MySQL Group Replication

An Overview

Safe Harbor Statement

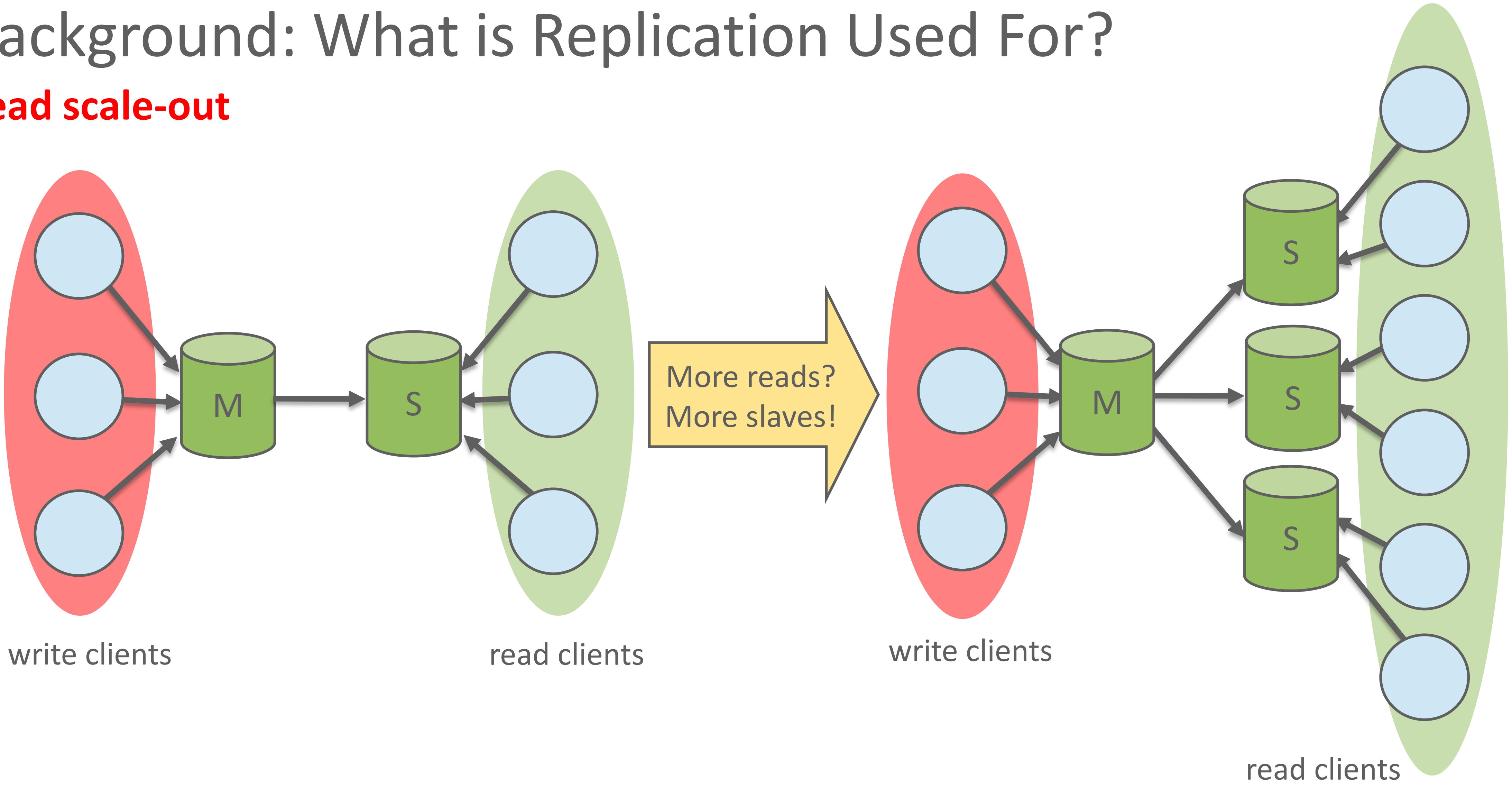
Program Agenda

Program Agenda

1 The Theory

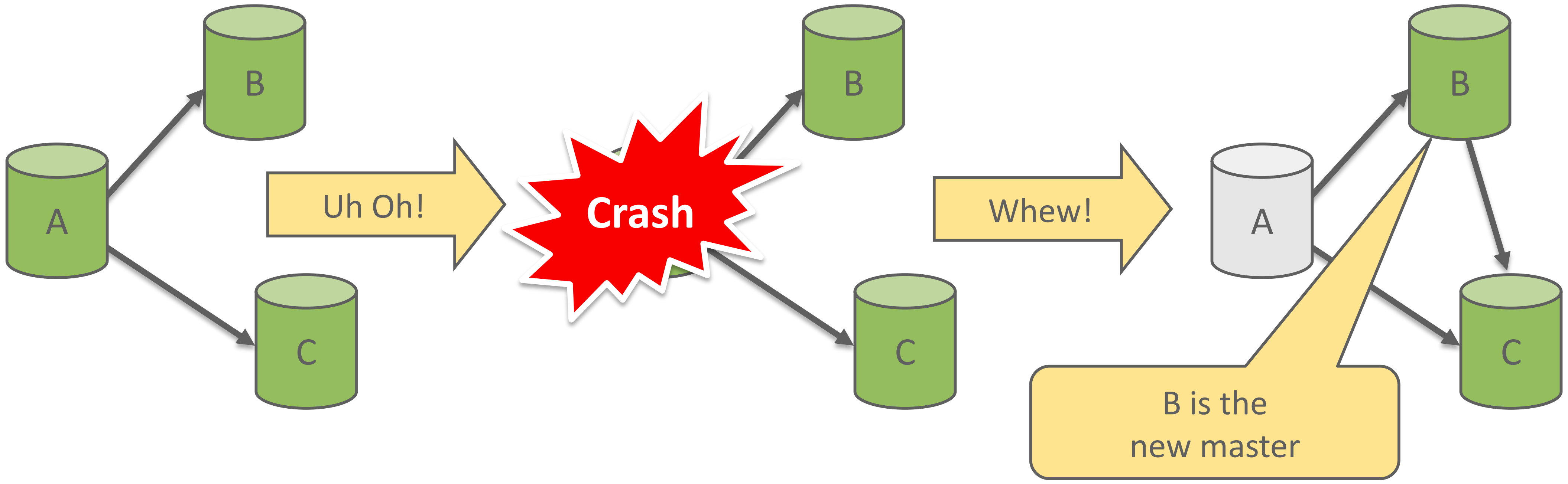
Background: What is Replication Used For?

