ORACLE®

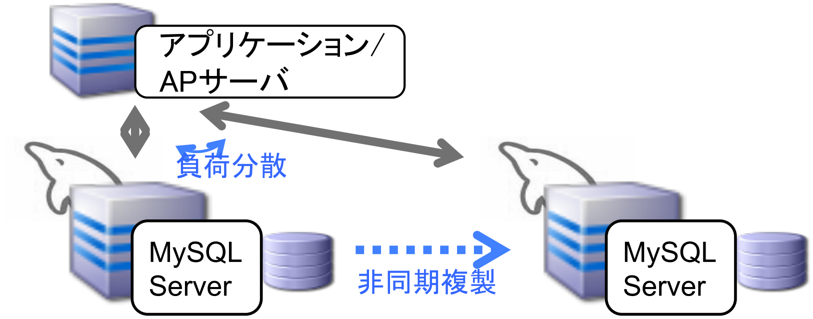
MySQL Replication & MySQL Fabric

MySQL Global Business Unit
Sales Consulting Manager, JAPAC
梶山 隆輔 / Ryusuke Kajiyama

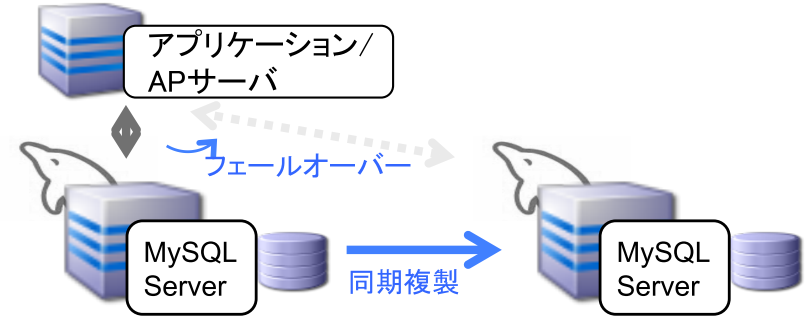
以下の事項は、弊社の一般的な製品の方向性に関する概要を説明するものです。また、情報提供を唯一の目的とするものであり、いかなる契約にも組み込むことはできません。以下の事項は、マテリアルやコード、機能を提供することをコミットメント（確約）するものではないため、購買決定を行う際の判断材料になさらないで下さい。オラクル製品に関して記載されている機能の開発、リリースおよび時期については、弊社の裁量により決定されます。

MySQLの高可用性構成

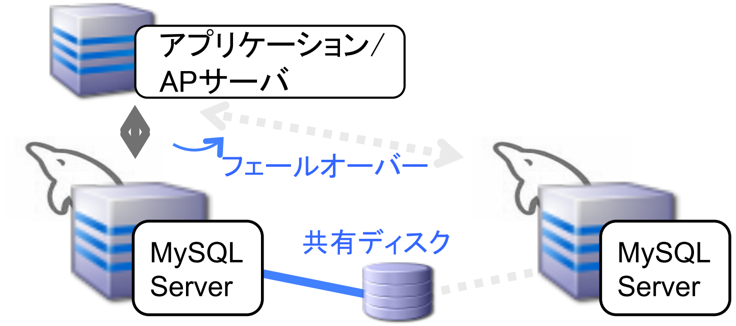
- レプリケーション（標準機能）
  非同期&準同期データレプリケーション



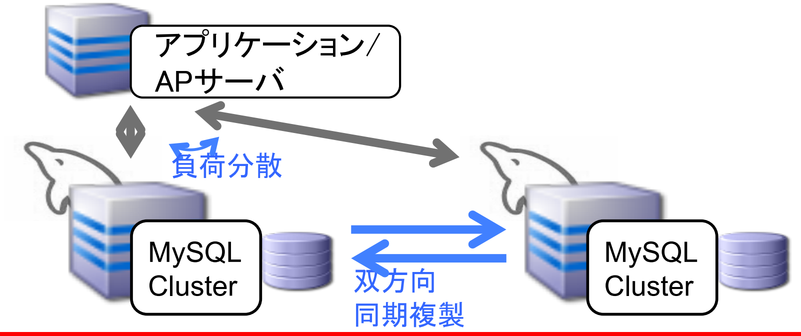
- MySQL+DRBD
  Linux用のノード間データコピー



- 3rdベンダ製HAソフト利用
  共有ディスクにデータを格納



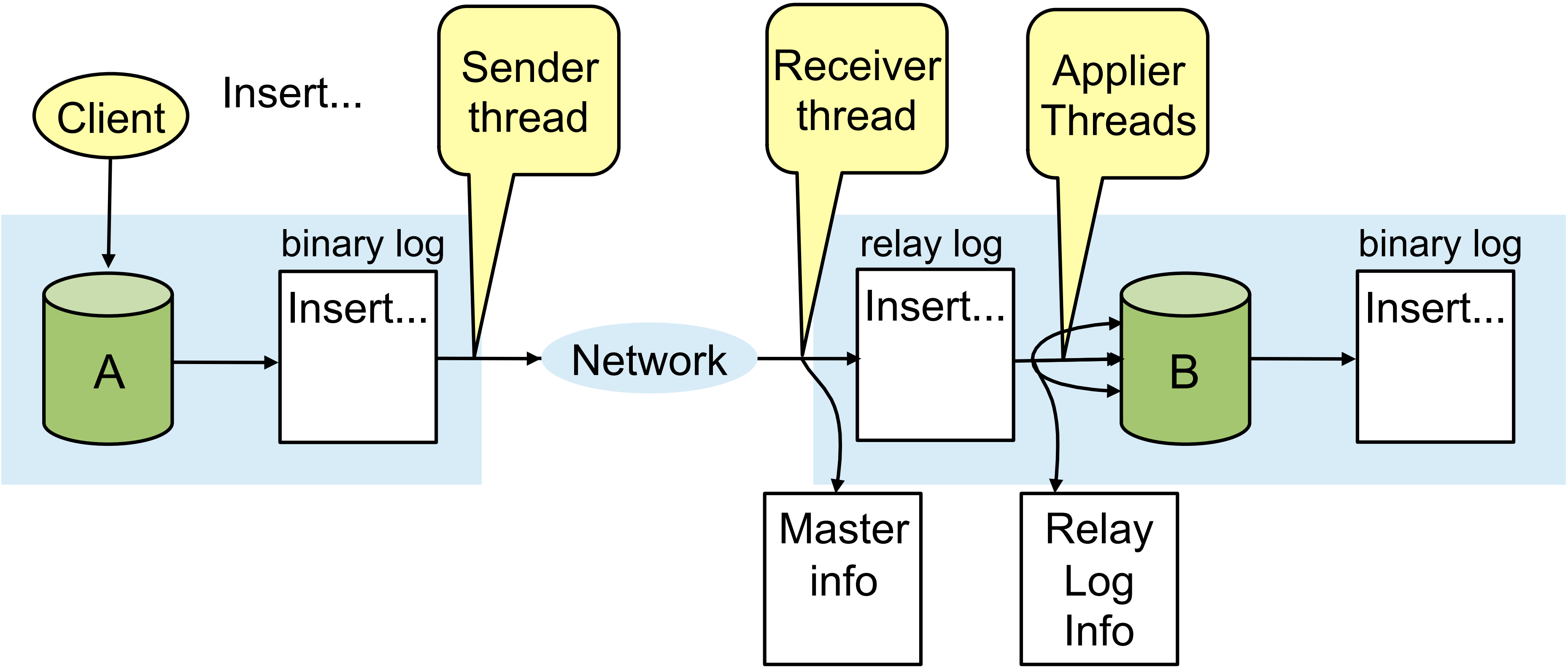
- MySQL Cluster
  シェアードナッシング型高性能クラスタ



MySQLの高可用性ソリューション

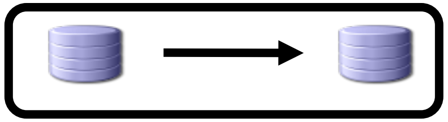
	MySQL 5.6 レプリケーション	Oracle VM Template	Solaris Cluster	Windows Cluster	DRBD	MySQL Cluster
自動フェールオーバー	✔	✔	✔	✔	✔	✔
データロス無し	Semi-Sync	✔	✔	✔	✔	✔
サポートOS	All	Linux	Solaris	Windows	Linux	All
クラスタモード	Master + Slaves	Active/ Passive	Active/ Passive	Active/ Passive	Active/ Passive	Multi-Master
共有ディスク	不要	必要	必要	必要	不要	不要
可用性デザイン	99.9%	99.99%	99.99%	99.95%	99.99%	99.999%
単一ベンダサポート	✔	✔	✔	✖	✔	✔

Background: Replication Components

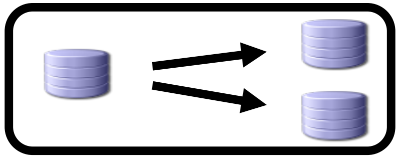


MySQL レプリケーションの構成パターン

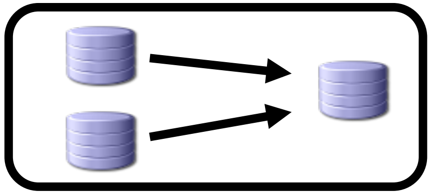
マスタ > スレーブ

✓ 

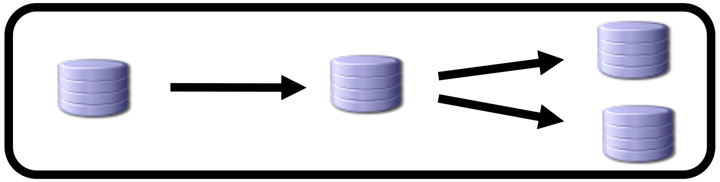
マスタ > マルチスレーブ

✓ 

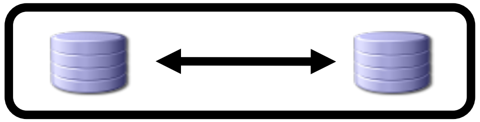
マルチマスタ > スレーブ (マルチソース)

⛔ 

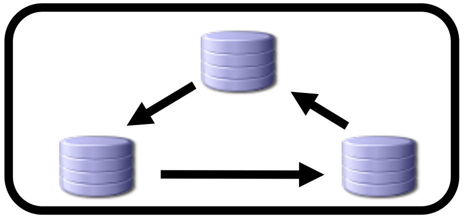
マスタ > スレーブ > マルチスレーブ

✓ 

マスタ < > マスタ (マルチマスタ)

⚠ 

循環型 (マルチマスタ)

⚠ 

Program Agenda

- MySQL 5.6での改良点
- MySQL 5.7での改良点
- MySQL Fabric

MySQL 5.6での改良点

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

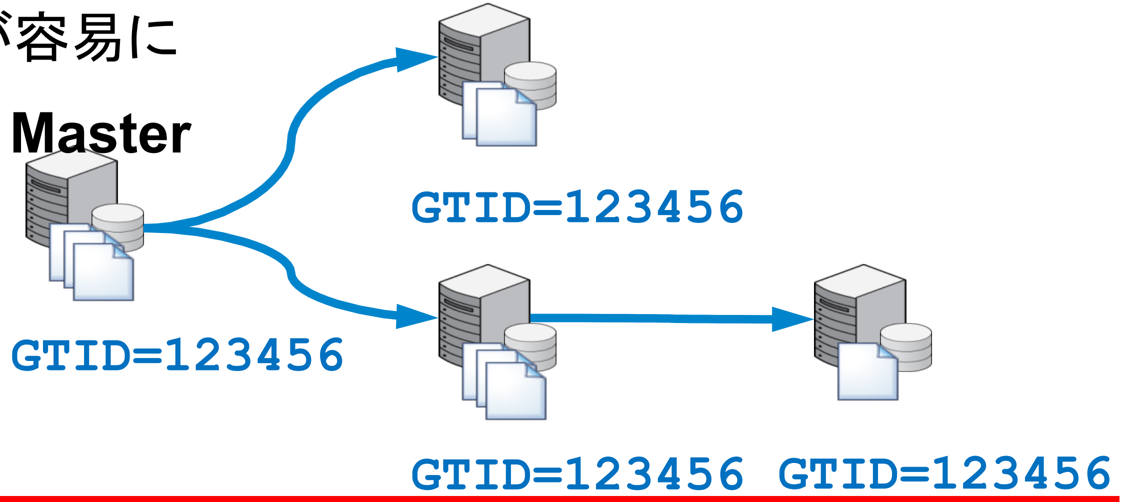
- レプリケーション チェックサム

開発 & 管理の簡素化

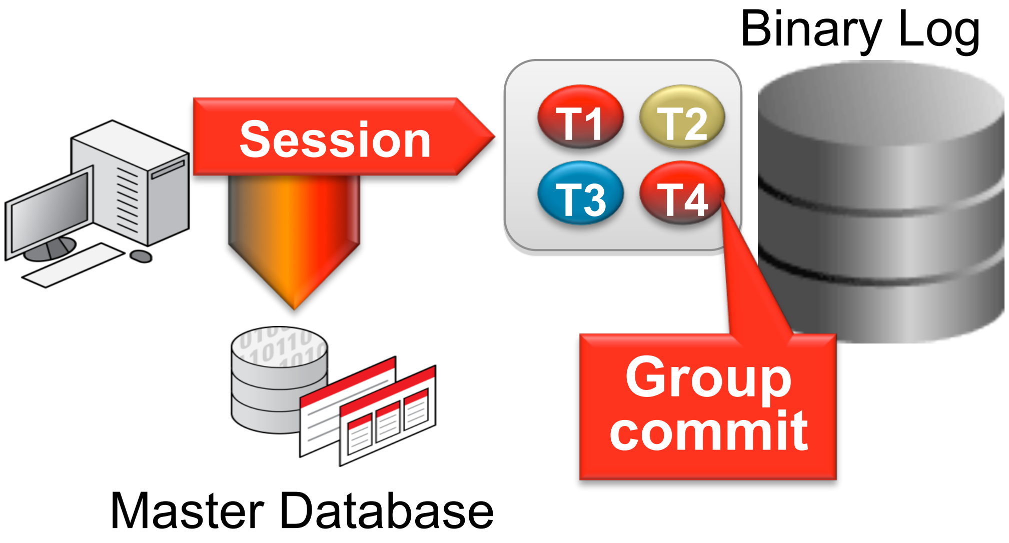
- 遅延レプリケーション
- リモートからのバイナリログのバックアップ
- ログへのメタデータの追加

