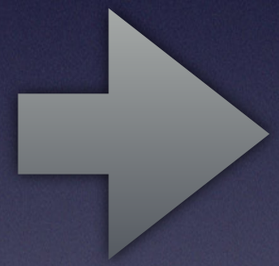
微博服务发现之
多机房高可用

新浪微博 姚四芳
@icycrystal4

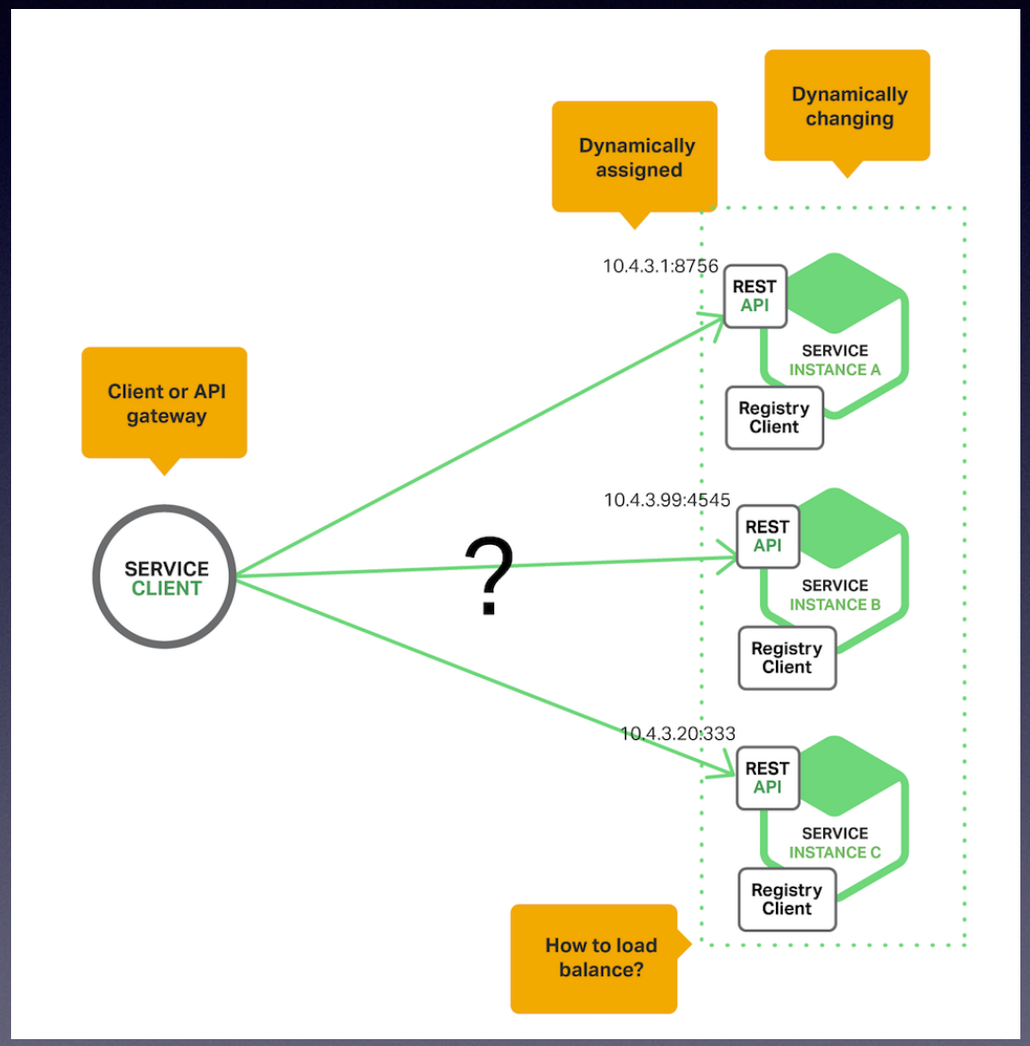
服务化的优势与挑战

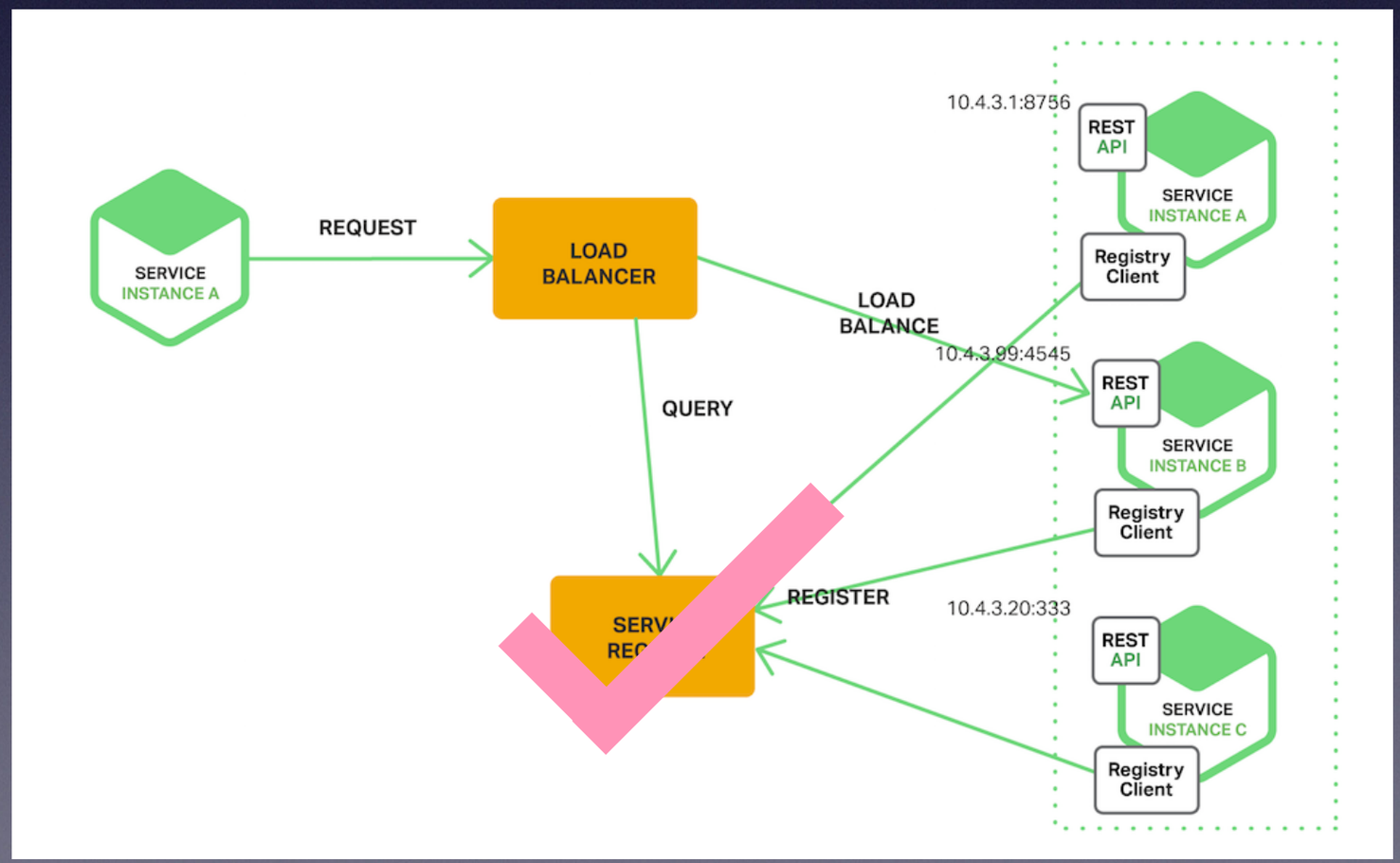
- 化繁为简、分而治之
- 最合适的技术栈
- 独立部署
- 独立扩缩容



- 服务的发现
- 状态管理
