实现虚拟存储器的硬件支持以及在请求调页管理方式下访问数据的各种情况，并分析系统消耗

----第十四组

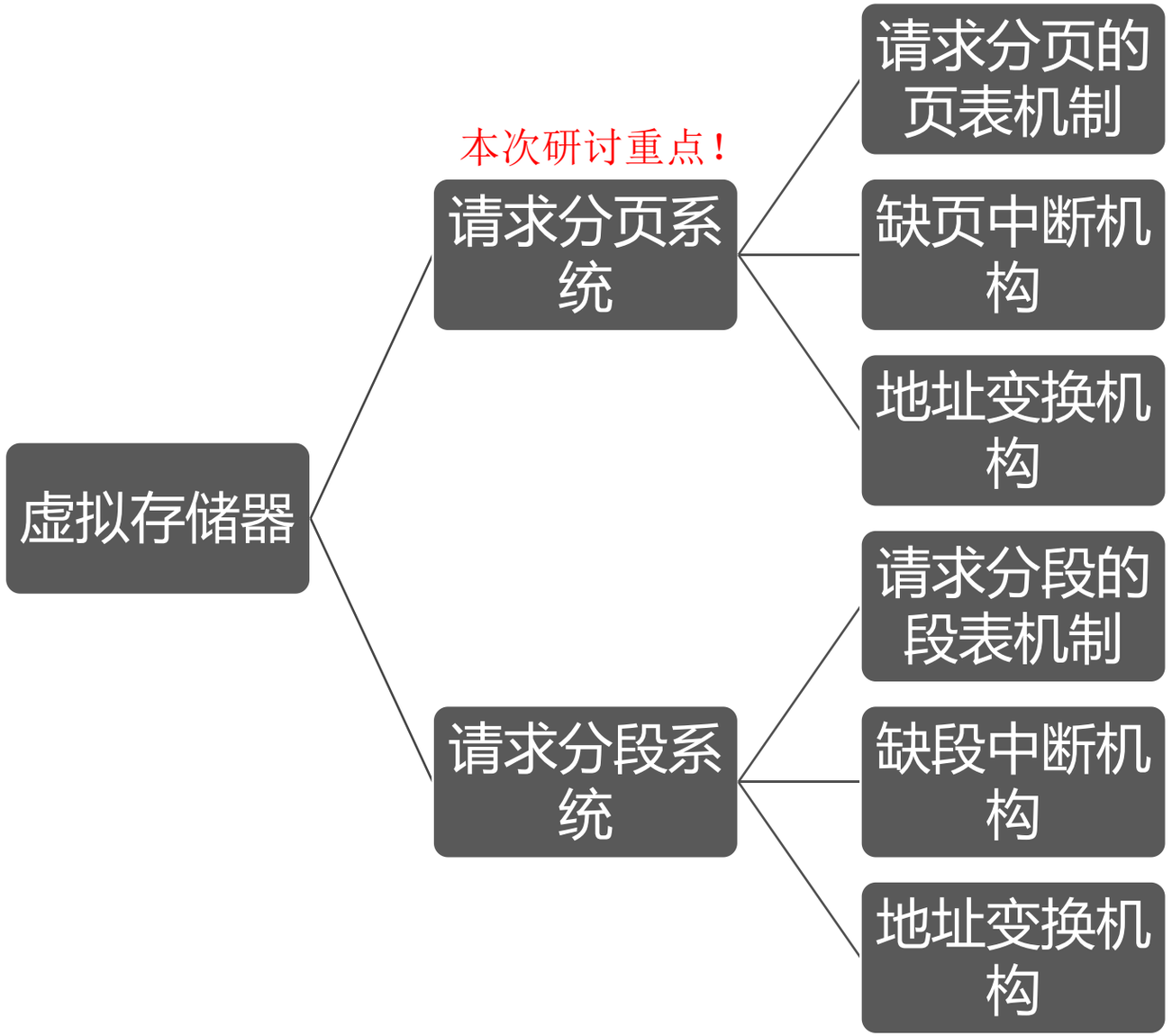
目录

CONTENTS

01

硬件支持

硬件支持


虚拟存储器
请求分页系统
本次研讨重点！
请求分页的页表机制
缺页中断机构
地址变换机构
请求分段系统
请求分段的段表机制
缺段中断机构
地址变换机构

请求页表机制

记录在一段时间内被访问的次数，或记录上次访问的时间，供置换算法选择换出页面时参考。

页面在外存上的地址。

指示该页是否已调入内存。

页面在调入内存后是否被修改过，若已被修改，必须重写到外存上。

1. 为了实现“请求调页”，操作系统需要知道每个页面是否已经调入内存；如果还没调入，那么也需要知道该页面在外存中存放的位置。

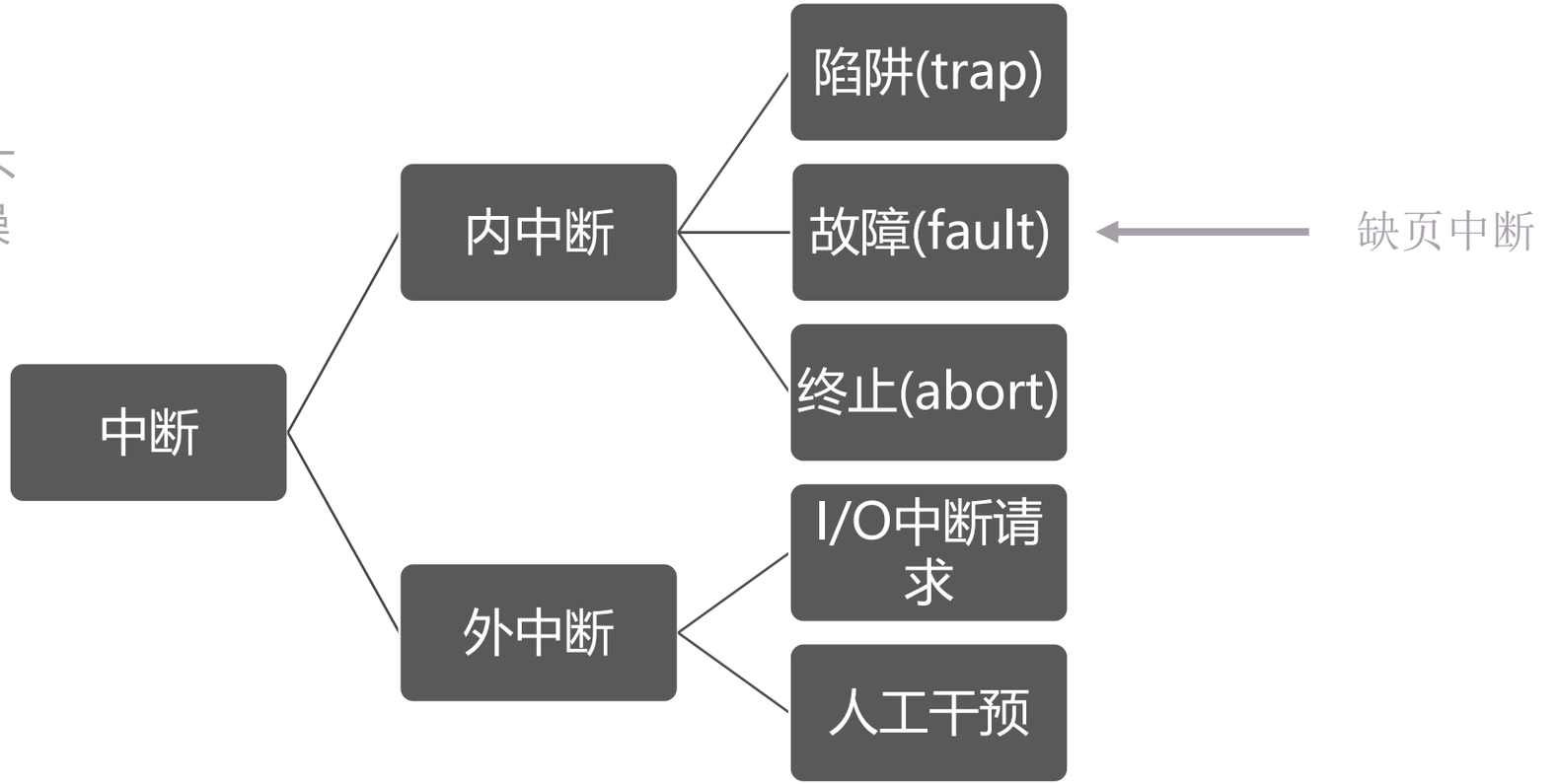
2. 当内存空间不够时，要实现“页面置换”，操作系统需要通过某些指标来决定换出哪个页面。

