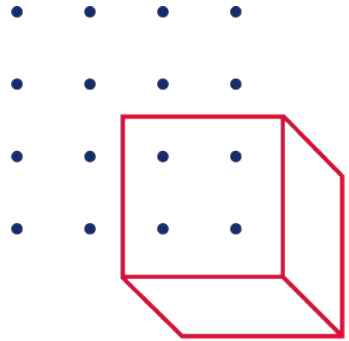
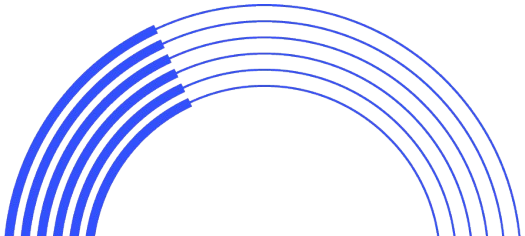
PingCAP



TiDB Workshop

Friday 28 July, Berlin, WeAreDevelopers



PingCAP

Table of Content

Introduction

PingCAP

- Mattias Jonsson
  - Working for PingCAP as Senior Database Engineer, developing TiDB
  - Previous:
    - Senior Developer / Engineering manager at Booking.com
    - Senior Software Engineer at MySQL/Sun/Oracle

- Daniël van Eeden
  - Working for PingCAP as Technical Support Engineer
  - Previous
    - Senior Database Engineer at Booking.com
    - Long time member in the MySQL Community

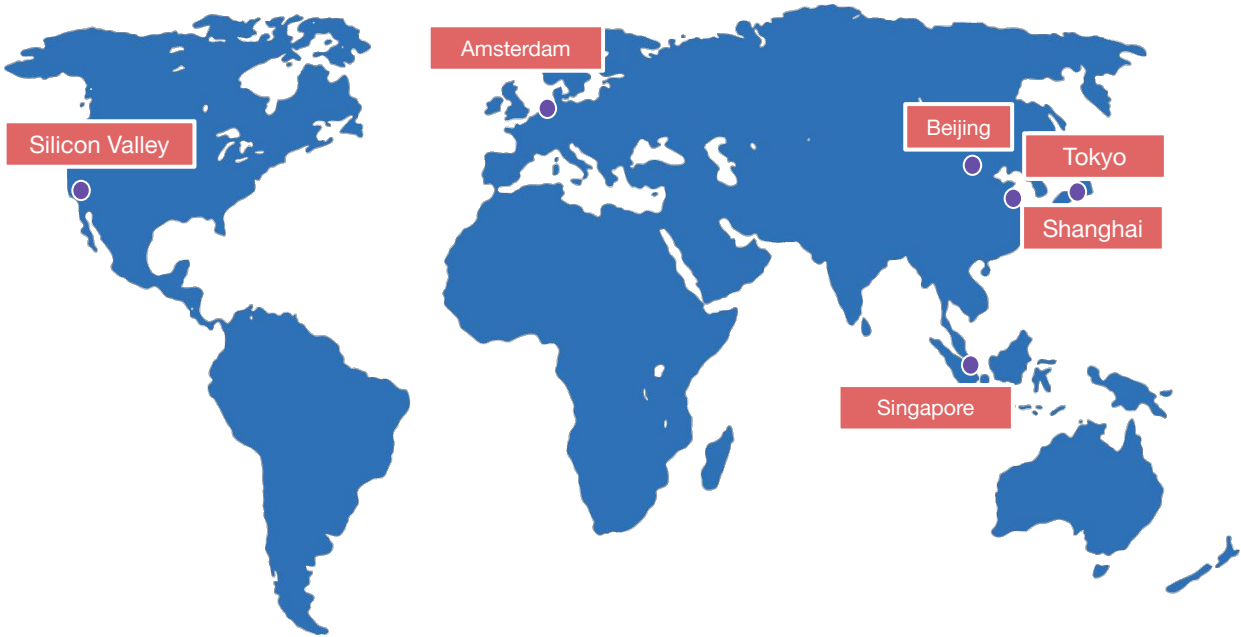
About PingCAP

PingCAP, the company behind ***TiDB***, provides enterprise-level and cloud-based services and technology for TiDB, so companies can count on TiDB as its core database solution to simplify the database infrastructure and create business value faster than ever.

Mission: Empower engineers to innovate with Speed, Agility, and Scale.

About PingCAP

- Founded in 2015
- The company behind TiDB
- Global
- Open source culture
- Strong investors
- 600+ employees



About PingCAP

Trusted and Verified by Vertical Leaders Globally

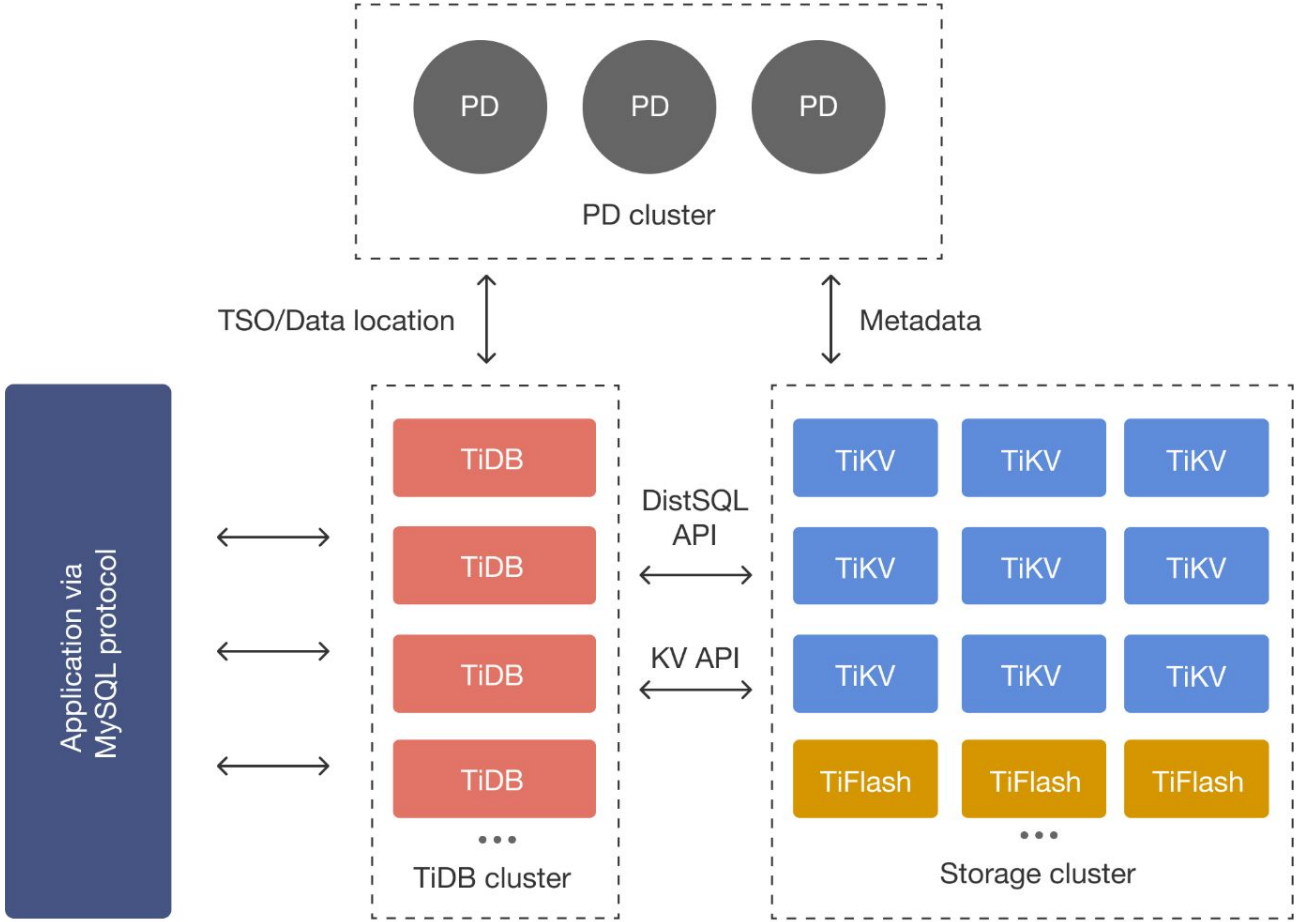
2000+ production adopters

And

Airbnb, Pinterest, Niantic

2 out of top 3 Cryptocurrency Exchanges

One of Top 3 biggest banks all over the world


PD
PD
PD
PD cluster
TSO/Data location
Metadata
Application via MySQL protocol
TiDB
TiDB
TiDB
TiDB
TiDB cluster
DistSQL API
KV API
TiKV
TiKV
TiKV
TiKV
TiKV
TiKV
TiKV
TiKV
TiKV
TiFlash
TiFlash
TiFlash
Storage cluster

Environment setup

PingCAP

Database:

- TiDB Locally with TiUP via https://tiup.io
- MySQL Locally
- Optional: TiDB Cloud
  - via https://tidbcloud.com (Free serverless tier)
  - select eu-central-1 as location for your cluster

Application:

- demoblog
- https://github.com/dveeden/demoblog

Database client:

- MySQL Client
- TiDB Cloud Console (Chat2Query)

Environment setup

PingCAP

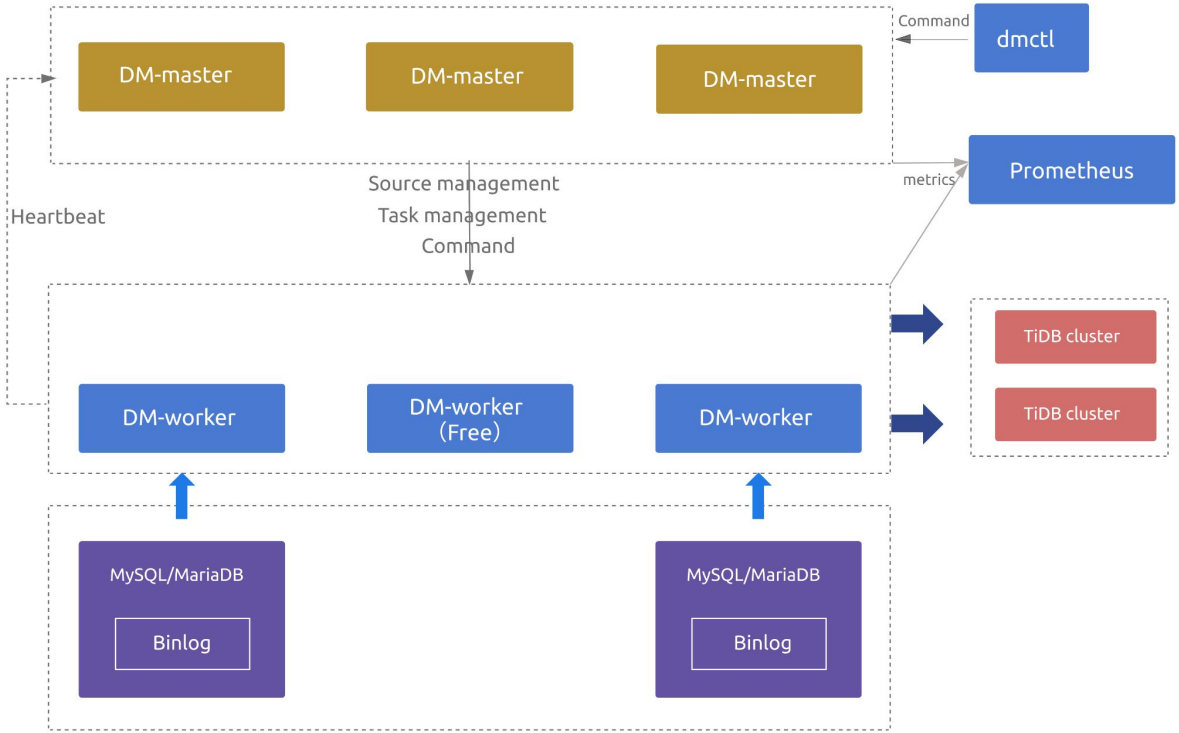
- AWS
- This presentation (for easier li
- EC2 hosts, pick one from this spreadsheet.
- shared keypair
  - For simplicity we use the same private key for ssh
  - Log in with your ssh client, like:
    `ssh -i tidb-workshop.pem ubuntu@3.78.221.19`
  - Download the demoblog repository:
    `git clone https://github.com/dveeden/demoblog.git`
  - Setup the environment (will also start MySQL Server)
    `./demoblog/AWS/setup_workshop.sh`

- port forwarding (for later)
  - `ssh -L8080:127.0.0.1:8080 -L2379:127.0.0.1:2379 -L 3000:127.0.0.1:3000 -i tidb-workshop.pem ubuntu@3.78.221.19`
  - http://127.0.0.1:8080 Demoblog
  - http://127.0.0.1:2379/dashboard PD dashboard (root/)
  - http://127.0.0.1:3000/ Grafana (admin/admin)

Migrate to TiDB from MySQL

Data Migration (DM)

- Replicate data from MySQL to TiDB.
- Uses Dumpling/Lightning to copy the initial copy.
- Note that this is also made to be high available.