Read scale-out

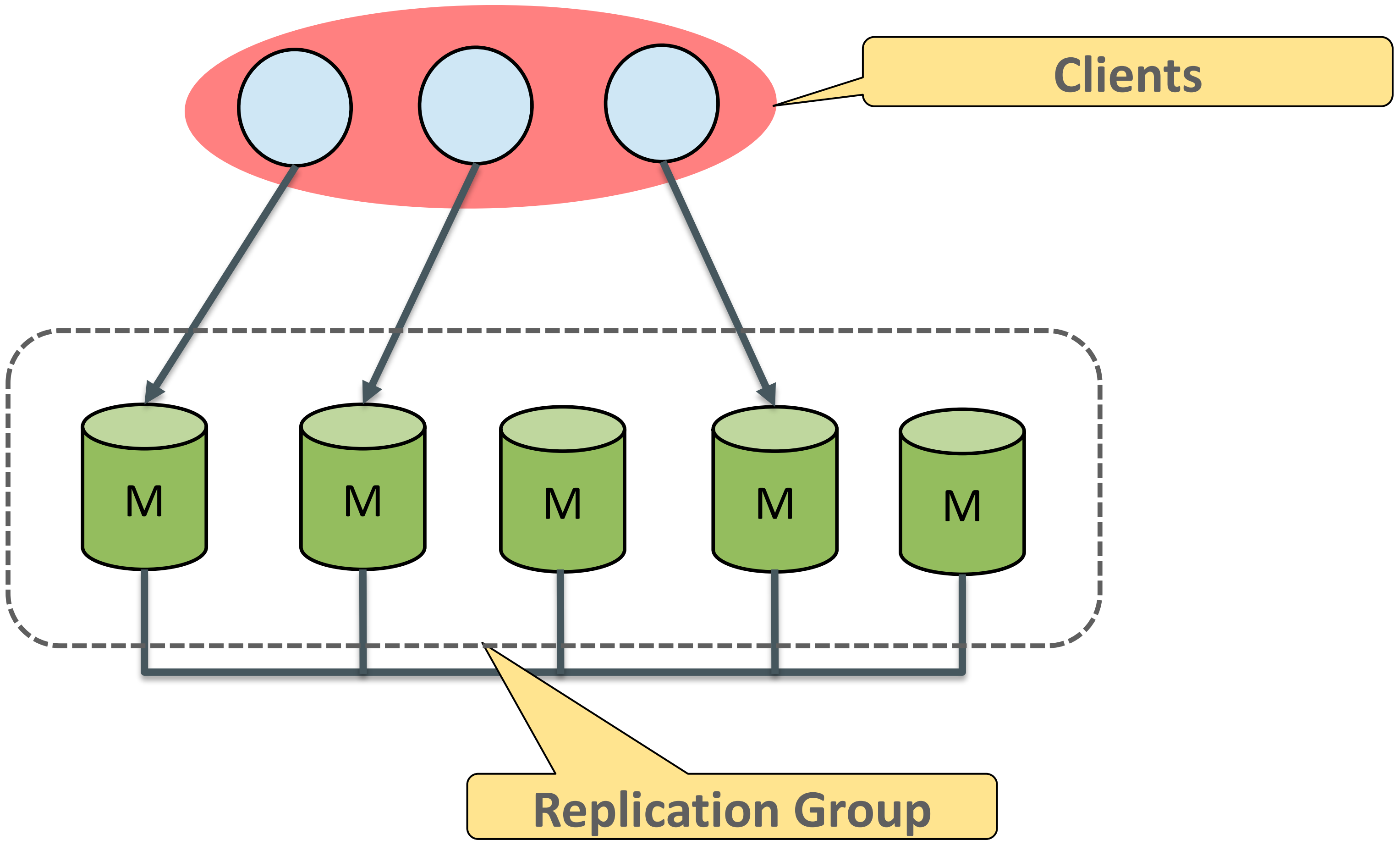


Background: What is Replication Used For?

Redundancy: If master crashes, promote slave to master



MySQL Group Replication



MySQL Group Replication

- **What is MySQL Group Replication?**

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

- **What does the MySQL Group Replication plugin do for the user?**
  - Removes the need for manually handling server fail-over
  - Provides distributed fault tolerance
  - Enables Active/Active update anywhere setups
  - Automates group reconfiguration (handling of crashes, failures, re-connects)
  - Provides a highly available distributed database service

Use Cases

- **Elastic Replication**
  - Environments that require a very fluid replication infrastructure, where the number of servers has to grow or shrink dynamically and with as little manual intervention as possible.
- **Highly Available Shards**
  - Sharding is a popular approach to achieve write scale-out. Users can use MySQL Group Replication to implement highly available shards. Each shard can map to an individual Replication Group.
- **Alternative to Master-Slave Replication**
  - It may be that a single master server makes it a single point of contention. Writing to an entire group may prove more scalable under certain circumstances.

The Theory Behind It

- Implementation is based on “Replicated Database State Machines”
  - Group Communication primitives resemble general properties of Databases
  - Distributed systems meet Databases: Pedone, Guerraoui, and Schiper paper
