数据库系统原理

教程：数据库系统概论（第5版）

结合：CMU 15-445/645 INTRO TO DATABASE SYSTEMS

华中科技大学 计算机学院

左琼

第八章　关系数据库引擎基础

Principles of Database Systems

第八章 关系数据库引擎基础

8.1 数据库存储

8.2 缓冲池

8.3 索引

8.2.1 缓冲池工作原理

8.2.2 缓冲池结构

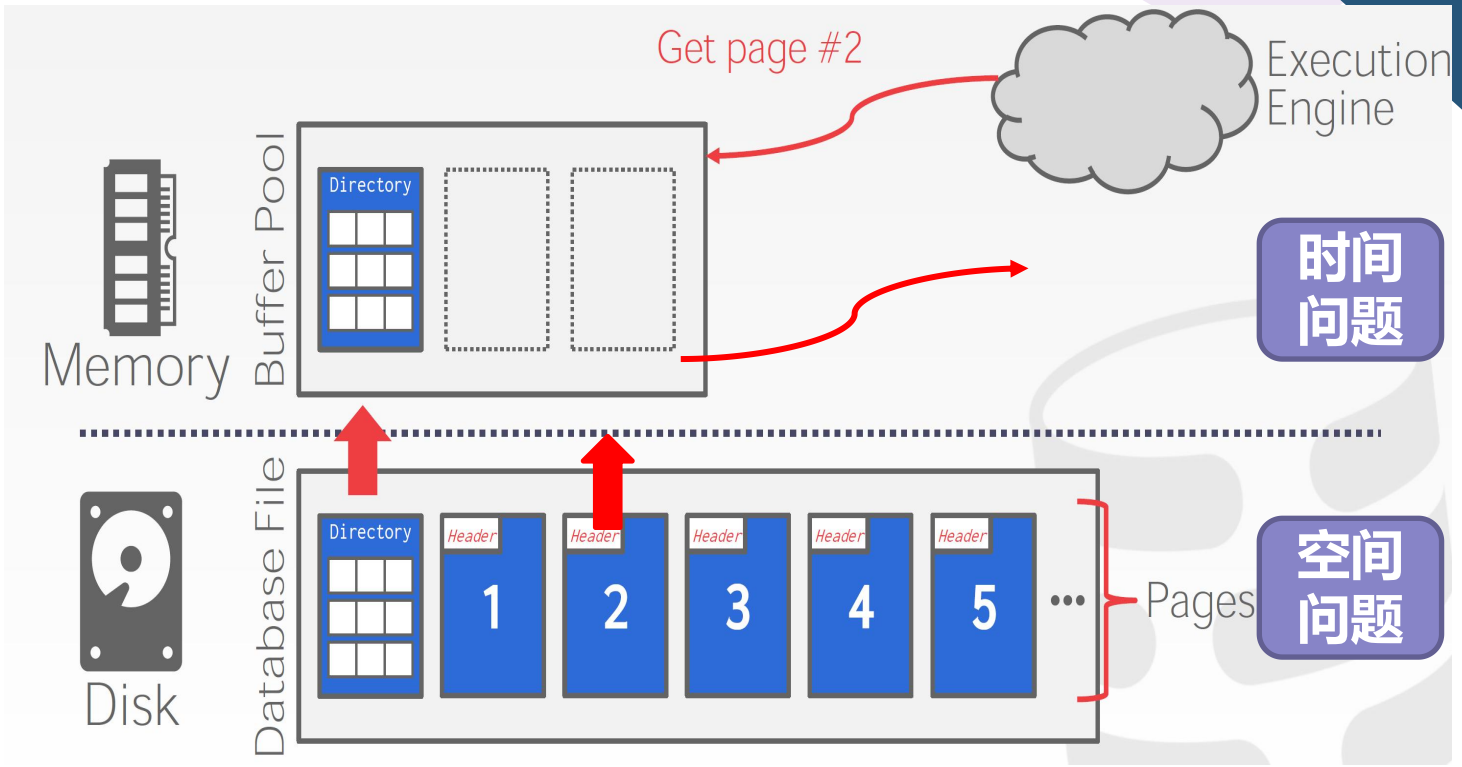
8.2.3 缓冲池使用

8.2.4 缓冲池替换算法

8.2.5 缓冲池的优化

8.2.1 缓冲池工作原理

- 执行引擎在语句处理过程中需要使用某个数据页时，会向缓冲池提出请求；
- 缓冲池管理器负责将该页从磁盘读入内存，并向执行引擎提供该页在内存中的指针；
- 当执行引擎操作那部分内存时，缓冲池管理器必须确保该页面始终驻留在那片内存区域中。



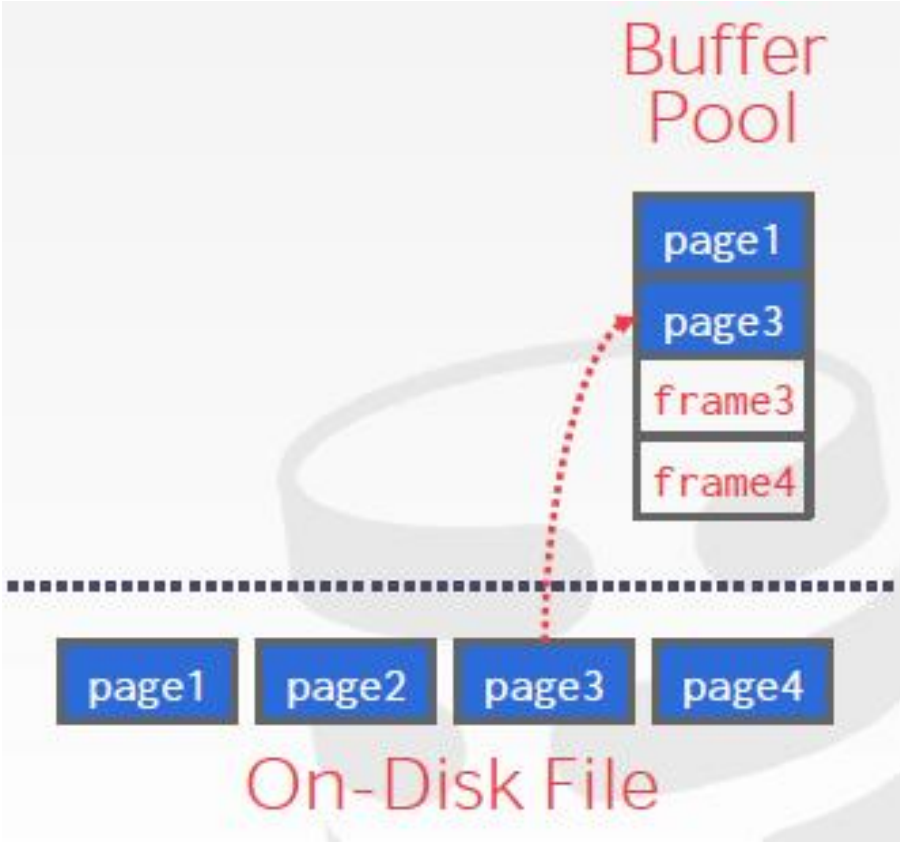
本节重点：DBMS如何管理数据在磁盘和缓存之间的交互，缩小CPU速度与磁盘速度之间的鸿沟。

缓冲池的作用

- 减少磁盘I/O，提升页面读写效率；
  - 读操作时，若所需页面已加载到缓冲池，就不需要从磁盘读了；
  - 写操作时，如果多次修改了同一页面，只需要写一次磁盘。
- 缓冲池管理器的目标：尽可能减少磁盘I/O带来的延时。

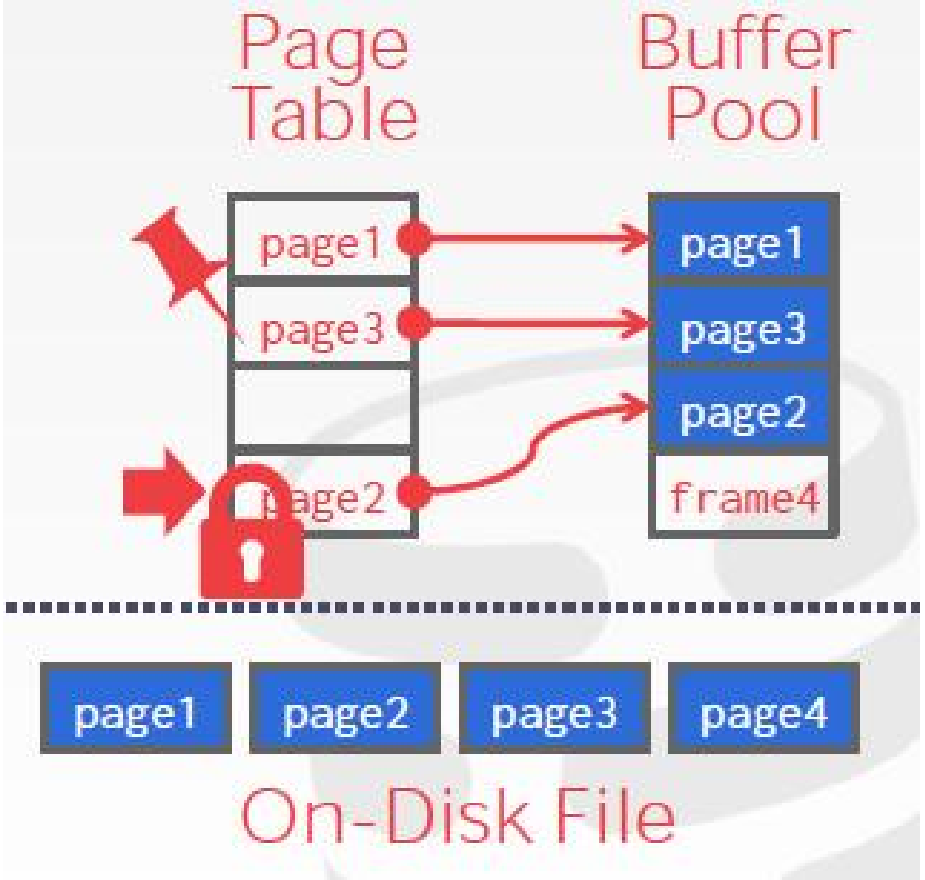
8.2.2 缓冲池结构

- 缓冲池是在DBMS内部分配的一大片内存区域，用于存储从磁盘获取的页面。
- 用于缓存页面的内存空间被组织为一个数组，其中每个数组项被称为一个帧（frame），一个帧正好能放置一个页面。
- 页表是缓冲池管理器用于维护缓冲池元数据的数据结构。