- 路由与流量控制
- RPC

注册中心机制

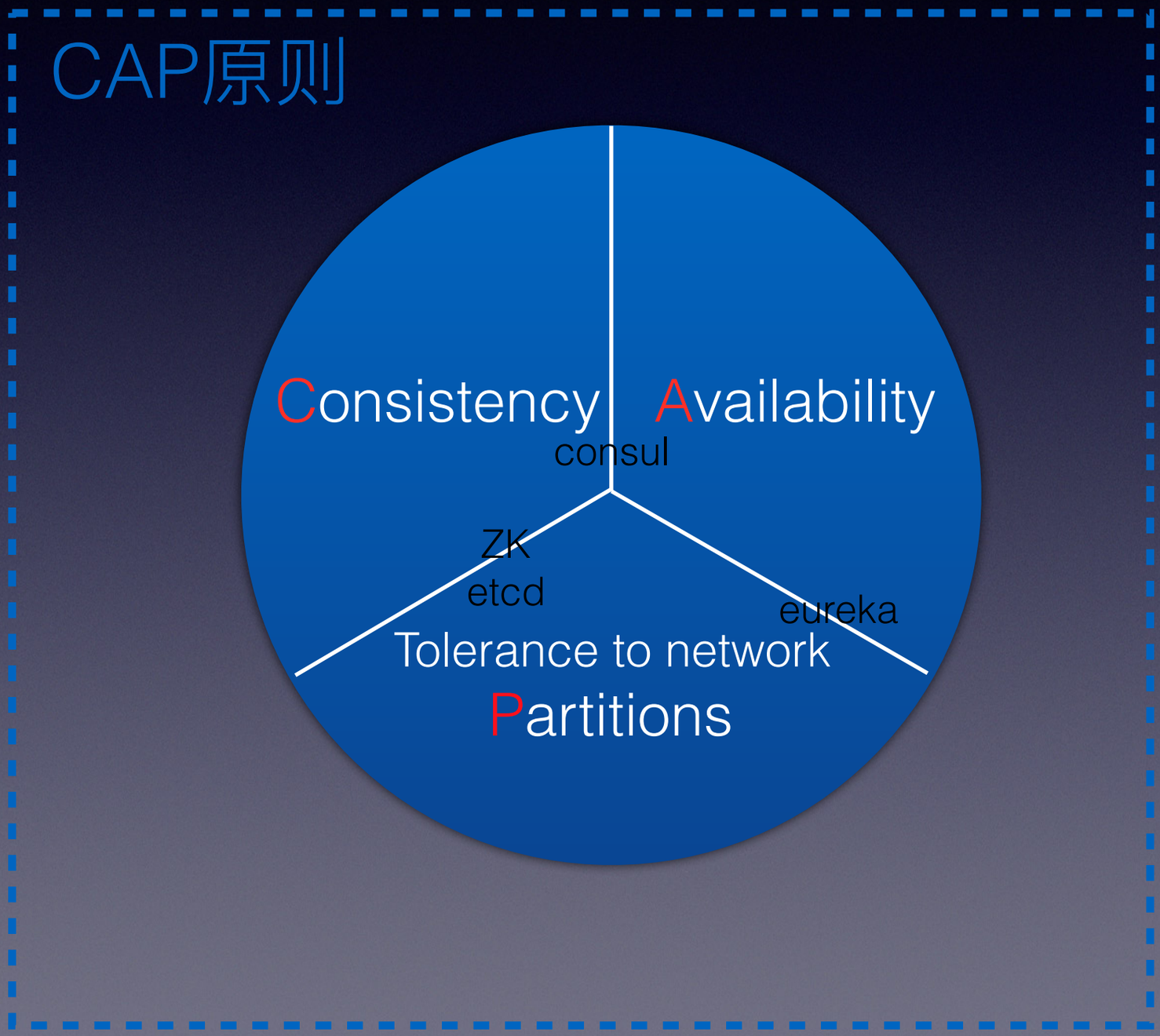




* 图片来源: nginx.com

服务发现生态

- zk&consul&etcd
  - 强调C舍弃A/P(脑裂)
  - 通知风暴
  - 选举: 多数原则
  - 缺乏生命周期管理
- eureka
  - EIP绑定(AWS)
  - 不支持推送、延时高
  - 重client(只支持java)



