MySQL Replication and MySQL Fabric

Ryusuke Kajiyama / 梶山隆輔

MySQL Sales Consulting Senior Manager, Asia Pacific & Japan

ORACLE

MySQL

MySQLレプリケーションの用途

スケーラビリティ

Webで中心となる参照処理を分散してシステムとしての拡張性を向上

可用性

データの複製を持たせ、マスターに障害が発生した場合は切り替え

集計処理やバックアップ

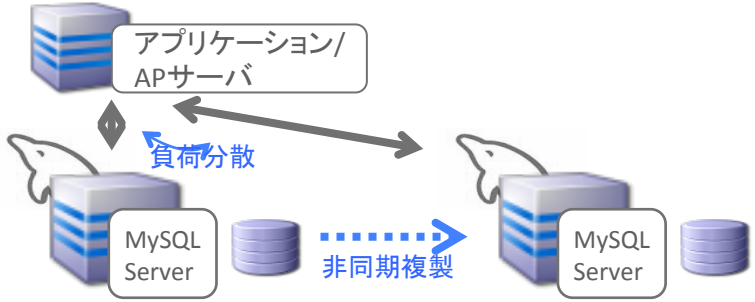
オンラインでの処理から処理を分離することによって全体の性能を維持

ディザスタリカバリ

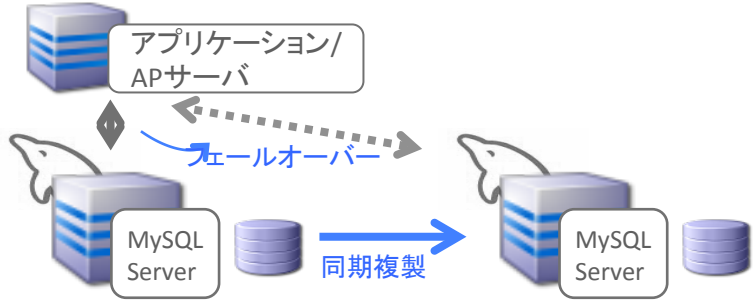
データセンター間や拠点間でデータを転送して地理的冗長性を確保

MySQLの高可用性構成

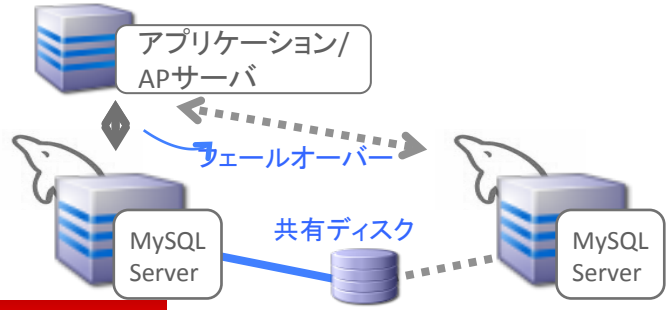
- レプリケーション（標準機能）
  非同期&準同期データレプリケーション



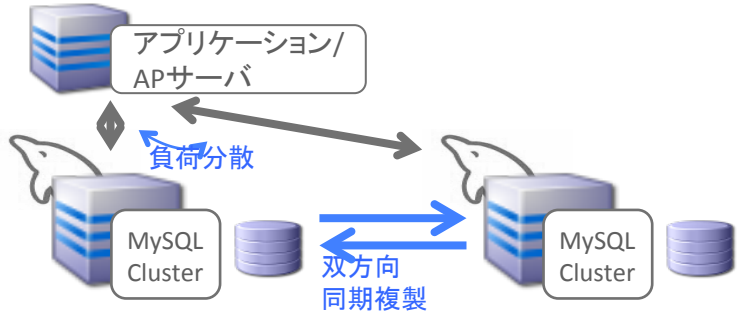
- MySQL+DRBD
  Linux用のノード間データコピー



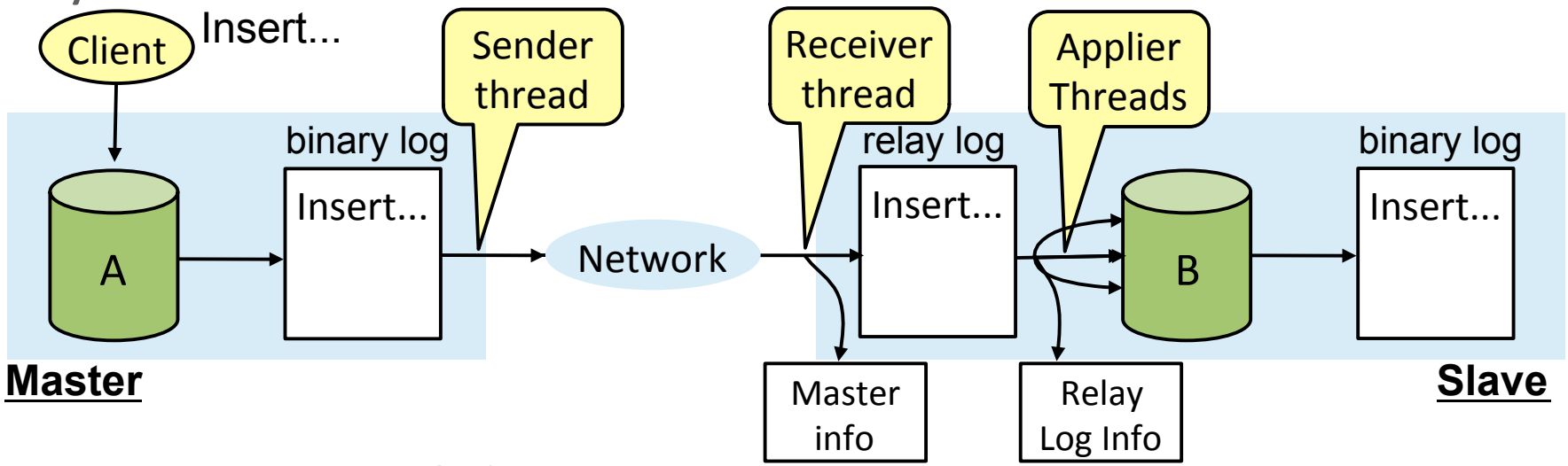
- クラスタリング ソフトウェア利用
  共有ディスクにデータを格納



- MySQL Cluster
  シェアードナッシング型高可用性クラスタ



MySQLレプリケーション

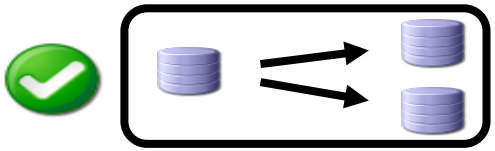


- MySQLの標準機能
- バイナリログの内容を転送する
  - 標準では非同期型、設定で準同期を選択可能
- スレーブでも参照処理可能

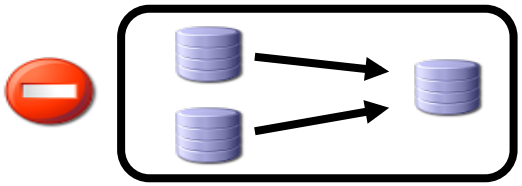
MySQL レプリケーションの構成パターン

マスタ > スレーブ

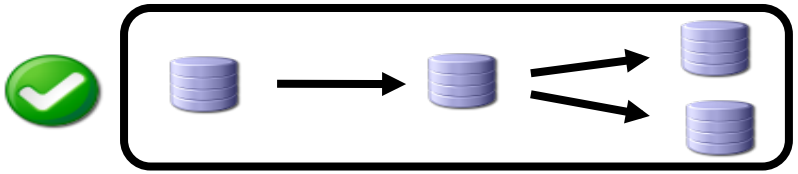
マスタ > マルチスレーブ



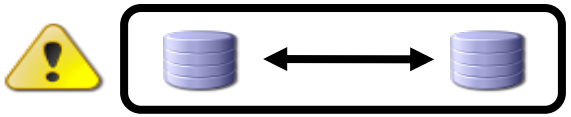
マルチマスタ > スレーブ (マルチソース)



