联盟链动态缩容方案

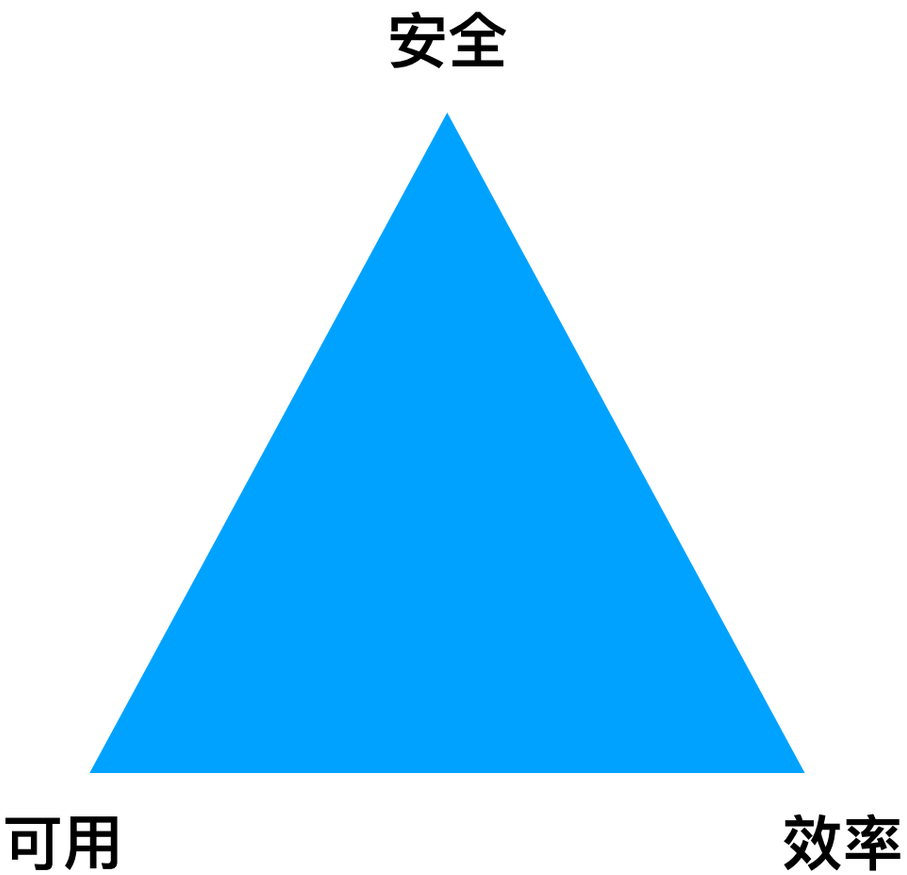
郭世清 2018.04

背景

- 世界状态快速膨胀
- 区块数据快速增长

问题

- 节点存储压力
- 网络冗余成本
- ”笨“ ”重“



目标

- 安全性、可用性不下降
- 运维友好
- 业务友好
- 更轻、更快

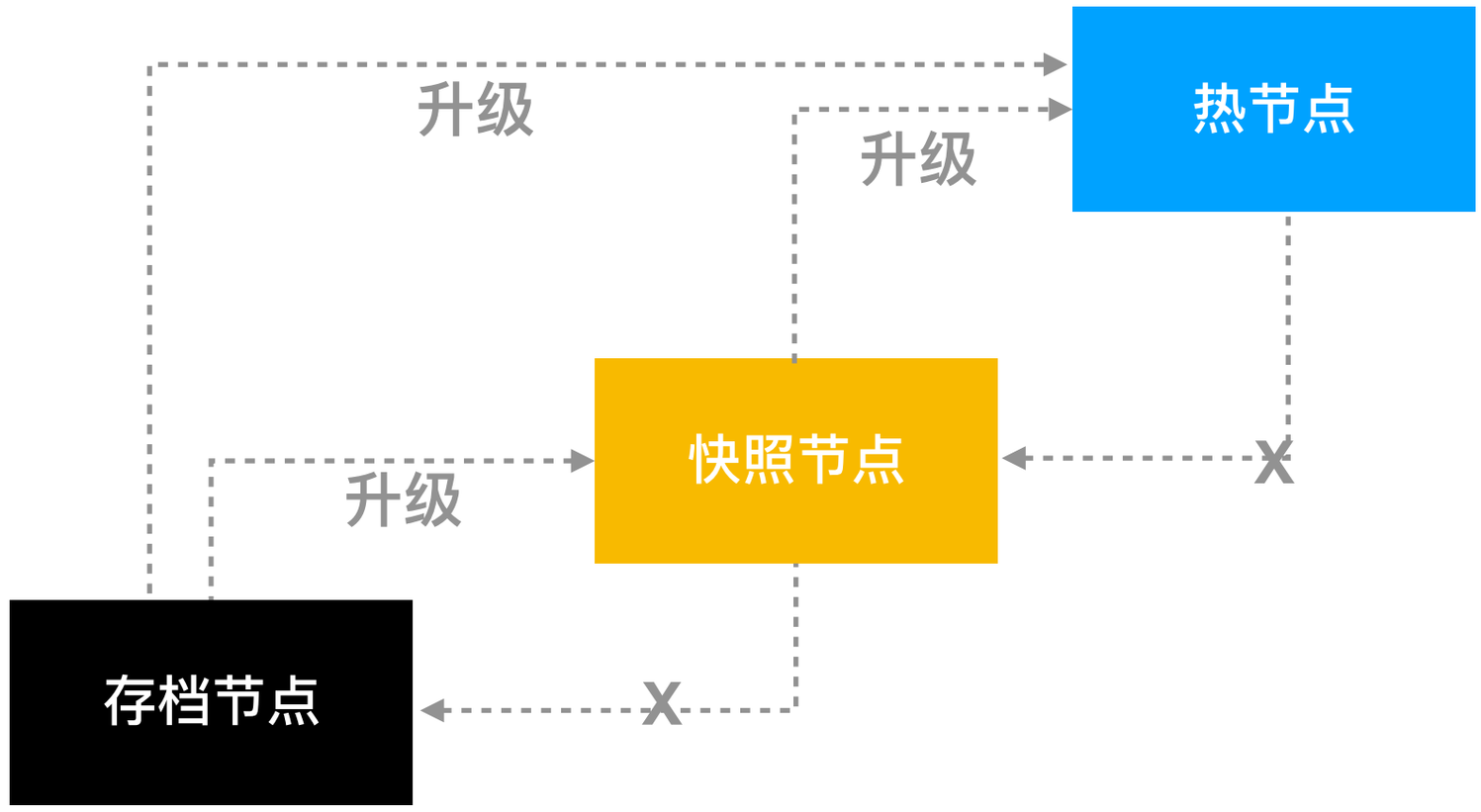
节点定义

- 存档节点
- 快照节点
- 热节点
- 轻节点

功能集\节点类型	存档节点	快照节点	热节点	轻节点
交易广播	Y	Y	Y	Y
交易执行	Y	Y	Y	
参与共识	Y	Y	Y	
所有区块头	Y	Y	Y	Y
最新区块体及状态数据存储	Y	Y	Y	
最新区块体及状态数据查询	Y	Y	Y	
历史区块体及状态数据存储	Y			
历史区块体及状态数据查询	Y	Y		

