PingCAP

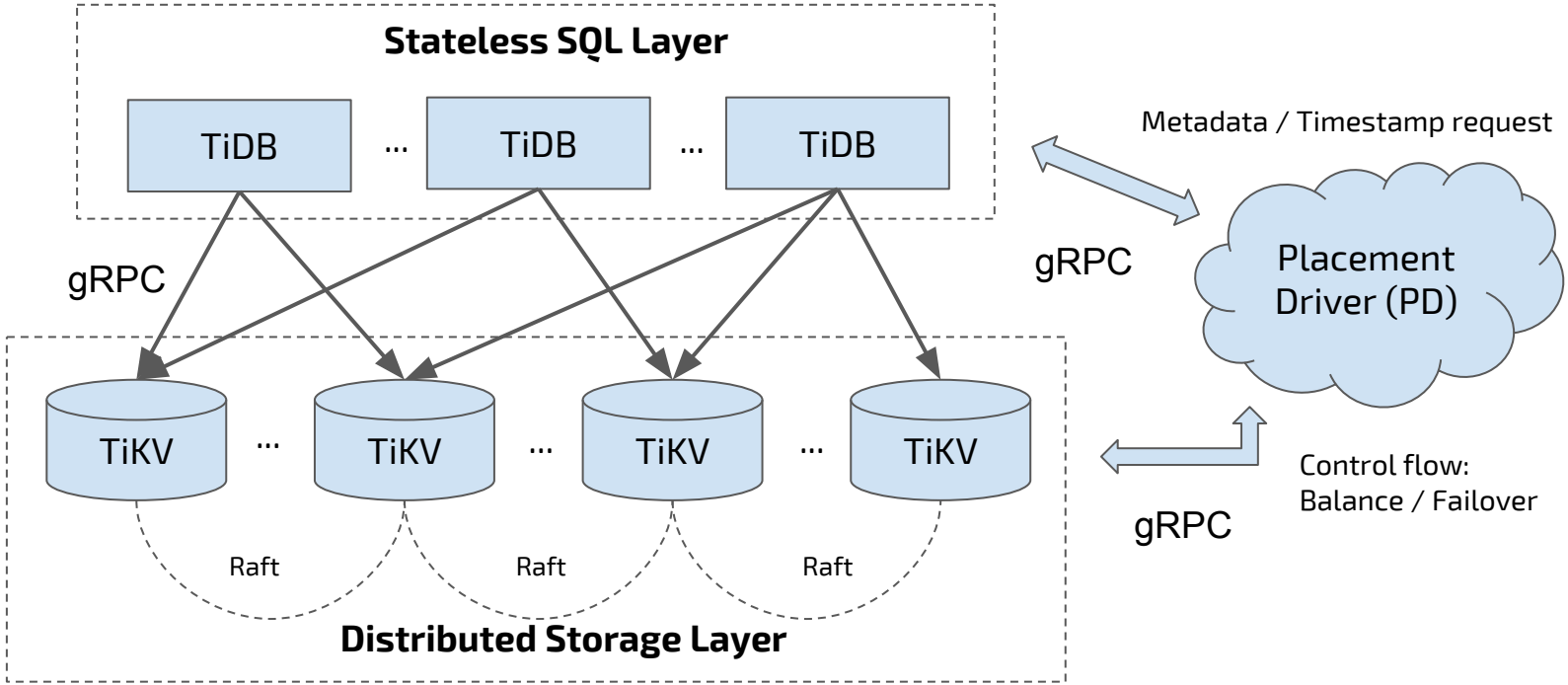
TiKV Internal

zhangjinpeng@pingcap.com

Summary

- Concepts
- Overview of TiKV
  - Scale
  - Two types of APIs
  - Layer structure
  - TiKV write flow
  - TiKV read flow
- TiKV Components Internal
  - gRPC
  - MultiRaft
  - RocksDB
  - Distributed Transaction
  - Coprocessor
  - GC
  - Balance
- New Features On The Way

