
内存管理

宿船长 B站专用



虚拟存储器

宿船长 B站专用

传统存储管理方式的特征、缺点

很多暂时用不到的数据也会长期占用内存，导致内存利用率不高

可用虚拟存储技术解决问题

传统存储管理

连续分配管理方式

单一连续分配

固定分区分配

动态分区分配

非连续分配管理方式

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

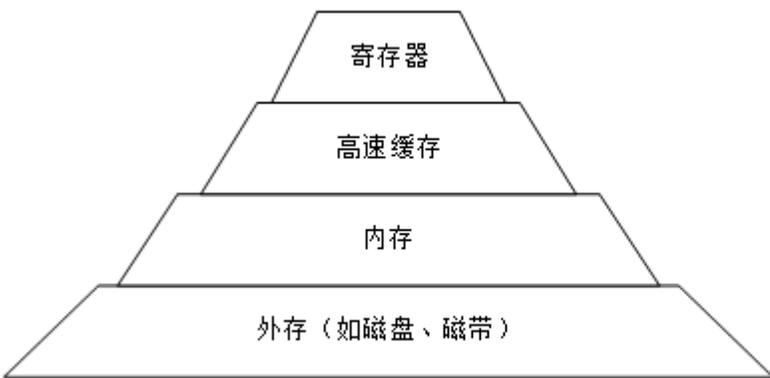
一次性：作业必须一次性全部装入内存后才能开始运行。这会造成两个问题：①作业很大时，不能全部装入内存，导致**大作业无法运行**；②当大量作业要求运行时，由于内存无法容纳所有作业，因此只有少量作业能运行，导致**多道程序并发度下降**。

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。

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）



计算机中存储器的层次结构

↑
容量小
速度快
成本高

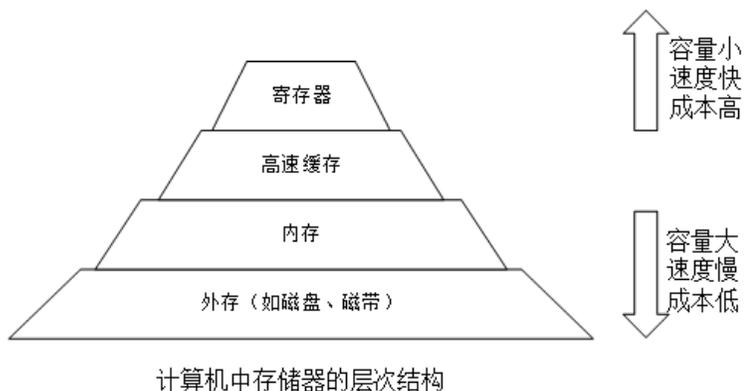
快表机构就是将近期常访问的页表项副本放到更高速的联想寄存器中

↓
容量大
速度慢
成本低

Q：如何应用局部性原理？

A：**高速缓冲技术**的思想：将近期会频繁访问到的数据放到更高速的存储器中，暂时用不到的数据放在更低速存储器中。

虚拟内存的定义和特征



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**

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**= $\min(\text{内存和外存容量之和}, \text{CPU寻址范围})$

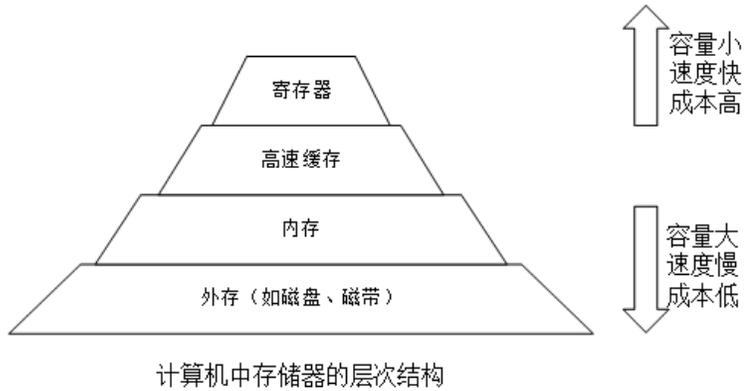
如：某计算机地址结构为32位，按字节编址，内存大小为512MB，外存大小为2GB。

