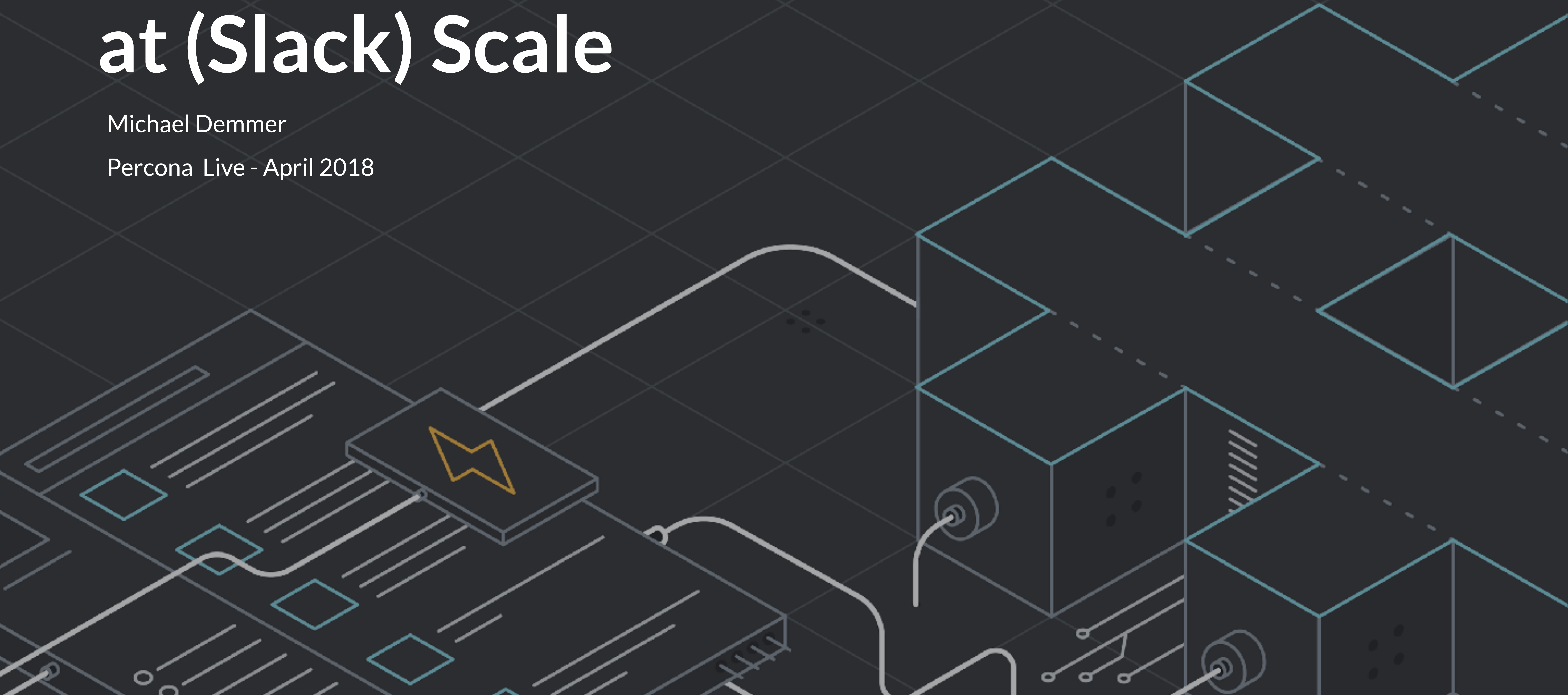
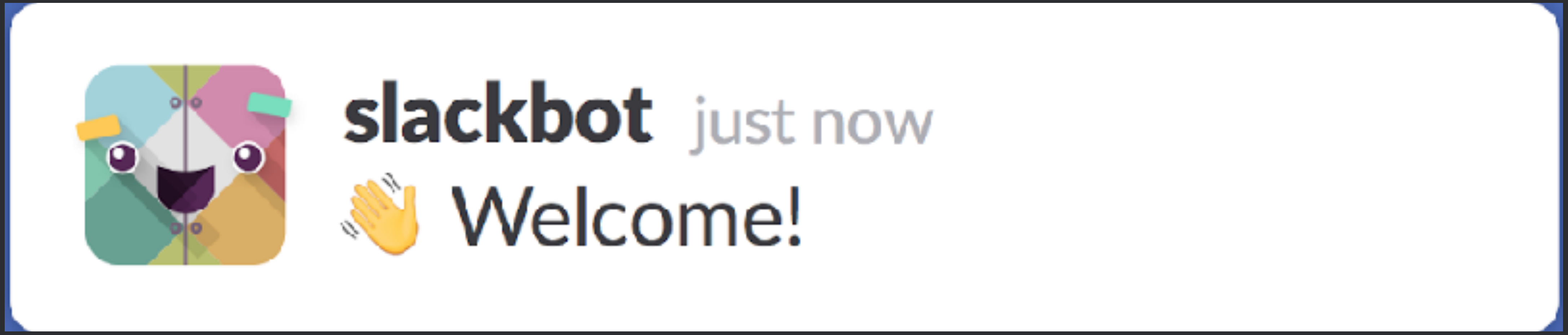
Migrating to Vitess at (Slack) Scale

Michael Demmer

Percona Live - April 2018





slackbot just now

👋 Welcome!

This is a (brief) story of how Slack's databases work today, why we're migrating to Vitess, and some lessons we've learned along the way. ✨

Michael Demmer

Senior Staff Engineer
Slack Infrastructure

- ~1.5 years at Slack, former startup junkie
- PhD in CS from UC Berkeley
- Long time interest in distributed systems
- (Fairly) new to databases

slack

Our Mission:

To make people’s working lives simpler, more pleasant,
and more productive.

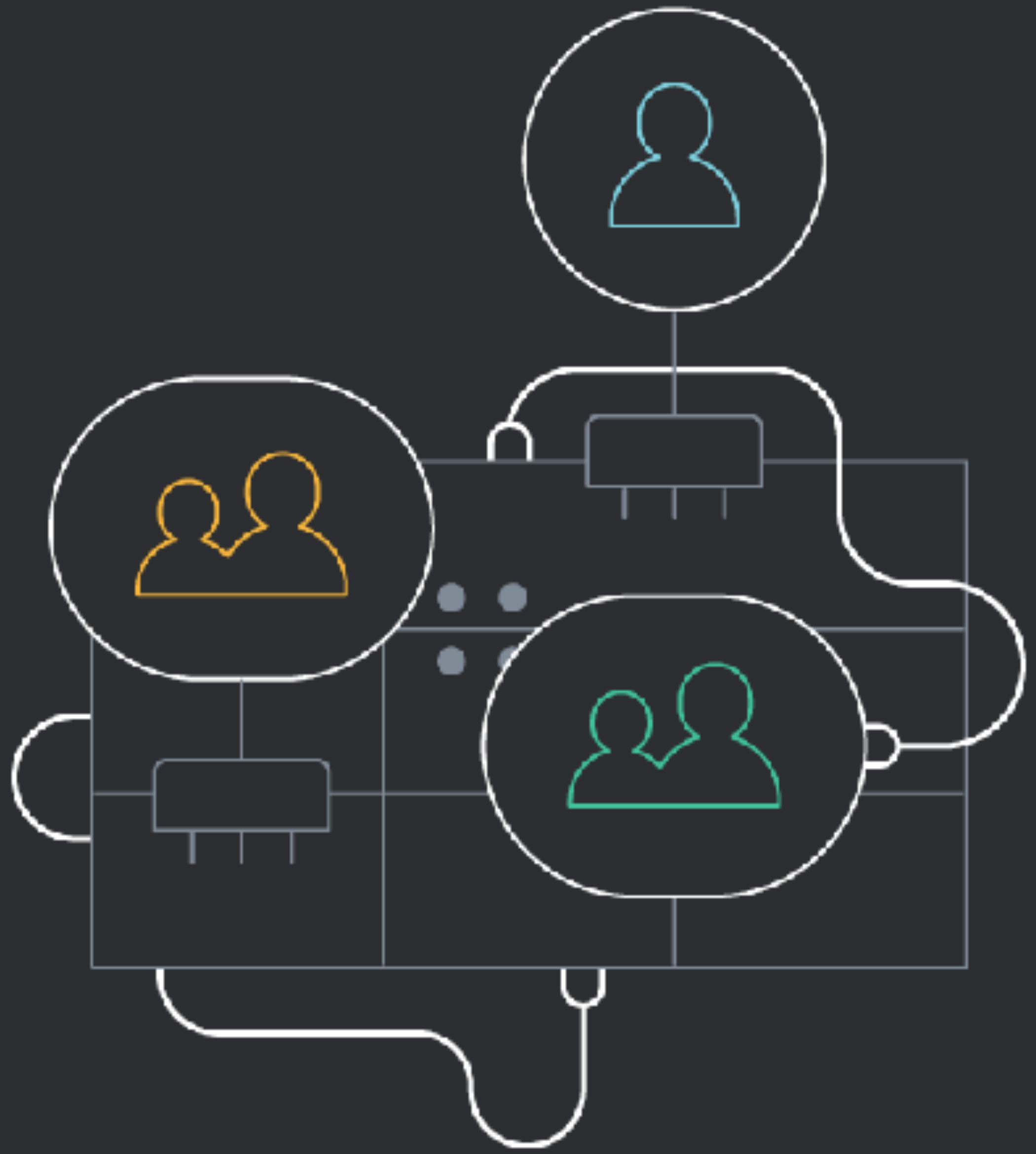
slack

- 9+ million weekly active users
- 4+ million simultaneously connected
- Average 10+ hours/ weekday connected

- $200M+ in annual recurring revenue
- 1000+ employees across 7 offices
- Customers include: Autodesk, Capital One, Dow Jones, EA, eBay, IBM, TicketMaster, Comcast

How Slack (Mostly) Works

Focusing on the MySQL parts



The Components

Linux

Apache

MySQL

HHVM

Real Time Messaging

Caching

The Components

Linux

Apache

MySQL

HHVM

Real Time Messaging

Caching

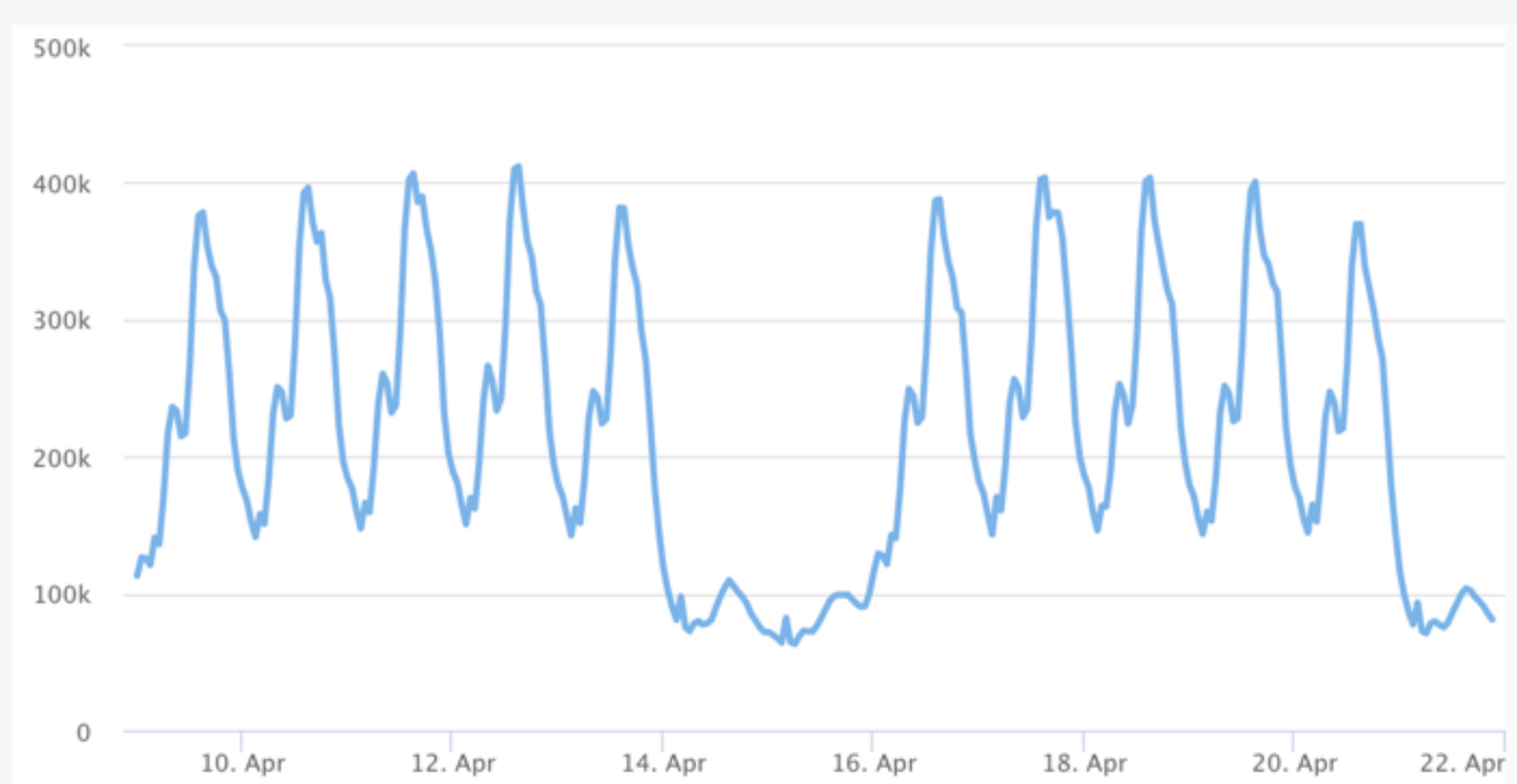
“Legacy” MySQL Numbers

Primary storage system for the Slack service
(File uploads in AWS S3)

