High Availability Solution for Large Scale Database Systems

GUOWEI ZENG

Baidu DBA

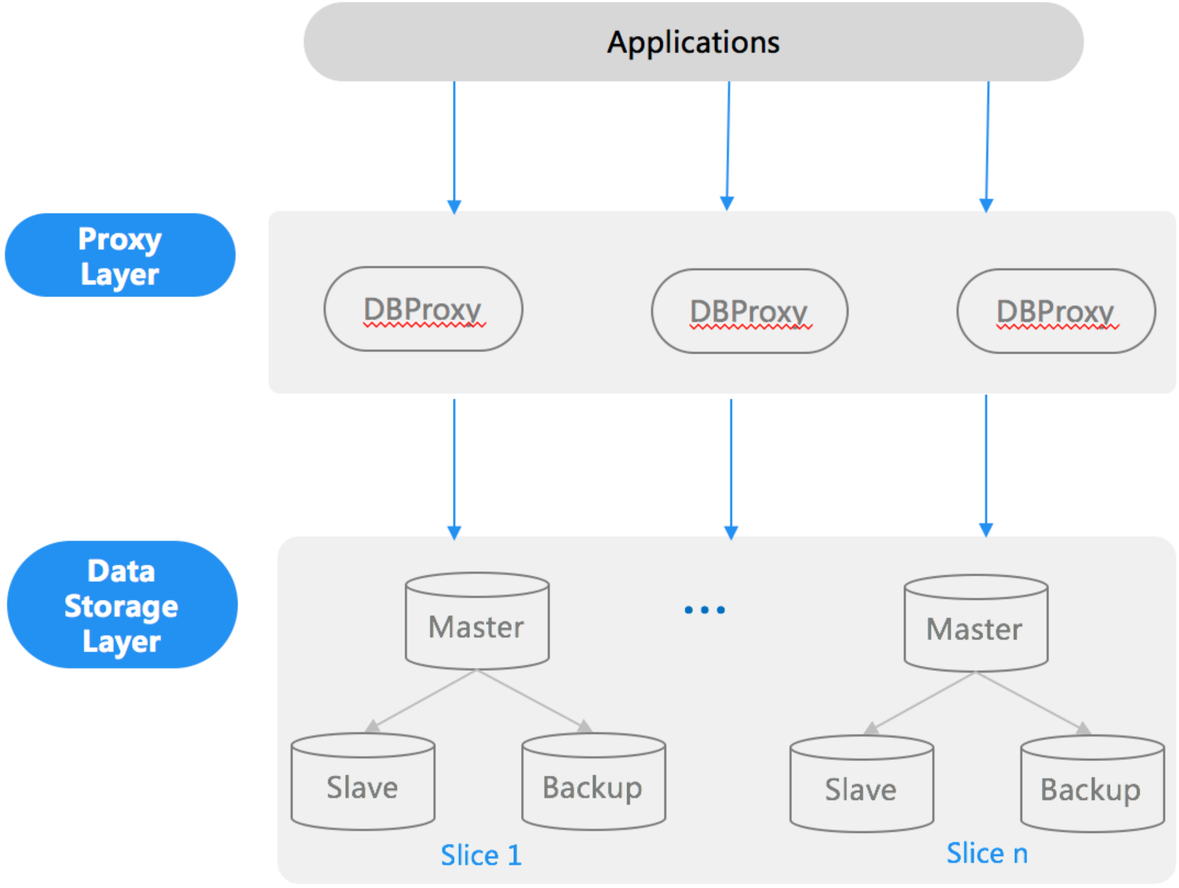
Agenda

- MySQL at Baidu, and the HA troubles
- Current HA solutions, and problems
- Baidu HA solution
- Benefits and experiences for applications

MySQL at Baidu：the main OLTP Database Services

- **Cover 95% of OLTP businesses**
  - 1,000+ clusters
  - 2,800+ slices (M-S)
  - 13,000+ MySQLs
  - PB-scale data size
  - 100+ billion queries per day
  - Clouds: public, private, hybrid