则虚拟内存的**最大容量**为 $2^{32} \text{ B} = 4\text{GB}$

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

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



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**。

虚拟内存有以下三个主要特征：

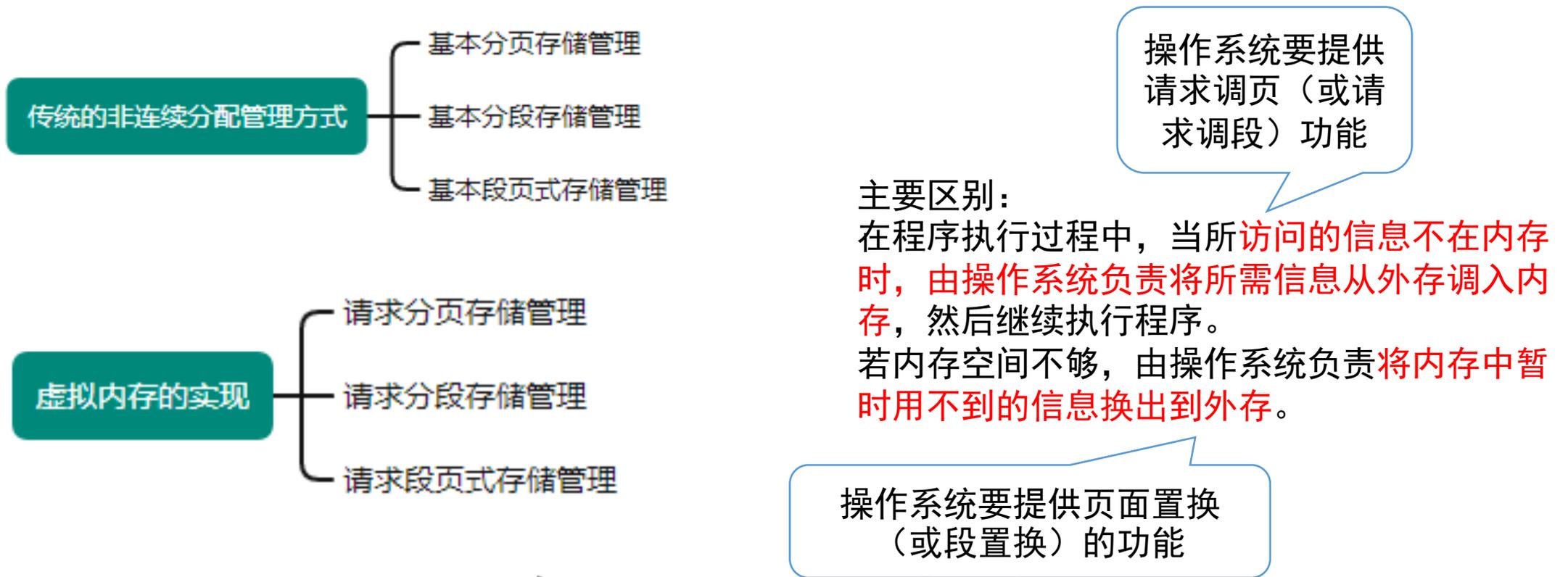
多次性：无需在作业运行时一次性全部装入内存，而是允许被分成多次调入内存。

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。

如何实现虚拟内存

虚拟内存技术，允许一个作业分多次调入内存。如果采用连续分配方式，会不方便实现。因此，虚拟内存的实现需要建立在**离散分配**的内存管理方式基础上。





请求分页管理

宿船长 B站专用

请求分页管理方式

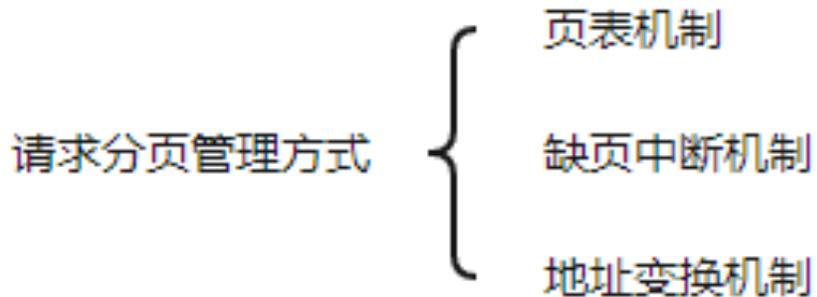
请求分页存储管理与基本分页存储管理的主要区别：

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

操作系统要提供页面置换的功能，
将暂时用不到的页面换出外存

操作系统要提供请求
调页功能，将缺失页
面从外存调入内存



宿船长 B站专用

请求分页管理方式（页表机制）

与基本分页管理相比，请求分页管理中，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。

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到底换出哪个页面；有的页面没有被修改过，就不用再浪费时间写回外存。有的页面修改过，就需要将外存中的旧数据覆盖，因此，操作系统也需要记录各个页面是否被修改的信息。

页号	内存块号
0	a
1	b
2	c

基本分页存储管理的页表

请求页表项增加了四个字段

是否已调入内存

可记录最近被访问过几次，或记录上次访问的时间，供置换算法选择换出页面时参考

页面调入内存后是否被修改过

页面在外存中的存放位置

页号	内存块号	状态位	访问字段	修改位	外存地址
0	无	0	0	0	x
1	b	1	10	0	y
2	c	1	6	1	z

请求分页存储管理的页表

请求分页管理方式（缺页中断机构）

页号	内存块号	状态位	访问字段	修改位	外存地址
0	a	1	0	0	x
1	b	1	10	0	y
2	c	1	6	1	z



假设此时要访问逻辑地址=（页号，页内偏移量）=（0，1024）

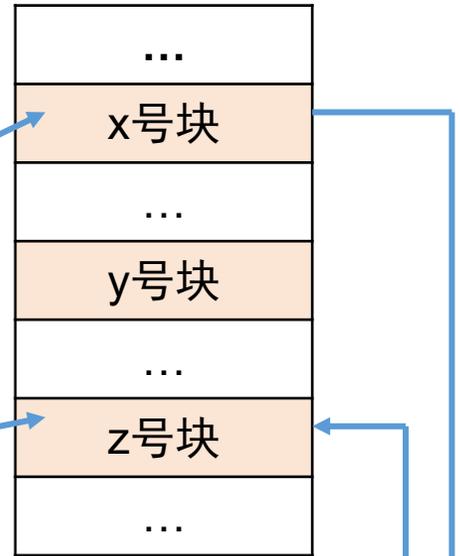
在请求分页系统中，每当要访问的页面不在内存时，便产生一个缺页中断，然后由操作系统的缺页中断处理程序处理中断。