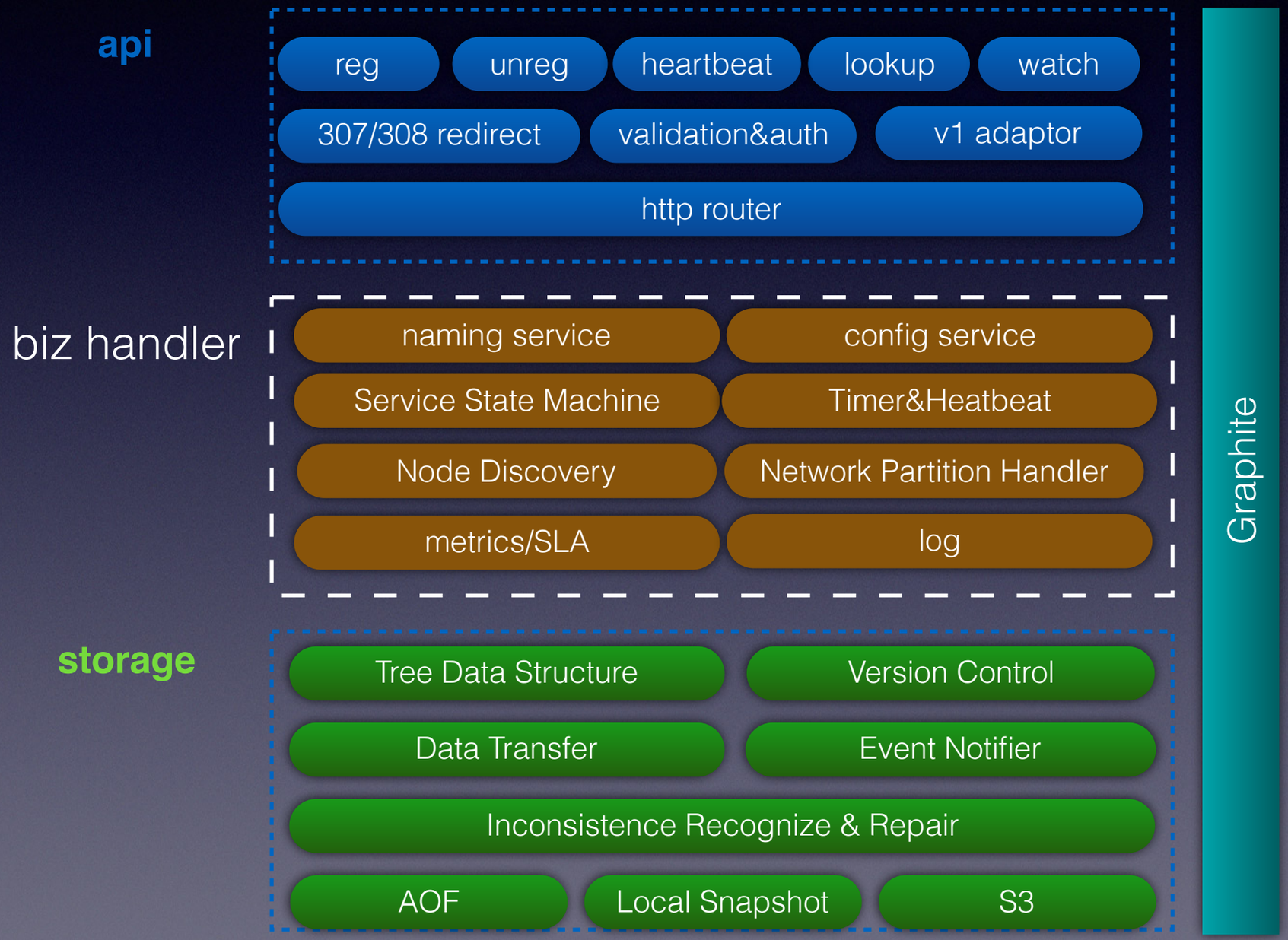
微博服务现状

- 可用性高于一切(A)
  - 日常晚高峰扩容
  - 热点临时扩容
- 网络抖动分区是常态(P)
  - 硬件故障
  - 带宽跑满

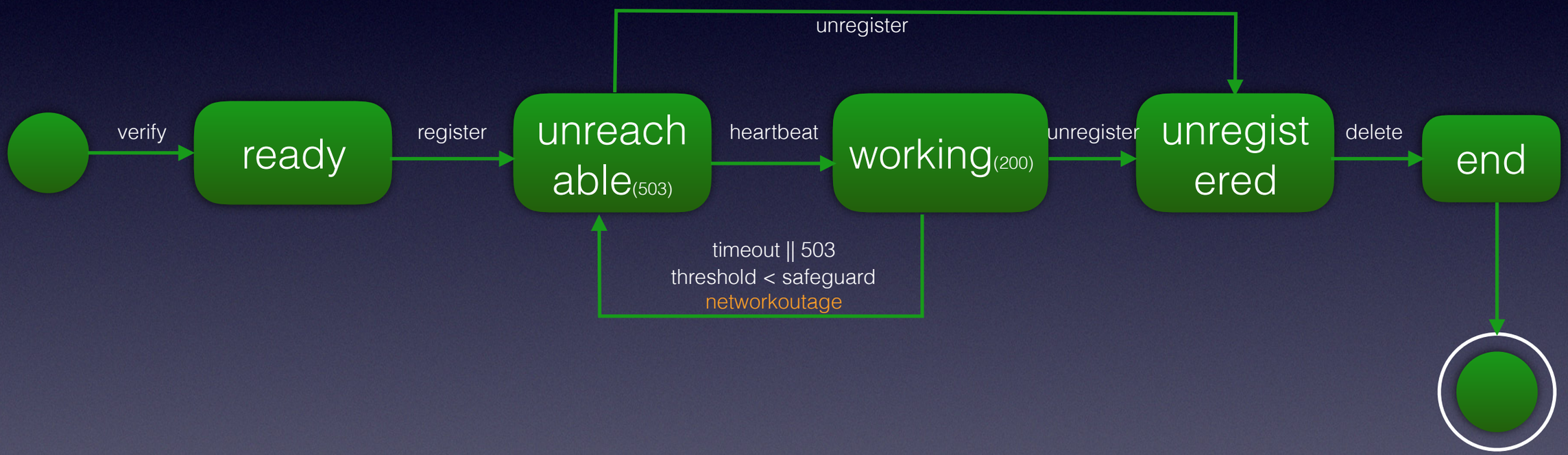
乐观但合理的假设

- 节点变更不频繁
- 并发不是大问题
- 脏数据好过没有数据
- 操作幂等性
- 写比读重要

vintage架构设计



服务生命周期



vintagecli register --service=friendship --ns=aliyun ip:port

working状态保护

- 200只需一个heartbeat
- timeout是503必要条件(第I II类错误)
- 60%的安全阀值(雪崩): working num/total num
- 网络分区后禁止503: frozen
  - 服务不可用: 交给client重试

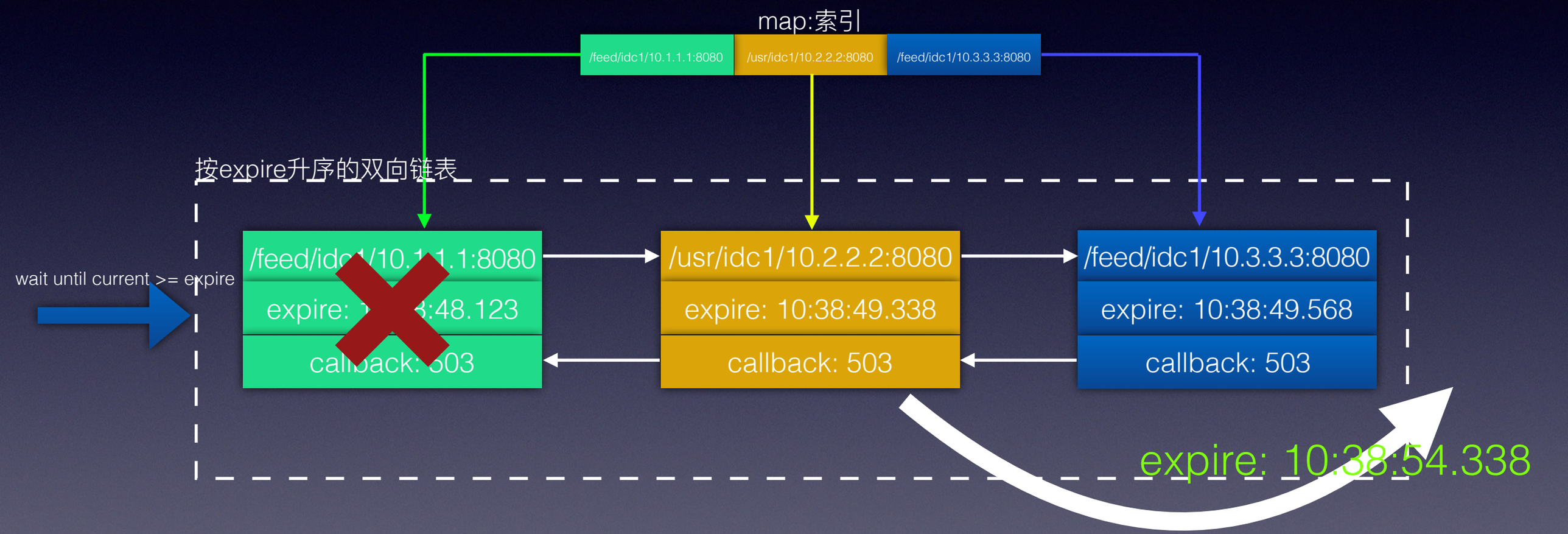
200—>503 变更机制

- 定时器驱动
- 支持10w级别定时器
- 精度要求在ms级
- 频繁刷新expire
- 服务timeout通常相同

数据结构	Update	Trigger
链表	O(n)	O(n)
最小堆	O(logn)	O(1)
时间轮	O(n)*	O(n)*
?	O(1)	O(1)

