Using GTID-based Replication for MySQL High Availability

Jacob Nikom

November 11, 2013

Outline

- **High Availability (HA) Basics**
  - What is HA and why do we need it?
  - Data Centers (DC) Downtime Causes and Consequences
  - High Availability Levels
  - How to Achieve Necessary Level of HA
- **MySQL Replication as High Availability Solution**
  - Major Oracle MySQL HA Solutions
  - Their Advantages and Disadvantages
  - Brief History of MySQL Replication
  - Replication Enhancements in MySQL 5.6
- **How Coordinate Replication Works**
  - Replication Data Files
  - Replication Execution
  - Replication Binary Log Coordinates
  - HA and Coordinate Replication
- **How GTID Replication Works**
  - What is GTID?
  - How to Configure GTID Replication?
  - GTID Replication Basics
  - Coordinate Replication Failover
  - GTID Replication Failover
- **Amazon Cloud-based HA Architecture**
  - AWS Main Components
  - AWS Failure Modes and Effects
  - Failover with GTID Replication and ZFS Snapshots
  - Failover Prototype and Demonstration
- **Summary**

What is HA? Why Do We Need it?

Availability of the service is a percentage of the time when the system is able to provide the service

("Service Availability: Principles and Practice" by Maria Toeroe, Francis Tam, 2012)

High Availability for Data Centers usually means:

- Guaranteed Throughput (number of transactions per second)
- Guaranteed Response time (latency)
- Guaranteed Uptime/Downtime per year (in percentiles/seconds, minutes, hours)

Definitions of some important HA terms

- **Uptime and Downtime**
  - The proportion of time a high availability service is up or down over the total time. Normally, uptime + downtime = 100%.
- **Single point of failure (SPOF)**
  - An isolated device or piece of software for which a failure will cause a downtime of the HA service.
  - The goal of an HA architecture is to remove the SPOFs.
- **Failover and Switchover**
  - Switching to a redundant or standby computer server.
  - Usually failover is automatic and operates without warning while switchover requires human intervention
- **Fencing/Stonith**
  - Often, an HA architecture is stuck by a non-responsive device that is not releasing a critical resource.
  - Fencing or Stonith (Shoot The Other Node In The Head) is then required.
- **Cluster**
  - A group of computers acting together to offer a service
- **Fault Tolerance**
  - Ability to handle failures with graceful degradation. Not all components need the same level of fault tolerance
- **Disaster Recovery**
  - The plan and technologies to restore the service in case of disaster. Often longer downtime allowed in this case.

What is HA? Why do we need it (cont.)?

High Availability vs. Continuous Availability

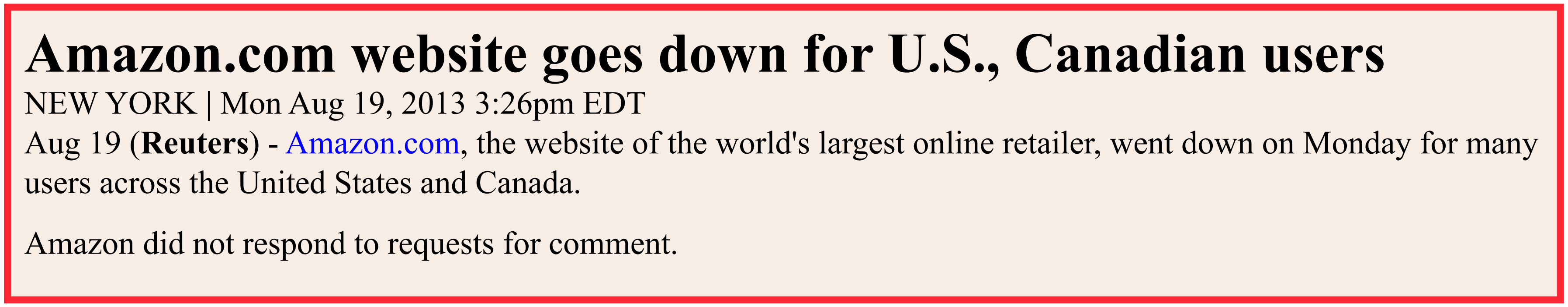
- A highly available system allows planned outages
- A continuously available system does not allow planned outages, essentially supporting no downtime operations

High Availability vs. Fault Tolerance

- A fault tolerant system in case of a component failure has no service interruption (higher solution cost)
- A highly available system has a minimal service interruption (lower solution cost)

Why we are so interested in High Availability?

Company's Worst Nightmare Scenario!



Amazon.com website goes down for U.S., Canadian users

NEW YORK | Mon Aug 19, 2013 3:26pm EDT

Aug 19 (**Reuters**) - Amazon.com, the website of the world's largest online retailer, went down on Monday for many users across the United States and Canada.

Amazon did not respond to requests for comment.

DC Downtime Causes and Consequences

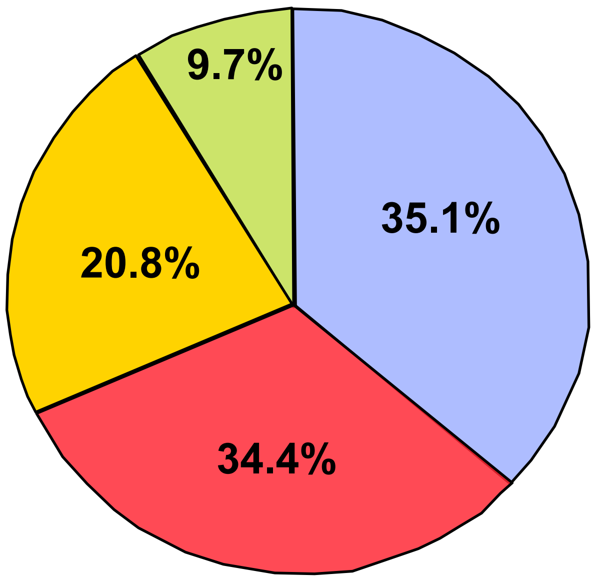
What Causes Data Center Service Downtime?

- System Failures
  - Hardware Faults
  - Software bugs or crashes
- Physical Disasters
- Scheduled Maintenance
- User Errors

MySQL Servers Downtime Causes

🟦 - Operating Environment
🟥 - Performance
🟧 - Replication
🟩 - Data Loss & Corruption

Baron Schwartz, Percona, 2011



What Are the System Downtime Effect and Impact?

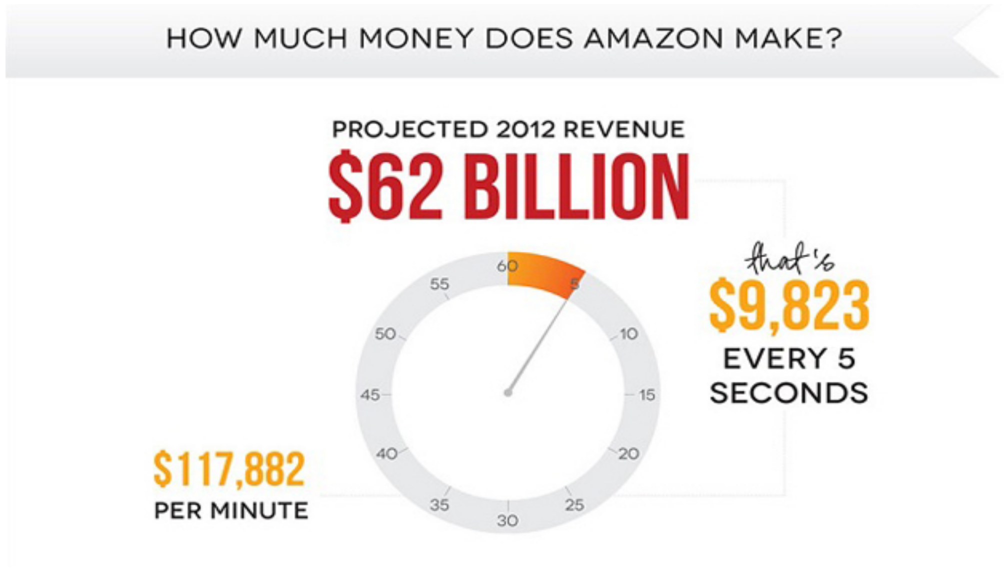
- **Effect**:
  - Service Unavailability
  - Bad response time
- **Impact**:
  - Revenue loss
  - Poor customer relationships
  - Reduced employee productivity
  - Regulatory issues

High Availability Levels

Availability Level often associated with **UPTIME**

Availability %	Downtime per year	Downtime per month	Downtime per week
90% ("one nine")	36.5 days	72 hours	16.8 hours
99% ("two nines")	3.65 days	7.20 hours	1.68 hours
99.9% ("three nines")	8.76 hours	43.8 minutes	10.1 minutes
99.99% ("four nines")	52.56 minutes	4.32 minutes	1.01 minutes
99.999% ("five nines")	5.26 minutes	25.9 seconds	6.05 seconds
99.9999% ("six nines")	31.5 seconds	2.59 seconds	0.605 seconds

Easy to calculate losses due to unavailability



How to Achieve Necessary Level of HA

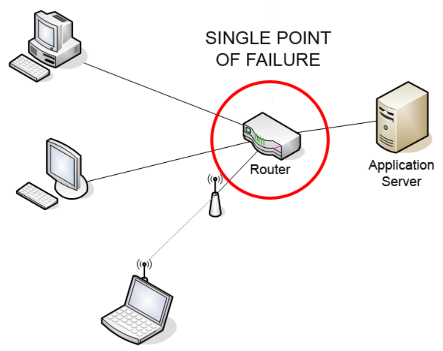
HA could be achieved by two ways:

1. Increasing the reliability of each components
2. Adding the redundant components

The first way is less efficient – the reliability of the system will be **lower** than the reliability of any individual component

The second way is more efficient - reliability of the system will be **higher** than the reliability of any individual component

Removing Single Point of Failures (**SPOF**)



#	Component	Technique	Explanation
1	Storage	RAID	If one disk crashes, the service still works
2	Servers	Clustering	If one server crashes, the service still works
3	Power Supply	UPS	If the power source fails, the UPS provides the power and the service still works
4	Network	Redundant routers	If a router were to fail connectivity would be preserved by routing traffic through a redundant connection and the service still works
5	Location	Another Data Center	If a datacenter is destroyed or disconnected, move all computation to another data center and the service still works

Why High Availability is so hard with databases?

1. High availability databases are essentially real-time systems or RTS. Sometimes they are even distributed RTS. That type of systems are traditionally very difficult to deal with.
2. Real-time data processing functionality (caches and dirty data logging) forces tight coupling between software and hardware components. Therefore software redundancy requires redundancy of corresponding hardware as well.
3. Real-time consistency between data stored on redundant components requires continuous and instantaneous synchronization. This is difficult to implement without significant overhead.

Major HA Solutions Using Oracle MySQL

#	HA Feature	MySQL Replication	DRBD	MySQL NDB Cluster
1	Platform Support	All supported by MySQL server	Linux	All Supported by MySQL cluster
2	Supported Storage Engine	Transactionality required for GTIDs	InnoDB	NDB
3	Automatic Failover	Yes, with MySQL 5.6 Utilities	Yes, with Corosync + Pacemaker	Yes
4	Failover Time	5 second + InnoDB Recovery time	Automatic failover in about 1 minute with InnoDB log files of about 100 MB	1 second or less
5	Replication Mode	Asynchronous+ Semi-synchronous	Asynchronous+ Synchronous	Synchronous
6	Shared Storage	No, distributed access nodes	No, distributed access nodes	No, distributed access nodes
7	Number of Nodes	Master + Multiple Slaves	Active/Passive Master + Multiple Slaves	2555 + Multiple Slaves
8	Availability Level	99.9%	99.99%	99.999%

Advantages and Disadvantages

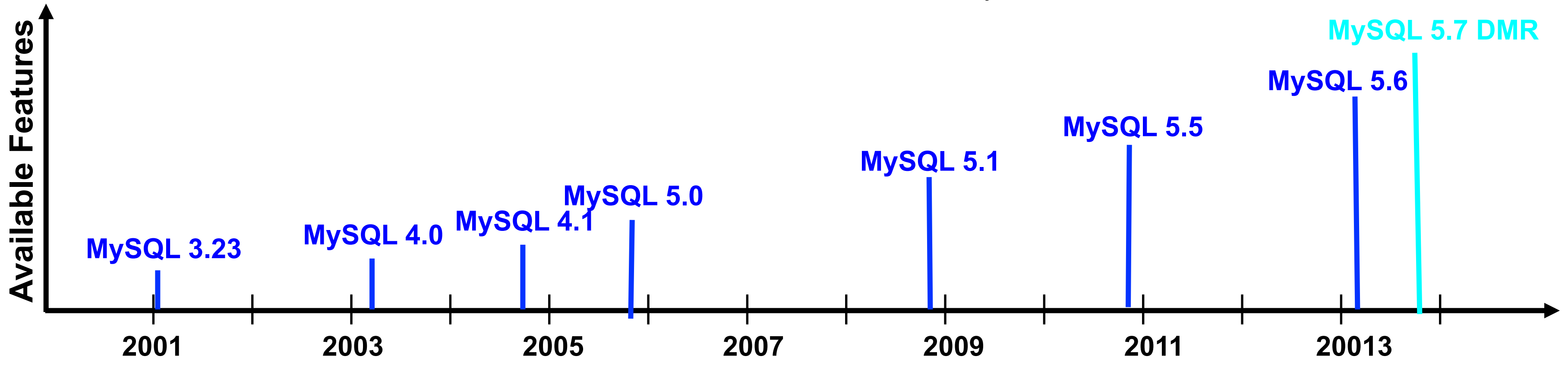
#	HA Solution	Advantages	Disadvantages
1	MySQL Replication (before 5.6)	• Simple • Inexpensive • Extends existing database architecture • All the servers can be used, no idle standby • Supports MyISAM • Caches on failover slave are not cold • Online schema changes • Low impact backups • **99.9% availability**	• Variable level of availability (98-99.9+%) • Could be a SPOF • Replication can break • Replication can lag behind • Replication can be out of sync • Not suitable for high write loads • Reads scale only if they are split from writes • Can lose data
2	DRBD	• No data loss • Much higher write capacity • No SPOF with DRBD • Provides high availability and data integrity across two servers in the event of hardware or system failure. • Ensures data integrity by enforcing write consistency on the primary and secondary nodes. • **99.99% availability**	• High load on the network • Only works with engine supporting auto-recovery • More complex: NIC bounding, fencing, etc. • Requires fencing • A server is standby, idle hardware • Cold cache after failover • No online schema change • Corruption propagation
3	MySQL NDB Cluster	• No Single Point of Failure • Auto-sharding for write-scalability • SQL and NoSQL interfaces • Real-time responsiveness • Active / active geographic replication • Online scaling and schema upgrades • **99.999% availability**	• Incompatible with typical database architecture • Complex, much than other solutions • Needs work on schema and queries for good performance • Higher skill set required • Poor for large joins • Size of dataset more limited, large memory footprint • Minimum of physical servers

MySQL Replication is the most convenient HA Solutions!

Must Increase MySQL Replication Availability!

