# TIDB架构原理简介

黄阳全

# 目录

# TIDB介绍

# 数据库技术发展演进

## 2008 年以前

### 单机关系型（SQL）

- 背景：应用最为广泛的数据库；能很好的解决复杂的数据运算及表间处理；多用于银行、电信等传统行业复杂业务逻辑场景中，以 Oracle 为代表
- 挑战：**成本高**，随着数据量增加，只能通过购买更贵更好的服务器；**无法线性扩容**，海量数据下处理能力大幅下降

## 2008 年至 2013 年

### 分布式非关系型（NoSQL）

- 背景：随着搜索 / 社交的发展，数据量爆发增长，传统数据库高成本，无法线性扩容问题日益突显；分布式及 NoSQL 开始快速发展，如 MongoDB，HBase
- 挑战：擅长简单读写，**无法处理交易类数据及复杂业务逻辑**的特性限制其在非互联网领域的发展

## 2013 年以后

### 分布式关系型（NewSQL）

- 背景：随着互联网向银行、电信、电力等方向的渗透，传统行业数据量迅速提升，需要同时满足低成本、线性扩容及能够处理交易类事务的新型数据库，大数据的存储刚需不可避免
- 挑战：基于 Google Spanner/F1 论文，基础软件最前沿的领域之一，技术门槛最高

**NewSQL: 兼具 NoSQL 扩展性又不丧失传统关系型数据库 ACID 特性的分布式数据库**

# TIDB是什么

TiDB PingCap公司自主研发且开源的一个分布式 、强一致、具备水平扩展能力的关系型数据库

关键特性（目标）

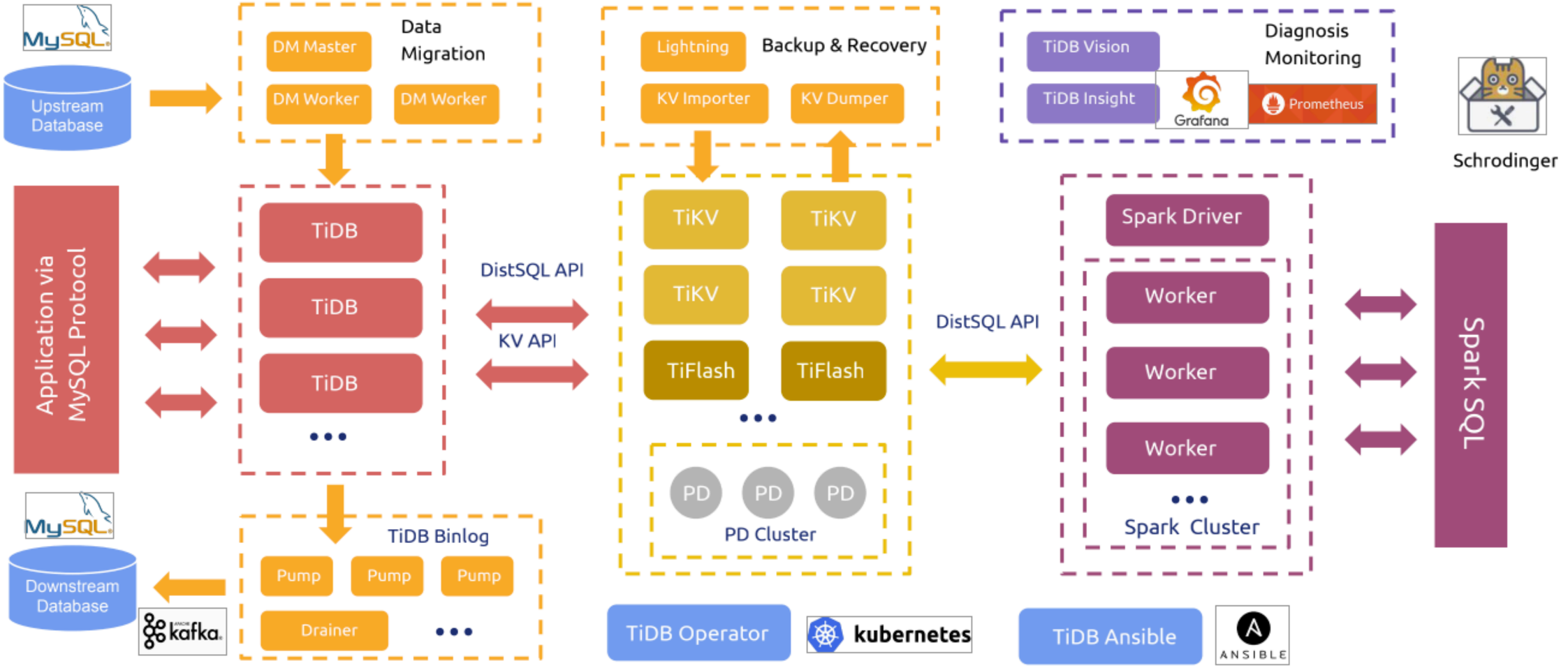
- 水平扩展
- 高可用
- ACID事务
- MySQL协议

# 一个疑问：why tidb

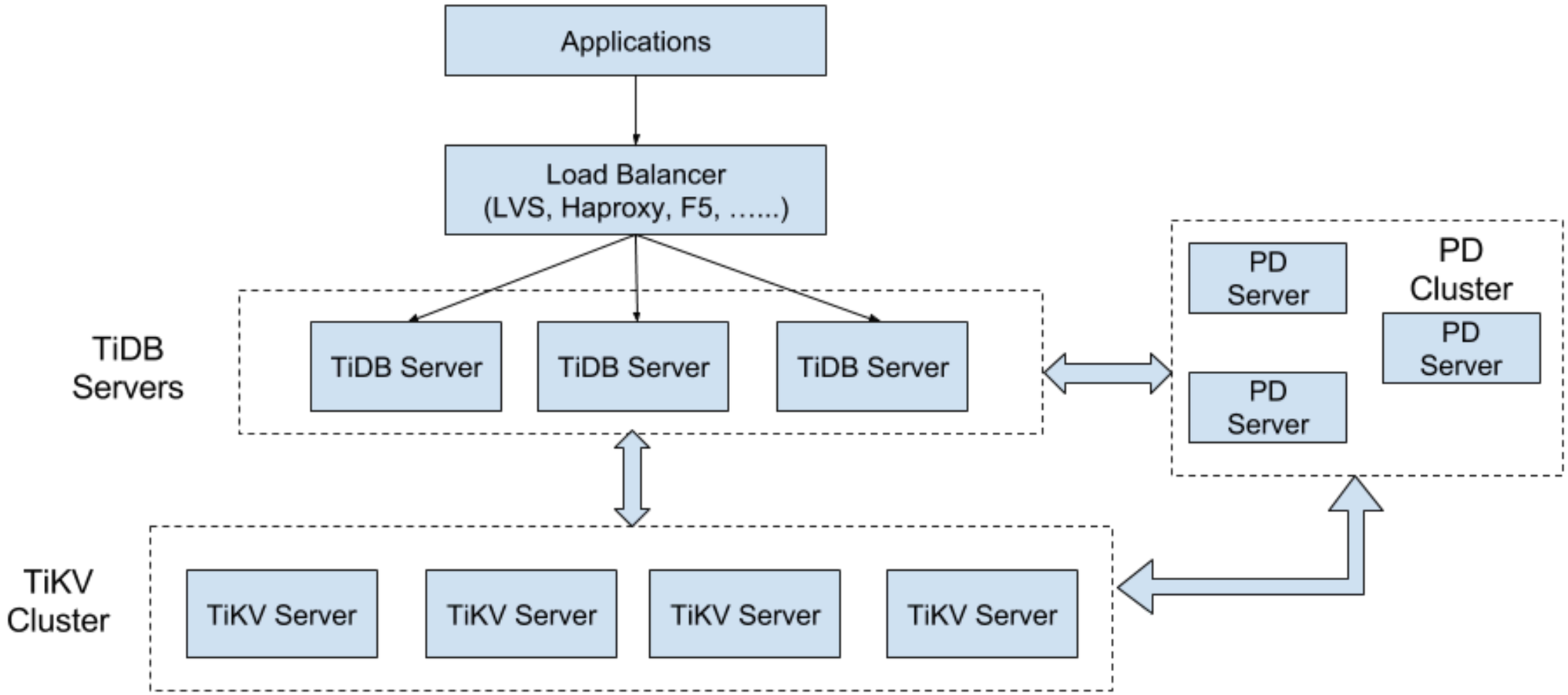
做一个增强版的Mysql Server 或者 Proxy，它不香嘛？

# TIDB整体架构

# TiDB 整体架构


MySQL
Upstream Database
Data Migration
DM Master
DM Worker
DM Worker
Lightning
Backup & Recovery
KV Importer
KV Dumper
TiDB Vision
TiDB Insight
Diagnosis
Monitoring
Grafana
Prometheus
Schrodinger
Application via MySQL Protocol
TiDB
TiDB
TiDB
DistSQL API
KV API
TiKV
TiKV
TiKV
TiKV
TiFlash
TiFlash
PD
PD
PD
PD Cluster
DistSQL API
Spark Driver
Worker
Worker
Worker
Spark Cluster
Spark SQL
MySQL
Downstream Database
kafka
TiDB Binlog
Pump
Pump
Pump
Drainer
TiDB Operator
kubernetes
TiDB Ansible
ANSIBLE

