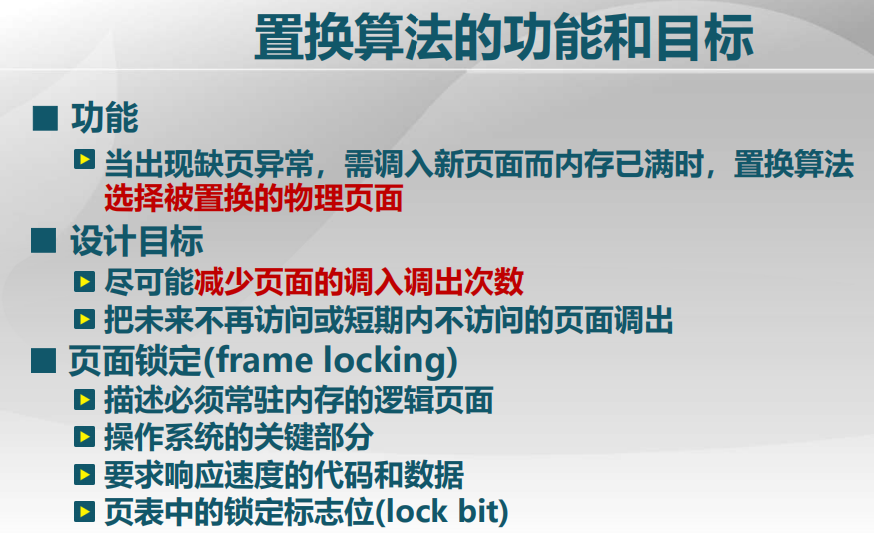
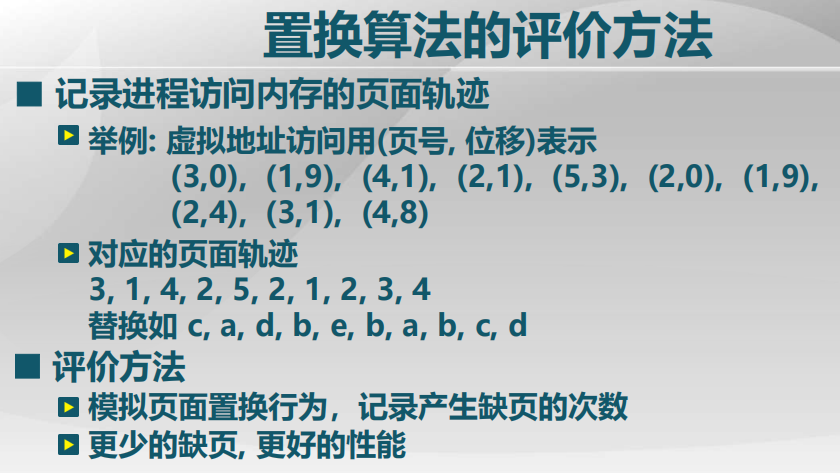
**第一讲 页面置换算法的概念**

本章将介绍两种类型的置换算法：①局部置换算法：假定分配给一个进程的物理页面数已经定了之后，然后去选择到底哪个被置换。即置换页面的选择范围仅限于当前进程占用的物理页面内②全局置换算法：不区分置换的页面到底属于哪个进程，这样的话，隐含着后边每个进程分配的物理页面数会做调整这样一种情况。即置换页面的选择范围是所有可换出的物理页面。

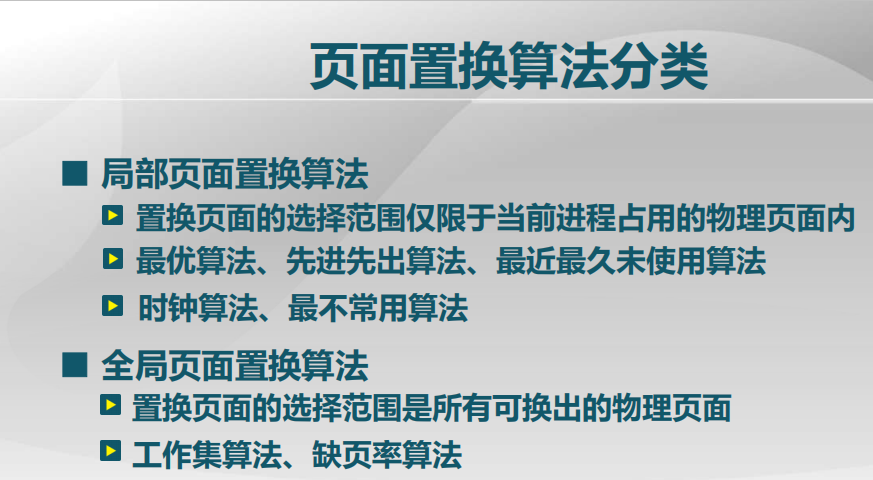


置换算法选择被置换的物理页面，将其从内存中淘汰。设计目标中，最大的问题是，如何知道某页面未来不再访问或短期内不访问，即如何对未来做一个预测？

页面锁定是虚拟存储中的一个概念，即在进行置换的时候，有一些页面是不可以把它换到外存中的，如：描述必须常驻内存的逻辑页面、操作系统的关键部分、要求响应速度的代码和数据。如何做到页面锁定？答：在页表里的页表项加上相应的锁定标志位，有了这个标志，置换算法在运行的过程当中就不会把这个页面放到外存中。



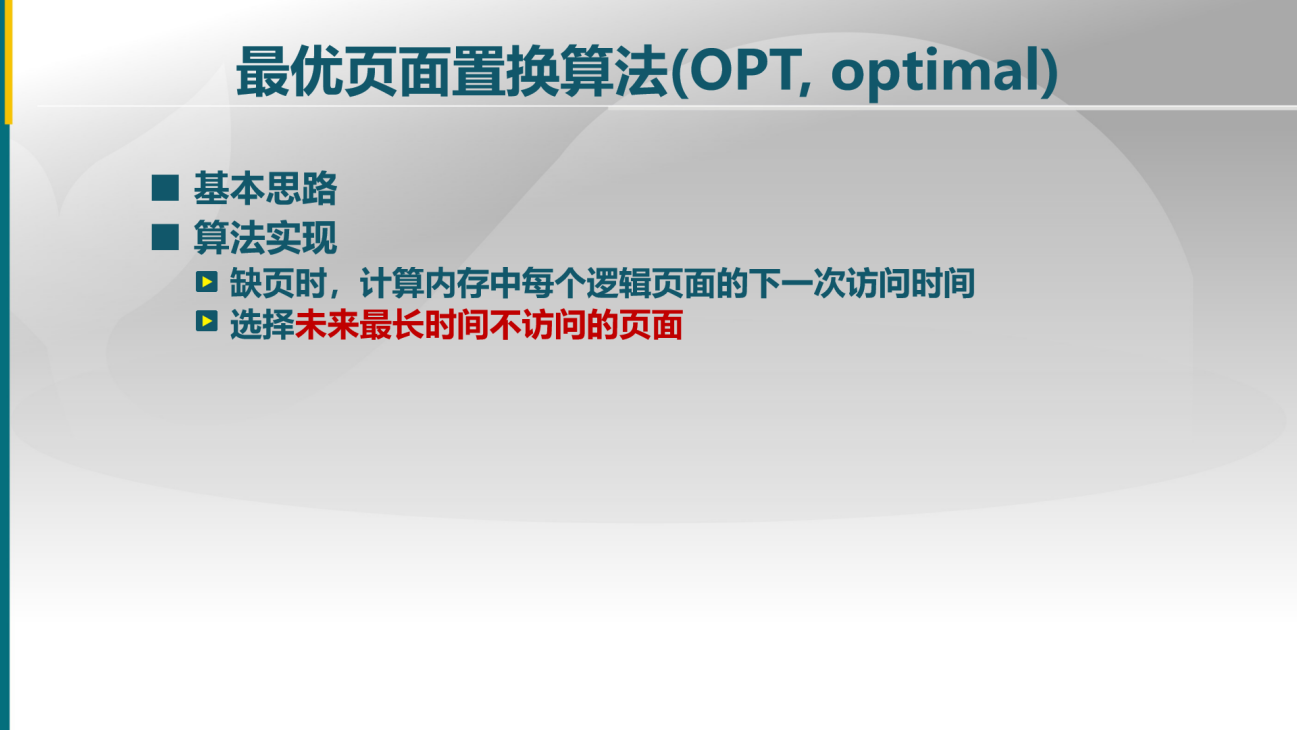
置换算法的评价是通过统计测试用例的形式给出的。记录测试用例即记录对应的页面轨迹，因此我们只对虚拟地址的页号一部分感兴趣。页面轨迹中的数字替换成字母，以避免混淆。置换算法的评价方法见上图。



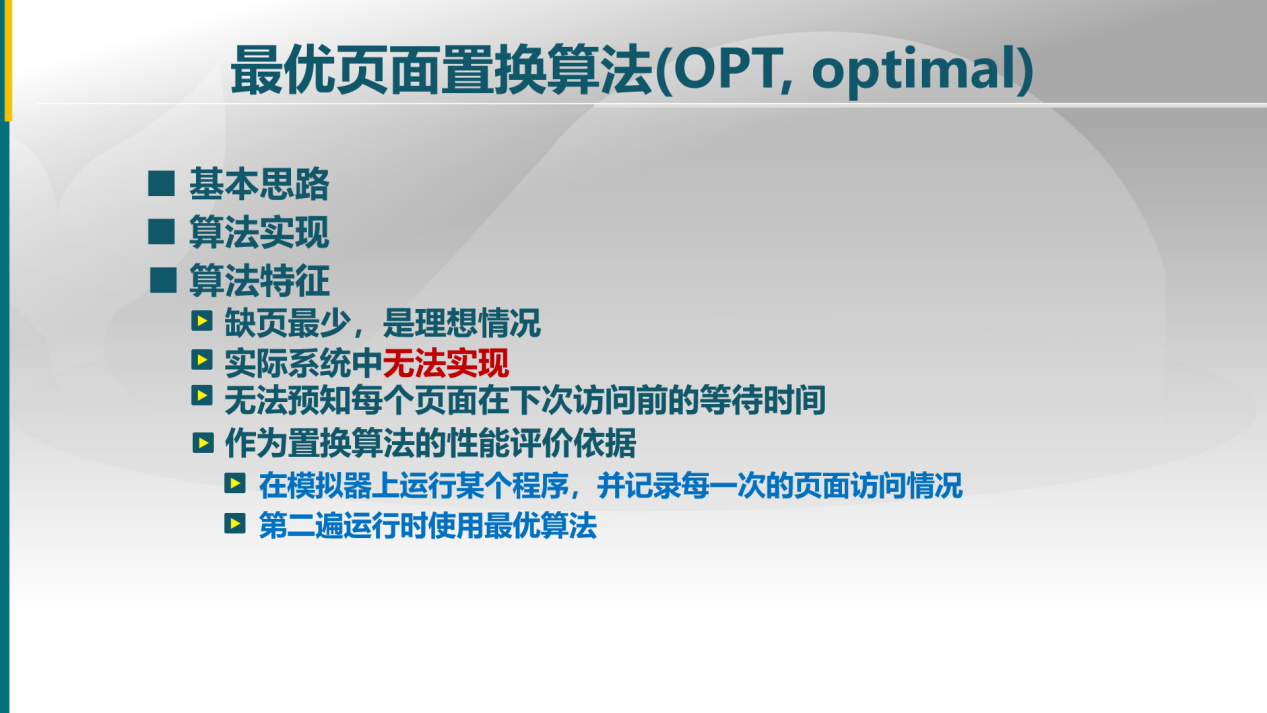
上图是页面置换算法的分类。局部页面置换算法分配给一个进程的物理页面数是确定的，我们在置换时，选择的范围仅限于当前进程。局部页面置换算法在置换的过程当中，每个进程分配的页面的总数是不会变化的。局部页面置换算法中，最优算法预测未来，实际系统中无法实现；先进先出不能反映实际的存储访问特征，性能不好；最近最久统计过去，复杂度较高，开销较大，这时有最近最久的两种近似算法：时钟算法和最不常用算法。

全局页面置换算法不关心当前换的页面属于哪个进程，置换页面的选择范围是所有可换出的物理页面。这隐含着，如果在置换时，要访问进程A的一个页面，被置换出去的可以是进程B的页面，那么这两个进程之间分配的(内存)物理页面数是可以变化的。