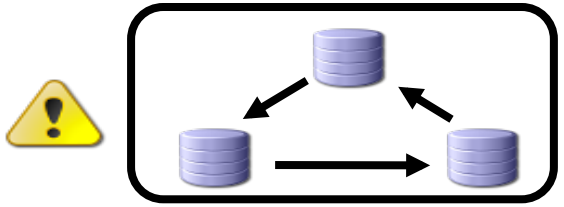
マスタ > スレーブ > マルチスレーブ



マスタ < > マスタ (マルチマスタ)



循環型 (マルチマスタ)



ORACLE

MySQL 5.6: Replication改善点

パフォーマンス

- マルチスレッド スレーブ
- バイナリログのグループコミット
- 行ベース レプリケーションの転送データ量の削減

フェールオーバー & リカバリ

- Global Transaction Identifiers
- レプリケーション フェールオーバー & 管理ユーティリティ
- スレーブ&バイナリログの耐障害性向上

データの正確性

- レプリケーション チェックサム

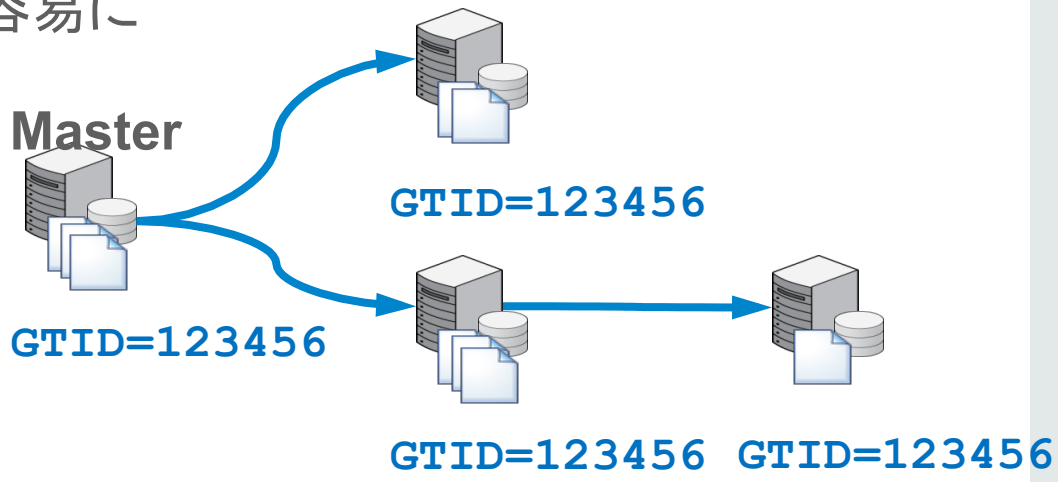
開発＆管理の簡素化

- 遅延レプリケーション
- リモートからのバイナリログのバックアップ
- ログへのメタデータの追加

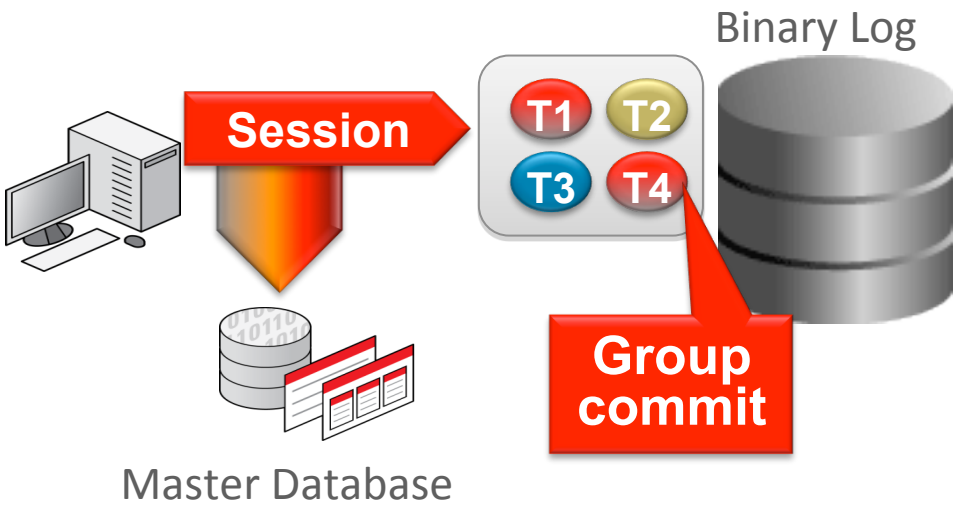
ORACLE

MySQL 5.6: グローバルトランザクションID

- レプリケーション環境でのトランザクションの追跡/比較が容易に可能
  - トランザクションを一意に識別できる識別子をバイナリログに記録
- フェイルオーバーのために、最も最新のスレーブを自動認識
- 多段構成のレプリケーションが容易に



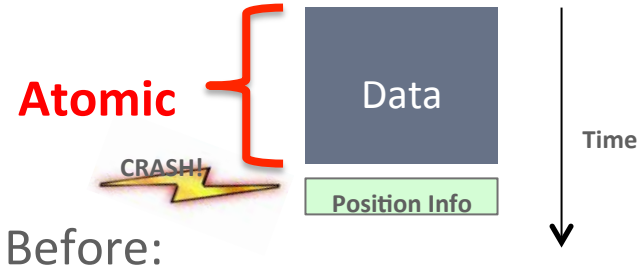
MySQL 5.6: バイナリログのグループコミット



- 複数トランザクションの情報をまとめてバイナリログに記録
- Sync_binlog=1の場合のオーバーヘッドを削減
- バイナリログコミット部分のソースをリファクタし、よりメンテナンスしやすいソースコードに改善

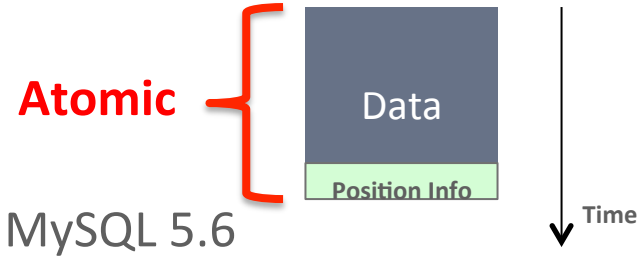
ORACLE

クラッシュセーフなスレーブ



Before:

- Transaction Data: **in tables**
- Replication Info: **in files**



MySQL 5.6

- Transaction Data: **in tables**
- Replication Info: **in tables**

- スレーブクラッシュ時の自動的なリカバリ
  - ポジション情報とデータは一貫性がある
- 管理者の介入なしにレプリケーションを再開可能
  - 最後にコミットしたイベントに自動的にロールバック

データの損失や破損のリスクを排除

Enhancements in MySQL 5.7

MySQL 5.6 GA以降の改良点

5.7.2 DMR

- マスタでの送信とバイナリログ書き込みスレッドの並列実行
- タイムスタンプベースでのマルチスレッドスレーブ
- パフォーマンススキーマでのレプリケーション関連性能統計情報
- “Lossless”準同期レプリケーション

