MySQL Conference & Expo

Presented by MySQL Conference & Expo O'REILLY

MySQL Replication Tutorial

Lars Thalmann
Technical lead
Replication, Backup, and Engine Technology

Mats Kindahl
Lead Developer
Replication Technology

MySQL Conference and Expo 2008

MySQL®
Conference & Expo

Presented by

MySQL Conference & Expo

O'REILLY

Concepts

MySQL Replication

Why?

1. **High Availability**
   Possibility of fail-over
2. **Load-balancing/Scale-out**
   Query multiple servers
3. **Off-site processing**
   Don't disturb master

How?

Snapshots (Backup)

1. **Client program mysqldump**
   With log coordinates
2. **Using backup**
   InnoDB, NDB

Binary log

1. **Replication**
   Asynchronous pushing to slave
2. **Point-in-time recovery**
   Roll-forward

Terminology

Master MySQL Server

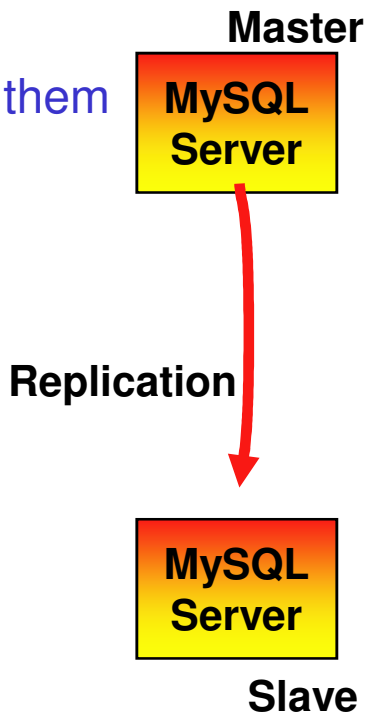
- Changes data
- Has binlog turned on
- Pushes binlog events to slave after slave has requested them

Slave MySQL Server

- Main control point of replication
- Asks master for replication log
- Gets binlog event from master

Binary log

- Log of everything executed
- Divided into transactional components
- Used for replication and point-in-time recovery



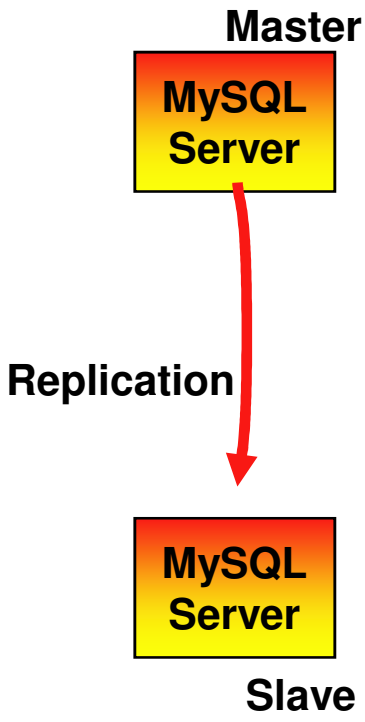
Terminology

Synchronous replication

- A transaction is not committed until the data has been replicated (and applied)
- Safer, but slower
- This is available in MySQL Cluster

Asynchronous replication

- A transaction is replicated after it has been committed
- Faster, but you can in some cases loose transactions if master fails
- Easy to set up between MySQL servers



MySQL®
Conference & Expo

Presented by

MySQL Conference & Expo

O'REILLY

Configuring Replication

Required configuration – my.cnf

- Replication Master

```
log-bin

server_id
```

- Replication Slave

```
server_id
```

Optional items in my.cnf – What to replicate?

- Replication Master

```
binlog-do-db

binlog-ignore-db
```

- Replication Slave

```
replicate-do-db, replicate-ignore-db

replicate-do-table, replicate-ignore-table

replicate-wild-do-table

replicate-wild-ignore-table
```

More optional configuration on the slave

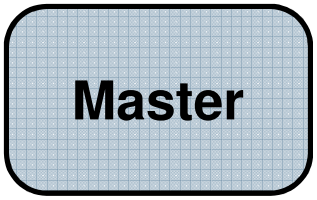
- `read-only`
- `log-slave-updates`
- `skip-slave-start`

Configuration – grants on master

```
GRANT REPLICATION SLAVE on *.*
TO 'rep_user'@'slave-host'
IDENTIFIED BY 'this-is-the-password'
```

How to deploy replication

Step 1: Make a backup of the master



Either an “offline backup”
or an “online backup”...

Configuration – Good advice

- Start the binary log on the master immediately following the backup. *e.g.:*

  Make the GRANTs on the master server

  Shut down mysqld on the master server

  Edit my.cnf

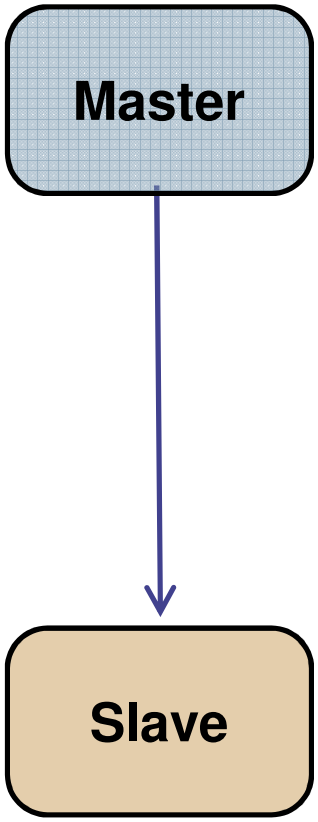
  Make the backup

  Restart mysqld on the master

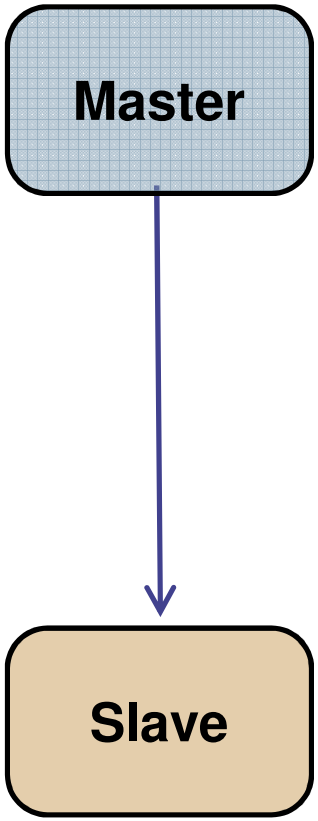
- Do *not* try to configure master_host, etc. in my.cnf on the slave.

  (this is still allowed, but it was always a bad idea)

Restore the backup onto the slave

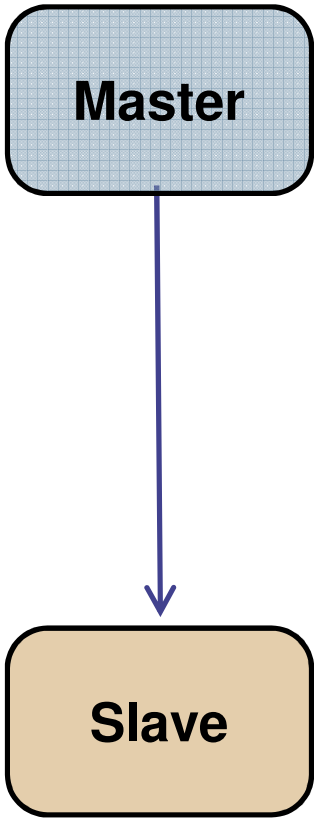


Configure the slave: part 1



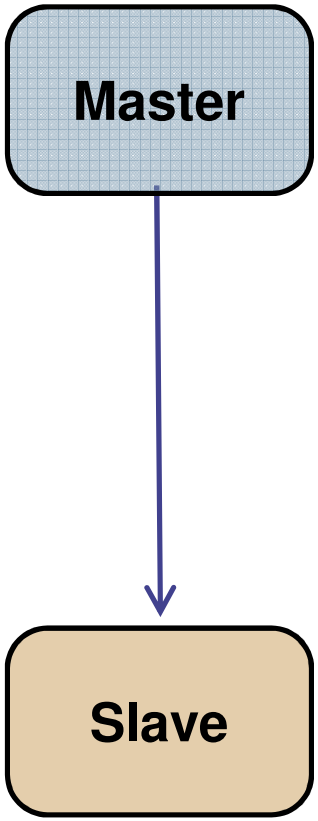

```
CHANGE MASTER TO
master_host = “dbserv1”,
master_user = “rep-user”,
master_password =
  “this-is-the-password”;
```

Configure the slave: part 2




```
CHANGE MASTER TO
master_host = “dbmaster.me.com”,
master_log_file = “binlog-00001”,
master_log_pos = 0;
```

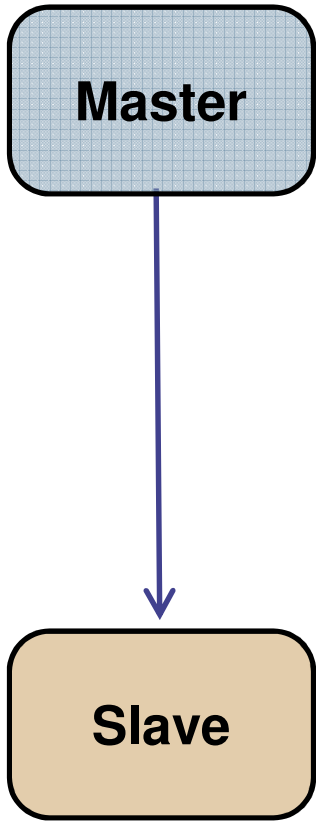
Start the slave!



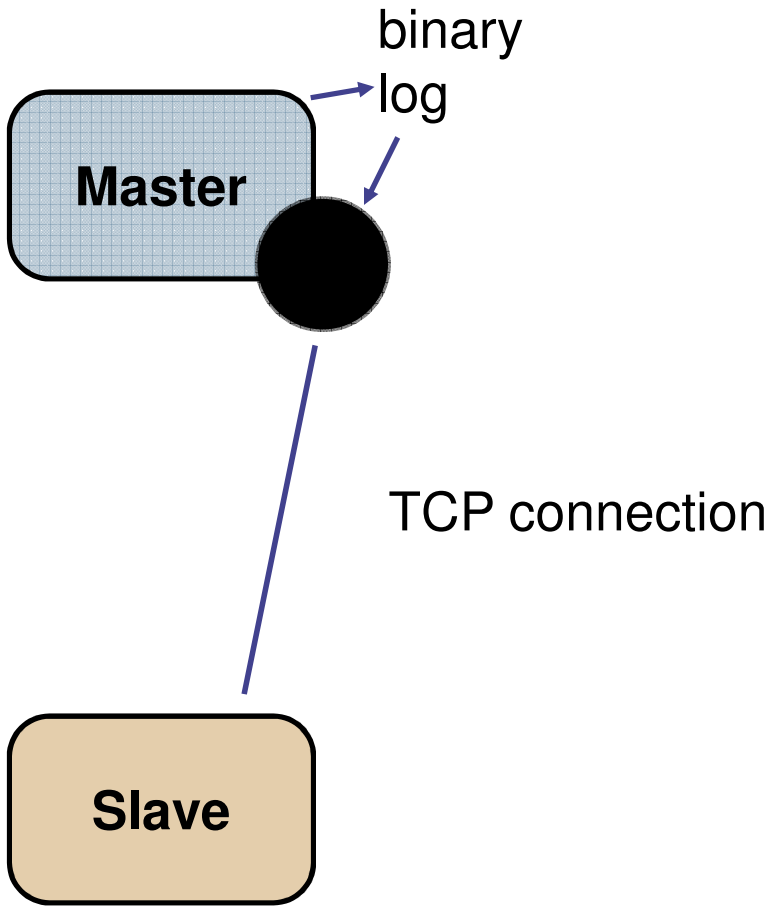

```
START SLAVE;
```

Replication Topologies

Master with Slave



Master with Slave



Replication is independent of Storage Engines

- You can replicate between any pair of engines

  InnoDB to InnoDB

  MyISAM to MyISAM

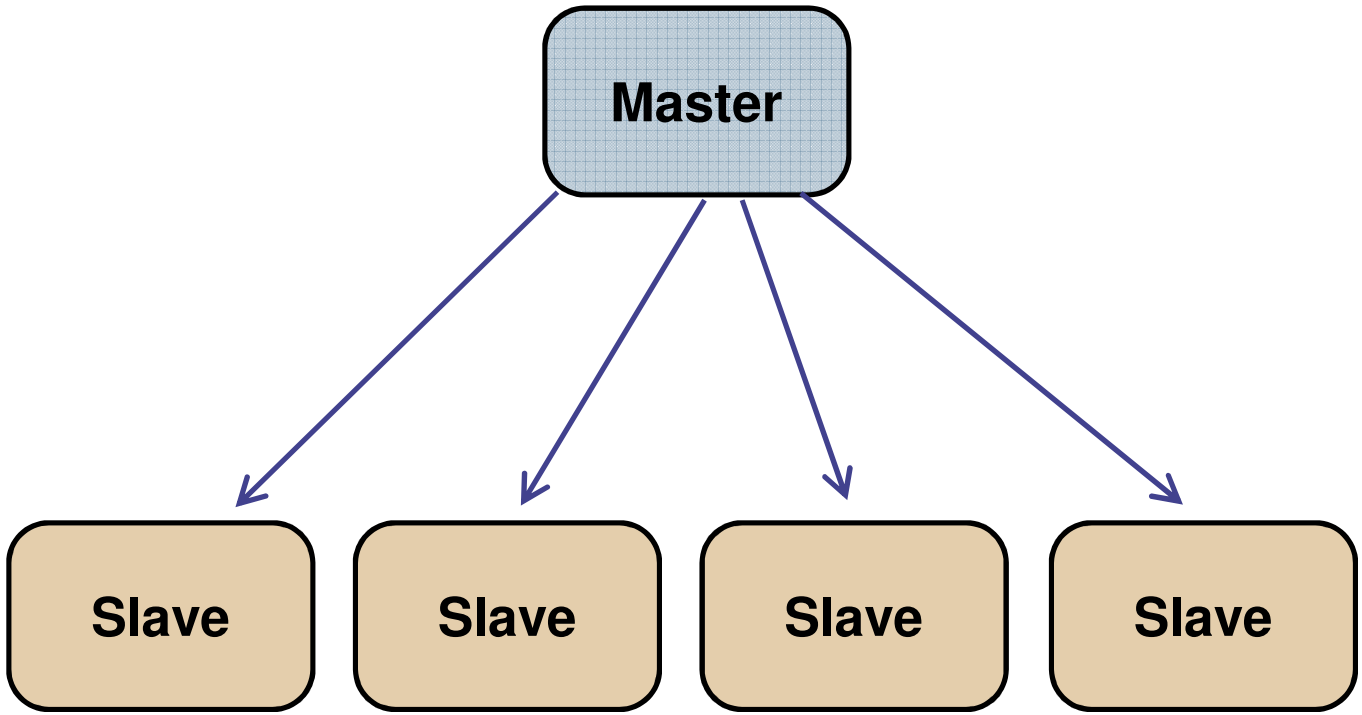
  InnoDB to MyISAM

  MEMORY to MyISAM

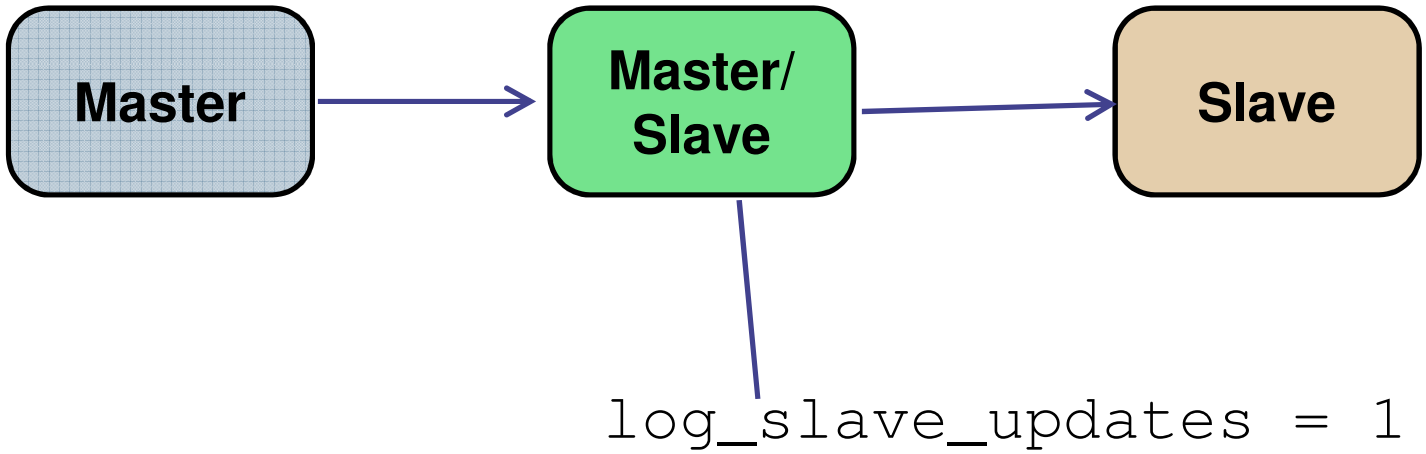
  etc...

- The binary log is **not** the InnoDB transaction log (or the Falcon log, or ...)

