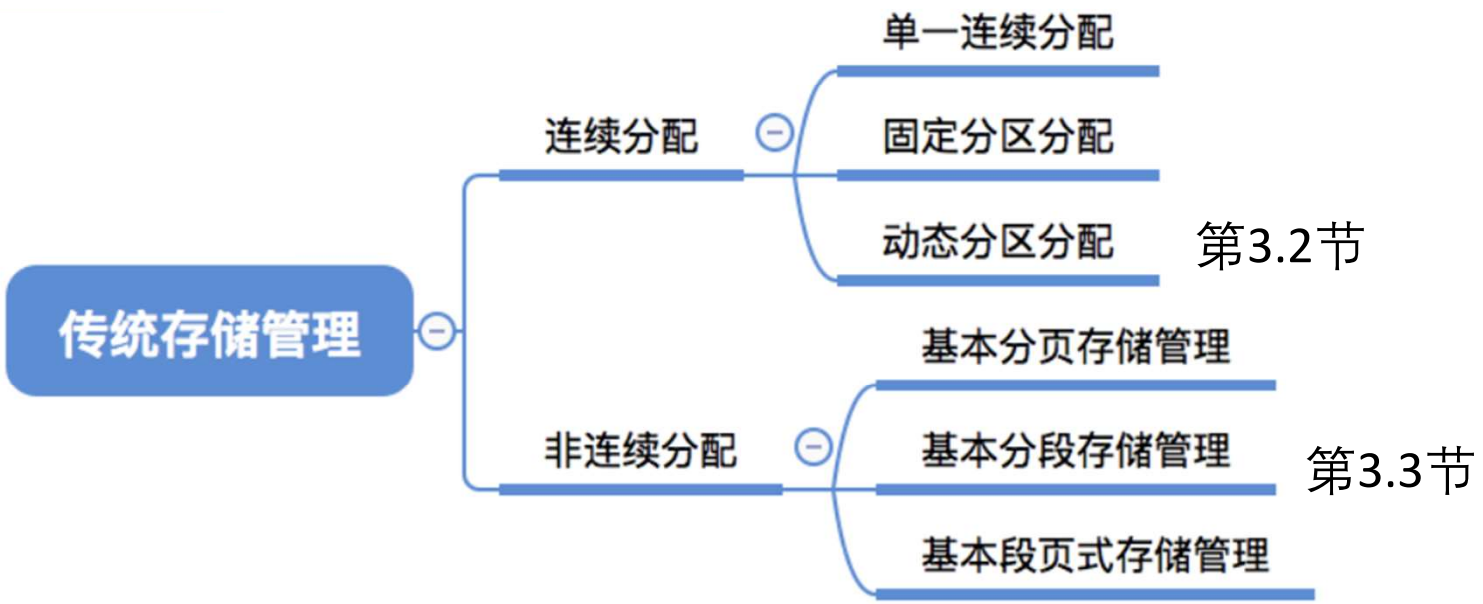
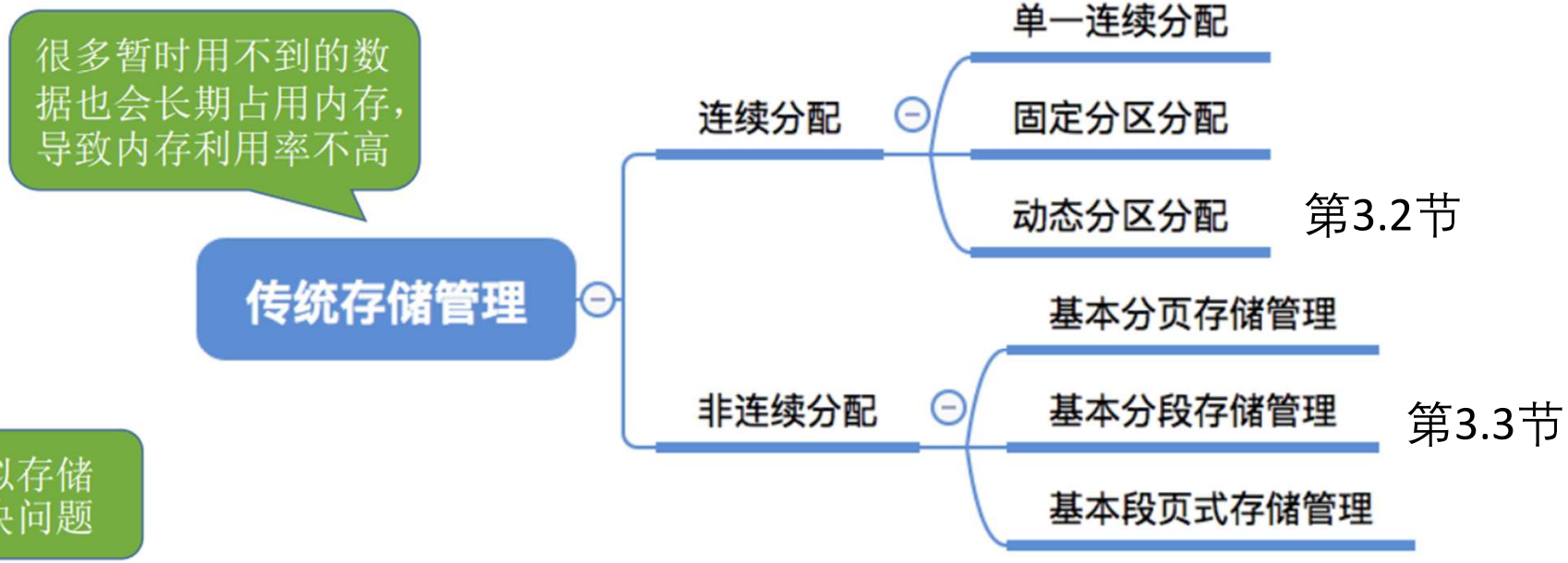
虚拟存储管理机制

传统存储管理方式的特征、缺点

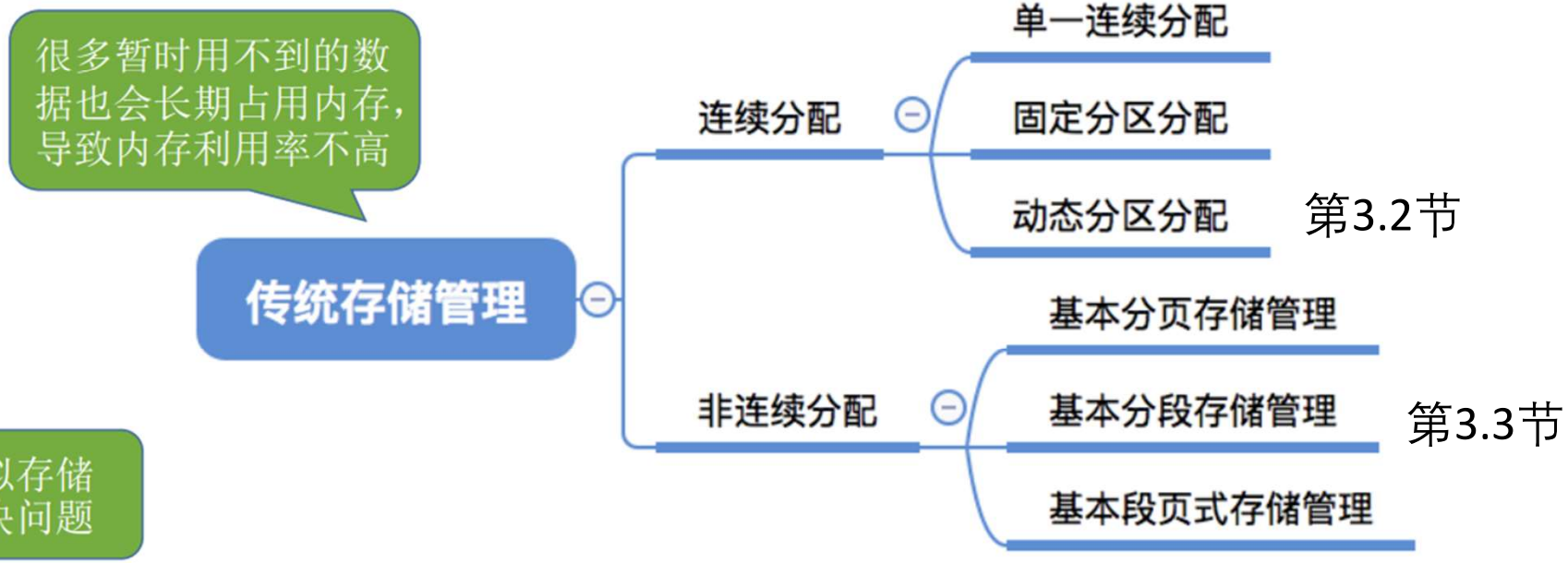


传统存储管理方式的特征、缺点



一次性：作业必须一次性全部装入内存后才能开始运行。这会造成两个问题：①作业很大时，不能全部装入内存，导致大作业无法运行；②当大量作业要求运行时，由于内存无法容纳所有作业，因此只有少量作业能运行，导致多道程序并发度下降。

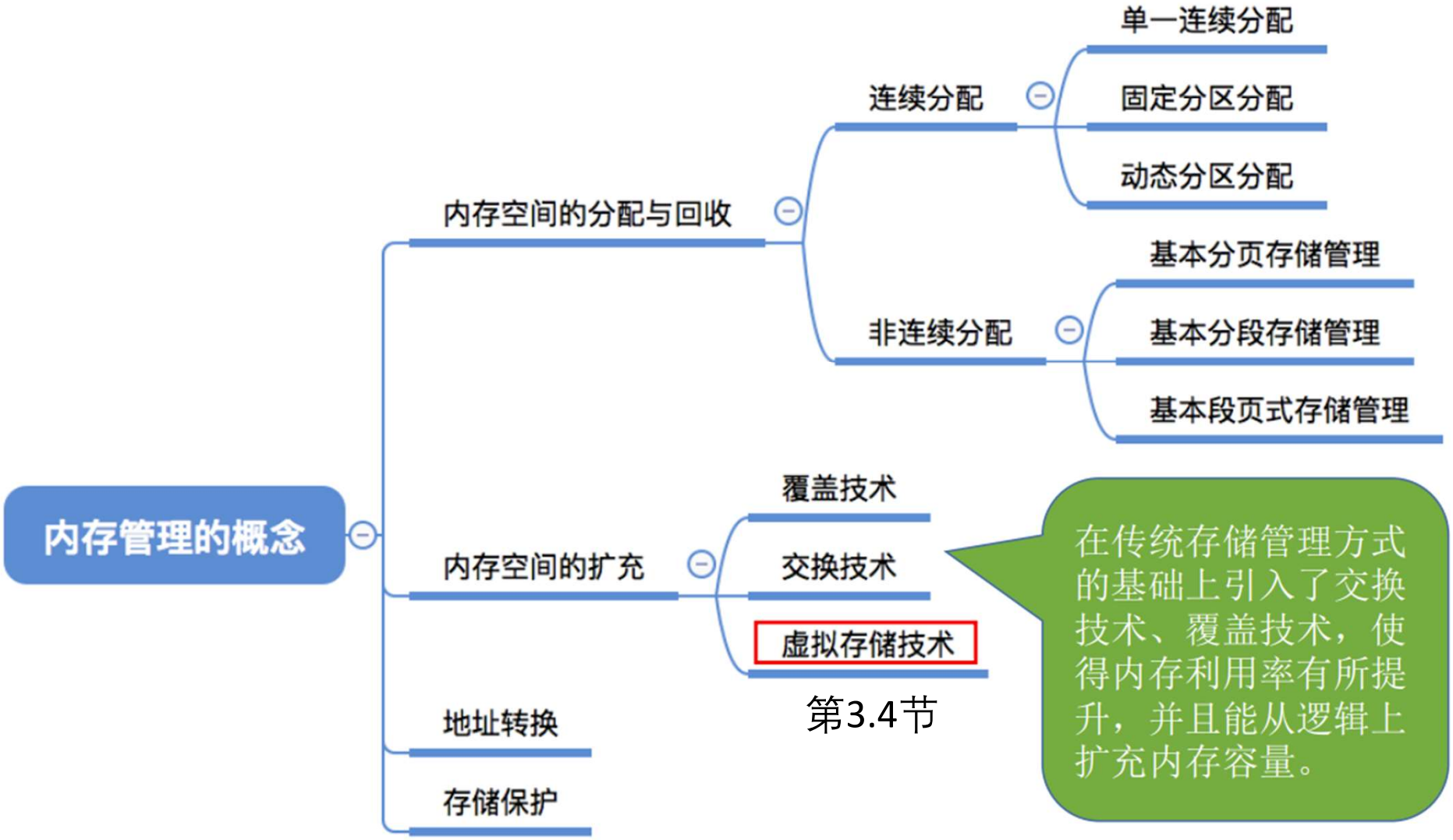
传统存储管理方式的特征、缺点



一次性：作业必须一次性全部装入内存后才能开始运行。这会造成两个问题：①作业很大时，不能全部装入内存，导致大作业无法运行；②当大量作业要求运行时，由于内存无法容纳所有作业，因此只有少量作业能运行，导致多道程序并发度下降。

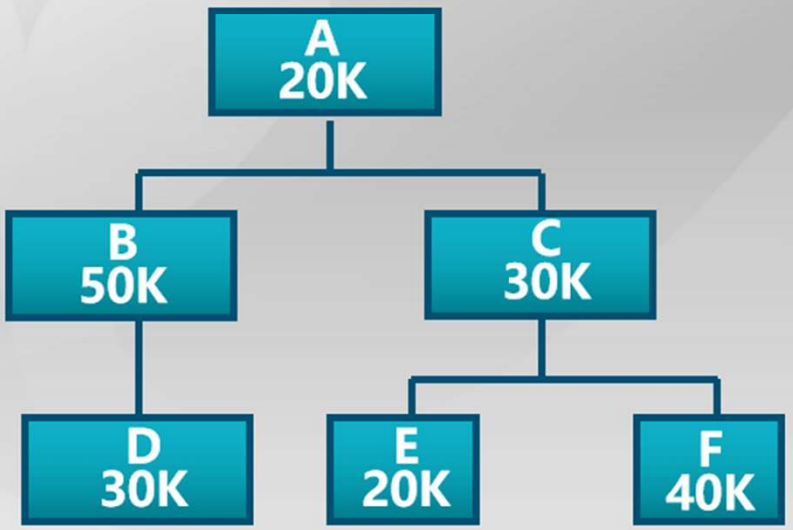
驻留性：一旦作业被装入内存，就会一直驻留在内存中，直至作业运行结束。事实上，在一个时间段内，只需要访问作业的一小部分数据即可正常运行，这就导致了内存中会驻留大量的、暂时用不到的数据，浪费了宝贵的内存资源。