Lab excercises: DM 1/2

PingCAP

1. Run demoblog with MySQL

```
~/demoblog/demoblog -db 'blog:blog@tcp(127.0.0.1)/blog?parseTime=true'
```

2. Make sure the application works (check http://127.0.0.1:8080 (via portforward))
3. Run TiUP Playground (in a new ssh session)

```
tiup playground v7.2.0
```

4. Run DM master (in a new ssh session)

```
./dm-master -master-addr 127.0.0.1:8261
```

5. Run DM worker (in a new ssh session)

```
./dm-worker -worker-addr 127.0.0.1:8262 -join 127.0.0.1:8261
```

6. Check DM status (in a new ssh session)

```
./dmctl -master-addr 127.0.0.1:8261 query-status
```

Lab excercises: DM 2/2

PingCAP

Repo:

- DM/mysql-source.yaml
- DM/dm-task.yaml

```
cd demoblog/DM
~/dmctl -master-addr 127.0.0.1:8261 operate-source create mysql-source.yaml
~/dmctl -master-addr 127.0.0.1:8261 start-task dm-task.yaml
~/dmctl -master-addr 127.0.0.1:8261 query-status mysql-to-tidb
```

Check:

```
sudo mysql blog
mysql -u root -h 127.0.0.1 -P 4000 blog
```

Query: `SELECT * FROM posts;`

Stop application and run again with TiDB (go back to the demoblog session, to simulate a switch to TiDB)

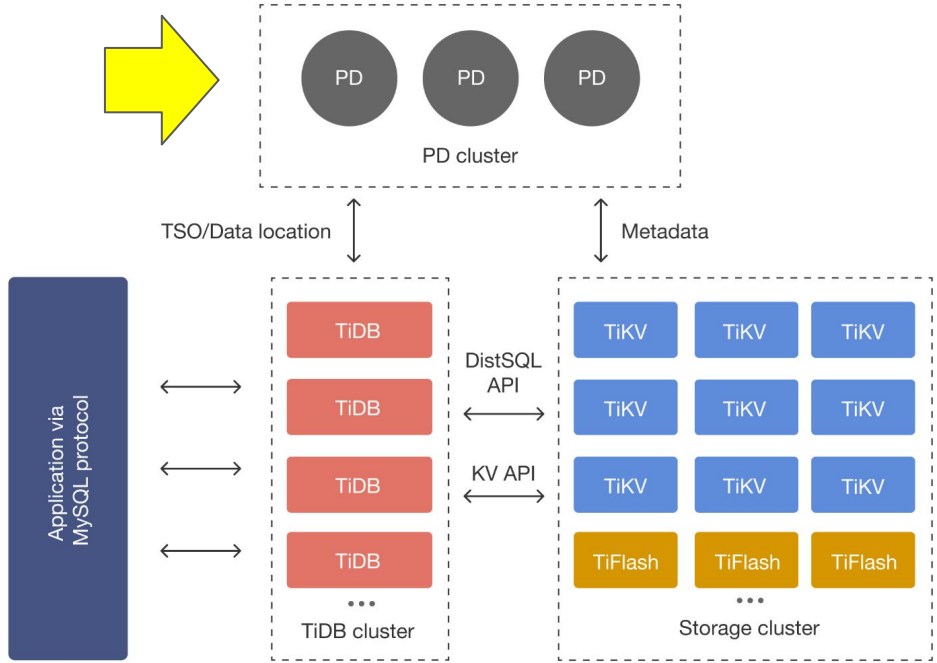
```
^C
# (demoblog uses TiDB by default, so no uri needed)
~/demoblog/demoblog
```

OK to stop the dm-master and dm-worker processes now, since traffic to MySQL is moved to TiDB.

Placement Driver (PD)

The placement driver:

- Has etcd embedded
- Is the Time Stamp Oracle (TSO): it gives out timestamps. These timestamps are used in transactions and in the MVCC system
- Takes care of data placement:
  - TiKV servers have labels
  - labels are used to ensure a raft group spans multiple availability zones
  - Splitting big data regions, merging small data regions, splitting hot data regions, evenly distributing regions across the cluster.



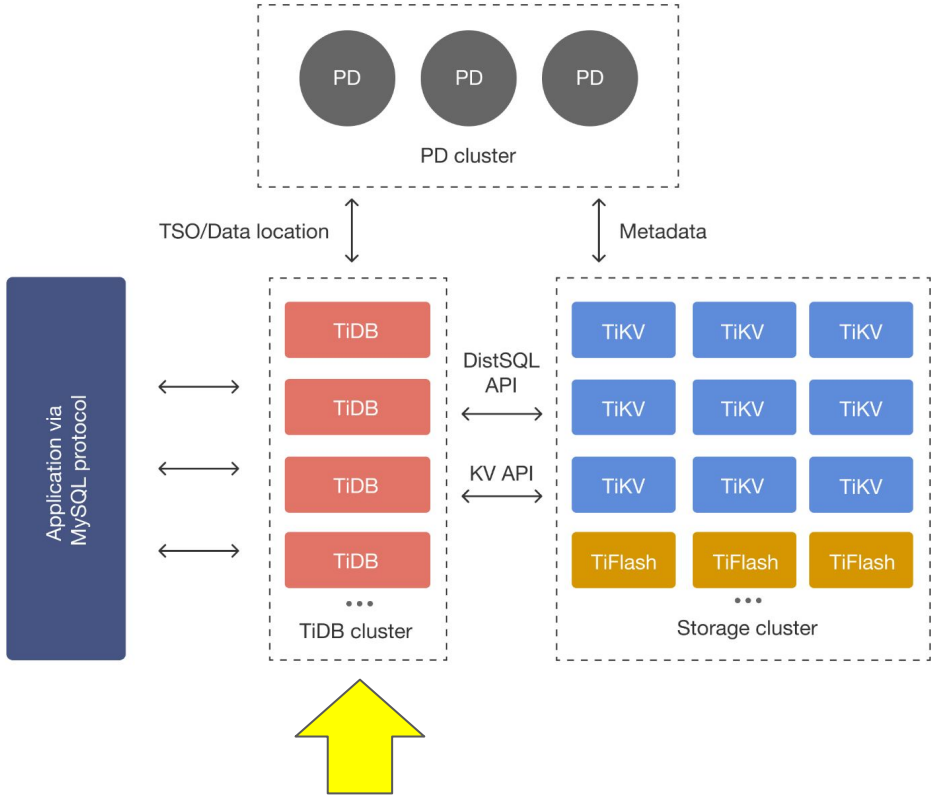
TiDB Server

TiDB Server is one of the components of the TiDB Platform. This is a bit confusing.

TiDB Server is written in Go and doesn't share any code with MySQL.

This implements the MySQL protocol, optimizer and executor.

TiDB sql node is stateless, it doesn't contain data.

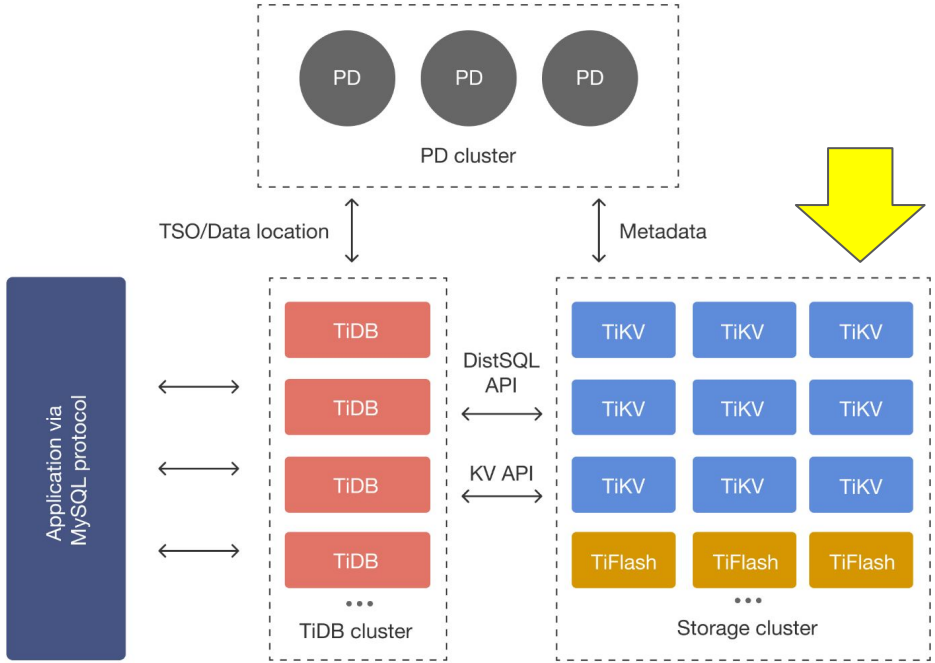


TiKV

TiKV is a distributed key-value store. This is a CNCF project.

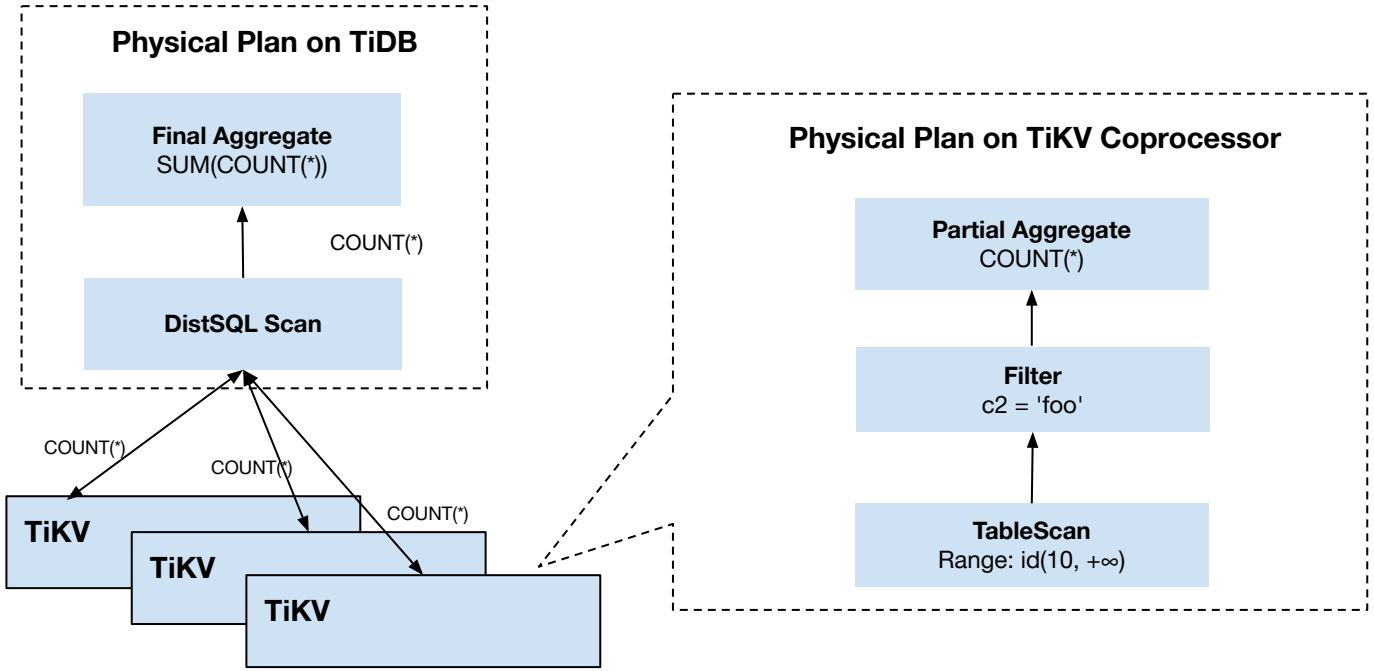
Database tables are stored with a RowID or PK as key and the columns as values.

A database table is split up into multiple data regions. Each data region is a raft group of (by default) three nodes.

