操作系统

L16 虚存（下）

胡燕
大连理工大学 软件学院

上讲简要回顾

虚存基本概念

虚存的理论支撑：局部性原理

虚存的目标：用小的物理内存支撑大的虚存空间 => 赋予程序员内存自由

虚存的实现机制：按需调页、缺页异常（或称页故障、缺页中断）、页置换

代表性页置换算法

FIFO	易于实现，性能较差，Belady异常
OPT	能实现页故障率最小化，但是过于理想化，无法落地实现
LRU	从理想回归现实

页置换算法实现

Implementation of Page Replacement Algorithms

01

算法的现实考虑

- LRU算法要落地，需要解决实际问题
  - 既要志存高远，又要脚踏实地
  - 既要明确方向，又要具体问题具体分析，给出能够实际落地应用的方案

需要做些近似与变通

LRU思想 → LRU具体实现

LRU精确实现：问题分析

精确实现 → 就得付出额外时间的代价

LRU精确实现选择1：Counters

associate with each page-table entry a time-of-use field and add to the CPU a logical clock or counter.

The clock is incremented for every memory reference.

Whenever a reference to a page is made, the contents of the clock register are copied to the time-of-use field in the page-table entry for that page.

We replace the page with the smallest time value（每次置换具有最小时间戳的页）. This scheme requires a search of the page table to find the LRU page and a write to memory

代价很大：维持引用的时间戳，以及搜索具有最小时间戳的置换页面

LRU精确实现：问题分析

精确实现 → 需要付出额外时间的代价

LRU精确实现选择2：维持一个先进先出队列（双向链表）

Whenever a page is referenced, it is removed from the stack and put on the top.

The least recently used page is kept at the tail of the list.

Each update, the worst case would require changing 6 pointers at least.

基于引用位的页置换算法

通过硬件为每个页面维护一个引用位（Reference bit）：

- 每次访问某个页面，将其关联的引用位置1
- 根据页置换的算法，在合理的时间将页面引用位归0

页置换：

- 优先置换引用位为0的页面

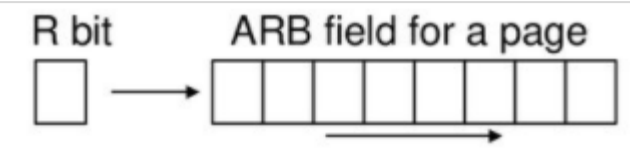
LRU近似算法2

附加引用位算法

基本配置：
- 页面引用位R
- 每个页面的固定数量的附加引用位ARB (Additional Reference Bit)

基本操作(通过定时器设定采样周期)：
- 每次定时器中断，ARB右移1位
- 将R移入ARB高位

附加引用位数=8的情形：

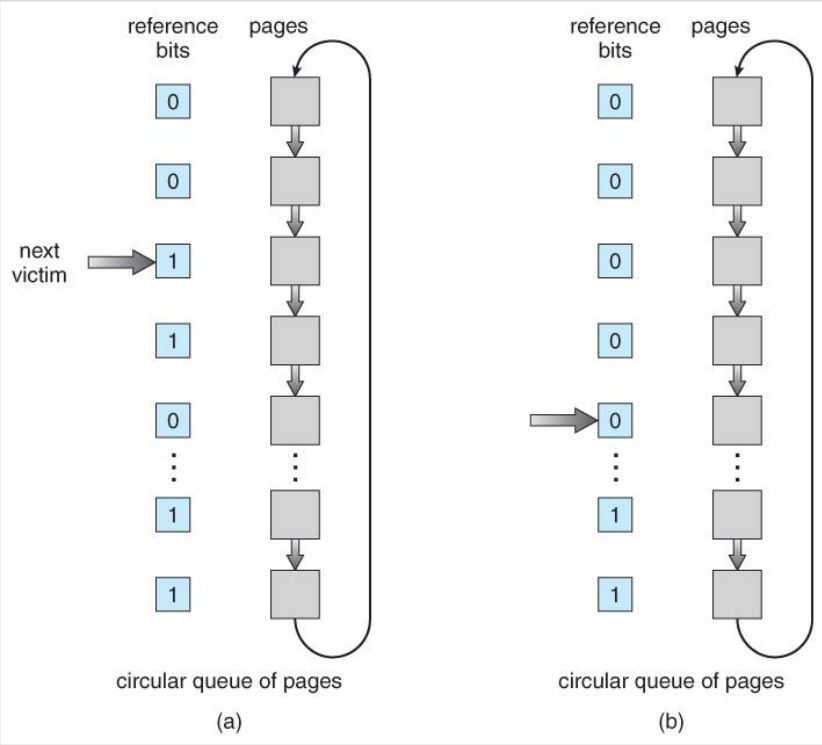


Q: 如何选择优先被淘汰的页帧?

