

How we build TiDB

dongxu | PingCAP

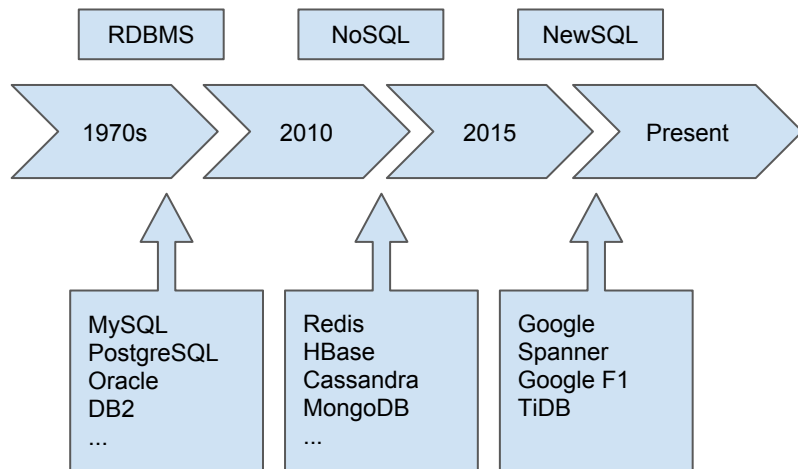


About me

- Dongxu Huang (黄东旭)
- Co-founder & CTO of PingCAP
- MSRA / Netease / Wandoulabs / PingCAP
- Infrastructure software engineer / Open source hacker
- Codis / TiDB / TiKV

Let's say we want to build a NewSQL Database

- From the beginning
- What's wrong with the existing DBs?
 - RDBMS
 - NoSQL
- NewSQL: F1 & Spanner



What to build?

- SQL
- Scalability
- ACID Transaction
- High Availability



A Distributed, Consistent, Scalable, SQL Database that supports the best features of both traditional RDBMS and NoSQL

Open source, of course

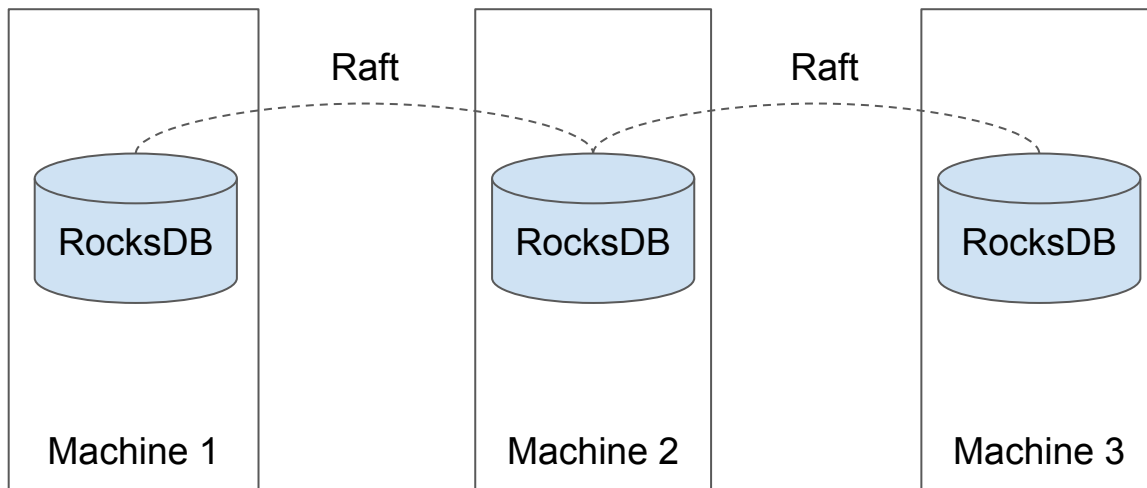
We have a key-value store (RocksDB)

- Good start, RocksDB is fast and stable.
 - Atomic batch write
 - Snapshot
- However... It's a local embedded kv store.
 - Can't [tolerate](#) machine [failures](#)
 - [Scalability](#) depends on the capacity of the disk

Let's fix Fault Tolerance

- Use Raft to replicate data
 - Key features of Raft
 - Strong leader: leader does most of the work, issue all log updates
 - Leader election
 - Membership changes
- Implementation:
 - Ported from etcd
- Replicas are distributed across datacenters

Let's fix Fault Tolerance



That's cool

- Basically we have a lite version of etcd or zookeeper.
 - Does not support watch command, and some other features
- Let's make it better.

How about Scalability?

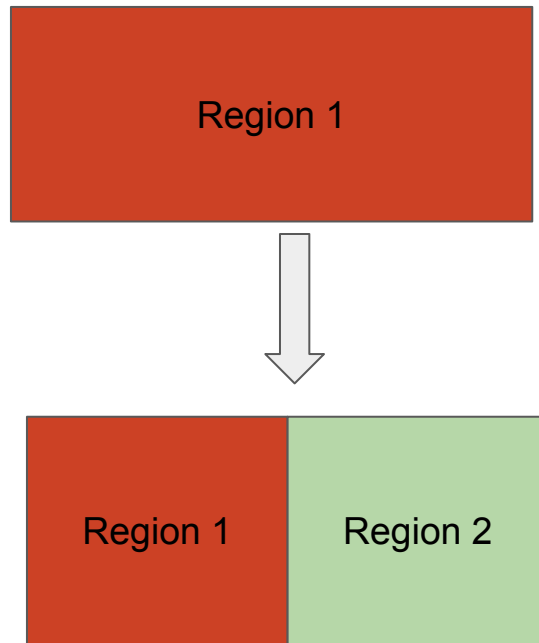
- What if we **SPLIT** data into many regions?
 - We got many Raft groups.
 - Region = Contiguous Keys
- Hash partitioning or Range partitioning
 - Redis: Hash partitioning
 - HBase: Range partitioning

That's Cool, but...