此时缺页的进程阻塞，放入阻塞队列，调页完成后将其唤醒，放回就绪队列。

如果内存中有空闲块，则为进程分配一个空闲块，将所缺页面装入该块，并修改页表中相应的页表项。

请求分页管理方式（缺页中断机构）

页号	内存块号	状态位	访问字段	修改位	外存地址
0	c	1	0	0	x
1	b	1	10	0	y
2	无	0	0	0	z



外存



内存

假设此时要访问逻辑地址=（页号，页内偏移量）=（0，1024）

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。

此时缺页的进程阻塞，放入阻塞队列，调页完成后将其唤醒，放回就绪队列。

如果内存中有空闲块，则为进程分配一个空闲块，将所缺页面装入该块，并修改页表中相应的页表项。

如果内存中没有空闲块，则由页面置换算法选择一个页面淘汰，若该页面在内存期间被修改过，则要将其写回外存。未修改过的页面不用写回外存。

宿船长 B站专用

请求分页管理方式（缺页中断机构）

缺页中断是因为当前执行的指令想要访问的目标页面未调入内存而产生的，因此**属于内中断一条指令**在执行期间，**可能产生多次缺页中断**。（如：copy A to B，即将逻辑地址A中的数据复制到逻辑地址B，而A、B属于不同的页面，则有可能产生两次中断）

请求分页管理方式（地址变换机构）

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

操作系统要提供页面置换的功能，将暂时用不到的页面换出外存

操作系统要提供请求调页功能，将缺失页面从外存调入内存

页号	内存块号	状态位	访问字段	修改位	外存地址
0	无	0	0	0	x
1	b	1	10	0	y
2	c	1	6	1	z

新增步骤1：请求调页（查到页表项时进行判断）

新增步骤2：页面置换（需要调入页面，但没有空闲内存块时进行）

新增步骤3：需要修改请求页表中新增的表项

宿船长B站专用

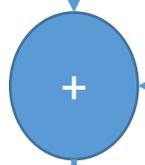
越界异常

快表中有的页面一定是在内存中的。若某个页面被换出外存，则快表中的相应表项也要删除，否则可能访问错误的页面

逻辑地址
页号 页内偏移量

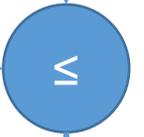


页表
页表始址F



是否命中

b	1	9	0	y
1	1	6	1	z



物理地址

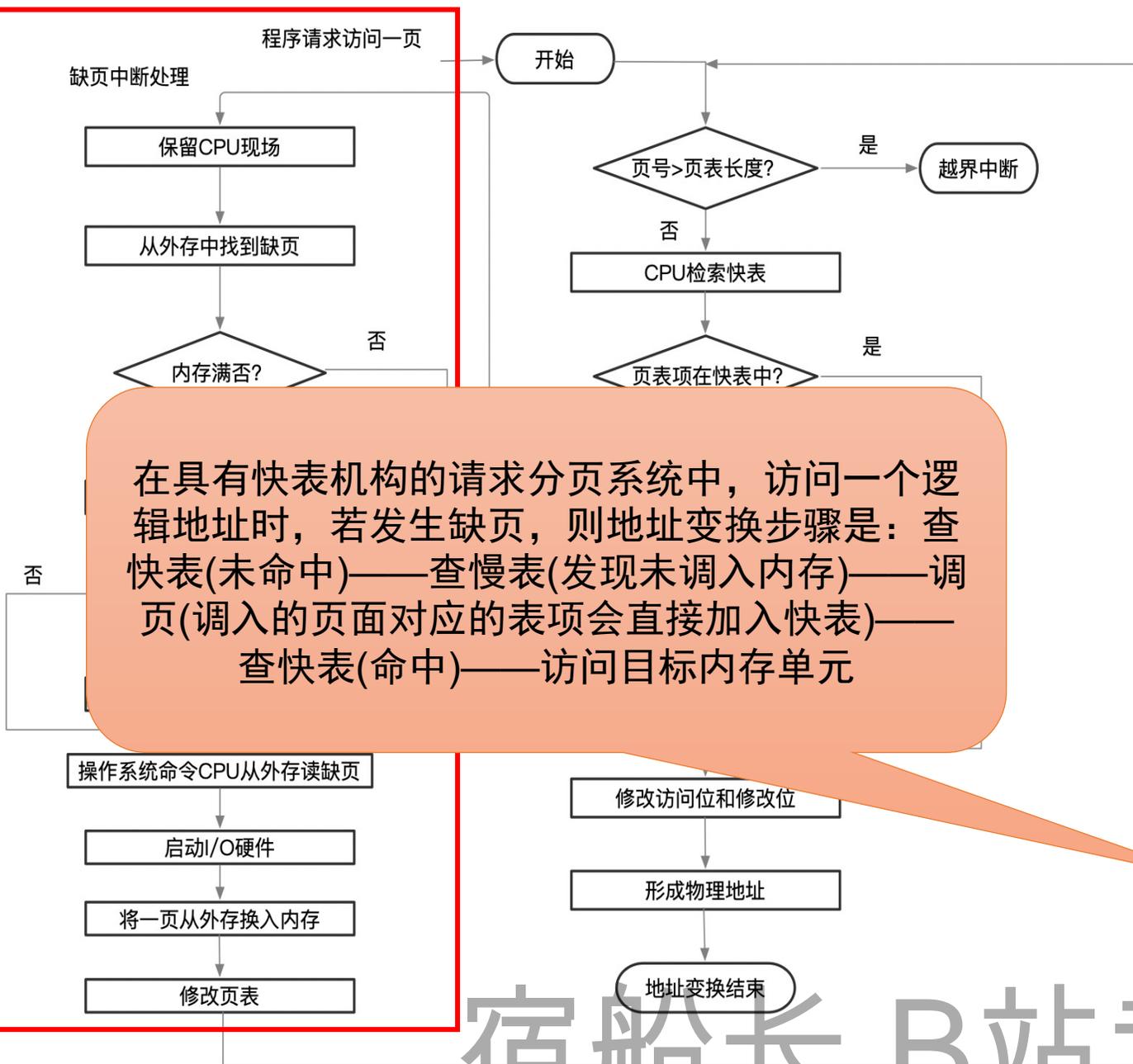
页号	内存块号	状态位	访问字段	修改位	外存地址
	无	0	0	0	x
		1	10	0	y
		1	6	1	z