选择ARB值最小的页进行淘汰

LRU近似算法2

二次机会算法（Second chance algorithm/CLOCK算法）



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的页面，引用位改为0

引用位为1的页面，会赢得驻留内存的第2次机会

填空题 10分

设置

在一个请求分页系统中，如果一个进程的页面访问串是2,3,2,1,5,2,4,5,3,2,5,2。目前没有任何页面装入内存，当分配给该作业的物理页帧数为3时，请计算采用CLOCK置换算法（二次机会置换算法），访问过程中所发生的缺页次数 =（ [填空1] ）。

作答

LRU近似算法2

二次机会算法

练习：

在一个请求分页系统中，如果一个进程的页面访问串是2,3,2,1,5,2,4,5,3,2,5,2。目前没有任何页面装入内存，当分配给该作业的物理页帧数为3时，请计算采用CLOCK置换算法（二次机会置换算法），访问过程中所发生的缺页次数 = （ ）。

需要页面2，内存中还有空闲位置，直接加入页面2,得到2(0)

需要页面3，内存中还有空闲位置，直接加入页面3,得到2(0),3(0)

需要页面2，命中，内存中已经存在页面2，得到2(1),3(0)

需要页面1，直接加入页面1，得到2(1),3(0),1(0)

需要页面5，扫描将页面2的引用位归0，然后替换掉页面3，得到2(0),5(0),1(0)

访问页面2，命中，将页面2的引用位置1，得到2(1),5(0),1(0)

访问页面4，由于当前指向位置为页面1，且其引用位为0，所以替换页面1，得到2(1),5(0),4(0)

需要页面5，内存中存在页面5，不改变，得到2(1),5(1),4(0)

访问页面3，从页面2开始扫描，将2,5的使用位都置为0，然后替换到页面4，得到2(0),5(0),3(0)

访问页面2，命中，将2的使用位置为1，得到2(1),5(0),3(0)

访问页面5，命中，页面5的引用变为1，得到2(1),5(1),3(0)

访问页面2，命中，得到2(1),5(1),3(0)

LRU近似算法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

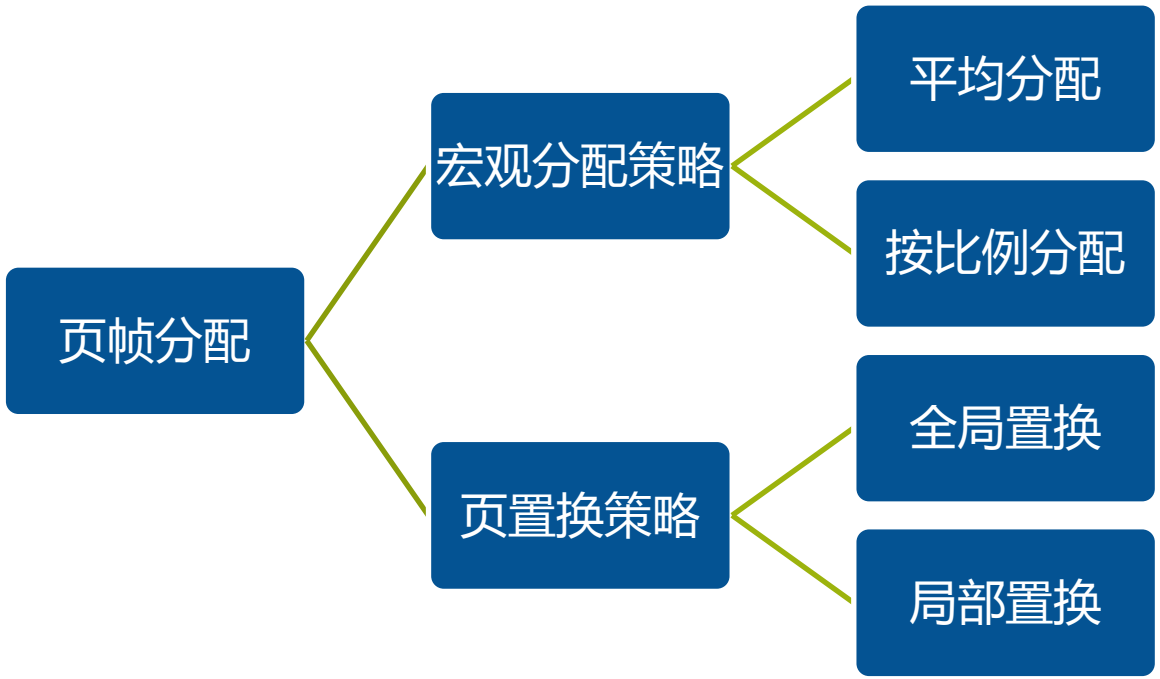
页帧分配

Page Frame Allocation

02

页帧分配宏观策略

目的：为每个进程进行合理的物理内存分配



每个用户进程被分配相同数量的页帧

按照进程的大小，成比例分配物理页

从系统全体物理页中选择牺牲页帧

从已分配给当前进程的物理页中选择牺牲页帧

页帧分配问题

Q1:如果为进程的物理内存分配做得不够好，可能会发生什么?

