阿里云.表格存储
Aliyun TableStore

@十品

NoSQL As A Service

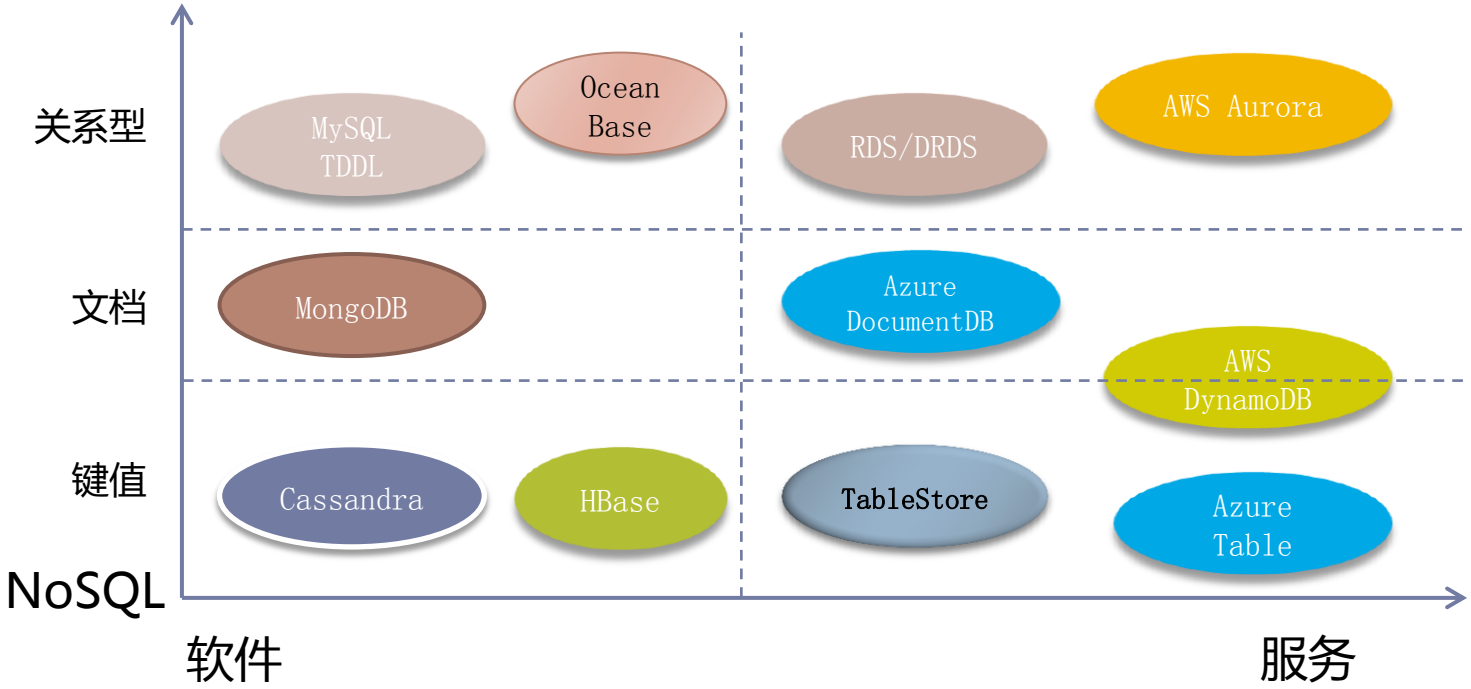
主题

- 需求背景
- 典型应用示例
- 数据模型
- 技术架构
  - 存储子系统
  - 调度子系统
  - 重要技术细节
- 服务化
  - 即开即用 & 按量付费
  - 认证 & 授权
  - 监控报警 & 调试
- 产品路线
- 附录

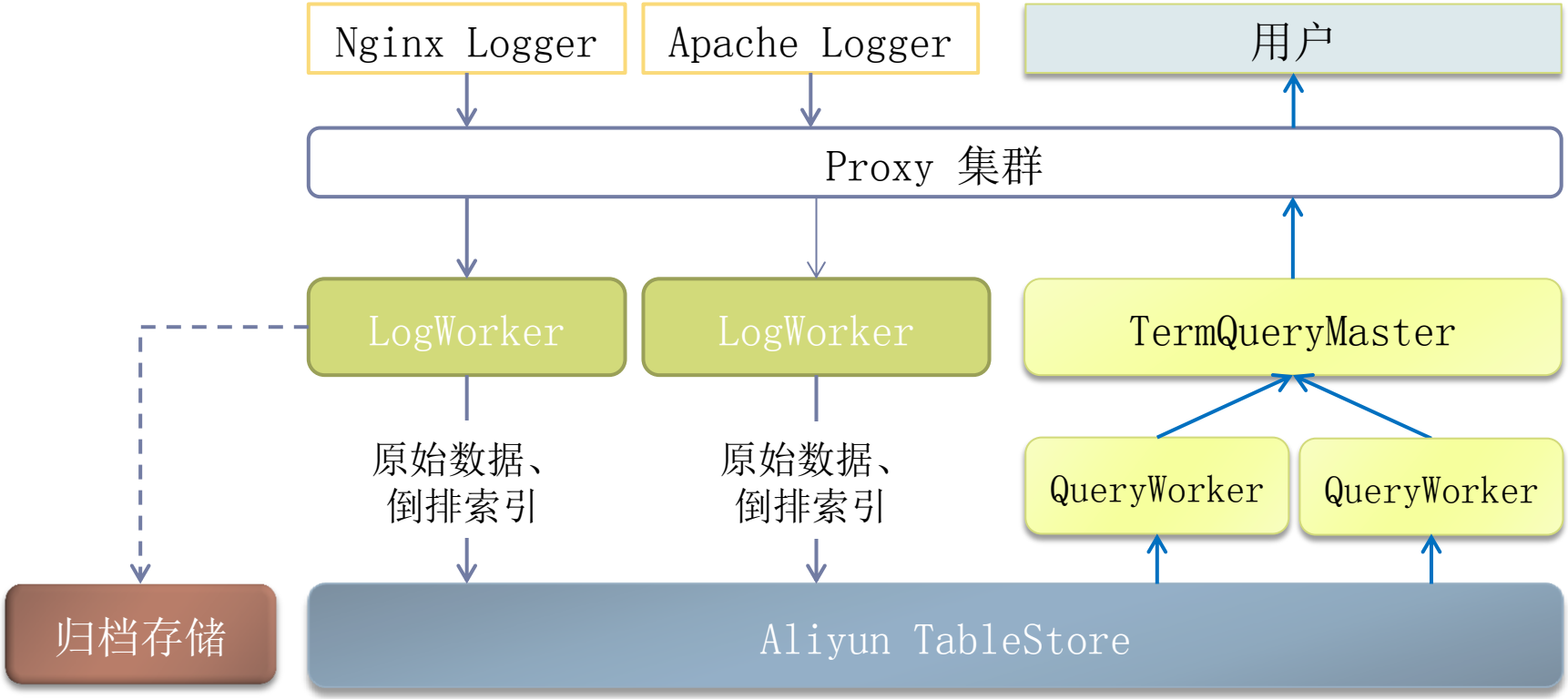
需求背景

- 传统数据库约束
  - 扩展性：资深工程师提前规划，业务逻辑配合调整
  - 编程灵活性：建表指定schema，新业务添加字段需要变更
  - 可用性：机器down机需要运维干预，业务很难无缝切换
- 新时代数据
  - 数据量大，增长快
  - 数据之间没有强关系
  - 高速读写
- NoSQL：基于云计算、灵活易扩展、高可用、弱关系

TableStore定位



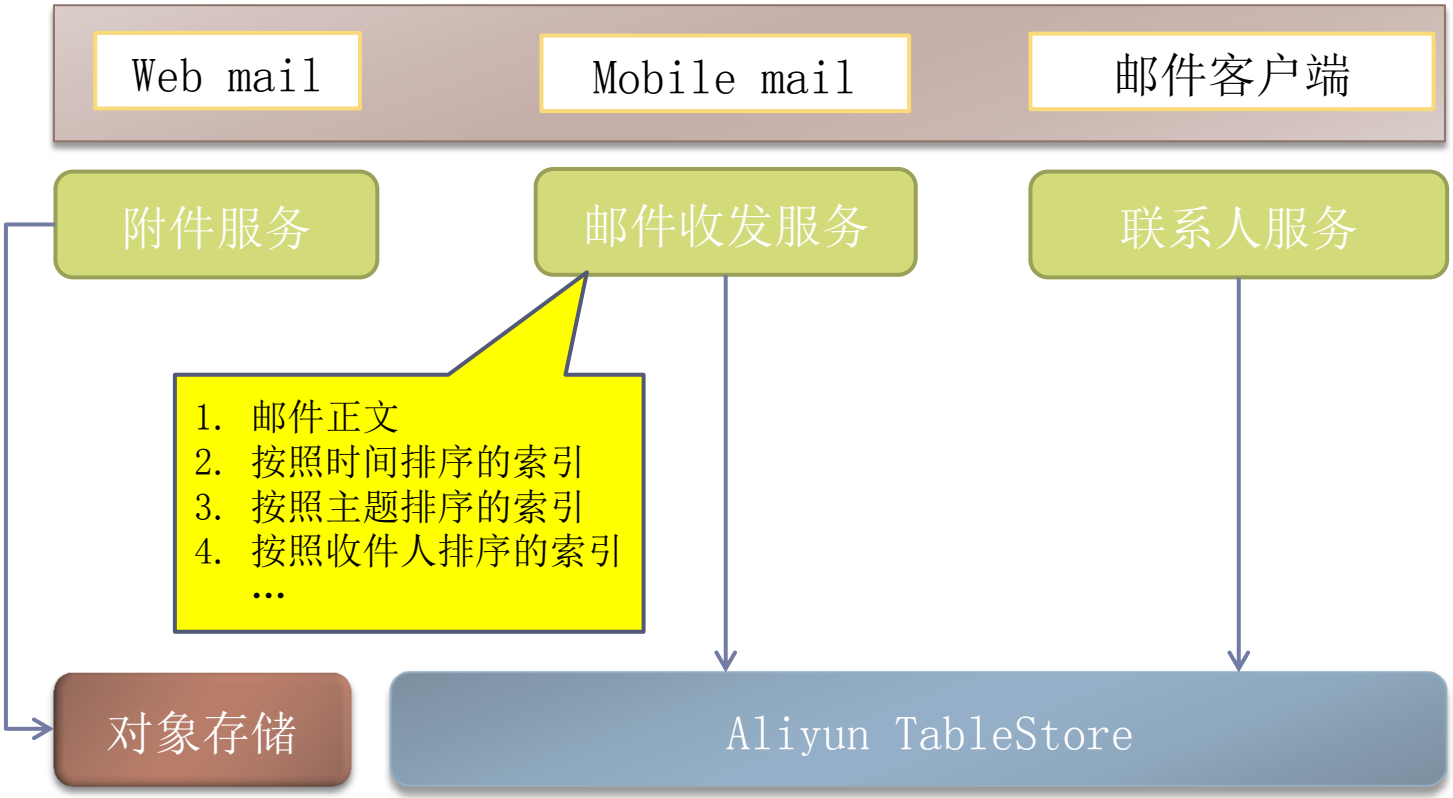
TableStore典型应用 1——日志搜索



TableStore典型应用 1——日志搜索

- 性能
  - 每日数百T数据写入
  - 批量写入平均延时低于50ms：单请求300行，单行100B
  - 批量读取平均延时低于100ms：频度远低于写
- 存储空间优化
  - 高压缩比算法
  - 冷数据选择使用ErasureCoding，空间从3X最低降到1.5X
  - TTL自动清理过期数据

