操作系统

Operating system

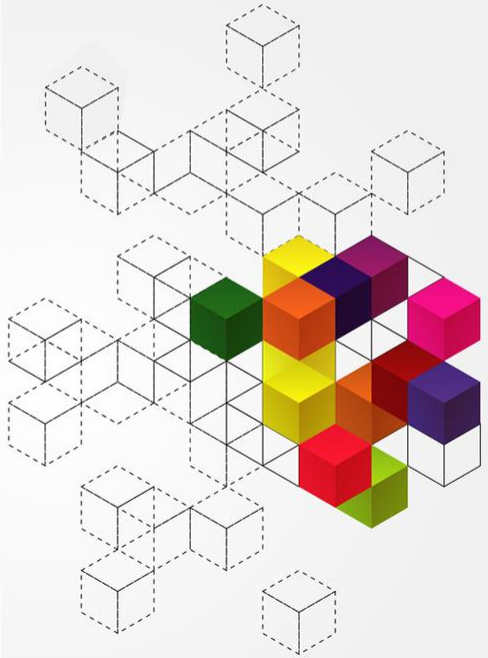
胡燕

大连理工大学

页置换算法练习

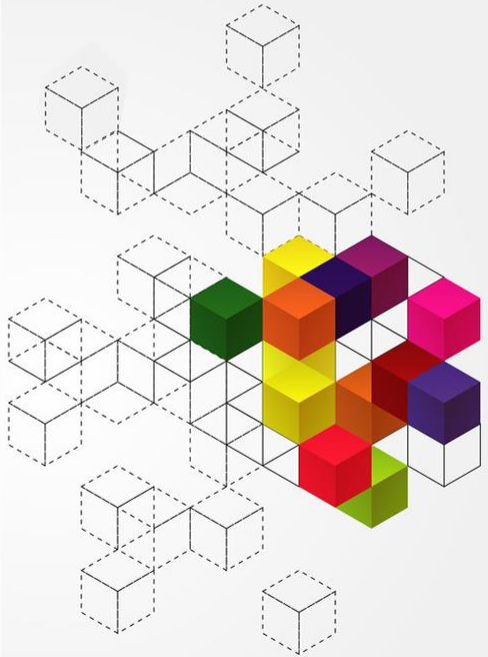
课前慕课堂练习。

注意：页故障次数、发生页置换次数的区别



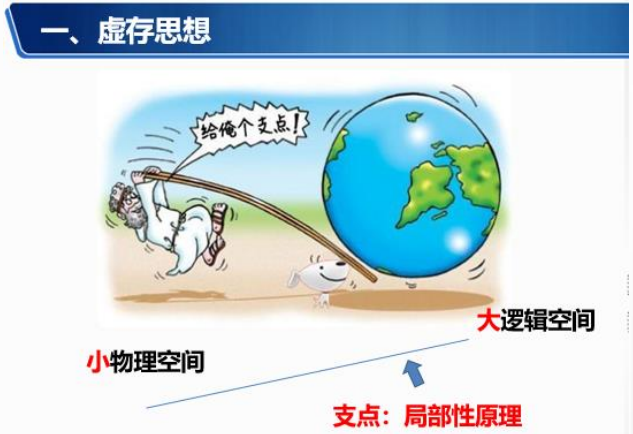
页置换算法练习

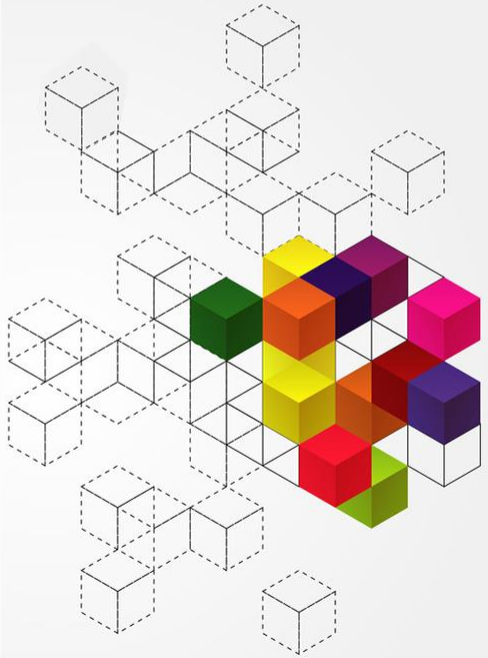
在一个请求分页系统中，一个进程的页面访问序列为：6,0,1,2,0,3,0,4,2,3，分配该进程的物理页数为3，请分别计算FIFO置换算法、LRU置换算法，访问过程中所发生的缺页次数和缺页率。



个人风貌展示

谈谈你对程序局部性（locality）的理解。
2021.5.20





内容纲要

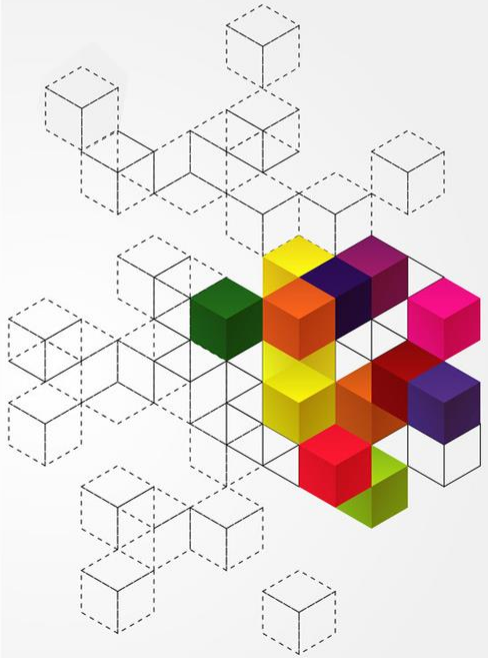
9.3 页置换算法实现

一、 页置换算法实现考虑

二、 近似实现1：附加引用位

三、 近似实现2：时钟算法

四、 近似实现3：增强型二次机会算法

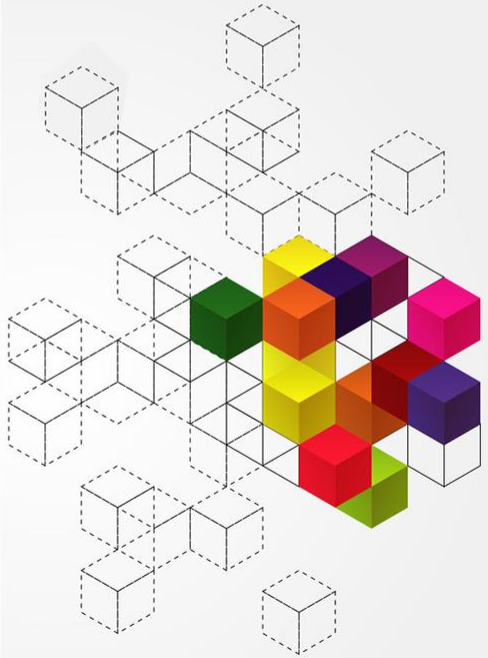


一、页置换算法的实现考虑

- **LRU算法是OPT算法的近似，不会产生Belady异常**
  - 是页置换算法的首选

问题： **使用栈来精确实现LRU算法，可能使得系统内存访问效率下降为原来的1/10**

不可接受！



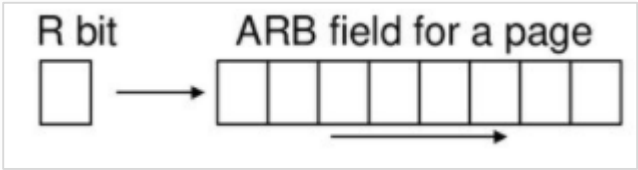
二、近似实现1：附加引用位

LRU精确实现，性能差，不实用

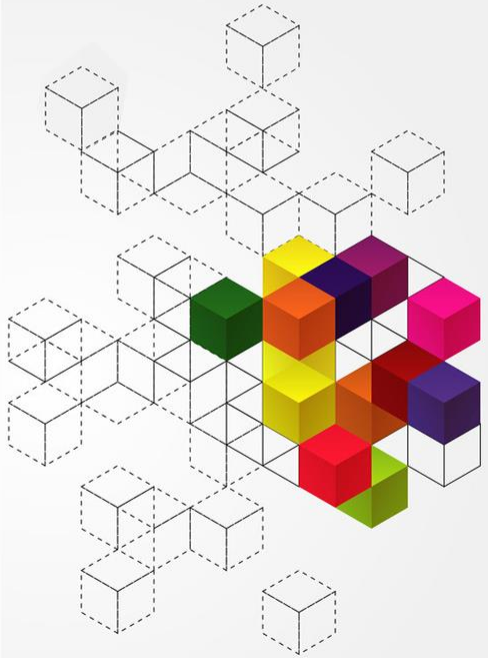
考虑对LRU进行近似实现

策略1：附加引用位

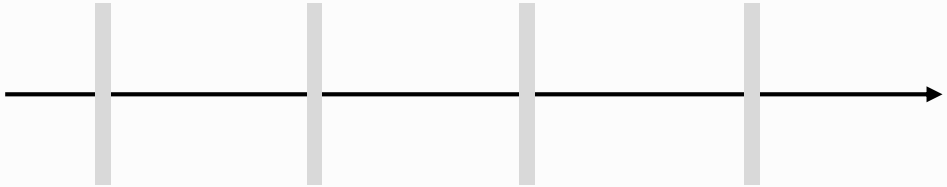
除了页面引用位R外，为每个页面增加额外的引用位
Additional Reference Bit (ARB)



At each timer interrupt, the R bit is shifted from left into the ARB, and the ARB shift to right accordingly



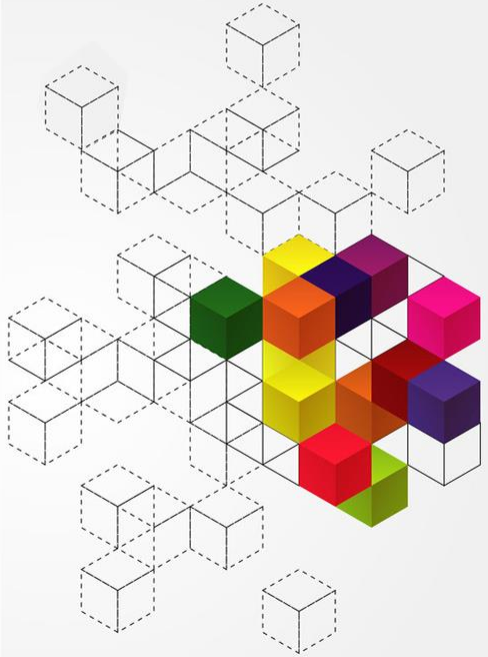
二、近似实现1：附加引用位



		tick1	tick2	tick3	tick4
R:		0	1	0	1
ARB:	0000000	0000000	1000000	0100000	1010000

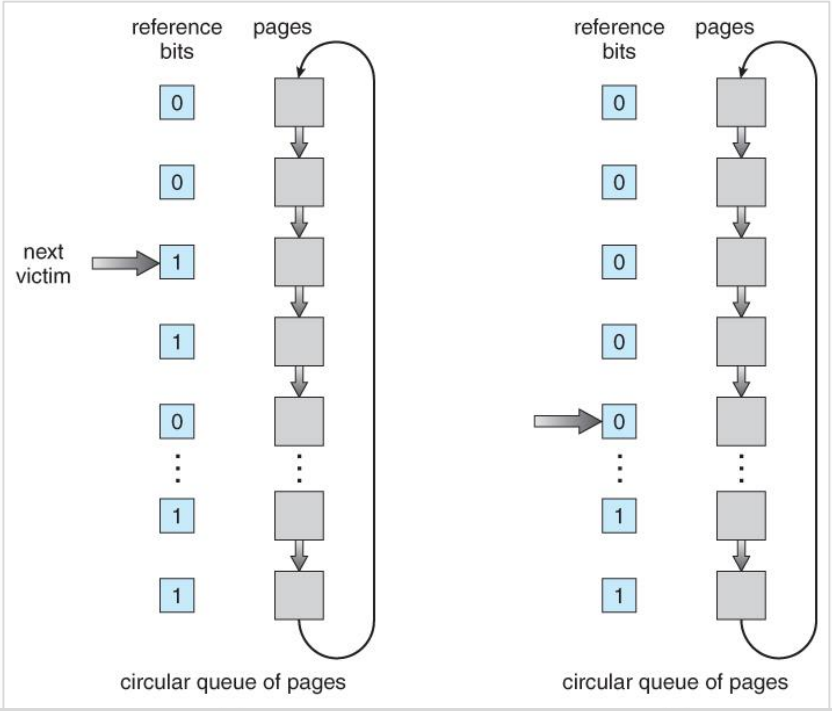
Q: How to select the victim frame?