节点关系

- 存档节点(必选)
- 快照节点
- 热节点



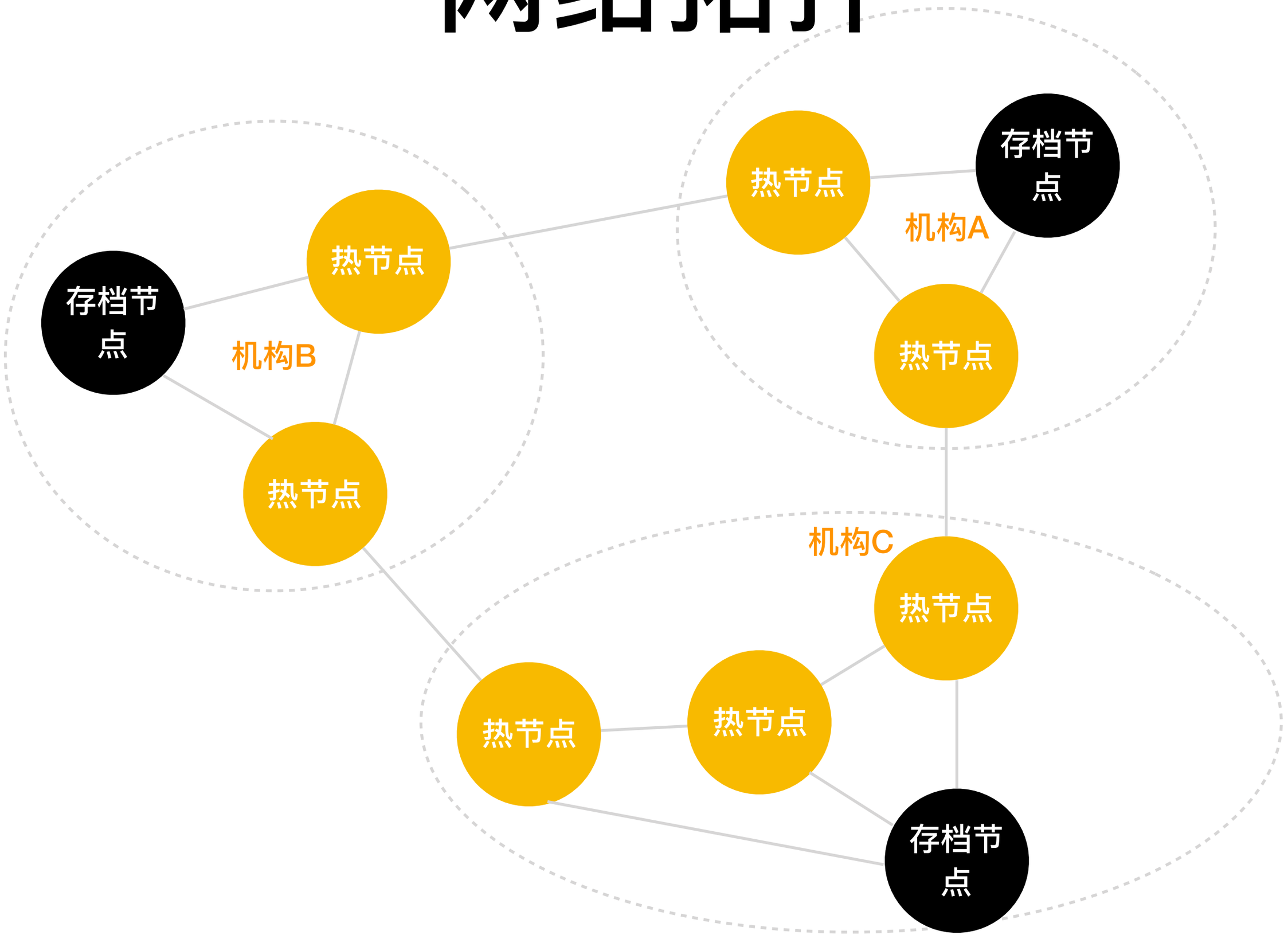
方案一

- 方案：世界状态导出新创世块，建新链
- 优点：实现简单、最大限度裁剪、无历史包袱
- 缺点：
  - 历史状态缺失
  - 运维不友好
  - 业务不友好
  - 可用率低

方案二

- 方案：热节点与存档节点
- 优点：业务友好、运维友好、历史状态友好、效果显著
- 缺点：
    - 协议复杂
    - 热节点查询不安全
    - 效果因业务不同(对纯增量业务无效)

