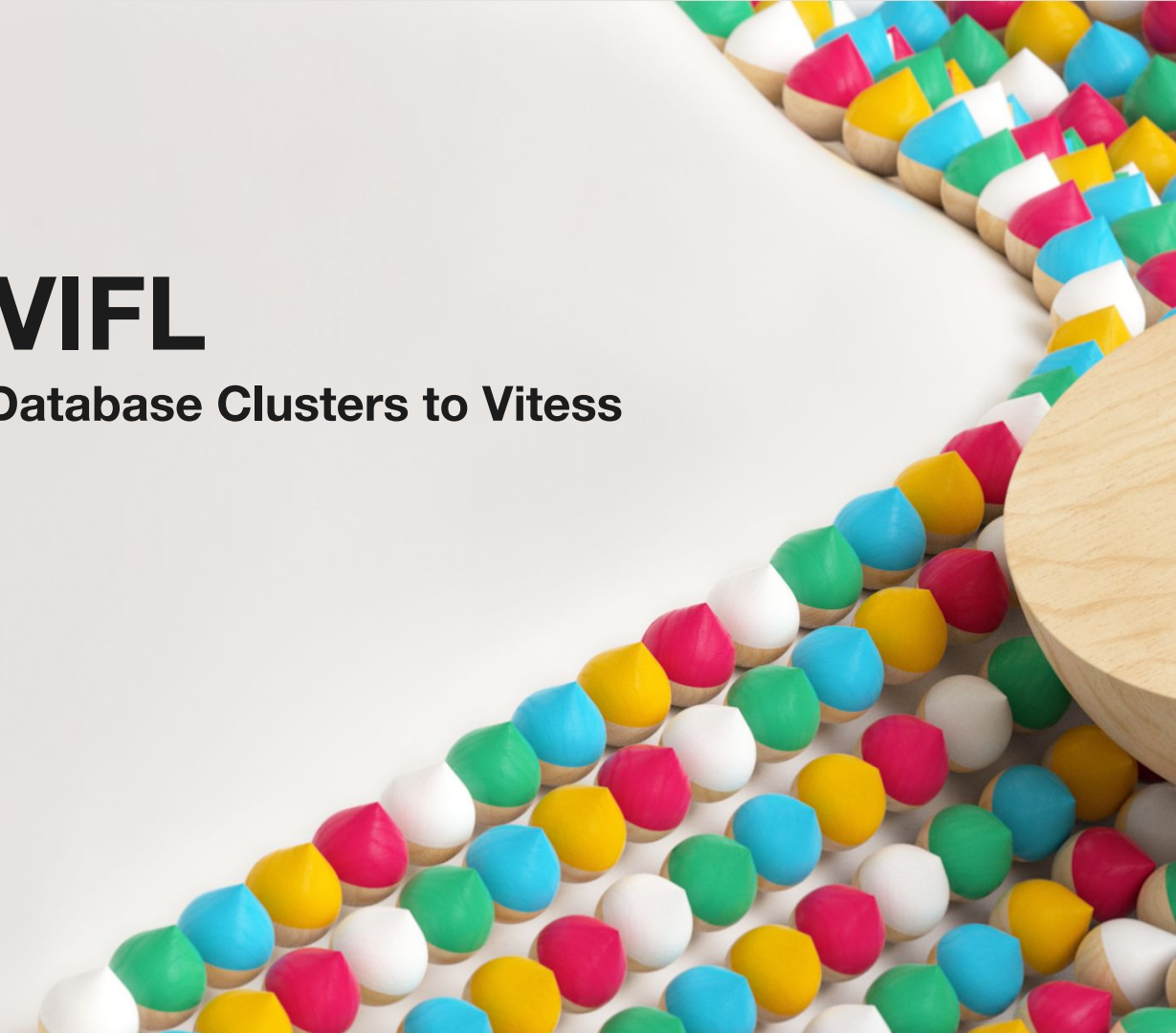
Codename VIFL

How to Migrate MySQL Database Clusters to Vitess

Rafael Chacón
Guido Iaquinti

slack



SPEAKERS

Rafael Chacón

he/him/his

twitter.com/rafaelchacon

Staff Software Engineer - Slack

Guido Iaquinti

he/him/his

twitter.com/guidoiaquinti

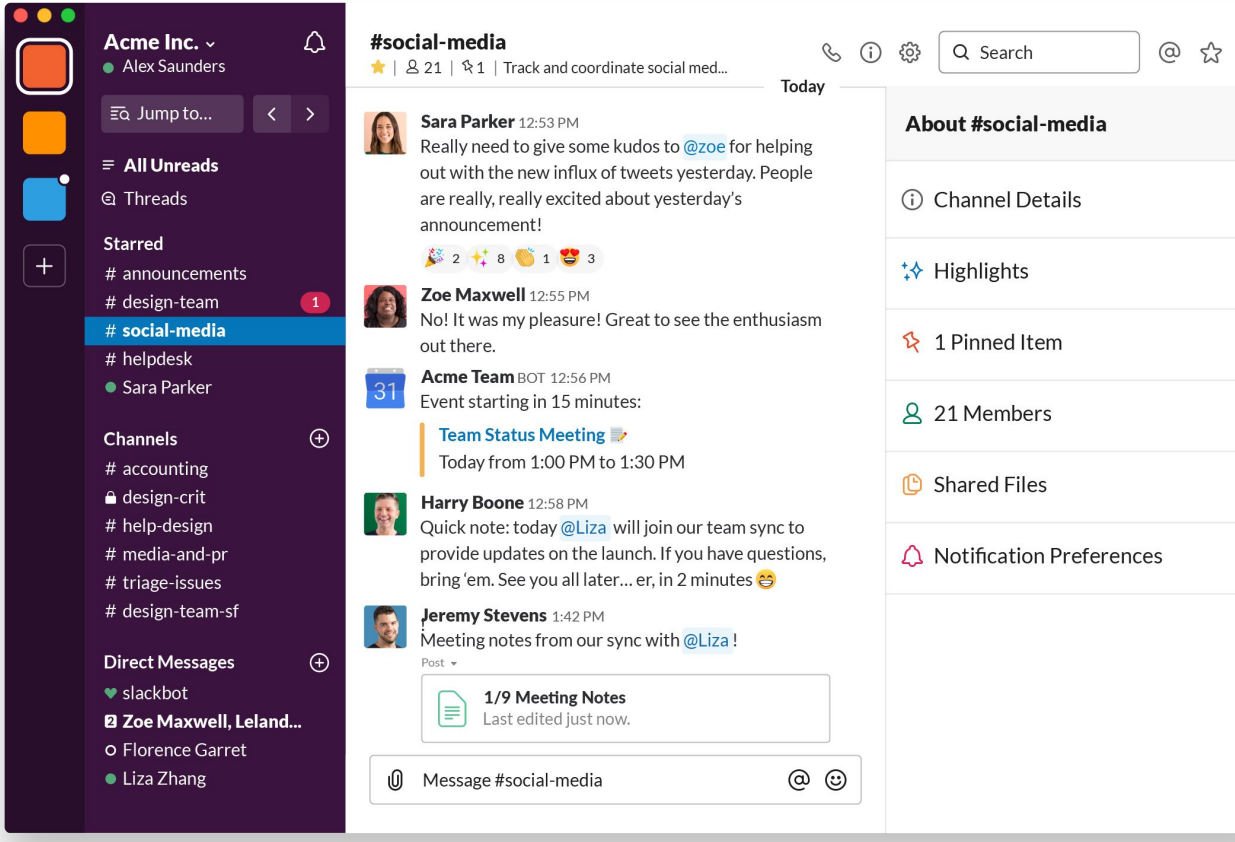
Senior Staff Software Engineer - Slack

Agenda

1. Databases at Slack.
2. The Legacy Shards.
3. Vitess in front of legacy shards (VIFL).
    a. The challenge and strategy.
    b. Validation.
    c. Automation.
4. Final remarks.
5. Q&A.

MISSION STATEMENT

Slack's mission is to make people's working lives **simpler**, more **pleasant**, and more **productive**.



