Booking.com

Pseudo-GTID and Easy MySQL Replication Management

Shlomi Noach

Percona Live, April 2015

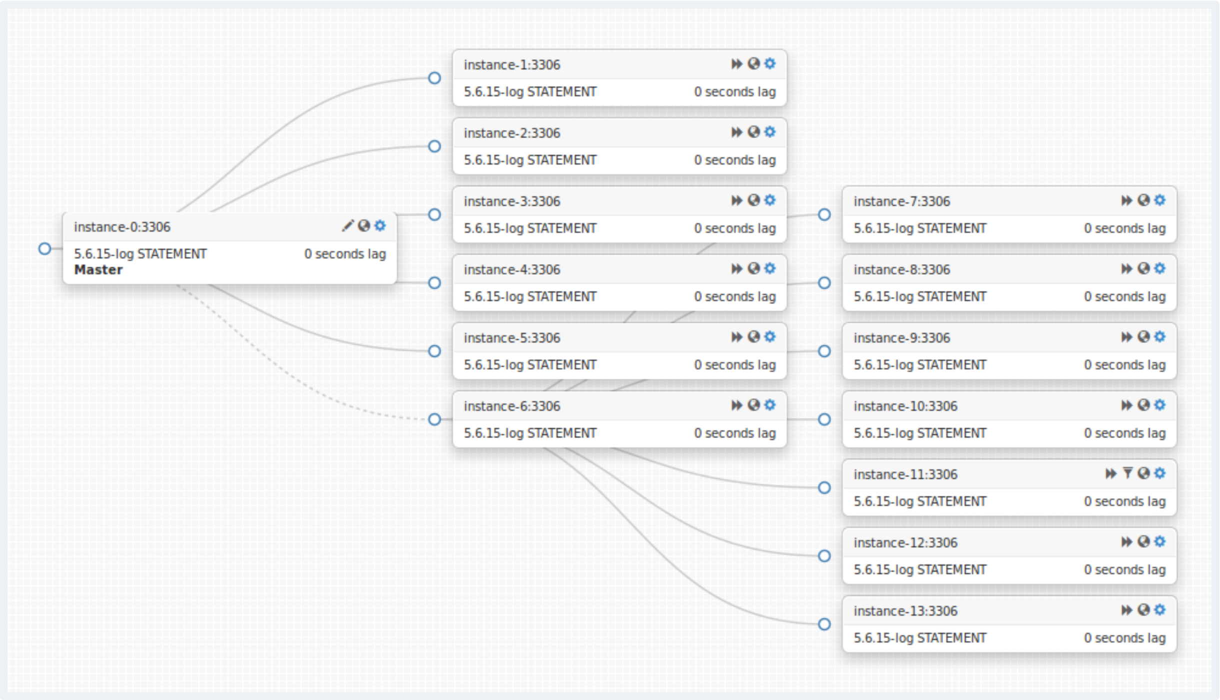
Overview:

- What? Why?
- Replication topologies, types
- Binary & relay logs
- GTID
- Pseudo GTID
- Failover with Pseudo GTID, bulk operations
- Orchestrator
- Pseudo GTID & orchestrator @ Booking.com
- Demo
- Considerations, gotchas & limitations

What? Why?

- Be happy!
- Avoid using GTID. Pseudo GTID offers what GTID offers, without GTID. This includes:
  - Slave repointing
  - Failover schemes
  - With less requirements
  - And, with larger topologies: faster!
- Without upgrading your servers; without installing anything on them; in short: not touching your beloved existing setup
- No vendor lockdown; no migration paths

MySQL replication topologies



instance-0:3306
5.6.15-log STATEMENT 0 seconds lag
Master

instance-1:3306
5.6.15-log STATEMENT 0 seconds lag

instance-2:3306
5.6.15-log STATEMENT 0 seconds lag

instance-3:3306
5.6.15-log STATEMENT 0 seconds lag

instance-4:3306
5.6.15-log STATEMENT 0 seconds lag

instance-5:3306
5.6.15-log STATEMENT 0 seconds lag

instance-6:3306
5.6.15-log STATEMENT 0 seconds lag

instance-7:3306
5.6.15-log STATEMENT 0 seconds lag

instance-8:3306
5.6.15-log STATEMENT 0 seconds lag

instance-9:3306
5.6.15-log STATEMENT 0 seconds lag

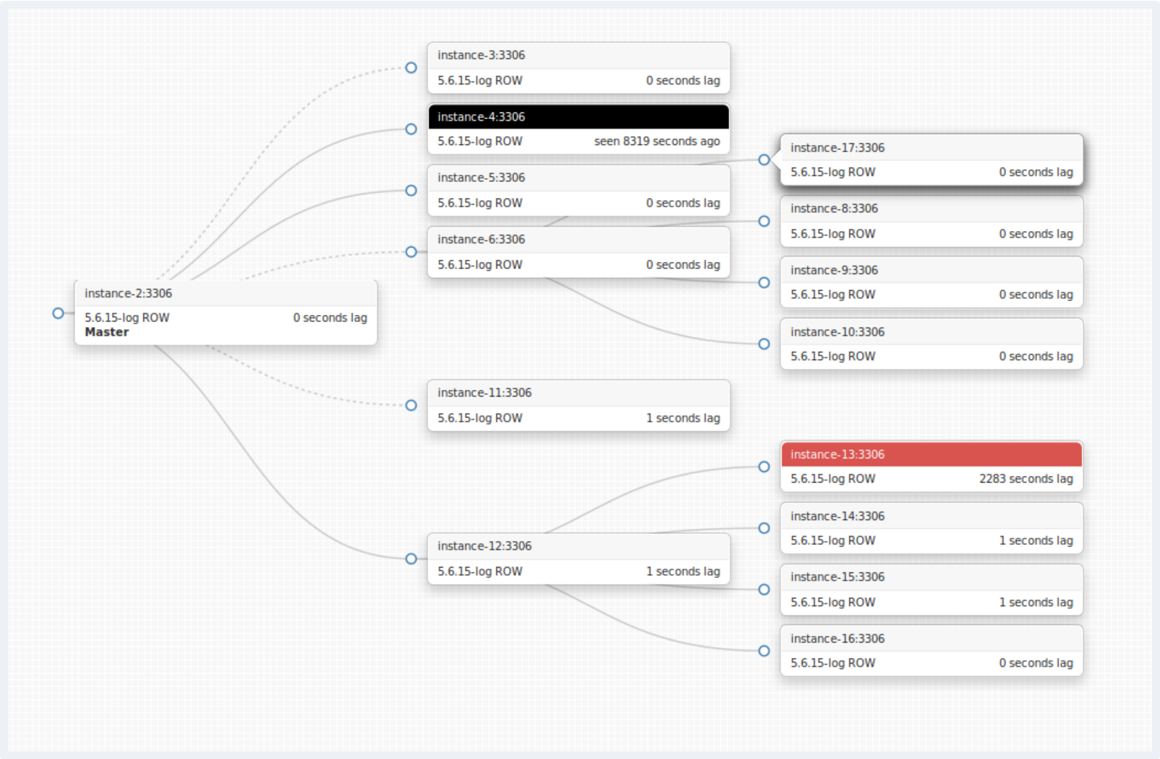
instance-10:3306
5.6.15-log STATEMENT 0 seconds lag

