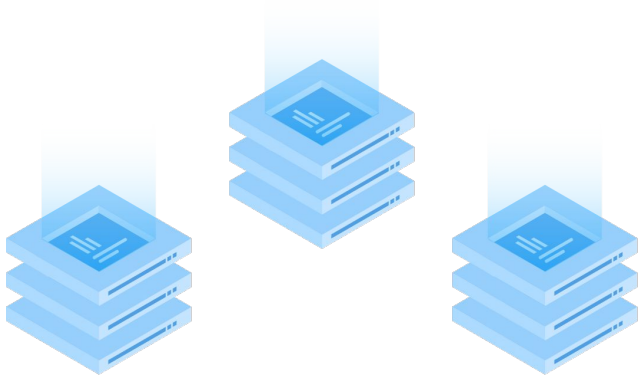
TiKV Best Practices

Presented by Jinpeng Zhang

PingCAP TiDB



About me

- Engineer @PingCAP
- TiKV senior maintainer
- Author of book <Principle and Implementation of MariaDB>
- 10 years experience on Storage engine & System Performance

Agenda

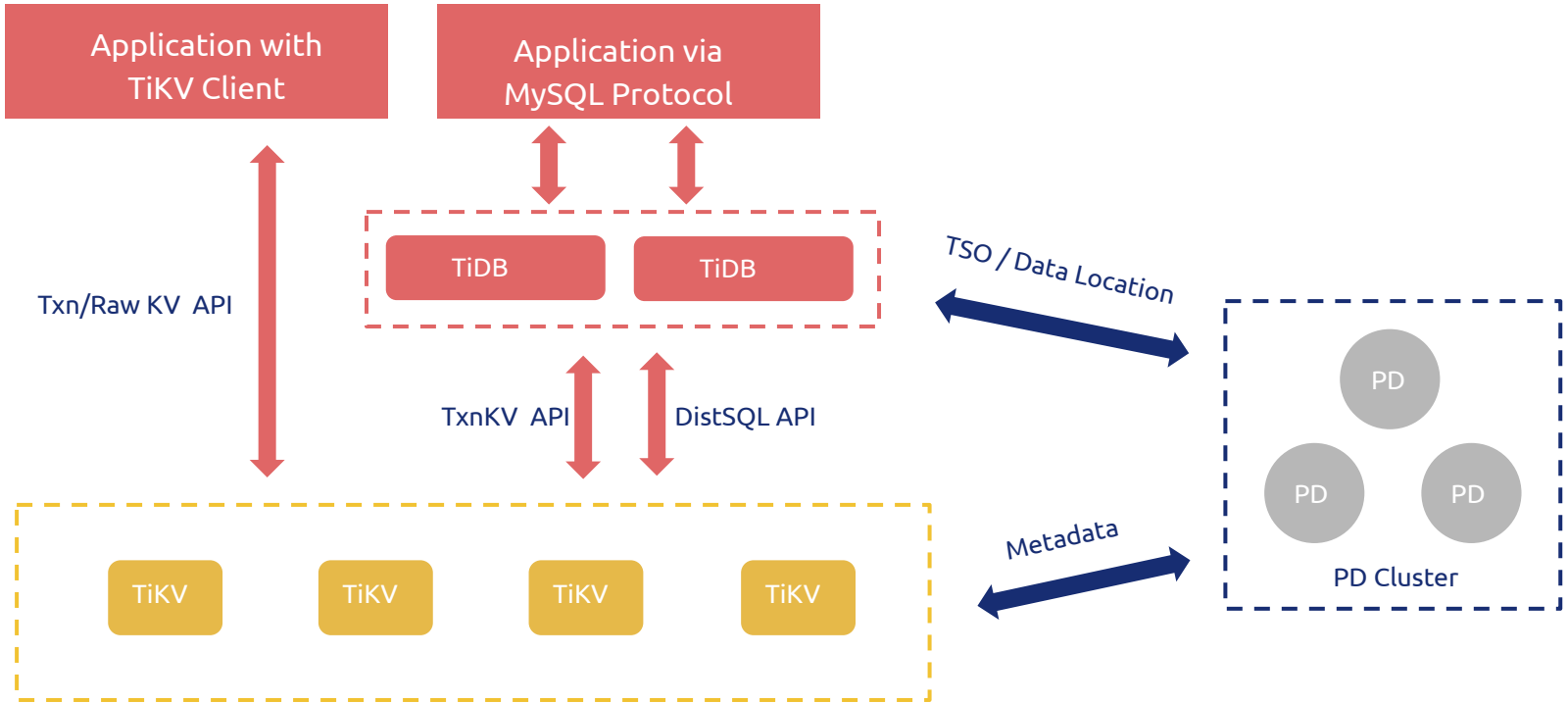
- Theories
  - The topology of a TiKV cluster
  - Multi-Raft
  - Scale
  - Ecosystem
- Practices
  - Deployment
  - Elasticly scale
  - Fight with hotspot
  - Performance tuning

1st Part: Theories

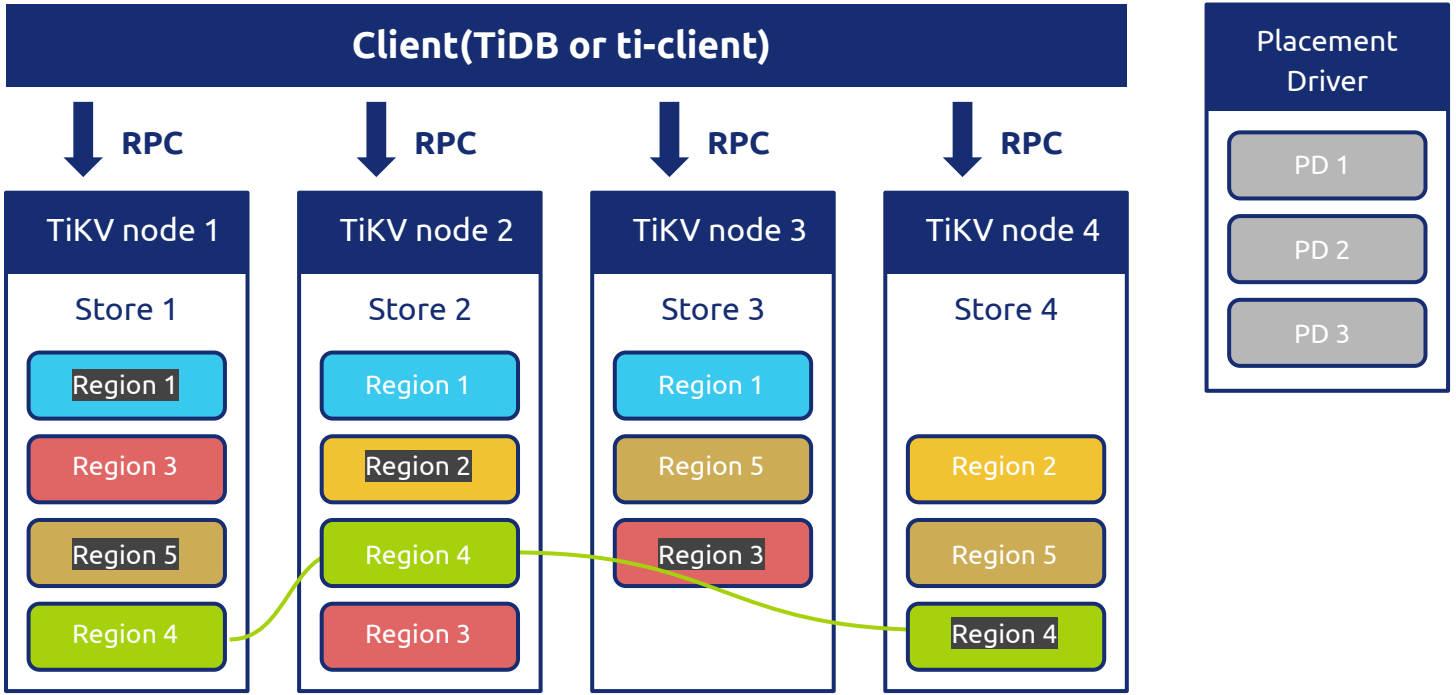
PingCAP

1. The topology of a TiKV cluster
2. Multi-Raft
    a. Region Split
    b. Region Merge
3. Scale
    a. Transfer Leader
    b. Replication and balancing
4. Ecosystem
    a. gRPC-rs
    b. raft-rs
    c. golang/rust/c clients