~1400 database hosts

~100,000-400,000 QPS with very high bursts

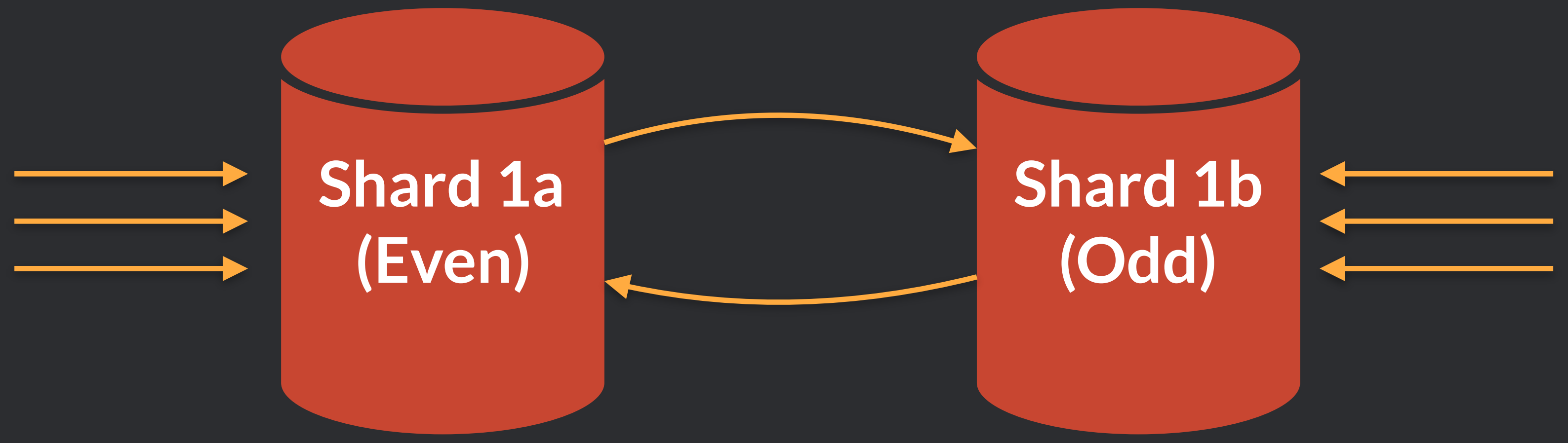
~24 billion queries / day



MySQL Details

- MySQL 5.6 (Percona Distribution)
- Run on AWS EC2 instances, no containers
- SSD-based instance storage (no EBS)
- Single region, multiple Availability Zones
- Webapp has many short-lived connections directly to mysql

Master / Master



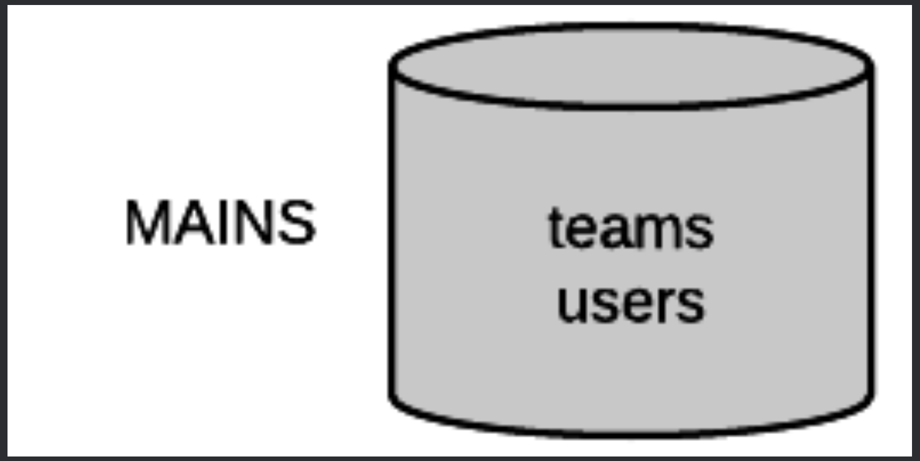
- Each is a *writable* master AND a replication slave of the other
- Fully async, statement-based replication, without GTIDs
- App *prefers* one "side" using *team_id % 2*, switches on failure
- Mitigate conflicts by using upsert, globally unique IDs, etc
- Yes, this is a bit odd... BUT it yields Availability >> Consistency

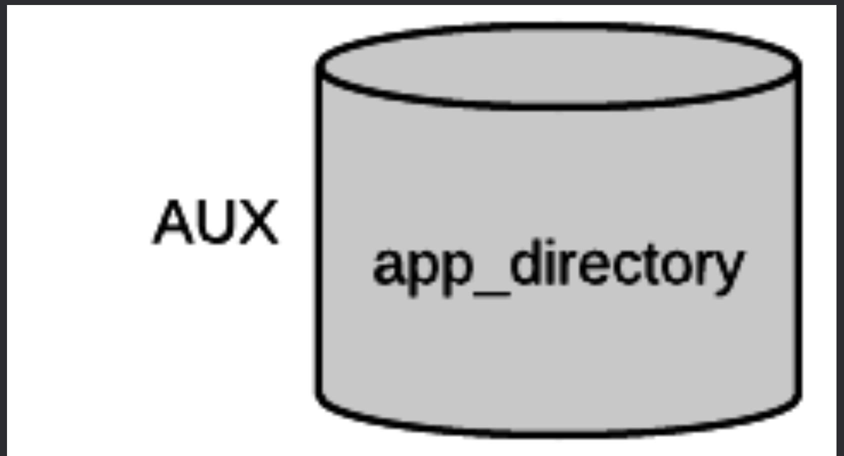
Sharding

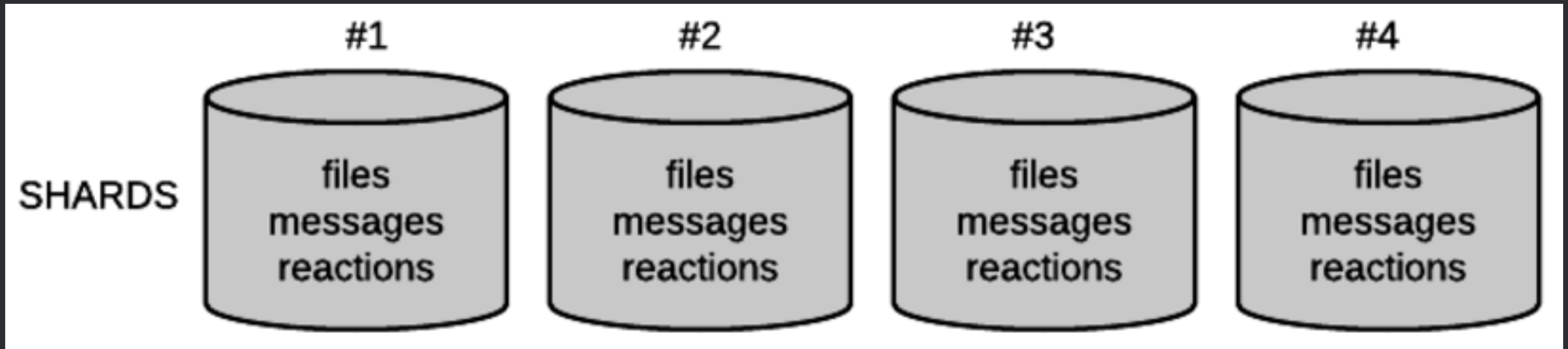
Workspace (aka "team") assigned to a shard at signup

App finds team:shard mapping in mains db

Globally Unique IDs via a dedicated service



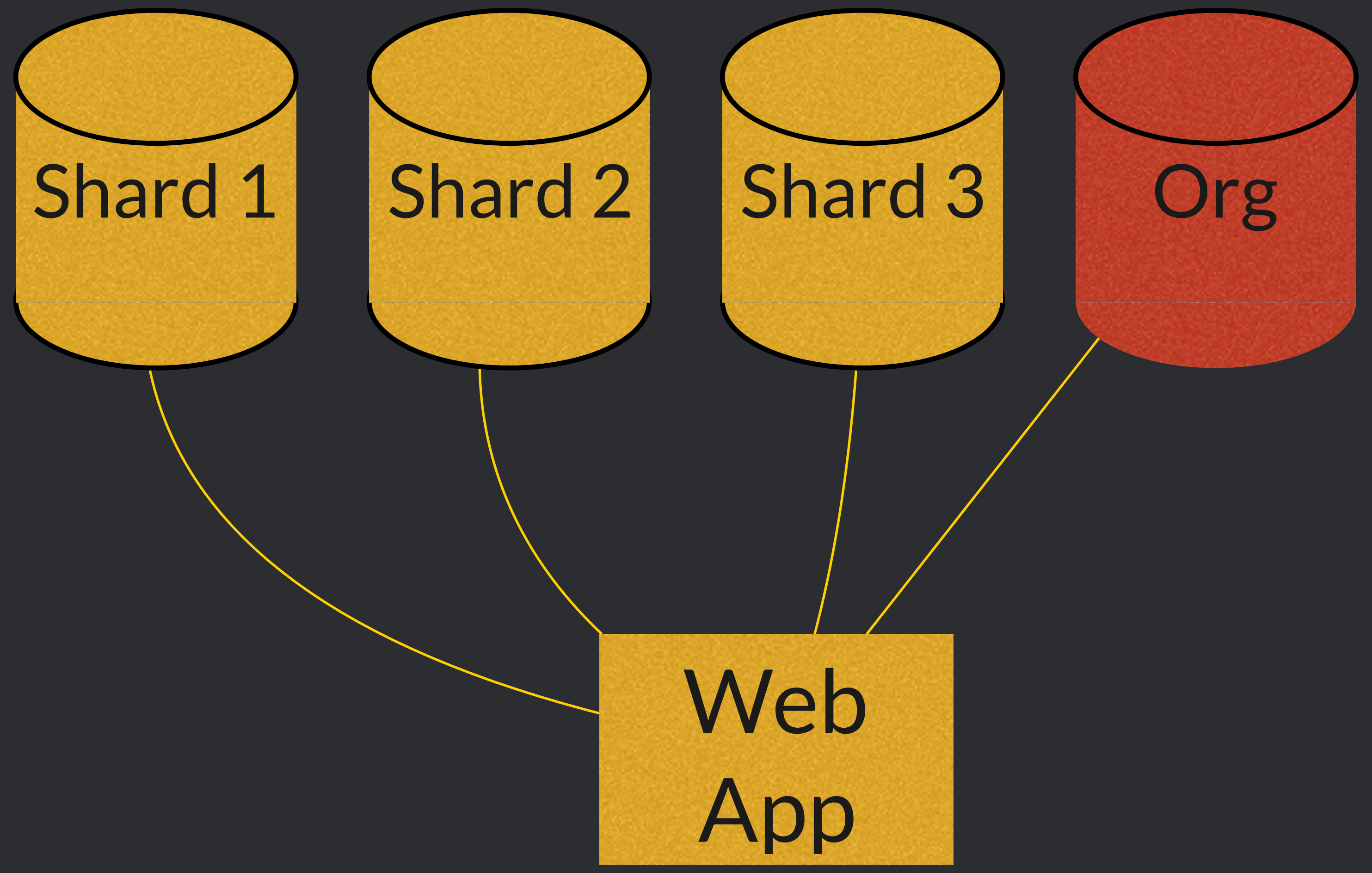




Added Complexity

Enterprise Grid:
Federate multiple workspaces into an org using N + 1 shards

Shared Channels:
Accessing across workspace shards



The Good Today

✓ Highly available for transient or permanent host failures

✓ Highly reliable with low rate of conflicts in practice

✓ Writes are as fast as a single node can accept

✓ Horizontally scale by splitting "hot" shards

✓ Can pin large teams to dedicated hosts

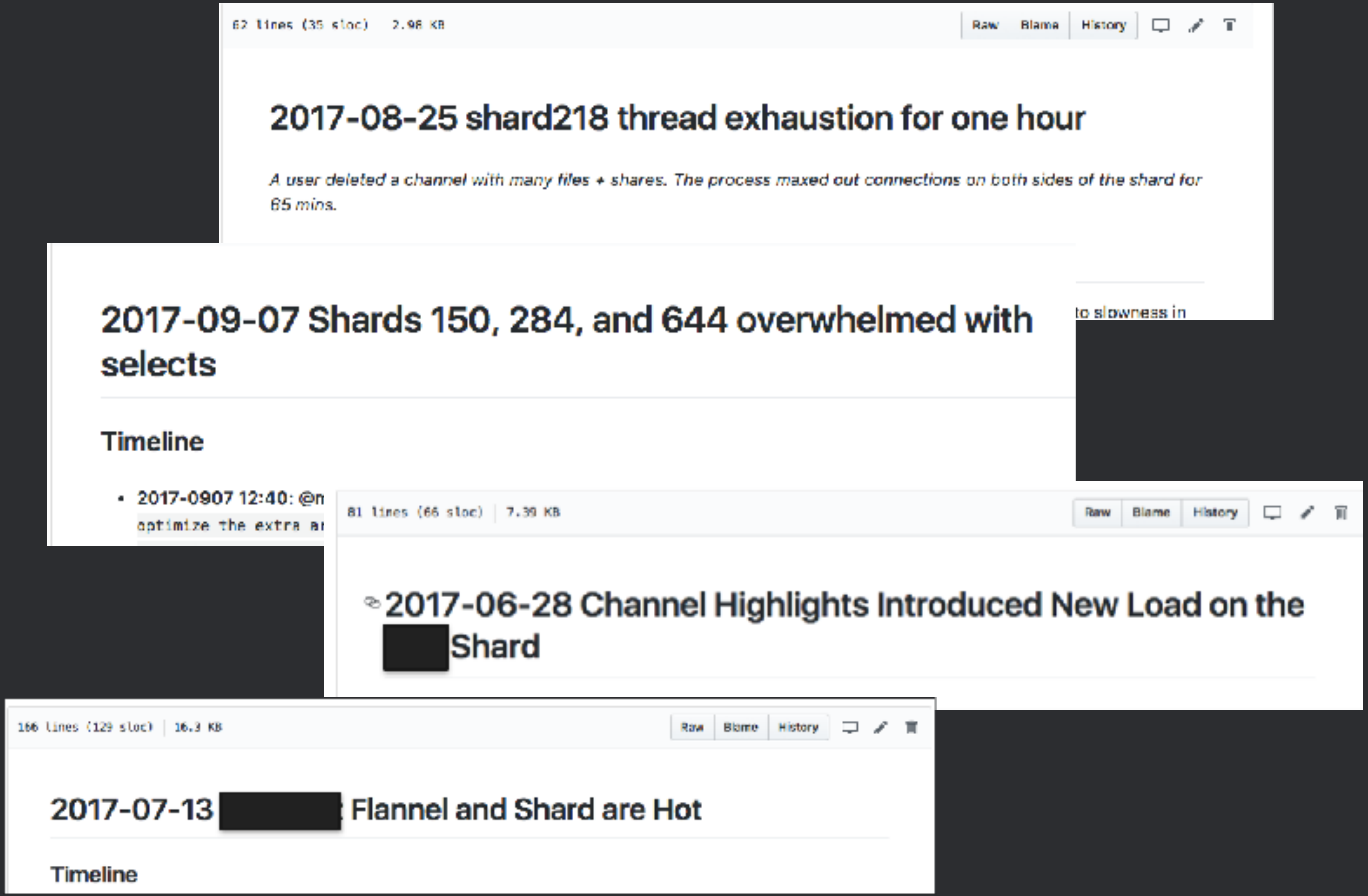
✓ Simple, well understood, easy to administer and debug