Master with Many Slaves



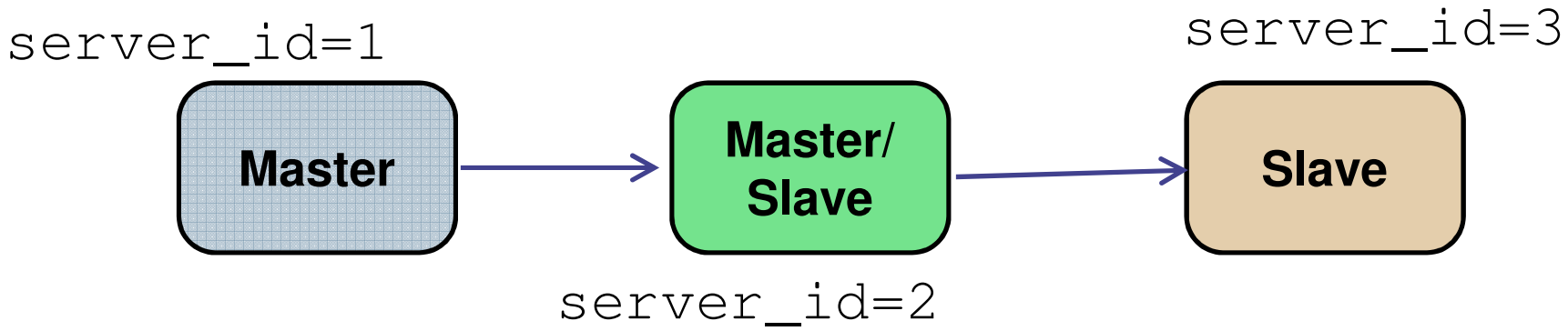
Chain



Chain – Server 2 goes down...

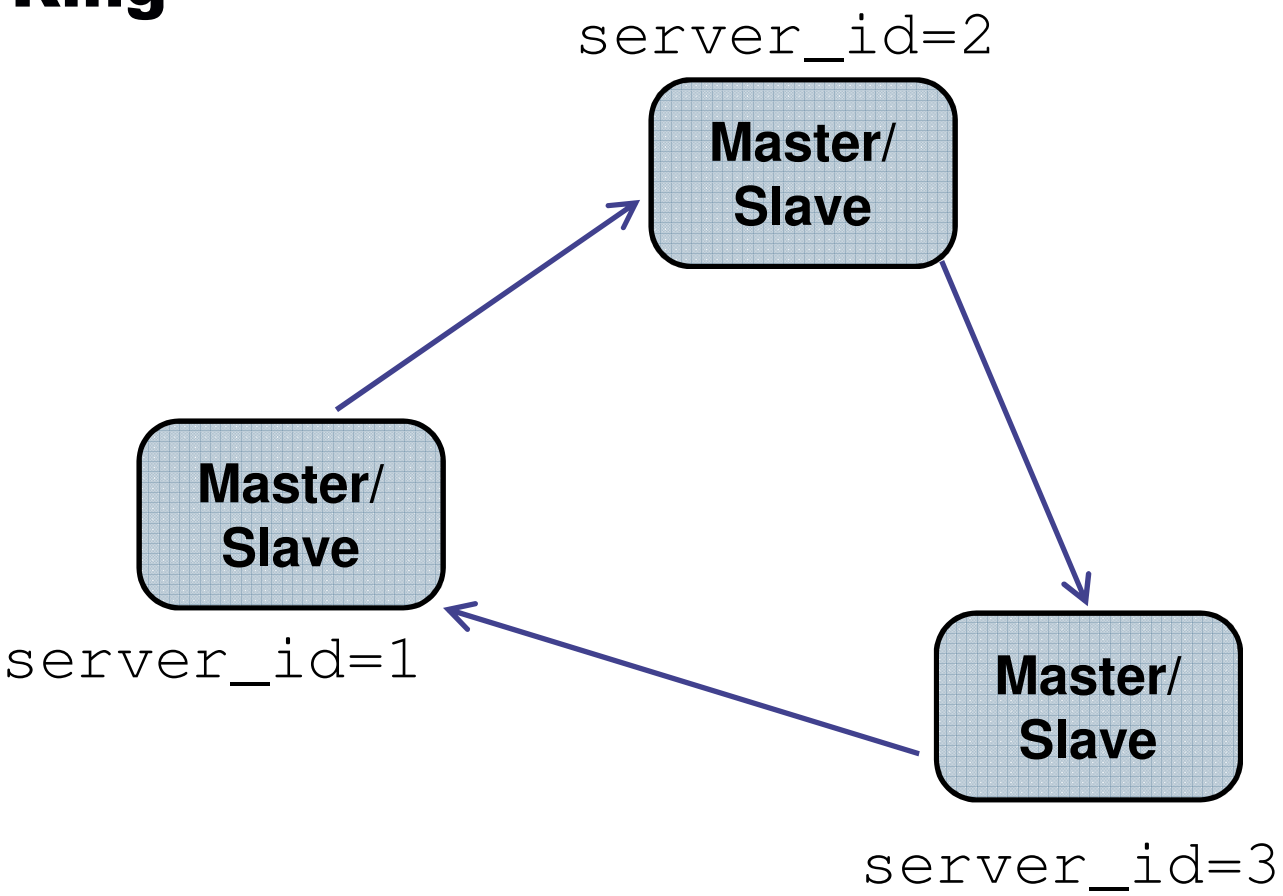
… Server 3 is still up, but out of sync

Each server has a unique “server_id”

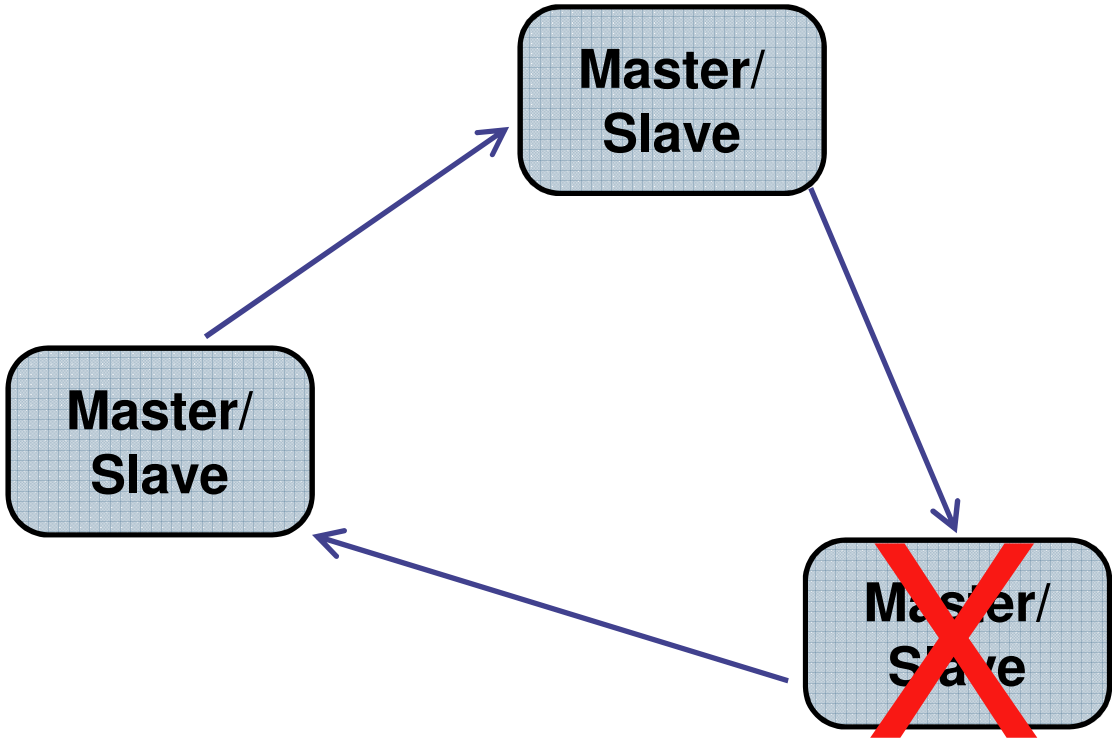


... and every event in a binary log file contains the server id number of the server where the event originated.

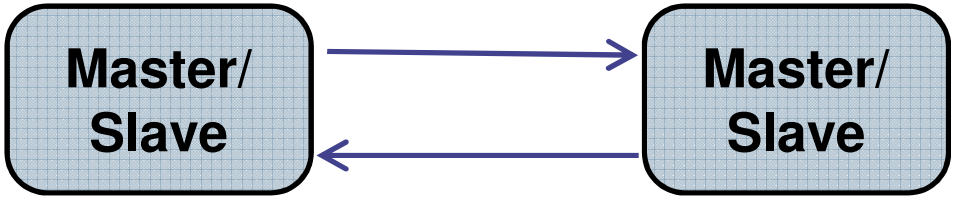
Ring


server_id=2
Master/
Slave
Master/
Slave
server_id=1
Master/
Slave
server_id=3

The ring topology is not a recommended configuration

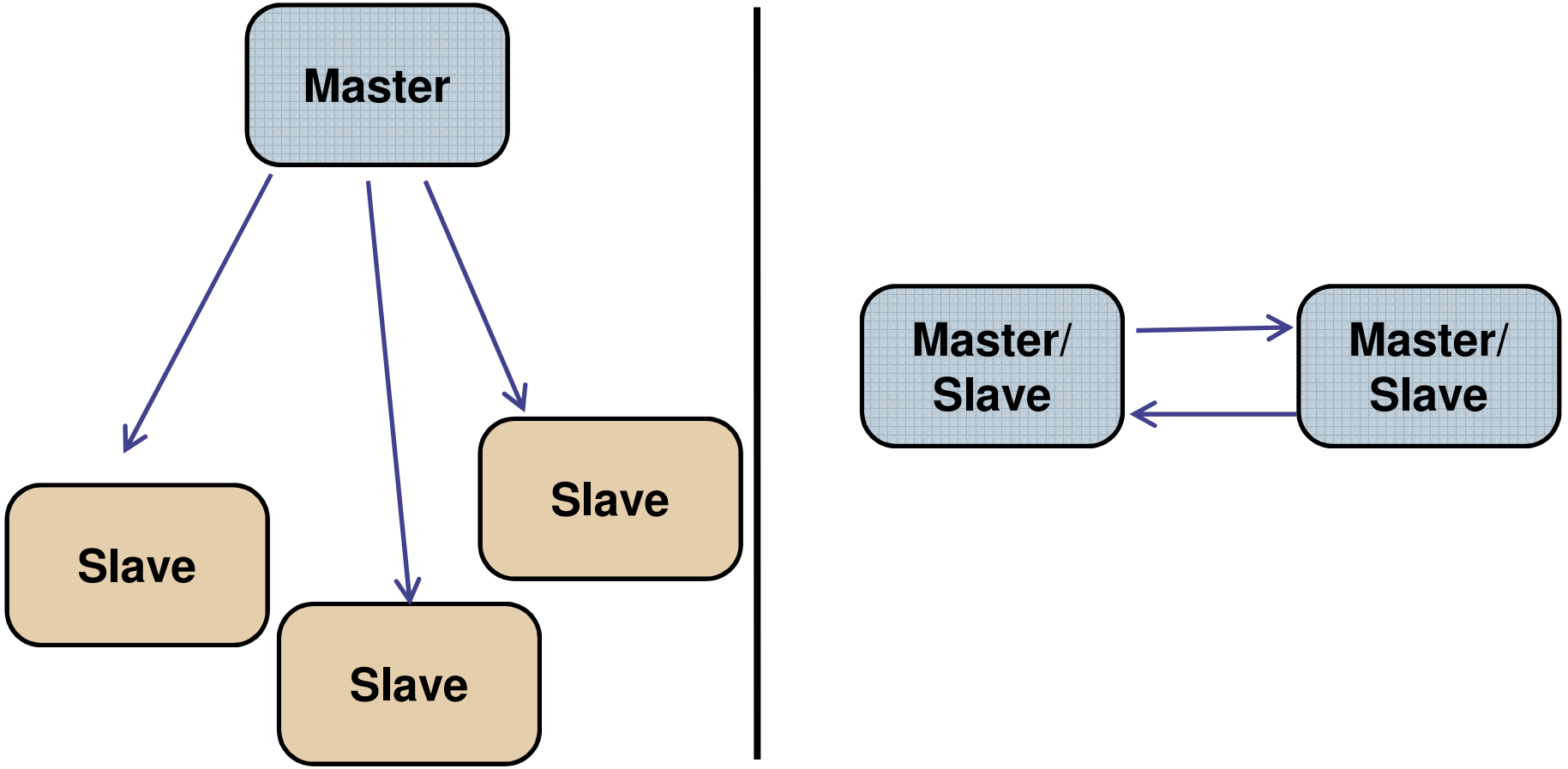


Pair of Masters



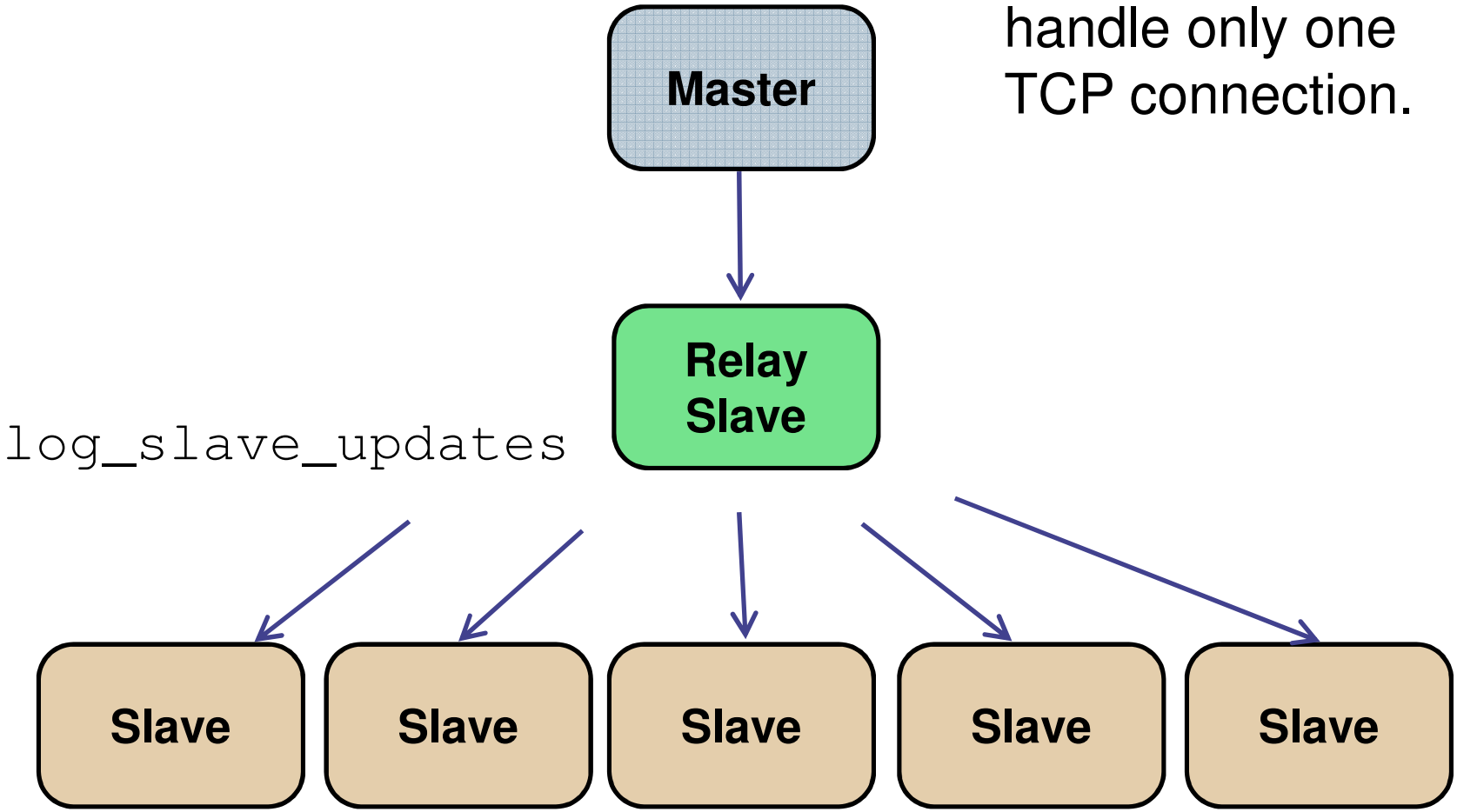
The pair is a “special case” of the ring topology used for
high availability.