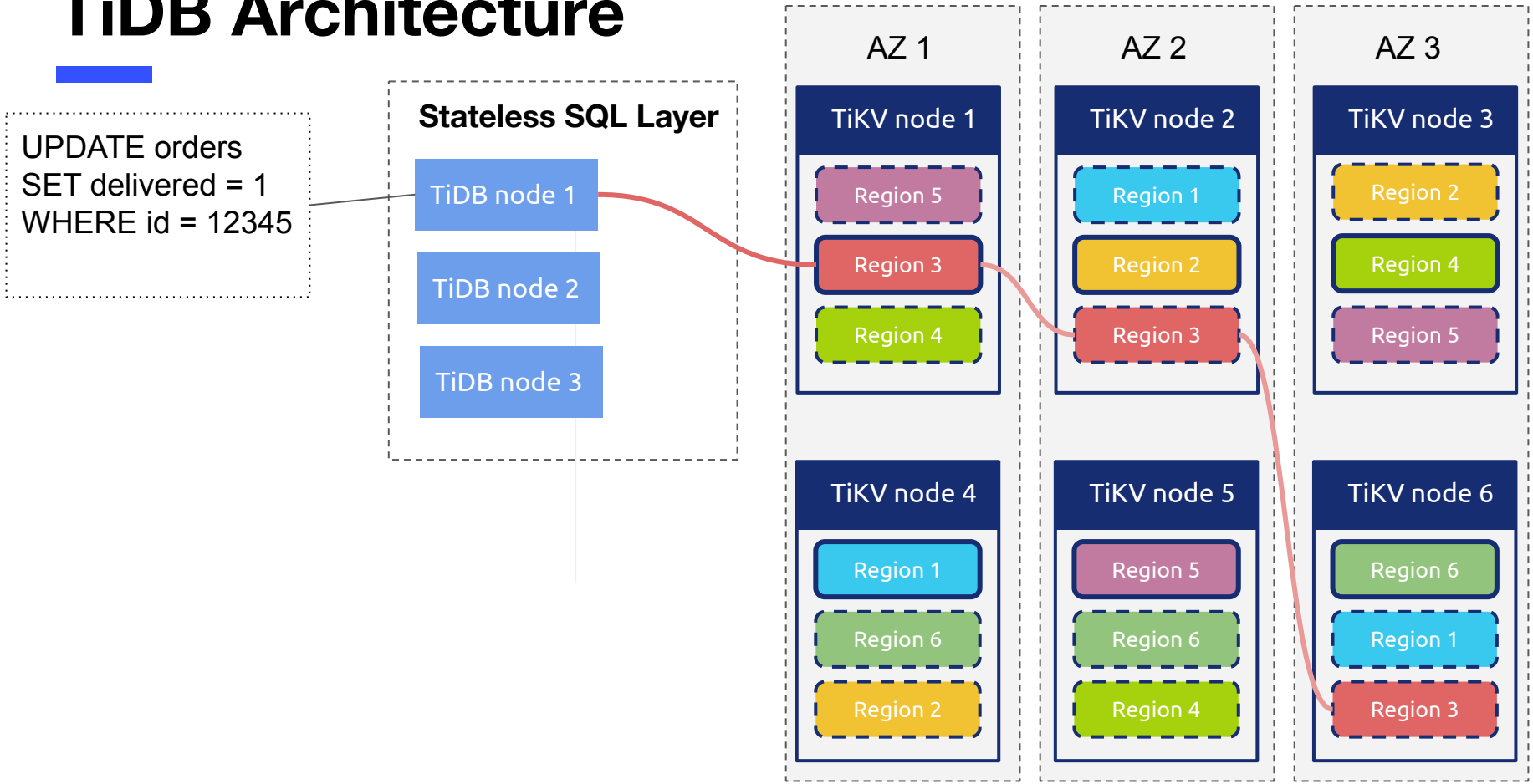


Query Execution/Distributed Computing

```
SELECT COUNT(*) FROM t WHERE id > 10 AND c2 = 'foo';
```



TiDB Architecture



Scalability with TiDB

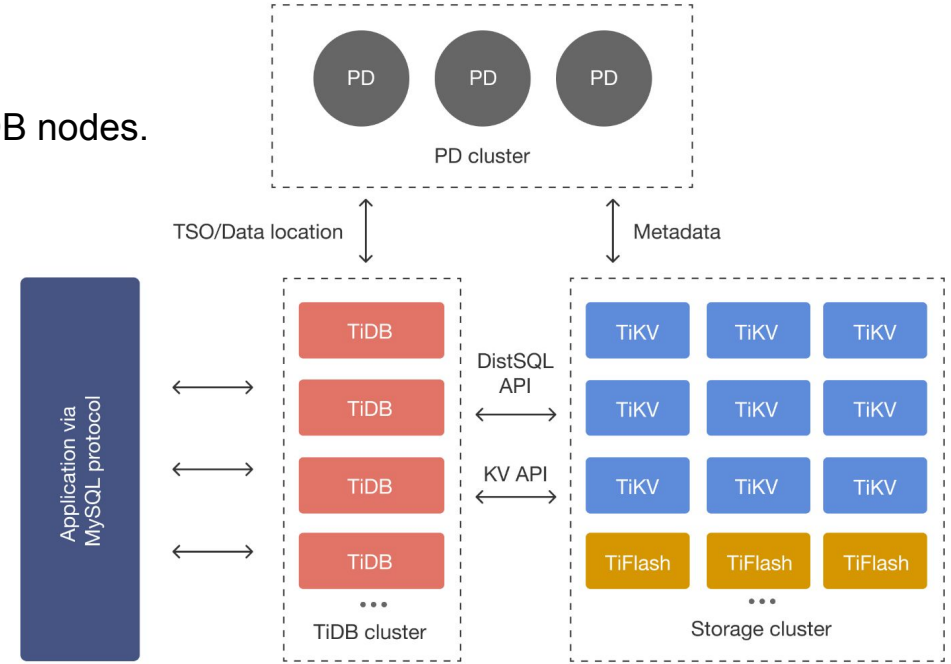
Both reads and writes can go to any of the TiDB nodes.