Challenges

Hot Spots

Large customers or unexpected usage concentrates load on a single shard

Can't scale up past the capabilities of a single database host


62 lines (35 sloc) 2.98 KB
Raw
Blame
History
81 lines (66 sloc) 7.39 KB
Raw
Blame
History
166 lines (129 sloc) 16.3 KB
Raw
Blame
History

2017-08-25 shard218 thread exhaustion for one hour

A user deleted a channel with many files + shares. The process maxed out connections on both sides of the shard for 65 mins.

to slowness in

2017-09-07 Shards 150, 284, and 644 overwhelmed with selects

Timeline

- 2017-0907 12:40: @n

`optimize the extra a`

2017-06-28 Channel Highlights Introduced New Load on the [redacted] Shard

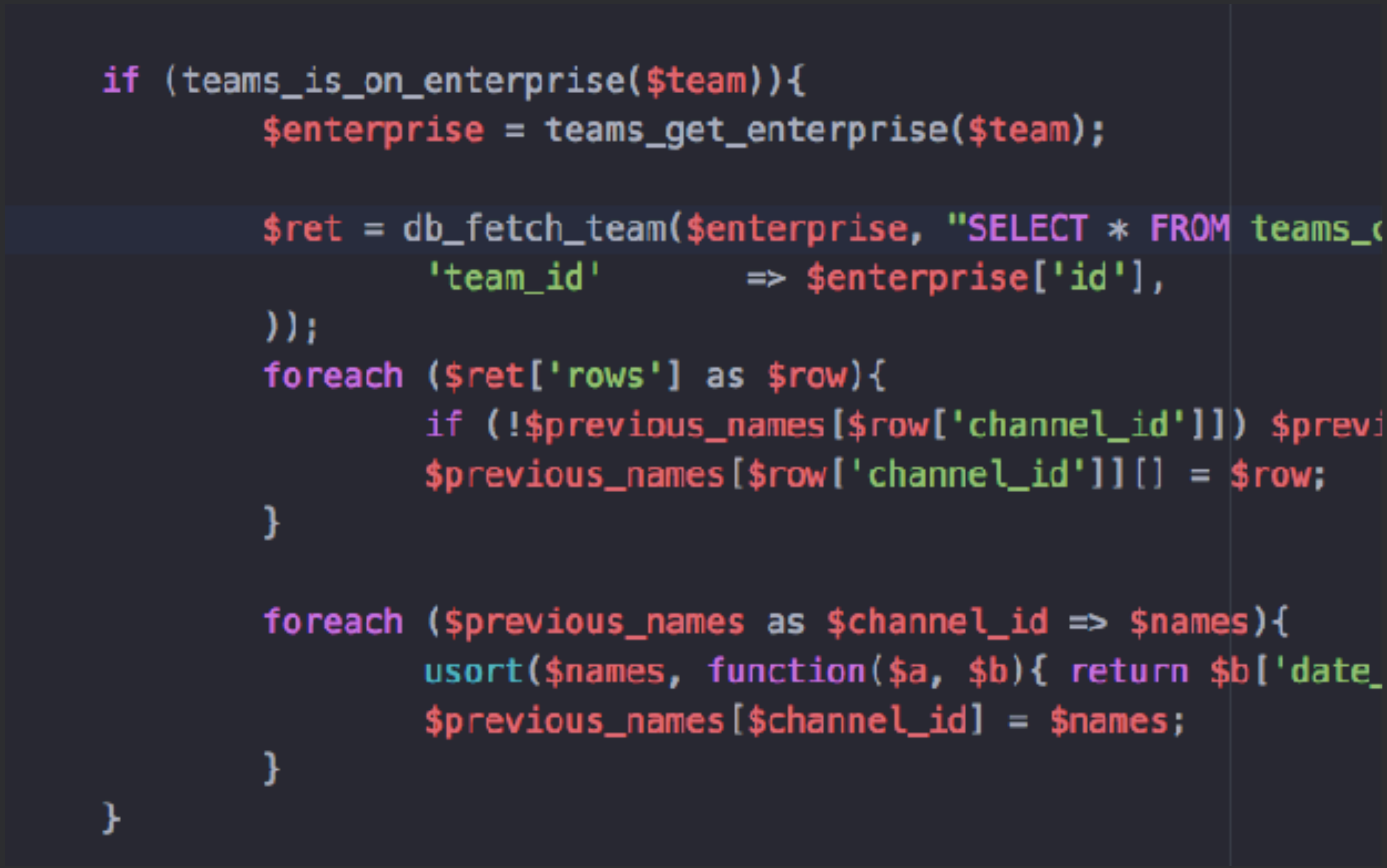
2017-07-13 [redacted] Flannel and Shard are Hot

Timeline

Application Complexity

Need the right context to route a query

Scatter query to many shards when the “owner” team is not known.

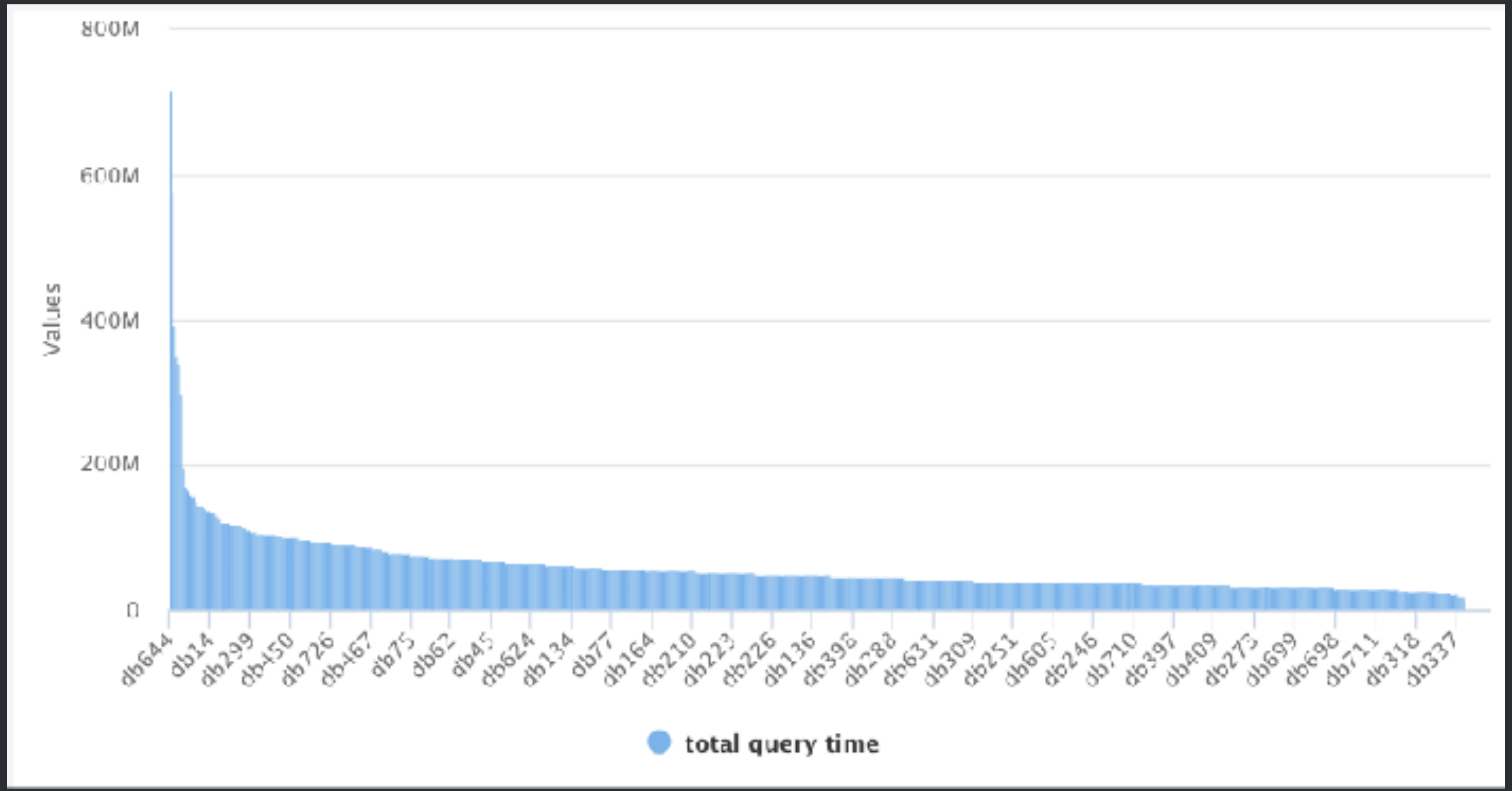


```
if (teams_is_on_enterprise($team)){
        $enterprise = teams_get_enterprise($team);

        $ret = db_fetch_team($enterprise, "SELECT * FROM teams_
                'team_id'       => $enterprise['id'],
        ));
        foreach ($ret['rows'] as $row){
                if (!$previous_names[$row['channel_id']]) $previ
                $previous_names[$row['channel_id']][] = $row;
        }

        foreach ($previous_names as $channel_id => $names){
                usort($names, function($a, $b){ return $b['date_
                $previous_names[$channel_id] = $names;
        }
}
```

Inefficient Usage

Average load (~200 qps) much lower than capacity to handle spikes

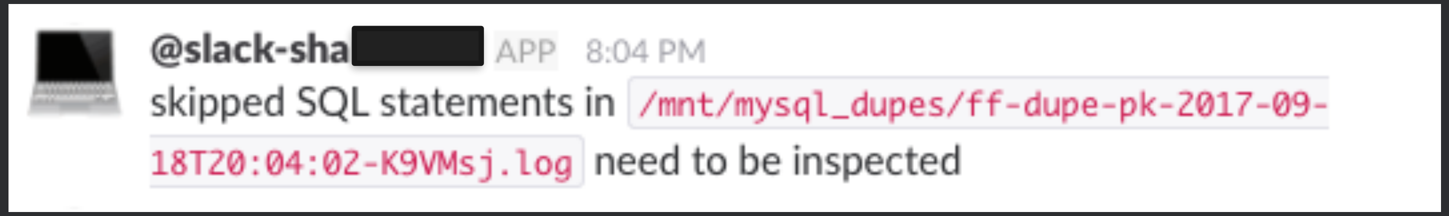
Very uneven distribution of queries across hosts



Operator Interventions

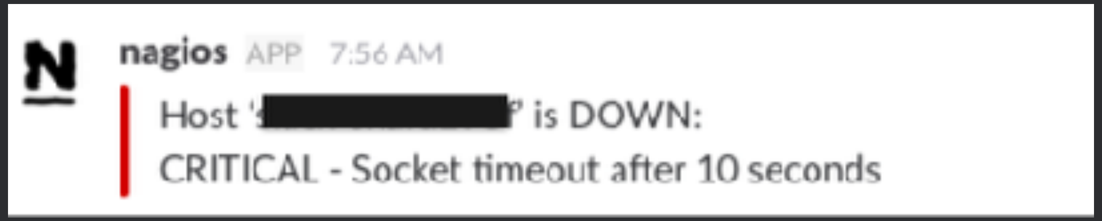
Operators need to manually repair conflicts and replace failed hosts.

Busy shards are split using manual processes and custom scripts



@slack-sha[redacted] APP 8:04 PM

skipped SQL statements in `/mnt/mysql_dupes/ff-dupe-pk-2017-09-18T20:04:02-K9VMsj.log` need to be inspected



nagios APP 7:56 AM

Host '[redacted]' is DOWN:
CRITICAL - Socket timeout after 10 seconds

So What To Do?

