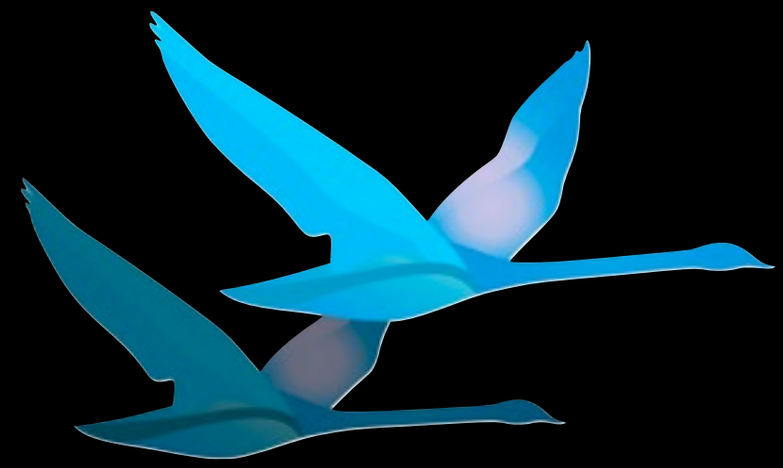
Advanced MySQL Replication Techniques



Giuseppe Maxia, QA Director, Continuent, Inc

Facts. And Demos. Possibly fun

about me - Giuseppe Maxia

- a.k.a. The Data Charmer
- QA Director at Continuent, Inc
- Long time hacking with MySQL features
- Formerly, community manager,db consultant, designer, coder.
- A passion for QA and open source
- Blogger
- http://datacharmer.blogspot.com

continuent
Open. Always Available.

Agenda

- Replication basics
- Replication blues
  - Master switch and failover
  - Slave lagging
  - Gotchas
- Replication power techniques and features
  - row-based, semi-synch, delayed
  - Circular, bi-directional
  - leveraging replication
- Tools and tips
- Replication outside the box
  - seamless failover, parallel, multi-master

AGENDA

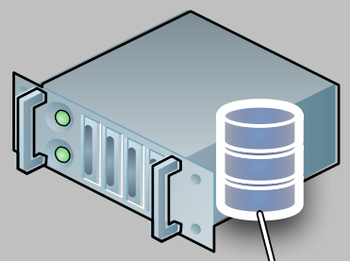
Replication basics

- Principles
- How to set up replication
- Monitoring replication

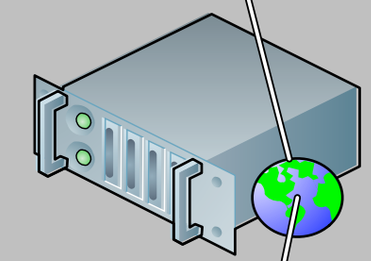
AGENDA

Principles

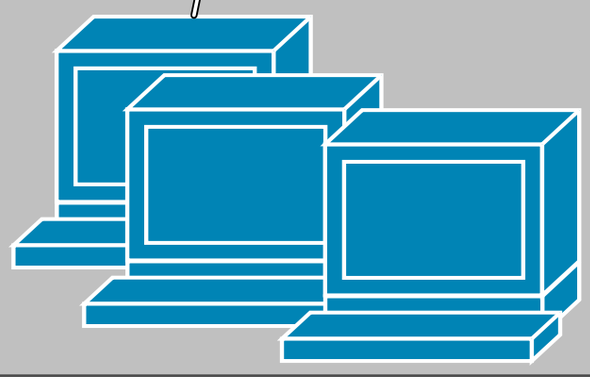
database server



r/w requests



web server



clients

a simple web application scheme

database server

scaling web requests

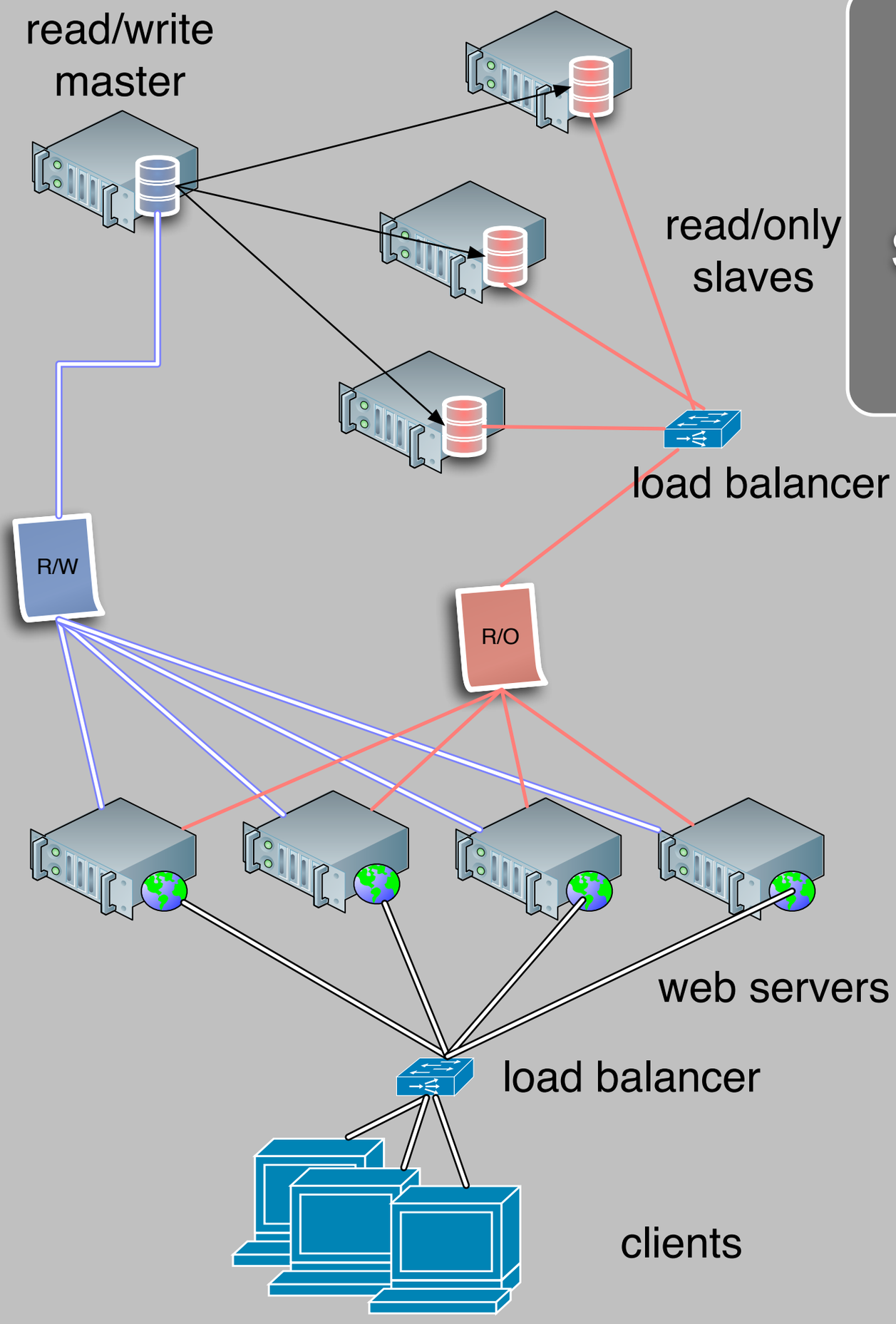
r/w requests

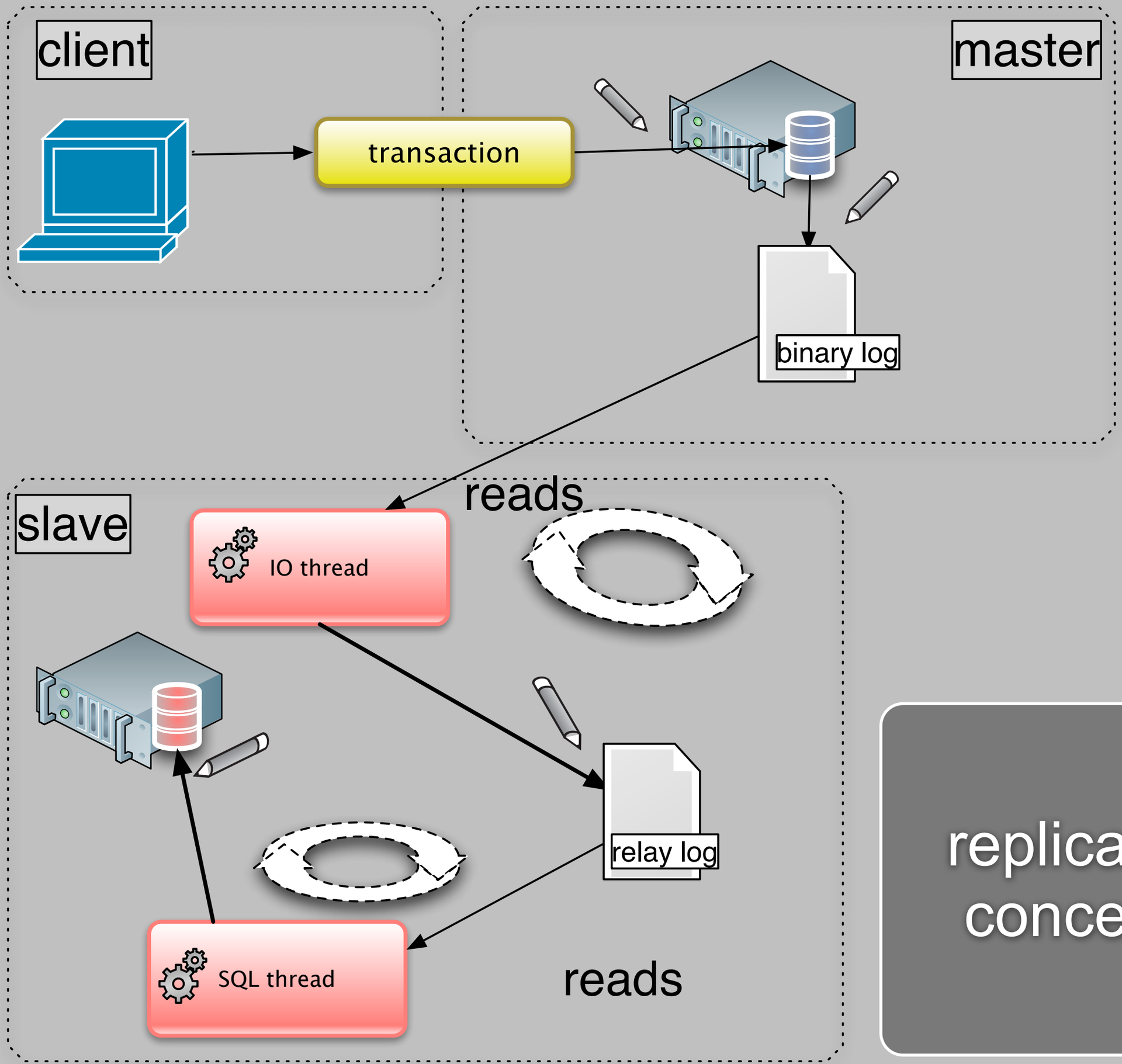
web servers

load balancer

clients

a web application scheme with replication



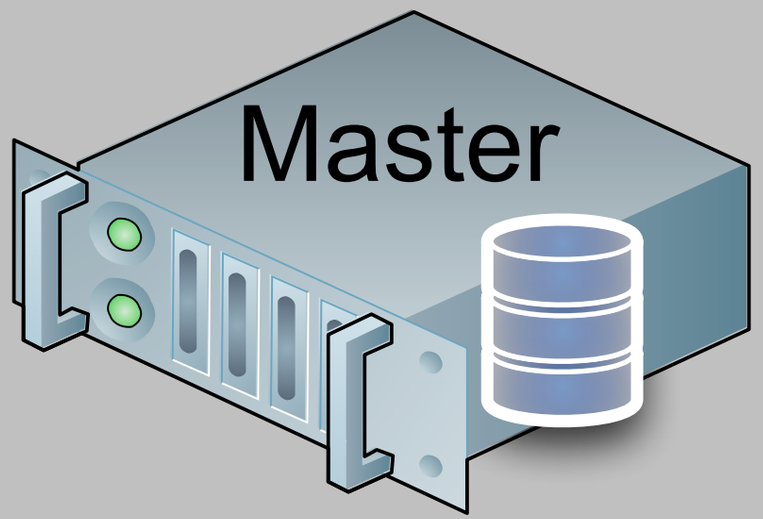


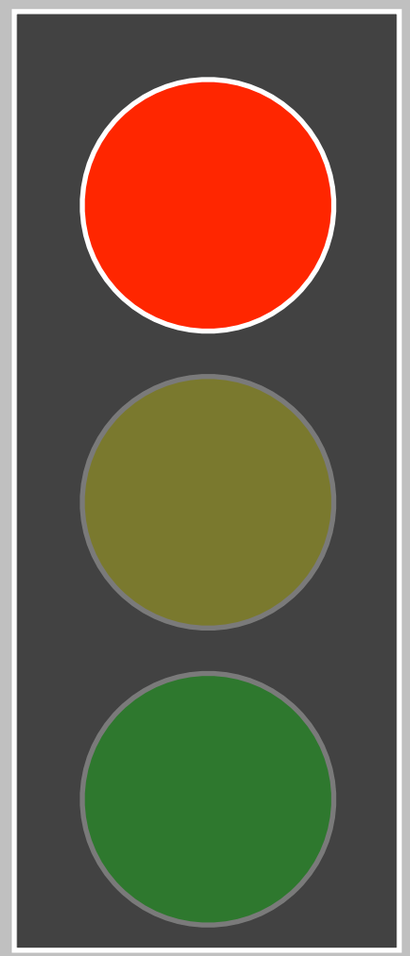
replication concepts

AGENDA

replication setup

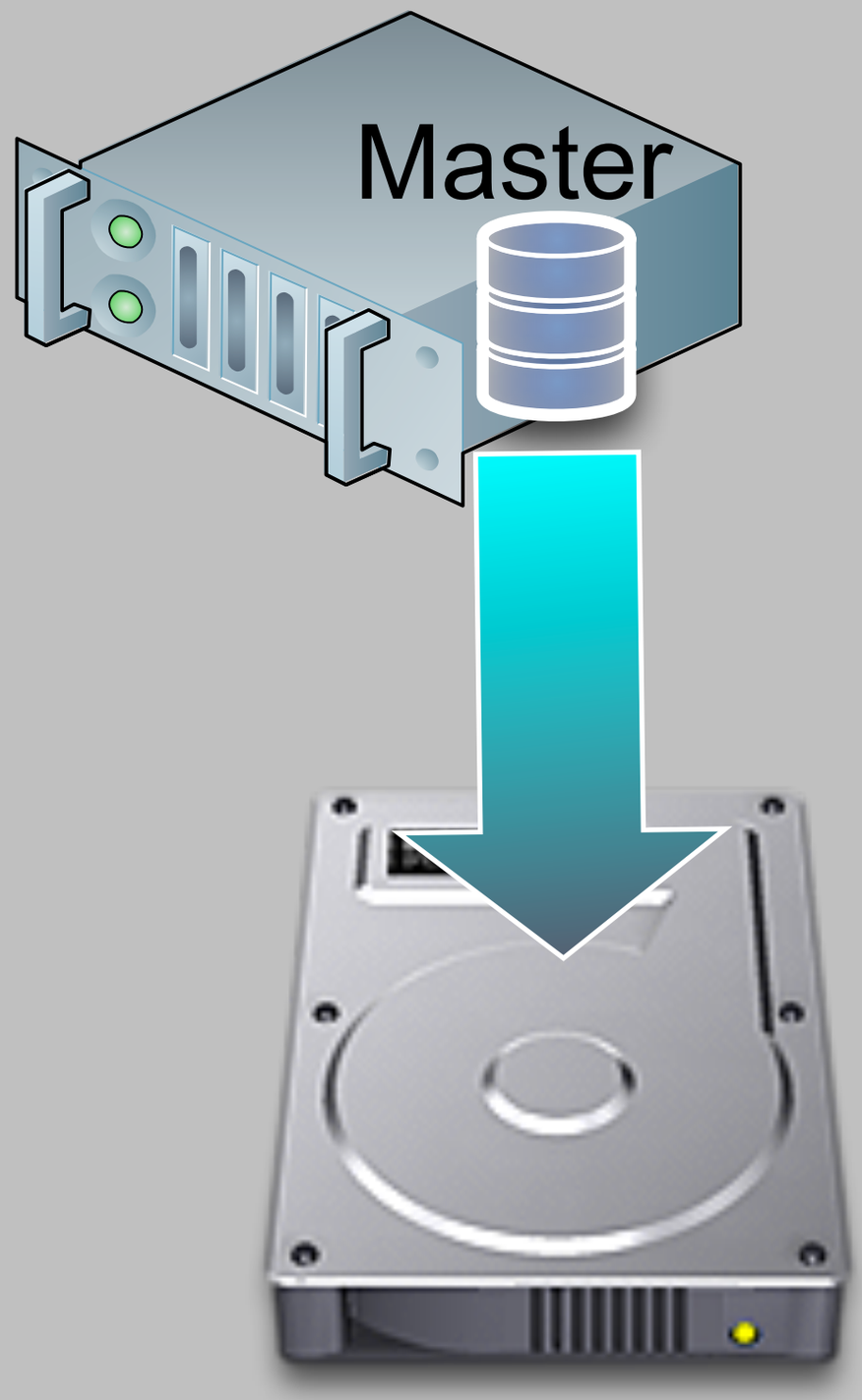
1 SHUT DOWN THE DATABASE SERVER





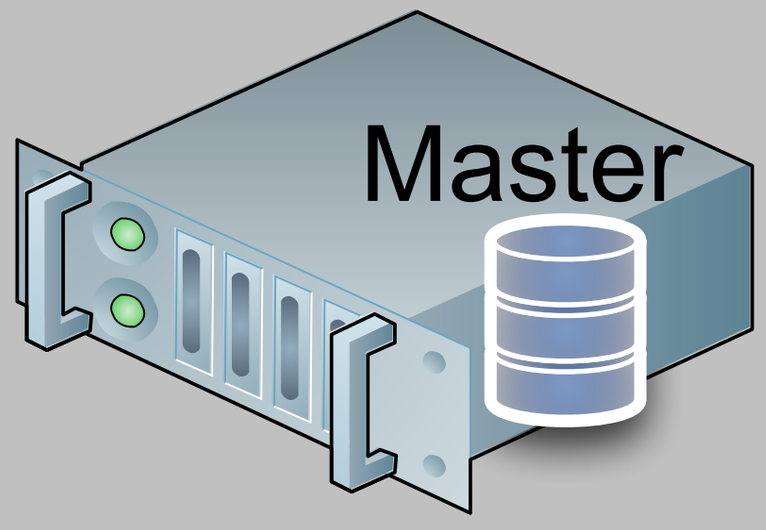
2

MAKE A BACKUP COPY



3

ENABLE THE MASTER

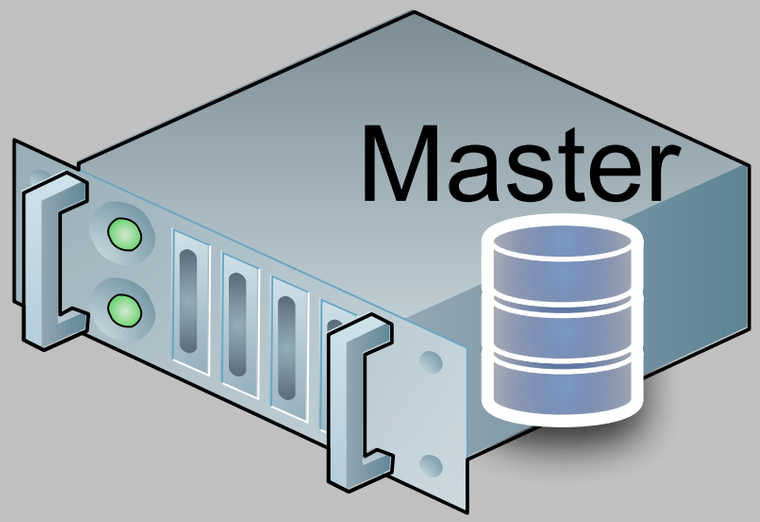


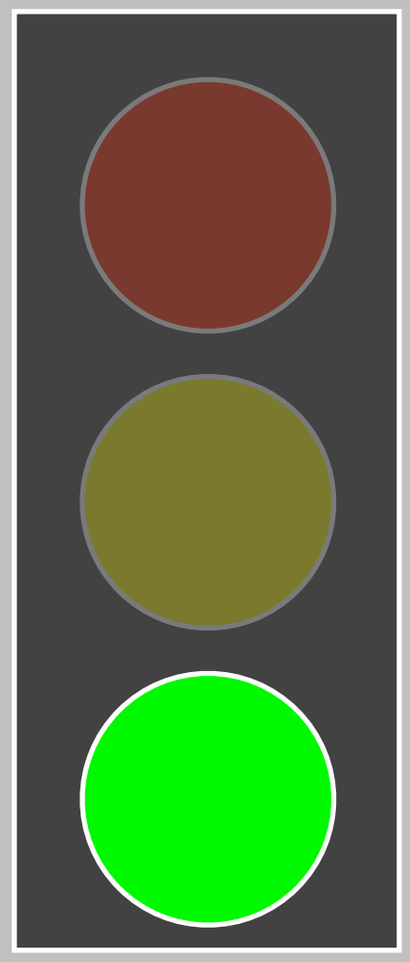
Configuration file

```
[mysqld]
log-bin=mysql-bin
server-id=1
```

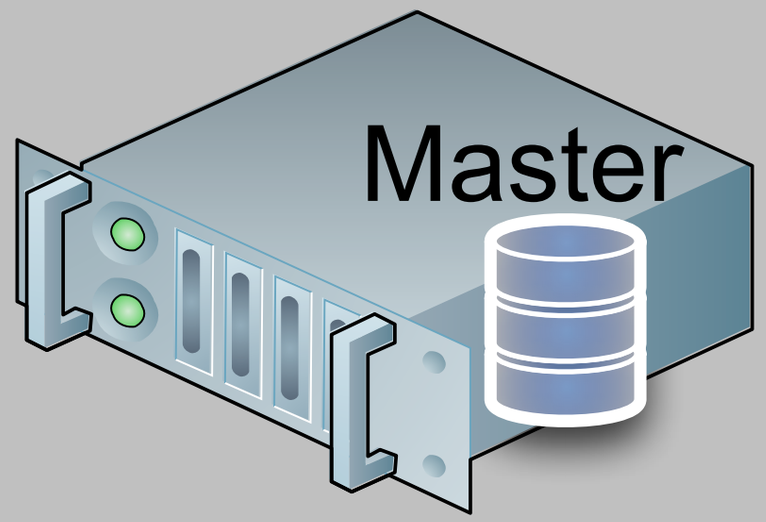
4

RESTART THE MASTER





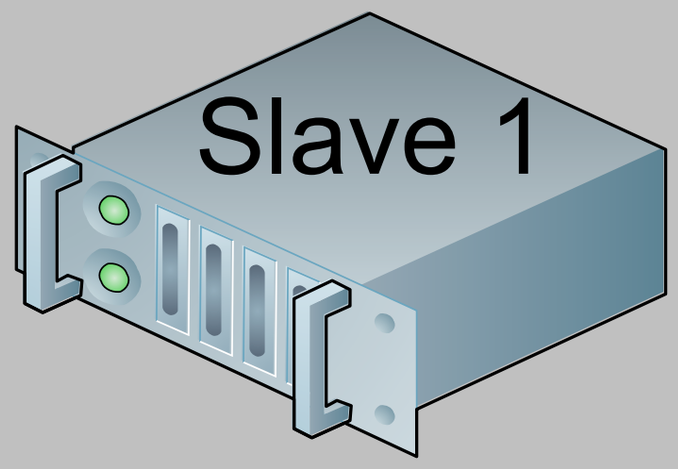
5 CREATE REPLICATION USER

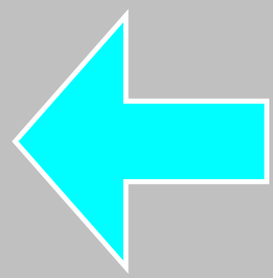


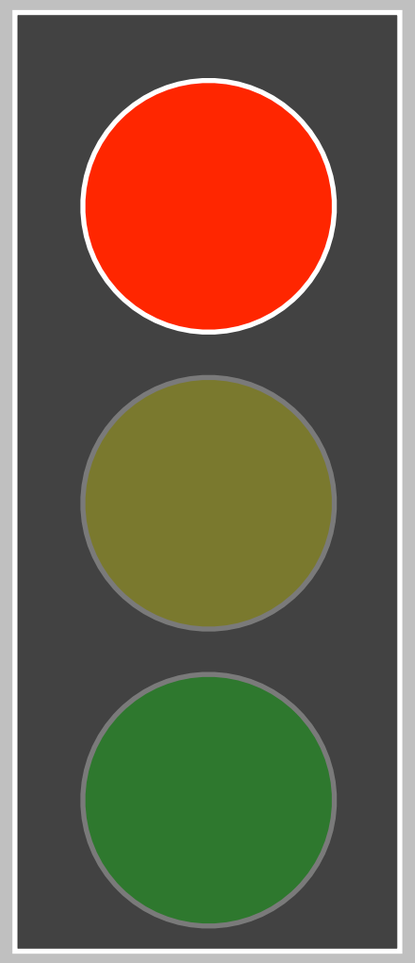
SQL command

```
GRANT REPLICATION SLAVE ON *.*
to 'slave_user@'10.10.100.%'
IDENTIFIED BY 'slave_pass';
```

6 INSTALL MySQL on the slave



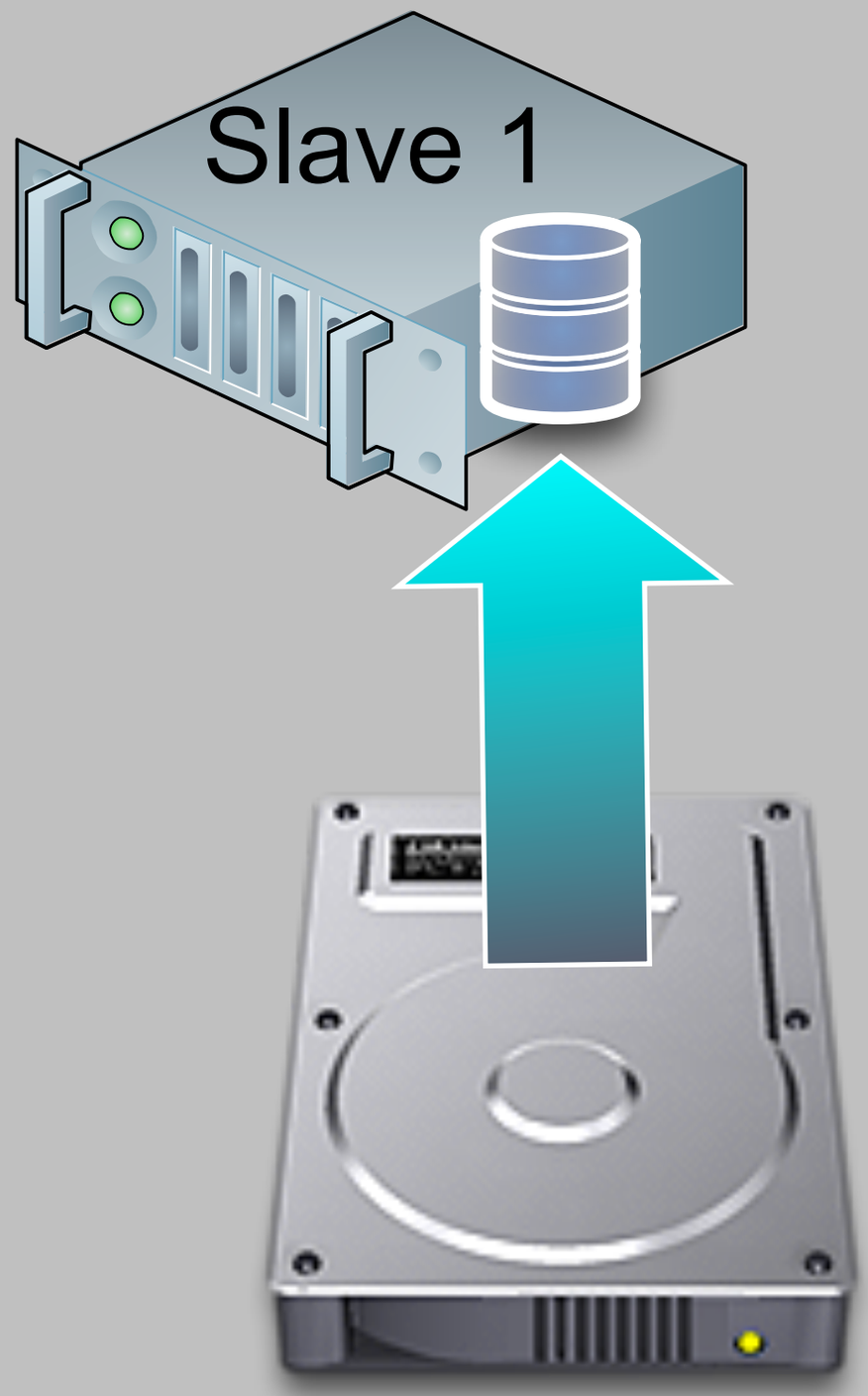




Make sure that:

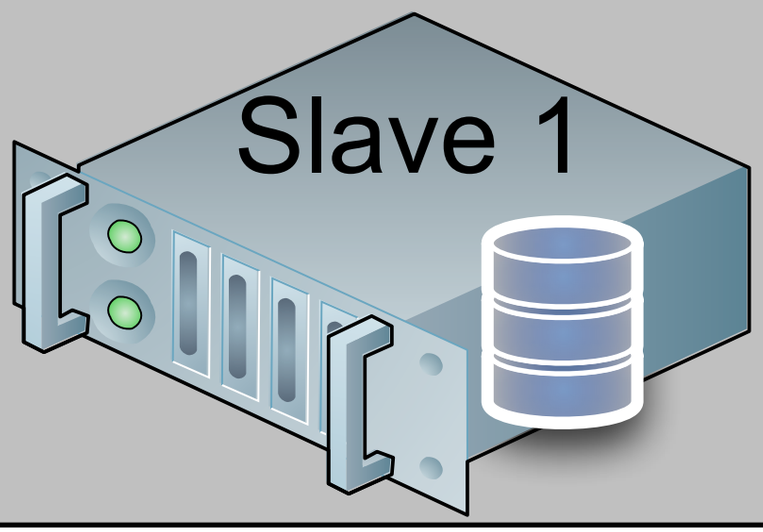
- You're using the same version of MySQL
- You have the same directory structure
- The server is not started yet

7 COPY THE MASTER DATA to the slave



8

ENABLE THE SLAVE

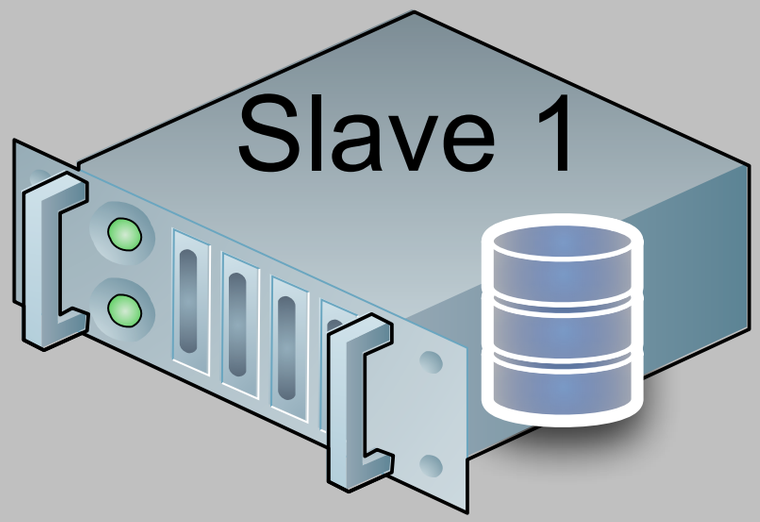


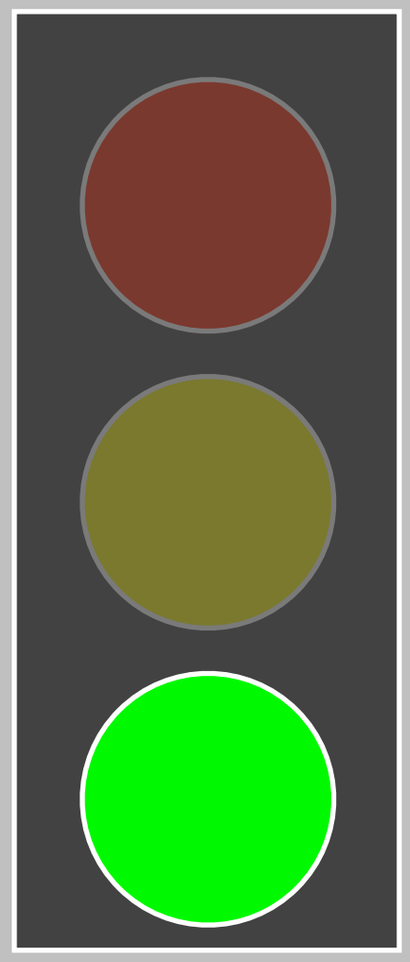
Configuration file

```
[mysqld]
server-id=2
relay-log=mysql-relay
read-only
# optional:
log-bin=mysql-bin
```

9

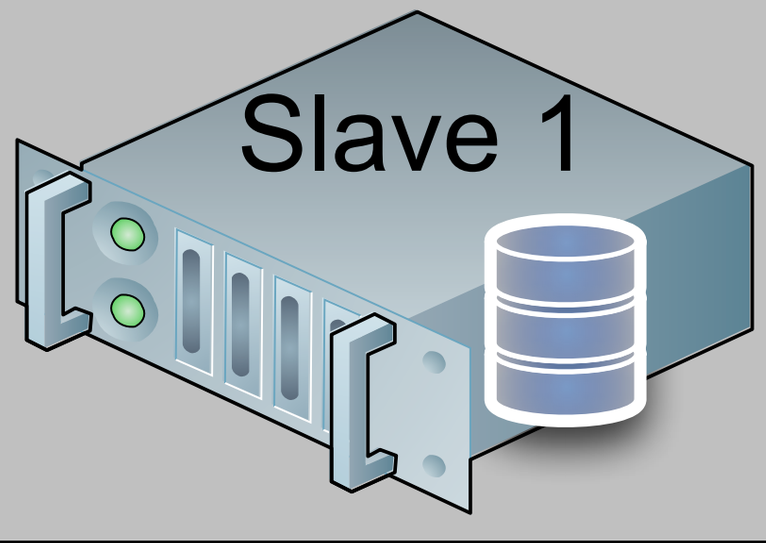
START THE SLAVE SERVER





10

INITIALIZE THE SLAVE

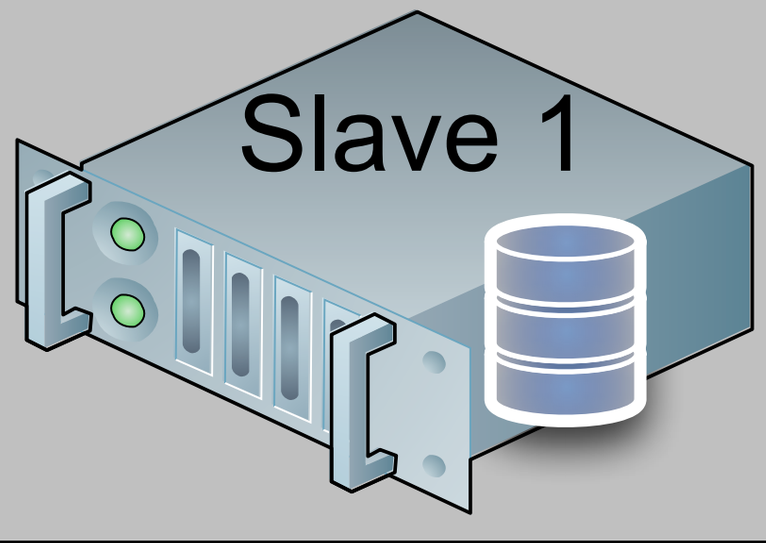


SQL command

```
CHANGE MASTER TO
MASTER_HOST=master_IP,
MASTER_PORT=3306,
MASTER_USER=slave_user,
MASTER_PASSWORD='slave_pwd';
```

11

START THE SLAVE SERVICE

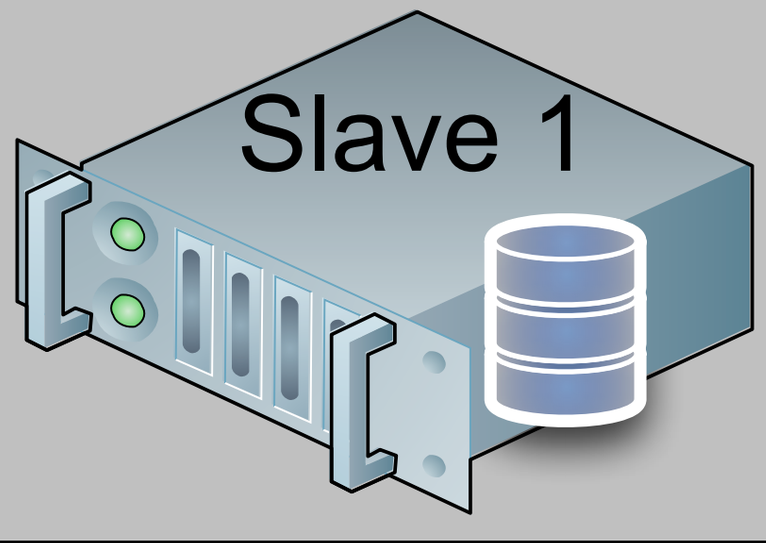


SQL command

```
START SLAVE;
```

12

CHECK THE SLAVE



SQL command

```
SHOW SLAVE STATUS \G

...
 Slave_IO_Running: Yes
Slave_SQL_Running: Yes
...
```