The two most common topologies for MySQL Replication

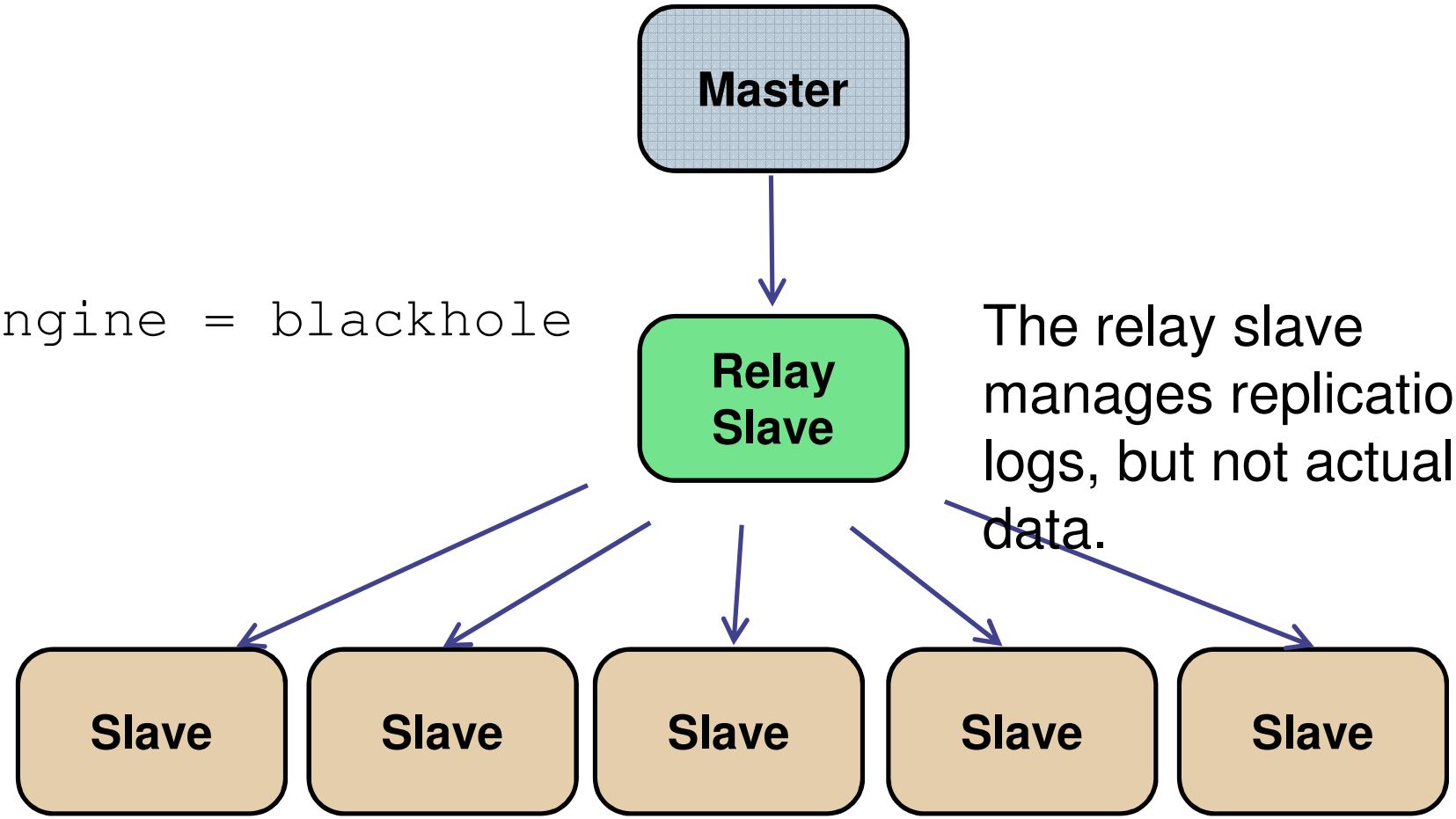


The “Relay Slave”

The master has to handle only one TCP connection.



And now introducing... the blackhole storage engine




```
engine = blackhole
```

The relay slave manages replication logs, but not actual data.

MySQL® Conference & Expo

Presented by MySQL Conference & Expo O'REILLY

Replication Commands

A quick run-through of the commands

SHOW MASTER STATUS

- Used on master
- Requires SUPER or REPLICATION CLIENT privileges
- Gives log file and position master is writing to
- Also shows database filters used

```
mysql> SHOW MASTER STATUS;
+----------------+----------+--------------+------------------+
| File           | Position | Binlog_Do_DB | Binlog_Ignore_DB |
+----------------+----------+--------------+------------------+
| mysql-bin.003  |       73 |         test |     manual,mysql |
+----------------+----------+--------------+------------------+
```

SHOW BINARY LOGS

- Used on master
- Requires SUPER privileges
- Will display a list of binary logs on the server
- Use it before using PURGE BINARY LOGS

```
mysql> SHOW BINARY LOGS;
+---------------+-----------+
| Log_name      | File_size |
+---------------+-----------+
| binlog.000015 |    724935 |
| binlog.000016 |    733481 |
+---------------+-----------+
```

SHOW BINLOG EVENTS

- Used on master
- Requires REPLICATION SLAVE privileges
- Show events in binary log
- Also check `mysqlbinlog` utility

```
mysql> SHOW BINLOG EVENTS FROM 390 LIMIT 1\G
*************************** 1. row ***************************
   Log_name: slave-bin.000001
        Pos: 390
 Event_type: Query
  Server_id: 2
End_log_pos: 476
       Info: use `test`; create table t1 (a int)
1 row in set (0.00 sec)
```

SHOW SLAVE HOSTS

- Used on master
- Requires REPLICATION SLAVE privileges
- Shows list of slaves *currently registered* with the master
- Only slaves started with `report-host` option are visible

```
mysql> SHOW SLAVE HOSTS;
+-----------+-----------+------+-----------+
| Server_id | Host      | Port | Master_id |
+-----------+-----------+------+-----------+
|         2 | 127.0.0.1 | 9308 |         1 |
+-----------+-----------+------+-----------+
1 row in set (0.00 sec)
```

PURGE BINARY LOGS

- Used on master
- Requires SUPER privileges
- Removes log files before a certain log file or date
- `MASTER` can be used in place of `BINARY`
- Alternative is to use variable `EXPIRE_LOGS_DAYS`

SET SQL_LOG_BIN

- Used on master
- Requires SUPER privileges
- Session variable
- Controls logging to binary log
- Does not work for NDB!

```
mysql> SET SQL_LOG_BIN=0;
mysql> INSERT INTO t1 VALUES (1,2,3);
mysql> SET SQL_LOG_BIN=1;
```

SET GLOBAL EXPIRE_LOGS_DAYS

- Used on master
- Require SUPER privileges
- 0 means ”never expire”
- Positive value means expire logs after this many days
- Logs will be removed at startup or binary log rotation
- Can be used with running slave
- ***Logs are removed! Make sure you have backup!***

RESET MASTER

- Used on master
- Requires RELOAD privileges
- ***Deletes all binary logs in the index file!***
- Resets binary log index
- Used to get a ”clean start”
- ***Use with caution! You lose data!***

SHOW SLAVE STATUS

- Used on slave
- Requires SUPER or REPLICATION CLIENT privileges
- Shows some interesting information:

If the slave threads are running

What position the I/O thread read last

What position the SQL thread executed last

Error message and code, if thread stopped due to an error

SHOW SLAVE STATUS (5.1)

- `mysql>` **`SHOW SLAVE STATUS\G`**

```
*************************** 1. row ***************************
               Slave_IO_State:
                  Master_Host: 127.0.0.1
                  Master_User: root
                  Master_Port: 10190
                Connect_Retry: 1
              Master_Log_File:
          Read_Master_Log_Pos: 4
               Relay_Log_File: slave-relay-bin.000001
                Relay_Log_Pos: 4
        Relay_Master_Log_File:
             Slave_IO_Running: No
            Slave_SQL_Running: No
              Replicate_Do_DB:
          Replicate_Ignore_DB:
           Replicate_Do_Table:
       Replicate_Ignore_Table:
      Replicate_Wild_Do_Table:
  Replicate_Wild_Ignore_Table:
                   Last_Errno: 0
                   Last_Error:
                 Skip_Counter: 0
          Exec_Master_Log_Pos: 0
              Relay_Log_Space: 102
              Until_Condition: None
               Until_Log_File:
                Until_Log_Pos: 0
           Master_SSL_Allowed: No
           Master_SSL_CA_File:
           Master_SSL_CA_Path:
              Master_SSL_Cert:
            Master_SSL_Cipher:
               Master_SSL_Key:
        Seconds_Behind_Master: NULL
                Last_IO_Errno: 0
                Last_IO_Error:
               Last_SQL_Errno: 0
               Last_SQL_Error:
1 row in set (0.00 sec)
```

CHANGE MASTER TO

- Used on slave
- Requires SUPER privileges
- Configures the slave server connection to the master
- Slave should not be running
- The user need REPLICATION SLAVE privileges on master

```
CHANGE MASTER TO
    MASTER_HOST='adventure.com',
    MASTER_USER='dragon',
    MASTER_PASSWORD='xyzzy';
```

START SLAVE and STOP SLAVE

- Used on slave
- Used to start or stop the slave threads
- Defaults to affecting both I/O and SQL thread
- ... but individual threads can be started or stopped

START SLAVE SQL_THREAD

START SLAVE IO_THREAD

RESET SLAVE

- Used on slave
- Removes all info on replication position

  Deletes `master.info`, `relay-log.info` and all relay logs
- ***Relay logs are unconditionally removed!***

  ... even if they have not been fully applied

SET GLOBAL SQL_SLAVE_SKIP_COUNTER

- Used on slave
- Global server variable
- Requires SUPER privileges
- Slave SQL thread shall not be running
- Slave will skip events when starting
- Useful when recovering from slave stops
- Might leave master and slave with different data in tables