Databases at Slack

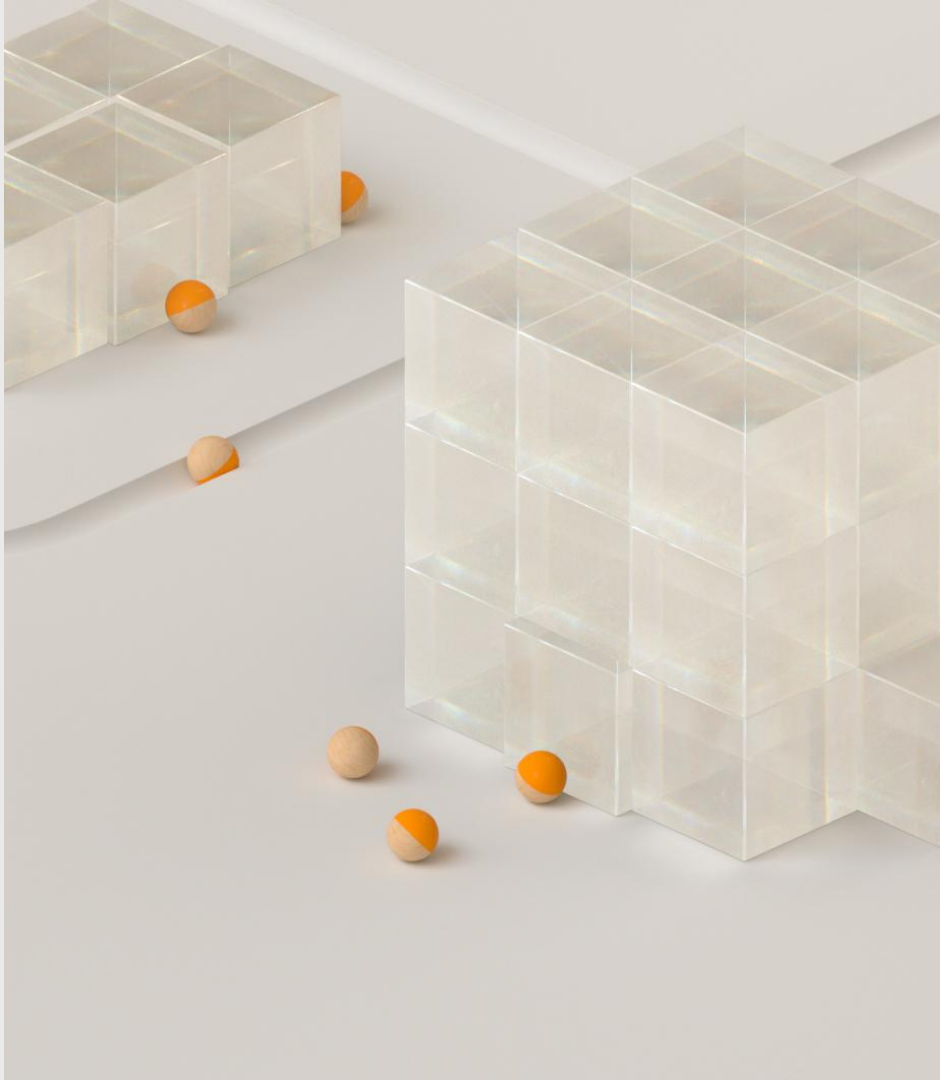
Stats

Daily Active Users: **12 million**

Queries per day: **65+ billion**

Storage provisioned: **9+ PB**

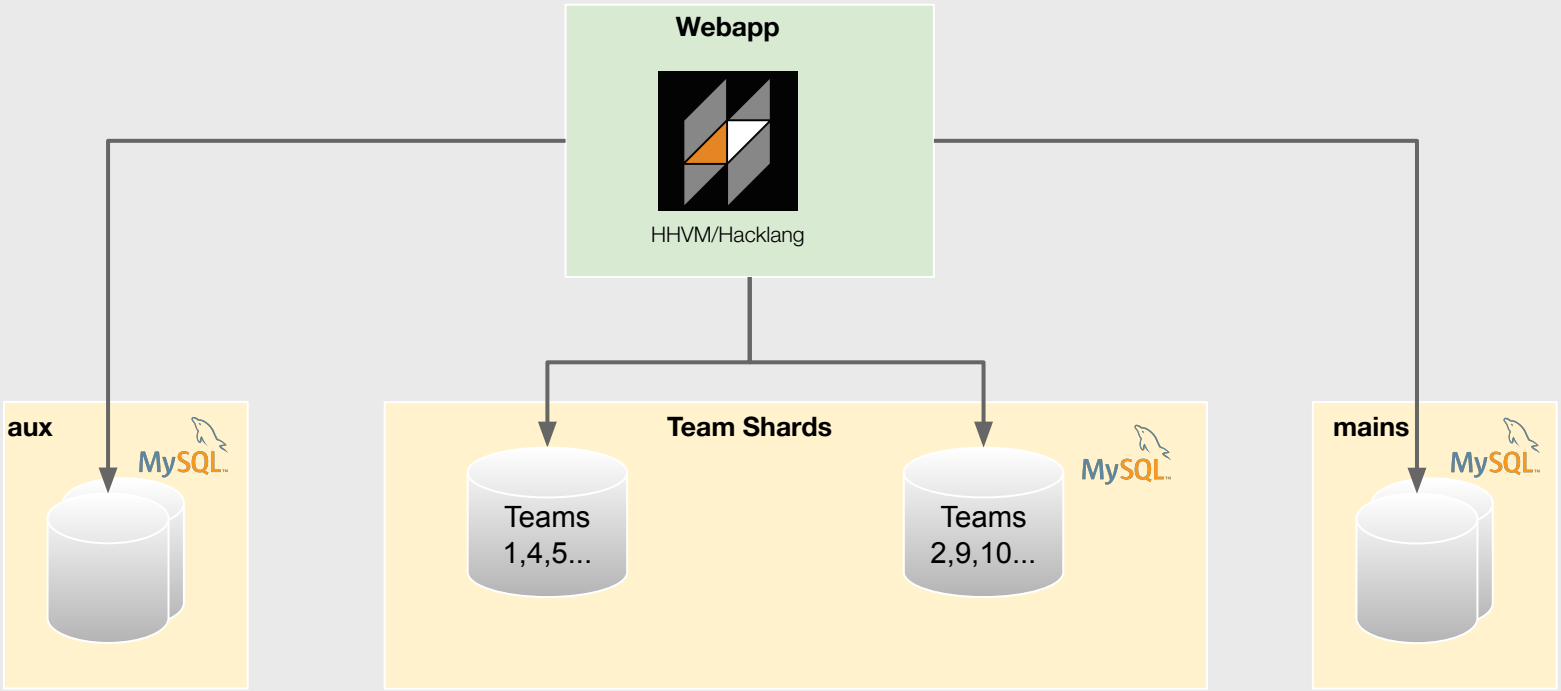
Thousands of database servers.



The Legacy (Shards)

slack

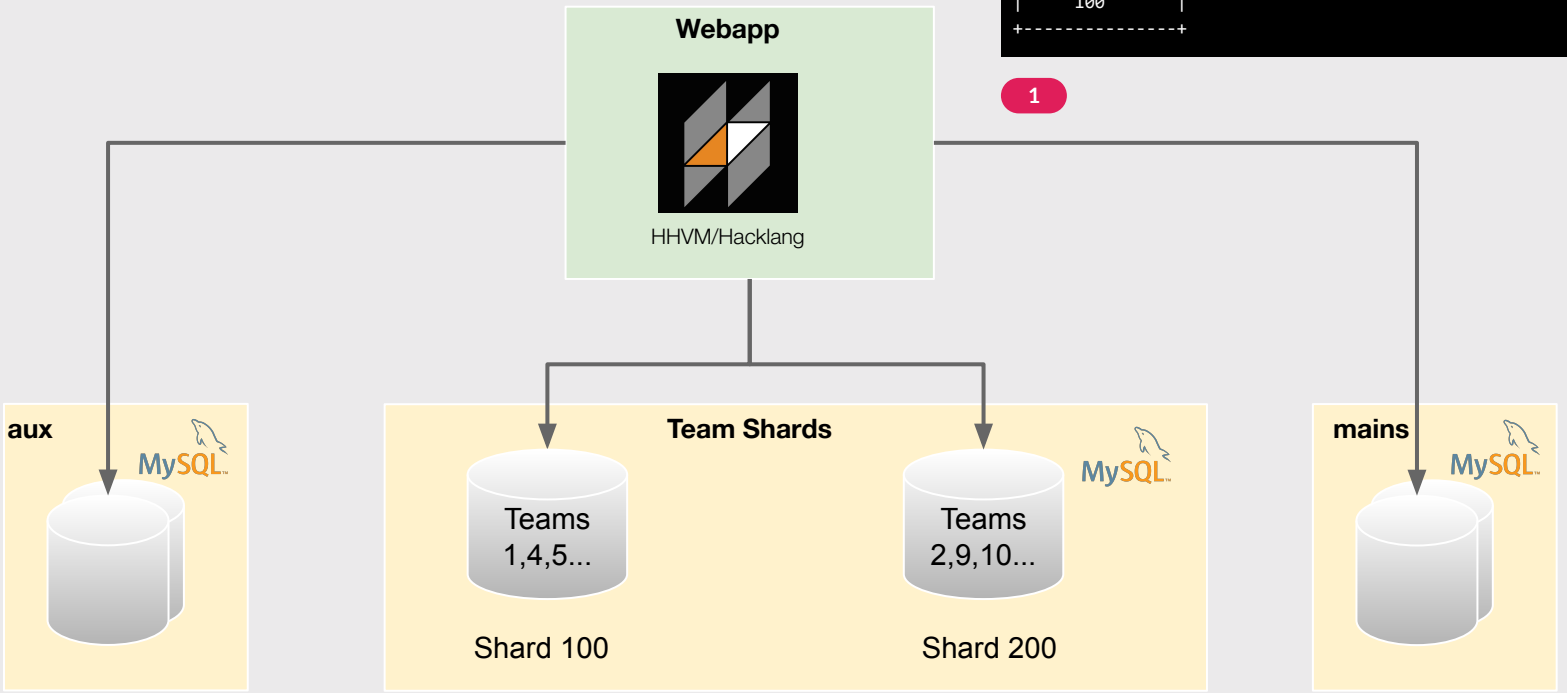
In the beginning...



In the beginning...

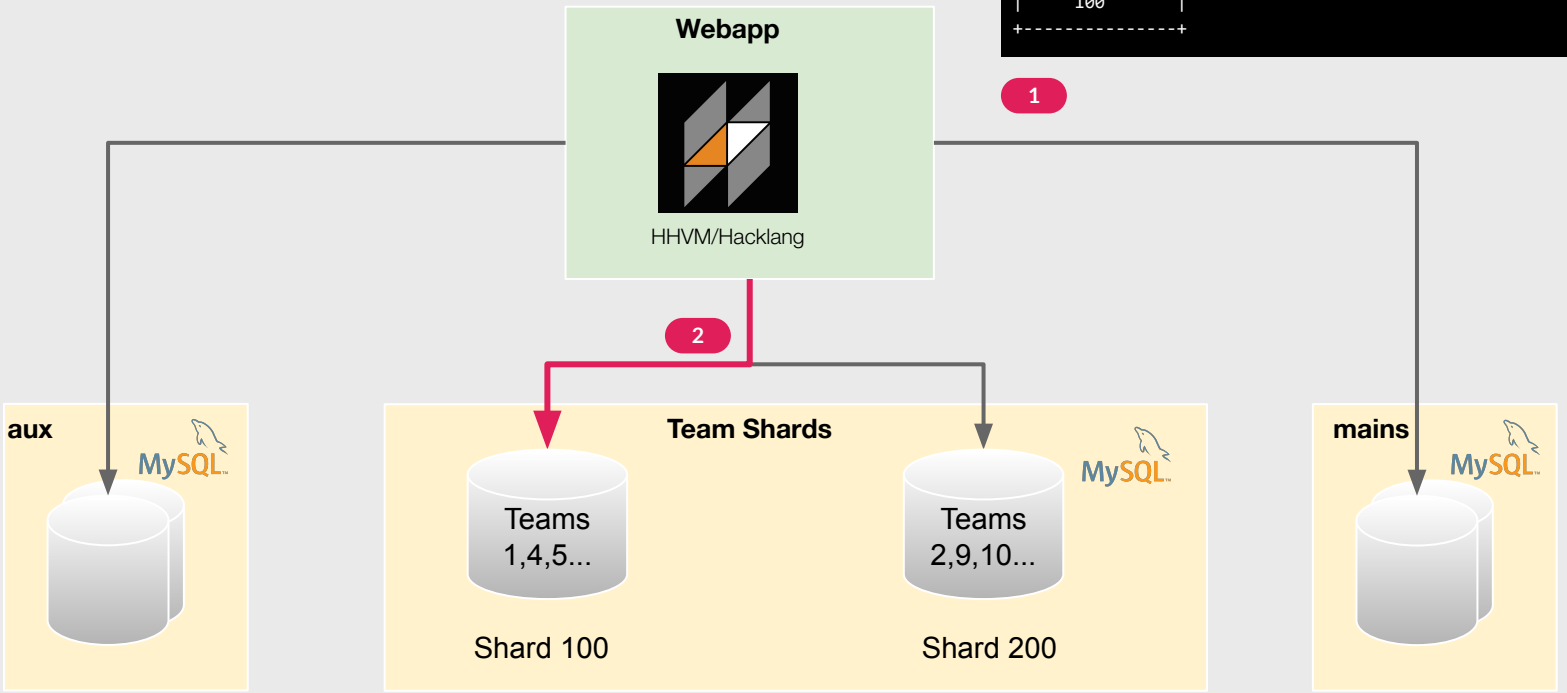
```
mysql > select db_shard from teams where id=1
+---------------+
|   db_shard    |
+---------------+
|     100       |
+---------------+
```

1



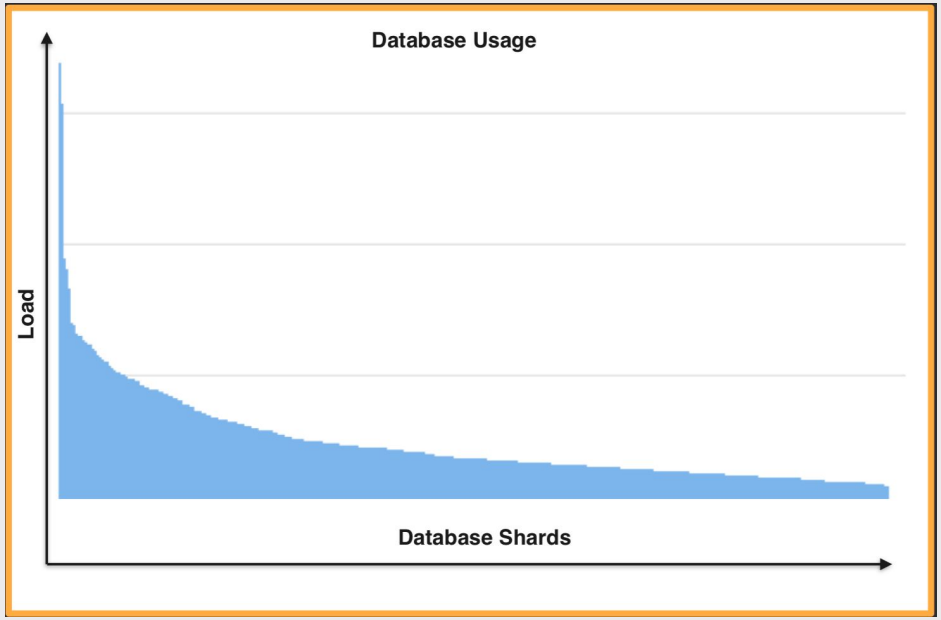
In the beginning...

```
mysql > select db_shard from teams where id=1
+---------------+
|   db_shard    |
+---------------+
|     100       |
+---------------+
```



Limitations

- Hotspots are a thing



Requirements

- Needs to support MySQL
- Provides flexible sharding strategy for different datasets
- Make sharding as transparent to the application as possible
- Horizontally scalable, highly available

Vitess

WHY VITESS?