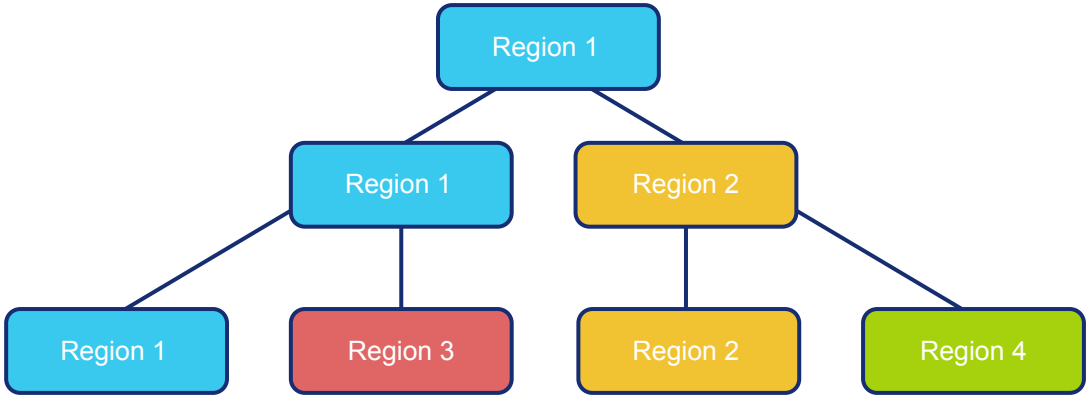
1 The topology of a TiKV cluster



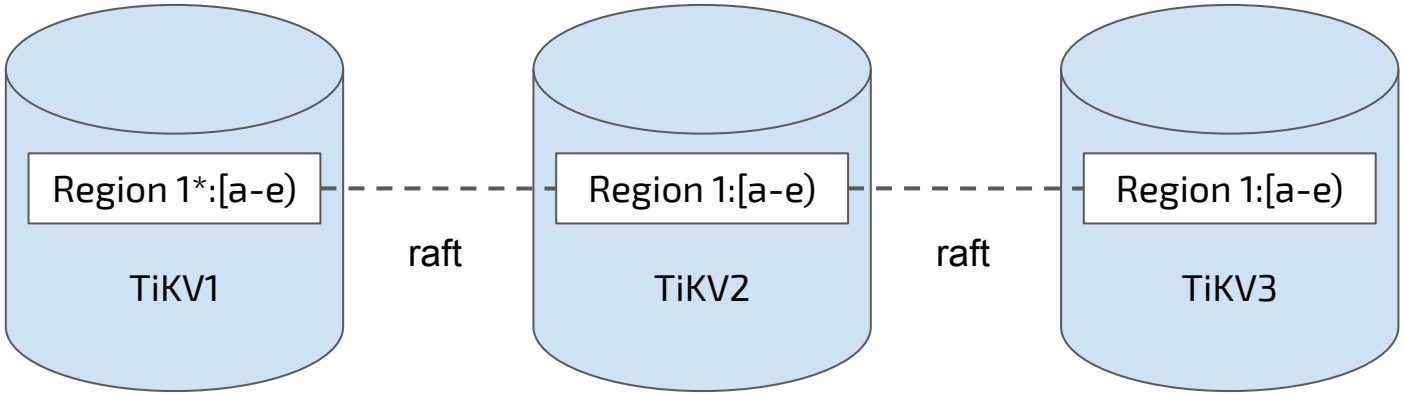
2 Multi-Raft


Client(TiDB or ti-client)
RPC
RPC
RPC
RPC
TiKV node 1
TiKV node 2
TiKV node 3
TiKV node 4
Store 1
Store 2
Store 3
Store 4
Region 1
Region 3
Region 5
Region 4
Region 1
Region 2
Region 4
Region 3
Region 1
Region 5
Region 3
Region 2
Region 5
Region 4
Placement Driver
PD 1
PD 2
PD 3

