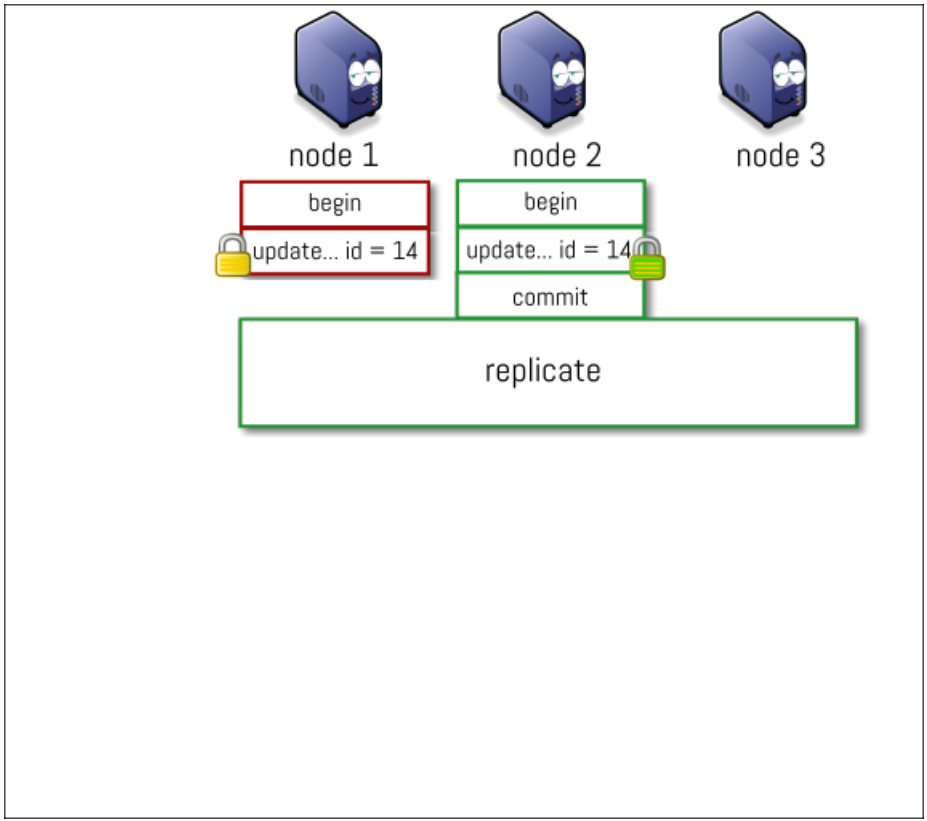
Brute Force Abort (bfa)



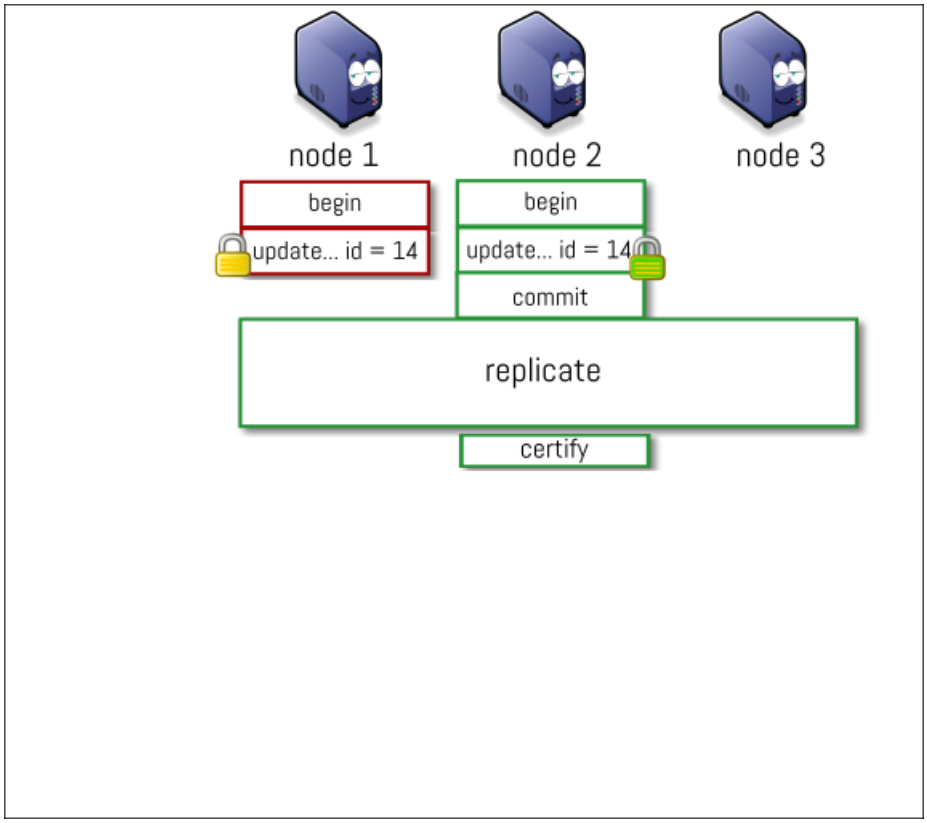
Brute Force Abort (bfa)



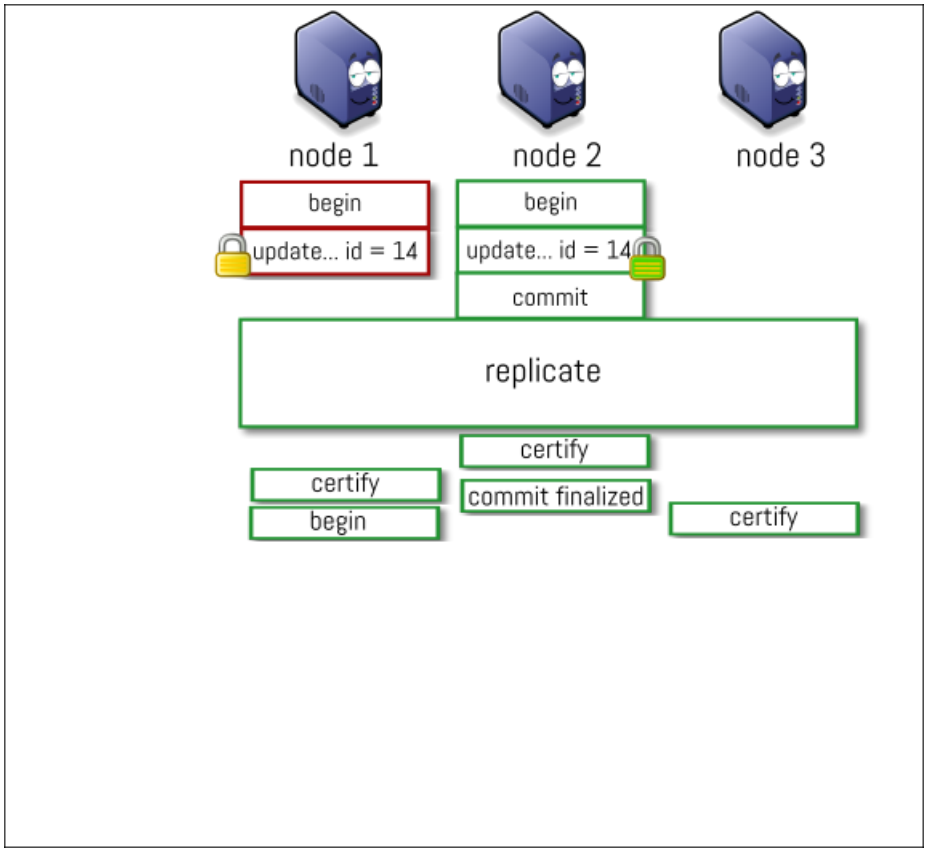
Brute Force Abort (bfa)



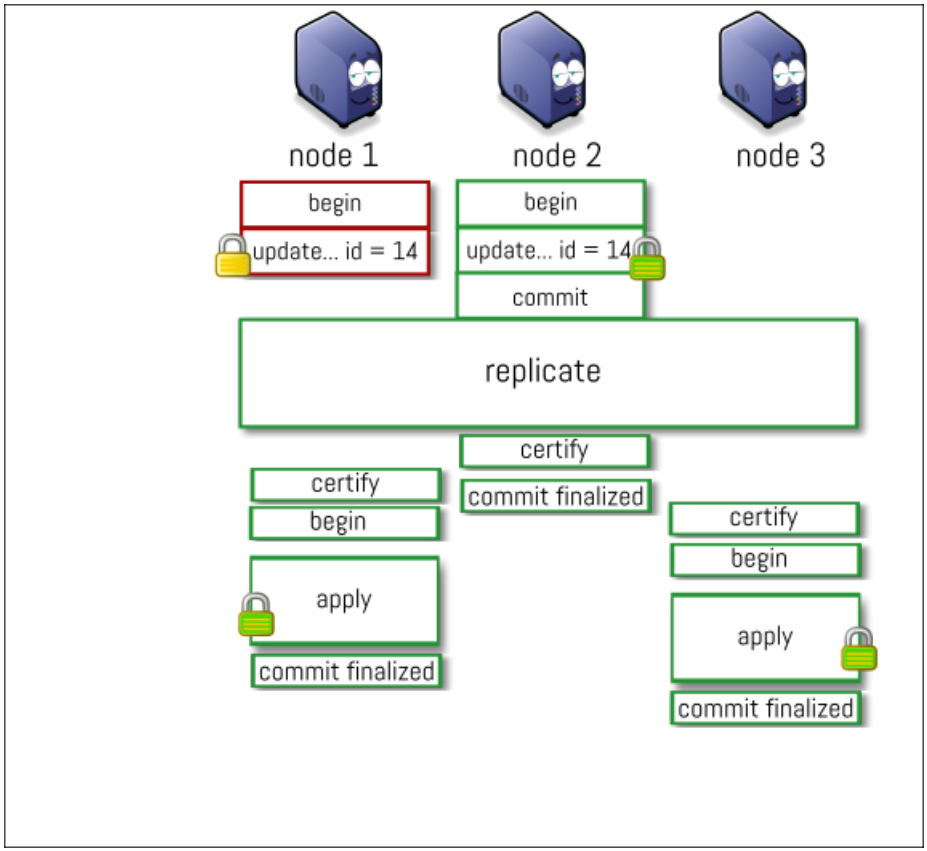
Brute Force Abort (bfa)



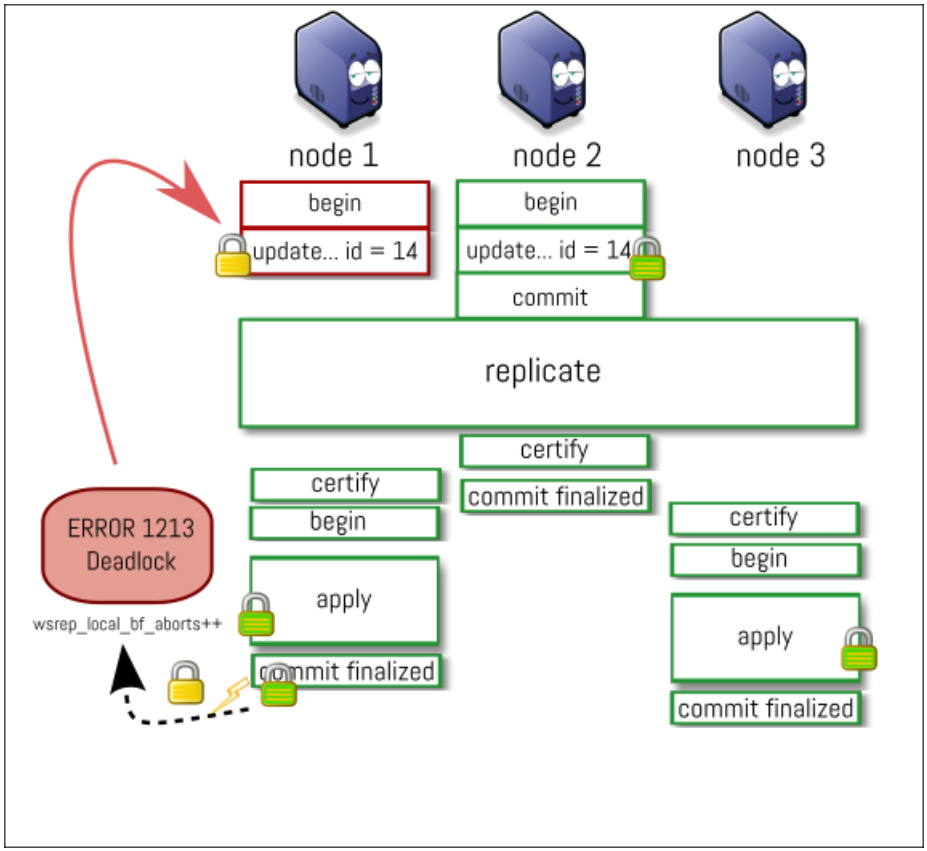
Brute Force Abort (bfa)



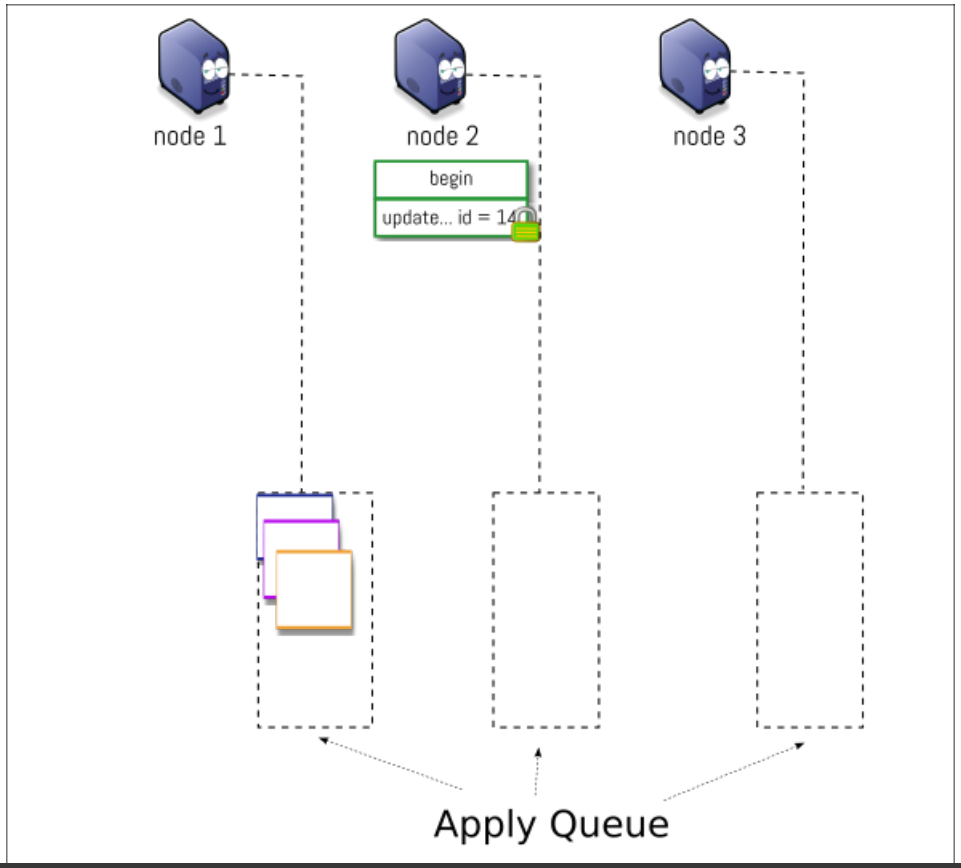
Brute Force Abort (bfa)