- Abstraction of one giant MySQL database, with any number of tables backed by any number of scaled-out hosts.

- Enables table-by-table configuration of different sharding policies so clients no longer need to care about routing queries to specific shards.

For more details please see the **presentations** on the right.

tl;dr; shard size limits, inefficient resource distribution, operational overhead, single sharding model

- *Scaling Resilient Systems: A Journey into Slack's Database Service - Rafael Chacón & Guido Iaquinti.*
- *“Migrating to Vitess at (Slack) Scale” - Mike Demmer.*
- *“Designing and launching the next-generation database system at Slack: from whiteboard to production” - Guido Iaquinti.*

Vitess in front of legacy shards (VIFL)

Problem statement

Our migration model did not fit for all tables.

- Migrating ~70% of the workload to Vitess took ~2.5 years and it involved around ~10 tables (the most complicated and highest volume ones).
  - Multi quarter projects.
  - Many engineers!

- It was not really a migration, it was a re-architecture.

Visually: A Traditional Migration

A new challenge

How to migrate the long tail?

Problem statement

- The 30% traffic missing was from > 200 tables.
- Best case to complete the migration following this approach:

1 month / 1 table / 1 engineer = **16.5 years**!

VIFL

Project requirements

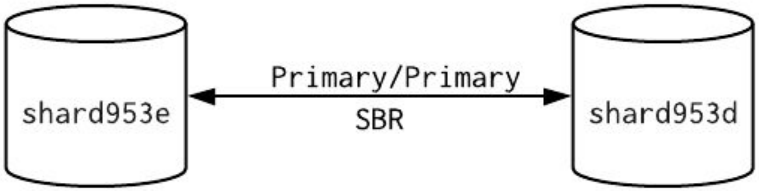
- Move the remaining 30% workloads in a year.
- Zero or minimal disruption of our development team workflow.
- Zero downtime allowed to perform the migration.

Visually: VIFL migration

VIFL Migration

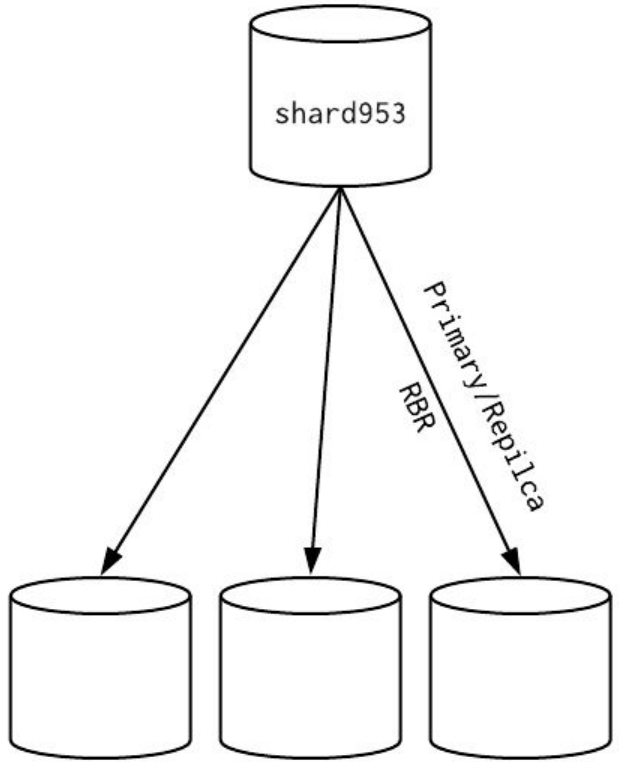
Legacy Topology

- MySQL 5.6.
- SBR: statement-based replication.
- Primary / Primary.
- Two servers per shard.
- Async replication.



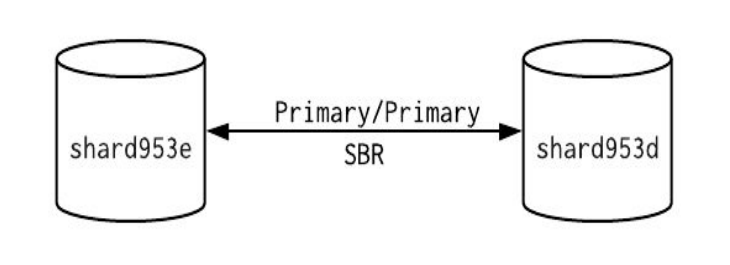
Vitess Topology

- MySQL 5.7.
- RBR: row-based replication.
- Primary / Replicas.
- N servers per shard.
- Semi-sync replication.

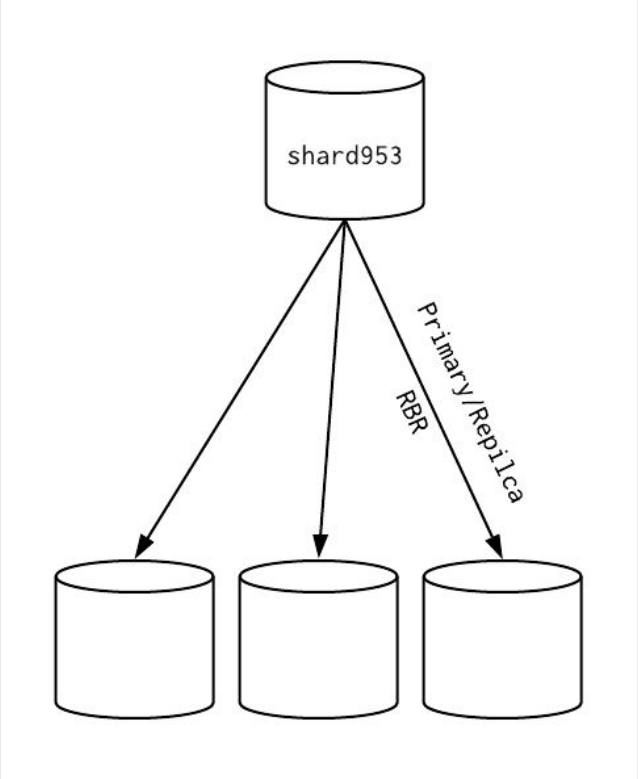


VIFL

Legacy Topology



Vitess Topology



VIFL

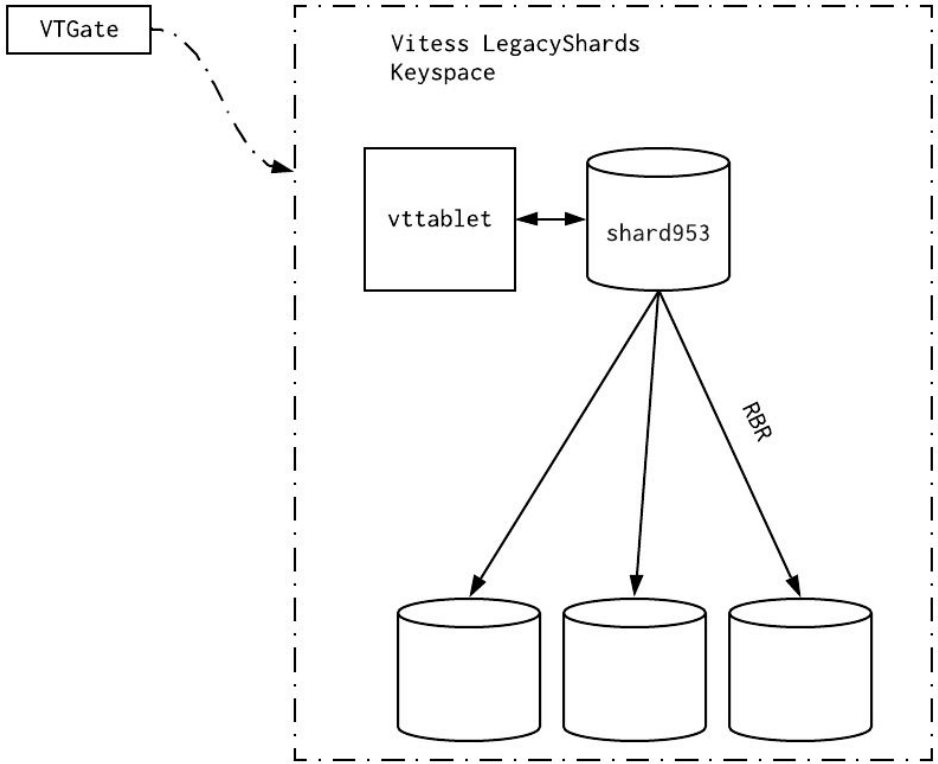
New migration framework

Set of steps

1. Restore and upgrade
2. Synchronize
3. Validate
4. Migrate

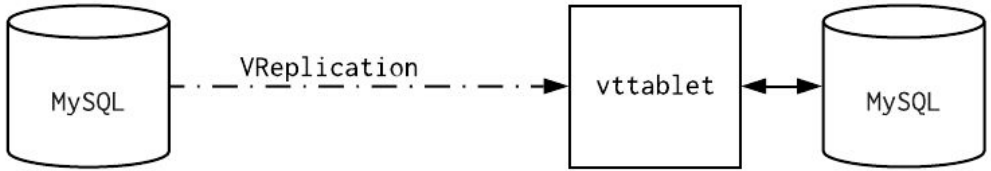
Restore and upgrade

- Create a new shard in Vitess corresponding to a legacy shard of the old architecture.
- Seed the new shard with the latest available backup from legacy.
- Once the restore is completed, perform an in-place upgrade of MySQL, migrating the dataset to the new version.
- Take a fresh backup that can be used to provision new hosts in the Vitess shard.



Synchronize

- Leveraging a core component of Vitess: **VReplication**.
- VReplication implements MySQL replication protocol to **shovel** data from an external database source to a target Vitess shard.