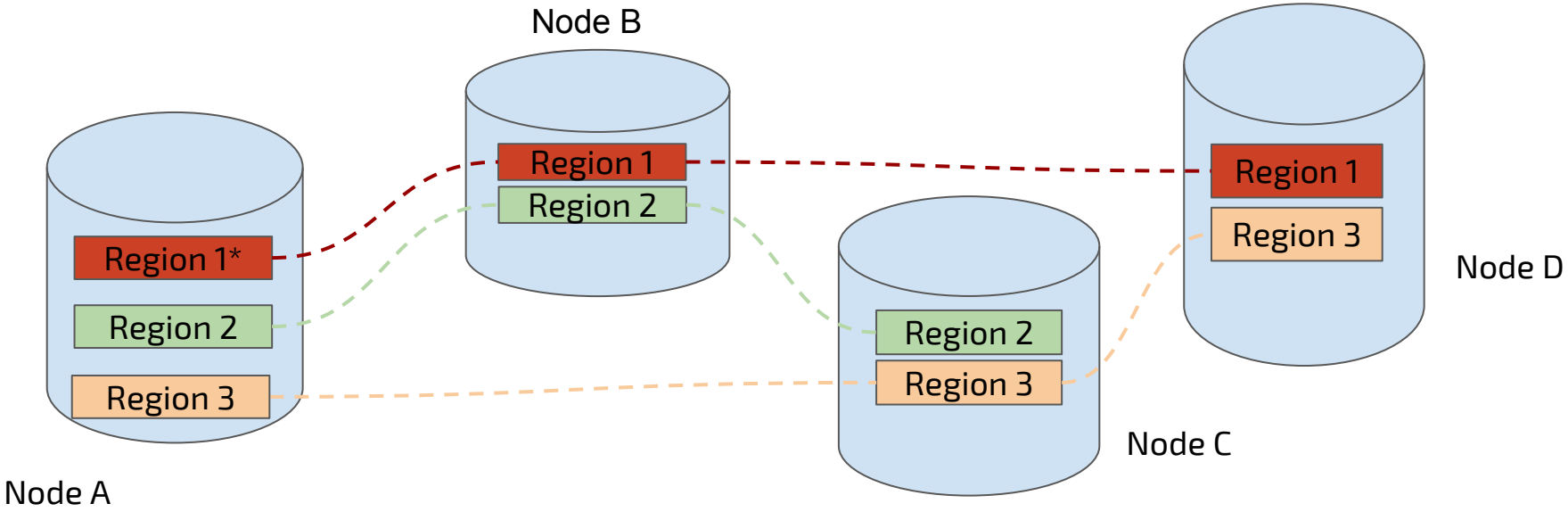
TiDB Cluster



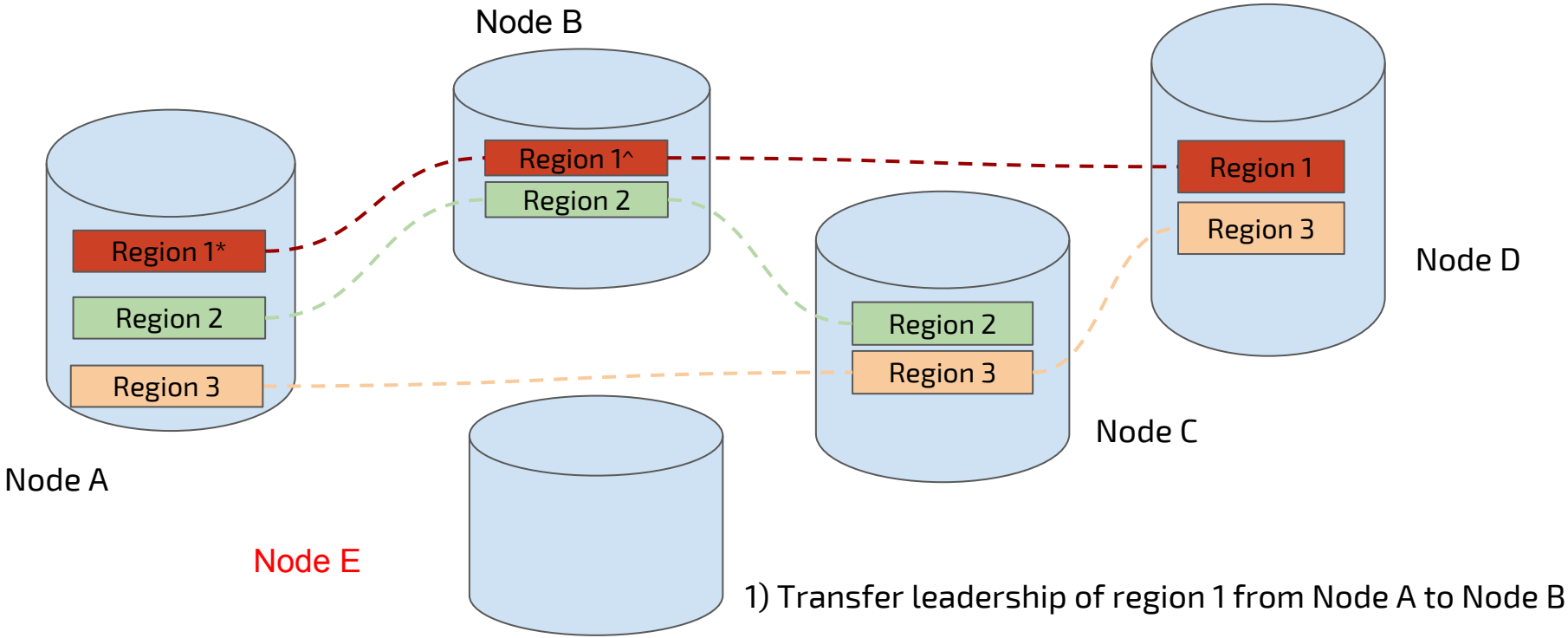
Concepts

- Region
  - A continuous range of data
- Peer
  - Replica of a Region
- ts
  - Timestamp, generated by PD

scale (initial state)

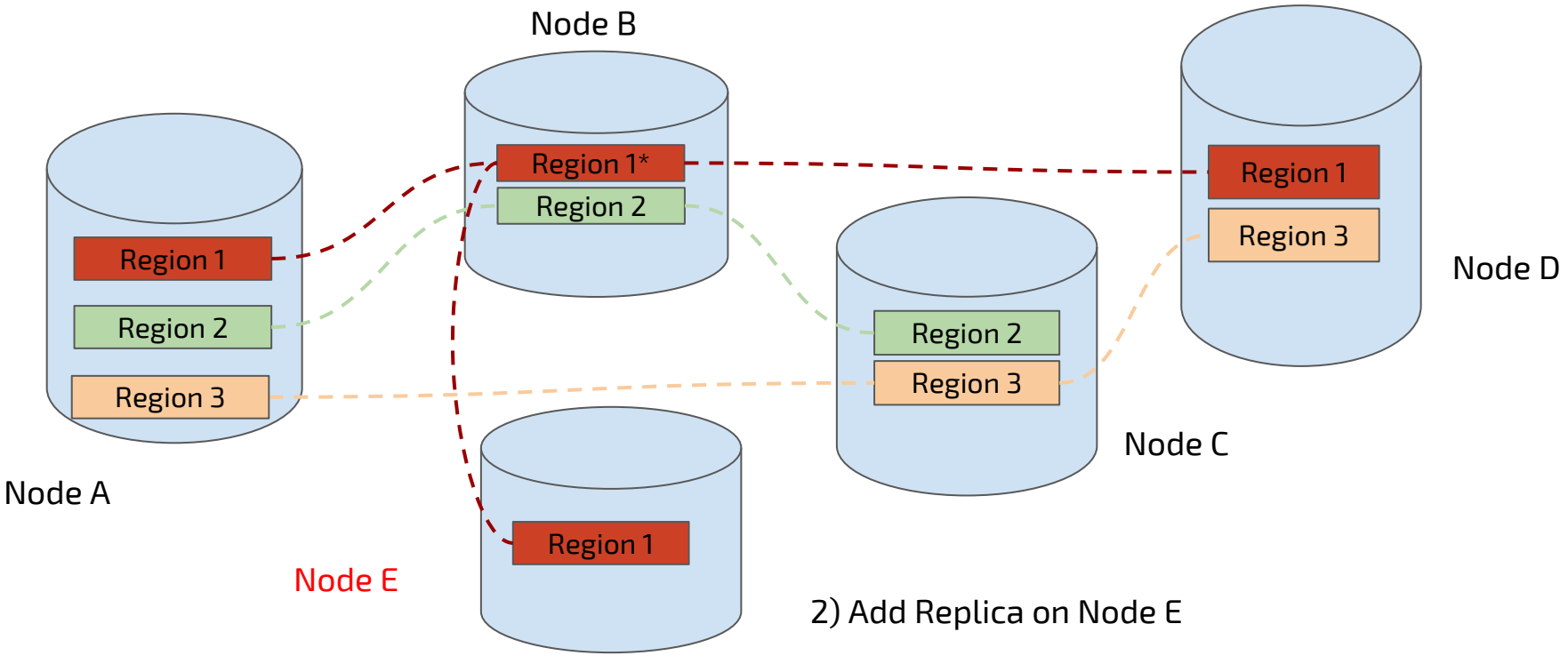


scale (add new node)