TableStore典型应用 2——邮箱服务



TableStore典型应用 2——邮箱服务

典型需求：

1. 主表按照接收时间索引
2. 能够对发件人、已读未读建立索引
3. 支持对邮件打标签并按照标签分类

TableStore 方案：

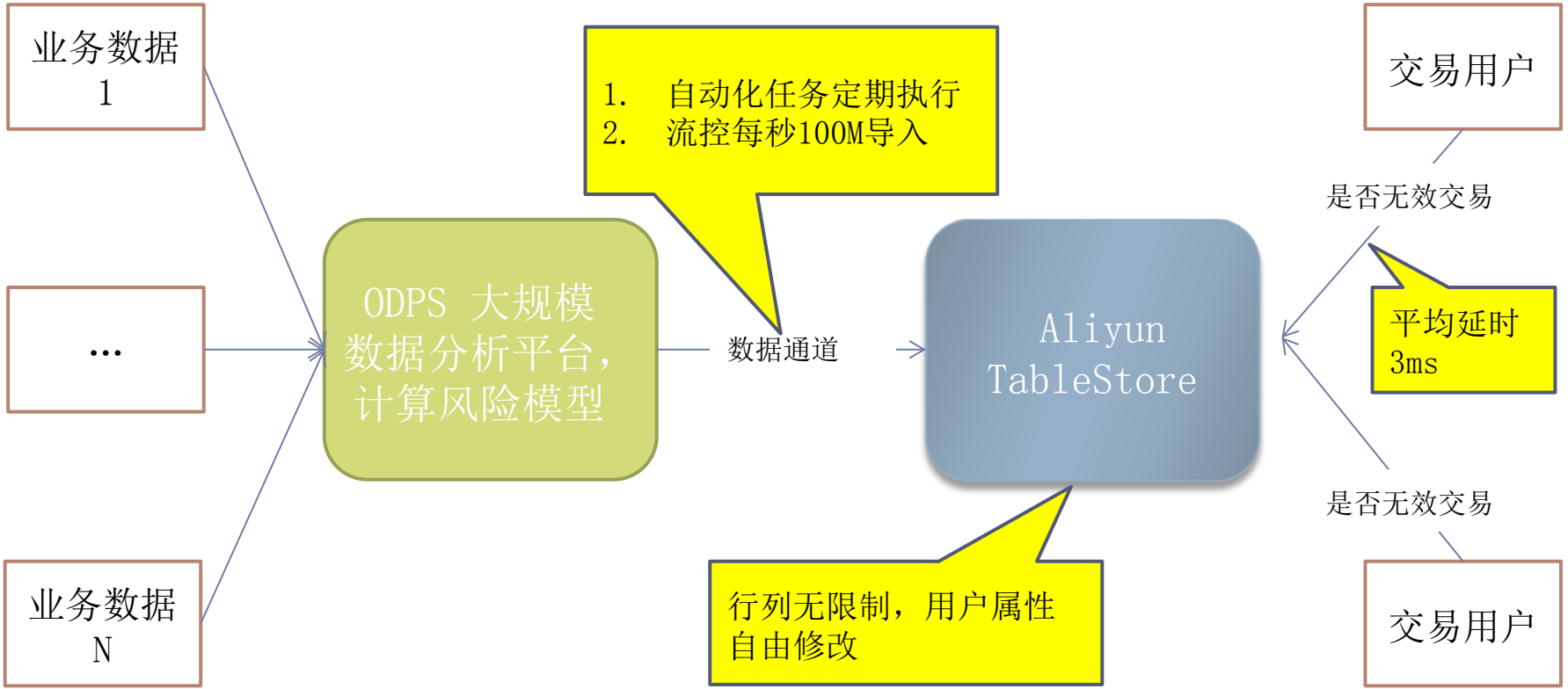
1. 一张大表存主数据和索引数据
2. 带条件更新
3. 第一列PK相同的多行修改保证原子性

Primary key		Columns	备注
user_id	receive_time,mail_id	正文，收件人列表等	主数据行
user_id	sender,send_time,mail_id	按需	发件人索引
user_id	subject,mail_id	按需	主题索引
user_id	tag_id,mail_id	按需	标签分类索引

TableStore典型应用 2——邮箱场景拓展

- 通用的用户信息多维度查询
  - 按照时间对所有信息排序：最近的N条用户活动信息
  - 按照时间对某类数据排序：最近的N条交易信息，最新的N条博客，最近的N条跟其他客户的聊天
- 社交场景可以复用邮箱场景的方案
  - 喜欢：用户最近喜欢的所有帖子
  - 评论：用户最近评论的所有帖子
  - 编辑文档：保存用户最近编辑的N个版本