- But what if we want to scan data?
 - How to support API: **scan**(startKey, endKey, limit)
- So, we need a **globally ordered map**
 - Can't use hash partitioning
 - Use **range** partitioning
 - Region 1 -> [a - d]
 - Region 2 -> [e - h]
 - ...
 - Region n -> [w - z]

How to scale? (1/2)

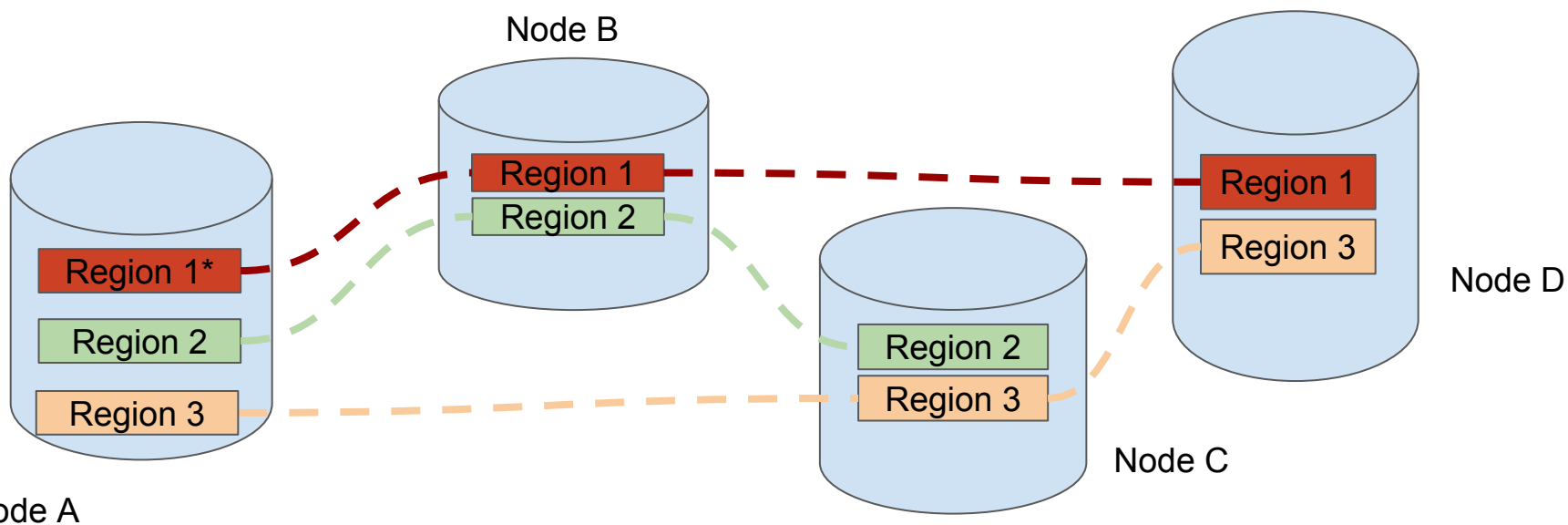
- That's simple
- Just Split & Move



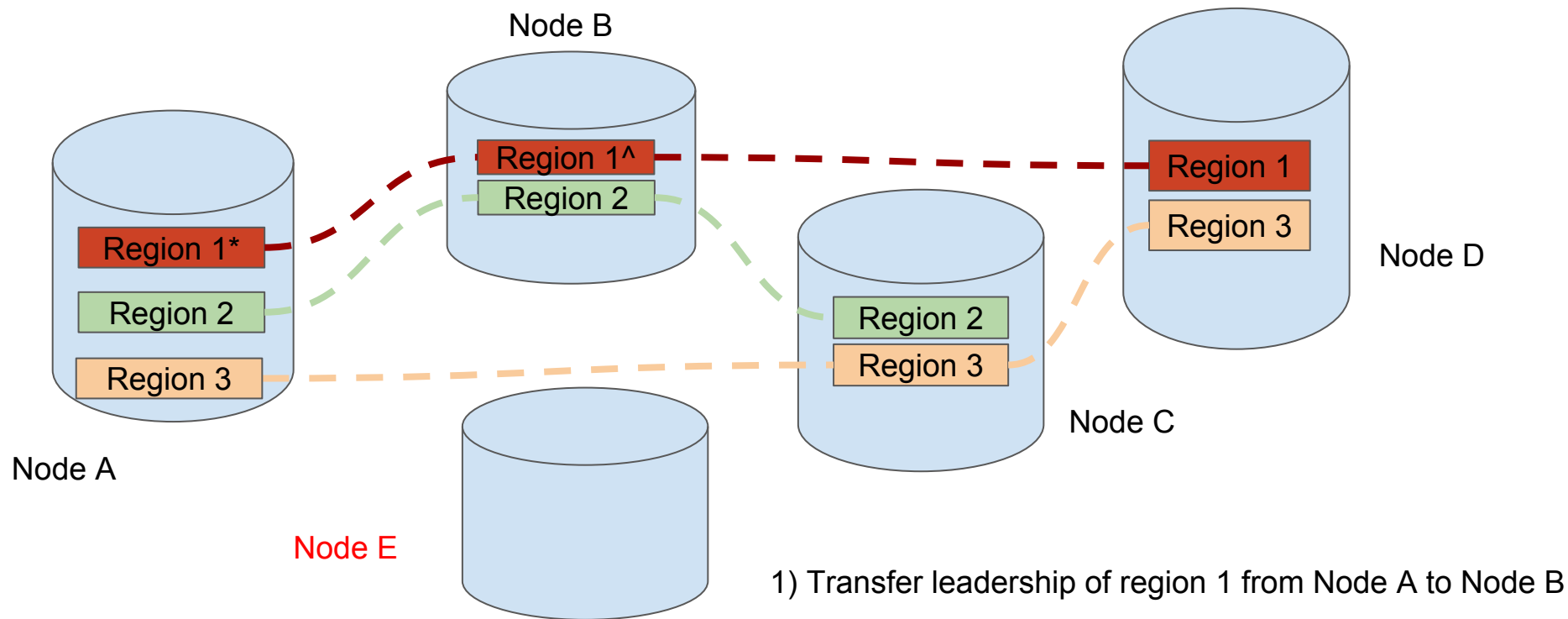
How to scale? (2/2)

- Raft comes for rescue again
 - Using Raft Membership changes, 2 steps:
 - Add a new replica
 - Destroy old region replica

