结算平台高性能设计与实践

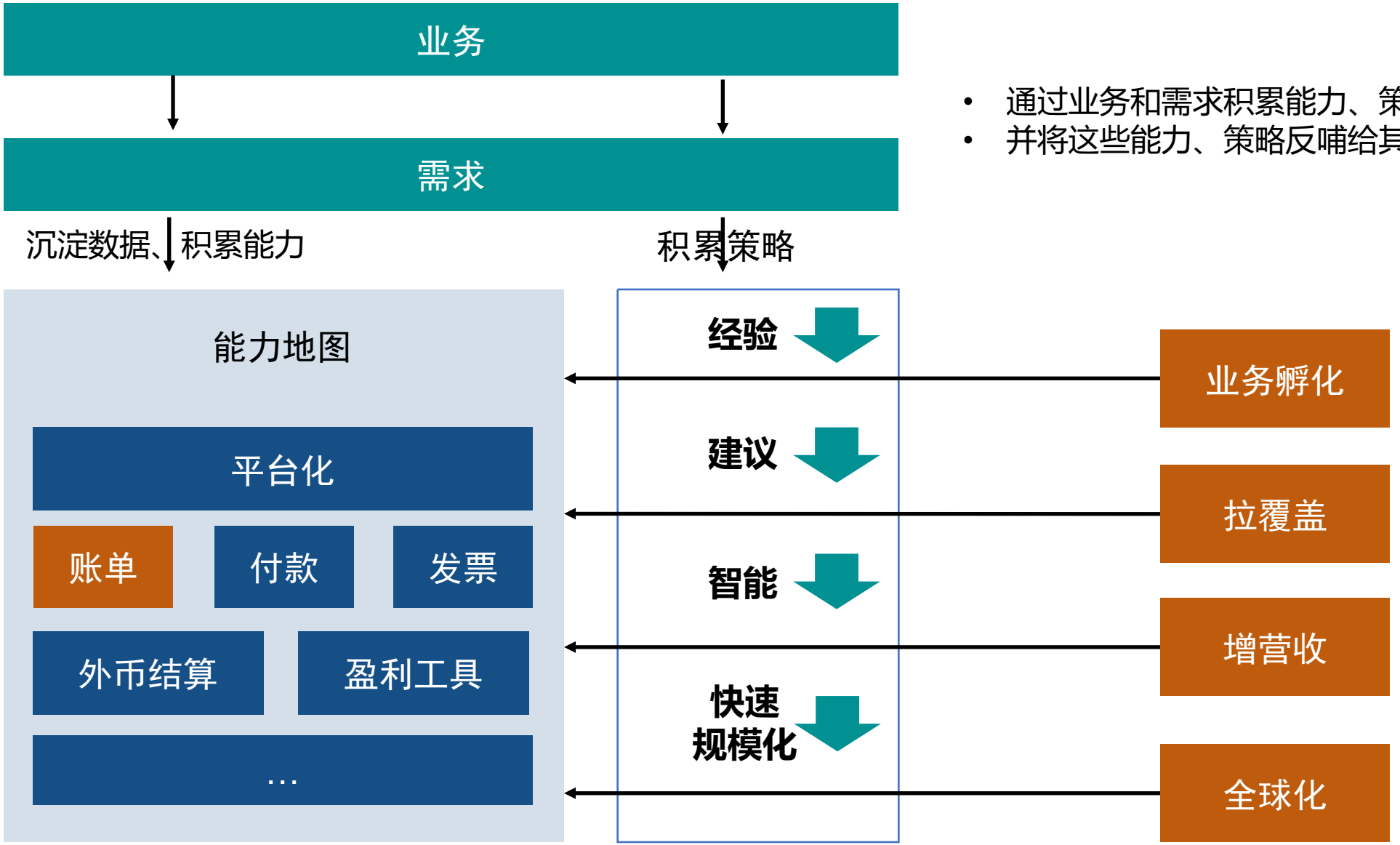
酒旅业务研发中心-张子鑫

CONTENTS

个人简介

CONTENTS

结算平台简介



- 通过业务和需求积累能力、策略
- 并将这些能力、策略反哺给其他业务

业务支持情况

住宿	境内度假	境外度假	交通&保险
预定	门票	境外	火车票
团购	跟团	X	船票
房惠买单	酒景		X
海外&高星			
打包			

- 支持四个事业部
- 17条业务线涵盖境内&境外结算

CONTENTS

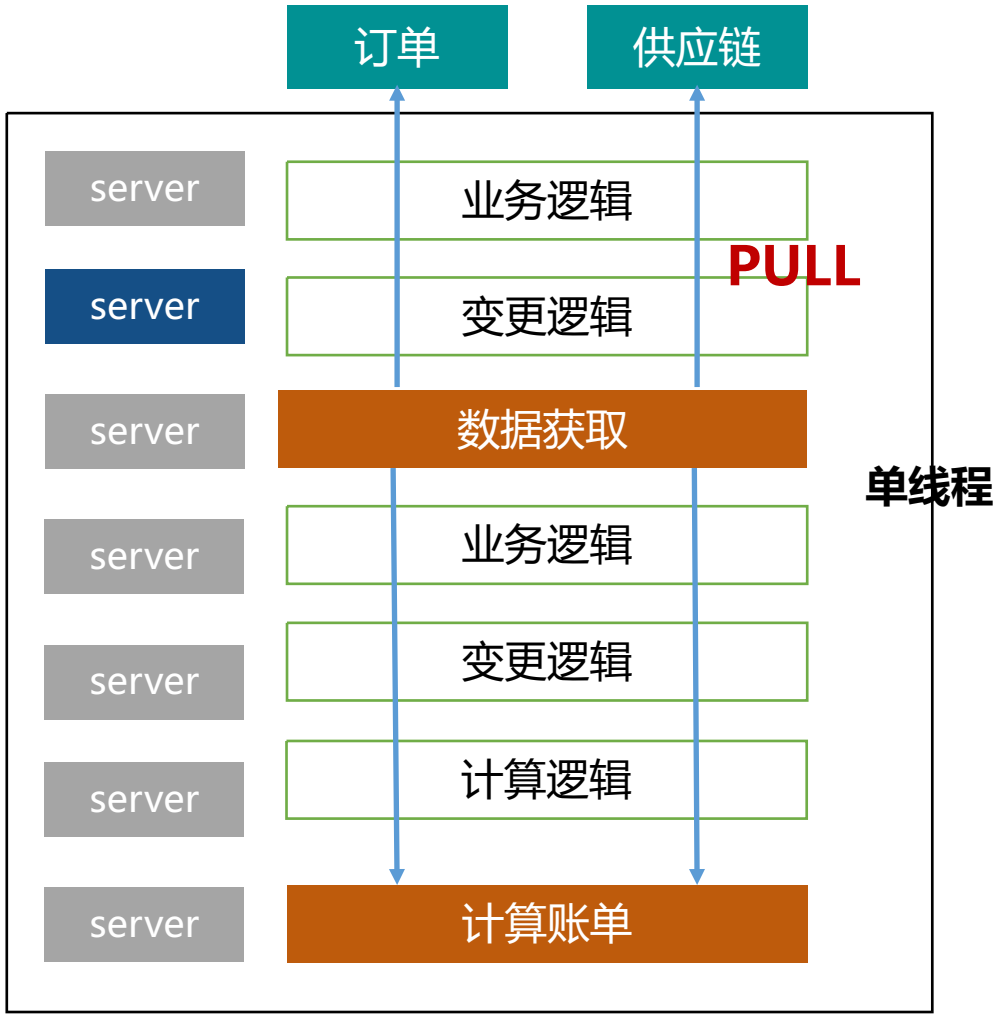
什么是账单？

- 业务的重要性
- 技术的挑战

账单				
账单ID	商家	账单期	金额	…
1	如家	周结算 20170703- 20170709	100	…

账单明细		
账单ID	订单ID	金额
1	1	5
1	2	10
1	3	5
1	4	80

最初的实现和问题



期末账单（账期6月1号~6月30号），7月1号出账

存在问题：

1. 业务逻辑耦合，PULL模式，重度外部依赖
2. 期末账单，计算压力大，风险不可控
3. 单线程，单server，浪费资源
4. 破产流言

X 17

早期的优化-按时出账

策略

数据分片

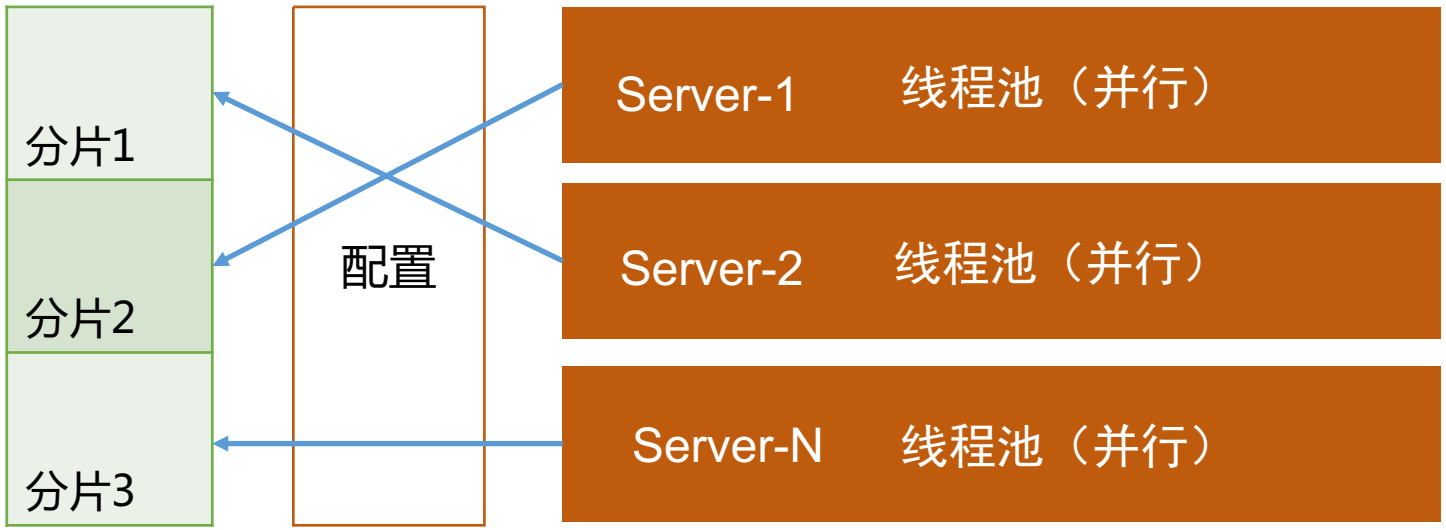
并行处理

合理利用资源

提高数据处理效率

商家ID取模分片

商家ID并行



单点问题

数据重分片问题

能扩展但不灵活

早期的优化-资源利用问题

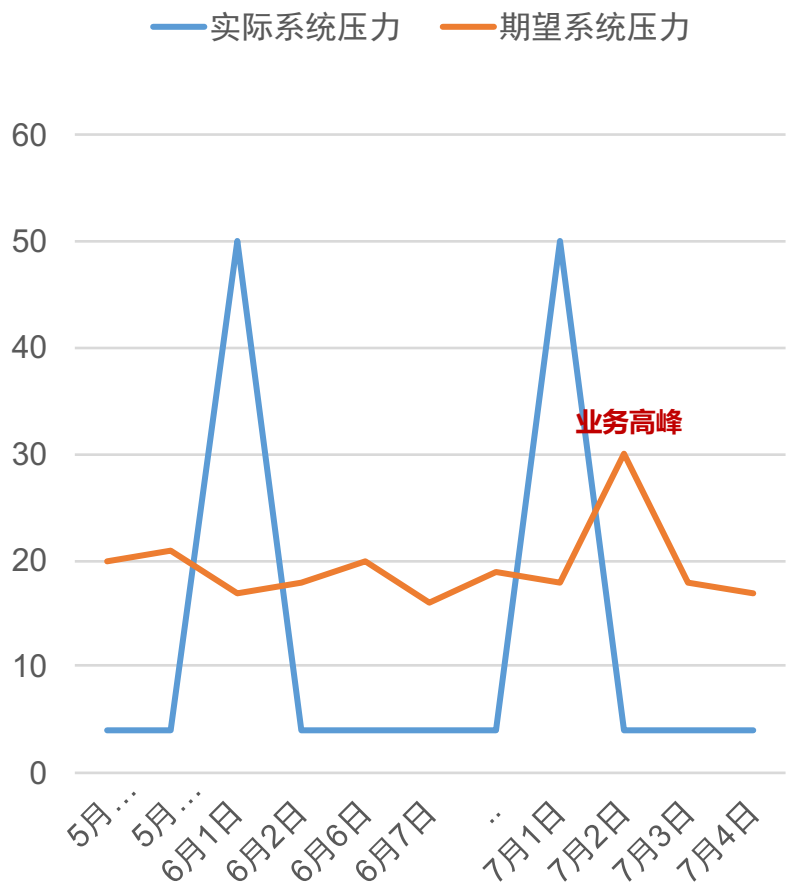
- 数据获取&计算，依赖上游、下游SLA
- 速度和资源的消耗求平衡
- 增加了架构的复杂性&产生资源浪费