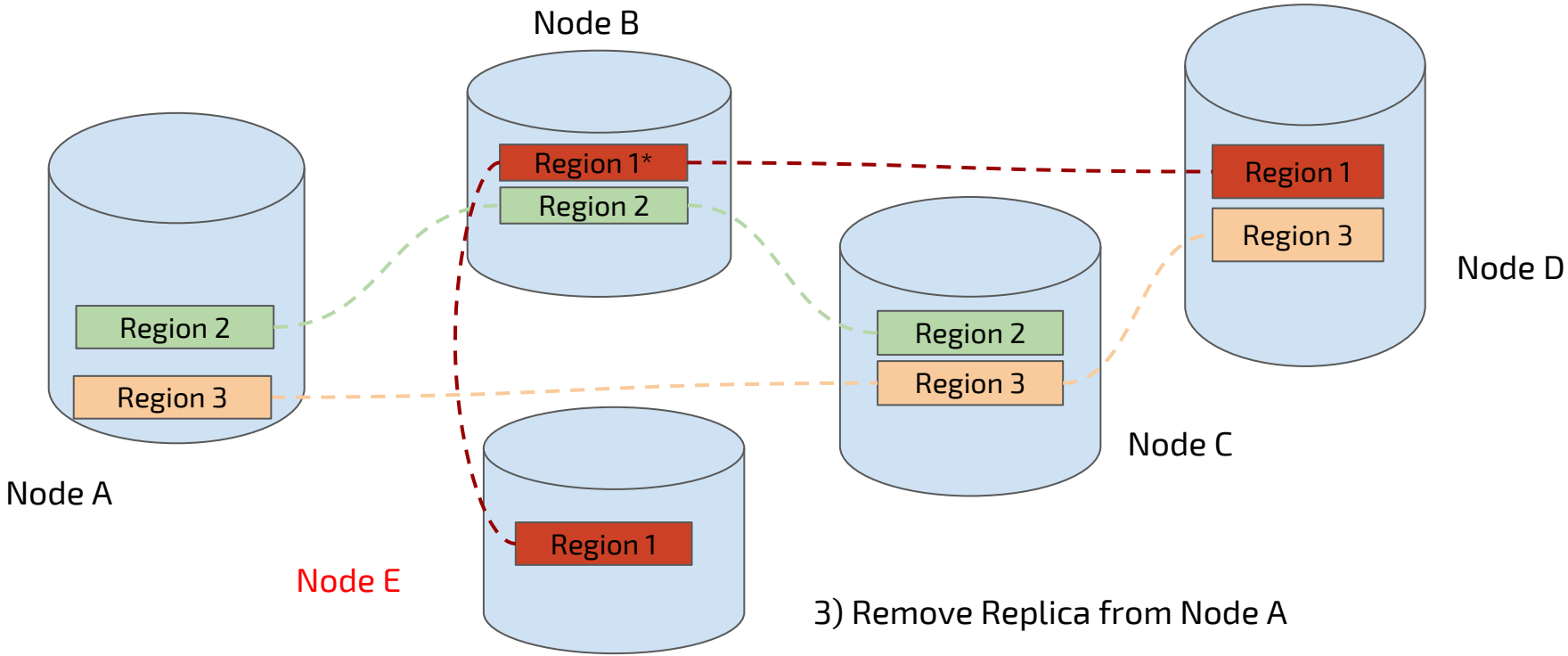


1) Transfer leadership of region 1 from Node A to Node B

scale (balancing)



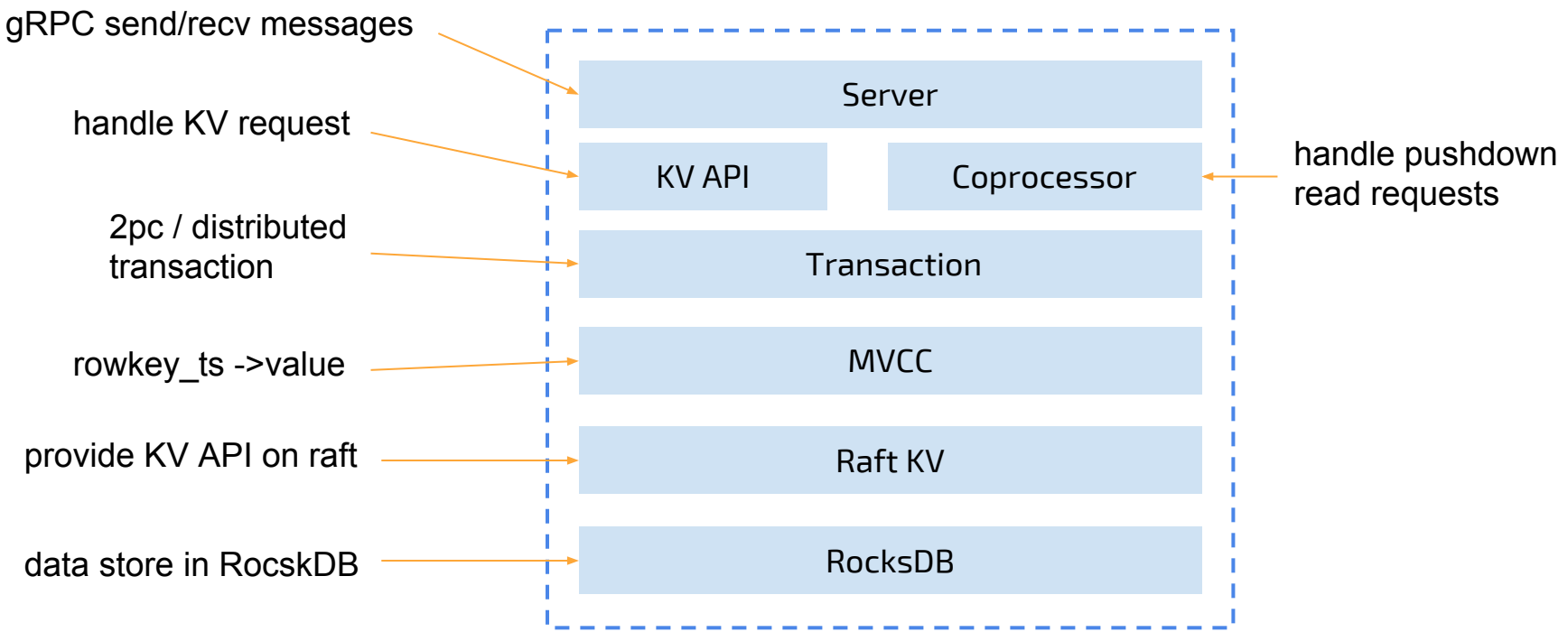
scale (balancing)