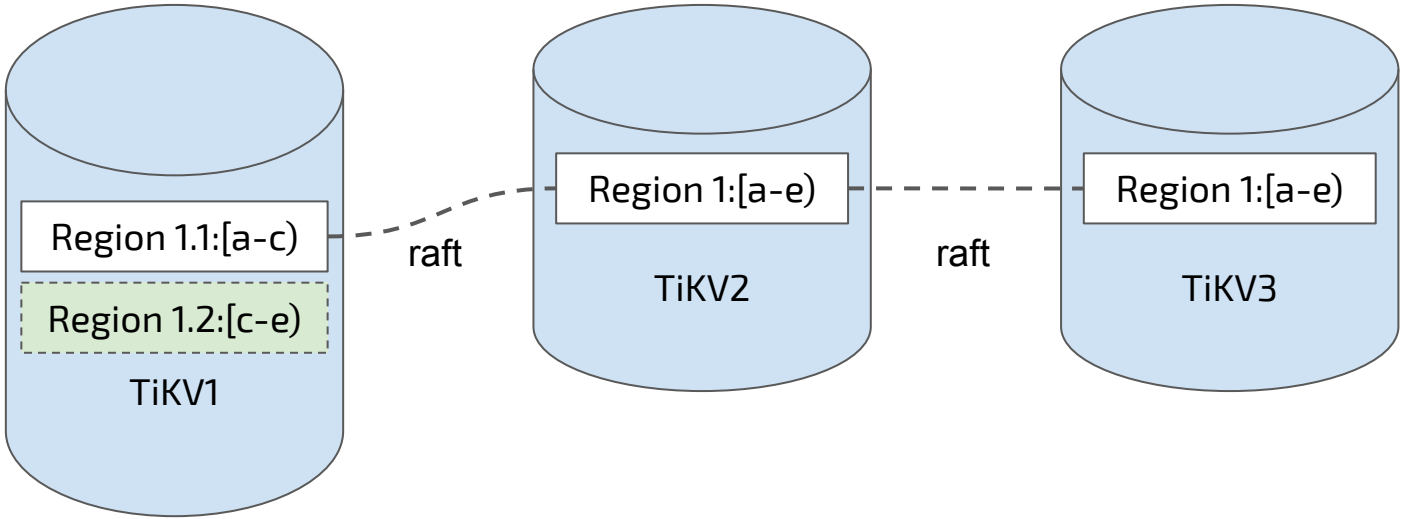
2.1 Multi-Raft - Split



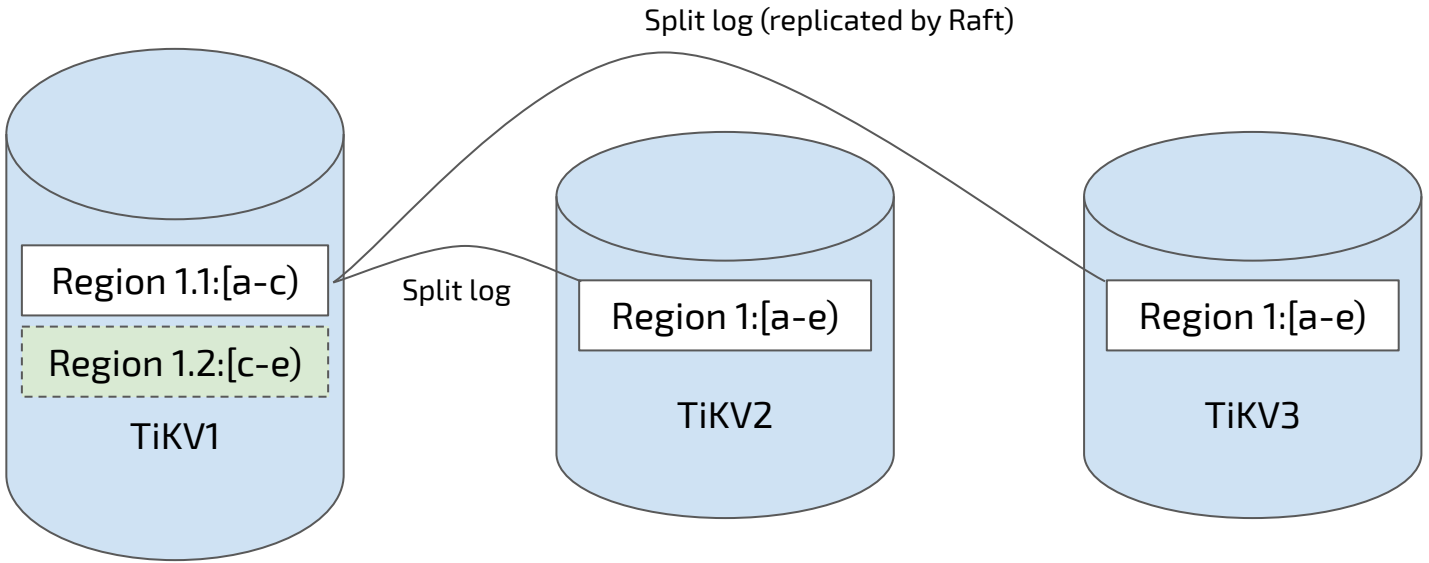
2.1 Multi-Raft - Split



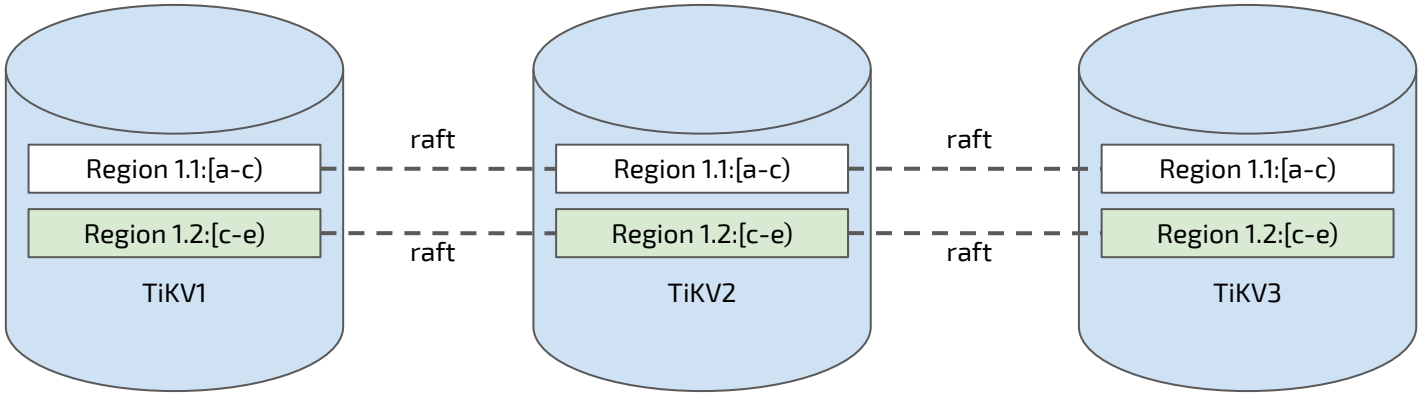
2.1 Multi-Raft - Split



2.1 Multi-Raft - Split

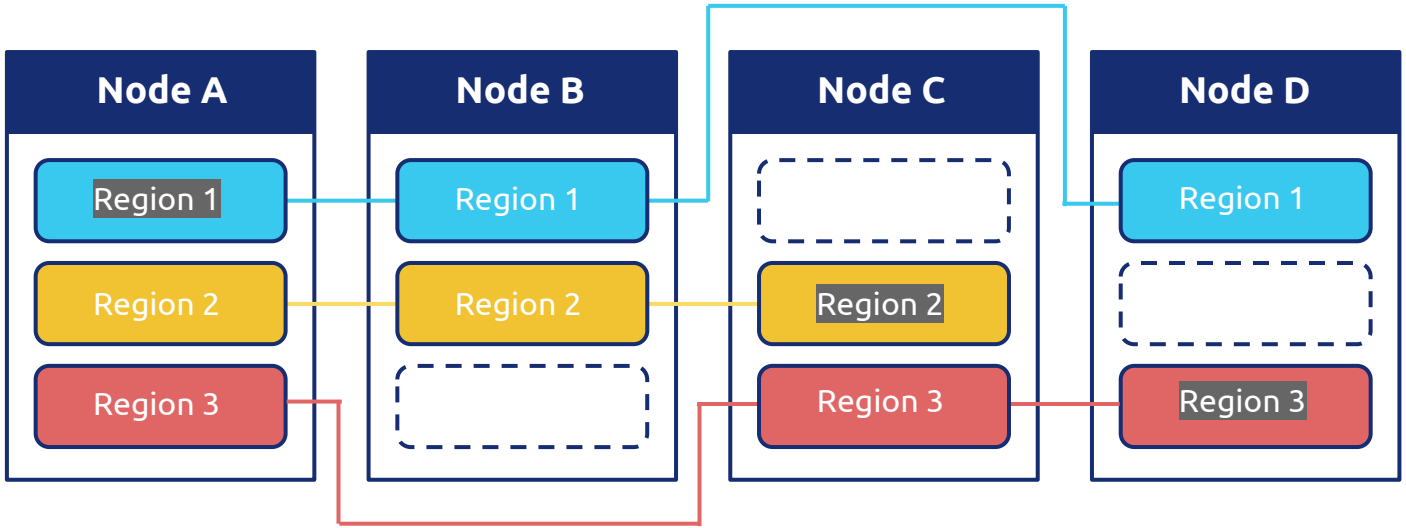


2.1 Multi-Raft - Split



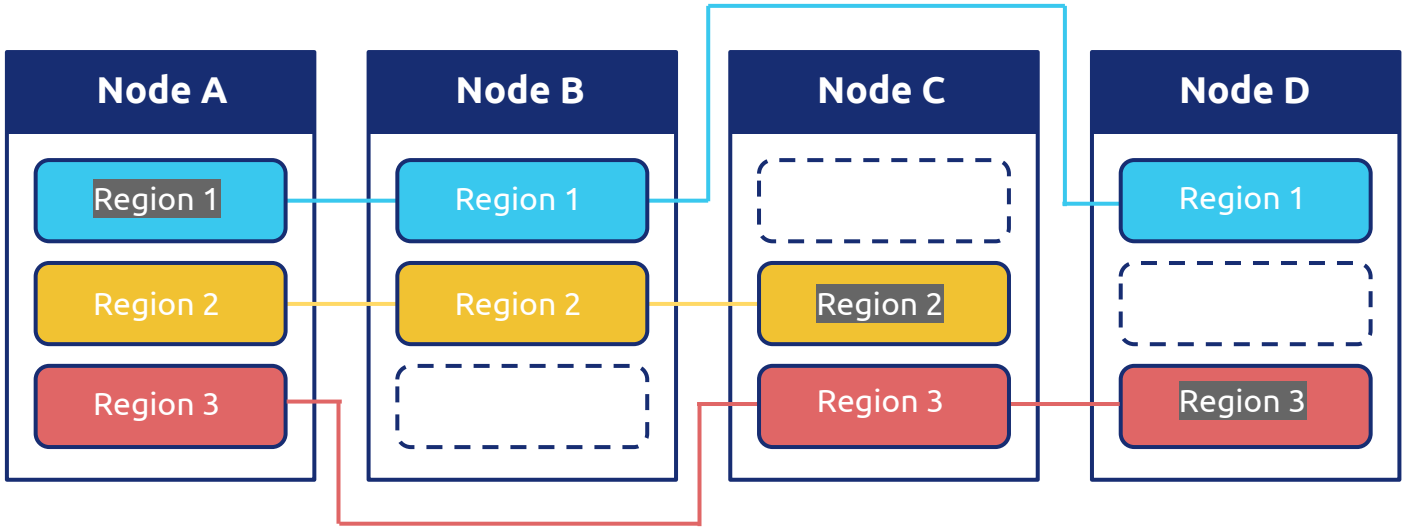
3 Scale

- 3 replicas for each region
- 1 leader and 2 followers for each region