MySQL 5.6: グローバルトランザクションID

- 複数台のレプリケーション環境でも容易にトランザクションの追跡/比較が可能
  - 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



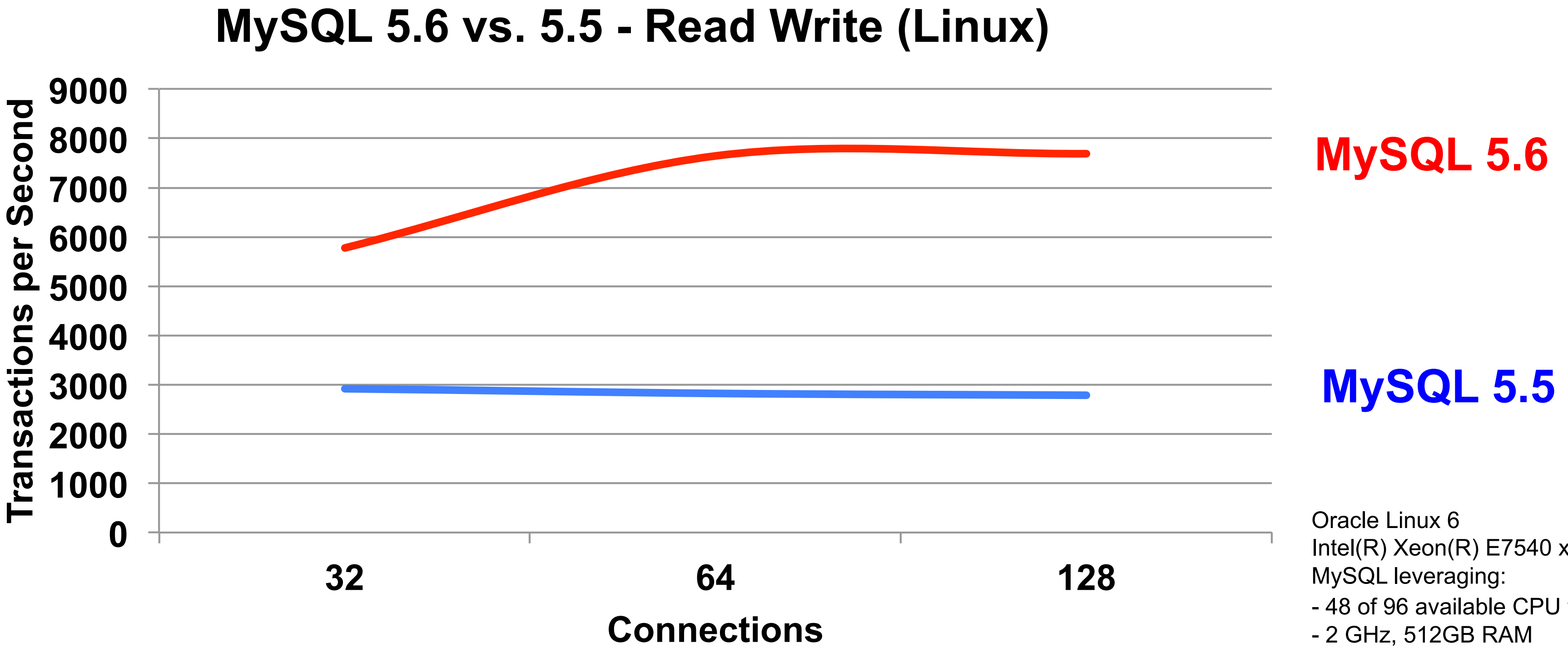
MySQL 5.6: バイナリログのグループコミット



- 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コードに改善

Binary Log Group Commit Performance

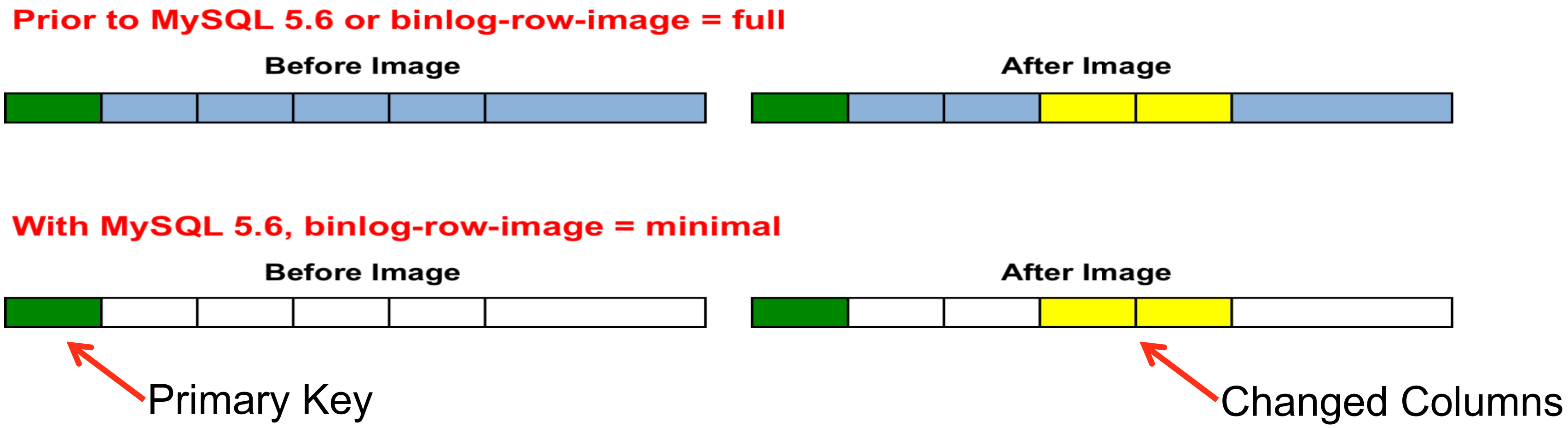
Binlog=1



Oracle Linux 6
Intel(R) Xeon(R) E7540 x86_64
MySQL leveraging:
- 48 of 96 available CPU threads
- 2 GHz, 512GB RAM

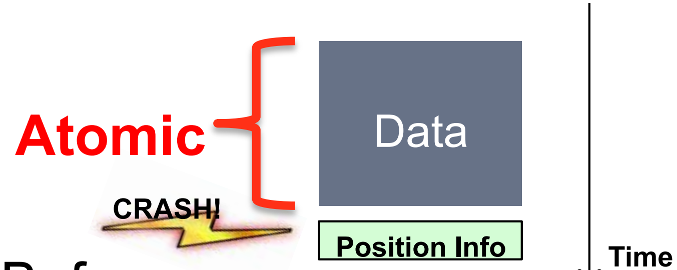
180% Performance Gain

MySQL 5.6: 行ベースレプリケーションの最適化


Prior to MySQL 5.6 or binlog-row-image = full
Before Image
After Image
With MySQL 5.6, binlog-row-image = minimal
Before Image
After Image
Primary Key
Changed Columns

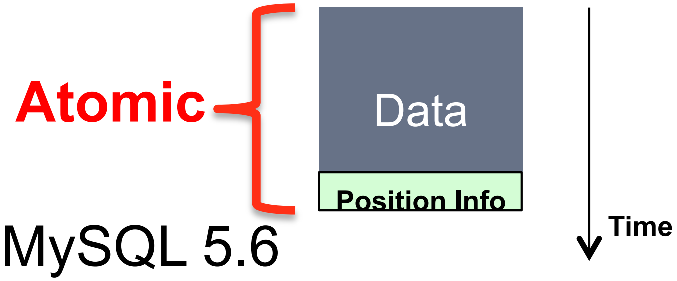
- 新しいオプション: `binlog-row-image=minimal`
- マスター/スレーブのスループットを向上
  - バイナリログのサイズ削減、メモリ使用量削減、ネットワーク転送量削減
- 変更された行イメージだけを保持

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

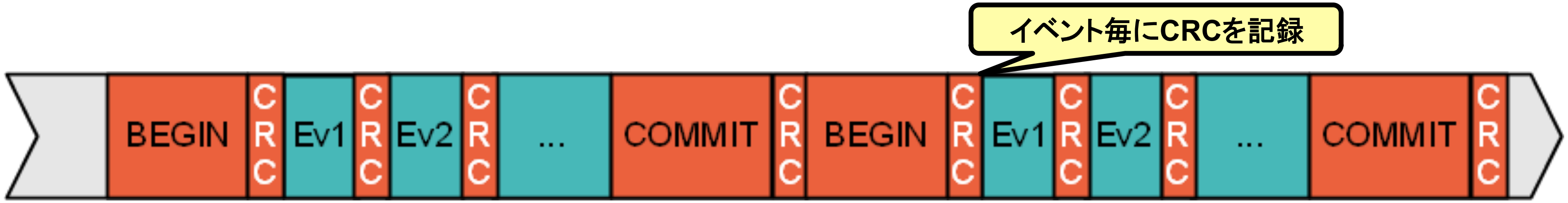
- Transaction Data: **in tables**
- Replication Info: **in tables**

- 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

チェックサムの追加

- バイナリログにチェックサムを記録可能(CRC32)
  - binlog-checksum オプションで有効/無効を制御
- マスターは、バイナリログにイベントを記録する時に、CRC32を生成
- 複数個所で誤り検出可能
  - マスターとスレーブでの誤り検出は、個別に有効/無効の設定が可能 (master-verify-checksum option、slave-sql-verify-checksum)



Binary Log File

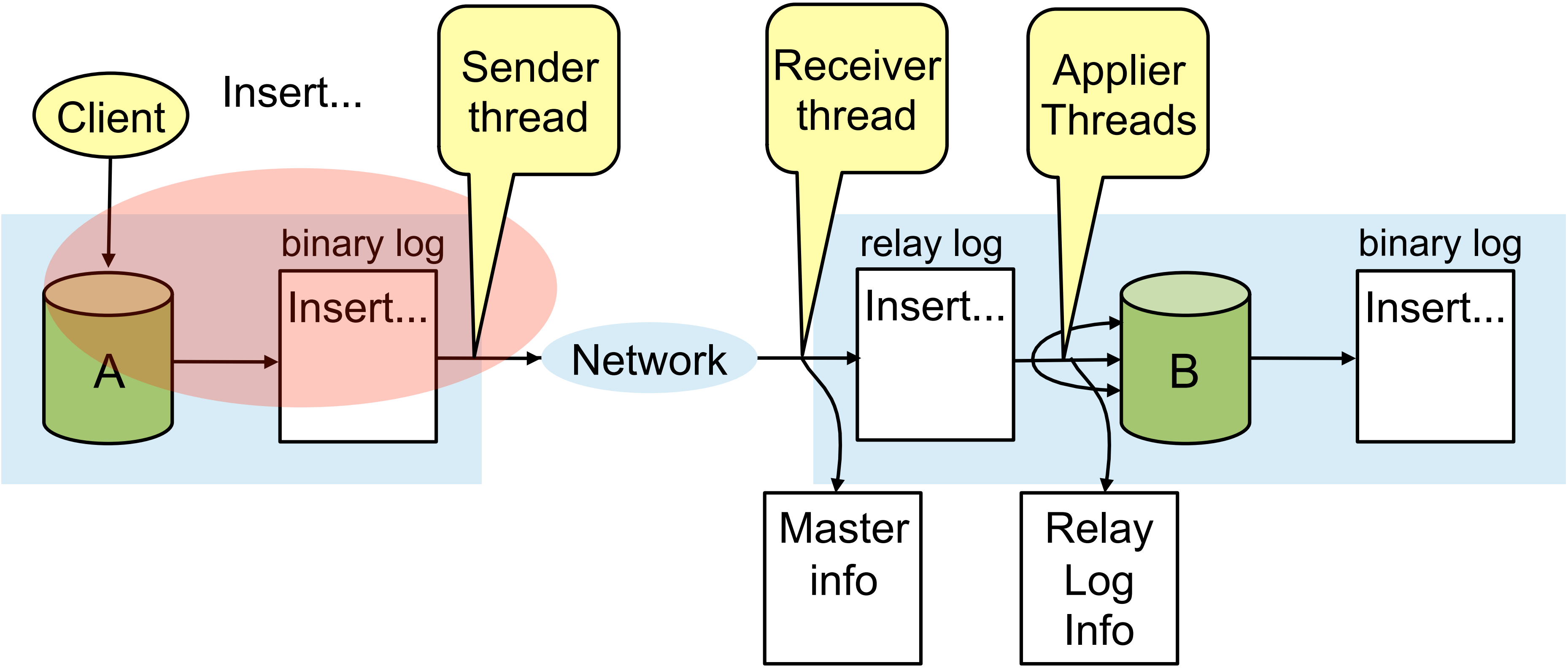
MySQL 5.7での改良点

MySQL 5.6 GA以降の改良点

MySQL 5.7.2 - Development Milestone Release, September 2013

- Higher Master Throughput
  - Sender, aka Dump, Thread Does Not Take the Binary Log Lock.
- Higher Slave Throughout
  - Multi-Threaded (Slave) Timestamp based Applier (MTS).
- Better Monitoring of Replication
  - Instrumentation for getting replication status through performance schema.
- Loss-less Semi-sync Replication.