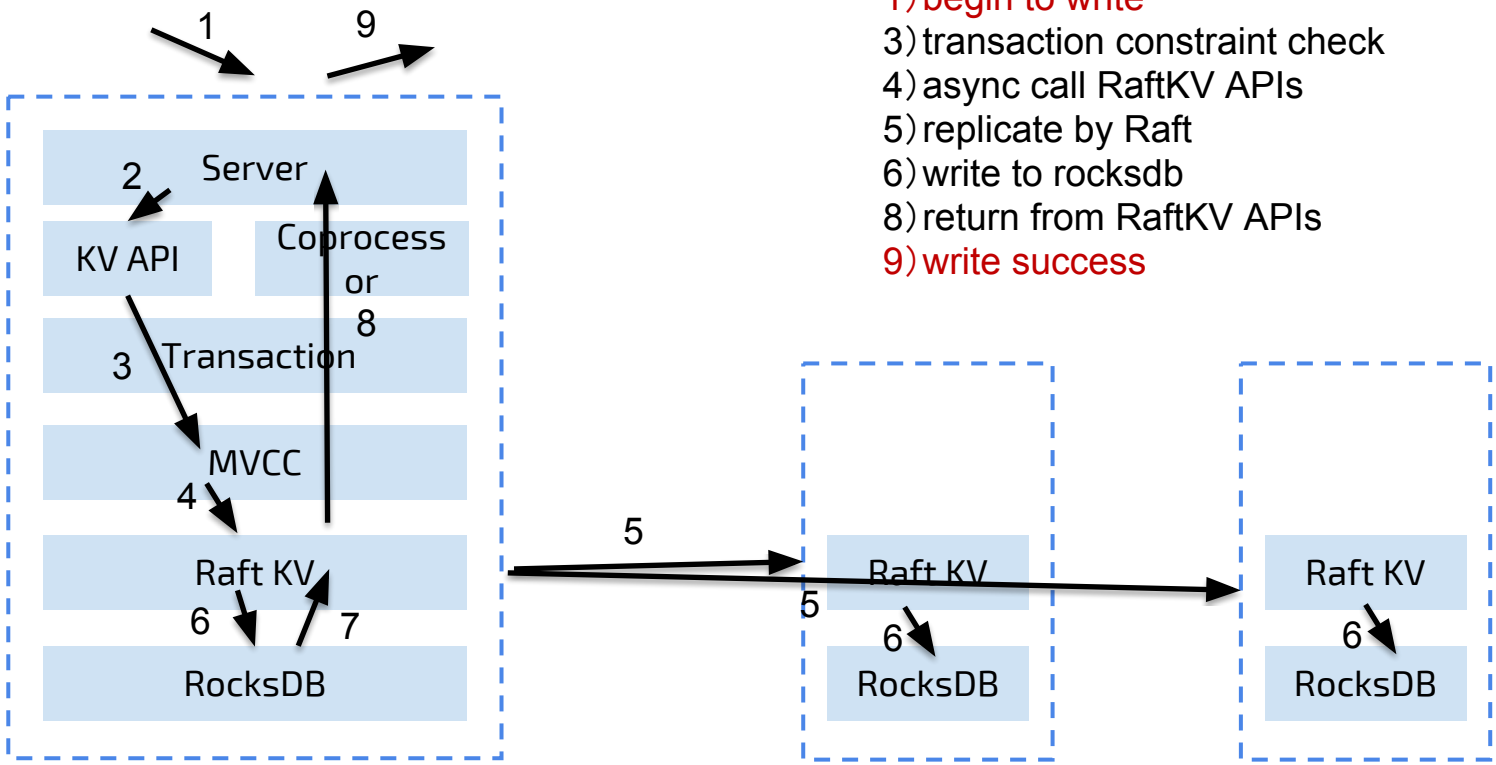
Two Types of APIs

- Raw Key-Value API
- Transaciton Key-Value API
- https://github.com/tikv/tikv/blob/master/docs/clients/go-client-api.md Please see more details in this documentation

TiKV Layer Structure

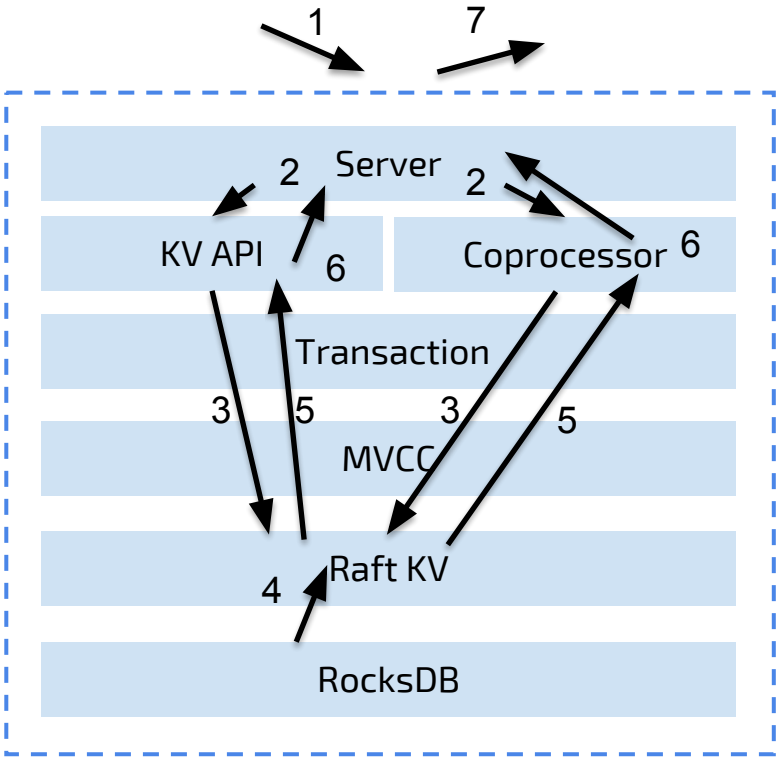


TiKV Write Flow



1) begin to write
3) transaction constraint check
4) async call RaftKV APIs
5) replicate by Raft
6) write to rocksdb
8) return from RaftKV APIs
9) write success