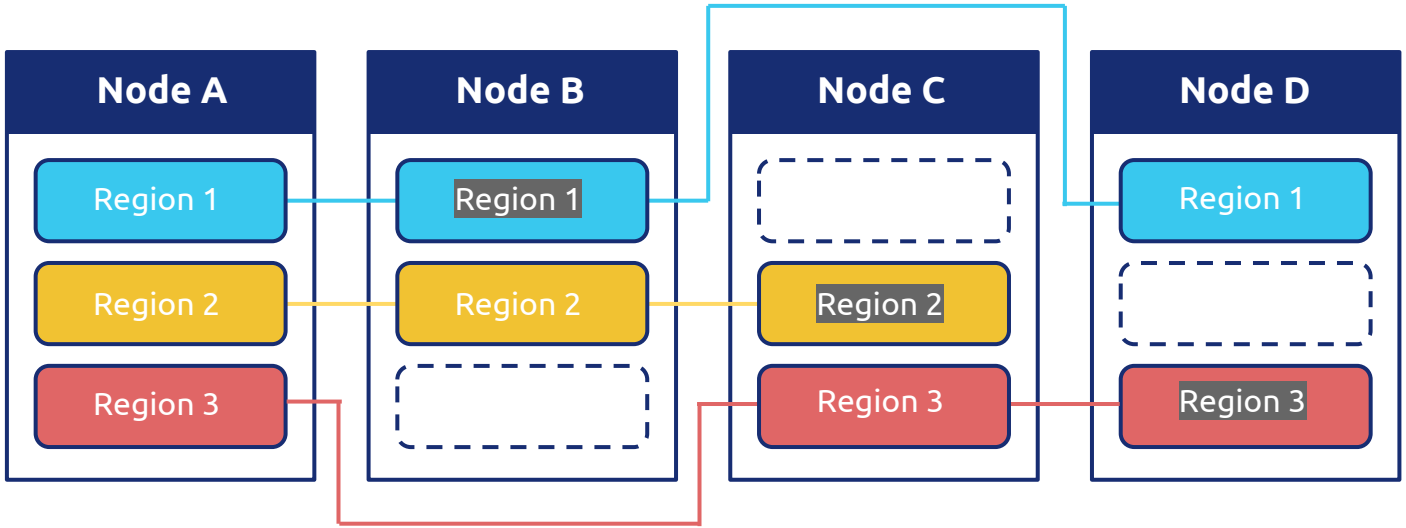
3.1 Scale - Transfer leader

- Region 1's leader is on Node A

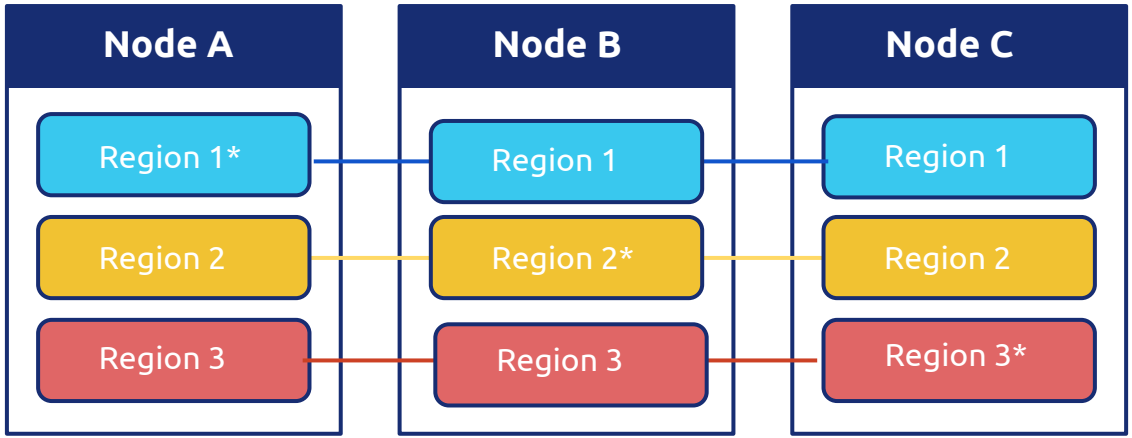


3.1 Scale - Transfer leader

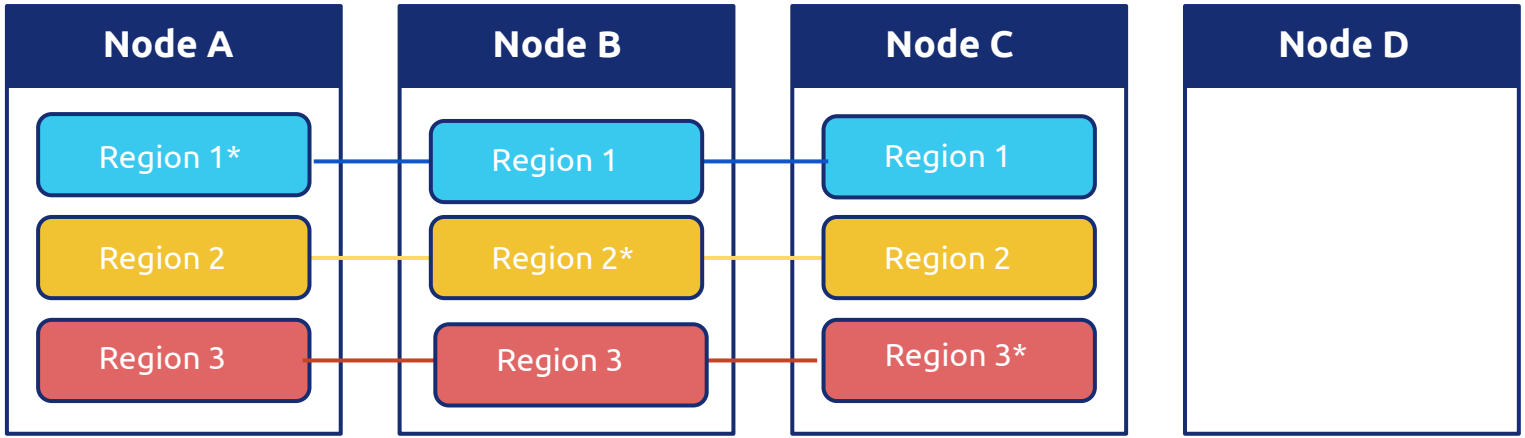
- Transfer to Node B



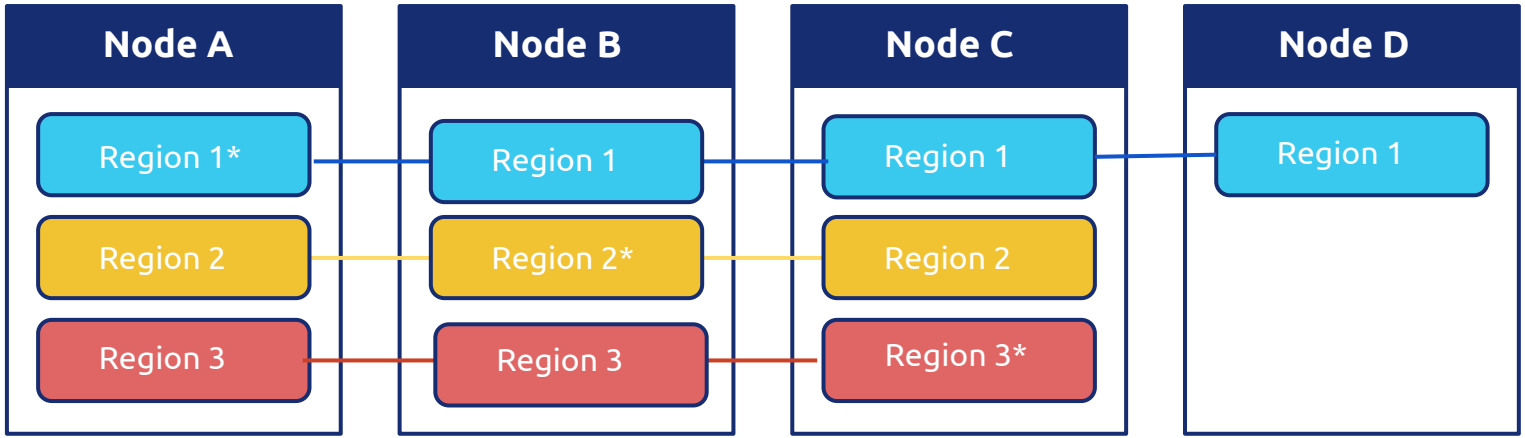
3.2 Scale - Initial state



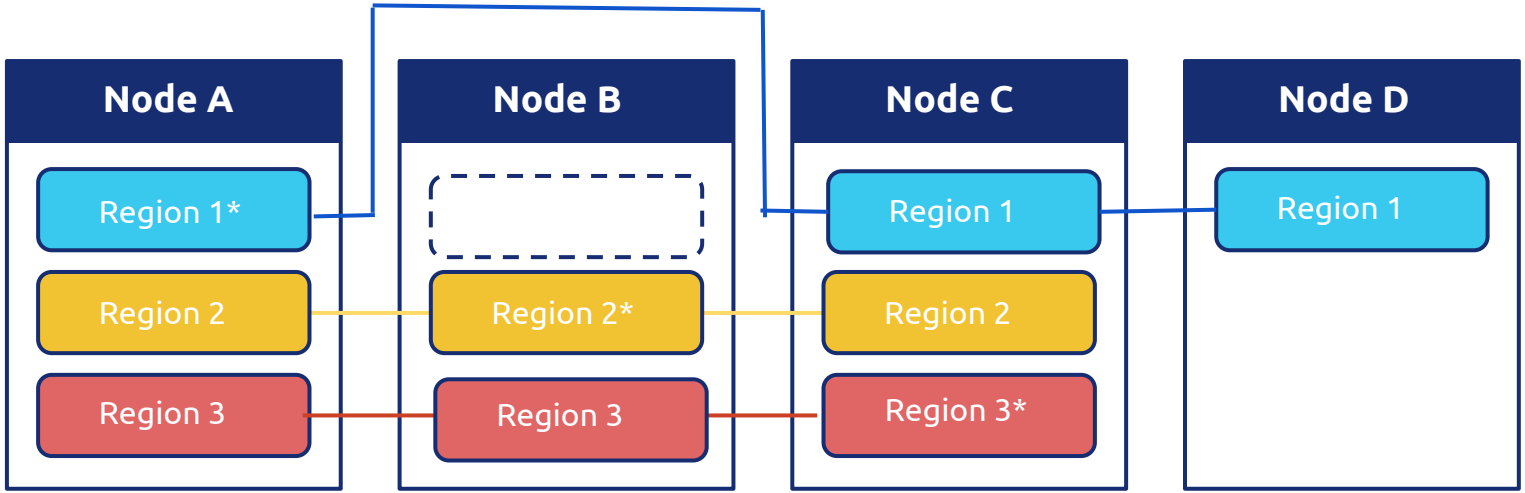
3.2 Scale - Add a new node