Brief History of MySQL Replication

- **MySQL 3.23 - Generally Available, January 2001**
  - MySQL Replication came to be (3.23.15 – May 2000).
  - Replication filters
- **MySQL 4.0 - Generally Available, March 2003**
  - Two Replication Threads instead of just one.
  - Slave Relay logs.
- **MySQL 4.1 - Generally Available, October 2004**
  - Replication over SSL.
  - Disk synchronization options for binary log.
- **MySQL 5.0 - Generally Available, October 2005**
  - Replication of Stored Routines and Triggers.
  - Slave retries transactions on transient errors.
- **MySQL 5.1 - Generally Available, November 2008**
  - Row-based Replication (RBR).
- **MySQL 5.5 - Generally Available, December 2010**
  - Semi-sync replication.
  - Replication Heartbeats.
  - RBR type conversion.
- **MySQL 5.6 - Generally Available, February 2013**
  - Crash-safe Slaves.
  - Global Transaction Ids.
  - Replication Event Checksums.
  - Binary Log Group Commit.
  - Multi-threaded Slaves.
  - RBR enhanced.
  - MySQL Utilities 1.3, GA on August 2013
- **MySQL 5.7.2 DMR, September 2013**
  - Multi-Threaded Inter-Transactional Replication
  - Lossless Semi-Synchronous Replication
  - MySQL Utilities 1.4



Replication Enhancements in MySQL 5.6

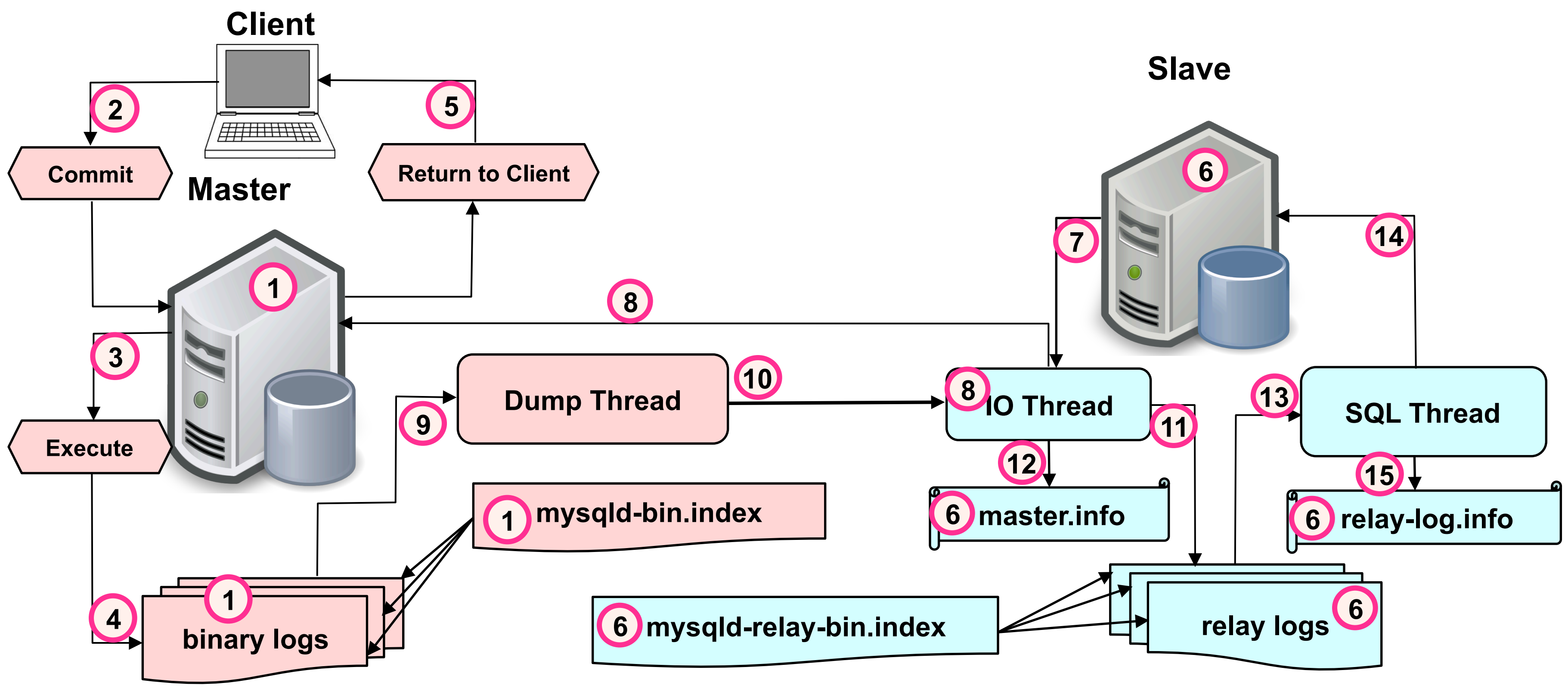
- **Failover & Recovery:**
  - Global Transaction Identifiers (GTID)
  - Server UUIDs
  - Crash Safe Slaves & Binary Logs
  - Replication Failover and Administration Utilities
- **Data Integrity**:
  - Replication Event Checksums
- **Performance**:
  - Multi-Threaded Slaves
  - Binary Log Group Commit
  - Optimized Row-Based Replication
- **Database Operations**:
  - Replication Utilities
  - Time-Delayed Replication
  - Remote Binlog Backup
  - Information Log Events

How Coordinate Replication Works

1 1. Master server enables binary log
2 2. Client commits query to the master
3 3. Master executes the query and commits it
4 4. Master stores the query in the binary log as en event
5 5. Master returns to the client with commit confirmation
6 6. Slave server configures replication

```
mysql> CHANGE MASTER TO
MASTER_HOST='12.34.56.789',MASTER_USER='slave_user',
MASTER_PASSWORD='password',
MASTER_LOG_FILE='mysql-bin.000001', MASTER_LOG_POS=  107;
```

7 7. Replication starts mysql> START SLAVE;
8 8. IO thread starts and initiates dump thread on the master
9 9. Dump thread reads events from binary log
10 10. Dump thread sends events to IO thread from the slave
11 11. IO thread writes events into relay log
12 12. IO thread updates master.info file parameters
13 13. SQL thread reads relay log events
14 14. SQL thread executes events on the slave
15 15. SQL thread updates relay-log.info file



Replication Data Files

Master Server 5.5

```
[root@node1 data]# ls -l
-rw-r----- 1 mysql mysql 144703488 Oct 22 22:33 ibdata1
-rw-r----- 1 mysql mysql  67108864 Oct 22 22:33 ib_logfile0
-rw-r----- 1 mysql mysql  67108864 Oct 22 22:33 ib_logfile1
drwx------ 2 mysql mysql        81 May 20 23:21 mysql
-rw-rw---- 1 mysql mysql       332 Oct 22 22:30 mysqld-bin.000001
-rw-rw---- 1 mysql mysql       354 Oct 22 22:33 mysqld-bin.000002
-rw-rw---- 1 mysql mysql        40 Oct 22 22:31 mysqld-bin.index
-rw-rw---- 1 mysql mysql         5 Oct 22 22:31 mysqld.pids
-rw-r----- 1 mysql root      12616 Oct 25 03:20 mysql-error.err
drwx------ 2 mysql mysql        55 May 20 23:21 performance_schema
drwx------ 2 mysql mysql         2 Oct 22 22:33 test
[root@node1 data]#
```

Slave Server 5.5

```
[root@node1 data]# ls -l
-rw-r----- 1 mysql mysql 144703488 Oct 27 19:47 ibdata1
-rw-r----- 1 mysql mysql  67108864 Oct 27 19:47 ib_logfile0
-rw-r----- 1 mysql mysql  67108864 Oct 27 19:47 ib_logfile1
-rw-rw---- 1 mysql mysql        60 Oct 22 22:31 master.info
drwx------ 2 mysql mysql        81 May 20 23:21 mysql
-rw-rw---- 1 mysql mysql         6 Oct 22 22:31 mysqld.pids
-rw-rw---- 1 mysql mysql       205 Oct 22 22:31 mysqld-relay-bin.000001
-rw-rw---- 1 mysql mysql       526 Oct 22 22:33 mysqld-relay-bin.000002
-rw-rw---- 1 mysql mysql        52 Oct 22 22:31 mysqld-relay-bin.index
-rw-rw---- 1 mysql root      11309 Oct 22 22:31 mysql-error.err
-rw-rw---- 1 mysql mysql        58 Oct 22 22:31 relay-log.info
drwx------ 2 mysql mysql        55 May 20 23:21 performance_schema
drwx------ 2 mysql mysql         2 Oct 22 22:33 test
```

File mysqld-bin.index

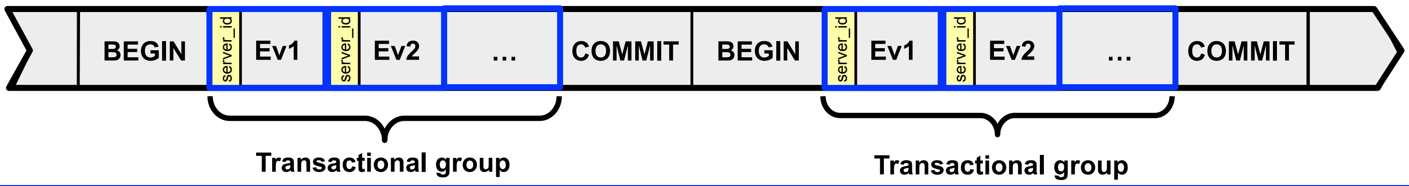
```
[root@node1 data]# more /usr/local/mysql/data/mysqld-bin.index
/usr/local/mysql/data/mysqld-bin.000001
/usr/local/mysql/data/mysqld-bin.000002
[root@node1 data]#
```

File mysqld-relay-bin.index

```
[root@node1 data]# more /usr/local/mysql/data/mysqld-relay-bin.index
/usr/local/mysql/data/mysqld-relay-bin.000001
/usr/local/mysql/data/mysqld-relay-bin.000002
[root@node1 data]#
```

Events Layout on a Binary Log File (or Relay Log File)

- File based log that records the changes on the master.
- Statement or Row based format (may be intermixed).
- Split into transactional groups containing multiple events
- Each event contains server_id value.



Replication Data Files (cont.)

master.info

1 15	Number of lines in the file
2 mysqld-relay-bin.000001	Current binlog file being read(**Master_Log_File**)
3 4723	Last binlog position read (**Read_Master_Log_Pos**)
4 node1	Master host connected to (**Master_Host**)
5 root	Replication user (**Master_User**)
6 kiva	Replication password
7 3306	Master port used (**Master_Port**)
8 60	How many times slave will try to reconnect (**Connect_Retry**)
9 0	If SSL is enabled is 1, 0 otherwise
10 - 15	SSL-related information

relay-log.info

1 ./mysqld-relay-bin.000001	Relay log file (**Relay_Log_File**)
2 874	Relay log position (**Relay_Log_Pos**)
3 mysql-bin.000001	Master log file (**Relay_Master_Log_File**)
4 729	Master log position (**Exec_Master_Log_Pos**)

Coordinate Replication Execution

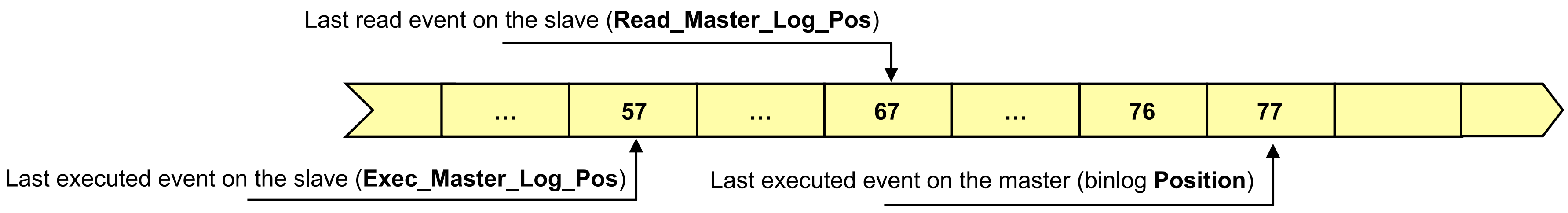
Replication coordinates:

1. Master binary log file name (**Master_Log_File**) - the name of the particular binary log on the master (like mysqld-bin.000001)
2. Master binary log position (**Binary_Log_Pos**) – the number of the last event executed on the master (end of the binlog file)
3. Position in the master binary log where IO thread read to (**Read_Master_Log_Pos**)
4. Position in the master binary log where SQL thread executed to (**Exec_Master_Log_Pos**)
5. Slave relay log name (**Relay_Log_File**) – the name of the particular relay log on the salve (like mysqld-relay-bin.000001)
6. Slave relay log position where SQL thread executed to (**Relay_Log_Pos**) – the last event in the relay log on the slave

```
mysql> SHOW MASTER STATUS;
+---------------+----------+--------------+------------------+
| File          | Position | Binlog_Do_DB | Binlog_Ignore_DB |
+---------------+----------+--------------+------------------+
| mysql-bin.003 | 77       |              |                  |
+---------------+----------+--------------+------------------+
```

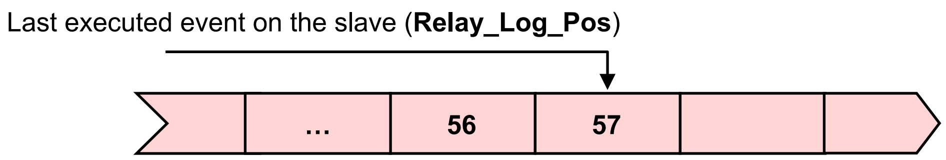
File: MySQL binary log file on the master
Position: Last executed position (next write) in the binary log. If the slave caught up with the master, it should execute next events from this position.

Master Binary Log



Last read event on the slave (Read_Master_Log_Pos)
…
57
…
67
…
76
77
Last executed event on the slave (Exec_Master_Log_Pos)
Last executed event on the master (binlog Position)

Slave Relay Log



Replication Binary Log Coordinates

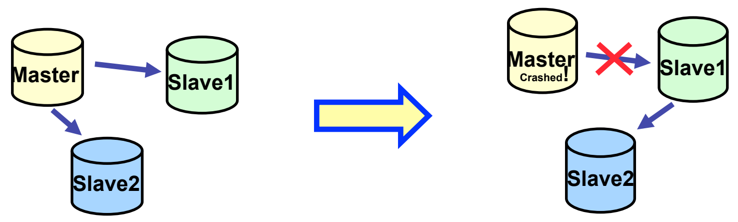
```
mysql> SHOW SLAVE STATUS\G
*************************** 1. row ***************************
              Slave_IO_State: Waiting for master to send event
                 Master_Host: 127.0.0.1
                 Master_User: master_user
                 Master_Port: 26768
               Connect_Retry: 60
             Master_Log_File: mysql-bin.000001 (IO Thread reads this file)
         Read_Master_Log_Pos: 4723 (Position in master binary log file where IO Thread has read to)
              Relay_Log_File: mysqld-relay-bin.000001
               Relay_Log_Pos: 874  (Position in the relay log file where SQL thread read and executed events
       Relay_Master_Log_File: mysql-relay-bin.000001
            Slave_IO_Running: Yes
           Slave_SQL_Running: Yes
. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . :  . . . . . . .
Last_Errno: 0 Last_Error:
                Skip_Counter: 0
         Exec_Master_Log_Pos: 729  (Position in master binary log file that SQL Thread read and executed up to
             Relay_Log_Space: 1042 The total combined size of all existing relay log files
             Until_Condition: None
              Until_Log_File:
               Until_Log_Pos: 0
          Master_SSL_Allowed: No
. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . : . . . .
Seconds_Behind_Master: 0
Master_SSL_Verify_Server_Cert: No
               Last_IO_Errno: 0
               Last_IO_Error:
              Last_SQL_Errno: 0
              Last_SQL_Error:
1 row in set (0.00 sec)
```

Coordinates usage examples:

Connect to the master using master's binary log
slave> CHANGE MASTER TO
MASTER_HOST='12.34.56.789',MASTER_USER='slave_user',
MASTER_PASSWORD='password',
MASTER_LOG_FILE='**mysql-bin.000001**', MASTER_LOG_POS=4723;

Connect to the new master/old slave using slave's relay log
slave> CHANGE MASTER TO
MASTER_HOST='12.34.56.789',MASTER_USER='slave_user',
MASTER_PASSWORD='password',
MASTER_LOG_FILE='**mysql-relay-bin.000001**', MASTER_LOG_POS=729;

Failover



HA and Coordinate Replication

- Coordinate based replication is **great** – it is easy to setup
- Coordinate based replication is **bad** – it is difficult to failover
  - ❑ When the master fails, the slaves are ready to replace it
  - ❑ However, the process of failure detection and acting upon in case of multiple servers requires significant DBA intervention
  - ❑ Difficult to follow changes through a complex replication stream that go to multiple servers

How to Improve It?