TiKV Read Flow



1）recv read request
2）dispatch to KV API or Coprocessor
3）send get snapshot request
4）check leader and get snapshot from RocksDB
5）return snapshot
6）read and return result
7）finish read

TiKV Components Internal

- gRPC
- Multi-Raft
- RocksDB
- Distributed Transaction
- Coprocessor
- GC
- Balance

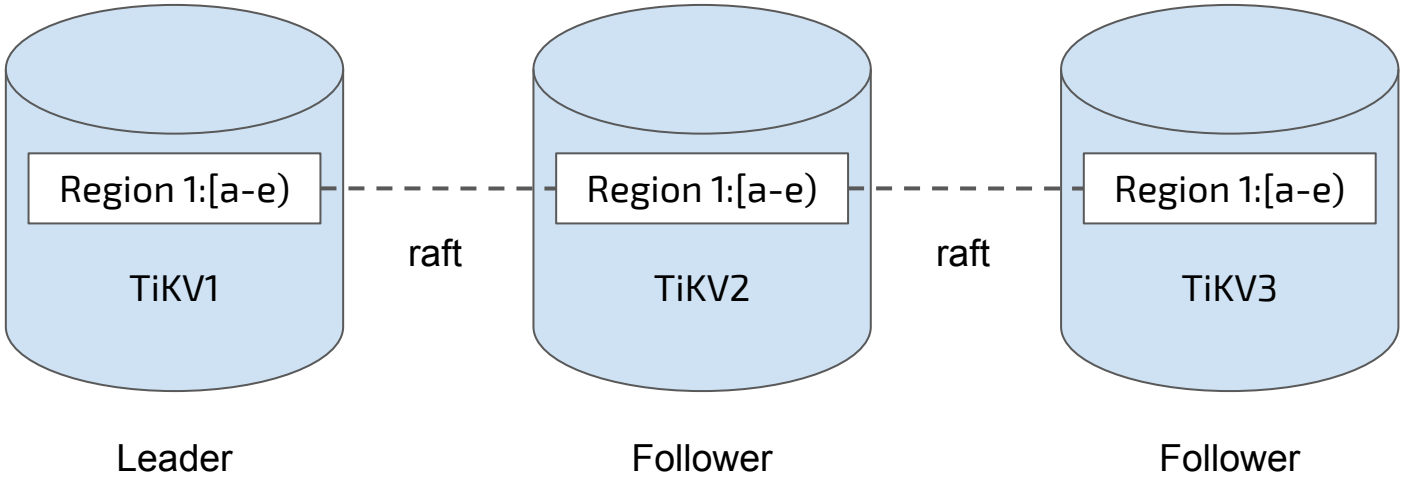
gRPC

- https://github.com/pingcap/grpc-rs The gRPC library for Rust built on C Core library and futures.
- Bi-directional streaming and fully integrated pluggable authentication with http/2 based transport
- Compression Support