**第二讲 最优算法、先进先出算法和最近最久未使用算法**



注意，算法实现中：缺页时，计算内存中每个逻辑页面的下一次访问时间,这是无法实现的，因为我们无法预测未来。

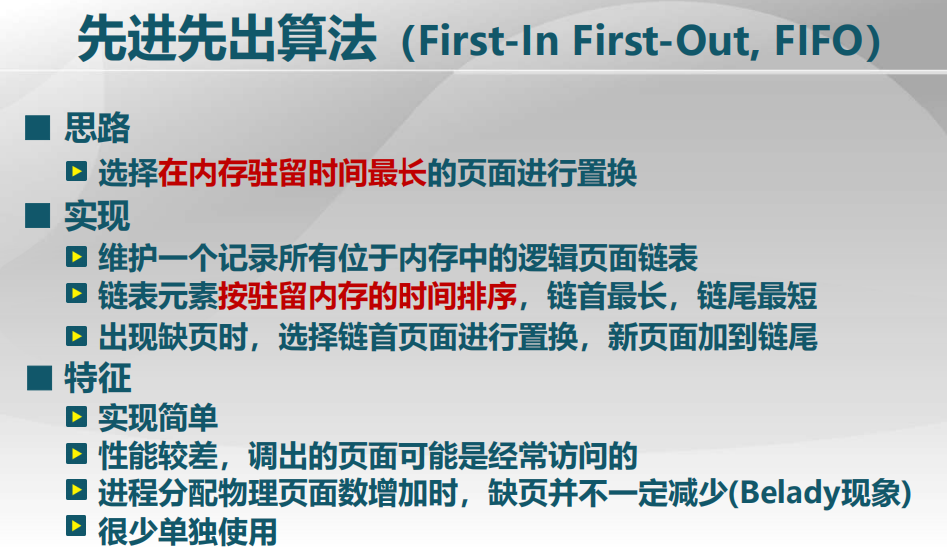


既然OPT(最优页面置换)算法无法实现，其意义何在？答：因为它是最理想情况，因此可作为置换算法的性能评价依据。

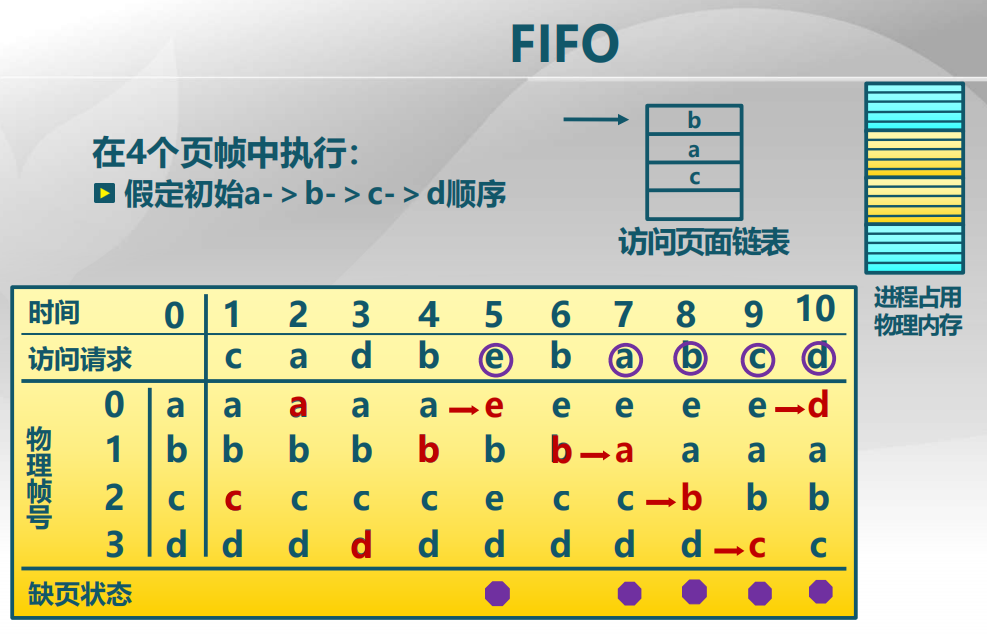


物理页面分配了4个页面。时间5发生缺页，而d在时间10才访问，因此d被换出，e被换入。在时间10，访问d，发生缺页，而这之后已经结束，没有未来了，这时将任何一页换出都可以。

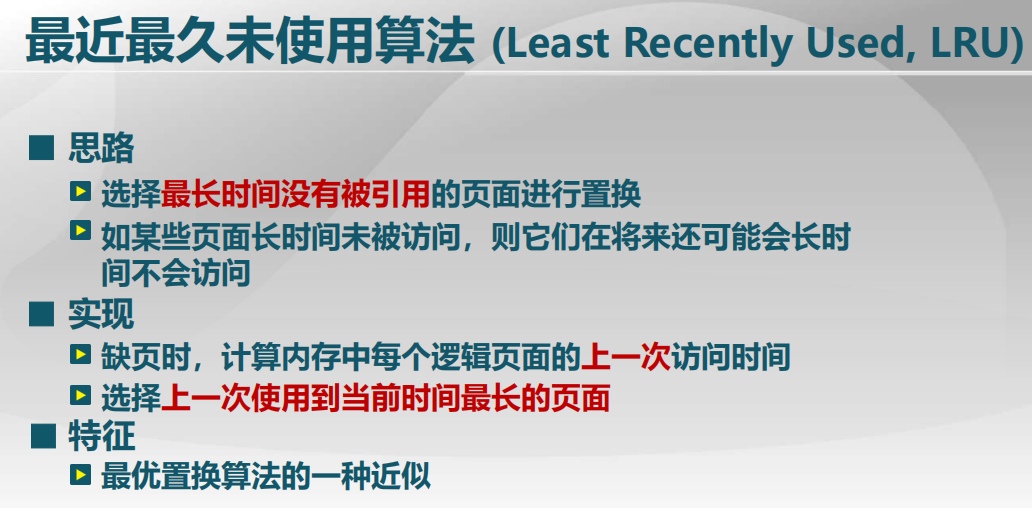
下面是先进先出算法：



例子：



对OPT这种不能实现的算法和FIFO这种过于简单的算法进行折中，得到最近最久未使用算法(LRU)：

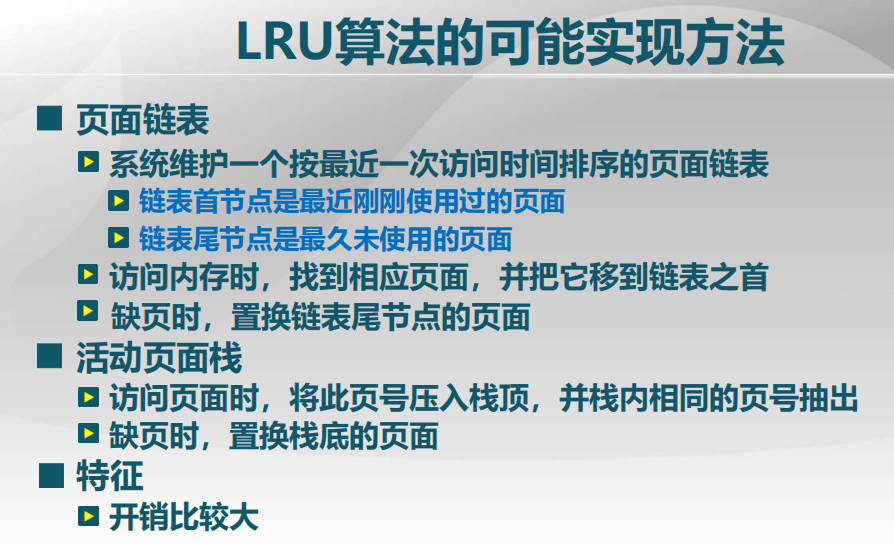


这种算法建立在假设：若某些页面(过去)长时间未被访问，则它们在将来还可能会长时间不会访问。这种算法实际上依然足够复杂，以至于其实施是困难的。

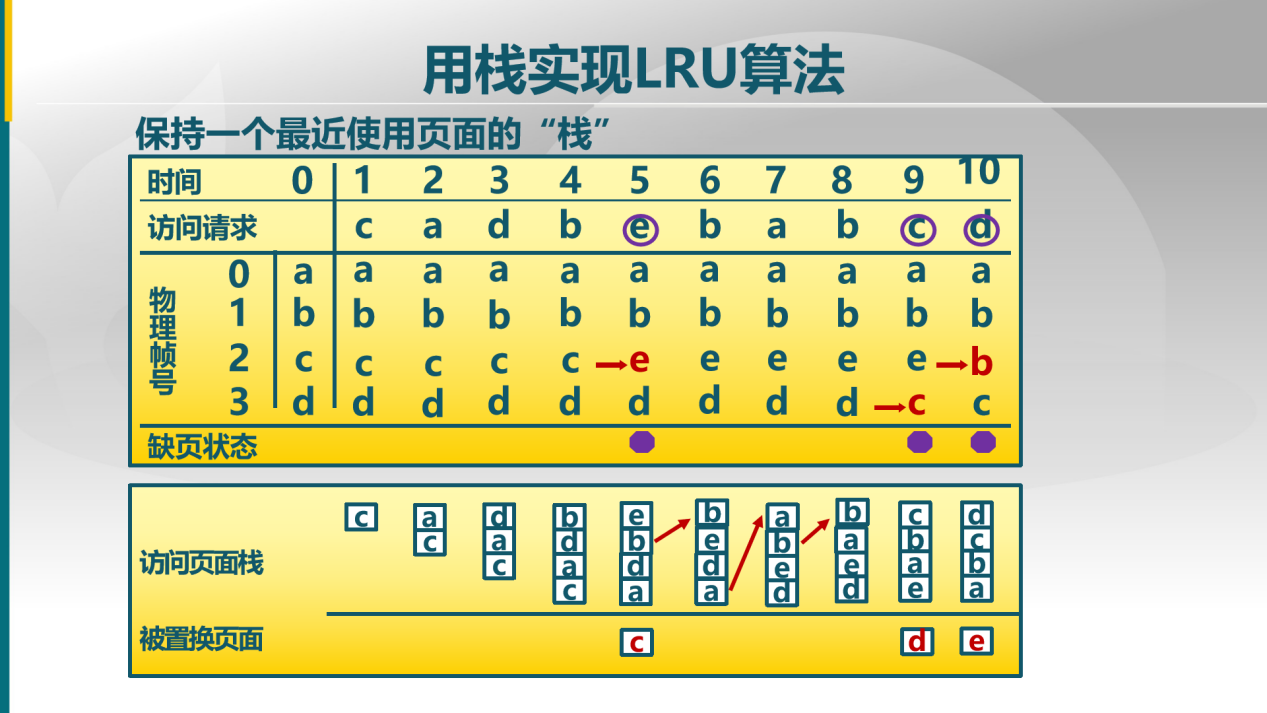
例子：



上图中，每页的下次访问时间，下次应该换成上次。



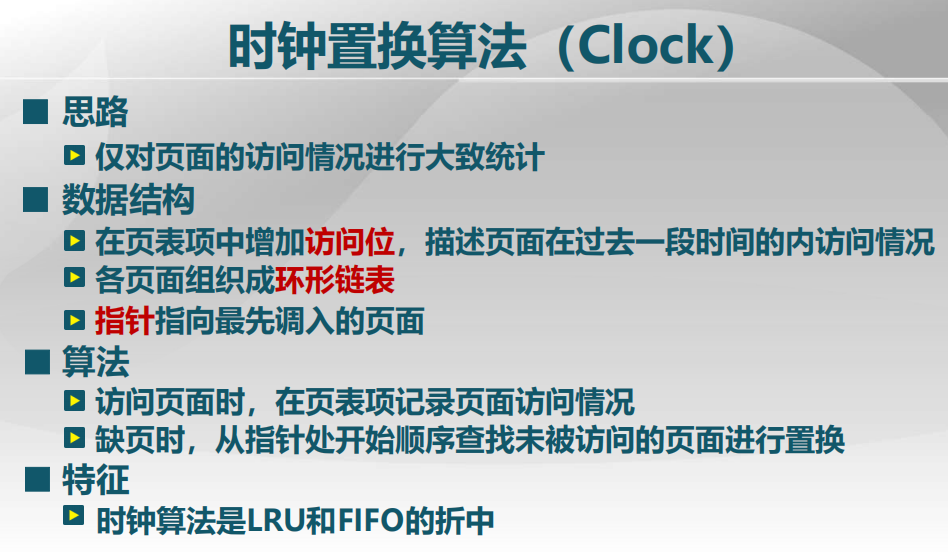
下图是用活动页面栈实现LRU算法的一个例子：



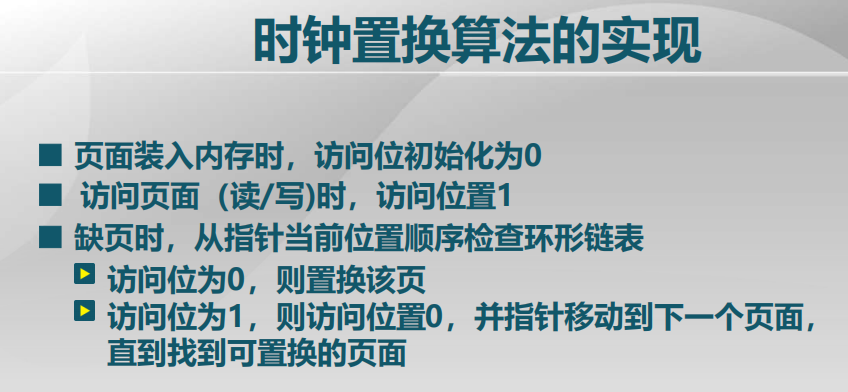
上图注意是从PPT上截的，pdf上因转换格式问题，画的不对。可见该算法开销是很大的，需要把栈整个遍历一遍，并且对栈的操作也是十分耗时的。