缺页中断机构

1. 在请求分页系统中，每当要访问的页面不在内存时，便产生一个缺页中断，然后由操作系统的缺页中断处理程序处理中断。

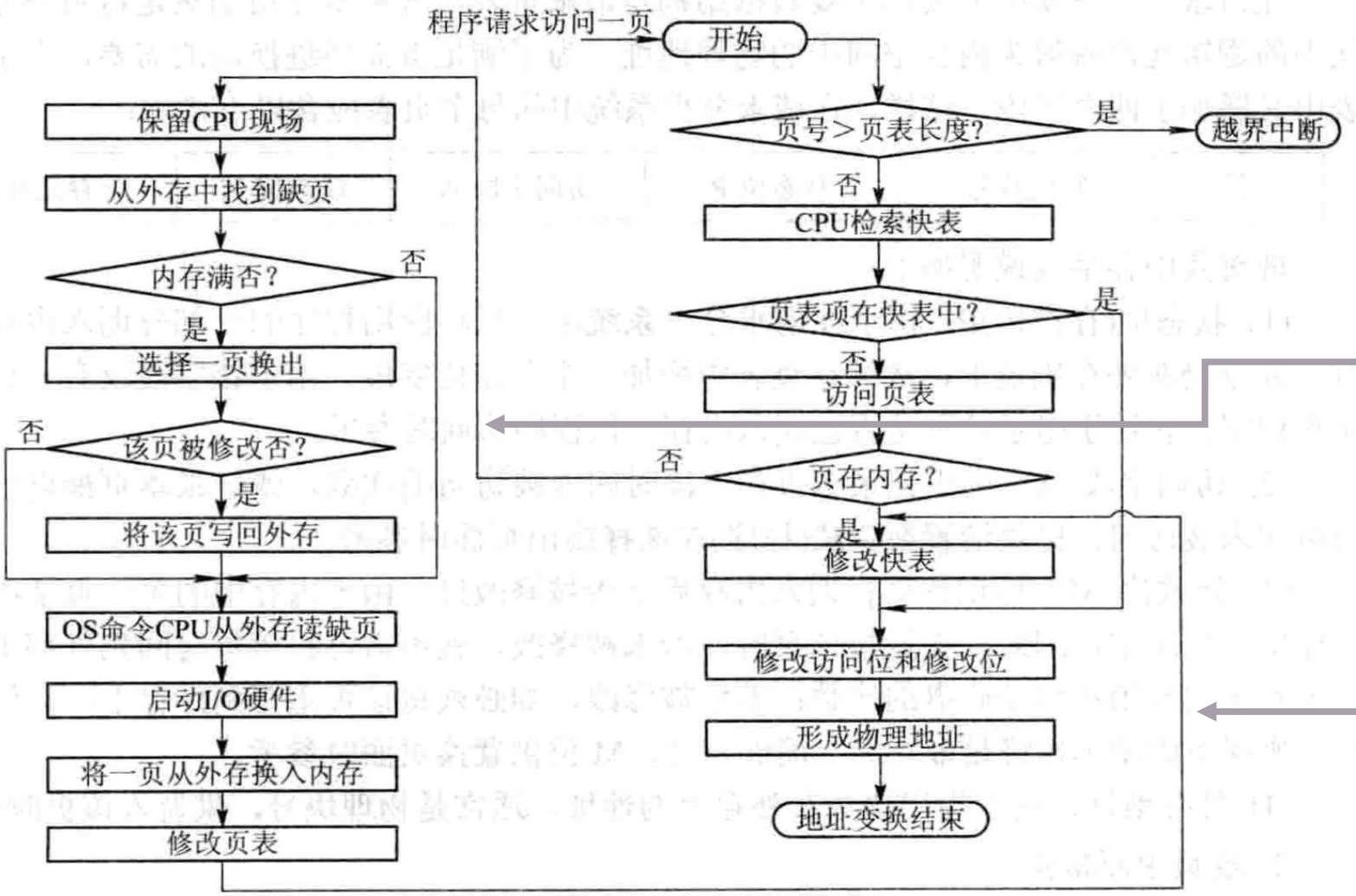
2. 此时缺页的进程阻塞，放入阻塞队列，调页完成后再将其唤醒，放回就绪队列。

3. 缺页中断处理中，需要将目标页面调入内存，有必要时还要换出页面。



与一般中断的区别：
①在指令执行期间产生和处理中断信号，这样能及时将所缺页面调入内存，而不是在一条指令执行完后，才去检查是否有中断请求。
②一条指令在执行期间可能产生多次缺页中断。