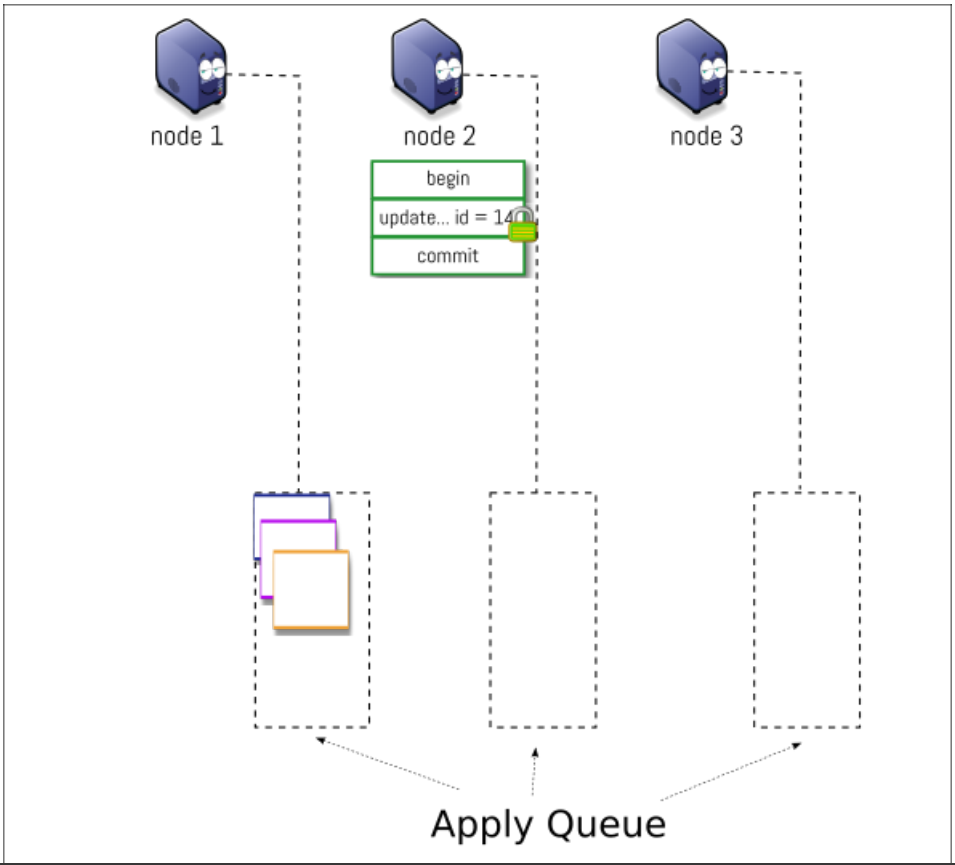
Brute Force Abort (bfa)



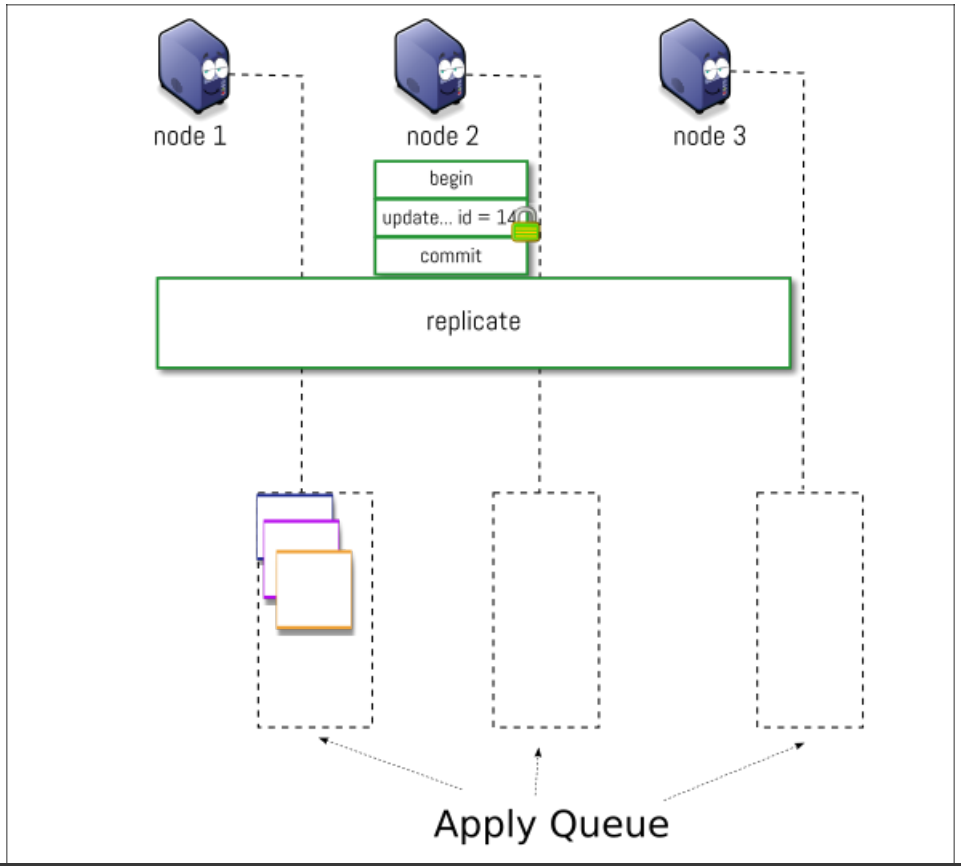
Local Certification Failure (lcf)



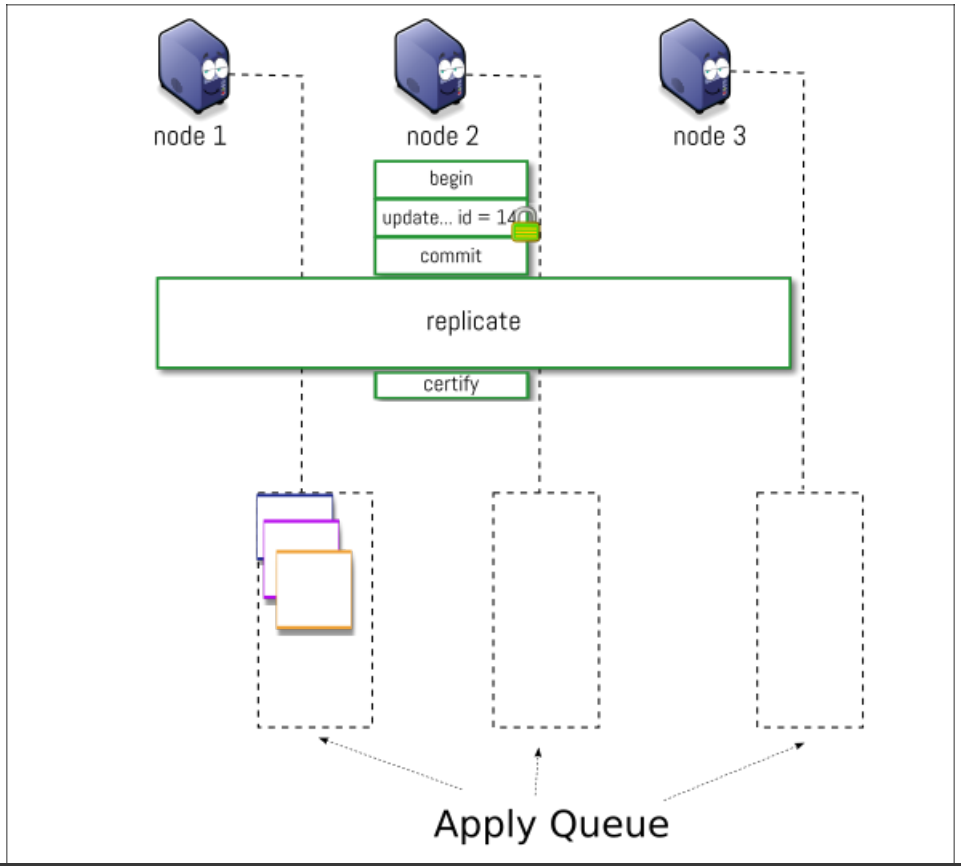
Local Certification Failure (lcf)



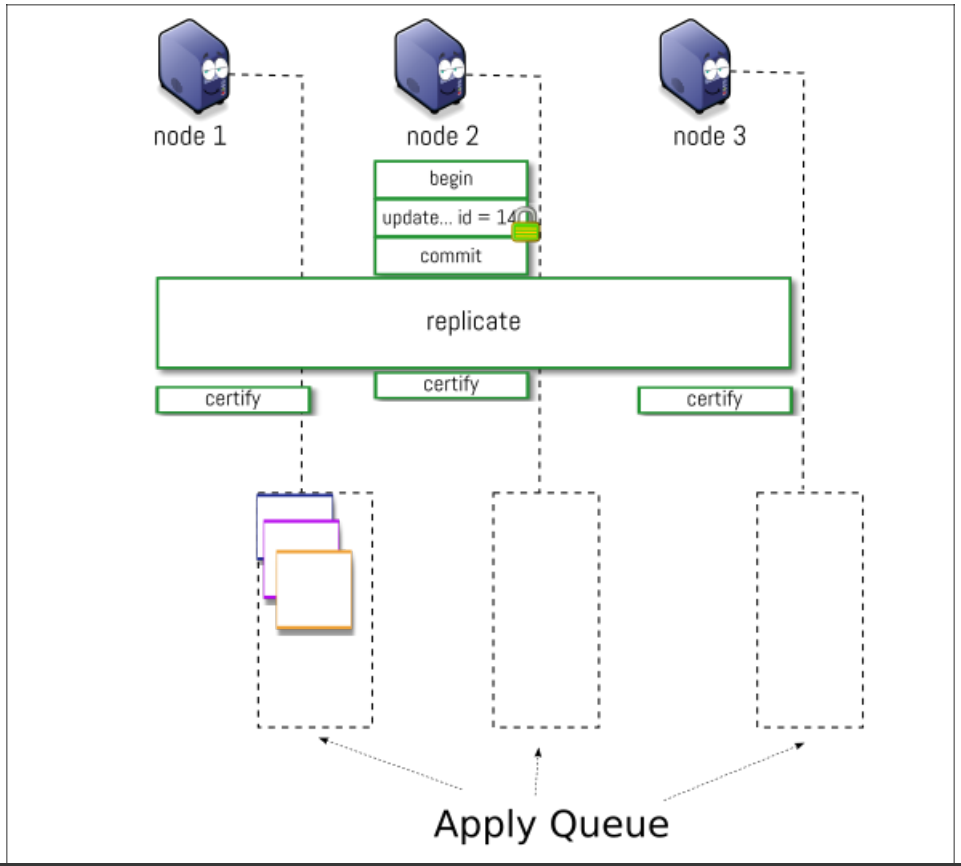
Local Certification Failure (lcf)



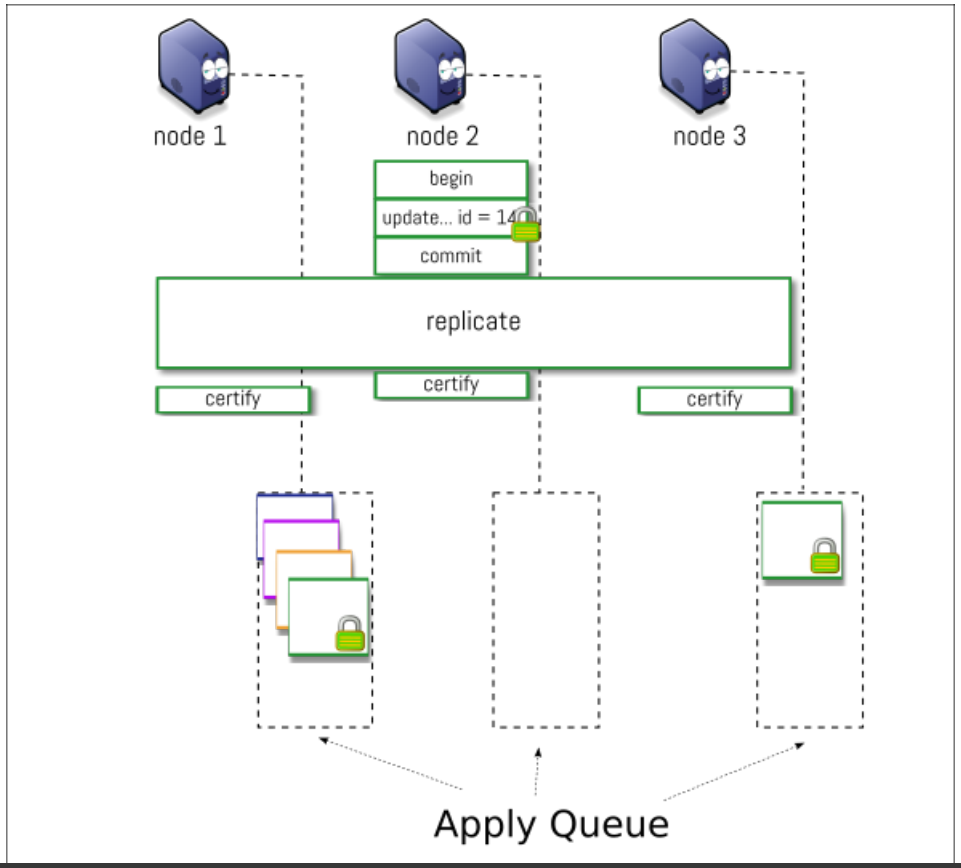
Local Certification Failure (lcf)



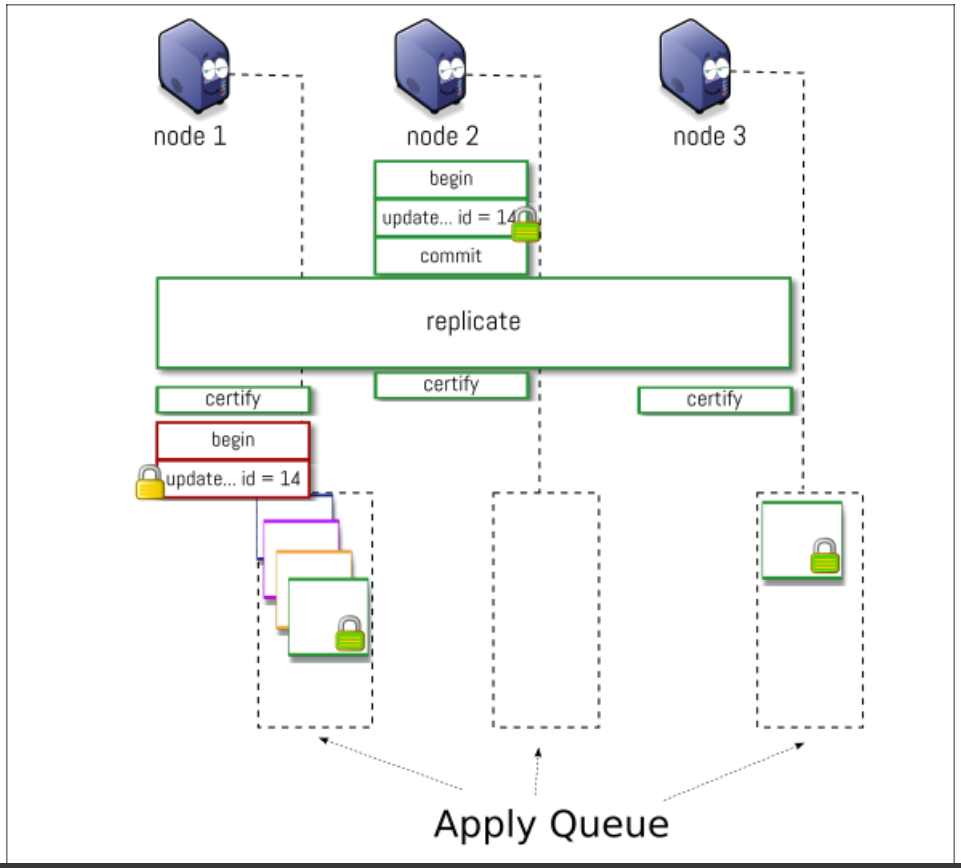
Local Certification Failure (lcf)



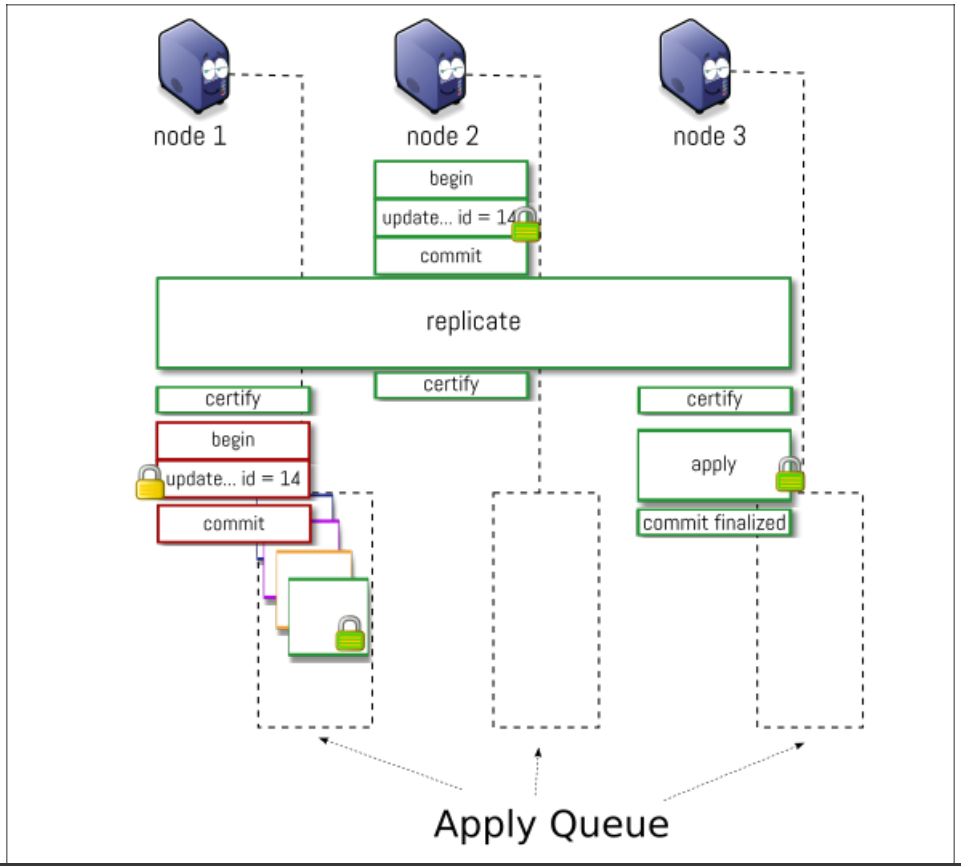
Local Certification Failure (lcf)



