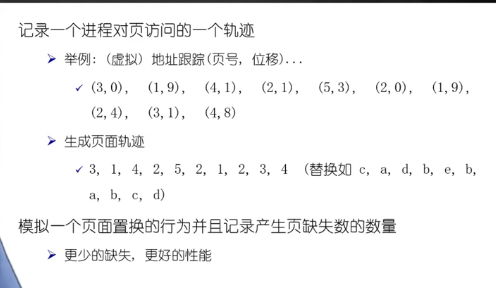
页面置换算法：

局部页面置换算法和全局页面置换算法：

页面替换出去会对硬盘进行读和写操作

目标尽量减少页面的换进换出次数

页面锁定，使相关页（操作系统很重要的代码段和数据）常驻内存中



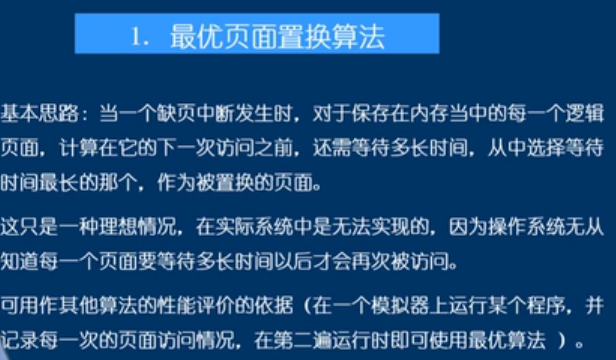
把内存地址访问的序列记录下来

（在考虑页面换出时不需要考虑页内偏移，只需要考虑整个页是否存在内存里）

最优页面置换算法：

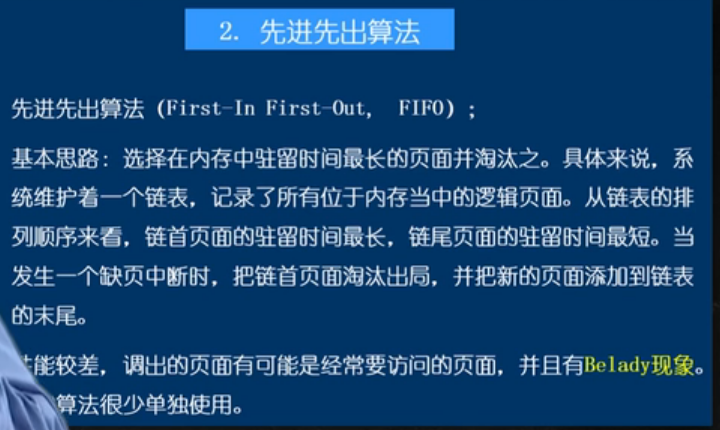
根据将来什么时候会访问来判断，需要知道未来的情况（这个很难实现）

OPT



（设计的算法能够逼近最优置换算法，但是最优实现不了）

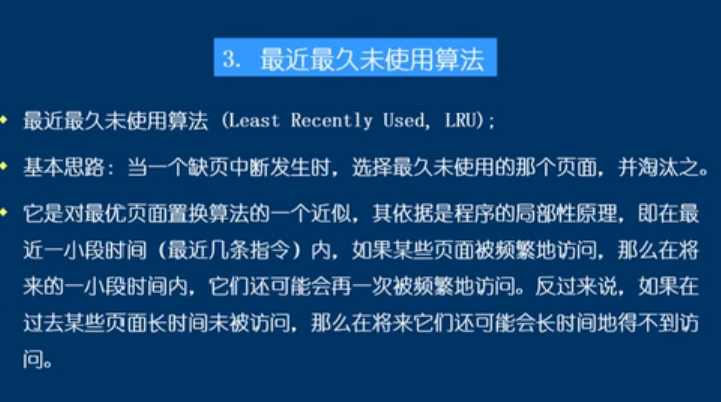
先进先出算法：FIFO



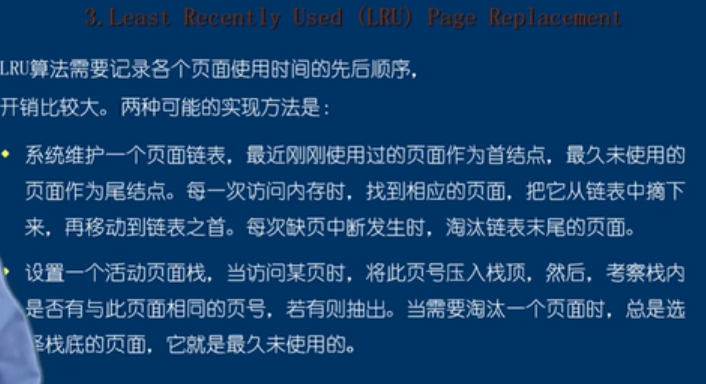
实现的简单，但是缺页可能比较多

Belady现象：分配的物理页越多缺页现象也应该越少，但FIFO可能会触发更多的缺页