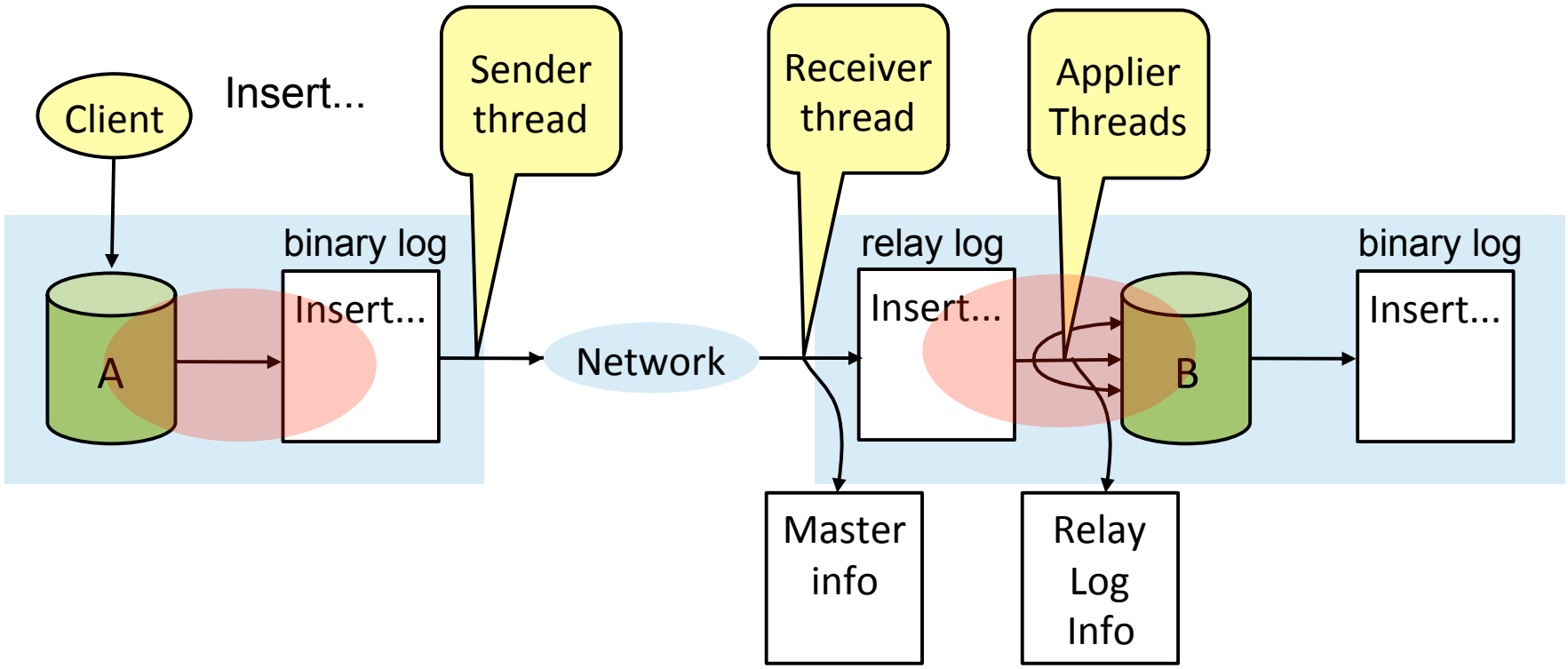
5.7.3 DMR

- mysqlbinlogコマンドへのSSLオプション追加
- スレーブでのフィルタリングの動的な変更
- 準同期レプリケーションにて複数のスレーブからの応答を待つ