缓冲池页表

- 页表：是一个内存散列（hash）表，用于登记当前已加载到缓冲池中的页面的信息。
- 页表为缓冲池中每个页面维护以下信息：
  - 位置信息：页面在缓冲池中对应帧的位置；
  - 脏标志：页面被修改过的标志。如果一个页面被设置了脏标志，则缓冲池管理器必须确保将其写回磁盘；
  - 引用计数：当前正在访问该页的线程数量。如果某页面的引用计数大于零，则不允许淘汰该页（钉住，pin）。
  - 并发时?



8.2.3 缓冲池管理策略

- 锁（lock）vs 闩（latch）
- 锁和闩分别用于DBMS中两个级别的并发控制。
- 锁：事务级
  - 保护的是数据库中的逻辑对象，如表、元组等；
  - 持锁时间较长，一般直到事务提交才释放；
  - 上锁期间对上锁对象的修改可以回滚。
- 闩：线程（进程）级
  - 保护的是DBMS中多线程/进程共享的内部数据结构（临界资源），如帧；
  - 一般用OS的同步机制（如信号量）实现，加闩时间短，操作完立刻释放；
  - 加闩期间的修改无需考虑回滚。

8.2.3 缓冲池管理策略

分配策略:

关于缓冲池中的内存空间如何分配的问题，缓冲池管理器可采取2种策略：

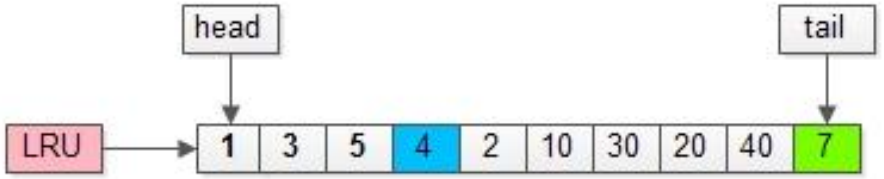
- 全局策略
  - 考虑所有活动事务，以找到分配内存的最佳方案。
- 局部策略
  - 以保证单个查询或事务运行得更快为目标，不考虑其他的并发事务；
  - 仍然需要支持共享页面。

8.2.4 缓冲池替换算法

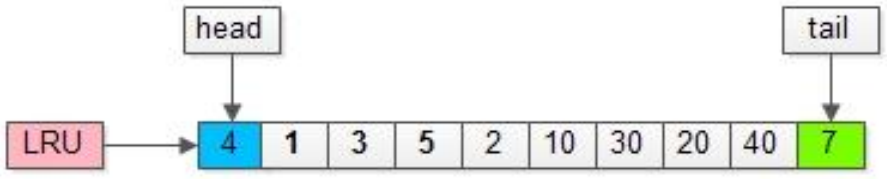
- DBMS采用的缓冲池替换算法：当DBMS需要释放一个帧来为新的页面腾出空间时，它必须决定从缓冲池中淘汰哪个页面。替换算法的目标是提高正确性、准确性、速度和减少元数据开销。
- 常用替换算法：
  - LRU算法：为每个页面维护其最后一次被访问的时间戳，需要淘汰页面时，DBMS总是选择淘汰时间戳最早的页面。
  - CLOCK算法：也称近似LRU算法，为每个页面维护一个引用位，当某个页面被访问时，就将其引用位的值置为1。需要淘汰页面时，循环扫描缓冲池中的页面，检查各页面的引用位是否为1。如是，则将引用位置为0并移动指针到下一个页面；否则淘汰当前页面。

传统LRU算法

- 把入缓冲池的页放到LRU的头部，作为最近访问的元素，从而最晚被淘汰。为了减少数据移动，LRU一般用链表实现。
- 两种情况：
  - **页已经在缓冲池里**：那就只做**移至**LRU头部的动作，而没有页被淘汰

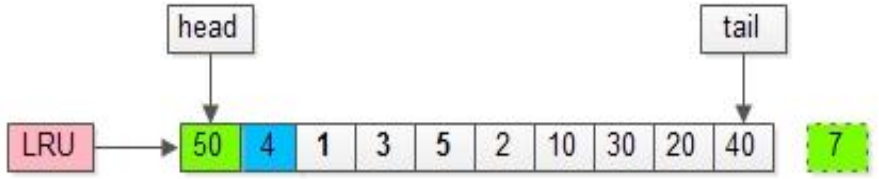


(a) 初始LRU链



(b) 访问页4，页4移动到LRU链头

  - **页不在缓冲池里**：除了**放入**LRU头部的动作，还要**淘汰**LRU尾部页的动作



(c) 访问页50，将页50放到LRU链头，淘汰链尾页7

2.4 缓冲池替换算法

- LRU算法和CLOCK算法的缺点：
  - 预读失效：由于预读，提前把页放入了缓冲池，但最终DBMS并没有从页中读取数据。
  - 顺序洪泛（sequential flooding）问题：因一次顺序扫描需将表的所有页面读入缓存，导致缓存污染问题（即：把不常用的数据读取到缓存中的现象）。

【例】当某一个SQL语句:

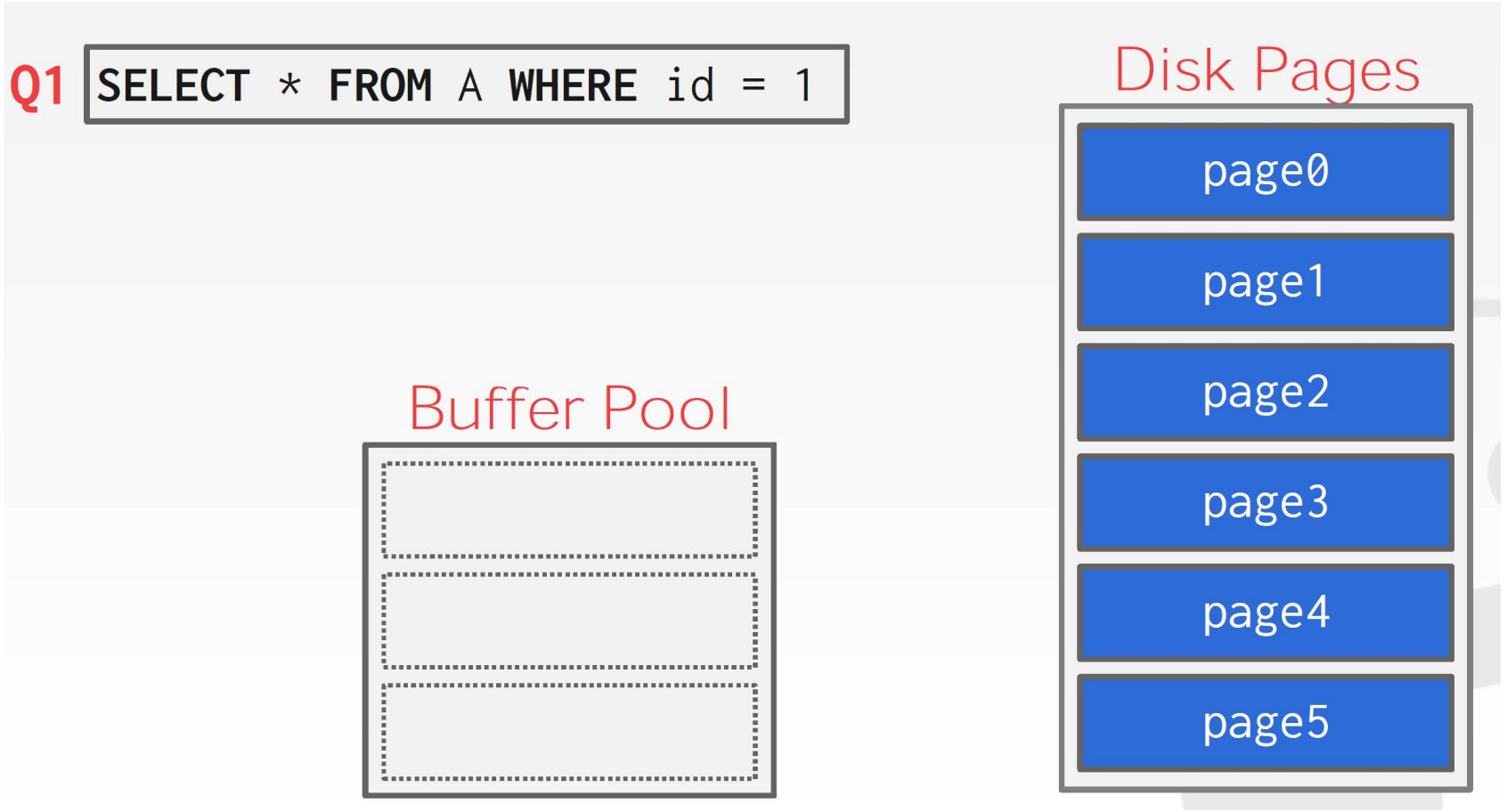
```
select * from user where name like  "%阳%";
```