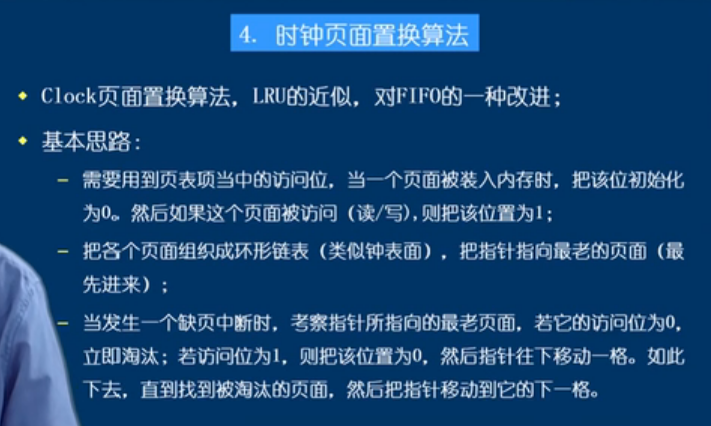
最近最久未使用：LRU

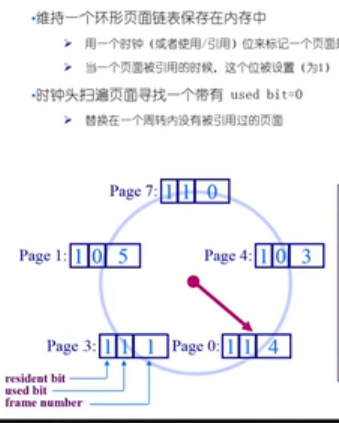


和FIFO不一样的是，FIFO在访问内存中已经有的页时链表不会发生改变，而LRU则会将该页对应的节点放到链表尾。



时钟页面替换算法：Clock





Resident bit表示是否存在，1为页在物理页中是存在的（invaild bit）

Used bit是访问位（access bit）

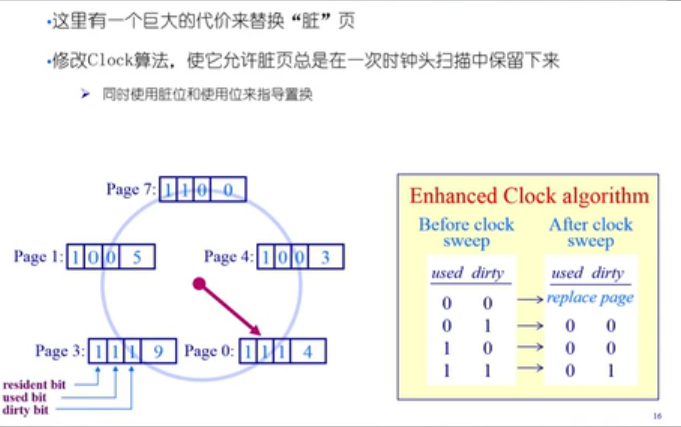
指针指向上一次访问的链表的一个位置。（0、3、4、1、5是对应的物理页帧号）

当替换的时候，是将逻辑页存在被替换的物理页帧中。

Dirty bit是脏位，就是写操作。

二次机会法：

Clock优化：（减少对硬盘的写回）



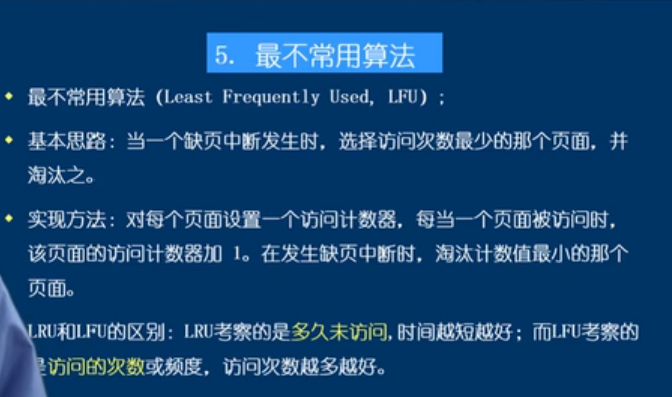
当对一个页面访问时只对access bit置位，但是写操作时将access和dirty都置位

