MySQL 5.7 でのレプリケーション強化点とグループ・レプリケーション

updated: 2016/05/18

日本オラクル株式会社
MySQL Global Business Unit
Yoshiaki Yamasaki / 山﨑 由章

Safe Harbor Statement

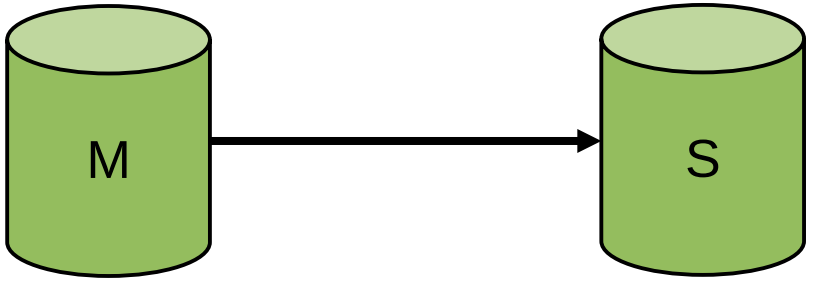
以下の事項は、弊社の一般的な製品の方向性に関する概要を説明するものです。
また、情報提供を唯一の目的とするものであり、いかなる契約にも組み込むことはできません。以下の事項は、マテリアルやコード、機能を提供することをコミットメントするものではない為、購買決定を行う際の判断材料になさらないで下さい。

オラクル製品に関して記載されている機能の開発、リリースおよび時期については、弊社の裁量により決定されます。

MySQLのレプリケーション機能

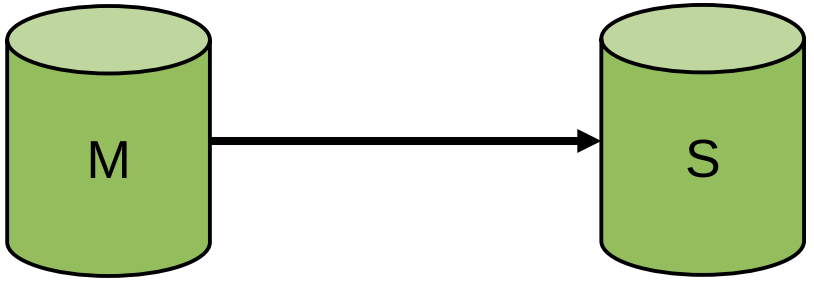
レプリケーションとは？

- データの複製(レプリカ)を別のサーバーに持てる機能
- MySQLの標準機能で、多数のWebサイト等で利用されている
  - シンプルな設定で利用可能
  - マスター→スレーブ 構成

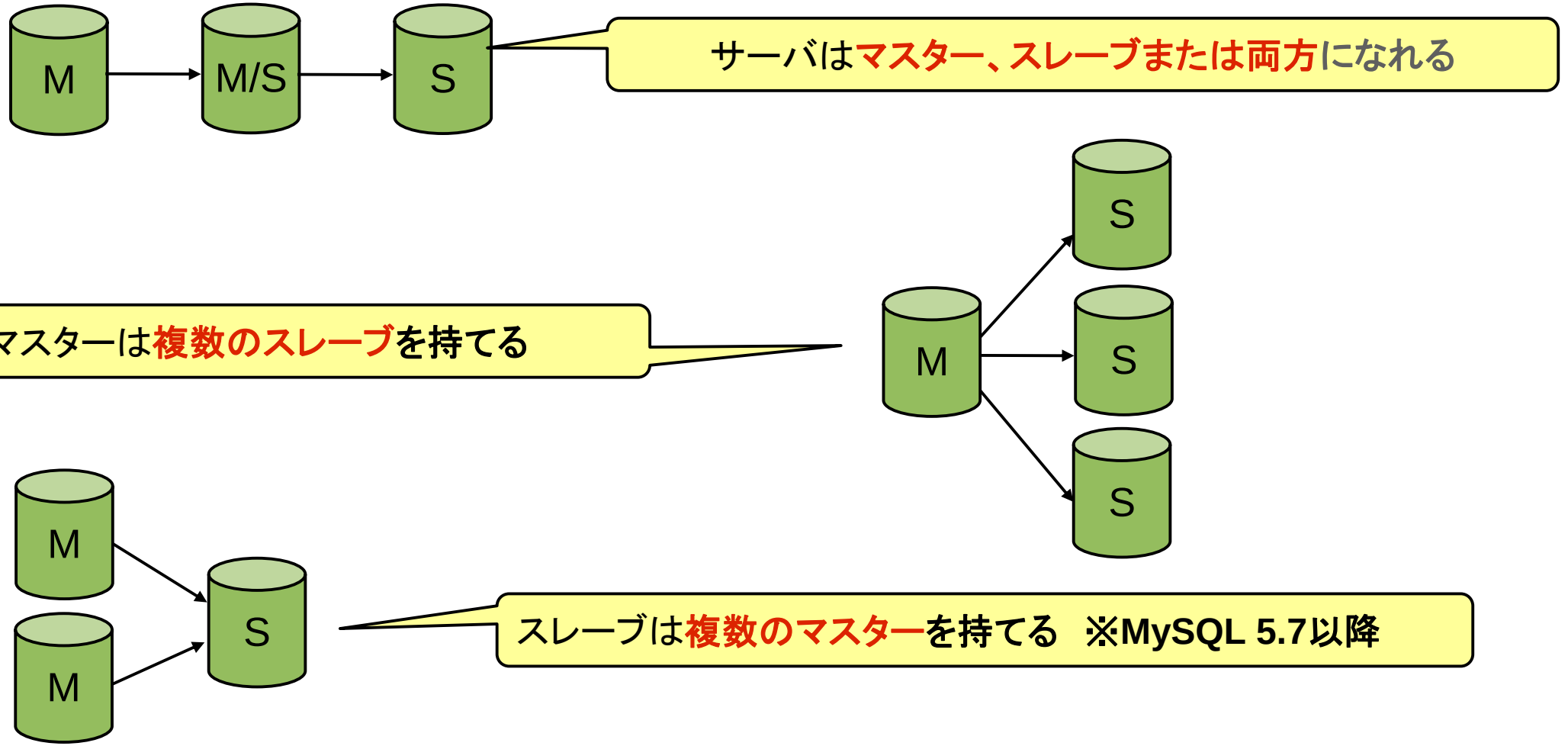


レプリケーションとは？

- マスターサーバー
  - データを変更
  - 変更内容をスレーブに転送
- スレーブサーバー
  - マスターでの変更内容を受け取る
  - 変更内容をデーターベースに反映