知识总览

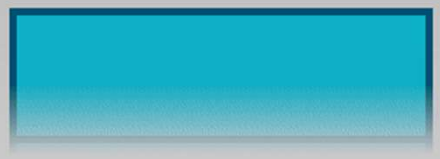

内存管理的概念
内存空间的分配与回收
连续分配
单一连续分配
固定分区分配
动态分区分配
非连续分配
基本分页存储管理
基本分段存储管理
基本段页式存储管理
内存空间的扩充
覆盖技术
交换技术
虚拟存储技术
第3.4节
在传统存储管理方式的基础上引入了交换技术、覆盖技术，使得内存利用率有所提升，并且能从逻辑上扩充内存容量。
地址转换
存储保护

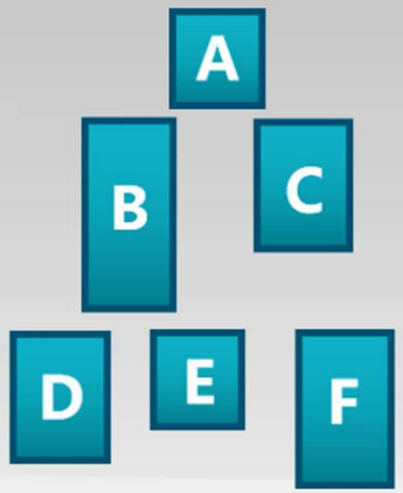
覆盖技术示例

程序调用结构：190K



内存总共：110K





应用程序**手动**把需要的指令和数据保存在内存中

交换技术

内存

外存

操作系统**自动**把暂时不能执行的程序保存到外存中

- 覆盖
  - 只能发生在没有调用关系的模块间
  - 程序员须给出模块间的逻辑覆盖结构
  - 发生在运行程序的内部模块间
- 交换
  - 以进程为单位
  - 不需要模块间的逻辑覆盖结构
  - 发生在内存进程间

https://blog.csdn.net/Chahot

虚拟存储需求

- 计算机系统时常出现**内存空间不够用**
  - 覆盖（overlay）
    应用程序**手动**把需要的指令和数据保存在内存中
  - 交换（swapping）
    操作系统**自动**把暂时不能执行的程序保存到外存中
  - 虚拟存储
    在有限容量的内存中，以页为单位**自动**装入**更多更大**的程序

https://blog.csdn.net/

虚拟存储管理机制

本讲内容

1. 程序访问局部性原理
2. 虚拟存储器基本原理
3. 分页式虚拟存储管理
4. 典型的页面置换算法
5. 分段式虚拟存储管理

程序访问局部性原理

1 基本思想

- 虚拟存储技术基于局部性原理
- P.Denning研究了程序执行时的局部性原理
- 一段时间内程序的执行呈现出高度局部性

局部性原理

时间局部性：如果执行了程序中的某条指令，那么不久后这条指令很有可能再次执行；如果某个数据被访问过，不久之后该数据很可能再次被访问。（因为程序中存在大量的循环）

```
int i = 0;
int a[100];
while (i < 100) {
  a[i] = i;
  i++;
}
```

局部性原理

时间局部性：如果执行了程序中的某条指令，那么不久后这条指令很有可能再次执行；如果某个数据被访问过，不久之后该数据很可能再次被访问。（因为程序中存在大量的循环）

空间局部性：一旦程序访问了某个存储单元，在不久之后，其附近的存储单元也很有可能被访问。（因为很多数据在内存中都是连续存放的，并且程序的指令也是顺序地在内存中存放的）

```
int i = 0;
int a[100];
while (i < 100) {
  a[i] = i;
  i++;
}
```

虚拟存储管理机制

本讲内容

1. 程序访问局部性原理
2. 虚拟存储器基本原理
3. 分页式虚拟存储管理
4. 典型的页面置换算法
5. 分段式虚拟存储管理

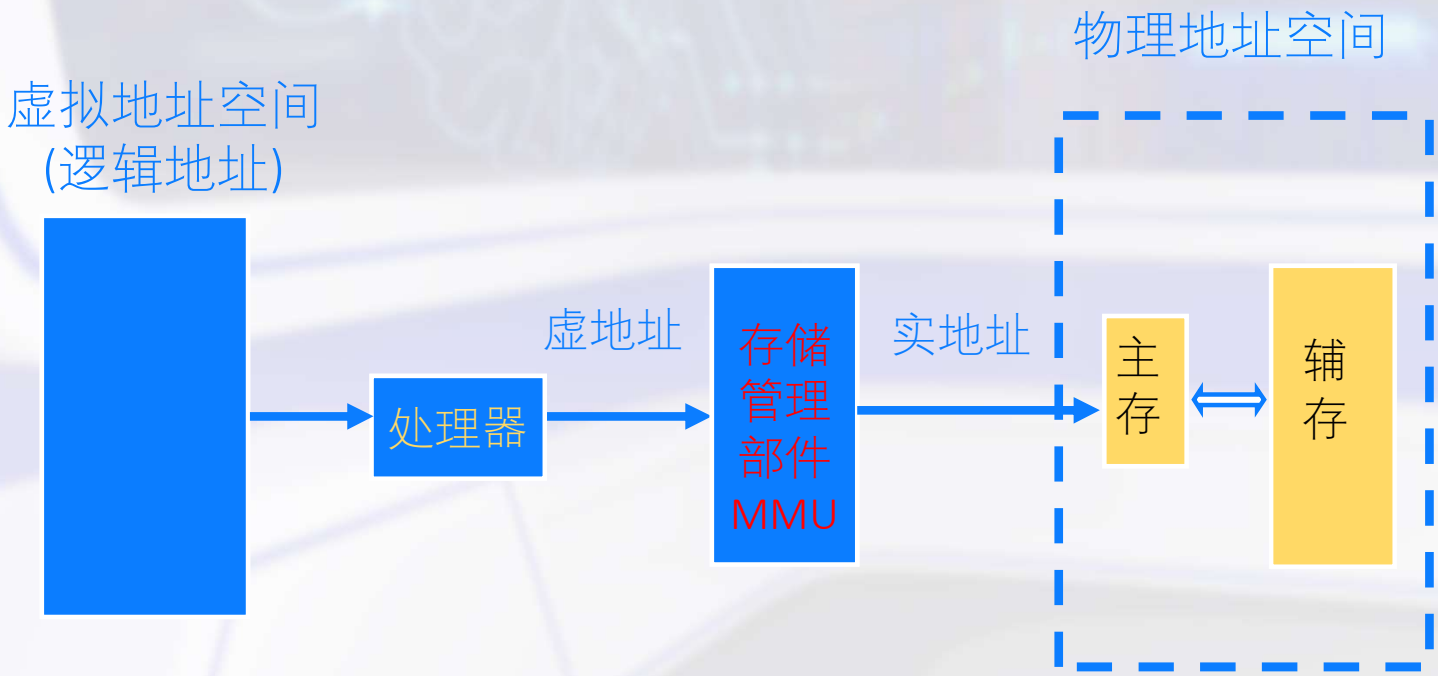
虚拟存储器基本原理

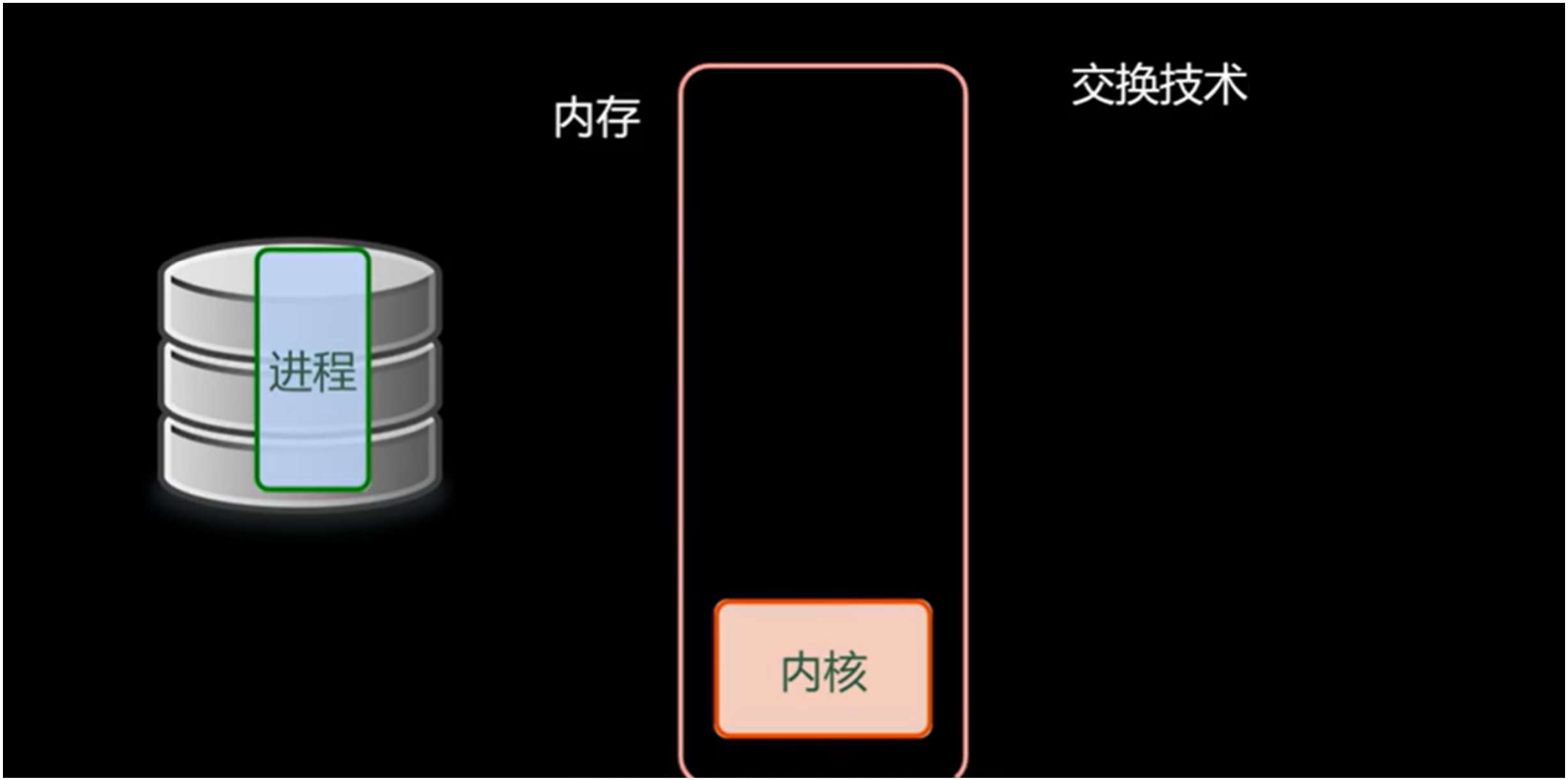
1 虚拟存储空间

- 虚拟存储器是由操作系统提供的假想存储器，它不是实际内存，而是系统对物理内存的逻辑扩充。
- 程序、数据、堆栈可以超过内存的大小，系统把当前使用的部分保留在内存，而把其它部分保存在辅存上，需要时在内存和辅存之间动态交换
- 虚拟存储空间受限于系统地址结构、内存大小及可用的辅存容量