# TIDB核心架构


Applications
Load Balancer
(LVS, Haproxy, F5, ……)
TiDB
Servers
TiDB Server
TiDB Server
TiDB Server
PD
Server
PD
Cluster
PD
Server
PD
Server
TiKV
Cluster
TiKV Server
TiKV Server
TiKV Server
TiKV Server

# 分布式存储引擎

# 做一个分布式存储引擎

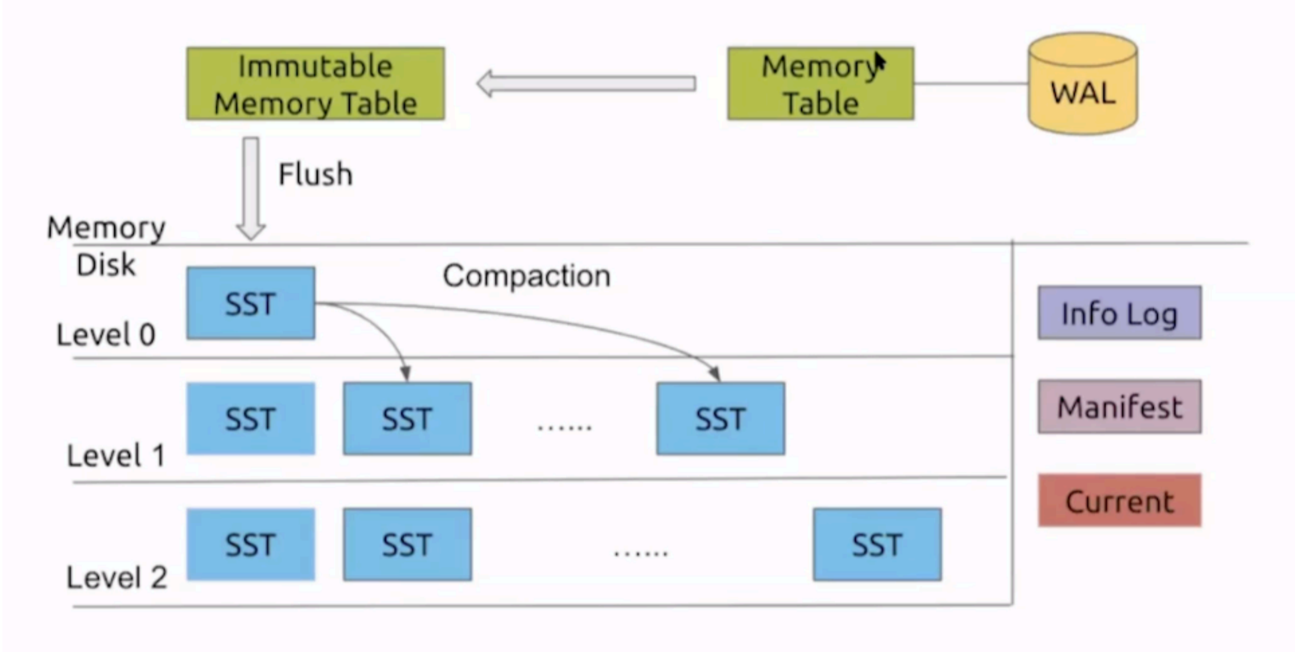
- 数据存在哪?
- 如何做容灾?
- 写入速度如何保证?
- 如何方便读取?
- 能否原子的修改多条记录?
- 如何做扩容?
- 如何保证多副本的数据一致性?
- …