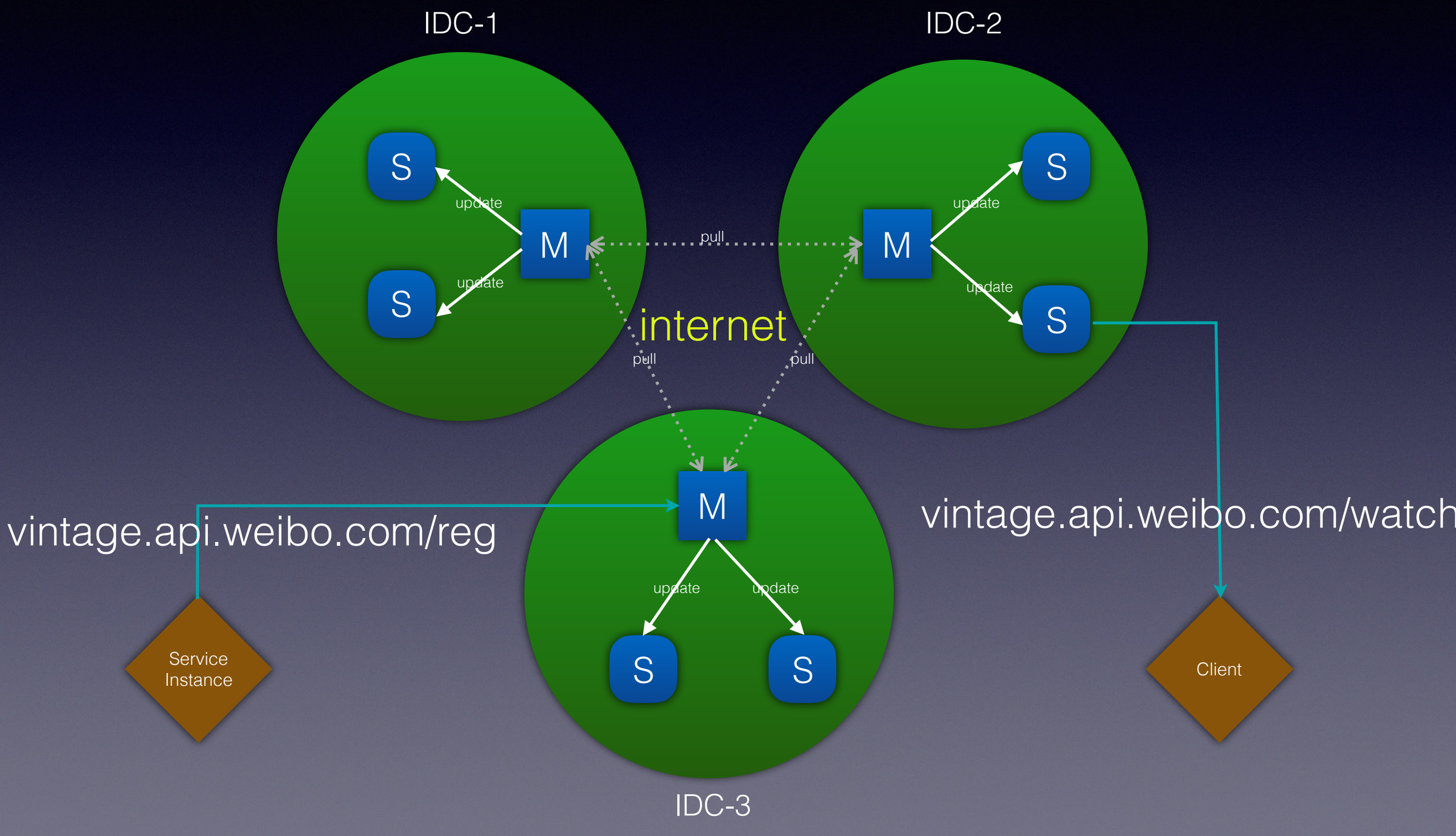
注:时间轮的时间复杂度受很多因素影响。在短时间内大量注册定时容易导致定时器集中在一个slot中。而vintage中的定时器注册通常是在启动时短时间内完成。

基于list+map的高性能定时器



- trigger: O(1)
- update: O(1)

vintage分布式部署架构



节点发现策略

- 本地IDC发现: dns
- master/slave: 307/308
    - master: 写请求
    - slave: 读请求
- master选举:
    - 基于raft的多master策略
    - 分区后每个分区独立选举
    - 不需要多数支持
- 节点&IDC发现: gossip