要全表扫描，批量扫描大量数据时，可能导致把缓冲池的所有页都替换出去，导致大量热数据被换出，DBMS性能急剧下降。

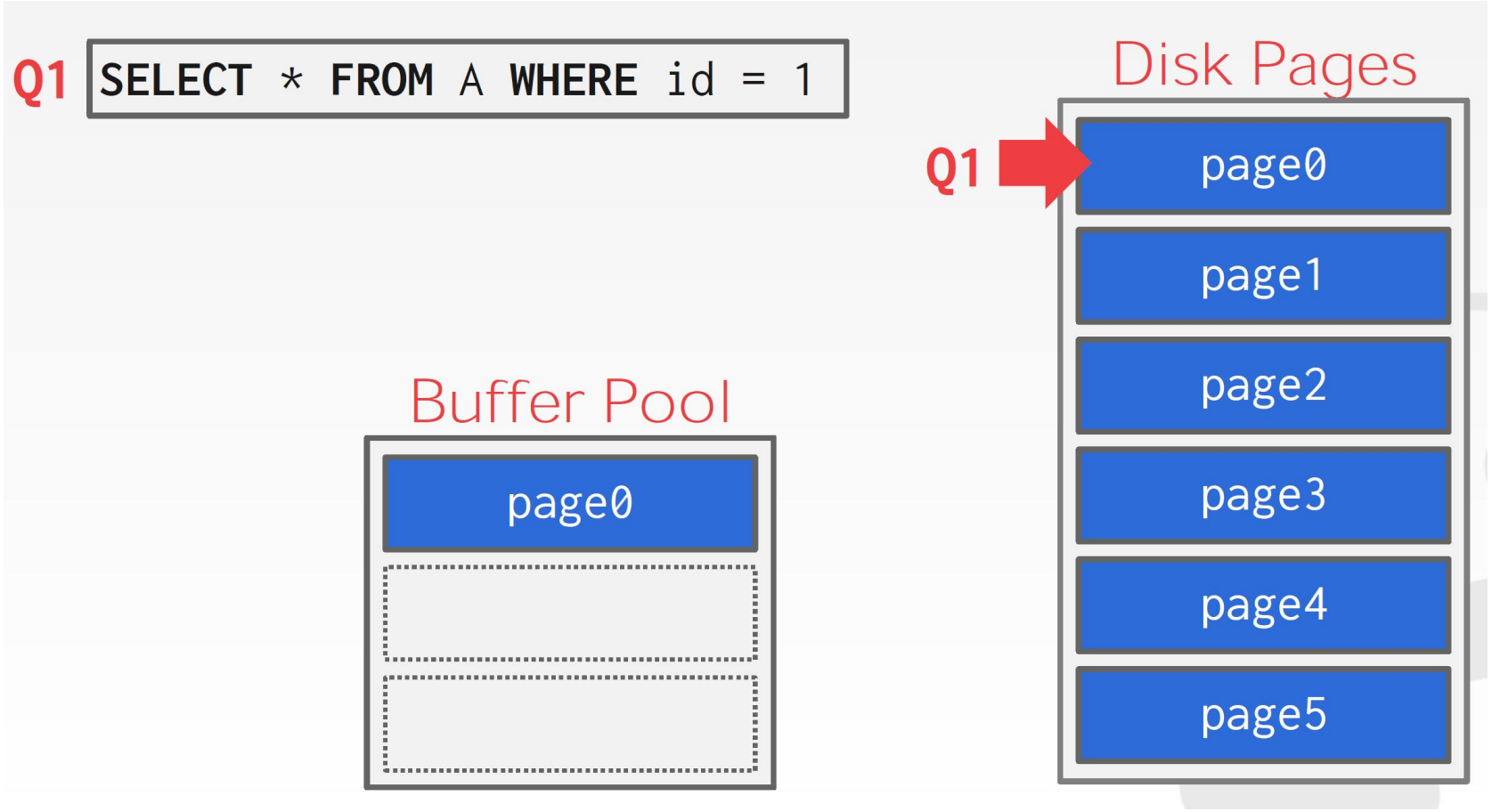
最近使用的页面可能是最不需要的页面！

顺序洪泛的例子

Q1 `SELECT * FROM A WHERE id = 1`



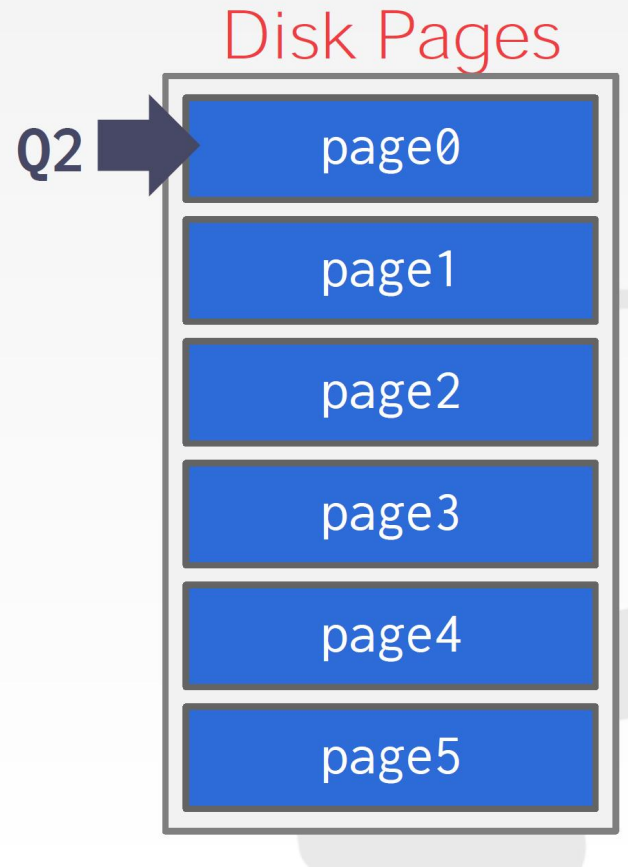
顺序洪泛的例子


Q1
SELECT * FROM A WHERE id = 1
Disk Pages
Q1
page0
page1
page2
page3
page4
page5
Buffer Pool
page0