If every transaction has its own globally unique identifier (GTID), it becomes a lot easier to track changes

Advantages

- It is possible to identify a transaction uniquely across the replication servers.
- Make the automation of failover process much easier. There is no need to do calculations, inspect the binary log and so on. Just execute the command **MASTER_AUTO_POSITION**=1.
- At application level it is easier to do WRITE/READ split. After a write on the MASTER you have a GTID so just check if that GTID has been executed on the SLAVE that you use for reads.
- Development of new automation tools isn't a pain now.

Drawbacks

- Additional complexity
- Incompatibility with existing solution – coordinate based replication

What is GTID?

Where GTID comes from?

```
# ls -l /usr/local/mysql/data
total 537180
-rw-r----- 1 mysql mysql        56 Oct 17 10:49 auto.cnf
drwx------ 2 mysql mysql      4096 Oct 17 10:49 bench/
-rw-r----- 1 mysql mysql 348127232 Oct 17 11:58 ibdata1
-rw-rw---- 1 mysql mysql 100663296 Oct 17 11:58 ib_logfile0
-rw-rw---- 1 mysql mysql 100663296 Oct 17 11:24 ib_logfile1
drwx------ 2 mysql mysql     32768 Oct 17 10:55 mhs/
drwx------ 2 mysql mysql      4096 Oct 17 10:49 mysql/
-rw-rw---- 1 mysql mysql         6 Oct 17 11:58 mysqld.pids
-rw-r----- 1 mysql root       9131 Oct 17 11:58 mysql-error.err
drwx------ 2 mysql mysql      4096 Oct 17 10:49 performance_schema/
drwxr-xr-x 2 mysql mysql      4096 Oct 17 10:49 test/
```

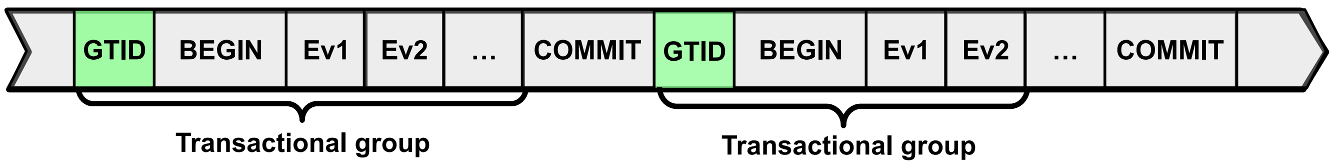
```
[root@jnikom-linux data]# more auto.cnf
[auto]
server-uuid=965d996a-fea7-11e2-ba15-001e4fb6d589
[root@jnikom-linux data]#
```

965d996a-fea7-11e2-ba15-001e4fb6d589:1

- **Server identifier** – 128-bit identification number (SERVER_UUID).
- It logically identifies the server where the transaction was originated.
- Every server has its own **SERVER_UUID**.
- If you deleted it it will be regenerated after you restarted your server
- **GTID is written into binary log**

- **TIN** – 64-bit transaction identification number.
- A sequence number incremented with every new transaction.
- It starts with 1. There is no 0

MySQL 5.6 binary log



How to Configure GTID Replication?

my.cnf new additional parameters

- **gtid_mode**
  - It could be ON or OFF (not 1 or 0)
  - It enables the GTID on the server
- **log_bin (existed)**
  - Enables binary logs
  - Mandatory to create a replication
- **log-slave-updates**
  - Slave servers must log its changes
  - Needed for server promotion/ demotion
- **enforce-gtid-consistency**
  - Forces the server to be safe by using only transactional tables
  - Non-transactional statements are denied by the server.

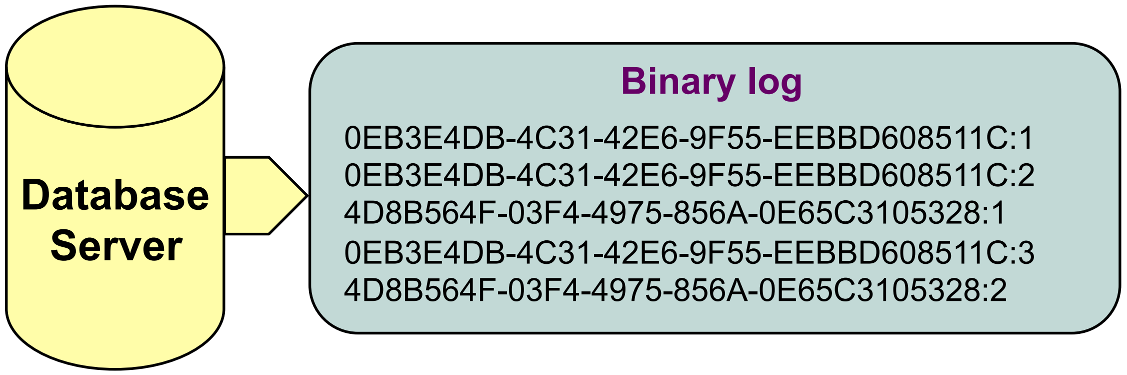
New replication configuration command

```
slave> CHANGE MASTER TO MASTER_HOST='node1',
                  MASTER_USER='roor',
                  MASTER_PASSWORD='kiva',
                  MASTER_AUTO_POSITION=1;
```

```
mysql> SHOW SLAVE STATUS\G
*************************** 1. row ***************************
                Slave_IO_State: Waiting for master to send event
                   Master_Host: node1
                   Master_User: root
                   Master_Port: 3306
                 Connect_Retry: 60
               Master_Log_File: mysqld-bin.000002
           Read_Master_Log_Pos: 354
                Relay_Log_File: mysqld-relay-bin.000002
                 Relay_Log_Pos: 526
         Relay_Master_Log_File: mysqld-bin.000002
              Slave_IO_Running: Yes
             Slave_SQL_Running: Yes
   . . . . . . . . . . . . . .:
                    Last_Errno: 0
                    Last_Error:
                  Skip_Counter: 0
           Exec_Master_Log_Pos: 354
               Relay_Log_Space: 731
               Until_Condition: None
                Until_Log_File:
                 Until_Log_Pos: 0
            Master_SSL_Allowed: No
   . . . . . . . . . . . . . .:
         Seconds_Behind_Master: 0
Master_SSL_Verify_Server_Cert: No
                 Last_IO_Errno: 0
                 Last_IO_Error:
                Last_SQL_Errno: 0
                Last_SQL_Error:
   Replicate_Ignore_Server_Ids:
              Master_Server_Id: 28
                   Master_UUID: b9ff49a4-3b50-11e3-85a5-12313d2d286c
              Master_Info_File: mysql.slave_master_info
                     SQL_Delay: 0
           SQL_Remaining_Delay: NULL
       Slave_SQL_Running_State: Slave has read all relay log; waiting for the slave I/O thread
            Master_Retry_Count: 86400
                   Master_Bind:
       Last_IO_Error_Timestamp:
      Last_SQL_Error_Timestamp:
                Master_SSL_Crl:
            Master_SSL_Crlpath:
            Retrieved_Gtid_Set: b9ff49a4-3b50-11e3-85a5-12313d2d286c:2
             Executed_Gtid_Set: b9ff49a4-3b50-11e3-85a5-12313d2d286c:1-2
                 Auto_Position: 1
1 row in set (0.00 sec)
```

GTID Replication Basics

- Each server has binary log (master and slave)
- GTIDs are written into binary log
- GTIDs executed on a server contained in a new, read-only, global server variable GTID_EXECUTED
- GTID_EXECUTED holds the range of GTIDs committed on this server as a string


Database Server
Binary log
0EB3E4DB-4C31-42E6-9F55-EEBBD608511C:1
0EB3E4DB-4C31-42E6-9F55-EEBBD608511C:2
4D8B564F-03F4-4975-856A-0E65C3105328:1
0EB3E4DB-4C31-42E6-9F55-EEBBD608511C:3
4D8B564F-03F4-4975-856A-0E65C3105328:2

```
mysql> SELECT @@GLOBAL.GTID_EXECUTED;
+------------------------------------------------------------------------------------+
| @@GLOBAL.GTID_EXECUTED                                                             |
+------------------------------------------------------------------------------------+
| 0EB3E4DB-4C31-42E6-9F55-EEBBD608511C:1-3,
4D8B564F-03F4-4975-856A-0E65C3105328:1-2 |
+------------------------------------------------------------------------------------+
1 row in set (0.00 sec)
```

For each server binary log serves as "GTID repository"

GTID Replication Basics (cont.)

- **GTIDs set possesses both cardinal and ordinal properties**
- **Two sets of GTIDs could be compared and sorted at the same time**
- **Those properties define powerful model for tracking transactions**

```
master> SELECT @@GLOBAL.GTID_EXECUTED;
+------------------------------------------------+
| @@GLOBAL.GTID_EXECUTED                         |
+------------------------------------------------+
| 4D8B564F-03F4-4975-856A-0E65C3105328:1-1000000 |
+------------------------------------------------+
```

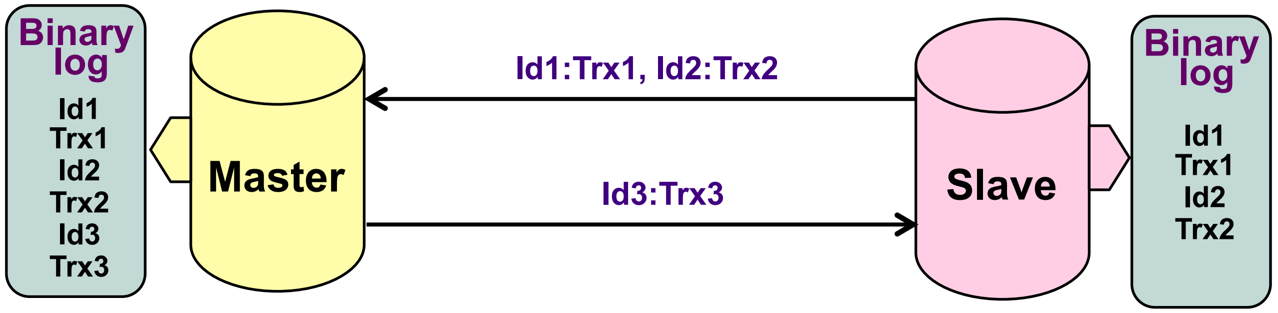
```
slave> SELECT @@GLOBAL.GTID_EXECUTED;
+-----------------------------------------------+
| @@GLOBAL.GTID_EXECUTED                        |
+-----------------------------------------------+
| 4D8B564F-03F4-4975-856A-0E65C3105328:1-999999 |
+-----------------------------------------------+
```

It is easy to find:

1. One transaction is missing 2. Which one is missing

New Replication Protocol

1. When slave connects to the master, it sends the range of GTIDs that slave has executed and committed
2. In response the master sends all ***other*** transactions, i.e. those that the slave has not yet executed

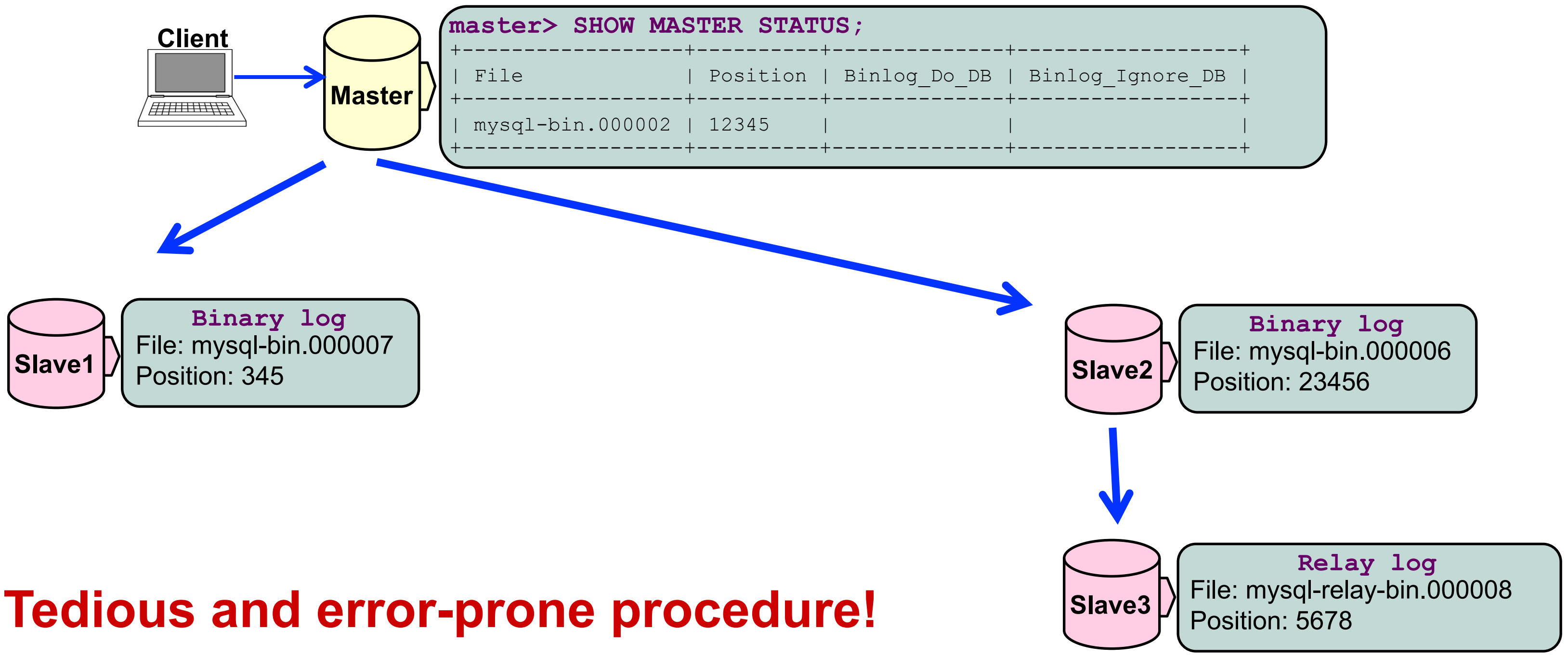


- SQL command to tell the server to use the new protocol is: **CHANGE MASTER TO MASTER_AUTO_POSITION = 1**;
- If **MASTER_AUTO_POSITION = 1**, you cannot specify **MASTER_LOG_FILE** or **MASTER_LOG_POS**.

