奈学教育

MySQL InnoDB存储引擎实现原理深入剖析与应用实践（上）

主讲人：陈东

2020.07.27

目录

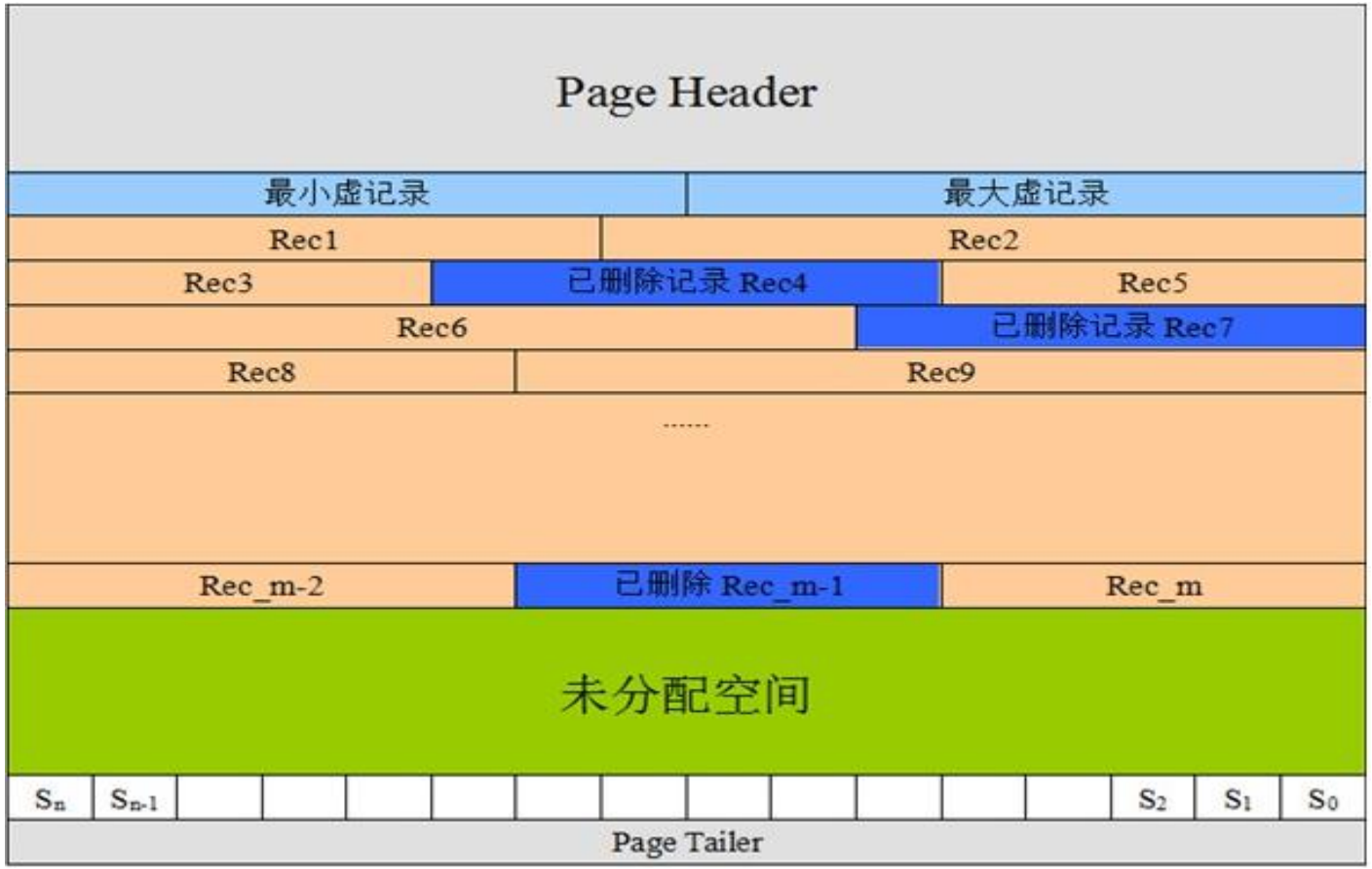
NX 奈学教育

NX 奈学教育

01.MySQL InnoDB存储原理深入剖析与技术分析

MySQL记录存储

- 页头
- 虚记录
- 记录堆
- 自由空间链表
- 未分配空间
- Slot区
- 页尾



MySQL记录存储

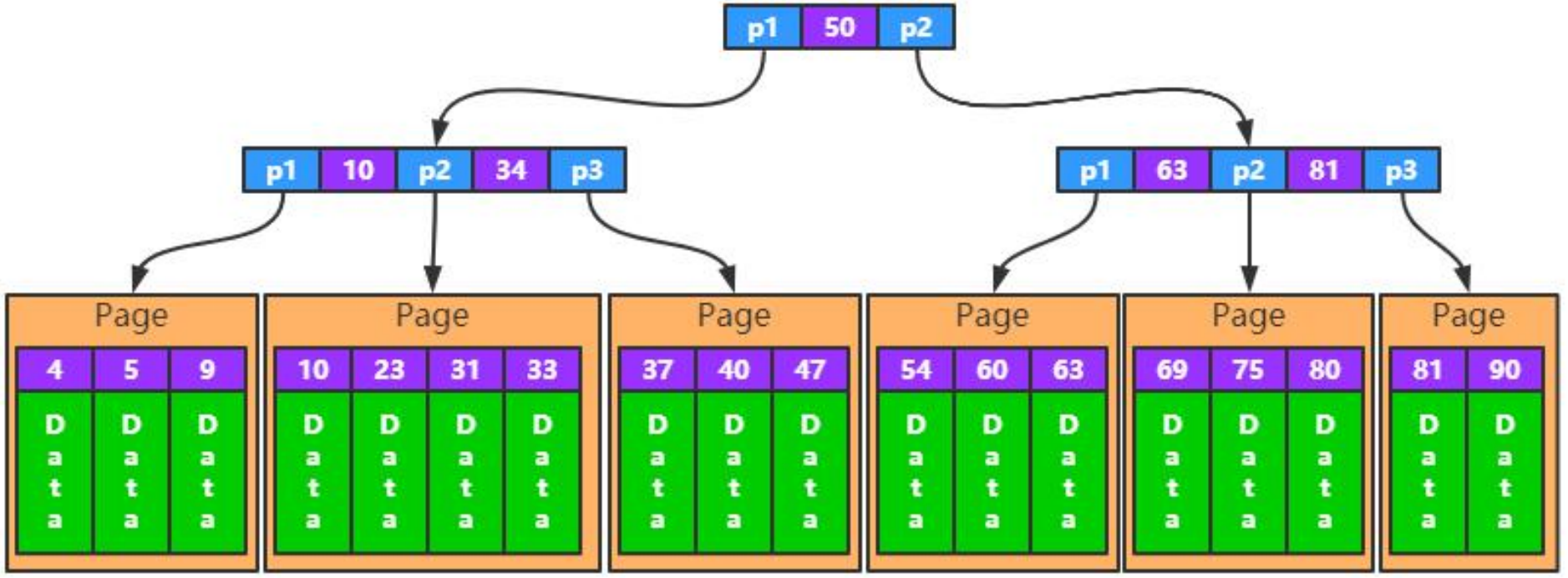
- 页头
  - 记录页面的控制信息，共占56字节，包括页的左右兄弟页面指针、页面空间使用情况等。
- 虚记录
  - 最大虚记录：比页内最大主键还大
  - 最小虚记录：比页内最小主键还小
- 记录堆
  - 行记录存储区，分为有效记录和已删除记录两种
- 自由空间链表
  - 已删除记录组成的链表