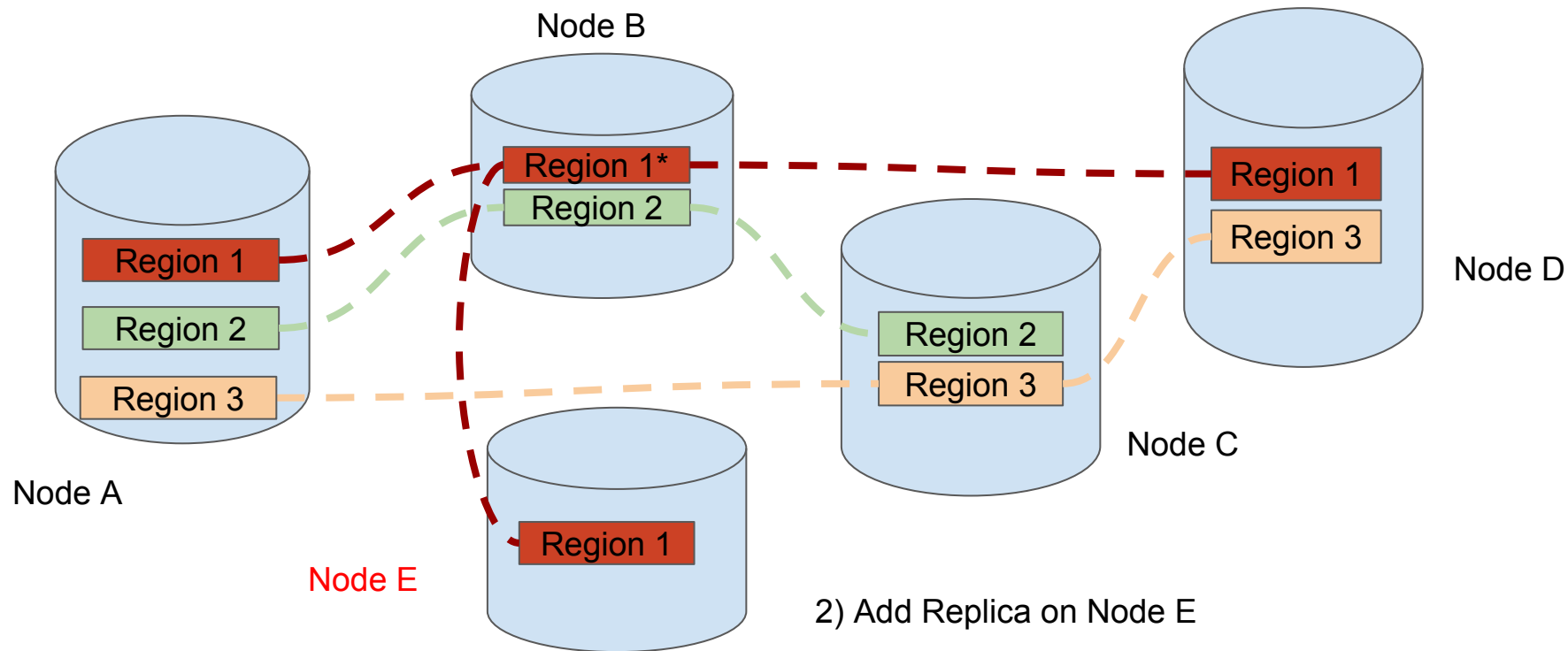
Scale-out (initial state)



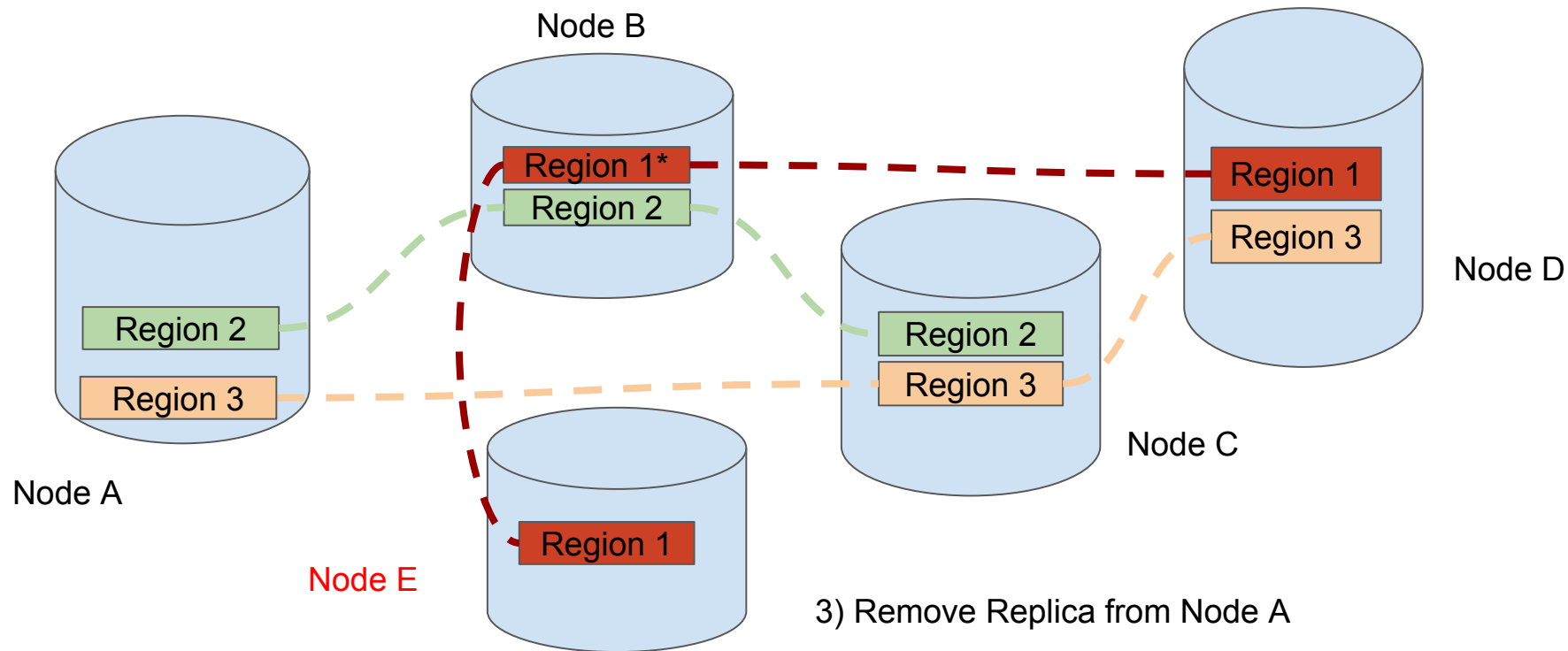
Scale-out (add new node)



Scale-out (balancing)



Scale-out (balancing)

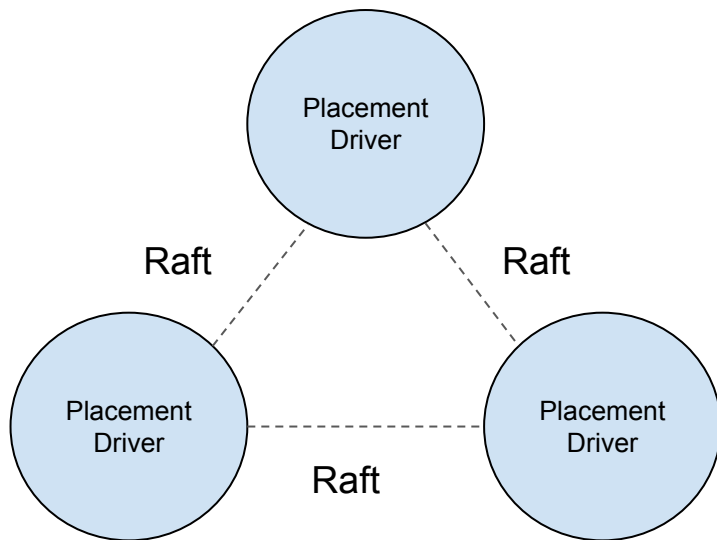


Now we have a **distributed** key-value store

- We want to keep replicas in different datacenters
 - For HA: any node might crash, even the whole Data center
 - And to balance the workload
- So, we need **Placement Driver** (PD) to act as cluster manager, for:
 - Replication constraint
 - Data movement