3.2 Scale - Add a replica in new node



3.2 Scale - Remove a replica in old node



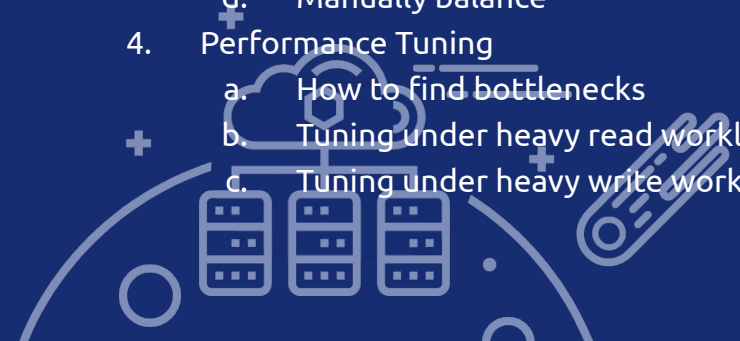
4 Ecosystem

- Rust wrapper or gRPC https://github.com/pingcap/grpc-rs
- Raft Implemetation in Rust https://github.com/pingcap/raft-rs
- Clients
  - https://github.com/tikv/client-go
  - https://github.com/tikv/client-rust
  - https://github.com/tikv/client-java

