PingCAP

How do we build TiDB

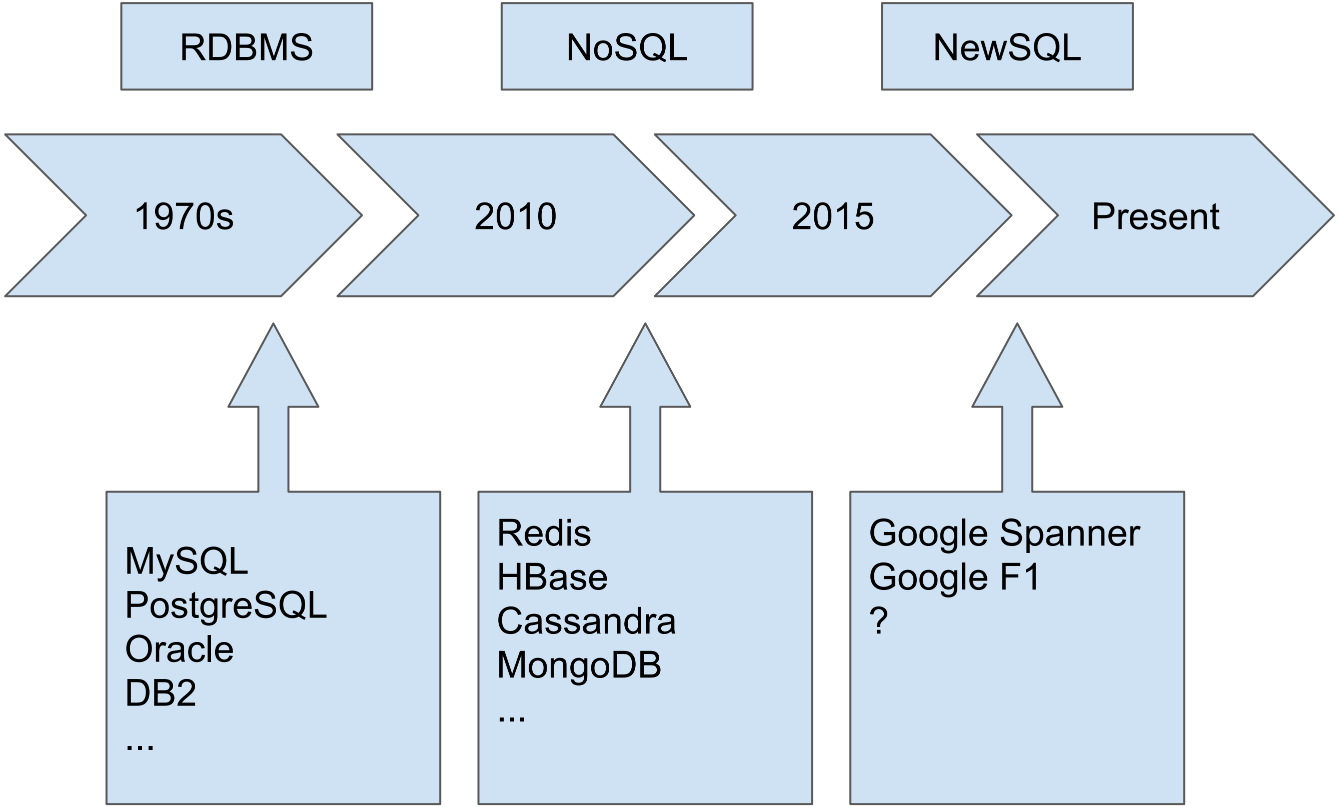
A Distributed, Consistent, Scalable, SQL Database

About me

- LiuQi (刘奇)
- JD / WandouLabs / PingCAP
- Co-founder / CEO of PingCAP
- Open-source hacker / Infrastructure software engineer
- Codis / TiDB / TiKV
- Twitter: @ngaut1
- Weibo: @goroutine
- Email: liuqi@pingcap.com

Topics discussed in the slides...

- A brief history of database
- Why do we need another database
- Google Spanner / F1
- TiDB and TiKV
    - Architecture overview
    - TiKV Overview
    - Database based on Raft
    - Scale-out / Replication / Failover
    - MVCC
    - Transaction model
    - Requests routing
- Typical scenario:
    - MySQL Sharding
    - Cross-datacenter HA



RDBMS

NoSQL

NewSQL

1970s

2010

2015

Present

MySQL
PostgreSQL
Oracle
DB2
...