instance-11:3306
5.6.15-log STATEMENT 0 seconds lag

instance-12:3306
5.6.15-log STATEMENT 0 seconds lag

instance-13:3306
5.6.15-log STATEMENT 0 seconds lag

More complex topologies



instance-3:3306
5.6.15-log ROW 0 seconds lag

instance-4:3306
5.6.15-log ROW seen 8319 seconds ago

instance-5:3306
5.6.15-log ROW 0 seconds lag

instance-6:3306
5.6.15-log ROW 0 seconds lag

instance-17:3306
5.6.15-log ROW 0 seconds lag

instance-8:3306
5.6.15-log ROW 0 seconds lag

instance-9:3306
5.6.15-log ROW 0 seconds lag

instance-10:3306
5.6.15-log ROW 0 seconds lag

instance-2:3306
5.6.15-log ROW 0 seconds lag
Master

instance-11:3306
5.6.15-log ROW 1 seconds lag

instance-13:3306
5.6.15-log ROW 2283 seconds lag

instance-14:3306
5.6.15-log ROW 1 seconds lag

instance-12:3306
5.6.15-log ROW 1 seconds lag

instance-15:3306
5.6.15-log ROW 1 seconds lag

instance-16:3306
5.6.15-log ROW 0 seconds lag

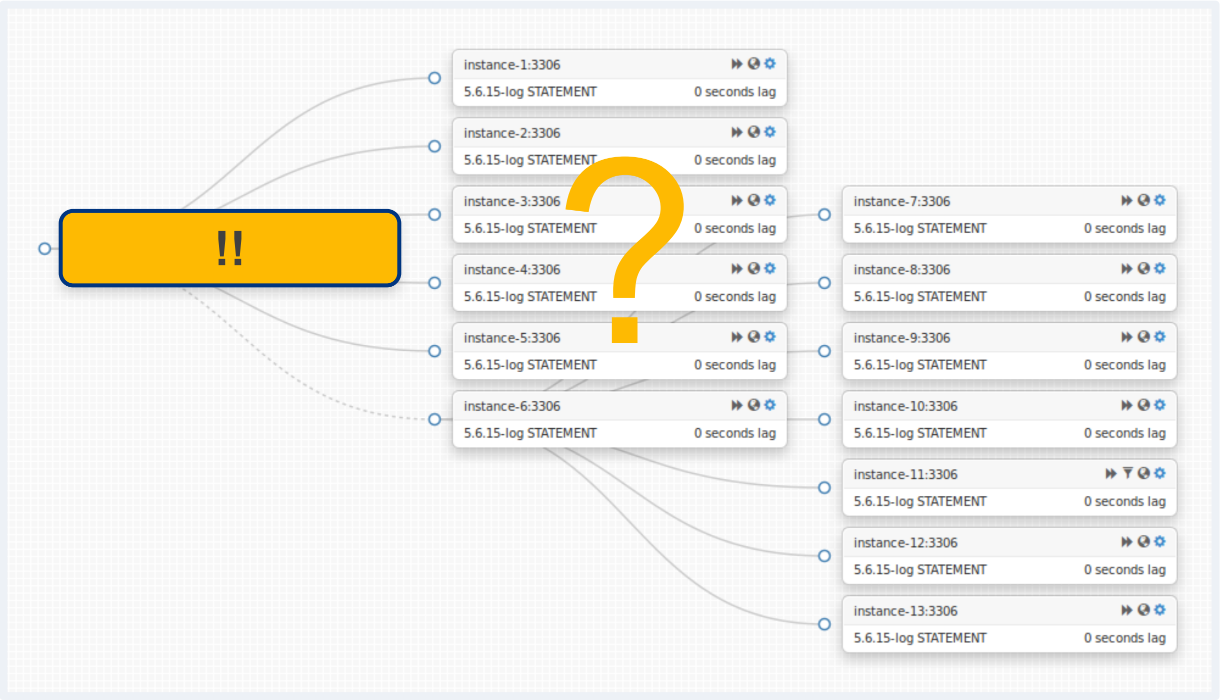
Replication topologies, “classic replication”

- Single master, multiple slaves
- Nested replication: slaves of slaves
- Replication load on master, on network
- Intermediate masters:
  - Upgrades
  - Schema changes
  - Switching datacenters
  - Experiments

Replication topologies, “classic replication”

- Too many slaves on a single master:
  - Can be too much load (network traffic, dedicated connections)
  - What happens when the master goes down?
- Using intermediate masters:
  - Reduced load
  - Accumulating slave lag
  - What happens when the intermediate master goes down?

Problem: master goes down



instance-1:3306
5.6.15-log STATEMENT 0 seconds lag

instance-2:3306
5.6.15-log STATEMENT 0 seconds lag

instance-3:3306
5.6.15-log STATEMENT 0 seconds lag

instance-4:3306
5.6.15-log STATEMENT 0 seconds lag

instance-5:3306
5.6.15-log STATEMENT 0 seconds lag

instance-6:3306
5.6.15-log STATEMENT 0 seconds lag

instance-7:3306
5.6.15-log STATEMENT 0 seconds lag

instance-8:3306
5.6.15-log STATEMENT 0 seconds lag

instance-9:3306
5.6.15-log STATEMENT 0 seconds lag

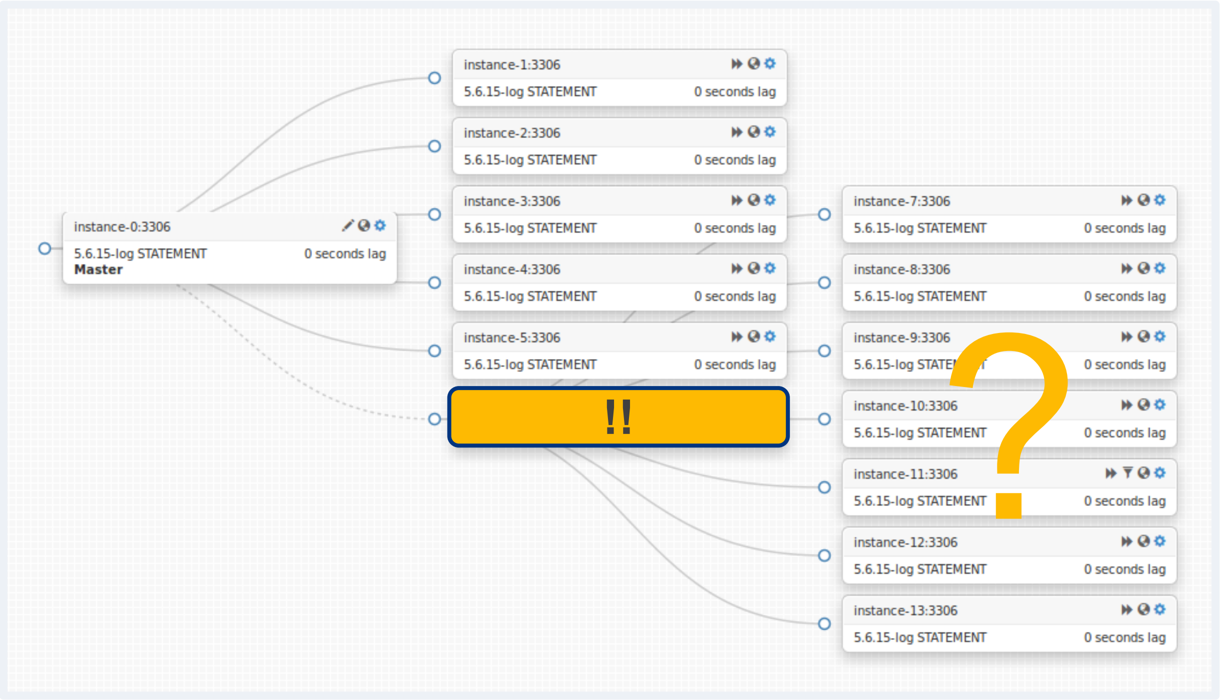
instance-10:3306
5.6.15-log STATEMENT 0 seconds lag