地址变换机构



若对应页面未调入内存，则产生缺页中断，之后由操作系统的缺页中断处理程序进行处理。

由于快表中的页面一定是在内存中的，所以若某个页面从外存调入内存，快表中的相应表项要进行修改。

02

情况分析

下面是一个进程实例：

某虚拟存储器的用户空间共有32 个页面，一个页面大小为1KB,内存大小为16KB。该进程长度共6页。（本例中的置换策略均采用FIFO）

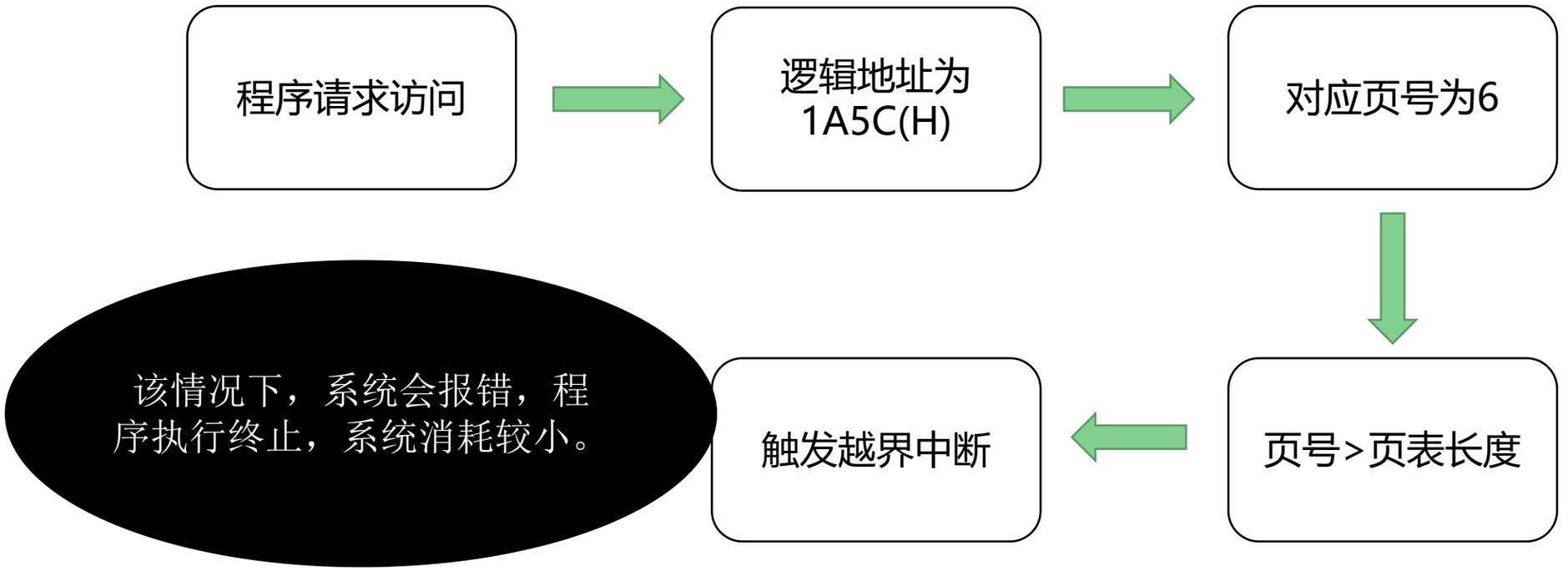
快表：

页号	物理块号	其他
0	5	…