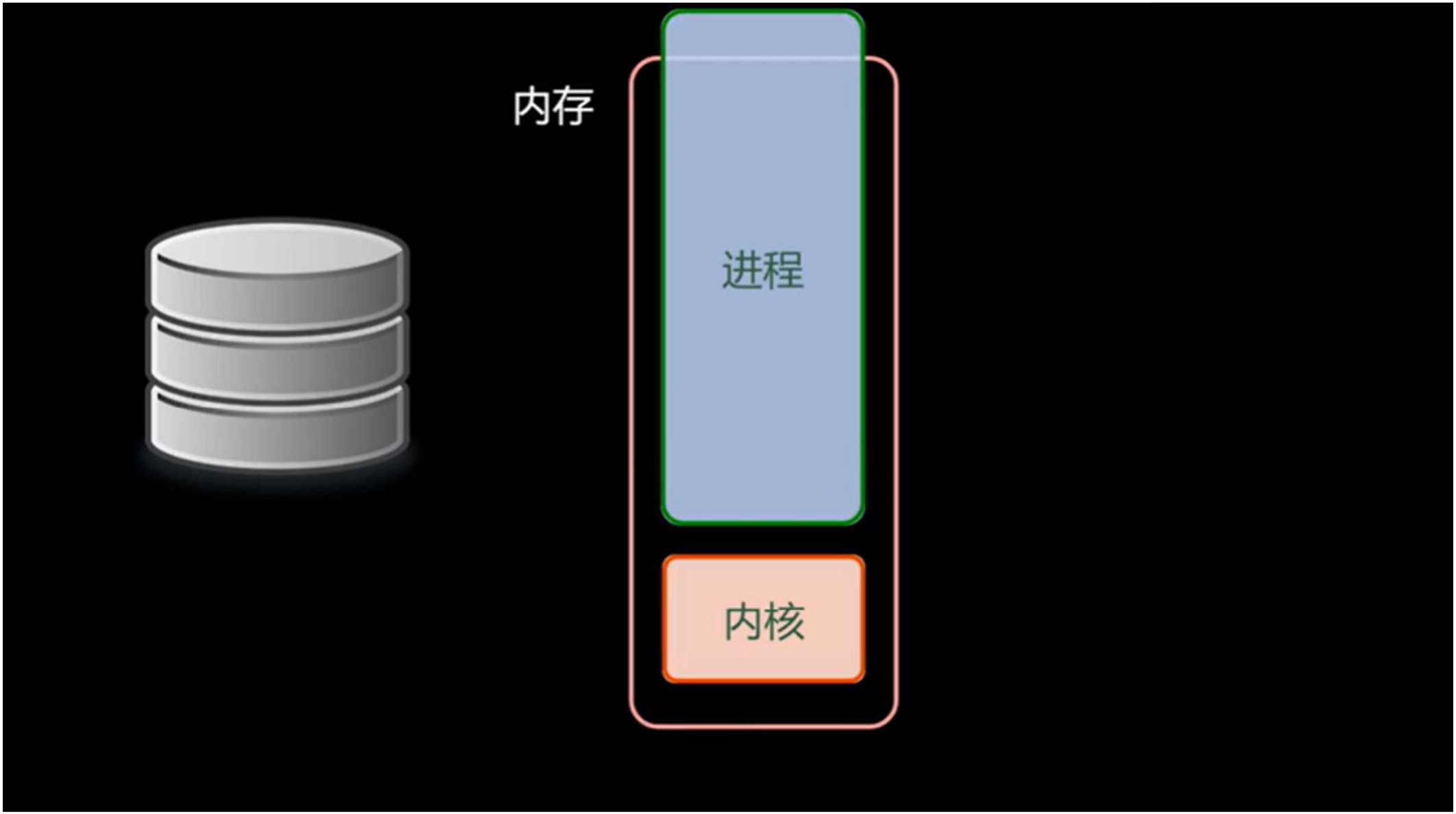
虚拟存储器基本原理

2 虚拟存储系统

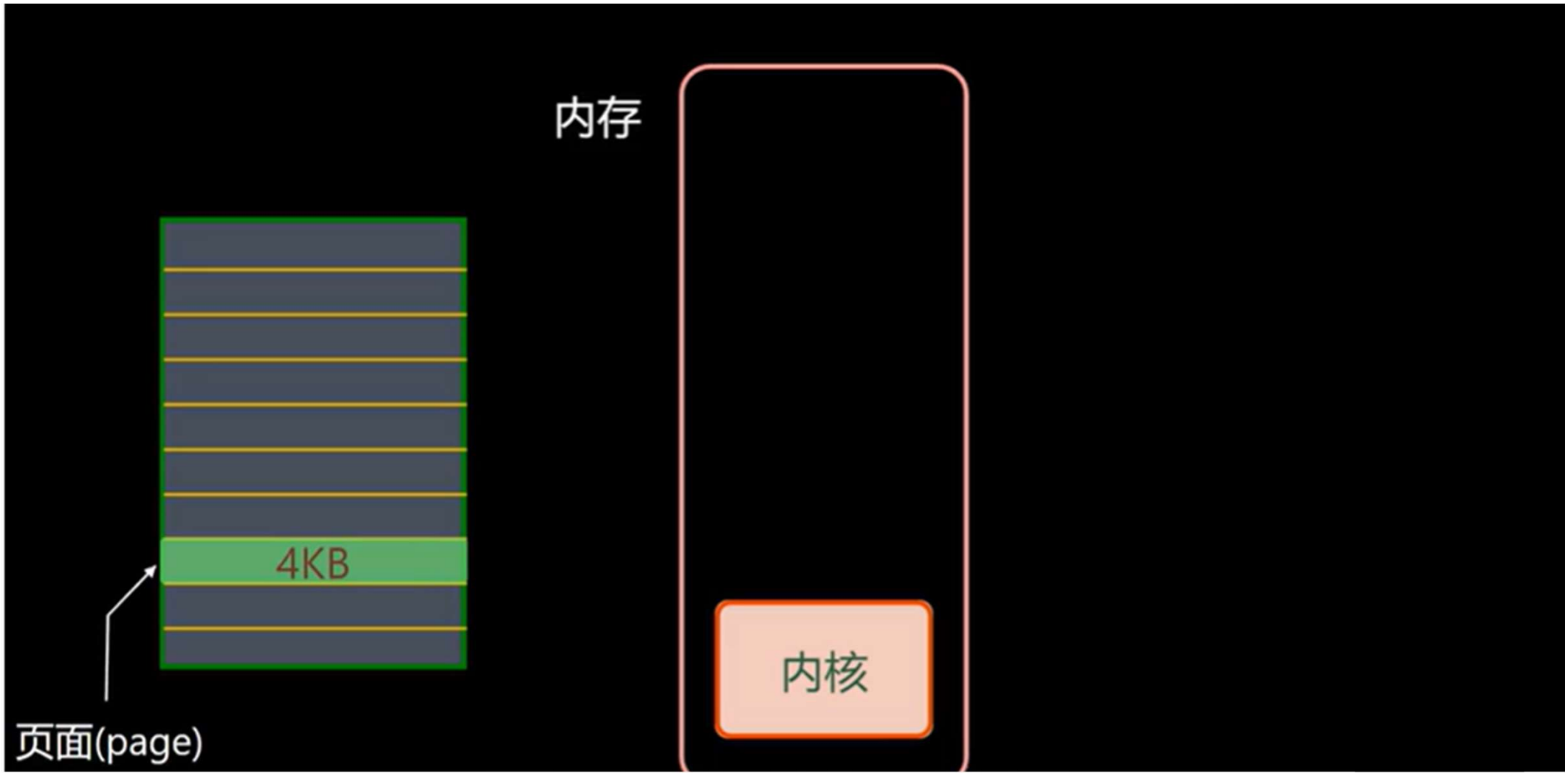




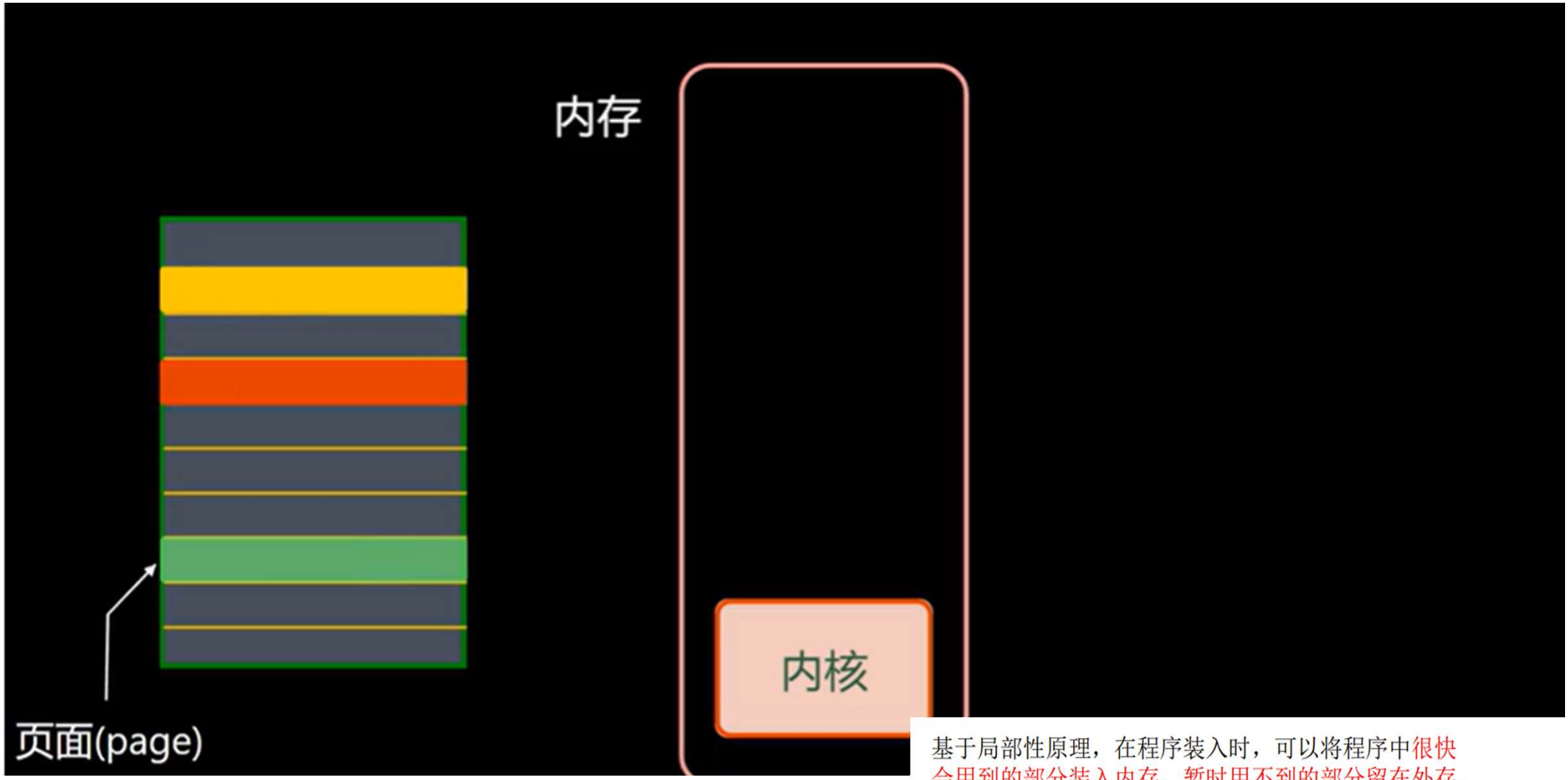
- 把整个进程从内存复制到磁盘中会花费相当多的时间和开销



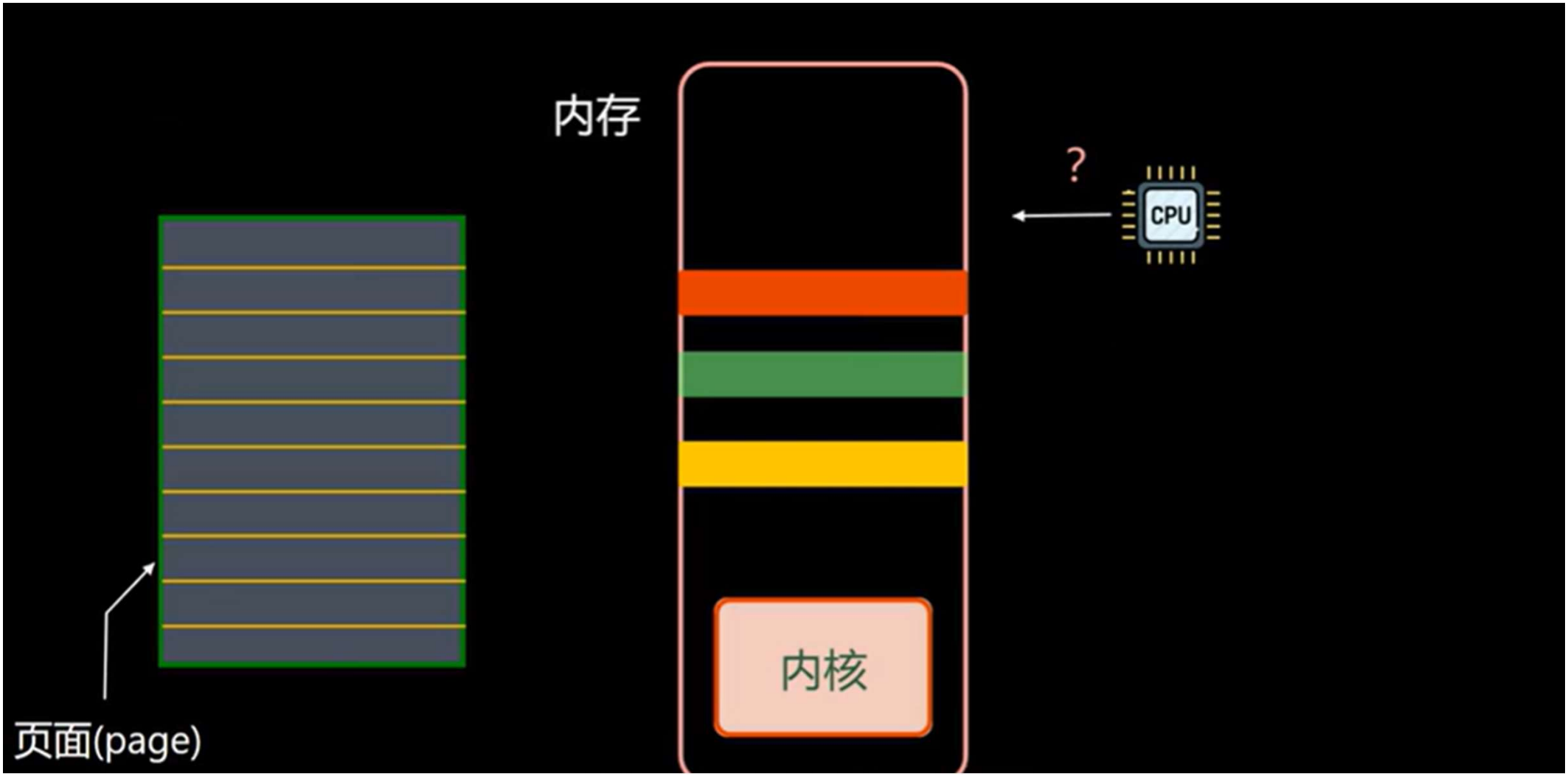
- 进程会随着数据增多而碰撞，超过内存大小，此时交换技术无法解决

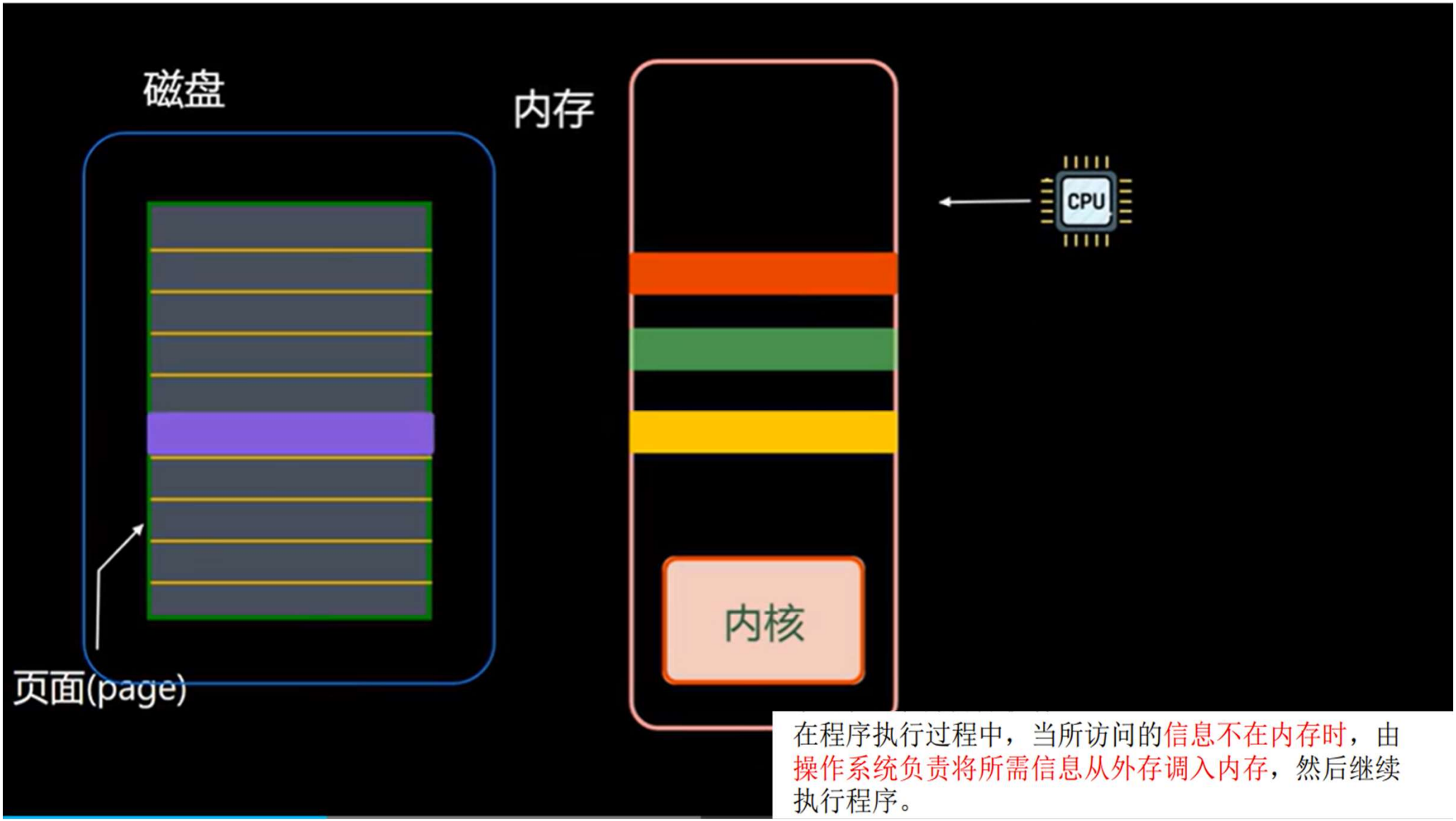


- 更好的做法就是把进程切碎，分成一个个的小单元页面
- 其中只有一部分的页面会存在内存中

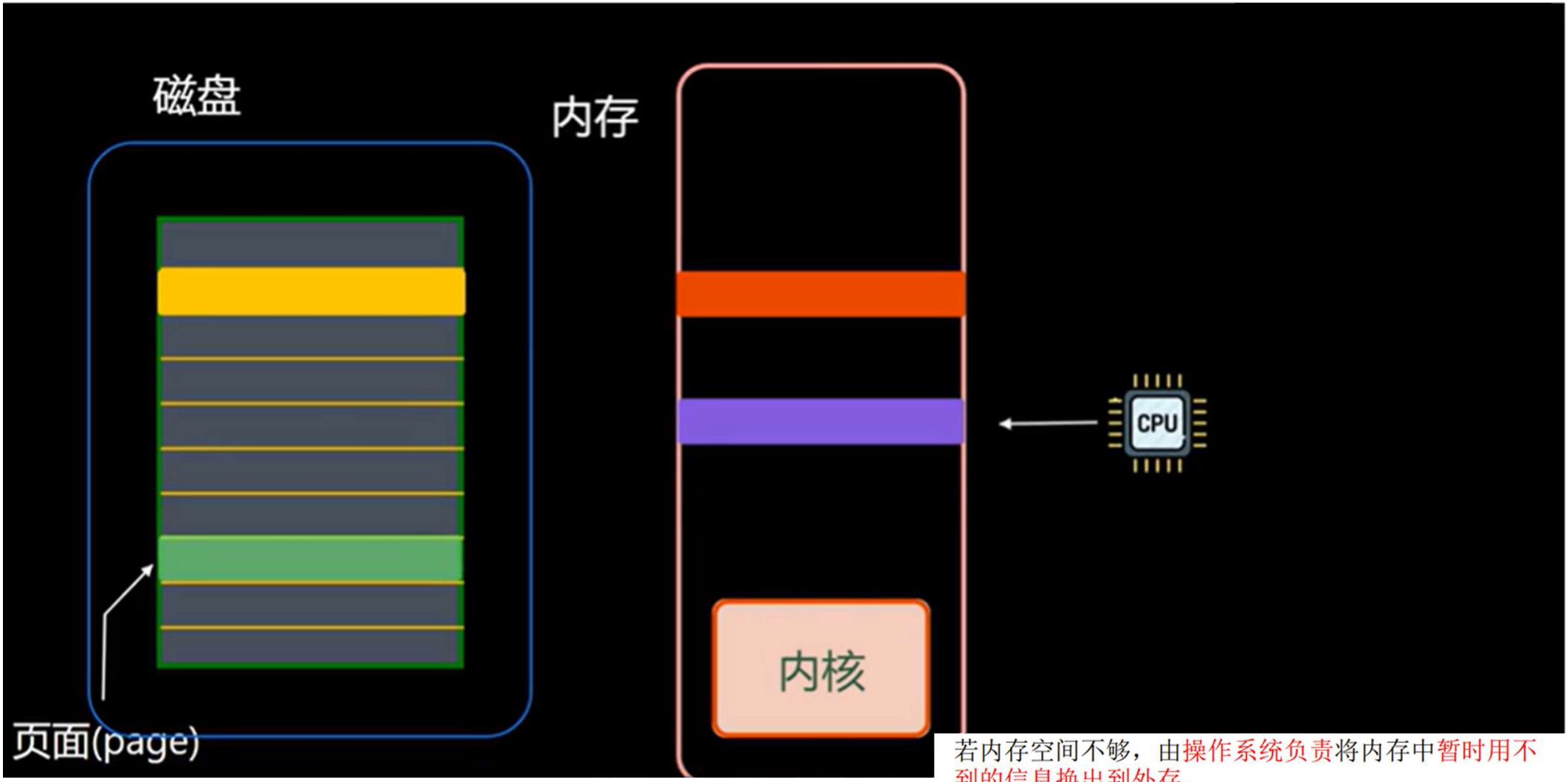


基于局部性原理，在程序装入时，可以将程序中很快会用到的部分装入内存，暂时用不到的部分留在外存，就可以让程序开始执行。

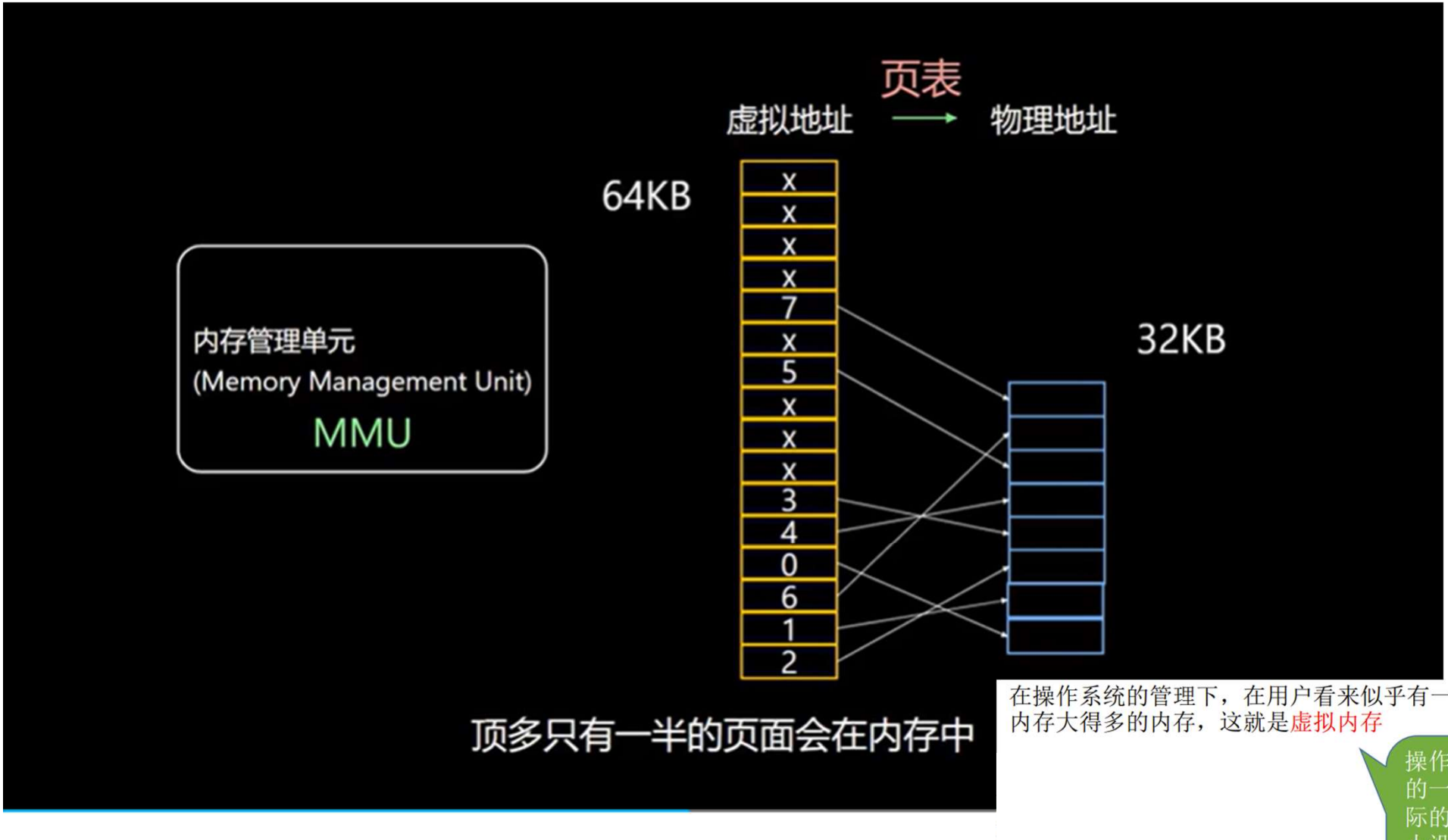

内存
?
CPU
内核
页面(page)



在程序执行过程中，当所访问的信息不在内存时，由操作系统负责将所需信息从外存调入内存，然后继续执行程序。



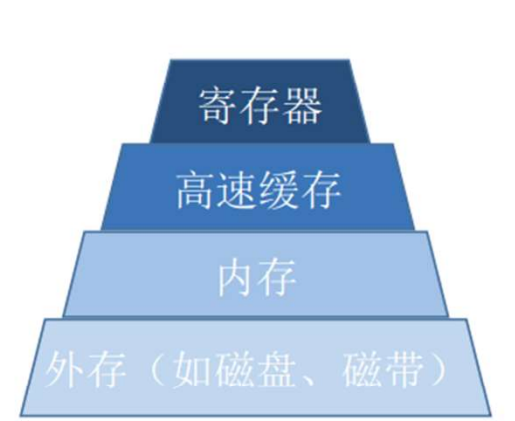
若内存空间不够，由操作系统负责将内存中暂时用不到的信息换出到外存。



在操作系统的管理下，在用户看来似乎有一个比实际内存大得多的内存，这就是虚拟内存

操作系统虚拟性的一个体现，实际的物理内存大小没有变，只是在逻辑上进行了扩充。

虚拟内存的定义和特征



计算机中存储器的层次结构

容量小
速度快
成本高

容量大
速度慢
成本低

基于局部性原理，在程序装入时，可以将程序中很快会用到的部分装入内存，暂时用不到的部分留在外存，就可以让程序开始执行。

在程序执行过程中，当所访问的信息不在内存时，由操作系统负责将所需信息从外存调入内存，然后继续执行程序。

若内存空间不够，由操作系统负责将内存中暂时用不到的信息换出到外存。

在操作系统的管理下，在用户看来似乎有一个比实际内存大得多的内存，这就是虚拟内存

操作系统虚拟性的一个体现，实际的物理内存大小没有变，只是在逻辑上进行了扩充。