MySQL记录存储

- 未分配空间
  - 页面未使用的存储空间；
- 页尾
  - 页面最后部分，占8个字节，主要存储页面的校验信息；

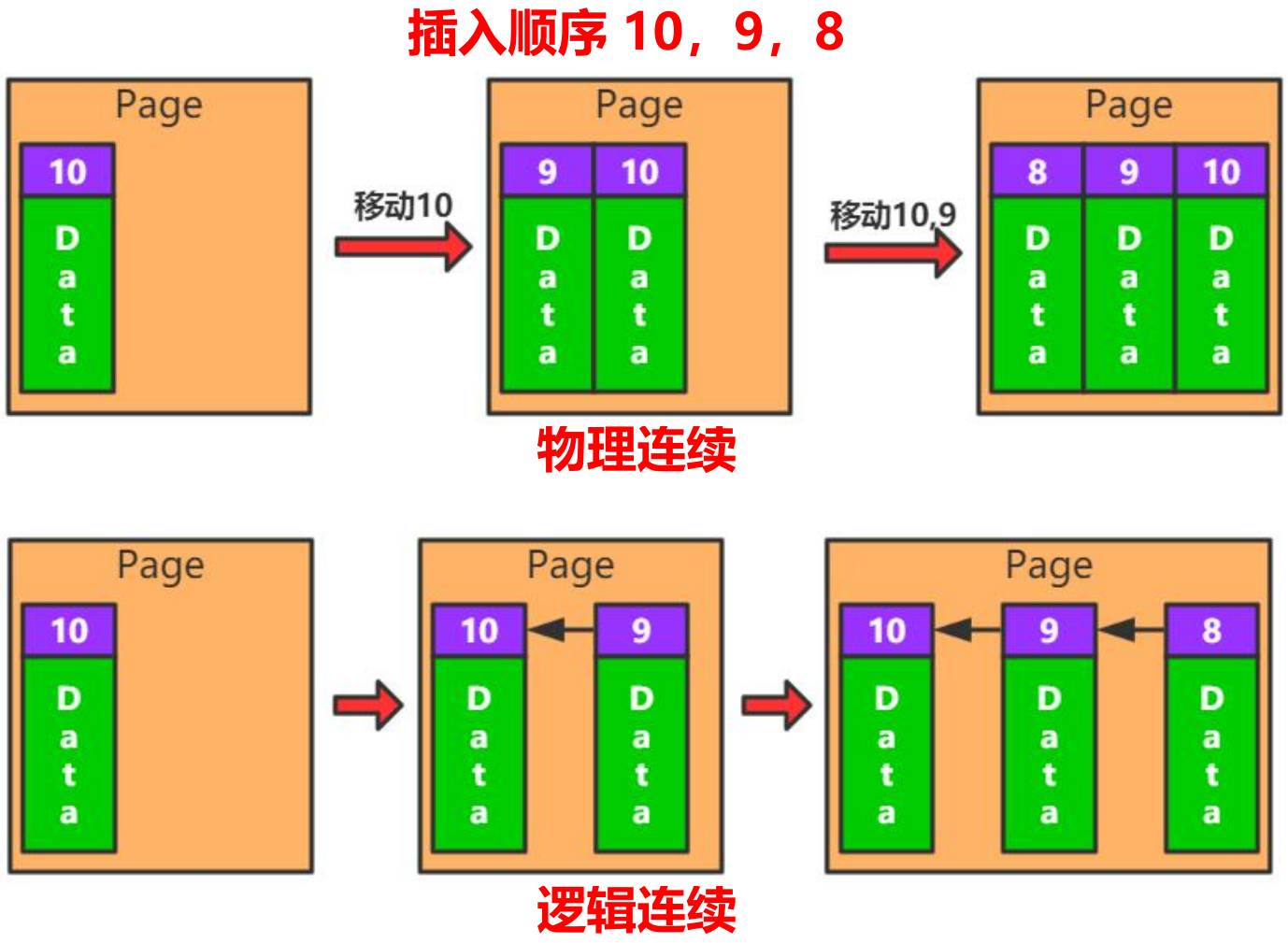
页内记录维护

- 顺序保证
- 插入策略
- 页内查询



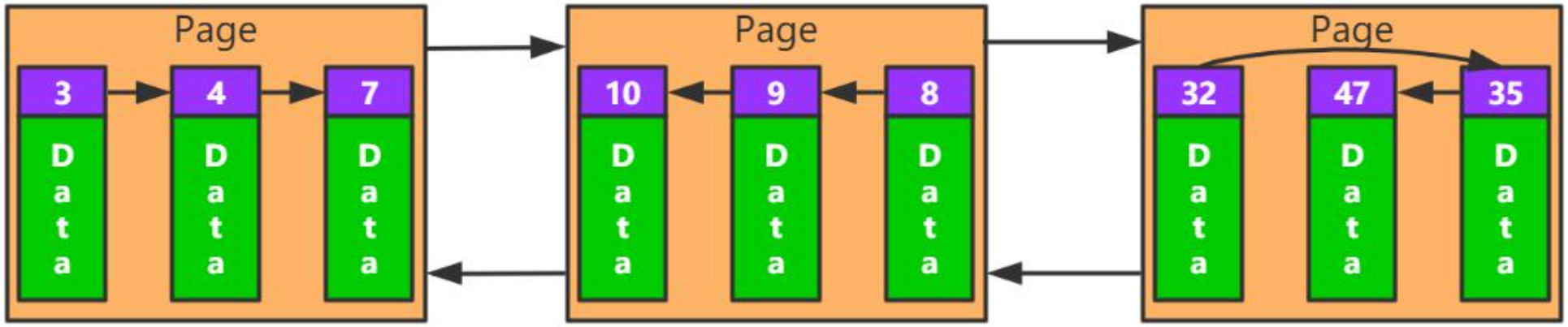
业内记录维护

- **顺序保证**
  - 物理连续
  - 逻辑连续
- 插入策略
- 页内查询



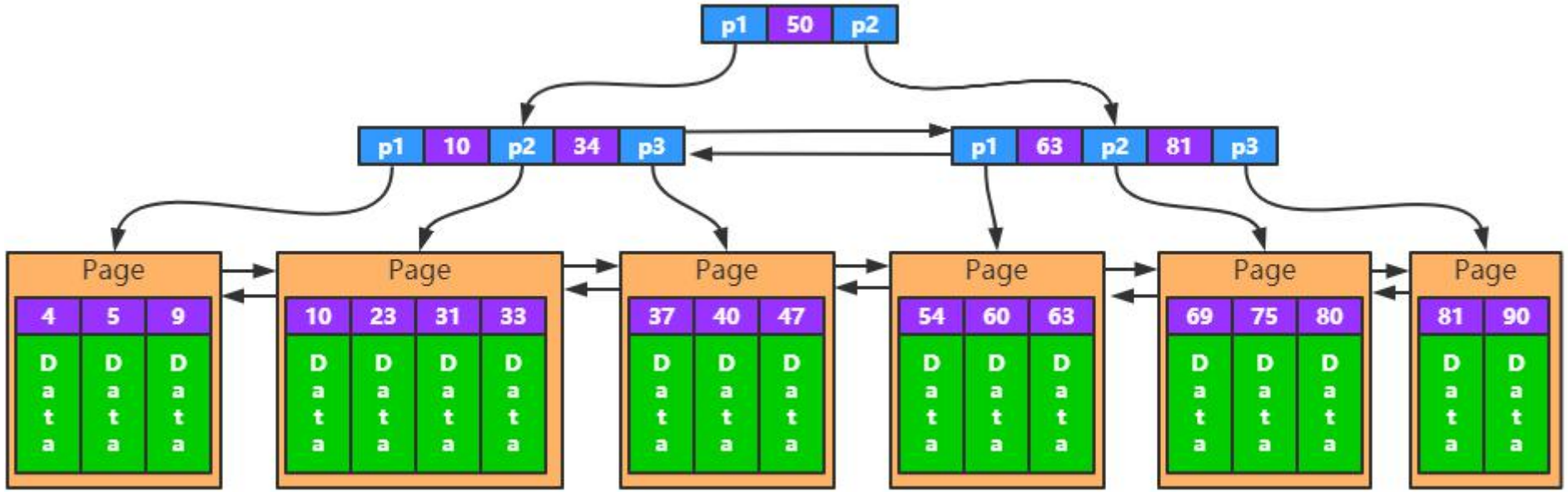
业内记录维护

- **顺序保证**
  - 物理连续
  - 逻辑连续
- 插入策略
- 页内查询



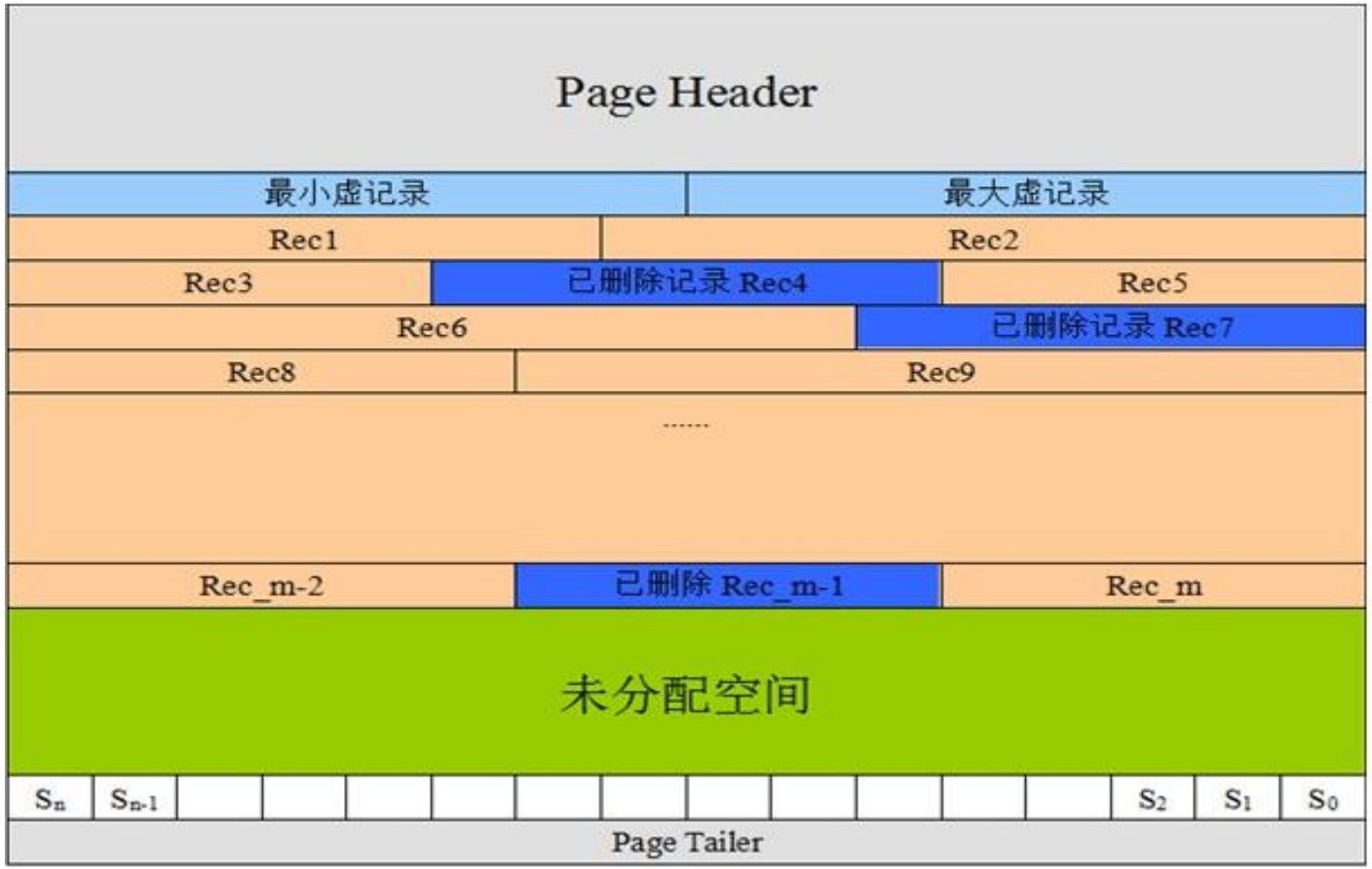
业内记录维护

- **顺序保证**
  - 物理连续
  - 逻辑连续
- 插入策略
- 页内查询



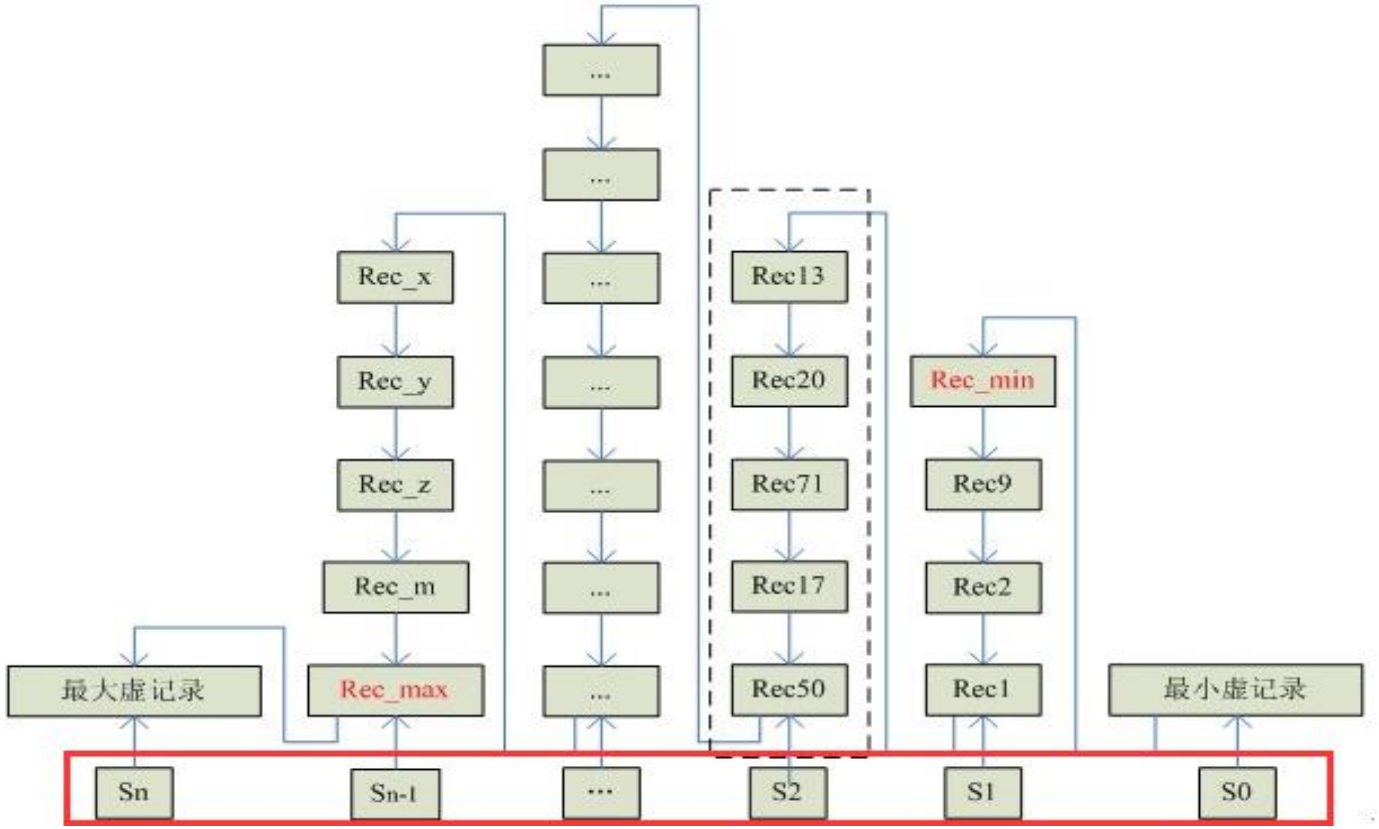
业内记录维护

- 顺序保证
- **插入策略**
  - 自由空间链表
  - 未使用空间
- 页内查询



业内记录维护

- 顺序保证
- 插入策略
- **页内查询**
  - 遍历
  - 二分查找



NX 奈学教育

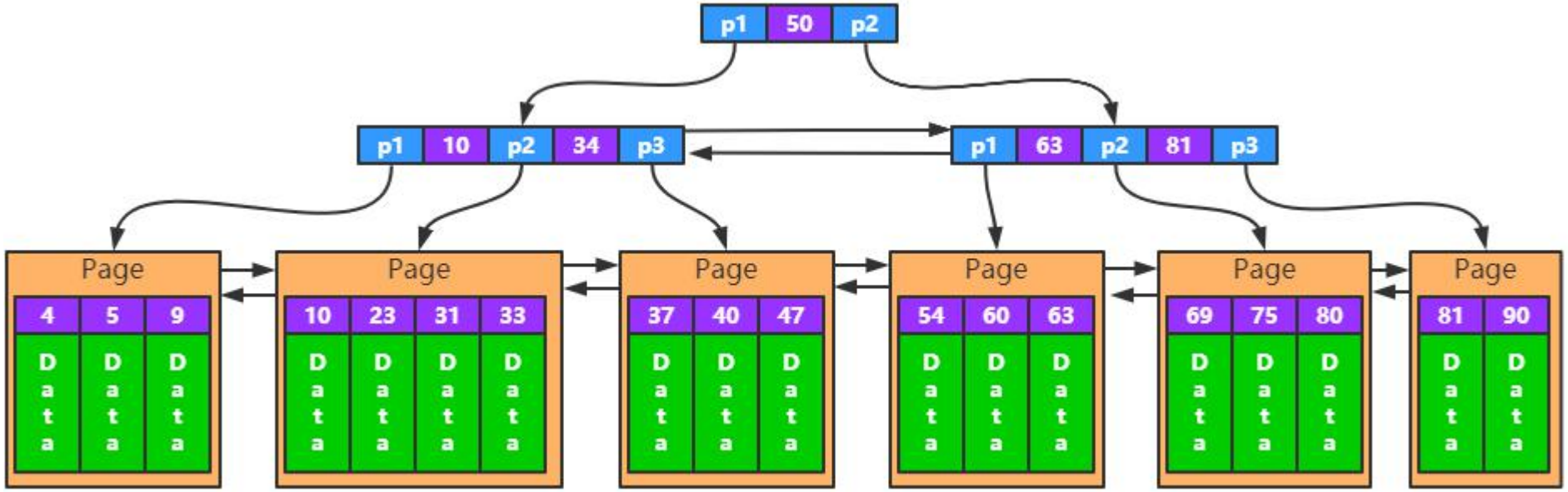