instance-11:3306
5.6.15-log STATEMENT 0 seconds lag

instance-12:3306
5.6.15-log STATEMENT 0 seconds lag

instance-13:3306
5.6.15-log STATEMENT 0 seconds lag

Problem: intermediate master goes down



instance-0:3306
5.6.15-log STATEMENT 0 seconds lag
Master

instance-1:3306
5.6.15-log STATEMENT 0 seconds lag

instance-2:3306
5.6.15-log STATEMENT 0 seconds lag

instance-3:3306
5.6.15-log STATEMENT 0 seconds lag

instance-4:3306
5.6.15-log STATEMENT 0 seconds lag

instance-5:3306
5.6.15-log STATEMENT 0 seconds lag

instance-7:3306
5.6.15-log STATEMENT 0 seconds lag

instance-8:3306
5.6.15-log STATEMENT 0 seconds lag

instance-9:3306
5.6.15-log STATE 0 seconds lag

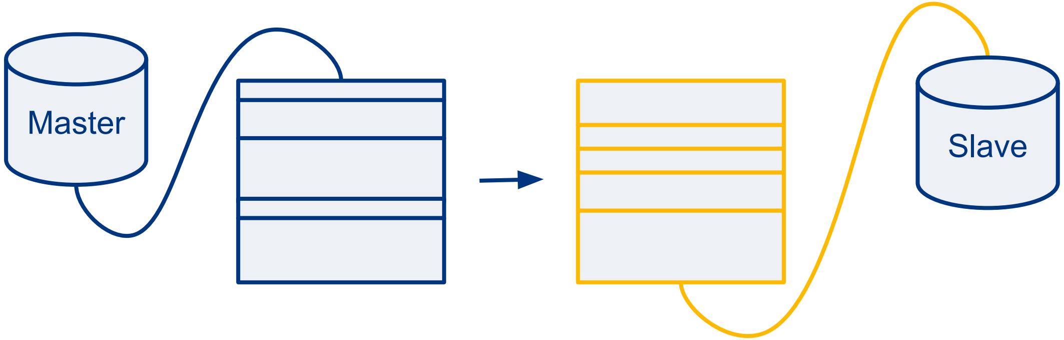
instance-10:3306
5.6.15-log STATEMENT 0 seconds lag

instance-11:3306
5.6.15-log STATEMENT 0 seconds lag

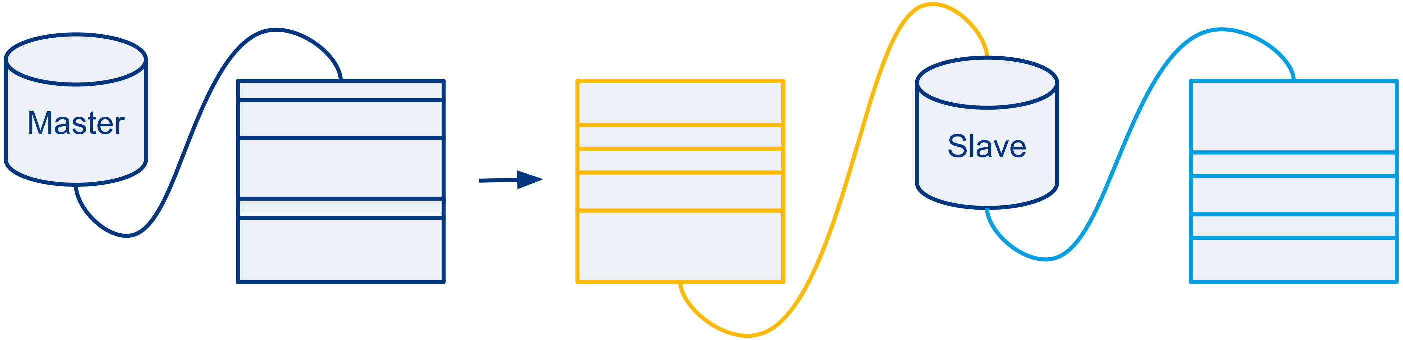
instance-12:3306
5.6.15-log STATEMENT 0 seconds lag

instance-13:3306
5.6.15-log STATEMENT 0 seconds lag

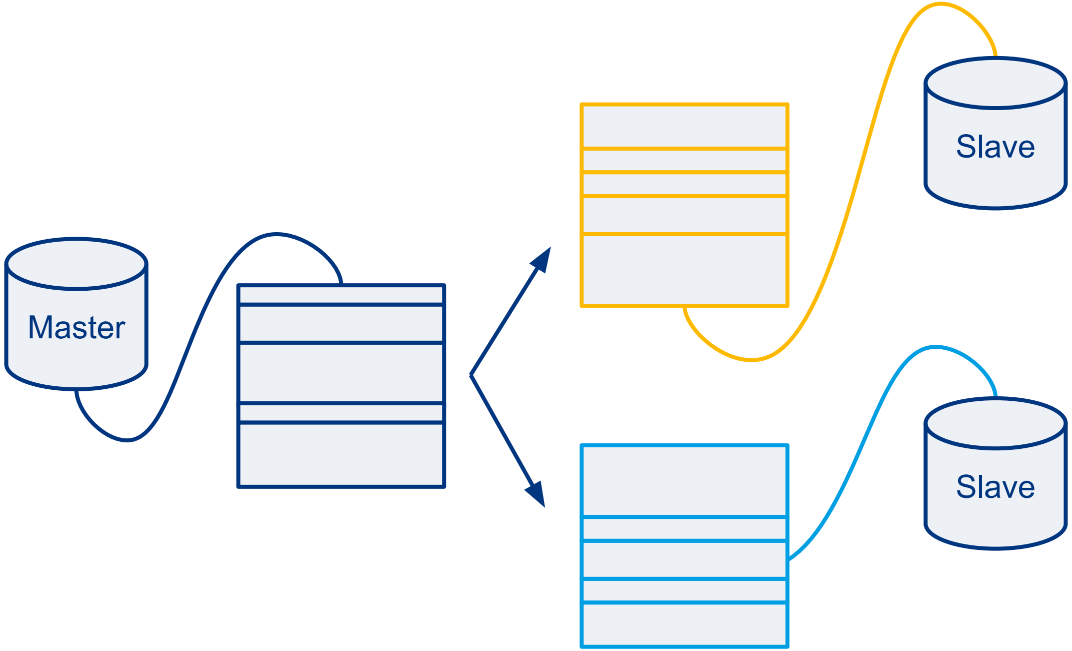
MySQL binary & relay logs



MySQL binary & relay logs: different languages



MySQL binary & relay logs: even more languages



GTID

- Every transaction has a unique identifier
- When a slave connects to a master, it looks for the last GTID statement it already executed
- Available in Oracle MySQL 5.6, MariaDB 10.0
    - Completely different implementations; may cause lockup
    - 5.6 migration path is unacceptable
    - 5.6 requires binary logs & **log-slave-updates** enabled on all slaves
    - 5.6 issues with errant transactions, unexecuted sequences, …
    - 5.6 requires adaptation of tools / understanding
    - 5.6 GTID will be the requirement in future Oracle features
    - MariaDB GTID supports domains; easy to use

Pseudo GTID

- Application-side enhancement
- We inject a uniquely identified statement every X seconds. We call it Pseudo GTID.
- Pseudo GTID statements are searchable and identifiable in binary and relay logs
- Make for “markers” in the binary/relay logs
- Injection can be made via MySQL event scheduler or externally
- Otherwise non intrusive. No changes to topology/versions/methodologies