**第三讲 时钟置换算法和最不常用算法**

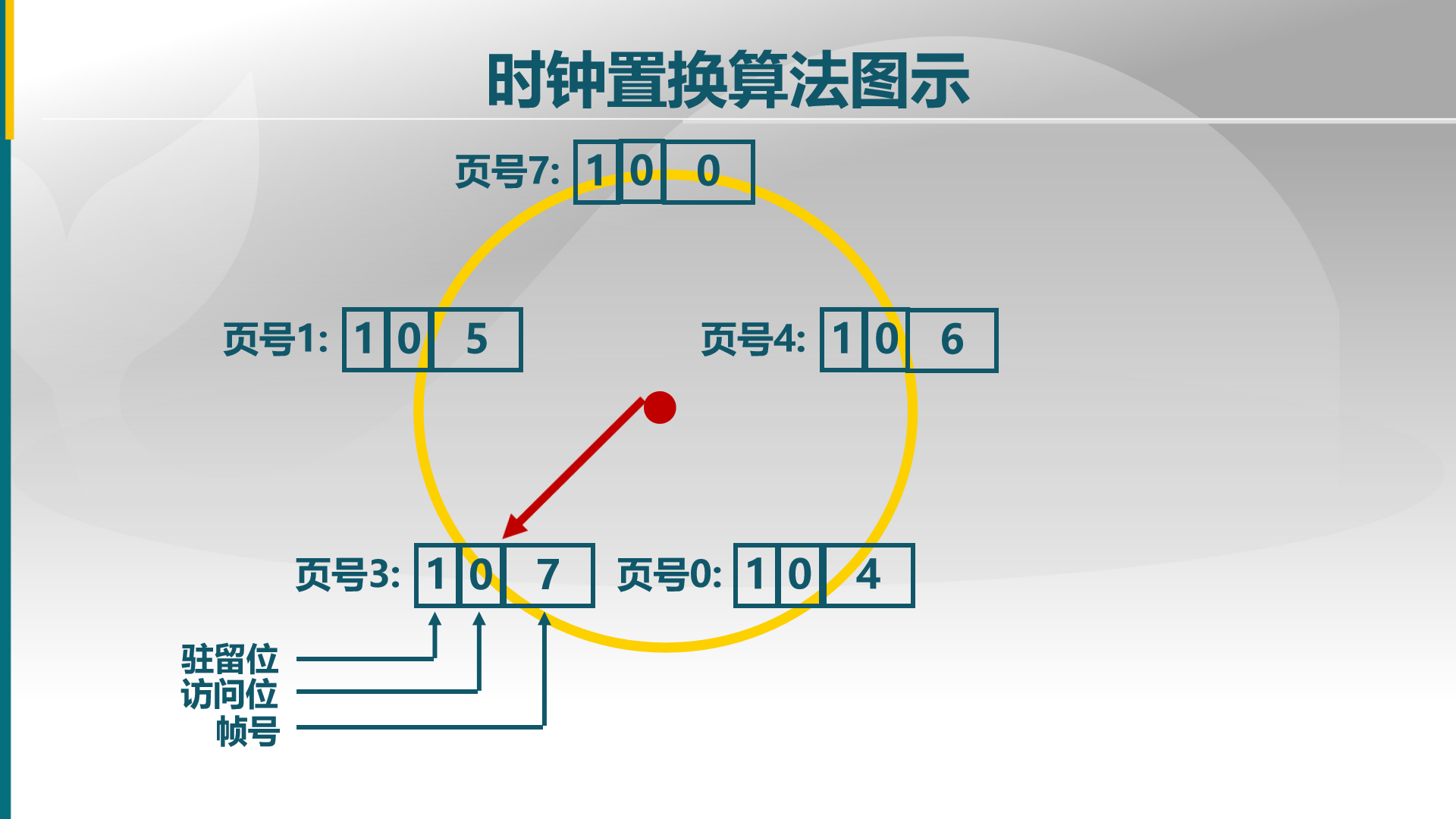
时钟置换算法和最不常用用算法是对LRU做的一种近似和简化。



折中是指：Clock算法对过去的访问情况有考虑(访问位)，做了统计，所以它不像FIFO完全不考虑过去访问情况，另外，它不像LRU，对过去的访问情况考虑的很详细，Clock只是找到环形链表中从指针开始处第一个过去未访问的做置换。具体实现如下：



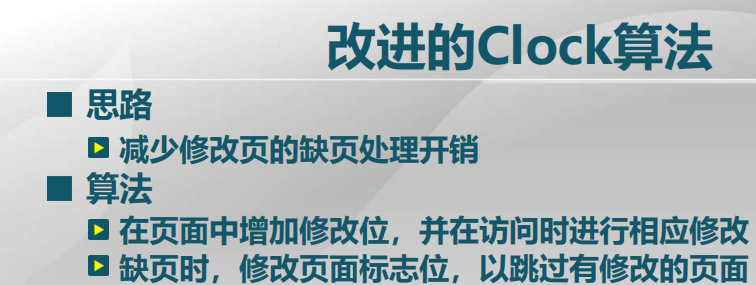
时钟置换算法的具体演示动画见PPT，不截图了，太麻烦：开始访问页号1，帧号为5的，访问位置1，若已经置1的对其再次访问，1不动即可。缺页时，从页号0开始，因页号0，3，1，7的访问位均为1，因此清0，最后找到页号4，其访问位为0，置换该页，将其置换为新的物理帧即可，然后对该页访问，访问位置1(这个我下的PPT上有误，但讲课PPT上无误)。随后又发生缺页，按上述规则，对页号为3的页置换即可。



注意，该PPT上，最后，页号3处应为117，页号4处应为116，PPT有误！



蓝色的页表项表明指针所在位置。注意时间5，初始时访问位全是1，那就会全改成0，改一圈回到最顶上，发现a的访问位为0，就对其进行置换。详细的演示过程见PPT。

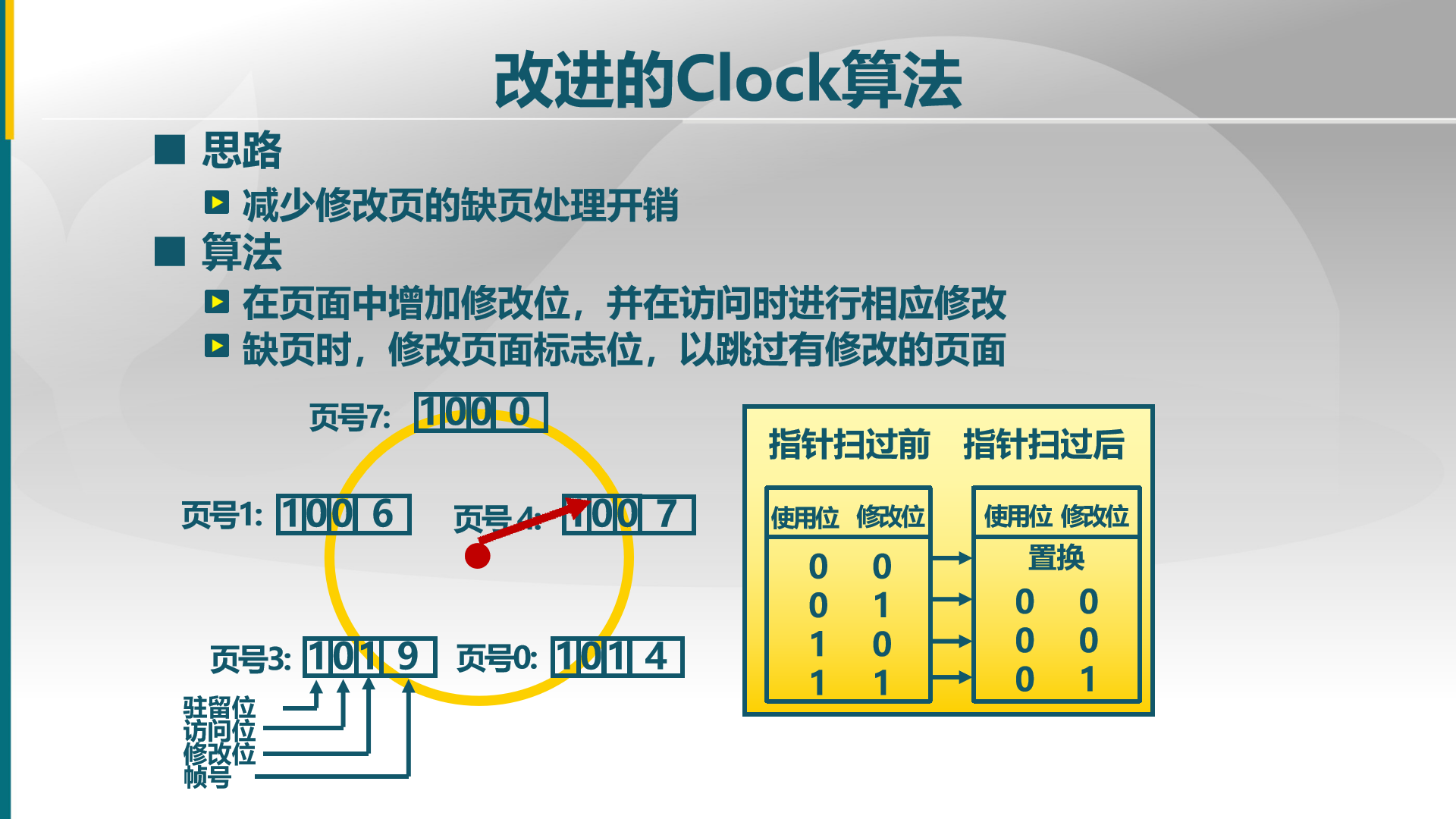


也就是说，如果是读，那就只改访问位为1，修改位还是0，如果是写，两位同时改。

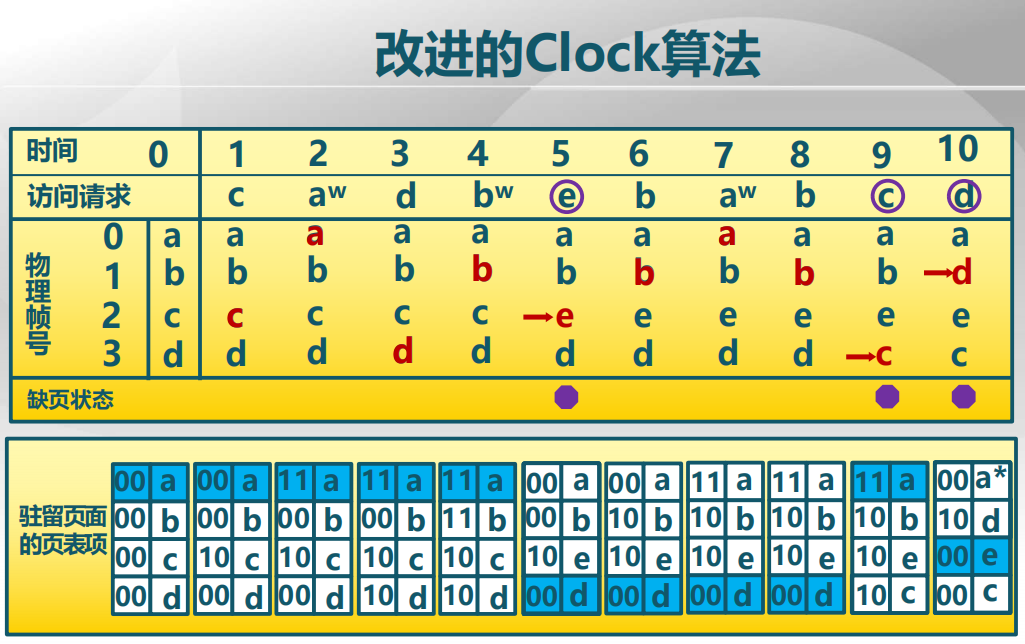
具体例子见PPT。



访问位，修改位均为0的可以置换；访问位为0，修改位为1，说明前一轮的修改没写出去，因此修改位清0并将修改写入外存；访问位为1，修改位为0，将访问位改为0即可；若两位均为1，只将访问位改成0即可。注意下图中，也有误：页号1处应为1106，页号4处应为1107