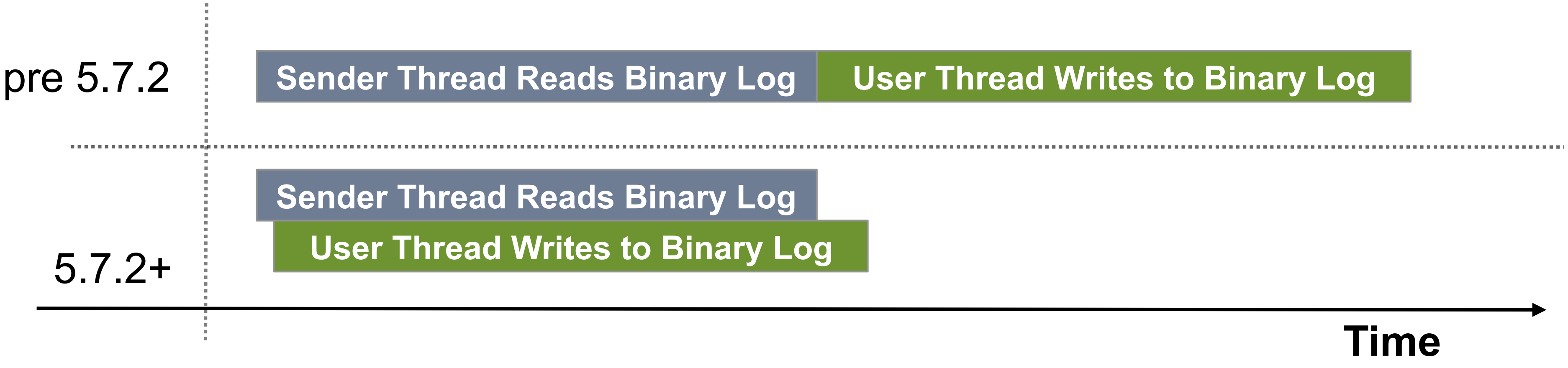
Higher Master Throughput: Sender, aka Dump, Thread Enhancement



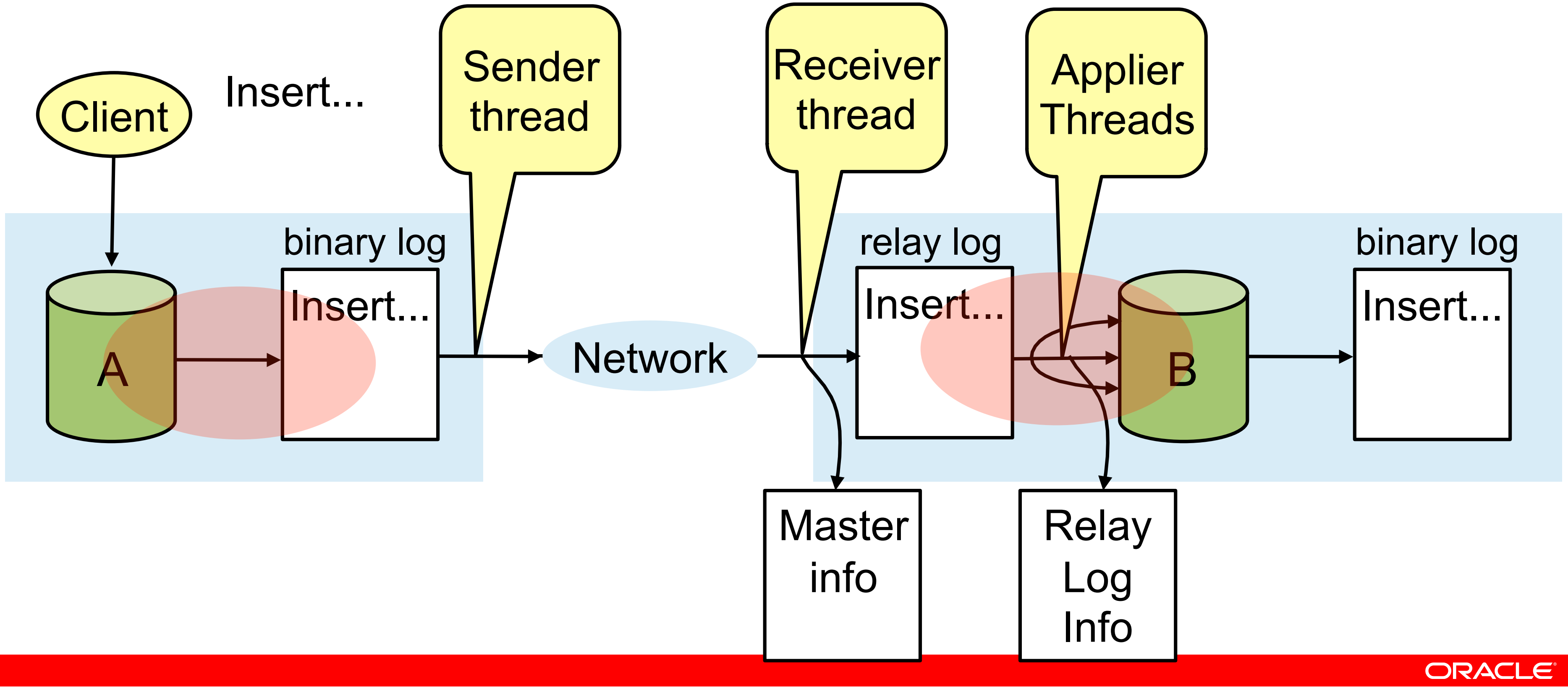
Higher Master Throughput: Sender, aka Dump, Thread Enhancement

Concurrent reads by the sender thread with ongoing writes from user threads.

- **Sender thread does not block user sessions more than necessary.**
- **Higher throughput for both sender threads and user sessions.**


pre 5.7.2
Sender Thread Reads Binary Log
User Thread Writes to Binary Log
Sender Thread Reads Binary Log
User Thread Writes to Binary Log
5.7.2+
Time

Higher Slave Throughput: Timestamp based Multi-threaded Applier

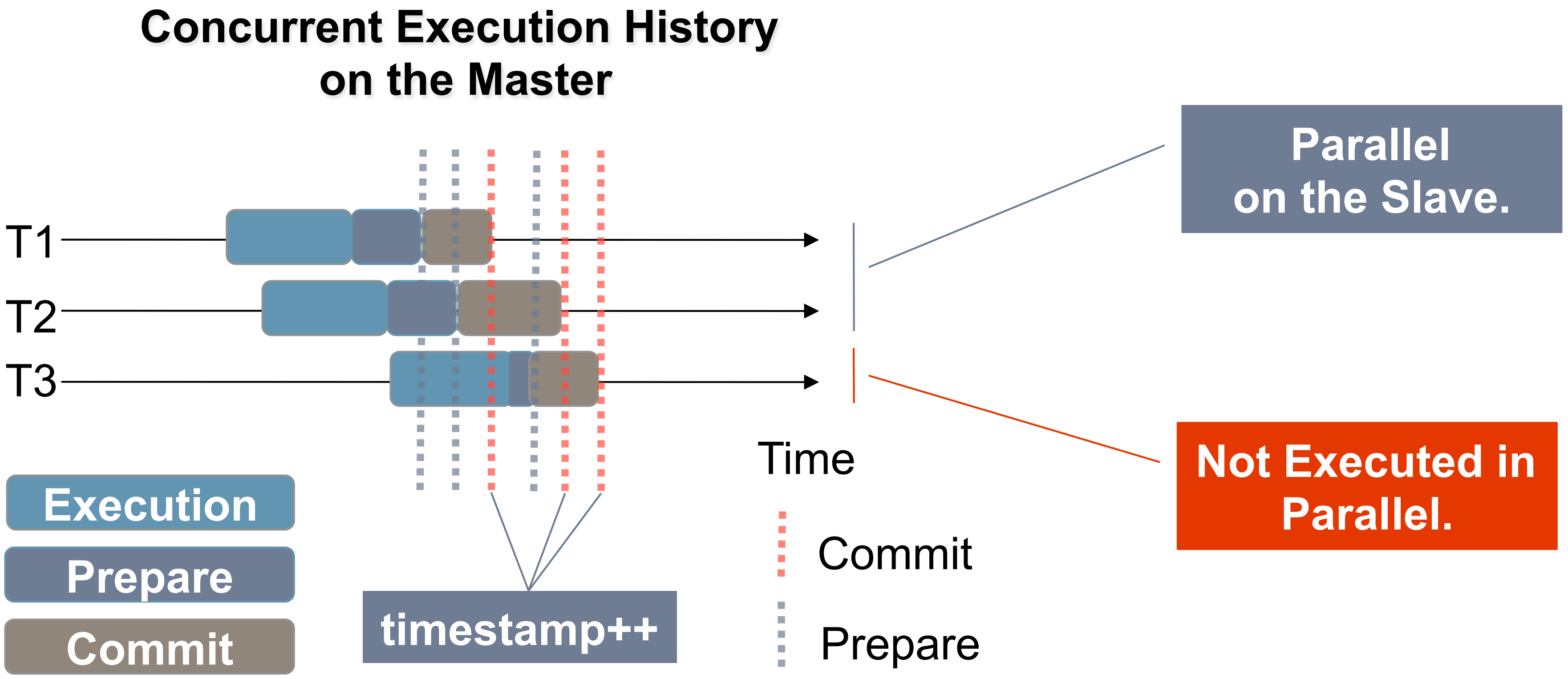


Higher Slave Throughput: Timestamp based Multi-threaded Applier

- Leverage parallelization information obtained from the execution on the master.
  - Transactions that prepare on the same “version” of the database, are assigned the same timestamp.

- Meanwhile, at the slave:
  - Transactions with the same timestamp can be executed in parallel;
  - Concurrent transactions commit independently, thus no waiting involved.

Higher Slave Throughput: Timestamp based Multi-threaded Applier



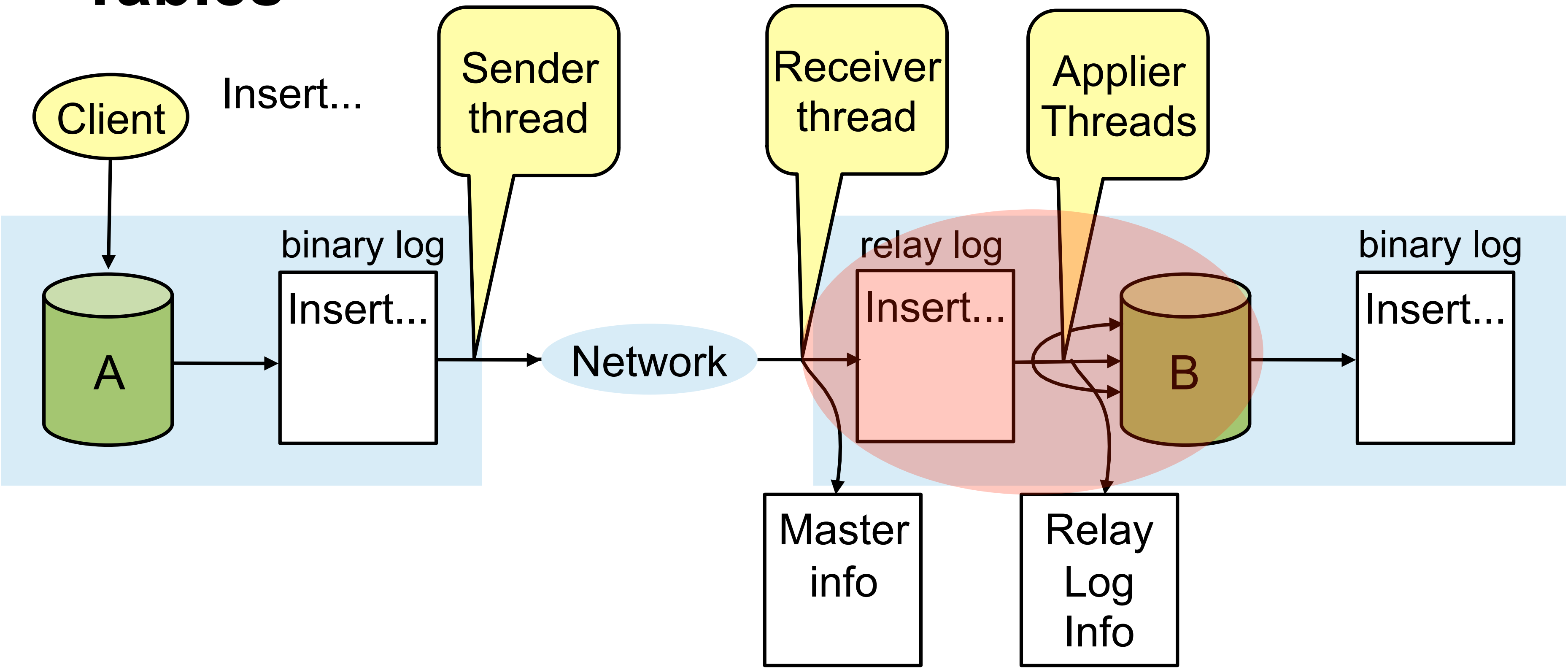
Higher Slave Throughput: Timestamp based Multi-threaded Applier

- Supports statement-based or row-based formats.
- Scheduling policy controlled through:

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

  - Logical_clock - means schedule based on the prepare timestamp
  - database - the scheduling policy from 5.6 (concurrency control done per database).
- Work to improve slave scalability continues, does not stop here.