1	8	…

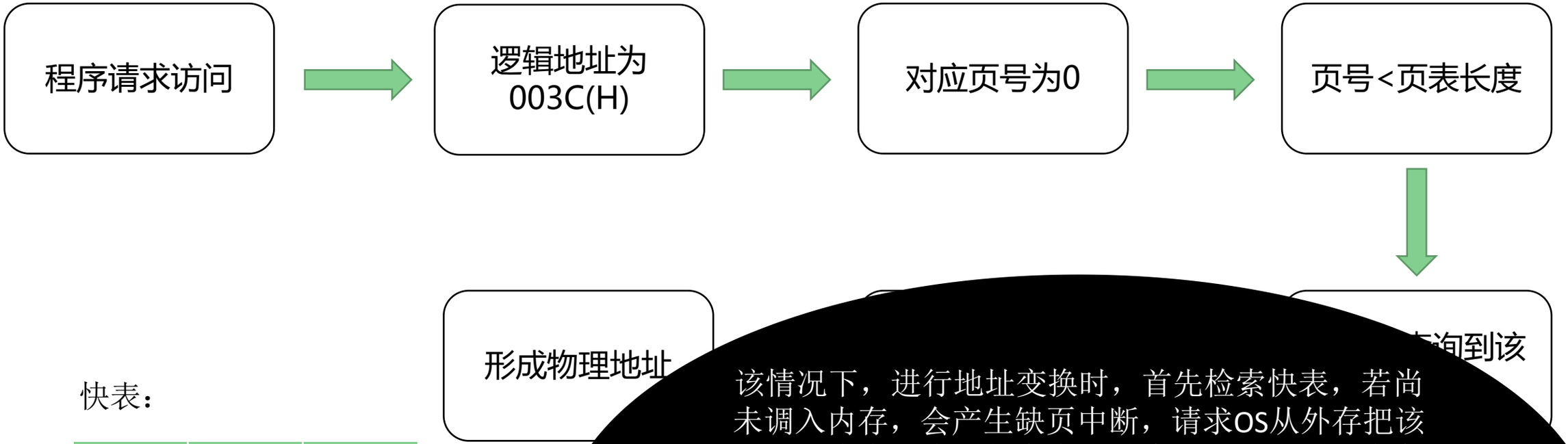
页表：

页号	物理块号	其他
0	5	…
1	8	…

1. 越界中断：



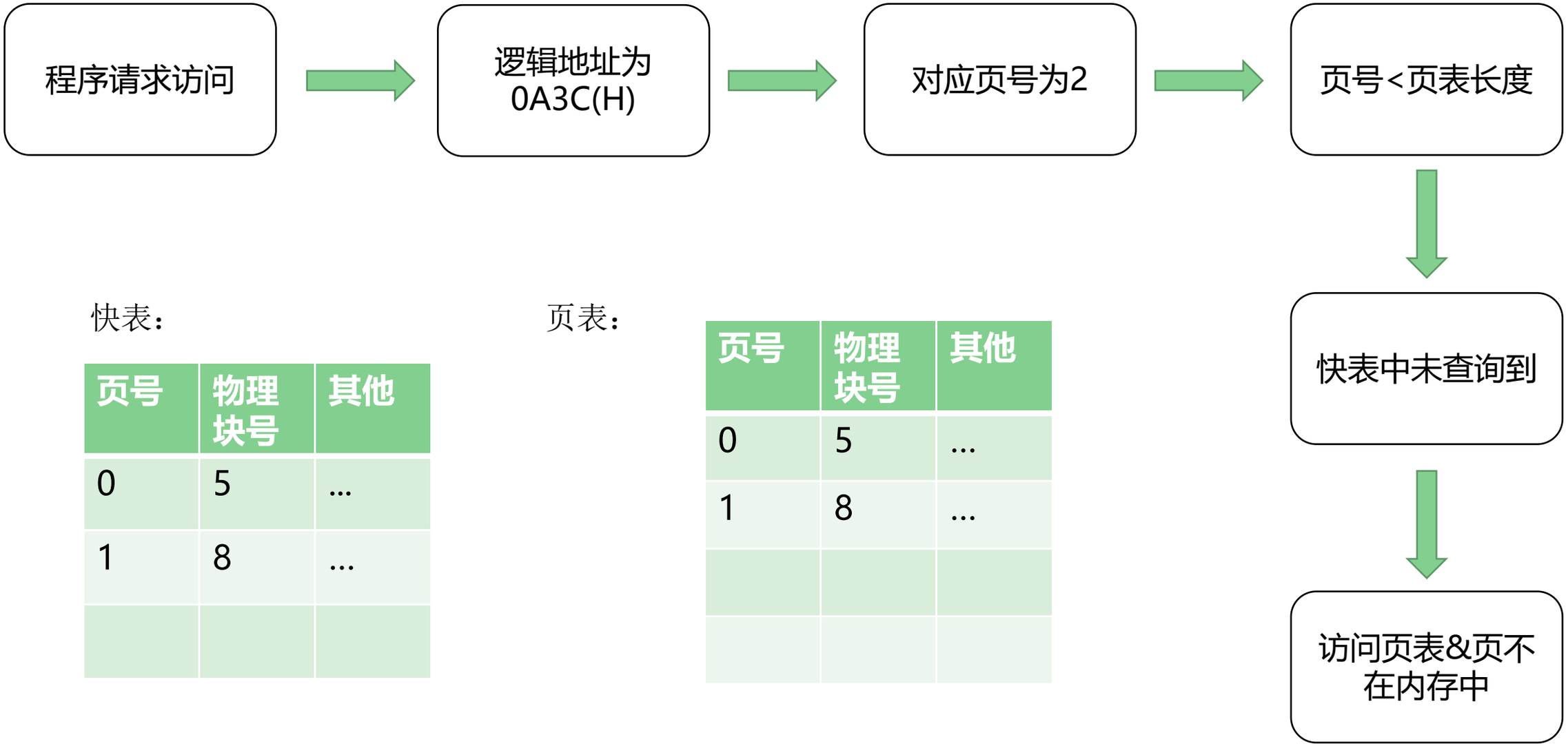
2.在快表中



快表：

页号	物理块号	其他
0	5	…
1	8	…

3. 不在页表和快表中，且页表和快表未满



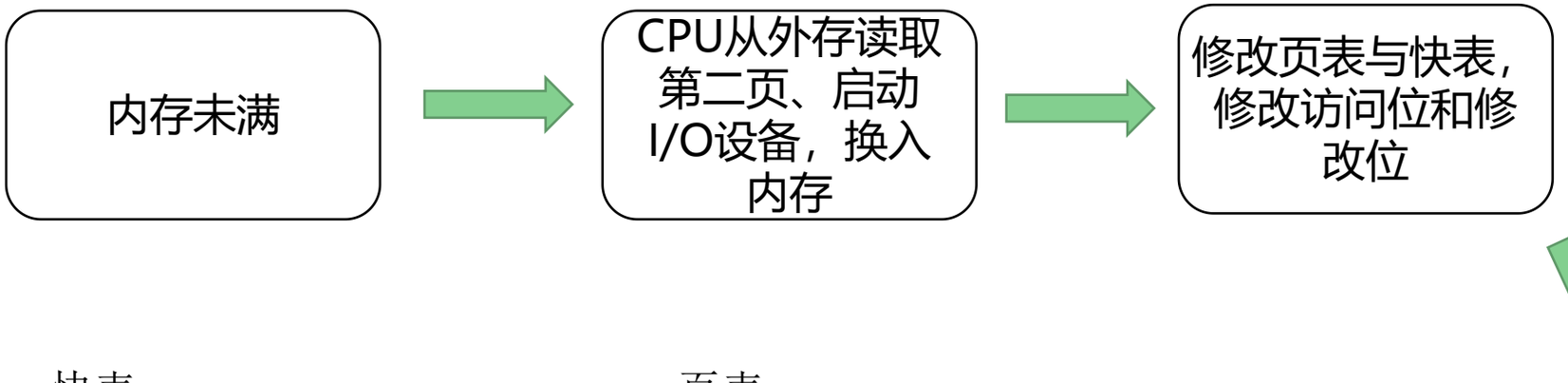
快表：

页号	物理块号	其他
0	5	…
