第14章 高级数据库技术

清华大学计算机系 李国良

liguoliang@tsinghua.edu.cn
http://dbgroup.cs.tsinghua.edu.cn/ligl

背景

大数据量超高并发	分布式数据库	分布式事务2PC+Paxos、分布式查询优化
海量数据分析决策	OLAP数据库	列存、向量化、MPP、物化视图
交易分析混合负载	HTAP数据库	行列混存、列存选择、行列混合优化器
实时交易超低时延	内存数据库	行级组织、内存索引、内存并发控制
云上弹性伸缩调度	云原生数据库	计算/缓存/存储解耦、日志即数据、多租
新型硬件算力发挥	新硬件数据库	NUMA并发控制、GPU并行处理、NVM组织、RDMA分布协同、CXL共享缓存、FGPA压缩
海量非结构化数据	NoSQL数据库	CAP、BASE、LSM-tree、多模处理
智能时代数智融合	AI原生数据库	ABO(基数/重写/计划)、AI4DB、DB4AI

目录

01 分布式数据库

02 OLAP数据库

03 HTAP数据库

04 内存数据库

05 云原生数据库

06 数据库与新硬件

07 NoSQL数据库

08 AI与数据库

01

分布式数据库

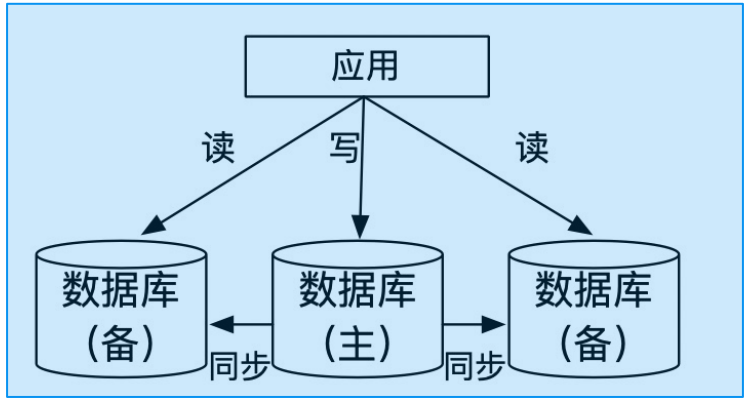
分布式数据库

- **背景**
  - 大数据：数据量超出单台机器存储容量与计算能力的上限
  - 高并发：海量用户并发请求
  - 高可用：许多场景下要求系统不会因为少量故障导致服务中断
- **分布式数据库：**分布在多个计算机节点的、逻辑相互关联的数据集合，计算机节点之间通过网络连接进行通信，作为一个整体对外提供数据存储与查询等服务。
  - 高并发性：多节点可同时响应客户端
  - 高可扩展性：节点数量可高效扩展
  - 高可用性：即使部分节点故障，系统整体也可正常运行

分布式数据库架构

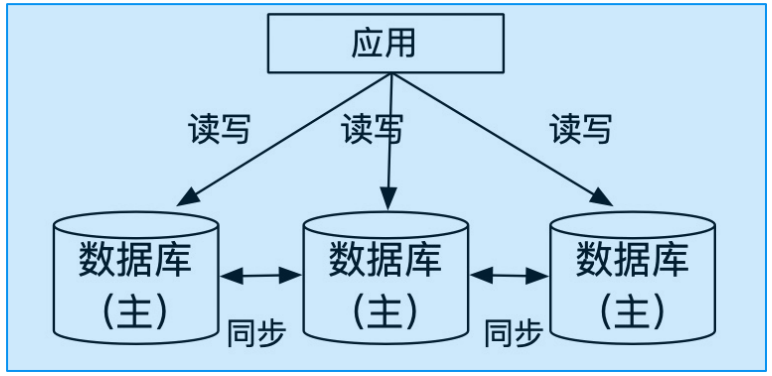
① 一主多备，一写多读

- 主节点读写，备节点只读



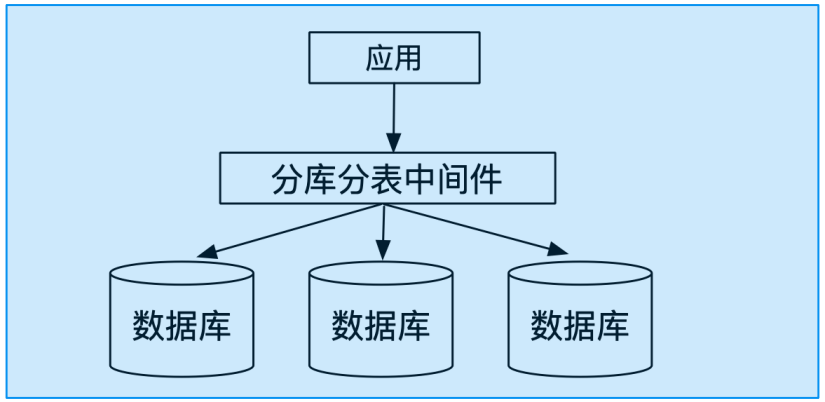
② 多主多写

- 各节点可读可写



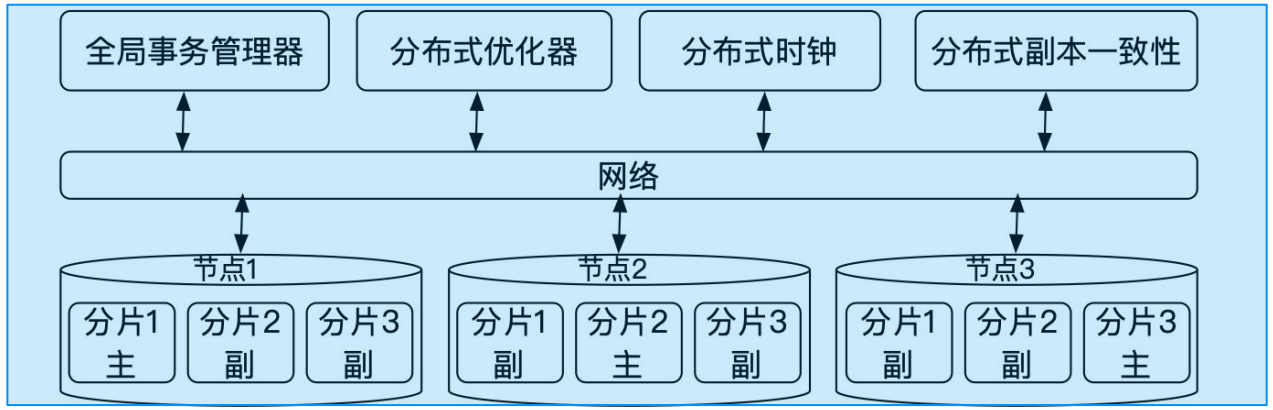
③ 分库分表中间件

- 搭建于多台单机数据库之上，中间件负责SQL转发和结果汇聚



④ 原生分布式数据库

- 节点间无共享架构，shared-nothing，分片间多写多读



分布式架构对比

	多写	强一致性	扩展性	读性能	写性能
一主多备，一写多读	N	Y	差	中	差
多写多读	Y	Y	中	中	中
分布式中间件	Y	N	好	中	好
原生分布式	Y	Y	好	好	好

- 原生分布式：shared nothing，数据分片，计算移动
- 多写多读集群式：shared storage，页面划分，数据移动
- 分布式中间件：不支持强一致，需要中间件分发和汇总

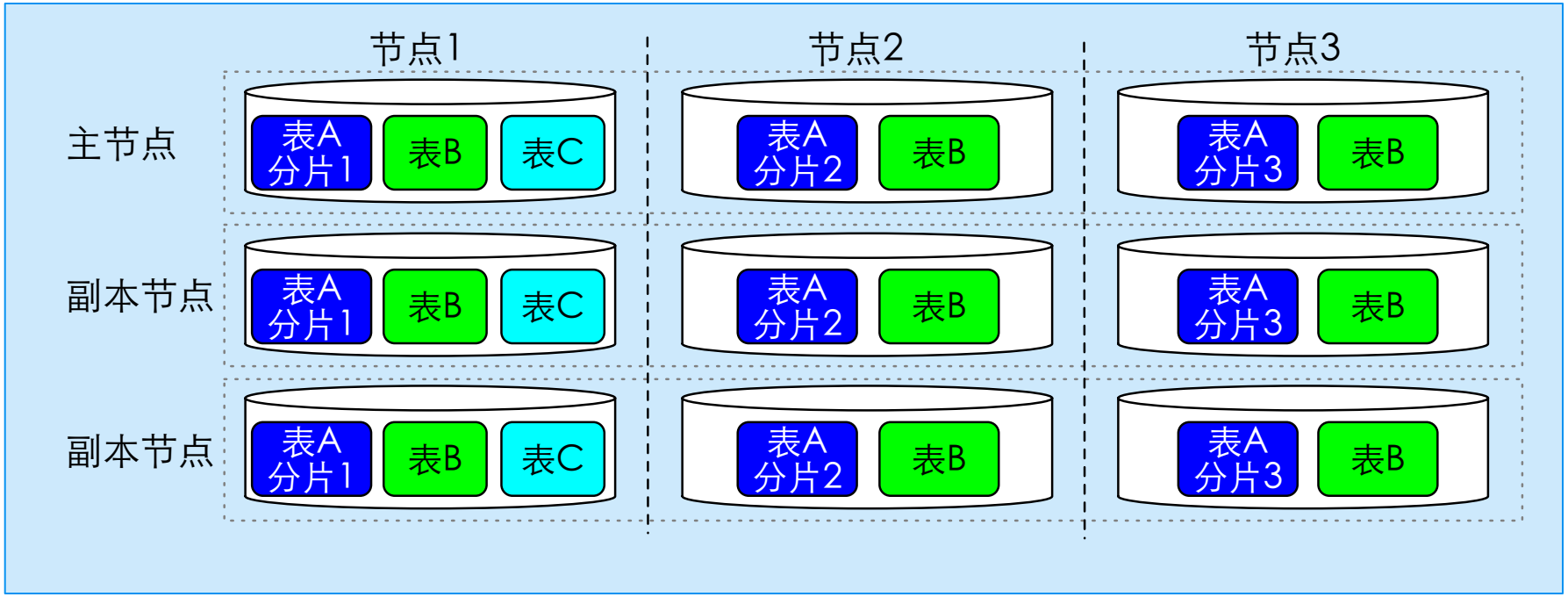
分布式数据库相关技术

- **分布式数据存储**
- **分布式事务**
  - 分布式原子提交
  - 并发控制
  - 分布式副本一致性
  - 分布式时钟
- **分布式查询优化**

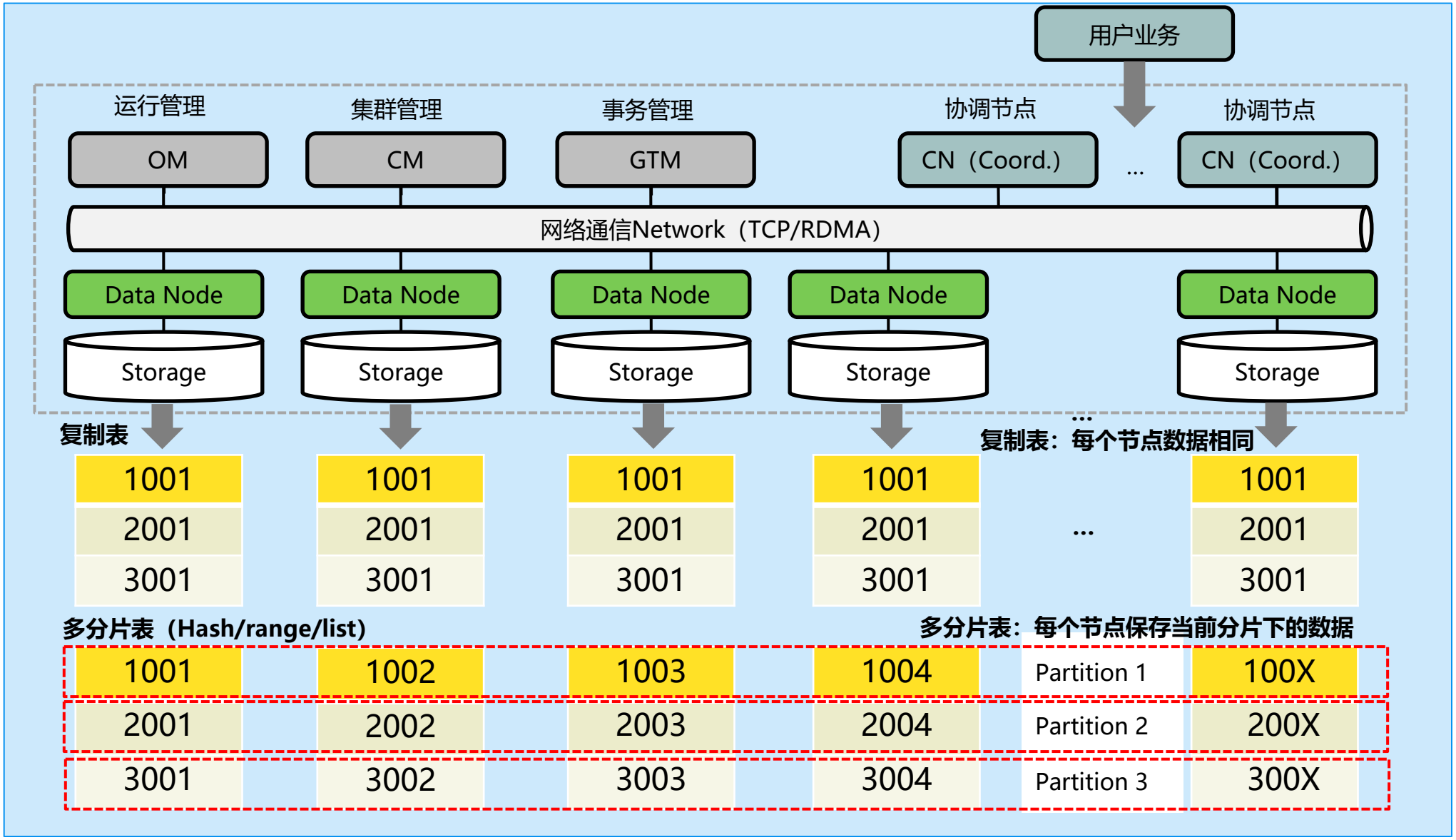
分布式数据存储

➢ 数据分布式存储方式

- 单节点表：只存在单个节点的表，例如表C
- 复制表：每个节点都存储副本的表，例如表B
- 多分片表：数据划分不同分片，每个节点存储一个分片，例如表A



分布式数据存储


用户业务
运行管理
集群管理
事务管理
协调节点
协调节点
OM
CM
GTM
CN（Coord.）
...
CN（Coord.）
网络通信Network（TCP/RDMA）
Data Node
Data Node
Data Node
Data Node
Data Node
Storage
Storage
Storage
Storage
Storage
复制表
...
复制表：每个节点数据相同
1001
2001
3001
1001
2001
3001
1001
2001
3001
1001
2001
3001
...
1001
2001
3001
多分片表（Hash/range/list）
多分片表：每个节点保存当前分片下的数据
1001
1002
1003
1004
Partition 1
100X
2001
2002
2003
2004
Partition 2
200X
3001
3002
3003
3004
Partition 3
300X

分布式数据存储

- **单节点表 vs 复制表 vs 多分片表**

	优势	劣势	适应场景
单节点表	节省空间，避免节点间数据同步	扩展性差，单点瓶颈	小表，写多读少
复制表	读取速度快	扩展性差，空间浪费， 节点间同步开销大	小表，读多写少
多分片表	分布并行，扩展性好	跨节点事务	大表

分布式数据存储

➢ 数据分片

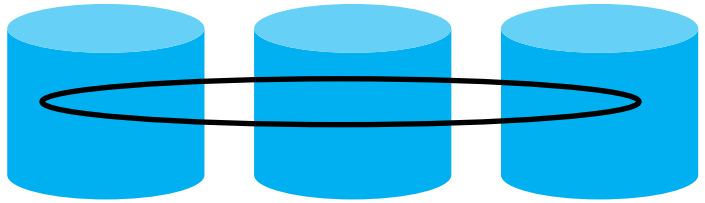
- 数据切分为多个分片，存储在不同节点，每个节点存储1个或多个分片
- 分片类型
  - ① 水平分片：按行切分，不同的行存储于不同节点
  - ② 垂直分片：按列切分，不同的列存储于不同节点
- 水平分片算法
  - ① 哈希分片Hash：将每条记录在分布列的哈希值相同的数据记录分到相同节点
  - ② 范围分片Range：根据每条记录在分布列的值进行范围划分，将同一范围内的数据记录划分到相同节点，例如按照学号
  - ③ 列表分片List：根据每条记录在分布列的值进行列表划分，即将该分布列的所有值进行分组，落在相同分组的数据记录放到同一个节点，例如按照省份

分布式数据存储

➢ 多副本

- 每个数据分片以多副本形式存储在多个节点中
- 防止由于单个节点故障造成数据损坏或丢失的情况
- 提高了系统可用性与吞吐量

多副本一致性



➢ 副本更新的一致性问题：

- 当数据更新时，由于其只更新了一个副本而将更新数据同步至其他副本的过程往往存在一定延时，会导致同时读取不同副本的数据时可能会得到不同的值
- 一致性协议：
  - 多数派：Paxos、Raft
  - Quorum：R+W>N

分布式事务

- **事务的ACID特性**
  - 原子性(A)，一致性(C)，隔离性(I)，持久性(D)
- **分布式事务**
  - 持久性：实现与单机数据库的实现方法基本相同
  - 一致性：
    - 副本一致性通过Paxos、Raft等一致性协议来解决
  - 原子性：分布式原子提交协议
  - 隔离性：并发控制，分布式时间戳

分布式事务

- **分布式原子提交 vs 单节点提交**
  - 单节点只有一个参与方；分布式有多个参与方
  - 事务在所有节点中的操作要么全部执行要么全部不执行
- **分布式原子提交方法**
  - 两阶段提交
  - 三阶段提交
  - Calvin
- **参与事务处理的节点**
  - 事务协调者：与参与者通信，协调与调度事务
  - 事务参与者：执行事务在单节点中的操作

两阶段提交

➢ 准备阶段

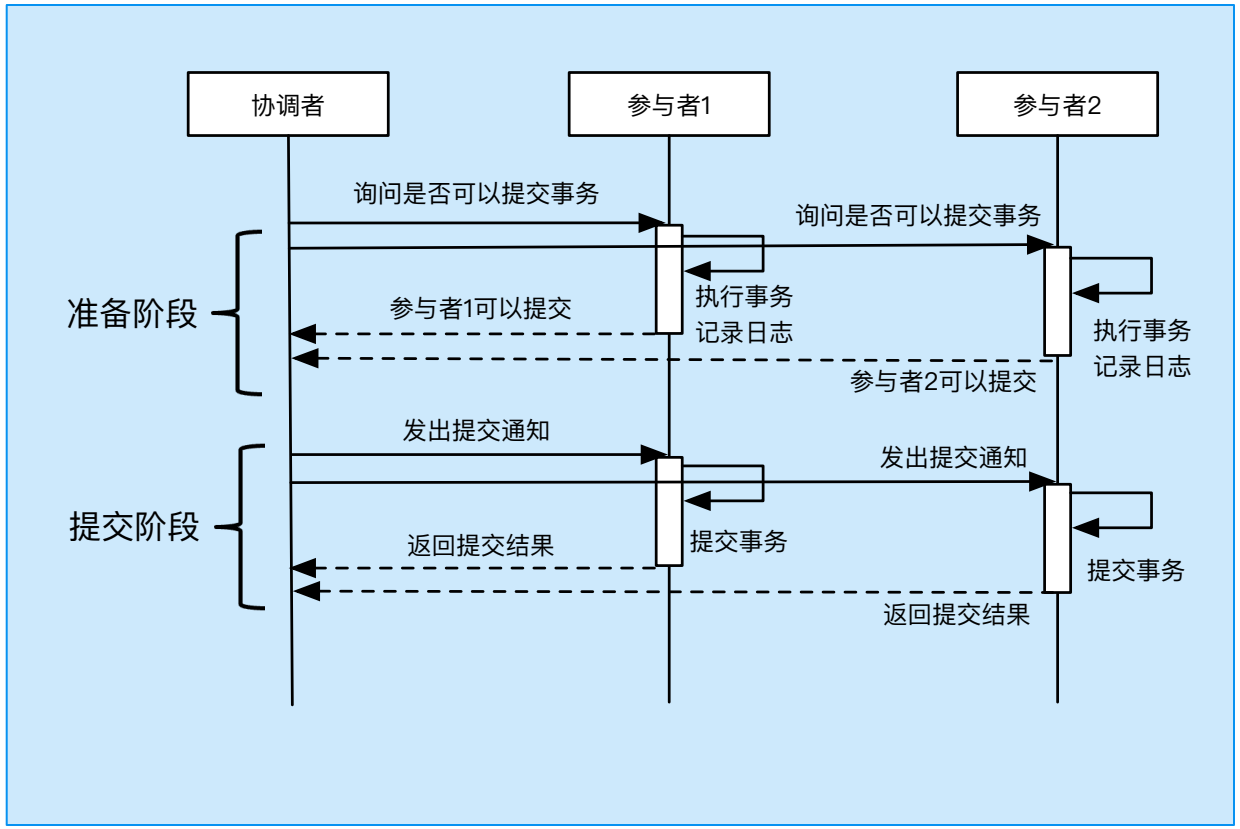
- 协调者询问参与者是否可以提交
- 参与者判断自身是否可以提交，并向协调者发送是否可以提交的消息
- 协调者执行事务操作，记录日志

➢ 提交阶段

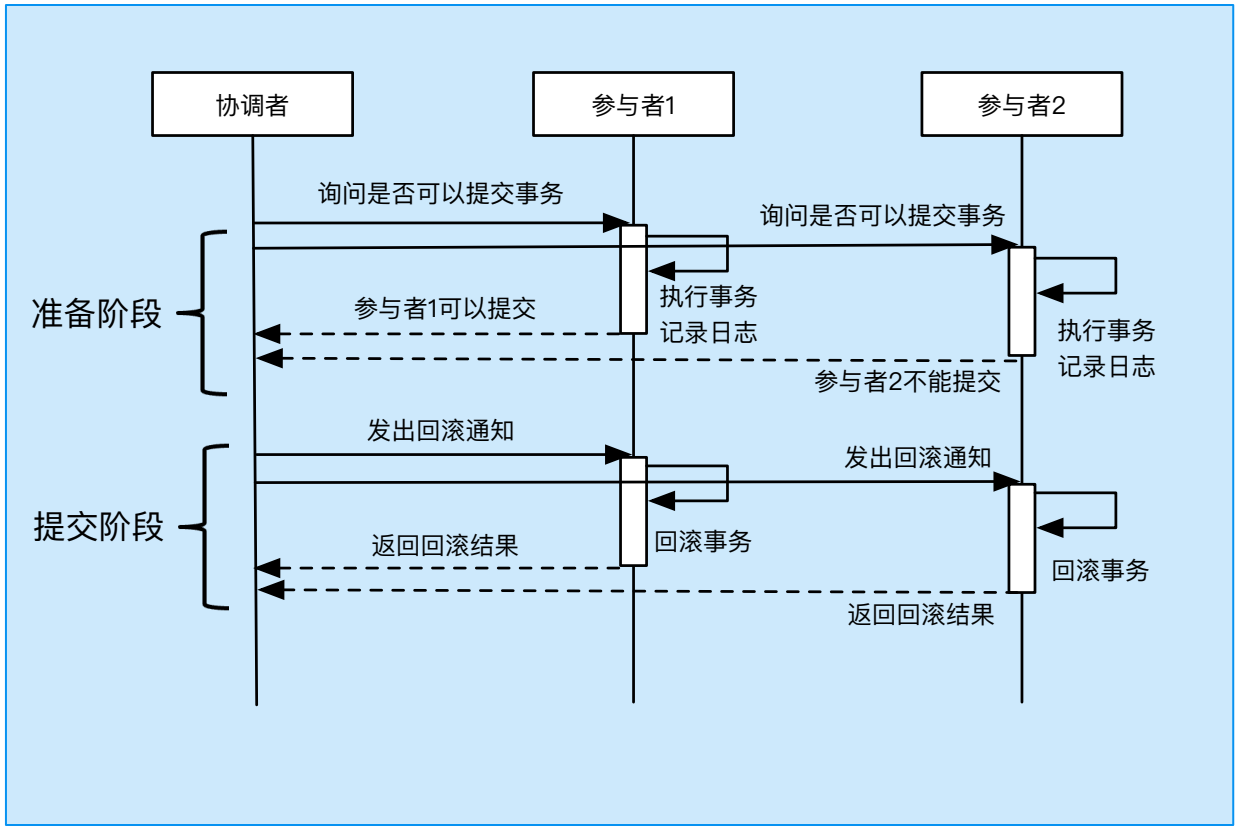
- 如果所有参与者都可以提交，则协调者发送提交指令
- 如果任何一个参与者无法提交，则协调者发送回滚指令
- 参与者收到指令后进行提交或回滚操作

两阶段提交

提交成功



提交失败



两阶段提交问题

① 性能较差

- 准备阶段协调者需要等待所有参与者回复才能进入提交阶段
- 准备阶段存在加锁操作，影响系统性能

② 单点故障

- 协调者发生故障时会影响所有参与者
- 若提交阶段协调者故障，所有参与者都会阻塞

③ 数据不一致风险

- 提交阶段协调者发送提交指令后出现网络故障
- 部分参与者收到提交指令而其他参与者没有收到提交指令

三阶段提交

➢ 解决两阶段提交的单点故障问题

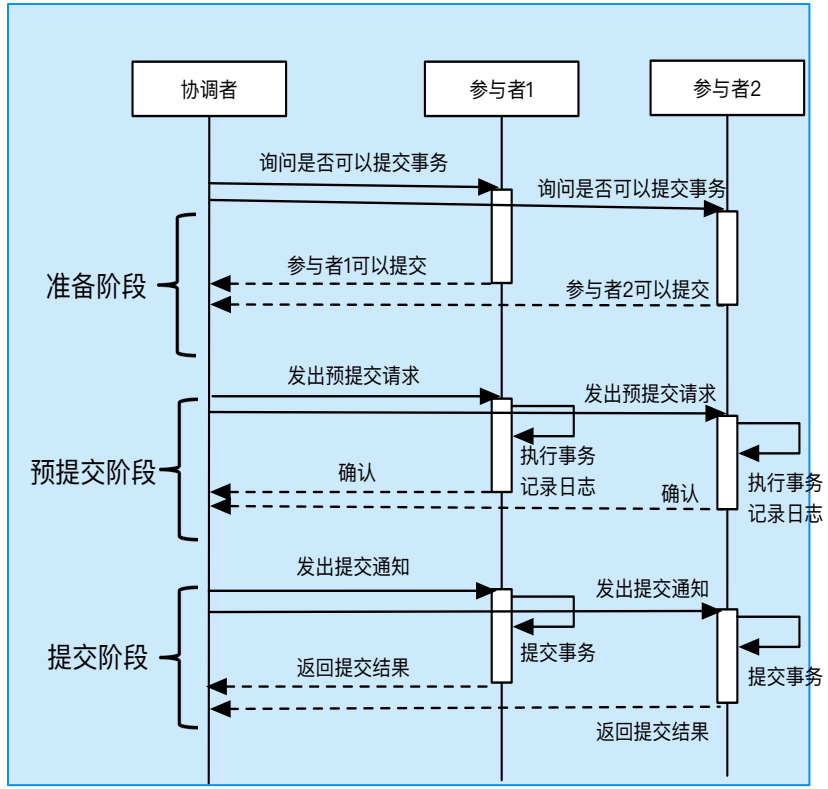
- 协调者故障时，参与者会选举出一个新的协调者

➢ 预提交阶段

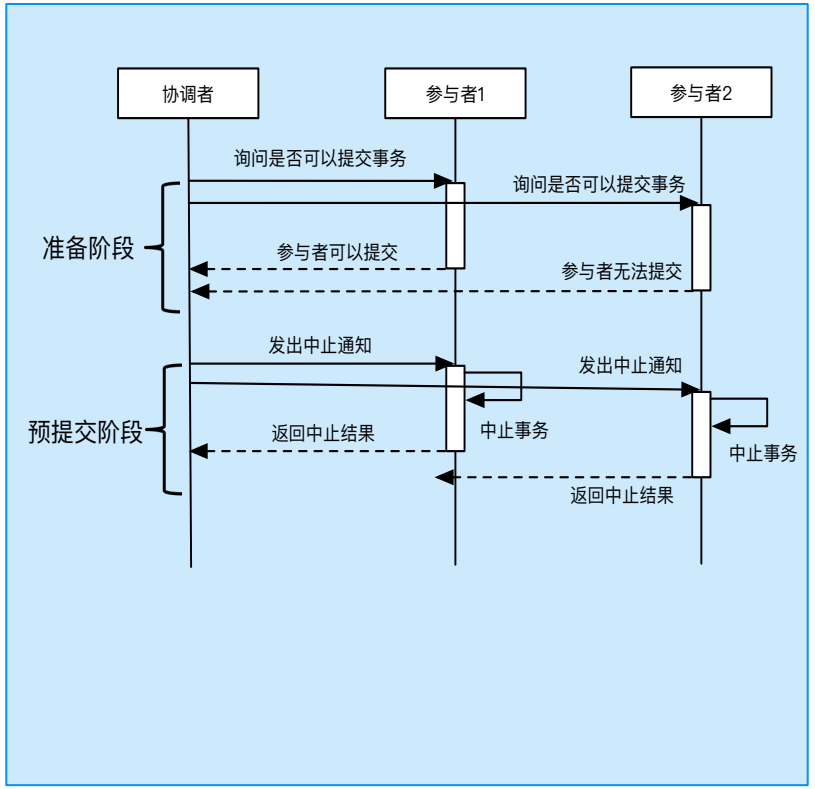
- 在准备阶段和提交阶段之间添加预提交阶段
  - 准备阶段不加锁，提升性能
- 协调者将准备阶段的投票结果发送给所有参与者
- 协调者故障后，新的协调者根据参与者记录的投票结果判断提交还是中止事务
  - 解决协调者单点故障

三阶段提交

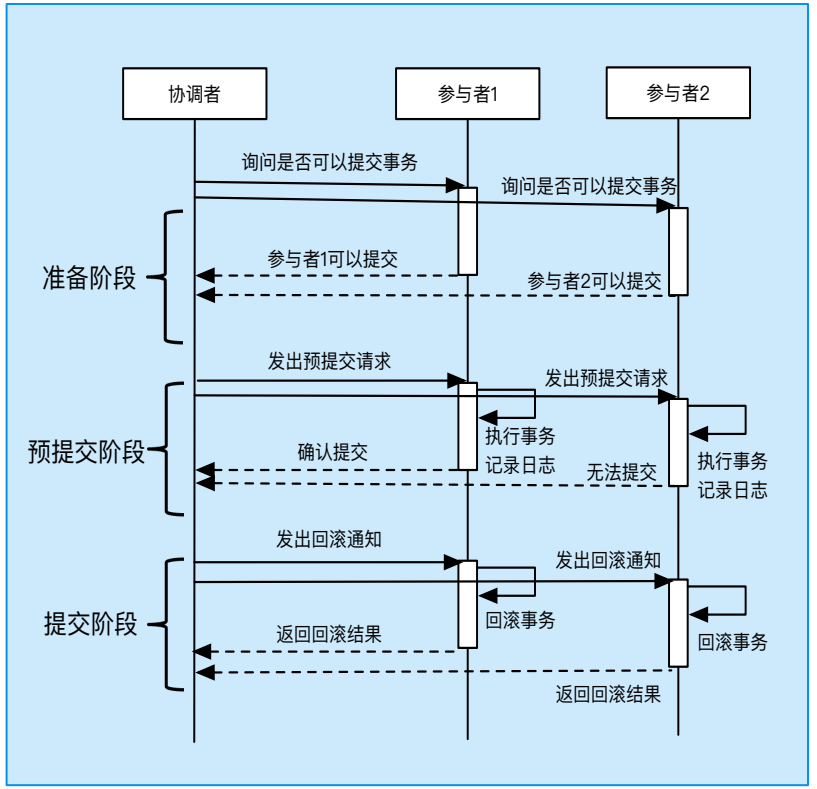
提交成功



第一阶段失败



第二阶段失败

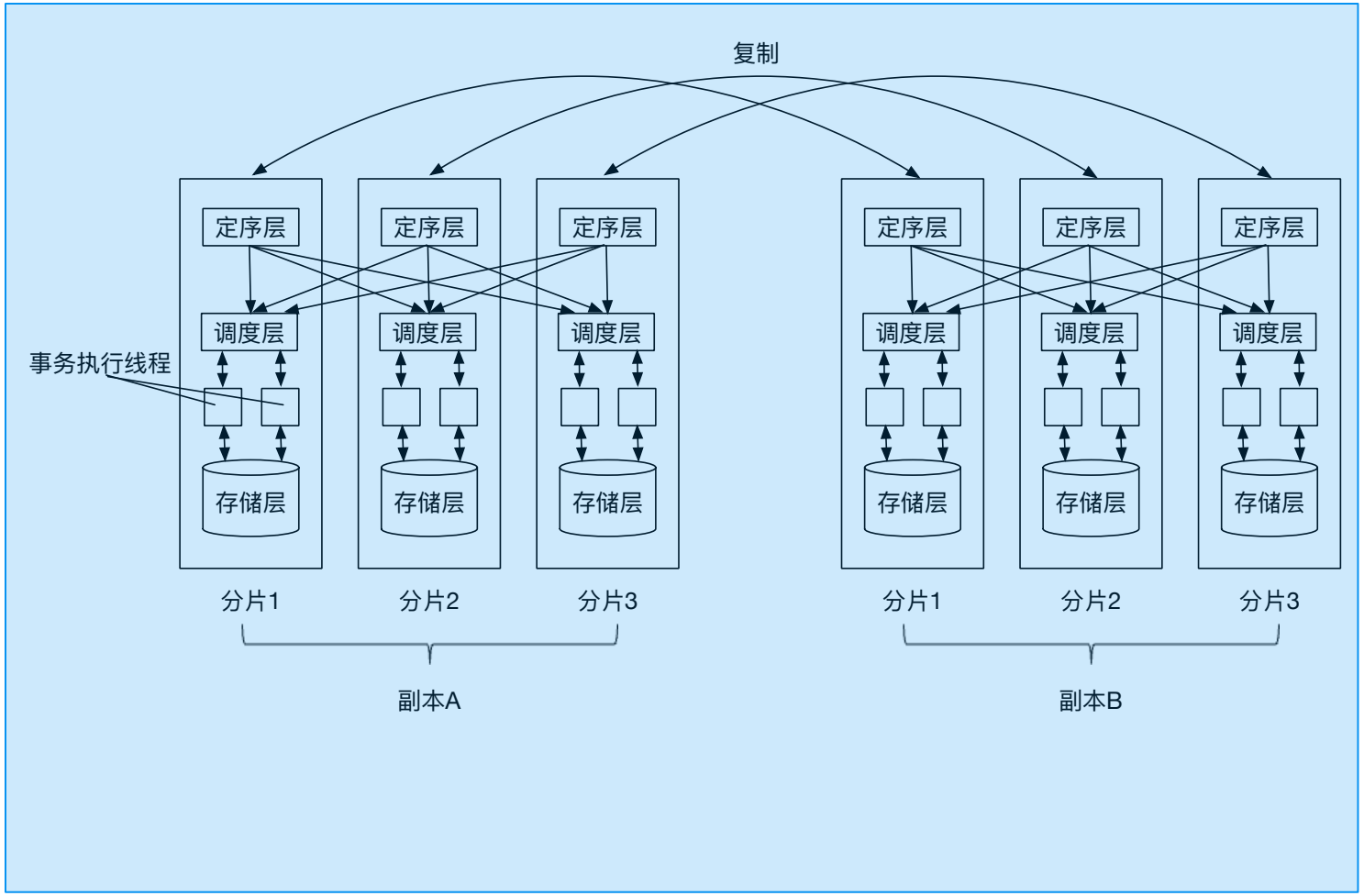


三阶段提交

- **解决了两阶段提交单点故障带来的阻塞问题**
- **解决了两阶段提交准备阶段存在加锁操作导致影响系统性能的问题**
- **通信代价高，性能差**
  - 事务提交需要协调者和参与者进行三轮通信
- **网络分区情况下会发生数据不一致**
  - 预提交阶段发生网络分区
  - 收到预提交请求的节点和协调者位于一个分区，未收到预提交请求的节点位于另一分区
  - 两个分区分别进行事务提交和事务中止，造成数据不一致
- **网络分区情况下会发生数据不一致 → Paxos多副本**

Calvin

- **确定性事务**
  - 各节点事务执行顺序保持一致
- **降低节点之间协调的通信开销**
  - 无需像两阶段提交和三阶段提交一样进行多轮通信
  - 提高事务吞吐量
- **要求事务运行之前确定读取集和写入集**



分布式并发控制

➢ 基于锁的方法

- 集中式锁管理器：实现简单，但锁管理器容易成为系统的性能瓶颈，且存在单点故障
- 分布式锁管理器：解决了单点故障和瓶颈，但死锁检测较为复杂

➢ 基于时间戳的方法

- 集中式时间戳：授时中心成为性能瓶颈，且单节点难以实现高可用
- 分布式时间戳：高可用，但时钟顺序维护较为复杂

分布式时钟

- **时钟的重要性**
  - 多版本并发控制中的时间戳
- **分布式系统中的时间问题**
  - 每台机器有自己的时钟，如何保证不同机器的时间一致
  - 时间同步存在网络延迟，如何判断不同节点上发生的事件的先后顺序
- **物理时钟**
  - 精度有限，存在时钟漂移

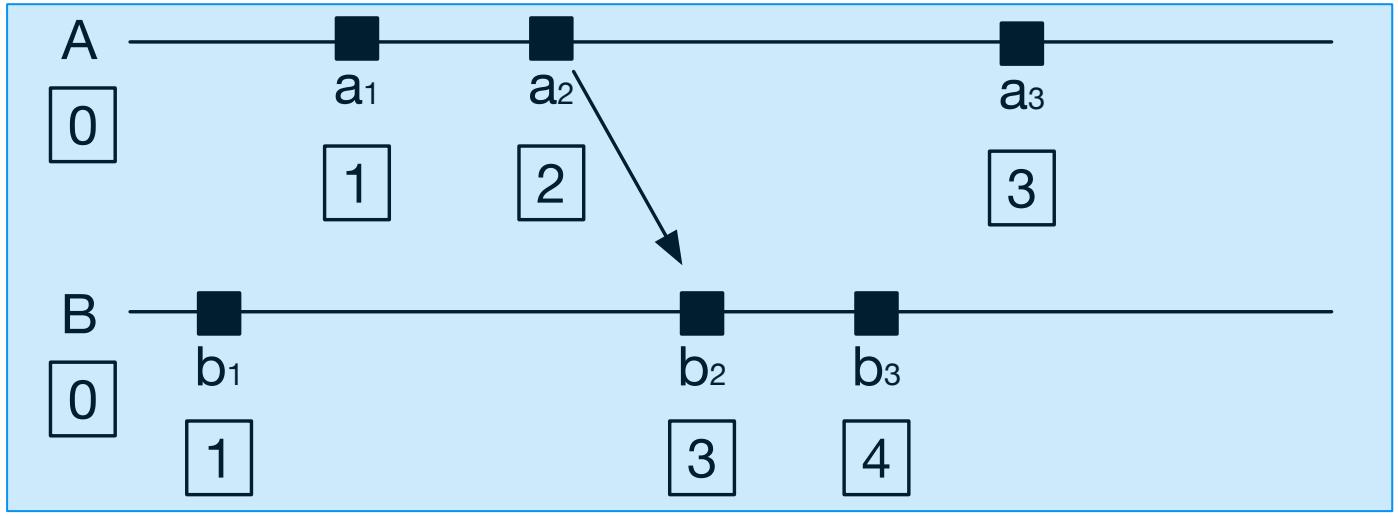
分布式时钟

逻辑时钟

- 每个节点内部维护一个本地逻辑时钟，初始值为0。
- 节点内部每发生一个事件，如事务开始，就将本地逻辑时钟的值加一。
- 如果节点A向节点B发送了一条消息，则节点A在发送消息时附带上本地逻辑时钟的值，节点B收到消息后，比较自身的本地时钟和节点A发送的本地时钟，将本地时钟更新为二者的较大值加一

特点

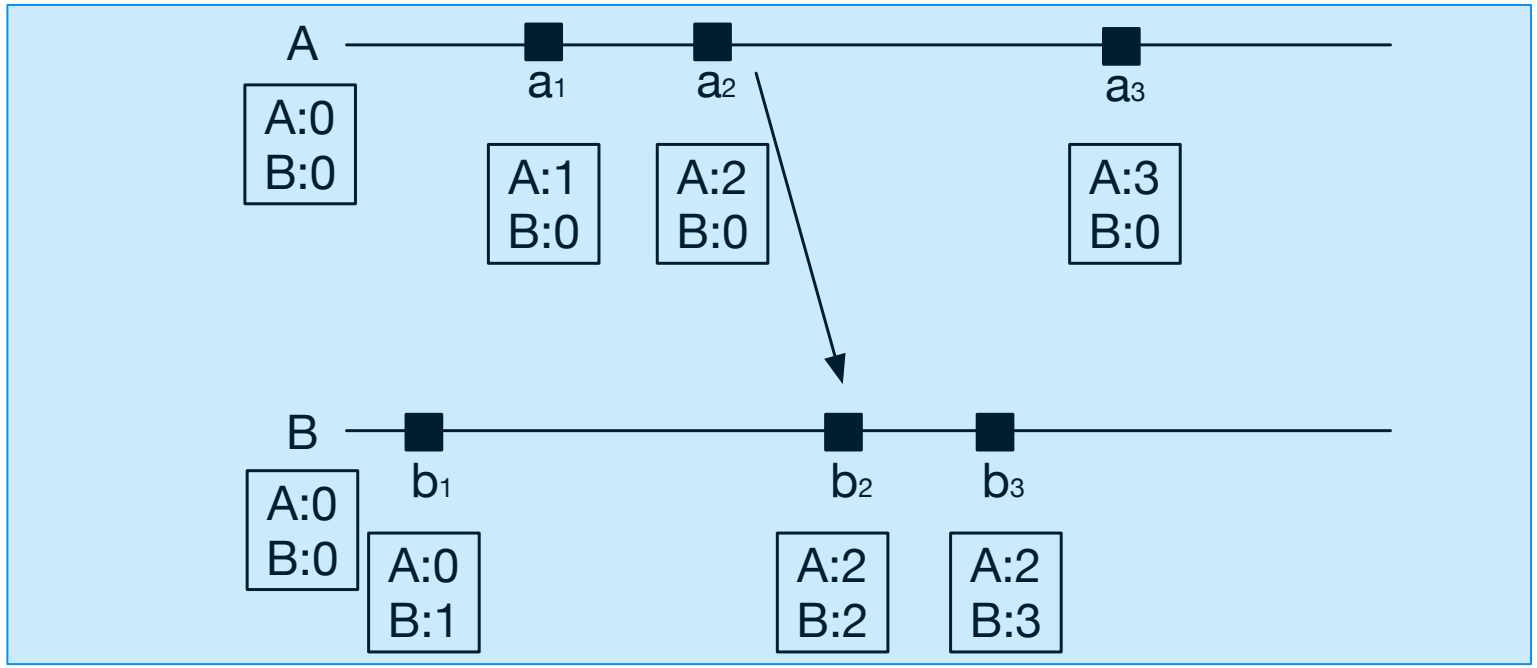
- 只是偏序，不是全序，某些场景无法判断两个事务先后，例如两个节点之间没有消息传递



分布式时钟

➢ 向量时钟

- 每个节点不仅存储本地时钟的值，还存储了其他节点的逻辑时钟。
- 所有节点的逻辑时钟值组成一个向量，向量维度与系统中的节点数目相同。
- 对于事件A和事件B对应的时钟V(A)和V(B)，如果V(A) < V(B)，即向量V(A)的每个分量都小于向量V(B)，则可以确定事件A发生在事件B之前。



分布式时钟

➢ 混合逻辑时钟

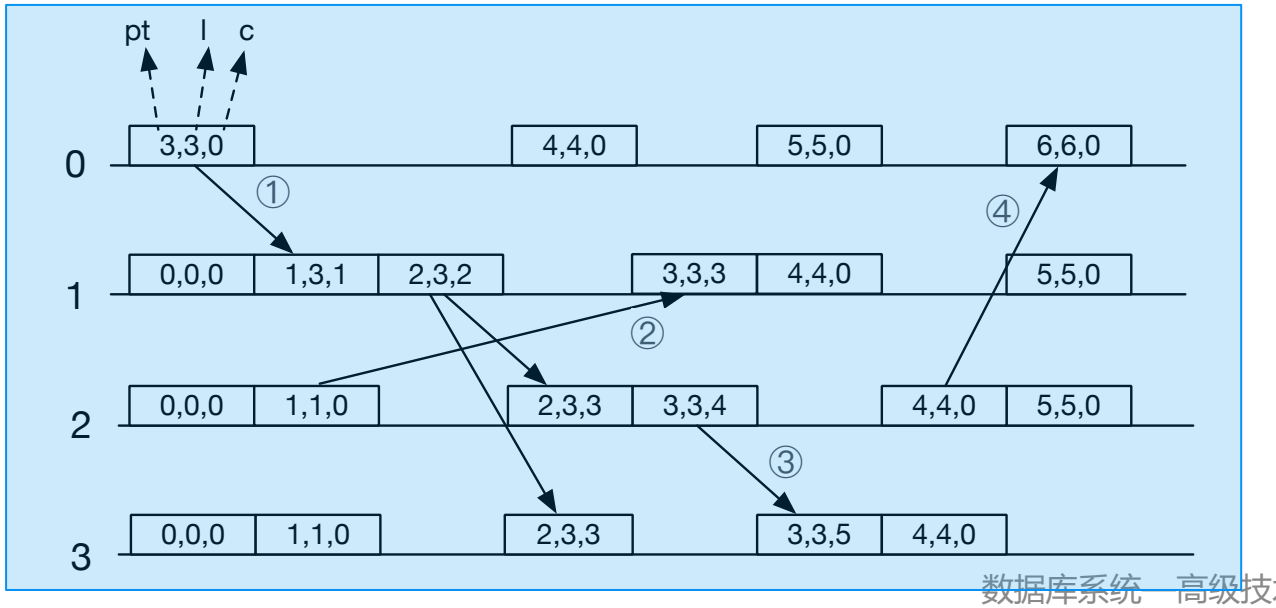
- 包含物理时钟和逻辑时钟，既可以如逻辑时钟一样表示事件之间的因果关系，又保证了混合逻辑时钟值与物理时钟尽可能接近

① 如果事件e发生在事件f之前，那么e的时间戳小于f的时间戳；

② 时间戳的存储空间大小是固定的，空间消耗不会随着系统节点数的增加而增大；

③ 时间戳值存在上界，不会无限增长；

④ 时间戳的值与物理时钟接近，二者的差值是有界的。



给定事件e

- e.pt为事件e发生的物理时间
- e.l表示事件e发生时所感知到的物理时钟的最大值
- e.c用于记录事件之间的因果关系

TrueTime

- 逻辑时钟和向量时钟定义了事件之间的偏序关系而非全序关系，仍存在部分无法判断先后顺序的事件，如两个节点之间没有发生过任何通信，则这两个节点上发生的事件无法进行先后顺序的比较
- 通过硬件（原子钟+GPS时钟）来解决时钟同步问题。每个数据中心都具有精确的原子钟，同时还可以得到GPS卫星提供的时钟信息，两种时钟均具有较高的准确性和可靠性。

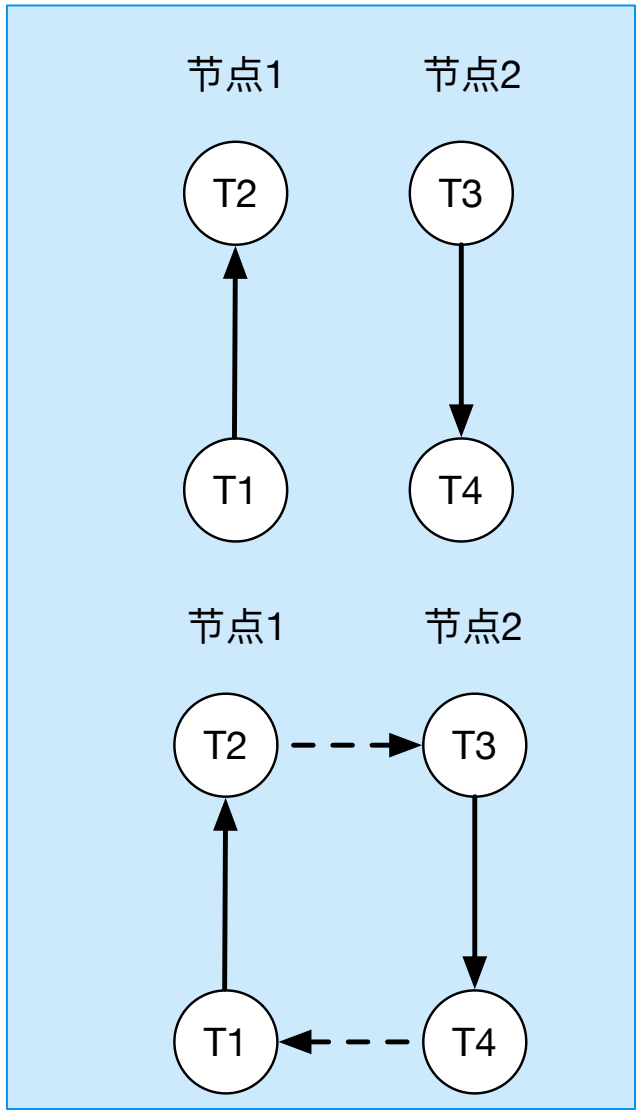
函数	返回值
TT.now()	时间区间[earliest, latest]
TT.after(t)	如果时间t已经过去，则返回true
TT.before(t)	如果时间t还未到达，则返回true

集中式时钟 vs. 分布式时钟

	集中式时钟	分布式时钟	TrueTime
优点	实现简单，全局一致	高可用，不存在中心性能瓶颈	简化应用程序开发
缺点	中心授时系统存在单点故障和性能瓶颈	缺乏中心时钟，逻辑较复杂	需要专有硬件支持 成本较高
代表系统	TiDB	CockroachDB	Spanner

分布式死锁检测

- **局部等待图**
  - 维护单个节点上事务之间的等待关系
- **全局等待图**
  - 维护系统所有节点之间的等待关系
- **并行构建全局等待图**



分布式副本一致性

副本一致性

- 分布式系统中，可能出现故障的多个节点存储的数据达成一致

为什么需要副本一致

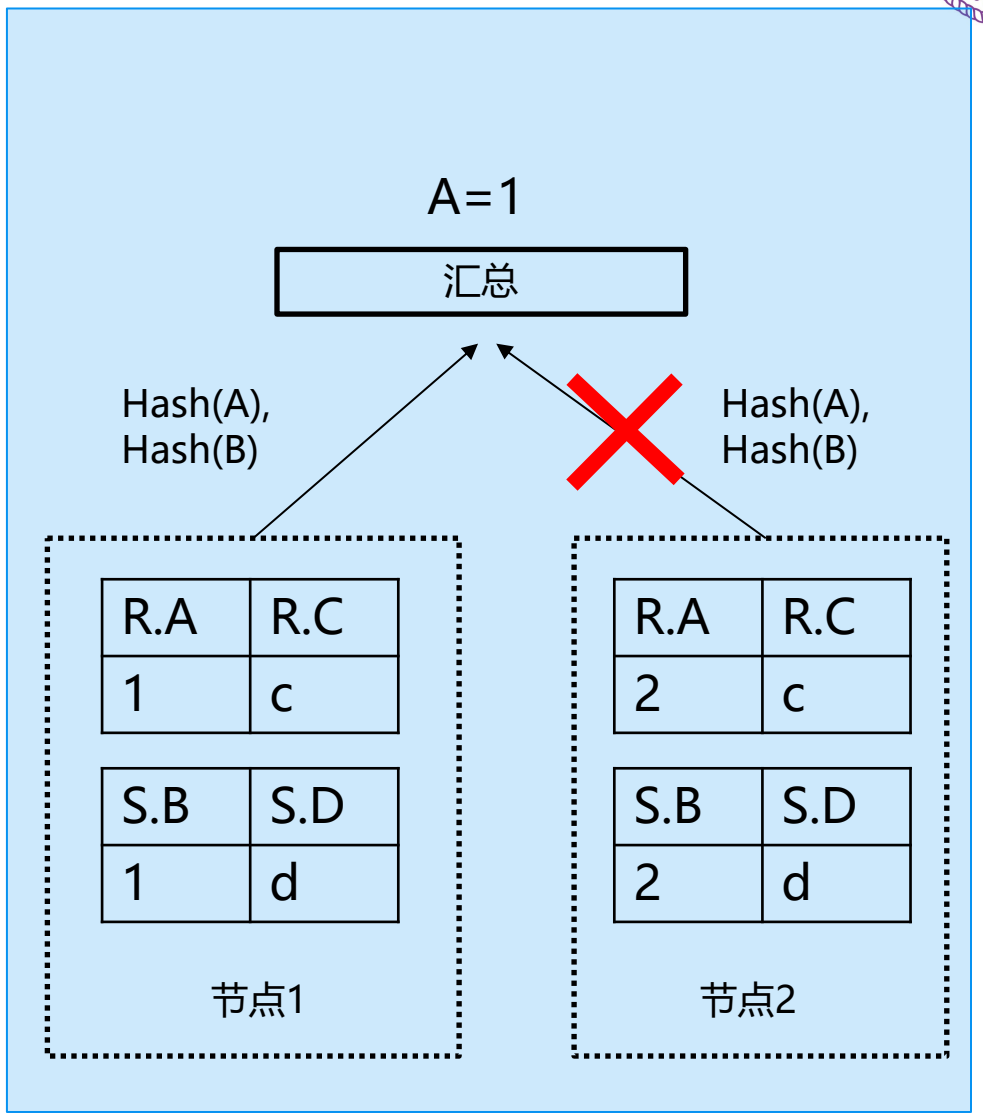
- 分布式系统中，许多问题需要多个节点协商达成一致
- 如某个事务是否提交，领导者选举等
- 使得分布式系统可以像单一节点一样工作

常用算法

- Paxos, Raft
- Quorum

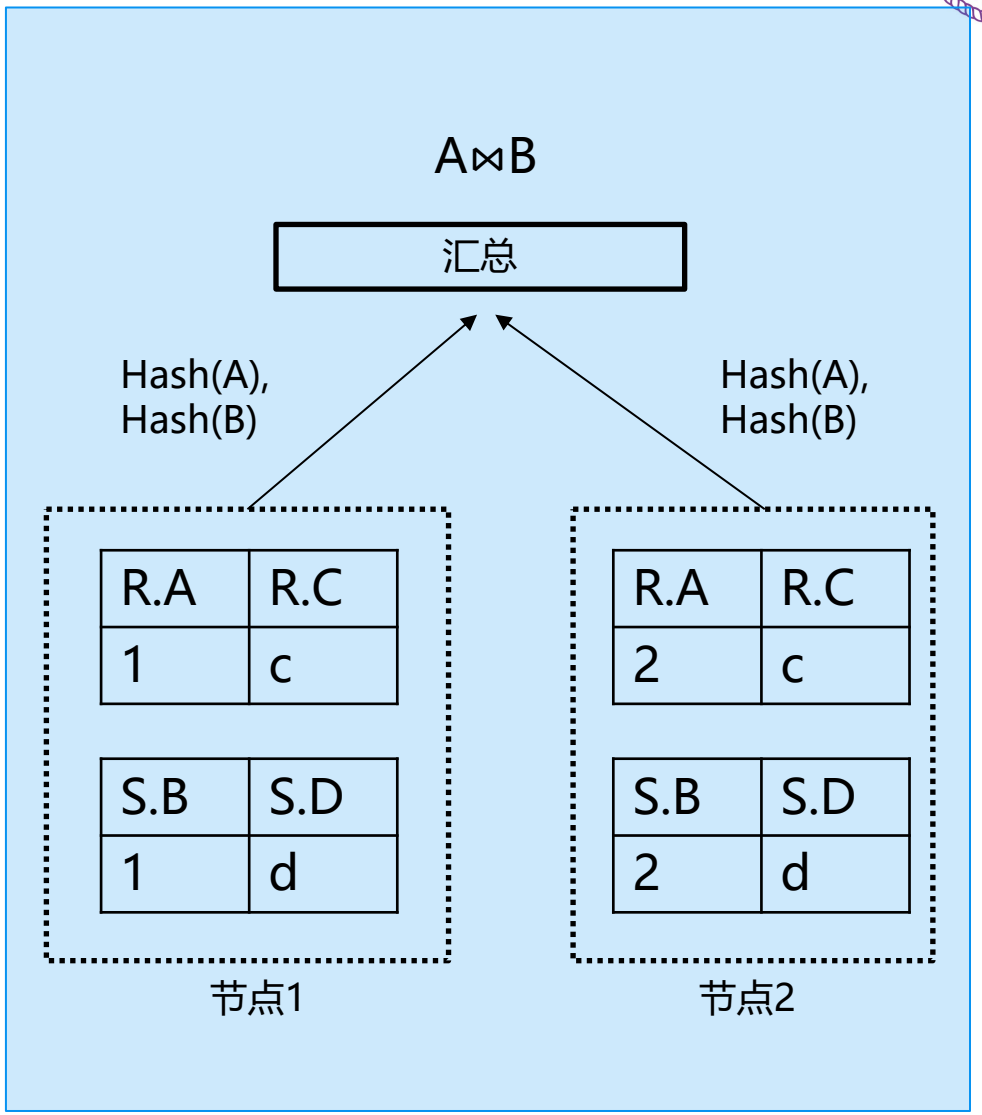
分布式查询优化

- **原则：**
  - 尽量少地进行跨节点数据交换
  - 利用分布式存储特性进行并行化的查询优化与执行
- **分片剪枝**
  - 通过分析查询条件提取出该查询需要扫描的数据分片（节点），避免扫描无关的数据分片
- **分片协同**
  - 尽量并行计算MPP、SMP、SIMD



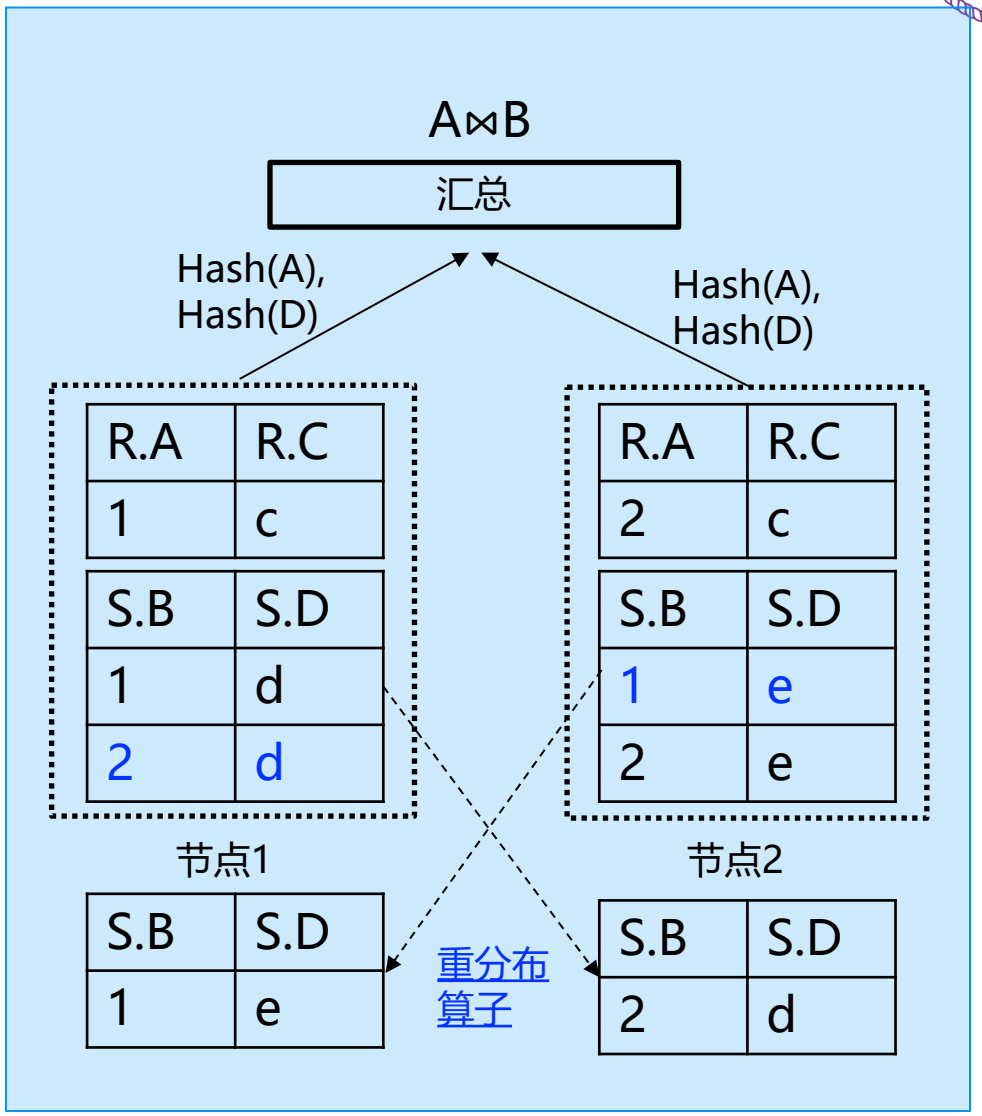
分布式查询优化

- **分布式连接运算（以哈希为例，相同哈希函数和桶数）**
  - ① 表R、S的数据分别按照属性A和B进行了哈希分布→各个节点进行局部连接运算，然后再汇总
  - ② 表R按照属性A进行了哈希分布，表S没有按照属性B进行分布→
    - 将表S按照属性B进行重分布
    - 也可以将表S的数据广播到各个节点
  - ③ 属性A和B都不是分布列
    - ① 将表R和S分别按照A和B 进行重分布再连接
    - ② 也可以将一个较小表进行广播