Troubleshooting

- **SHOW SLAVE STATUS says SLAVE_IO_RUNNING=No**
  - Make sure that the slave host can connect to the master
  - Make sure that master and slave have different Server-id
  - Check the error log of both master and slave

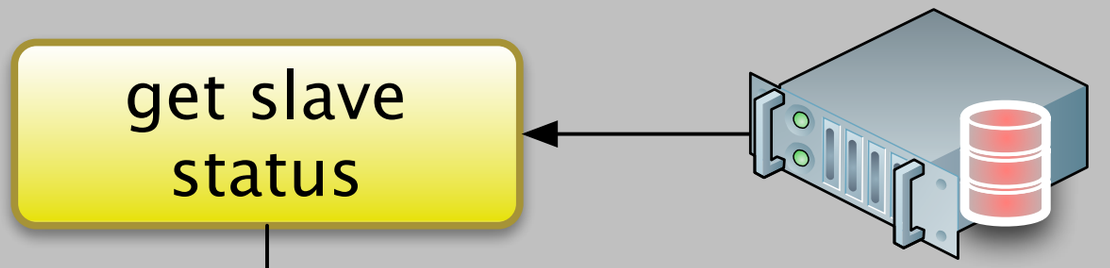
Testing the slave

- Create a table in the master.
- Make sure that the slave has replicated the table.
- Insert data in the master
- read that data in the slave

AGENDA

Monitoring

Sample monitoring



monitoring replication

```
master> SHOW MASTER STATUS

slave> SHOW SLAVE STATUS
```

FULL SCRIPTS:

http://datacharmer.blogspot.com/2011/04/refactored-again-poor-mans-mysql.html

http://forge.mysql.com/tools/tool.php?id=6

show master status

```
            File: mysql-bin.000002
        Position: 78045744
    Binlog_Do_DB:
Binlog_Ignore_DB:
```

show slave status

```
Slave_IO_State: Waiting for master to send event
   Master_Host: 127.0.0.1
   Master_User: rsandbox
   Master_Port: 27371
  Connect_Retry: 60
Master_Log_File: mysql-bin.000002
Read_Master_Log_Pos: 78045744
Relay_Log_File: mysql_sandbox27372-relay-bin.000055
Relay_Log_Pos: 78045889
Relay_Master_Log_File: mysql-bin.000002
   Slave_IO_Running: Yes
  Slave_SQL_Running: Yes
```

show slave status

```
...
               Replicate_Do_DB:
           Replicate_Ignore_DB:
            Replicate_Do_Table:
        Replicate_Ignore_Table:
       Replicate_Wild_Do_Table:
   Replicate_Wild_Ignore_Table:
 ...
  . . .
```

show slave status

```
...
                Last_Errno: 0
                Last_Error:
              Skip_Counter: 0
       Exec_Master_Log_Pos: 78045744
           Relay_Log_Space: 78046100
...
      Seconds_Behind_Master: 0
...
               Last_IO_Errno: 0
               Last_IO_Error:
              Last_SQL_Errno: 0
              Last_SQL_Error:
```

AGENDA

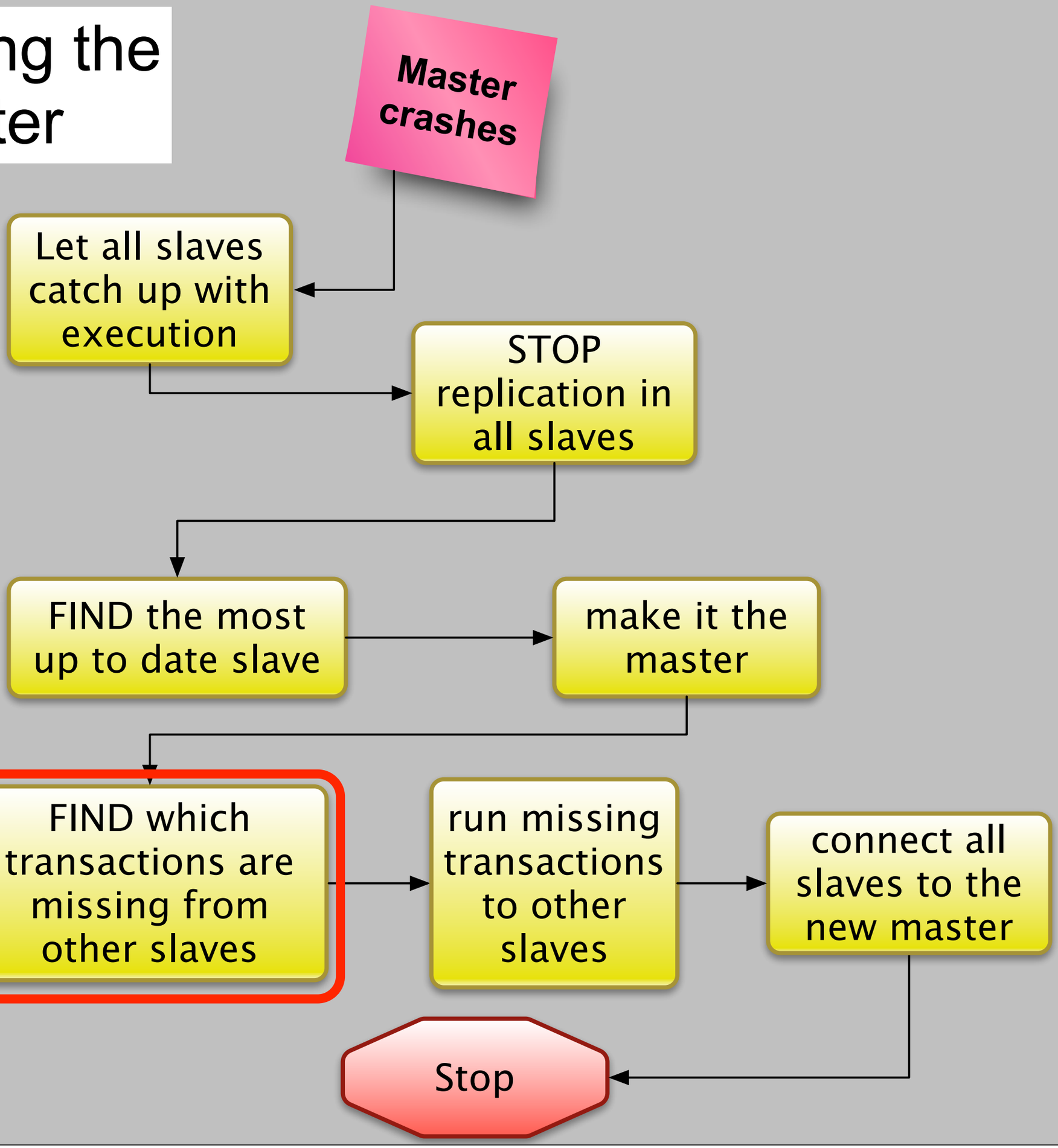
Replication blues

- Master switch and failover
- Slave lagging
- Gotchas

AGENDA

Master switch and failover

Replacing the master



Planned master switch

- Stop accepting writes
- Wait until all slaves have caught up
- Stop replication in all slaves
- Promote a slave to master
- Point all slaves to the new master

Changing a failed master

- Pre-requisite:
- log_bin and log_slave_updates must be enabled in all the slaves
- If not, there is more manual labor

Changing a failed master (1)

- Wait until all slaves have caught up
- Identify the most advanced slave
- Make that slave the new master
- ... so far, so good

Changing a failed master (2)

- For each remaining slave:
- Find the missing statements

- Find the **LAST** statement replicated by the slave
- Find the same statement in the new master binlog (*)
- get the position of the **NEXT** statement

(*) if log_slave_updates was not enabled, you need to convert the relay log statements to SQL and do the next step manually

Changing a failed master (3)

- For each remaining slave:
- Apply the missing statements
  - CHANGE MASTER TO master_host=”new_master_hostname”, master_port=new_master_port, master_log_file=”mysql-bin.xxxxxx”, master_log_pos=YYYY

Reasons for complexity

- No global transaction ID

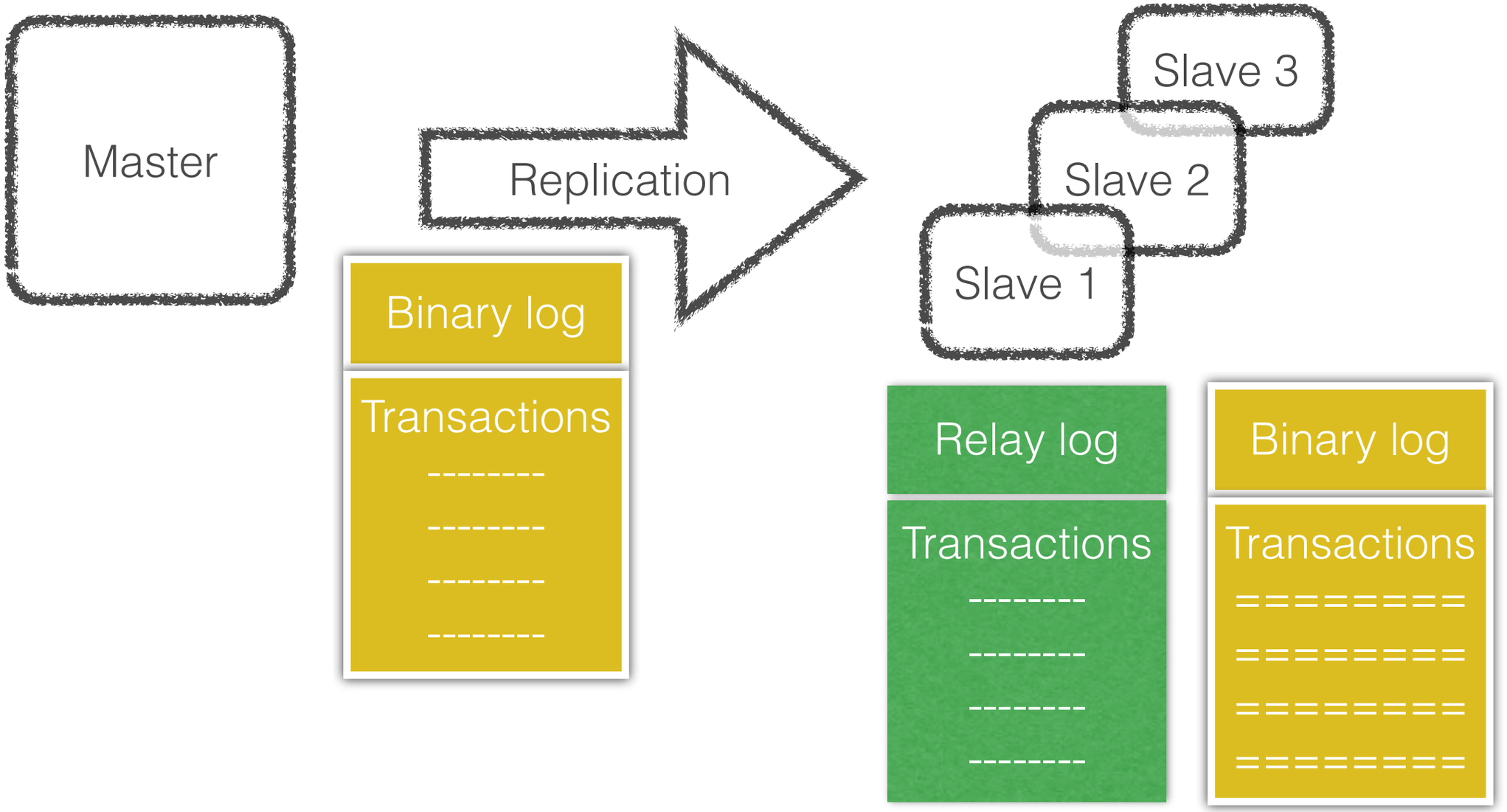
What is a global transaction ID

- A unique identifier of a transaction
- Unique for the whole cluster, not for each node
- Generated by the ultimate source (the master)
- Does not change when the transaction goes through an intermediate master

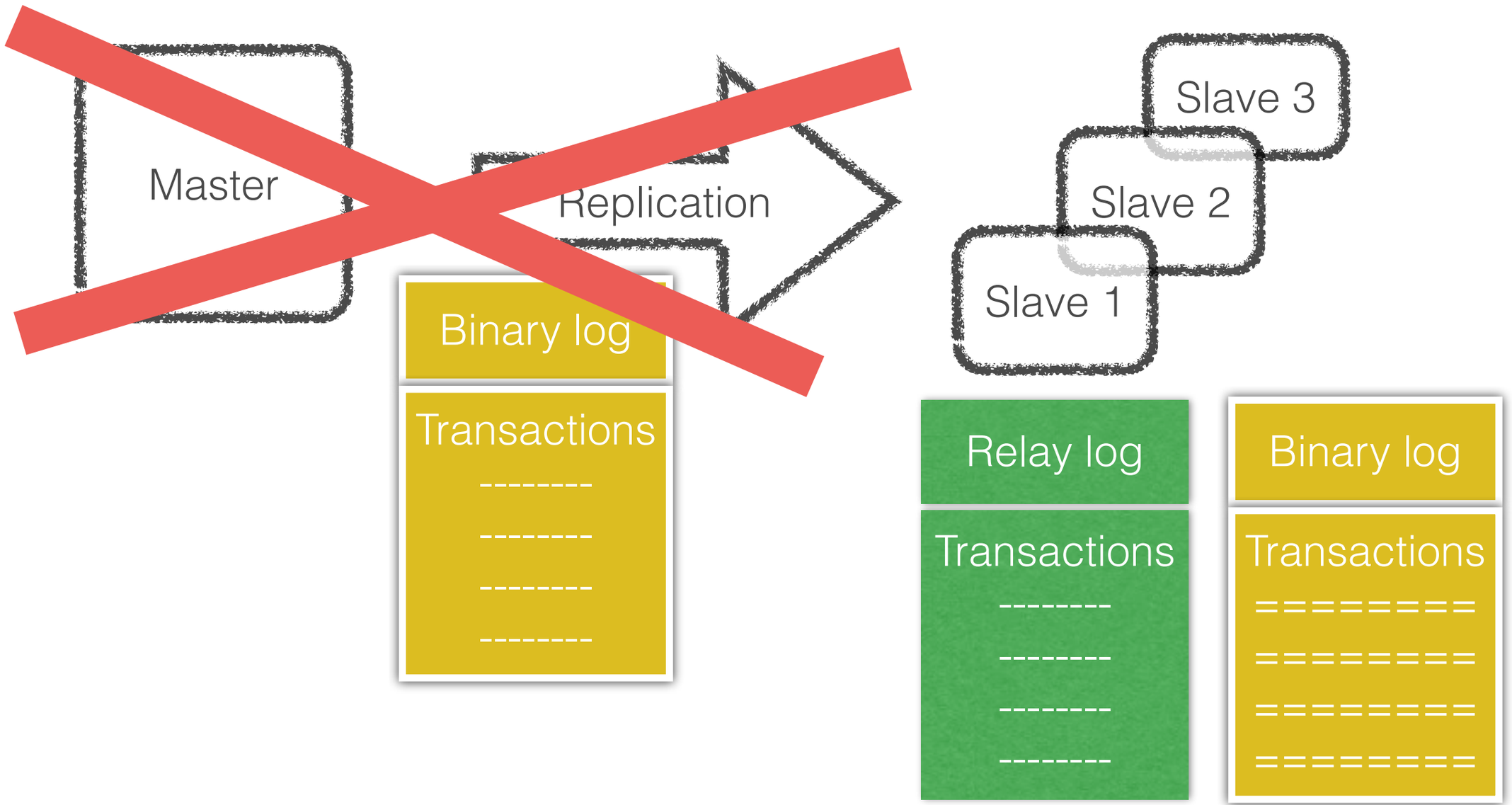
Why should you care

- Failure recovery
- MySQL DOES NOT have it

Defining the problem



The master crashes

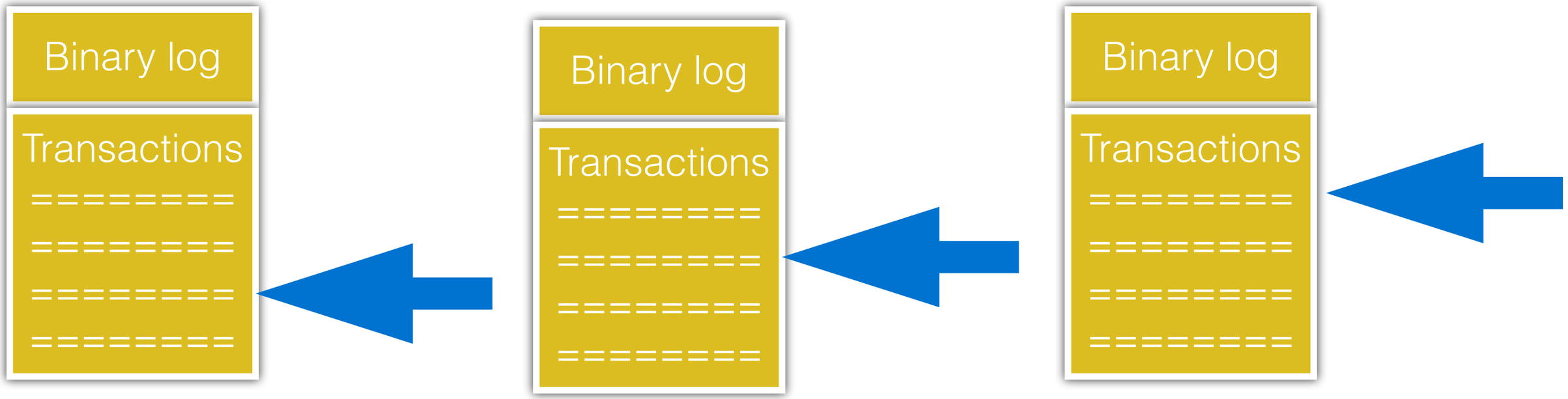


Where do the slaves stand?

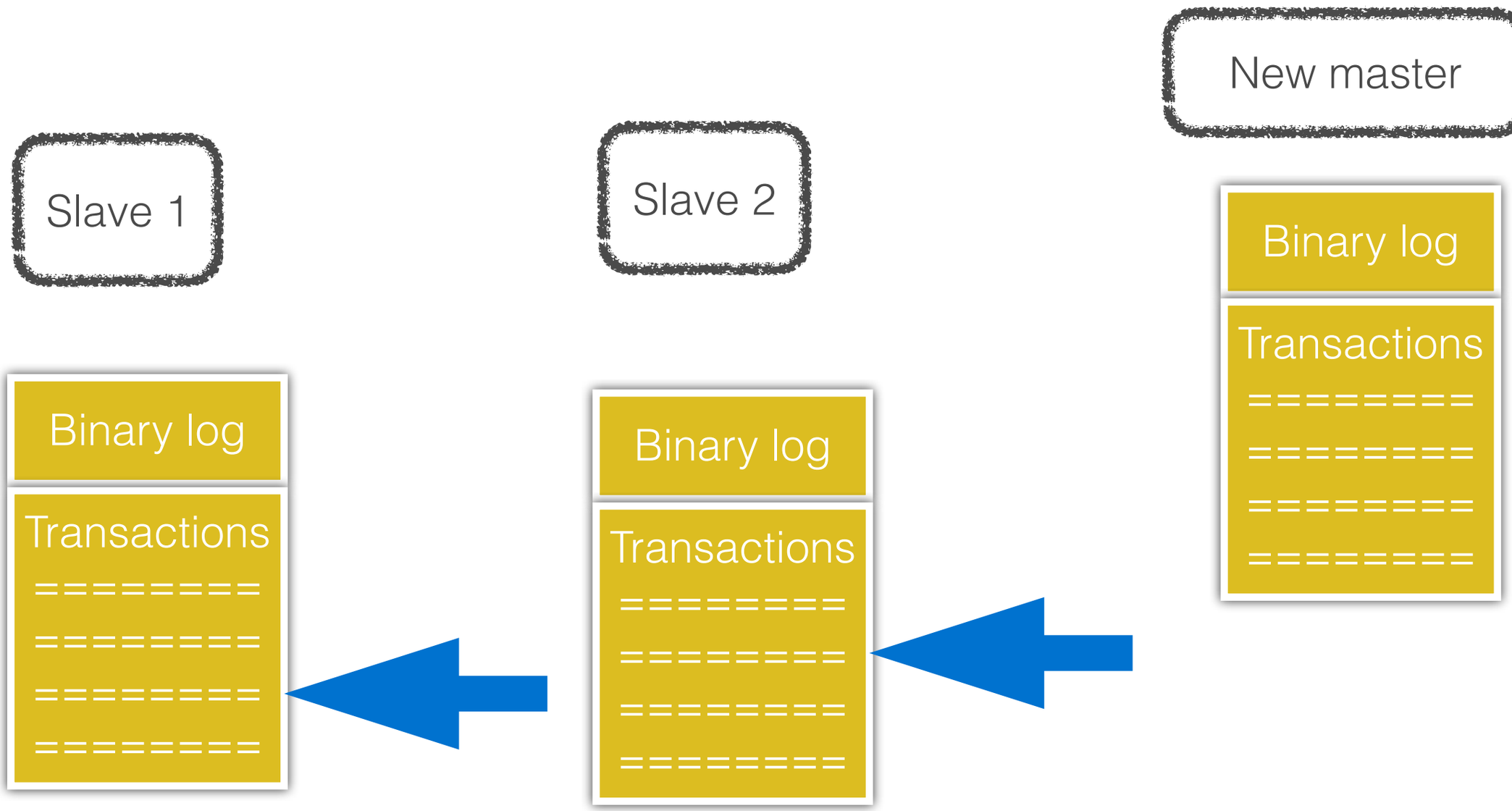
Slave 1

Slave 2

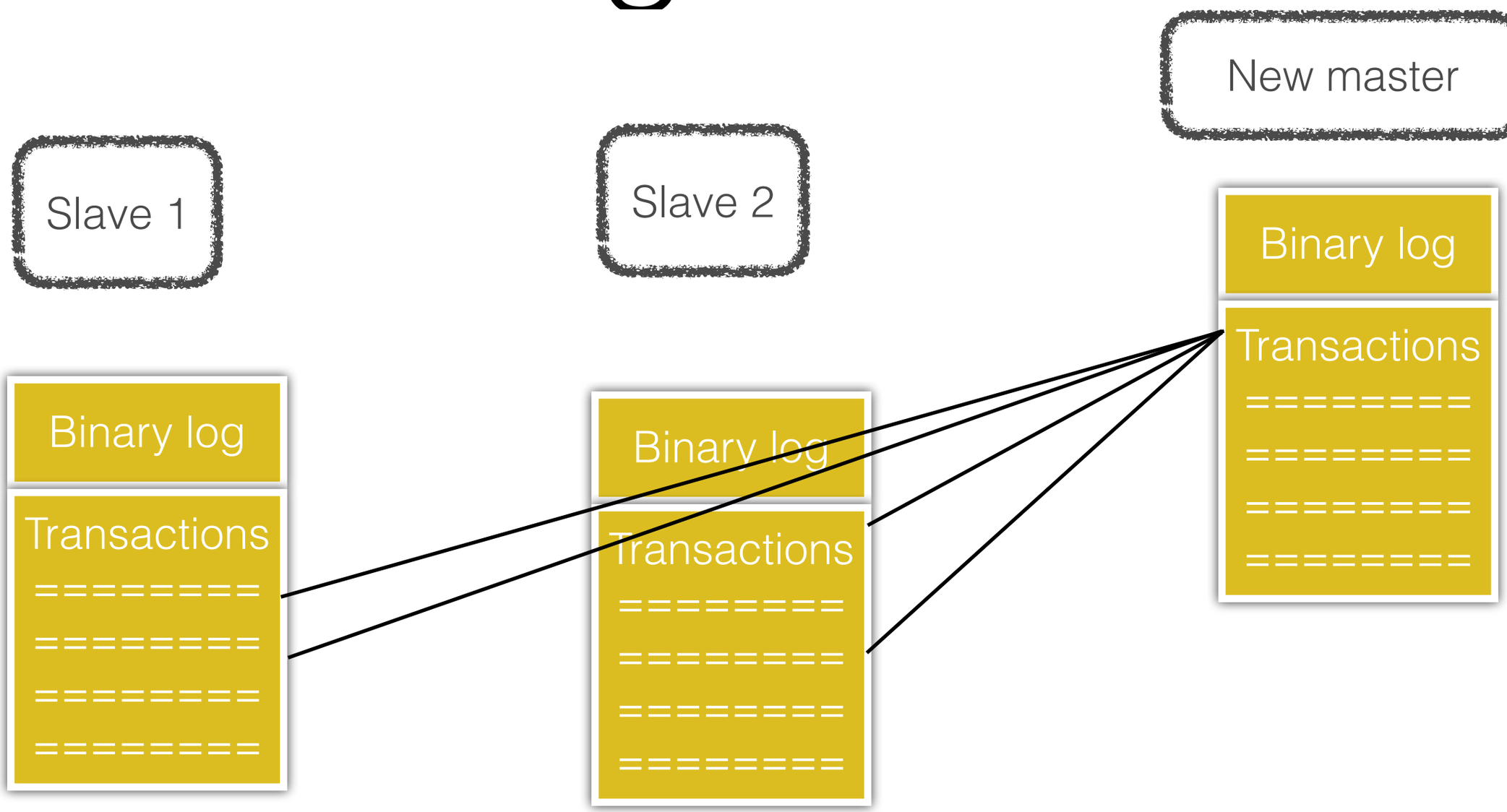
Slave 3



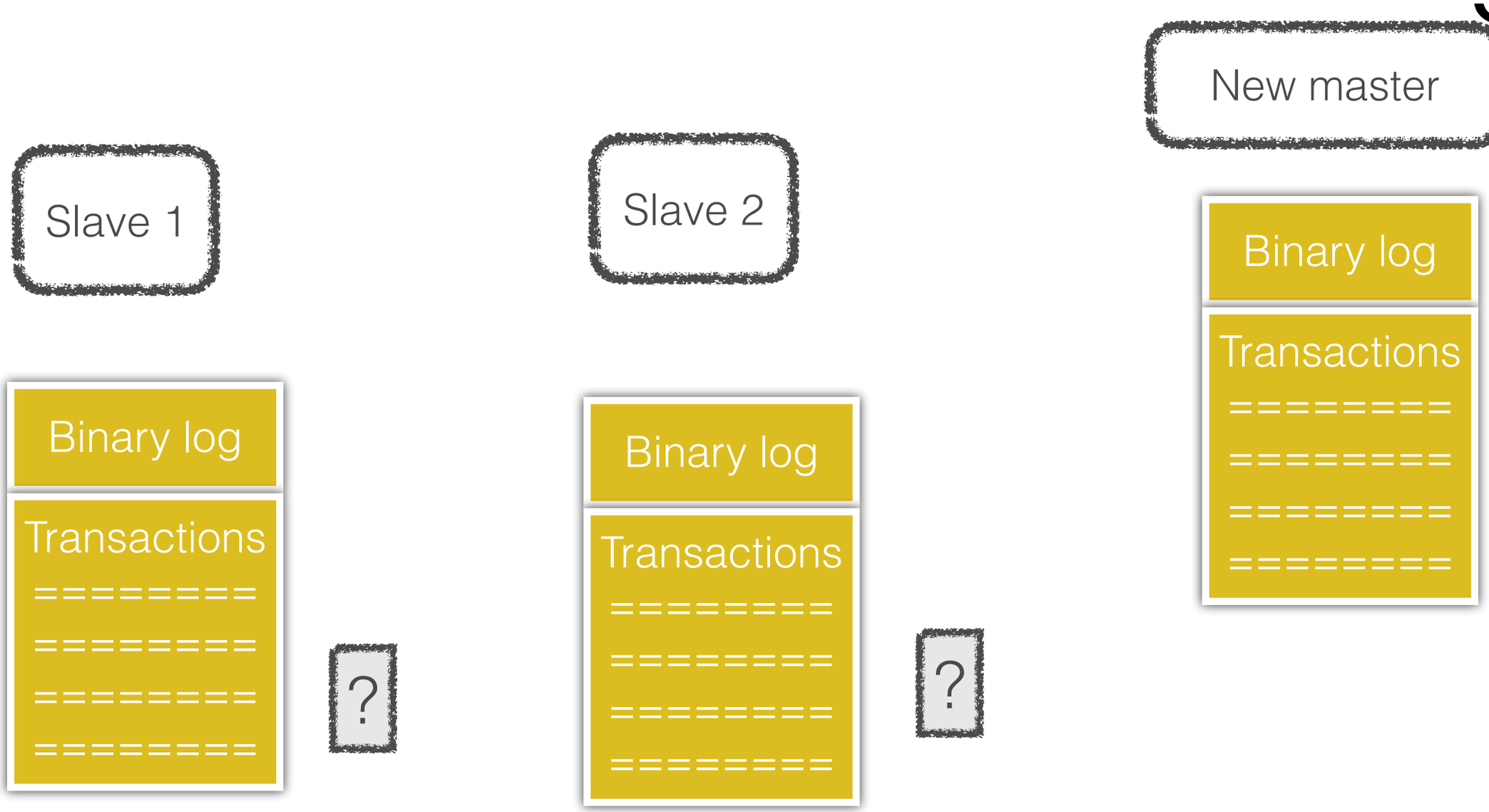
Slave 3 becomes master



Need to synch the missing transactions



Transaction position in the new master binlog



An existing solution

- Google has made a patch to implement global transaction IDs

- **HOWEVER**
- It only works with 5.0
- It doesn't work with row based replication

Tools for Building Robust Data Centers

MySQL High Availability

Charles Bell, Mats Kindahl
& Lars Thalmann
Foreword by Mark Callaghan

O'REILLY®

Hack ahead!

```
CREATE TABLE Global_Trans_ID (
    number INT UNSIGNED AUTO_INCREMENT PRIMARY KEY
) ENGINE = MyISAM;


CREATE TABLE Last_Exec_Trans (
    server_id INT UNSIGNED,
    trans_id INT UNSIGNED
) ENGINE = InnoDB;
```

```
CREATE PROCEDURE commit_trans ()

BEGIN

  DECLARE trans_id, server_id INT UNSIGNED;

  SET SQL_LOG_BIN = 0;

  INSERT INTO global_trans_id() values ();

  SELECT LAST_INSERT_ID() INTO trans_id,

         @@server_id INTO server_id;

   DELETE FROM global_trans_id where number < trans_Id;

  SET SQL_LOG_BIN = 1;

  INSERT INTO last_exec_trans(server_id, trans_id)

       VALUES (server_id, trans_id);

COMMIT;

END
```

Hacking the hack

Simplifying the scheme

- **No** MyISAM table and LAST_INSERT_ID()
- Use UUID_SHORT() instead
- Replace the InnoDB table with a **BlackHole** one

Advantages

- **10** times faster
- No additional tables and storage required

Disadvantages

- The master SERVER ID must be < 256

Another solution

- We'll see that in the second part of the talk

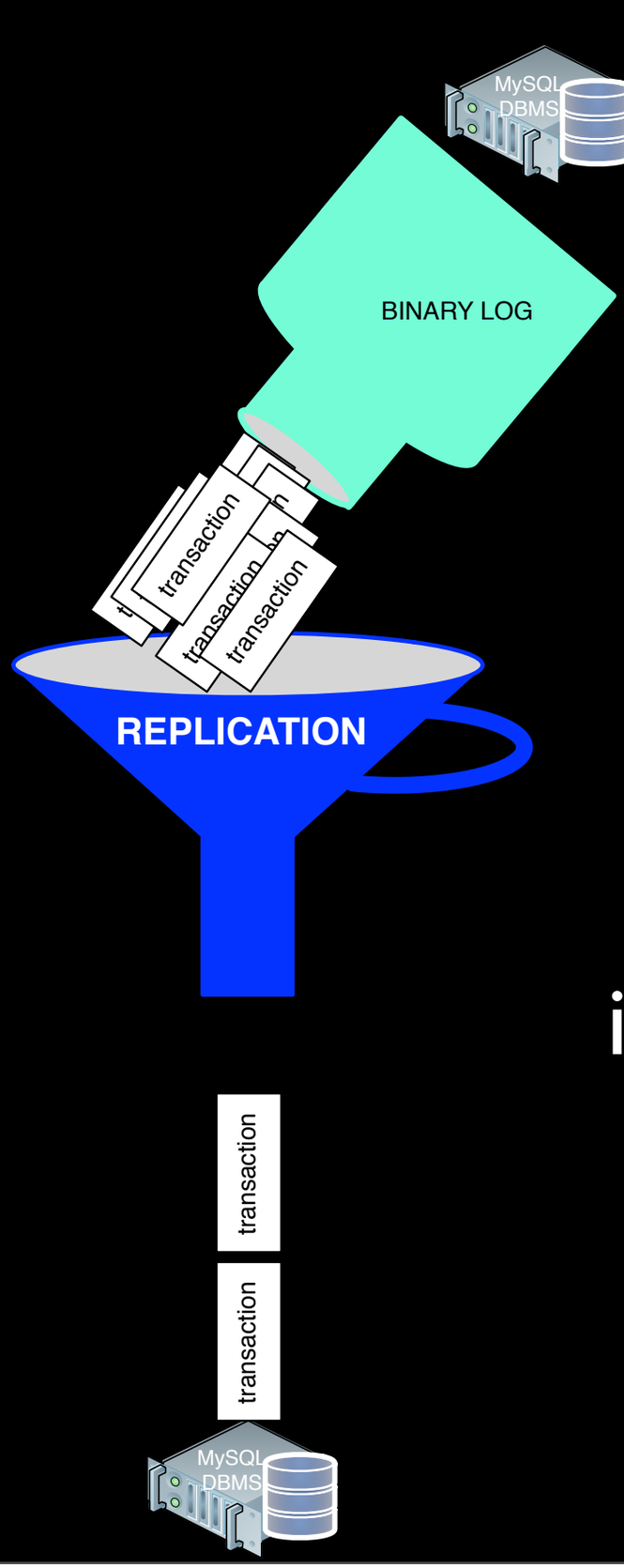
AGENDA

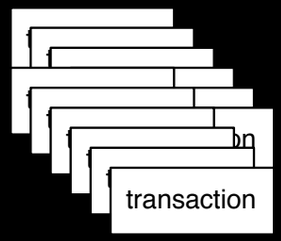
Slave lagging

Slave lagging

- Common causes:
  - Excess of traffic
  - Slave restart
  - Long DDL operations

Slave lagging





MySQL replication
is SINGLE THREAD

Slave lagging remedies

- row-based replication (helps in some cases)
- Expensive query split
- Double apply (dangerous)
- Slave query prefetch (requires external app)
- Parallel replication (requires external app)

statement-based replication

```
master> set global binlog_format='statement';
master> insert into test.t1 select count(*) from
   information_schema.columns;
master> show global status like 'opened_tables';
+----------------+-------+
| Variable_name | Value |
+----------------+-------+
| Opened_tables | 41    |
+----------------+-------+
slave> show global status like 'opened_tables';
+----------------+-------+
| Variable_name | Value |
+----------------+-------+
| Opened_tables | 41    |
+----------------+-------+
```

row-based replication

```
master> set global binlog_format='row';
master> insert into test.t1 select count(*) from
  information_schema.columns;
master> show global status like 'opened_tables';
+---------------+-------+
| Variable_name | Value |
+---------------+-------+
| Opened_tables | 41    |
+---------------+-------+
slave> show global status like 'opened_tables';
+---------------+-------+
| Variable_name | Value |
+---------------+-------+
| Opened_tables | 17    |
+---------------+-------+
```

Expensive query split

```
# instead of this:
insert into t1 select benchmark(10000000, sha('abc'));
Query OK, 1 row affected (4.19 sec)
Records: 1  Duplicates: 0  Warnings: 0


# you may do this
set @result=(select benchmark(10000000, sha('abc')));
Query OK, 0 rows affected (4.19 sec)

insert into t1 values (@result);
Query OK, 1 row affected (0.00 sec)
```

Expensive query split

```
# at 471
#110410  8:21:21 server id 1  end_log_pos 518
User_var
SET @`result`:=0/*!*/;
# at 518
#110410  8:21:21 server id 1  end_log_pos 612   Query
thread_id=3     exec_time=0     error_code=0
SET TIMESTAMP=1302448881/*!*/;
insert into t1 values (@result)
```

Slave query prefetch

- What is it?
  - a technique that fetches queries from the relay logs, and runs them as SELECT statements, to warm up the indexes
- How do you do it?
  - mk-slave-prefetch (http://www.maatkit.org)
  - Slave readahead http://sourceforge.net/projects/slavereadahead/

AGENDA

Gotchas

Gotchas

- When you thought that you had figured out everything ...