Redis
HBase
Cassandra
MongoDB
...

Google Spanner
Google F1
?

Why do we need another database

Applications nowadays...

- Workload
- The amount of data
- Complexity

What do we expect from a database:

- Scalability
- SQL
- ACID Transaction
- High Availability / Auto-Failover

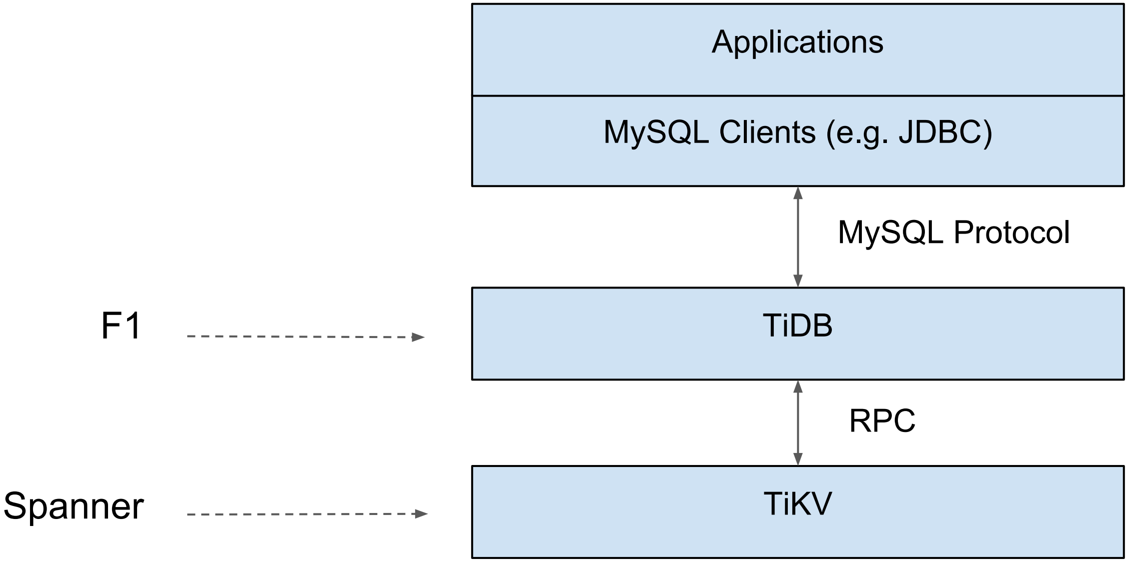
Google Spanner / F1

- Globally-distributed
  - Paxos
- SQL above NoSQL
- ACID Transaction support
  - TrueTime API
- Designed for Google AdWords, originally
  - became the successors of BigTable

TiDB and TiKV

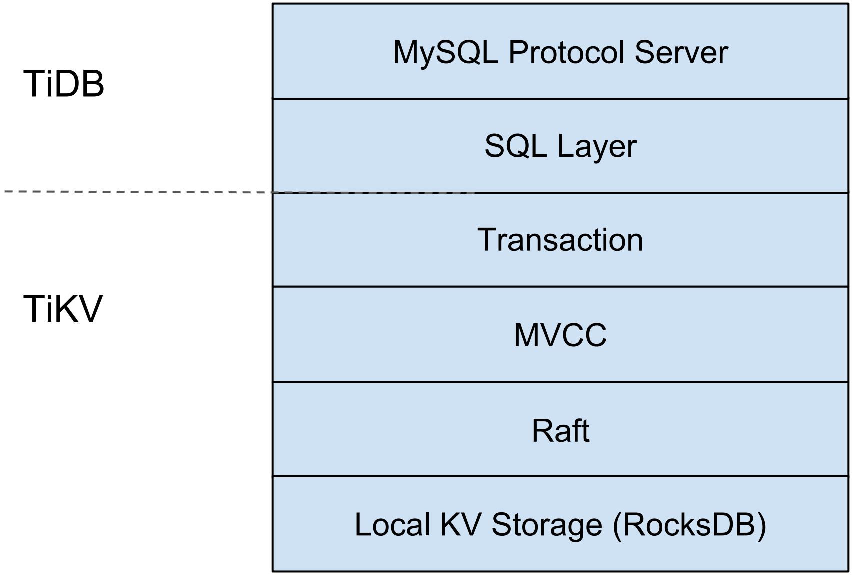
- Building NewSQL outside Google
- Of course, it’s open-sourced.
  - https://github.com/pingcap/tikv
  - https://github.com/pingcap/tidb
- 「You can no longer win with a closed-source platform.」

