

乾坤

领域模型+内存计算+微服务的协奏曲

主讲人：深蓝·陈秋余



个人简介



陈秋余，深蓝

原卷皮网架构研发总监，现支付宝高级专家。

资深架构师&技术管理专家，十多年传统企业应用与互联网从业经验。

曾主导**中国移动集团**、**南方电网**IT战略架构规划；

负责**卷皮网**从0到1的技术架构落地和团队搭建，支撑数亿PV，百万级日订单。

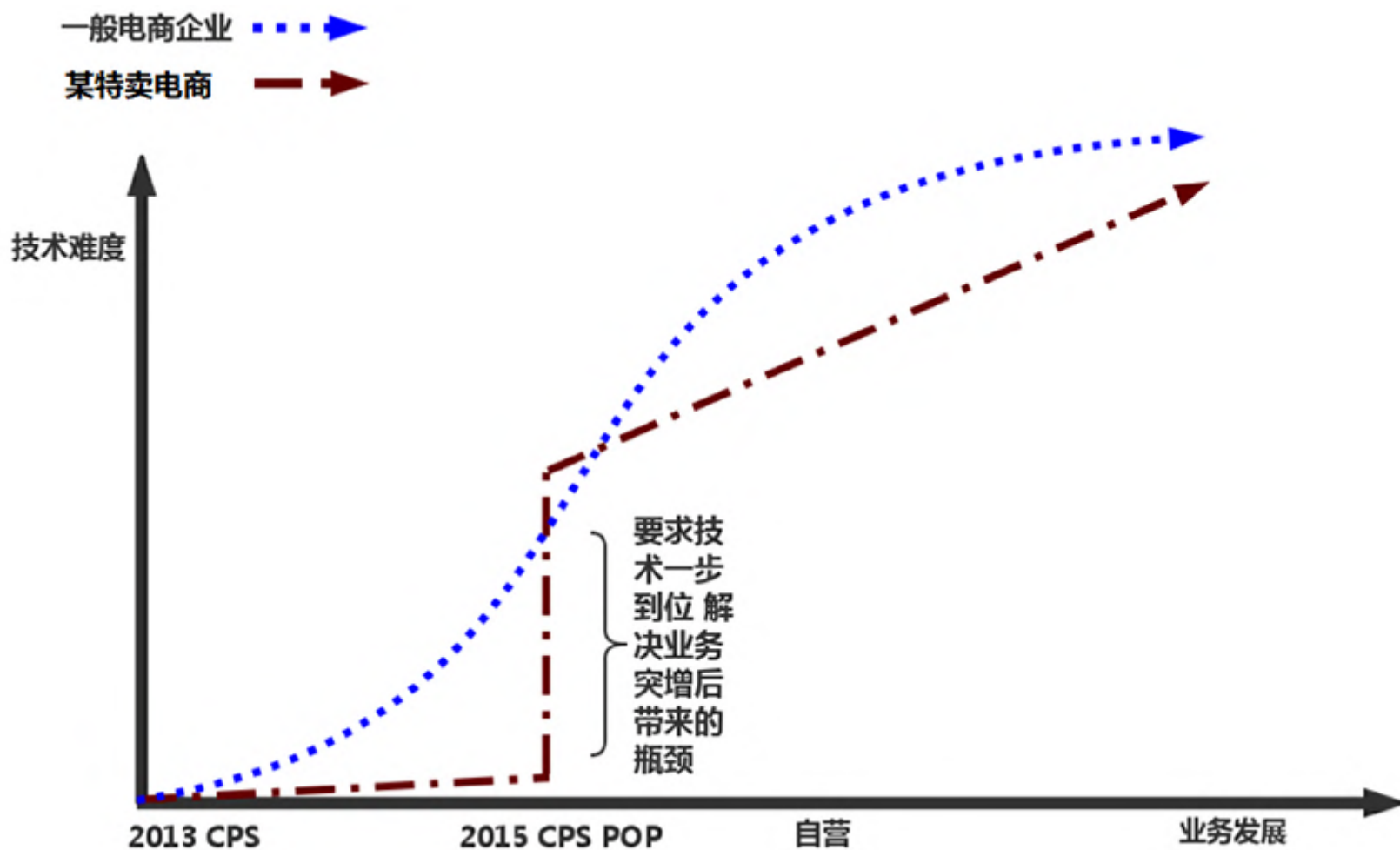
经历过多种类型，不同阶段的技术团队，注重团队组织与技术架构融合互促，对敏捷有深刻理解。