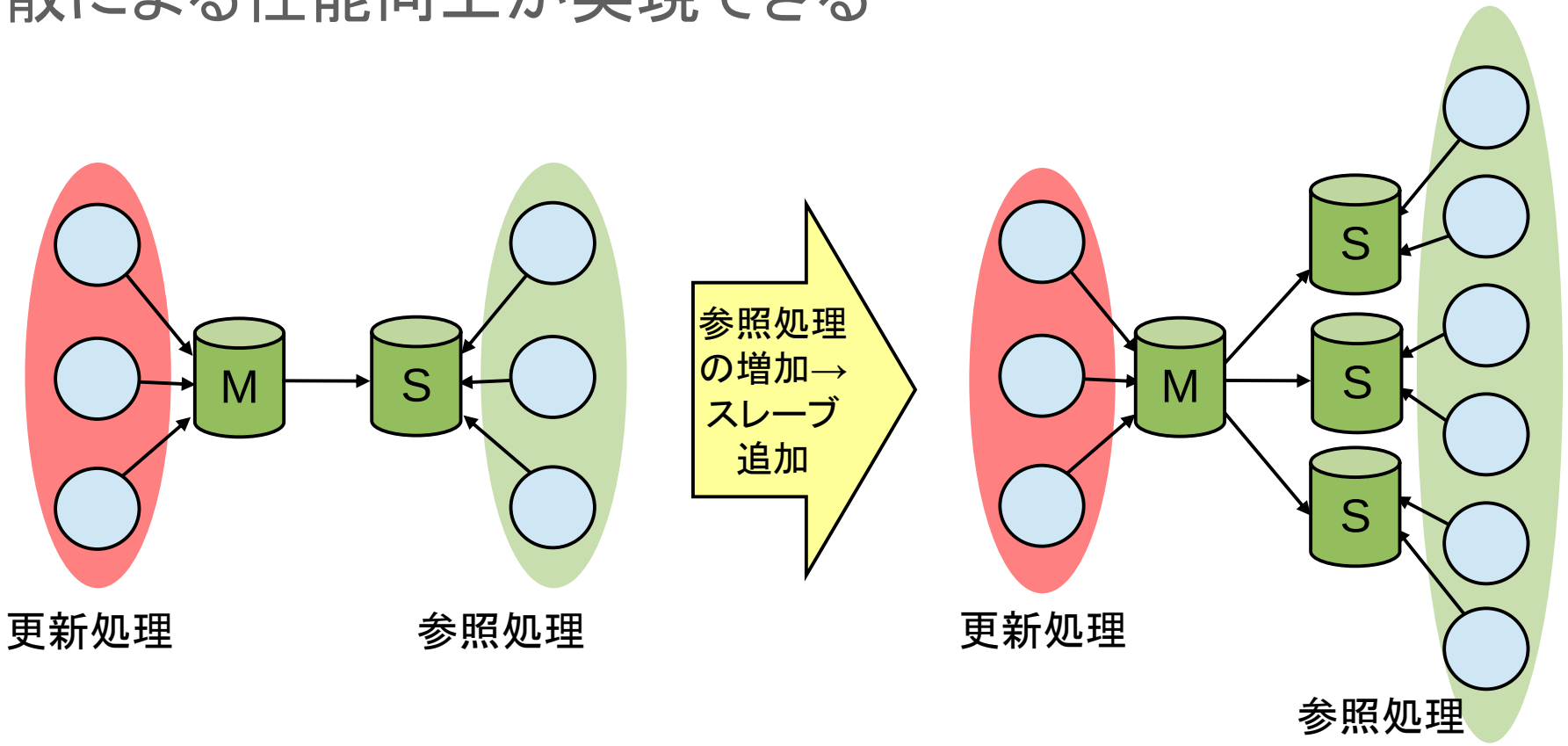


レプリケーション構成



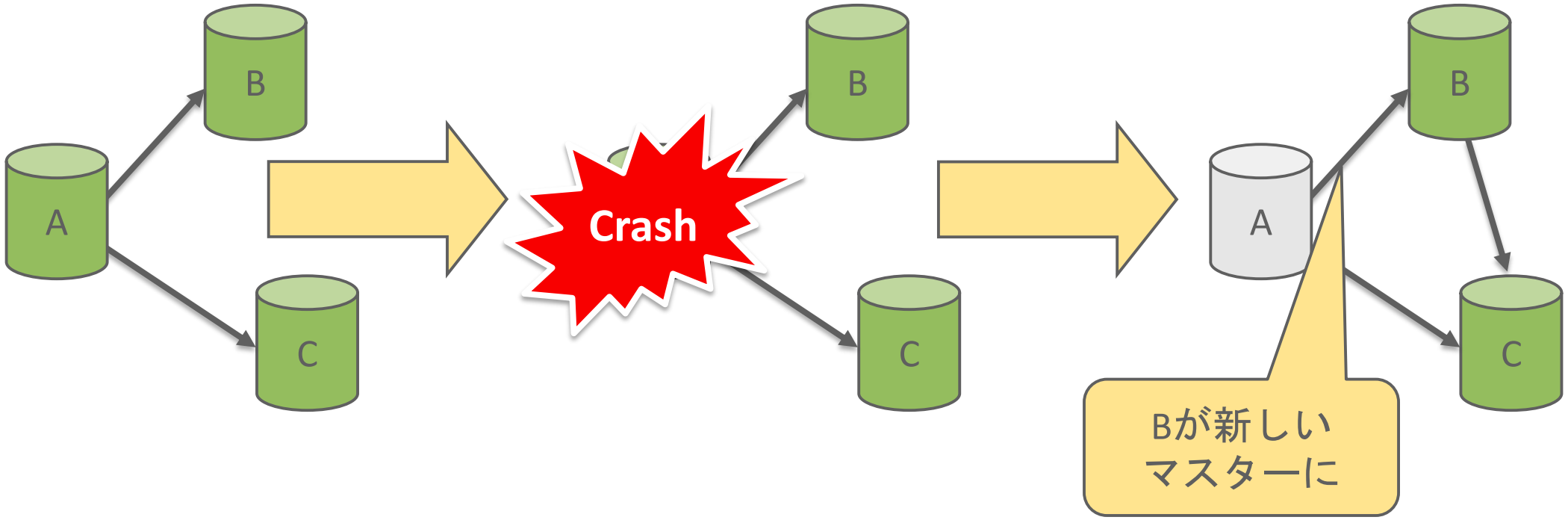
レプリケーションの利点：参照性能の向上

- 参照処理の負荷が高い場合は、スレーブサーバーを追加することで、負荷分散による性能向上が実現できる



レプリケーションの利点：高可用性構成の実現

- マスターの障害時に、スレーブをマスターに昇格することで高可用性を実現可能



レプリケーションの利点：地理的冗長性の実現

- 地理的に離れた場所に、災害対策サイトを構築可能

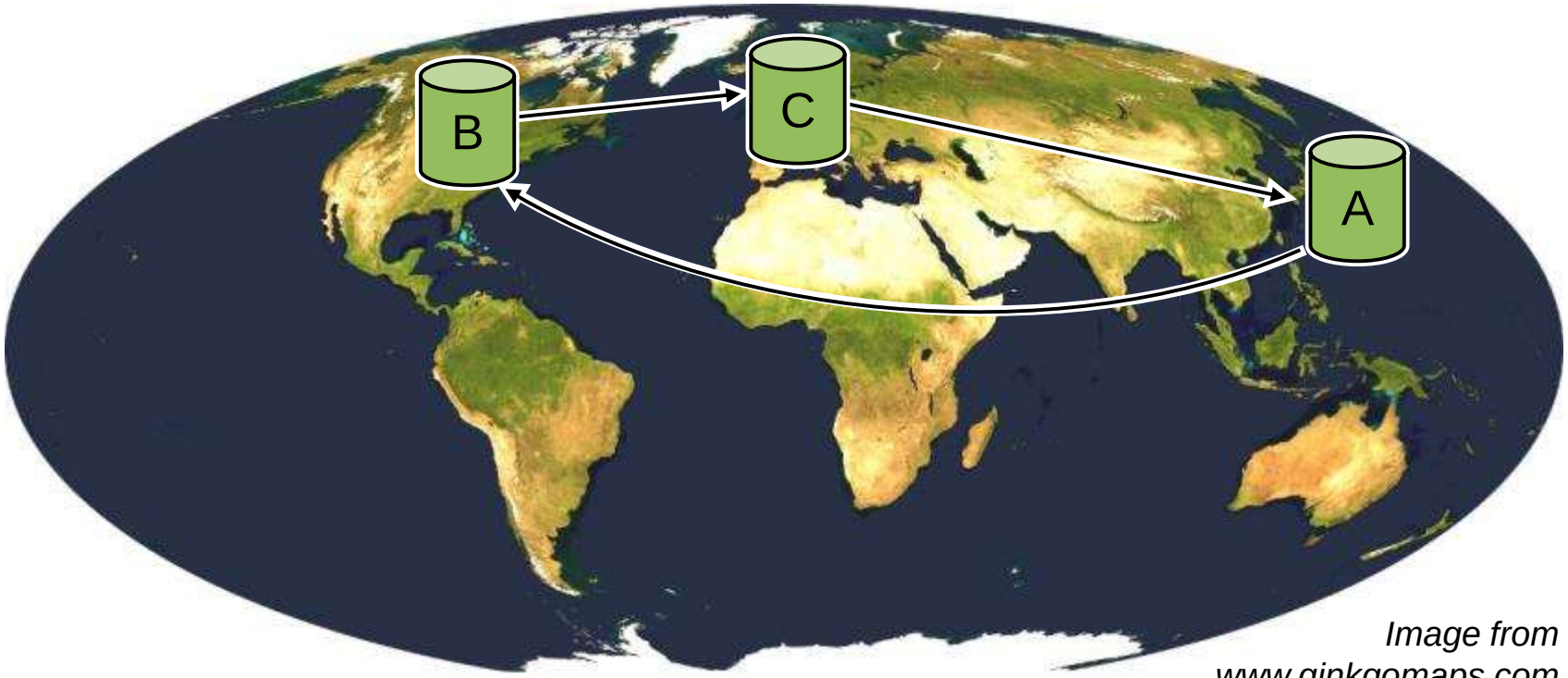
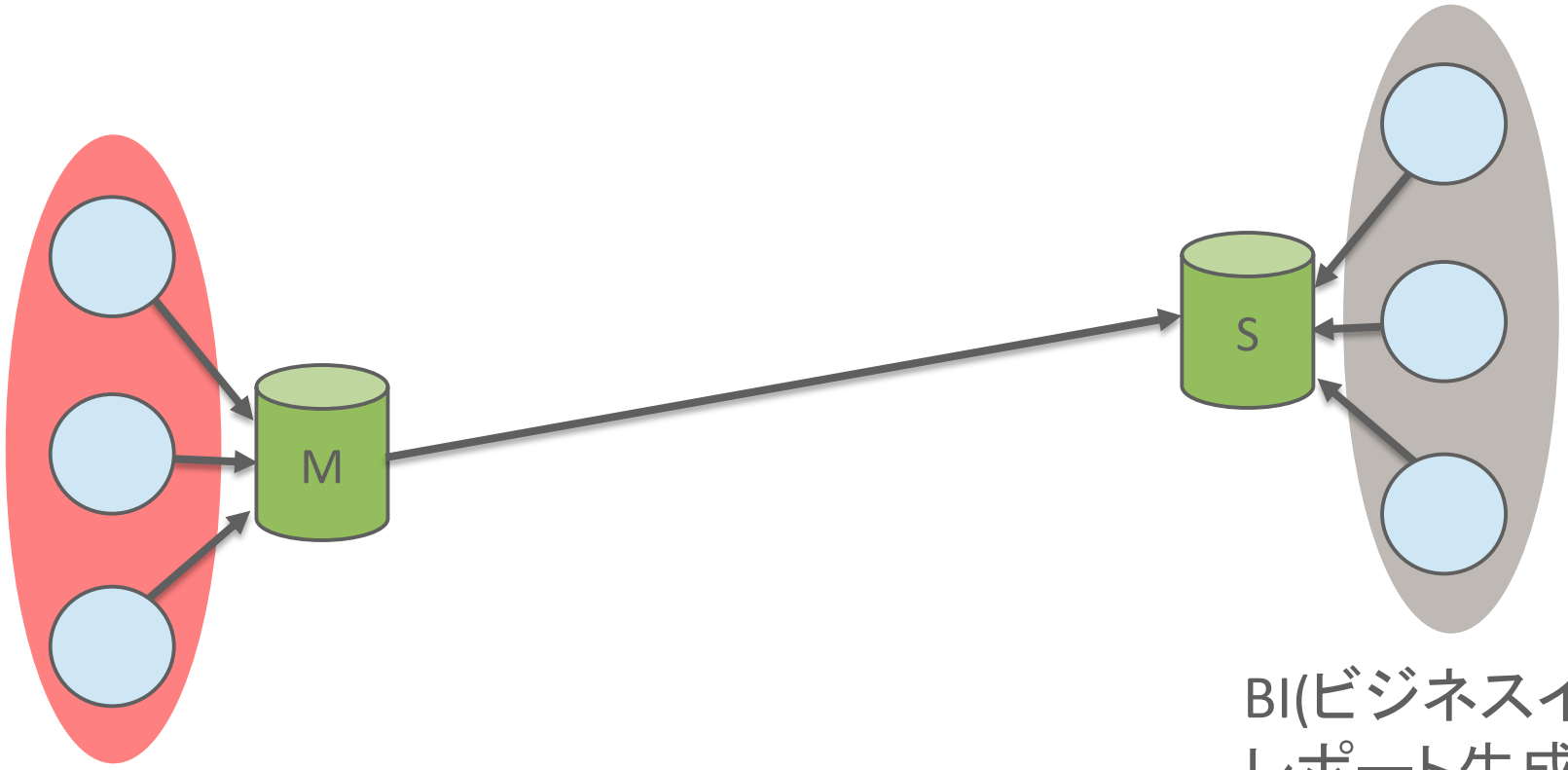


Image from www.ginkgomaps.com

レプリケーションの利点：レポート生成やバックアップ取得

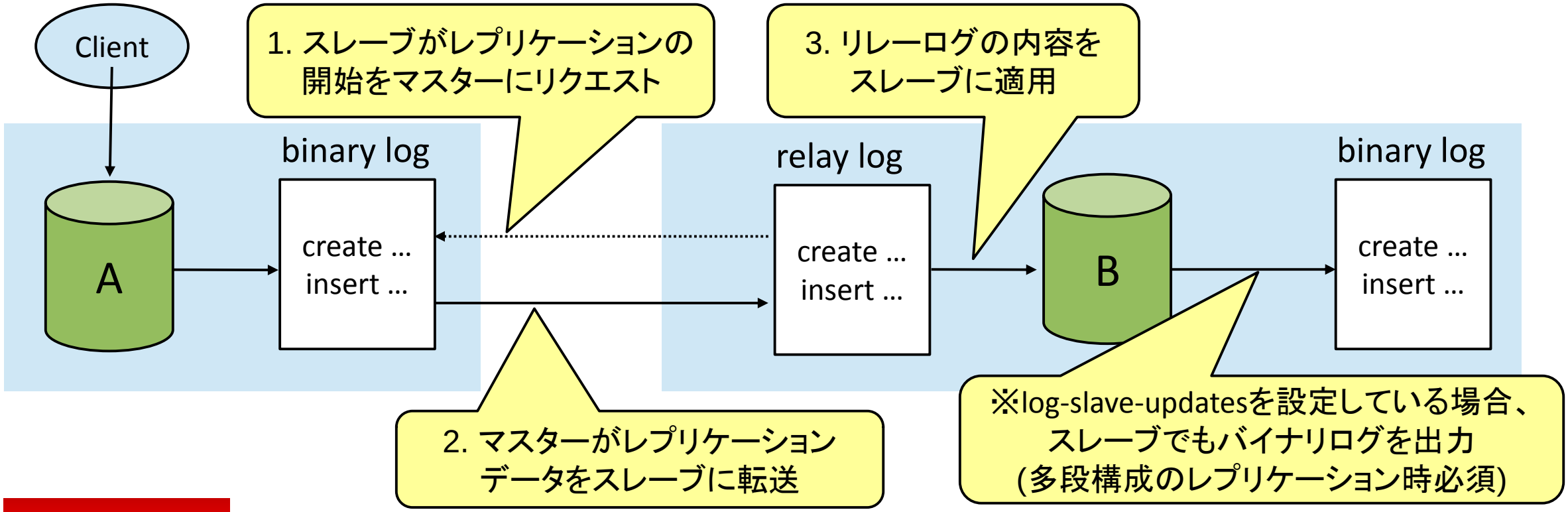


更新処理

BI(ビジネスインテリジェンス)や
レポート生成アプリ、データ分析アプリ、
バックアップ取得、など

レプリケーションの仕組み

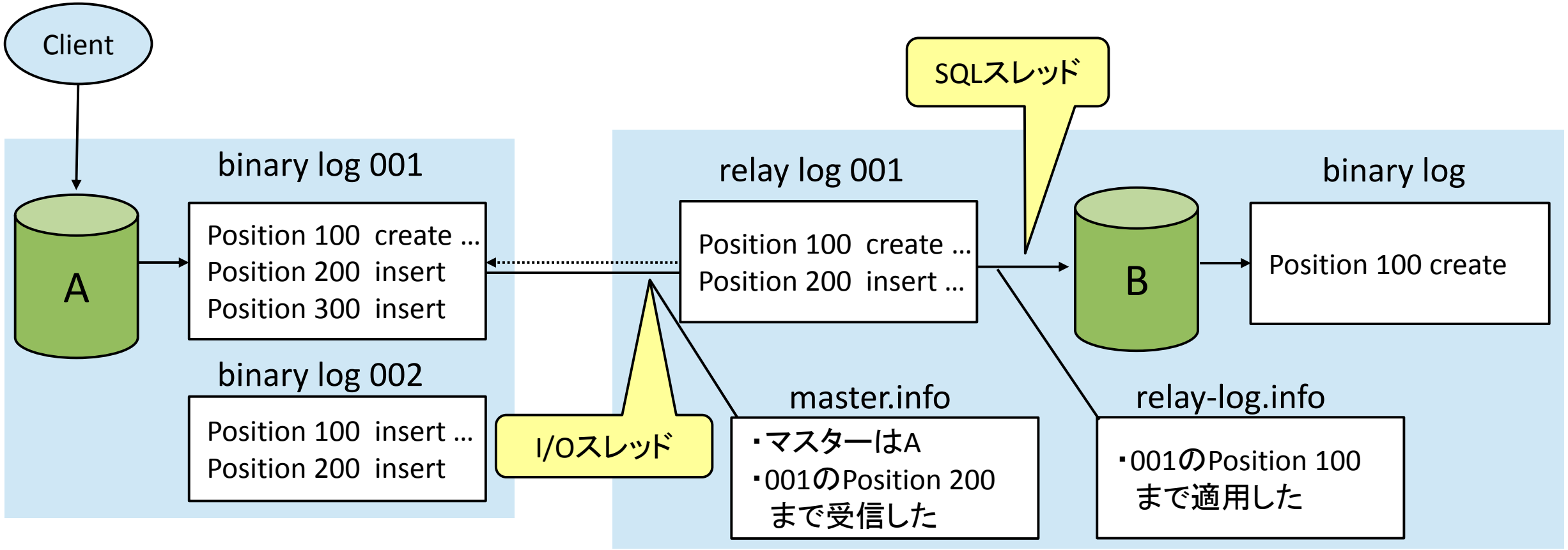
- スレーブからレプリケーション開始
- バイナリログの内容をスレーブに転送し、実行



スレーブ上に存在するファイル、スレッド

- ファイル
  - リレーログファイル：マスターから受信した変更点を記録したファイル
  - バイナリログファイル：スレーブで実行した変更点を記録したファイル
    (log-slave-updatesを有効にしている場合のみ出力)
  - master.info：マスターへの接続に必要な情報や、読み取りを開始する
    バイナリログの位置情報(バイナリログファイル名とポジション)が
    記録されているOS上のファイル。(MySQL 5.6からテーブル内に格納可能)
  - relay-log.info：リレーログをどこまで適用したかを記録しているOS上のファイル
    (MySQL 5.6からテーブル内に格納可能)
- スレッド
  - I/Oスレッド：マスターから受信したバイナリログをリレーログファイルとして保存
  - SQLスレッド：リレーログファイル内の更新内容をDBへ反映する

イメージ図



MySQL 5.7 でのレプリケーション強化点

GTID(Global Transaction Identifiers)のオンラインでの設定

- GTIDの設定変更がオンラインに
  - GTIDの設定変更(onまたはoff)の変更中も参照更新可能
- サーバ間でのデータ同期不要
- サーバ再起動不要
- レプリケーション構成の変更不要
  - 任意のレプリケーション構成で利用可能
- 運用中にGITDの利用開始または停止が可能

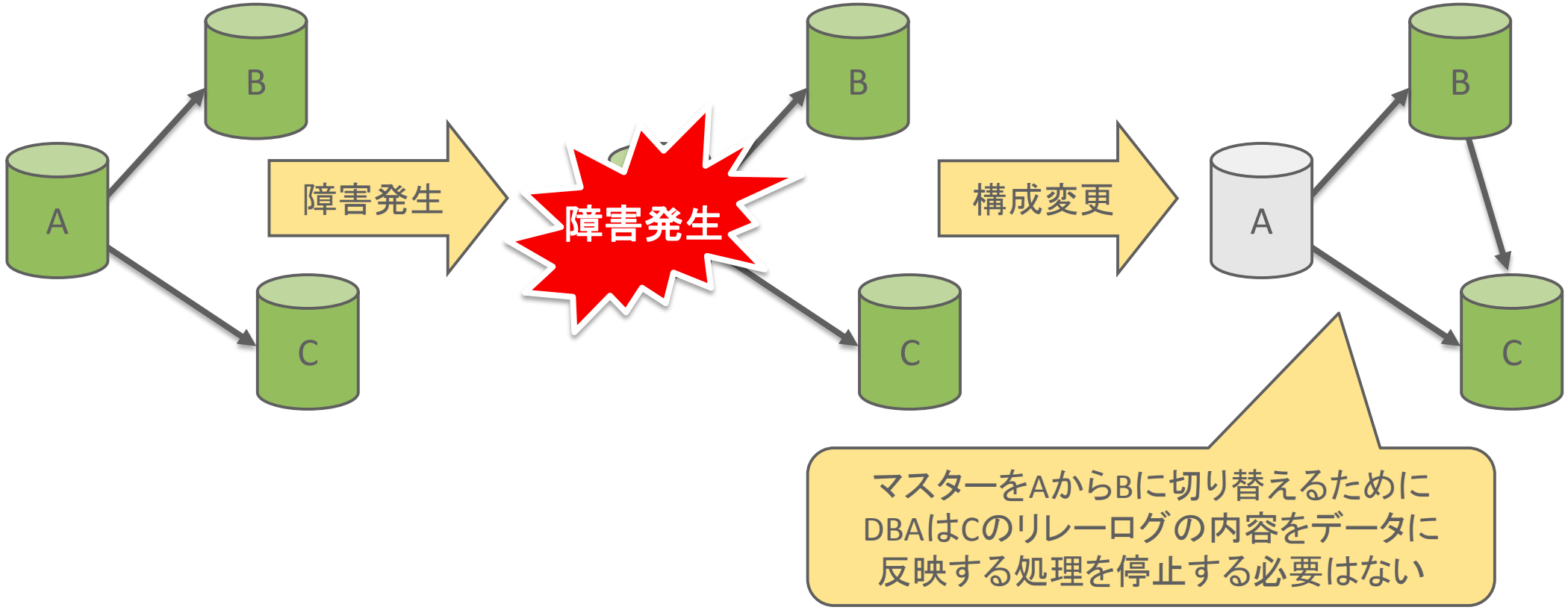
レプリケーションのフィルタをオンラインで変更

- スレーブのレプリケーションフィルタを動的に変更
  - スレーブサーバの再起動不要
  - 全てのスレーブでのフィルタをサポート
  - 各種の文字コードによる値の設定が可能

```
mysql> CHANGE REPLICATION FILTER REPLICATE_DO_DB= (db1, db2)
```