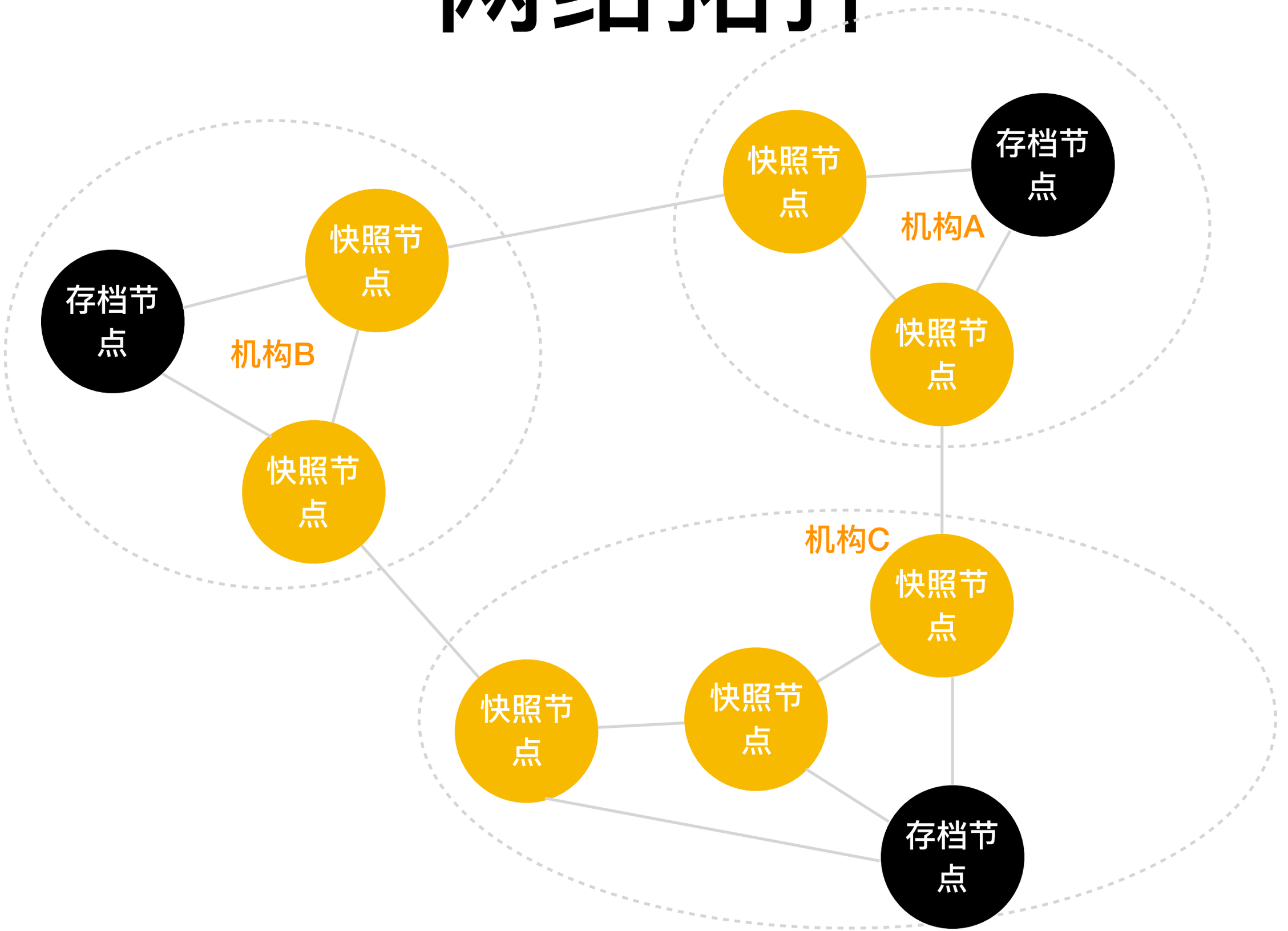
网络拓扑



方案三

- 方案：快照节点与存档节点
- 优点：业务友好、运维友好、历史状态友好、高安全性
- 缺点：
  - 实现复杂
  - 协议复杂
  - 效果因业务不同(对纯增量业务无效)

网络拓扑

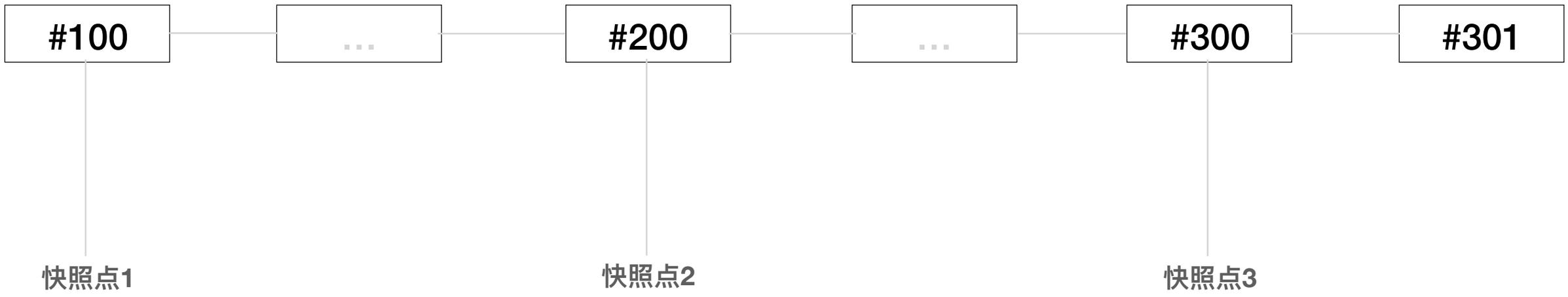


方案对比

	方案二 热节点与存档节点	方案三 快照节点与存档节点
安全性	中	高
可用性	高	高
效率	高	中
缩容比例	高	中
实现复杂度	中	高

实现架构

动态快照



- 设变更空间C(i)：块i的世界状态的变更合集{k,v}
- 则在快照点i时，执行：
- 1）C=C(i-99)+…+C(i)
- 2）对C稳定排序
- 3）将C中重复k对应的数据删除
- 4）将B(i-99),B(i-99)…B(i-1) 数据清理

