# *A Dance on Raft*

# About me

- Ed Huang
- CTO, PingCAP
- Co-founder of TiDB
- huang@pingcap.com

# Disitrubted Database 101

- Traditional RDBMS
  - MySQL / PostgreSQL / Oracle / DB2
- NoSQL
  - MongoDB / Cassandra / HBase
- NewSQL
  - **Shared-nothing**: Google Spanner / TiDB / CockroachDB
  - Shared-everything: Amazon Aurora
- **HTAP** (Hybrid Transactional/Analytical Processing)

# Challenges of HTAP systems:

## 0. Scale-out without pain

# Challenges of HTAP systems:

1. **Hybrid, but don't interfere with each other**

# Challenges of HTAP systems:

## 2. Data synchronization shouldn't be a bottleneck

# Overview

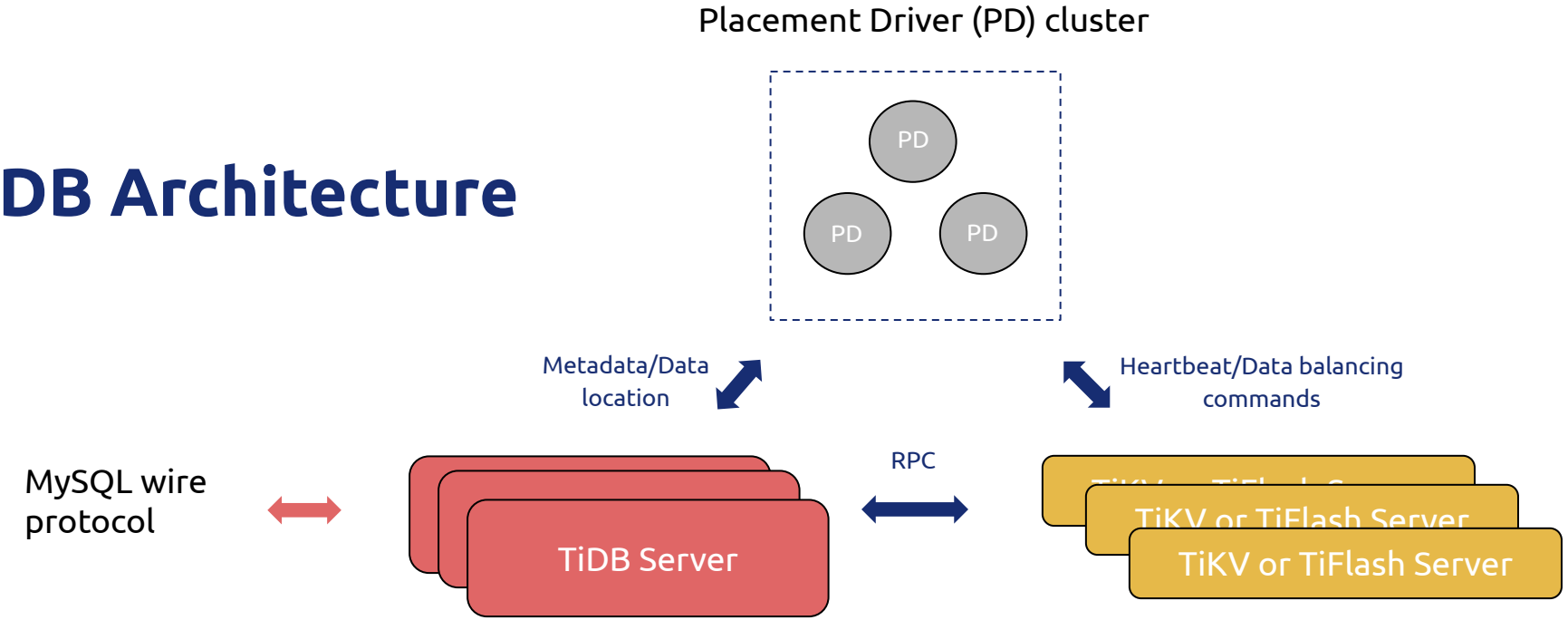
- TiDB's Multi-Raft Architecture
  - Key design points
    - Is Raft slow?
      - Optimizations on Real-World Raft
    - How to scale-out?
- How to apply Raft into a HTAP system
  - Raft Learner
  - DeltaTree Engine: a mutable columnar storage engine