Gotchas: LOAD DATA

- LOAD DATA INFILE is replicated as a temporary file
- The slave needs three times the size of the data:
  - the relay log
  - the temporary file
  - the data itself

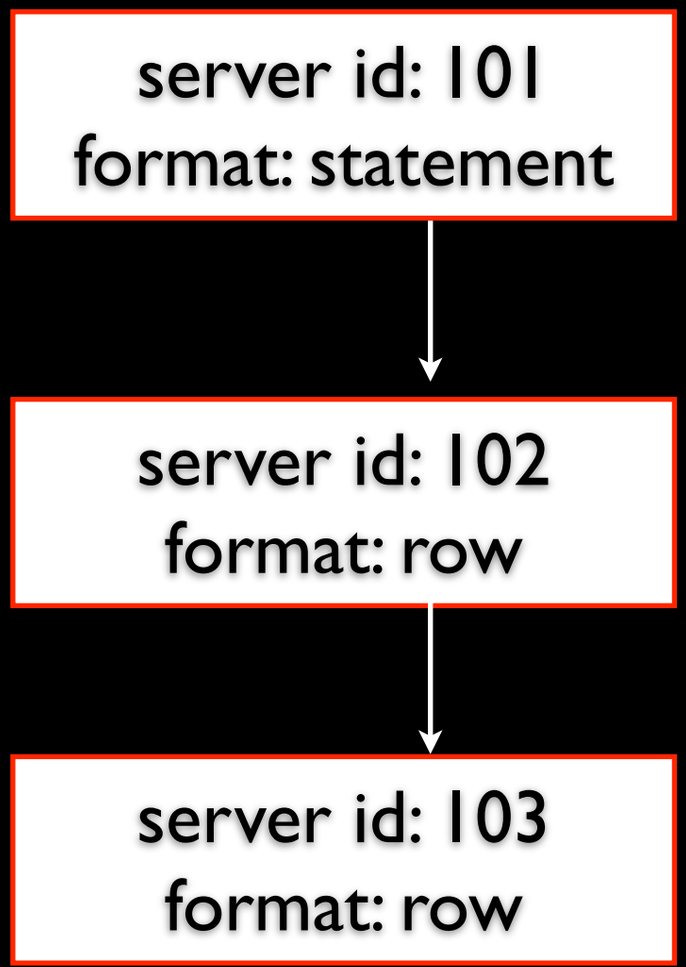
Gotchas: default engine

- If you define storage_engine in the master, it does not get replicated

Gotchas: binlog format

- In circular replication, and relay slaves
- Changes to binlog_format are not propagated

Gotchas: binlog format




```
insert into t1
values (@@server_id)
```

101

102

102

Gotchas: set global not really global

- set global binlog_format=row
- Works for all new connections after the change
- It does not work for the event scheduler!

Gotchas: triggers and row replication

- In statement based replication, triggers are fired both on master and slave
- in row-based replication, triggers are fired on master only, and the resulting rows are moved to the slave

AGENDA

Replication power techniques and features

- row-based
- semi-synch
- delayed
- Circular
- bi-directional
- Leveraging replication

AGENDA

row-based replication

row-based replication

- Available in 5.1 and later
- can be changed using “binlog_format”
  - row
  - mixed
  - statement (default)

row-based replication

- DO
  - when you have expensive queries in the master
  - when you are using non-deterministic functions
- DON’T
  - When you are updating a lot of rows

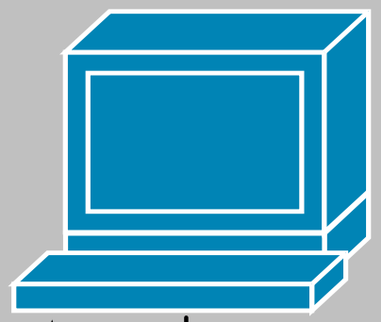
AGENDA

semi-synch replication

semi-synchronous replication

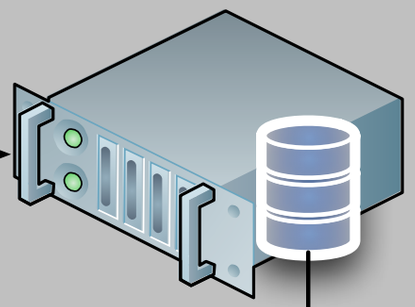
- Available in 5.5 and higher
- Makes sure that at least one slave has copied the data.
- Increases reliability

transaction with regular replication



client

master



1

commit

2

execute

3

binary log

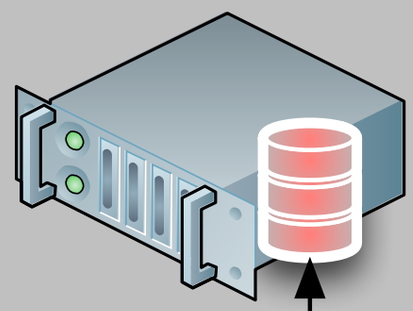
4

returns to client

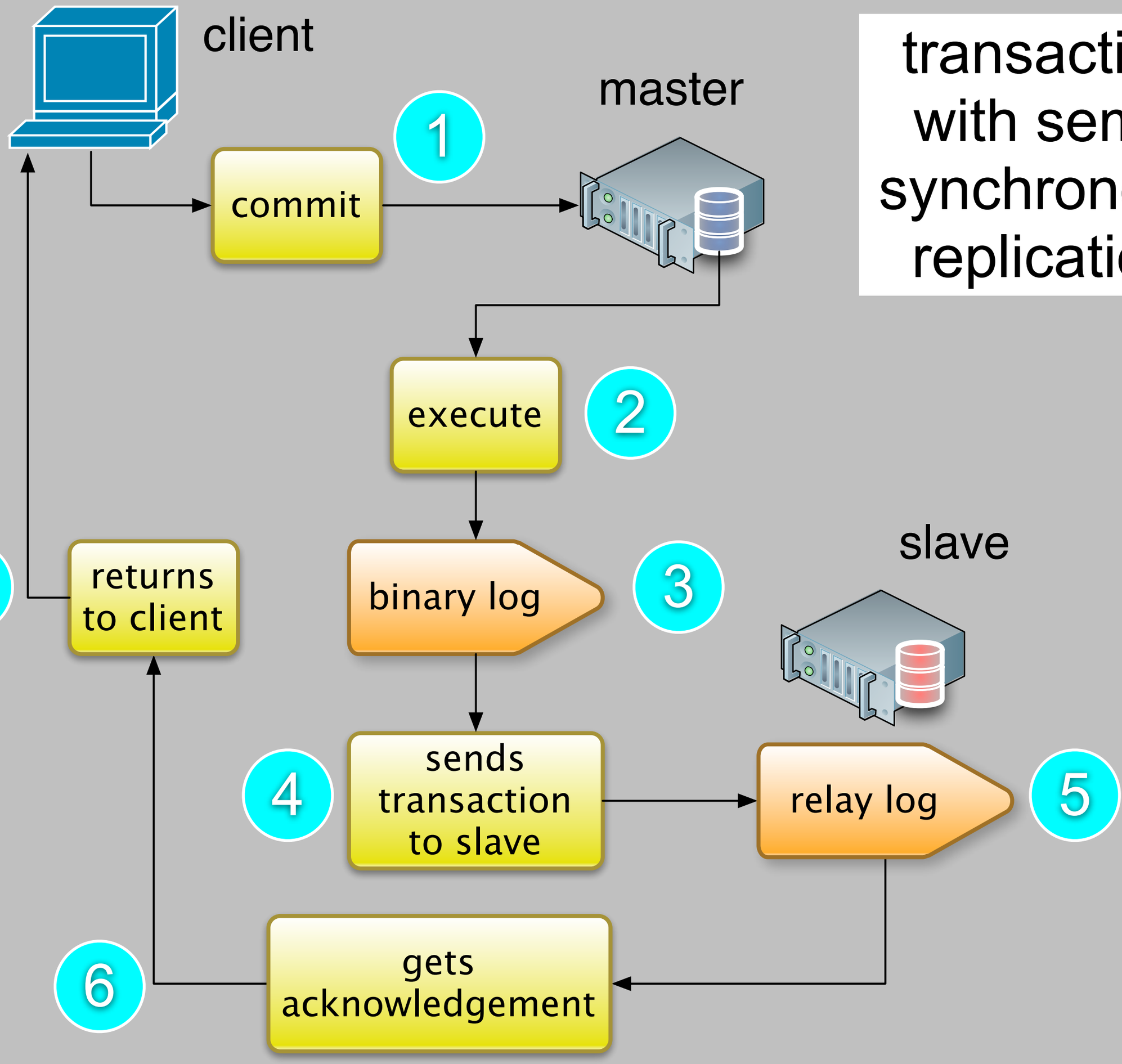
5

replication

slave



transaction with semi-synchronous replication



semi-synchronous replication in practice

- installation:
  - it's a plugin.
  - Actually, two plugins

semi-synch replication install

```
# in the master
plugin-load=rpl_semi_sync_master=semisync_master.so
rpl_semi_sync_master_enabled=1


# in each slave
plugin-load=rpl_semi_sync_slave=semisync_slave.so
rpl_semi_sync_slave_enabled=1


# restar all servers
```

semi-synch replication check

```
# in the master
show variables like 'rpl_semi%';
+-------------------------------+-------+
| Variable_name                 | Value |
+-------------------------------+-------+
| rpl_semi_sync_master_enabled  | ON    |
| rpl_semi_sync_master_timeout  | 10000 |
| rpl_semi_sync_master_trace_level | 32    |
| rpl_semi_sync_master_wait_no_slave | ON    |
+-------------------------------+-------+
```

semi-synch replication check

```
show status like "rpl_semi_%tx";
+-----------------------------+-------+
| variable_name               | value |
+-----------------------------+-------+
| RPL_SEMI_SYNC_MASTER_NO_TX  | 0     |
| RPL_SEMI_SYNC_MASTER_YES_TX | 0     |
+-----------------------------+-------+
```

semi-synch replication test

```
master> create table t1 ( i int);
Query OK, 0 rows affected (0.01 sec)

master> show status like "rpl_semi_%tx";
+-------------------------------+-------+
| Variable_name                 | Value |
+-------------------------------+-------+
| Rpl_semi_sync_master_no_tx    | 0     |
| Rpl_semi_sync_master_yes_tx   | 1     |
+-------------------------------+-------+
```

disabling semi-synch

```
# for each slave

set global rpl_semi_sync_slave_enabled=0;
stop slave io_thread;
start slave io_thread;
```

disabled semi-synch replication test

```
master> insert into t1 values (1);
Query OK, 1 row affected (10.00 sec)

master> show status like "rpl_semi_%tx";
+------------------------------+-------+
| Variable_name                | Value |
+------------------------------+-------+
| Rpl_semi_sync_master_no_tx   | 1     |
| Rpl_semi_sync_master_yes_tx  | 1     |
+------------------------------+-------+
2 rows in set (0.00 sec)
```

disabled semi-synch replication test

```
master> insert into t1 values (2);
Query OK, 1 row affected (0.01 sec)

master> show status like "rpl_semi_%tx";
+-------------------------------+-------+
| Variable_name                 | Value |
+-------------------------------+-------+
| Rpl_semi_sync_master_no_tx    | 2     |
| Rpl_semi_sync_master_yes_tx   | 1     |
+-------------------------------+-------+
2 rows in set (0.00 sec)
```

re-enabling semi-synch

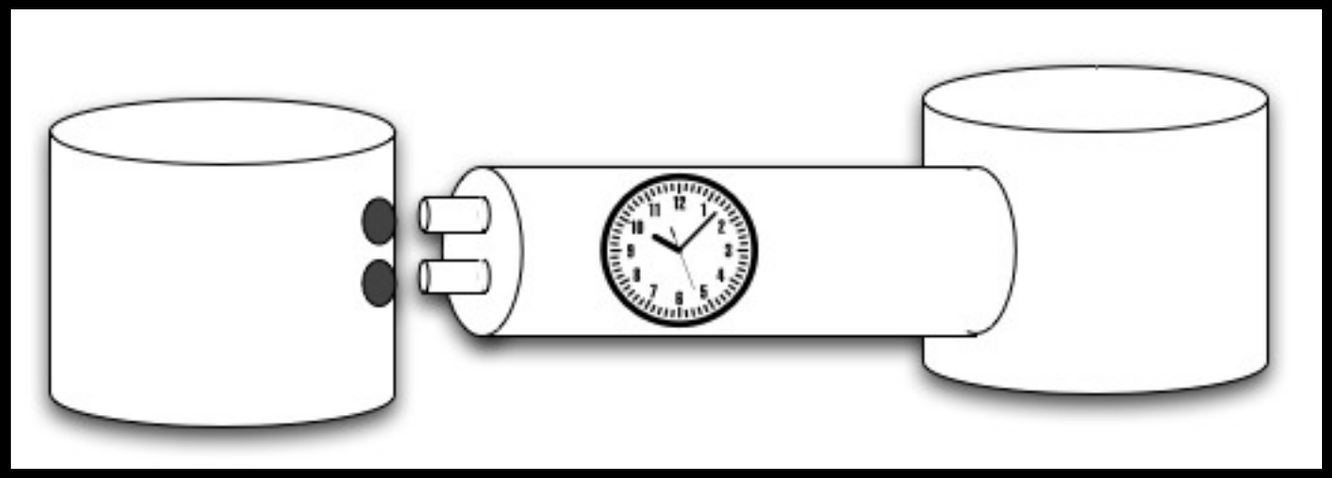
```
# in one slave

set global rpl_semi_sync_slave_enabled=1;
stop slave io_thread;
start slave io_thread;
```

reenabled semi-synch replication test

```
master> insert into t1 values (3);
Query OK, 1 row affected (0.01 sec)

master> show status like "rpl_semi_%tx";
+-------------------------------+-------+
| Variable_name                 | Value |
+-------------------------------+-------+
| Rpl_semi_sync_master_no_tx    | 2     |
| Rpl_semi_sync_master_yes_tx   | 2     |
+-------------------------------+-------+
2 rows in set (0.00 sec)
```

AGENDA

delayed replication



delayed replication in practice

```
STOP SLAVE;
change master to master_delay=60;
START SLAVE;
```

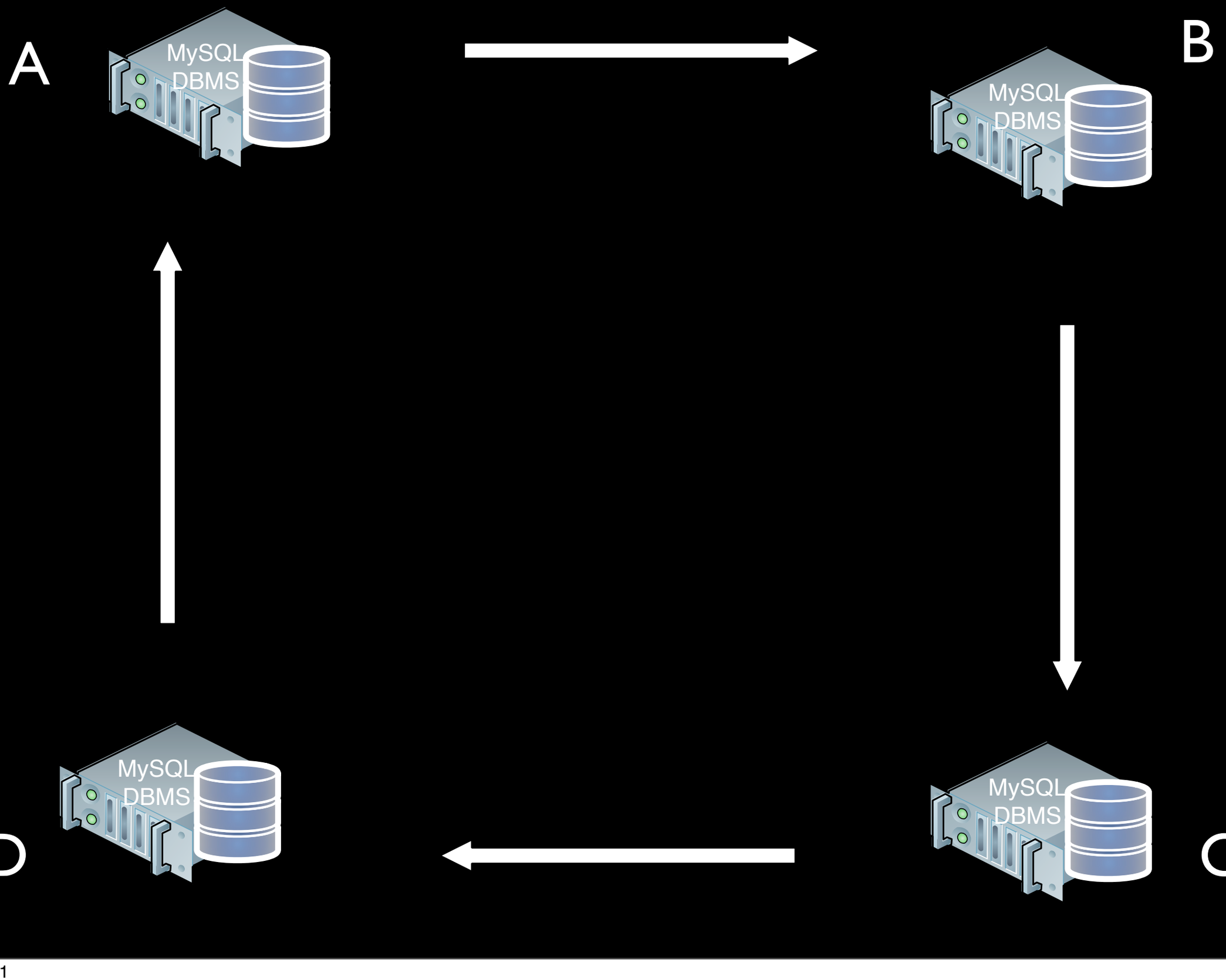
delayed replication

- DEMO

AGENDA

Circular replication

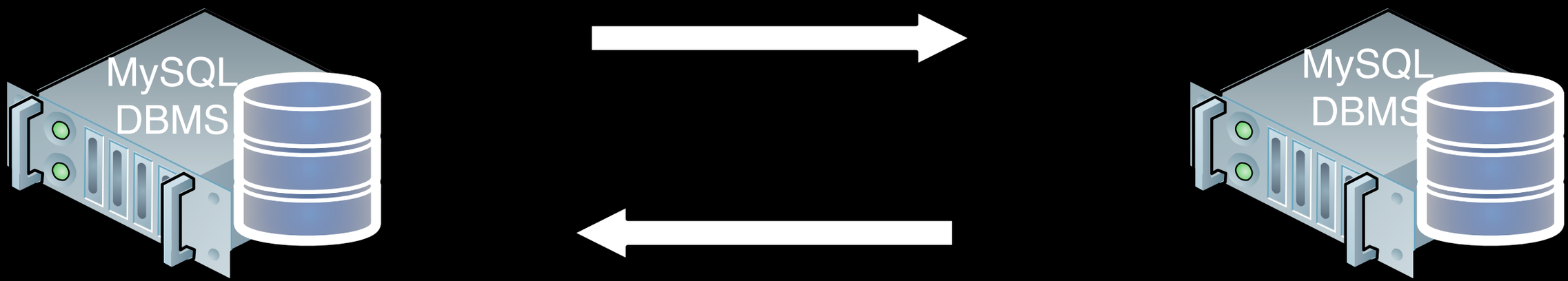
Circular



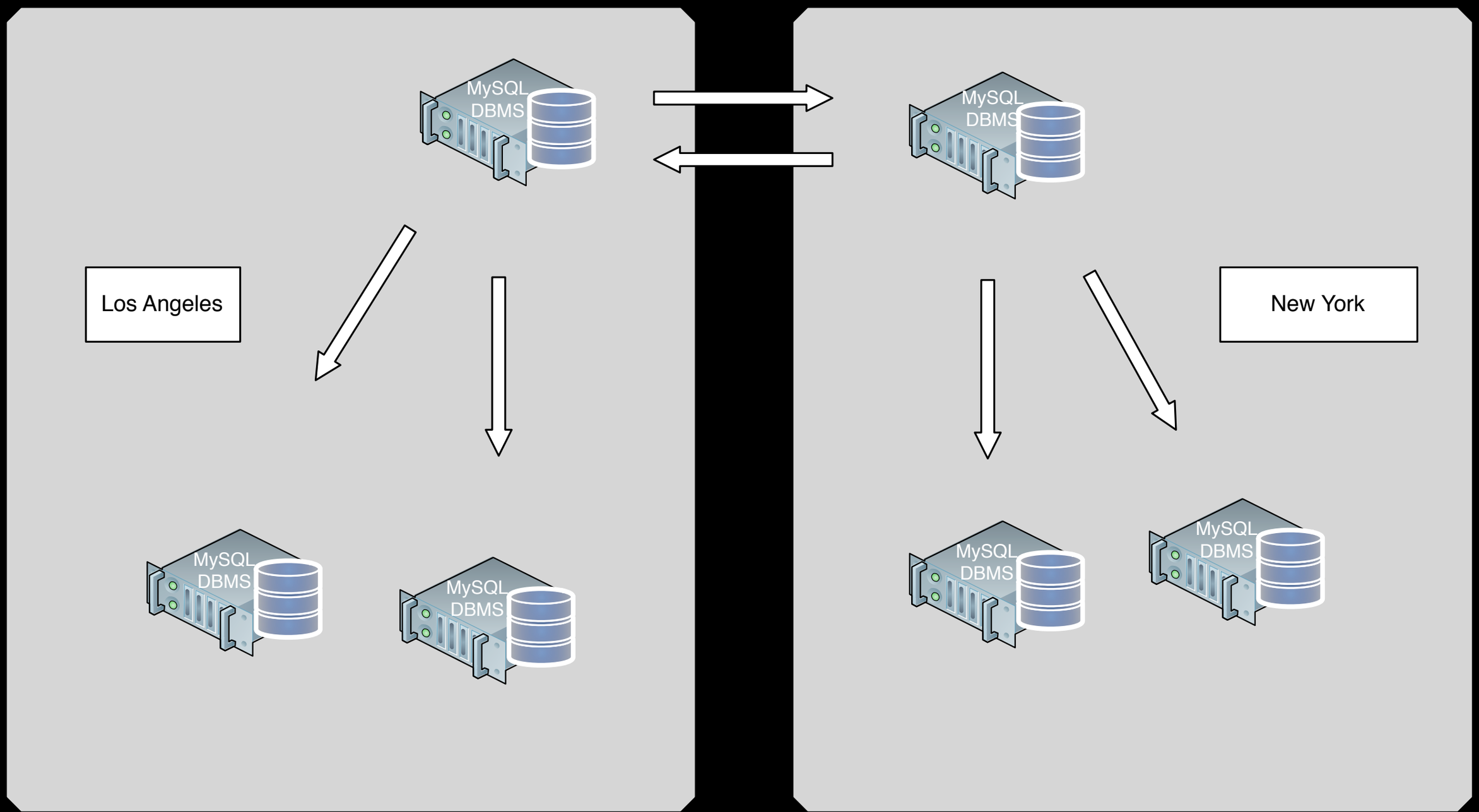
AGENDA

bi-directional replication

bi-directional replication



bi-directional replication

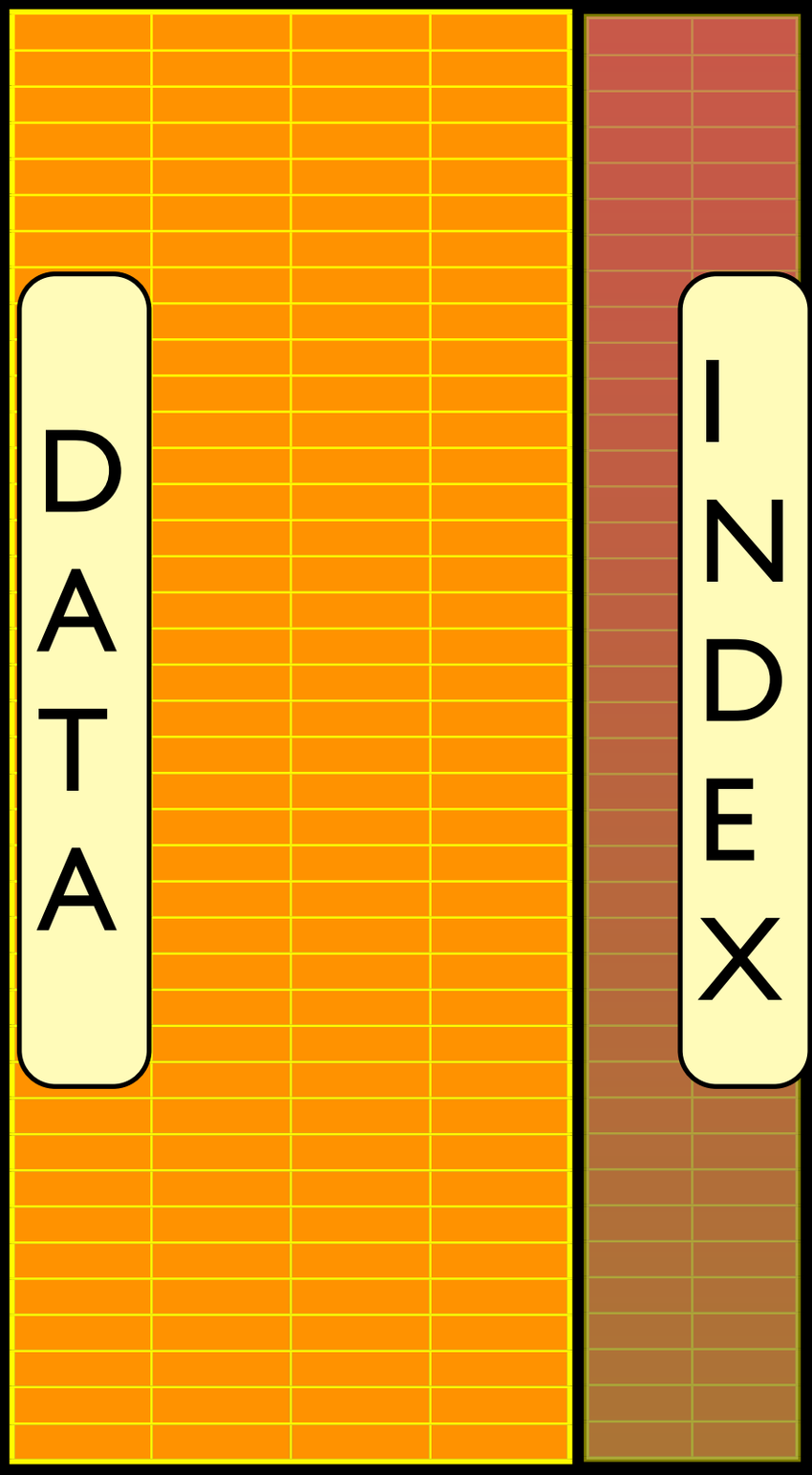


AGENDA

Leveraging replication

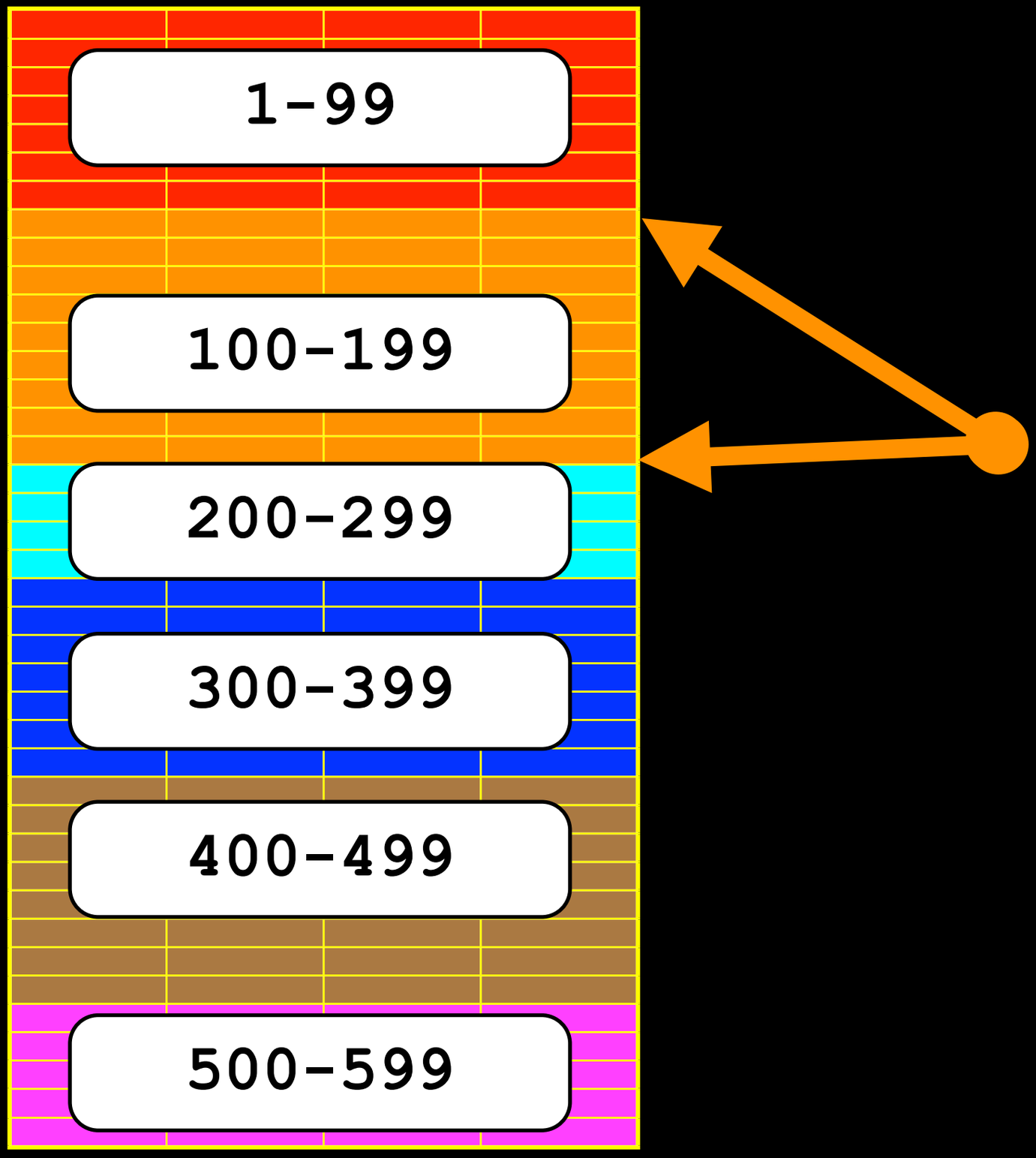
Overview of MySQL partitions

Partition pruning
1a - unpartitioned table - SINGLE RECORD



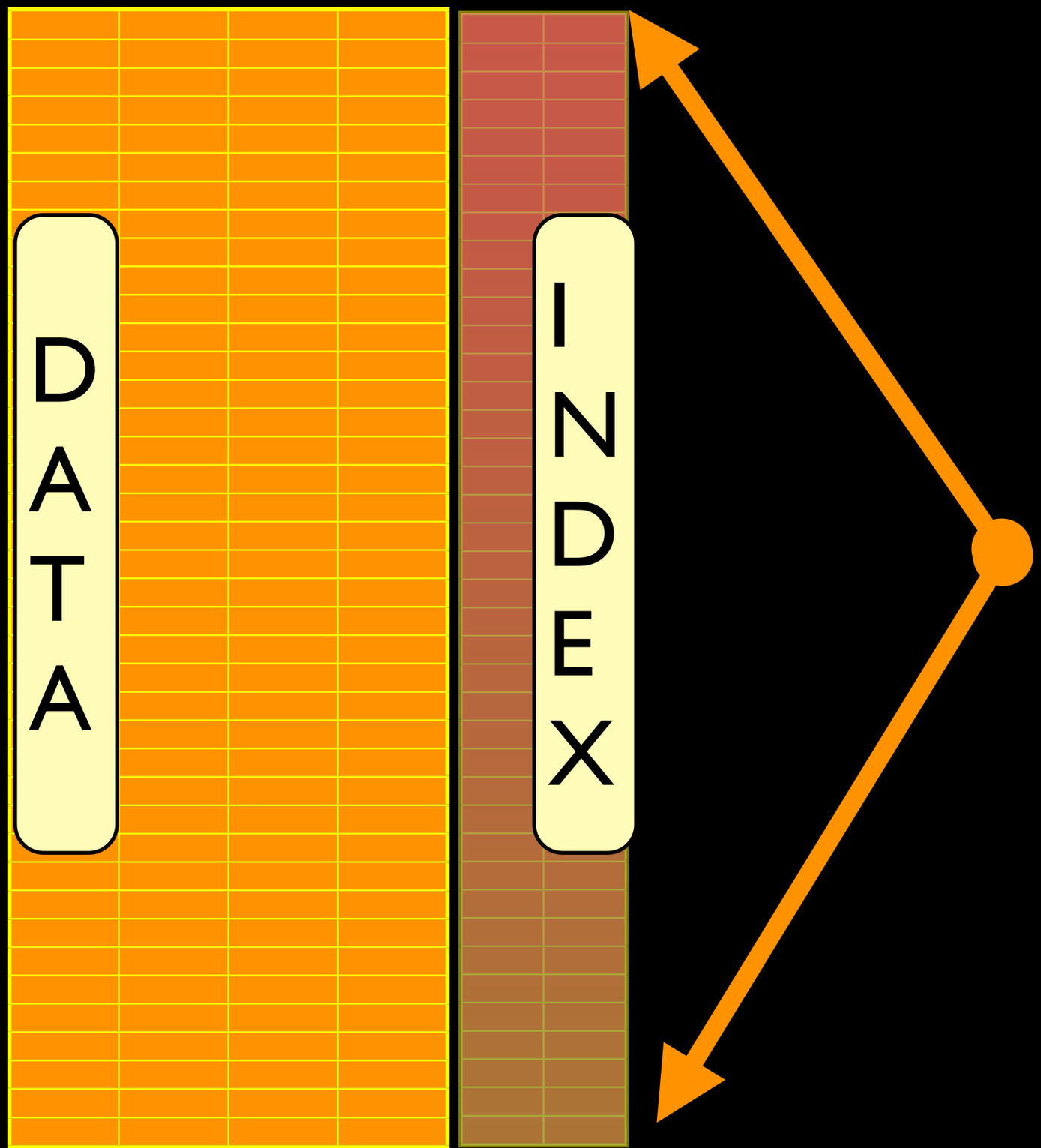

```
select *
from table_name
where colx =
120
```

Partition pruning
2a - table partitioned by colx - SINGLE REC



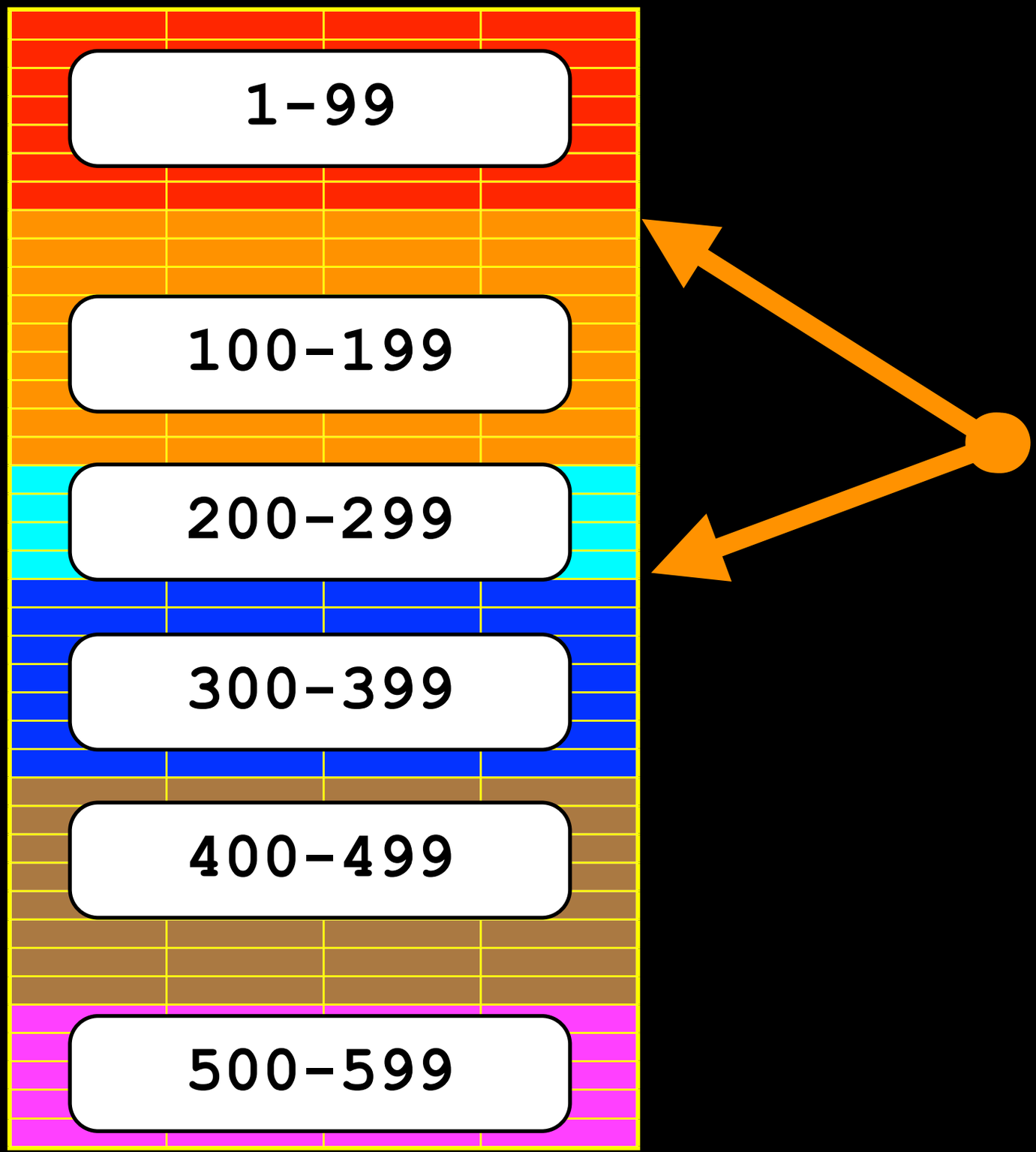

```
select *
from table_name
where colx =
120
```