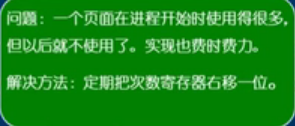
我们要找的是两个标志位都是0的页，同时在更新标记位时则是如图：

0 1 -> 0 0 1 0 -> 0 0 1 1 -> 0 1

在替换之后，指针会移到下一位。

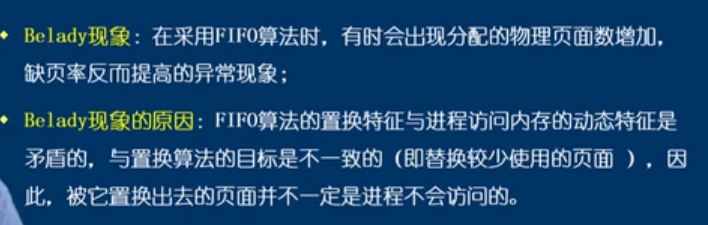
最不常用算法：LFU





存在的问题和解决办法

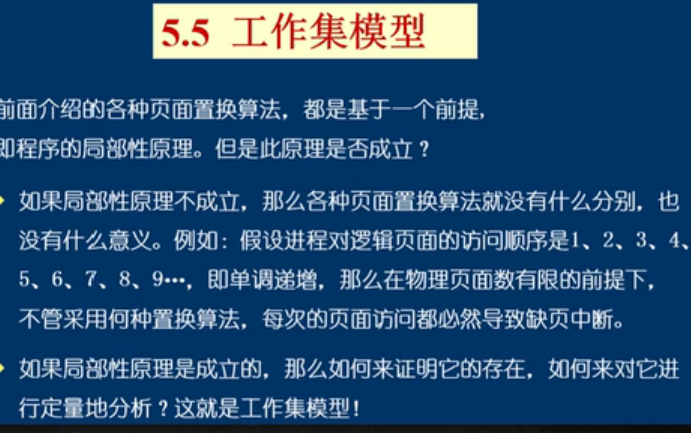
Belady现象：在使用FIFO算法时，有时会出现分配的物理页面数增加，但缺页率反而提高的异常现象。