TableStore典型应用 3——互联网金融风控



TableStore 关键词

高可用

高性价比

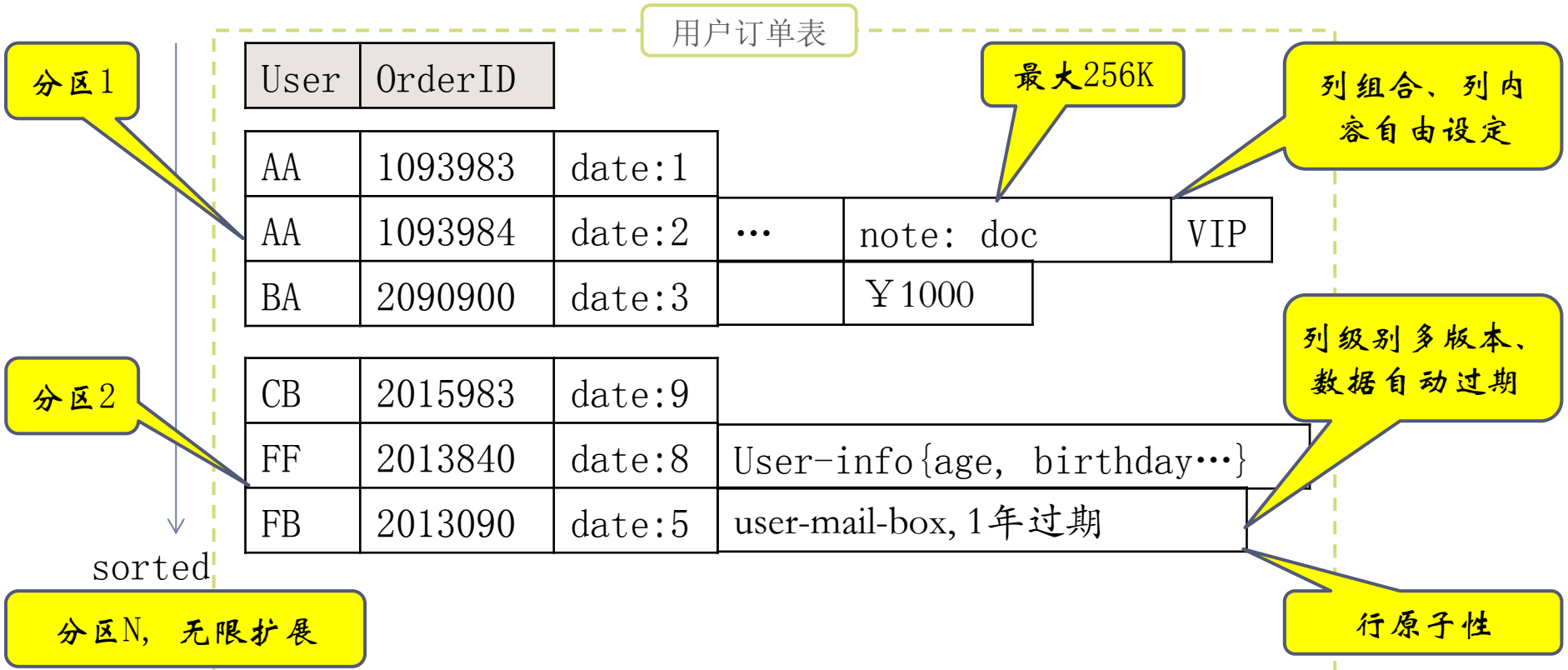
高可靠

水平扩展

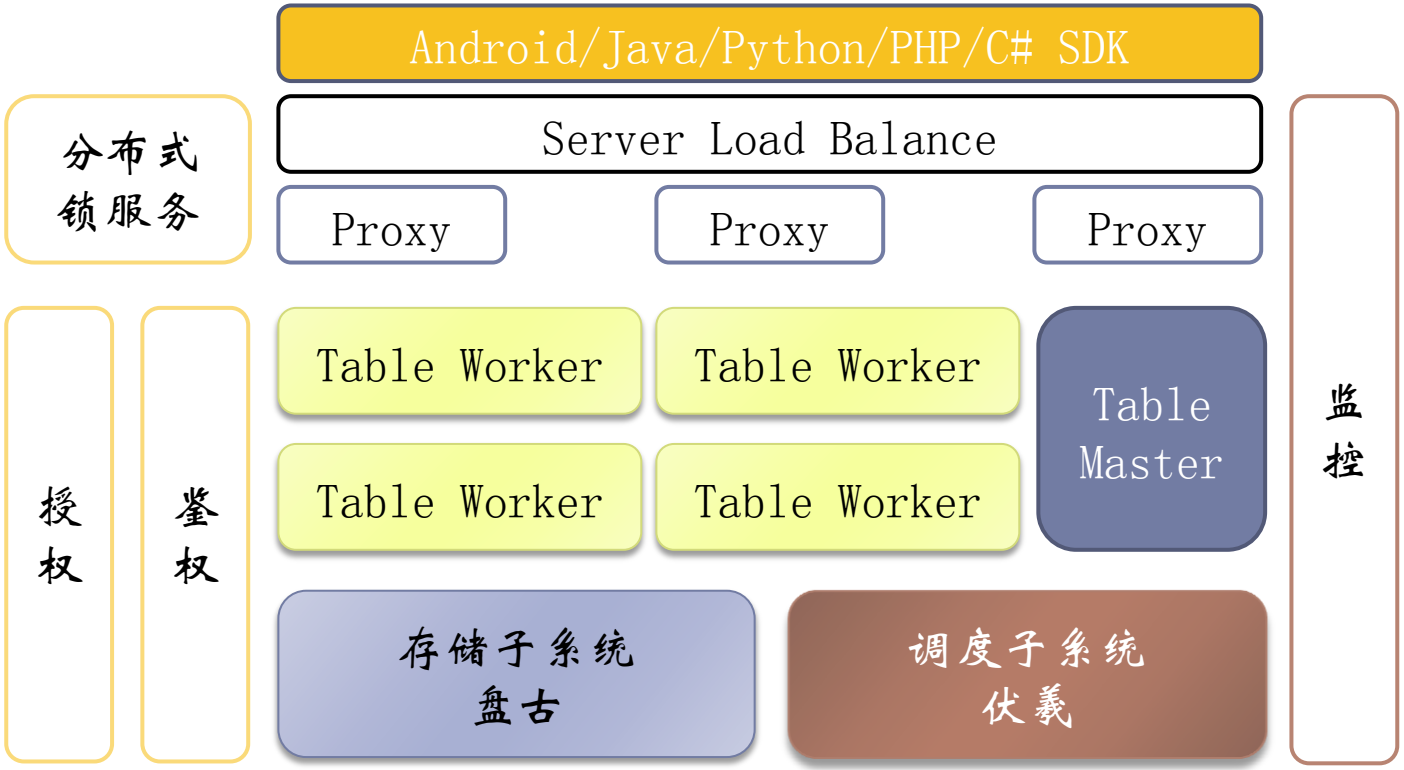
宽行

授权访问

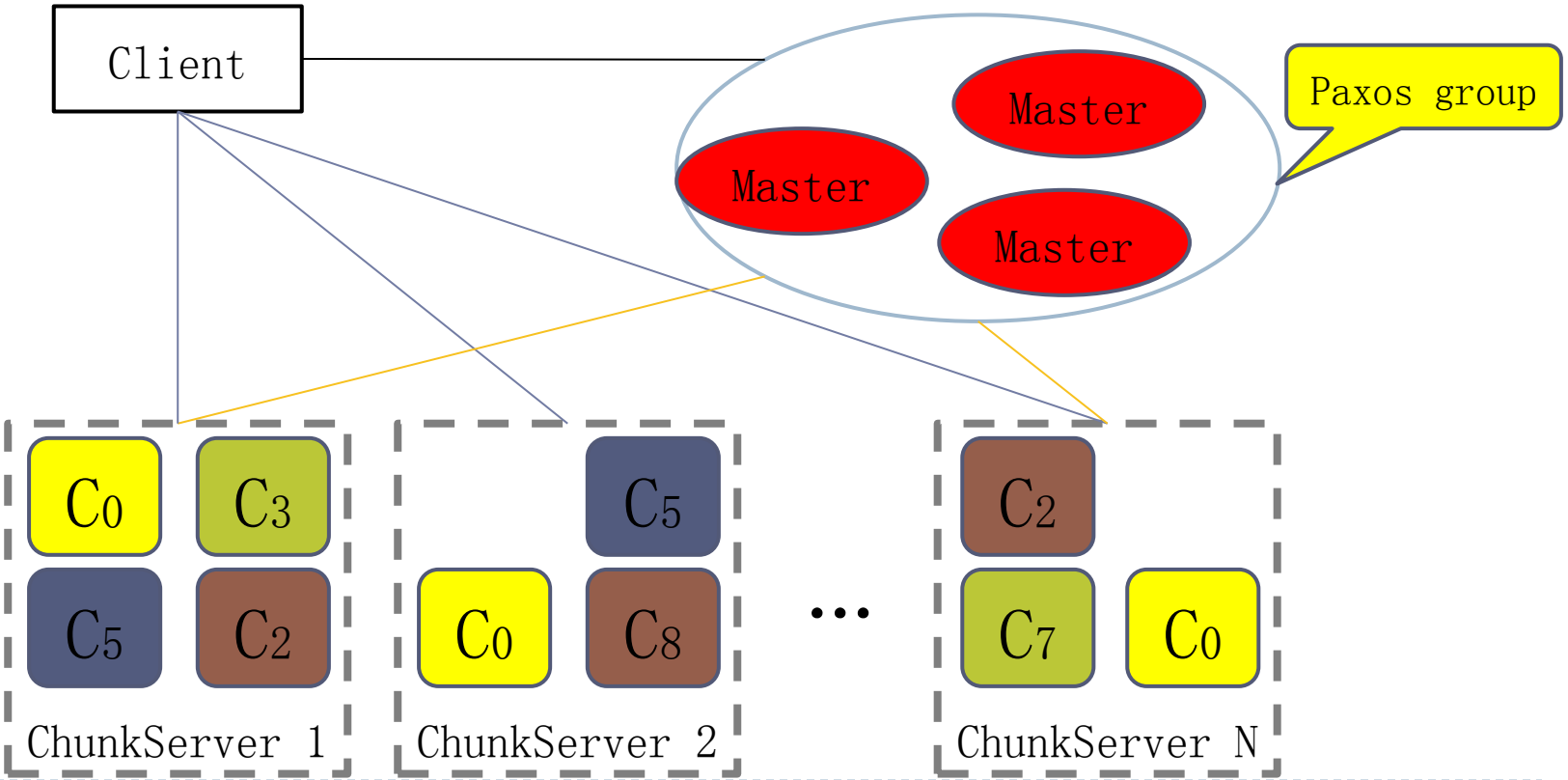
数据模型



技术架构——概览



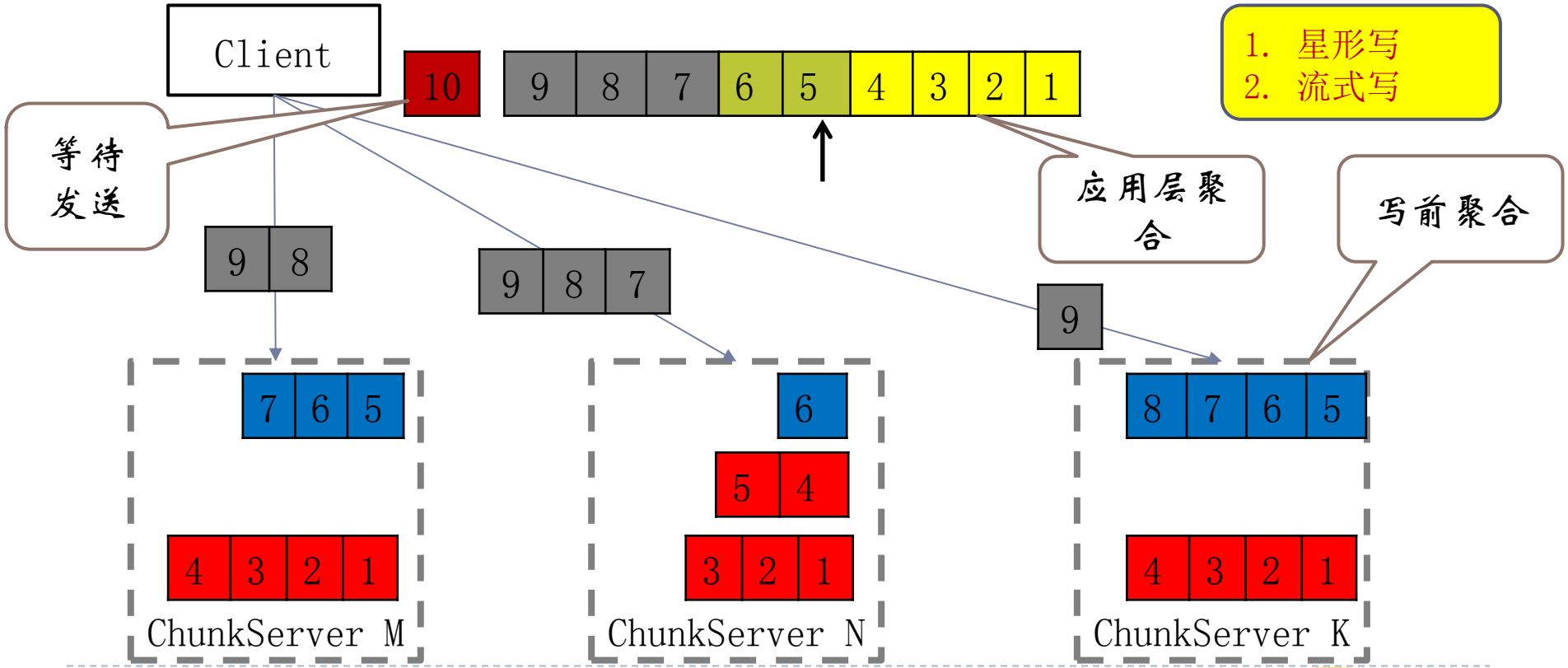
技术架构——盘古存储子系统



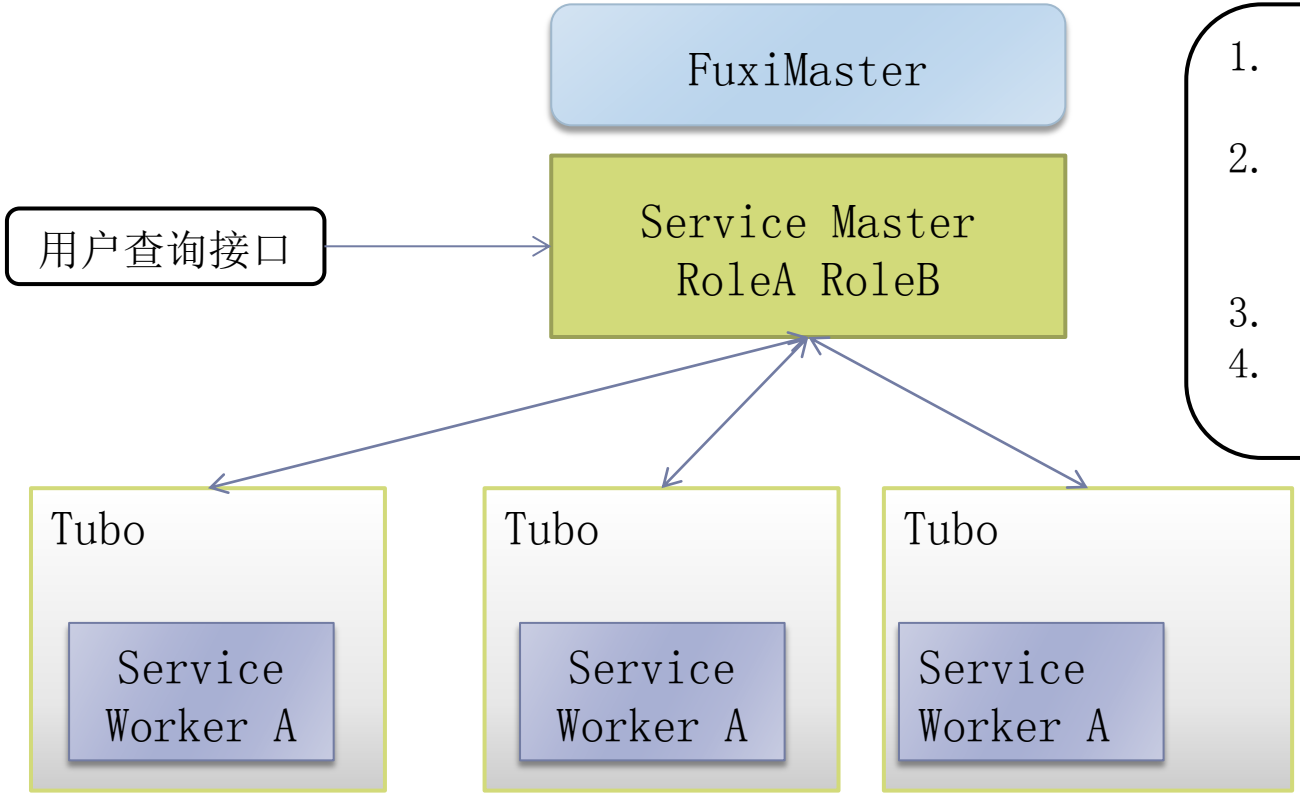
盘古存储系统

- 多Master，秒级Failover
- 请求级别数据校验
- 后台定期数据校验
- 多副本保证，坏盘20分钟完成复制，自动判断优先级
- 慢节点自动规避（慢盘、慢机器）
- IO优先级调度
- ErasureCoding降低存储成本
- 多文件类型支持（LogFile、DataFile）
- 多类型存储设备支持（SATA、SSD、SATA+SSD）
- 坏盘自动隔离