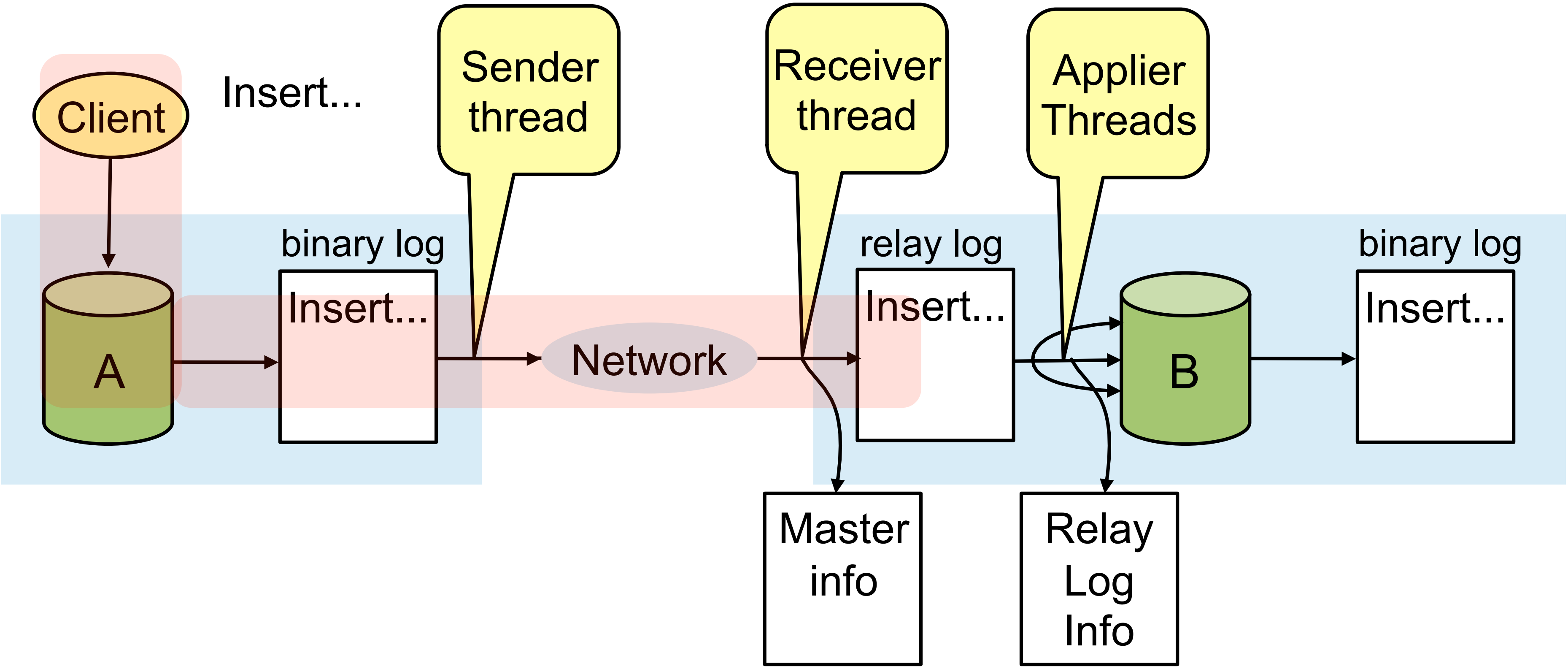
Better Replication Monitoring: P_S Replication Tables



Better Replication Monitoring: P_S Replication Tables

- Access monitoring information through an SQL interface.
- Write stored functions or procedures with input from replication internals.
- Logically unrelated information into different places.
- Flexible and easier to extend and adapt as new feature get into the server.
- More user friendly names identifying the monitoring fields.

Loss-less Semi-sync Replication

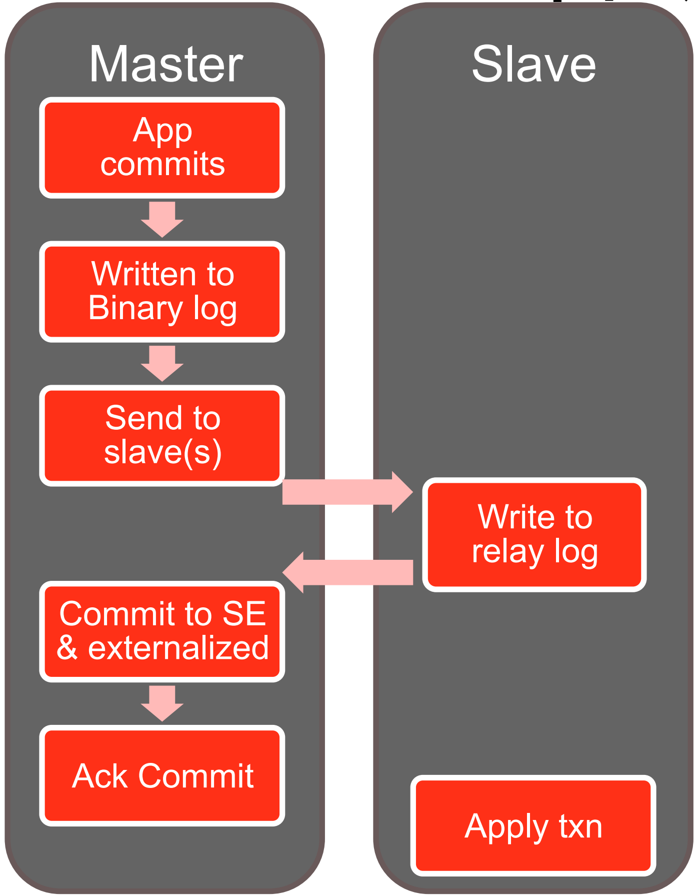


Loss-less Semi-sync Replication

- Master does not commit transaction until it receives an ACK from a slave.
  - (as opposed to: Master does not release the session until it receives an ACK from the slave.)
  - Therefore, concurrent transactions do not externalize changes while waiting for ACK.
- Should a master fail, then any transaction that it may have externalized is also persisted on a slave.
- User can choose between the original semisync behavior and the new one.

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

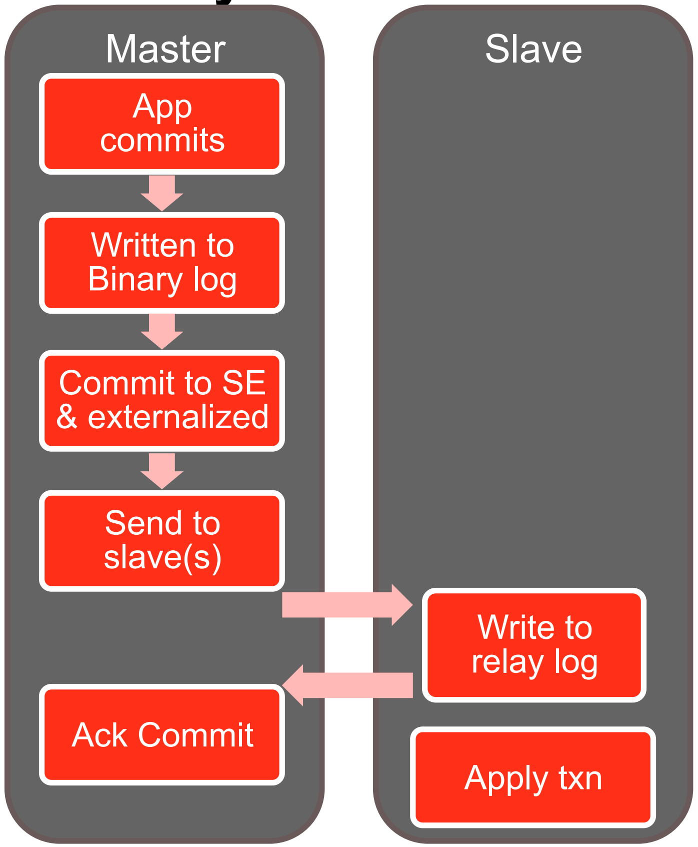
“Lossless”準同期レプリケーション



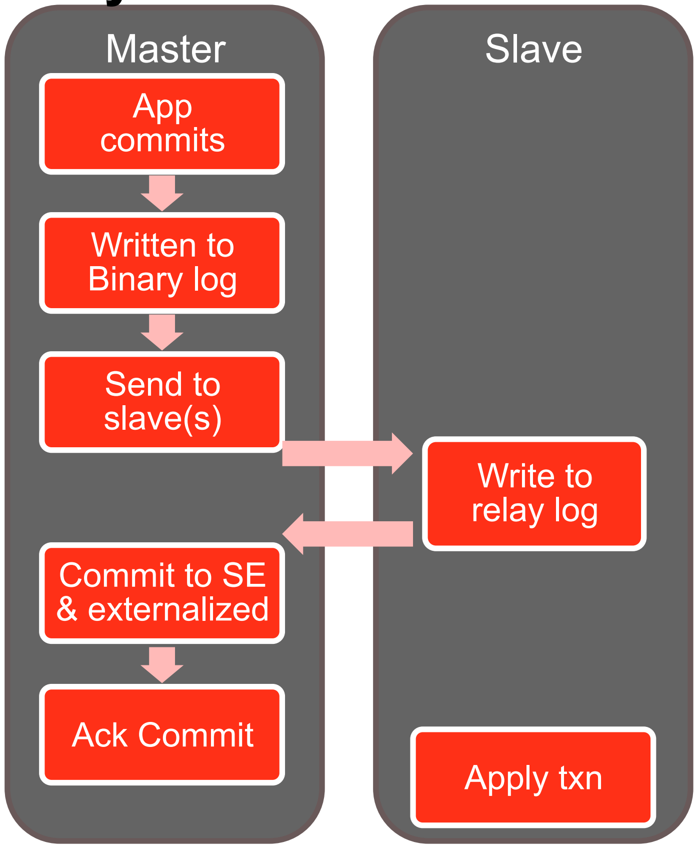
- マスターは指定のスレーブがトランザクションを受信するまで以下の処理を待つ
  - ストレージエンジンへのコミット
  - 他のクライアントから変更へのアクセス
  - アプリケーションへのコミットの応答
- スレーブが変更点を反映させるまでは待たない
  - 遅延を最小化
- スレーブに安全にコピーされるまで他のトランザクションが新しいデータを変更できないように

準同期レプリケーション

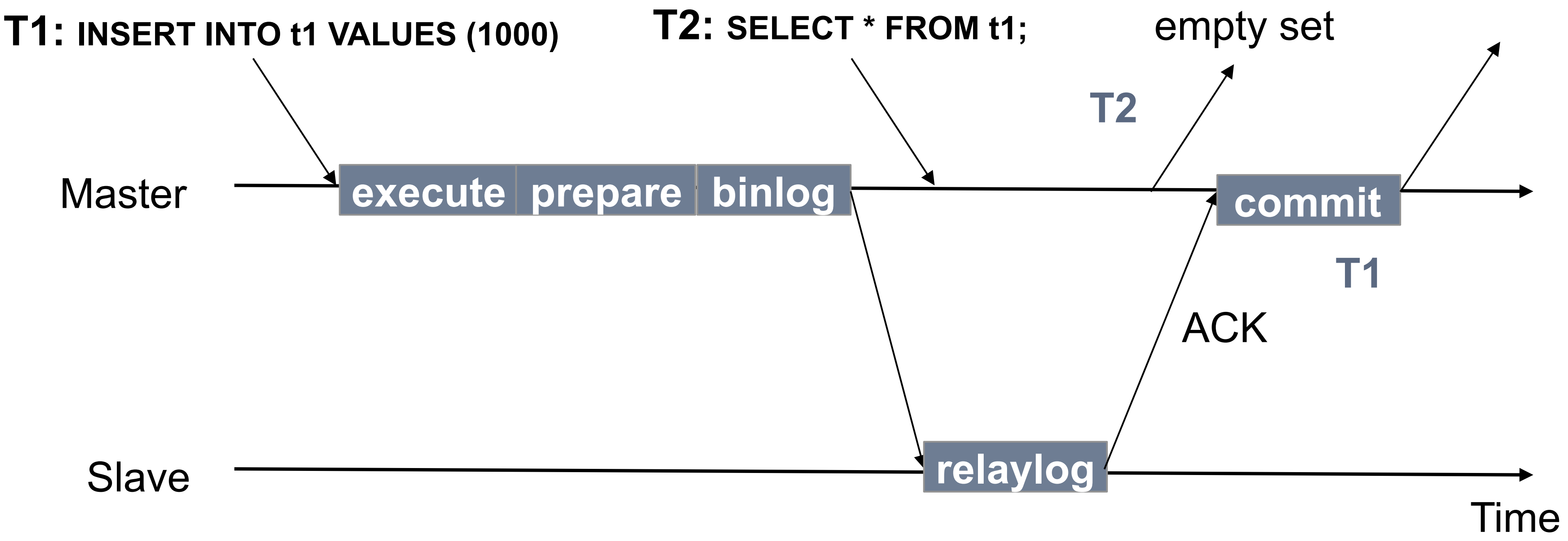
MySQL 5.6



MySQL 5.7 “Lossless”