易混知识点：

虚拟内存的最大容量是由计算机的地址结构（CPU寻址范围）确定的

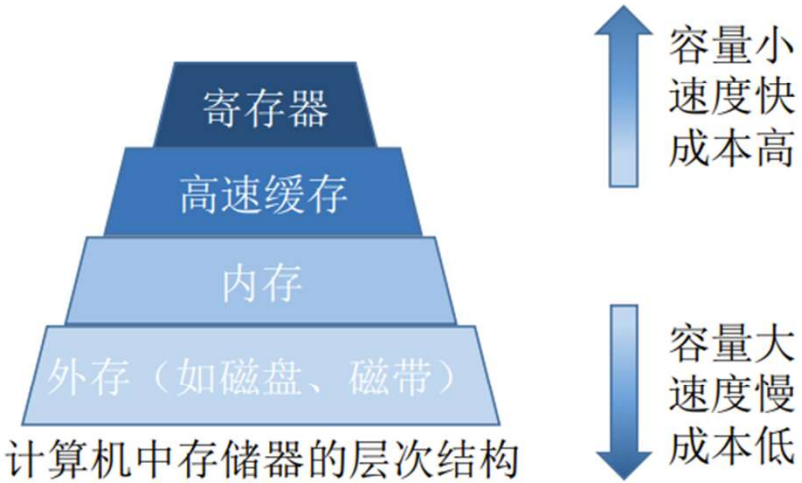
虚拟内存的实际容量 = min（内存和外存容量之和，CPU寻址范围）

如：某计算机地址结构为32位，按字节编址，内存大小为512MB，外存大小为2GB。

则虚拟内存的最大容量为 2^{32} B = 4GB

虚拟内存的实际容量 = min (2^{32}B, 512MB+2GB) = 2GB+512MB

虚拟内存的定义和特征



计算机中存储器的层次结构

虚拟内存有一下三个主要特征：

多次性：无需在作业运行时一次性全部装入内存，而是允许被分成多次调入内存。

对换性：在作业运行时无需一直常驻内存，而是允许在作业运行过程中，将作业换入、换出。

虚拟性：从逻辑上扩充了内存的容量，使用户看到的内存容量，远大于实际的容量。

覆盖技术和虚拟内存区别

- 覆盖程序段的最大长度要受内存容量大小的限制，而虚拟存储器中程序的最大长度不受内存容量的限制，只受计算机地址结构的限制。
- 覆盖技术中的覆盖段由程序员设计，且要求覆盖段中的各个覆盖具有相对独立性，不存在直接联系或相互交叉访问；而虚拟存储技术对用户的程序段之间没有这种要求。

交换技术和虚拟内存区别

- 交换技术是在不同的进程（作业）间的，虚拟存储技术是在一个作业间的。
- 交换技术需要导出整个程序到硬盘和导入内存，而虚存技术只把程序的一部分数据导入导出硬盘，减少交换技术的开销。
- 交换技术是以进程为单位，若进程所需内存大于系统内存，则此进程无法进行。而虚拟存储是以页或段为单位，是把进程再分为页或段对内存进行分化，若进程所需内存大于系统内存，进程也可以运行，因为该进程的一部分可换到外存上。

虚拟存储管理机制

本讲内容

1. 程序访问局部性原理
2. 虚拟存储器基本原理
3. 分页式虚拟存储管理
4. 典型的页面置换算法
5. 分段式虚拟存储管理

分页式虚拟存储管理

1 基本原理

- 进程运行前，不装入全部页面，而是装入一个或数个页面 (根据局部性原理)
- 根据进程运行的需要，动态装入其它页面
- 内存空间已满，又需要装入新页面时，则淘汰内存部分页面，装入新的页面)
- 这种淘汰装入，叫做页面置换