盘古存储系统——日志型文件



技术架构——伏羲调度子系统



1. FuxiMaster 管理集群资源（内存、CPU）
2. ServiceMaster批发资源，为当前服务启动进程
3. 监控&重启服务进程
4. 发现新加入机器并按需启动服务进程

技术架构——TableStore

集群管理

用户实例管理

用户权限管理

集群A

Proxy

1. Validate/TableMetaCache/UserInfoCache
2. ProtocolTransformation

分布式锁服务

集群监控 & 报警

Table Worker

1. RowLock
2. LogFiles
3. DataFiles
4. BlockCache

…

Table Master

1. Meta Storage
2. Plugable scheduler
3. Load balancer
4. Failover
5. Admin API

存储子系统盘古

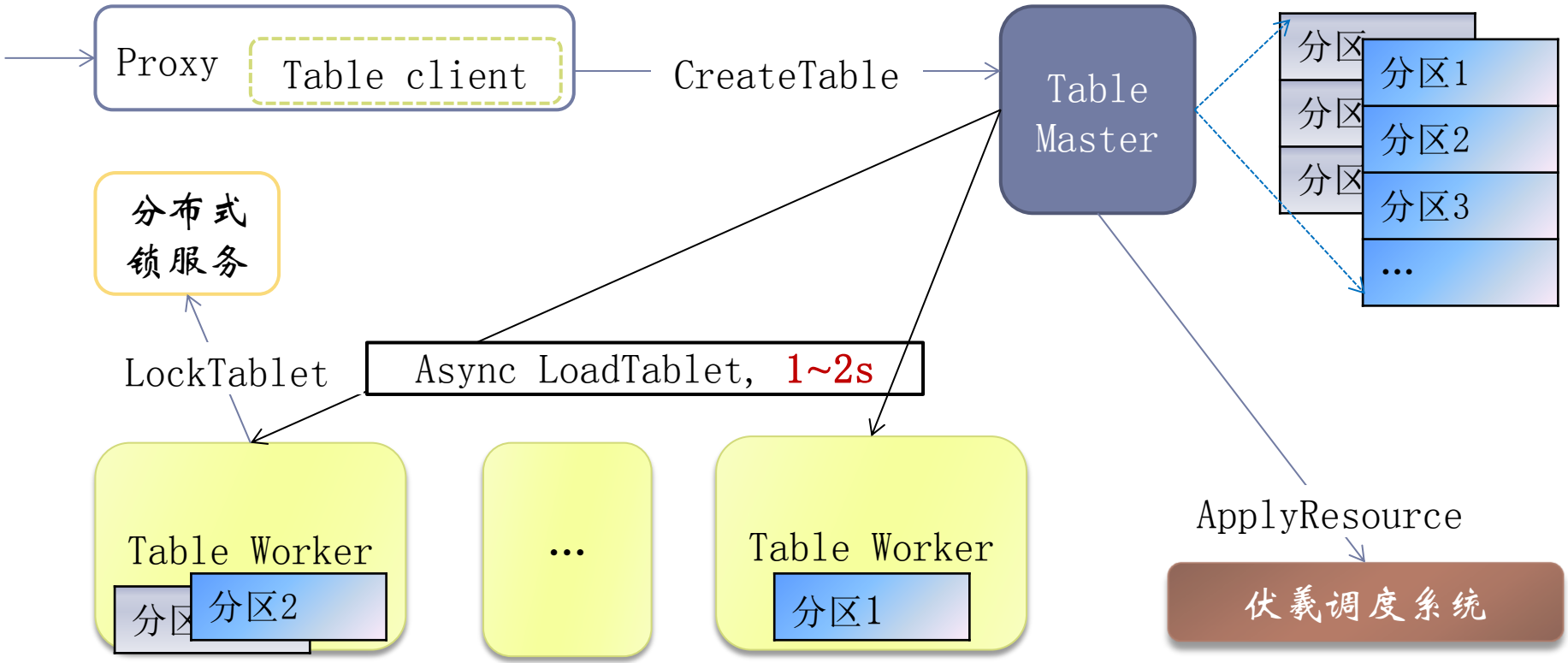
调度子系统伏羲

集群B

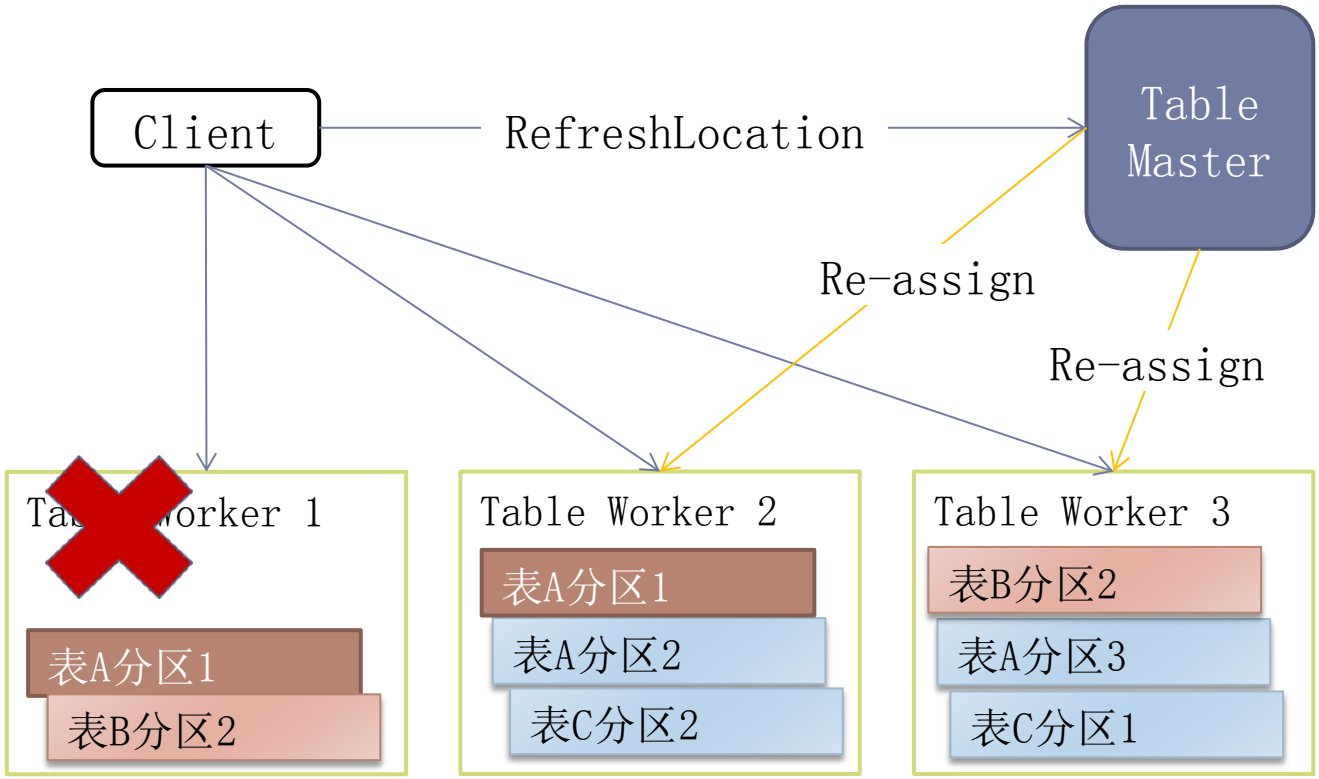
TableStore——用户交互协议

- http/https
- 协议头
  - API 版本，SDK 类型，数据压缩选项，请求ID，请求签名
- 协议内数据格式
  - ProtoBuf2.4：性能和易用性的平衡，30+编程语言支持
  - 性能优化版本5月开放：中间链路零拷贝
- 官方支持SDK：Java/Python/C#/PHP

TableStore——建表

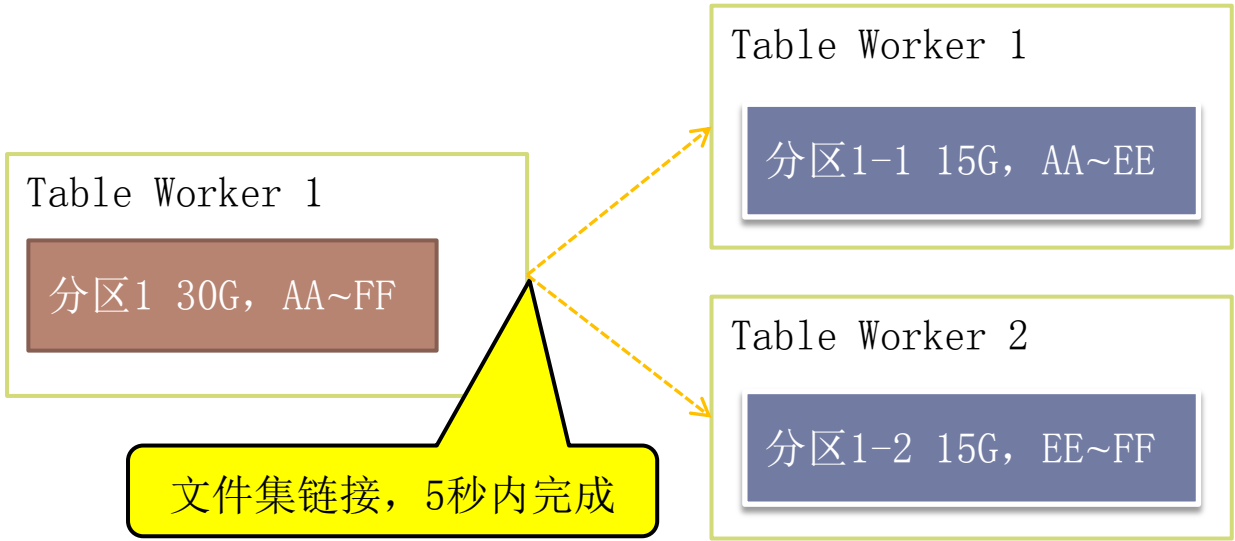


TableStore——自动Failover



1. 进程crash、机器down由伏羲重启
2. TableMaster启动后收集分区加载信息
3. TableWorker crash，重启后分区分配关系不变
4. TableWorker down，TableMaster调度分区到其他低负载机器

