How we build TiDB

Max Liu | PingCAP

Amsterdam, Netherlands | October 5, 2016

PERCONA LIVE EUROPE AMSTERDAM

OPEN SOURCE DATABASE CONFERENCE

About me

- Infrastructure engineer / CEO of PingCAP
- Working on open source projects:
  - TiDB: https://github.com/pingcap/tidb
  - TiKV: https://github.com/pingcap/tikv
- Email: liuqi@pingcap.com

Agenda

- Why another database?
- What to build?
- How to design?
  - *The principles or the philosophy*
  - *The logical architecture*
  - *The alternatives*
- How to develop?
  - *The architecture*
  - *TiKV core technologies*
  - *TiDB core technologies*
- How to test?

Why another database ?

- RDBMS
- NoSQL
- NewSQL: F1 and Spanner

What to build?

- SQL
- Scalability
- ACID Transaction
- High Availability

A Distributed, Consistent, Scalable, SQL Database.

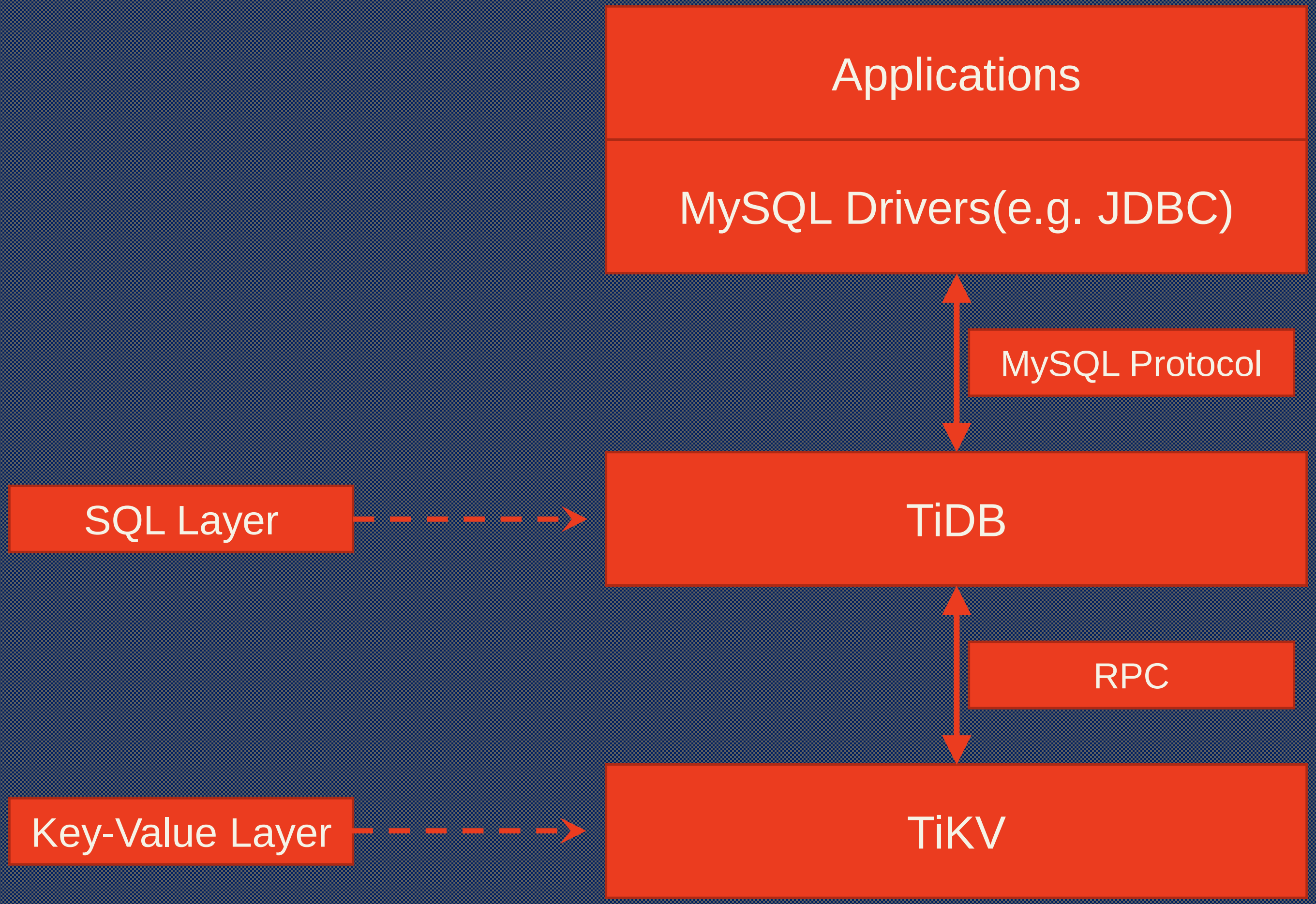
How to design

How to design

- The design principles
- The logical architecture
- The alternatives

The design principles

- User-oriented
  - *Disaster recovery*
  - *Easy to use*
  - *The community and ecosystem*
- Loose coupling
- The alternatives



Disaster recovery

Make sure we never lose any data!

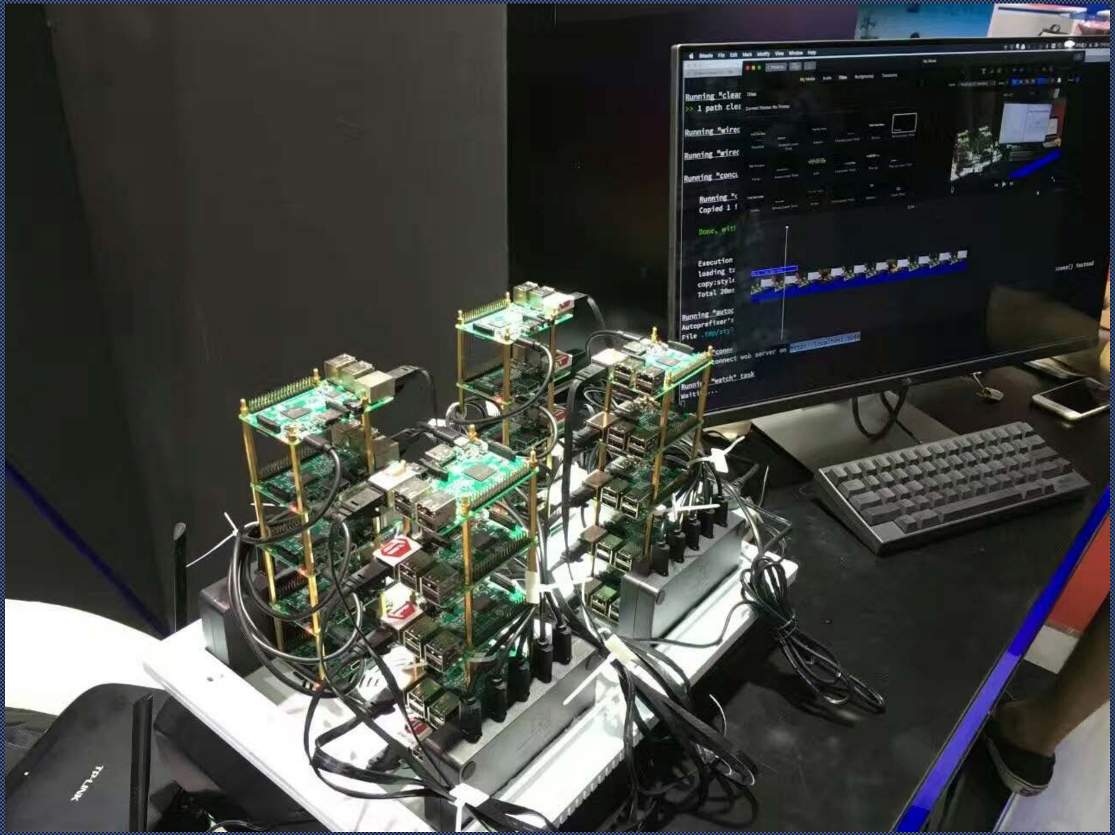
- Multiple replicas are just not enough
- Still need to keep Binlog in both the SQL layer and the Key-Value layer
- Make sure we have a backup in case the entire cluster crashes