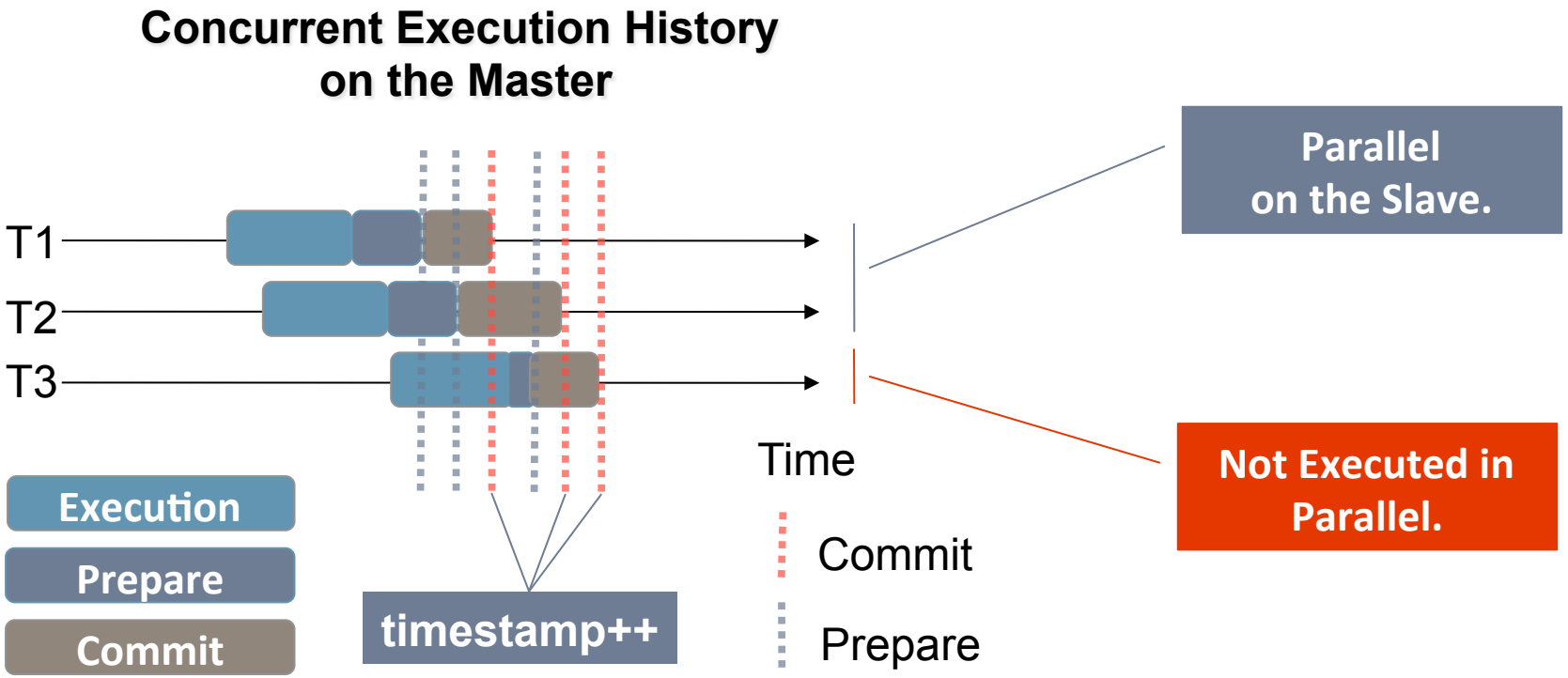
5.7.4 DMR

- マスターの送信スレッド用バッファの改良
- 準同期レプリケーションのマスターでのACK受信専用スレッド
- SQLスレッドを動作させたままのマスター切り替え

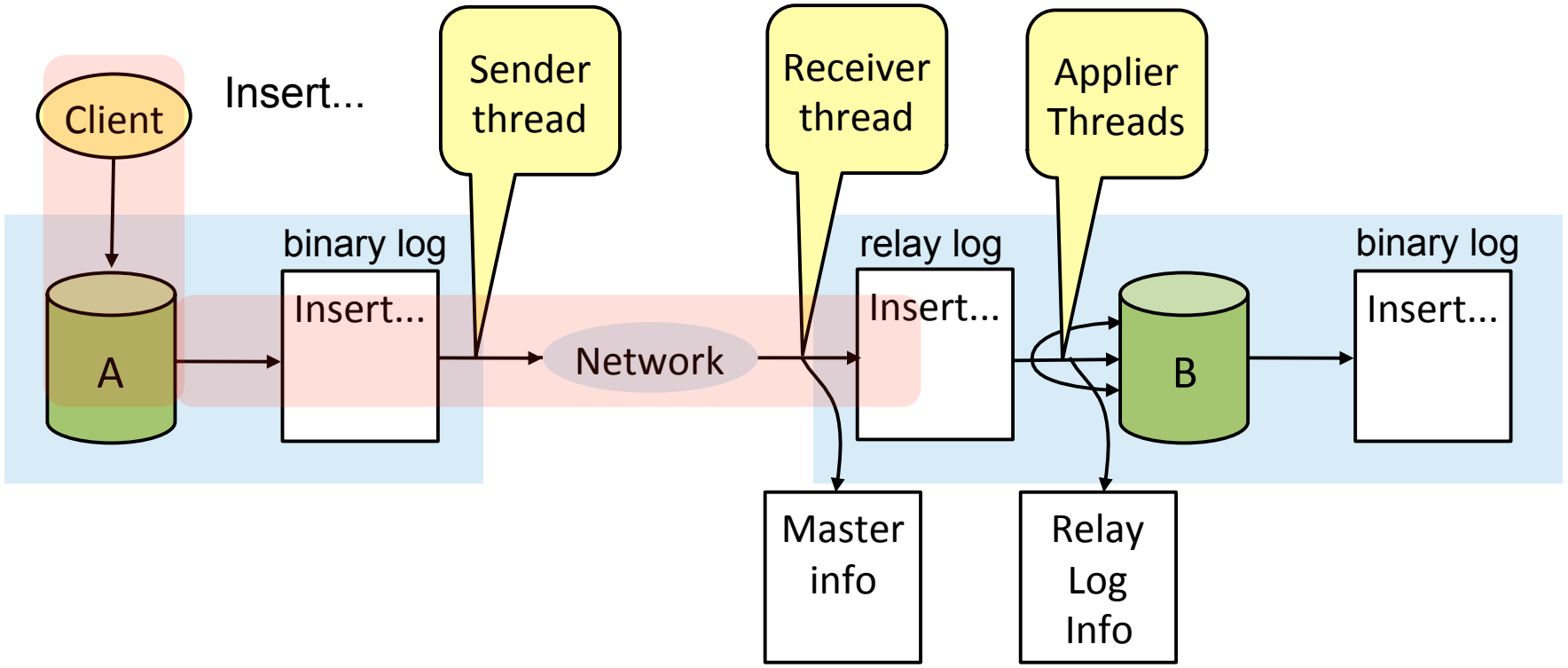
Higher Slave Throughput: Timestamp based Multi-threaded Applier



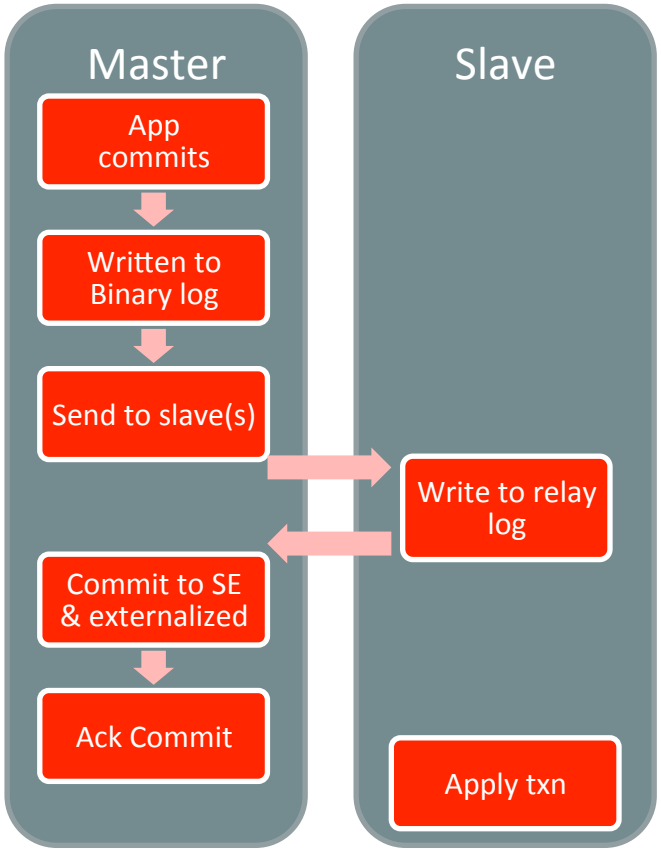
Higher Slave Throughput: Timestamp based Multi-threaded Applier