Next Gen Database Goals

✨ Shard by Anything! (Channel, File, User, etc)

💻 Maintain Existing Development Model

🕐 Highly Available (but a bit more consistent)

📈 Efficient System Utilization

👌 Operable In Slack's Environment

Possible Approaches

Shard by X in PHP

+ no new components
+ easiest migration

- lots of development and operations effort

NoSQL

+ flexible sharding
+ proven at scale

- major change to app
- new operations burden

NewSQL

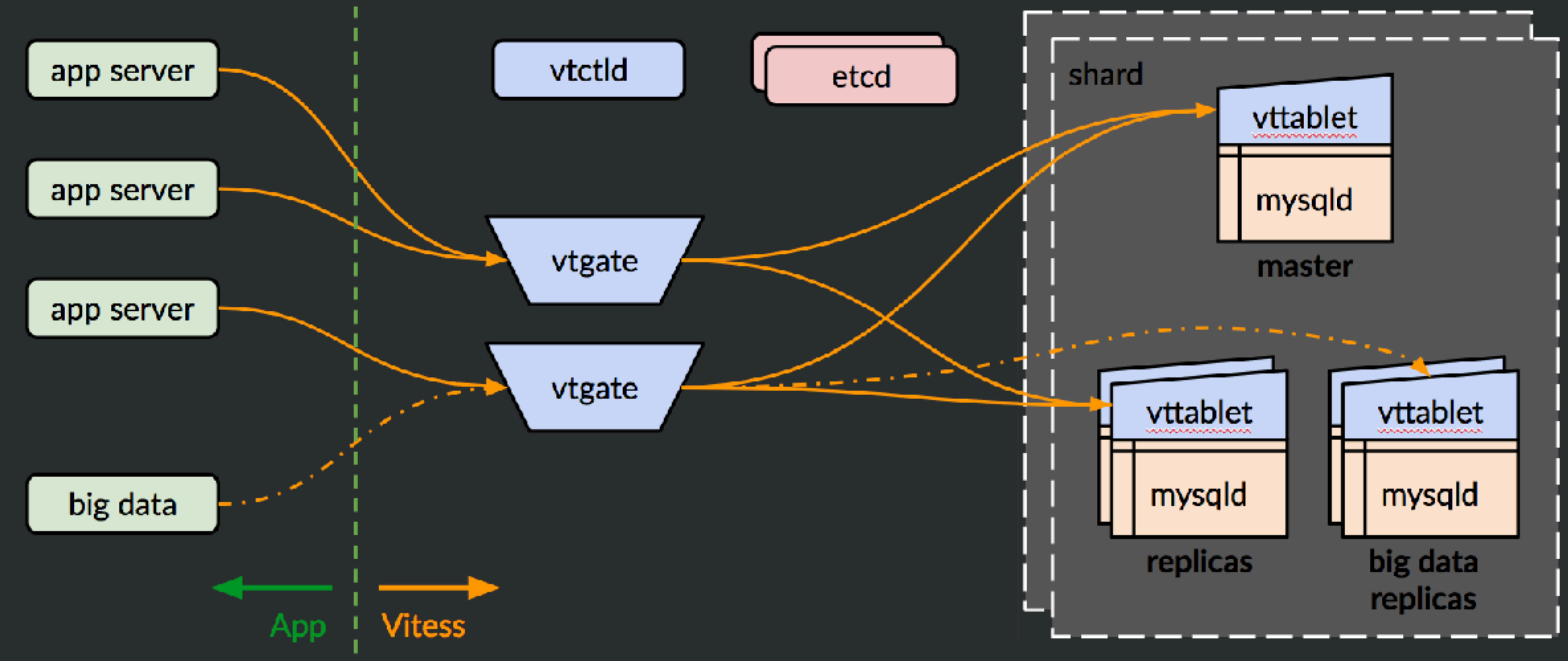
+ flexible sharding
+ scale-out storage
+ SQL compatibility!

- least well known

Why Vitess?

- Scaling and sharding flexibility without changing SQL (much)
- MySQL core maintains operator and developer know-how
- Proven at scale at YouTube and more recently others
- Active developer community and approachable code base

Vitess In One Slide



Credit: Sugu Sougoumarane <sougou@google.com>

Shard by Anything

- Applications issue queries as if there was one giant database, Vtgate routes to the right shard(s)
- "Vindex" configures most natural sharding key for each table
- Aggregations / joins pushed down to MySQL when possible
- Secondary lookup indexes (unique and non-unique)
- Still supports inefficient (but rare) patterns: Scatter / gather, cross-shard aggregations / joins

Easy Development Model

- Vitess supports the mysql server protocol end to end
- App connects to any Vtgate host to access *all* tables
- Most SQL queries are supported (with some caveats)
- Additional features: connection pooling, hot row protection, introspection, metrics

Highly Available (and more consistent)

- Vitess topology manager handles master / replica config
- Actual replication still performed by MySQL
- Changed to row-based, semi-sync replication using GTIDs
- Deployed Orchestrator to manage failover in seconds

Efficient System Usage

- Vitess components are performant and well tuned from production experience at YouTube
- Can split load vertically among different pools of shards
- Even distribution of fine grained shard keys spreads load to run hosts with higher average utilization

Operable in Slack's Environment

- MySQL is production hardened and well understood
- Leverage team know-how and tooling
- Replication still uses built-in mysql support
- New tools for topology management, shard splitting / merging
- Amenable to run in AWS without containers

Vitess Adoption: Approach and Experiences

Migration Approaches

Migrate individual tables / features one by one

Run Vitess in front of existing DBs

Migration Approaches

Migrate individual tables / features one by one ✅

- Only approach that enables resharding (for now)
- Methodical approach to reduce risk

Run Vitess in front of existing DBs 🚫

- Could make it work with custom sharding scheme in Vitess
- But we run master/master
- And doesn't help to avoid hot spots!

Migration Plan

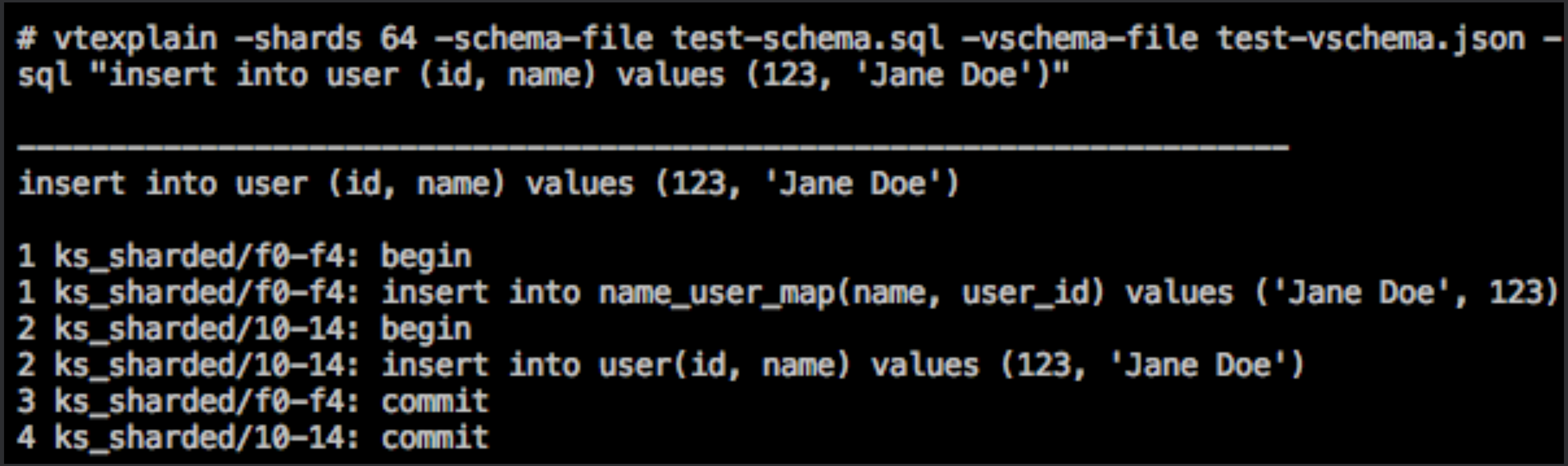
- For each table to migrate:

1. Analyze queries for common patterns
2. Pick a keyspace (i.e. set of shards) and sharding key
3. Double-write from the app and backfill the data
4. Switch the app to use vitess

- But we also need to find and migrate all joined tables
  ... and queries that aren't supported or efficient any more
  ... and whether the old data model even makes sense!!

Offline analysis (vtexplain)

- Analysis tool to show what actually runs on each shard
- Query support is not yet (likely never be) 100% MySQL
- Choice of sharding key is crucial for efficiency



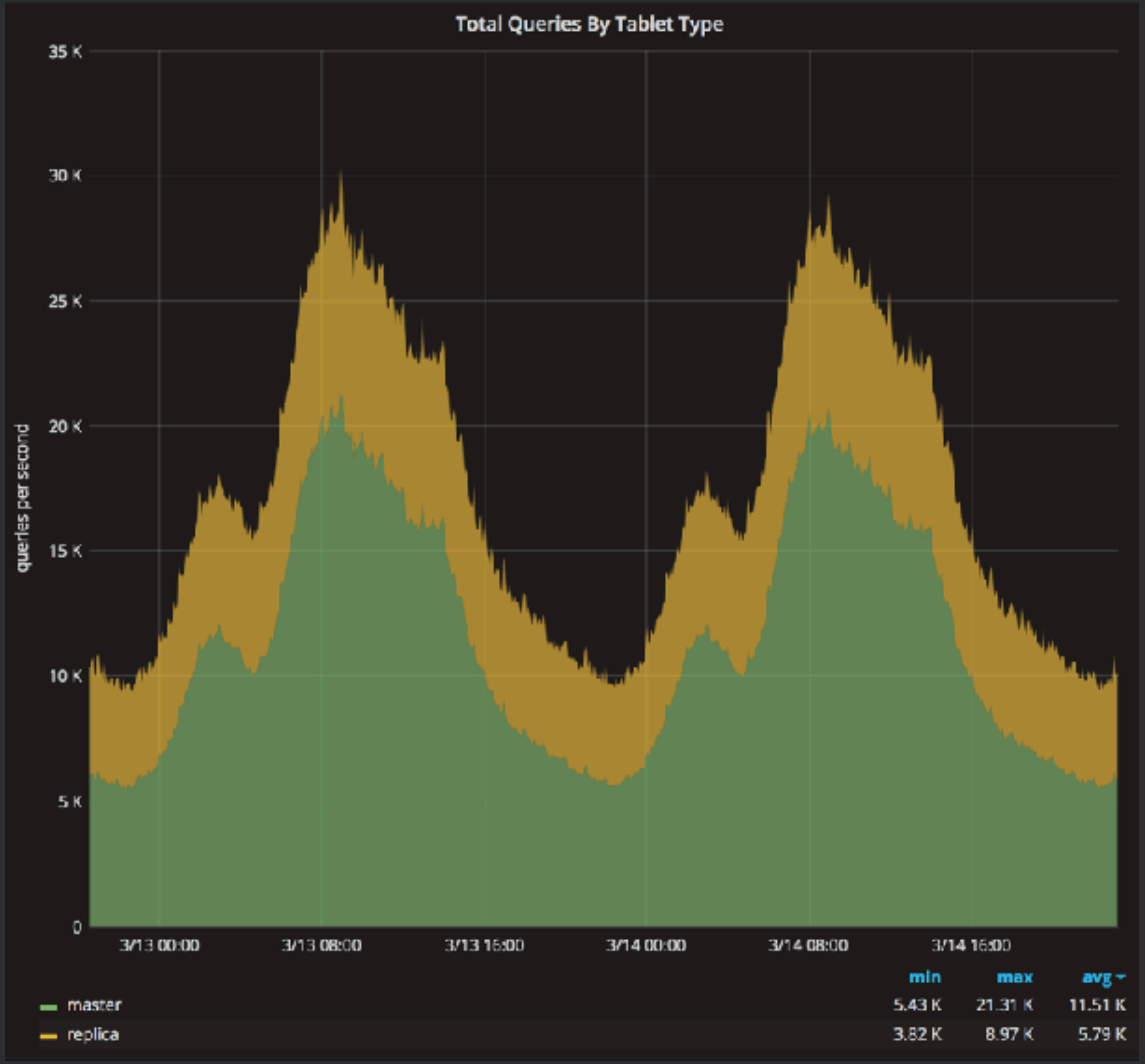
```
# vtexplain -shards 64 -schema-file test-schema.sql -vschema-file test-vschema.json -
sql "insert into user (id, name) values (123, 'Jane Doe')"

----------------------------------------------------------------------
insert into user (id, name) values (123, 'Jane Doe')

1 ks_sharded/f0-f4: begin
1 ks_sharded/f0-f4: insert into name_user_map(name, user_id) values ('Jane Doe', 123)
2 ks_sharded/10-14: begin
2 ks_sharded/10-14: insert into user(id, name) values (123, 'Jane Doe')
3 ks_sharded/f0-f4: commit
4 ks_sharded/10-14: commit
```

Migration Stages

- 🚇 PASSTHROUGH: Convert call sites
- 🏗️ BACKFILL: Double-write & bulk copy, read legacy
- 🌃 DARK: Double-read/write, app sees legacy results
- 🌅 LIGHT: Double-read/write, app sees Vitess results
- 🌇 SUNSET: Read/write only from Vitess

Current Status

- 🎉 Running in production for 10 months
- Serving ~10% of all queries, part of the critical path for Slack
- All new features use Vitess
- Migrating other core tables this year