... so be careful when you use it

MySQL®
Conference & Expo

Presented by

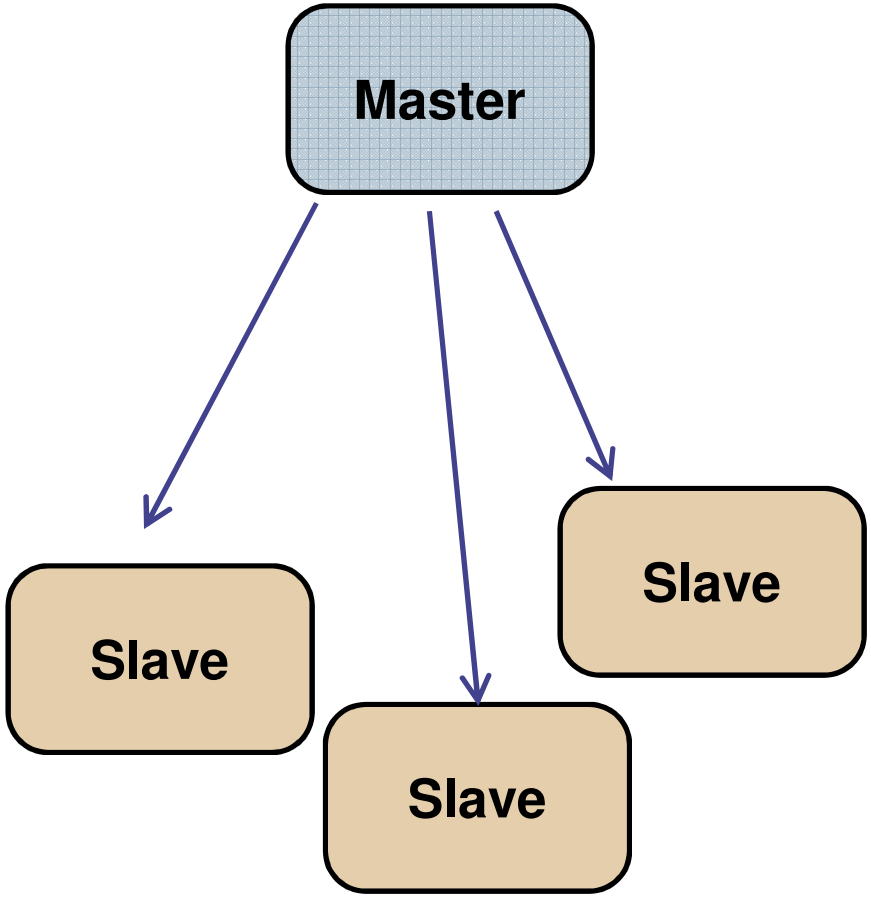
MySQL Conference & Expo

O'REILLY

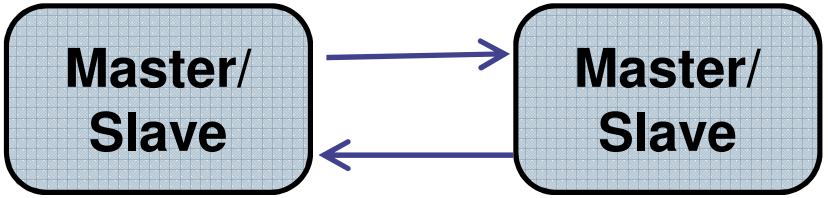
Use Cases

Use Cases, Part 1 – Basic Replication

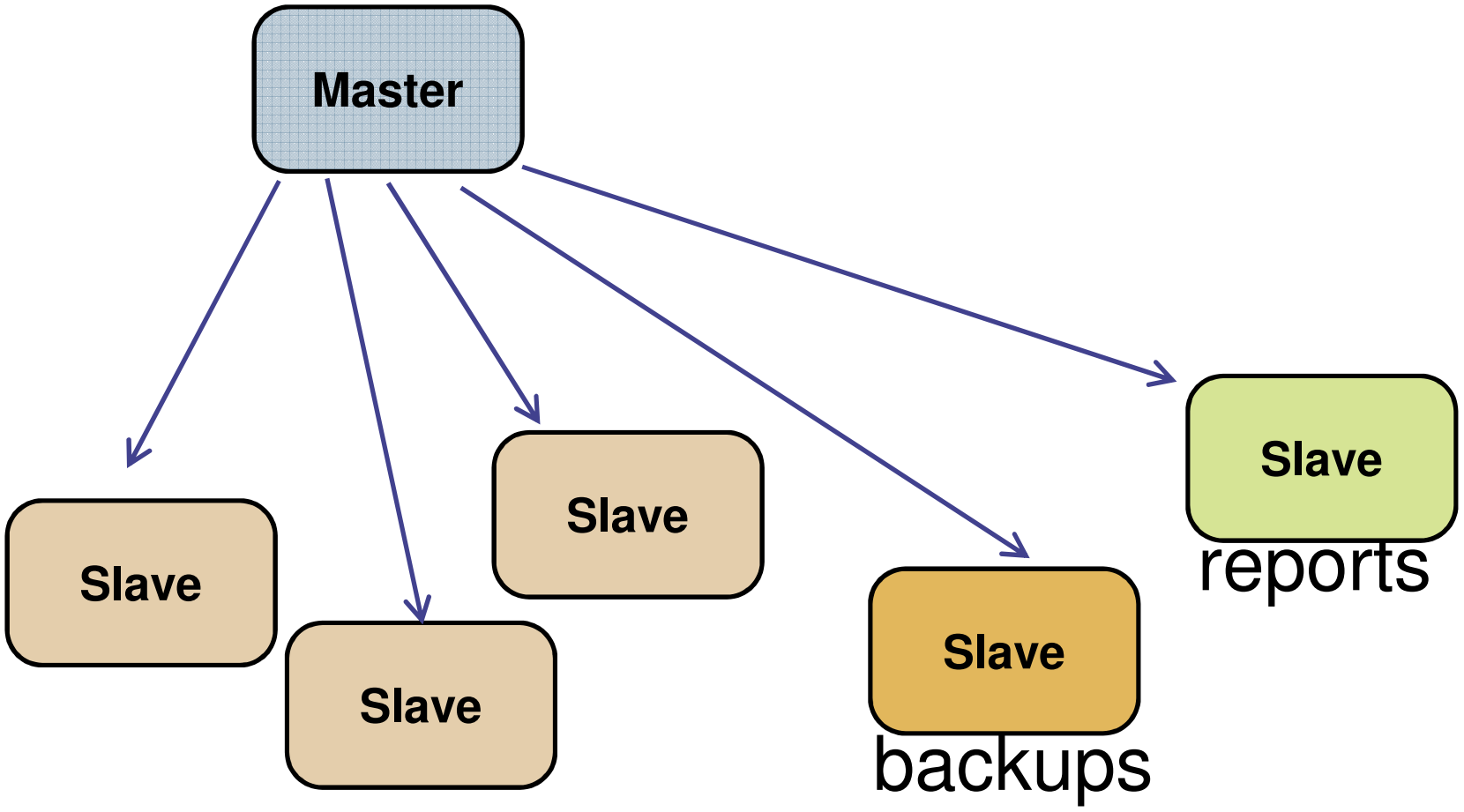
Intensive Reads



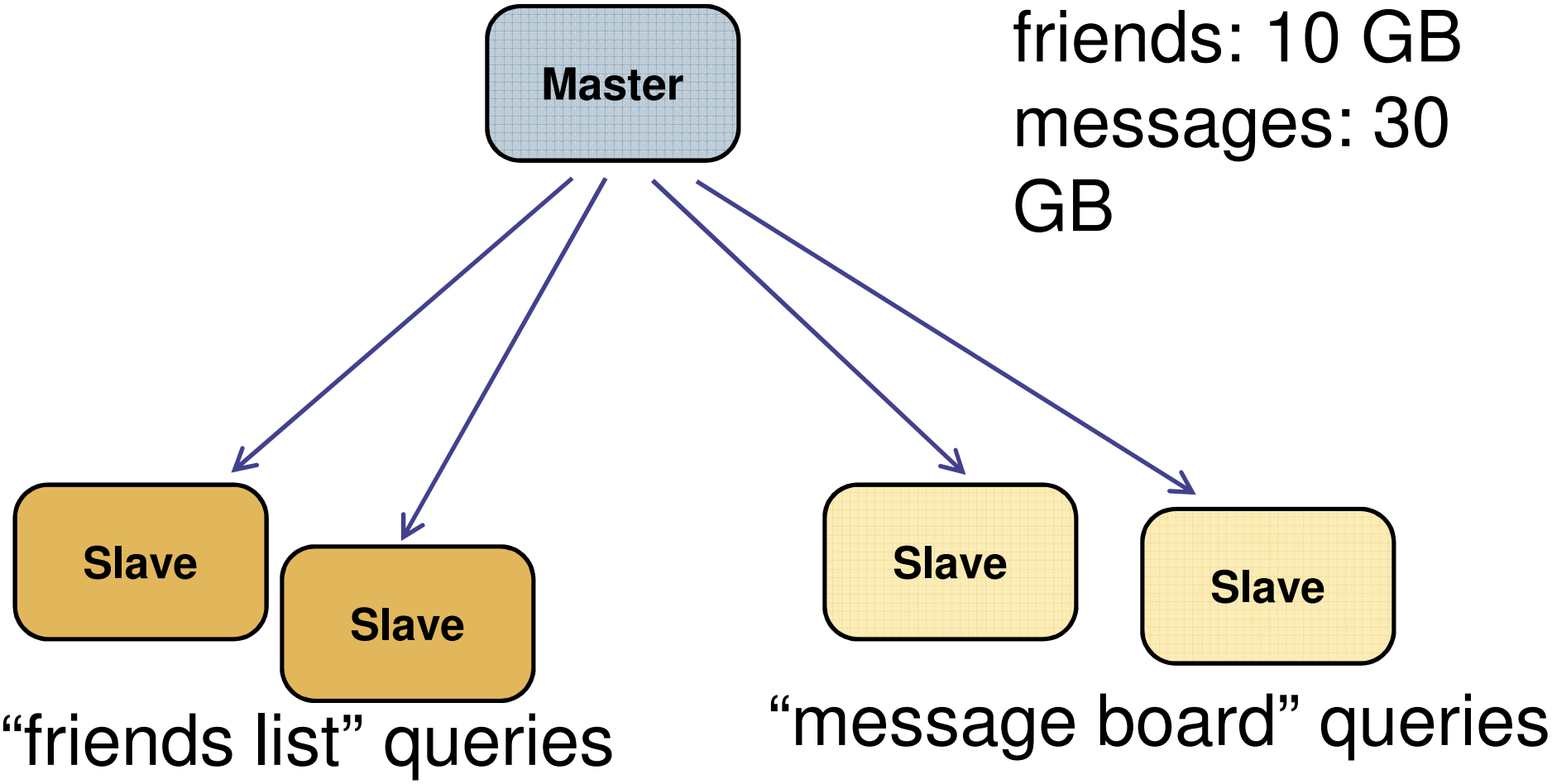
High Availability



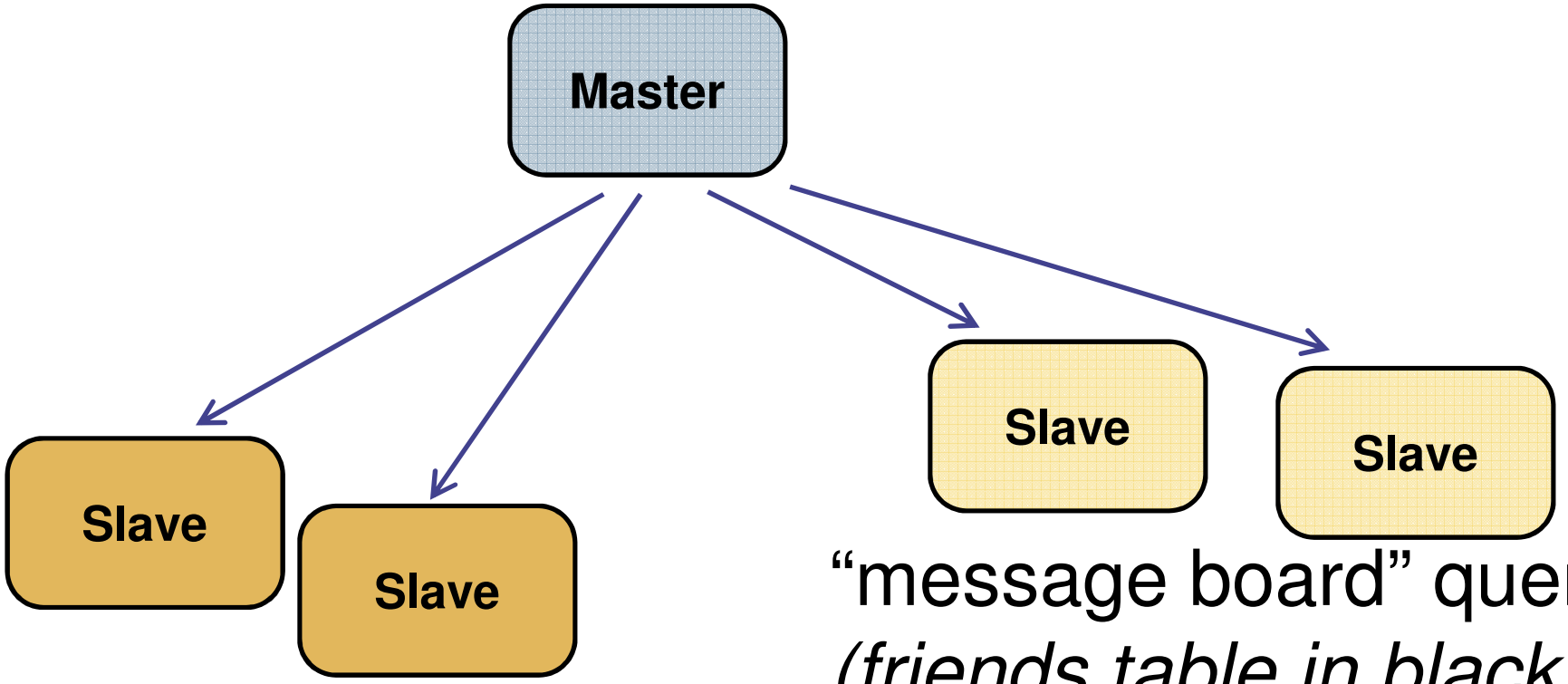
“Specialist” slaves – backups and reporting



“Specialist” slaves – per-application



“Specialist” slaves – Blackhole Engine



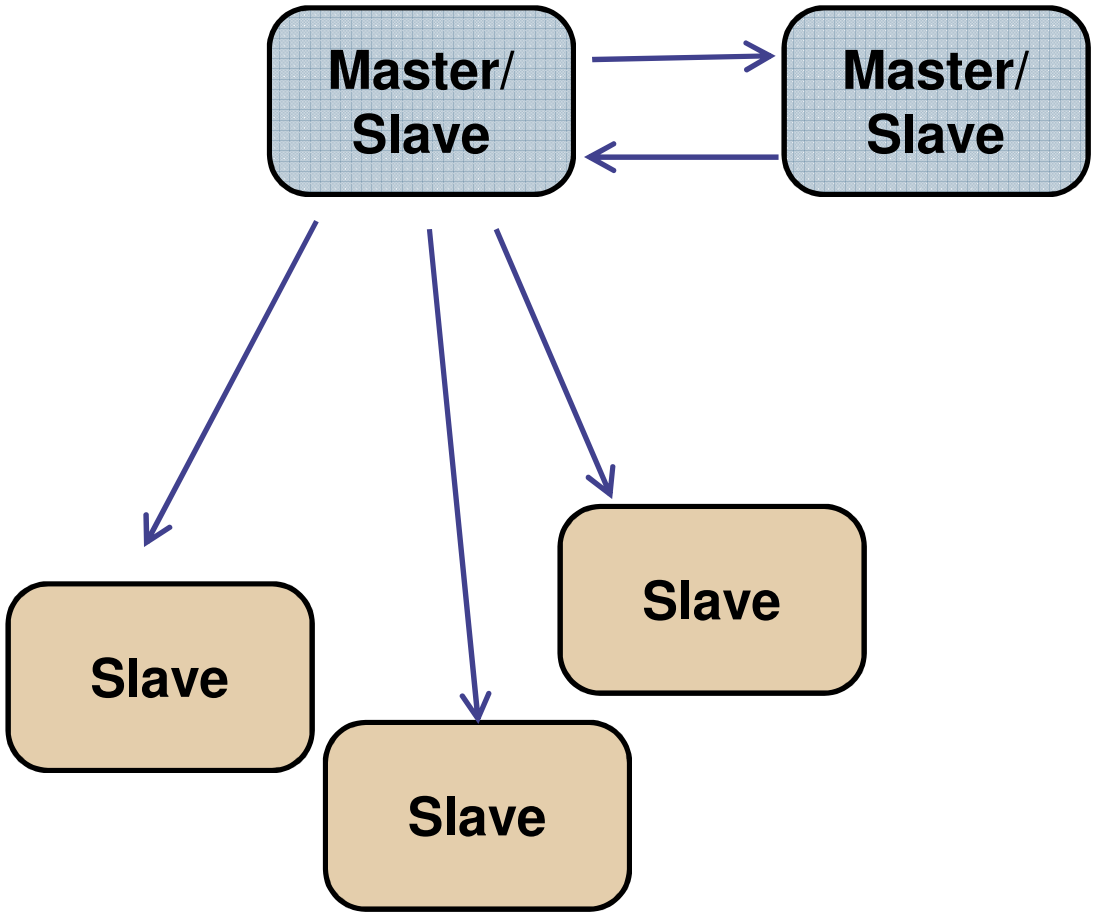
“friends list” queries *(message table in black hole)*

“message board” queries *(friends table in black hole)*

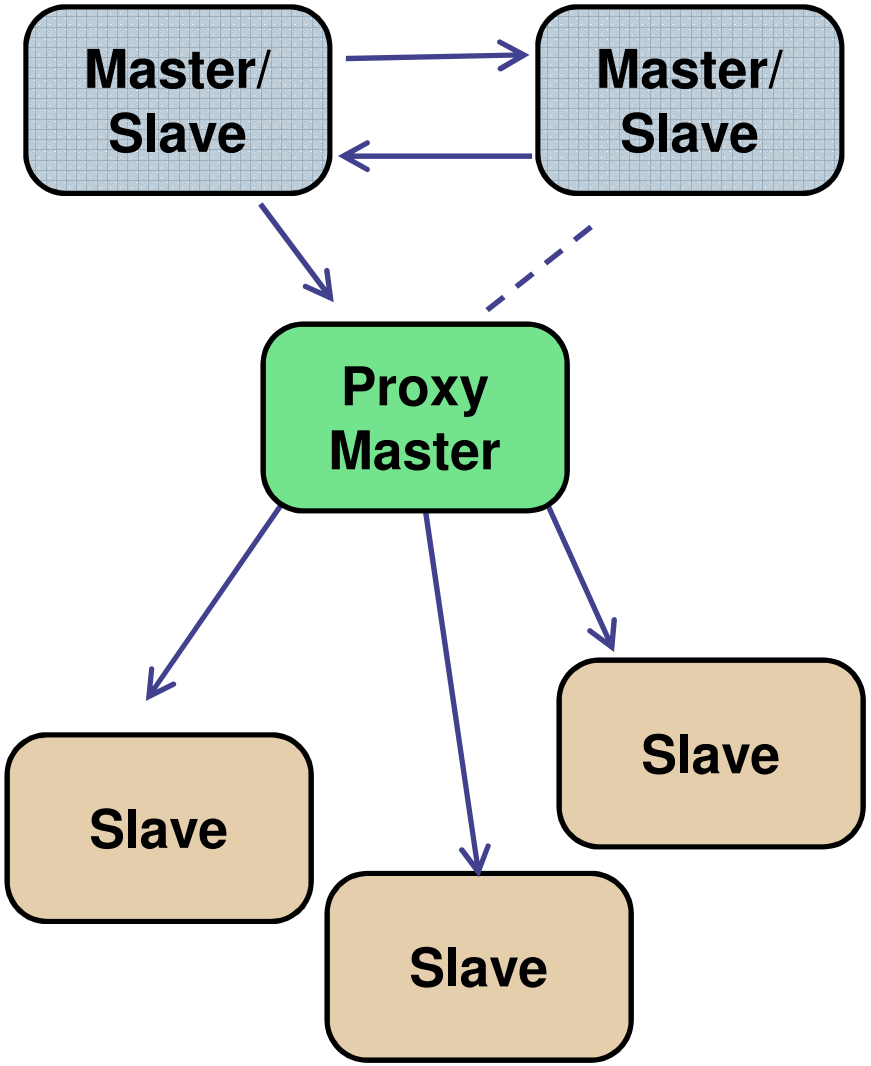
Things to think about in basic replication

- Initial snapshot of slaves
- load balancing of clients
- Failover of clients to new master

HA + Scale out?



Any better?



Problem: slave failover to a new master

- Look at SHOW SLAVE STATUS. This gives the file and position on the failed master.
- "File 34 position 6000" on the failed master may correspond to "File 33 position 22000" on the new master. Find the corresponding file and position.
- CHANGE MASTER TO

  master_host = ...

  master_log_file = ...