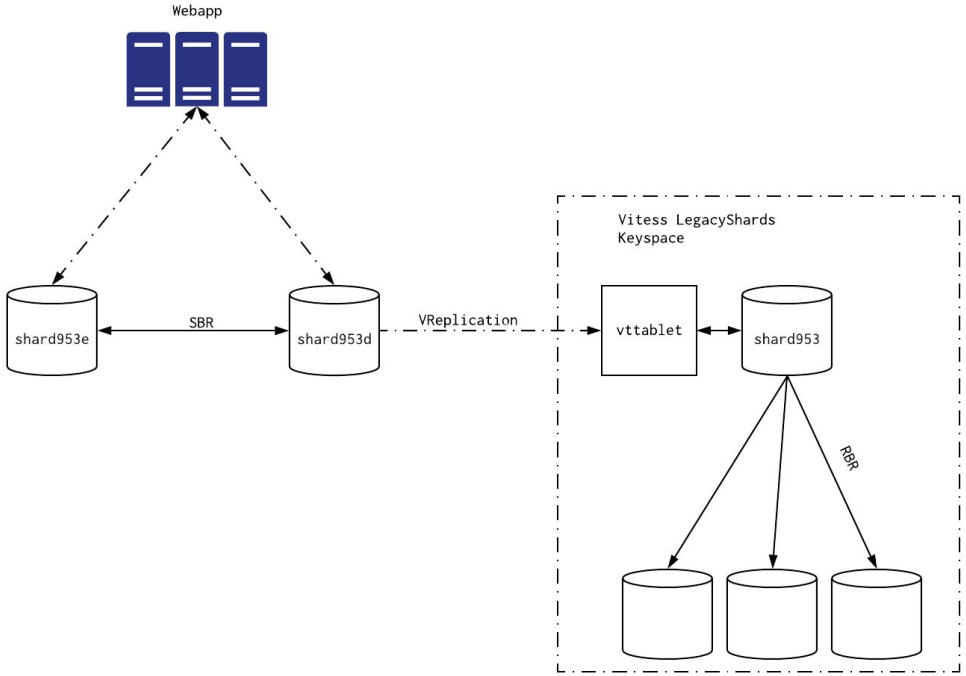
Synchronize

- Leveraging a core component of Vitess: **VReplication**.
- VReplication implements MySQL replication protocol to **shovel** data from an external database source to a target Vitess shard.

```
mysql> select * from vreplication\G
*********************** 1. row ***********************
    id: 1
    workflow: vtshovel
    source: filter:<rules:<match:"/.*" > > on_ddl:EXEC
external_mysql:"source_mysql_db"
    pos: FilePos/mysql-bin.000048:59491335
```

Synchronize

- Leveraging a core component of Vitess: **VReplication**.
- VReplication implements MySQL replication protocol to **shovel** data from an external database source to a target Vitess shard.



Validation

- How do we know we didn’t leave any data behind?
- Is the data matching from the application perspective?
- Is this process reliable?

Two types of validation

1. Database
2. Application

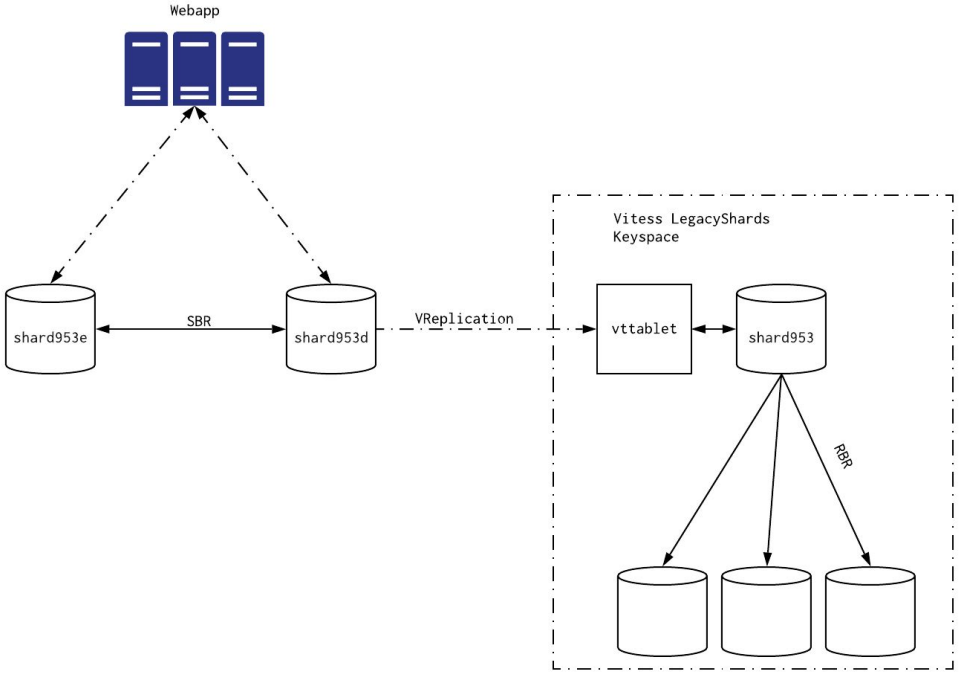
Validation (databases)

- In vacuum this would have been an easy task.

```
for table in tables {
  source_table :='select * from source_db.$table'
  dest_table :='select * from dest_db.$table'
  if source_table != dest_table {
     panic "there are diffs"
  }
}
```

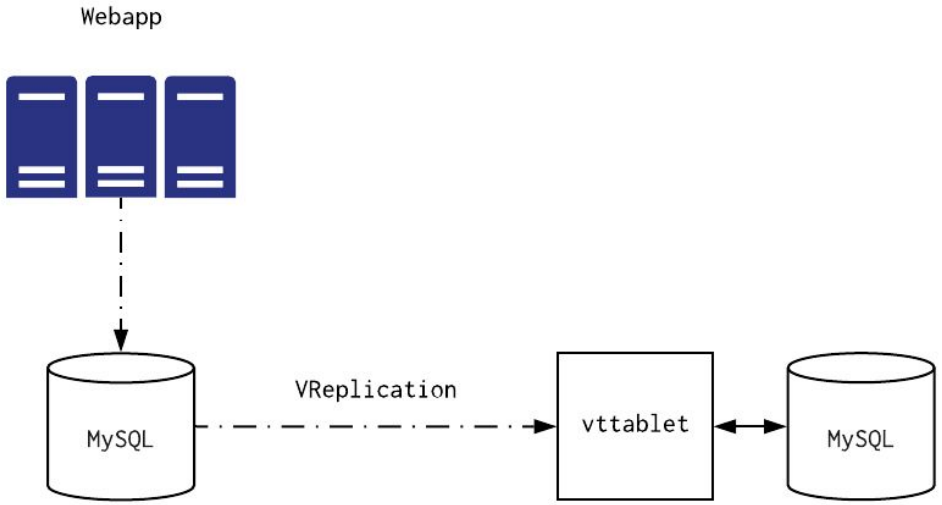
Validation (databases)

- In vacuum this would have been an easy task.
- Unfortunately both databases are constantly changing (due to write traffic), we are not in the vacuum and an atomic comparison of potentially terabytes of data is not feasible.



Validation (databases)

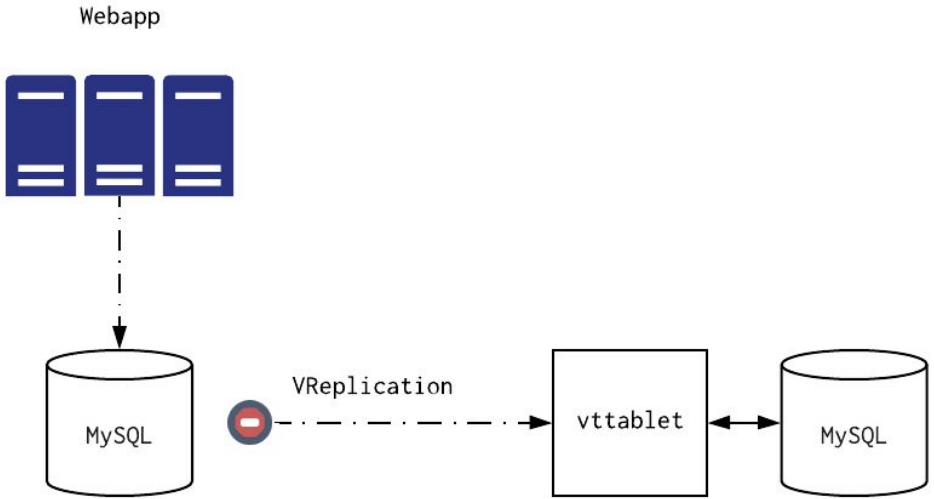
We want a consistent snapshot between two databases that are taking traffic at the same time.



Validation (databases)

Step 1

- stop VReplication

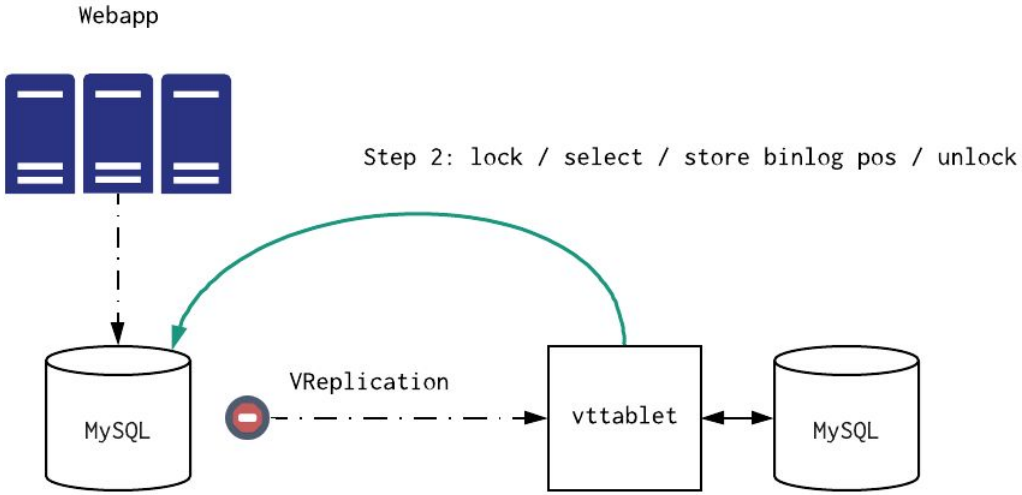


Validation (databases)

Step 2

- In the source: lock table X
- Issue streaming SELECT * from X in the source.
- Record binlog position (**alpha**).
- In the source: unlock table X.