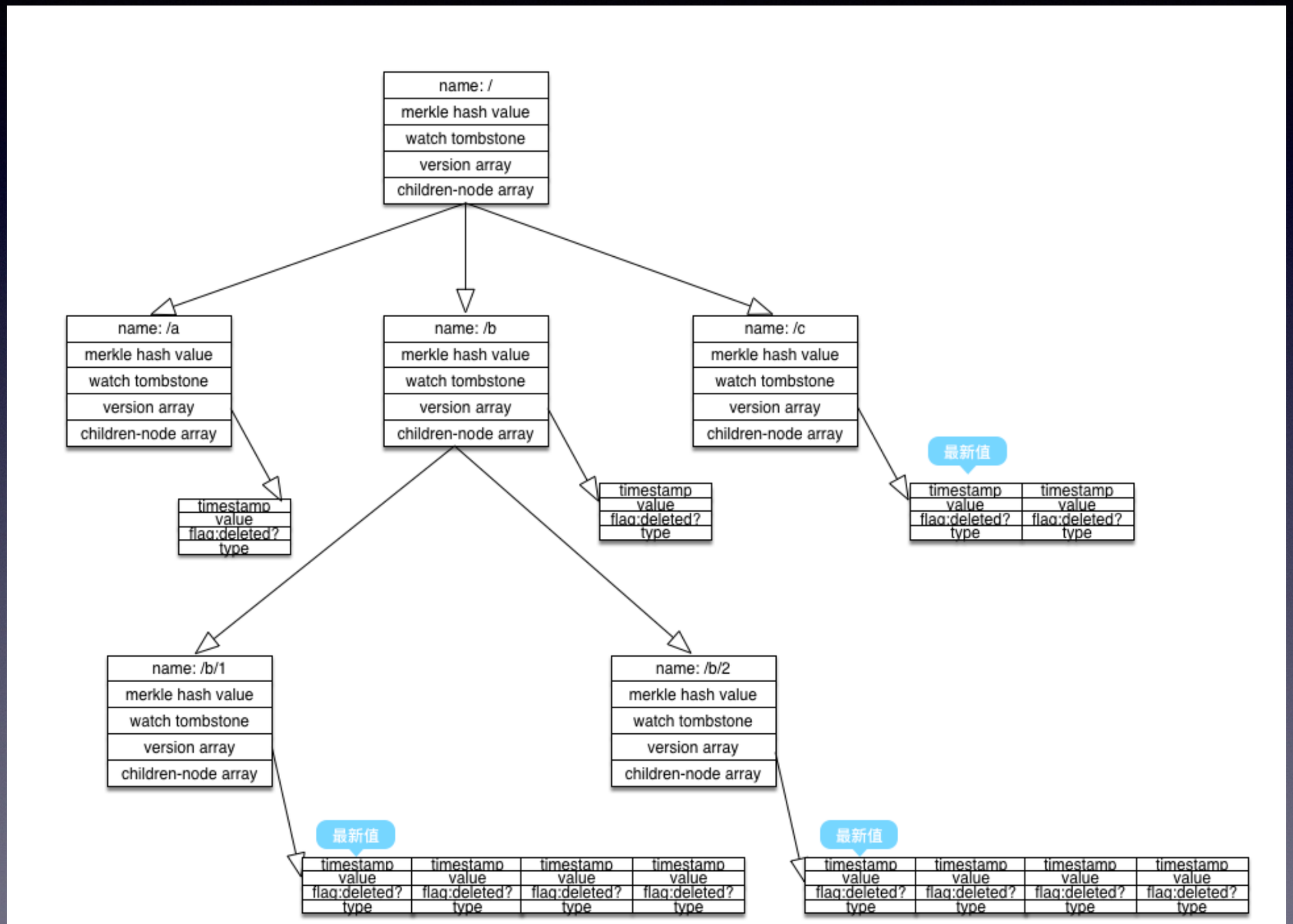
状态与数据存储设计

- 分布式数据同步: 简化上层设计
- 多版本支持回溯
- 数据一致性校验
- 数据自修复策略
- 节点变更通知
- 容灾与快速恢复

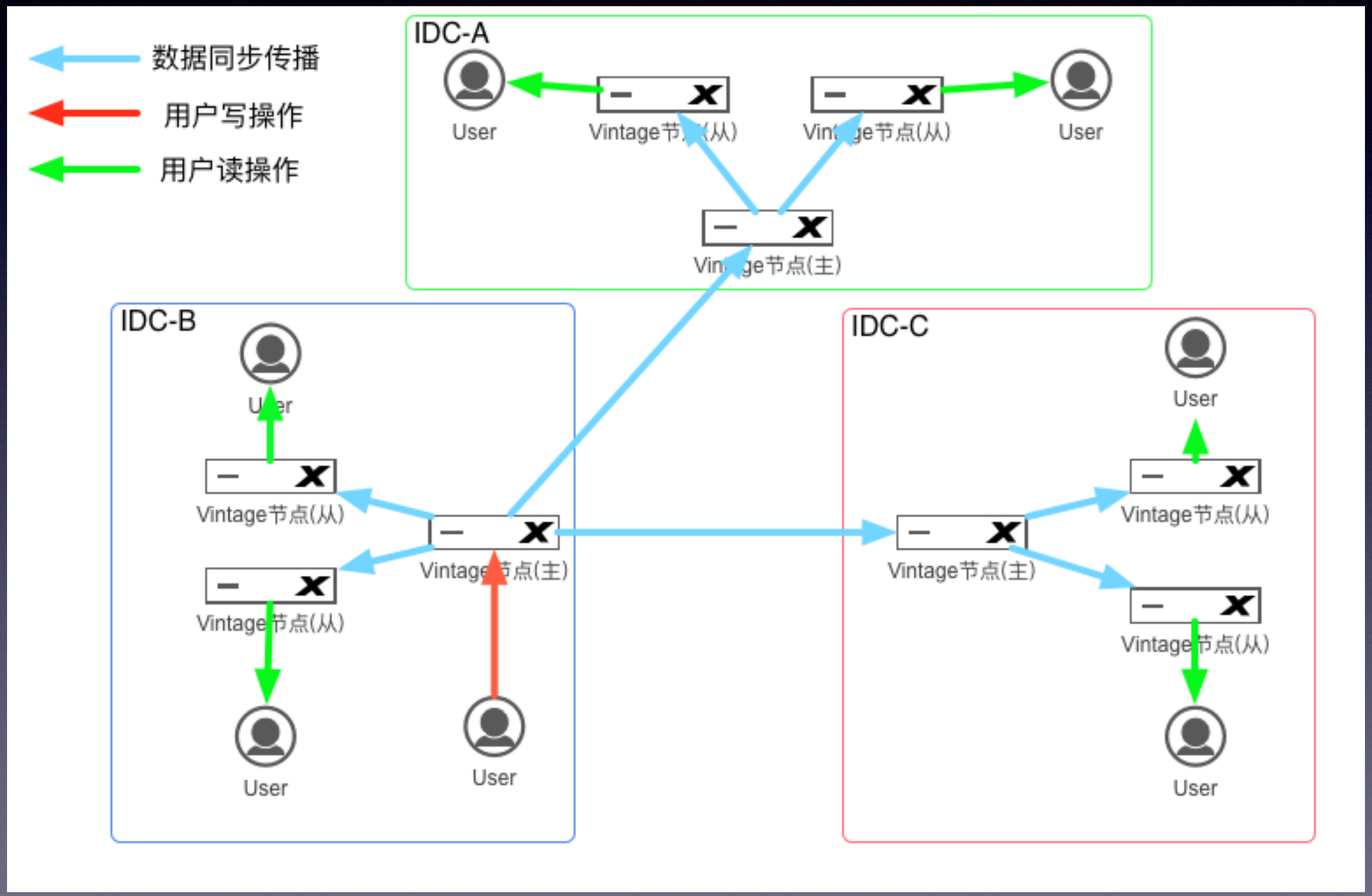
存储选型设计

- redis、mysql...
  - 分区后不可用
  - 不支持多IDC
  - 引入第三方组件
  - 不支持回调
  - 计算离存储太远

数据存储结构设计


name: /
merkle hash value
watch tombstone
version array
children-node array
name: /a
merkle hash value
watch tombstone
version array
children-node array
name: /b
merkle hash value
watch tombstone
version array
children-node array
name: /c
merkle hash value
watch tombstone
version array
children-node array
timestamp
value
flag:deleted?
type
timestamp
value
flag:deleted?
type
最新值
timestamp
value
flag:deleted?
type
timestamp
value
flag:deleted?
type
name: /b/1
merkle hash value
watch tombstone
version array
children-node array
name: /b/2
merkle hash value
watch tombstone
version array
children-node array
最新值
timestamp
value
flag:deleted?
type
最新值
timestamp
value
flag:deleted?
type