# 做一个分布式存储引擎

第一步：一个性能强大的单机存储引擎

# RocksDB

- LSM-Tree（顺序写，压缩友好）
- 速度快
- 有原子性的batch write
- 多线程的 compaction 优化



做一个分布式存储引擎

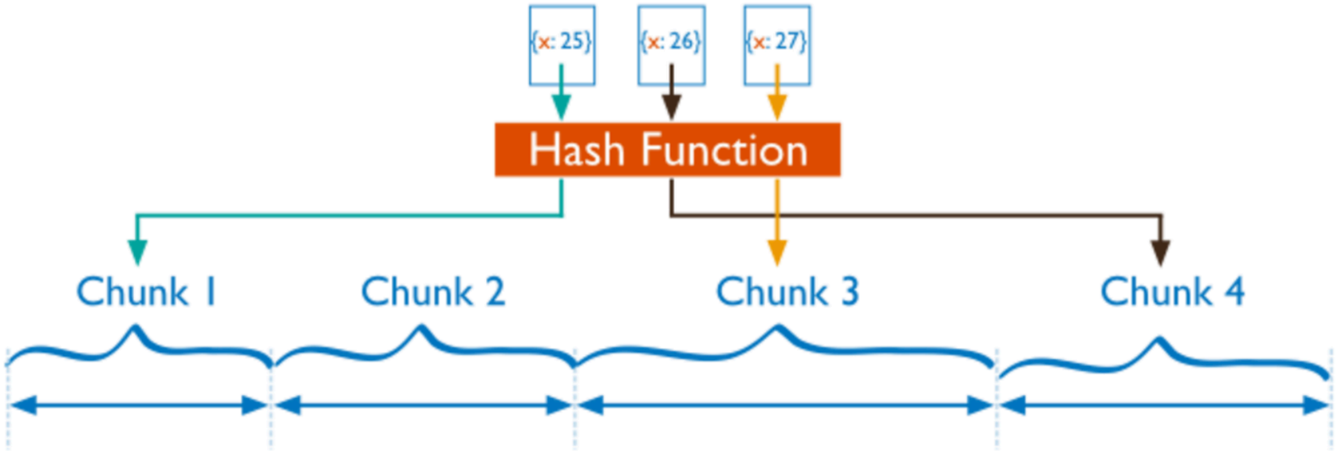
第二步：将数据分片

# Hash分片

特点
随机分布，均匀

优点
- 分散写压力
- 利于随机读

缺点
- 不能range scan
- Re-hash 代价较高

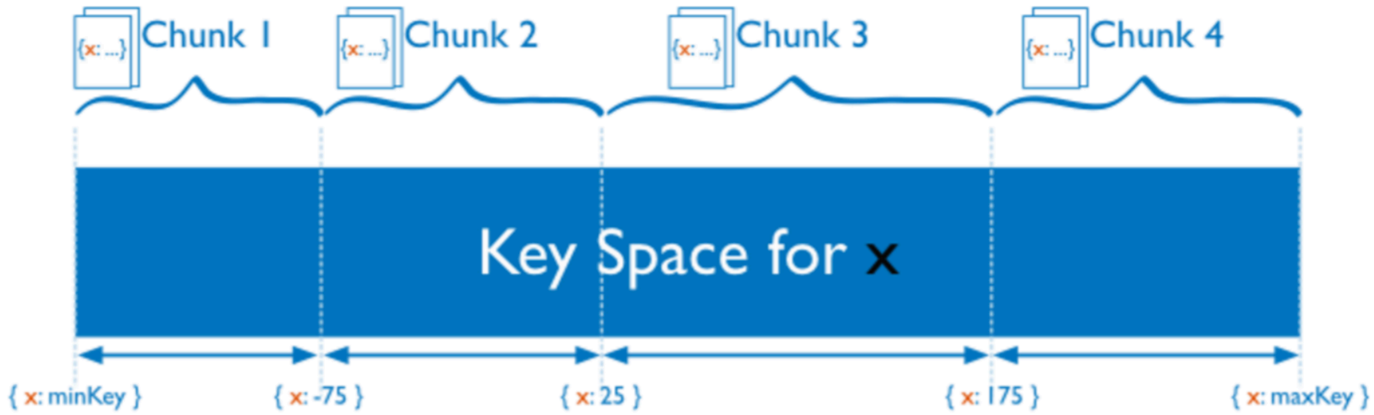


# Range分片

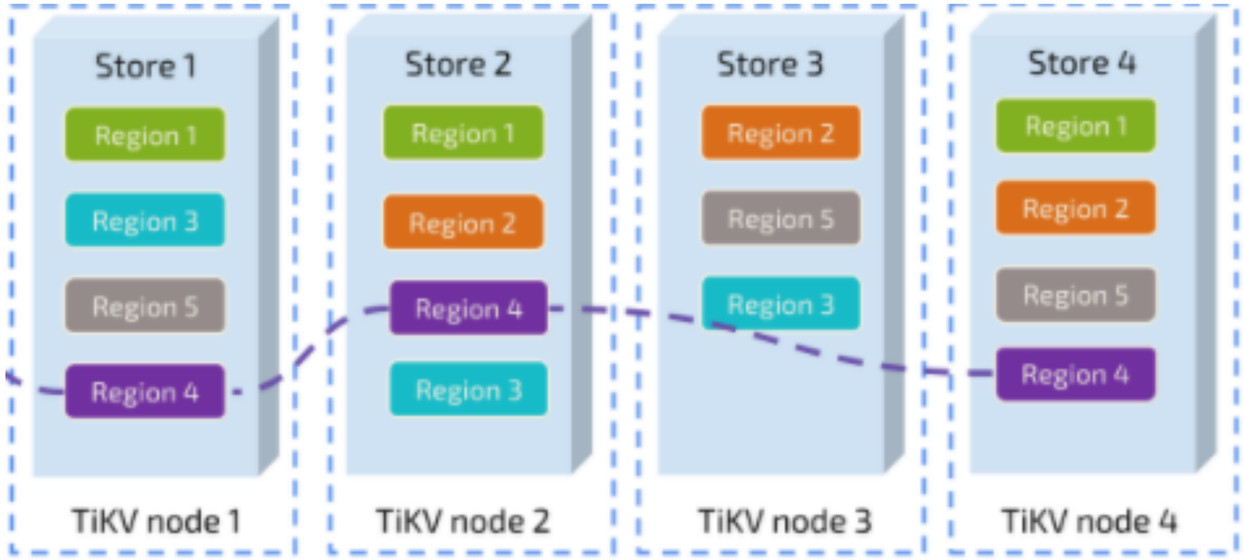
特点
顺序存储，有序

优点
- 便于range scan
- 方便做动态扩展

缺点
- 顺序写是不太友好，压力永远在最后的Region



# 做一个分布式存储引擎



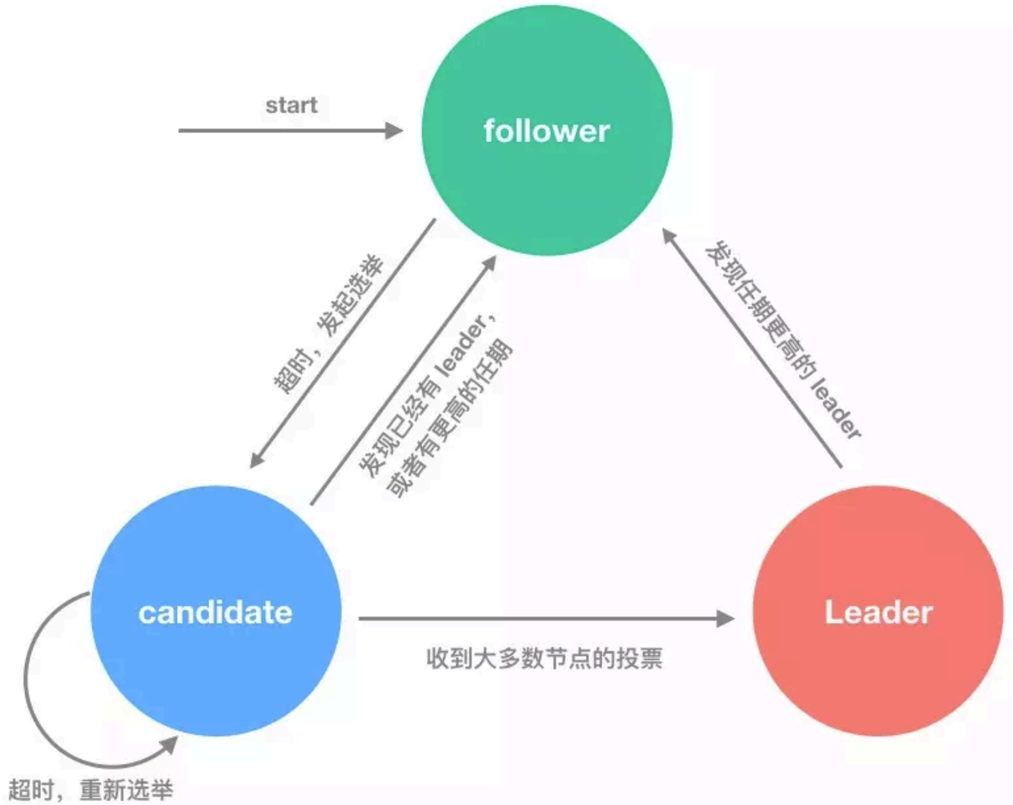