Placement Driver

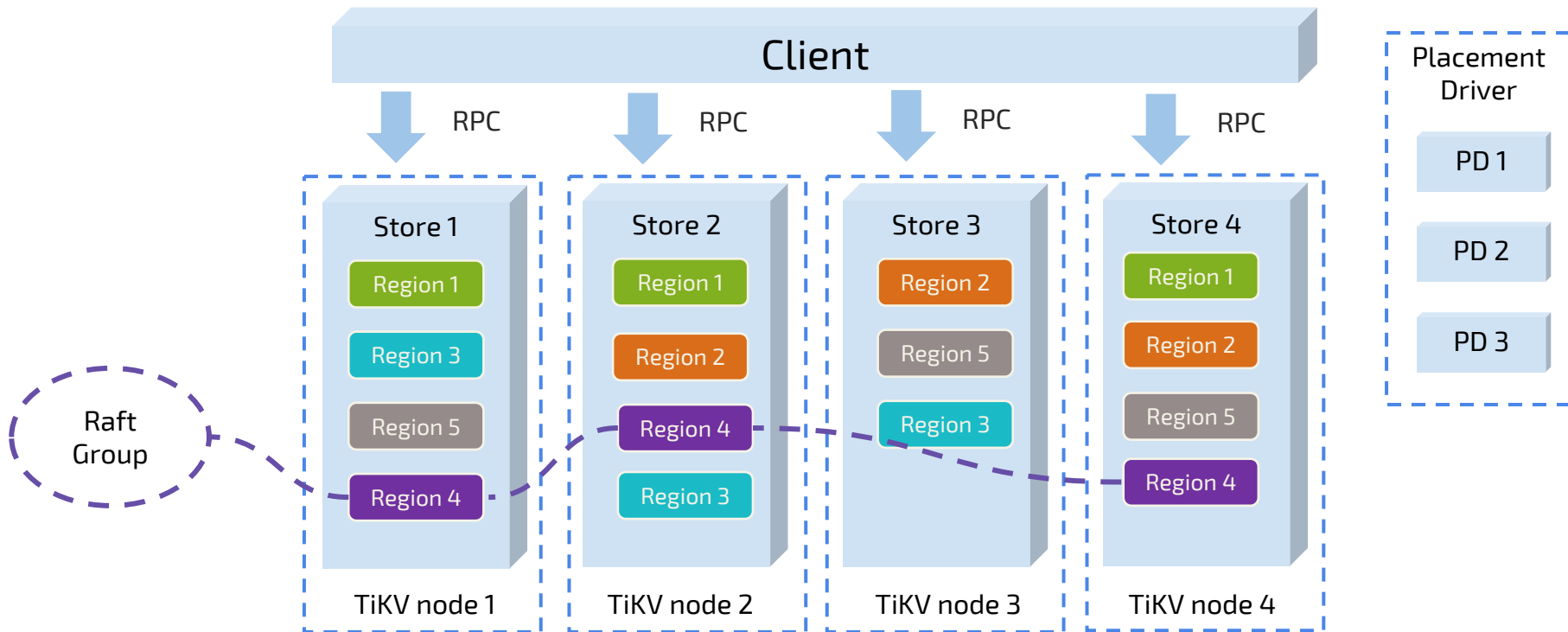
- The concept comes from Spanner
- Provide the God's view of the entire cluster
- Store the metadata
 - Clients have cache of placement information.
- Maintain the replication constraint
 - 3 replicas, by default
- Data movement for balancing the workload
- It's a cluster too, of course.
 - Thanks to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



Placement Driver

- Rebalance workload without moving data.
 - Raft: Leadership transfer extension
- Moving data is a slow operation.
- We need fast rebalance.

TiKV: The whole picture



That's Cool, but hold on...

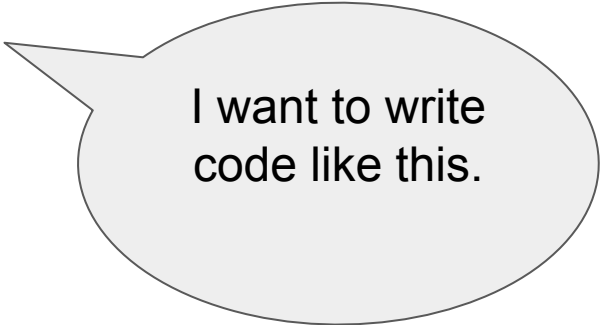
- It could be cooler if we have:
 - MVCC
 - ACID Transaction
 - Transaction mode: Google Percolator (2PC)

MVCC (Multi-Version Concurrency Control)

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

Transaction API style (go code)

```
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
- 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

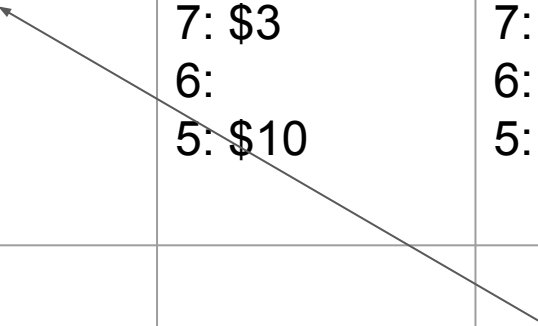
Key	Bal: Data	Bal: Lock	Bal: Write
Bob	6: 5: \$10	6: 5:	6: data @ 5 5:
Joe	6: 5: \$2	6: 5:	6: data @ 5 5:

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 (commit point)

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 6: 5:	8: data @ 7 7: 6: data @ 5 5:

Transaction Model

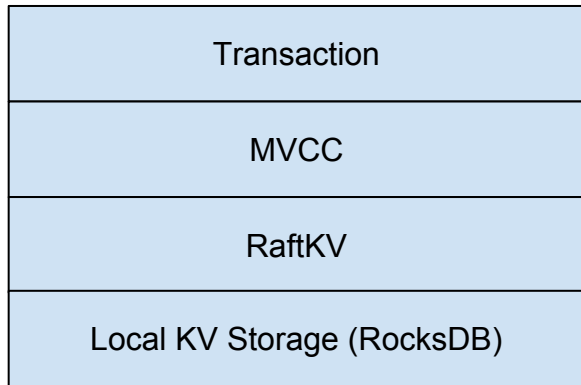
Key	Bal: Data	Bal: Lock	Bal: Write
Bob	8: 7: \$6 6: 5: \$10	8: 7: 6: 5:	8: data @ 7 7: 6: data @ 5 5:
Joe	8: 7: \$6 6: 5: \$2	8: 7: Primary @ Bob 6: 5:	8: data @ 7 7: 6: data @ 5 5:

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:

TiKV: Architecture overview (Logical)

- Highly layered
- Raft for consistency and scalability
- No distributed file system
 - For better performance and lower latency



That's really really Cool

- We have A **Distributed Key-Value Database** with
 - Geo-Replication / Auto Rebalance
 - **ACID Transaction** support
 - Horizontal **Scalability**

What if we support SQL?

- SQL is simple and very productive
- We want to write code like this:

```
SELECT COUNT(*) FROM user
      WHERE age > 20 and age < 30;
```

And this...

```
BEGIN;  
    INSERT INTO person VALUES('tom', 25);  
    INSERT INTO person VALUES('jerry', 30);  
COMMIT;
```

First of all, map table data to key value store

- What happens behind:

```
CREATE TABLE user (  
    id          INT PRIMARY KEY,  
    name       TEXT,  
    email      TEXT  
);
```

Mapping table data to kv store

```
INSERT INTO user VALUES (1, "bob", "bob@pingcap.com");  
INSERT INTO user VALUES (2, "tom", "tom@pingcap.com");
```

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

Secondary index is necessary

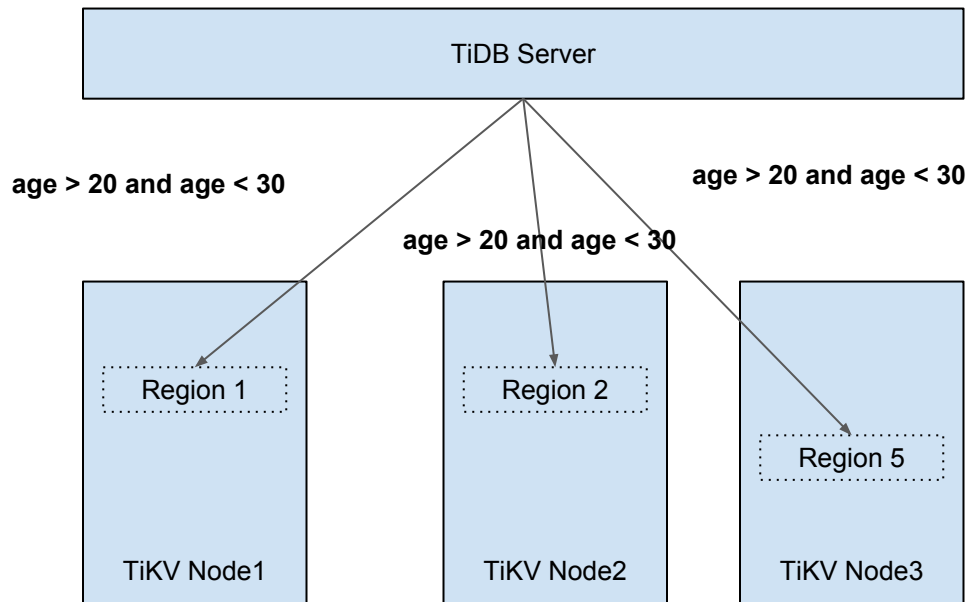
- Global index
 - All indexes in TiDB are transactional and fully consistent
 - Stored as separate key-value pairs in TiKV
- Keyed by a concatenation of the index prefix and primary key in TiKV
 - For example: table := {id, name} , id is primary key. If we want to build an index on the name column, for example we have a row $r := (1, 'tom')$, we could store another kv pair just like:
 - name_index/tom_1 => nil
 - name_index/tom_2 => nil
 - For unique index