Loss-less Semi-sync Replication

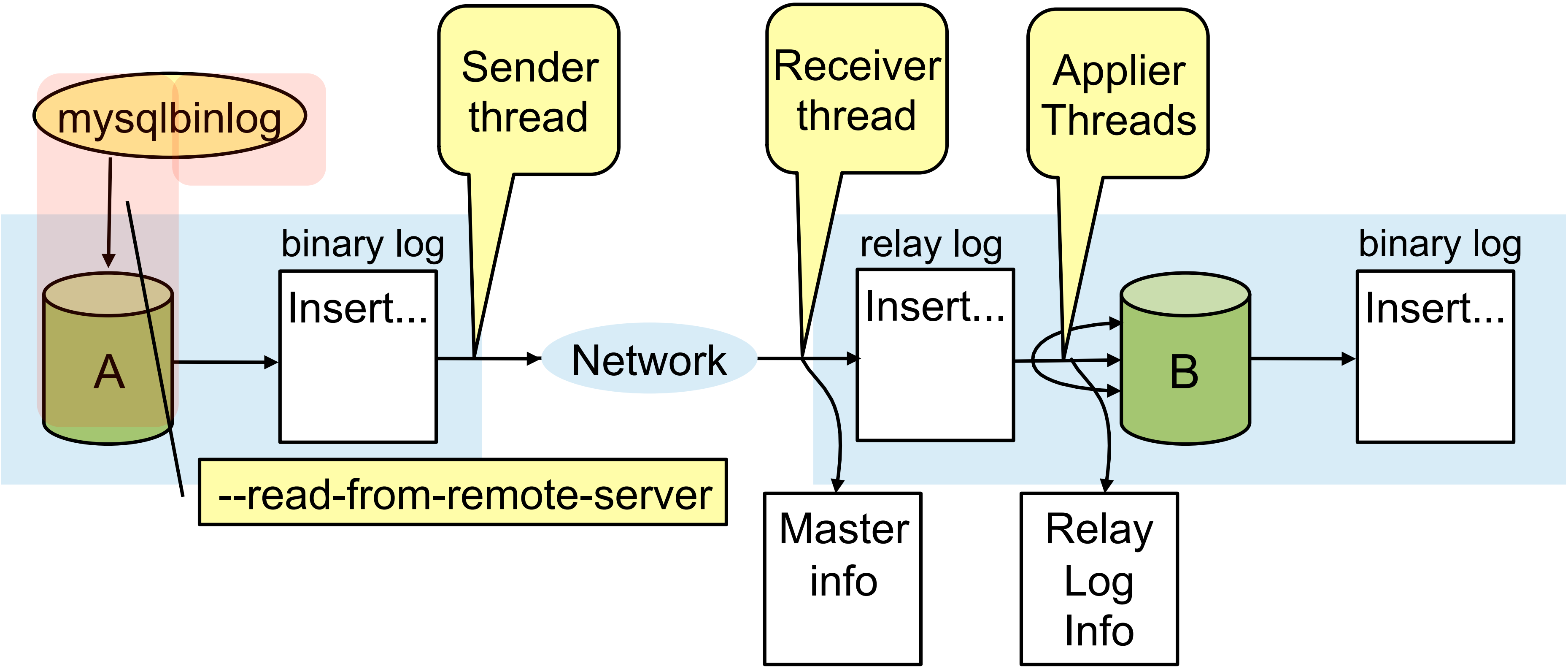


MySQL 5.6 GA以降の改良点

MySQL 5.7.3 - Development Milestone Release, December 2013

- Improved Security
  - SSL options for mysqlbinlog
- Flexible Semisync Durability
  - Configure master to wait for more than one semisync slave to ACK back.
- More Production Friendly
  - Changing Replication Filters Dynamically.

SSL options for mysqlbinlog



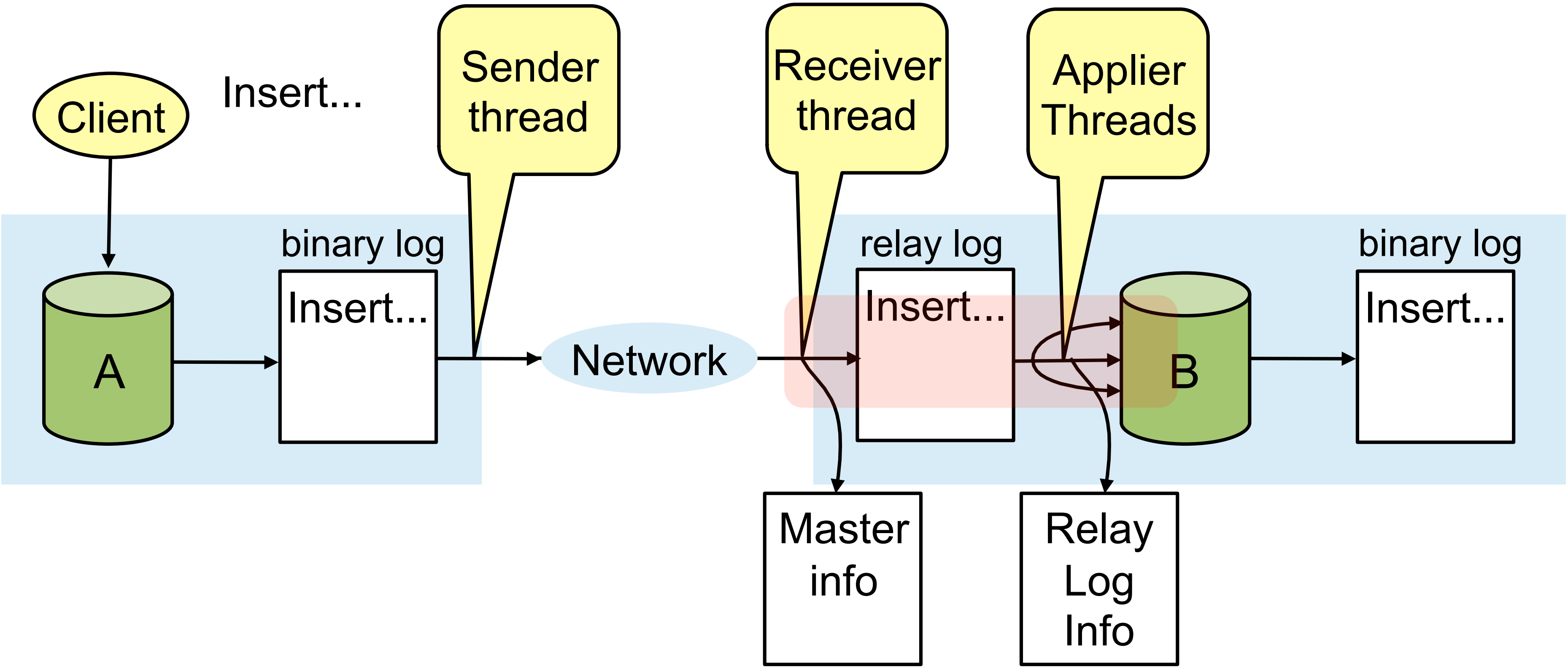
SSL options for mysqlbinlog

- --ssl* options to mysqlbinlog
  - Reading binary logs from remote servers through a secure channel.
  - Supports all SSL options that other client tools support.

```
mysql>  GRANT USAGE ON *.* TO 'rpluser'@'localhost' REQUIRE SSL;
```

```
shell>  mysqlbinlog --read-from-remote-server –ssl -u rpluser ...
```