Current Status: Details

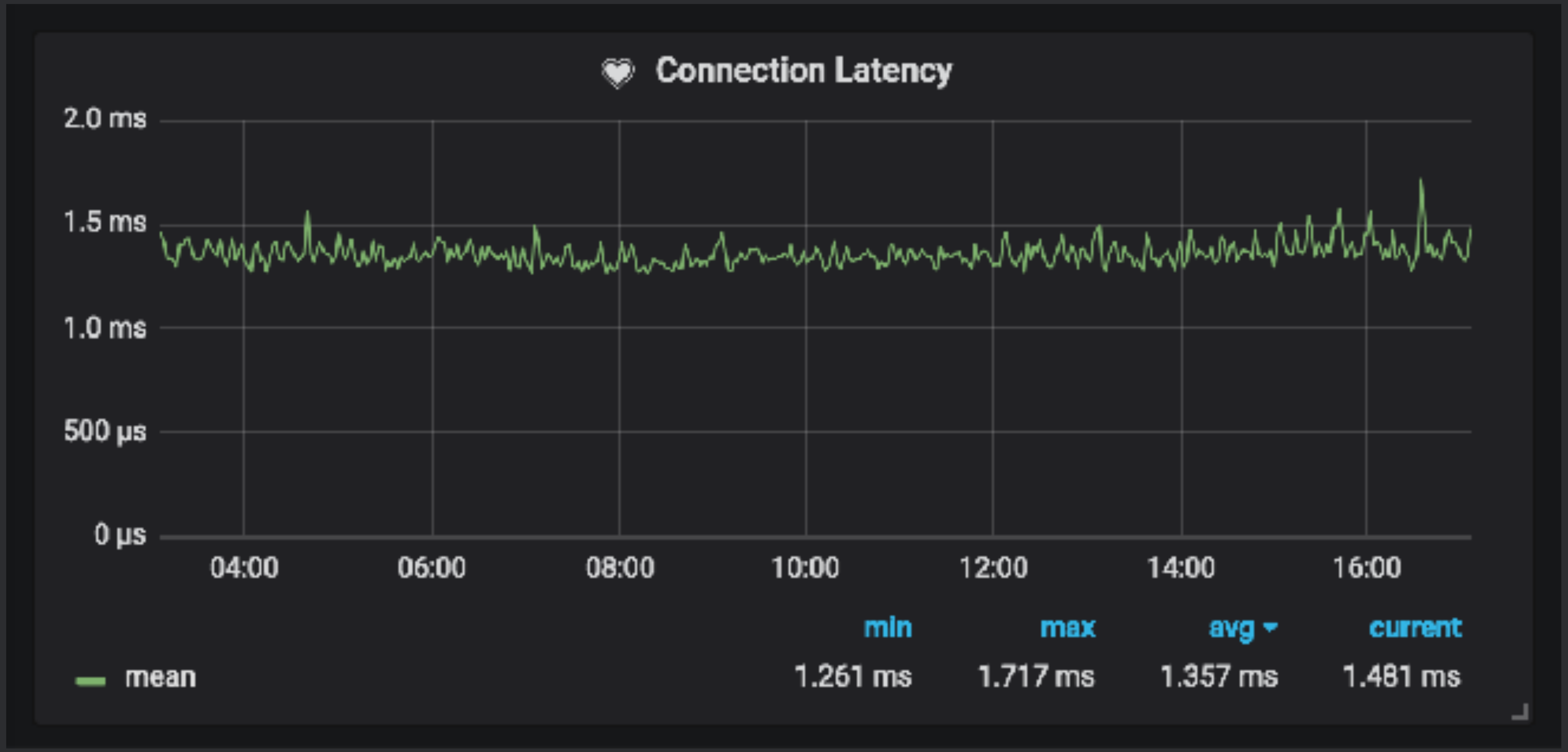
- ~30,000 QPS at peak times, occasional spikes above 50,000
- 8 keyspaces, 3 replicas per shard, 316 tablets, 32 vtgates
- Query mix is ~80% read, 20% write
- Currently ~75% queries go to masters

Performance

Millisecond latencies for connect/read/write

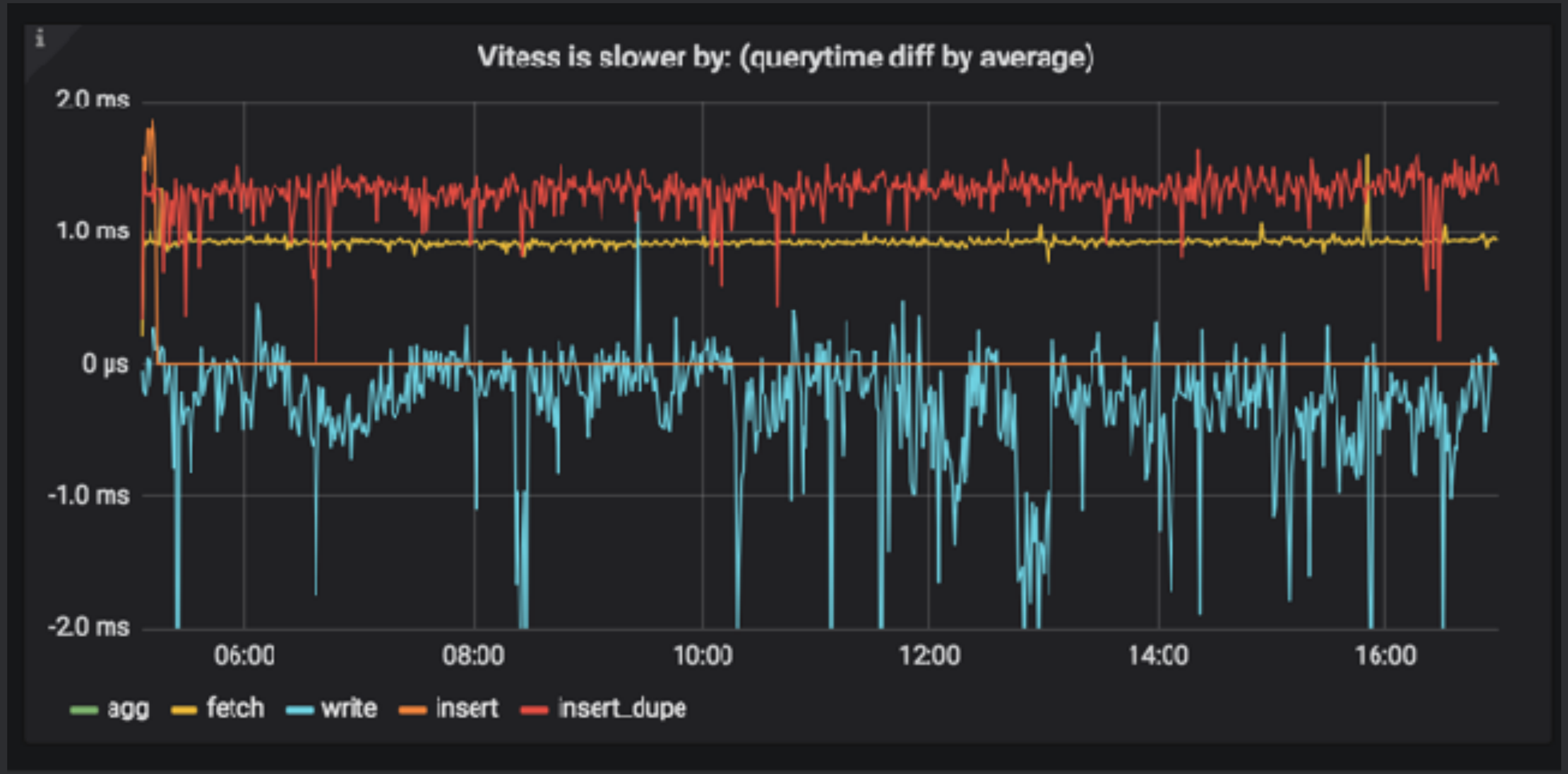
Slower due to extra network hops, semi-sync waits, and Vitess overhead

So far as expected — slightly slower but steadier



Connection Latency

	min	max	avg ▾	current
mean	1.261 ms	1.717 ms	1.357 ms	1.481 ms



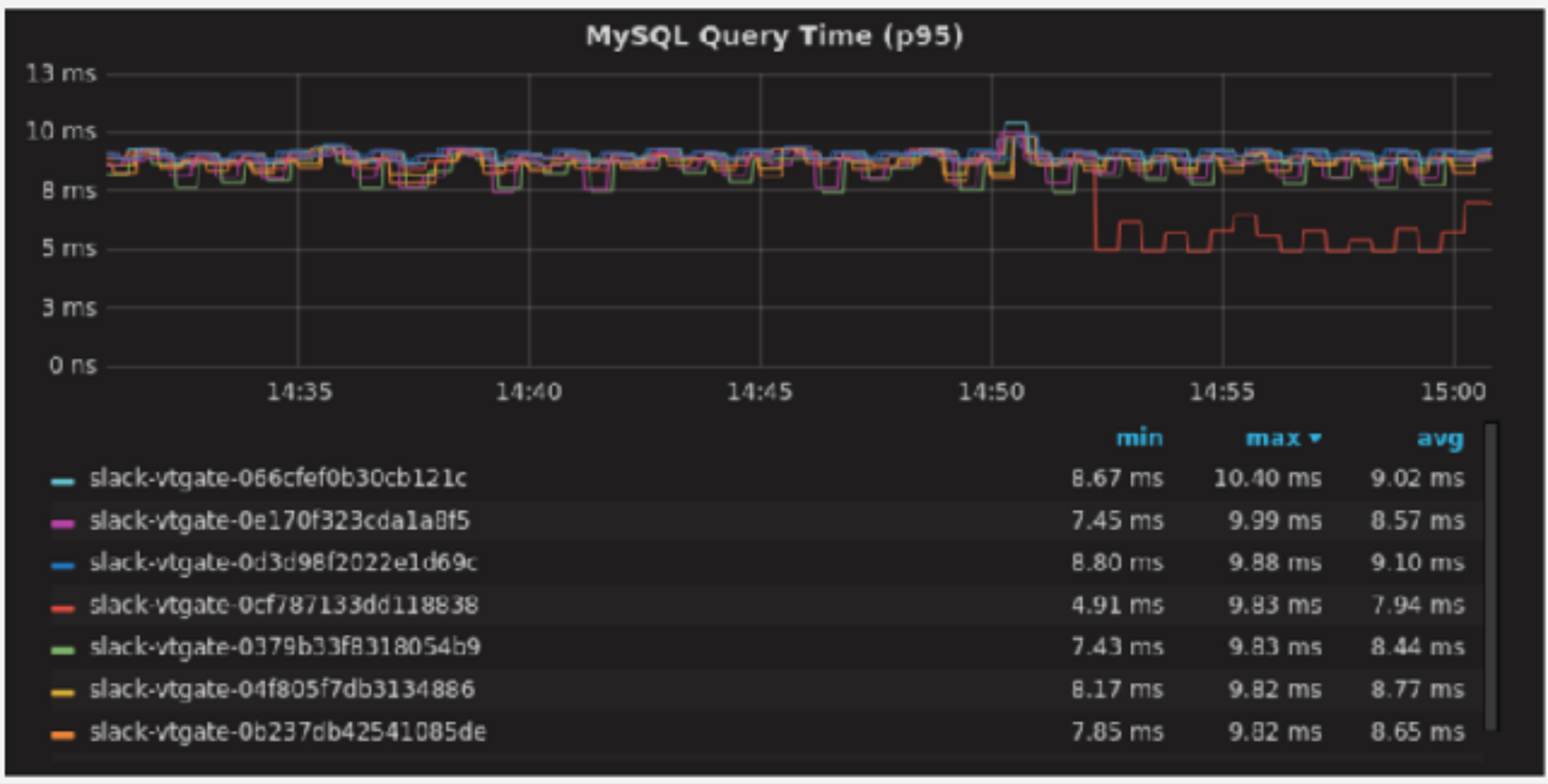
Vitess is slower by: (querytime diff by average)

Performance Improvements

Vitess modifications:

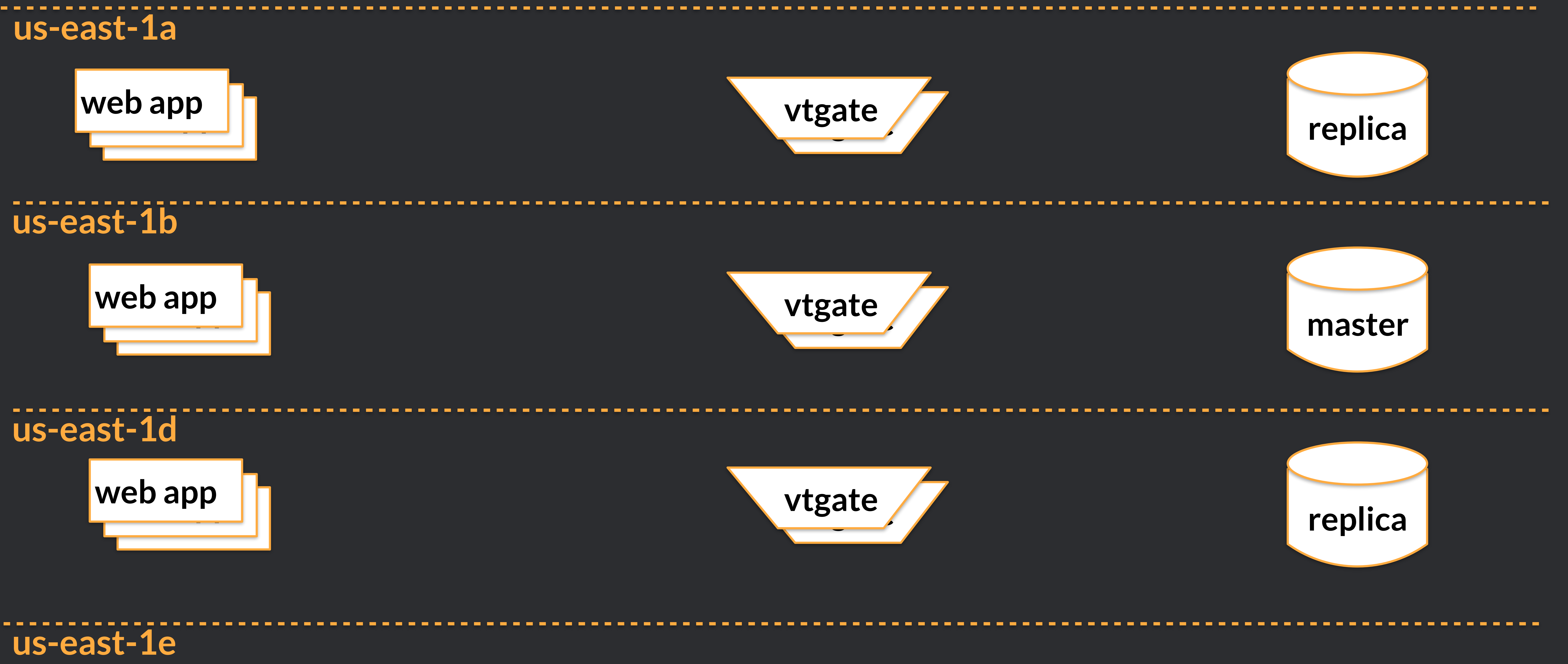
- Avoid round trips for autocommit transactions
- Scatter DML queries
- Query pool timeouts