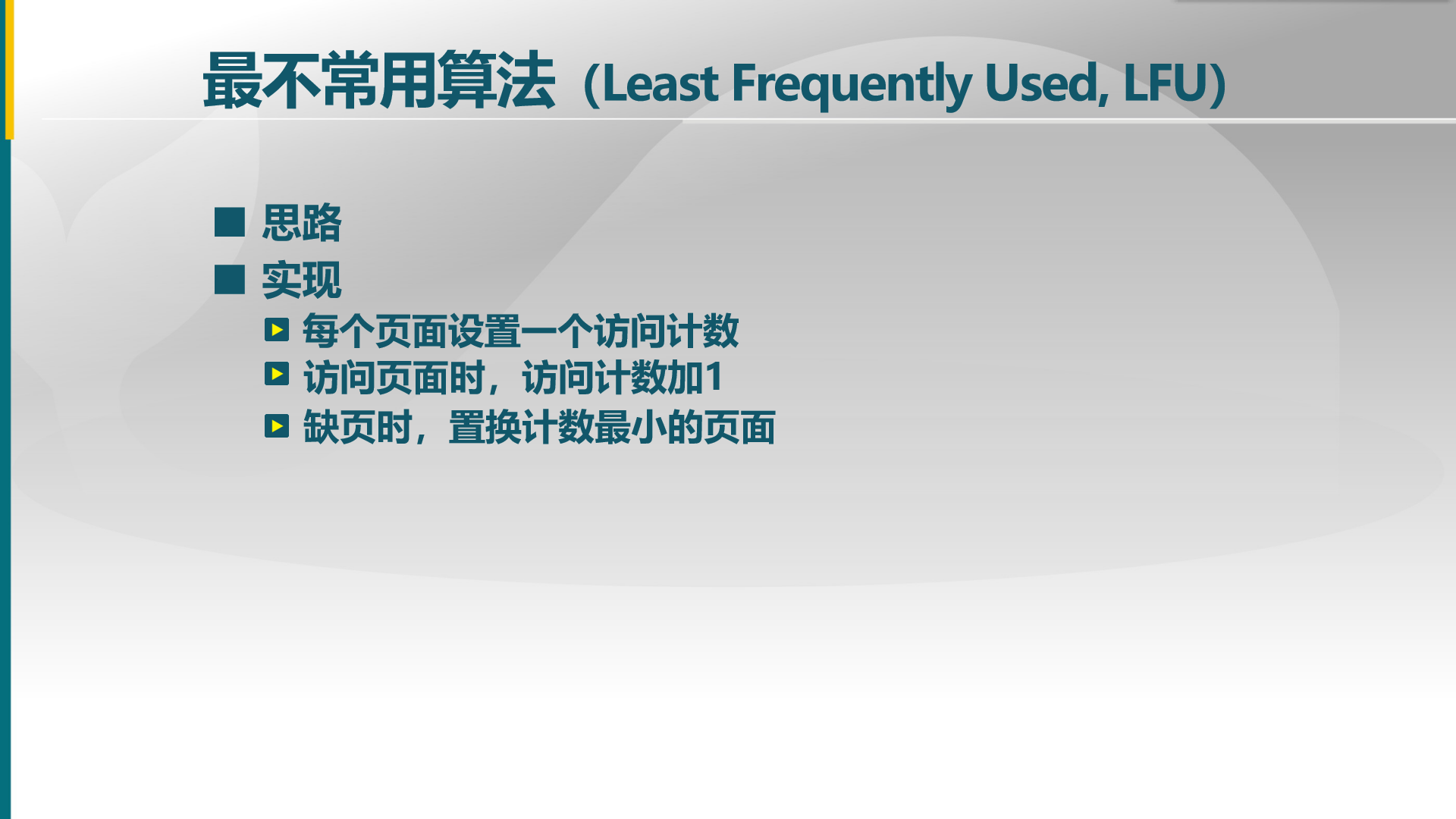
改进的时钟算法实例：

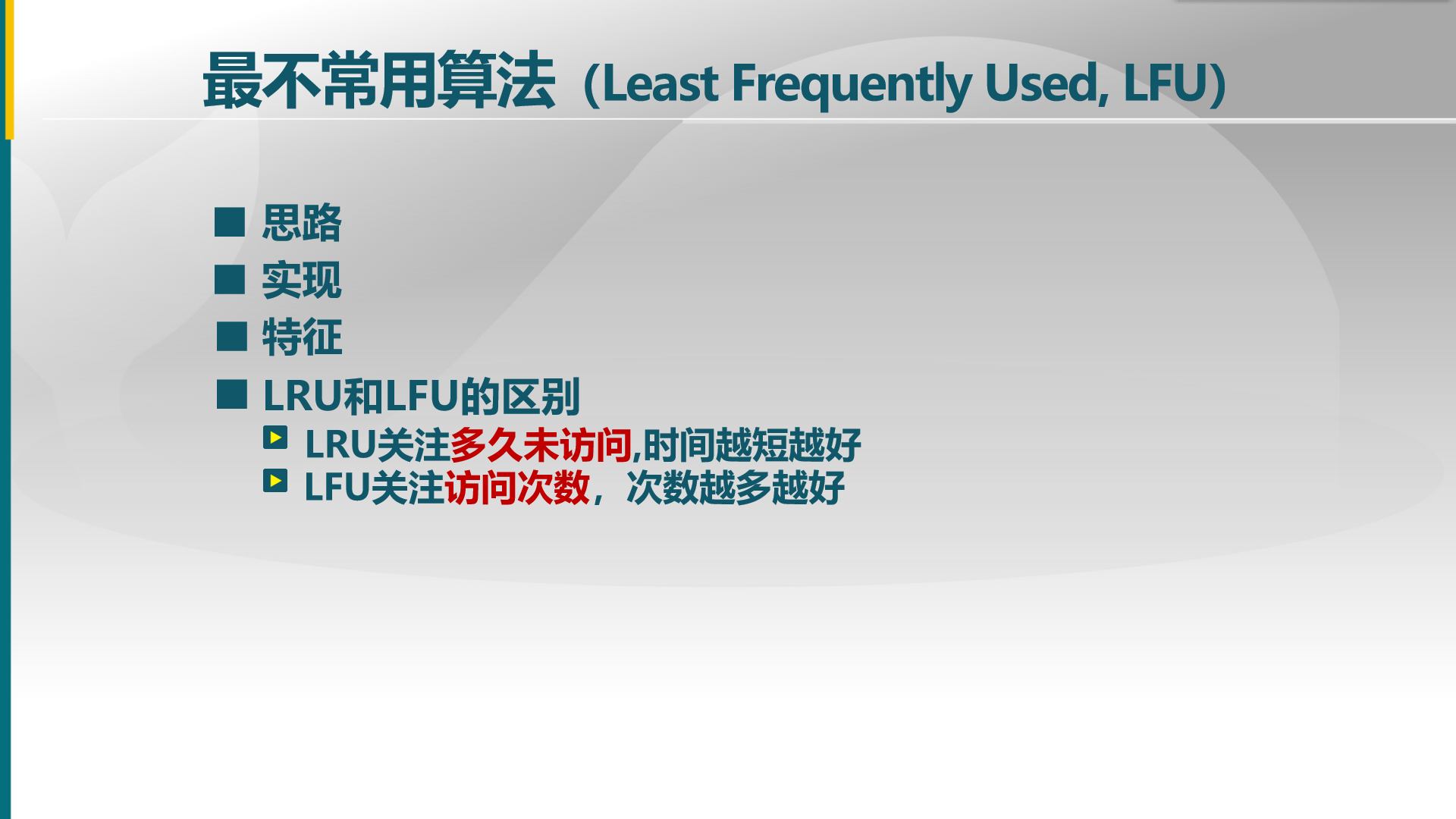
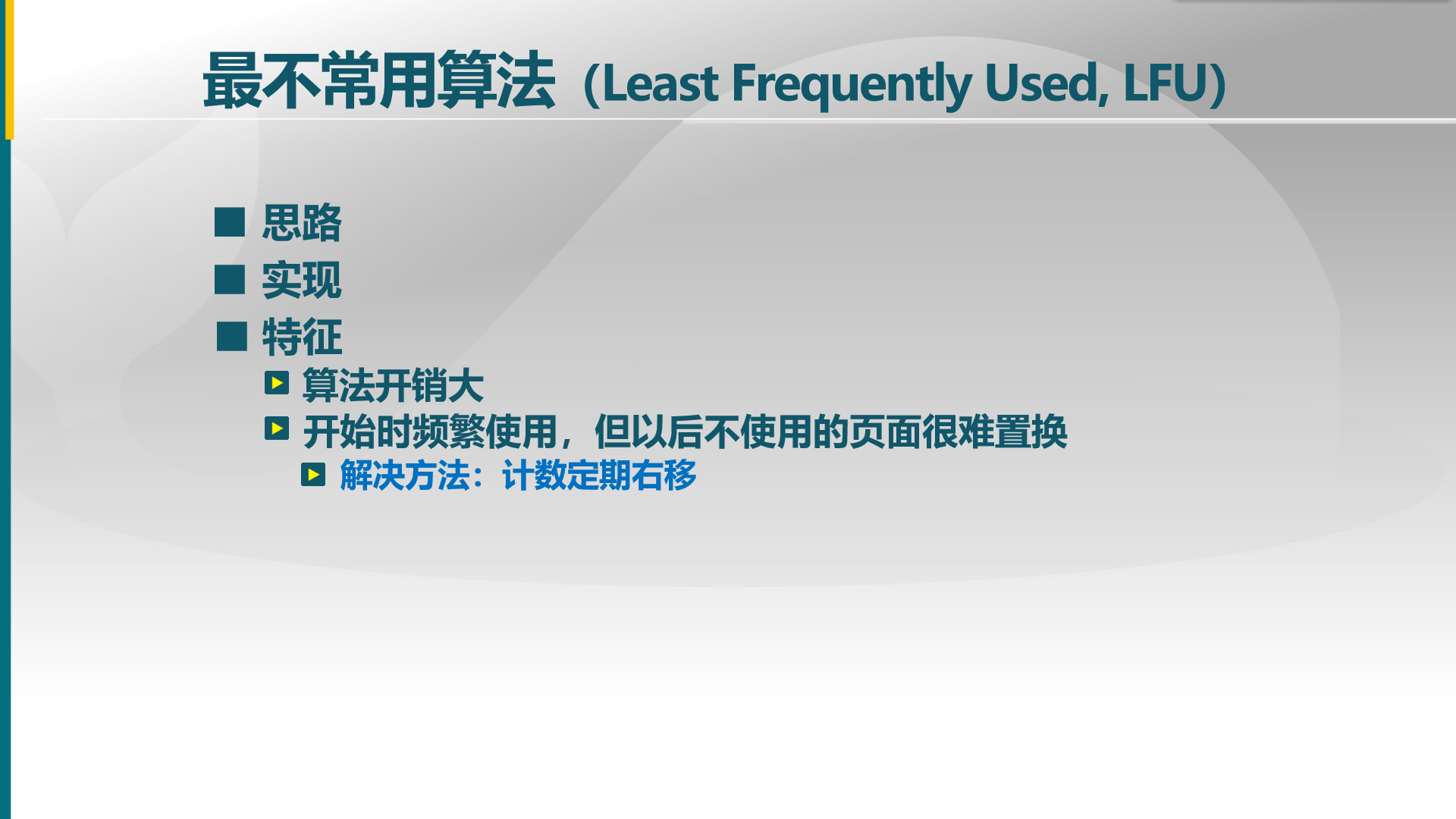