在**TOGAF**，**SOA**以及**微服务**领域有相当建树。

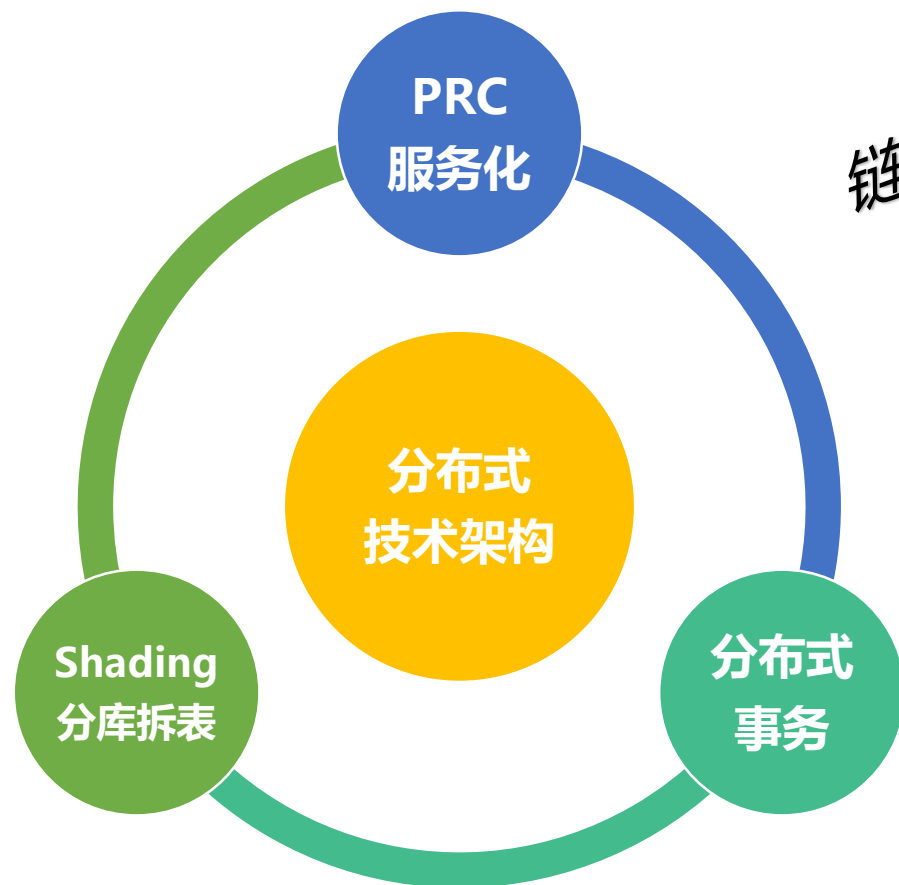
什么是乾坤？

- 特卖类电商的核心架构体系
- 特卖类电商场景和技术特点？
 - 早10点，晚8点特惠商品上架，类似全场秒杀场景
 - 加购物袋同时必须实时扣库存，超卖容忍度极低
- **技术特征&指标**
 - **非主流**分布式交易系统架构：非RPC体系，类事件驱动架构；
 - 超过**150万**日订单量，核心接口TPS峰值**6万+**；
 - 仅用**20台**应用服务器、**30多台**数据服务器达成；
 - *压测性能：单应用节点TPS轻松过10万+；

乾坤的诞生



当前主流技术架构三板斧及主要困境



链路依赖

线程&连接阻塞

分布式事务性能堪忧

服务的边界和粒度

微服务 VS 跨域查询

当前主流技术架构三板斧及主要困境



我们有:

- ✓ 2PC->3PC->TCC
- ✓ 数据库中间件->Ocean base
- ✓ Dubbo->熔断器
- ✓

与其一直苦逼补天，不如换个思路吧



乾坤的架构哲学

1. 解决高并发争抢问题的最好办法就是完全避免争抢的发生；
2. 解决分布式事务问题的最佳选择就是根本不用分布式事务；

从1000万用户并发修改用户资料的假设场景开始

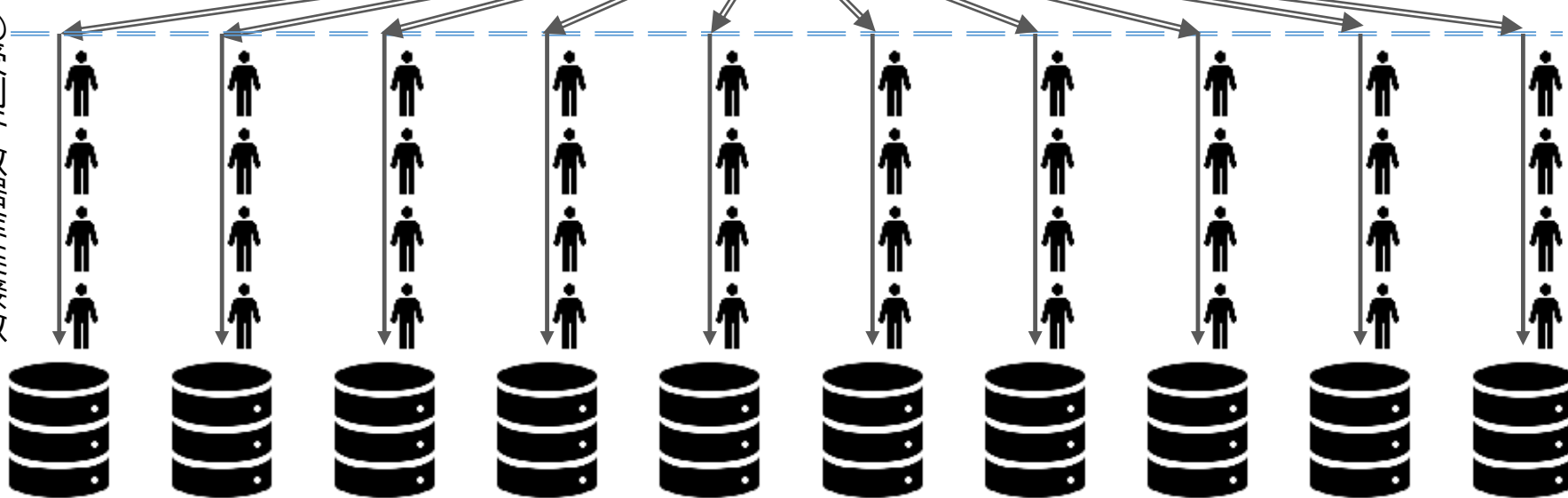
主流技术架构方案示意

10,000,000+用户



①每个库100万用户请求

②受制于数据库连接数



③每次修改操作耗时200毫秒

高并发 ≠ 高并发争抢，两张图你就懂了



图中有4个并发

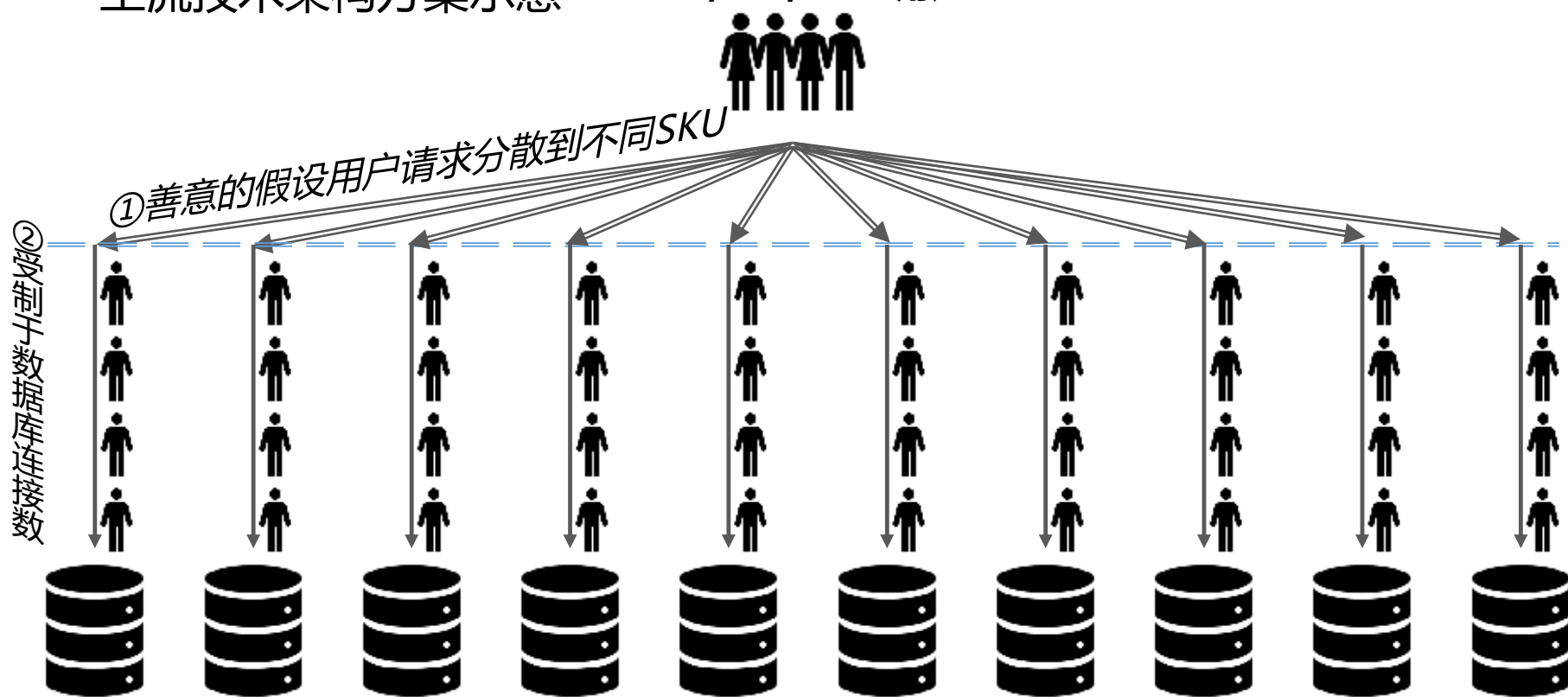


这是7个并发争抢1个
感受一下.....