Loss-less Semi-sync Replication



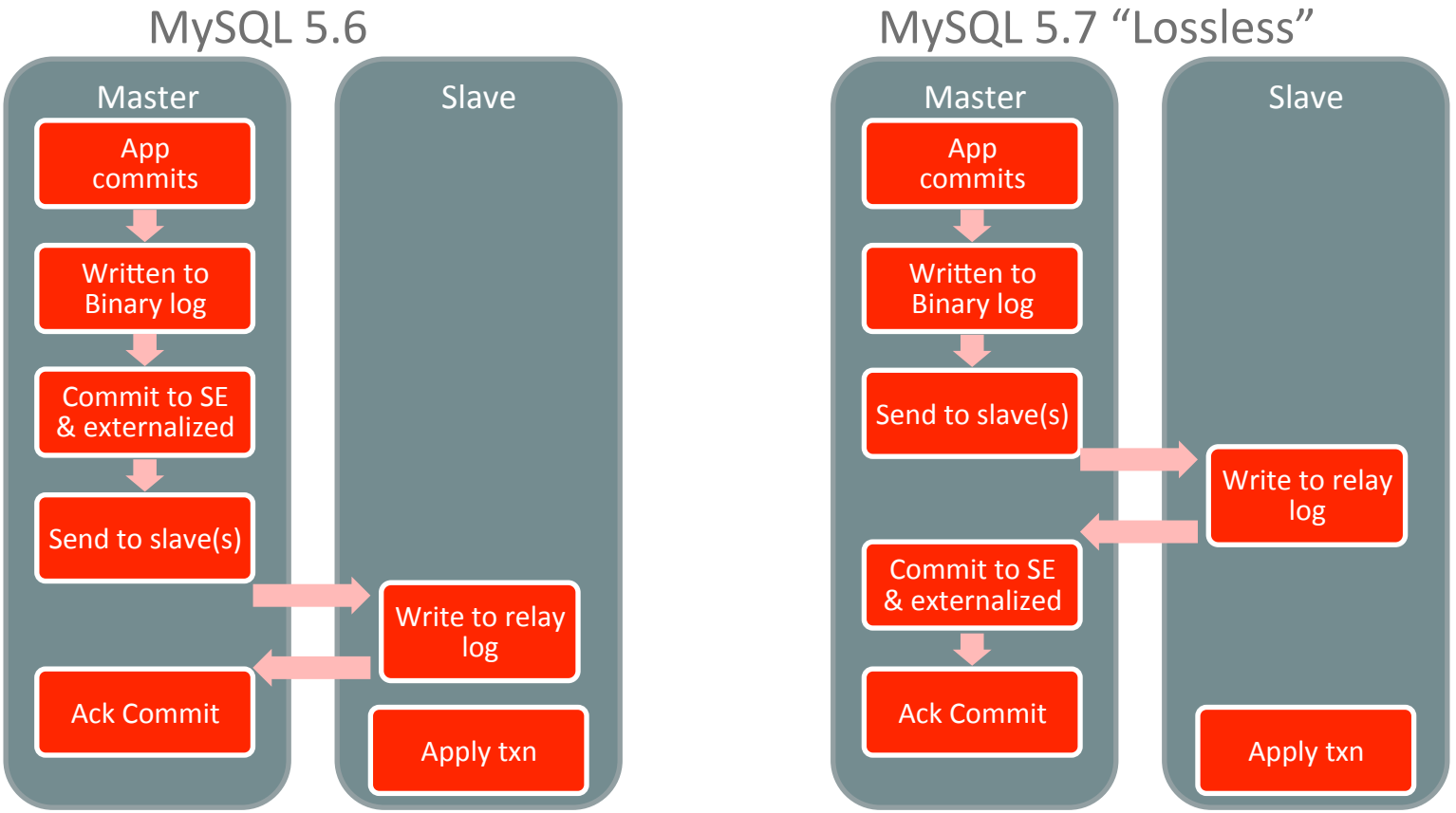
“Lossless”準同期レプリケーション



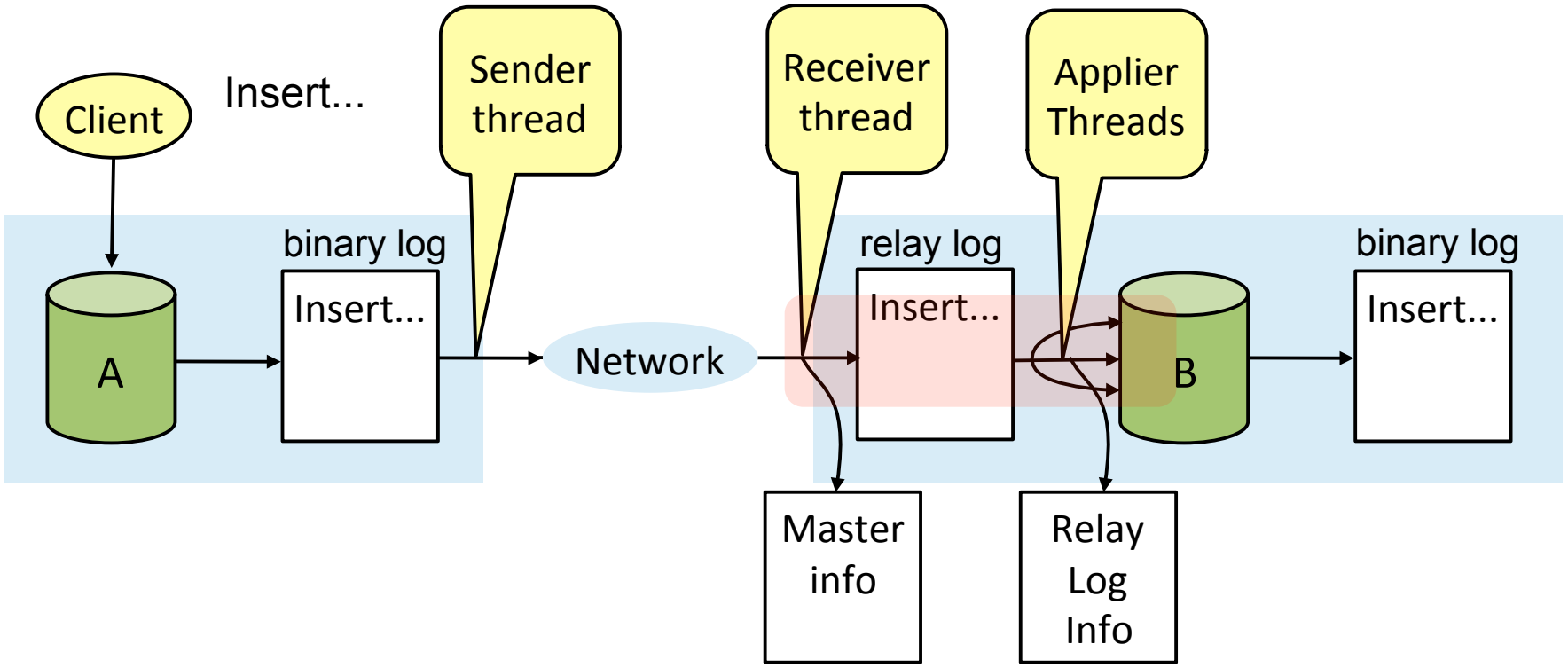
- マスターは指定のスレーブがトランザクションを受信するまで以下の処理を待つ
  - ストレージエンジンへのコミット
  - 他のクライアントから変更へのアクセス
  - アプリケーションへのコミットの応答
- スレーブが変更点を反映させるまでは待たない
  - 遅延を最小化
- スレーブに安全にコピーされるまで他のトランザクションが新しいデータを変更できないように

ORACLE

準同期レプリケーション



Dynamic Slave Replication Filters



Dynamic Slave Replication Filters

- Change Slave's Replication Filters dynamically.
  - No need to stop and restart slave for establishing new replication filtering rules.
  - All slave filters are supported.
  - Values can be input in various character sets.

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

MySQL 5.6 GA以降の改良点

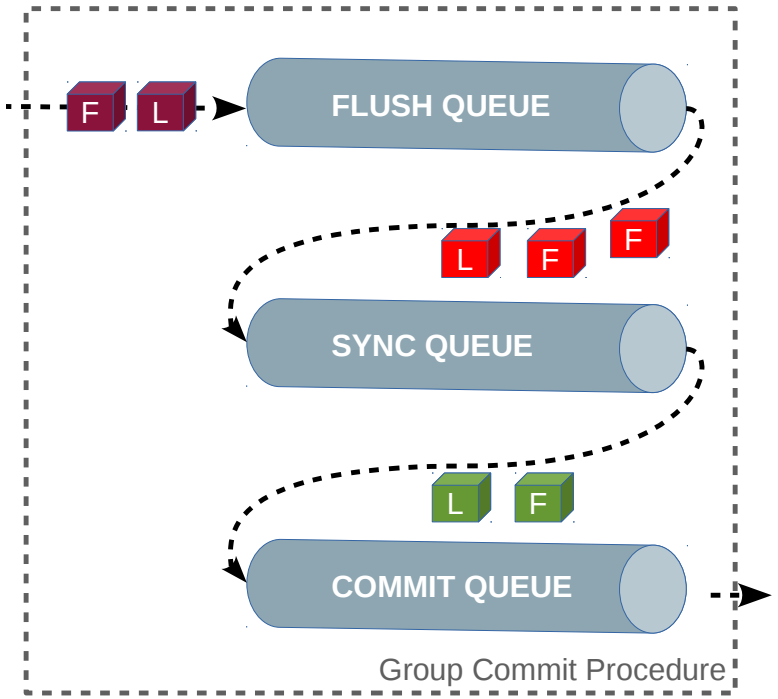
MySQL 5.7.5 - Development Milestone Release, September 2014

- More options to tune Binary Log Group Commit
  - Improving performance of Master and more parallelism on Slaves
- Storing GTID into MySQL system table
  - Not in log files

The Binary Log Group Commit

- MySQL 5.7.5 has more options to tweak the pipeline.
  - binlog_group_commit_sync_delay
    - To introduce an artificial delay in the SYNC stage.
  - binlog_group_commit_sync_no_delay_count
    - To stop waiting if delay_count was reached.

The Binary Log Group Commit



- If the transaction incoming rate is moderate.
- If there are many small groups being fsync'ed.
- TIPS:

```
SET binlog_group_commit_sync_delay
> 0
```

- Artificially delays the leader thread on the SYNC stage, to increase the chance to group more flushes together before fsyncing.
- By **tuning BGC** on the **master**, we get **more parallelism** on the **slave**.

ORACLE

Storing GTIDs In a Table

- GTIDs are saved into a mysql system table.

```
CREATE TABLE gtid_executed(
   source_uuid CHAR(36) NOT NULL,
   interval_start BIGINT NOT NULL,
   interval_end BIGINT NOT NULL,
   PRIMARY KEY(source_uuid, interval_start)
 );
```