请求页表



内存

找到对应页表项后，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



在具有快表机构的请求分页系统中，访问一个逻辑地址时，若发生缺页，则地址变换步骤是：查快表(未命中)——查慢表(发现未调入内存)——调页(调入的页面对应的表项会直接加入快表)——查快表(命中)——访问目标内存单元

补充细节：

- ①只有“写指令”才需要修改“修改位”。并且，一般来说只需修改快表中的数据，只有要将快表项删除时才需要写回内存中的慢表。这样可以减少访存次数。
- ②和普通的中断处理一样，缺页中断处理依然需要保留CPU现场。
- ③需要用某种“页面置换算法”来决定一个换出页面（下节内容）
- ④换入/换出页面都需要启动慢速的I/O操作，可见，如果换入/换出太频繁，会有很大的开销。
- ⑤页面调入内存后，需要修改慢表，同时也需要将表项复制到快表中。



页面置换算法

宿船长 B站专用

页面置换算法（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

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 = $9/20 = 45\%$

宿船长 B站专用

页面置换算法（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。

页面置换算法（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

页面置换算法 (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。

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**

页面置换算法（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
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。

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			
是否缺页	√	√		√		√					√						√			

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。

页面置换算法（时钟置换算法，CLOCK）

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）

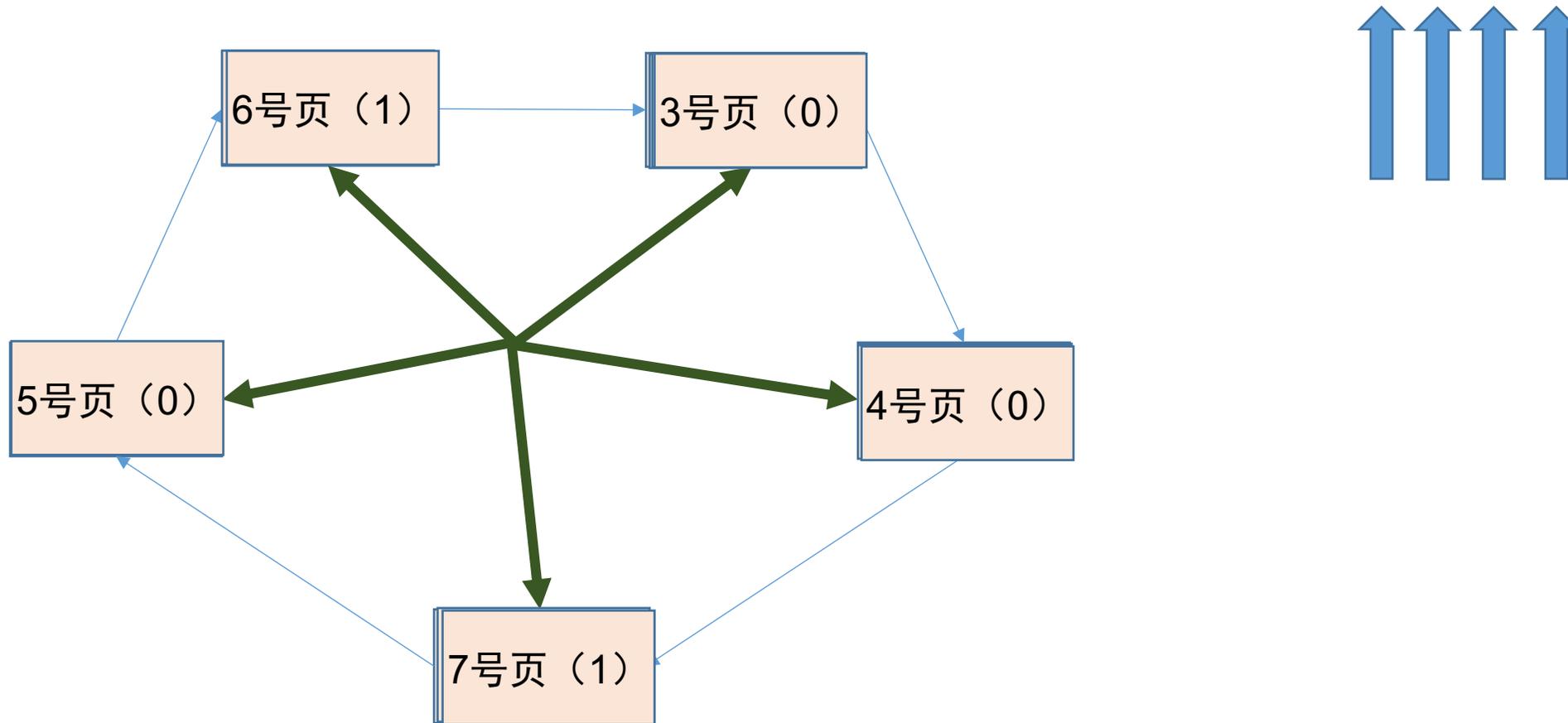
简单的CLOCK算法实现方法：为每个页面设置一个**访问位**，再将内存中的页面都通过链接指针**链接成一个循环队列**。当某页被访问时，其访问位置为1。当需要淘汰一个页面时，只需检查页的访问位。如果是0，就选择该页换出；如果是1，则将它置为0，暂不换出，继续检查下一个页面，若第一轮扫描中所有页面都是1，则将这些页面的访问位依次置为0后，再进行第二轮扫描（第二轮扫描中一定会有访问位为0的页面，因此**简单的CLOCK算法**选择一个淘汰页面**最多会经过两轮扫描**）