Dramatically improved both average and tail latencies

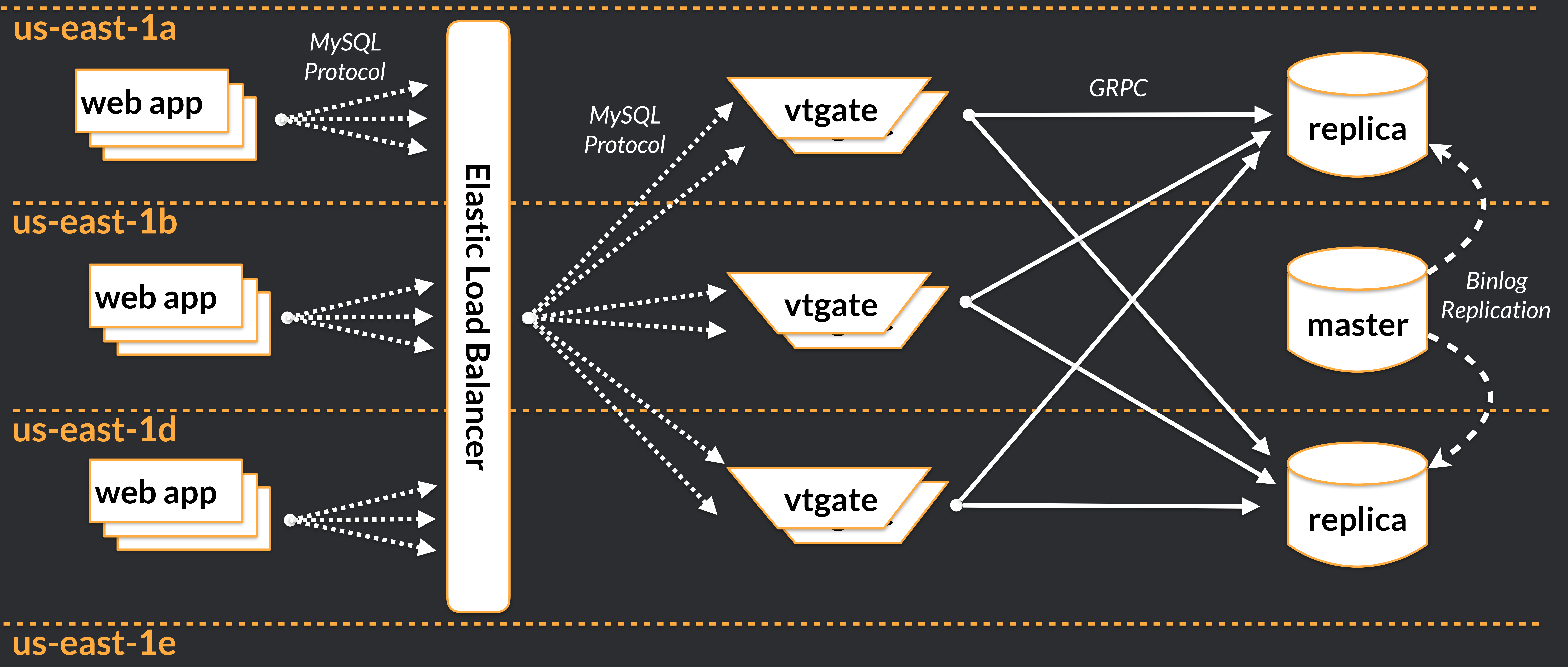


		min	max ▾	avg
🟦	slack-vtgate-066cfef0b30cb121c	8.67 ms	10.40 ms	9.02 ms
🟪	slack-vtgate-0e170f323cda1a8f5	7.45 ms	9.99 ms	8.57 ms
🟦	slack-vtgate-0d3d98f2022e1d69c	8.80 ms	9.88 ms	9.10 ms
🟥	slack-vtgate-0cf787133dd118838	4.91 ms	9.83 ms	7.94 ms
🟩	slack-vtgate-0379b33f8318054b9	7.43 ms	9.83 ms	8.44 ms
🟨	slack-vtgate-04f805f7db3134886	8.17 ms	9.82 ms	8.77 ms
🟧	slack-vtgate-0b237db42541085de	7.85 ms	9.82 ms	8.65 ms