新マスターへの切り替え/フェールオーバーをオンラインで

マスターをAからBへの切り替える際にSQLスレッドの停止不要

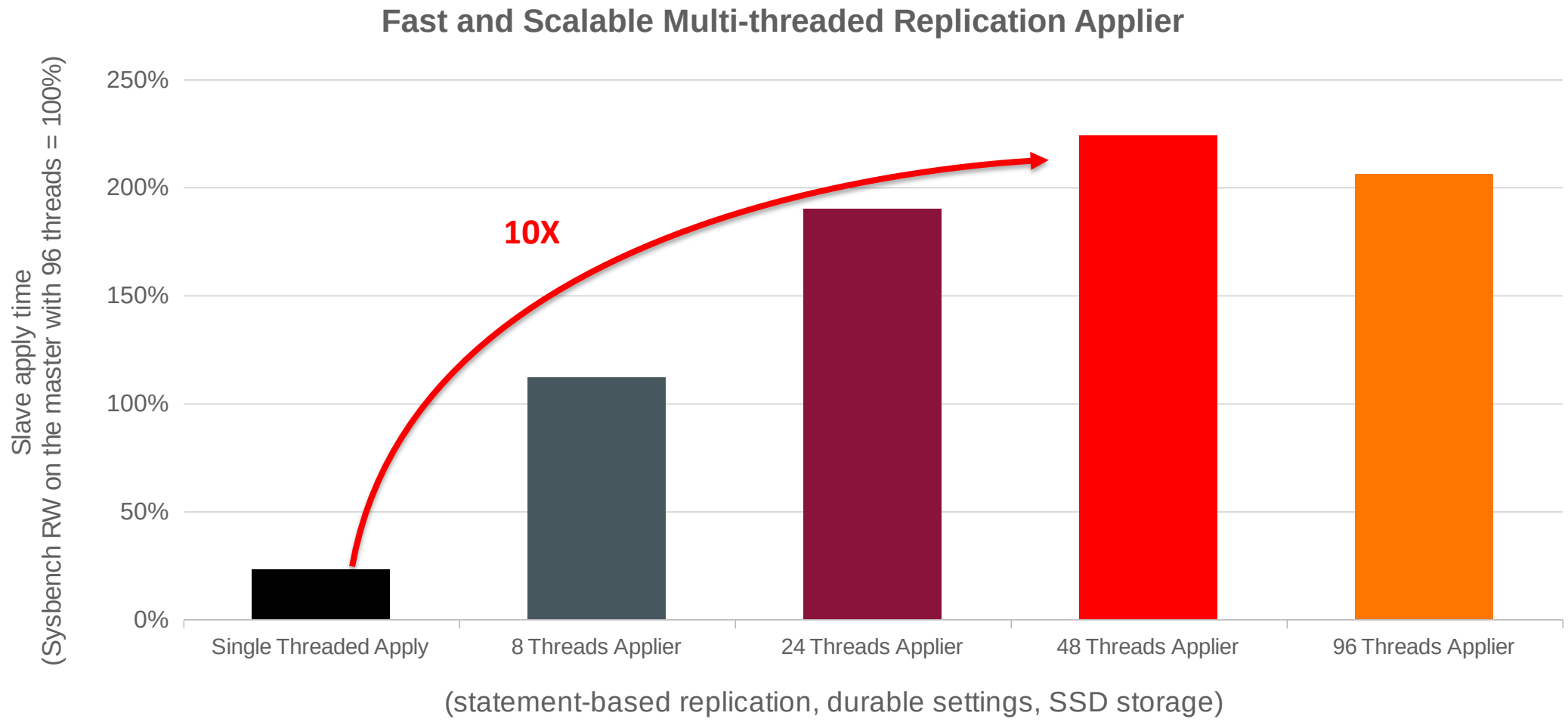


レプリケーション監視の改善

Performance SchemaのReplication関連テーブル

- SQL文にて監視
- 意味の異なる情報は別々の場所に格納
- 拡張可能, 新しい機能との連携
- より正確で一貫性のある識別子の名称
- マスター/スレーブ、マルチソース、グループレプリケーション対応

SQLスレッドの性能向上 – ロックベースの並列処理



SQLスレッドの性能向上 – ロックベースの並列処理

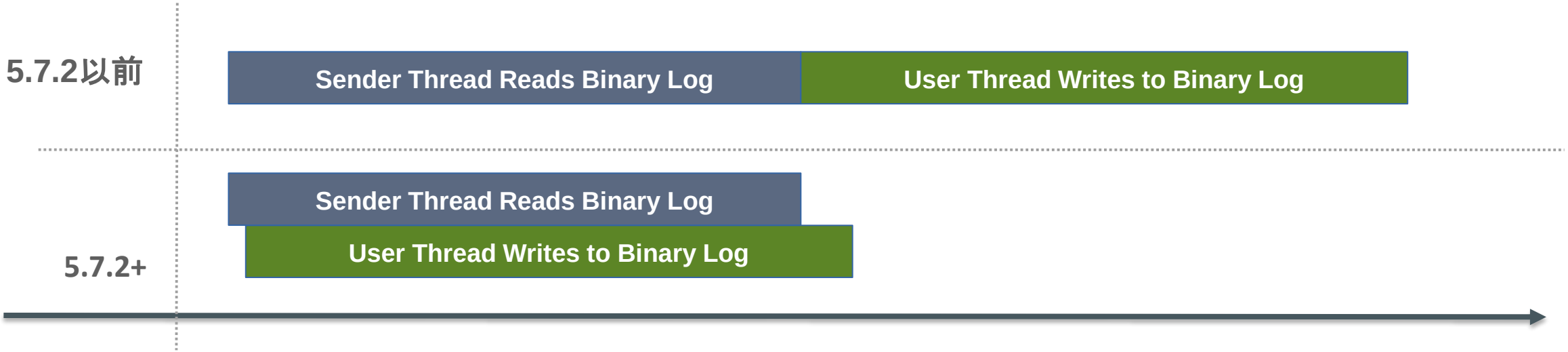
- 文(STATEMENT)ベースおよび行(ROW)ベースの両フォーマット対応
- スケジュールのポリシーは下記コマンドで制御:

```
mysql> SET slave_parallel_type= [logical_clock|database]
```

  - `logical_clock` – ロックのタイムスタンプを通じてスケジューリング
  - `database` – 5.6と同様のスケジューリング (データベースが異なれば並列実行)
- スレーブの性能拡張性は引き続き改良！

ユーザスレッドとSenderスレッドの同期の改善

- バイナリログに対するユーザスレッドでの書き込みとSenderスレッドでの読み込みを並列化
  - Senderスレッドによるユーザセッション処理のブロックを最小化
  - ユーザセッションとSenderスレッドのスループットを向上



Senderスレッドの高速化

- アロケートされた送信バッファを毎回解放しないように変更
- より大きな送信バッファが必要な場合は、その際に拡張
- 拡張された送信バッファが使われなくなると動的に縮小
- MySQL 5.7.2でのSenderスレッドの改良:
  - マスターの性能拡張性の向上;
  - リソース消費の削減 (CPU);
  - 負荷のピーク時のマスター特にDumpスレッドの処理負荷の軽減

準同期レプリケーションの高速化 – ACK Receiverスレッド

- 連続したトランザクションがスレーブからのACKをお互いに待たない
  - トランザクションt1とt2を同時にSenderスレッドがスレーブに送る
  - それぞれのACKを別のスレッドが受け取る
  - t2の準同期のレイテンシにはt1の処理によるレイテンシは含まれない
- 準同期レプリケーション有効時にスレッドが開始される

```
mysql> SET GLOBAL rpl_semi_master_enabled= ON
```

- 準同期レプリケーション無効時にスレッドが停止される

```
mysql> SET GLOBAL rpl_semi_master_enabled= OFF
```

Loss-less準同期レプリケーション

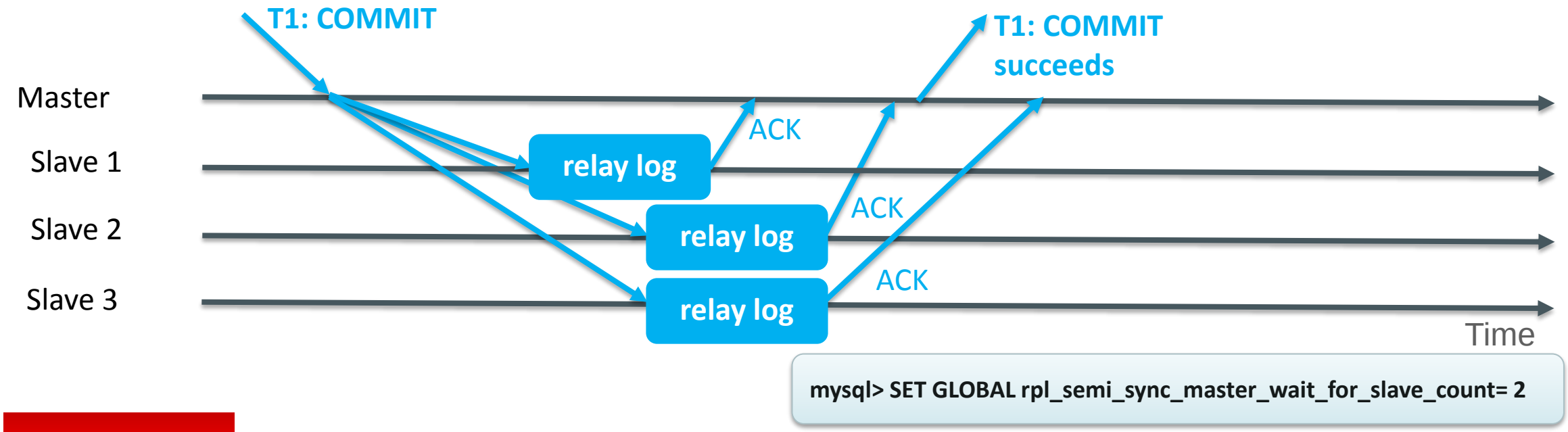
- マスターはスレーブからのACKを受け取ってからコミット
  (5.6まではコミット後にスレーブに処理を転送)
  - 他のトランザクションはACK待ちの間は該当トランザクションによる変更は見えない
- マスターに障害が発生した際でも、スレーブに転送されたトランザクションのみが他のトランザクションから見える状態
- MySQL 5.6までの挙動か新しいLoss-lessを選択可能

```
mysql> SET rpl_semi_sync_master_wait_point= [AFTER_SYNC|AFTER_COMMIT]
```

準同期レプリケーション – 複数ACKを待つ

- 指定したN台のスレーブからACKを受信するまでコミットを行わない
- 動的に設定可能:

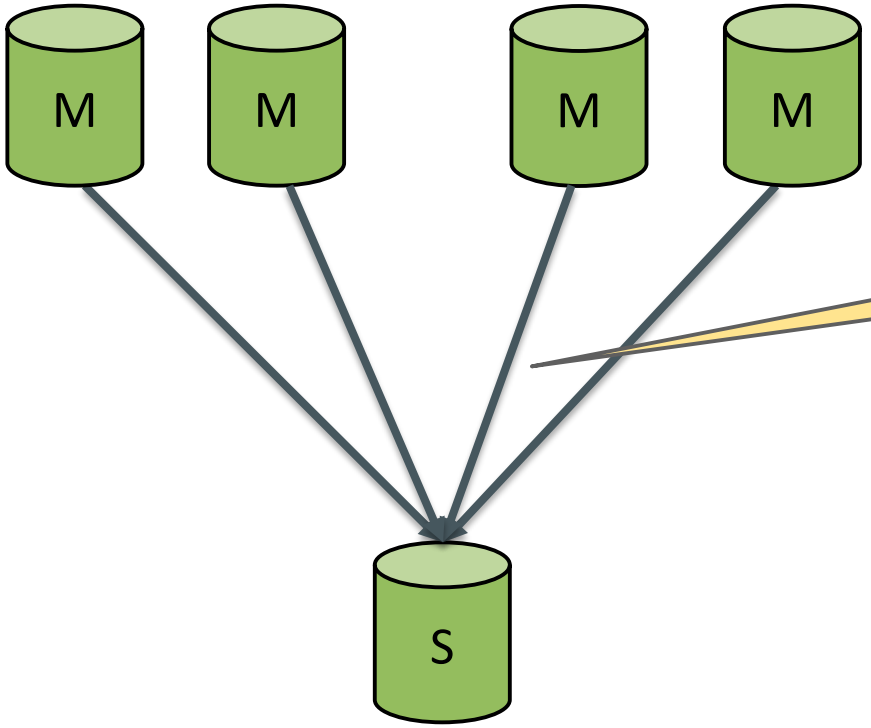
```
mysql> SET GLOBAL rpl_semi_sync_master_wait_for_slave_count= N
```



GTIDをテーブルに格納

- バイナリログを有効にしていないスレーブでもGTIDを利用可能
  - マスターに昇格する予定のないスレーブでもGTIDを利用した auto positioning(自動的にトランザクションの進捗を見つける仕組み)を利用可能

マルチソースレプリケーション



1台のスレーブが複数のマスターを持つ構成

1台のサーバにデータを集約する構成:

- バックアップを1台で実行;
- 分析用途の複雑なクエリの実行;
- クラスタ間レプリケーションのデータハブ

その他の改良点

- マルチスレッドのApplierスレッドで、失敗した**トランザクションの再試行**が可能
- マルチスレッドのApplierスレッドの**コミット順を維持**するオプション
- **mysqlbinlog**ツールに**SSL**オプション
- **mysqlbinlog**に**データベース名のリライトルール**適用可能
- 指定されたGTIDになるまでApplierスレッドの処理を一時的に停止する関数
  - **`WAIT_UNTIL_SQL_THREAD_AFTER_GTIDS(`*`gtid_set`*`[, `*`timeout`*`][,`*`channel`*`])`**
  - **`WAIT_FOR_EXECUTED_GTID_SET(`*`gtid_set`*`[, `*`timeout`*`])`**
- MySQLのクライアントサーバ間**プロトコルにGITD**の情報を返す
- バイナリログ利用時にも**XAトランザクション**をサポート
- デフォルト値の変更 例) binlog_format=**ROW**, sync_binlog=**1**

Appendix

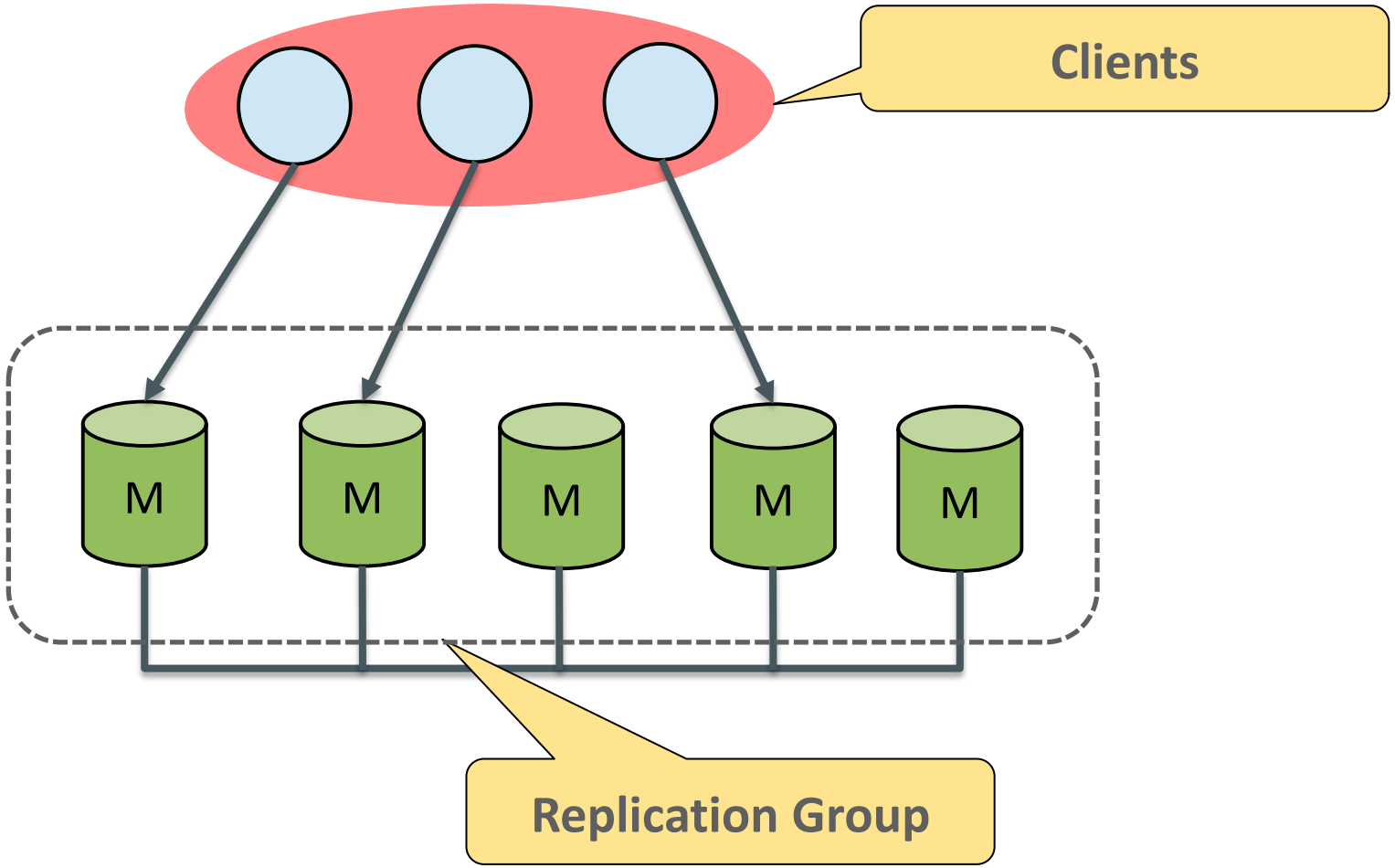
- MySQL 5.7でのレプリケーション強化点に関するより詳細な資料
  - MySQL 5.7 レプリケーション最新機能とロードマップ
    http://downloads.mysql.com/presentations/20151030_03_MySQL5.7_Replication_jp.pdf