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

页面置换算法（时钟置换算法，CLOCK）

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



宿船长 B站专用

页面置换算法（改进型的时钟置换算法）

简单的时钟置换算法仅考虑到一个页面最近是否被访问过。事实上，如果被淘汰的页面没有被修改过，就不需要执行I/O操作写回外存。**只有被淘汰的页面被修改过时，才需要写回外存。**

因此，除了考虑一个页面最近有没有被访问过之外，操作系统还应考虑页面有没有被修改过。**在其他条件都相同时，应优先淘汰没有修改过的页面**，避免I/O操作。这就是改进型的时钟置换算法的思想。

修改位=0，表示页面没有被修改过；**修改位=1**，表示页面被修改过。

为方便讨论，用**（访问位，修改位）**的形式表示各页面状态。如（1，1）表示一个页面近期被访问过，且被修改过。

算法规则：将所有可能被置换的页面排成一个循环队列

第一轮：从当前位置开始扫描到第一个（0，0）的帧用于替换。本轮扫描不修改任何标志位

第二轮：若第一轮扫描失败，则重新扫描，查找第一个（0，1）的帧用于替换。本轮将所有扫描过的帧访问位设为0

第三轮：若第二轮扫描失败，则重新扫描，查找第一个（0，0）的帧用于替换。本轮扫描不修改任何标志位

第四轮：若第三轮扫描失败，则重新扫描，查找第一个（0，1）的帧用于替换。

由于第二轮已将所有帧的访问位设为0，因此经过第三轮、第四轮扫描一定会有一个帧被选中，因此**改进型CLOCK置换算法**选择一个淘汰页面**最多会进行四轮扫描**

页面置换算法（改进型的时钟置换算法）

算法规则：将所有可能被置换的页面排成一个循环队列

第一轮：从当前位置开始扫描到第一个 (0, 0) 的帧用于替换。本轮扫描不修改任何标志位

第二轮：若第一轮扫描失败，则重新扫描，查找第一个 (0, 1) 的帧用于替换。本轮将所有扫描过的帧访问位设为0

第三轮：若第二轮扫描失败，则重新扫描，查找第一个 (0, 0) 的帧用于替换。本轮扫描不修改任何标志位

第四轮：若第三轮扫描失败，则重新扫描，查找第一个 (0, 1) 的帧用于替换。

由于第二轮已将所有帧的访问位设为0，因此经过第三轮、第四轮扫描一定会有一个帧被选中，因此**改进型CLOCK置换算法**选择一个淘汰页面**最多会进行四轮扫描**

第一优先级：最近没访问，且没修改的页面

第二优先级：最近没访问，但修改过的页面

第三优先级：最近访问过，但没修改的页面

第四优先级：最近访问过，且修改过的页面

页面置换算法对比

	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0。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。
改进型 CLOCK (改进型NRU)	若用（访问位,修改位）的形式表述，则 第一轮：淘汰 (0, 0) 第二轮：淘汰 (0, 1)，并将扫描过的页面访问位都置为0 第三轮：淘汰 (0, 0) 第四轮：淘汰 (0, 1)	算法开销较小，性能也不错



页面分配策略

宿船长 B站专用

页面分配策略（页面分配、置换策略）

驻留集：指请求分页存储管理中给进程分配的物理块的集合。

在采用了虚拟存储技术的系统中，驻留集大小一般小于进程的总大小。

若驻留集太小，会导致缺页频繁，系统要花大量的时间来处理缺页，若驻留集太大，又会导致多道程序并发度下降，资源利用率降低。所以应该适当选择驻留集大小。

考虑一个极端情况，若某进程共有100个页面，则该进程的驻留集大小为100时进程可以全部放入内存，运行期间不可能再发生缺页。若驻留集大小为1，则进程运行期间必定会极频繁地缺页

固定分配：操作系统为每个进程分配一组固定数目的物理块，在进程运行期间不再改变。即，**驻留集大小不变**

可变分配：先为每个进程分配一定数目的物理块，在进程运行期间，可根据情况做适当的增加或减少。即，**驻留集大小可变**

局部置换：发生缺页时只能选进程自己的物理块进行置换。

全局置换：可以将操作系统保留的空闲物理块分配给缺页进程，也可以将别的进程持有的物理块置换到外存，再分配给缺页进程。

	局部置换	全局置换
固定分配	√	—
可变分配	√	√

全局置换意味着一个进程拥有的物理块数量必然会改变，因此不可能是固定分配

页面分配策略（页面分配、置换策略）

固定分配局部置换：系统为每个进程分配一定数量的物理块，在整个运行期间保持不变。当进程在运行中发生缺页，则只能从该进程在内存中的页面中选出一页换出，然后再调入所需页面。这种策略的难点在刚开始就确定应为每个进程分配多少个物理块才算合理。（采用这种策略时，可根据进程的优先级、或是根据程序员给出的参数来确定为一个进程分配的内存块数）