分页式虚拟存储管理

2 关键问题

- 系统如何获知进程当前所需页面不在内存中
- 发现缺页时，如何把所缺页面调入内存
- 需要淘汰页面时，根据什么策略选择淘汰页面

分页式虚拟存储管理

3 页表扩充

页号	中断位	页框号	辅存地址	访问位	修改位

- 页号和页框号——其作用与分页存储管理相同
- 中断位——指示页面是否在内存中，若不在内存，则产生缺页中断
- 访问位——记录该页是否被访问或者访问次数
- 修改位——指示页面在调入内存后是否被修改过
- 辅存地址——指出该页在辅存上的地址

虚拟存储管理机制

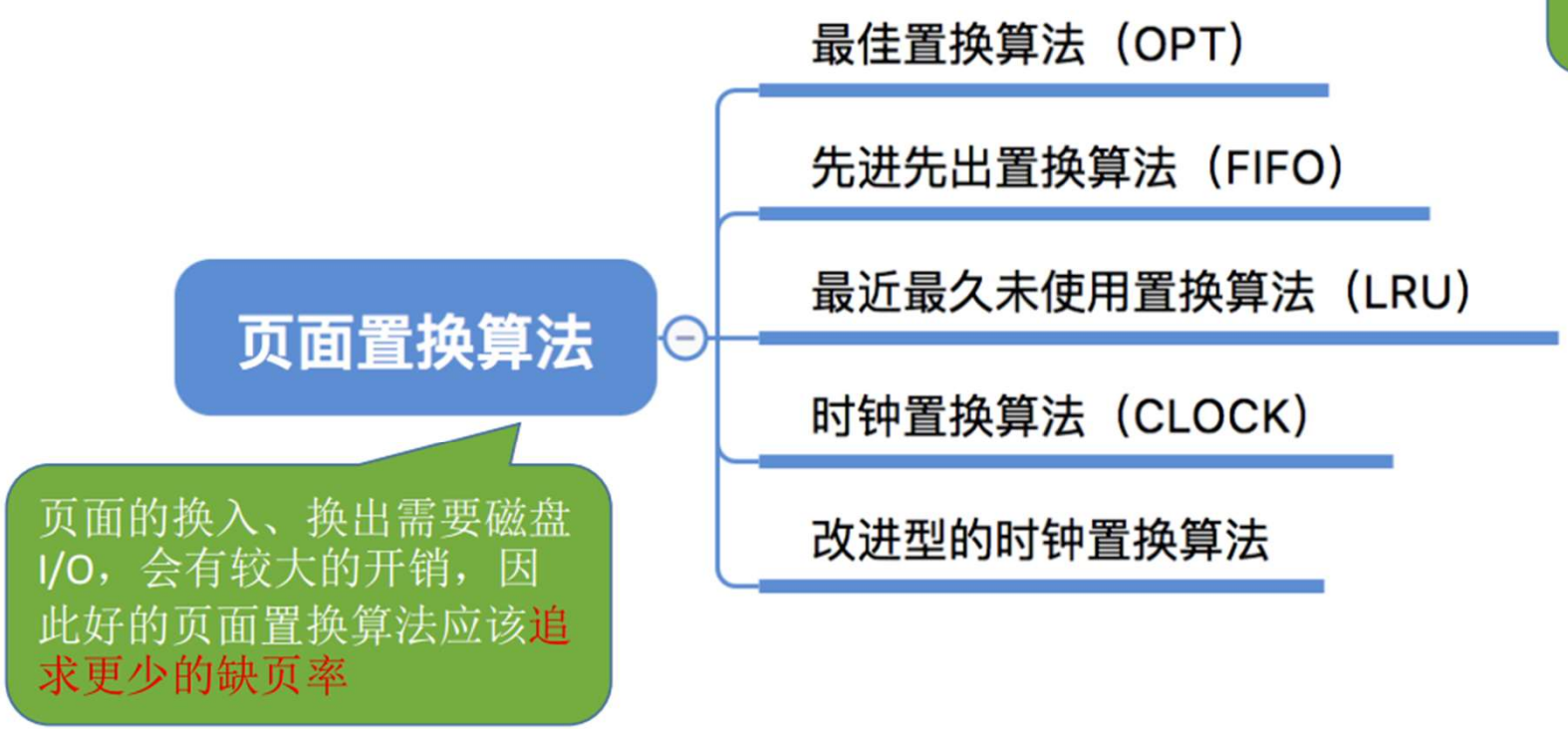
本讲内容

1. 程序访问局部性原理
2. 虚拟存储器基本原理
3. 分页式虚拟存储管理
4. 典型的页面置换算法
5. 分段式虚拟存储管理

知识总览

请求分页存储管理与基本分页存储管理的主要区别：
在程序执行过程中，当所访问的信息不在内存时，由操作系统负责将所需信息从外存调入内存，然后继续执行程序。
若内存空间不够，由操作系统负责将内存中暂时用不到的信息换出到外存。

用页面置换算法决定应该换出哪个页面



典型的页面置换算法

1 性能指标

进程P在运行中成功的内存访问次数为*S*，不成功的访问次数为*F*，则缺页中断率为：

$$R=F/(S+F)$$

越低越好

- 分配进程页框数
- 页框本身大小
- 程序的编制方法
- 页面置换算法

典型的页面置换算法

2 理想页面置换算法

- 调入新的一页面而必须淘汰一个旧页时，淘汰页是以后不再访问或最长时间以后再访问的页面
- 可作为衡量各种实用页面置换算法的标准

缺页中断率

最佳置换算法（OPT）

最佳置换算法（OPT，Optimal）：每次选择淘汰的页面将是以后永不使用，或者在最长时间内不再被访问的页面，这样可以保证最低的缺页率。

例：假设系统为某进程分配了三个内存块，并考虑到有一下页面号引用串（会依次访问这些页面）：
7, 0, 1, 2, 0, 3, 0, 4, 2, 3, 0, 3, 2, 1, 2, 0, 1, 7, 0, 1

访问页面	7	0	1	2	0	3	0	4	2	3	0	3	2	1	2	0	1	7	0	1
内存块1	7	7	7	2		2		2			2			2				7		
内存块2		0	0	0		0		4			0			0				0		
内存块3			1	1		3		3			3			1				1		
是否缺页	√	√	√	√		√		√			√			√				√		

最佳置换算法（OPT）

最佳置换算法（OPT，Optimal）：每次选择淘汰的页面将是以后永不使用，或者在最长时间内不再被访问的页面，这样可以保证最低的缺页率。

例：假设系统为某进程分配了三个内存块，并考虑到有一下页面号引用串（会依次访问这些页面）：
7, 0, 1, 2, 0, 3, 0, 4, 2, 3, 0, 3, 2, 1, 2, 0, 1, 7, 0, 1

访问页面	7	0	1	2	0	3	0	4	2	3	0	3	2	1	2	0	1	7	0	1
内存块1	7	7	7	2		2		2			2			2				7		
内存块2		0	0	0		0		4			0			0				0		
内存块3			1	1		3		3			3			1				1		
是否缺页	√	√	√	√		√		√			√			√				√		

选择从0，1，7中淘汰一页。按最佳置换的规则，往后寻找，最后一个出现的页号就是要淘汰的页面

最佳置换算法（OPT）

最佳置换算法（OPT，Optimal）：每次选择淘汰的页面将是以后永不使用，或者在最长时间内不再被访问的页面，这样可以保证最低的缺页率。

例：假设系统为某进程分配了三个内存块，并考虑到有一下页面号引用串（会依次访问这些页面）：
7, 0, 1, 2, 0, 3, 0, 4, 2, 3, 0, 3, 2, 1, 2, 0, 1, 7, 0, 1

访问页面	7	0	1	2	0	3	0	4	2	3	0	3	2	1	2	0	1	7	0	1
内存块1	7	7	7	2		2		2			2			2				7		
内存块2		0	0	0		0		4			0			0				0		
内存块3			1	1		3		3			3			1				1		
是否缺页	√	√	√	√		√		√			√			√				√		

选择从0，1，7中淘汰一页。按最佳置换的规则，往后寻找，最后一个出现的页号就是要淘汰的页面

最佳置换算法（OPT）

最佳置换算法（OPT，Optimal）：每次选择淘汰的页面将是以后永不使用，或者在最长时间内不再被访问的页面，这样可以保证最低的缺页率。

例：假设系统为某进程分配了三个内存块，并考虑到有一下页面号引用串（会依次访问这些页面）：
7, 0, 1, 2, 0, 3, 0, 4, 2, 3, 0, 3, 2, 1, 2, 0, 1, 7, 0, 1

访问页面	7	0	1	2	0	3	0	4	2	3	0	3	2	1	2	0	1	7	0	1
内存块1	7	7	7	2		2		2			2			2				7		
内存块2		0	0	0		0		4			0			0				0		
内存块3			1	1		3		3			3			1				1		
是否缺页	√	√	√	√		√		√			√			√				√		

(打钩的位置)　　(沾满之后，淘汰换页)

选择从0，1，7中淘汰一页。按最佳置换的规则，往后寻找，最后一个出现的页号就是要淘汰的页面

整个过程缺页中断发生了9次，页面置换发生了6次。
注意：缺页时未必发生页面置换。若还有可用的空闲内存块，就不用进行页面置换。

缺页率 = 9/20 = 45%

总访问页面次数20

最佳置换算法（OPT）

最佳置换算法（OPT，Optimal）：每次选择淘汰的页面将是以后永不使用，或者在最长时间内不再被访问的页面，这样可以保证最低的缺页率。

最佳置换算法可以保证最低的缺页率，但实际上，只有在进程执行的过程中才能知道接下来会访问到的是哪个页面。操作系统无法提前预判页面访问序列。因此，最佳置换算法是无法实现的。

典型的页面置换算法

3 FIFO页面置换算法

- ✓ 选择内存中驻留时间最长的页面并淘汰之
- ✓ 部分程序总体按线性顺序来访问物理空间

优点: 直观，实现简单

缺点: 与进程实际运行的规律不相适应，性能较差，可能出现分配页框数增加、缺页次数反而增加的异常现象

先进先出置换算法（FIFO）

先进先出置换算法（FIFO）：每次选择淘汰的页面是最早进入内存的页面
实现方法：把调入内存的页面根据调入的先后顺序排成一个队列，需要换出页面时选择队头页面即可。
队列的最大长度取决于系统为进程分配了多少个内存块。
例：假设系统为某进程分配了三个内存块，并考虑到有以下页面号引用串：

3, 2, 1, 0, 3, 2, 4, 3, 2, 1, 0, 4

访问页面	3	2	1	0	3	2	4	3	2	1	0	4
内存块1	3	3	3	0	0	0	4			4	4	
内存块2		2	2	2	3	3	3			1	1	
内存块3			1	1	1	2	2			2	0	
是否缺页	√	√	√	√	√	√	√			√	√	

先进先出置换算法（FIFO）

先进先出置换算法（FIFO）：每次选择淘汰的页面是最早进入内存的页面
实现方法：把调入内存的页面根据调入的先后顺序排成一个队列，需要换出页面时选择队头页面即可。
队列的最大长度取决于系统为进程分配了多少个内存块。
例：假设系统为某进程分配了三个内存块，并考虑到有以下页面号引用串：

3, 2, 1, 0, 3, 2, 4, 3, 2, 1, 0, 4

访问页面	3	2	1	0	3	2	4	3	2	1	0	4
内存块1	3	3	3	0	0	0	4			4	4	
内存块2		2	2	2	3	3	3			1	1	
内存块3			1	1	1	2	2			2	0	
是否缺页	√	√	√	√	√	√	√			√	√	

先进先出置换算法（FIFO）

先进先出置换算法（FIFO）：每次选择淘汰的页面是最早进入内存的页面
实现方法：把调入内存的页面根据调入的先后顺序排成一个队列，需要换出页面时选择队头页面即可。
队列的最大长度取决于系统为进程分配了多少个内存块。
例：假设系统为某进程分配了三个内存块，并考虑到有以下页面号引用串：

3, 2, 1, 0, 3, 2, 4, 3, 2, 1, 0, 4

访问页面	3	2	1	0	3	2	4	3	2	1	0	4
内存块1	3	3	3	0	0	0	4			4	4	
内存块2		2	2	2	3	3	3			1	1	
内存块3			1	1	1	2	2			2	0	
是否缺页	√	√	√	√	√	√	√			√	√	

先进先出置换算法（FIFO）

先进先出置换算法（FIFO）：每次选择淘汰的页面是最早进入内存的页面

实现方法：把调入内存的页面根据调入的先后顺序排成一个队列，需要换出页面时选择队头页面即可。

队列的最大长度取决于系统为进程分配了多少个内存块。

例：假设系统为某进程分配了三个内存块，并考虑到有以下页面号引用串：

3, 2, 1, 0, 3, 2, 4, 3, 2, 1, 0, 4

访问页面	3	2	1	0	3	2	4	3	2	1	0	4
内存块1	3	3	3	0	0	0	4			4	4	
内存块2		2	2	2	3	3	3			1	1	
内存块3			1	1	1	2	2			2	0	
是否缺页	√	√	√	√	√	√	√			√	√	

先进先出置换算法（FIFO）

先进先出置换算法（FIFO）：每次选择淘汰的页面是最早进入内存的页面

实现方法：把调入内存的页面根据调入的先后顺序排成一个队列，需要换出页面时选择队头页面即可。

队列的最大长度取决于系统为进程分配了多少个内存块。

例：假设系统为某进程分配了三个内存块，并考虑到有以下页面号引用串：

3, 2, 1, 0, 3, 2, 4, 3, 2, 1, 0, 4

访问页面	3	2	1	0	3	2	4	3	2	1	0	4
内存块1	3	3	3	0	0	0	4			4	4	
内存块2		2	2	2	3	3	3			1	1	
内存块3			1	1	1	2	2			2	0	
是否缺页	√	√	√	√	√	√	√			√	√	

先进先出置换算法（FIFO）

先进先出置换算法（FIFO）：每次选择淘汰的页面是最早进入内存的页面

实现方法：把调入内存的页面根据调入的先后顺序排成一个队列，需要换出页面时选择队头页面即可。

队列的最大长度取决于系统为进程分配了多少个内存块。

例：假设系统为某进程分配了三个内存块，并考虑到有以下页面号引用串：

3, 2, 1, 0, 3, 2, 4, 3, 2, 1, 0, 4

访问页面	3	2	1	0	3	2	4	3	2	1	0	4
内存块1	3	3	3	0	0	0	4			4	4	
内存块2		2	2	2	3	3	3			1	1	
内存块3			1	1	1	2	2			2	0	
是否缺页	√	√	√	√	√	√	√			√	√	

分配三个内存块时，缺页次数：9次

先进先出置换算法（FIFO）

先进先出置换算法（FIFO）：每次选择淘汰的页面是最早进入内存的页面
实现方法：把调入内存的页面根据调入的先后顺序排成一个队列，需要换出页面时选择队头页面即可。
队列的最大长度取决于系统为进程分配了多少个内存块。
例：假设系统为某进程分配了四个内存块，并考虑到有以下页面号引用串：