Architecture overview (Software)



Architecture overview (Logical)

- Highly layered
- Separate SQL and Key-Value layers
- Using Raft for consistency and scaling
- Without a distributed file system



TiDB

MySQL Protocol Server

SQL Layer

TiKV

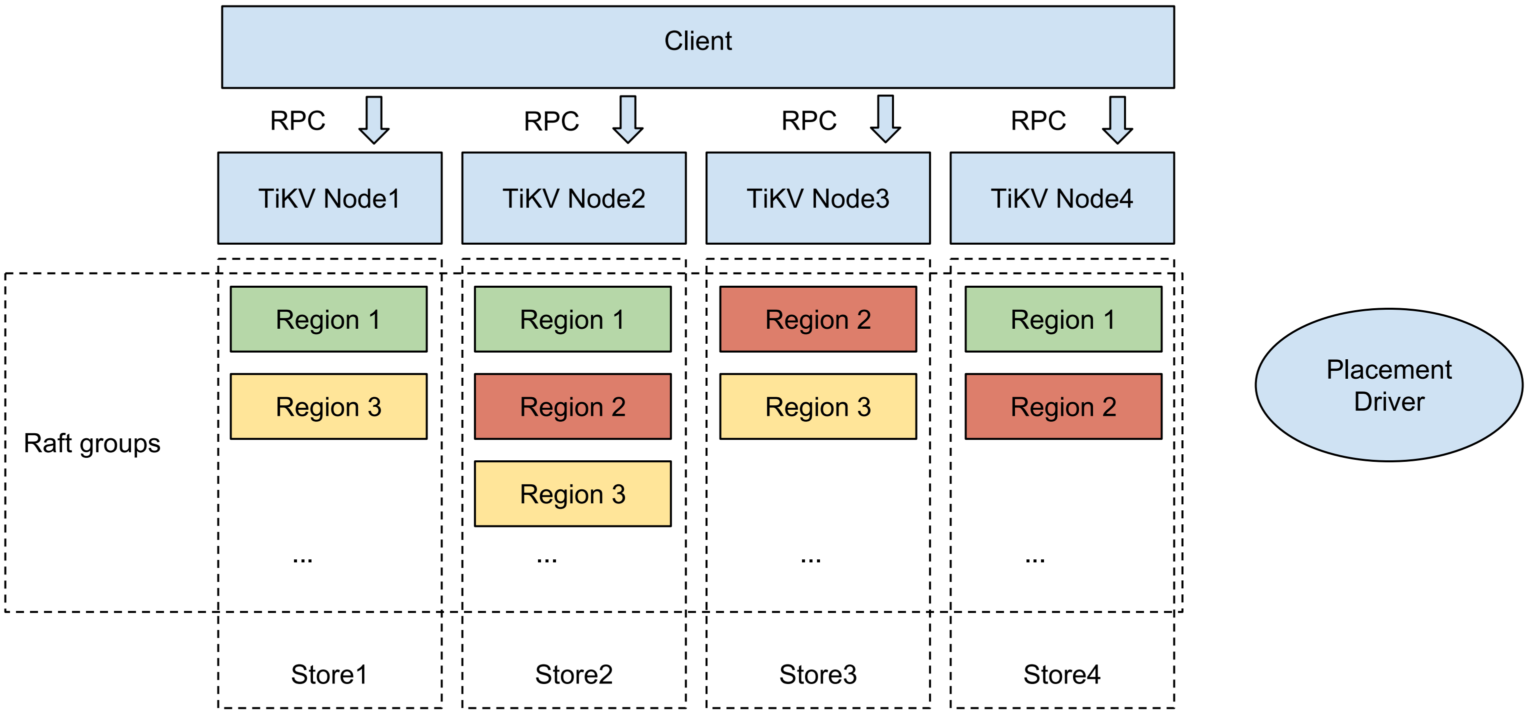
Transaction

MVCC

Raft

Local KV Storage (RocksDB)