- **Baidu MySQL Database Architecture: based on Proxy**



Troubles: how to guarantee the availability efficiently

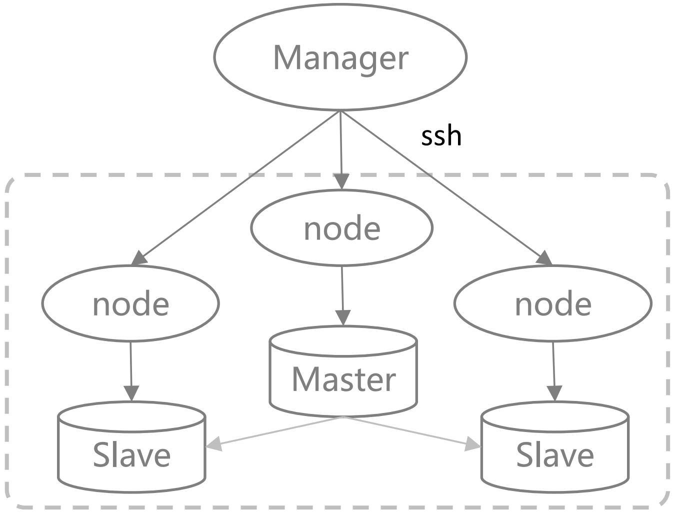
- **Stateful services: async replication**
  - Single-node write
  - Data consistency for OLTP
- **What we met in Baidu?**
  - Mass clusters: core businesses
  - Disaster tolerance: machine, network, etc.
  - Multi-version: 5.0~5.7

Automation is inevitable

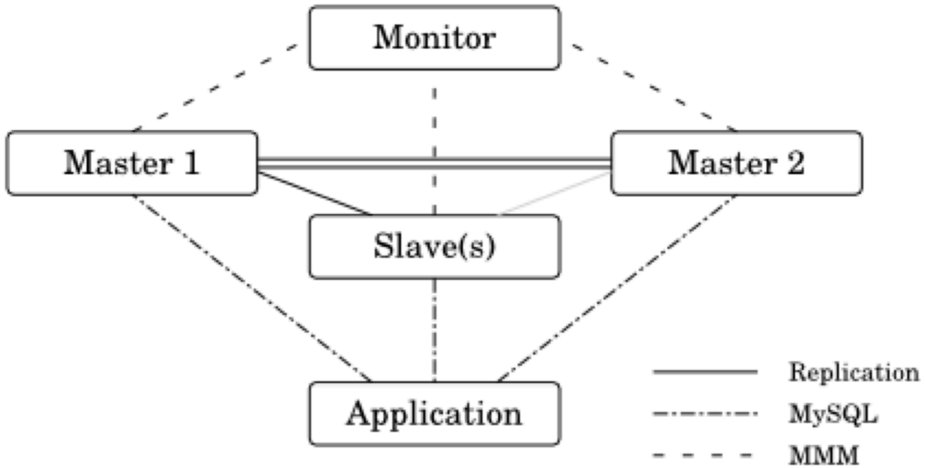
- **Manual HA**
  - Skill level: practice regularly
  - Occasion: 3am alert
  - Concurrency
- **HA focus on**
  - High concurrency
  - Failure detection: accurate
  - Recovery: data consistency