- Deferred update replication: before committing locally we certify it on all nodes
  - In order to implement it one needs Atomic Broadcast – the change occurs everywhere or nowhere
  - This is necessary to ensure data consistency across all nodes
- Membership Service
  - Group Membership: it allows one to know that at a given moment in time all the members that are participating in the protocol are associated with the same logical identifier (view id)
  - View Synchrony: ensure that messages from past views are all delivered before a new view is installed

2 How It Works

Example: Adding a Node to an Existing Group

- Create a user for automated distributed recovery:

```
./bin/mysql -uroot -h 127.0.0.1 -P 13001 -p --prompt='server1>'

server1>
CREATE USER 'rpl_user'@'%' IDENTIFIED BY 'rpl_pass';
GRANT REPLICATION SLAVE ON *.* TO rpl_user@'%';
```

- Server needs to be started with a valid configuration:

```
./bin/mysqld --no-defaults --basedir=. --datadir=<DATADIR_LOCATION> -P <PORT> \
--socket=mysqld<ID>.sock --log-bin=master-bin --server-id=<ID>          \
--gtid-mode=on --enforce-gtid-consistency --log-slave-updates    \
--binlog-checksum=NONE --binlog-format=row                       \
--master-info-repository=TABLE --relay-log-info-repository=TABLE \
--transaction-write-set-extraction=MURMUR32                      \
--plugin-dir=lib/plugin --plugin-load=group_replication.so
```

Example: Adding a Node to an Existing Group (2)

- Now lets configure the replication user used for recovery:

```
SET GLOBAL group_replication_recovery_user='rpl_user';
SET GLOBAL group_replication_recovery_password='rpl_pass';
```