Partition pruning
1a - unpartitioned table - RANGE




```
select *
from
table_name
where colx
between 120
and 230
```

Partition pruning
2a - table partitioned by colx - RANGE




```
select *
from
table_name
where colx
between 120
and 230
```

Storage comparison

engine	storage (GB)
innodb (with PK)	330
myisam (with PK)	141
archive	13
innodb partitioned (no PK)	237
myisam partitioned (no PK)	107
archive partitioned	13

Benchmarking results

engine	6 month range
InnoDB	4 min 30s
MyISAM	25.03s
Archive	22 min 25s
InnoDB partitioned by month	13.19
MyISAM partitioned by year	6.31
MyISAM partitioned by month	4.45
Archive partitioned by year	16.67
Archive partitioned by month	8.97

Partitions limits

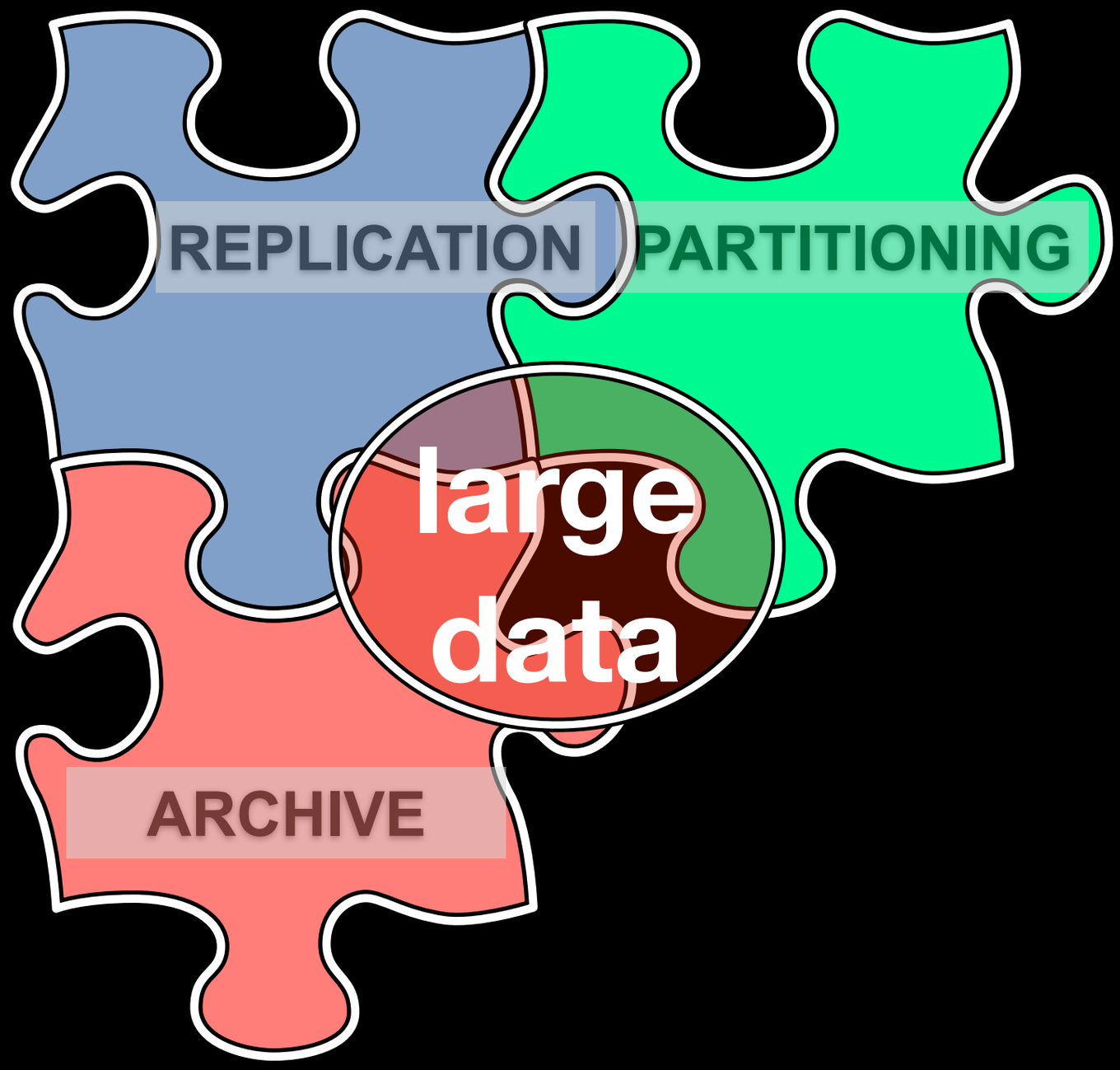
- Little concurrency
- Need of using the partitioning column

The ARCHIVE storage engine

Size matters

Innodb

Archive

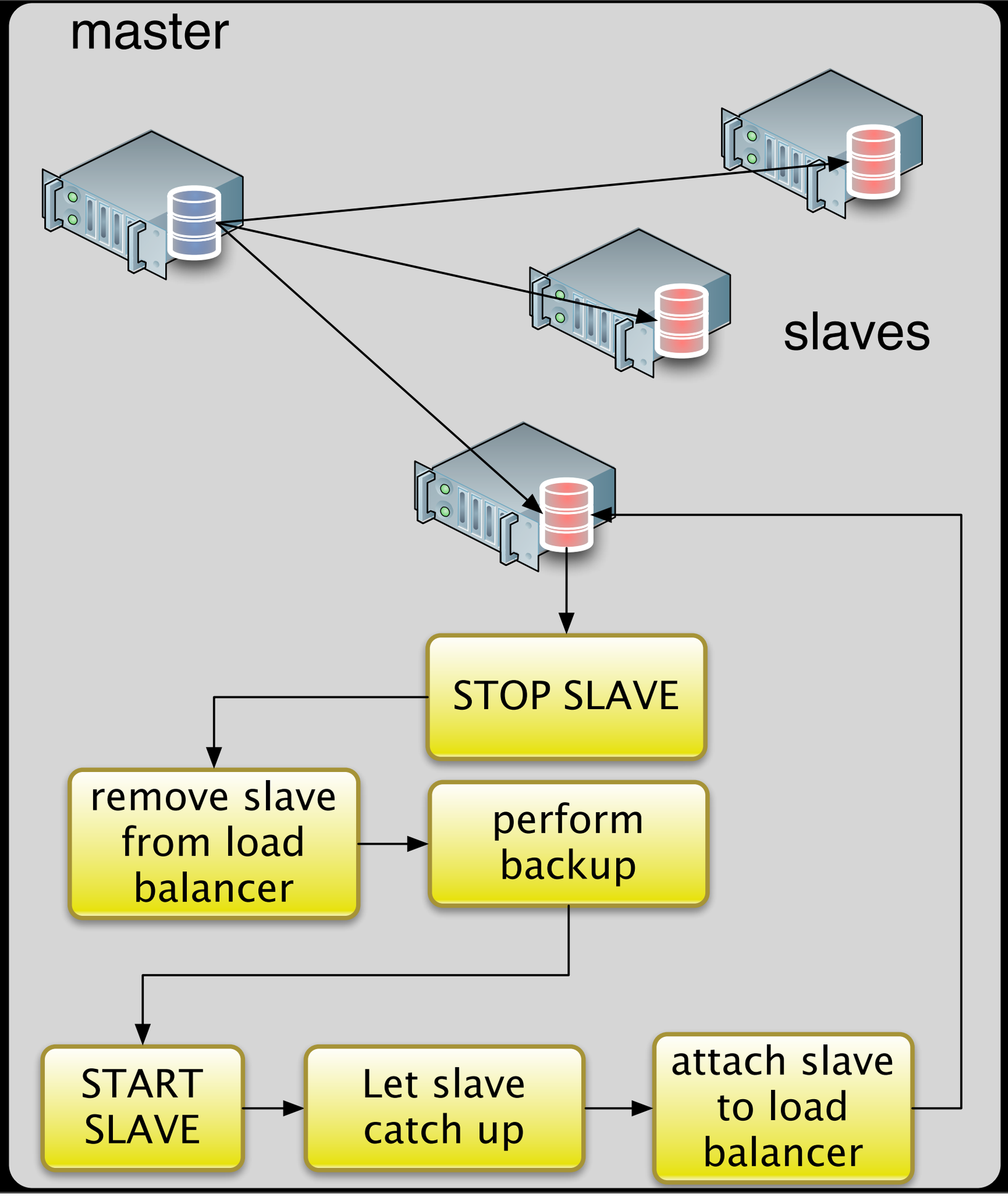

REPLICATION
PARTITIONING
large
data
ARCHIVE

Replication

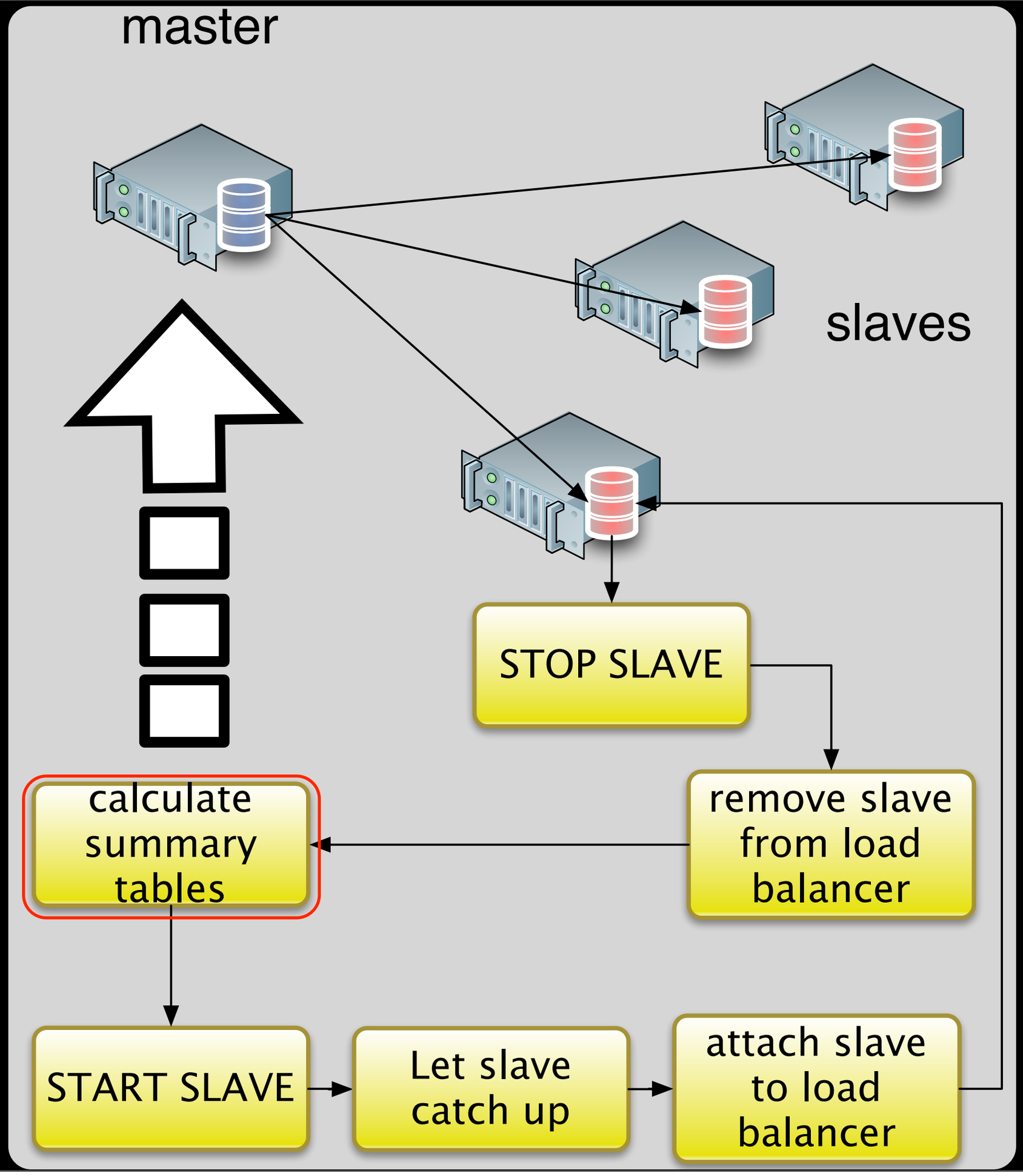
=

let the slave do the dirty work

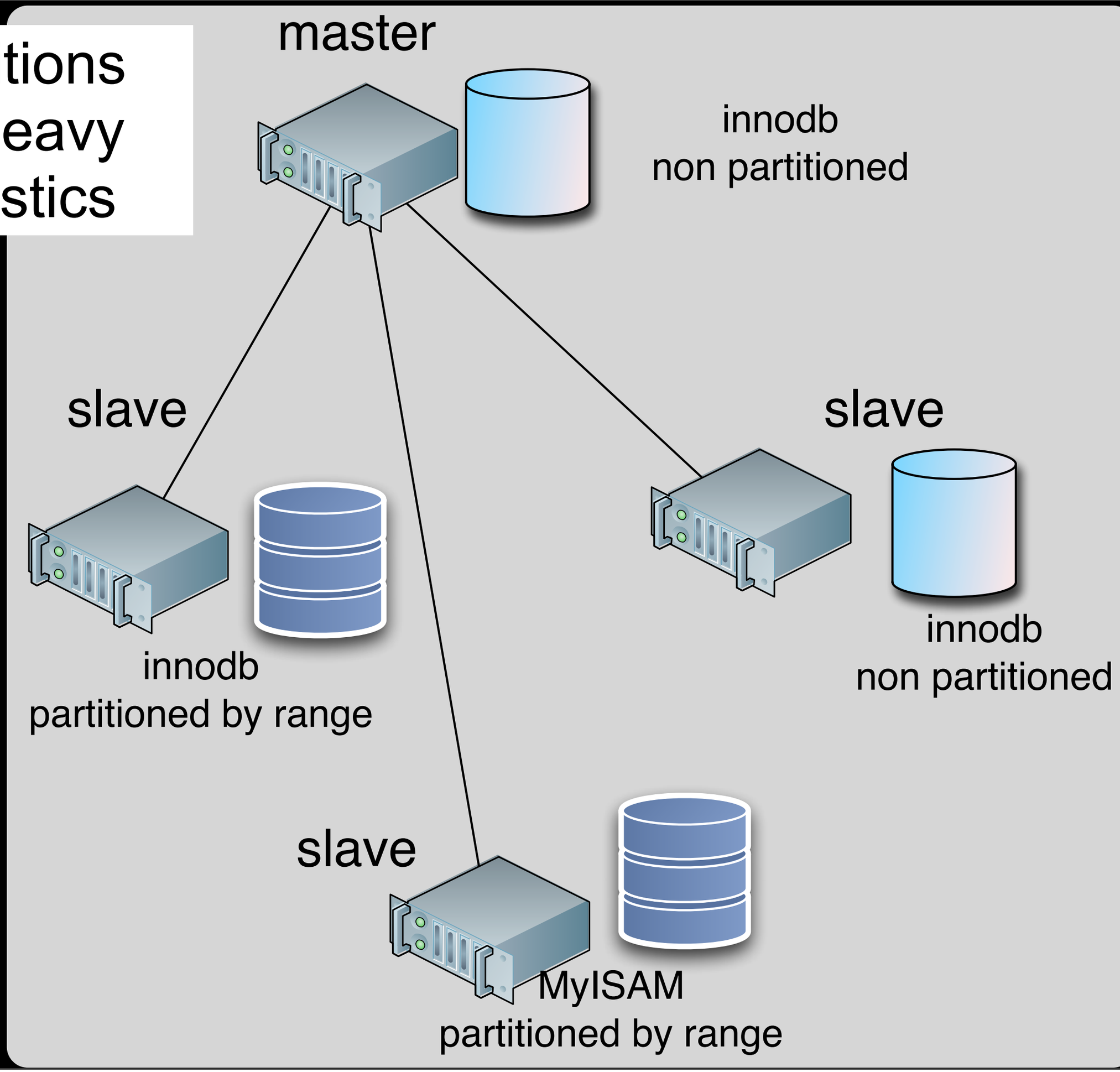
backup


master
slaves
STOP SLAVE
remove slave from load balancer
perform backup
START SLAVE
Let slave catch up
attach slave to load balancer

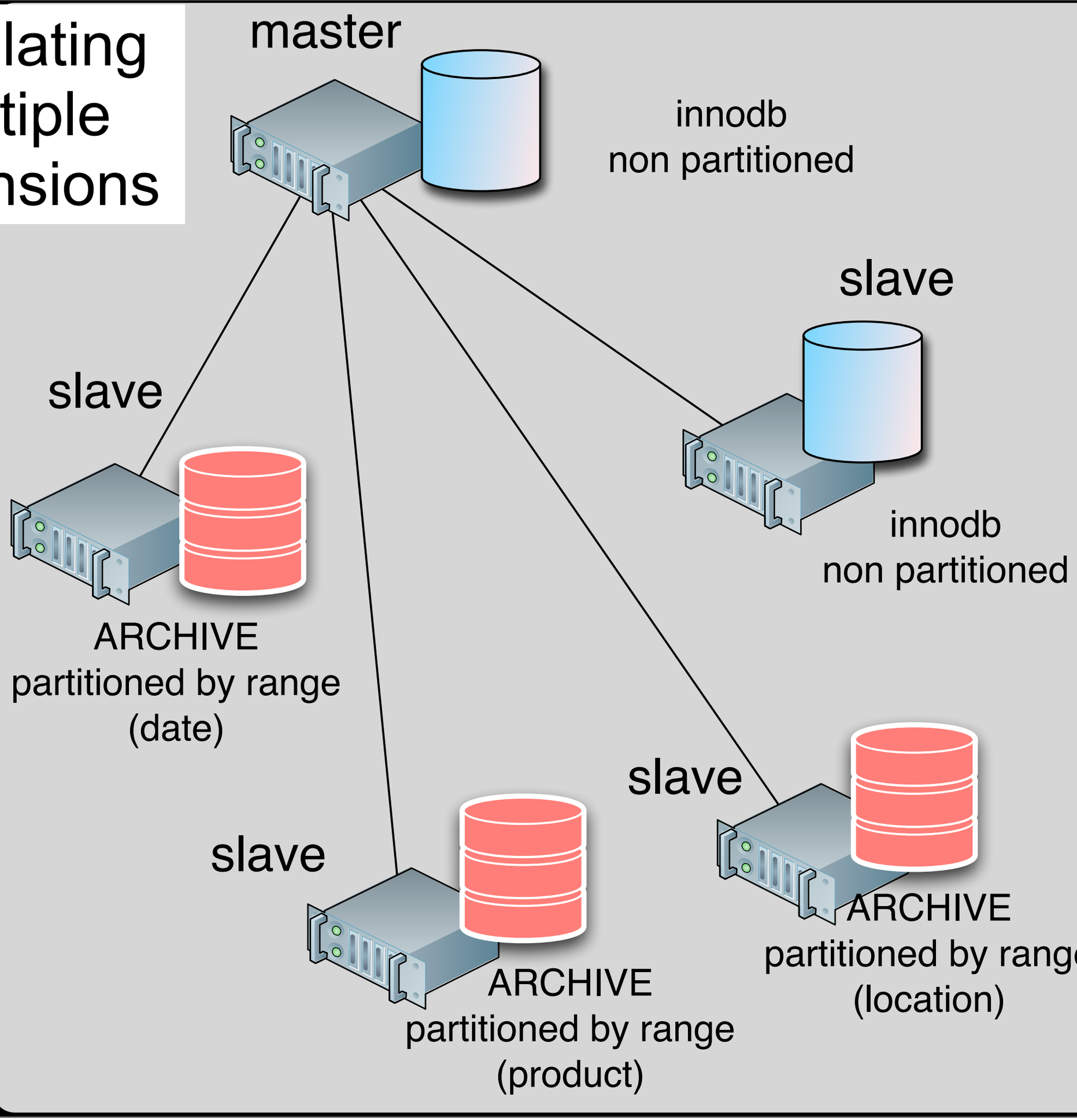
make summary tables



Partitions for heavy statistics



Simulating multiple dimensions



AGENDA

Tools and tips

- MySQL Sandbox
  - Replaying binary logs
  - Filtering binary logs
- Baron Schwartz's Maatkit
- Shlomi Noach's openark kit
- Facebook's Online Schema Change
- Measuring replication speed
- Detecting if replication is on

AGENDA

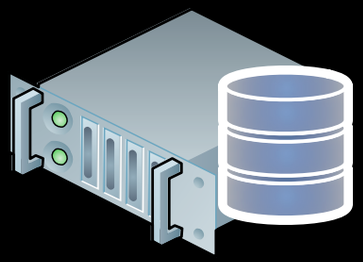
MySQL Sandbox

MySQL Sandbox

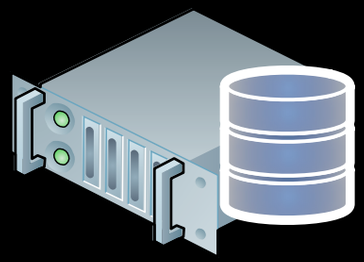
http://mysqlsandbox.net

- Free software (Perl under GPL)
- One (unix) host
- Many database servers
- Single or multiple sandboxes
- Customized scripts to use the servers
- Standard or circular replication
- Installs **IN SECONDS**

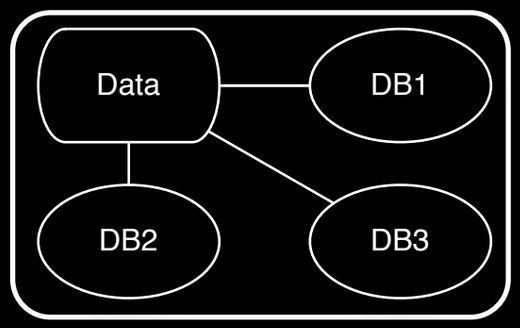
overview



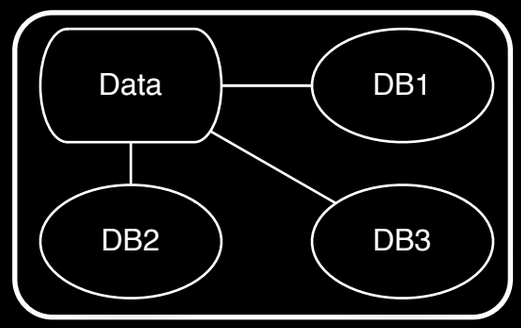
MySQL
server



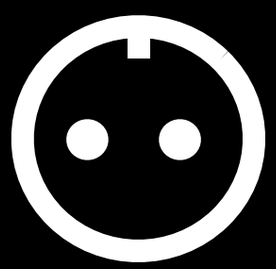
MySQL
server



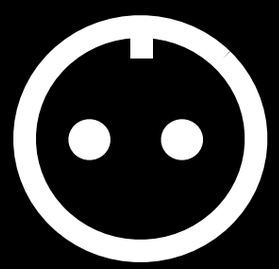
DATA DIRECTORY



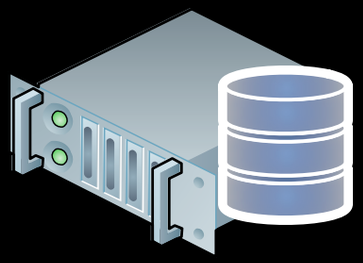
PORT



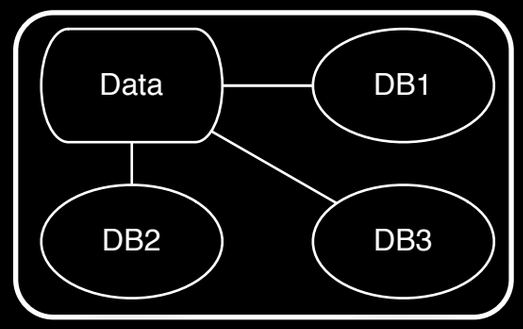
SOCKET



overview

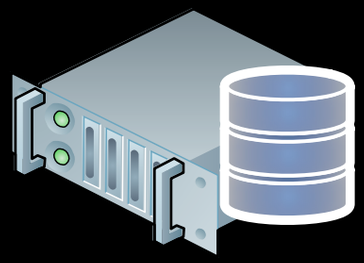


MySQL
server

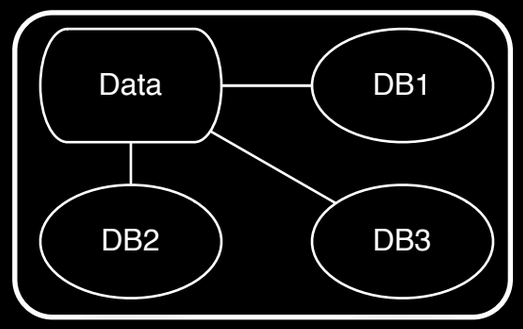


/var/lib/mysql

SAME
DATA
DIRECTORY?



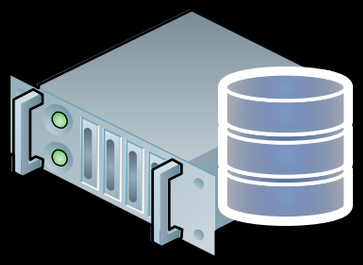
MySQL
server



/var/lib/mysql

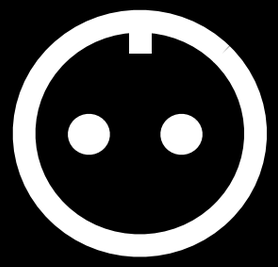
DATA CORRUPTION

overview



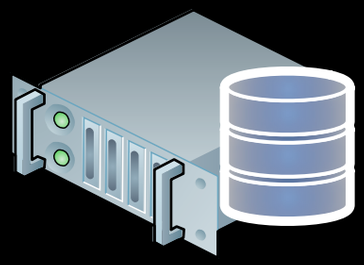
MySQL
server

3306



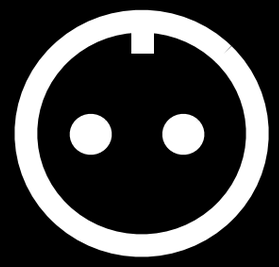
/tmp/mysql.sock

SAME
PORT or
SOCKET?



MySQL
server

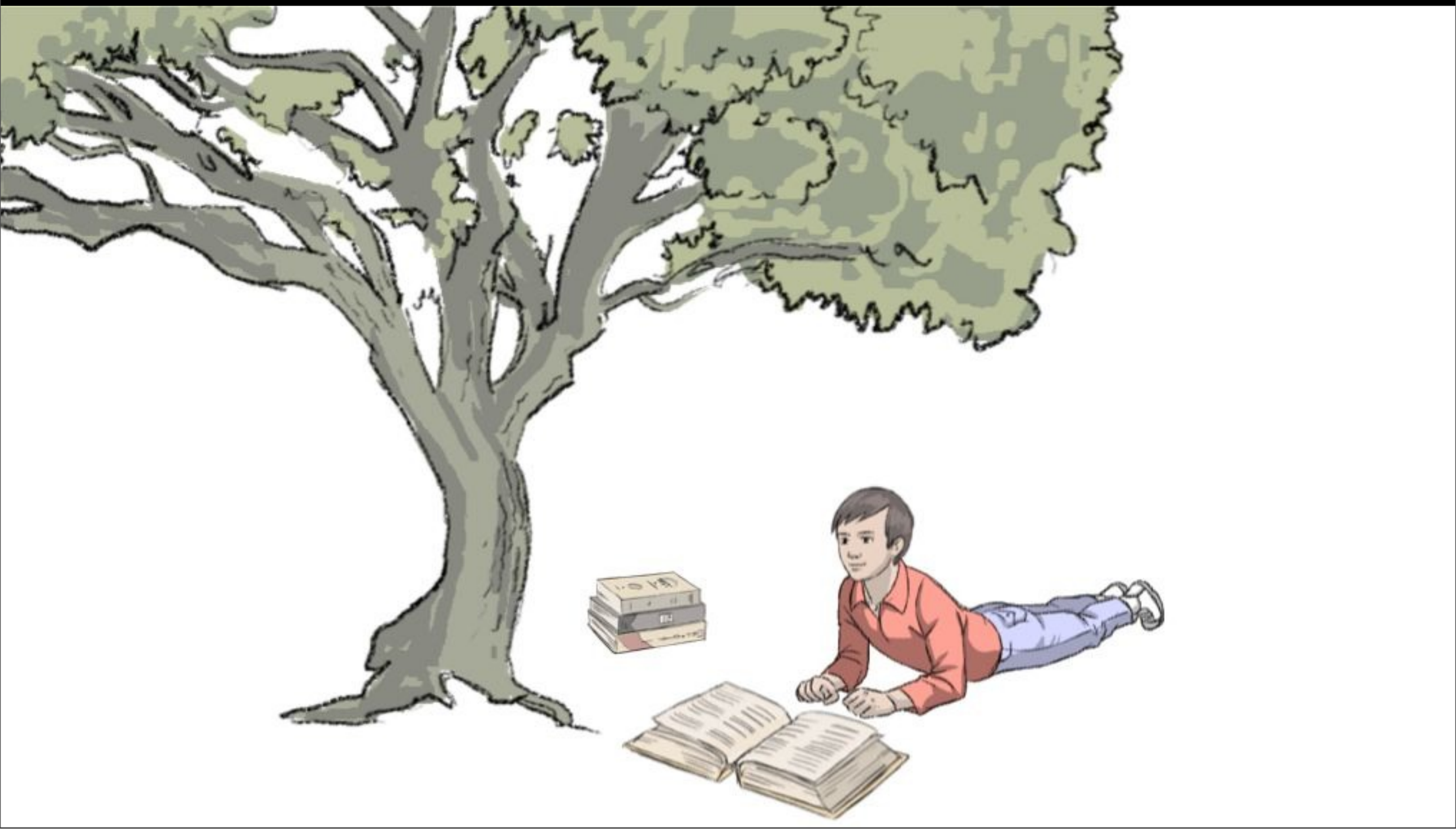
3306



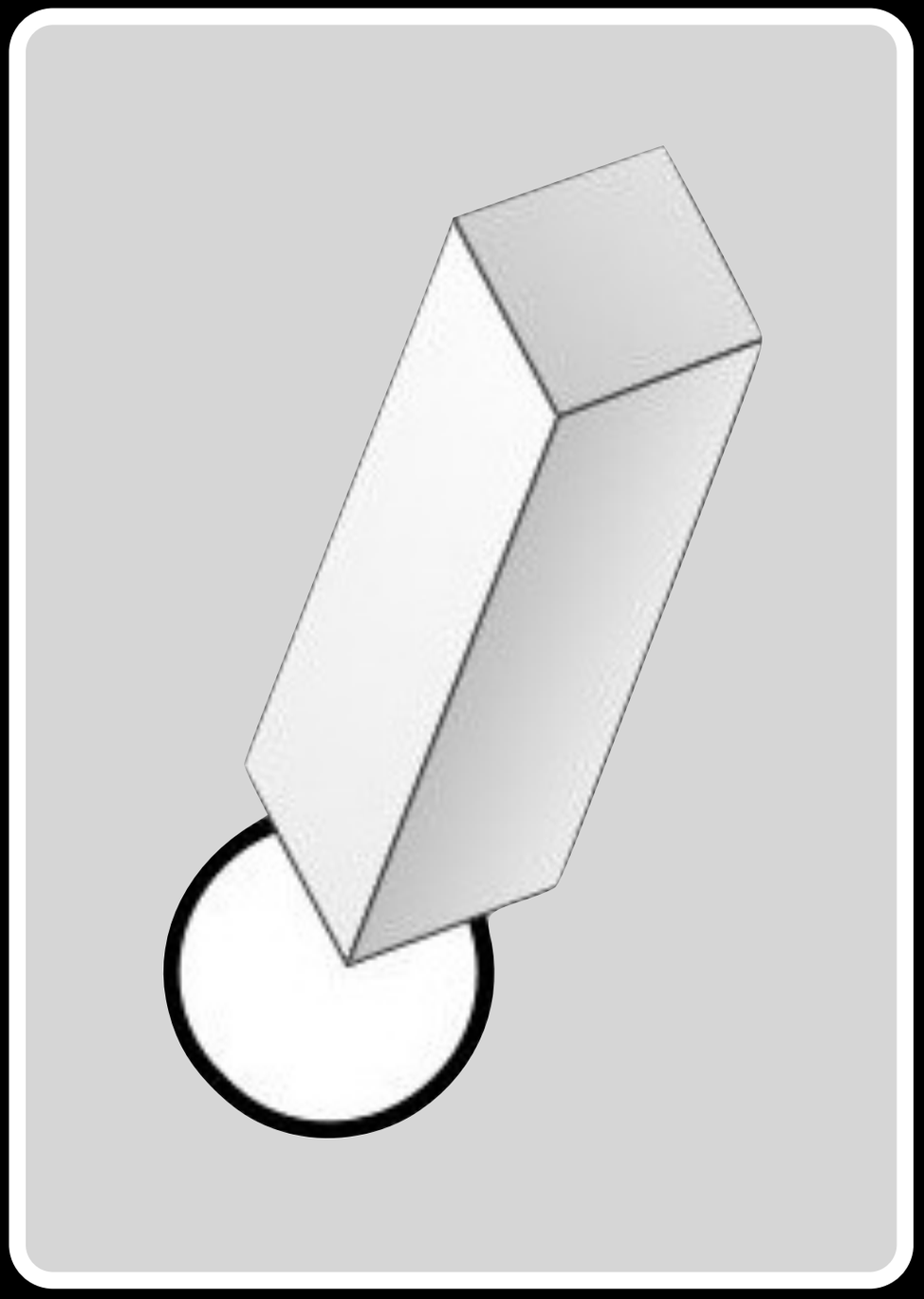
/tmp/mysql.sock

DOES NOT START

The hard way (1)



the hard way (2)



The easy way

```
$ make_sandbox \
    /path/to/mysql-5.1.54_linux.tar.gz


# it should work always
```

The easier way

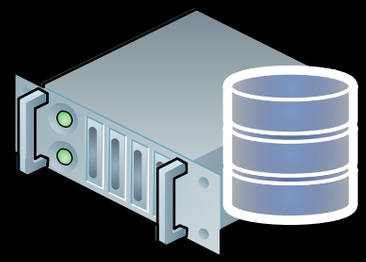
```
$ make_sandbox 5.1.54


# Needs some preliminary work
```

The easiest way

```
$ sb 5.1.54
```

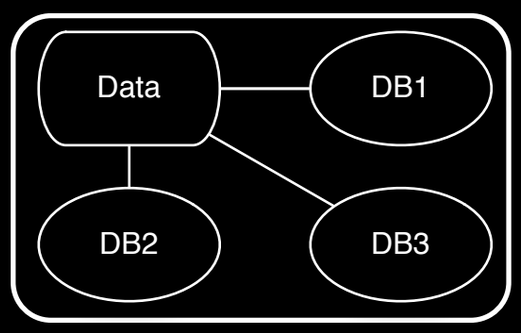
```
# Needs the same preliminary work
```