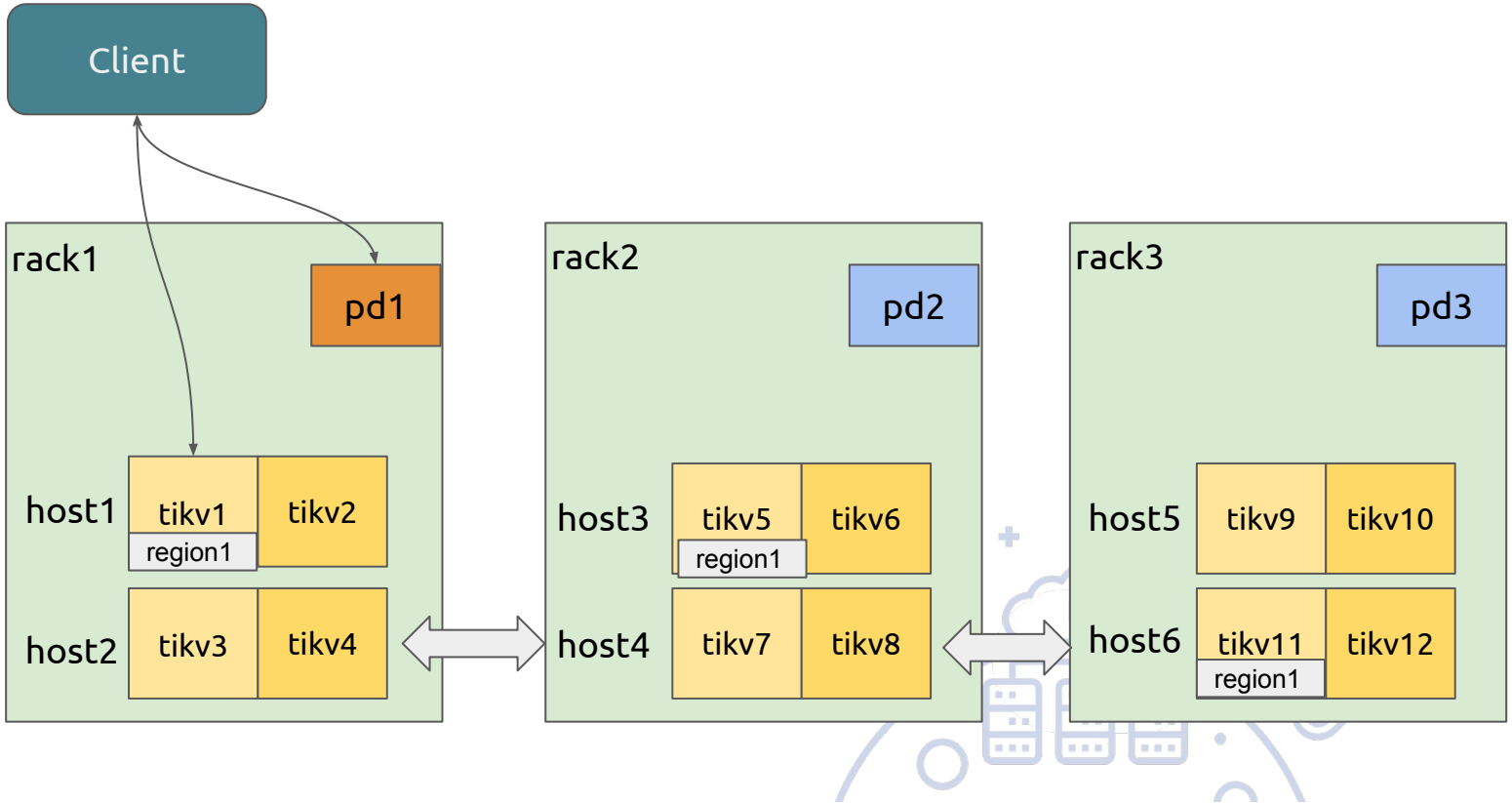
2rd Part: Practices

PingCAP

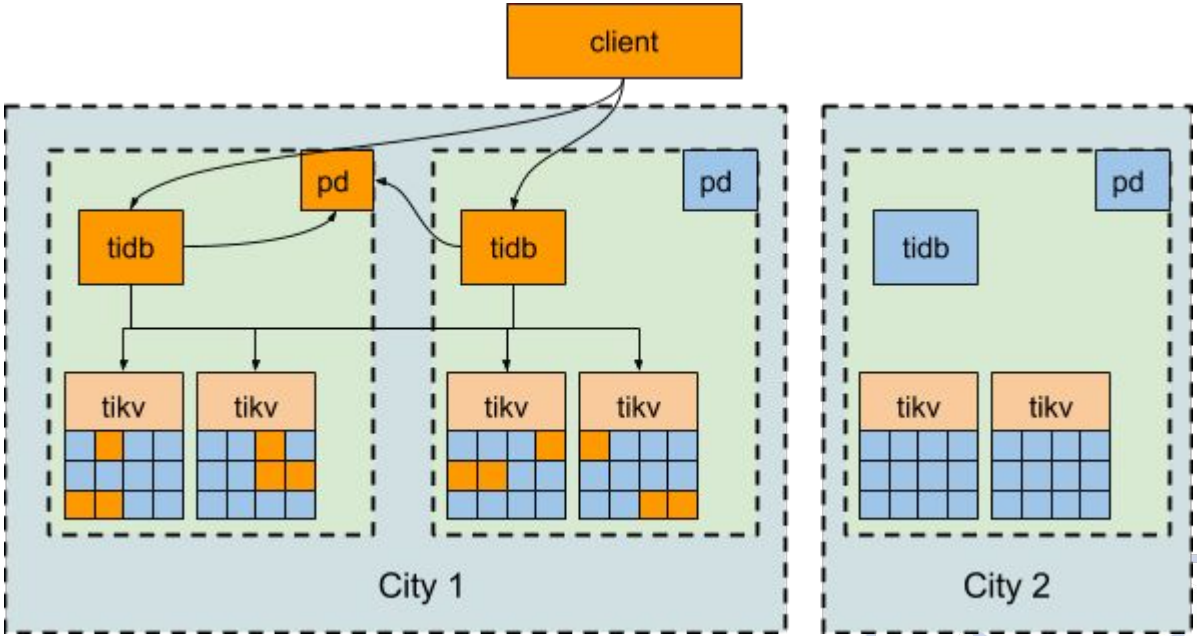
1. Deployment
    a. Single DC deployment
    b. Cross DC deployment
2. Elasticly Scale
    a. Scale out
    b. Scale in
3. Fight with hotspot
    a. Good design to avoid hotspot
    b. Finding hot read/write hotspot
    c. Automatic hot region balancing based on statistics
    d. Manually balance
4. Performance Tuning
    a. How to find bottlenecks
    b. Tuning under heavy read workload
    c. Tuning under heavy write workload



1.1 Deployment - single IDC



1.2 Deployment - Cross IDC



3-DC in 2 regions deployment

1.3 Demployment - Configurations

- Configurations of Deployment

```
[replication]

max-replicas = 3

location-labels = [“zone”, “rack”, “host”]
```

- TiKV start with labels

```
tikv-server --labels zone=”z1”,rack=”r1”,host=”h1”

tikv-server --labels zone=”z2”,rack=”r2”,host=”h2”
```

2.1 Scale - Add new nodes

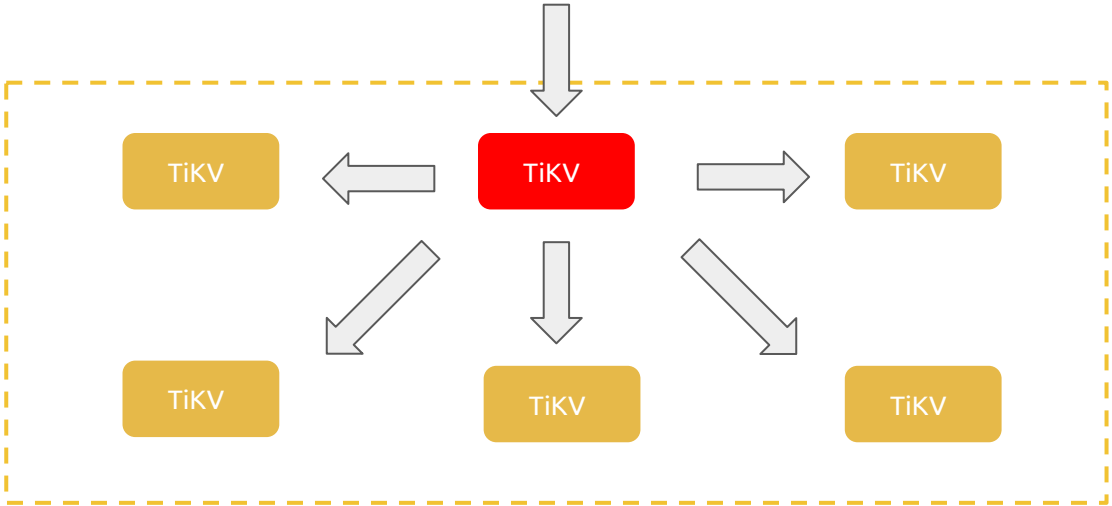
- Add new nodes is very simple, just start new TiKV nodes with correct pd address
  - tikv-server --addr 0.0.0.0:20171 --advertise-addr 172.16.4.56:20171 --status-addr 172.16.4.56:20181 --pd 172.16.4.51:2379,172.16.4.52:2379,172.16.4.53:2379 --data-dir /data3/deploy/data --config conf/tikv.toml --log-file /data3/deploy/log/tikv.log

2.2 Scale - Remove old nodes

- Use pd-ctl to remove old nodes
  - `>>` store // Display informations of all stores
  - `>>` store [store-id] // Get the store informations for specified store
  - `>>` store delete [store-id] // Delete specified store,
- After delete store use pd-ctl the state of this store will turn to Offline from Up. Don't close the deleted store now, because the replication works of this store's regions is still on-going
- After the deleted store's status truns to Tombstone, you can stop this TiKV

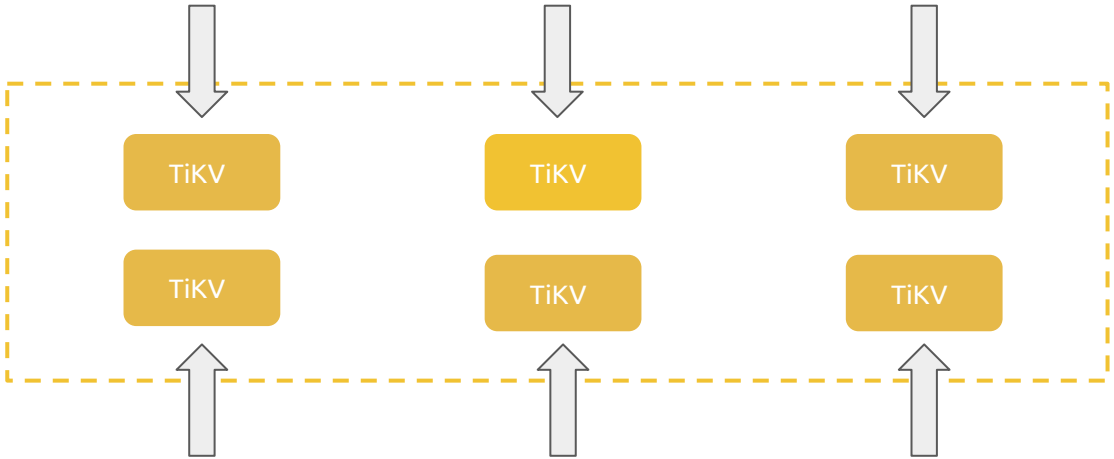
3.1 Fight with hotspot

- Issues with write hotspot
  - Single node becomes the bottleneck of whole cluster
  - Balance cost