01.MySQL InnoDB索引实现原理及使用优化分析

索引原理分析

- 聚簇索引
- 二级索引
- 联合索引

索引原理分析

- **聚簇索引**
- 二级索引
- 联合索引



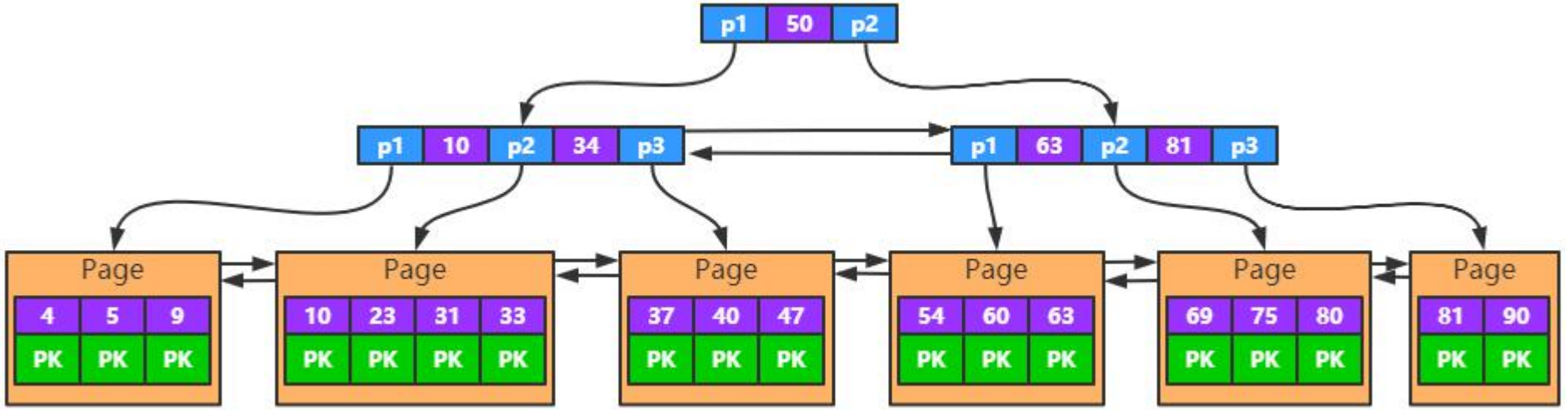
聚簇索引

- 数据存储在主键索引中
- 数据按主键顺序存储

自增主键 VS 随机主键

索引原理分析

- 聚簇索引
- **二级索引**
- 联合索引

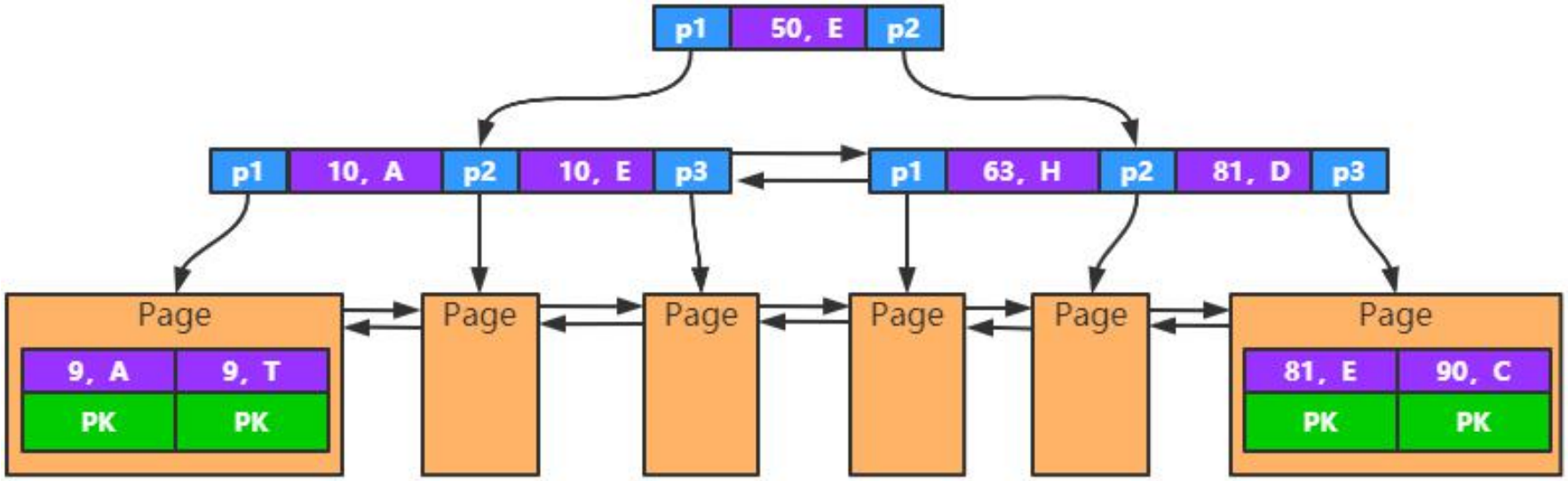


二级索引

- 除主键索引以外的索引
- 叶子中存储主键值
- 一次查询需要走两遍索引
- 主键大小会影响所有索引的大小

索引原理分析

- 聚簇索引
- 二级索引
- **联合索引**



联合索引

- Key由多个字段组成
- 最左匹配原则
- 一个索引只创建一棵树
- 按第一列排序，第一列相同按第二列排序

如果不是按照最左开始查找，无法使用索引

不能跳过中间列

某列使用范围查询，后面的列不能使用索引

索引使用优化分析

- 存储空间
- 主键选择
- 联合索引使用
- 字符串索引

索引实现原理

- **存储空间**
  - 索引文件大小；
  - 字段大小->页内节点个数->树的层数；
- 主键选择
- 联合索引使用
- 字符串索引

BIGINT类型主键3层可以存储约10亿条数据

16KB/(8B(key)+8B(指针))=1K