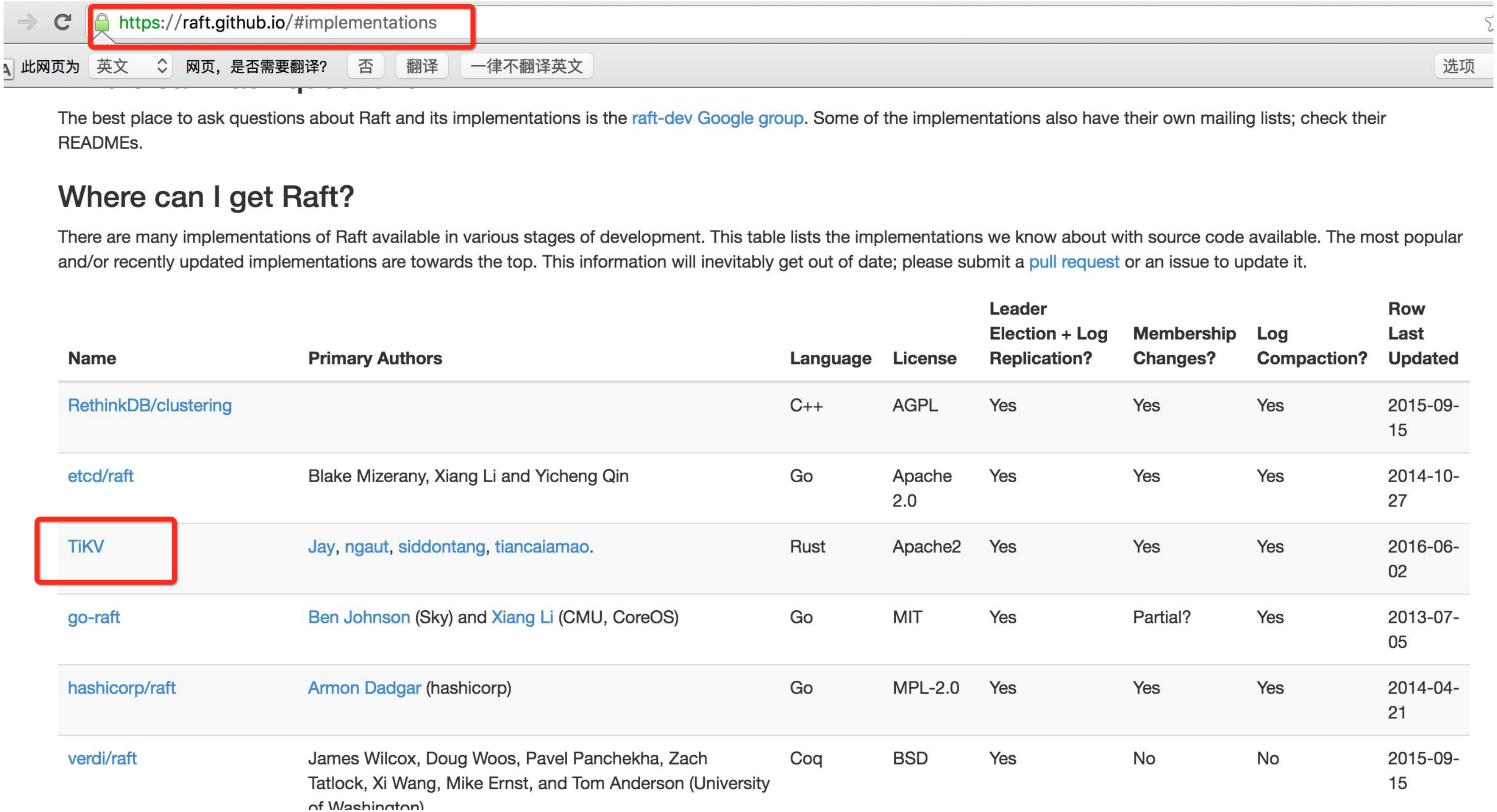
TiKV Overview



Database based on Raft

- Data is organized in **Regions**
- The replicas of one region is 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 splitted into two smaller regions
    - Just like cell division

Implementations of Raft


https://raft.github.io/#implementations
此网页为 英文 网页，是否需要翻译？ 否 翻译 一律不翻译英文 选项

The best place to ask questions about Raft and its implementations is the raft-dev Google group. Some of the implementations also have their own mailing lists; check their READMEs.