select the page with the largest ARB



三、近似实现2：时钟算法

Second-Chance Page Replacement Algorithm

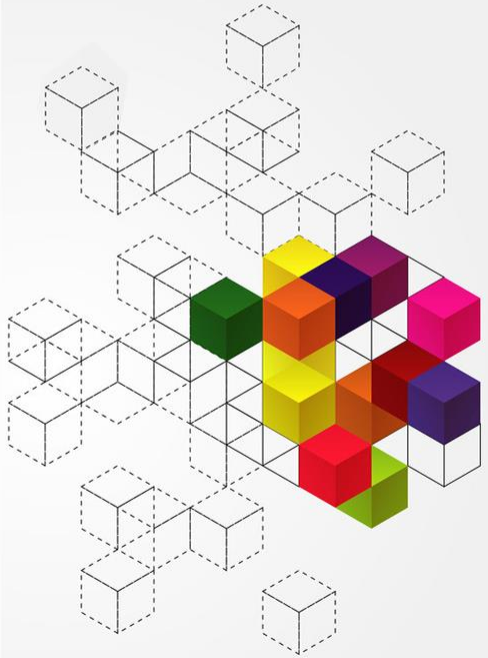


① 页面组织成环

② 每次页置换后，将next victim指针指向被置换的后继页面

③ 如需要进行置换，从当前next victim指向的位置开始，寻找第一个引用为为0的页，搜索途中遇到的引用位为1的改为0

引用位为1的页面，会赢得驻留内存的第2次机会

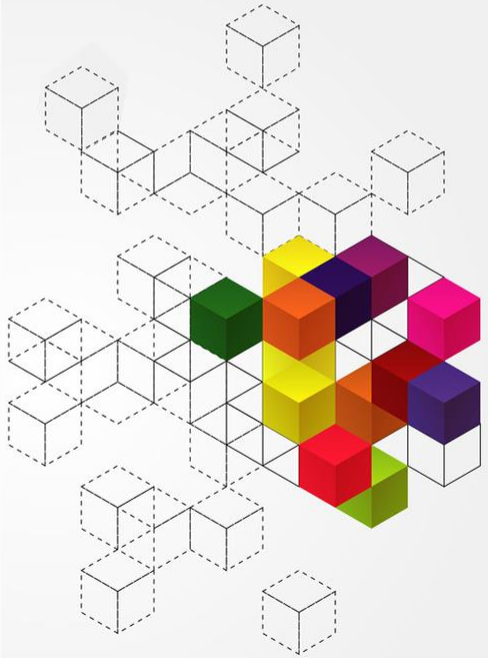


四、近似实现3：增强型二次机会算法

- **原理**：引入引用位(r)和修改位(c)作为有序对

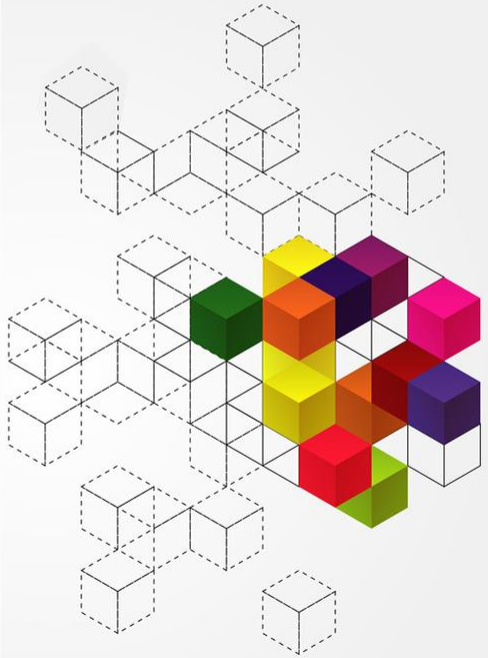
(r,c)	页面描述
(0,0)	最近未使用，也未修改过（置换最佳候选）
(0,1)	最近未使用，但被修改过
(1,0)	最近使用过，但未被修改
(1,1)	最近使用过，也被修改过

置换时，优先选择rc位为00的页面置换，01，10，11次之



本讲小结

- 页置换算法近似实现方法讨论



内容纲要

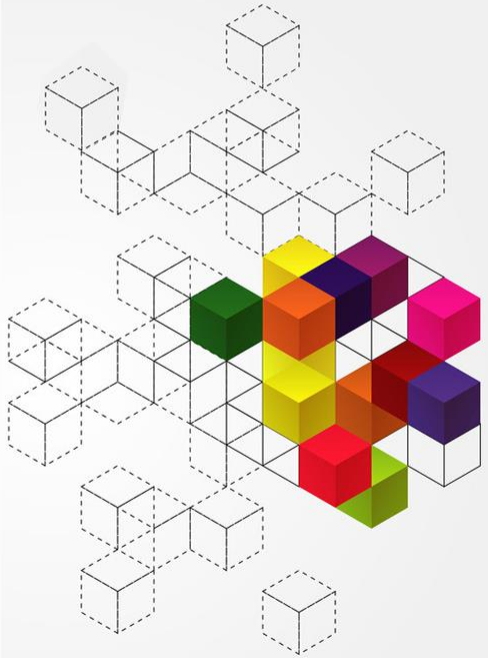
9.4 页帧分配

一、 页帧分配策略

二、 内存抖动

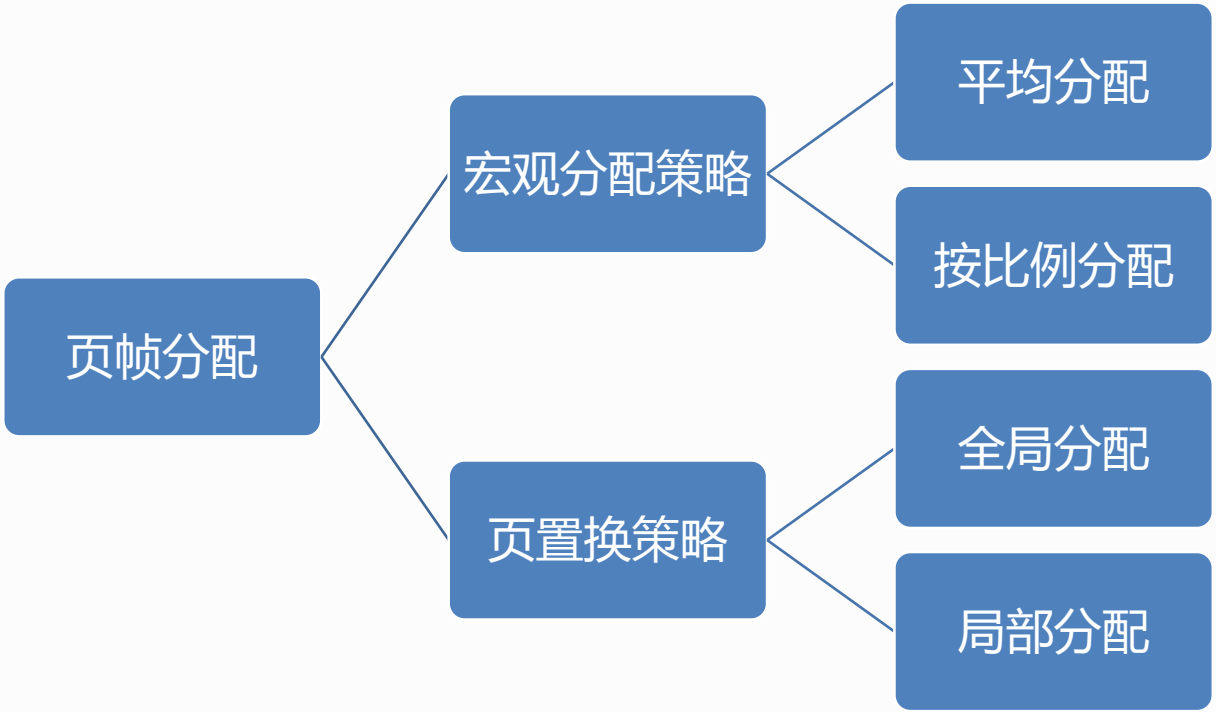
三、 工作集模型简介

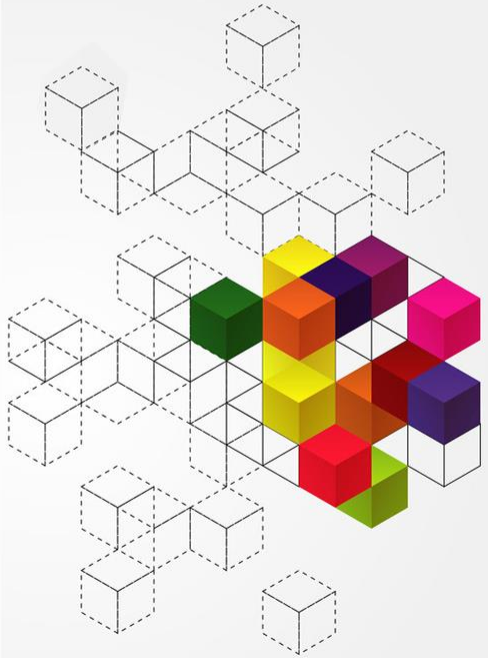
四、 工作集模型实现



一、页帧分配原则

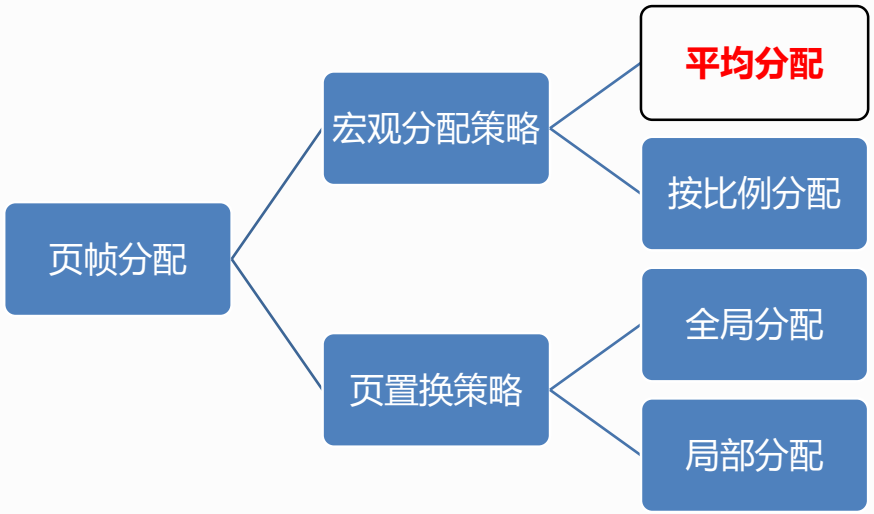
- 要解决的基本问题：**如何为每个进程分配物理内存，应该分配多大的物理内存?**

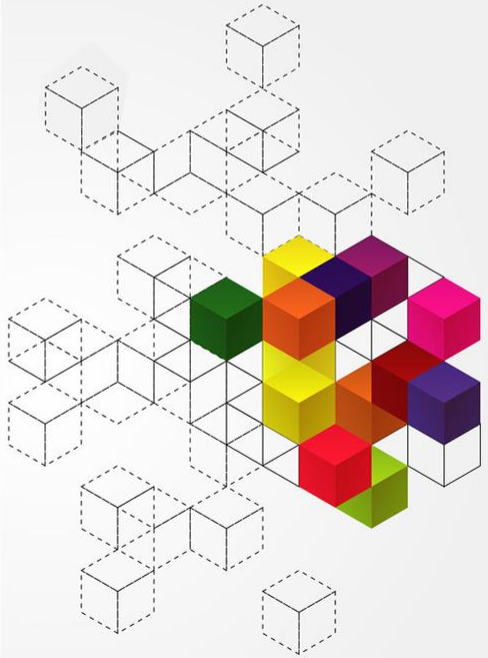




一、页帧分配原则

- 宏观分配策略1：**平局分配（Equal Allocation）**
  - **每个用户进程被分配相同数量的页帧**





一、页帧分配原则

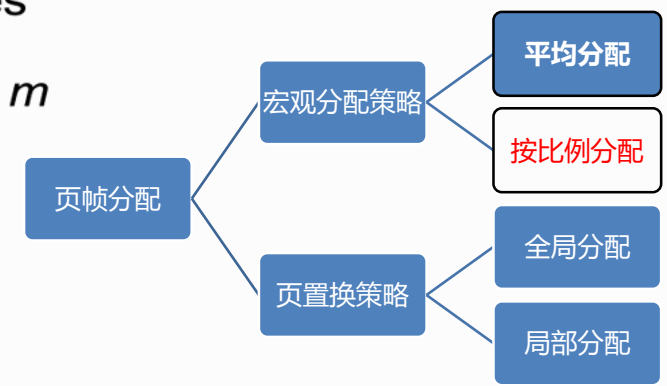
- 宏观分配策略2：**按比例分配（Priority Allocation）**
  - **按照进程的大小，成比例分配**