Current HA Solutions and Problems

MHA: MySQL Master High Availability



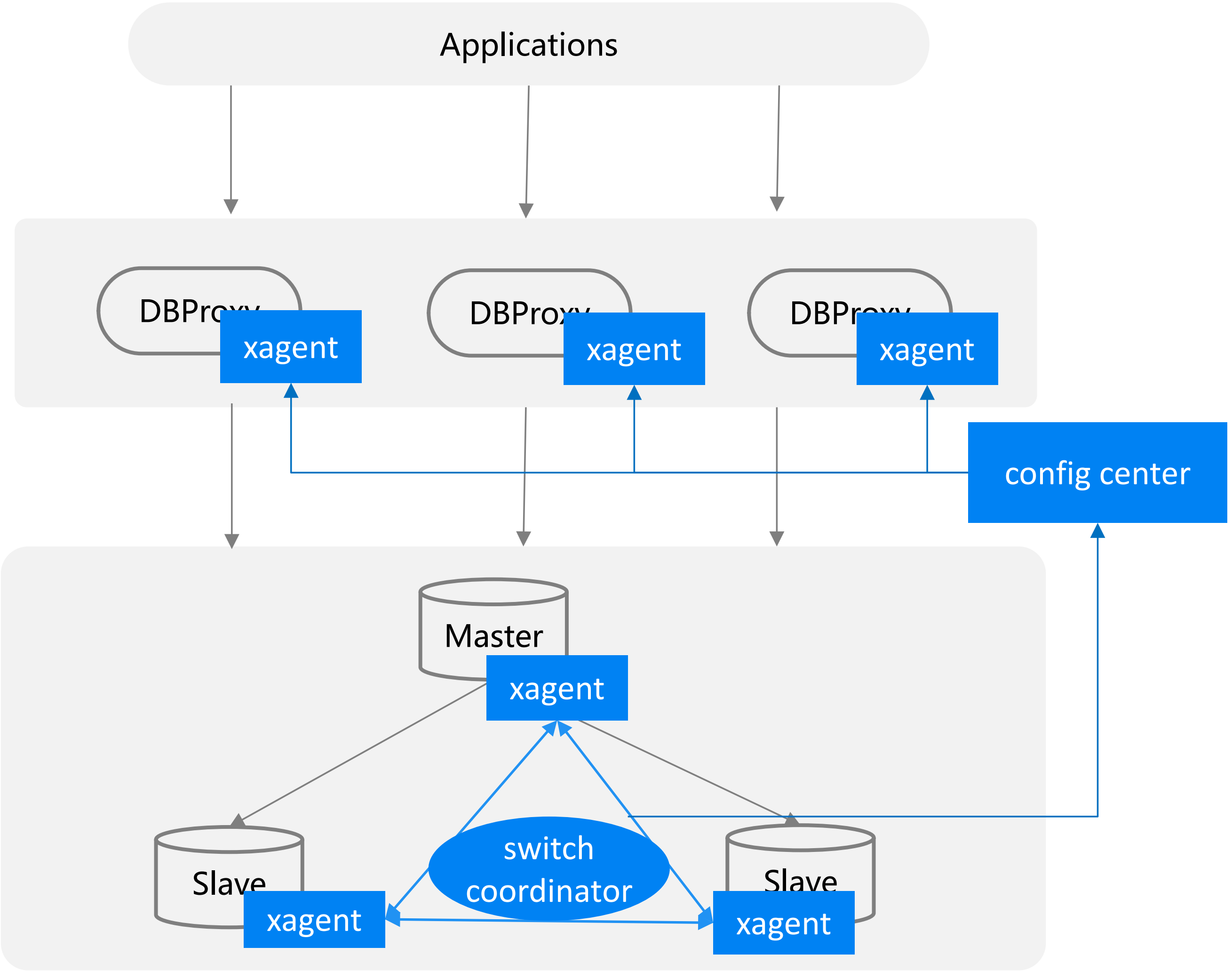
MMM: Multi-Master replication manager



- **Architecture**
  - Centralized: concurrency, not support data center(DC) failure
  - Requirements: trust building for 10000+ machines? M-M
- **Failure detections**
  - False positives: overloads
  - False negative: freezing, hardware
- **Recovery:** data consistency
  - Error on some version
  - Poor performance on some cases

Baidu HA Solution: Architecture

- **Decentralization**
  - **XAgent**
    - swithcover coordinator
    - monitor
    - operating
  - **Configure center**
    - topology storage
    - push/pull mode
- **Disaster tolerance**
  - Multi-levels: machine, DC, region
- **Scalability**
  - 5000 MySQLs
  - easily deploy



Baidu HA Solution: Failure Detection

- **Current Solution**
  - Ping/ssh (MHA)
  - Agent: write or read queries (MMM)
- **Problems**
  - Misjudgments

Case1: False positives

Overload

ping ✔
conn ✔
queries ✘

ping ✔
conn ✘

Case2: False negatives:

Disk failure

ping ✔
conn ✔
read ✔

ping ✔
conn ✔
write ✘

ping ✔
conn ✘

Machine freezing

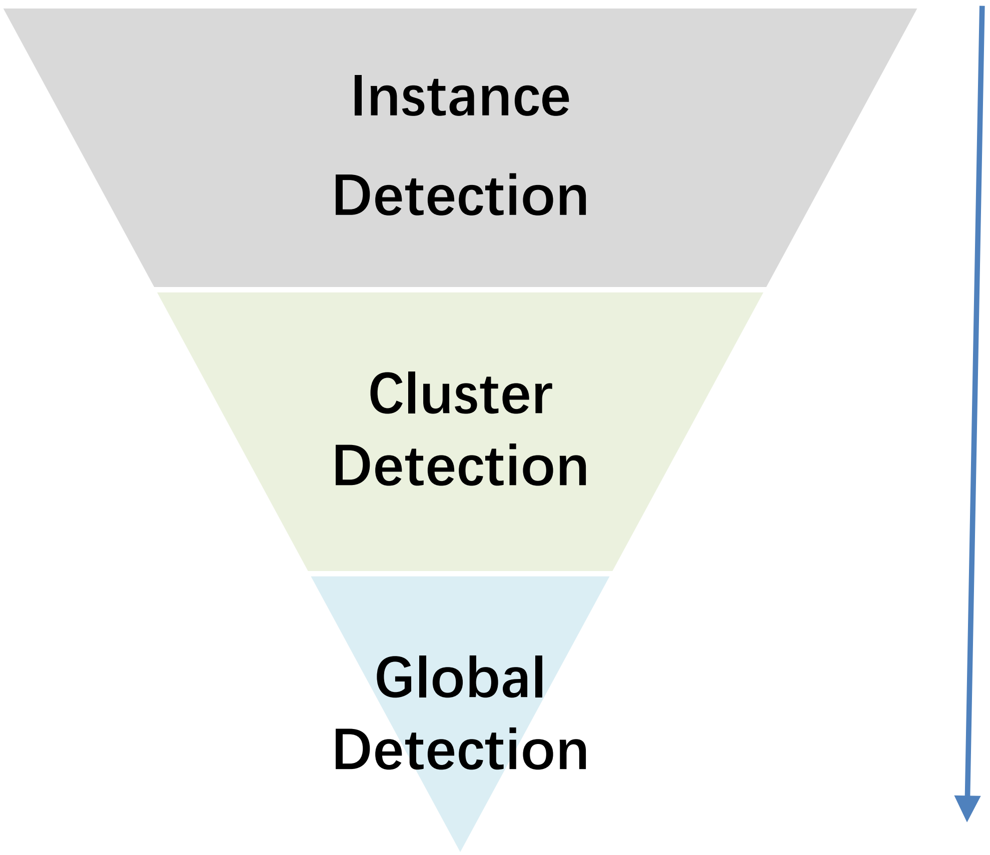
ping ✔
conn ✘
*slave status ✔

Master Failure Detection: 3-Layers Strategy

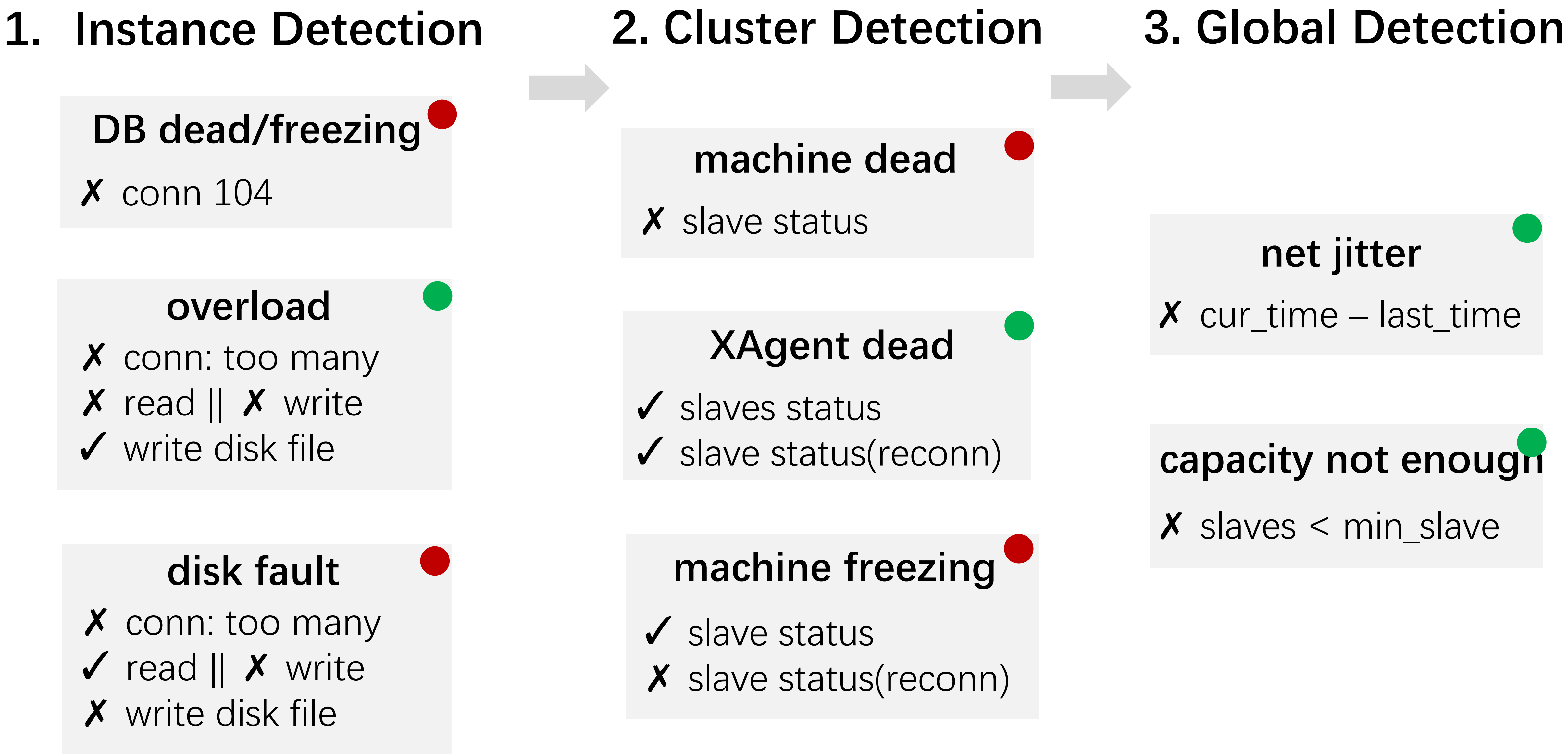
Classify of failures

ITEM	SUBITEMS	SWITCH
Instance Failure	dead	yes
	freezing	yes
	repeated dead	no
	overload	no
Machine Failure	disk failure	yes
	dead	yes
	freezing	yes
Network Failure	dead	yes
	jitter	no

Detection Pyramid



Master Failure Detection: Cases



1. Instance Detection

DB dead/freezing 🔴

✗ conn 104

overload 🟢

✗ conn: too many
✗ read || ✗ write
✓ write disk file

disk fault 🔴

✗ conn: too many
✓ read || ✗ write
✗ write disk file

2. Cluster Detection

machine dead 🔴

✗ slave status

XAgent dead 🟢

✓ slaves status
✓ slave status(reconn)

machine freezing 🔴

✓ slave status
✗ slave status(reconn)