Easy to use

- No scary sharding key
- No partition
- No explicit local index/global index
- Making scale transparent

Cross-platform

- On-premise devices: like Raspberry Pi
- Container: Docker
- Cloud: public, private, hybrid



The community and ecosystem

To engage, contribute and collaborate

- TiDB and MySQL
  - Compatible with most of the MySQL drivers(ODBC, JDBC) and SQL syntax
  - Compatible with MySQL clients and ORM
  - Compatible with MySQL management tools and bench tools
    - MySQLdump, mydumper, myloader
    - phpMyAdmin, Navicat, Workbench...
    - Sysbench

The community and ecosystem

To engage, contribute and collaborate

- TiKV and etcd
  - Share the Raft implementation
  - Peer review the Raft module

The community and ecosystem

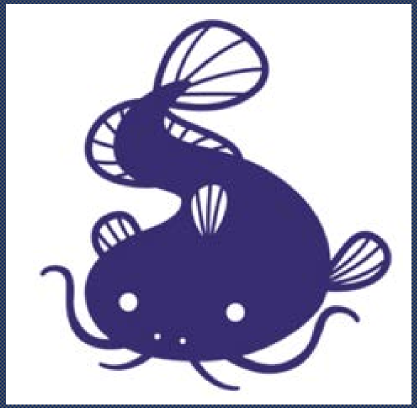
To engage, contribute and collaborate

- TiKV and RocksDB

The community and ecosystem

To engage, contribute and collaborate

- TiKV and Namazu
  - https://github.com/pingcap/tikv/issues/846



The community and ecosystem

To engage, contribute and collaborate

- The Rust community:
  - Prometheus

Thanks to the Rust team, gRPC, Prometheus!

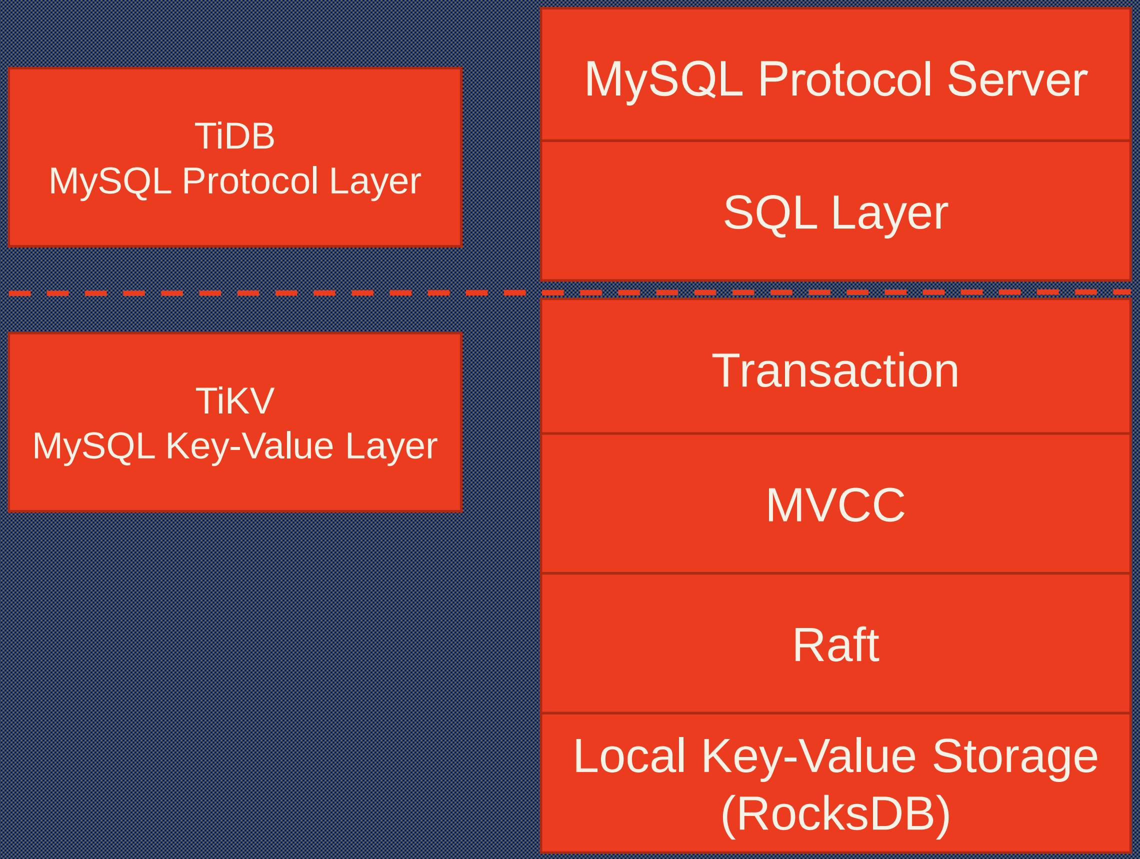
The community and ecosystem

To engage, contribute and collaborate

- Spark connector
  - Why Spark connector?
    - *TiDB is great for small or medium queries, such as 10 million * 1 million filter*
    - *Spark is better for complex queries with lots of data.*

Loose coupling - the logical architecture

- Highly layered
- Raft for consistency and scalability
- No distributed file system



The alternatives

Atomic clocks or GPS clocks

Distributed file system

Paxos

C++

VS

Timestamp Allocator

RocksDB

Raft

Go & Rust

Atomic clocks / GPS clocks VS TimeStamp Allocator

Time is important.

Real time is vital in distributed systems.

Can we get real time?

Clock drift.

Real time

We can’t get real time precisely because of clock drift, whether it’s GPS or Atomic Clocks…

