



# Pegasus : 一个分布式KV系统的设计过程

覃左言@小米





# 关于我

姓名：覃左言

经历：腾讯/百度/小米

关注：基础架构、分布式系统

爱好：运动、开源

开发过微服务框架

写过RPC框架：<https://github.com/baidu/sofa-pbrpc>

参与过分布式框架：<https://github.com/Microsoft/rDSN>

正在做KV存储系统：<https://github.com/XiaoMi/pegasus>



# 大纲

背景  
设计  
总结



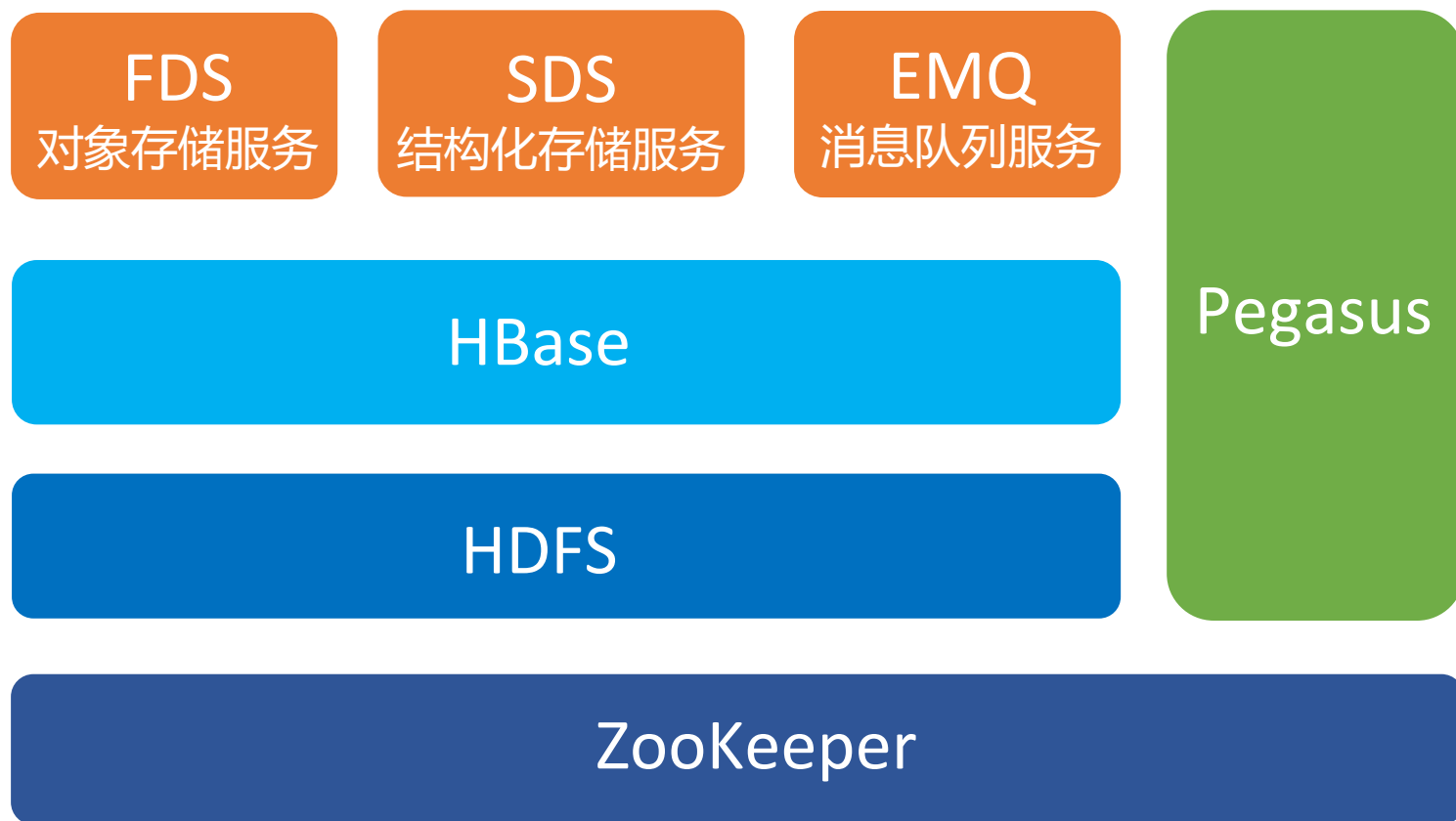
# 大纲

背景

设计

总结

# Storage Service in Xiaomi





# Storage Service in Xiaomi

10PB级

上百个业务

数百TB/day

>99.95%

数万亿行

5 HBase Committers

千万级QPS



# HBase Is Good, but Not Enough

Layered Structure

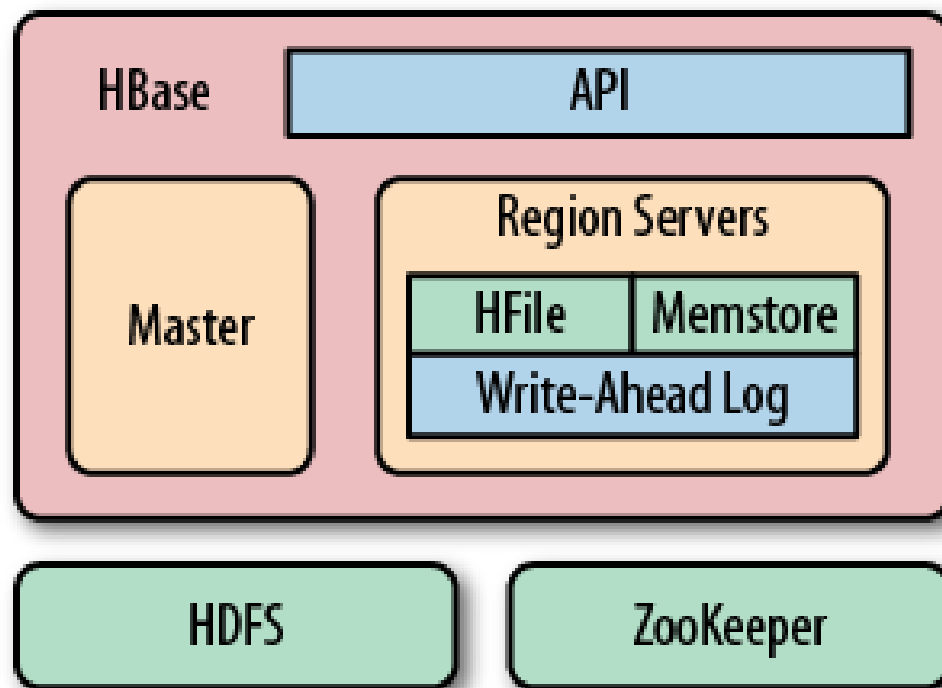
Weak Data Locality

Longtime Recovery

JVM Garbage Collection

可用性

性能



# What We Want

高可用

高性能

强一致

易使用

目标用户

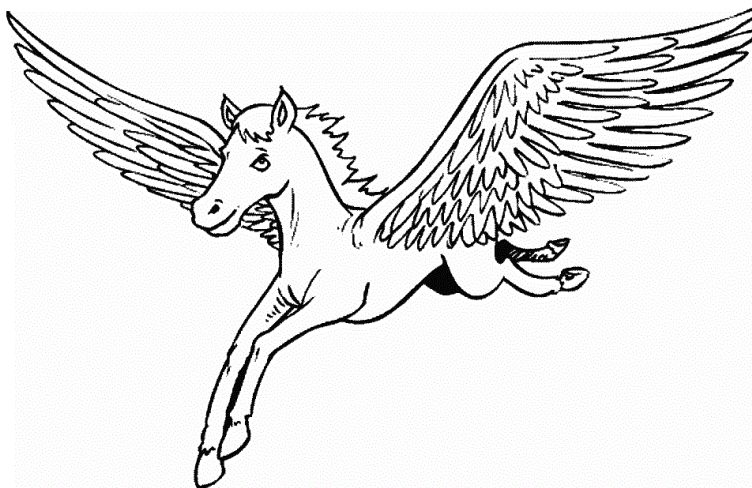
- 对延迟较敏感的在线业务：广告、支付
- 对可用性要求很高
- 希望提供强一致性的语义



# What We Did

## Pegasus

一个高可用、高性能、强一致的分布式KV存储系统





# 大纲

背景

设计

总结

# Many Choices ...



数据视图：KV系统 还是 表格系统？

数据分布：Hash 还是 Range？

系统架构：Centralized 还是 De-Centralized？

实现语言：C++、Java 还是 Go？

存储介质：HDD、SSD 还是 Memory？

一致性协议：Paxos、Raft 还是其他？

... ..

围绕需求

先做容易的选择

不要太纠结

留有切换的余地

# Basic Choices

实现语言：C++ ~~Java~~

- Java有GC问题
- C++性能高，风险小

性能

开发难度

存储介质：SSD ~~HDD~~ ~~Memory~~

- 性能、成本

风险

单机引擎：RocksDB ~~BDB~~ ~~LevelDB~~

- LSMT ( Log Structured Merge Tree ) 保证写性能
- 针对SSD和多核优化

# Model Choices

数据视图：KV系统 ~~Tabular系统~~

- 关注点在架构可行性
- KV系统更易实现
- 将来可改造为Tabular系统

数据分布：固定Hash分片 ~~一致性Hash~~ Range分片

- 实现简单
- 数据倾斜：合理设计Hash键和Hash函数
- 可伸缩性：预设 Partition Count 远大于 Server Count
- 热点问题：Hash分片和Range分片都不易解决

# 数据模型

- 组合主键：HashKey + SortKey

- HashKey用于Hash分片

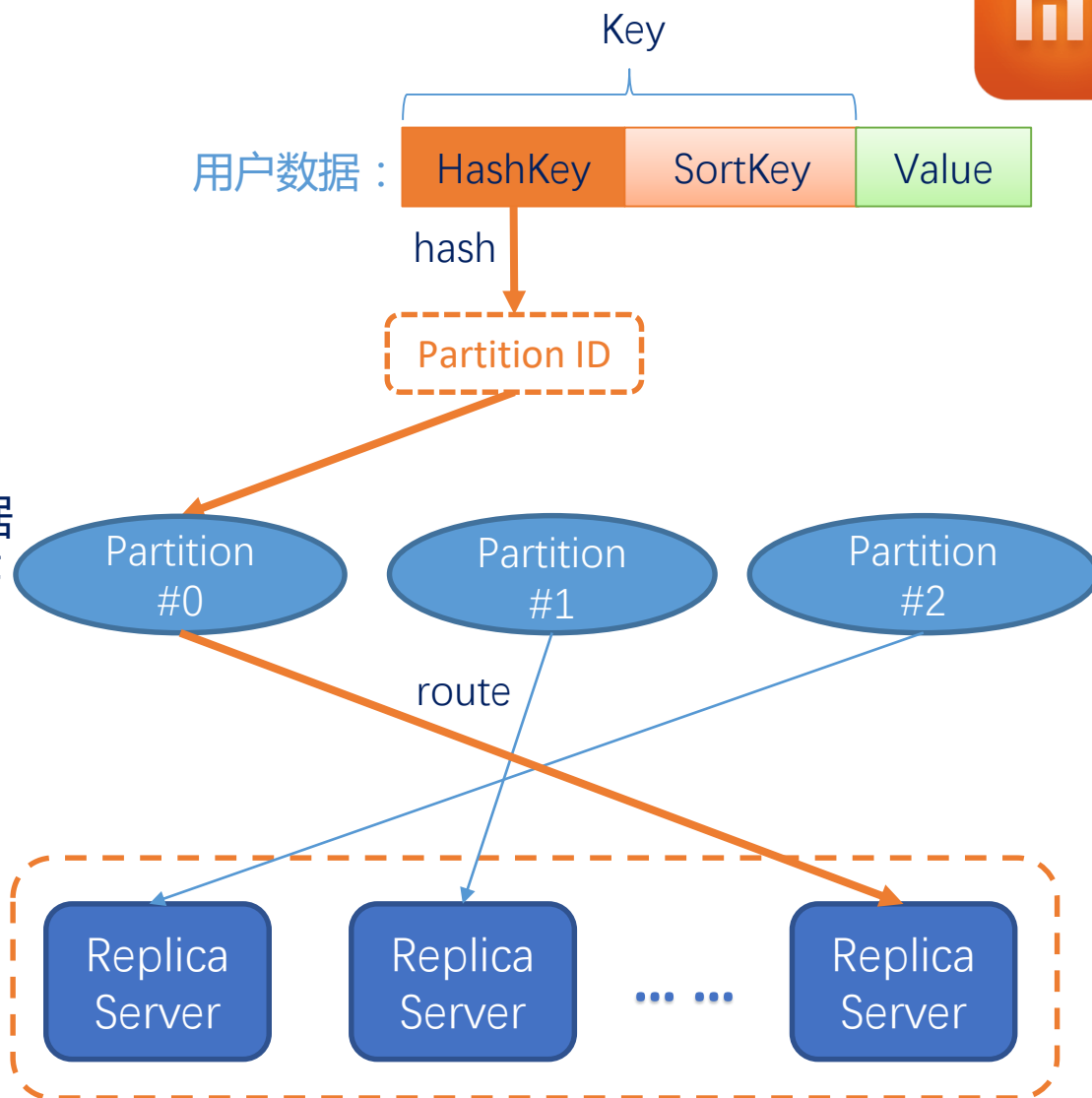
- 同一个分片(Partition)中的数据按照 [HashKey + SortKey] 排序