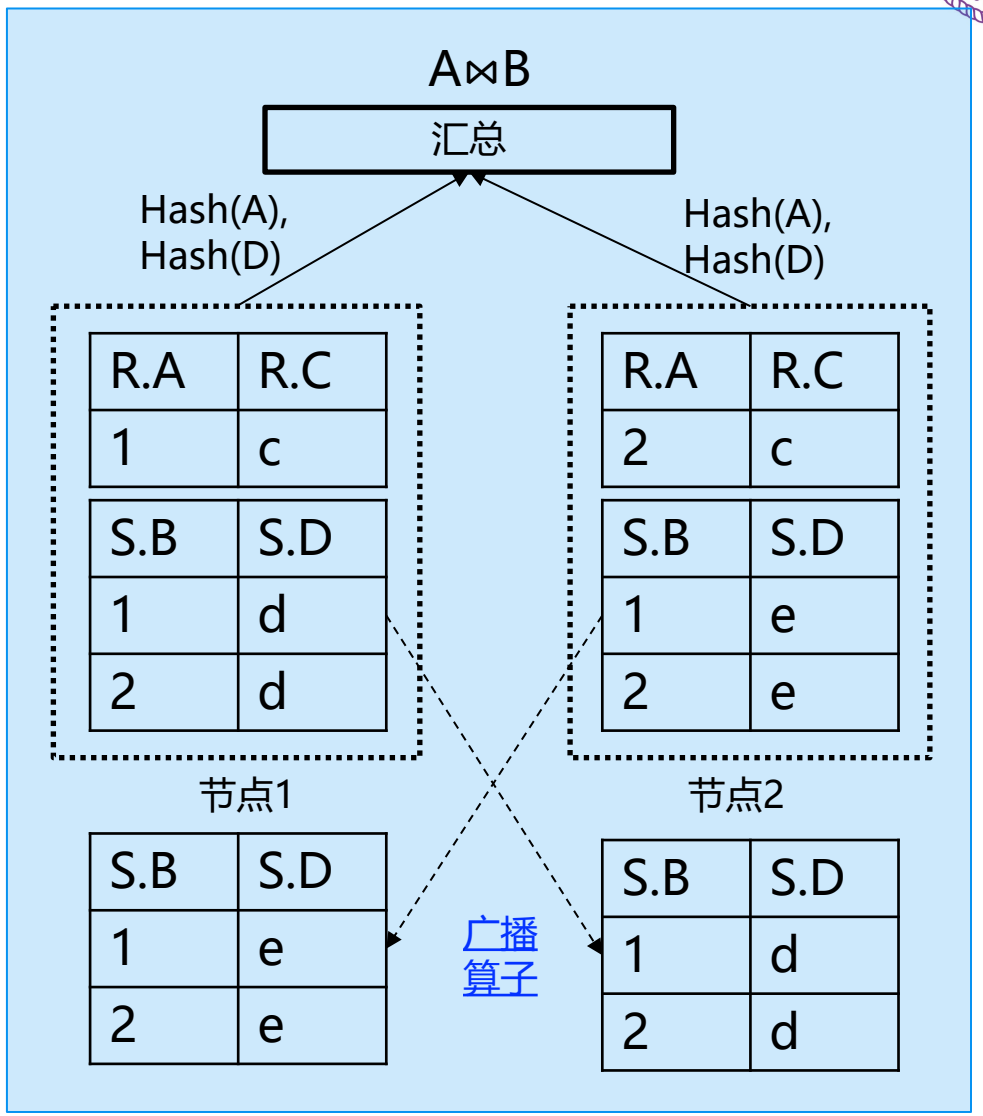
分布式查询优化

- **分布式连接运算（以哈希为例，相同哈希函数和桶数）**
  - ① 表R、S的数据分别按照属性A和B进行了哈希分布→各个节点进行局部连接运算，然后再汇总
  - ② 表R按照属性A进行了哈希分布，表S没有按照属性B进行分布→
    - a) 将表S按照属性B进行重分布，转化为1
    - b) 将表S的数据广播到各个节点，转化为1
  - ③ 属性A和B都不是分布列
    - a) 将表R和S分别按照A和B进行重分布再连接
    - b) 将一个小表（例如R）数据广播到各个节点



分布式查询优化

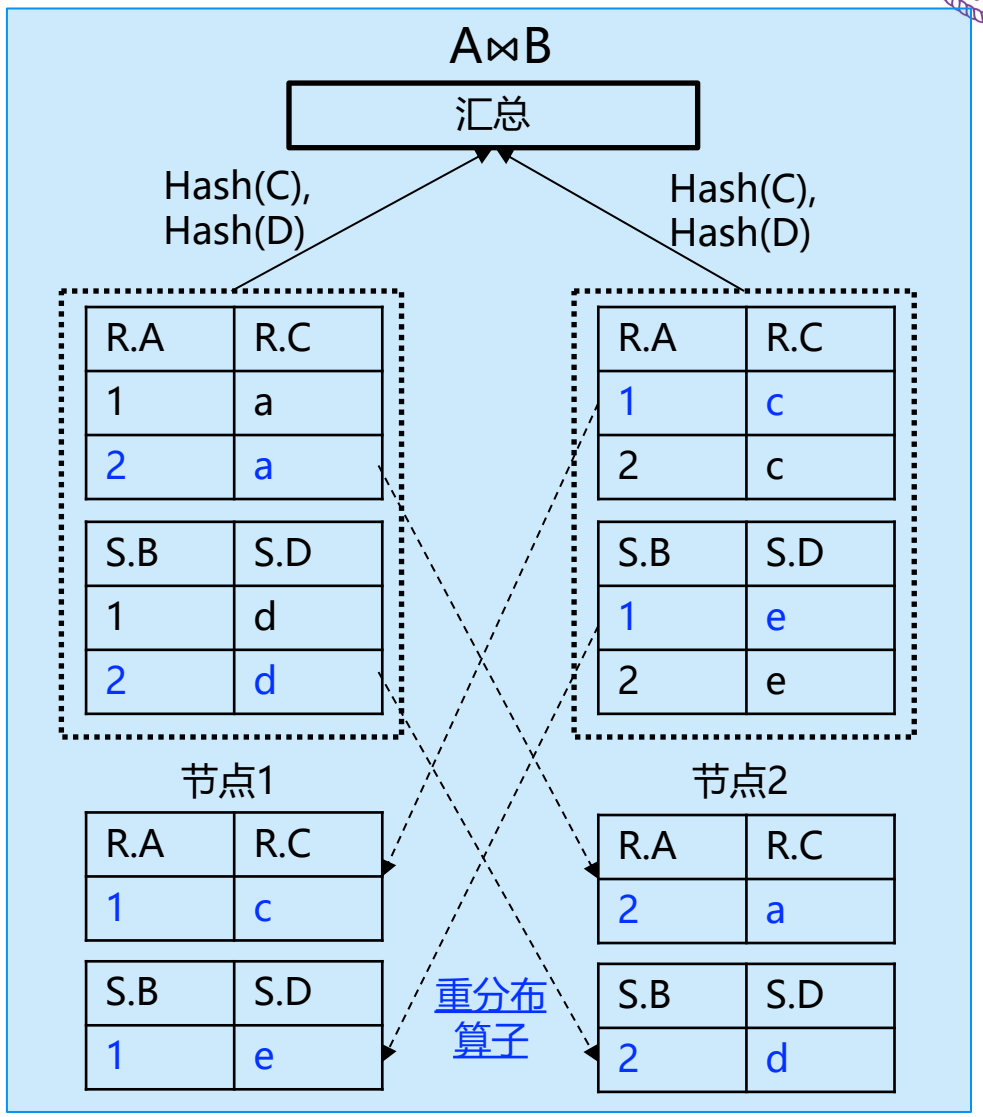
- **分布式连接运算（以哈希为例，相同哈希函数和桶数）**
  - ① 表R、S的数据分别按照属性A和B进行了哈希分布→各个节点进行局部连接运算，然后再汇总
  - ② 表R按照属性A进行了哈希分布，表S没有按照属性B进行分布→
    - a) 将表S按照属性B进行重分布，转化为1
    - b) 将表S的数据广播到各个节点，转化为1
  - ③ 属性A和B都不是分布列
    - a) 将表R和S分别按照A和B进行重分布再连接
    - b) 将一个小表（例如R）数据广播到各个节点



也可能广播R表的数据（代价决定）

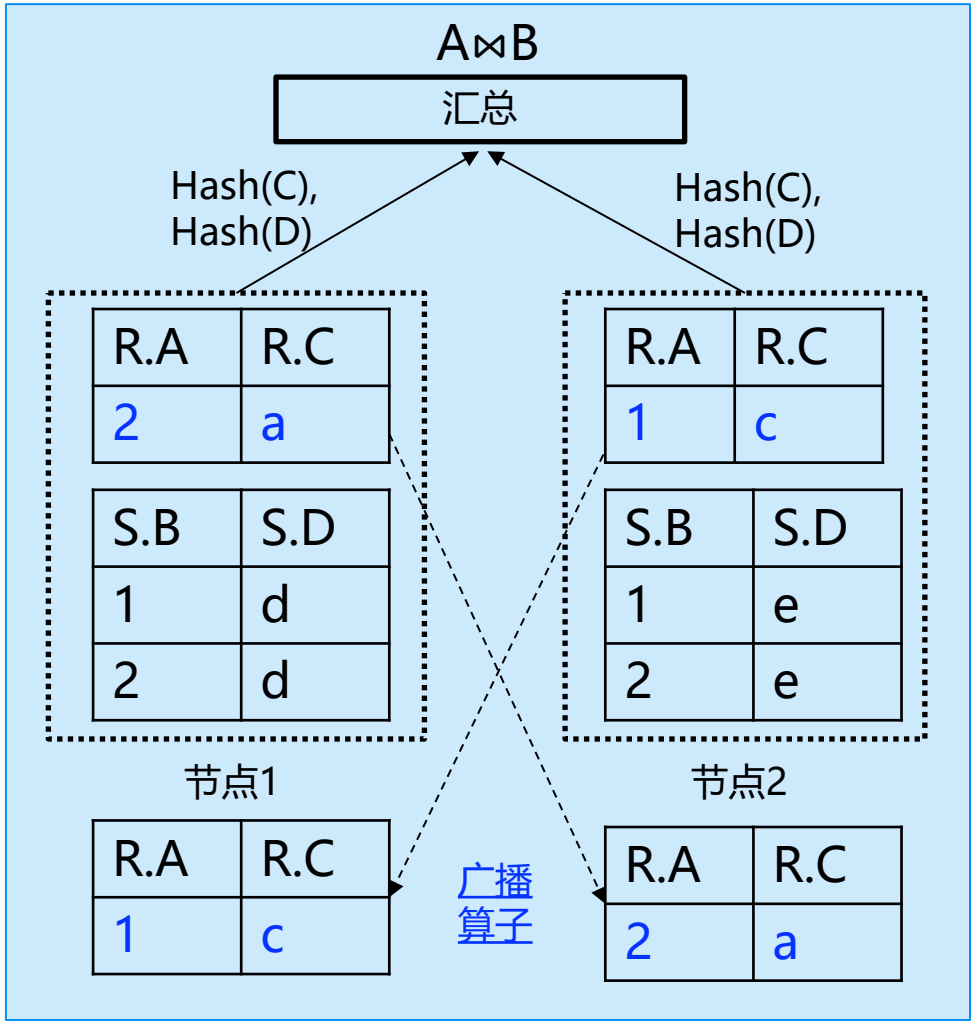
分布式查询优化

- **分布式连接运算（以哈希为例，相同哈希函数和桶数）**
  - ① 表R、S的数据分别按照属性A和B进行了哈希分布→各个节点进行局部连接运算，然后再汇总
  - ② 表R按照属性A进行了哈希分布，表S没有按照属性B进行分布→
    - a) 将表S按照属性B进行重分布，转化为1
    - b) 将表S的数据广播到各个节点，转化为1
  - ③ 属性A和B都不是分布列
    - a) 将表R和S分别按照A和B进行重分布再连接
    - b) 将一个小表（例如R）数据广播到各个节点



分布式查询优化

- **分布式连接运算（以哈希为例，相同哈希函数和桶数）**
  - ① 表R、S的数据分别按照属性A和B进行了哈希分布→各个节点进行局部连接运算，然后再汇总
  - ② 表R按照属性A进行了哈希分布，表S没有按照属性B进行分布→
    - a) 将表S按照属性B进行重分布，转化为1
    - b) 将表S的数据广播到各个节点，转化为1
  - ③ 属性A和B都不是分布列
    - a) 将表R和S分别按照A和B进行重分布再连接
    - b) 将一个小表（例如R）数据广播到各个节点

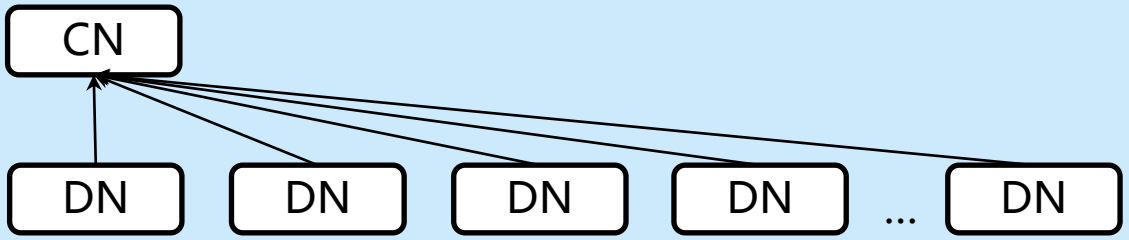


分布式查询优化

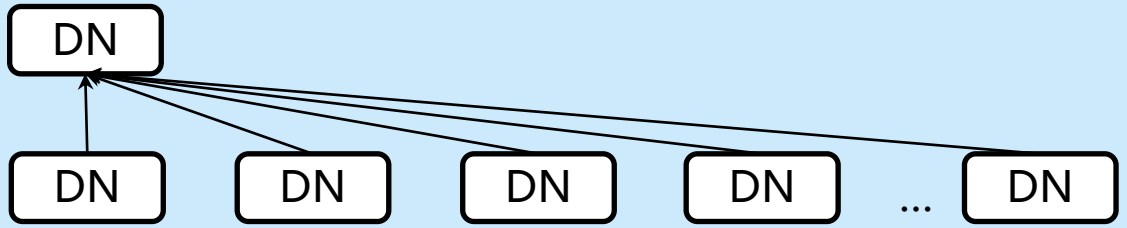
R表Join列是否=分布列	S表Join列是否=分布列	可能生成的路径
是	是	local(R) ⋈ local(S)
是	否	① local(R) ⋈ redistribute(S) ② broadcast(R) ⋈ local(S)
否	是	① redistribute(R) ⋈ local(S) ② local(R) ⋈ broadcast(S)
否	否	① redistribute(R) ⋈ redistribute(S) ② broadcast(R) ⋈ local(S) ③ local(R) ⋈ broadcast(S)

分布式查询优化

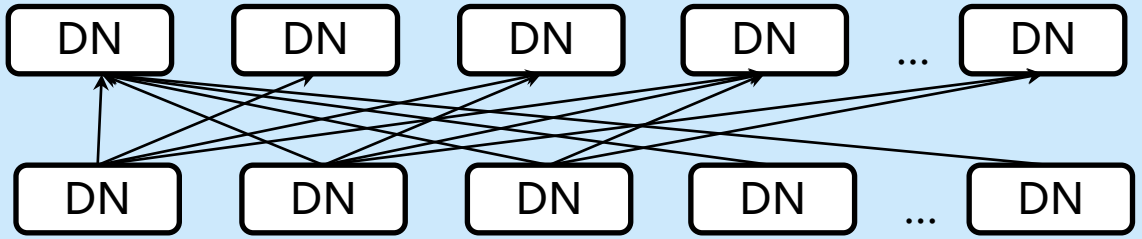
Gather Stream(N:1)：N个DN节点发送数据给CN节点，一般用于汇总DN节点到CN节点的过程



Gather Broadcast(1:N)：由1个DN节点发送全量数据给其他DN节点，一般用于Join复制全量数据



Redistribute Stream(M:N)：M个DN节点将其数据根据关联条件、聚集分组表达式算出Hash值，根据重新计算的Hash值进行分布，发送数据到对应的目标节点。一般用于Join、Agg中的重分布场景



02

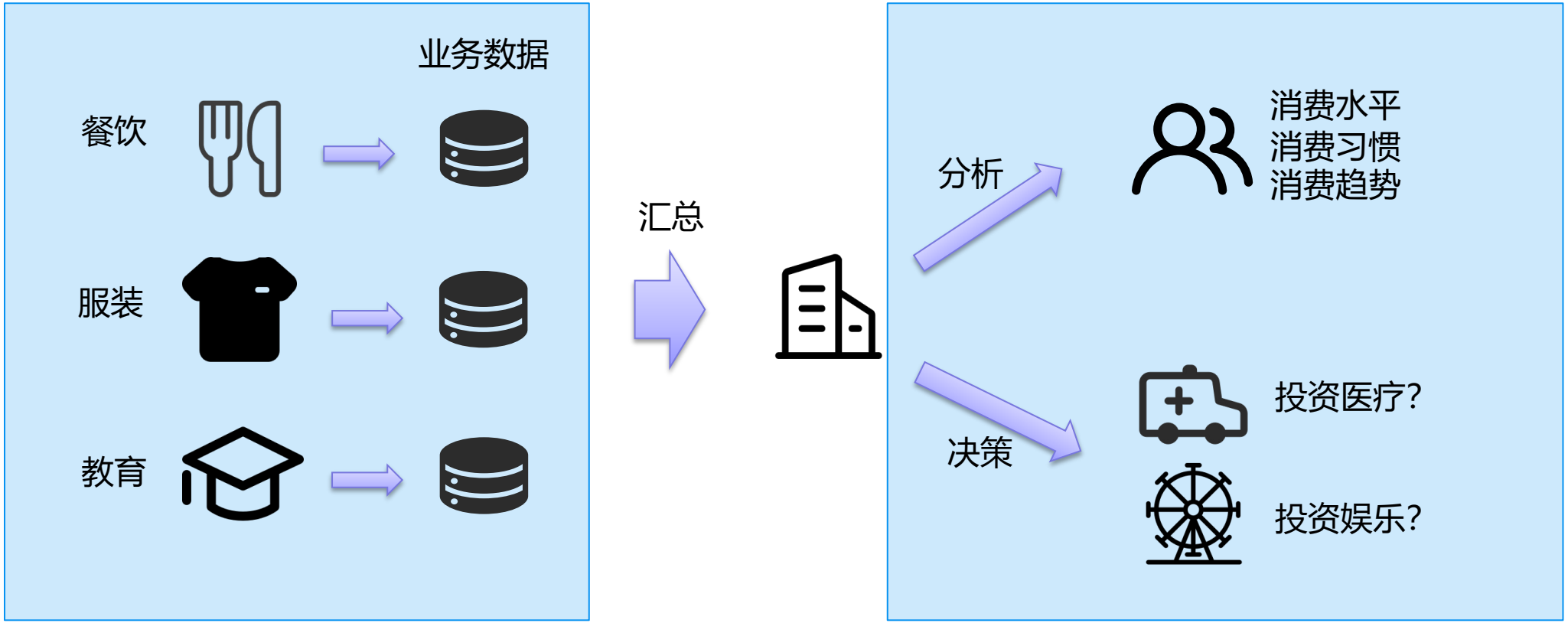
OLAP数据库

OLAP数据库

- **OLTP（OnLine Transactional Processing，联机事务处理）**
  - 主要关注对某类业务事件的记录，并提供增删改查等处理数据的基本操作
  - 以事务作为数据库中数据处理的基本单位
- **OLAP (OnLine Analytical Processing，联机分析处理)**
  - 旨在利用数据进行统计分析，以便为企业提供决策支持
- **OLAP与OLTP**
  - 目的：OLTP处理交易；OLAP支持决策
  - 面向对象：OLTP面向事务处理人员；OLAP面向分析决策人员
  - 工作负载：OLTP主要是增、删、改，简单查询；OLAP主要是复杂查询（和即席查询）
  - 主要衡量指标：OLTP事务吞吐量；OLAP查询响应速度

OLAP数据库

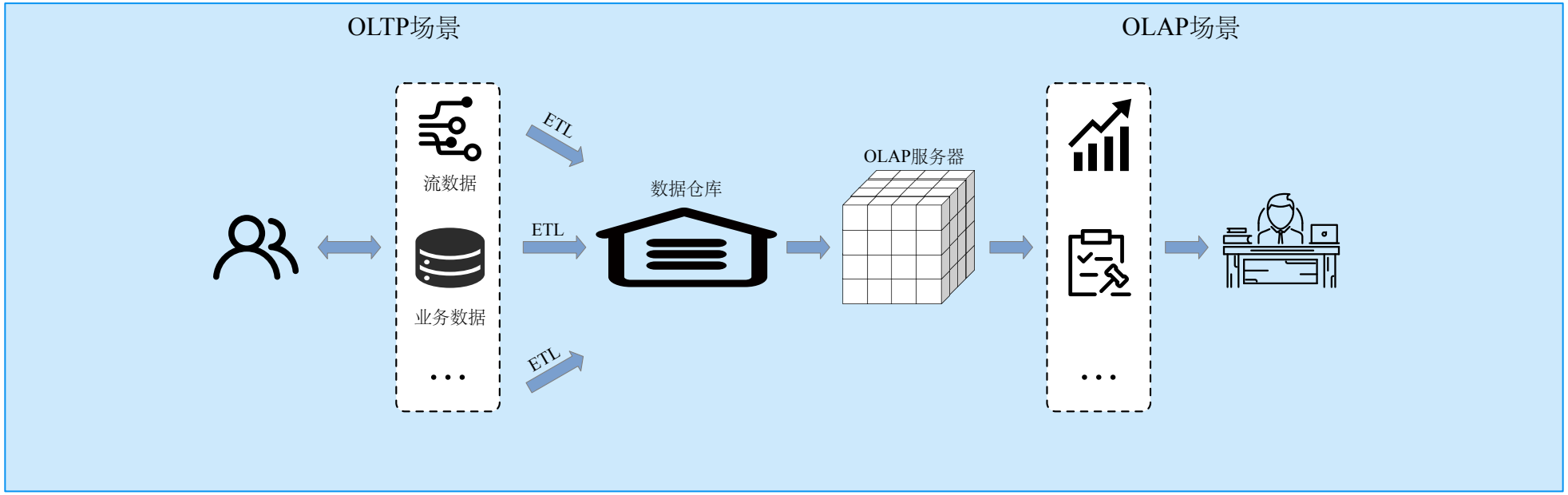
➢ 一个OLAP场景



OLAP与数据仓库

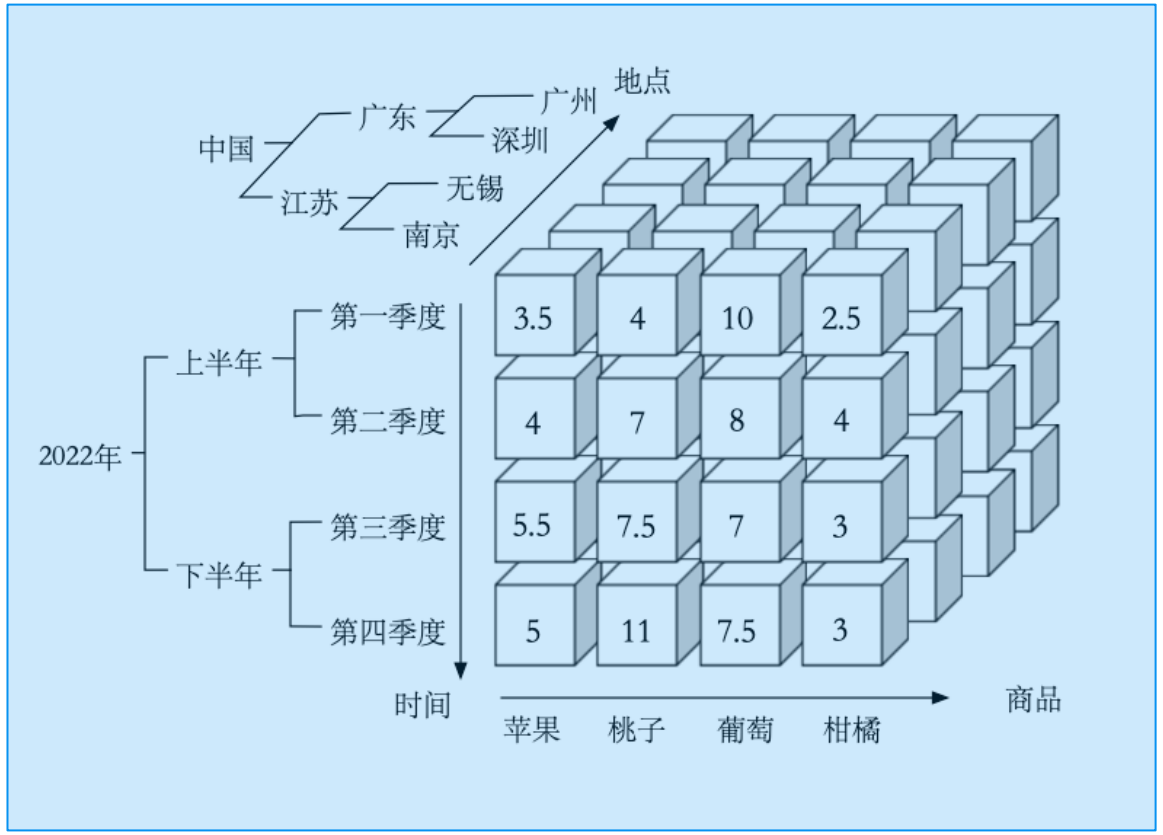
➢ 端到端部署

- 在OLTP场景下，用户在不同的业务系统中产生业务数据；
- **数据仓库**收集整合不同源的数据并进行统一存储（ETL操作）
- **OLAP服务器**对这些数据进行统一管理并支持上层分析操作
  1. 多维OLAP（MOLAP）
  2. 关系OLAP（ROLAP）
  3. 混合OLAP（HOLAP）



OLAP与数据仓库

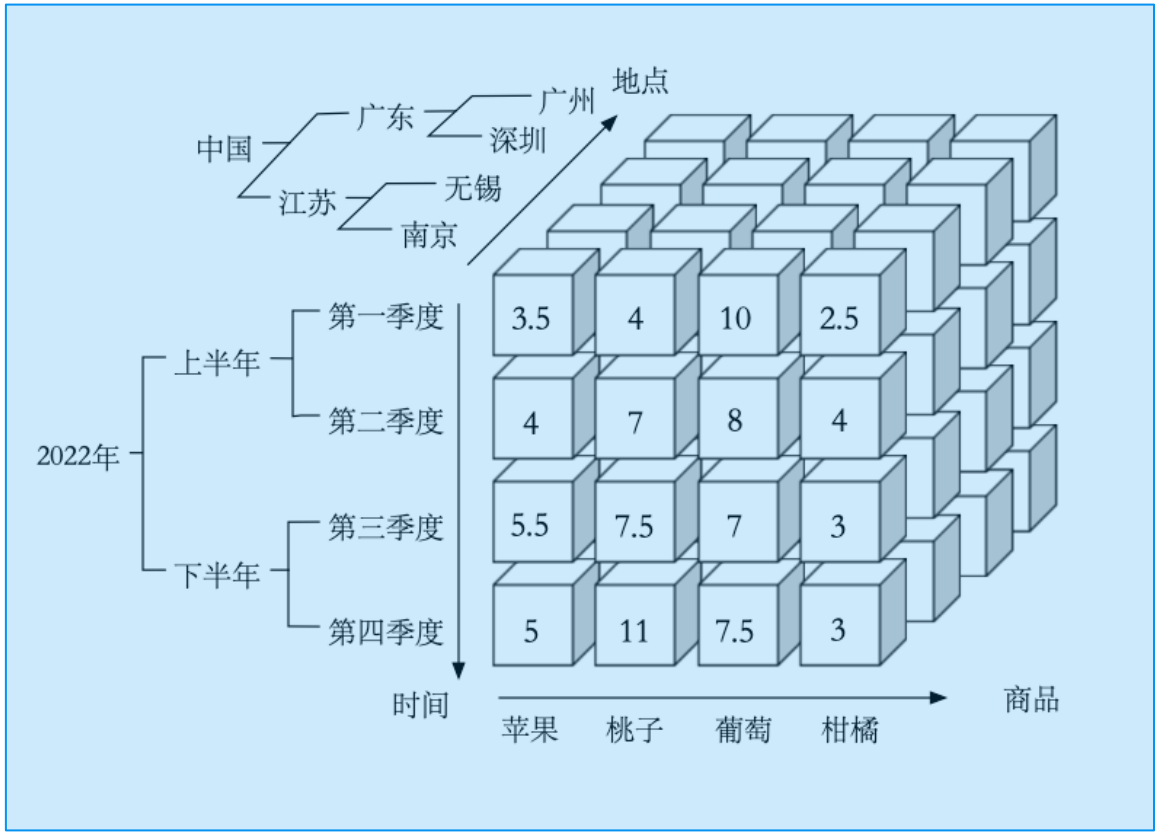
- **数据仓库是一个面向主题的、集成的、相对稳定的、反映历史变化的、用于支持管理决策的数据集合。**
- **在实际应用中，一个数据仓库通常都基于业务中的某个明确主题（如某类商品的生产与销售情况）而建立，其中的数据可能来自多个不同模式的数据源，且其中的关键数据往往会随着时间发生变化。**
  - 数据抽取 Extraction
  - 数据转换 Transformation
  - 数据加载 Loading
- **复杂查询**
  - 分组聚集查询
  - 多维度分析
  - 即席查询



数据仓库

➢ 数据仓库概念

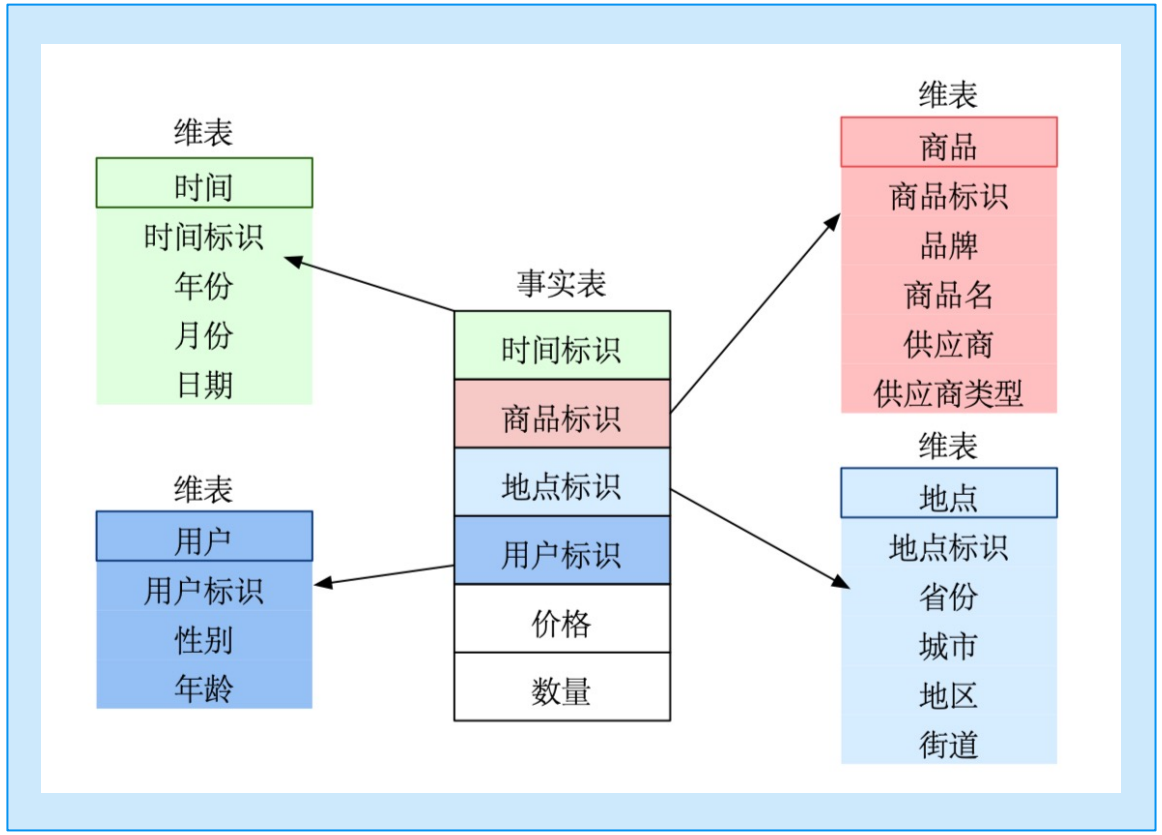
- 主题（subject）：要分析的具体内容，例如分析某时某地某商品的销售情况，在设计数据仓库模式之前就会被确定；
- 维度（dimension）：分析角度，例如时间和位置；
- 度量（measure）：要分析的指标，通常是数值类型，例如商品价格及其销售数量；
- 事实表（fact table）：存储主题内容的全量信息；
- 维表（dimension table）：存储事实表中事件描述角度的具体信息。



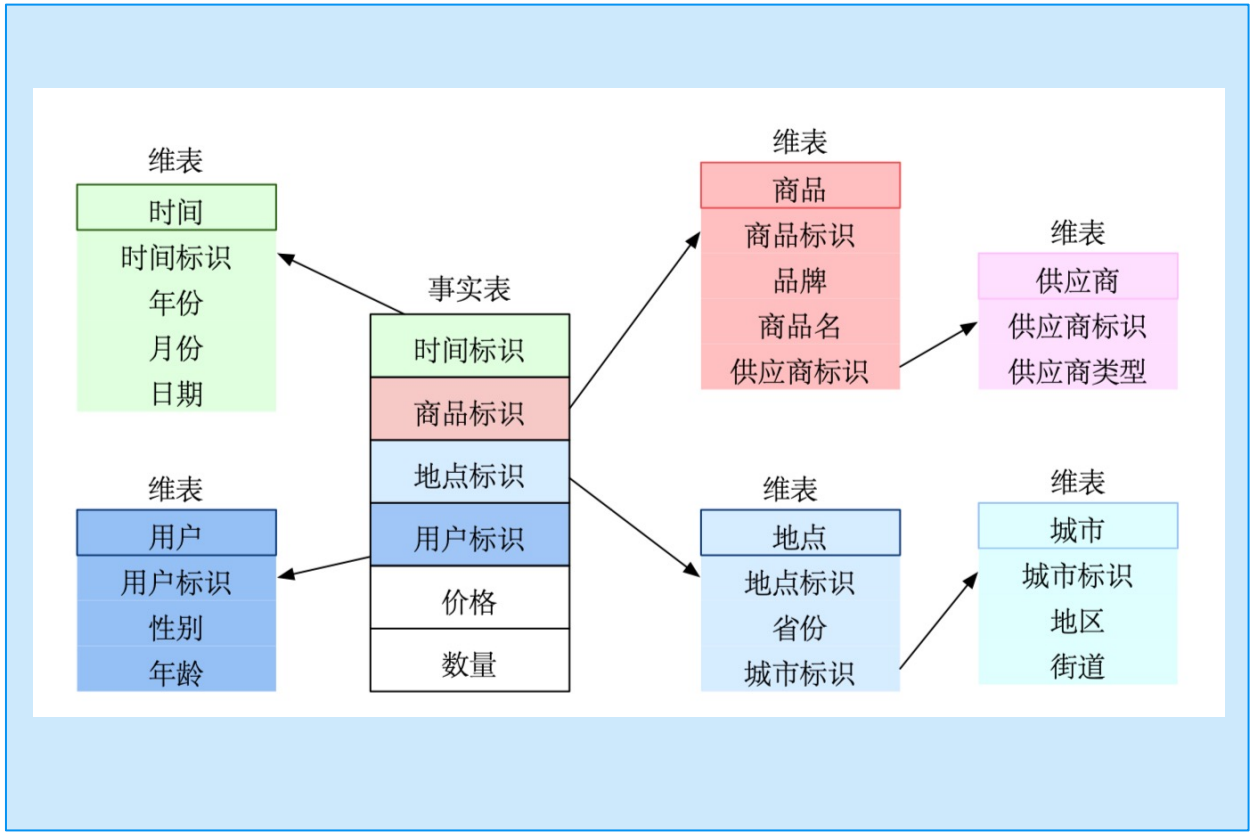
数据仓库

➢ 数据仓库模式

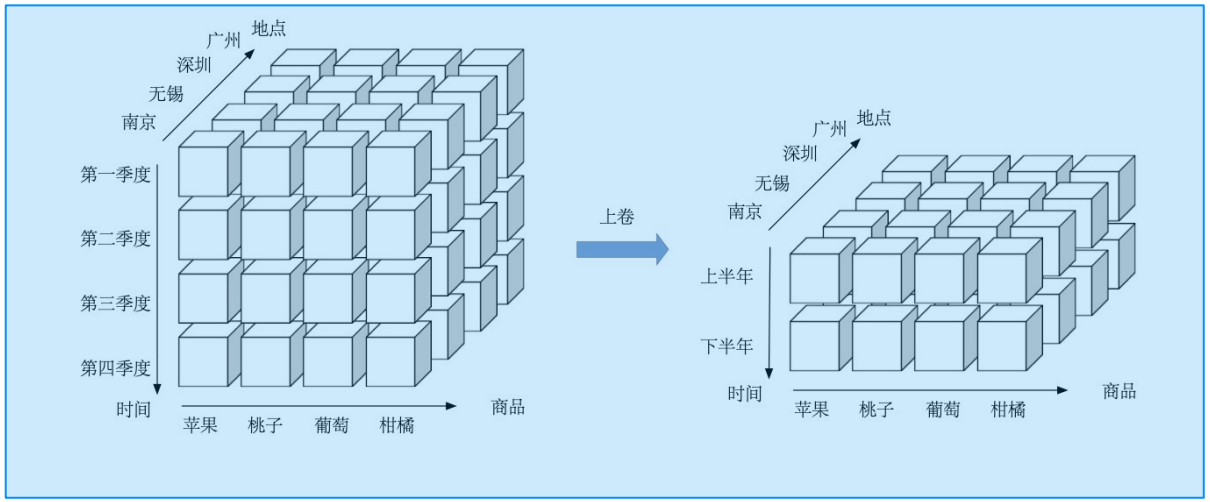
- 星型模式

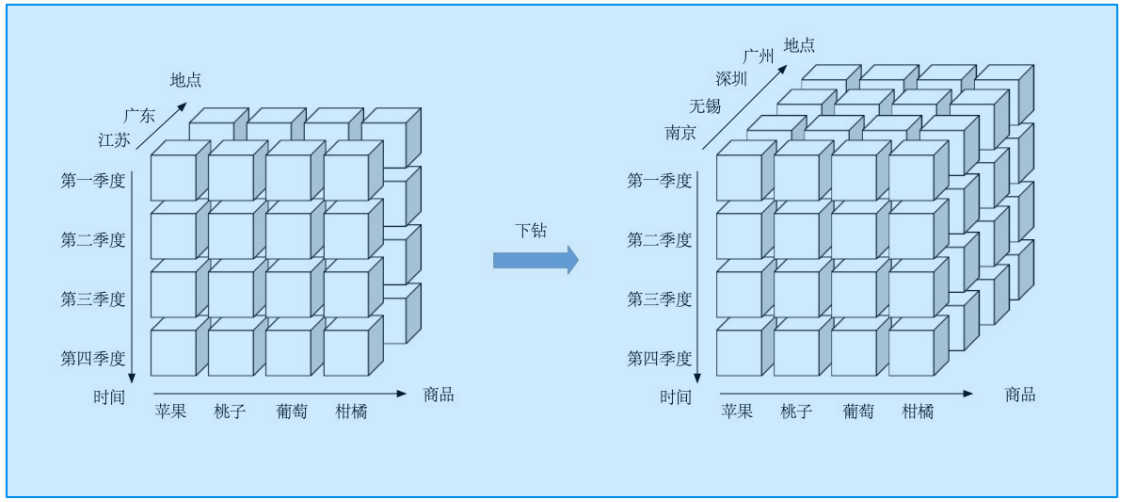


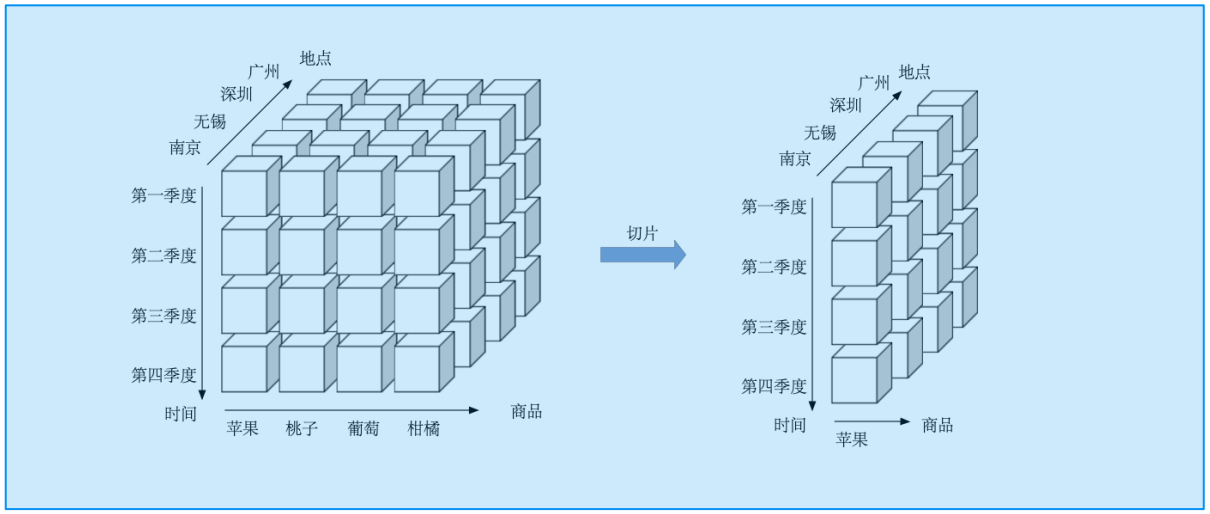
- 雪花模式

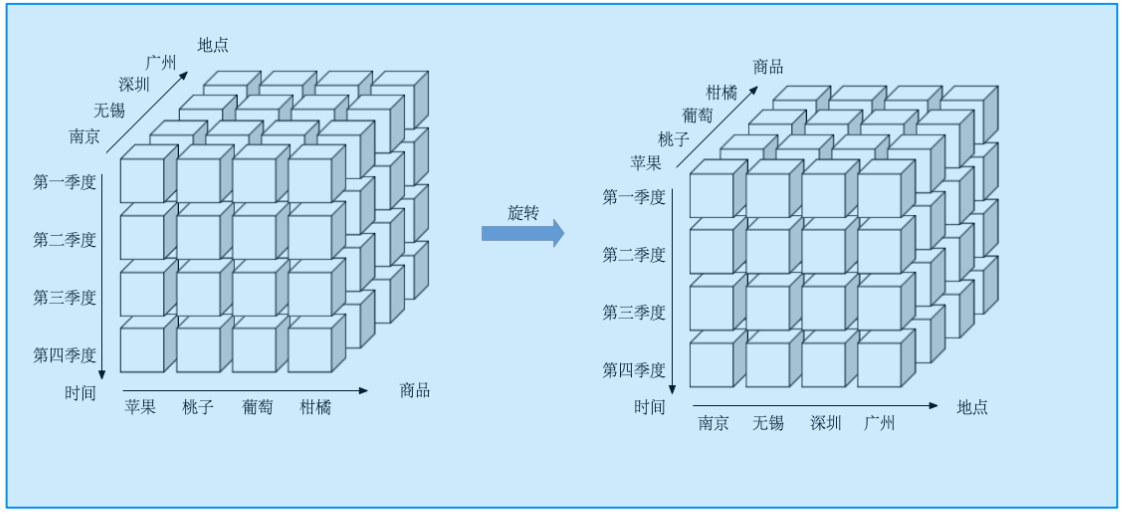


数据仓库操作


地点
广州
深圳
无锡
南京
第一季度
第二季度
第三季度
第四季度
时间
苹果 桃子 葡萄 柑橘
商品
上卷
上半年
下半年

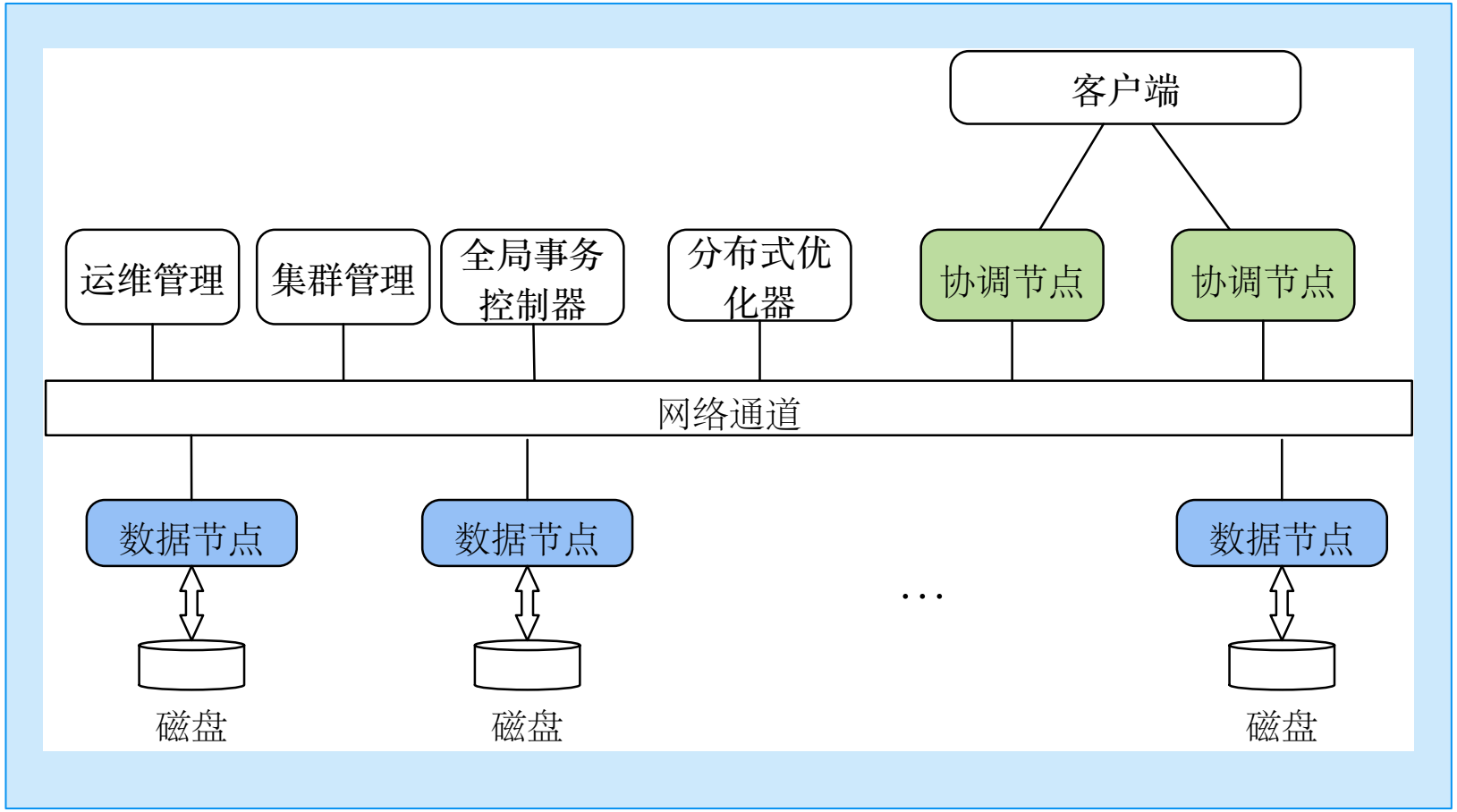

地点
广东
江苏
第一季度
第二季度
第三季度
第四季度
时间
苹果 桃子 葡萄 柑橘
商品
下钻


地点
广州
深圳
无锡
南京
第一季度
第二季度
第三季度
第四季度
时间
苹果 桃子 葡萄 柑橘
商品
切片


地点
广州
深圳
无锡
南京
第一季度
第二季度
第三季度
第四季度
时间
苹果 桃子 葡萄 柑橘
商品
旋转
南京 无锡 深圳 广州

OLAP数据库架构

- **MPP无共享架构**
  - 协调节点
  - 数据节点
  - 全局事务控制器
  - 分布式优化器
  - 集群管理
  - 运维管理

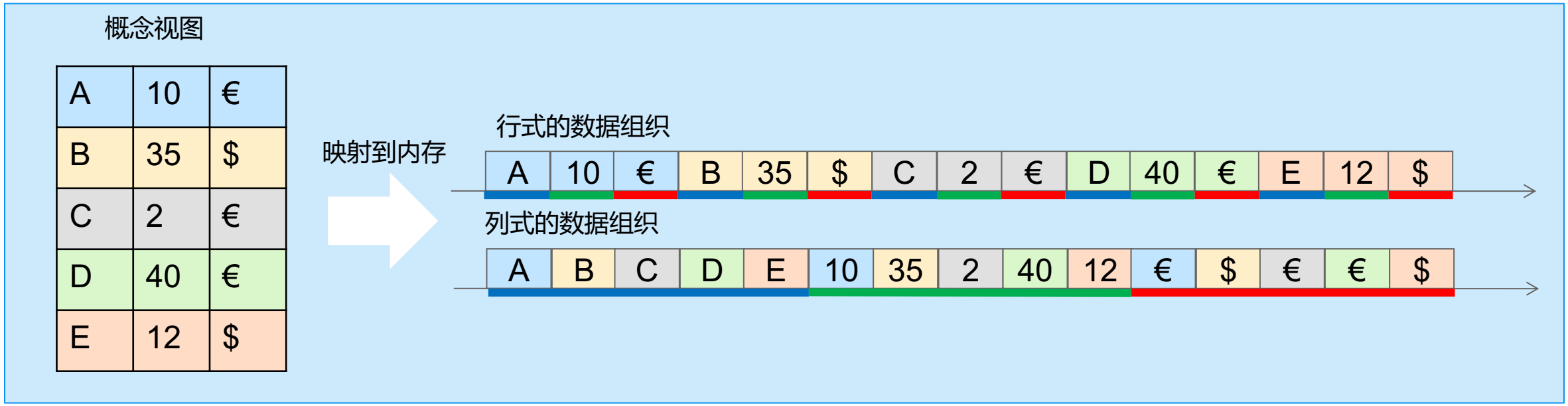


OLAP数据库关键技术

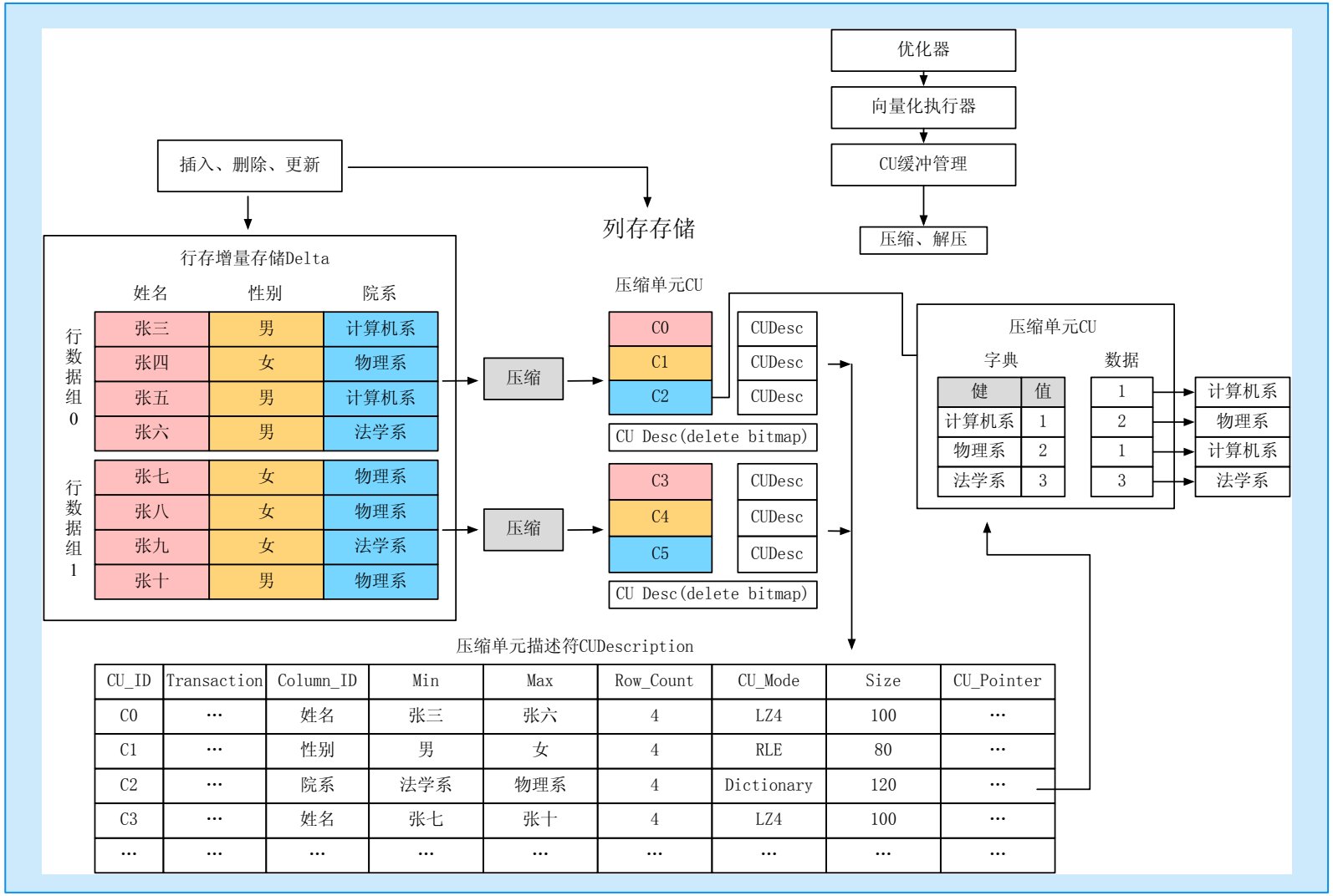
- **列存储技术**
- **列存编码与压缩**
  - 游程编码
  - 字典编码
- **列索引**
  - 区域图索引
  - 布隆过滤器索引
- **向量化执行**
- **编译执行**
- **物化视图**

OLAP数据库存储

- **OLAP查询主要关注少数列的大量行**
  - 行存储：需要读取其他无用列的数据 ×
  - 列存储：可以有效提高OLAP查询速度 √



OLAP数据库存储


优化器
向量化执行器
CU缓冲管理
压缩、解压
插入、删除、更新
列存存储
行存增量存储Delta
姓名 性别 院系
行数据组0
张三 男 计算机系
张四 女 物理系
张五 男 计算机系
张六 男 法学系
行数据组1
张七 女 物理系
张八 女 物理系
张九 女 法学系
张十 男 物理系
压缩
压缩
压缩单元CU
C0
C1
C2
C3
C4
C5
CUDesc
CU Desc(delete bitmap)
压缩单元CU
字典
健 值
计算机系 1
物理系 2
法学系 3
数据
1
2
1
3
计算机系
物理系
计算机系
法学系
压缩单元描述符CUDescription
CU_ID Transaction Column_ID Min Max Row_Count CU_Mode Size CU_Pointer
C0 … 姓名 张三 张六 4 LZ4 100 …
C1 … 性别 男 女 4 RLE 80 …
C2 … 院系 法学系 物理系 4 Dictionary 120 …
C3 … 姓名 张七 张十 4 LZ4 100 …
… … … … … … … … …

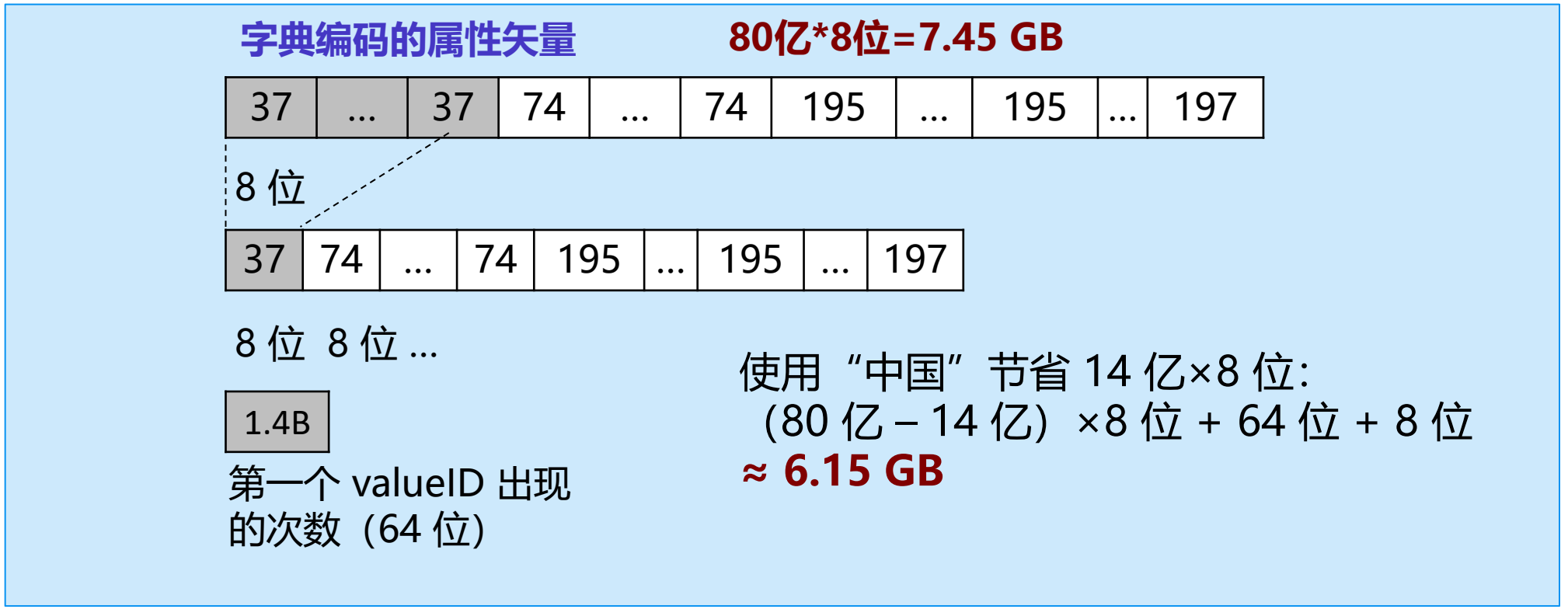
OLAP数据库存储：编码与压缩

- 200 个国家 = 8 位； 100 万个城市 = 20 位
- 100 个不同的第二国籍 = 7 位；500 万个名字 = 23 位

recID	fname	lname	gender	country	city	birthday	2nd_nationality
0	Martin	Albrecht	m	GER	Berlin	08-05-1955	n/a
1	Michael	Berg	m	GER	Berlin	03-05-1970	n/a
2	Hanna	Schulze	f	GER	Hamburg	04-04-1968	n/a
3	Anton	Meyer	m	AUT	Innsbruck	10-20-1992	US
4	Ulrike	Schulze	f	GER	Potsdam	09-03-1977	n/a
5	Martin	Schulz	m	GER	Mainz	06-04-1980	GER
6	Sushi	Pao	f	CN	Peking	09-12-1954	n/a
7	Chen	Su Wong	m	CN	Shanghai	27-06-1999	n/a
...	...	...	...	...	...	...	...