一台机器上多个Region，加一个新node，只需要让部分region挪到新node就可以了

做一个分布式存储引擎

# 第三步：Region 高可用

# Raft协议

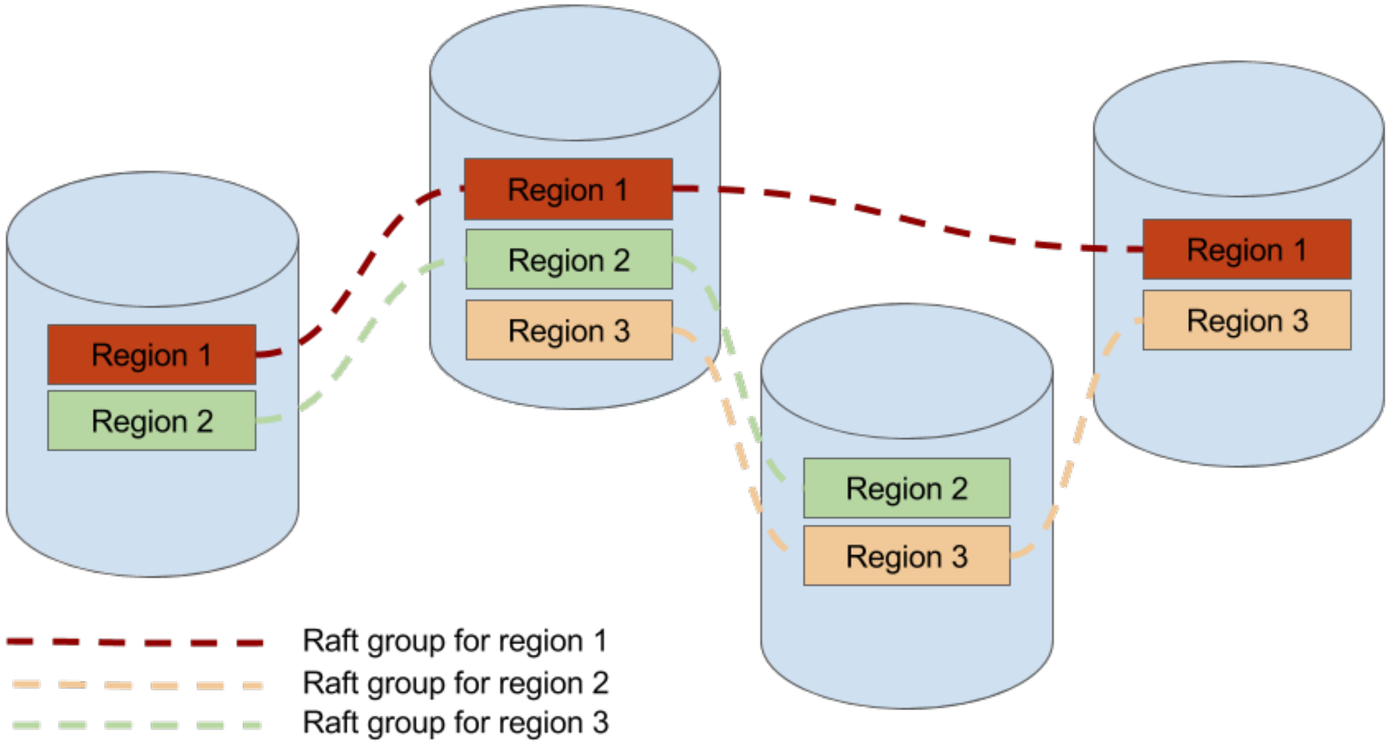
- 多数派写入
- 强Leader
- Leader选举
- 成员变更



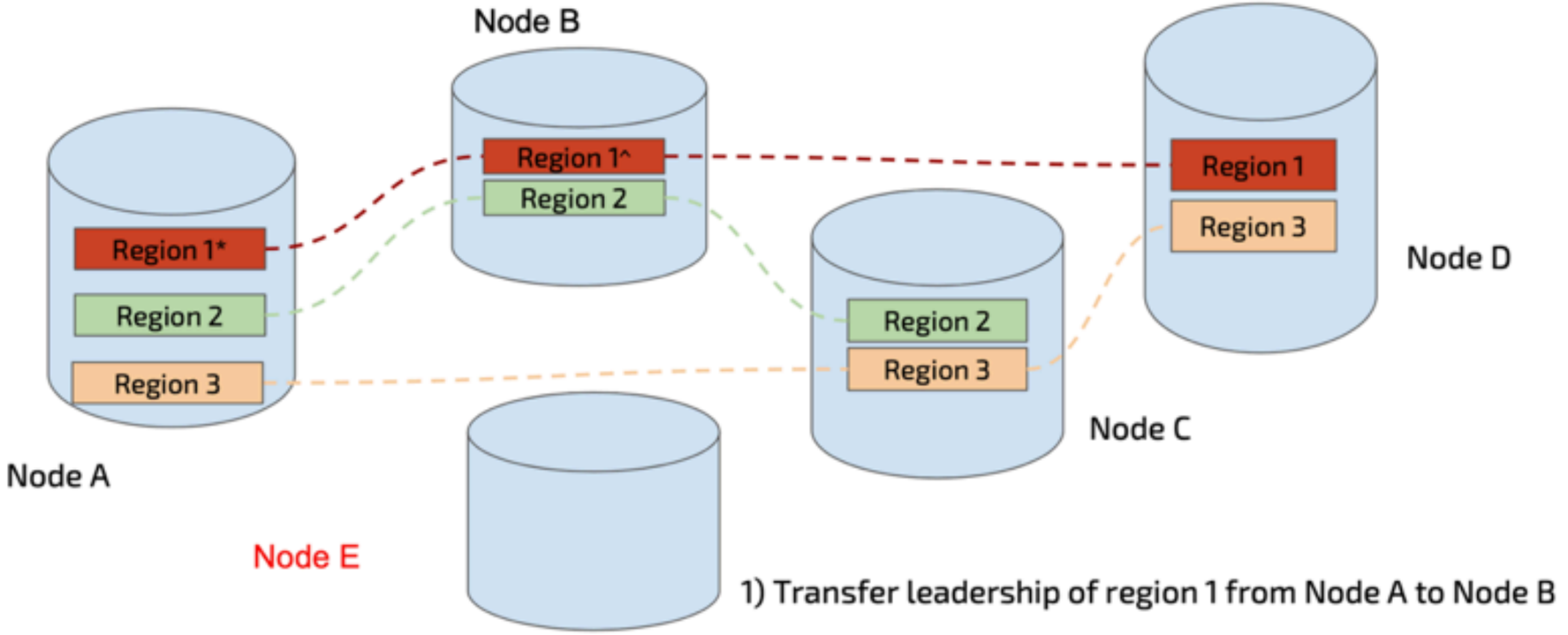
做一个分布式存储引擎

# 第四步：可伸缩

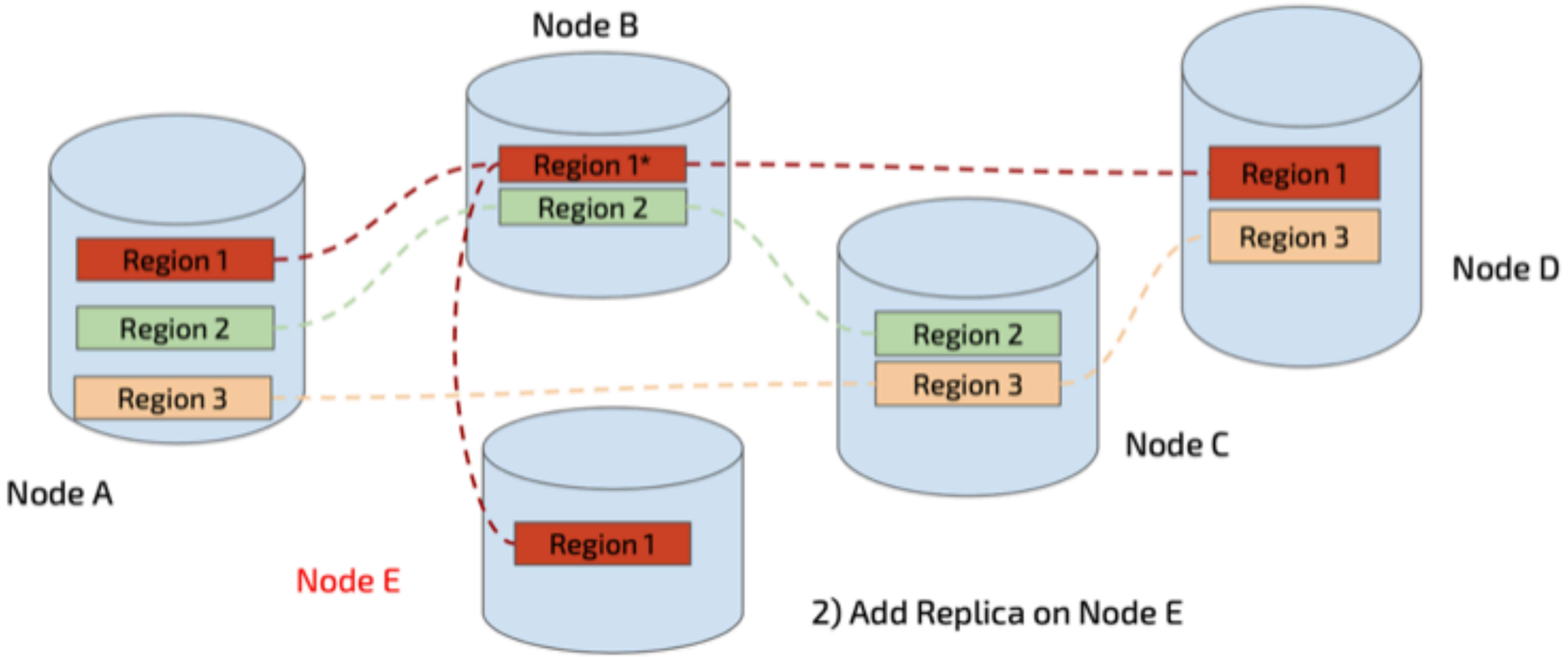
# 做一个分布式存储引擎


Region 1
Region 2
Region 1
Region 2
Region 3
Region 1
Region 3
Region 2
Region 3
Raft group for region 1
Raft group for region 2
Raft group for region 3