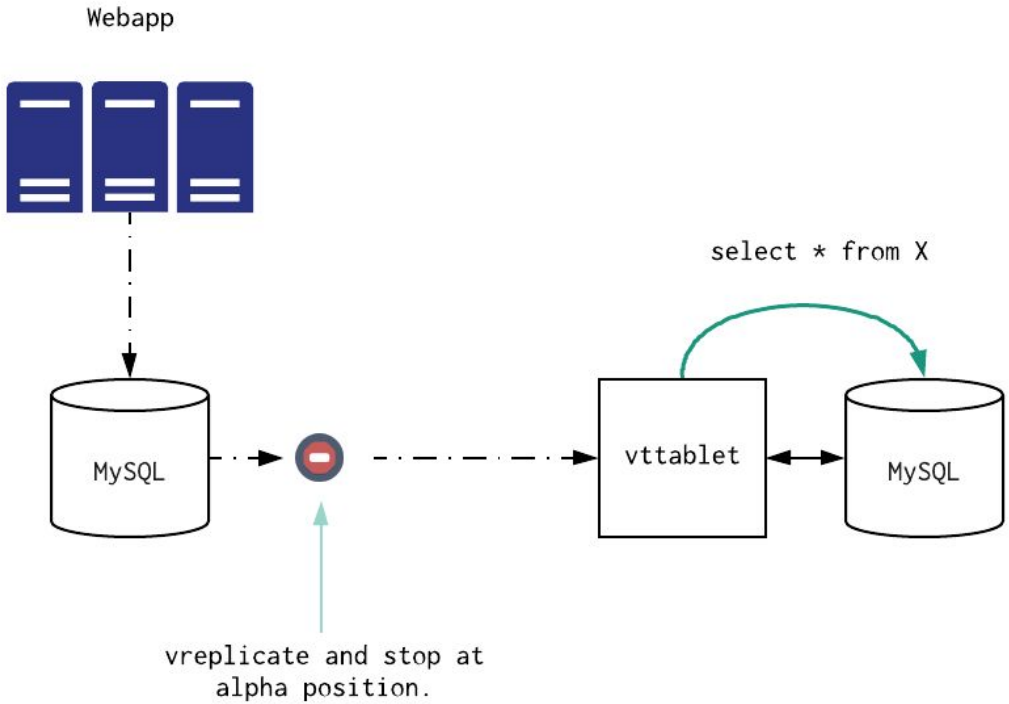
This whole operation is really fast, you only lock the table for a few milliseconds.



Validation (databases)

Step 3

- Start vreplication and stop it again at **alpha** position.
- Issue streaming SELECT for X in the destination.
- Start vreplication.

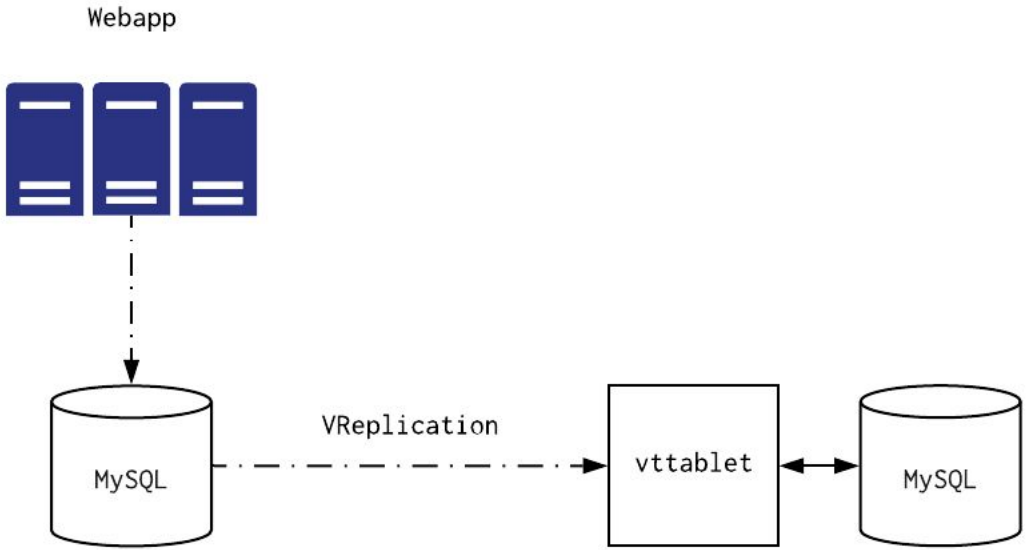


Validation (databases)

Step 4

- Compare the data of the table.
- Iterate for all the tables present in the source.

Done!



Validation (databases)

How do we know we didn’t leave any data behind? ✔️

Do we have the same data in the source and destination datastores? ✔️

The database validation is assuring us that the content of all tables is matching **at a specific transaction/timestamp**.

Validation (application)

Some assumptions.

Scale
We expected a performance regressions mostly driven by the additional network latency:

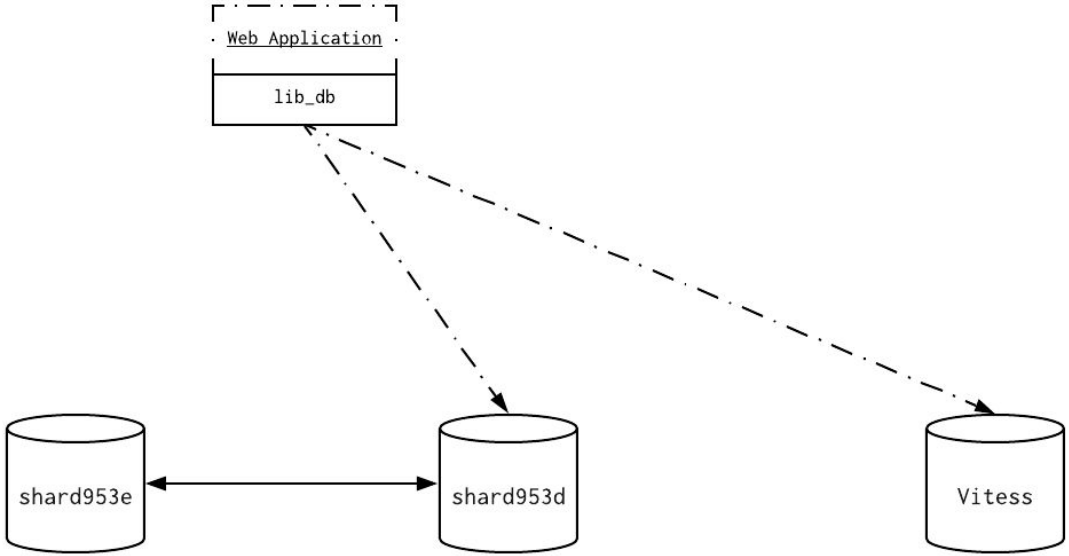
- 1 x RTT during reads (due to the extra network hop in vtgate).
- 2 x RTT during writes (due to the extra network hop in vtgate + semi-sync ack from MySQL replication).

This slowdown is acceptable by the application and it won't be noticeable in the overall performance.

Correctness
All queries should be compatible (Vitess V2 "passthrough" routing). We needed to verify that.

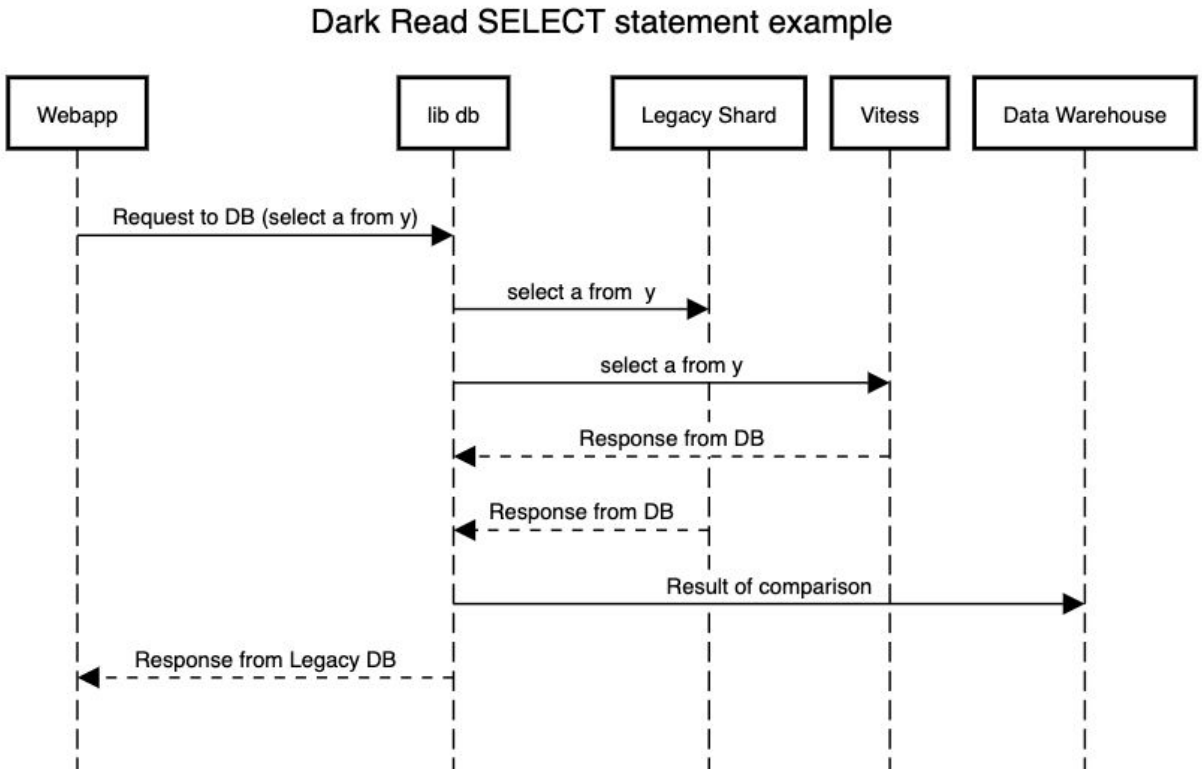
Validation (application)

We built a framework within our app to validate scale and correctness of the the new system.



Validation (application)

We built a framework within our app to validate scale and correctness of the the new system.



Validation (application)

The majority of query results were matching but we had to manually investigate some outliers.