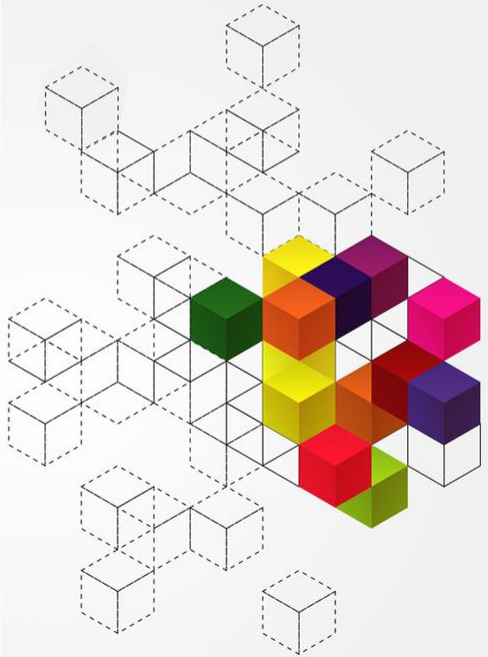
s_i = size of process p_i

$S = \sum s_i$

m = total number of frames

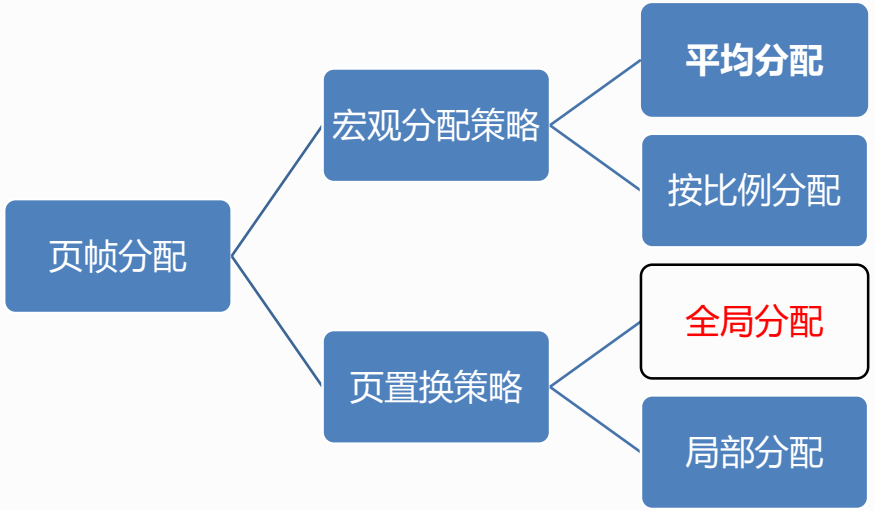
a_i = allocation for $p_i = \frac{s_i}{S} \times m$

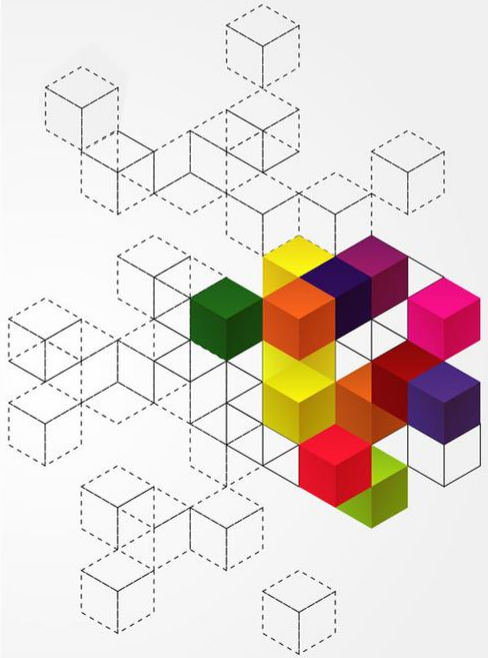




一、页帧分配原则

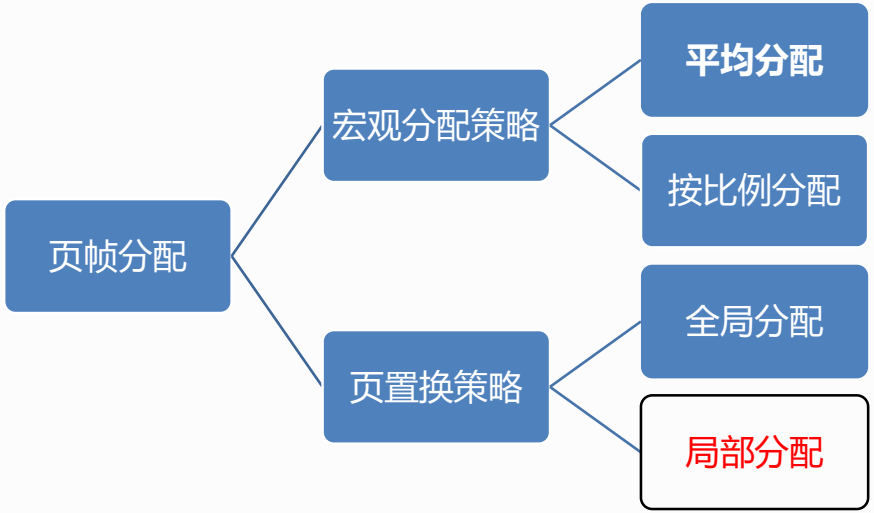
- 页置换策略1：**全局置换（Global Replacement）**
  - **当需要进行页置换时，选择牺牲页帧的范围为所有进程的已分配物理页框集合**

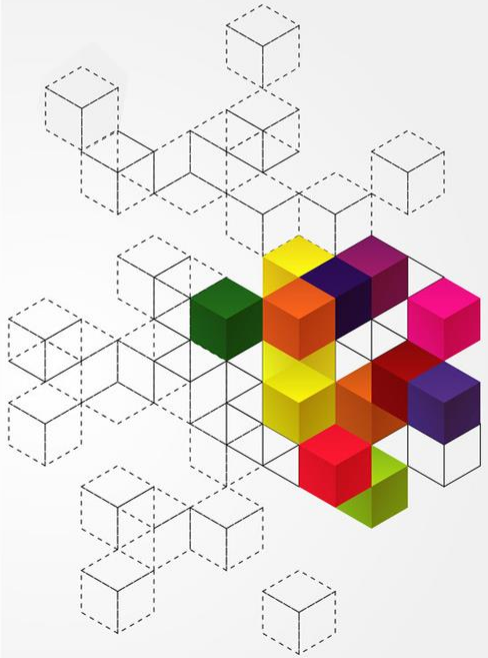




一、页帧分配原则

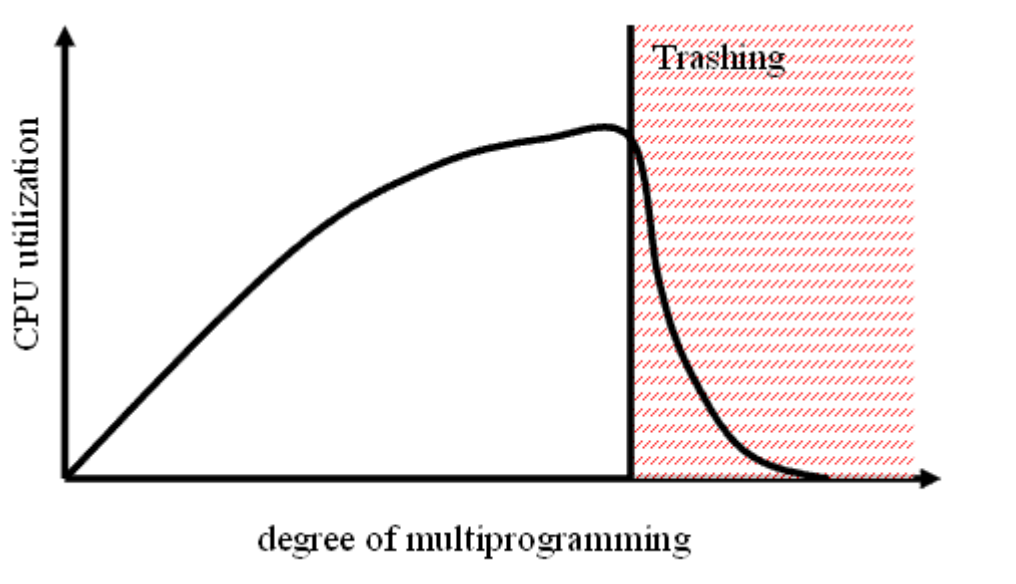
- 页置换策略2：局部置换（Local Replacement）
  - 当需要进行页置换时，选择牺牲页帧的范围为当前发生置换的进程的已分配物理页框集合

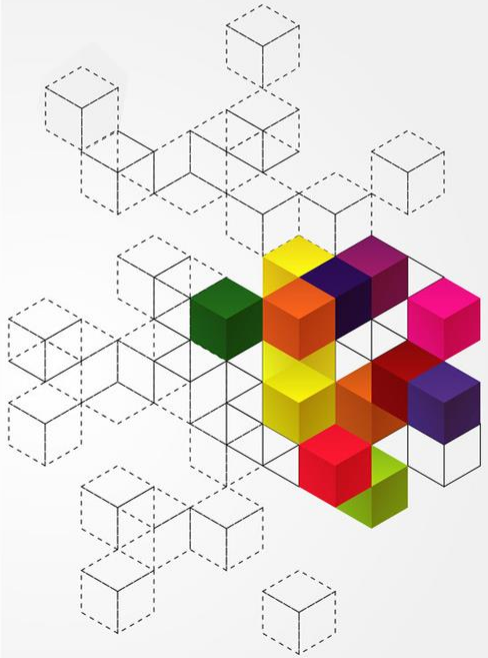




二、内存抖动

- **Thrashing**
  - 当进程分配的物理内存不足时，导致内存与外存之间的数据交换异常增加的现象，被称为抖动





三、工作集模型简介

- **为了避免抖动，同时又能够进行有效的物理内存分配，需要对进程局部性进行建模**

工作集模型是进行局部性建模的一种代表性方法

- **工作集模型基本思想：观察定长时间段内进程访问的内存集合，并以此为依据确定该时间段内为该进程分配物理页（页帧）的数量**

工作集用途：刻画程序的局部性信息

三、工作集模型简介

- **设定工作集的时间窗口=10**

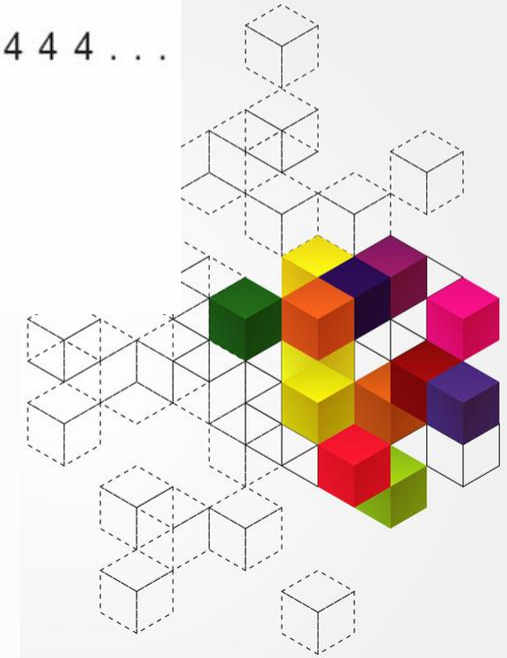
page reference table

. . . 2 6 1 5 7 7 7 7 5 1 6 2 3 4 1 2 3 4 4 4 3 4 3 4 4 4 1 3 2 3 4 4 4 3 4 4 4 . . .

令$D=\Sigma\ WS_i(t)$

m=t时刻操作系统为所有进程分配的物理页数

如果D>m，会有什么后果?