注：页表项01表示访问位为0，修改位为1，aw那种意思是对其访问是写的。具体操作规则已经讲过，更详细的动态演示见PPT。

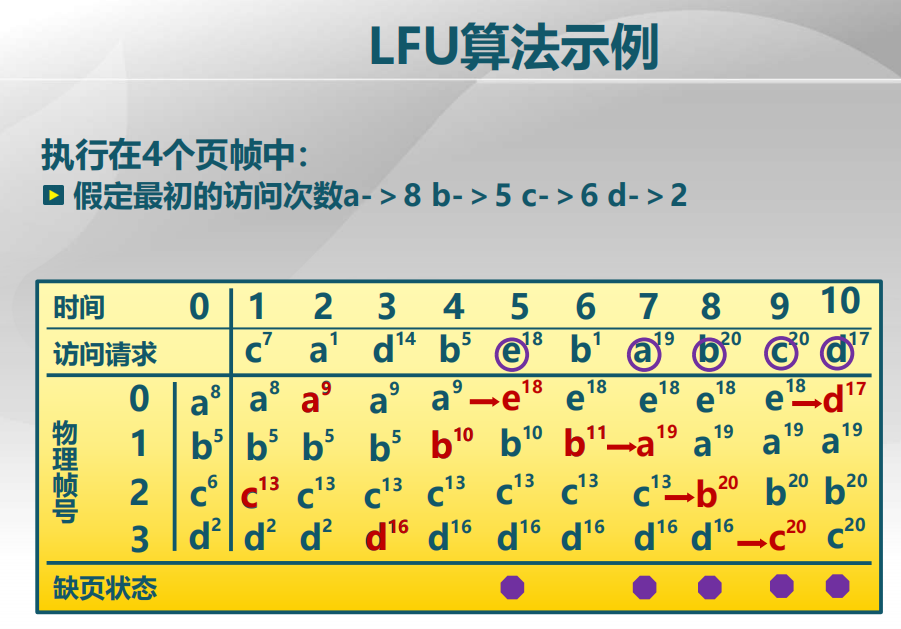
LFU也是对LRU的一种简化：







例子：

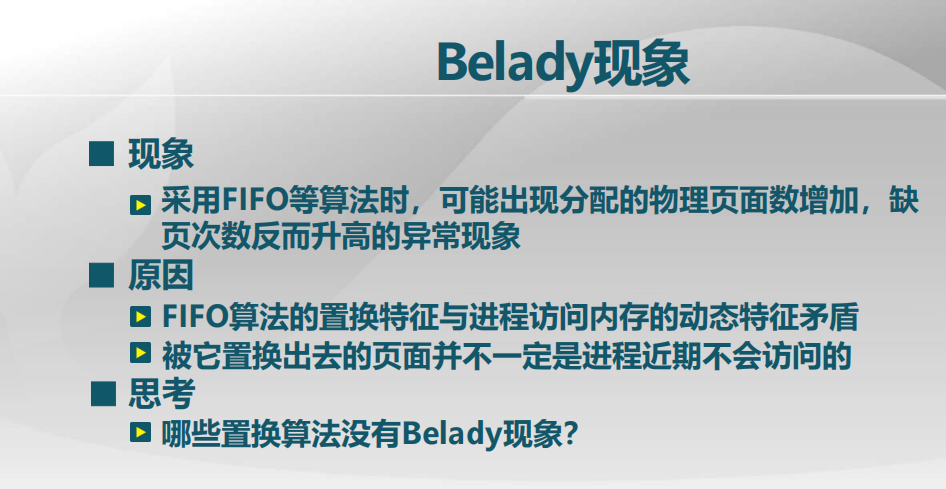


本节习题有一题是：

物理页帧数量为4，且初始时没有对应的虚拟页。虚拟页访问序列为 0,3,2,0,1,3,4,3,1,0,3,2,1,3,4 ，请问采用CLOCK置换算法（用2个关联，bit表示存在时间,可以表示4）的缺页次数为？

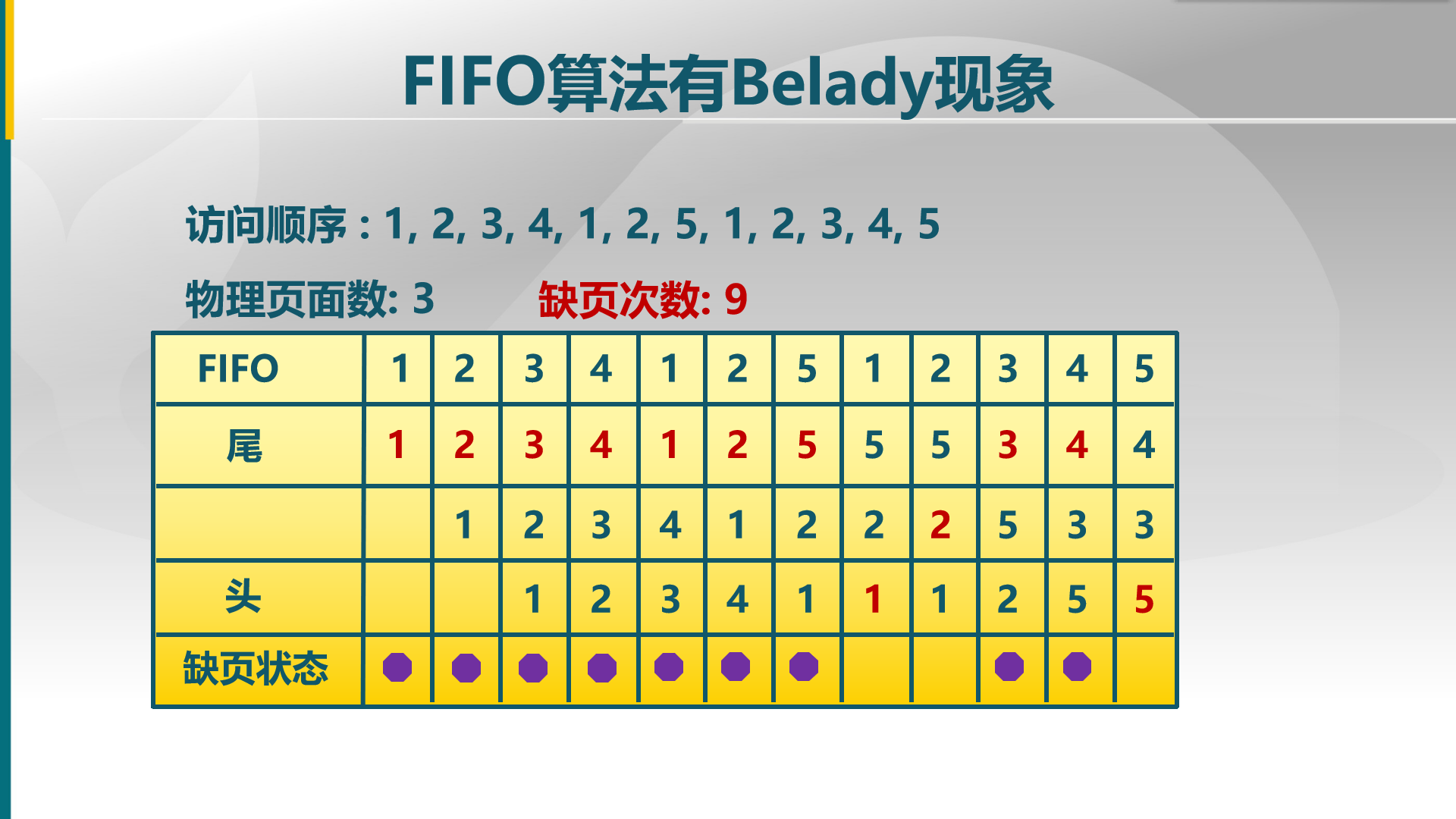
答案为7，学堂在线给的4个答案都是错的。这题其实就是2个访问位，规则和一个访问位大同小异，当物理页帧满了时，只有遇到访问位是00的才可以替换，否则将访问位减一转向下一个。本来就在在物理帧中的页，访问一次，访问位就加1.注意，指针移动只在缺页时发生。