10^3 * 10^3 * 10^3 = 10亿

32字节主键3层可以存储6400W

索引使用优化分析

- 存储空间
- **主键选择**
  - 自增主键，顺序写入，效率高;
  - 随机主键，结点分裂、数据移动;
- 联合索引使用
- 字符串索引

自增主键：写入磁盘利用率高，每次查询走两级索引；

随机主键：写入磁盘利用率低，每次查询走两级索引；

业务主键：写入、查询磁盘利用率都高，可以使用一级索引；

联合主键：影响索引大小，不易维护，不建议使用；

索引使用优化分析

- 存储空间
- 主键选择
- **联合索引使用**
  - 按索引区分度排序
  - 覆盖索引
- 字符串索引

索引使用优化分析

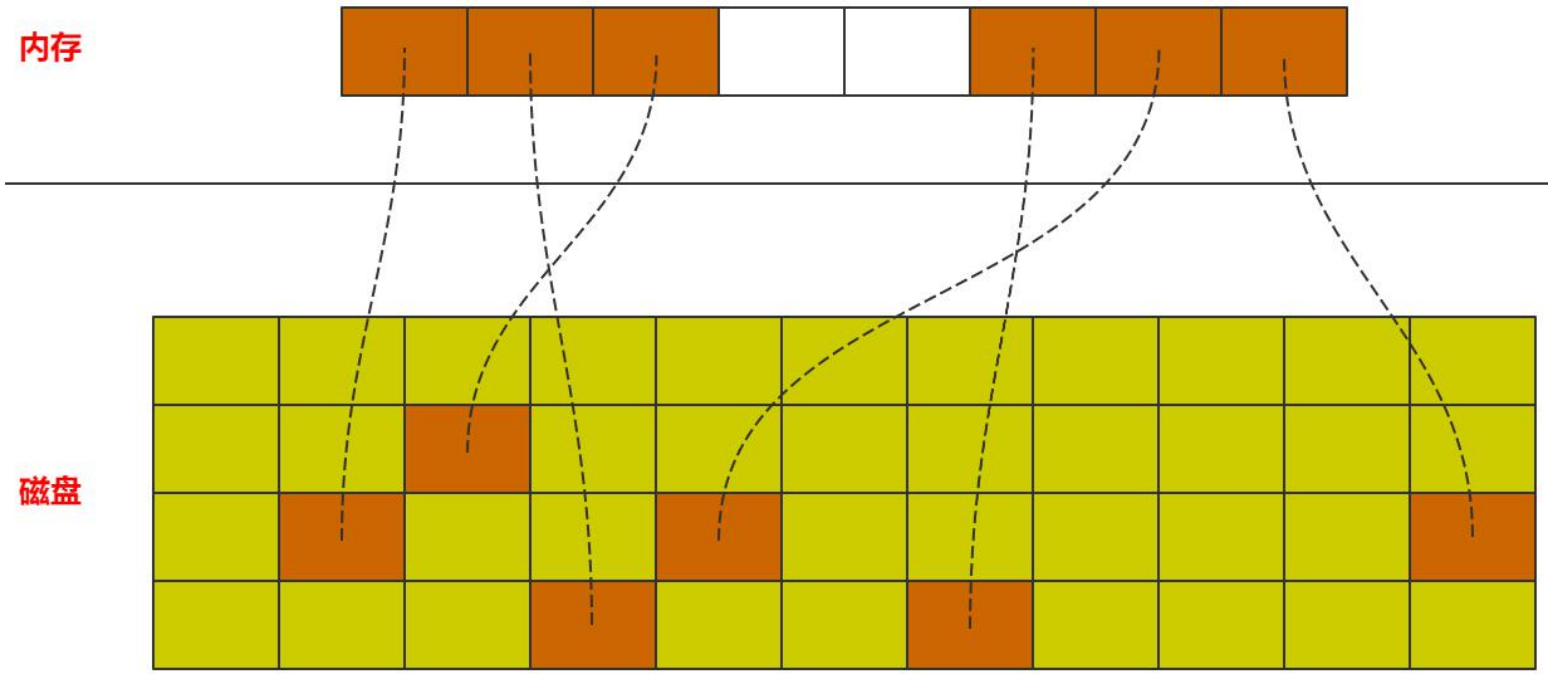
- 存储空间
- 主键选择
- 联合索引使用
- **字符串索引**
  - 设置合理长度
  - 不支持%开头的模糊查询

NX 奈学教育

03.MySQL InnoDB存储引擎内存管理

InnoDB内存管理

- 预分配内存空间
- 数据以页为单位加载
- 数据内外存交换



InnoDB内存管理—技术点

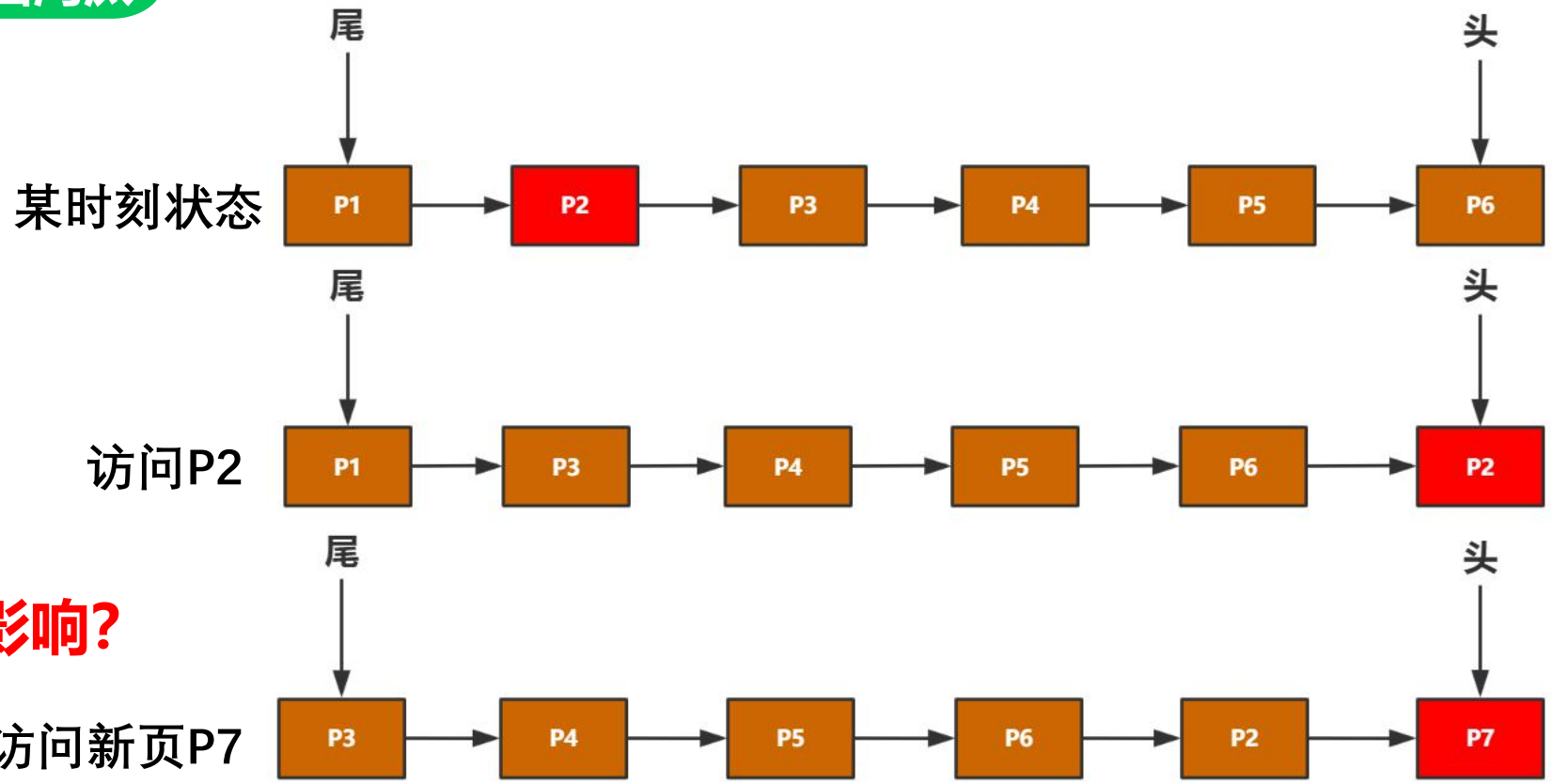
- 内存池
- 内存页面管理
  - 页面映射
  - 页面数据管理
- 数据淘汰
  - 内存页都被使用
  - 需要加载新数据