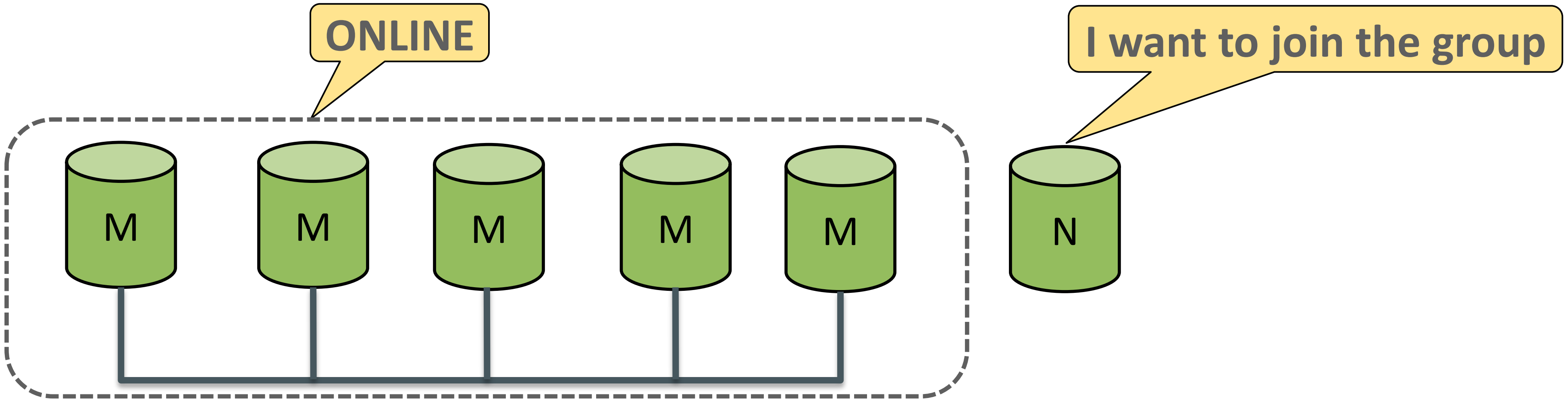
- Now the group communication backbone:

```
SET GLOBAL group_replication_group_name= <valid UUID>;
SET GLOBAL group_replication_local_address=<this node address:port for the
communication backbone>;
SET GLOBAL group_replication_peer_addresses= <comma-separated list of all other
nodes in the group>;
SET GLOBAL group_replication_bootstrap_group= 0;
```

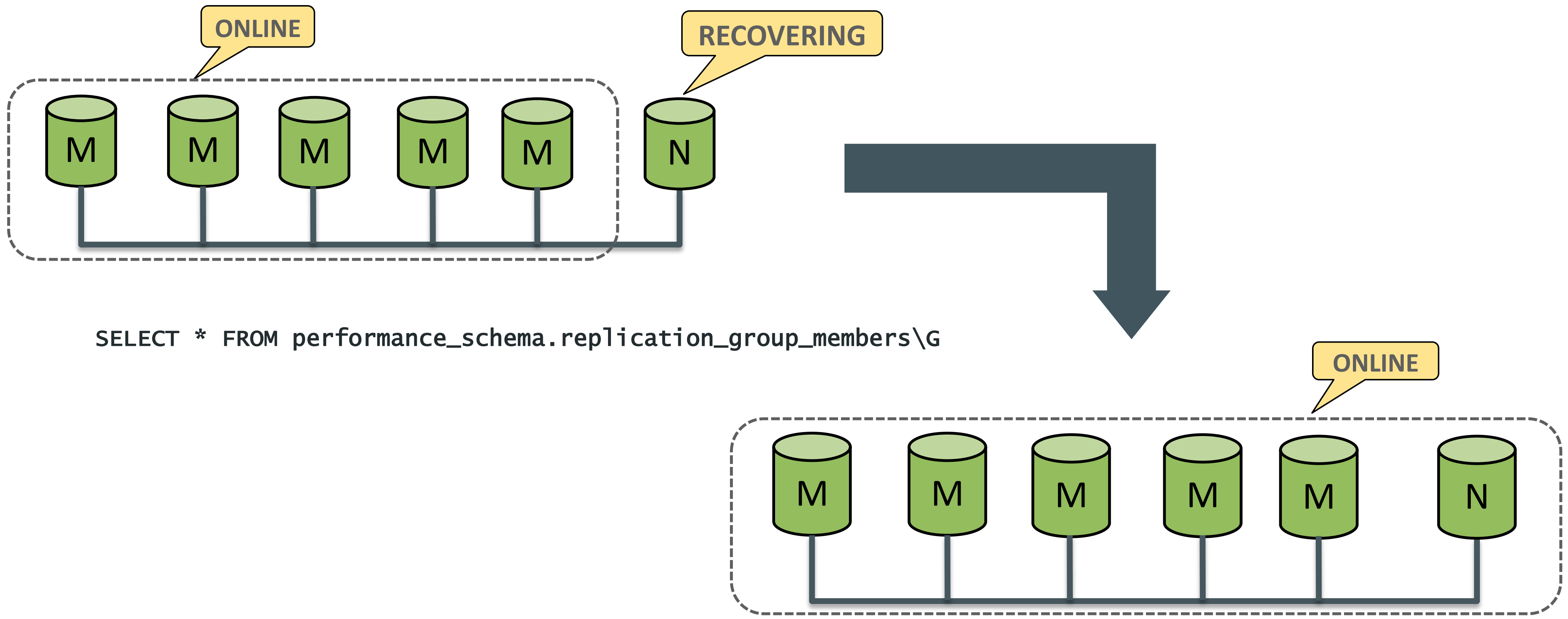
Example: Adding a Node to an Existing Group (3)