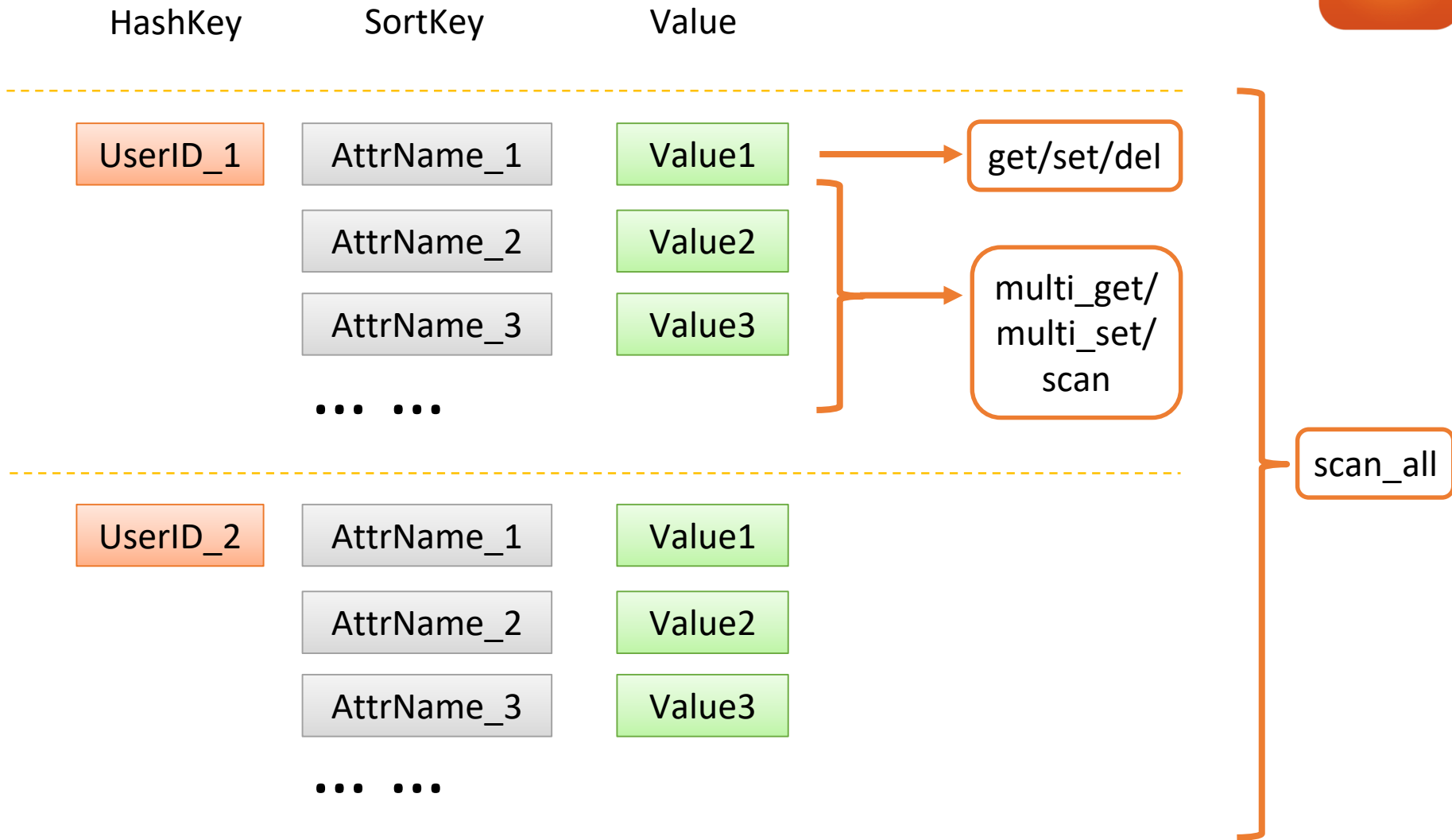
- 利用Table进行空间隔离

- 随着业务需求增加功能

简单

灵活







## 数据接口

- `get(HashKey, SortKey) → Value` 读单条数据
- `set(HashKey, SortKey, Value, TTL) → Bool` 写单条数据
- `del(HashKey, SortKey) → Bool` 删单条数据
- `multi_get(HashKey, SortKey[]) → Value[]` 读相同HashKey的多条数据
- `multi_set(HashKey, SortKey[], Value[]) → Bool` 写相同HashKey的多条数据
- `scan(HashKey, SortKeyBegin, SortKeyEnd) → Iterator` 扫描相同HashKey的数据
- `scan_all() → Iterator` 扫描全部数据