三、工作集模型实现

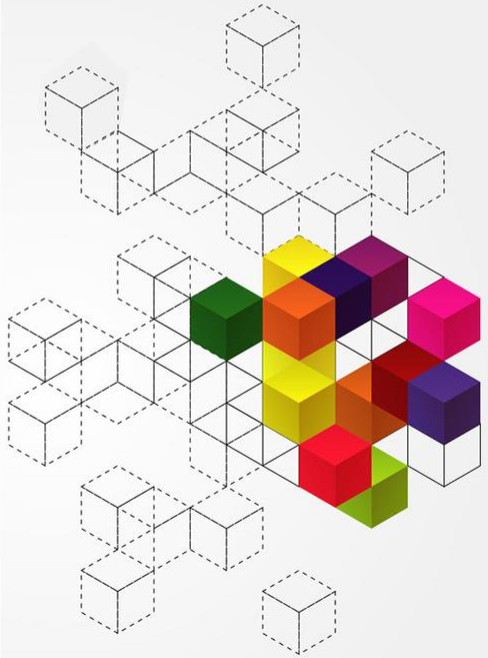
- **基于工作集模型，为进程分配物理页帧的方法**

  计算进程的工作集

  为进程分配大于等于工作集大小的物理页帧

为了合理地分配物理页，必须持续更新进程的工作集

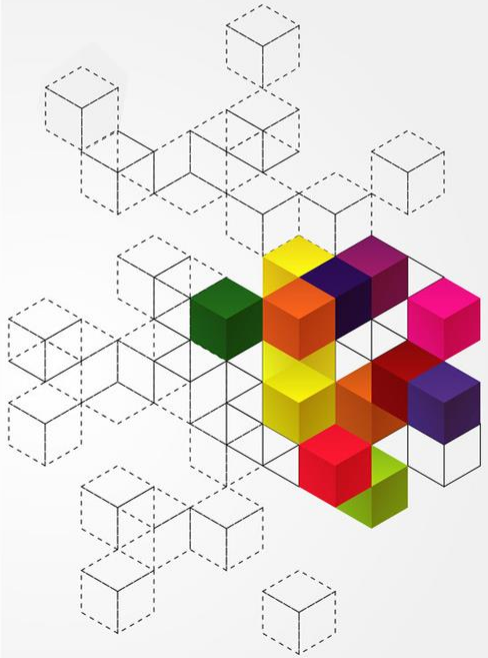
问题：过于频繁的工作集更新，会对程序执行性能产生较大负面影响



三、工作集模型实现

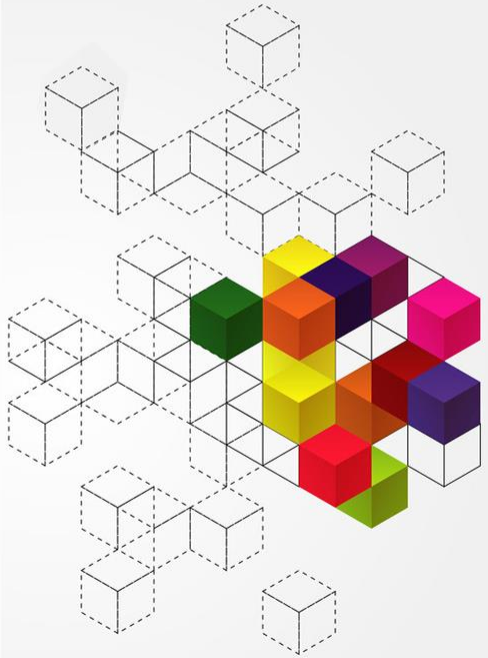
- 实现策略：减少采样频率
- 实际做法：计时器 + 引用位
- 关键参数：工作集窗口Δ，时钟中断间隔ti

Δ=10000，ti=5000
为每个页面维护一个引用位，以及2个额外的位
2个额外的位用来记录在过去的两个长为5000的时间区间内页面有没有被引用
时钟中断发生时，如果这3个位至少有一个值为1，则对应的页面在工作集中



本讲小结

- 页帧分配策略及实现方法



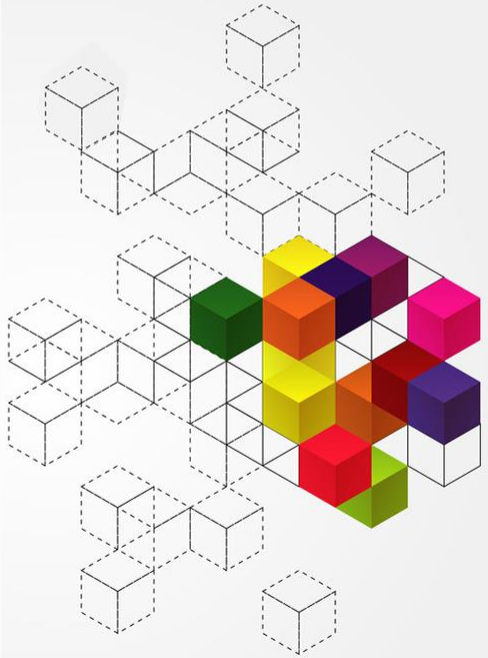
内容纲要

9.6 内核内存分配

一、 内核内存分配需求

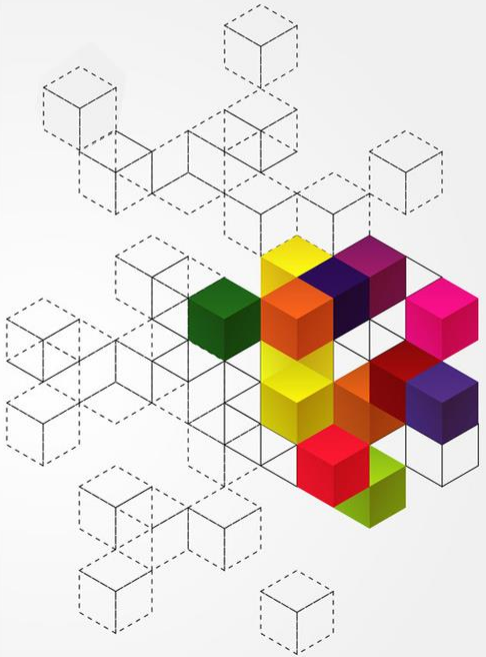
二、 伙伴算法

三、 Slab算法



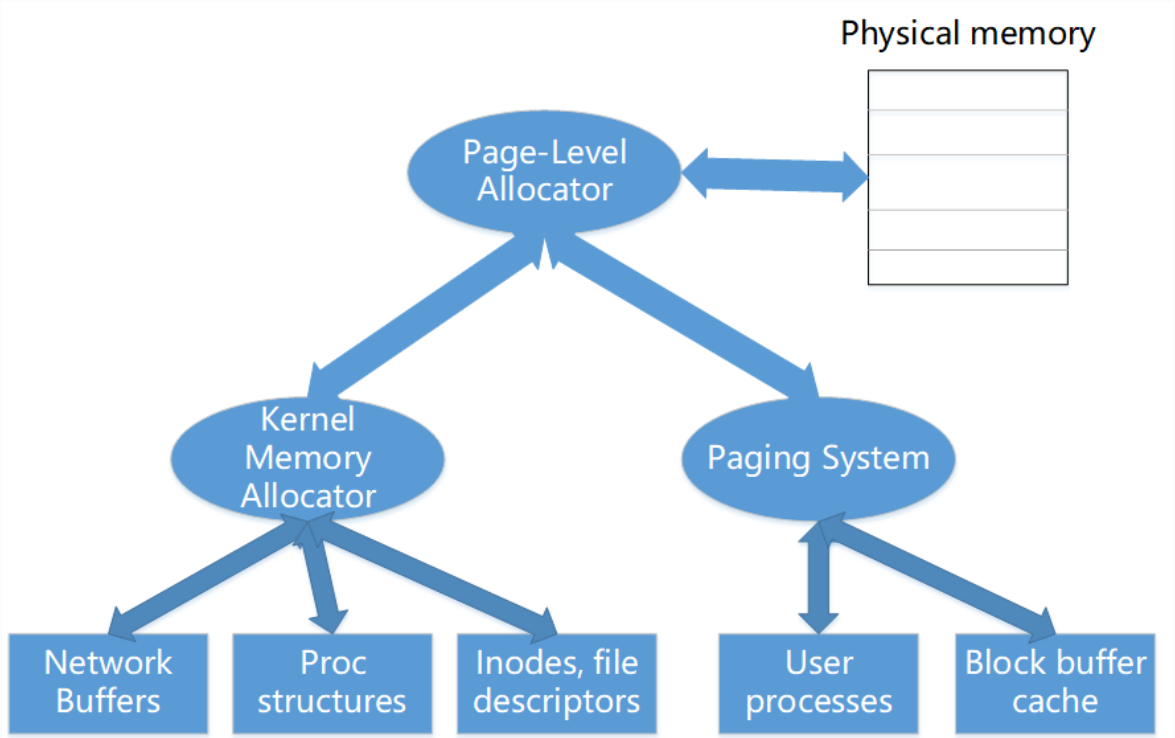
一、内核内存分配需求

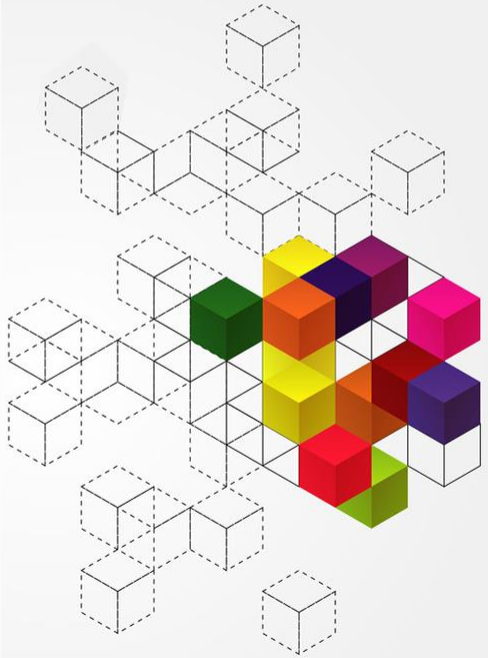
- **操作系统内核的内存分配需求**
  - 与用户态内存分配需求有所区别
  - 内核中有多种固定结构的内核对象，它们通常需要固定大小的内存
  - 内核对象通常要求放在连续的物理内存，以保证内核执行效率



一、内核内存分配需求

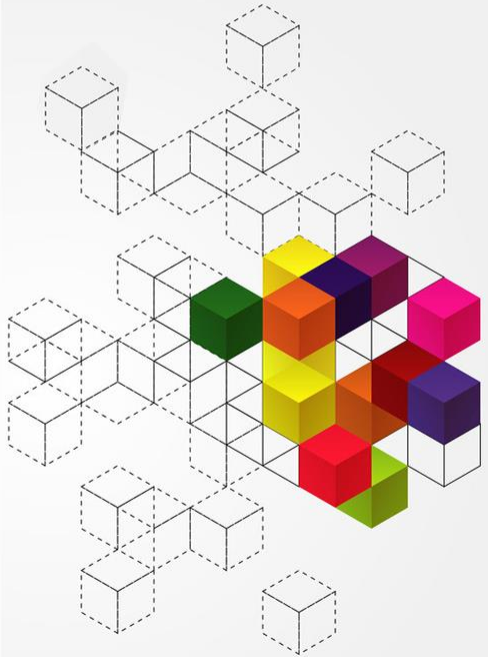
- OS页面级内存分配器与内核内存分配器





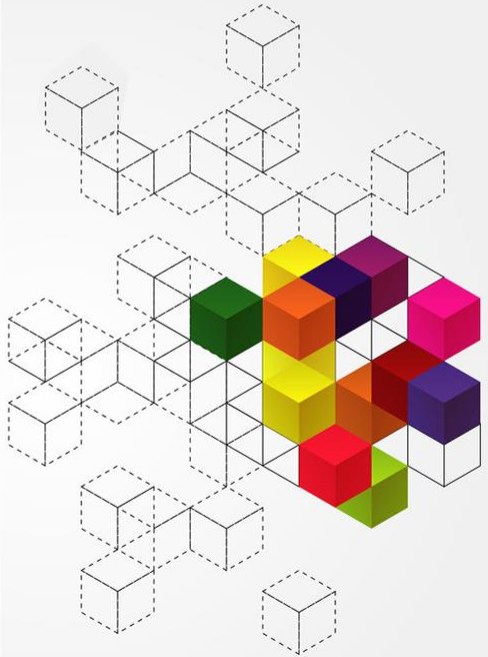
二、伙伴算法

- **基本思想：从物理上连续的内存区域分配连续的内存块**
  - Create small buffers by repeatedly halving a large buffer (buddy pairs) and coalescing adjacent buffers when possible
  - Requests rounded up to a power of two



二、伙伴算法

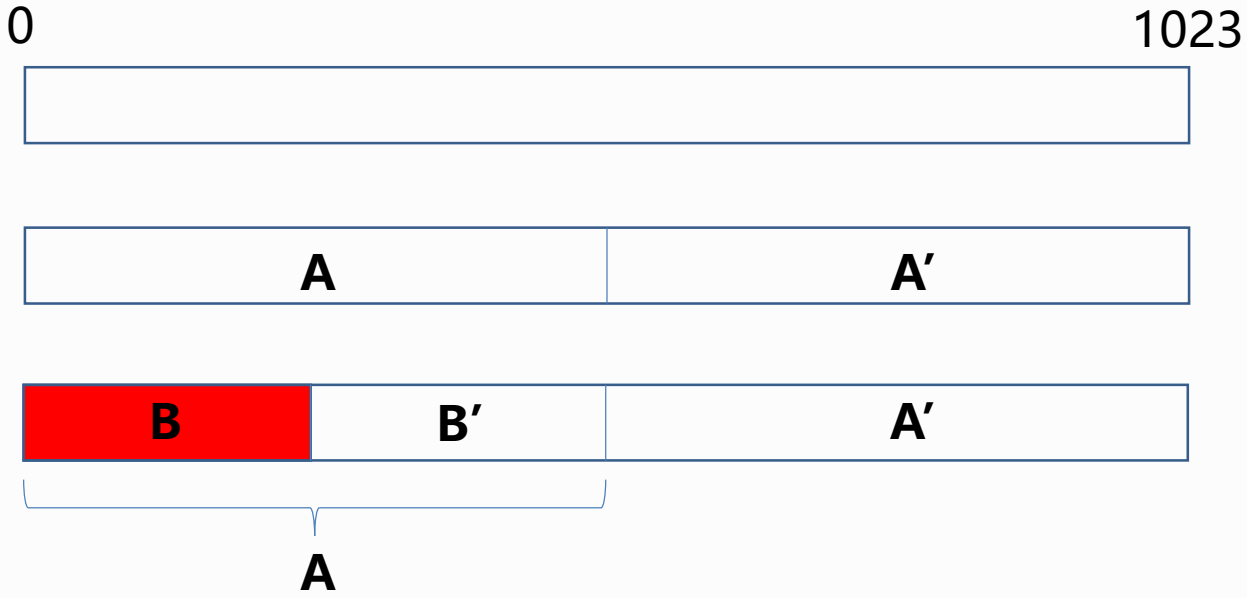
- **Buddy System Example**
  - Minimum allocation size = 32 Bytes
  - Initial free memory size is 1024
  - Use a bitmap to monitor 32 Byte chunks
    - Bit set if chunk is used
    - Bit clear if chunk is free
  - Maintain freelist for each possible buffer size
    - Power of 2 buffer sizes from 32 to 512
    - Sizes = {32, 64, 128, 256, 512}
  - Initial one block = entire buffer