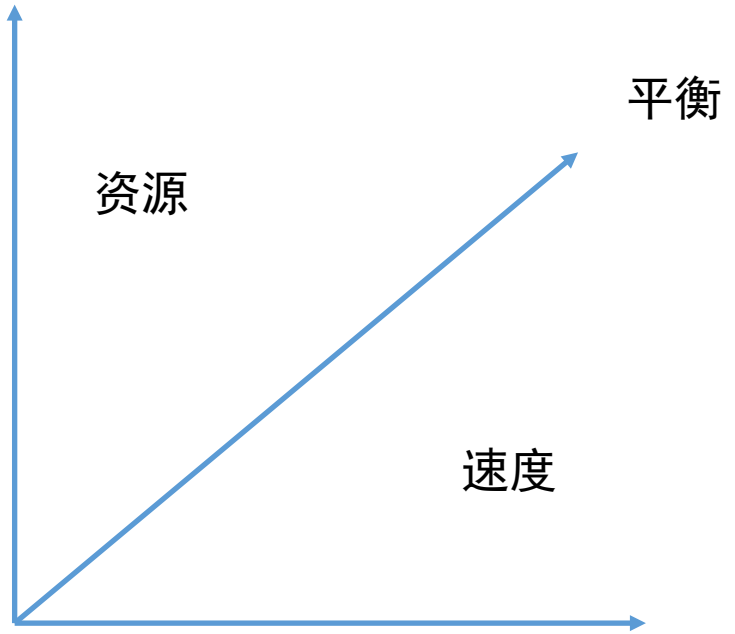
订单系统

供应链

账单Server

账单DB

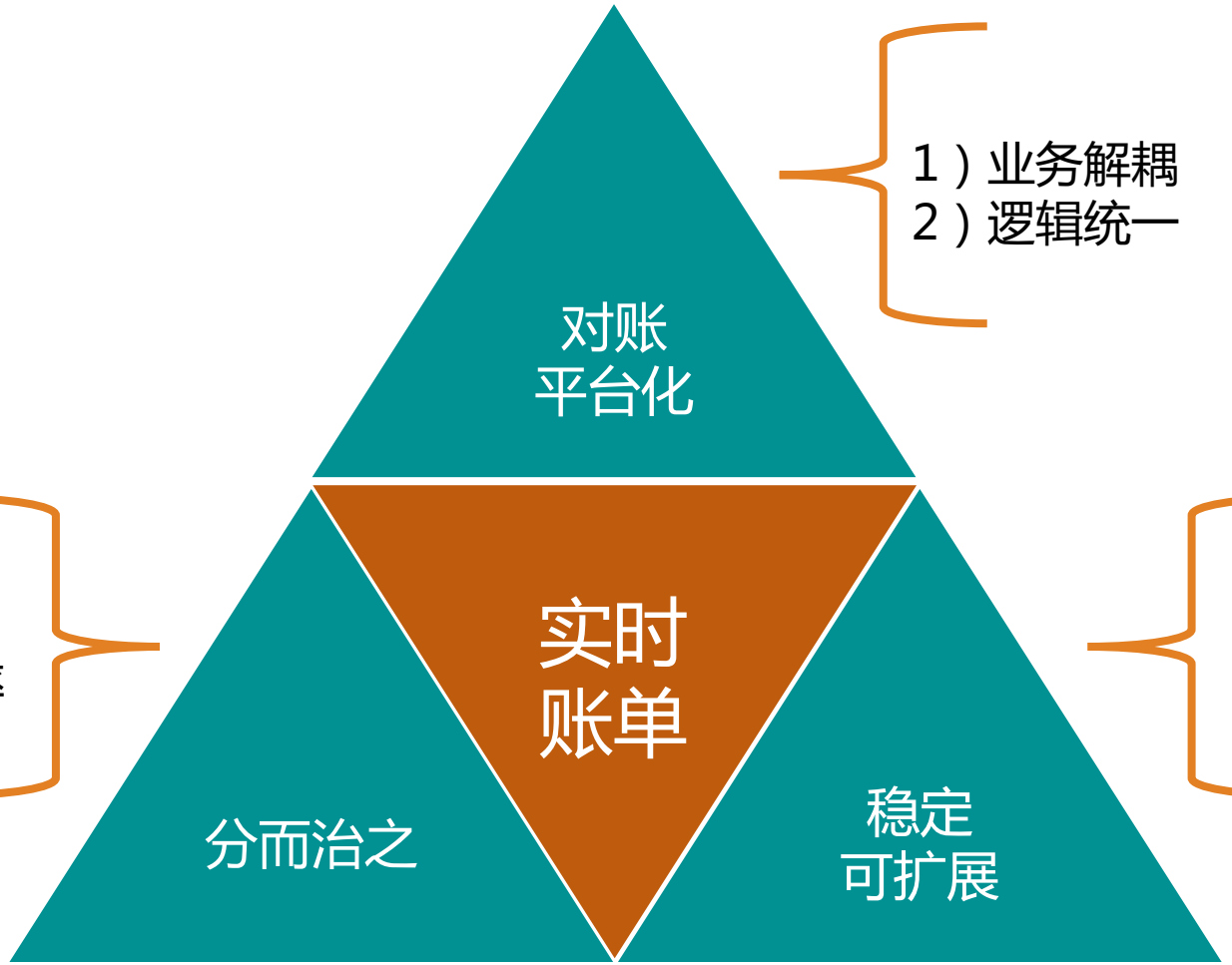




早期的优化效果

- 串行计算->并行计算
- 出账延迟5-15天->1天内
- **业务逻辑耦合，重度外部依赖**
- **系统负载不均衡，资源浪费**
- **单点问题**

破局思路

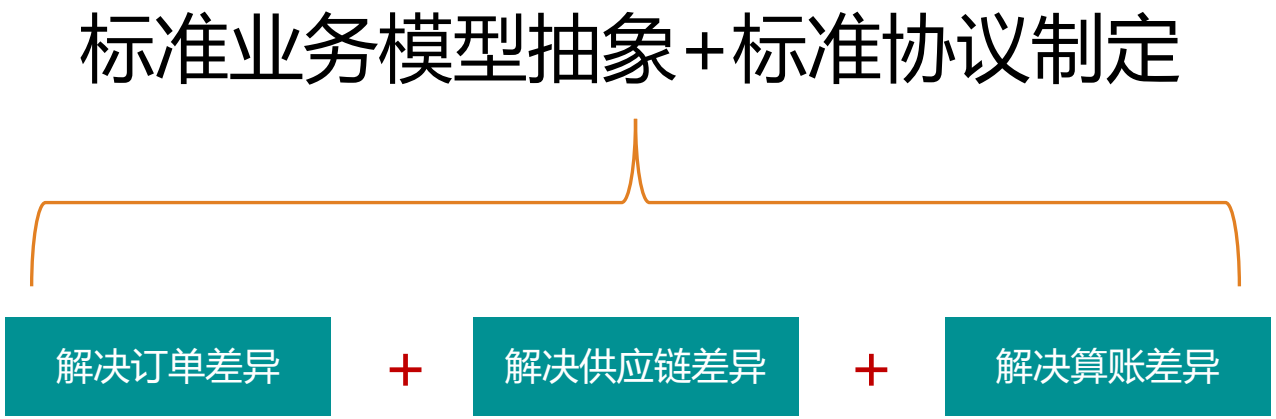


1）业务解耦
2）逻辑统一

1）高性能实时计算
2）平稳的资源利用率
3）杜绝浪费

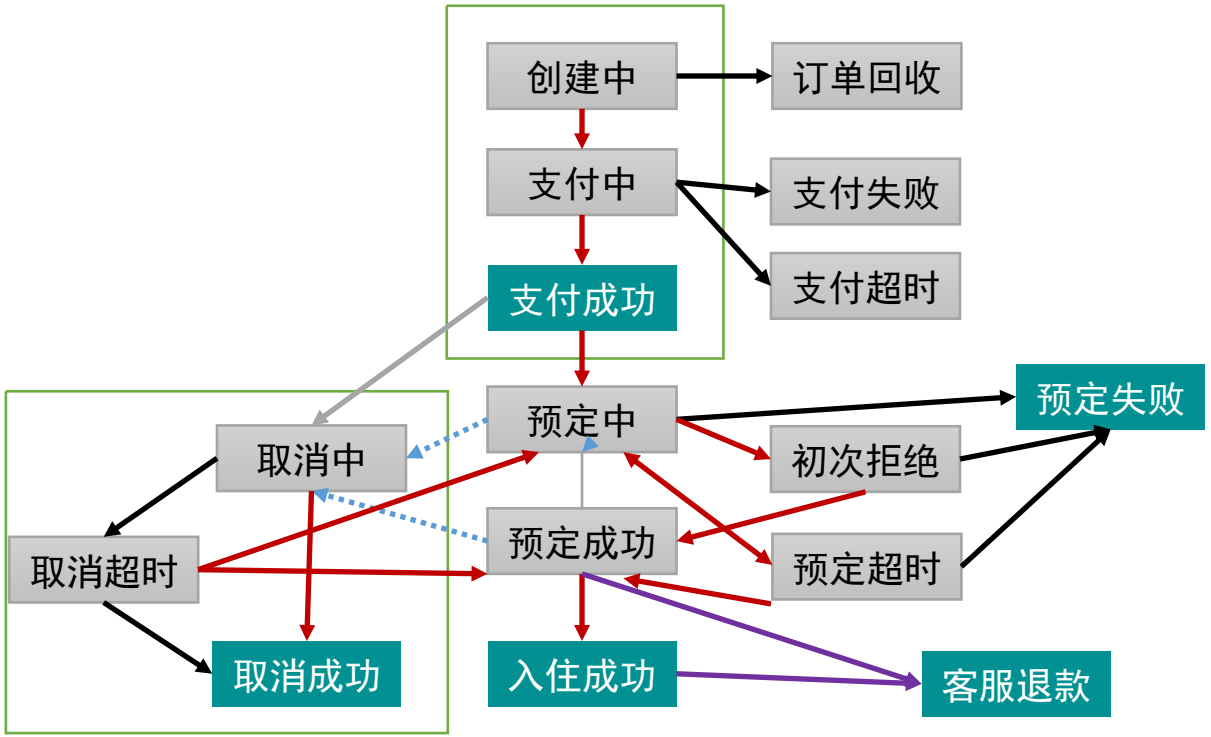
1）消除单点
2）水平扩展

对账平台化



订单业务梳理

业务变更过滤规则必改，过滤规则无法复用



抽象统一订单状态

结算关注金钱变更和结算时机

订单业务梳理

酒店计算因子

- 订单基础信息
- 订单间夜信息
- 订单支付模型
- 订单退款记录
- 订单促销模型
- 退款承担方
- 促销承担方
- 退款是否退促销
- X

旅游计算因子

业务变更计算规则必改，计算规则无法复用

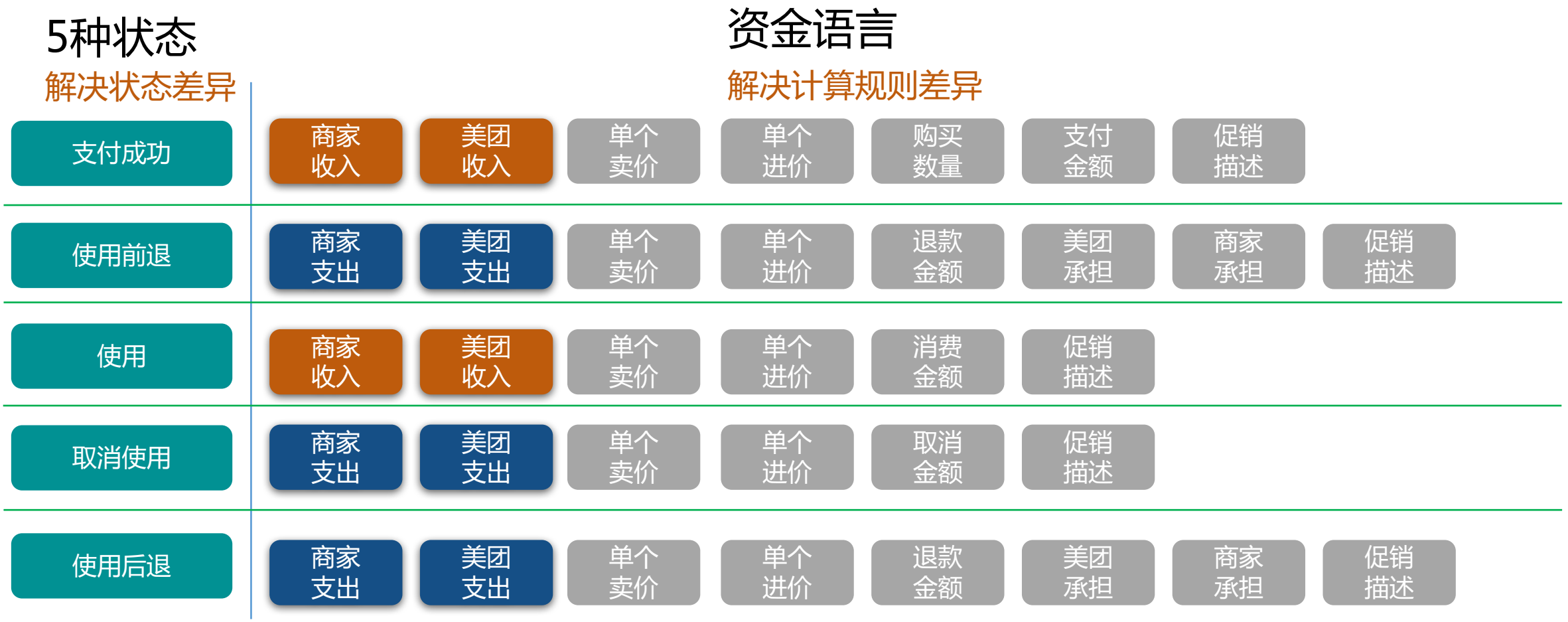
计算规则

结算关注资金

- 收入
- 支出

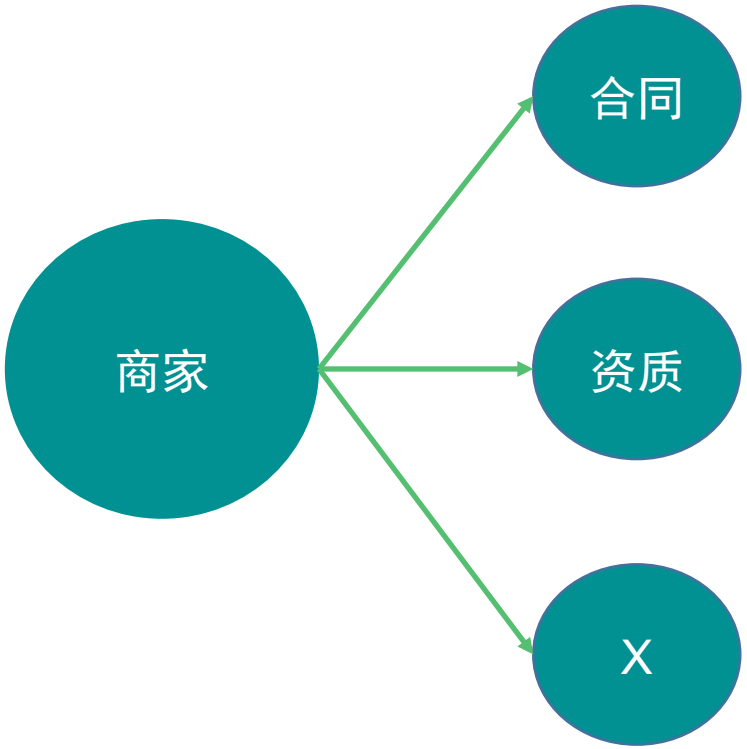
订单模型抽象-解决订单差异

抽象资金语言、所有订单通用、结算和订单解耦



供应链模型抽象-解决供应链差异

抽象商家结算模型、所有供应链通用



统一商家信息模型

基本信息	结算信息
业务线归属	门店
商家名称	合同
X	商家
	周
	日
	月
	满额

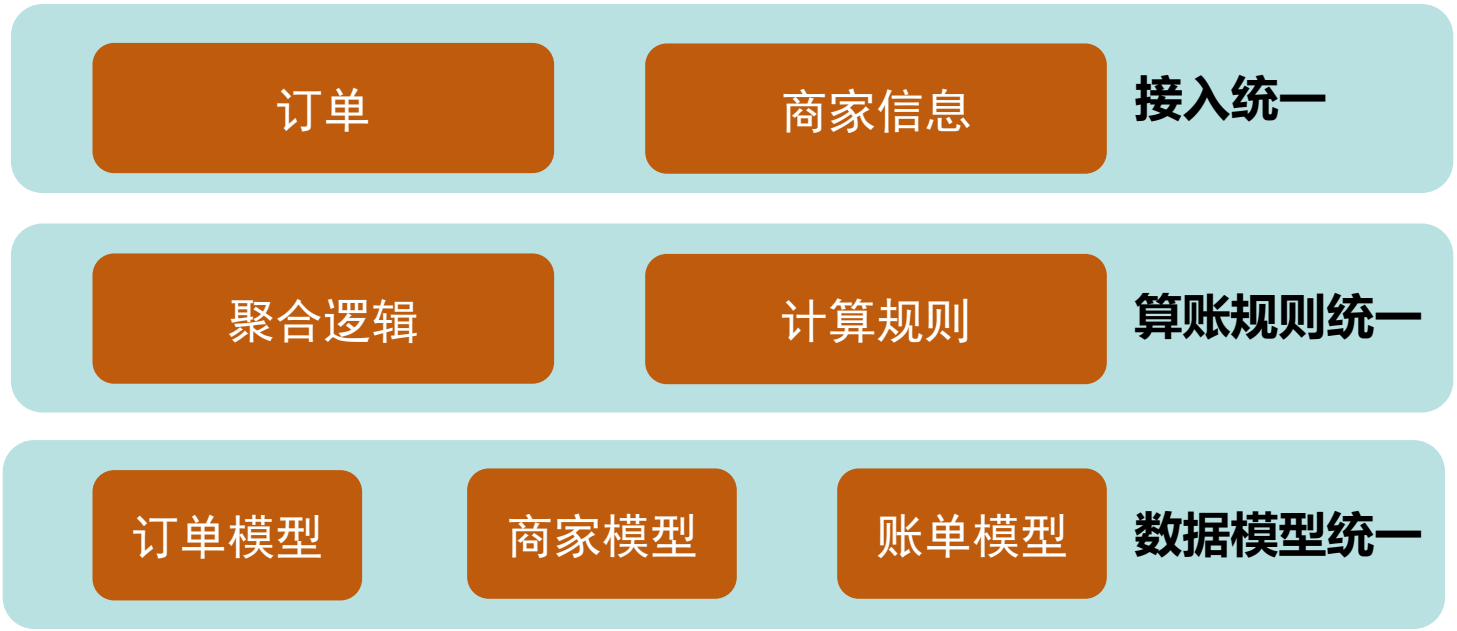
酒店
商家对账--商家付款
门店对账--商家付款
旅游
合同对账--合同付款
其他
商家对账--商家付款
…