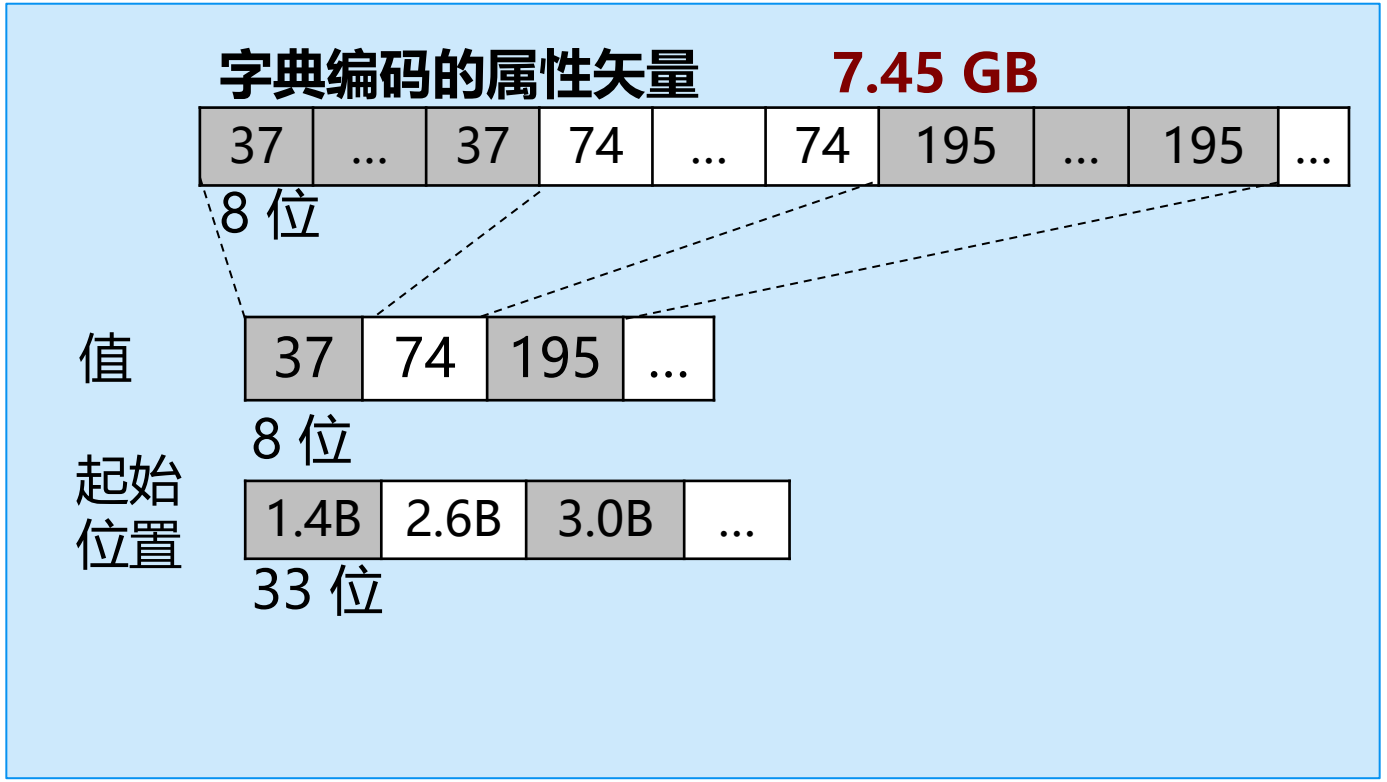
OLAP数据库存储：首字符编码

- 当数据列以一长序列相同的值开始时使用
- 数据列中的一个主值，其余的值大多是唯一的或者重复率非常低
- 例如：国家列，按国家人口排序的表



OLAP数据库存储：游程编码

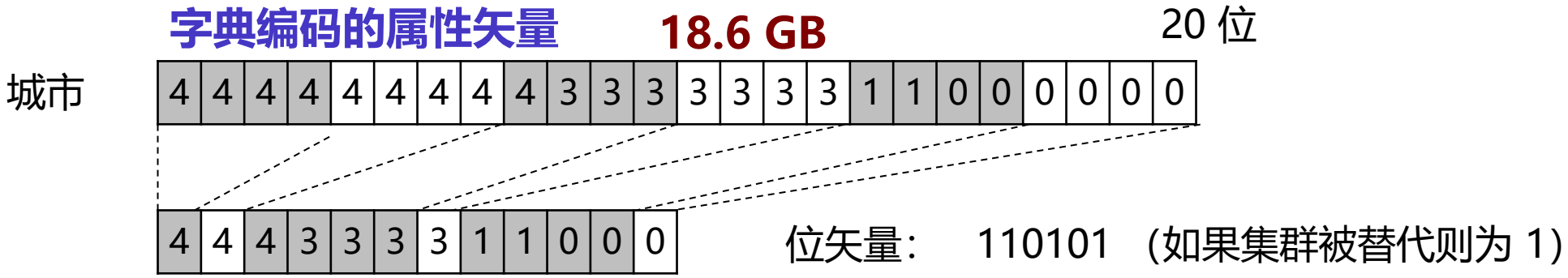
- 将相同值序列替换为该值的单一实例及其出现次数和起始位置
- 例如：国家列，按国家人口排序的表（假设200个国家）



游程编码
200×（33 位 + 8 位） ≈ 1 KB

OLAP数据库存储：集群编码

- 属性矢量划分为N个固定大小的数据块（一般为 1024）
- 如果一个集群只包含一个值，则使用第一次出现的该值替换集群
- 长度N的位矢量表明哪些集群被单一值替代



示例：城市数据列，按国家、城市排序的表

集群大小： 一个块1024 个元素 → 80亿数据共780 万个数据块

最坏情况下的假设：每个城市 1 个不可压缩的数据块

不可压缩的数据块：100 万×1024×20 位 2441 MB

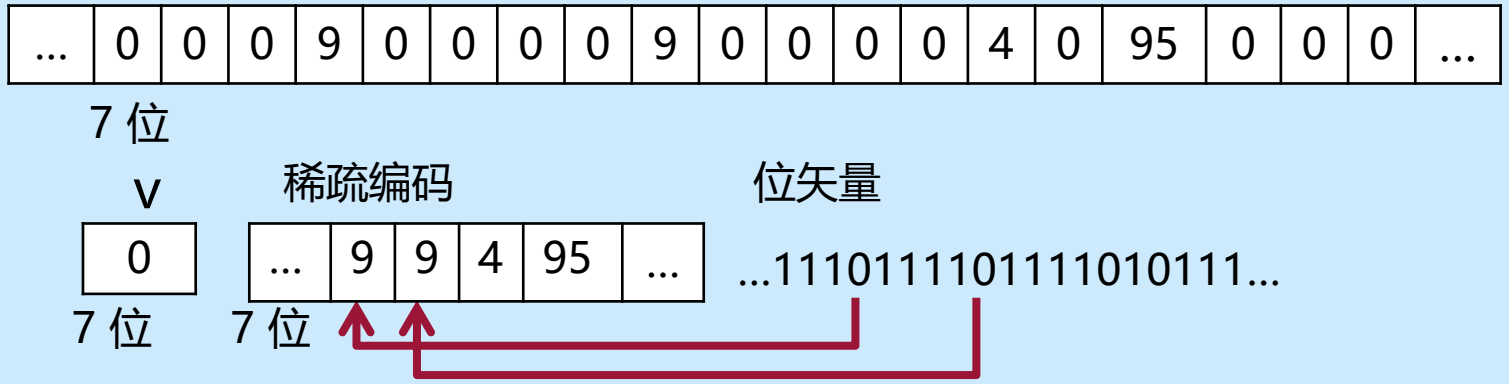
可压缩的数据块： (780 万 - 100 万)×20 位 + 16 MB

位矢量： 780 万×1 位 + 1 MB ≈ **2.4 GB**

OLAP数据库存储：稀疏编码

- 删除出现最频繁的值 v
- 一个位矢量表明在哪些位置将 v 从原始序列中删除

字典编码的属性矢量



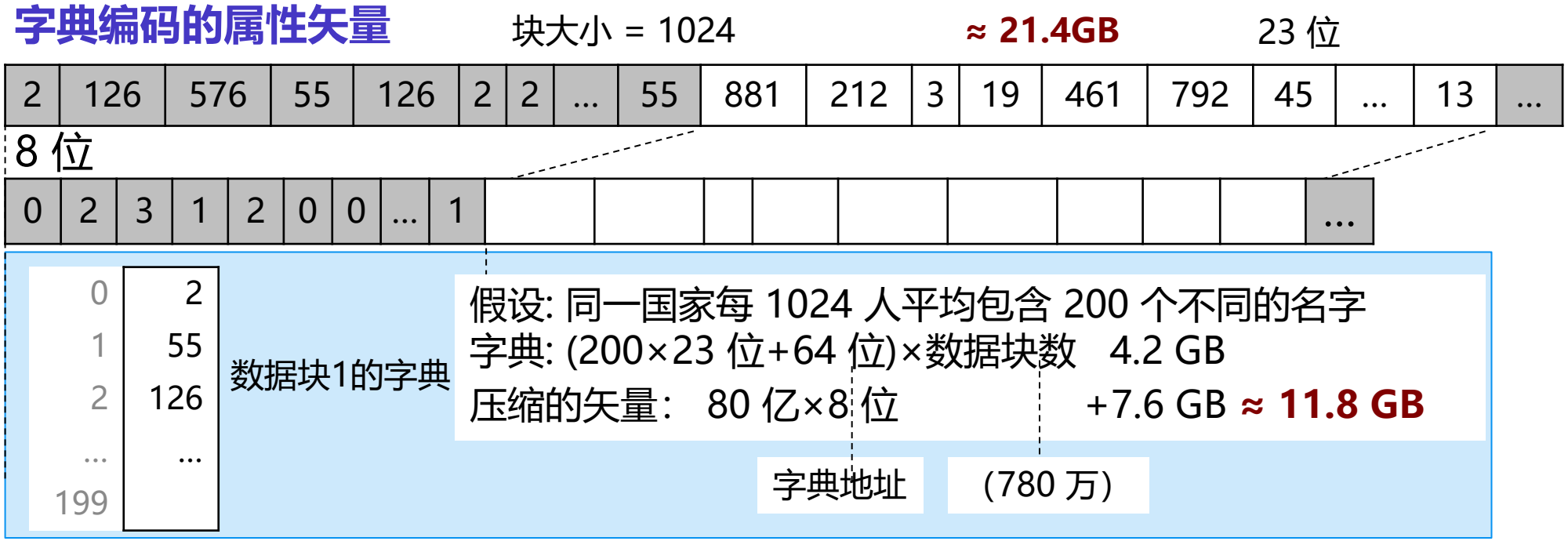
假设：　99 % 的人没有第二国籍

位矢量：	80 亿×1 位	954 MB
稀疏编码的属性矢量：	8,000 万×7 位	+ 67 MB
最频繁的值：		+ 7 位 ≈ 1 GB

OLAP数据库存储：间接编码

- **序列被划分为 N 个 S 大小的数据块（一般为 1024）**
- **如果一个数据块只包含几个完全不同的值，则构建单独的字典对该数据块中的值进行编码，并链接到新字典 + 有字典的数据块**

示例：fname 数据列，按国家排序的表



OLAP数据库存储：字典编码

- 用字典编码(位数少)来代替数据（位数多）
- 城市原始数据20字节，编码3个字节

字典

valueID	value
0	Hamburg
1	Innsbruck
2	Potsdam
3	Berlin
4	Mainz
5	Peking
6	Shanghai
…	…
32722	…

	fname	lname	gender	country	city	city
0	Martin	Albrecht	m	GER	Berlin	3
1	recID	Berg	m	GER	Berlin	3
2	Hanna	Schulze	f	GER	Hamburg	0
3	Anton	Meyer	m	AUT	Innsbruck	1
4	Ulrike	Schulze	f	GER	Potsdam	2
5	Martin	Schulz	m	GER	Mainz	4
6	Sushi	Pao	f	CN	Peking	5
7	Chen	Su Wong	m	CN	Shanghai	6
...	...	...	...	...	...	...

OLAP数据库存储：字典编码

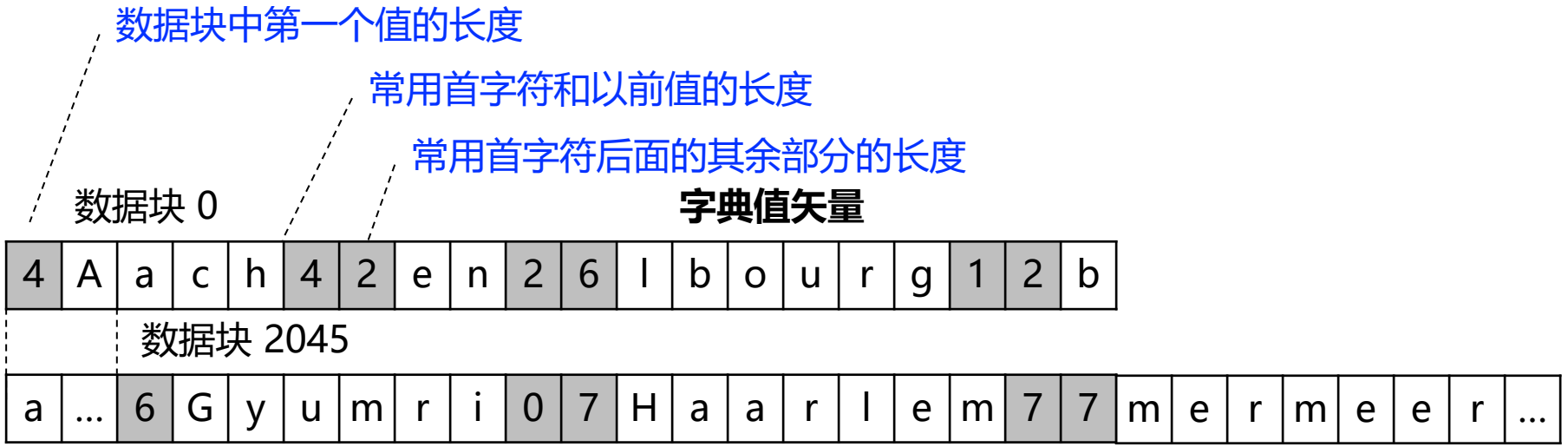
列	列基数	Entropy	条目大小	普通文本大小	字典大小（字典 + 列）
名字	500 万 23 位	6.25×10^{-4}	49 字节	365.10 GB ≈ 373,840 MB	234 MB + 21.42 GB ≈ 22,168 MB
姓氏	800 万 23 位	1×10^{-3}	50 字节	372.5 GB ≈ 381,470 MB	381 MB + 21.42 GB ≈ 22,316 MB
性别	2 1 位	2.5×10^{-10}	1 字节	7.45 GB ≈ 7,630 MB	2 字节 + 0.93 GB ≈ 954 MB
城市	100 万 20 位	1.25×10^{-4}	49 字节	365.08 GB ≈ 373,840 MB	46.73 MB + 18.62 GB ≈ 19,120 MB
国家	200 8 位	2.5×10^{-8}	49 字节	365.08 GB ≈ 373,840 MB	6.09 KB + 7.45 GB ≈ 7,629 MB
出生日期	40,000 16 位	5×10^{-6}	2 字节	14.90 GB ≈ 15,260 MB	76.29 KB + 14.90 GB ≈ 15,259 MB

OLAP数据库存储：差值编码

- **对排序字符串分块压缩（每数据块16个字符串,100万数据62500块）**

字典

	valueID	value
数据块 0	0	Aach
	1	Aachen
	2	Aalbourg
	3	Aba
数据块 2045	…	…
	32720	Gyumri
	32721	Haarlem
	32722	Haarlemmer-meer
	…	…



字典：100 万个城市（49 字节）**≈ 46.7 MB**

假设： 城市名字的平均长度为 7，平均重复 3 个字母

对于“数量”：最长的城市名字 49 个字母，需要 6 位（bits）

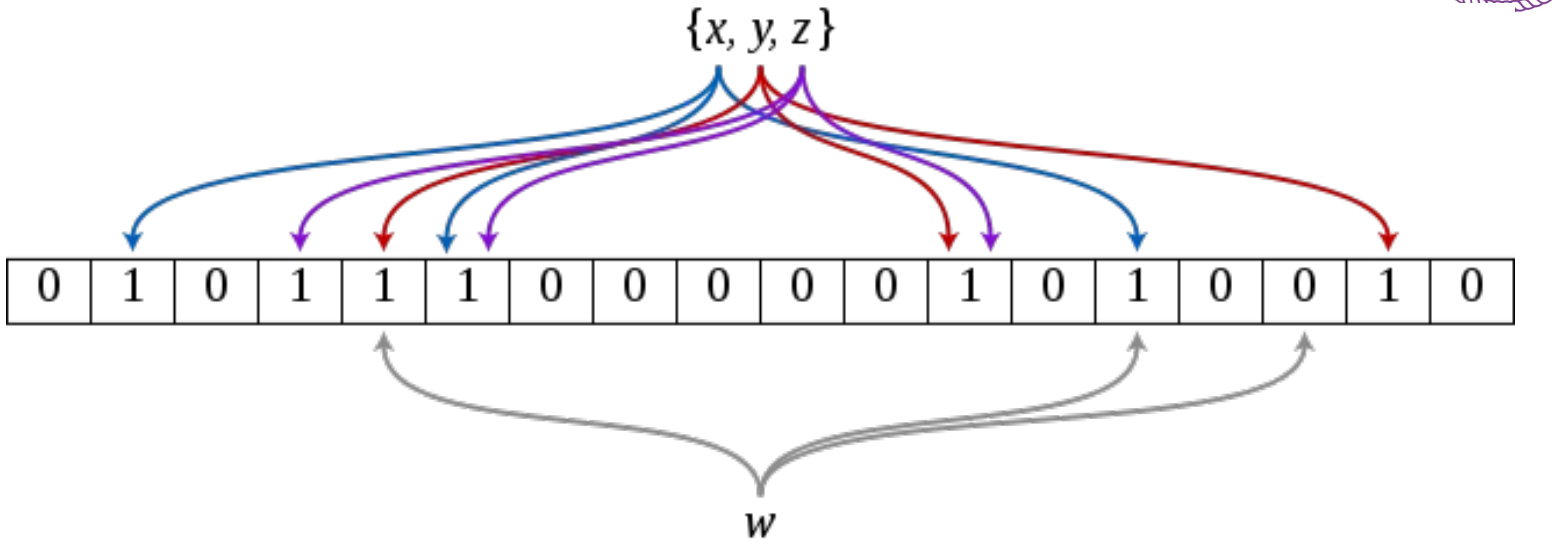
数据块大小×数据块数=(编码数字+第 1个城市 + 15 个其它城市)×数据块数

((1+15×2)×6 位+7×1 字节+15×(7-3)×1 字节)×62500**≈5.4 MB**

OLAP数据库索引

➢列存储索引

- 区域图索引
  - 存该区域的最小最大值
- 布隆过滤器索引
  - 存该区域的布隆过滤器



用户表 *T*

uid	name	
1	Alice	文件 1
2	Max	
3	Bob	
4	Neil	文件 2
5	Thomas	
6	Florian	

(a) 基于元数据的剪枝

元数据 M1

TIMESTAMP: Mon, 01 May
Added: 文件 1, 文件 2
Deleted:
Stats:
-uid: 1-3, 4-6
-name: A-M, F-T

SQL
Select * from T where uid=2
ID: "8e5d0ca9-005e-44e6"

M2

TIMESTAMP: Tue, 02 May
Added: 文件 1, 文件 2
Deleted: 文件 2
Stats:
-uid: 1-3, 4-6
-name: A-M, F-T

DDL
Create table T2 clone table T
ID: "8e5d0ca9-005e-44e6"

(b) 零克隆技术

绝对时间点查询

```
Select * from T AT (TIMESTAMP=>
"Mon, 01 May 2022":: timestamp)
```

相对时间点查询−5 分钟前

```
Select * from T AT (OFFSET=>−60×5)
```

基于声明 ID 的查询

```
Select * from T BEFORE (STATEMENT=>
"8e5d0ca9-005e-44e6")
```

(c) 历史数据版本查询

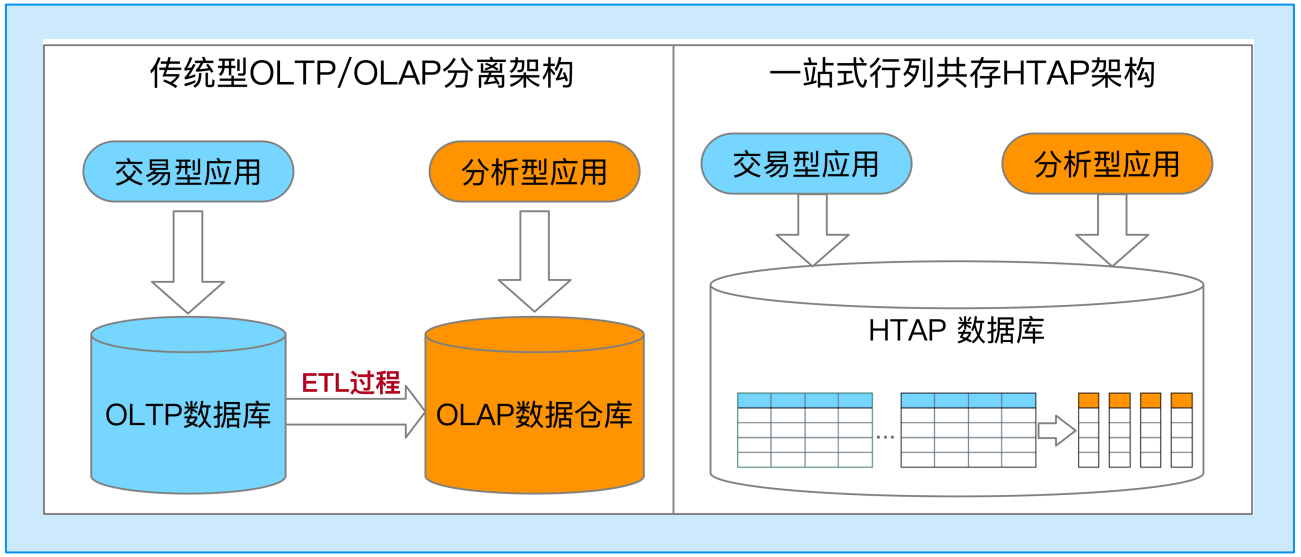
03 HTAP数据库

Survey: http://dbgroup.cs.tsinghua.edu.cn/ligl/papers/HTAP-software-2022.pdf

Tutorial: http://dbgroup.cs.tsinghua.edu.cn/ligl/papers/sigmod22-htap-slides.pdf

HTAP数据库

- **HTAP：Hybrid Transactional Analytical Processing（混合事务与分析处理）**
  - 统一支持OLTP和OLAP场景
  - 常用场景：在线电商平台、金融反欺诈系统等
- **HTAP相比于传统OLTP与OLAP技术的优势**
  - 更简单的数据准备过程：避免了耗时的ETL操作
  - 更快的数据处理与分析：使OLTP与OLAP负载都可以被快速执行与响应
  - 更高效的数据管理方式：使用统一的系统管理数据



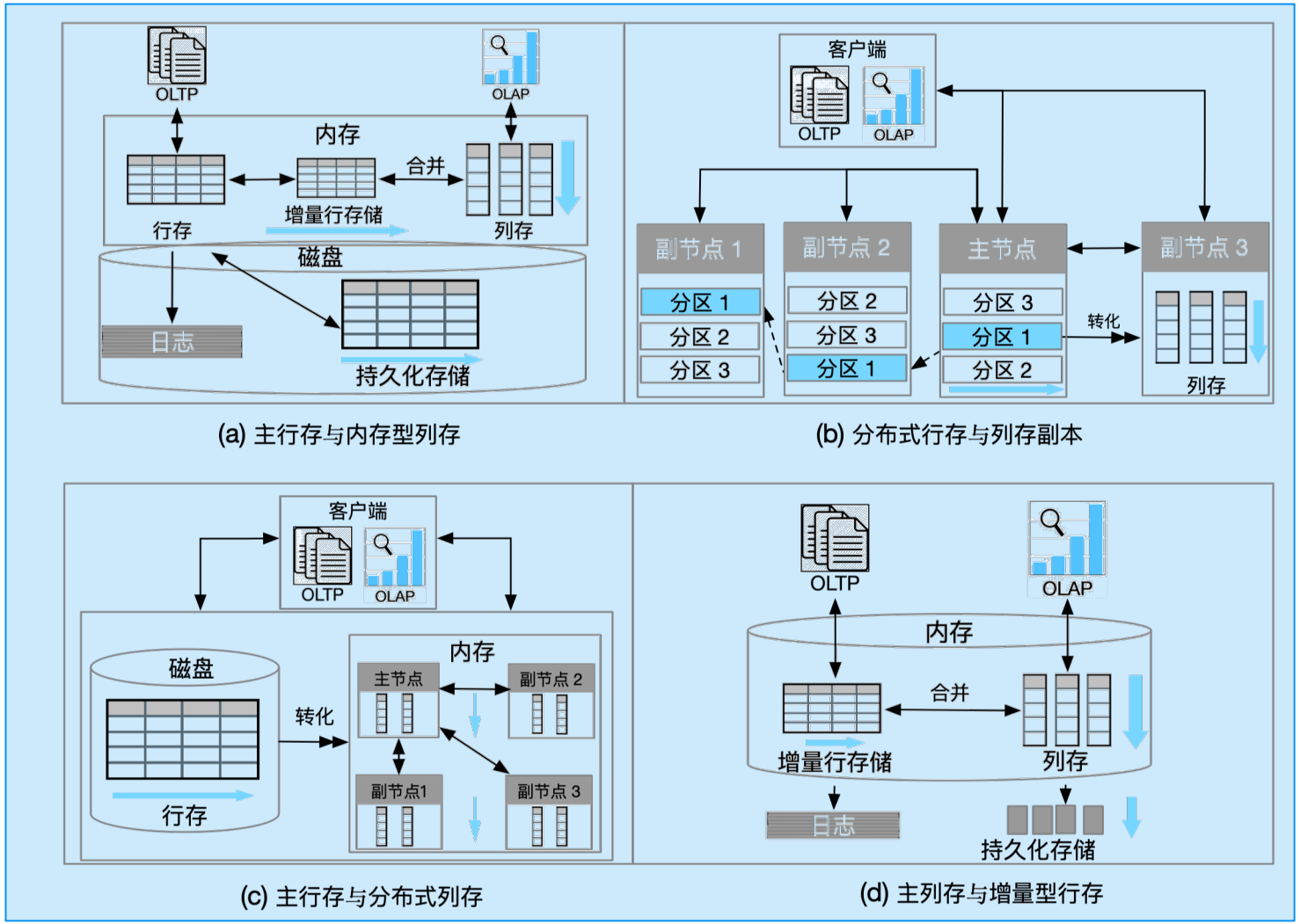
HTAP数据库评价指标

- **数据新鲜度：OLTP和OLAP数据之间是否有时差**
  - OLTP行存转到OLAP列存往往有时差
  - OLAP是否能读到最新数据
- **系统隔离性：OLTP和OLAP是否隔离**
  - OLAP不影响OLTP
- **业务透明度：是否对业务透明**
  - 数据新鲜度为0透明，否则不透明
  - 能否自动选择用行存还是列存
- **业务吞吐量和时延**
  - OLTP需要时延短、高并发

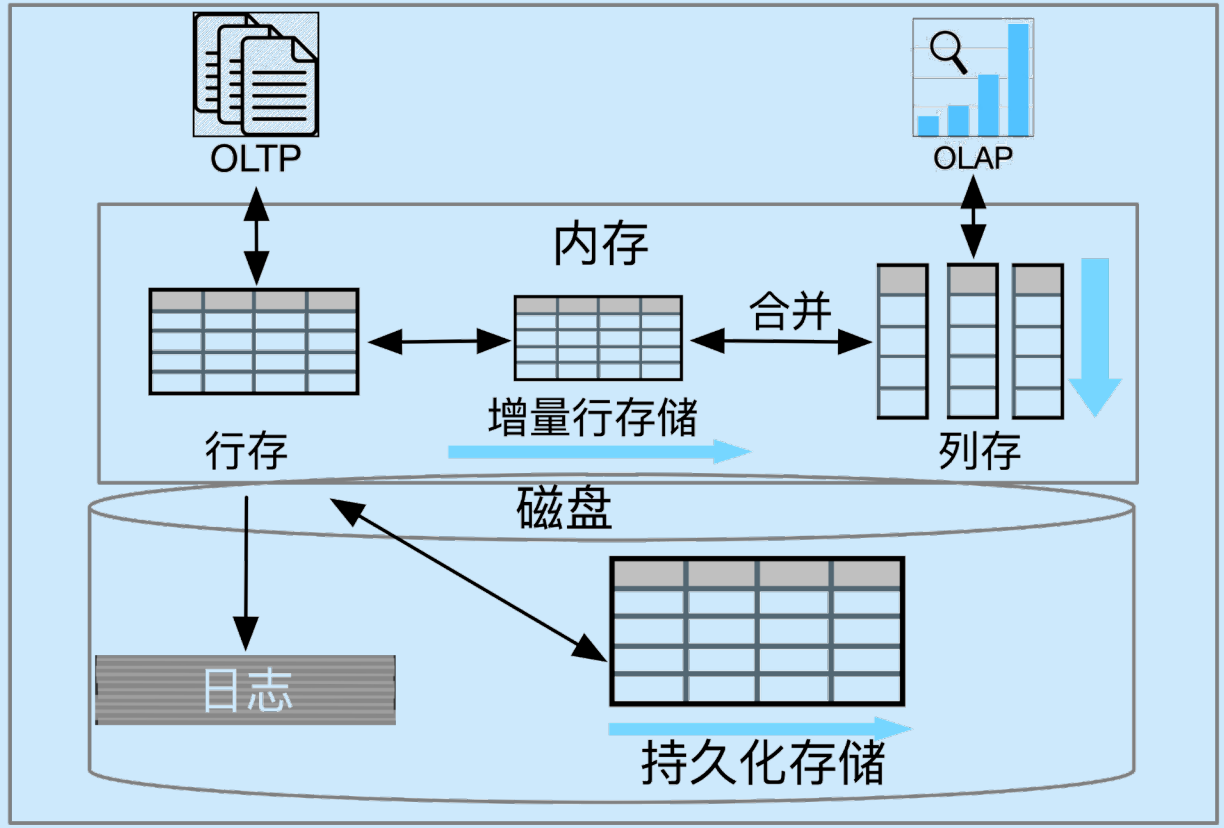
HTAP数据库的存储架构

➢ HTAP存储架构

- 主行存储与内存型列存储架构
- 分布式行存储与列存储副本架构
- 主行存储与分布式列存储架构
- 主列存储与增量型行存储架构



主行存与内存型列存



(a) 主行存与内存型列存

优点：

- OLTP吞吐量高
- OLAP吞吐量高
- 数据新鲜度高

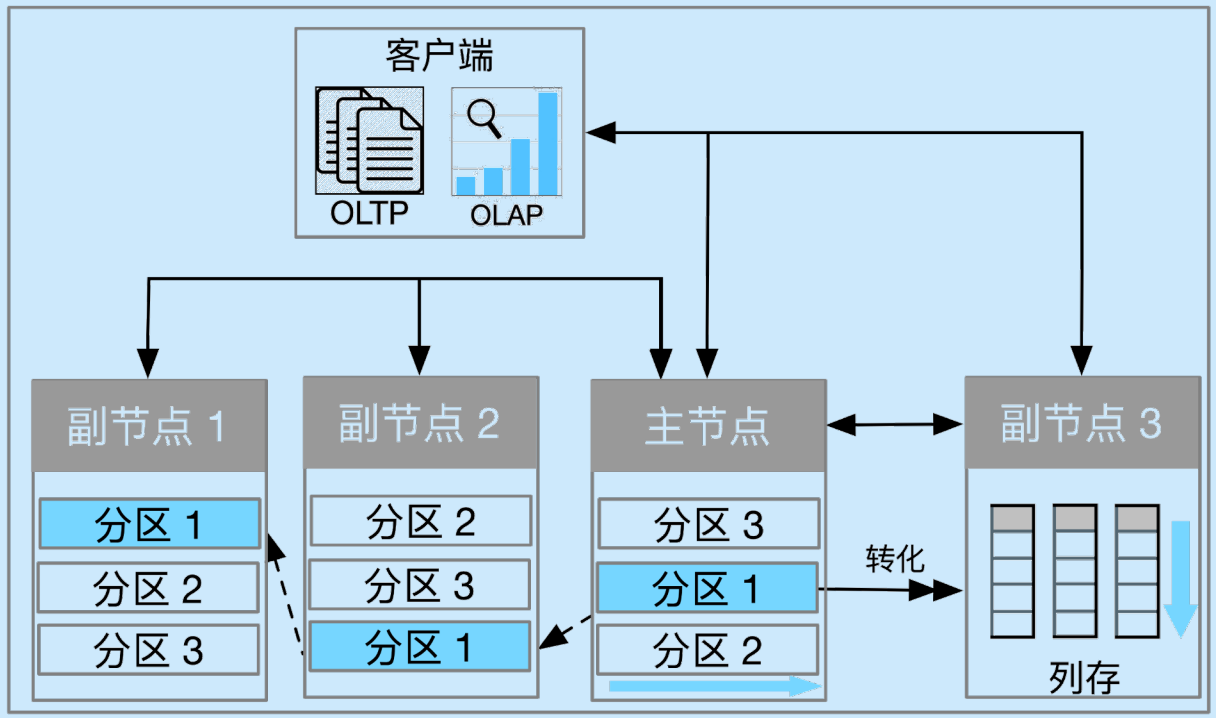
缺点：

- OLAP扩展性低
- 负载隔离性低

应用场景：高吞吐量与低扩展性应用

如银行金融业（事务处理与实时报表）

分布式行存与列存副本



(b) 分布式行存与列存副本

优点：

- 负载隔离性高
- 系统扩展性高

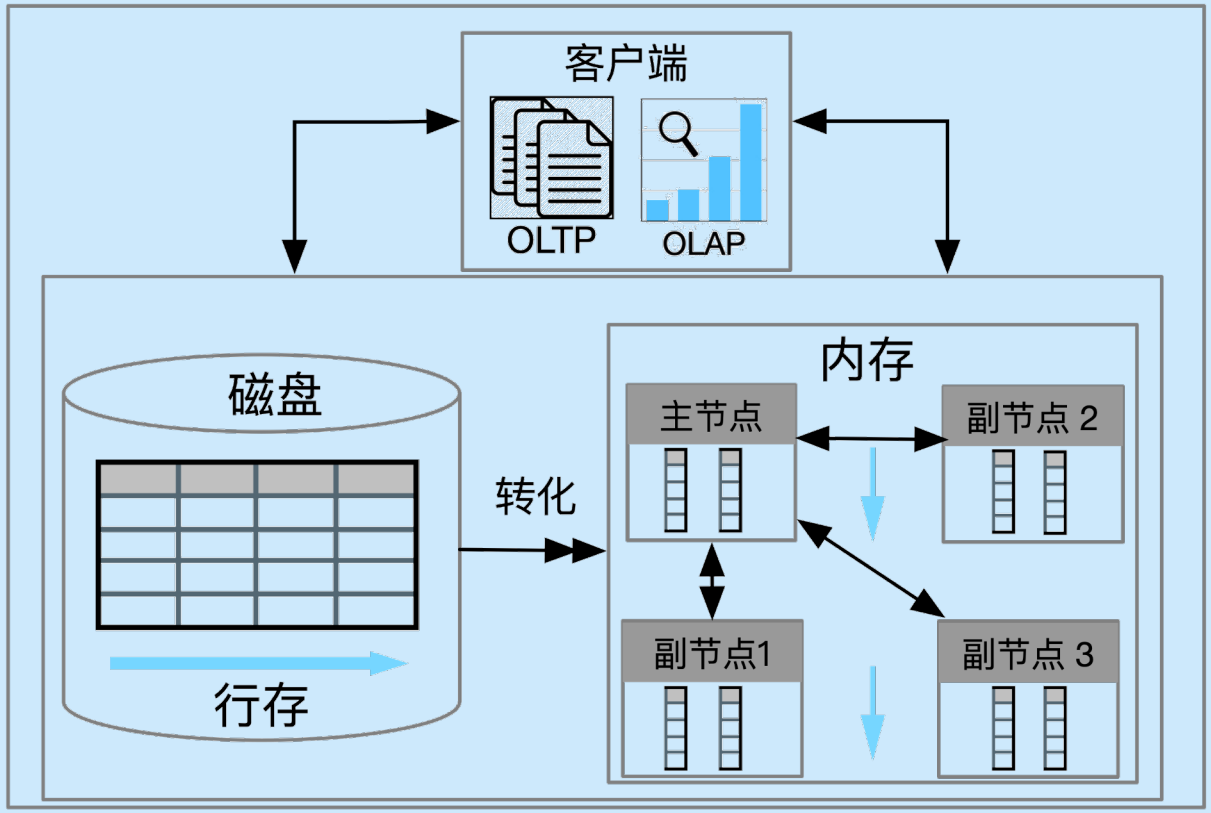
缺点：

- 数据新鲜度低

应用场景：高扩展性与秒级分析时延

如大型电子事务（事务处理与T+0报表）

基于主行存储与分布式列存



(c) 磁盘型行存与分布式列存

优点：

- 负载隔离性高
- OLAP吞吐量高
- OLAP扩展性高

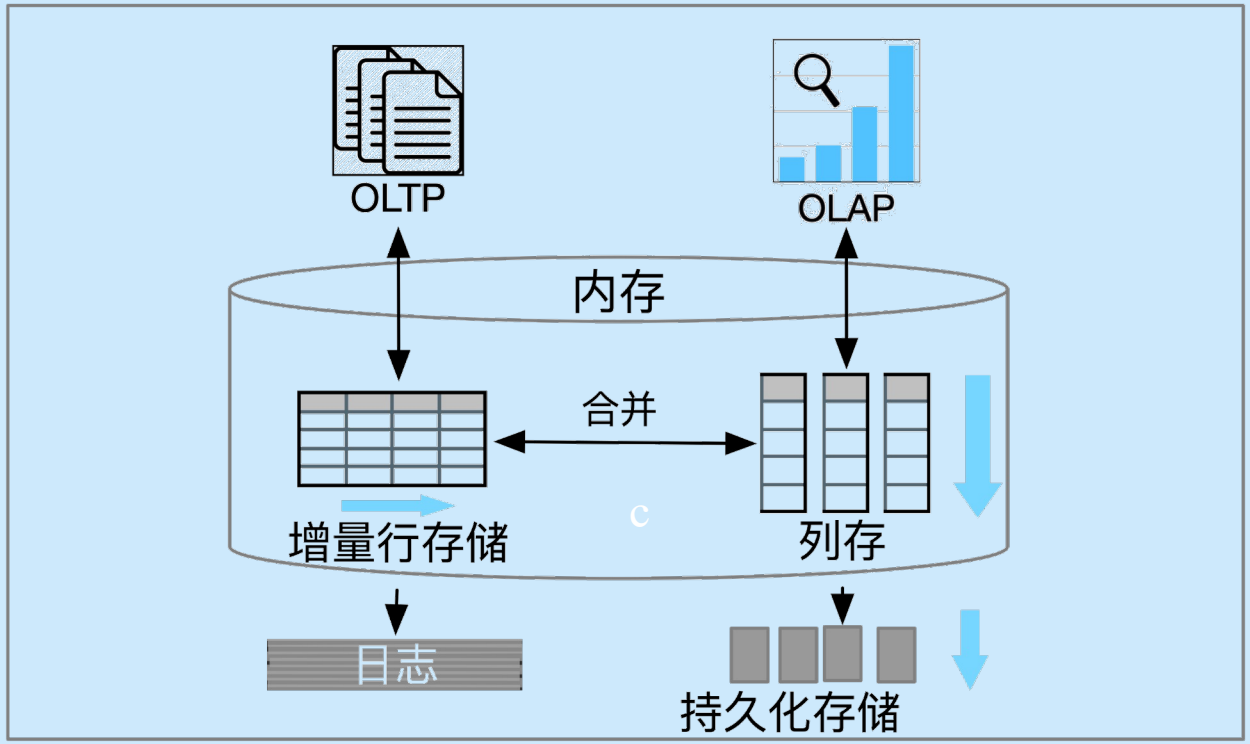
缺点：

- 数据新鲜度低

场景：高AP吞吐量与低扩展性应用

如实时分析型物联网应用

主列存与增量型行存



(d) 主列存与增量型行存

优点：

- 数据新鲜度高
- OLAP吞吐量高

缺点：

- OLTP扩展性低
- 负载隔离性低

场景：高AP吞吐量与高数据新鲜度

如实时欺诈与异常检测应用

HTAP数据库的存储架构

存储架构	代表性系统	OLTP		OLAP		隔离性	新鲜度
		性能	扩展性	性能	扩展性		
主行存储与内存型列存储架构	Oracle, SQL Server, DB2	高	中	高	低	低	高
分布式行存储与列存储副本架构	TiDB, SingleStore	中	高	中	高	高	低
主行存储与分布式列存储架构	MySQL Heatwave	中	中	中	高	高	中
主列存储与增量型行存储架构	SAP HANA	中	低	高	中	低	高

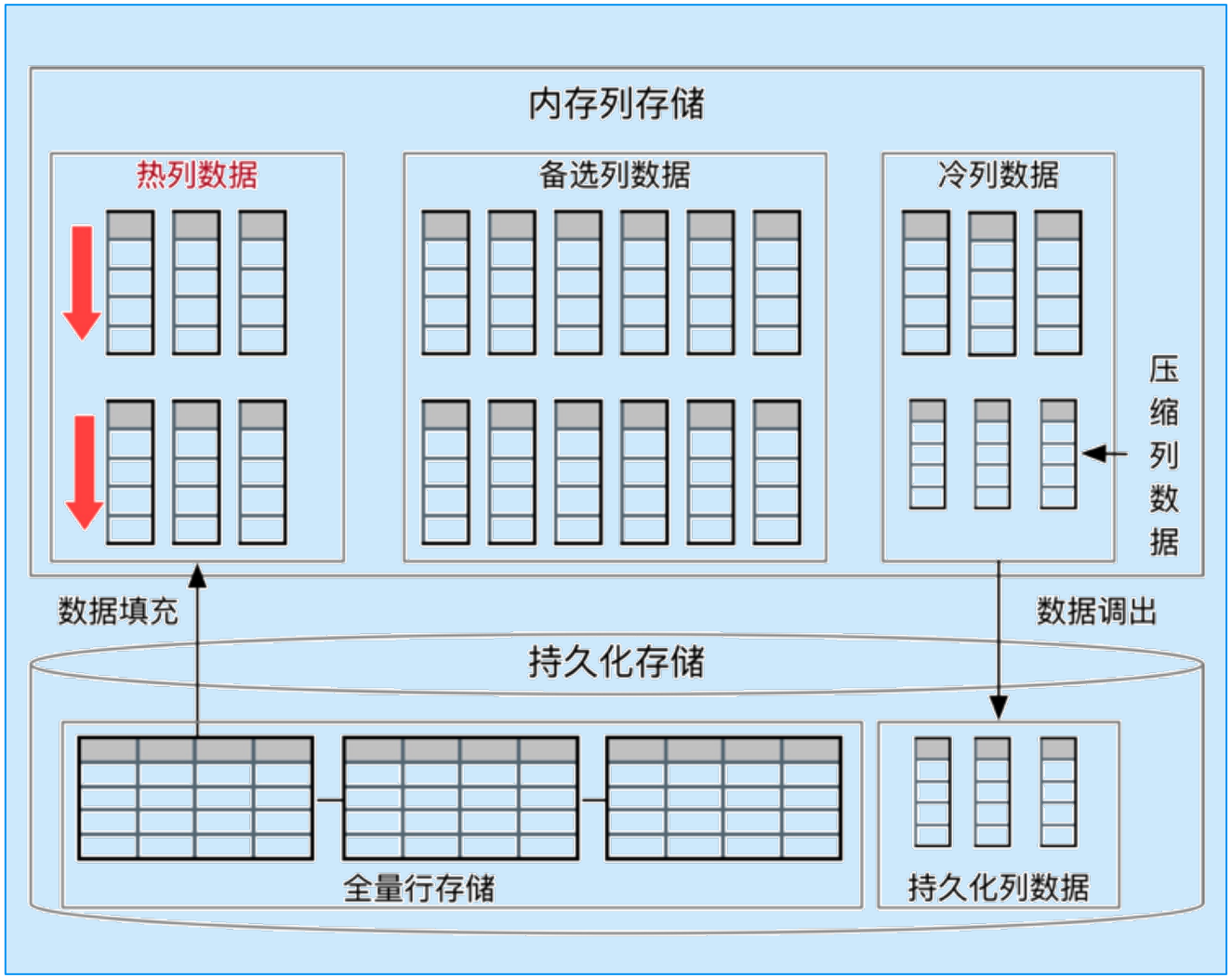
HTAP数据库关键技术

HTAP 技术类别	关键技术	代表性工作	主要优点	主要缺点
数据组织技术	基于主行存的内存列选择	MySQLHeatwave[9], Oracle[10], [11]	事务性能高	分析性能低
	基于负载驱动的行列混合存储	[12-15]	存储代价低	系统复杂度高
数据同步技术	基于内存增量表与内存型列存的数据同步	Oracle[3], SQL Server[4], SAP HANA[6]	性能高	扩展性低
	基于增量日志与持久化列存的数据同步	TiDB[7], F1 Lightning[8]	扩展性高	合并代价高
查询优化技术	混合行/列存储扫描	TiDB[7], Oracle[3] SQL Server[4]	分析性能高	搜索空间大
	异构 CPU/GPU 硬件加速	RateupDB[17], Caldera[16]	分析性能高	事务性能低
	面向 HTAP 负载的索引技术	[18], [19]	事务性能高	内存空间大
资源调度技术	基于负载驱动的资源调度	[20], [21]	性能高	新鲜度低
	基于新鲜度驱动的资源调度	[22]	新鲜度高	性能不高

HTAP数据库关键技术

➢ HTAP数据组织

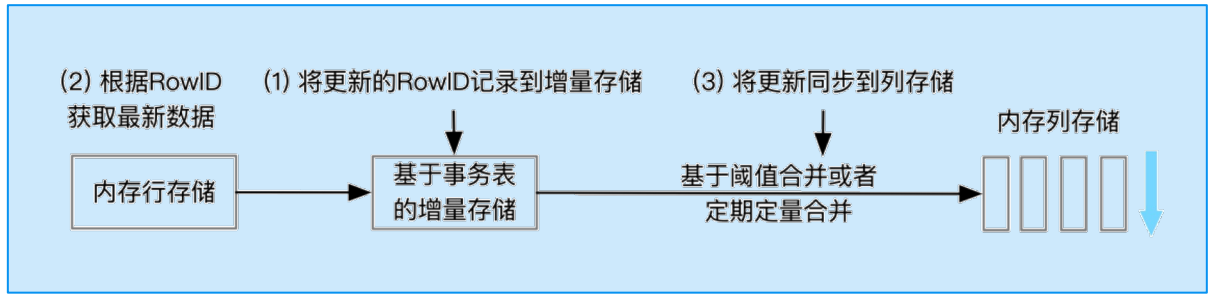
- 基于主行存的内存列选择
- 基于负载驱动的行列混合存储

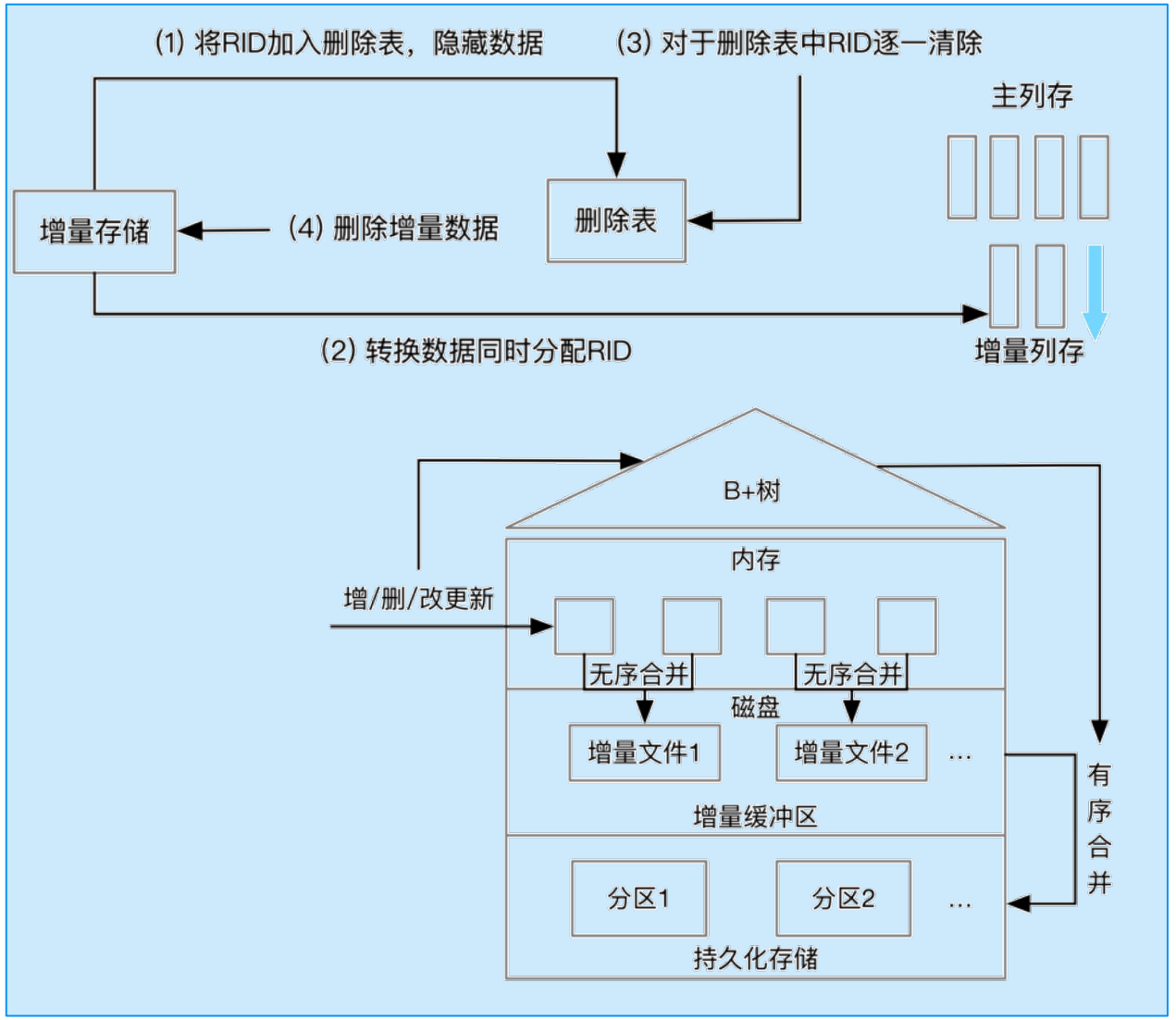


HTAP数据库关键技术

➢ HTAP数据同步

- 基于内存的增量数据合并
  - 在内存中进行数据同步
  - 性能较高、但扩展性较低
- 基于日志的增量数据合并
  - 基于持久化的日志在分布式系统中进行数据同步
  - 扩展性较好、但同步代价较大
- 基于主行存储的列存储重建
  - 通过重建列存储来完成数据同步
  - 不需要过高的内存空间，但重建代价较大

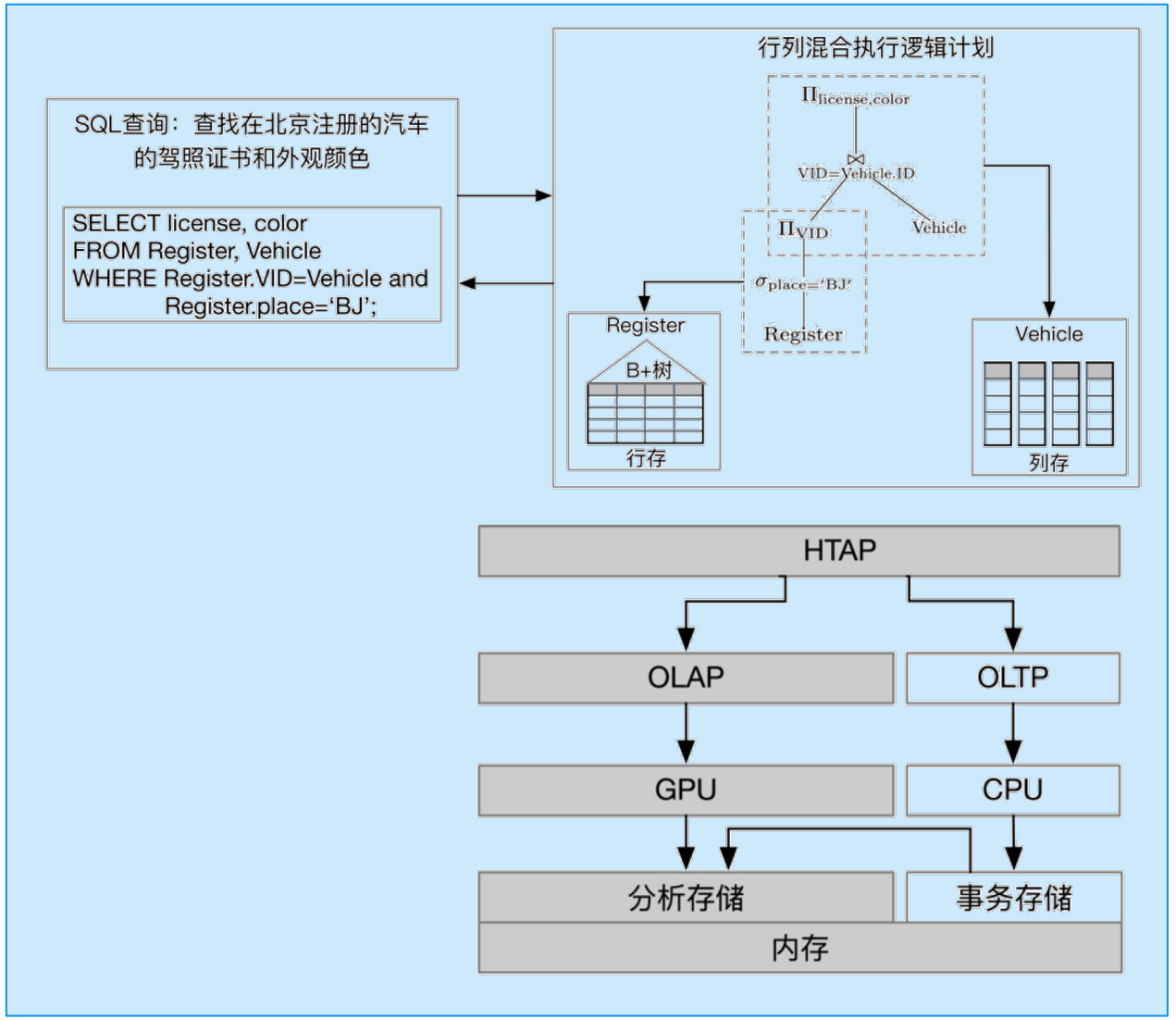




HTAP数据库关键技术

➢ HTAP查询优化

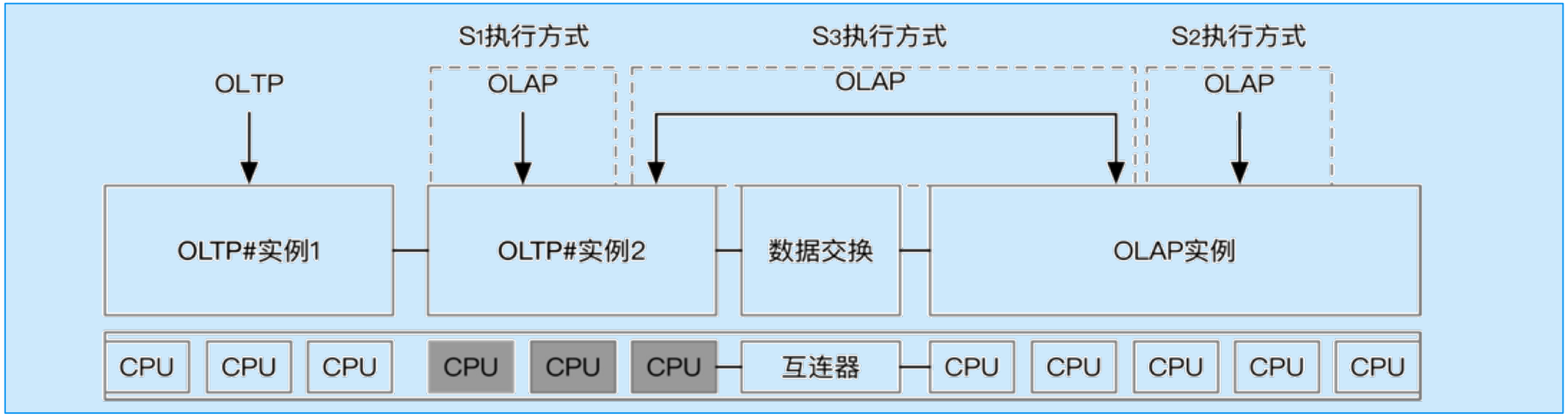
- 内存列存储加速
  - 基于历史负载自动选择列并加载到内存中
  - 当查询命中加载到内存的列时可较好地加速查询、但未命中时分析性能会下降
- 混合行/列存储扫描
  - 通过混合扫描两种存储进行加速
  - 可以利用二者的长处提升分析性能、但算子的搜索空间很大
- 异构CPU/GPU硬件加速
  - 利用硬件进行查询加速
  - 其分析性能较高、但事务处理时的性能会因为资源共享而降低



HTAP数据库关键技术

➢ HTAP资源调度

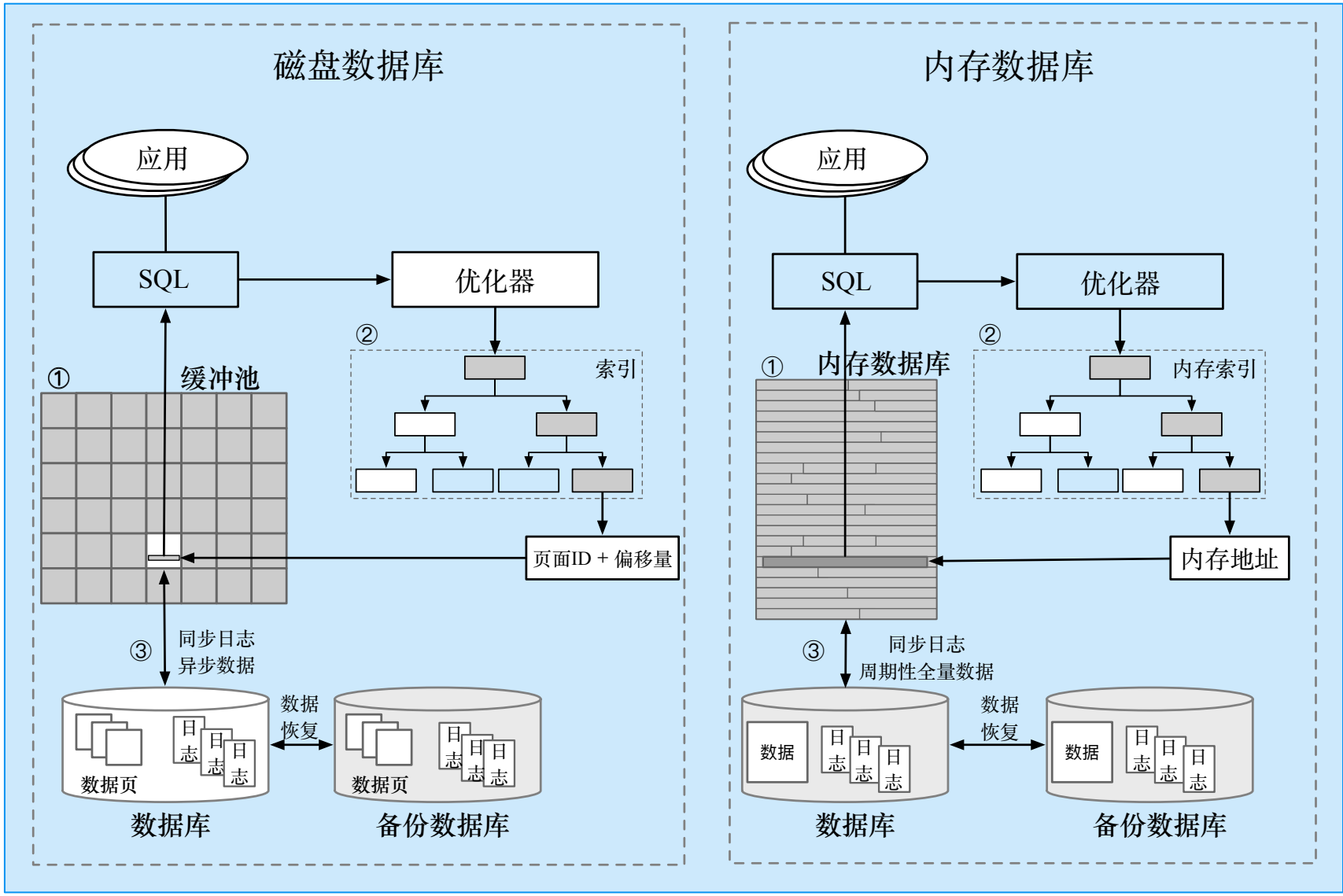
- 基于负载驱动的资源调度
  - 通过监控当前混合负载的执行情况来动态调度系统资源
  - 分析性能较高、但没有考虑到数据分析时的新鲜度
- 基于新鲜度驱动的资源调度
  - 以给定的新鲜度阈值为基准，动态调整资源调度的执行方式
  - 数据分析时的新鲜度有保证、但会牺牲一部分性能



04 内存数据库

内存数据库

- **内存数据库 vs 磁盘数据库**
  - 数据组织格式
    - 页面 vs 行
  - 索引
    - B+树 vs ART树
  - 事务
    - 2PL vs OCC
  - 持久化
    - 日志+更新数据 vs 日志+定期全量数据
  - 恢复
    - 日志 vs 日志+数据+索引



内存数据库

➢ 关键技术

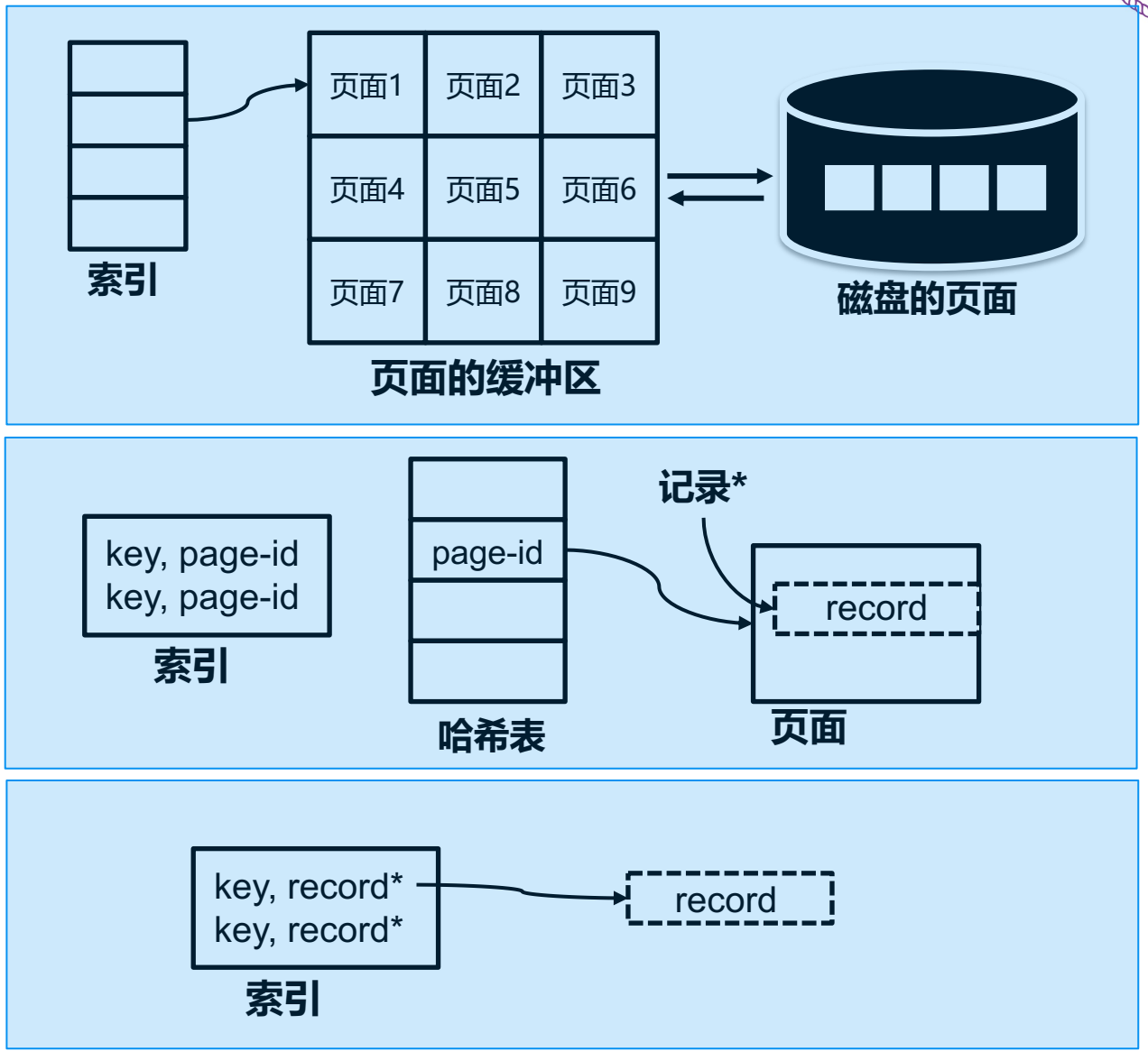
- 数据组织
  - 直接存储在内存中；无需页面，以内存地址标识每条数据
- 索引
  - 直接索引到内存地址
  - 主要关注于优化内存空间的利用率等目标
- 并发控制
  - 大多数数据库使用乐观并发控制技术或多版本机制来实现
- 数据持久化与恢复
  - 需要使用磁盘等非易失性介质以日志等方式进行数据持久化与备份
- 查询执行
  - 选择编译执行的方式将查询编译为优化后的机器代码并直接在内存中执行以提升性能

内存数据库

	磁盘数据库	内存数据库
数据存储	全部数据存储在磁盘中，需要磁盘I/O将数据从磁盘移入内存	全部数据存储在内存中，查询更新数据时无需执行磁盘I/O
数据组织形式	页	记录
数据持久化	使用磁盘进行数据持久化	使用非易失性存储介质数据持久化
索引	使用传统的数据结构，如B树索引，关注I/O效率	使用特殊的数据结构以及索引结构，如T、ART树，关注内存以及缓存的利用率
并发控制	支持多粒度以及多种类型的锁	多使用乐观锁或者多版本并发控制技术
查询处理及执行	基于I/O代价	基于CPU代价及缓存代价
应用场景	常见的工作负载	极低时延的场景（例如计费）

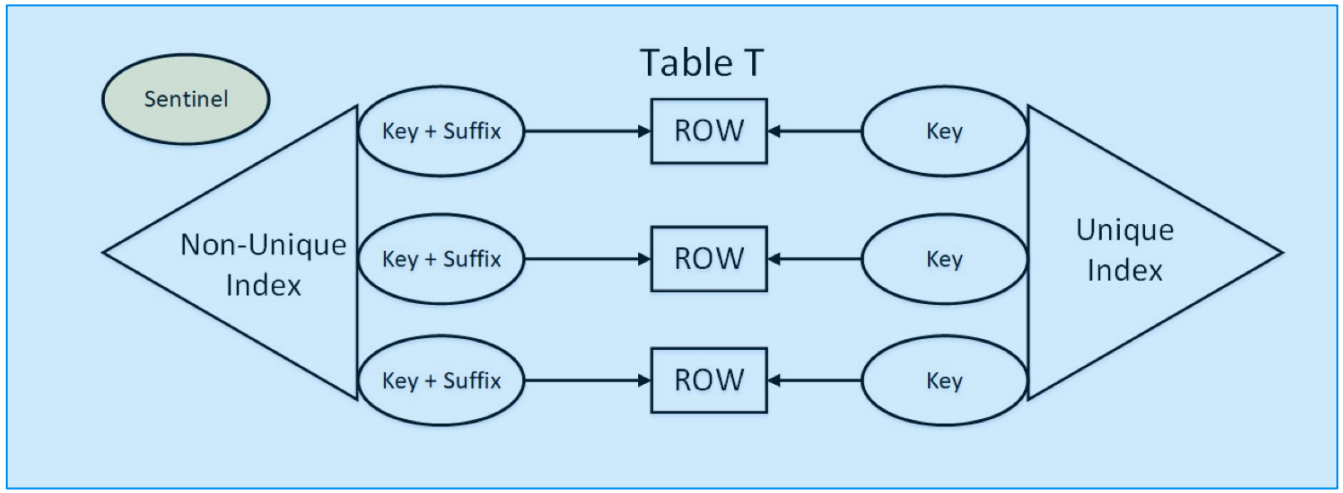
内存数据库的数据组织

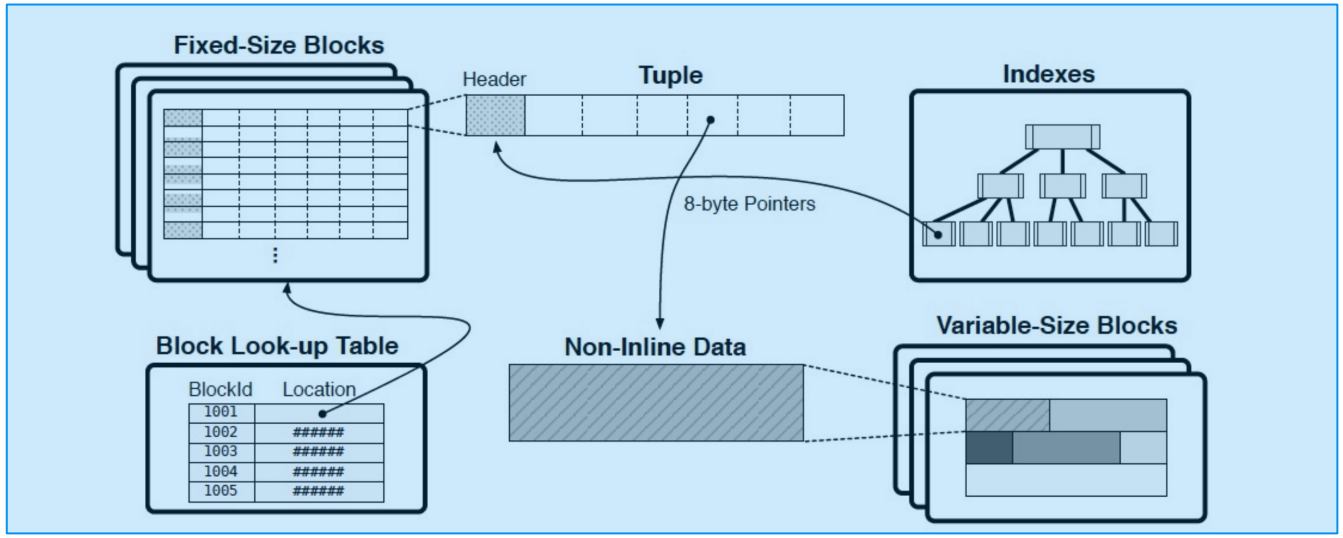
- **磁盘数据库：基于页面的数据组织**
  - 粗粒度冲突管理
  - 页面级冲突
- **内存数据库：基于记录的数据组织**
  - 细粒度冲突管理
  - 行级冲突
- **优化手段**
  - 分区
  - MVCC多版本