Atomic clocks / GPS clocks VS TimeStamp Allocator

Pros

- Easy to implement
- Don't depend on hardware

Cons

Latency is high for multiple DCs

Distributed file system VS RocksDB

Pros

- Simple
- Fast
- Easy to tune

Cons

Not easy to work with k8s properly

Paxos VS Raft

Pros

- Easy to understand and implement
- Widely used and well tested

Cons

Almost none ☺

C++ VS Go & Rust

Pros

- Go for fast development and concurrency
- Rust for quality and performance

Cons

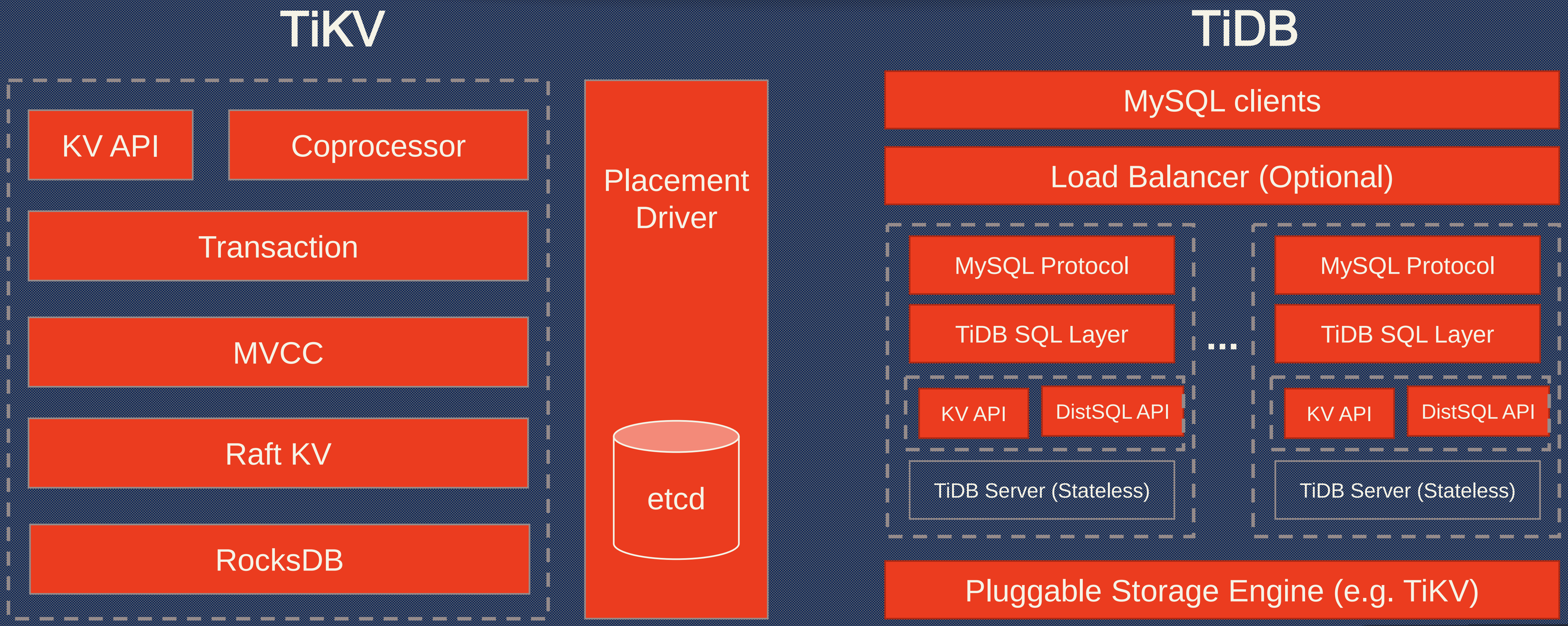
Third-party libraries

How to develop

How to develop

- The architecture
- The core technologies
  - TiKV – written in Rust
  - TiDB – written in Go

The architecture



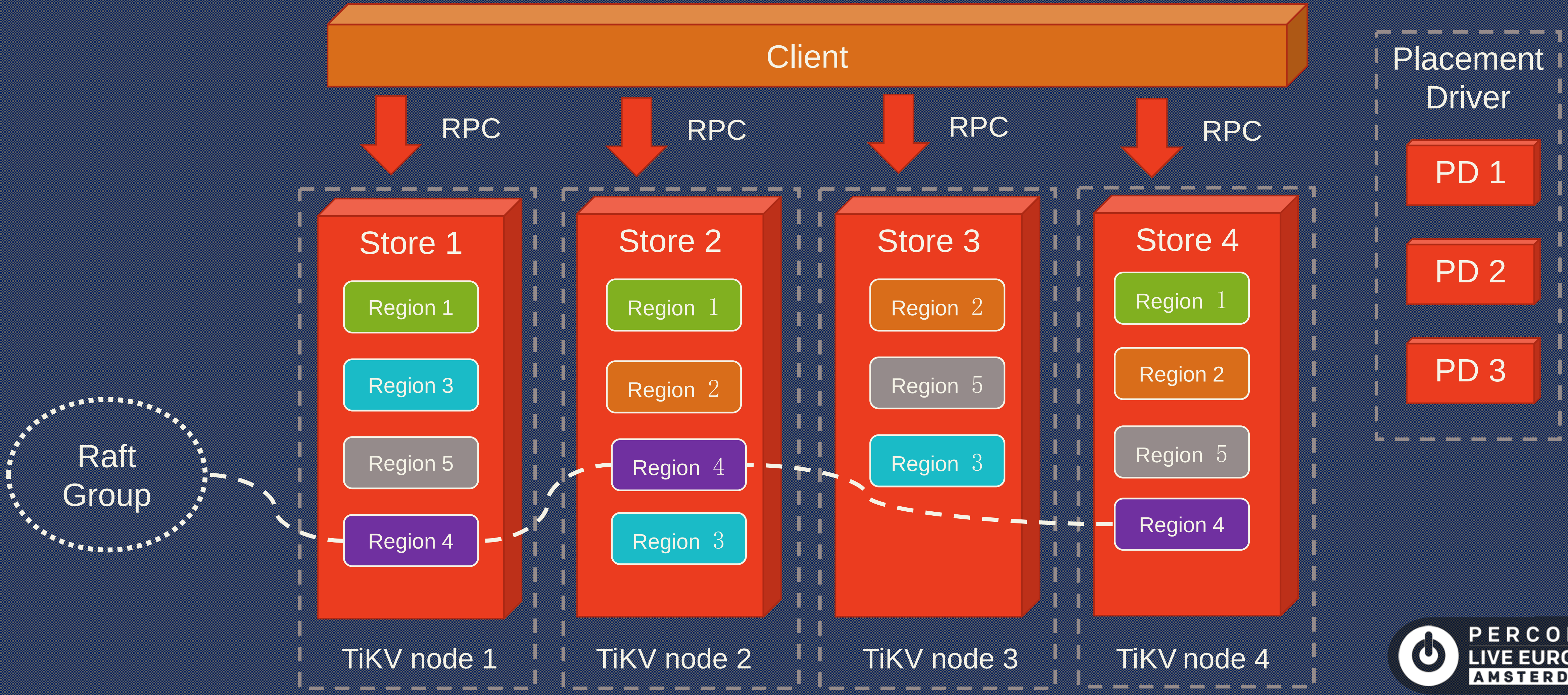
TiKV core technologies

TiKV core technologies

A distributed Key-Value layer to store data

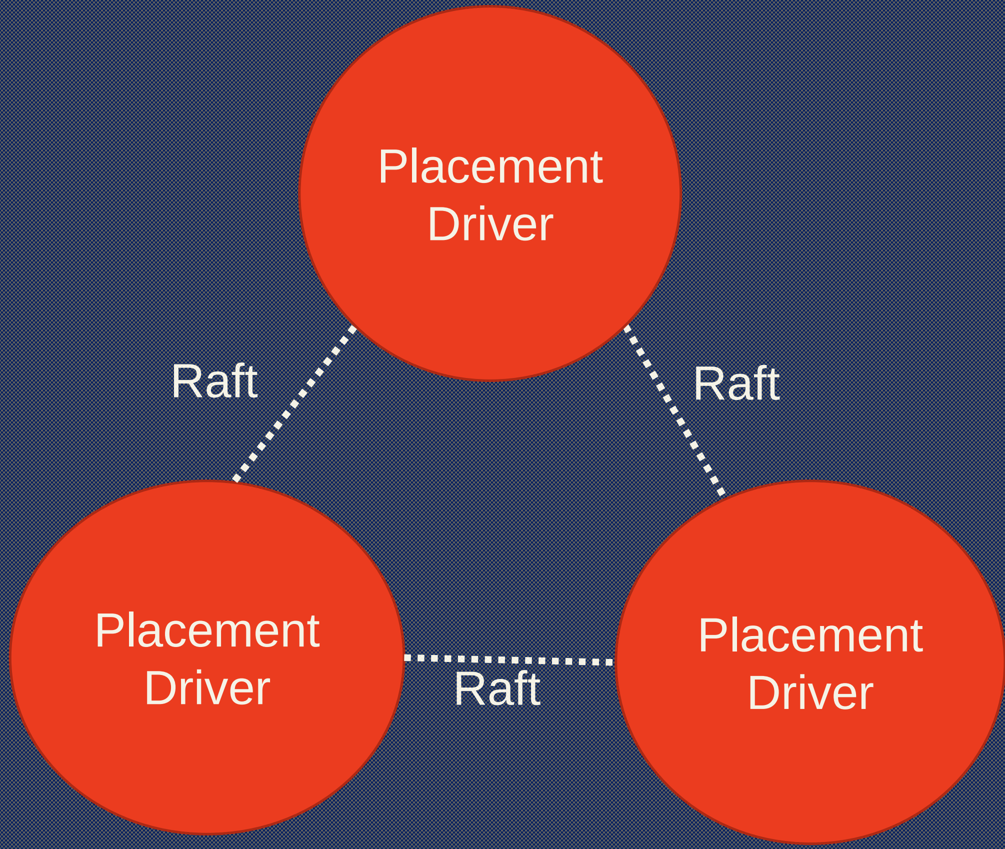
- TiKV software stack
- Placement Driver
- Raft
  - Scale and failover
- Multi-Version Concurrency Control (MVCC)
- Transaction