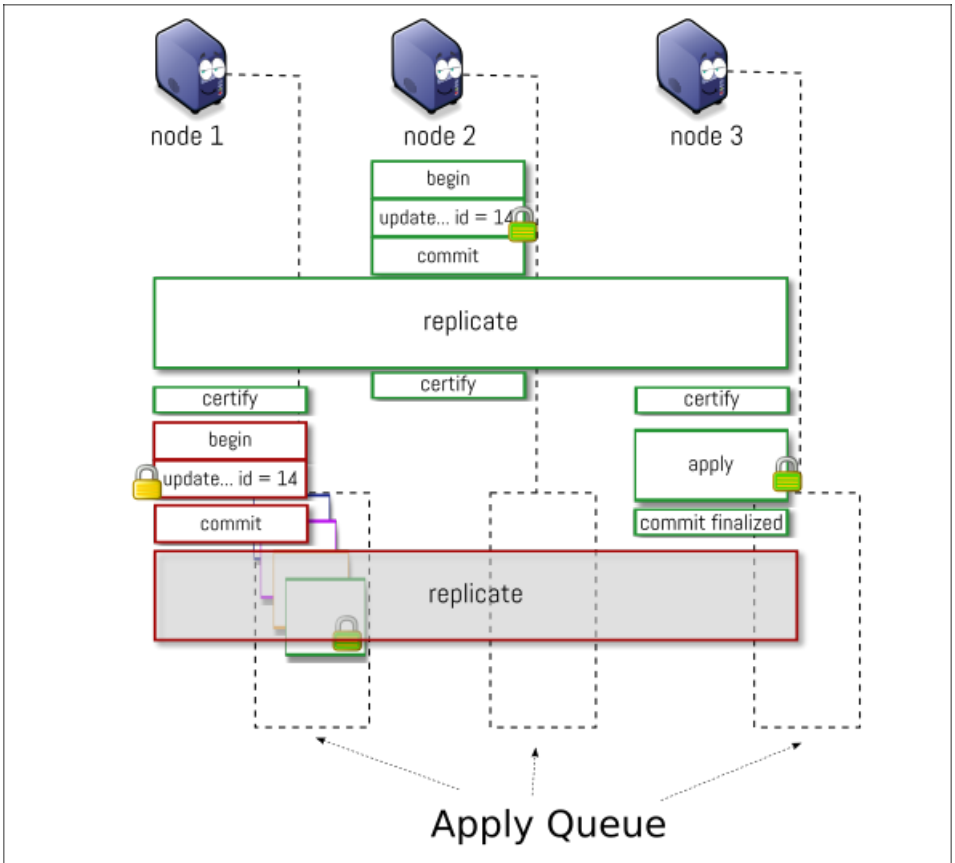
Local Certification Failure (lcf)



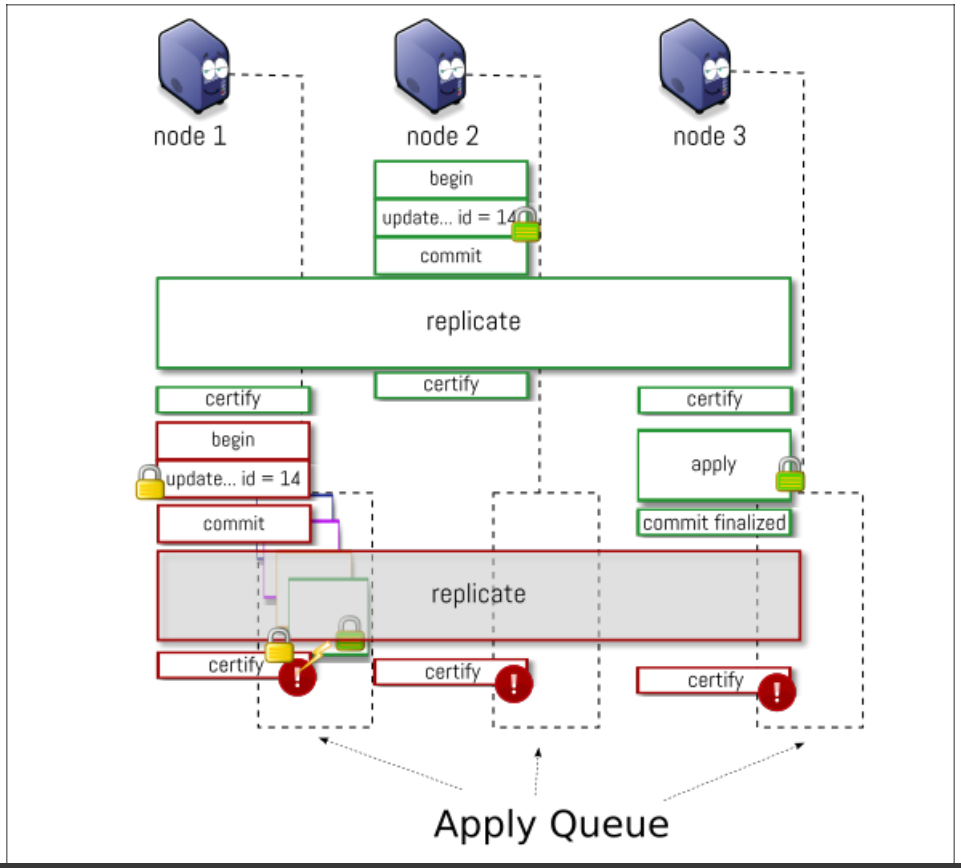
Local Certification Failure (lcf)



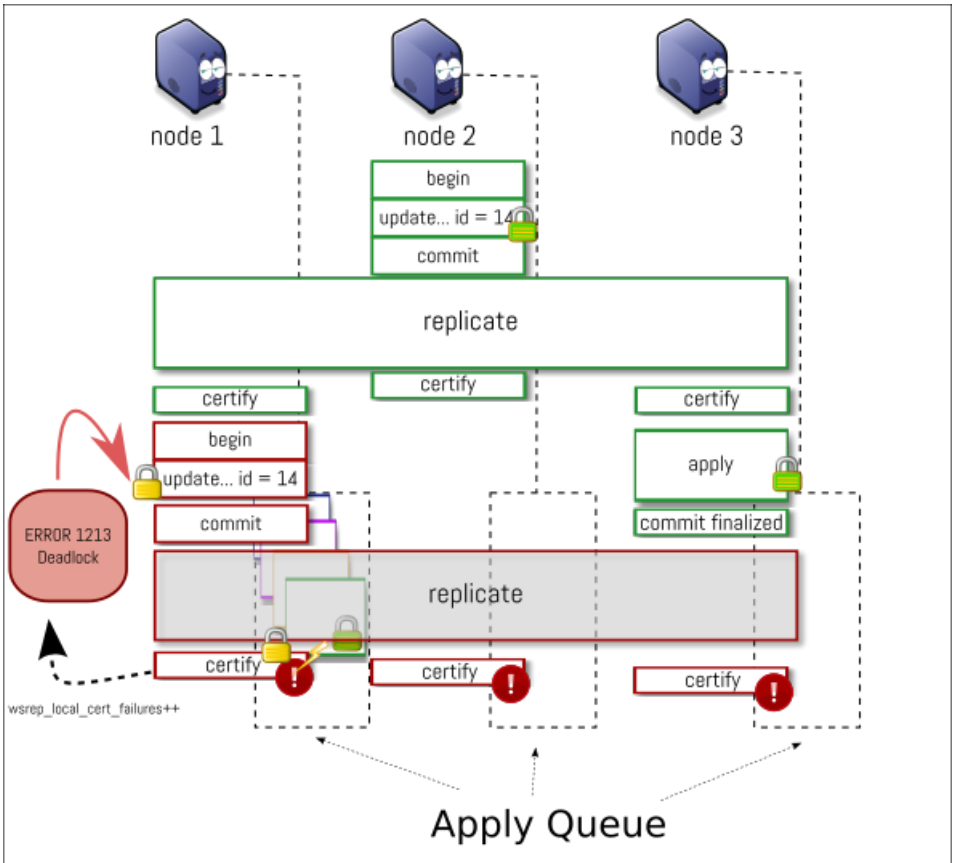
Local Certification Failure (lcf)



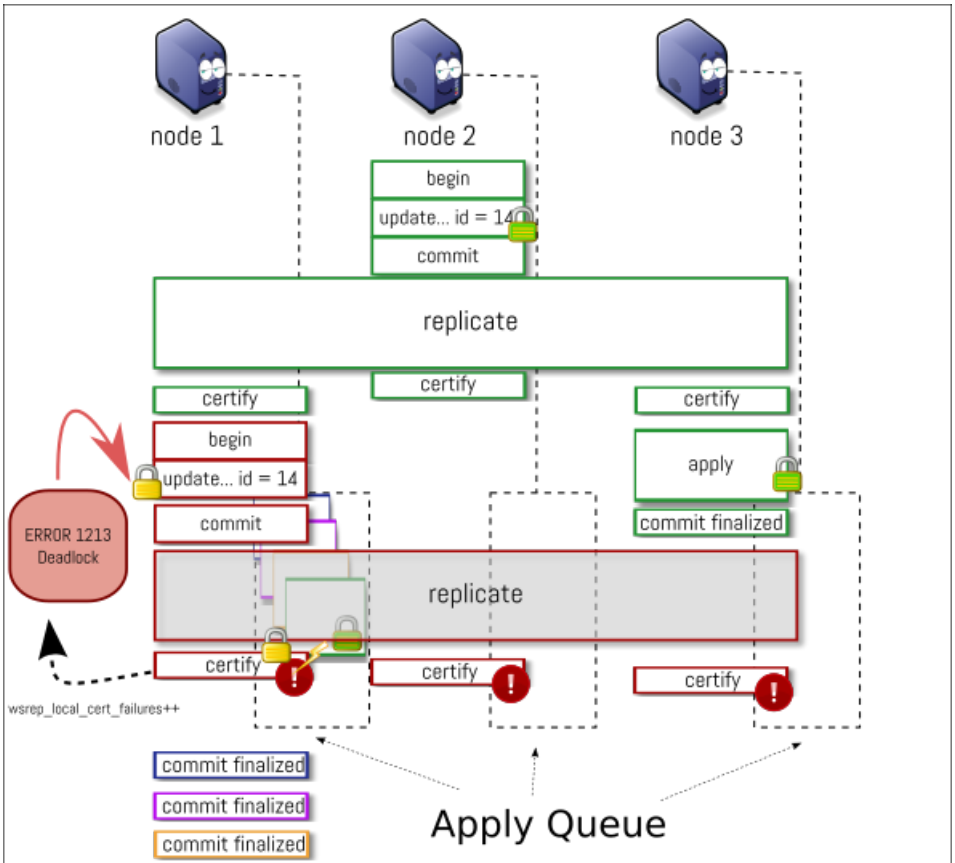
Local Certification Failure (lcf)



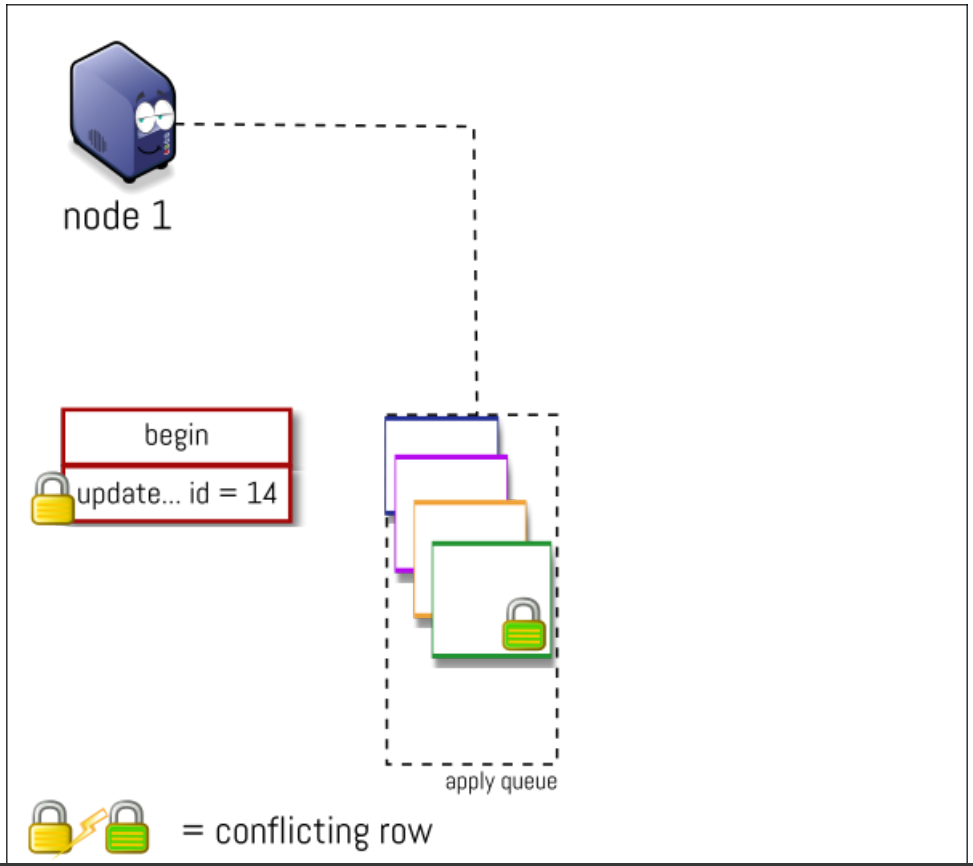
Local Certification Failure (lcf)



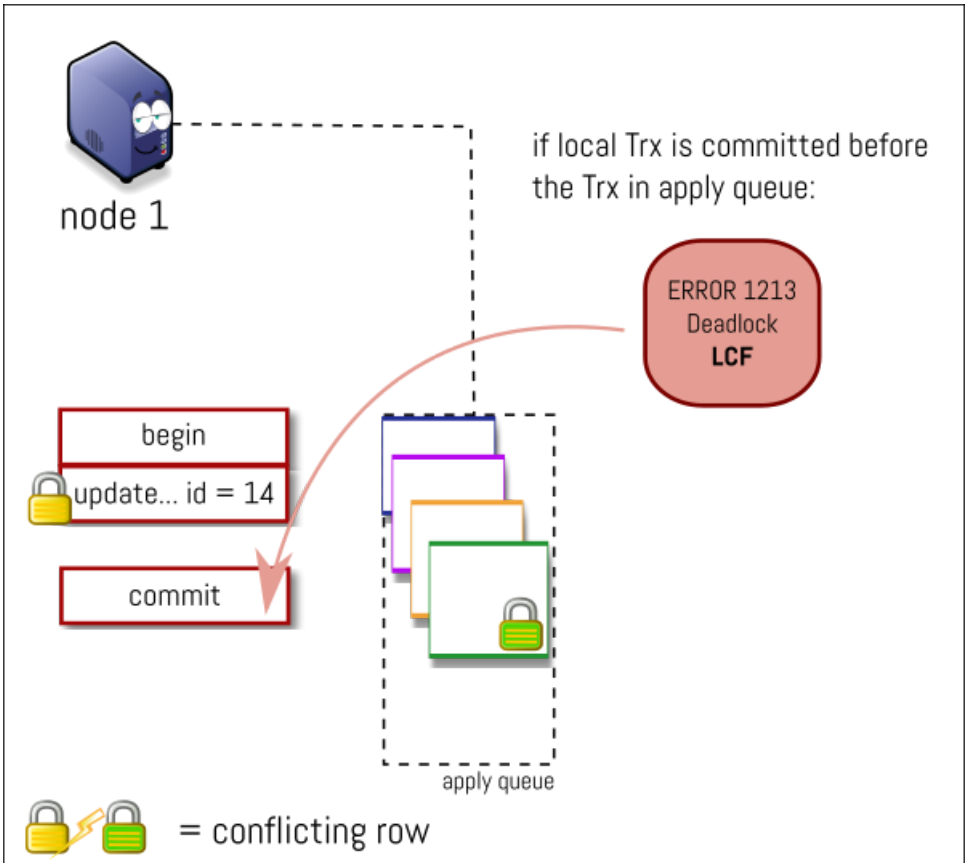
Local Certification Failure (lcf)