 - id_index/tom => 1,

Index is just not enough...

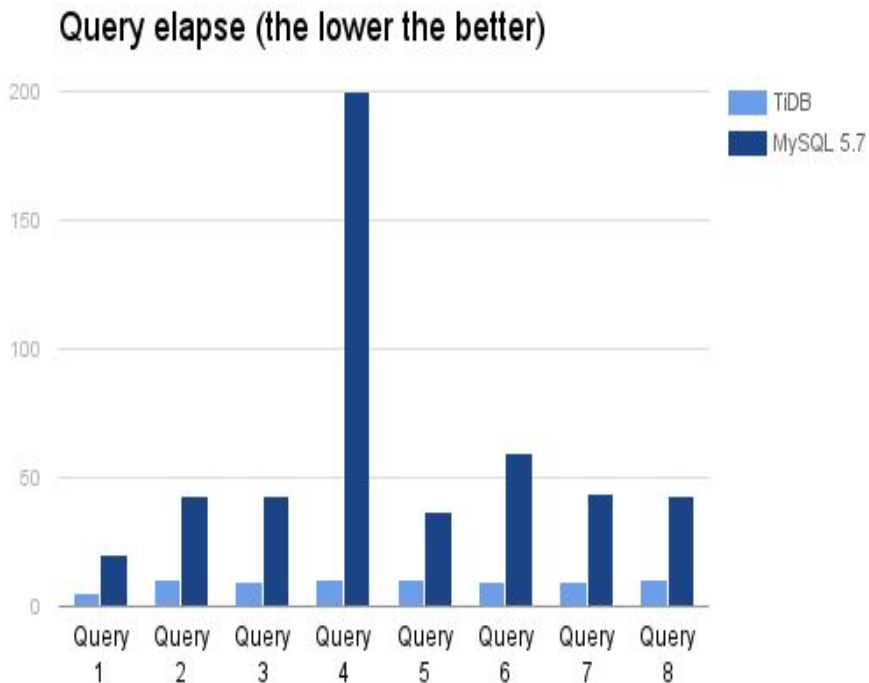
- Can we push down filters?
 - `select count(*) from person`
 where `age > 20 and age < 30`
- It should be much faster, maybe 100x
 - Less RPC round trip
 - Less transferring data

Predicate pushdown

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



Query performance



TiDB Elapse	MySQL Elapse
5.07699437s	19.93s
10.524703077s	43.23s
10.077812714s	43.33s
10.285957629s	>20 mins
10.462306097s	36.81s
9.968078965s	1 min 0.27 sec
9.998030375s	44.05s
10.866549284s	43.18s

- Rule based optimization
 - predicate push down
 - nested subqueries elimination
 - column pruning
 - unused projection elimination
 - eager aggregation
- cost based optimization
 - Equi-Depth Histogram
 - 基于interesting order/group 的Dynamic Programming 做 Plan generation.
 - Join ReOrder
 - 消除笛卡尔积
- 未来计划做的优化
 - DNF -> CNF (or -> and 支持更多的 push down)
 - Sketches based Synopses
 - Memo Framework, 并在其基础上做Join Re-Order)
 - View Merging

But TiKV doesn't know the schema

- Key-value database doesn't have any information about table and row