*I'm not going talk about SQL and Transaction in this talk, maybe next time :)*

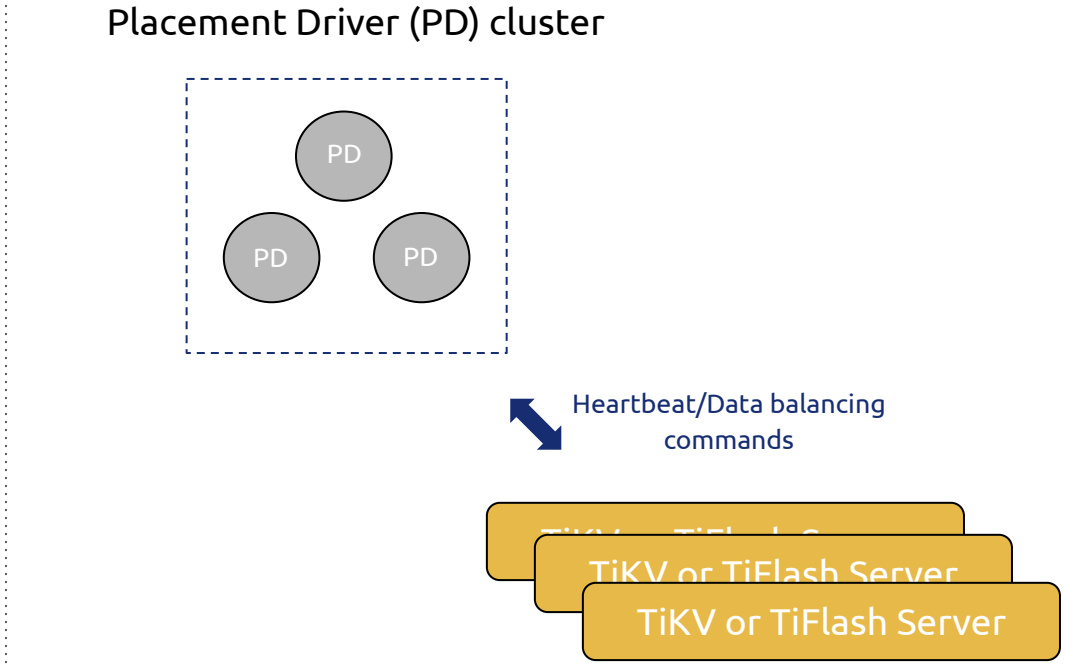
PingCAP

# TiDB's Architecture

# TiDB Architecture

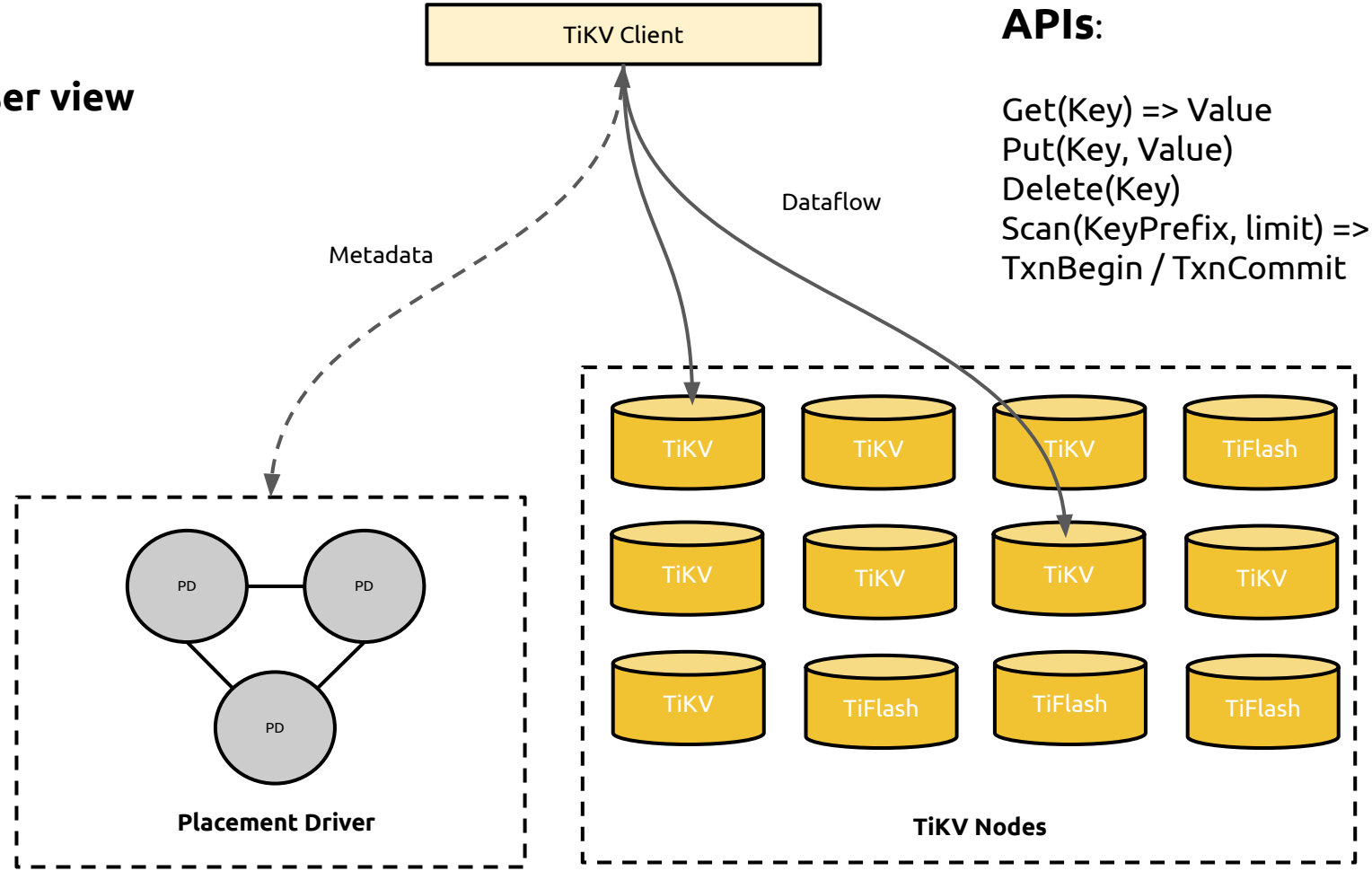


# TiDB Architecture




**We're focusing on these parts**

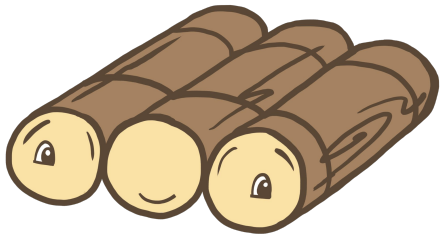
## Closer view



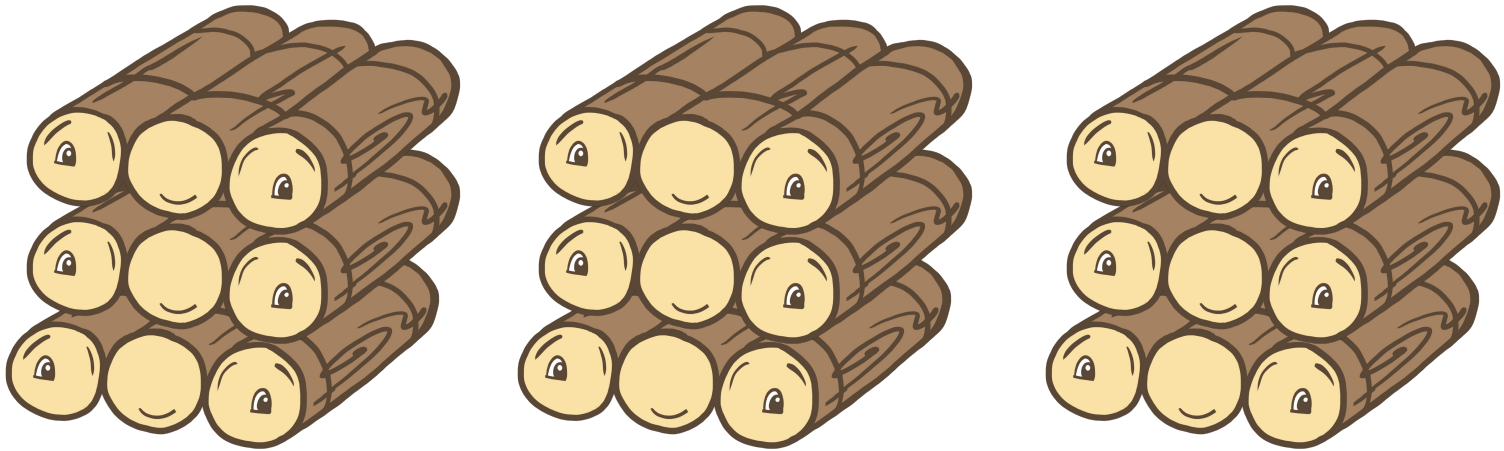

## APIs:

Get(Key) => Value
Put(Key, Value)
Delete(Key)
Scan(KeyPrefix, limit) => KV pairs
TxnBegin / TxnCommit

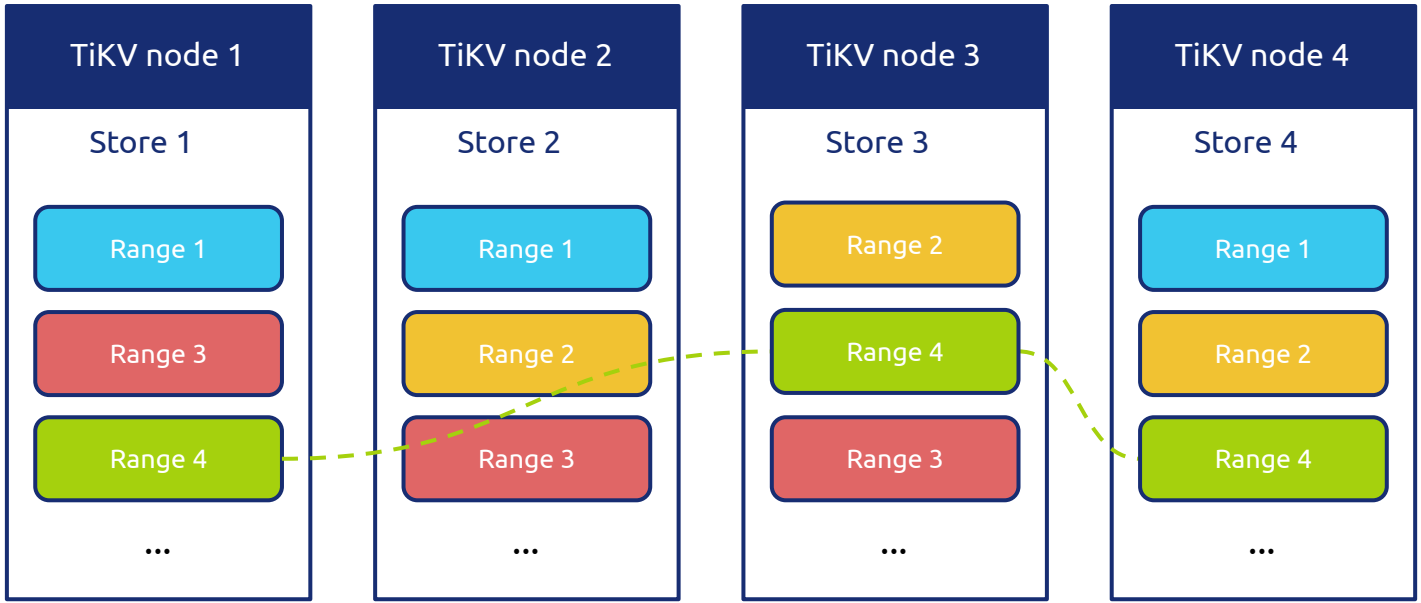
# What's Multi-Raft?



# What's Multi-Raft?



# Yes, Multi-Raft == Multiple Raft Groups :)




Every data Range is a Raft group

# Challenges

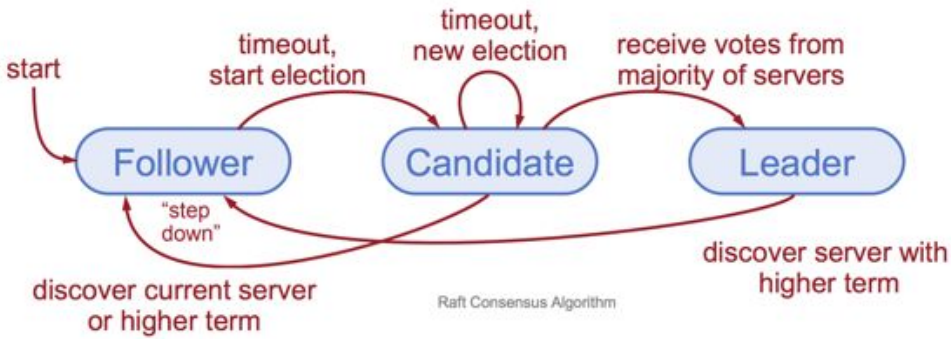
- Is Raft slow?
  - How to make Raft fast?
- How to safely split/merge data Range (aka Raft group)?
- How to move data around without pain?
  - Load balancing
  - Application-Aware data placement

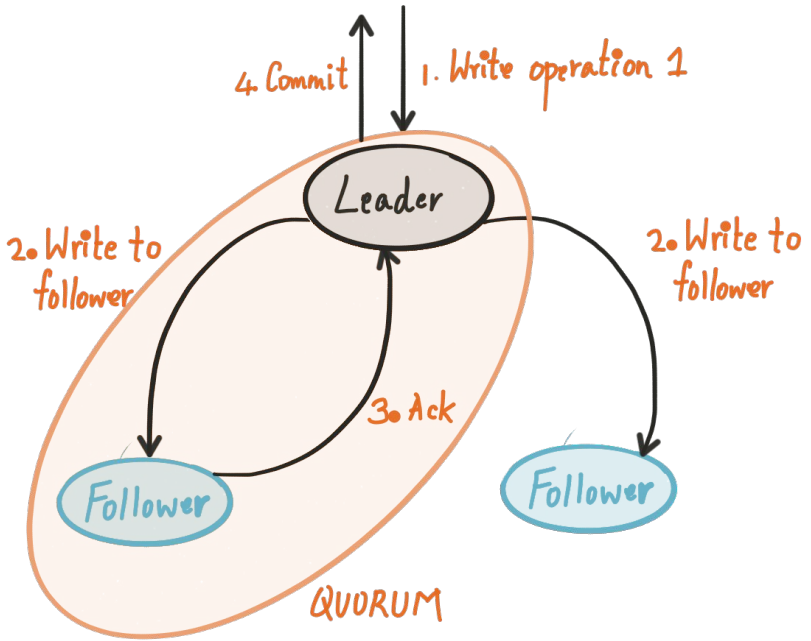
# Challenges

- Is Raft slow?
  - How to make Raft fast?
- How to safely split/merge data Range (aka Raft group)?
- How to move data around without pain?
  - Load balancing
  - Application-Aware data placement

# Raft 101

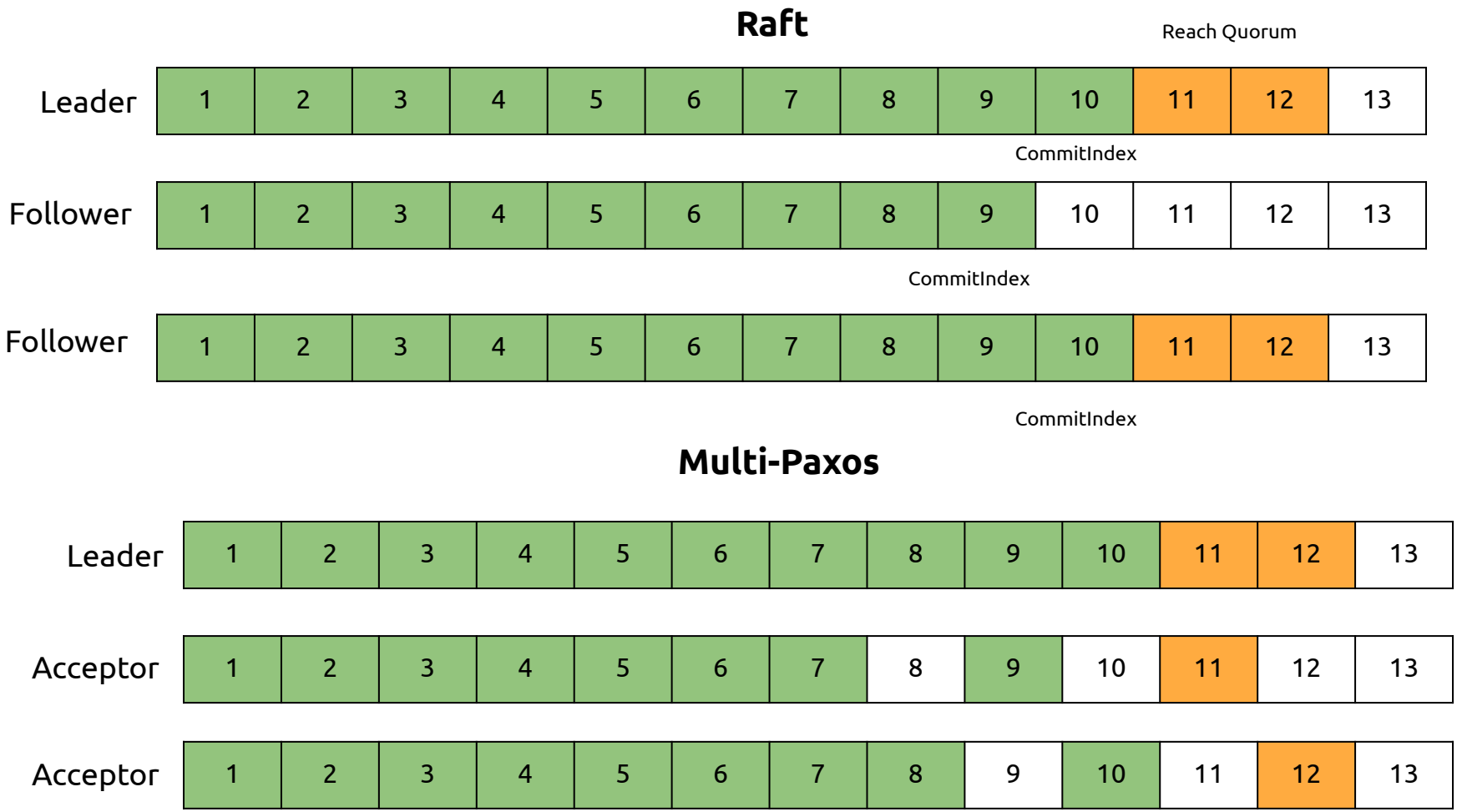
- **At any given time, each server is either:**
  - Leader: handles all client interactions, log replication
    - At most 1 viable leader at a time
  - Follower: completely passive (issues no RPCs, responds to incoming RPCs)
  - Candidate: used to elect a new leader
- **Normal operation: 1 leader, N-1 followers**