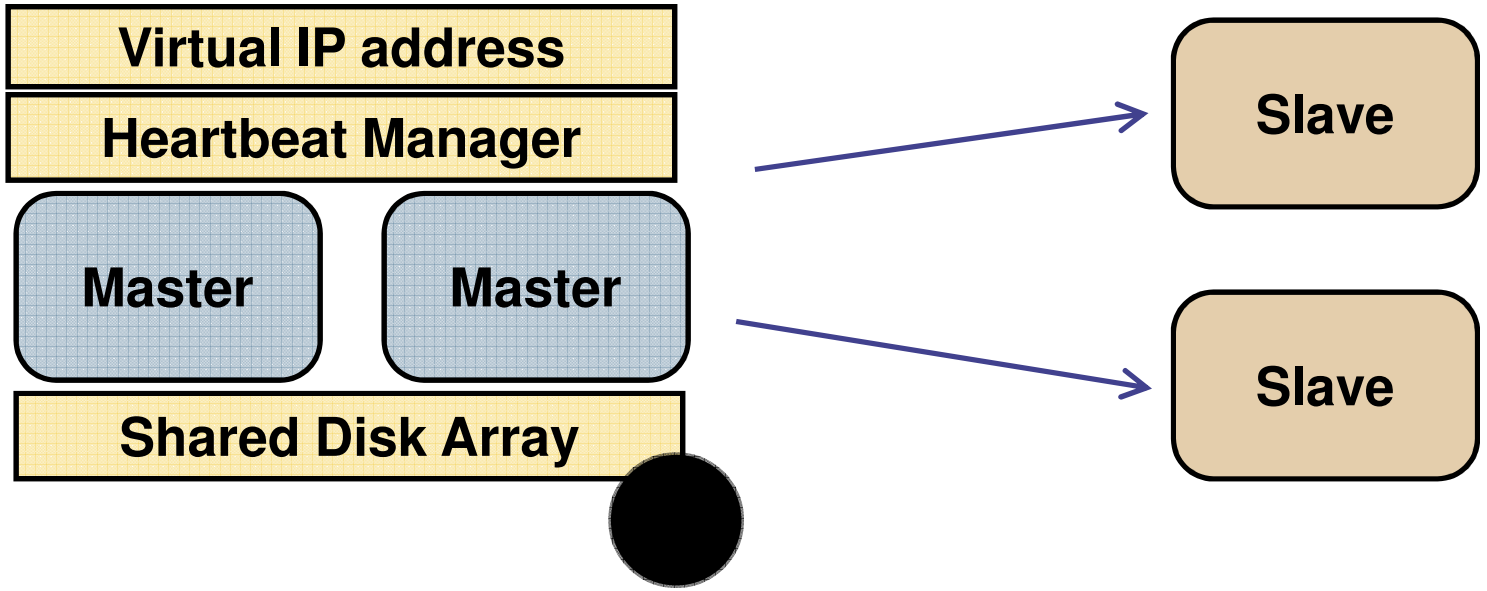
  master_log_pos = ...
- START SLAVE

Handling the failover problem

1. Automate it (scripting)
2. Avoid it

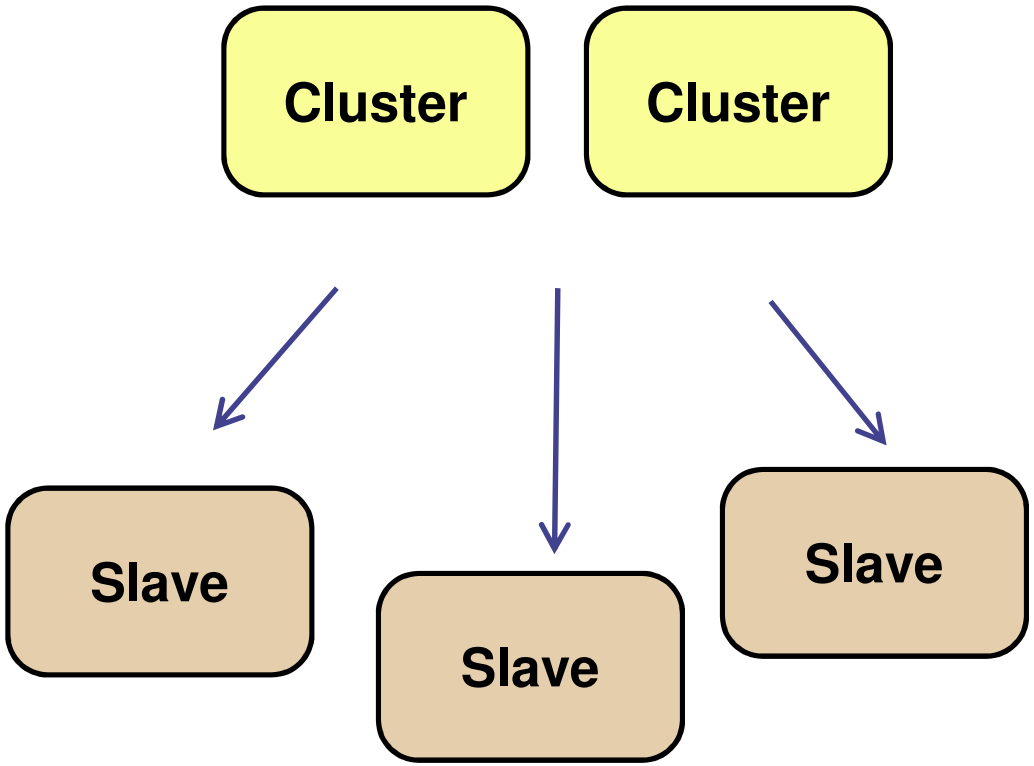
Use Cases, Part 2 – HA and Scale Out

Architecture 1: Pair of masters – Active & Standby

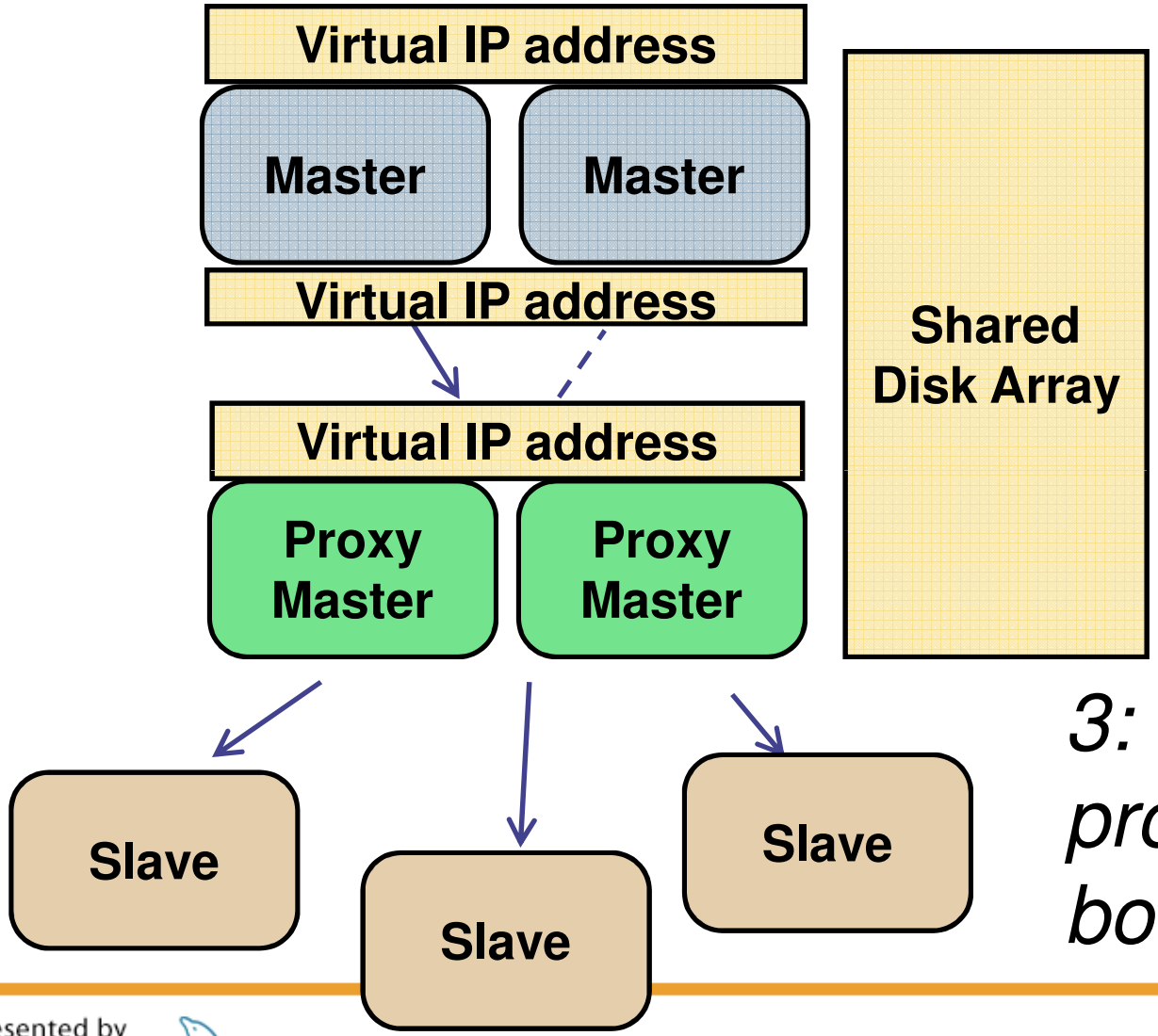


Use Cases, Part 2 – HA and Scale Out

2: MySQL Cluster as master, MySQL slaves



Use Cases, Part 2 – HA and Scale Out



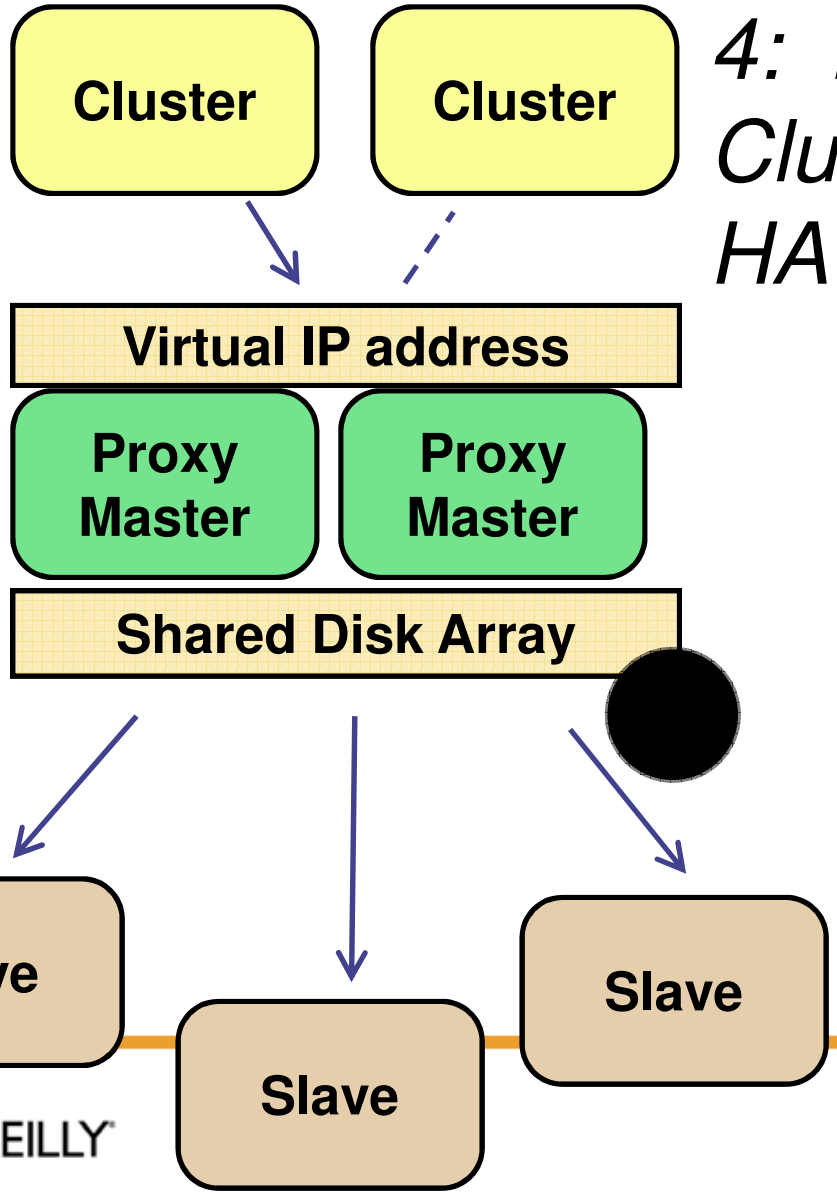
3: Master and proxy master are both HA pairs