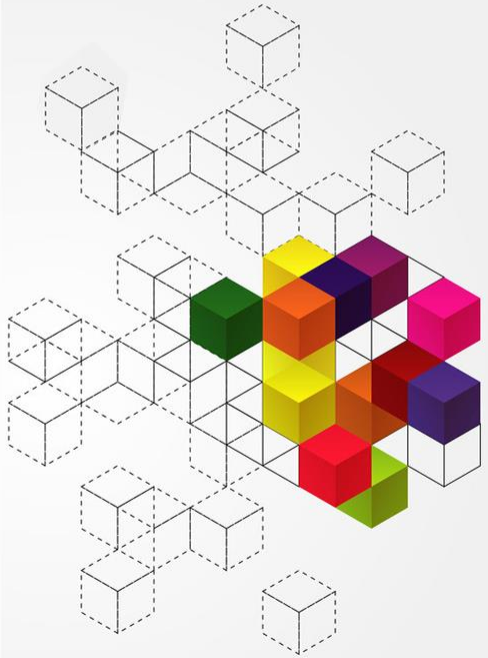


二、伙伴算法

- 伙伴算法示例

Allocate(256)

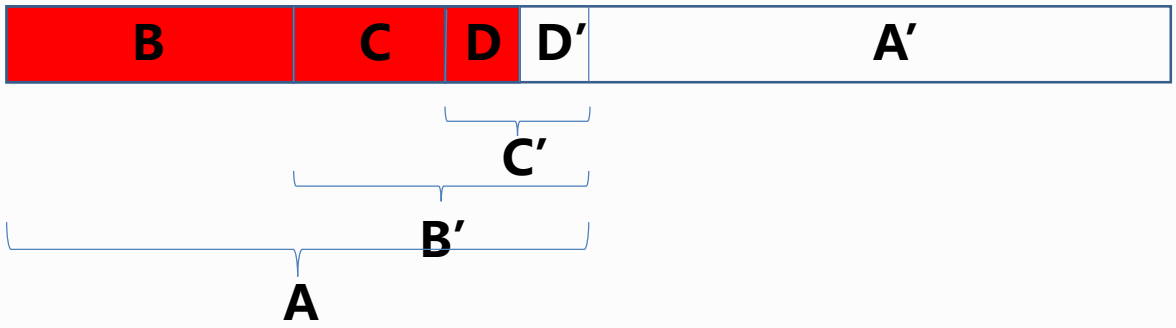


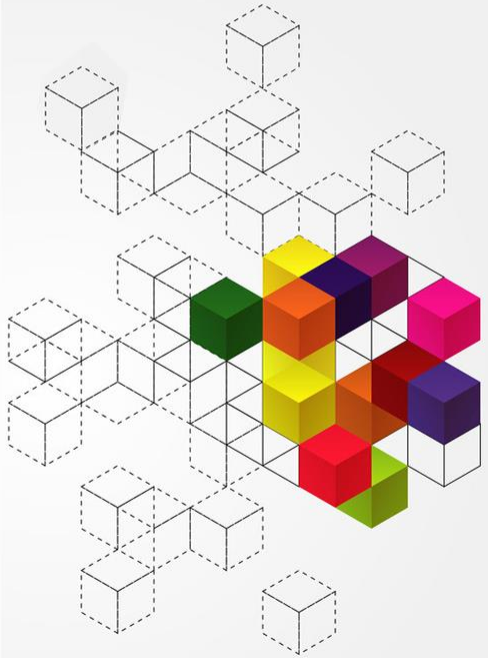


二、伙伴算法

- 伙伴算法示例

→ Allocate(128) → Allocate(64)

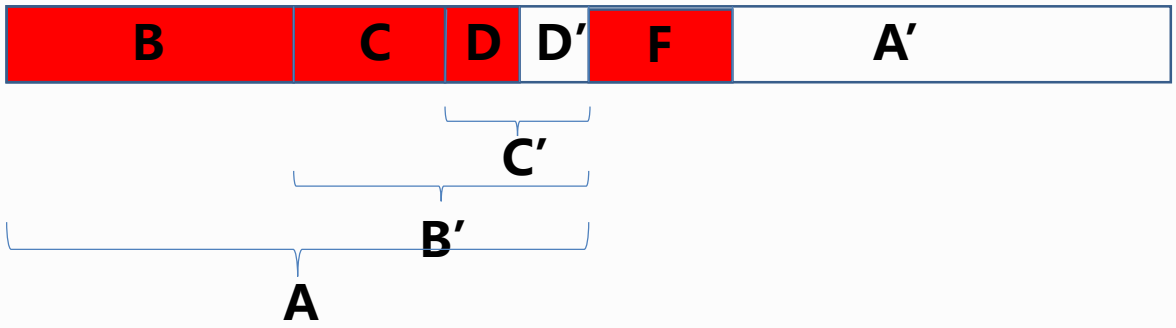


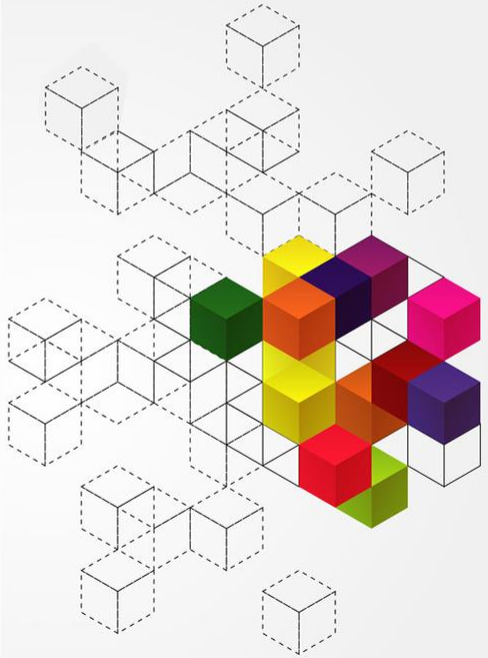


二、伙伴算法

- 伙伴算法示例

→ Allocate(128)



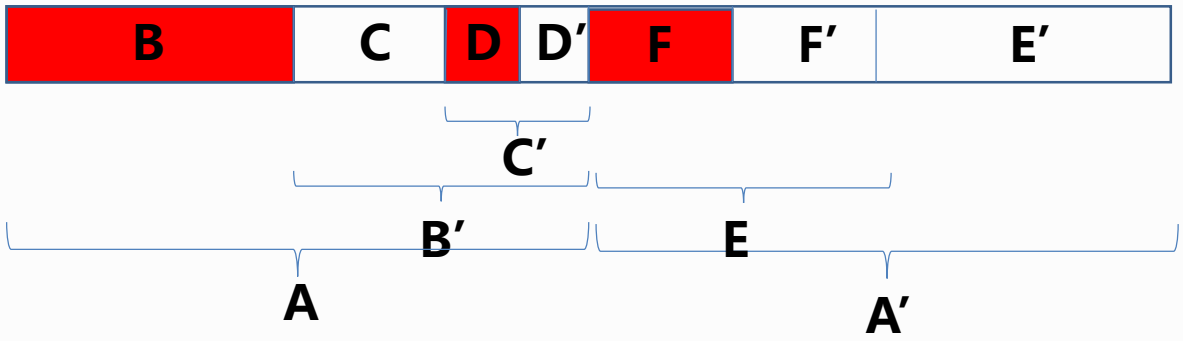


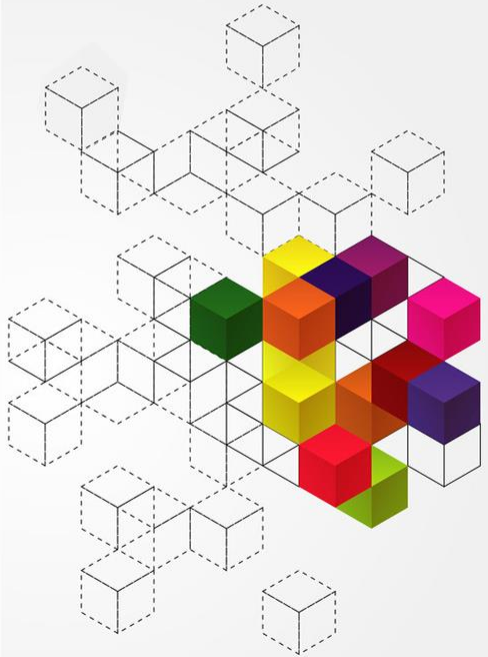
二、伙伴算法

- 伙伴算法示例

→ release(C,128)

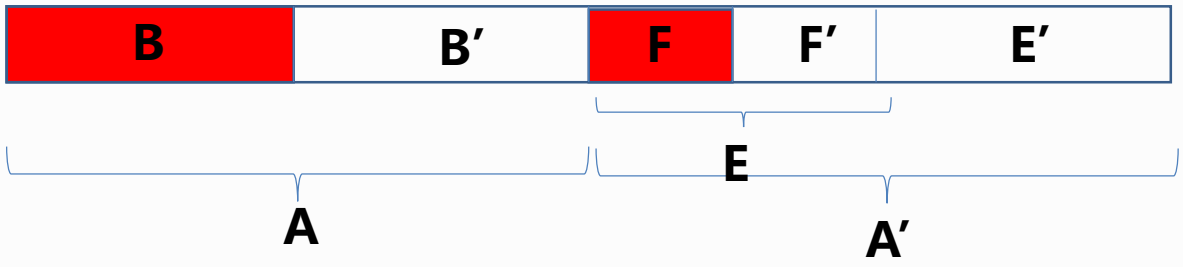
→ release(D,64)