顺序洪泛的例子

Q1 `SELECT * FROM A WHERE id = 1`

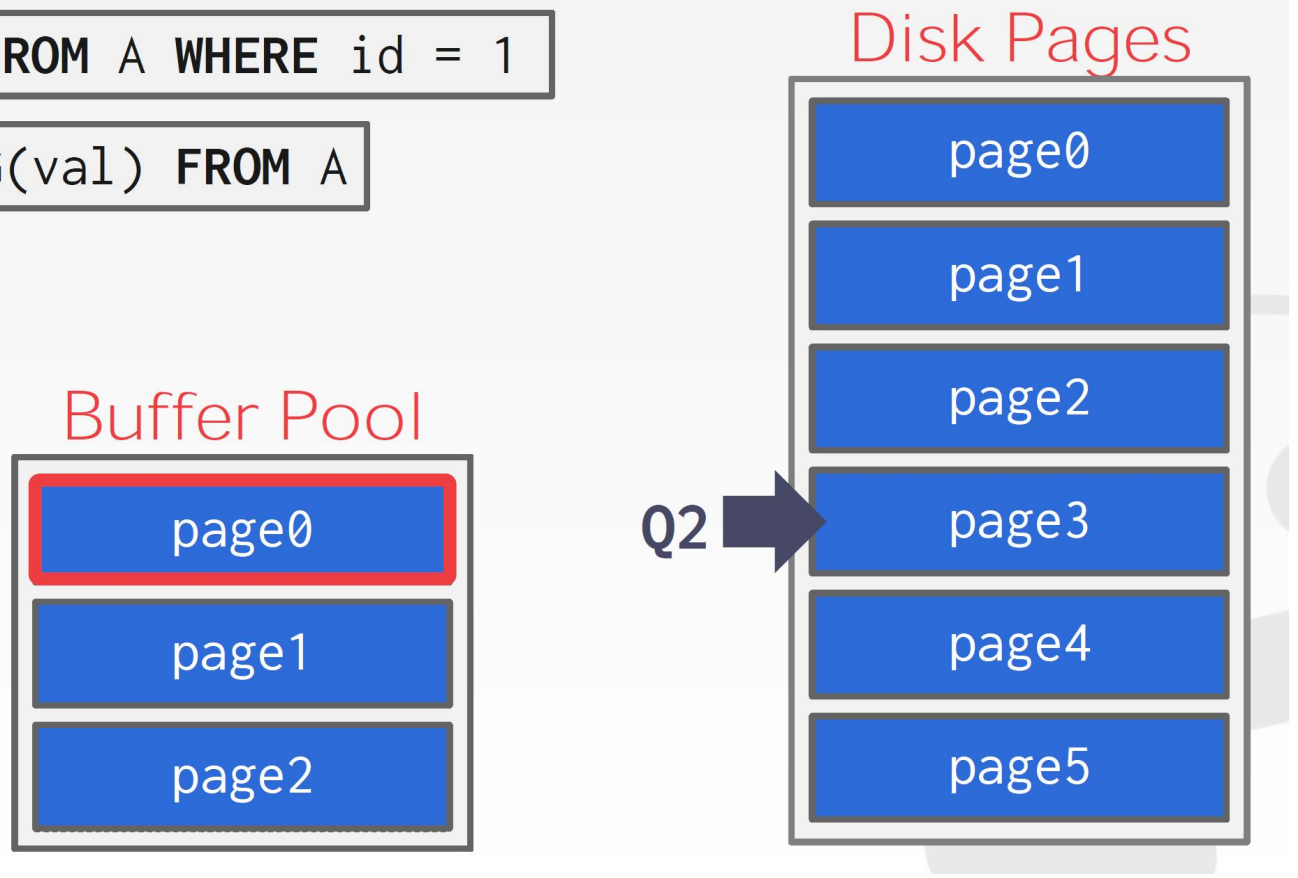
Q2 `SELECT AVG(val) FROM A`



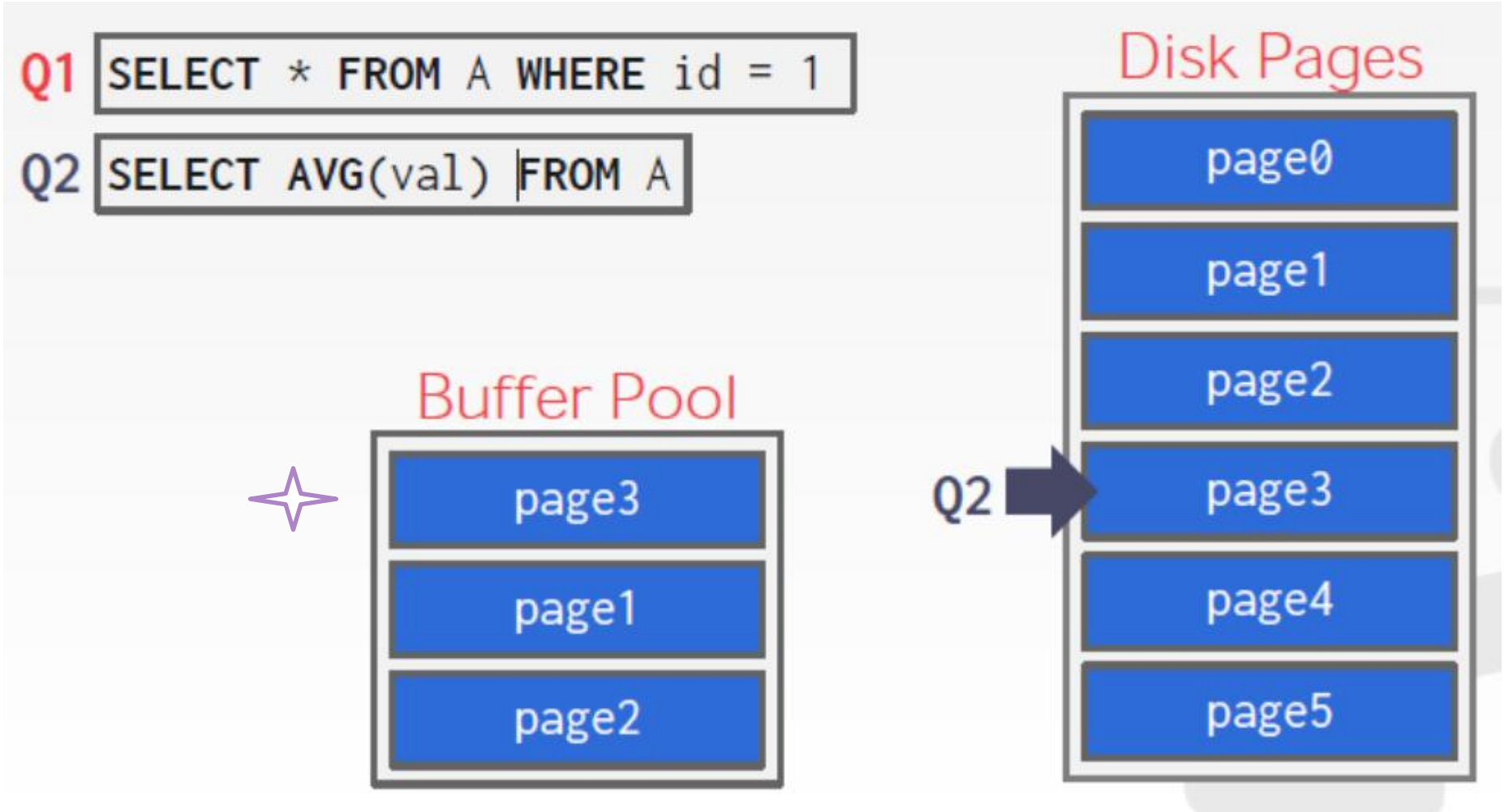
顺序洪泛的例子

Q1 `SELECT * FROM A WHERE id = 1`

Q2 `SELECT AVG(val) FROM A`

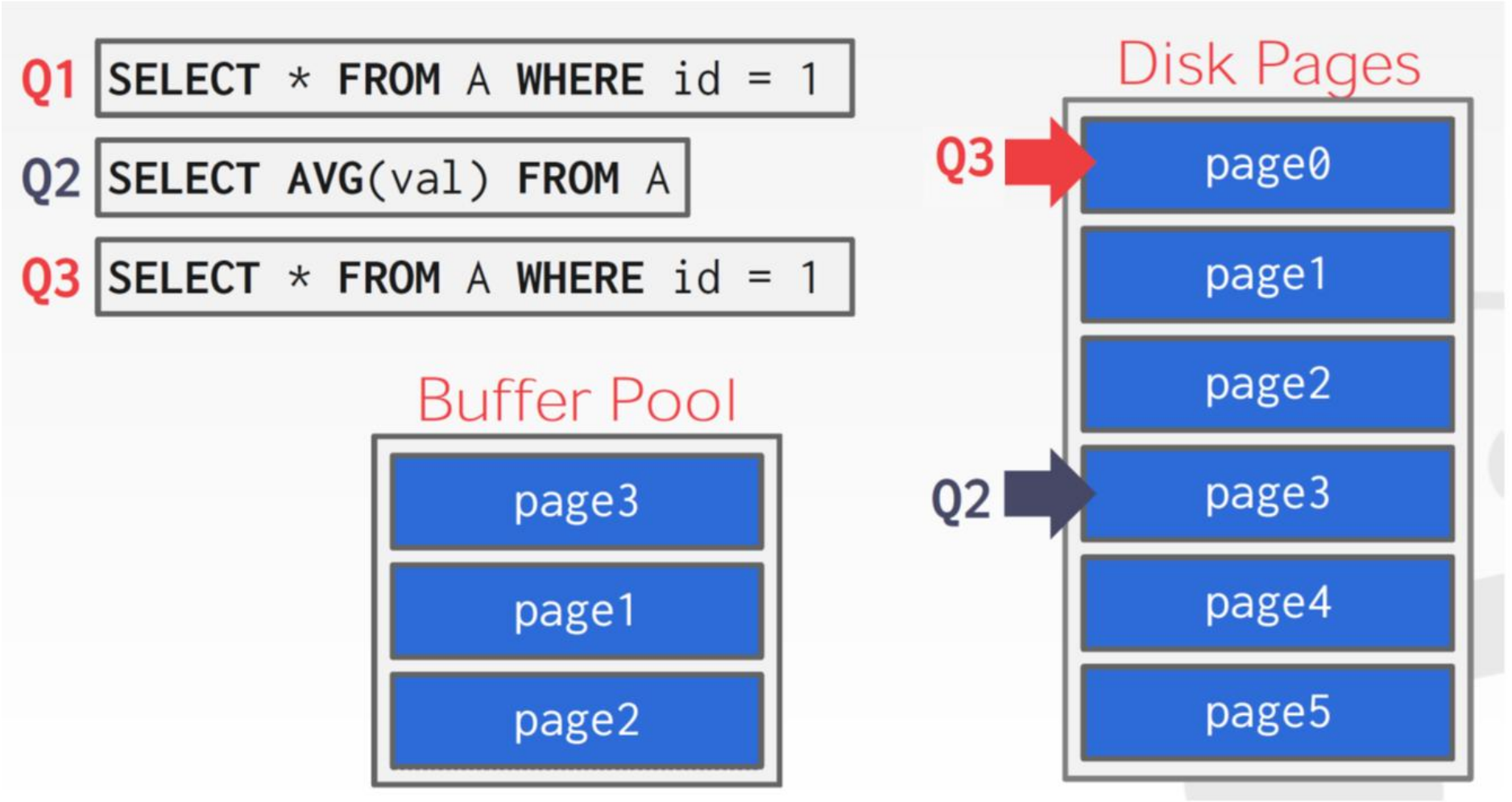


顺序洪泛的例子


Q1
SELECT * FROM A WHERE id = 1
Q2
SELECT AVG(val) FROM A
Buffer Pool
page3
page1
page2
Disk Pages
page0
page1
page2
Q2
page3
page4
page5