- Server that joins the group will automatically synchronize with the others
  - It will retrieve the diff between its data and the rest of the group
  - Hint: provision the new node with base data (i.e. restore a backup) before joining an existing group

START GROUP REPLICATION;



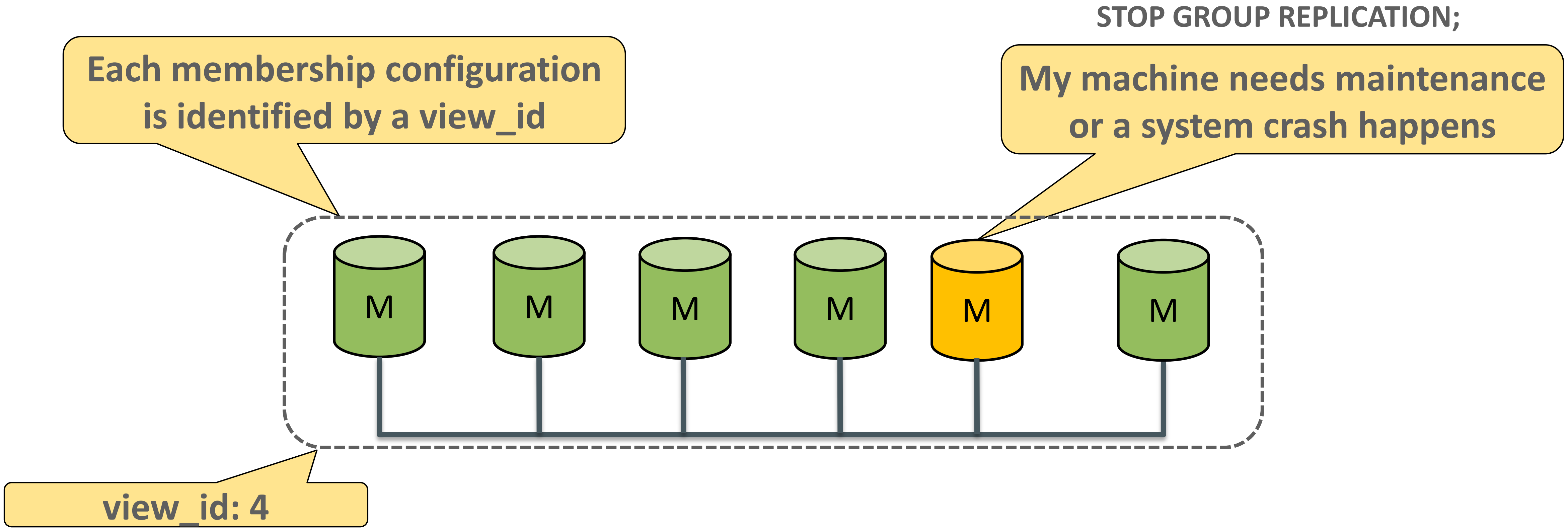
Example: Adding a Node to an Existing Group (4)