Scaling reads?
Add more nodes

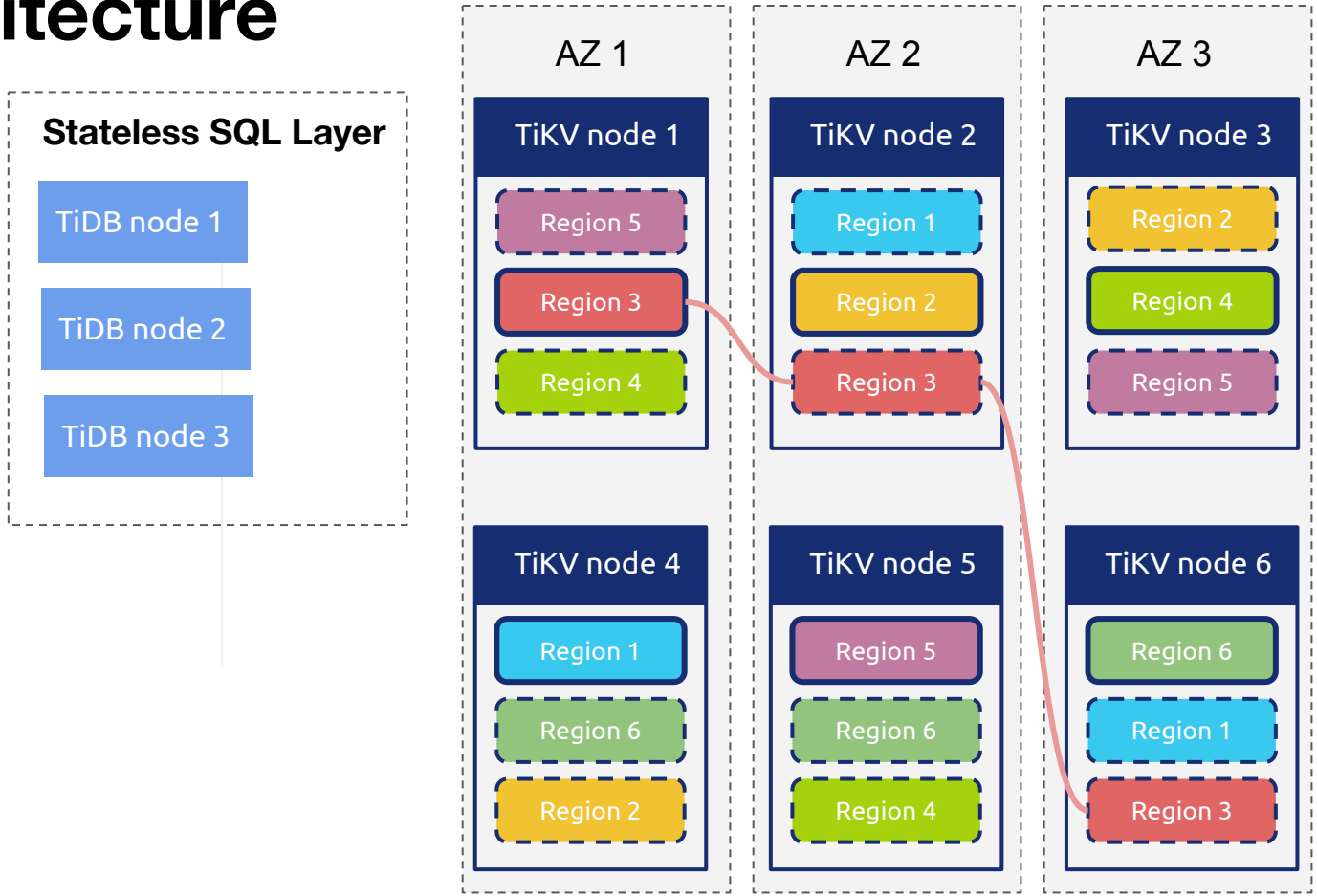
Scaling writes?
Add more nodes

Scaling data volume?
Add more nodes

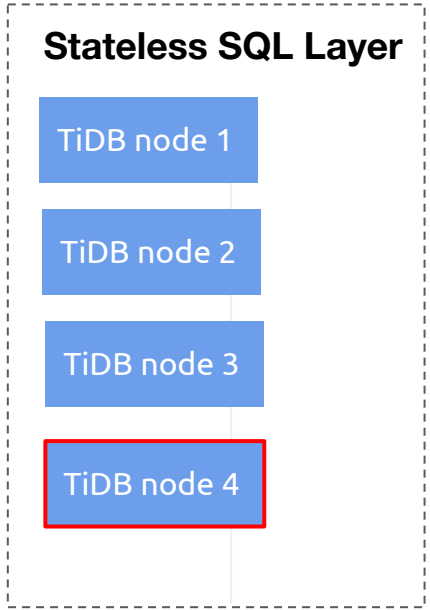
Need to scale more?
Add more nodes

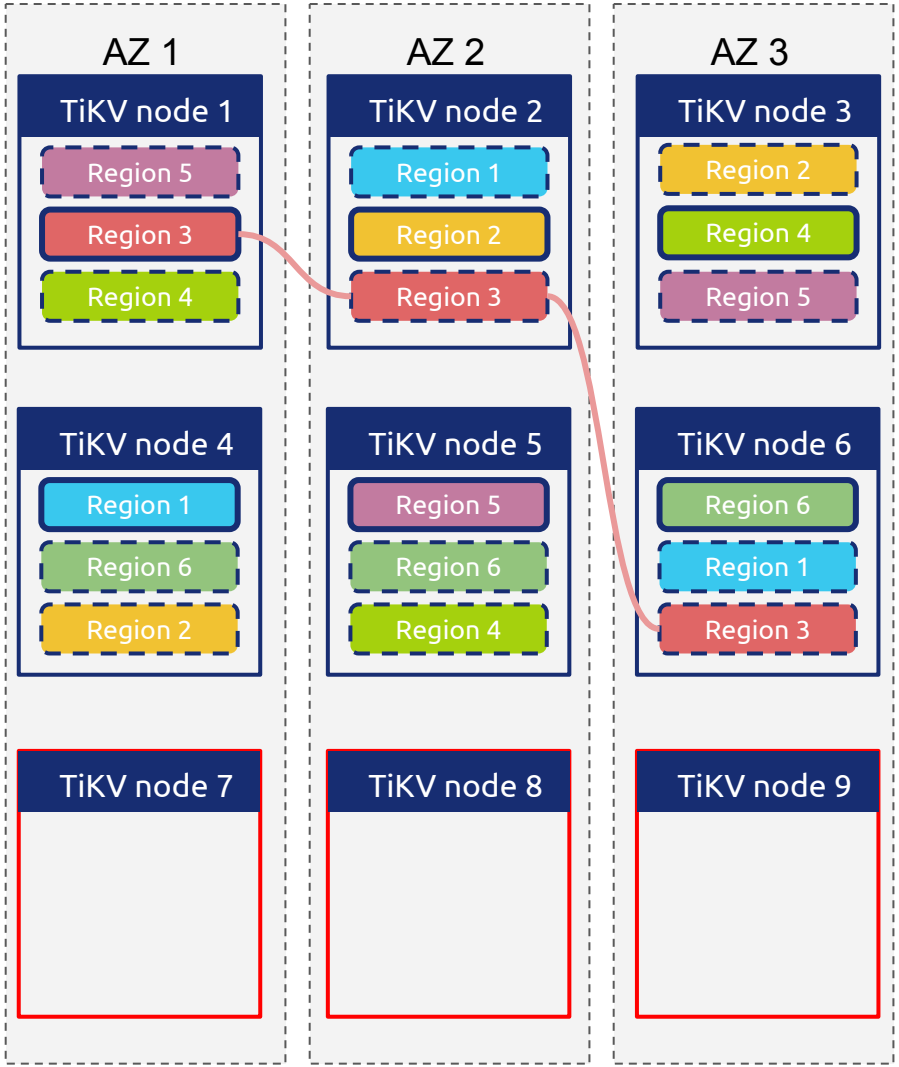


TiDB Architecture

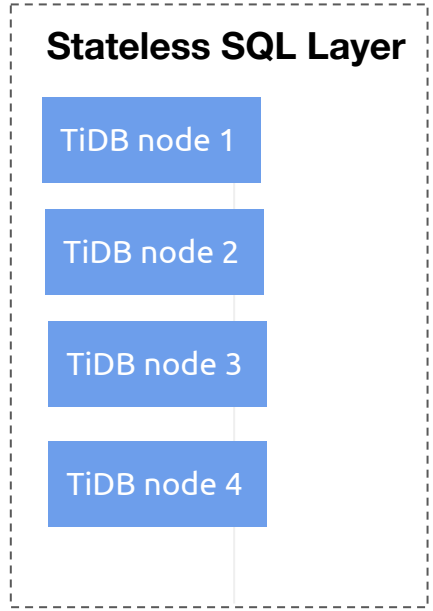


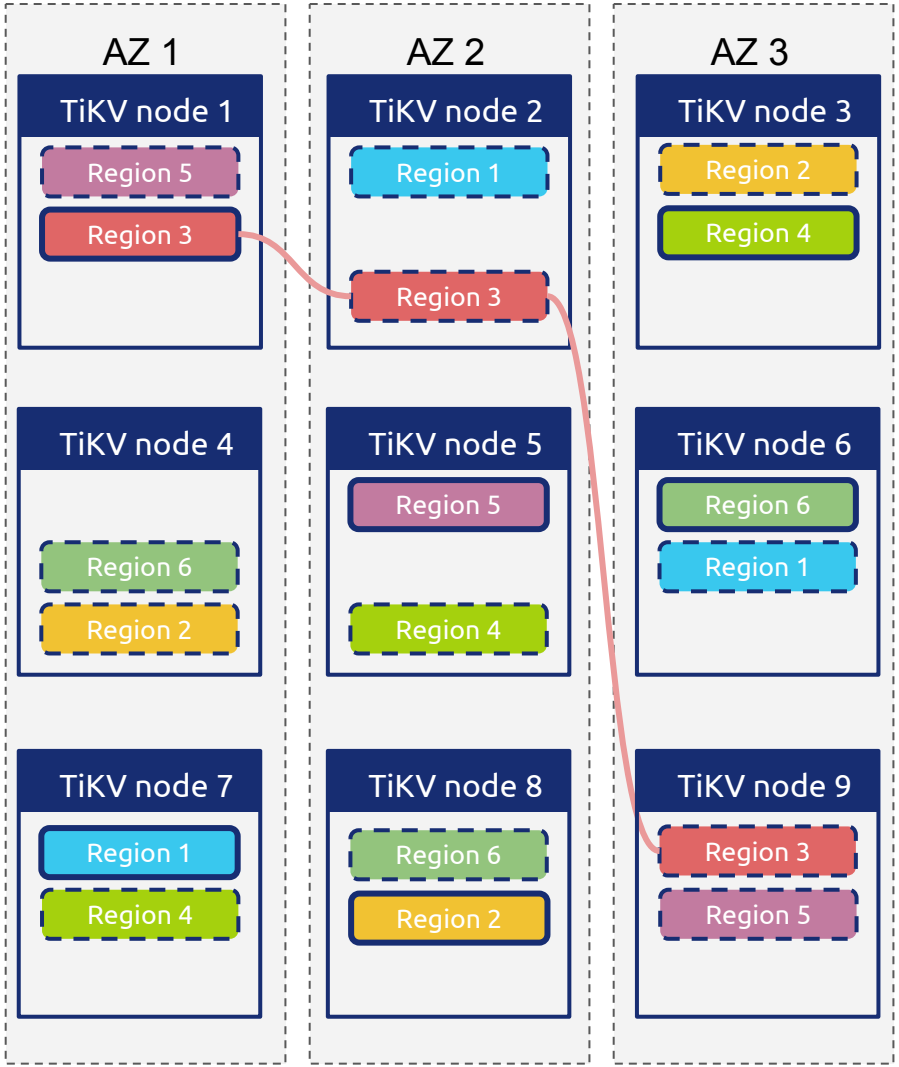
TiDB Architecture





TiDB Architecture





Lab excercises: TiKV

PingCAP

MySQL Client:

```
mysql -u root -h 127.0.0.1 -P 4000 blog

SHOW TABLE comments REGIONS;
```

Run loadGen (in a new ssh session)

```
cd demoblog/loadGen
./loadGen

SHOW TABLE comments REGIONS;
SELECT REGION_ID,START_KEY,END_KEY,TABLE_NAME,INDEX_NAME FROM
information_schema.TIKV_REGION_STATUS WHERE DB_NAME='blog';
```

Check Key Visualizer in TiDB Dashboard

```
ssh -L2379:127.0.0.1:2379 …
```

http://127.0.0.1:2379/dashboard/

Lab excercises

PingCAP

Scale out via TiUP Playground:

```
tiup playground display
tiup playground scale-out --kv 2
tiup playground display
tiup ctl:v7.2.0 pd store
curl http://127.0.0.1:2379/pd/api/v1/stores
```

SQL:

```
SELECT * FROM information_schema.TIKV_STORE_STATUS;

SELECT REGION_ID,START_KEY,END_KEY,TABLE_NAME,INDEX_NAME FROM
information_schema.TIKV_REGION_STATUS WHERE DB_NAME='blog';

SELECT * FROM information_schema.TIKV_REGION_PEERS WHERE
REGION_ID=<id from one of the regions above>;
```

```
mysql> SELECT REGION_ID,START_KEY,END_KEY,TABLE_NAME,INDEX_NAME FROM information_schema.TIKV_REGION_STATUS WHERE DB_NAME='blog';
+-----------+----------------------------------------+----------------------------------------+------------+------------+
| REGION_ID | START_KEY                              | END_KEY                                | TABLE_NAME | INDEX_NAME |
+-----------+----------------------------------------+----------------------------------------+------------+------------+
|       131 | 7480000000000000FF7200000000000000F8 | 7480000000000000FF7500000000000000F8 | posts      | NULL       |
|       133 | 7480000000000000FF7500000000000000F8 | 7480000000000000FF7700000000000000F8 | comments   | post_id    |
|       133 | 7480000000000000FF7500000000000000F8 | 7480000000000000FF7700000000000000F8 | comments   | NULL       |
|        14 | 7480000000000000FF7700000000000000F8 | 748000FFFFFFFFFFFF9000000000000000F8 | authors    | NULL       |
|       129 | 7480000000000000FF7100000000000000F8 | 7480000000000000FF7200000000000000F8 | ticker     | NULL       |
+-----------+----------------------------------------+----------------------------------------+------------+------------+
5 rows in set (0.01 sec)

mysql> SELECT * FROM information_schema.TIKV_REGION_PEERS WHERE REGION_ID=133;
+-----------+---------+----------+------------+-----------+--------+--------------+
| REGION_ID | PEER_ID | STORE_ID | IS_LEARNER | IS_LEADER | STATUS | DOWN_SECONDS |
+-----------+---------+----------+------------+-----------+--------+--------------+
|       133 |     134 |        1 |          0 |         1 | NORMAL |         NULL |
|       133 |     193 |      136 |          0 |         0 | NORMAL |         NULL |
|       133 |     269 |      135 |          0 |         0 | NORMAL |         NULL |
+-----------+---------+----------+------------+-----------+--------+--------------+
3 rows in set (0.00 sec)
```

```
mysql> SELECT REGION_ID,TABLE_ID,INDEX_NAME,TIDB_DECODE_KEY(START_KEY) 'start',TIDB_DECODE_KEY(END_KEY) 'end',APPROXIMATE_SIZE
    -> FROM information_schema.TIKV_REGION_STATUS WHERE DB_NAME='blog' AND TABLE_NAME='comments'
    -> ORDER BY INDEX_NAME,START_KEY,END_KEY;
+-----------+----------+------------+-----------------------------------------------------------+-----------------------------------------------------------------------------------------------------------------------------------+-----------------+
| REGION_ID | TABLE_ID | INDEX_NAME | start                                                     | end                                                                                                                               | APPROXIMATE_SIZE |
+-----------+----------+------------+-----------------------------------------------------------+-----------------------------------------------------------------------------------------------------------------------------------+-----------------+
|       149 |      100 | NULL       | 7480000000000000FF645F720000000000FA                      | {"id":1357949,"table_id":"100"}                                                                                                   |             116 |
|       134 |      100 | NULL       | {"id":1357949,"table_id":"100"}                           | {"table_id":101}                                                                                                                  |             131 |
|       123 |      100 | post_id    | {"table_id":100}                                          | {"index_id":1,"index_vals":{"post_id":"1"},"table_id":100}                                                                        |               1 |
|       125 |      100 | post_id    | {"index_id":1,"index_vals":{"post_id":"1"},"table_id":100} | {"index_id":1,"index_vals":{"post_id":"1"},"table_id":100}                                                                        |               1 |
|       152 |      100 | post_id    | {"index_id":1,"index_vals":{"post_id":"1"},"table_id":100} | {"index_id":1,"index_vals":{"post_id":"5"},"table_id":100}                                                                        |              91 |
|       137 |      100 | post_id    | {"index_id":1,"index_vals":{"post_id":"5"},"table_id":100} | 7480000000000000FF645F698000000000FF0000020003800000FF0000000001000000FC                                                          |              64 |
+-----------+----------+------------+-----------------------------------------------------------+-----------------------------------------------------------------------------------------------------------------------------------+-----------------+
6 rows in set, 2 warnings (0.00 sec)
```

- TIDB_DECODE_KEY() to decode keys

Lab excercises

PingCAP

Add yet another TiKV node, so we safely can remove one:

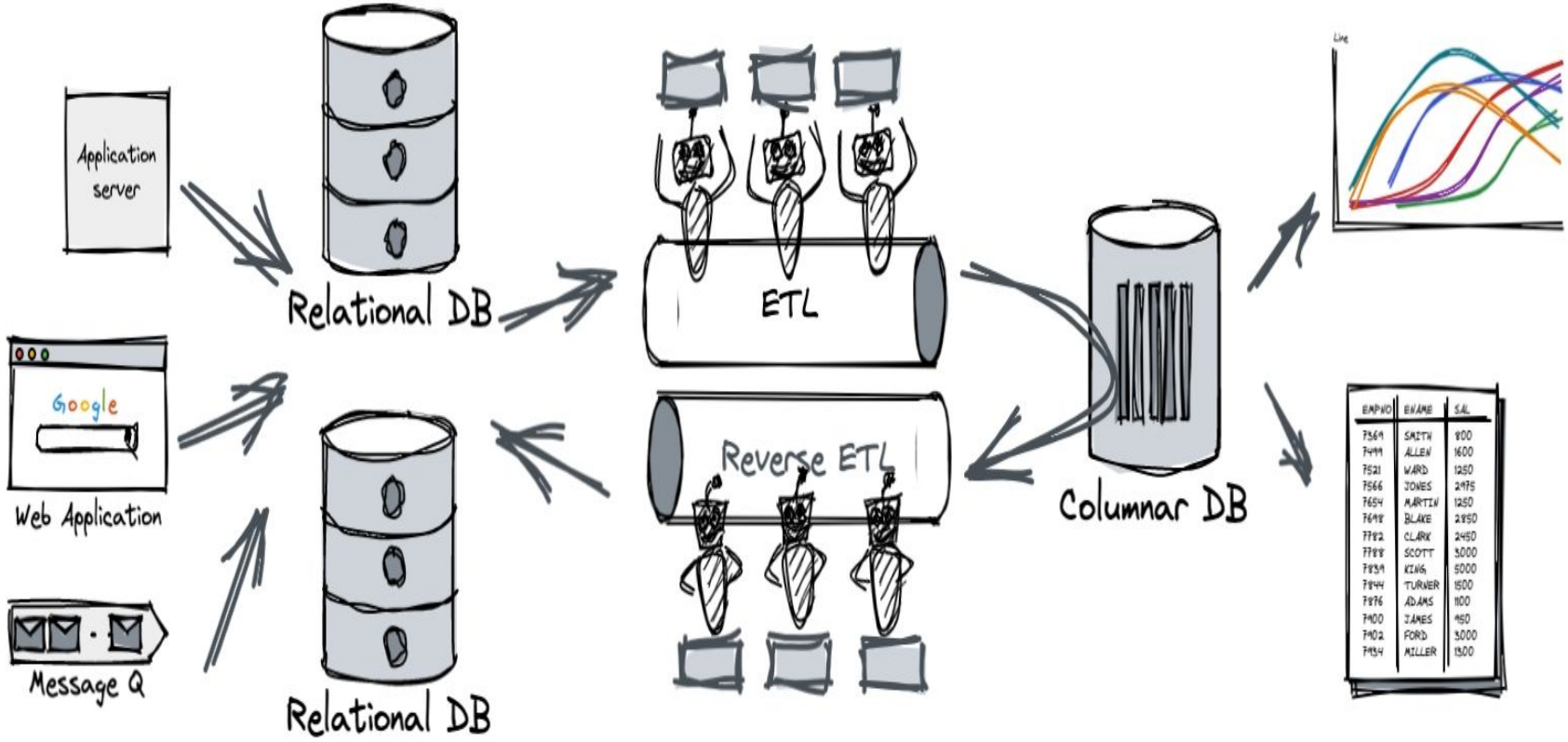
```
tiup playground scale-out --kv 1
```

Scale in via TiUP Playground:

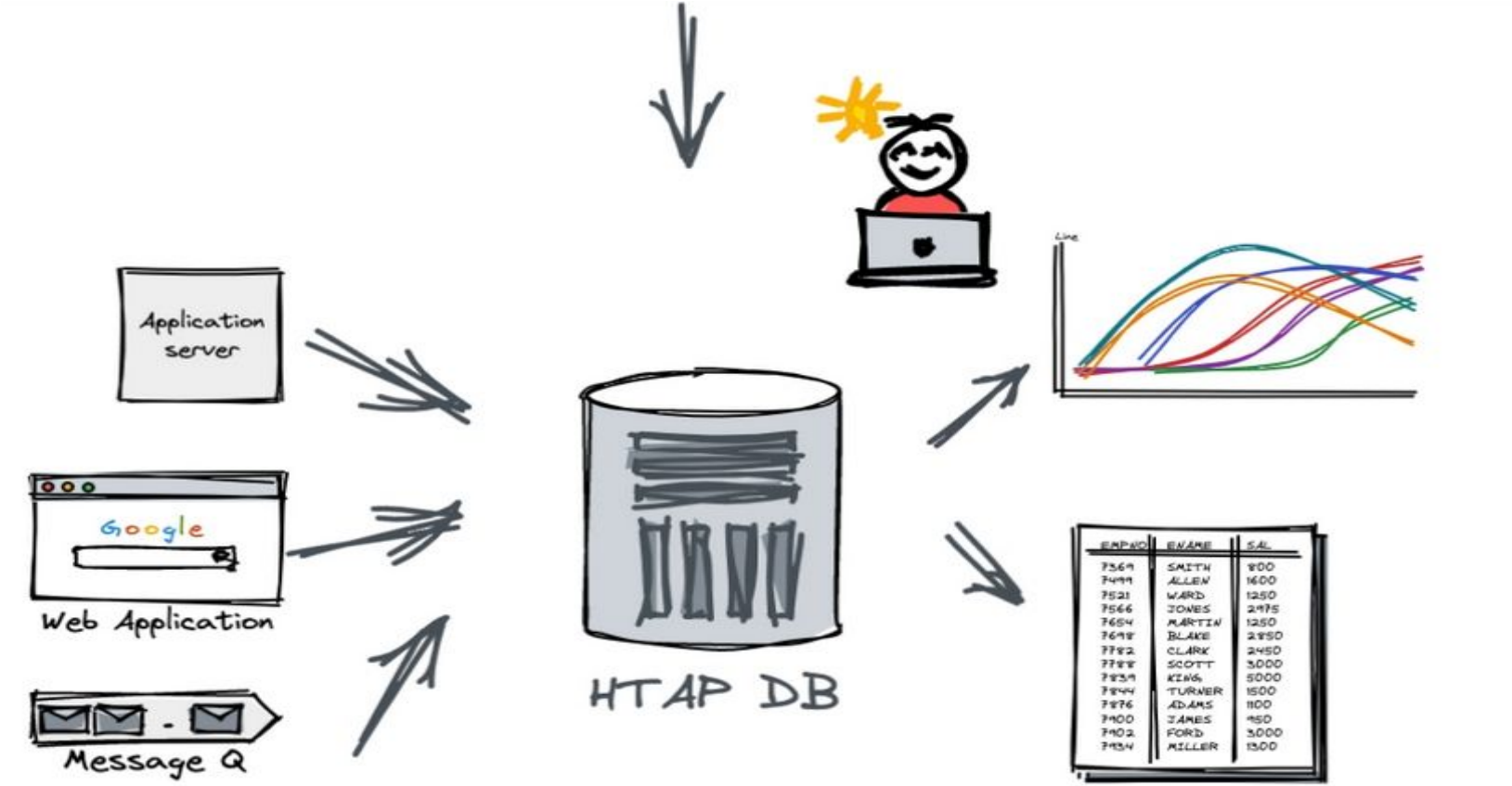
```
tiup playground display
tiup playground scale-in --pid $pid
```

Scaling in TiDB nodes will be quick. TiKV might need to move data.

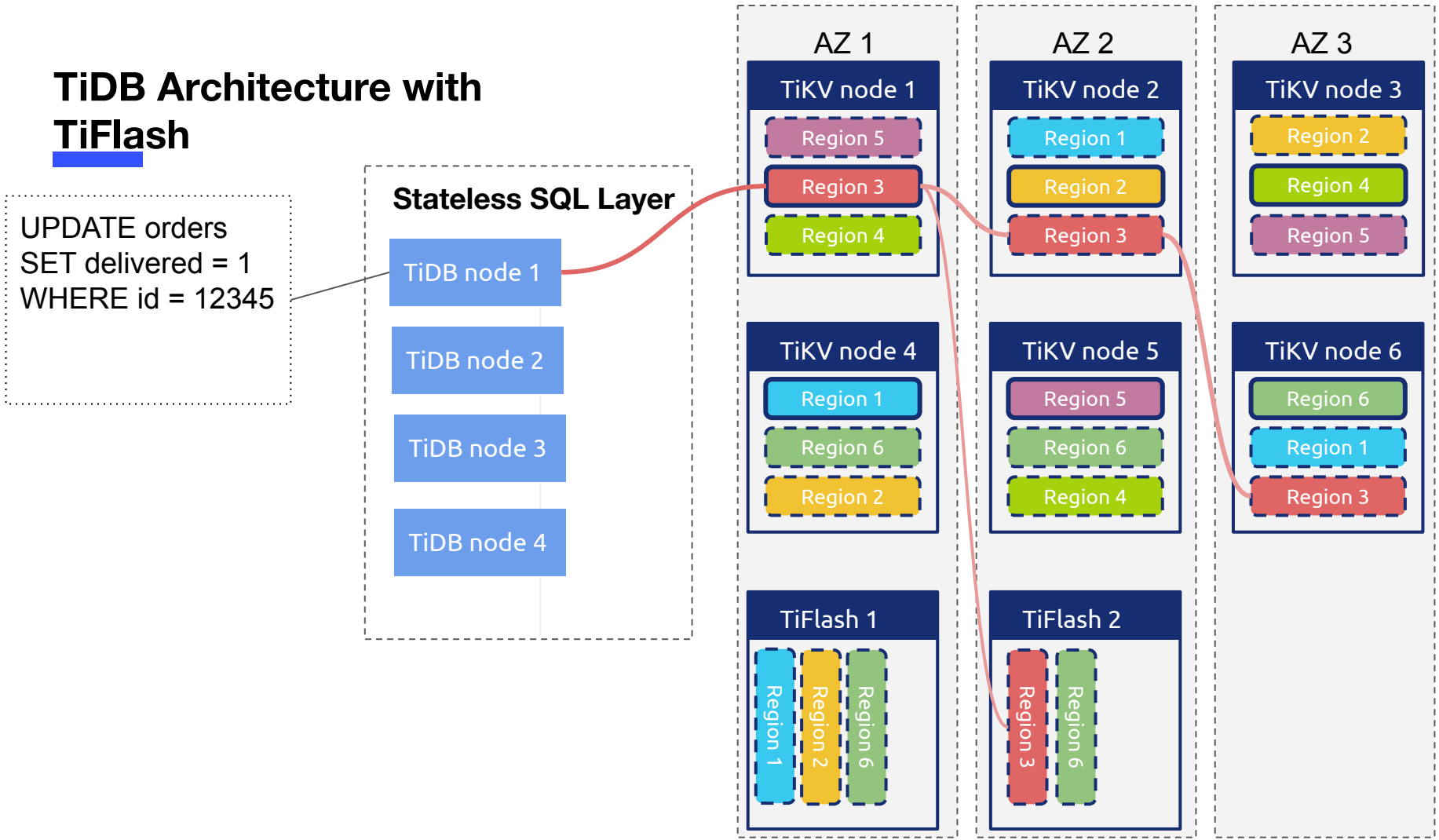
Anyone recognizes similar architecture?



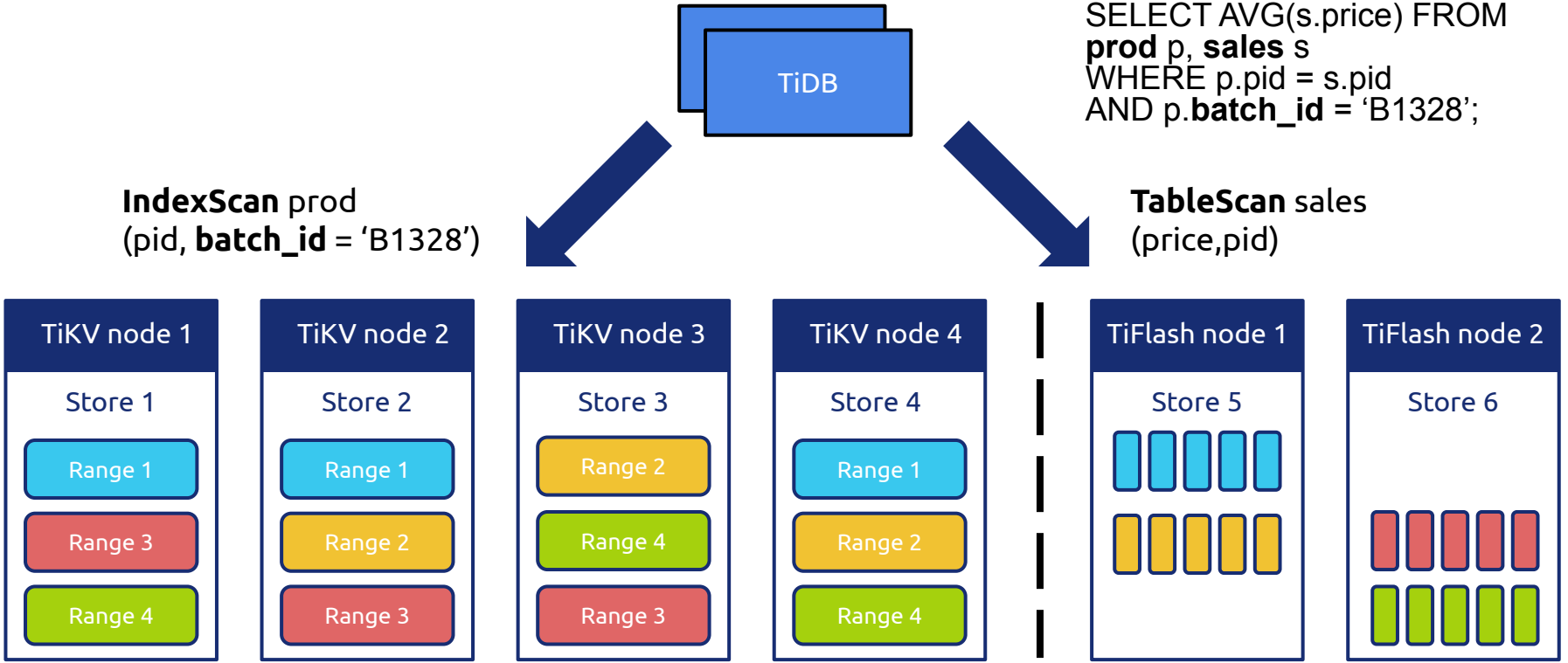
What if it could be simplified to this?



TiDB Architecture with TiFlash



The optimizer can pick the best store for each case



Lab excercises: TiFlash

PingCAP

Setup TiFlash replicas

```
ALTER TABLE posts SET TIFLASH REPLICA 1;
ALTER TABLE comments SET TIFLASH REPLICA 1;

EXPLAIN SELECT p.title, SUM(c.likes) FROM posts p LEFT JOIN comments c ON
c.post_id=p.id GROUP BY p.id;
```

Run loadGen for a bit and run explain again.
Try `EXPLAIN ANALYZE <query>`

Try engine isolation:

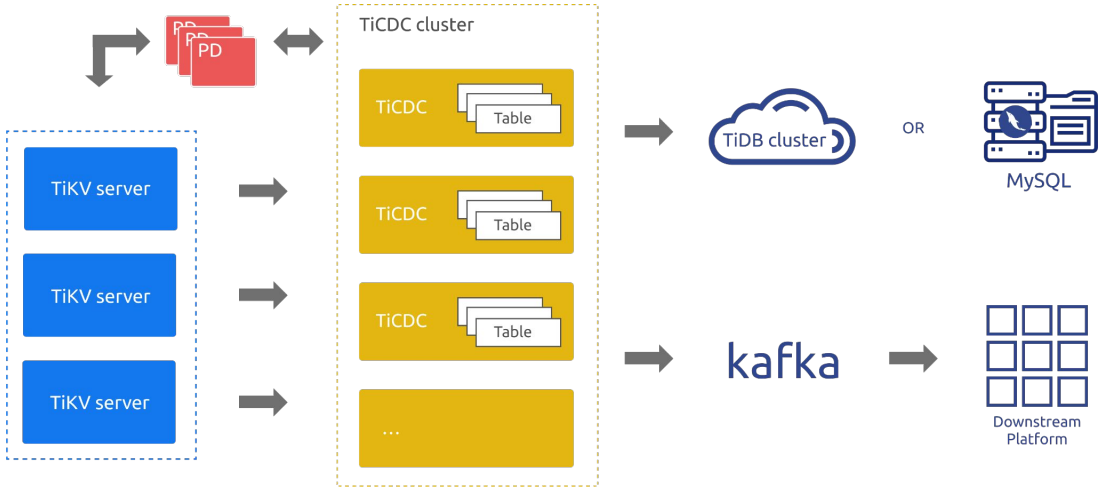
```
SET SESSION tidb_isolation_read_engines = "tikv";
EXPLAIN …
SET SESSION tidb_isolation_read_engines = "tiflash,tikv";
```

Only adding TiFlash for comments allows joins between comments and posts. This combines TiKV and TiFlash

Batteries included: TiCDC

TiCDC

- Change Data Capture
- Send events to Kafka, MySQL or another TiDB Cluster.
- Also high available



Lab excercises: TiCDC

PingCAP

Try ticketStat:

- Setup Kafka
- Setup TiCDC changefeed to kafka
- Run tickerStat (consumes kafka)

cd demoblog/tickerStat
cat README.md

Compatibility

PingCAP

TiDB is compatible with the MySQL protocol and syntax of MySQL 5.7 and MySQL 8.x. There are a few differences.