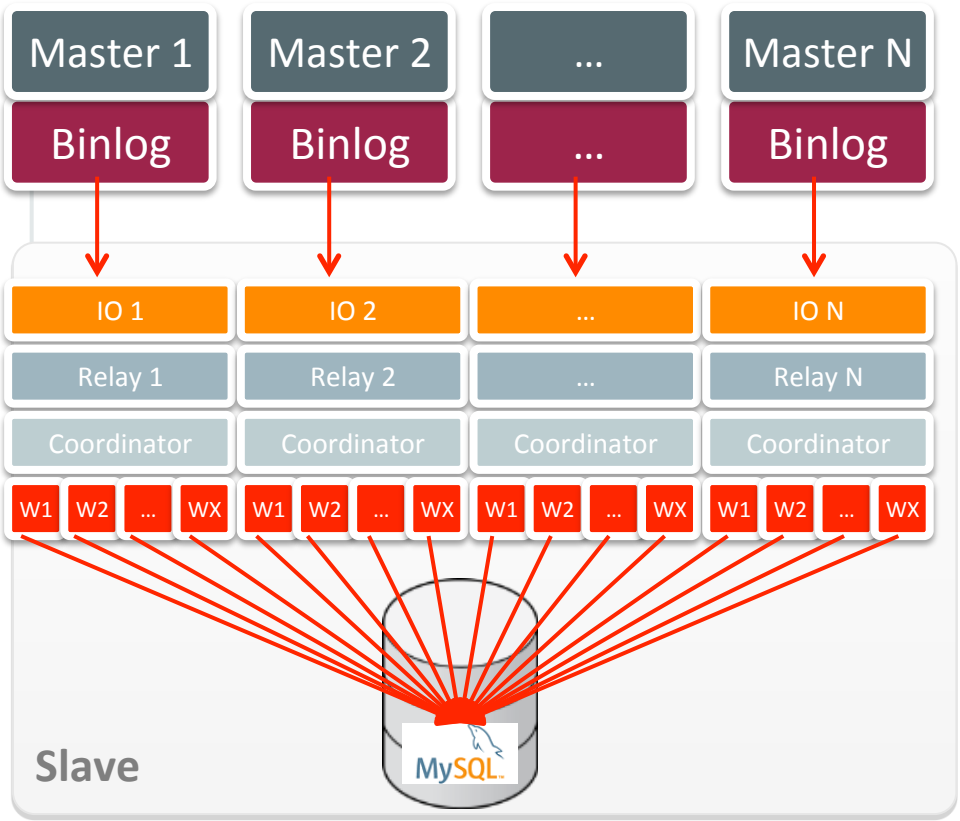
- GTIDs are inserted into the table as transactions commit.
- A compression thread runs periodically and compress the table into ranges.
  - New global variable controls the period: executed_gtids_compression_period.

```
mysql> SET GLOBAL executed_gtids_compression_period= N;
```

MySQL 5.7: Multi-Source Replication

labs.mysql.com

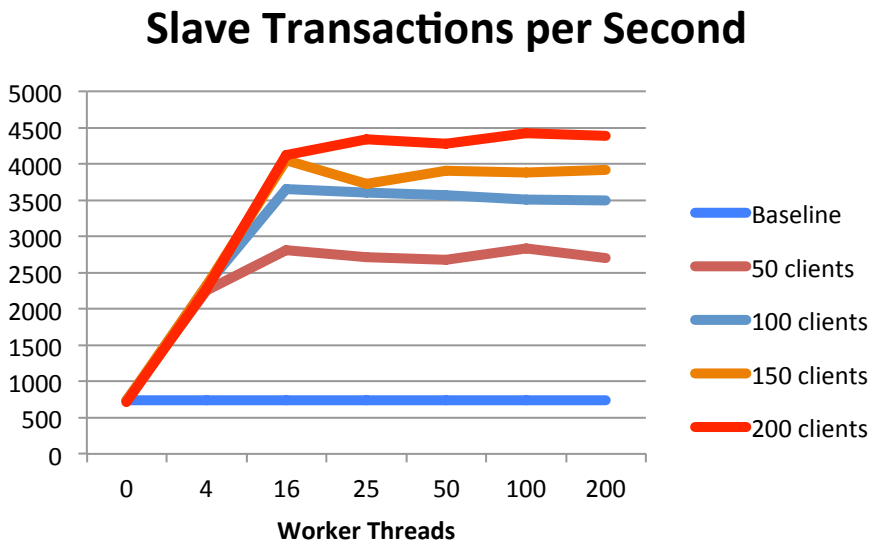
- 複数のマスターでの変更点を1台のスレーブに集約
  - 全てのシャードのデータを集約
  - より柔軟なレプリケーション構成
  - バックアップ処理を集約
- 準同期レプリケーション&改良版マルチスレッドスレーブ対応
- スレーブ側でのフィルタリング可能



MySQL 5.7: スキーマ内マルチスレッドスレーブ

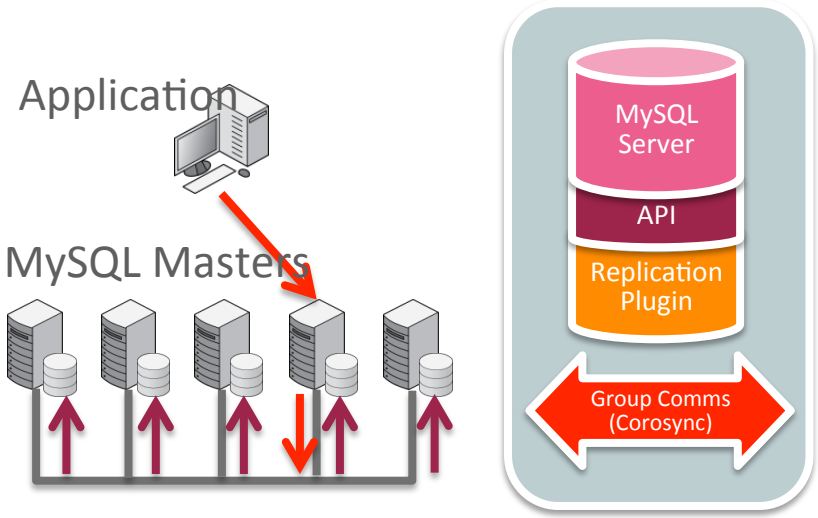
labs.mysql.com

- シングルスレッドのスレーブと比較して 5倍 のスループット
  - アプリケーション側での変更不要
  - バイナリログのグループコミットでの遅延を伴う操作不要
- GTID & クラッシュセーフスレーブ利用
- Sysbench OLTP test
  - 1,000万行
  - SSD / 48 core HT / 512 GB RAM



MySQL 5.7: グループレプリケーション

labs.mysql.com



- シェアード・ナッシング型”疑似”同期レプリケーション
- 更新はマルチ・マスタ型でどこでも可能
  - 矛盾の検知と解決(トランザクションのロールバック)
  - “Optimistic State Machine” レプリケーション
- グループメンバーの管理と障害検知を自動化
  - サーバのフェールオーバー不要
  - 構成の拡張/縮小の柔軟性
  - 単一障害点無し
  - 自動再構成
- 既存構成との統合
  - InnoDB
  - GTIDベースのレプリケーション
  - PERFORMANCE_SCHEMA