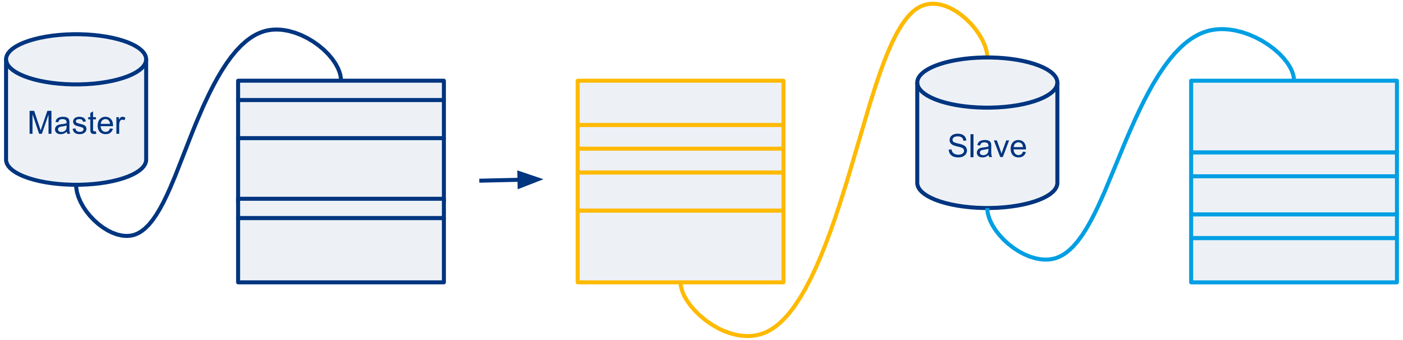
Injecting Pseudo-GTID

```
create event if not exists create_pseudo_gtid_event
  on schedule every 5 second starts current_timestamp
  on completion preserve enable
  do begin
      set @pseudo_gtid_hint := uuid();
      set @_create_statement := concat('drop ',
        'view if exists `meta`.`_pseudo_gtid_hint__', @pseudo_gtid_hint, '`');
      PREPARE st FROM @_create_statement;
      EXECUTE st;
      DEALLOCATE PREPARE st;
  end $$
```

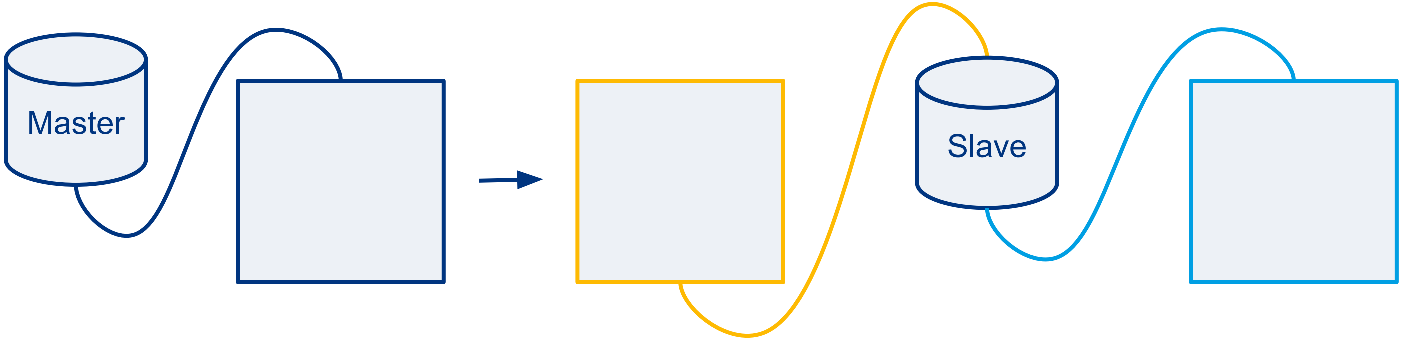
In the binary logs

```
mysql> show binlog events in 'mysql-bin.015631' \G
...
Log_name: mysql-bin.015631
Pos: 1632
Event_type: Query
Server_id: 1
End_log_pos: 1799
Info: use `meta`; drop view if exists `meta`.`_pseudo_gtid_hint__50731a22-9ca4-11e4-aec4-e25ec4bd144f`
...
```