Coordinate Replication Failover

Switching to the new master

1. Find new master binary log coordinates (file name and position) using "SHOW MASTER STATUS" command
2. Switch to the new master using **"CHANGE MASTER TO MASTER_HOST …"** command using new master binary log coordinates



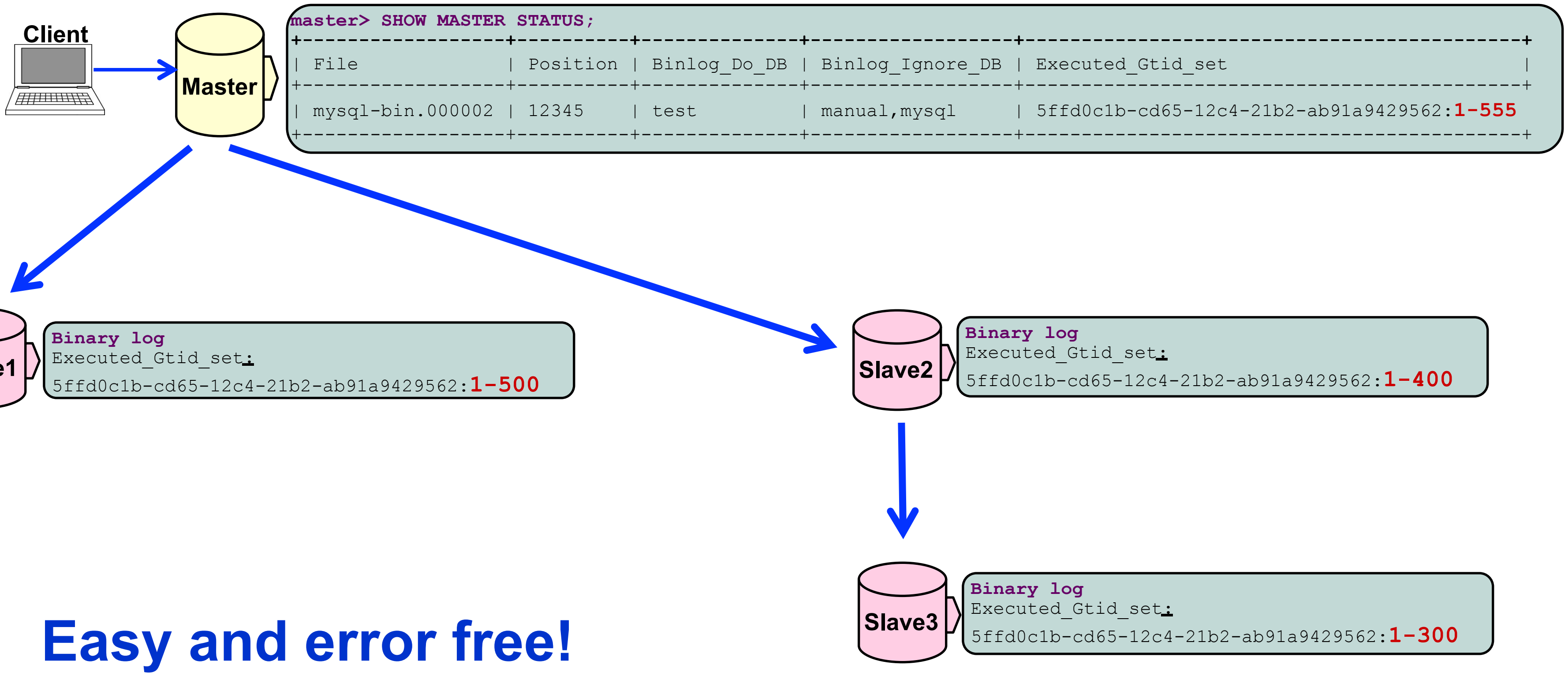
```
master> SHOW MASTER STATUS;
+------------------+----------+--------------+------------------+
| File             | Position | Binlog_Do_DB | Binlog_Ignore_DB |
+------------------+----------+--------------+------------------+
| mysql-bin.000002 | 12345    |              |                  |
+------------------+----------+--------------+------------------+
```

Tedious and error-prone procedure!

GTID Replication Failover

Switching to the new master

1. Switch to the new master using **"CHANGE MASTER TO MASTER_AUTO_POSITION = 1;"** command



```
master> SHOW MASTER STATUS;
+------------------+----------+--------------+------------------+------------------------------------------+
| File             | Position | Binlog_Do_DB | Binlog_Ignore_DB | Executed_Gtid_set                        |
+------------------+----------+--------------+------------------+------------------------------------------+
| mysql-bin.000002 | 12345    | test         | manual,mysql     | 5ffd0c1b-cd65-12c4-21b2-ab91a9429562:1-555
+------------------+----------+--------------+------------------+------------------------------------------+
```

```
Binary log
Executed_Gtid_set:
5ffd0c1b-cd65-12c4-21b2-ab91a9429562:1-500
```

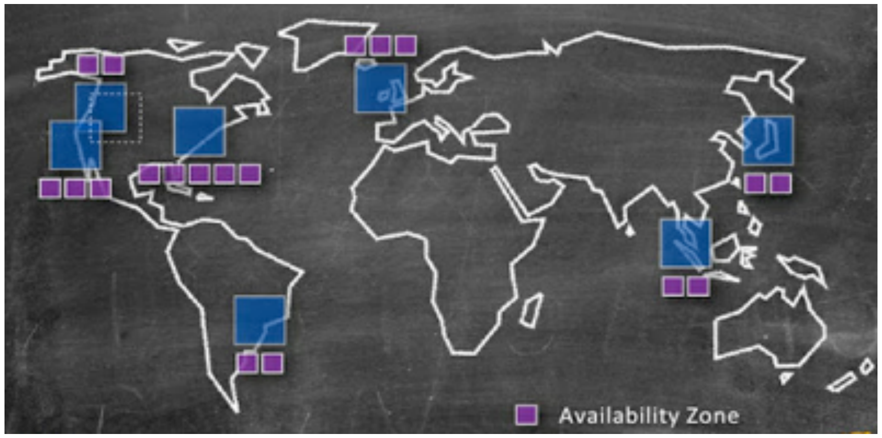
```
Binary log
Executed_Gtid_set:
5ffd0c1b-cd65-12c4-21b2-ab91a9429562:1-400
```

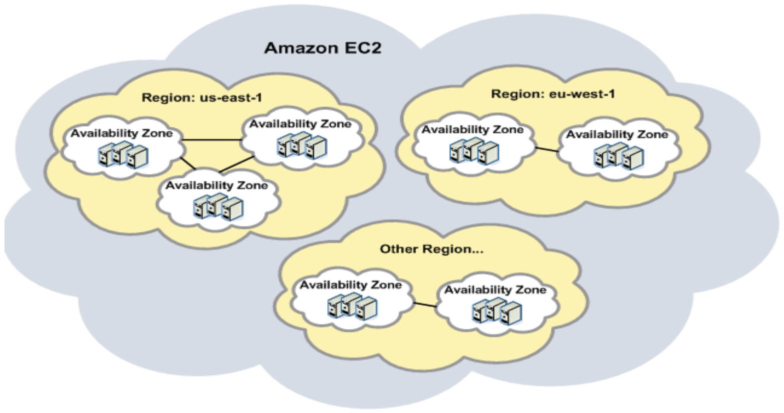
```
Binary log
Executed_Gtid_set:
5ffd0c1b-cd65-12c4-21b2-ab91a9429562:1-300
```

Easy and error free!

Amazon Cloud-based HA Architecture

Regions and Availability Zones (AZ) (as of 07-24-2-12)



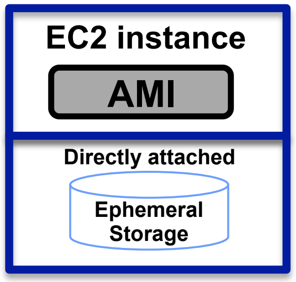


Country	Region	State/City	AZ
USA	US-West	Oregon	A,B
USA	US-West	California	A,B,C
USA	US-East	Virginia	A,B,C,D,E
Brazil	San Paulo	San Paulo	A,B
Ireland	EU	Dublin	A,B,C
Japan	Asia-Pacific	Tokyo	A,B
Singapore	Asia-Pacific	Singapore	A,B

07-24-2012

Connection points	Connectivity Type	Average Latencies[1], [2]
Region-to-Another-Region	WAN	100 – 500 ms
AZ-to-Another-AZ	LAN	10-50 ms
AZ-to-Same-AZ	LAN	2 - 10 ms

AWS Main Components



Definition: EC2 instance is a server running MHS application using Amazon Machine Image (AMI) software.

Properties:

1. The server can fail due to own hardware problems or due to AZ outage.
2. Performance varies up to 60% between instance of the same type.

Price: depends upon instance type ($0.03 – 3.10 per hour)

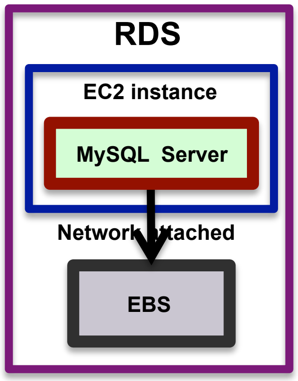
Definition: Ephemeral or instance storage is the HDD or SSD directly attached to the EC2 instance (physical node). This storage exists for every single EC2 instance even if it's not used

Performance: HDD – 0.1 ms, SSD 0.001 ms, no network latency

Properties:

1. Good for short term persistence
2. The fastest and the most predicable performance
3. It is not shared with other instances
4. Does not rely on network for its access
5. After an accidental reboot, like power outage, the content of the storage remains intact and readily available
6. After shutdown the content of the storage no longer exists. Therefore it has to be copied periodically to EBS or S3 to ensure long term persistence

Price: it comes completely free of charge including I/O operations.



Definition: RDS is instance of MySQL server running on an EC2 platform. Persistent storage (for back-ups, etc.) is allocated in EBS volumes. However, neither can you access the underlying EC2 instance nor can you use S3 to access your stored database snapshots. Since you do not get access to the native EC2 instance, you cannot install additional software on the MySQL host.

Multi-AZ deployment - RDS automatically provisions and manages a “standby” replica in a different AZ. Database updates are made synchronously on the primary and standby resources to prevent replication lag. In the event of planned database maintenance, DB Instance failure, or an AZ failure, RDS automatically failovers to the up-to-date standby so that database operations can resume quickly without administrative intervention. Prior to failover you cannot directly access the standby, and it cannot be used to serve read traffic.

Read Replicas – You can create replicas of a given source DB Instance that serve high-volume application read traffic from multiple copies of your data. RDS uses MySQL's asynchronous replication to propagate changes made to a source DB Instance to any associated Read Replicas.

Price: $0.4 - $0.8 per hour

AWS Main Components (continued)

EBS

Definition: EBS provides block level storage volumes for use with EC2 instances. The volumes are network-attached, and persist independently from the life of an instance that it is attached to

Performance: HDD – 0.1 ms; network latency – 2 ms

Properties:

1. Good for short and medium term persistence
2. The performance varies with out the Provisioned IOPS and Optimized instances (not all instances)
3. Throughput is hared with other instances
4. Rely on network for its access
5. After an accidental reboot, like power outage, the content of the storage remains intact. However, the
availability could be impacted by the network overloading with multi-tenant recoveries

EBS provisions a specific level of I/O performance by choosing a Provisioned IOPS volume. EBS volumes are in one Availability Zone (AZ), and can only be attached to instances also in that same AZ. Each storage volume is automatically replicated within the same AZ. EBS can create point-in-time snapshots of volumes, which are persisted to S3.

CloudWatch shows performance metrics for EBS volumes: bandwidth, throughput, latency, and queue depth

.

Price: $0.10 per GB-month of provisioned storage; $0.10 per 1 million I/O requests

S3

Definition: S3 provides a simple web interface that can be used to store and retrieve any amount of data, at any time, from anywhere on the web (multiple AZ storage). You can write, read, and delete objects containing from 1 byte to 5 terabytes of data each. The number of objects you can store is unlimited. Each object is stored in a bucket and retrieved via a unique, developer-assigned key.

Price: $0.1 GB/month

AWS Failure Modes and Effects [1]

Failure Mode	Probability	Mitigation Plan
Application Failure	High	Automatic degraded response
AWS Region Failure	Low	Wait for the region to recover
AWS Zone Failure	Medium	Continue to run on the remaining zone
Data Storage Failure	Medium	Restore from S3 backup
S3 Failure	Low	Restore from remote archive (disaster recovery)

Zone Failure Situations

1. **Power Outage**
    1. Instances lost
    2. Ephemeral storage unavailable; readily available after power restoration
    3. EBS Storage unavailable; not readily available after power restoration
2. **Network Outage**
    1. Instances unavailable
    2. Ephemeral storage unavailable
    3. EBS Storage unavailable; could be not readily available after network restoration

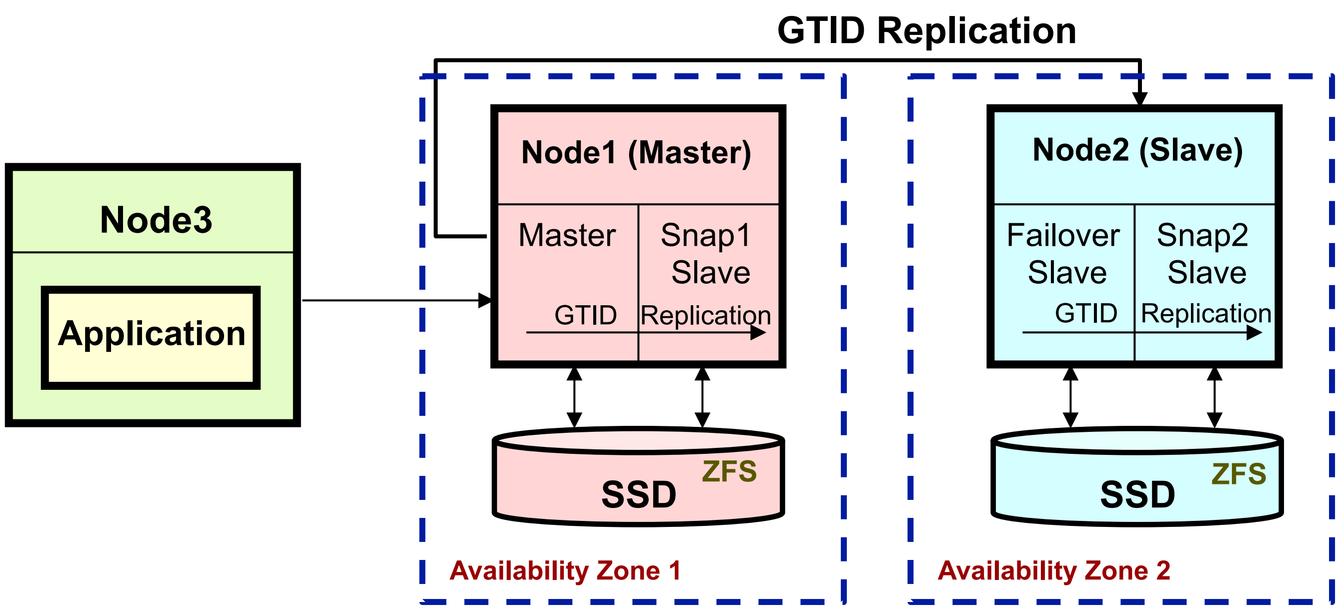
Region Failure Situations

1. "Control Plane" for creating new instances failure [2]
    1. New instances could not be created
    2. Lost control of remaining zones infrastructure
    3. EBS Storage unavailable; not readily available after power restoration
2. Network Outage
    1. Instances unavailable
    2. Ephemeral storage unavailable
    3. EBS Storage unavailable; could be not readily available after network restoration