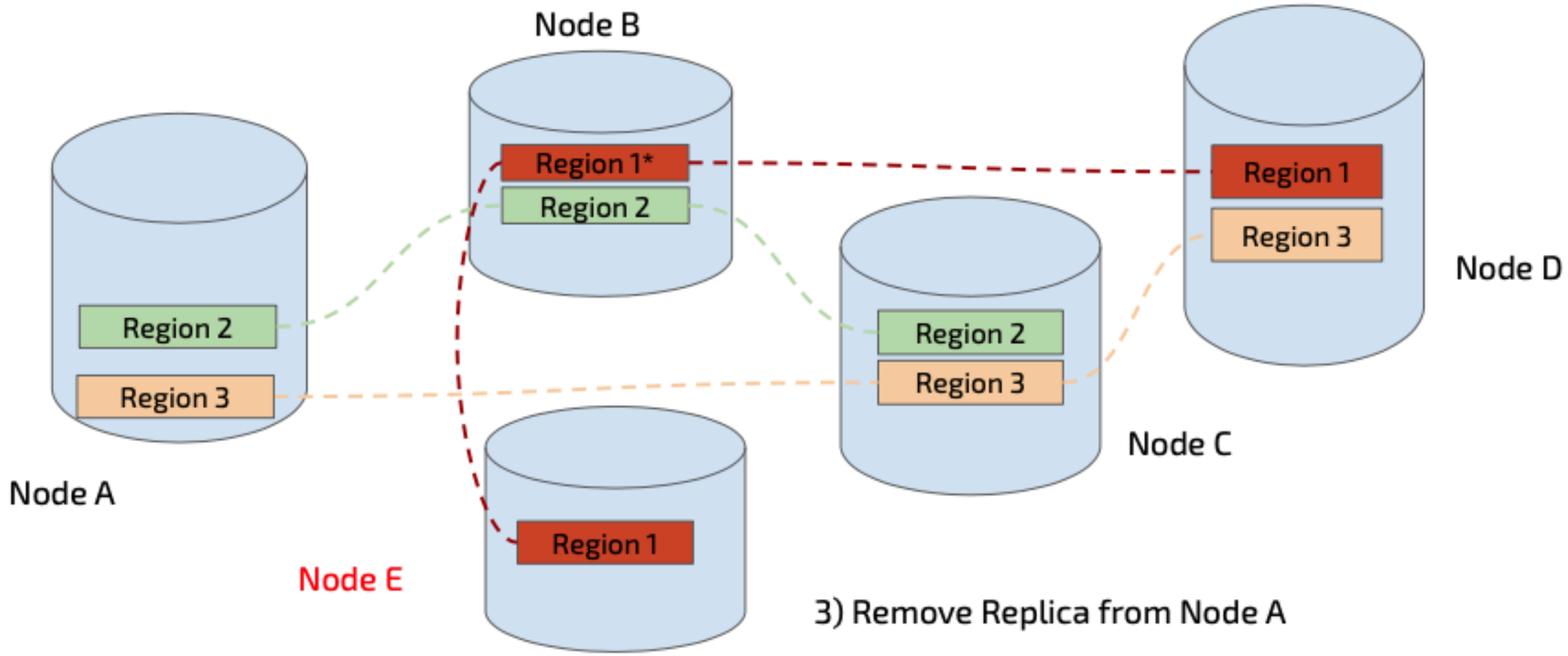
# 添加一个节点


Node B
Region 1^
Region 2
Region 1
Region 3
Node D
Region 1*
Region 2
Region 3
Node A
Region 2
Region 3
Node C
Node E
1) Transfer leadership of region 1 from Node A to Node B

# 平衡节点


Node B
Region 1*
Region 2
Region 1
Region 2
Region 3
Node A
Region 1
Region 3
Node D
Region 2
Region 3
Node C
Region 1
Node E
2) Add Replica on Node E