内存数据库的数据组织

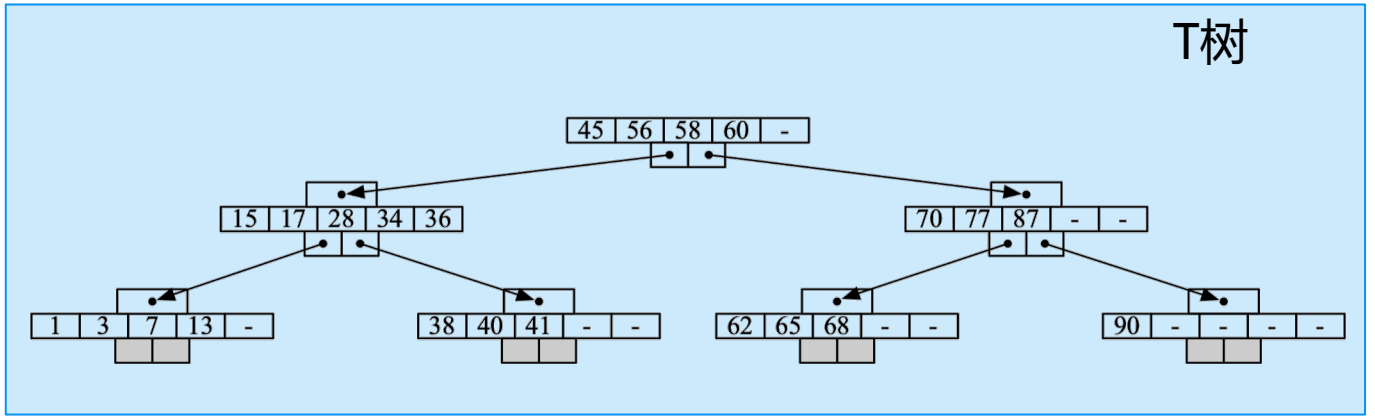
- **以内存地址来访问数据，无需页面**
- **行存储/列存储**
  - OLTP内存数据库：行存储
  - OLAP内存数据库：列存储
  - 混合存储模式
- **分区**
  - 将数据表划分为多个独立的部分
  - 可为分区分配硬件资源
  - 提升数据库的性能与可用性

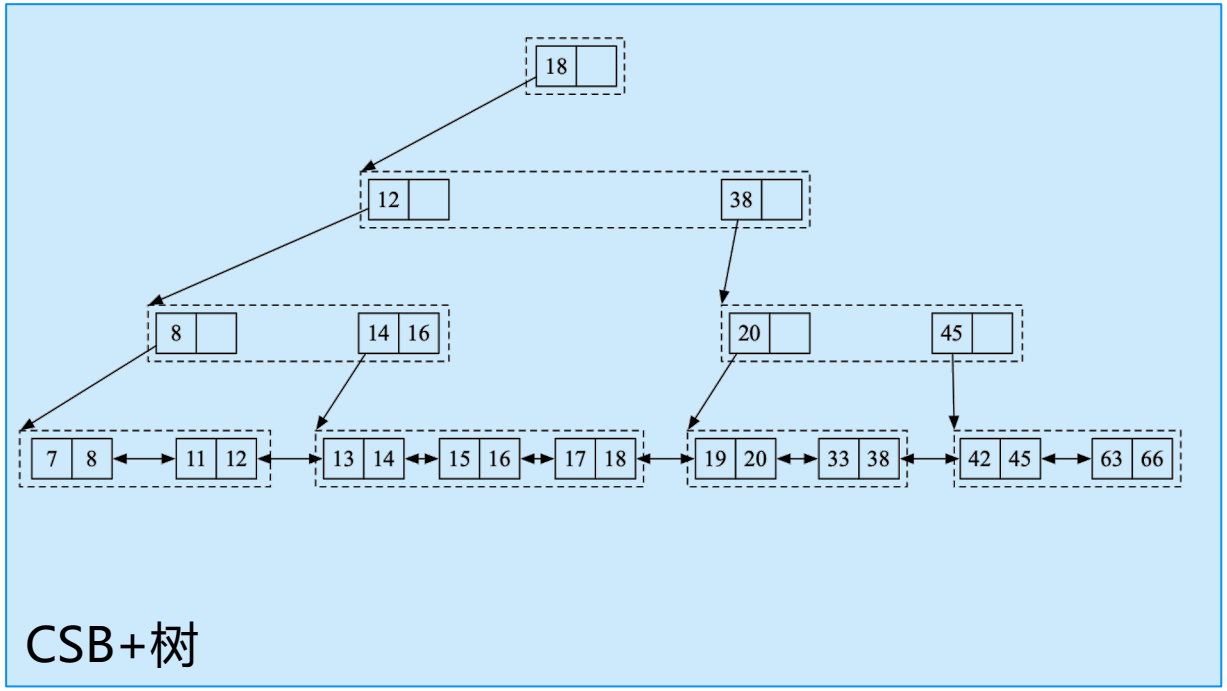


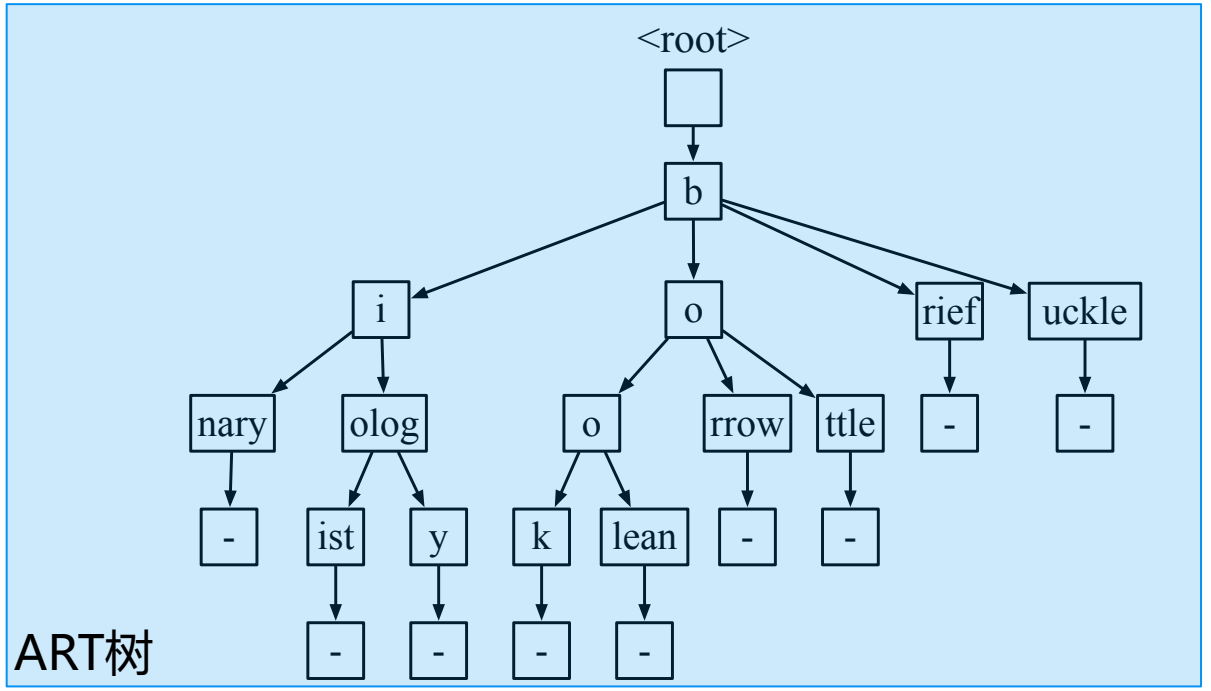


内存数据库的索引

- **常见优化目标：**
  - 提升内存空间利用率：T树、ART树
  - 缓存感知：CSS树、CSB+树
  - 适应新硬件：Mass树

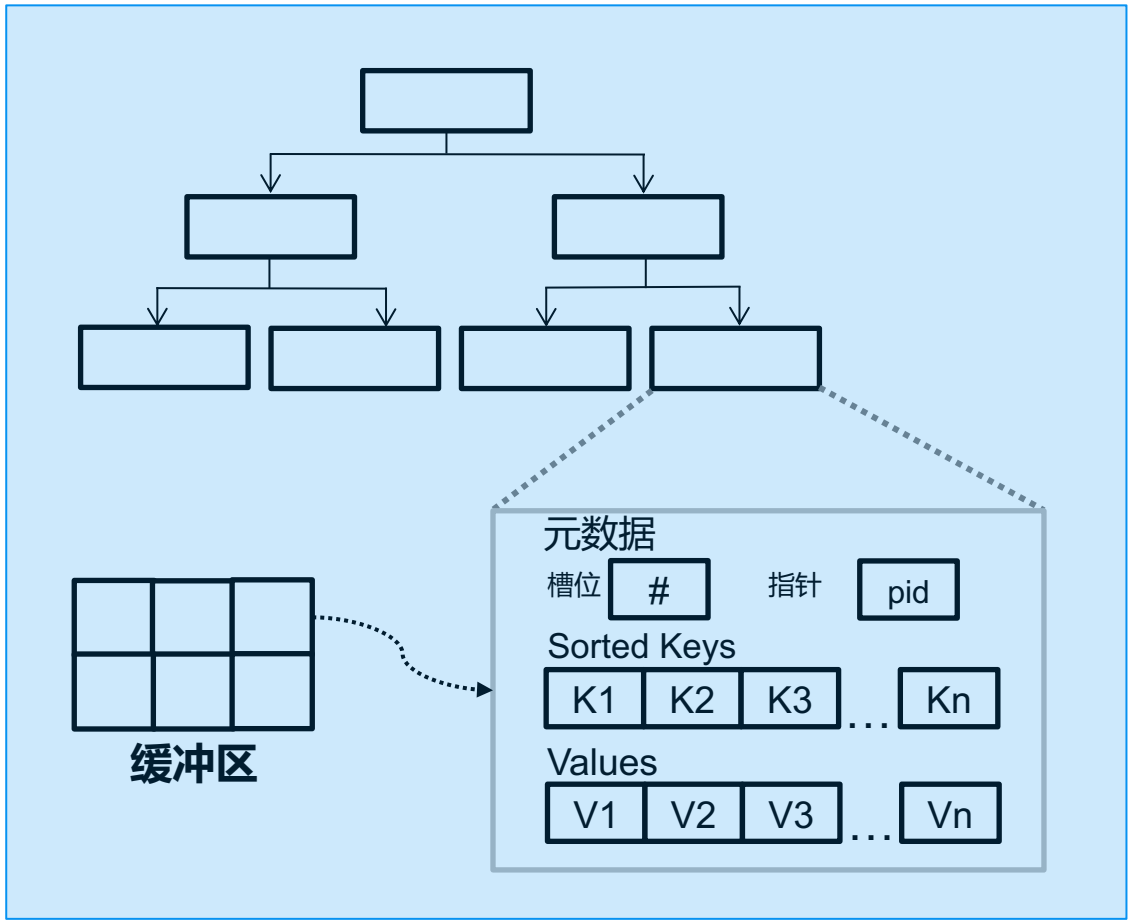


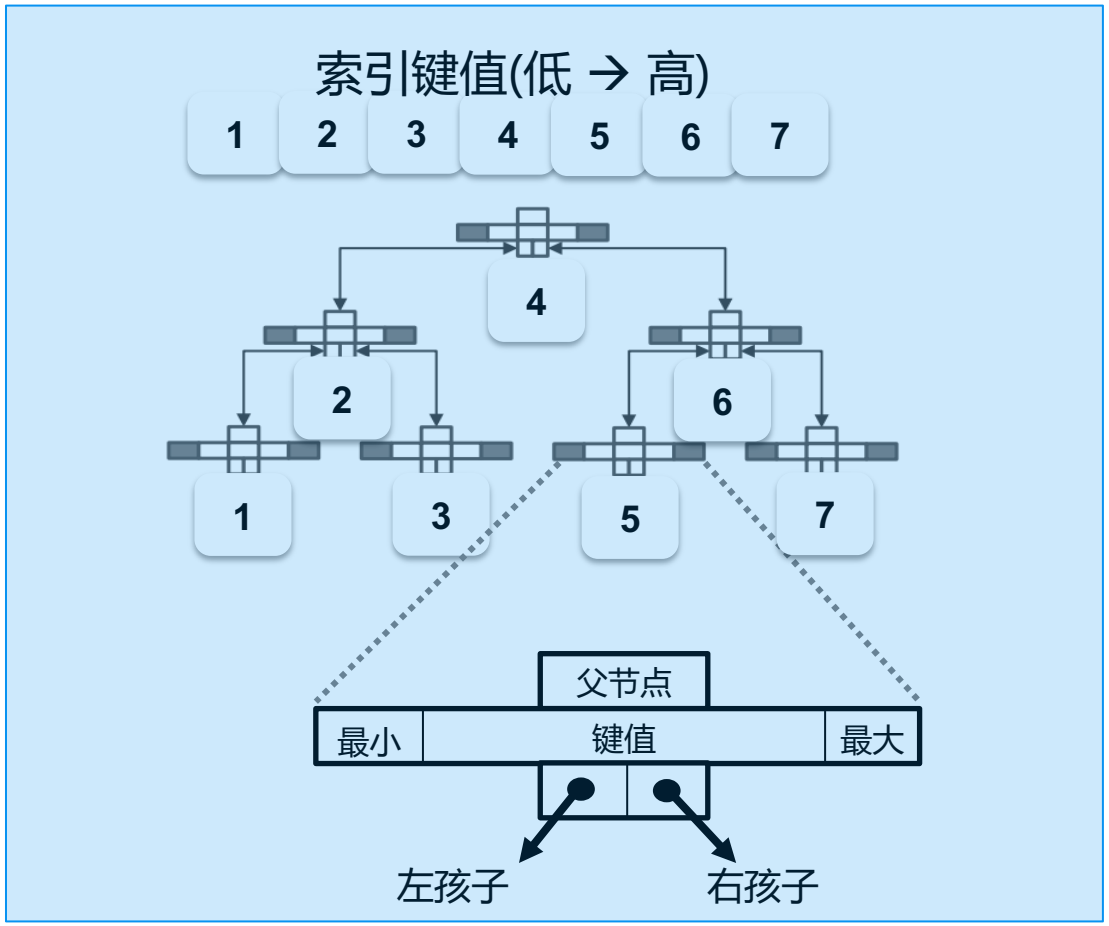




内存数据库的索引：T树

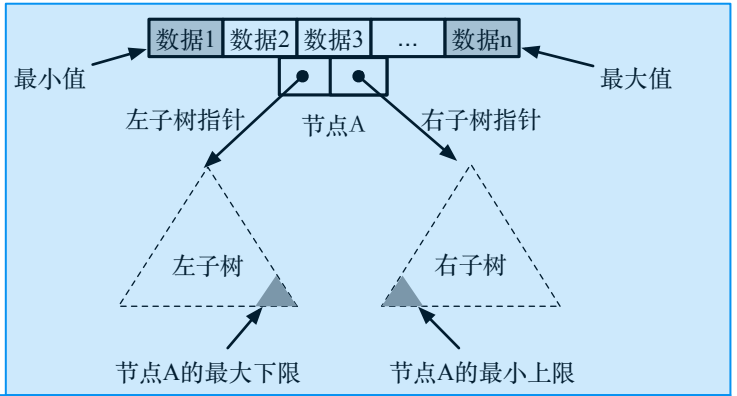
- 更少内存，更多数据
- 更高效

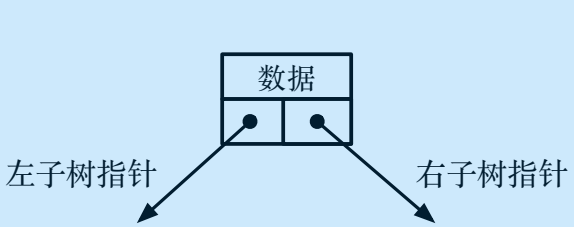




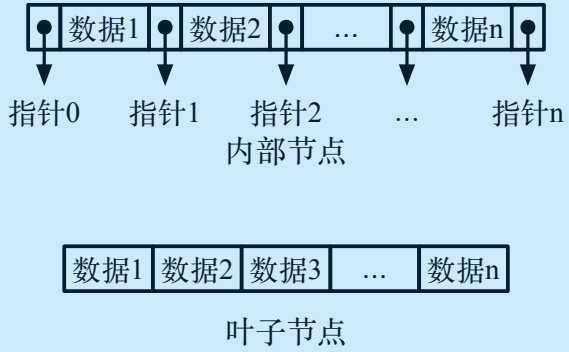
内存索引：T树

- 提升内存利用率，AVL树和B树结合。一种平衡二叉树，每个节点存放多个索引项
- 在T树中，每个节点都按顺序存储了一些关键字，每个节点的第一个值以及最后一个值为该节点的最小值以及最大值。节点中的所有键都在这个上下限的范围之间。
- 每个节点两个孩子节点：
  - 节点左子树包含的键均小于节点A的最小值
  - 节点右子树包含的键均大于节点A的最大值。

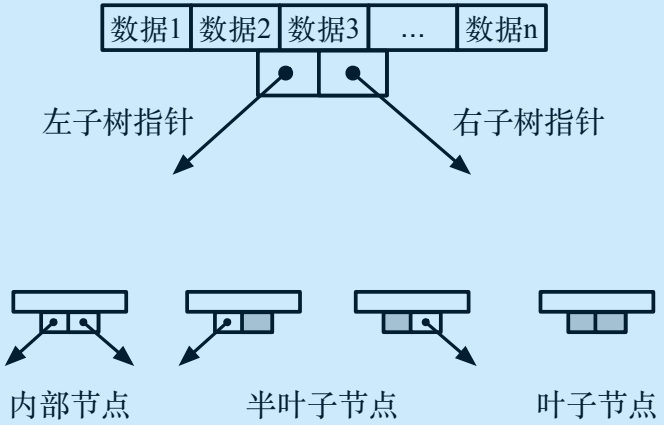




(a) AVL树节点的结构



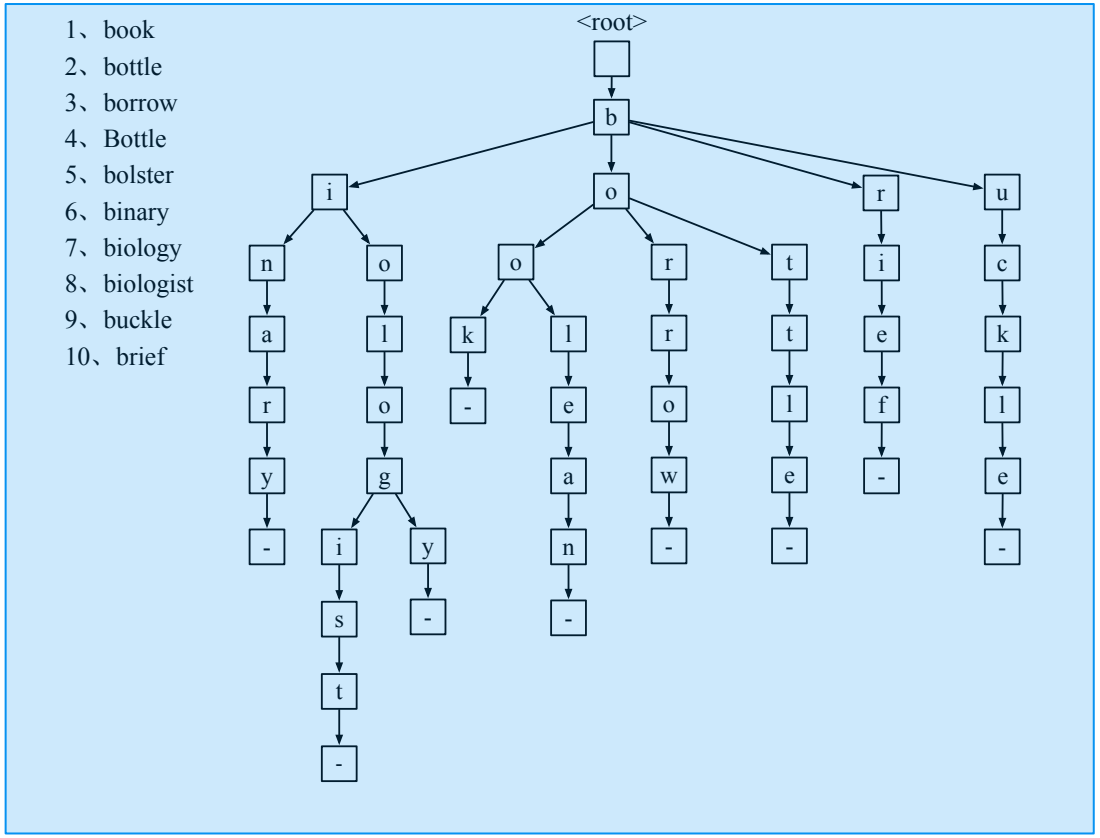
(b) B树节点的结构

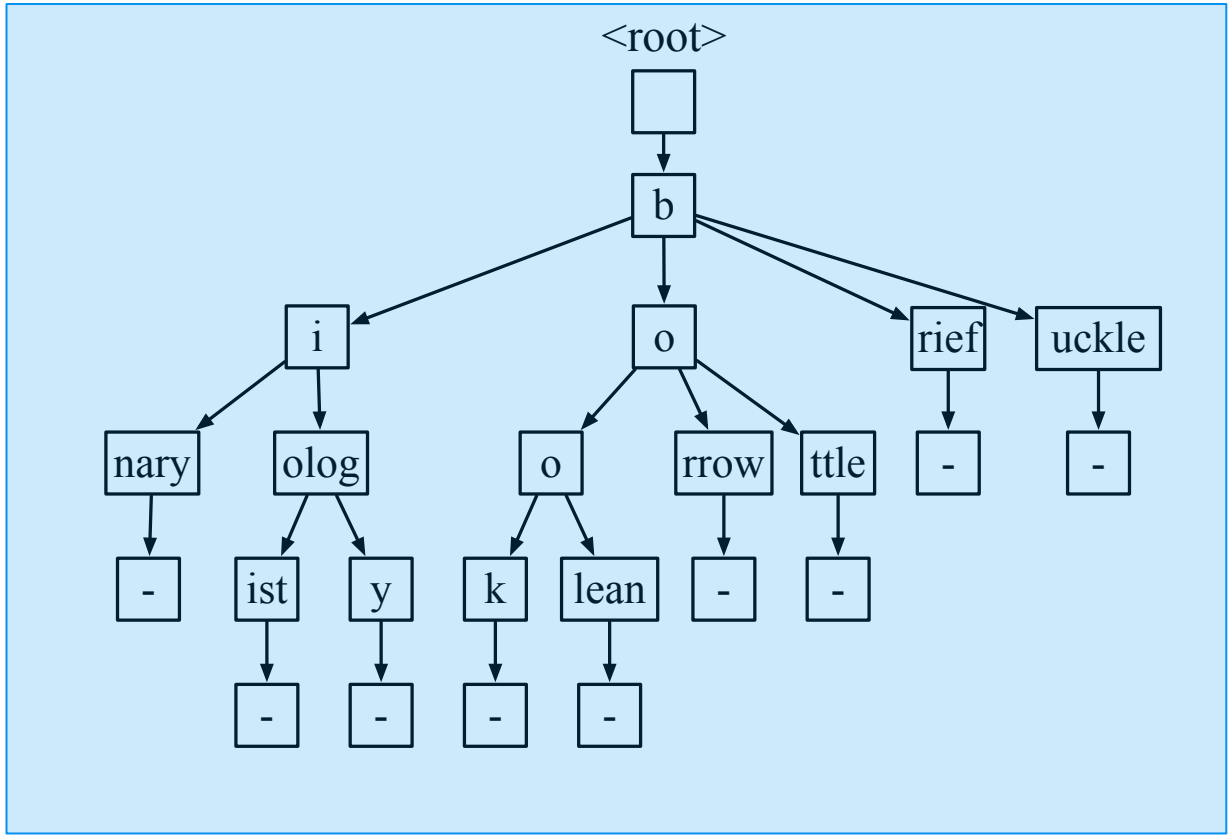


(c) T树节点的结构

内存索引：字典树

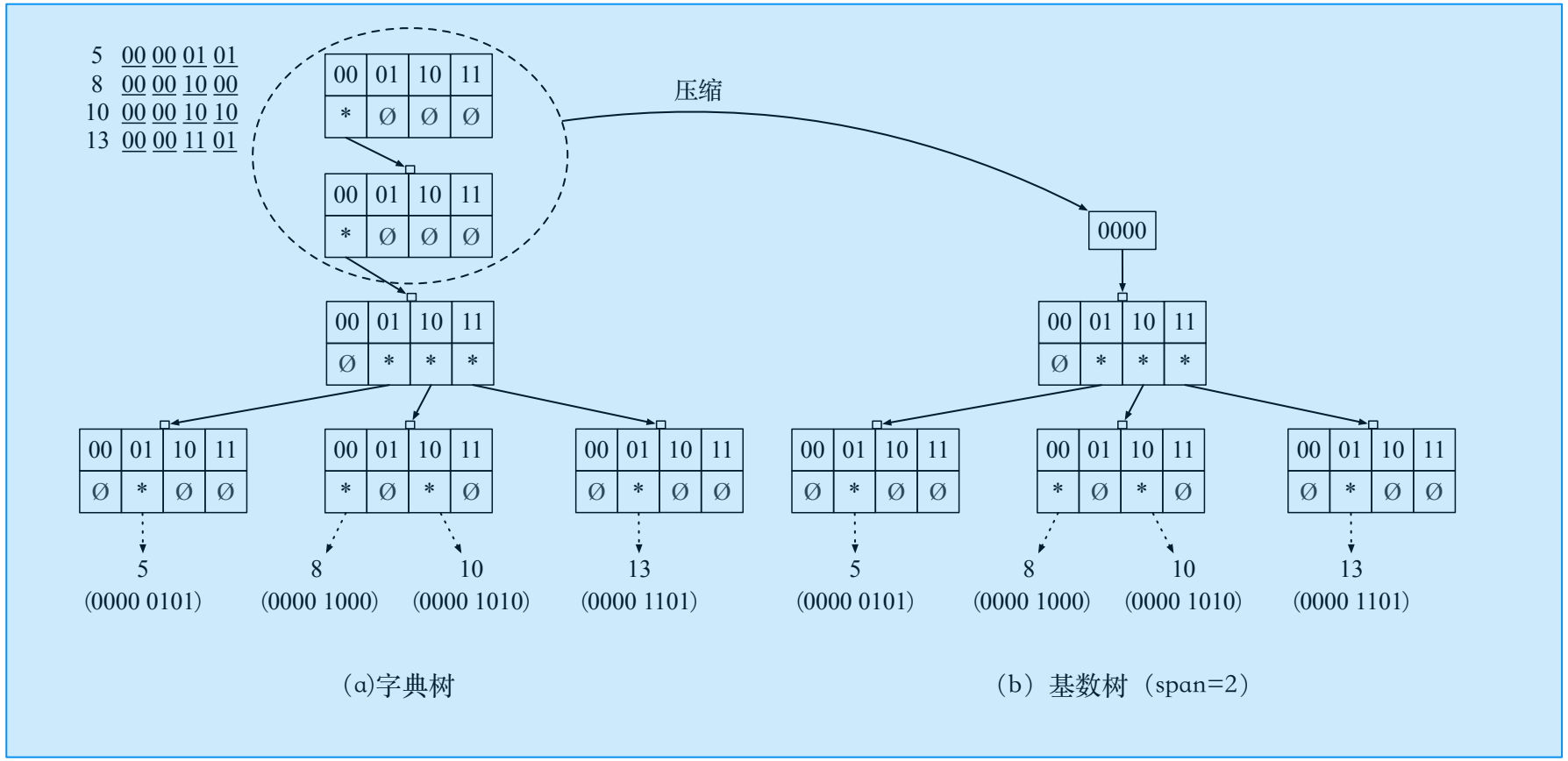
- 字典树
- 基数树（radix tree）是一种按照词典顺序存储键的压缩前缀树





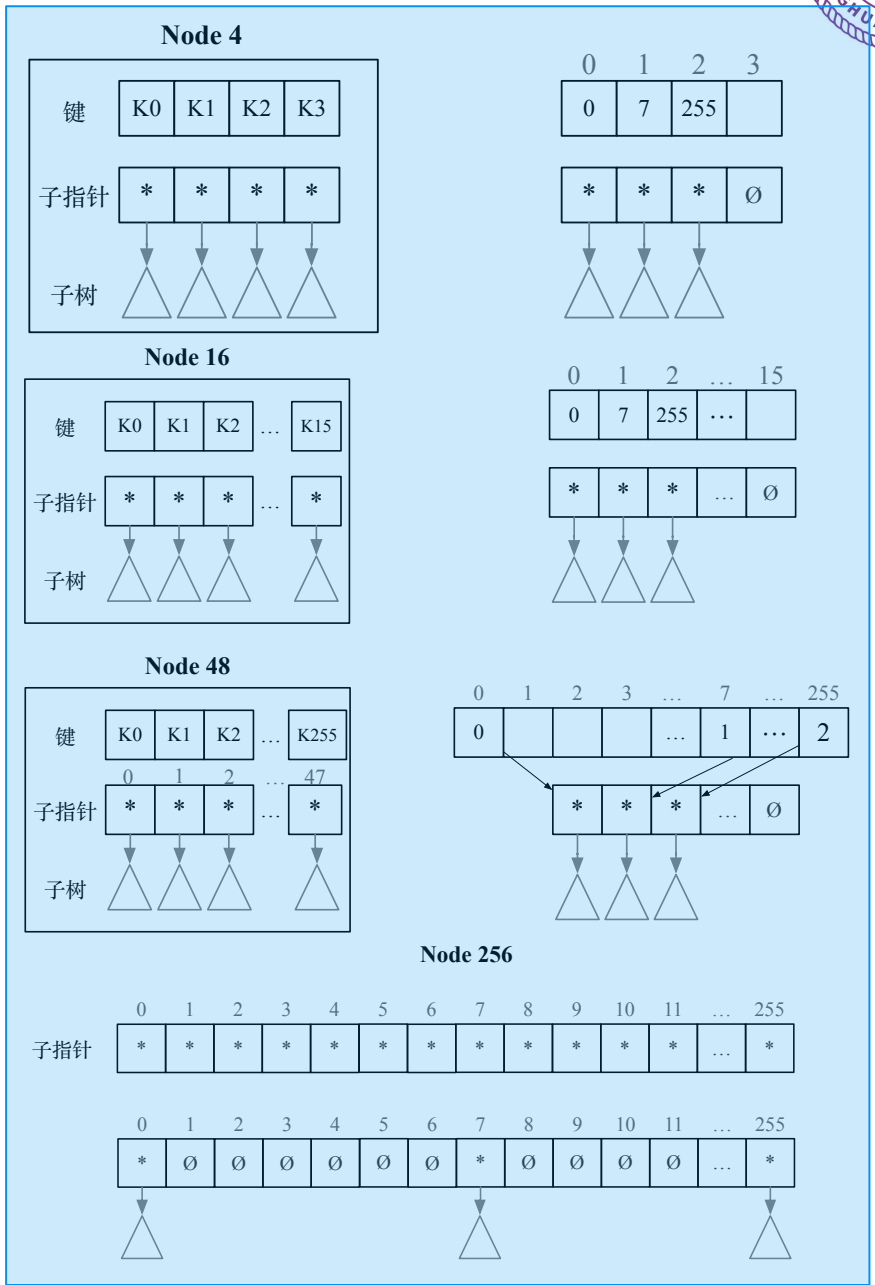
内存索引：压缩的字典树

- 每个节点用固定长度位数来表示
- 孩子节点存在时表示为*，不存在时表示为∅



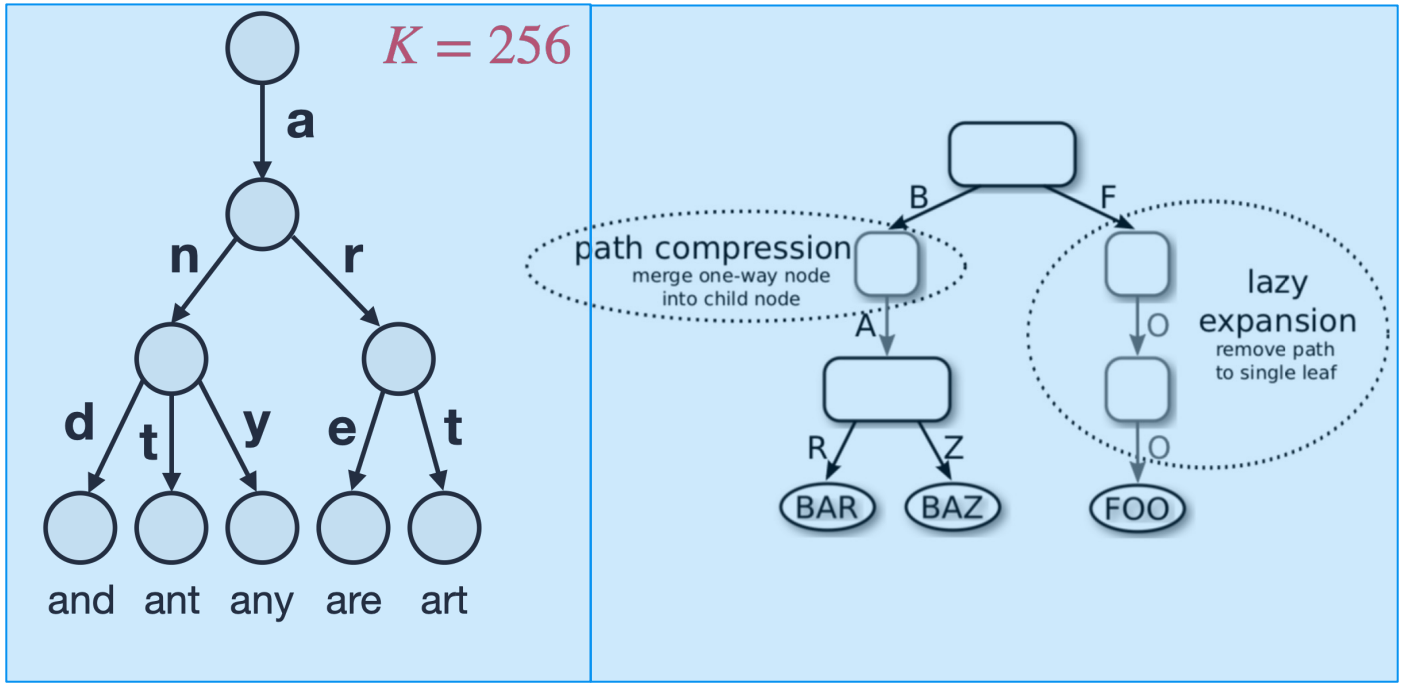
内存索引：ART树

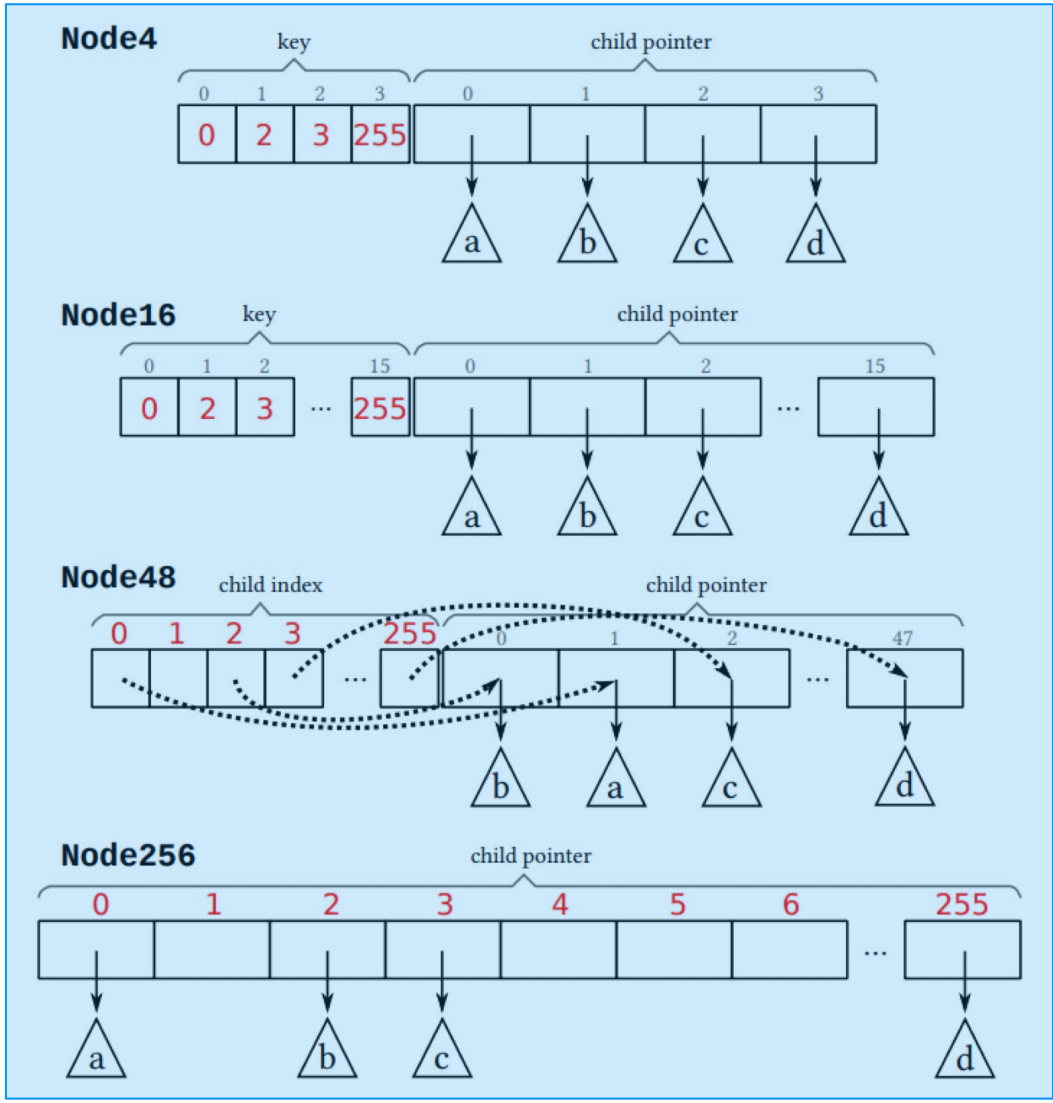
- **自适应基数树（adaptive radix tree，ART树）**
- **一个节点孩子空值比较多，如何处理?**
- **四类节点**
  - Node4：一个key数组(4)，一个child pointer数组(4)
    - key数组存key，child pointer指向孩子
    - 0-4个孩子
  - Node16：一个key数组(16)，一个child pointer数组(16)
    - key数组存key，child pointer指向孩子
    - 5-15个孩子
  - Node64：一个key数组(256)，一个child pointer数组(48)
    - 数组下标是key，数值指向child下标
    - 16-47个孩子
  - Node256：一个key数组(256)
    - 数组下标是key，数值是child指针
    - 48-256个孩子



内存索引：ART树

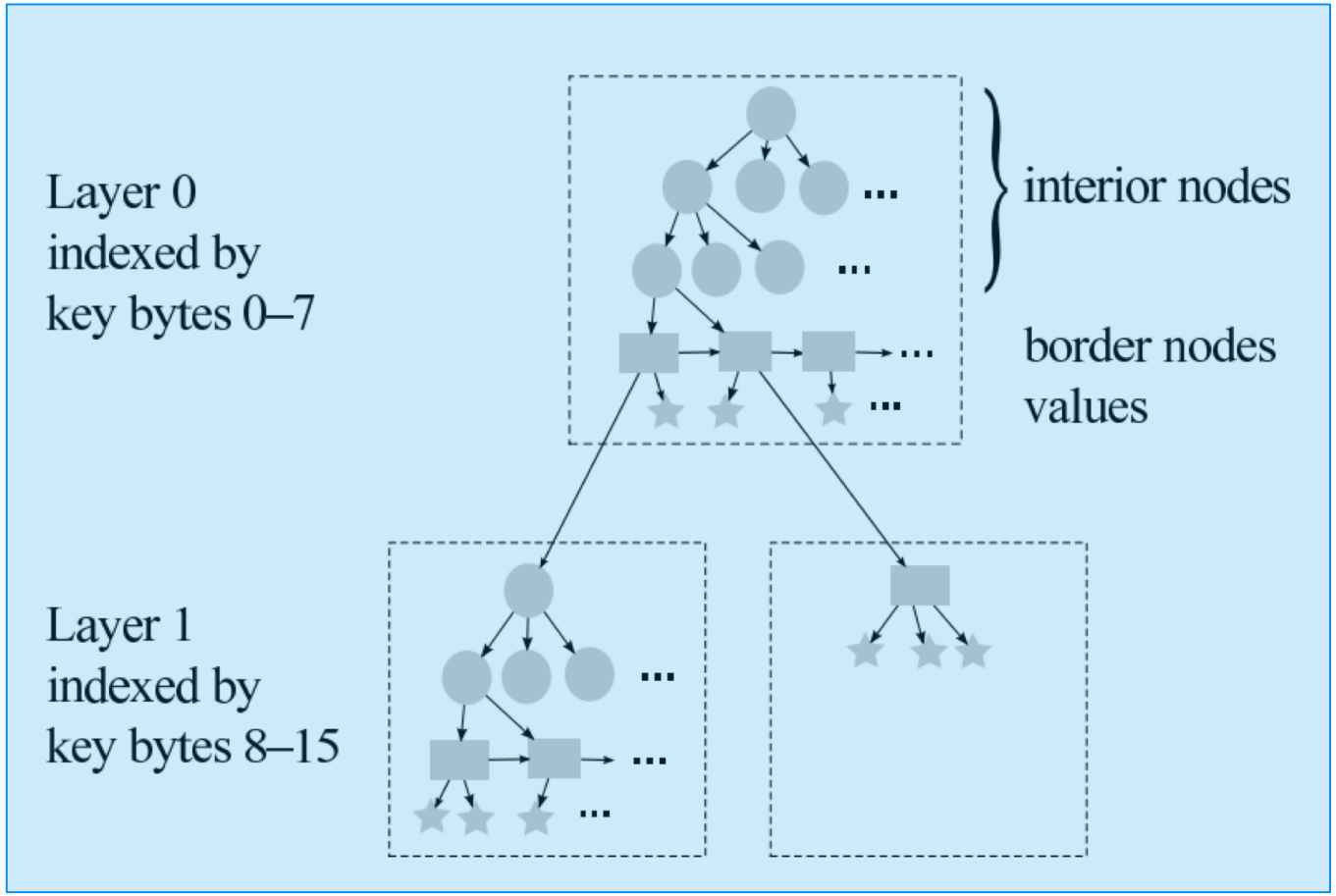
- 减少空值数量
- 一个快速和空间效率高的内存索引结构
- 索引小，速度快
  - SIMD，cache line，page blocking





内存索引：Mass树

- **如何支持任意长度的key?**
- **Mass树**
  - B+树和Trie树和结合体
    - 每层是一个B树，多层组成一个Trie
  - 圆形的节点称为 interior node，
    即 B+ 树中的非叶子节点
  - 矩形称为 border node，
    即 B+ 树中的叶子节点。
  - mass tree 的 border node
    存放的是值，或者下一层 B+ 树的根节点
    - 五角星代表值



内存索引：CSS树

- **缓存感知的索引：CPU的处理速度与内存的访问速度的差距越来越大，CPU的缓存失效逐渐成为限制数据库索引性能的重要因素。**
- **CSS（Cache-Sensitive Search Trees）树：**
  - 节点大小等于cache line（缓存行）大小
  - 不使用指针，而是计算子节点的偏移量

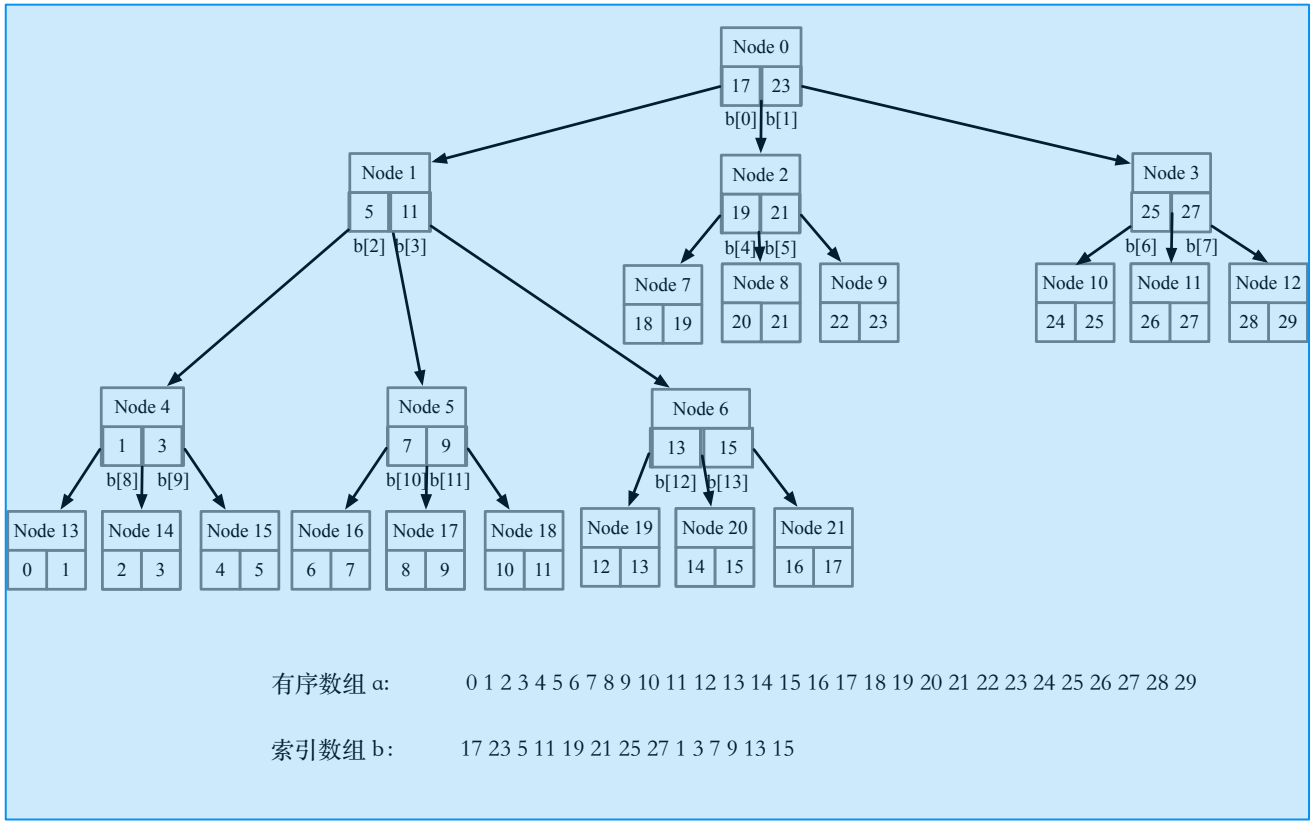
（1）叶子节点个数：由$B = \mathrm{N}/m$，计算出叶子结点的个数B。

（2）内部节点个数：由$\frac{(m+1)^k-1}{m} - \left\lfloor \frac{(m+1)^k-B}{m} \right\rfloor$计算。

（3）最底层第一个叶子节点编号：由$\frac{(m+1)^k-1}{m}$计算，

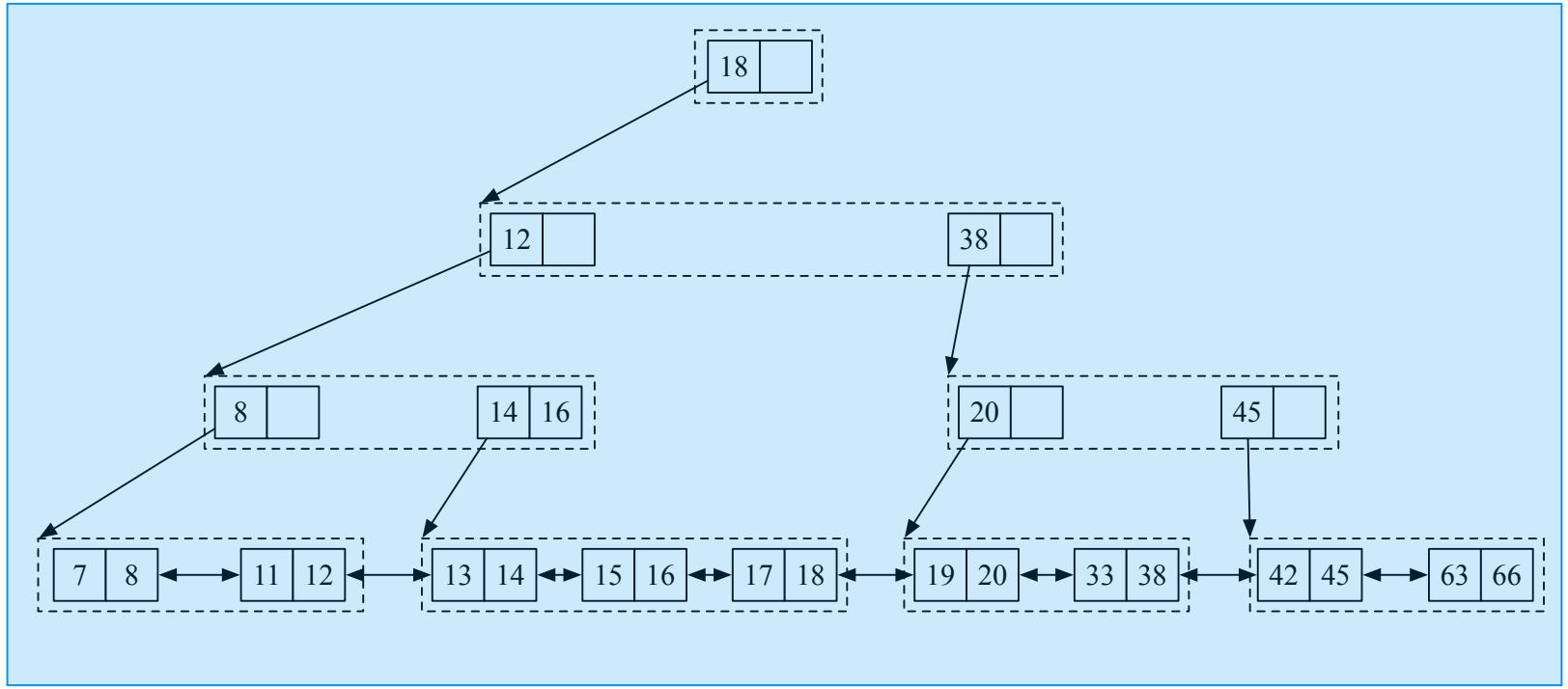
其中，k的大小为$\lceil \log_{m+1}(B) \rceil$。

N：节点总数，m为节点出度



内存索引：CSB+树

- **CSB+树(CacheSensitive B+-Trees)**
  - CSS树和B+树结合体
  - 每个节点的大小是一个Cache Line长度的倍数
  - 一个节点的所有的子节点组织成Children Group，
    一个父节点通过一个指针指向它的Children Group，
    减少数据结构中的指针数量。



内存数据库的并发控制

- **悲观并发控制技术**
  - 锁与锁管理器的开销影响性能，一般内存数据库不使用
- **乐观并发控制技术**
  - 传统方法使得只读事务也需进行验证，带来开销，较少使用
  - 改进方法：Silo使用的基于单版本乐观并发控制的三阶段提交协议
    - Epoch提交（适合于多核并行）；无全局临界区
- **多版本机制**
  - 多版本时间戳协议、多版本两阶段锁协议、多版本乐观并发控制协议等
- **基于分区的并发控制**
  - 分区内串行、分区间并行；
  - 涉及多个分区时，只有当所涉的所有分区都可用时，事务才能执行

内存数据库的数据持久化与恢复

- **数据持久化与恢复**
  - 定期将全量数据备份到磁盘等非易失性存储介质中
  - 断电等故障后将备份数据加载到内存中
- **使用日志与恢复点等作为数据恢复技术**
  - 日志操作会影响内存数据库性能
  - 侧重优化日志内容与减少日志数量
- **数据持久化与恢复**
  - 本机恢复：恢复大量索引和数据，速度较慢
  - 备机恢复：恢复日志，速度较快

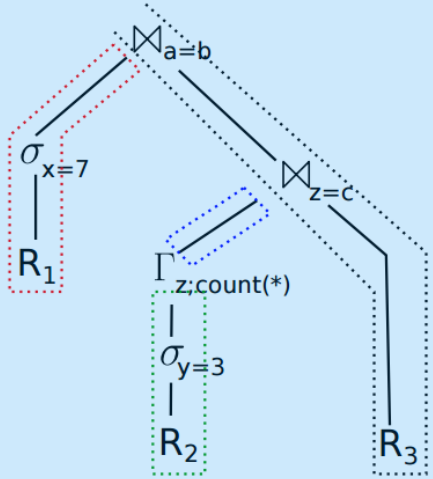
内存数据库的查询执行

➢ 避免使用火山模型

- 涉及大量虚函数调用从而会极大降低查询性能
- 使用推送式模型

➢ 编译执行

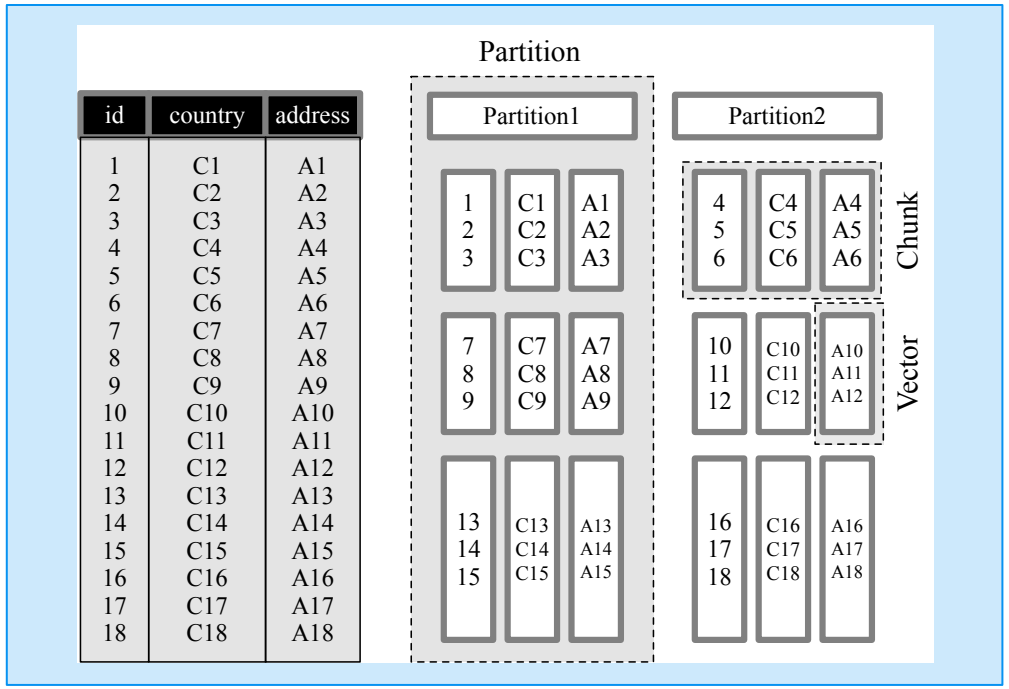
- 在编译过程中优化查询代码，以生成计算量更低的低级语言代码，从而提升查询性能
- 基于LLVM的编译执行

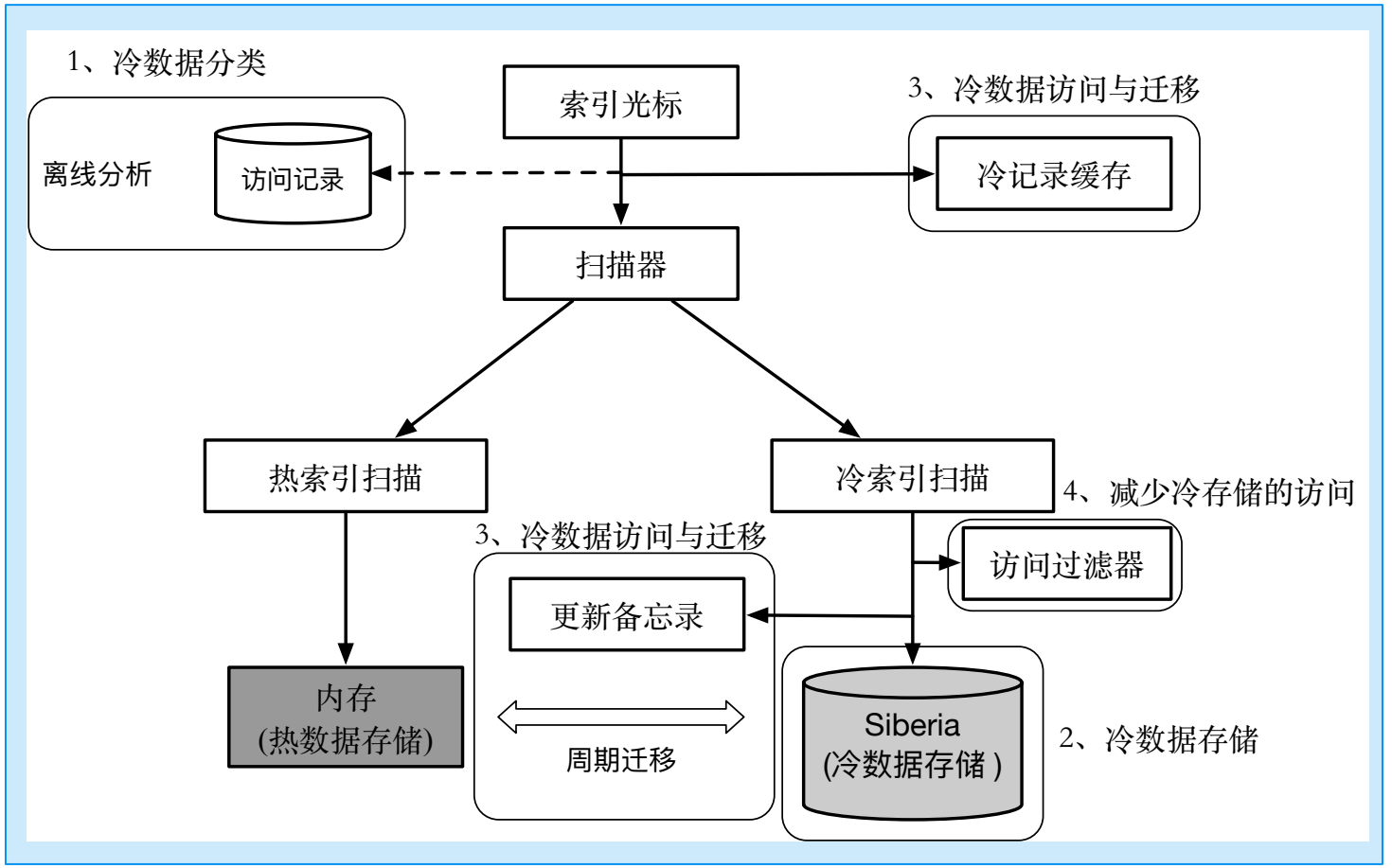


initialize memory of $\bowtie_{a=b}$, $\bowtie_{c=z}$, and Γ_z
for each tuple t in R_1
  if $t.x = 7$
    materialize t in hash table of $\bowtie_{a=b}$
for each tuple t in R_2
  if $t.y = 3$
    aggregate t in hash table of Γ_z
for each tuple t in Γ_z
  materialize t in hash table of $\bowtie_{z=c}$
for each tuple t_3 in R_3
  for each match t_2 in $\bowtie_{z=c}[t_3.c]$
    for each match t_1 in $\bowtie_{a=b}[t_3.b]$
      output $t_1 \circ t_2 \circ t_3$

