MySQL Scalability and Reliability for Replicated Environment

by Jean-François Gagné - *Presented at* dataops.barcelona (June 2019)
Senior Infrastructure Engineer / System and MySQL Expert
jeanfrancois AT messagebird DOT com / @jfg956

MessageBird

MessageBird is a cloud communications platform founded in Amsterdam 2011.

Example of our messaging and voice SaaS:

SMS in and out, call in (IVR) and out (alert), SIP, WhatsApp, Facebook, Telegram, Twitter, WeChat, …
Omni-Channel Conversation

Details at www.messagebird.com

225+ Direct-to-Carrier Agreements

With operators from around the world

15,000+ Customers

In over 60+ countries

200+ Employees

Engineering office in Amsterdam
Sales and support offices worldwide

We are expanding:

{Software, Front-End, Infrastructure, Data, Security, Telecom, QA} Engineers
{Team, Tech, Product} Leads, Product Owners, Customer Support
{Commercial, Connectivity, Partnership} Managers

www.messagebird.com/careers

Summary

(MySQL Scalability and Reliability for Replication – do.bcn 2019)

This talk is an overview of the MySQL Replication landscape:

- Introduction to replication (what, why and how)
- Replications generalities (including tooling and lag)
- Zoom in some replication use cases (read scaling, resilience and failover)
- War story + more failover topics (bad ideas, master/master, semi-sync, …)
- Some closing remarks on scalability and sharding

I will also mention Advances Replication Subjects and provide pointers

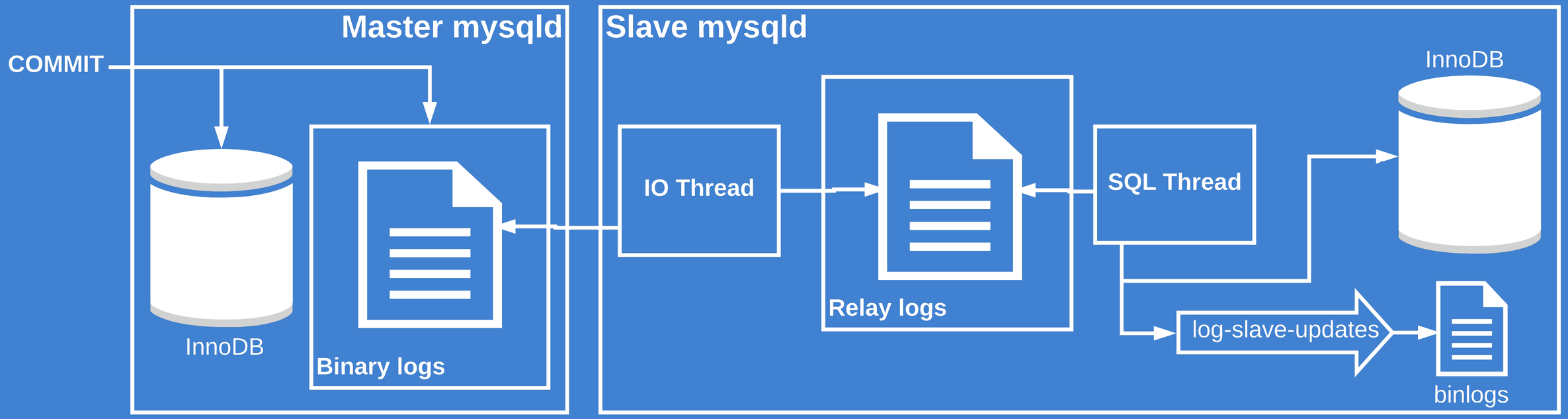
(No need to take pictures of the slides, they will be online)

Overview of MySQL Replication

(MySQL Scalability and Reliability for Replication – do.bcn 2019)

In a replicated environment, you have one master with one or more slaves:

- The master records transactions in a journal (binary logs); and each slave:
  - Downloads the journal and saves it locally in the relay logs (done by the IO thread)
  - Executes the relay logs on its local database (done by the SQL thread)
  - Could also produce binary logs to be an intermediate master (`log-slave-updates`)
  - Delay between writing to the binary log on the master and execution on slave (lag)



Why would you want replication

(MySQL Scalability and Reliability for Replication – do.bcn 2019)

- Point-in-time recovery with the master binary logs
- Stability: taking backups from a slave
- Risk management: data from X hours ago on “delayed” slaves

- Reliability of reads: reading from a slave or from the master
- Scalability: spread read load on many slaves
- Availability of writes: failing over to a slave if the master crashes

- Other reasons involving MySQL slaves (aggregation with multi-source)
- Other reasons for leveraging the binlogs (gh-ost, replicating to Hadoop)
- And much more (testing new version before upgrade, rollback of upgrade, …)

Setting-up replication [1 of 3]

(MySQL Scalability and Reliability for Replication – do.bcn 2019)

Three things to configure for replication:

- Enabling binary logging (`log_bin` is not dynamic 😐)
  (disabled by default before MySQL 8.0 😐, enabled in 8.0 ☺)
- Setting a unique server id (`server_id` is a dynamic global variable ☺)
  (the last 3 numbers of the IP address is a safe-ish choice: 10.98.76.54 → 98076054)
- Create a replication user (How-to in the manual)
- MySQL 5.6+ also have a server uuid (for GITDs)
  (generated magically, saved in auto.cnf, remember to delete if cloning from file cp)
- Many binlog format (`binlog_format` is dynamic but care as global & session)
  (ROW default in MySQL 5.7+, STATEMENT before → consider RBR if 5.6 or 5.5)
  (because Group Replication and latest parallel replication need RBR)
  (less important for MariaDB where parallel replication works well with SBR)

Then restore a backup, setup the slave, and start replicating !

Setting-up replication [2 of 3]

(MySQL Scalability and Reliability for Replication – do.bcn 2019)

GTIDs are a useful reliability/repointing tool, you should probably use them !

- If new database, I advise to enable GTIDs (and use 5.7+)
  (gtid_mode is OFF by default in all of MySQL 5.6, 5.7 and 8.0)
- If existing application with MySQL, enabling GTIDs is not straightforward
  - Needs `enforce_gtid_consistency` (not dynamic in 5.6 and only hard fail)
    (Dynamic in 5.7 and has a warning option: hopefully you are logging warnings)
    (https://jfg-mysql.blogspot.com/2017/01/do-not-ignore-warnings-in-mysql-mariadb.html)
- If MariaDB 10.0+, then you are lucky: GTIDs come for free

I have a love/hate relationship with GTIDs: they are not my favourite tool

- I prefer Binlog Servers, but the open-source implementation is very young (Ripple)
  (And it is an Advance Replication Subject not detailed here)
  https://jfg-mysql.blogspot.com/2015/10/binlog-servers-for-backups-and-point-in-time-recovery.html

Setting-up replication [3 of 3]

(MySQL Scalability and Reliability for Replication – do.bcn 2019)

Care with `binlog-do-db` and `binlog-ignore-db`:

- They compromise PiT recovery and might not do what you expect
  https://www.percona.com/blog/2009/05/14/why-mysqls-binlog-do-db-option-is-dangerous/

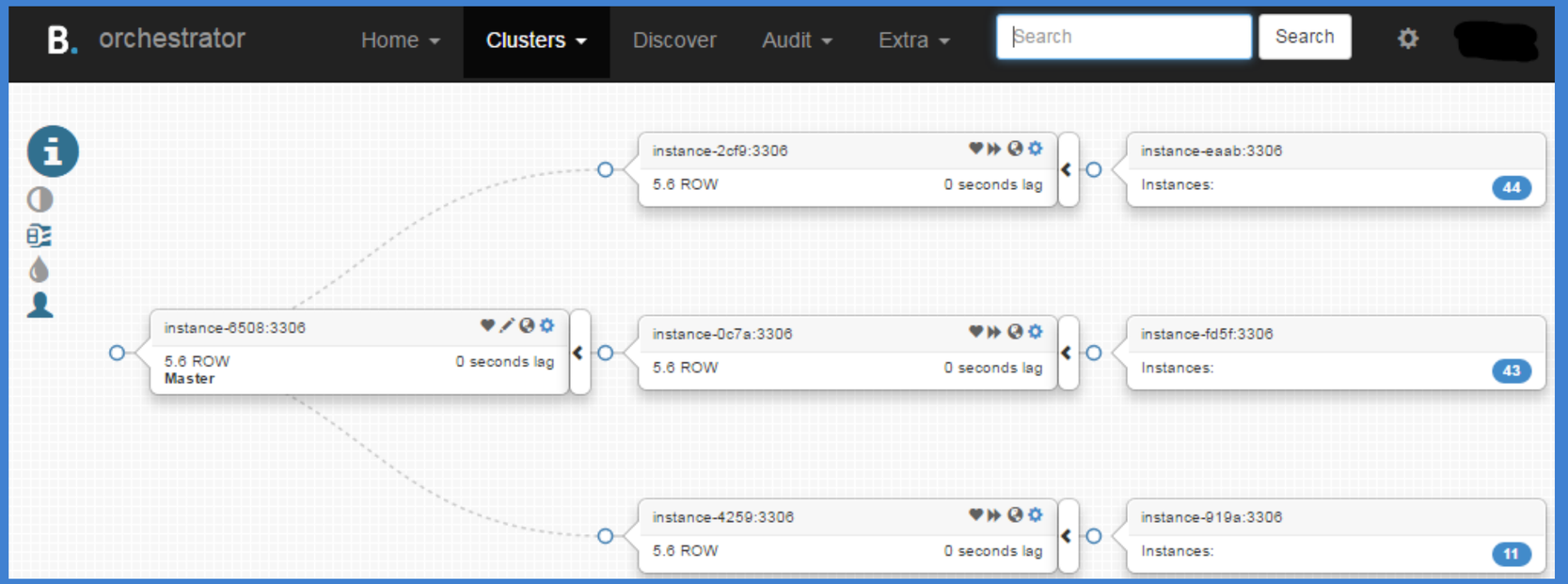
A few notes about slaves (for when we will get there):