Row size is limited to 120 MiB.

TiDB differences:

- AUTO_INCREMENT, AUTO_RANDOM (but there is a MySQL compatible mode)
- Explain output is different
- No triggers, stored procedures, etc: Use TiCDC changefeed instead.
- No fulltext index
- No geospatial support

https://docs.pingcap.com/tidb/stable/mysql-compatibility

Lab excercises: Compatibility

PingCAP

Try queries:

```
SELECT p.title,p.likes,SUM(c.likes) 'liked comments'
FROM posts p
LEFT JOIN comments c ON c.post_id=p.id GROUP BY p.id;
```

Try some tools like MySQL Workbench, DBeaver, etc. This might need portforwarding of port 4000: `ssh -L4000:127.0.0.1:4000 …`

https://docs.pingcap.com/tidb/stable/mysql-compatibility

Most ORM's just work with MySQL compatibility. There are some that implement TiDB specific features and settings (e.g. default port as 4000)

Lab excercises: Online DDL

PingCAP

Drop and Add INDEX

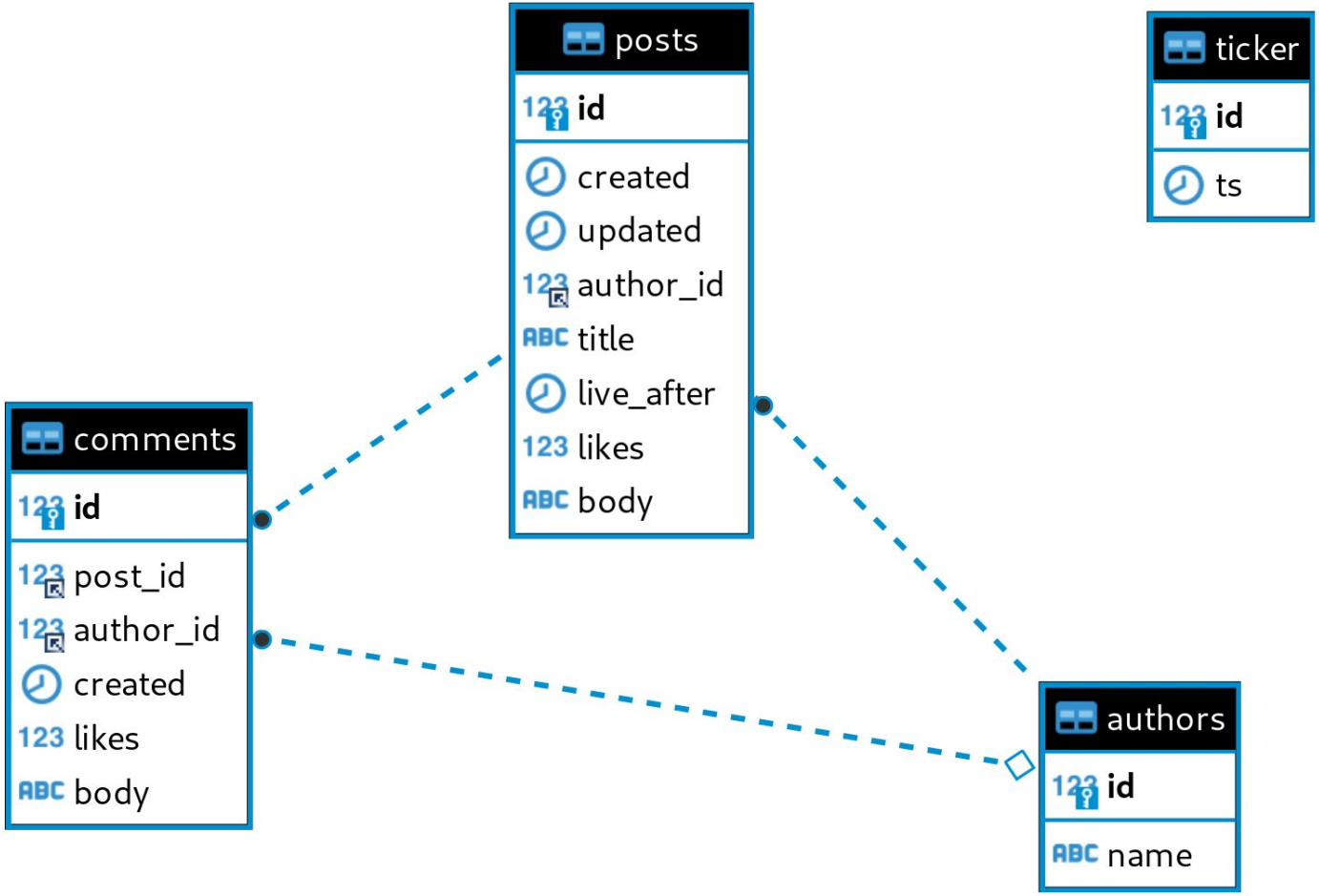
```
ALTER TABLE comments DROP INDEX post_id;
ALTER TABLE comments ADD INDEX post_id (post_id);
```

Foreign Keys

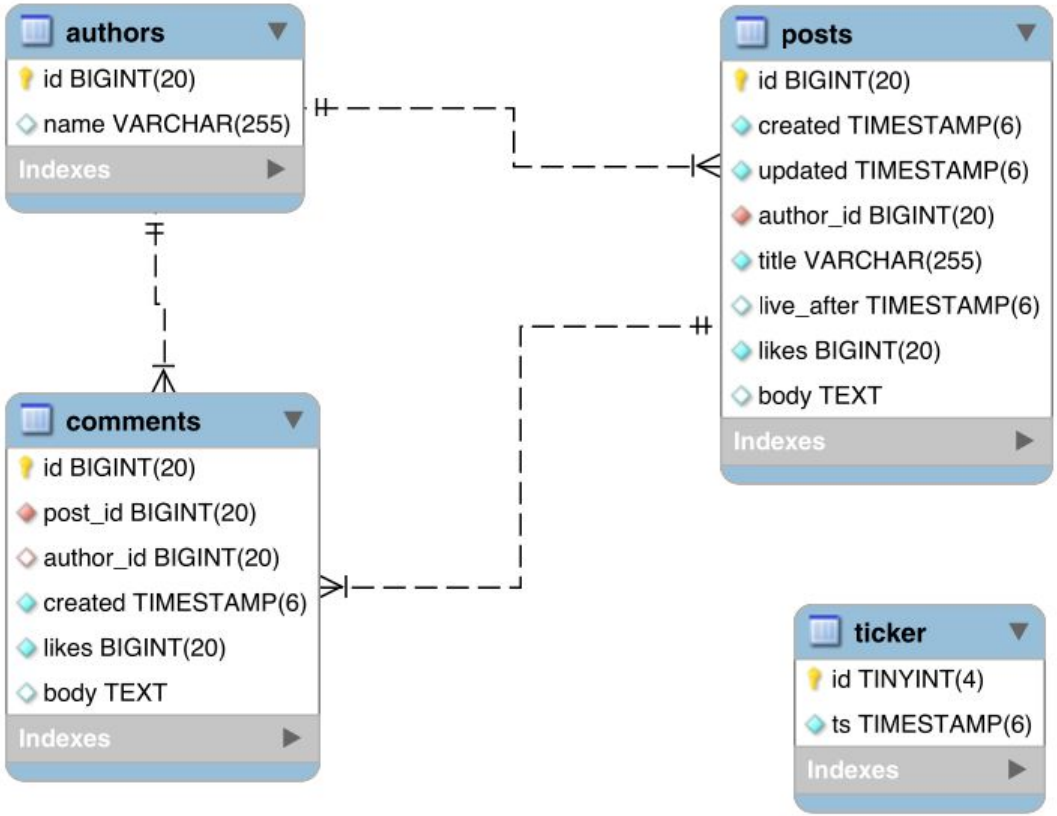
```
ALTER TABLE comments
    ADD CONSTRAINT fk_postid  FOREIGN KEY (post_id) REFERENCES posts(id),
    ADD CONSTRAINT fk_authorid FOREIGN KEY (author_id) REFERENCES authors(id);

ALTER TABLE posts
    ADD CONSTRAINT fk_authorid FOREIGN KEY (author_id) REFERENCES authors(id);

-- This should fail as there is no author with that ID.
UPDATE posts SET author_id=2 WHERE id=1;
```


posts
id
created
updated
author_id
title
live_after
likes
body
ticker
id
ts
comments
id
post_id
author_id
created
likes
body
authors
id
name

DBeaver


authors
id BIGINT(20)
name VARCHAR(255)
Indexes
posts
id BIGINT(20)
created TIMESTAMP(6)
updated TIMESTAMP(6)
author_id BIGINT(20)
title VARCHAR(255)
live_after TIMESTAMP(6)
likes BIGINT(20)
body TEXT
Indexes
comments
id BIGINT(20)
post_id BIGINT(20)
author_id BIGINT(20)
created TIMESTAMP(6)
likes BIGINT(20)
body TEXT
Indexes
ticker
id TINYINT(4)
ts TIMESTAMP(6)
Indexes

MySQL™

Time travel with TiDB

- Data is never actually removed directly, it is garbage collected instead.
- This allows you to see any version between now and the point of the GC.
- Use `SELECT ... FROM ... AS OF TIMESTAMP`
- Works for both table data and table structure
- Dumpling can dump data of a specific time
- Default GC life time is 10 minutes
- `FLASHBACK` statement can recover the last version of a table, database or a full cluster

Lab excercises

PingCAP

- Time travel: AS OF…

```
SELECT post_id,COUNT(*),SUM(likes)
FROM comments AS OF TIMESTAMP '2023-07-28 15:25:11'
GROUP BY post_id;
```

- FLASHBACK TABLE & BR

This was in dev environment right?!

```
DROP TABLE comments;
```

Oh no, prod :o (loadGen probably crashed…)

```
FLASHBACK TABLE comments;
```

Restart the loadGen and it should work again \o/

Lab excercises

PingCAP

Now check the various dashboards:

- http://127.0.0.1:2379/dashboard PD dashboard (root/)
- http://127.0.0.1:3000/ Grafana (admin/admin)

Backup and Restore

PingCAP

BR: Backup and Restore
Every TiKV node writes it's own backup to S3 or other shared storage. Similar for restores. This makes backups scalable.

Lab excercises: BR

PingCAP

Create a backup

```
tiup br backup full -s file:///tmp/b1
```

Decode backup metadata

```
tiup br validate decode -s /tmp/b1/
```

Get the TSO of the backup

```
jq .end_version /tmp/b1/backupmeta.json
```

Get the timestamp for the TSO

```
tiup ctl:v7.2.0 pd tso $tso
```

Simlar, via SQL: `SELECT TIDB_PARSE_TSO($tso)`

Lab excercises: BR & TSO

PingCAP

```
WITH tso AS (
    SELECT 443165077170552833 v
)
SELECT
    v 'TSO raw',
    (v >> 18)/1000 'TSO unix timestamp',
    FROM_UNIXTIME((v >> 18) / 1000) 'TSO timestamp',
    TIDB_PARSE_TSO(v),
    v - ((v >> 18) << 18) 'TSO logical part'
FROM tso\G
```

443165077170552833 in binary:

```
00000110001001100111000001001010001010011001111 00000000000000001
<            46 bits physical timestamp        >< 18 bit logical
>
```

Dumpling and Lightning

PingCAP

Dumpling does parallel export of data from TiDB or MySQL

Lightning does:

- Data loading directly into TiKV (ingesting SST files, "local" backend)
- Data loading via TiDB (via SQL, "tidb" backend)

By default filenames are used as metadata, just like MyDumper and MyLoader.

Files can be local or on S3. (Not on AWS? use S3 compatible storage like MinIO)

TiDB Cloud has a webinterface for imporing data from S3.