内存数据库内存溢出控制

- 反缓存anti-caching
- 冷热数据管理
- 数据压缩





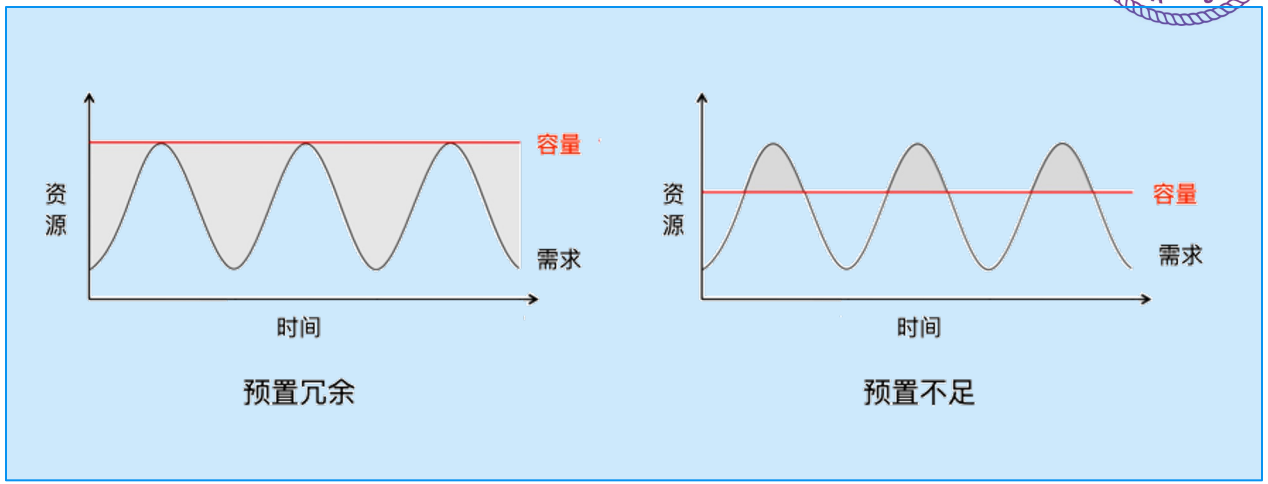
05 云原生数据库

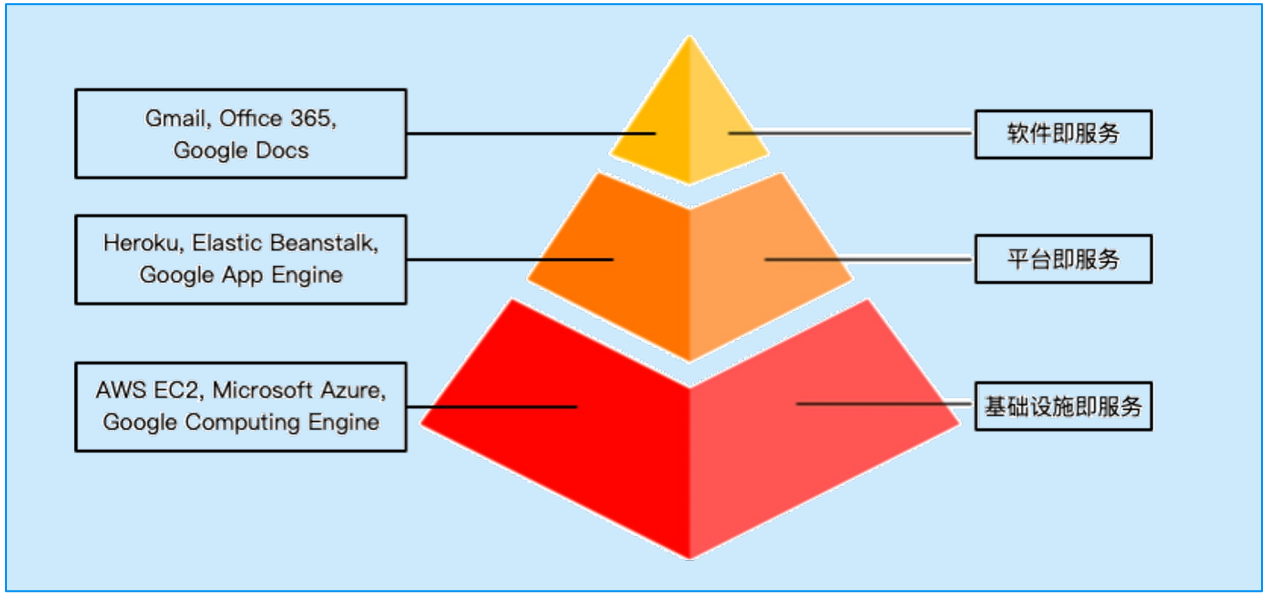
Survey: http://dbgroup.cs.tsinghua.edu.cn/ligl/papers/cloud-native-chinese.pdf

Tutorial: http://dbgroup.cs.tsinghua.edu.cn/ligl/papers/tutorial-cloud-naitve-databases.pdf

云服务趋势

- **云服务的核心特性：弹性**
- **云服务的层级：IaaS, PaaS, SaaS**
- **云服务的特性：**
  - 兼容异构的基础资源（虚拟化）
  - 支持异构的业务体系（容器化）
  - 支持动态的资源扩展（容灾能力）
  - 超大规模的业务支持（无限资源）
  - 弹性的按需计费机制（弹性计费）



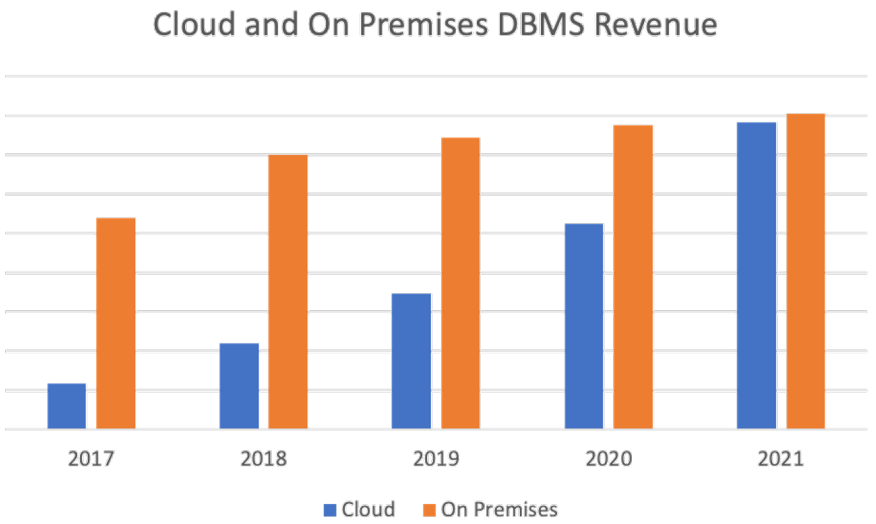


云数据库趋势

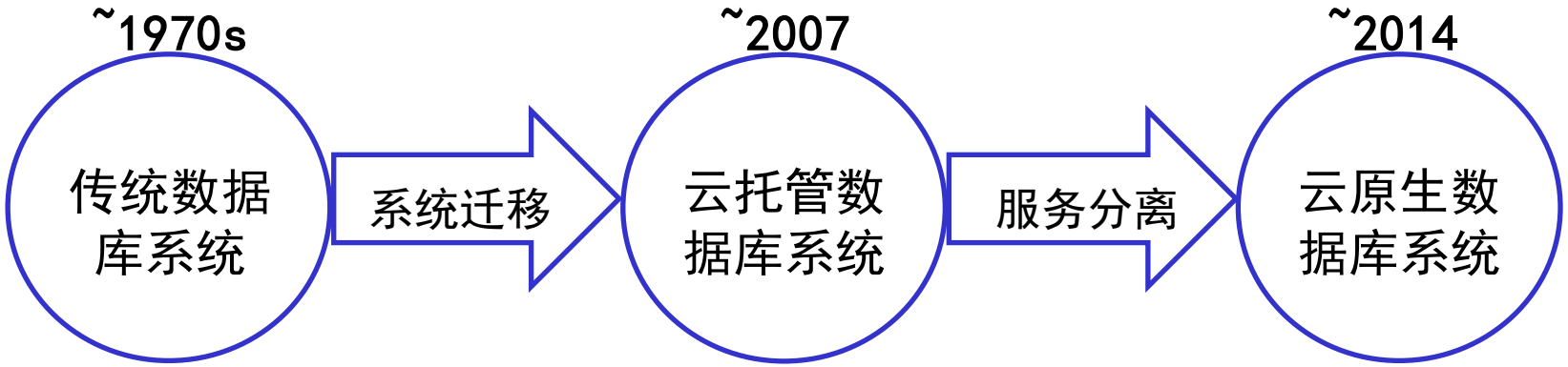
➢ 市场趋势：数据库系统云端化

- 增长速度
  - 68%的数据库市场增长来源于云数据库
  - 云数据库15.7%平均复合年化市场增长速度
- 市场收益
  - 2021年云数据库市场收益392亿美元
  - 2021年全部数据库市场的49%
- 市场份额
  - 预期在2023年云数据库市场占比将达75%





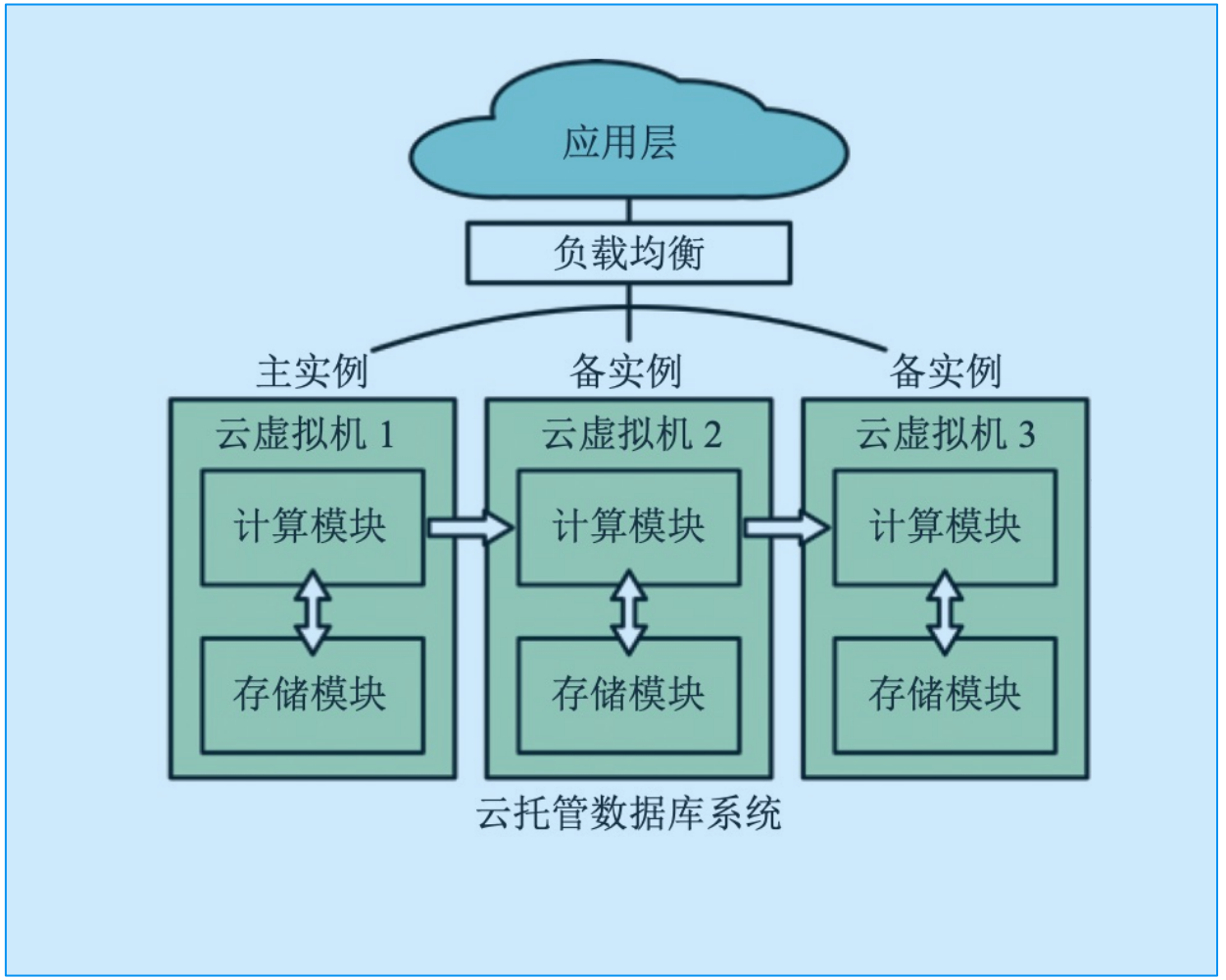
云数据库简要发展历史



类型	传统数据库系统	云托管数据库系统	云原生数据库系统
产品	本地数据库实例	远程数据库实例	数据库即服务
场景	特定用户群组	多租户(虚拟机层级)	多租户(服务层级)
弹性	预置规模，不可调整	实例整体扩缩容	各项服务独立扩缩容
可用性	实例本地恢复	切换新实例	切换新的节点
持久性	本地多副本	跨容错域多数据库实例	跨容错域多存储副本
灵活性	自行部署 & 自行运维	自动部署 & 自行运维	自动部署、运维、调优

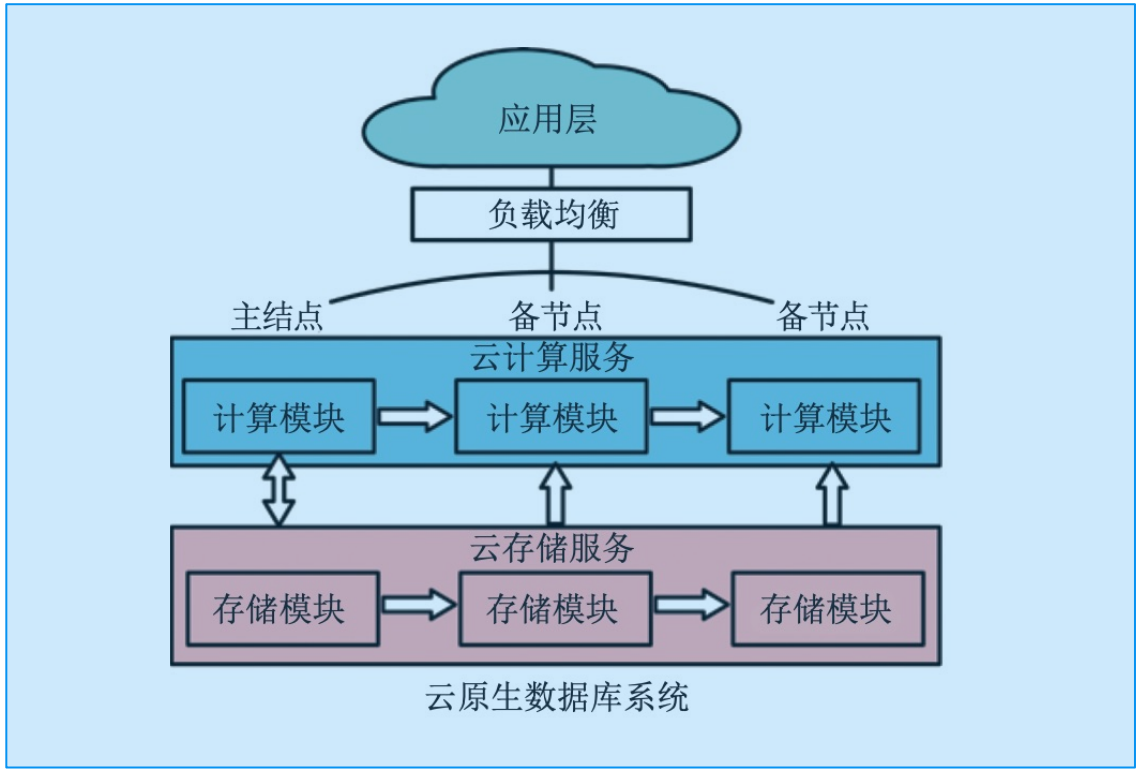
云托管数据库

- **云托管数据库系统特点：**
  - Shared Everything架构
  - 多数据库实例实现多副本
  - 实例内部计算存储耦合
- **优势：**
  - 易于实现，降低额外开发成本
  - 易于迁移，未改变原始系统架构
- **缺陷：**
  - 写放大问题造成网络瓶颈
  - 计算存储耦合调度，资源浪费
  - 不支持节点级别的故障恢复

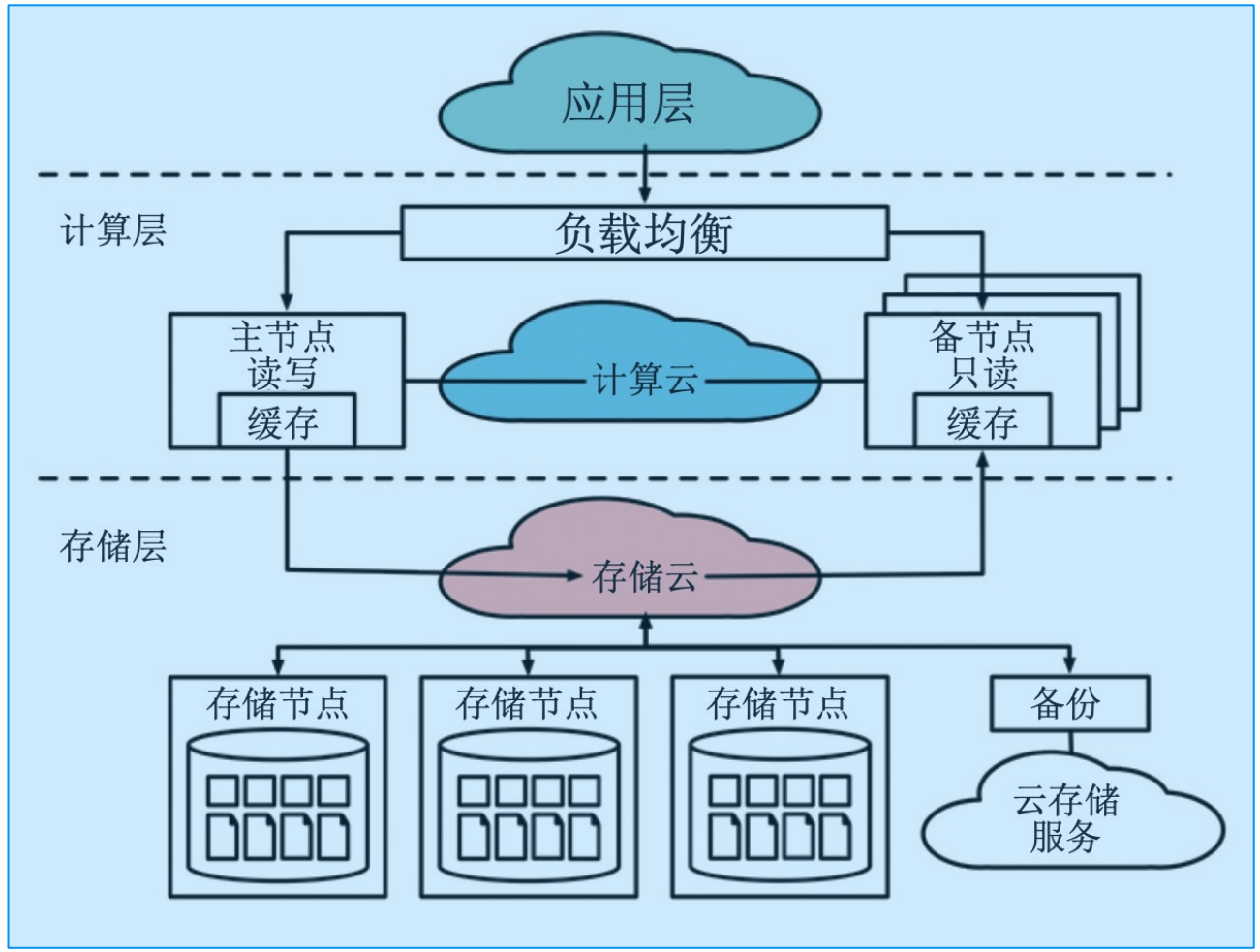


云原生数据库

- **架构设计差异：计算存储分离**
  - OLTP：适用于事务处理场景，要求事务满足ACID准则
  - OLAP：适用于数据分析场景，强调分析并发处理能力
- **OLTP云原生数据库架构**
  - 计算-存储分离架构
  - 计算-日志-存储分离架构
  - 计算-缓存-存储分离架构

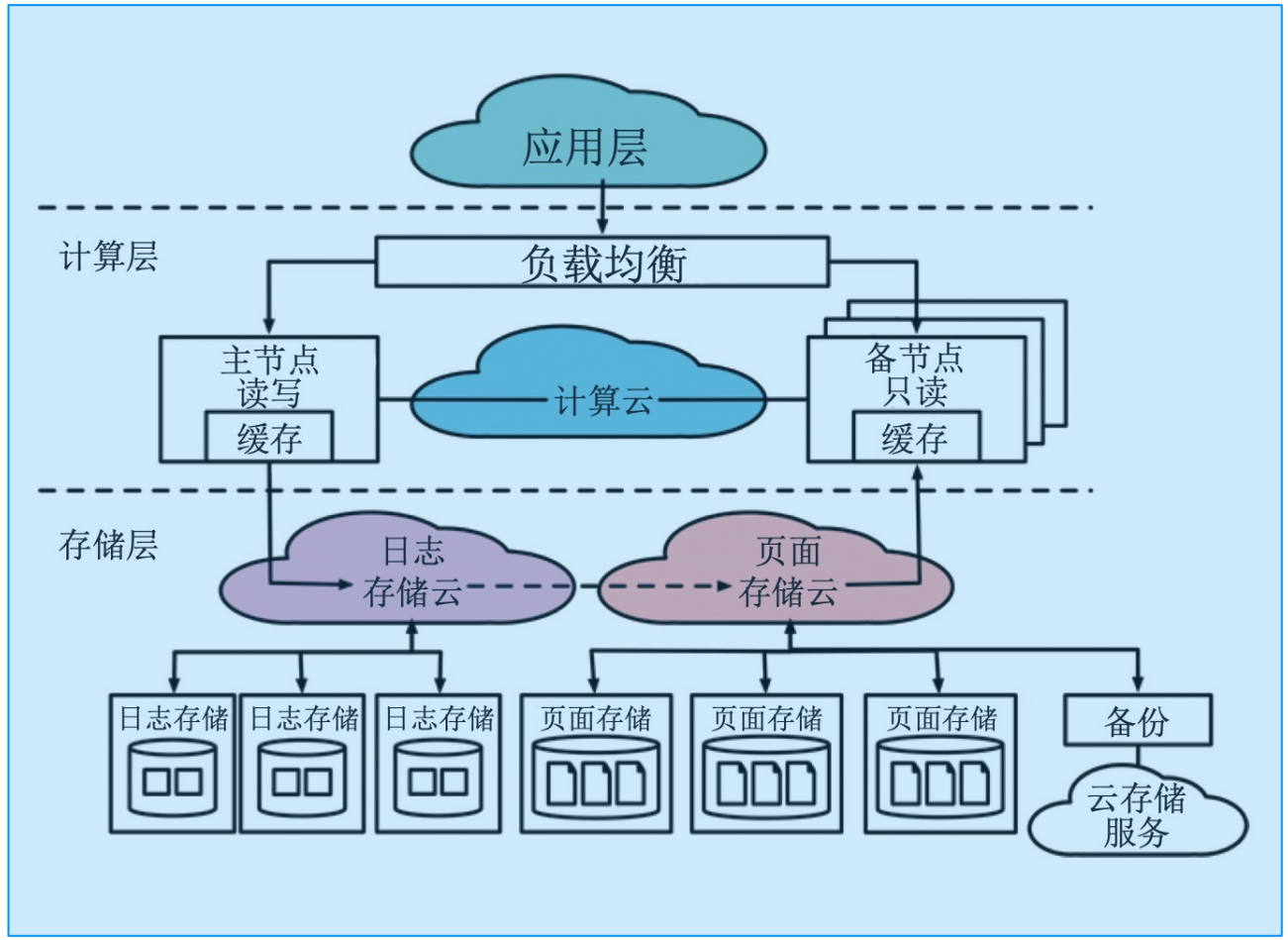


计算-存储分离



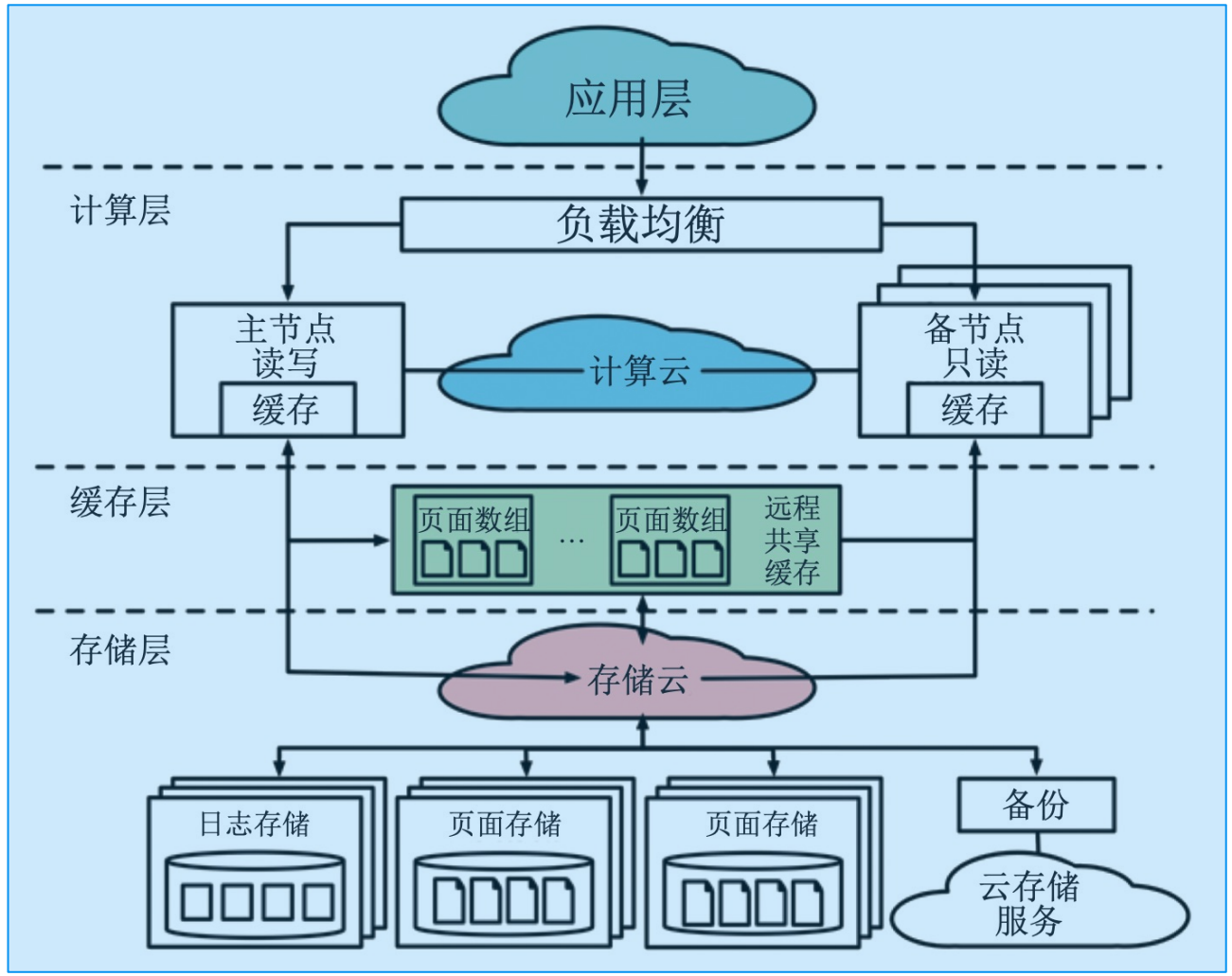
- **架构优势：**
  - **弹性** 计算资源和存储资源可以独立调度。
  - **效率** Shared Storage架构，缓解写放大问题。
  - **可用性** 节点级别的故障恢复，降低故障恢复时间。
- **关键特性：**
  - 计算-存储分离
  - 日志即数据库

计算-日志-存储分离



- **架构优势:**
  - **效率** 利用高速日志存储服务提高写入操作执行速度。
  - **弹性** 多种类型存储服务独立调度
- **关键特性:**
  - 日志存储于页面存储分离
  - 充分利用不同存储服务优势：高速的日志存储与廉价的页面存储相结合

OLTP 计算-缓存-存储分离



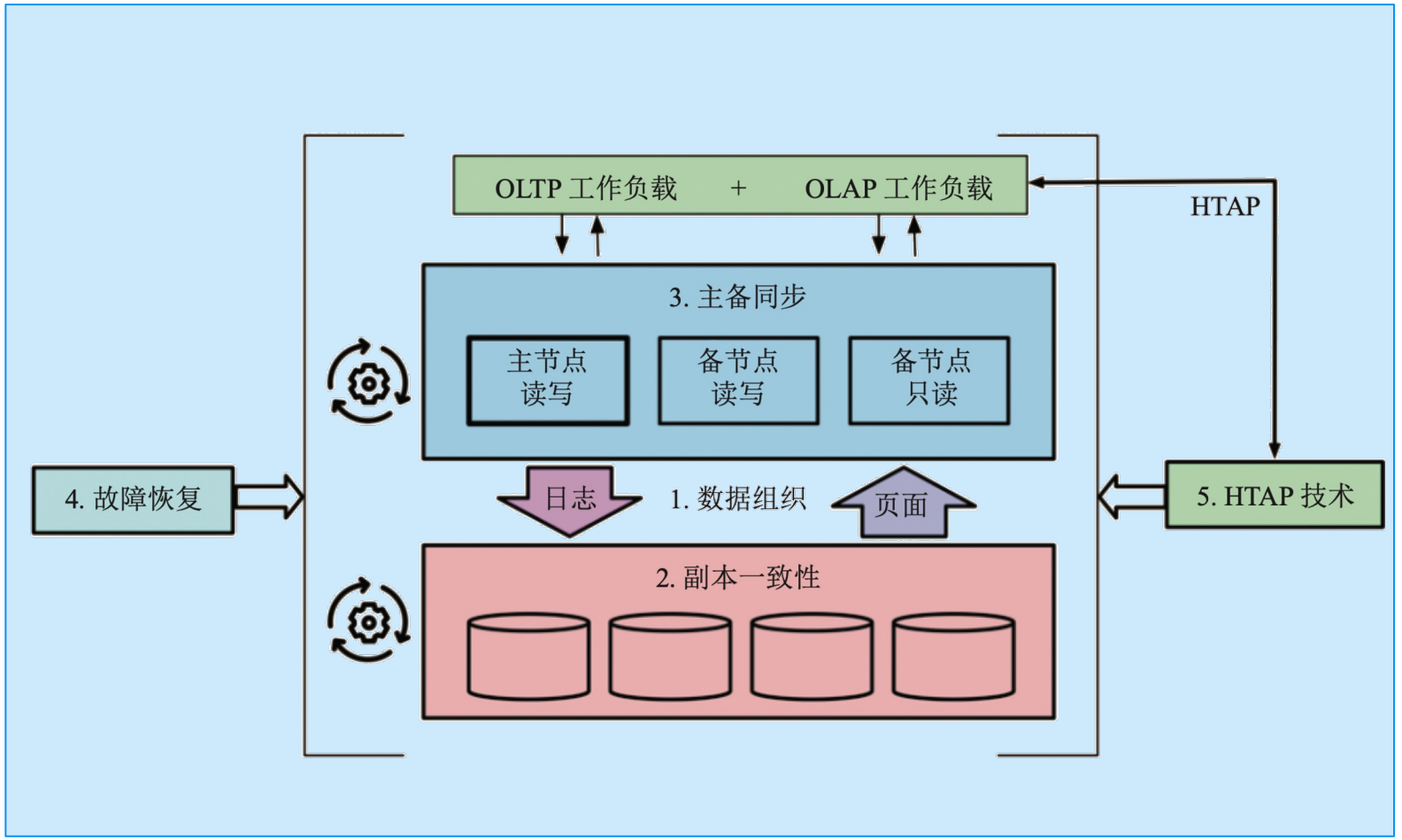
- **架构优势：**
  - **低时延** 利用共享内存，提高数据读取速度。
  - **效率** 缓解不同存储节点向存储层读取重复数据的问题。
  - **弹性** 内存资源可以基于使用需求动态调度。
- **关键特性：**
  - 所有计算节点共享的远程共享内存区域，共享内存容量支持弹性调度

OLTP云数据库三类架构对比

OTLP架构	读吞吐	写吞吐	可用性	弹性	成本
计算-存储分离	高	中等	高	高	中等
计算-日志-存储分离	高	高	极高	极高	中等
计算-缓存-存储分离	极高	高	极高	极高	高

云原生OLTP数据库关键技术

- 数据组织技术
- 存储层副本一致性技术
- 计算层主备同步技术
- 故障恢复技术
- HTAP技术

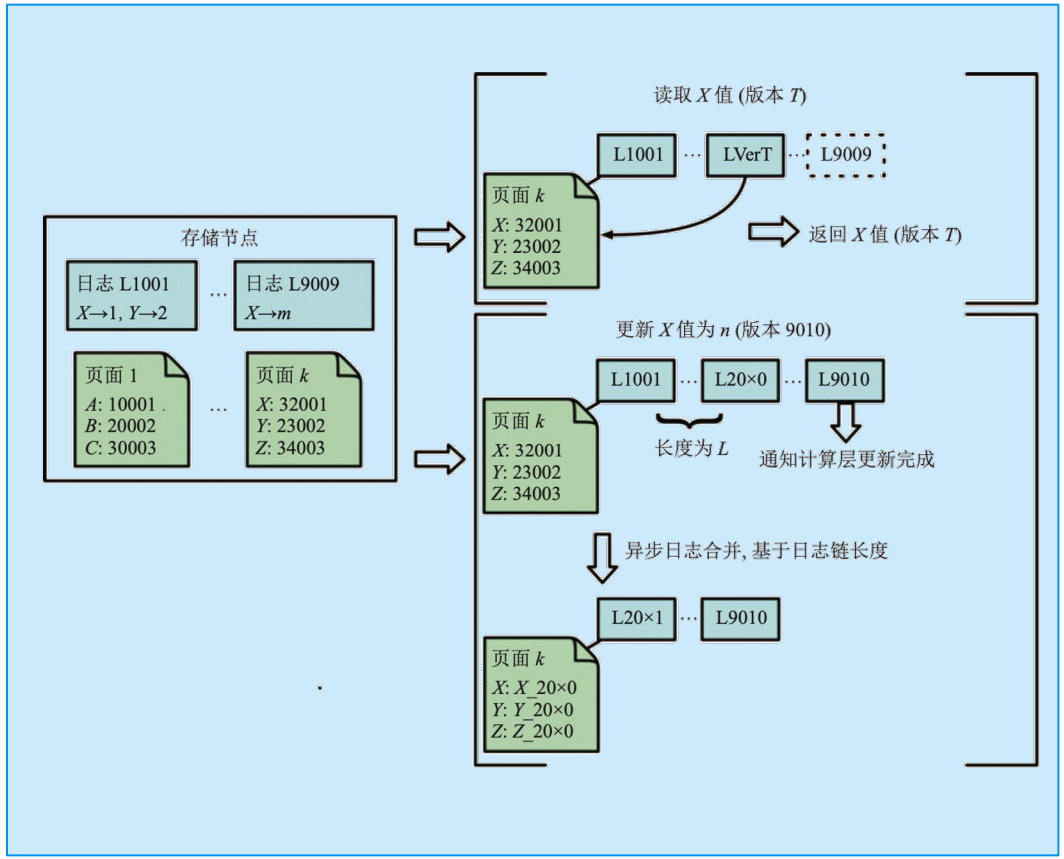


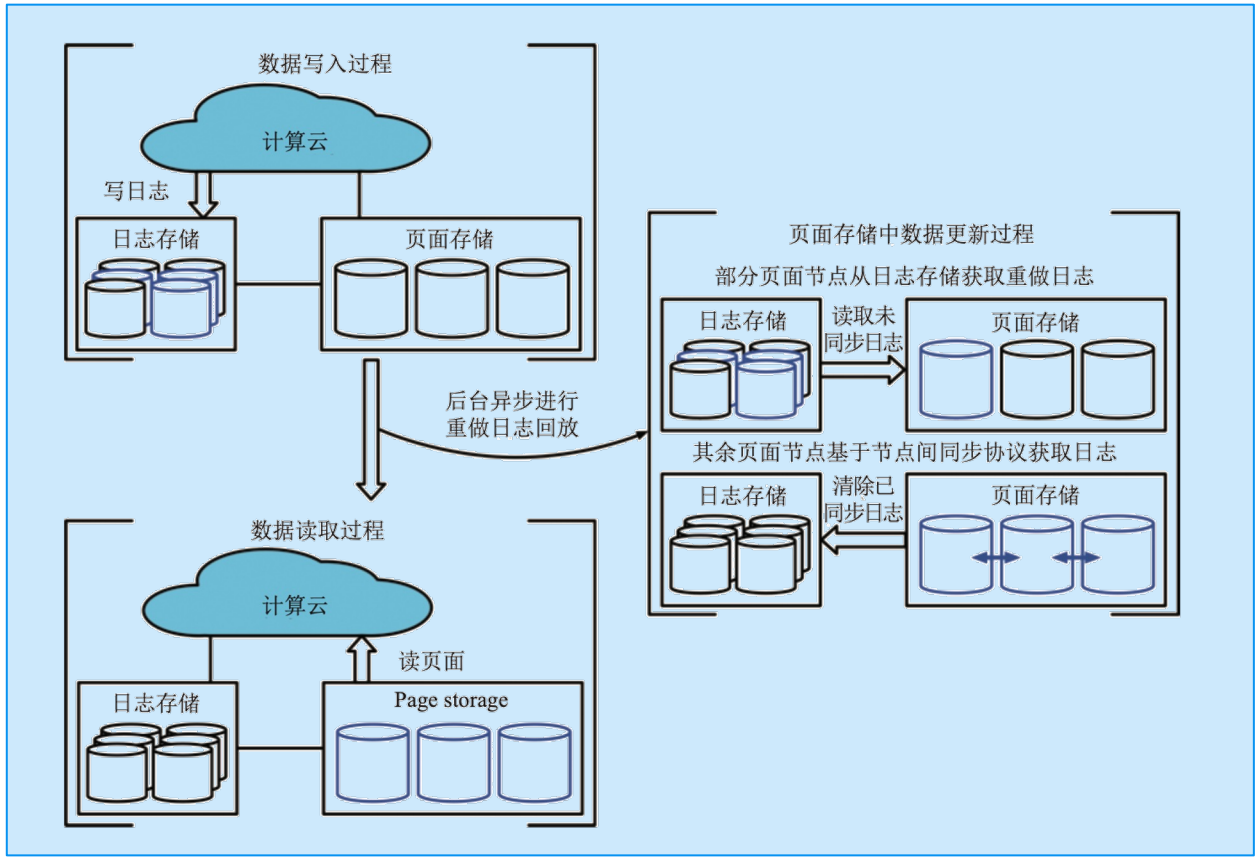
云原生OLTP数据库关键技术

技术类别	关键技术	代表性工作	主要优点	主要缺点
数据组织技术	日志-页面耦合存储策略	Aurora[8]	写入效率高	额外的日志处理
	日志-页面分离存储策略	HyperScale[14] Taurus[23]	弹性调度能力强	阻塞的更新同步
存储层 副本一致性技术	基于Quorum算法的协议	Aurora[8,24]	并发能力强	额外的同步过程
	基于Paxos类算法的协议	PolarFS[25]	可线性化	实现复杂
计算层 主备同步技术	基于持久化存储同步	Aurora[8]	可靠性高	同步延时高
	基于本地缓存同步	HyperScale[14]	同步延时低	不保证可靠性
	基于远程共享缓存同步	PolarDB Serverless[15]	读取延时低	缓存一致性问题
故障恢复技术	无重做的计算节点恢复	Aurora[8]	计算节点省略 Redo过程	存储节点 承担Redo压力
	两层ARIES容错协议	LegoBase[26]	计算节点通过缓存层加速恢复	仅为优化手段 不保证可靠性
HTAP技术	计算层动态行存转列存	AlloyDB[27]	节约存储空间	列存选择难度大
	存储层行列混存数据副本	TiDB[28]	隔离性好	数据新鲜度低
	内存型行存加持久化列存	SinglestoreDB[29]	读写延时低	内存开销大

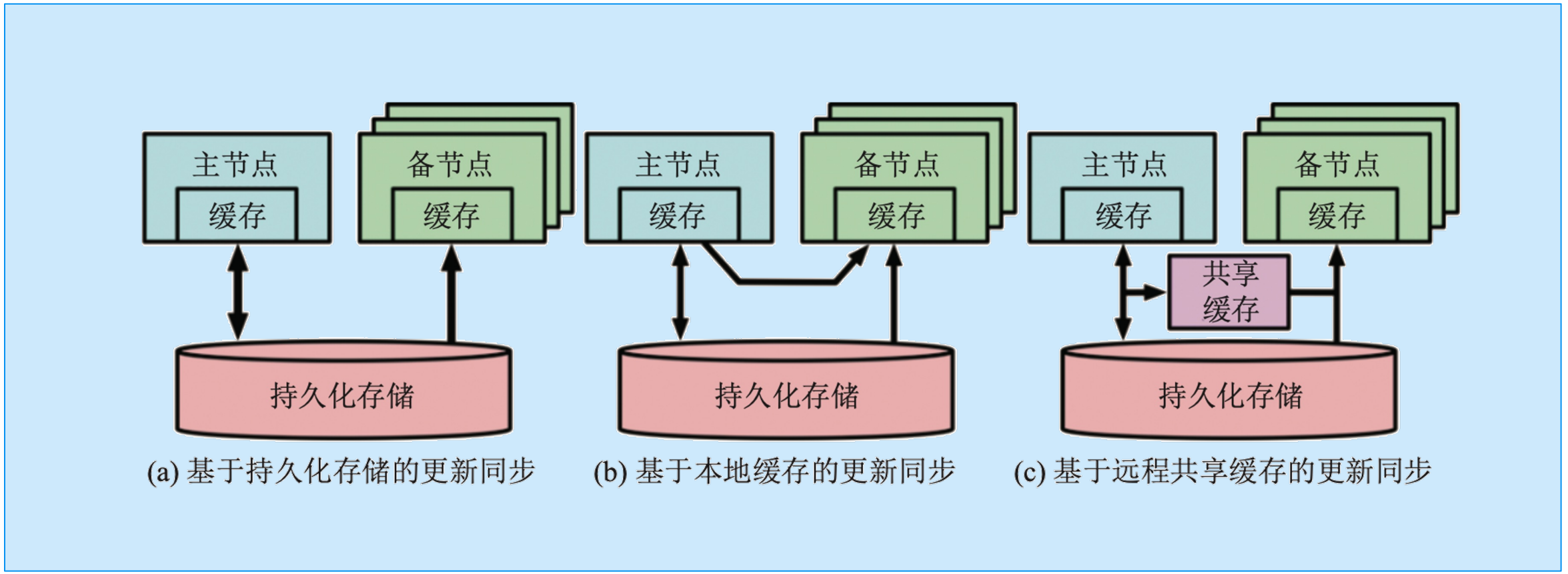
云原生OLTP数据库：数据组织

- 日志-页面耦合的存储策略
- 日志-页面分离的存储策略



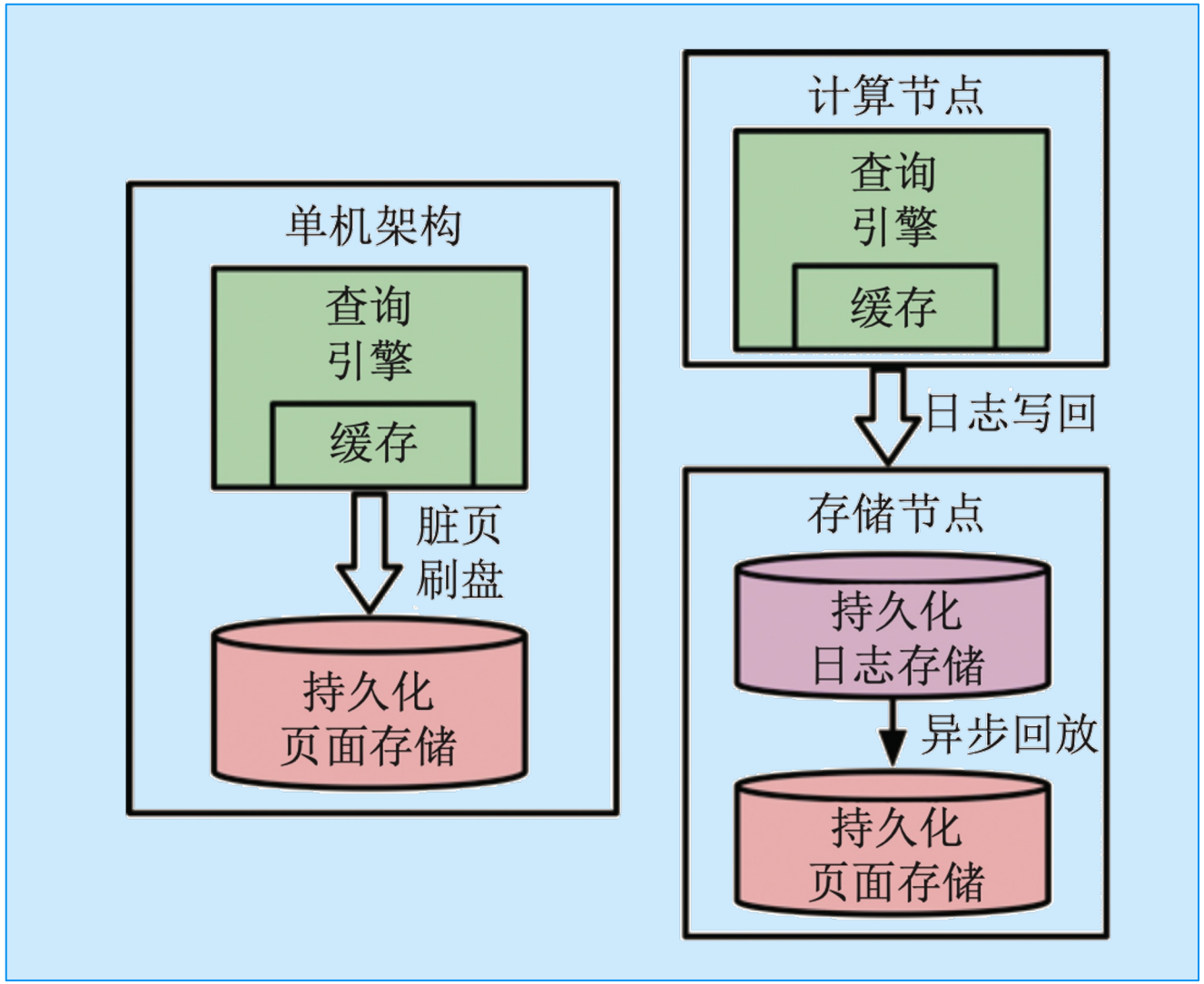


云原生OLTP数据库：主备同步技术

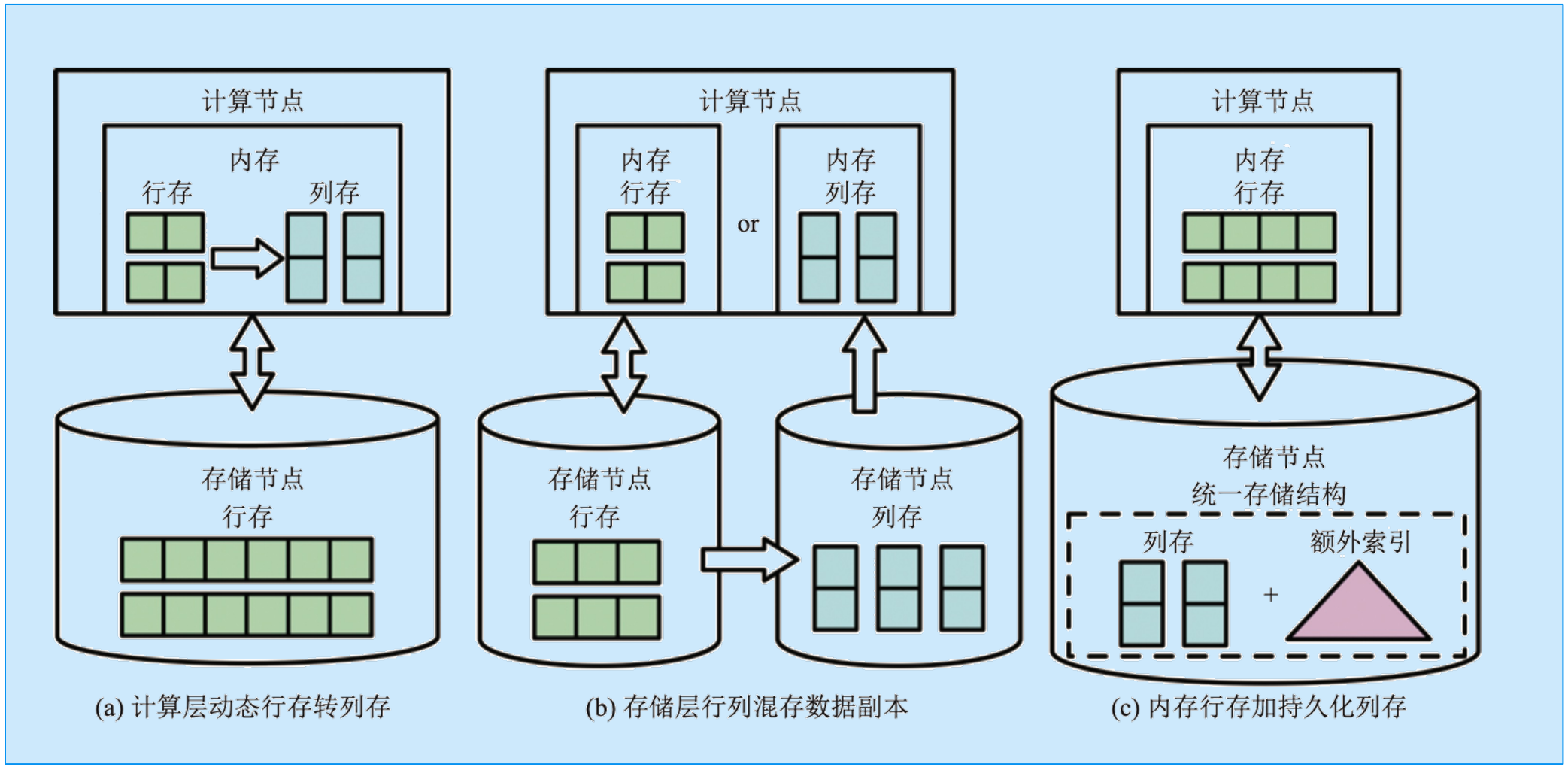


云原生OLTP 数据库：故障恢复

- **计算层无重做过程的计算节点恢复**
  - 计算层只淘汰脏页，不做回放
  - 存储层重做恢复
- **计算-缓存双层容错协议**
  - 通过缓存层恢复
  - 缓存层不能恢复通过存储层恢复



云原生OLTP数据库：HTAP

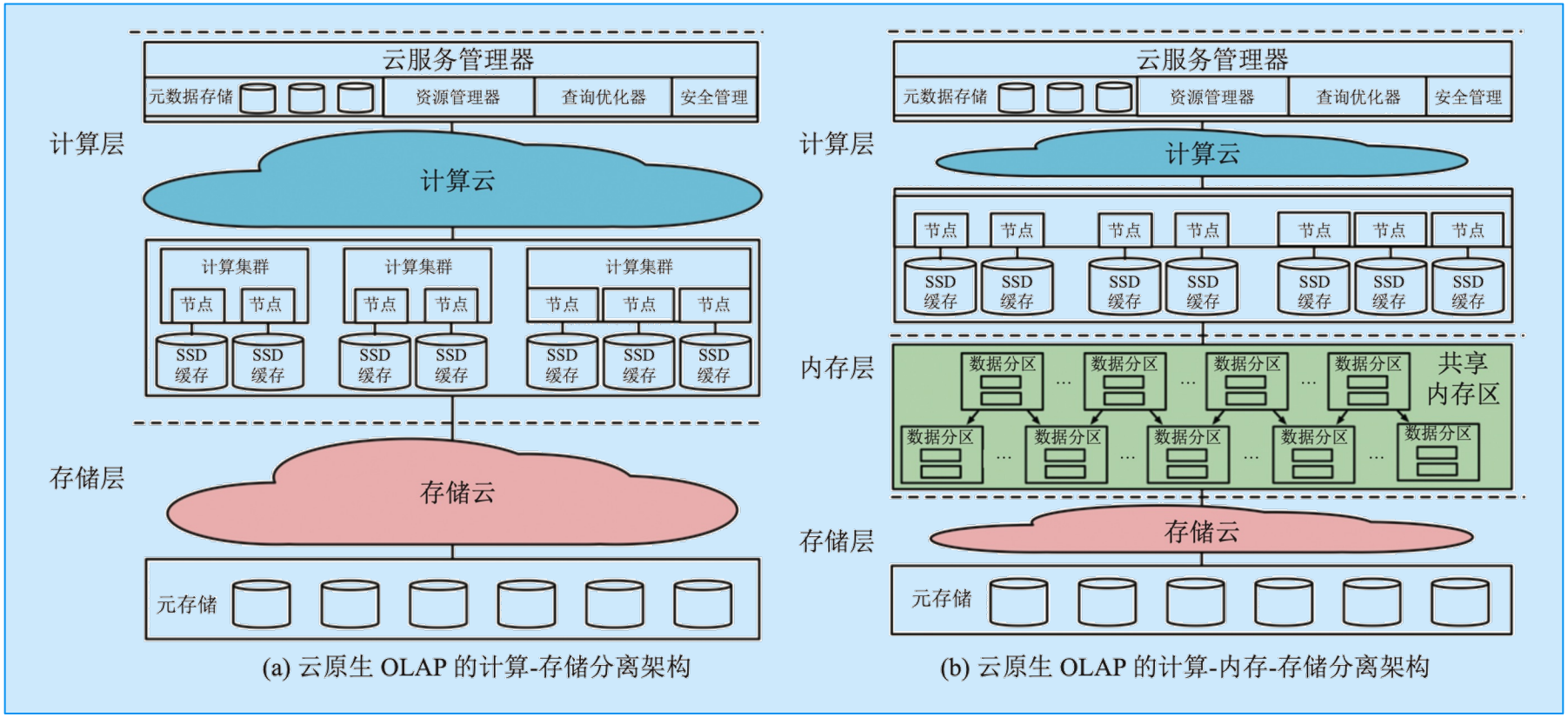


(a) 计算层动态行存转列存　(b) 存储层行列混存数据副本　(c) 内存行存加持久化列存

云原生OLAP数据库架构

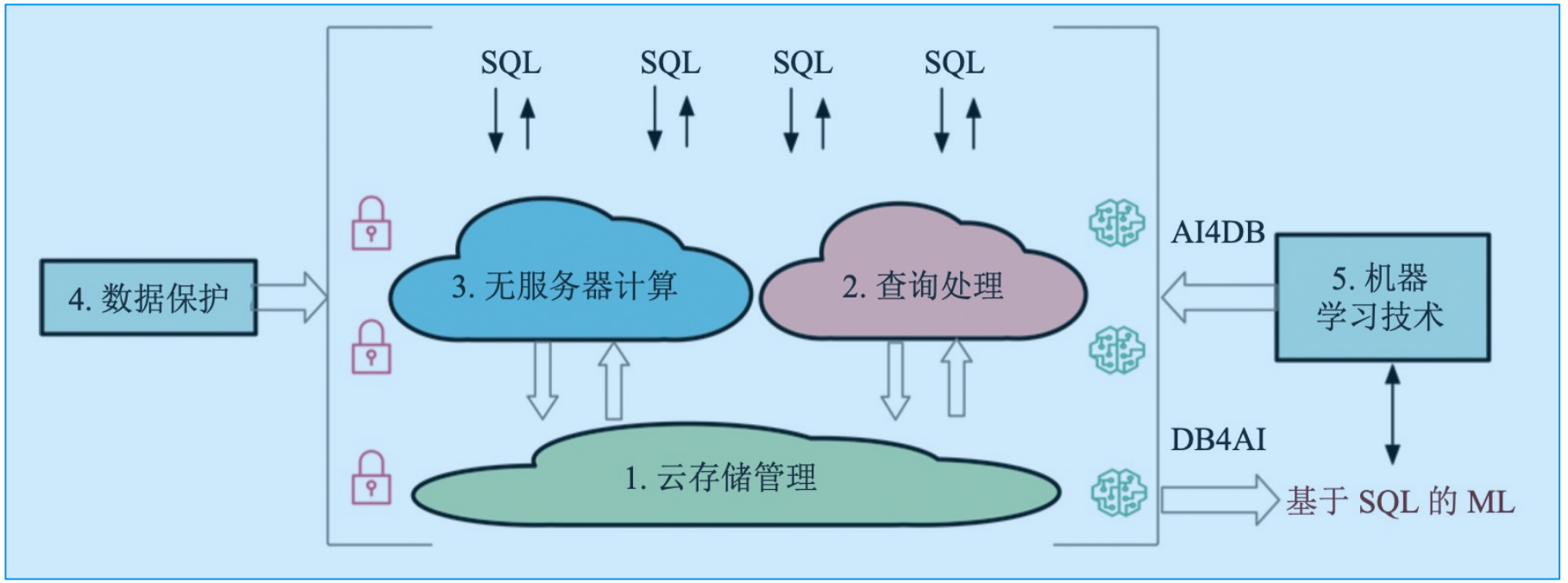
➢ OLAP云原生数据库架构

- 采用计算与存储分离的架构来提高系统弹性与高可用性
- 存储云节点按照数据分片进行管理，保证了数据分析过程的高吞吐率
- 上层计算云节点不区分主和备节点，计算集群中的工作节点独立地处理各个数据分片的分析任务



云原生OLAP数据库关键技术

- **存储管理技术**
  - 元数据键值
  - 分片划分
- **查询处理技术**
  - 计算下推
  - 分布式共享内存
- **无服务器感知计算**
- **数据保护技术**
- **机器学习技术**

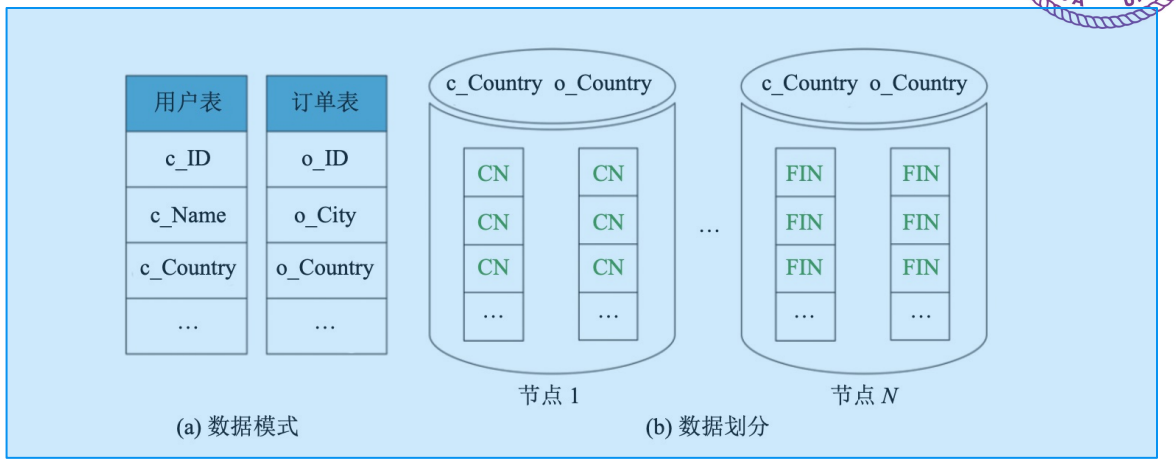


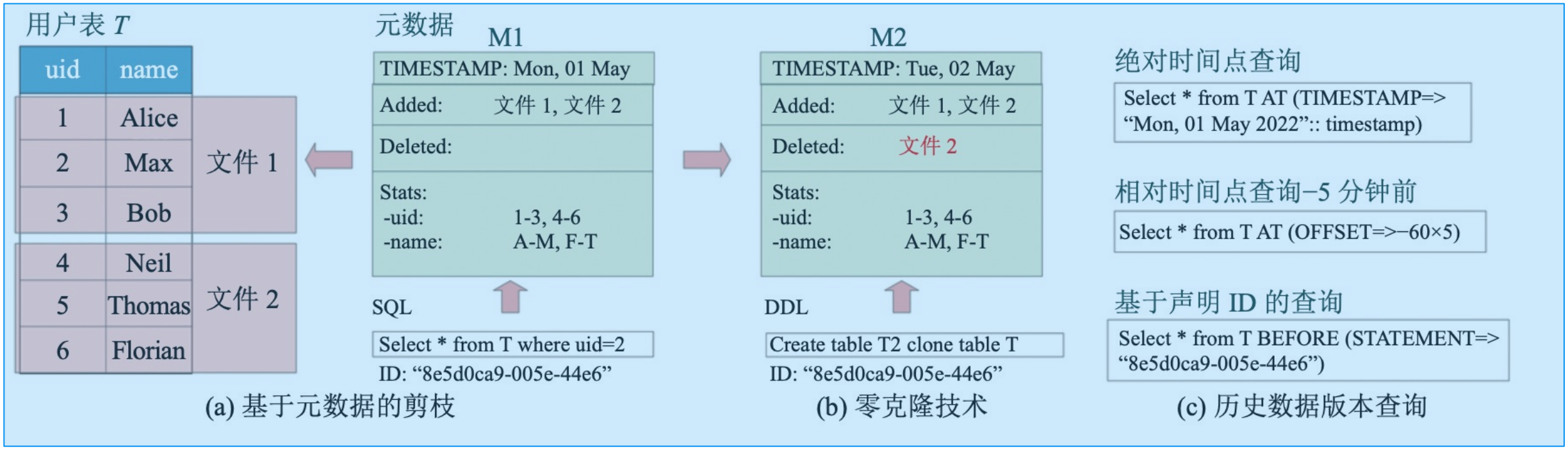
云原生OLAP数据库关键技术

技术类别	关键技术	代表性工作	主要优点	主要缺点
存储管理技术	基于元数据的剪枝、拷贝与MVCC	Snowflake[9]	高吞吐	数据更新代价高
	基于连接图的关系数据划分	Redshift[40]	高效率	未考虑查询代价
	基于列存的半结构化数据表示	BigQuery[10]	存储代价低	查询延迟高
查询处理技术	列式扫描结合下推计算	PushdownDB[43]	低代价	没有结合缓存
	列式扫描结合Shuffle内存层	BigQuery[10]	高吞吐	高代价
	列式扫描结合下推计算和缓存	FlexpushdownDB[44]	高吞吐	扩展性低
无服务器感知计算	基于数据库实例的Serverless	Athena[45]	高吞吐	高代价
	基于函数服务的Serverless	Starling[46]	高度弹性	函数之间无法通信与交换状态
数据保护技术	基于密钥管理算法的数据保护	Snowflake[9]	高度扩展性	明文数据查询
	基于Enclave的数据保护	Azure SQL Database[47]	高度安全性	查询效率低
机器学习技术	基于机器学习的云原生数据库技术	Redshift[40]	高准确度与高性能	模型的适应性低
	面向云原生数据库的自动机器学习	Sagemaker[48]	高度扩展性与弹性	模型训练代价高