3, 2, 1, 0, 3, 2, 4, 3, 2, 1, 0, 4

访问页面	3	2	1	0	3	2	4	3	2	1	0	4
内存块1	3	3	3	3			4	4	4	4	0	0
内存块2		2	2	2			2	3	3	3	3	4
内存块3			1	1			1	1	2	2	2	2
内存块4				0			0	0	0	1	1	1
是否缺页	√	√	√	√			√	√	√	√	√	√

分配四个内存块时，缺页次数：10次
分配三个内存块时，缺页次数：9次

Belady 异常——当为进程分配的物理块数增大时，缺页次数不减反增的异常现象。
贝拉迪异常
只有 FIFO 算法会产生 Belady 异常。另外，FIFO算法虽然实现简单，但是该算法与进程实际运行时的规律不适应，因为先进入的页面也有可能最经常被访问。因此，算法性能差

典型的页面置换算法

4 最近最少用页面置换算法

- 淘汰的页面是在最近较久未被访问或访问次数最少的页面
- 基于程序局部性原理，刚被访问过的页面，马上再被访问的概率将会较高

典型的页面置换算法

4 最近最少用页面置换算法

- 设置一个队列，存放当前在主存中的页号
- 页面访问后，需要从队列中把该页调整到队列尾
- 队列尾总指向最近访问的页，队列头就是最近最少用的页面
- 缺页中断时，总淘汰队列头所指示的页面

最近最久未使用置换算法（LRU）

最近最久未使用置换算法（LRU，least recently used）：每次淘汰的页面是最近最久未使用的页面
实现方法：赋予每个页面对应的页表项中，用访问字段记录该页面自上次被访问以来所经历的时间t。
当需要淘汰一个页面时，选择现有页面中 t 值最大的，即最近最久未使用的页面。

页号	内存块号	状态位	访问字段	修改位	外存地址

例：假设系统为某进程分配了四个内存块，并考虑到有以下页面号引用串
1, 8, 1, 7, 8, 2, 7, 2, 1, 8, 3, 8, 2, 1, 3, 1, 7, 1, 3, 7

最近最久未使用置换算法（LRU）

最近最久未使用置换算法（LRU，least recently used）：每次淘汰的页面是最近最久未使用的页面
实现方法：赋予每个页面对应的页表项中，用访问字段记录该页面自上次被访问以来所经历的时间t。
当需要淘汰一个页面时，选择现有页面中 t 值最大的，即最近最久未使用的页面。

页号	内存块号	状态位	访问字段	修改位	外存地址

例：假设系统为某进程分配了四个内存块，并考虑到有以下页面号引用串
1, 8, 1, 7, 8, 2, 7, 2, 1, 8, 3, 8, 2, 1, 3, 1, 7, 1, 3, 7

访问页面	1	8	1	7	8	2	7	2	1	8	3	8	2	1	3	1	7	1	3	7
内存块1	1	1		1		1					1						1			
内存块2		8		8		8					8						7			
内存块3				7		7					3						3			
内存块4						2					2						2			
缺页否	√	√		√		√					√						√			

最近最久未使用置换算法（LRU）

最近最久未使用置换算法（LRU，least recently used）：每次淘汰的页面是最近最久未使用的页面
实现方法：赋予每个页面对应的页表项中，用访问字段记录该页面自上次被访问以来所经历的时间t。
当需要淘汰一个页面时，选择现有页面中 t 值最大的，即最近最久未使用的页面。

页号	内存块号	状态位	访问字段	修改位	外存地址

例：假设系统为某进程分配了四个内存块，并考虑到有以下页面号引用串
1, 8, 1, 7, 8, 2, 7, 2, 1, 8, 3, 8, 2, 1, 3, 1, 7, 1, 3, 7

访问页面	1	8	1	7	8	2	7	2	1	8	3	8	2	1	3	1	7	1	3	7
内存块1	1	1		1		1					1						1			
内存块2		8		8		8					8						7			
内存块3				7		7					3						3			
内存块4						2					2						2			
缺页否	√	√		√		√					√						√			

最近最久未使用置换算法（LRU）

最近最久未使用置换算法（LRU，least recently used）：每次淘汰的页面是最近最久未使用的页面
实现方法：赋予每个页面对应的页表项中，用访问字段记录该页面自上次被访问以来所经历的时间t。
当需要淘汰一个页面时，选择现有页面中 t 值最大的，即最近最久未使用的页面。

页号	内存块号	状态位	访问字段	修改位	外存地址

例：假设系统为某进程分配了四个内存块，并考虑到有以下页面号引用串
1, 8, 1, 7, 8, 2, 7, 2, 1, 8, 3, 8, 2, 1, 3, 1, 7, 1, 3, 7

访问页面	1	8	1	7	8	2	7	2	1	8	3	8	2	1	3	1	7	1	3	7
内存块1	1	1		1		1					1						1			
内存块2		8		8		8					8						7			
内存块3				7		7					3						3			
内存块4						2					2						2			
缺页否	√	√		√		√					√						√			

最近最久未使用置换算法（LRU）

最近最久未使用置换算法（LRU，least recently used）：每次淘汰的页面是最近最久未使用的页面
实现方法：赋予每个页面对应的页表项中，用访问字段记录该页面自上次被访问以来所经历的时间t。
当需要淘汰一个页面时，选择现有页面中 t 值最大的，即最近最久未使用的页面。

页号	内存块号	状态位	访问字段	修改位	外存地址

例：假设系统为某进程分配了四个内存块，并考虑到有以下页面号引用串
1, 8, 1, 7, 8, 2, 7, 2, 1, 8, 3, 8, 2, 1, 3, 1, 7, 1, 3, 7

OPT向前看算法，LRU向后看算法

访问页面	1	8	1	7	8	2	7	2	1	8	3	8	2	1	3	1	7	1	3	7
内存块1	1	1		1		1					1						1			
内存块2		8		8		8					8						7			
内存块3				7		7					3						3			
内存块4						2					2						2			
缺页否	√	√		√		√					√						√			

在手动做题时，若需要淘汰页面，可以逆向检查此时在内存中的几个页面号。在逆向扫描过程中最后一个出现的页号就是要淘汰的页面。

最近最久未使用置换算法（LRU）

最近最久未使用置换算法（LRU，least recently used）：每次淘汰的页面是最近最久未使用的页面
实现方法：赋予每个页面对应的页表项中，用访问字段记录该页面自上次被访问以来所经历的时间t。
当需要淘汰一个页面时，选择现有页面中 t 值最大的，即最近最久未使用的页面。

页号	内存块号	状态位	访问字段	修改位	外存地址

例：假设系统为某进程分配了四个内存块，并考虑到有以下页面号引用串
1, 8, 1, 7, 8, 2, 7, 2, 1, 8, 3, 8, 2, 1, 3, 1, 7, 1, 3, 7

访问页面	1	8	1	7	8	2	7	2	1	8	3	8	2	1	3	1	7	1	3	7
内存块1	1	1		1		1					1						1			
内存块2		8		8		8					8						7			
内存块3				7		7					3						3			
内存块4						2					2						2			
缺页否	√	√		√		√					√						√			

在手动做题时，若需要淘汰页面，可以逆向检查此时在内存中的几个页面号。在逆向扫描过程中最后一个出现的页号就是要淘汰的页面。

最近最久未使用置换算法（LRU）

最近最久未使用置换算法（LRU，least recently used）：每次淘汰的页面是最近最久未使用的页面
实现方法：赋予每个页面对应的页表项中，用访问字段记录该页面自上次被访问以来所经历的时间t。
当需要淘汰一个页面时，选择现有页面中 t 值最大的，即最近最久未使用的页面。

页号	内存块号	状态位	访问字段	修改位	外存地址

该算法的实现需要专门的硬件支持，虽然算法性能好，但是实现困难，开销大

例：假设系统为某进程分配了四个内存块，并考虑到有以下页面号引用串
1, 8, 1, 7, 8, 2, 7, 2, 1, 8, 3, 8, 2, 1, 3, 1, 7, 1, 3, 7

访问页面	1	8	1	7	8	2	7	2	1	8	3	8	2	1	3	1	7	1	3	7
内存块1	1	1		1		1					1						1			
内存块2		8		8		8					8						7			
内存块3				7		7					3						3			
内存块4						2					2						2			
缺页否	√	√		√		√					√						√			

在手动做题时，若需要淘汰页面，可以逆向检查此时在内存中的几个页面号。在逆向扫描过程中最后一个出现的页号就是要淘汰的页面。

时钟置换算法（CLOCK）

最近最少用置换算法的一种实现方式

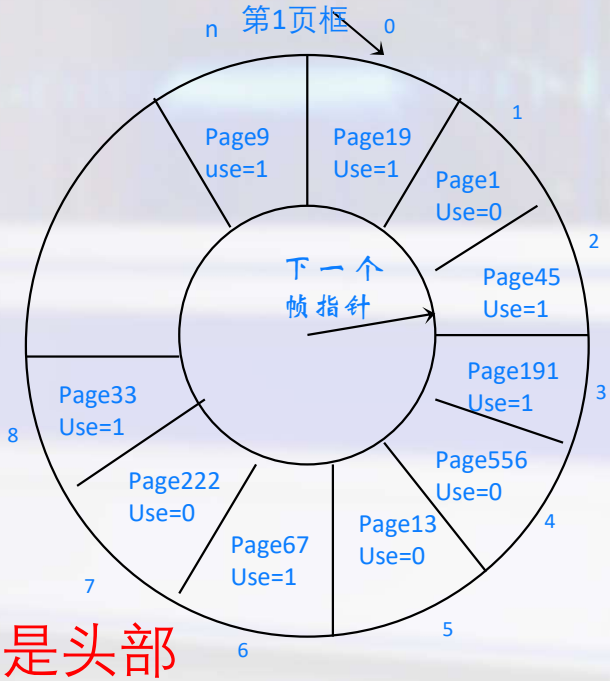
最佳置换算法性能最好，但无法实现；先进先出置换算法实现简单，但算法性能差；最近最久未使用置换算法性能好，是最接近OPT算法性能的，但是实现起来需要专门的硬件支持，算法开销大。

时钟置换算法是一种性能和开销较均衡的算法，又称CLOCK算法，或最近未用算法（NRU，Not Recently Used）

典型的页面置换算法

5 时钟页面置换算法

- ✓ 一个页面首次装入主存，“访问位”置1
- ✓ 主存中的任何页面被访问时，“访问位”置1

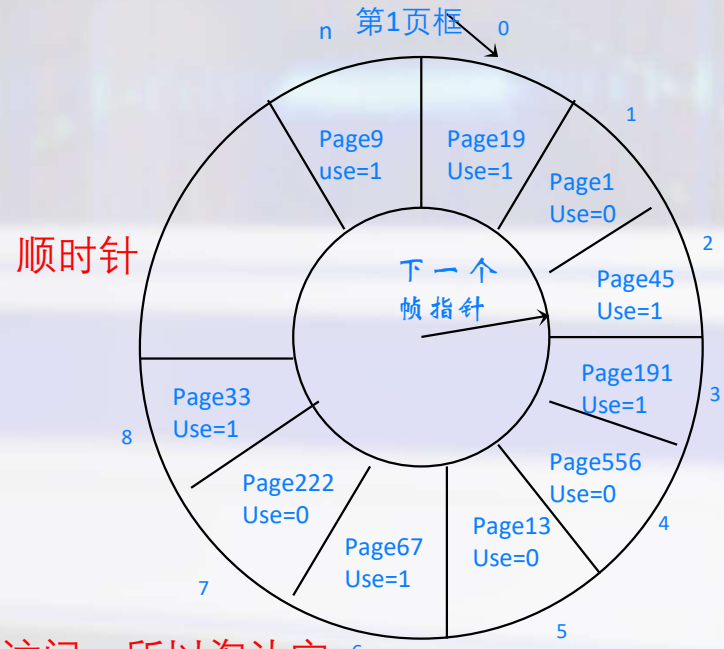


所有逻辑页面放到，循环队列，指针指向谁谁就是头部

典型的页面置换算法

5 时钟页面置换算法

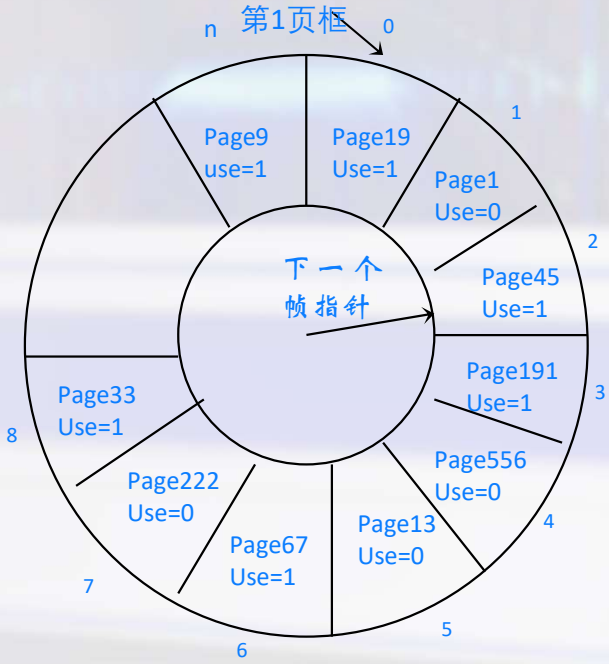
- ✓ 淘汰页面时，从指针当前指向的页面开始扫描循环队列
- ✓ 遇到的“访问位”是1的页面的“访问位”清0，跳过这个页面
- ✓ 把所遇到的“访问位”是0的页面置换掉，指针推进一步



典型的页面置换算法

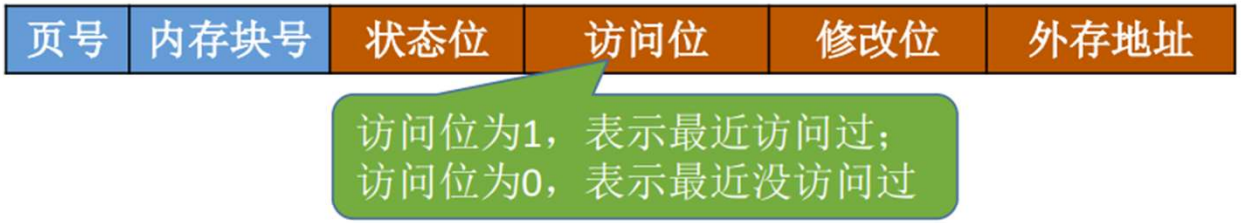
5 时钟页面置换算法

- ✓ 如果遇到的所有页面的“访问位”为1，指针就会绕整个循环队列一圈
- ✓ 把碰到的所有页面的“访问位”清0
- ✓ 指针回到起始位置，并淘汰掉这一页，指针推进一步