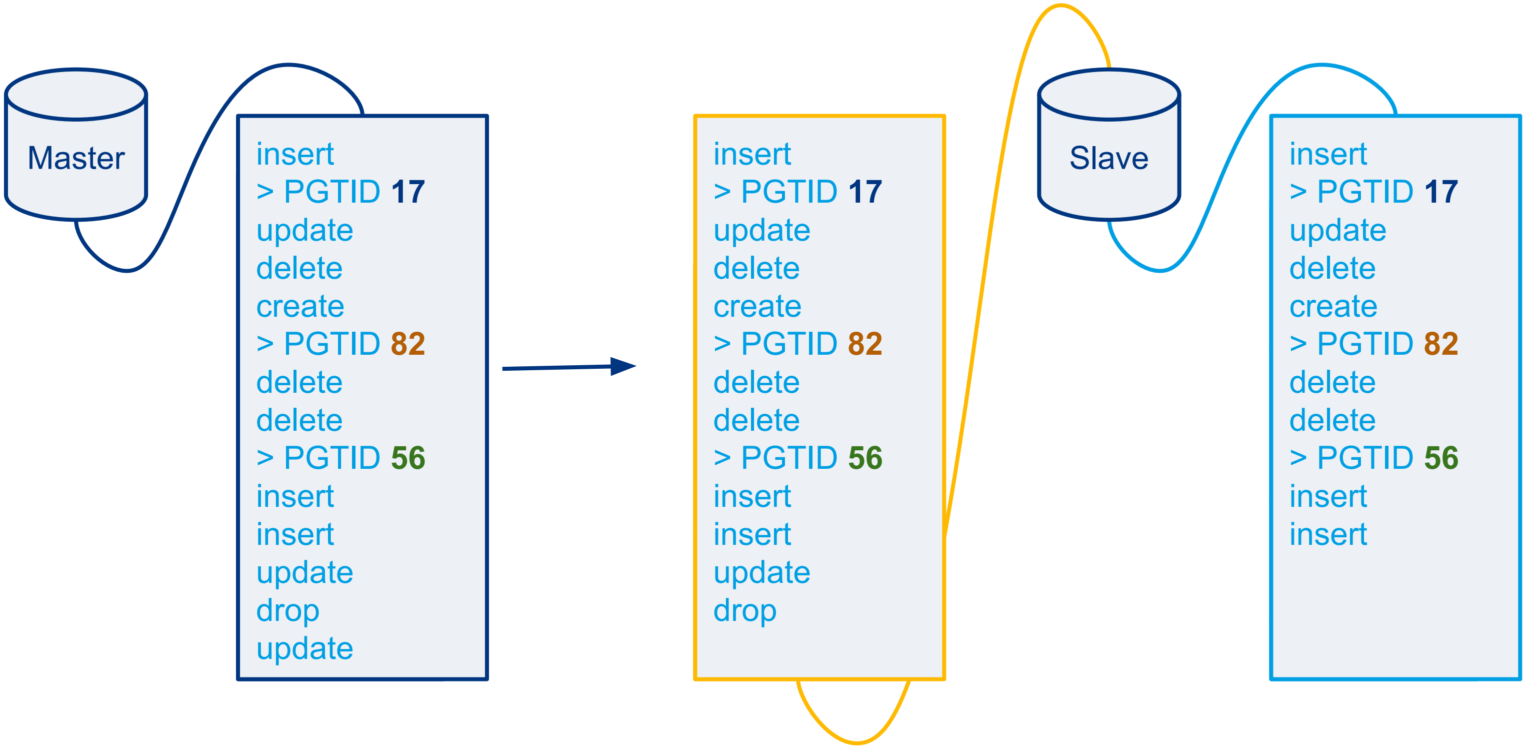
Recap: MySQL binary & relay logs



MySQL binary & relay logs: a virtual contiguous log file



MySQL binary & relay logs: Pseudo GTID injection



Master

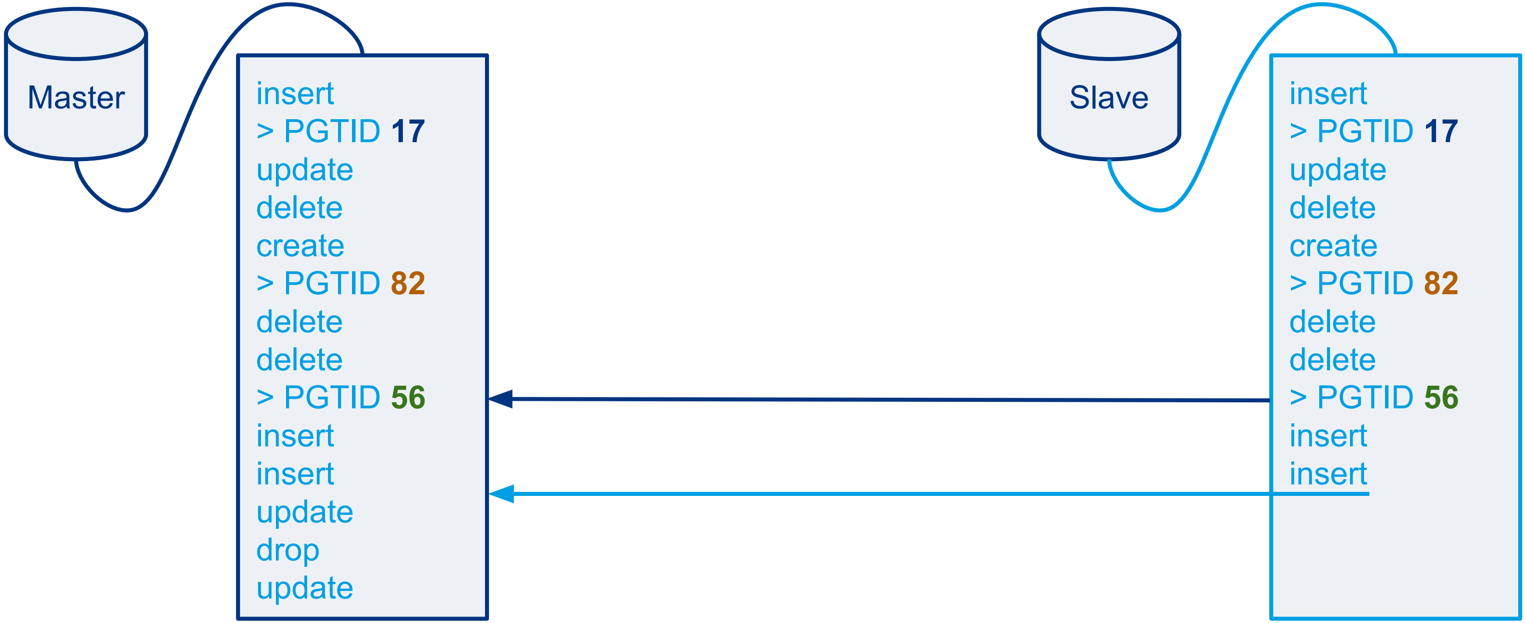
insert
> PGTID **17**
update
delete
create
> PGTID **82**
delete
delete
> PGTID **56**
insert
insert
update
drop
update

insert
> PGTID **17**
update
delete
create
> PGTID **82**
delete
delete
> PGTID **56**
insert
insert
update
drop

Slave

insert
> PGTID **17**
update
delete
create
> PGTID **82**
delete
delete
> PGTID **56**
insert
insert

Pseudo GTID: repoint, based on binary logs



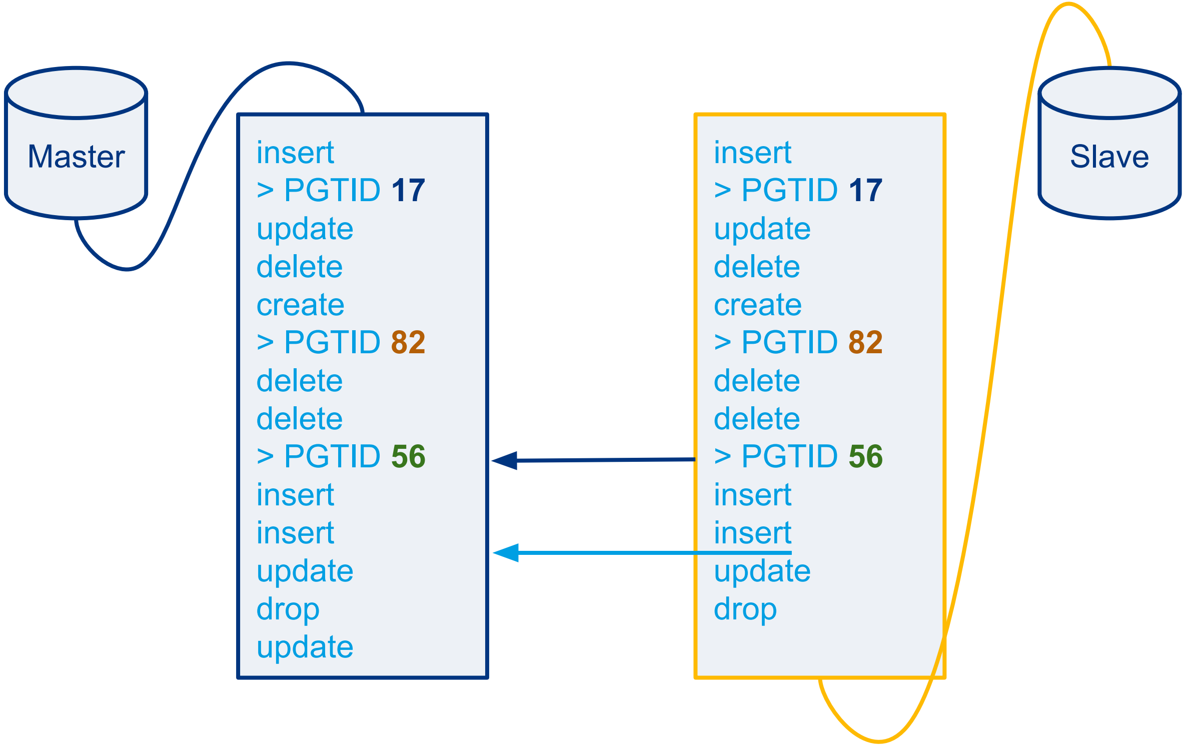
Master

insert
> PGTID **17**
update
delete
create
> PGTID **82**
delete
delete
> PGTID **56**
insert
insert
update
drop
update