# 删除原node的region

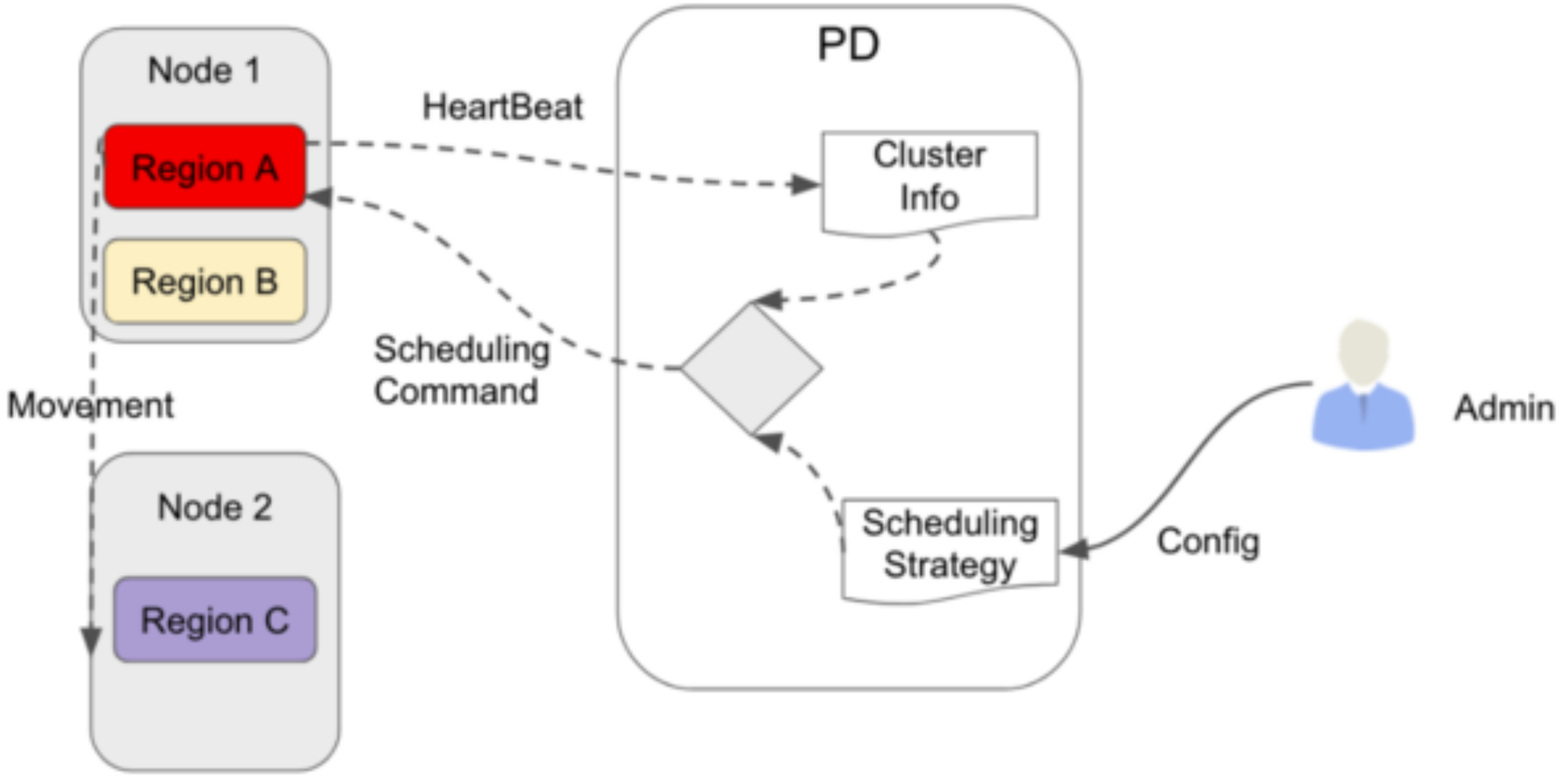


做一个分布式存储引擎

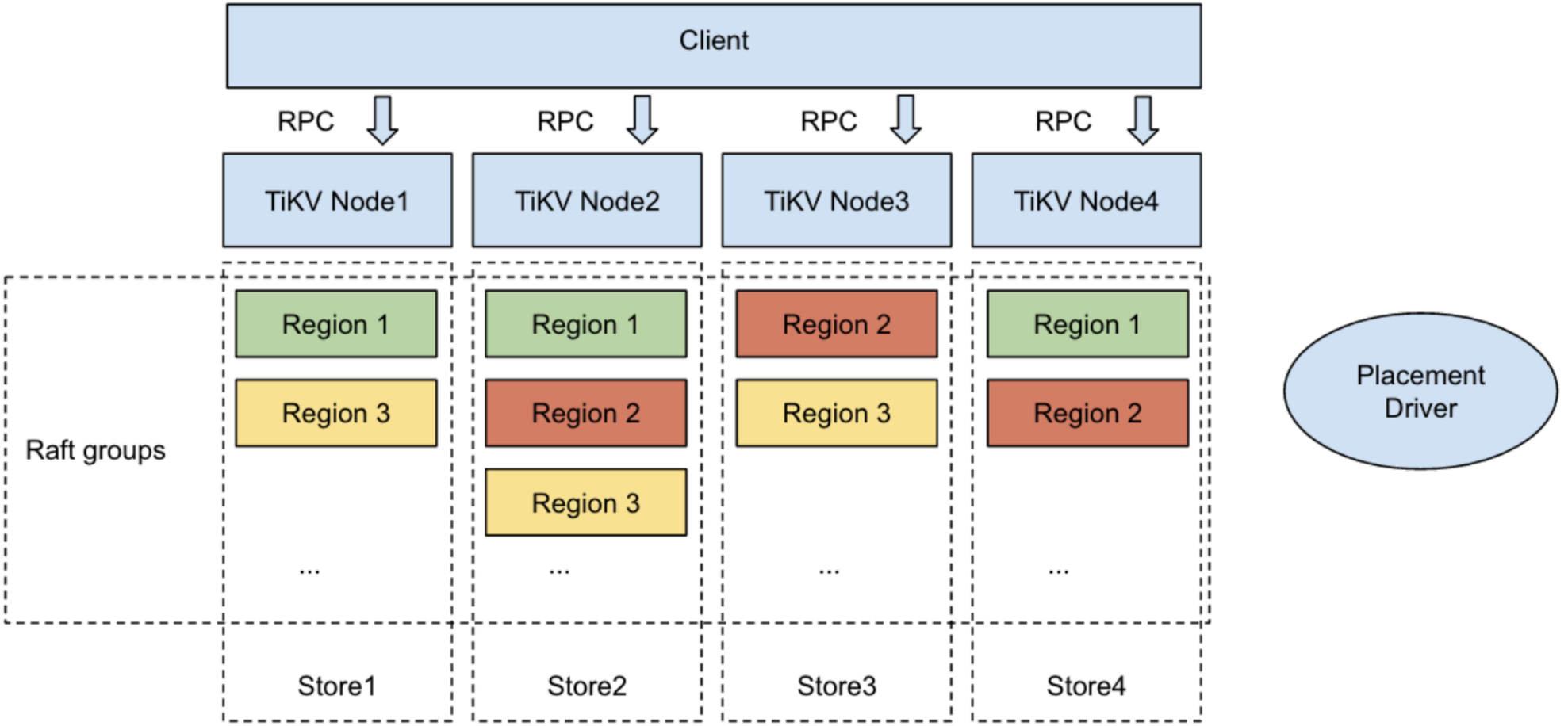
第五步：自动化

# 做一个分布式存储引擎

- 提供上帝视角
- 保存集群元数据
- 维持副本数量
- 均衡存储
- 控制调度速度

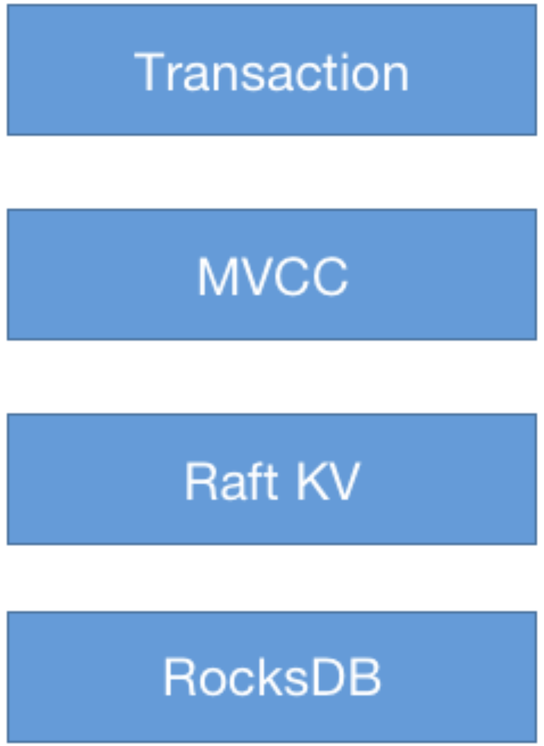


# TIKV全图


Client
RPC
RPC
RPC
RPC
TiKV Node1
TiKV Node2
TiKV Node3
TiKV Node4
Raft groups
Region 1
Region 3
...
Region 1
Region 2
Region 3
...
Region 2
Region 3
...
Region 1
Region 2
...
Store1
Store2
Store3
Store4
Placement Driver