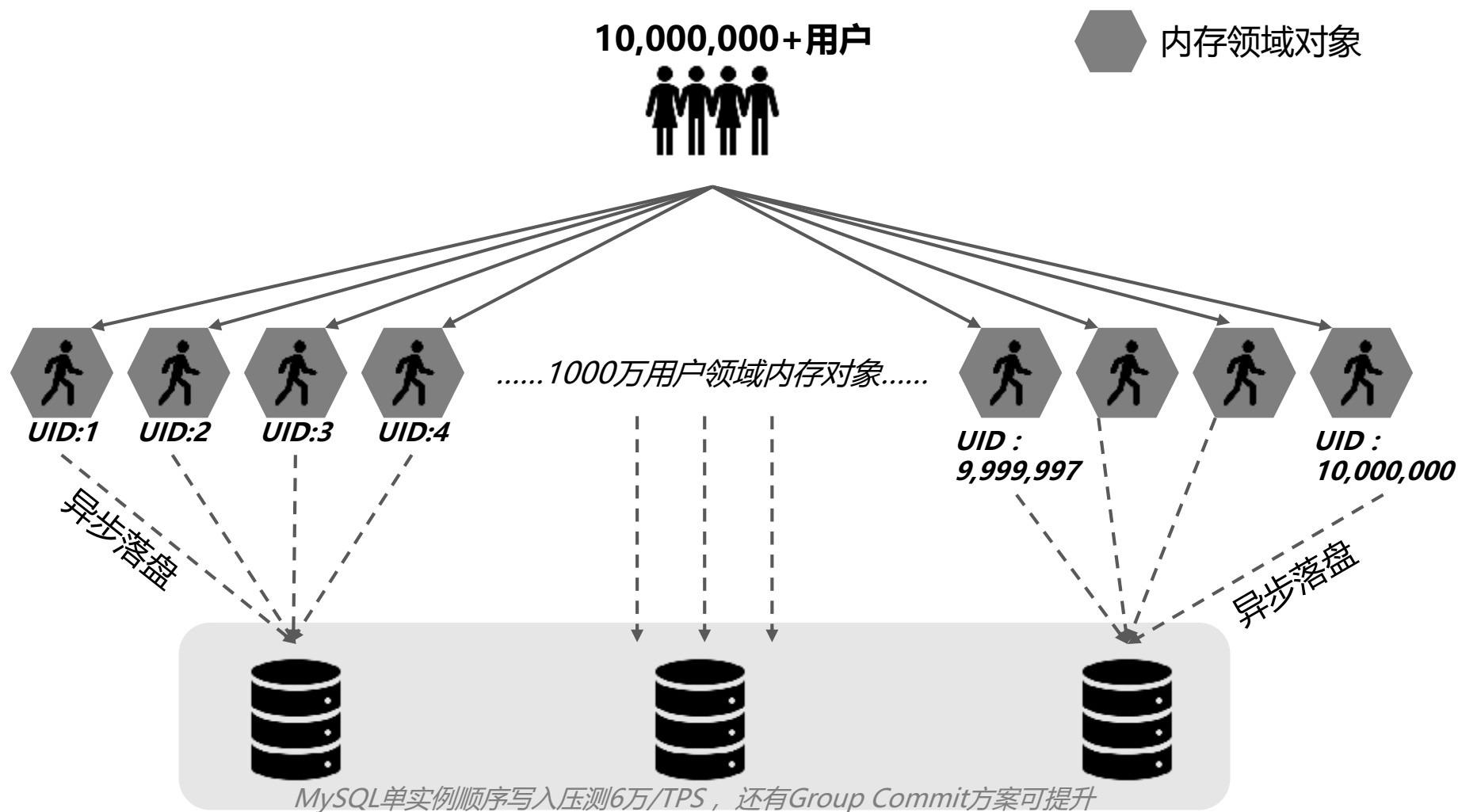
改成1000万用户疯抢200万SKU的商品会咋样？

主流技术架构方案示意

10,000,000+用户



乾坤又是如何解决高并发的性能问题



乾坤又是如何解决高并发争抢的性能问题

领域对象上，

单次请求处理时间：**微秒级**

10,000,000+用户



内存领域对象

无竞争争抢

领域内单线程顺序执行



内存领域对象 VS. 分布式缓存



从技术实现角度看：

1. 额外的性能开支；
2. Redis单线程模式的优势与劣势都很明显；
3. 数据结构的灵活性；

从架构和设计角度看：

领域模型和表模式的区别

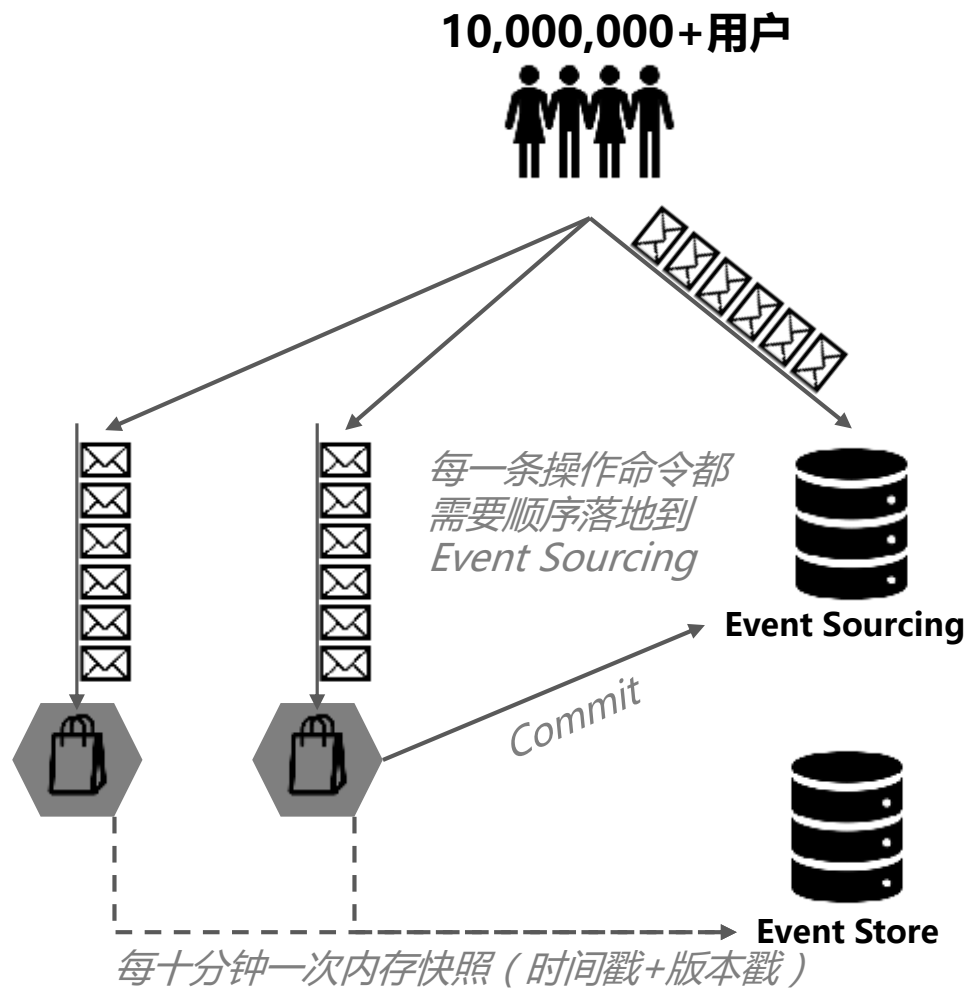
- 这个.....
- 那个.....还有.....

你单独请我喝咖啡，我们再秉烛夜谈吧.....