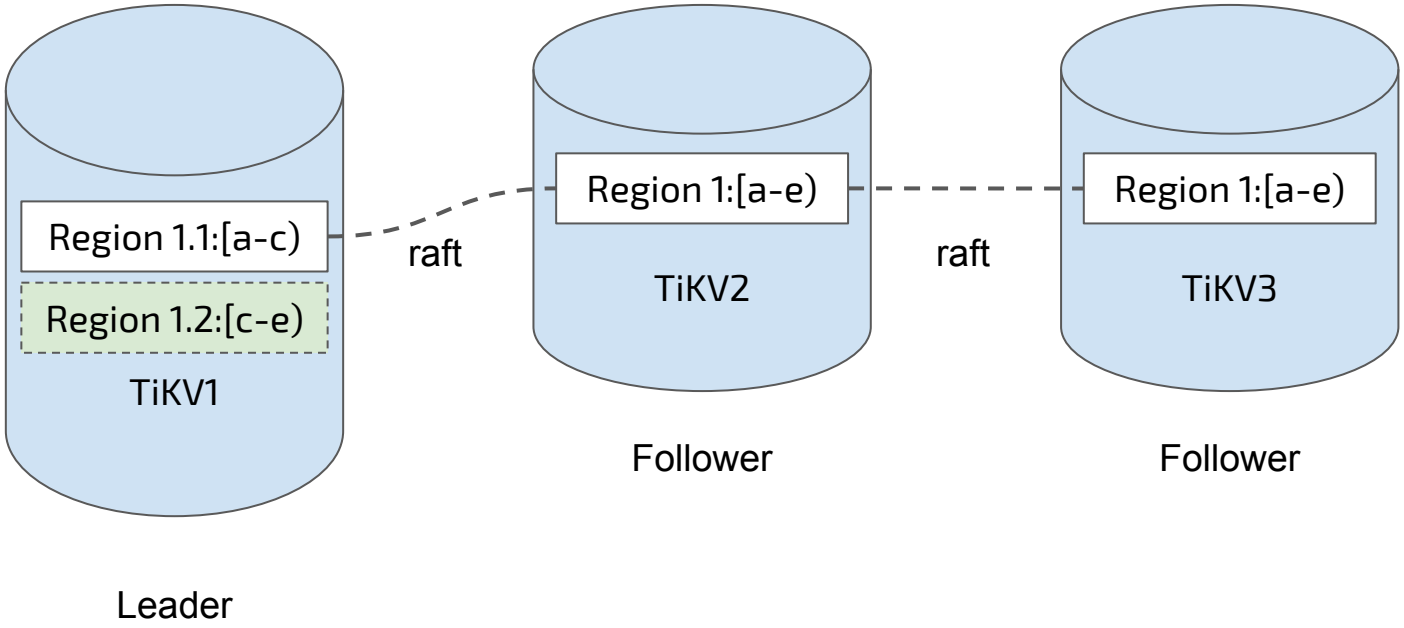
Multi-Raft

- Split
- Merge
- Leader lease
- Pre-vote
- Leaner

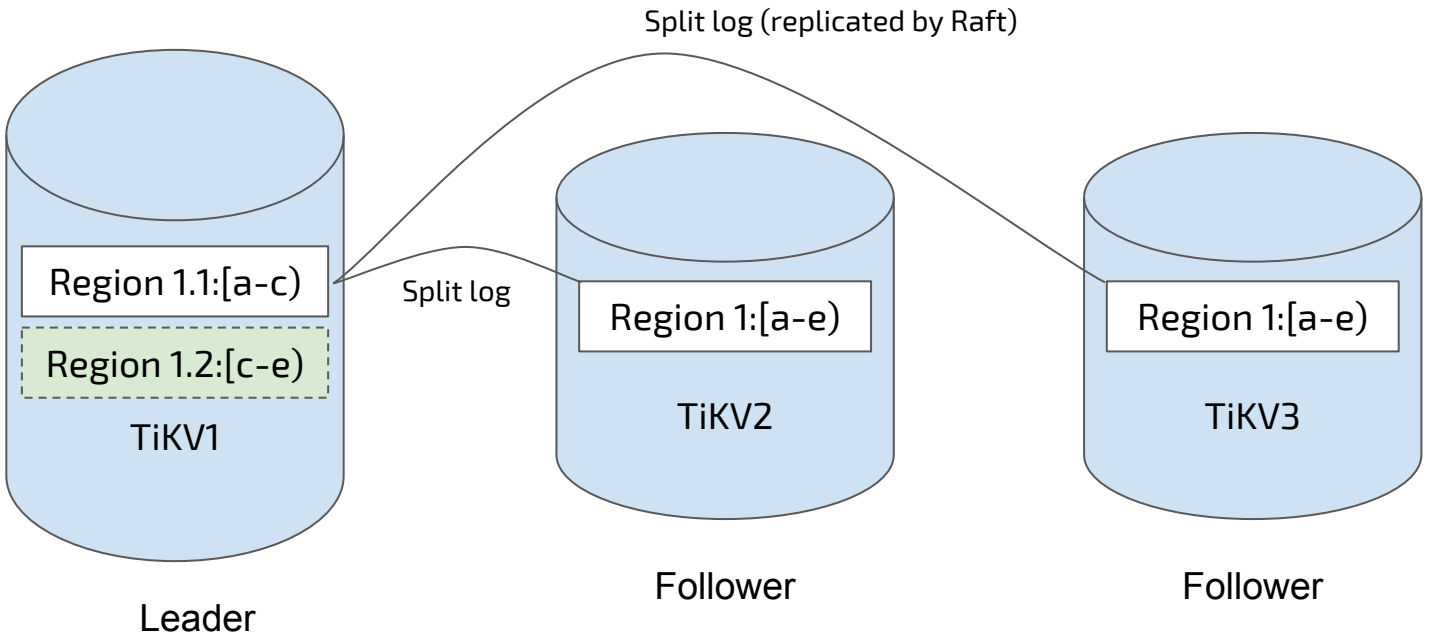
Multi-Raft (Region Split 1/4)



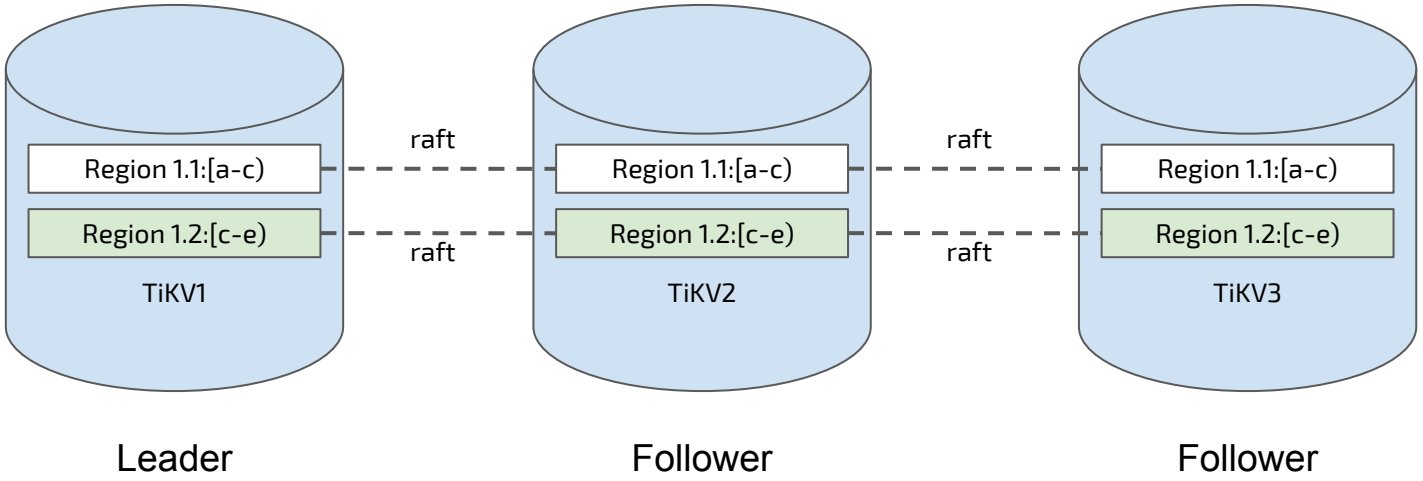
Multi-Raft (Region Split 2/4)



Multi-Raft (Region Split 3/4)



Multi-Raft (Region Split 4/4)



Multi-Raft (leader lease)

- No leader lease
  - leader recv read request
  - propose read on leader
  - leader send read message to followers
  - followers response to leader
  - leader get majority followers' response
  - read on leader
- Leader lease
  - leader recv read request
  - if leader in lease read directly
  - if not in lease, send a read index and update lease
- Follower promise that it will not vote for others in lease
- Update lease when write / raft read

Multi-Raft (pre-vote)

- Problem to fix:
  - When isolated node rejoin the cluster, its high term will make current leader step down and trigger a new election.
- Don’t increase term before receive majority votes

Multi-Raft (learner)

- Problem to fix:
  - Peers A, B, C
  - add a peer D, when C down before D has applied snapshot
  - D can’t vote, so A and B can’t receive the majority(3) votes
- Add D as learner not member, so D not belongs quorum, the majority is still 2
- After D has applied snapshot and catch up the raft log, it will become follower