- 举例：

C(98)={{k1,v0},{k3,v5}}

C(99)={{k1,v1},{k2,v2}}

C(100)={{k1,v3},{k3,v4}}

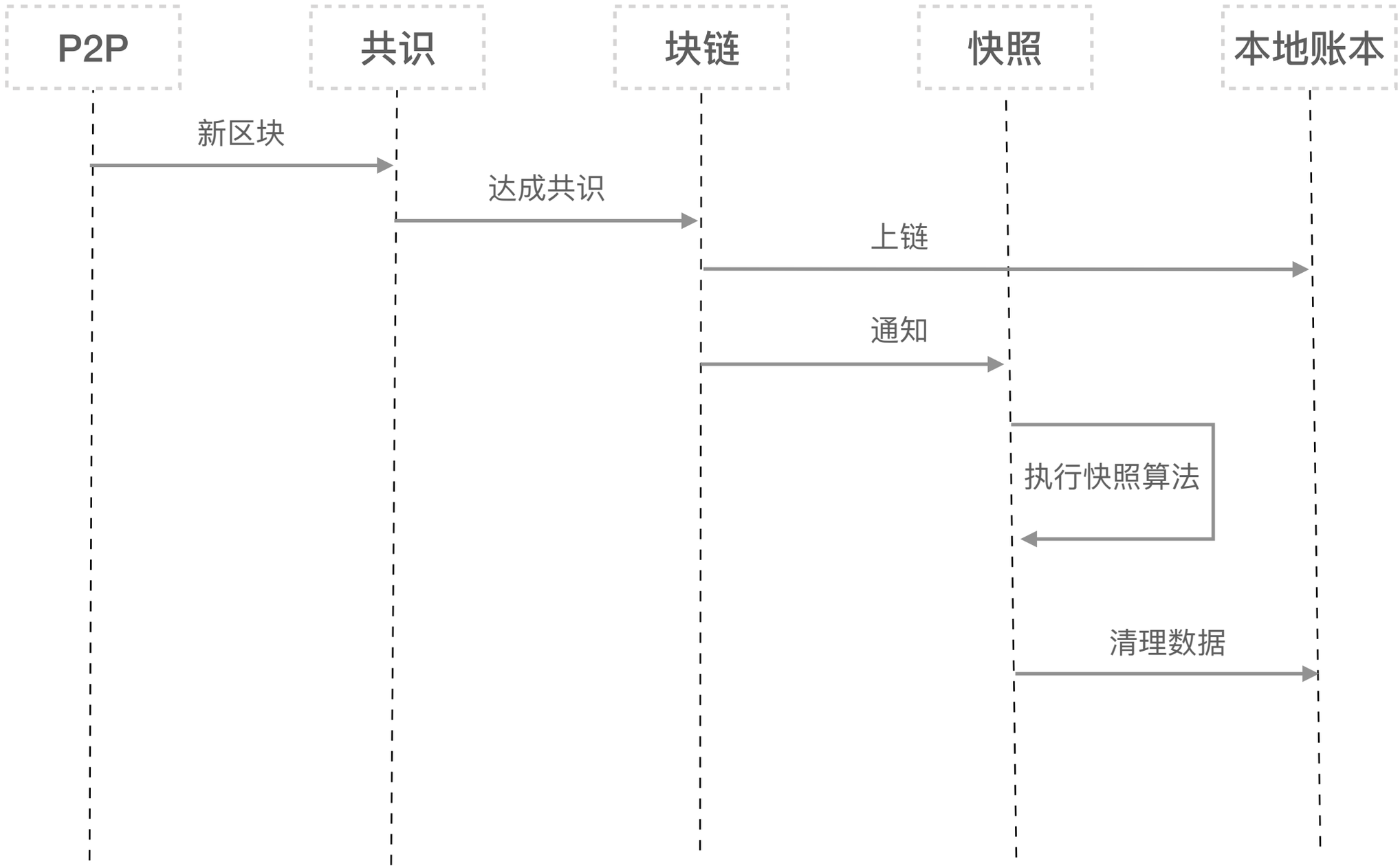
假设100为快照点，则

1）C={{k1,v0},{k3,v5},{k1,v1},{k2,v2},{k1,v3}，{k3,v4}}

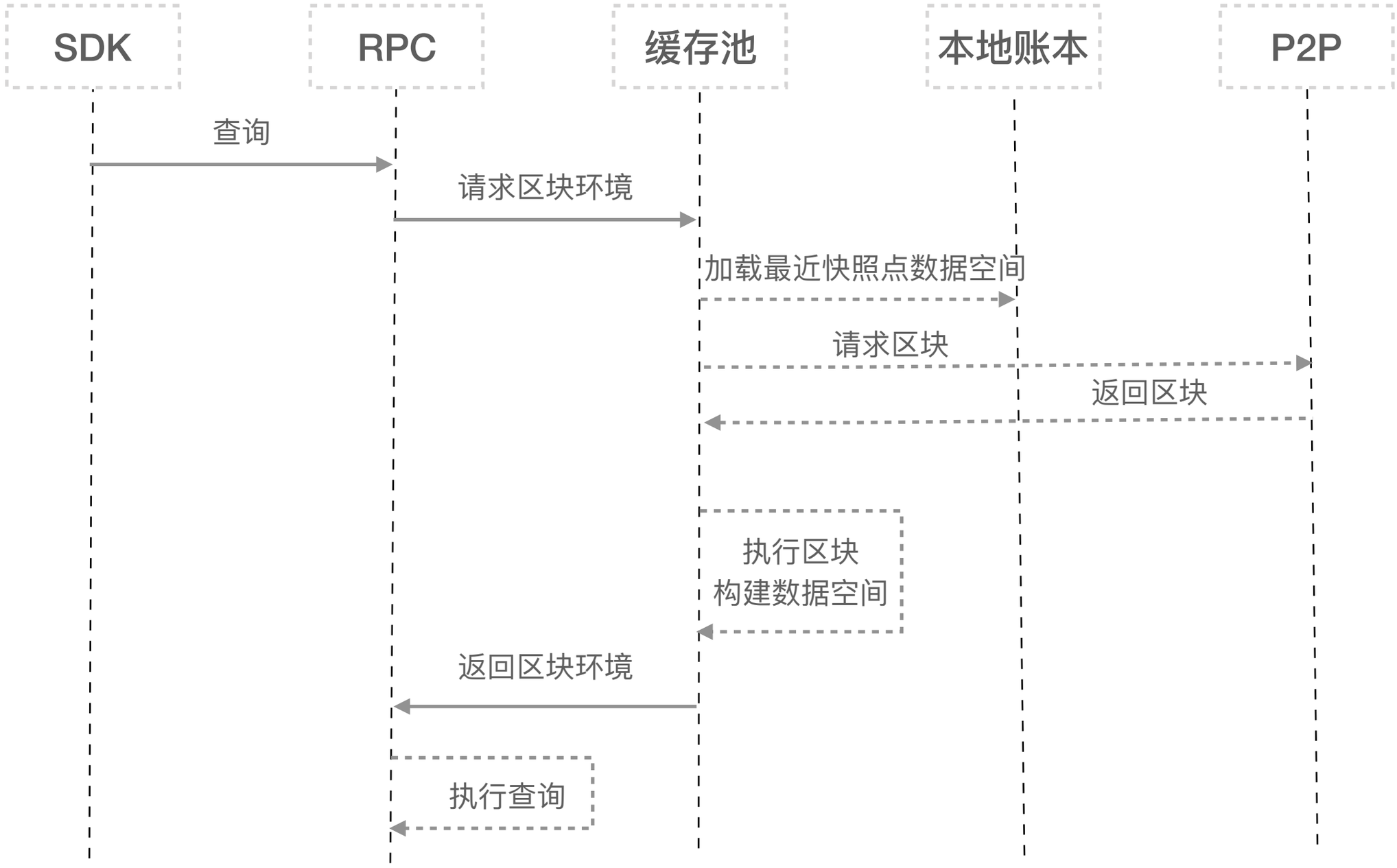
2）C={{k1,v0},{k1,v1},{k1,v3},{k2,v2},{k3,v5},{k3,v4}}

3）将{k1,v0},{k1,v1},{k3,v5}清理

关键流程-快照



关键流程-查询


SDK
RPC
缓存池
本地账本
P2P
查询
请求区块环境
加载最近快照点数据空间
请求区块
返回区块
执行区块
构建数据空间
返回区块环境
执行查询

同步协议

- 面向区块（简单,CPU换网络）

- 面向快照状态（复杂，网络换CPU）

节点配置

- 节点类型
- 节点快照周期
- 节点缓存池大小

谢谢

愿区块链世界里没有”胖子“

— *Vitalik*和中本聪