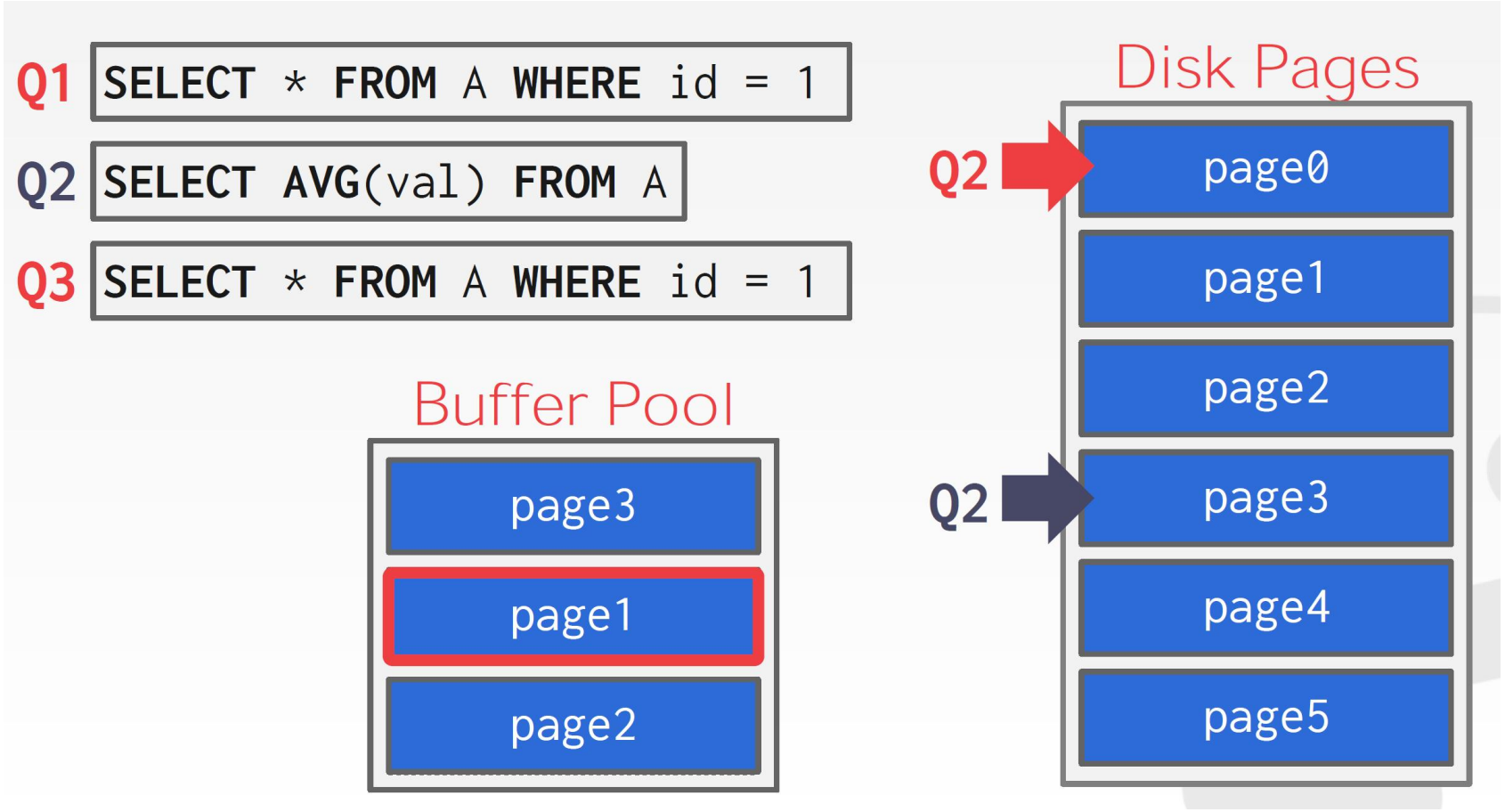
顺序洪泛的例子

- Q3


Q1 SELECT * FROM A WHERE id = 1
Q2 SELECT AVG(val) FROM A
Q3 SELECT * FROM A WHERE id = 1
Buffer Pool
page3
page1
page2
Disk Pages
Q3
Q2
page0
page1
page2
page3
page4
page5

顺序洪泛的例子

- Q3

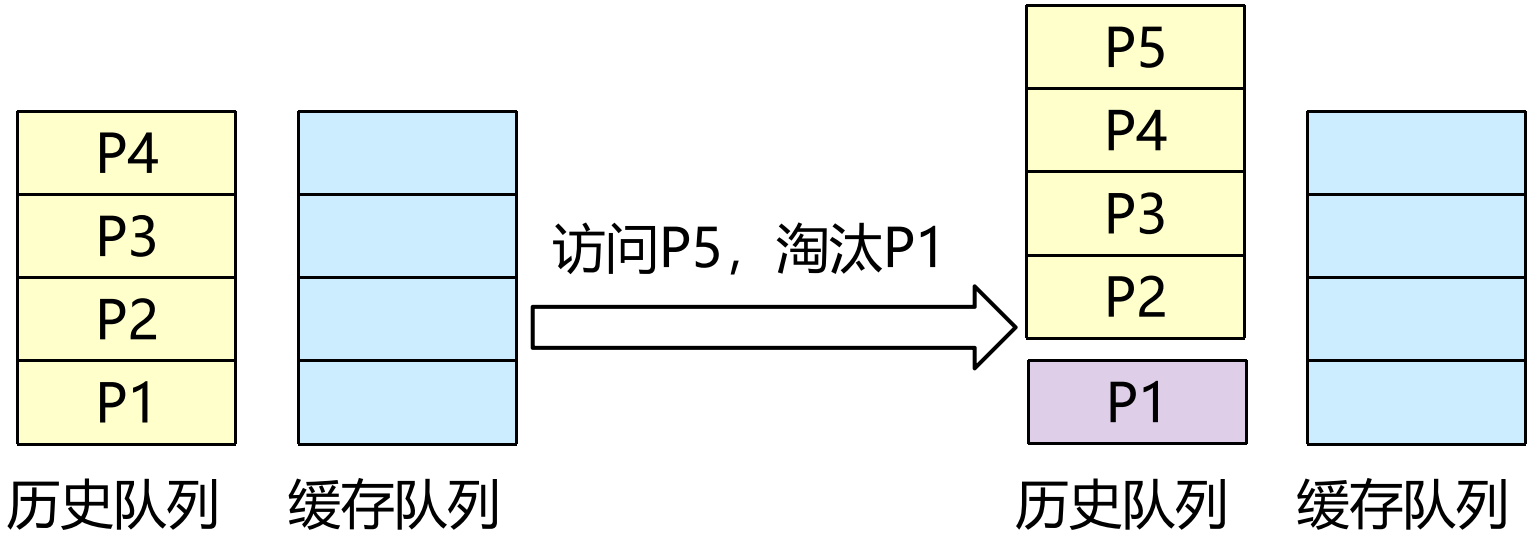


- 由于Q2的顺序扫描操作，导致真正会被再次访问的page0页面被淘汰。

LUR-K算法

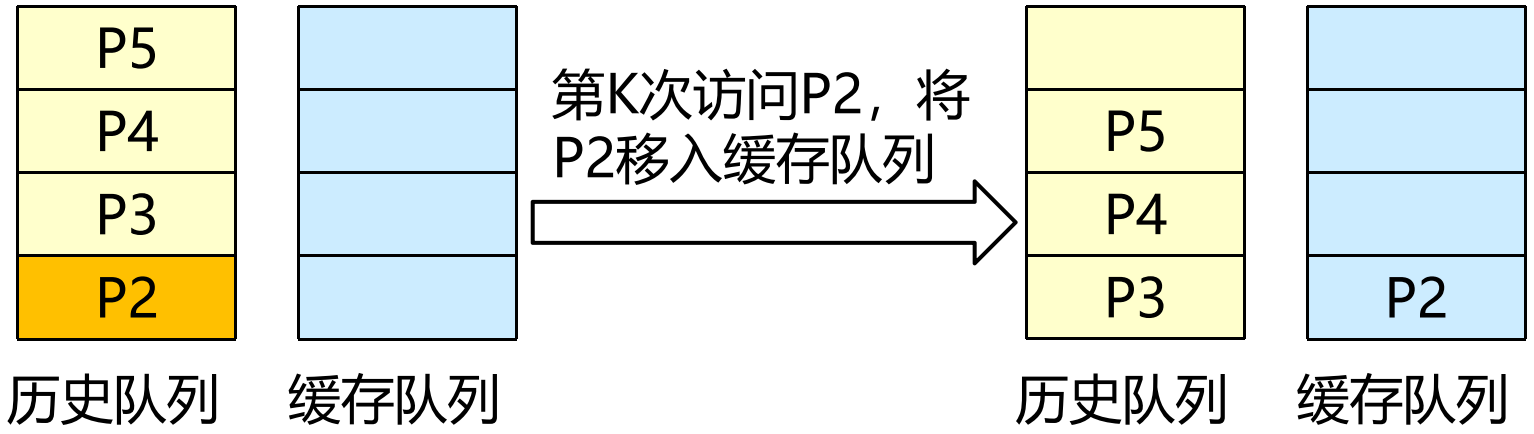
- LRU-K中的K是指最近访问页面的次数，LRU算法可看作是LRU-1，但是由于淘汰页面时仅考虑最近1次访问的情况，可能会因偶发的批量访问导致缓存污染，因此提出了LRU-K的概念。
- LRU-K算法的核心思想：将淘汰页面时所考察的最近访问次数由1提升为K。
- LRU-K算法维护两个队列：
  - 历史队列：保存着新进入内存的页面及其访问次数，还有每一次的访问时间，当一个页面的访问次数达到K次，则将该页面保存至缓存队列；若历史队列满了，则根据一定的淘汰策略（FIFO、LRU）淘汰。
  - 缓存队列：保存着已经访问过K次的页面，当该队列满了之后，则根据LRU策略淘汰倒数第K次访问距离现在最久的那个页面。