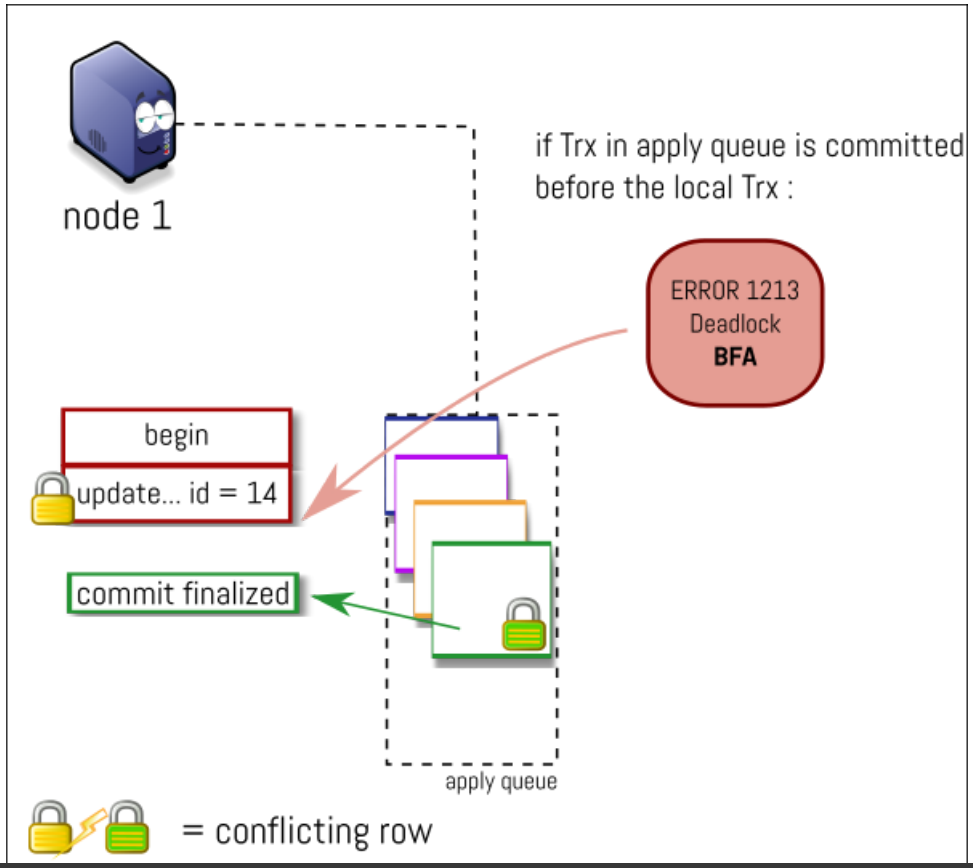
Certification Errors: summary



Certification Errors: summary



Certification Errors: summary



Flow Control

- Ability of any node in the cluster to ask the rest of the nodes to pause writes while it catches up

Flow Control

- Ability of any node in the cluster to ask the rest of the nodes to pause writes while it catches up
- Feedback mechanism for replication process

Flow Control

- Ability of any node in the cluster to ask the rest of the nodes to pause writes while it catches up
- Feedback mechanism for replication process
- ONLY caused by `wsrep_local_recv_queue` exceeding a node's `fc_limit`

Flow Control

- Ability of any node in the cluster to ask the rest of the nodes to pause writes while it catches up
- Feedback mechanism for replication process
- ONLY caused by `wsrep_local_recv_queue` exceeding a node's `fc_limit`
- CAN **pause the entire cluster** and look like a cluster stall !

Tuning Flow Control

- Too low:

Tuning Flow Control

- Too low:
  - frequent FC from any and all nodes in the cluster

Tuning Flow Control

- Too low:
  - frequent FC from any and all nodes in the cluster
- Too high:

Tuning Flow Control

- Too low:
  - frequent FC from any and all nodes in the cluster
- Too high:
  - increase in replication conflicts in multi-node writings

Tuning Flow Control

- Too low:
  - frequent FC from any and all nodes in the cluster
- Too high:
  - increase in replication conflicts in multi-node writings
  - increase in apply lag on nodes

Tuning Flow Control

- Too low:
  - frequent FC from any and all nodes in the cluster
- Too high:
  - increase in replication conflicts in multi-node writings
  - increase in apply lag on nodes
  - increase in commit lag on nodes

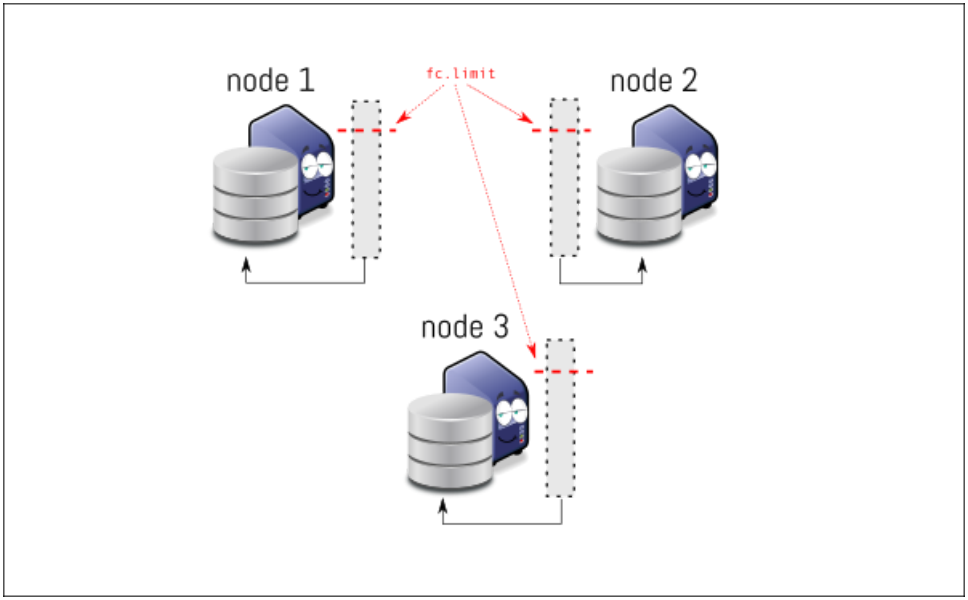
Tuning Flow Control

- Too low:
  - frequent FC from any and all nodes in the cluster
- Too high:
  - increase in replication conflicts in multi-node writings
  - increase in apply lag on nodes
  - increase in commit lag on nodes
- One node with FC issues

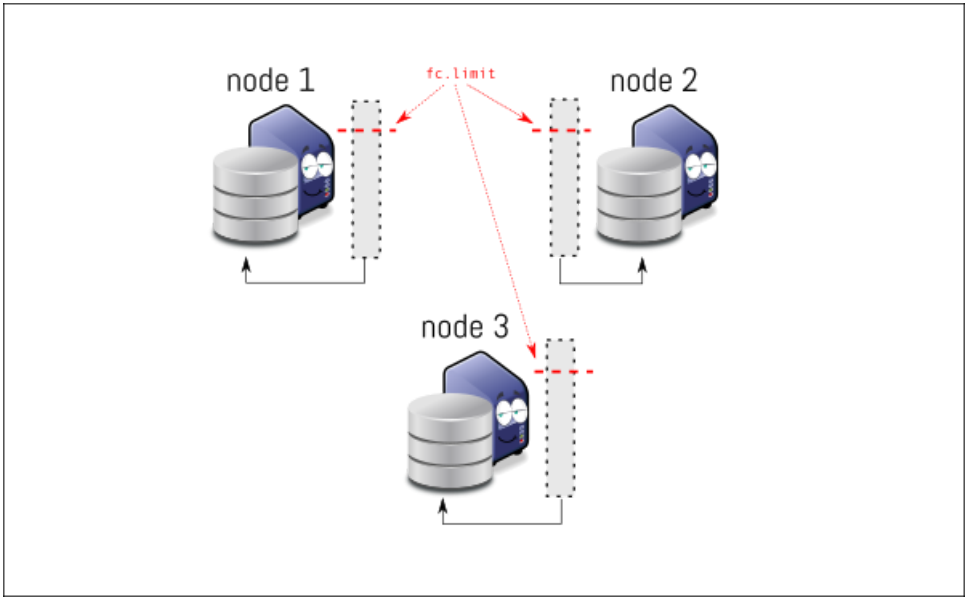
Tuning Flow Control

- Too low:
  - frequent FC from any and all nodes in the cluster
- Too high:
  - increase in replication conflicts in multi-node writings
  - increase in apply lag on nodes
  - increase in commit lag on nodes
- One node with FC issues
  - deal with that node -bad hardware? too slow ?