数据传输机制


数据同步传播
用户写操作
用户读操作
IDC-A
User
Vintage节点(从)
Vintage节点(从)
User
Vintage节点(主)
IDC-B
User
Vintage节点(从)
Vintage节点(从)
Vintage节点(主)
User
User
IDC-C
User
Vintage节点(从)
Vintage节点(主)
Vintage节点(从)
User

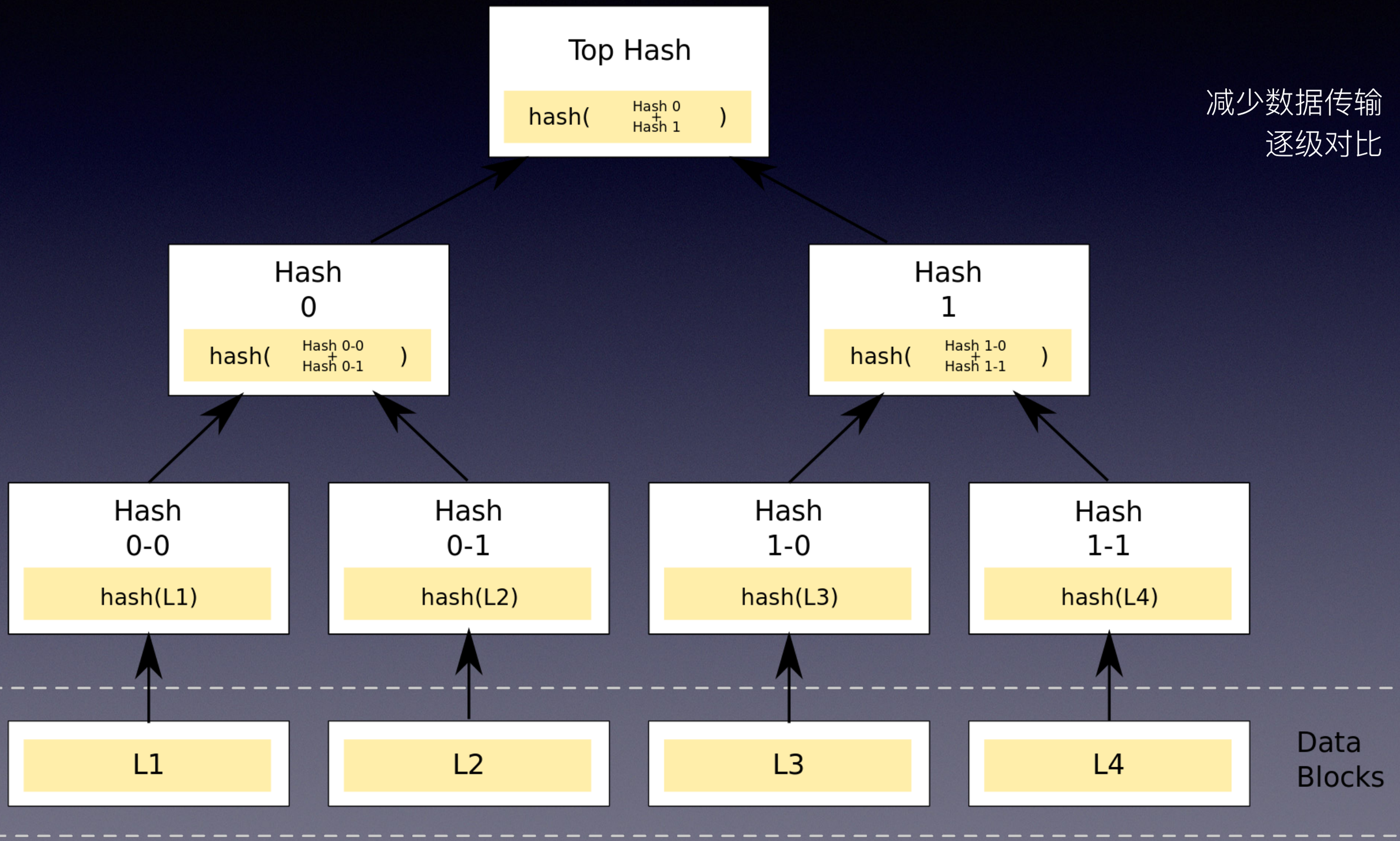
状态数据传输机制

- 数据流向: M->S; M<->M
- 滚动的WAL event pool:
  - sequence
  - offset
- 增量同步: offset未溢出
- 全量同步: offset溢出
- event tag: 解决传输loop问题

client获取事件通知

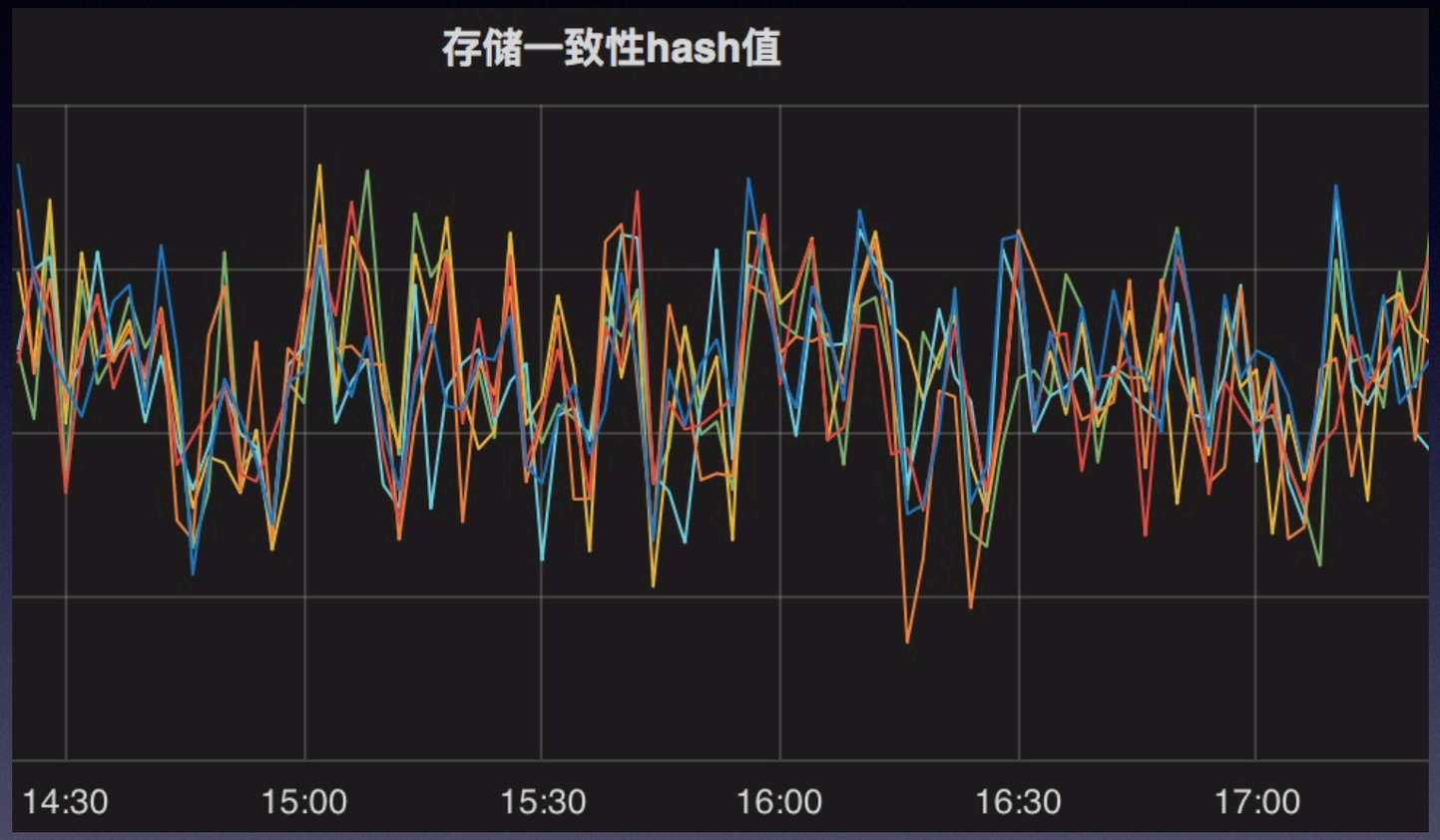
- watch hub 递归watch
- 所有事件进入watch hub
- client注册到watch hub
- 指定index开始watch
- index溢出时lookup