MySQL グループ・レプリケーション

ORACLE

1 概要

MySQL グループ・レプリケーション



MySQL グループ・レプリケーション

- **MySQL グループ・レプリケーションとは？**
  - どのノードでもデータ更新可能なレプリケーションプラグイン
  - 自動的な競合処理、リカバリ、グループ・メンバーシップ
- **MySQL グループ・レプリケーションの利点**
  - 障害発生時にフェイルオーバー不要
  - フォールトトレラント
  - どのノードでもデータ更新可能
  - 自動的なグループ再構成（障害発生時、再接続時）
  - レプリケートされた高可用性データベースを提供.
  - 自動分散強調（スプリットブレインとメッセージの損失から保護）
  - 少ない管理コスト

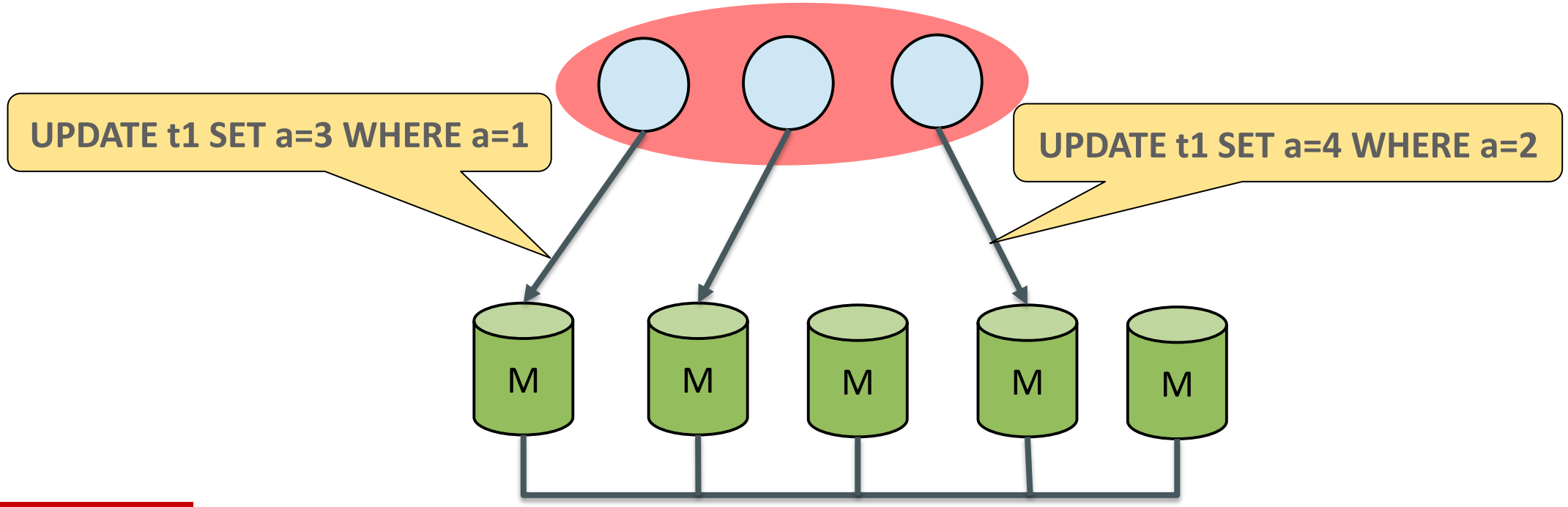
※2016年5月現在、GAになっていません。
今後Pluginとして追加リリースする予定です。