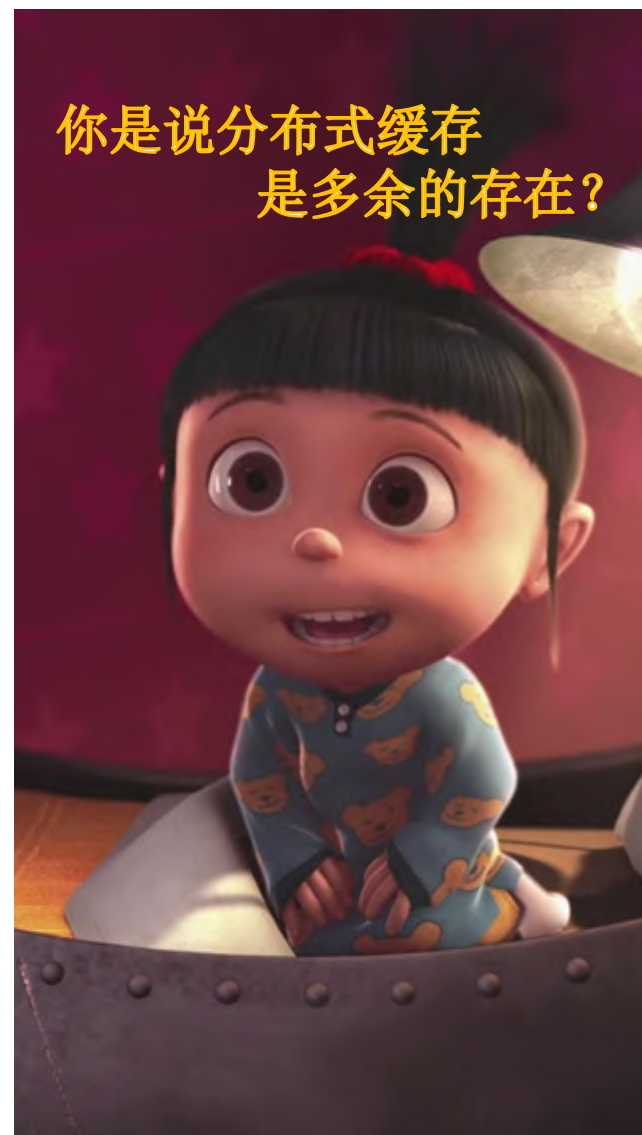


**业务逻辑应尽可能少的被技术落地姿势约束，
业务逻辑不应依赖或受制于技术实现的基础设施！！**

内存领域对象的高可用保障



你是说分布式缓存
是多余的存在？



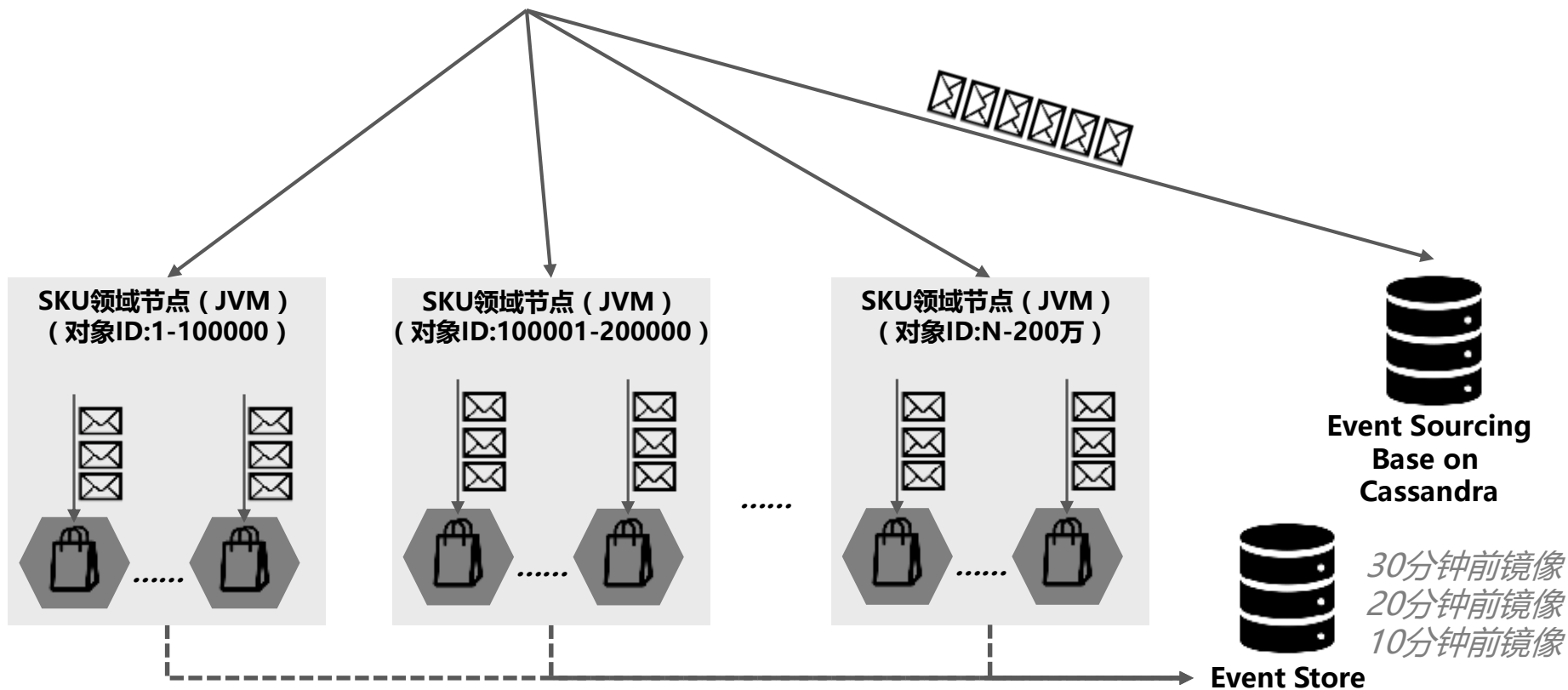
内存领域对象的高可用保障

10,000,000+用户

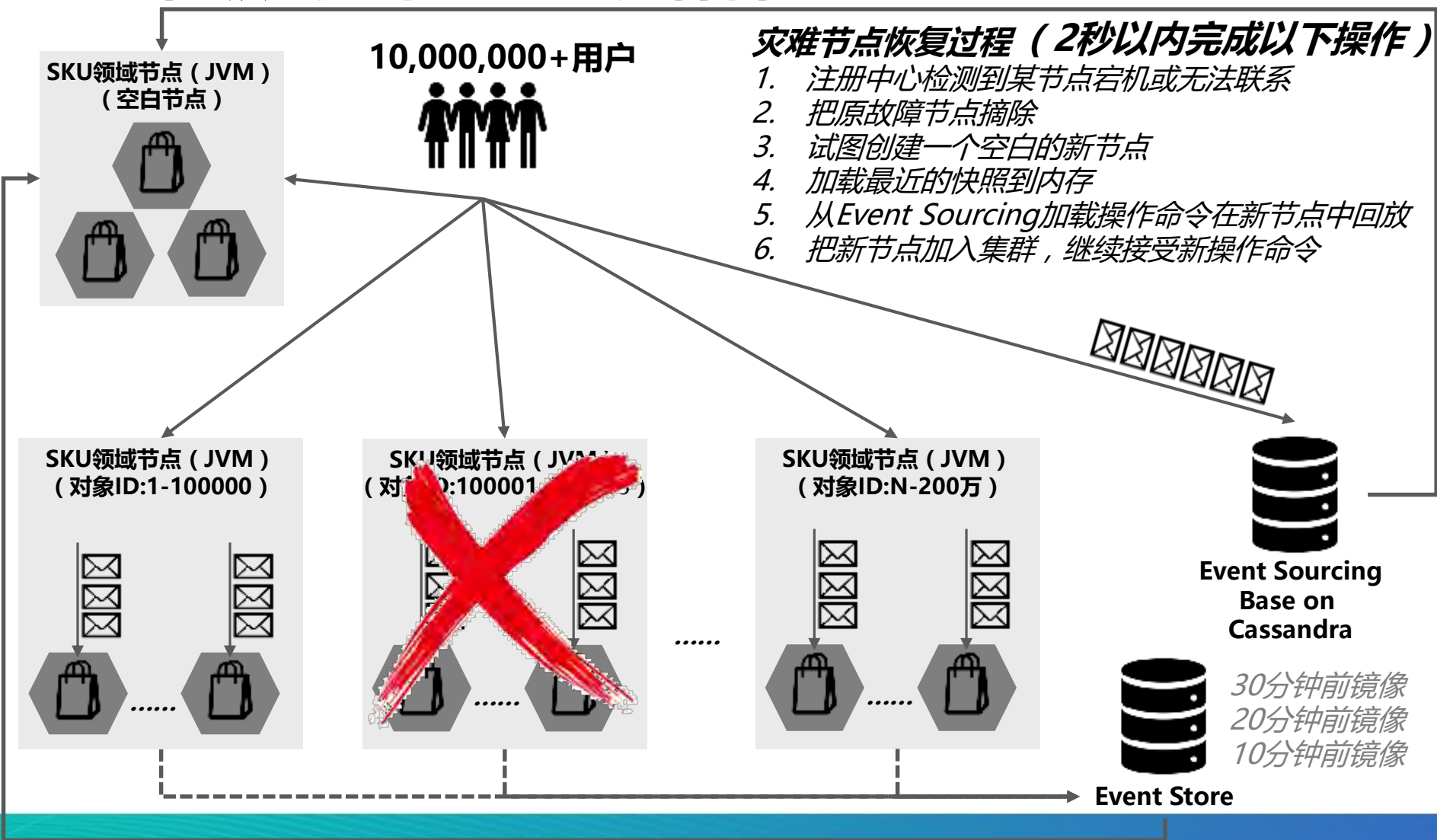


部署实践

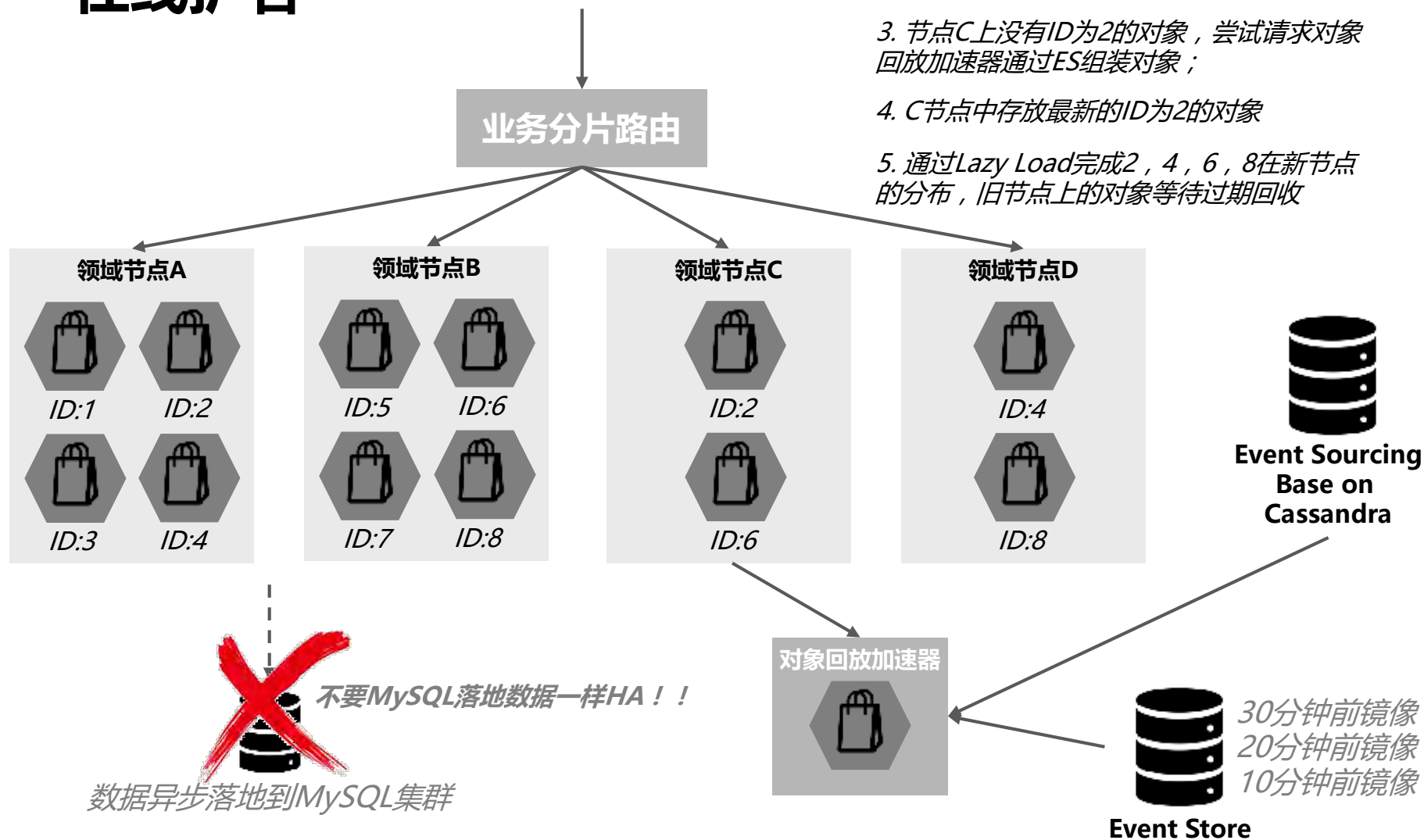
每64GRAM物理机上部署**3个**节点
每个节点20万SKU领域对象



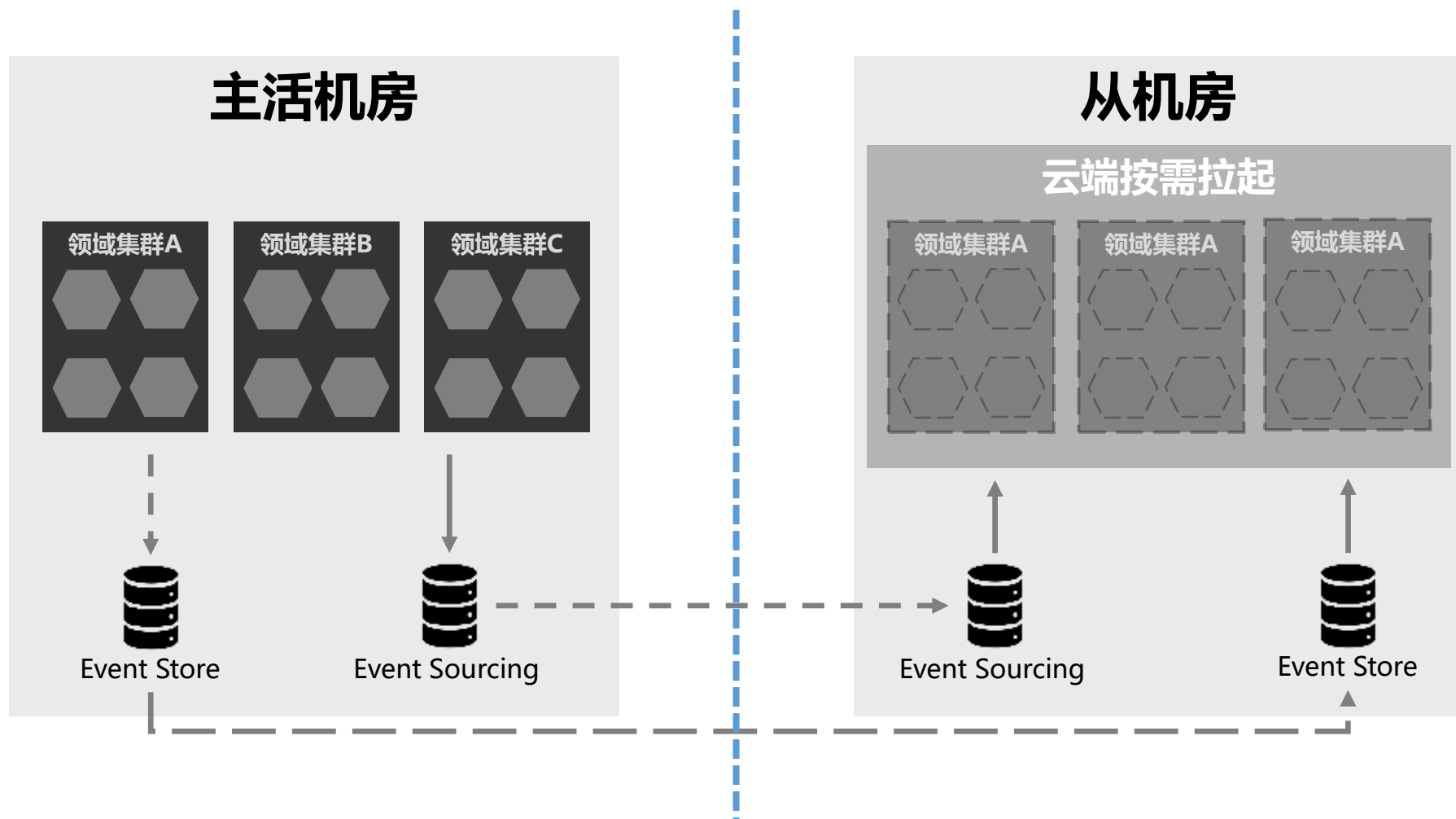