InnoDB内存管理—页面管理

- 空闲页
- 数据页
- 脏页

MySQL内存管理—页面淘汰

- LRU



思考：
全表扫描对内存的影响？

MySQL内存管理—页面淘汰

解决问题

- 避免热数据被淘汰

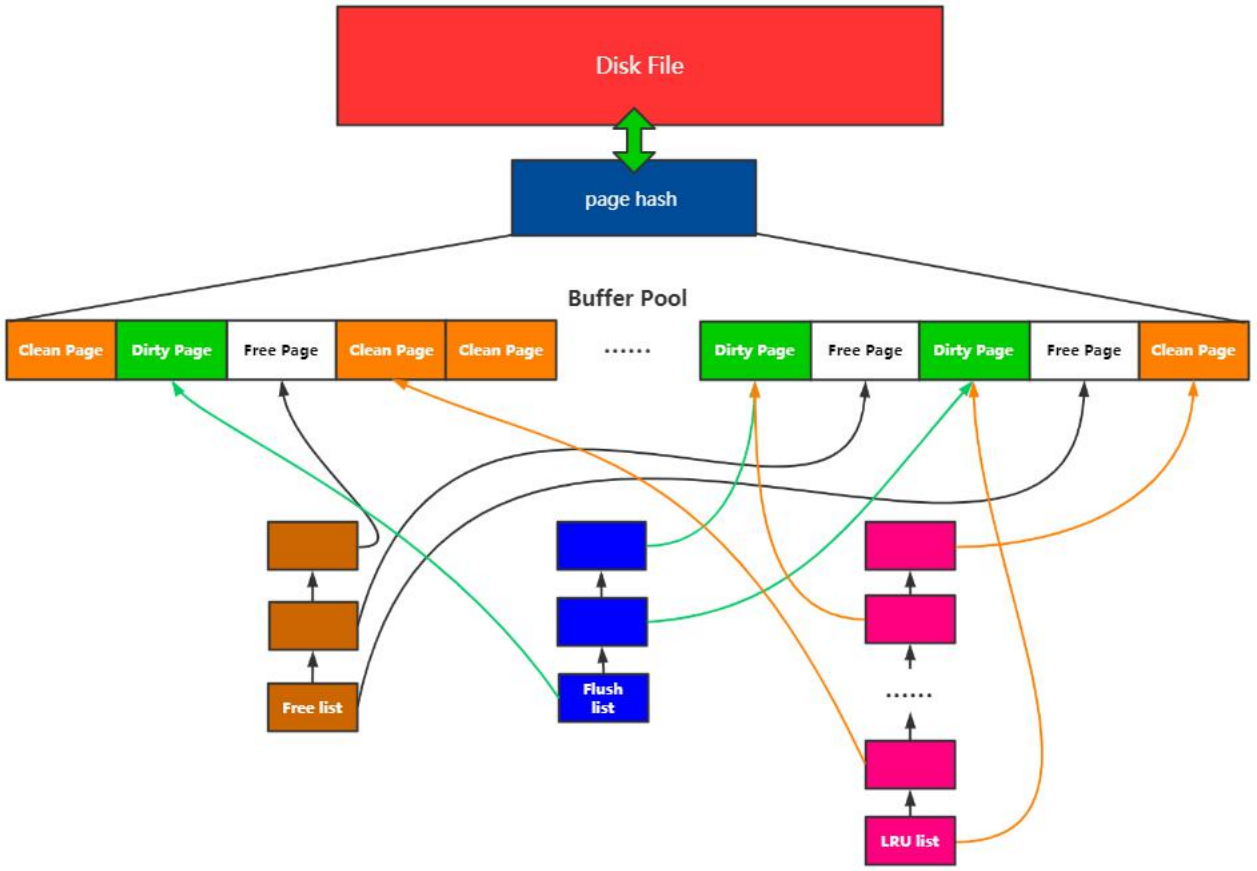
思路

- **访问时间 + 频率?**
- **两个LRU表?**

MySQL是如何解决的.....

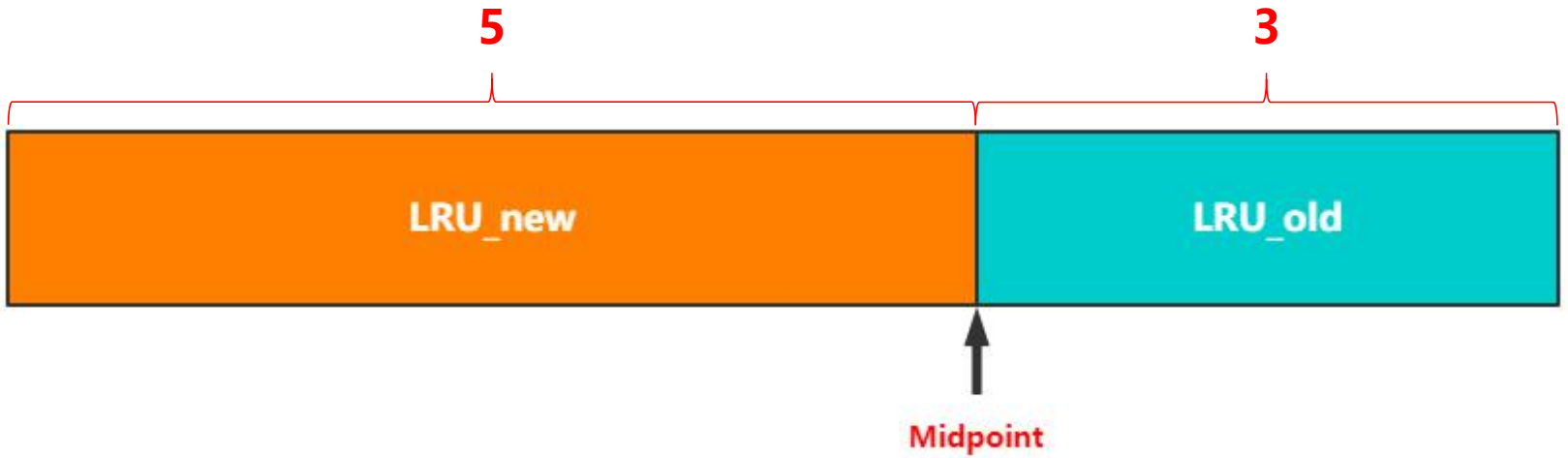
MySQL内存管理

- Buffer Pool
  - 预分配的内存池
- Page
  - Buffer Pool的最小单位
- Free list
  - 空闲Page组成的链表
- Flush list
  - 脏页链表
- Page hash 表
  - 维护内存Page和文件Page的映射关系
- LRU
  - 内存淘汰算法