2 MySQL グループ・レプリケーション

2.1 マルチマスター

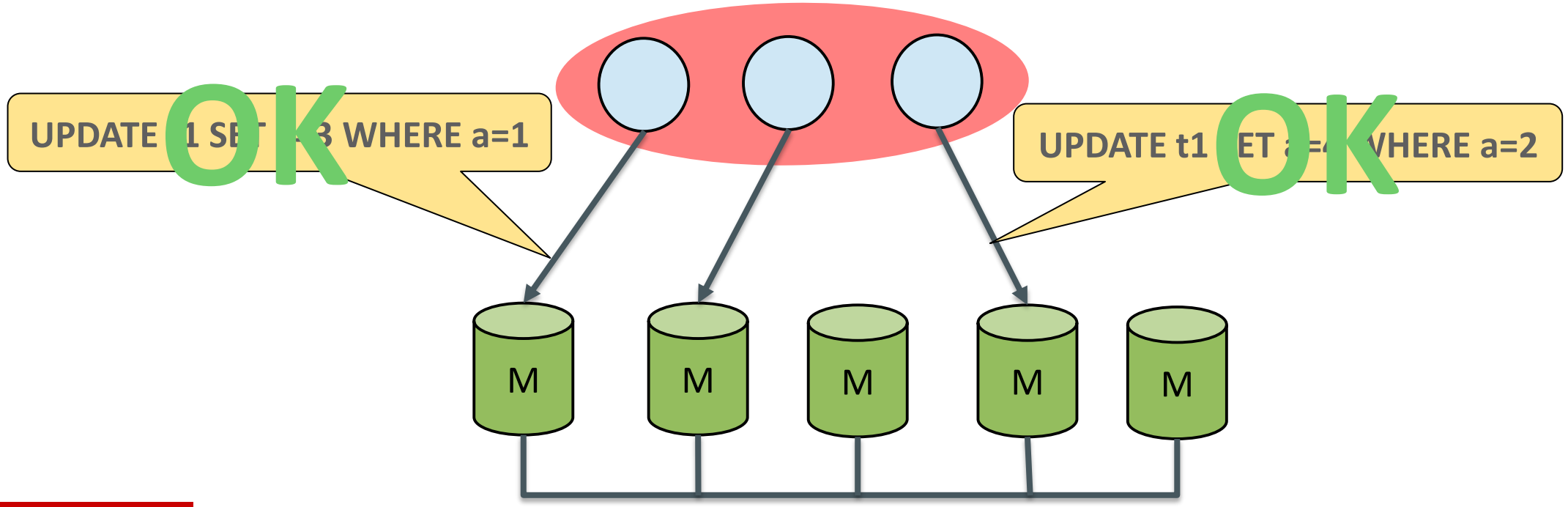
マルチマスター、どこでも更新可能！

- 異なるサーバー上での2つのトランザクションは、同じデータを更新可能
- 競合が検出され、対処される
  - 先にコミットしたトランザクションが優先される



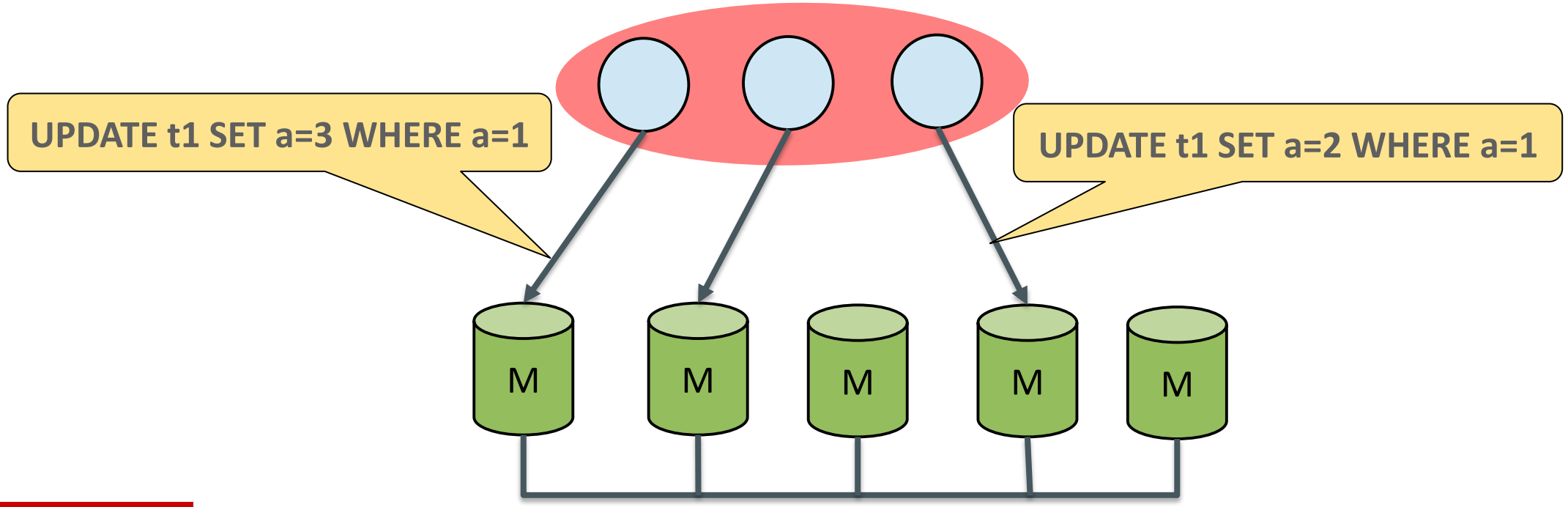
マルチマスター、どこでも更新可能！

- 異なるサーバー上での2つのトランザクションは、同じデータを更新可能
- 競合が検出され、対処される
  - 先にコミットしたトランザクションが優先される



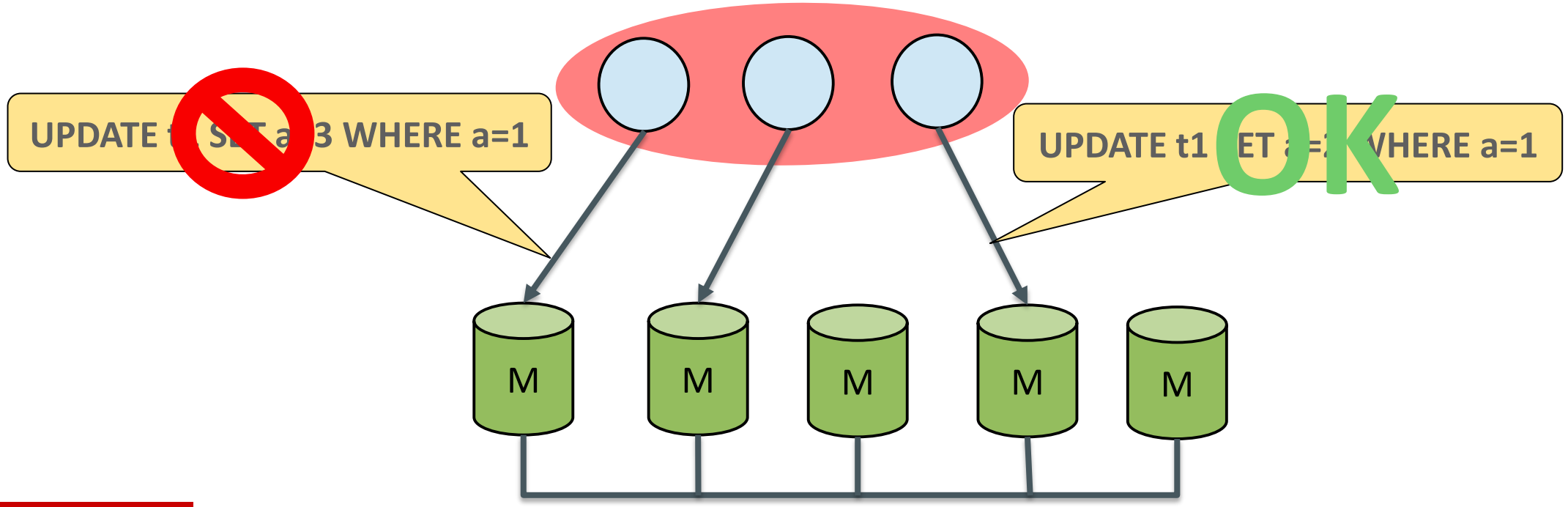
マルチマスター、どこでも更新可能！

- 異なるサーバー上での2つのトランザクションは、同じデータを更新可能
- 競合が検出され、対処される
  - 先にコミットしたトランザクションが優先される



マルチマスター、どこでも更新可能！

- 異なるサーバー上での2つのトランザクションは、同じデータを更新可能
- 競合が検出され、対処される
  - 先にコミットしたトランザクションが優先される



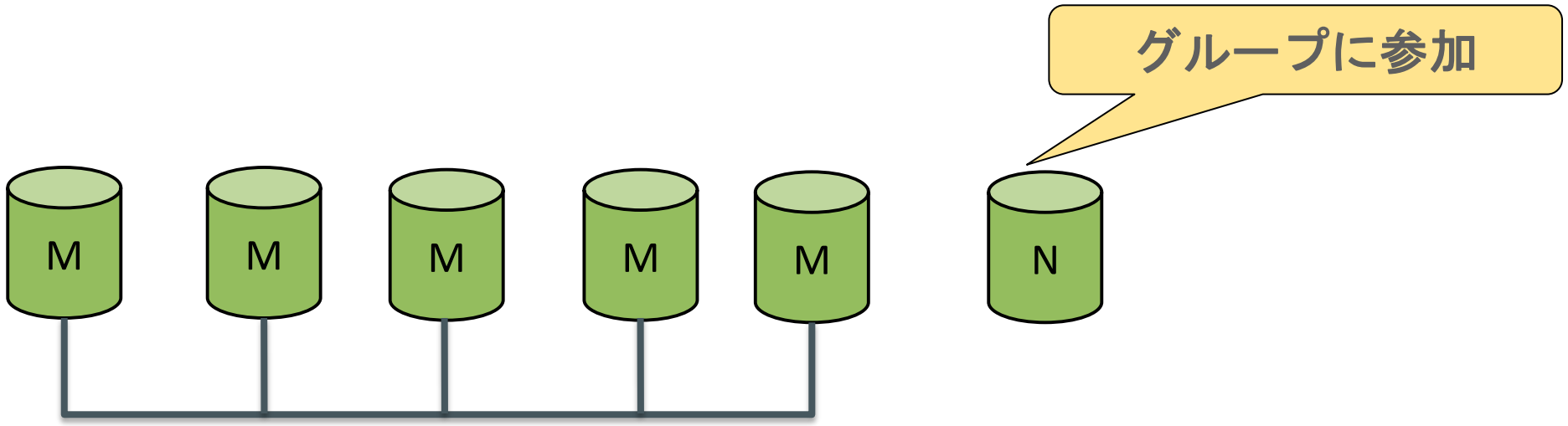
2 MySQL グループ・レプリケーション

2.1 マルチマスター

2.2 自動分散型サーバー復旧

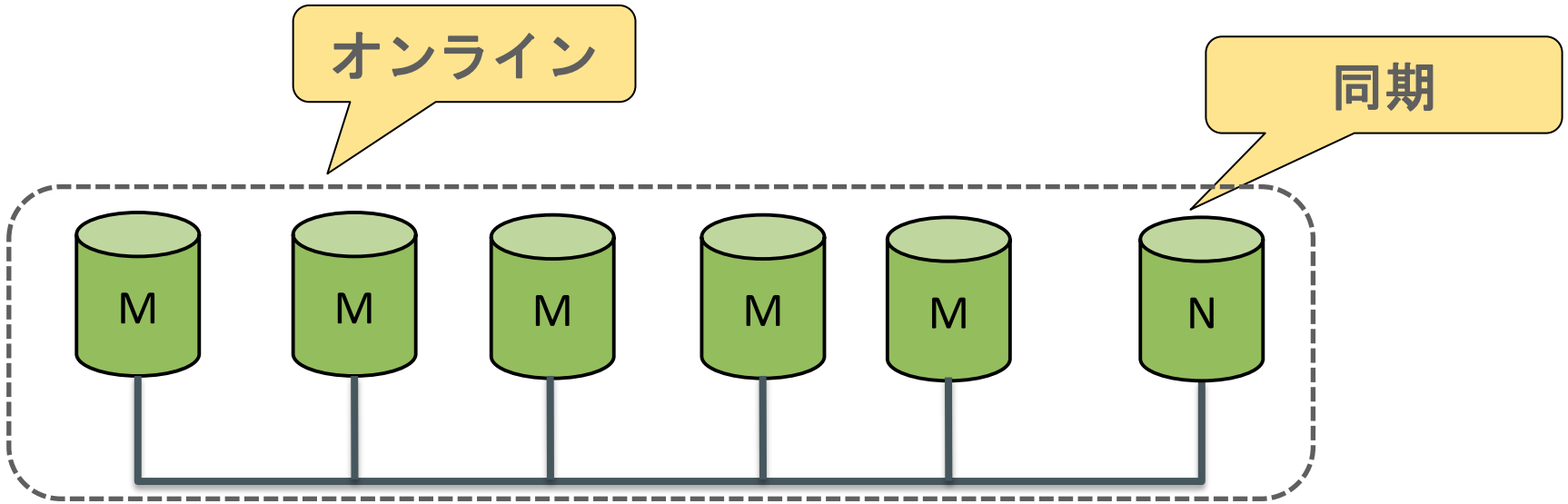
自動分散型サーバー復旧

- サーバーがグループに参加すると自動的に他のサーバーと同期する



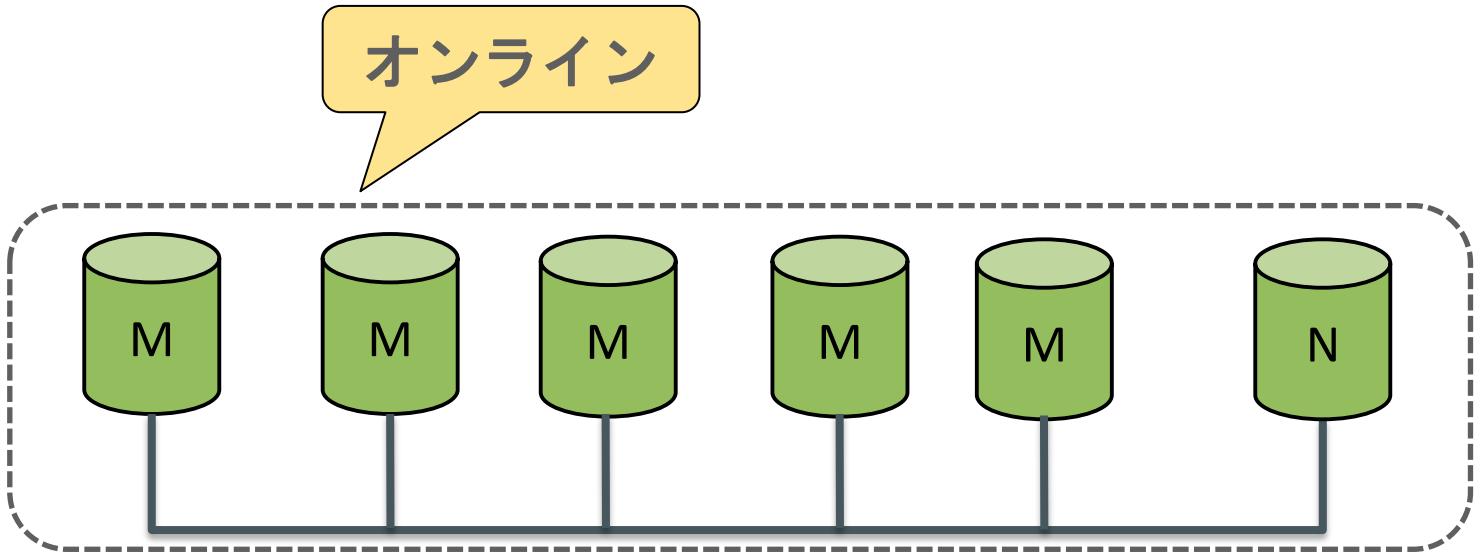
自動分散型サーバー復旧

- サーバーがグループに参加すると自動的に他のサーバーと同期する



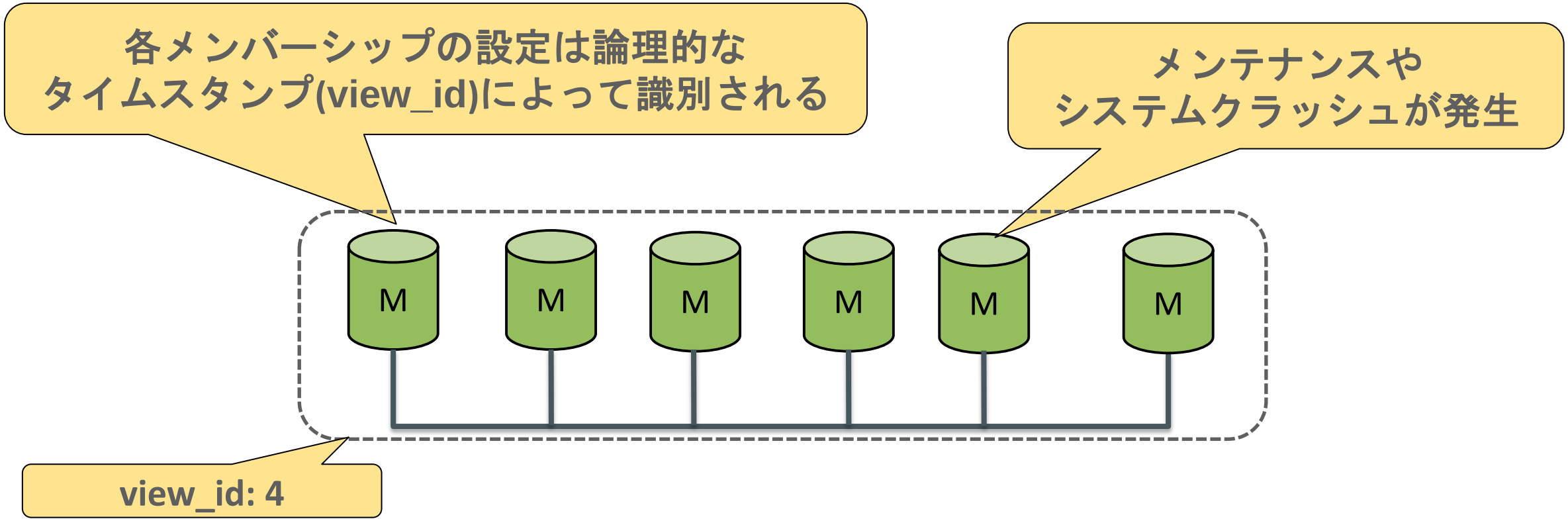
自動分散型サーバー復旧

- サーバーがグループに参加すると自動的に他のサーバーと同期する



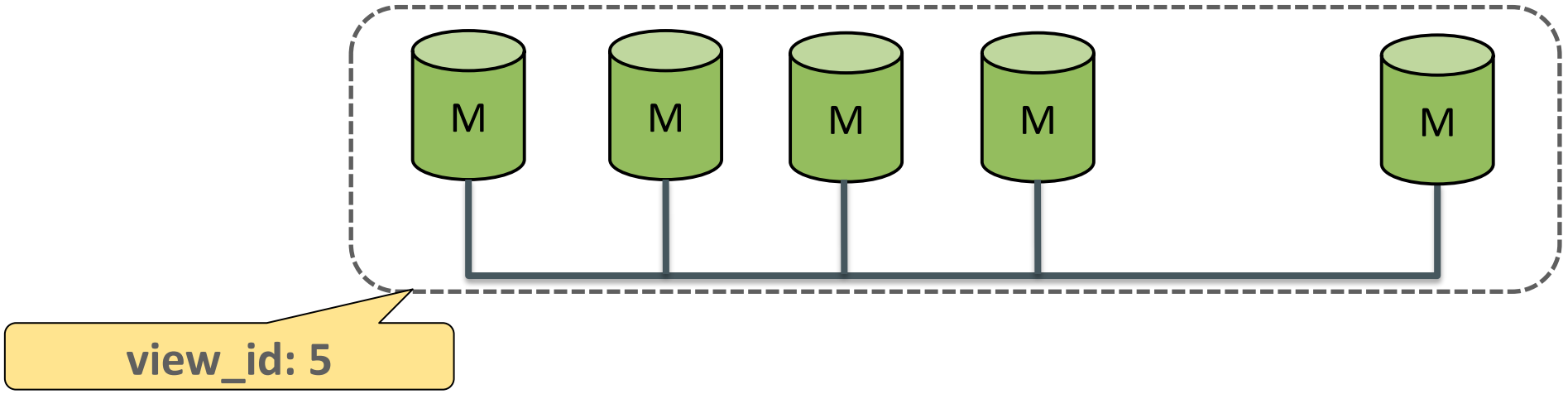
自動分散型サーバー復旧

- サーバーがグループから外れると、他のサーバーは自動的に検知する



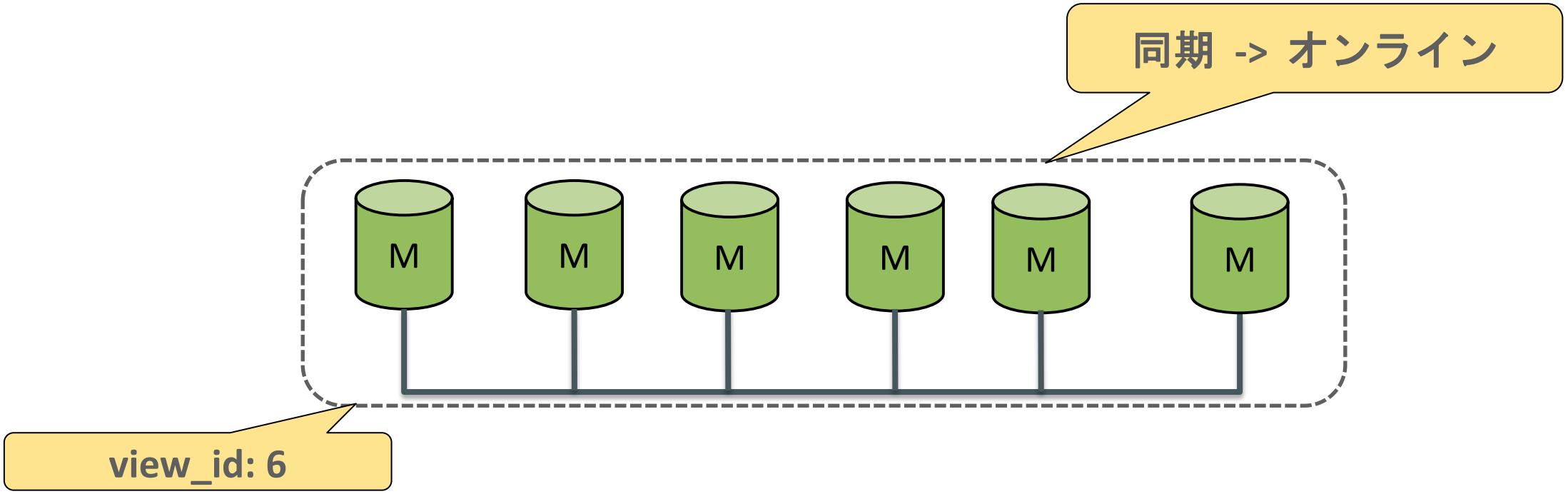
自動分散型サーバー復旧

- サーバーがグループから外れると、他のサーバーは自動的に検知する



自動分散型サーバー復旧

- サーバーが再びグループに参加すると、自動的に他のサーバーと同期する



2 MySQL グループ・レプリケーション

2.1 マルチマスター

2.2 自動分散型サーバー復旧

2.3 MySQL/InnoDB look & feel

MySQL/InnoDB look & feel

- プラグインをロードし、レプリケーションをスタート
- グループ・レプリケーションの状態をパフォーマンス・スキーマから確認

```
mysql> SET GLOBAL group_replication_group_name= "9eb07c6d-5e24-11e5-854b-34028662c0cd";
mysql> START GROUP_REPLICATION;

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 ***************************
 ...
```

MySQL/InnoDB look & feel

- プラグインをロードし、レプリケーションをスタート
- グループ・レプリケーションの状態をパフォーマンス・スキーマから確認

```
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/InnoDB look & feel

- プラグインをロードし、レプリケーションをスタート
- グループ・レプリケーションの状態をパフォーマンス・スキーマから確認