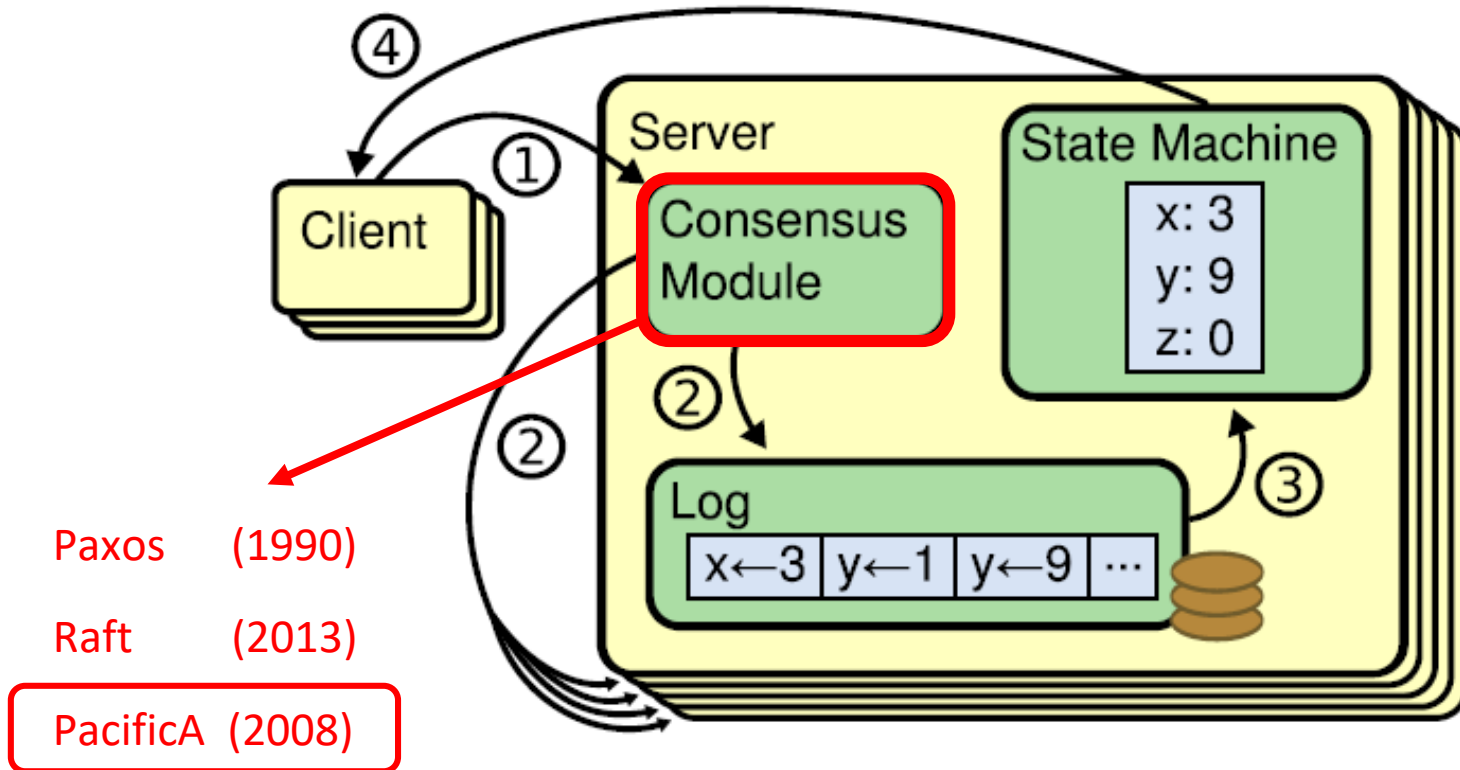
## Redis适配

SET SETEX GET DEL INCR INCRBY DECR DECRBY TTL

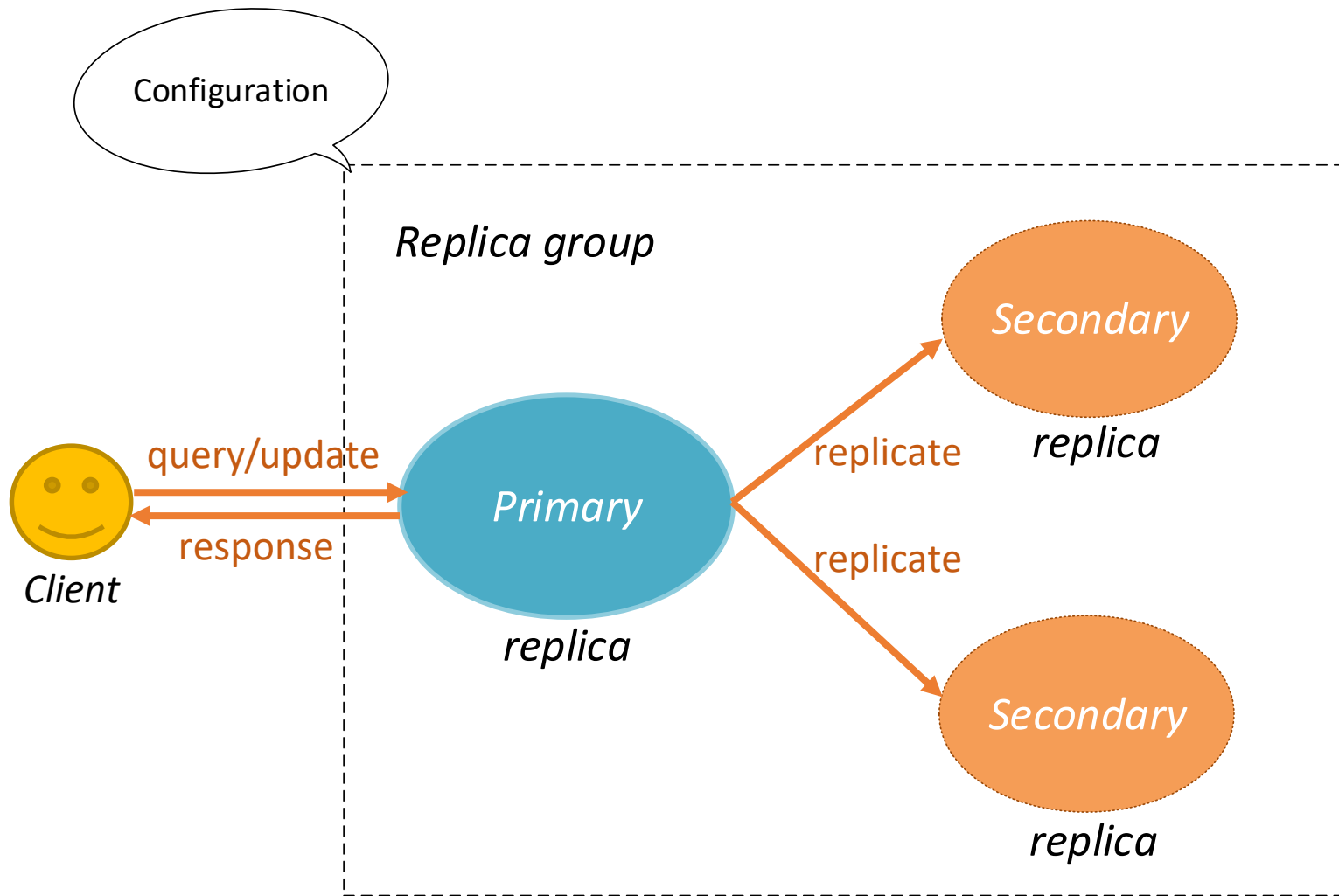
易使用



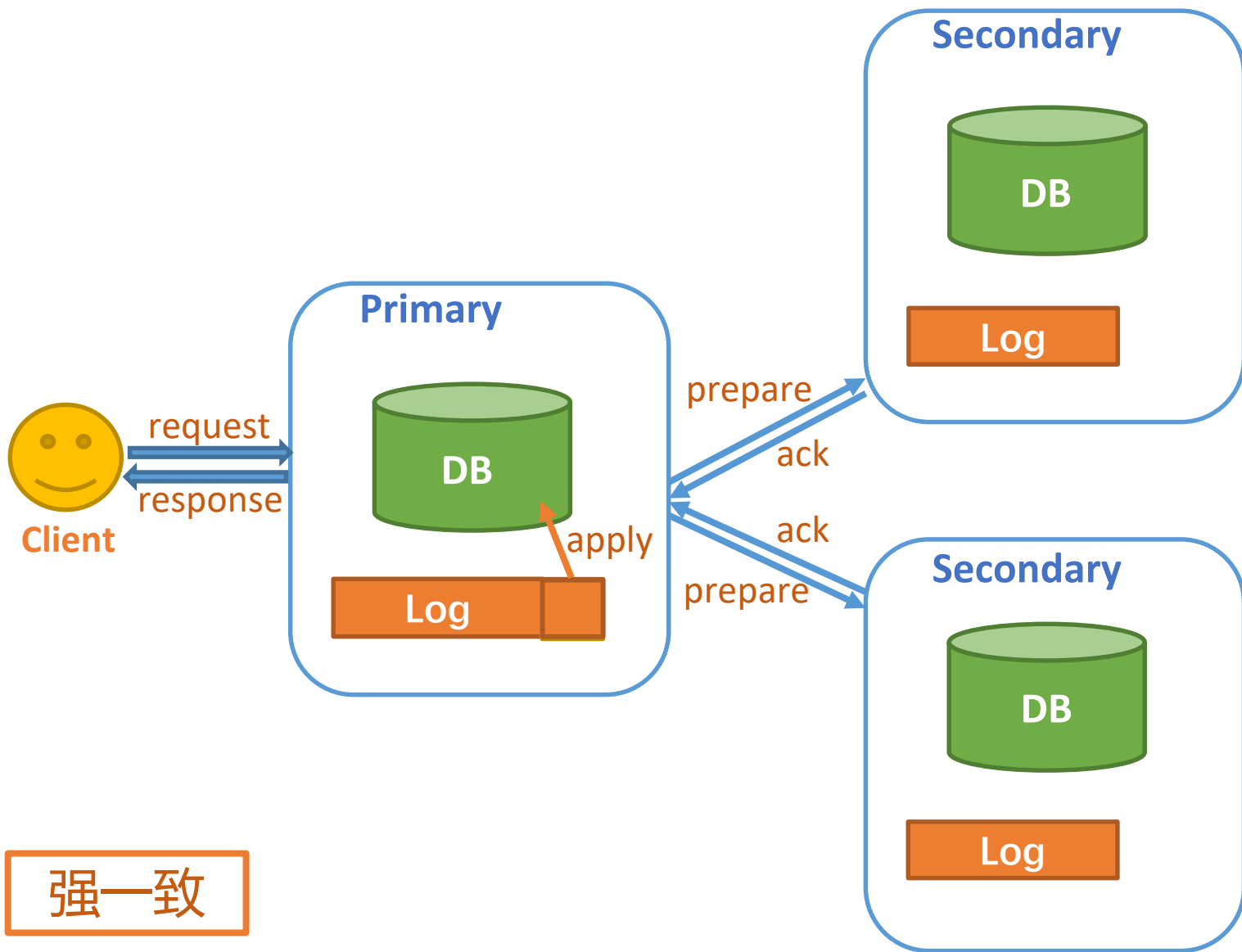
# Consensus Algorithm Choices



**Replicated State Machine Architecture**



## Primary/Backup Paradigm of PacificA

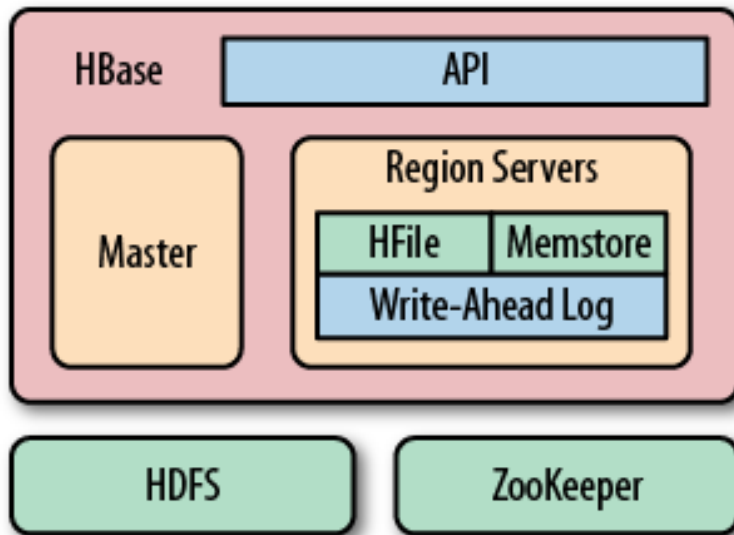


Two-Phrase Commit of PacificA

# Architecture Choices

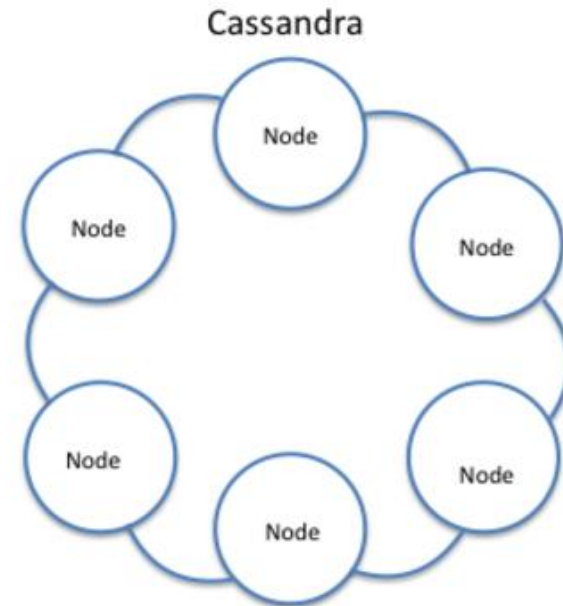
Centralized

Vs. Decentralized

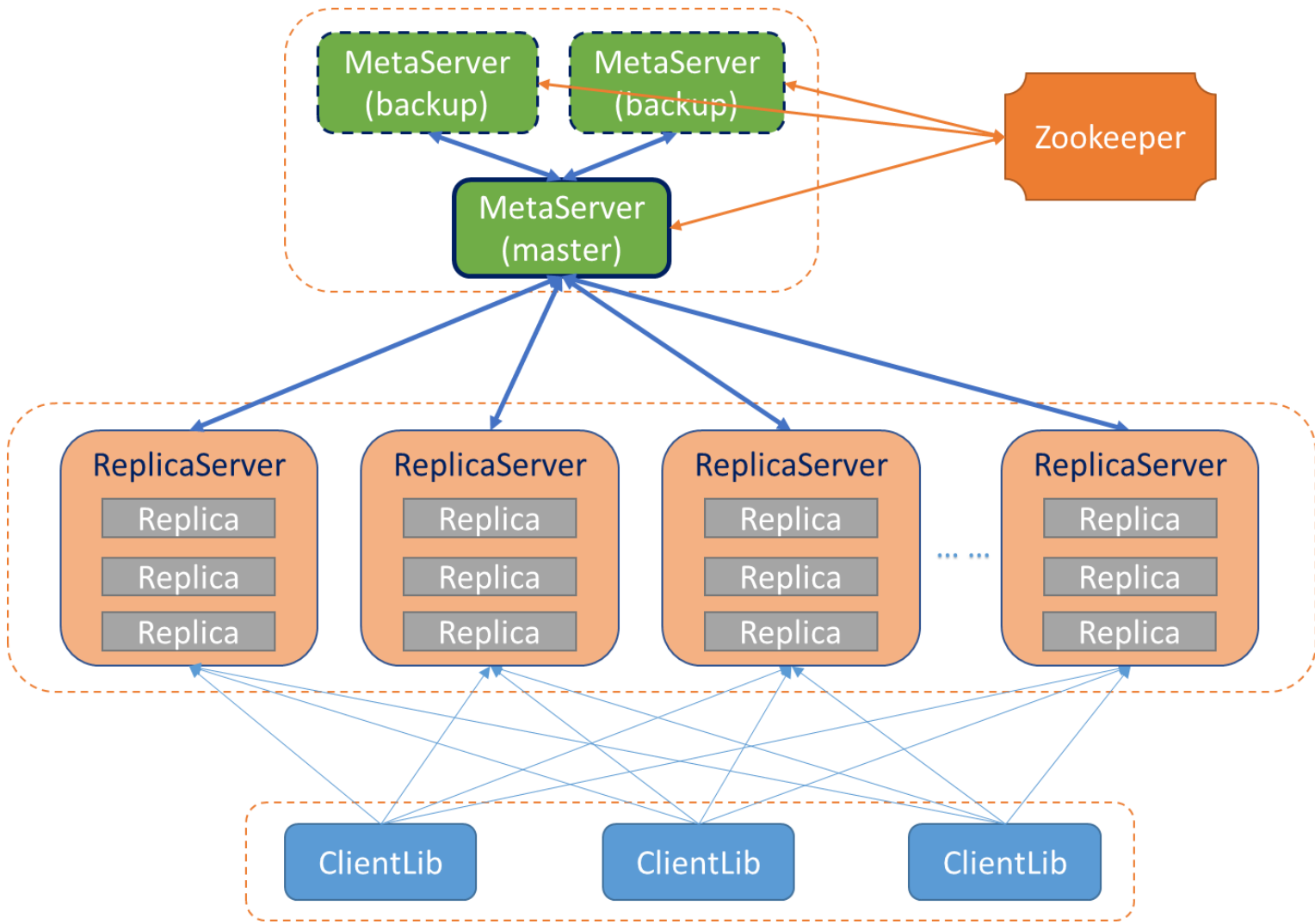


Layered

Vs.