MySQL内存管理—LRU

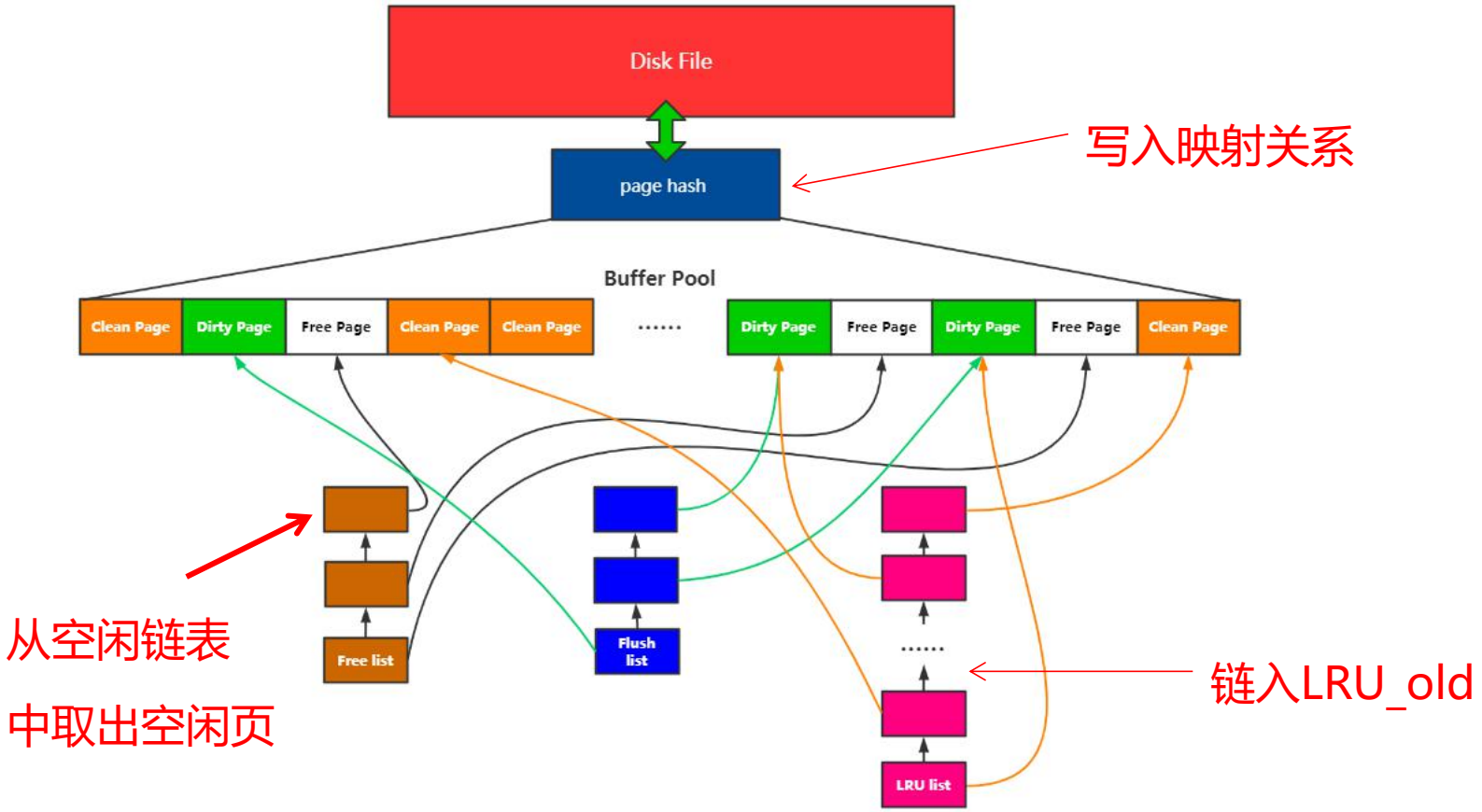
- LRU_new
- LRU_old
- Midpoint



冷热分离

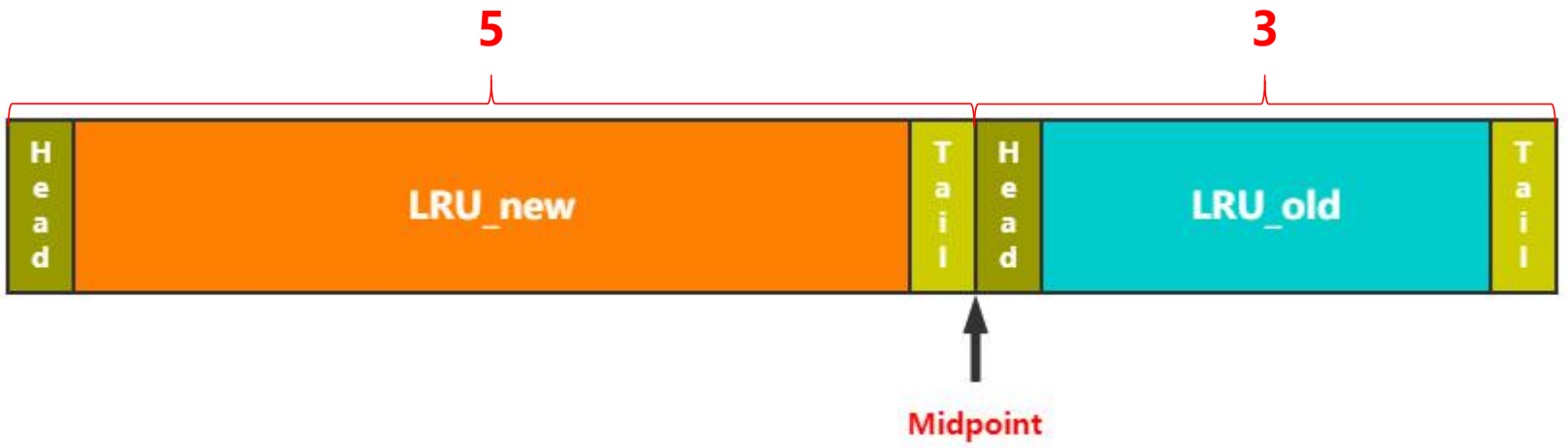
MySQL内存管理—LRU

- **页面装载**
  - 磁盘数据到内存
- 页面淘汰
- 位置移动
- LRU_new的操作



MySQL内存管理—LRU

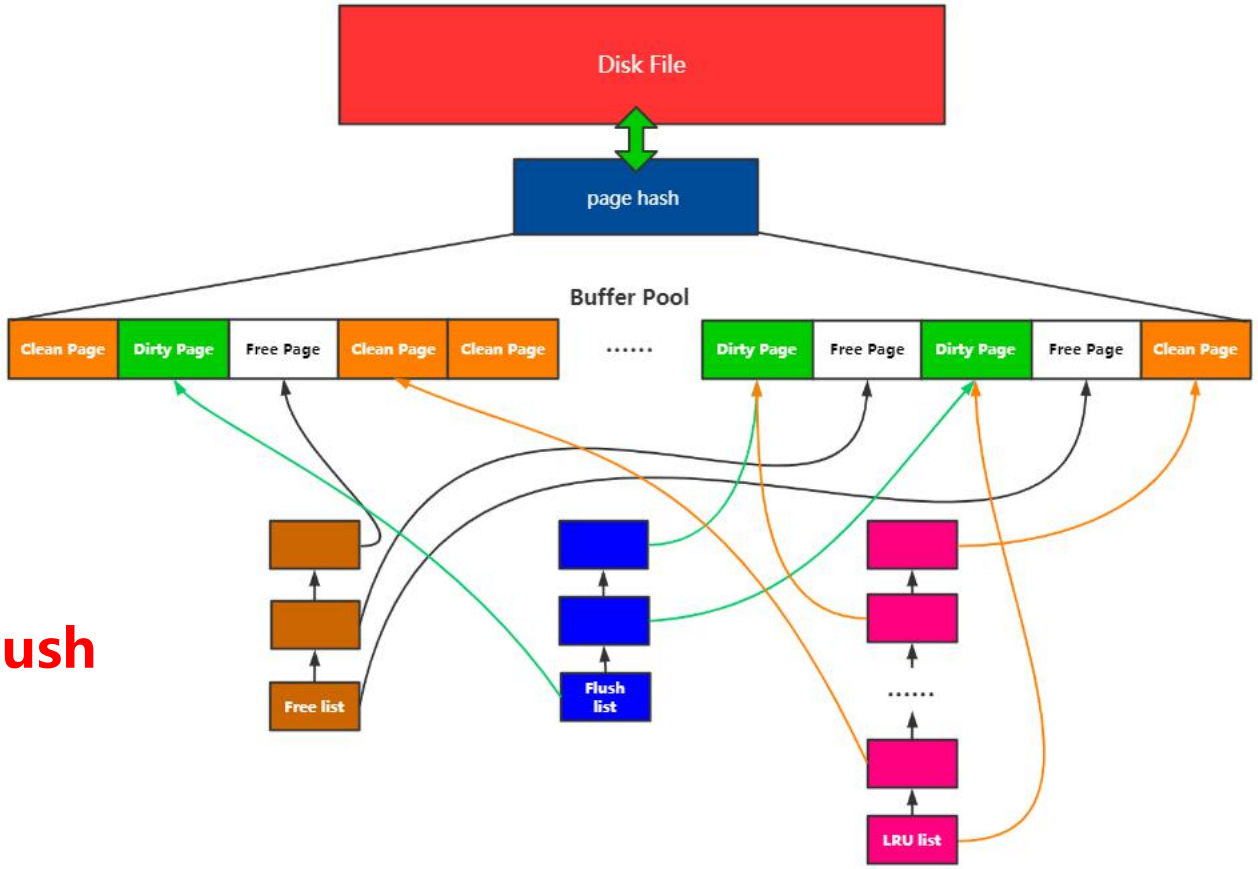
- **页面装载**
  - 磁盘数据到内存
- 页面淘汰
- 位置移动
- LRU_new的操作



没有空闲页怎么办?

MySQL内存管理—LRU

- **页面装载**
  - 磁盘数据到内存
- 页面淘汰
- 位置移动
- LRU_new的操作



Free list中取 > LRU中淘汰 > LRU Flush

MySQL内存管理—LRU

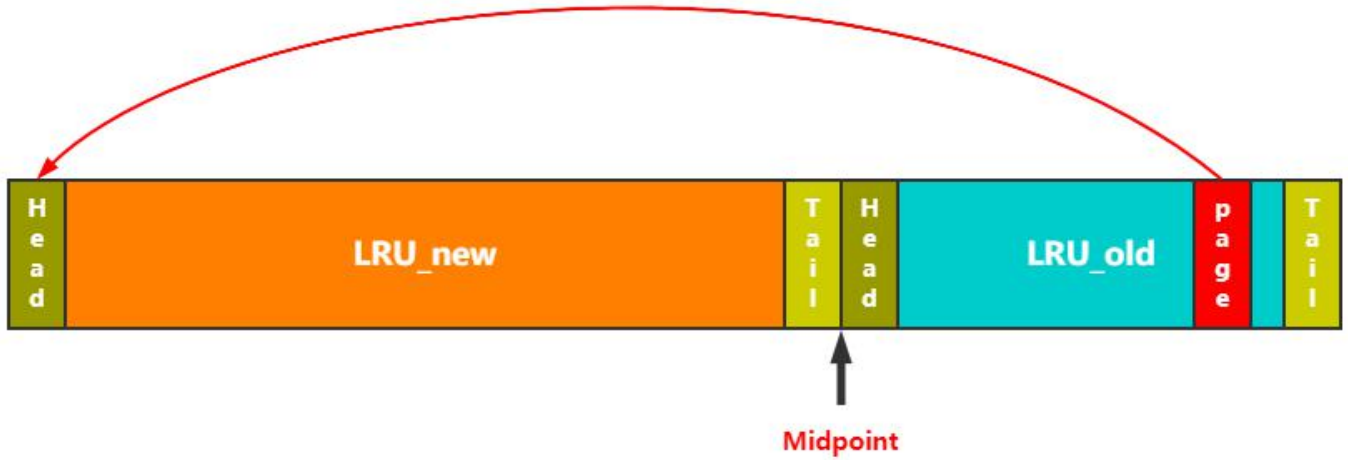
- 页面装载
- **页面淘汰**
  - LRU尾部淘汰
  - Flush LRU淘汰
- 位置移动
- LRU_new的操作

LRU链表中将第一个脏页刷盘并“释放”