- Consider enabling binary logging on slaves (**`log-slave-updates`**)
  - For failover to slaves and a good way to keep copies of binary logs for PiTR
- Replication crash-safety: enabled by default in MySQL 8.0, not before
  - If not 8.0 and for enabling replication crash safety on slaves
    set **`relay-log-info-repository = TABLE`** and **`relay-log-recovery = 1`**
    https://www.slideshare.net/JeanFranoisGagn/demystifying-mysql-replication-crash-safety
  - But with GTID, it needs sync_binlog = 1 and trx_commit = 1 ☹ (Bug#92109)
- Be careful with replication filters:
  - Might not do what you expect (see binlog-do-db and binlog-ignore-db above)

Replication visualization tooling

(MySQL Scalability and Reliability for Replication – do.bcn 2019)

Orchestrator…



… and pt-slave-find:

```
10.3.54.241:3306
+- 10.23.12.1:3306
 +- 10.2.18.42:3306
+- 10.23.78.5:3306
 +- 10.2.34.56:3306
```

- Orchestrator needs some effort in deployment and configuration
- And it can also be used as a failover tool
- But in the meantime, pt-slave-find can help

Replication lag [1 of 7]

(MySQL Scalability and Reliability for Replication – do.bcn 2019)

Biggest enemy in a replication setup: replication lag

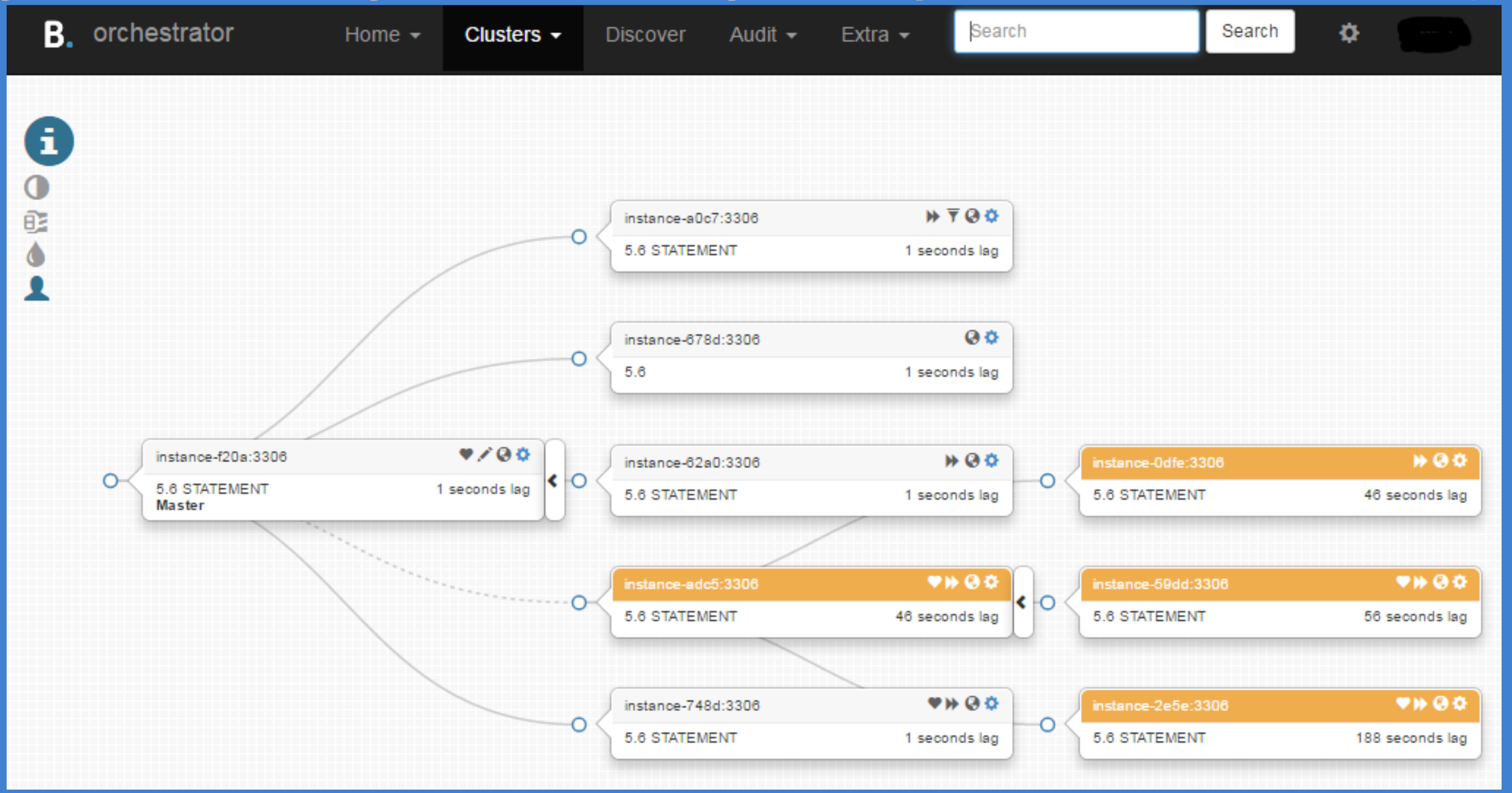
- Temporary lag: caused by a cold cache after a restart
- Occasional lag: caused by a write burst or some long transactions
- Perpetual lag of death: when slaves almost never catch-up

Monitoring lag:

- **`Second Behind Master`** from **`SHOW SLAVE STATUS`**
- From the Percona Toolkit: pt-heartbeat (or variation)
- (Also a new feature in MySQL 8.0 which I am not familiar with)

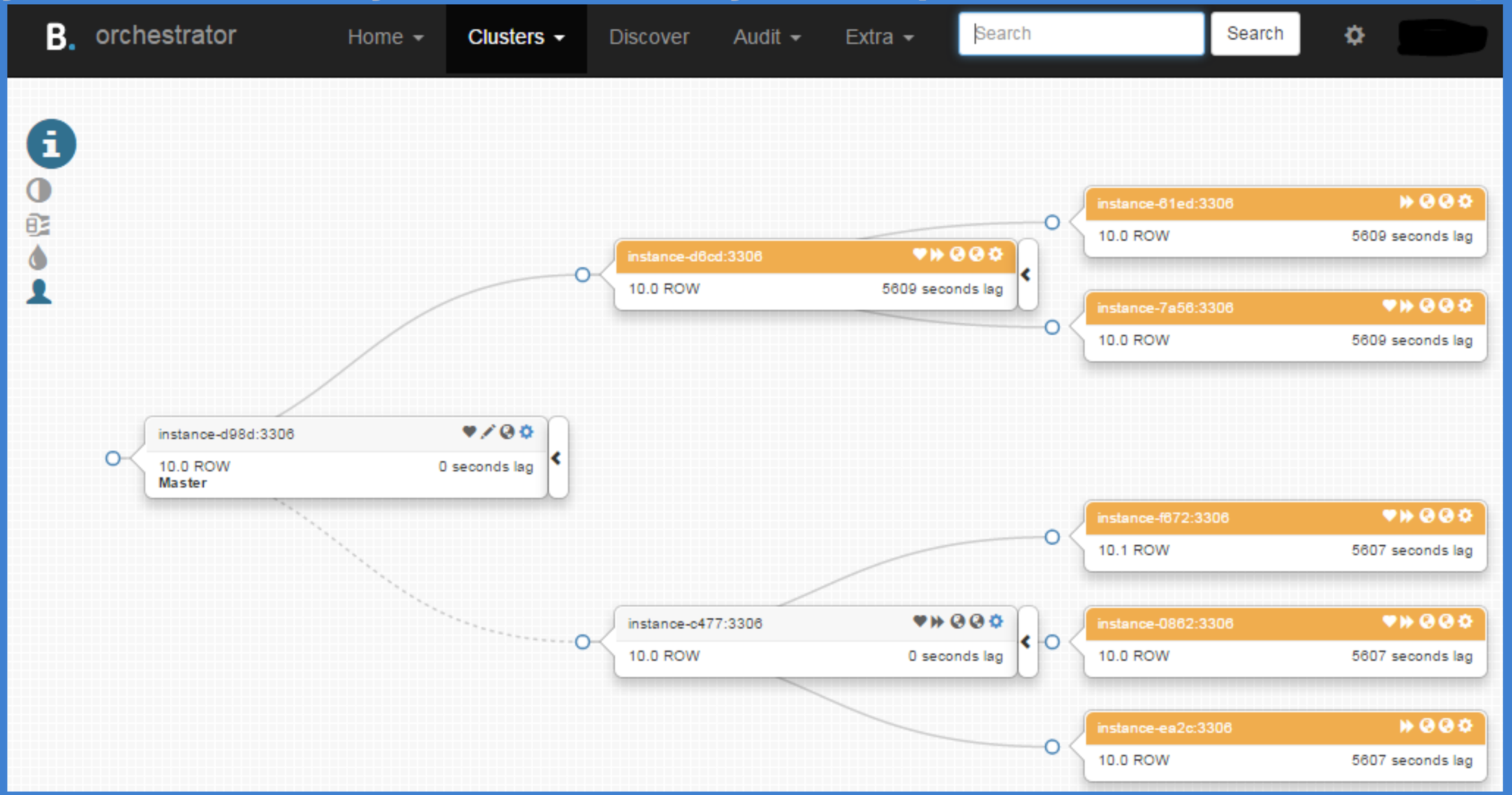
Replication lag [2 of 7]