```
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
 ...
```

2 MySQL グループ・レプリケーション

2.1 マルチマスター

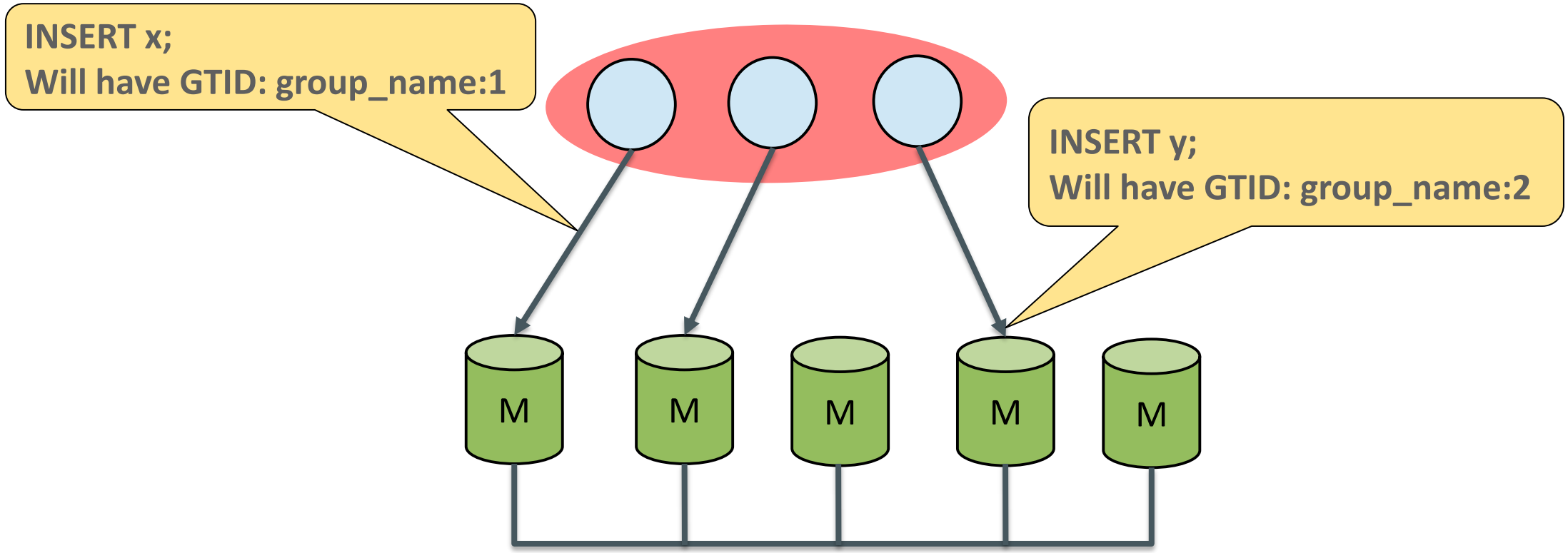
2.2 自動分散型サーバー復旧

2.3 MySQL/InnoDB look & feel

2.4 GTIDサポート

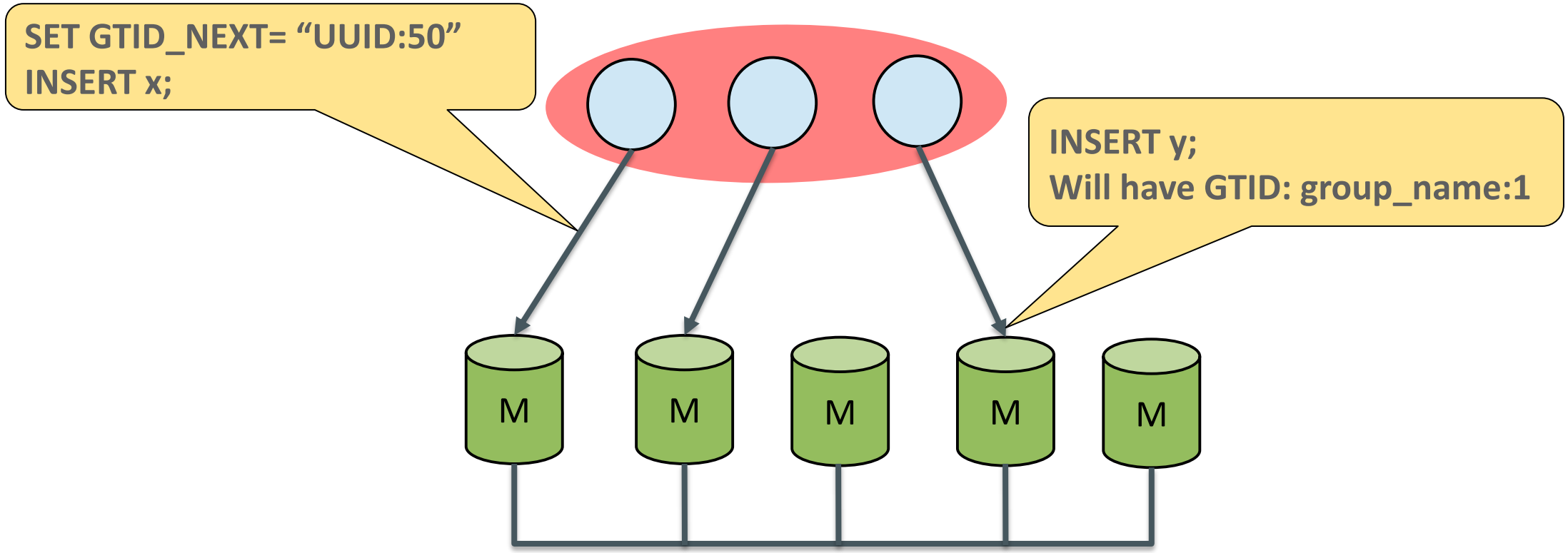
GTIDサポート

- 全グループメンバーは同じUUID、グループ名を持つ



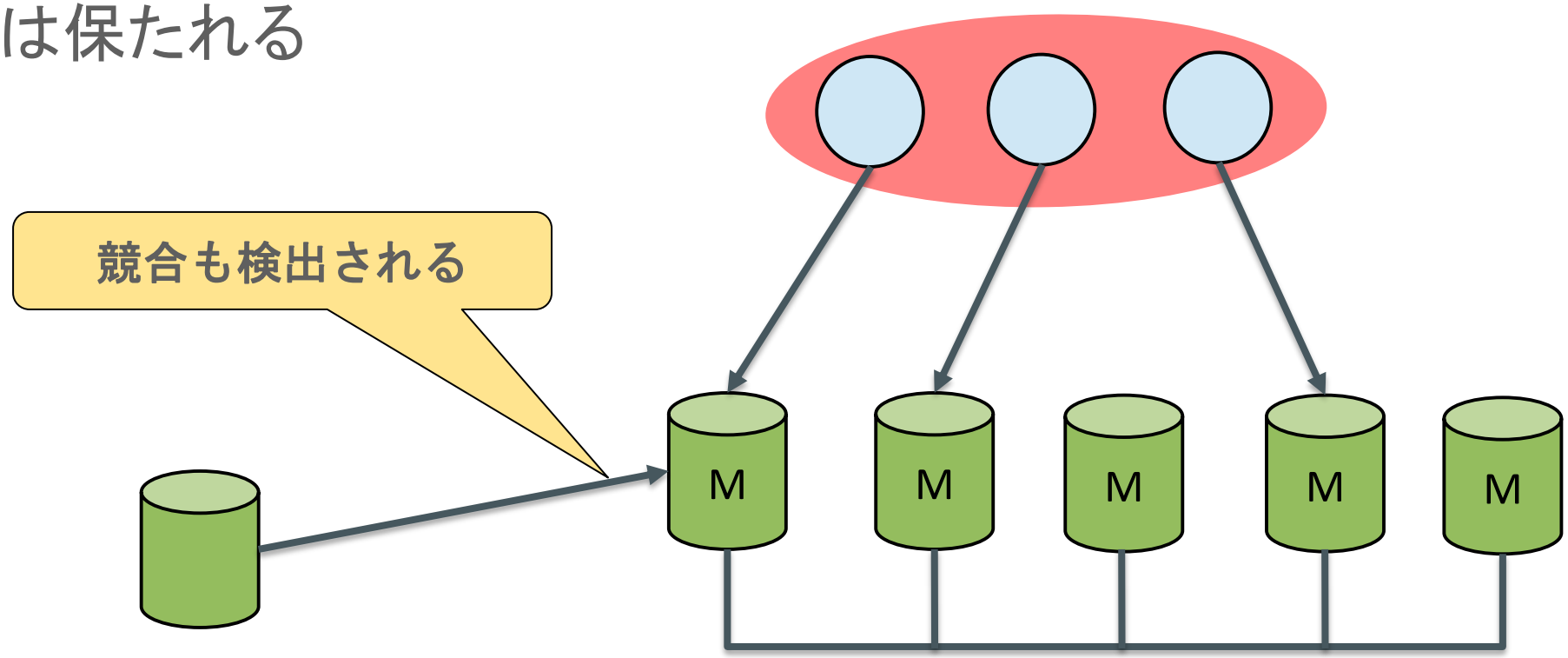
GTIDサポート

- 全グループメンバーは同じUUID、グループ名を持つ



GTIDサポート

- グループ外のサーバーからグループにレプリケート可能
- GTIDは保たれる

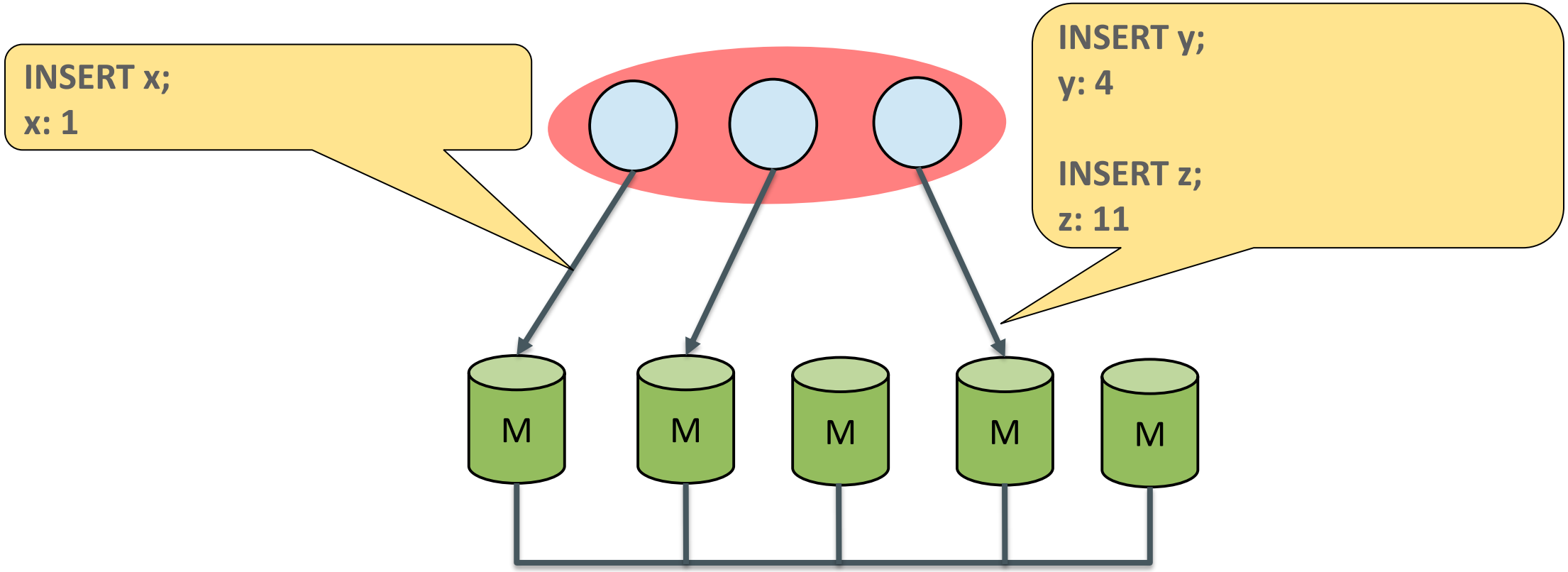


2 MySQLグループ・レプリケーション

- 2.1 マルチマスター
- 2.2 自動分散型サーバー復旧
- 2.3 MySQL/InnoDB look & feel
- 2.4 GTIDサポート
- 2.5 オートインクリメント値の設定/処理

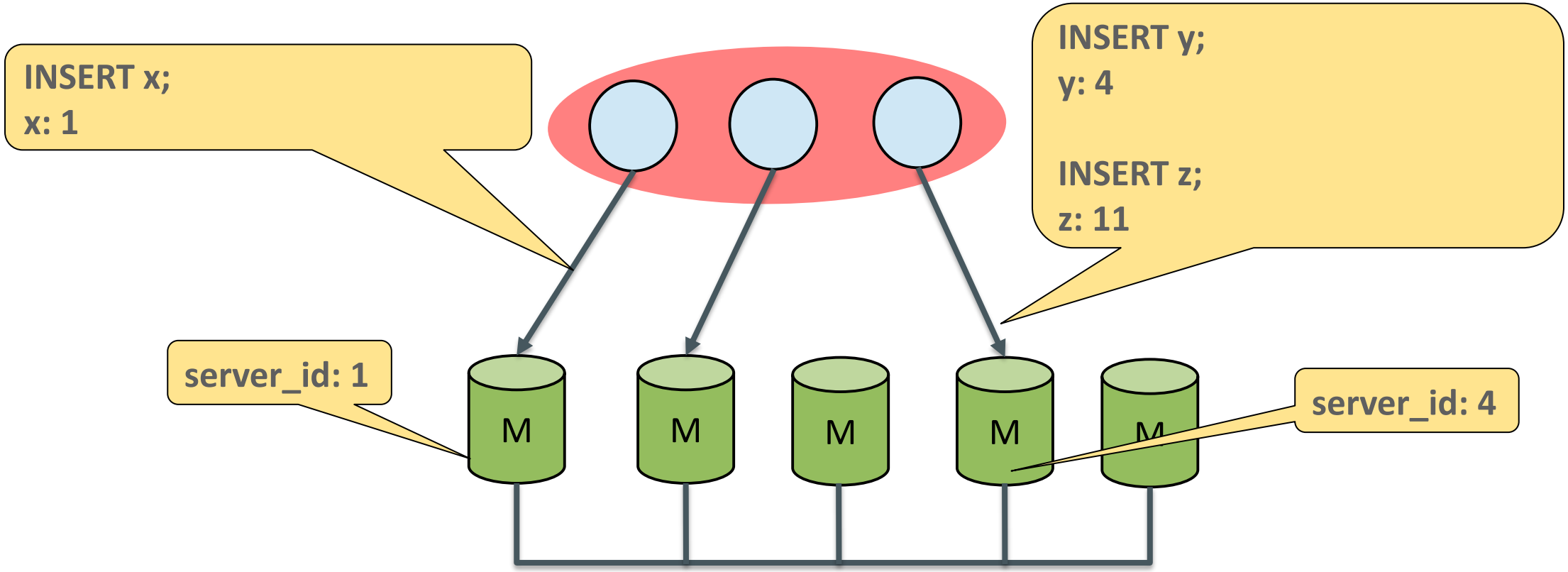
オートインクリメント値の設定/処理

- オートインクリメント値は、グループ内のメンバーと同じ値を生成しないように構成されている



オートインクリメント値の設定/処理

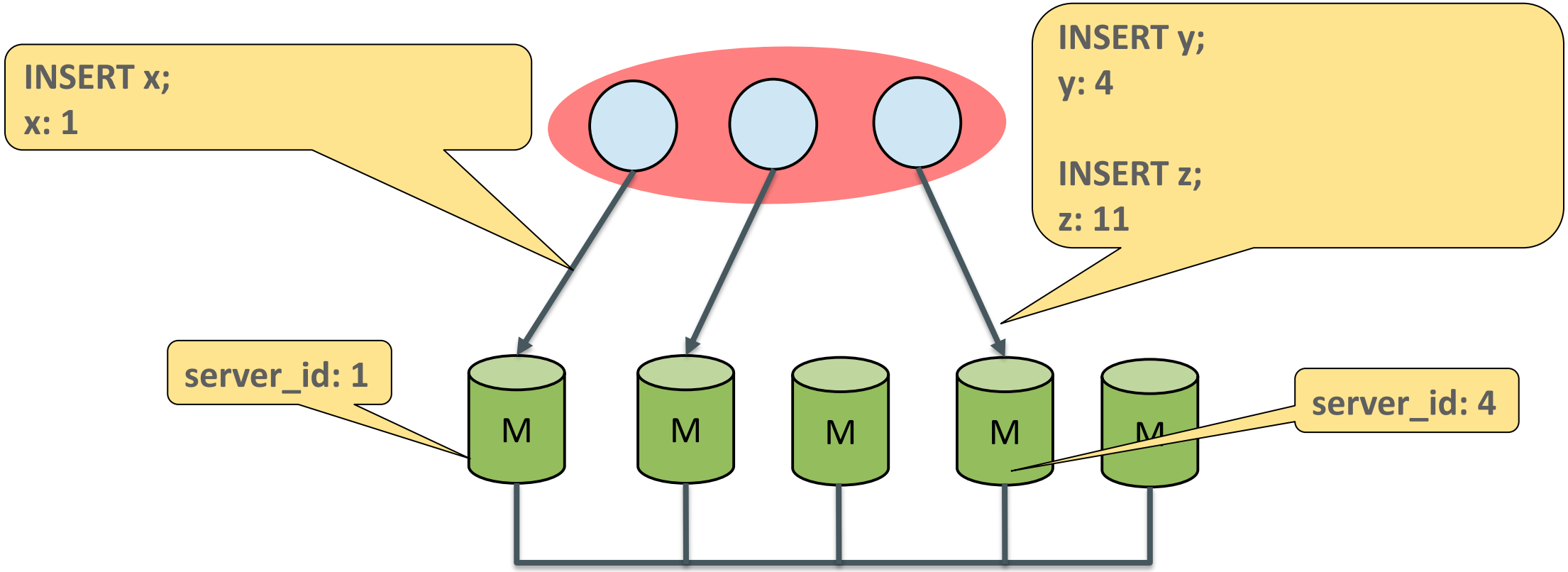
- デフォルトではオフセットはserver_idによって提供され、増分値は7



[1]: http://mysqlhighavailability.com/mysql-group-replication-auto-increment-configuration-handling/

オートインクリメント値の設定/処理

- GROUP_REPLICATION_AUTO_INCREMENT_INCREMENTオプションで、増分値を変更可能



2 MySQL グループ・レプリケーション

- 2.1 マルチマスター
- 2.2 自動分散型サーバー復旧
- 2.3 MySQL/InnoDB look & feel
- 2.4 GTIDサポート
- 2.5 オートインクリメント値の設定/処理
- 2.6 新しい分散コミュニケーションエンジン

ORACLE

新しい分散コミュニケーションエンジン

- マルチOSサポート
  - Linux, Windows, OSX, Solaris, FreeBSD…
- サードパーティソフトウェア不要
- ネットワークのマルチキャストサポート不要
  - MySQL グループ・レプリケーションは、マルチキャストが使用できないクラウド環境でも使用可能
- メッセージサイズの制限なし
- 追加プロセス無し
  - MySQL グループ・レプリケーションは同じソフトウェアスタックに含まれる

2 MySQL グループ・レプリケーション

- 2.1 マルチマスター
- 2.2 自動分散型サーバー復旧
- 2.3 MySQL/InnoDB look & feel
- 2.4 GTIDサポート
- 2.5 オートインクリメント値の設定/処理
- 2.6 新しい分散コミュニケーションエンジン
- 2.7 設計上の要件

ORACLE

設計上の要件

- InnoDBのみをサポート
- 全テーブルに主キーが必要
- GTID有効
- 楽観実行
  - 競合によって、トランザクションはabortされるかもしれないし、COMMITされるかもしれない
  - データ競合発生時は、先にCOMMITしたトランザクションが優先して確定される。後からCOMMITしたトランザクションはabortされる。
    ⇒データ競合の発生は、トランザクションCOMMIT時に検知される

2 MySQL グループ・レプリケーション

- 2.1 マルチマスター
- 2.2 自動分散型サーバー復旧
- 2.3 MySQL/InnoDB look & feel
- 2.4 GTIDサポート
- 2.5 オートインクリメント値の設定/処理
- 2.6 新しい分散コミュニケーションエンジン
- 2.7 設計上の要件
- 2.8 制限事項

制限事項

- オンラインでのスキーマ変更（オンラインDDL）はサポートされない

※スキーマ変更手順：以下の手順を全ノードに対して順番に実行する

1.グループから離脱

2.バイナリログの出力を停止して、DDLを実行

3.グループに参加

3 アーキテクチャ

3.1 Introduction

MySQL Group Replication is

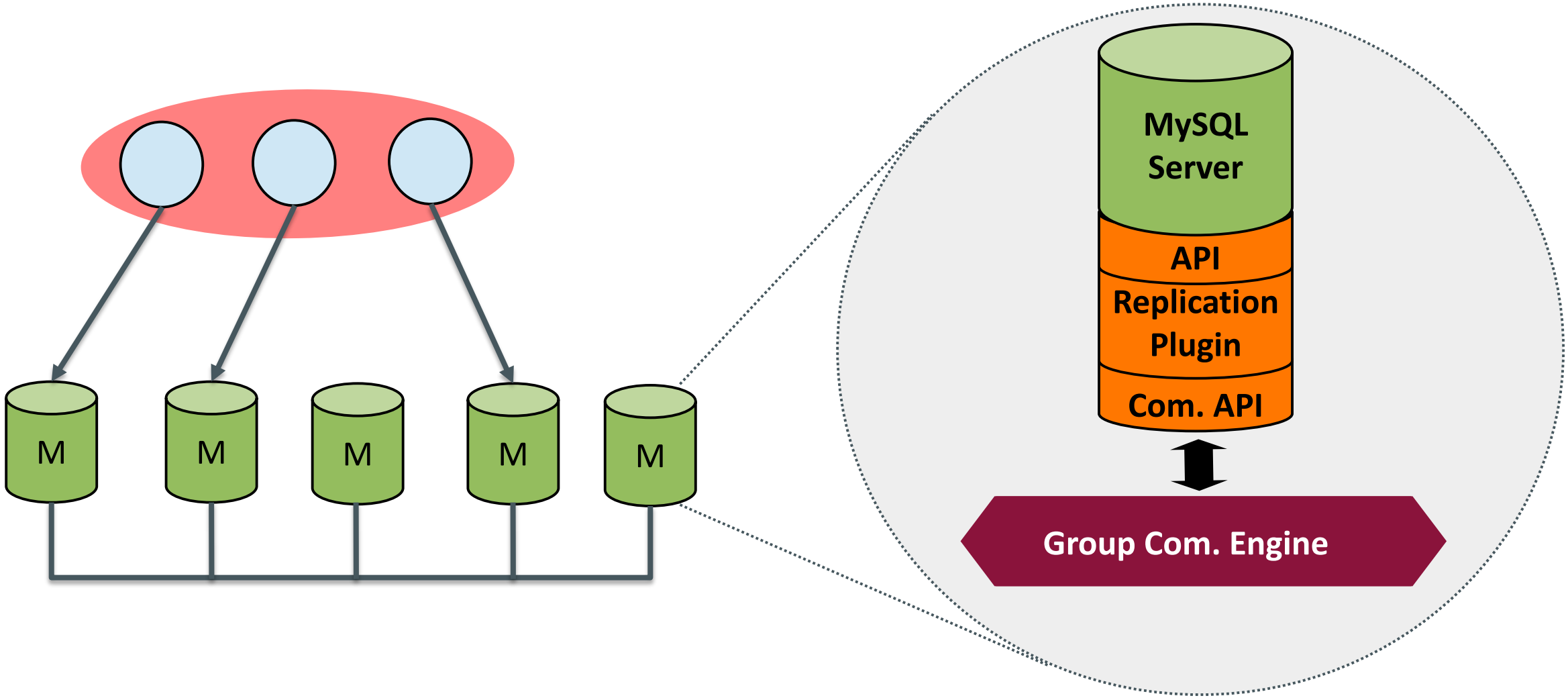
- **Built on top of proven technology!**
  - Shares much of MySQL Replication infrastructure – thence does not feel alien!
  - Multi-Master approach to replication.
- **Built on reusable components!**
  - Layered implementation approach.
  - Interface driven development.
  - Decoupled from the server core.
  - The plugin registers as listener to server events.
  - Reuses the capture procedure from regular replication.
  - Provides further decoupling from the communication infrasctructure.

3 アーキテクチャ

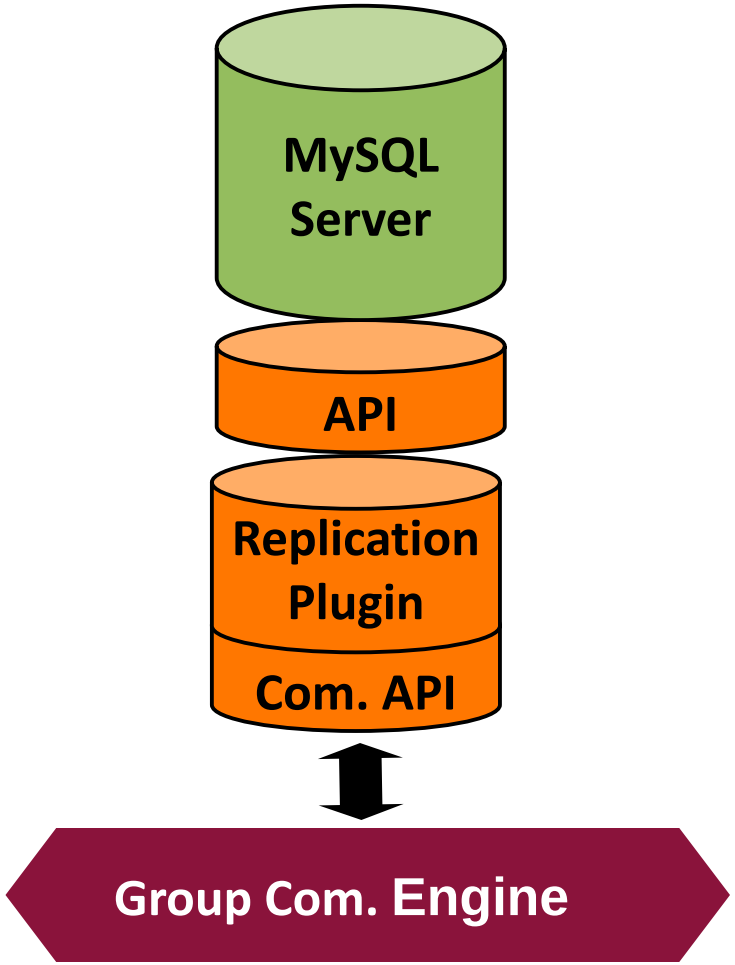
3.1 Introduction

3.2 Major Building Blocks

Major Building Blocks (1)

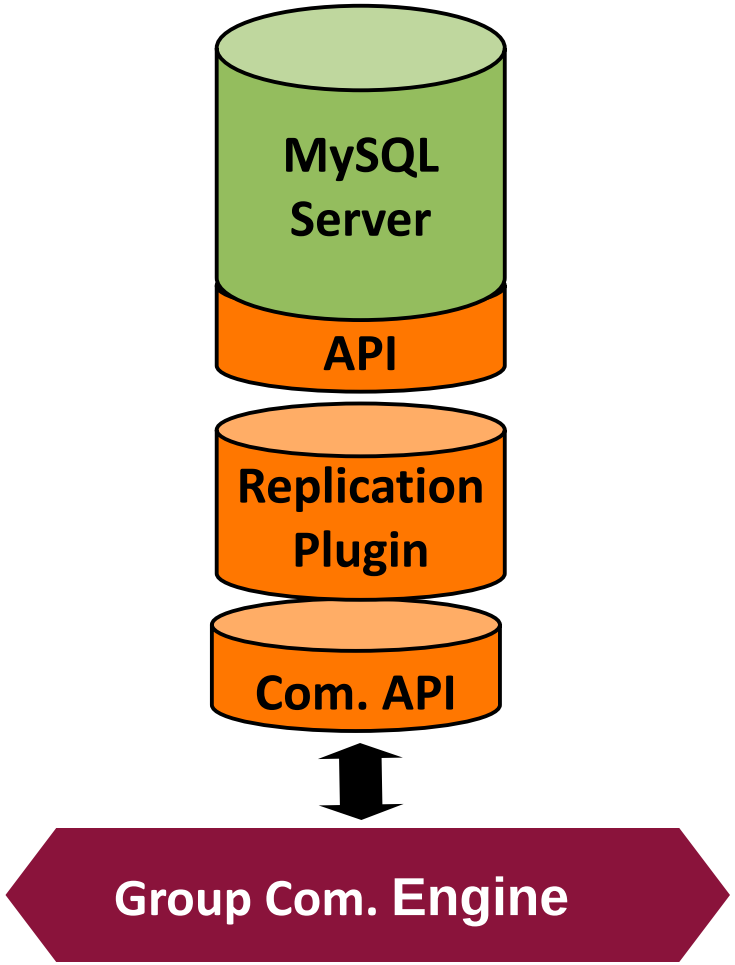


Major Building Blocks (2)



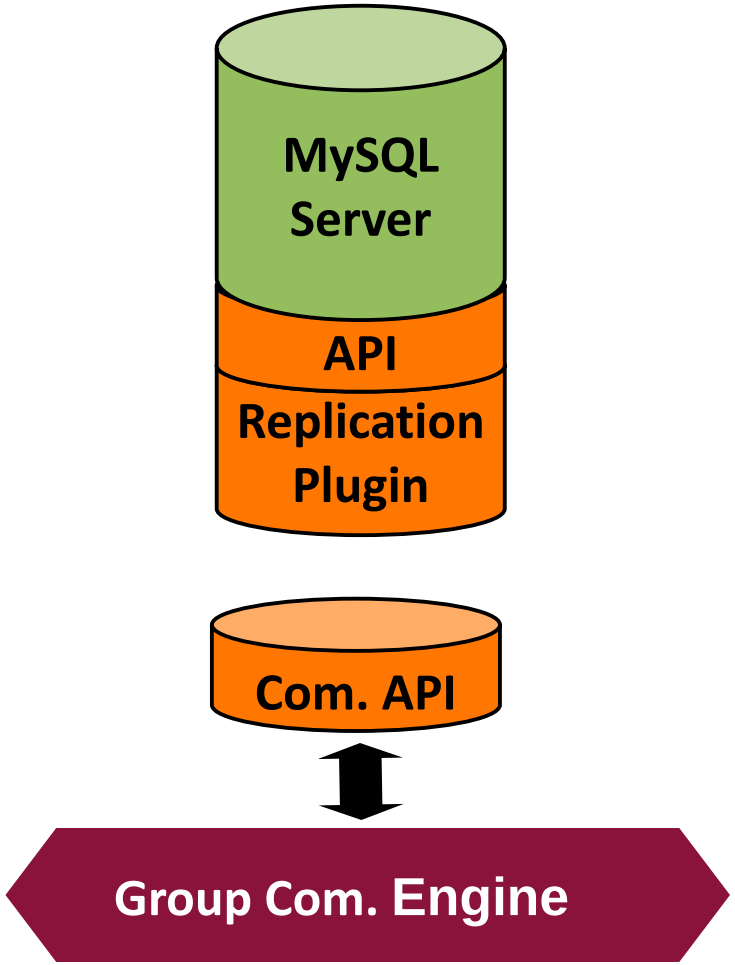
- Server calls into the plugin through a generic interface
  - (Most of server) internals are hidden from the plugin.
  - Some of the semi-sync interfaces were reused. Others were deployed.