[1] http://www.slideshare.net/adrianco/high-availability-architecture-at-netflix

[2] http://readwrite.com/2011/04/25/almost-as-galling-as-the#awesm=~ommLY1YhK9eiOz

Failover with GTID Replication and ZFS Snapshots



```
sync_binlog = 1
sync_master_info = 1
sync_relay_log = 1
innodb_support_xa = 1
master_info_repisitory=TABLE
relay_log_info_repository = TABLE
log-slave-updates =TRUE
```

- We use ZFS snapshots for the Master and Slave backups
- We must have Slave node to make ZFS snapshots
- Master server cannot stop without stopping all warehouse system

Failover cases

1) Service failures:
- Node1 ***master*** process failure - service moves to node2
- Node1 ***slave*** process failure – service restarts

2) Node failures:
- node1 failure - services move to node2
- node2 failure – restart node2

3) Network failures:
- node1 network failure - services move to node2
- node2 (slave) network failure – restart services

4) Server data corruption:
- node1 master – get the snapshot from the snap slave
- node2 master – get the snapshot from the snap slave

High-Level Block Diagram of the Demo Script

Start main script

1. Verify initial conditions
2. Destroy snapshot if exists
3. Stop and cleanup the servers: slave, snap and master

4. Start the servers: slave, snap and master
5. Setup GTID replication from master to snap
6. Setup GTID replication from master to main slave
7. Initialized and start local application

8. Prepare to take snapshot of the snap server
9. Flush tables with read lock
10. Take snapshot
11. Unlock tables on the snap server

12. Crash master
13. Crash application
14. Promote main slave to new master
15. Restart application and point it to the new master

16. Destroy local volume
17. Send snapshot to the master to replace the original directory
18. Delete old UUID from old master directory

19. Demote old master to the new main slave
20. Setup replication from new master to new slave

21. Continue to run and finish application

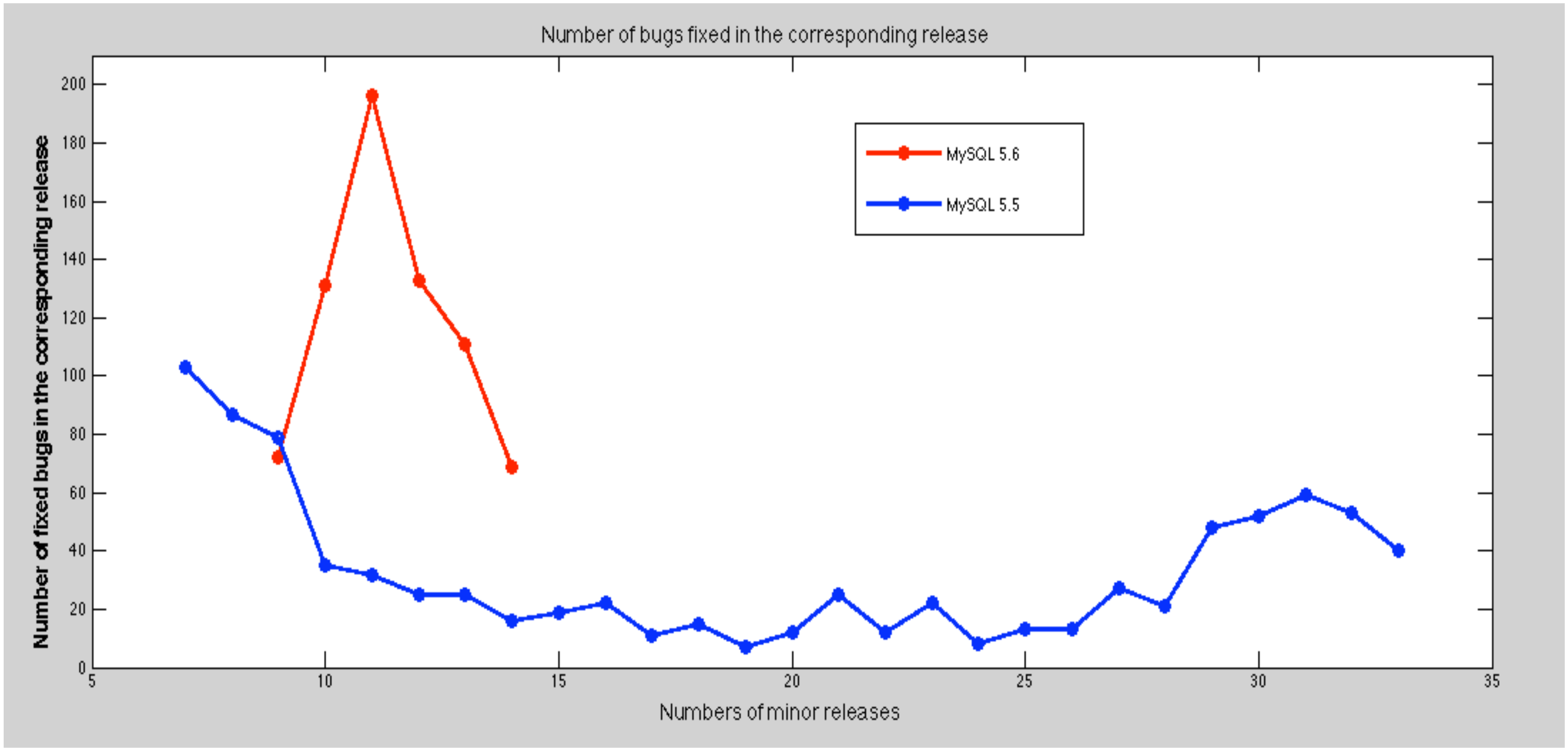
Stop main script

Summary

- MySQL replication is the most popular High Availability solution
- To increase server availability MySQL team introduced new features
  - Global Transaction Identifiers (GTIDs)
  - Server UUIDs
  - Crush Safe Slaves and Binary Logs
  - Replication Events Checksum
- New features increased availability and allowed automation of Failover procedure

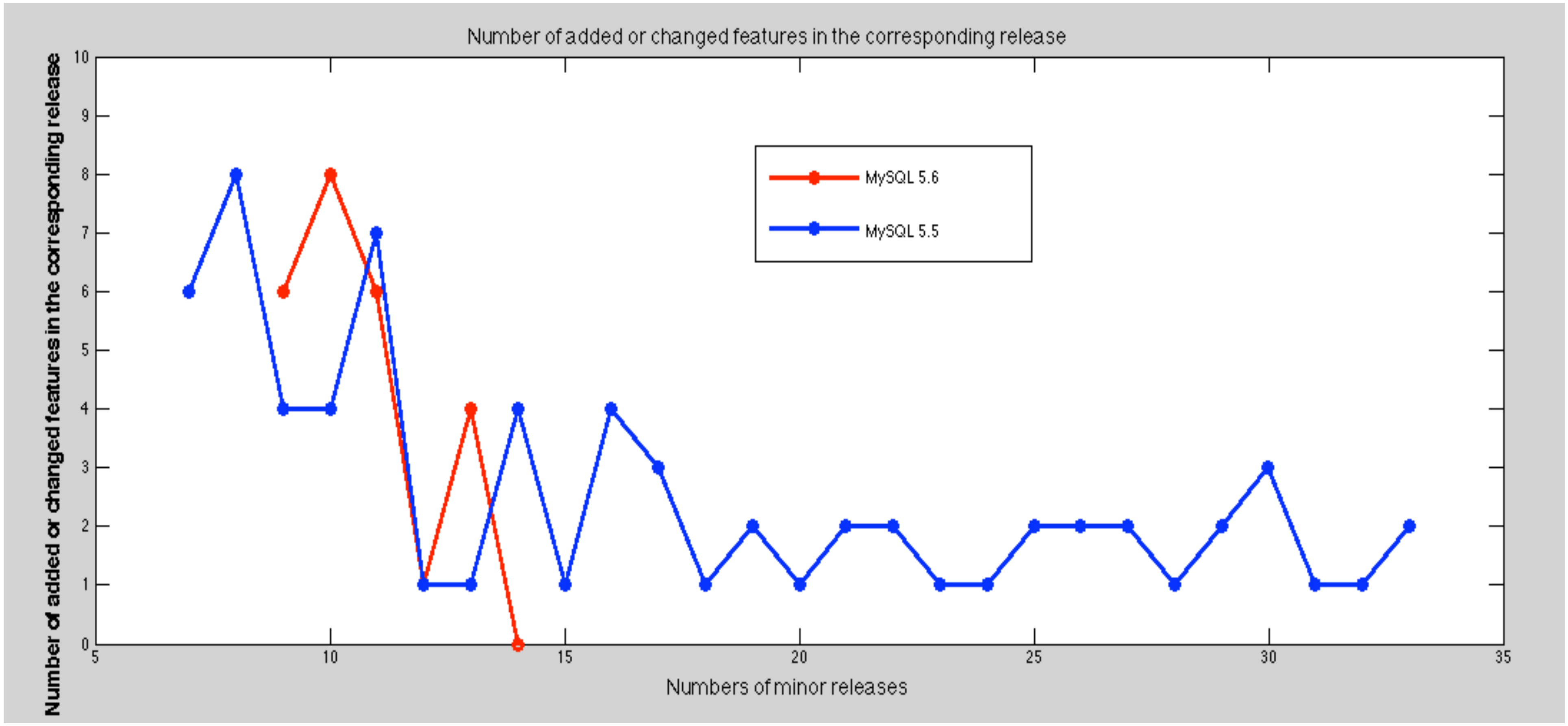
Backup

Number of fixed bugs in MySQL 5.5 and 5.6 releases



- First production release for MySQL 5.6.10 had 40% more bugs than first production release of MySQL 5.5.9
- The number of bugs for subsequent release of MySQL 5.6 was significantly higher than for production release. In case of 5.5 the situation was different
- The number of bugs in 5.6. is still significantly higher than for similar situation with 5.5

Number of new/changed features in releases

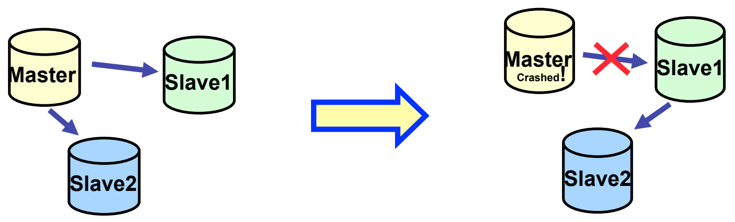


The number of improvements for subsequent release of MySQL 5.6 was very similar to MySQL 5.5 subsequent releases

Replication Binary Log Coordinates

```
mysql> SHOW SLAVE STATUS\G
*************************** 1. row ***************************
               Slave_IO_State: Waiting for master to send event
                  Master_Host: 127.0.0.1
                  Master_User: msandbox
                  Master_Port: 26768
                Connect_Retry: 60
              Master_Log_File: mysql-bin.000001 (IO Thread reads this file)
          Read_Master_Log_Pos: 4723 (Position in master binary log file where IO Thread has read to)
               Relay_Log_File: mysqld-relay-bin.000001
                Relay_Log_Pos: 874  (Position in the relay log file where SQL thread read and executed events
        Relay_Master_Log_File: mysql-relay-bin.000001
             Slave_IO_Running: Yes
            Slave_SQL_Running: Yes
. . . . . . . . . . . . . . . . .:  . . . . . .
Last_Errno: 0 Last_Error:
                 Skip_Counter: 0
          Exec_Master_Log_Pos: 729  (Position in master binary log file that SQL Thread read and executed up to
              Relay_Log_Space: 1042 The total combined size of all existing relay log files
              Until_Condition: None
               Until_Log_File:
                Until_Log_Pos: 0
           Master_SSL_Allowed: No
. . . . . . . . . . . . . . . . .: . . .
Seconds_Behind_Master: 0
Master_SSL_Verify_Server_Cert: No
                Last_IO_Errno: 0
                Last_IO_Error:
               Last_SQL_Errno: 0
               Last_SQL_Error:
1 row in set (0.00 sec)
```

Failover



Coordinates usage examples:

Connect to the master using master's binary log
slave> CHANGE MASTER TO
MASTER_HOST='12.34.56.789',MASTER_USER='slave_user',
MASTER_PASSWORD='password',
MASTER_LOG_FILE='**mysql-bin.000001**', MASTER_LOG_POS=4723;

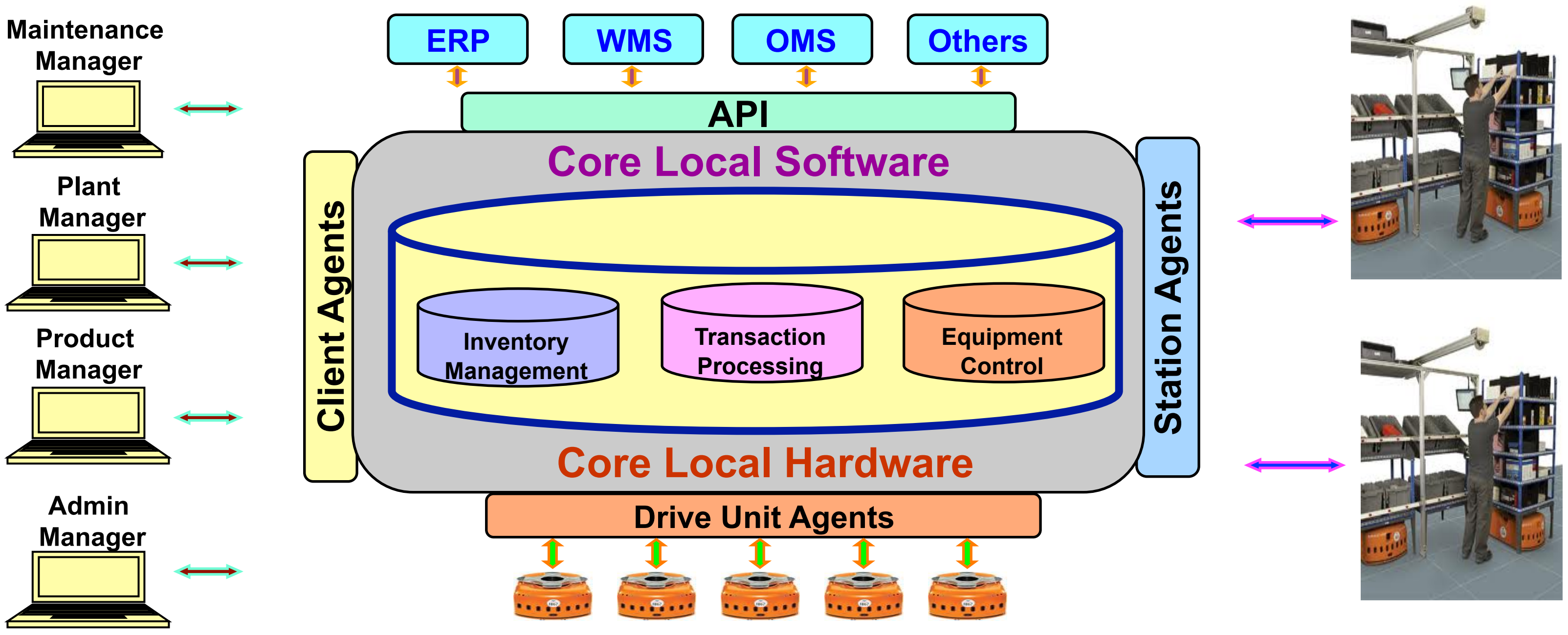
Connect to the new master/old slave using slave's relay log
slave> CHANGE MASTER TO
MASTER_HOST='12.34.56.789',MASTER_USER='slave_user',
MASTER_PASSWORD='password',
MASTER_LOG_FILE='**mysql-relay-bin.000001**', MASTER_LOG_POS=729;

What is the Kiva Mobile-Robotic Fulfillment System?