TiKV software stack



Placement Driver

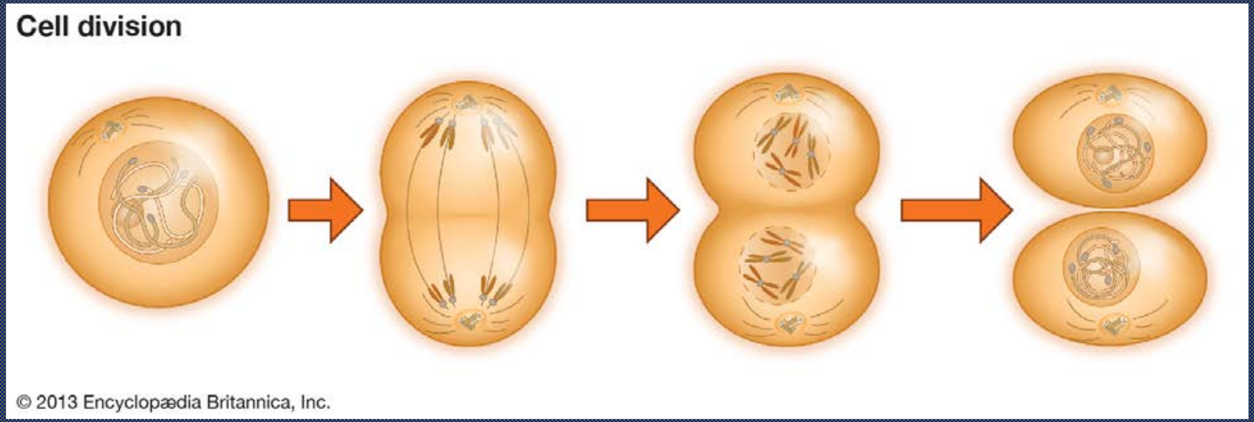
- The concept comes from Spanner
- Provide the God's view of the entire cluster
- Store the metadata
  - Clients have cache of the placement information
- Maintain the replication constraint
  - 3 replicas by default
- Data movement for balancing the workload
- It's a cluster too, of course
  - Thanks to Raft



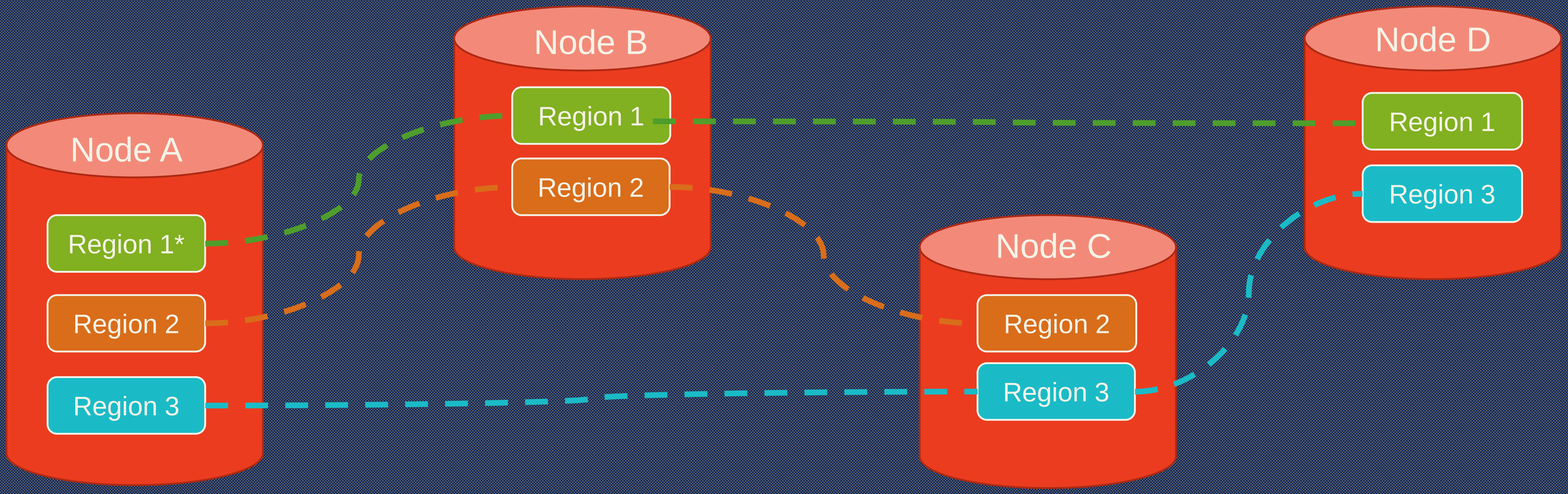
Raft

For scaling-out and failover/replication

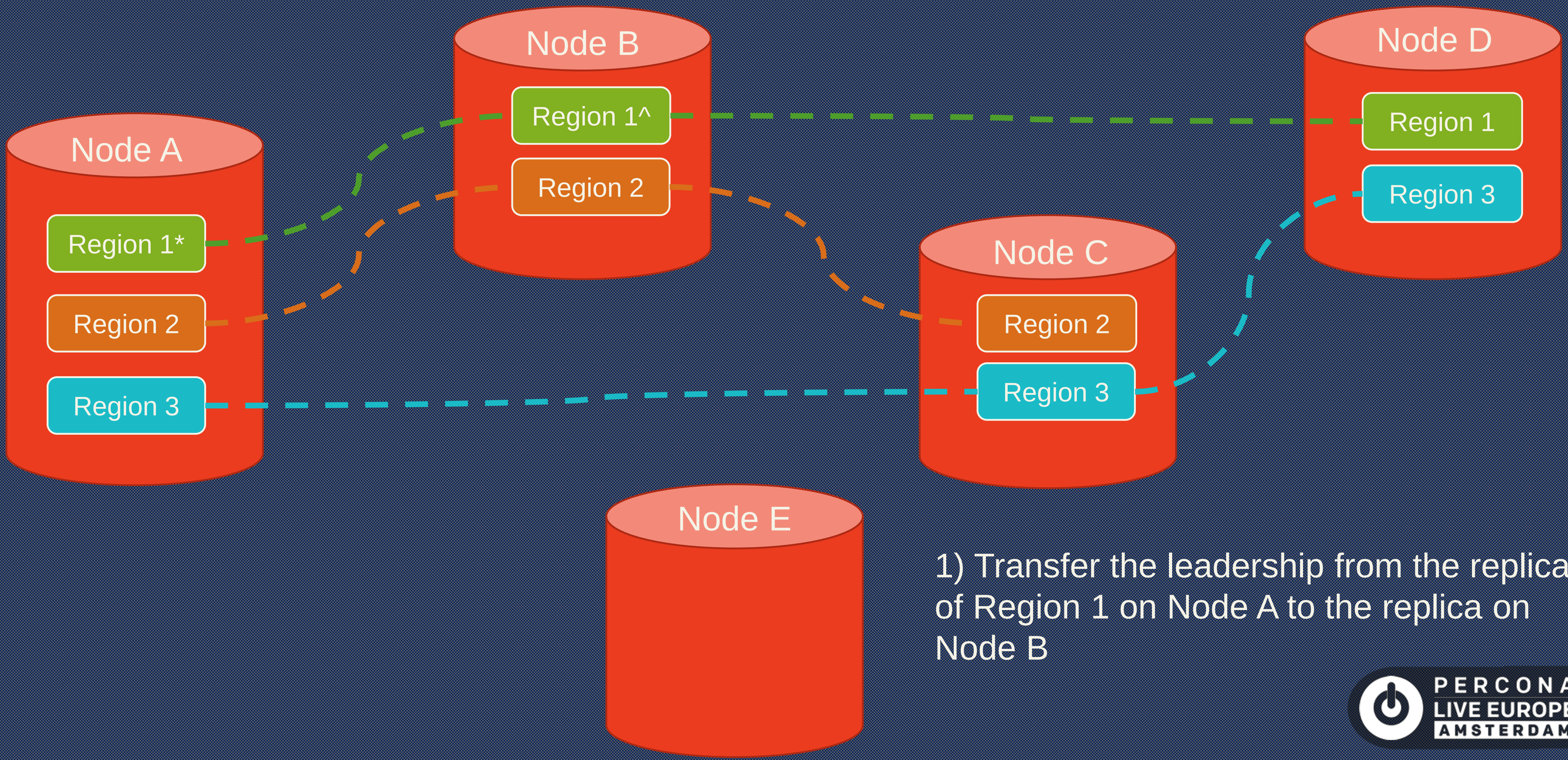
- Data is organized in **Regions**
- The replicas of one region form a Raft group
  - Multi-Raft
- Workload is distributed among multiple regions
  - There could be millions of regions in one big cluster
- Once a region is too large, it will be split to two smaller regions.
  - Just like a cell division