云原生OLAP数据库存储优化

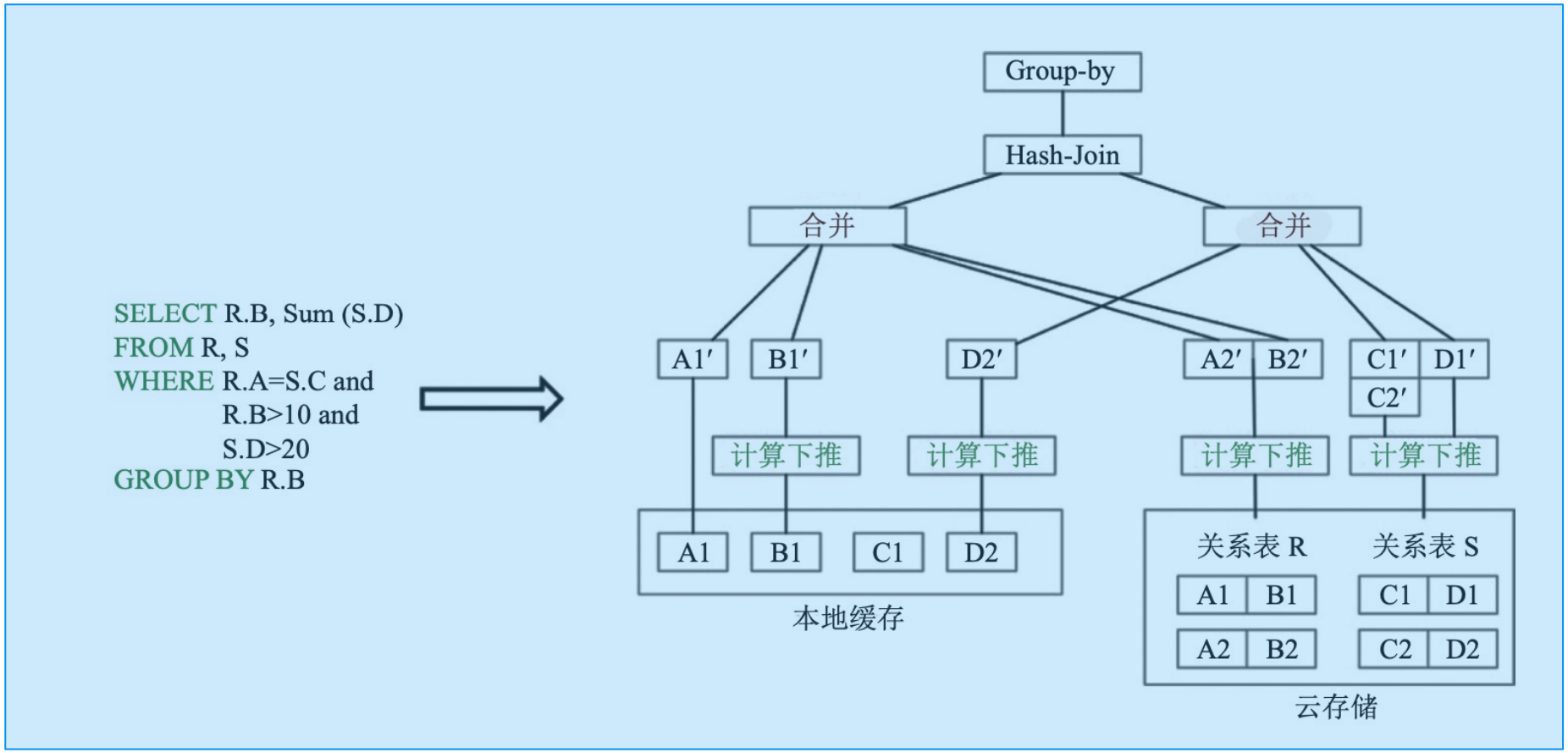
- 基于元数据的剪枝、拷贝与多版本并发控制
- 基于连接图的关系数据划分





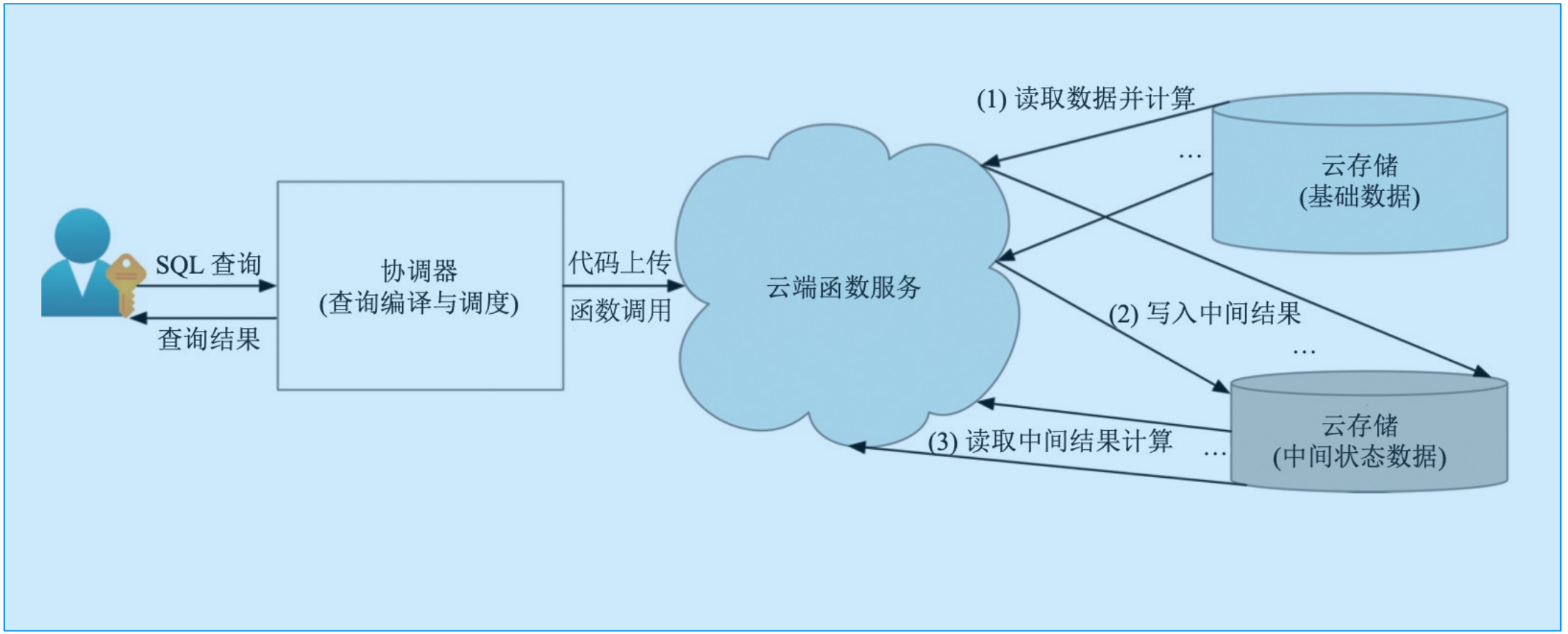
云原生OLAP数据库查询处理

- 基于计算下推的查询处理
- 结合本地缓存与计算下推的查询处理



云原生OLAP数据库无服务器感知计算

- 基于数据库实例进行弹性调度资源
- 基于 Lambda 函数的弹性计算



目录

06

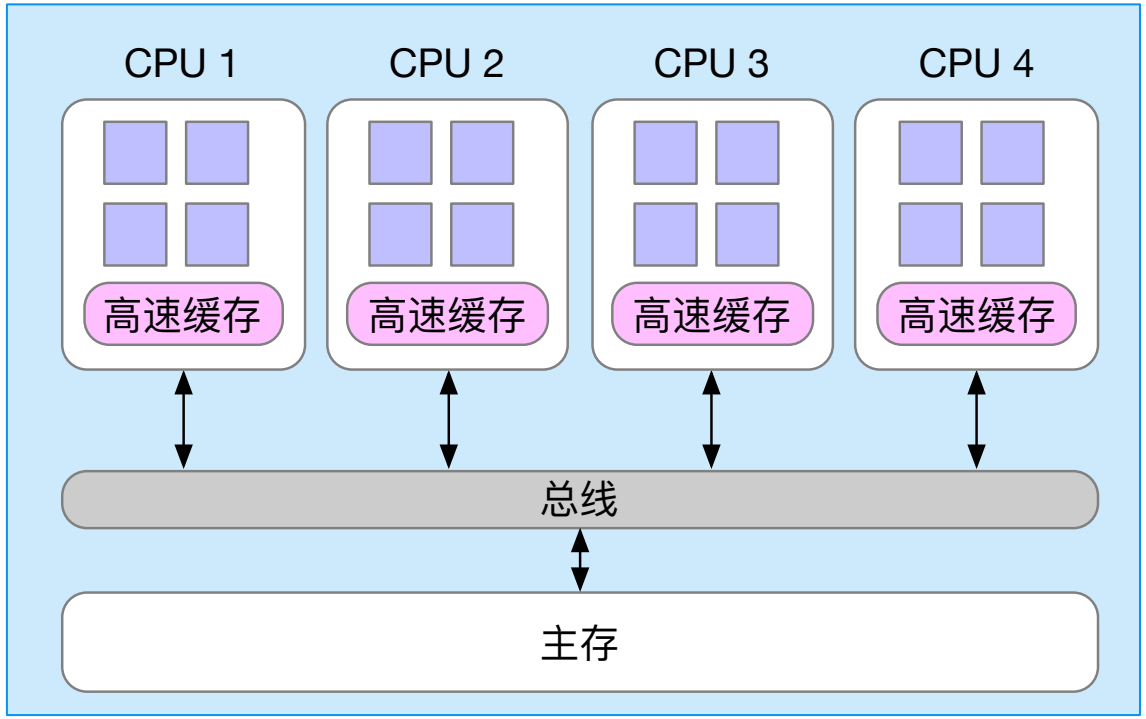
数据库与新硬件

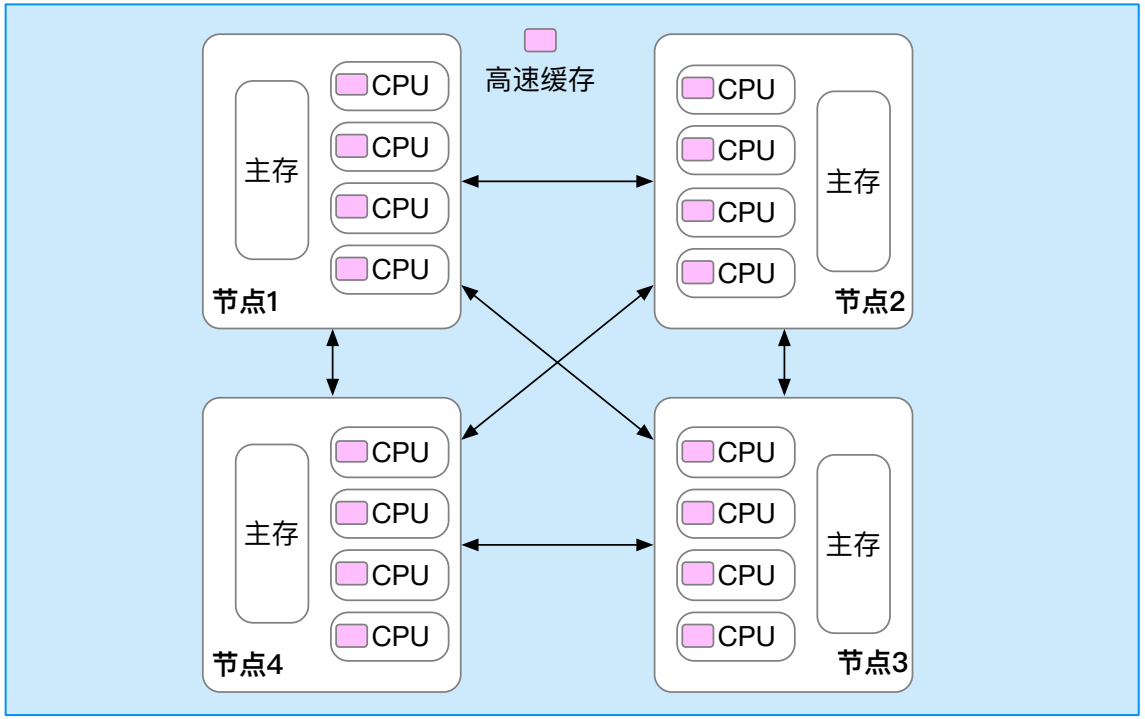
新硬件

- **中央处理器**
  - 多核处理器：
    - SMP架构：扁平化组织
    - NUMA架构：分组组织
- **协处理器**
    - GPU：graphics processing unit（图形处理器）
    - FPGA：field programmable gate array（现场可编程逻辑门阵列）
- **存储介质**
    - NVM：non-volatile memory（非易失性内存）
- **网络传输硬件**
    - RDMA：remote direct memory access（远程内存直接访问）
    - CXL： Compute Express Link（新型的高速接口技术）

多核处理器

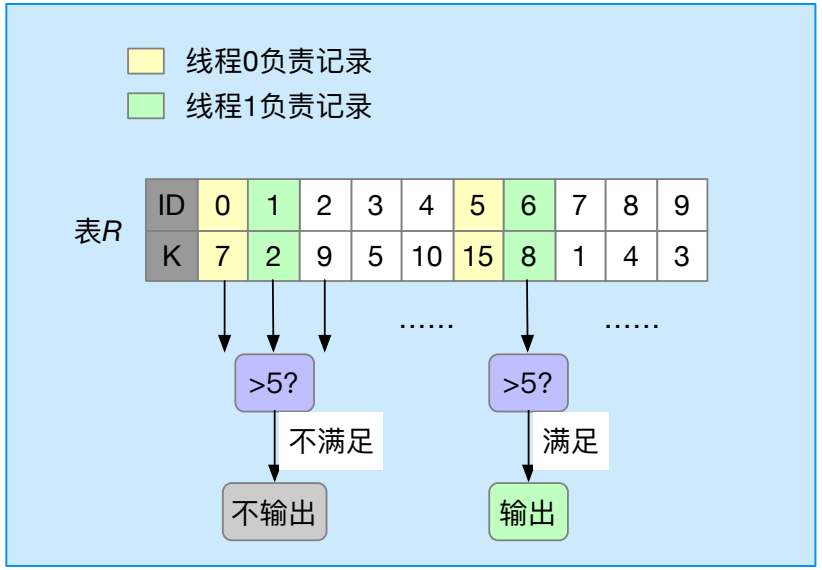
- SMP：symmetric multi-processing
- NUMA：non-uniform memory access

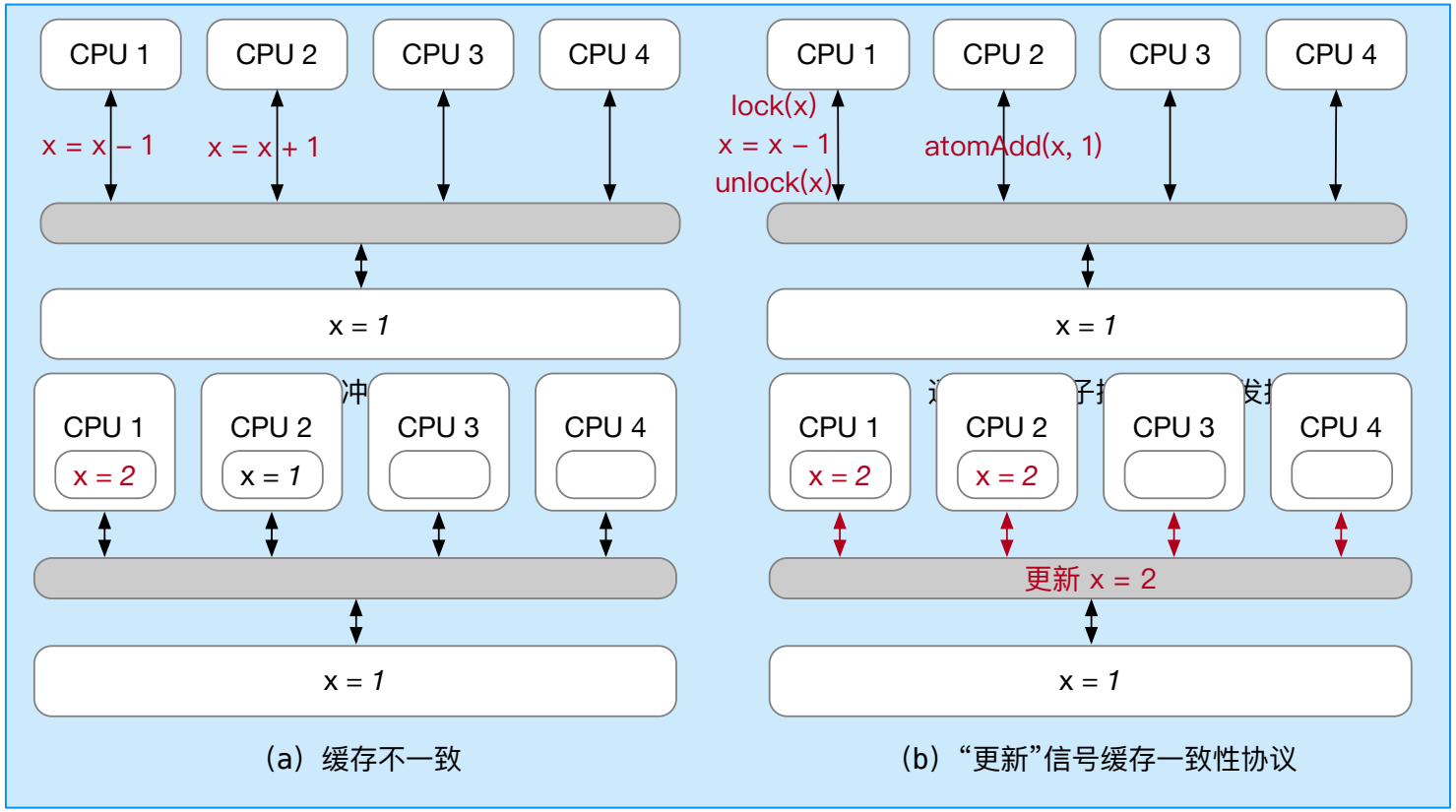




多核处理器与并行计算

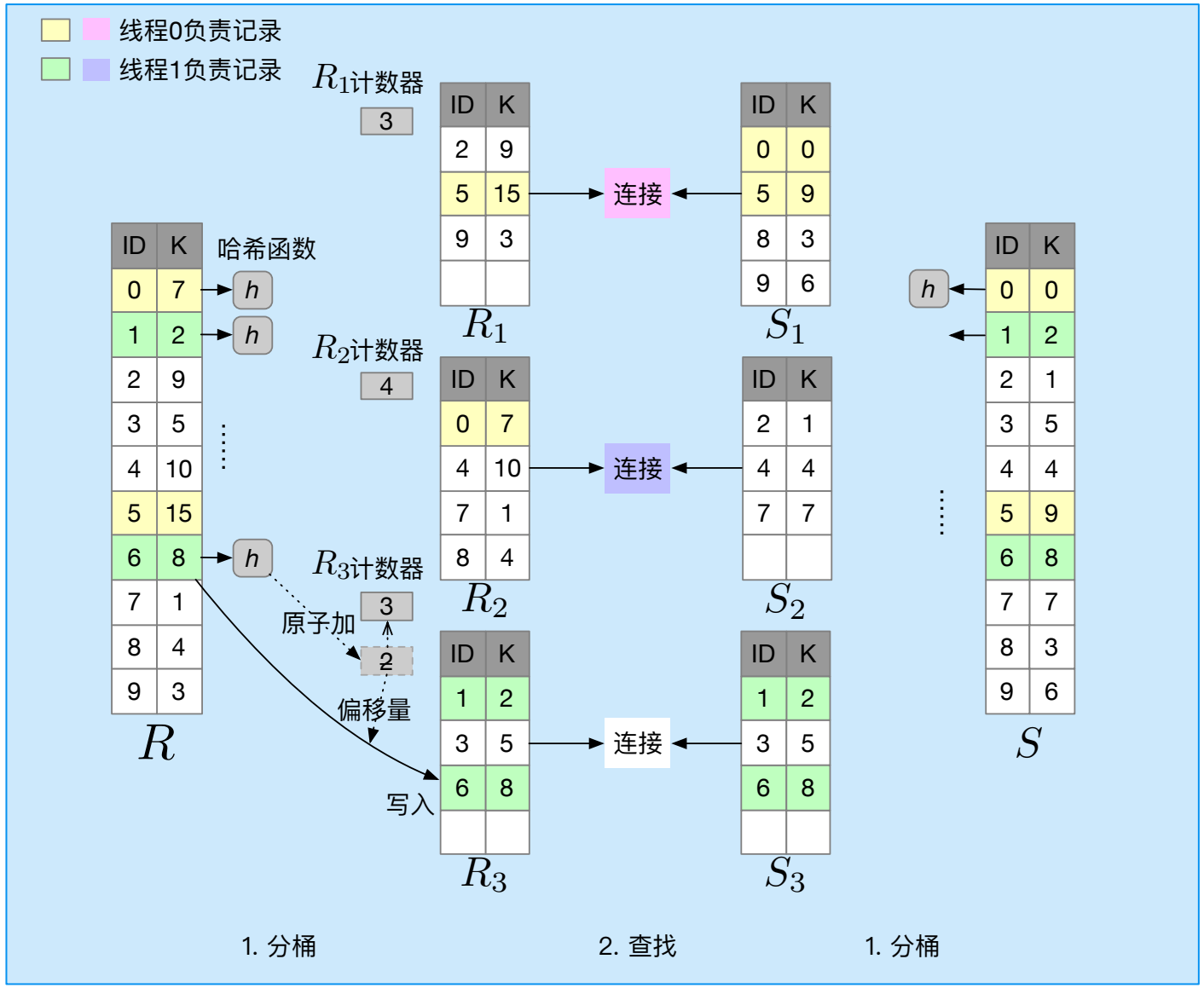
- **多核处理器与并行计算**
  - 将计算任务分解为多个并行的子任务
  - 并发冲突与资源抢占
  - 原子操作
- **数据库操作与并行计算**
  - 选择操作：并行扫描
  - 连接操作：并行哈希连接算法





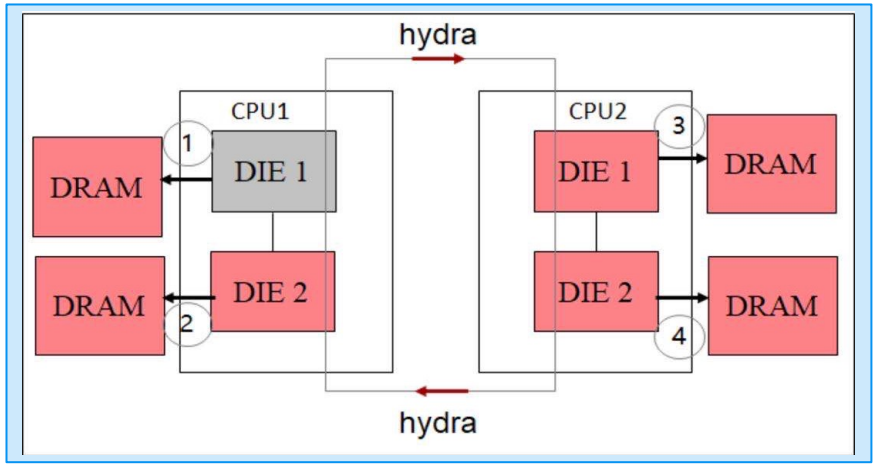
SMP与并行连接

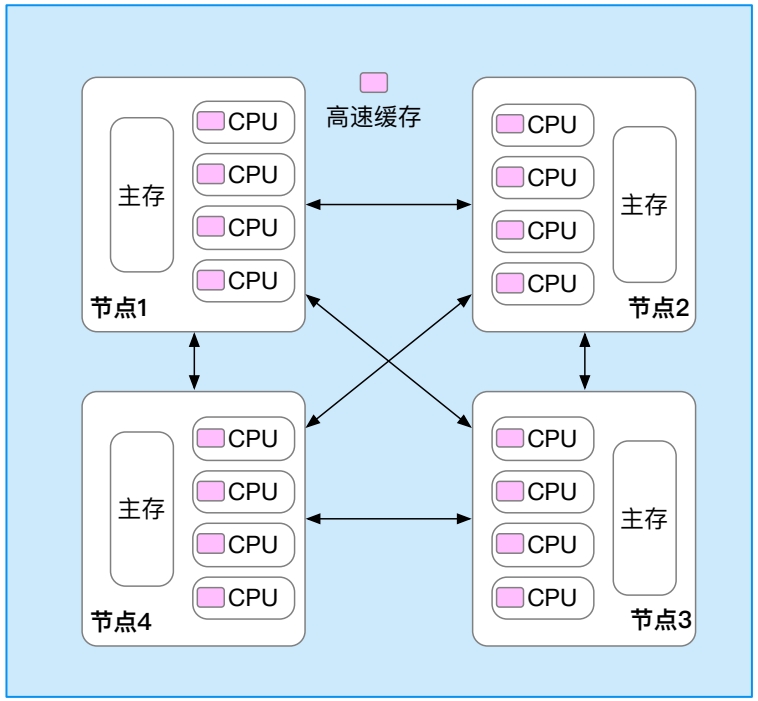
- 分桶阶段
- 查找阶段

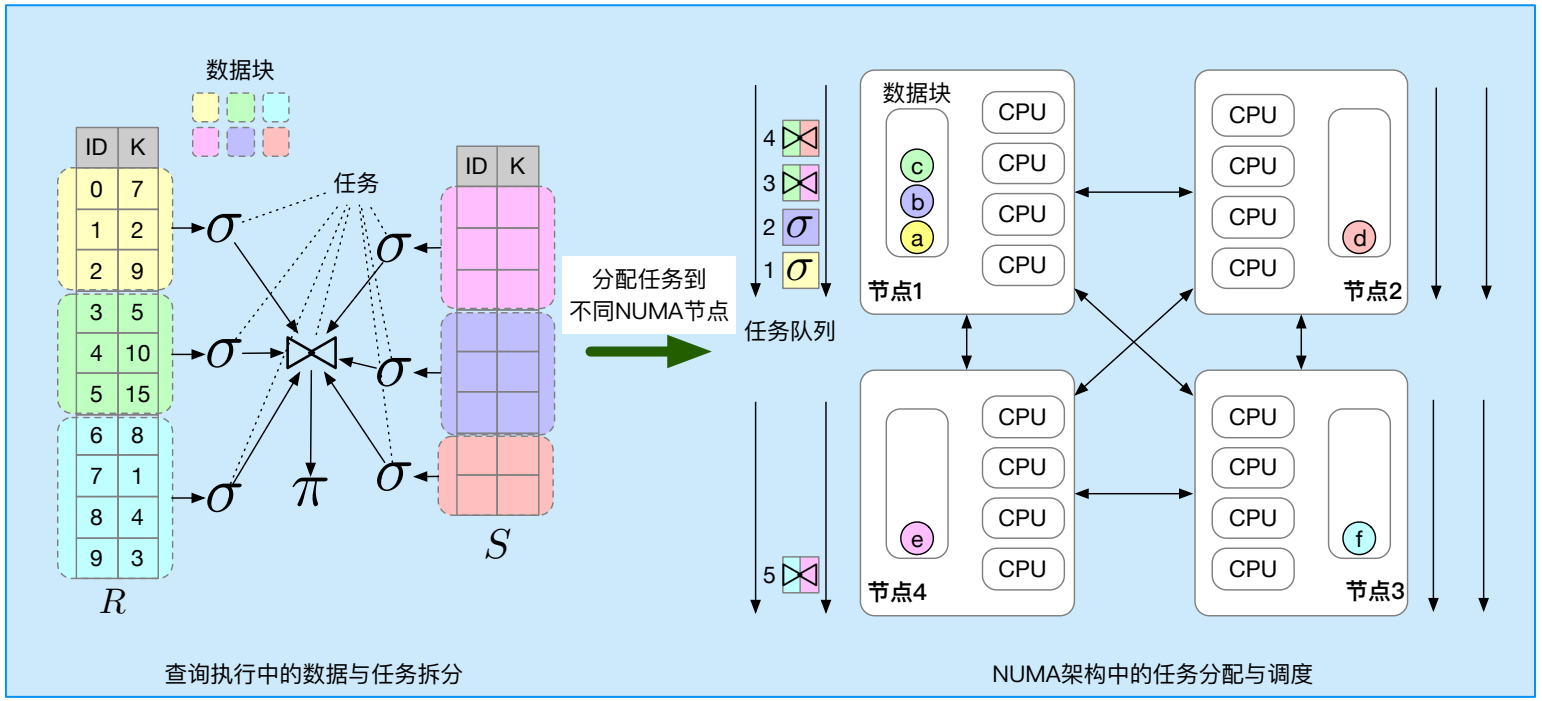


NUMA与并行计算

- NUMA感知的查询执行
- 任务偷取（work stealing）
- 数据位置调整（data replacement）

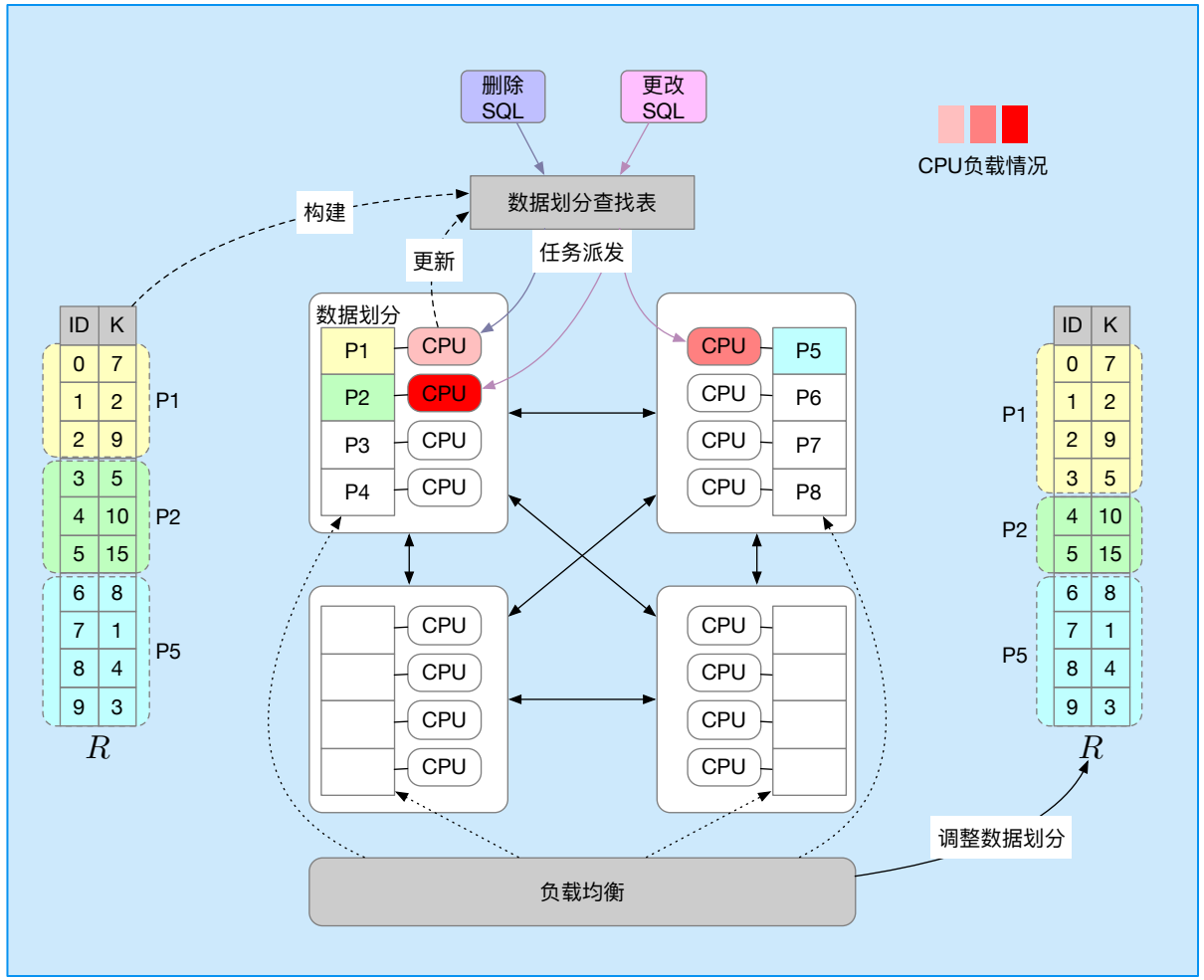






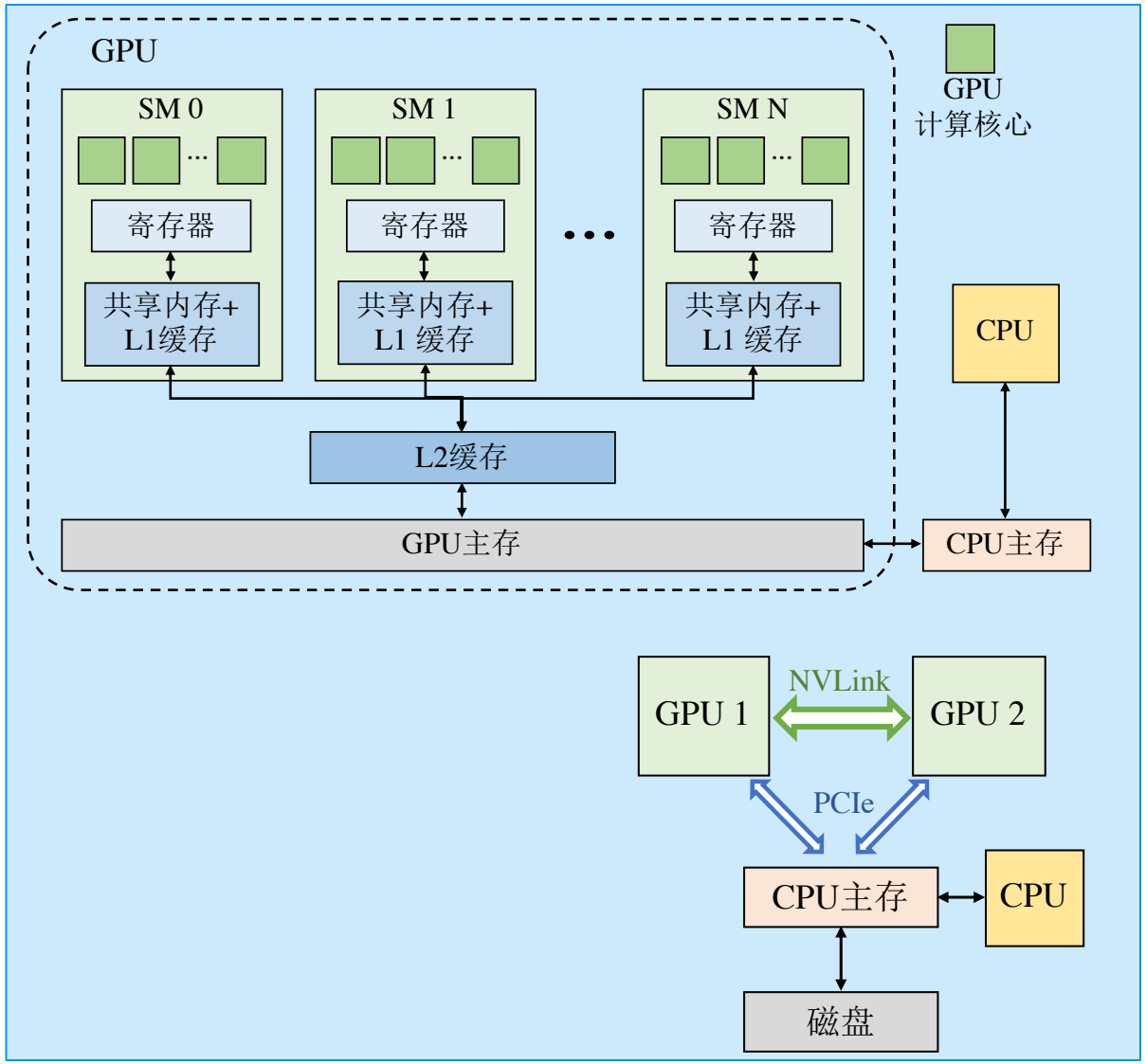
NUMA与事务处理

- **数据划分**：将数据均匀切成多个数据划分，不同划分的数据之间无交集，分配到不同CPU。
- **事务派发**：每个CPU负责执行其对应数据划分中的数据处理请求（包括插入、删除、更改、查找等），当数据库接收到事务请求时，通过数据划分查找表找到该事务涉及到的所有数据划分，将事务派发给对应的CPU处理。
- **动态数据划分调整**：一个全局的负载均衡模块定期地监控系统中各CPU的负载，若发现不均衡的情况，则调整数据划分方式使负载更为均衡，并通知各个CPU通过互相交换数据来调整各自数据划分。



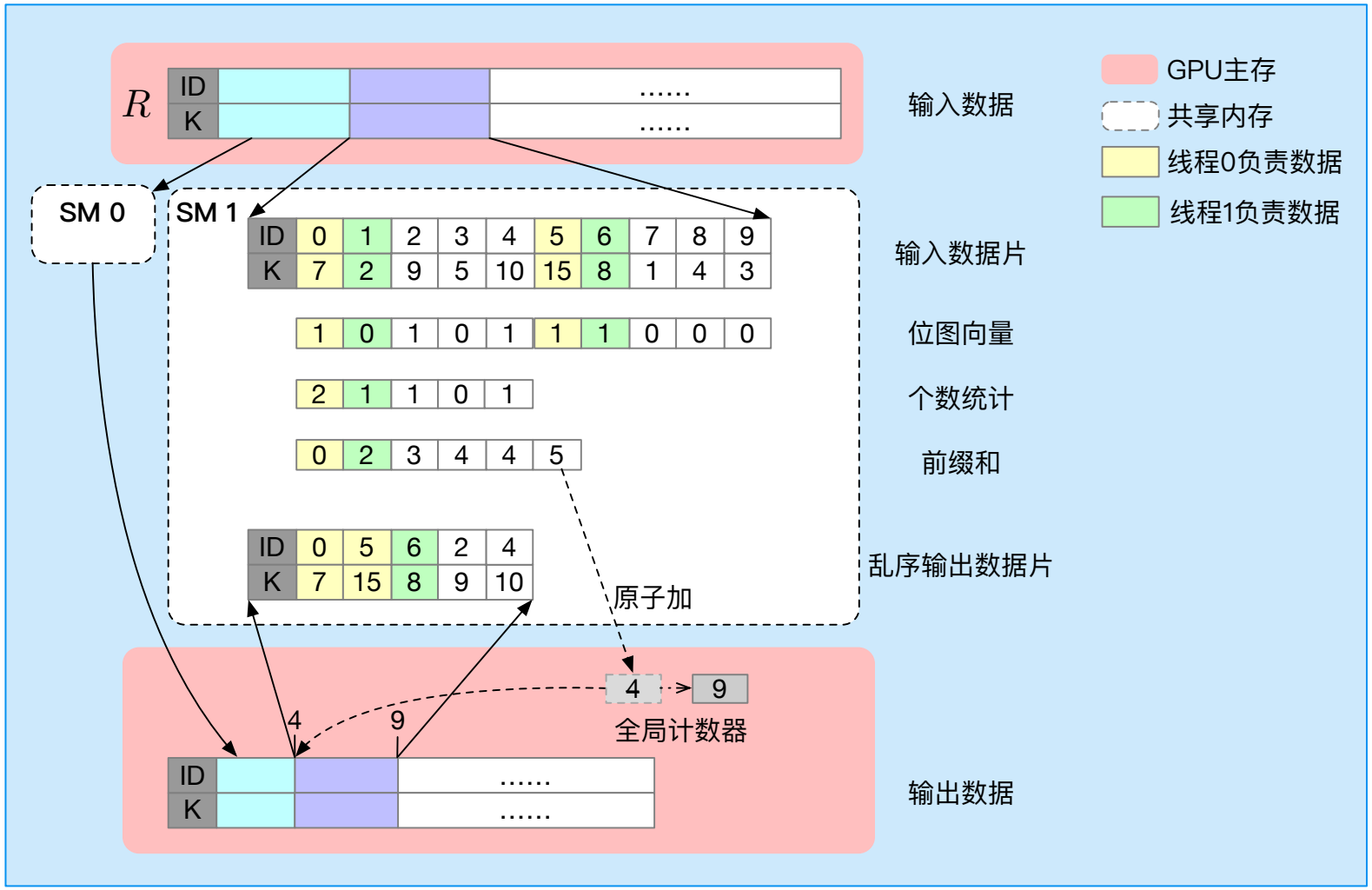
GPU与并行计算

- **GPU特性**
  - 大量核心
  - GPU核心与GPU主存间的传输速度高
  - 大量简单运算可快速并行执行
- **数据库与GPU**
  - 充分利用有限的GPU主存
  - 充分利用GPU所有核心进行并行计算
  - 尽可能少利用中心化组件（如全局计数器）以避免并发冲突
  - 数据压缩
  - 选择操作与连接操作的GPU并行计算



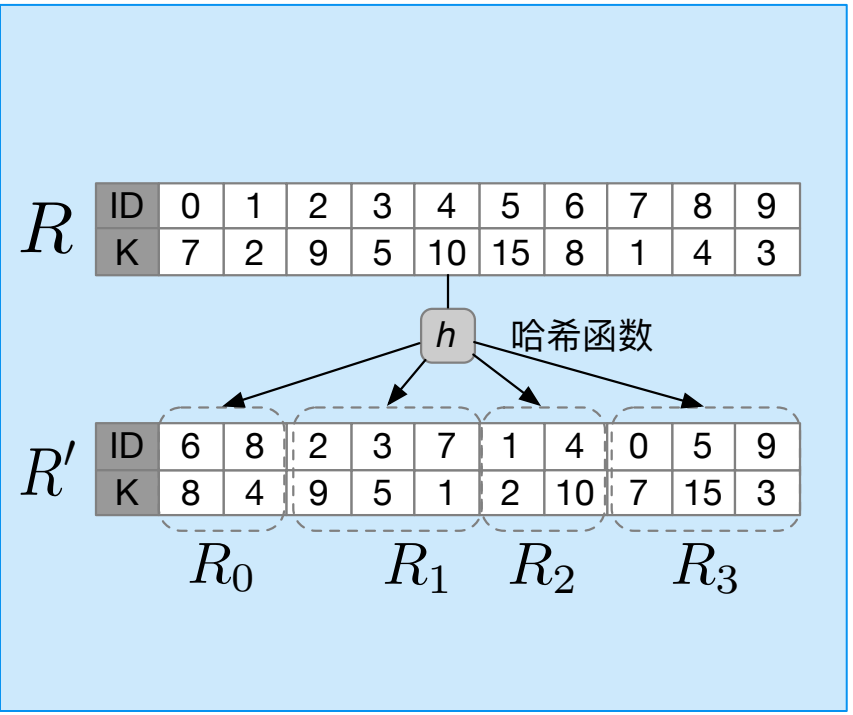
GPU 与并行选择

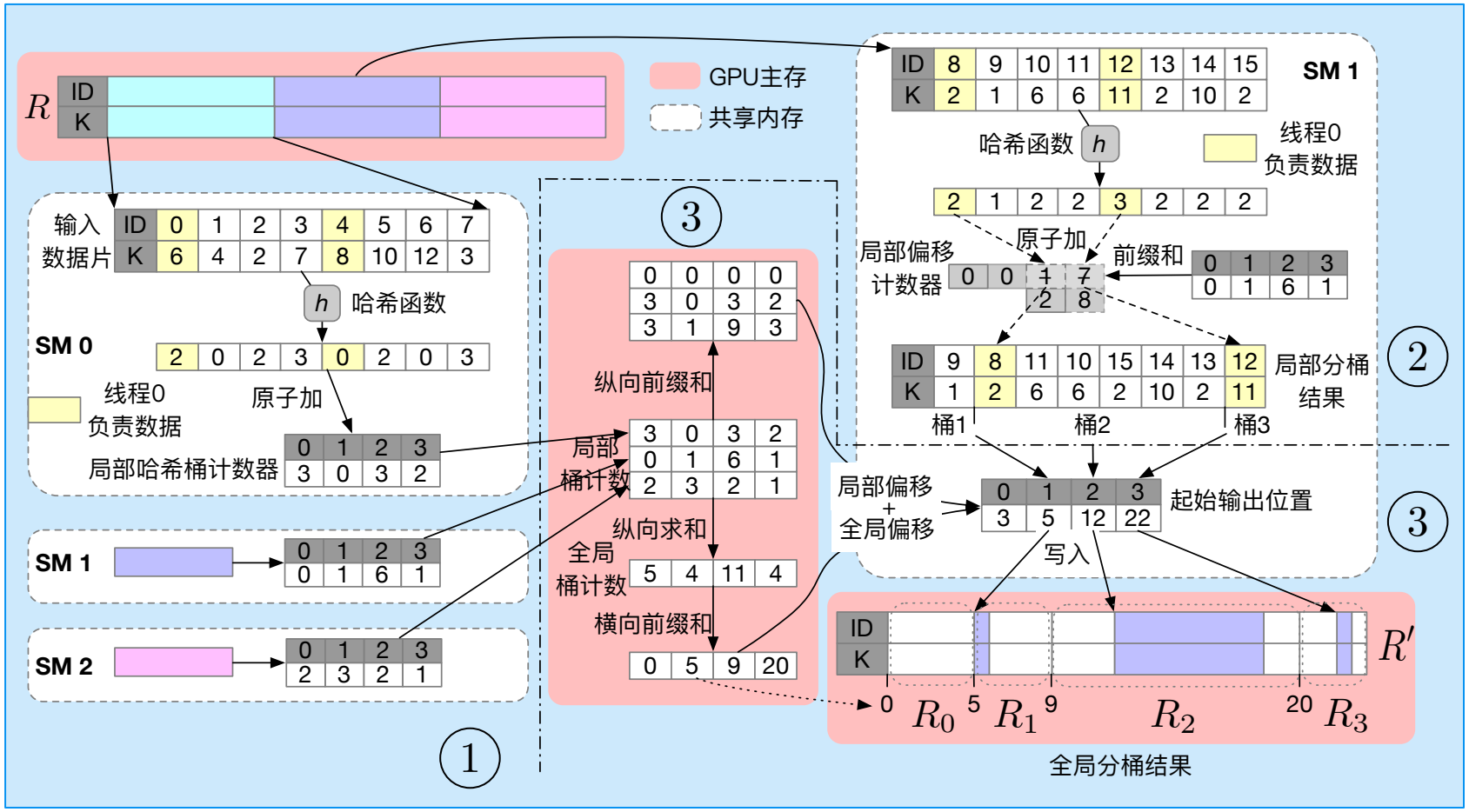
- 按照SM（streaming multi-processors，流处理器）分数据片



GPU 与并行连接

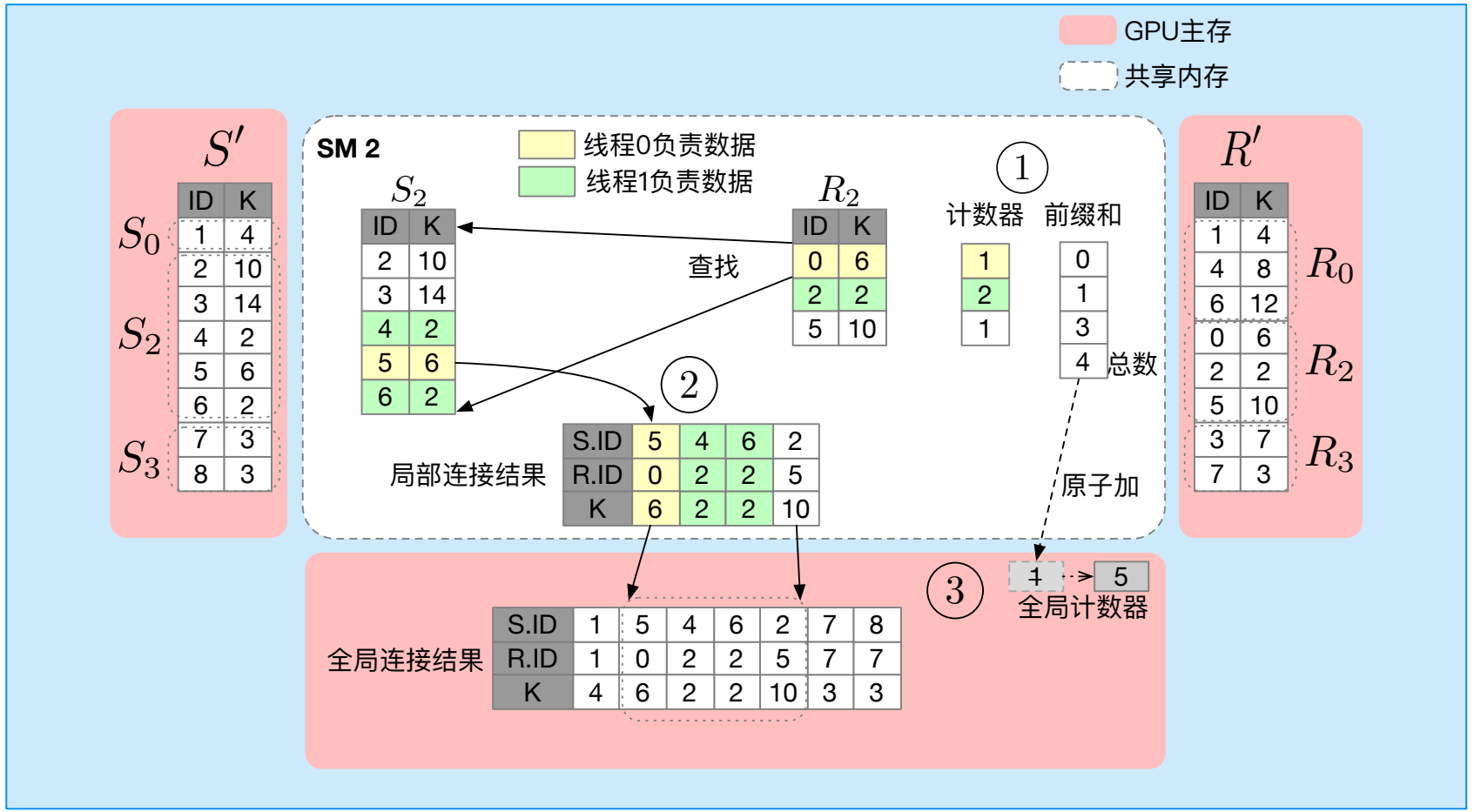
- **分桶：在GPU中应将各桶共同放置在GPU主存中的连续空间中，以尽量节约GPU主存空间，同时避免碎片化读写GPU主存以提高访存效率。**





GPU 与并行连接

- 归并：确定SM中各线程负责记录查找到的连接结果应放在局部连接结果中的何处，以及确定各SM的局部连接结果应放在全局连接结果中的何处



FPGA特性

➢ FPGA特性

- 硬件编程
- 能够以很快的运算速度实现固定的简单逻辑处理

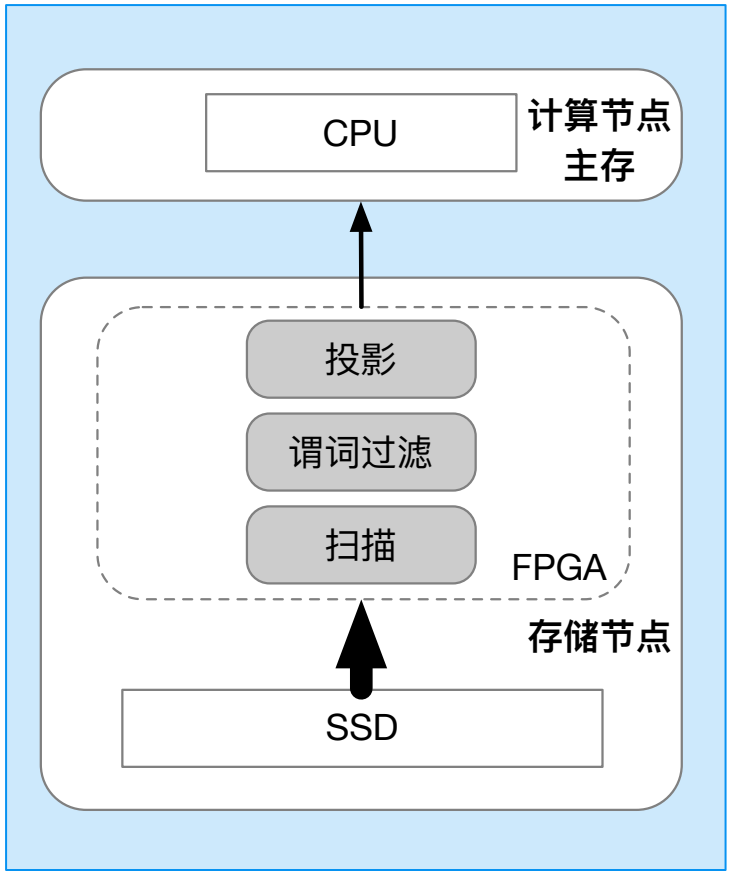
	ASIC	FPGA	CPU
灵活性	低	较高	高
运算速度	高	较高	低
功耗	较低	低	高

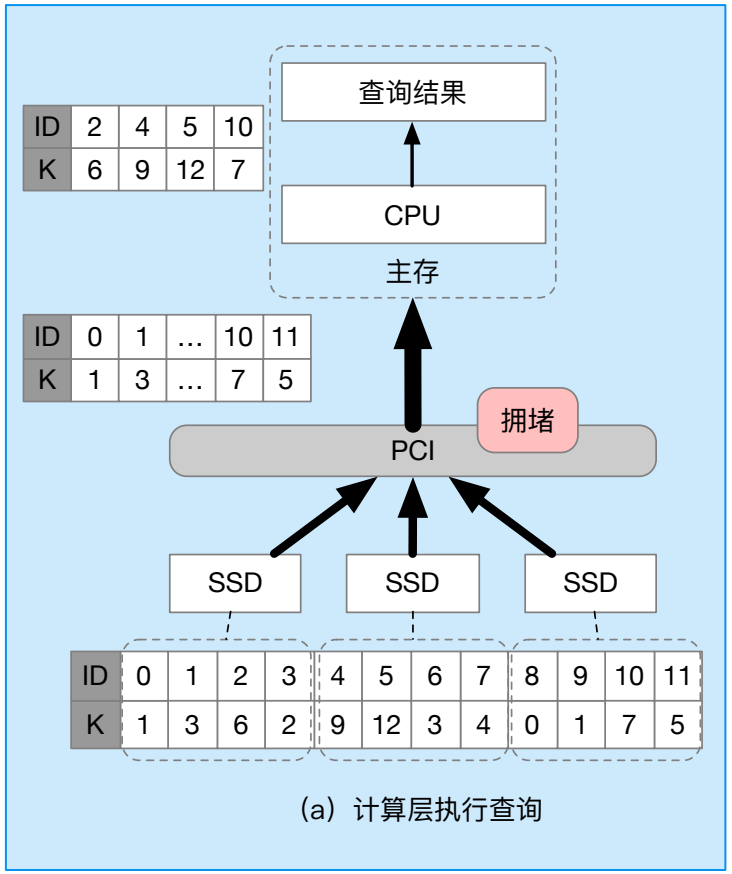
➢ 数据库与FPGA

- 查询下推：在分布式场景下，将FPGA部署在存储节点上，使用FPGA提前执行部分查询操作（如根据选择操作的条件过滤数据），然后再将结果传输到计算节点，就可以减少待传输的数据量从而减轻数据传输总线的压力
- 基于压缩的I/O加速：利用FPGA这一硬件来完成数据的压缩和解压操作，减少数据的I/O数据量

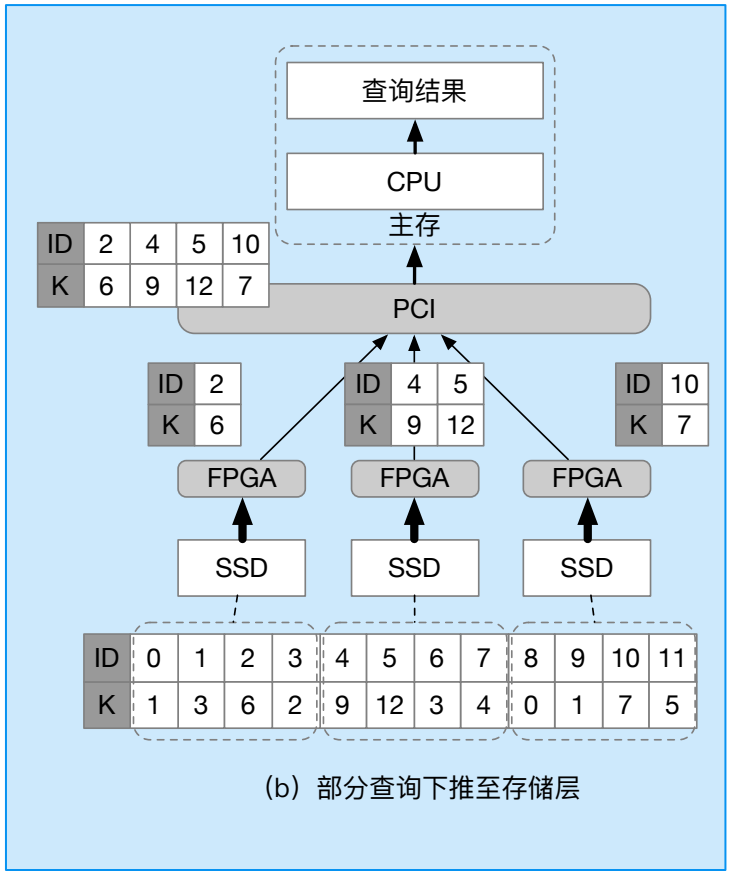
FPGA与谓词下推

- 将FPGA部署在存储节点上，在此过程中提前执行部分查询操作（如过滤），再将得到的结果传输到计算节点，大大减少传输路径上的数据量，即查询下推（offload）。





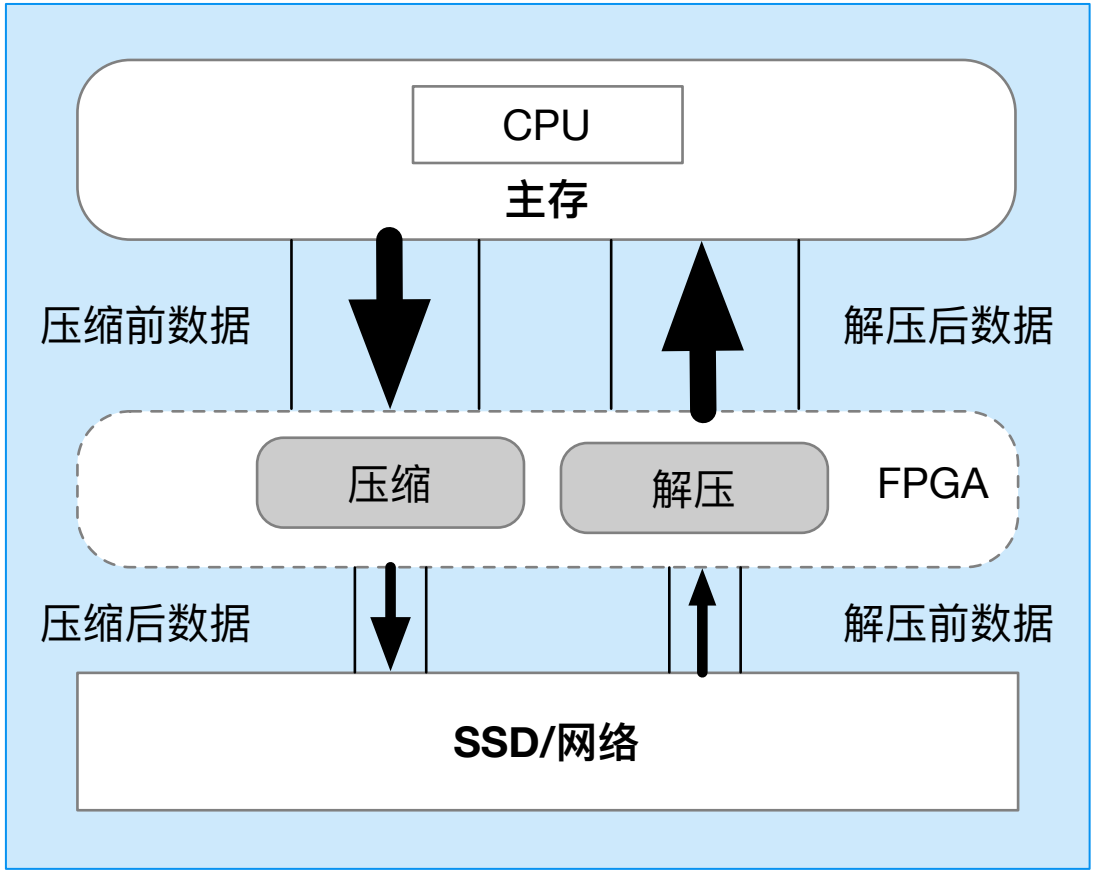
(a) 计算层执行查询



(b) 部分查询下推至存储层

FPGA与基于压缩的I/O加速

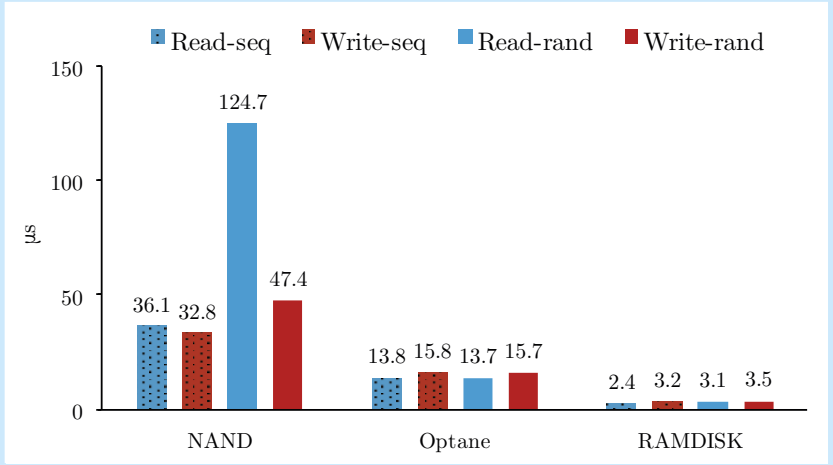
- **动机：**硬盘、网络的I/O速度远小于主存读写、计算处理速度。系统会采用压缩存储或压缩传输的方式减少I/O数据量。但是加解压开销大。
- **方案：**利用FPGA来执行压缩和解压计算，从而加速I/O操作。



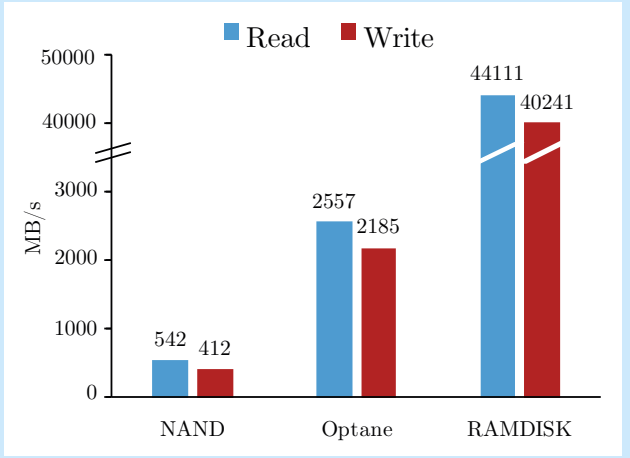
NVM特性

- **NVM特性**
  - 非易失性存储介质
  - 高随机读写速度
  - 按字节寻址
  - 写速度慢于读速度，且对内存单元的过度写操作可能会造成损坏

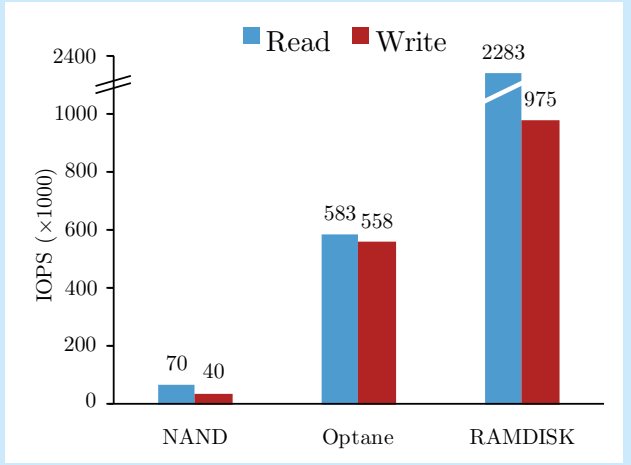
	Byte Addressable	Volatile	Speed
SRAM	Yes	No	Fastest
DRAM	Yes	No	Faster
NVM	Yes	Yes	Fast
NAND	No	Yes	Slow