对账&付款维度

对账&付款时机

火车-满100W结算一次
其他-时间

对账平台化思路

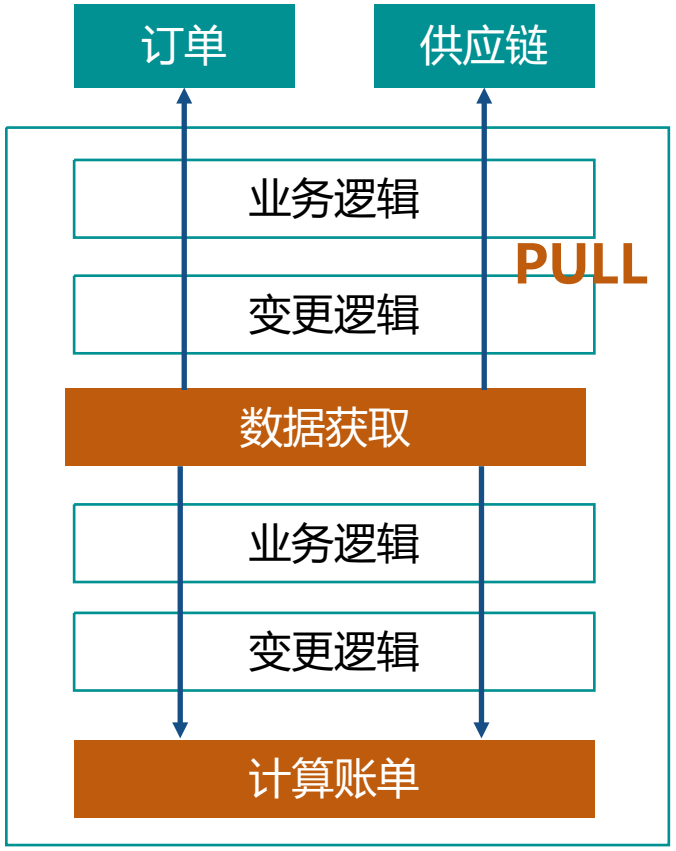


模型统一、关系统一、减少依赖

对账平台-设计

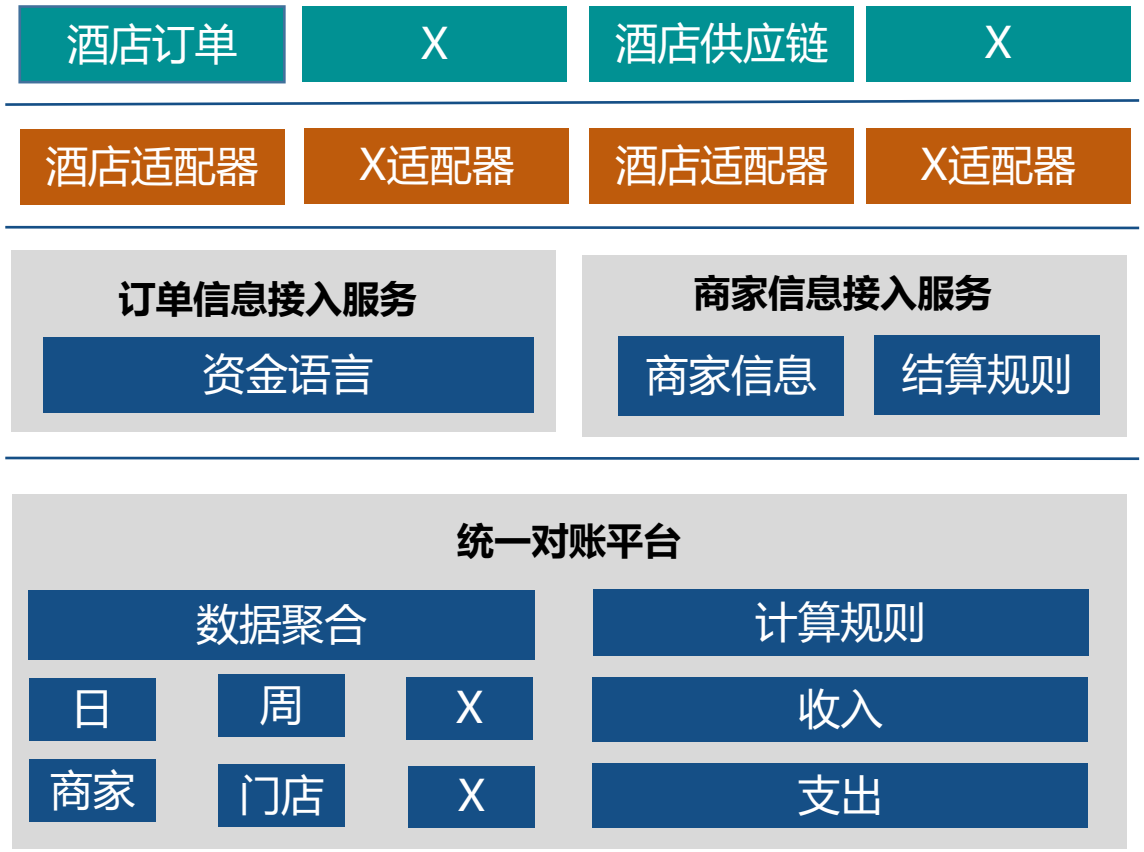
老架构

业务逻辑耦合
难以复用
重度外部依赖



新架构

业务逻辑解耦
高度复用
弱化外部依赖



业务层

适配层（业务逻辑解耦）

接入层（规范、**PUSH**）

计算层（统一）

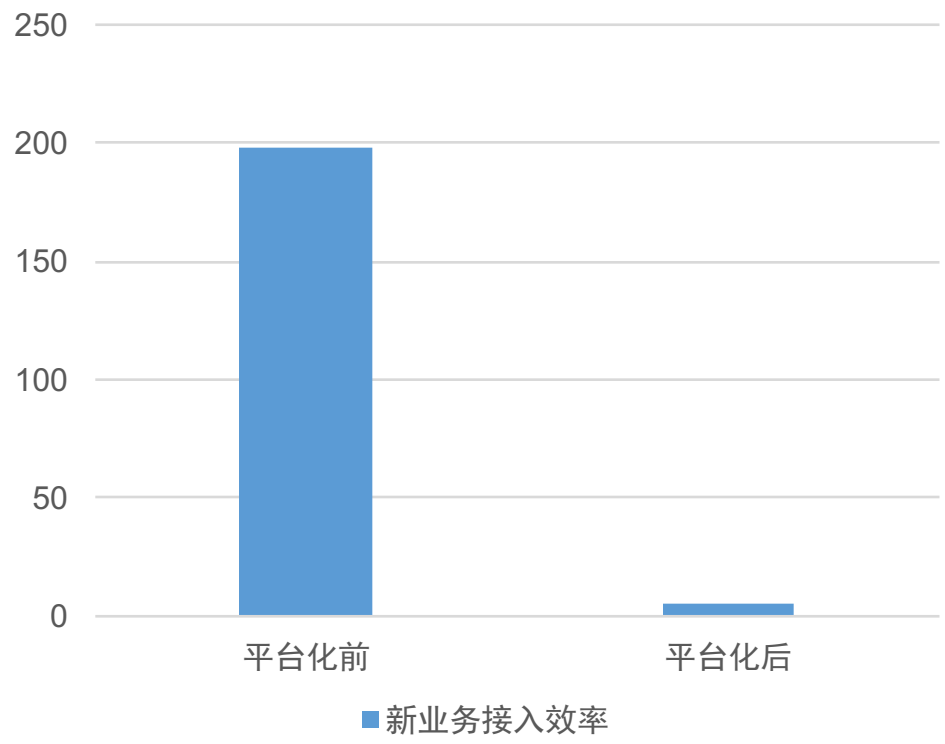
数据聚合统一

计算规则统一

数据模型统一

对账平台化-效果

人效提升



人效提升38倍

标准业务接入

3X3人/月 → 5PD

综合效果

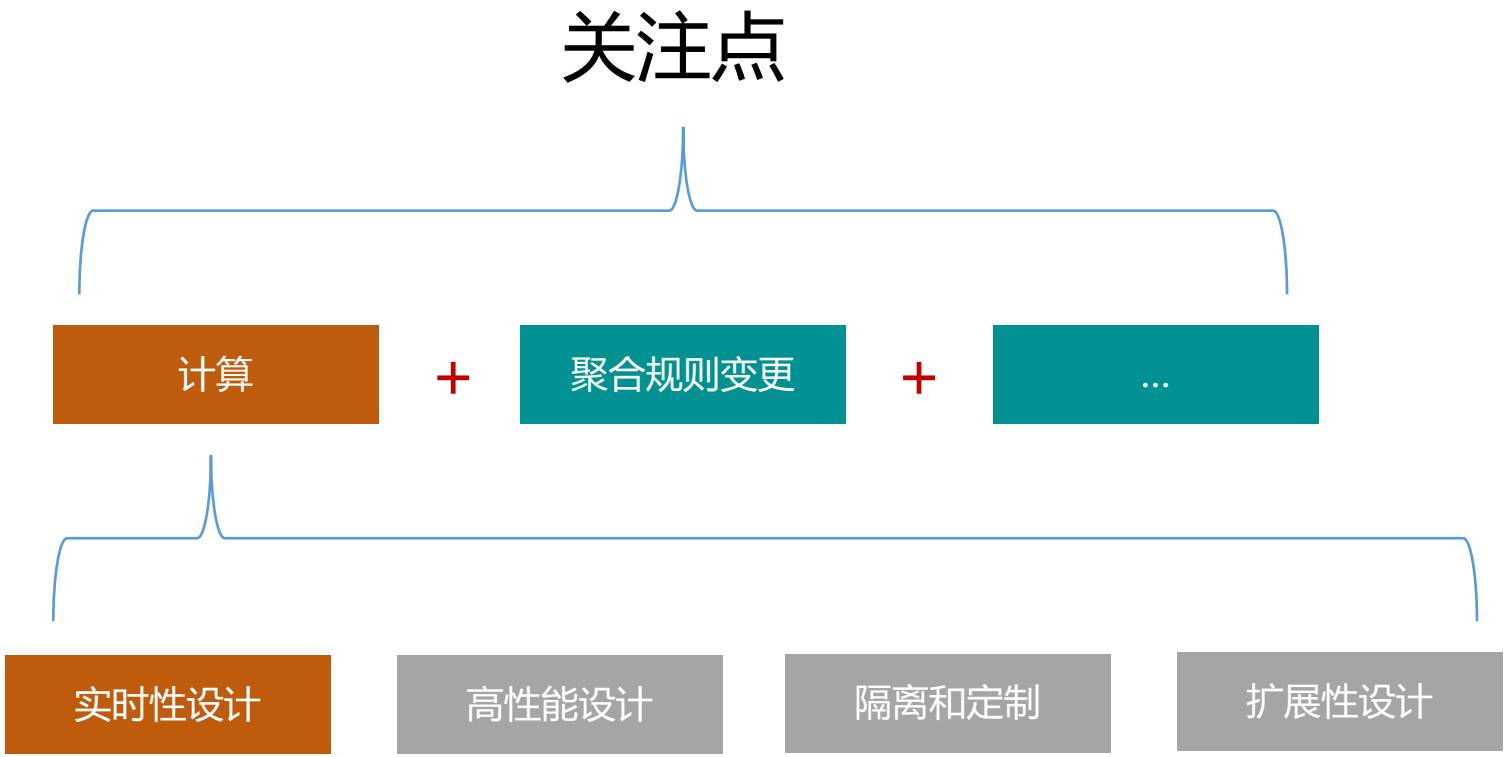
一、彻底解耦

旅游+酒店订单大重构结算没影响

二、人员复用

4名RD对接17条业务线
消除17倍问题

对账平台实践关注点



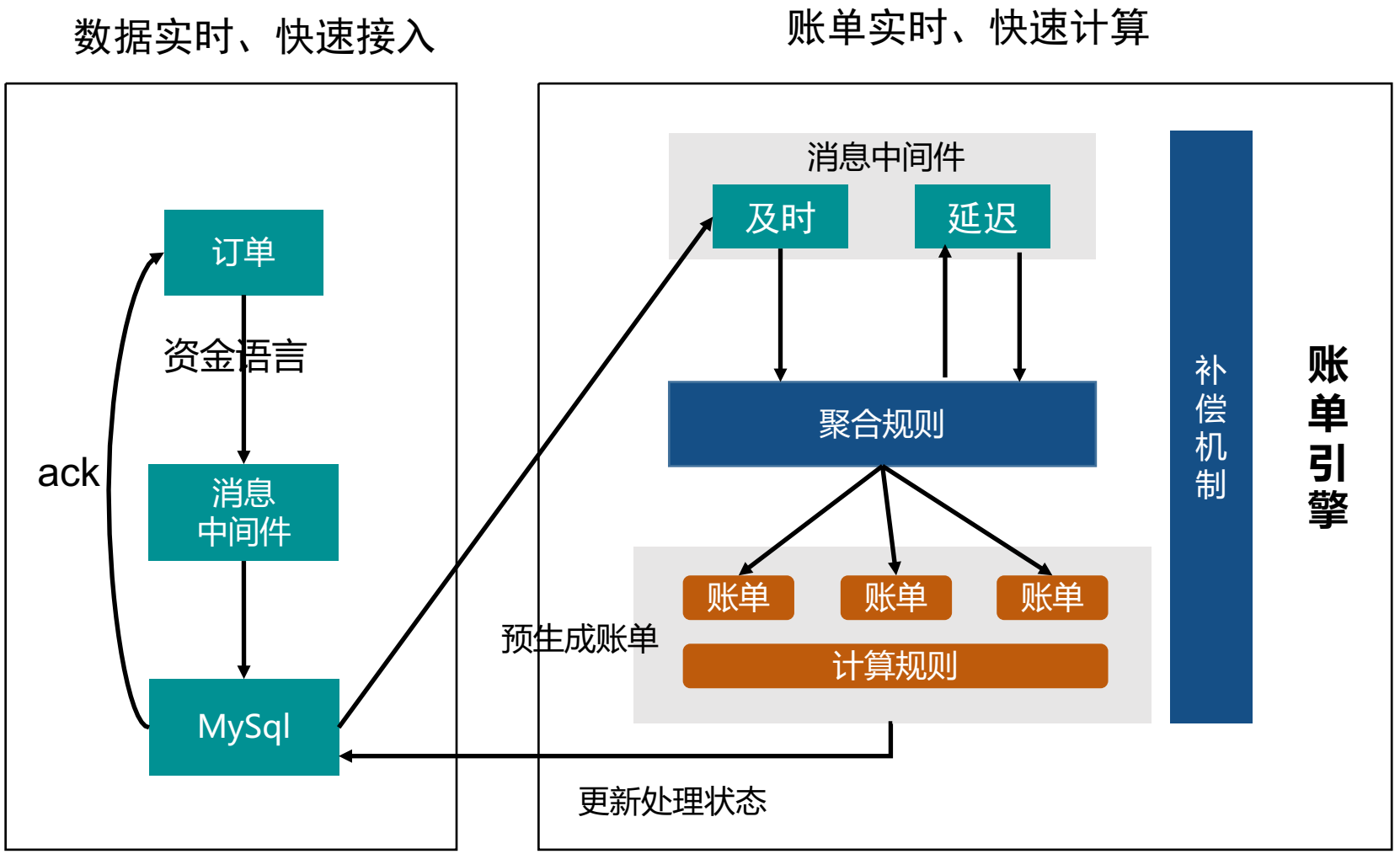
实时性设计

数据接入

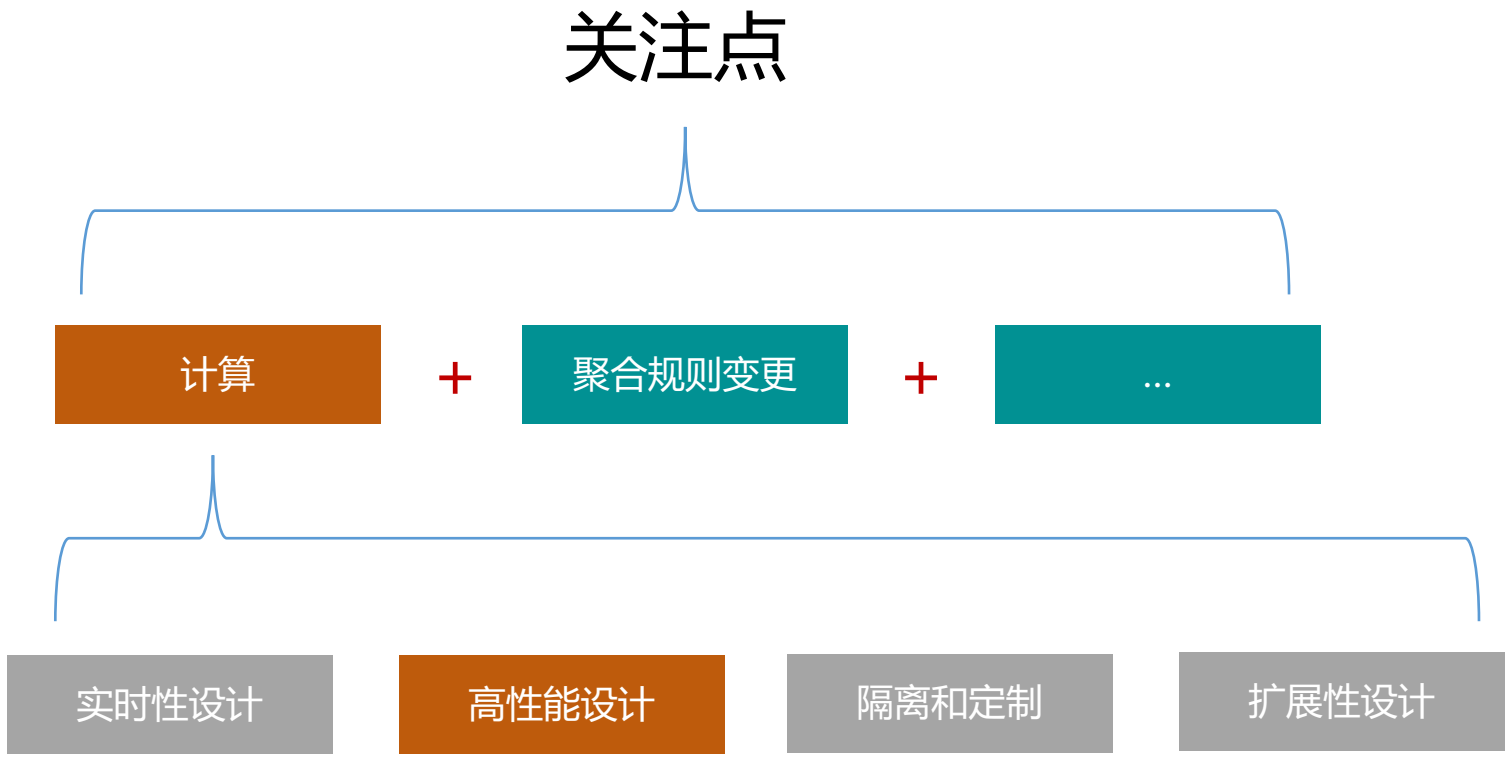