Use Cases, Part 2 – HA and Scale Out

NDB

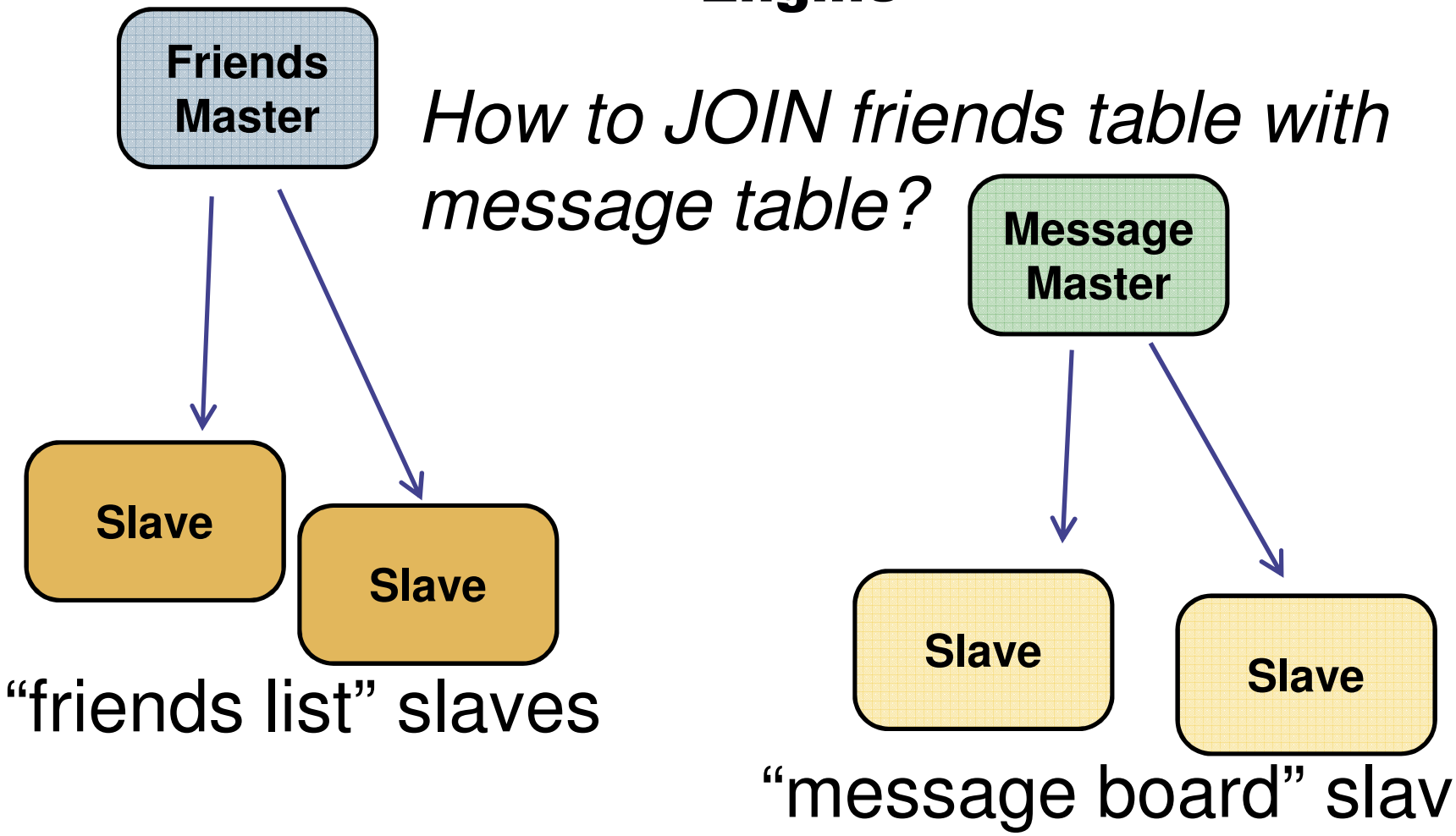
Blackhole

InnoDB

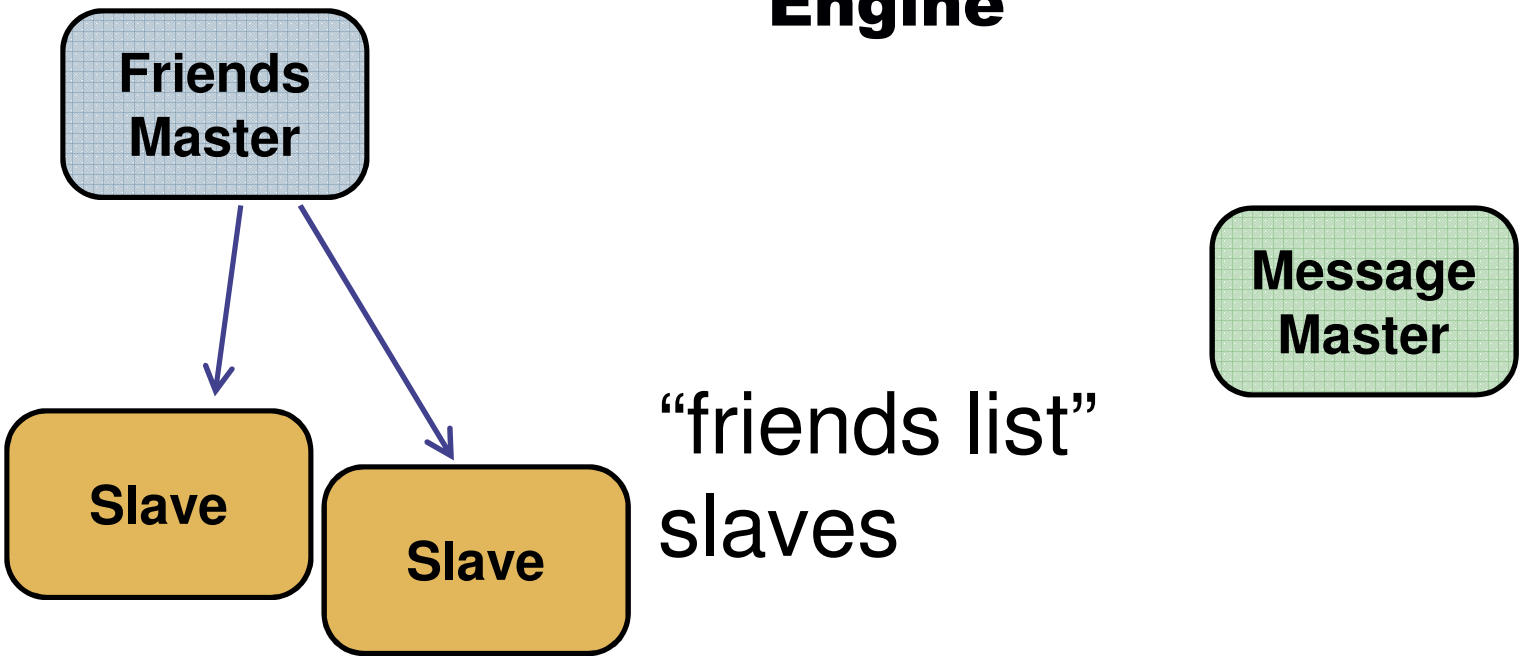


4: Replicate from Cluster through HA proxy pair

Application-level partitioning and the Federated Engine

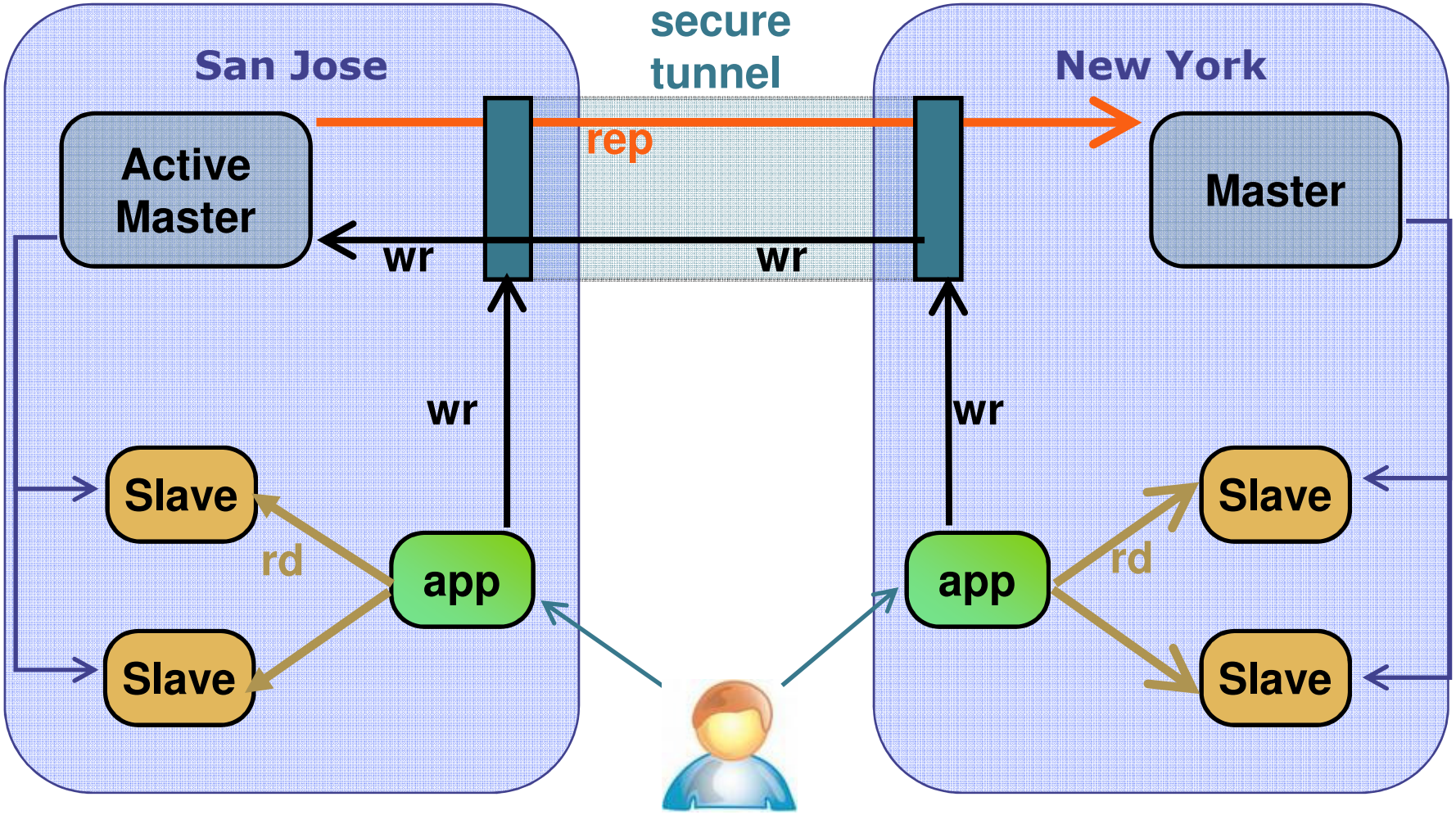


Application-level partitioning and the Federated Engine

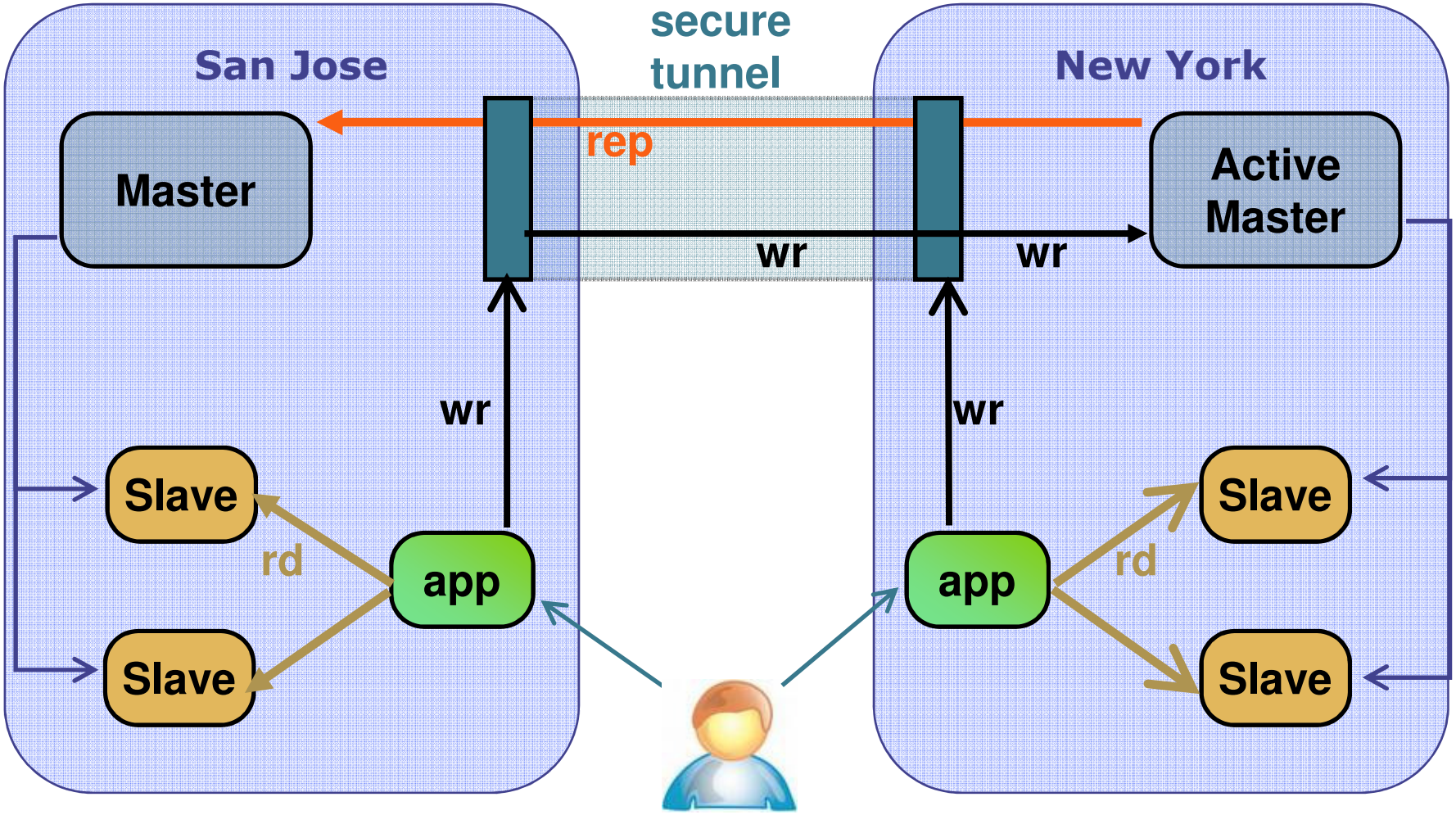



```
CREATE TABLE messages (
 id int unsigned ...
) ENGINE=FEDERATED
CONNECTION="mysql://feduser:fedpass@message-master/
            friendschema/messages";
```

Use Cases, Part 3 – Multiple Data Centers



After Failover



MySQL®
Conference & Expo

Presented by

MySQL Conference & Expo

O'REILLY

Row-based replication

Row-based replication (MySQL 5.1)

- **Statement-based replication**
  - Replicate statement doing changes
  - Requires up-to-date slave
  - Requires determinism
- **Row-based replication**
  - Replicate actual row changes
  - Does not require up-to-date slave
  - Can handle any statement

Comparison of replication methods

- **Row-based replication**
  - Can handle ”difficult” statements
  - Required by cluster
- **Statement-based replication**
  - Sometimes smaller binary log
  - Binary log can be used for auditing

Row-based replication features

- **Log is idempotent**

  ... provided all tables in log have primary key

- **Statement events and row events can be mixed in log**

  ... so format can be switched during run-time

  (slave switches automatically as required)

  ... and even different formats for different threads

Row-based replication as a foundation

- Conflict detection and conflict resolution
- Fine-grained filtering
- NDB Cluster replication
- Multi-channel replication
- Horizontal partitioning

  ... sending different rows to different slaves

Filtering

- **For statement-based replication:**

  Statements are filtered

  Filtering is based on current (used) database

  Master filtering are on database only

- **For row-based replication:**

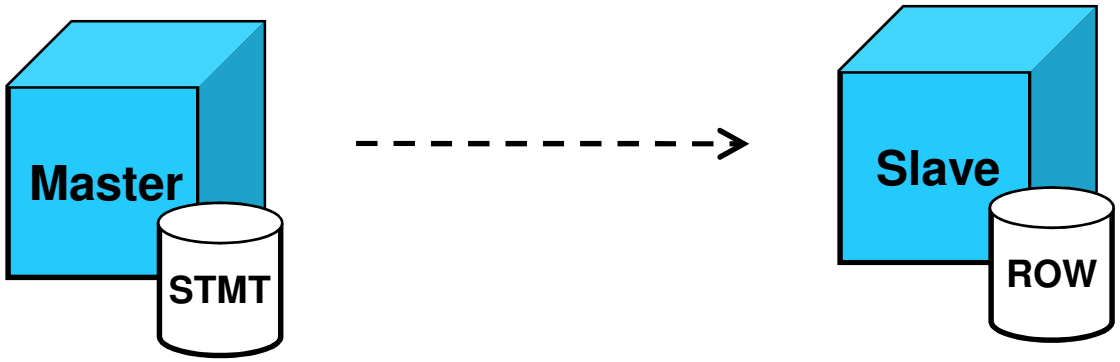
  Rows are filtered

  Filtering is based on actual database and table

  Master filtering for individual tables possible

  ... but not implemented

Want both statement and row format?



- Master in STATEMENT mode, slave in ROW mode
- Slave converts statements executed into row format
- Once in row format, it stays in row format

MySQL® Conference & Expo

Presented by MySQL Conference & Expo O'REILLY

Binary Log

Modes and Formats of the Binary Log

Logging modes

- Three modes: `STATEMENT`, `MIXED`, and `ROW`
- Server variable `BINLOG_FORMAT` controls mode
- Mode is used to decide *logging format* for statements

Logging format is representation of changes

More about that in just a bit

SET BINLOG_MODE

- `SET BINLOG_FORMAT=`*`mode`*

- Session and global variable
- *Mode* is one of `STATEMENT`, `ROW`, or `MIXED`
- `STATEMENT`: statements are logged in statement format
- `ROW`: statements are logged in row format
- `MIXED` (default)

  Statements are logged in statement format by default

  Statements are logged in row format in some cases

Switching modes

- Mode can be switched at run-time

  ... even inside a transaction

- Switching mode is *not* allowed:

  If session has open temporary tables

  From inside stored functions or triggers

  If ‘ndb’ is enabled

MIXED mode

- Safe statements are usually logged in statement format
- Unsafe statements are logged in row format
- Heuristic decision on what is unsafe, currently:

Statement containing `UUID()` or calls to UDFs

Statements updating >1 table with auto-increment columns

`INSERT DELAYED` statements

problems with `RAND()` and user-defined variables

Binary logging formats

- The *format* tells how changes are stored in log
- Two formats: statement and row
- Formats can be mixed in binary log

```
mysql> show binlog events;
+----------+-----+-------------+---+--------------------------------------+
| Log_name | Pos | Event_type  | … | Info                                 |
+----------+-----+-------------+---+--------------------------------------+
| ...      |   4 | Format_desc | … | Server ver: 5.1.17-beta-debug-log...  |
| ...      | 105 | Query       | … | use `test`; CREATE TABLE tbl (a INT)  |
| ...      | 199 | Query       | … | use `test`; INSERT INTO tbl VALUES (1) |
| ...      | 290 | Table_map   | … | table_id: 16 (test.tbl)               |
| ...      | 331 | Write_rows  | … | table_id: 16 flags: STMT_END_F        |
+----------+-----+-------------+---+--------------------------------------+
5 rows in set (0.00 sec)
```

Statement logging format

- The *statement executed* is logged to the binary log
- Statement logged *after* statement has been executed
- Pro:

  Usually smaller binary logs

  Binary log can be used for auditing
- Cons:

  Cannot handle partially executed statements

  Cannot handle non-deterministic data

  Does not work with all engines (e.g., NDB)

Row logging format

- The actual rows being changed are logged
- Rows are grouped into events
- Pro:

  Can handle non-deterministic statements

  Can handle UDF execution

  Idempotent
- Cons:

  No easy way to see what rows are logged

  Does not work with all engines (e.g., blackhole)

Example: multi-table update

- `UPDATE t1,t2 SET t1.b = ..., t2.b = ...`

```
mysql> show binlog events from 480;
+----------+-----+-------------+---+-------------------------------------+
| Log_name | Pos | Event_type  | … | Info                                |
+----------+-----+-------------+---+-------------------------------------+
| ...      | 480 | Table_map   | … | table_id: 16 (test.t1)              |
| ...      | 520 | Table_map   | … | table_id: 17 (test.t2)              |
| ...      | 560 | Update_rows | … | table_id: 16                        |
| ...      | 625 | Update_rows | … | table_id: 17 flags: STMT_END_F      |
+----------+-----+-------------+---+-------------------------------------+
4 rows in set (0.00 sec)
```

Example: CREATE-SELECT

- `CREATE t3 SELECT * FROM t1`

```
mysql> show binlog events from 690;
+----------+-----+-------------+---+-------------------------------------+
| Log_name | Pos | Event_type  | … | Info                                |
+----------+-----+-------------+---+-------------------------------------+
| ...      | 480 | Table_map   | … | use `test`; CREATE TABLE `t3` (
a INT(11) DEFAULT NULL,
b INT(11) DEFAULT NULL
) |
| ...      | 520 | Table_map   | … | table_id: 18 (test.t3)              |
| ...      | 625 | Write_rows  | … | table_id: 18 flags: STMT_END_F      |
+----------+-----+-------------+---+-------------------------------------+
3 rows in set (0.00 sec)
```

Special cases

- `TRUNCATE` vs. `DELETE` in row mode

  `TRUNCATE` is logged in statement format

  `DELETE` is logged in row format

- `GRANT`, `REVOKE`, and `SET PASSWORD`

  These statements changes rows in `mysql` tables:

  `tables_priv`, `columns_priv`, and `user`

  Replicated in statement format

  Other statements on these tables are replicated in row format

How objects are logged

- Databases
- Tables
- Views
- Stored procedures
- Stored functions
- Triggers
- Events
- Users

We are here only considering how these objects are logged when using row mode

For statement mode, everything is logged in statement format

Databases and Tables

- **Database manipulation statements**

  Logged in statement format

- **Table manipulation statements**

  Statement format: `CREATE`, `ALTER`, and `DROP`

  Row format: `INSERT`, `DELETE`, `UPDATE`, etc.

Views

- `CREATE`, `ALTER`, and `DROP` logged in statement format
- Changes are logged by logging changes to the tables

```
mysql> UPDATE living_in SET name='Matz' WHERE name='Mats';
Query OK, 1 row affected (0.00 sec)
Rows matched: 1  Changed: 1  Warnings: 0

mysql> show binlog events from 1605;
+----------+------+-------------+-----+-------------------------------+
| Log_name | Pos  | Event_type  | ... | Info                          |
+----------+------+-------------+-----+-------------------------------+
| maste... | 1605 | Table_map   | ... | table_id: 17 (test.names)     |
| maste... | 1648 | Update_rows | ... | table_id: 17 flags: STMT_END_F |
+----------+------+-------------+-----+-------------------------------+
2 rows in set (0.01 sec)
```

Stored procedures

- `CREATE`, `ALTER`, and `DROP` are replicated in statement format (with a `DEFINER`)
- `CALL` is logged in row format by logging all changes done by the call

```
mysql> create procedure foo(a int) insert into t1 values(a);
mysql> show binlog events from 102\G
*************************** 1. row ***************************
   Log_name: master-bin.000001
        Pos: 102
 Event_type: Query
  Server_id: 1
End_log_pos: 244
       Info: use `test`; CREATE DEFINER=`root`@`localhost` procedure foo(a
int) insert into t1 values(a)
1 row in set (0.00 sec)
```

Stored functions

- `CREATE`, `ALTER`, and `DROP` are replicated in statement format (with a `DEFINER`)
- The effects of calling a stored function are logged in row format

```
mysql> select a, bar(a) from t2;
mysql> show binlog events from 557;
+----------+-----+------------+-----+-------------------------------+
| Log_name | Pos | Event_type | ... | Info                          |
+----------+-----+------------+-----+-------------------------------+
| maste... | 557 | Table_map  | ... | table_id: 18 (test.t1)        |
| maste... | 596 | Write_rows | ... | table_id: 18 flags: STMT_END_F |
+----------+-----+------------+-----+-------------------------------+
2 rows in set (0.01 sec)
```

Triggers

- `CREATE`, `ALTER`, and `DROP` are replicated in statement format (with a `DEFINER`)
- The effects of a trigger are logged in row format

```
mysql> insert into t1 values (1,2);
mysql> show binlog events from 780;
+----------+-----+-------------+---+---------------------------------------+
| Log_name | Pos | Event_type  | … | Info                                  |
+----------+-----+-------------+---+---------------------------------------+
| ...      | 780 | Table_map   | … | table_id: 16 (test.t1)                |
| ...      | 820 | Table_map   | … | table_id: 17 (test.t2)                |
| ...      | 860 | Write_rows  | … | table_id: 16                          |
| ...      | 925 | Write_rows  | … | table_id: 17 flags: STMT_END_F        |
+----------+-----+-------------+---+---------------------------------------+
4 rows in set (0.00 sec)
```

Events

- `CREATE`, `ALTER`, and `DROP` are replicated in statement format (with a `DEFINER`)
- *The event is disabled on the slave*
- Effects of a event are logged in row format