TableStore 分区自动扩展

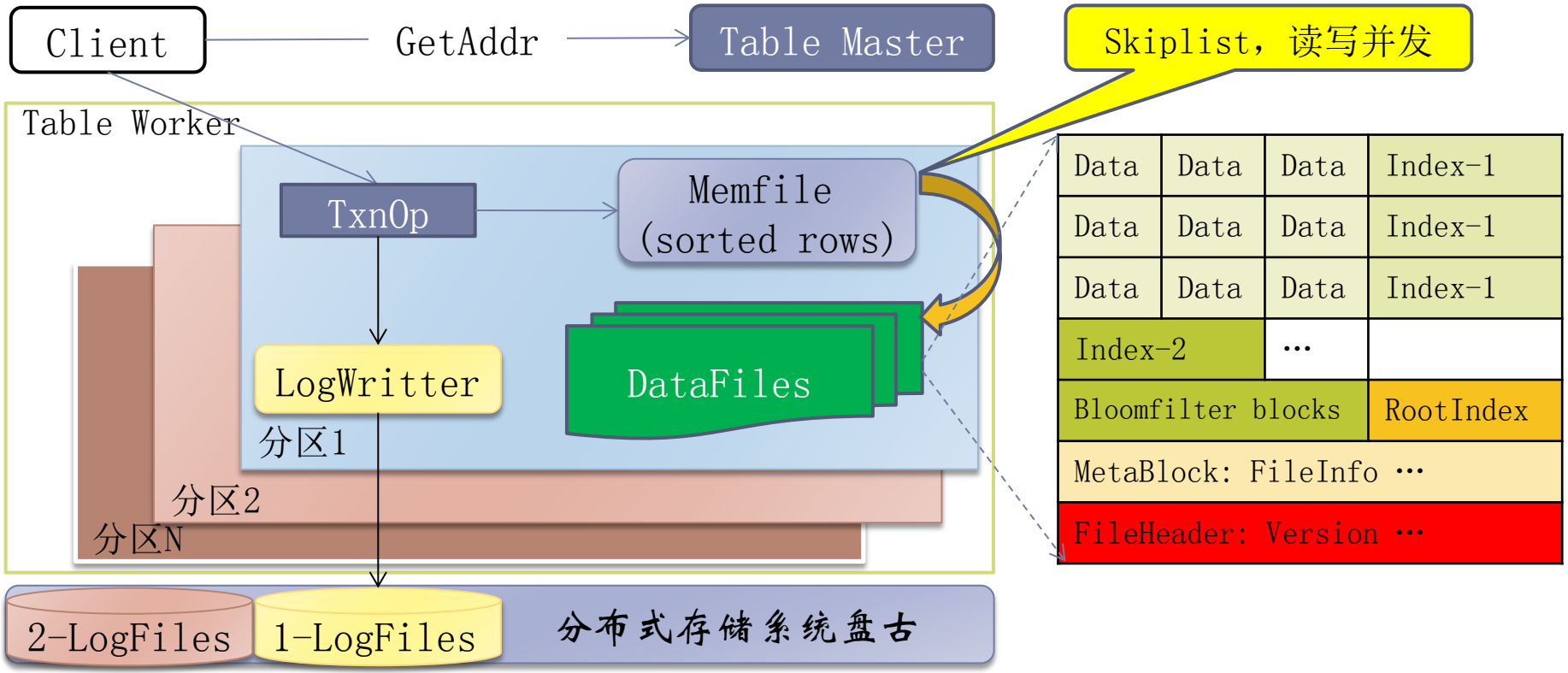


自动分区条件：
1. 分区数据量大于指定值
2. 请求队列长度持续大于制定值
3. 每秒请求个数持续大于指定值
4. CPU使用持续大于指定值
5. 运维控制（数据导入）

分区1分裂时，分区1文件集会被链接到分区1-1和分区1-2内，避免数据拷贝

分布式存储系统存储系统盘古

TableStore——写流程



TableStore——单行并发写优化IO

```
Write-1 {row-A:
                col-X:value-X,
                col-Y:value-Y}
```

```
Write-2 {row-A:
                col-Z:value-Z,
                col-K:value-K}
```

```
Write-3 {row-A:
                col-X:value-P,
                col-V:value-V}
```

同时到达的三个请求

Lock Row-A

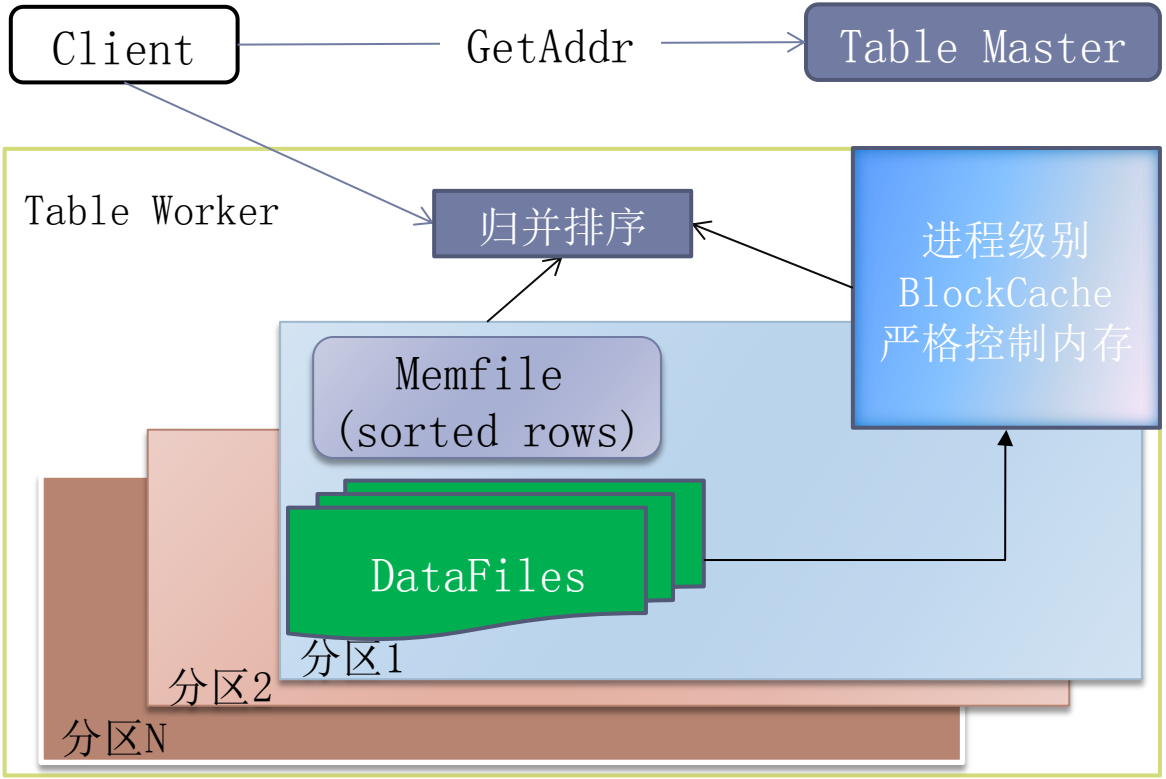
日志合并提交，一次IO

```
LogEntry { row-A: col-X:value-X,
                  col-Y:value-Y,
                  col-Z:value-Z,
                  col-K:value-K,
                  col-X:value-P,
                  col-V:valueV }
```

内存提交

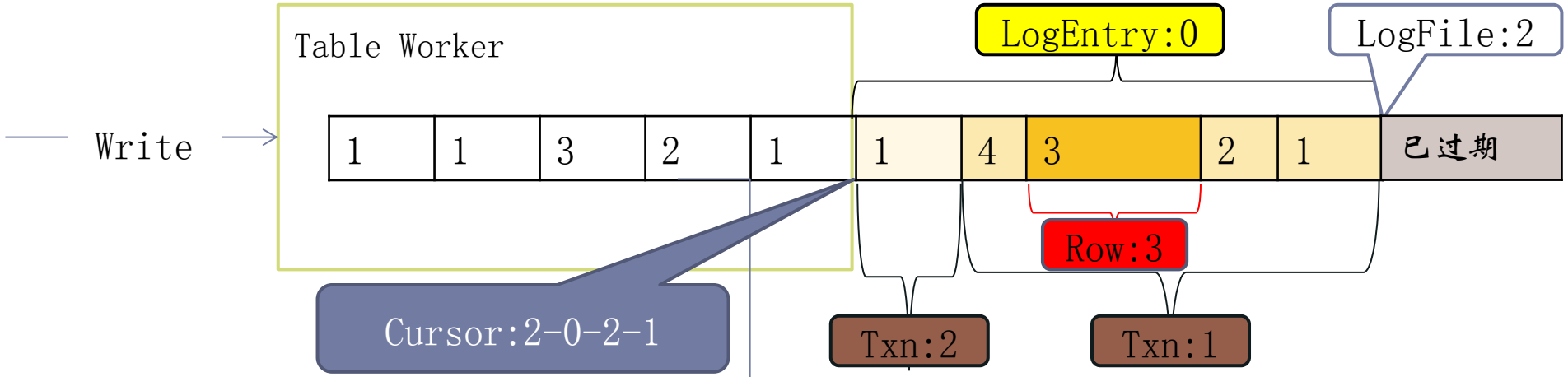
```
Memfile { row-A: col-X:value-P,
                 col-Y:value-Y,
                 col-Z:value-Z,
                 col-K:value-K,
                 col-V:valueV }
```

TableStore——读流程



1. 一份Replica在本地，读数据不需跨机器
2. BlockCache分冷热两级、LRU淘汰
3. 多种策略提高读性能：数据属性启发文件合并、并行文件合并、写流控
4. Scan支持流式读

TableStore——日志流



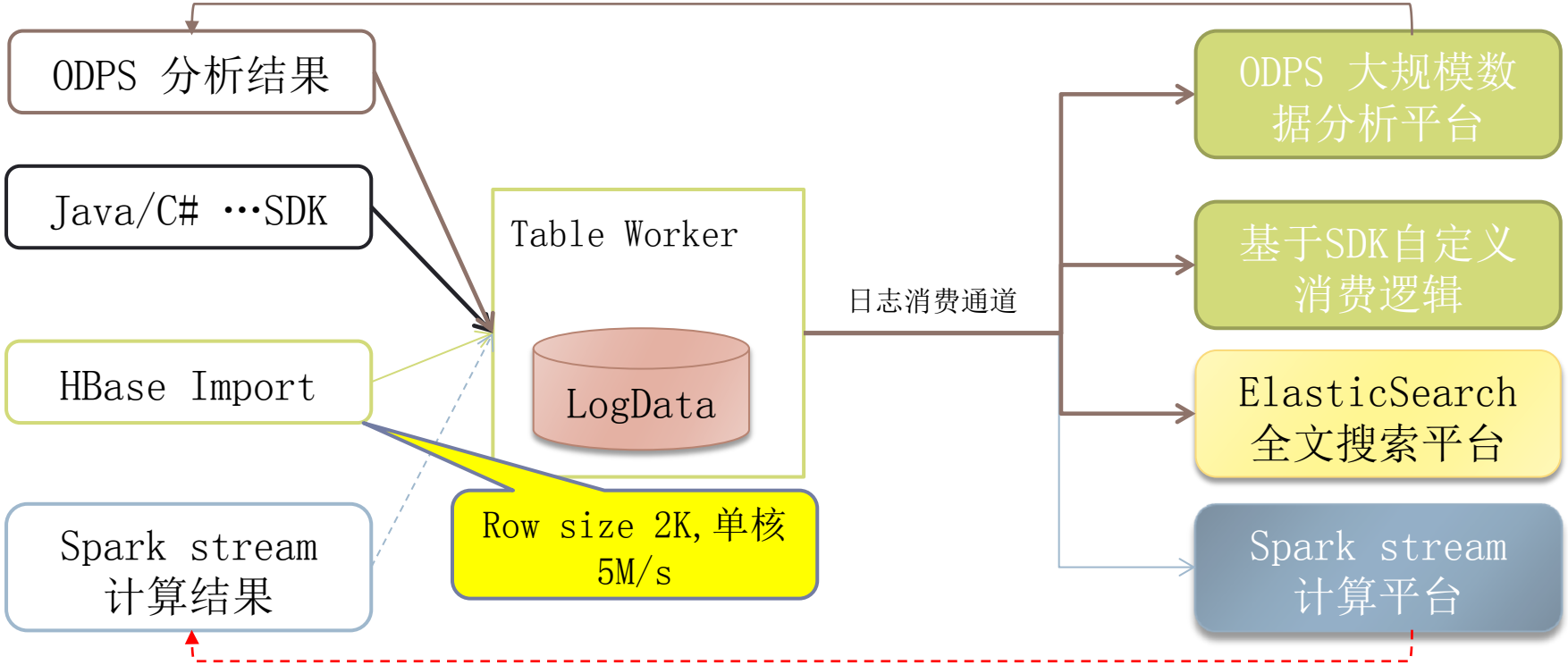
1. 24小时过期
2. ListShards 获取所有日志流分区
3. For each shard，读取未过期数据，Worker返回cursor用于下一次读取
4. 返回空表示没有写入，sleep and retry