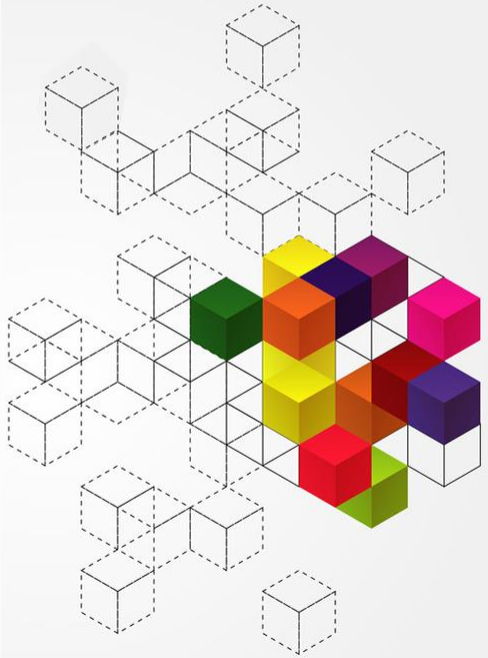




二、伙伴算法

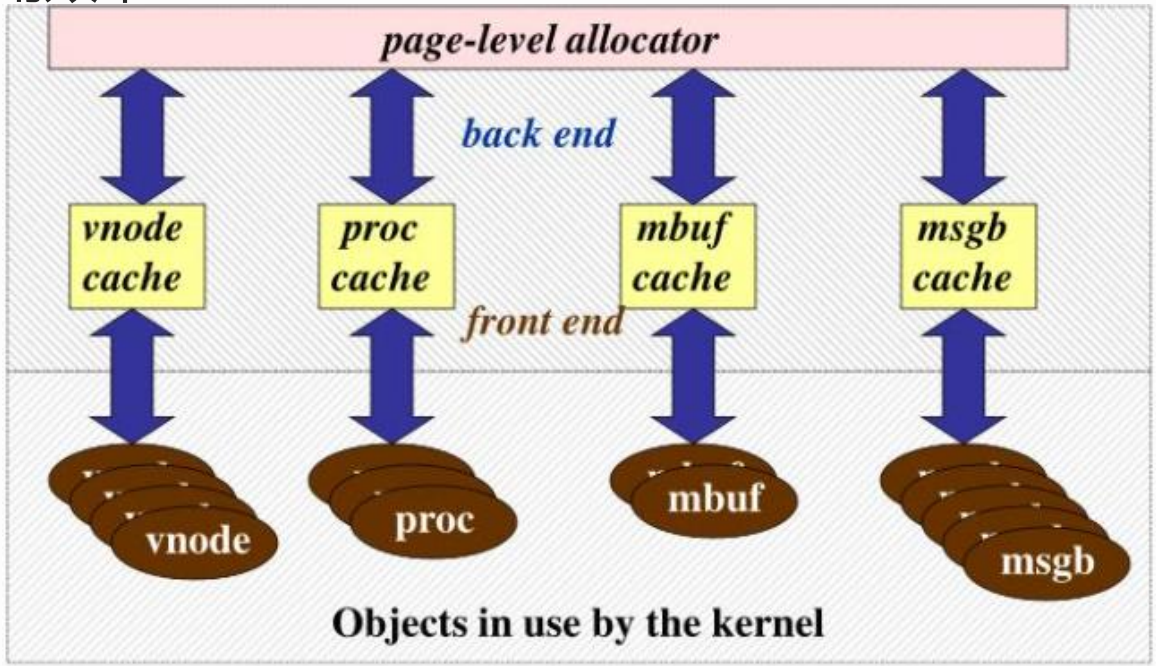
- 伙伴算法示例

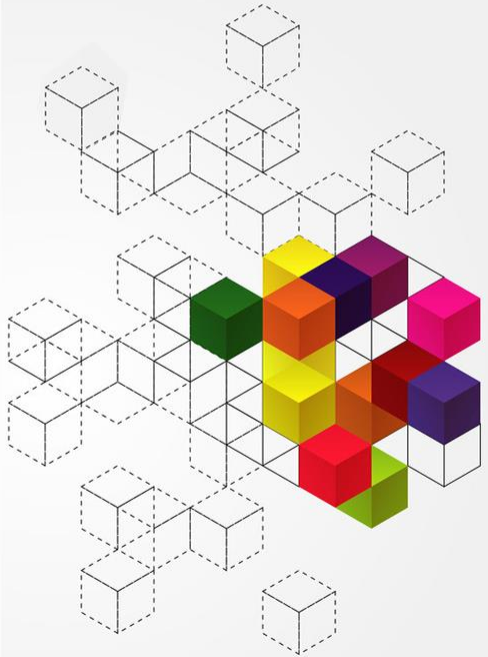




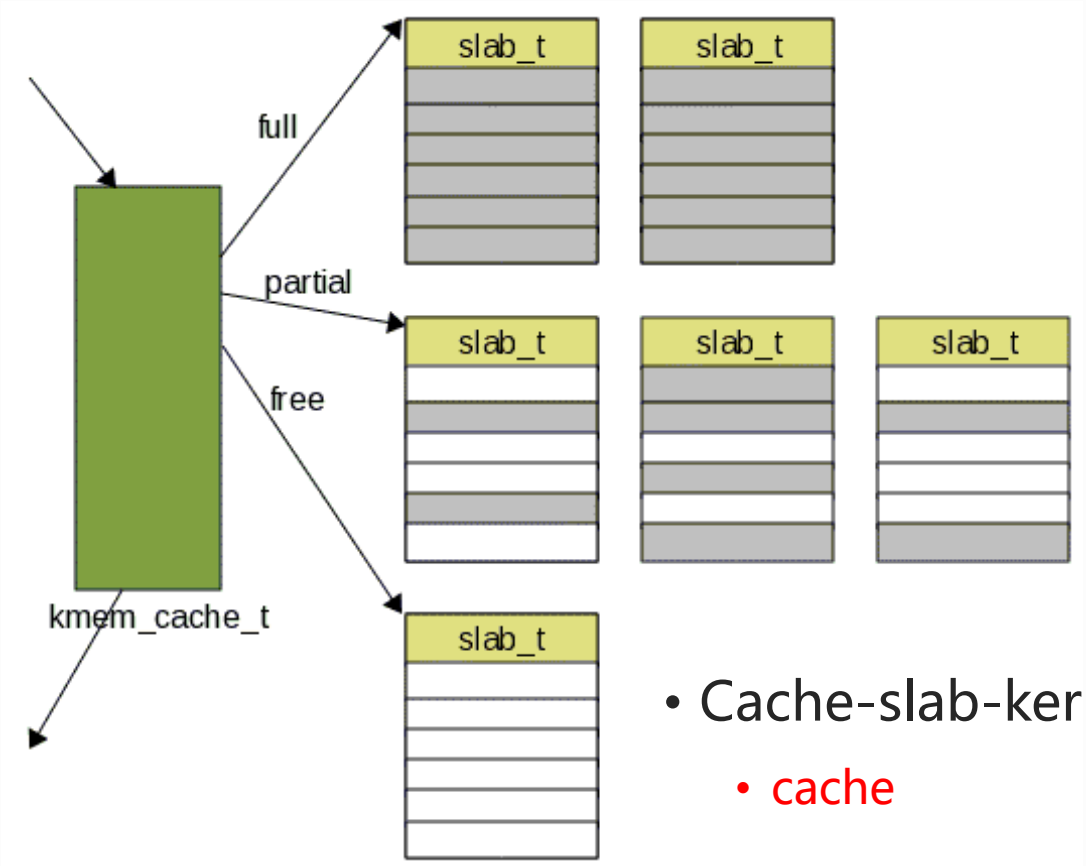
三、Slab算法

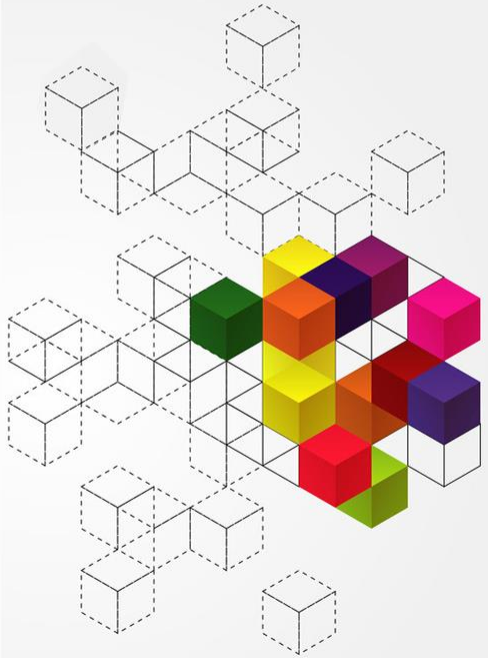
- 基本思想：针对内核数据结构对象的特点，设置不同大小的对象缓冲区，以获得较高的内存分配与使用效率





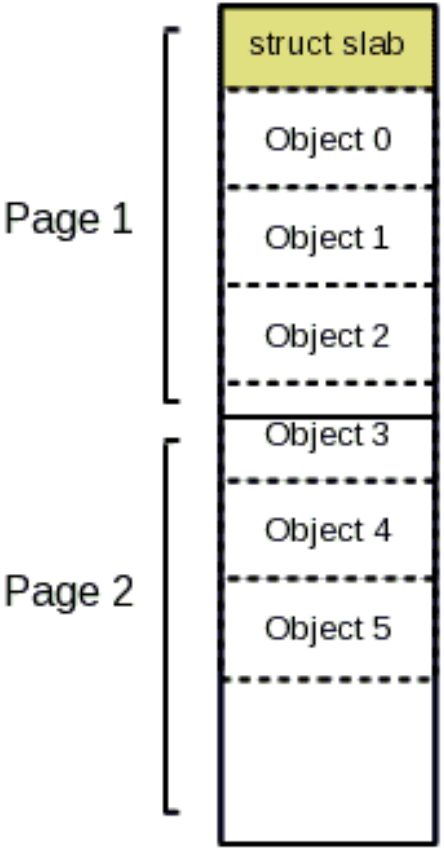
三、Slab算法



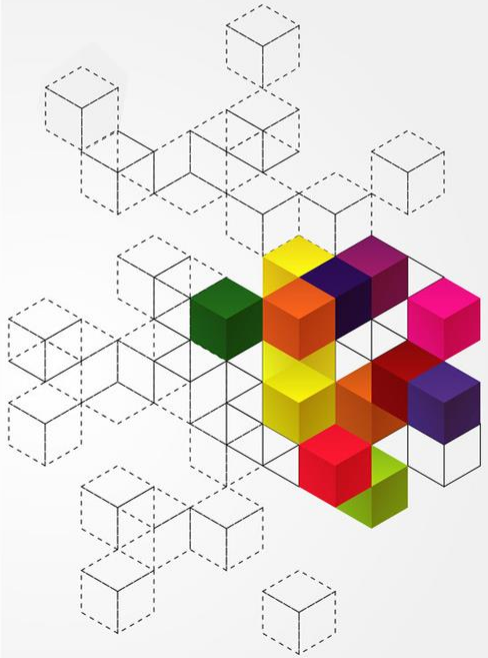


- Cache-slab-kernel object
  - cache

三、Slab算法

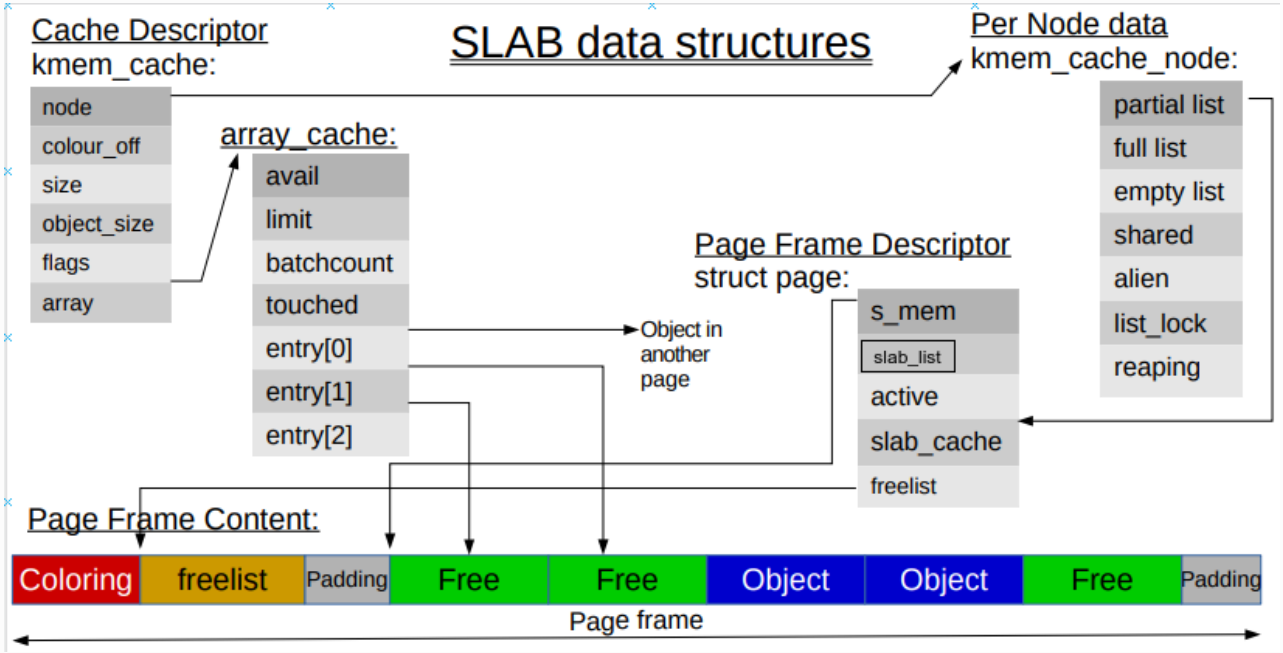


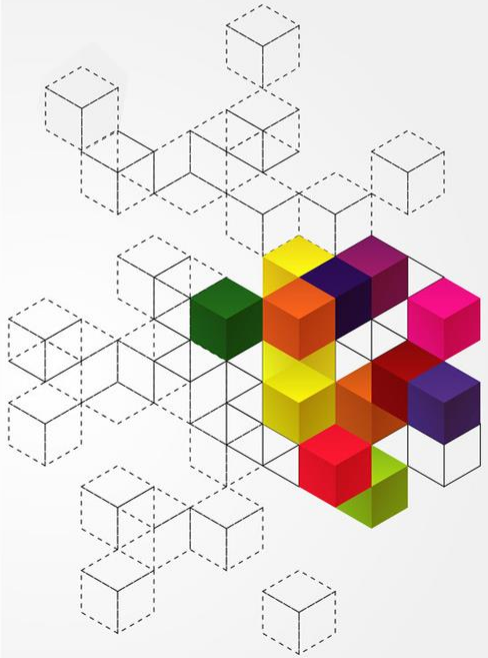
- Cache-slab-kernel object
  - Slab



三、Slab算法

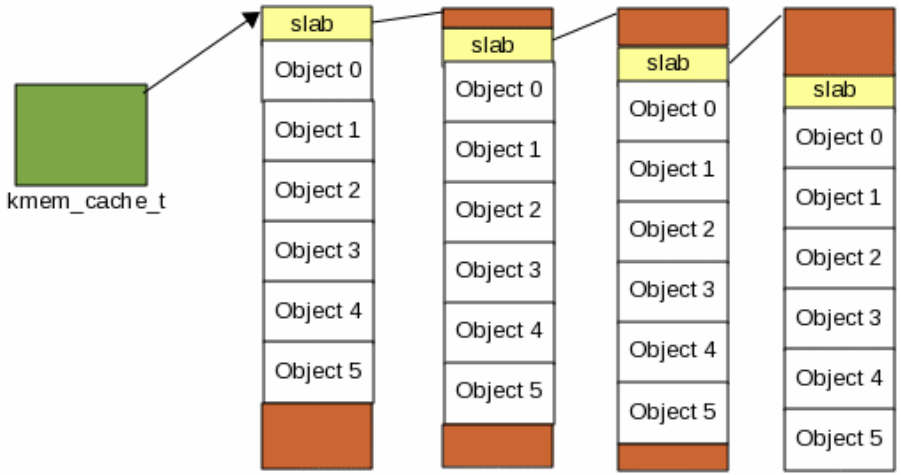
- **Cache-slab-kernel object**
  - Slab (Data Structure)