# Why do you think Raft is slow?

- Multi-Paxos vs Raft
  - Multi-Paxos allows 'holes' in logs!
    - That means higher throughput

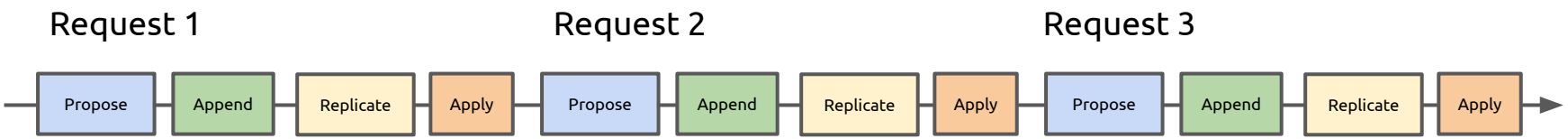

Raft
Reach Quorum
Leader
1 2 3 4 5 6 7 8 9 10 11 12 13
CommitIndex
Follower
1 2 3 4 5 6 7 8 9 10 11 12 13
CommitIndex
Follower
1 2 3 4 5 6 7 8 9 10 11 12 13
CommitIndex
Multi-Paxos
Leader
1 2 3 4 5 6 7 8 9 10 11 12 13
Acceptor
1 2 3 4 5 6 7 8 9 10 11 12 13
Acceptor
1 2 3 4 5 6 7 8 9 10 11 12 13


Raft
Reach Quorum
Leader
1 2 3 4 5 6 7 8 9 10 11 12 13
CommitIndex
Follower
1 2 3 4 5 6 7 8 9 10 11 12 13
CommitIndex
Follower
1 2 3 4 5 6 7 8 9 10 11 12 13
CommitIndex
Multi-Paxos
Leader
1 2 3 4 5 6 7 8 9 10 11 12 13
Hole!
Hole!
Acceptor
1 2 3 4 5 6 7 8 9 10 11 12 13
Hole!
Hole!
Acceptor
1 2 3 4 5 6 7 8 9 10 11 12 13

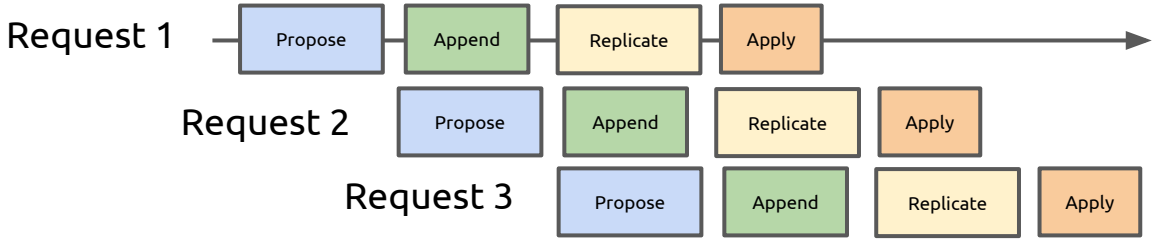
# Real-world Raft optimizations

- Pipeline
- Batch
- Decouple Append & Log replication (Leader only)
- Parallel Commit

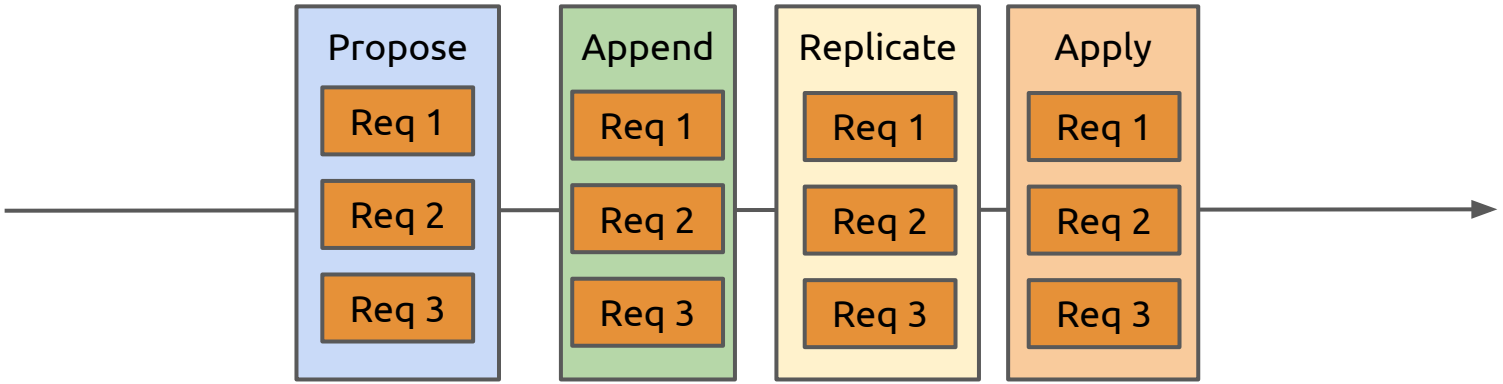
# Real-world Raft optimizations: Pipeline




VS

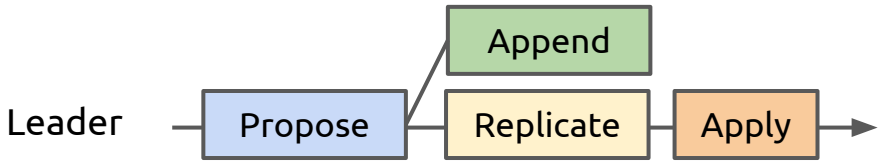


# Real-world Raft optimizations: Batch



# Real-world Raft optimizations:
# Decouple Append & Log Replication  (Leader only)


VS



# Real-world Raft optimizations: Parallel commit

- Notice: Commit != Apply
- Out-of-order Apply needs external information (dependencies of logs)
  - No expect for Multi-paxos

**PolarFS: An Ultra-low Latency and Failure Resilient Distributed File System for Shared Storage Cloud Database**

Wei Cao, Zhenjun Liu, Peng Wang, Sen Chen, Caifeng Zhu,
Song Zheng, Yuhui Wang, Guoqing Ma
{mingsong.cw, zhenjun.lzj, wangpeng.wangp, chensen.cs, caifeng.zhucaifeng,
catttree.zs, yuhui.wyh, guoqing.mgq}@alibaba-inc.com

**ABSTRACT**

PolarFS is a distributed file system with ultra-low latency and high availability, designed for the POLARDB database service, which is now available on the Alibaba Cloud. PolarFS utilizes a lightweight network stack and I/O stack in user-space, taking full advantage of the emerging techniques

The capacity and throughput of a storage cluster can be easily scaled out transparently. (3) Since data are all stored on the storage cluster, there is no local persistent state on compute nodes, making it easier and faster to perform database migration. Data reliability can also be improved because of the data replication and other high availability features of

# ParallelRaft

Key points:

- Introducing parallel Commit/Apply into Raft (allow holes in logs)
- Using 3rd acknowledge decide if logs could be apply in parallel

can be maintained.

**Out-of-Order Acknowledge.** After receiving a log entry replicated from the leader, a Raft follower would not acknowledge it until all preceding log entries are stored persistently, which introduces an extra waiting time, and the average latency increases significantly when there are lots of concurrent I/O writes executing. However, in ParallelRaft, once the log entry has been written successfully, the follower can acknowledge it immediately. In this way, the extra waiting time is avoided, so that the average latency is optimized.

**Out-of-Order Commit.** A raft leader commits log entries in serial order, and a log entry cannot be committed until all preceding log entries are committed. Whereas in ParallelRaft a log entry can be committed immediately after a majority of replicas have been acknowledged. This commit semantics is acceptable for a storage system which usually do not promise strong consistency semantics like a TP system. For example, NVMe does not check the LBA of read or write commands to ensure any type of ordering between concurrent commands and there is no guarantee of

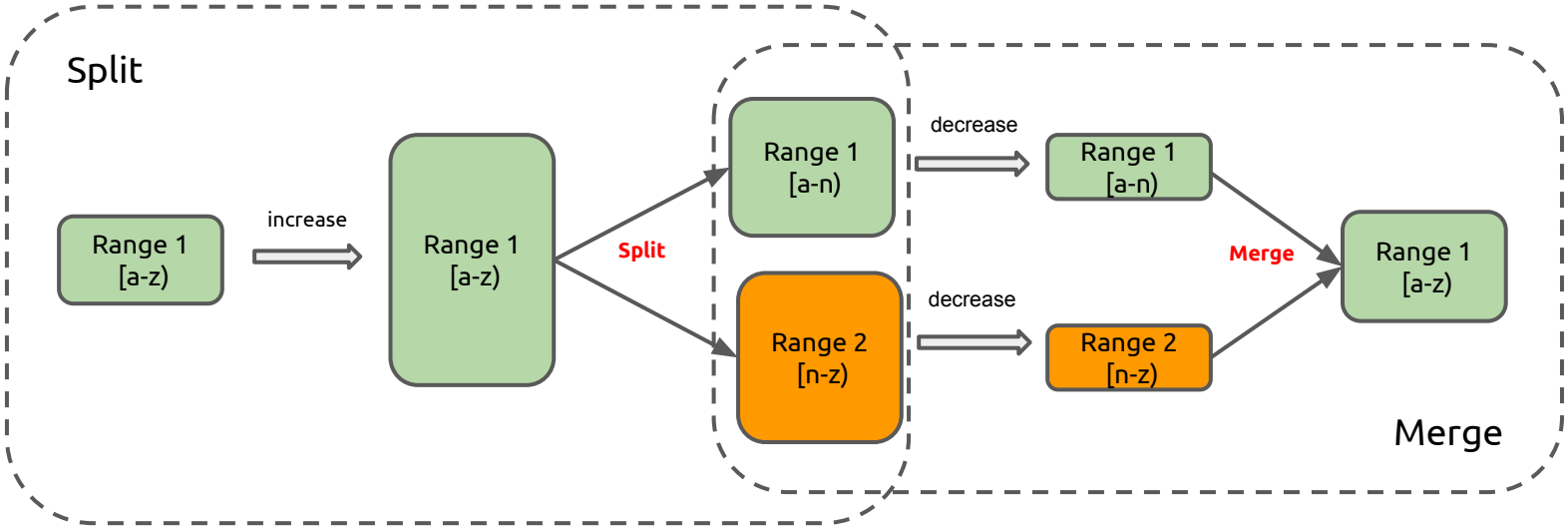
https://github.com/tikv/raft-rs

# Challenges

- Is Raft slow?
  - How to make Raft fast?
- How to safely split/merge data Range (aka Raft group)?
- How to move data around without pain?
  - Load balancing
  - Application-Aware data placement