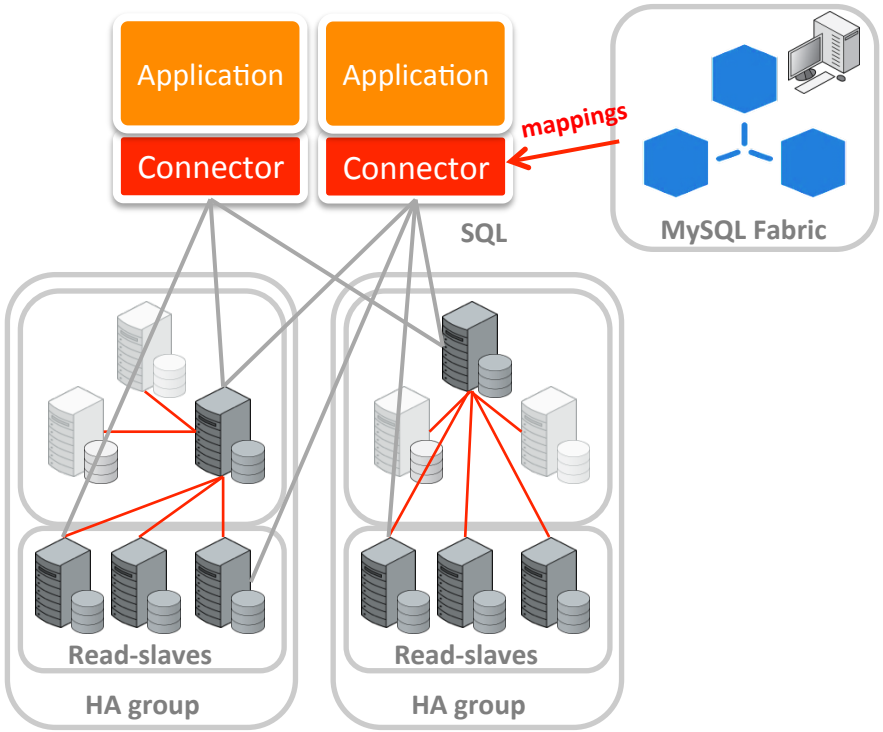
ORACLE

MySQL Fabric

for high availability and scalability with MySQL Replication

MySQL Fabric 1.5: 高可用性 & シャーディング

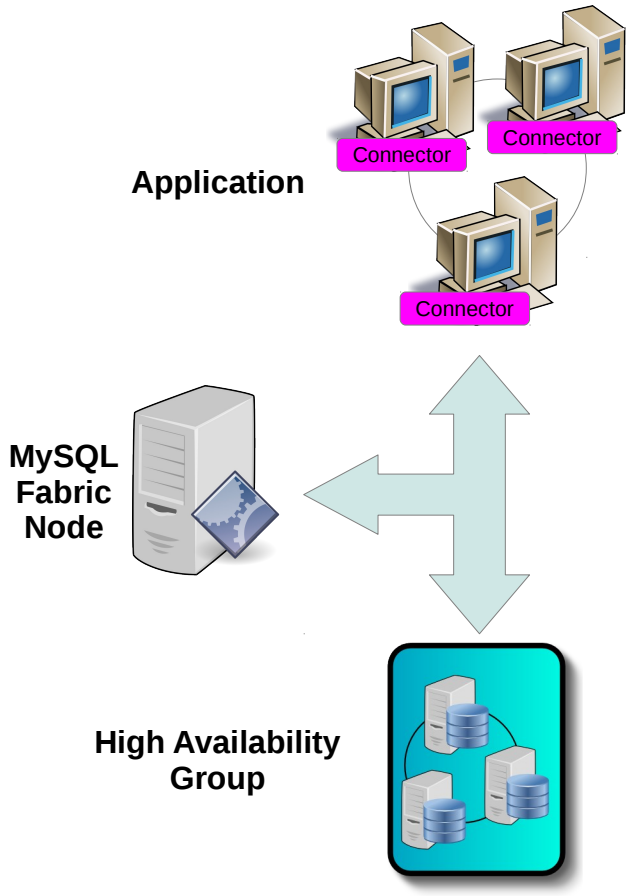
GA



- OpenStack との統合
- 高可用性
  - サーバの監視; スレーブの自動昇格と透過的なレプリケーション切り替え
- シャーディングによる拡張性
  - アプリケーションがシャードのキーを提供
    - 整数型、日付型、文字列型
  - レンジまたはハッシュ
  - シャード再構成可能
- Fabric対応コネクタ利用: Python, Java, PHP, .NET, C (labs)
  - プロキシを使わないので低レイテンシ、ボトルネック無し

High-Level Components

- Fabric-aware Connectors
  - Python, PHP, and Java
  - Enhanced Connector API
- MySQL Fabric Node
  - Manage information about farm
  - Provide status information
- Execute procedures MySQL Servers
  - Organized in High-Availability Groups
  - Handling application data



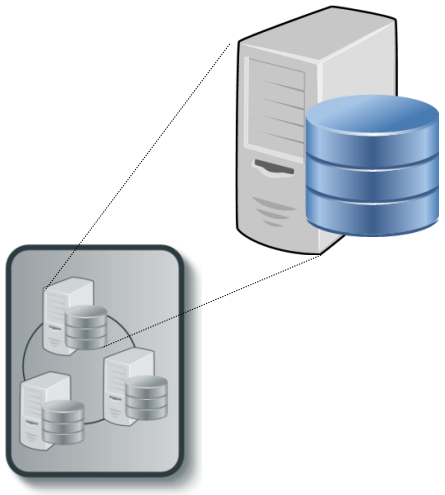
MySQL Fabric Configuration

- Backing Store
  - MySQL server
  - Persistent storage for state
  - Storage engine-agnostic
- Protocol
  - Address where node will be
  - Currently only XML-RPC
- Logging
  - Chatty: INFO (default)
  - Moderate: WARNING
  - URL for rotating log

```
[storage]
address = localhost:3306
user = fabric
password =
database = fabric

[servers]
user = fabric
password =

[protocol.xmlrpc]
address = localhost:32274
threads = 5
disable_authentication = yes

[logging]
level = INFO
url = file:///var/log/fabric.log
```

High-Availability Group Concept

- Abstract Concept
  - Set of servers
  - Server attributes
- Connector Attributes
  - Connection information
  - **Mode**: read-only, read-write, ...
  - **Weight**: distribute load
- Management Attributes
  - **State**: state/role of the server



State: Primary
Mode: Read-Write
Host: server-1.example.com

Create HA Groups and add Servers

- Define a group

```
mysqlfabric group create my_group
```

- Add servers to group

```
mysqlfabric group add my_group server1.example.com

mysqlfabric group add my_group server2.example.com
```

Create HA Groups and add Servers

- Promote one server to be primary

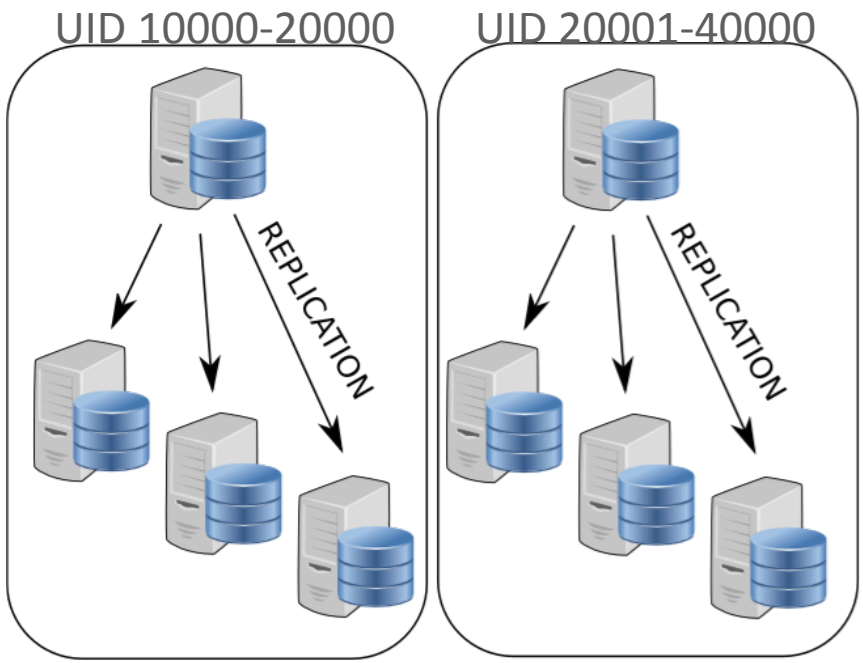
```
mysqlfabric group promote my_group
```