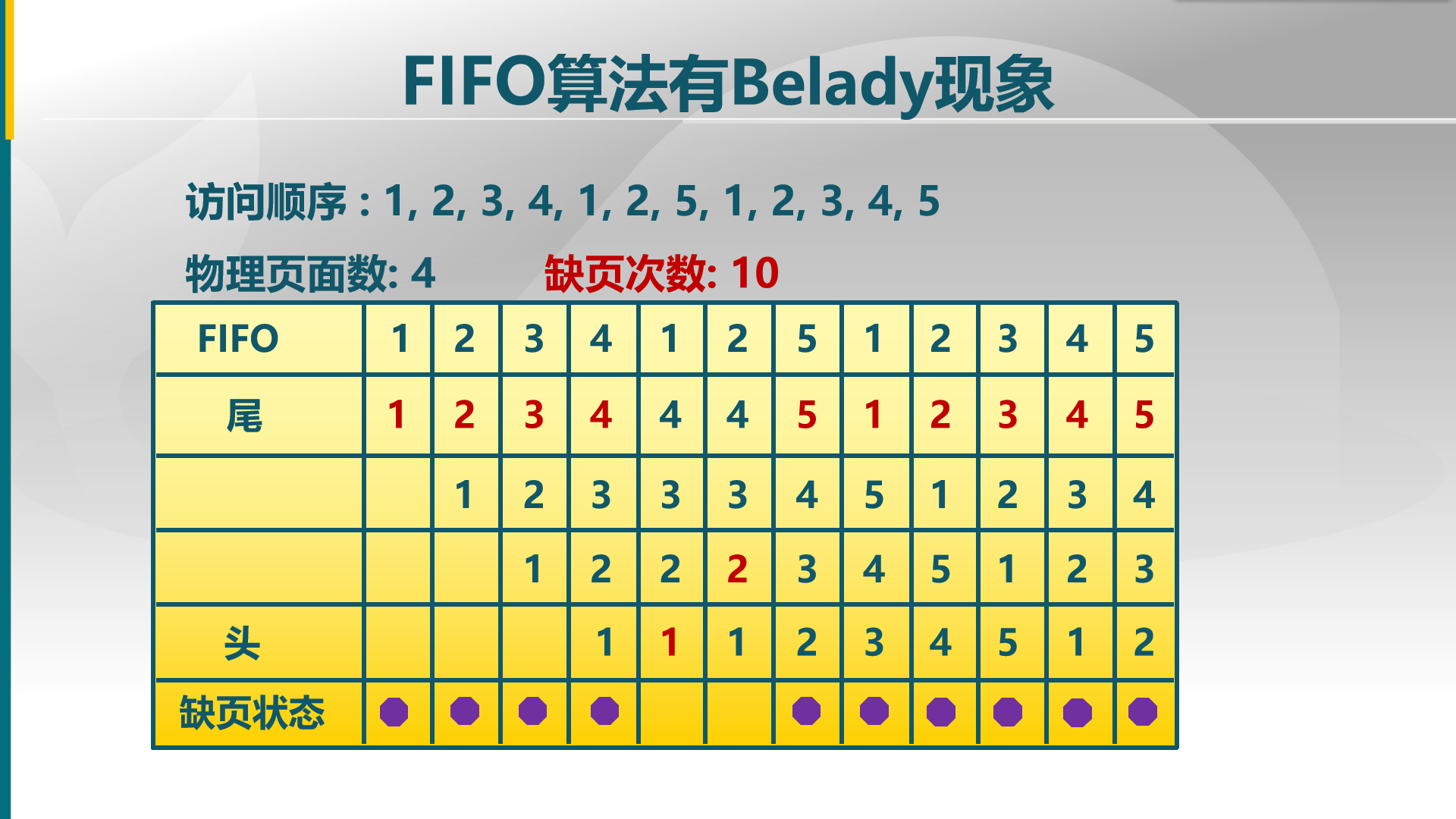
**第四讲 Belady(读作bi lei di)现象和局部置换算法比较**

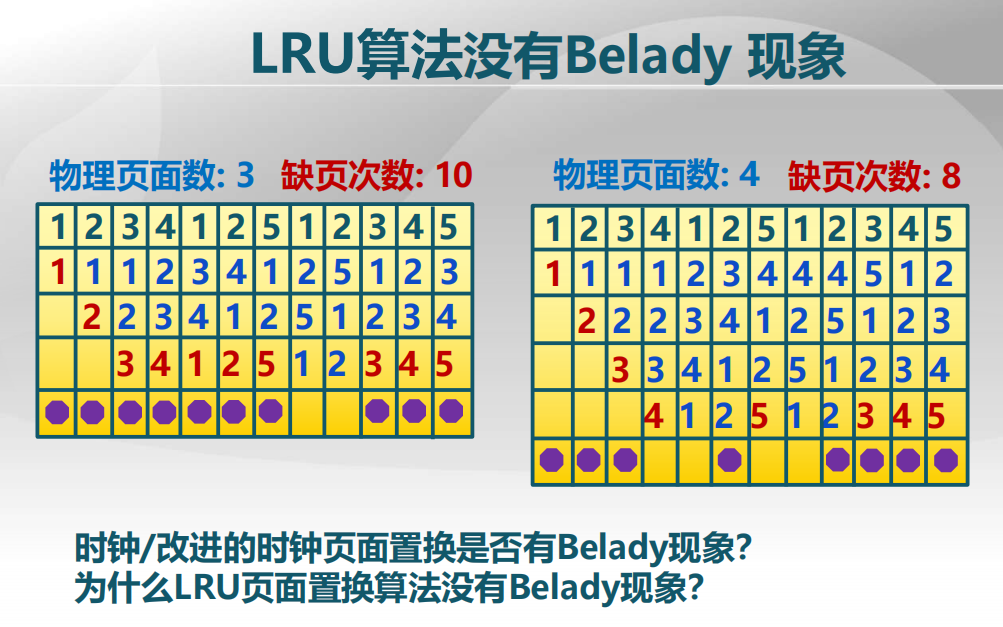


Belady现象：为某进程分配的物理页面数增加，缺页次数反而升高。

下面是对FIFO算法Belady现象的示意：







时钟/改进的时钟页面置换也是有Belady现象的，具体例子用上面的例子即可。

LRU算法是基于堆栈的算法，这种算法没有Belady现象，具体原因见：

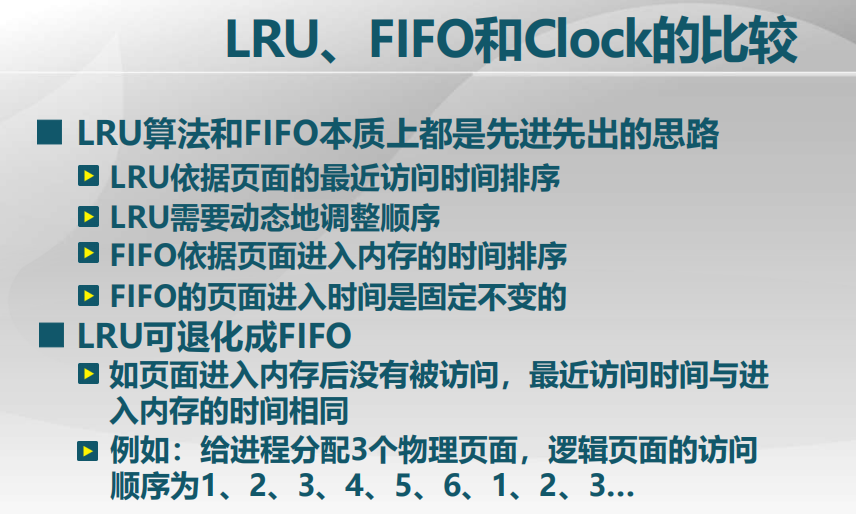
<https://www.geeksforgeeks.org/beladys-anomaly-in-page-replacement-algorithms/>

<https://www.geeksforgeeks.org/second-chance-or-clock-page-replacement-policy/>

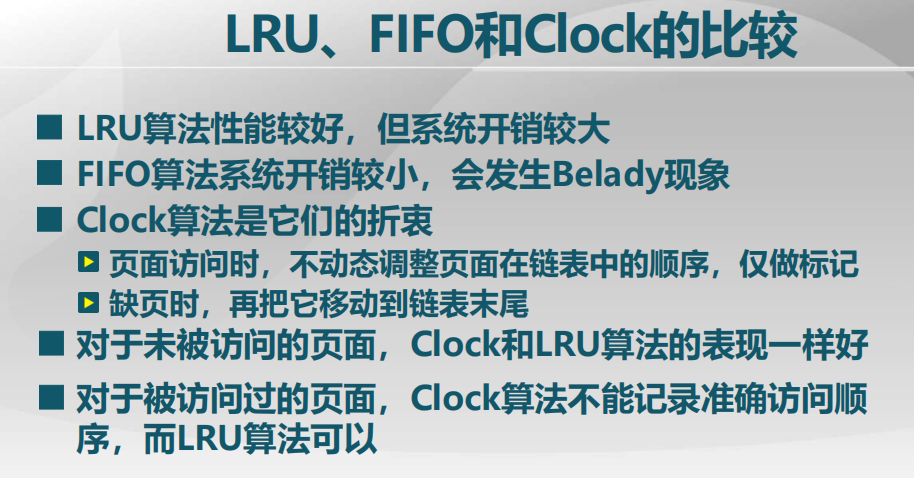
<https://pages.mtu.edu/~shene/FORUM/Taiwan-Forum/ComputerScience/003-OS/SLIDES/chap09.pdf>

机翻：所有基于堆栈的算法都不会遭受Belady异常，因为这些类型的算法为页面（用于替换）分配的优先级与页面帧数无关。此类策略的示例是OPT，LRU和LFU。

在LRU算法中，每次引用页面时都会将其移动到堆栈的顶部，因此，堆栈的前n个页面是n个最近使用的页面。即使帧数增加到n + 1，top的堆栈中将有n + 1个最近使用过的页面。而FIFO以及Clock算法会把最近使用过的页面替换出去。



LRU需要动态的调整顺序，开销大，性能好。FIFO不需动态调整顺序，开销小，性能差。

LRU可退化成FIFO:如果页面在内存中只访问一次就不再访问了，LRU就退化成FIFO了。

Clock算法缺页时，扫描第一个访问位为0的换出，可以理解为某种顺序的调整。对于未被访问的页面，实际上，LRU、FIFO、Clock三者是一样的，因为相当于没有可以借用的以前的信息。

课后有一题：

下面哪些页面淘汰算法不会产生Belady异常现象?

A.先进先出页面置换算法（FIFO)

B.时钟页面置换算法（CLOCK)

C.最佳页面置换算法（OPT）

D.最近最少使用页面置换算法（LRU）

应选择C,D,学堂在线上题干落了不字，还给的C,D,给错了。

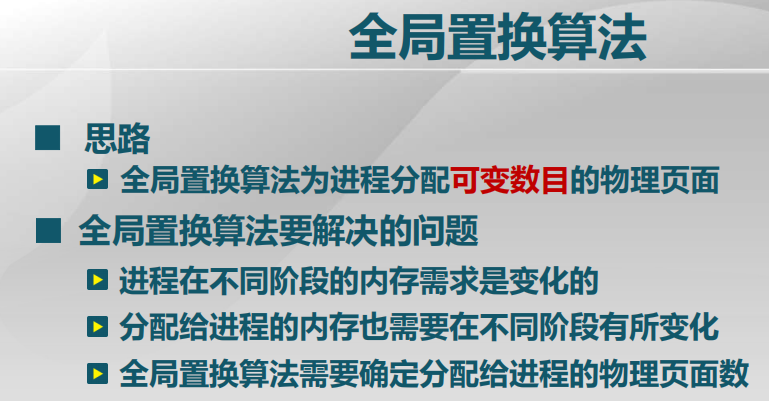
**第五讲 工作集置换**

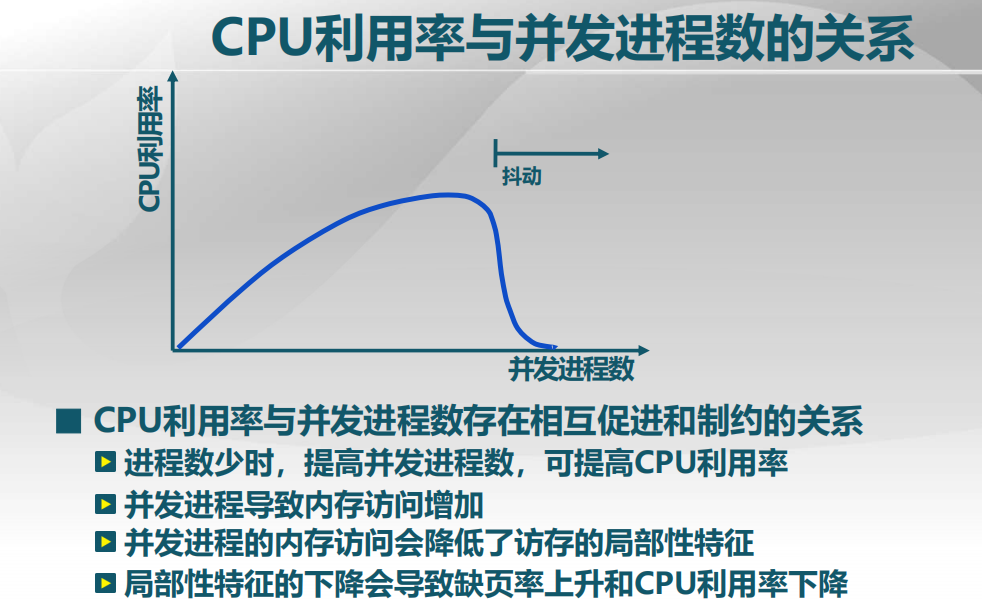
下面开始介绍全局页面置换算法：工作集置换算法、缺页率置换算法。工作集置换算法可理解为最优算法在全局里的一种体现，它们之间有一定的关系。