内存领域对象的高可用保障



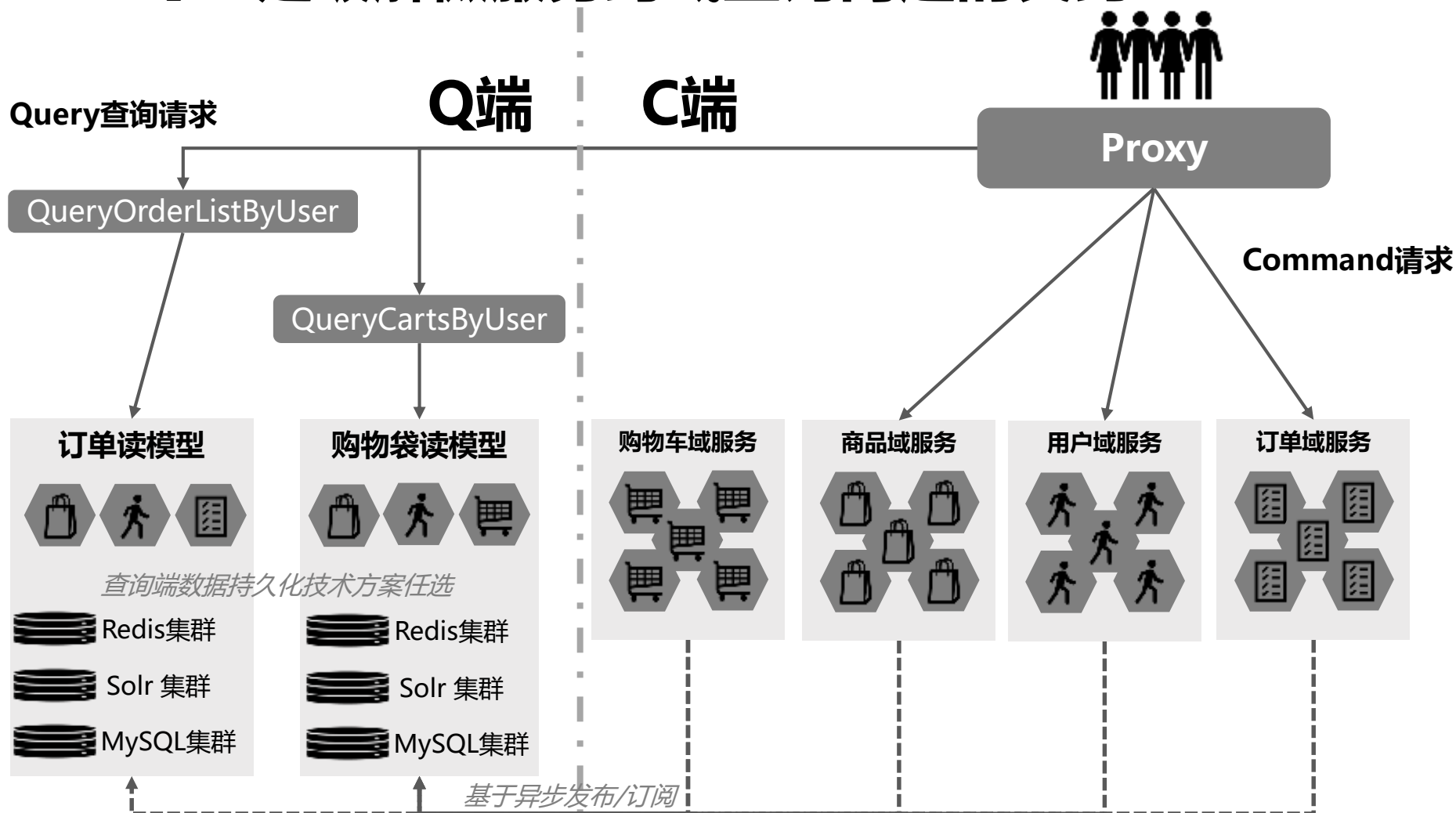
在线扩容



内存领域对象的高可用保障-跨机房



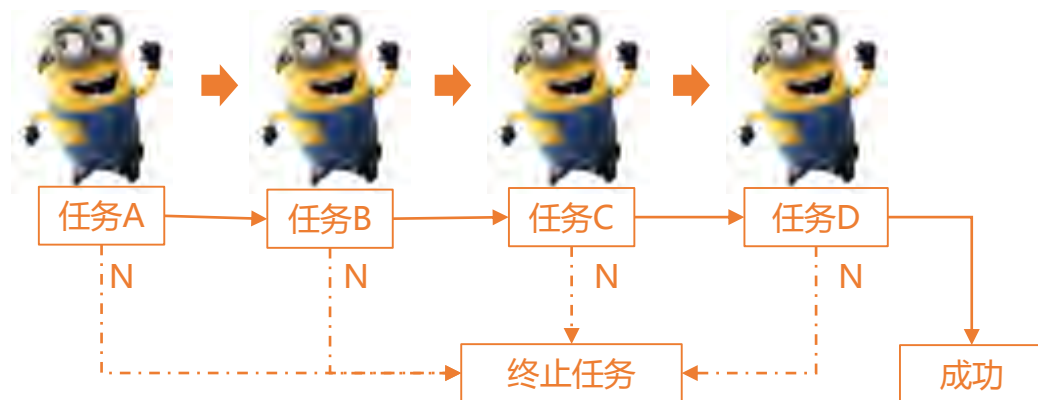
CQRS是破解微服务跨域查询问题的良药



乾坤的最终一致性事务模型



常规事务处理模式



乾坤采用纯异步并发事务处理模式



乾坤的最终一致性事务模型

老板的紧急任务

- 强调响应速度，最大并行化；
- 假设一定成功，不依赖其前置任务结果；
- 设定固定执行时间限制（超时设定）；
- 汇总执行结果判定老板的紧急任务是否达成。



优势与缺陷

- 并行度高，无阻塞，处理响应快；
- 增加额外的无效处理，以及回退操作开支；
- 实现中无法完全做到前后任务不依赖，但数字时间几乎不受这个限制；

乾坤的最终一致性事务模型

API 服务

Cart.Msg.AddCartGoods.1.2.3
(添加购物袋操作)

业务编排服务

Cart Processor

Loader (数据准备)

获取会员级别

获取营销决策

Saga (事务性处理)

增加会员积分

加购物袋

扣库存

操作成功并返回

执行失败流程

领域原子服务

GetByKey

GetByKey

营销域

+GetByKey()

+GetRuleByUser

.....
.....
.....

会员域

+GetByKey()

+AddScore()

.....
.....
.....