Internet Order Fulfillment Operation is the core of Amazon business

Kiva uses hundreds of mobile robotic drive units to bring inventory on mobile shelves directly to workers, allowing access to all inventory items at all times

Kiva software is integrated with the client's enterprise systems, including: warehouse management systems (WMS), order management systems (OMS), and enterprise resource planning systems (ERP)

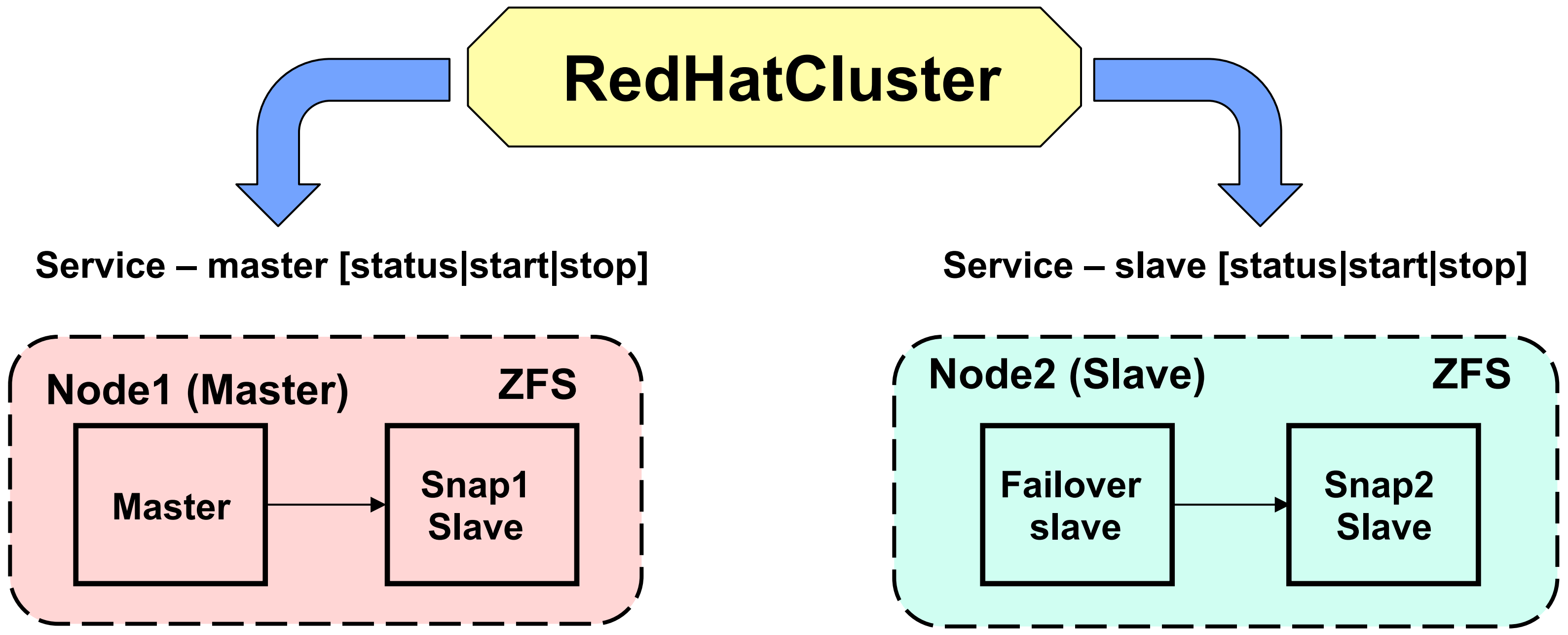


- **Current Kiva software runs locally**
- **Database server HA provided by the local SAN storage and RedHat Cluster**

How to provide Database Server High Availability when Kiva software and hardware run in the Cloud?

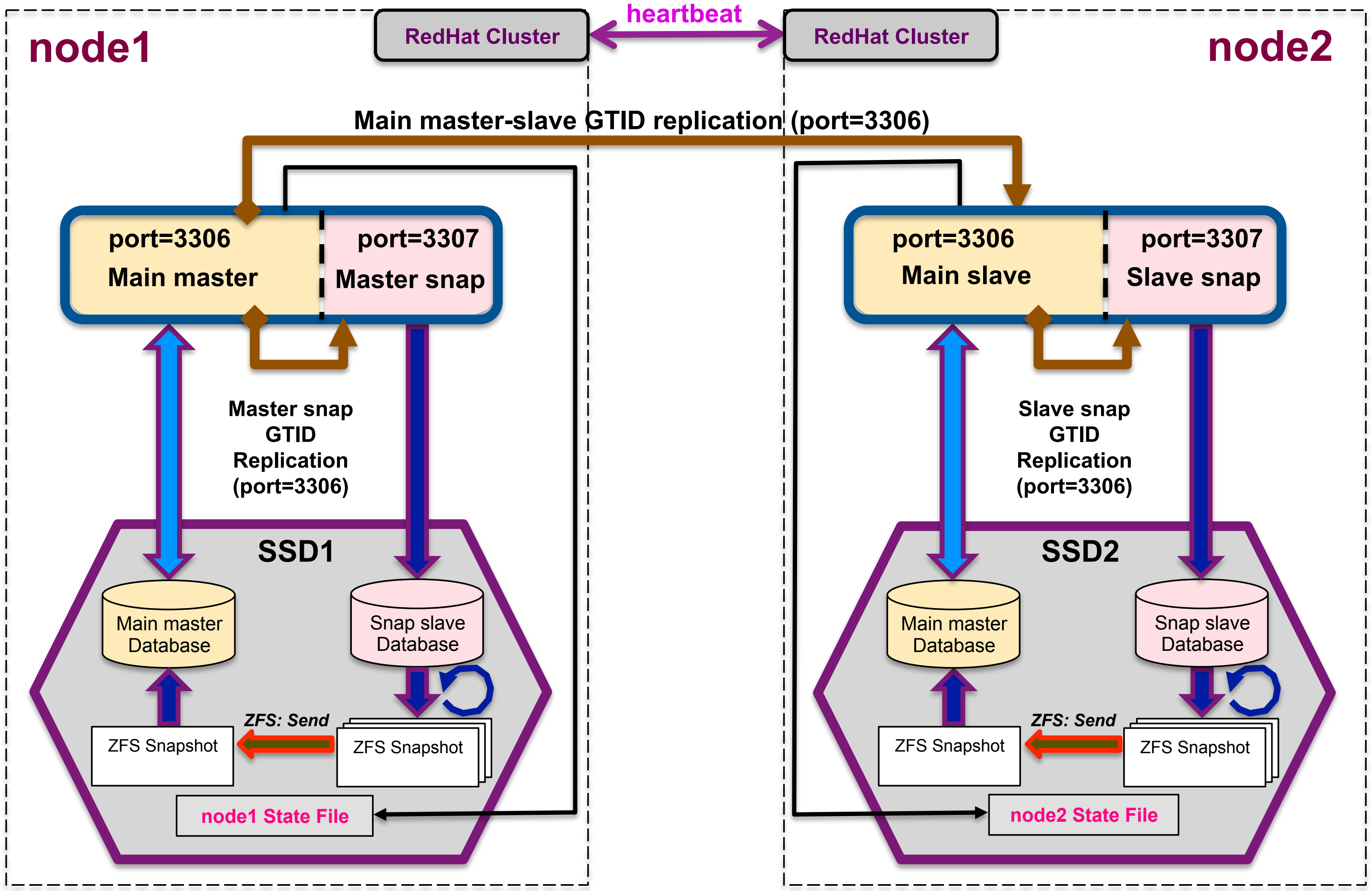
RedHat Cluster and Master/Slave Nodes

RedHat Cluster monitors Master and Slave processes on Master and Slave nodes

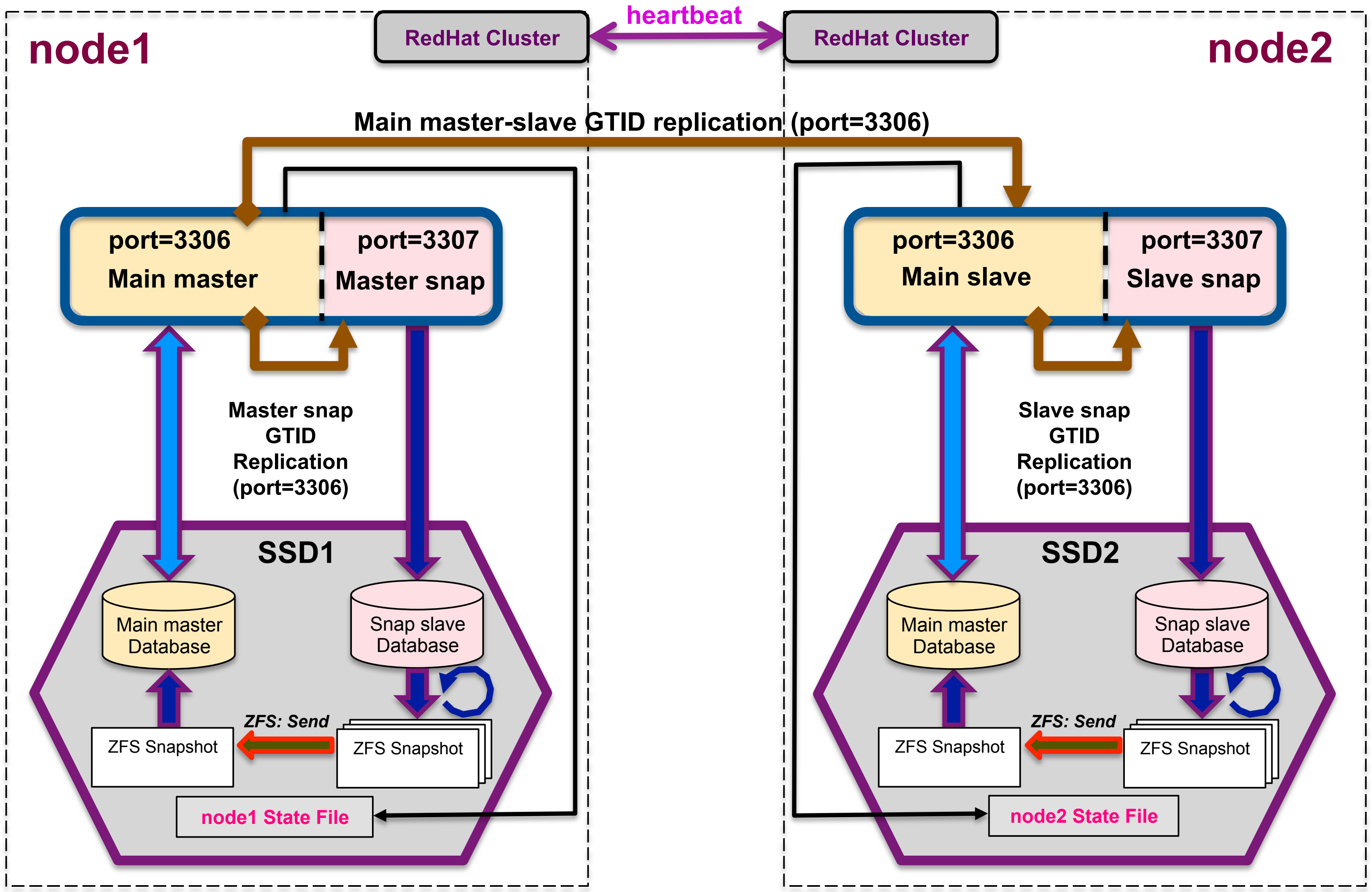


- We use ZFS snapshots for the Master and Slave backups
- We must have Slave node to make ZFS snapshots
- Master server cannot stop without stopping all warehouse system

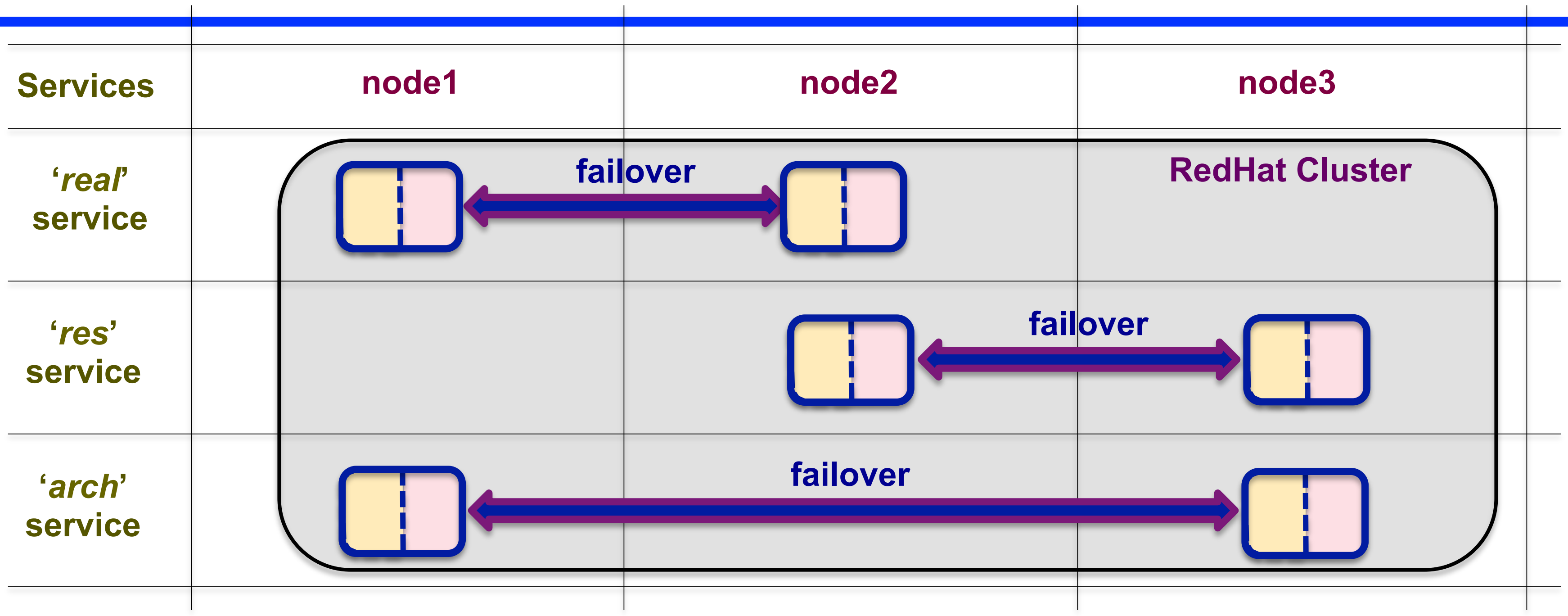
Cloud HA Solution with GTID Replication



Cloud HA Solution with GTID Replication



Services, Nodes, Network and Corruption Failover Scenarios



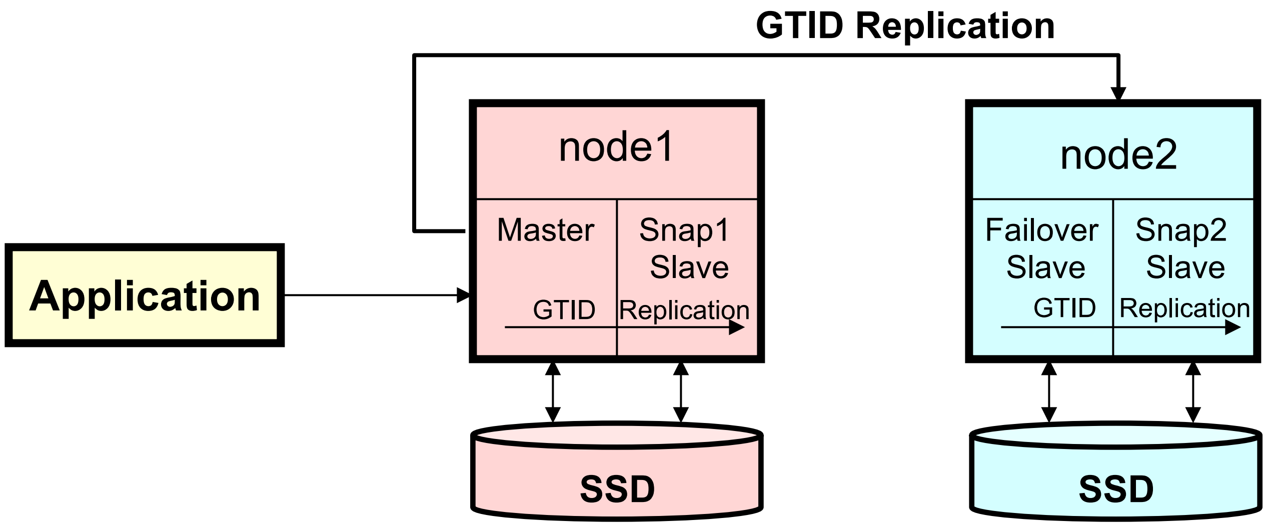
1) Service failures:
- node1 ***real*** service failure - service moves to node2
- node2 ***res*** service failure - service moves to node3
- node1 ***arch*** service failure – service moves to node3
- node3 (slave) service failure – service restarts
- node3 (master) service failure – service moves to node2 or node1