3.2 Fight with hotspots - Good design

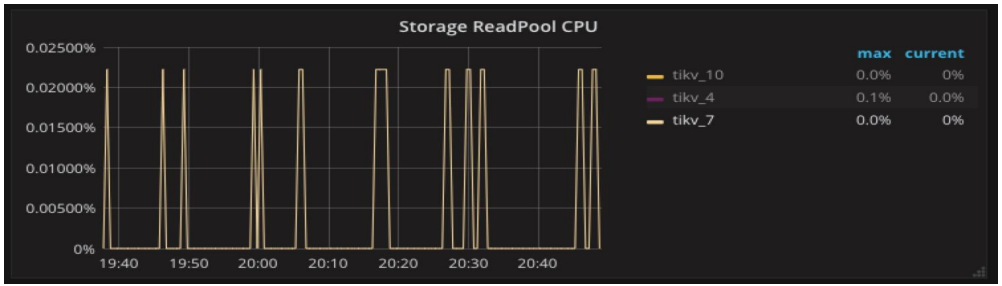
- No incremental key
  - update-time = now()
  - auto incremental id

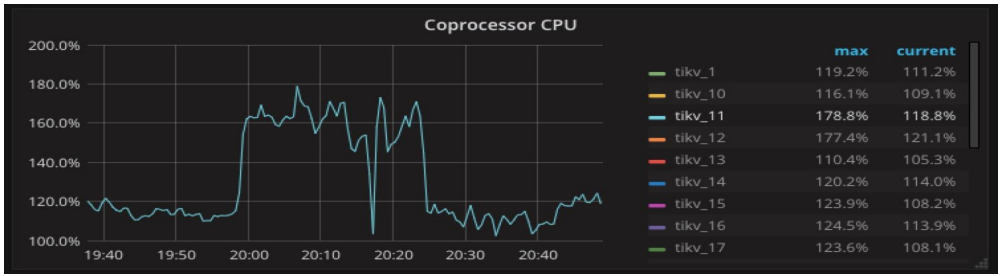


3.4 Fight with hotspots - Find write hotspot

3.5 Fight with hotspot - Find read hotspot

- Storage ReadPool handles KV read, Coprocessor handles DistSQL read
- Find which TiKV is more busier than others





3.6 Fight with hotspot - Auto balancing based on statistics

- TiKV Collect the write/read flow for each regions
- TiKV Send heartbeat(with read/write flow for each regions) to PD
- PD learns from the collected data and distinguash hot write/read regions
- Balance hot regions between TiKVs
    - Transfer leader
    - Move replica

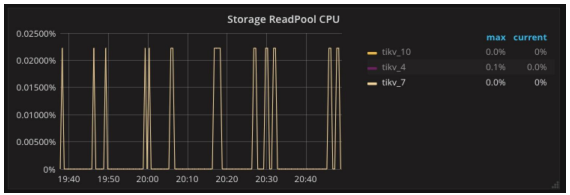
3.7 Fight with hotspot - Manual balance

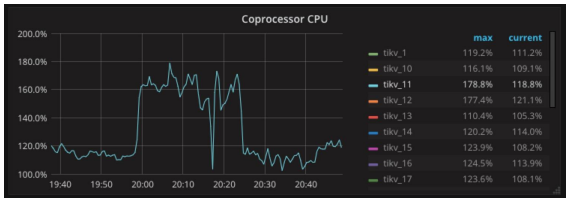
- Small table only contains one region
- Read workload is heavy in this small table
- Split this region by hand
  - pd-ctl -u http://{pd-host}:{pd-port}
  - >> operator add split-region <region_id> [--policy=scan|approximate]
  - >> operator add split-region 1 --policy=approximate // Split Region 1 into two Regions in halves, based on approximately estimated value
  - >> operator add split-region 1 --policy=scan // Split Region 1 into two Regions in halves, based on accurate scan value
- Transfer leaders & move replicas of regions
  - >> operator add transfer-leader <region_id> <to_store_id>
  - >> operator add transfer-peer <region_id> <from_store_id> <to_store_id>

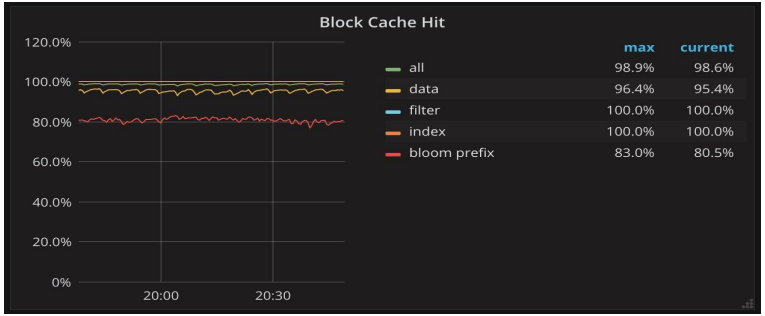
4.1 Performance tuning - Find bottlenecks

- Write
  - Does the raftstore thread pool is the bottleneck?
  - Does the apply thread pool is the bottleneck?
  - Does the DISK IO is the bottleneck?
  - Does the CPU is the bottleneck?
- Read
  - Does the storage read pool is the bottleneck?
  - How about the cache hit rate of RocksDB’s block-cache?