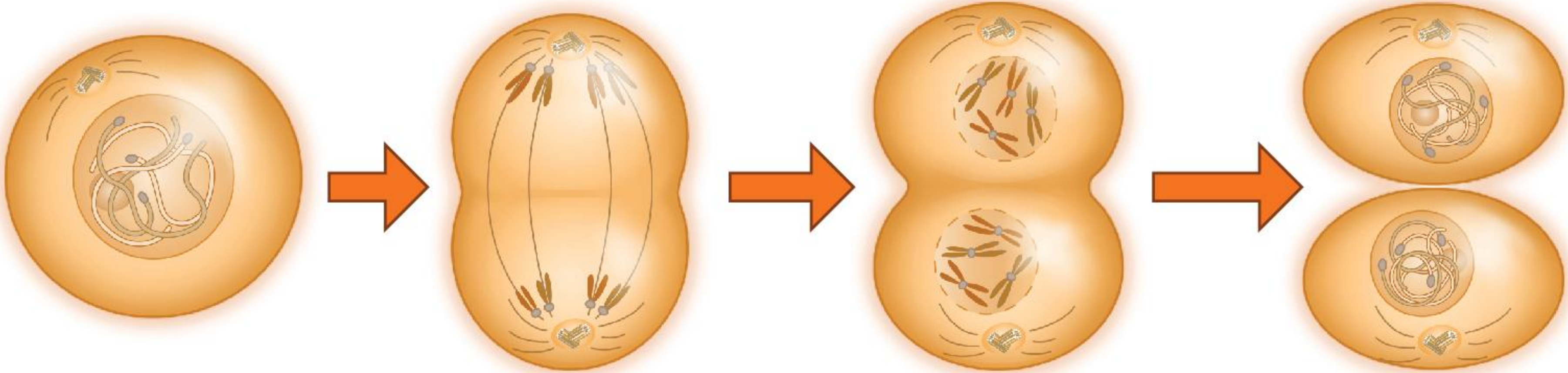
Where can I get Raft?

There are many implementations of Raft available in various stages of development. This table lists the implementations we know about with source code available. The most popular and/or recently updated implementations are towards the top. This information will inevitably get out of date; please submit a pull request or an issue to update it.