放到LRU尾部？直接放FreeList？

MySQL内存管理—LRU

- 页面装载
- 页面淘汰
- **位置移动**
  - **old 到 new**
  - new 到 old
- LRU_new的操作

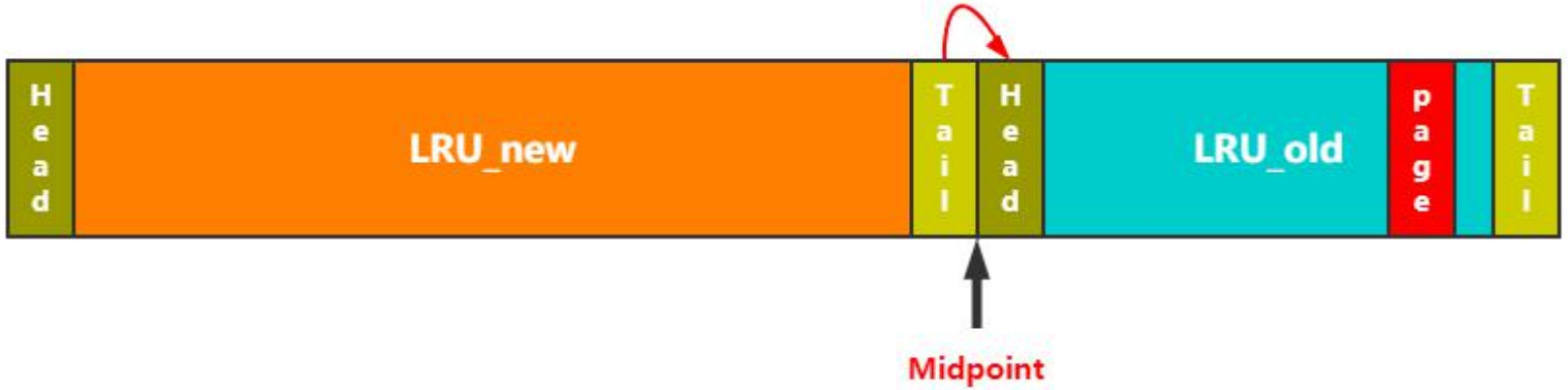


思考：移动时机

innodb_old_blocks_time
old区存活时间，大于此值，有机会进入new区

MySQL内存管理—LRU

- 页面装载
- 页面淘汰
- **位置移动**
  - old 到 new
  - **new 到 old**
- LRU_new的操作



Midpoint：指向5/8位置

MySQL内存管理—LRU

- 页面装载
- 页面淘汰
- 位置移动
- **LRU_new的操作**

链表操作效率很高，有访问移动到表头？

Lock！！！

MySQL设计思路：减少移动次数

两个重要参考：1、freed_page_clock：Buffer Pool淘汰页数

2、LRU_new长度1/4

当前freed_page_clock - 上次移动到Header时freed_page_clock

>

LRU_new长度1/4

NX 奈学教育

04.MySQL事务管理机制原理分析

MySQL事务基本概念

- 事务特性
- 并发问题
- 隔离级别

MySQL事务基本概念

- **事务特性**
  - A（Atomicity原子性）：全部成功或全部失败
  - I（Isolation隔离性）：并行事务之间互不干扰
  - D（Durability持久性）：事务提交后，永久生效
  - C（Consistency一致性）：通过AID保证
- 并发问题
- 隔离级别

MySQL事务基本概念

- 事务特性
- **并发问题**
  - **脏读(Drity Read)**：读取到未提交的数据
  - **不可重复读(Non-repeatable read)**：两次读取结果不同
  - **幻读(Phantom Read)**：select 操作得到的结果所表征的数据状态无法支撑后续的业务操作
- 隔离级别

MySQL事务基本概念

- 事务特性
- 并发问题
- **隔离级别**
  - **Read Uncommitted**（读取未提交内容）：最低隔离级别，会读取到其他事务未提交的数据，脏读；
  - **Read Committed**（读取提交内容）：事务过程中可以读取到其他事务已提交的数据，不可重复读；
  - **Repeatable Read**（可重复读）：每次读取相同结果集，不管其他事务是否提交，幻读；
  - **Serializable**（串行化）：事务排队，隔离级别最高，性能最差；

MySQL事务实现原理

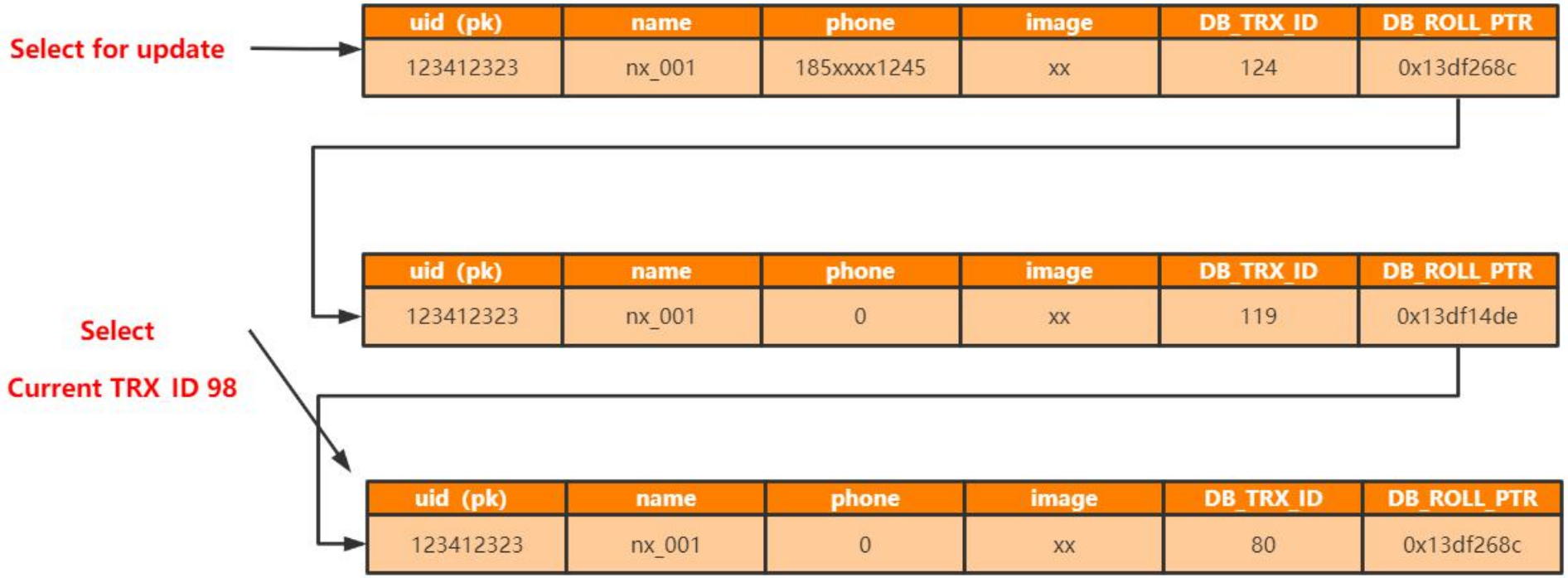
- MVCC
- undo log
- redo log

MySQL事务实现原理

- **MVCC**
  - 多版本并发控制
  - 解决读-写冲突
  - 隐藏列
- undo log
- redo log

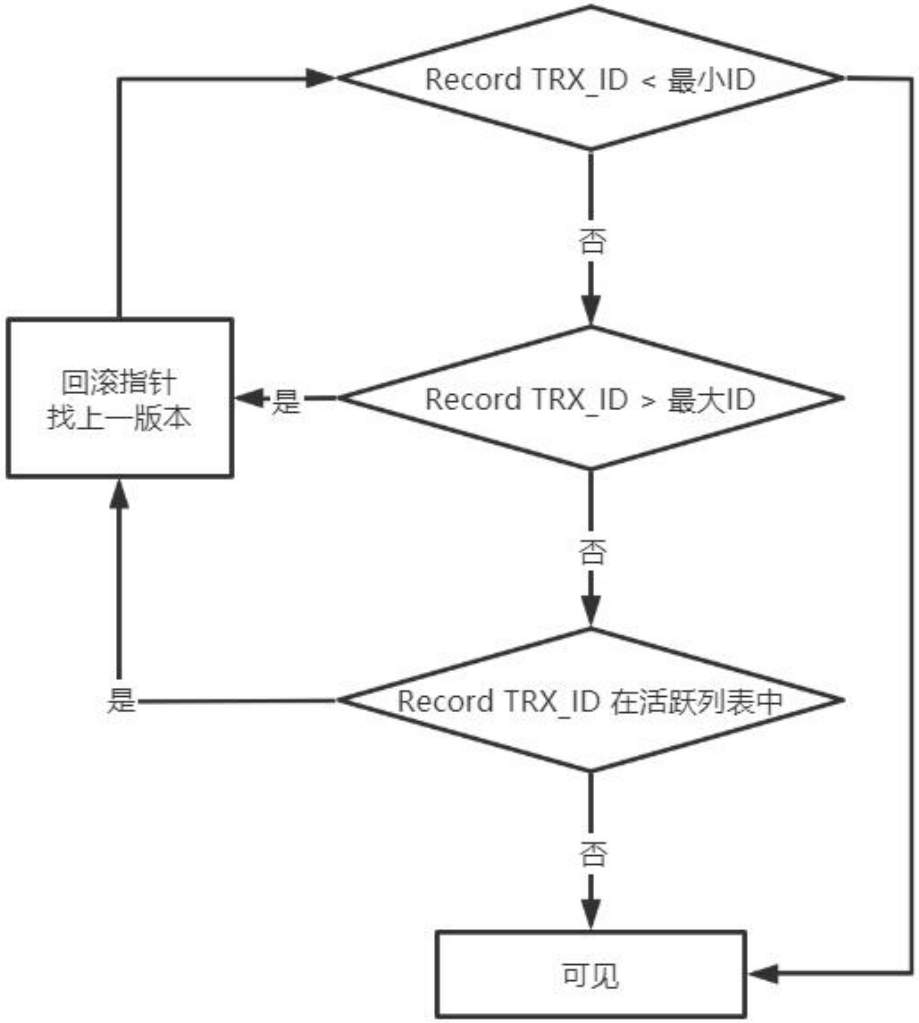
MySQL—MVCC

- 当前读
- 快照读



MySQL—MVCC

- 可见性判断
  - 创建快照这一刻，还未提交的事务；
  - 创建快照之后创建的事务；
- Read View
  - 快照读 活跃事务列表
  - 列表中最小事务ID
  - 列表中最大事务ID