merkle hash tree数据对比



减少数据传输
逐级对比

动态识别数据不一致



- 杰卡德相似性

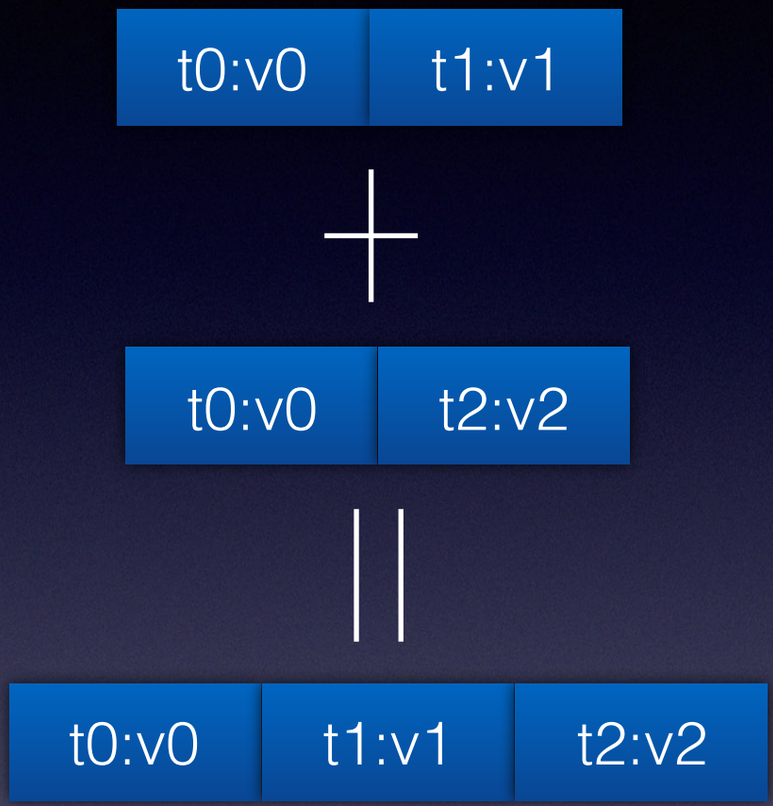
$$J(A,B) = \frac{|A \cap B|}{|A \cup B|}$$

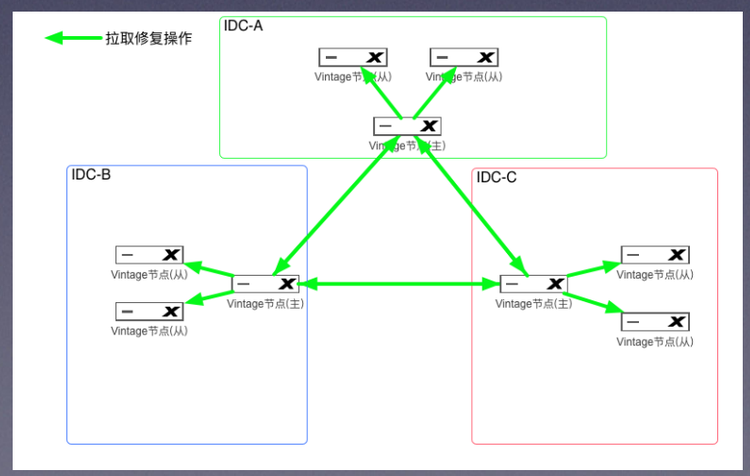
- 余弦距离相似性

$$\cos A = \frac{<b,c>}{|b\|c|}, \quad sim(X,Y) = cos\theta = \frac{\vec{x} \cdot \vec{y}}{\|x\| \cdot \|y\|}$$