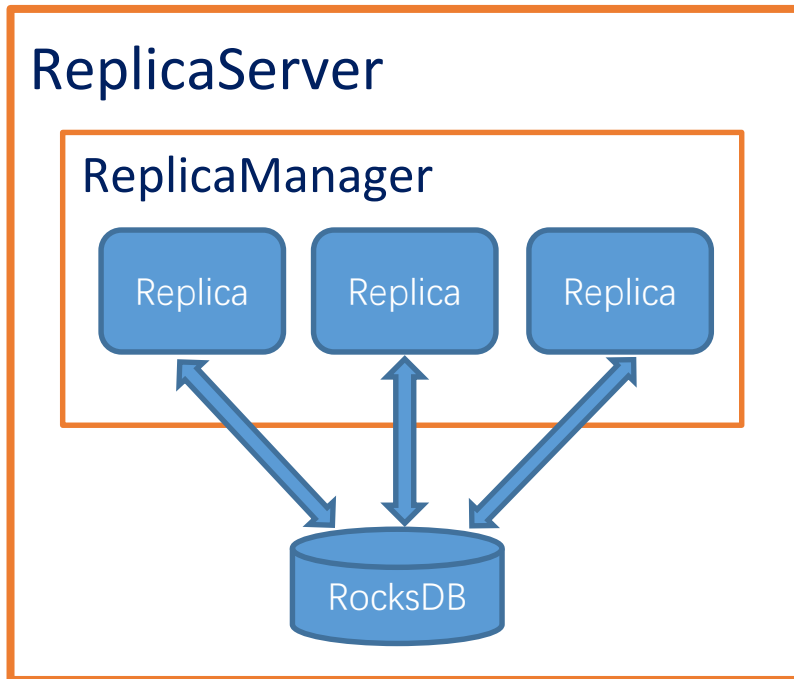
Integrated



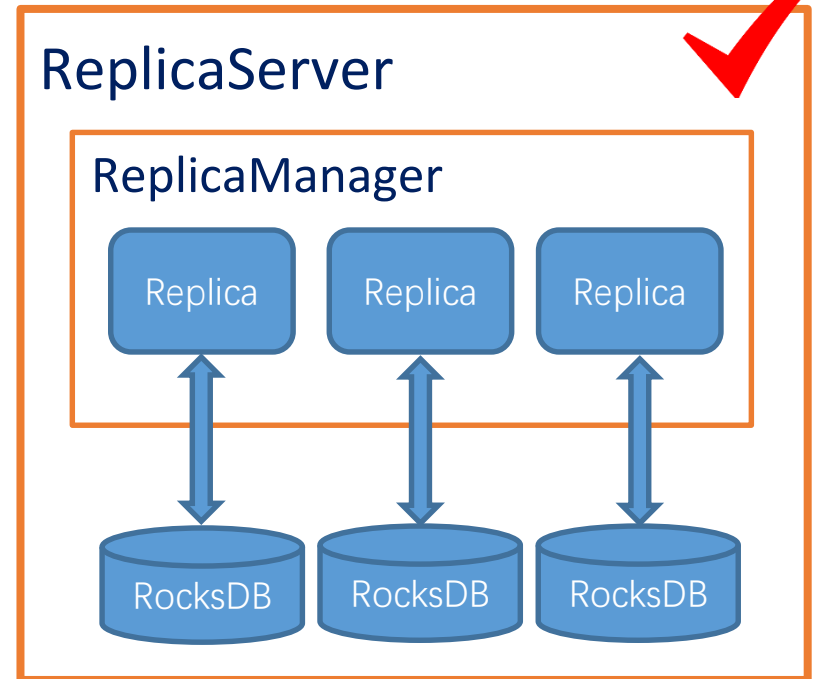
**Pegasus Architecture**



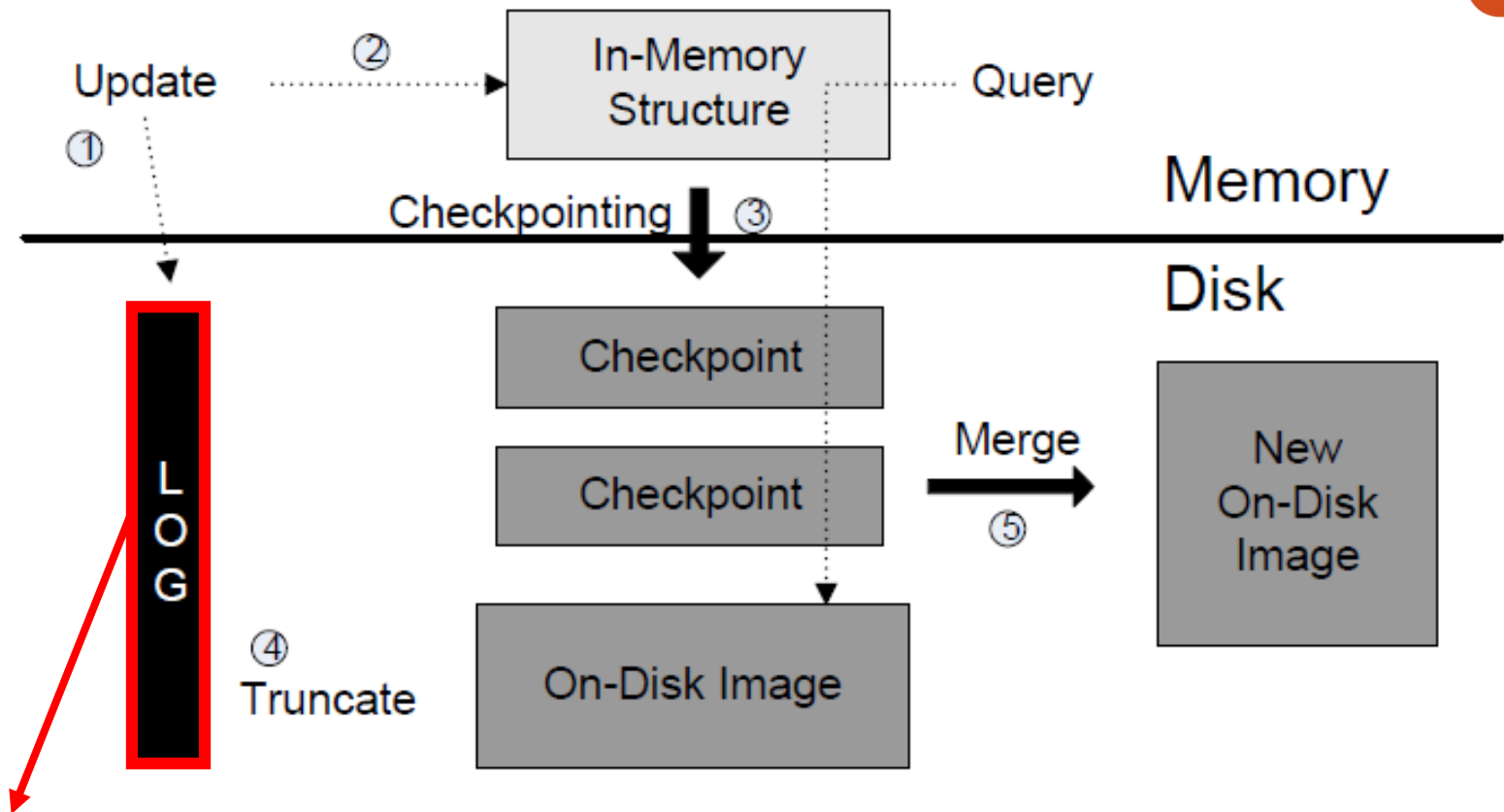
# ReplicaServer Design Choices



VS



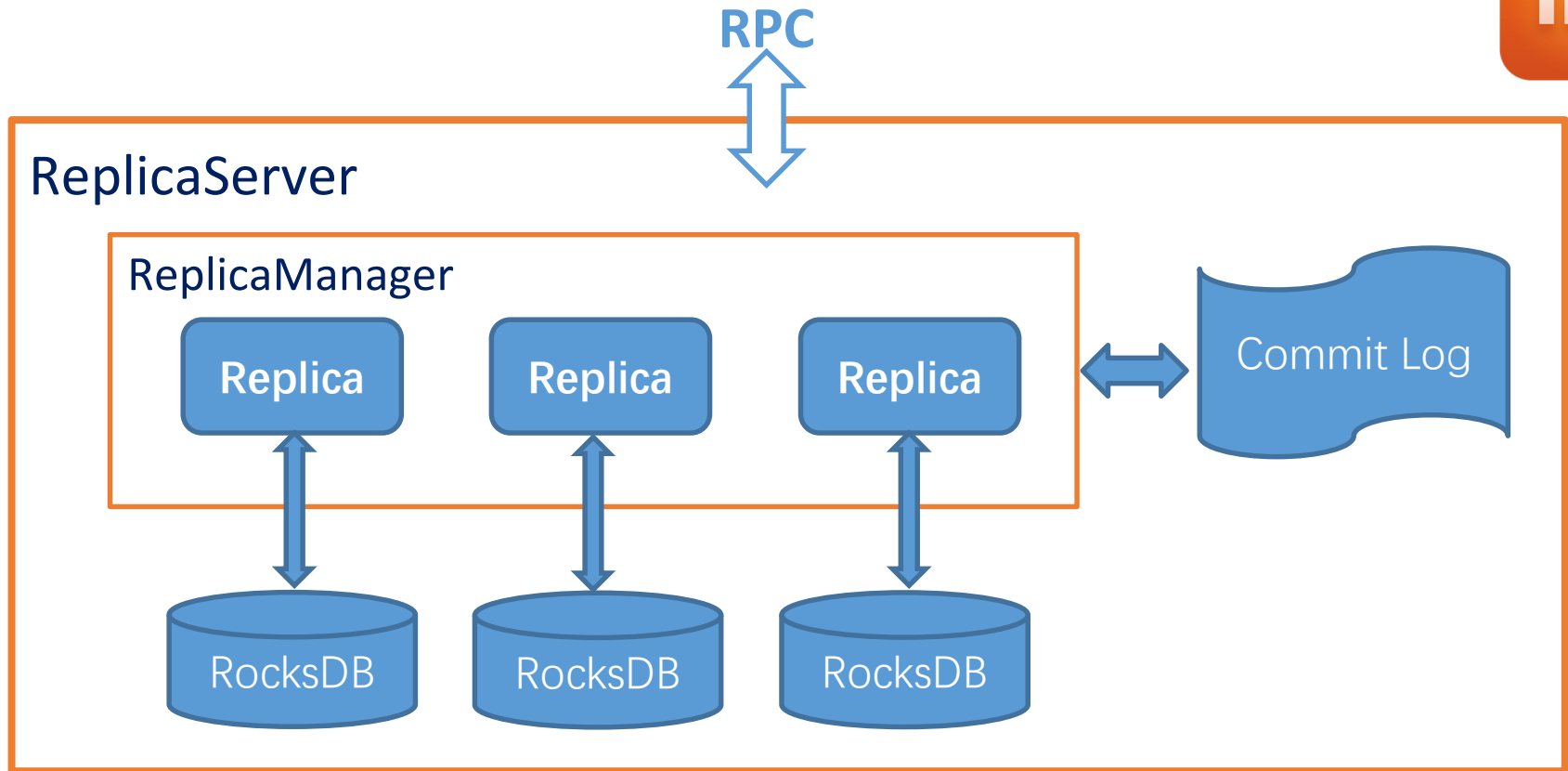
- 避免不同Table/Partition的数据相互影响
- 更容易实现 Load Balance 和 Drop Table
- 方便将数据分散到多个SSD盘，提高并发能力



Make commit log shared by all replicas

- 减少 write compete
- 适合 batch write

## RocksDB: Log-Based Storage System



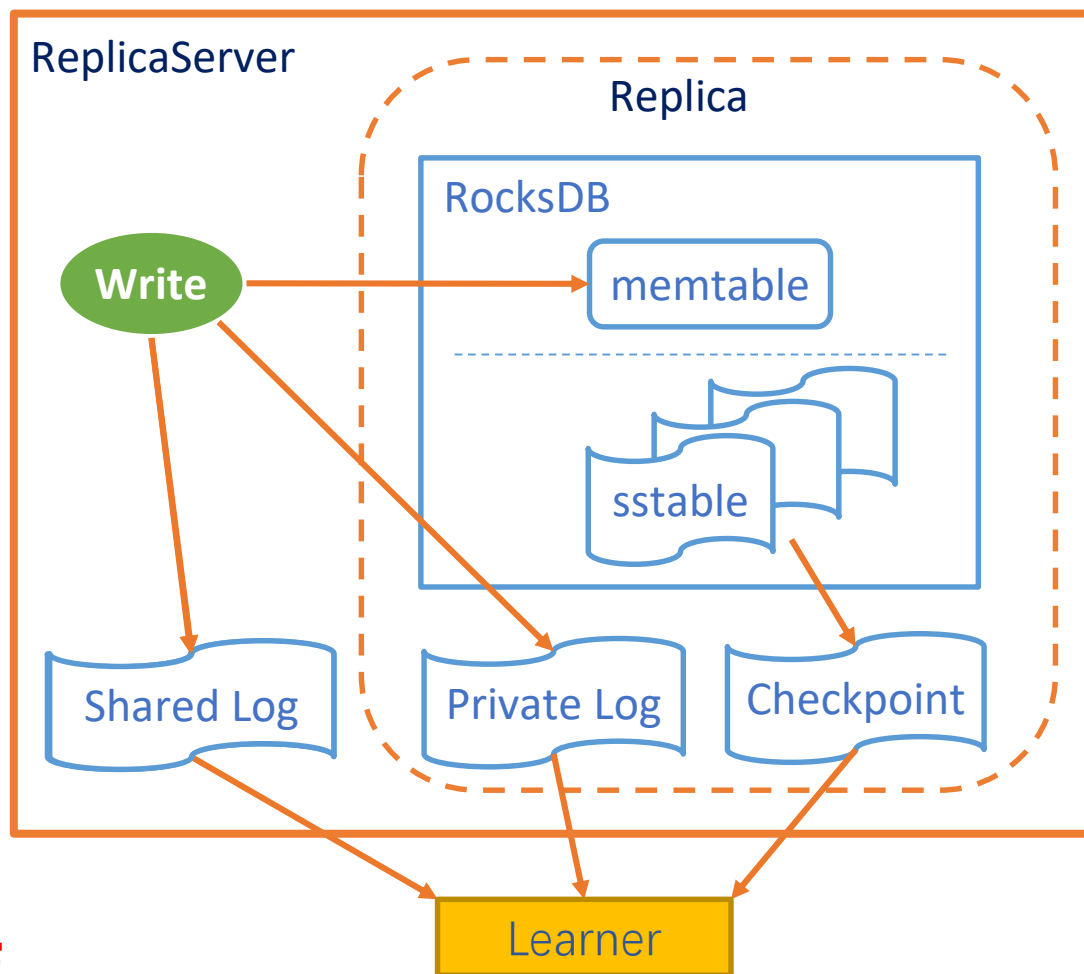
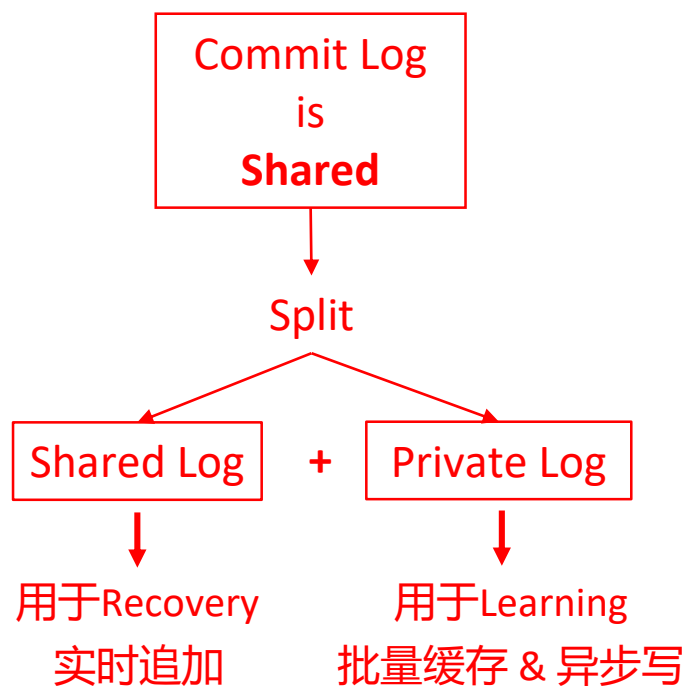
- ReplicaServer管理多个Replica
- 每个Replica 独享 RocksDB
- 所有Replica 共享 Commit Log