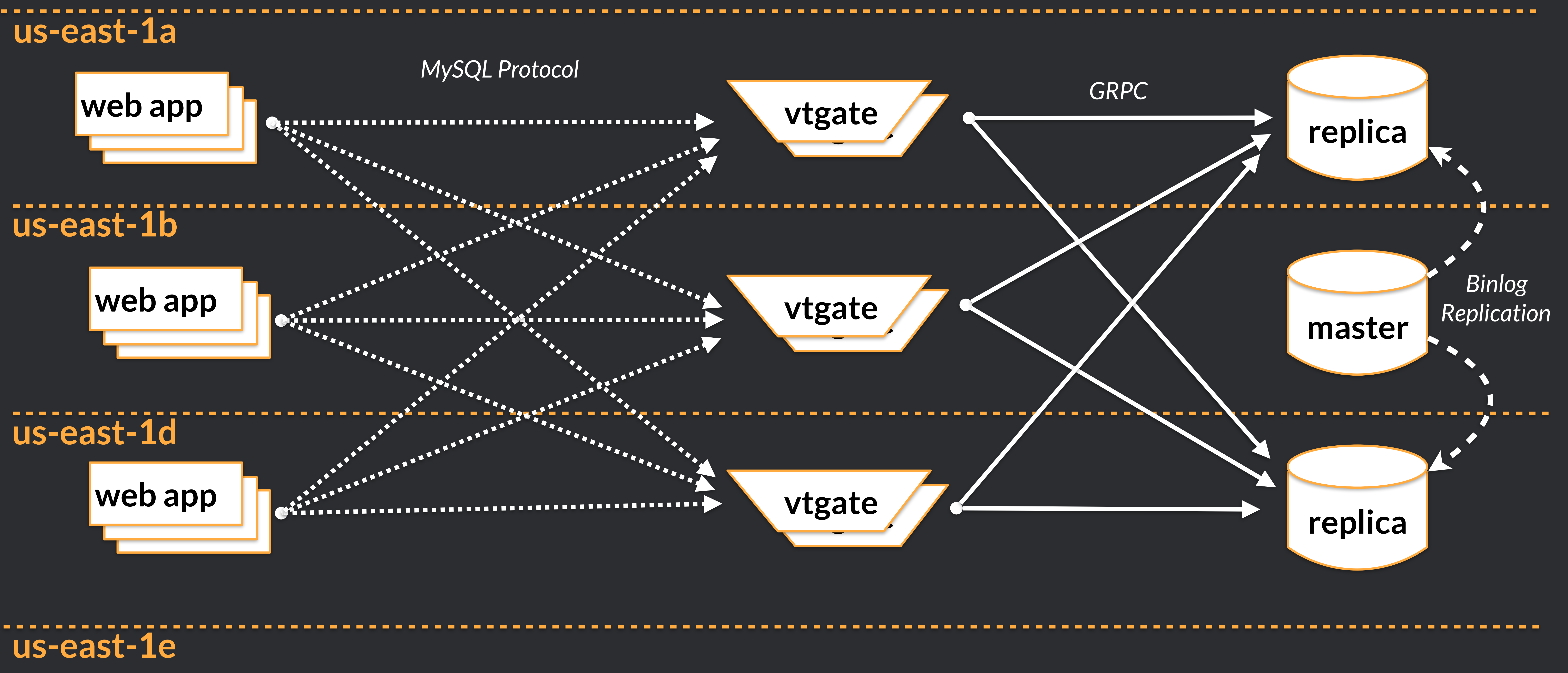
Vitess Deployment: Multi AZ



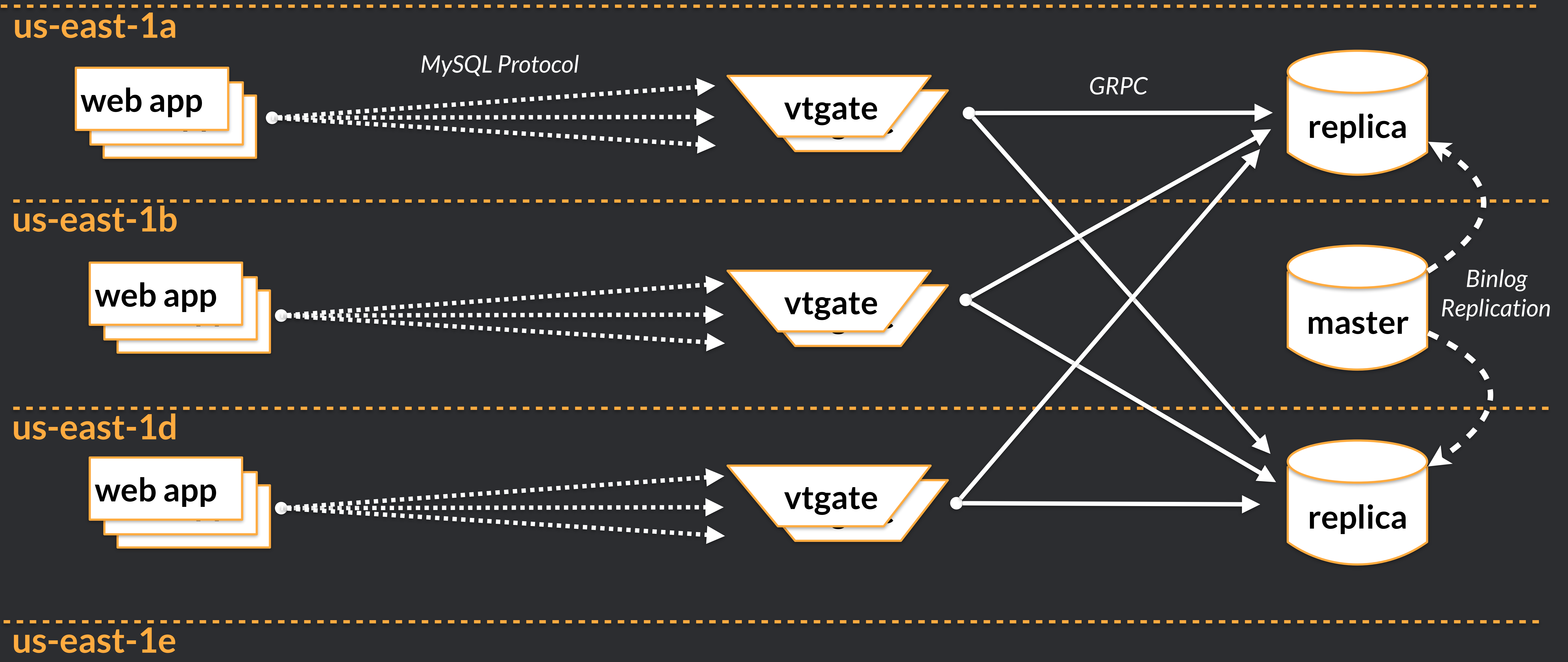
Initial Deployment



Client Side Load Balancing



AZ Aware Routing



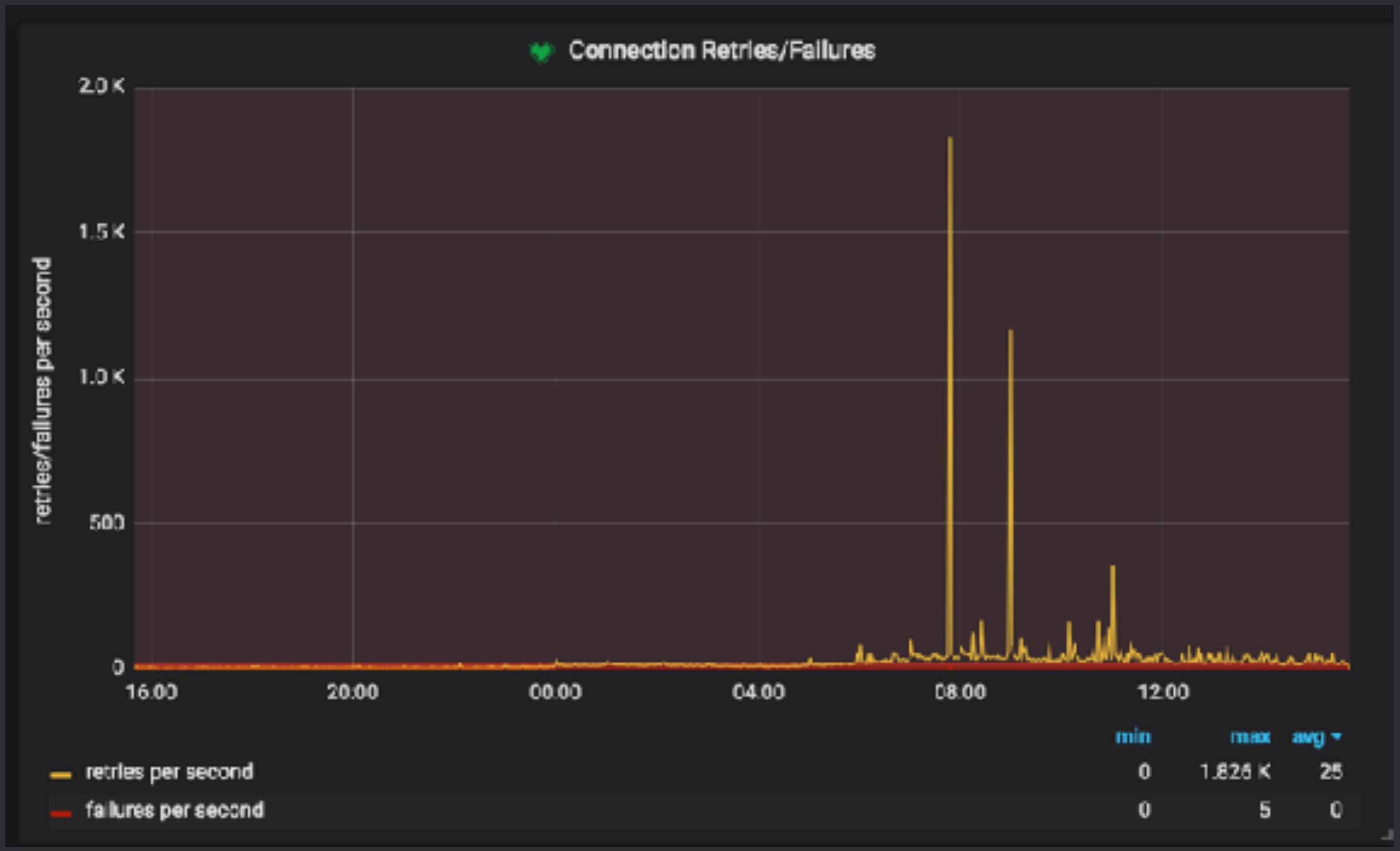
Improved... but still not great

Short-lived connections require rapid open / close

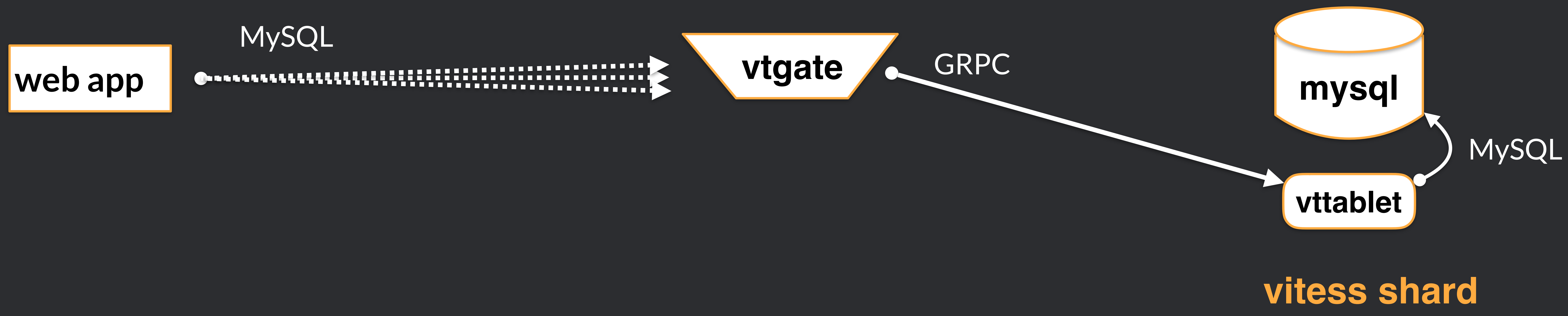
To mitigate packet loss, app quickly fails over to try another vtgate / shard

Under load this causes delays, brownouts

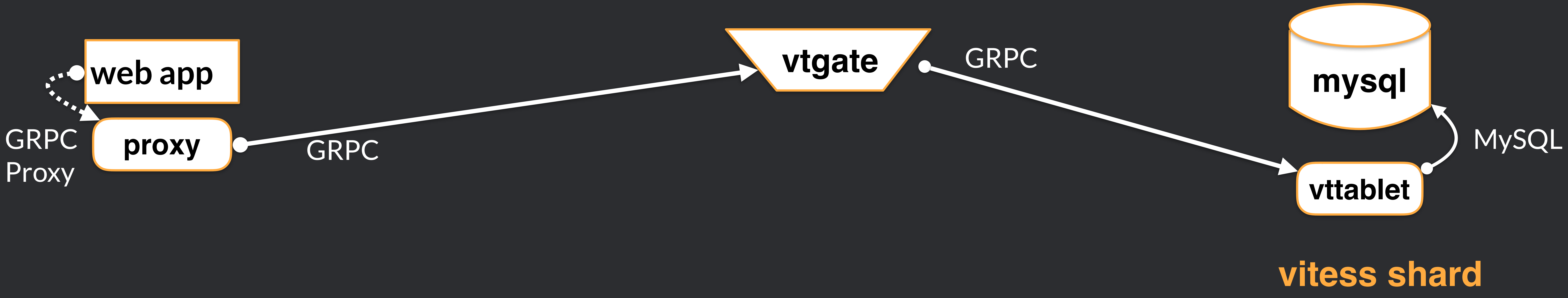
Long term goal: sticky connections everywhere



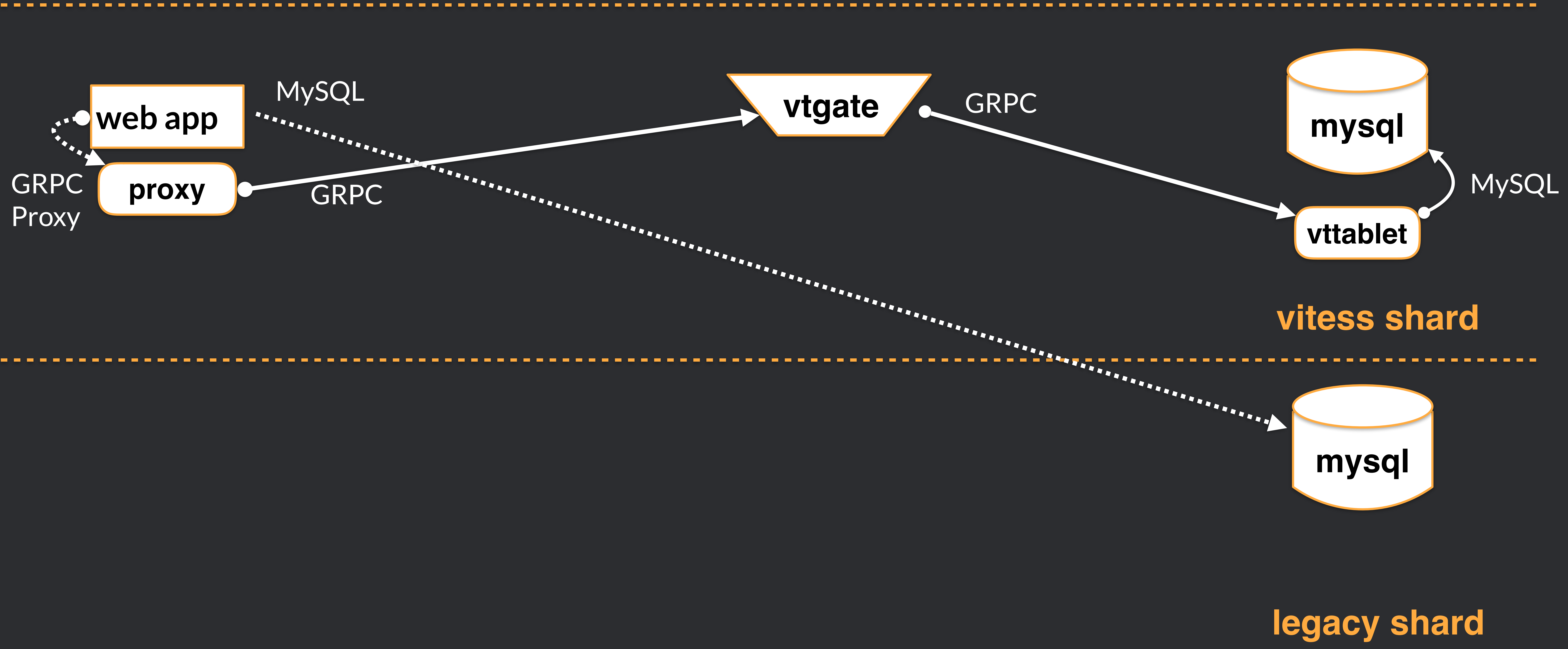
MySQL Connections



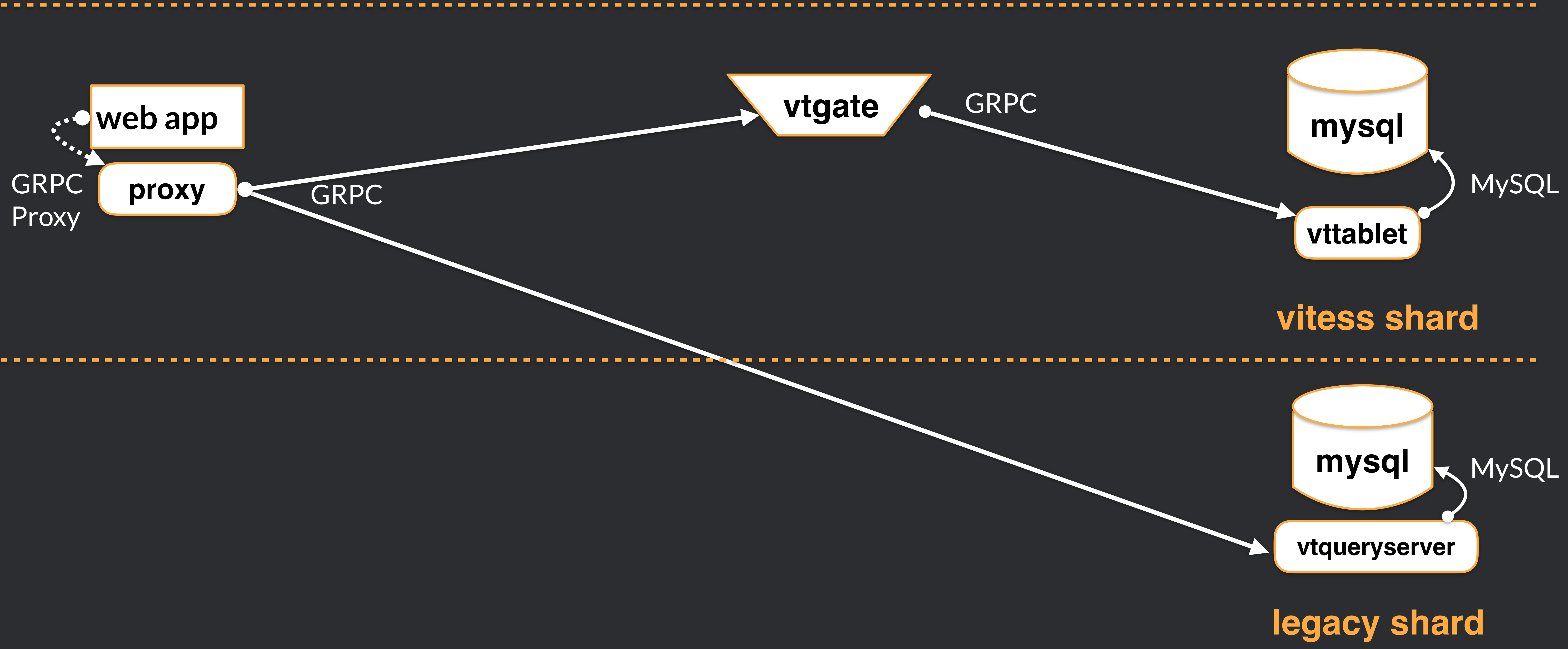
GRPC End to End



"Legacy" Databases



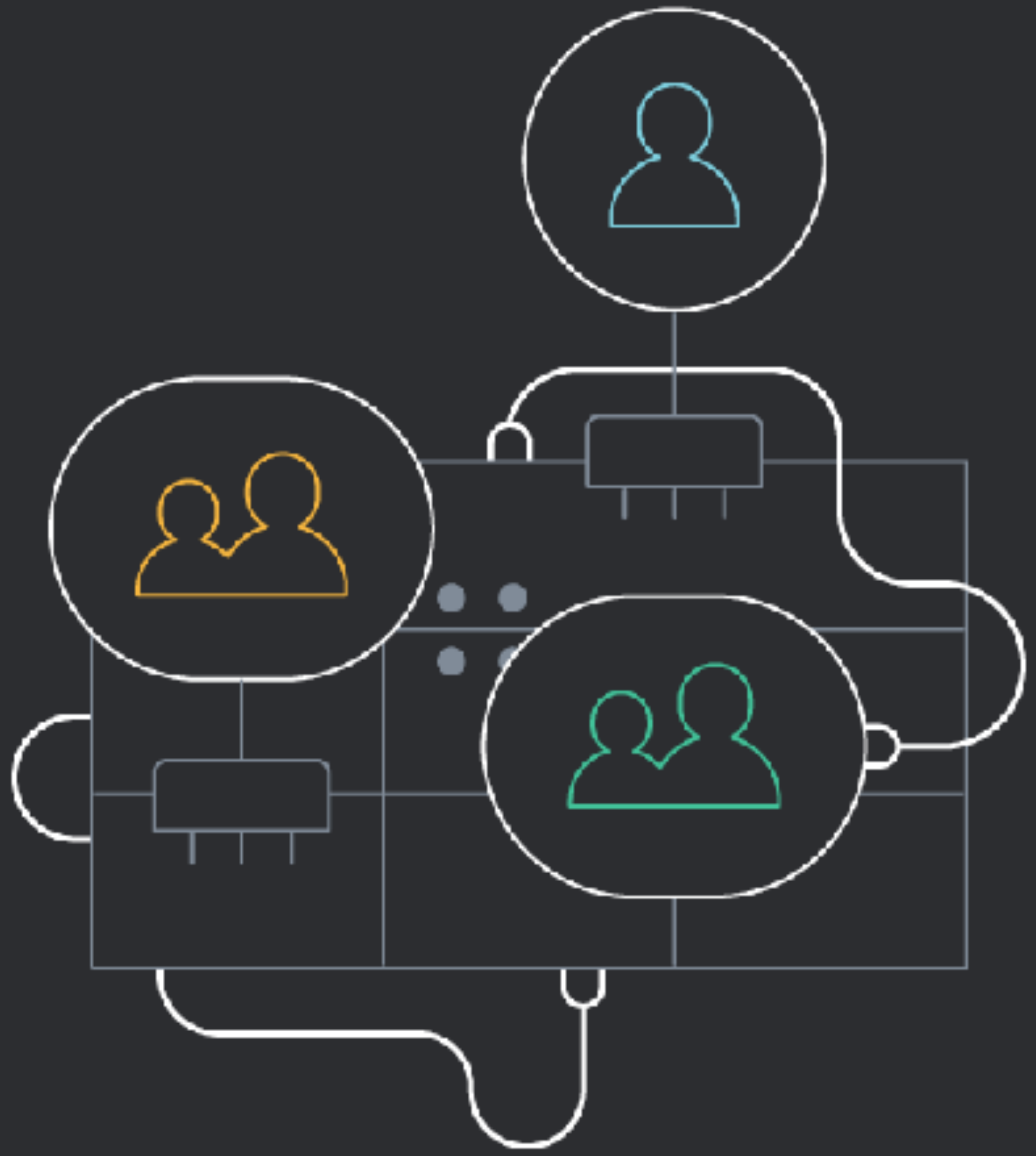
"Legacy" Databases (Future)



VTQueryserver Experiment

- Combine the vtgate query API (grpc + mysql) with the vttablet execution engine
- Helps protect mysql from query storms using connection pooling, hot row protection, query limits, etc
- Enables long lived GRPC connections from the web app
- Challenge to get the connection pool settings correct and to implement end-to-end prioritization

High Level Takeaways



Change All The Things

Because of Vitess, we had to:

switch to master / replica...

use semi-sync with gtid...

and orchestrator for failover...