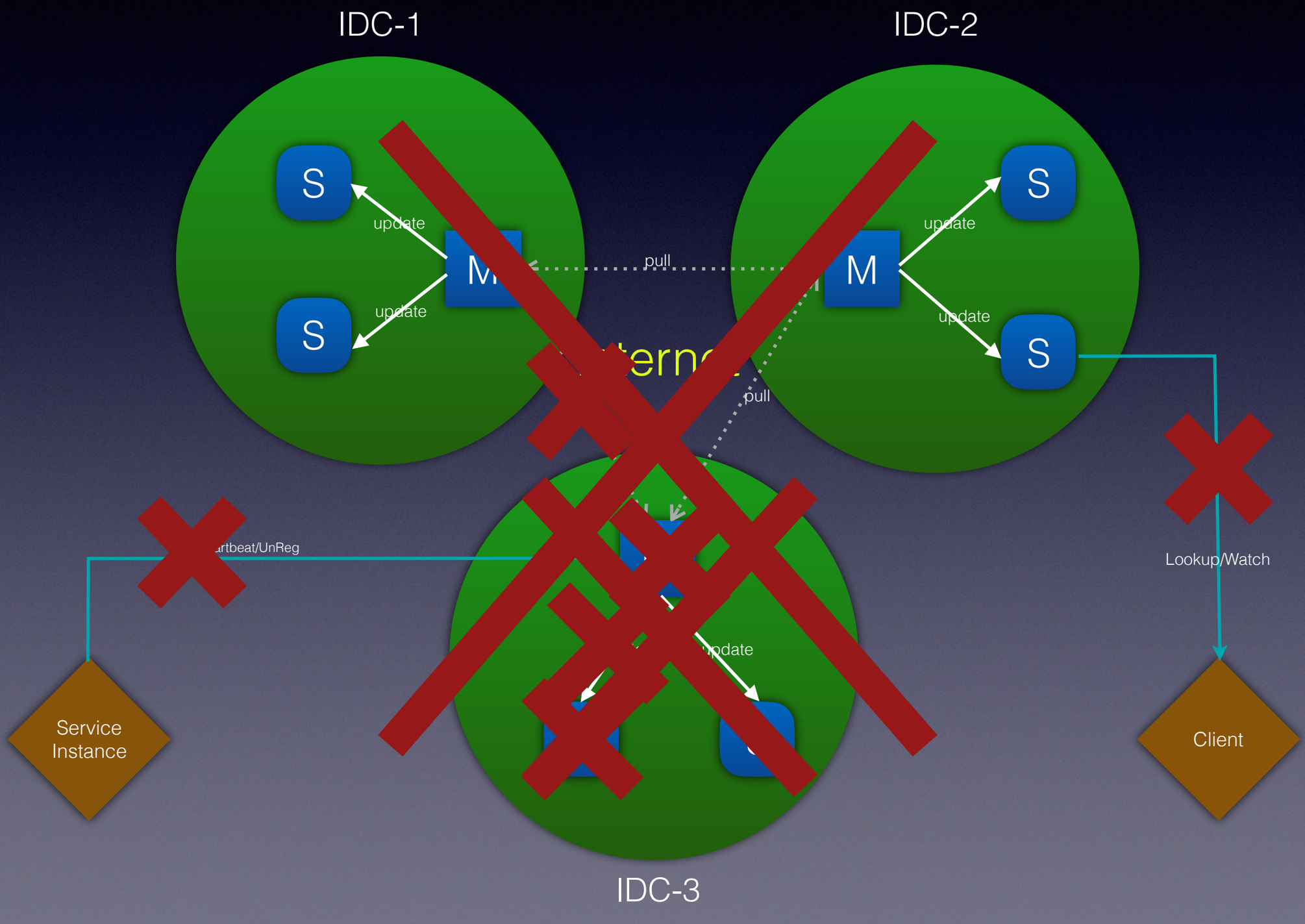
基于全局index进行数据自修复

- 独立分布式index生成器
- 基于机器时钟
  - 基于递增的逻辑机器时钟
    - 避免时钟调整时震荡
    - 允许短时间内回溯(3s)





vintage容错处理

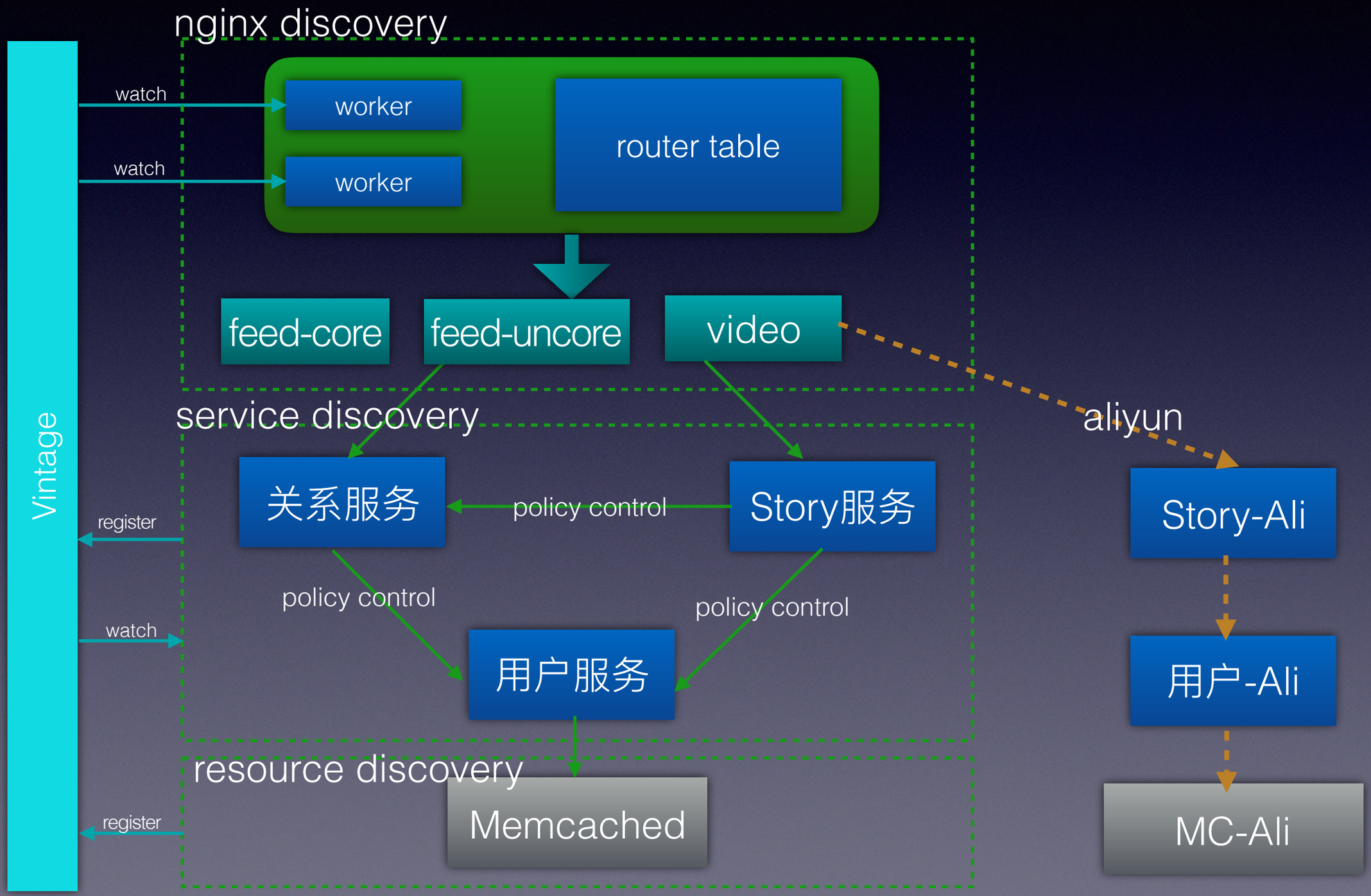


99.9999

结合motan支撑微博服务化

- 面临的挑战
  - 业务扩容(Feed、视频、直播)常态化、热点随机化、aliyun动态扩容
  - 服务间依赖关系复杂
  - 扩容、流量迁移时效性

服务发现与流量控制


nginx discovery
watch
watch
worker
worker
router table
feed-core
feed-uncore
video
service discovery
aliyun
Vintage
关系服务
policy control
Story服务
Story-Ali
register
policy control
policy control
watch
用户服务
用户-Ali
resource discovery
Memcached
MC-Ali
register

效果与收益

- 成本: 业务服务器的buffer从40%—>10%
- 服务扩容流量迁移优化至秒级(之前为天级)
- 6个9的可用性(机器宕机、网络QoS、人为失误)
- 动态schedule进行多IDC部署

Thank You