- Plugin interacts with the server through a generic interface
  - Replication plugin determines the fate of the commit operation through a well defined server interface.
  - The plugin makes use of the relay log infrastructure to inject changes in the receiving server.

Major Building Blocks (3)



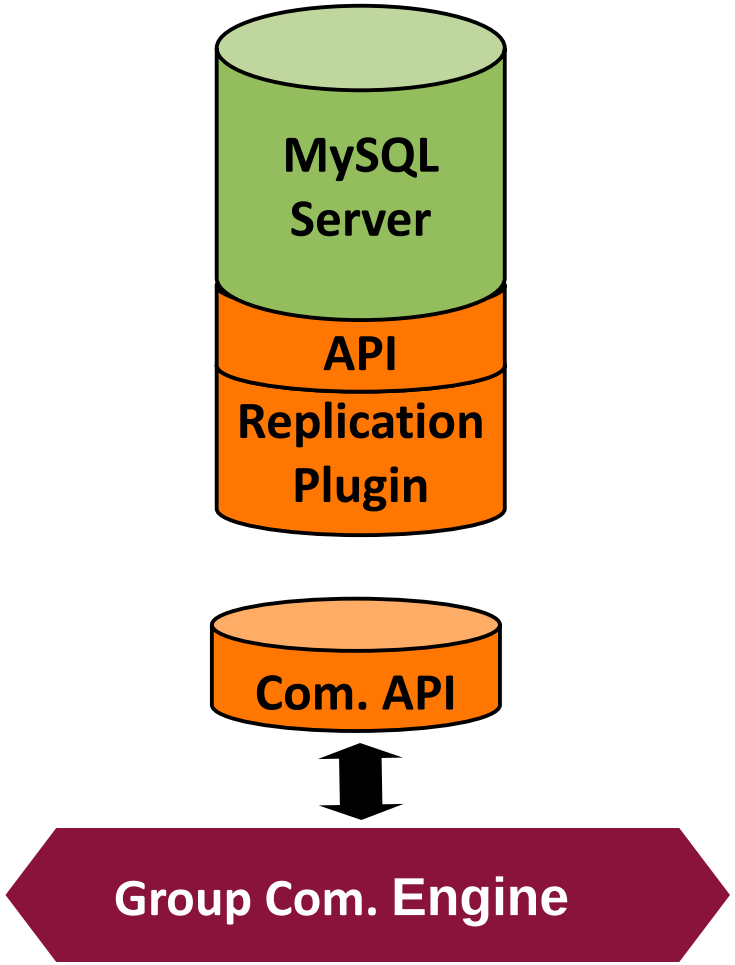
- The plugin is responsible for
  - Maintaining distributed execution context.
  - Detecting conflicts.
  - Handling distributed recovery:
    - Detect membership changes;
    - Donate state if needed;
    - Collect state if needed.
  - Receiving and handling transactions from other members.
  - Deciding the fate of on-going transactions.

Major Building Blocks (4)



- The communication API (and bindings) is responsible for:
  - Abstracting the underlaying communication engine from the plugin itself.
  - Maping the interface to a specific communication engine.

Major Building Blocks (5)



- The communication engine:
  - Variant of Paxos developed at MySQL.
  - Building block to provide distributed agreement between servers.

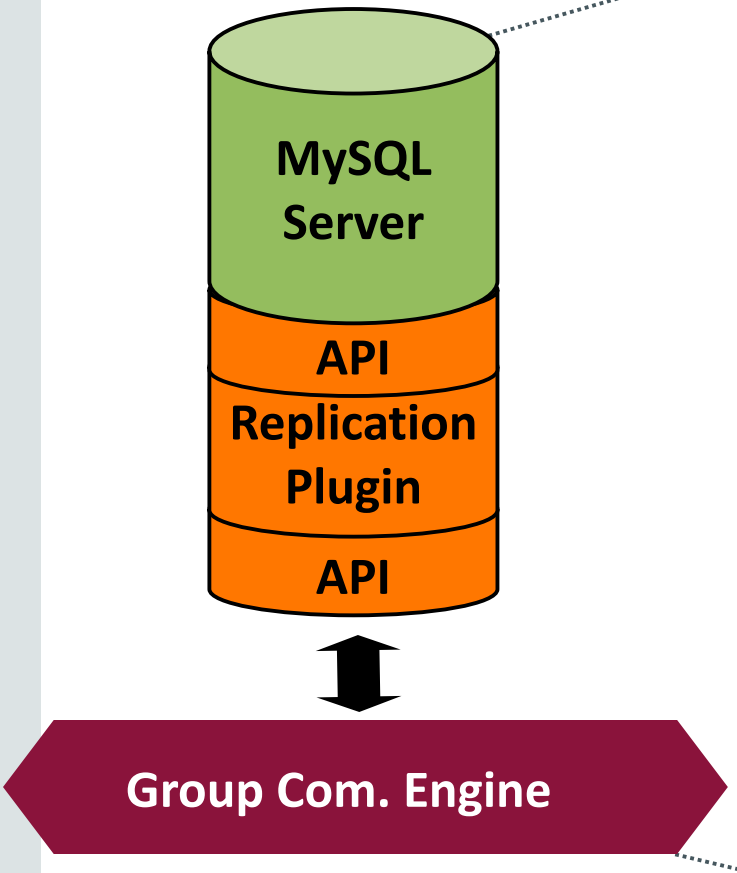
3 アーキテクチャ

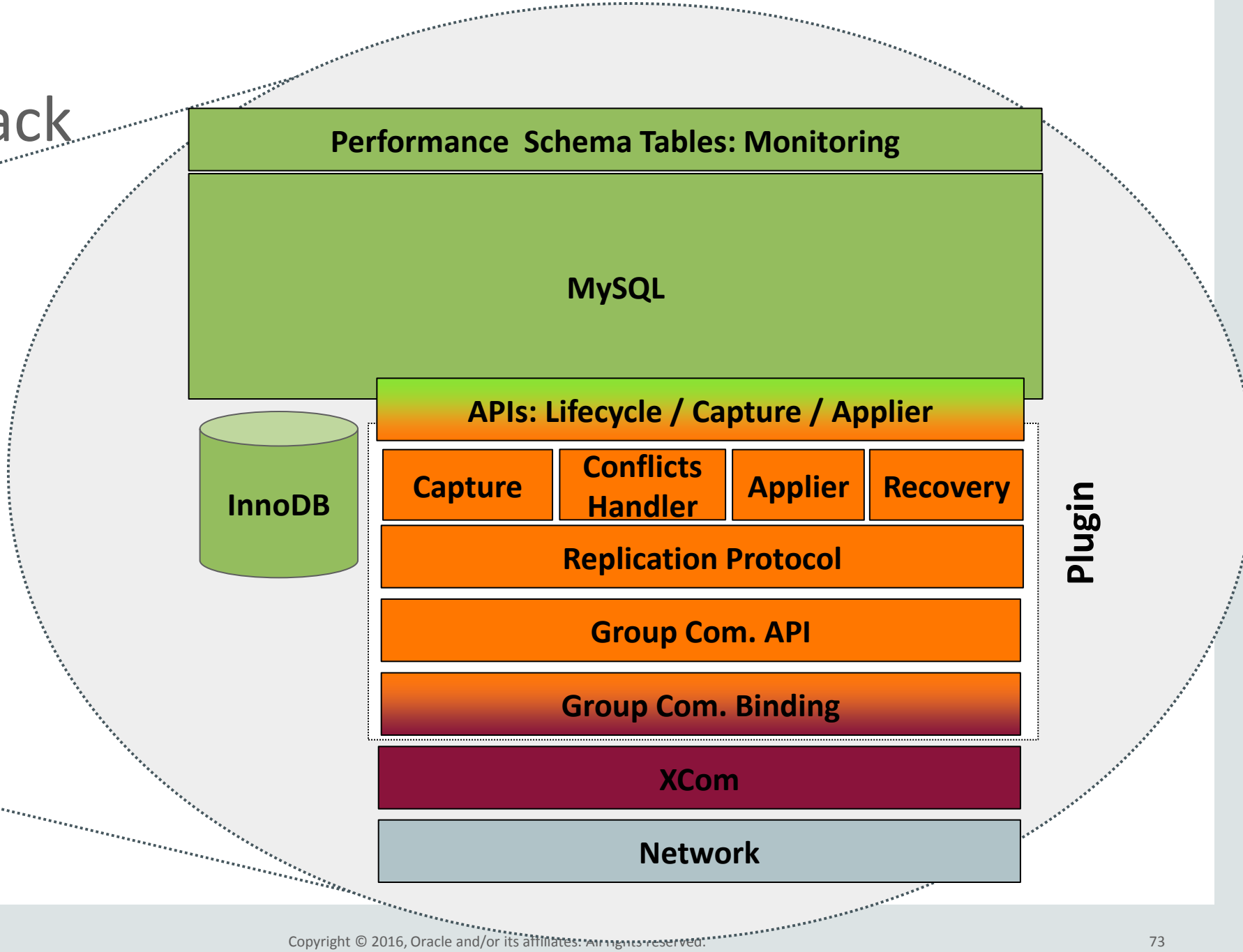
3.1 Introduction

3.2 Major Building Blocks

3.3 The Complete Stack

The Complete Stack





4 ユースケース

4.1 ユースケース

ユースケース

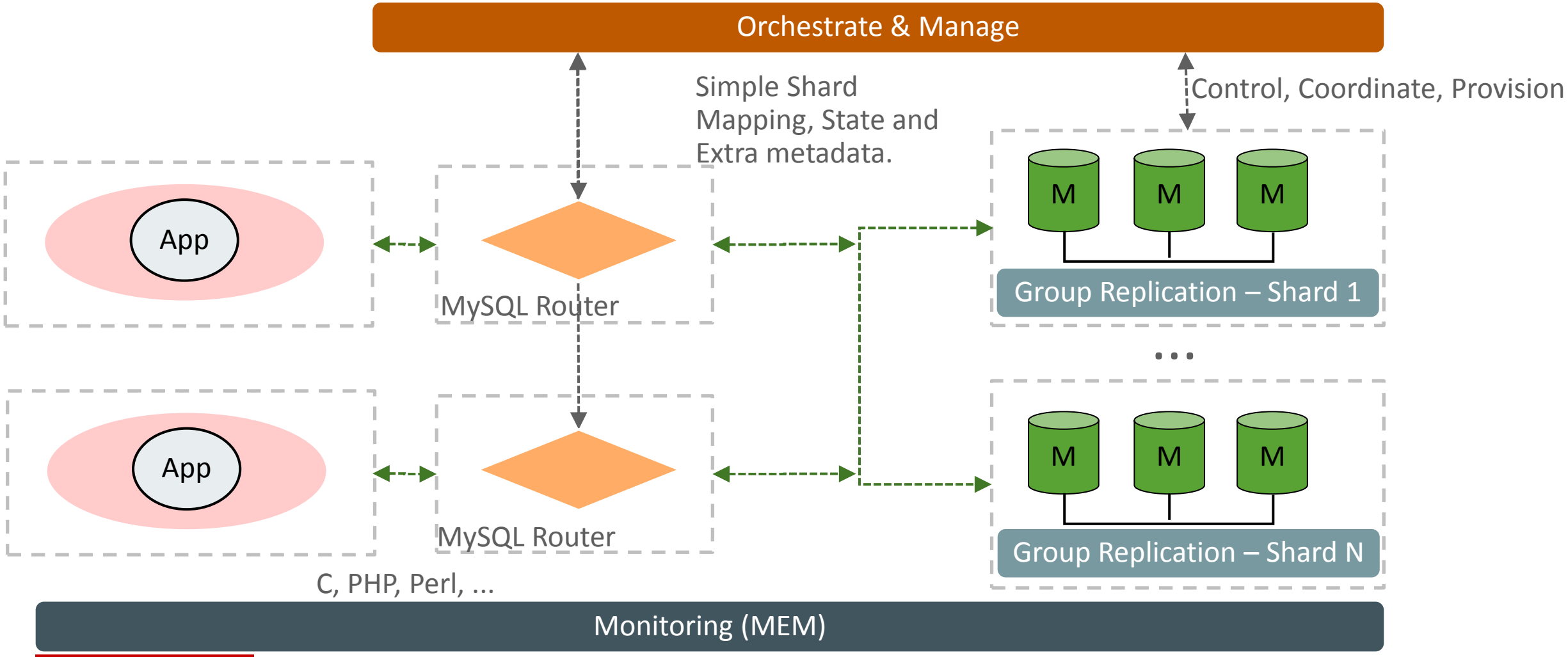
- **エラスティック・レプリケーション**
  - サーバー数を増やしたり減らしたり、非常に流動的なレプリケーション環境を必要とする場合
- **高可用性シャード**
  - シャーディングは書き込み処理の拡張性を実現する一般的な方法
  - ユーザーは高可用性シャードを実現するために、MySQL グループ・レプリケーションを使用可能
  - それぞれのシャード毎にレプリケーション・グループを作成
- **マスター/スレーブレプリケーションの代替**
  - 単一のマスターサーバーがボトルネックになる場合がある
  - グループ全体への書き込みは、特定の状況下でよりスケーラブルになる場合がある

4 ユースケース

4.1 ユースケース

4.2 全体像

拡張性、信頼性が高いMySQL環境



5 結論

まとめ

- **クラウドフレンドリー**
  - クラウドベースのインフラのようなエラスティック性が求められる環境と相性がいい技術
- **MySQLサーバーとの統合**
  - よく知られたMySQLサーバーのAPIとの統合
  - GTID、行ベースレプリケーション、パフォーマンス・スキーマとの統合
- **自己回復、管理負荷の軽減**
  - 自己回復：サーバーのフェイルオーバーを処理するための管理負荷無し
  - マルチマスター、フォールトトレラントで信頼性の高いMySQLサービスを提供

Appendix

- モジュールのダウンロード
  - http://labs.mysql.com

  ※2016年5月18日時点で公開されているモジュールは
  「MySQL Group Replication for MySQL Server 5.7.10」となっています。
  試すためにはMySQL 5.7.10をご用意ください。

- エンジニアブログ (news, technical information, and much more)
  - http://mysqlhighavailability.com

  ※チュートリアル記事
  Getting started with MySQL Group Replication
  http://mysqlhighavailability.com/getting-started-with-mysql-group-replication/

Q&A

Q&A①

Q：GTIDをベースに、トランザクションが実行された順番を確認できるか？

A：確認できます。

次ページの補足の通り、グループメンバーは全て同じGTIDを使用し、コミットが確定したトランザクションのみバイナリログに記録されます。そのため、GTIDの順番でトランザクションが実行された順番を確認できます。

また、それぞれのトランザクションがどのサーバーで実行されたのかは、GTIDからは判断できませんが、server_idもバイナリログに記録されているため、server_idを確認することで、それぞれのトランザクションがどのサーバーで実行されたのかも確認できます。

補足：グループレプリケーションでのコミット時の動作

- トランザクションコミット時は、グループ内の全サーバーと通信し、データ競合が発生しているか否かを確認する
- 競合が発生していなければ、そのトランザクションはコミットされる
  - 更新処理が実行され、バイナリログにGTIDと共に更新内容が記録される

  ※GTIDは、グループレプリケーションの全メンバーで共通
- 競合が発生していた場合は、そのトランザクションは取り消される
  - 更新処理は実行されないため、バイナリログにも記録されない

補足:グループレプリケーション使用時のバイナリログの内容

- test.t1テーブルに対して、server_id=1のサーバーからデータを更新した後、server_id=2のサーバーからデータを更新した時のバイナリログ例