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

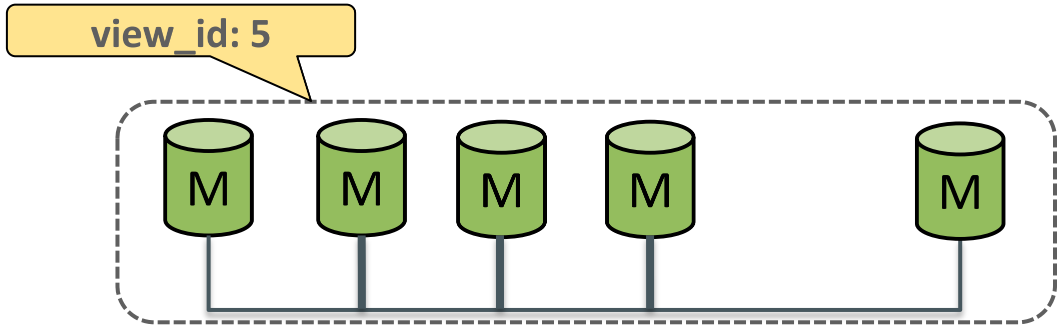
Example: Automated Group Membership

- If a server leaves the group, the others will automatically be informed

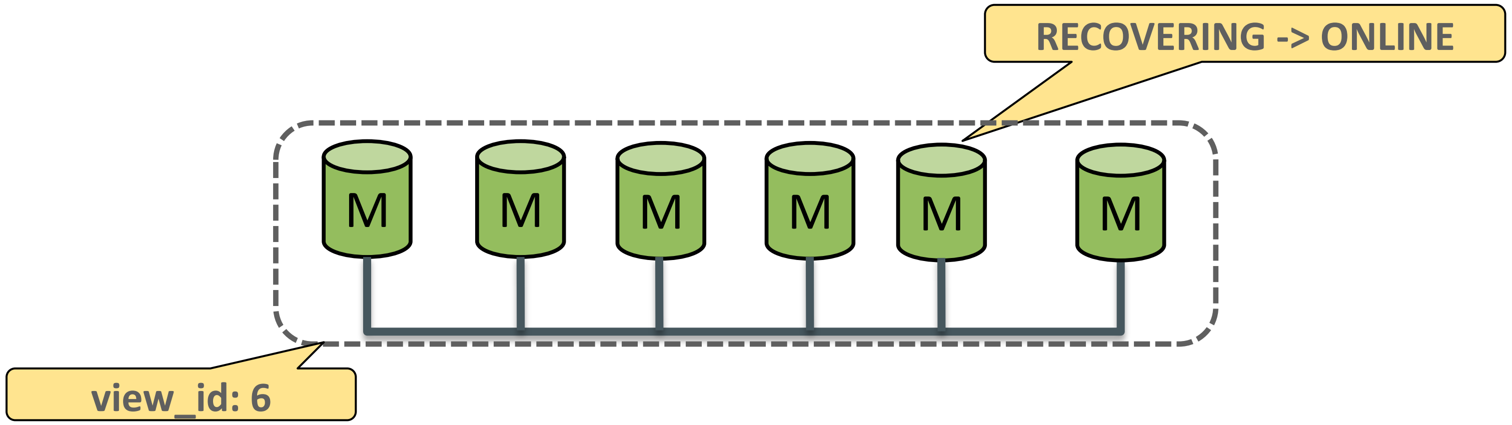

STOP GROUP REPLICATION;
Each membership configuration is identified by a view_id
My machine needs maintenance or a system crash happens
M
M
M
M
M
M
view_id: 4

Example: Automated Group Membership (2)