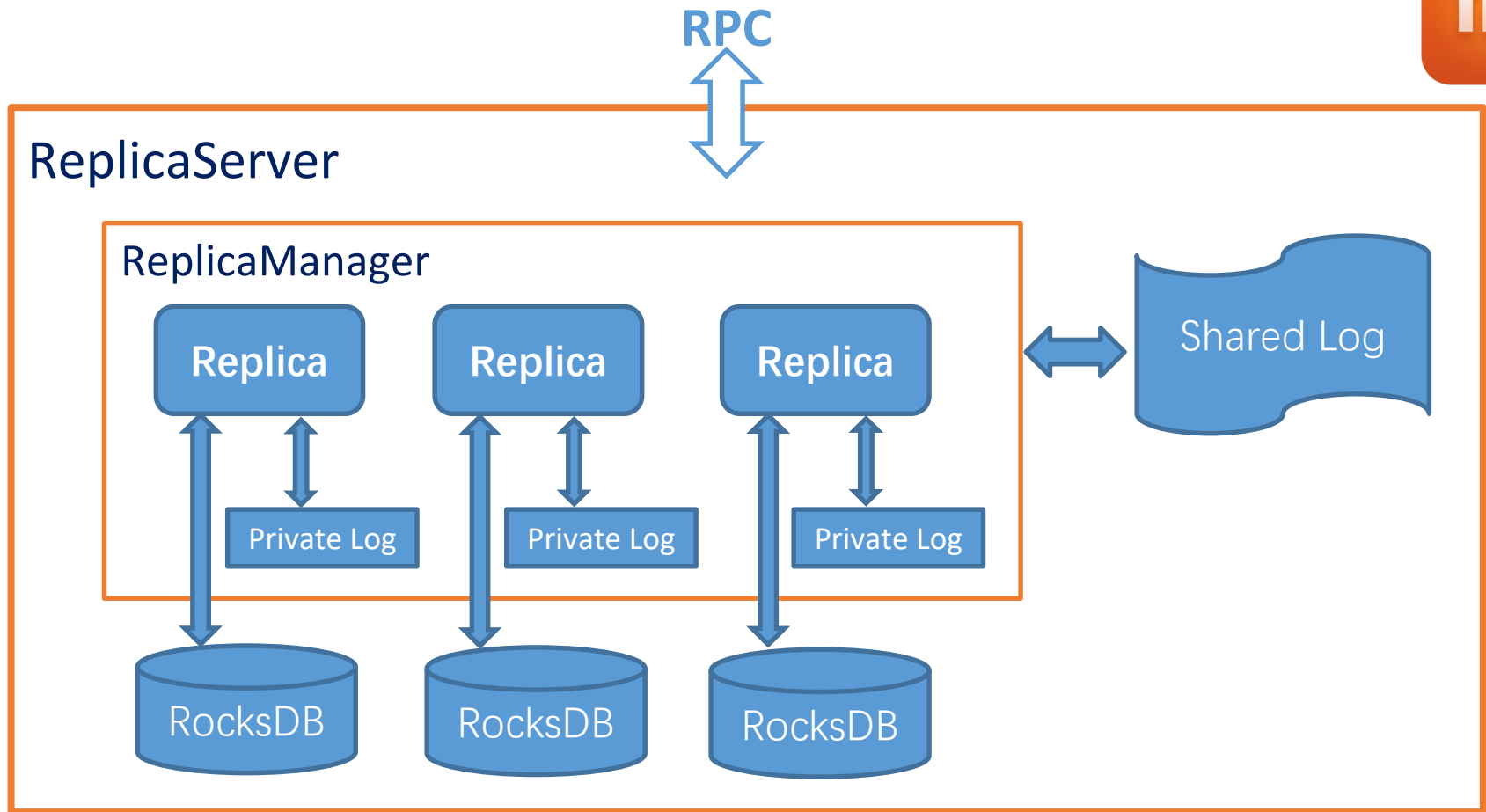
## Architecture of ReplicaServer



# Considering Catch-Up ...

- How to catch up:
  - Learn checkpoint
  - Learn commit log



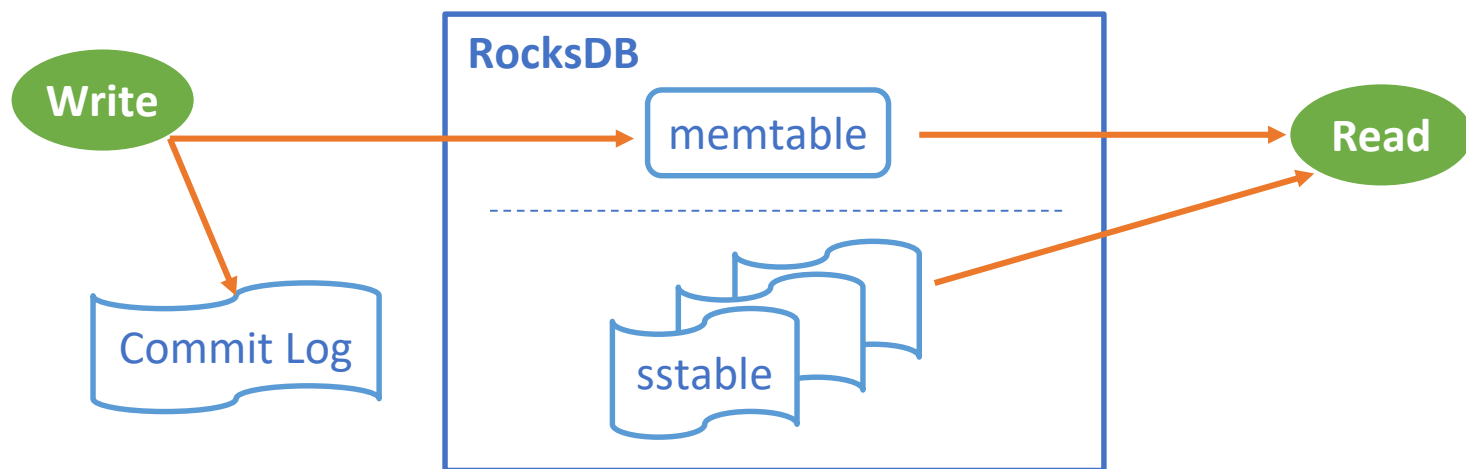


- Shared Log for Recovery
- Private Log for Learning

## Improved Architecture of ReplicaServer

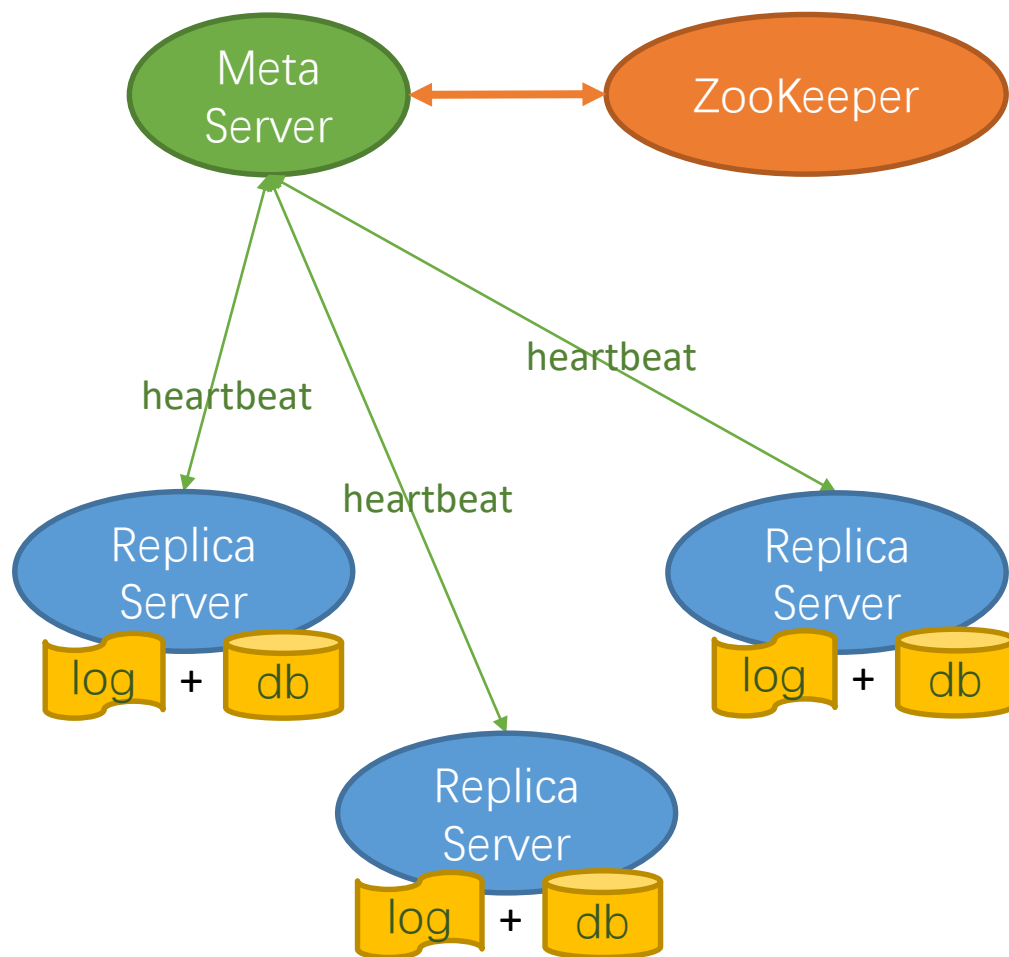
# RocksDB as Storage Engine

- 只使用Default Column Family
- 禁用Write Ahead Log
- 并发控制：写操作在单线程中串行执行，读操作允许多线程并发执行
- 数据扩展：写操作会额外传入数据在Log中的序号（Decree）
- 优化快照：同步 → 异步，避免同步过程阻塞写操作



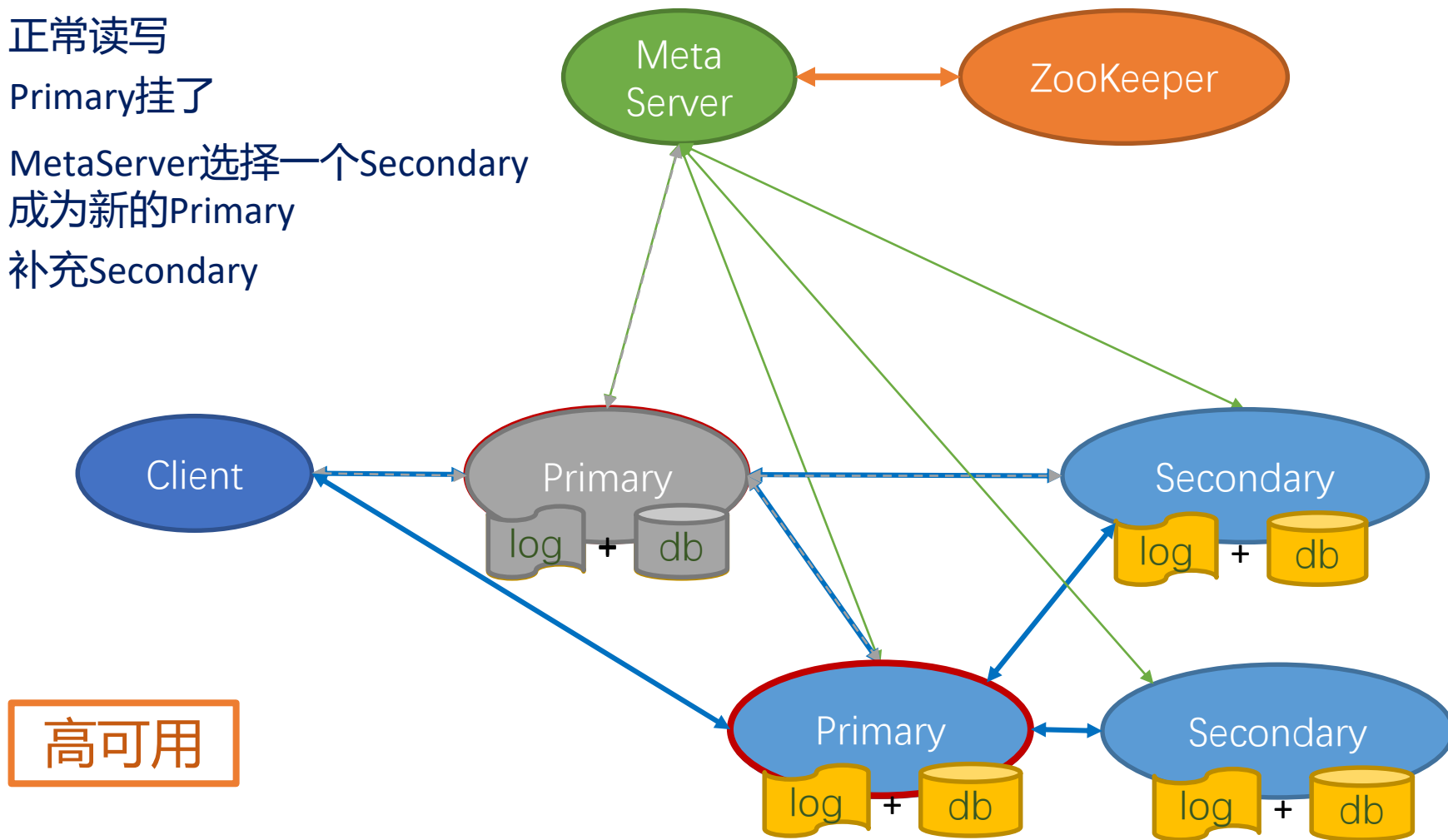
# Failover

- MetaServer和所有的ReplicaServer维持心跳
- Failure Detection通过心跳来实现
- Failover有三种类型：
  - Primary Failover
  - Secondary Failover
  - MetaServer Failover