1	8	…

页表：

页号	物理块号	其他
0	5	…
1	8	…

3. 不在页表和快表中，
且页表和快表未满



快表：

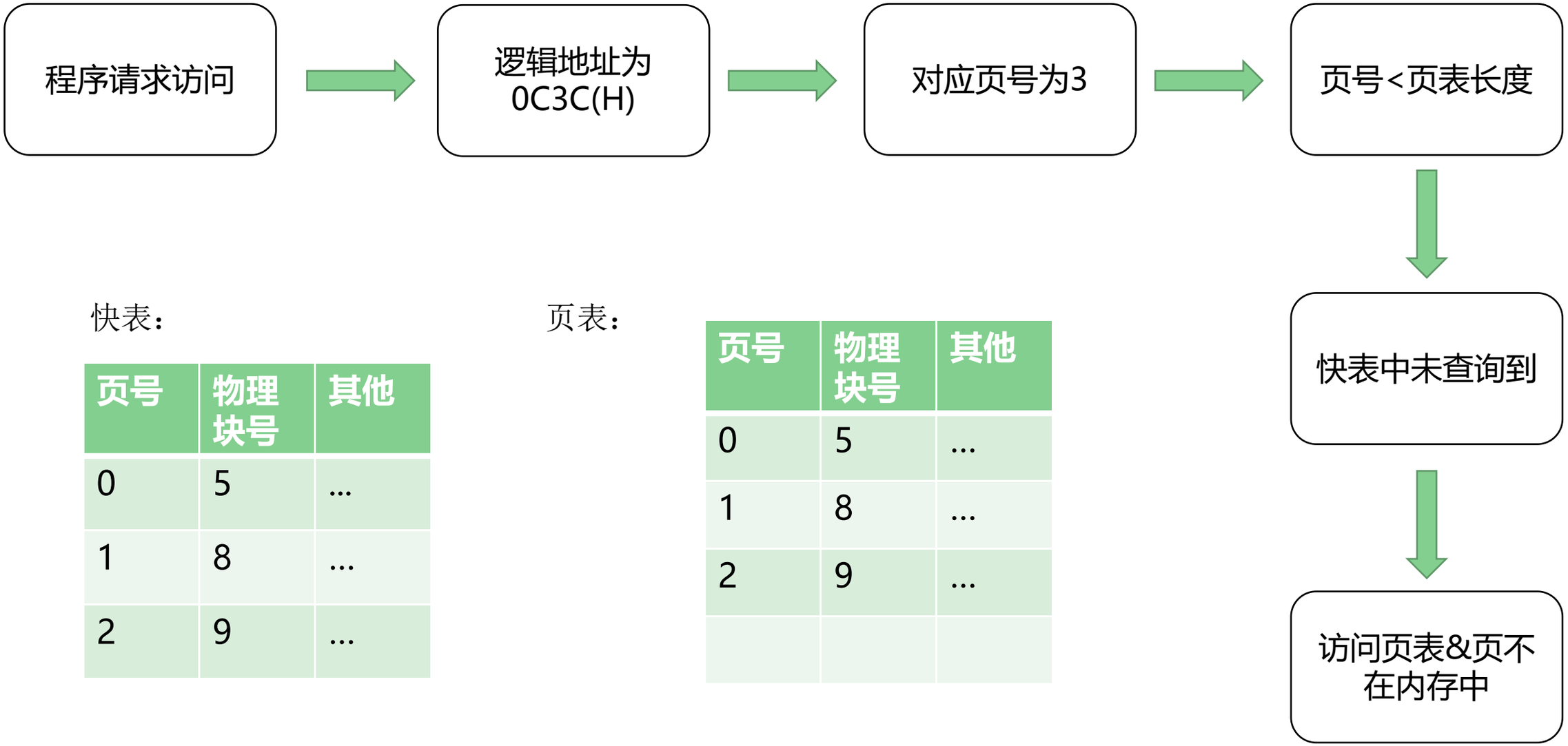
页号	物理块号	其他
0	5	…
1	8	…
2	9	…

页表：

页号	物理块号	其他
0	5	…
1	8	
2	9	

该情况下，产生缺页中断，需要请求OS从外存把该页调入内存，并且需要写入快表，从而造成系统的消耗。此时缺页的进程阻塞，放入阻塞队列，调页完成后再将其唤醒，放回就绪队列。