Lab excercises: Dumpling

PingCAP

```
tiup dumpling --database blog --filetype csv -o /tmp/dump
find /tmp/dump
```

Lab excercises: Lightning

PingCAP

Create data:

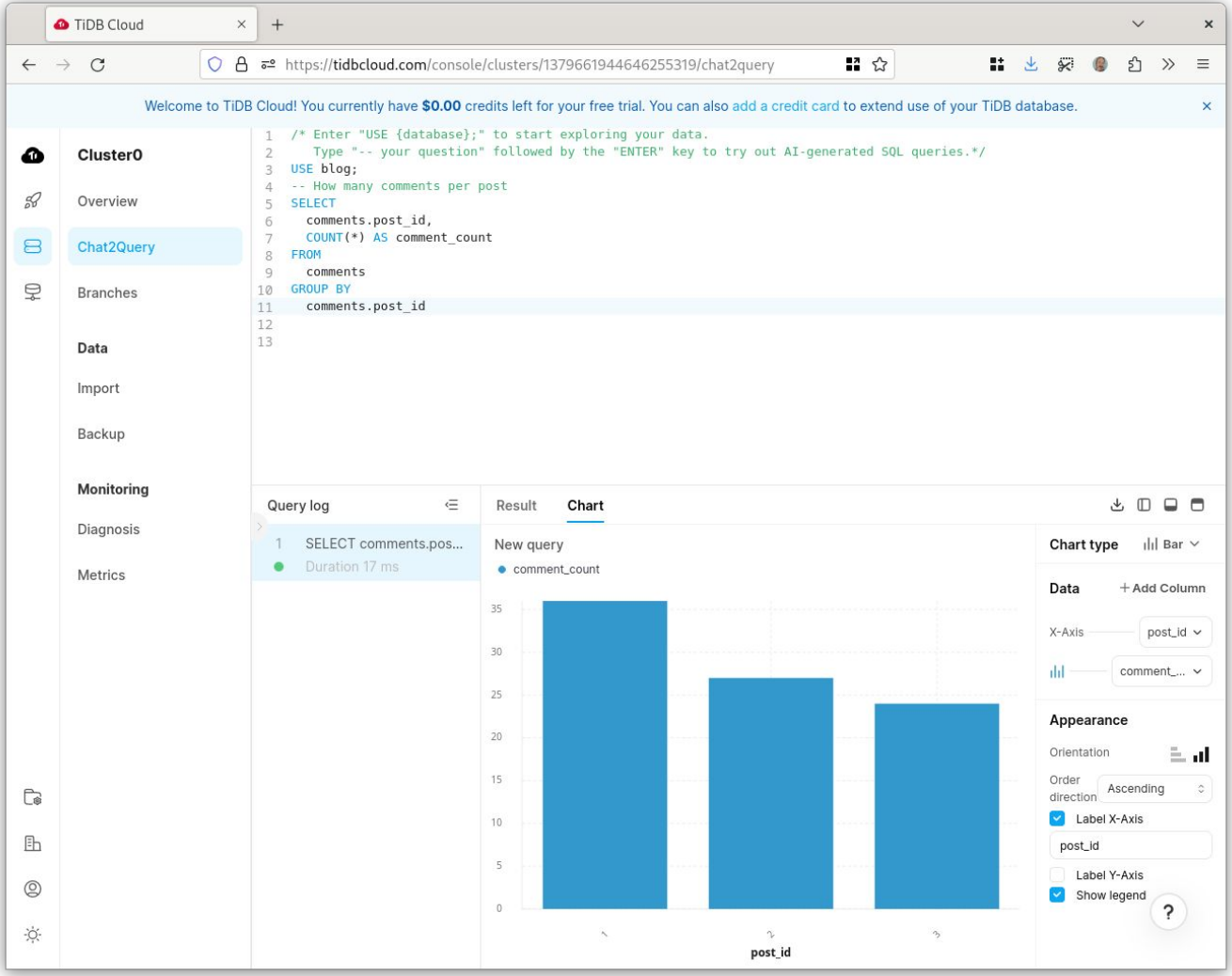
```
mkdir /tmp/authors
cat > /tmp/authors/blog.authors.000000000.csv
"id","name"
2,"Jane Doe"
```

Import data:

```
tiup tidb-lightning -d /tmp/authors -backend tidb -tidb-port
4000
```

Check via SQL:

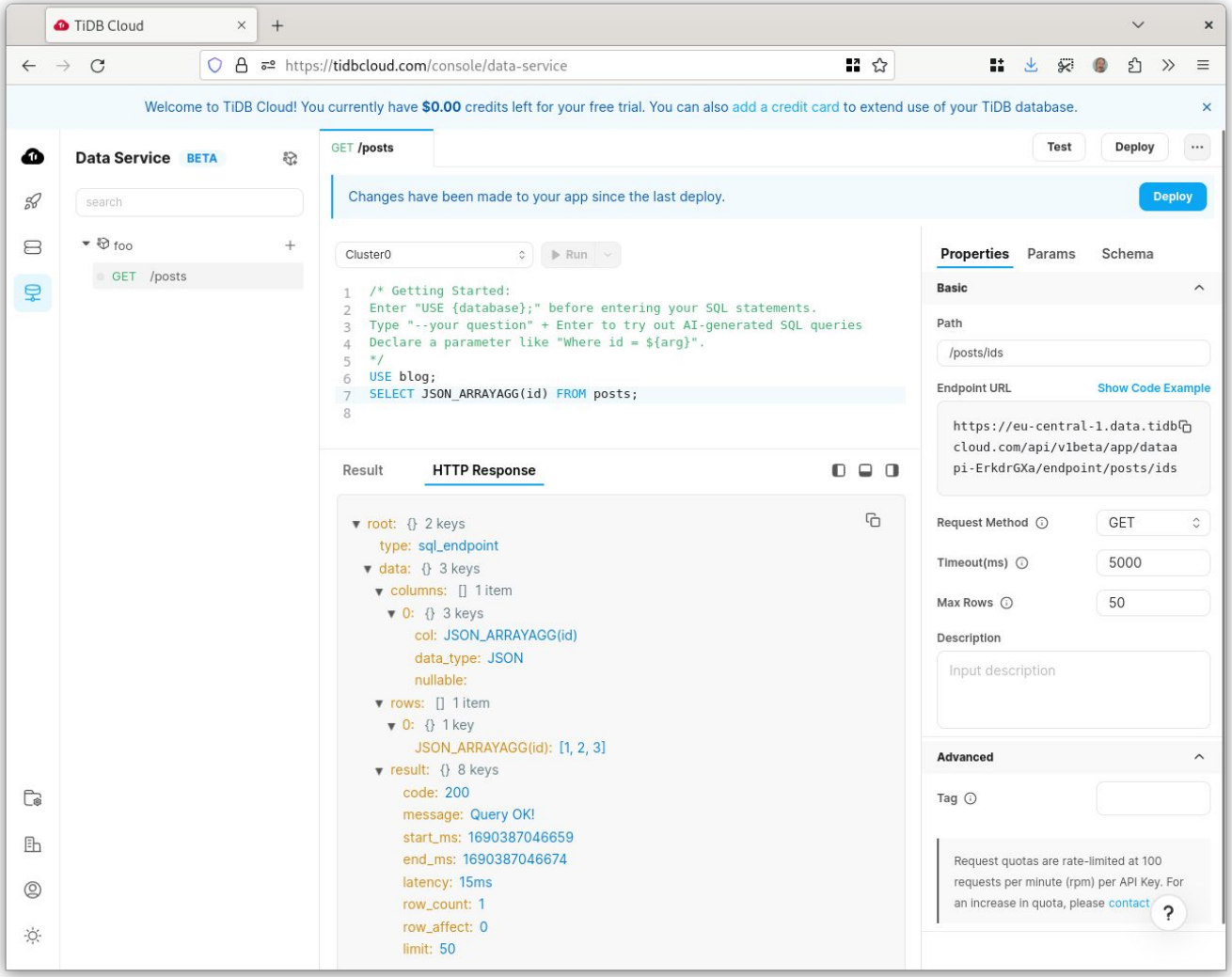
```
TABLE authors;
```



PingCAP

Chat2Query:

- AI Generated SQL
- Charts

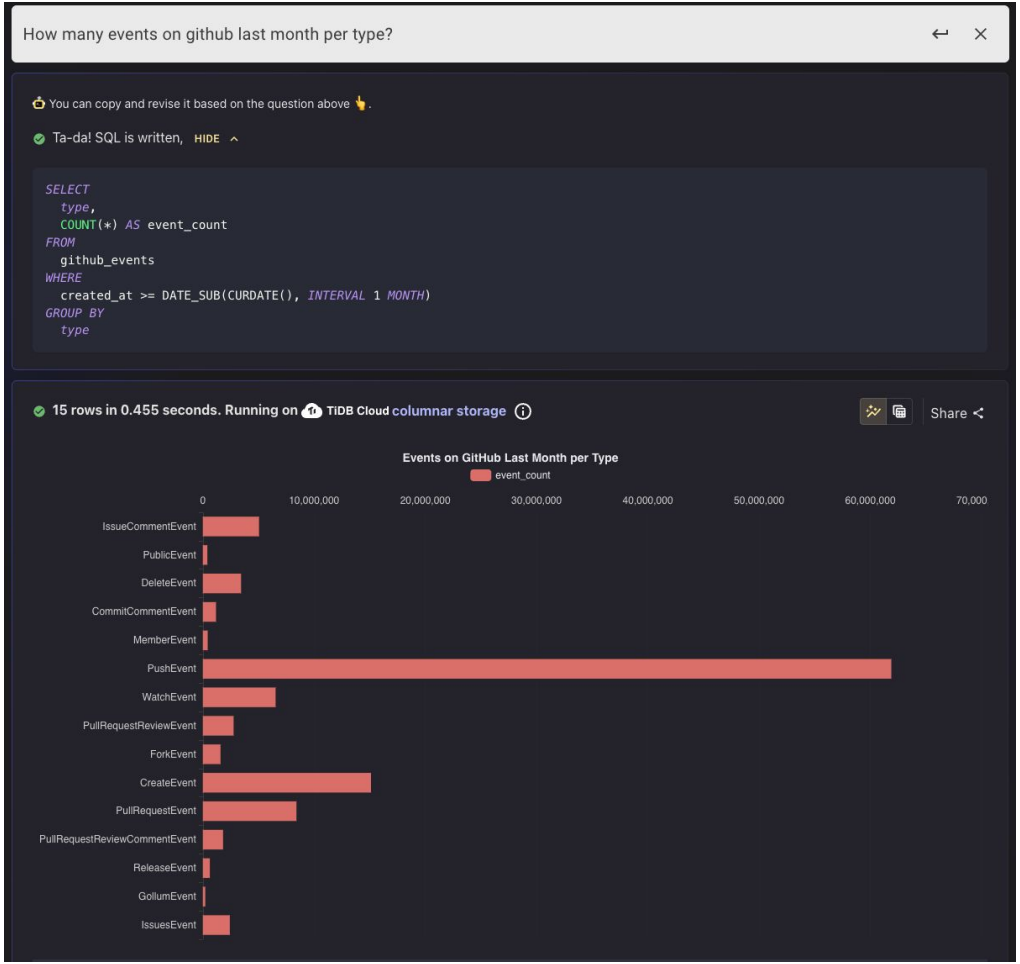


PingCAP

Data Service: HTTP REST interface based on SQL queries

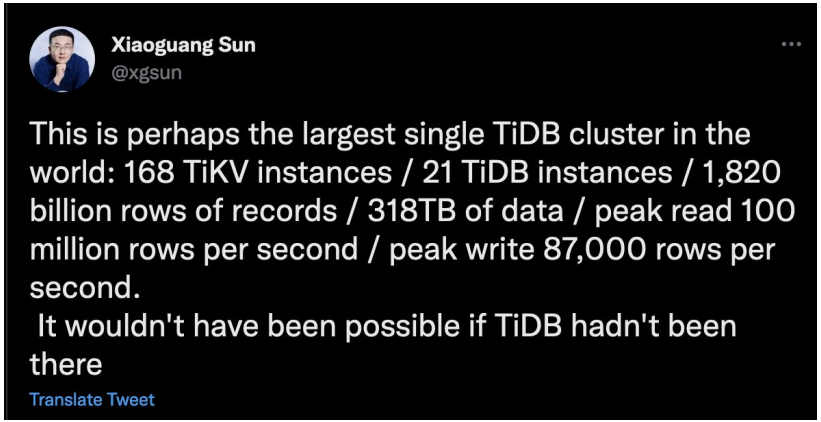
OSS Insight

- https://ossinsight.io
- Built on TiDB.
- 6 Billion rows of GitHub events (Push, Pull, Watch/Star, Issue etc.)
- Allows you to look at the SQL, including explain plans.
- No ETL.
- Chat2Query