NX 奈学教育

欢迎关注本人公众号
“架构之美”

MySQL事务实现原理

- MVCC
- **undo log**
  - 回滚日志
  - 保证事务原子性
  - 实现数据多版本
  - delete undo log：用于回滚，提交即清理；
  - update undo log：用于回滚，同时实现快照读，不能随便删除
- redo log

MySQL—undolog

MySQL—undolog

思考：undolog如何清理

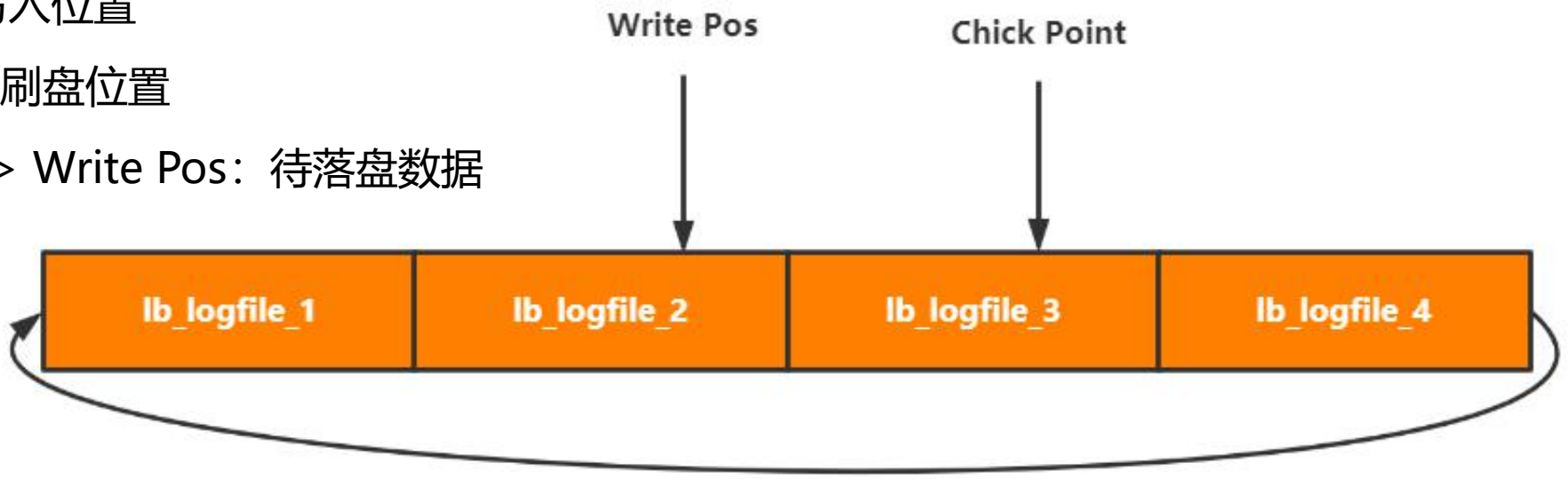
依据系统活跃的最小活跃事务ID Read view

MySQL事务实现原理

- MVCC
- undo log
- **redo log**
  - 实现事务持久性

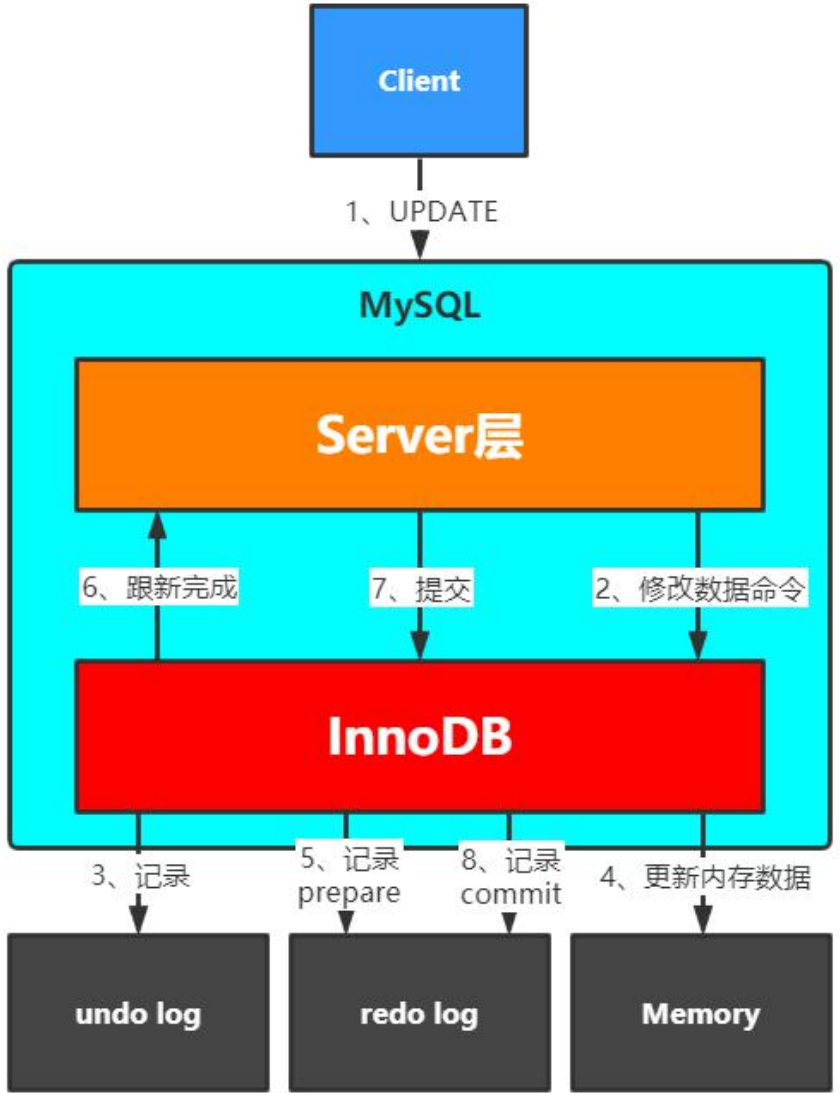
MySQL—redolog

- 记录修改
- 用于异常恢复
- 循环写文件
  - Write Pos：写入位置
  - Chick Point：刷盘位置
  - Chick Point -> Write Pos：待落盘数据



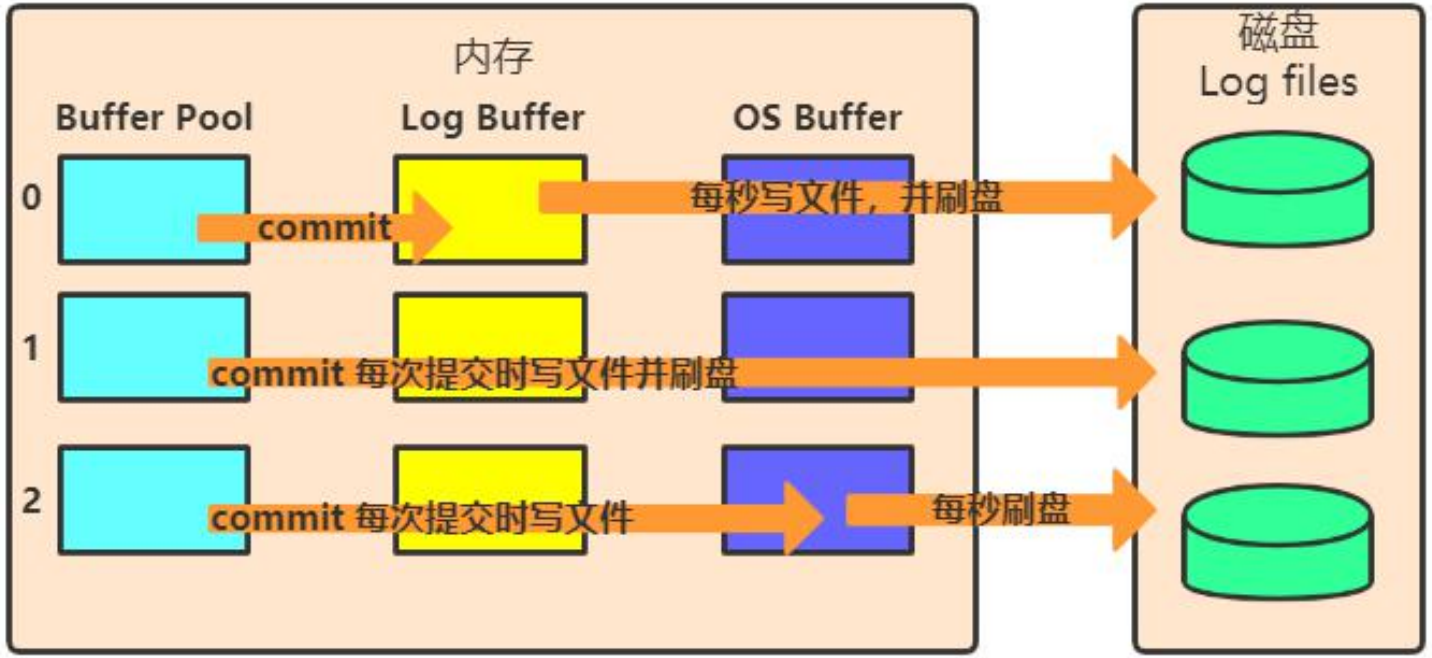
MySQL—redolog

- 写入流程
  - 记录页的修改，状态为prepare
  - 事务提交，讲事务记录为commit状态



MySQL—redolog

- 刷盘时机
  - innodb_flush_log_at_trx_commit



MySQL—redolog

- 意义
  - 体积小，记录页的修改，比写入页代价低
  - 末尾追加，随机写变顺序写，发生改变的页不固定

NX 奈学教育

欢迎关注本人公众号
“架构之美”