系统会锁定一些页面，这些页面中的内容不能置换出外存（如：重要的内核数据可以设为“锁定”）

可变分配全局置换：刚开始会为每个进程分配一定数量的物理块。操作系统会保持一个空闲物理块队列。当某进程发生缺页时，从空闲物理块中取出一块分配给该进程；若已无空闲物理块，则可选择的一个**未锁定的**页面换出外存，再将该物理块分配给缺页的进程。采用这种策略时，**只要某进程发生缺页，都将获得新的物理块**，仅当空闲物理块用完时，系统才选择一个未锁定的页面调出。被选择调出的页可能是系统中任何一个进程中的页，因此这个**被选中的进程拥有的物理块会减少，缺页率会增加**。

可变分配局部：刚开始会为每个进程分配一定数量的物理块。当某进程发生缺页时，只允许从该进程自己的物理块中选出一个进行换出外存。如果进程在运行中频繁地缺页，系统会为该进程多分配几个物理块，直至该进程缺页率趋势适当程度；反之，如果进程在运行中缺页率特别低，则可适当减少分配给该进程的物理块。

可变分配**全局**置换：只要缺页就给分配新物理块

可变分配**局部**置换：要根据发生**缺页的频率**来动态地增加或减少进程的物理块

宿船长 B站专用

主要指空间局部性，即：如果当前访问了某个内存单元，在之后很有可能会接着访问与其相邻的那些内存单元。
(页面)

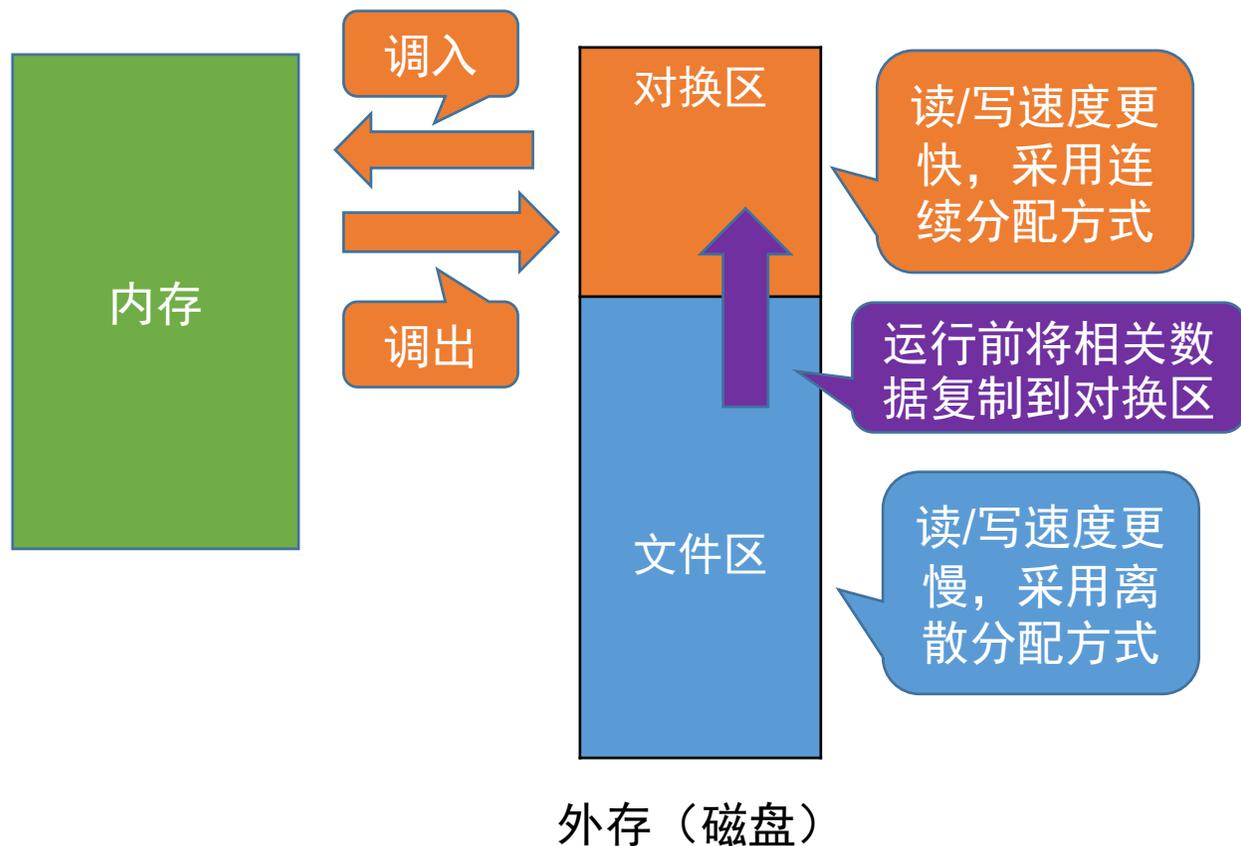
1. 预调页策略：根据局部性原理，一次调入若干个相邻的页面可能比一次调入一个页面更高效。但如果提前调入的页面中大多数都没被访问过，则又是低效的。因此可以预测不久之后可能访问到的页面，将它们预先调入内存，但目前预测成功率只有50%左右。故这种策略**主要用于进程的首次调入**，由程序员指出应该先调入哪些部分。