MySQL® Conference & Expo

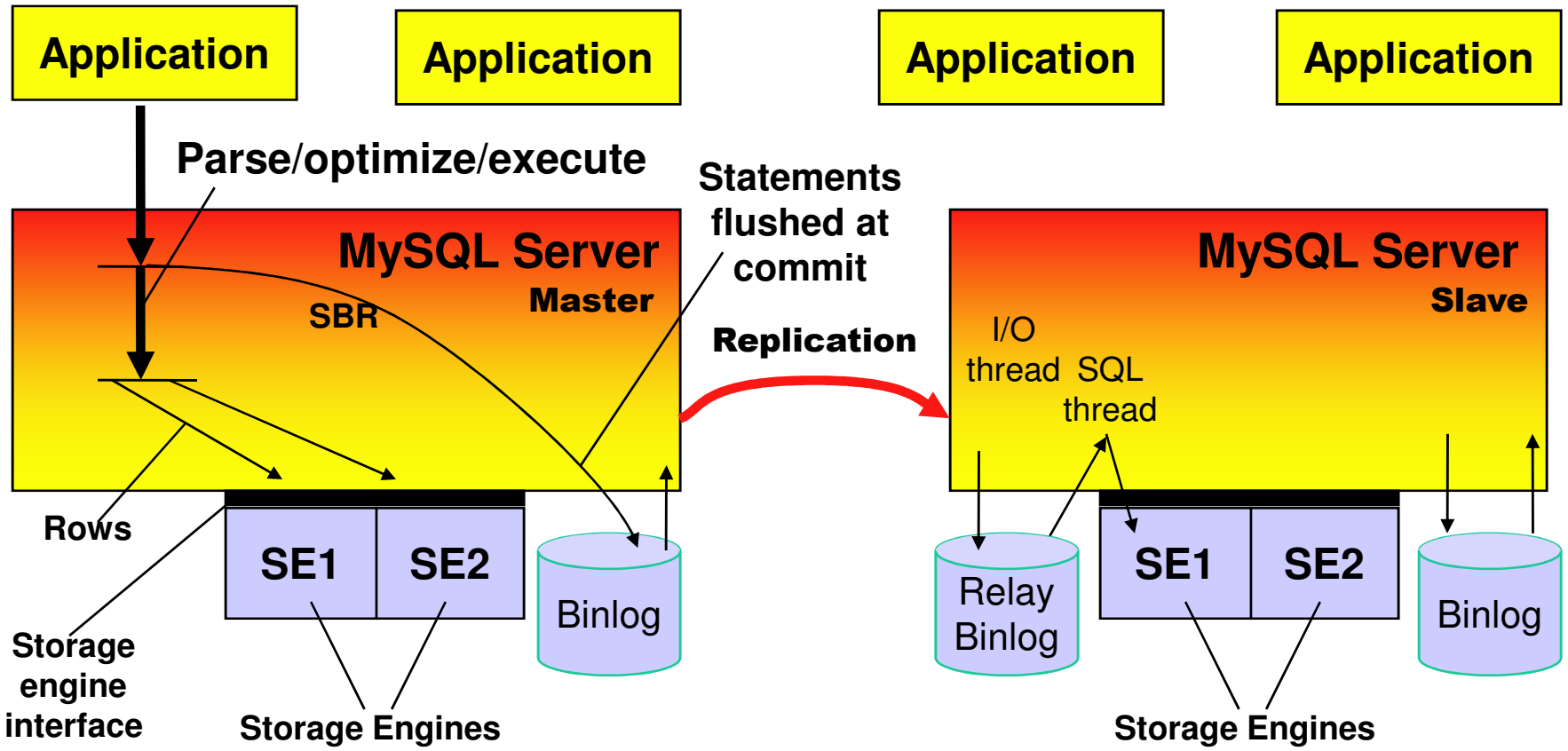
Presented by MySQL Conference & Expo O'REILLY

Implementation

How replication works

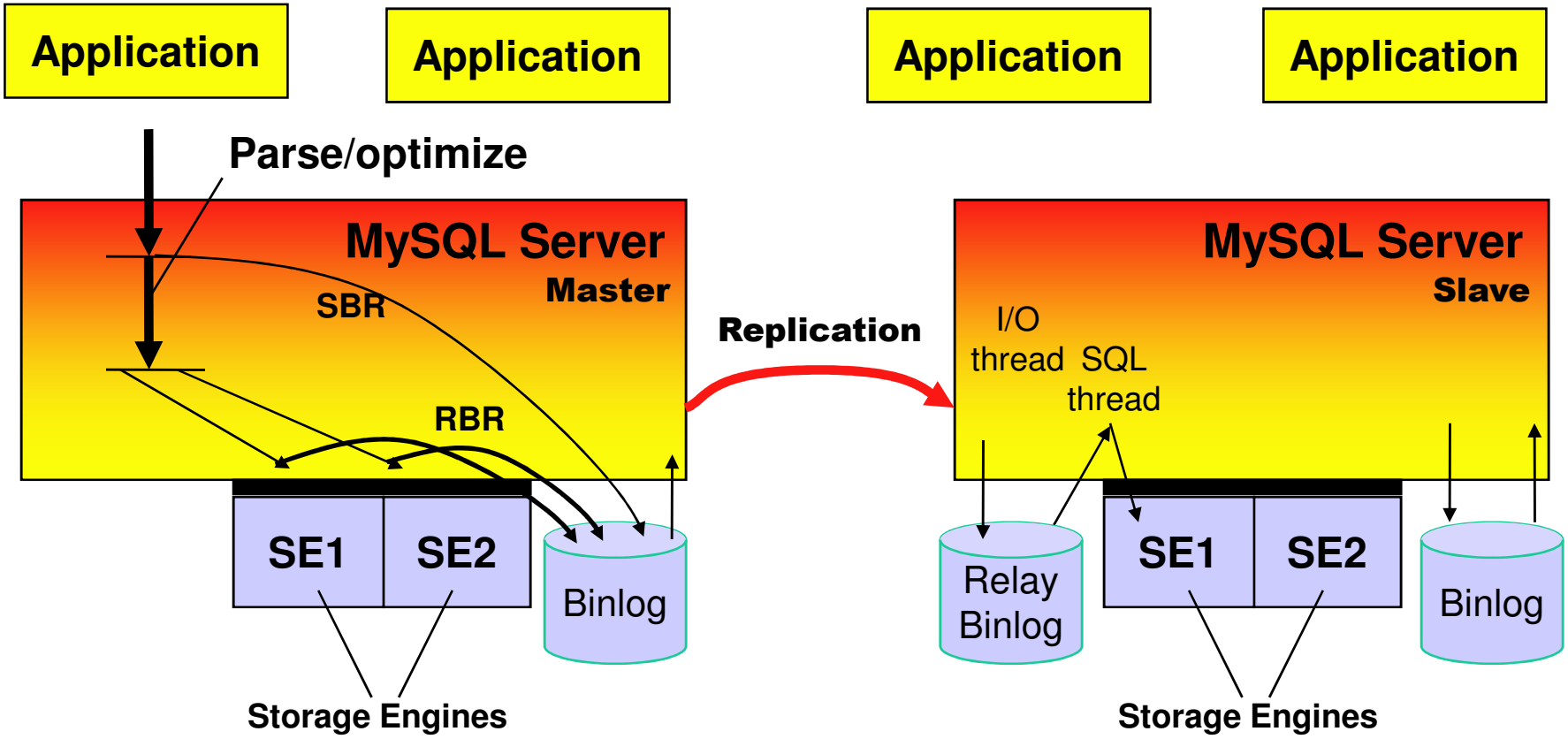
MySQL Replication Architecture

MySQL 4.0-5.0



MySQL Replication Architecture

MySQL 5.1: Row-based replication (RBR)



Row-based Replication
Comparision between SBR and RBR

Advantages of Row-based Replication (RBR)

- Can replicate non-deterministic statements (e.g. UDFs, LOAD_FILE(), UUID(), USER(), FOUND_ROWS())
- Makes it possible to replicate between MySQL Clusters (having multiple MySQL servers or using NDB API)
- Less execution time on slave
- Simple conflict detection (that is currently being extended)

Advantages of Statement-based Replication (SBR)

- Proven technology (since MySQL 3.23)
- Sometimes produces smaller log files
- Binary log can be used for auditing

Four new binlog events

1.Table map event

–Semantics: “This table id matches this table definition”

2.Write event (After image)

–Semantics: “This row shall exist in slave database”

3.Update event (Before image, After image)

–Semantics: “This row shall be changed in slave database”

4.Delete event (Before image)

–Semantics: “This row shall not exist in the slave database”

Various optimizations:

- **Only primary key in before image.** Works if table has PK
- **Only changed column values in after image.** Works if table has PK

Log is *idempotent* if PK exists and there are only RBR events in log.

Slave can execute both SBR and RBR events.

MySQL®
Conference & Expo

Presented by

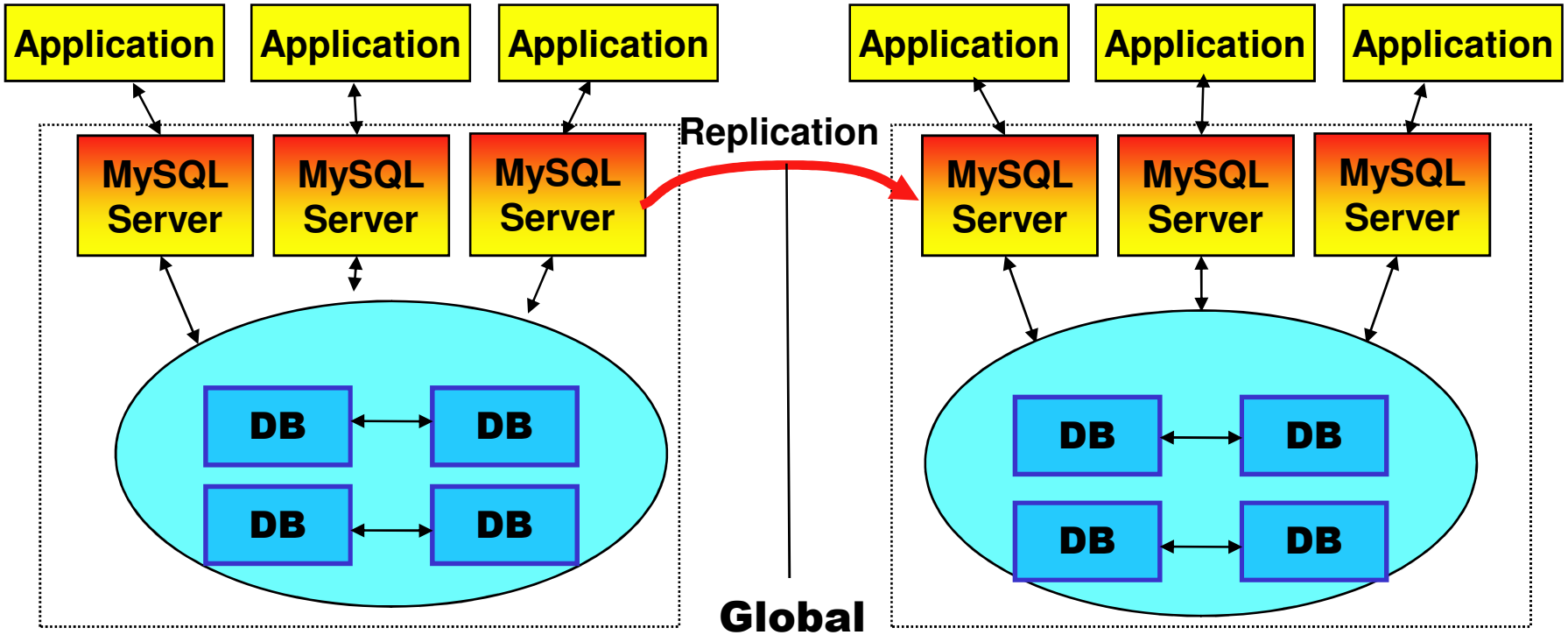
MySQL Conference & Expo

O'REILLY

Cluster Replication

MySQL Cluster Replication

Local and Global Redundancy



MySQL®
Conference & Expo

Presented by

MySQL Conference & Expo

O'REILLY

Tools and Techniques

Making a snapshot from a master database

- This is necessary for bringing new slaves online.
- Options:

Shut down master & take offline backup

Use “ibbackup” to make an online physical backup

www.innodb.com

Use `mysqldump --master-data`

Table Checksums

- How do you know the slave really has the same data as the master?
- Guiseppe Maxia

Taming the Distributed Data Problem – MySQL Users Conf 2003

- Baron Schwartz

MySQL Table Checksum

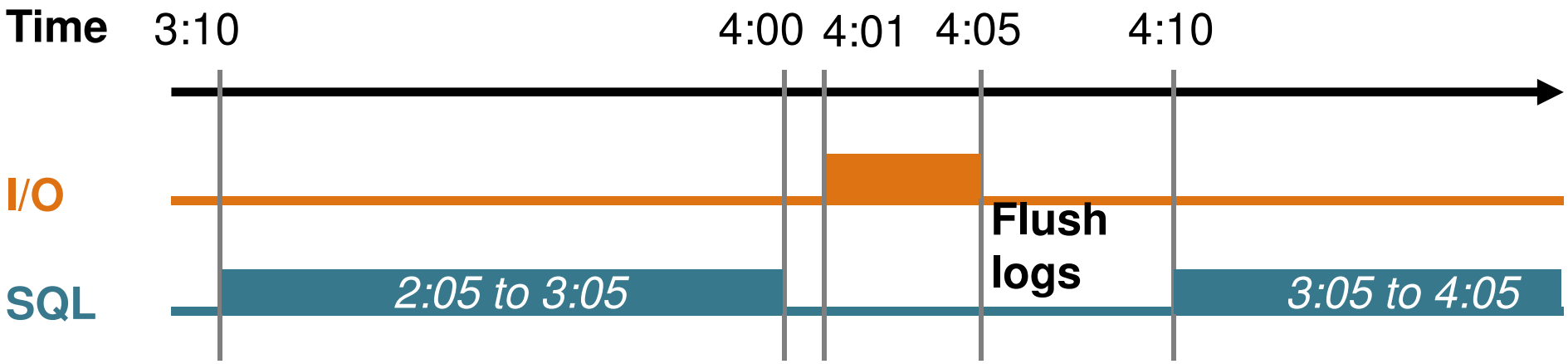
http://sourceforge.net/projects/mysqltoolkit

“Delayed Replication”

- Bruce Dembecki, LiveWorld

  Lessons from an Interactive Environment – MySQL Users Conf 2005

- Provides hourly log snapshots and protection against “user error” (*e.g.* `DELETE FROM important_table`)



Managing Virtual IP addresses

- Fof failover and high availability. (Always prefer virtual IP addresses rather than DNS changes)
- **Heartbeat** – *www.linux-ha.org*

  also runs on Solaris, BSD, Mac OS X
- Several other software alternatives

  Sun Cluster, HP ServiceGuard, etc.
- Or a hardware load balancer

  F5 Big IP, Foundry ServerIron, etc.

Shared Storage for Active/Standby pairs

- **DRBD**

  www.drbd.org

- **Hardware SAN**
- **Hardware NAS**

  NetApp

Tunnels & proxies to use for managing multiple data centers

- Master & slaves can use SSL
- ... or offload the SSL processing to other servers using **stunnel**

www.stunnel.org

- Proxy writes to masters as in Jeremy Cole's example

TCP Proxy software

Hardware load balancer

References

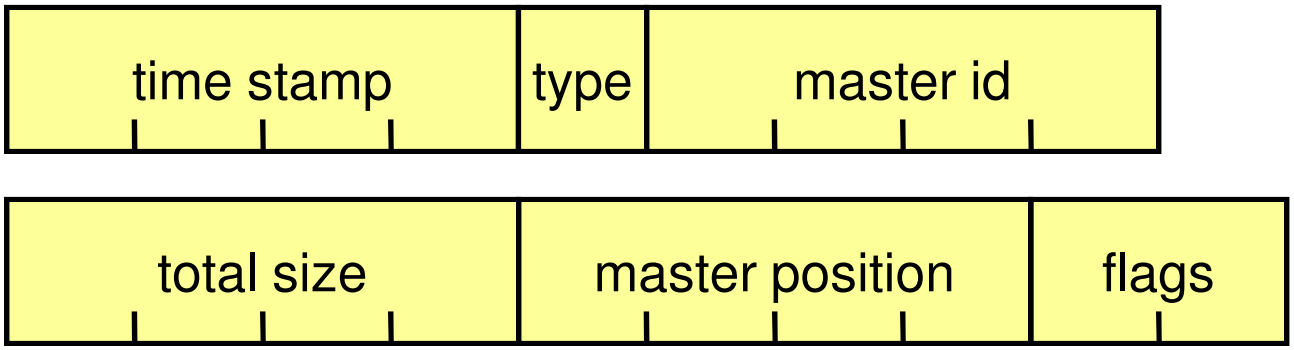
- **MySQL Manual (http://dev.mysql.com/doc/)**
  Chapter: Replication
- **MySQL Manual (http://dev.mysql.com/doc/)**
  Chapter: MySQL Cluster Replication
- **MySQL Forums (http://forums.mysql.com/)**
  MySQL Replication forum

- **Replication Tricks and Tips**
  Tuesday 4:25pm
- **BOF: Replication**
  Tuesday evening, first slot (probably 7:30pm)

lars@mysql.com, mats@mysql.com
www.mysql.com

Common Event Header – 19 bytes

Field	Length	Description
Timestamp	4 bytes	Seconds since 1970
Type	1 byte	Event type
Master Id	4 bytes	Server Id of server that created this event
Total size	4 bytes	Event total size in bytes
Master position	4 bytes	Position of next event in master binary log
Flags	2 bytes	Flags for event



Statement-based INSERT 1/2: Query event header

$ mysqlbinlog --hexdump master-bin.000001

```
# at 235
#060420 20:16:02 server id 1  end_log_pos 351
# Position         Timestamp      Type      Master ID
# 000000eb       e2 cf 47 44      02      01 00 00 00
#      Size       Master Pos    Flags
# 74 00 00 00    5f 01 00 00    10 00
```