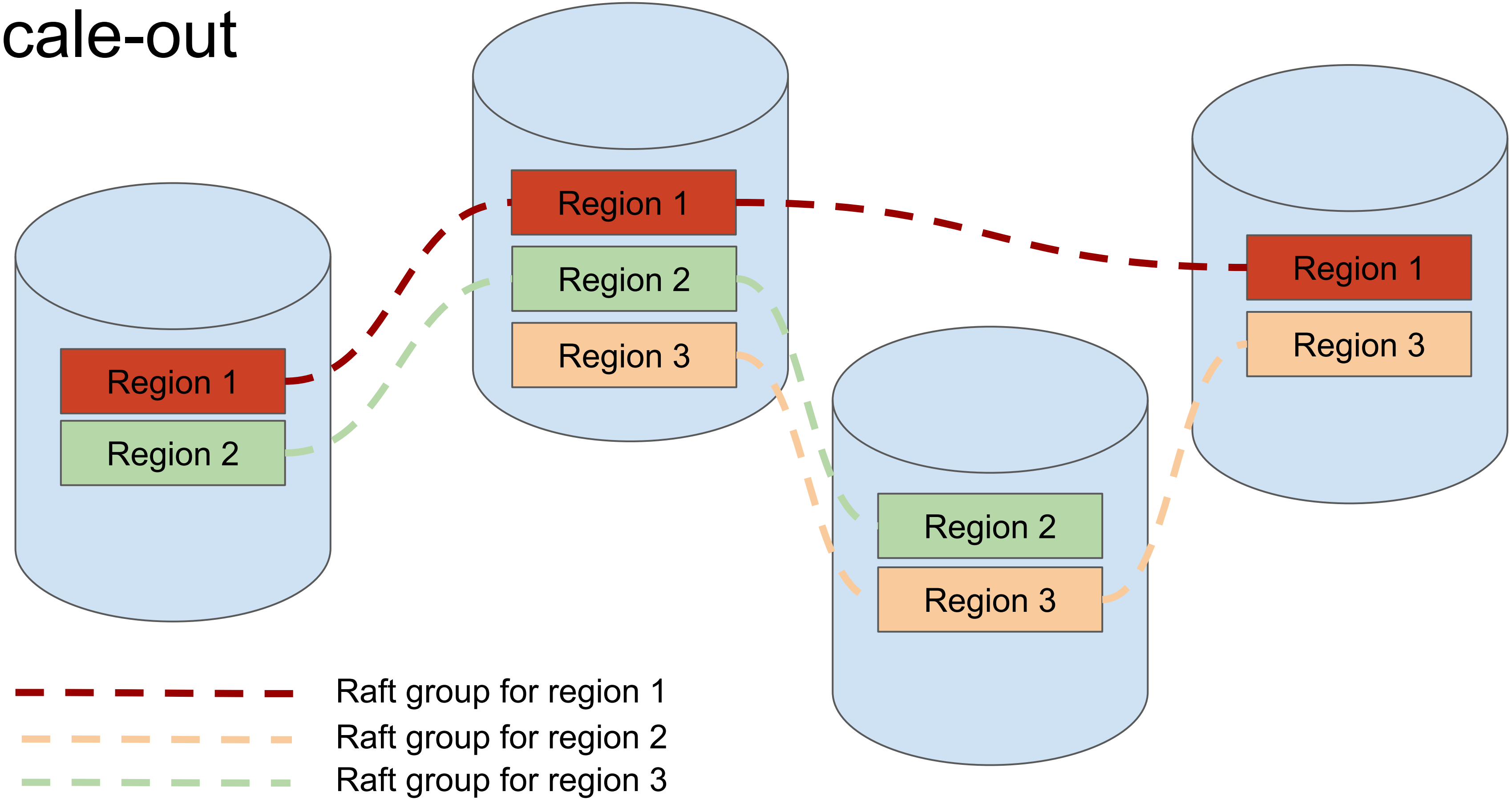
Name	Primary Authors	Language	License	Leader Election + Log Replication?	Membership Changes?	Log Compaction?	Row Last Updated
RethinkDB/clustering		C++	AGPL	Yes	Yes	Yes	2015-09-15
etcd/raft	Blake Mizerany, Xiang Li and Yicheng Qin	Go	Apache 2.0	Yes	Yes	Yes	2014-10-27
TiKV	Jay, ngaut, siddontang, tiancaiamao.	Rust	Apache2	Yes	Yes	Yes	2016-06-02
go-raft	Ben Johnson (Sky) and Xiang Li (CMU, CoreOS)	Go	MIT	Yes	Partial?	Yes	2013-07-05
hashicorp/raft	Armon Dadgar (hashicorp)	Go	MPL-2.0	Yes	Yes	Yes	2014-04-21
verdi/raft	James Wilcox, Doug Woos, Pavel Panchekha, Zach Tatlock, Xi Wang, Mike Ernst, and Tom Anderson (University of Washington)	Coq	BSD	Yes	No	No	2015-09-15

PingCAP

Cell division



Scale-out



- ▬ ▬ ▬ Raft group for region 1
- ▬ ▬ ▬ Raft group for region 2
- ▬ ▬ ▬ Raft group for region 3

Scale-out

DEMO

Replication / Failover

- 3 Replicas, by default.
- Raft consensus algorithm will ensure:
  - When the existing leader is down, the remaining servers will run an election to elect a new leader
  - The new leader always contains the newest committed logs
- Replicas are distributed across datacenters