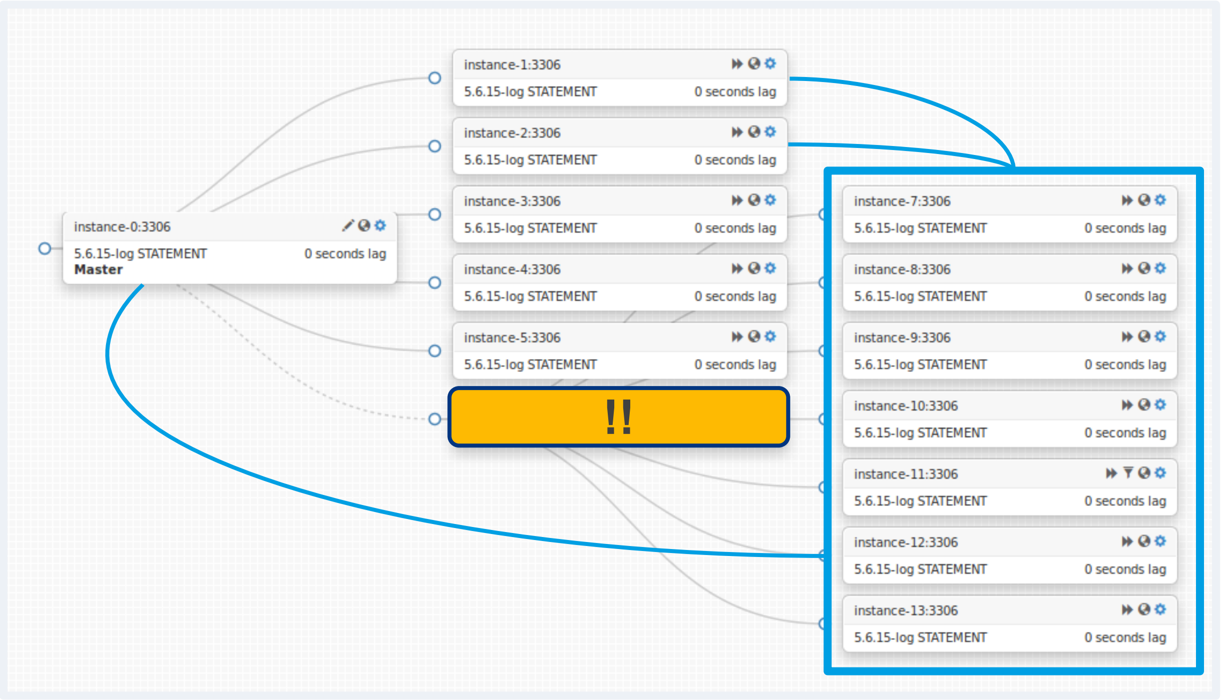
Slave

insert
> PGTID **17**
update
delete
create
> PGTID **82**
delete
delete
> PGTID **56**
insert
insert

Pseudo GTID: repoint, based on relay logs



Multiple possible destinations



instance-0:3306
5.6.15-log STATEMENT 0 seconds lag
Master

instance-1:3306
5.6.15-log STATEMENT 0 seconds lag

instance-2:3306
5.6.15-log STATEMENT 0 seconds lag

instance-3:3306
5.6.15-log STATEMENT 0 seconds lag

instance-4:3306
5.6.15-log STATEMENT 0 seconds lag

instance-5:3306
5.6.15-log STATEMENT 0 seconds lag

!!

instance-7:3306
5.6.15-log STATEMENT 0 seconds lag

instance-8:3306
5.6.15-log STATEMENT 0 seconds lag

instance-9:3306
5.6.15-log STATEMENT 0 seconds lag

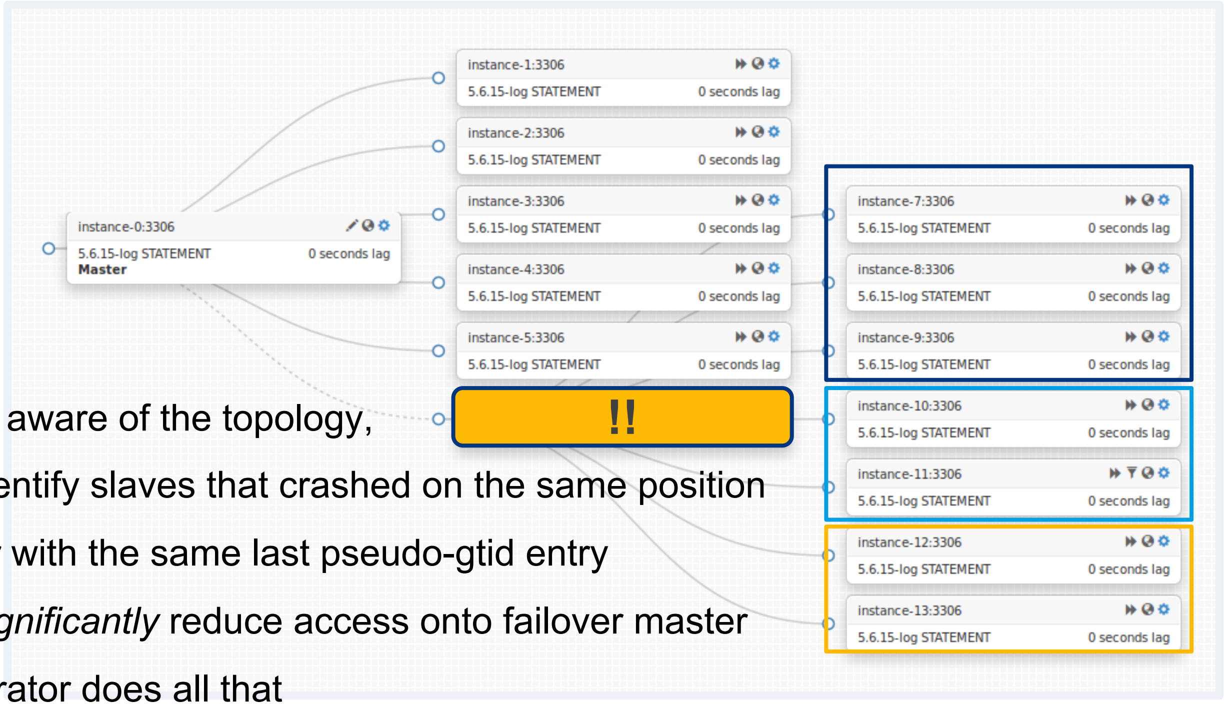
instance-10:3306
5.6.15-log STATEMENT 0 seconds lag

instance-11:3306
5.6.15-log STATEMENT 0 seconds lag

instance-12:3306
5.6.15-log STATEMENT 0 seconds lag

instance-13:3306
5.6.15-log STATEMENT 0 seconds lag

Bulk operations



- If you’re aware of the topology,
  - Identify slaves that crashed on the same position
  - Or with the same last pseudo-gtid entry
  - *Significantly* reduce access onto failover master
- Orchestrator does all that