```
mysql> show binlog events in 'yyamasaki-bin.000004' from 2580;
+----------------------+------+------------+-----------+-------------+-----------------------------------------------------------------+
| Log_name             | Pos  | Event_type | Server_id | End_log_pos | Info                                                            |
+----------------------+------+------------+-----------+-------------+-----------------------------------------------------------------+
| yyamasaki-bin.000004 | 2580 | Gtid       |         1 |        2641 | SET @@SESSION.GTID_NEXT= '0a06e216-1bdb-11e6-b1e0-080027ff56de:7' |
| yyamasaki-bin.000004 | 2641 | Query      |         1 |        2700 | BEGIN                                                           |
| yyamasaki-bin.000004 | 2700 | Table_map  |         1 |        2741 | table_id: 108 (test.t1)                                         |
| yyamasaki-bin.000004 | 2741 | Write_rows |         1 |        2777 | table_id: 108 flags: STMT_END_F                                 |
| yyamasaki-bin.000004 | 2777 | Xid        |         1 |        2804 | COMMIT /* xid=42 */                                             |
| yyamasaki-bin.000004 | 2804 | Gtid       |         2 |        2865 | SET @@SESSION.GTID_NEXT= '0a06e216-1bdb-11e6-b1e0-080027ff56de:8' |
| yyamasaki-bin.000004 | 2865 | Query      |         2 |        2929 | BEGIN                                                           |
| yyamasaki-bin.000004 | 2929 | Table_map  |         2 |        2970 | table_id: 108 (test.t1)                                         |
| yyamasaki-bin.000004 | 2970 | Write_rows |         2 |        3006 | table_id: 108 flags: STMT_END_F                                 |
| yyamasaki-bin.000004 | 3006 | Xid        |         2 |        3033 | COMMIT /* xid=49 */                                             |
+----------------------+------+------------+-----------+-------------+-----------------------------------------------------------------+
10 rows in set (0.00 sec)
```

どのサーバーから実行された
トランザクションか判断可能

GTIDが増えている

Q&A②

Q：新しいメンバーがグループに参加した時に、どのような仕組みでデータを同期しているか？

A：GTIDレプリケーションの仕組みでデータを同期しています。

※グループ内のメンバーのバイナリログから更新内容の差分を受取ってデータを同期する為、同期に必要なバイナリログが欠損している場合は、リカバリ処理が完了せずに、グループに参加できません。

Q&A③

Q:オンラインDDLが実行できないのであれば、DDLはどういう手順で
　実行するのか？

A:以下の手順を全ノードに対して順番に実行することでDDLを実行可能です。

1.グループから離脱

2.バイナリログの出力を停止して、DDLを実行

3.グループに参加

※同様の説明を63ページにも追記しました。

※（他の部分での説明内容も含め）あくまで2016年5月18日時点のLab版(実験版)での挙動です。、
今後のリリースにおいて予告なく挙動が変更になる可能性があります。特にオンラインDDLついては、
開発チーム内でも、どのように対処すべきかディスカッションが継続されています。


ORACLE®