时钟置换算法（CLOCK）

简单的CLOCK 算法实现方法：为每个页面设置一个访问位，再将内存中的页面都通过链接指针链接成一个循环队列。



时钟置换算法（CLOCK）

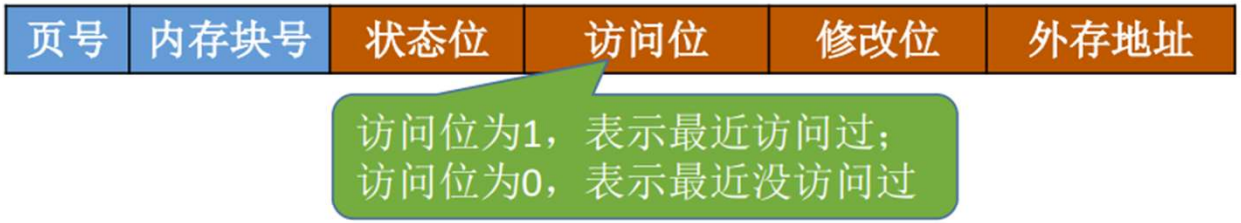
当需要淘汰一个页面时，只需检查页的访问位。

页号	内存块号	状态位	访问位	修改位	外存地址

访问位为1，表示最近访问过；
访问位为0，表示最近没访问过

时钟置换算法（CLOCK）

如果是0，就选择该页换出；如果是1，则将它置为0，暂不换出，继续检查下一个页面，



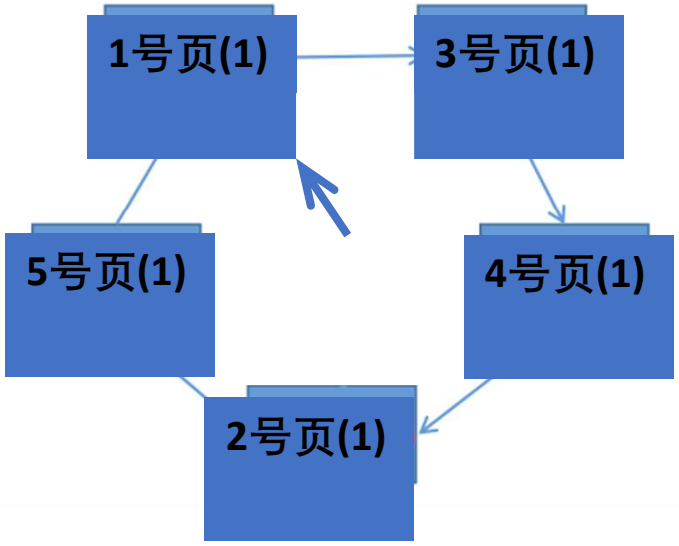
时钟置换算法（CLOCK）

如果是0，就选择该页换出；如果是1，则将它置为0，暂不换出，继续检查下一个页面，

页号	内存块号	状态位	访问位	修改位	外存地址

访问位为1，表示最近访问过；
访问位为0，表示最近没访问过

例：假设系统为某进程分配了五个内存块，并考虑到有以下页面号引用串：
1, 3, 4, 2, 5, 6, 3, 4, 7



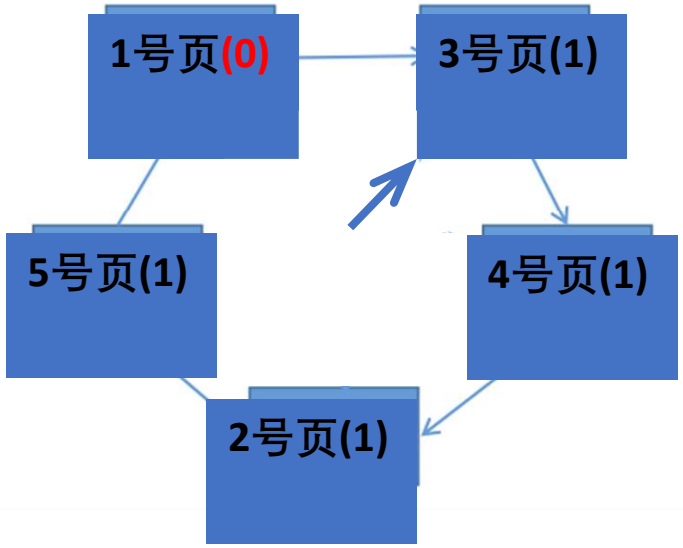
时钟置换算法（CLOCK）

如果是0，就选择该页换出；如果是1，则将它置为0，暂不换出，继续检查下一个页面，

页号	内存块号	状态位	访问位	修改位	外存地址

访问位为1，表示最近访问过；
访问位为0，表示最近没访问过

例：假设系统为某进程分配了五个内存块，并考虑到有以下页面号引用串：
1, 3, 4, 2, 5, 6, 3, 4, 7



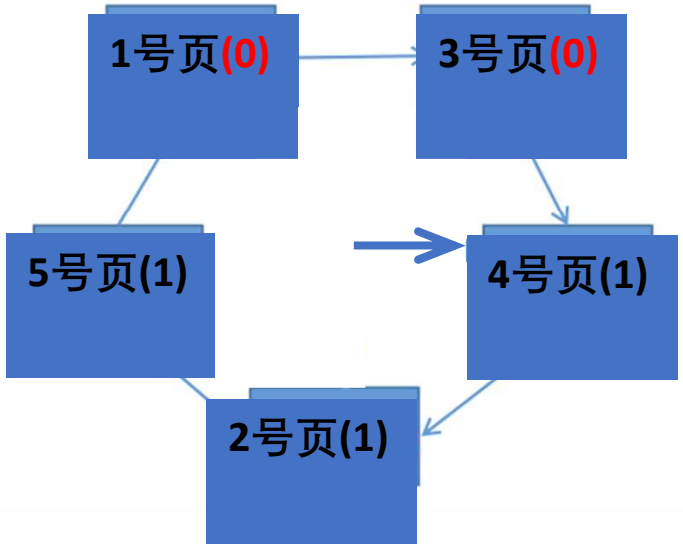
时钟置换算法（CLOCK）

如果是0，就选择该页换出；如果是1，则将它置为0，暂不换出，继续检查下一个页面，

页号	内存块号	状态位	访问位	修改位	外存地址

访问位为1，表示最近访问过；
访问位为0，表示最近没访问过

例：假设系统为某进程分配了五个内存块，并考虑到有以下页面号引用串：
1, 3, 4, 2, 5, 6, 3, 4, 7



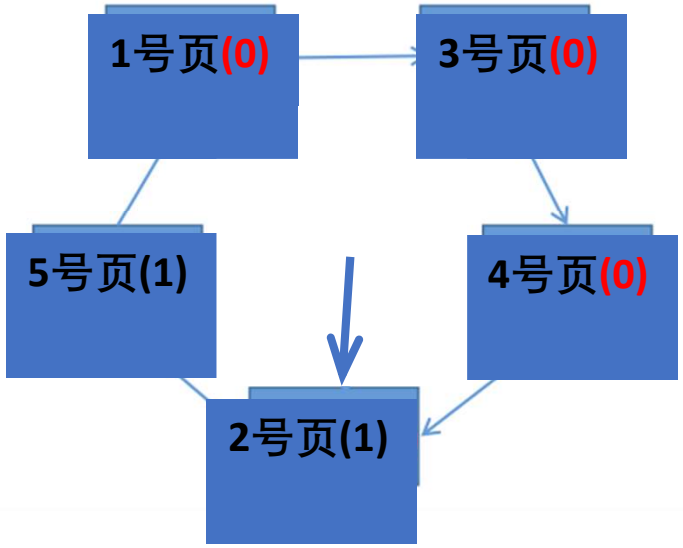
时钟置换算法（CLOCK）

如果是0，就选择该页换出；如果是1，则将它置为0，暂不换出，继续检查下一个页面，

页号	内存块号	状态位	访问位	修改位	外存地址

访问位为1，表示最近访问过；
访问位为0，表示最近没访问过

例：假设系统为某进程分配了五个内存块，并考虑到有以下页面号引用串：
1, 3, 4, 2, 5, 6, 3, 4, 7



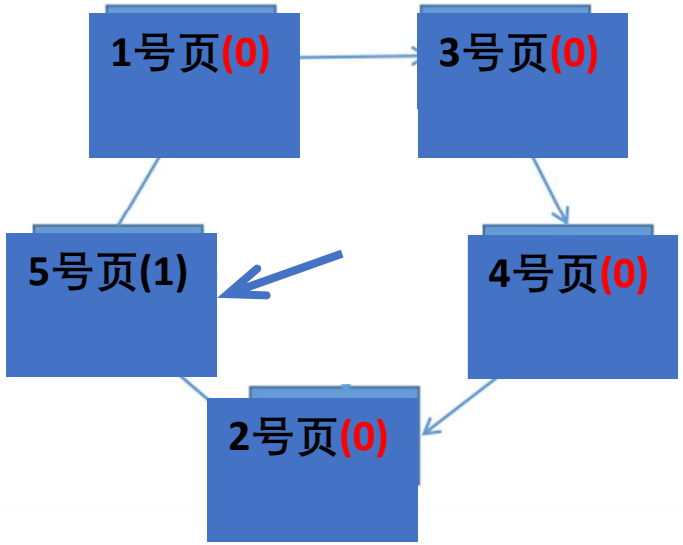
时钟置换算法（CLOCK）

如果是0，就选择该页换出；如果是1，则将它置为0，暂不换出，继续检查下一个页面，

页号	内存块号	状态位	访问位	修改位	外存地址

访问位为1，表示最近访问过；
访问位为0，表示最近没访问过

例：假设系统为某进程分配了五个内存块，并考虑到有以下页面号引用串：
1, 3, 4, 2, 5, 6, 3, 4, 7



时钟置换算法（CLOCK）

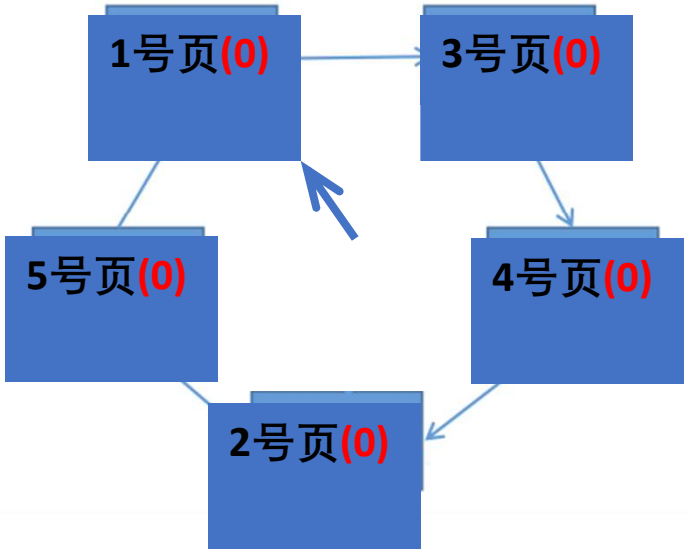
如果是0，就选择该页换出；如果是1，则将它置为0，暂不换出，继续检查下一个页面，若第一轮扫描中所有页面都是1，则将这些页面的访问位依次置为0后，再进行第二轮扫描（第二轮扫描中一定会有访问位为0的页面，因此简单的CLOCK 算法选择一个淘汰页面最多会经过两轮扫描）

页号	内存块号	状态位	访问位	修改位	外存地址

访问位为1，表示最近访问过；
访问位为0，表示最近没访问过

例：假设系统为某进程分配了五个内存块，并考虑到有以下页面号引用串：

1, 3, 4, 2, 5, 6, 3, 4, 7



时钟置换算法（CLOCK）

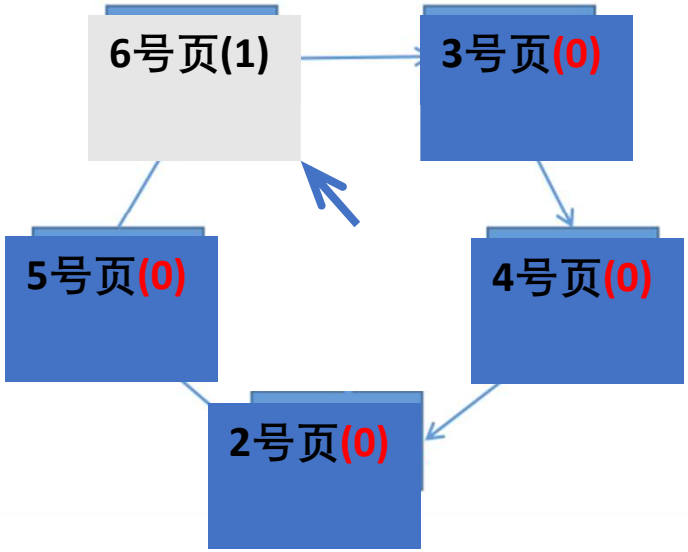
如果是0，就选择该页换出；如果是1，则将它置为0，暂不换出，继续检查下一个页面，若第一轮扫描中所有页面都是1，则将这些页面的访问位依次置为0后，再进行第二轮扫描（第二轮扫描中一定会有访问位为0的页面，因此简单的CLOCK 算法选择一个淘汰页面最多会经过两轮扫描）

页号	内存块号	状态位	访问位	修改位	外存地址

访问位为1，表示最近访问过；
访问位为0，表示最近没访问过

例：假设系统为某进程分配了五个内存块，并考虑到有以下页面号引用串：

1, 3, 4, 2, 5, 6, 3, 4, 7



时钟置换算法（CLOCK）

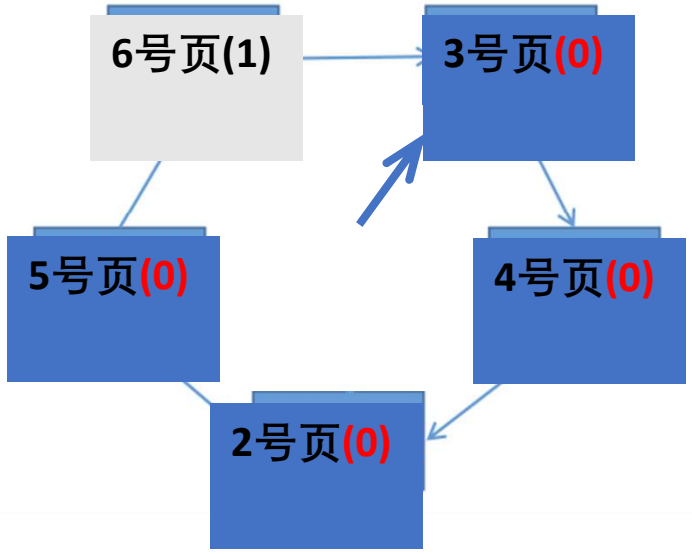
如果是0，就选择该页换出；如果是1，则将它置为0，暂不换出，继续检查下一个页面，若第一轮扫描中所有页面都是1，则将这些页面的访问位依次置为0后，再进行第二轮扫描（第二轮扫描中一定会有访问位为0的页面，因此简单的CLOCK 算法选择一个淘汰页面最多会经过两轮扫描）