(a) Latency



(b) Bandwidth

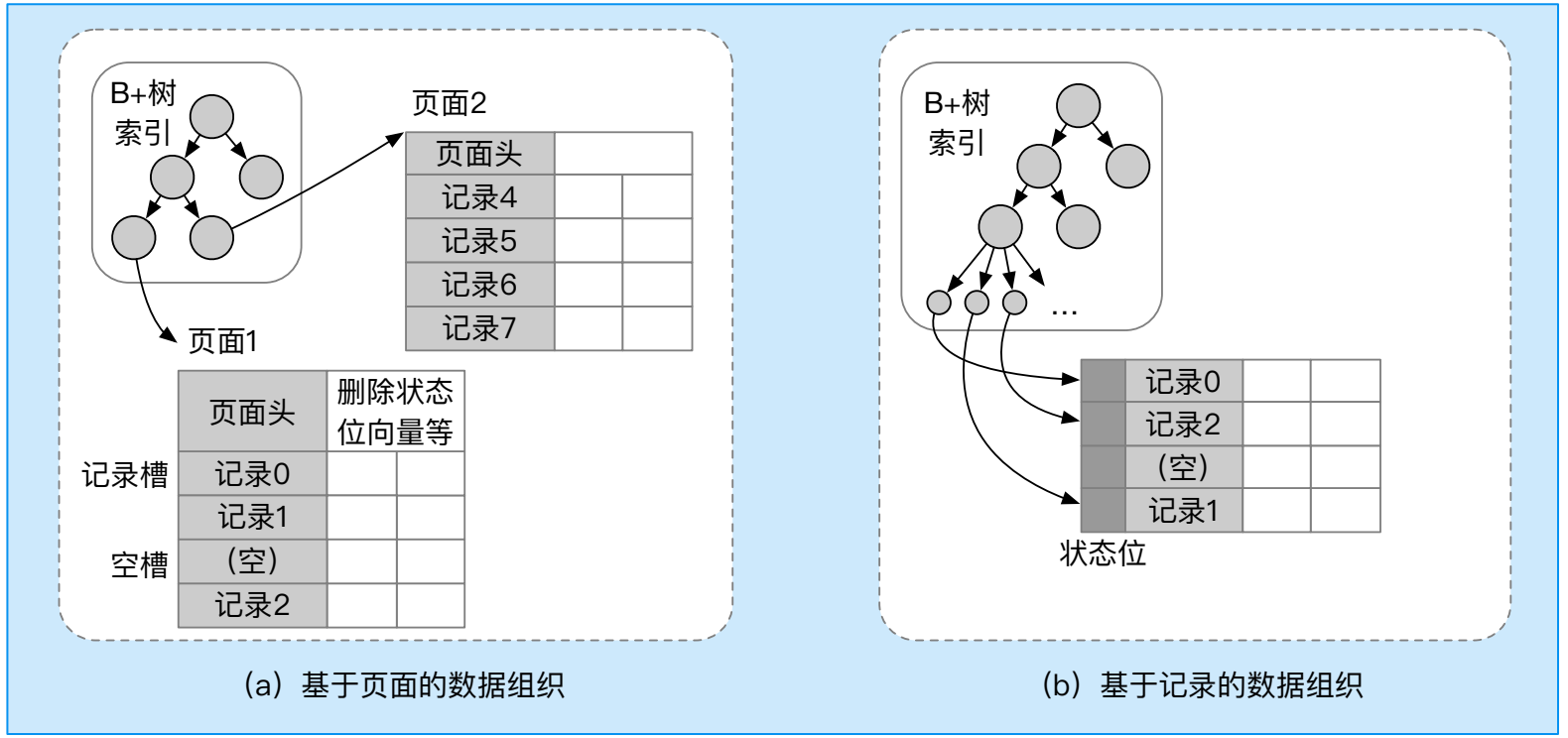


(c) IOPS

NVM与数据组织

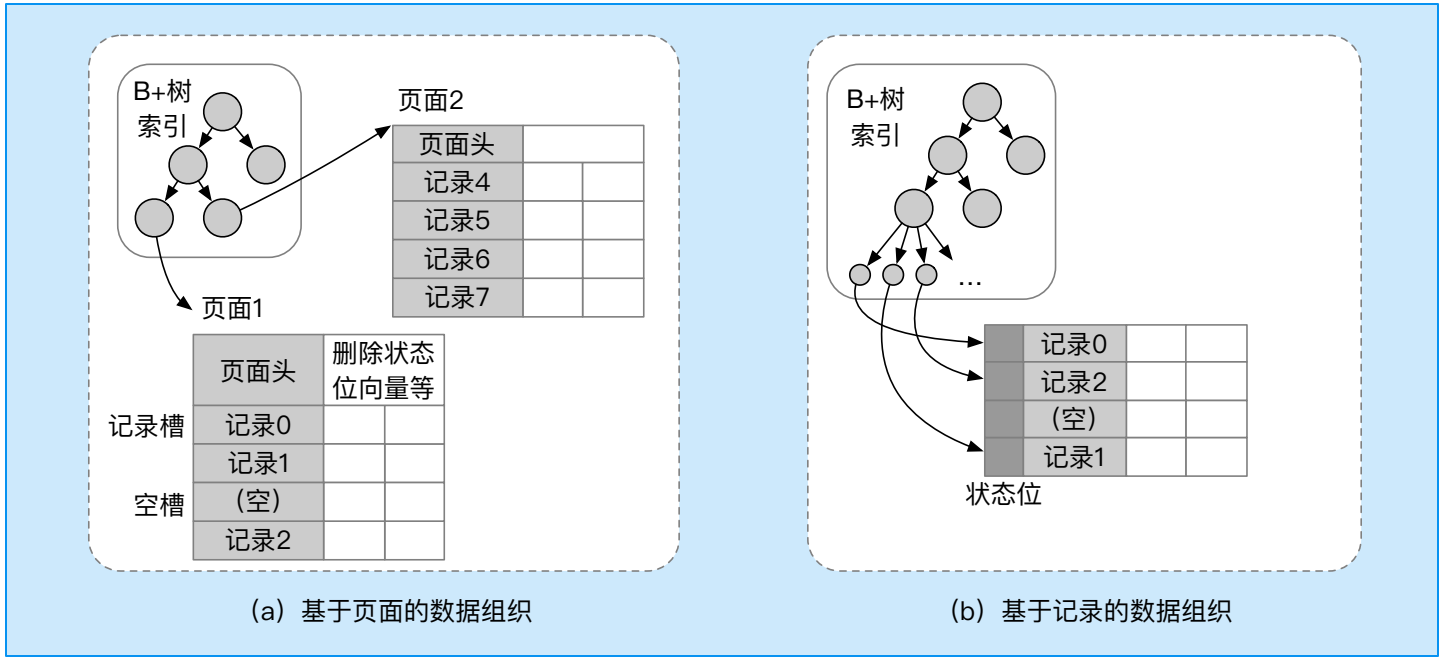
➢ 数据库与NVM

- 数据存储：直接在NVM上进行数据读写，按字节寻址
- 事务处理：快速读写支持STEAL+FORCE条件下的回复算法，仅使用undo日志
- 查询处理：将无法放入主存中的大表放置在NVM中从而加速数据访问



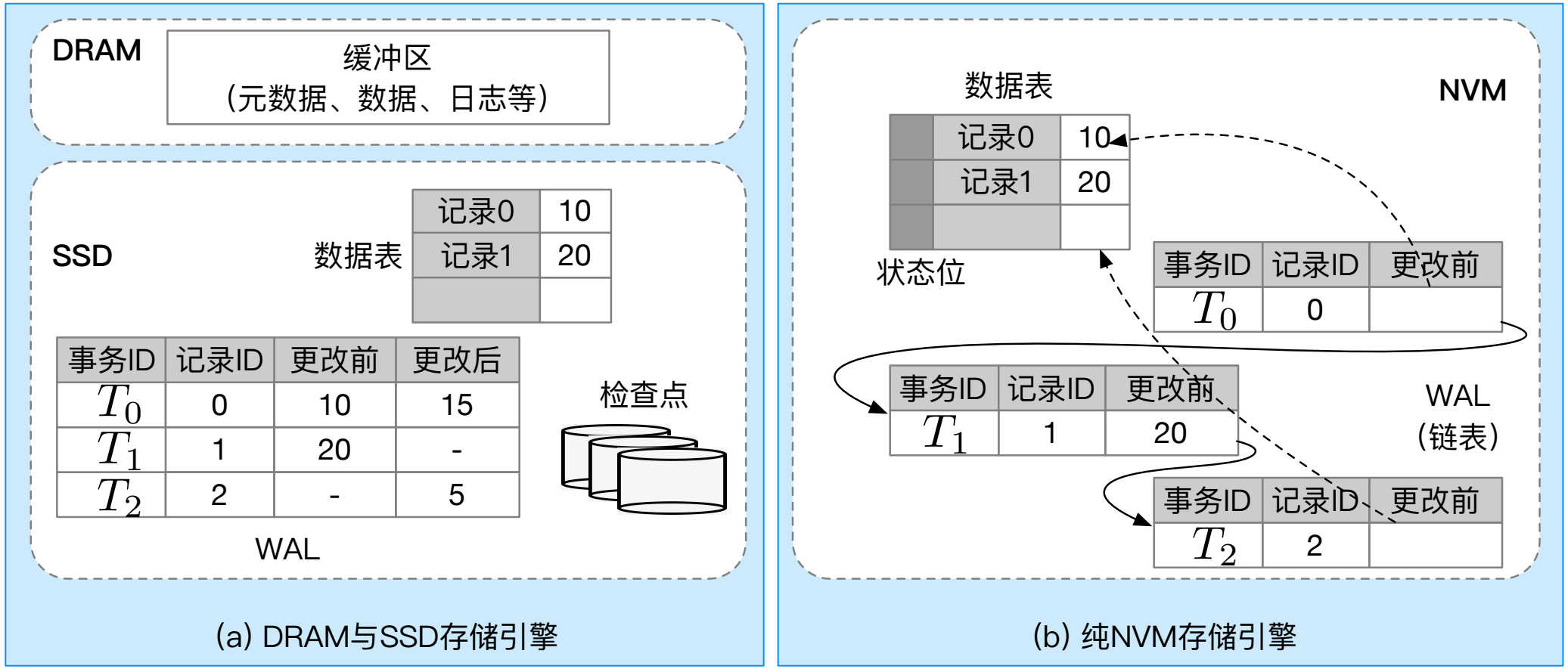
NVM与数据组织

- 用B+树来索引表的主键，每个中间节点可以包含任意个索引项（而不一定是以页面为单位组织节点），每个底层节点为一个记录。
- 仍在连续空间上以槽为单位组织各记录，其中每个记录多保留一个状态位，除了标识此槽是否有记录之外，还标识记录是否已被持久化保存。
- 以字节为粒度与上层缓存进行数据交换，而无需整页替换，更为充分地利用缓存空间。



NVM与故障恢复

- **Force、Steal方式**



NVM与MVCC

- 持久化提交序列号Cp代表此时所有LSN不超过Cp的事务都已提交；
- 脏提交序列号Cd是系统分配的一个序列号，系统会确保下一次事务提交产生之前，不会有序列号超过Cd的事务进行NVM的刷写。
- 故障时，取最后一条日志项的Cp和Cd，则不超过Cp的事务都已提交成功，其对应的数据版本可见，而版本号大于Cp且小于等于Cd的数据需要回滚到Cp前的版本。
- 这一回滚可以“懒惰”地进行（按需），也就是不立即去找到这些数据版本将其删除，而是在查询过程中访问到该区间内的数据时直接忽略之。

状态位　　版本序列号范围

	记录0	(0，10)	10
	记录1	(0，11)	20
	记录0	(10，∞)	15
	记录2	(12，∞)	5

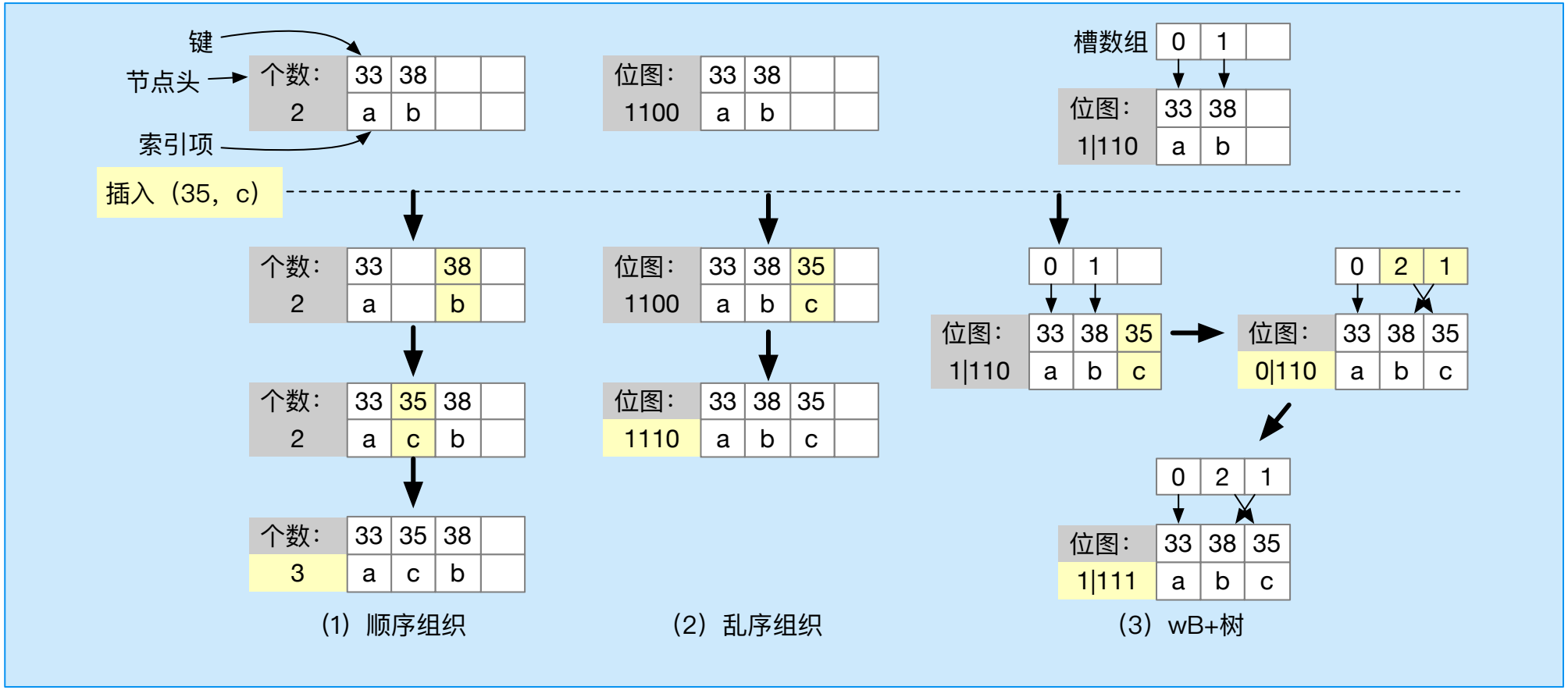
（a）MVCC数据表

事务ID	LSN	*Cp*	*Cd*
T_0	10	8	11
T_1	11	11	12
T_2	12	12	12

（b）WBL

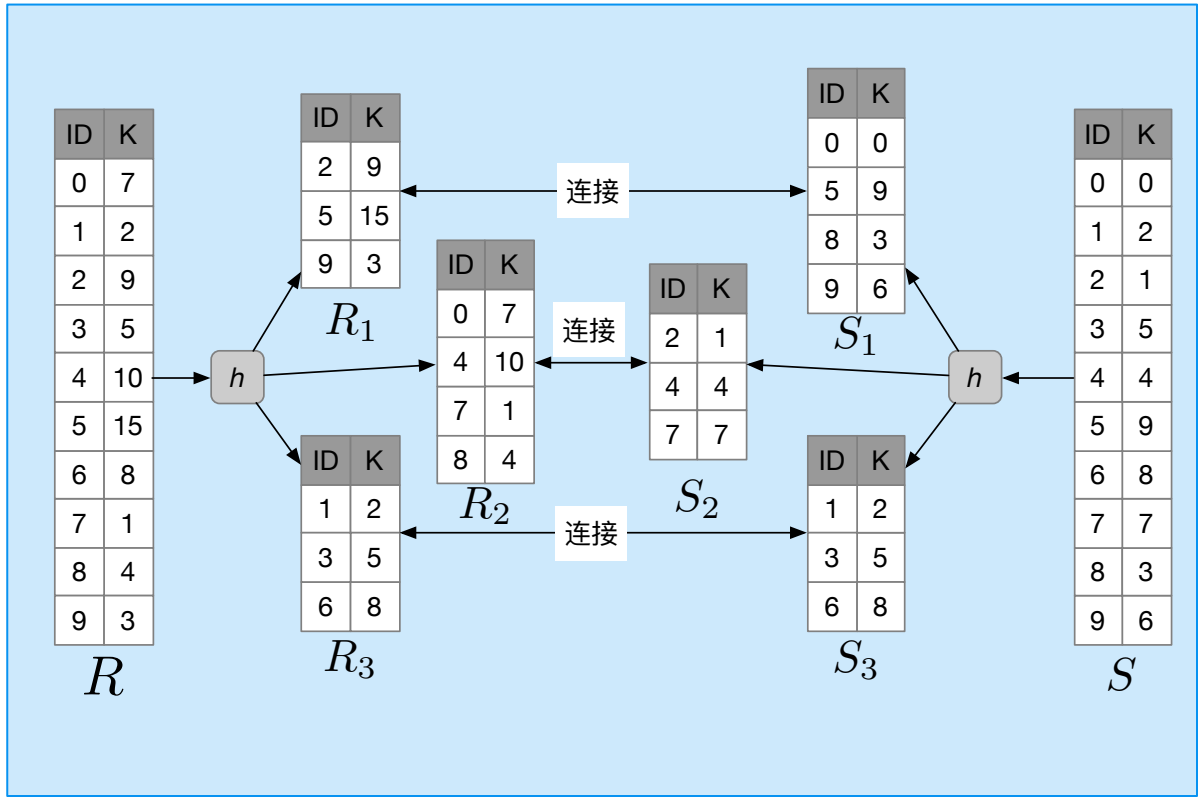
NVM与索引

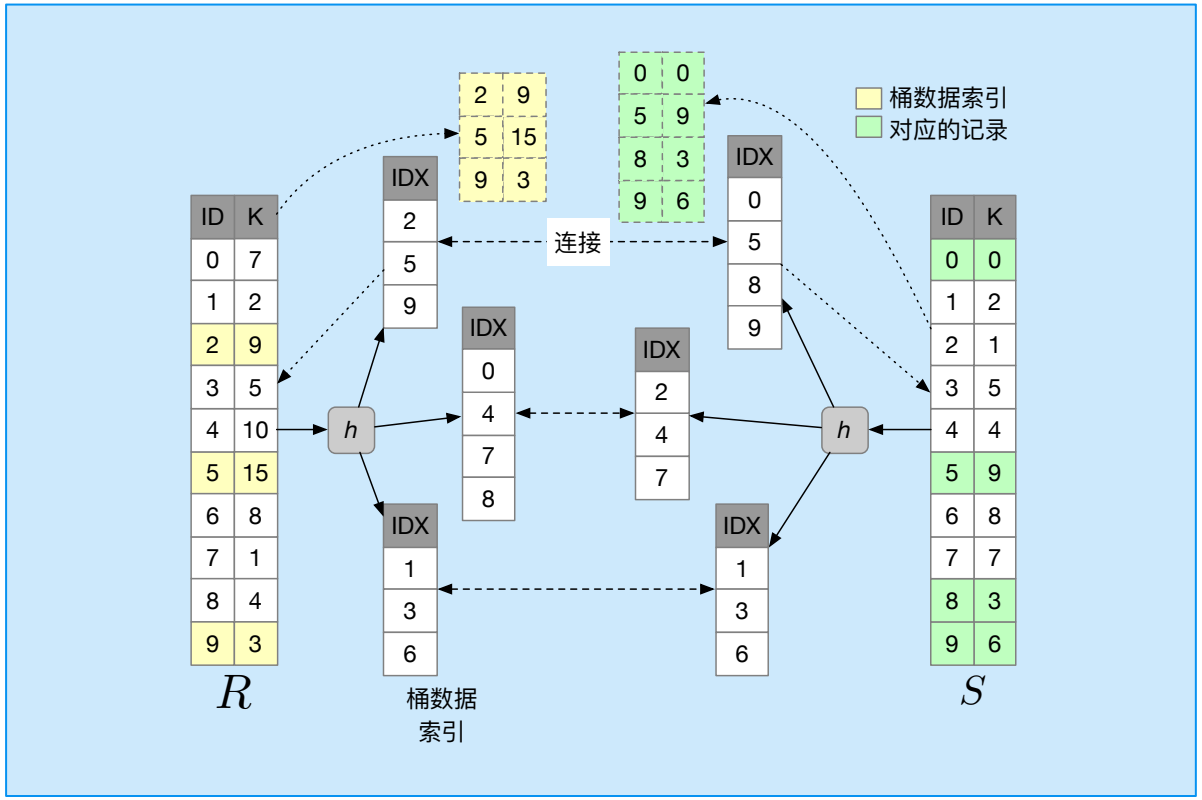
- 为了避免顺序组织中更新时大量的写操作（移动索引项），同时避免乱序组织中查找时低效的搜索，wB+树（write-atomic B+-tree，原子写B+树）节点的键值是乱序。



NVM与连接运算

- 为了降低哈希后空间占用，物理分桶改为虚拟分桶

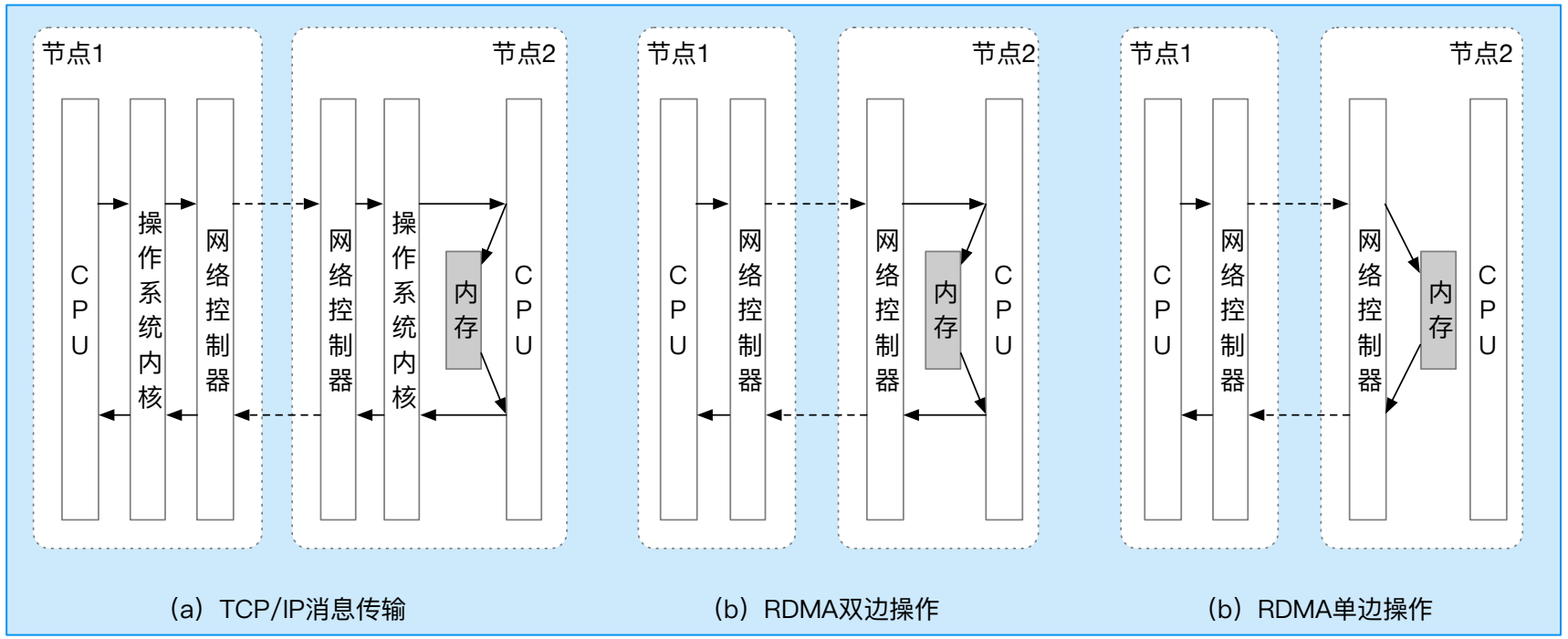




RDMA特性

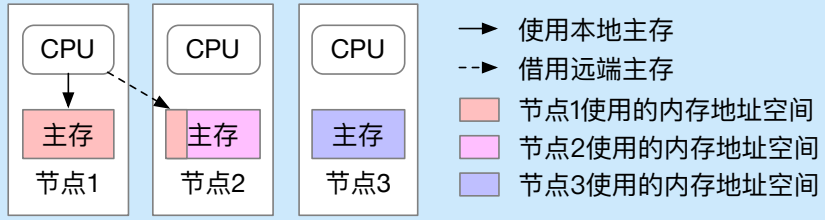
➢ RDMA特性

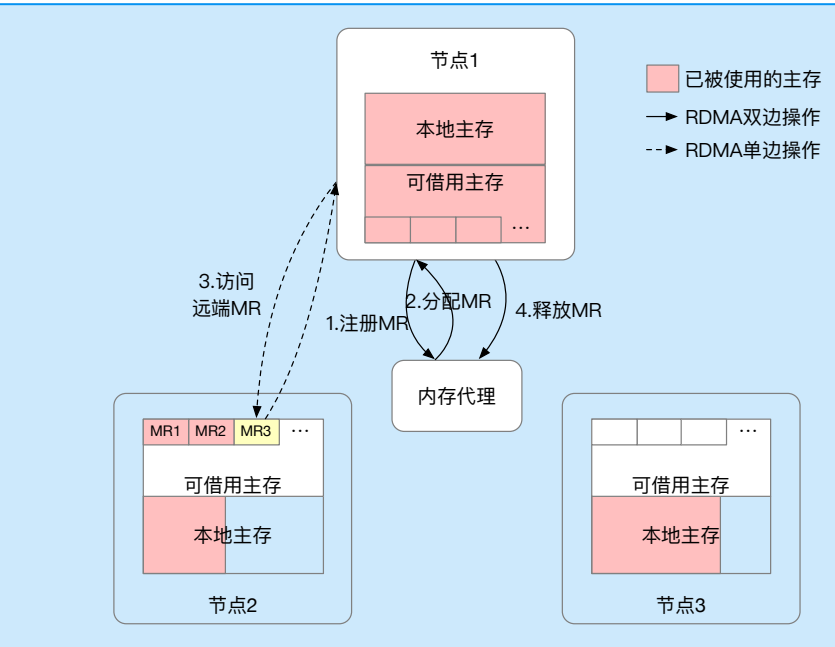
- 零拷贝：应用层直接将数据发送到缓冲区，目标机器直接从缓冲区接收，无需复制到网络层。
- 内核旁路：应用程序直接在用户态进行数据传输，无需切换到内核态。
- 不占用CPU：单边操作中，访问远程机器的主存无需该机器CPU参与，也不占用其CPU缓存。



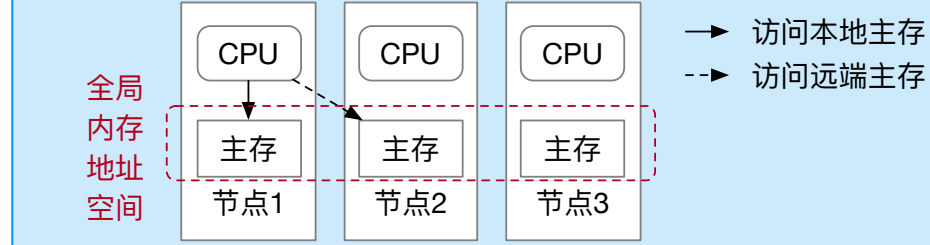
RDMA与分布式共享内存

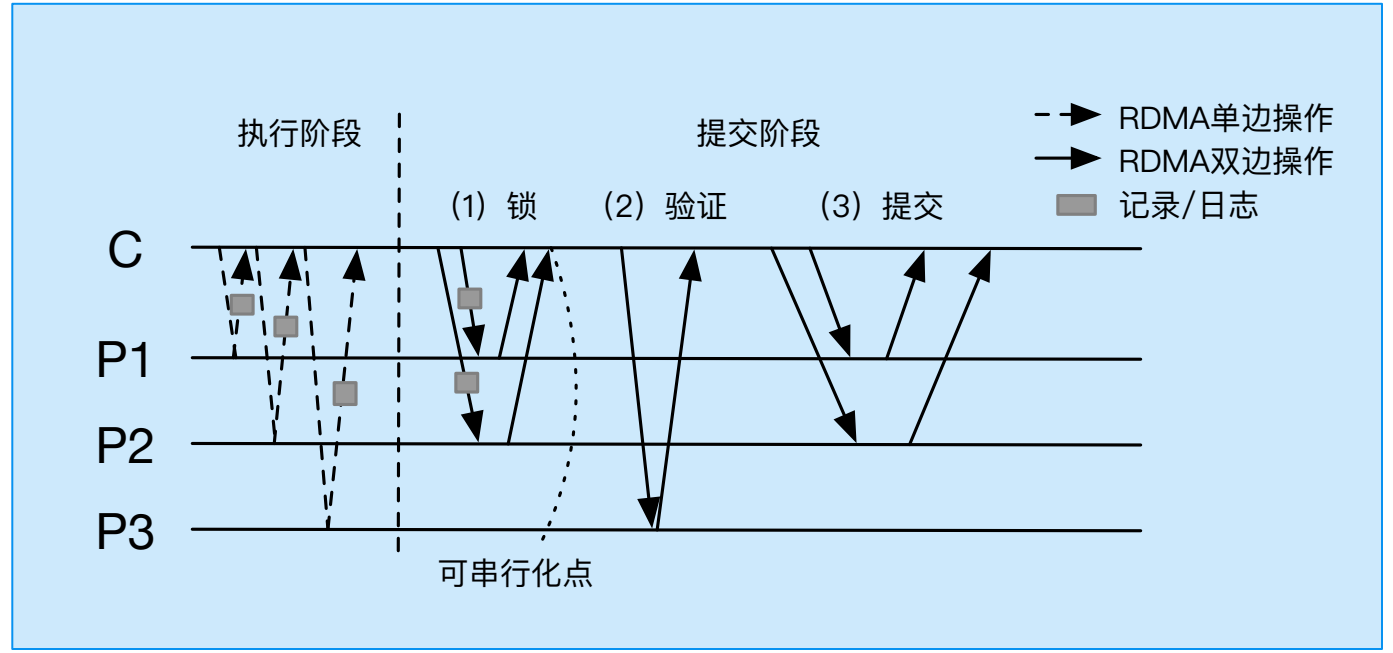
➢ 独立地址空间





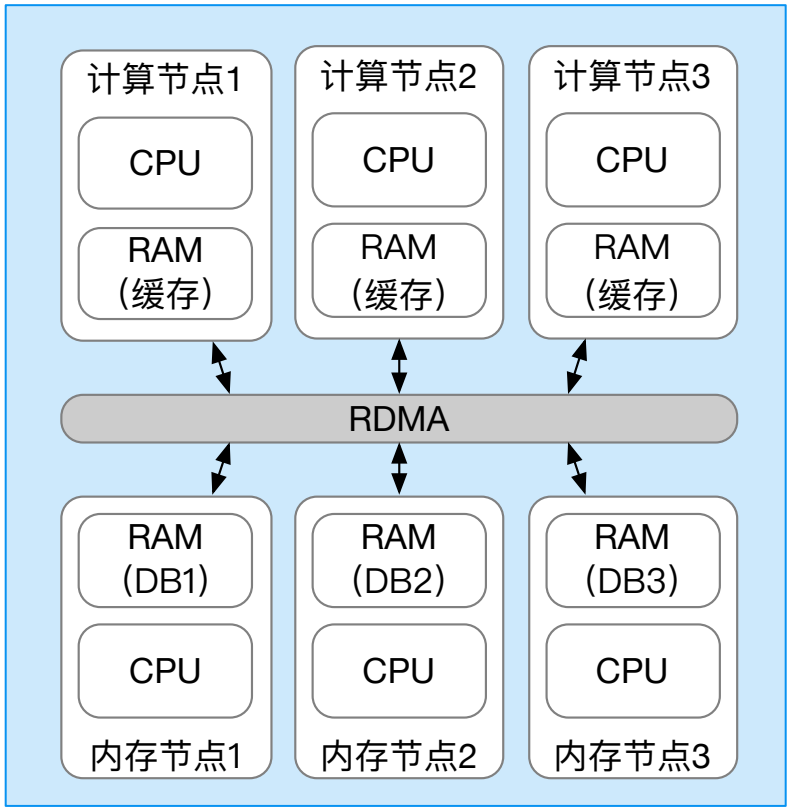
➢ 共享内存地址空间

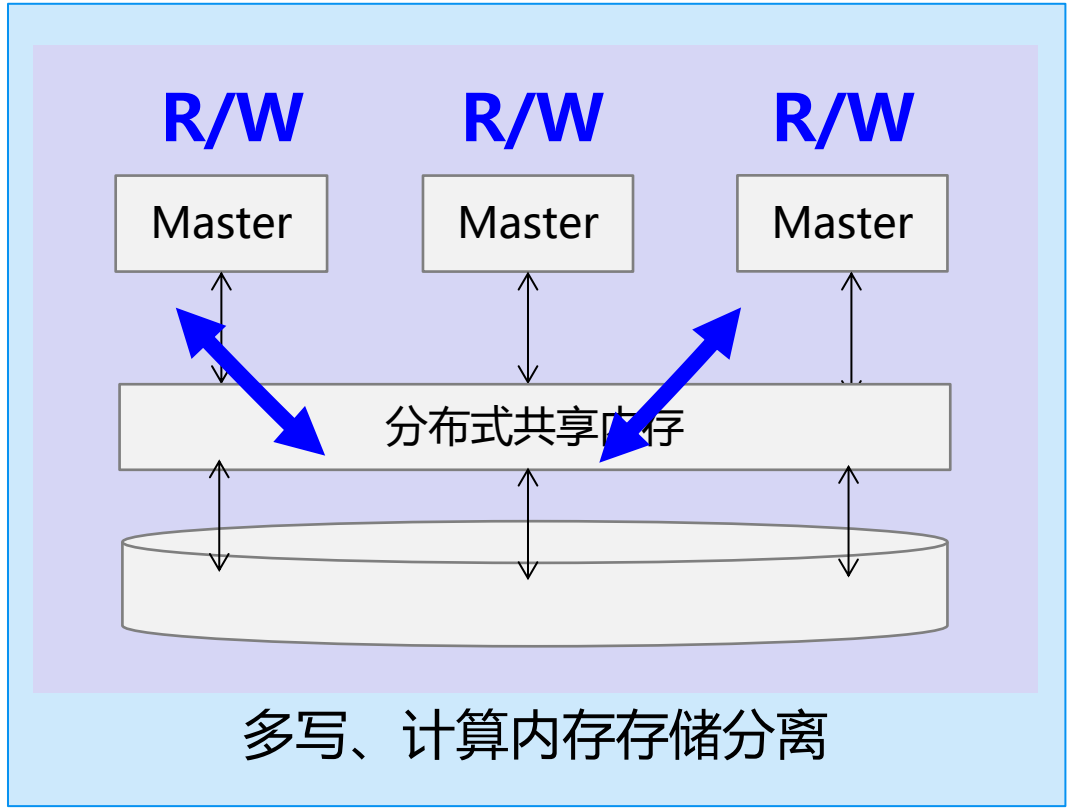




RDMA与分布式共享内存

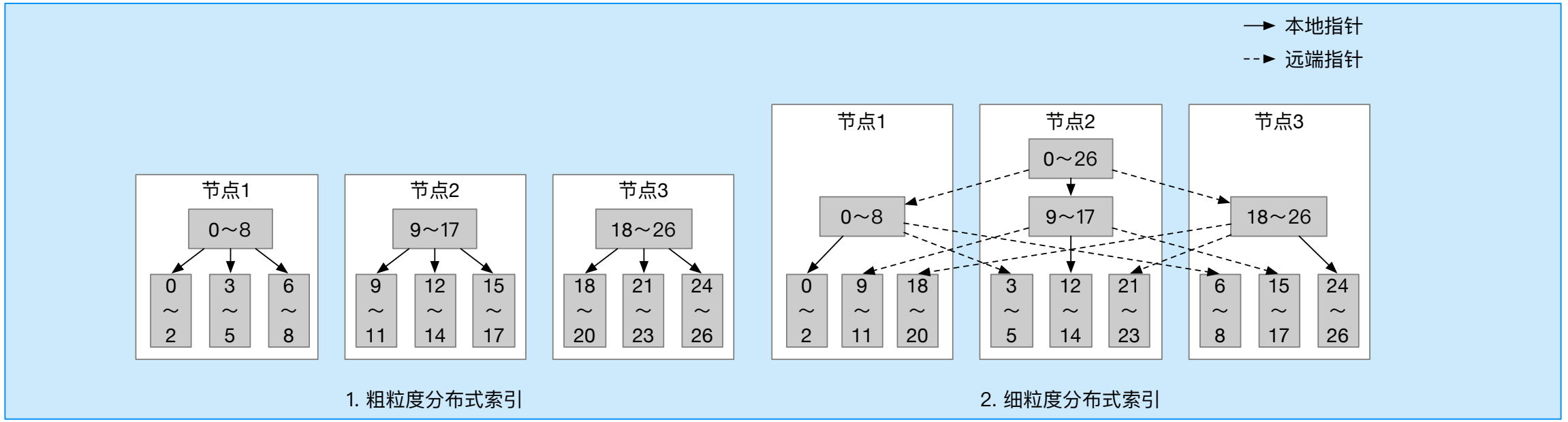
➢ 计算-内存-存储分离架构





RDMA与分布式索引

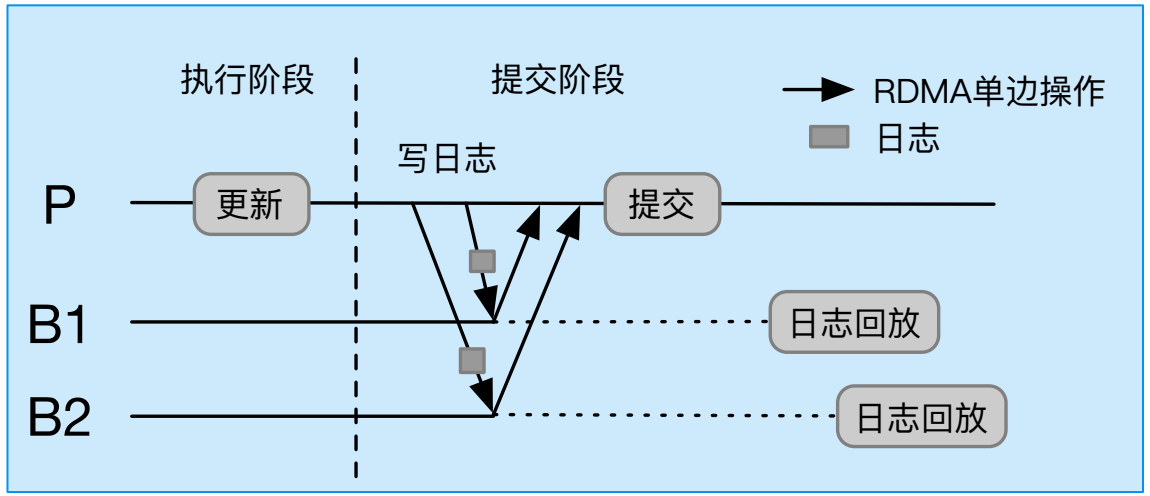
- **粗粒度索引**
  - 将索引的键均匀划分为若干个连续的部分，把不同部分放在不同的节点上，然后对各部分分别建立B+树索引
- **细粒度索引**
  - 对所有键统一建立一个全局索引，各个树节点分散在各个节点的主存中

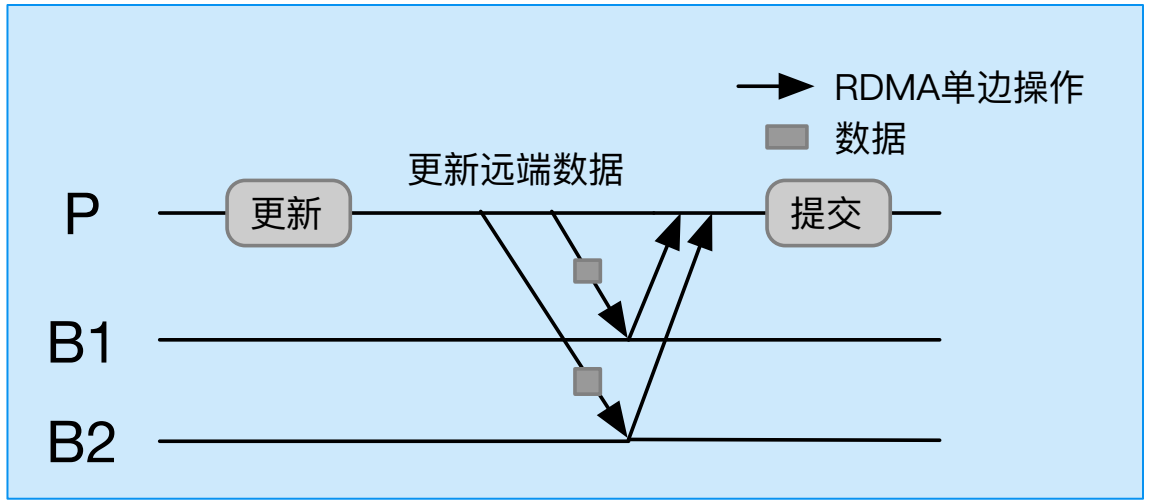


RDMA与高可用

- **主备间同步日志**
  - 主节点利用RDMA单边写操作将日志写入各个备节点

- **主备间同步数据**
  - 主节点在备份过程中不传输日志，而是直接将数据更新通过RDMA单边操作写入备节点的主存，这是原地更新备份方式





目录

07

NoSQL数据库

NoSQL数据库

- **NoSQL数据库**
  - Not Only SQL：指不同于关系型数据库的数据库管理系统
- **产生背景**
  - 互联网时代以来，数据指数级爆炸性增长
  - 半结构化、非结构化等异构数据处理需求增长
  - 关系型数据库的强一致性并非所有场景的刚需
- **NoSQL的特点**
  - 模式灵活：无需预先指定数据的模式，支持混合数据类型
  - 高可扩展：摆脱关系模型约束，简化提升了数据水平扩展能力
  - 高可用：基于主主或主从集群模式，支持用户从不同数据节点查询数据

NoSQL数据库

NoSQL = Not only SQL

模式灵活

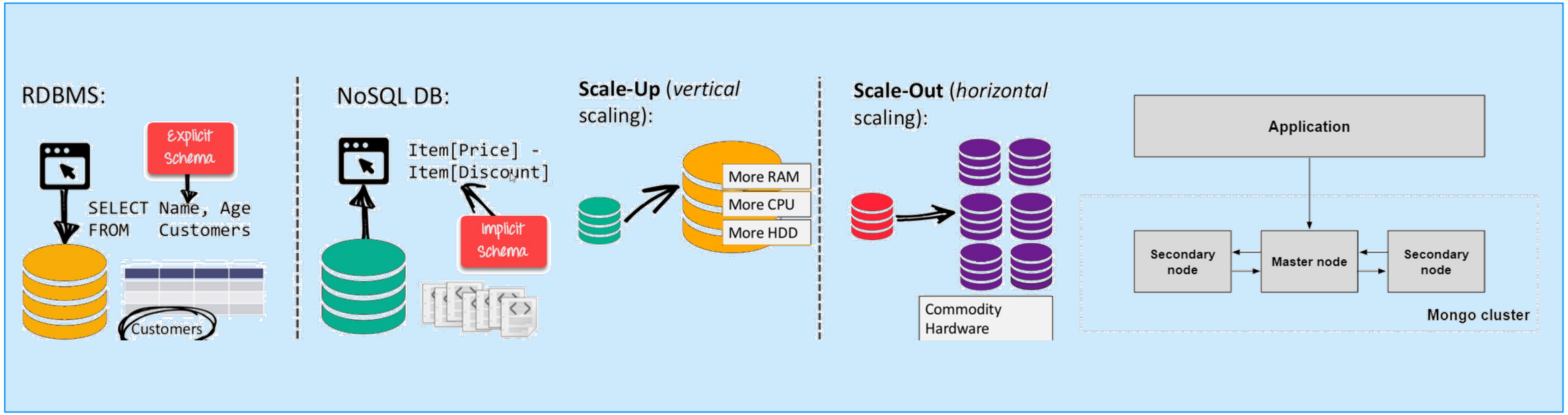
NoSQL数据库无需预先指定数据的模式，并支持混合数据类型

高可扩展

摆脱关系模型约束，NoSQL简化提升了数据水平扩展能力

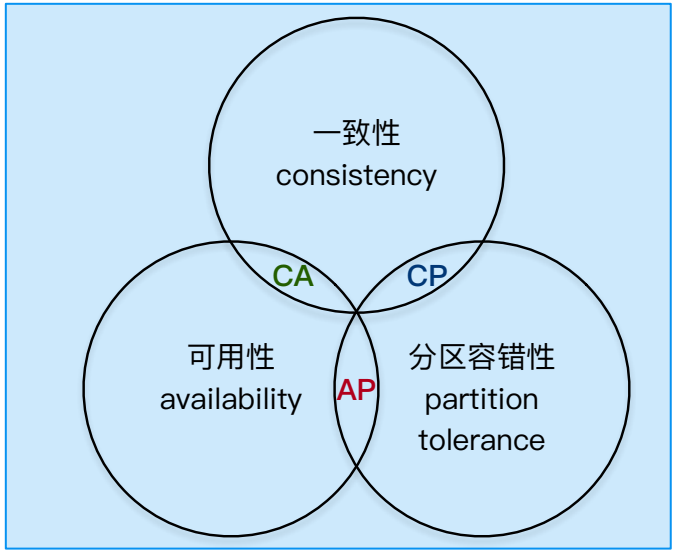
高可用

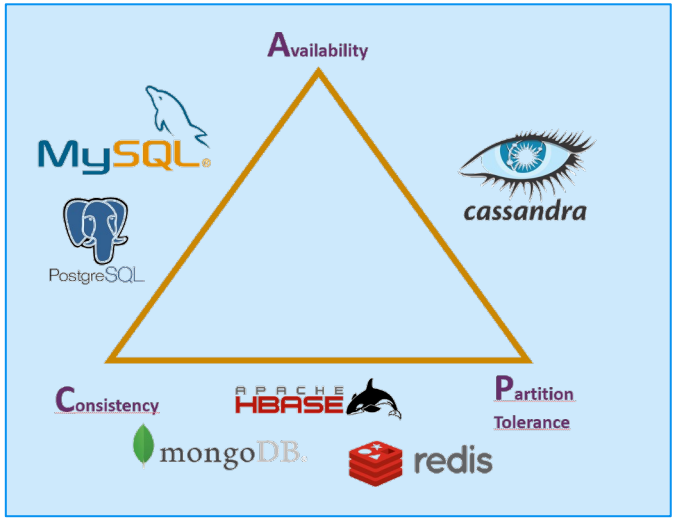
基于主主或主从集群模式，支持用户从不同数据节点查询数据



NoSQL数据一致性理论

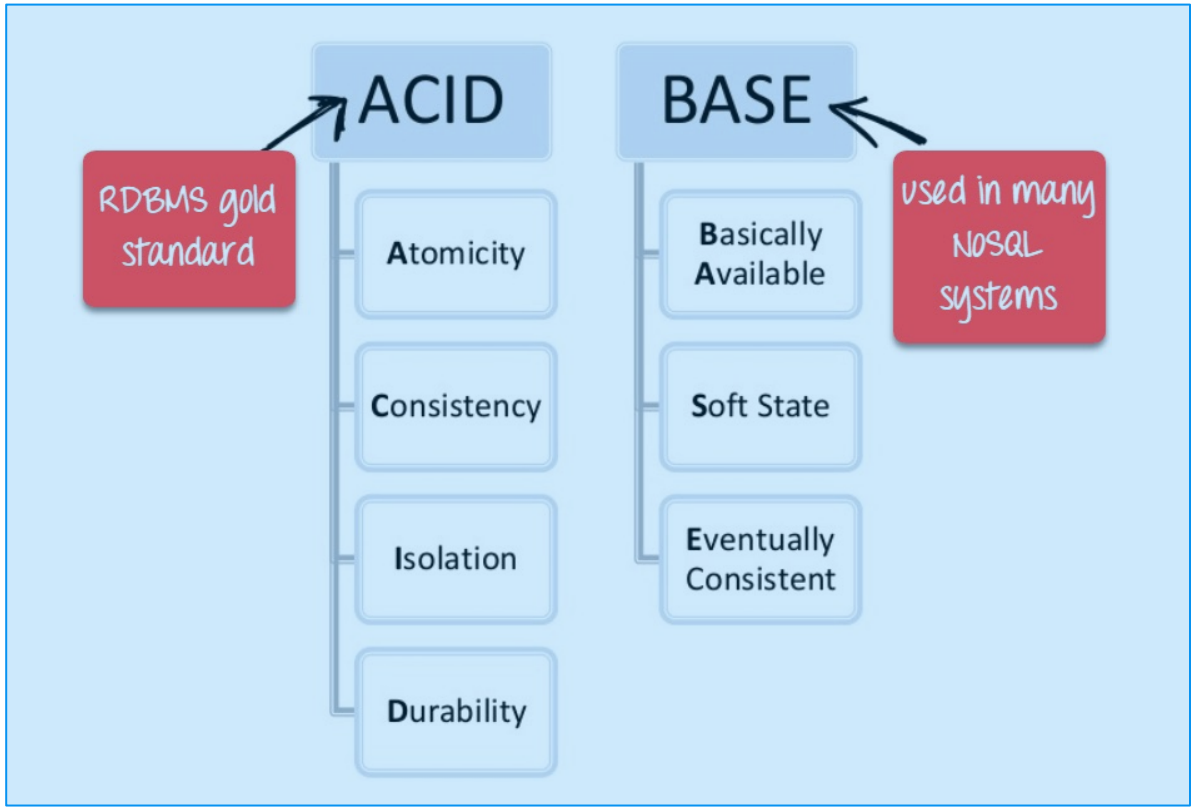
- **CAP理论**
  - 一个分布式系统只能保证一致性（consistency）、可用性（availability）和分区容错性（partition tolerance）三个特性中的任意两点。
- **分布式系统CAP理论：三种特性只能满足其二**
  - 一致性：连接不同节点同时查询的数据一致
  - 可用性：存在节点故障时仍可响应客户端请求
  - 分区容错：存在网络中断时系统仍正常服务
- **CAP数据库分类：**
  - AC数据库：牺牲分区容错，如银行场景
  - AP数据库：牺牲一致性，如电商场景





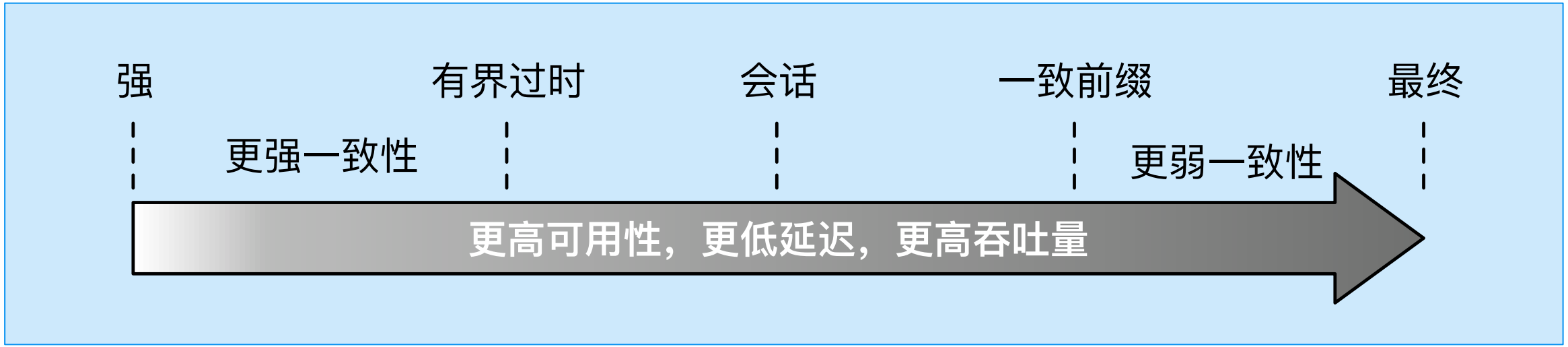
NoSQL的BASE特性

- **BASE特性：NoSQL一定程度上牺牲一致性，实现数据可用可扩展**
  - 基本可用性：数据拥有多节点备份，可以保证少量节点故障下的数据可用
  - 软状态：数据节点可以向客户端反馈过时数据
  - 最终一致性：系统最终会完成多备份节点的写操作，实现多节点数据一致



NoSQL数据一致性

- **NoSQL根据不同场景的需求，降低数据一致性，以换取更高的数据可用性**
  - 一致性由强至弱包括：强一致性、有界过时一致性、会话一致性、前缀一致性和最终一致性



NoSQL一致性

强一致性：读到最新版本数据

		W1	W2	W3	W4	W5	W6	…
地区 A	写操作							
地区 A	读操作	R1	R2	R3	R4	R5	R6	…
地区 B	读操作	R1	R2	R3	R4	R5	R6	…
地区 C	读操作	R1	R2	R3	R4	R5	R6	…
		T1	T2	T3	T4	T5	T6	时间单元 T

有界过时一致性：保证落后于写操作k个版本数或者T个时间单元以内（过时窗口）

		W1	W2	W3	W4	W5	W6	…
地区 A	写操作							
地区 A	读操作	R1	R2	R3	R4	R5	R6	…
地区 B	读操作				R1	R2	R3	…
地区 C	读操作				R1	R2	R3	…
		T1	T2	T3	T4	T5	T6	时间单元 T

最大滞后100个版本
或者3个时间单元

会话一致性：在会话层面数据一致

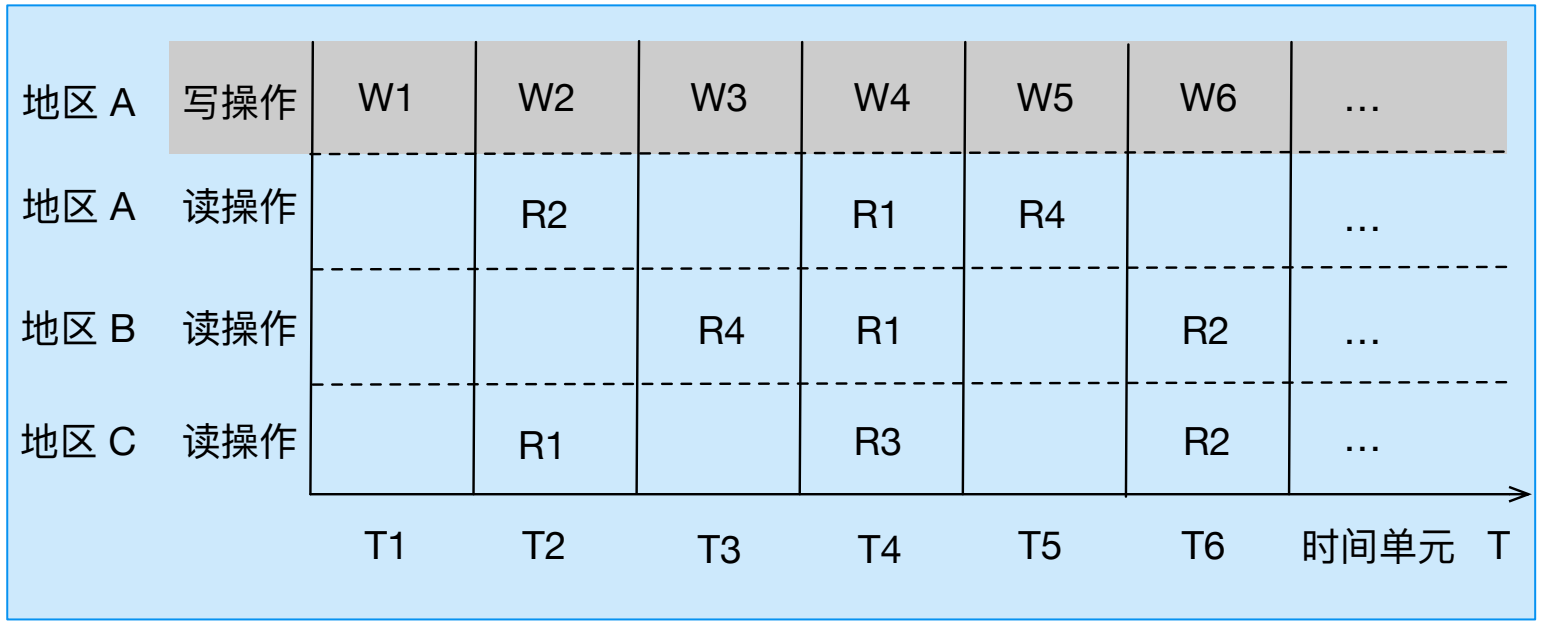
			W1	W2	W3	W4	W5	W6	…
	地区 A	写操作							
会话 1	地区 A	读操作	R1	R2	R3	R4	R5	R6	…
	地区 B	读操作	R1	R2	R3	R4	R5	R6	…
会话 2	地区 C	读操作			R1		R2	R3	…
			T1	T2	T3	T4	T5	T6	时间单元 T

前缀一致性：读写操作执行单调有序

		W1	W2	W3	W4	W5	W6	…
地区 A	写操作							
地区 A	读操作	R1	R2		R3	R4		…
地区 B	读操作	R1		R2	R3		R4	…
地区 C	读操作		R1		R2		R3	…
		T1	T2	T3	T4	T5	T6	时间单元 T

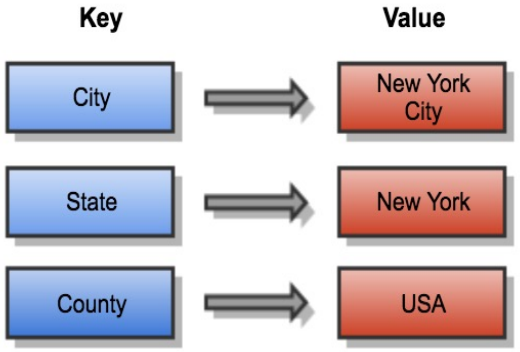
NoSQL一致性

最终一致性：最终完成写操作同步，实现与其他节点的数据一致性

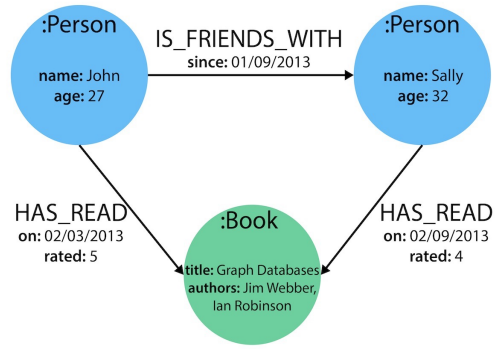


NoSQL数据库分类

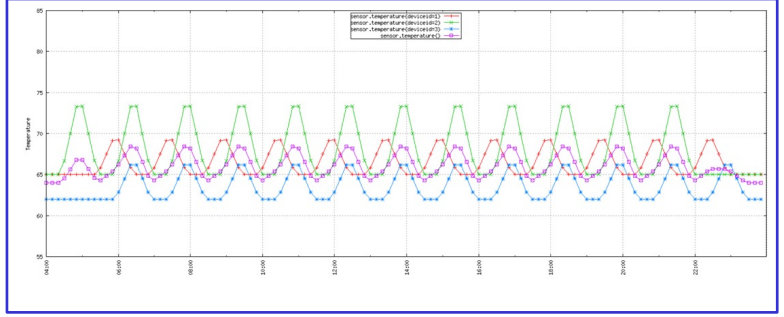
- 键值数据库
- 列族数据库
- 文档数据库
- 图数据库
- 时序数据库
- 多模数据库
- 向量数据库