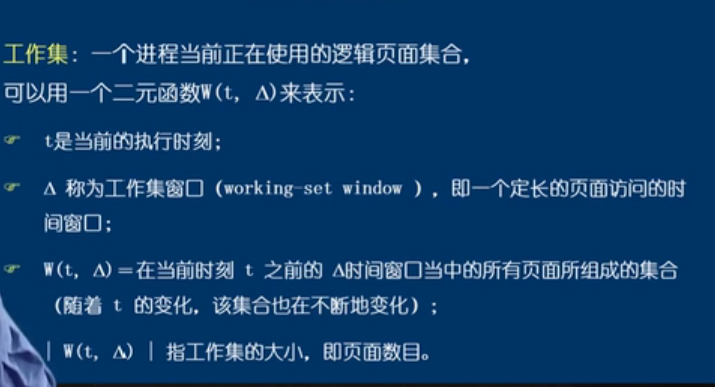


LRU满足栈算法的特点，但是FIFO不符合栈算法的特点。

序列最好具备局部性的特征，这样LRU和CLOCK算法才能起作用。

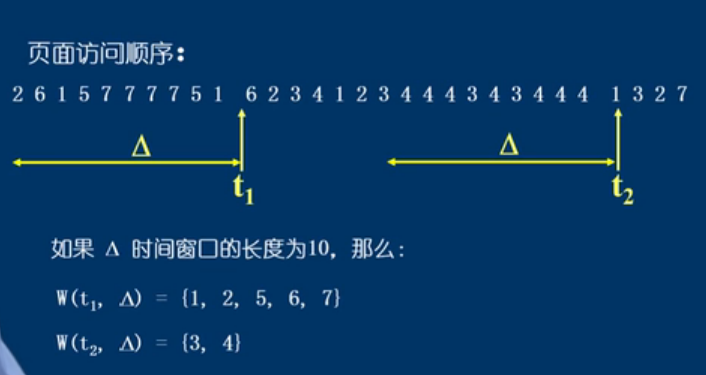
对于FIFO而言，物理页帧的数量/大小，会明显的影响效果。





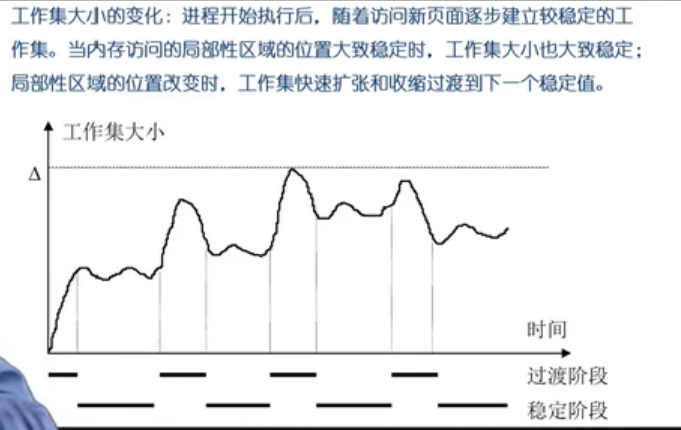
时间段是（t-△, t）（从当前往过去推△工作集窗口）

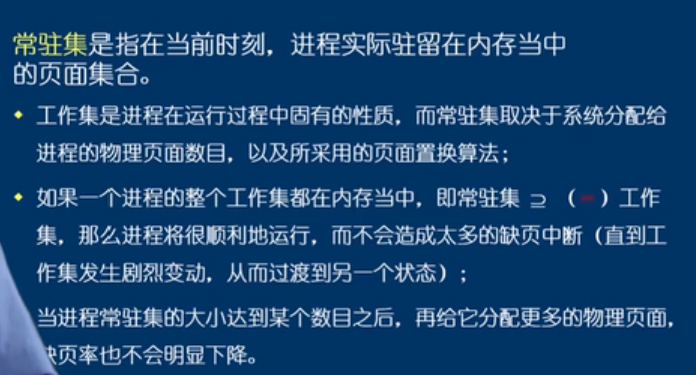
例子：



从t2之前推，所以没算t2.

对于t2的局部性较强，但是t1只有一定的局部性，比t2弱





而工作集则是程序要访问的页面，可能不在内存中，触发缺页。我们希望工作集能都是常驻集。

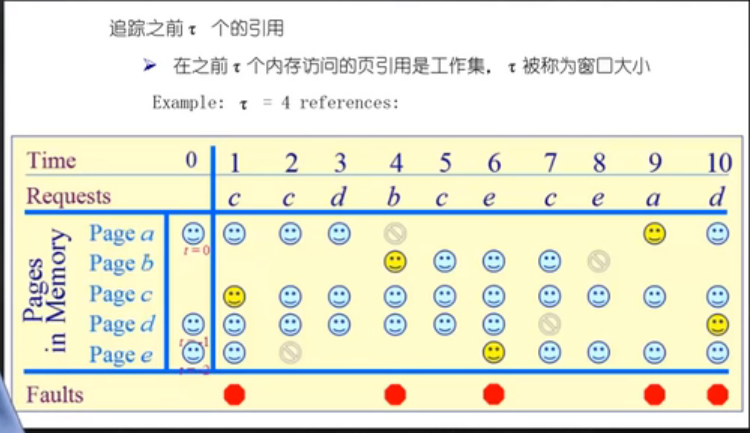
全局页面置换算法：

工作集页置换算法：

随着程序的执行，只要**页不在工作集窗口中了就会被丢掉**，不只是在被换出时。

（工作集窗口不是一个物理空间，其实是**一个时间空间**，现在一个时间窗口）

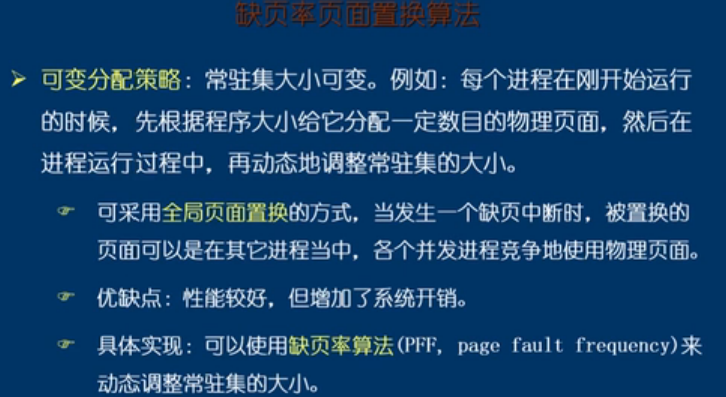
对于该算法的换出不是因为页面置换，而是因为超出了工作集，对于**缺页则会补充在超出工作集的空间**。（**删去超出工作集的页面并不触发缺页**）