Scale-out (initial state)

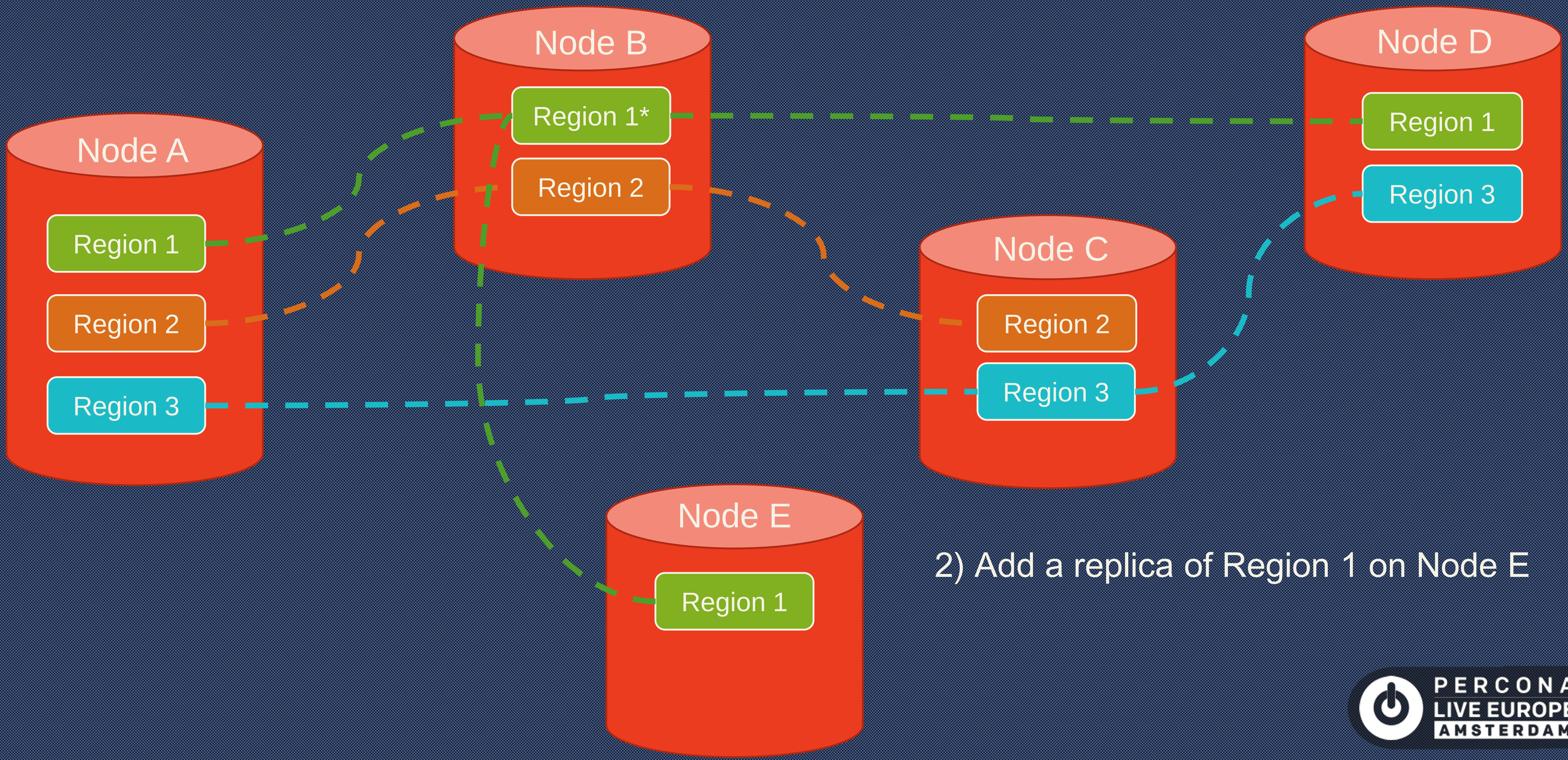


Scale-out (balancing)



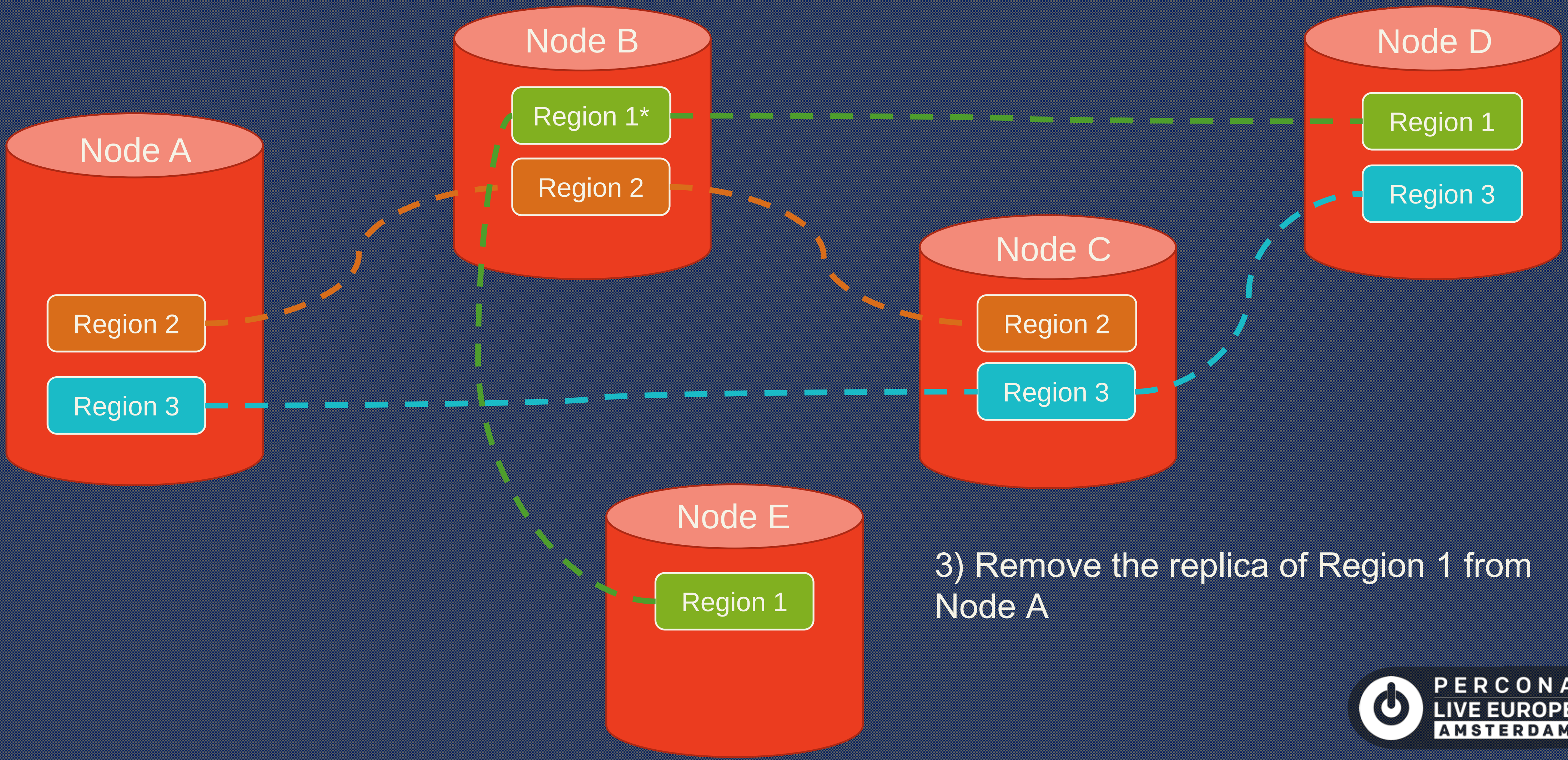
1) Transfer the leadership from the replica of Region 1 on Node A to the replica on Node B