3. Global Detection

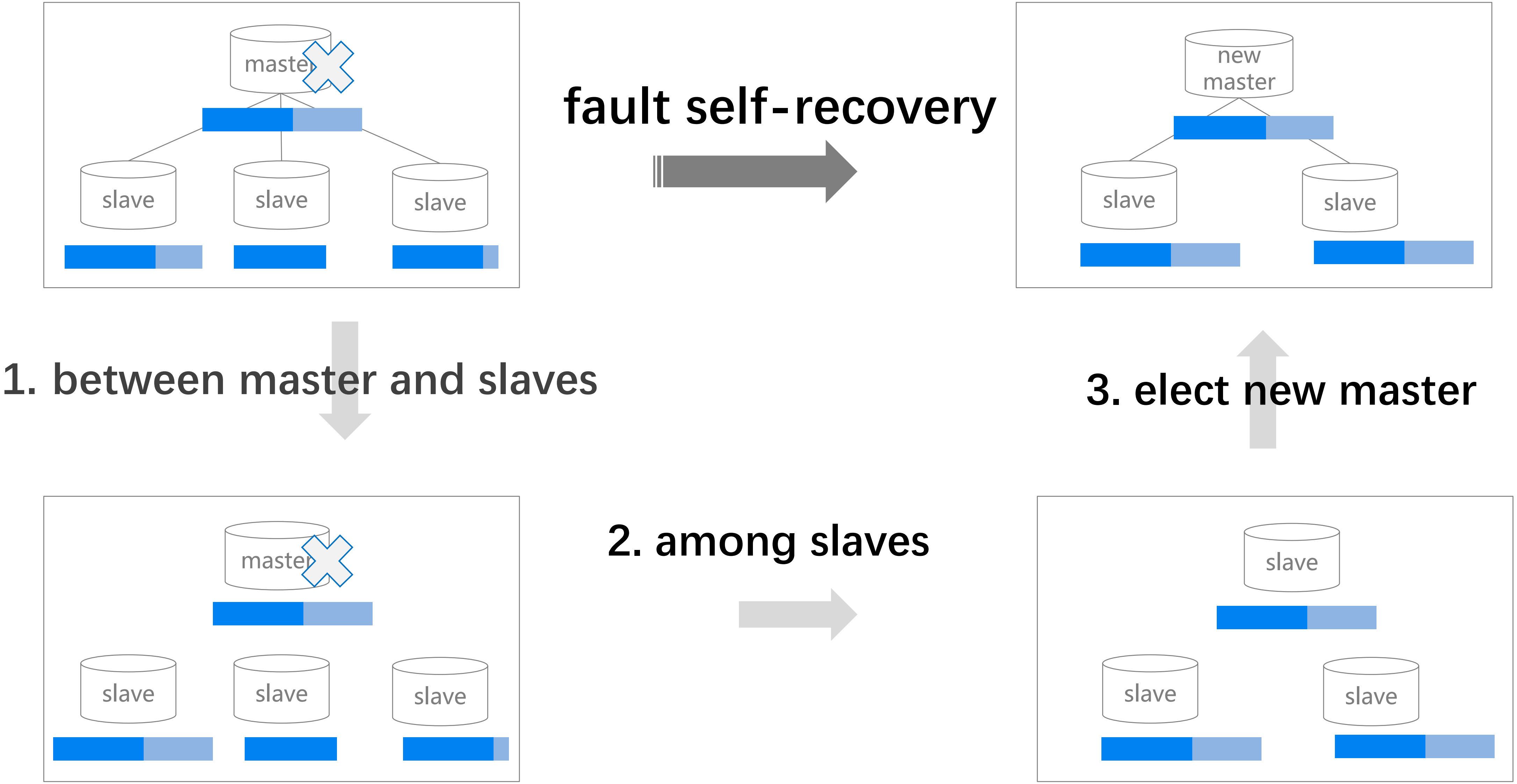
net jitter 🟢

✗ cur_time – last_time

capacity not enough 🟢

✗ slaves < min_slave

Fault Recovery: Data Consistency



fault self-recovery

1. between master and slaves

2. among slaves

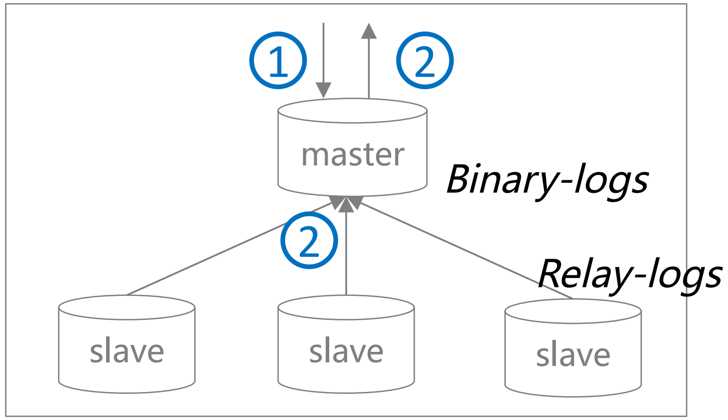
3. elect new master

Fault Recovery: Data Consistency between M&S

Master-Slave Replication Solution

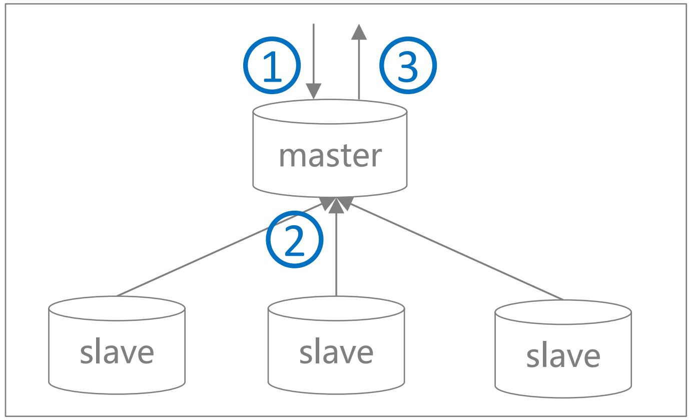
- Trade-off: data consistency, response time

async



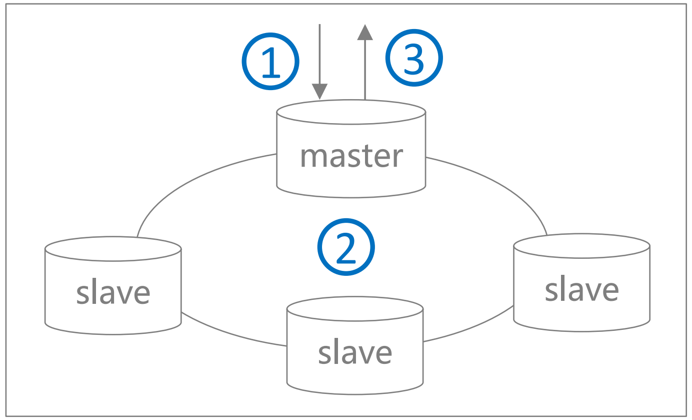
- **MAY lost data (try to)**
- High concurrency
- Low response time
- Forum, linkcache, etc

semi-sync (group)