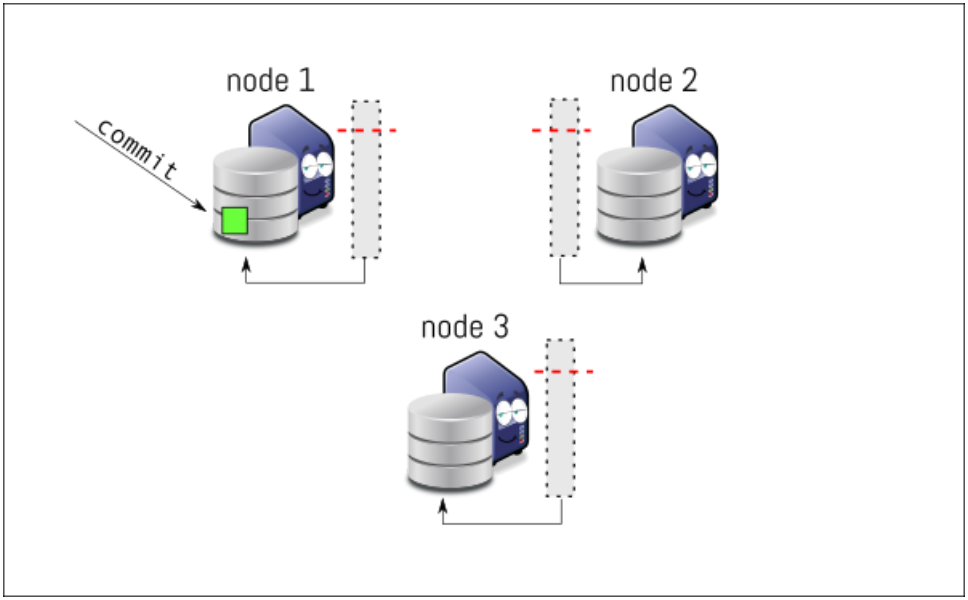
Flow Control



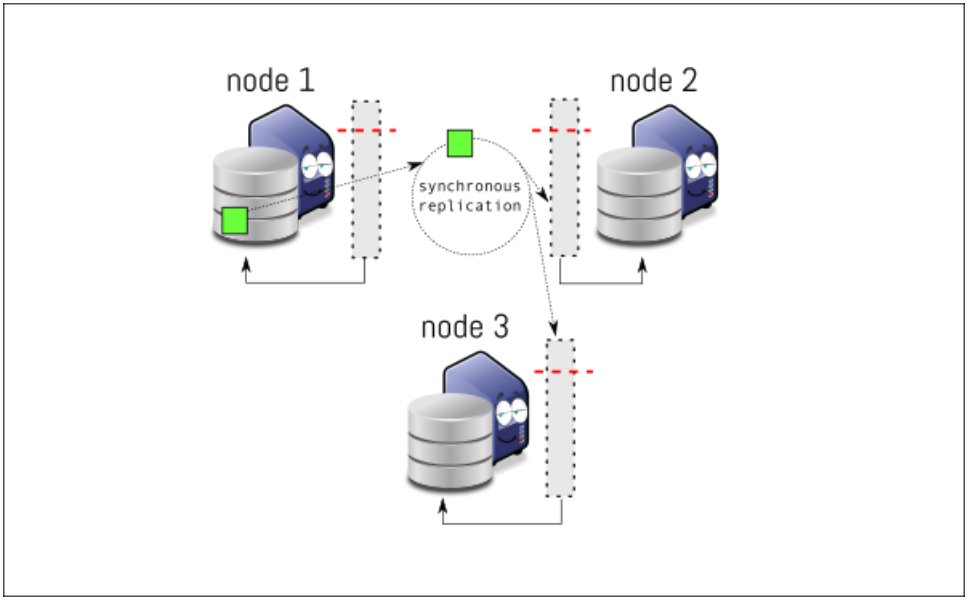
Flow Control



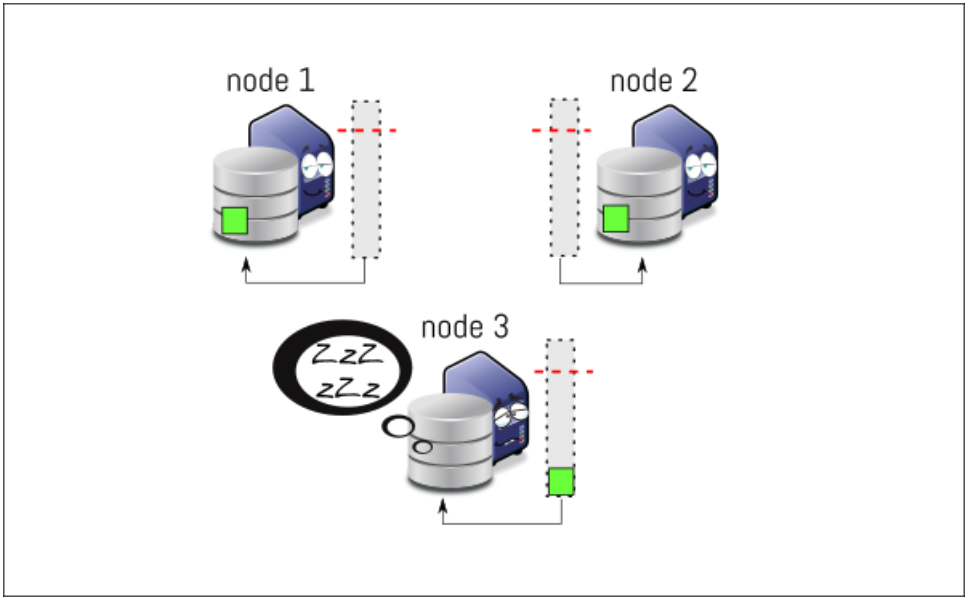
Flow Control



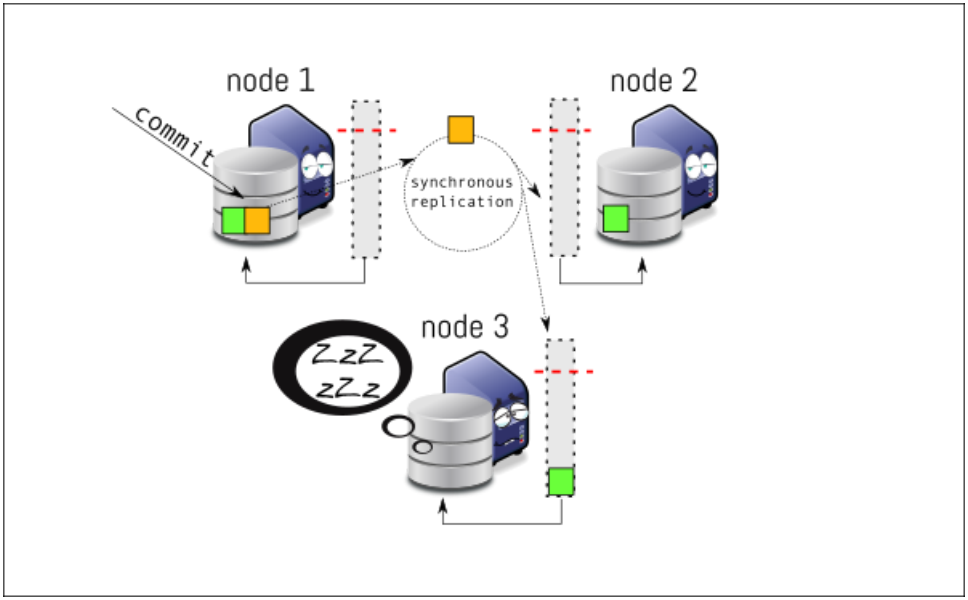
Flow Control



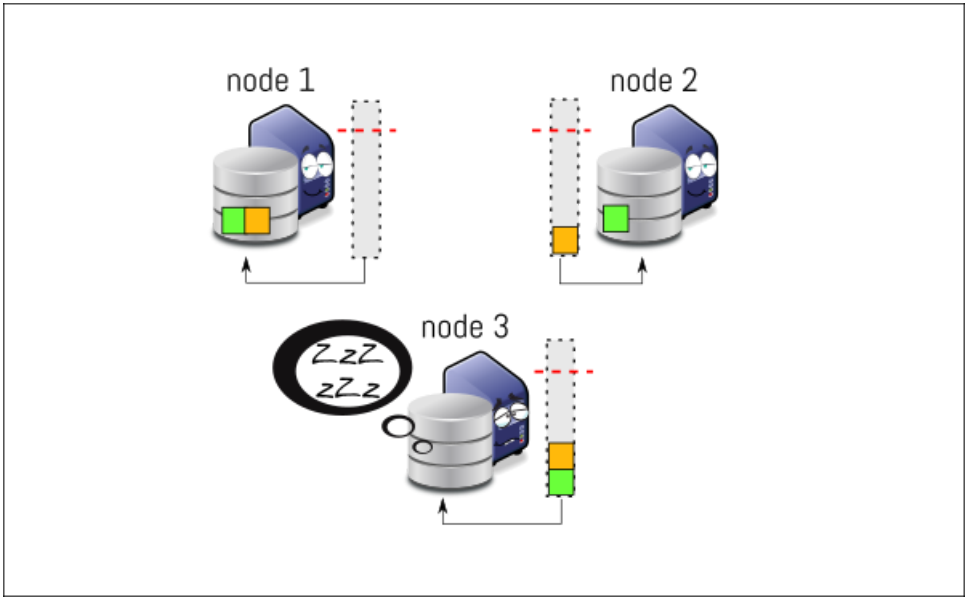
Flow Control



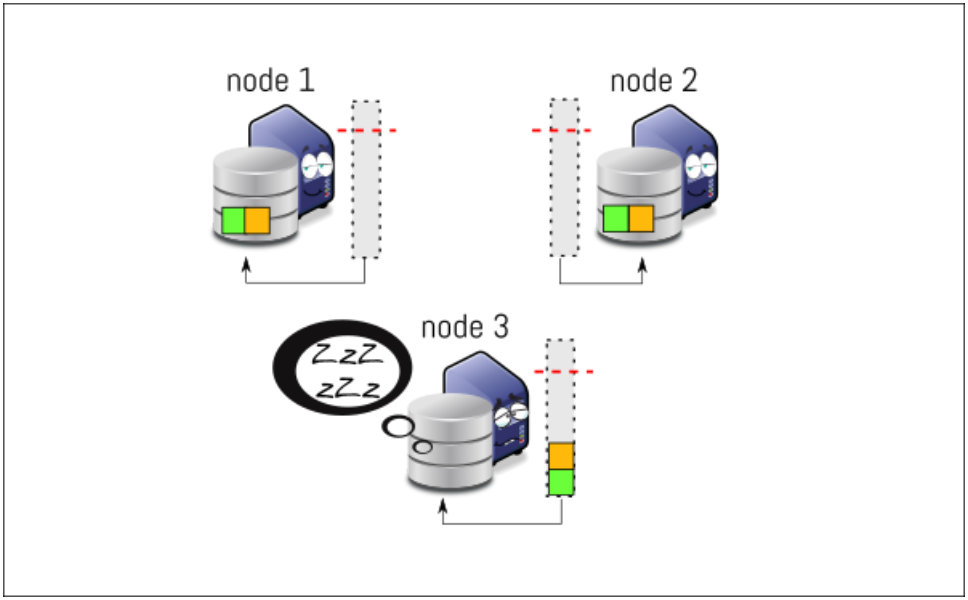
Flow Control



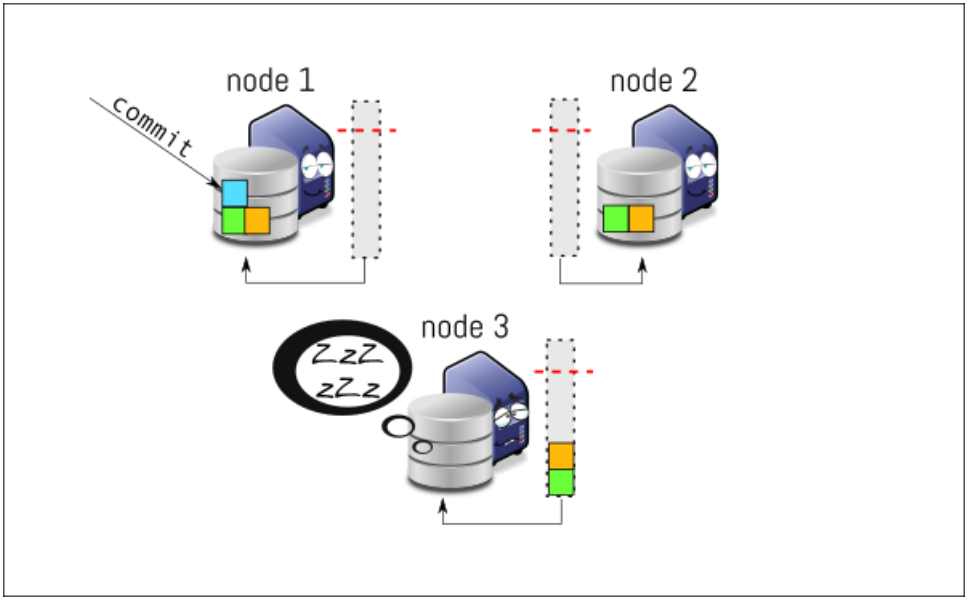
Flow Control



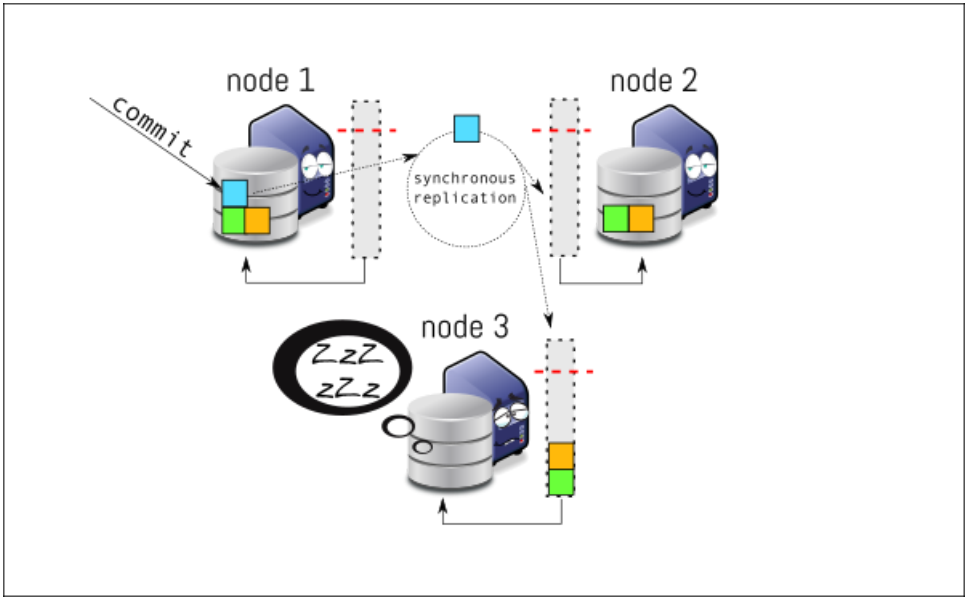
Flow Control



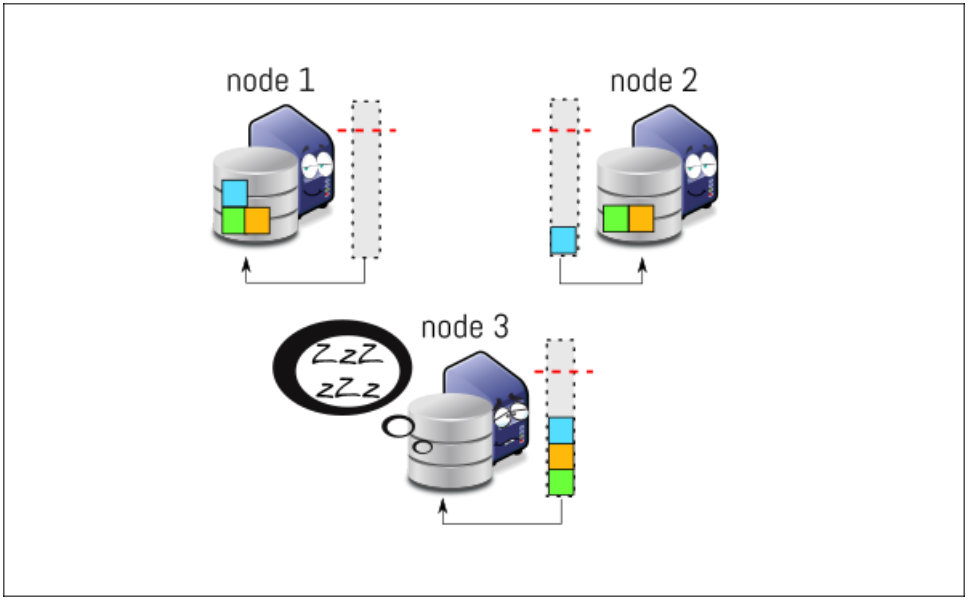
Flow Control



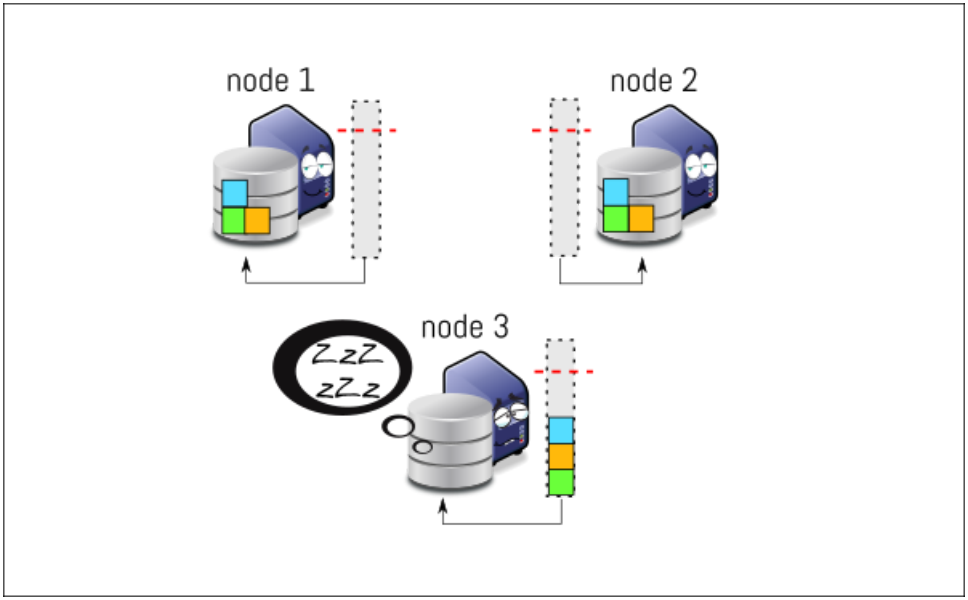
Flow Control



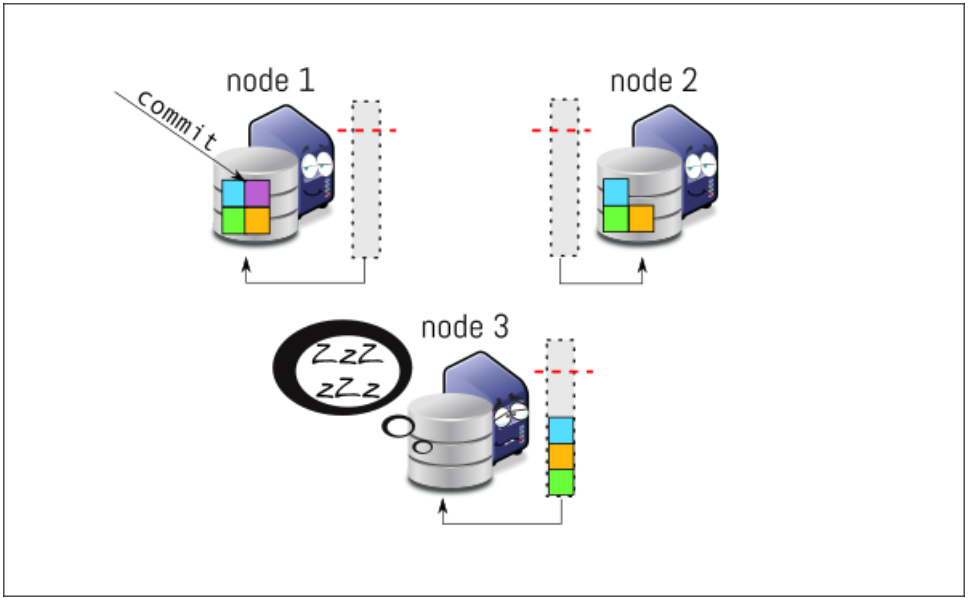
Flow Control



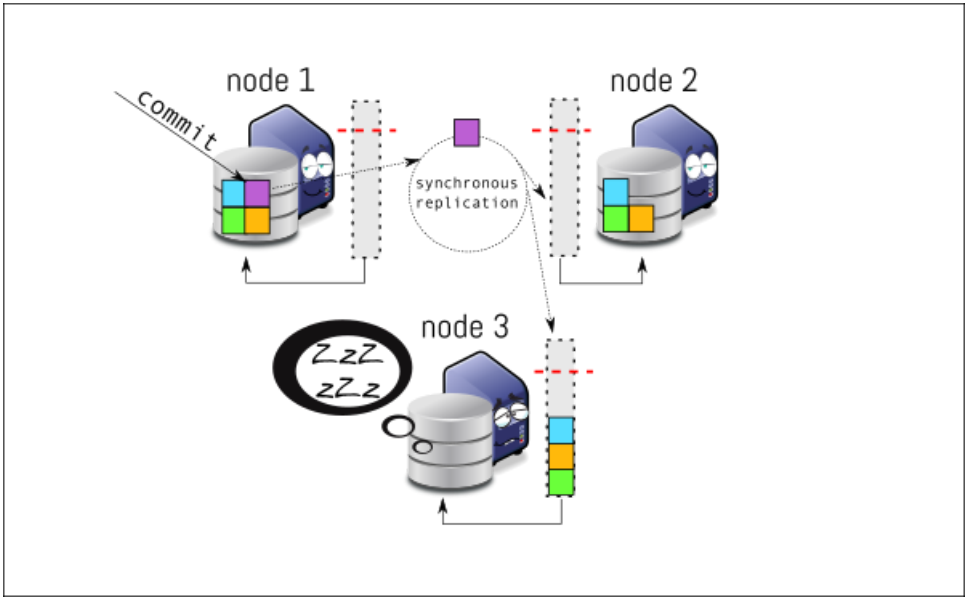
Flow Control



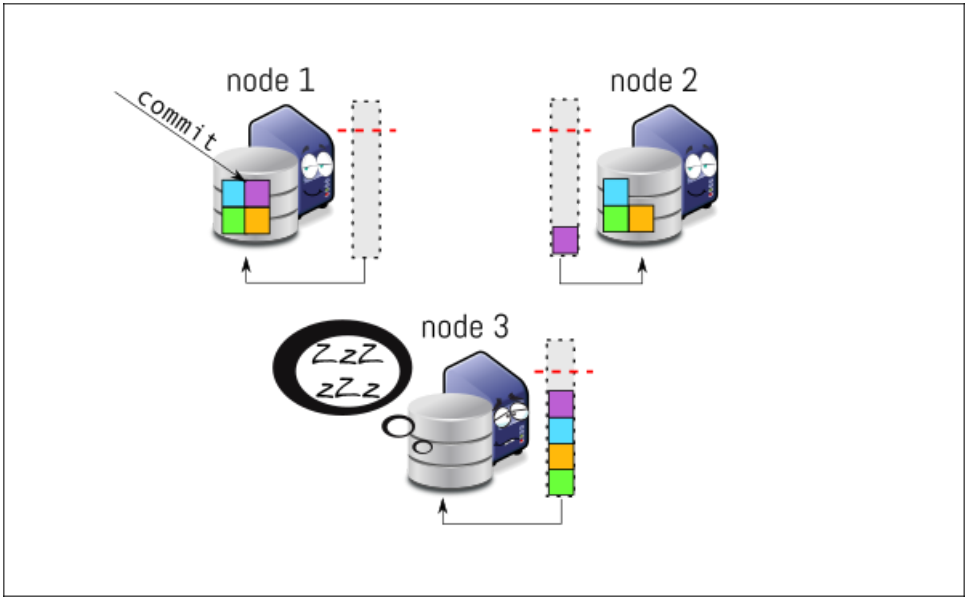
Flow Control



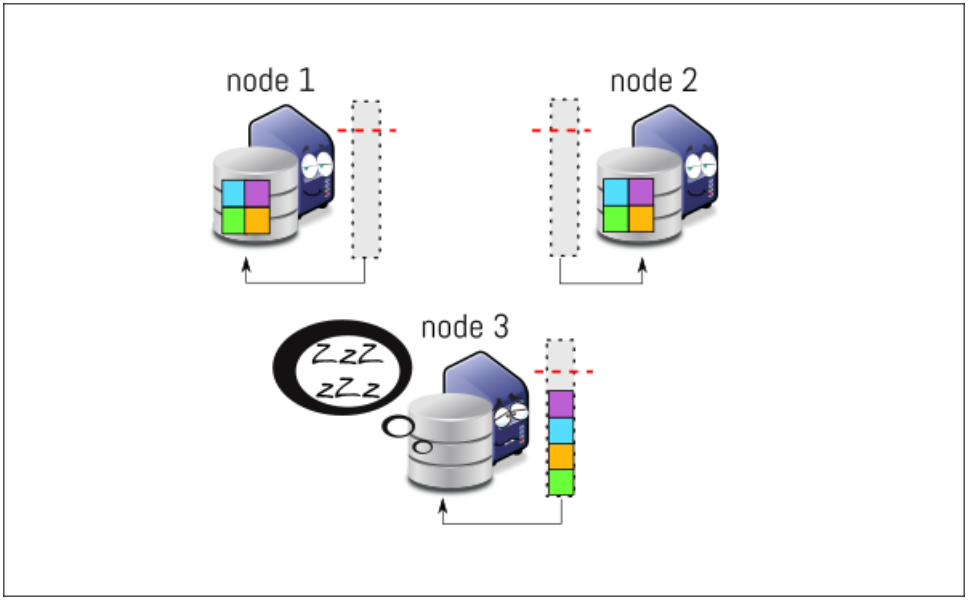
Flow Control



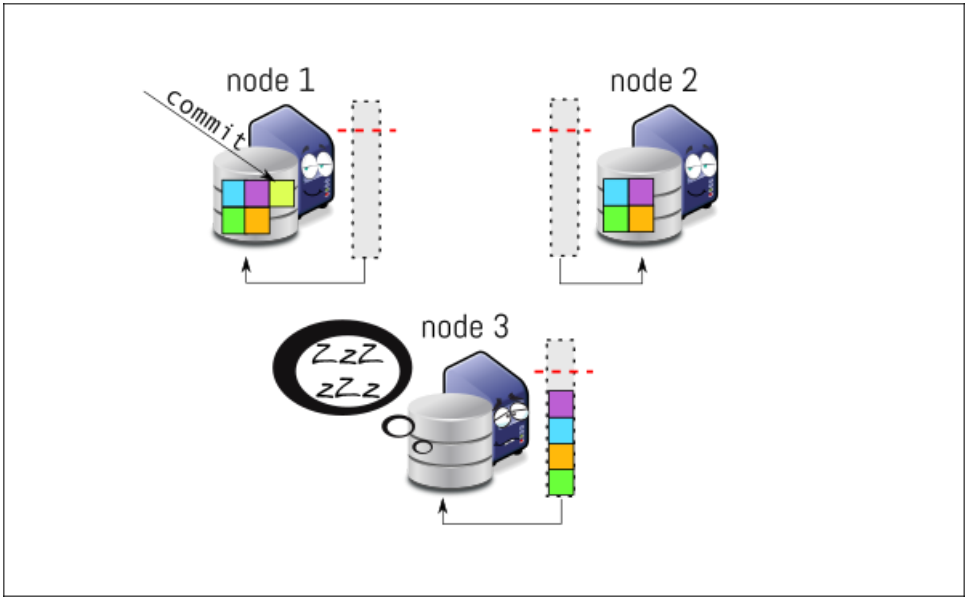
Flow Control



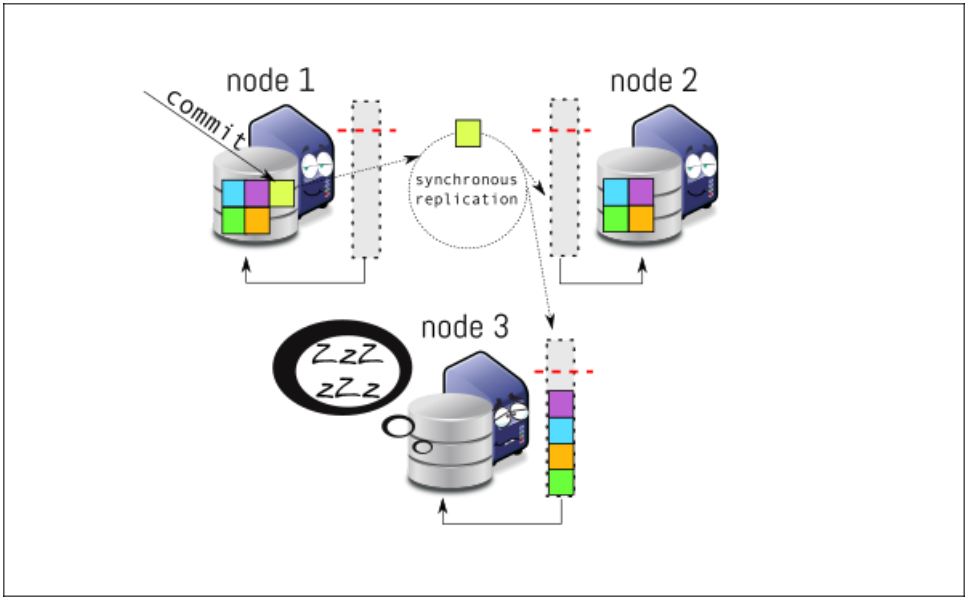
Flow Control



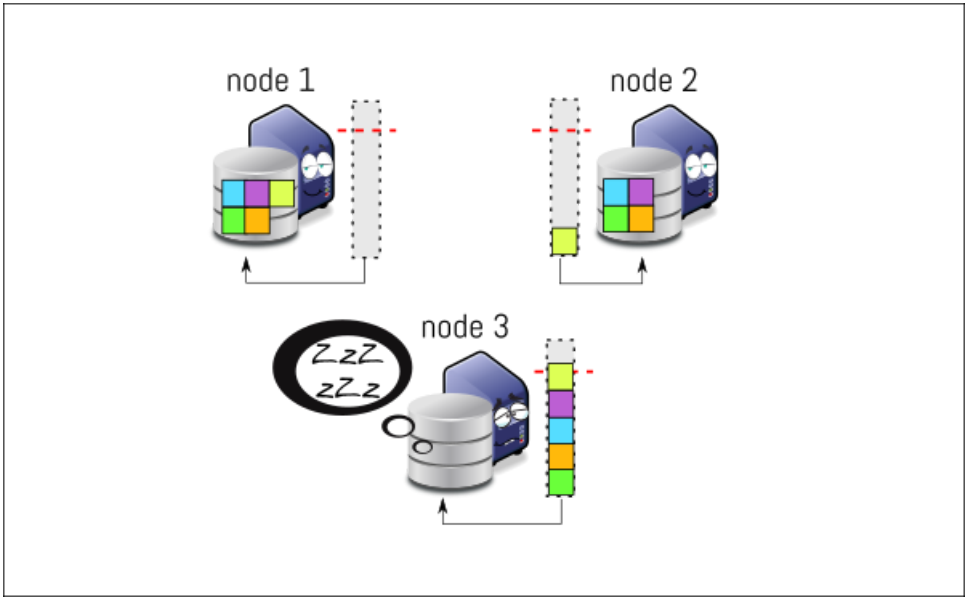
Flow Control



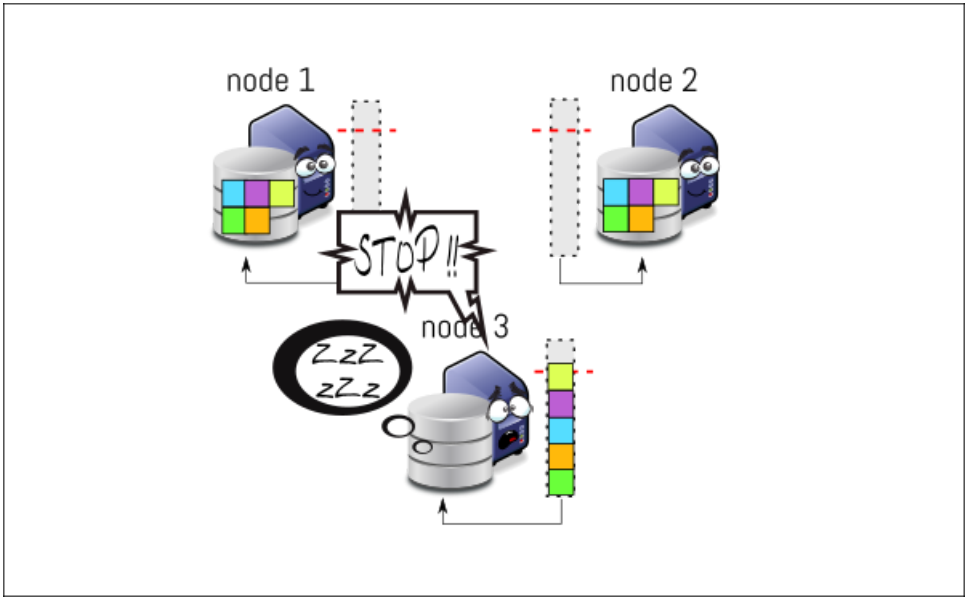
Flow Control



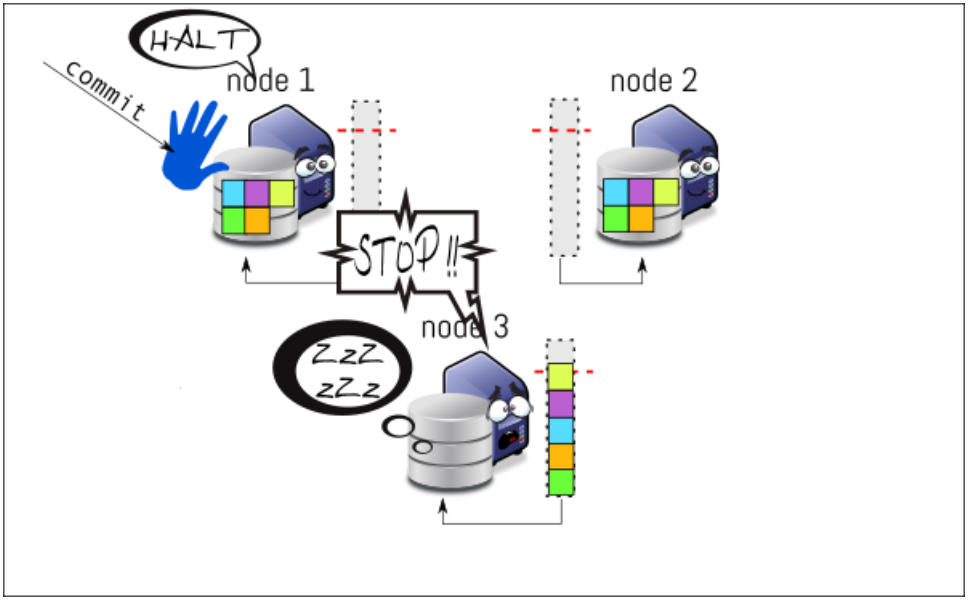
Flow Control



Flow Control



Flow Control



State Transfer

There are two types of State Transfer in Galera:

State Transfer

There are two types of State Transfer in Galera:

1. **SST** (Snapshot State Transfer): full data copy

State Transfer

There are two types of State Transfer in Galera:

1. **SST** (Snapshot State Transfer): full data copy
   - rsync
   - mysqldump
   - xtrabackup

State Transfer

There are two types of State Transfer in Galera:

1. **SST** (Snapshot State Transfer): full data copy
   - rsync
   - mysqldump
   - xtrabackup
2. **IST** (Incremental State Transfer): only copy the missing events

State Transfer

There are two types of State Transfer in Galera:

1. **SST** (Snapshot State Transfer): full data copy
   - rsync
   - mysqldump
   - xtrabackup
2. **IST** (Incremental State Transfer): only copy the missing events

It's always better to try to avoid SST!

State Transfer

There are two types of State Transfer in Galera:

1. **SST** (Snapshot State Transfer): full data copy
   - rsync
   - mysqldump
   - xtrabackup
2. **IST** (Incremental State Transfer): only copy the missing events

It's always better to try to avoid SST!

`wsrep_sst_donor` can be used to specify the donor

State Transfer: grastate.dat

- When MySQL starts it checks `grastate.dat` file (in datadir)

State Transfer: grastate.dat

- When MySQL starts it checks `grastate.dat` file (in datadir)
- This file placeholds GTID between MySQL restarts

State Transfer: grastate.dat

- When MySQL starts it checks `grastate.dat` file (in datadir)
- This file placeholds GTID between MySQL restarts