Cursor + Rows

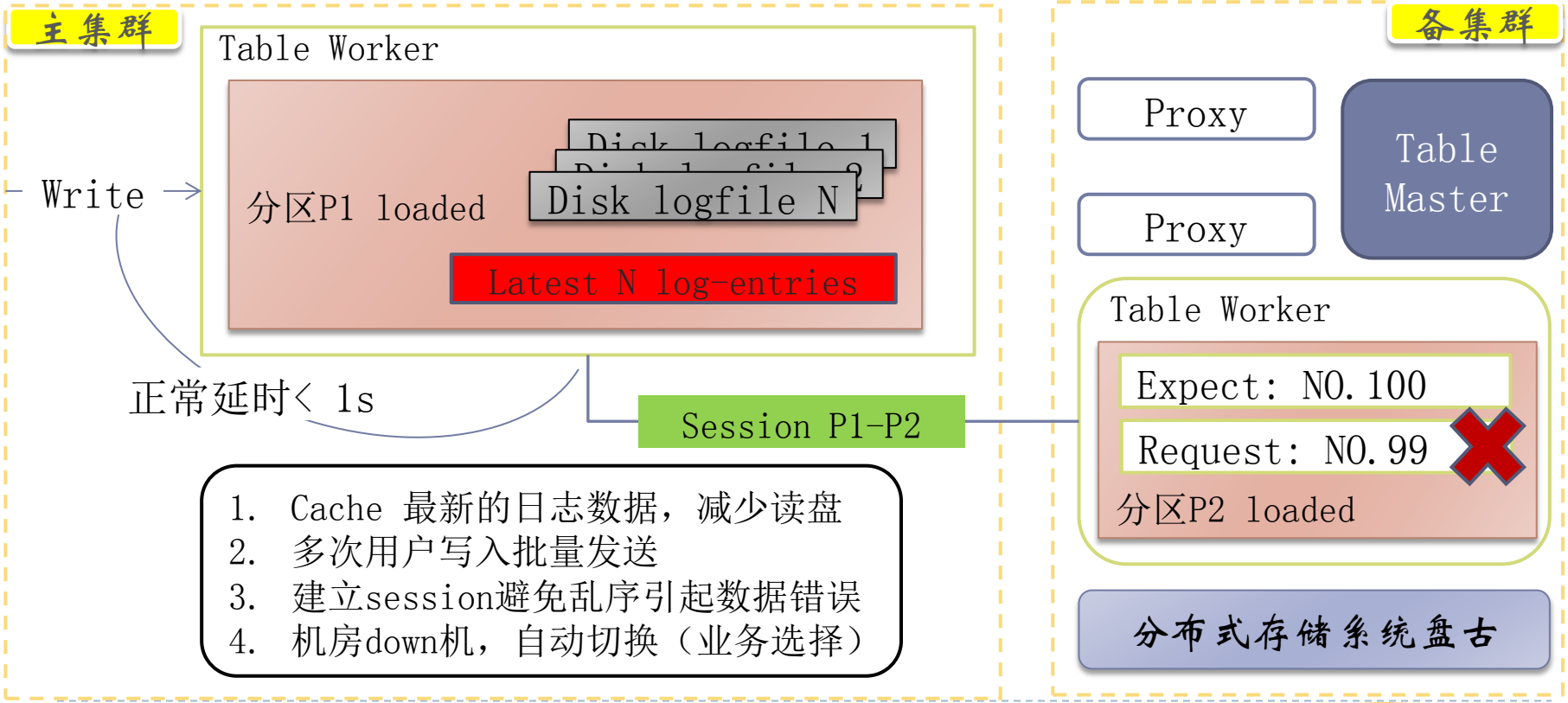
Log consumer

SDK Stream client

TableStore——数据流转



TableStore——跨集群数据同步



TableStore——服务化

- 即开即用：服务自主开通，开通后即可测试示例代码
- 按量付费
- 多租户：削峰填谷降成本
- 请求认证：防止伪造请求、保护数据安全
- 授权模式多样：满足企业级权限管理需求
- 监控报警自动化：随时了解应用状态
- 公平流控
  - 保证预留读写资源的用户请求
  - 超过预留读写资源的请求按照超出量流控，超出多则被流控多

TableStore——请求认证

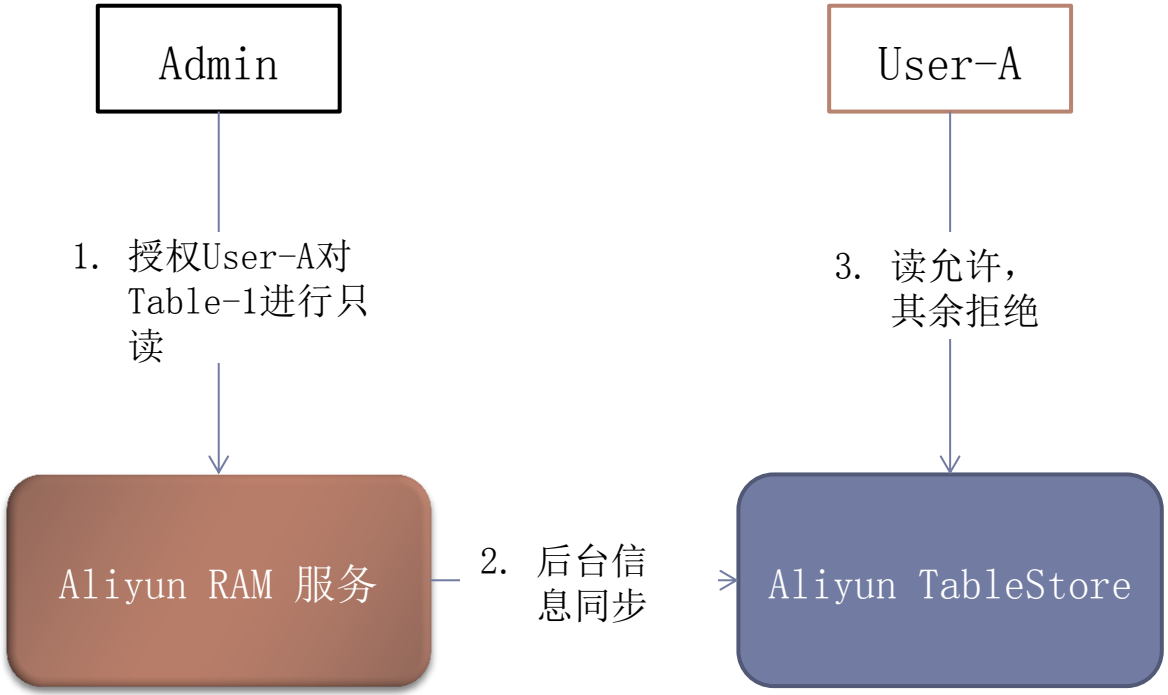
SDK 请求中包含 ：
1. accessId
2. 请求数据
3. 使用accessKey的签名
4. 请求发送时间

Proxy 验证请求：
1. 请求发送时间在最近15分钟内
2. 根据accessId获取accessKey
3. 使用accessKey签名请求数据
4. 校验请求数据签名与用户传入一致

响应数据中包含 ：
1. accessId
2. 响应数据
3. 使用accessKey的签名
4. 响应发送时间

SDK 验证响应：
1. 响应发送时间在最近15分钟内
2. 使用用户提供的accessKey签名响应数据
3. 校验响应数据签名与Proxy返回一致

TableStore——权限管理



1. 主、子账号访问
2. 基于用户组和角色的授权
3. IP黑白名单
4. 强制SSL访问
5. 过期权限自动清除
6. 权限粒度到实例-表-操作级别

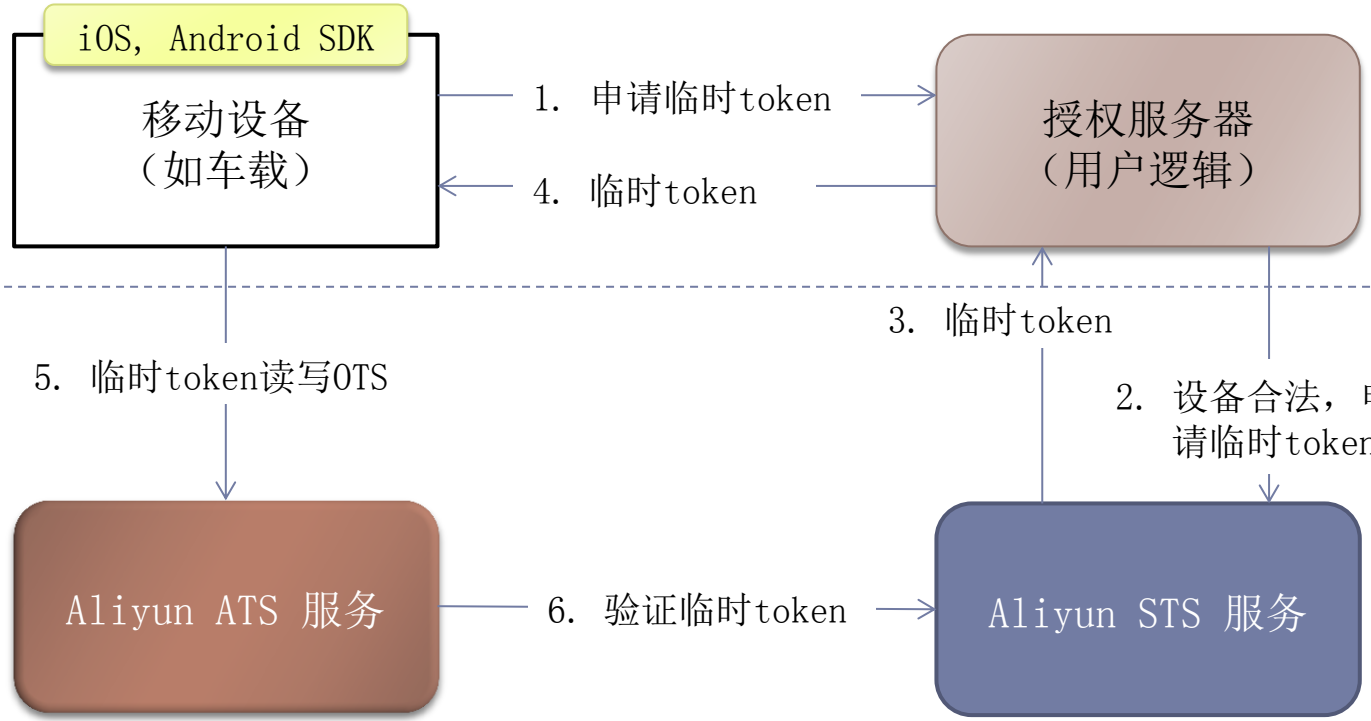
TableStore——VPC

- 支持阿里云VPC设置
- 两种安全模式
  - 允许任意网络访问
  - 只允许VPC内网络访问
- 支持混合云，企业逐步上云
  - 高速通道提供10G带宽，稳定快速
  - 自建VPN网关，便宜方便