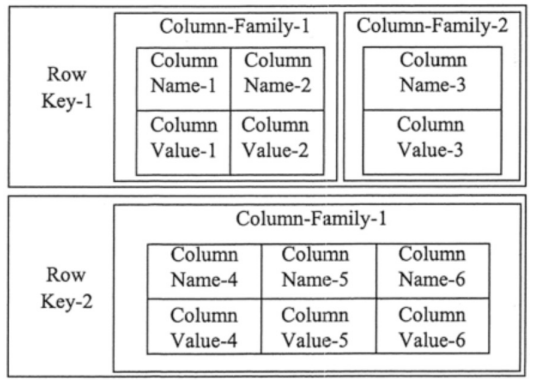
Graph



Time-series



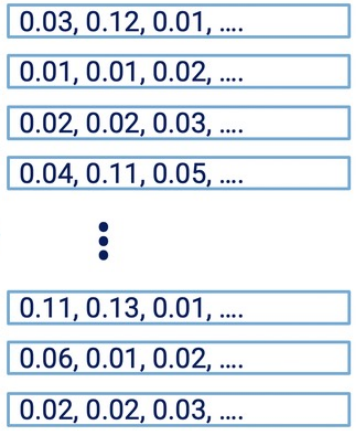
Column-family



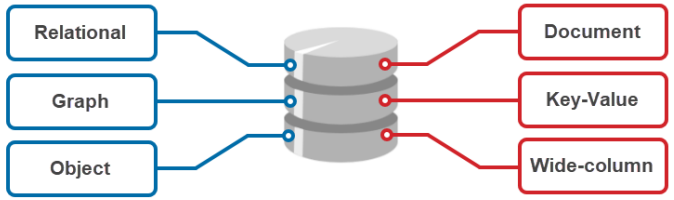
Document



vector

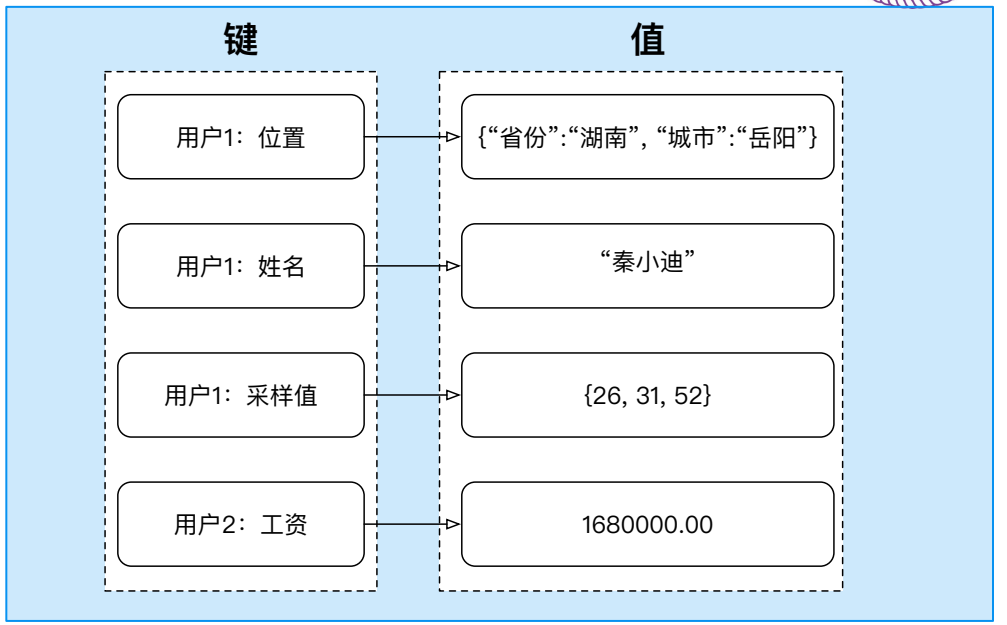


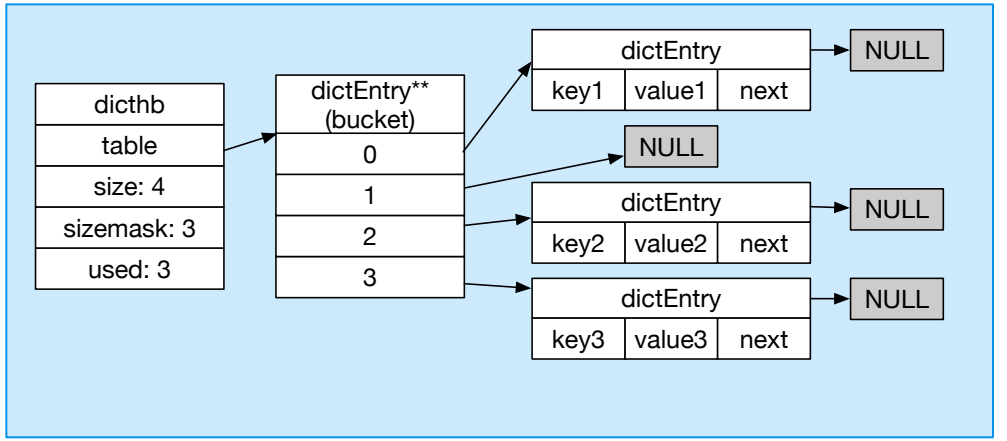
Multi-Model Database



键值数据库

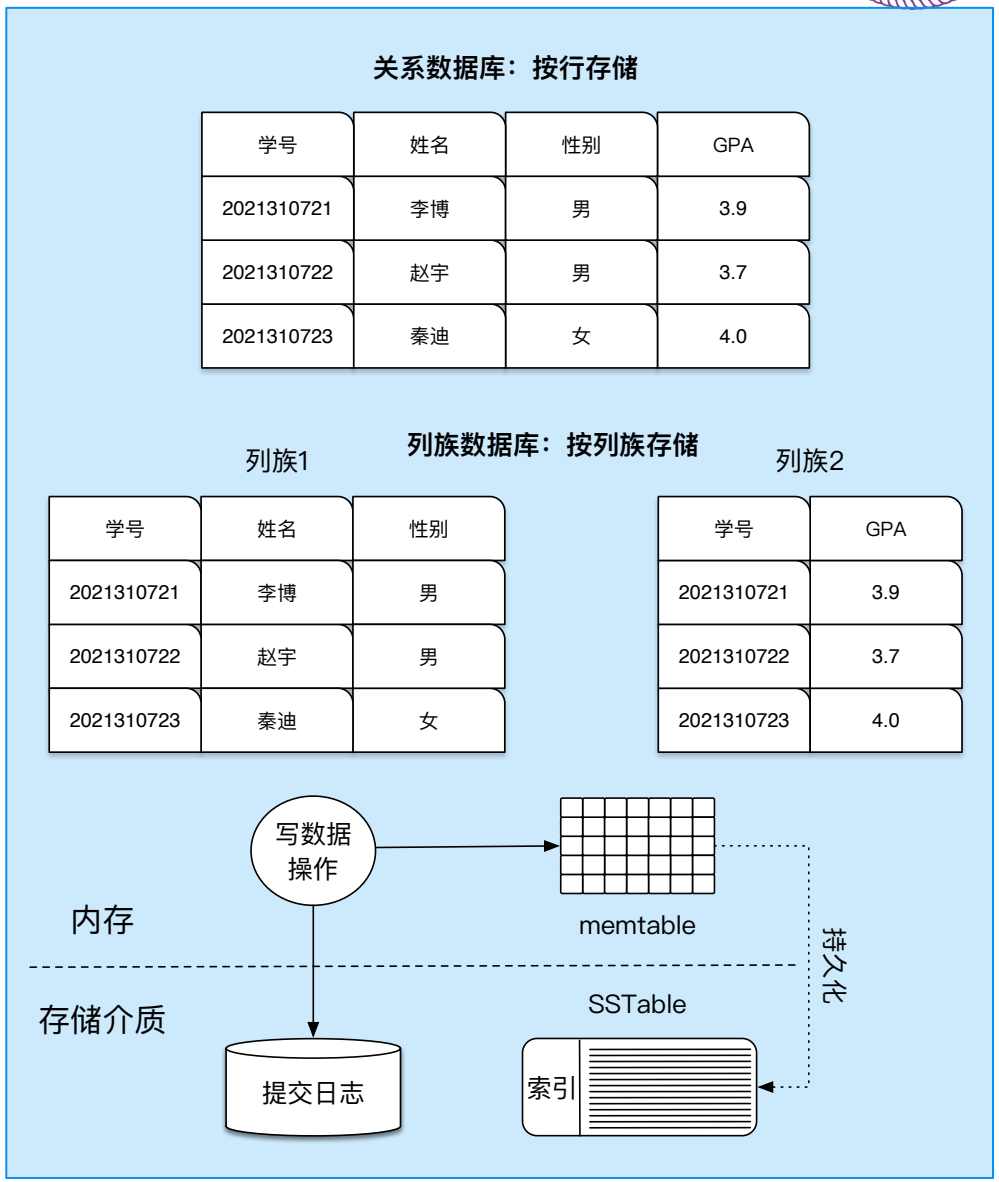
- **用键值对结构去存储数据的NoSQL数据库**
- **键和值可以是任何对象，如简单数值和包含多属性的复杂类对象**
- **主要支持键值对的查询、插入、更新和删除操作**
- **主要技术：**
  - 哈希表：组织数据，一致性哈希
  - 范围表： LSM-tree
- **应用场景：**
  - 缓存用户的会话数据、用户信息数据、在线购物平台的购物车数据
- **常见产品：Redis、RocksDB**





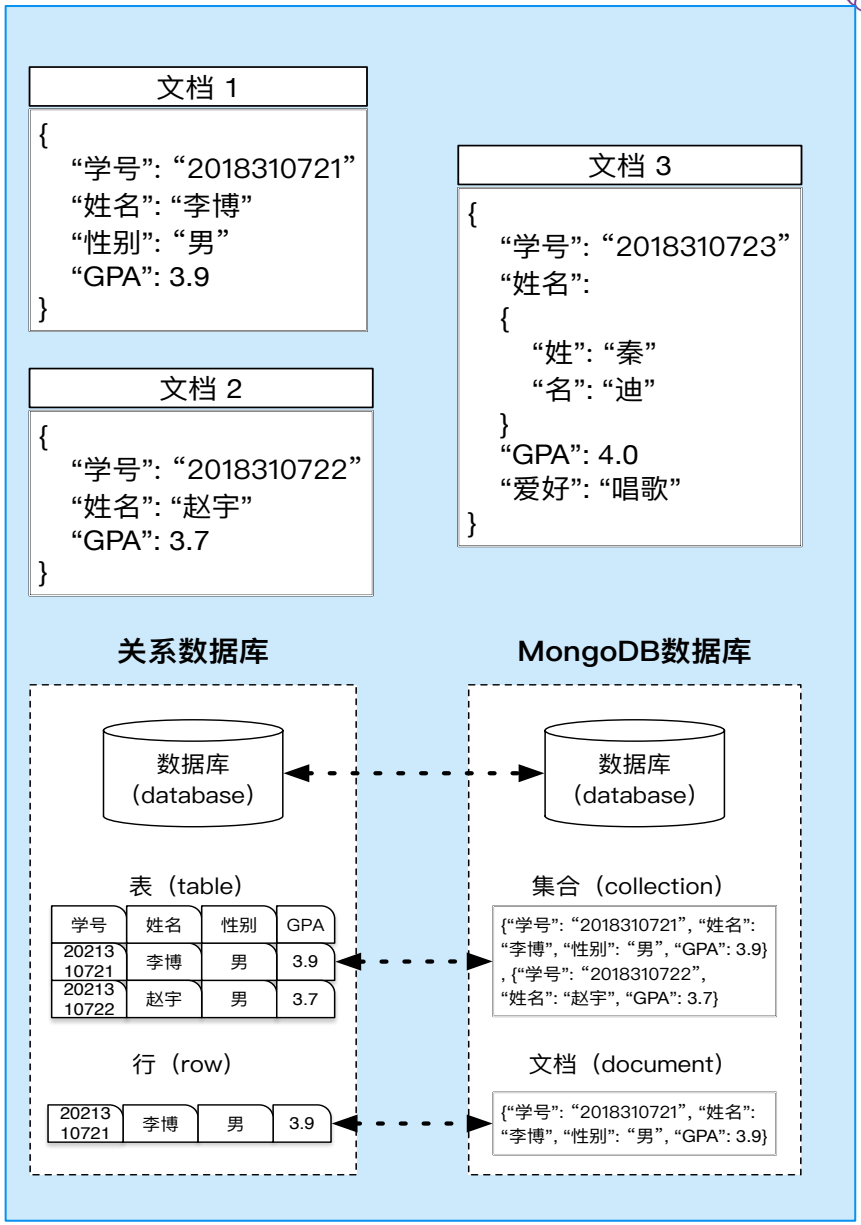
列族数据库

- **按查询需求组织数据列，由一到多个列族组成，每个列族可视为一个关系表**
- **避免了分析等查询加载不相关列数据，有效提升性能**
- **主要技术：**
  - 列族分离存储，列族内按行存储
  - LSM树：应对大规模写操作
- **应用场景：分析查询较多的应用场景**
  - 网络广告
  - 日志数据处理
- **常见产品：Cassandra**



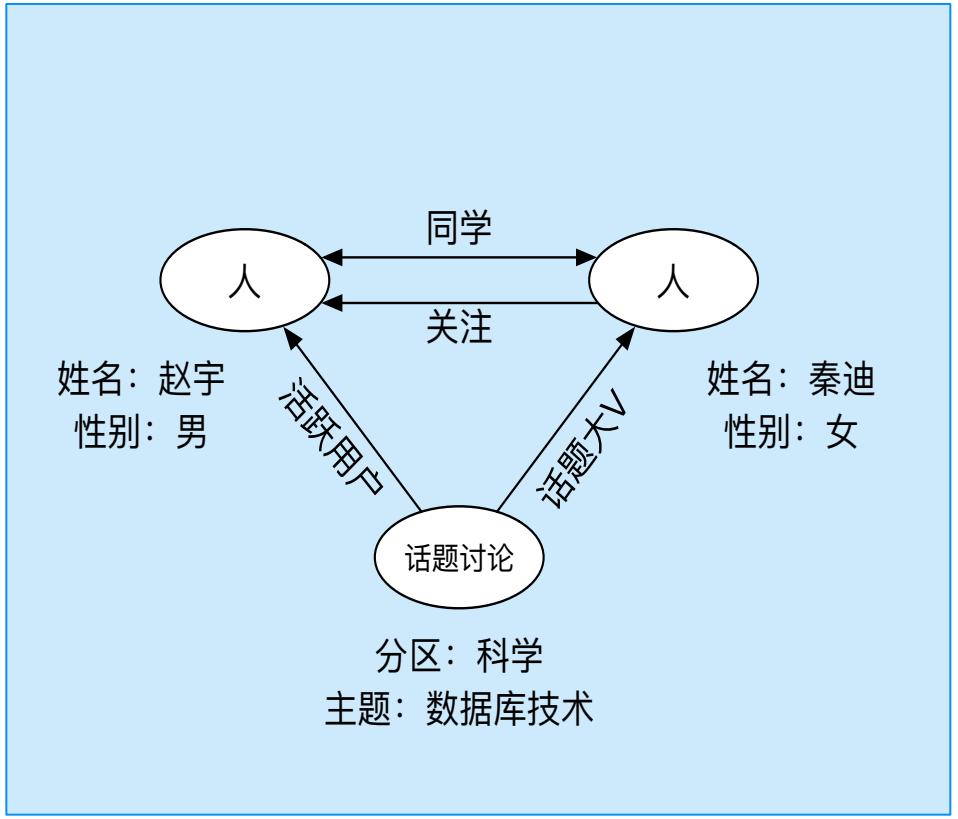
文档数据库

- 支持类JSON数据存储和查询，即层次化的键值对结构，每条数据都是可嵌套的键值对集合
- 数据库中的数据与应用程序中的数据保持相同结构，便于数据传递
- 数据不具有固定模式，摆脱了关系数据库中的数据模式限制
- 关键技术
  - 多粒度并发控制、数据压缩、分布式处理
- 应用场景：
  - 用户个人画像的存储、互联网应用实时数据存储、异构化业务数据存储、网站开发
- 常见产品：MongoDB



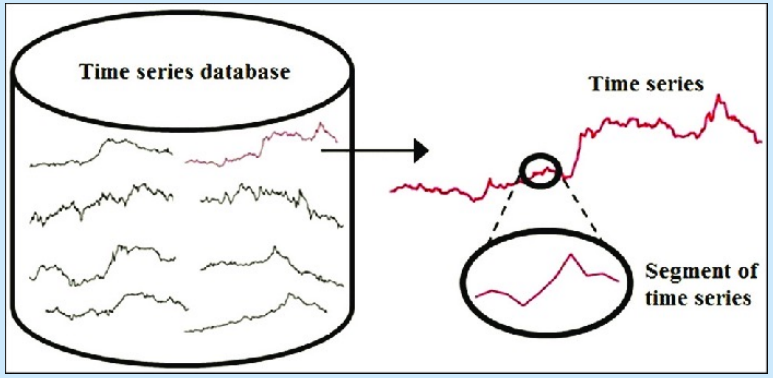
图数据库

- **存储图结构的数据，描述实体与实体间的关系**
  - 节点与边
- **使用图数据查询语言来为数据操作提供支持**
- **关键技术**
  - 图原生技术
    - 以点为中心、以边为中心的数据划分
    - 图遍历、图迭代、图学习
  - 基于关系的技术
- **应用场景：**
  - 社交网络
  - 欺诈分析
- **常见产品：Neo4j**



时序数据库

- **存储和查询以时间为索引的数据序列**
- **数据主要包含要测量的主体与对该主体的一到多个测量值，以及对应的时间戳。**
- **时间戳与测量值数据一般按列单独存储，且可以针对各自特性选择适当的编码方式来进行数据压缩。**
- **关键技术**
  - 数据压缩，数据编码
  - 数据快速入库
  - 数据多周期性分析
- **应用场景：**
  - 物联网设备传感器数据IOT
  - 监控设备和运维设备的数据APM
- **常见产品：InfluxDB**

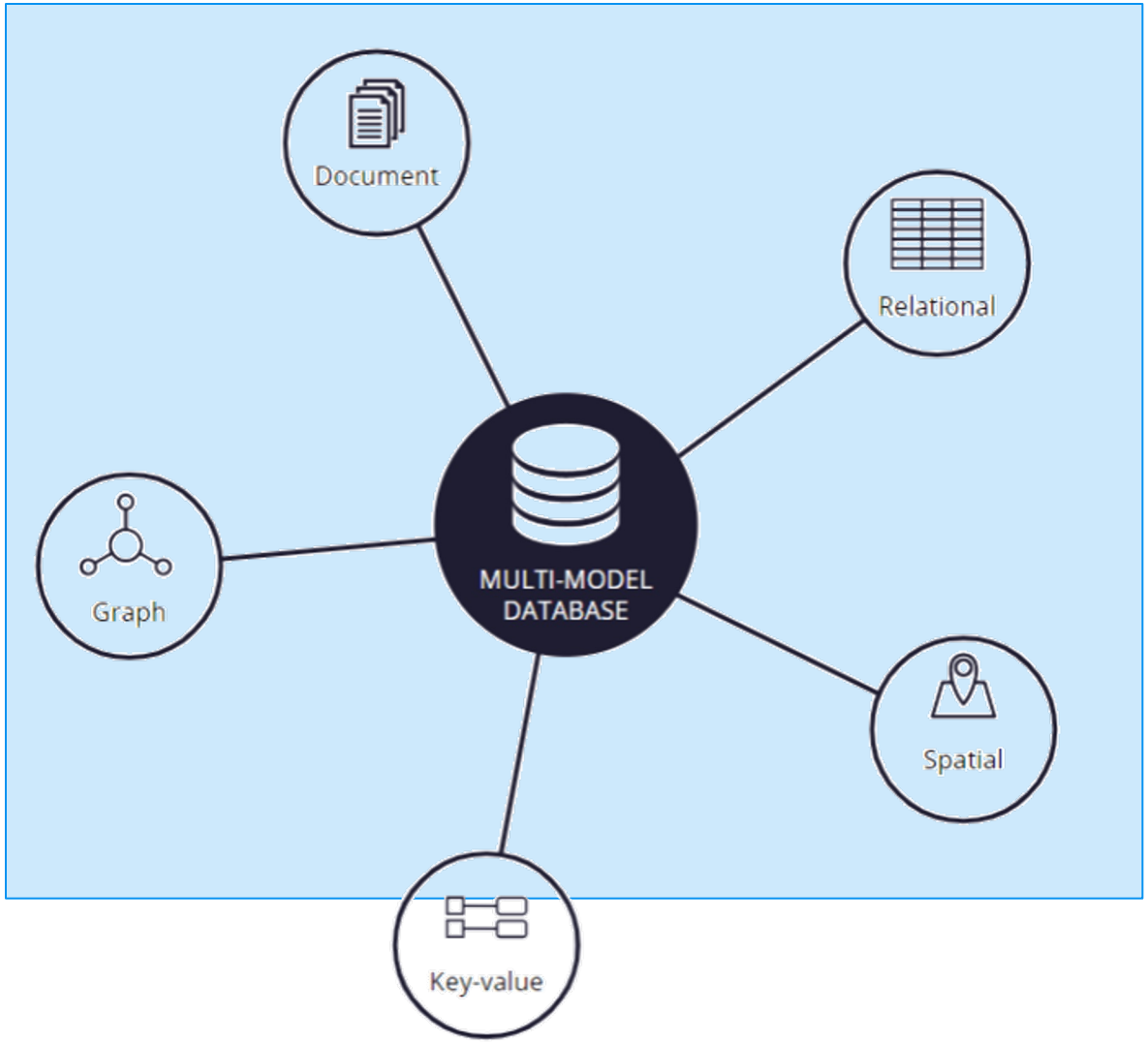


Timestamp	Cluster	Hostname	CPU	IOPS
2021-12-25T 19:19:00Z	Cluster-A	Host-a	99%	1000
2021-12-25T 19:19:10Z	Cluster-A	Host-b	50%	500
2021-12-25T 19:19:20Z	Cluster-A	Host-a	60%	600

时间戳　　主体属性（数据源）　　测量标准及测量值

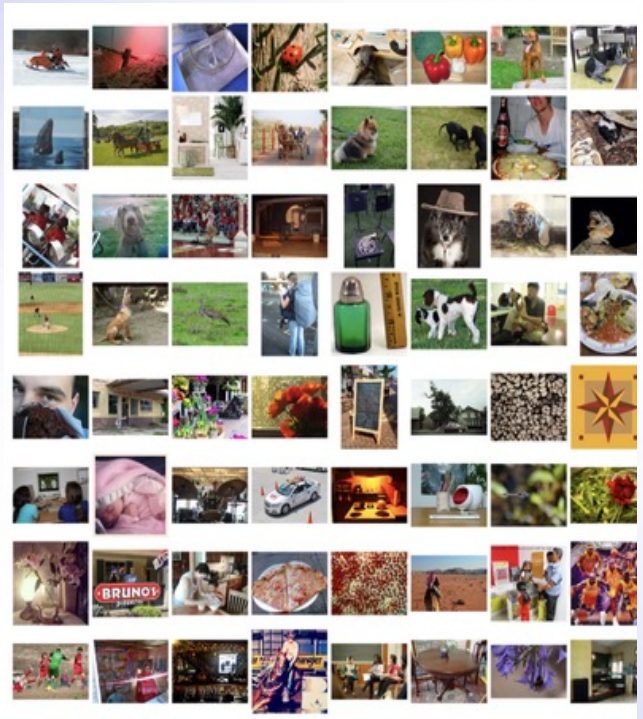
多模数据库

- **支持多种数据模型的数据库系统**
- **支持多种数据类型，并且所有数据模型可以使用统一的查询引擎或统一的查询接口**
- **关键技术**
  - 可调一致性
  - 统一存储
  - 统一计算模型
- **应用场景：**
  - 管理多种异构数据
  - 电子商务、消息管理服务
- **常见产品：Cosmos**



向量数据库

非结构化数据



向量（embedding）

0.03, 0.12, 0.01, ...

0.01, 0.01, 0.02, ...

0.02, 0.02, 0.03, ...

0.04, 0.11, 0.05, ...

⋮

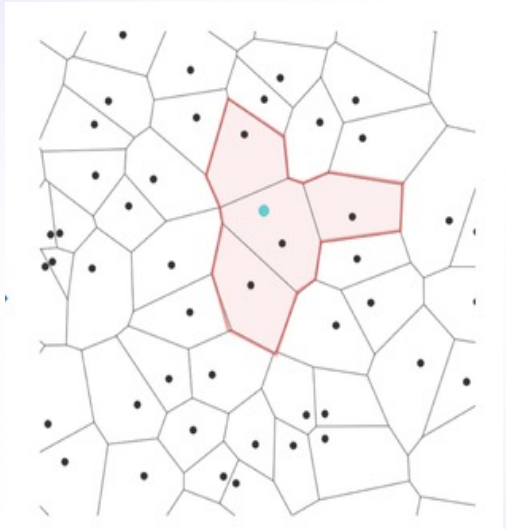
0.11, 0.13, 0.01, ...

0.06, 0.01, 0.02, ...

0.02, 0.02, 0.03, ...

向量空间/向量搜索

KNN：精准匹配TOP K
ANN：模糊匹配TOP K



距离远近决定向量的相似度
挑战：维度灾难

向量数据库

原生向量数据库

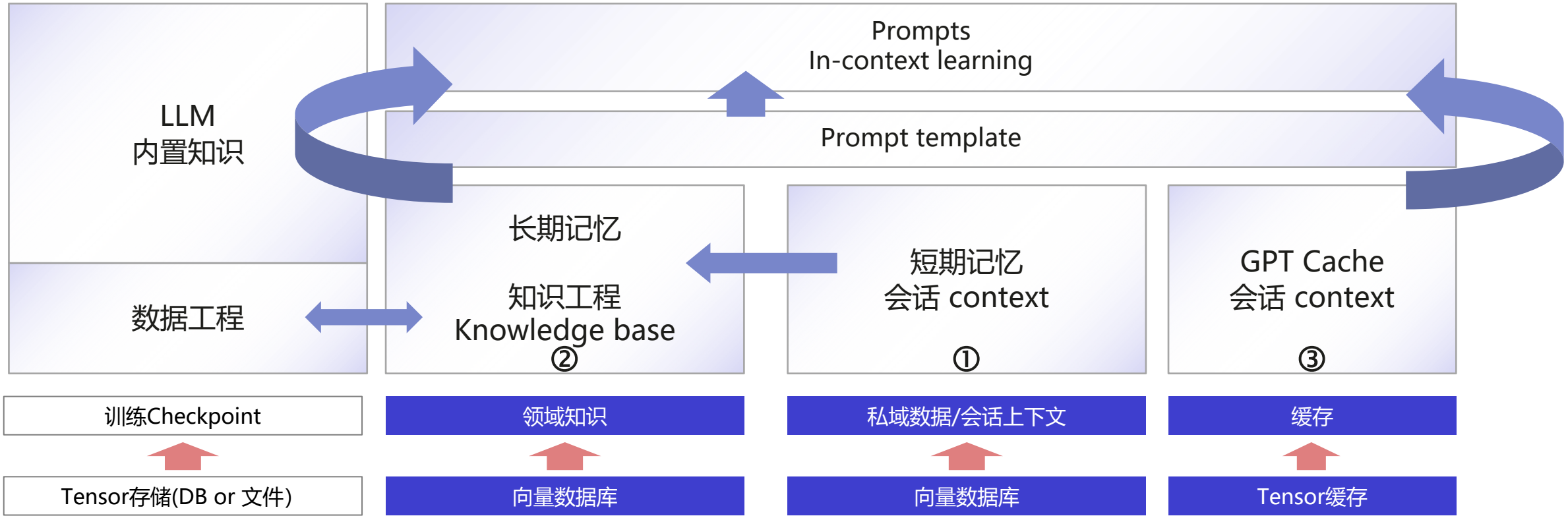
接入层	向量检索	标量查询	...
计算层	分割	向量化	精排	聚合
存储层	对象存储	KV存储	向量索引

支持向量类型关系型数据库

插件化引入
扩展向量类型
支持向量计算

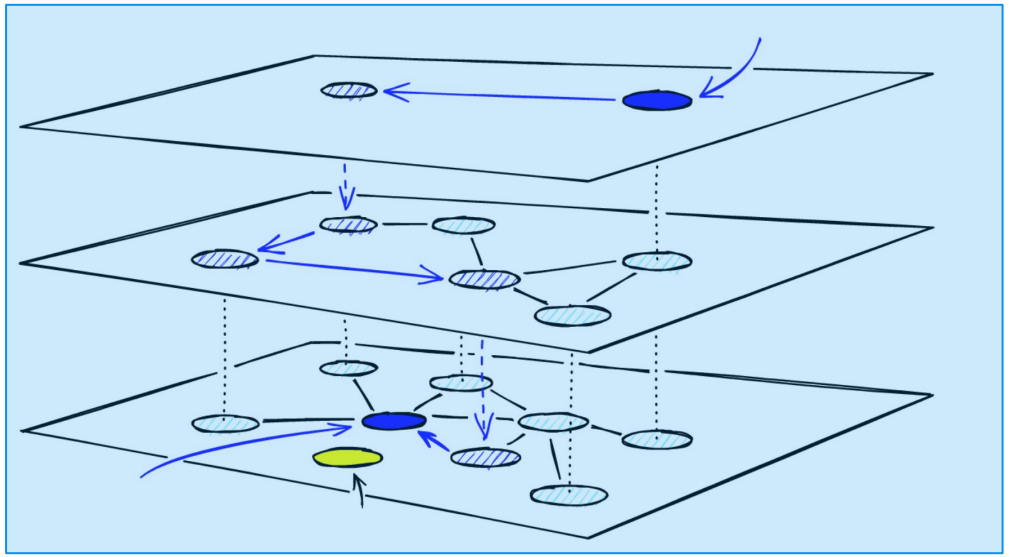
接入层	SQL	KV	向量查询
计算层	优化器	执行器	向量算子
存储	缓存	堆表	向量索引

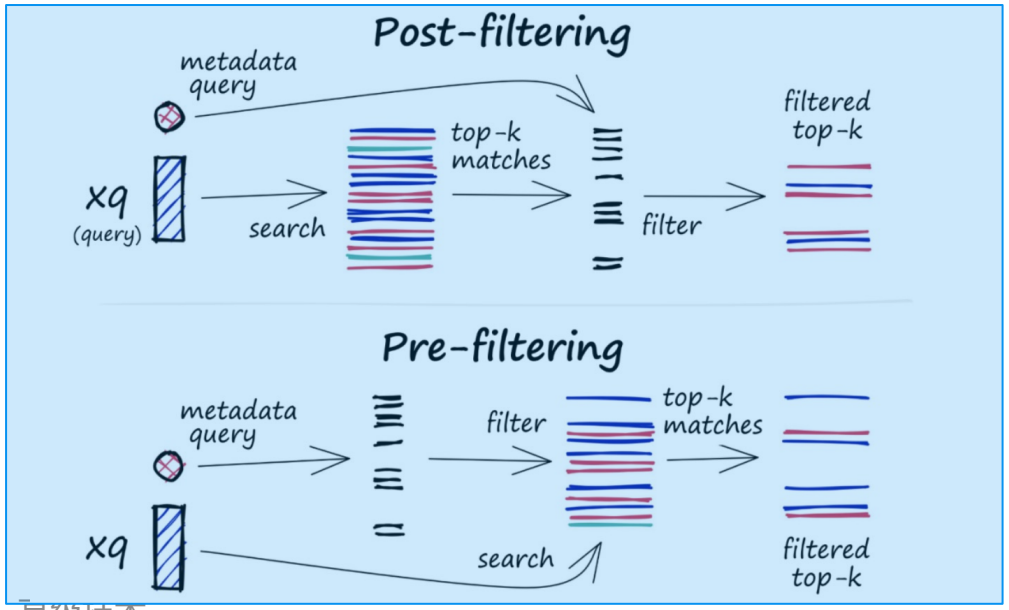
向量数据库


Prompts
In-context learning
Prompt template
LLM
内置知识
数据工程
长期记忆
知识工程
Knowledge base
②
短期记忆
会话 context
①
GPT Cache
会话 context
③
训练Checkpoint
领域知识
私域数据/会话上下文
缓存
Tensor存储(DB or 文件)
向量数据库
向量数据库
Tensor缓存
Batch
万亿向量
知识存储
千亿向量
降低幻觉
私域隐私
百亿向量
存储换算力

向量数据库挑战

- **目标：大并发、低时延、高召回**
- **向量索引**
  - 索引更新、可扩展性、持久化与恢复
- **多属性+向量查询**
- **多向量查询**
- **向量变换与降维**
- **软硬结合的向量查询**





NoSQL数据库对比

类型	数据库产品	集群模式	事务特性	应用场景
键值	Redis	主从模式	ACID	缓存、队列和过滤等
列族	Cassandra	主主模式	最终一致性	实时分析，零售和信息收发
文档	MongoDB	主从模式	ACID	网站开发，游戏
图	Neo4J	主从模式	ACID	推荐业务和反欺诈保护
时序	InfluxDB	主主模式	最终一致性	传感器数据，应用状态监控
向量	Milvus	主主模式	最终一致性	大模型，图片搜索
多模	Cosmos	主从模式	可调一致性	应用开发，数据存储

08 AI与数据库

Survey: http://dbgroup.cs.tsinghua.edu.cn/ligl/papers/aidb.pdf

Tutorial: http://dbgroup.cs.tsinghua.edu.cn/ligl/papers/aidb-slides.pdf

AI原生数据库

- **传统数据库上云，面临诸多挑战**



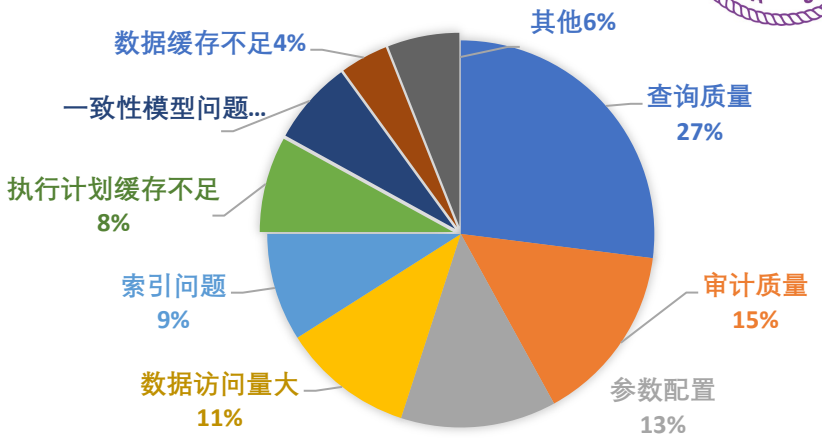
"one-size-not-fits-all"

需要根据应用自适应
调整数据库配置



"challenges in big data"

数据难划分，查询难优化
实时数仓 (速度；一致性)



"issues in db maintenance"

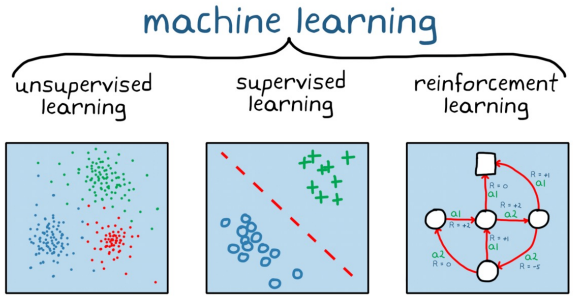
人力或简单的运维工具
难以支撑云上庞杂的运维任务

- **AI在数据、算法、新硬件的驱动下取得突破进展**

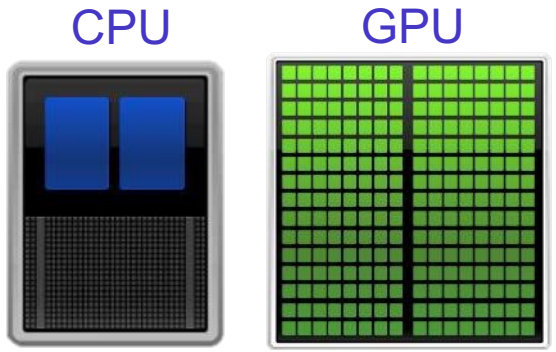
TPCH　JOB　TPCDS

TPCC　Sysbench　SQLSmith

训练数据管理，提效率



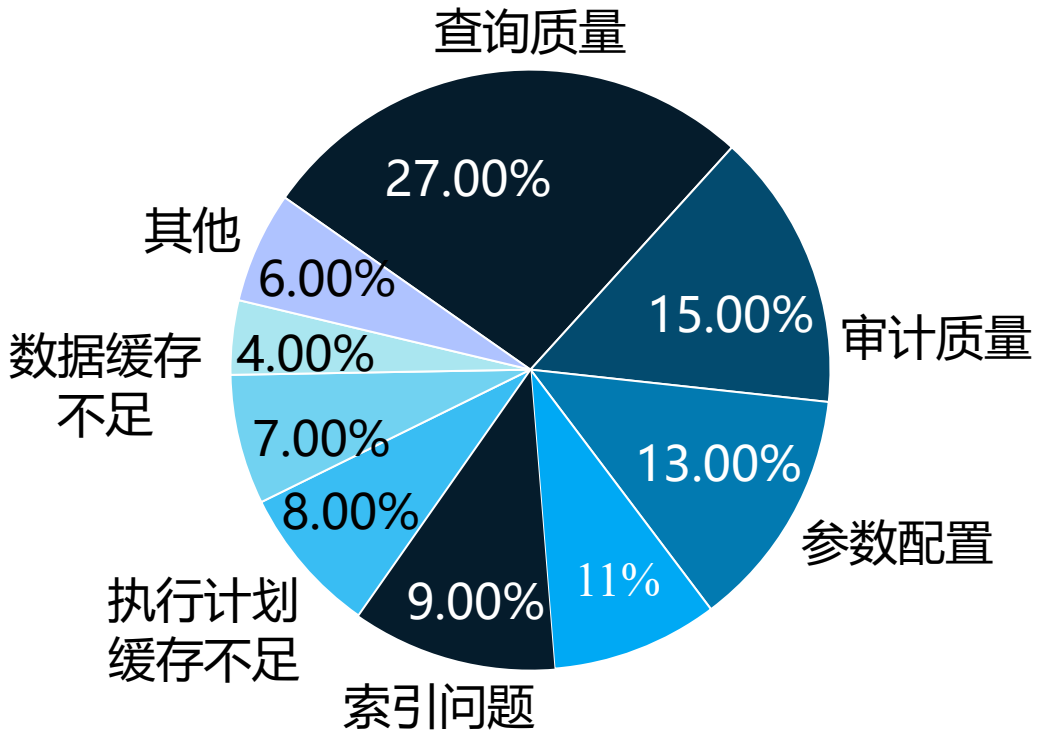
多种机器学习算法，库原生



异构硬件，加速执行

AI4DB

数据库性能瓶颈的成因复杂，性能优化是一项高门槛、高投入的工作，资深DBA也需要花费大量时间理解当前情况进行手动调整



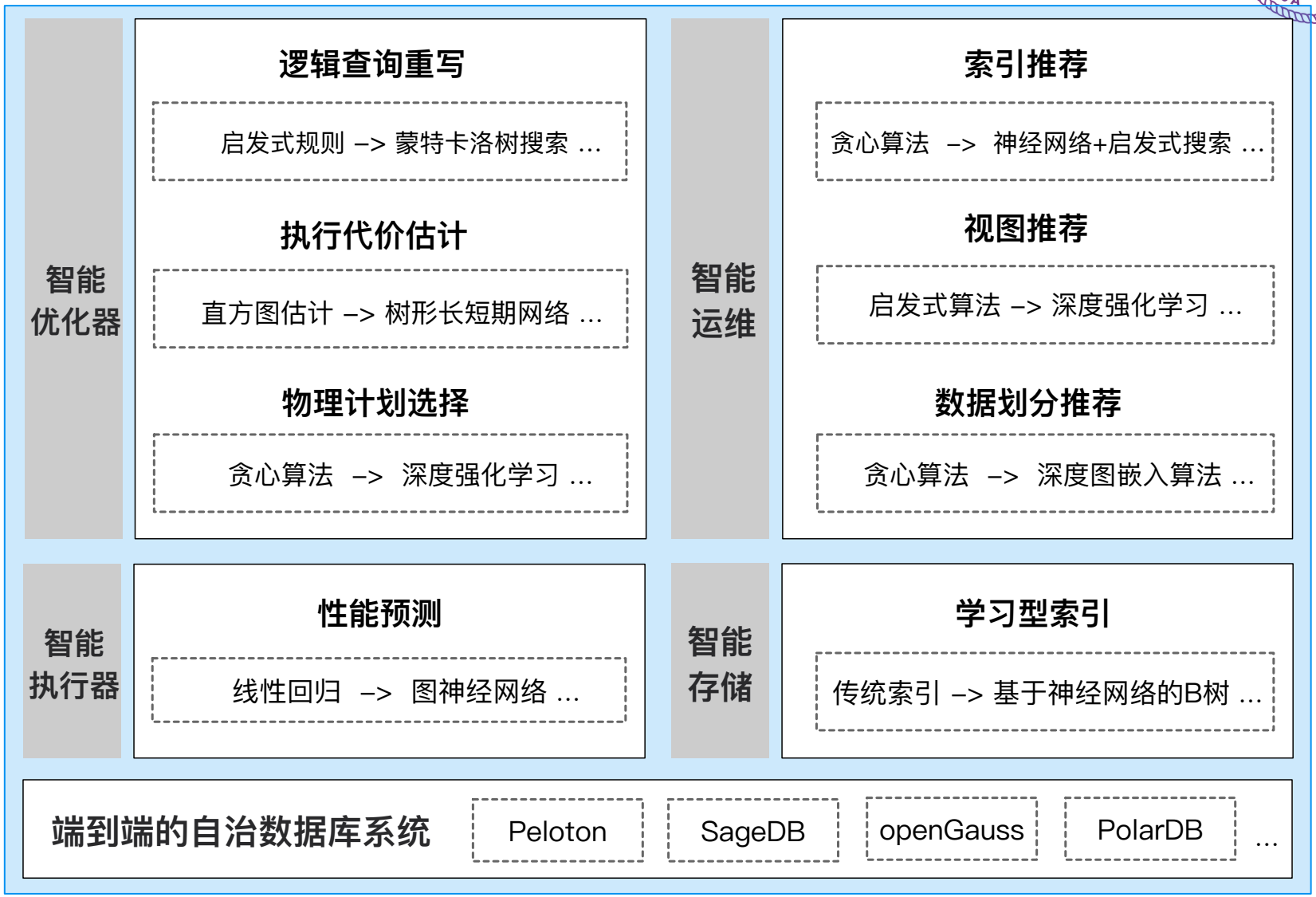
数据库性能问题成因调查

解决方案

- 粗放管理，不断扩容高性能服务器
- 高价雇佣资深DBA，利用经验手动控制
- 特定场景的启发式算法优化
- 基于AI和专家经验的数据库自治服务

AI4DB

- **ABO优化器**
  - 逻辑查询重写
  - 执行代价估计
  - 物理计划选择
- **智能执行器**
  - 并行性能预测
- **智能数据存储**
  - 学习型索引
- **智能运维优化**
  - 智能索引推荐
  - 智能视图推荐
  - 智能数据划分
- **自治数据库系统**



AI4DB – NP优化问题

- **在线NP**
  - 物理计划生成：快速找到代价较小的物理计划
  - 查询重写：快速找到等价的逻辑计划
- **离线NP**
  - 索引选择：选择一些列建立索引
    - 索引代价收益
    - 索引选择
  - 视图选择：选择一些频繁子查询建立视图
    - 视图代价收益
    - 视图选择
    - 基于视图的查询重写
    - 视图更新

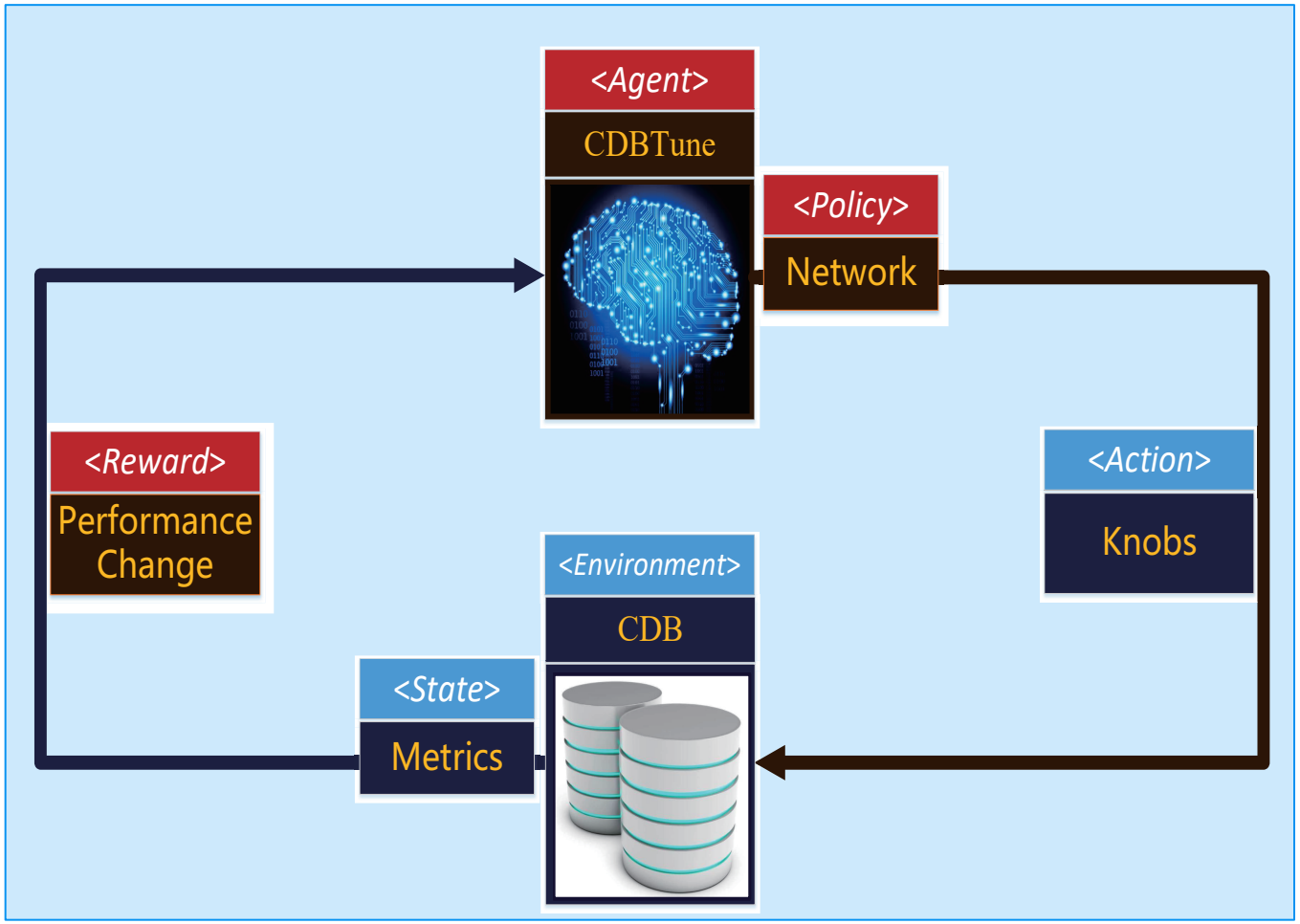
AI4DB – NP优化问题

方法	优化目标	损失函数 / 反馈函数	训练数据	适应性(负载级别)
传统梯度计算	性能	加权系数等于目标的平均值	高	--
深度学习	资源利用率	$L:(e,\lambda)\rightarrow\begin{cases} l(e)I(e\geq 0) \\ \lambda l(e)I(e<0) \end{cases}$ **l(e):** 平方误差; **λ:** 控制过拟合对训练结果的影响	高	--
迁移学习	资源利用率	损失函数：模型预测错误导致误排配置的比例	高	✓
深度强化学习	性能	$r=\begin{cases} ((1+\Delta_{t\rightarrow 0})^2-1)\lvert 1+\Delta_{t\rightarrow t-1}\rvert, \Delta_{t\rightarrow 0}>0 \\ -((1-\Delta_{t\rightarrow 0})^2-1)\lvert 1-\Delta_{t\rightarrow t-1}\rvert, \Delta_{t\rightarrow 0}\leqslant 0 \end{cases}$ r: 反馈值; $\Delta T_{t\rightarrow t\text{-}1}$ / $\Delta T_{t\rightarrow 0}$: 性能变化	低 (强化学习不需要提前准备数据)	✓ (基于大量SQL语料的预训练)

AI4DB – NP优化问题

- **数据库配置空间巨大、不同配置影响查询表现和资源利用**
  - 将配置调优转化为强化学习问题

强化学习	CDBTune
Agent	调优系统
Environment	数据库实例
State	内核监控指标
Reward	性能变化
Action	配置信息
Policy	深度神经网络



AI4DB - 回归问题

- **基数估计：估计查询的结果数**
  - 轻量、高效、准确率高
- **代价估计：估计查询的代价**
  - 轻量、高效、准确率高
- **索引收益估计：估计索引的收益**
  - 建索引之后对查询性能的提升
  - 索引对数据更新的性能影响
- **视图收益估计：估计视图的收益**
  - 建物化视图之后对查询性能的提升
  - 物化视图的更新对性能的影响

AI4DB – 回归问题

方法	任务	特征空间	特征类型	训练数据
传统机器学习 (如 tree-ensemble、高斯分布、自回归算法等)	代价估计, 索引收益估计, 视图收益估计	较小	连续值	较多
Sum-Product网络	代价估计	较小	离散值	较少
深度学习	代价估计, 收益估计, 执行时间预测	较大	连续值	较多
深度图嵌入	收益估计, 执行时间预测	较大	连续值	较多

AI4DB – 回归问题

➢ **表中存在不同的数据分布，一些分布之间相互独立**

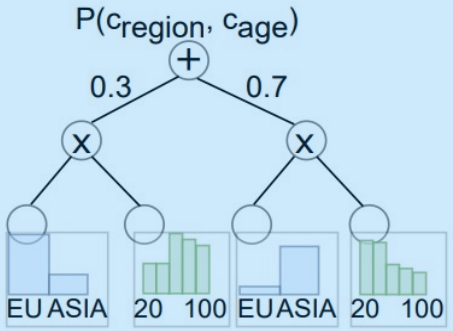
- 对于每个数据表，根据分布特征拆分成接近相互独立的数据块
- **Sum-Product网络：**Sum拟合过滤操作，Product拟合连接操作
- **扩展功能：**支持Null值表示；叶节点利用分段线性函数拟合数据分布；多列联合分布

c_id	c_age	c_region
1	80	EU
2	70	EU
3	60	ASIA
4	20	EU
…	…	…
998	20	ASIA
998	25	EU
999	30	ASIA
1000	70	ASIA

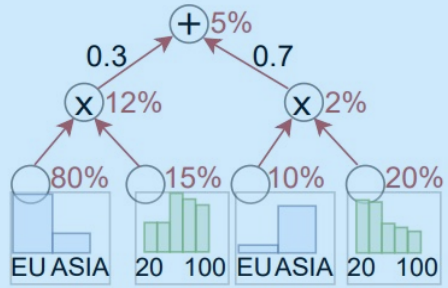
(a) Example Table

c_age	c_region
80	EU
70	EU
60	ASIA
20	EU
…	…
…	…
20	ASIA
25	EU
30	ASIA
70	ASIA

(b) Learning with Row/Column Clustering



(c) Resulting SPN



(d) Probability of European Customers younger than 30

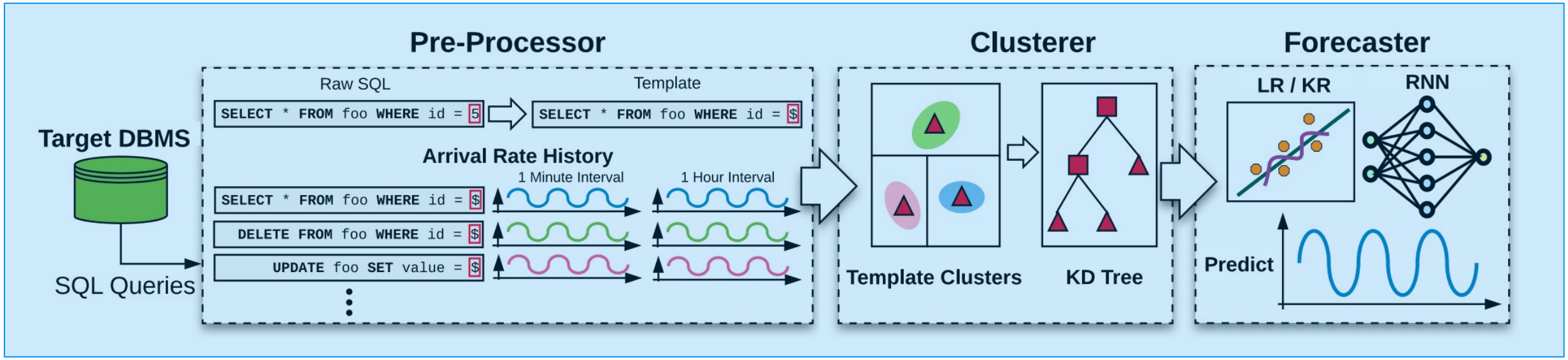
AI4DB – 时序预测问题

- **查询预测**

方法	任务	目标	训练数据
聚类算法	负载趋势预测	预测准确度	较多
强化学习	负载调度	性能	--

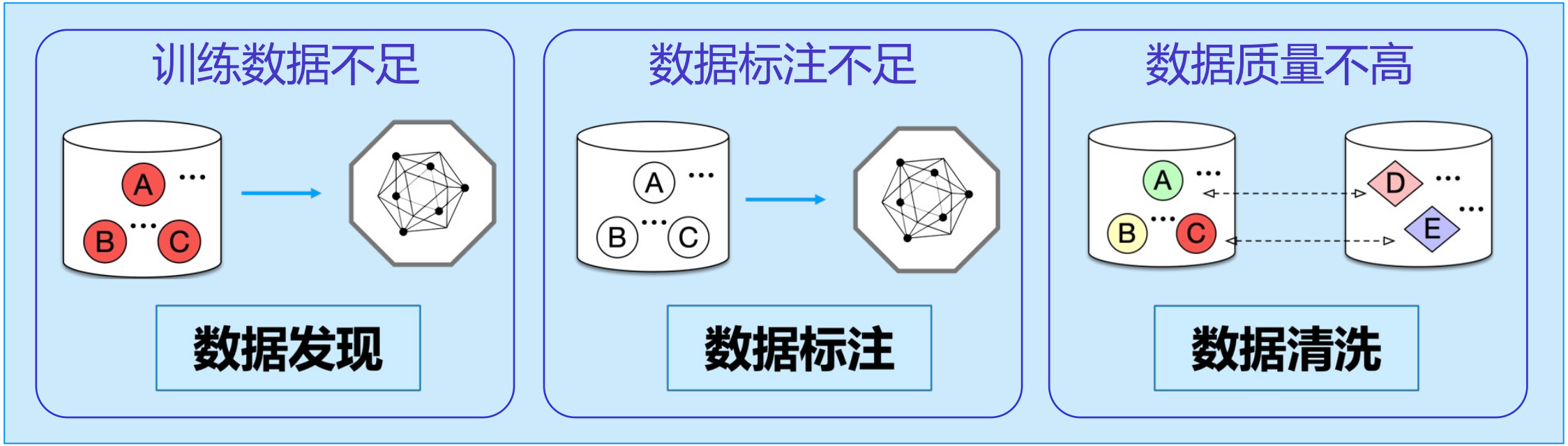
AI4DB – 时序预测问题

- **不同的负载可能有不同的趋势变化曲线（如查询数目）**
  - **预处理器**：识别工作负载中的查询模板和到达率
  - **聚类器**：将具有相似到达率模式的模板组合在一起
  - **预测器**：利用线性回归/RNN等模型预测每个聚类中的查询到达率



DB4AI

- **人工智能：算力、算法、数据**
  - 著名数据科学社区Anaconda指出，**数据准备**占数据从业者**最多时间**
  - 据统计，全球数据软件市场规模已达32.9亿美元，并快速增长
- **以数据为中心的下一代AI技术（data-centric AI）**
  - 系统地构建用于构建人工智能系统的数据基座



DB4AI

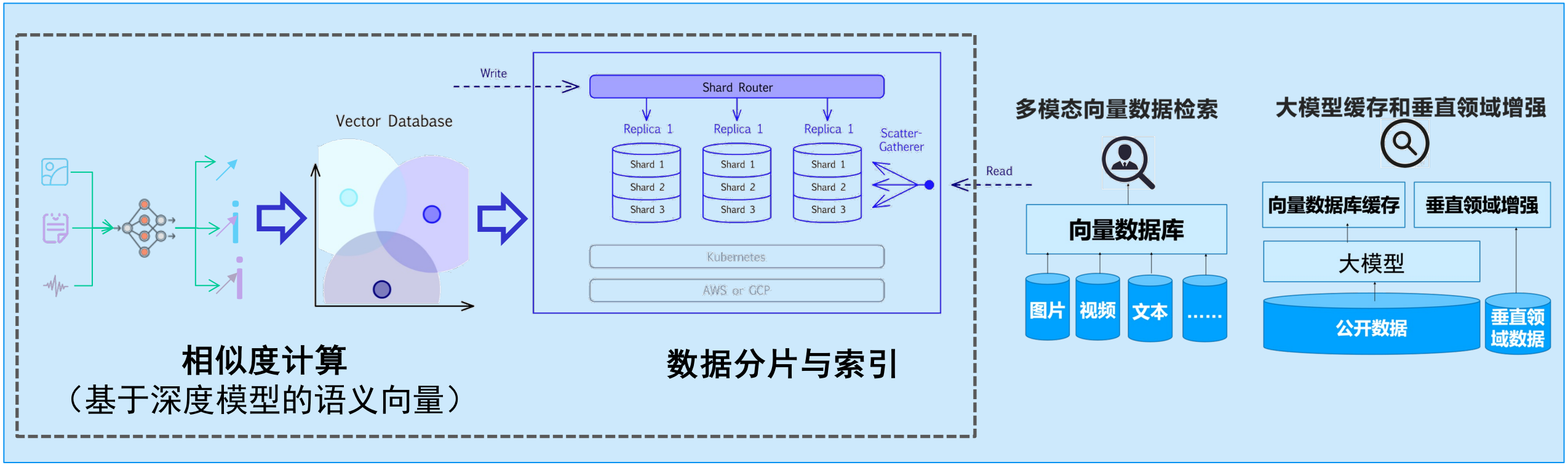
- **声明式的语言模型**
- **面向AI的数据治理**
  - 数据发现和清洗
  - 数据血缘和标注
- **面向AI的模型推理**
  - AI算子支持
  - AI算子选择
  - 执行加速算法
- **面向AI的模型训练**
  - 特征选择与计算
  - 模型类型选择
  - 模型分布式训练
  - 模型统一管理



DB4AI

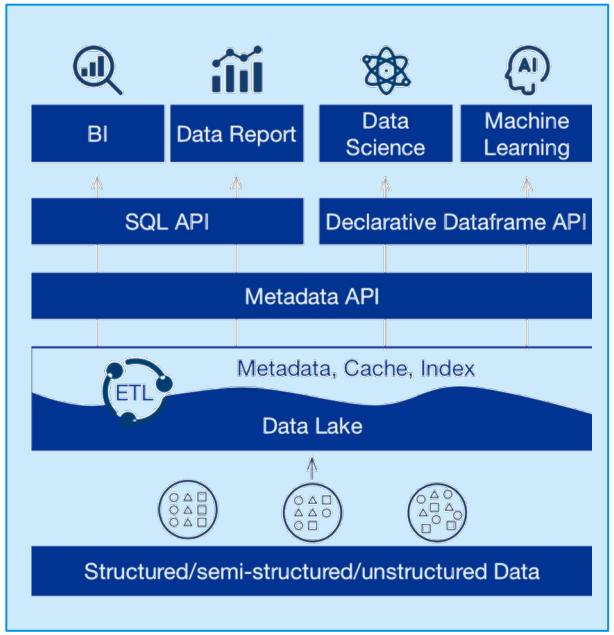
➢ 向量数据库赋能大模型应用

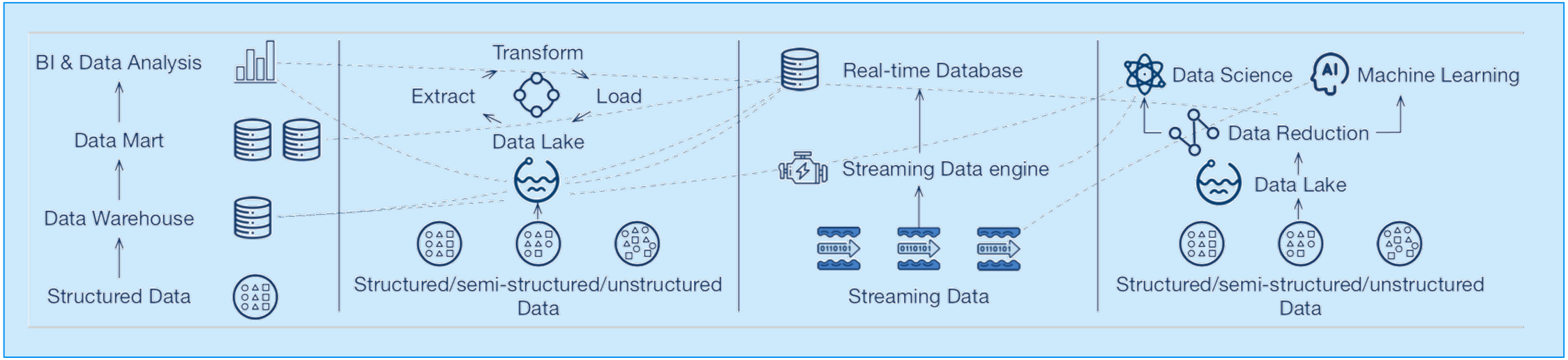
- 用户查询缓存，降低用户时延，**提升吞吐**
- 利用隐层向量近似搜索**相似历史检索结果**
- 提升大模型业务实时性与吞吐，**降低大模型服务开销**



数智融合

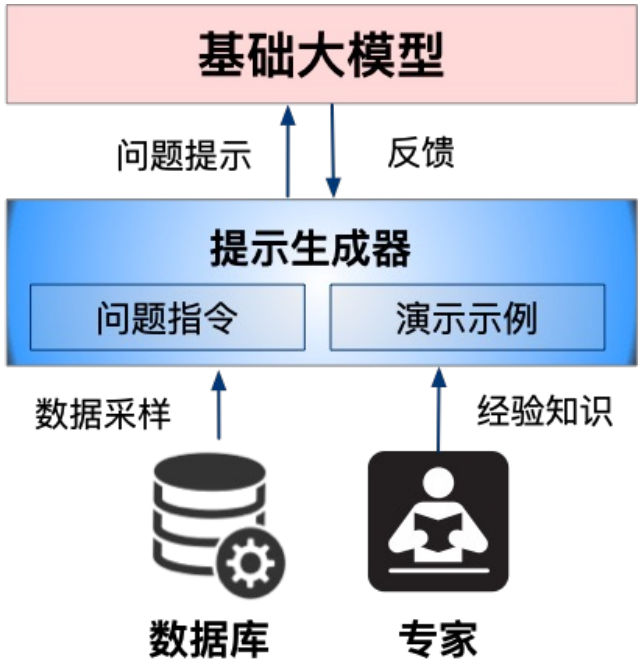
- **优势**
  - 在线推理: T+1 → T+0
  - 数据安全: ETL → In-DB
  - 资源利用率: 冗余 → 一份数据
- **统一数据、统一查询、数智融合**

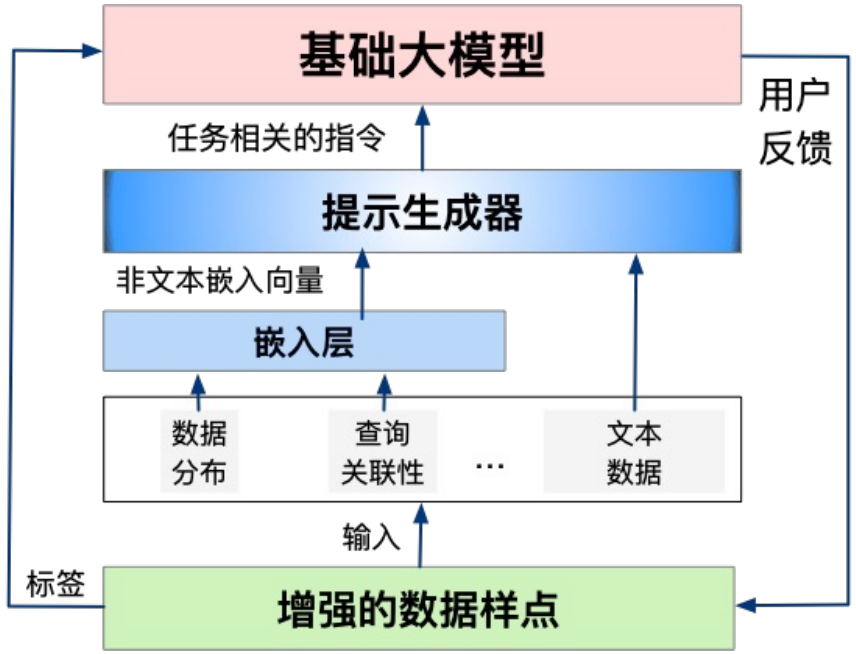


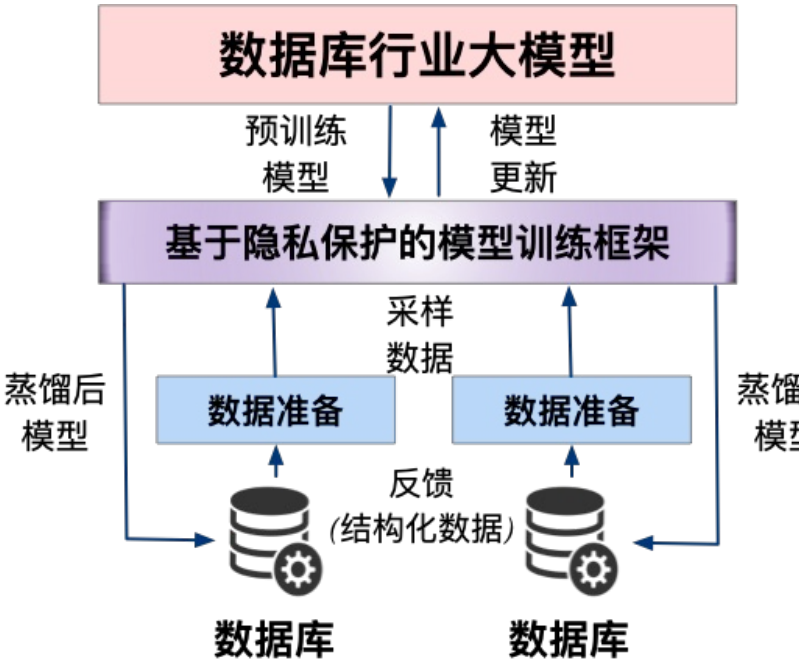


大模型与数据库

- 提升数据库性能：LLM as Optimizer：端到端 SQL→执行计划
- 提升数据库易维护性：LLM as DBA： 运维客服, DBA助手, 自治平台
- 提升大模型的数据增强：向量数据库、大模型数据库







高级数据库技术小结

大数据量超高并发	分布式数据库	分布式事务2PC+Paxos、分布式查询优化
海量数据分析决策	OLAP数据库	列存、向量化、MPP、物化视图
交易分析混合负载	HTAP数据库	行列混存、列存选择、行列混合优化器
实时交易超低时延	内存数据库	行级组织、内存索引、内存并发控制
云上弹性伸缩调度	云原生数据库	计算/缓存/存储解耦、日志即数据、多租
新型硬件算力发挥	新硬件数据库	NUMA并发控制、GPU并行处理、NVM组织、RDMA分布协同、CXL共享缓存、FGPA压缩
海量非结构化数据	NoSQL数据库	CAP、BASE、LSM-tree、多模处理
智能时代数智融合	AI原生数据库	ABO(基数/重写/计划)、AI4DB、DB4AI