MySQL Sandbox



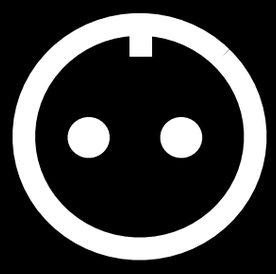
MySQL
server

VERSION



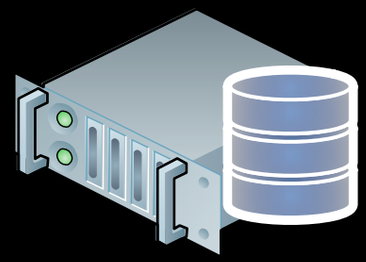
$SANDBOX_HOME/msb_VERSION/data

VERSION



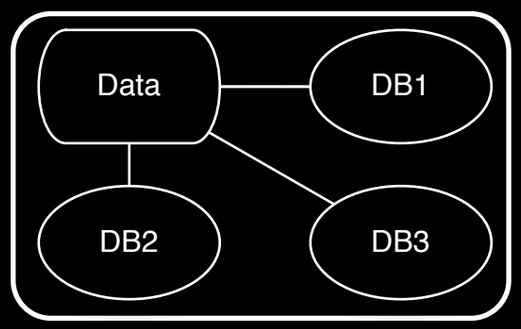
/tmp/mysql_VERSION.sock

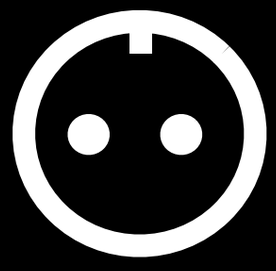
MySQL Sandbox



MySQL
server

5.1.54



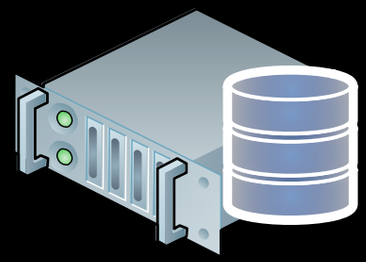


```
$SANDBOX_HOME/msb_5_1_54/data
```

5154

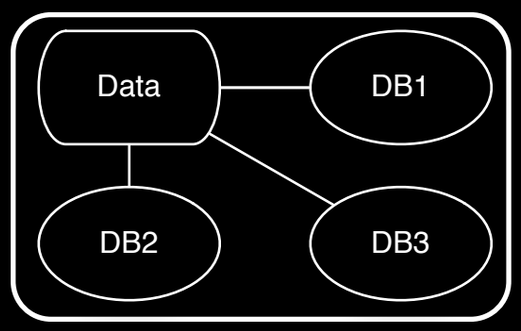
```
/tmp/mysql_5154.sock
```

MySQL Sandbox



MySQL
server

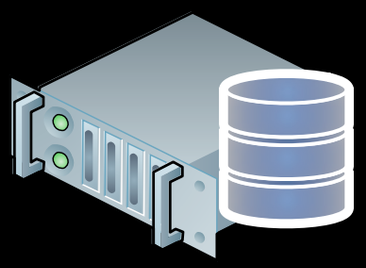
5.5.9



$SANDBOX_HOME/msb_5_5_09/data

5509

/tmp/mysql_5509.sock

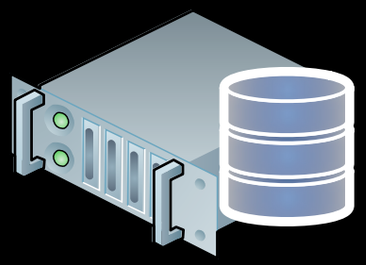


MySQL
server

Single Sandbox

customized scripts

```
start
stop
restart
status
clear
send_kill
use
```



MySQL
server

Multiple Sandbox

customized scripts

```
start_all
stop_all
restart_all
status_all
clear_all
send_kill_a
ll
use_all
```

m	n1
s1	n2
s2	n3

Where do you get it

- from CPAN

```
sudo cpan MySQL::Sandbox
```

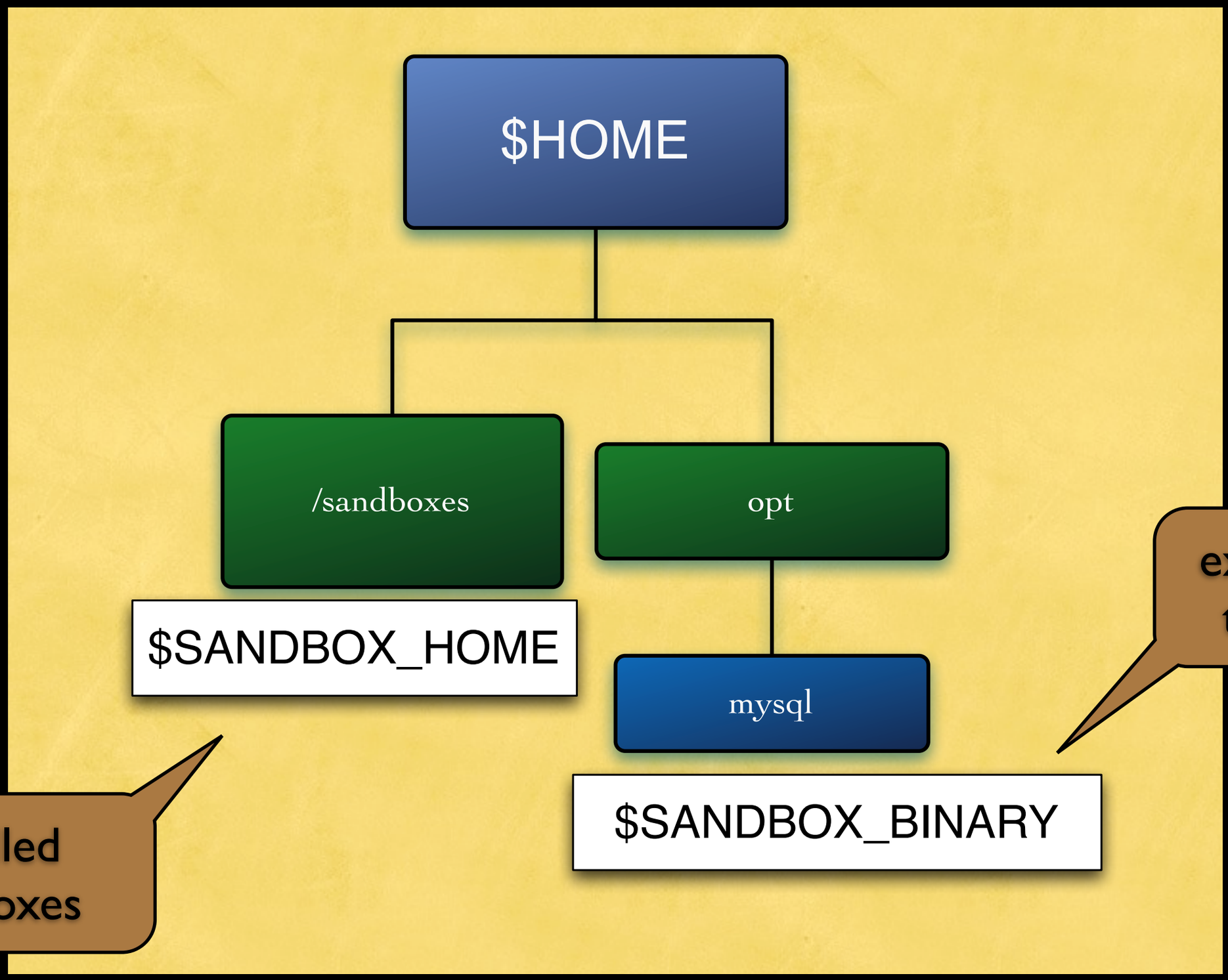
- from launchpad

```
http://launchpad.net/mysql-sandbox
```

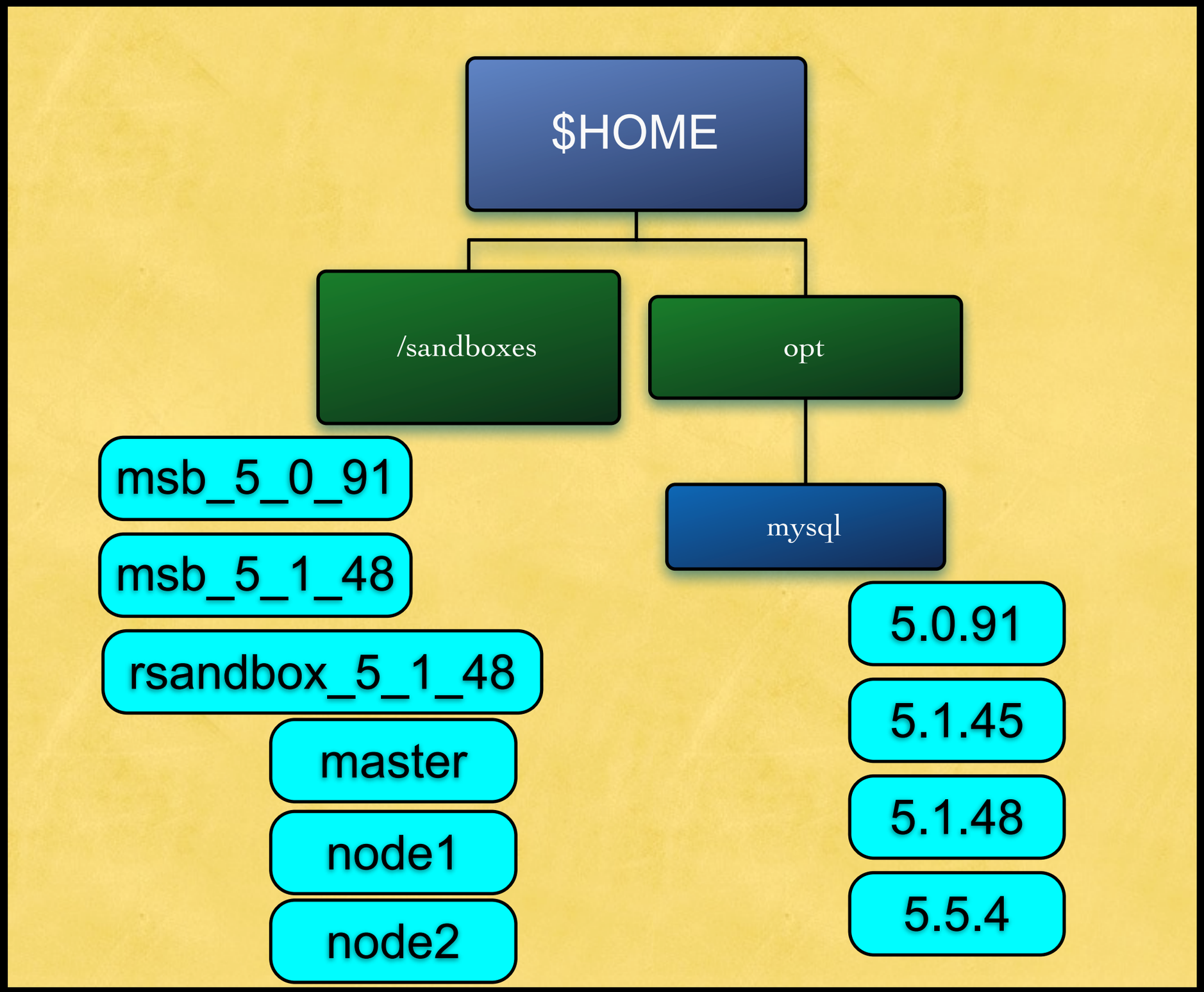
The easy replication way

```
$ make_replication_sandbox \
    /path/to/mysql-5.1.54_linux.tar.gz

# or, after some preparation

$ make_replication_sandbox 5.1.54
```

default architecture



default architecture




COOKBOOK

creating a single sanbox

```
make_sandbox \
    /path/to/mysql-X.X.XX-OS.tar.gz
```

using a single sanbox

```
# after
# make_sandbox \
#   /path/to/mysql-X.X.XX-OS.tar.gz

$ cd $SANDBOX_HOME/msb_X_X_XX
$ ./use
```

creating a single sanbox with a specific options file

```
make_sandbox \
    /path/to/mysql-X.X.XX-OS.tar.gz \
    --my_file=/path/to/my.cnf
```

easily create a sandbox after the first one

The long way

```
$ cd $HOME/opt/mysql      # $SANDBOX_BINARY
$   gunzip -c \
  /path/to/mysql-5.1.34-osx10.5-x86.tar.gz \
   | tar -xf -
$ mv mysql-5.1.34-osx10.5-x86 5.1.34
$ make sandbox 5.1.34
```

easily create a sandbox after the first one

The short way

```
$ make_sandbox \
  path/to/mysql-5.1.34-osx10.5-x86.tar.gz \
  --export_binaries
```

starting a single sanbox

```
$ cd $SANDBOX_HOME/msb_X_X_XX
$ ./start
```

starting a single sanbox with temporary options

```
$ cd $SANDBOX_HOME/msb_X_X_XX
$ ./start --option=value
```

```
$ ./restart --option=value
```

```
$ ./start --key-buffer=20000000
```

creating a sandbox with custom port and directory

```
$ make_sandbox 5.1.34 \
    --sandbox_port=7800 \
    --sandbox_directory=mickeymouse
```

creating a sandbox with automatic port checking

```
$ make_sandbox 5.1.34 --check_port


# if 5.1.34 is free
#     port=5134
#     directory=msb_5_1_34
# else
#     port=5135 (or the first free)
#     directory=msb_5_1_34_a
```

create a replication sandbox

```
$ make_replication_sandbox \
  path/to/mysql-5.1.34-osx10.5-x86.tar.gz
```

create a circular replication sandbox

```
$ make_replication_sandbox \
  --circular=4 \
  path/to/mysql-5.1.34-osx10.5-x86.tar.gz
```

changing port to an existing sandbox

```
$ sbtool -o port \
    -s /path/to/source/sandbox \
    --new_port=XXXX
```

installing the innodb plugin

```
$ sbtool -o plugin \
    --plugin=innodb \
    -s /path/to/source/sandbox
```

creating a replication sandbox with new base port

```
$ make_replication_sandbox \
    --replication_directory=newwdir \
    --check_base_port 5.0.79

# Creates a replication directory under
# $SANDBOX_HOME/newdir
# The previous one is preserved.
# No conflicts happen
```

AGENDA

Replaying binary logs

AGENDA

Filtering binary logs

AGENDA

Maatkit

maatkit

- mk-heartbeat Monitor MySQL replication delay.
- mk-purge-logs Purge binary logs on a master based on purge rules.
- mk-slave-delay Make a MySQL slave server lag behind its master.
- mk-slave-find Find and print replication hierarchy tree of MySQL slaves.
- mk-slave-move Move a MySQL slave around in the replication hierarchy.
- mk-slave-prefetch Pipeline relay logs on a MySQL slave to pre-warm caches.
- mk-slave-restart Watch and restart MySQL replication after errors.
- mk-table-checksum Perform an online replication consistency check, or checksum MySQL tables efficiently on one or many servers.
- mk-table-sync Synchronize MySQL table data efficiently.
- http://www.maatkit.org/

Facebook Online Schema Change

AGENDA

Facebook Online Schema Change

- Based on one of Shlomi Noach OpenArk utilities
- Developed in PHP by Facebook

- http://www.facebook.com/note.php?note_id=430801045932

AGENDA

OpenArk

openark

- oak-get-slave-lag: print slave replication lag and terminate with respective exit code.
- oak-online-alter-table: perform a non-blocking ALTER TABLE operation.
- oak-purge-master-logs: purge master logs, depending on the state of replicating slaves.
- oak-show-replication-status: show how far behind are replicating slaves on a given master.
- http://code.openark.org/forge/openark-kit

AGENDA

Check replication speed

check replication speed

- Write a time to the master
- Retrieve it from the slave
- Compare times
- http://forge.mysql.com/tools/tool.php?id=5

AGENDA

Replication outside the box

- Seamless failover
- Parallel replication
- Multiple source replication
- Multiple master replication

Advanced MySQL Replication for the masses

Once Upon A Time,

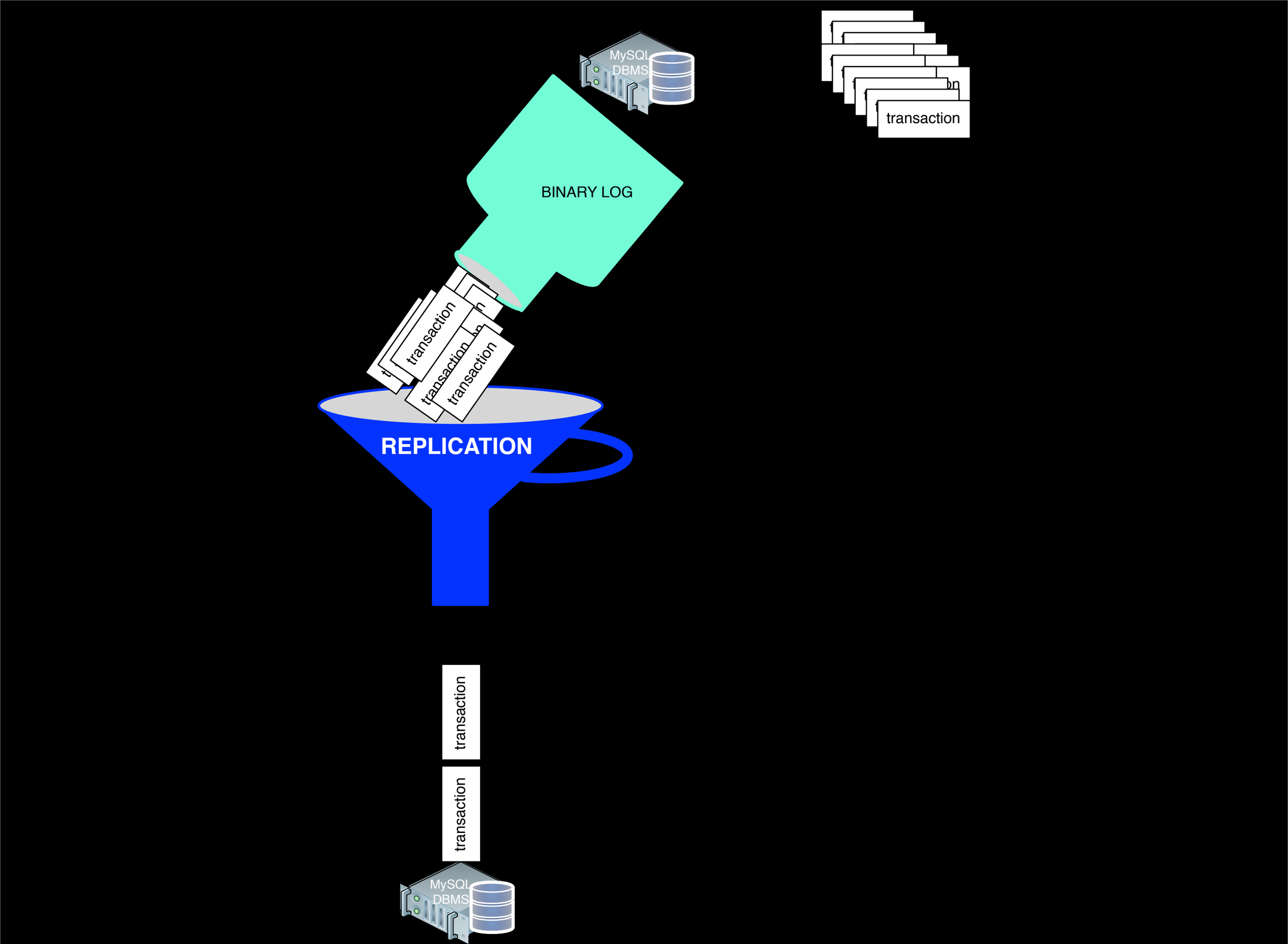
In The Life Of A

Database

Consultant ...

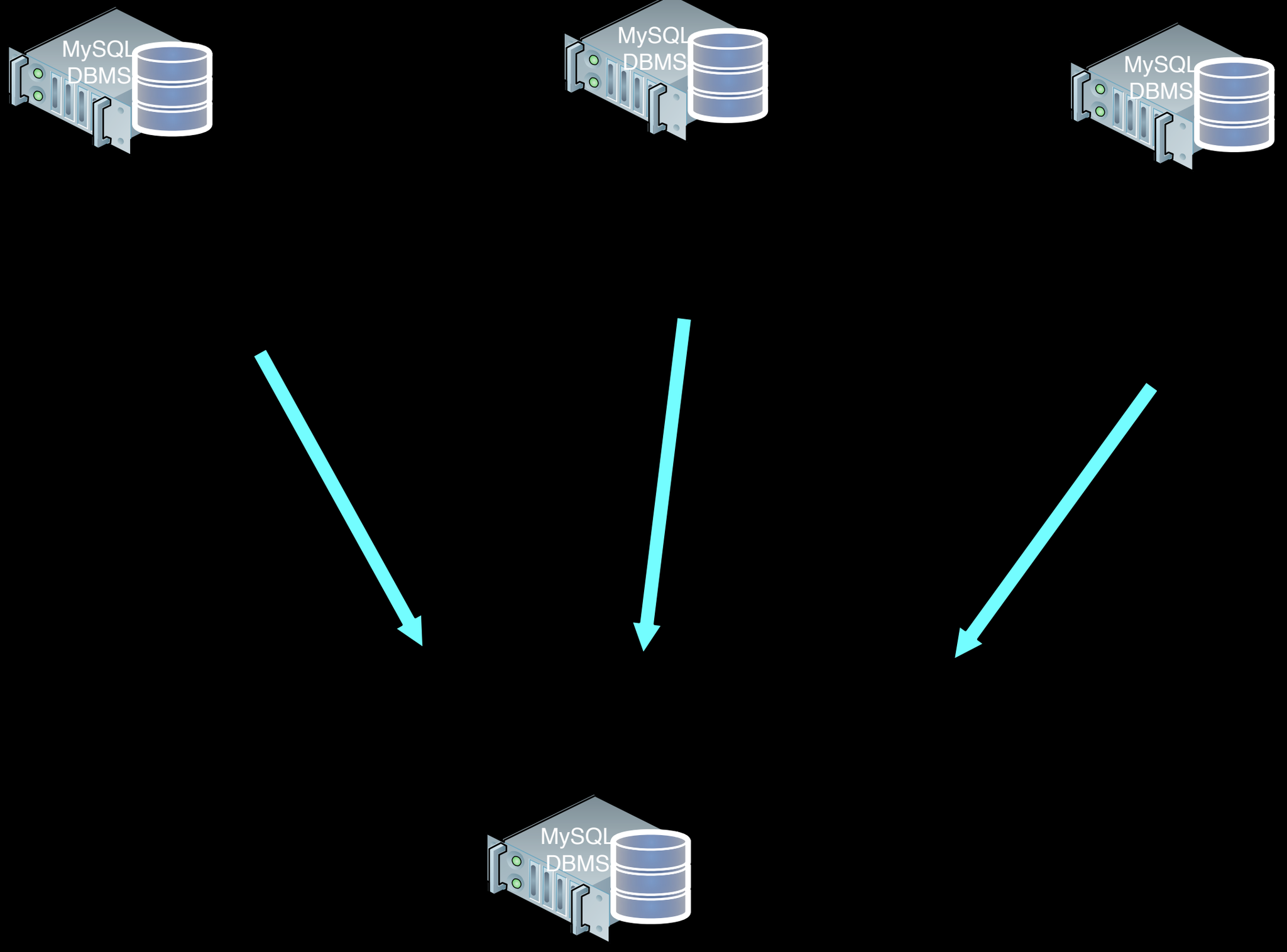
The story of a steel foundry

- Used MySQL databases to store production monitoring data
- Inserted a zillion records per second.
- Slaves often lagged behind


MySQL
DBMS
transaction
BINARY LOG
transaction
transaction
transaction
REPLICATION
transaction
transaction
MySQL
DBMS

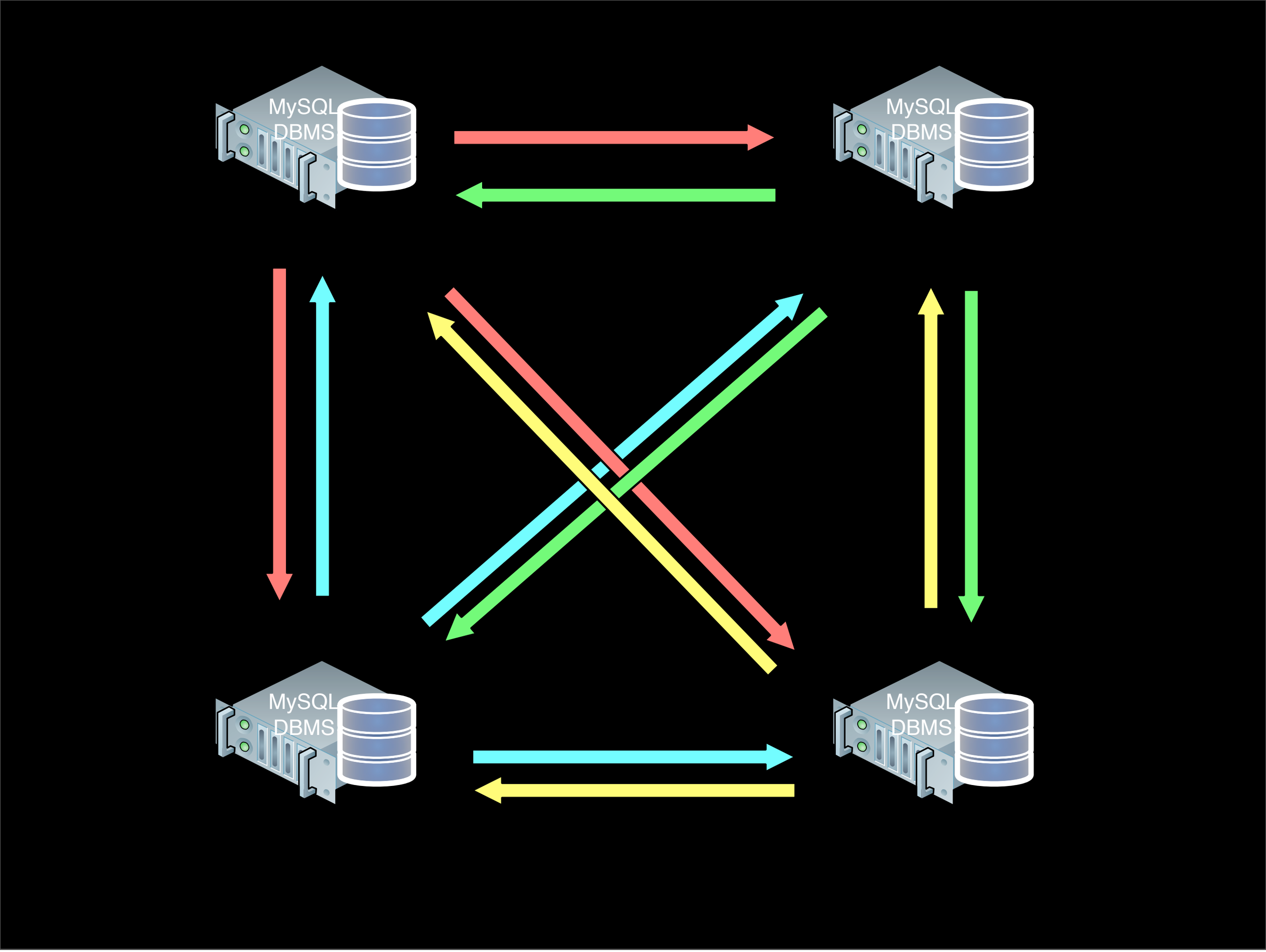
The story of a shoe maker

- Had a successful business, spread to a dozen stores.
- Needed to aggregate the data from the stores in his headquarters database.


MySQL
DBMS
MySQL
DBMS
MySQL
DBMS
MySQL
DBMS

The story of a widgets seller

- Had a successful business, designed for one server.
- Products were created in several sites.
- Needed to allow insertions from more than one master at the time


MySQL
DBMS
MySQL
DBMS
MySQL
DBMS
MySQL
DBMS

All these stories tell us:

Nice dream, but MySQL can't do it

Enter Tungsten Replicator

continuent
Open. Always Available.

Tungsten Replicator 2.0

- What is it?

http://code.google.com/p/tungsten-replicator

tungsten-replicator

A high performance, open source, data replication engine for MySQL

Search projects

Project Home | Downloads | Wiki | Issues | Source | Administer

Summary | Updates | People

Project Information

Star project
Activity High
Project feeds

Code license
GNU GPL v2

Labels
Database, MySQL, Cluster, PostgreSQL, Performance, Persistence, Java, Replication, HighAvailability, Failover

Members
g.maxia, gilles.rayrat, robert.hodges@continuent.com
2 committers

Your role
Owner

Featured

Downloads
tungsten-replicator-2.0.2-28.tar.gz
tungsten_deployer_1.12.tar.gz
Show all »

Wiki pages
Cheat_Sheet
FAQ
HowToBuild
Release_Notes
TungstenReplicatorIntro
Show all »

Links

Blogs
The Data Charmer
The Scale Out blog
Continuent Tungsten Blog

Tungsten Replicator is a high performance, open source, data replication engine for MySQL. It offers a set of features that surpass any open source replicator available today: global transaction IDs to support failover, flexible transaction filtering, extensible transaction metadata, sharding, multiple replication services per process, high performance, and simple, well-documented operation.

Tungsten Replicator helps technically focused users solve problems like promoting masters easily from pools of slaves, replicating data between different database versions, replicating efficiently across sites, building complex topologies, and parallelizing data flow between servers. Tungsten Replicator runs equally well in cloud as well as locally hosted environments. Tungsten Replicator users range from tiny start-ups to the largest web properties on the planet.

Tungsten Replicator is sponsored by Continuent, Inc., which offers support as well as sponsored development of new replication features. Tungsten Replicator is also the base for Tungsten Enterprise, a commercial database clustering product that improves transaction throughput and keeps data highly available to applications. Please visit the Continuent website for more information about services as well as commercial offerings.

Open Source

100% GPL v2

What can it do?

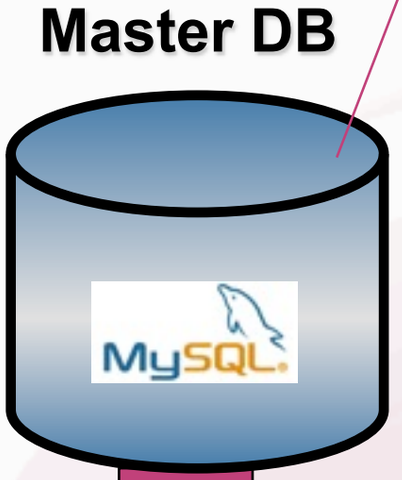
- Easy failover
- Multiple masters
- Multiple sources to a single slave
- Parallel replication
- Replicate to Oracle and PostgreSQL database

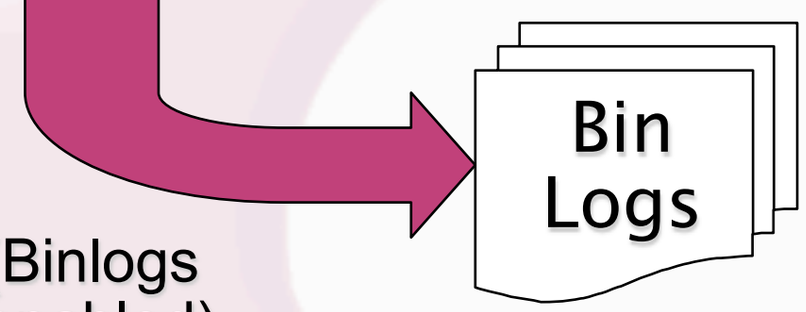
MySQL to foreign services

MySQL setup to run as MySQL master

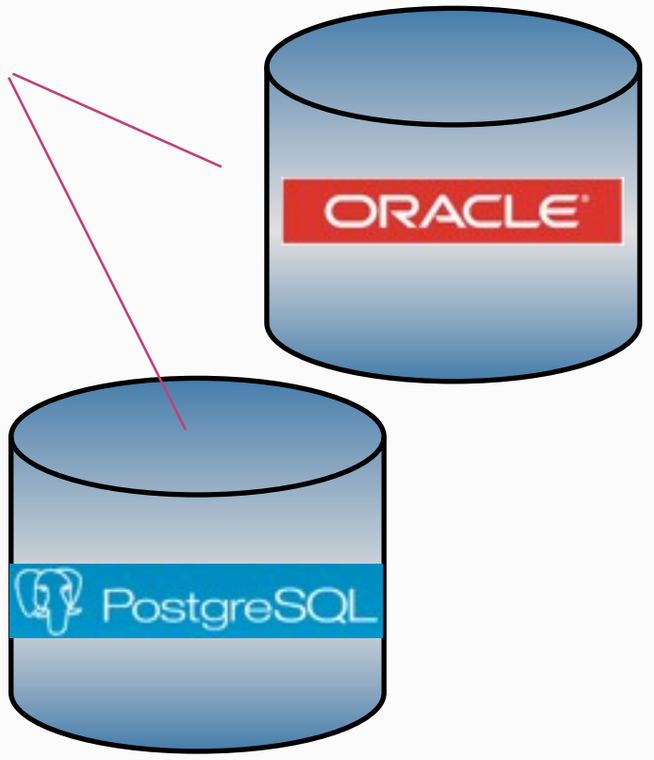
Data is applied to PostgreSQL or Oracle

Master DB





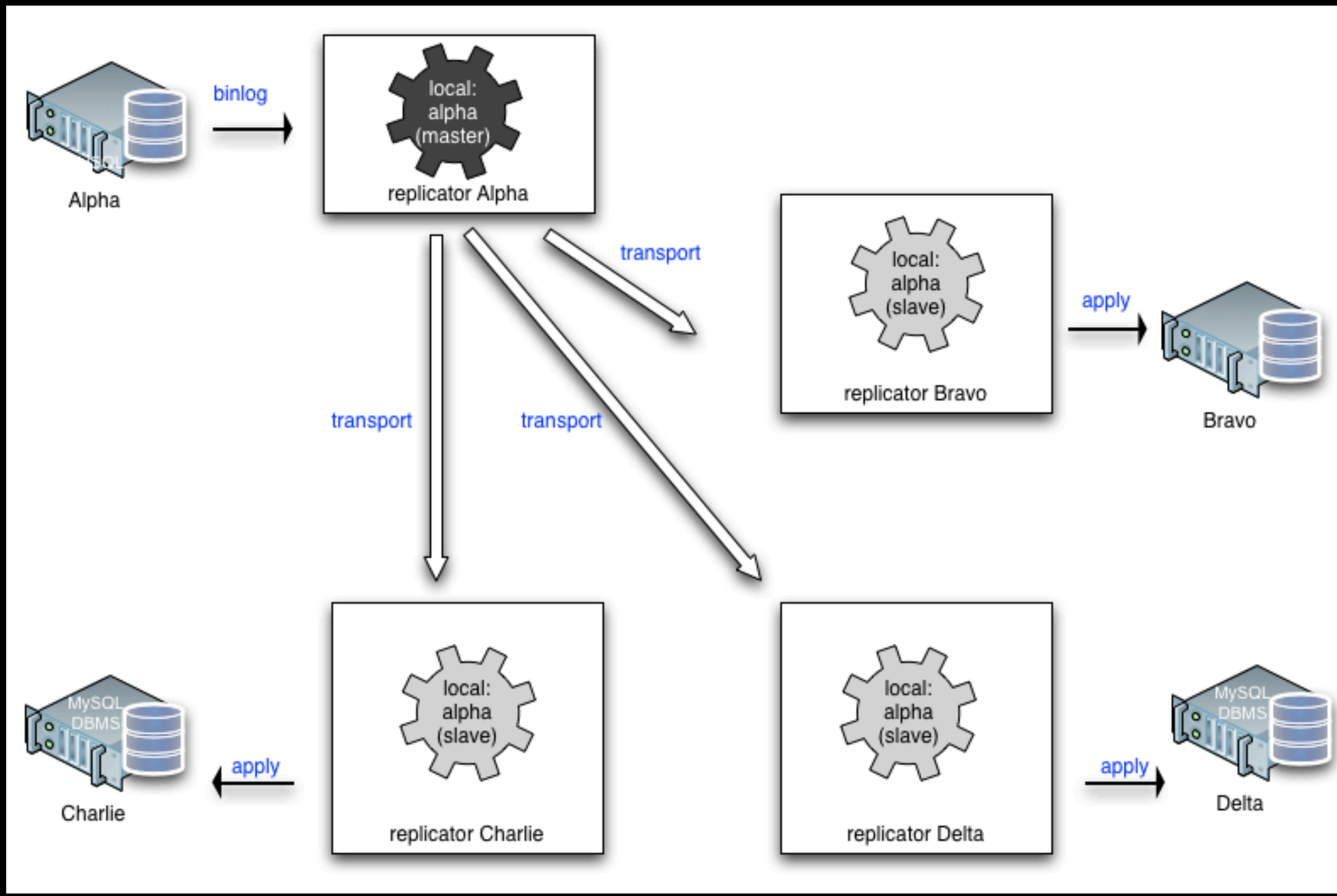
(Binlogs enabled)



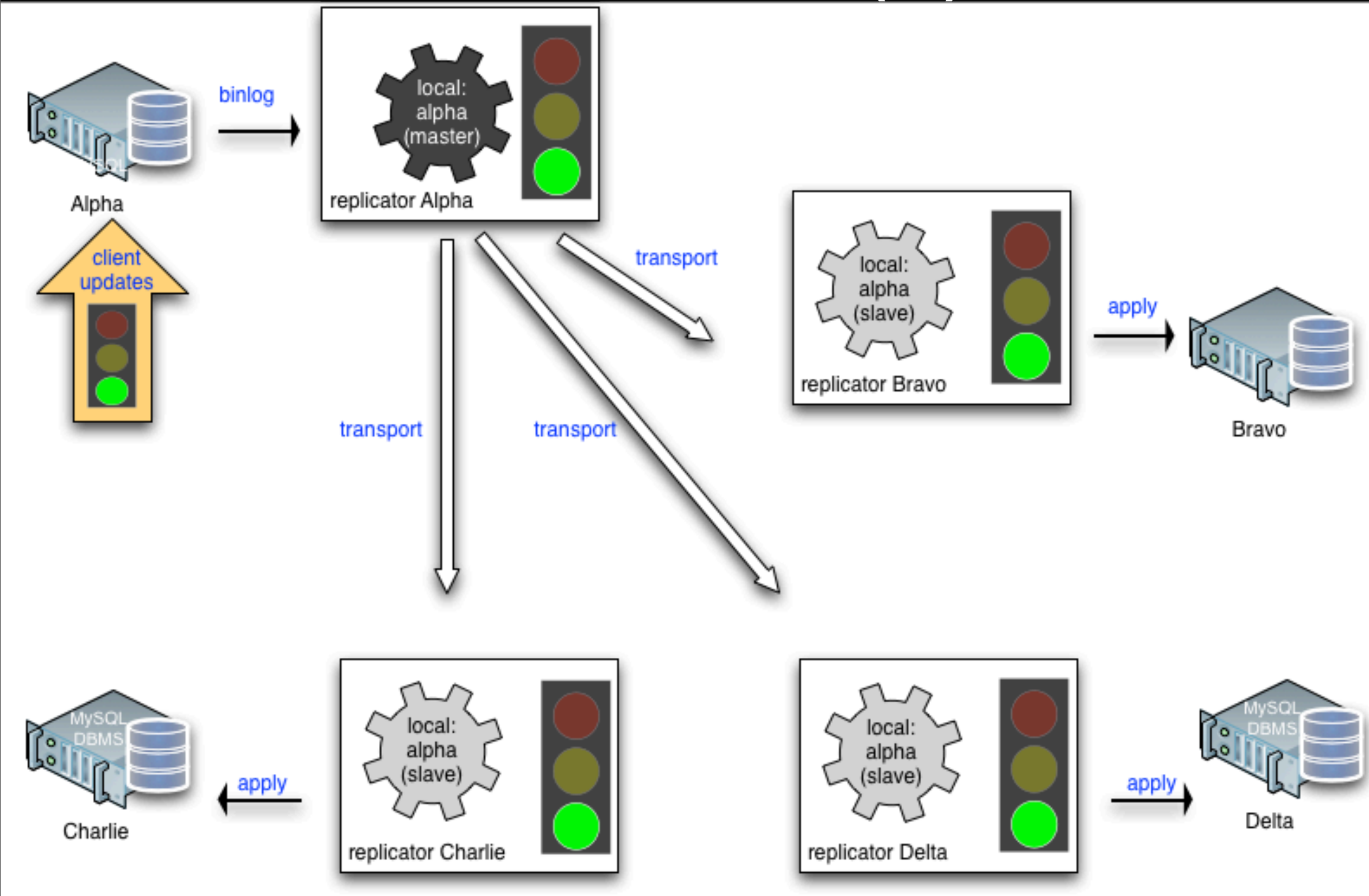
Slave DB

continuent
Open. Always Available.