LUR-K算法



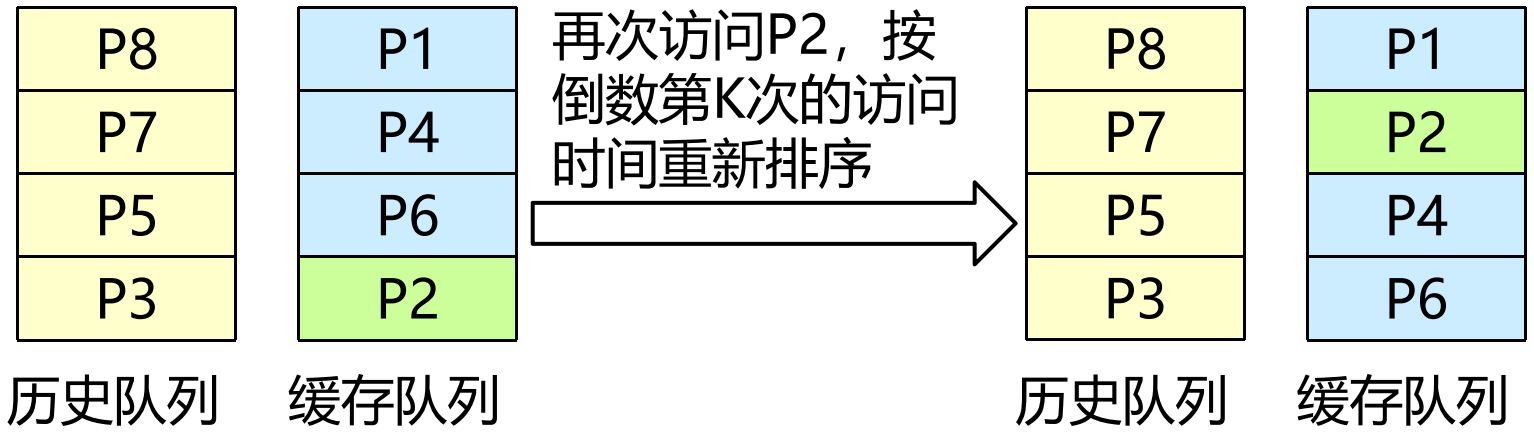
- 页面第一次被访问，添加到历史队列中。
- 若历史队列满了，根据一定的缓存策略进行淘汰。

LUR-K算法



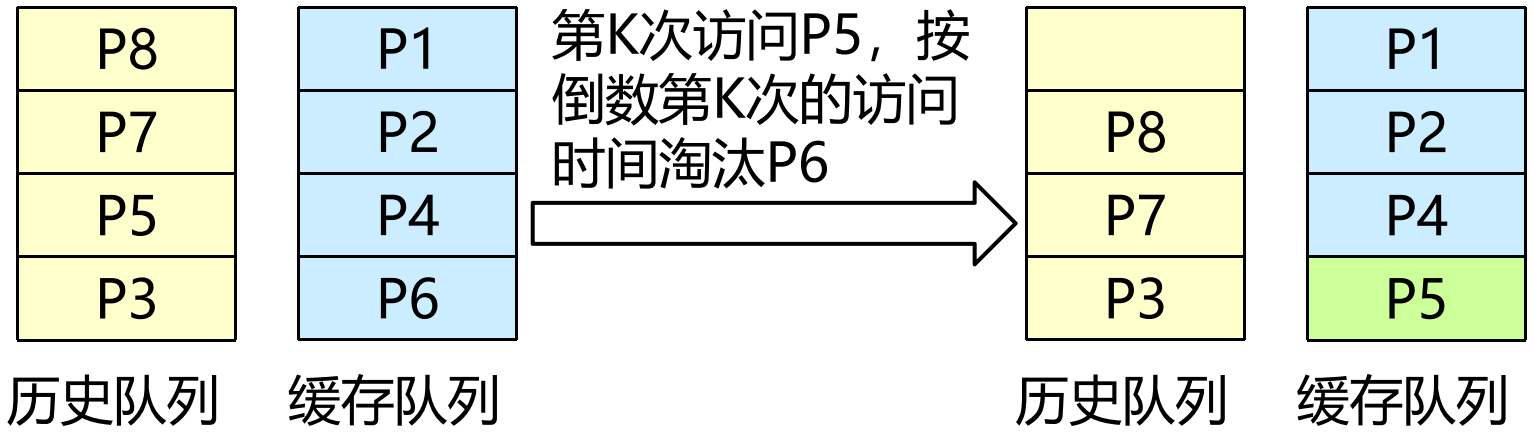
- 当历史队列中的某个页面第K次被访问时，该页面从历史队列中出栈，并存放至缓存队列。

LUR-K算法



- 缓存队列中的页面再次被访问时，更新缓存队列中该页面的位置。

LUR-K算法

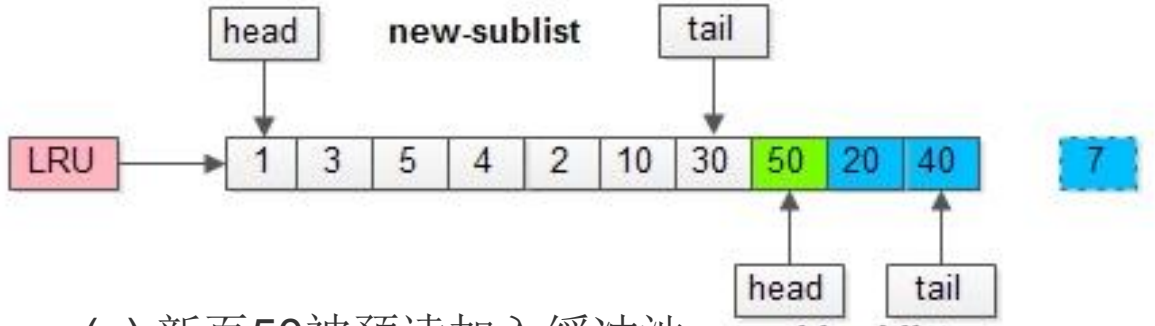


- 当缓存队列需要淘汰页面时，淘汰倒数第K次访问距离现在最久的页面。

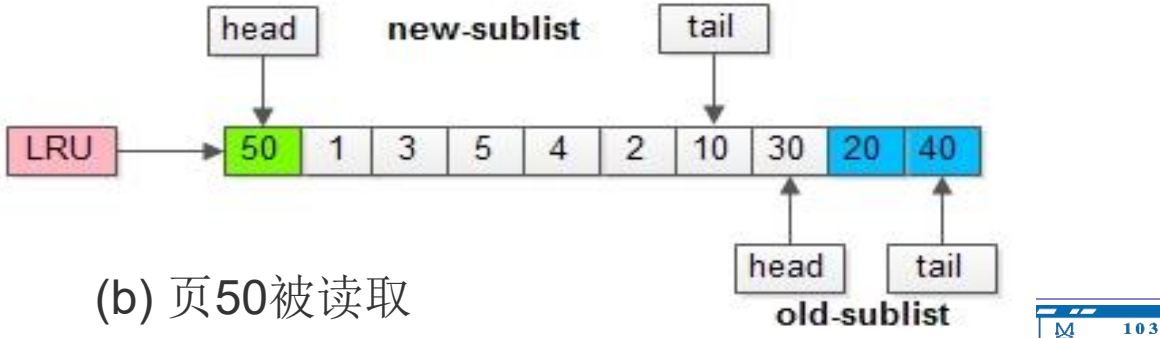
MySQL的LRU

- 要优化预读失效，思路是：
  （1）让预读失败的页，停留在缓冲池LRU里的时间尽可能短
  （2）让真正被读取的页，才挪到缓冲池LRU的头部
  以保证，真正被读取的热数据留在缓冲池里的时间尽可能长。

- 具体方法：
  （1）将LRU分为：新生代（new sublist）、老生代（old sublist）
  （2）新老生代收尾相连；
  （3）新页加入缓冲池时，只加入到老生代头部;
  (4) 若数据真正被读取（预读成功），才会加入到新生代的头部