(MySQL Scalability and Reliability for Replication – do.bcn 2019)



Replication lag [3 of 7]

(MySQL Scalability and Reliability for Replication – do.bcn 2019)



Replication lag [4 of 7]

(MySQL Scalability and Reliability for Replication – do.bcn 2019)

Techniques for preventing lag:

- Avoid long/big transactions
- Manage your batches
- Increase the replication throughput

If all above fail, you have a capacity problem → sharding

Replication lag [5 of 7]

(MySQL Scalability and Reliability for Replication – do.bcn 2019)

Avoiding replication lag by not generating long transactions:

- Avoid unbounded/unpredictable writes
  (we do not like `FOREIGN KEY … ON DELETE/UPDATE CASCADE`)
  (we do not like unbounded `UPDATE/DELETE` and we do not like `LOAD DATA`)
- Avoid `ALTER TABLE`, use non-blocking methods (pt-osc / ghost)
  (note that an online `ALTER` on the master blocks replication on slaves)
- Row-Based-Replication needs a primary key to run efficiently on slaves
  https://jfg-mysql.blogspot.com/2017/08/danger-no-pk-with-RBR-and-mariadb-protection.html

- Long transactions compound with Intermediate Masters
- And they prevent parallel replication from working well
  https://medium.com/booking-com-infrastructure/better-parallel-replication-for-mysql-14e2d7857813

Replication lag [6 of 7]

(MySQL Scalability and Reliability for Replication – do.bcn 2019)

Avoiding replication lag by throttling batches:

- Split a batch in many small chunks
- Have a parameter to tune the chunk sizes (small transactions)
- Do not hit the master hard with all the small transactions
- Slow down: sleep between chucks (parameter), but by how much…

Advanced replication lag avoidance techniques for batches:

- Throttling with DB-Waypoints (how to get the sleep between chunks right)
  https://www.slideshare.net/JeanFranoisGagn/how-bookingcom-avoids-and-deals-with-replication-lag-74772032
- Also the MariaDB No-Slave-Left-Behind Patch/Feature
  https://jira.mariadb.org/browse/MDEV-8112

Replication lag [7 of 7]

(MySQL Scalability and Reliability for Replication – do.bcn 2019)

Avoiding lag by increasing write capacity:

- Make writes more efficient
  (Many database/table optimisations possible: https://dbahire.com/pleu18/)
- Use a faster storage (SSD for read before write & cache for quicker syncs)
- Consider parallel replication for improved slave throughput
  https://www.slideshare.net/JeanFranoisGagn/the-full-mysql-and-mariadb-parallel-replication-tutorial

- Care with `sync_binlog = 0` and `trx_commit = 2`
  They have consequences…
  https://jfg-mysql.blogspot.com/2018/10/consequences-sync-binlog-neq-1-part-1.html

Let’s dive in the use cases of MySQL Replication

Backups and related subjects [1 of 2]

(MySQL Scalability and Reliability for Replication – do.bcn 2019)

Without slaves, backups must be done on the master

- Options are mysqldump/pump, XtraBackup, stop MySQL + file cp…
- All are disruptive in some ways

It is better to do backup on slaves (less disruptive):

- Same solutions as above (in the worst case, cause some replication lag)
- If you are HA, you should not afraid to stop a slave for taking a backup (mysqldump & XtraBackup have drawbacks avoided by stop MySQL + file cp) or from wiping data on a slave for restoring – and testing – a backup (tested backups are the only good backups)

Backups and related subjects [2 of 2]

(MySQL Scalability and Reliability for Replication – do.bcn 2019)

Also, having slaves is a way to keep extra copies of the binlogs:
(Binlog Servers might be a better way to achieve that, more on this later)

- Allows PiT Recovery of a backup (restore and apply the binlogs)
- Care with `sync_binlog != 1`: missing trx in binlogs on OS crash
  https://jfg-mysql.blogspot.com/2018/10/consequences-sync-binlog-neq-1-part-1.html
  and not replication crash safe with GTIDs: Bug#70659 and #92109

Delayed replication:

- Useful to recover from whoopsies: `DROP`ing wrong table/database
- Original tool in the Percona Toolkit: pt-slave-delay
- Not needed since MySQL 5.6 and MariaDB 10.2
  (`MASTER_DELAY` option included in `CHANGE MASTER TO`)

Reading from Slave(s) [1 of 5]

(MySQL Scalability and Reliability for Replication – do.bcn 2019)

Why would you want reading from slave(s):

- Resilience: still being able to read when a node is down
- Capacity/Scaling: when the master is overloaded by reads
- Performance: splitting fast/slow queries to different slaves
- Latency: bringing data closer to readers in a multi data-center setup

Complexity of reading from slaves: dealing with lag (consistency)

- Stale read on a lagging slave (not always a problem if locally consistent)
- Temporal inconsistencies (going from a lagging to a non-lagging slave)

Reading from Slave(s) [2 of 5]

(MySQL Scalability and Reliability for Replication – do.bcn 2019)

More complexity: service discovery at scale (where to read from)

- DNS, proxy, pooling… (not discussed in detail in this talk)
  (ProxySQL with a LoadBalancer for High Availability is a good start)
- At scale, all centralised solutions will be a bottleneck (Proxies and DNS)

<u>Bad way</u> for increasing Read Reliability:

- Normally read on the master and when it fails read on a slave

➢ This hides consistency/lagging problem most of the time
and makes these visible only when the master fails

- When few slaves → read from them and fallback to the master
- When many slaves → never fallback to the master (embrace lag)

Reading from Slave(s) [3 of 5]

(MySQL Scalability and Reliability for Replication – do.bcn 2019)

Bad solution #1 to cope with lag:

Falling back to reading on the master

- If slaves are lagging, maybe we should read from the master…
- This looks like an attractive solution to avoid stale reads, but…
- It does not scale (why are you reading from slaves ?)
- This will cause a sudden load on the master (in case of lag)
- And it might cause an outage on the master (and this would be bad)

- It might be better to fail a read than to fallback to (and kill) the master
- Reading from the master might be okay in very specific cases
  (but make sure to avoid the problem described above)

Reading from Slave(s) [4 of 5]

(MySQL Scalability and Reliability for Replication – do.bcn 2019)

Bad solution #2 to cope with lag:

Retry on another slave

- When reading from a slave: if lag, then retry on another slave
- This scales better and is OK-ish (when few slaves are lagging)
- But what happens if many slaves are lagging ?
- Increased load (retries) can slowdown replication
- This might overload the slaves and cause a good slave to start lagging
- In the worst case, this might kill slaves and cause a domino effect

- Again: probably better to fail a read than to cause a bigger problem
- Consider separate “interactive” and “reporting” slaves (for expensive reads)

Reading from Slave(s) [5 of 5]

(MySQL Scalability and Reliability for Replication – do.bcn 2019)

Advanced Replication Lag Coping Solutions:

- ProxySQL GTID consistent reads
  https://proxysql.com/blog/proxysql-gtid-causal-reads
- DB-Waypoints for consistent read (Booking.com solution)
  (subject of Simon's talk)

Read views:

- Lag might not be a problem if you can "read-your-own-writes"…
  (subject of Simon's talk)

High Availability of Writes [1 of 6]

(MySQL Scalability and Reliability for Replication – do.bcn 2019)

In a MySQL replication deployment, the master is a SPOF

- Being able to react to its failure is crucial for high availability of writes
  (another solution is to **`INSERT`** in many places – much harder for **`UPDATE`**s)
  (not just master/master but also possible 2 completely separate databases)

Solutions to restore the availability of writes after a master failure

- Restart the master
  (what if hardware fails, delay of crash recovery, and the cache starts cold)
- Database on shared storage (storage fails, crash recovery, and cold cache)
- Failover to a slave (what if **`DROP DATABASE`**)

High Availability of Writes [2 of 6]

(MySQL Scalability and Reliability for Replication – do.bcn 2019)

Failing-over the master to a slave is my favorite high availability method

- But it is not as easy as it sounds, and it is hard to automate well
- There are many corner cases, and things will sometimes go wrong
  (what if slaves are lagging, what if some data is only on the master, …)
- An example of complete failover solution in production:
  https://githubengineering.com/mysql-high-availability-at-github/

The five considerations when failing-over the master to a slave:
(https://jfg-mysql.blogspot.com/2019/02/mysql-master-high-availability-and-failover-more-thoughts.html)

- **Plan** how are your doing Write high availability
- Decide **when** you apply your plan (Failure Detection – *FD*)
- **Tell** the application about the change (Service Discovery – *SD*)
- **Protect** against the limit of FD and SD for avoiding split-brains (fencing)
- **Fix** your data if something goes wrong

High Availability of Writes [3 of 6]

(MySQL Scalability and Reliability for Replication – do.bcn 2019)

Failure detection (*FD*) is the 1st part (and 1st challenge) of failover

- It is a very hard problem: partial failure, unreliable network, partitions, …
- It is impossible to be 100% sure of a failure, and confidence needs time
  → quick FD is unreliable, relatively reliable FD implies longer unavailability

➢ You need to accept that FD generates false positive (and/or false negative)