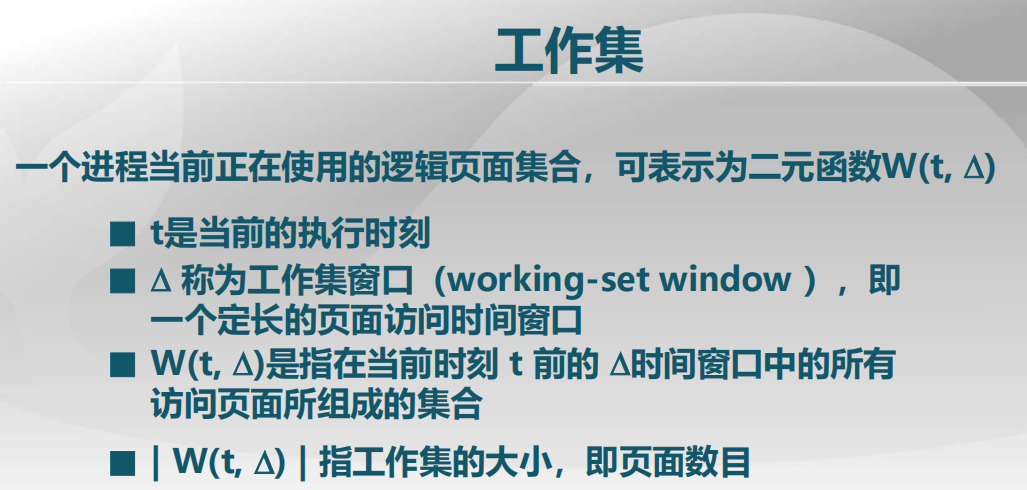
针对局部置换算法的缺陷，有了全局置换算法：



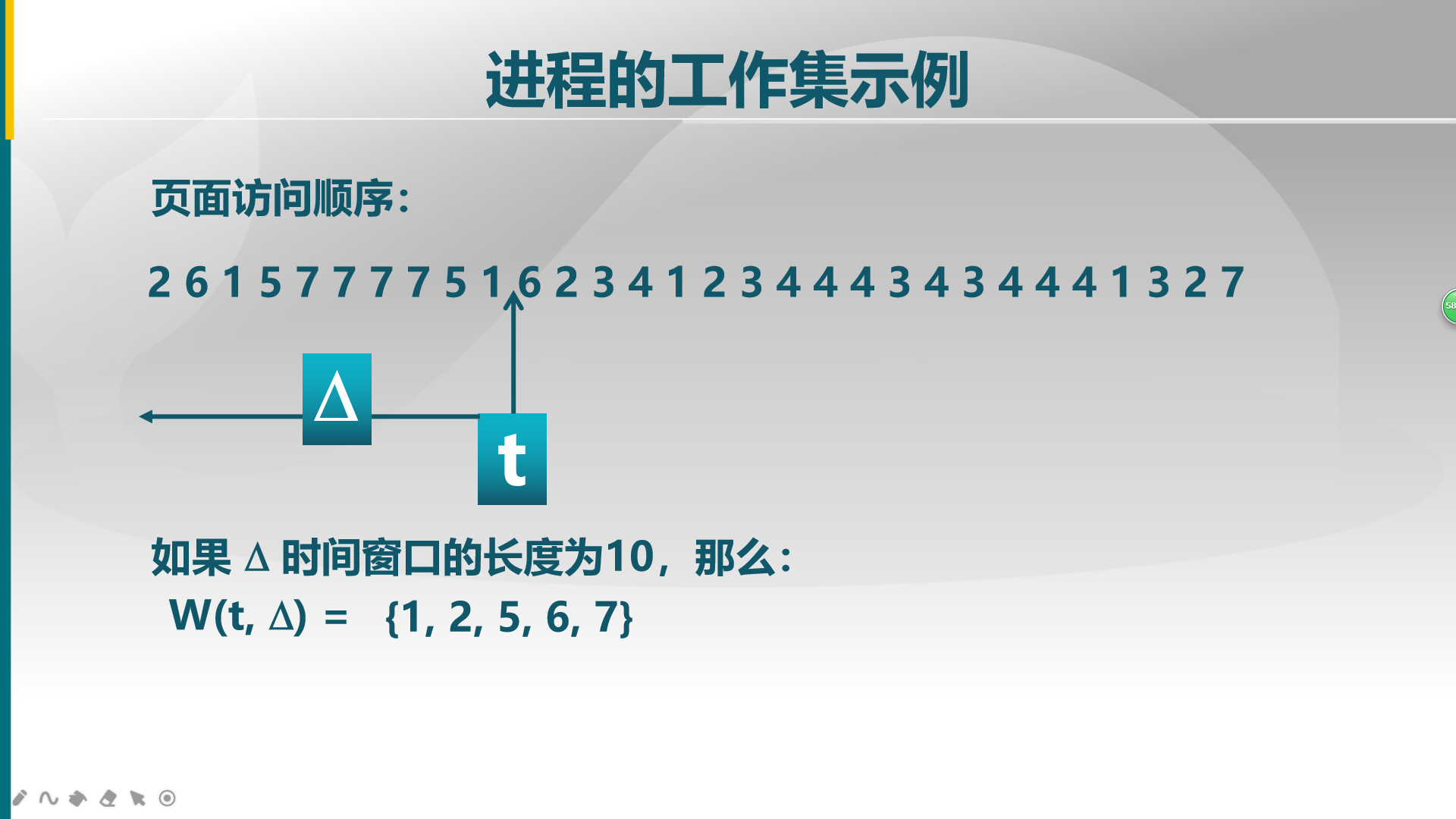


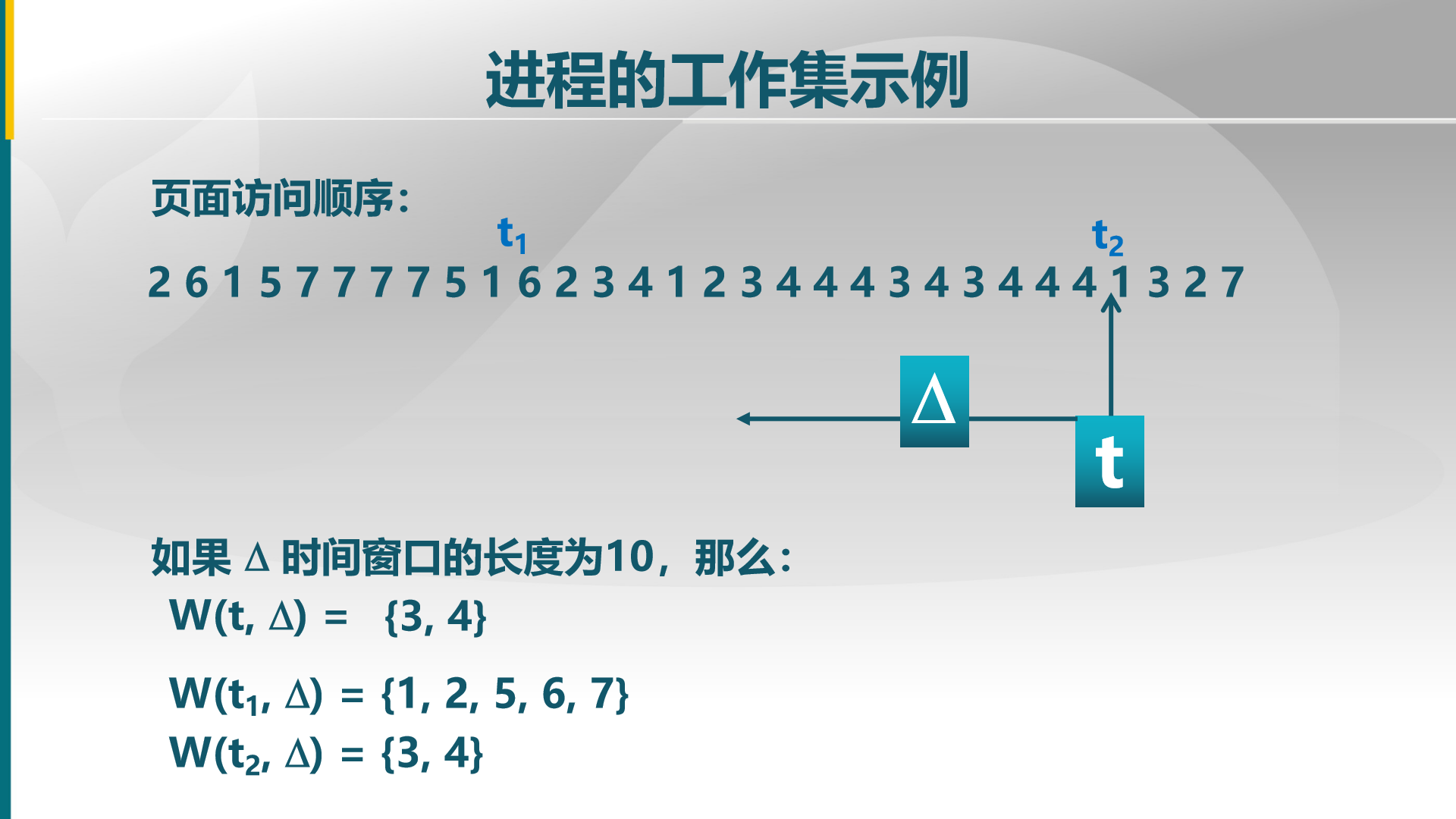
并发进程过多，访问的局部性降低，导致缺页率上升，大部分时间都在内存和外存之间换页，从而CPU利用率下降。

引入工作集的概念：



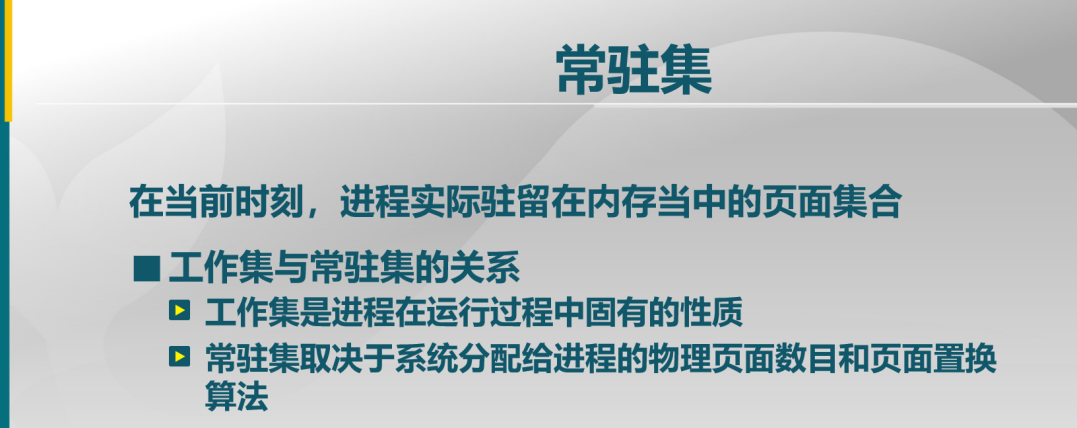
下面是工作集的示例，更加动态的演示见PPT(从该时间往前数10个，看有哪些页面)：