- If a server leaves the group, the others will automatically be informed



- Server that (re)joins the group will automatically synch with the others



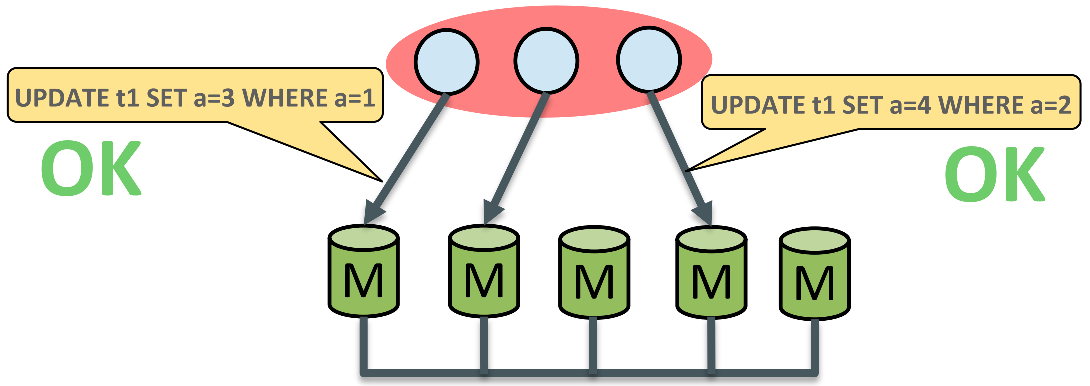
3

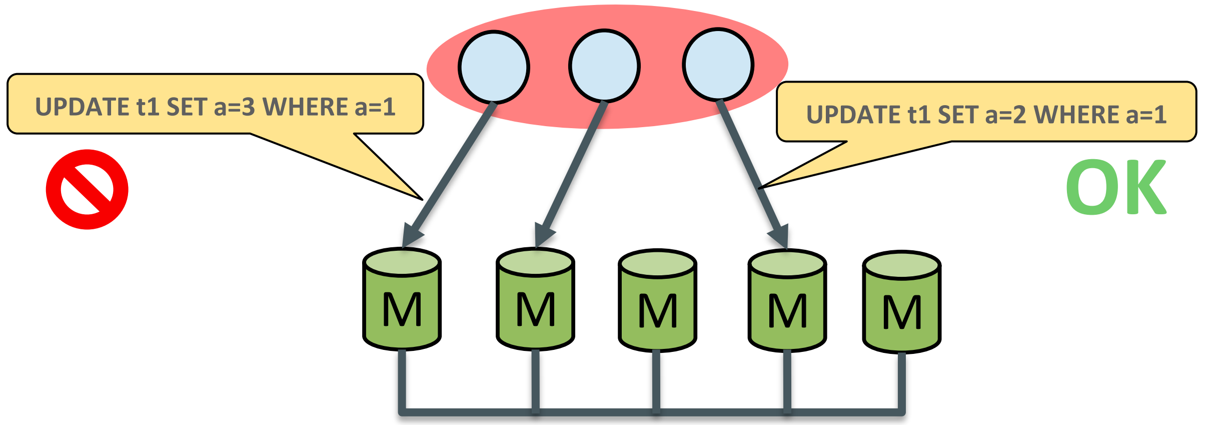
How to use

ORACLE®

Multi-Master Update Anywhere!

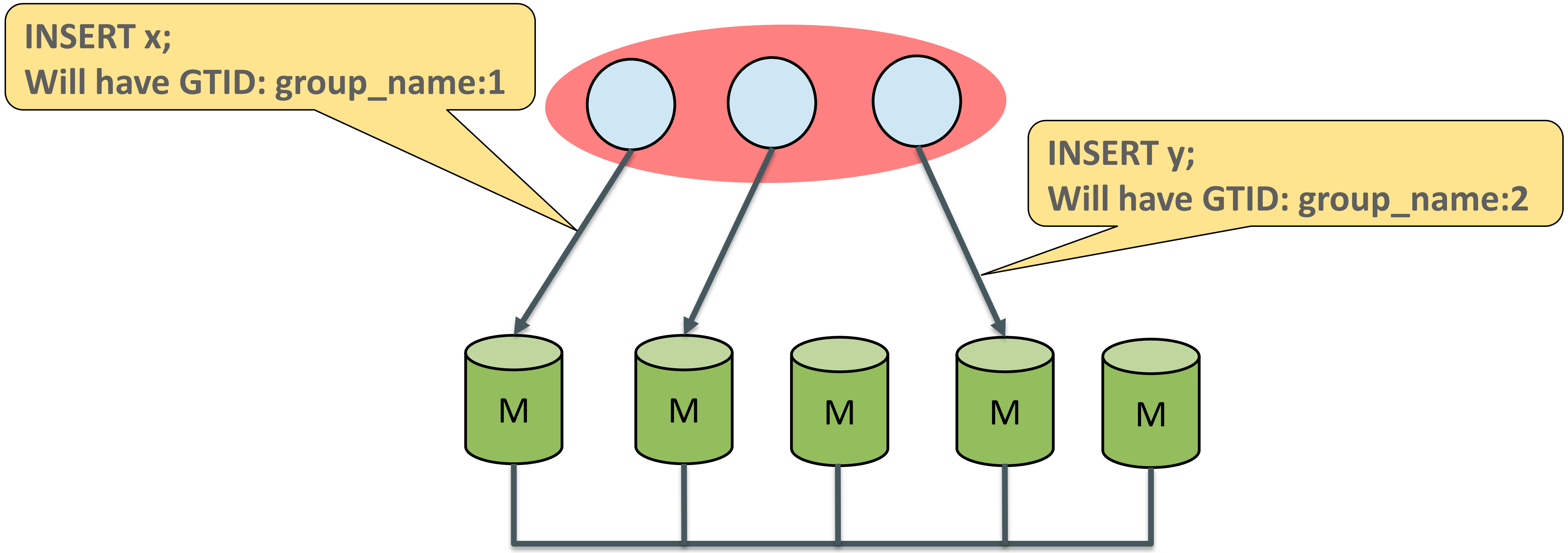
- Any two transactions on different servers can write to the same row
- Conflicts will automatically be detected and handled
  - *First committer wins rule*