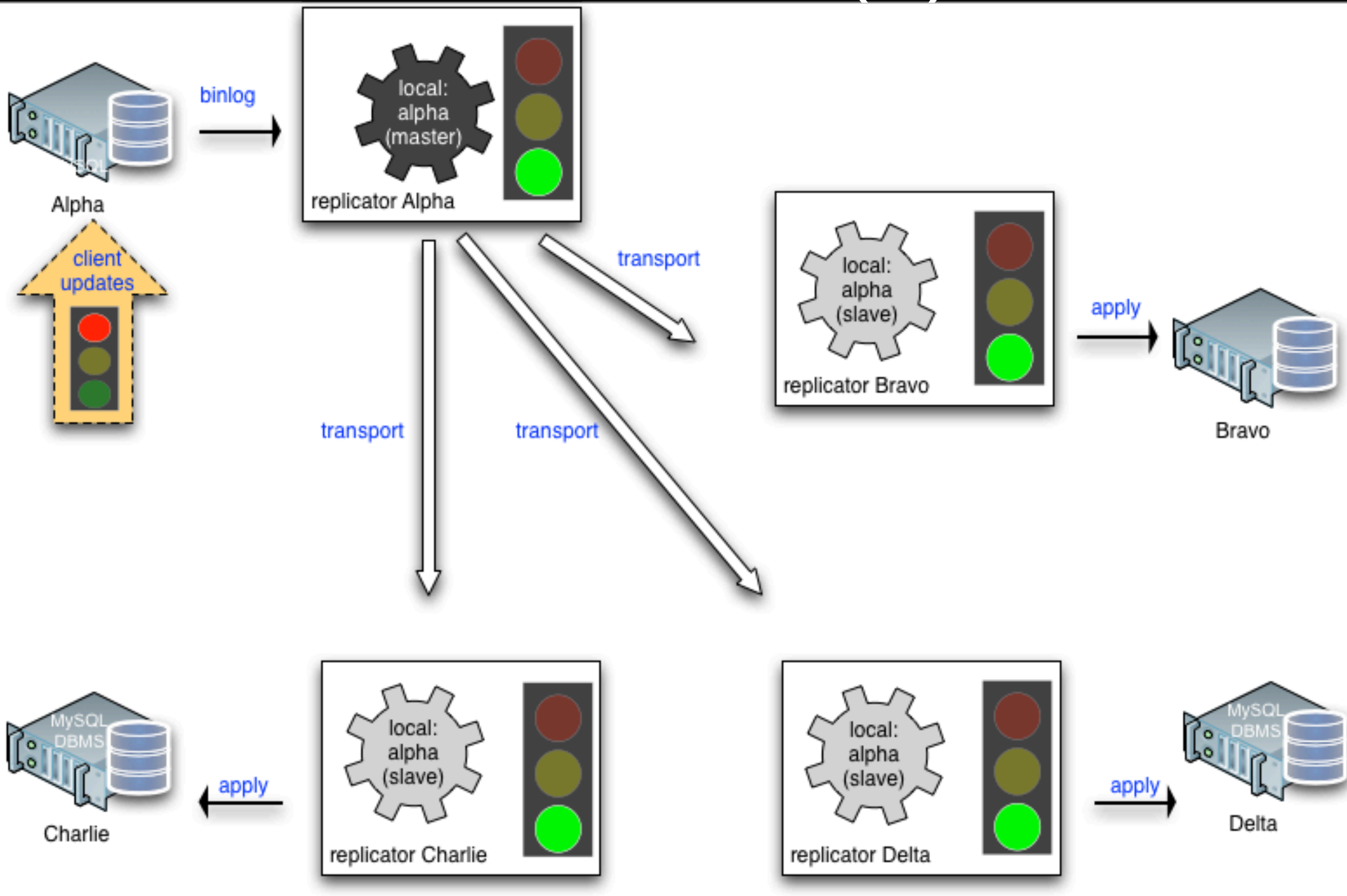
From the beginning ...



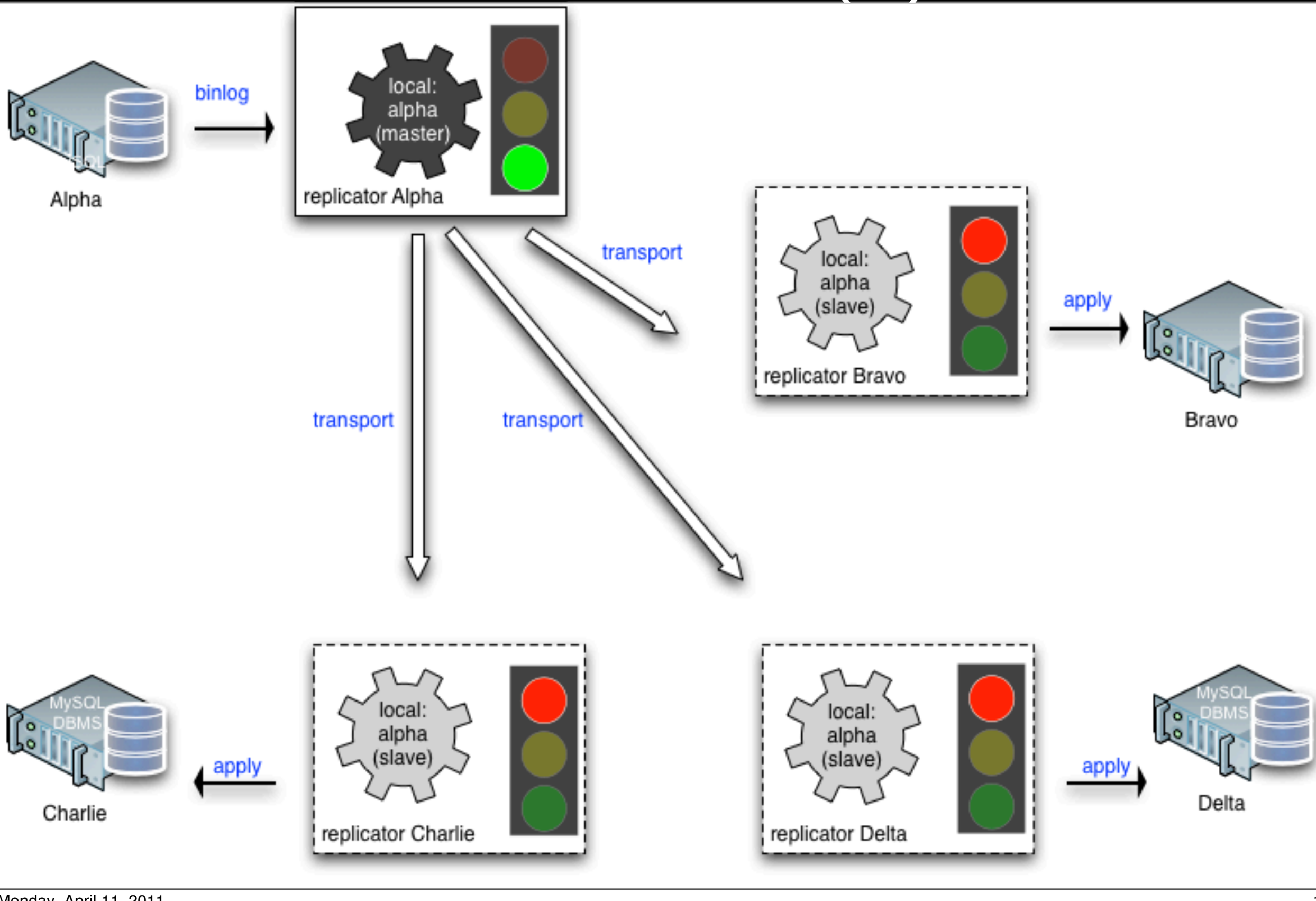
Fail-over (1)


binlog
Alpha
local:
alpha
(master)
replicator Alpha
client
updates
transport
transport
transport
local:
alpha
(slave)
replicator Bravo
apply
Bravo
MySQL
DBMS
Charlie
apply
local:
alpha
(slave)
replicator Charlie
local:
alpha
(slave)
replicator Delta
apply
MySQL
DBMS
Delta

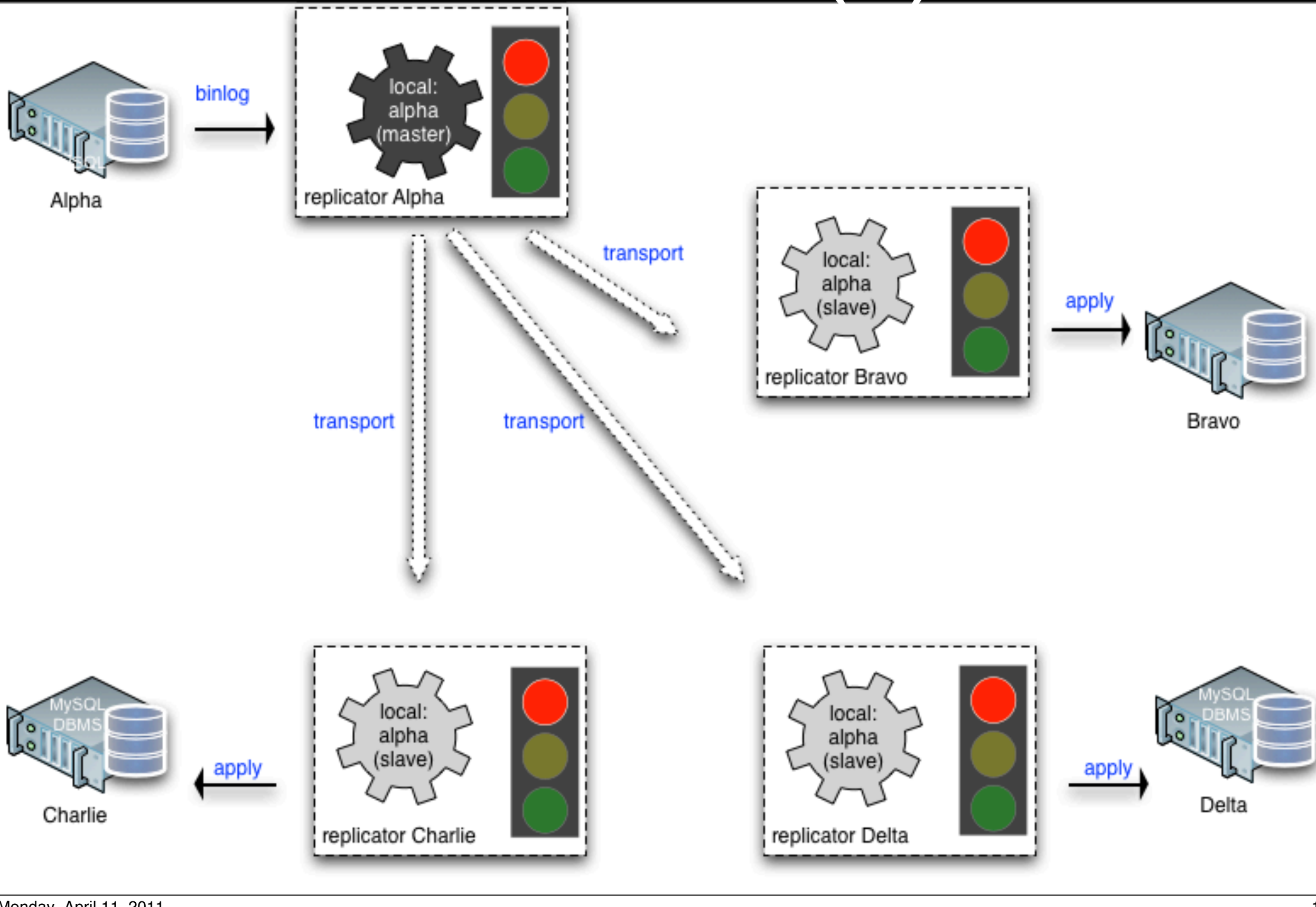
Fail-over (2)


binlog
Alpha
client updates
local: alpha (master)
replicator Alpha
transport
transport
transport
local: alpha (slave)
replicator Bravo
apply
Bravo
MySQL DBMS
Charlie
apply
local: alpha (slave)
replicator Charlie
local: alpha (slave)
replicator Delta
apply
MySQL DBMS
Delta

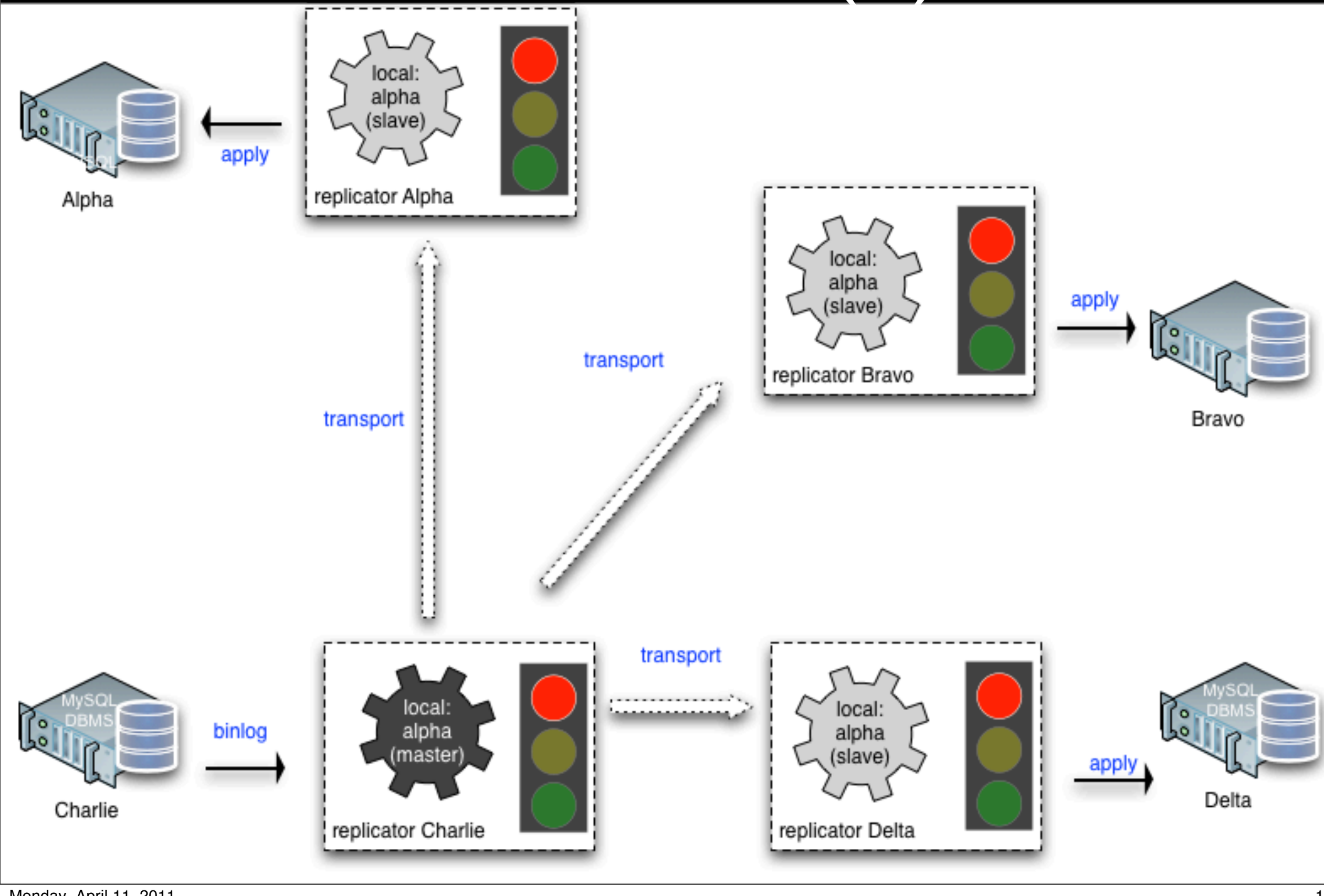
Fail-over (3)


Alpha
binlog
local:
alpha
(master)
replicator Alpha
transport
transport
transport
local:
alpha
(slave)
replicator Bravo
apply
Bravo
MySQL
DBMS
Charlie
apply
local:
alpha
(slave)
replicator Charlie
local:
alpha
(slave)
replicator Delta
apply
MySQL
DBMS
Delta

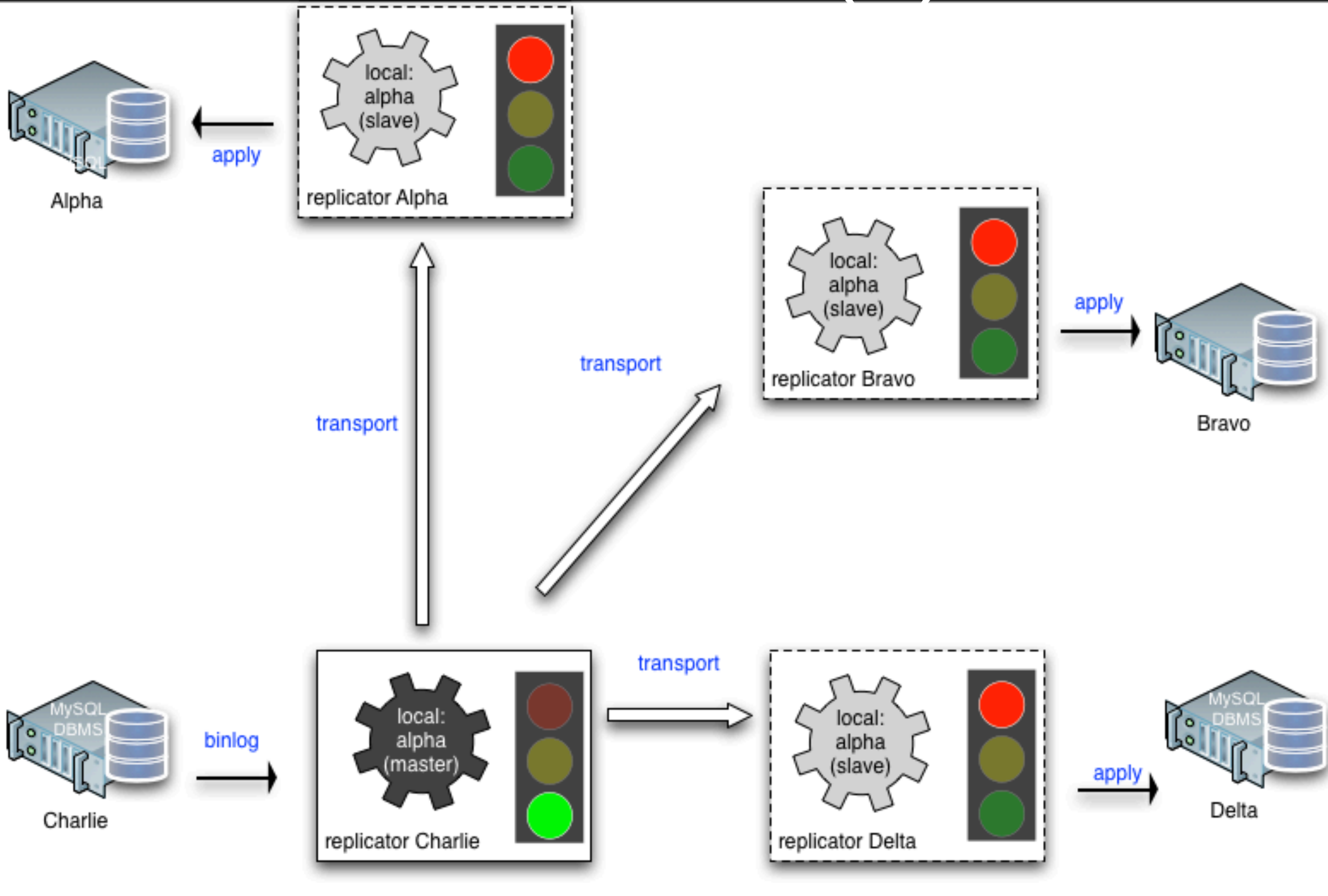
Fail-over (4)


Alpha
binlog
local:
alpha
(master)
replicator Alpha
transport
transport
transport
local:
alpha
(slave)
replicator Bravo
apply
Bravo
MySQL
DBMS
Charlie
apply
local:
alpha
(slave)
replicator Charlie
local:
alpha
(slave)
replicator Delta
apply
MySQL
DBMS
Delta

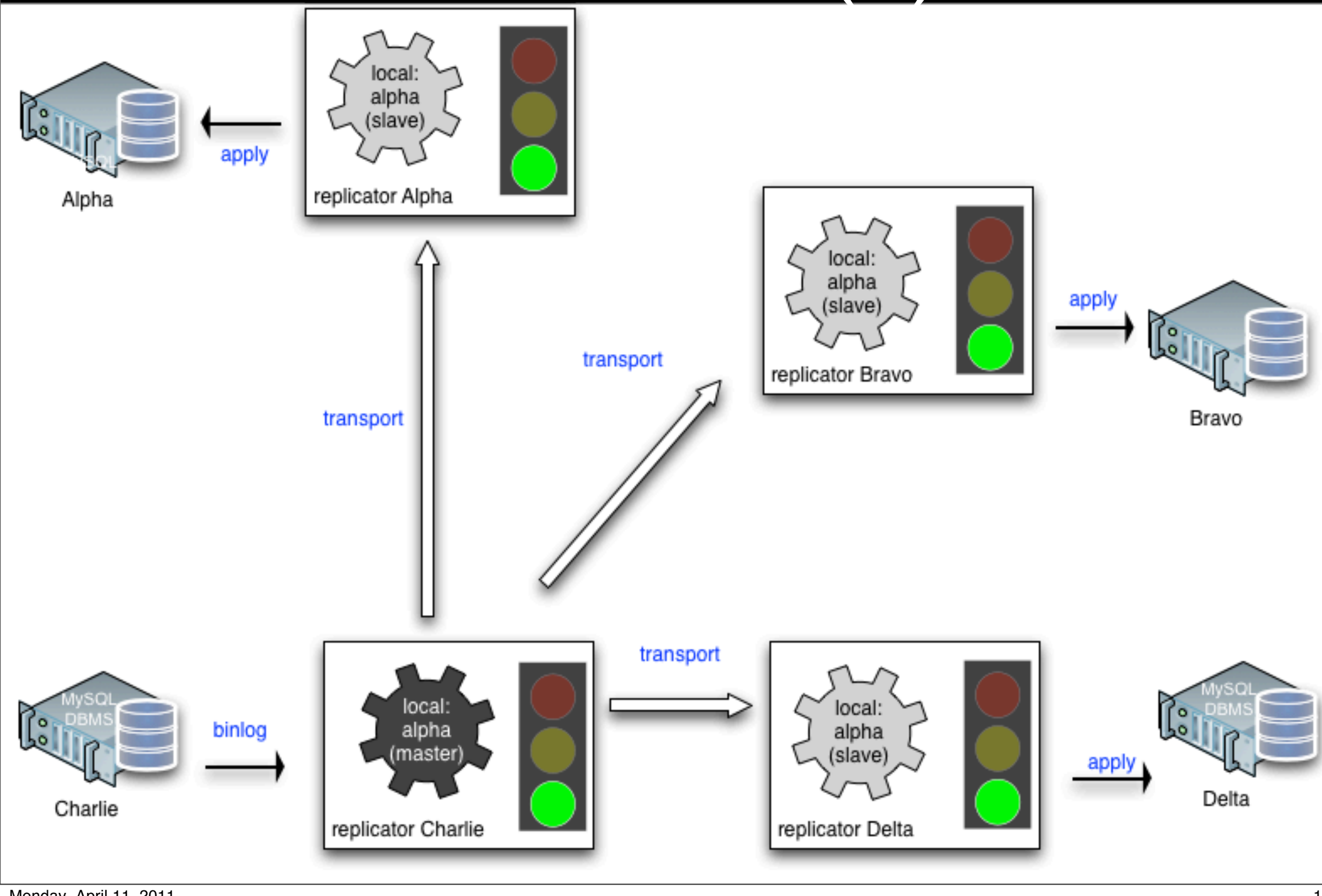
Fail-over (5)


Alpha
apply
local:
alpha
(slave)
replicator Alpha
transport
transport
local:
alpha
(slave)
replicator Bravo
apply
Bravo
MySQL
DBMS
Charlie
binlog
local:
alpha
(master)
replicator Charlie
transport
local:
alpha
(slave)
replicator Delta
apply
MySQL
DBMS
Delta

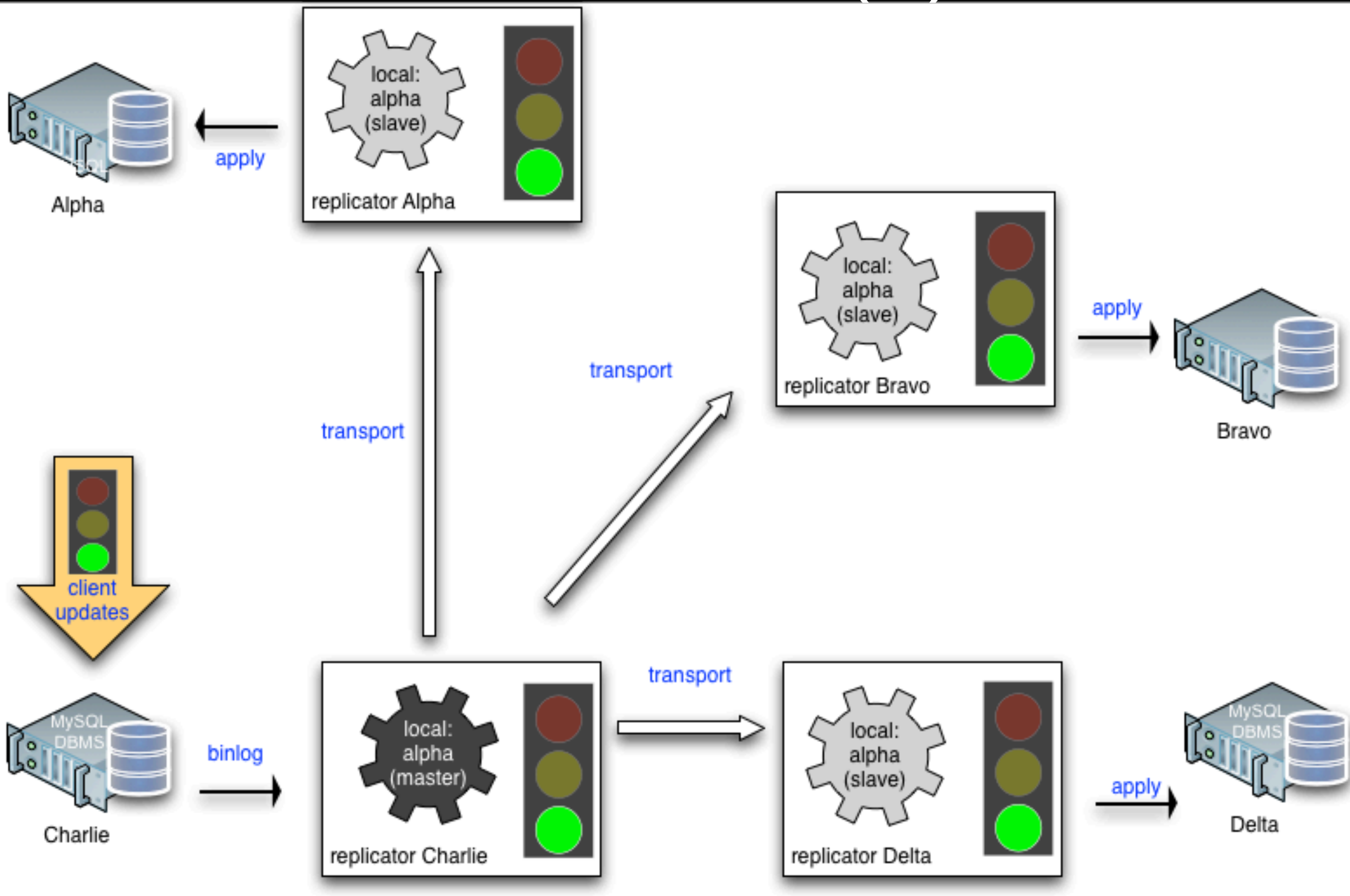
Fail-over (6)


Alpha
apply
local:
alpha
(slave)
replicator Alpha
transport
transport
local:
alpha
(slave)
replicator Bravo
apply
Bravo
MySQL
DBMS
Charlie
binlog
local:
alpha
(master)
replicator Charlie
transport
local:
alpha
(slave)
replicator Delta
apply
MySQL
DBMS
Delta

Fail-over (7)


local:
alpha
(slave)
replicator Alpha
apply
Alpha
transport
transport
local:
alpha
(slave)
replicator Bravo
apply
Bravo
MySQL
DBMS
binlog
Charlie
local:
alpha
(master)
replicator Charlie
transport
local:
alpha
(slave)
replicator Delta
apply
MySQL
DBMS
Delta

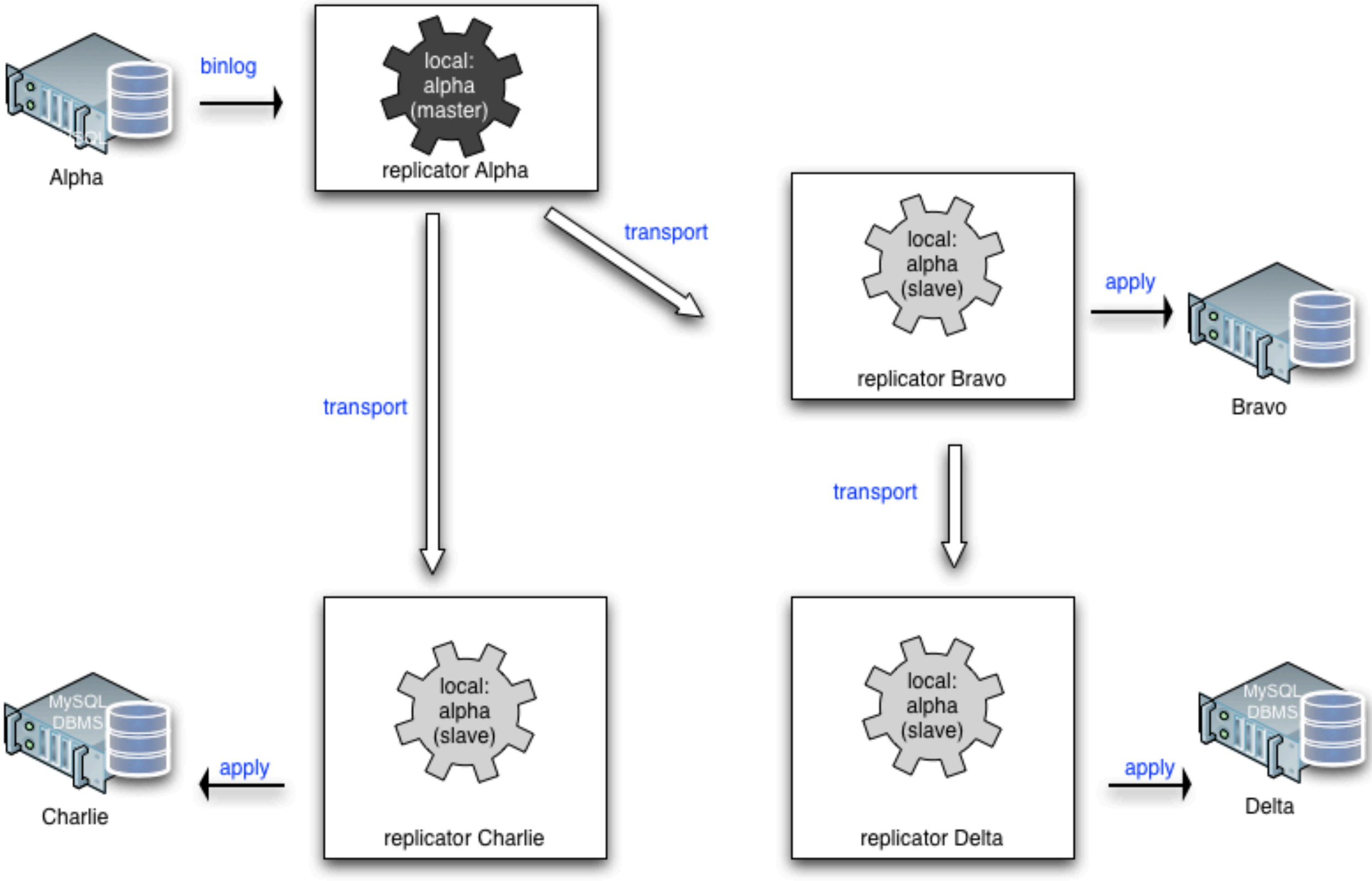
Fail-over (8)


Alpha
apply
local:
alpha
(slave)
replicator Alpha
transport
transport
local:
alpha
(slave)
replicator Bravo
apply
Bravo
client
updates
MySQL
DBMS
Charlie
binlog
local:
alpha
(master)
replicator Charlie
transport
local:
alpha
(slave)
replicator Delta
apply
MySQL
DBMS
Delta