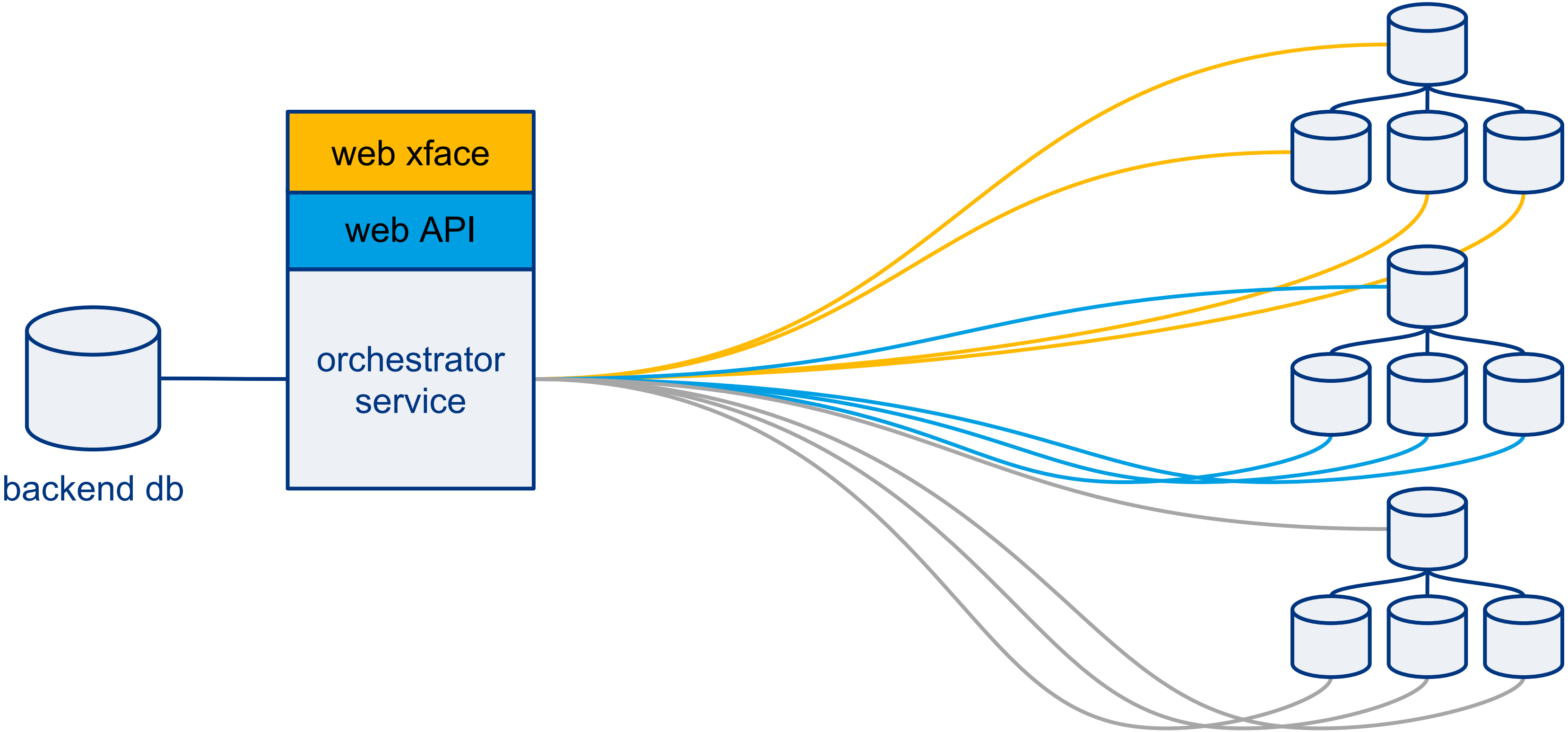
MySQL @ Booking.com

- We are a big MySQL shop
- We have >**2600** production servers (~3300 including experiments & tests) on >**110** topologies (aka chains, aka clusters)
  - As small as 1 server per topology, as large as 400 servers per topology
  - Two major data centers
- All chains are deployed with Pseudo-GTID and controlled by orchestrator

Orchestrator: MySQL replication management & visualization tool

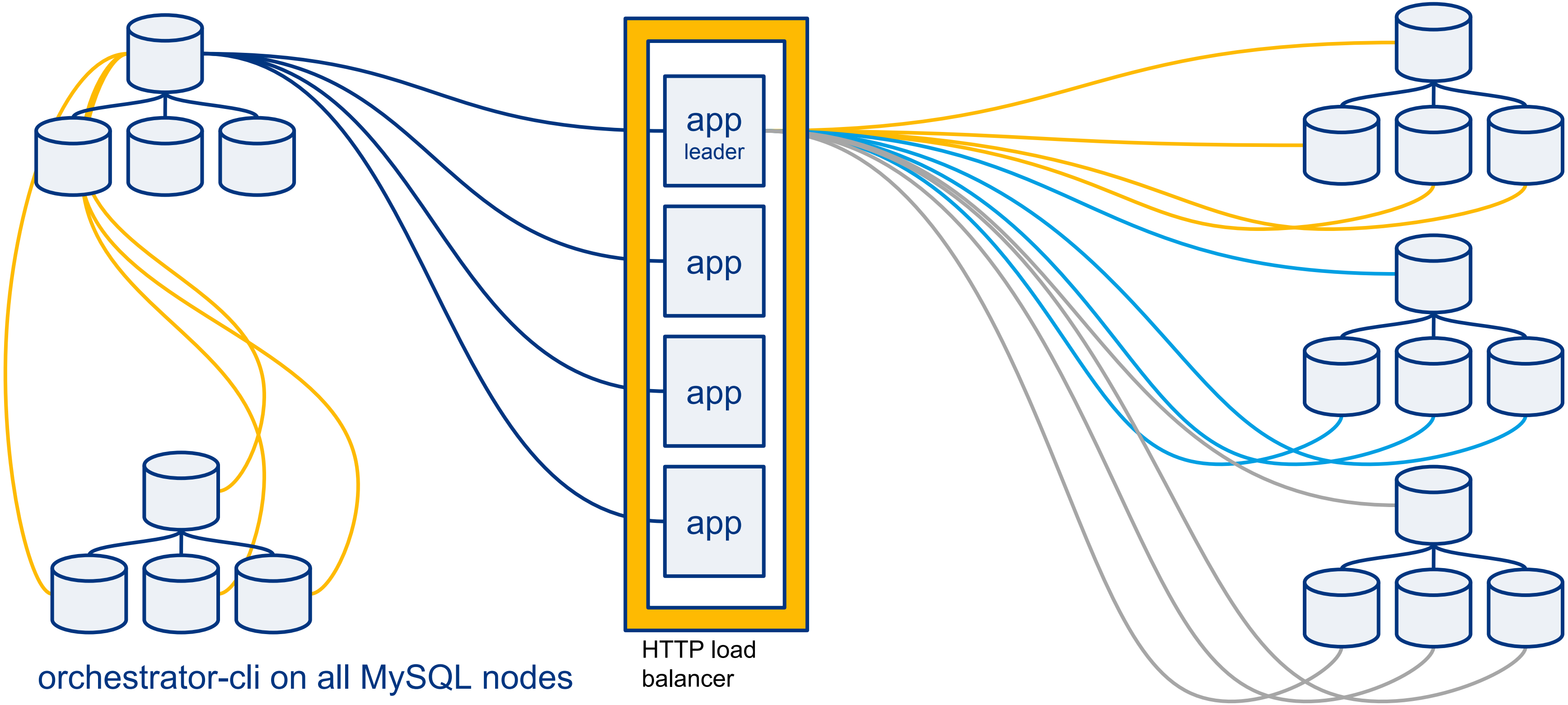
- command line, web API, web interface
- Crawls through your topologies, maps them, persists to backend database
- Understands replication, gathers metadata on replicating slaves (Which cluster? Depth?)
- Understands rules of replication (SBR, RBR, version compatibility, other configurations you wish you had never heard of)
- Can refactor/manipulate topologies
- Understands Pseudo-GTID
- Detects and recovers outage scenarios