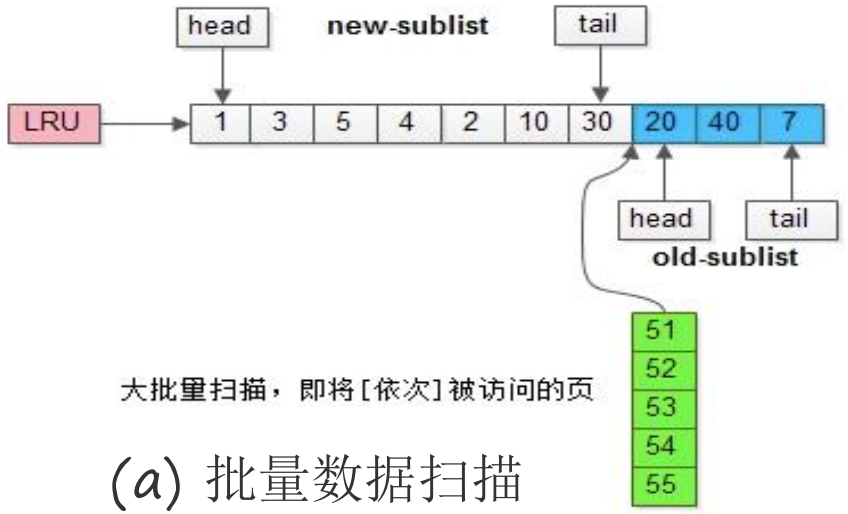
(a) 新页50被预读加入缓冲池



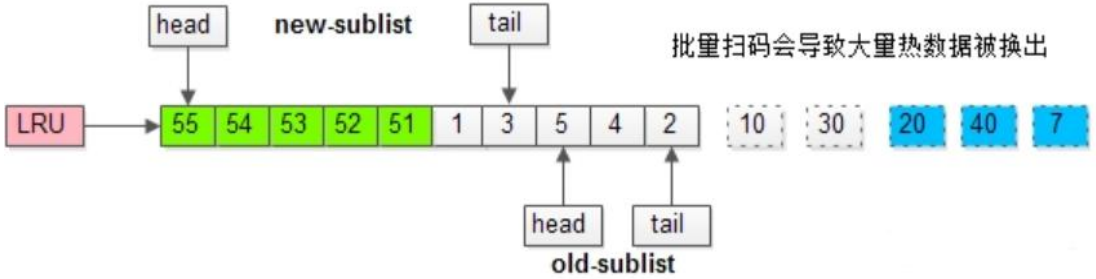
(b) 页50被读取

MySQL的LRU

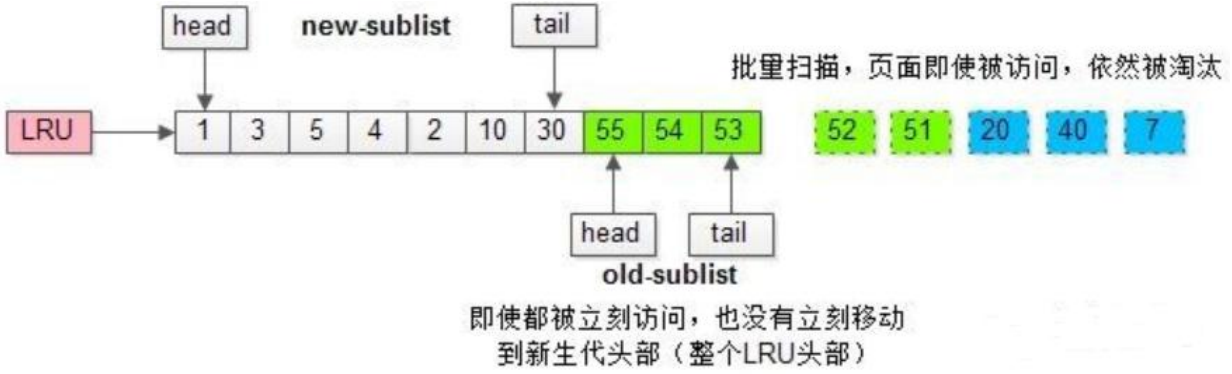
- 面对缓冲区污染问题：MySQL缓冲池加入了一个“老生代停留时间窗口”(old_blocks_time 参数)的机制



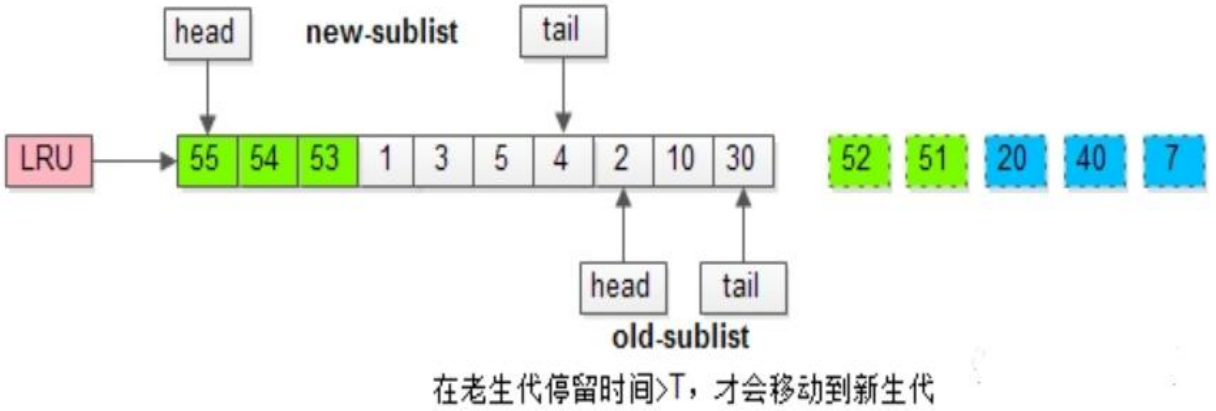
(a) 批量数据扫描



(b) 若无“老生代停留时间窗口”



(c) “老生代停留时间窗口”策略(1)(2)



淘汰策略的其他优化思路

□局部化

DBMS追踪每个查询或事务的页面访问轨迹，并在此局部范围内选择要淘汰的页面，这样可以让一个查询可能带来的缓存污染最小化。

例：PostgreSQL为每个查询维护一个局部环形缓冲。

□优先级提示

事务了解查询执行期间所访问的每个页面的上下文，并根据页面的上下文提示缓冲池管理器该页面是否重要，从而影响淘汰页面的选择。

脏页的处理

在淘汰页面时，对于脏页可以有两种情况：

- 快速：优先淘汰非脏页面（可能将未来不需要的脏页留在缓冲池）；
- 慢速：将脏页写回磁盘后再将其淘汰（这将降低替换页面的速度）。

另一种处理方法是**后台写**：

DBMS定期遍历页表并将脏页写入磁盘，避免在淘汰页面时才执行页面写出操作。

8.2.5 缓冲池的优化

多缓冲池

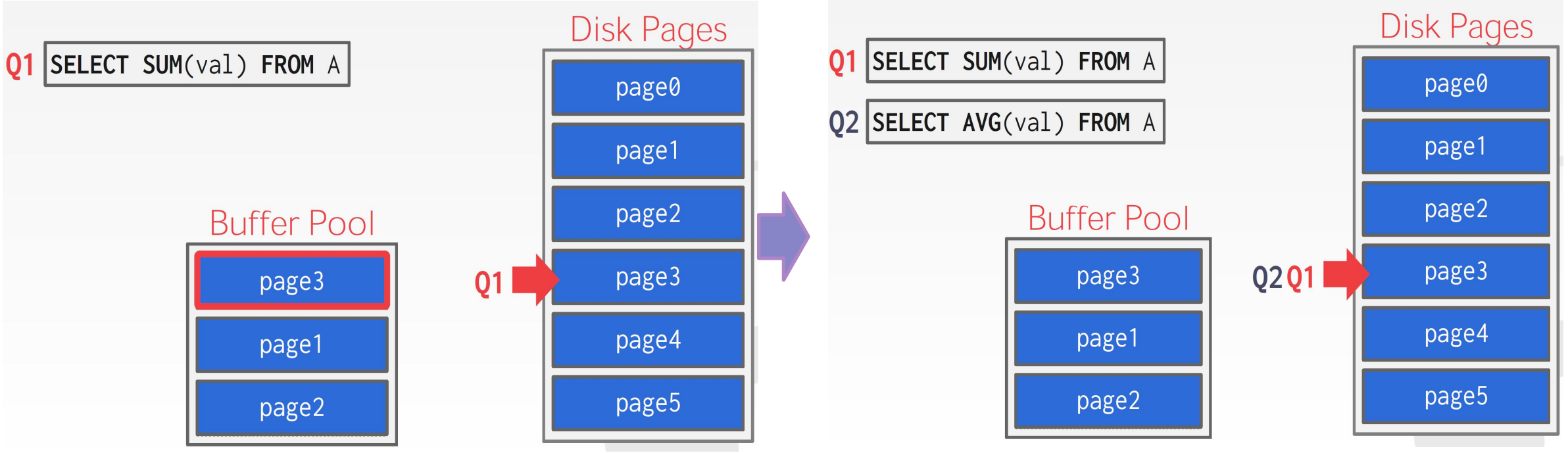
- DBMS维护多个用于不同目的的缓冲池。
- 可以是：多个缓冲池实例、每个数据库使用一个缓冲池每种页面类型使用一个缓冲池。
- 各缓冲池可以采用量身定制的管理策略。