2) Node failures:
- node1 failure - services move to node2 and node3
- node2 failure - services move to node3
- node3 (slave) failure – restart node3
- node3 (master) failure – services move to node1 or node2

3) Network failures:
- node1 network failure - services move to node2 and node3
- node2 network failure - services move to node3
- node3 (slave) network failure – restart services
- node3 (master) network failure – services move to node2 or node3

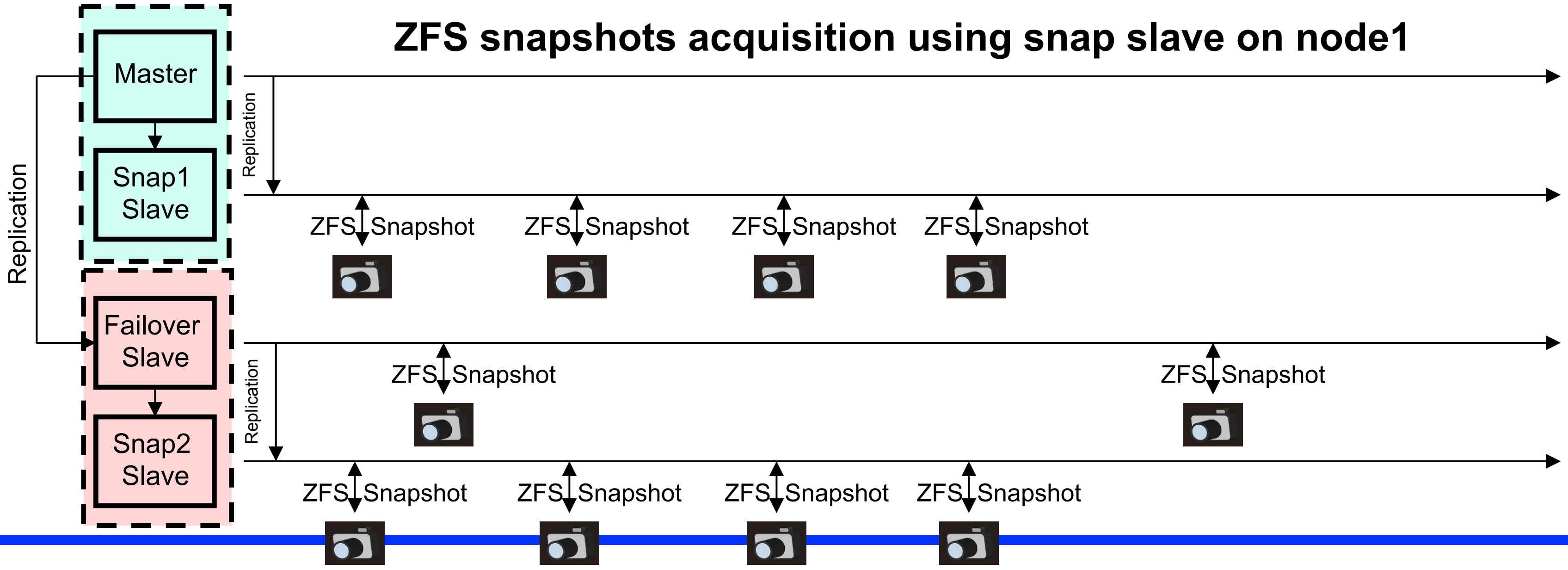
4) Server data corruption:
- node1 master – get the snapshot from the snap slave
- node2 master – get the snapshot from the snap slave
- node3 master – get the snapshot from the snap slave

General Architecture of the System Failover

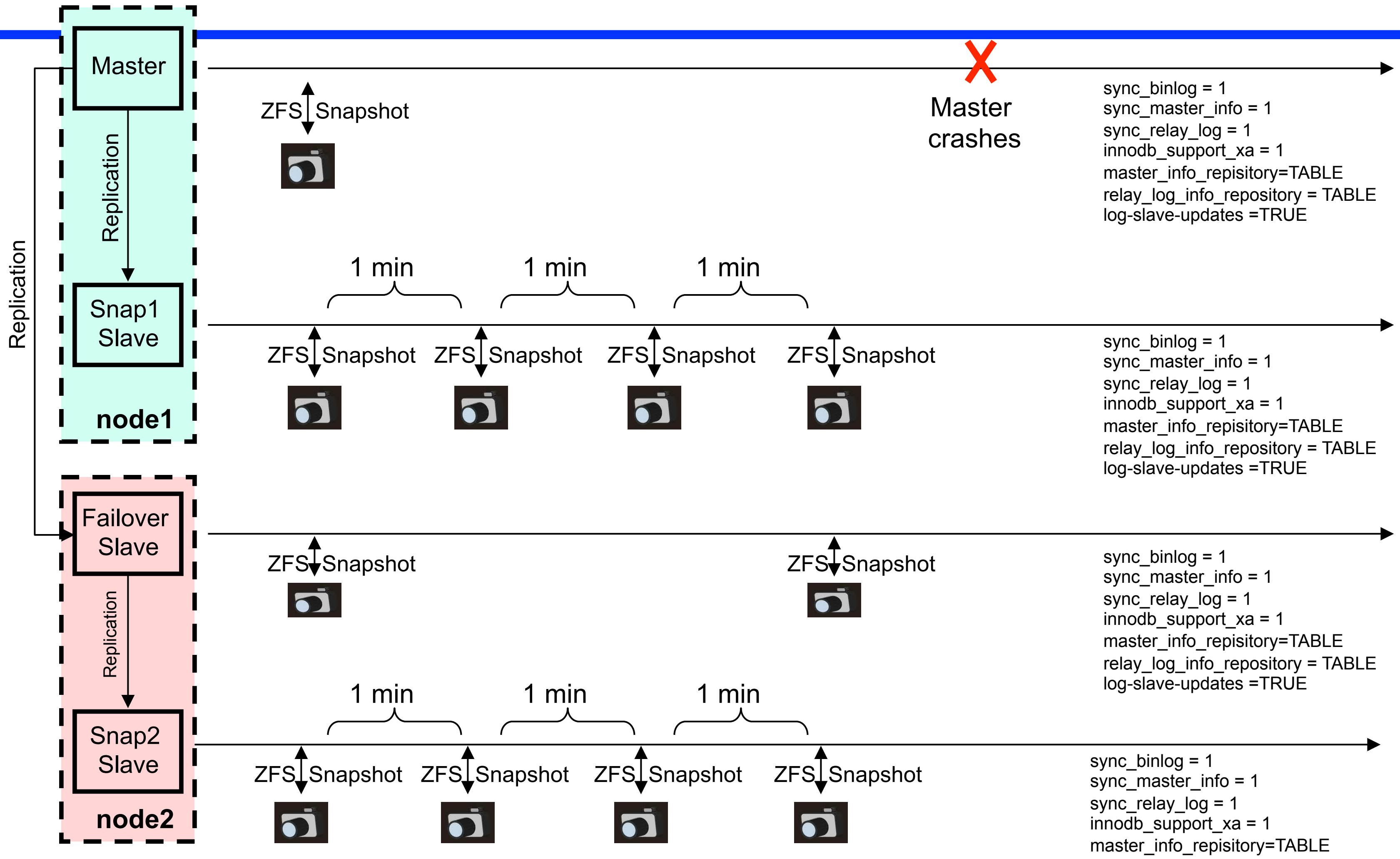


sync_binlog = 1
sync_master_info = 1
sync_relay_log = 1
innodb_support_xa = 1
master_info_repisitory=TABLE
relay_log_info_repository = TABLE
log-slave-updates =TRUE

ZFS snapshots acquisition using snap slave on node1



Master (node1) works with snap slave(node1) and failover slave(node2)



```
sync_binlog = 1
sync_master_info = 1
sync_relay_log = 1
innodb_support_xa = 1
master_info_repisitory=TABLE
relay_log_info_repository = TABLE
log-slave-updates =TRUE
```

```
sync_binlog = 1
sync_master_info = 1
sync_relay_log = 1
innodb_support_xa = 1
master_info_repisitory=TABLE
relay_log_info_repository = TABLE
log-slave-updates =TRUE
```

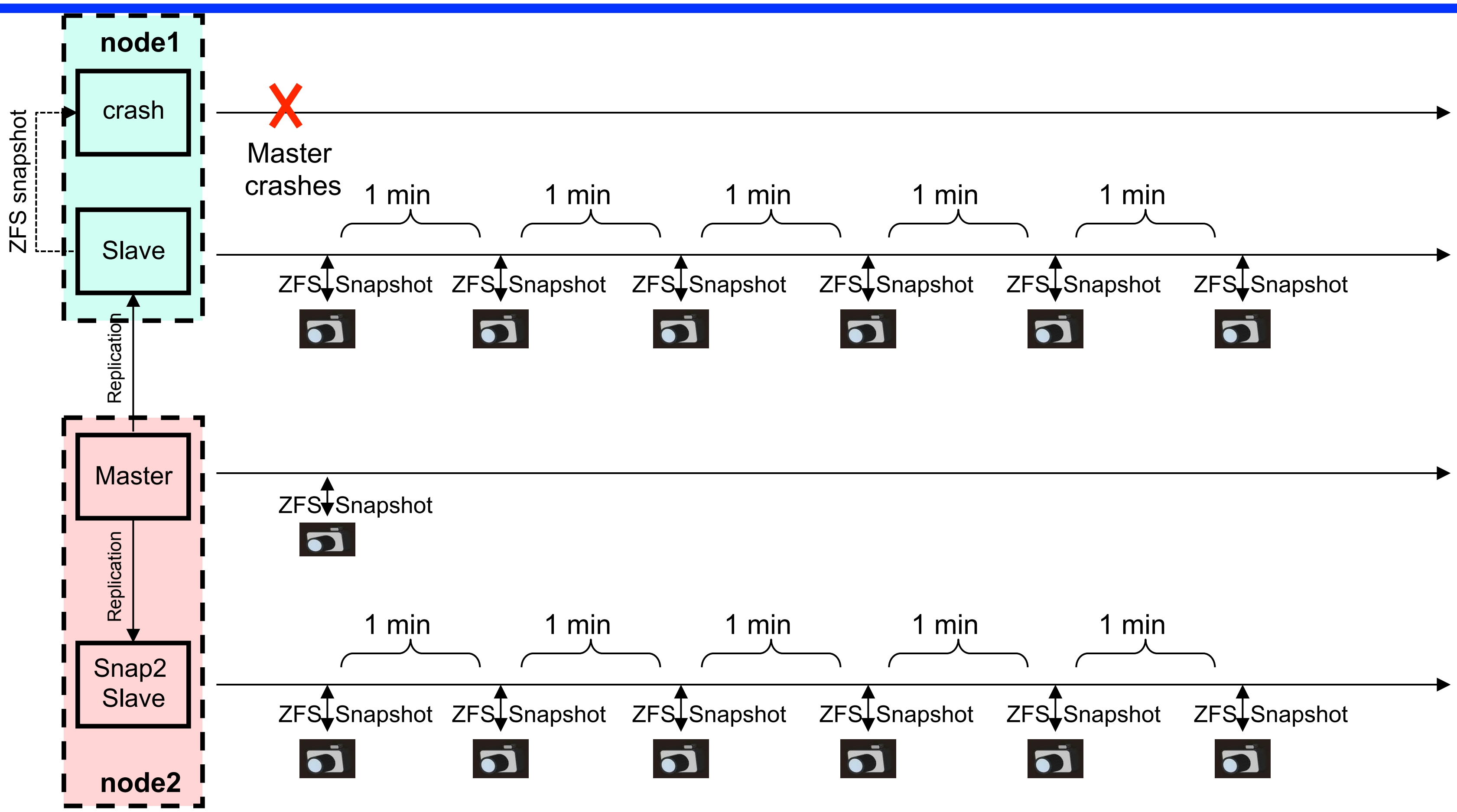
```
sync_binlog = 1
sync_master_info = 1
sync_relay_log = 1
innodb_support_xa = 1
master_info_repisitory=TABLE
relay_log_info_repository = TABLE
log-slave-updates =TRUE
```

```
sync_binlog = 1
sync_master_info = 1
sync_relay_log = 1
innodb_support_xa = 1
master_info_repisitory=TABLE
relay_log_info_repository = TABLE
log-slave-updates =TRUE
```

Master works with snap slave and failover slave:

1. Snap1 slave and failover slave replicate from the master
2. Snap1 slave takes ZFS snapshots every minute
3. Failover slave has ZFS snapshots every few hours
4. Master has ZFS snapshot every few days
5. Snap2 slave takes ZFS snapshots every minute (symmetrical to snap1)