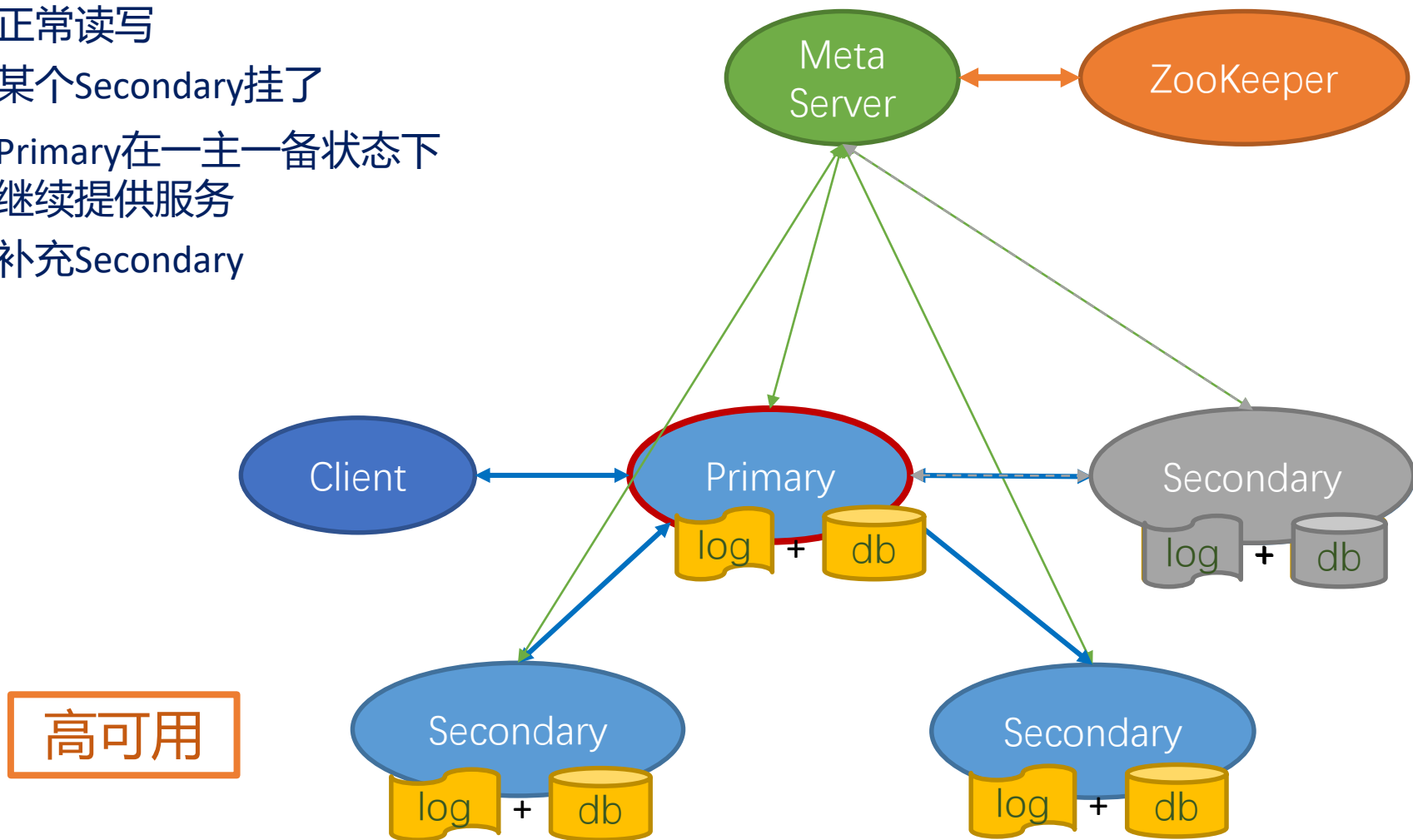
# Primary Failover

1. 正常读写
2. Primary挂了
3. MetaServer选择一个Secondary成为新的Primary
4. 补充Secondary



# Secondary Failover

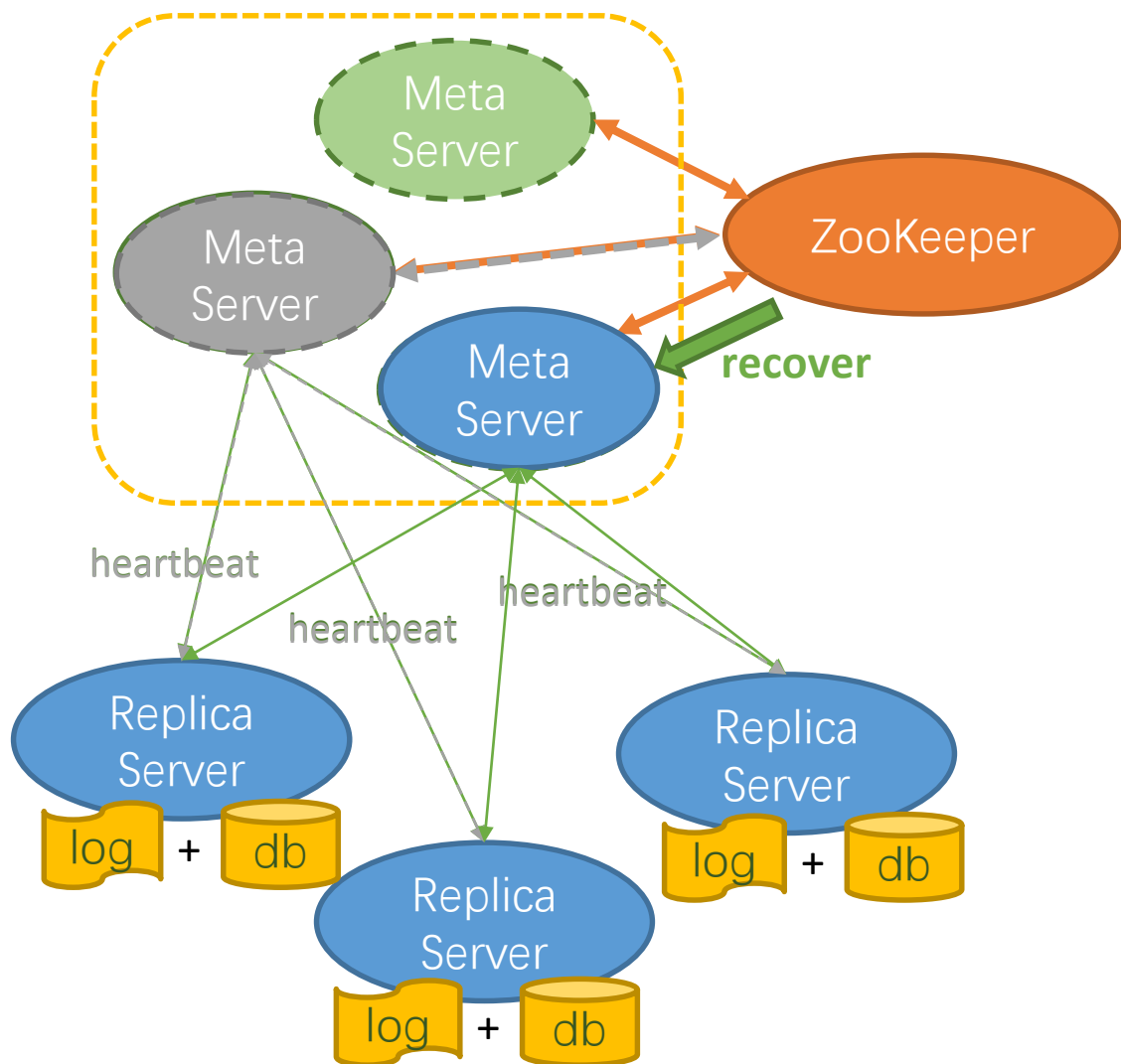
1. 正常读写
2. 某个Secondary挂了
3. Primary在一主一备状态下继续提供服务
4. 补充Secondary



# MetaServer Failover

1. 主MetaServer和所有的ReplicaServer维持心跳
2. 主MetaServer挂了
3. 某个备MetaServer通过ZooKeeper抢主成为新的主MetaServer
4. 从ZooKeeper恢复状态
5. 重新和所有ReplicaServer建立心跳

高可用





# Data Safety

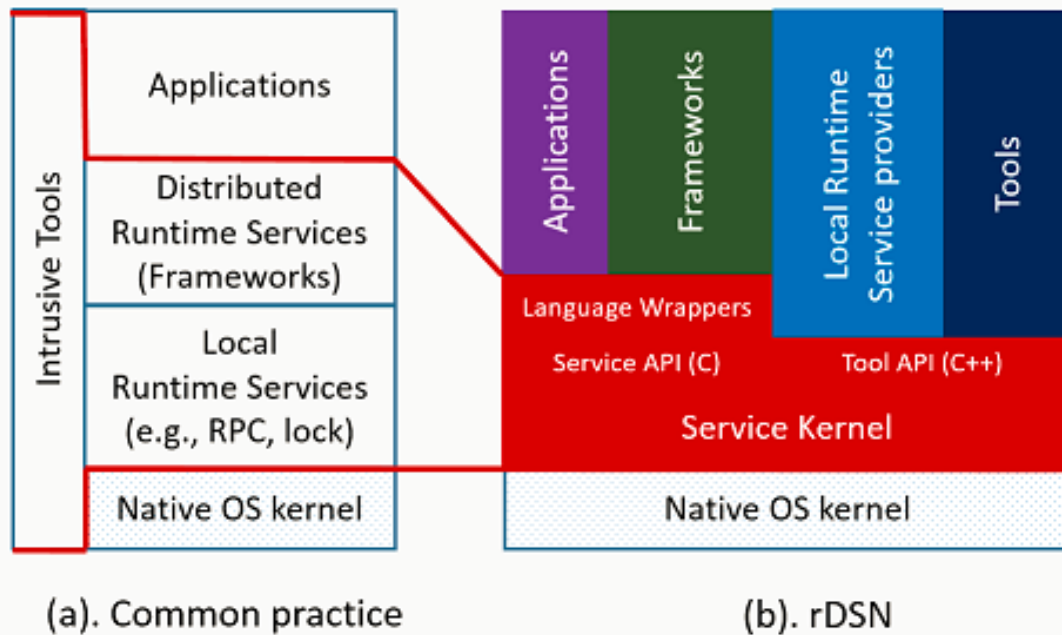
- Table软删除
  - Table删除后，数据会保留一段时间，防止误删除
- 元数据恢复
  - 元数据丢失或者损坏时，从各ReplicaServer收集并重建元数据
- 定期冷备份
  - 数据定期备份到多种介质，譬如HDFS
  - 在需要的时候快速恢复
- 跨机房热备
  - 在多个机房部署集群
  - 采用异步复制的方式实时同步数据
  - 多集群可同时对外提供服务





# Engineering ...

- Built on rDSN (**Robust Distributed System Nucleus**) framework
  - <https://github.com/Microsoft/rDSN>



Microkernel Architecture

Most Things Are Pluggable

Lock

RPC

AIO

Timer


Logging

Replication

... ..

Friendly for Testing,  
Debugging and Monitoring

# How to Test

- Unit Test
- Integration Test
- Simulation Test
  - Simulate cluster in
  - Processing can be r
- Scenario Test 
  - Declarative language
  - Construct different scenarios (executing paths / corner cases)

```

1 # Scenaria 201: inject on_aio_call of primary log write
2
3 set:load_balance_for_test=1 → 设置属性
4
5 # wait for server ready
6 config:{3,r1,[r2,r3]}
7 state:{{r1,pri,3,0},{r2,sec,3,0},{r3,sec,3,0}} → 等待状态
8
9 # begin to write k1
10 client:begin_write:id=1,key=k1,value=v1,timeout=0 → 发起动作
11
12 # inject aio error on primary r1
13 inject:on_aio_call:node=r1,task_code=WRITE_LOG → 注入错误
14
15 # error should occur on r1
16 state:{{r1,err,3,0},{r2,sec,3,0},{r3,sec,3,0}} → 等待状态
17
18 # r1 should drop itself
19 state:{{r2,sec,3,0},{r3,sec,3,0}}
20 config:{4,-,[r2,r3]}
21
22 # r2 will become the primary
23 state:{{r2,ina,4,0},{r3,sec,3,0}}
24 config:{5,r2,[r3]}
25

```



# 大纲

背景  
设计  
**总结**



Why → How → What

Many choices ...