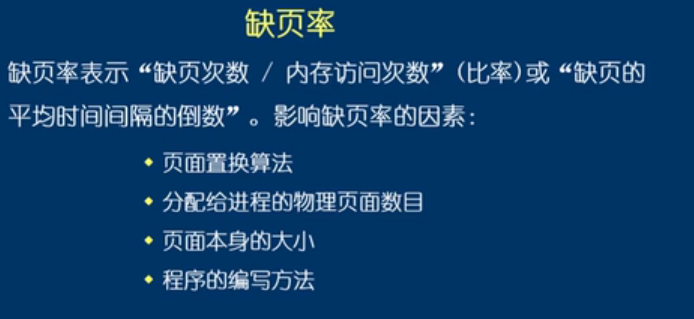


**物理页帧里放哪些也都是有工作集窗口判断的。**

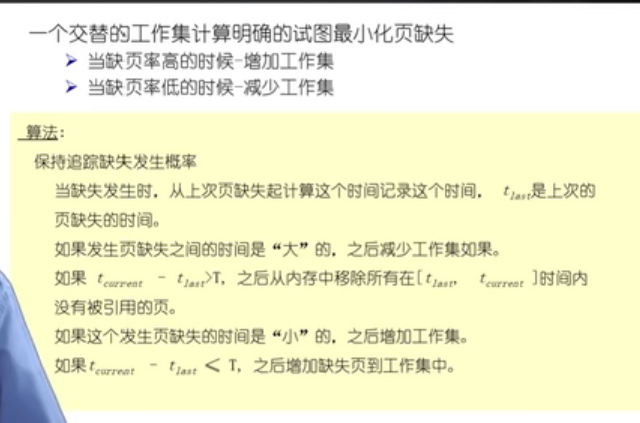
**缺页率页面置换算法：PFF**

工作集窗口可以变化，常驻集（物理内存）也可以变化

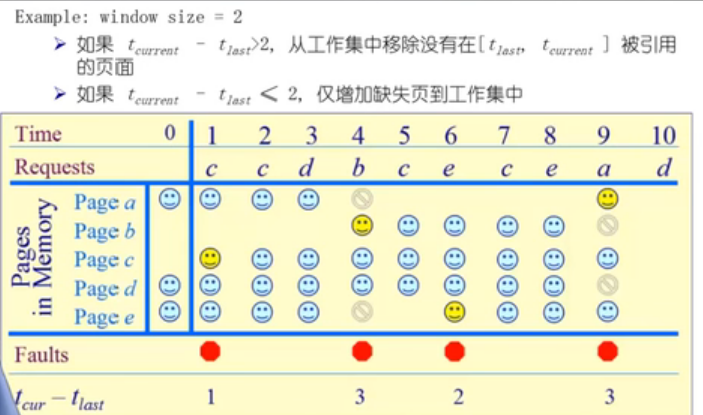




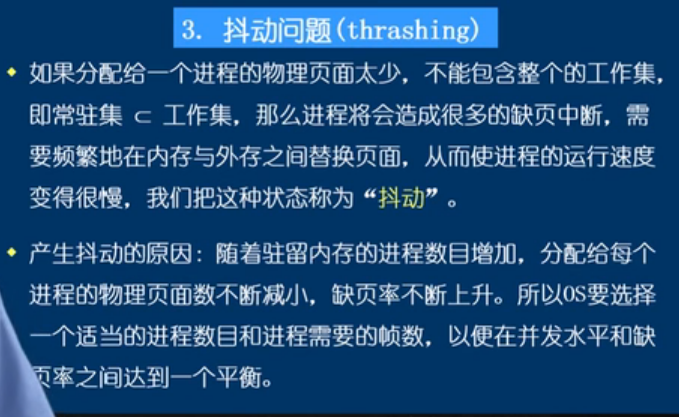
缺页率高便增加工作集，缺页率低则减少工作集，让缺页率保持在一个合理的范围内。



例子：



抖动问题：



找到缺页的平均时间几乎等于缺页的服务时间，整体的利用不错并且能很高的跑很多的程序，高并发性，第二个N值。