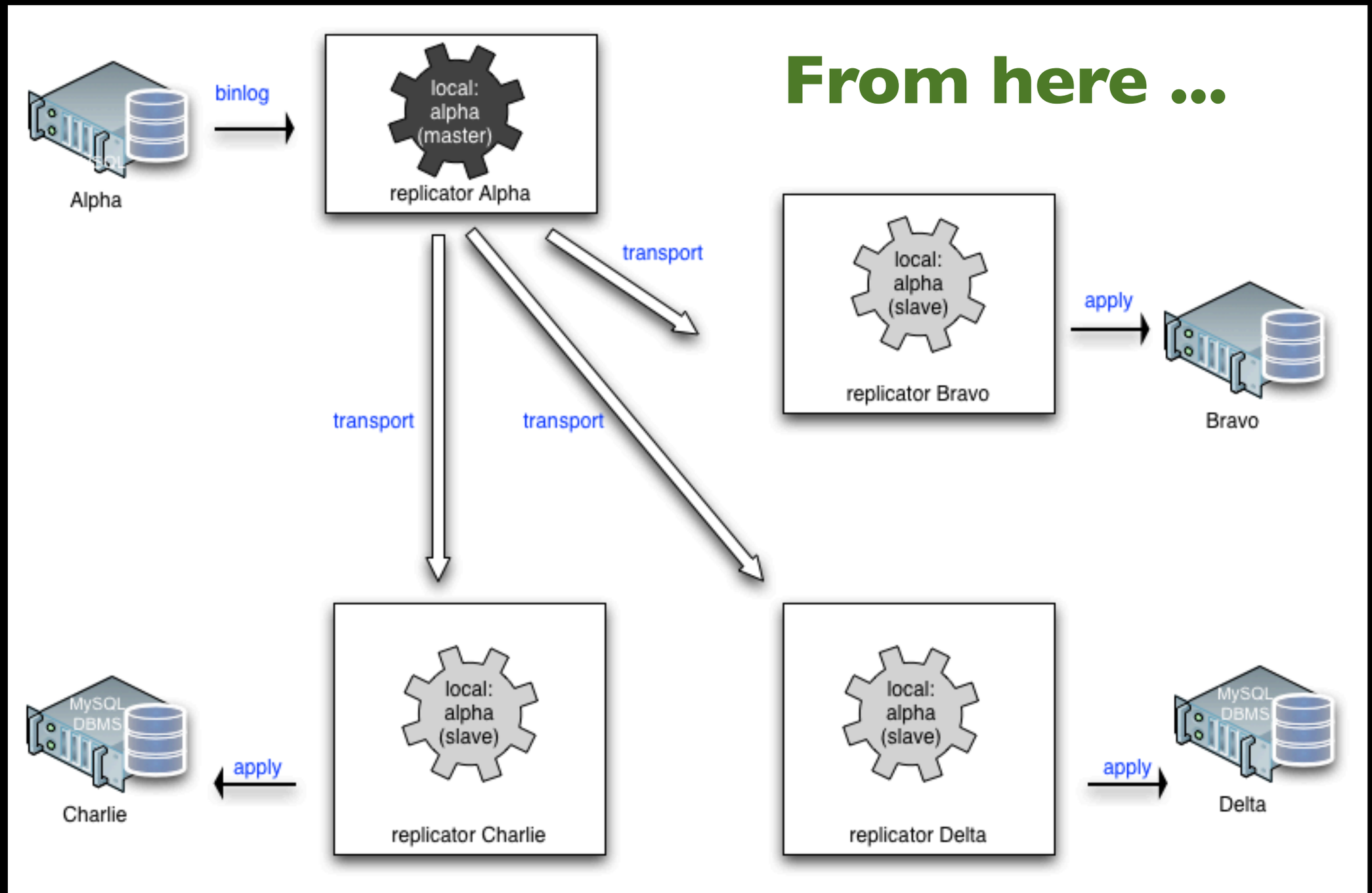
Failover

- DEMO

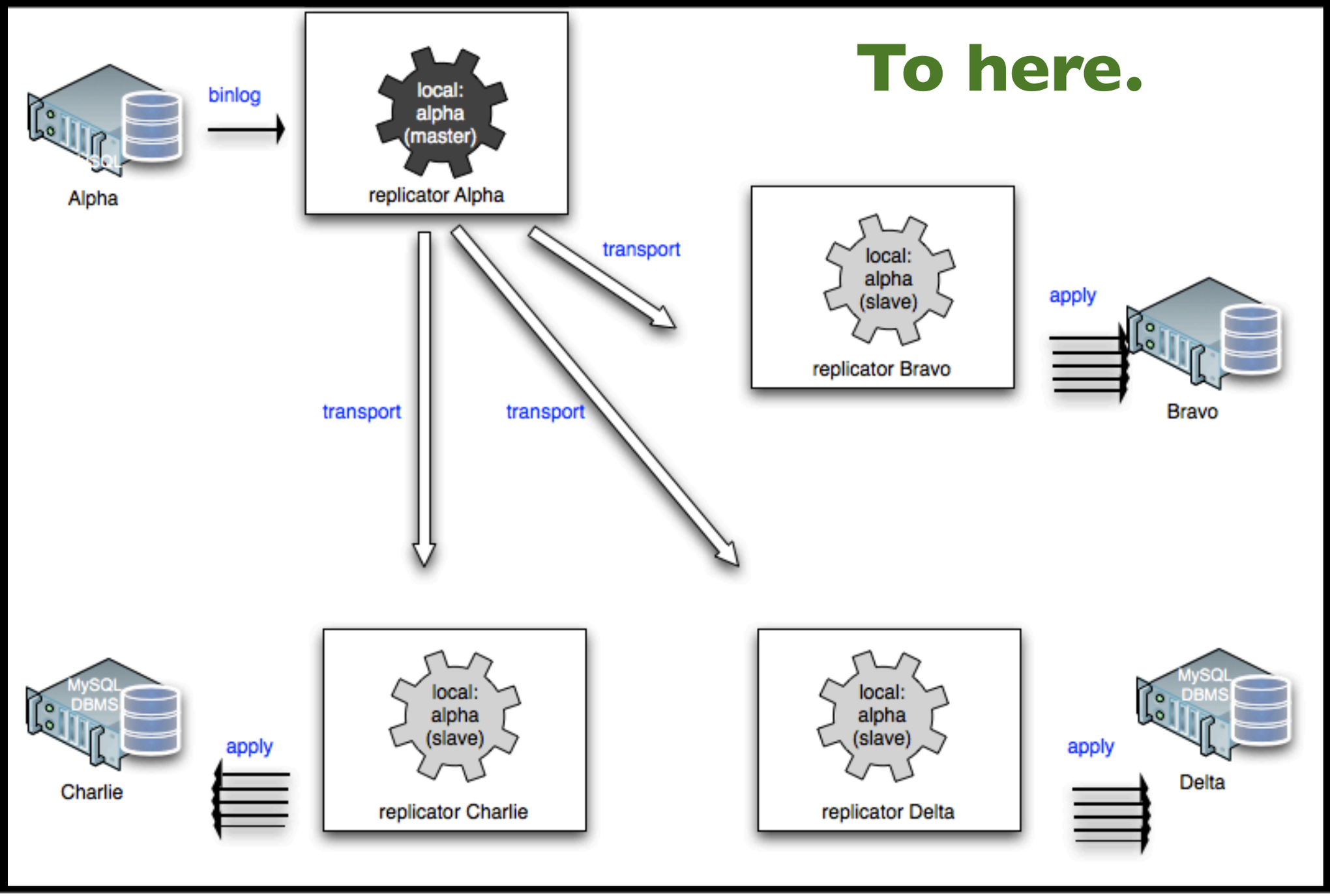
master/slave with an attitude



The steel foundry dream or parallel replication



The steel foundry dream or parallel replication

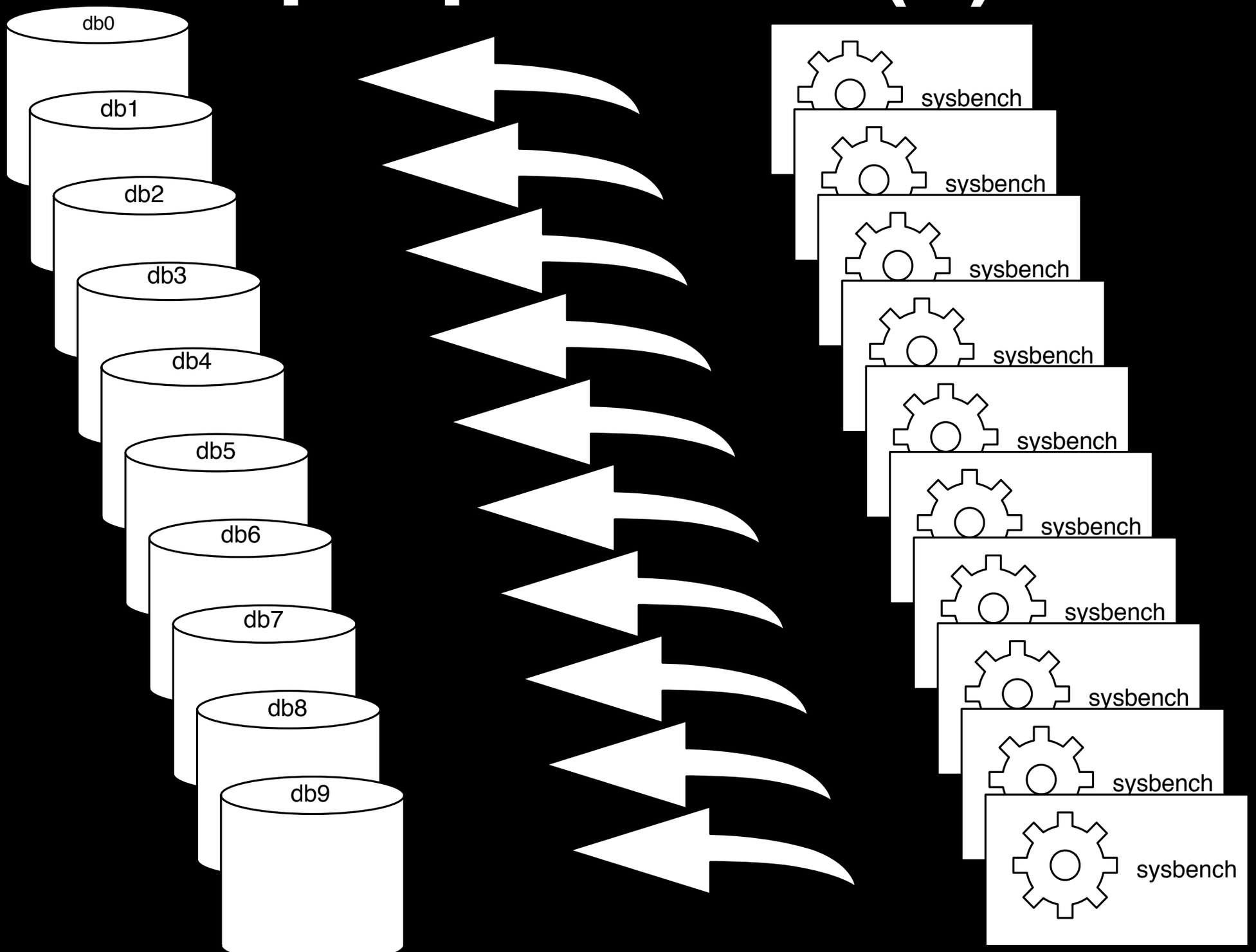


Parallel replication facts

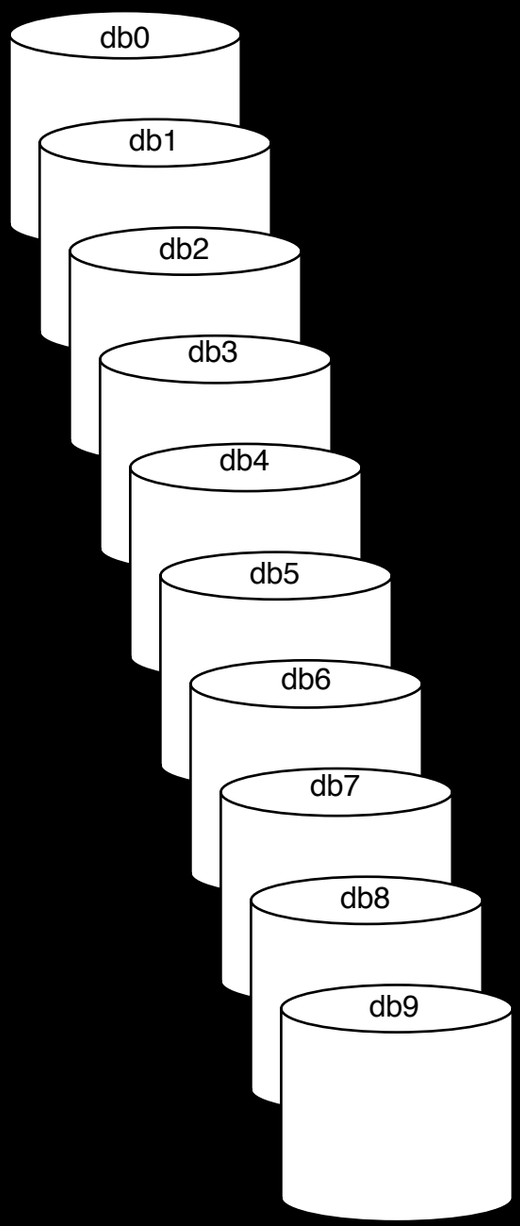
- Sharded by database
- Good choice for slave lag problems
- Bad choice for single database projects

Testing parallel replication

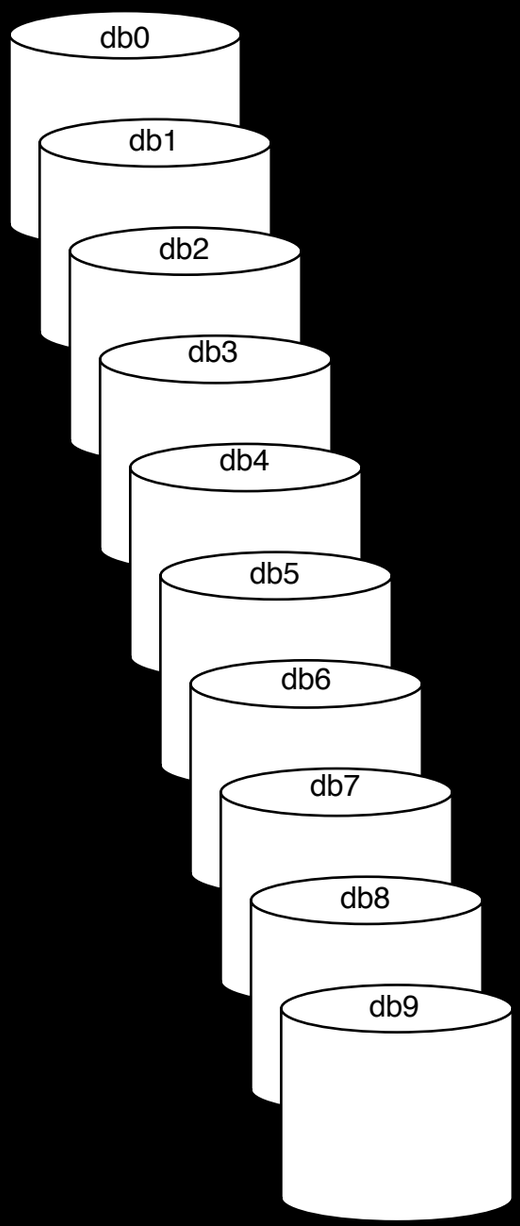
preparation (1)

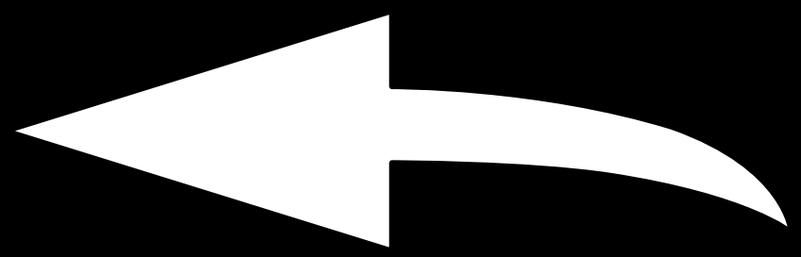


preparation (2)

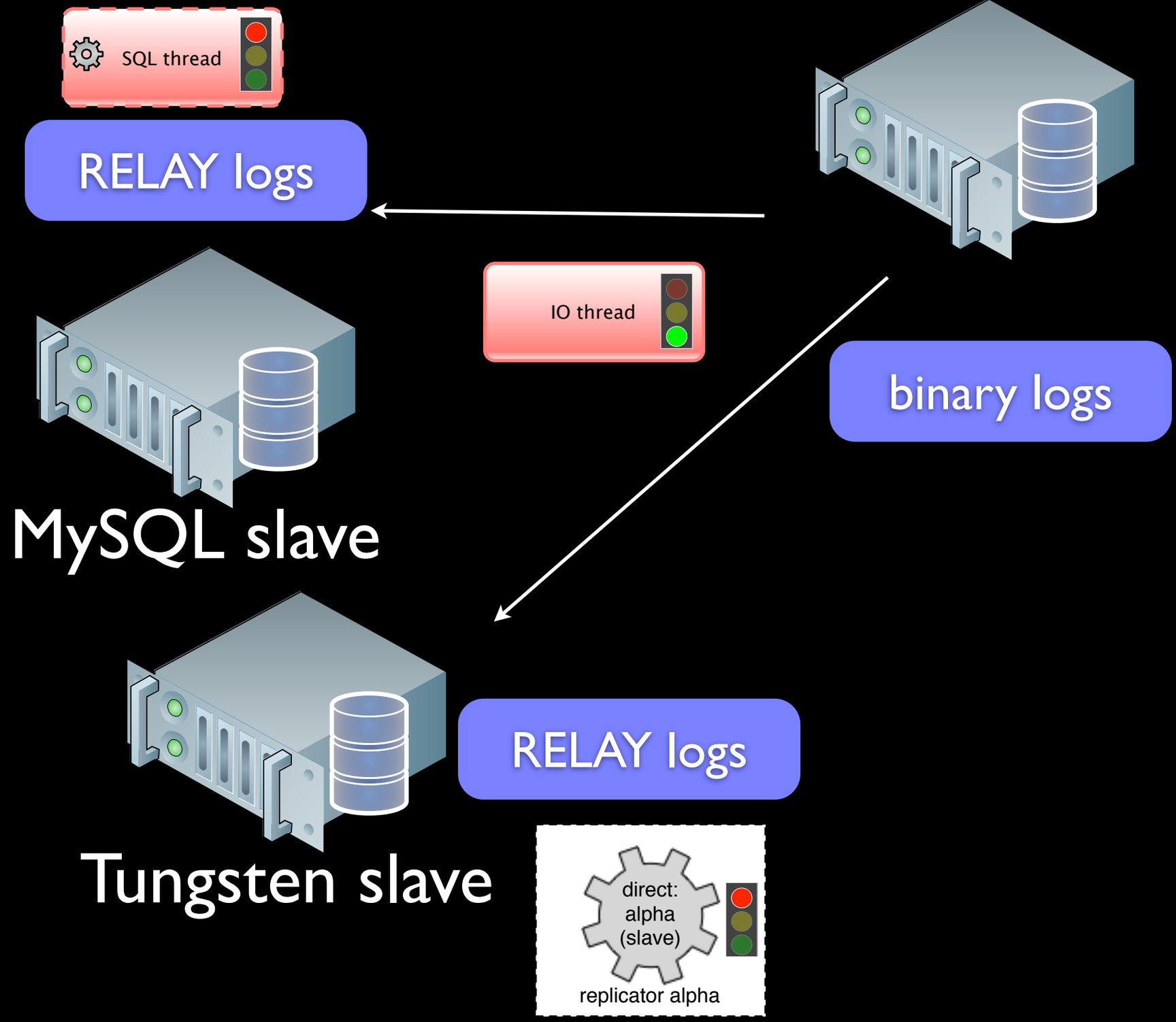


before the test (1)

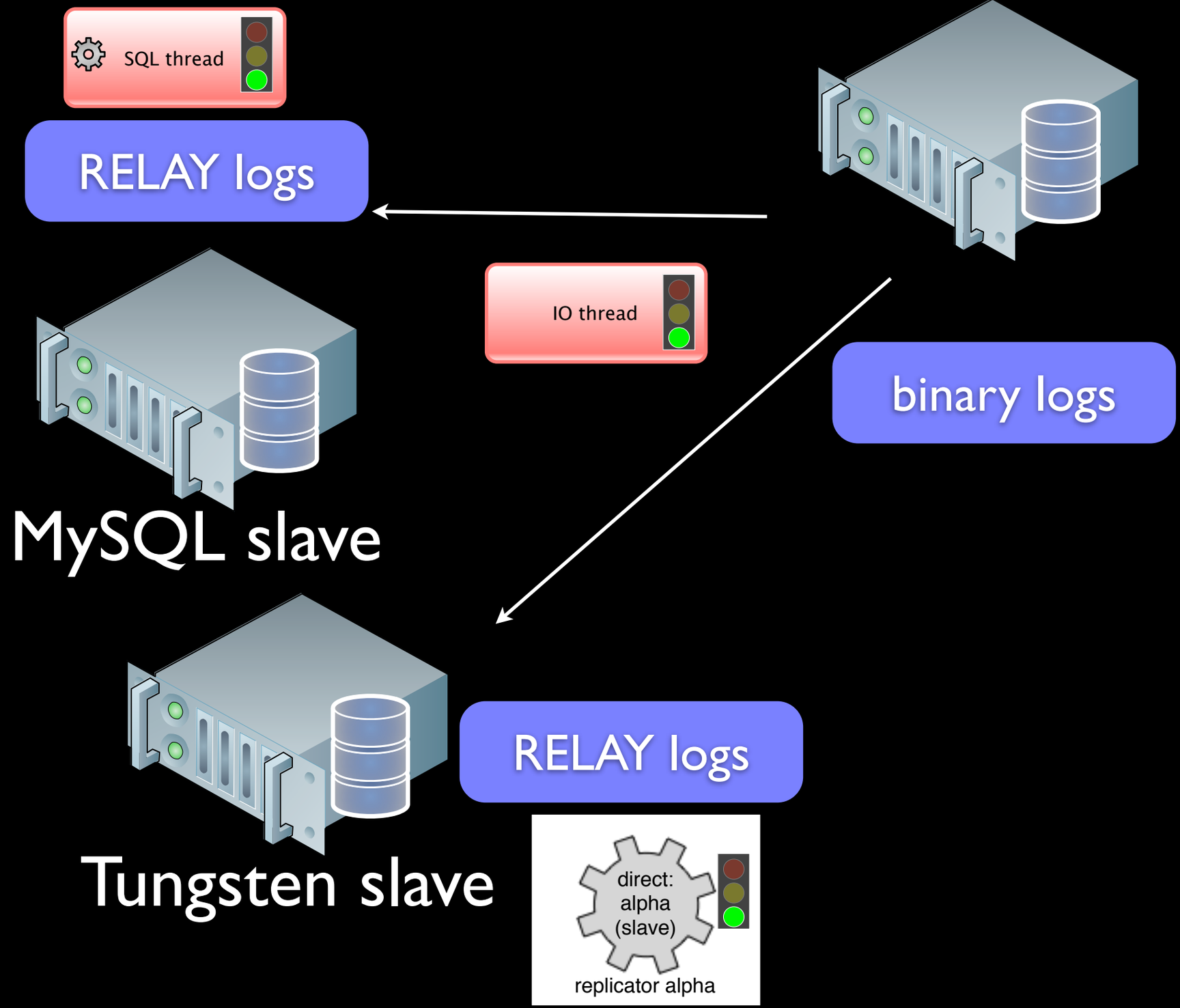




before the test (2)



starting the test



MySQL native replication

slave catch up in **00:02:30**

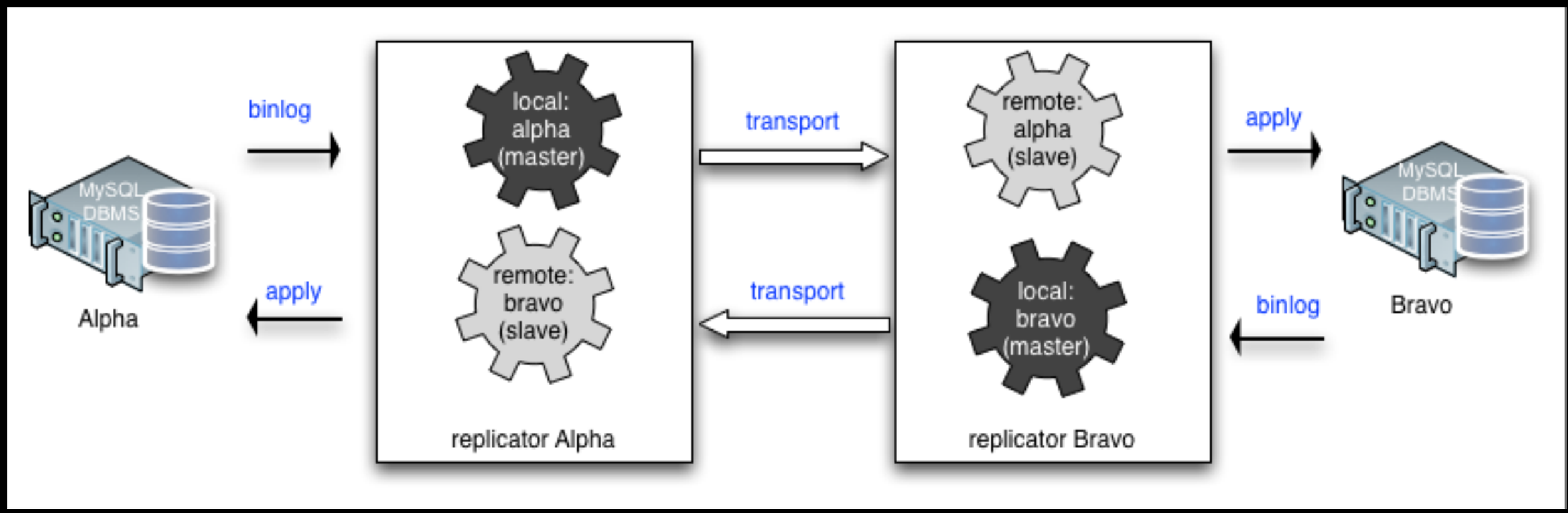
Tungsten parallel replication

slave catch up in **00:01:20**

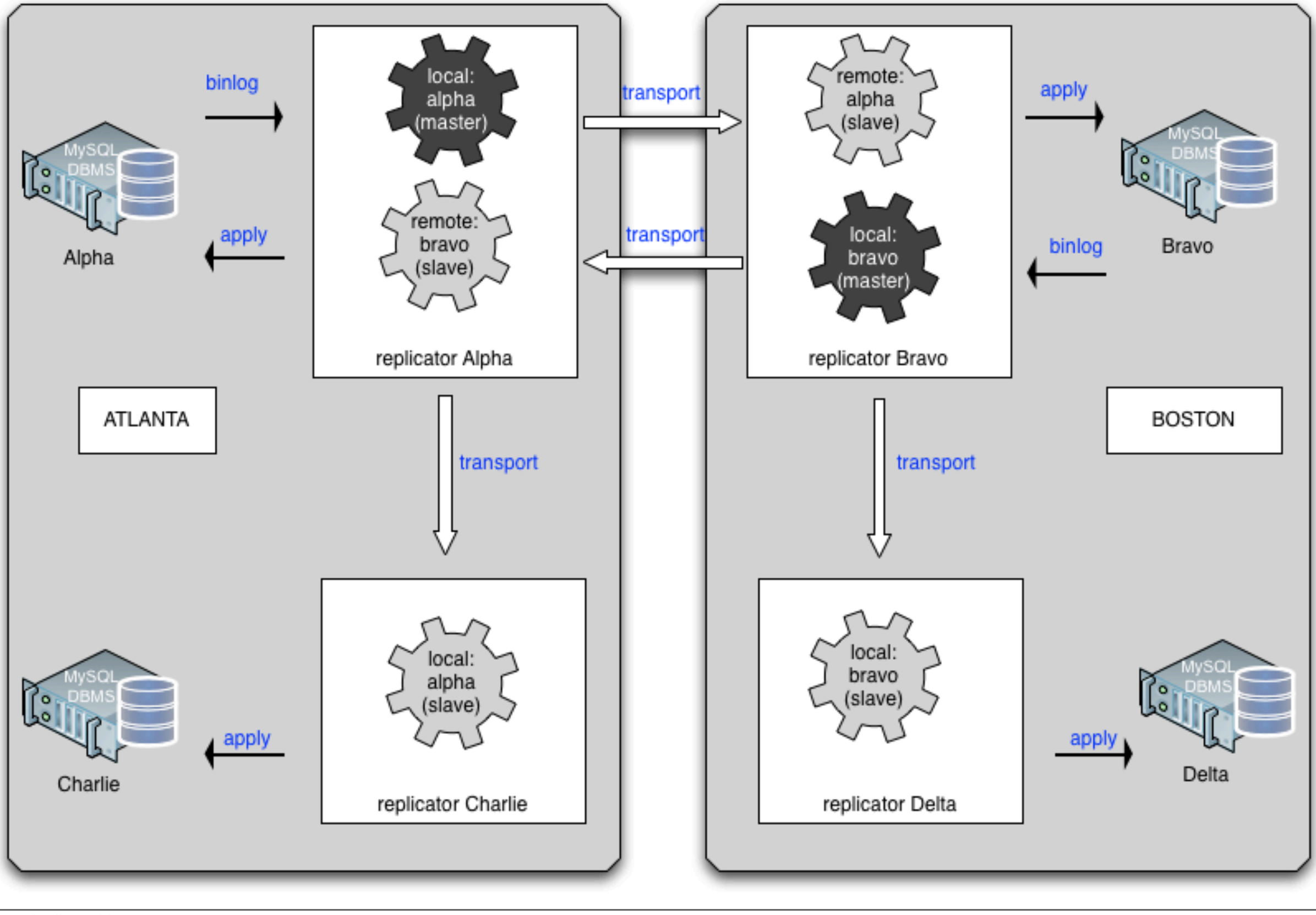
The widget seller dream, or multi masters

- Tungsten Replicator recipe: use more services

Bi-directional replication



Bi-directional replication with slaves



True multiple master

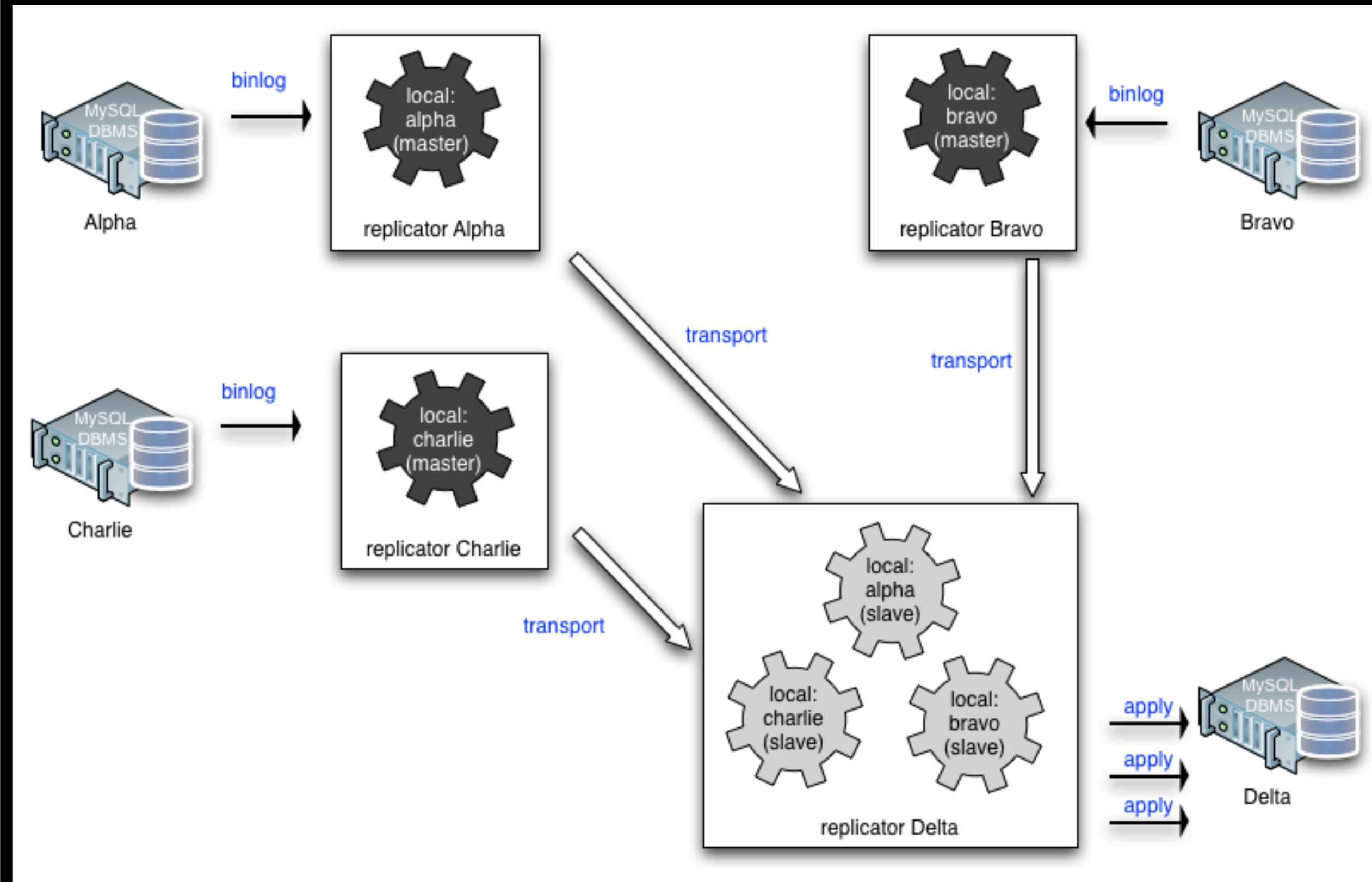
We'll see that in a moment.

But first

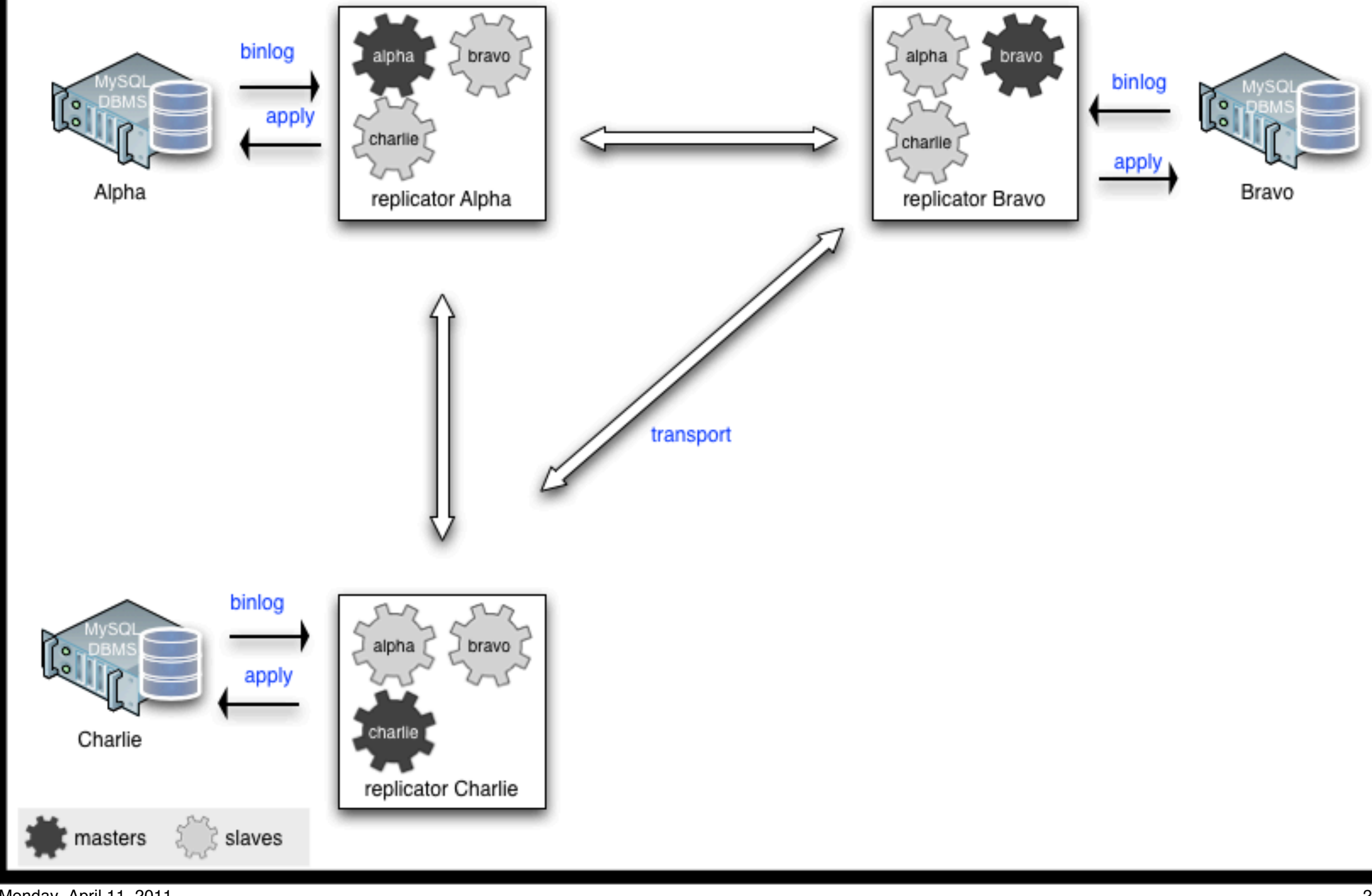
The shoe maker dream, or multiple sources