- Contains UUID and Seqno

State Transfer: grastate.dat

- When MySQL starts it checks `grastate.dat` file (in datadir)
- This file placeholds GTID between MySQL restarts
- Contains UUID and Seqno

```
# GALERA saved state
version: 2.1
uuid:    dbd5a104-5cad-11e5-809a-e279a68254b7
seqno:   14001
cert_index:
```

State Transfer: grastate.dat

- When MySQL starts it checks `grastate.dat` file (in datadir)
- This file placeholds GTID between MySQL restarts
- Contains UUID and Seqno

```
# GALERA saved state
version: 2.1
uuid:    dbd5a104-5cad-11e5-809a-e279a68254b7
seqno:   14001
cert_index:
```

node shutdown cleanly

grastate.dat - example

```
# GALERA saved state
version: 2.1
uuid:    dbd5a104-5cad-11e5-809a-e279a68254b7
seqno:   -1
cert_index:
```

grastate.dat - example

```
# GALERA saved state
version: 2.1
uuid:    dbd5a104-5cad-11e5-809a-e279a68254b7
seqno:   -1
cert_index:
```

node is running or did not shutdown cleanly

grastate.dat - example

```
# GALERA saved state
version: 2.1
uuid:    00000000-0000-000000000-000000000000
seqno:   -1
cert_index:
```

grastate.dat - example

```
# GALERA saved state
version: 2.1
uuid:    00000000-0000-000000000-000000000000
seqno:   -1
cert_index:
```

node aborted, SST on next restart

State Transfer: IST

When a node starts, it knowns the **UUID** of the cluster it belonged and the **last sequence number** it applied.

So, it sends that position to the other members of the cluster and if a node can send the next events (ws/trx), IST will be performed, if none, then SST will be triggered.

Galera Cache

Those events are stored on the `galera.cache` file.

Galera Cache

Those events are stored on the `galera.cache` file.

- preallocated file with a specific size

Galera Cache

Those events are stored on the `galera.cache` file.

- preallocated file with a specific size
- used to store the writesets in circular buffer style

Galera Cache

Those events are stored on the `galera.cache` file.

- preallocated file with a specific size
- used to store the writesets in circular buffer style
- default size is 128M

Galera Cache

Those events are stored on the `galera.cache` file.

- preallocated file with a specific size
- used to store the writesets in circular buffer style
- default size is 128M
- can be increase via provider option `gcache.size`

Galera Cache

Those events are stored on the `galera.cache` file.

- preallocated file with a specific size
- used to store the writesets in circular buffer style
- default size is 128M
- can be increase via provider option `gcache.size`
  - `wsrep_provider_options = "gcache.size=1G"`

Galera Cache

Those events are stored on the `galera.cache` file.