- 接入&计算分离，快速接入
- ACK机制保证数据不丢
- 延迟毫秒级别

计算引擎

- 预生成账单&实时入账
- 消息驱动、双队列设计
- 状态机设计保证数据完整性
- 延迟秒级

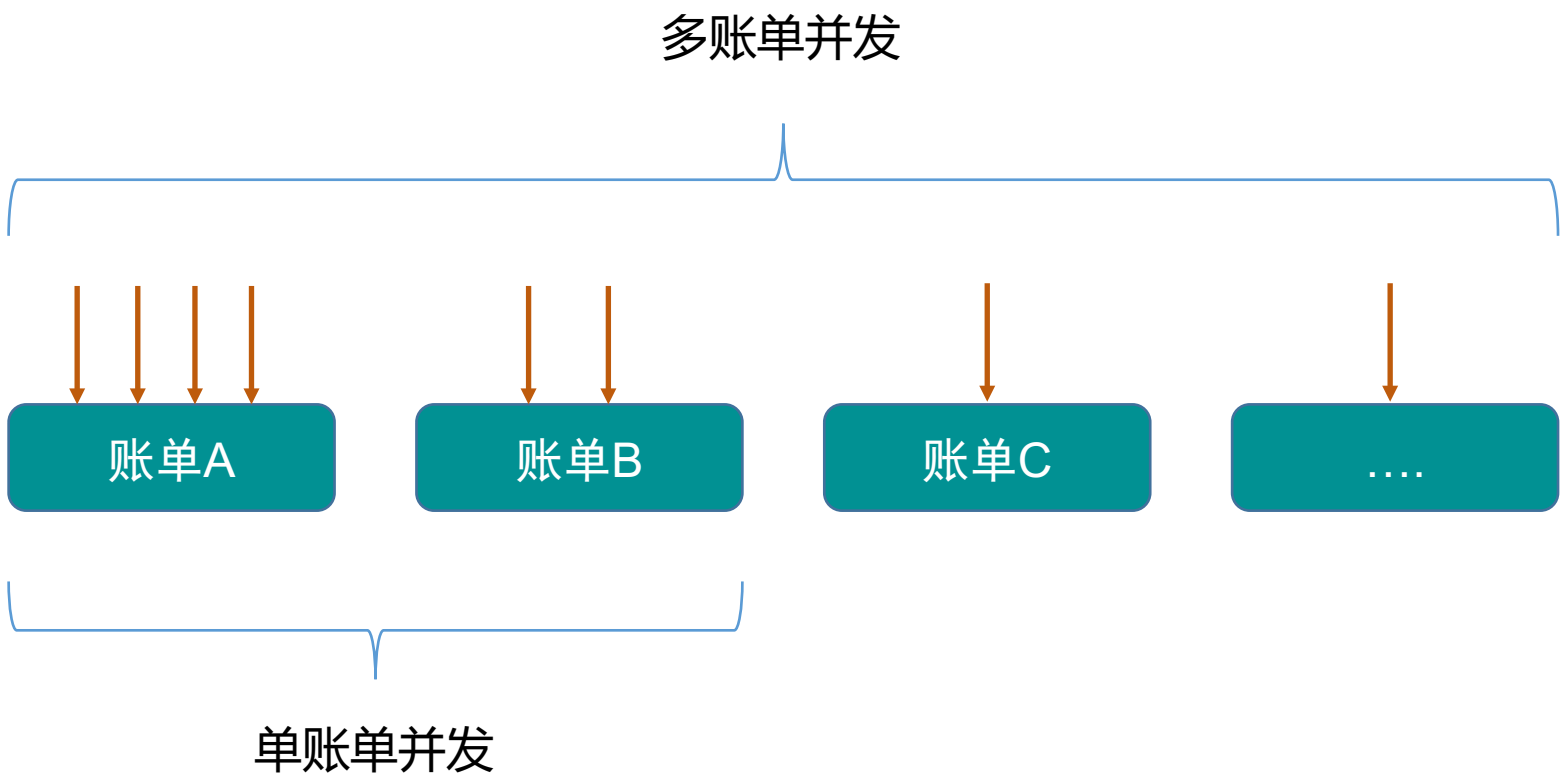


对账平台实践关注点



高性能设计-提高并发度

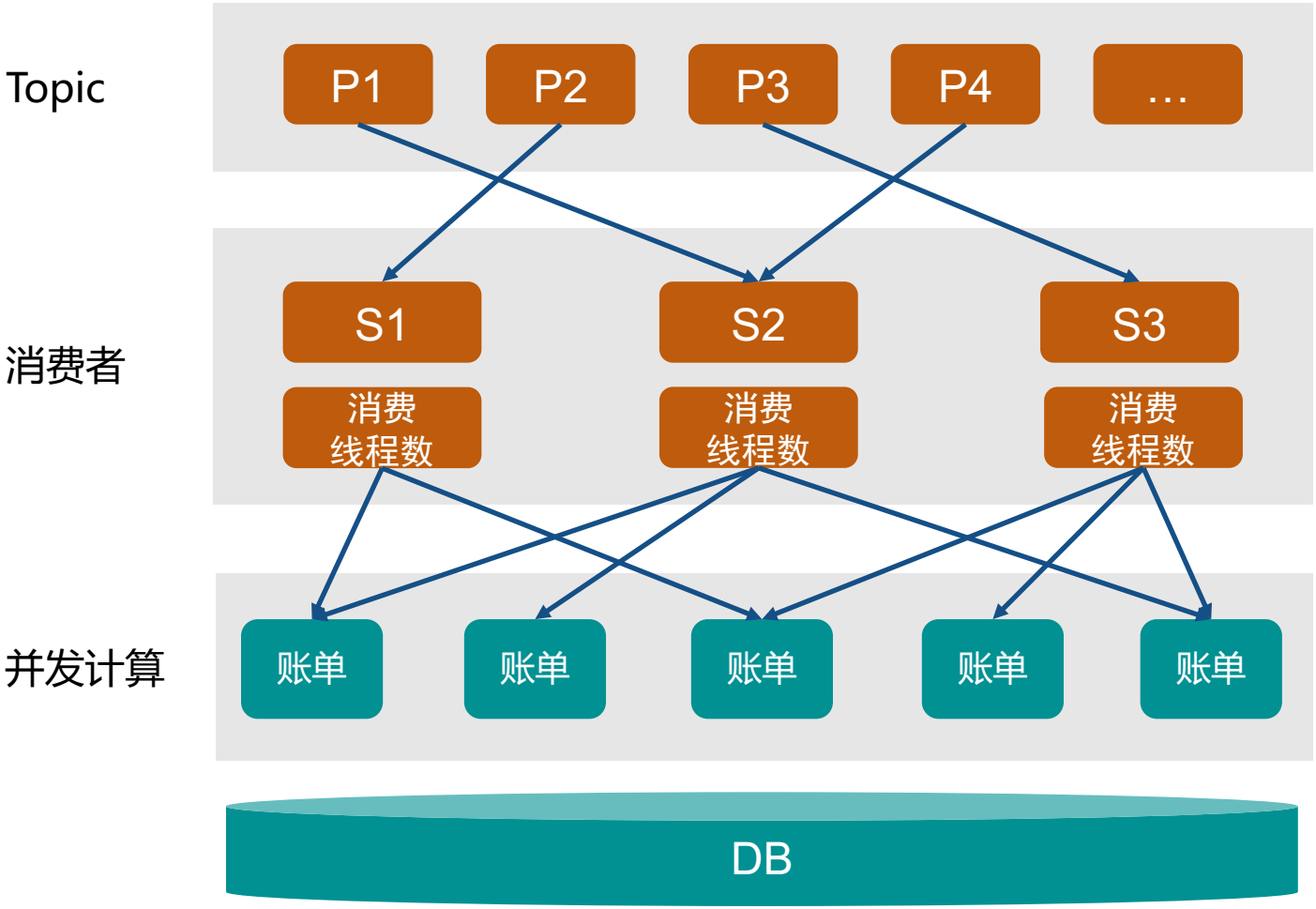
账单并发度越高计算越快



高性能设计-提高并发度

消息中间件（Mafka）

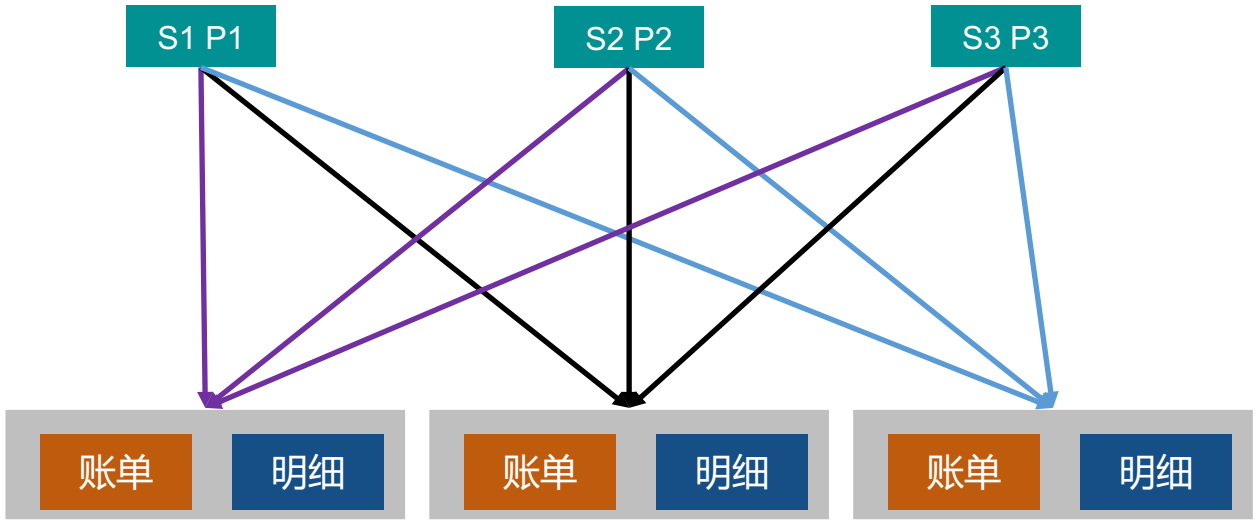
基于消息中间件特性控制并发度，账单维度高并发计算



- Topic级别按需配置Partition数量
- ConsumerGroup级别按需配置消费线程数
- Mafka基于滑动窗口机制支持多线程消费partition
- 并发度=分区数 X 消费线程数
- 计算幂等性保障
- 除了DB全链路无单点

高性能设计-并发带来的问题

流水入账 = 写明细 + 计算账单金额



问题

并发环境下怎么保证账单的金额正确？
怎么保证前边的流水不阻塞后边的流水？

出列，减少阻塞

高性能设计-并发带来的问题

常规实现

```
//事务保证原子性