# 事务的实现

受GooGle percolator启发
基本的去中心化的两阶段提交
乐观锁、悲观锁
默认的隔离级别 Snapshot

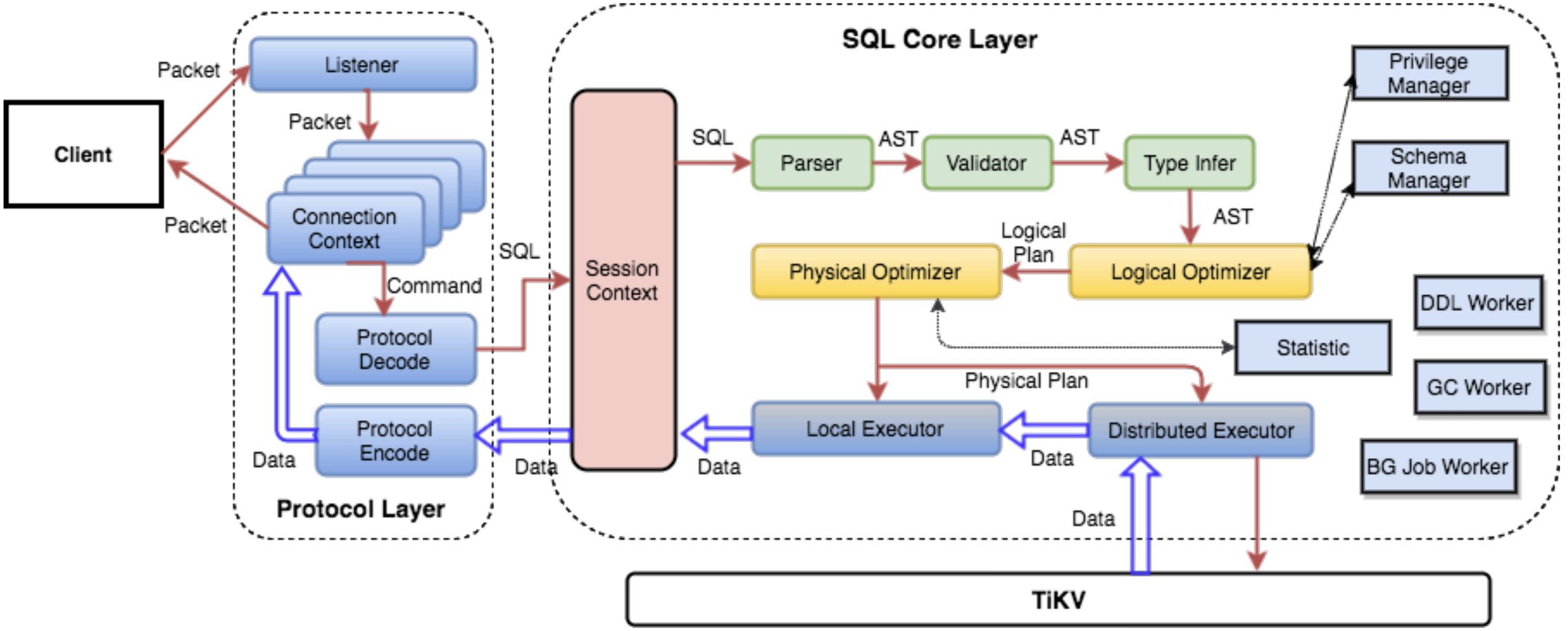


# 分布式计算引擎

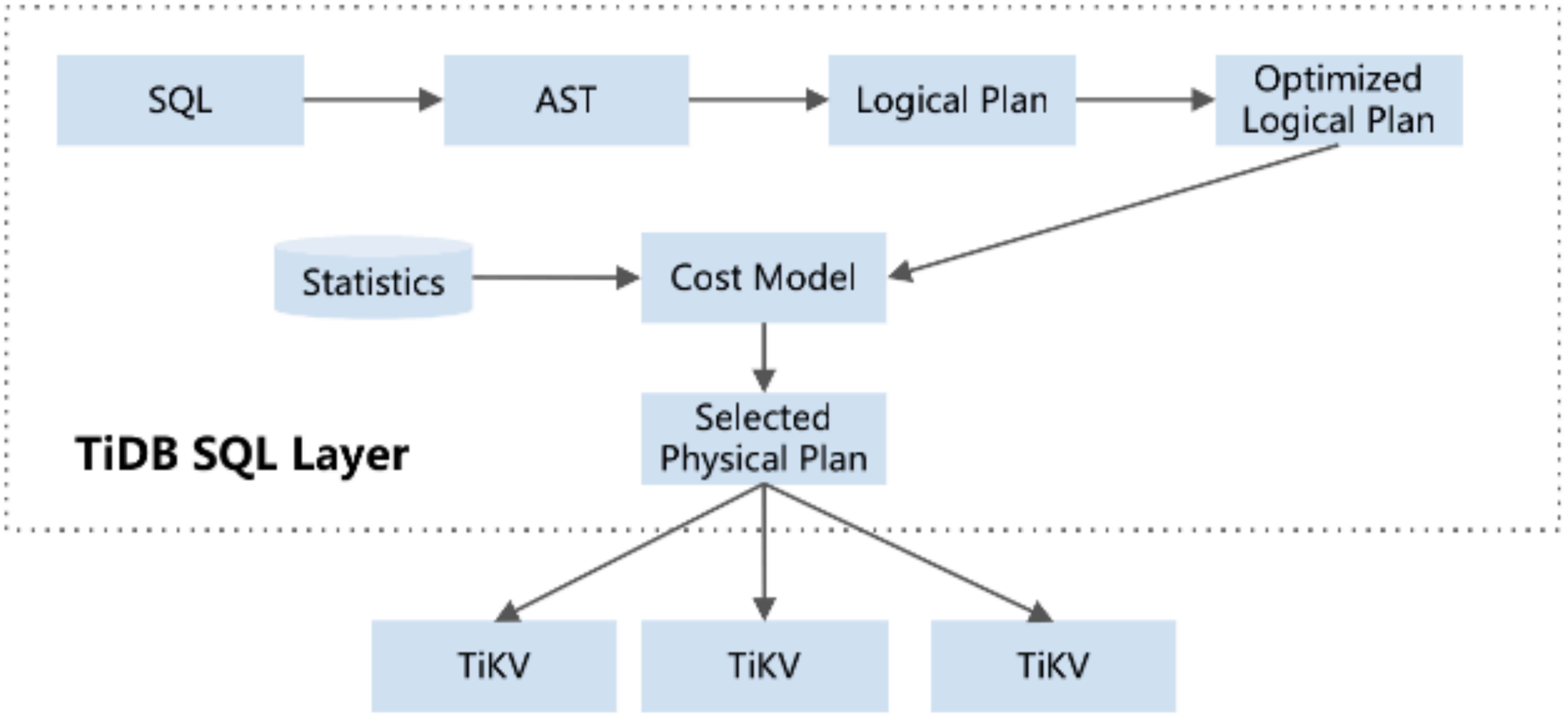
# 主要内容

- SQL引擎结构
- Query处理流程
- 计算优化

# SQL 引擎架构


Client
Packet
Listener
Packet
Connection Context
Packet
Command
Protocol Decode
Protocol Encode
Data
Protocol Layer
SQL
Session Context
Data
SQL Core Layer
SQL
Parser
AST
Validator
AST
Type Infer
AST
Logical Optimizer
Logical Plan
Physical Optimizer
Physical Plan
Local Executor
Data
Distributed Executor
Data
Data
Privilege Manager
Schema Manager
Statistic
DDL Worker
GC Worker
BG Job Worker
TiKV

# SQL处理流程


SQL
AST
Logical Plan
Optimized Logical Plan
Statistics
Cost Model
Selected Physical Plan
TiDB SQL Layer
TiKV
TiKV
TiKV

# 关系模型到KV模型的映射

SQL Model

| id (primary) | name (unique) | age( non-unique) | score |
|---|---|---|---|
| 1 | Bob | 12 | 99 |
| 2 | John | 14 | 98 |

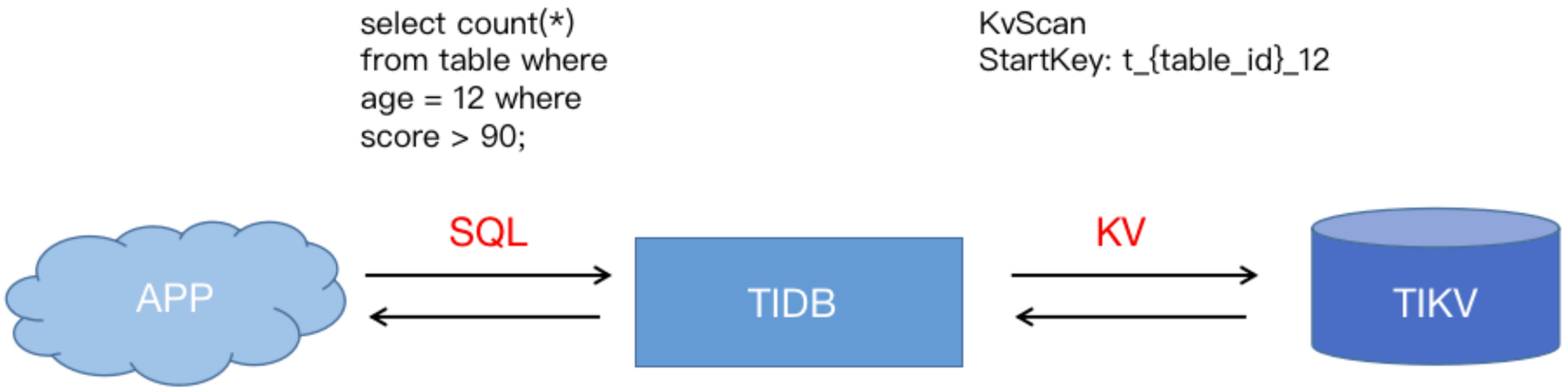
KV Model in TiKV

| index_type | key | value |
|---|---|---|
| primary_index | t_{table_id}_1 | (Bob,12,99) |
| primary_index | t_{table_id}_2 | (John,14,98) |
| name (unique) | t_{table_id}_Bob | 1 |
| name (unique) | t_{table_id}_John | 1 |
| age (non-unique) | t_{table_id}_12_1 | null |
| age (non-unique) | t_{table_id}_14_2 | null |

# 关系模型到KV模型的映射

```
select count(*) from table where id > 12 where score = 98;
```

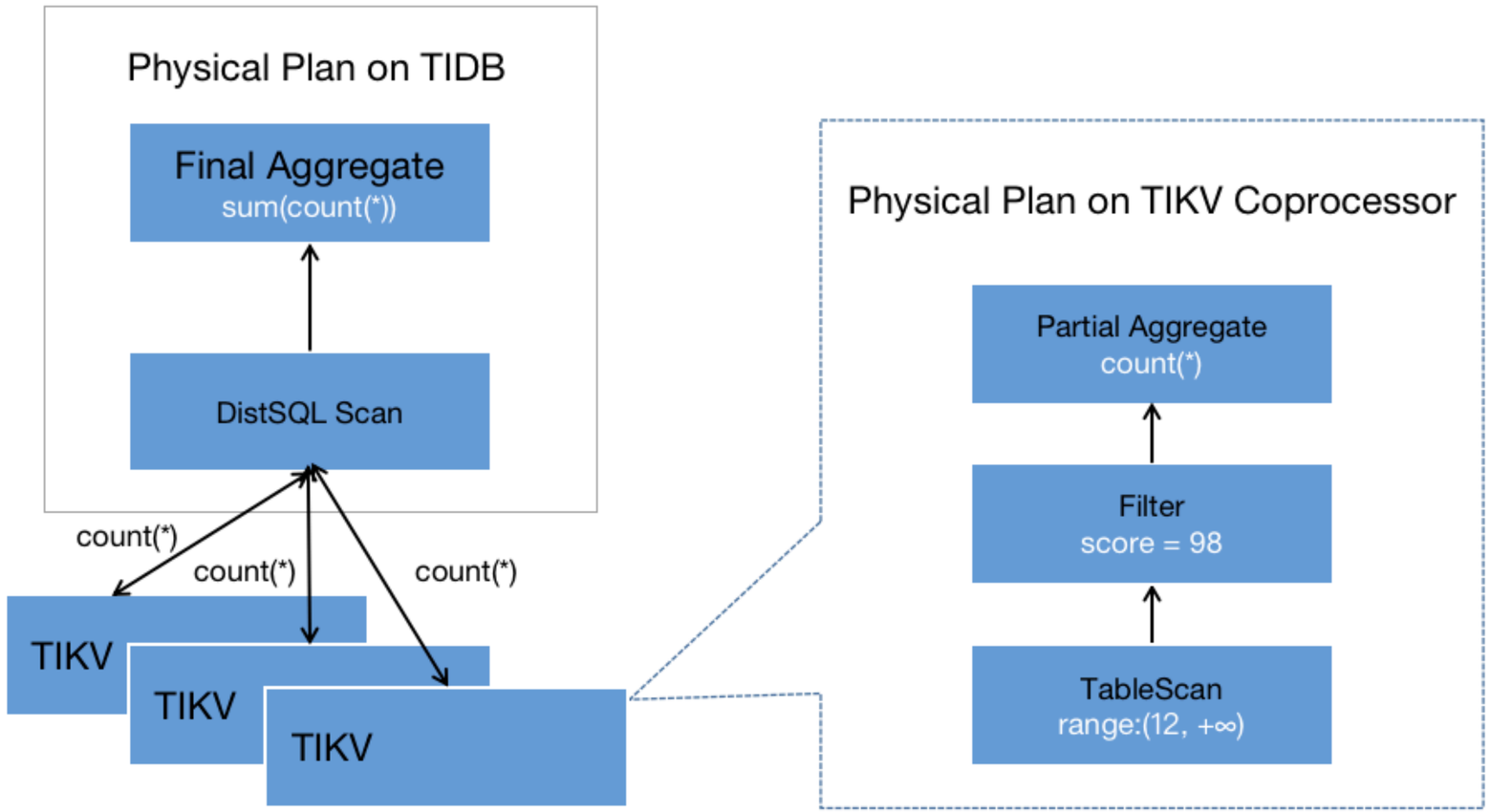
# 存在的问题




- 序列化
- 反序列化
- 网络传输

# 算子下推 – Coprocessor

**select** count(*) **from table where** id > 12 **where** score = 98;



# HTAP

HTAP（混合事务 / 分析处理，Hybrid Transactional/Analytical Processing）

# 矛盾

- 行存对于分析场景不友好
- 无法做到 Workload 隔离（跑分析查询 CPU 就 1000%）
- TiSpark 场景下会更糟糕

# 行存 VS 列存

**行存**

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

SELECT avg(age) from emp;