Full GTID Support

- All group members share the same UUID, which is the group name



Monitoring the Replication Group

- Monitor group health and stats using Performance Schema tables

```
mysql> SELECT * FROM
performance_schema.replication_connection_status\G
*************************** 1. row ***************************
 CHANNEL_NAME: group_replication_applier
 GROUP_NAME: 8a94f357-aab4-11df-86ab-c80aa9429563
 SOURCE_UUID: 8a94f357-aab4-11df-86ab-c80aa9429563
 THREAD_ID: NULL
 SERVICE_STATE: ON
 ...
```

```
mysql> SELECT * FROM performance_schema.replication_group_member_stats\G
*************************** 1. row ***************************
 CHANNEL_NAME: group_replication_applier
 VIEW_ID: 1428497631:3
 MEMBER_ID: e38fdea8-dded-11e4-b211-e8b1fc3848de
 COUNT_TRANSACTIONS_IN_QUEUE: 0
 COUNT_TRANSACTIONS_CHECKED: 12
 COUNT_CONFLICTS_DETECTED: 5
 COUNT_TRANSACTIONS_VALIDATING: 6
 TRANSACTIONS_COMMITTED_ALL_MEMBERS: 8a84f397-aaa4-18df-89ab-
c70aa9823561:1-7
 LAST_CONFLICT_FREE_TRANSACTION: 8a84f397-aaa4-18df-89ab-c70aa9823561:7
```

```
mysql> SELECT * FROM performance_schema.replication_group_members\G
*************************** 1. row ***************************
 CHANNEL_NAME: group_replication_applier
 MEMBER_ID: 597dbb72-3e2c-11e4-9d9d-ecf4bb227f3b
 MEMBER_HOST: nightfury
 MEMBER_PORT: 13000
 MEMBER_STATE: ONLINE
*************************** 2. row ***************************
 ...
```

4 Conclusion

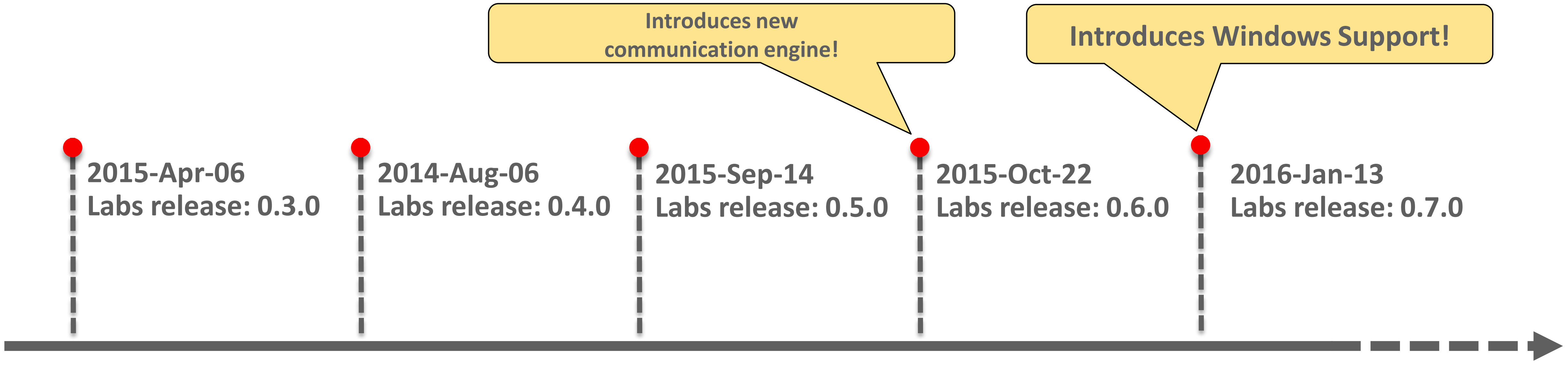
ORACLE

Summary

- **Cloud Friendly**
  - Great techonology for deployments where high availability and elasticity is a requirement, such as Cloud based infrastructures
- **Integrated**
  - With standard MySQL servers through a well defined API
  - With standard GTIDs, ROW based replication, and Performance Schema tables
- **Autonomic and Operations Friendly**
  - It is self-healing: no admin overhead for handling server fail-overs
  - Provides fault-tolerance: enables multi-master update anywhere and a resilient distributed MySQL service
- Lab releases provide a sneak peek at what is coming -- a new replication plugin and exciting new infrastructure: **MySQL Group Replication and MySQL Router**

Releases

- Strong development cycles and continuous community engagement through regular lab releases


Introduces new communication engine!
Introduces Windows Support!
2015-Apr-06 Labs release: 0.3.0
2014-Aug-06 Labs release: 0.4.0
2015-Sep-14 Labs release: 0.5.0
2015-Oct-22 Labs release: 0.6.0
2016-Jan-13 Labs release: 0.7.0

Where to go from here?

- Packages
  - http://labs.mysql.com

- Blogs from the Engineers (news, technical information, and much more)
  - http://mysqlhighavailability.com/tag/mysql-group-replication/