try {
    Lock lock = distributedLockManager
            .getReentrantLock(statementId);//账单维度锁
    lock.lock(); //获取锁，存在阻塞
    insert(detail);//写入明细
    compute(statementId);//计算账单
} finally {
    lock.unlock();// 释放锁
}
```

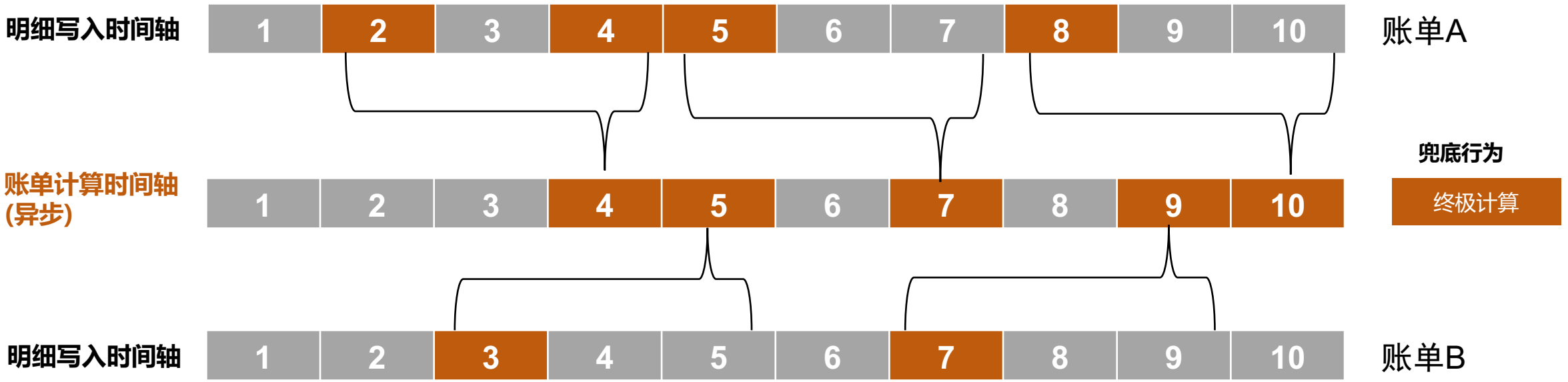
问题

- 账单加锁，同一账单的**并发度为N**，**并行度为1**，造成队列阻塞
- 分布式锁、事务存在一定的性能开销，影响账单计算效率

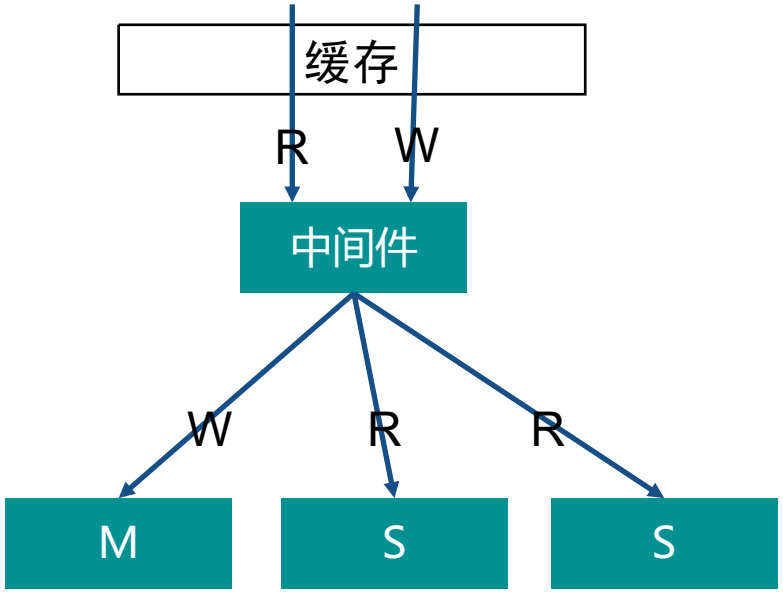
高并行度
- **无锁** > **降低锁粒度**
- **无事务** > **降低事务粒度**

高性能设计-提高并行度

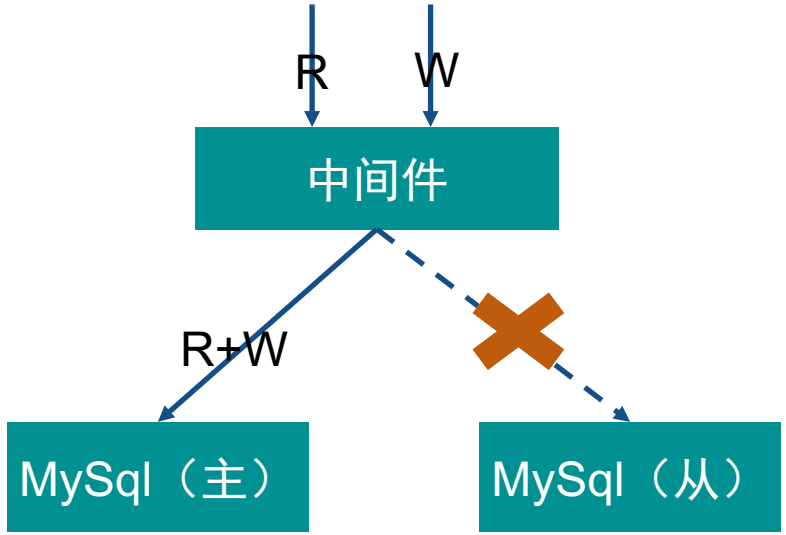
- 明细写入&账单计算分离，较短时间窗口的数据不一致换性能
- 无锁、无事务、同一账单的并发度为N
- 分散账单计算时机，避免同一时刻大量的计算，减轻DB压力