# Dynamic Split/Merge

- Split/Merge based on **data size (or workload)**
  - 96 MB by default to split
  - 20 MB by default to merge

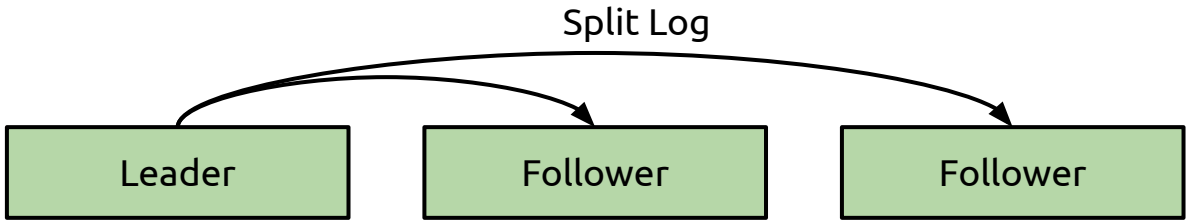


# Wait...

- How to make sure all replicas are successfully split?
  - 2 phase lock?

# Wait...

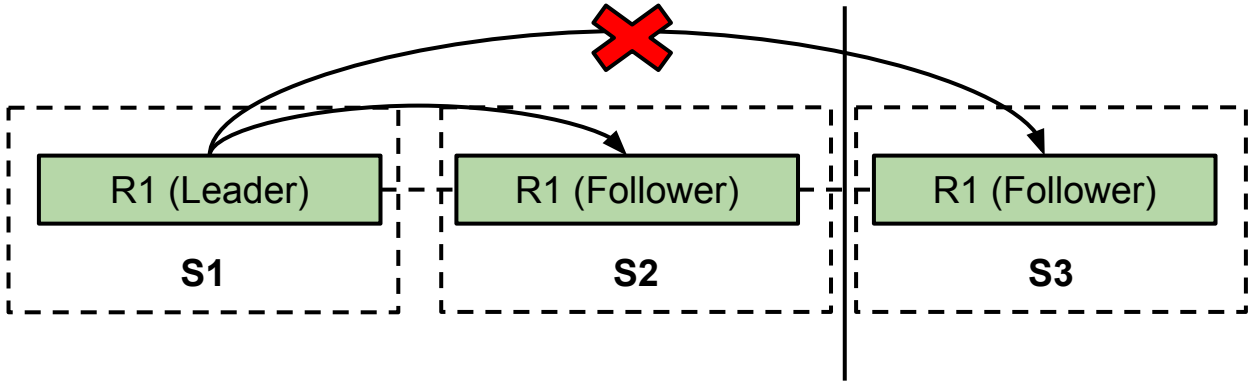
- How to make sure all replicas are successfully split?
  - ~~2 phase lock?~~ ←*Too complicated*
  - In TiKV, we take *Split Operation* as a specical Raft log
    - once this log is committed, that means the quorum is split successfully



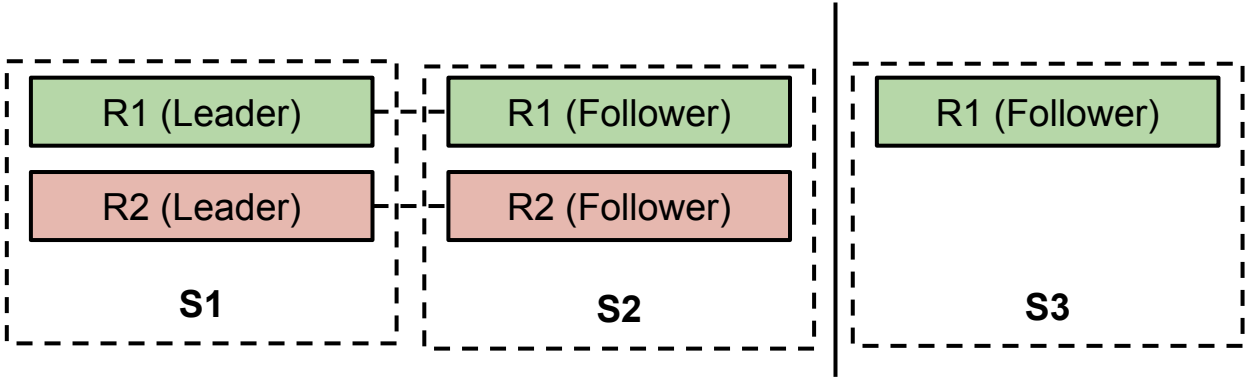

# Simple...Huh?

# An abnormal situation...

(1)

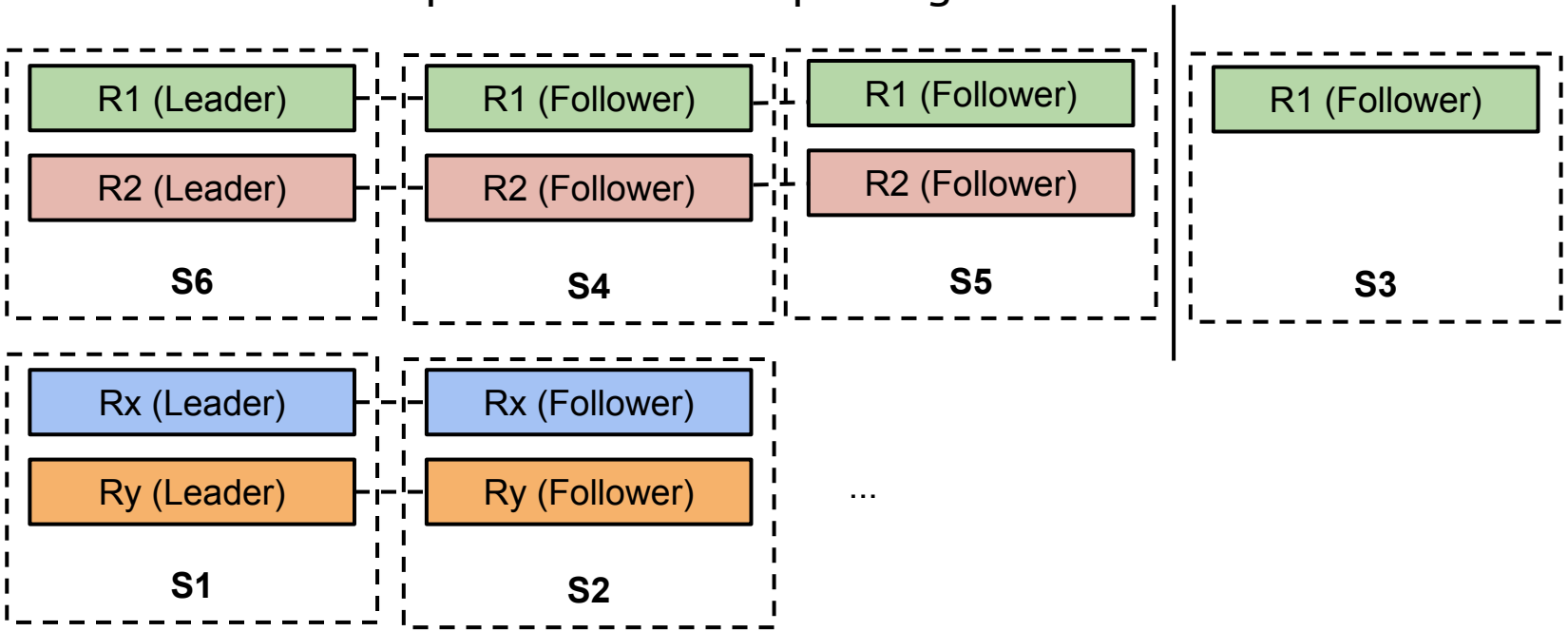



(2)



# An abnormal situation...

(3) After N rounds of split or membership changes...



# Introduce Range Epoch

- Epoch(Range X) := {ConfVer, SplitVer}
- Every configuration change in Range X will increase the ConfVer
- Every split occurs in Range X will increase the SplitVer
- Let’s say Epoch(R1) >= Epoch(R2), if and only if:
  - ConfVer(R1) >= ConfVer(R2) && SplitVer(R1) >= SplitVer(R2)
- Larger epoch always win

# Challenges

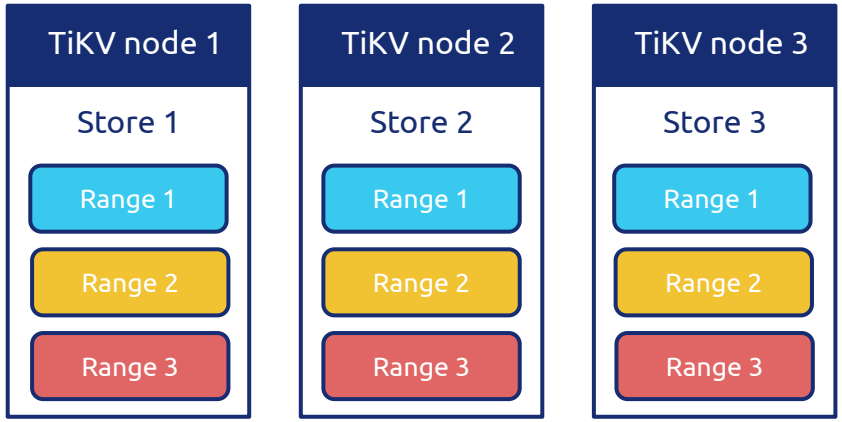
- Is Raft slow?
  - How to make Raft fast?
- How to safely split/merge data Range (aka Raft group)?
- How to move data around without pain?
  - Load balancing
  - Application-Aware data placement

# Data movement

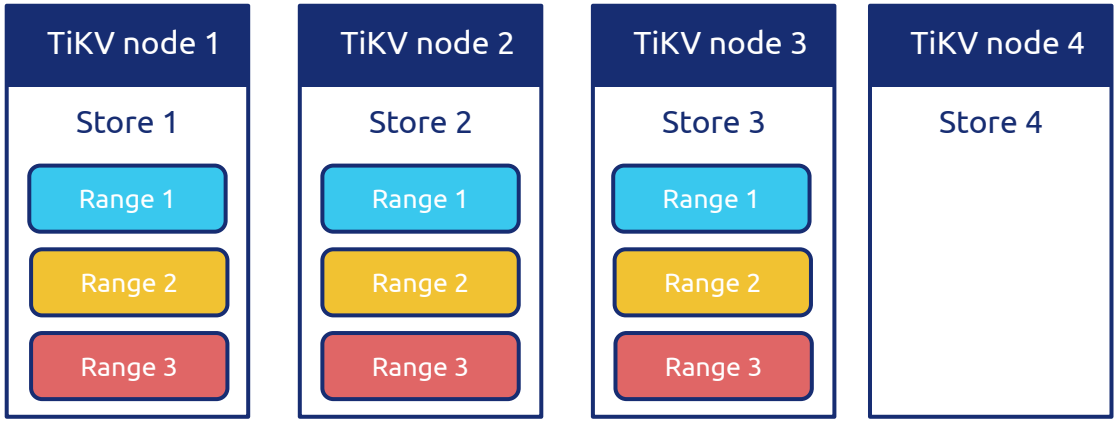
**Kudos to Raft Membership Change Protocol**