- Coprocessor comes for help:
 - Concept comes from HBase
 - Inject your own logic to data nodes

What about drivers for every language?

- We have to build drivers for Java, Python, PHP, C/C++, Rust, Go...
- It needs lots of time and code.
 - Trust me, you don't want to do that.

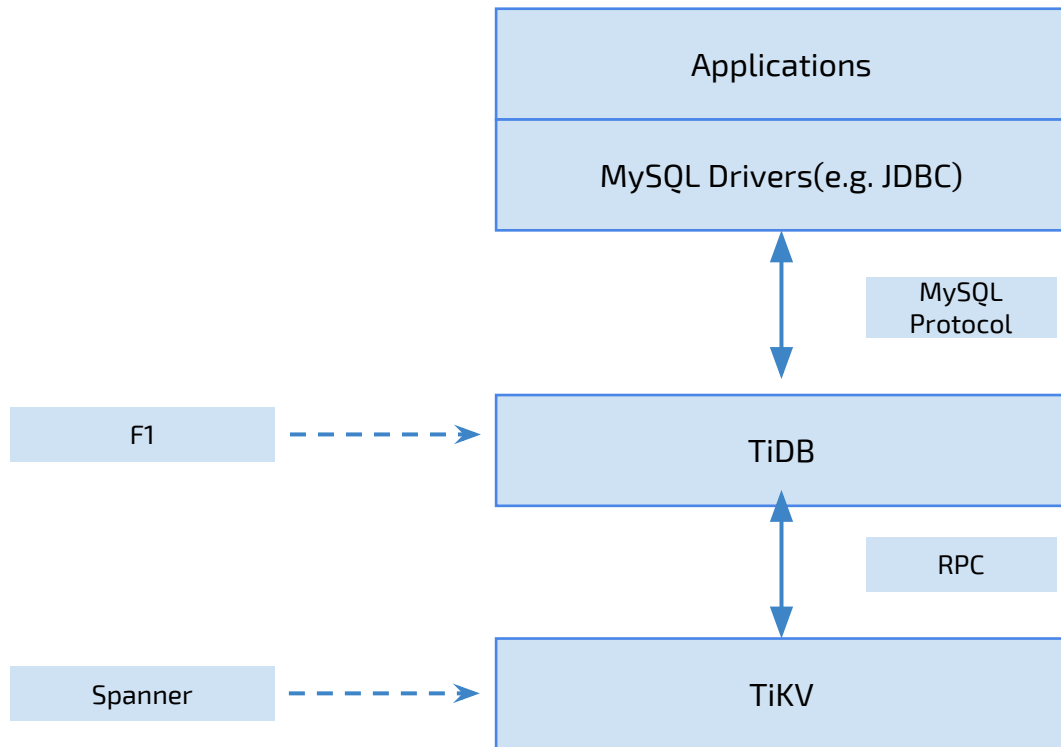
OR ...

- We just build a protocol layer that is **compatible with MySQL**. Then we have all the MySQL drivers.
 - All the tools
 - All the ORMs
 - All the applications
- That's what TiDB does.

Schema change in distributed RDBMS?

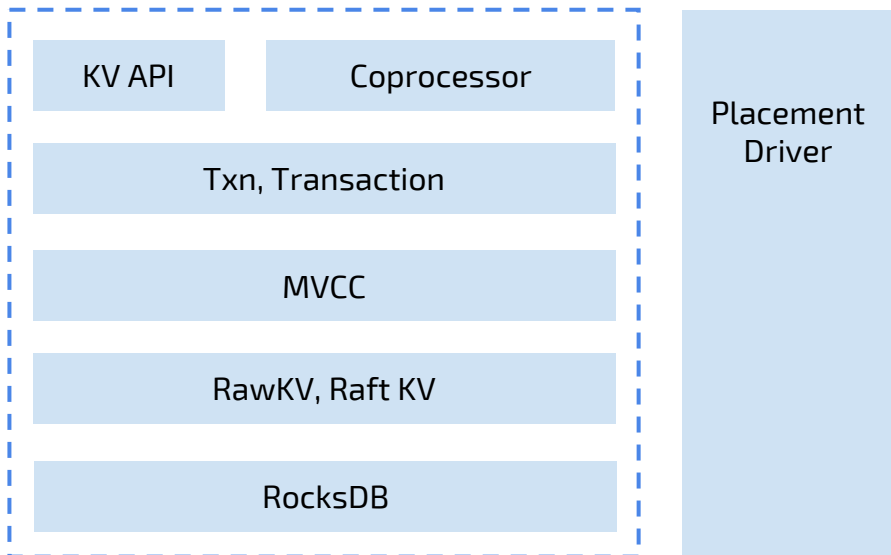
- A must-have feature!
- But you don't want to lock the whole table while changing schema.
 - Usually distributed database stores tons of data spanning multiple machines
- We need a **non-blocking** schema change algorithm
- Thanks F1 again
 - Similar to《Online, Asynchronous Schema Change in F1》- VLDB 2013 Google

Architecture

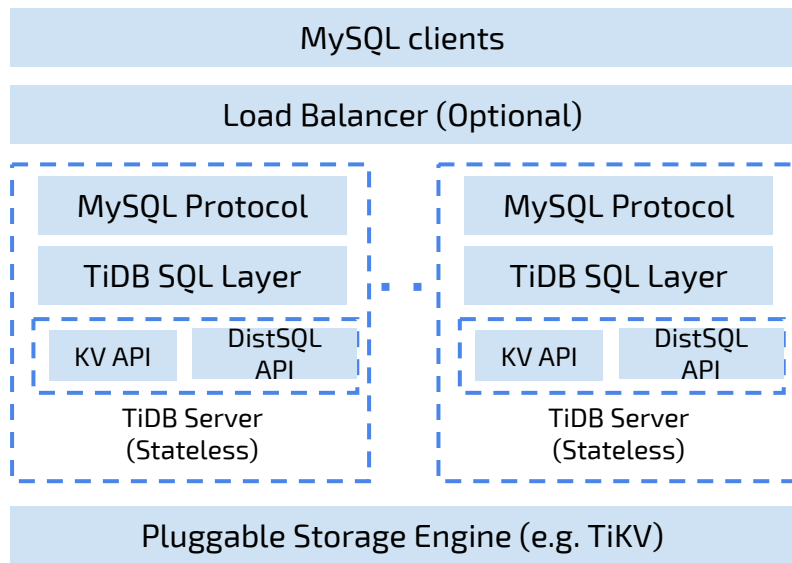


Architecture

TiKV



TiDB



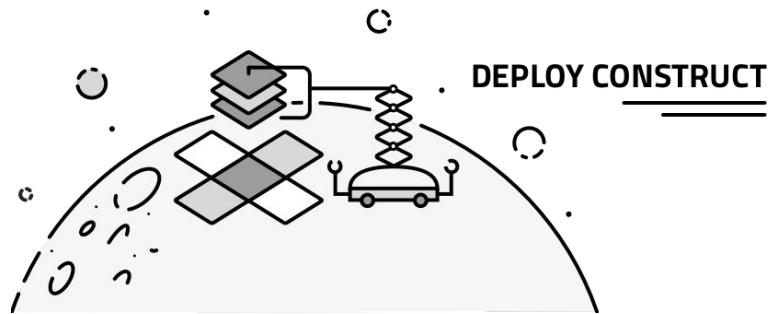
How to test

- Testing in distributed system is really hard