- **data consistent**
- Higher response time
- Financial, order, etc

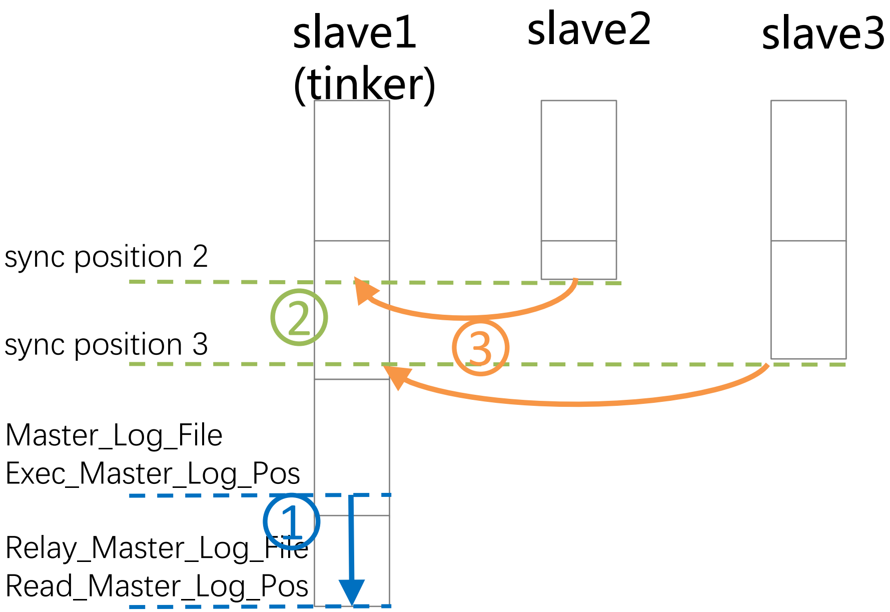
raft/MGR



- **data consistent**
- Highest response time
- Allowed sensitive switchover
- exploring on commercial

Fault Recovery: Data Consistency among Slaves

- General Process



Step 0: find the tinker(slave with complete data)

Step 1: waiting for all slaves to finish executing relaylogs

Step 2: find the sync position of all slaves

Step 3: other slaves complete data from tinker

* *(If GTID mode over 5.5, skip Step 2&3)*

- **Current Solution**

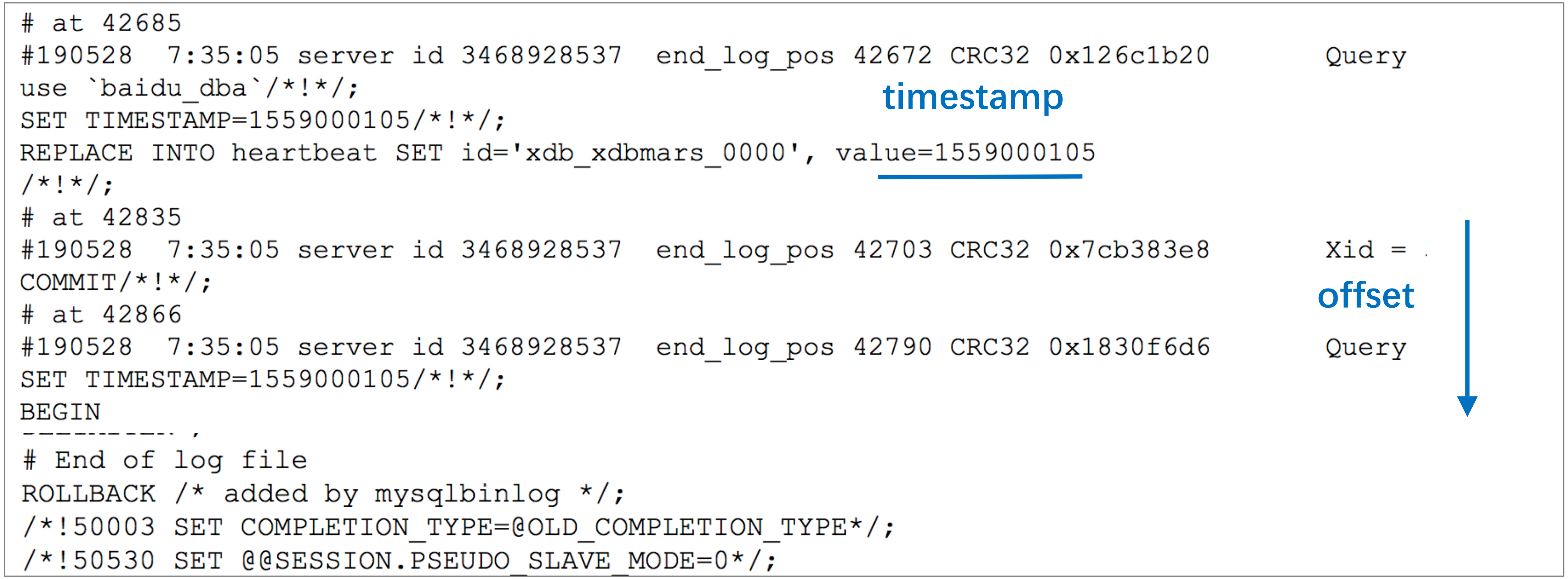
  - Find sync pos*(step 2):* compare with relay-logs pos