1. Add a new member to Raft group (new replica on new node)
2. Remove an old member from Raft group (remove one of the replicas)

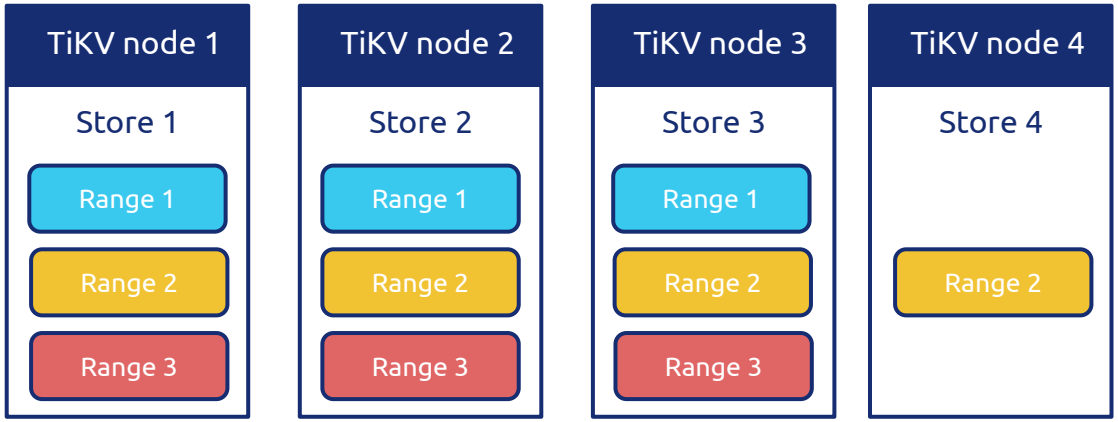
# Data movement



# Data movement

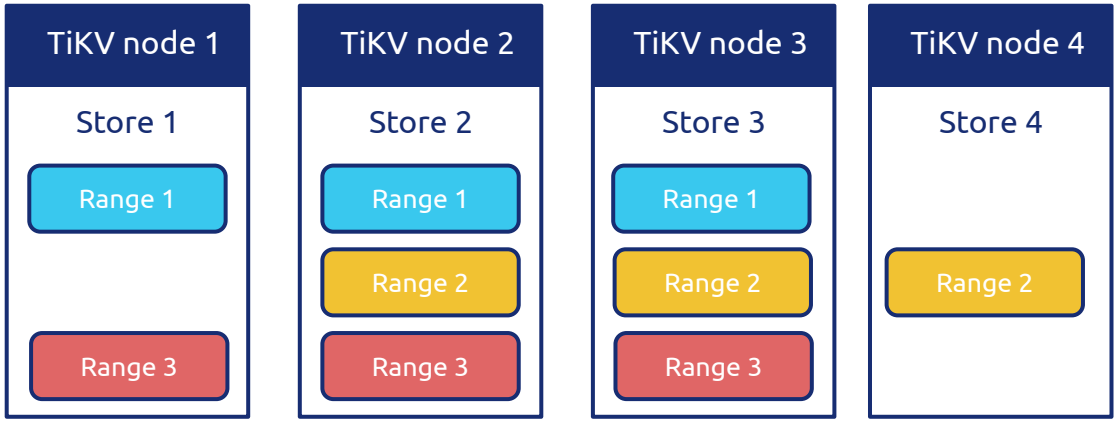


# Data movement



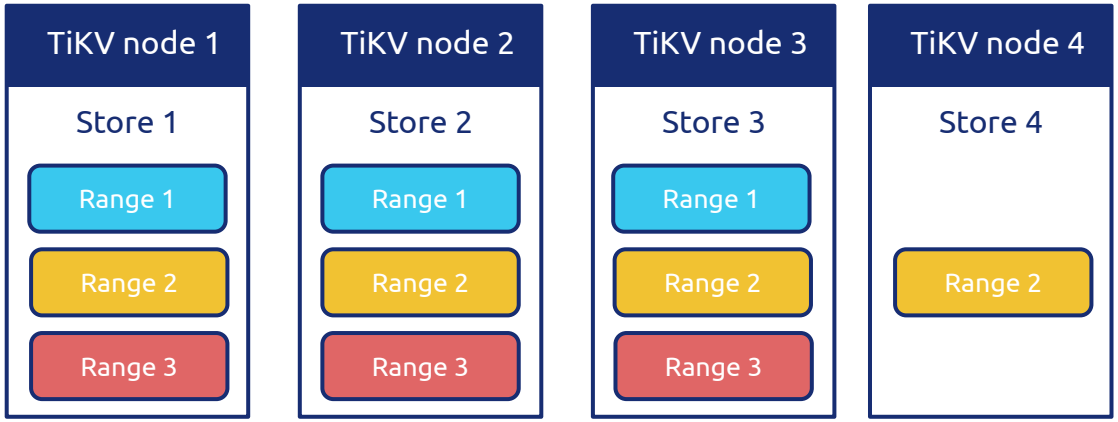

Add a new member to Raft group (new replica on new node)

# Data movement

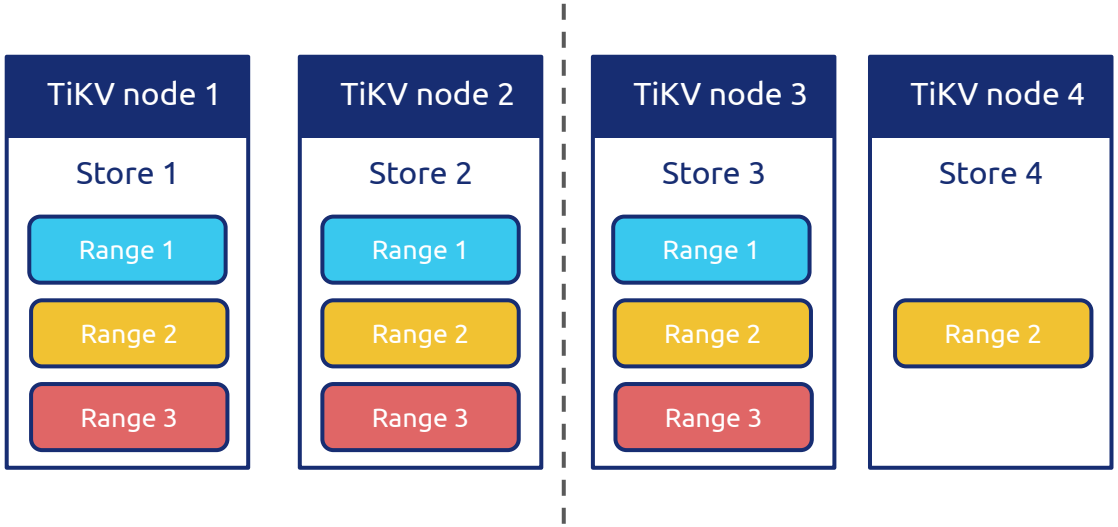



Remove a old member from Raft group (remove one of the replicas)

# Wait, there's a tricky moment...

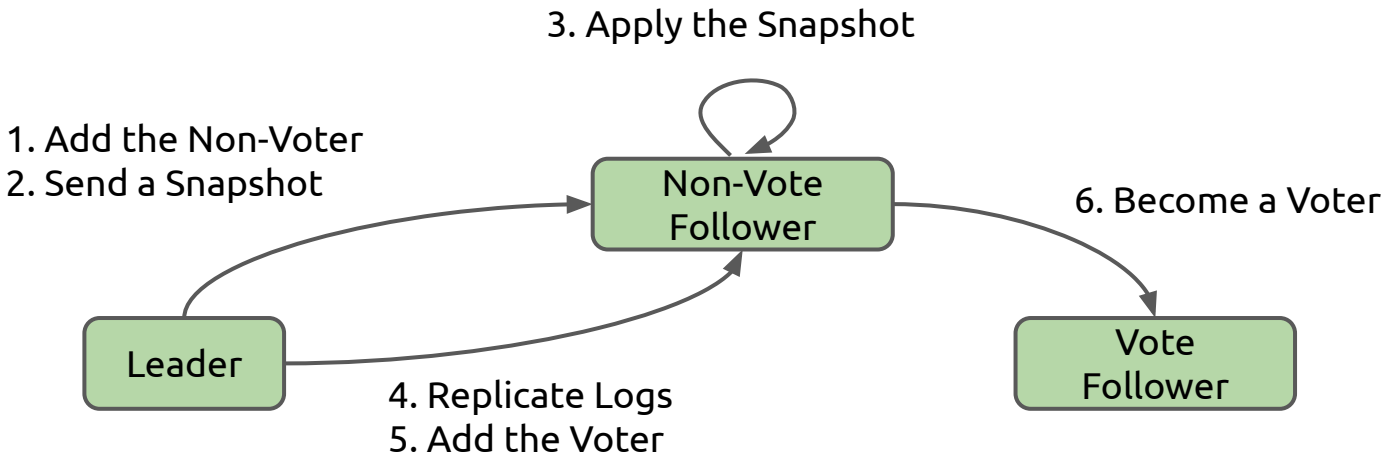



# At this moment, we have 4 peers for Range1 (when Range2 in Store4 is syncing data)

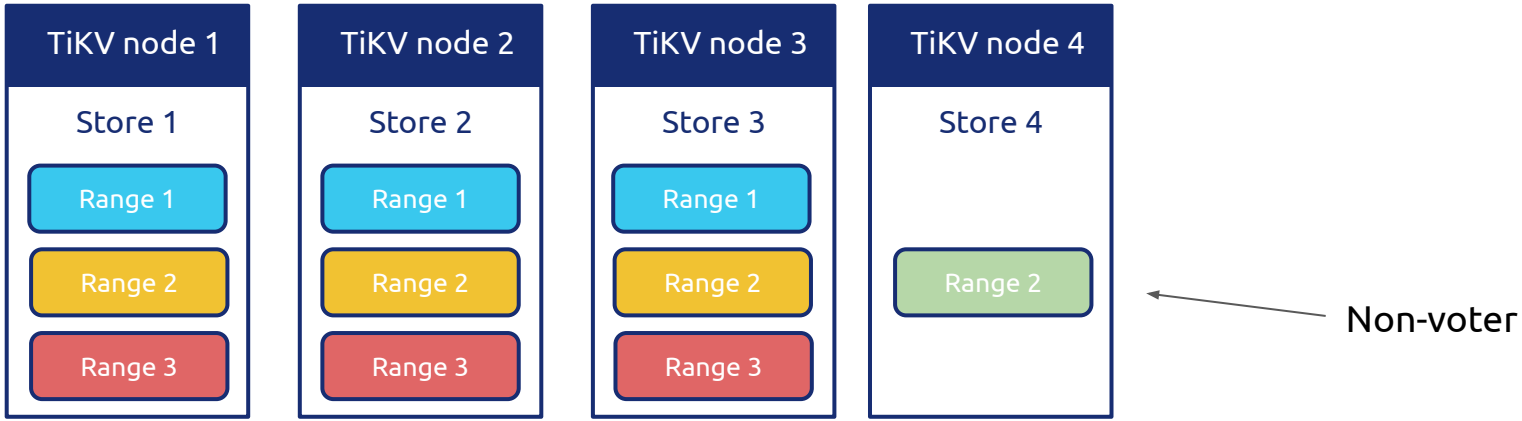



# What if network partition happens? Opps. (Quorum is 3)

# Optimization: Non-Vote Follower



# Optimization: Non-Vote Follower




So, new peer will be non-voter first, so the number of voter is still 3
(yes, I know there's still have a very short period of time we will have 4 voters)