Dynamic Slave Replication Filters

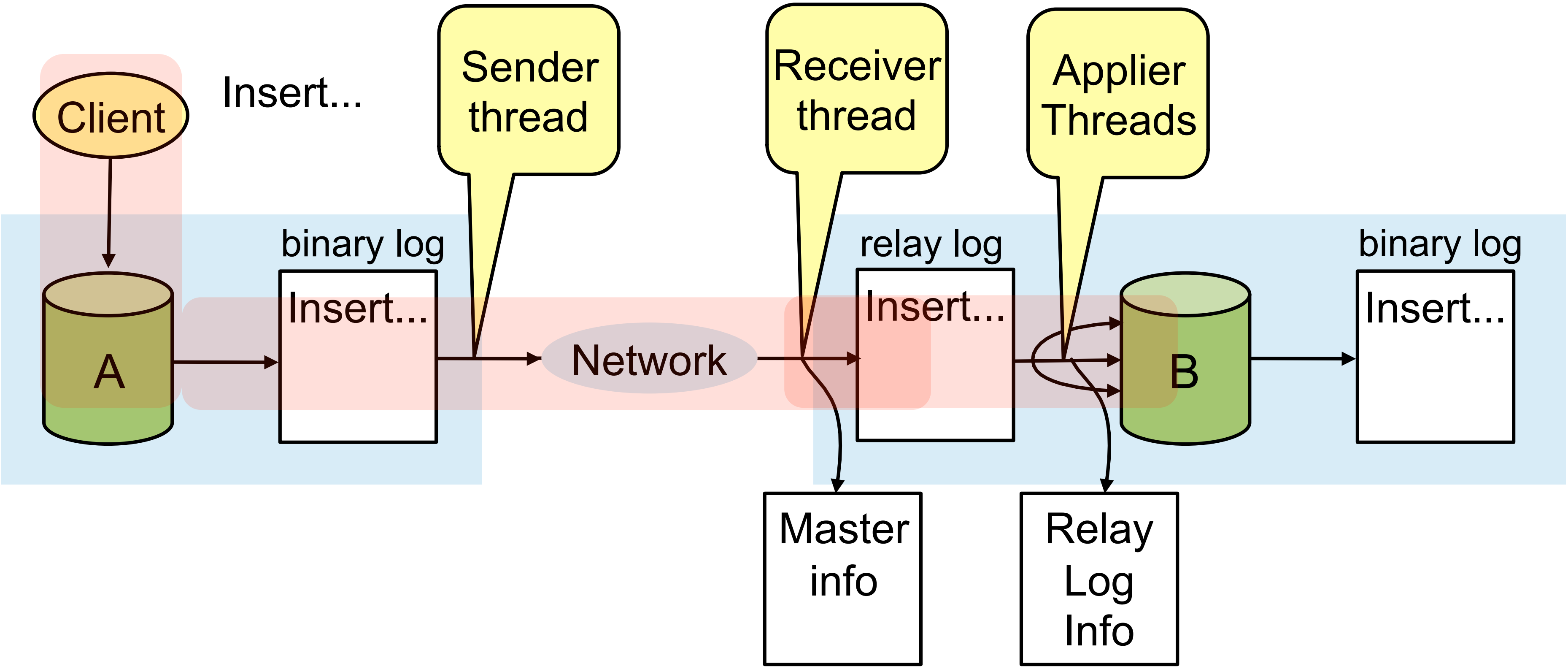


Dynamic Slave Replication Filters

- 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)
```

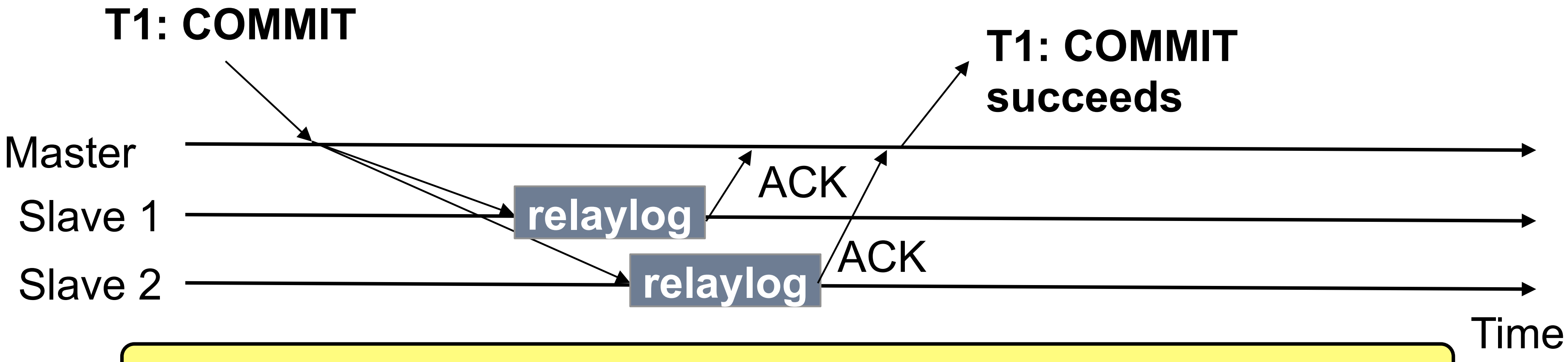
Semi-sync Replication – Wait for Multiple ACKs



Semi-sync Replication – Wait for Multiple ACKs

- Master does not commit transaction until it receives N ACKs from N slaves.
- Dynamically settable:

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



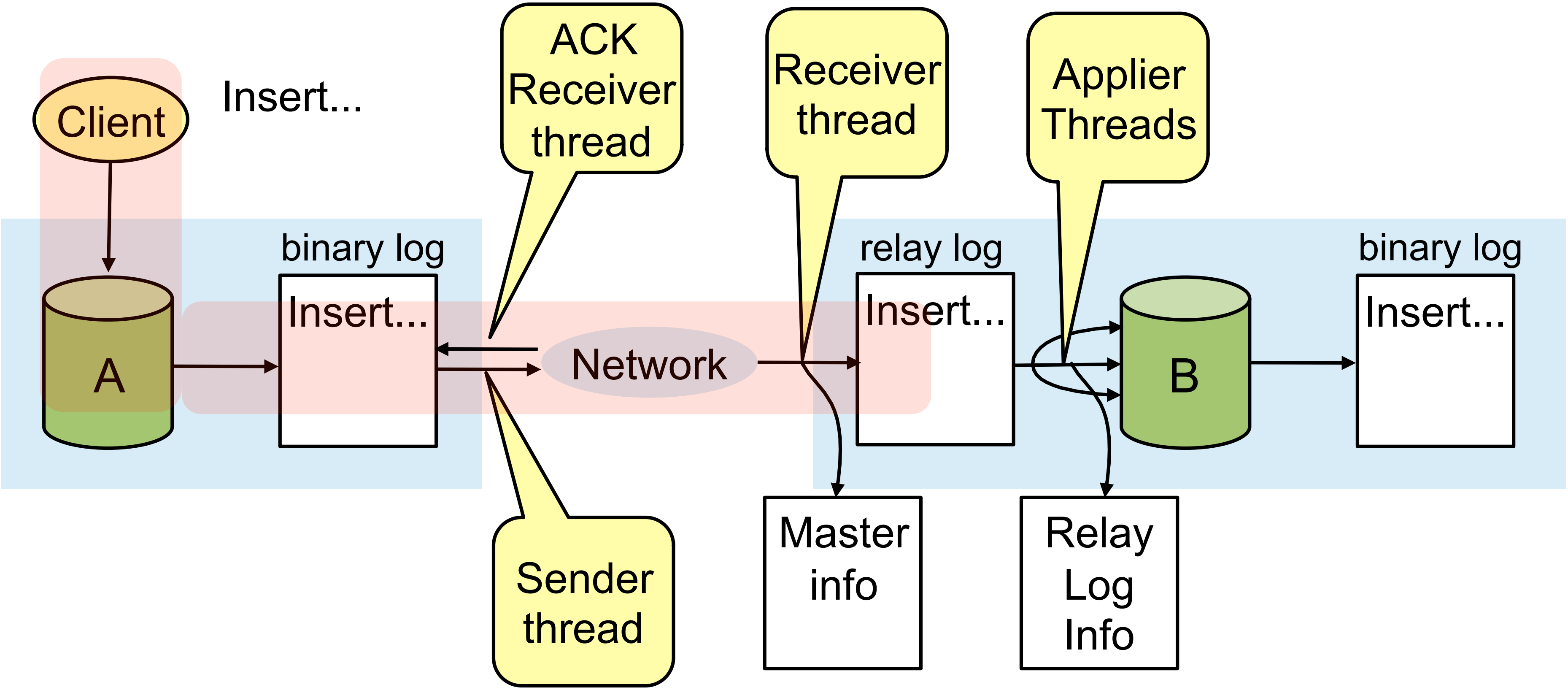
```
mysql> SET GLOBAL rpl_semi_sync_master_wait_for_slave_count= 2
```

MySQL 5.6 GA以降の改良点

MySQL 5.7.4 - Development Milestone Release, April 2014

- Performance
  - Sender thread enhancements.
  - Semisync ACK Receiver thread.
- Flexibility
  - Redirect to new master without stopping applying transactions.

Semi-sync Replication – ACK Receiver Thread



Semi-sync Replication – ACK Receiver Thread

- Consecutive transactions do not block each other while waiting for ACKs
- Transaction t1 and t2 are sent immediately to the slave by the sender thread
- ACKs are received only by a special thread
- Transaction t2 does not include t1 round trip in its semisync overall latency

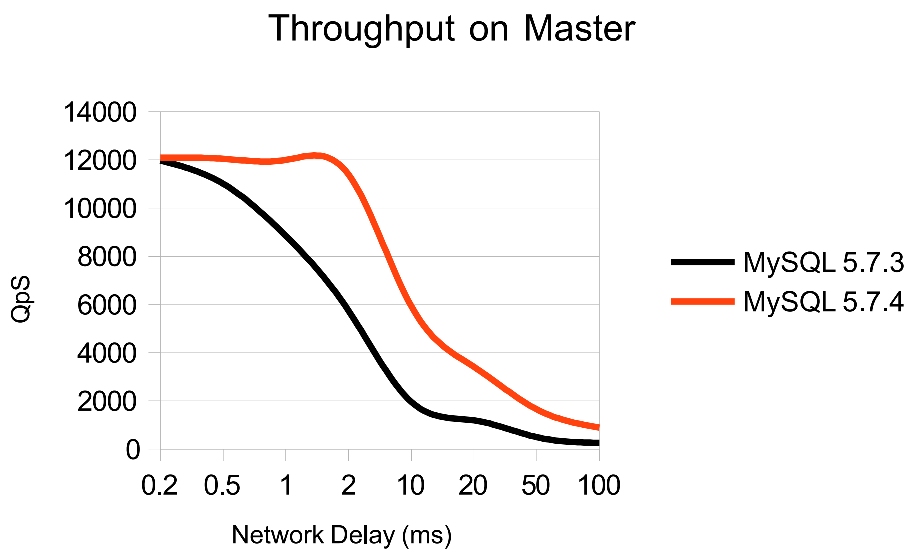
- Thread starts when semisync is activated

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

- Thread stops when semisync is deactivated

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

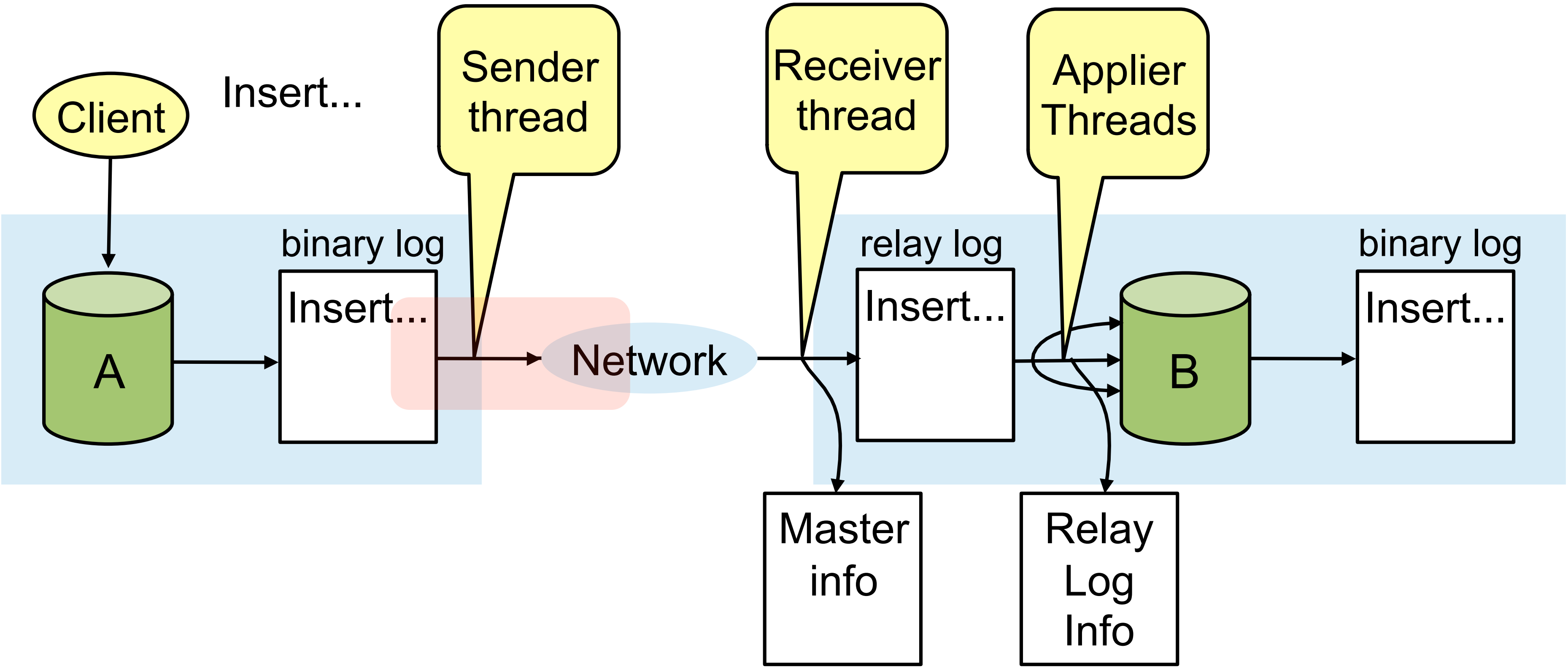
Semi-sync Replication – ACK Receiver Thread



- Ad hoc microbenchmark using mysqlslap
- 100K queries, commit=100, iterations=3
- 1 semisync slave
- Network delay example:
  tc qdisc add dev lo root netem delay 100ms

- 10X larger latency show **50%** less throughput in 5.7.3 under peak load.
- 10X larger latency show **0%** less throughput in 5.7.4 under peak load.

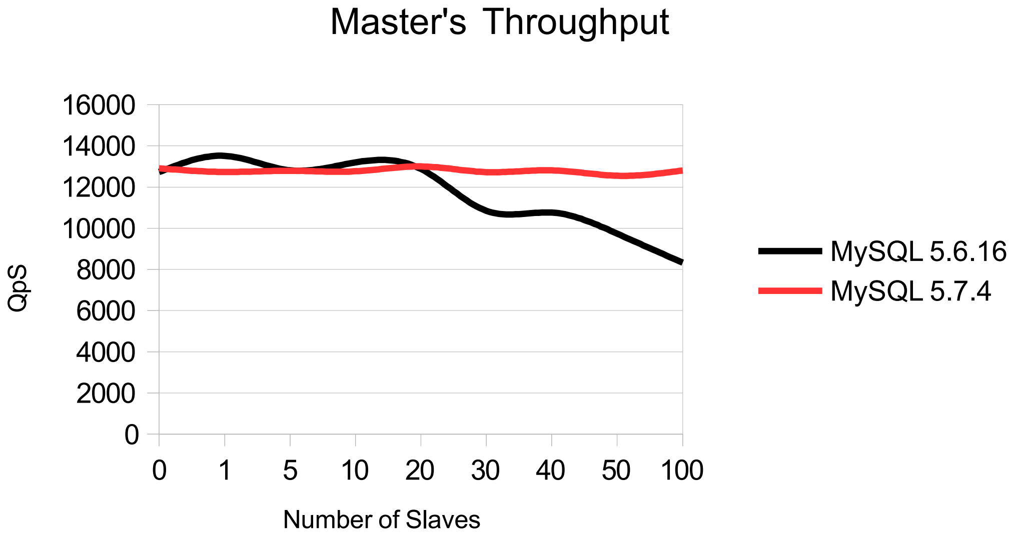
More Sender Thread Enhancements



More Sender Thread Enhancement

- Send buffer is not allocated and freed every time an event is sent.
- When sender threads require larger send buffer
  → buffer grows as needed.
- When buffer is larger than needed→ buffer shrinks dynamically.
- Together with the sender thread enhancements released on MySQL 5.7.2:
  - increase master scalability;
  - reduces resource consumption (CPU);
  - Master, with dump threads attached, copes better with peak loads.

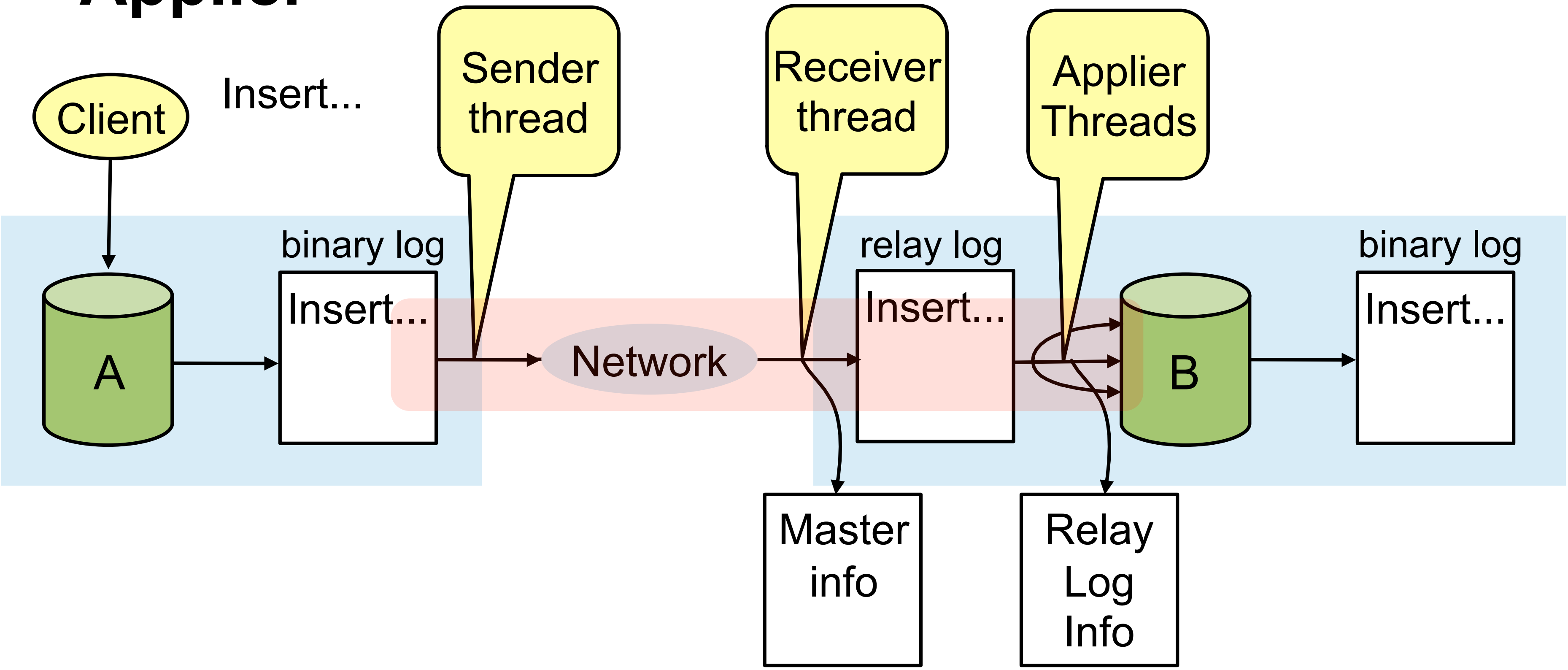
More Sender Thread Enhancements



- Ad hoc microbenchmark using mysqlslap
- 1M queries, concurrency=200, commit=1
- N slaves attached (fake slaves using remote mysqlbinlogs)
- 48 cores HT / 512 GB RAM / HDD

- Small number of slaves: **no difference** between 5.6 and 5.7.
- Larger number of slaves: **5.7 is able to sustain the same throughput.**

Redirect to new Master without interrupting the Applier



Redirect to new Master without interrupting the Applier

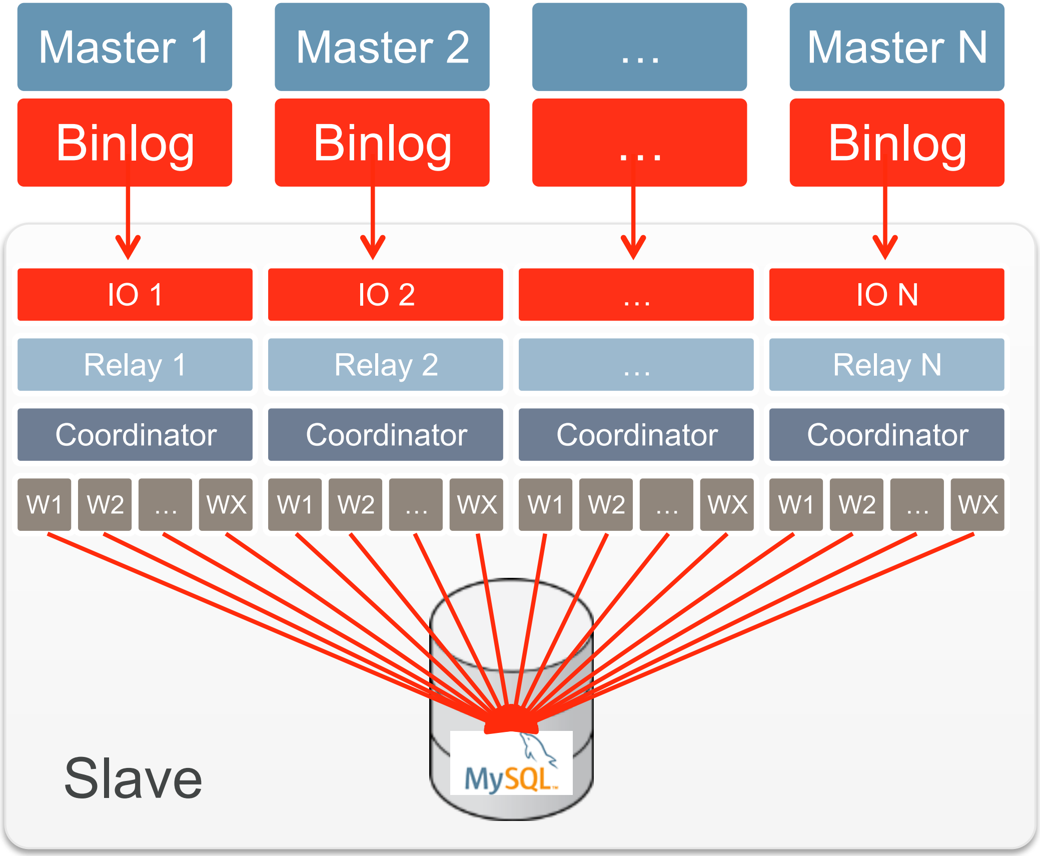
- More “onlineness” during fail-over:

```
mysql> STOP SLAVE IO_TRHEAD;
mysql> CHANGE MASTER TO MASTER_HOST='master2', …;
Mysql> START SLAVE IO_THREAD;
```

- Stopping, changing master and restarting the receiver thread is all done while the applier thread is running.

マルチソース レプリケーション

labs.mysql.com

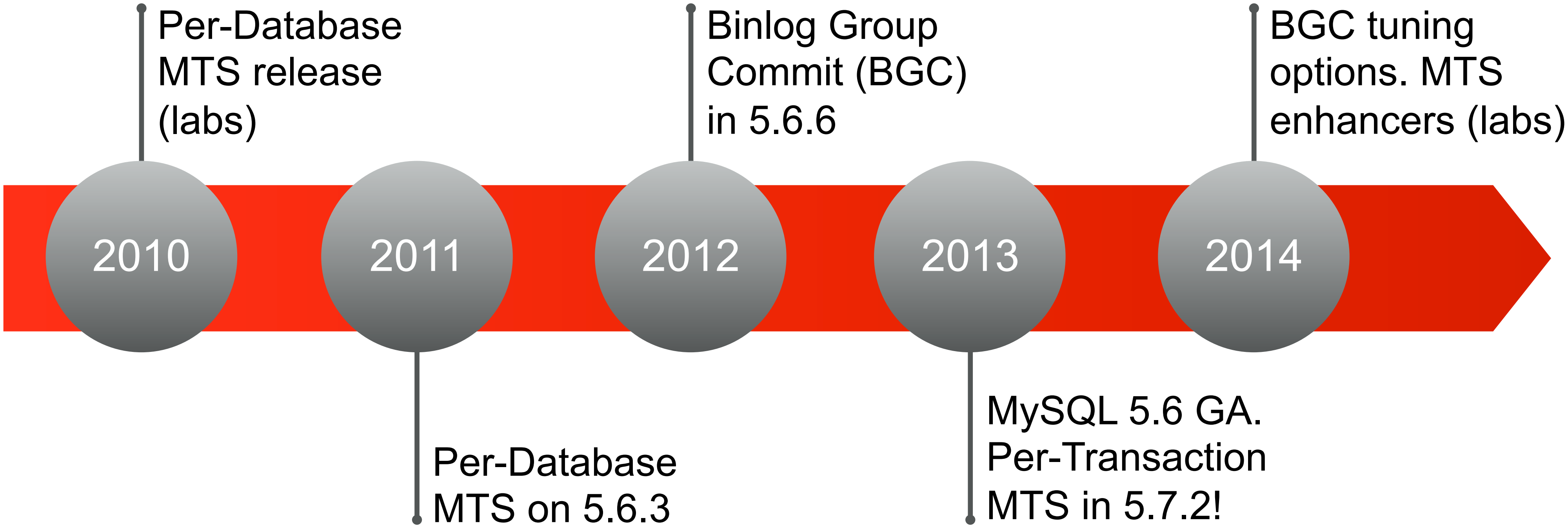


- 複数のマスターでの変更点を1台のスレーブに集約
  - 全ての「シャード」のデータを単一のビューで分析
  - バックアップ用にデータを集約
- 準同期レプリケーションおよびマルチスレッド スレーブに対応
- マスターごとにフィルタと制御可能となる予定
- アプリケーションはマスタごとに個別にアクセス

|

Multi-Threaded Slave (MTS)

開発の履歴



2010

Per-Database MTS release (labs)

2011

Per-Database MTS on 5.6.3

2012

Binlog Group Commit (BGC) in 5.6.6

2013

MySQL 5.6 GA. Per-Transaction MTS in 5.7.2!

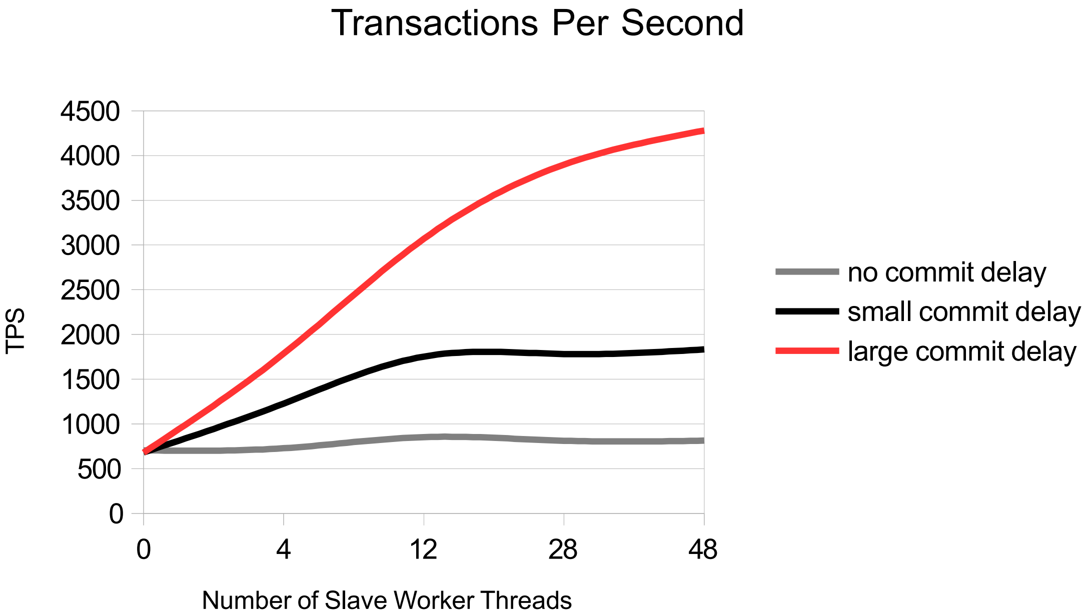
2014

BGC tuning options. MTS enhancers (labs)

Multi-Threaded Slave (MTS)

MySQL labs.mysql.com

- By tuning BGC on the master, we get more parallelism on the slave:
  - binlog-group-commit-delay
  - binlog-group-commit-count
- Tuning means more transactions preparing together (no magic formula – highly dependent on hardware and workload).
- Still working on improving it and on the analysis of experimental results.

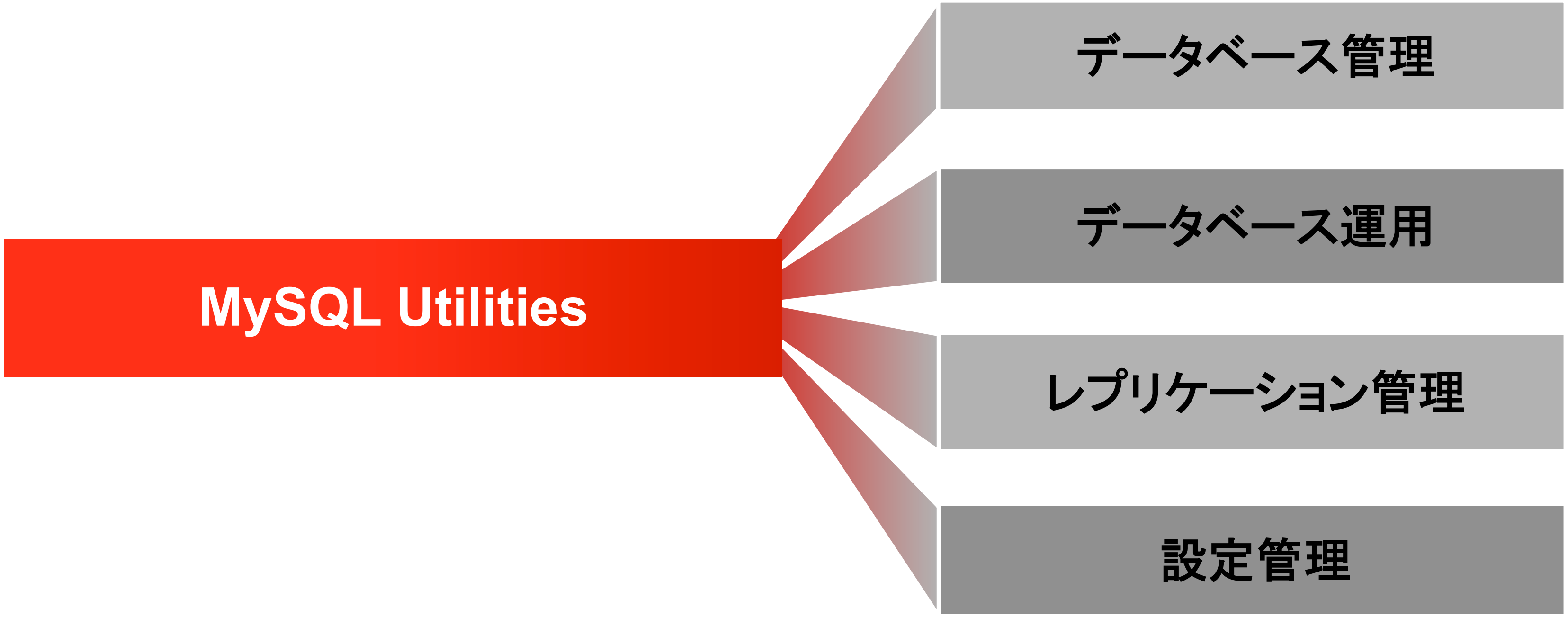


6X slave throughput – large commit delay
3X slave throughput – small commit delay

MySQL Fabric

MySQL Utilities

運用管理に関するPythonスクリプト



MySQL Utilities

データベース管理

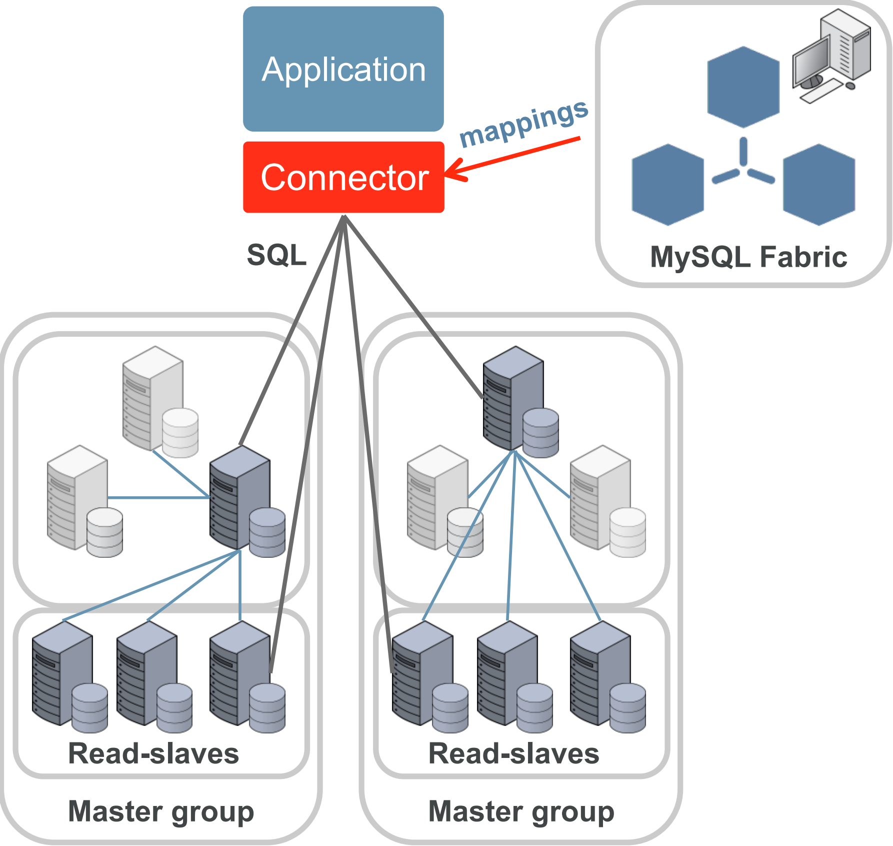
データベース運用

レプリケーション管理

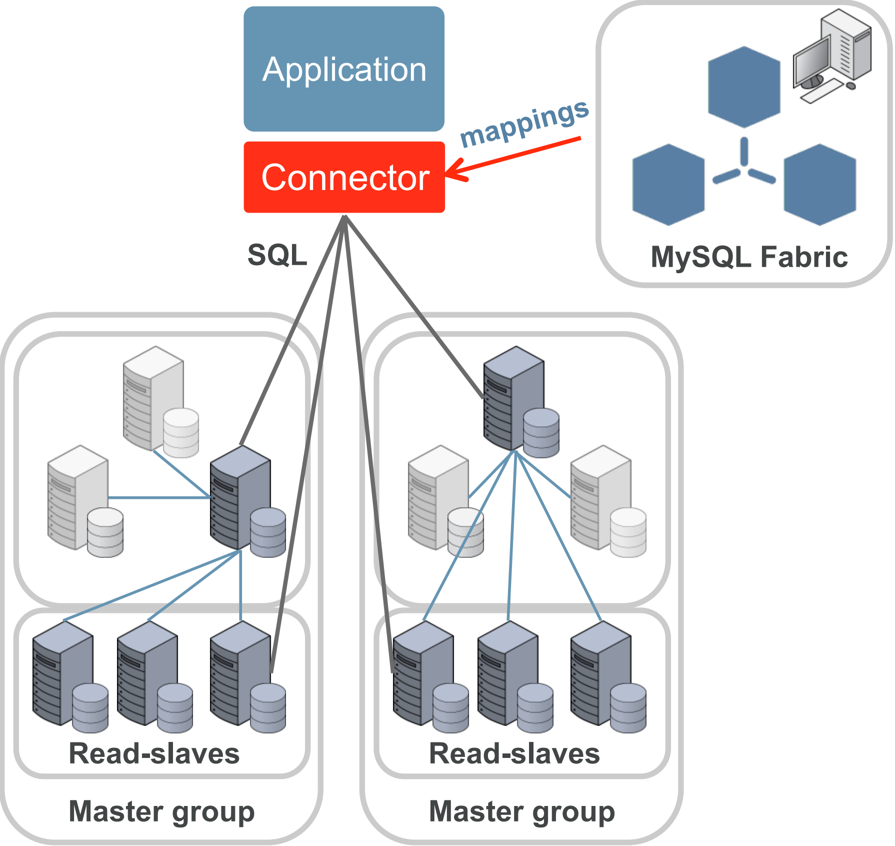
設定管理

MySQL Fabric

An extensible and easy-to-use framework for managing a farm of MySQL servers supporting high-availability and sharding



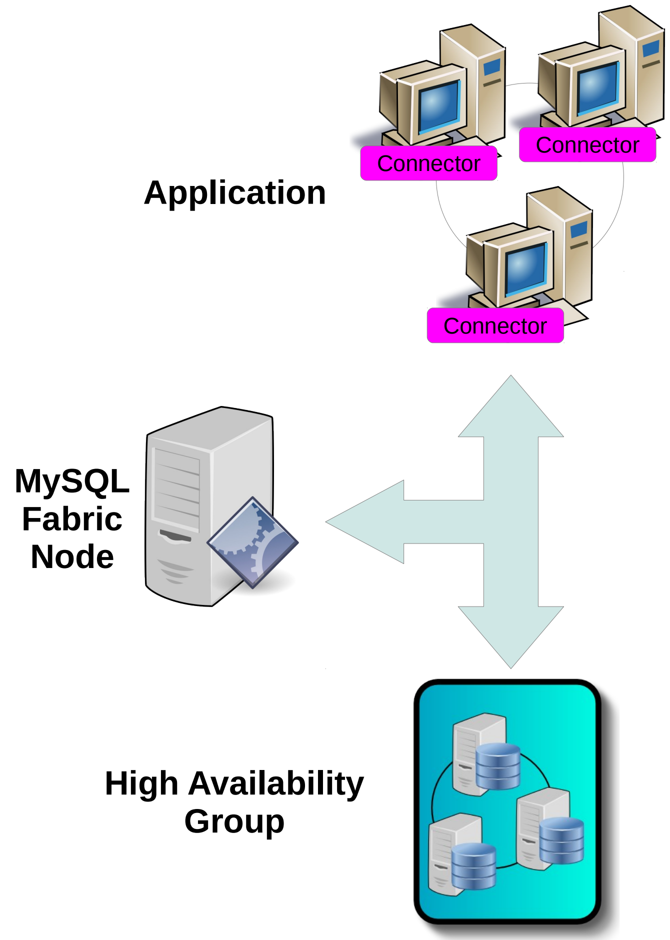
MySQL Utilities - Fabric



- 「シャーディング」による拡張性
- コネクタ
  - Python
  - Java
  - PHP
- アプリケーションでの分割キー
  - Range または Hash
  - シャード再構成も可能
  - シャード全体の更新も可能
- MySQL Utilities 1.4.0として提供

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
```