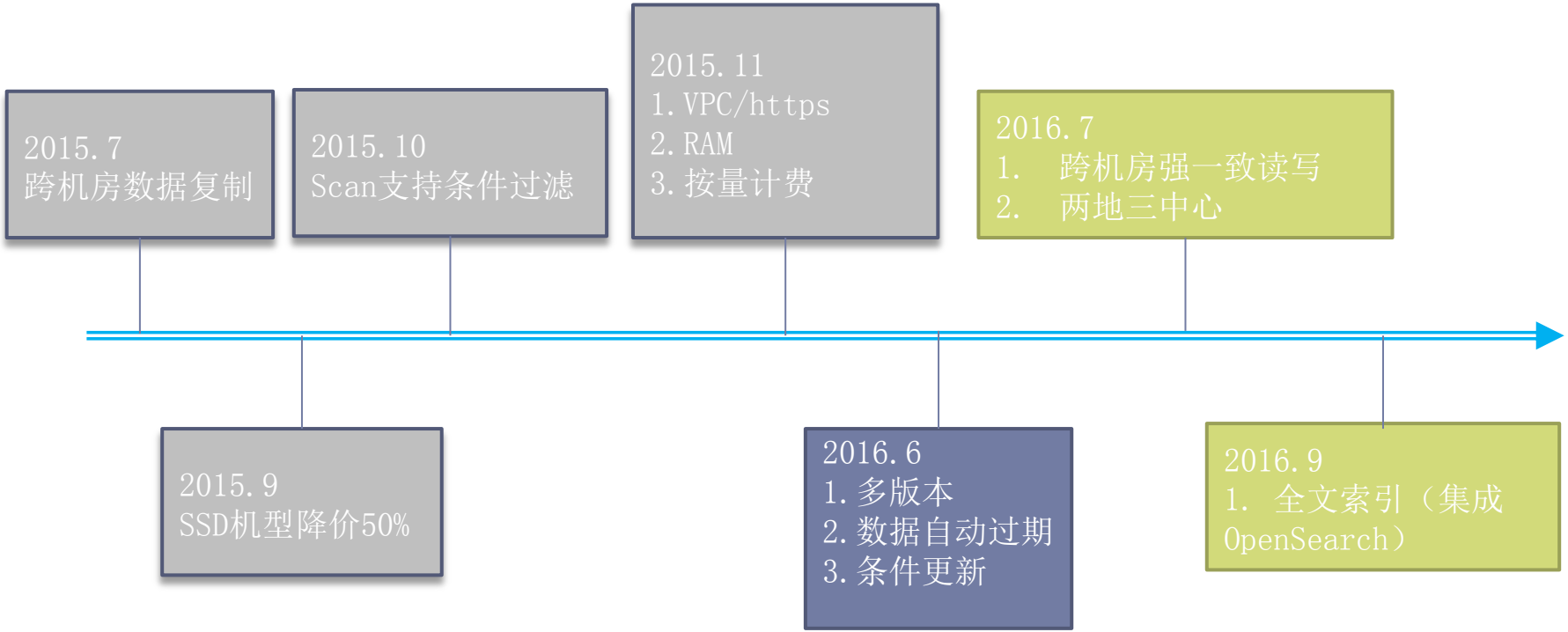
TableStore——集成调试系统

- 全链路请求追踪
  - 全局唯一请求ID标识每个请求
  - 请求ID贯穿SDK、Proxy、TableWorker、盘古文件系统、磁盘读写模块
  - 异常请求全链路记录日志
  - 延时过长请求全链路记录日志
- 动态更改日志配置
- 热点PK即时定位
  - 协助用户发现异常请求，迅速定位问题

TableStore——移动开发



TableStore——产品路线



附录1 联系TableStore

官方论坛

该二维码7天内(5月10日前)有效，重新进入将更新

微信交流

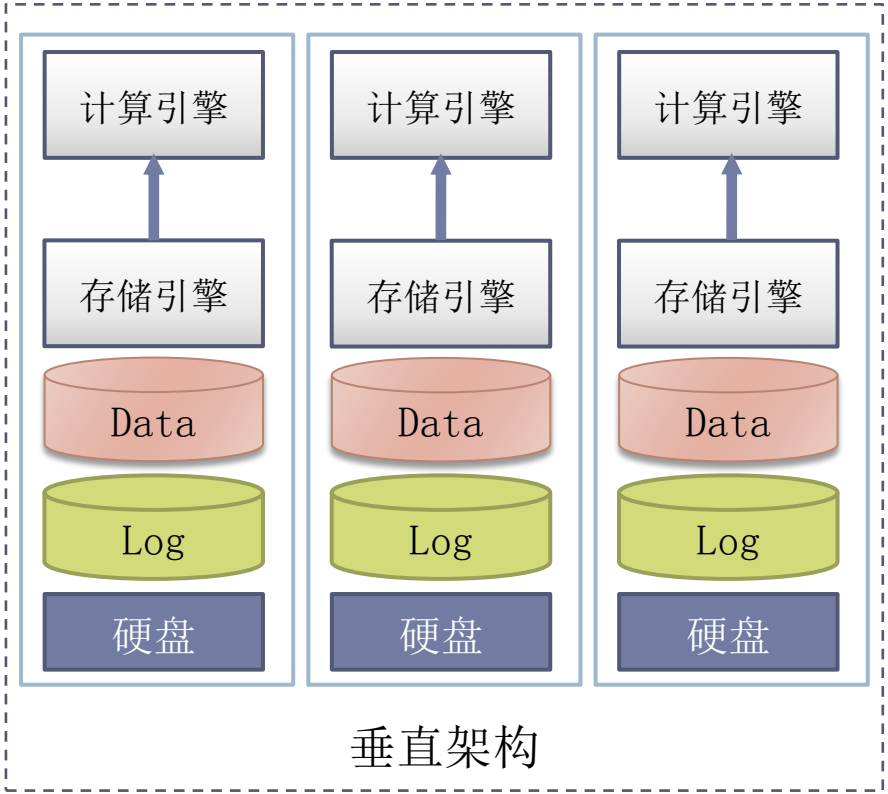
该钉钉群二维码将在2016-05-10失效

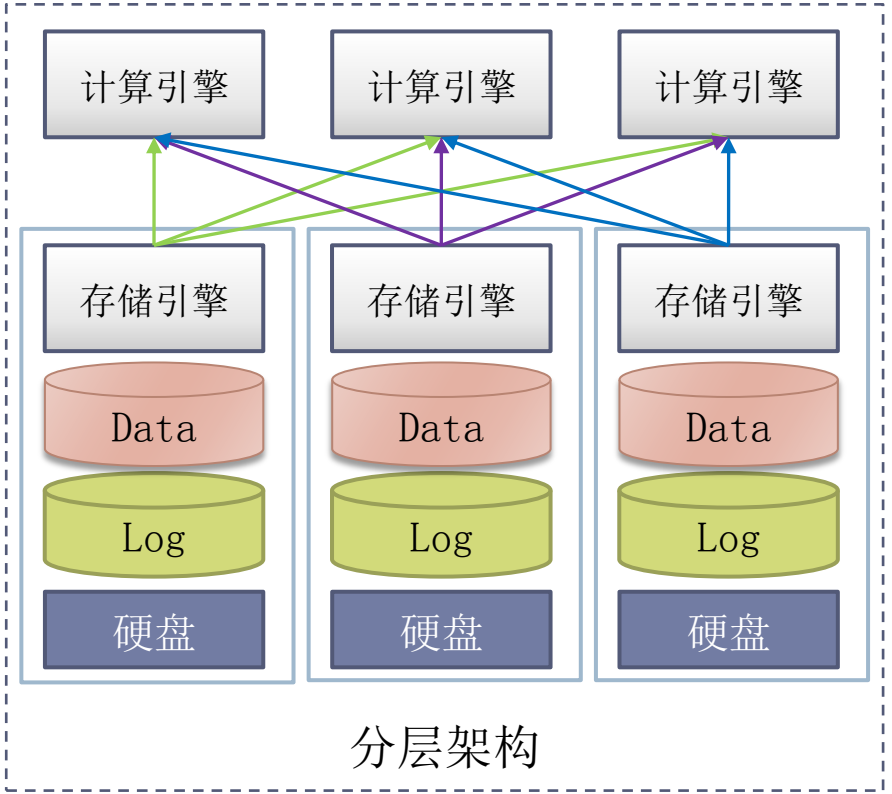
钉钉交流

附录2 TableStore调优经验

- Proxy 全协程化，减少线程切换开销
- 前后端统一协议格式，避免协议转换和拷贝
- ProtoBuf支持指针赋值，消除数据拷贝
- 更快的压缩、解压算法(lz4解压性能超过snappy)
- 网卡中断平衡
- ext4预分文件空间减少meta刷盘
- 全链路tracer精确定位问题
- Pipeline 日志写

附录3 架构探讨 垂直 VS 分层

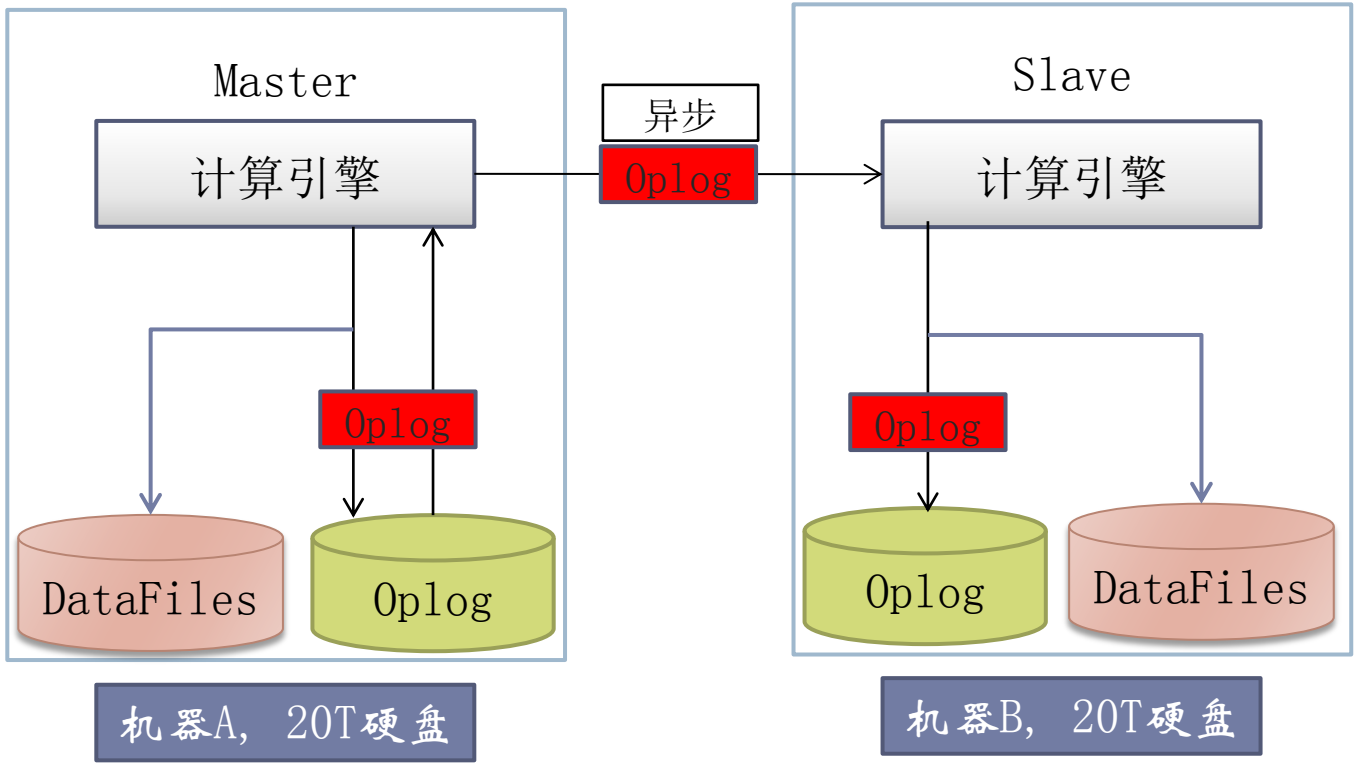




附录4 架构探讨 垂直 VS 分层 cont.