Scale-out (balancing)



2) Add a replica of Region 1 on Node E

Scale-out (balancing)



3) Remove the replica of Region 1 from Node A

MVCC

- Each transaction sees a snapshot of the database at the beginning time of this transaction. Any changes made by this transaction will not seen by other transactions until the transaction is committed.
- Data is tagged with versions
  - Key_version: value
- Lock-free snapshot reads

Transaction API

```
txn := store.Begin() // start a transaction
txn.Set([]byte("key1"), []byte("value1"))
txn.Set([]byte("key2"), []byte("value2"))
err = txn.Commit() // commit transaction
if err != nil {
     txn.Rollback()
}
```

I want to write
code like this!

Transaction model

- Inspired by Google Percolator, 2-phase commit
- 3 column families
  - **cf:lock**: An uncommitted transaction is writing this cell; contains the location/pointer of primary lock
  - **cf: write**: it stores the commit timestamp of the data
  - **cf: data**: Stores the data itself

Transaction model

Bob wants to transfer $7 to Joe

Key	Bal: Data	Bal: Lock	Bal: Write
Bob	6: 5: $10	6: 5:	6: data @ 5 5:
Joe	6: 5: $2	6: 5:	6: data @ 5 5:

Transaction model

Key	Bal: Data	Bal: Lock	Bal: Write
Bob	7: $3 6: 5: $10	7: I am Primary 6: 5:	7: 6: data @ 5 5:
Joe	6: 5: $2	6: 5:	6: data @ 5 5:

Transaction model

Key	Bal: Data	Bal: Lock	Bal: Write
Bob	7: \$3 6: 5: \$10	7: I am Primary 6: 5:	7: 6: data @ 5 5:
Joe	7: \$9 6: 5: \$2	7: Primary@Bob.bal 6: 5:	7: 6: data @ 5 5:

Transaction model

Key	Bal: Data	Bal: Lock	Bal: Write
Bob	8: 7: $3 6: 5: $10	8: 7: ~~I am Primary~~ 6: 5:	8: data @ 7 7: 6: data @ 5 5:
Joe	8: 7: $9 6: 5: $2	8: 7: Primary@Bob.bal 6: 5:	8: data @ 7 7: 6: data @ 5 5:

Transaction model

Key	Bal: Data	Bal: Lock	Bal: Write
Bob	8: 7: $6 6: 5: $10	8: 7: 6: 5:	8: data @ 7 7: 6: data @ 5 5:
Joe	8: 7: $6 6: 5: $2	8: 7:~~Primary@Bob.bal~~ 6: 5:	8: data @ 7 7: 6: data @ 5 5:

Transaction model

Key	Bal: Data	Bal: Lock	Bal: Write
Bob	8: 7: $6 6: 5: $10	8: 7: 6: 5:	8: data @ 7 7: 6: data @ 5 5:
Joe	8: 7: $6 6: 5: $2	8: 7: 6: 5:	8: data @ 7 7: 6: data @ 5 5:

TiDB core technologies

PERCONA LIVE EUROPE AMSTERDAM

TiDB core technologies

A protocol layer that is compatible with MySQL

- Mapping table data to Key-Value store
- Predicate push-down
- Online DDL changes

Mapping table data to Key-Value store

INSERT INTO user VALUES (1, “bob”, “huang@pingcap.com”);

INSERT INTO user VALUES (2, “tom”, “tom@pingcap.com”);