**列存**

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

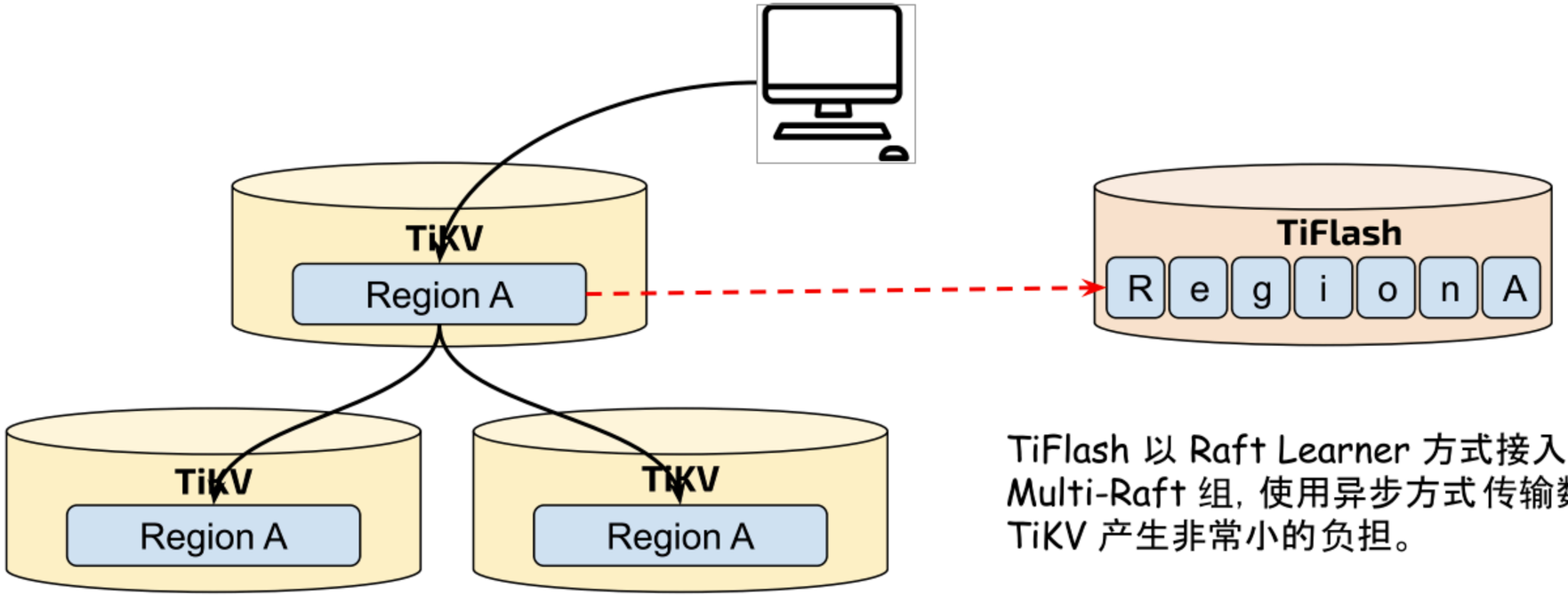
| name |
|---|
| Jane |
| John |
| Jim |
| Susan |

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

# 行存 VS 列存

| | 行式存储 | 列式存储 |
|---|---|---|
| **优点** | Ø 数据被保存在一起<br>Ø INSERT/UPDATE容易 | Ø 查询时只有涉及到的列会被读取<br>Ø 投影(projection)很高效<br>Ø 任何列都能作为索引 |
| **缺点** | Ø 选择(Selection)时即使只涉及某几列，所有数据也都会被读取 | Ø 选择完成时，被选择的列要重新组装<br>Ø INSERT/UPDATE比较麻烦 |

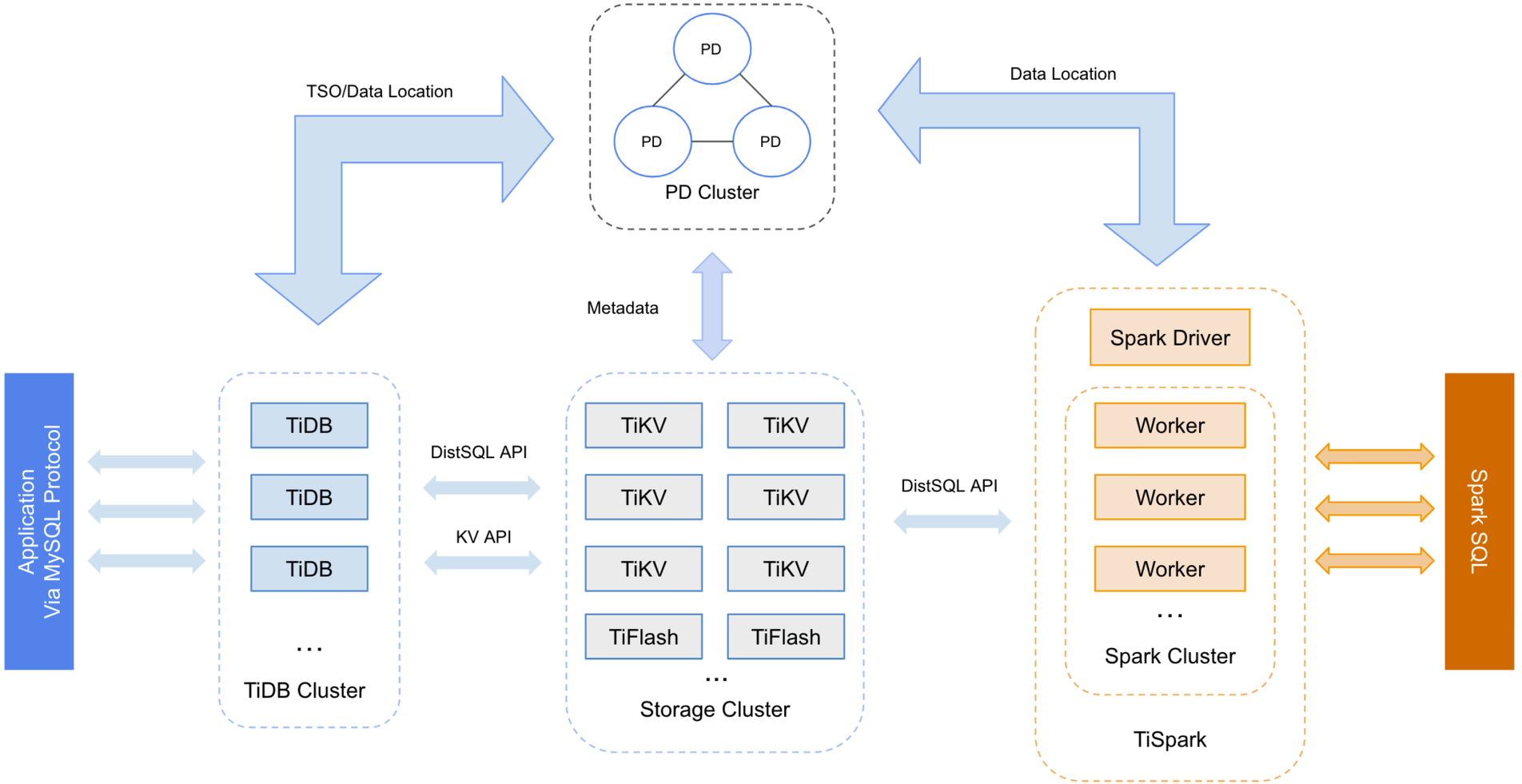
# Raft Learner – Sync




TiFlash 以 Raft Learner 方式接入 Multi-Raft 组，使用异步方式传输数据，对 TiKV 产生非常小的负担。

当数据同步到 TiFlash 时，会被从行格式拆解为列格式。

# HTAP


PD
PD
PD
PD Cluster
TSO/Data Location
Data Location
Metadata
Application
Via MySQL Protocol
TiDB
TiDB
TiDB
...
TiDB Cluster
DistSQL API
KV API
TiKV
TiKV
TiKV
TiKV
TiKV
TiKV
TiFlash
TiFlash
...
Storage Cluster
DistSQL API
Spark Driver
Worker
Worker
Worker
...
Spark Cluster
TiSpark
Spark SQL

# 总结

Make it work, make it right, make it fast

谢谢大家！