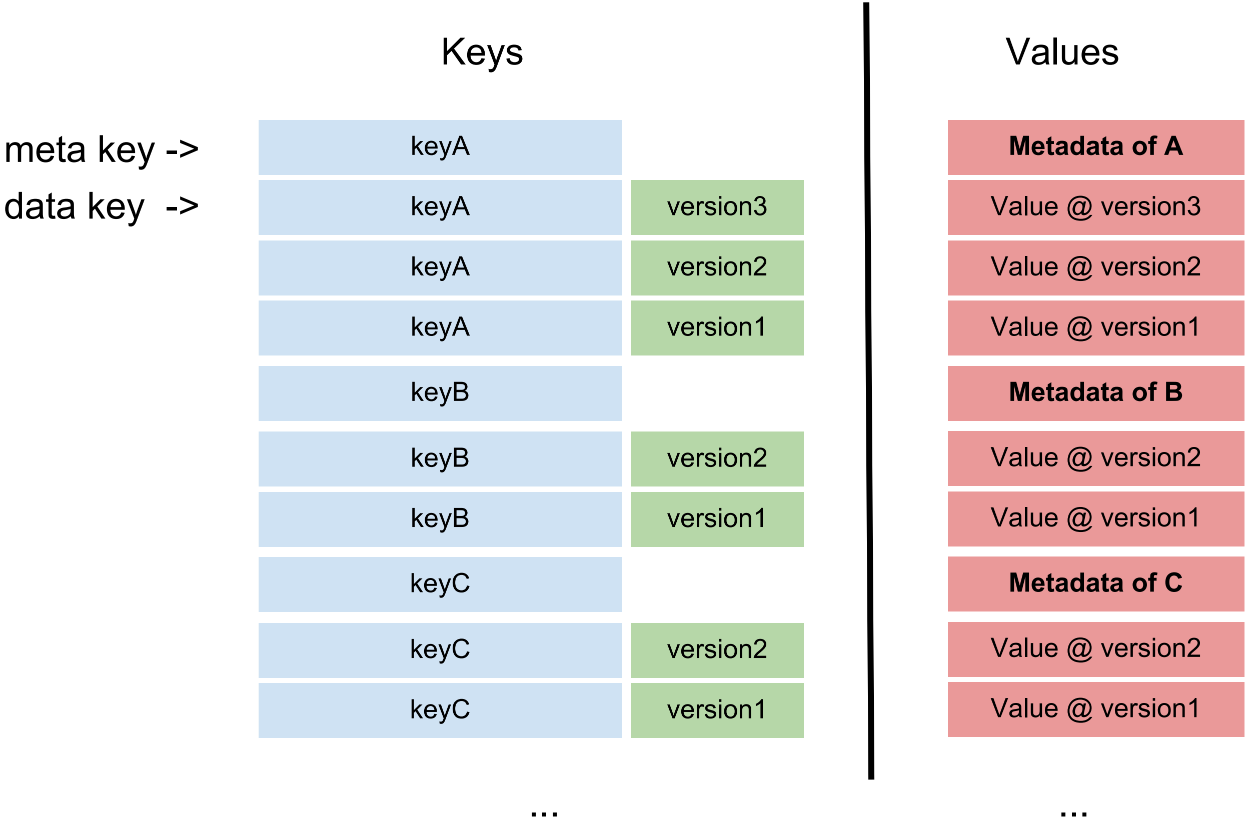
Replication / Failover

Just remember:

- **NEVER LOSE ANY DATA**
- The whole failover process is **AUTOMATIC**

MVCC

- **M**ulti-**V**ersion **C**oncurrency **C**ontrol
- Lock-free snapshot read
- Isolation of transaction
- GC Problem



	Keys		Values
meta key ->	keyA		**Metadata of A**
data key ->	keyA	version3	Value @ version3
	keyA	version2	Value @ version2
	keyA	version1	Value @ version1
	keyB		**Metadata of B**
	keyB	version2	Value @ version2
	keyB	version1	Value @ version1
	keyC		**Metadata of C**
	keyC	version2	Value @ version2
	keyC	version1	Value @ version1
	...		...

in RocksDB

Transaction model

- Based on Google Percolator
    - with some practical optimizations
- 2PL / 2PC

Requests routing

- TiClient
- Placement Driver

Typical scenario: MySQL Sharding

Trait:

- High throughput
- Huge amounts of concurrent small transactions
- Always have a sharding key
- No complex query

Pain:

- Scaling and DDL
- Lack of cross-shard transactions

Typical scenario: MySQL Sharding

What TiDB can do:

- Online DDL
- Elastic scalability and workload auto-balancing
- Distributed ACID transaction support

Typical scenario: Cross-datacenter HA

Trait:

- Data is extremely critical
- Zero downtime, even when one of the datacenters is down
- Multi-master architecture

Pain:

- No database nowadays provides these features!
- Master-slave model is slow and not so reliable
- Manual recovery is error-prone

Typical scenario: Cross-datacenter HA

What TiDB can do:

- Data is safely replicated by Raft across datacenters
- The latency depends on the communication latency of the majority datacenters.
  - e.g. There are 3 replicas, 2 of them in Beijing, the other one in Hong Kong. When a write request is replicated by the 2 nodes in Beijing, it’s committed safely.
- Even if less than N/2 nodes fail, the system is still alive.

How to contribute?

BETA

github.com/pingcap/tidb

github.com/pingcap/tikv

github.com/pingcap/pd

Thanks

TiDB

A Distributed SQL Database

https://github.com/pingcap/tidb