Key	Value
user/1	bob | huang@pingcap.com
user/2	tom | tom@pingcap.com
…	…

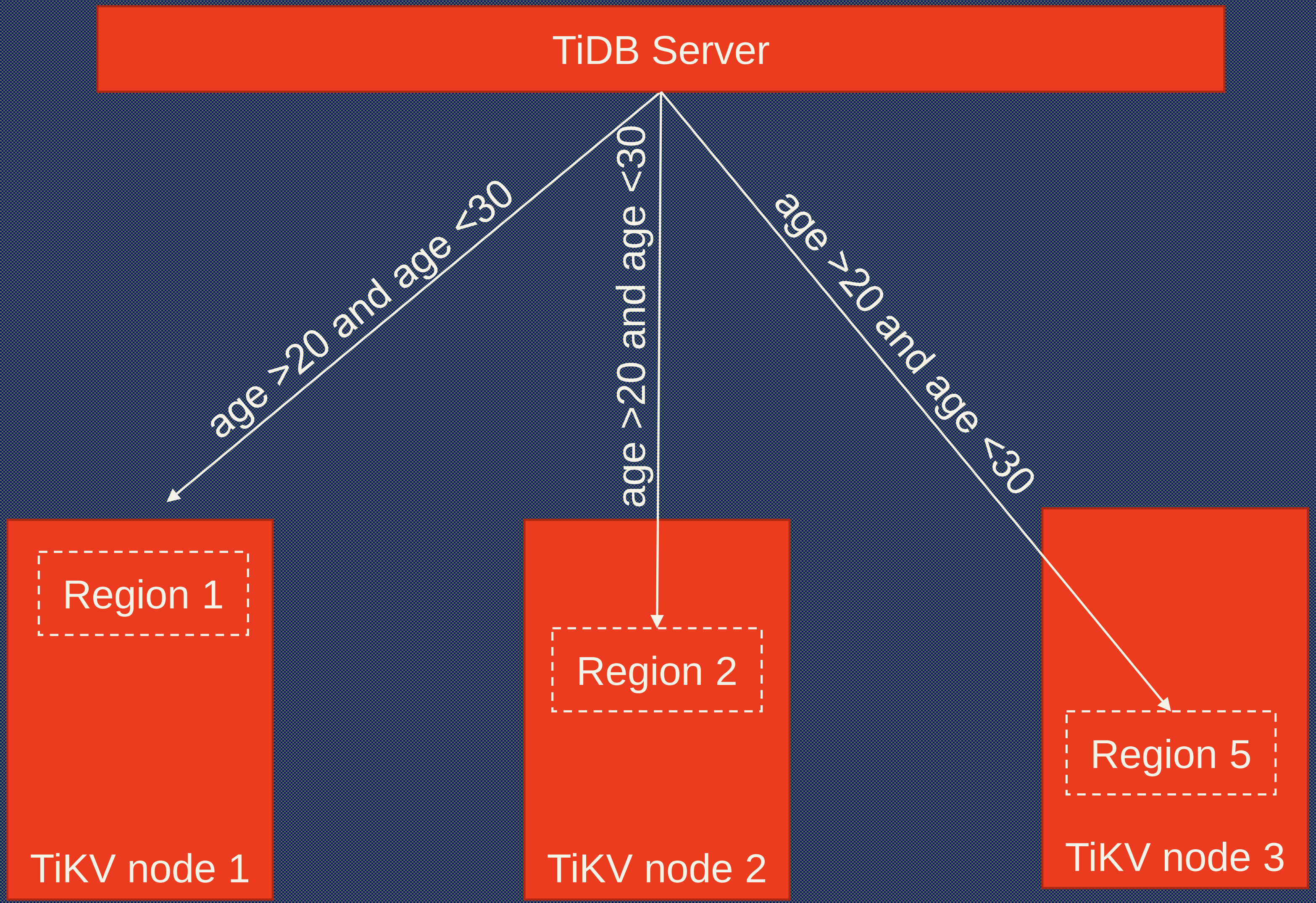
Secondary index is necessary

- Global index
  - All indexes in TiDB are transactional and fully consistent
  - Stored as separate key-value pairs in TiKV
- Example: create index for user name

Key	Value()
bob_1	user/1
tom_2	user/2
…	…

Key	Value()
user/1	bob | bob@pingcap.com
user/2	tom | tom@pingcap.com
…	…

Predicate push-down



TiDB knows that Region 1/2/5 stores the data of the person table.

Schema changes

- Why online schema change is a must-have?
  - Full data availability
  - Minimal performance impact
- Example: add index for a column

Something the same as Google F1

The main features of TiDB that impact schema changes are:

- Distributed
  - An instance of TiDB consists of many individual TiDB servers
- Relational schema
  - Each TiDB server has a copy of a relational schema that describes tables, columns, indexes, and constraints.
  - Any modification to the schema requires a distributed schema change to update all servers.
- Shared data storage
  - All TiDB servers in all datacenters have access to all data stored in TiKV.
  - There is no partitioning of data among TiDB servers.
- No global membership
  - Because TiDB servers are stateless, there is no need for TiDB to implement a global membership protocol. This means there is no reliable mechanism to determine currently running TiDB servers, and explicit global synchronization is not possible.

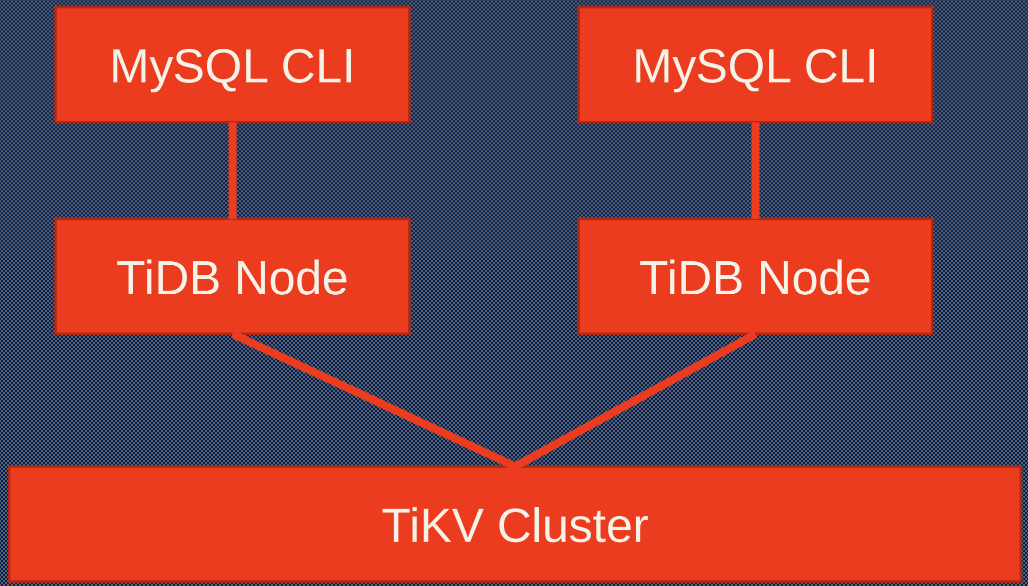
Something different from Google F1

- TiDB speaks MySQL protocol
- Statements inside of a single transaction cannot cross different TiDB servers

http://static.googleusercontent.com/media/research.google.com/zh-CN//pubs/archive/41376.pdf

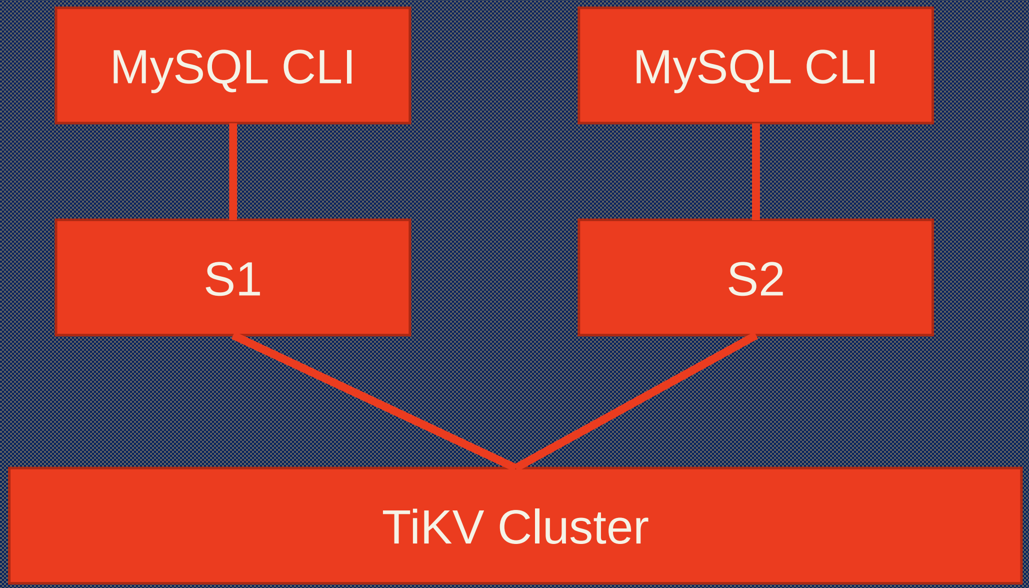
One more thing before schema change

The big picture of SQL



Overview of a TiDB instance during a schema change

TiDB Servers
Schema 1 and
Schema 2



Schema change: Adding index

Servers using different schema versions may **corrupt** the database if we are not careful.

Consider a schema change from schema S1 to schema S2 that adds index I on table T. Assume two different servers, M1 and M2, execute the following sequence of operations:

1. Server M2, using schema S2, inserts a new row r to table T. Because S2 contains index I, server M2 also adds a new index entry corresponding to r to the key–value store.
2. Server M1, using schema S1, deletes r. Because S1 does not contain I, M1 removes r from the key–value store but fails to remove the corresponding index entry in I.

Schema states

- There are two states which we consider to be non-intermediate: absent and public
- There are two internal, intermediate states: **delete-only** and write-only
  - **Delete-only**: A delete-only table, column, or index cannot have their key–value pairs read by user transactions and

1. If E is a table or column, it can be modified only by the delete operations.
2. If E is an index, it is modified only by the delete and update operations. Moreover, the update operations can delete key–value pairs corresponding to updated index keys, but they cannot create any new one.