Fit is the best

Pegasus是对HBase的有力补充

- 更优异的性能：C++、No GC、Data Locality
- 更高的可用性：No GC、Faster Failover
- 更轻量的部署：No HDFS、No ZooKeeper (in the future)
- 更简单的接口：Key-Value



## Future Work

- PacificA → Raft
- Key-Value → Tabular
- Remove ZooKeeper Dependency
- Performance Tuning
- Improve Load Balance
- Cross Row Transaction
- Open Source



# Thanks

个人微信号：



欢迎有志之士加入我们：分布式相关、人工智能相关  
简历请发至邮箱：[qinzuoyan@xiaomi.com](mailto:qinzuoyan@xiaomi.com)

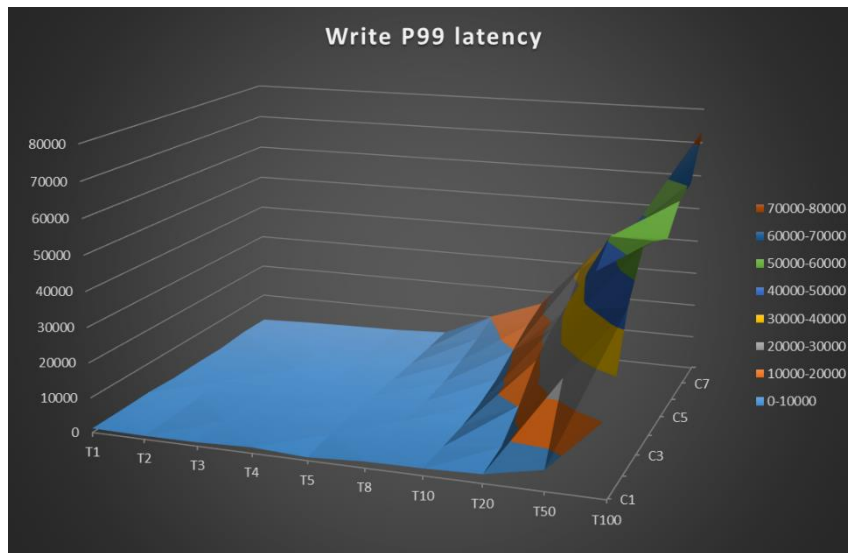
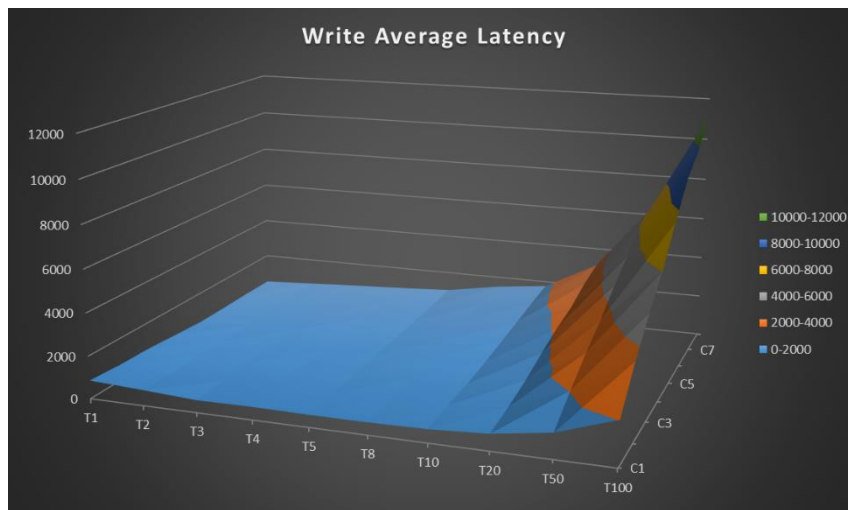


# 附录



## Pegasus写性能

- 单机QPS 可达 1.5w+
- 单机QPS<1w 时，平均延迟<5ms
- 单机QPS<1w 时，P99 延迟<20ms



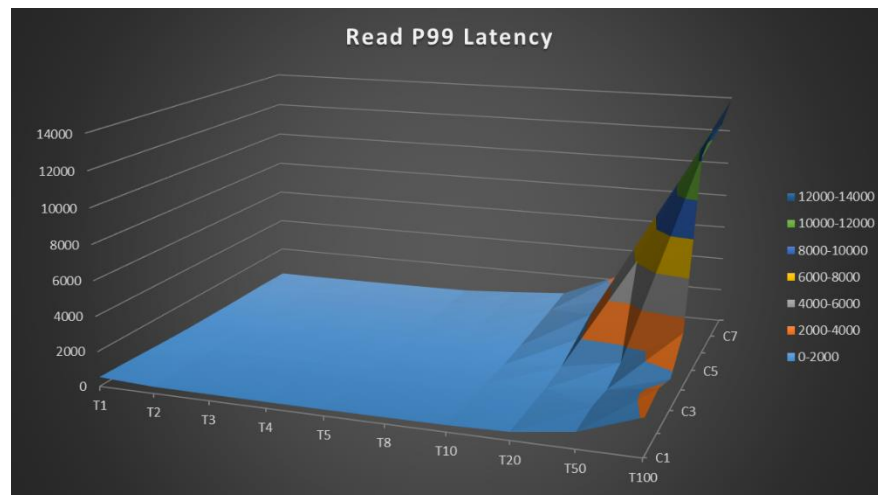
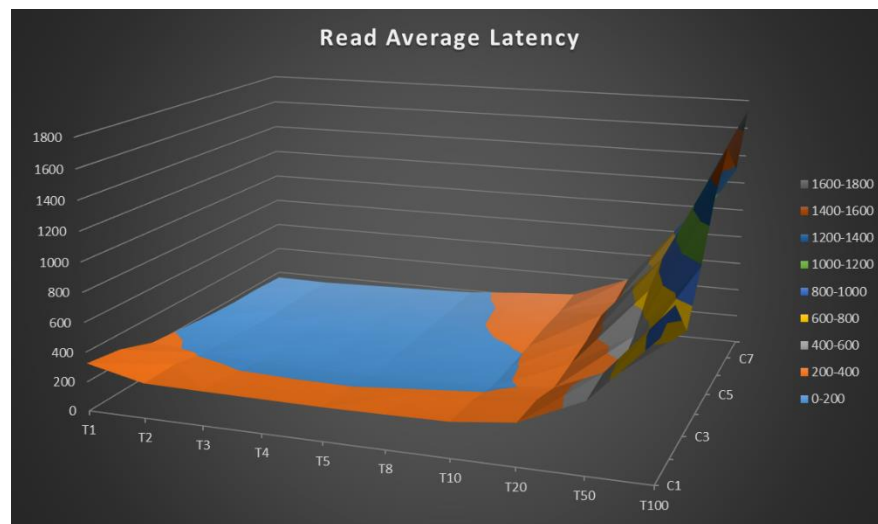
注：X轴 $T_n$ 为单Client线程数，Y轴 $C_n$ 为Client进程数，Z轴延迟单位为微妙，集群使用5个数据节点