运行前调入

运行时调入

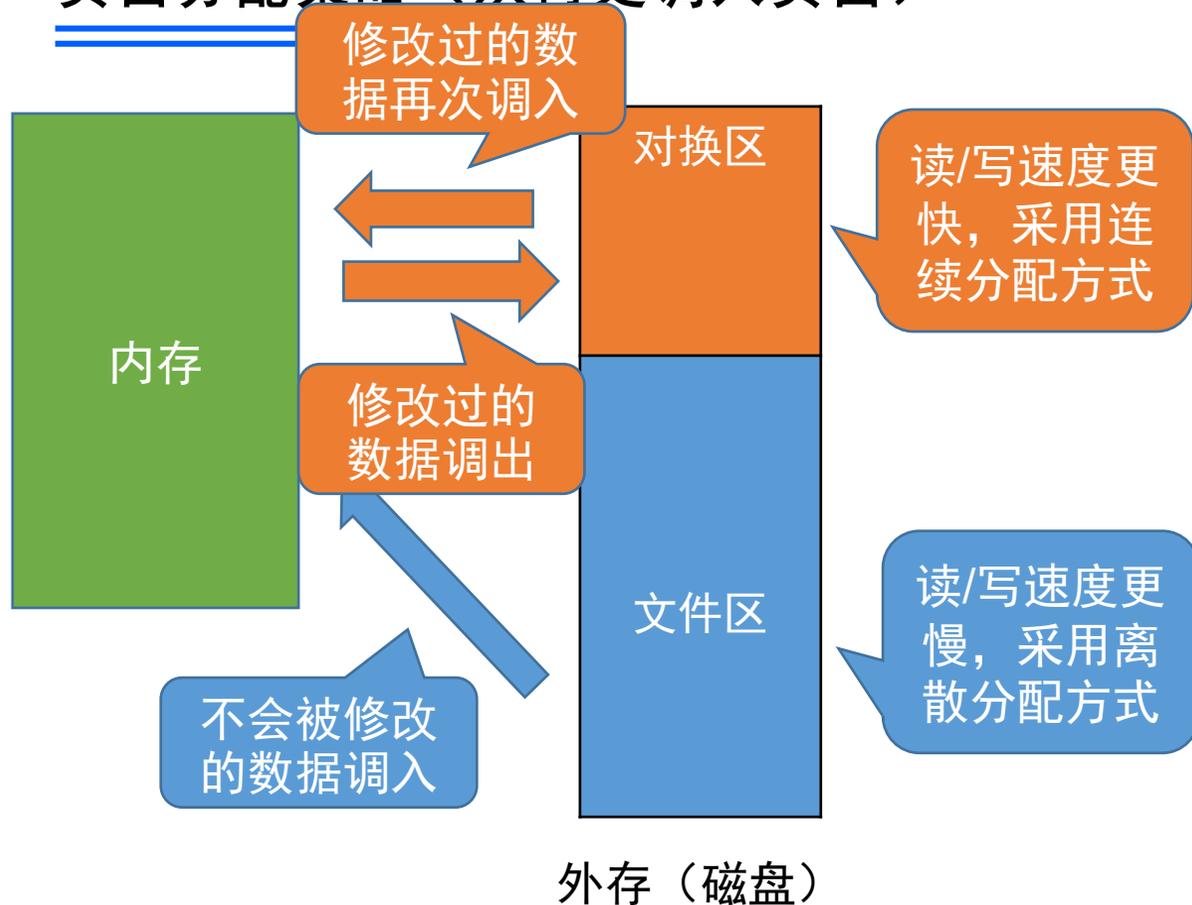
2. 请求调页策略：进程**在运行期间发现缺页时才将所缺页面调入内存**。由这种策略调入的页面一定会被访问到，但由于每次只能调入一页，而每次调页都要磁盘I/O操作，因此I/O开销较大。

页面分配策略（从何处调入页面）



1. 系统拥有足够的对换区空间：页面的调入、调出都是在内存与对换区之间进行，这样可以保证页面的调入、调出速度很快。在进程运行前，需将进程相关的数据从文件区复制到对换区。