- Tungsten Replicator recipe is still valid: use more services

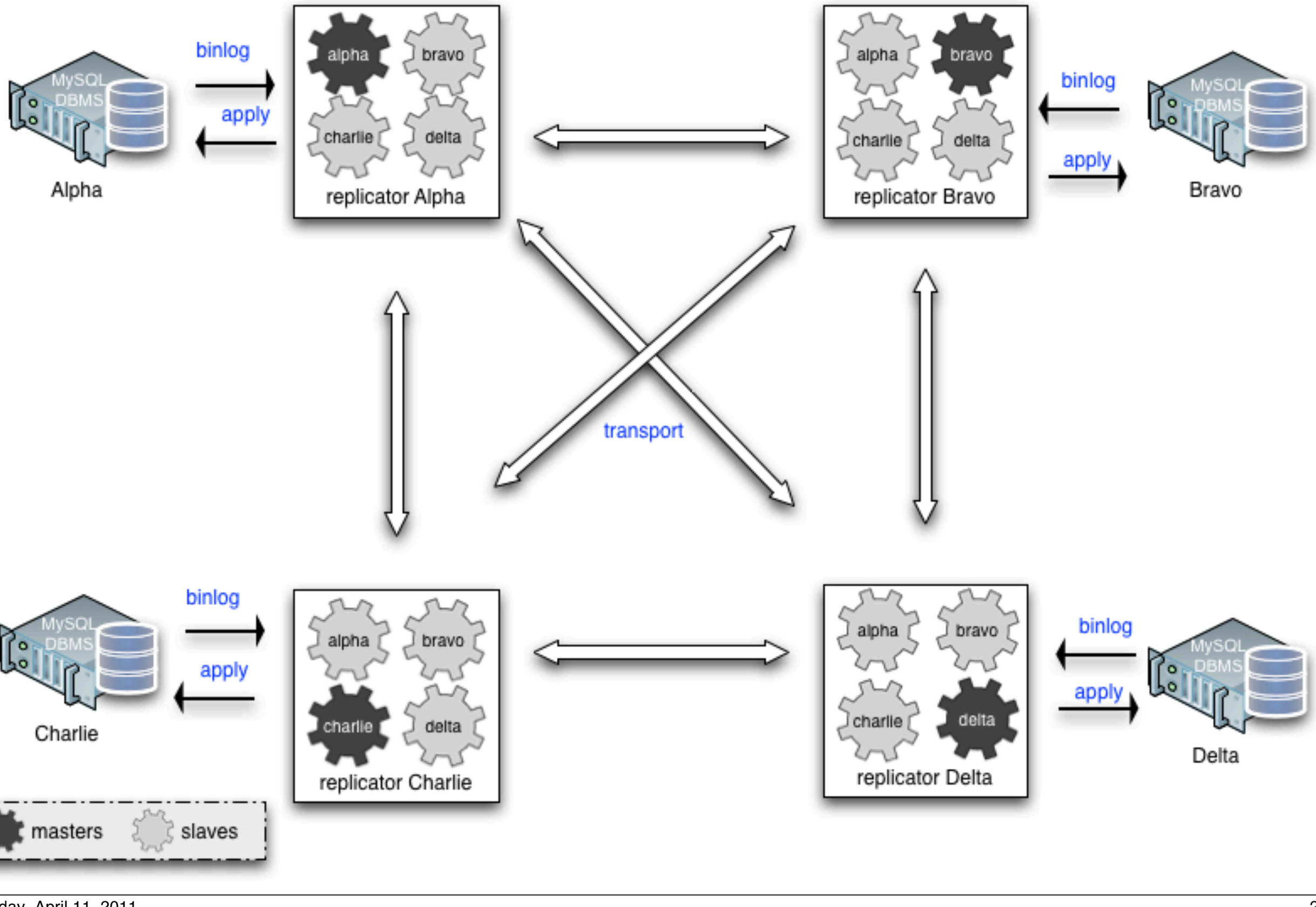
Multiple source replication



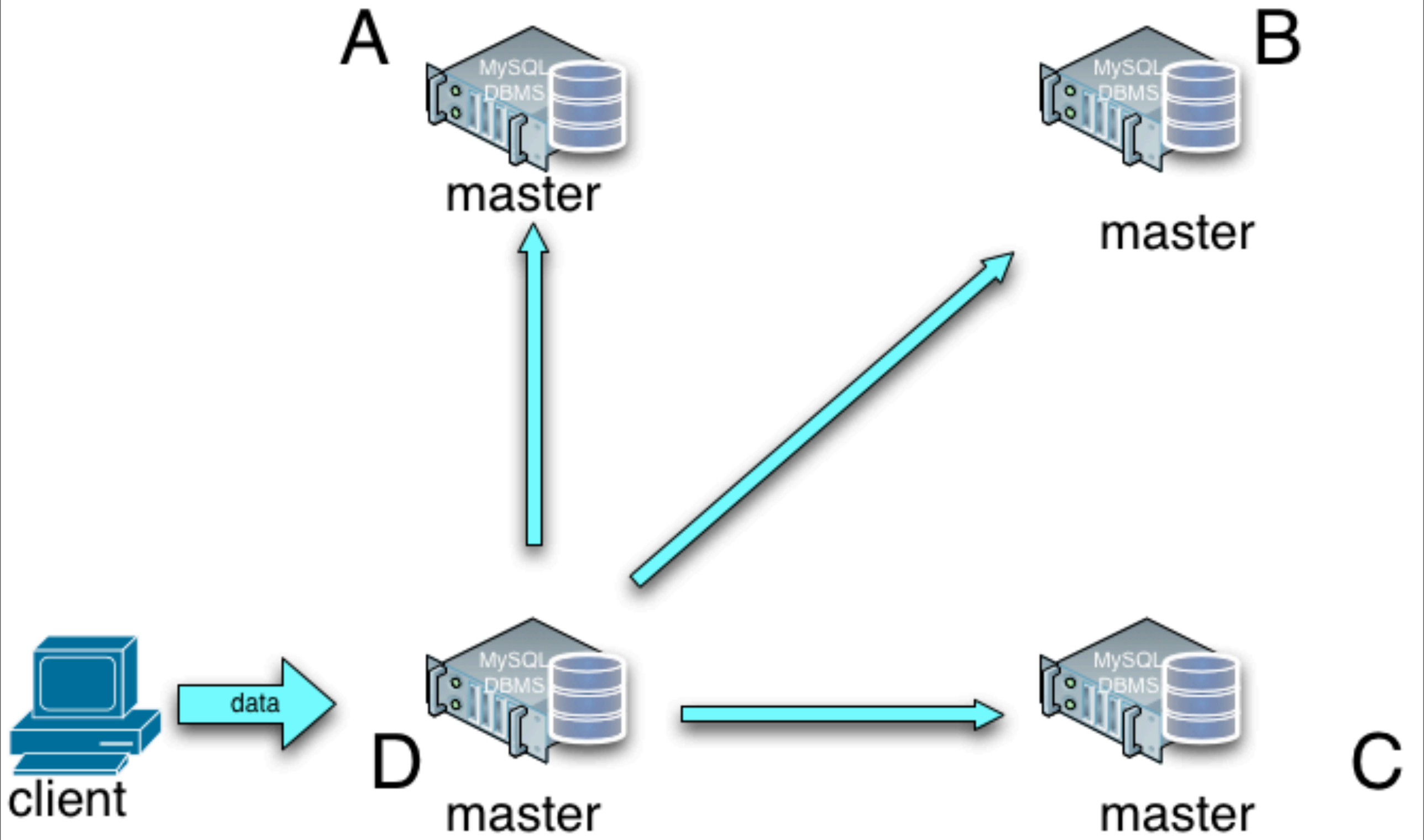
Multiple masters replication: 3 nodes



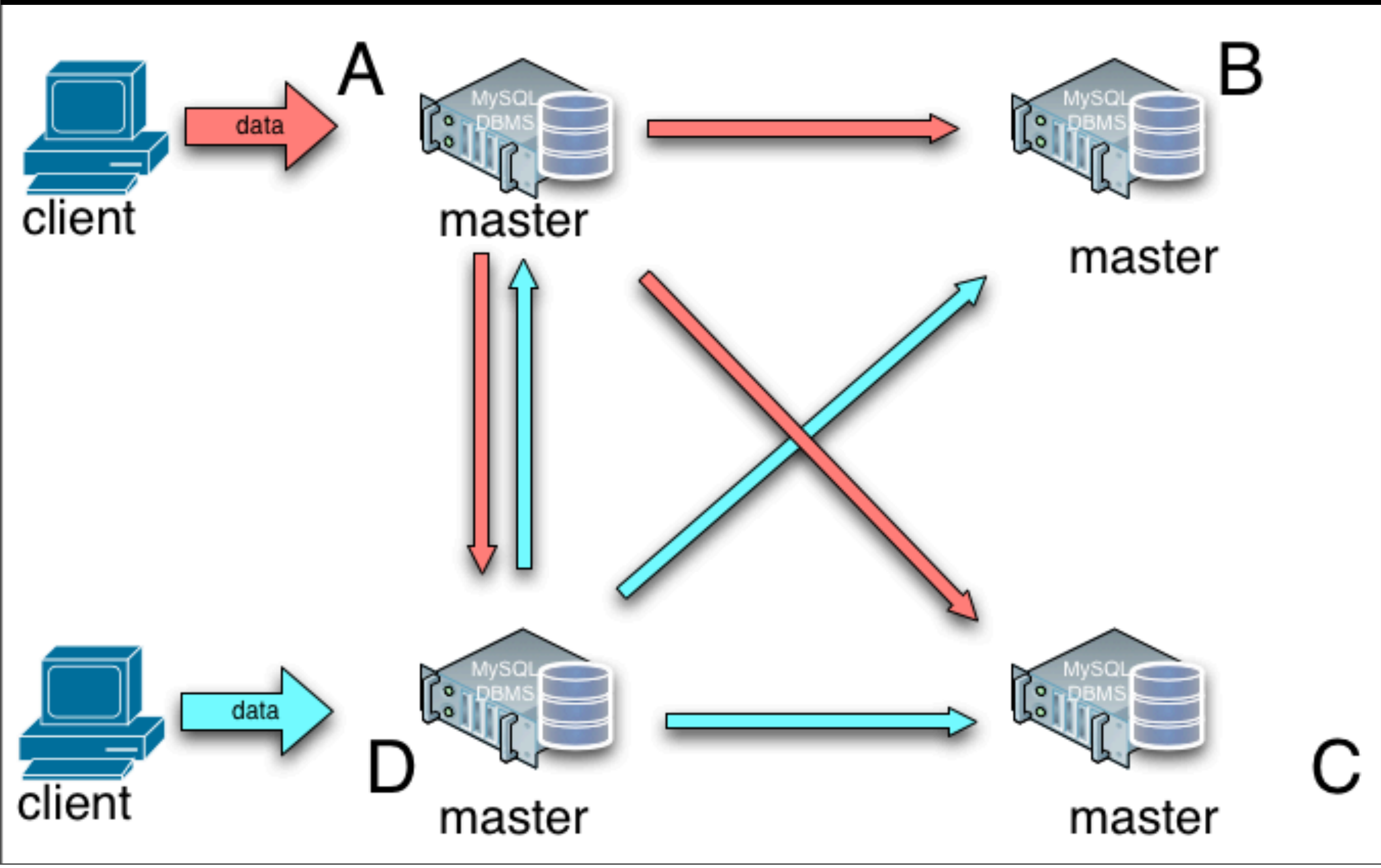
Multiple masters replication: 4 nodes



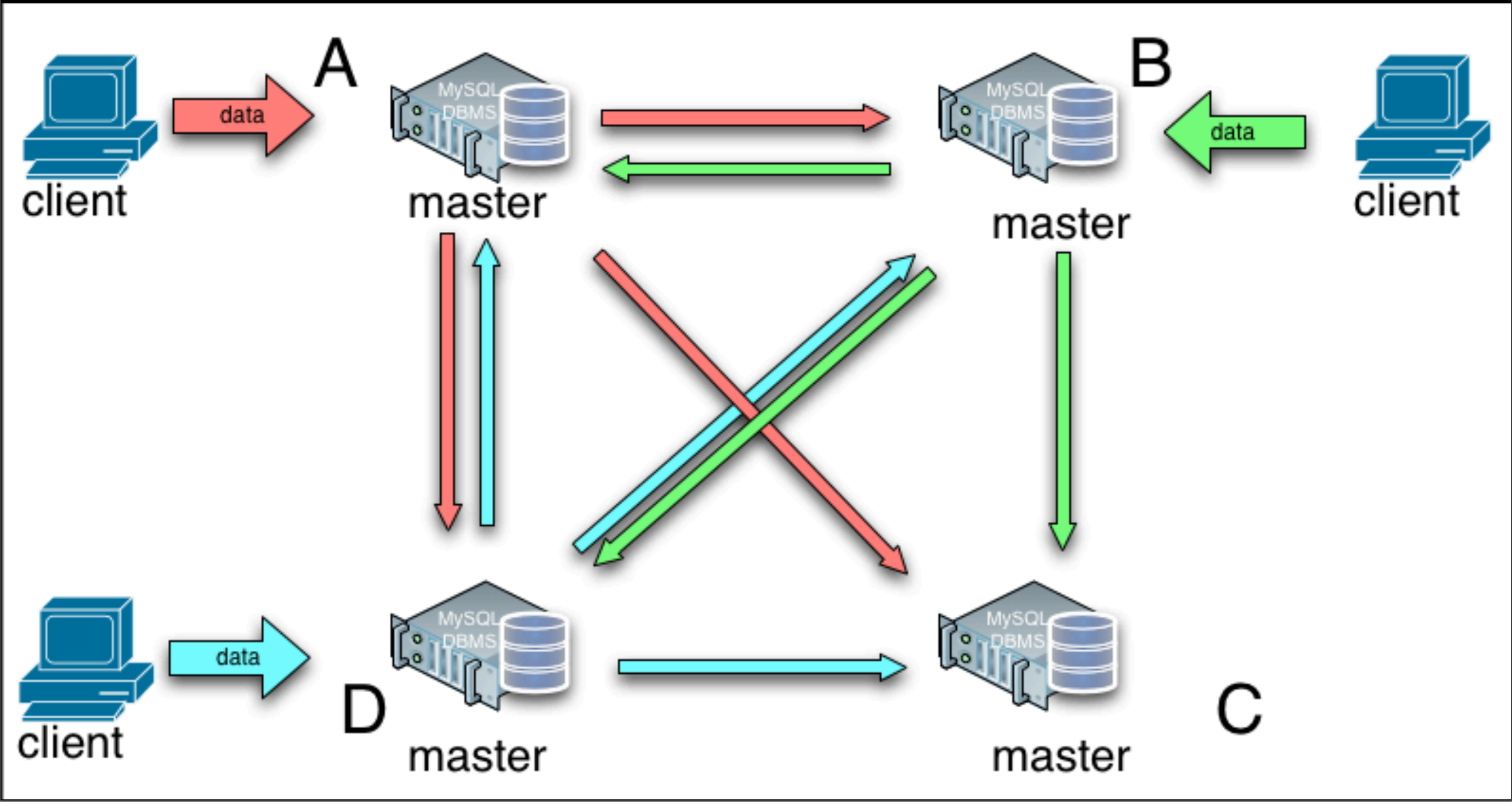
Updating 4 masters : 1 flow



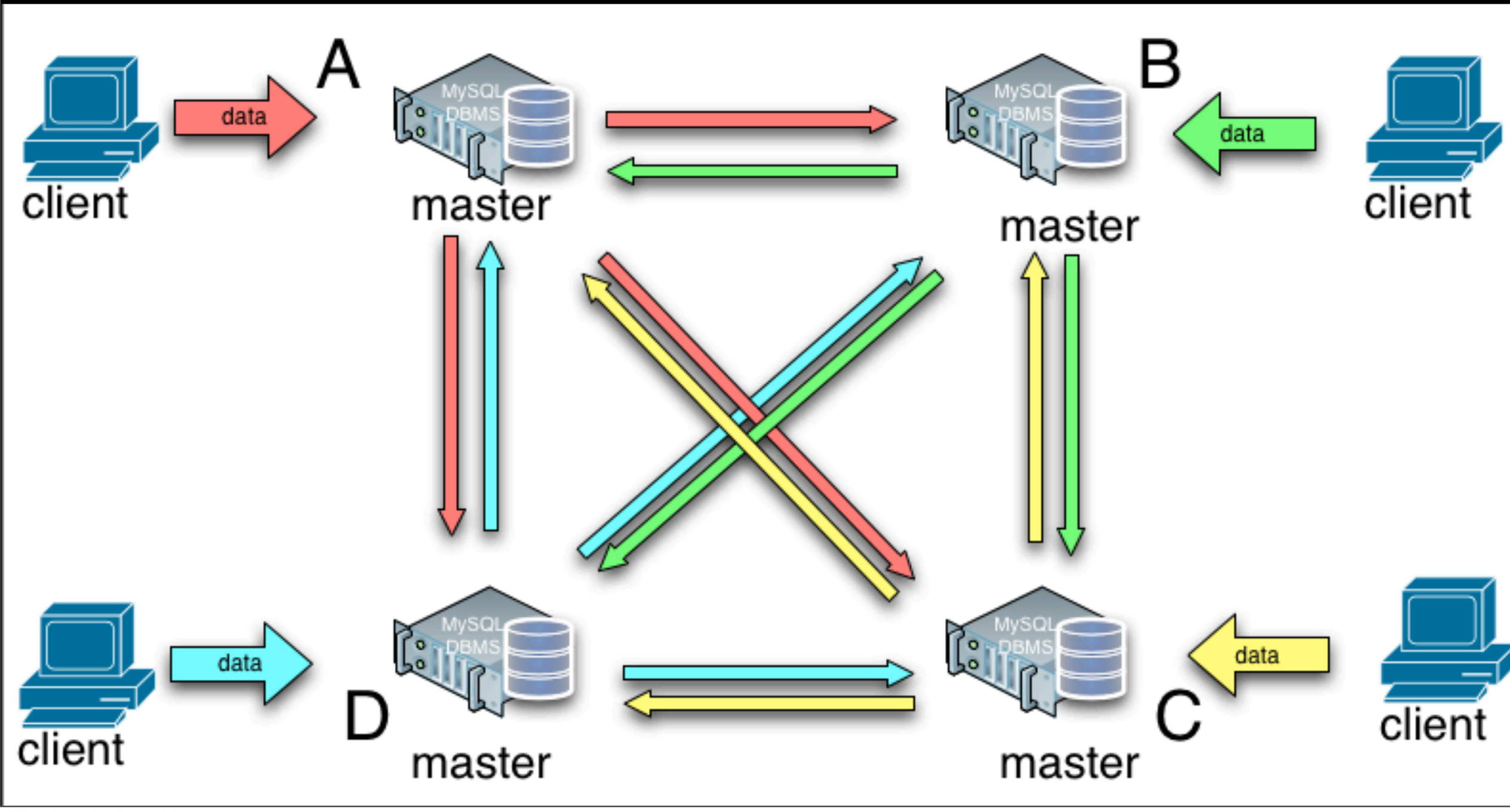
Updating 4 masters : 2 flows



Updating 4 masters : 3 flows



Updating 4 masters : 4 flows



Tungsten in practice

- installation

Installation: the long way

- Get the binaries
- Expand the tarball
- Check the requirements in the manual
- Run ./configure
- Run ./configure-service

Installation: the quick way

- Get the tarball
- Get the tungsten deployer from Google Code
- Sit back and enjoy!

Tools

- replicator
- trepctl
- thl

replicator

- It's the service provider
- You launch it once when you start
- You may restart it when you change config

trepctl

- **T**ungsten **Rep**licator **C**on**T**ro**L**ler
- It's the driving seat for your replication
- You can start, update, and stop services
- You can get specific info

thl

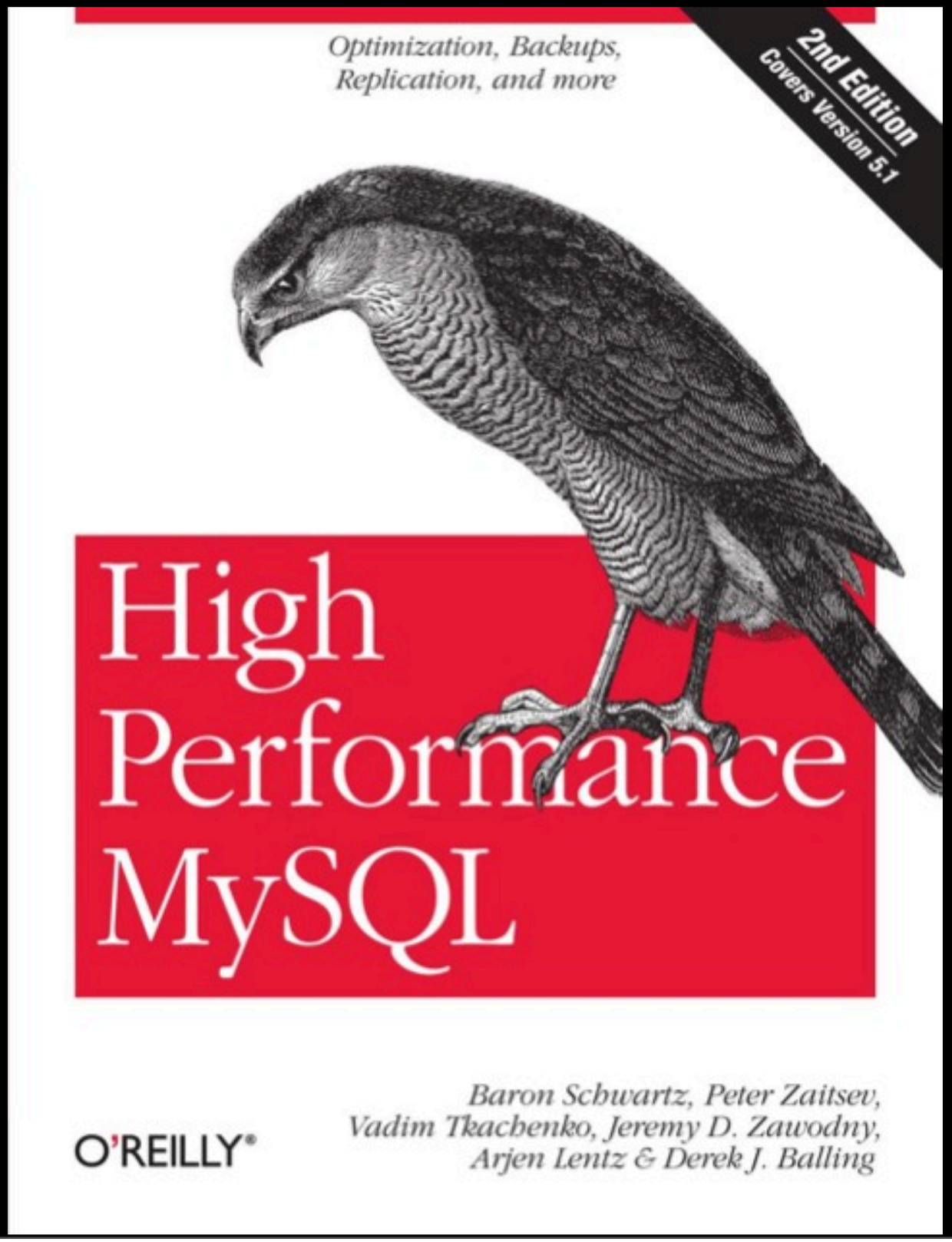
- **T**ransaction **H**istory **L**ist
- Gives you access to the Tungsten relay logs

AGENDA

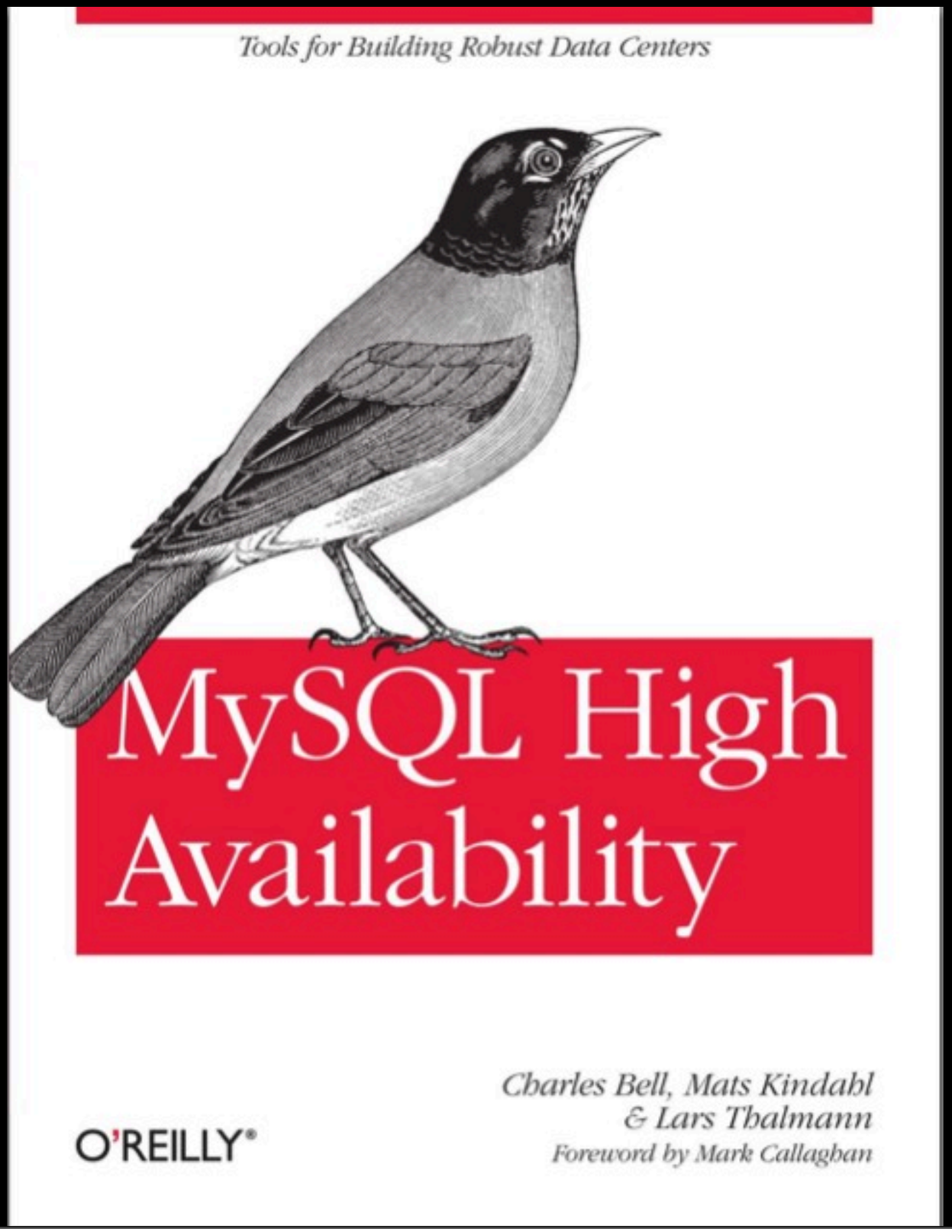
More info on replication

Books

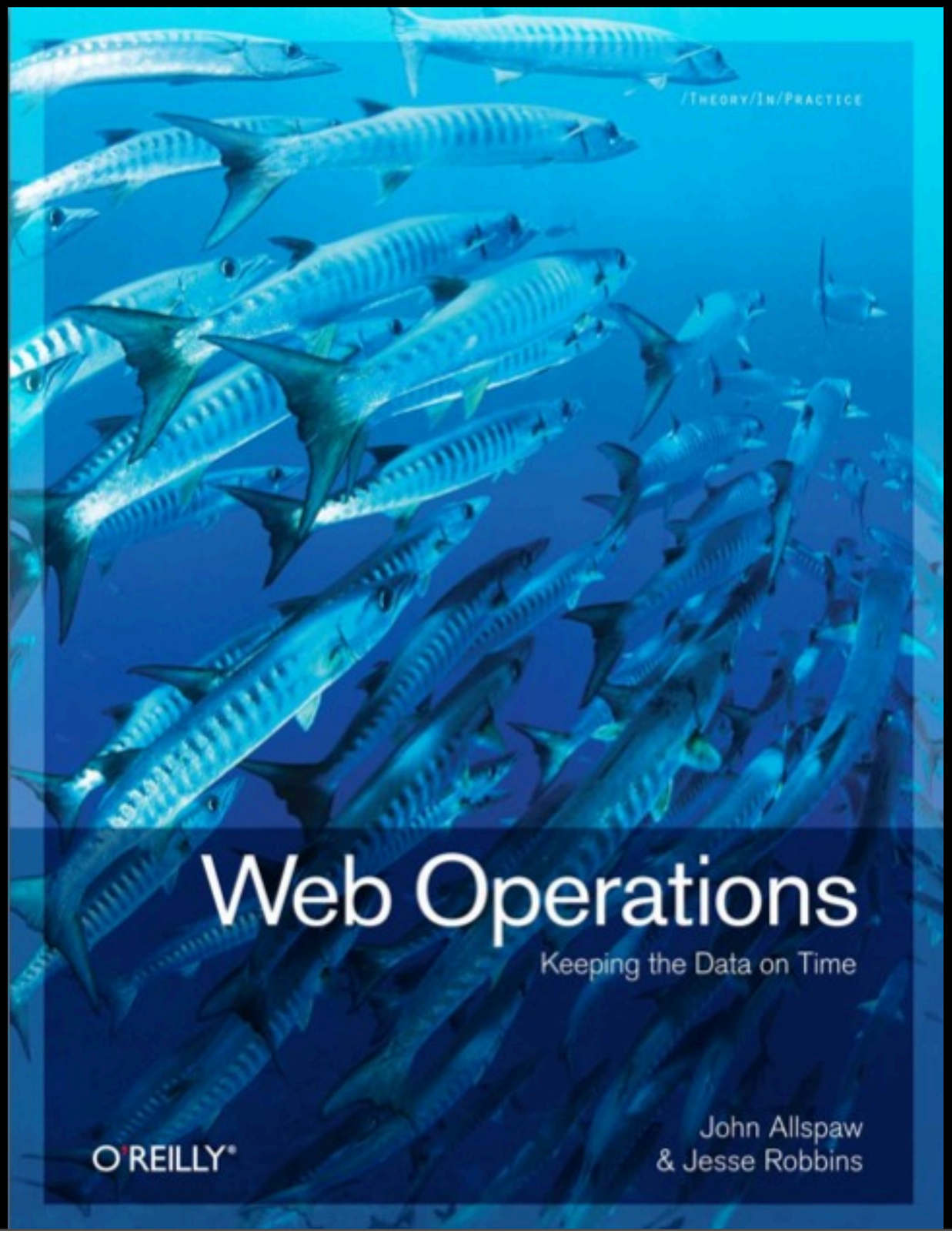
High Performance MySQL



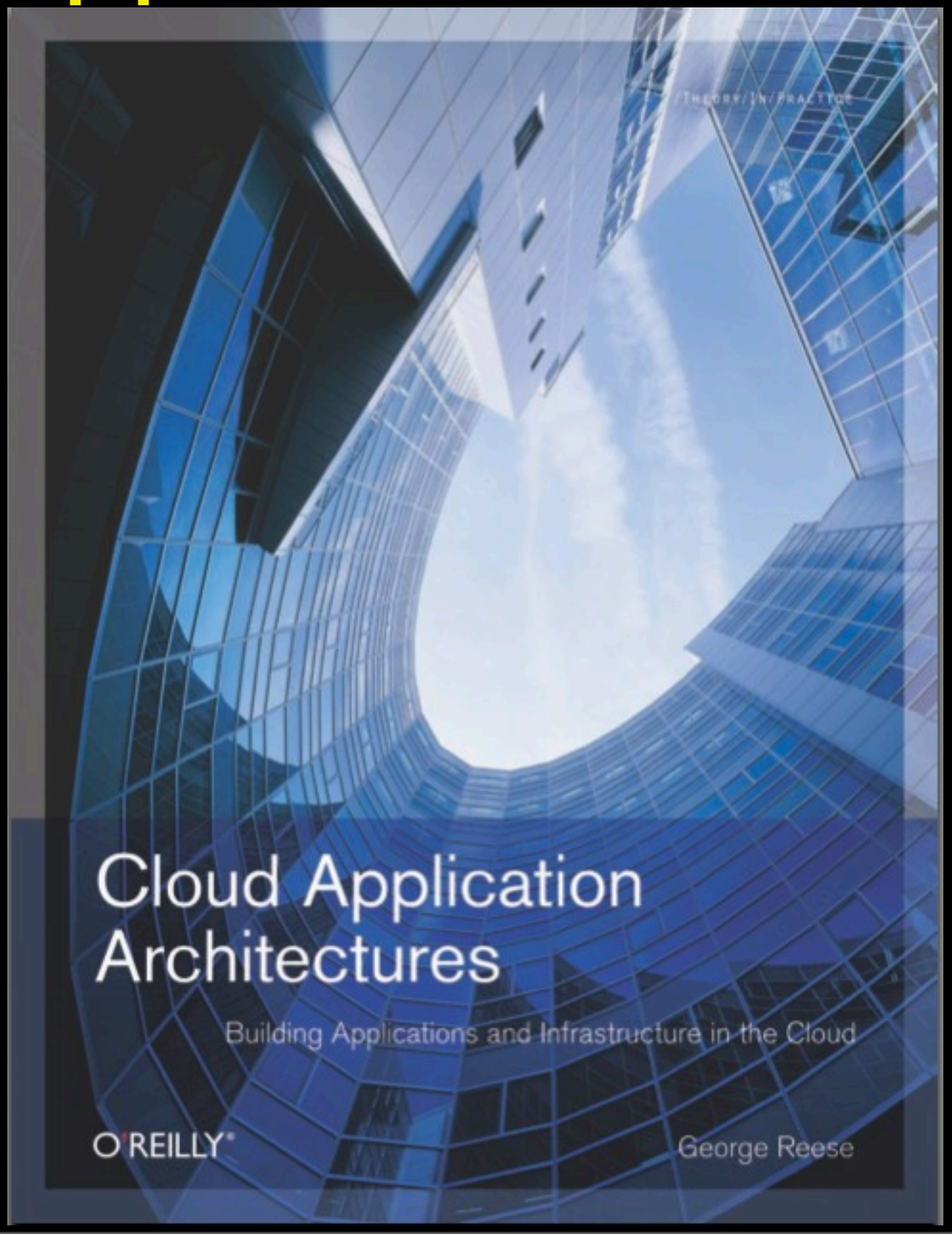
MySQL High Availability



Web Operations



Cloud Application Architectures



Talks

- Monday, 1:30pm “Learn how to cure replication deprivation with Tungsten”
- Tuesday 10:50am “Replication Update”
- Tuesday 2:00pm “Diagnosing replication failures”
- Tuesday 3:05 “Advanced replication monitoring”
- Thursday 10:50am “Replication for Availability & Durability with MySQL and Amazon RDS”
- Thursday 11:55am “Build your own PaaS for MySQL with Tungsten Enterprise”
- Thursday 2:00pm “Error detection and correction with MySQL Replication”
- Thursday 2:50pm “Performance comparisons and trade-offs for various MySQL replication schemes”

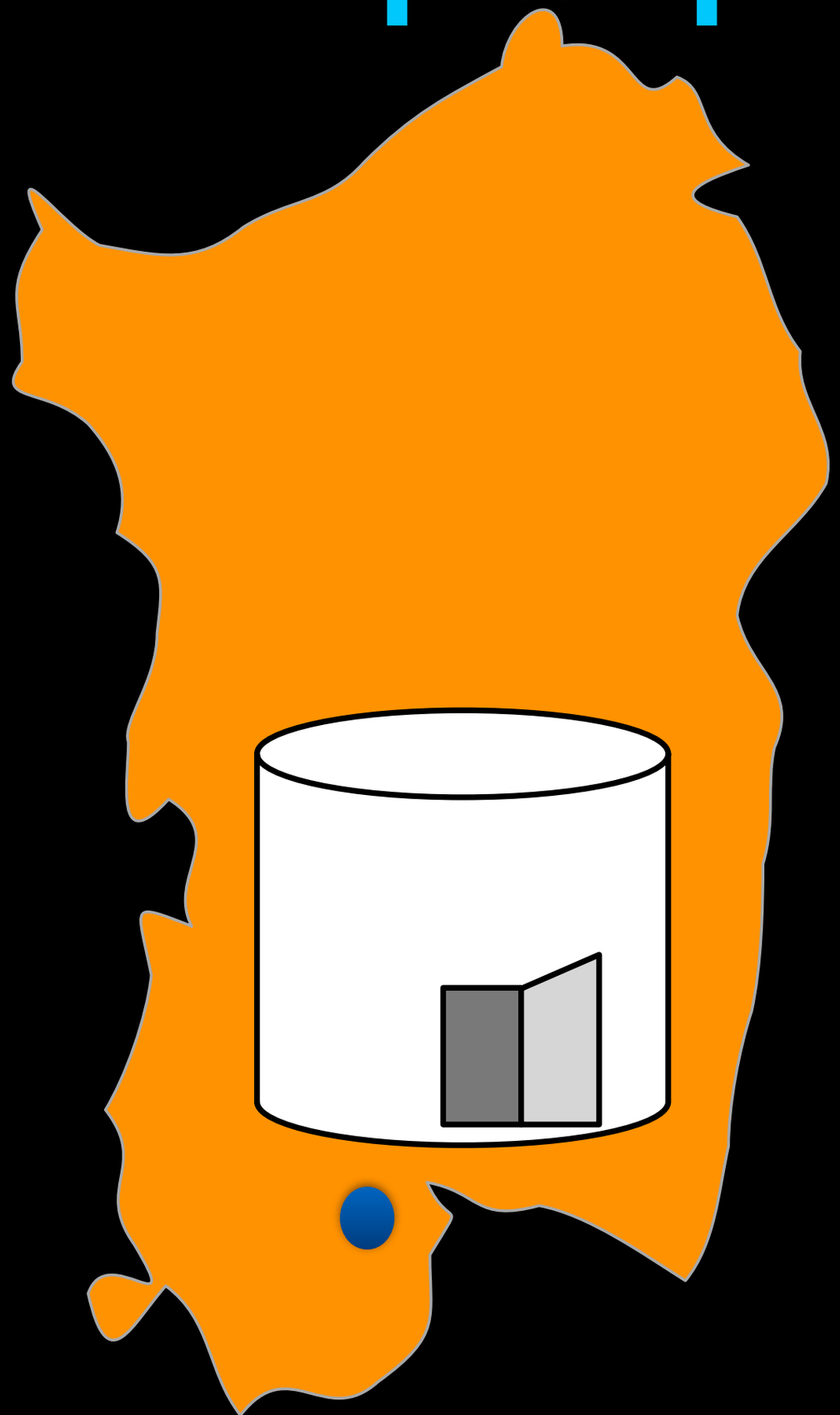
ADVERTISING

WE ARE HIRING!

QA and support engineers

http://www.continuent.com/about/careers

http://opendbcamp.org



Open Database Camp

Sardinia Technology Park

6-7-8 May 2011

Contact Information

Worldwide
560 S. Winchester Blvd., Suite 500
San Jose, CA 95128
Tel (866) 998-3642
Fax (408) 668-1009
e-mail: sales@continuent.com

Continuent Web Site:
http://www.continuent.com

Tungsten Project
http://code.google.com/p/tungsten-replicator

continuent
Open. Always Available.