## Pegasus读性能

- 单机QPS 可达 10w+
- 单机QPS<5w 时，平均延迟<1ms
- 单机QPS<5w 时，P99 延迟<5ms

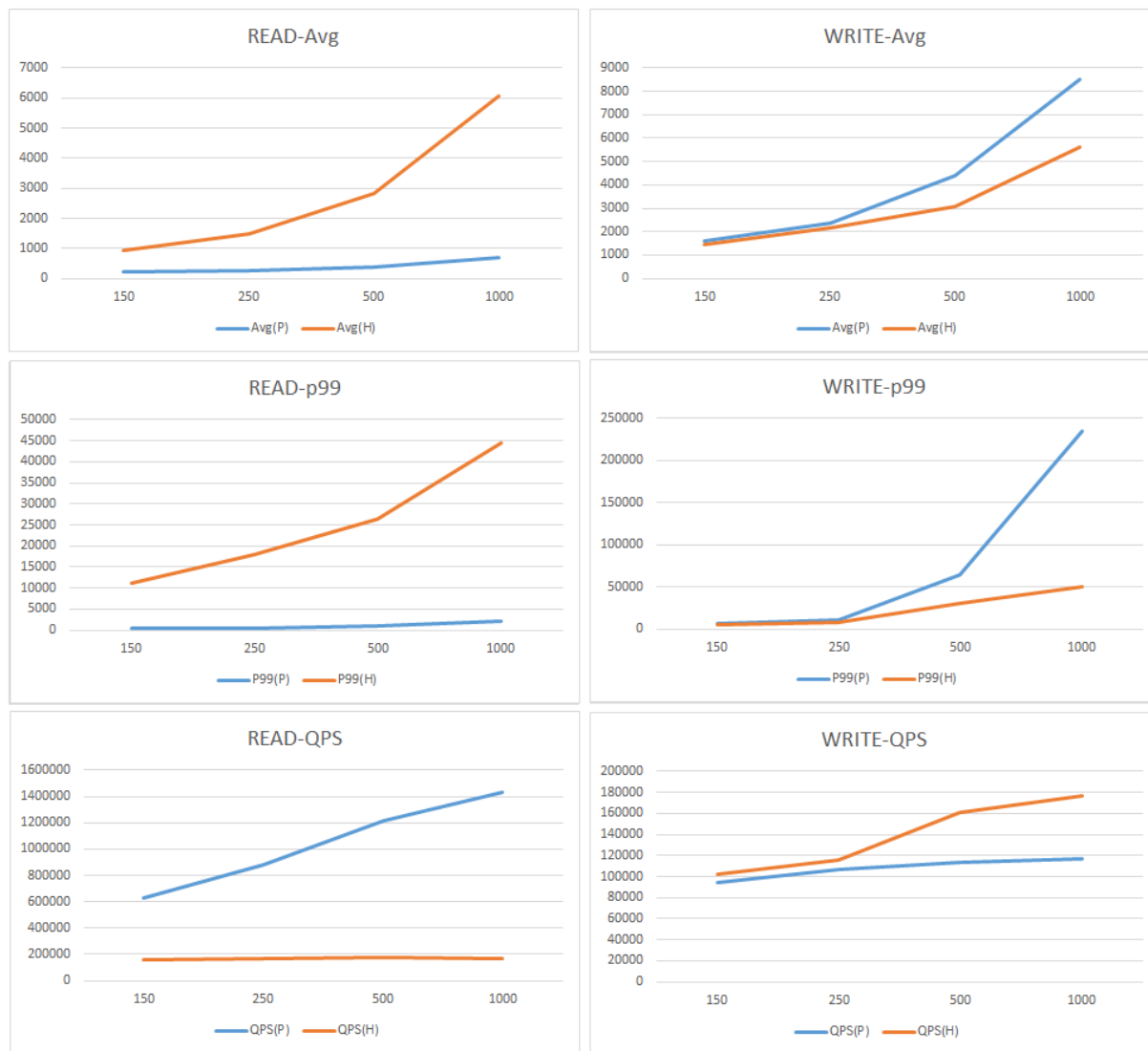


注：X轴 $T_n$ 为单Client线程数，Y轴 $C_n$ 为Client进程数，Z轴延迟单位为微妙，集群使用5个数据节点

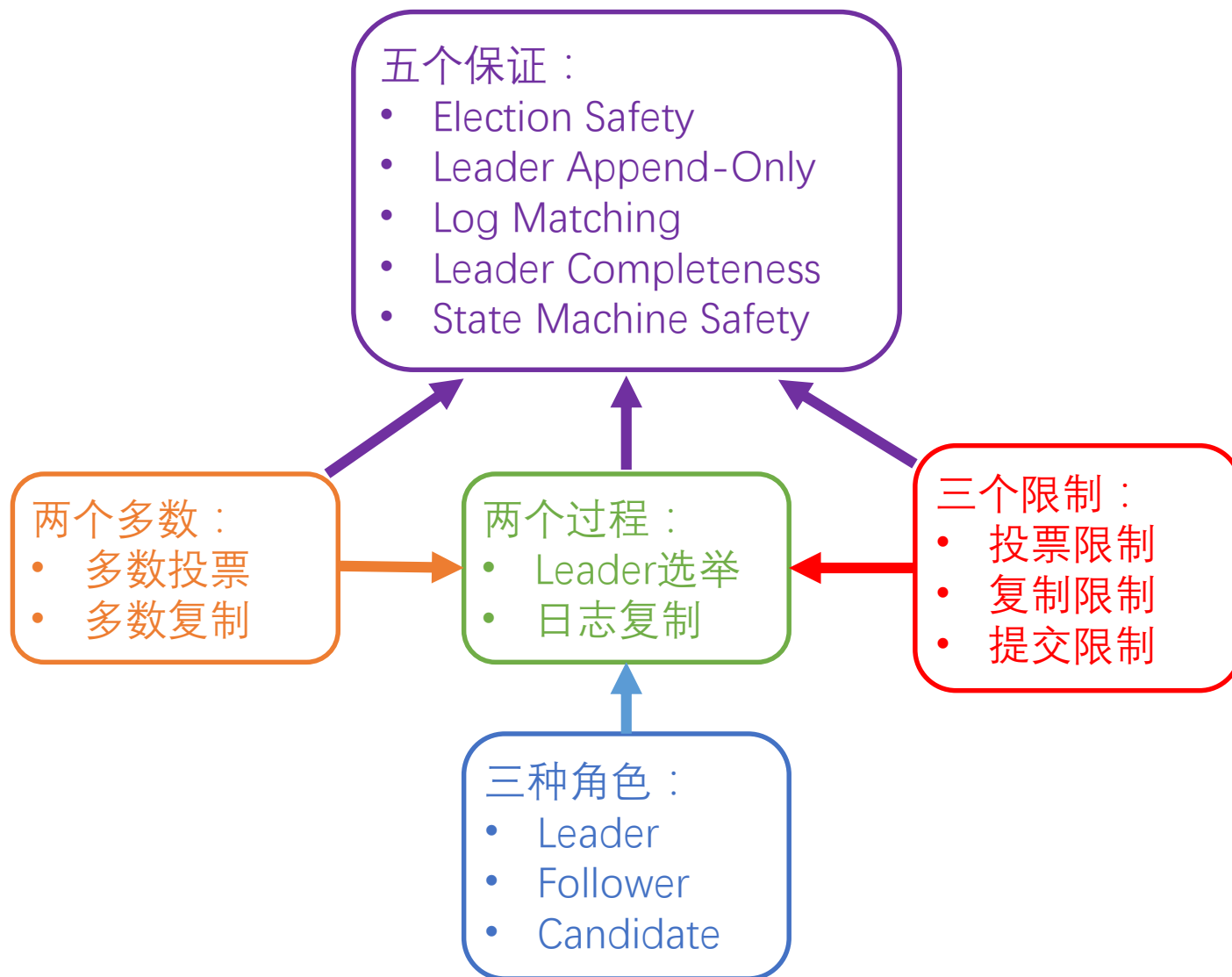


## 对比HBase

- 读性能优势明显
- 写性能略差



注：蓝色曲线为Pegasus，红色曲线为HBase，延迟单位为微妙，集群使用10个数据节点



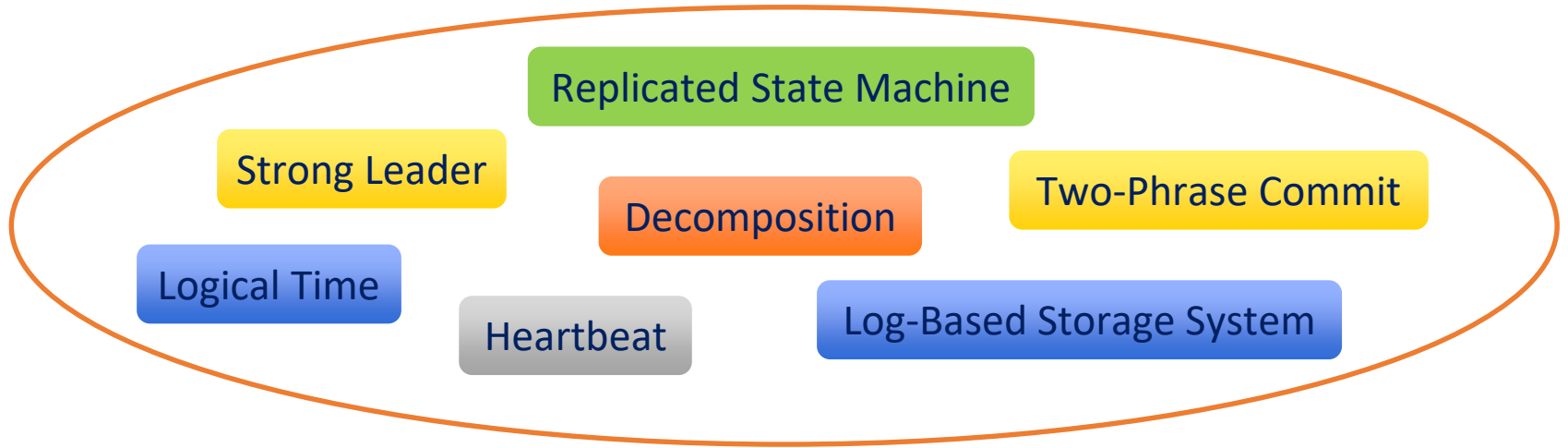
## Raft Algorithm



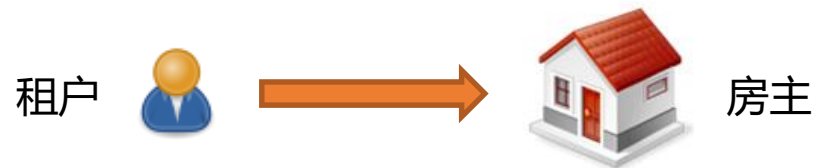
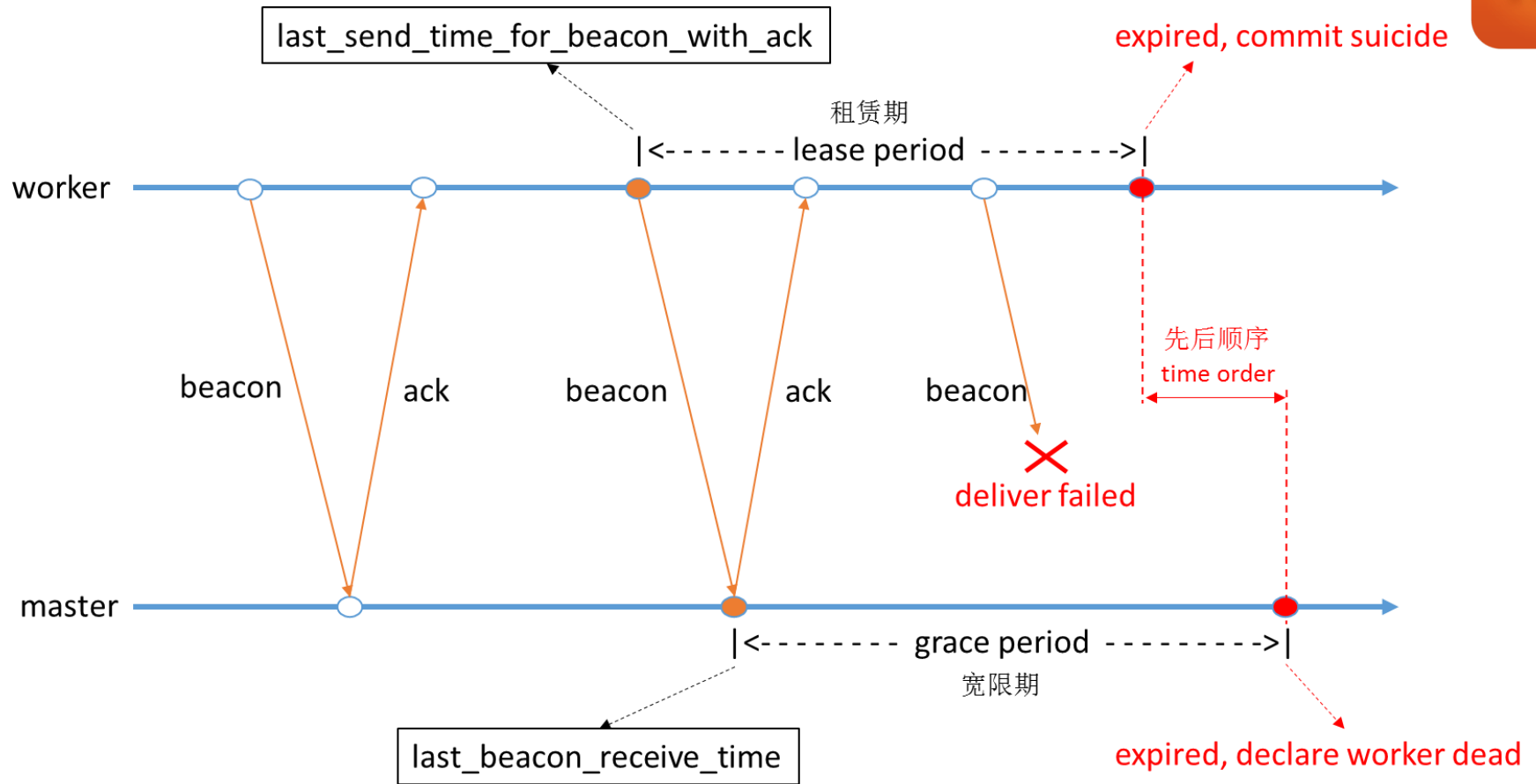
# Raft

# VS

# PacificA



Leader Election	Majority Vote	First Come First Served
Commit Condition	Majority Replication	Fully Replication
Safety Guarantee	Additional Restriction	Fully Replication

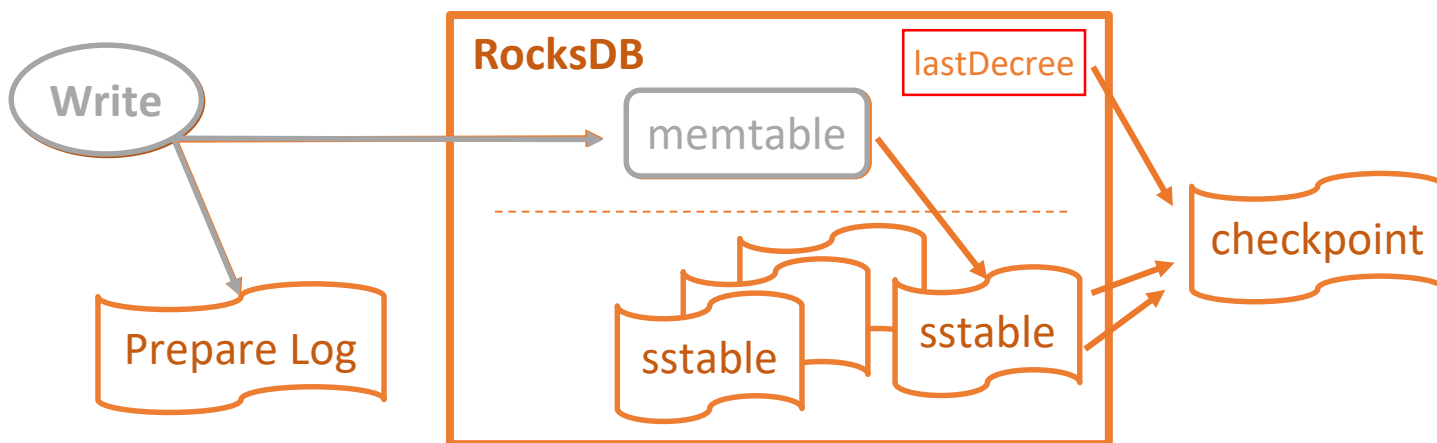


## Lease-Based Failure Detection

# RocksDB supports async-checkpoint

## RocksDB本身支持 Sync-Checkpoint :

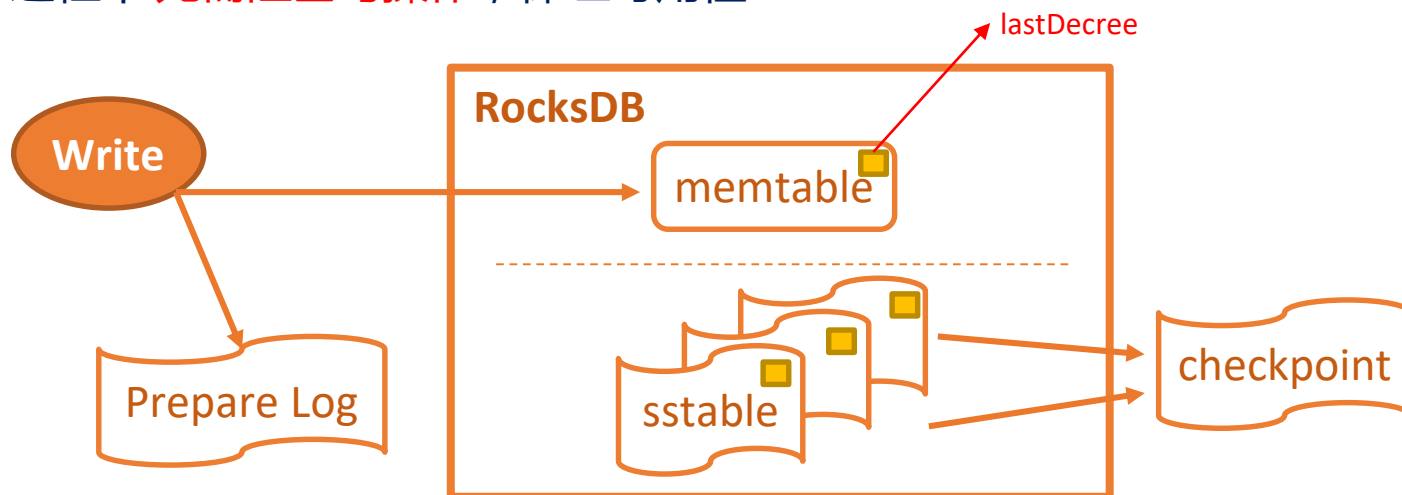
- RocksDB记录最后一次Write的 **lastDecree**
- 在Sync-Checkpoint时，需先 dump memtable，然后 **同步等待** dump完成
- Dump完成后，将元数据和sstable拷贝至Checkpoint，然后使用 lastDecree 进行标记
- 在整个过程中 **需阻塞写操作**，降低可用性



# RocksDB supports async-checkpoint

## 改进的RocksDB支持 Async-Checkpoint :

- RocksDB的memtable/sstable都会记录 自己当前的lastDecree
- 在Async-Checkpoint时，直接忽略memtable的数据，将元数据和sstable拷贝至Checkpoint，并使用所有sstable的 最大lastDecree 作为Checkpoint的decree标记
- 在整个过程中 无需阻塞写操作，保证可用性





## 参考

- PacificA Algorithm
  - <https://www.microsoft.com/en-us/research/wp-content/uploads/2008/02/tr-2008-25.pdf>
- Raft Algorithm
  - <https://raft.github.io/raft.pdf>
- rDSN Framework
  - <https://github.com/Microsoft/rDSN>
- Pegasus System
  - <https://github.com/XiaoMi/pegasus> (coming soon)