MySQL Fabric: Basic Commands and Help

- Command Structure
  - `mysqlfabric group command ...`
- Getting help

  mysqlfabric help

  ```
  mysqlfabric help commands
  mysqlfabric help manage
  mysqlfabric help manage setup
  ```
- MySQL Utilities Documentation:
  - http://dev.mysql.com/doc/mysql-utilities/1.4/en/index.html
- MySQL Fabric Documentation:
  - http://dev.mysql.com/doc/mysql-utilities/1.4/en/fabric.html

Setting up and Tearing down MySQL Fabric

- Create and populate the necessary tables in backing store

```
mysqlfabric manage setup
```

- Remove the tables from backing store

```
mysqlfabric manage teardown
```

- Connects to the database server in “storage” section
  - Ensure that you have the necessary users and privileges

Starting and Stopping MySQL Fabric

- Start MySQL Fabric node in foreground – print log to terminal

```
mysqlfabric manage start
```

- Start MySQL Fabric node in background – print log to file

```
mysqlfabric manage start --daemonize
```

- Stop MySQL Fabric node

```
mysqlfabric manage stop
```

Create 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
```

Activate High-Availability Group

- Promote one server to be primary

```
mysqlfabric group promote my_group
```

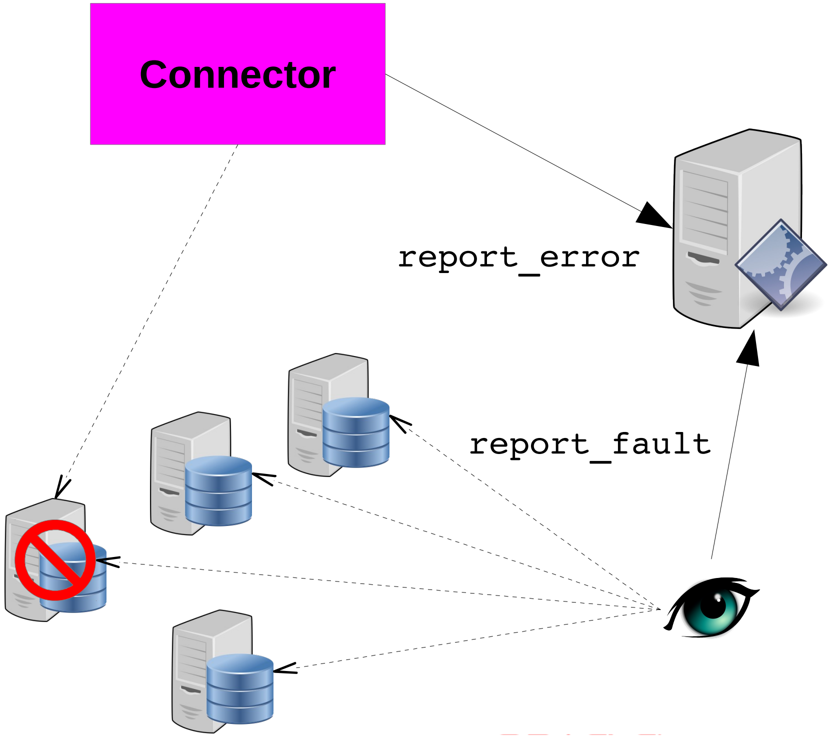
- Tell built-in failure detector to monitor group

```
mysqlfabric group activate my_group
```

Distributed Failure Detector

New in MySQL Fabric 1.4.2

- Connectors report errors
  - Report that an error was noticed
  - Failover based on statistics
  - report_error(server, source, error)
- Report failure
  - A server is known to have failed
  - Failover occurs immediately
  - report_fault(server, source, error)



Fabric-aware Connector

- Fabric-aware Connectors
  - Connector/J
  - Connector/Python
  - Connector/PHP
- Fabric-aware Frameworks
  - Doctrine
  - Hibernate
- In this presentation:
  - Connector/Python

- Connector API Extensions
  - Support Transactions
  - Support full SQL
- Decision logic in connector
  - Reducing network load
- Load Balancing
  - Read-Write Split
  - Distribute transactions

Fabric-aware Connector API

- Establish a “virtual” connection
  - Real server connection established lazily
- Provide connection information for the *Fabric node*
  - Connector will fetch information about servers

```
import mysql.connector

conn = mysql.connector.connect(
    fabric={"host": "fabric.example.com"},
    user='mats', password='xyzzy', database="employees"
)
```

Enable Connector/Python Error Reporting

New in Connector/Python 1.2.1

- Connectors can report errors to Fabric node
  - Enable using report_error
  - Defaults to False
  - Require MySQL Fabric 1.4.2

```
import mysql.connector

conn = mysql.connector.connect(
    fabric={"host": "fabric.example.com"},
    user='mats', password='xyzzy', database="employees",
    report_error=True,
)
```

Connector API: Executing a Transaction

- Provide group name
  - **Property:** group
  - Fabric will compute candidate servers

- Provide transaction mode
  - **Property:** mode
  - Fabric will pick server in right mode

Same as before

```
conn.set_property(group='my_group', mode=MODE_READWRITE)
cur = conn.cursor()
cur.execute("INSERT INTO employees VALUES (%s,%s,%s)",
            (emp_no, first_name, last_name))
cur.execute("INSERT INTO titles(emp_no,title,from_date)"
            “ VALUES (%s,%s,CURDATE())",
            (emp_no, 'Intern'));
conn.commit()
```

MySQL Fabric: Set up Shard Mapping

Will return a
shard map identifier

- Define shard mapping

```
mysqlfabric sharding \
      create_definition hash my_global
```

Shard map identifier

- Add tables that should be sharded

```
mysqlfabric sharding add_table 1 \
      employees.employees emp_no
mysqlfabric sharding add_table 1 \
      employees.salaries emp_no
```

- *Tables not added are considered global*

MySQL Fabric: Add Shards

- Add shards to shard mapping

```
mysqlfabric sharding add_shard 1 \
     “my_group.1,...,my_group.N” --state=ENABLED
```

Shard map identifier

MySQL Fabric: Moving and Splitting Shards

Shard ID

- Moving a shard from one group to another

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

- Splitting a shard into two pieces (hash)

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

Connector API: Shard Specific Query

- Provide tables in query
  - **Property:** tables
  - Fabric will compute map

- Provide sharding key
  - **Property:** key
  - Fabric will compute shard

```
conn.set_property(tables=['employees.employees', 'employees.titles'],
                  key=emp_no)
cur = conn.cursor()
cur.execute("INSERT INTO employees VALUES (%s,%s,%s)",
            (emp_no, first_name, last_name))
cur.execute("INSERT INTO titles(emp_no, title, from_date)"
            " VALUES (%s, %s, CURDATE())",
            (emp_no, 'Intern'));
conn.commit()
```

Connector API: Global Update

- Provide tables in query
  - **Property:** tables
  - Fabric will compute map
  - (Not necessary)

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

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

What do we have now?

- MySQL Farm Management
  - High-Availability
  - Sharding
- High-Availability
  - Group Concept
  - Slave promotion
- Sharding
  - Range and hash sharding
  - Shard move and shard split
- Connector APIs
  - Transaction properties
  - “Virtual” connections
- Enhanced Connectors
  - Connector/Python
  - Connector/PHP
  - Connector/J
- Command-line Interface
- XML-RPC Interfaces
- Distributed failure detector
  - Connectors report failures
  - Custom failure detectors
- Credentials
  - RFC 2617
  - SSL support

Thoughts for the Future

- Connector multi-cast
  - Scatter-gather
  - UNION of result sets
  - More complex operations?
- Extension interfaces
  - Improve extension support
  - Improve procedures support
- Command-line interface
  - Improving usability
  - Focus on ease-of-use
- More protocols
  - MySQL-RPC Protocol?
- More frameworks?
- More connectors?
  - C/C++?
  - Fabric-unaware connectors?
- More HA group types
  - DRBD
  - MySQL Cluster

Thoughts for the Future

- "Transparent" Sharding
  - Single-query transactions?
  - Speculative execution?
  - Cross-shard join?
- Multiple shard mappings
  - Independent tables
- Multi-way shard split
  - Efficient initial sharding
  - Better use of resources

- High-availability executor
  - Node failure stop execution
  - Replicated State Machine
    - Paxos?
    - Raft?
  - Continue execution on other Fabric node
- Session Consistency
  - We have a distributed database
  - It should look like a single database

WE ARE
HIRING

Hardware and Software

ORACLE®

Engineered to Work Together

ORACLE®