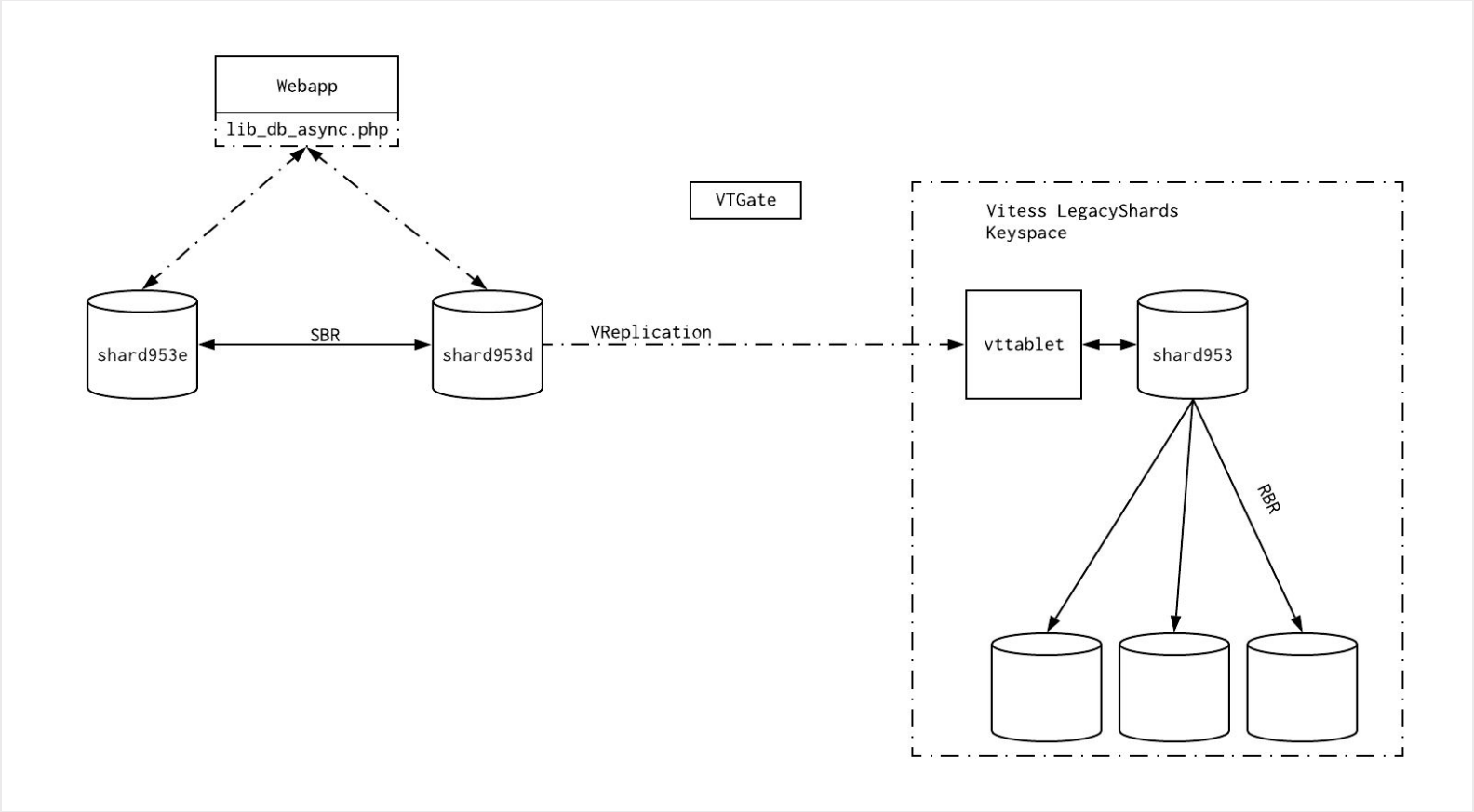
	A	B	C	D	E	F
1	Claimed by	Query Hash	Query Count	Match %	Resolution	Notes
2	rafael	13402318e5486375ac443f5f7acbcd85	42635	0.02345490794	Probably OK	Another team_channels_shared with no order clause. This is part of db_teams_channels_shared which is autogenerated code. I think we should just ignore this hash.
3	rafael	65cfb17de05fc0e626f12bd9d6dcc6be	50649	0.03356433493	Probably OK	Same as 13402318e5486375ac443f5f7acbcd85. Resolution is to ignore.
4	rafael	a23d4d6e22507e0bae6cead394a360ae	130	7.692307692	Needs further investigation	This might be another ordering issue, but is not obvious. To be sure, I propose we change the code to be explicit about it.
5	rafael	afefa5ba61b4a6f24697de89b3cc4995	206	44.17475728	OK	
6	rafael	eeef5322dd20c47e3a6b42c01c813eb1	55	45.45454545	Probably OK	Order by id issue. Can be verified by
7	rafael	8c83af210055c27de5b2470781eb6b2d	149	55.03355705	Needs further investigation	Most shards are matching 100%. 648 reporint 95% due to content. This seems to be an order issue as well. Very similar to a23d4d6e22507e0bae6cead394a360ae
8	rafael	b48d0b55a06af11d8b0a1e540e5a2790	3626	79.45394374	Probably OK	Matching for most shards. Only a few reporting diffs. Also confirmed that diffs in tinyspeck are due to ordering. The following query matches in both vifl and legacy:
9	rafael	5bc3a35f33bb288743f1d4e1d8a38df9	8	87.5		This is from db_teams_channels_shared_fetch_team_async which is autogenerated, we can't apply the order by here. Should go away with @rbailey fancy diffing.
10	rafael	bb9a5b273b4c6aed8c0a7eafd300b257	35926	90.0128041	OK	Query is missing order clause sometimes. I think solving that should make the problem go away. This PR addresses this one:
11	rafael	dc8e1dcf72f688d29957167f9e0de80e	3926	97.3764646	Probably OK	Most shards have 100% match rate for this query. Given that this is also a channels query without order, I'm leaning to think that in some cases is not returning values with the same order.
12	rafael	e4efd7db5a5f973170dc7f17bca5434e	113	98.2300885	OK	This was an update query than in two instances affected rows had a different value. This could be easily explained to a race. All the other shards are at 100%.
13	rafael	19902c9ca133f61dd2fdca49faa56044	594	98.98989899	OK	Same as above, this table seems hot and sometimes affected_rows does not match.

Validation (application)

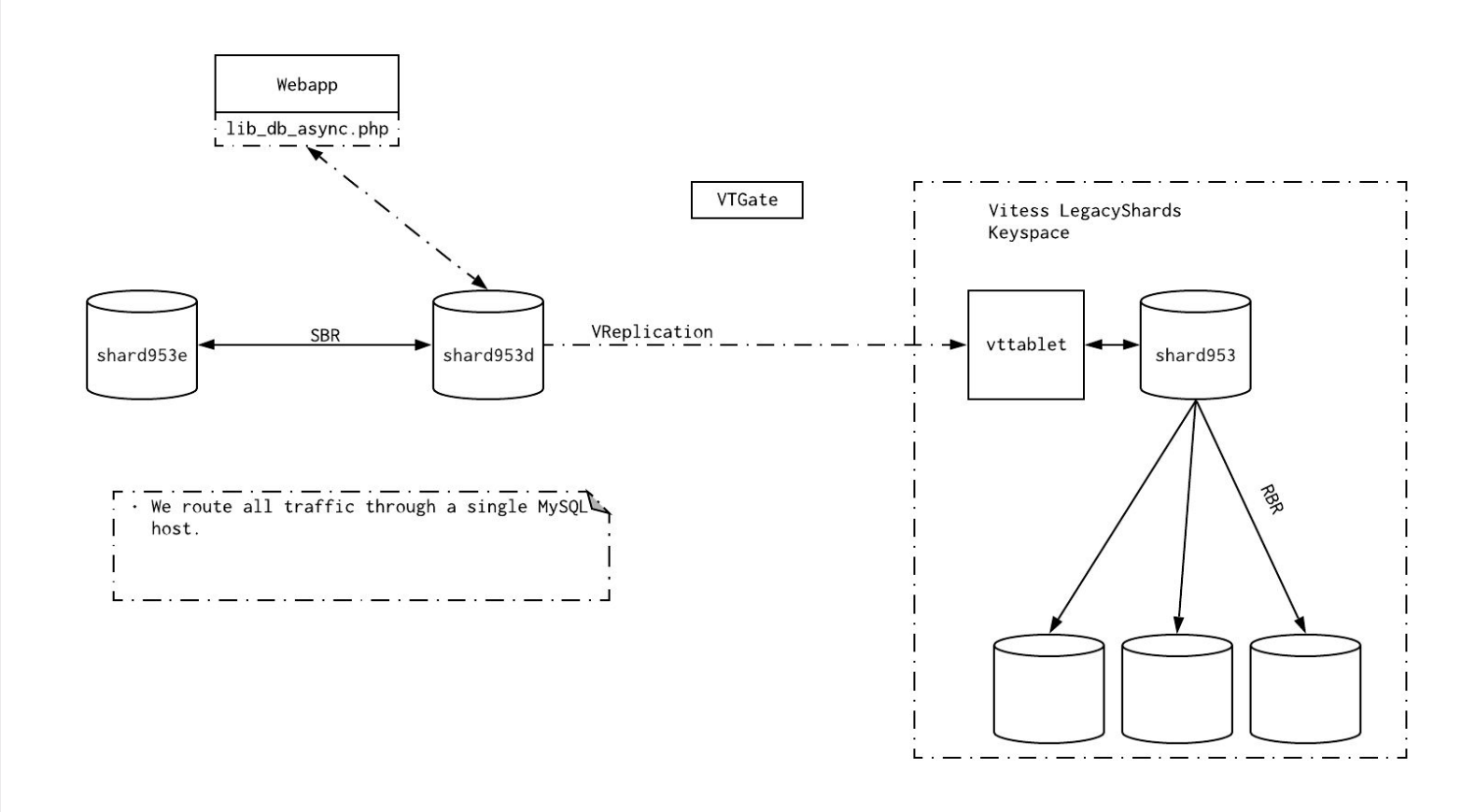
The majority of query results were matching but we had to manually investigate some outliers.

- We ran these tests for over two months.
- We analyzed every single diff.
- Most of the errors were driven by:
  - changes in order preferences in MySQL from 5.6 to 5.7 when explicit order was not provided.
  - places where the application expected read after write semantics.
- We got to this phase within the first 4 months of the project.
- We concluded it was safe to proceed.