# Overview

- TiDB's Multi-Raft Architecture
  - Key design points
    - Is Raft slow?
      - Optimizations on Real-World Raft
    - How to scale-out?
- How to apply Raft into a HTAP system
  - Raft Learner
  - DeltaMerge Engine: a mutable colunmar storage engine

# HTAP: Hybrid Transactional/Analytical Processing

# HTAP: Hybrid Transactional/**Analytical** Processing

# Rule #1 for an OLAP system:

## Columnar storage is a necessary

# Columnar VS Row-based

**Rowstore**

| id | name | age |
|---|---|---|
| 0962 | Jane | 30 |
| 7658 | John | 45 |
| 3589 | Jim | 20 |
| 5523 | Susan | 52 |

SELECT AVG(age) FROM emp;

**Columnstore**

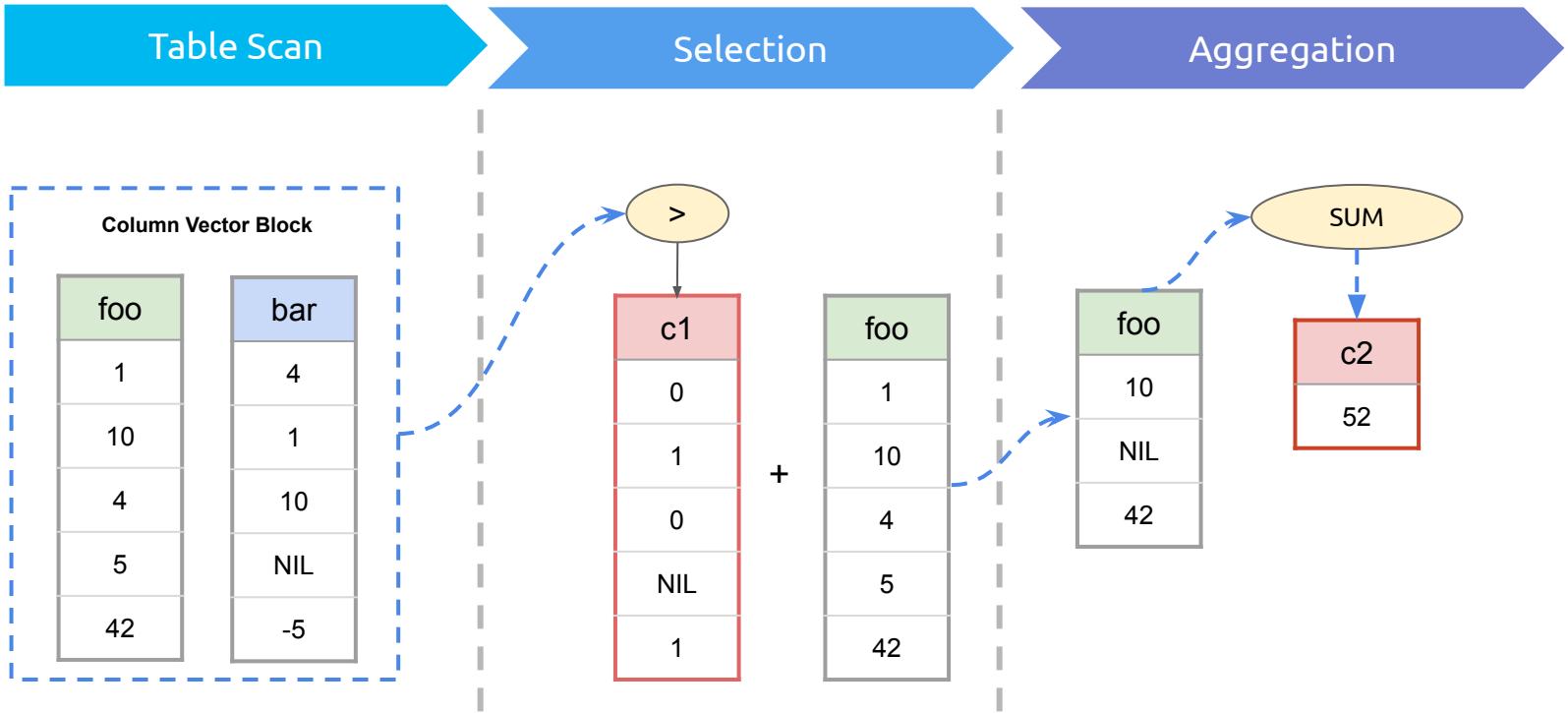
| id |
|---|
| 0962 |
| 7658 |
| 3589 |
| 5523 |

| name |
|---|
| Jane |
| John |
| Jim |
| Susan |

| age |
|---|
| 30 |
| 45 |
| 20 |
| 52 |

# Vectorized Processing

SELECT SUM(foo) FROM Table WHERE foo > bar



# A Popular Solution

- Different types of databases for different workload
  - OLTP specialized database for transactional data
  - Hadoop / analytical database for historical data

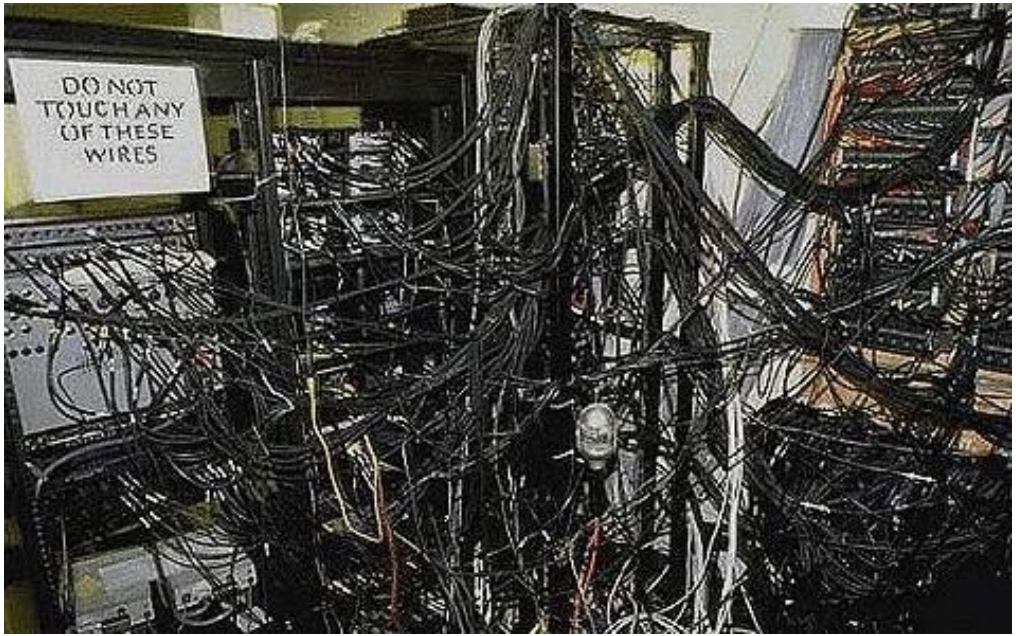
- Offload transactional data via ETL process into Hadoop / analytical database
  - Periodically, usually per day

# Good enough?

# Data processing stack today


- Data cannot be seamlessly connected between different data stores

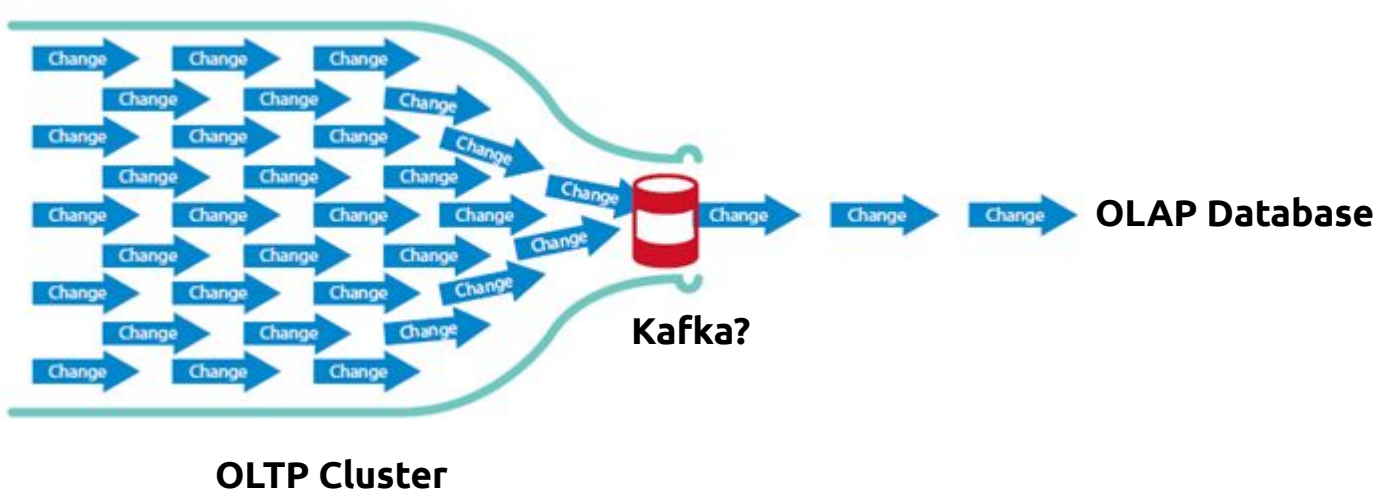
# Data processing stack today




- Adding a new data source is hard
- It is painful to maintain multiple systems.