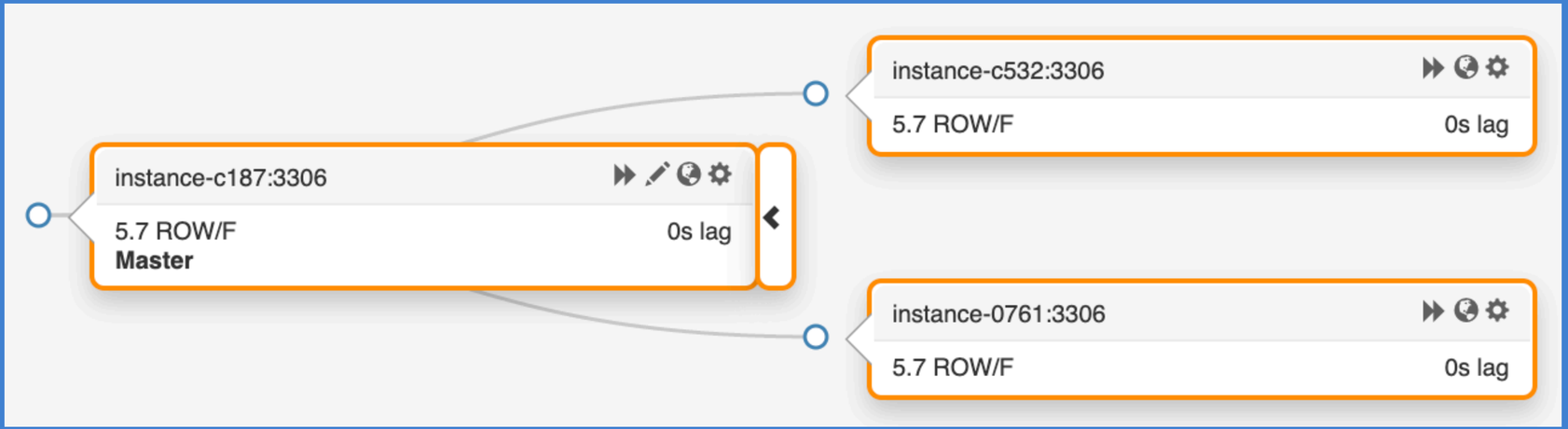
Repointing is the 2nd part of failover:

- Relatively easy with the right tools: MHA, GTID, Pseudo-GTID, Binlog Servers, …
- Complexity grows with the number of direct slaves of the master
  (what if you cannot contact some of those slaves…)
- Some software for easing repointing:
  - Orchestrator, Ripple Binlog Server,
    Replication Manager, MHA, Cluster Control, MaxScale, …

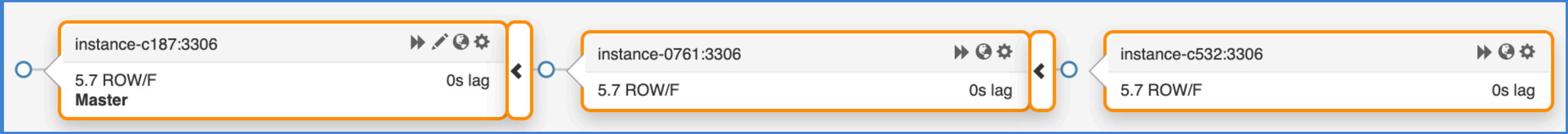
High Availability of Writes [4 of 6]

(MySQL Scalability and Reliability for Replication – do.bcn 2019)

In this configuration, when the master fails, one of the slave needs to be repointed:



This configuration might simplify repointing, but…

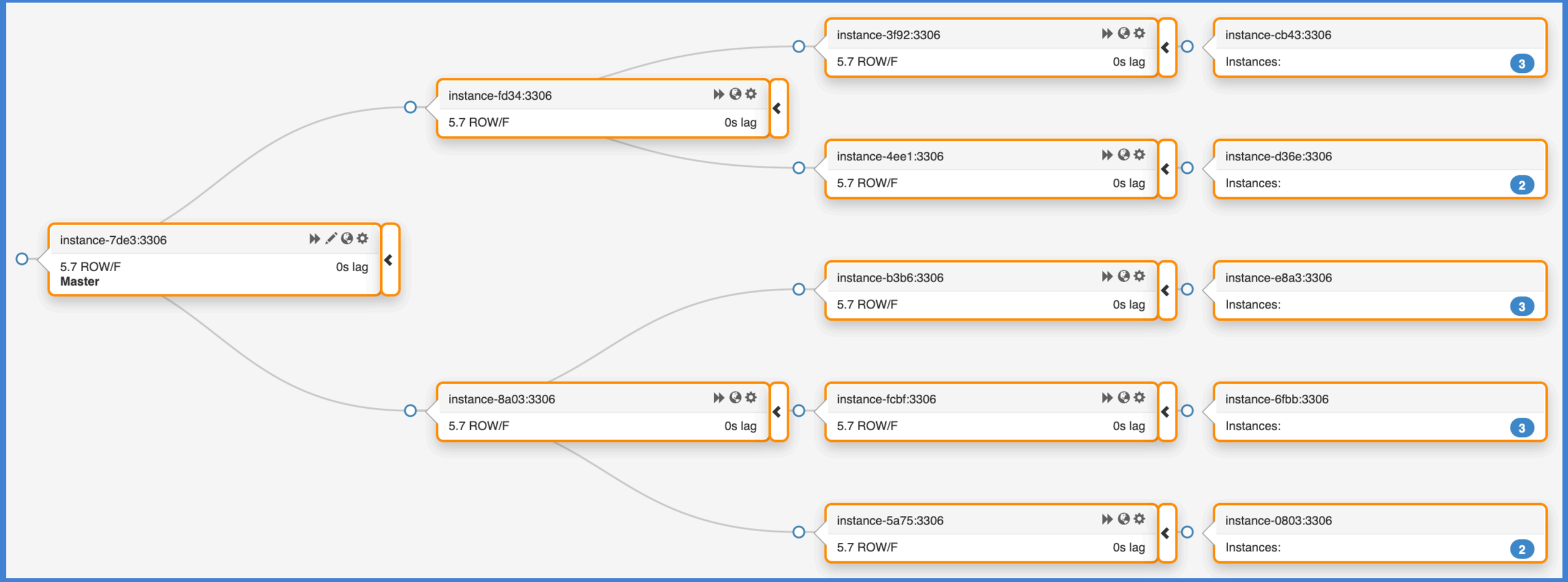


… what if the intermediate master fails ?

High Availability of Writes [5 of 6]

(MySQL Scalability and Reliability for Replication – do.bcn 2019)

… and you cannot avoid this problem in large deployments !



High Availability of Writes [6 of 6]

(MySQL Scalability and Reliability for Replication – do.bcn 2019)

Service Discovery (*SD*) is the 3rd part (and 2nd challenge) of failover:

- If centralised, it is a SPOF; if distributed, impossible to update atomically
- SD will either introduce a bottleneck (including performance limits) or will be unreliable in some way (pointing to the wrong master)
- Some ways to implement Service Discover: DNS, ViP, Proxy, Zookeeper, …
  http://code.openark.org/blog/mysql/mysql-master-discovery-methods-part-1-dns

➢ Unreliable FD and unreliable SD is a receipt for split-brains !

Protecting against split-brains: Adv. Subject – not many solutions at scale
(Proxies and well configured semi-synchronous replication might help)

Fixing your data in case of a split-brain: only you can know how to do this !
(Avoiding auto-increment by using UUID might help)

Failover War Story [1 of 6]

(MySQL Scalability and Reliability for Replication – do.bcn 2019)

Master failover does not always go as planned

We will look at a War Story from Booking.com

- Autopsy of a [MySQL] automation disaster (FOSDEM 2017)
  https://www.slideshare.net/JeanFranoisGagn/autopsy-of-an-automation-disaster

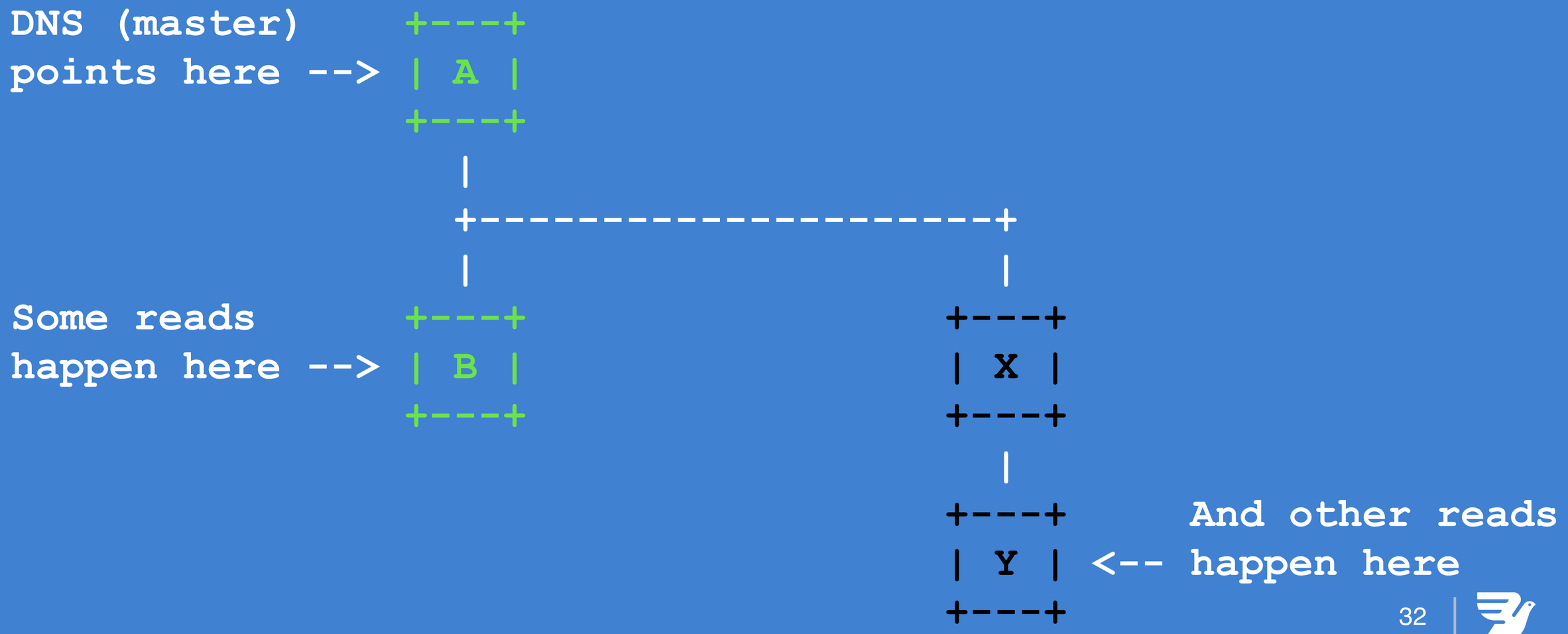
GitHub also has a recent incident with a public Post-Mortem:

- October 21 post-incident analysis
  https://blog.github.com/2018-10-30-oct21-post-incident-analysis/

(Please share your war stories so we can learn from each-others' experience)

Failover War Story [2 of 6]

(MySQL Scalability and Reliability for Replication – do.bcn 2019)



Failover War Story [3 of 6]

(MySQL Scalability and Reliability for Replication – do.bcn 2019)

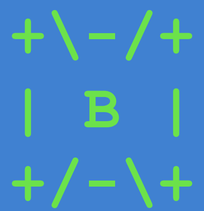
```
DNS (master)      +\-/+
points here --> | A |
but they are      +/-\+
now failing
```

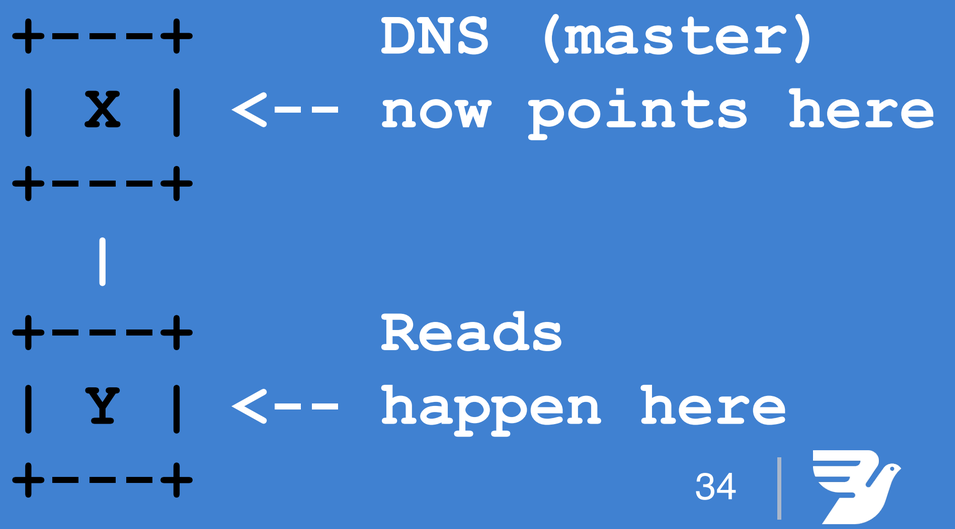
```
Some reads        +\-/+
happen here --> | B |
but they are      +/-\+
now failing
```

```
+---+
| X |
+---+
  |
+---+       Some reads
| Y | <-- happen here
+---+
```


Failover War Story [4 of 6]

(MySQL Scalability and Reliability for Replication – do.bcn 2019)

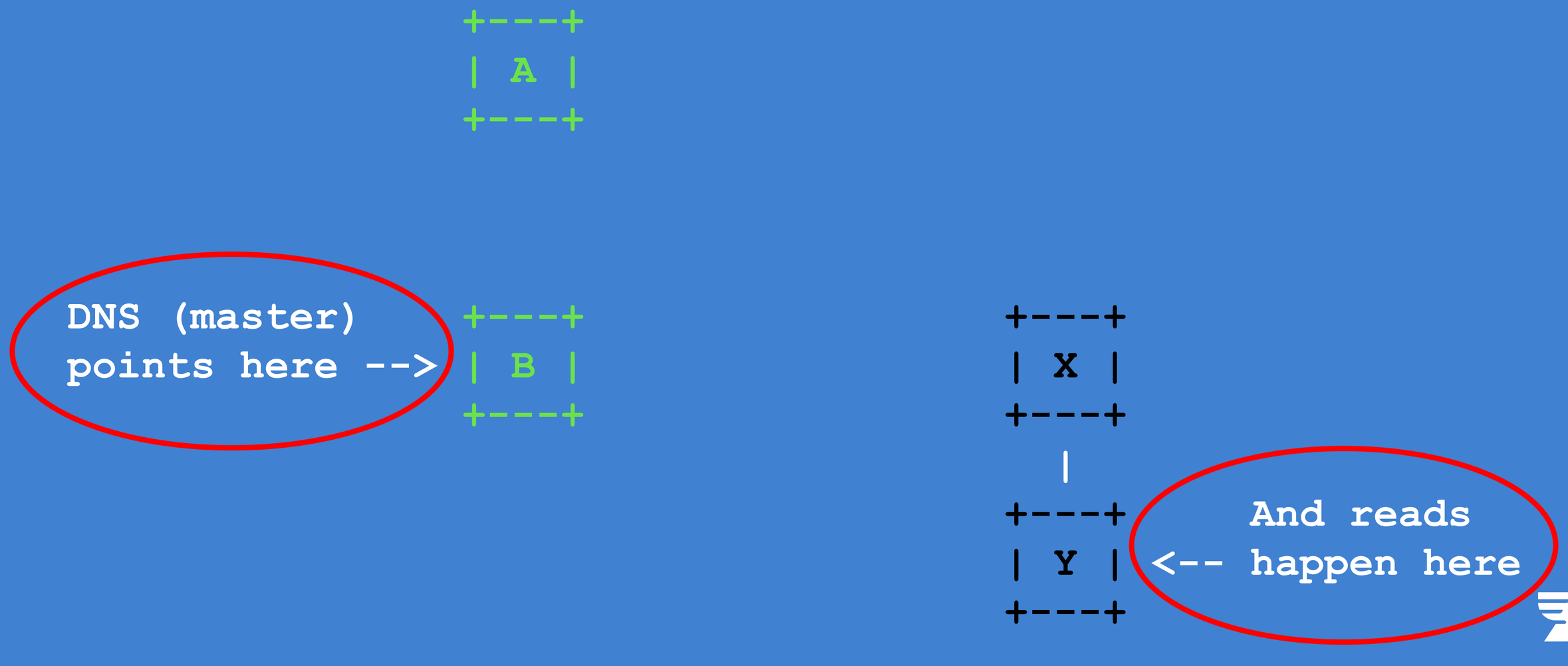




Failover War Story [5 of 6]

(MySQL Scalability and Reliability for Replication – do.bcn 2019)

After some time...

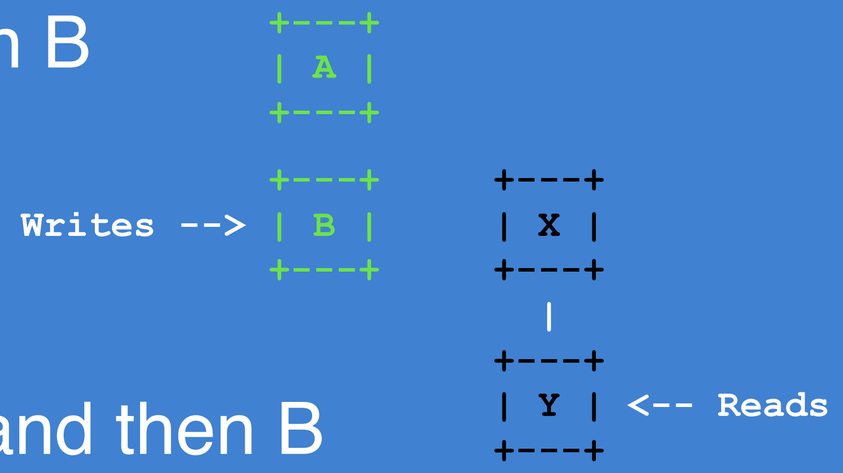


Failover War Story [6 of 6]

(MySQL Scalability and Reliability for Replication – do.bcn 2019)

This is not good:

- When A and B failed, X was promoted as the new master
- Something made DNS point to B → writes are now happening there
- But B is outdated: all writes to X (after the failure of A) did not reach B
- So we have data on X that cannot be read on B
- And we have new data on B that is not on Y



Fixing the data would not have been easy:

- Same auto-increments were allocated on X and then B

➢ UUID would have been better (increasing for **INSERT**ing in PK order)

http://mysql.rjweb.org/doc.php/uuid

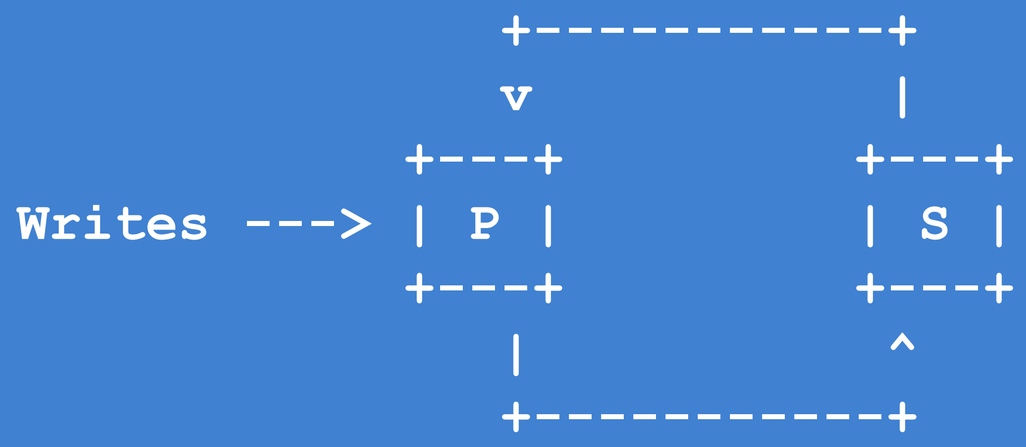
https://www.percona.com/blog/2014/12/19/store-uuid-optimized-way/

Failover of writes is not easy !