Statement-based INSERT 2/2: Query event data

$ mysqlbinlog --hexdump master-bin.000001

```
# 000000fe 02 00 00 00 00 00 00 00
#          04 00 00 1a 00 00 00 40 |................|
# 0000010e 00 00 ...               |............std|
# 0000011e 04 08 ...               |.......test.INSE|
# 0000012e 52 54 ...               |RT.INTO.t1.VALUE|
# 0000013e 53 20 ...               |S...A...B......X|
# 0000014e 27 2c ...               |...Y......X...X.|
# 0000015e 29                      |.|
# Query   thread_id=2     exec_time=0     error_code=0

SET TIMESTAMP=1145556962;
INSERT INTO t1 VALUES ('A','B'), ('X','Y'), ('X','X');
```

Row-based INSERT 1/2: Table map event

$ mysqlbinlog --hexdump master-bin.000001

```
# at 235
#060420 20:07:01 server id 1  end_log_pos 275
# Position          Timestamp      Type     Master ID
# 000000eb        c5 cd 47 44      13     01 00 00 00
#      Size          Master Pos      Flags
# 28 00 00 00     13 01 00 00     00 00
# 000000fe 0f 00 00 00 00 00 00 00
            04 74 65 73 74 00 02 74 |.........test..t|
# 0000010e 31 00 02 fe fe           |1....|
# Table_map: `test`.`t1` mapped to number 15
BINLOG 'xc1HRBMBAAAAKAAAABMBA...3QAAnQxAAL+/g==';
```

Row-based INSERT 2/2: Write event

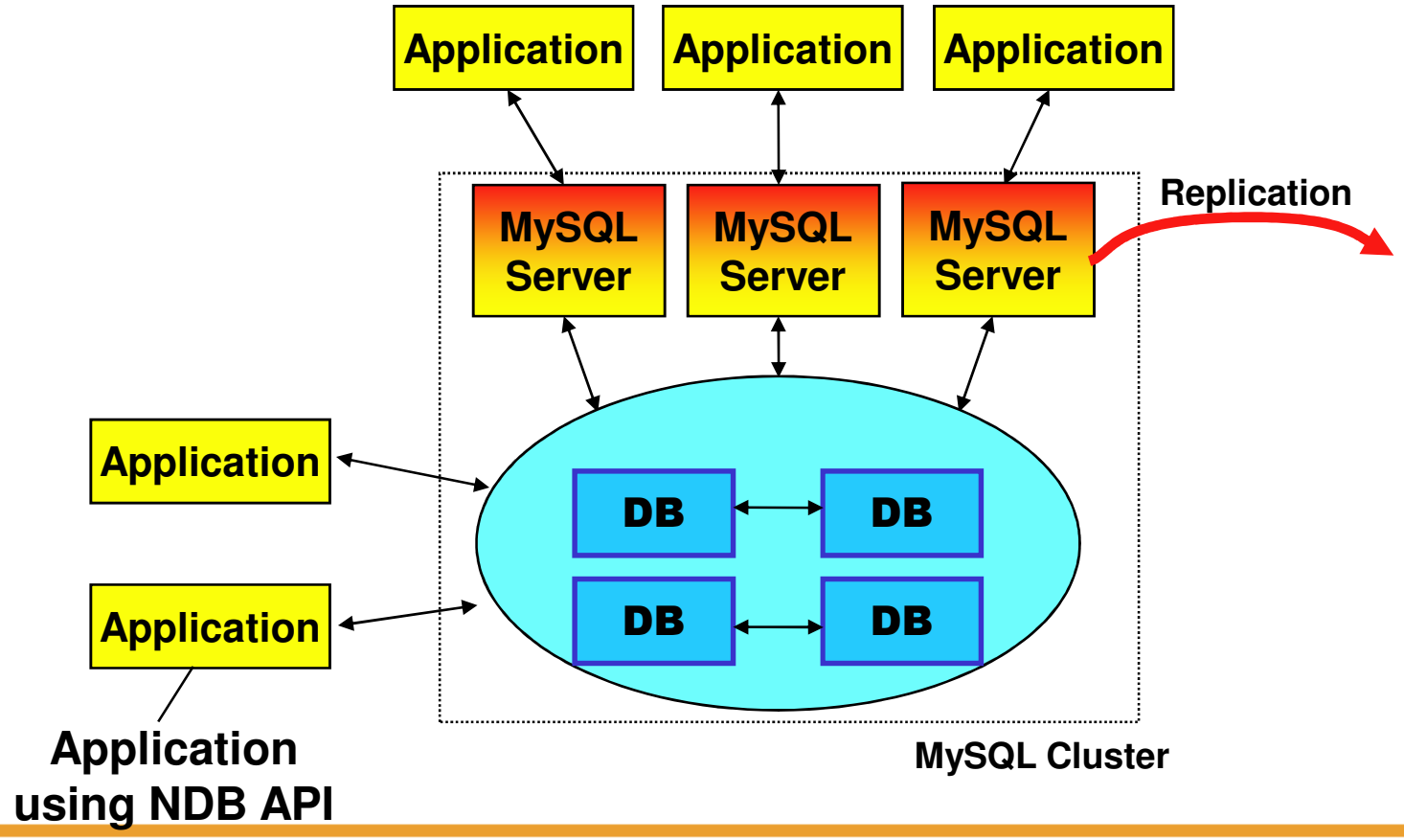
$ mysqlbinlog --hexdump master-bin.000001

```
# at 275
#060420 20:07:01 server id 1  end_log_pos 319
# Position         Timestamp      Type     Master ID
# 00000113         c5 cd 47 44     14     01 00 00 00
#     Size         Master Pos      Flags
# 2c 00 00 00      3f 01 00 00     10 00
# 00000126 0f 00 00 00 00 00 01 00
#          02 ff f9 01 41 01 42 f9 |............A.B.|
# 00000136 01 58 01 59 f9 01 58 01
#          58                     |.X.Y..X.X|
# Write_rows: table id 15

BINLOG 'xc1HRBQBAAAALAAAAD...EBQvkBWAFZ+QFYAVg=';
```

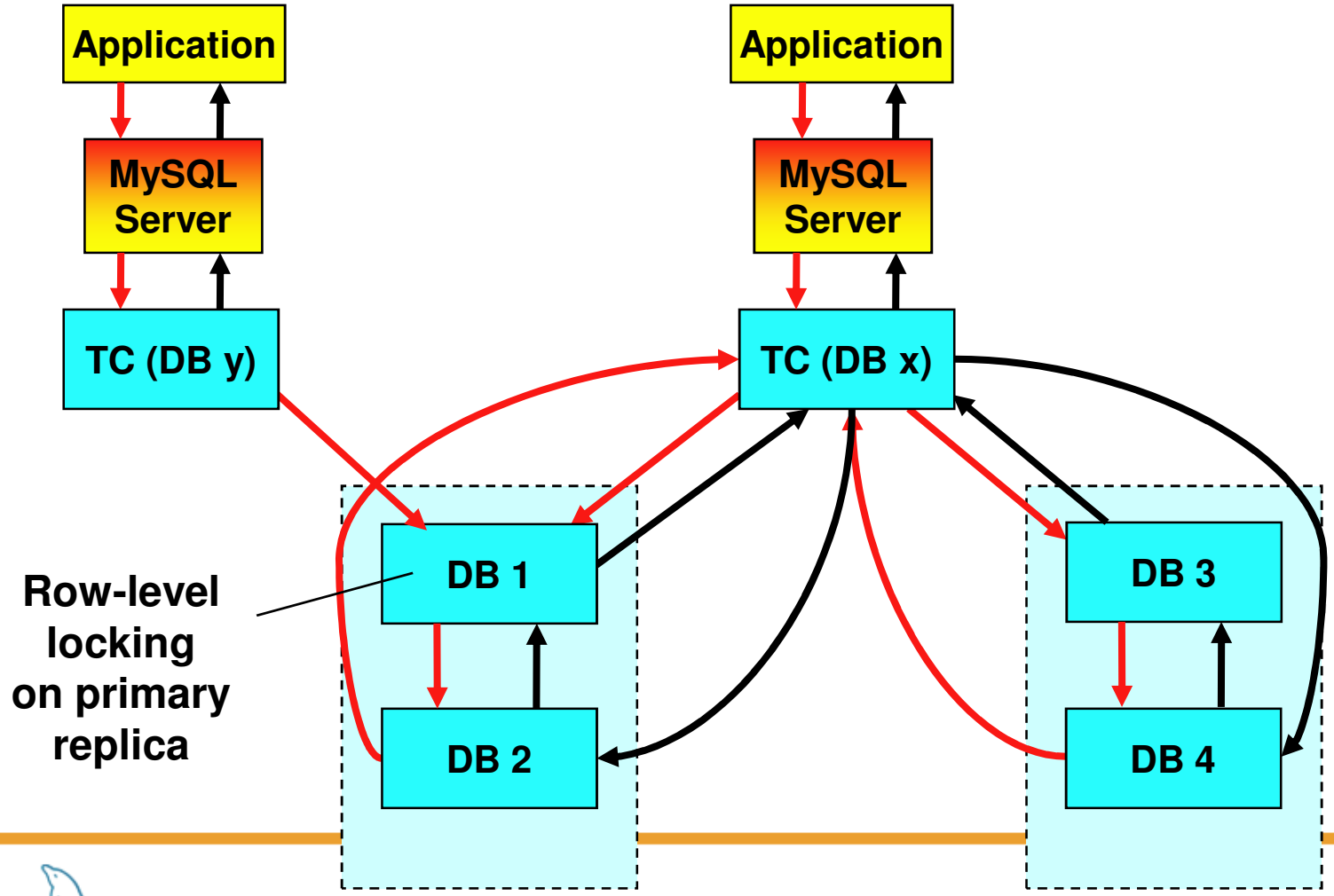
MySQL Cluster Replication

Where to get the log events?



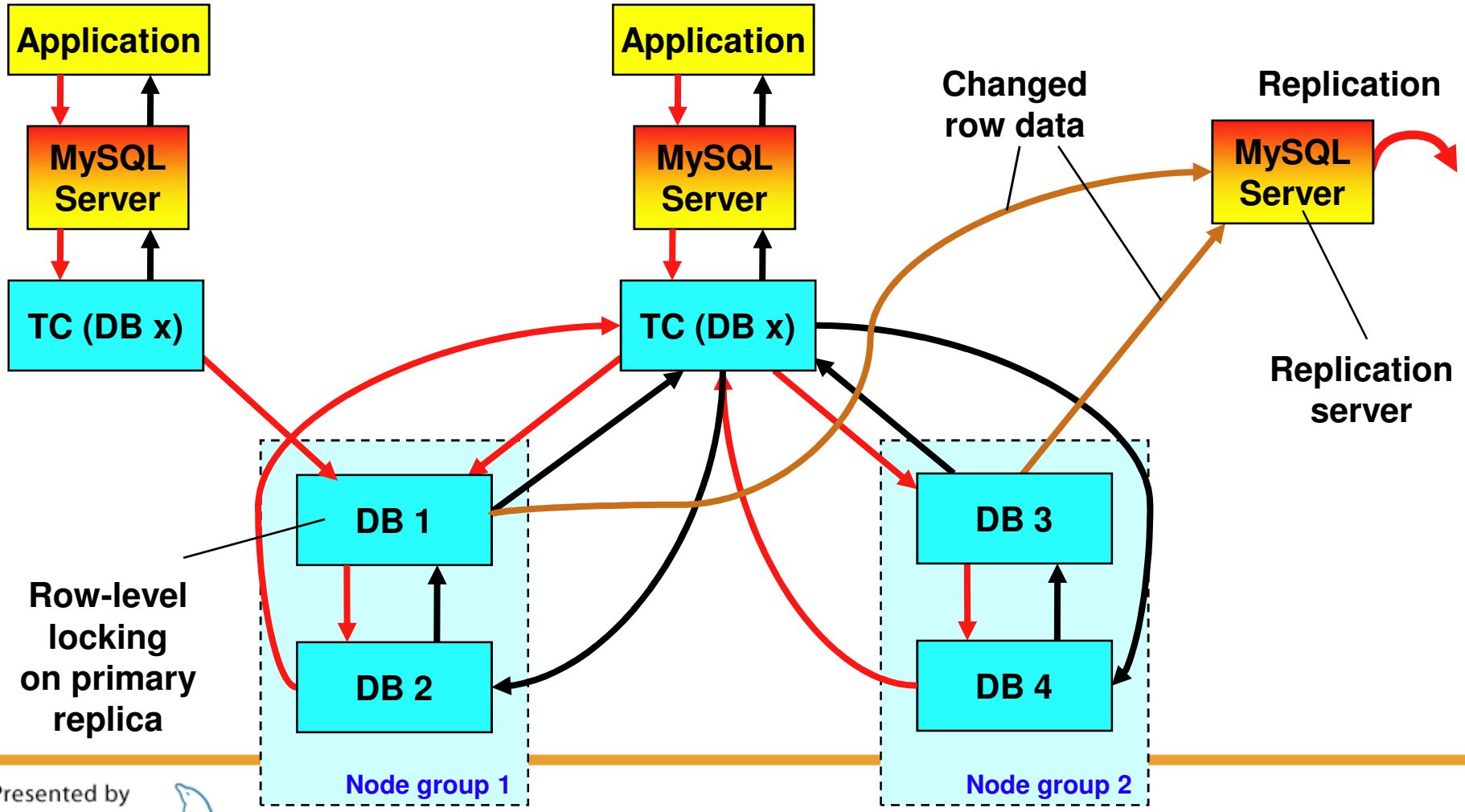
MySQL Cluster Replication

Concurrency control inside master cluster



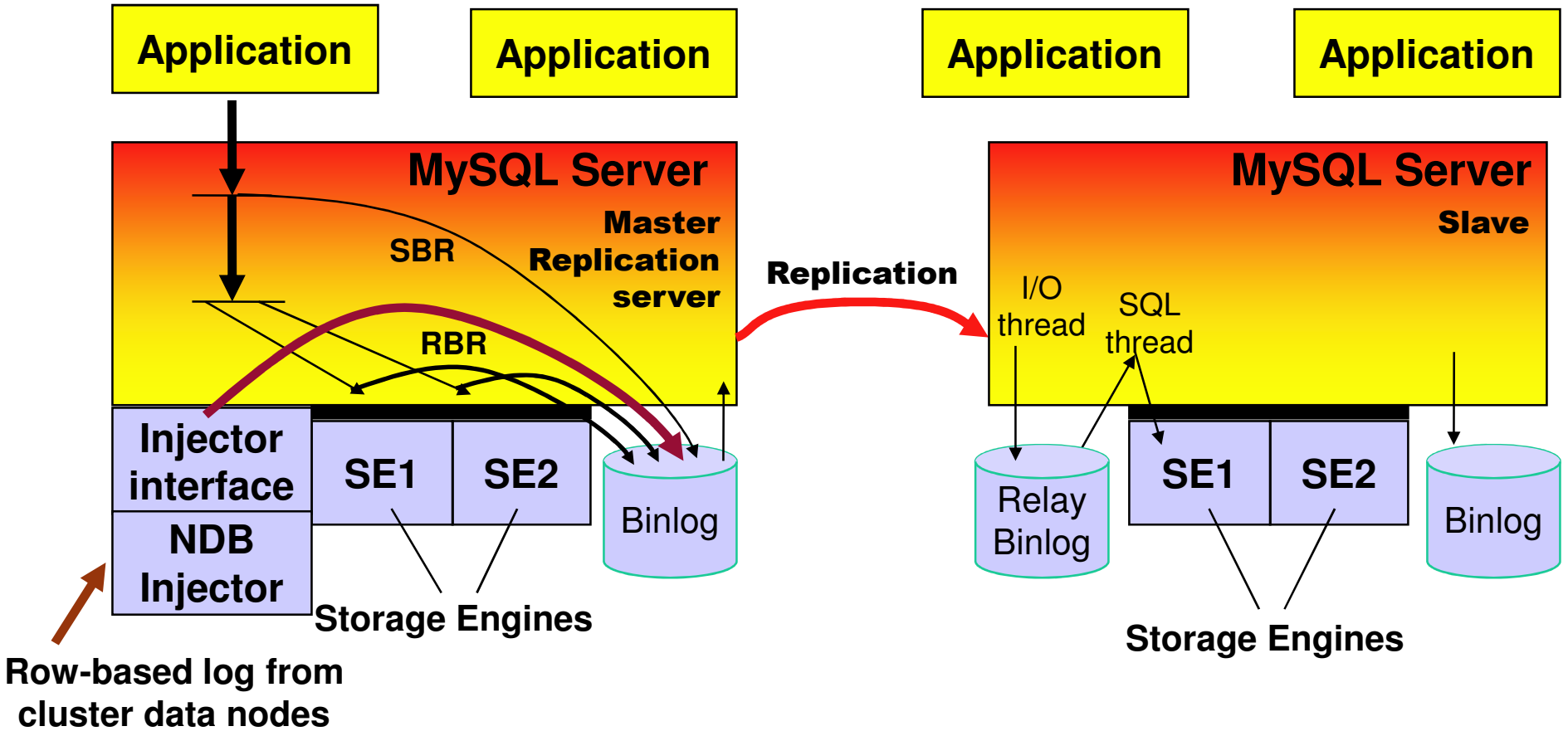
MySQL Cluster Replication

Log shipping inside master cluster



MySQL Replication Architecture

MySQL 5.1



MySQL Cluster Replication

Behaves like ordinary MySQL Replication