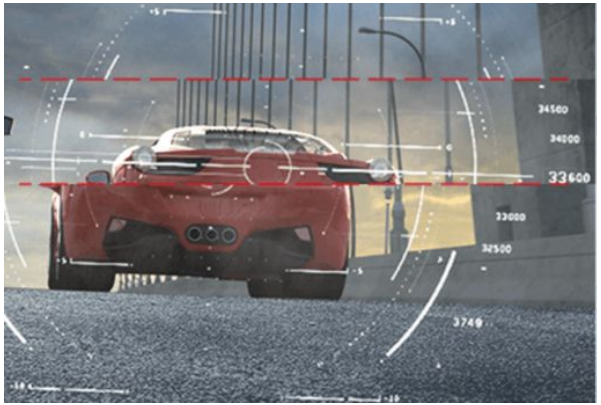
# Data processing stack today

- Data pipeline may easily become the bottleneck



# Data processing stack today

- The process of ETL usually loses the the transaction info

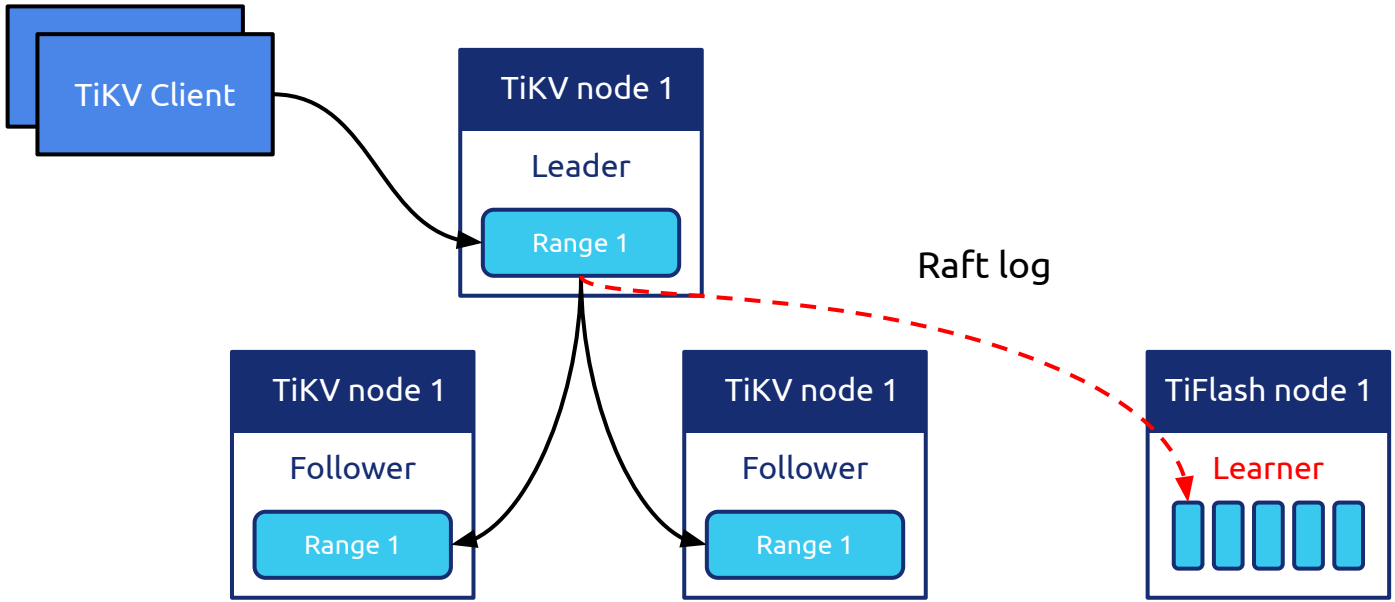
# Challenges of HTAP systems:

**Data synchronization shouldn't be bottleneck**

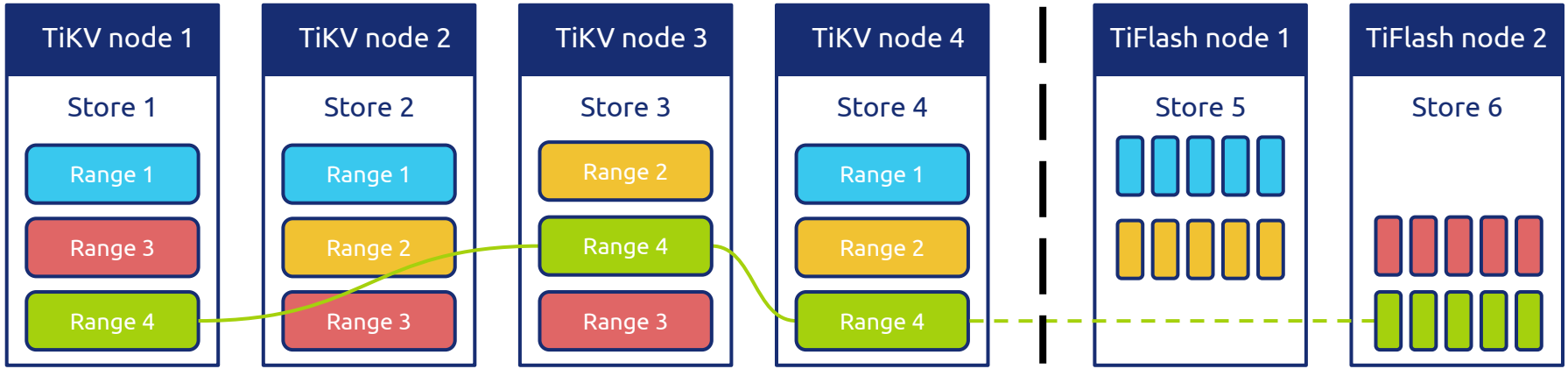
# Low-cost Data Replication

- We found that TiKV's Multi-Raft architecture is a great foundation!
- Data is replicated to TiFlash nodes via ***Raft Learner***
  - Extended Raft consensus algorithm
    - Remember Non-voting member :)
  - Async replication
    - Only sync Raft log
    - Almost zero overhead to OLTP workload

# Low-cost Data Replication



# Low-cost Data Replication



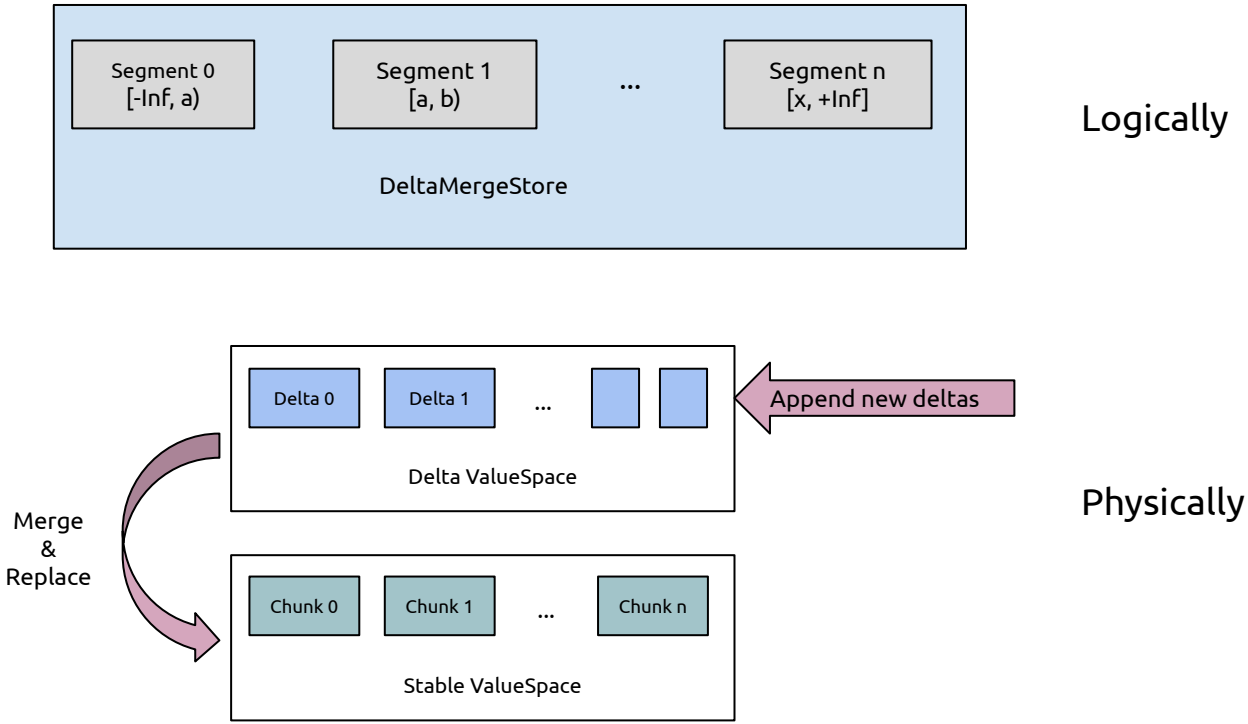
# Benefits of replicating log via Raft Learner

- Inherited TiKV's elastic scalability
  - *Multi-Raft rocks!*
- What's more, beautiful thing happened:
  - Transaction information (e.g. MVCC Version) is in Raft log
    - That means we can still keep transaction isolation level in the AP part

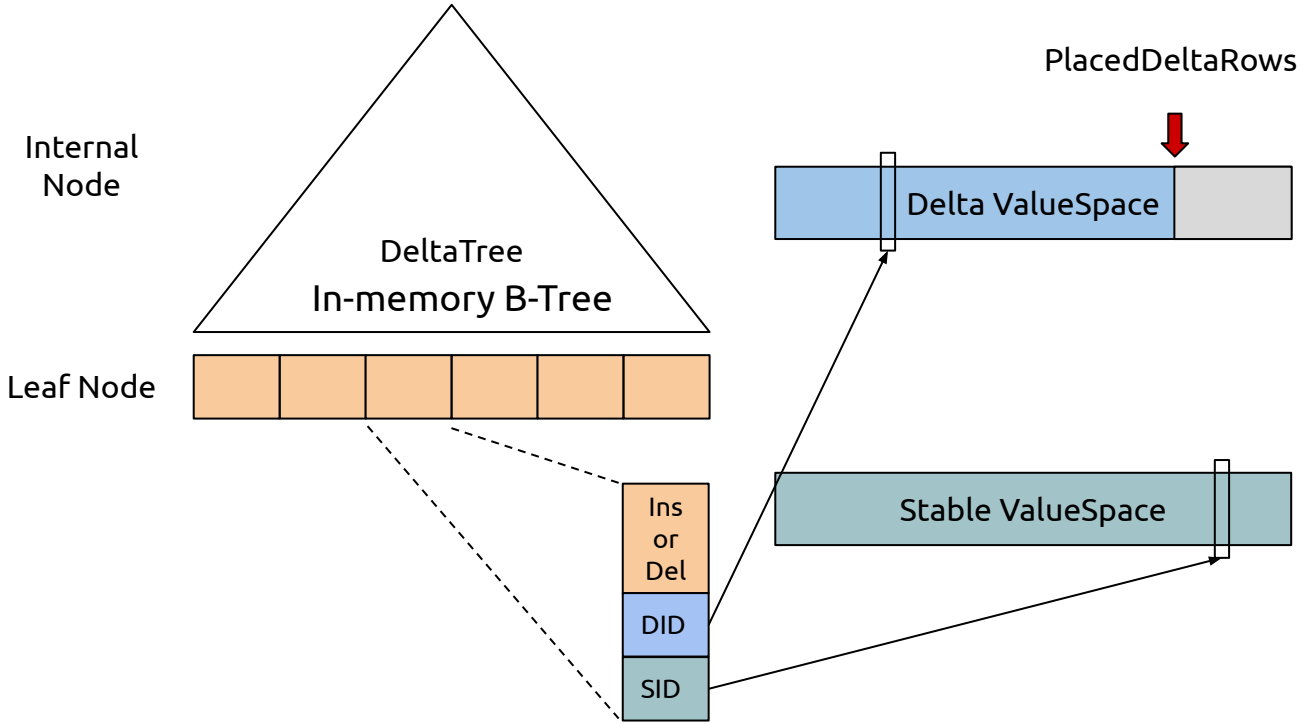
# Wait!