4. 不在页表和快表中，且页表未满



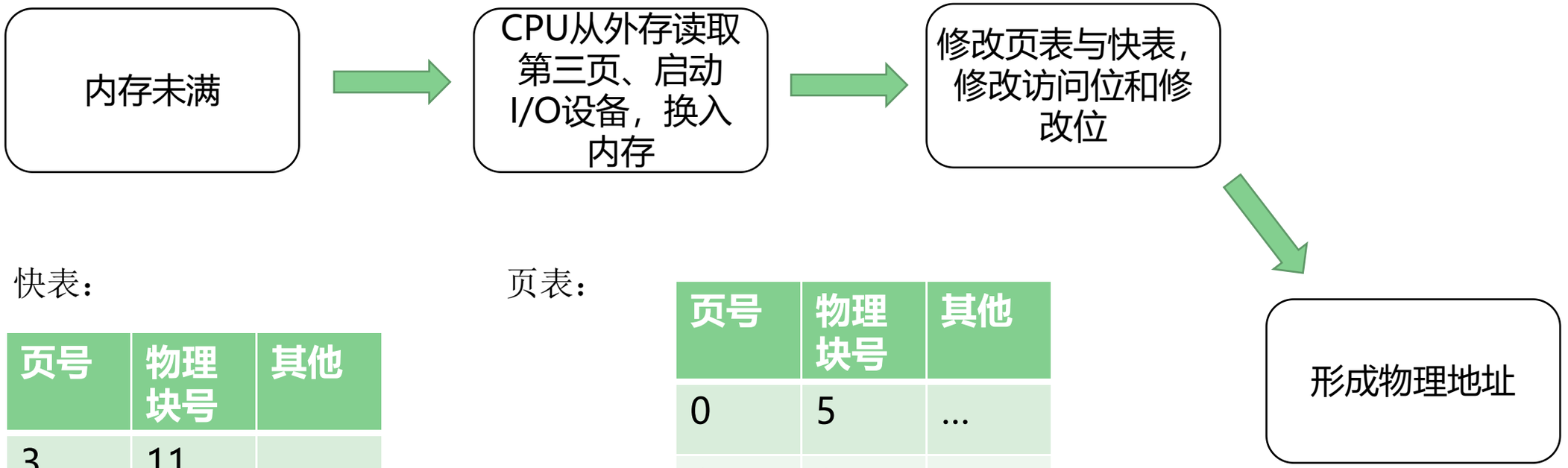
快表：

页号	物理块号	其他
0	5	…
1	8	…
2	9	…

页表：

页号	物理块号	其他
0	5	…
1	8	…
2	9	…

4. 不在页表和快表中，
且页表未满



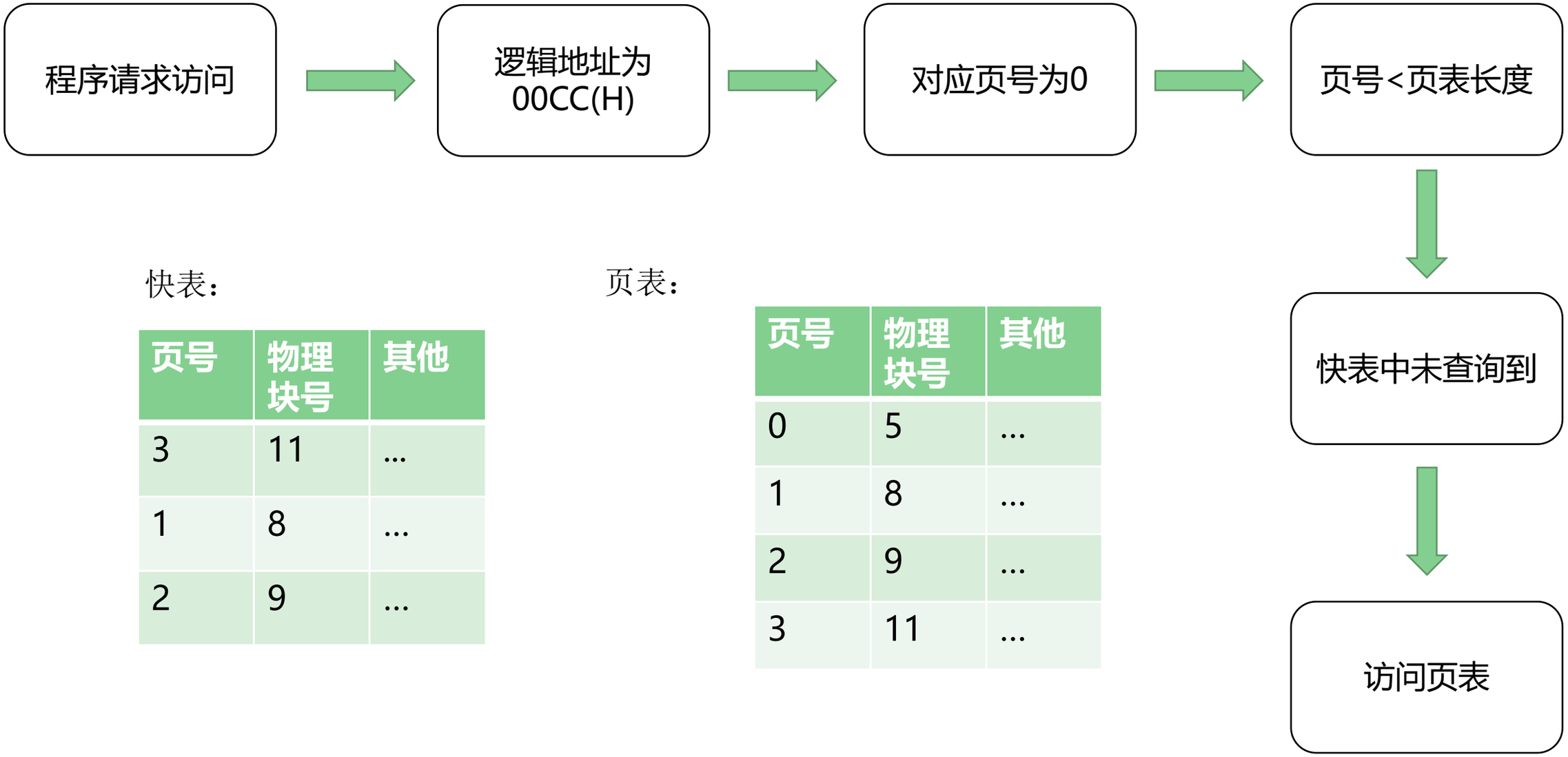
快表：

页号	物理块号	其他
3	11	…
1	8	…
2	9	…

页表：

页号	物理块号	其他
0	5	…
1	8	…
2	9	…
3	11	…

5.不在快表中但在页表中



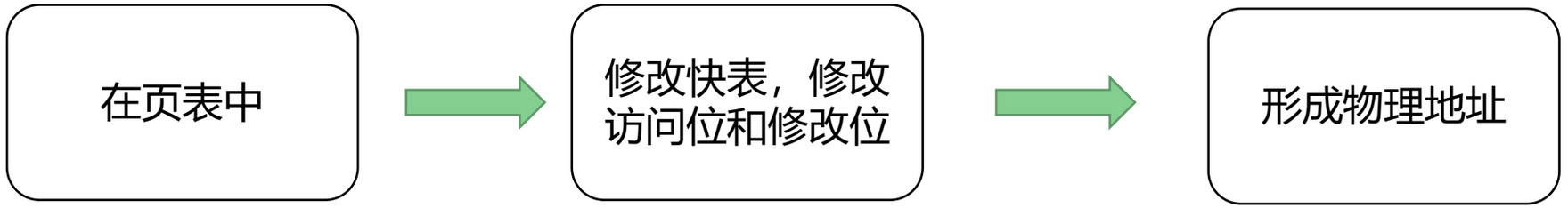
快表：

页号	物理块号	其他
3	11	…
1	8	…
2	9	…

页表：

页号	物理块号	其他
0	5	…
1	8	…
2	9	…
3	11	…

5.不在快表中但在页表中



快表：

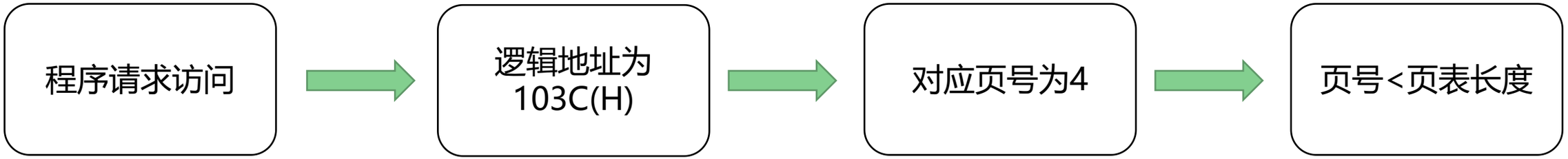
页号	物理块号	其他