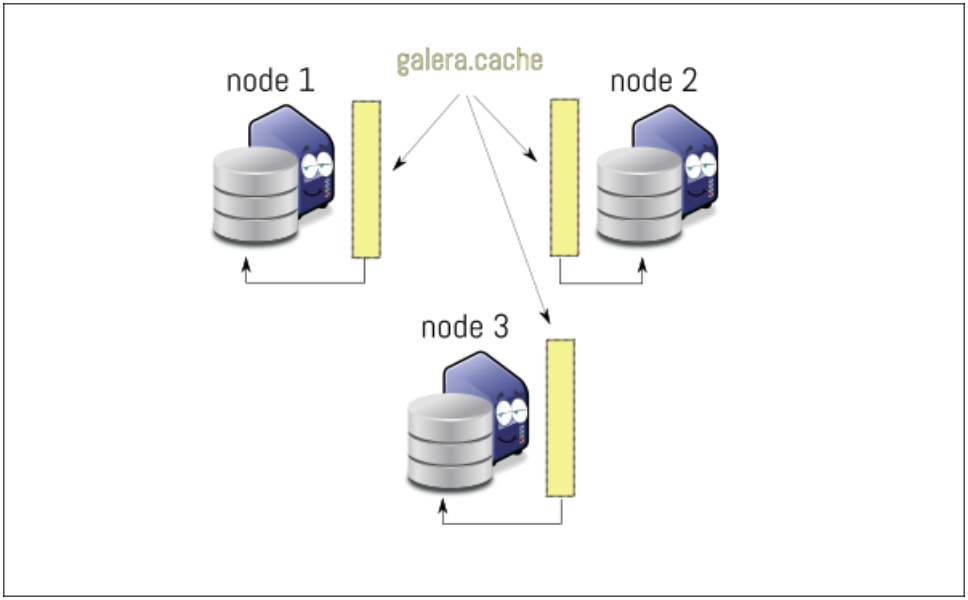
- preallocated file with a specific size
- used to store the writesets in circular buffer style
- default size is 128M
- can be increase via provider option `gcache.size`
  - `wsrep_provider_options = "gcache.size=1G"`
- Galare Cache is mmaped (I/O buffered to memory)

Galera Cache

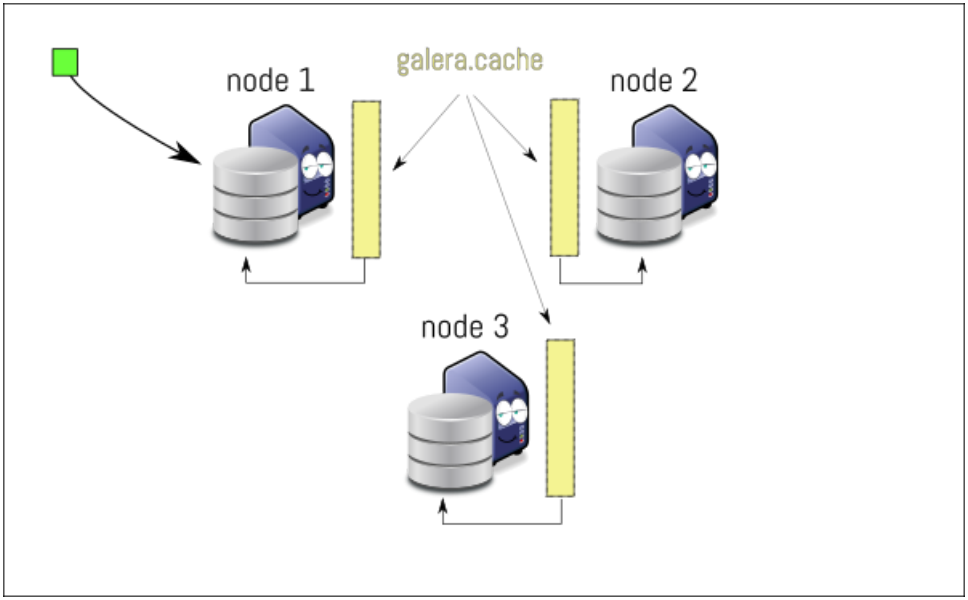
Those events are stored on the `galera.cache` file.

- preallocated file with a specific size
- used to store the writesets in circular buffer style
- default size is 128M
- can be increase via provider option `gcache.size`
  - `wsrep_provider_options = "gcache.size=1G"`
- Galare Cache is mmaped (I/O buffered to memory)
- `wsrep_local_cached_downto` provide the first seqno present in the cache for that node