- Tell failure detector to monitor group

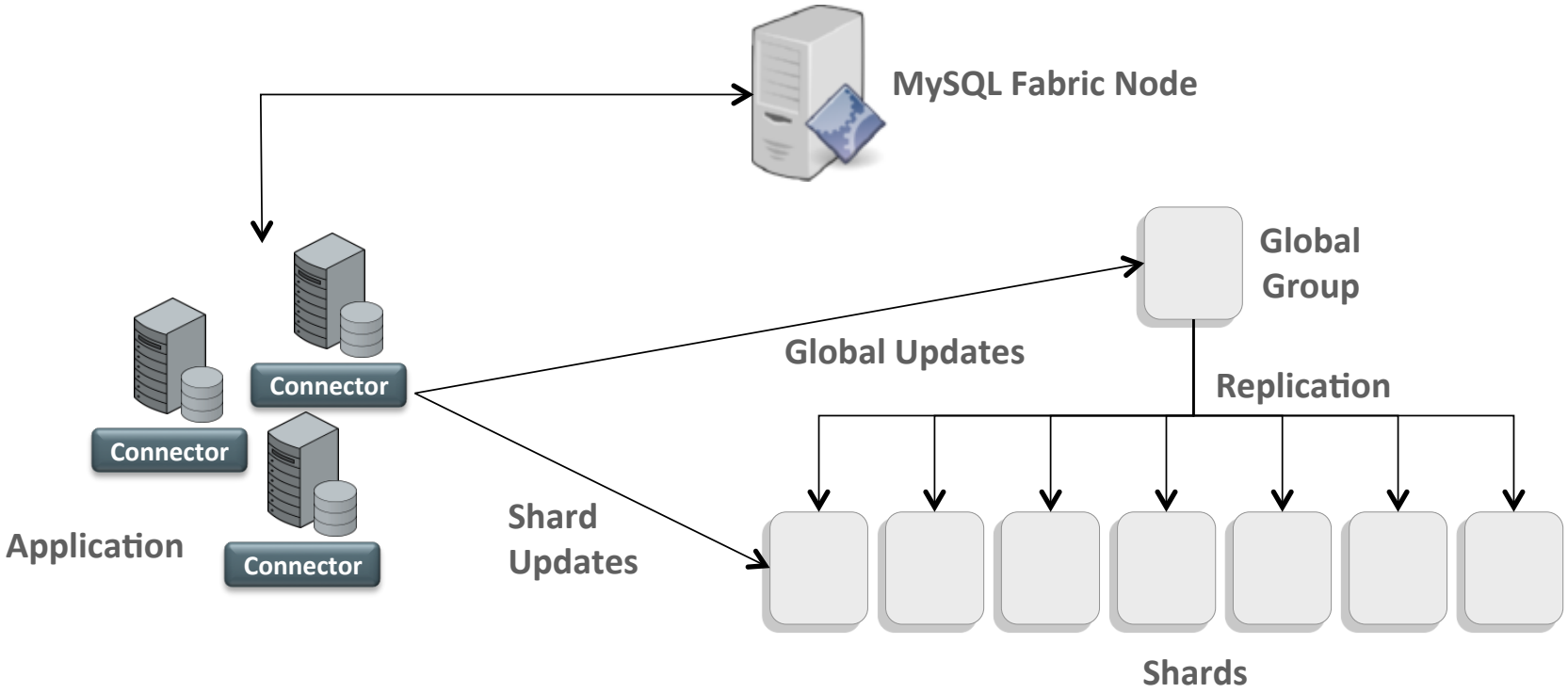
```
mysqlfabric group activate my_group
```

Benefits of Sharding

- Write scalability
  - Can handle more writes
- Large data set
  - Database too large
  - Does not fit on single server
- Improved performance
  - Smaller index size
  - Smaller working set
  - Improve performance (reads and writes)



Sharding Architecture



MySQL Fabric: Sharding Setup

- Set up some groups
  - **`my_global`** – for global updates
  - **`my_group.N`** – for the shards
  - Add servers to the groups
- Create a shard mapping
  - A “distributed database”
  - Mapping keys to shards
  - Give information on what tables are sharded
- Add shards

MySQL Fabric:

Moving and Splitting Shards

- Moving a shard (id=5) from existing group to another (my_group.8)

```
mysqlfabric sharding move 5 my_group.8
```

- Splitting a shard (id=5) into two parts with new half stored in group my_group.6

```
mysqlfabric sharding split 5 my_group.6
```

Connector API: Shard Specific Query

- Indicate tables to be used in query
  - **Property**: tables
  - Fabric will compute map
- Indicate read-only queries
  - **Property:** mode
- Provide sharding key
  - **Property**: key
  - Fabric will compute shard
- Joins within the shard (or with global tables) supported

```
conn.set_property(tables=["test.subscribers"], key=sub_no, mode=fabric.MODE_READONLY)
cur = conn.cursor()
cur.execute(
    "SELECT first_name, last_name FROM subscribers WHERE sub_no = %s", (sub_no)
    )
for row in cur:
    print row
```

Connector API: Global Update

- Set global scope
  - **Property**: scope
  - Query goes to global group

```
conn.set_property(tables=[], scope='GLOBAL')
cur = conn.cursor()
cur.execute("ALTER TABLE test.subscribers ADD nickname VARCHAR(64)")
```

MySQL Fabric: OpenStackとの連携

クラウド環境での運用効率化

- MySQL Fabric
  - 高可用性 & シャーディング
- マシンとMySQLのプロビジョニング
  - OpenStack Nova
  - 対応予定: Trove, AWS
- サーバのセットアップ
  - スレーブの複製
  - レプリケーションの設定

ORACLE

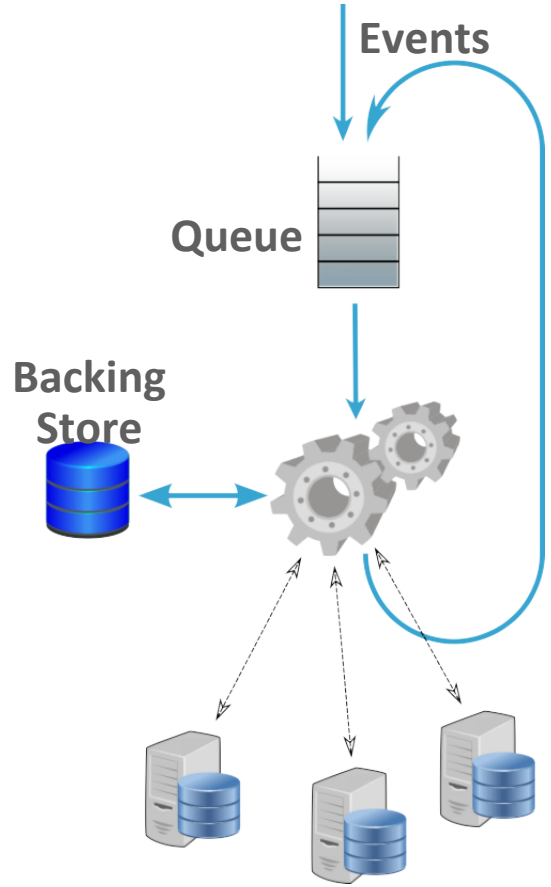
Server Provisioning – OpenStack Nova Integration

```
> mysqlfabric provider register
my_stack \
 my_user my_password \
 http://8.21.28.222:5000/v2.0/ \
 --tenant=my_user_role \
 --provider_type=OPENSTACK

> mysqlfabric machine create
my_stack \
 --image id=8c92f0d9-79f1-4d95-
b398-86bda7342a2d \
 --flavor name=m1.small

> mysqlfabric machine list my_stack
```

- Fabric creates new machines, & MySQL Servers
  - Initially using OpenStack Nova
  - Other frameworks on the way (OpenStack Trove, AWS,…)
- Server setup
  - Clones slave
  - Sets up replication
  - Performs custom operations

MySQL Fabric executor

- Event driven
  - Events will trigger execution of procedures
  - Procedures can trigger events themselves
  - Each step of a procedure is called a *job*
- Procedures
  - Written in Python
  - Interacts with servers
  - Write state changes into backing store
  - Lock manager for conflict resolution
    - Conservative two-phase locking strategy
    - Avoid deadlocks



ORACLE

MySQL Fabric – Current Limitations

- Routing is dependent on Fabric-aware connectors
  - Currently Java (+ Hibernate), PHP (+ Doctrine), Python, .NET & C (labs)
- MySQL Fabric node is a single (non-redundant process)
  - HA Maintained as connectors continue to route using local caches
- Establishes asynchronous replication
  - Manual steps to switch to semisynchronous
- Sharding not completely transparent to application (must provide shard key – column from application schema)
- No cross-shard joins or other queries
- Management is through CLI, MySQL protocol or XML/RPC API
  - No GUI

MySQLレプリケーション+MySQL Fabricの用途

スケーラビリティ

大規模なWebやクラウドで求められる**参照更新**処理の拡張性向上を実現

可用性

データの複製を持たせ、マスターに障害が発生した場合は**自動的に**切り替え

集計処理やバックアップ

オンラインでの処理から処理を分離することによって全体の性能を維持

クラウド対応

地理的冗長性のみでなく、クラウドへのMySQLサーバ群の動的な展開

ORACLE