	垂直架构	分层架构
基本特性	计算存储强绑定，存储引擎为计算引擎量身定做，很难接入其他计算引擎	计算存储分离，存储引擎可以方便接入不同计算引擎
扩展能力	需要拷贝数据，小时级别完成	无需拷贝数据，秒级完成
数据一致性	强一致性能差，主流方案不保证主备强一致，备可能丢失最近更新	保证强一致
运维成本	备库down需要重新从主库拷贝数据，主库成为单点，一般需要运维守护	自动完成数据复制，运维定期报修机器即可
计算成本	主备计算引擎各自独立，无法共享计算结果	计算引擎计算结果拷贝到存储引擎，无需重新计算
存储成本	主备存储引擎独立，无法共享信息降低存储空间占用	共享存储信息，ErasureCoding技术降低存储空间一半以上
故障隔离	计算、存储完全隔离，灰度升级容易做	存储共享，灰度升级不容易做
典型产品	MySQL，MongoDB，Redis，Cassandra	TableStore，HBase，HyperTable

附录5 垂直架构数据库主备



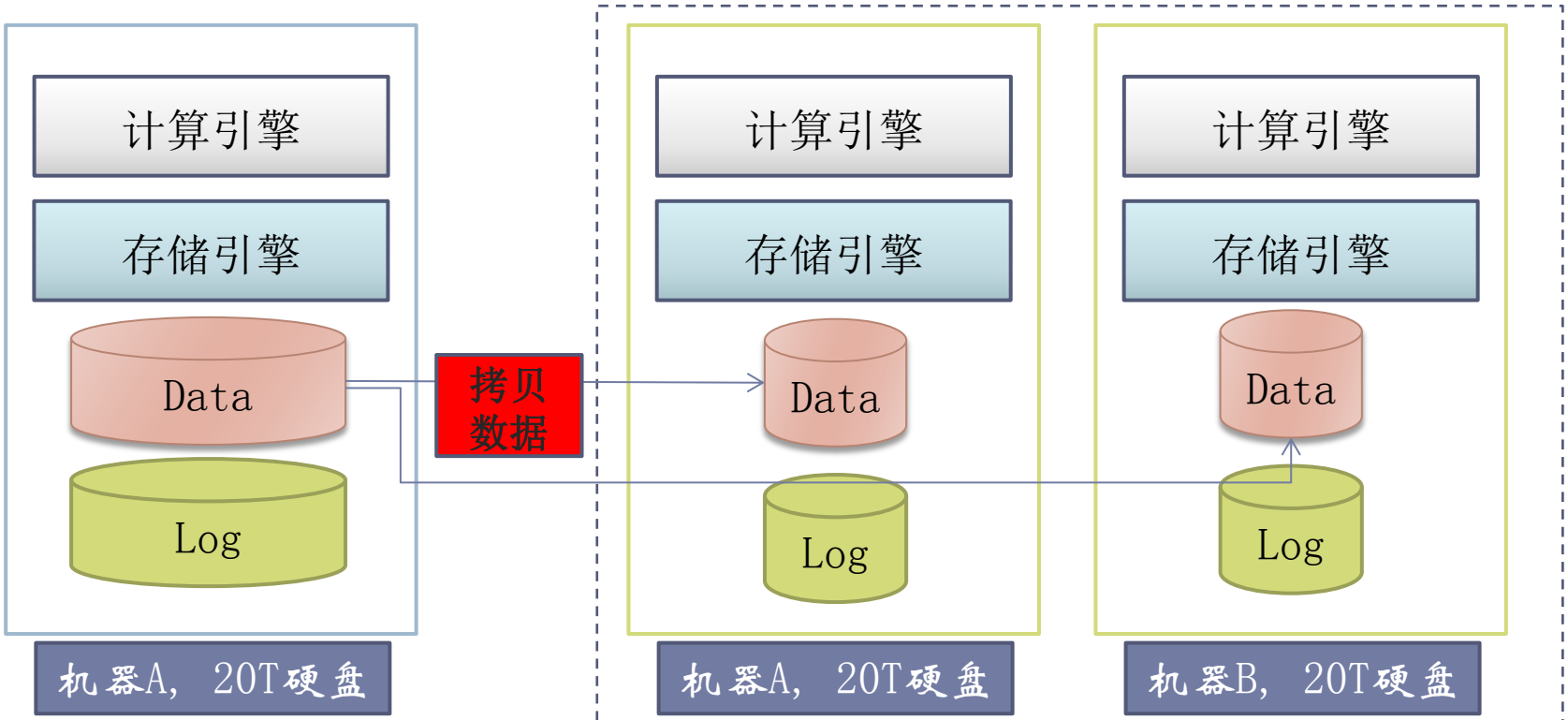
附录6 运维代价——垂直 VS 分层

机器硬盘坏，无法启动	
垂直架构（MySQL、MongoDB）	水平架构（TableStore/HBase）
1. 找台机器准备启动slave，要求硬盘空间够，机器不是特别忙 2. Master做snapshot，将数据文件拷贝到新的slave，考虑到不能影响master访问，如果slave不是空机器，不能影响slave访问，30M/s已经很快了，5T数据需要5*1024*1024/30/3600=48小时 3. 数据拷贝完成后，将snapshot之后的日志文件传输到slave并应用上去，直到追平 4. 一个额外的问题是不少数据库产品的snapshot功能并不完善，拷贝数据文件跟提供在线服务是矛盾的，长时间拷贝数据文件无法支持，通过一主多备可以缓解这种情况 5. 一般情况下，运维起夜处理	1. 盘古Master发现ChunkServer下线 2. 将该ChunkServer上的文件列表散布在整个集群进行M对N的复制，流控30M，集群规模100台，5*1024*1024/30/60/3600=1小时，60台机器是考虑了数据在不同交换机的分布安全 3. 运维无感知

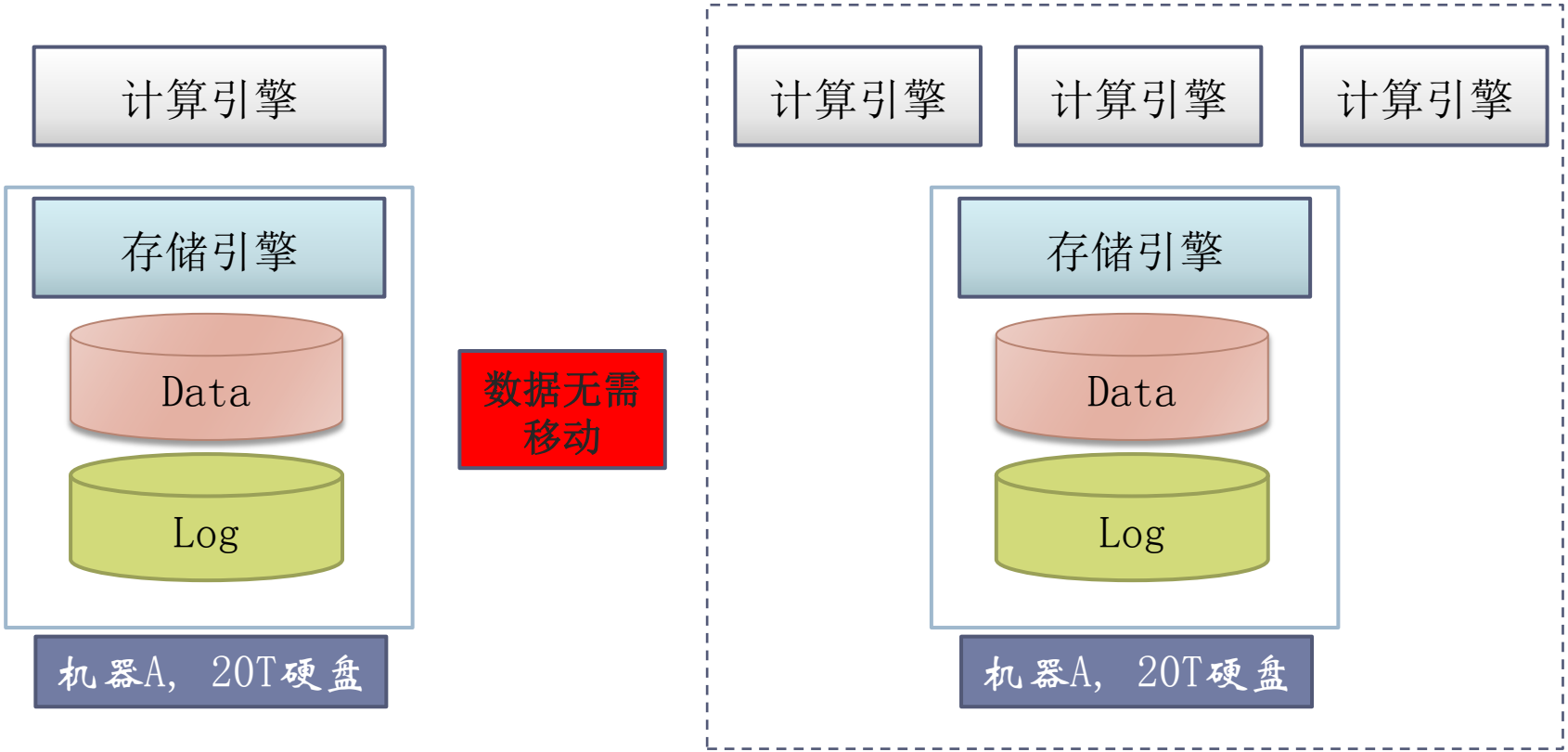
附录7 数据一致性——垂直 VS 分层

- 垂直架构
  - 异步：主库down，最近的更新丢失；备机读获取的是不一致的数据
  - 半同步：先主后从，延时翻倍；从库down机，自动切换成异步
  - 同步：延时、可用性基本不可忍受
- 分层架构
  - 分布式存储系统同时写出N(一般是3)份日志数据，不会导致延时上升
  - 等待N份都写入完毕，返回成功，保证数据一致性
  - 没有主从的概念，任何机器down导致其中一份写入失败，从其余可用机器中再选择一台，重新写入

附录8.1 扩展性——垂直



附录8.2 扩展性——分层



附录9.1 计算成本——垂直

1. 主备完成的计算量相当，备很难重用主完成的计算结果
2. 有些数据库主需要为备准备专门格式的日志，增加了计算量

附录9.2 计算成本——分层

计算引擎只处理一遍用户请求，处理的结果直接拷贝到多个存储节点上，比起垂直结构，节约大量计算资源

附录10 存储成本——垂直 VS 分层

- 垂直
  - 主备（或者多个备）无法共享存储信息，N份冗余需要N倍存储空间
- 分层
  - 分布式存储系统可以共享冗余信息，ErasureCoding技术可以将N份冗余需要的存储空间降低到低于N/2；较耗费CPU资源，一般针对较冷数据

附录11 TableStore VS 自建HBase

对比项	Aliyun TableStore	自建HBase
数据模型	基于行，列数不限	基于行，列数不限
数据类型	String, Integer, Boolean, Double, Binary	Binary
数据多版本	支持	支持
数据自动过期	支持	支持
日志流	支持	不支持
二级索引	不支持	官方不支持，有相关实现
事务级别	行级事务	行级事务
规模线性扩展	支持	支持
高可用性	一键开通跨机房容灾模式	自建跨机房容灾
数据通道	无缝集成大规模数据计算平台ODPS	集成Hadoop生态组件
安全性	请求校验、主子账户、操作授权、https	不支持
服务托管	用户无需关心硬件维修、软件升级	用户负责硬件维修、软件调优

附录12

- 表格存储（TableStore）曾用名：Open Table Service (OTS)