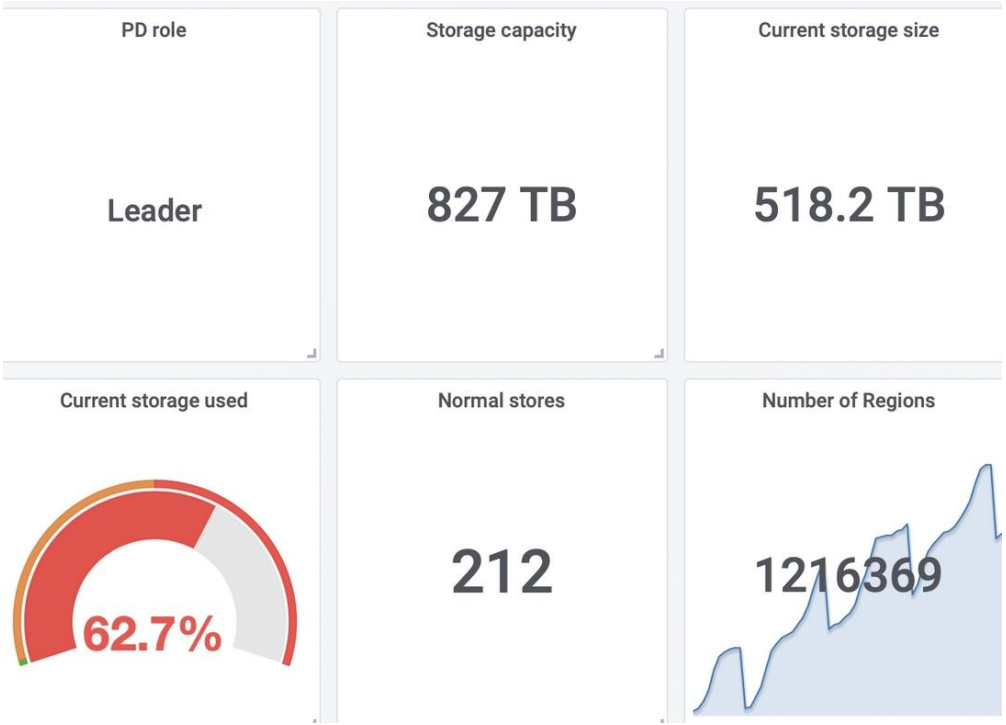


A real-world TiDB cluster in production

Tweets from Xiaoguang Sun, Head of Infrastructure @ Zhihu



Only two engineers are maintaining TiDB at Zhihu →



Dashboard Screenshot of the biggest TiDB cluster running in Zhihu.com

Conclusions and next steps

PingCAP

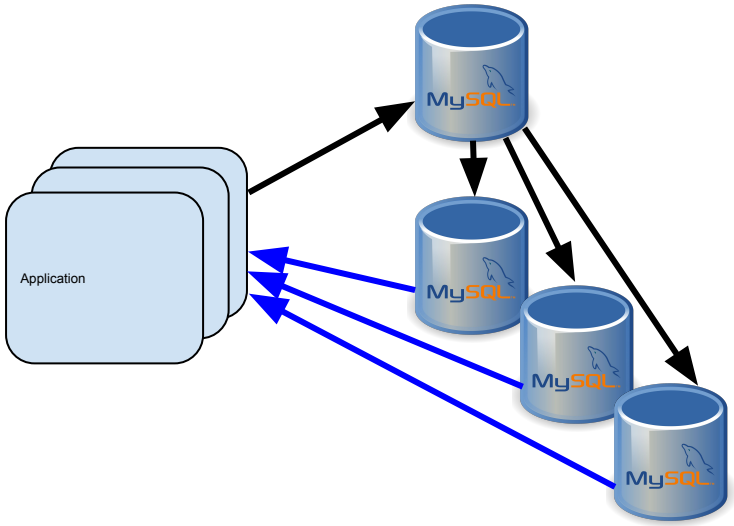
- TiDB is MySQL compatible
- TiDB has fully automatic scaling
- TiDB has high availability builtin
- TiDB is developer friendly
- TiDB allows for online DDL
- TiDB can be deployed
  - on-prem with TiUP Cluster
  - on-prem via Kubernetes with TiDB Operator
  - via TiDB Cloud on AWS or GCP

- Ask questions via Slack (via QR code)
- Give TiDB a try for your application

High availability with MySQL

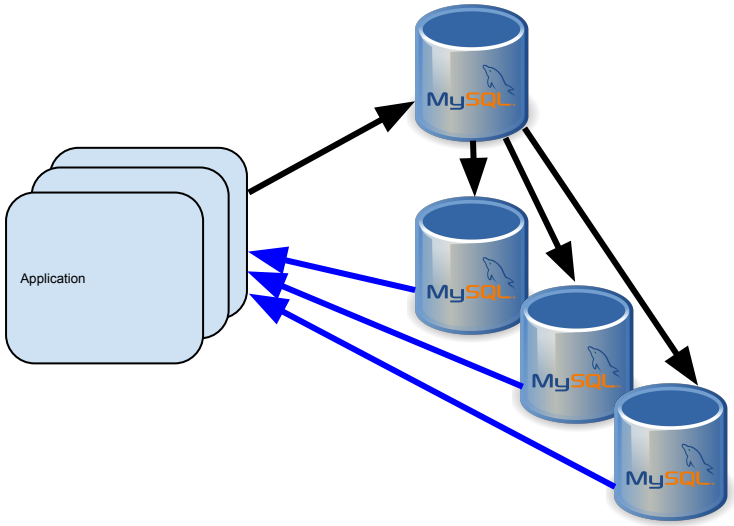
An often-used setup with MySQL is to have a primary and multiple replicas.
Then if the primary fails you promote a replica.

Replica promotion is not automated.
Is your replica up to date?
Use a load-balancer for service discovery?
How to fail back once the primary is back?
There is no leadership election.



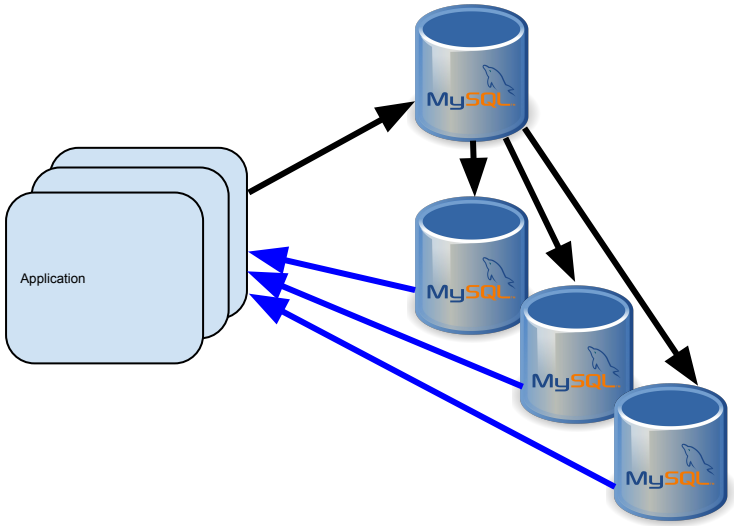
Scalability with MySQL

With MySQL all writes goes first to the primary, then replicated to all others.
All nodes store a complete copy of the data and applies all the changes.



Scalability with MySQL

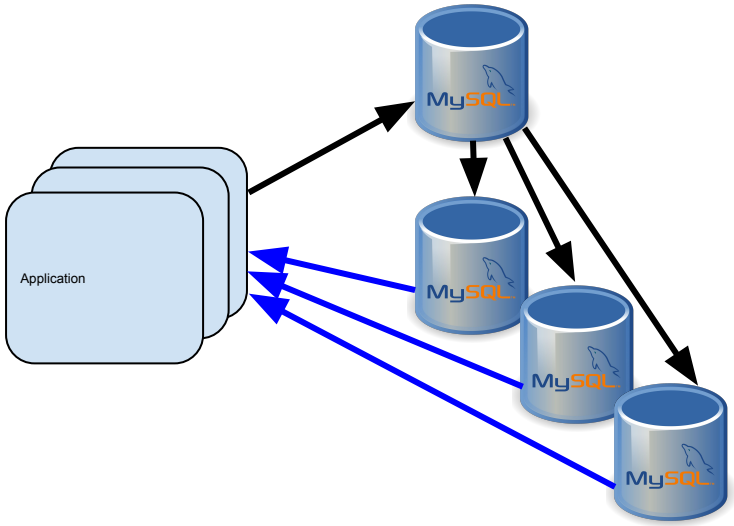
Scaling reads?
    Add more replicas



Scalability with MySQL

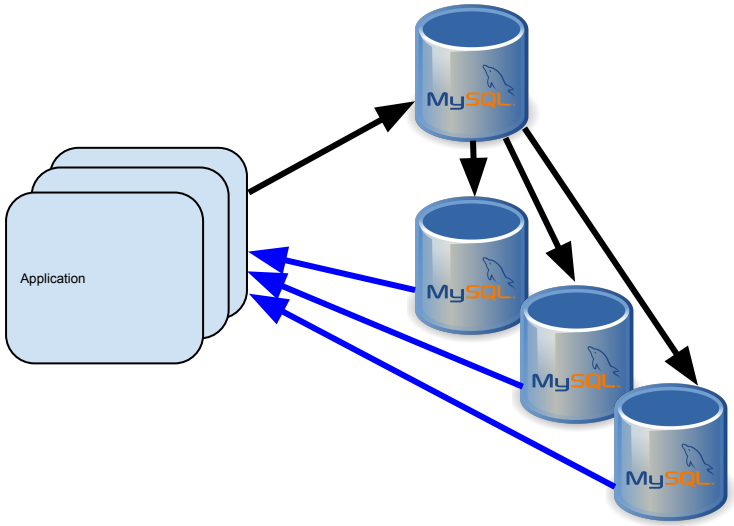
Scaling writes?

Replace the primary with a bigger machine



Scalability with MySQL

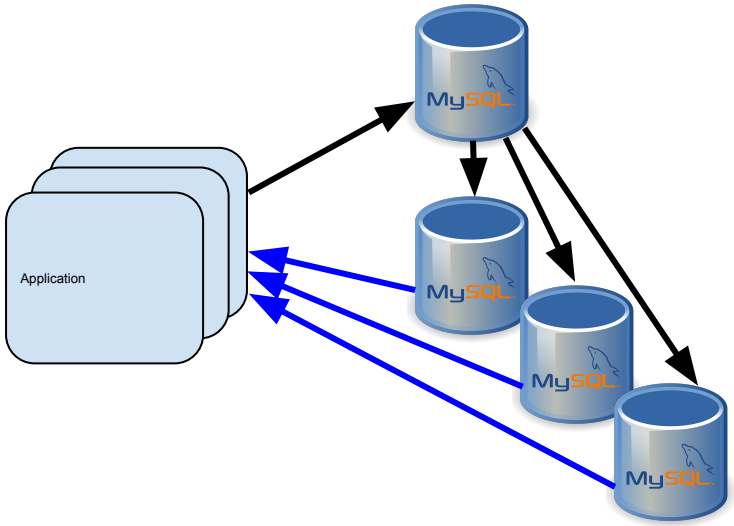
Scaling data volume?
Add bigger disks



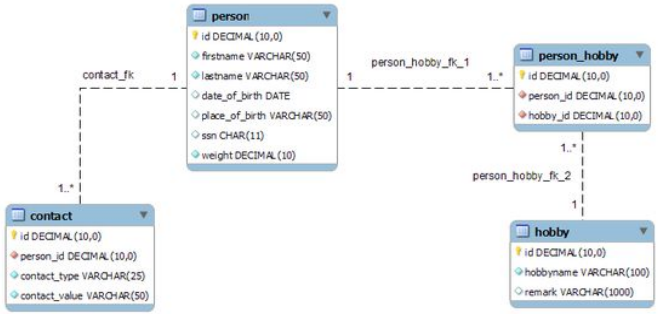
Scalability with MySQL

Need to scale more?

Shard on the application side



Data organization within TiDB



Tables (Rows with columns)

tidb-server

Dictionary order

TiKV	
t1_r1	{col1, col2, col3 ...}
t1_r2	{col1, col2, col3 ...}
...	...
t2_r1	{col1, col2 ...}
...	...

Key-value pairs within TiKV ordered by key

Data organization within TiDB

- For a row in a Table, row data is encoded in key-value pairs like this:

t<<tableID>>_r<<rowID>> => [col1, col2, col3, col4]

- For unique secondary index:

t<<tableID>>_i<<indexID>>_indexedColumnsValue => rowID

- For non-unique secondary index:

t<<tableID>>_i<<indexID>>_indexedColumnsValue_rowID => nil

Data organization within TiDB

```
CREATE TABLE user_table (
        id INT,
        name VARCHAR(64),
        email VARCHAR(1024),
        PRIMARY KEY(id)
);
```

user_table		
1	John	john@example.com
2	Jane	jane@example.com
...	...	...

Within TiKV:

t101_r1 => [John,john@example.com]
t101_r2 => [Jane,jane@example.com]
t101_r… => ...

Data organization within TiDB

```
CREATE TABLE user_table (
        id INT,
        name VARCHAR(64),
        email VARCHAR(1024),
        KEY (name),
        PRIMARY KEY(id)
);
```

user_table		
1	John	john@example.com
2	Jane	jane@example.com
...	...	...

Within TiKV:

t101_r1 => [John,john@example.com]
t101_r2 => [Jane,jane@example.com]
t101_r… => ...
...
t101_i1_Jane_2 => nil
t101_i1_John_1 => nil
t101_i1_… => nil

Table Partitioning

PingCAP

- Depends on use case, normally TiDB will already improve performance and scalability by distributing the data (can be seen as transparent range sharding/partitioning)
- CREATE TABLE t (...) PARTITION BY RANGE (col) (partition p1 values less than (1000000),...)
- Good use cases are for fast delete of large amount of data, instead of a big transactional
  DELETE FROM t WHERE col < 1000000
  do a DDL like
  ALTER TABLE t DROP PARTITION p1
  which is as fast and efficient as a drop table, but only for a part[ition] to the table.
- Another use case is to speed up partial scans in TiFlash, like reports for a single range, then range partitioning makes the scanning to only read matching partitions.