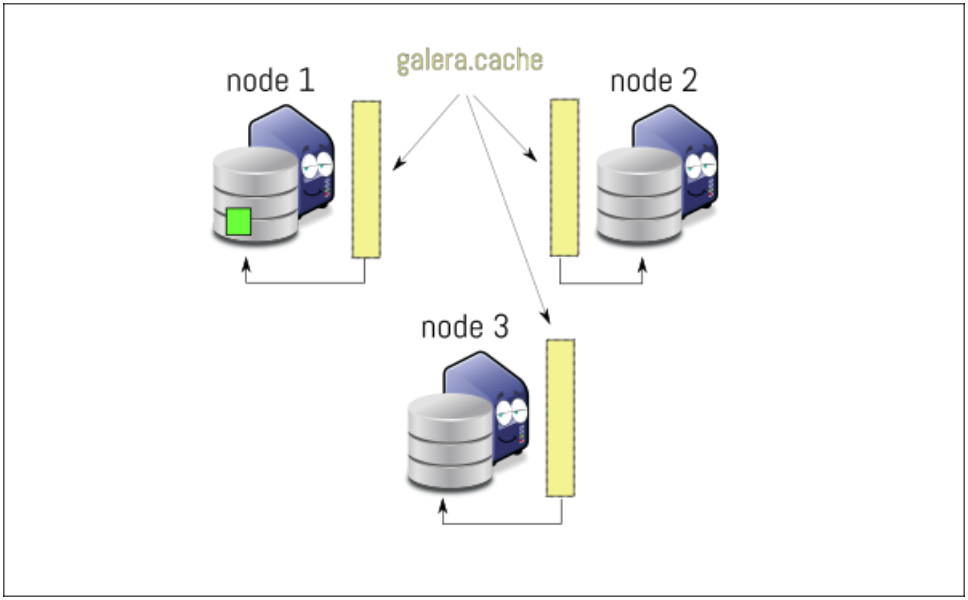
Galera Cache & IST



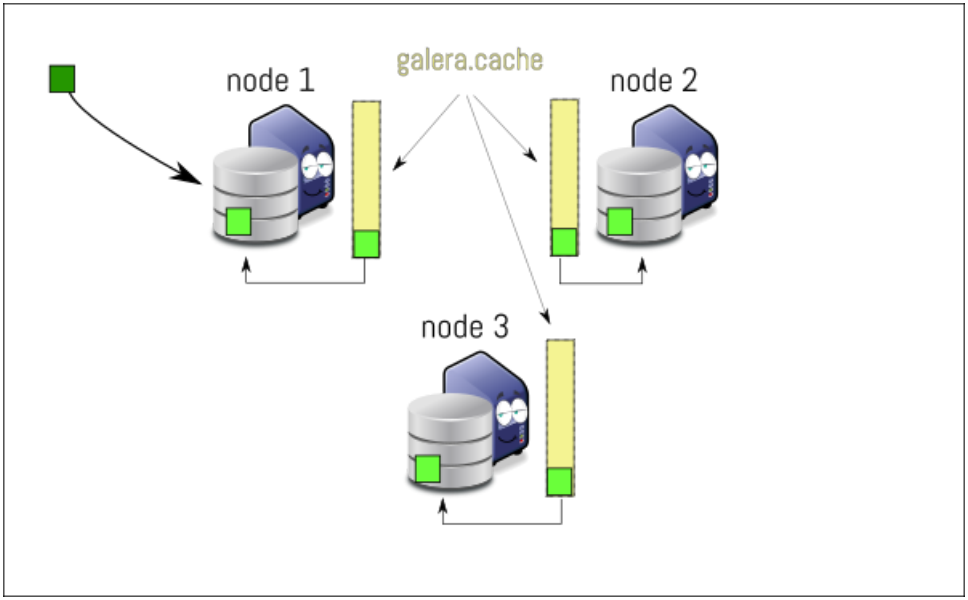
Galera Cache & IST



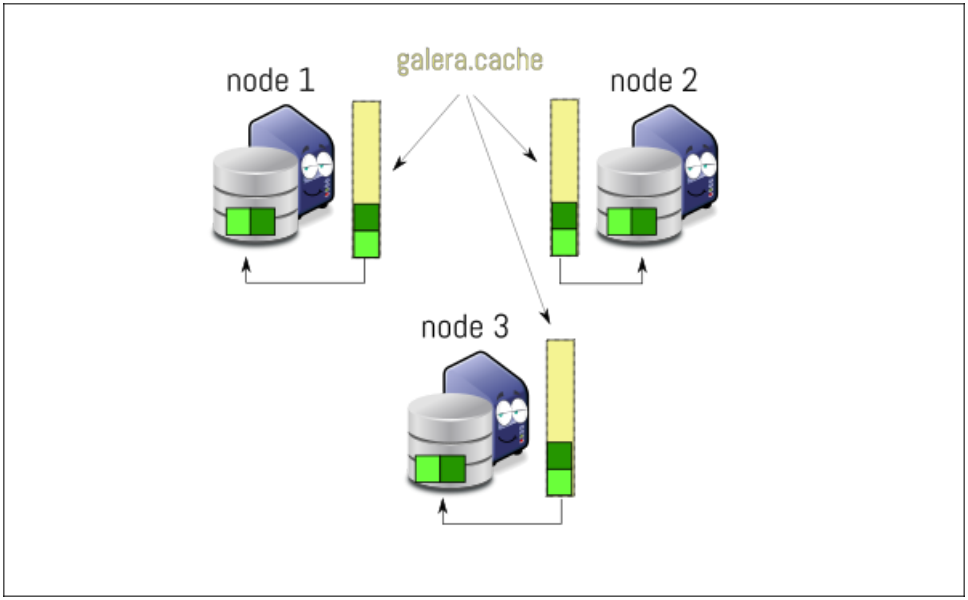
Galera Cache & IST



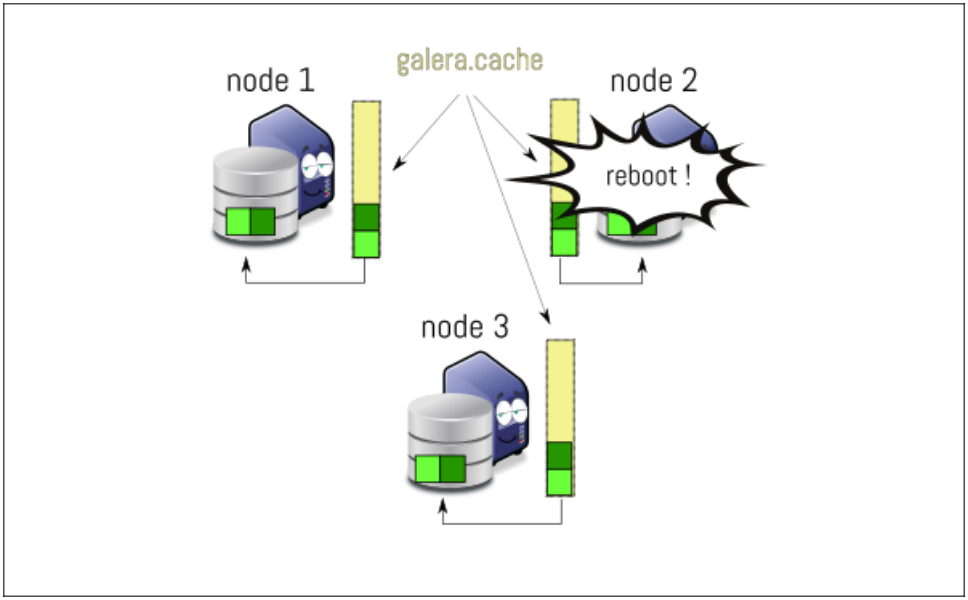
Galera Cache & IST



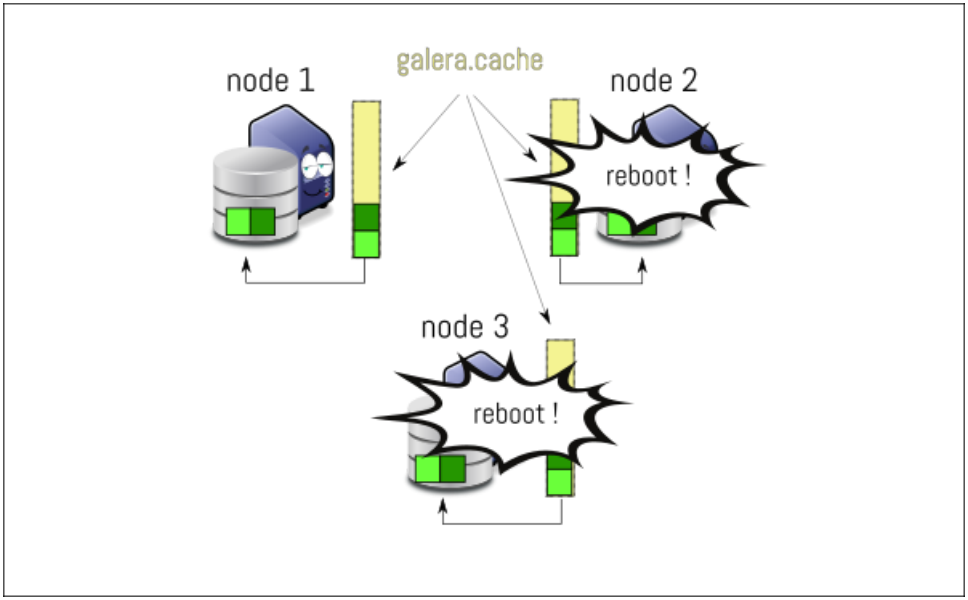
Galera Cache & IST



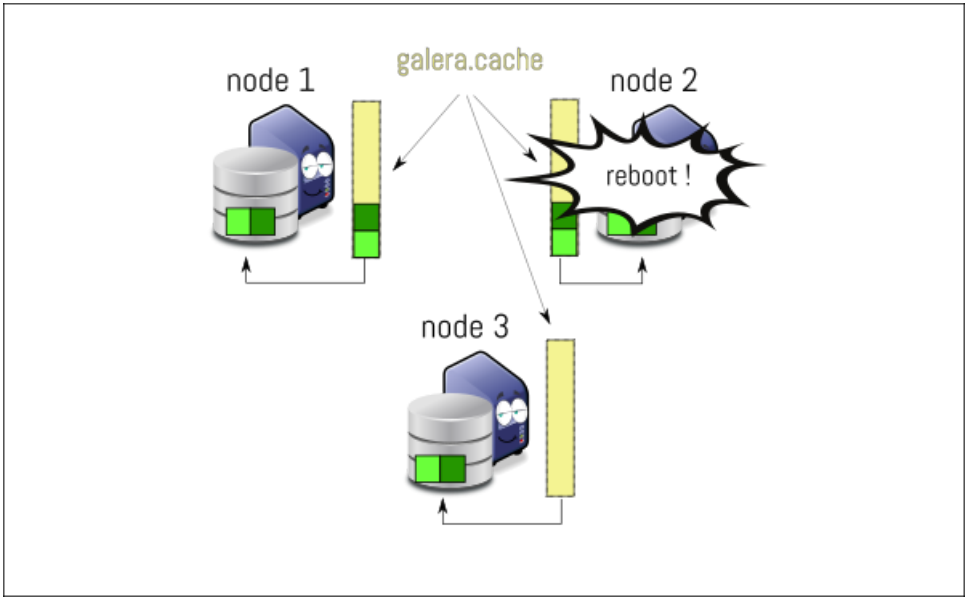
Galera Cache & IST



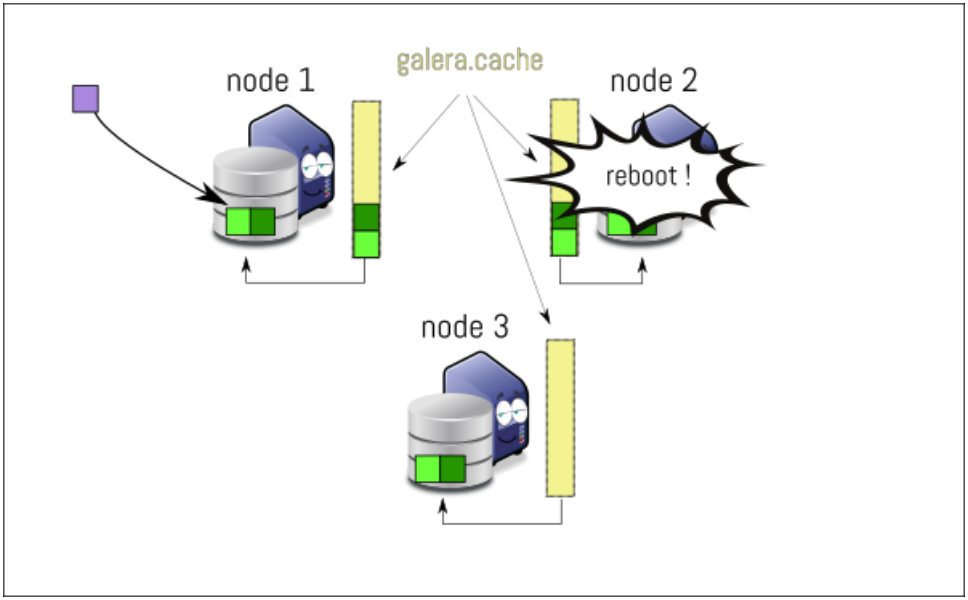
Galera Cache & IST



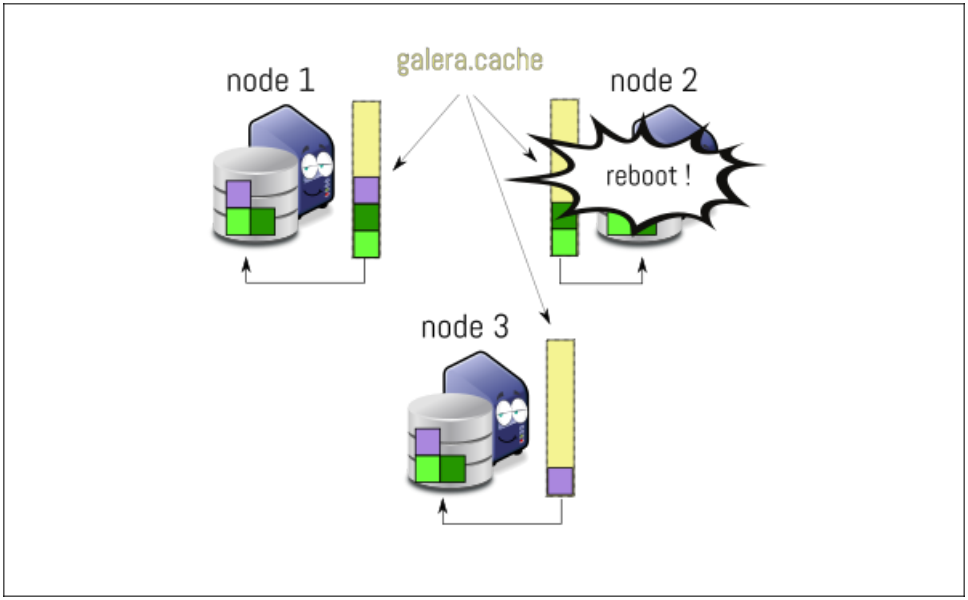
Galera Cache & IST



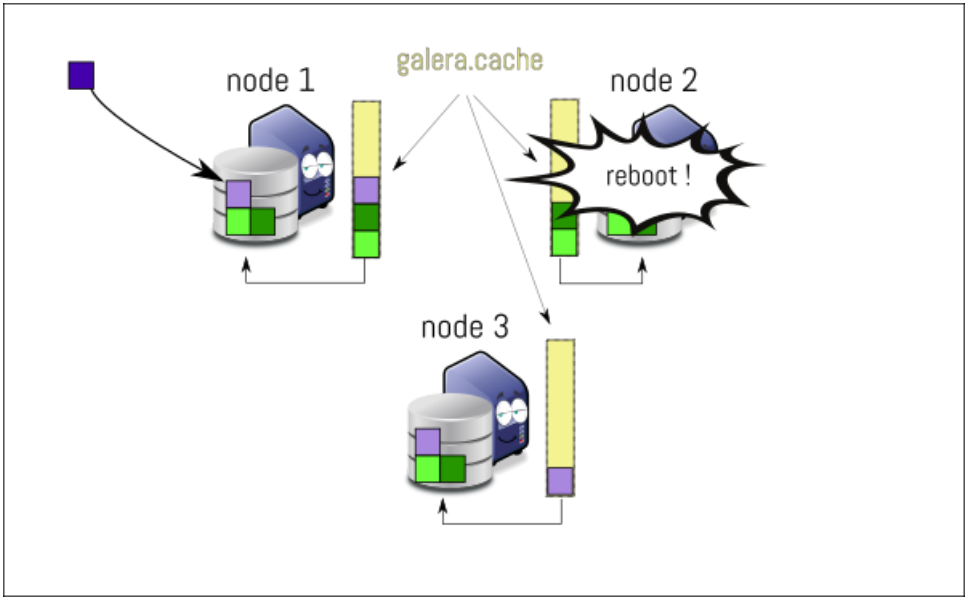
Galera Cache & IST



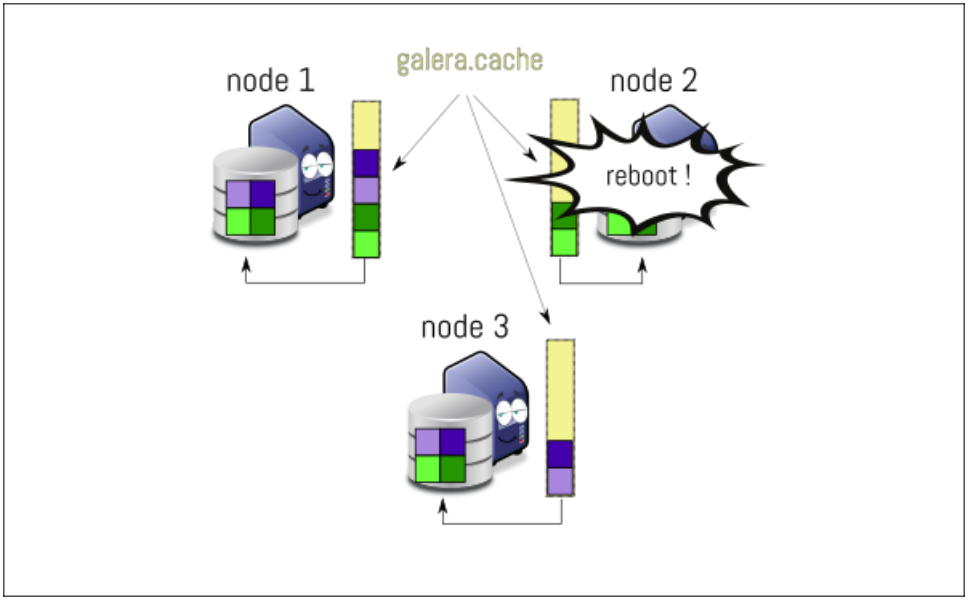
Galera Cache & IST



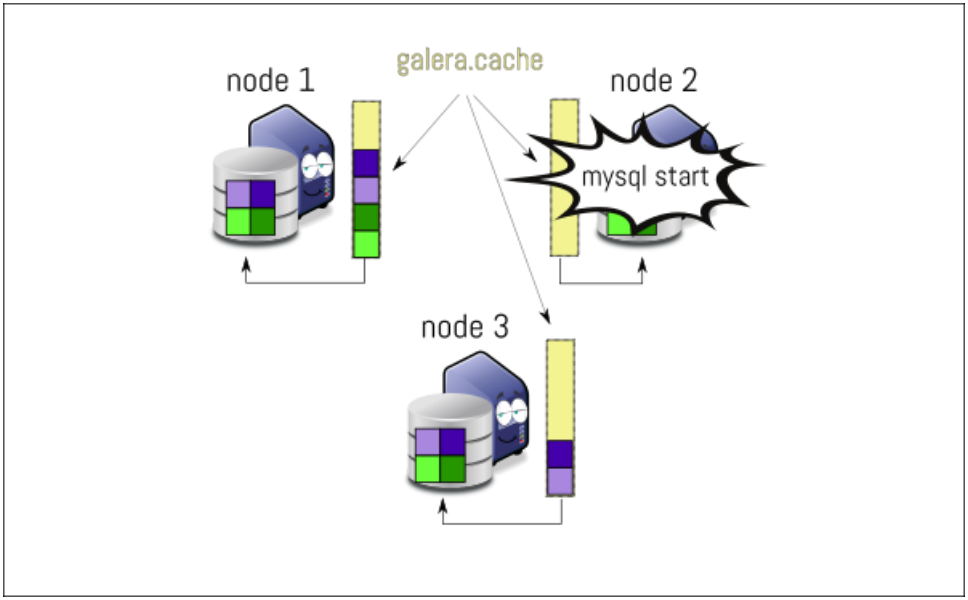
Galera Cache & IST



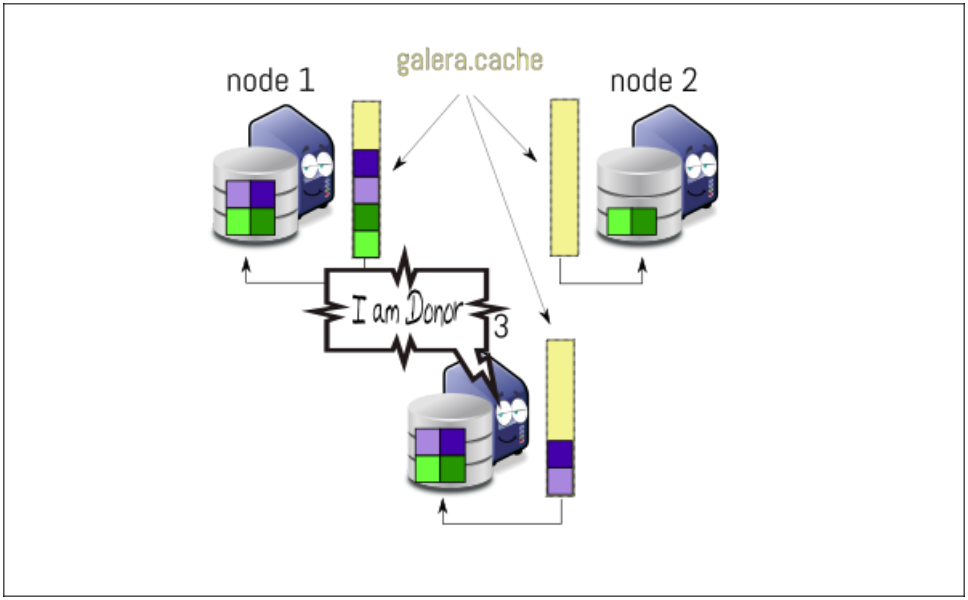
Galera Cache & IST



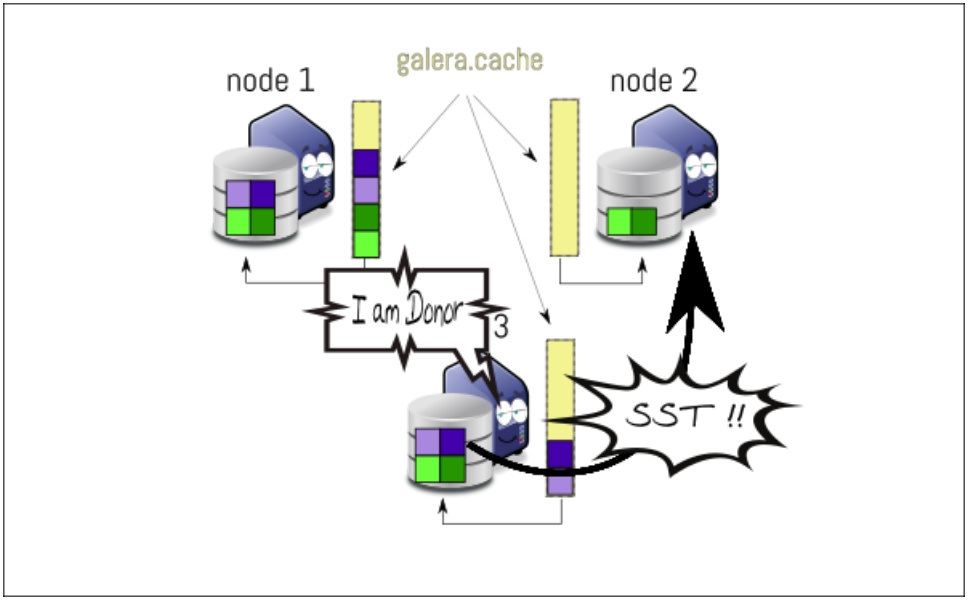
Galera Cache & IST



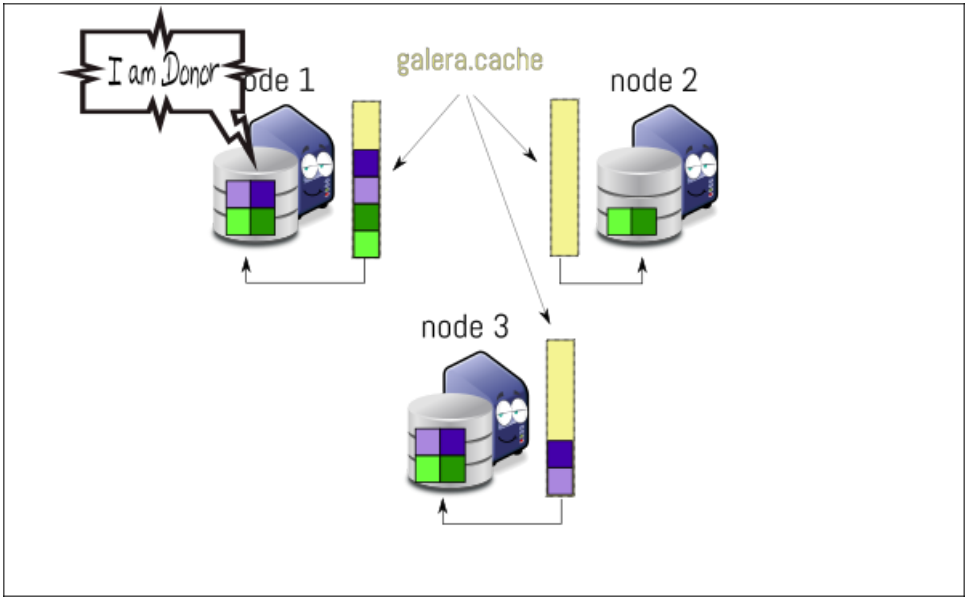
Galera Cache & IST



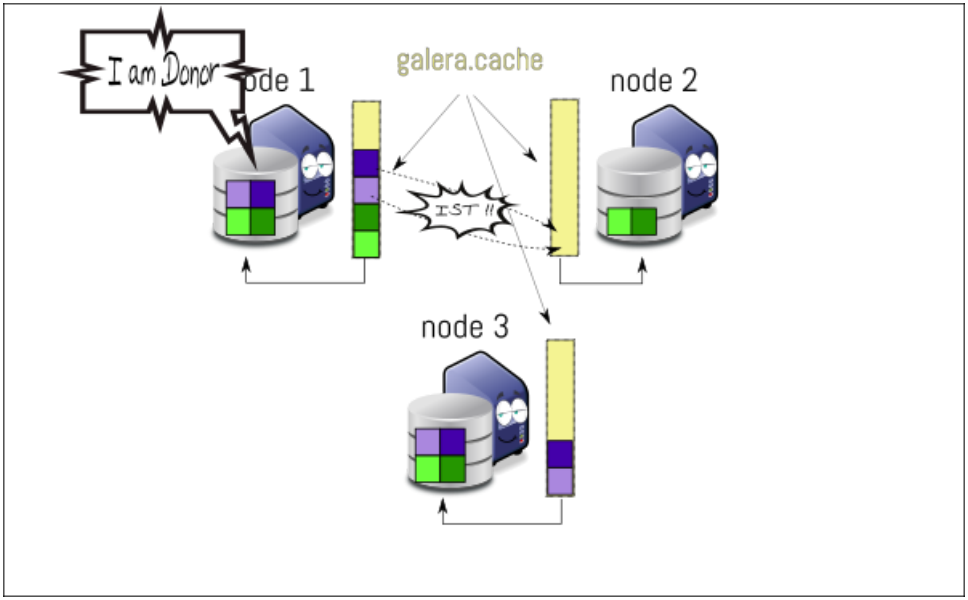
Galera Cache & IST



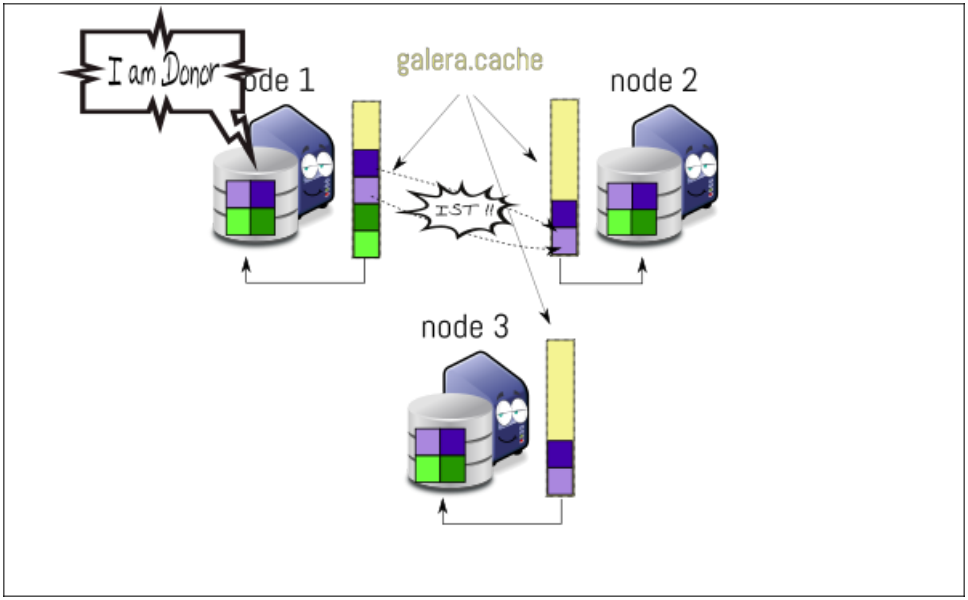
Galera Cache & IST



Galera Cache & IST



Galera Cache & IST



Thank you !