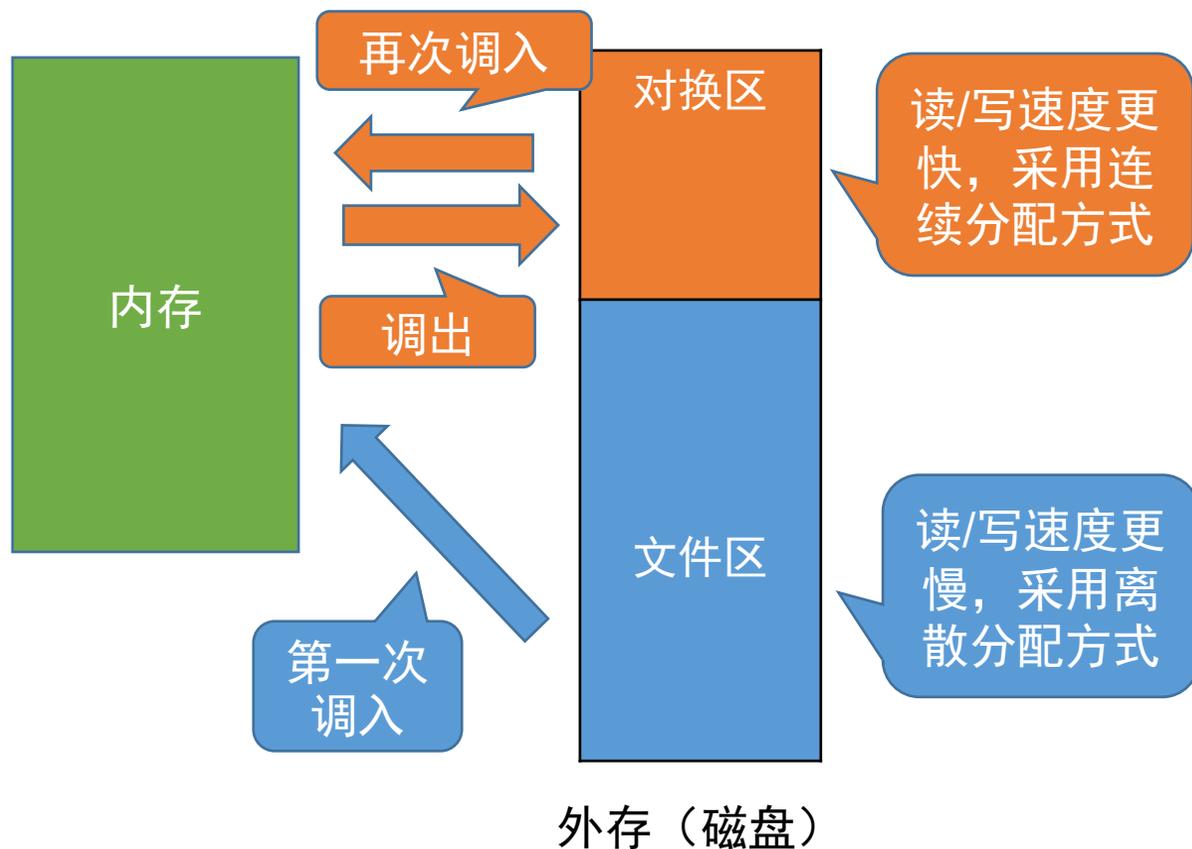
页面分配策略（从何处调入页面）



1. 系统拥有足够的对换区空间：页面的调入、调出都是在内存与对换区之间进行，这样可以保证页面的调入、调出速度很快。在进程运行前，需将进程相关的数据从文件区复制到对换区。

2. 系统缺少足够的对换区空间：凡是不会被修改的数据都直接从文件区调入，由于这些页面不会被修改，因此换出时不必写回磁盘，下次需要时再从文件区调入即可。对于可能被修改的部分，换出时需写回磁盘对换区，下次需要时再从对换区调入。

页面分配策略（从何处调入页面）



1. 系统拥有足够的对换区空间：页面的调入、调出都是在内存与对换区之间进行，这样可以保证页面的调入、调出速度很快。在进程运行前，需将进程相关的数据从文件区复制到对换区。

2. 系统缺少足够的对换区空间：凡是不会被修改的数据都直接从文件区调入，由于这些页面不会被修改，因此换出时不必写回磁盘，下次需要时再从文件区调入即可。对于可能被修改的部分，换出时需写回磁盘对换区，下次需要时再从对换区调入。

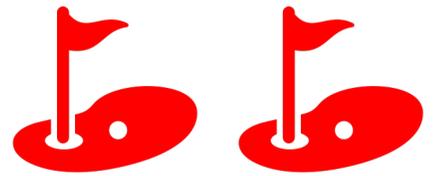
3. UNIX方式：运行之前进程有关的数据全部放在文件区，故未使用过的页面，都可从文件区调入。若被使用过的页面需要换出，则写回对换区，下次需要时从对换区调入。

页面分配策略（抖动（颠簸）现象）

刚刚换出的页面马上又要换入内存，刚刚换入的页面马上又要换出外存，这种频繁的页面调度行为称为**抖动**，或**颠簸**。产生抖动的**主要原因**是进程频繁访问的页面数目高于可用的物理块数（**分配给进程的物理块不够**）

为进程分配的物理块太少，会使进程发生抖动现象。为进程分配的物理块太多，又会降低系统整体的并发度，降低某些资源的利用率

为了研究为应该为每个进程分配多少个物理块，Denning提出了进程“工作集”的概念

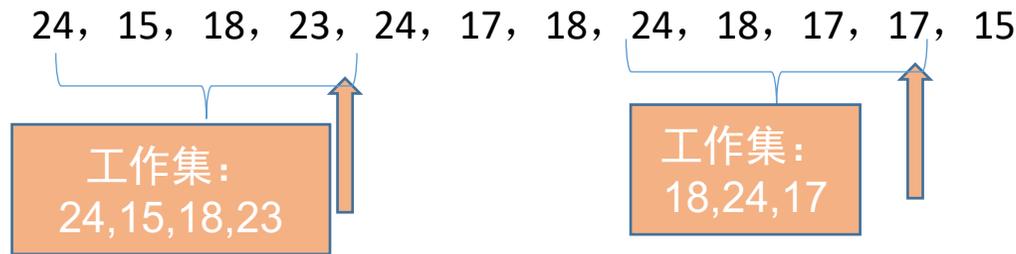


页面分配策略（工作集）

驻留集：指请求分页存储管理中给进程分配的内存块的集合。

工作集：指在某段时间间隔里，进程实际访问页面的集合。

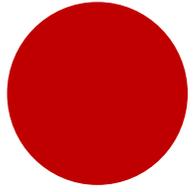
操作系统会根据“窗口尺寸”来算出工作集。例：某进程的页面访问序列如下，窗口尺寸为4，各时刻的工作集为？



工作集大小可能小于窗口尺寸，实际应用中，操作系统可以统计进程的工作集大小，根据工作集大小给进程分配若干内存块。如：窗口尺寸为5，经过一段时间的监测发现某进程的工作集最大为3，那么说明该进程有很好的局部性，可以给这个进程分配3个以上的内存块即可满足进程的运行需要。

一般来说，**驻留集大小不能小于工作集大小**，否则进程运行过程中将频繁缺页。

拓展：基于局部性原理可知，进程在一段时间内访问的页面与不久之后会访问的页面是有相关性的。因此，可以根据进程近期访问的页面集合（工作集）来设计一种页面置换算法——选择一个不在工作集中的页面进行淘汰。



拜拜

宿船长 B站专用