But at the same time, we:

switched to row based replication...

on mysql 5.7 on new i3 EC2 hosts...

and an updated Ubuntu release...

using hhvm's async mysql driver...

and start reads from replicas...

Cha

Becau

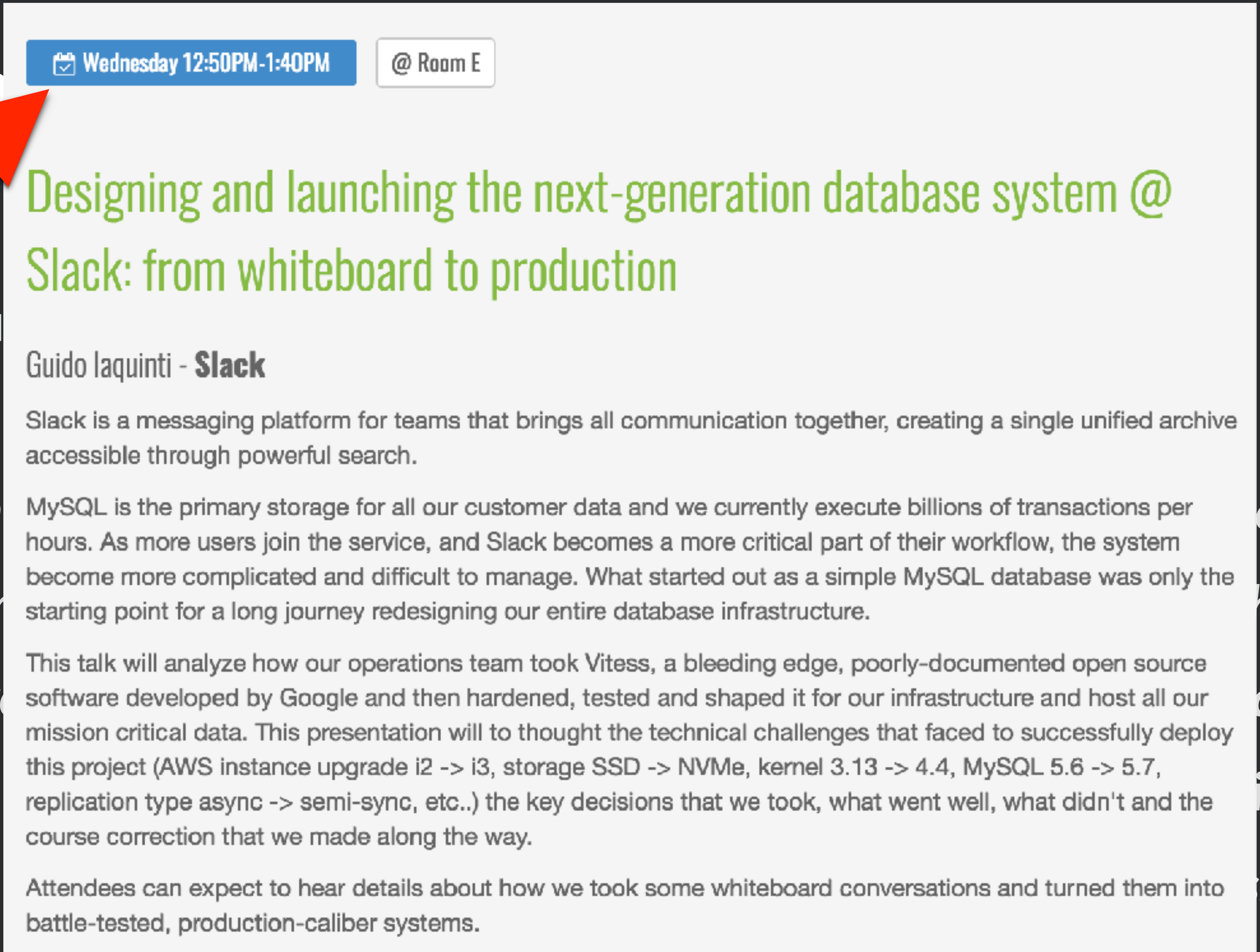
switch *ation...*

use ser *hosts...*

and or *ase...*

iver...

...



Designing and launching the next-generation database system @ Slack: from whiteboard to production

Guido Iaquinti - **Slack**

Slack is a messaging platform for teams that brings all communication together, creating a single unified archive accessible through powerful search.

MySQL is the primary storage for all our customer data and we currently execute billions of transactions per hours. As more users join the service, and Slack becomes a more critical part of their workflow, the system become more complicated and difficult to manage. What started out as a simple MySQL database was only the starting point for a long journey redesigning our entire database infrastructure.

This talk will analyze how our operations team took Vitess, a bleeding edge, poorly-documented open source software developed by Google and then hardened, tested and shaped it for our infrastructure and host all our mission critical data. This presentation will to thought the technical challenges that faced to successfully deploy this project (AWS instance upgrade i2 -> i3, storage SSD -> NVMe, kernel 3.13 -> 4.4, MySQL 5.6 -> 5.7, replication type async -> semi-sync, etc..) the key decisions that we took, what went well, what didn't and the course correction that we made along the way.

Attendees can expect to hear details about how we took some whiteboard conversations and turned them into battle-tested, production-caliber systems.

Networking Matters

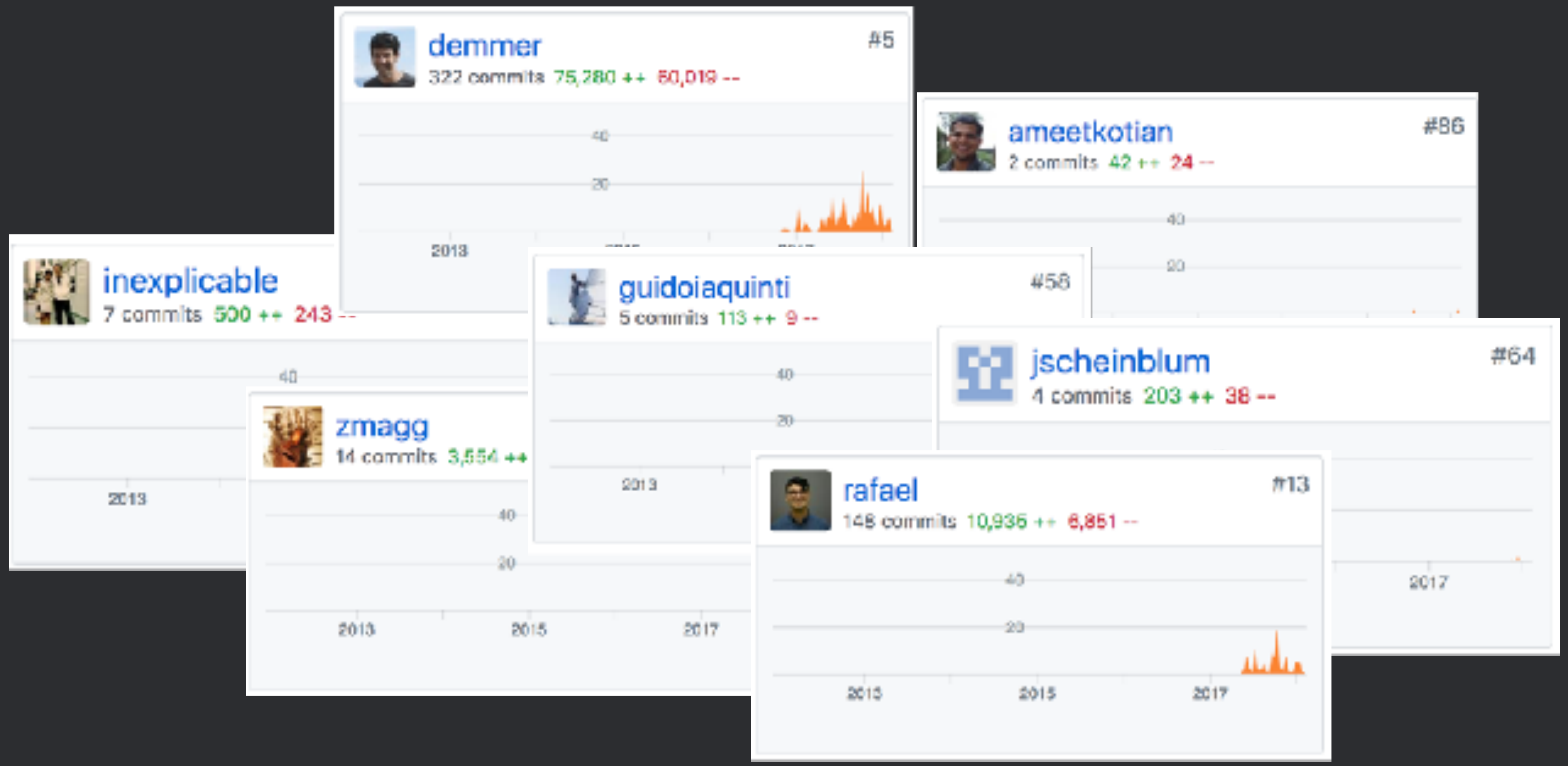
- Vitess is intrinsically more network dependent than our existing database architecture
- Performance depends (a lot) on network quality
- Improved consistency (single master / semi-sync) comes at the expense of availability and performance
- Able to work around some issues by kernel tuning, host placement, application routing to vtgate

Vitess: “Build” *and* “Buy”

The core of Vitess is stable, performant, and robust

But Slack’s use case differs from YouTube's (and others)

Adoption required significant changes, all contributed back upstream



demmer #5
322 commits 75,280 ++ 60,019 --

ameetkotian #86
2 commits 42 ++ 24 --

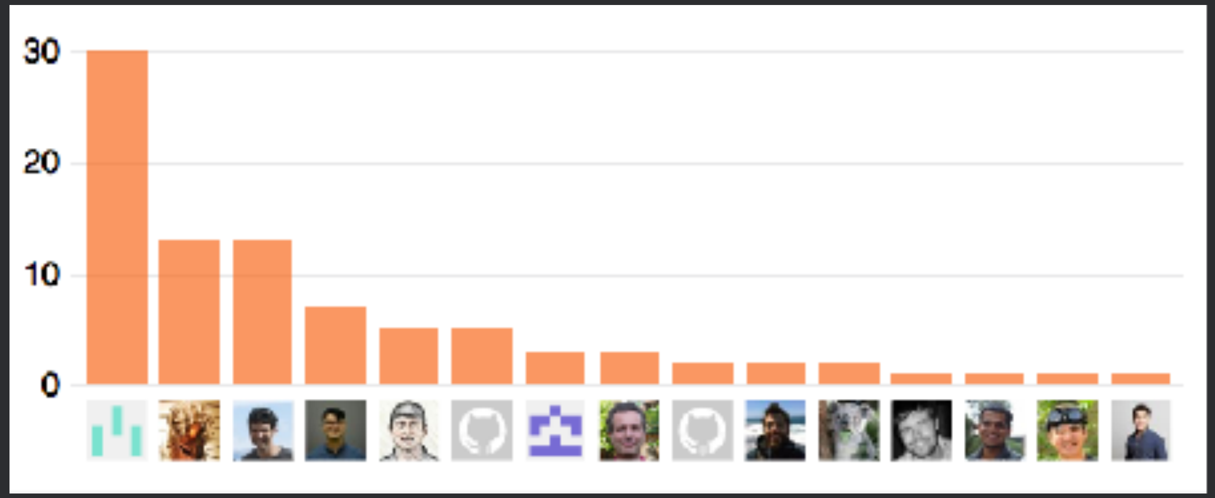
inexplicable
7 commits 500 ++ 243 --

guidoiaquinti #58
5 commits 113 ++ 9 --

jscheinblum #64
4 commits 203 ++ 38 --

zmagg
14 commits 3,554 ++

rafael #13
148 commits 10,936 ++ 6,851 --



"Vitess is magical but not magic"

- ⁉️ Besides MySQL, there are a still lot of new moving parts
- 😳 No ability (yet) to change sharding key
- 🚫 Still some unsupported queries (though not as many)
- ⚠️ Scalability / efficiency requires stale reads from replica
- 😞 Can't (yet) use familiar tools like phpmyadmin
- 🔎 Documentation!! -- many, many options to understand

Vitess At Slack: Thriving

- In production for ~10 months after ~7 months of effort
- Leadership buy in as the future for Slack databases
- Stable and performs well (so far)

We have a long but exciting road ahead...

And we are hiring!

Thank you!