  - Fullfil data*(step 3):* dump to SQL file to execute

- **Problems**
  - Accuracy: binlog bugs in early version
  - Performance: waiting for all slaves finishing executing relaylogs(step1), and fullfil data(step3)
  - Safety: trust building

Fault Recovery: Data Consistency among Slaves

- Our Solution
  - timestamp per 3s
  - data progress: <last timestamp, offset>
  - tinker: t1 > t2 || (t1 == t2 && o1 > o2)



```
# at 42685
#190528  7:35:05 server id 3468928537  end_log_pos 42672 CRC32 0x126c1b20       Query
use `baidu_dba`/*!*/;
SET TIMESTAMP=1559000105/*!*/;
REPLACE INTO heartbeat SET id='xdb_xdbmars_0000', value=1559000105
/*!*/;
# at 42835
#190528  7:35:05 server id 3468928537  end_log_pos 42703 CRC32 0x7cb383e8       Xid =
COMMIT/*!*/;
# at 42866
#190528  7:35:05 server id 3468928537  end_log_pos 42790 CRC32 0x1830f6d6       Query
SET TIMESTAMP=1559000105/*!*/;
BEGIN
# End of log file
ROLLBACK /* added by mysqlbinlog */;
/*!50003 SET COMPLETION_TYPE=@OLD_COMPLETION_TYPE*/;
/*!50530 SET @@SESSION.PSEUDO_SLAVE_MODE=0*/;
```

Fault Recovery: Data Consistency among Slaves

- slave completed data from tinkers

slave1 (tinker)

Step1: execute all relaylogs

Step2.2: find sync pos of other slaves in tinker

```
relay-log.000001   (pos 10378 for slave2)
relay-log.000002   (pos 4387 for slave3)
mysql-bin.000005
```

Step3.1: flush logs n+1 times on tinker,
backup 1~nth binlogs, link to relaylogs.

```
mysql-bin.000005
mysql-bin.000006    -> relay-log.000001
mysql-bin.000007    -> relay-log.000002
mysql-bin.000008
```

End: reset tinker to defaults

```
mysql-bin.000005
…
mysql-bin.000008
```

slave2

slave3

Step2.1: stop slave; return data progress executed

Perfomance: Needn’t finish relaylog execution

Step3.2: other slaves *change master to* sync pos

```
change master to ${tinker}         change master to ${tinker}
     master_log_file =                  master_log_file =
mysql-bin. 00006,                  mysql-bin.000007,
     master_log_pos =                   master_log_pos =
10378                              4387
```

cluster recovery here
(if tinker can be master)

Benefits

- **Cover all MySQL in Baidu: 5.0~5.7**
- **Online fault recovery: 100% success**
    - master fault: 3000+ times, MTTR < 50s
    - Dataceneter fault: 10+ times, 106 simulative, MTTR < 5min
    - online switching: MTTR < 10s
- **Support Baidu financial cloud**
    - AI Bank: **first** MySQL+X86 on core banking in China.
    - China UMS: Top1 Acquirers in Asia-Pacific.
- **HA framework for other databases**

Summary

- **Complete and Automatic HA Solution**
  - **HA architecture: decentralized**
    - xagent, config center
  - **Accurate failure detection**
    - three layer detecting strategy
      - instance, cluster, global
  - **Fault Recovery: preserving data consistency**
    - master-slave synchronization: async, semi-sync, sync on raft
    - among slaves: support multi-version

Thank you !