- How to support efficient (in-place) insert/update/delete operations on columnar storage?
  - without sacrificing in Read

# DeltaTree Engine

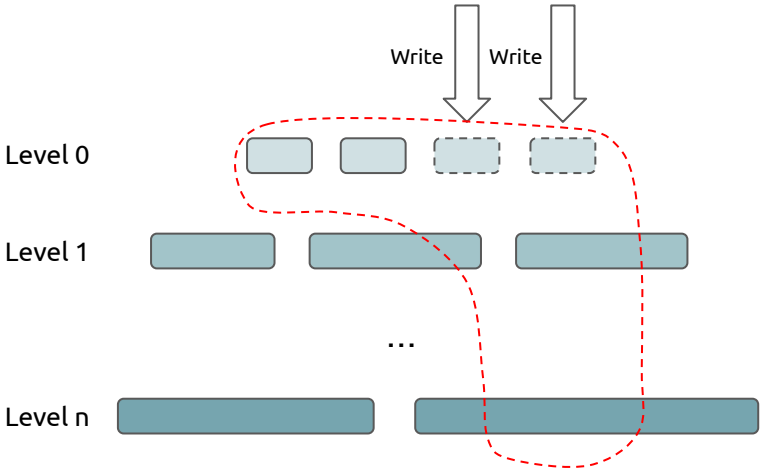


# DeltaTree Engine



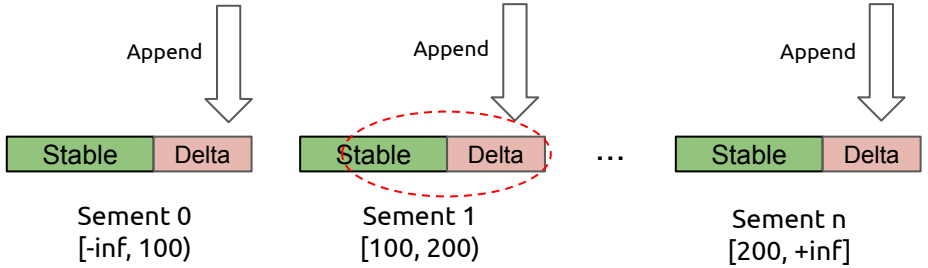
# DeltaTree Engine

SELECT ... WHERE x BETWEEN (150, 160)




LSM-Tree

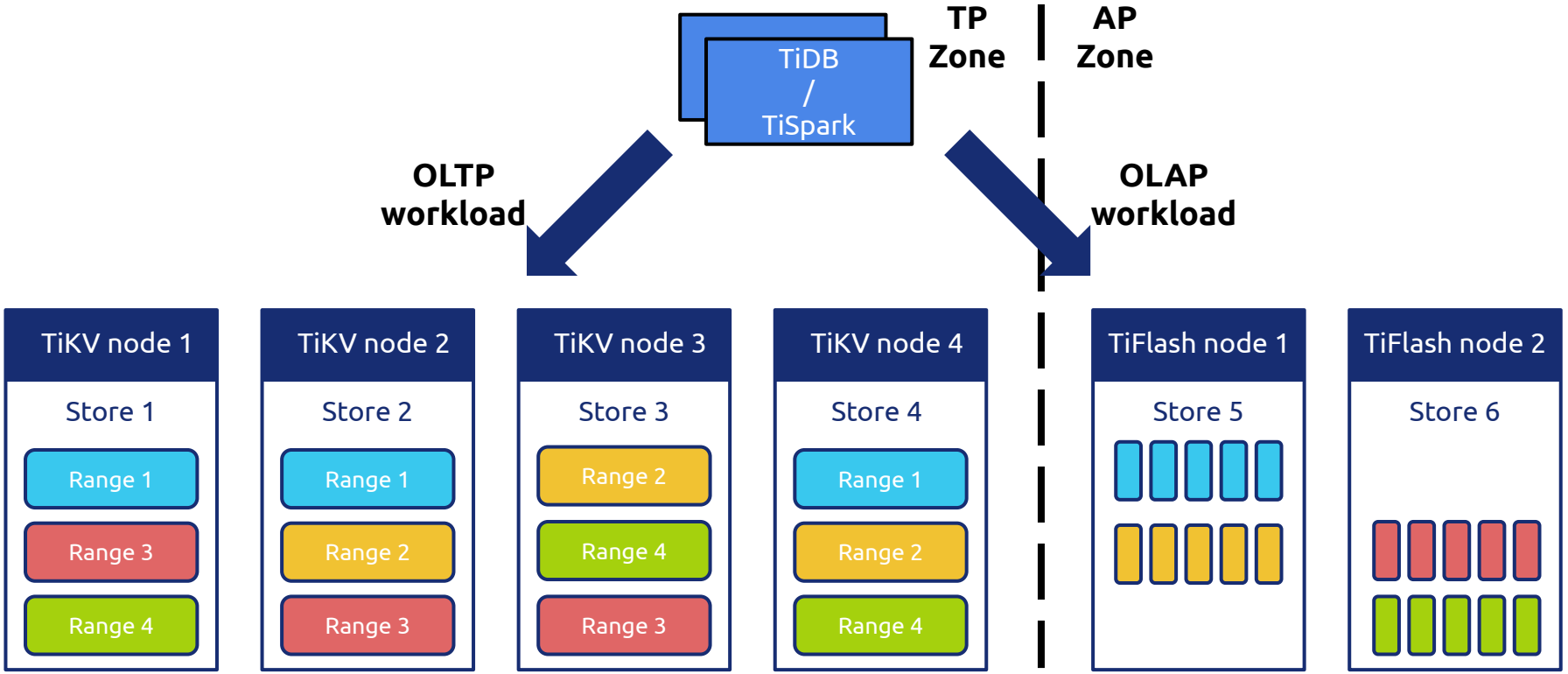
VS




DeltaMerge

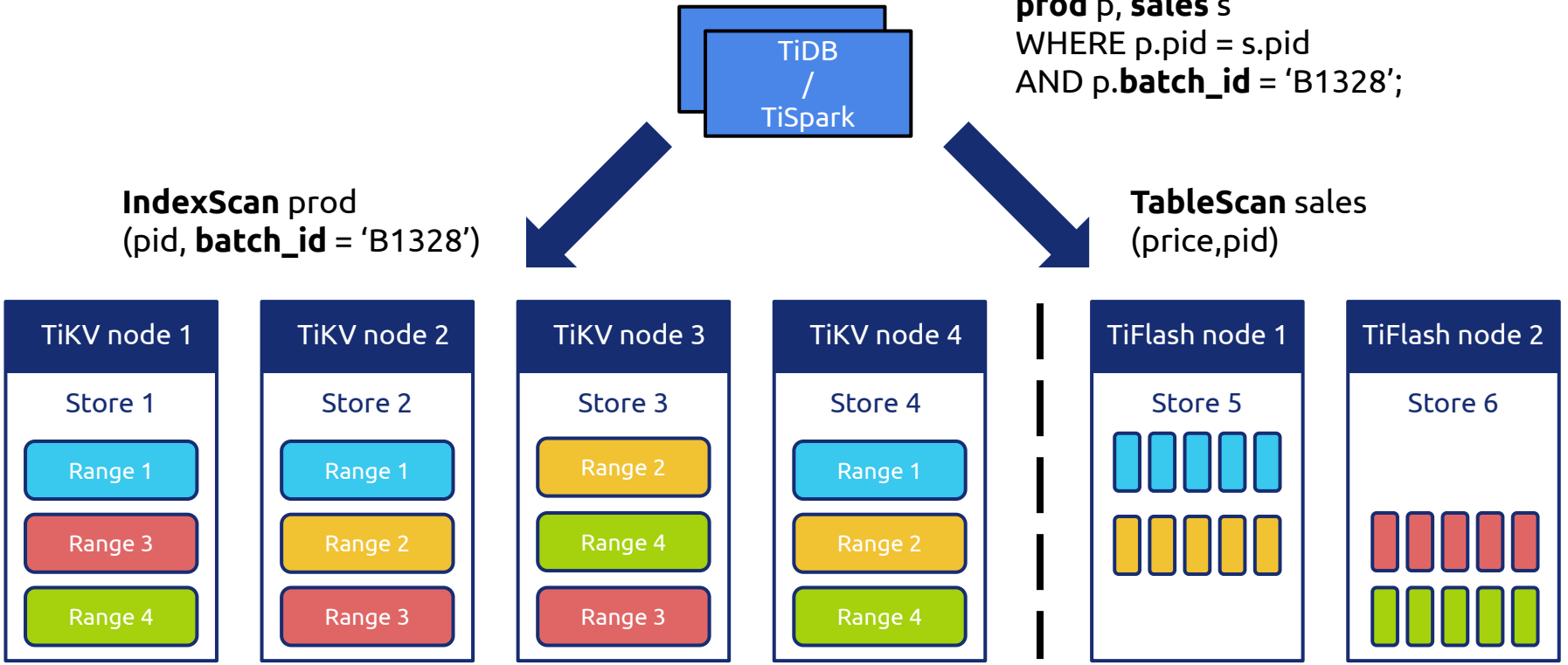
# Combine all components together!

# Isolation


TiDB
/
TiSpark
TP
Zone
AP
Zone
OLTP
workload
OLAP
workload
TiKV node 1
Store 1
Range 1
Range 3
Range 4
TiKV node 2
Store 2
Range 1
Range 2
Range 3
TiKV node 3
Store 3
Range 2
Range 4
Range 3
TiKV node 4
Store 4
Range 1
Range 2
Range 4
TiFlash node 1
Store 5
TiFlash node 2
Store 6

# Integration

SELECT AVG(s.price) FROM
**prod** p, **sales** s
WHERE p.pid = s.pid
AND p.**batch_id** = 'B1328';



- TPC-H 20G
- TiDB + TiFlash vs Aurora

| | TiDB+TiFlash | Aurora |
|---|---|---|
| q1 | 3.54 | 3min 45.73s |
| q2 | 4.37 | 12.15 |
| q3 | 3.73 | 40.78 |
| q4 | 4.96 | 13.66 |
| q5 | 5.05 | 3min 1.35s |
| q6 | 0.54 | 41.05 |
| q7 | 6.77 | 25.04 |
| q8 | 2.33 | 6min 27.99s |
| q9 | 18.32 | 2min 32.51s |
| q10 | 10.90 | 2min 33.69 |
| q11 | 1.68 | 5.19 |
| q12 | 3.38 | 1min 4.24 |
| q13 | 15.83 | 1min 49.82 |
| q14 | 1.94 | 46.26 |
| q15 | 3.41 | 1min 28.73 |
| q16 | 2.60 | 17.38 |
| q17 | 18.20 | 1min 25.77 |
| q18 | 11.86 | 2min 27.52 |
| q19 | 9.80 | 1min 4.29 |
| q20 | 4.85 | 21.35 |
| q21 | 12.21 | 10min 30.32 |
| q22 | 14.94 | 2.69 |