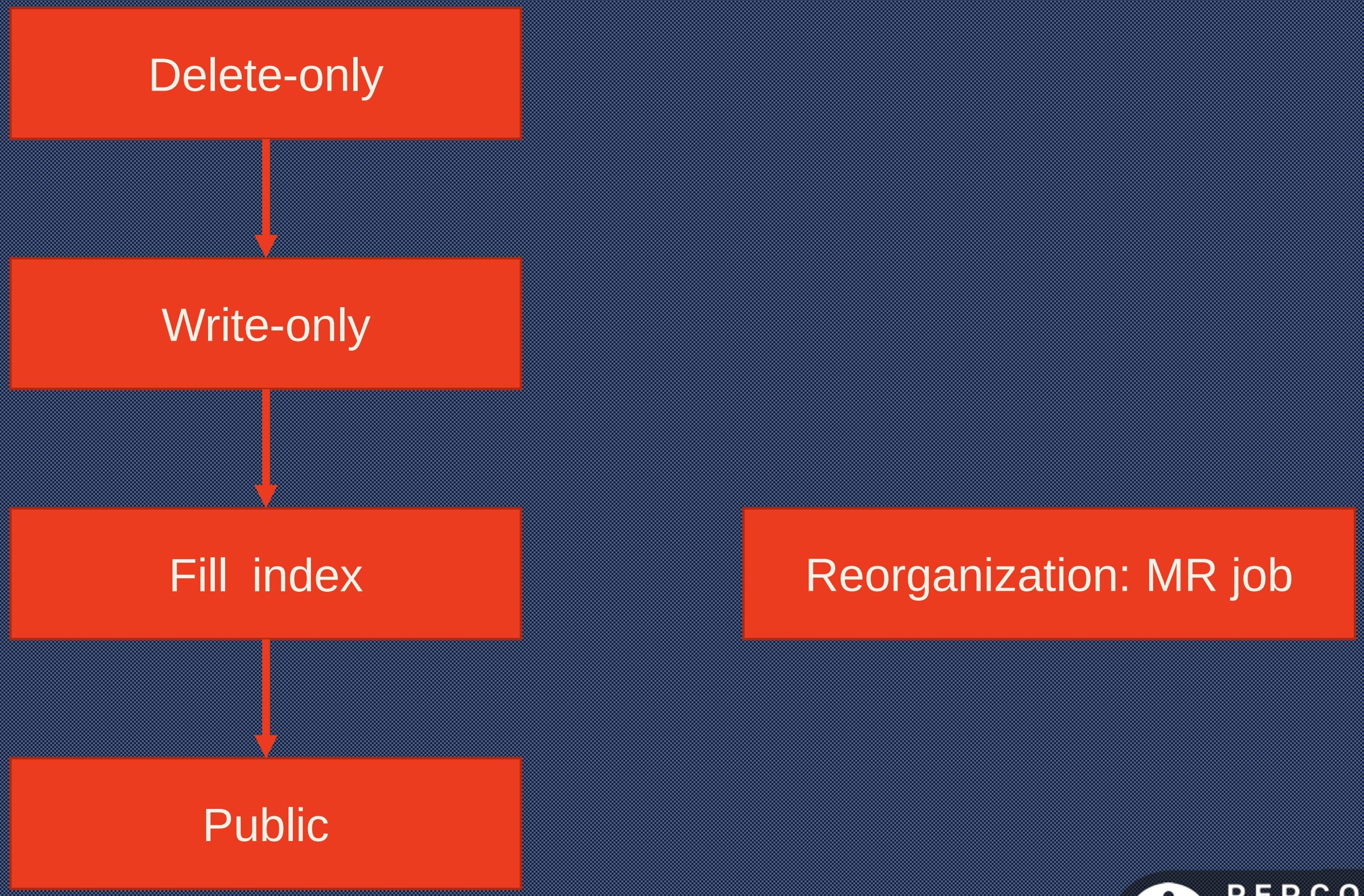
Schema states

There are two internal, intermediate states: delete-only and **write-only**

- **Write-only**: The write-only state is defined for columns and indexes as follows:

A write-only column or index can have their key–value pairs modified by the insert, delete, and update operations, but none of their pairs can be read by user transactions.

Schema change flow: Add index



TiDB: status of Adding index (delete-only)

```
MySQL [test]> show status;
+-------------------------------+--------------------------------------+
| Variable_name                 | Value                                |
+-------------------------------+--------------------------------------+
| bg_schema_version             | 204                                  |
| ddl_job_args                  | []                                   |
| server_id                     | c39ac3ec-5b1d-481e-a0df-9cf71004c615 |
| ddl_last_reload_schema_ts     | 1474463322                           |
| ddl_job_schema_id             | 1                                    |
| bg_owner_id                   | c39ac3ec-5b1d-481e-a0df-9cf71004c615 |
| ddl_job_state                 | running                              |
| ddl_job_error                 |                                      |
| ddl_job_id                    | 248                                  |
| ddl_job_snapshot_ver          | 0                                    |
| ddl_schema_version            | 204                                  |
| ddl_job_last_update_ts        | 1474463321                           |
| ddl_job_schema_state          | delete only                          |
| ddl_job_reorg_handle          | 0                                    |
| ddl_owner_id                  | c39ac3ec-5b1d-481e-a0df-9cf71004c615 |
| ddl_job_action                | add index                            |
| ddl_owner_last_update_ts      | 1474463321                           |
| bg_owner_last_update_ts       | 1474463322                           |
| ddl_job_table_id              | 103                                  |
+-------------------------------+--------------------------------------+
19 rows in set (0.00 sec)
```

TiDB: status of Adding index (add index)

```
MySQL [test]> show status;
+-----------------------------+---------------------------------------+
| Variable_name               | Value                                 |
+-----------------------------+---------------------------------------+
| ddl_owner_last_update_ts    | 1474461957                            |
| ddl_schema_version          | 179                                   |
| ddl_job_schema_state        | write reorganization                  |
| ddl_job_error               |                                       |
| bg_schema_version           | 179                                   |
| ddl_job_schema_id           | 1                                     |
| bg_owner_id                 | c39ac3ec-5b1d-481e-a0df-9cf71004c615 |
| ddl_owner_id                | c39ac3ec-5b1d-481e-a0df-9cf71004c615 |
| ddl_job_action              | add index                             |
| ddl_last_reload_schema_ts   | 1474461959                            |
| ddl_job_table_id            | 176                                   |
| bg_owner_last_update_ts     | 1474461958                            |
| ddl_job_state               | running                               |
| ddl_job_args                | []                                    |
| ddl_job_snapshot_ver        | 386521338774814722                    |
| ddl_job_last_update_ts      | 1474461957                            |
| ddl_job_reorg_handle        | 18996                                 |
| server_id                   | c39ac3ec-5b1d-481e-a0df-9cf71004c615 |
| ddl_job_id                  | 246                                   |
+-----------------------------+---------------------------------------+
19 rows in set (0.00 sec)
```

How to test

How to test

- Test cases from community
  - Lots of test cases in MySQL drivers/connectors
  - Lots of ORMs
  - Lots of applications
- Fault injection
  - Hardware: disk error, network card, cpu, clock
  - Software: file system, network and protocol
- Simulate Network: https://github.com/pingcap/tikv/blob/master/tests/raftstore/transport_simulate.rs
- Distribute testing
  - Jepsen
  - Namazu

The future

- GPS and Atomic clock
- Better query optimizer
- Better compatibility with MySQL
- Support the JSON and document type
- Push down more aggregation and built-in functions
- Using gPRC instead of customizing the RPC implementation

Here we are!

TiDB: https://github.com/pingcap/tidb
TiKV: https://github.com/pingcap/tikv

Email: liuqi@pingcap.com

Thank you!

Questions?

PERCONA LIVE EUROPE AMSTERDAM

Rate My Session!