购物袋域

+GetByKey()

+AddCartGoods()

.....
.....
.....

商品域

+GetByKey()

+DeductSKU ()

.....
.....
.....

乾坤的最终一致性事务模型

API 服务

Cart.Msg.AddCartGoods.1.2.3
(添加购物袋操作)

epoll模式返回响应

业务编排服务

Cart Processor

Loader (数据准备)

获取会员级别

Level 3

获取营销决策

Ture

Saga (事务性处理)

增加会员积分

0

加购物袋

1

扣库存

1

判定主流程执行成功

领域原子服务

GetByKey

GetByKey

营销域

+GetByKey()

+GetRuleByUser

.....

.....

.....

.....

会员域

+GetByKey()

+AddScore()

.....

.....

.....

.....

购物袋域

+GetByKey()

+AddCartGoods()

.....

.....

.....

.....

商品域

+GetByKey()

+DeductSKU ()

.....

.....

.....

.....

一直尝试重试操作

乾坤的最终一致性事务模型

API 服务

Cart.Msg.AddCartGoods.1.2.3
(添加购物袋操作)

Cart.Msg.FailedToAddCartGoods.1.2.3

epoll模式返回响应

业务编排服务

Cart Processor

Loader (数据准备)

获取会员级别

Level 3

获取营销决策

Ture

Saga (事务性处理)

增加会员积分

1

加购物袋

0

扣库存

1

判定主流程执行失败

领域原子服务

GetByKey

GetByKey

营销域

+GetByKey()

+GetRuleByUser

.....

.....

.....

.....

会员域

+GetByKey()

+AddScore()

.....

.....

.....

.....

购物袋域

+GetByKey()

+AddCartGoods()

.....

.....

.....

.....

商品域

+GetByKey()

+DeductSKU ()

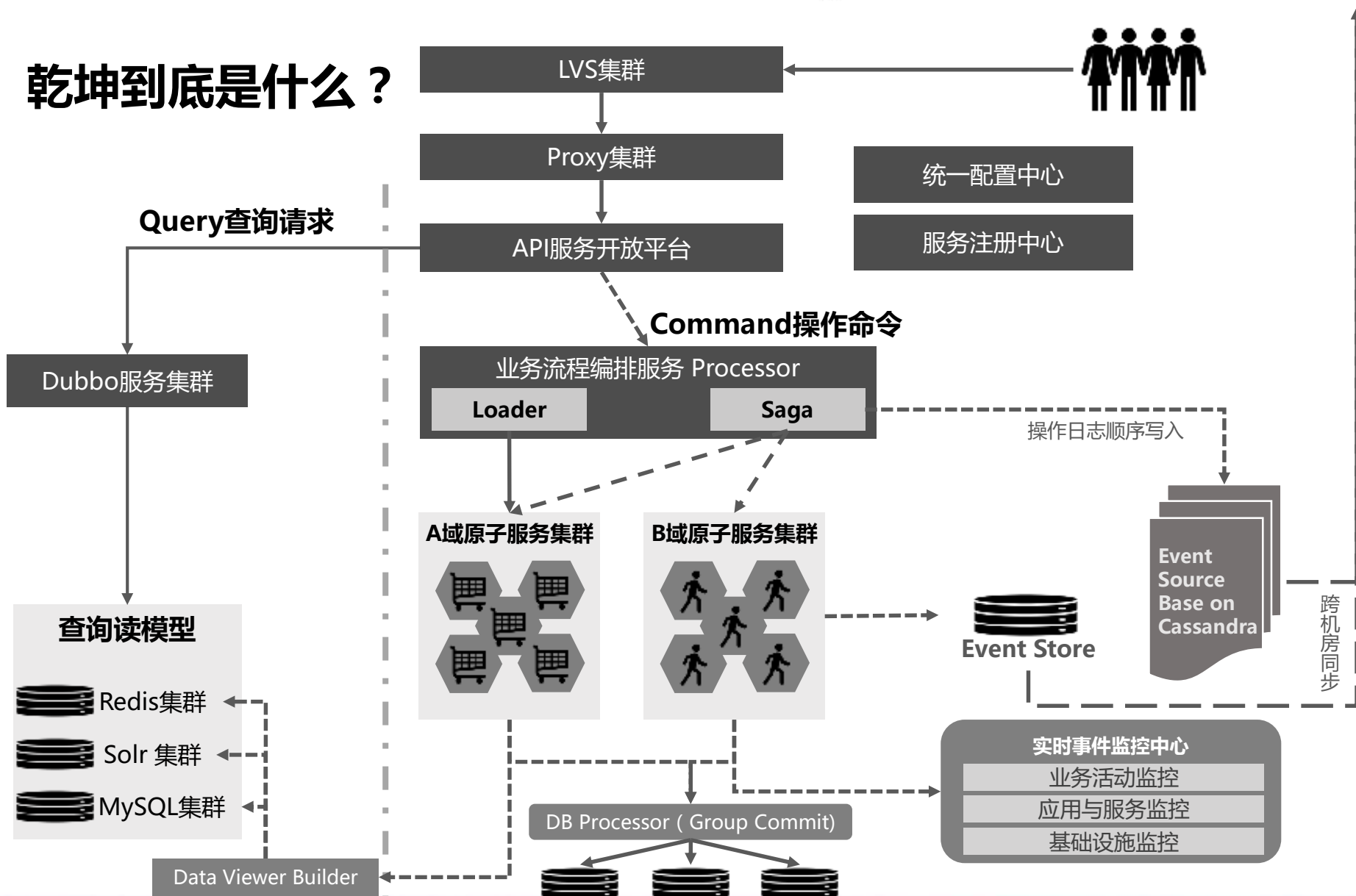
.....

.....

.....

.....

乾坤到底是什么？



乾坤到底是什么？

- 一种**领域驱动设计的微服务架构**最佳实践落地方案

- 高度解耦：读写分离，领域隔离，发布/订阅；
- 无需关心高并发争抢；
- 无需关心数据持久化方式；
- 无需过多考虑分布式事务；
- 无需过多担心性能、扩展性问题；
- Actor & Service模式，即领域对象Command & 读模型Query;

- **轩辕在团队的实践经验**

- 需求分析人员：领域模型、业务流程（正向流程、失败处理流程）；
- 设计人员：服务接口定义，接口验收；
- 开发人员：领域对象功能实现、读模型查询实现；

一些扩展话题

1. AWS和Azure Fabric Service提到的Actor & Service模式是云计算的未来；
2. 基于事件机制的应用与业务监控体系；
3. ES体系与区块链技术的亲缘性；
4. 架构参考：Axon Framework, Akka, Disruptor , ENode；

常见问题Q&A