Do not do circular replication [1 of 3]

(MySQL Scalability and Reliability for Replication – do.bcn 2019)

Some people are deploying ring/circular replication for HA of writes



- If the primary fails, the application switches writes (and read) to the standby

This is a bad idea in the general case !

(Friends do not let friends do ring/circular replication)

Do not do circular replication [2 of 3]

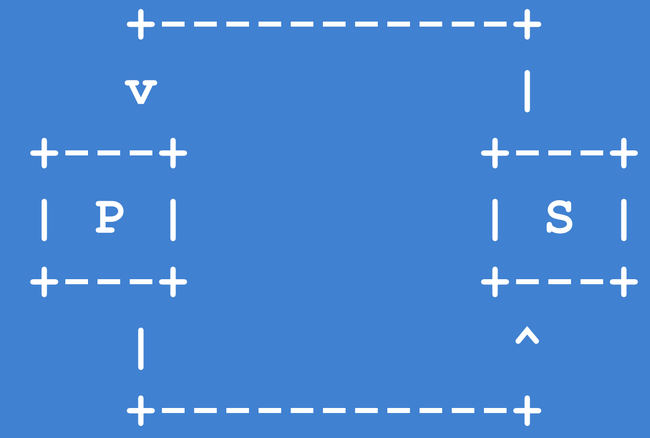
(MySQL Scalability and Reliability for Replication – do.bcn 2019)

Some people are deploying ring/circular replication for HA of writes

But this is a bad idea in the general case !

- ~~Looks attractive as nothing to do on the DB side~~
- Implies S writable: easy to write there by mistake
- Split-brain if apps write simultaneously on P and S
- What is S is lagging when P fails ?
- Out-of-order writes on S when P comes back up
- Replication breakage on P when it gets writes from S

➢ Maintenance nightmare: do not go there !



https://www.percona.com/blog/2014/10/07/mysql-ring-replication-why-it-is-a-bad-option/

(and many more horror stories in the DBA community)

The right way is failover to S and discard P → FD + failover + SD

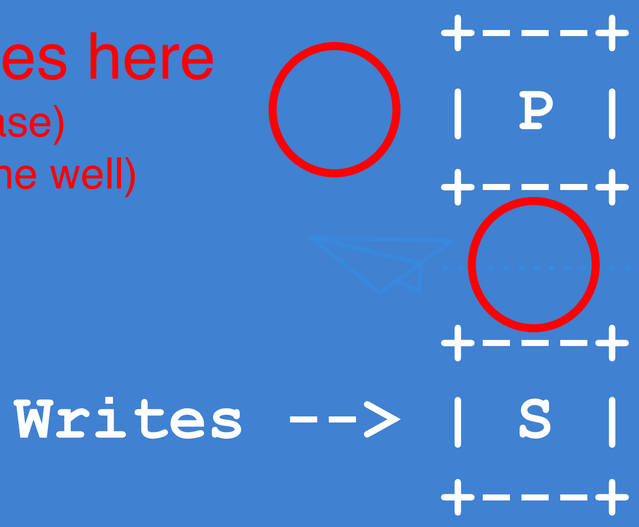
The right way is failover to S and discard P
1) Failure Detection

The right way is failover to S and discard P

2) Failover + Service Discovery

The right way is failover to S and discard P 3) Fencing

Hopefully no writes here
(if no SD edge-case)
(and/or fencing is done well)



← No replication here if P is back !

Do not do circular replication [3 of 3]

(MySQL Scalability and Reliability for Replication – do.bcn 2019)

Some people are deploying ring/circular replication for HA of writes:

- If the primary fails, the application switches writes (and read) to the standby

This is a bad idea in the general case !

- The use cases for ring replication are niche (and easy to slip out of by mistake)

If you still want to go there (and think you can tackle the Dragon):

- Embrace split-brain and write on both nodes all the time !
- Enable chaos monkey to see what happens with out-of-order writes
- Be prepared to have replication breakages and be ready to fix them

Galera/Percona XtraDB Cluster and Group Replication
are the only true (and safe) master-master solutions

Galera/XtraDB Cluster and Group Replication

(MySQL Scalability and Reliability for Replication – do.bcn 2019)

(IANAE in those technologies and I have little experience with them)

Galera/Percona XtraDB Cluster and Group Replication:
are the only true and safe master-master solutions

- They do not provide a better failure detection (only agreement on FD)
- They guarantee data consistency in a master/master setup
  which is interesting to work around the limitations of FD and Service Discovery
- The drawback is failing conflicting transactions on commit (optimistic locking)
  or failing transaction when no quorum (and avoiding split-brain)

- IMHO, these are consistency solutions (failing transactions does not look like HA to me)
  which is still very interesting in some respects (fencing + HA of writes by multi-master)
- Does not (yet) work in a multi data-center deployment (still need standard replication)
- This is clearly an Advance Replication Subject

Semi-Syncronous Replication

(MySQL Scalability and Reliability for Replication – do.bcn 2019)

When failing-over to a slave, committed transactions can be lost
(Some transactions on the crashed master might not have reached slaves)
→ violation of durability (ACI**D**) in the replication topology

Except if Lossless Semi-Sync is used (MySQL 5.7+)

Semi-sync makes transactions durable by replicating them on commit:

- The cost is higher commit latency
- And the risk of blocking on commit (lower availability, but interesting fencing)

Semi-sync can also be used to fence the master by refusing commit
(And it is an Advanced Subject that I cannot describe in more details here)

https://www.percona.com/community-blog/2018/08/23/question-about-semi-synchronous-replication-answer-with-all-the-details/

Pseudo GTIDs

(MySQL Scalability and Reliability for Replication – do.bcn 2019)

Pseudo-GTIDs:

- A way to get GTID-like features without GTIDs
- They work with any version of MySQL/MariaDB (even 5.5)
- But they assume in-order-commit → does not work with MTS

They also provide slave replication crash safety:

- With `log-slave-updates` and `sync_binlog = 1`

https://github.com/github/orchestrator/blob/master/docs/pseudo-gtid.md

The *JFG* rules of replication (and failover) [1 of 4]

(MySQL Scalability and Reliability for Replication – do.bcn 2019)

A slave that is not a candidate for failover should not directly replicate from the master (unless you accept to maybe losing it when failing-over)

Because it might be ahead of the others (and semi-sync does not help)

In more details:

- A slave without binlogs should not replicate from the master (unless…)
- A slave with RBR binlogs when the master is SBR should not…
- A slave in a VLAN (or data-center) not suitable for a master should not…
- A slave with replication filters should not…
- A mutli-source slave should not…
- Probably more…

So you need to replicate via Intermediate Master

(Binlog Servers would make this obsolete)

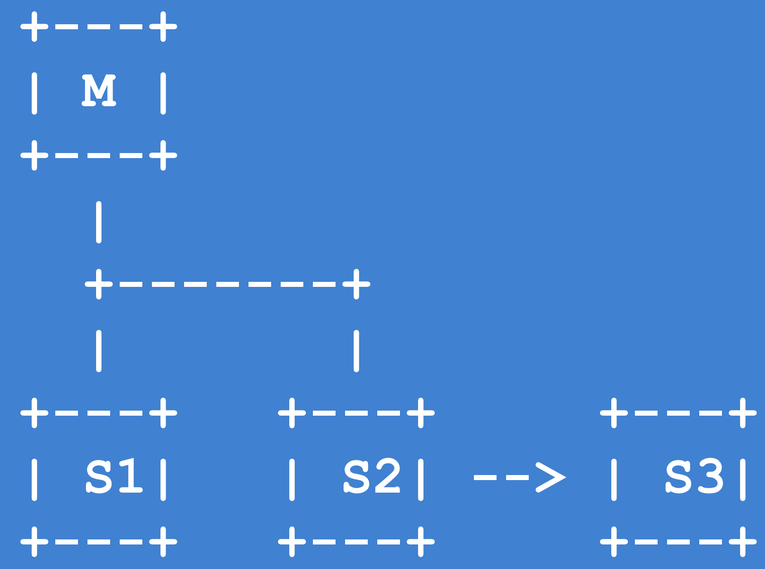
The *JFG* rules of replication (and failover) [2 of 4]

(MySQL Scalability and Reliability for Replication – do.bcn 2019)

Potential violations of the rules:

If S3 has binary logging disabled,
Or if S3 is RBR and the others SBR,
Or if S3 is in the wrong VLAN/data-center,
Or <many other reasons>…