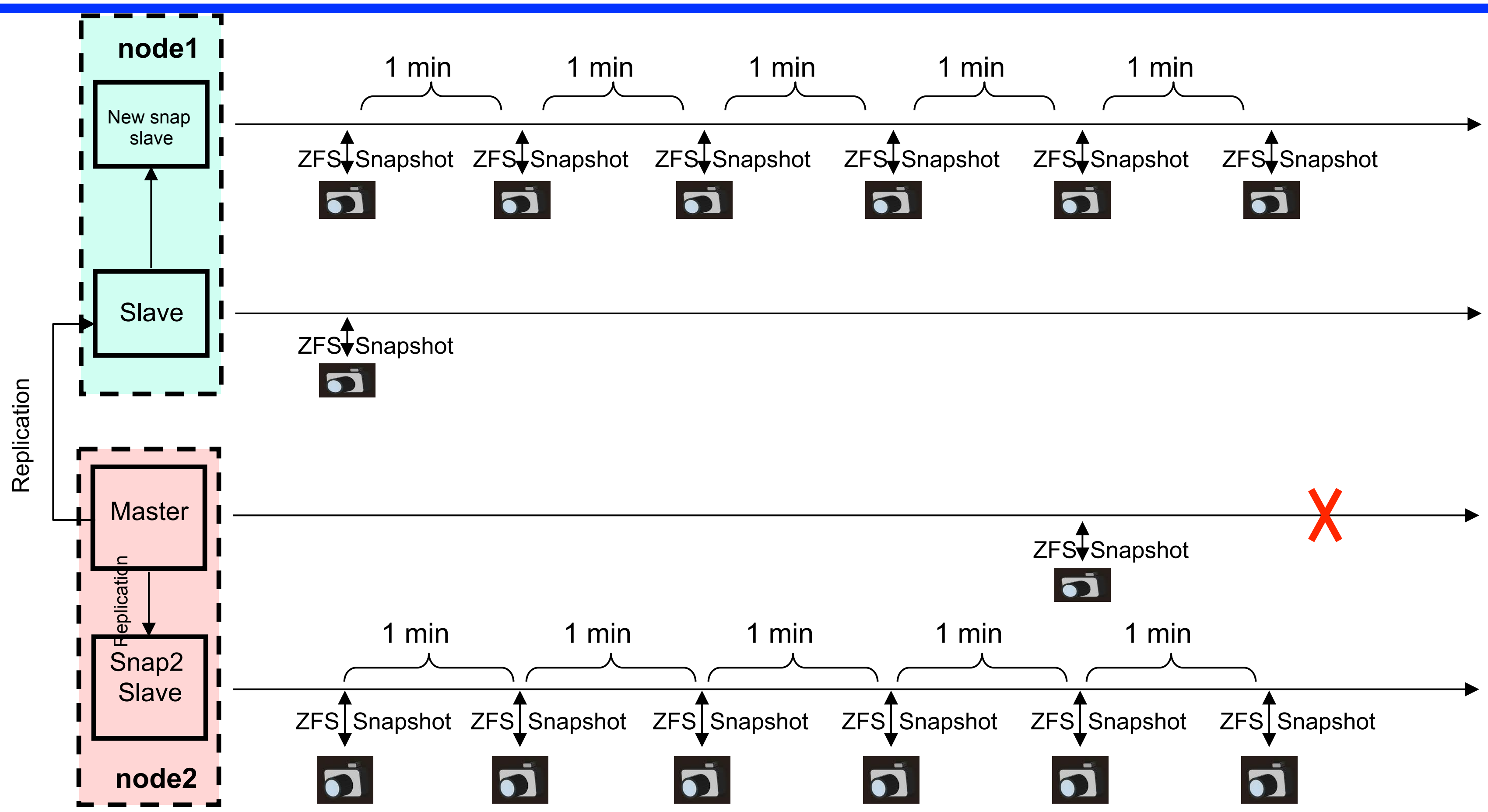
Recovery after master (node1) failure: node1 continues to work, failover slave works as master



Failover steps in case of the master crash:

1. Master crashes, but the node1 continues to work
2. Application redirected to node2 and failover slave becomes new master
3. The latest ZFS snapshot replaces old master data directory
4. Snap1 slave replicates from node2 catching up with new master
5. Snap2 slave continues to get replicated data from new master

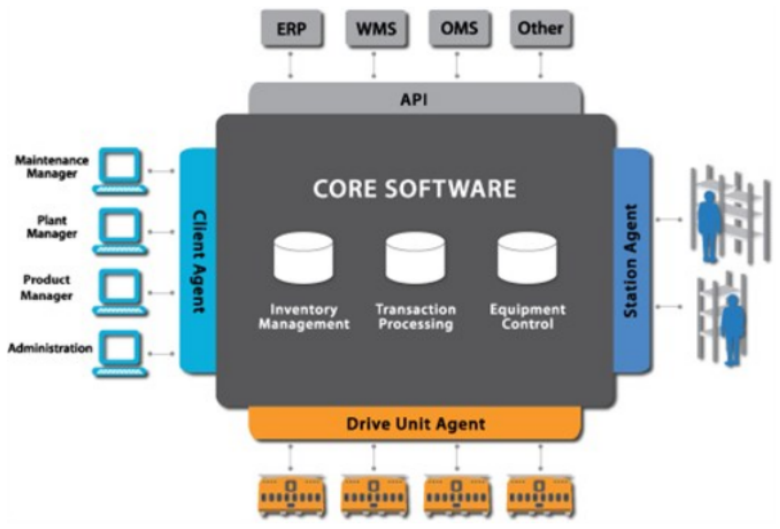
Recovery after master (node1) failure, node1 continues to work, failover slave works as master



Recovery steps in case of the master crash:

1. Old master recovers, and catches up with new master using data from old snap1 slave
2. Old snap1 slave becomes new failover slave
3. Old master becomes new snap1 slave making frequent ZFS snapshots
4. In case of new master crash new failover slave becomes new master
5. node2 supposed to have the same architecture as node1 with node2 snap slave

Backup


ERP
WMS
OMS
Other
API
CORE SOFTWARE
Inventory Management
Transaction Processing
Equipment Control
Client Agent
Station Agent
Drive Unit Agent
Maintenance Manager
Plant Manager
Product Manager
Administration

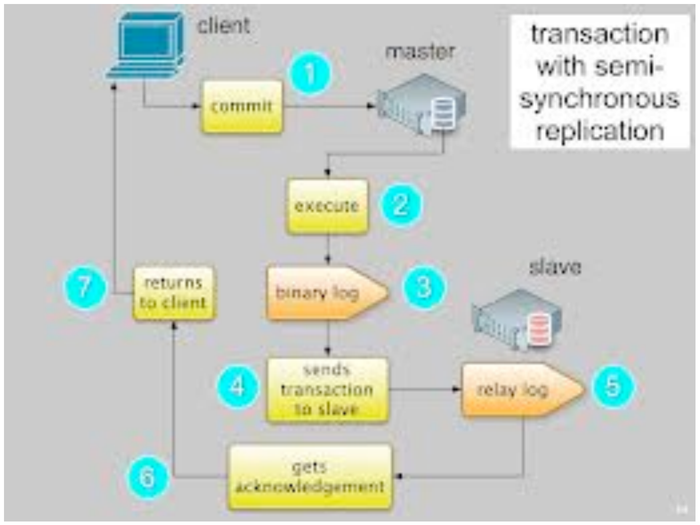
Backup

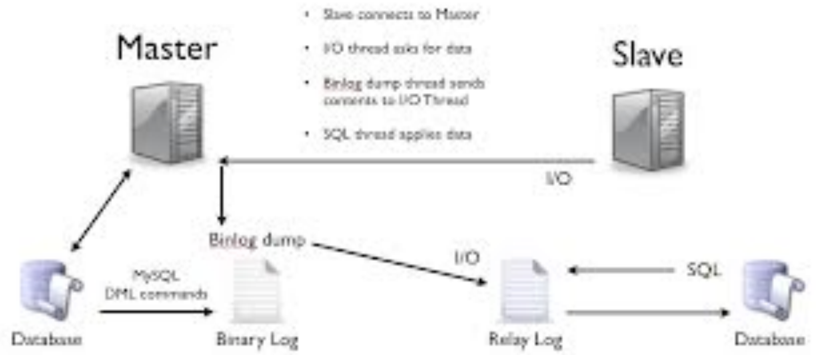
Replication Data Files

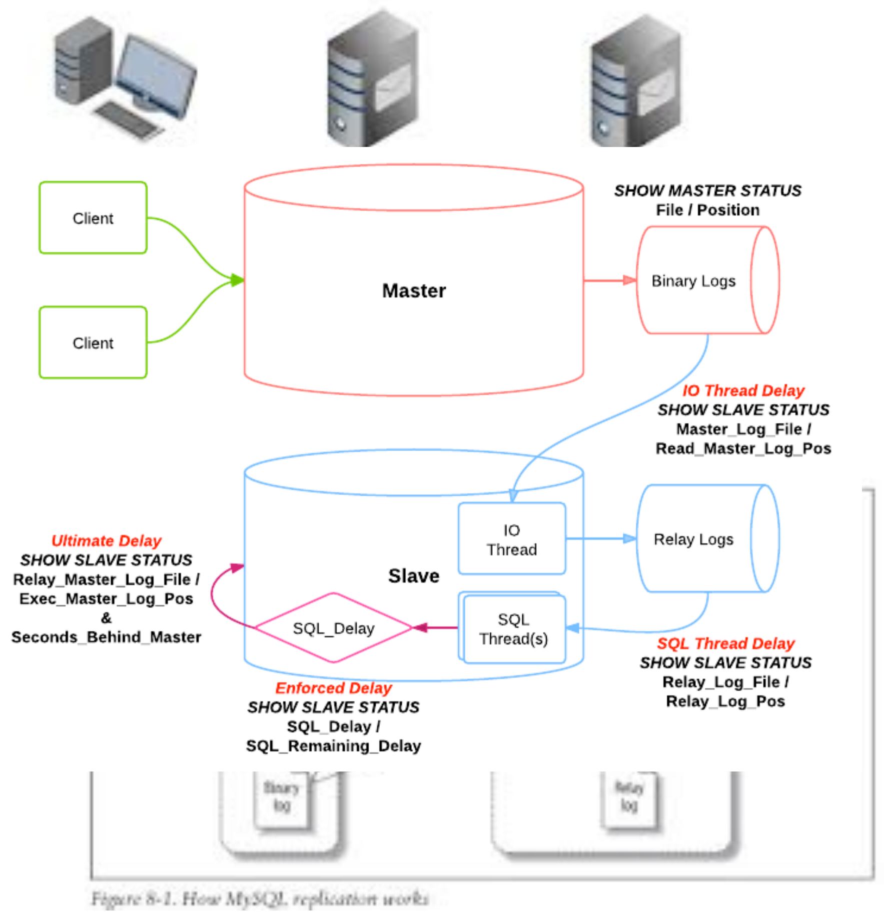
Line #	SHOW SLAVE STATUS Column	Description
1		Number of lines in the file
2	Master_Log_File	The name of the master binary log currently being read from the master
3	Read_Master_Log_Pos	The current position within the master binary log that have been read from the master
4	Master_Host	The host name of the master
5	Master_User	The user name used to connect to the master
6	Password (not shown by SHOW SLAVE STATUS)	The password used to connect to the master
7	Master_Port	The network port used to connect to the master
8	Connect_Retry	The period (in seconds) that the slave will wait before trying to reconnect to the master
9	Master_SSL_Allowed	Indicates whether the server supports SSL connections
10	Master_SSL_CA_File	The file used for the Certificate Authority (CA) certificate
11	Master_SSL_CA_Path	The path to the Certificate Authority (CA) certificates
12	Master_SSL_Cert	The name of the SSL certificate file
13	Master_SSL_Cipher	The list of possible ciphers used in the handshake for the SSL connection
14	Master_SSL_Key	The name of the SSL key file
15	Master_SSL_Verify_Server_Cert	Whether to verify the server certificate
17	Replicate_Ignore_Server_Ids	The number of server IDs to be ignored, followed by the actual server IDs

Line #	SHOW SLAVE STATUS Column	Description
1	Relay_Log_File	The name of the current relay log file
2	Relay_Log_Pos	The current position within the relay log file; events up to this position have been executed on the slave database
3	Relay_Master_Log_File	The name of the master binary log file from which the events in the relay log file were read
4	Exec_Master_Log_Pos	The equivalent position within the master's binary log file of events that have already been executed

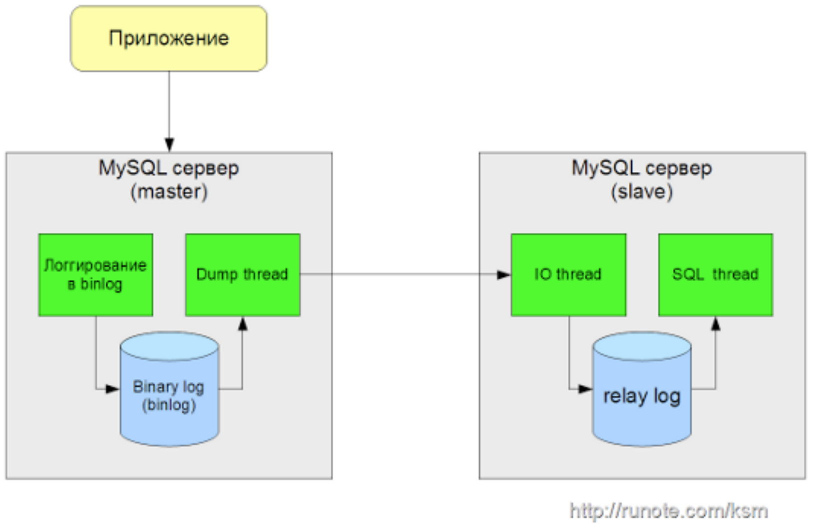
How Replication Worked Before 5.6?

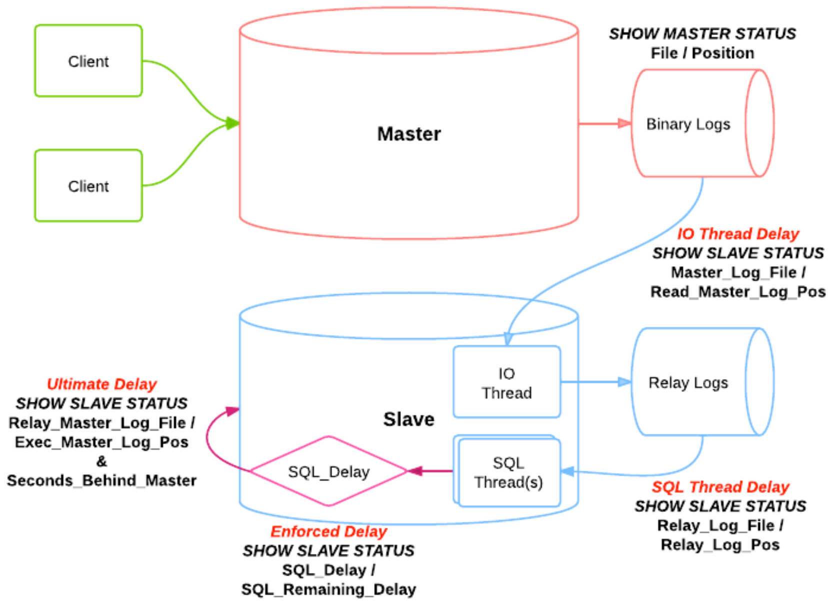






How Replication Worked Before 5.6?


Приложение
MySQL сервер (master)
Логгирование в binlog
Dump thread
Binary log (binlog)
MySQL сервер (slave)
IO thread
SQL thread
relay log
http://runote.com/ksm


Client
Client
Master
SHOW MASTER STATUS
File / Position
Binary Logs
IO Thread Delay
SHOW SLAVE STATUS
Master_Log_File /
Read_Master_Log_Pos
Slave
IO Thread
Relay Logs
Ultimate Delay
SHOW SLAVE STATUS
Relay_Master_Log_File /
Exec_Master_Log_Pos
&
Seconds_Behind_Master
SQL_Delay
SQL Thread(s)
SQL Thread Delay
SHOW SLAVE STATUS
Relay_Log_File /
Relay_Log_Pos
Enforced Delay
SHOW SLAVE STATUS
SQL_Delay /
SQL_Remaining_Delay