3	11	…
0	5	…
2	9	…

页表：

页号	物理块号	其他
0	5	
1	8	
2	9	
3	11	

该情况下，程序访问的数据在内存中已经存在，则程序可以直接访问该数据，并修改快表中的表项，不需要执行页面置换算法和数据调入调出操作，因此系统消耗会比较小。

6. 不在页表和快表中，
且内存已满

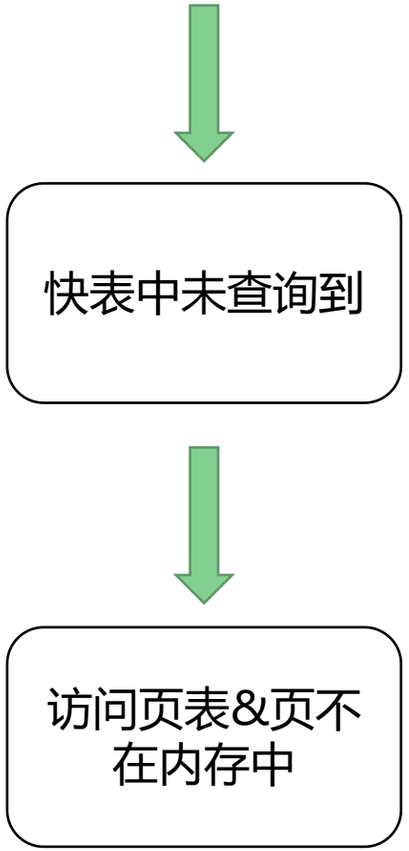


快表：

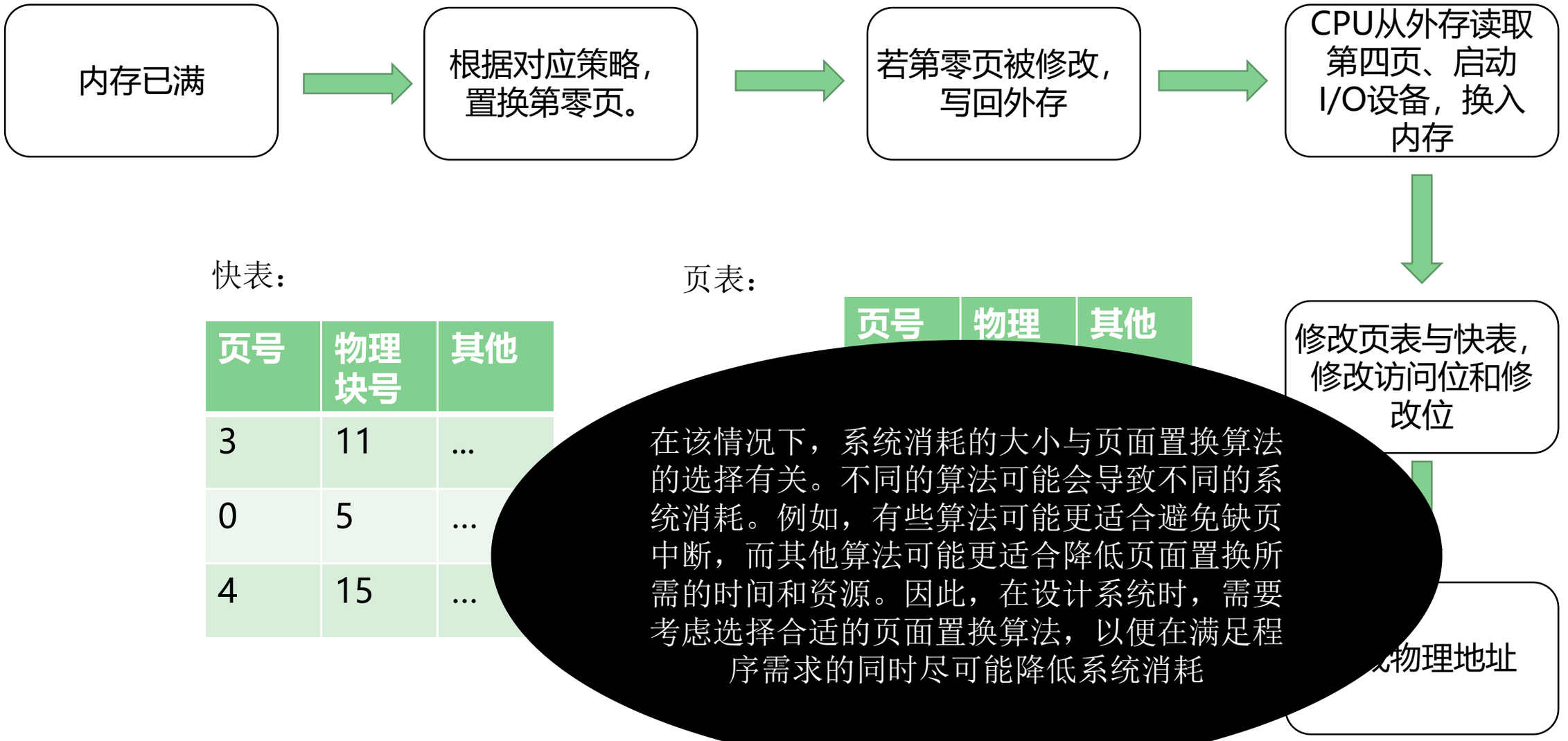
页号	物理块号	其他
3	11	…
0	5	…
2	9	…

页表：

页号	物理块号	其他
0	5	…
1	8	…
2	9	…
3	11	…



6. 不在页表和快表中，且内存已满



快表：

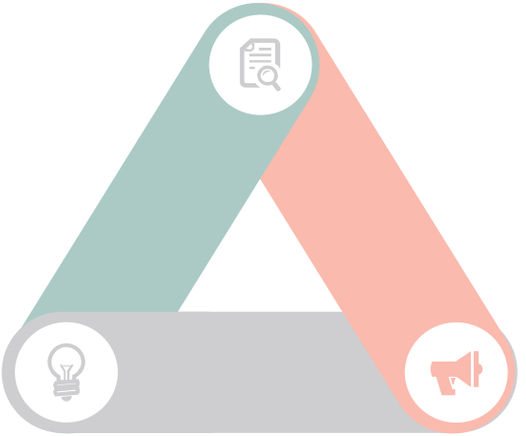
页号	物理块号	其他
3	11	…
0	5	…
4	15	…

页表：

页号	物理	其他