页号	内存块号	状态位	访问位	修改位	外存地址

访问位为1，表示最近访问过；
访问位为0，表示最近没访问过

例：假设系统为某进程分配了五个内存块，并考虑到有以下页面号引用串：

1, 3, 4, 2, 5, 6, 3, 4, 7



时钟置换算法（CLOCK）

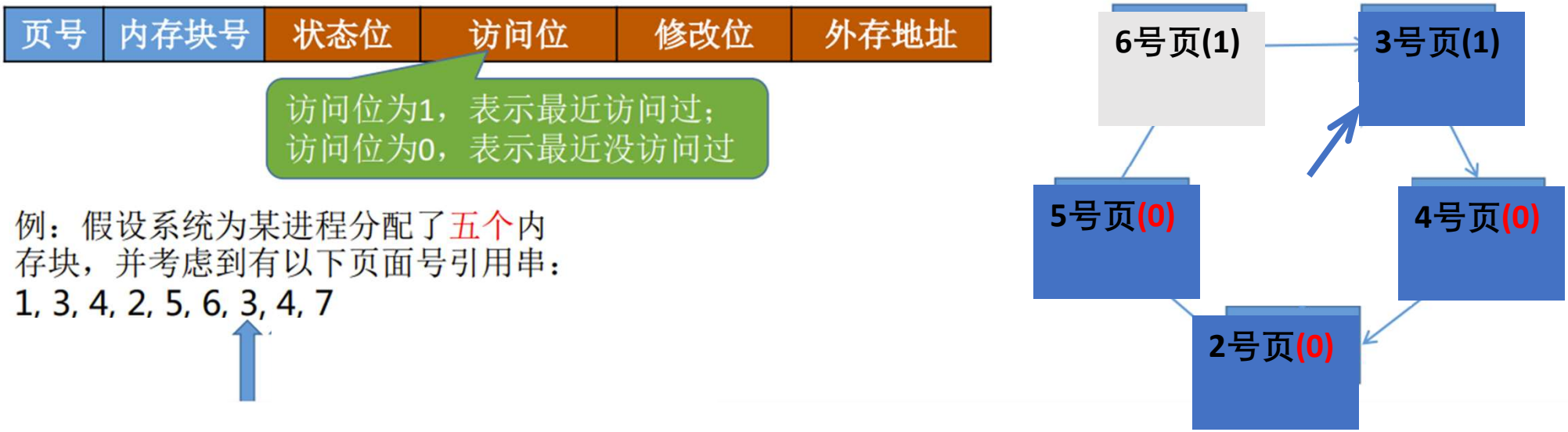
如果是0，就选择该页换出；如果是1，则将它置为0，暂不换出，继续检查下一个页面，若第一轮扫描中所有页面都是1，则将这些页面的访问位依次置为0后，再进行第二轮扫描（第二轮扫描中一定会有访问位为0的页面，因此简单的CLOCK 算法选择一个淘汰页面最多会经过两轮扫描）

页号	内存块号	状态位	访问位	修改位	外存地址

访问位为1，表示最近访问过；
访问位为0，表示最近没访问过

例：假设系统为某进程分配了五个内存块，并考虑到有以下页面号引用串：

1, 3, 4, 2, 5, 6, 3, 4, 7



时钟置换算法（CLOCK）

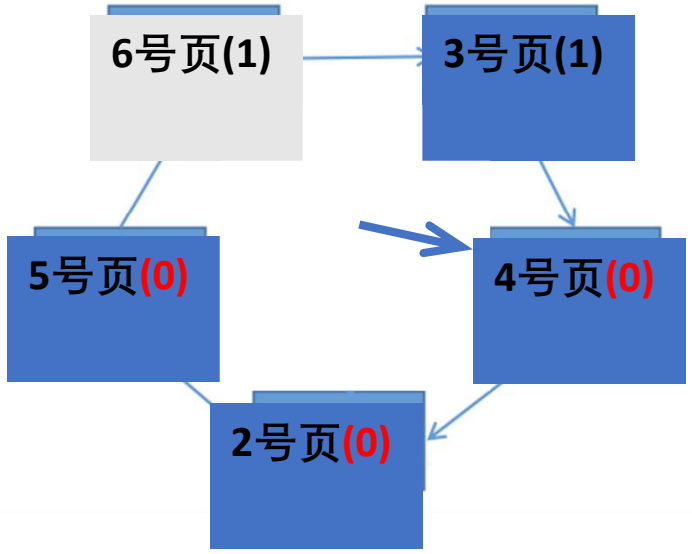
如果是0，就选择该页换出；如果是1，则将它置为0，暂不换出，继续检查下一个页面，若第一轮扫描中所有页面都是1，则将这些页面的访问位依次置为0后，再进行第二轮扫描（第二轮扫描中一定会有访问位为0的页面，因此简单的CLOCK 算法选择一个淘汰页面最多会经过两轮扫描）

页号	内存块号	状态位	访问位	修改位	外存地址

访问位为1，表示最近访问过；
访问位为0，表示最近没访问过

例：假设系统为某进程分配了五个内存块，并考虑到有以下页面号引用串：

1, 3, 4, 2, 5, 6, 3, 4, 7



时钟置换算法（CLOCK）

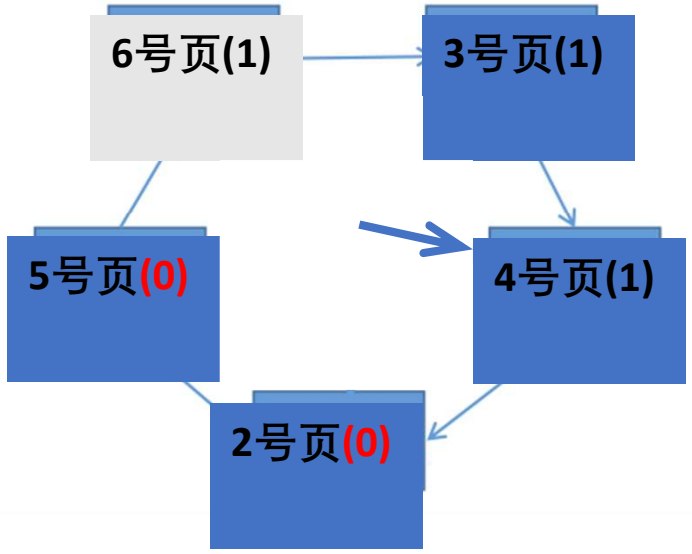
如果是0，就选择该页换出；如果是1，则将它置为0，暂不换出，继续检查下一个页面，若第一轮扫描中所有页面都是1，则将这些页面的访问位依次置为0后，再进行第二轮扫描（第二轮扫描中一定会有访问位为0的页面，因此简单的CLOCK 算法选择一个淘汰页面最多会经过两轮扫描）

页号	内存块号	状态位	访问位	修改位	外存地址

访问位为1，表示最近访问过；
访问位为0，表示最近没访问过

例：假设系统为某进程分配了五个内存块，并考虑到有以下页面号引用串：

1, 3, 4, 2, 5, 6, 3, 4, 7



时钟置换算法（CLOCK）

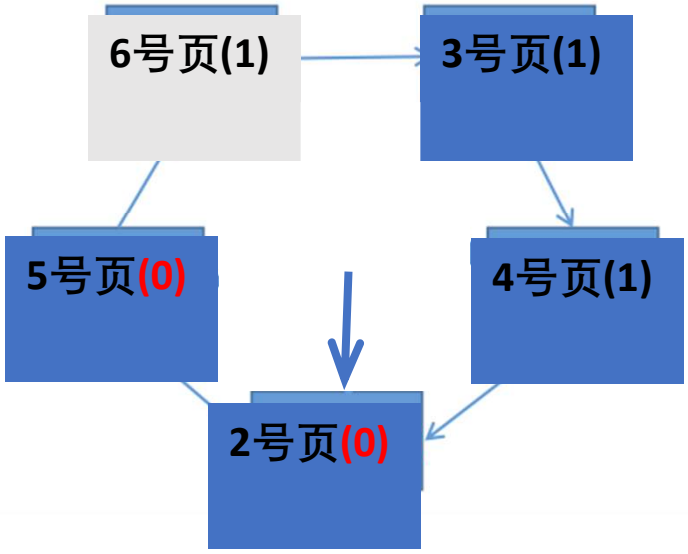
如果是0，就选择该页换出；如果是1，则将它置为0，暂不换出，继续检查下一个页面，若第一轮扫描中所有页面都是1，则将这些页面的访问位依次置为0后，再进行第二轮扫描（第二轮扫描中一定会有访问位为0的页面，因此简单的CLOCK 算法选择一个淘汰页面最多会经过两轮扫描）

页号	内存块号	状态位	访问位	修改位	外存地址

访问位为1，表示最近访问过；
访问位为0，表示最近没访问过

例：假设系统为某进程分配了五个内存块，并考虑到有以下页面号引用串：

1, 3, 4, 2, 5, 6, 3, 4, 7



时钟置换算法（CLOCK）

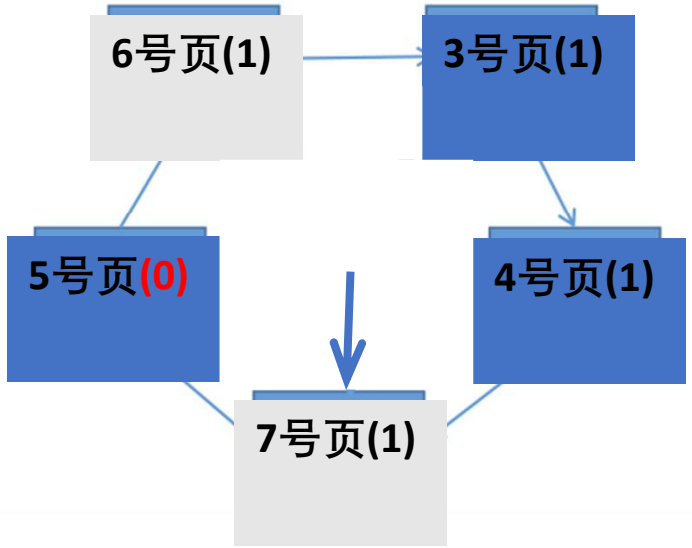
如果是0，就选择该页换出；如果是1，则将它置为0，暂不换出，继续检查下一个页面，若第一轮扫描中所有页面都是1，则将这些页面的访问位依次置为0后，再进行第二轮扫描（第二轮扫描中一定会有访问位为0的页面，因此简单的CLOCK 算法选择一个淘汰页面最多会经过两轮扫描）

页号	内存块号	状态位	访问位	修改位	外存地址

访问位为1，表示最近访问过；
访问位为0，表示最近没访问过

例：假设系统为某进程分配了五个内存块，并考虑到有以下页面号引用串：

1, 3, 4, 2, 5, 6, 3, 4, 7



知识回顾与重要考点

	算法规则	优缺点
OPT	优先淘汰最长时间内不会被访问的页面	缺页率最小，性能最好；但无法实现
FIFO	优先淘汰最先进入内存的页面	实现简单；但性能很差，可能出现Belady异常
LRU	优先淘汰最近最久没访问的页面	性能很好；但需要硬件支持，算法开销大
CLOCK（NRU）	循环扫描各页面 第一轮淘汰访问位=0的，并将扫描过的页面访问位改为1。若第一轮没选中，则进行第二轮扫描。	实现简单，算法开销小；但未考虑页面是否被修改过。

虚拟存储管理机制

本讲内容

1. 程序访问局部性原理
2. 虚拟存储器基本原理
3. 分页式虚拟存储管理
4. 典型的页面置换算法
5. 分段式虚拟存储管理

分段式虚拟存储管理

- 分段式虚拟存储系统把所有分段的副本都存放在辅助存储器中
- 进程被调度投入运行时，把当前需要的一段或几段装入主存
- 在执行过程中访问到不在主存的段时再动态装入

课堂讨论：

描述执行 ./a.out 之后程序的加载和运行过程。

越详细越好。至少要包含以下信息：

1）进程创建过程中，内存和辅存是如何参与其中的？

2）进程运行过程中，内存和辅存是如何参与其中的？

3）TLB未命中之后，系统中会发生什么事情？

4）操作系统如何处理 page fault ？

执行 ./a.out 后程序的加载和运行过程

1) 进程创建过程中，内存和辅存是如何参与其中的?

- **内存参与:**
  - 当执行 ./a.out 启动程序时，操作系统会为新进程分配内存空间来存储程序的代码段、数据段和堆栈段。
  - 初始的程序代码和数据被加载到进程的内存空间中，以便 CPU 能够执行该程序。
- **辅存参与:**
  - 如果内存不足以容纳整个程序，操作系统会将部分程序代码和数据从辅存（如硬盘）加载到内存，这包括使用页面文件或交换空间技术。加载过程中，操作系统还会进行地址重定位等操作，确保程序在虚拟内存中的地址是正确的。

2) 进程运行过程中，内存和辅存是如何参与其中的?

- **内存参与:**
  - 进程在运行过程中，会使用内存来存储当前正在执行的指令、数据和堆栈信息。
  - 内存管理单元（MMU）负责将虚拟地址翻译成物理地址，CPU 通过内存中的页表来访问和管理进程的内存空间。
- **辅存参与:**
  - 如果需要的数据或指令不在内存中，会发生缺页中断（page fault），此时操作系统会介入，将相应的页面从辅存中调入内存，然后重新执行引发缺页中断的指令。

3) TLB未命中之后，系统中会发生什么事情?

- 当 TLB（Translation Lookaside Buffer）未命中时，意味着 CPU 需要查询页表来获取虚拟地址到物理地址的映射关系。
- 未命中时，CPU 会在页表中进行查找，找到对应的物理页框，并将结果存储在 TLB 中，以便后续访问可以快速完成。

4) 操作系统如何处理 page fault?

- 当程序访问的页面不在内存中时，会发生 page fault。
- 操作系统会进行以下处理:
  - 中断当前程序的执行，转到内核态执行 page fault 的处理程序。
  - 操作系统会检查访问的页面是否在辅存中，如果是，会将相应的页面调入内存并更新页表。
  - 如果访问的页面不在辅存中，会发生访问错误，操作系统会向程序发送一个信号，通知程序访问错误。

综上所述，执行 ./a.out 的过程涉及到内存和辅存的协同工作，以及操作系统对于进程创建、加载、运行和处理异常情况的处理。