RocksDB

- High performance persistent KV storage
- Support put / get / delete / scan
- WriteBatch support atomic write
- New features: column family(namespace) / Delete Rang / sub-compaction / multiple threads compaction / ingest sst file / delete files in range

```
WriteBatch wb;
wb.put(k1, v1);
wb.put(k2, v2);
wb.delete(k3);
db.write(wb);
```

RocksDB

- Cache hot data by block cache
  - [rocksdb.defaultcf] block-cache-size = “1GB”
  - [rocksdb.writecf] block-cache-size = “1GB”
- Using multiple column families
- Tuning for each column family

How table data map into KV pairs

- create table:

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

Table rows map to KV

```
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
...	...

Table rows/index map to KV

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

MVCC && Distributed Transaction

- 2PC
  - prewrite + commite
- MVCC
  - rowkey_ts -> value
- Percolater

MVCC

- All modifies are adding a new version
- The same row may has multiple versions
- Old versions dropped by GC

SQL type	key	value	op type
insert	bob_1	abc	put
update	bob_10	ccc	put
delete	bob_30		delete

Percolater Transaction Model (MVCC)

data	lock	Write(commit)
a_1 = x		a_5, put, start ts = 1
a_18 = y		a_30, put, start ts = 18
b_5 = z	b, put, start_ts = 5 primary key = a	

Percolater Transaction Model(MVCC Read)

data	lock	write
a_1 = x		a_5, put, start ts = 1
a_18 = y		a_30, put, start ts = 18
b_5 = z	b, put, start_ts = 5 primary key = b	

	ts	result
get a	4	nil
get a	20	x
get a	35	y
get b	35	?

Percolater Transaction Model(single row)

Prewrite

data	lock	Write(commit)
a_1 = x	a, put, start_ts = 1 primary key = a	

Commit

data	lock	Write(commit)
a_1 = x	~~a, put, start_ts = 1~~ ~~primary key = a~~	a_5, put, start_ts = 1

Percolater Transaction Model(multiple row prewrite)

Prewrite stage

1. concurrent prewrite all keys
2. start to commit when all prewrite success

data	lock	Write(commit)
a_1 = x	a, put, start_ts = 1, primary key = a	
…	…	
d_1 = y	d, put, start_ts = 1, primary key = a	