Orchestrator general architecture



Booking.com

Orchestrator architecture @ Booking.com



orchestrator-cli on all MySQL nodes

HTTP load
balancer

Orchestrator stack & development

- Stack:
  - golang - in retrospect a very good choice: a lot of concurrency; easy deployment; rapid development
  - MySQL as backend database (duh)
  - go-martini web framework
  - Page generation via dirty JavaScript/jQuery (sue me)
  - Twitter bootstrap
  - Graphs via D3, integrated with bootstrap
- Development:
  - Github, completely open source; as generic as possible

    https://github.com/outbrain/orchestrator/

Live demo

In-production experiments, trust

- Tested:
  - 21,138 rematch experiments on 7 topologies (based on binlogs)
  - 13,872 rematch experiments on 6 topologies (based on relay logs)
  - 6,246 bounce up and back experiments on 6 topologies
  - 8,699 regroup, bounce up and back experiments on 9 topologies
  - ~180 intermediate master automated failover (clean shutdown)
  - A few dozens intermediate master automated failover (kill -9 / iptables)
  - Many intermediate master manual failovers
- Todo:
  - Daily (!) controlled intermediate master failover
  - Not so far in the future: daily (!) controlled master failover

Considerations, requirements

- Works with:
  - MySQL, MariaDB, using standard, single threaded replication
  - Supports SRB & RBR
  - Supports Binlog Servers
- When slave has **log-slave-updates** & **sync_binlog=1**, implies crash safe replication
- **log-slave-updates** required when slave should be considered to be promoted
- Otherwise relay logs work well
  - But change of master clears relay logs; an additional crash during < injection time may render the instance lost

Considerations, requirements

- Will not work with **5.6** per-schema-parallel-replication (no intended work on that)
- Will work with In-order binlog statements applier on slave (true in MariaDB and in MySQL **5.7.5** with **slave_preserve_commit_order**)
- No thoughts yet on multisource

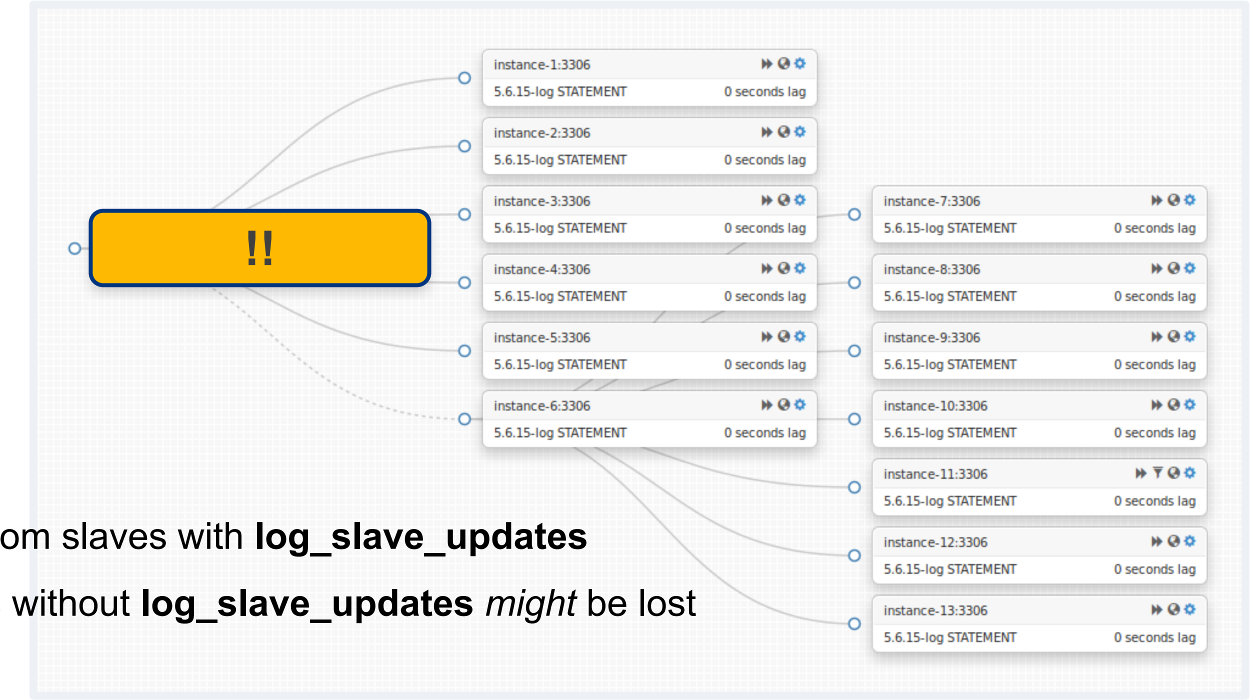
Considerations, requirements

- Allows for queries to execute on slave
  - But not after the last Pseudo-GTID entry
- Will succeed when:
  - Matching a slave up the topology
  - Matching below a sibling known to be more advanced
- Can succeed when:
  - Matching below an “uncle”/”cousin”/other relative
    - If not - then the opposite direction should work
- Cannot move slave underneath its own sibling (singularity, universe will collapse)
- Replication filters are your own risk

Considerations, requirements

- Therefore, can always recover the death of an intermediate master
  - (This is partly automated at Booking.com)
- Master death topology recovery possible when all immediate slaves have **log-slave-updates**
- Consider actually enforcing such a layer

Auto pick replacement master



!!

instance-1:3306
5.6.15-log STATEMENT 0 seconds lag

instance-2:3306
5.6.15-log STATEMENT 0 seconds lag

instance-3:3306
5.6.15-log STATEMENT 0 seconds lag

instance-4:3306
5.6.15-log STATEMENT 0 seconds lag

instance-5:3306
5.6.15-log STATEMENT 0 seconds lag

instance-6:3306
5.6.15-log STATEMENT 0 seconds lag

instance-7:3306
5.6.15-log STATEMENT 0 seconds lag

instance-8:3306
5.6.15-log STATEMENT 0 seconds lag

instance-9:3306
5.6.15-log STATEMENT 0 seconds lag

instance-10:3306
5.6.15-log STATEMENT 0 seconds lag

instance-11:3306
5.6.15-log STATEMENT 0 seconds lag

instance-12:3306
5.6.15-log STATEMENT 0 seconds lag

instance-13:3306
5.6.15-log STATEMENT 0 seconds lag

- Only from slaves with **log_slave_updates**
- Slaves without **log_slave_updates** *might* be lost

Considerations, requirements

- Recovery time depends on binary log parsing speed. Typically, you will need to search throughout the last binary logs
    - Reduce **max_binlog_size**, **max_relay_log_size**
    - Means more files
    - Orchestrator already tackled plenty issues involving scanning (many) binlog files

Gotchas, careful!

- **SHOW BINLOG EVENTS** lockdown! Keep chunk size small
  http://bugs.mysql.com/bug.php?id=76618
- Make sure Pseudo-GTID injected on master only
- **log-slave-updates** have I/O overhead; incurs more lag; experiments with **5.7** show reduces parallelism
- Replication filters may be a necessary evil -- but they *are* evil!
- Relay log purging is is not user-controlled

Further ideas

- Reduce binlog scan time by injecting the master’s binlog position (e.g. output of **SHOW MASTER STATUS**) within the Pseugo-GTID entry
    - This allows starting the scan from the given position
    - Likely to end quickly
    - Applies for masters only, not for intermediate masters
- Use monotonically increasing Pseudo-GTID values
    - Allows skipping of binary logs that begin with later/greater value than desired one
- Agents:
    - Index the binary logs
    - Full visibility even with RBR (mysqlbinlog more detailed than **SHOW BINLOG EVENTS**)

See also

- **Binlog Servers at Booking.com**
  Jean-François Gagné
  15 April 2:00PM - 2:50PM @ Ballroom G

- **Booking.com: Evolution of MySQL System Design**
  Nicolai Plum
  16 April 12:50PM - 1:40PM @ Ballroom E

Thank you!

Questions?

@ShlomiNoach

http://openark.org

http://blog.booking.com