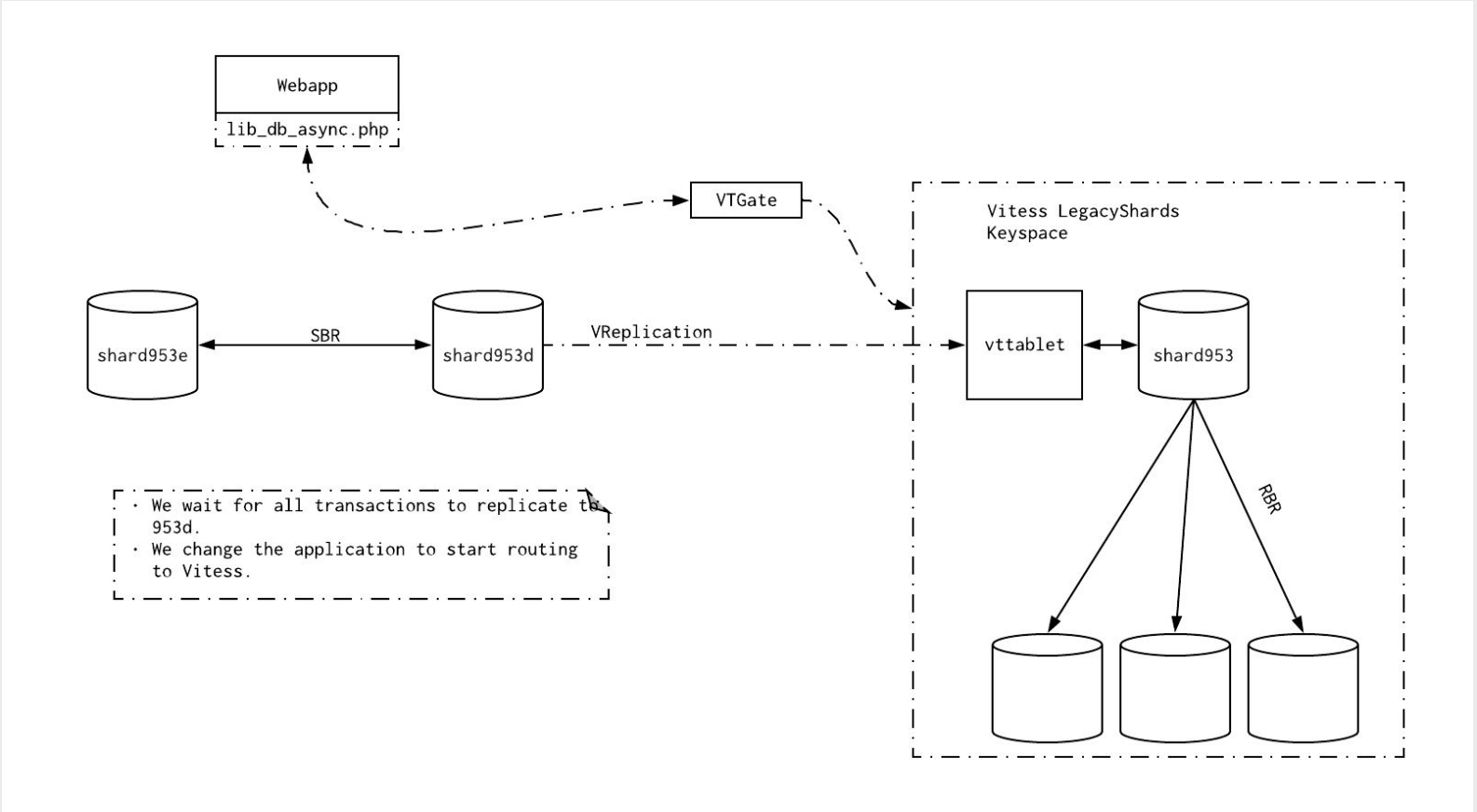
VIFL - MIGRATION



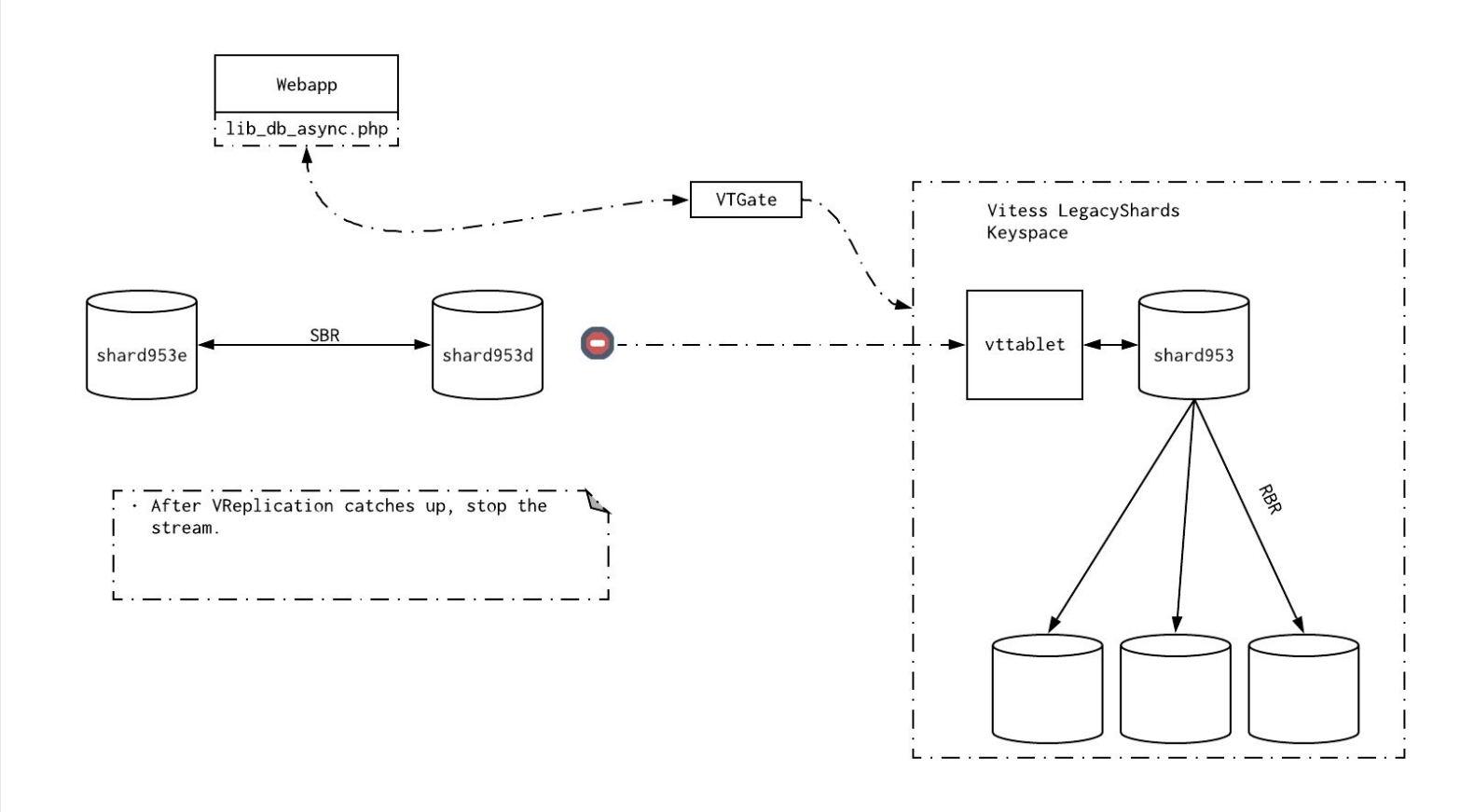
VIFL - MIGRATION



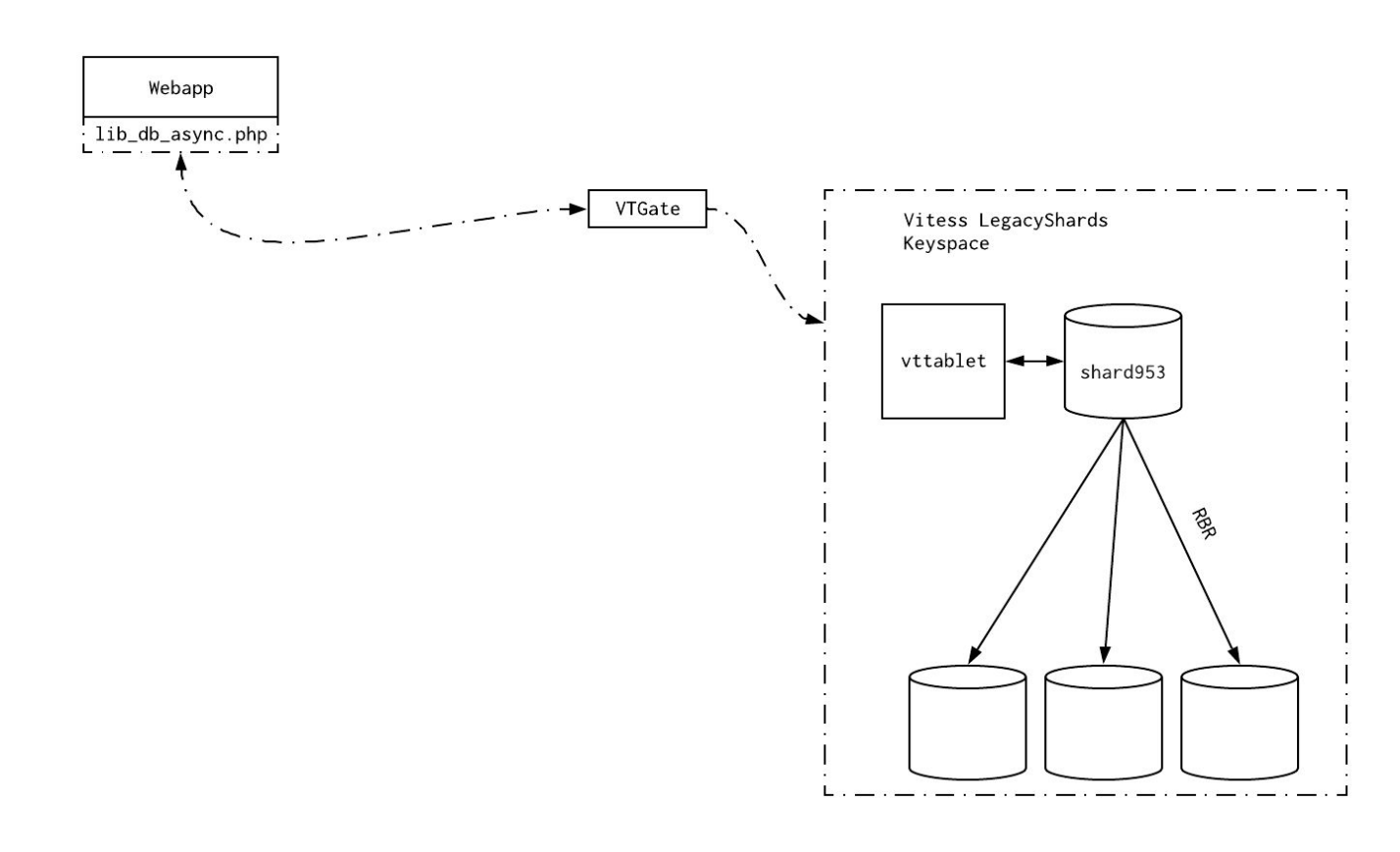
VIFL - MIGRATION



VIFL - MIGRATION



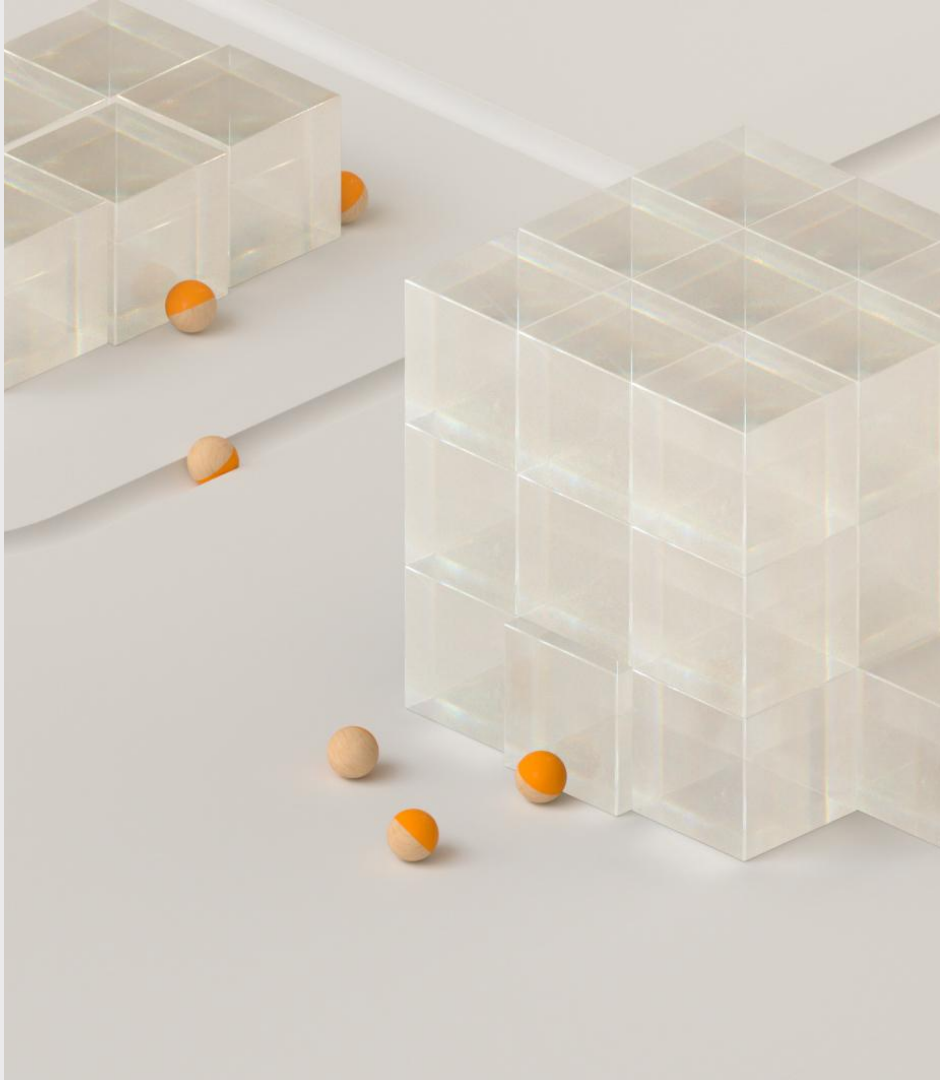
VIFL - MIGRATION



We validated that the core idea is working

Now we only have to:

- repeat this process more than a thousand times.
- with no errors.
- with zero downtime.