4.1 Performance tuning - Find bottlenecks

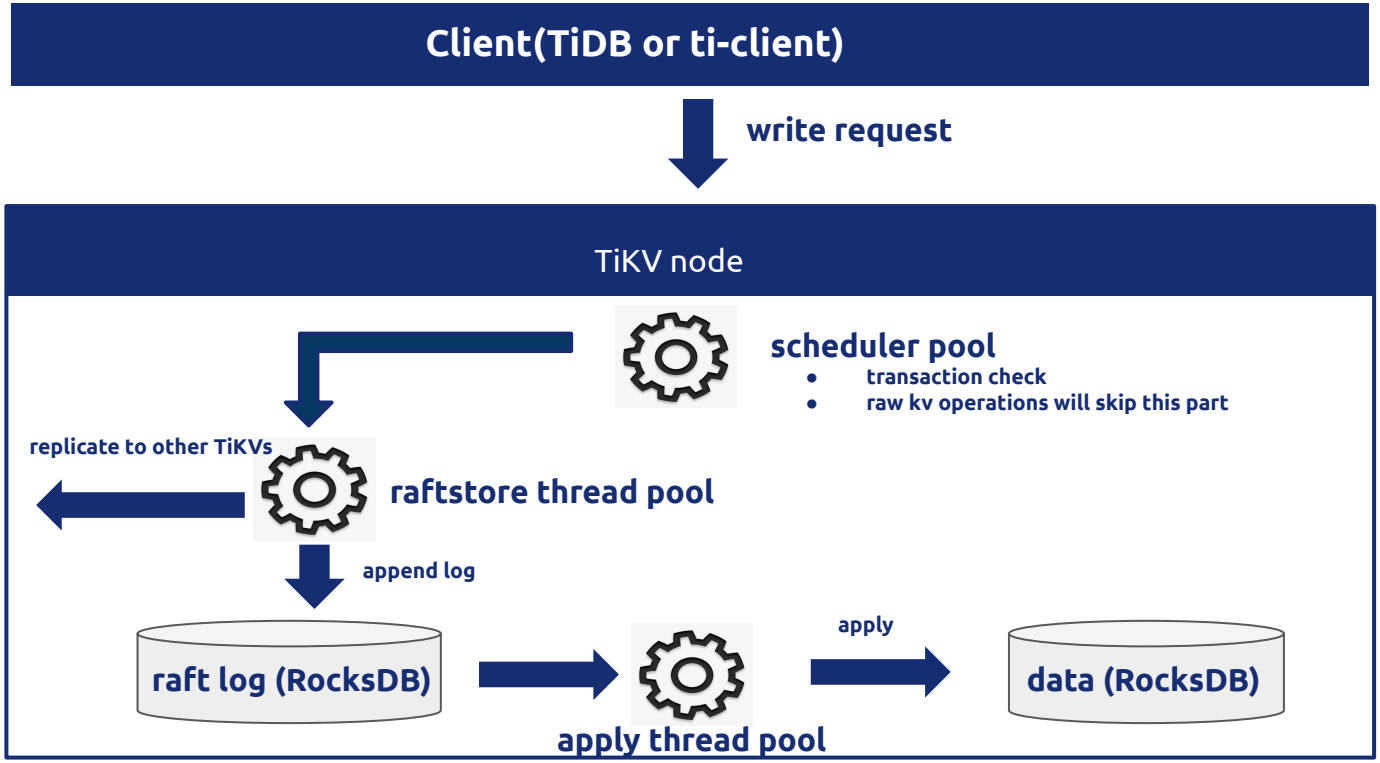






PingCAP

4.2 Performance tuning - Write in TiKV



[storage]
scheduler-worker-pool-size = 4

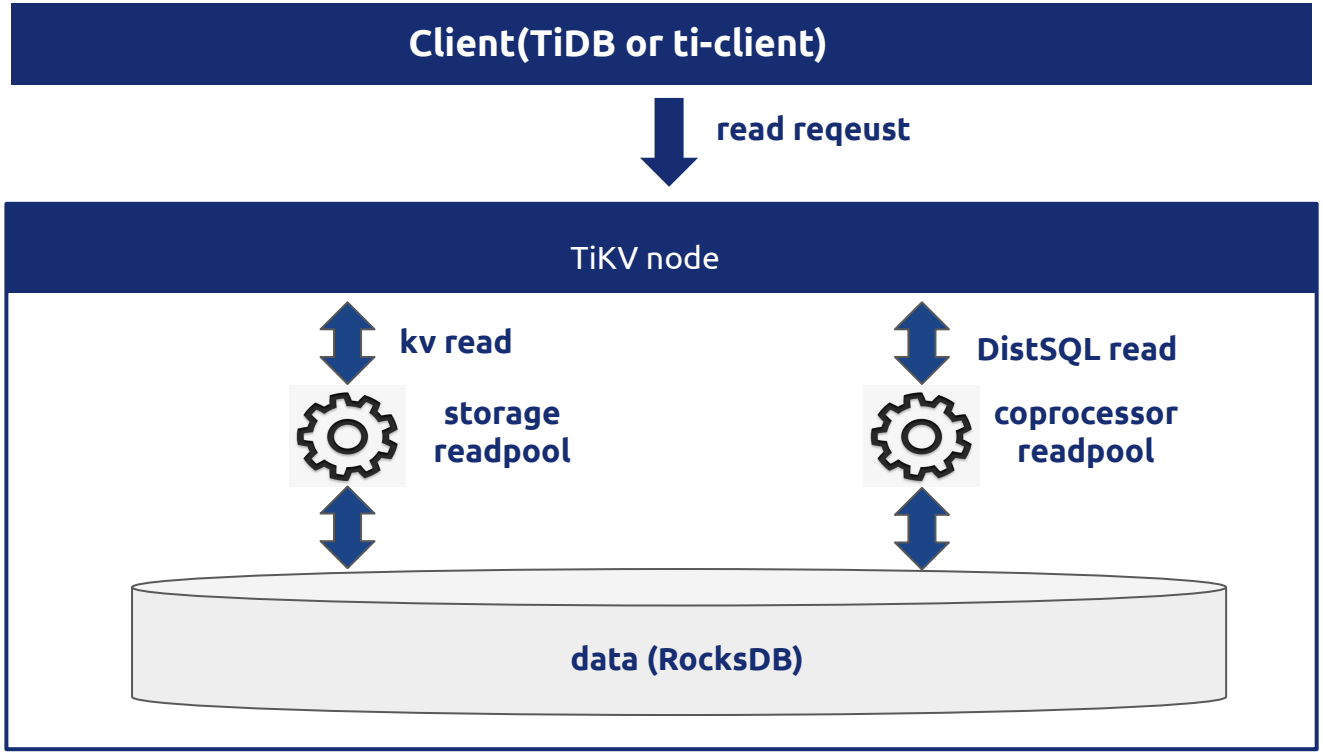
Version 3.0

[raftstore]
store-pool-size = 2
apply-pool-size = 2

4.2 Performance tuning - Write

- Rafttore thread pool is busy
  - [raftstore] store-pool-size = 2
- Apply thread pool is busy
  - [raftstore] apply-pool-size = 2
- DISK IO util is high, use compression type with higher compression rate
  - compression-per-level = [“no”, “no”, “lz4”, “lz4”, “lz4”, “zstd”, “zstd”]
- CPU usage is high, use compression type with low CPU cost
  - compression-per-level = [“no”, “no”, “no”, “no”, “lz4”, “lz4”, “lz4”]

4.3 Performance tuning - Read in TiKV

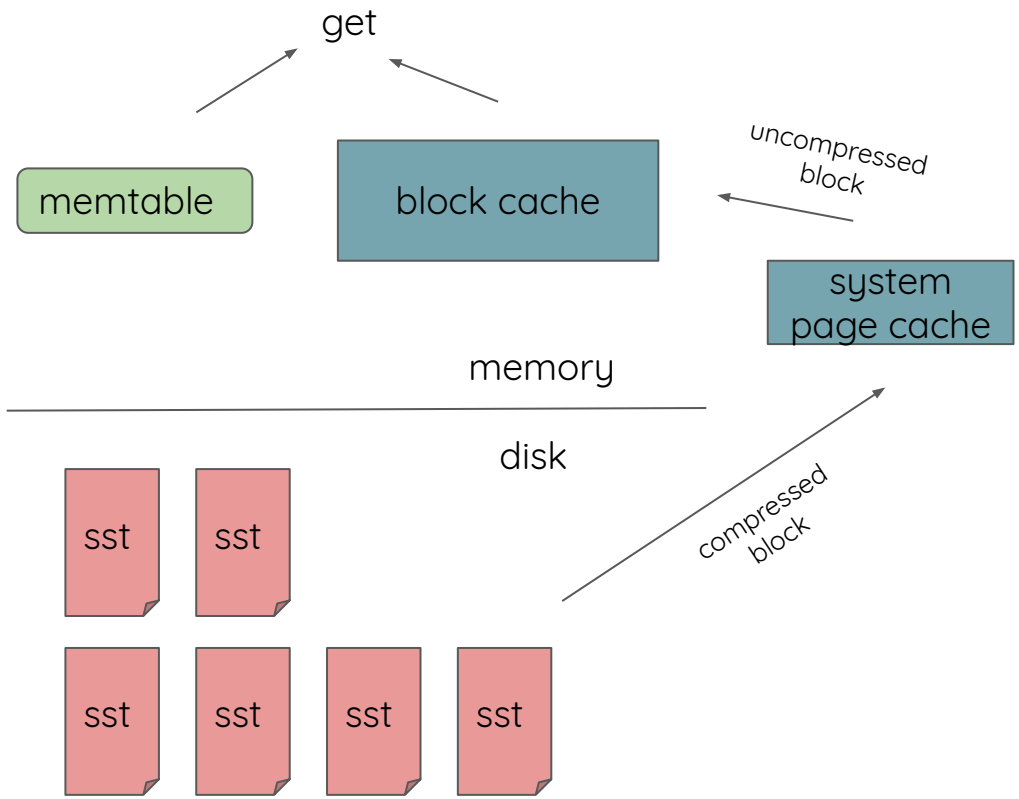



```
[readpool.storage]
high-concurrency = 4
normal-concurrency = 4
low-concurrency = 4

[readpool.coprocessor]
# default value is 80% *
core number
high-concurrency = 8
normal-concurrency = 8
low-concurrency = 9
```

4.3 Performance tuning - Read in RocksDB

- Get from memtable
- Get from block cache
- Reserve enough memory for page cache (30%~50%)
- [storage.block-cache] capacity = “20GB”



Thank You !