高性能设计-读性能



VS



冗余

- 引入缓存
- 读写分离
- 增加从库
- 数据不一致风险

- 读写不分离
- 主库压力大
- 读场景设计

高性能设计-读性能-场景一

场景

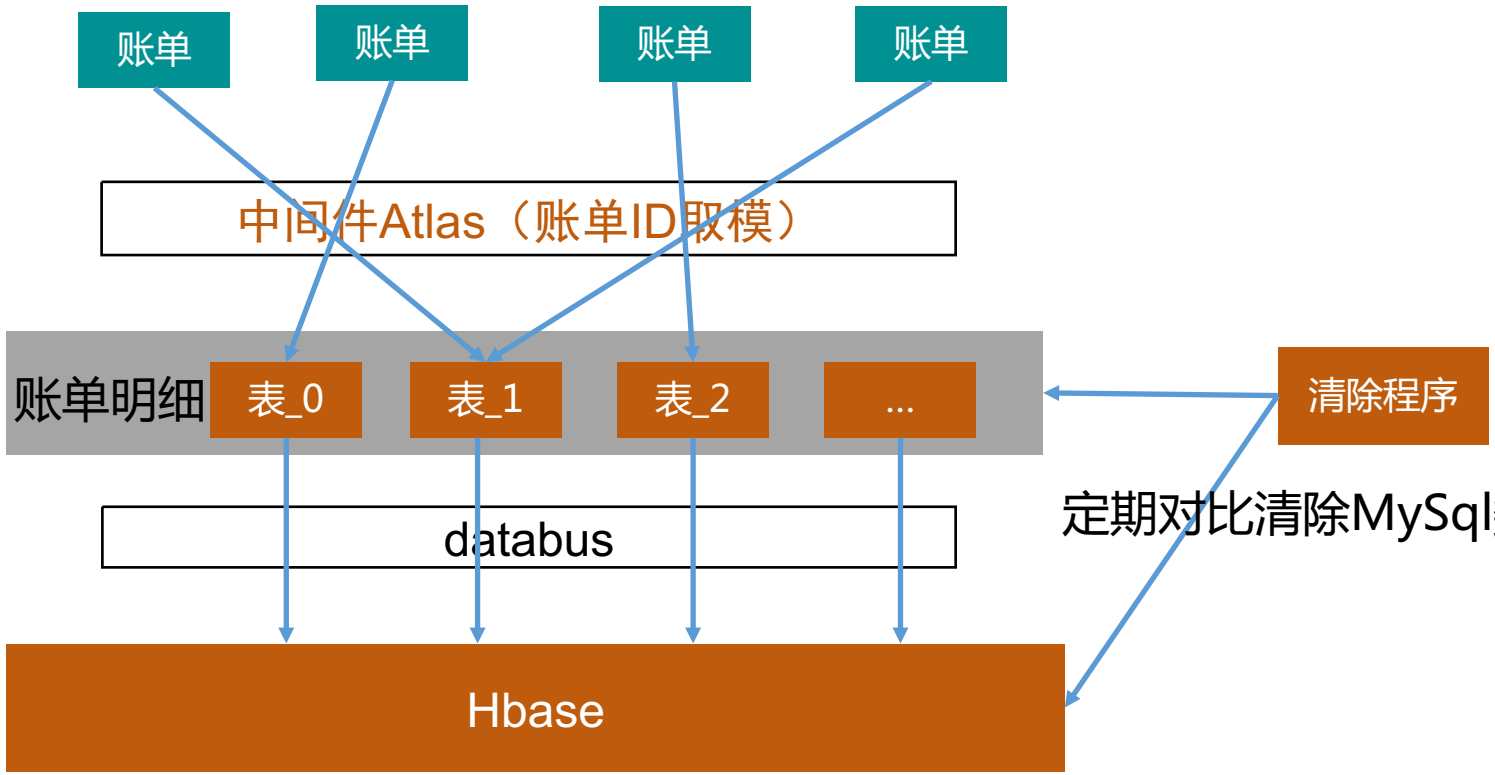
- 计算账单金额
- 亿级数据

核心语句

- select sum(金额) from 账单明细表 where 账单ID = X

高性能设计-读性能-场景一

- 账单明细分表
- 预估容量避免迁移
- 减少单表数据量
- 提高SUM性能



定期对比清除MySql数据释放存储空间

- 支持水平扩展
- 写多、读少符合Hbase特性

高性能设计-读性能-场景二

场景

- 账单数据完整性检测
- 亿级数据

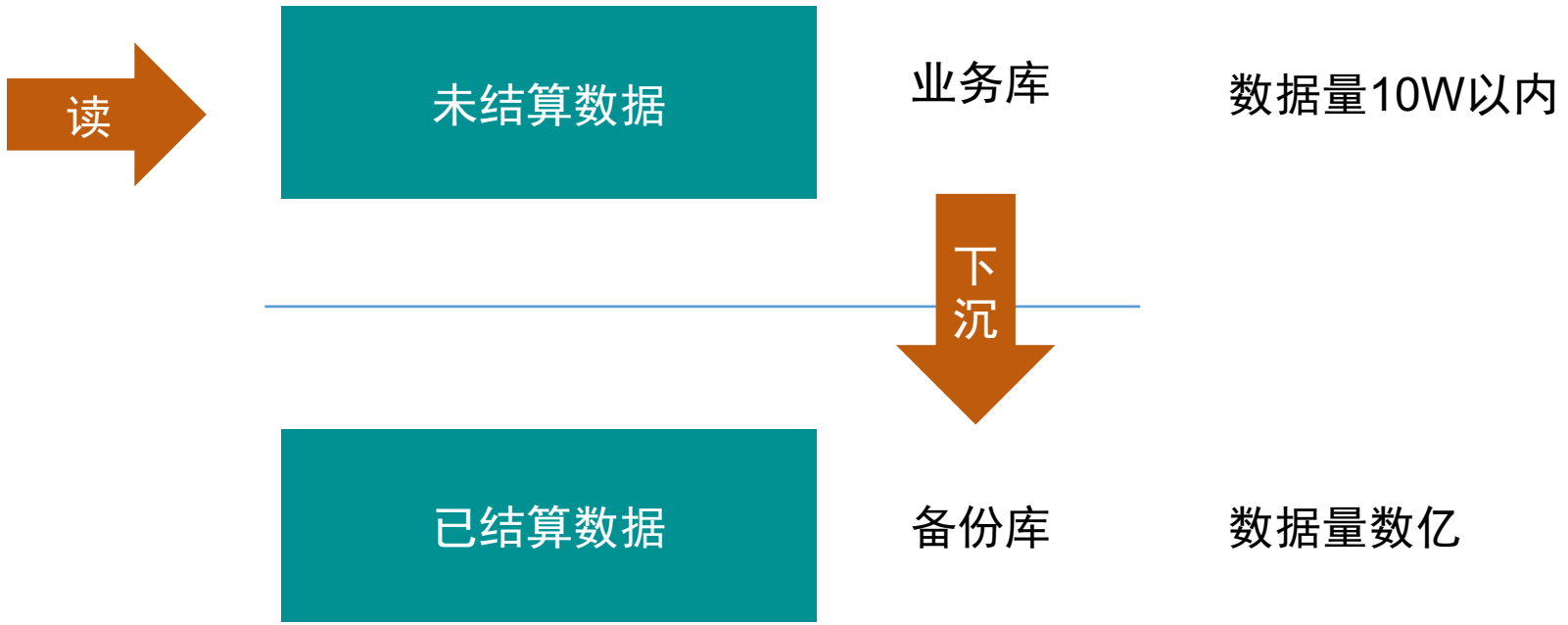
核心语句

- select count(1) from 流水表 where 业务线=酒店 and 商家ID=1 and 流水时间 > X and 流水时间 < Y and 结算状态 = 未结算 。。。;