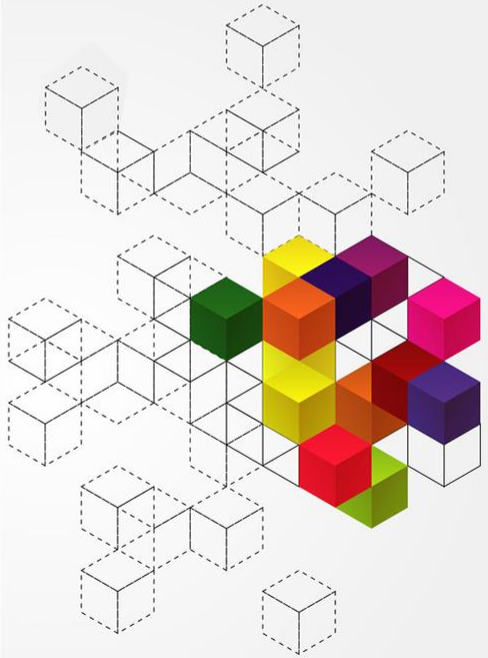




三、Slab算法

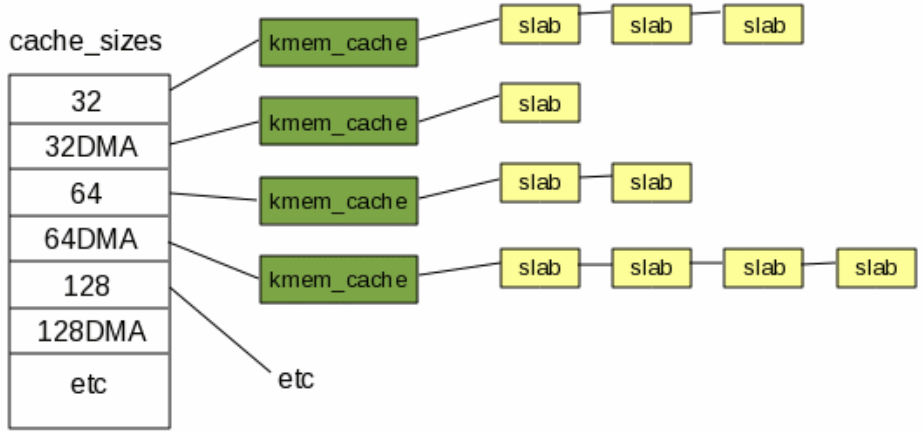
- Cache-slab-kernel object
  - Slab (Cache Coloring)

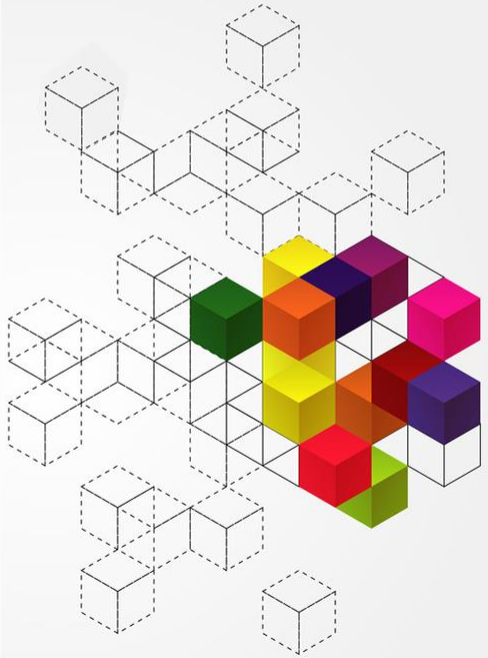




三、Slab算法

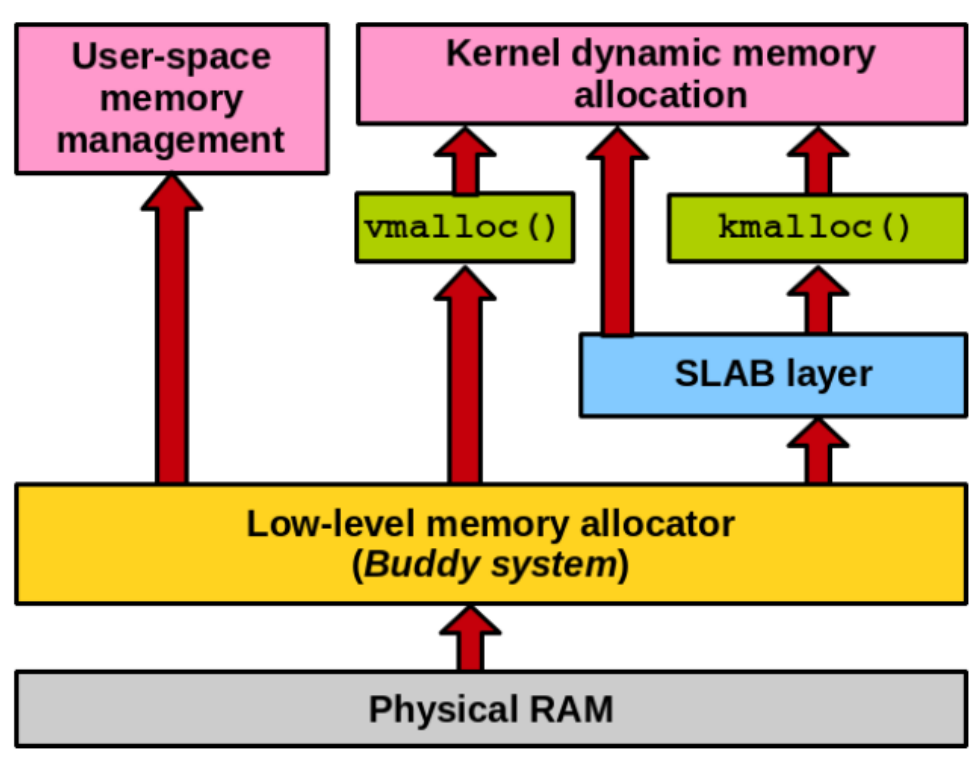
- Cache-slab-kernel object
  - Kernel object