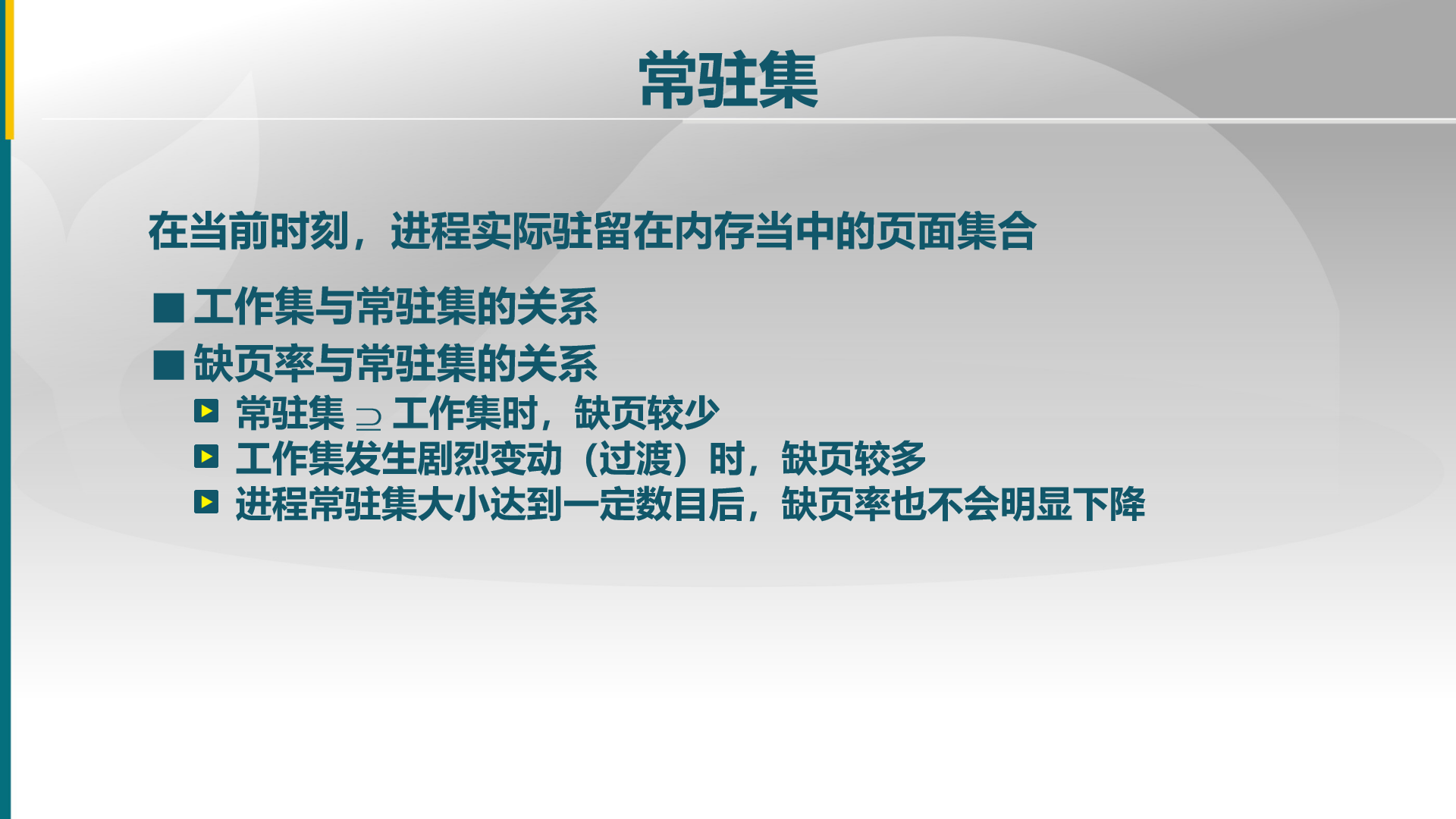


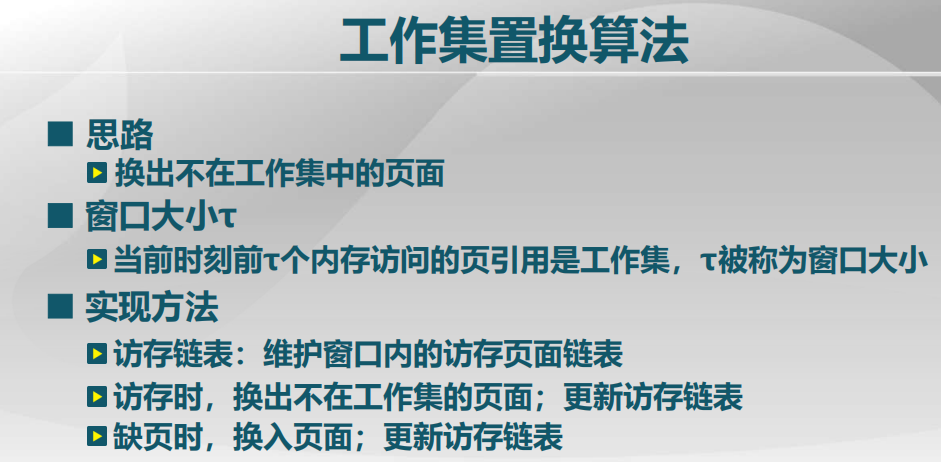




下面定义常驻集：







工作集置换算法并不是在缺页时才换页，而是时刻换出不在工作集中的页面。实际上，访存很频繁，换出不在工作集的页面开销是很大的，缺页时反倒简单，直接换进来就行了。

例子：



τ相当于Δ，即时间窗口大小。时刻1，-2 -1 0 1:e d a c，工作集为{a,c,d,e}

时刻2，-1 0 1 2:d a c c,工作集为{a,c,d}

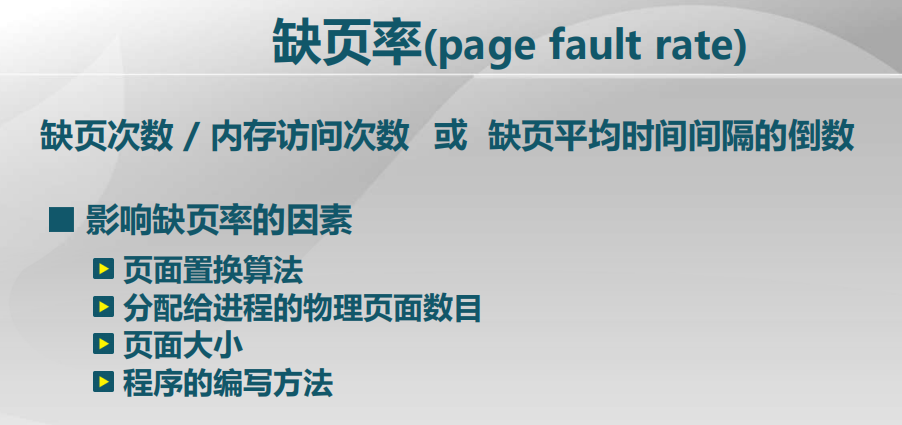
时刻3，0 1 2 3:a c c d,工作集为{a,c,d}

时刻4，1 2 3 4:c c d b,工作集为{b,c,d}等等，以此类推。

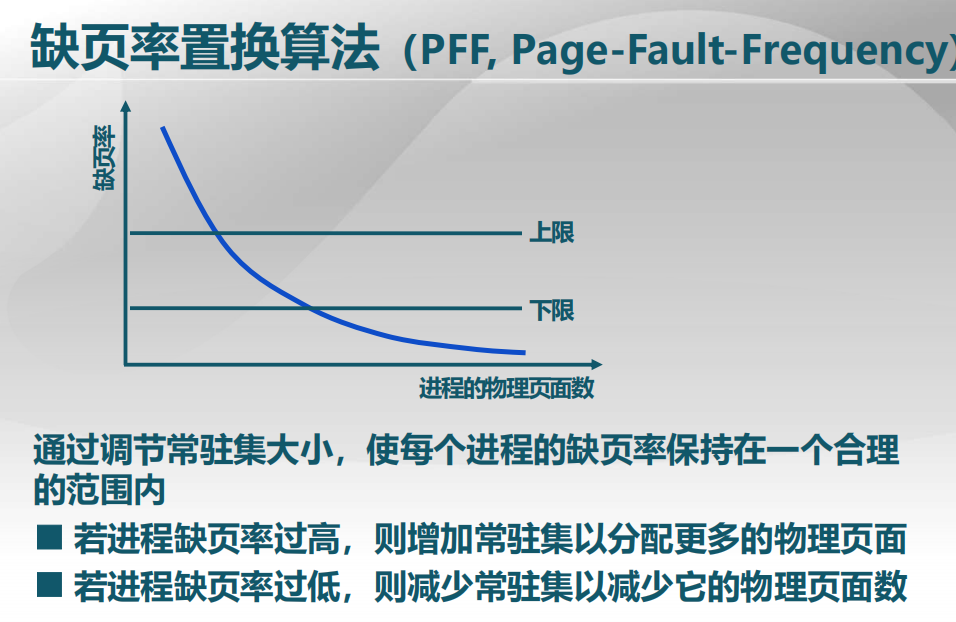
课后题注意：

物理页帧数量为5，虚拟页访问序列为 4,3,0,2,2,3,1,2,4,2,4,0,3，请问采用工作集置换算法（工作集窗口T=4）的缺页次数为：5次。 该题前3个页面4,3,0默认已经在内存中，是按PPT上的例子走的。

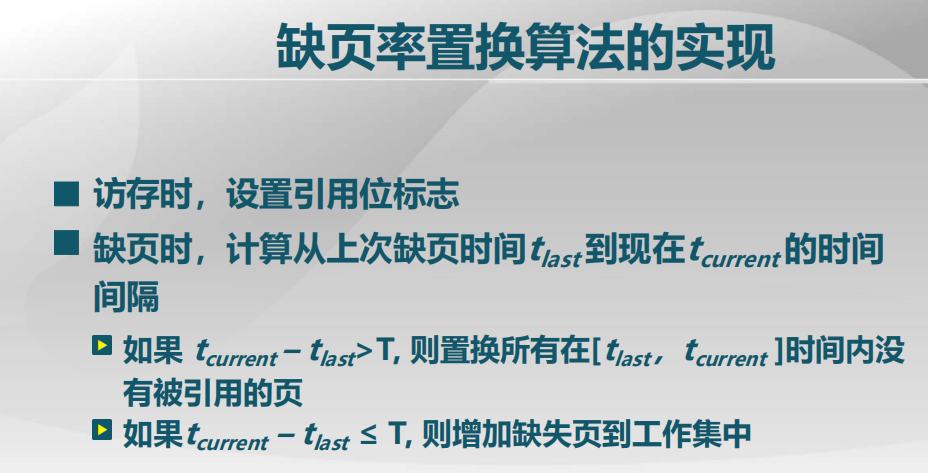
**第六讲 缺页率置换算法**



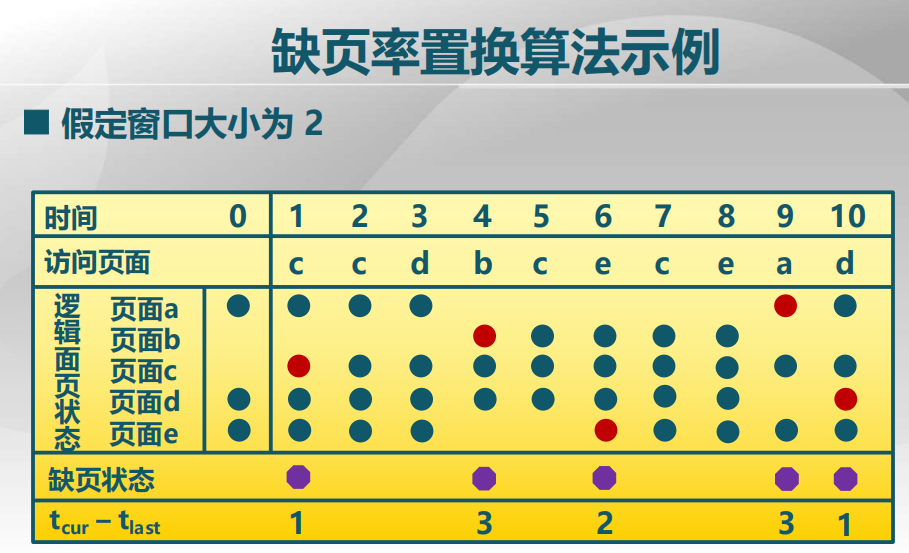
在实际缺页率算法中，缺页次数/内存访问次数不好把握，我们更多使用缺页平均时间间隔的倒数作为缺页率。程序编写方法：如局部性和程序的编写方法有关。



缺页率若过低，则并发度会过低，CPU利用效率会下降。



实例：



时间1发生缺页，时间间隔：1-0=1

时间4发生缺页，时间间隔：4-1=3>2，这时需要把这段时间，即[1,4]没有访问的页面剔除。

时间6出现缺页，时间间隔：6-4=2≤2，直接加上即可。

时间9出现缺页，时间间隔：9-6=3>2，置换在[6,9]中没被访问的页。

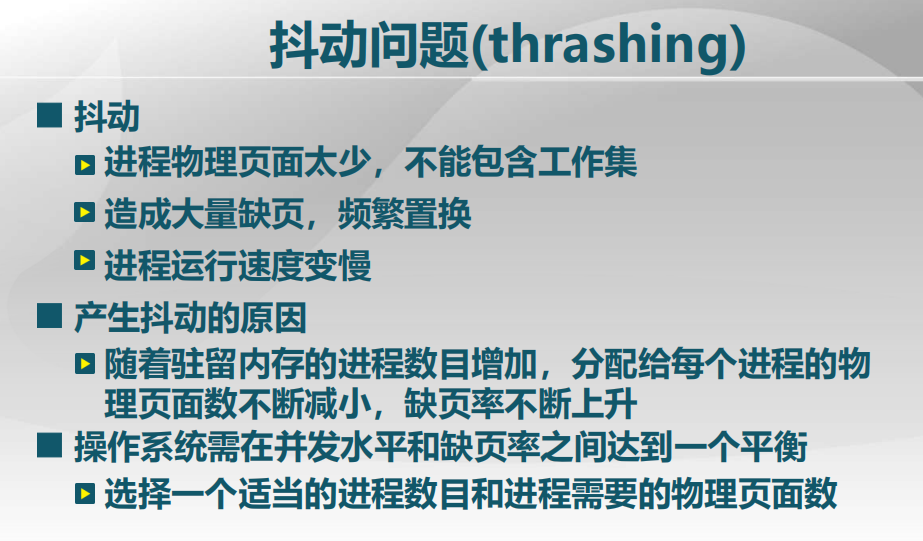
以上，可见在该进程中，随时间变化，页面数会发生变化。该算法在逼近工作集大小曲线，同时努力降低开销。缺页率置换算法在时间间隔够大时才在缺页时进行页面的淘汰，而不是像工作集算法一样每个时间都进行页面淘汰，这就降低了开销。

课后题目：物理页帧数量为5，虚拟页访问序列为 4,3,0,2,2,3,1,2,4,2,4,0,3，请问采用缺页率置换算法（窗口T=2）的缺页次数为：5次。 该题前3个页面4,3,0默认已经在内存中，是按PPT上的例子走的。

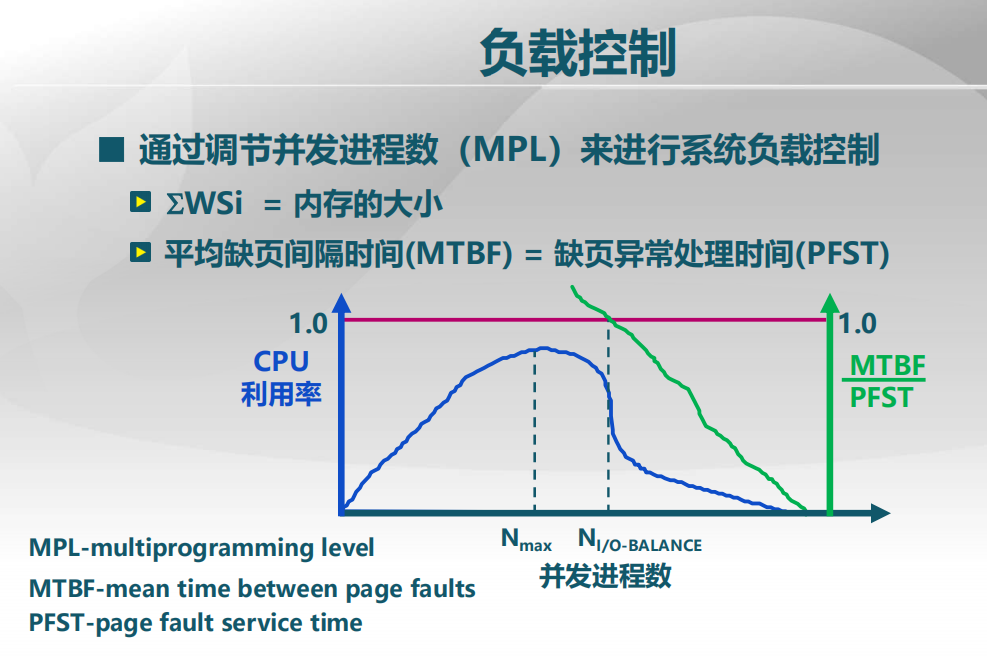
**第七讲 抖动和负载控制**

局部置换算法和全局置换算法都涉及到一个问题：一个系统里有多少个并发执行的进程一起执行对于系统来说效率是最高的，这就是要研究抖动和负载控制。

首先看抖动：



内存中进程数目太多，每个进程分配的物理页面太少，造成抖动问题。进程太少，系统的并发性不好；进程太多，缺页率会过高。我们需要找一个折中点。



WSi是各工作集的大小。负载控制想通过控制各工作集大小达到：各进程工作集大小之和等于内存大小。在上面曲线图的Nmax位置，各进程工作集大小之和等于内存大小。Nmax的位置比较难考察，这时利用平均缺页间隔时间(MTBF) = 缺页异常处理时间(PFST)，若间隔>处理时间，这时处理来的及完成，也就是红线和绿线的交点之前；若间隔<处理时间，那满负荷处理也忙不过来，可知Nmax应该在NI/O-BALANCE点之前，负载均衡的平衡点应在Nmax和NI/O-BALANCE之间。