Percolater Transaction Model(multiple row commit)

Commit stage

1. commit primary key first and return success or fail
2. and then async commit or rollback secondary

data	lock	Write(commit)
a_1 = x	~~a, put, start_ts = 1, primary key = a~~	a_5, put, start_ts = 1
…	…	
d_1 = y	d, put, start_ts = 1, primary key = a	

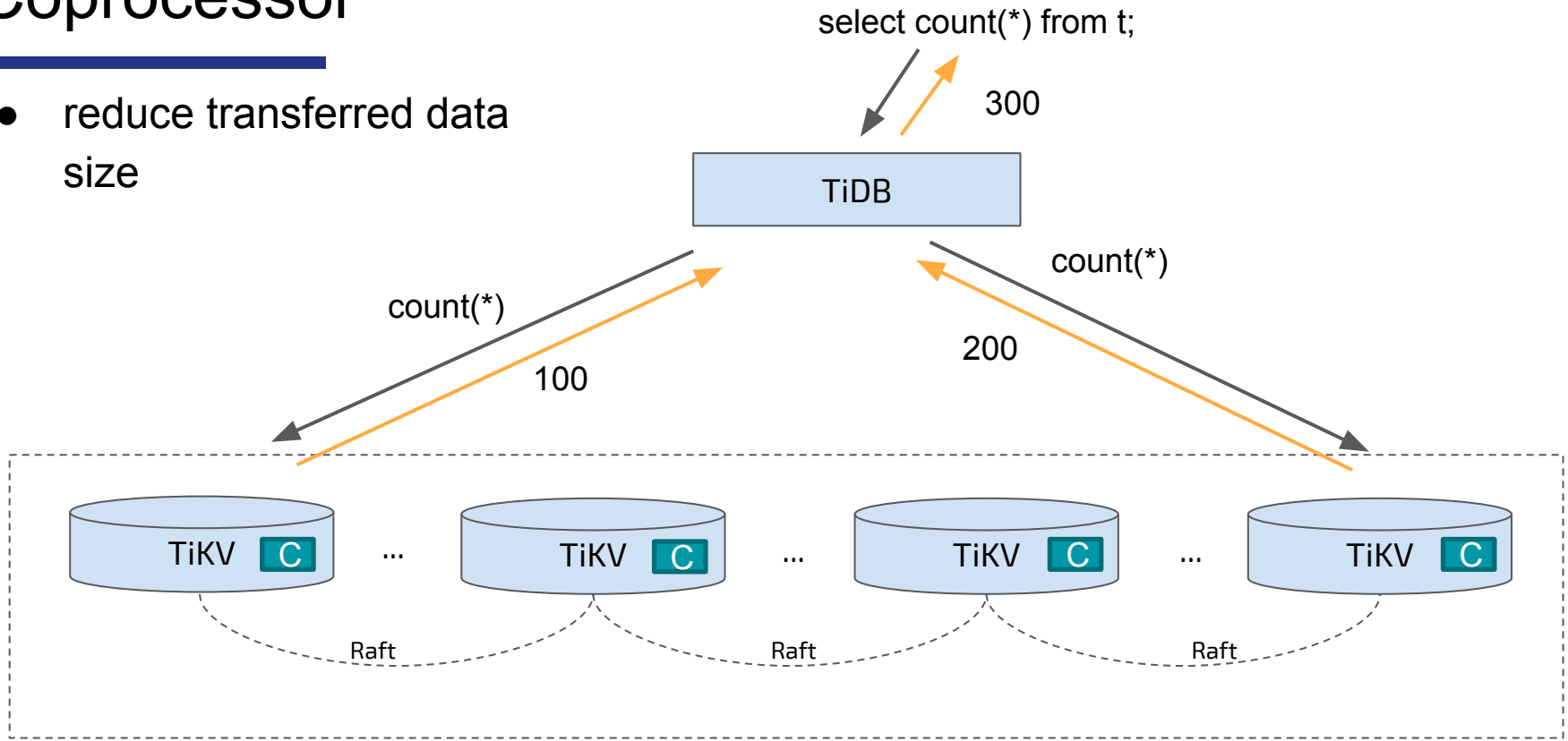
Percolater Transaction Model(multiple row final status)

Final status

data	lock	Write(commit)
a_1 = x	~~a, put, start_ts = 1, primary key = a~~	a_5, put, start_ts = 1
…	…	
d_1 = y	~~d, put, start_ts = 1, primary key = a~~	d_5, put, start_ts = 1

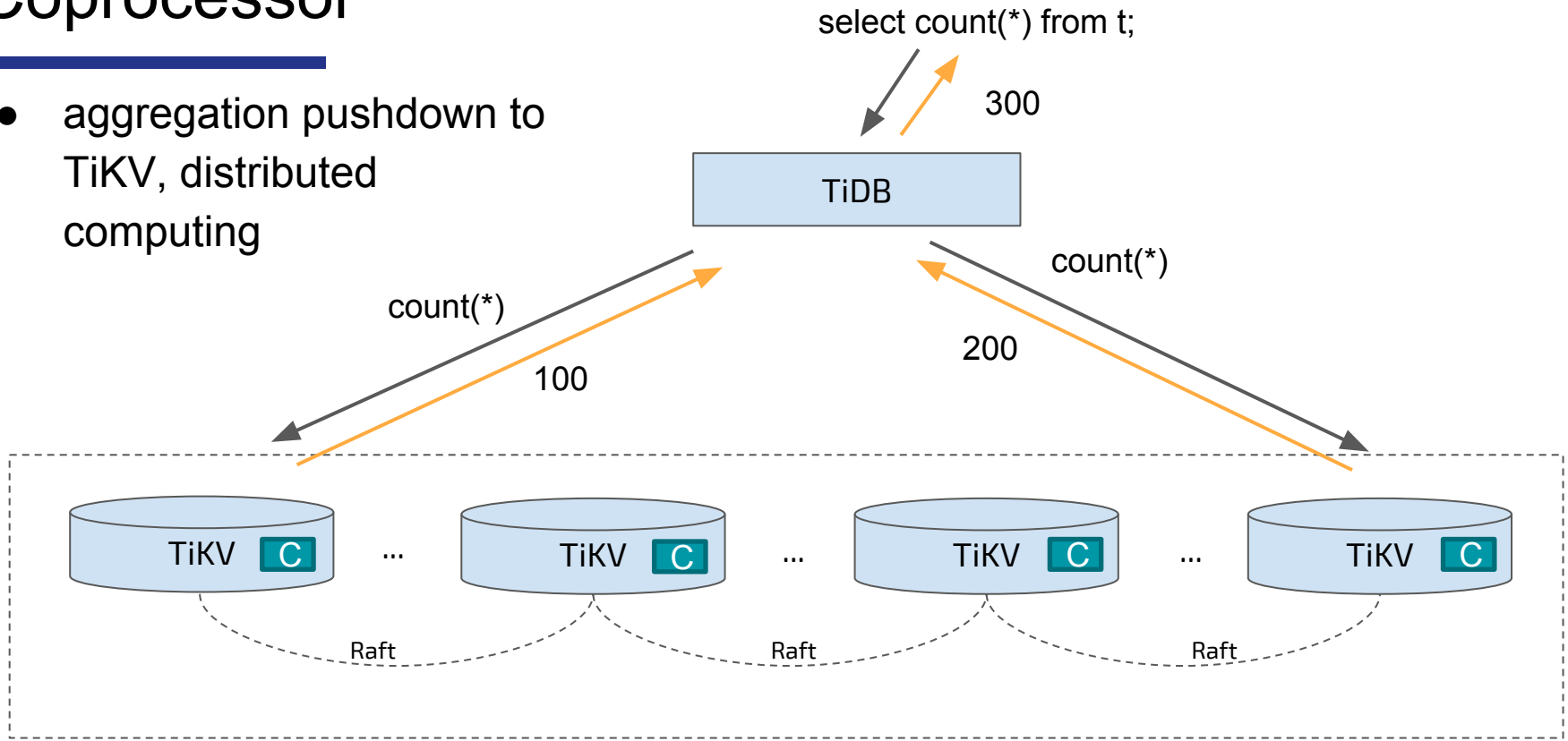
Coprocessor

- reduce transferred data size



Coprocessor

- aggregation pushdown to TiKV, distributed computing



GC

- Old versions before GC lifetime can be deleted
- Truncate/Drop a huge table
  - modify table meta and record the data range should be deleted
  - use delete files in range to reclaim space ASAP after GC lifetime reach
  - scan + delete the remaining data
- Remove peer by Balance
  - delete is write for RocksDB
  - reclaim space by delete files in range

Balance

- Move peers to keep each node’s space balance
- Hot read/write region Balancing
  - TiKV report read/write bytes for each region
  - PD evaluate and balance hot regions

New Features On The Way

- Multiple raftstore threads
- Multiple apply threads
- Joint consensus
  - add several peers at a time
  - add node + remove node = move
- New engine: TitanDB
  - seperate big value and key, reduce write amplification
  - more efficient GC mechanism
- Distributed GC
  - drop garbage data as soon as possible

How to become TiKV contributor in 30 minutes

https://mp.weixin.qq.com/s/lqJLvHcnCIB5UVpox3ZyXg 三十分钟成为 Contributor | 为 TiKV 添加 built-in 函数

PingCAP

Thanks

Q&A

https://github.com/pingcap/tidb

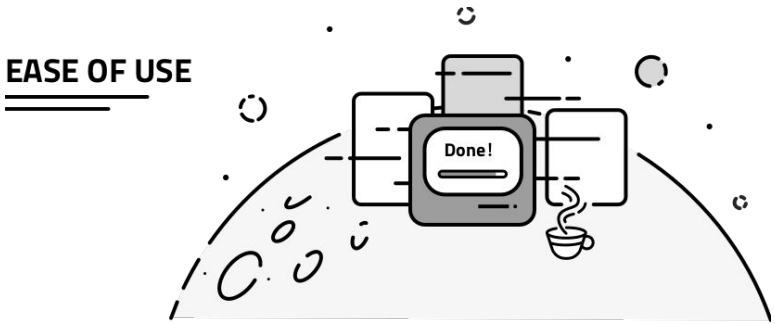
https://github.com/tikv/tikv

https://github.com/pingcap/pd

https://github.com/pingcap/tispark

https://github.com/pingcap/docs

https://github.com/pingcap/docs-cn



PingCAP

Thanks!

We Are Hiring
hire@pingcap.com