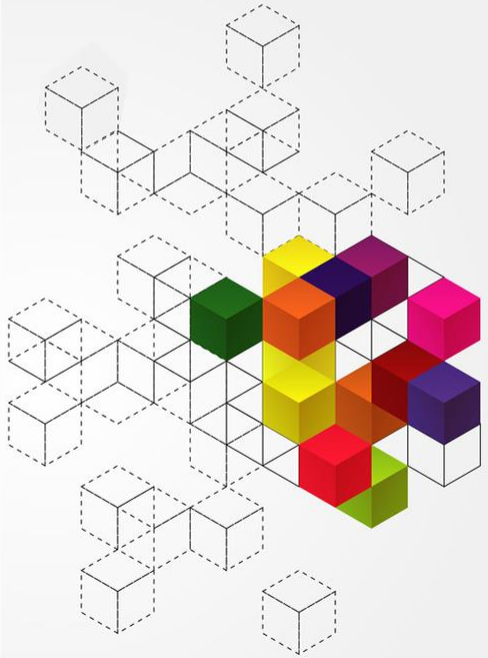




三、Slab算法

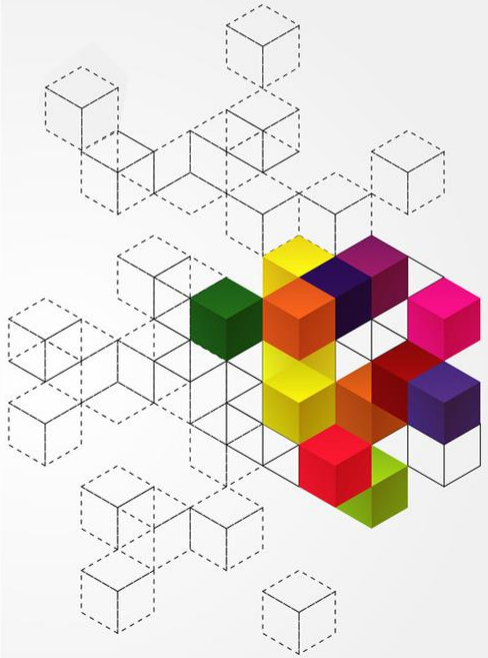
- Linux Kernel Memory Management



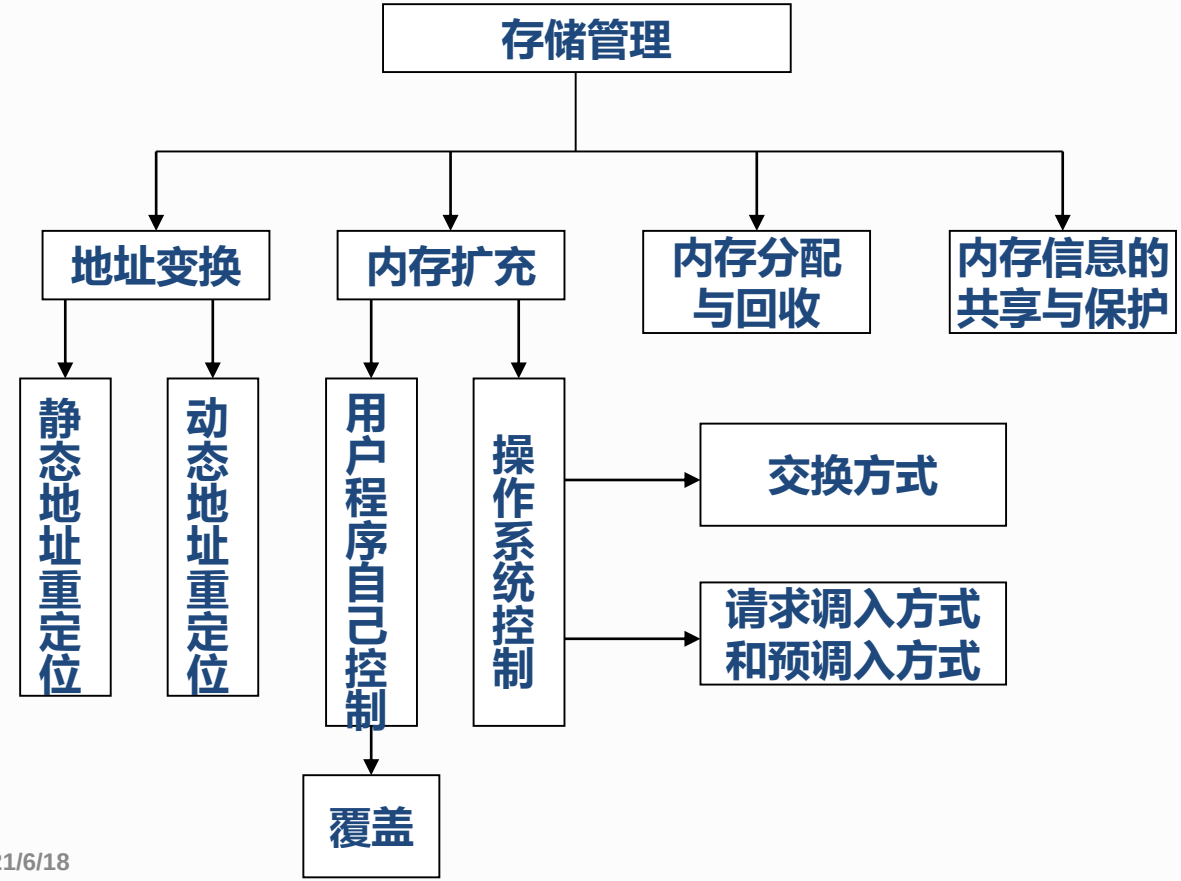


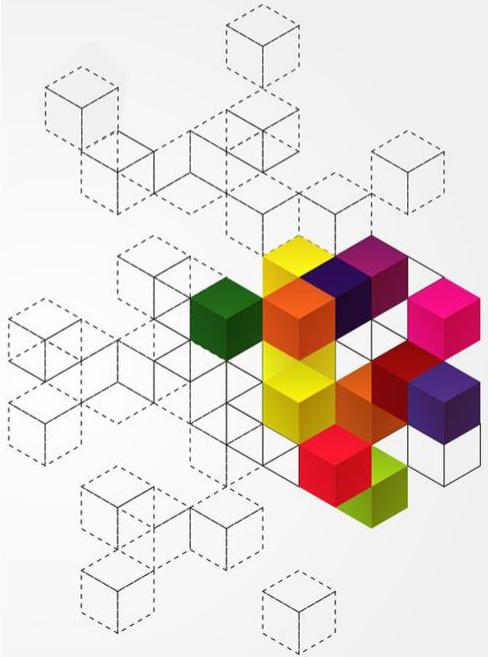
本讲小结

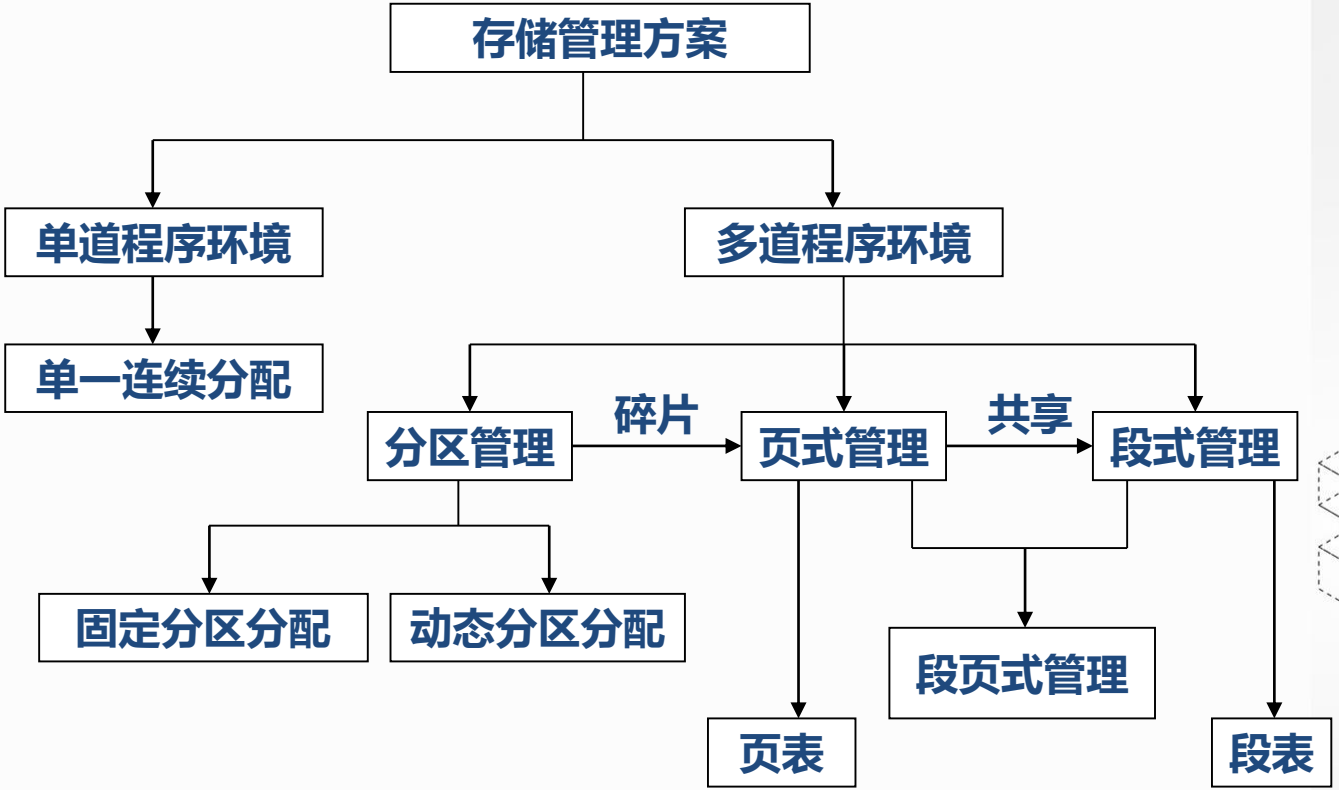
- 内核内存分配需求
- 伙伴算法
- Slab算法

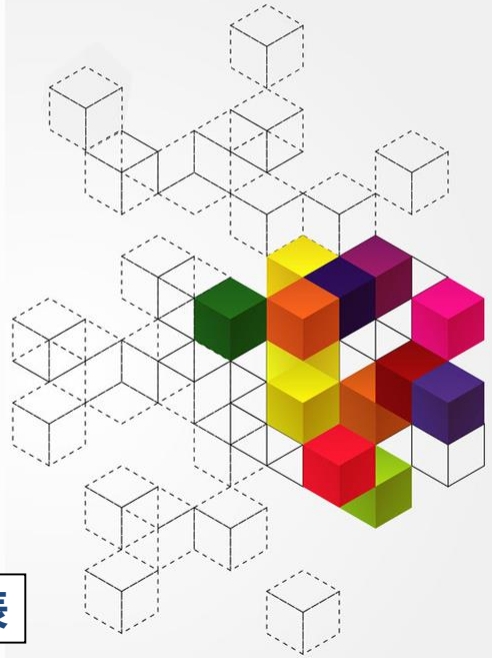


E、内存管理小结


存储管理
地址变换
内存扩充
内存分配与回收
内存信息的共享与保护
静态地址重定位
动态地址重定位
用户程序自己控制
操作系统控制
交换方式
请求调入方式和预调入方式
覆盖

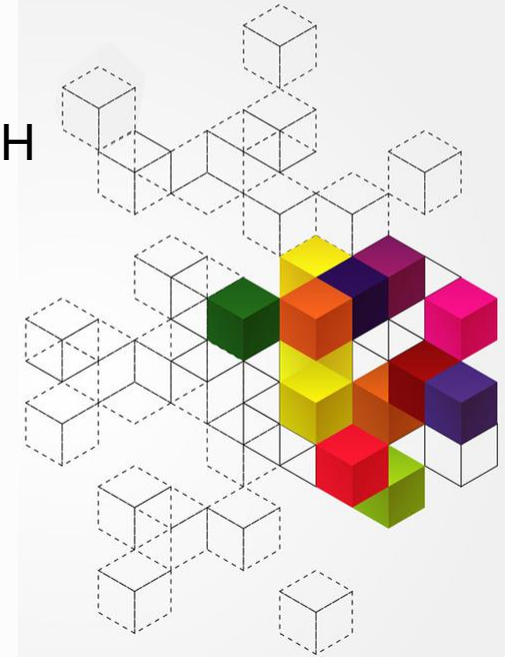



存储管理方案
单道程序环境
多道程序环境
单一连续分配
分区管理
碎片
页式管理
共享
段式管理
固定分区分配
动态分区分配
段页式管理
页表
段表



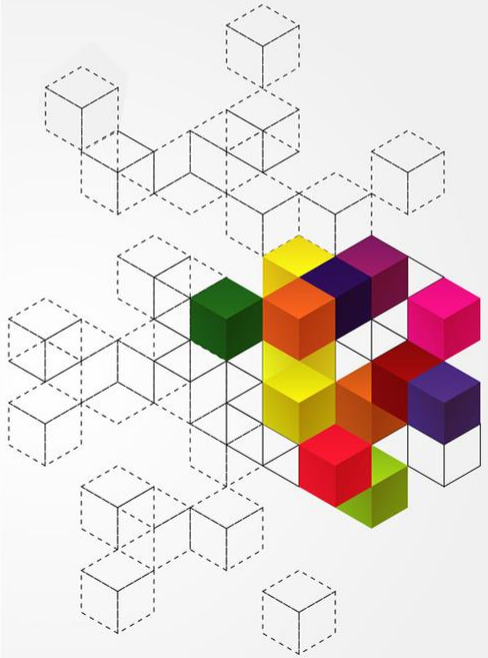
E1、练习

- 某请求页式存储管理中，允许用户编程空间为32个页面（每页2KB），主存为32KB，假定一个用户程序有5页长，且某时刻该用户页面映射如下表.
- （1）试述有效位、修改位的物理意义。
- （2）如果分别对以下3个虚地址：0AC5H、1AC5H、3AC5H处进行访问，试计算并说明存储管理系统将如何处理。

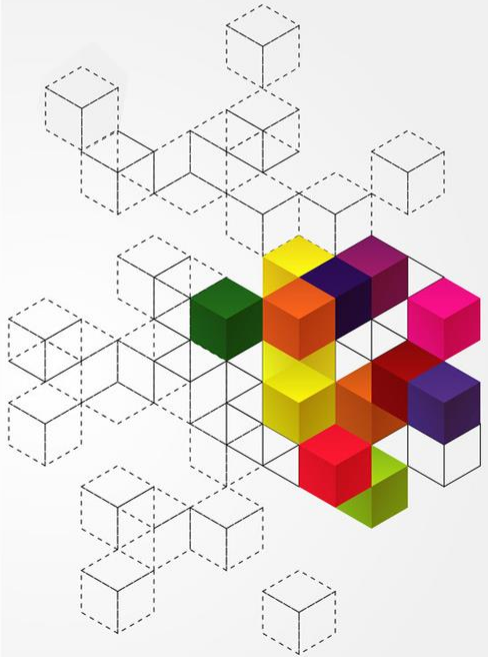


虚页号	有效位	访问位	修改位	页面号	外存始址
0	1	0	0	8	5000H
1	1	1	1	4	5800H
2	1	1	0	7	6000H
3	0	0	0	-	6800H
4	0	0	0	-	7000H

- 答：
- (1) 有效位表征该页是否在实际内存中，如果在，则状态位为1，反之为0；修改位表征该页被调入内存后是否被修改，如果修改，为1，则通过置换算法被换出时，需要写到外存中；反之，通过置换算法被换出时，不需要重新写到外存中。

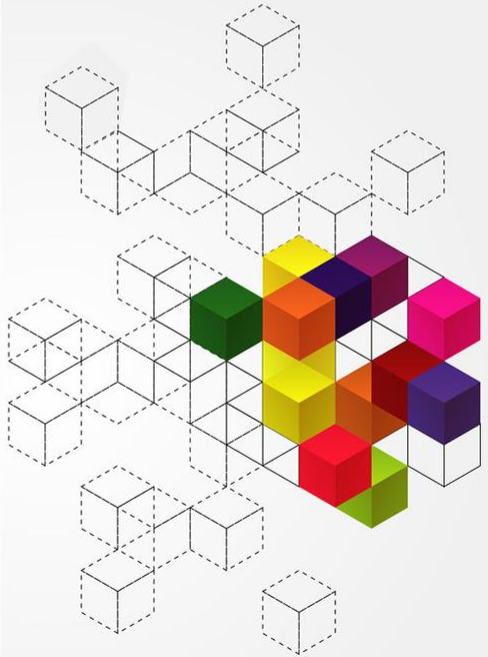


- 答:
- (2)页面大小为2KB，在虚地址中占有11个二进制位，用户地址空间有32页，虚页号占5位，因此虚地址长度为16位。又因为主存大小为32KB，因此实际物理地址为15位。
- 0AC5H的二进制形式为 0000 1010 1100 0101 ，其中虚页号为00001，后面部分为页内偏移，00001对应十进制的1，由上表可知对应实际页面号为4，化为2进制为 0100，所以应访问的实际物理地址是 0100 010 1100 0101，对应于16进制是 22C5H。
- 1AC5H的二进制形式为 0001 1010 1100 0101 ，其中虚页号为00011，后面部分为页内偏移，00011对应十进制的3，由上表可知虚页号为3的页面没有在内存中，因此发生缺页异常，系统从外存中把第6页调入内存，然后更新页表。
- 3AC5H的二进制形式为 0011 1010 1100 0101 ，其中虚页号为00111，后面部分为页内偏移，00111对应十进制的7，超过了进程的地址空间长度，系统发生地址越界中断，程序运行终止。



什么是虚拟存储器?其特点是什么?

- 由进程中的目标代码、数据等的逻辑地址组成的虚拟空间称为虚拟存储器。虚拟存储器不考虑 物理存储器的大小和信息存放的实际位置，只规定每个进程中相互关联信息的相对位置。每个进程都拥有自己的虚拟存储器，且虚拟存储器的容量是由计算机的地址结构和寻址方式来确定。
- 实现虚拟存储器要求有相应的地址转换机构，以便把指令的虚拟地址变换为实际物理地址；另 外，由于内存空间较小，进程只有部分内容存放于内存中，待执行时根据需要再调指令入内存。



实现地址重定位的方法有哪几类?

- 实现地址重定位的方法有两种：静态地址重定位和动态地址重定位。
- (1)静态地址重定位是在虚空间程序执行之前由装配程序完成地址映射工作。静态重定位的优点是不需要硬件支持，但是用静态地址重定位方法，程序经地址重定位后就不能移动了，因而不能重新分配内存，不利于内存的有效利用。静 态重定位的另一个缺点是必须占用连续的内存空间和难以做到程序和数据的共享。
- (2)动态地址重定位是在程序执行过程中，在CPU访问内存之前由硬件地址变换机构将要访问的 程序或数据地址转换成内存地址。动态地址重定位的主要优点有：①可以对内存进行非连续分配。②动态重定位提供了实现虚拟存储器的基础。③动态重定位有利于程序段的共享。