在该情况下，系统消耗的大小与页面置换算法的选择有关。不同的算法可能会导致不同的系统消耗。例如，有些算法可能更适合避免缺页中断，而其他算法可能更适合降低页面置换所需的时间和资源。因此，在设计系统时，需要考虑选择合适的页面置换算法，以便在满足程序需求的同时尽可能降低系统消耗

系统消耗



总体来说：
1. 页在内存或在快表时，系统消耗较小。
2. 页不在内存中或内存满时，系统消耗较大。
①若该页表已登记在快表中，可快速查找，并提高指令的执行速度，减少了系统消耗。

②如果程序访问的数据在内存中已经存在，则程序可以直接访问该数据，不需要执行页面置换算法和数据调入调出操作，因此系统消耗会比较小。

③如果访问页表后，该页尚未调入内存，产生缺页中断。缺页中断时，需要请求OS从外存把该页调入内存，从而造成系统的消耗。

④ 缺页中断时没有空闲块的话，则增大了系统消耗。若该页面在内存期间被修改过，则要将其写回外存，这样显然增大了系统消耗。

⑤系统消耗的大小与页面置换算法的选择有关。不同的算法可能会导致不同的系统消耗。在设计系统时，需要考虑选择合适的页面置换算法，以便在满足程序需求的同时尽可能降低系统消耗。

感谢您的观看