- Test-Driven Development
- Test cases from community
 - Lots of tests in MySQL drivers/connectors
 - Lots of ORMs
 - Lots of applications (Record---replay)
- Fault injection
 - Hardware: disk error, network card, cpu, clock
 - Software: file system, network and protocol
- Simulate everything : Network
- Distribute testing
 - Jepsen
 - Namazu

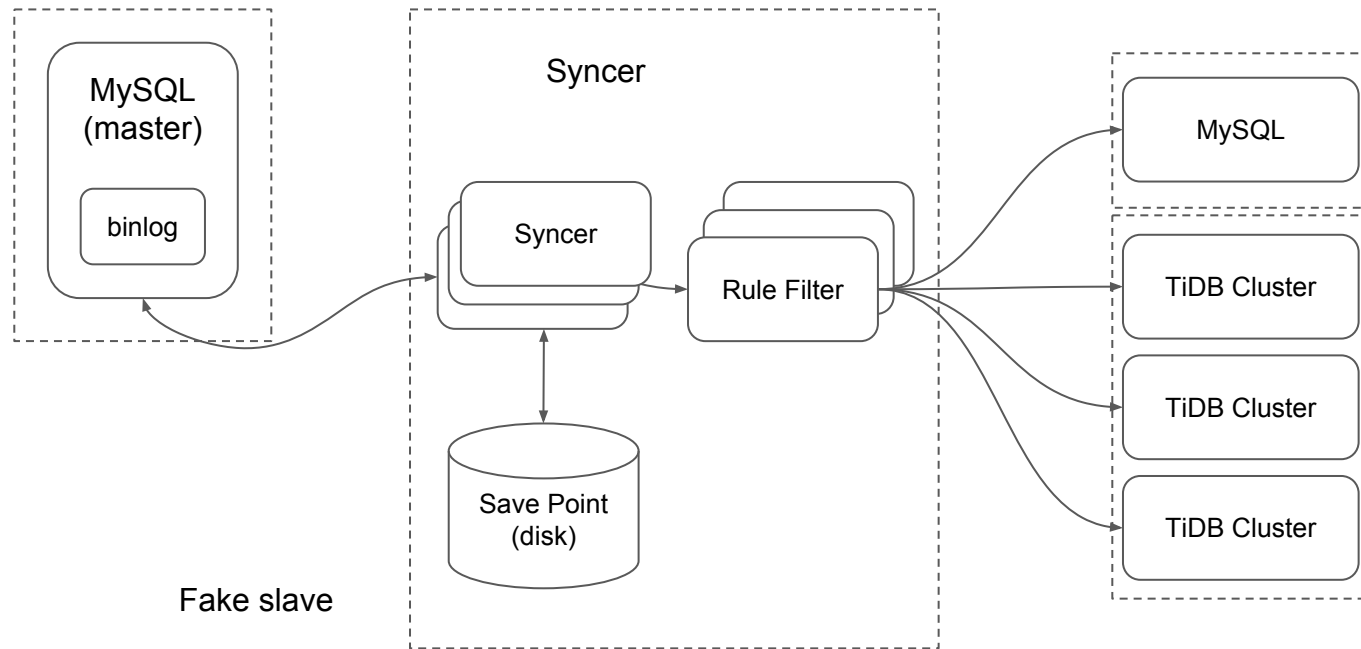
Tools matter

Make miracles happen

- Syncer
- TiDB-Binlog
- Backup & recovery tools

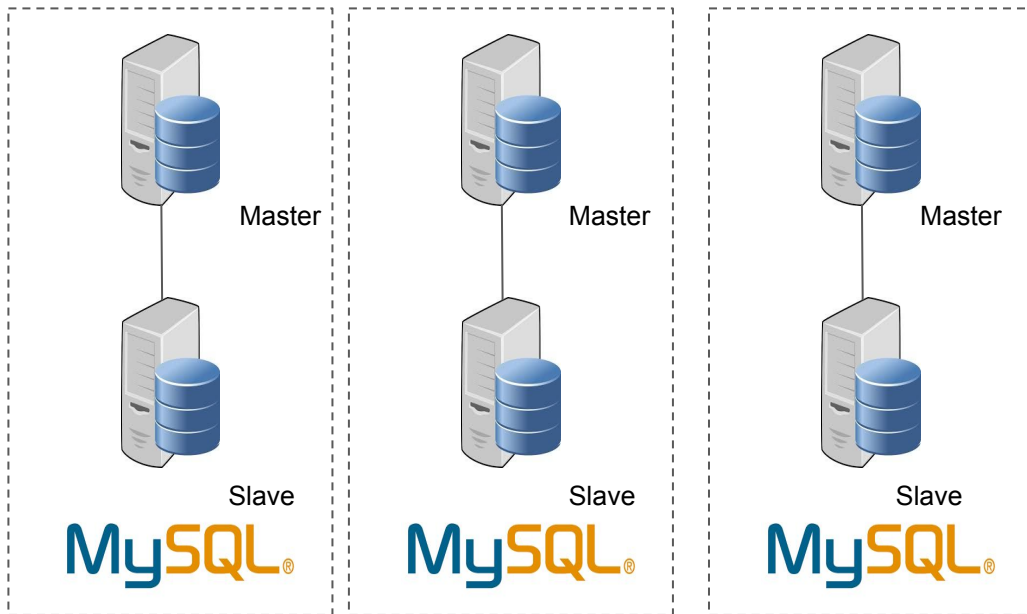


Syncer



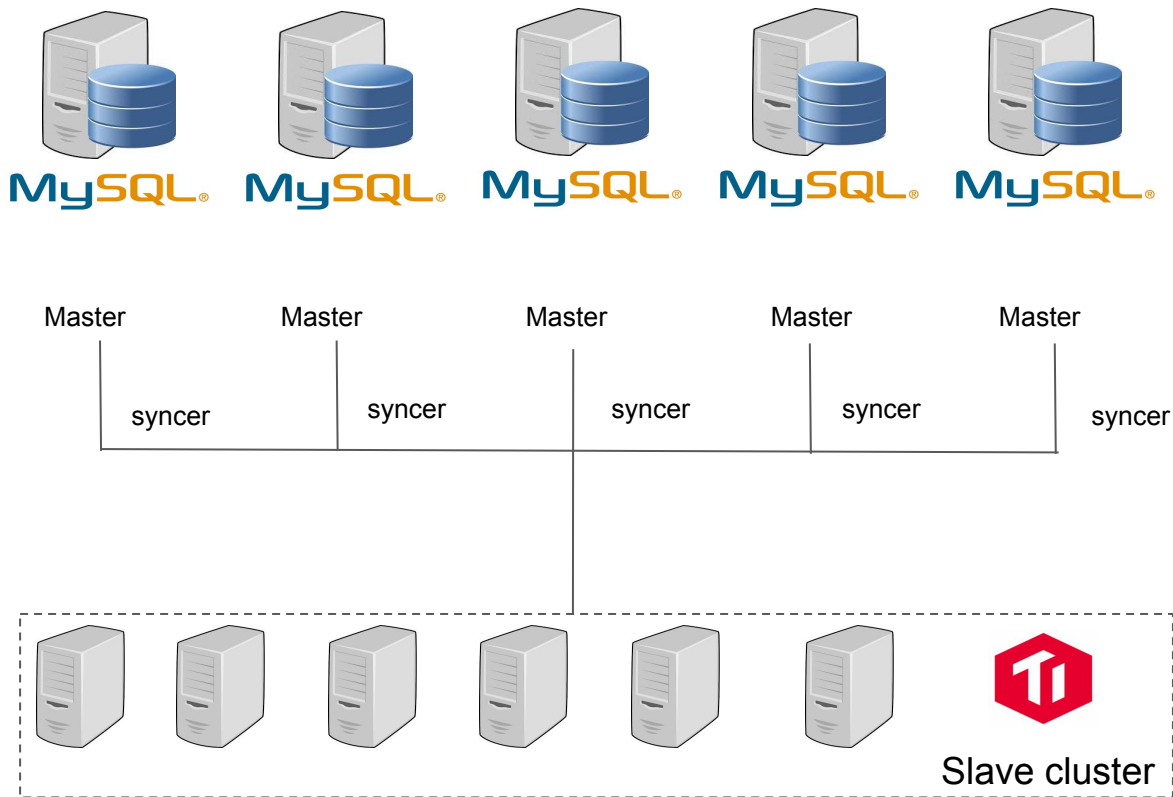
Syncer

Old days:



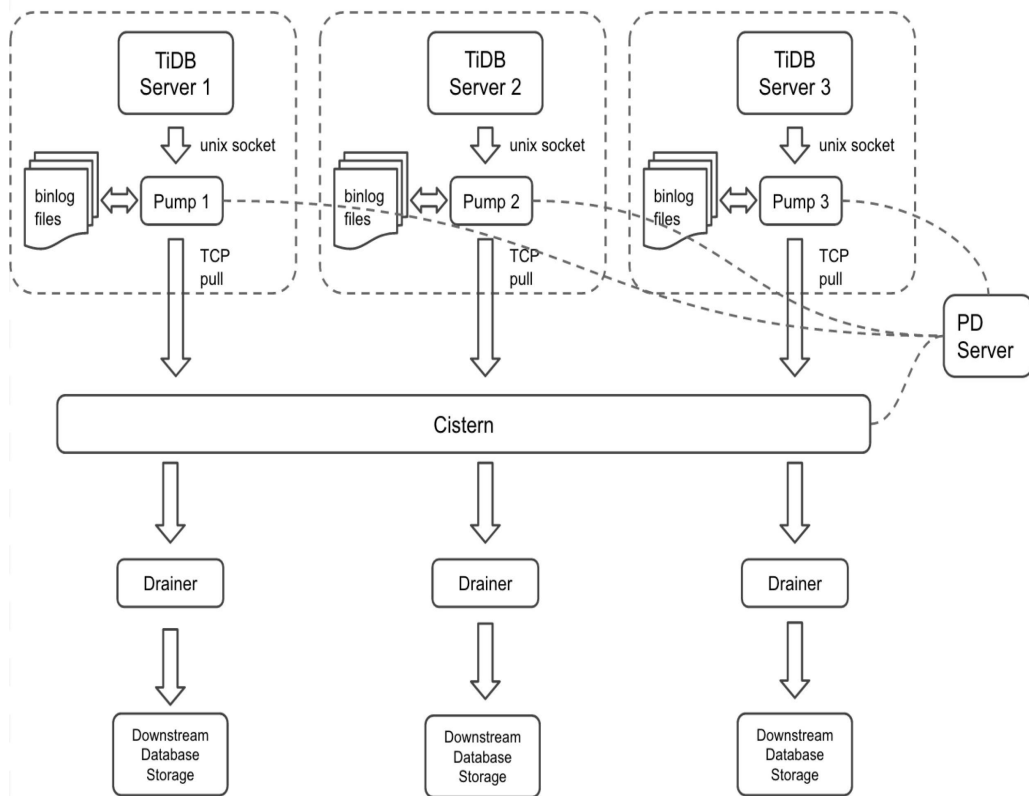
Syncer

Now:



TiDB-Binlog

- Describe change history of TiDB
 - DDL / DML
- Synchronize downstream data storage or database system
 - MySQL
 - Another TiDB cluster
 - Spark / Impala / ... (ongoing)
- Data migration
- Backup in real time



Backup & recovery tools

- mydumper
 - Multi-thread data import tool for MySQL
 - Faster than mysqldump
- Loader
 - Optimized for TiDB
 - Resume from checkpoint

Thanks

Q&A

<https://github.com/pingcap/tidb>

<https://github.com/pingcap/tikv>

EASE OF USE