高性能设计-读性能-场景二

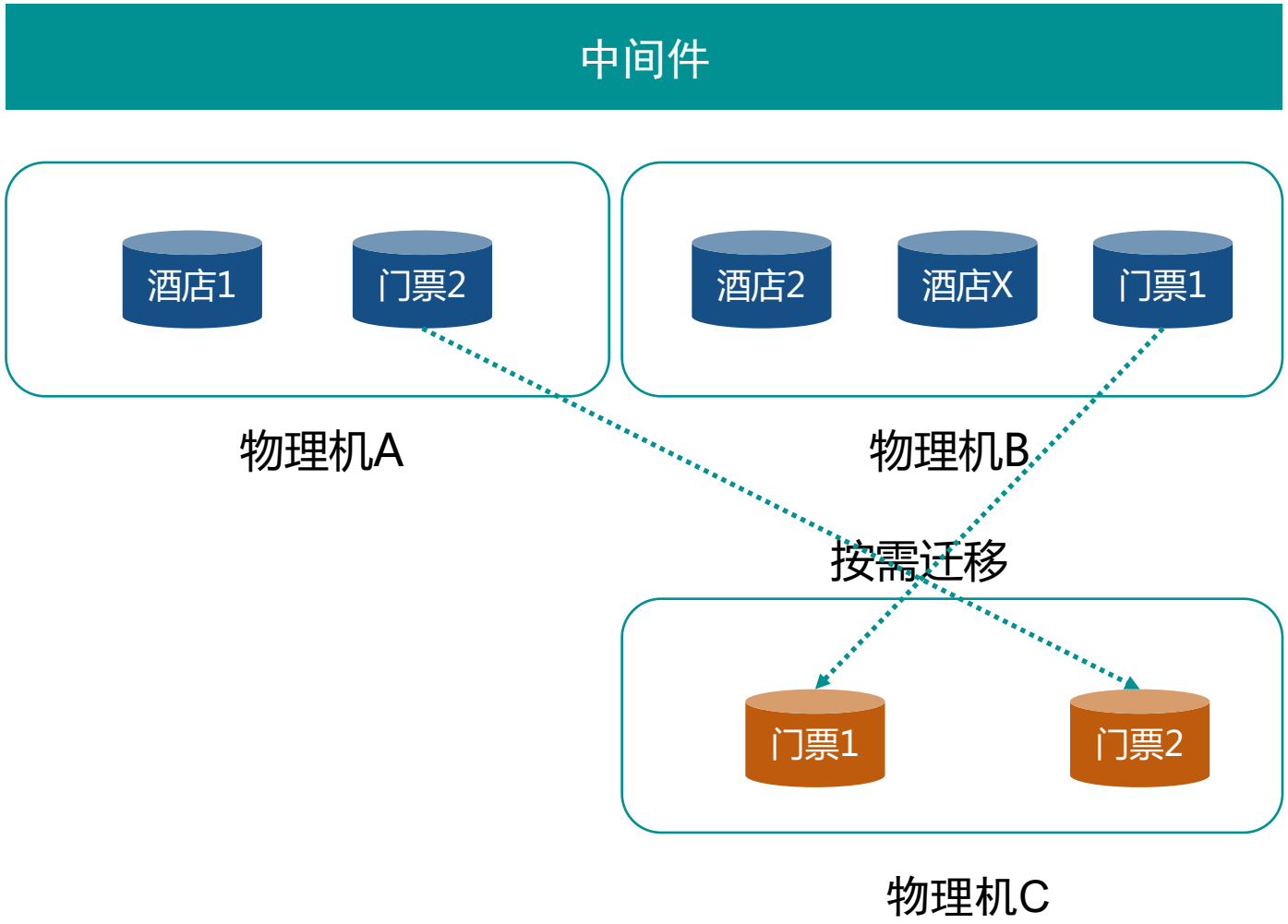
策略

- 数据冷热隔离
- 减少数据量



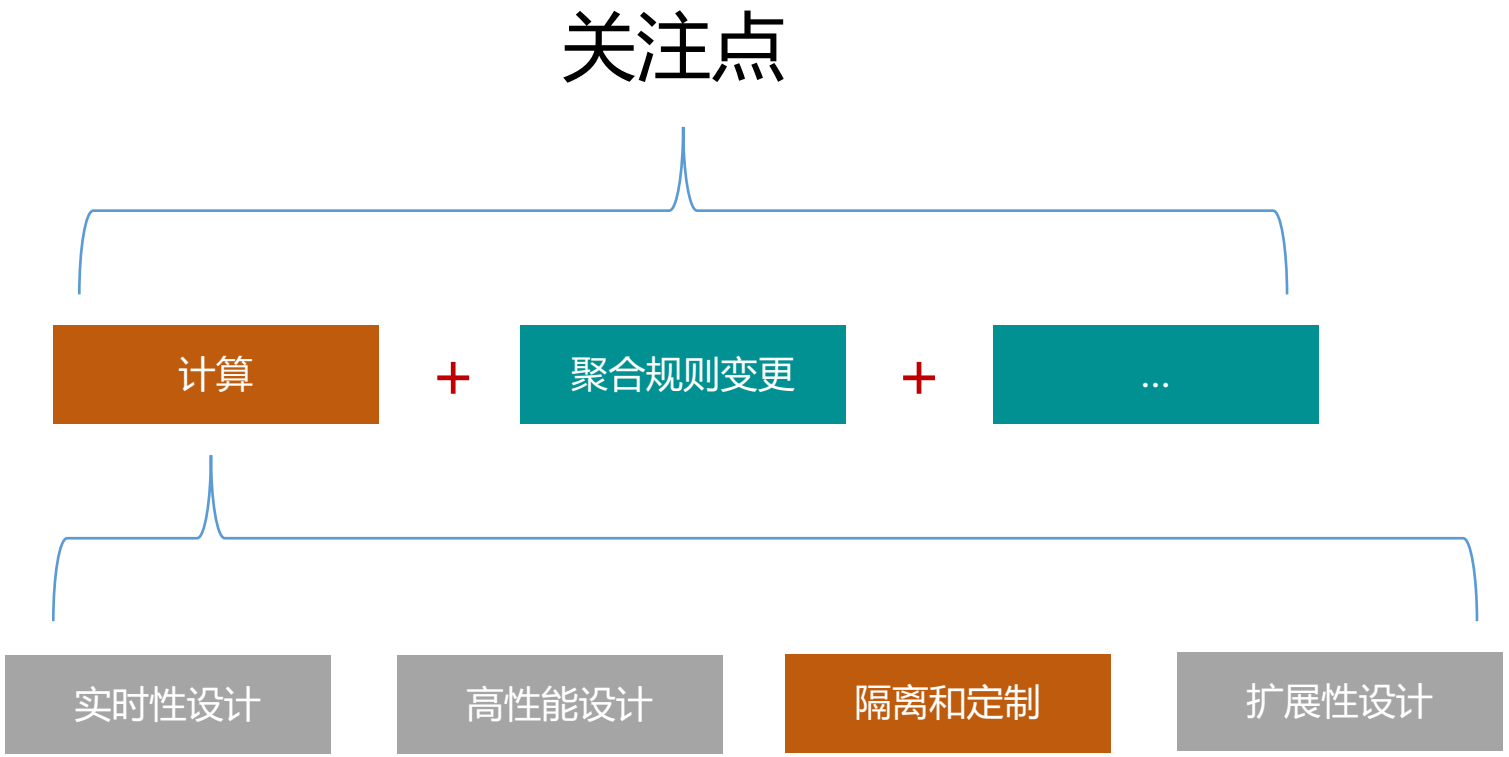
高性能设计- 读、写未来规划

可靠性、扩展性、稳定性



- 数据按业务，商家分库
- 数据库混合部署到不同物理机
- 多库、少物理机方式部署，按需扩容迁移
- 故障影响部分商家，部分业务

对账平台实践关注点

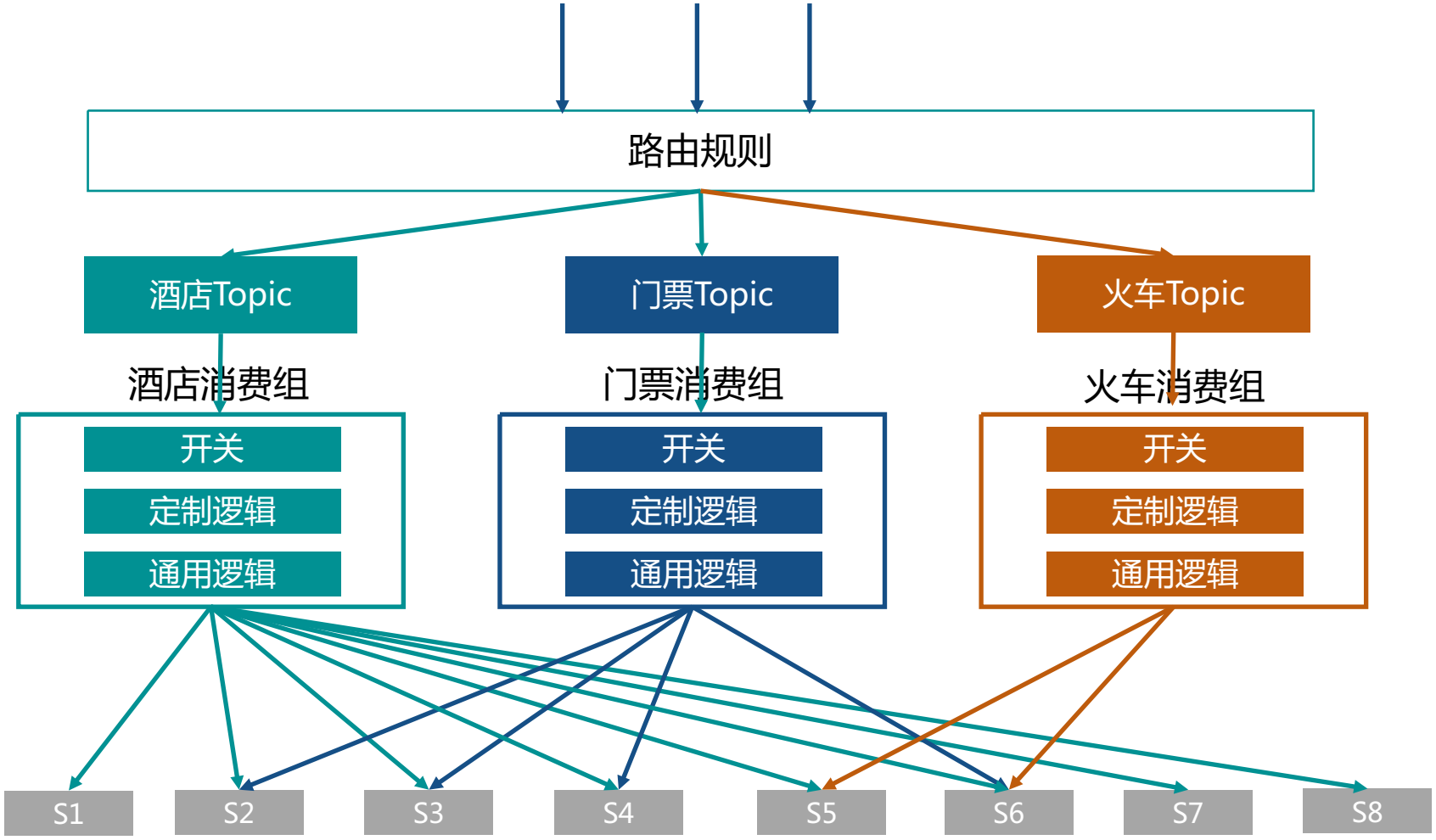


隔离&定制性设计

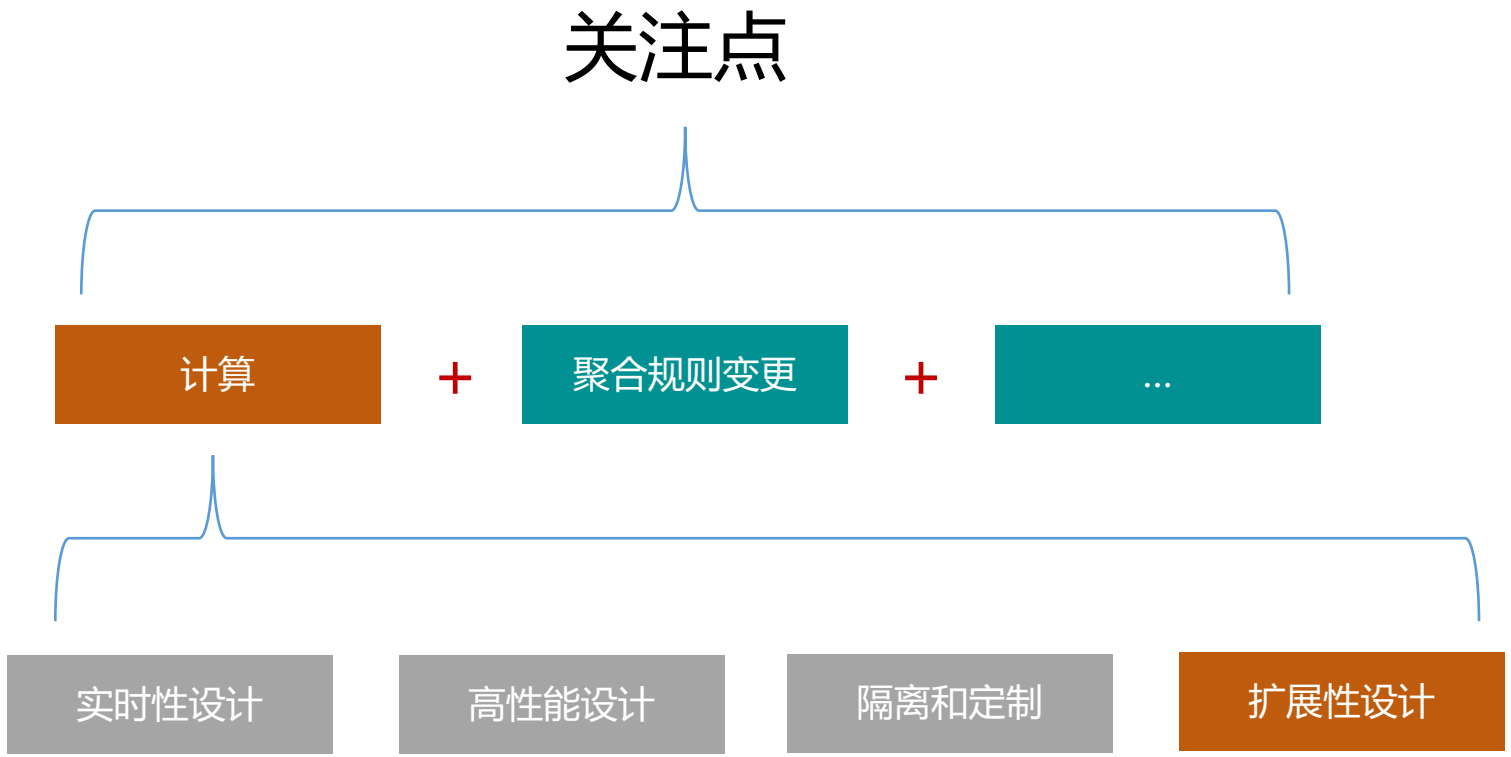