Goal

Build an automation to execute the VIFL migration procedure that is **repeatable** and **safe**.

Repeatable: suitable to be executed thousands times with little overhead and zero or limited human intervention.

Safe: no room for mistakes (from human or robots).

Toolkit

- Built using Python 2.7 with the use of our internal (but soon deprecated) SlackOps library.

- A lot of cross system and service interactions
  - OS
  - MySQL
  - DBConfig (our legacy database service discovery system)

Automation was built and completed between January and April 2020.

Defensive coding

Always plan for the unexpected.

We built the automation as a **state machine** made by several **idempotent** steps:

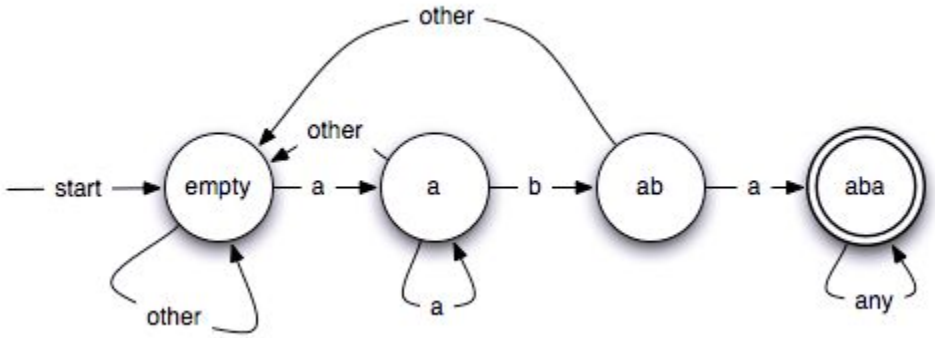
- check_prerequisites
- seed
- provision
- validation
- migration
- cleanup

State machine

- Predictable components.
- Steps are easy to implement, change, test, and reuse.

We defined very **strict boundaries** and **allowed actions** for each step of the state machine.

Each step interacted with the previous/following one only via **predictable interfaces**. The compartmentalization was very useful to isolate each step and make easier the development, test and reuse of components.



Idempotence

We also made sure that every step of the automation was also **idempotent** (safe to be re-run multiple times without changing the final result).

Implementing this property for all the steps helped us to **build confidence** in the tool as well as allowing robots and humans to **recover from transient issues** during the execution.

Safety guardrails

State machine properties and idempotence are very valuable characteristics for an automation but are them enough to archive our strict requirements? Not in our case. We also built safety guardrails against:

- **robots**: other automation tools concurring for shared resources (e.g. schema change, backup, shard split)
- **ourself**: human errors (e.g. try to migrate shard id 0123 when validation was executed for shard id 4567)

Final remarks

FINAL REMARKS

Results (1 of 2)

- This project was designed, built and executed by a team of 4 engineers in a timeframe of ~1 year. We calculated that by following the legacy migration path it would have took > 70 engineers to deliver the same result in a similar timeframe.

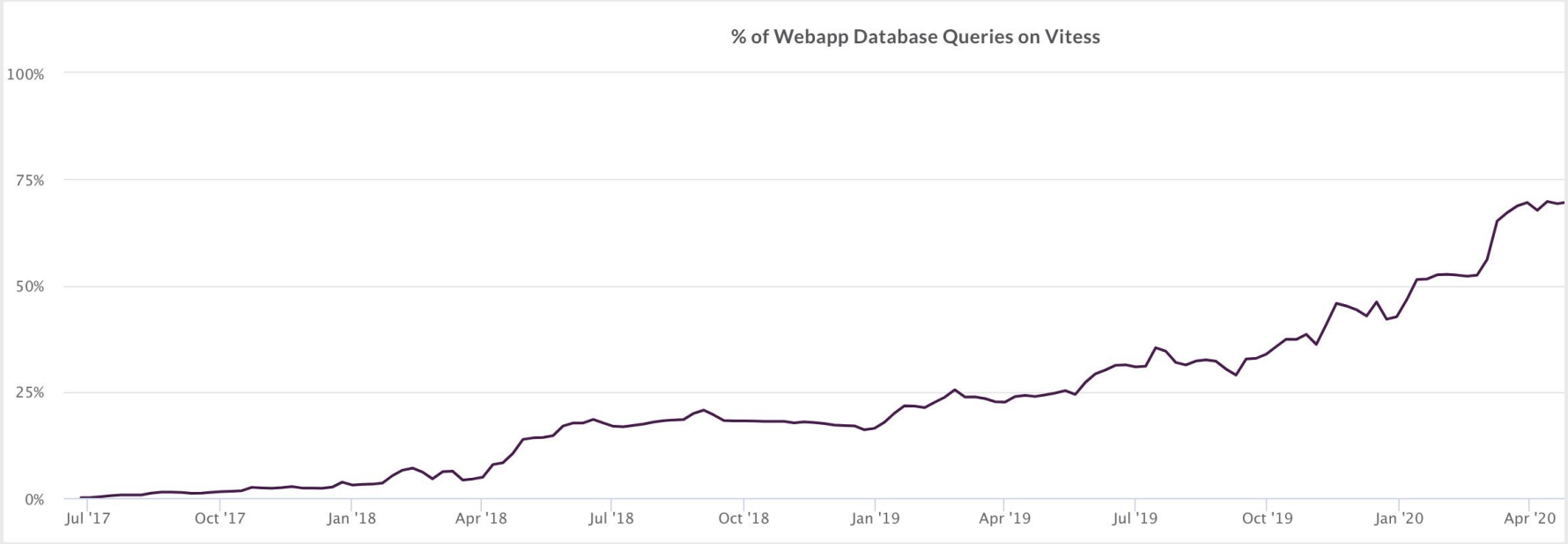
- Equally important, this migration was completely transparent to our application engineers as well as end users.

FINAL REMARKS

Results (2 of 2)

- By leveraging the VReplication functionality in Vitess, we came up with a strategy that enabled us to migrate entire clusters to Vitess instead of the previous table by table procedure.
- We moved hundreds of terabytes of data. Over 1000 MySQL shards. Zero downtime, not a single outage.
- Today, 99% of Slack traffic is on Vitess and we are expected to wrap up this migration by the end of the year.

FINAL REMARKS

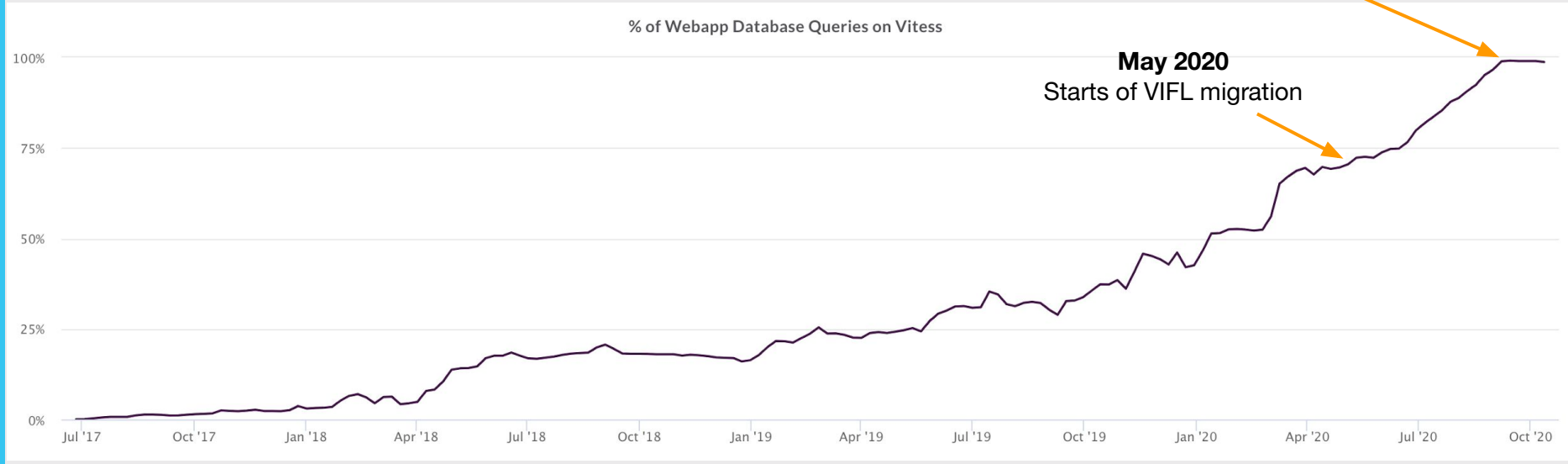


Jul 2017
Start of the
Vitess journey

Apr 2020
'message' table is migrated,
70% of adoption is reached

FINAL REMARKS

Sep 2020
99% of the VIFL migrations are completed
+30% adoption in < 6 months



May 2020
Starts of VIFL migration

Apr 2020
'message' table is migrated,
70% of adoption is reached

FINAL REMARKS

Unexpected Challenges

Things never goes as expected.

We unfortunately also faced some unexpected challenges while preparing the migration of few of our busiest shards:

- MySQL 5.7 optimizer performance regression
- overhead of vttablet was noticeable
- overhead of golang GC
- latency regression at high percentiles (p99, p999)

FINAL REMARKS

Conclusions

- Breaking down the **validation** step and the **migration** mechanism was key for us.
- We iterated over the initial design several times, balancing speed, execution time and safety.
- Was this a success? We accomplished everything that we said we were going to do despite some hiccups. Today 99% of Slack's databases are running on Vitess.
- Could you replicate this elsewhere? Yes if, same assumptions hold:
  - Hit on P50 latency (due to additional network hops)
  - Enough headroom to absorb the cost of running the vttablet process in front of MySQL.

Suggested session

Vitess: Introduction and New Features
Sugu Sugumarane & Deepthi Sigireddi, Planetscale, Inc

Wednesday, November 18 • 5:45pm - 6:20pm

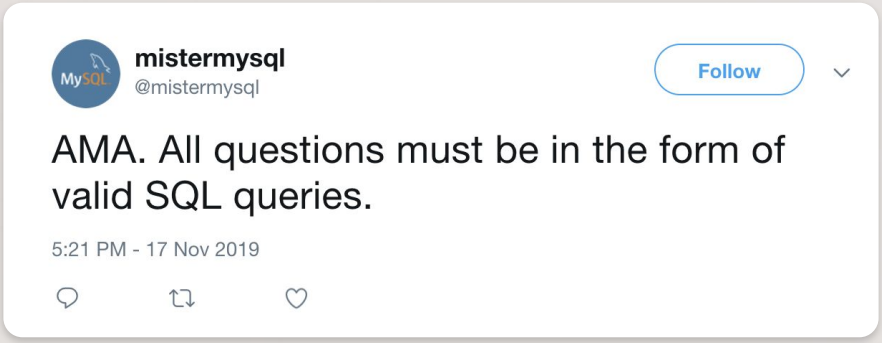
slack

Thank you!

P.S. We are hiring!

slack

Q&A


mistermysql
@mistermysql
Follow
AMA. All questions must be in the form of valid SQL queries.
5:21 PM - 17 Nov 2019

slack