预取

- 在处理第一组页面时，DBMS预先将第二组页面读取到缓冲池中。
- 这种方法通常在顺序访问多个页面时使用。

8.2.5 缓冲池的优化

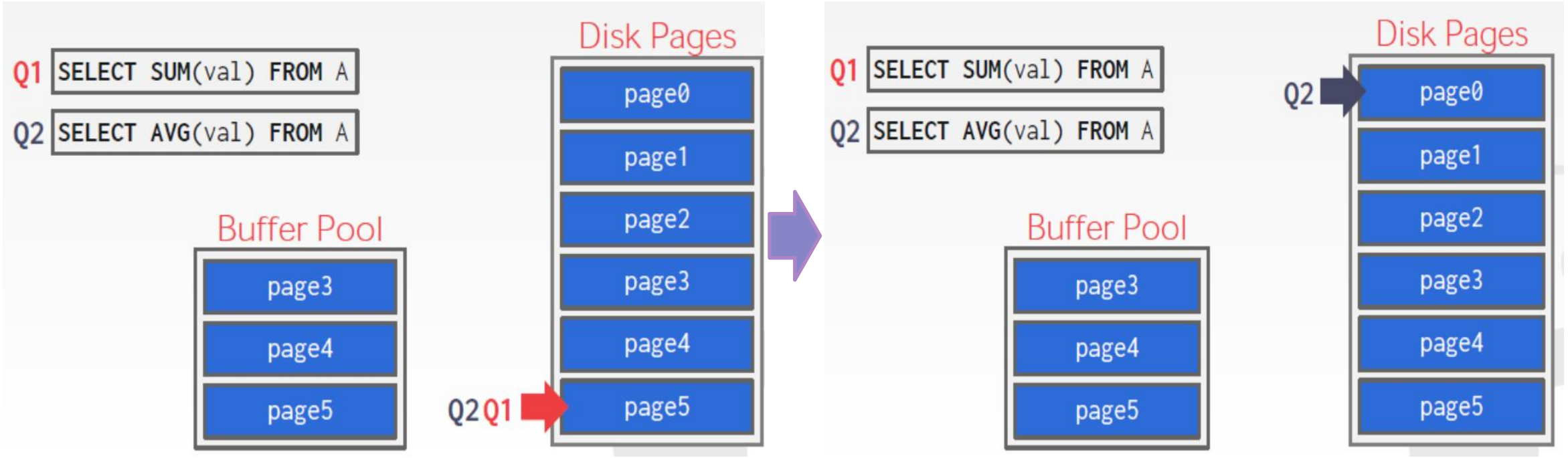
- 扫描共享: 某些查询结果可公用
  - 将多个查询附加到一个游标上，查询游标可以重用从磁盘读入的数据或操作符的计算结果，减少重复访问的次数。



8.2.5 缓冲池的优化

扫描共享

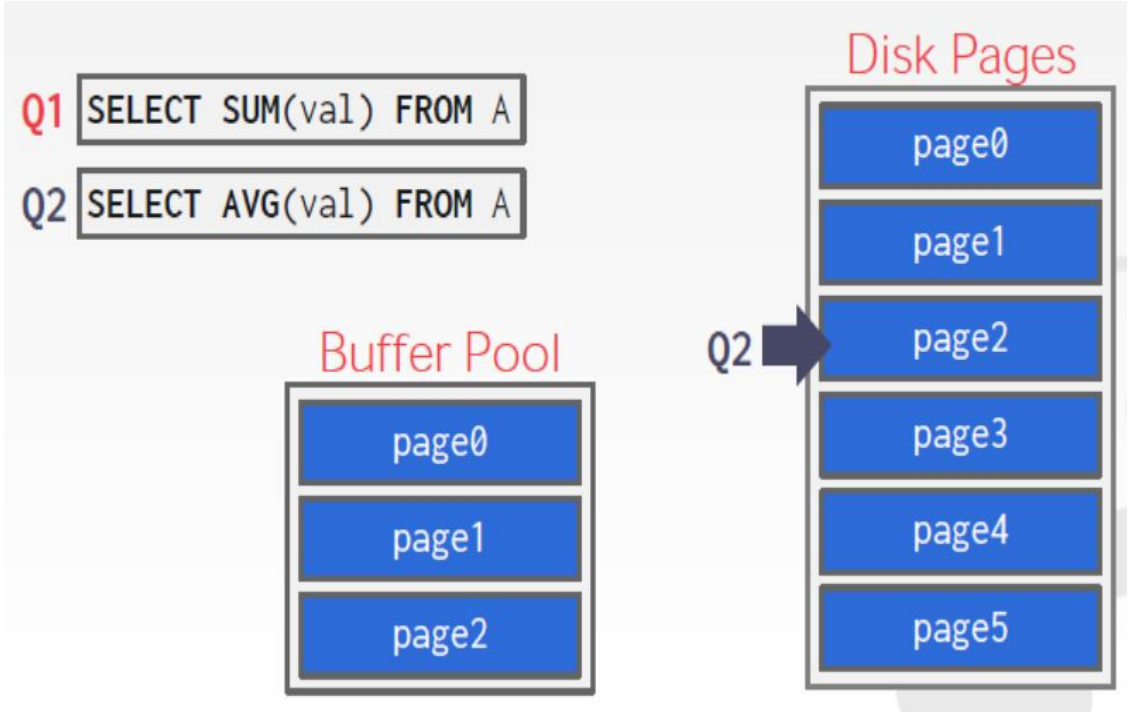
- 将多个查询附加到一个游标上，查询游标可以重用从磁盘读入的数据或操作符的计算结果，减少重复访问的次数。



8.2.5 缓冲池的优化

扫描共享

- 将多个查询附加到一个游标上，减少重复访问的次数。



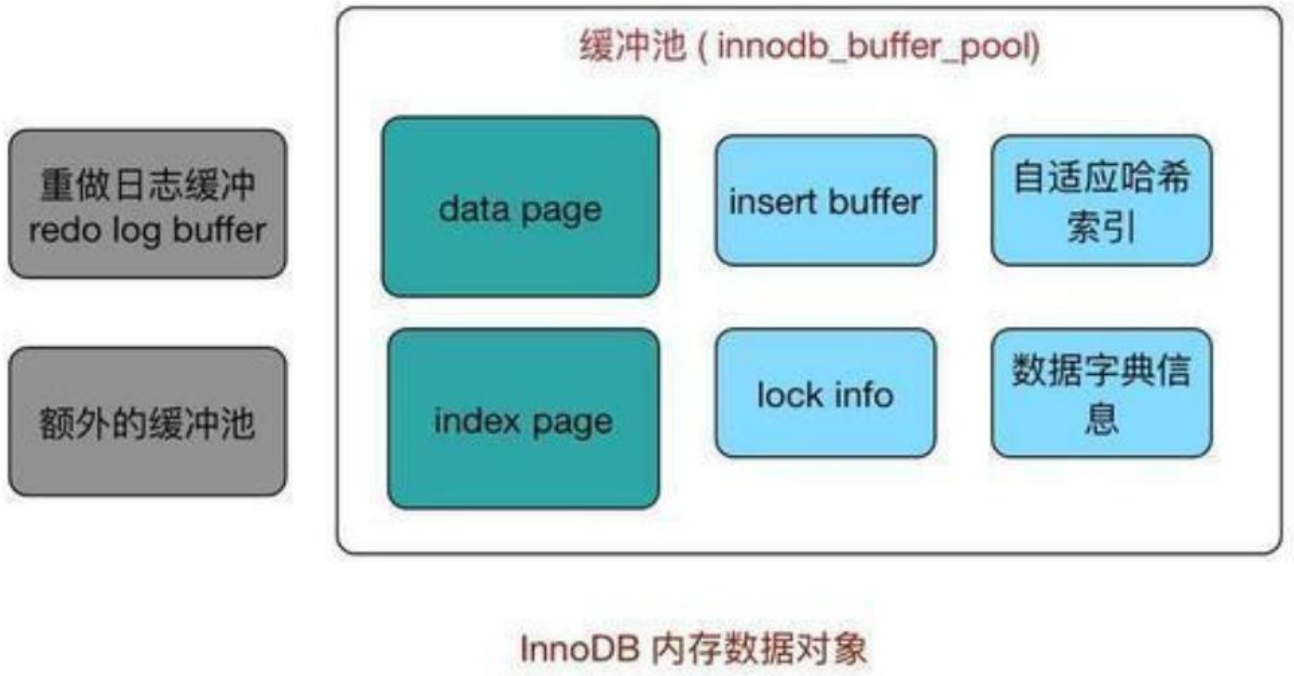
8.2.5 缓冲池的优化

缓冲池旁路

- 为了避免开销，顺序扫描操作符不会将获取的页存储在缓冲池中，而是使用正在运行的查询的本地内存。
- 如果操作符需要读取磁盘上连续的大量页序列，那么这种方法可以很好地工作。
- 缓冲池旁路也可以用于临时数据，如排序、连接。

8.2.6 其他专用缓冲池

- 其他缓冲池:
  - Sorting + Join Buffers
  - Query Caches
  - Maintenance Buffers
  - Log Buffers
  - Dictionary Caches



InnoDB 内存数据对象