topic隔离
按需设置partition数量

消费组隔离
允许逻辑定制和隔离
按需设置消费线程数

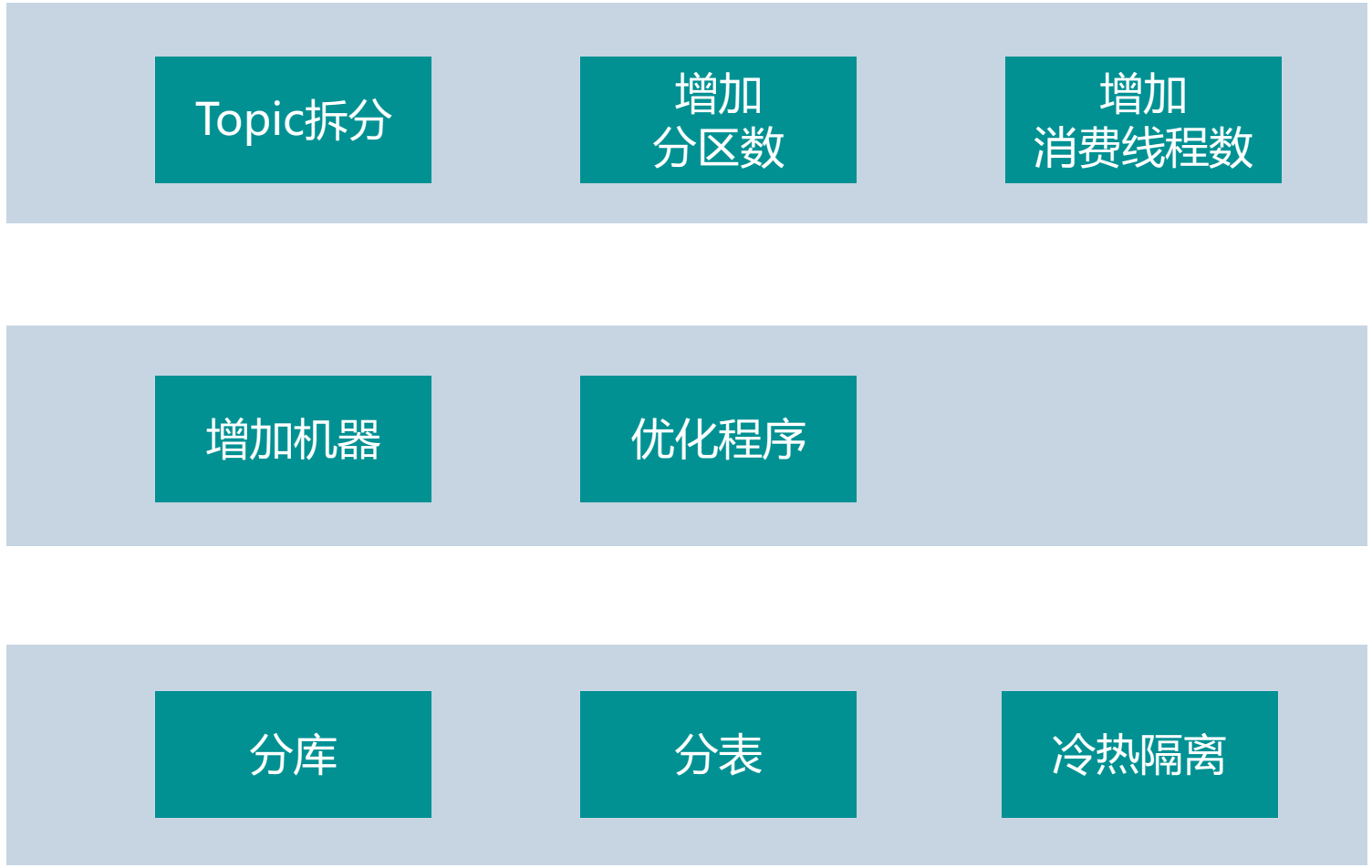
server集群公用
根据topic和partition划分资源



对账平台实践关注点



扩展性设计


Mafka
Topic拆分
增加
分区数
增加
消费线程数
数据分片、增加并发度
应用
增加机器
优化程序
增加处理能力
DB
分库
分表
冷热隔离
增加读、写能力

高性能账单设计-效果

- 数据收集延迟2天->实时接入，延迟毫秒级别
- 结算延迟5-15天->提前看账单，交易立即体现，提升商家体验，延迟秒级
- 17条业务线相互隔离，0交叉影响

Q&A