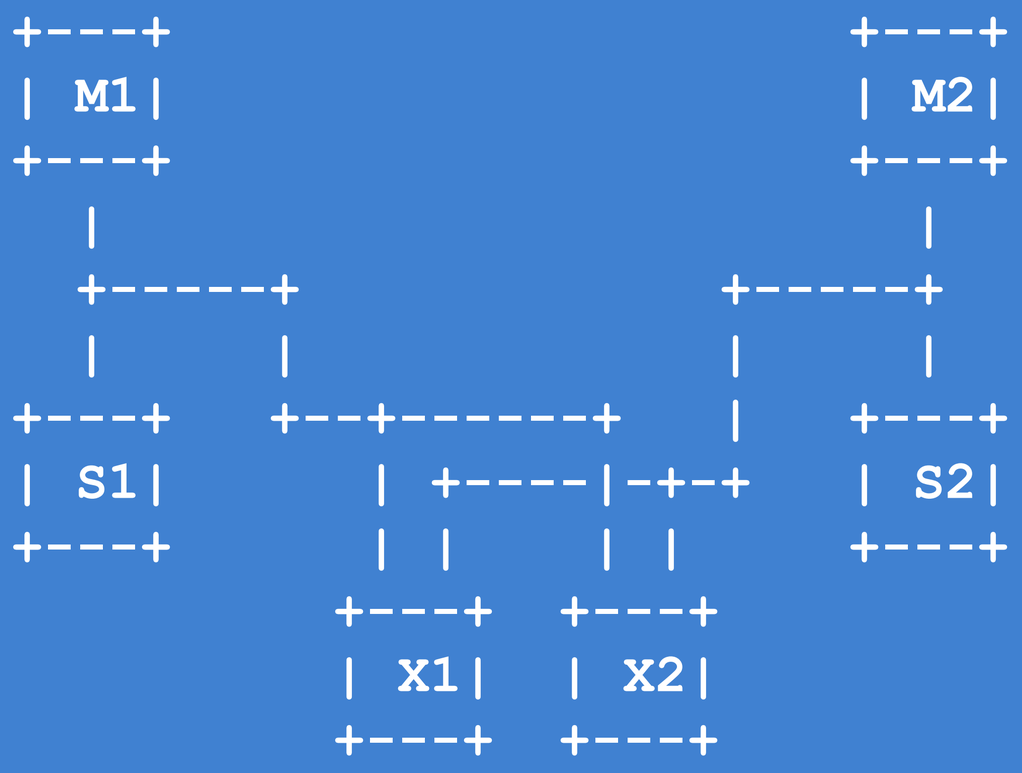
This is ok:

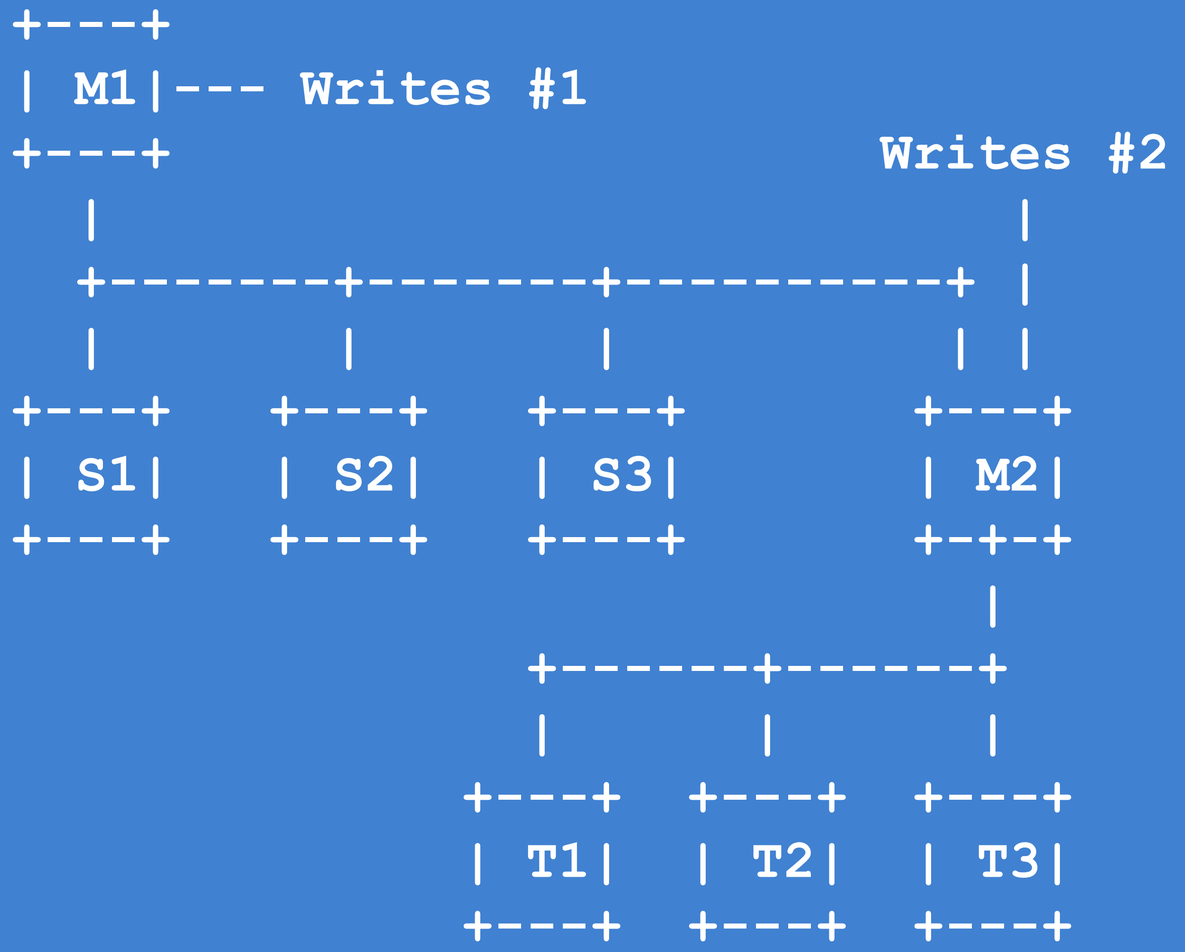


The *JFG* rules of replication (and failover) [3 of 4]

(MySQL Scalability and Reliability for Replication – do.bcn 2019)

More violations of the rules:

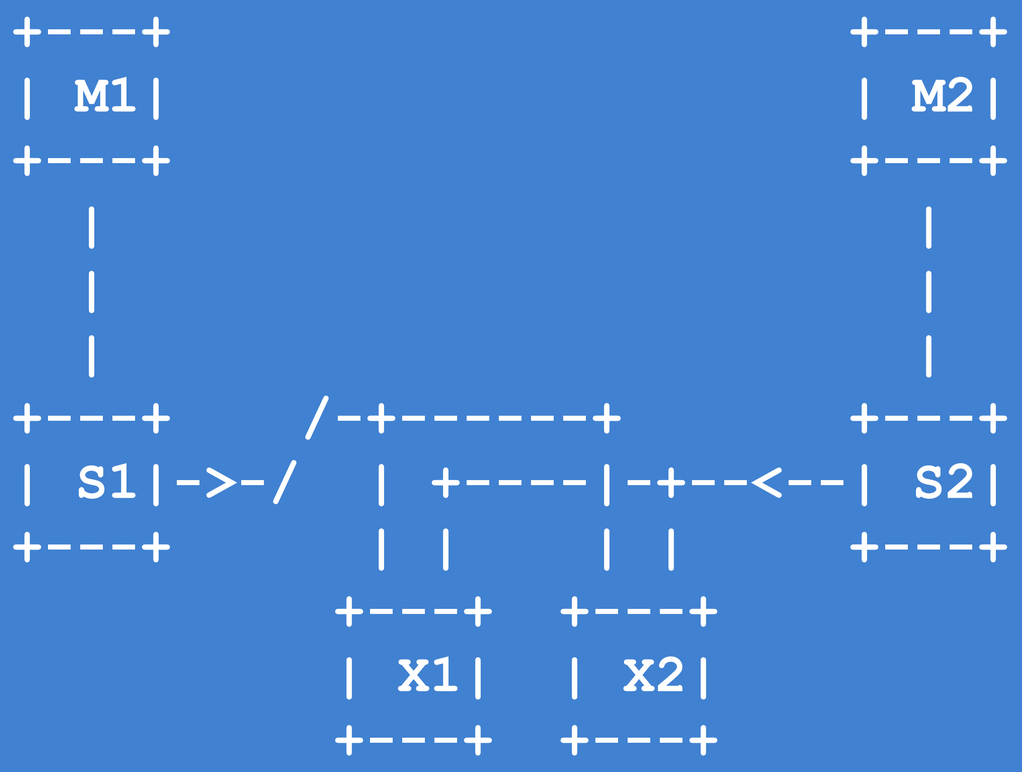


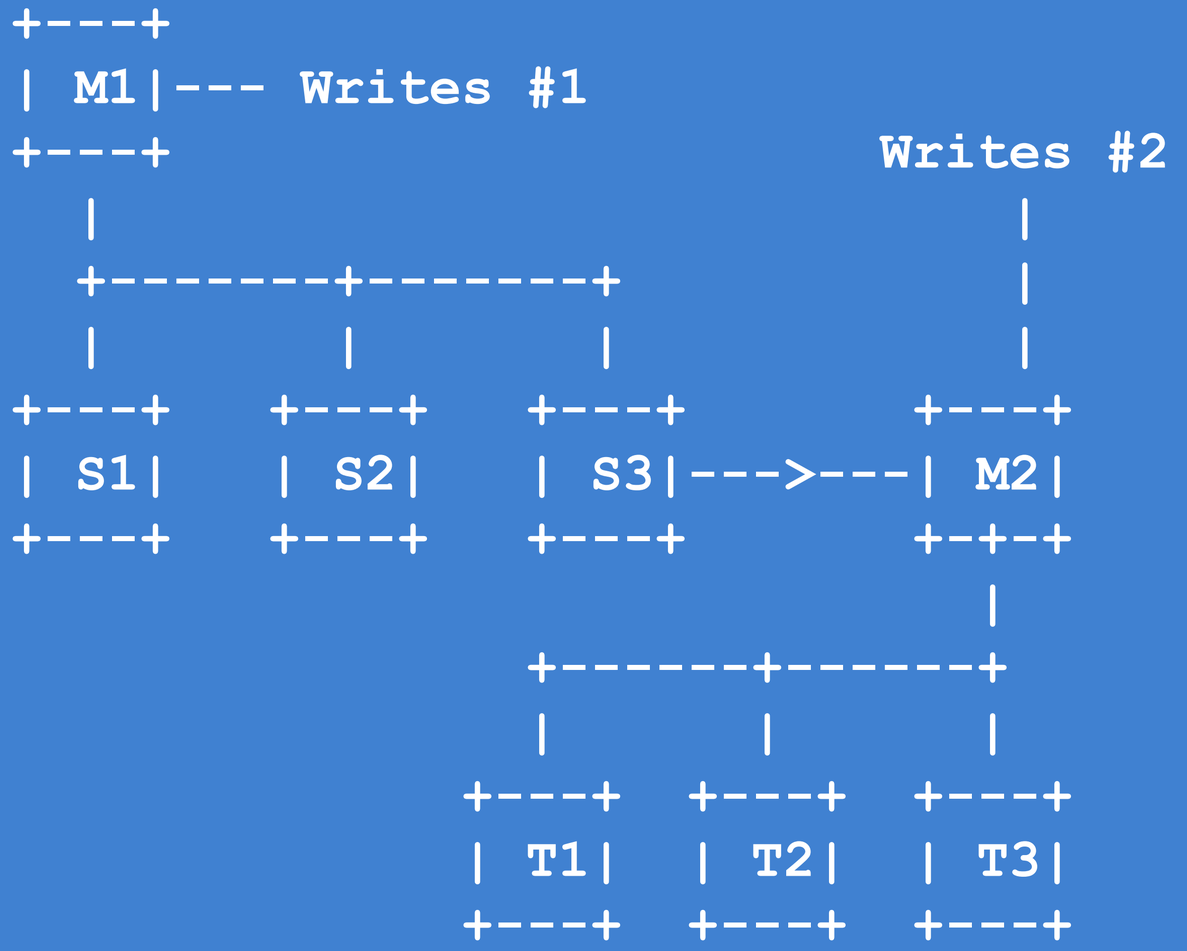


The *JFG* rules of replication (and failover) [4 of 4]

(MySQL Scalability and Reliability for Replication – do.bcn 2019)

This is ok:





Thoughts on Intermediate Masters

(MySQL Scalability and Reliability for Replication – do.bcn 2019)

Intermediate Masters (and `log-slave-updates`) introduce many problems:

- They slow-down replication
- They increase the risk of propagating errant transactions
- They are an expensive solution to artificial problems
- And more…

Binlog Servers are a better alternative
for most (all?) Intermediate Masters use cases

https://jfg-mysql.blogspot.com/2015/10/binlog-servers-for-backups-and-point-in-time-recovery.html

(also includes links to other use cases like simple failover and scaling)

Conclusion [1 of 2]

(MySQL Scalability and Reliability for Replication – do.bcn 2019)

- Replication is a wide subject
- I hope you are now equipped to dive deeper in the topics you need

- **`FOREIGN KEY`**s at scale: **`DELETE/UPDATE CASCADE`** → big/long trx ☹
- The downsides of ***`AUTO_INCREMENT`***s: hard data reconciliation

- But I did not talk much about
  - **`UNIQUE KEY`**s at scale
  - the limits of archiving

Conclusion [2 of 2]

(MySQL Scalability and Reliability for Replication – do.bcn 2019)

Two types of sharding:

- Vertical (Split): some schemas/tables above, some below
- Horizontal: some data in shard #1, in #2, … #N (left to right)
  (Archiving is a type of horizontal sharding with N usually equal 2)

- Splits and archiving divide a problem by 2 or 3 → short term win
- **`UNIQUE`** and **`FOREIGN KEY`**s are hard to enforce on a sharded dataset

Links [1 of 5]

(MySQL Scalability and Reliability for Replication – do.bcn 2019)

- Parts of this talk are inspired by some of Nicolai Plum's talks:
  - Booking.com: Evolution of MySQL System Design (PLMCE 2015)
    https://www.percona.com/live/mysql-conference-2015/sessions/bookingcom-evolution-mysql-system-design
  - MySQL Scalability: Dimensions, Methods and Technologies (PLDPC 2016)
    https://www.percona.com/live/data-performance-conference-2016/sessions/mysql-scalability-dimensions-methods-and-technologies
  - Datastore axes: Choosing the scalability direction you need (PLAMS 2016)
    https://www.percona.com/live/plam16/sessions/datastore-axes-choosing-scalability-direction-you-need
- Combined with material from some of previous talks:
  - Demystifying MySQL Replication Crash Safety (PLEU 2018 and HL18Moscow)
    https://www.slideshare.net/JeanFranoisGagn/demystifying-mysql-replication-crash-safety
  - The Full MySQL Parallel Replication Tutorial (PLSC 2018 and many others)
    https://www.slideshare.net/JeanFranoisGagn/the-full-mysql-and-mariadb-parallel-replication-tutorial
  - How B.com avoids and deals with replication lag (MariaDB Dev meeting 2017 and more)
    https://www.slideshare.net/JeanFranoisGagn/how-bookingcom-avoids-and-deals-with-replication-lag-74772032
  - Autopsy of a [MySQL] automation disaster (FOSDEM 2017)
    https://www.slideshare.net/JeanFranoisGagn/autopsy-of-an-automation-disaster

Links [2 of 5]

(MySQL Scalability and Reliability for Replication – do.bcn 2019)

Some posts about advanced replication topics:

- The five things to consider when implementing master high-availability
  https://jfg-mysql.blogspot.com/2019/02/mysql-master-high-availability-and-failover-more-thoughts.html
- Binlog Servers for Simplifying Point in Time Recovery
  (also includes links to other use cases like simple failover and scaling)
  https://jfg-mysql.blogspot.com/2015/10/binlog-servers-for-backups-and-point-in-time-recovery.html
- On the consequences of sync_binlog != 1 (part #1)
  https://jfg-mysql.blogspot.com/2018/10/consequences-sync-binlog-neq-1-part-1.html
- Question about Semi-Synchronous Replication: the Answer with All the Details
  https://www.percona.com/community-blog/2018/08/23/question-about-semi-synchronous-replication-answer-with-all-the-details/
- The danger of no Primary Key when replicating in RBR
  https://jfg-mysql.blogspot.com/2017/08/danger-no-pk-with-RBR-and-mariadb-protection.html
- Better Parallel Replication for MySQL
  https://medium.com/booking-com-infrastructure/better-parallel-replication-for-mysql-14e2d7857813

Links [3 of 5]

(MySQL Scalability and Reliability for Replication – do.bcn 2019)

More posts about an advanced replication topic (parallel replication):

- A Metric for Tuning Parallel Replication in MySQL 5.7
  https://jfg-mysql.blogspot.com/2017/02/metric-for-tuning-parallel-replication-mysql-5-7.html
- Evaluating MySQL Parallel Replication Part 4: More Benchmarks in Production
  https://medium.com/booking-com-infrastructure/evaluating-mysql-parallel-replication-part-4-more-benchmarks-in-production-49ee255043ab
  (links to other parts of the series in the post)
- An update on Write Set (parallel replication) bug fix in MySQL 8.0
  https://jfg-mysql.blogspot.com/2018/01/an-update-on-write-set-parallel-replication-bug-fix-in-mysql-8-0.html

Post on related subjects:

- Why I wrote "do not ignore warnings" and "always investigate/fix warnings”
  https://jfg-mysql.blogspot.com/2017/01/do-not-ignore-warnings-in-mysql-mariadb.html

Links [4 of 5]

(MySQL Scalability and Reliability for Replication – do.bcn 2019)

Other interesting posts:

- MySQL Ripple: The First Impression of a MySQL Binlog Server
  https://www.percona.com/blog/2019/03/15/mysql-ripple-first-impression-of-mysql-binlog-server/
- Why MySQL's binlog-do-db option is dangerous
  https://www.percona.com/blog/2009/05/14/why-mysqls-binlog-do-db-option-is-dangerous/
- MySQL ring replication: Why it is a bad option
  https://www.percona.com/blog/2014/10/07/mysql-ring-replication-why-it-is-a-bad-option/
- Orchestrator and Pseudo-GTID:
  https://github.com/github/orchestrator/blob/master/docs/pseudo-gtid.md
- ProxySQL GTID consistent reads:
  https://proxysql.com/blog/proxysql-gtid-causal-reads
- MySQL High Availability at GitHub
  https://githubengineering.com/mysql-high-availability-at-github/
- GitHub October 21 post-incident analysis
  https://blog.github.com/2018-10-30-oct21-post-incident-analysis/

Links [5 of 5]

(MySQL Scalability and Reliability for Replication – do.bcn 2019)

Other interesting tools, etc…:

- Percona Toolkit: https://www.percona.com/doc/percona-toolkit/LATEST/index.html
  - pt-heartbeat and pt-online-schema-change
  - pt-slave-find, pt-slave-restart, pt-table-checksum and pt-table-sync
  - pt-slave-delay (not needed since 5.6 and 10.2)
- **G**itHub **O**nline **S**chema Migra**t**ion/gh-ost: https://github.com/github/gh-ost
- Monotonically increasing UUID
  http://mysql.rjweb.org/doc.php/uuid
  https://www.percona.com/blog/2014/12/19/store-uuid-optimized-way/
- Query Optimization: The Basics https://dbahire.com/pleu18/
- About sharding:
  - Vitess: https://vitess.io/
  - Spider: https://mariadb.com/kb/en/library/spider-storage-engine-overview/
  - MySQL Cluster: https://www.mysql.com/products/cluster/

Thanks !

by Jean-François Gagné - *Presented at* dataops.barcelona (June 2019)
Senior Infrastructure Engineer / System and MySQL Expert
jeanfrancois AT messagebird DOT com / @jfg956