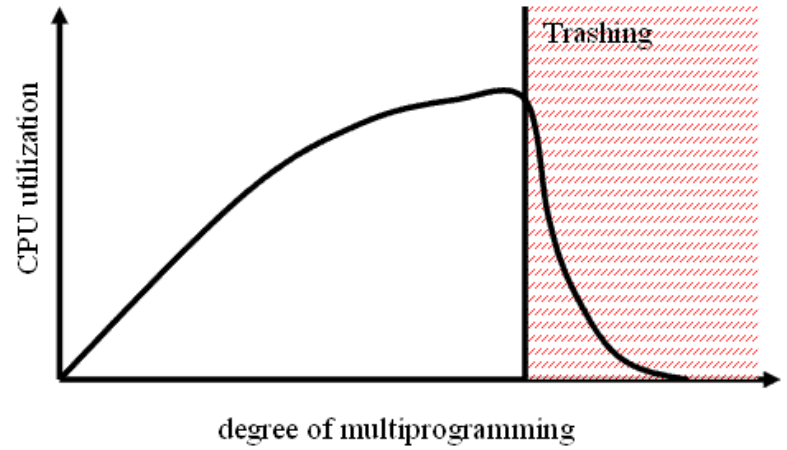
内存抖动（Thrashing）

进程执行过程中，页面被频繁地换入换出，导致进程执行的大部分时间都在进行换入换出的I/O

Q2:导致内存抖动的主要原因是什么?

引起抖动的主要原因:

1. 页置换算法选择不当
2. 系统内有过多进程，太高的多道程序度



拆东墙，补西墙

工作集模型

如何为进程合理分配内存

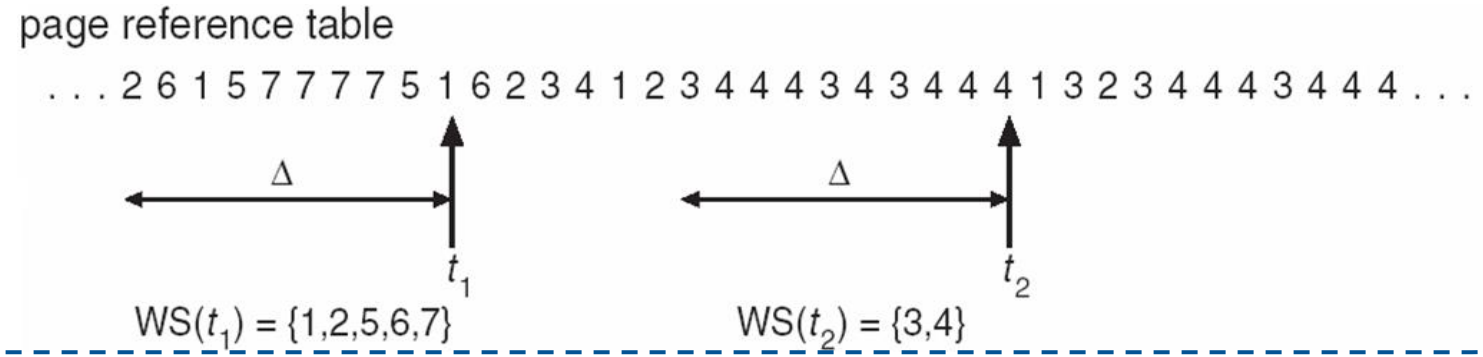
- 如果分配过大，会存在对物理内存的浪费
- 如果分配过小，内存抖动

解决问题的关键：准确刻画程序执行的局部性

工作集模型：给出了进行程序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

关键参数：工作集窗口大小



工作集模型实现

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

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

基于工作集的内存分配应保持 m > D 这一目标

Q:能不能将精确实现的工作集模型实际用于操作系统中的物理内存分配过程?

不可。
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

工作集模型的近似实现

- 实现策略：降低采样频率
- 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
定时器中断发生时，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

页帧分配原则

实际操作系统的内存分配机制

Linux

- Demand paging + global page replacement
- Use page replacement algorithm similar to the LRU-approximation clock algorithm.
- lists for relatively active and relatively inactive pages.

Windows

- demand paging, copy-on-write, paging, and memory compression.
- Using working-set based page frame allocation strategy.

内核内存分配

Kernel Memory Allocation

03

内核内存分配简介

内核内存分配，用以满足操作系统内核模块的内存需求

inode	task_struct
128字节（ext2,ext3） 160字节（ext4）	约1.7KB[1]

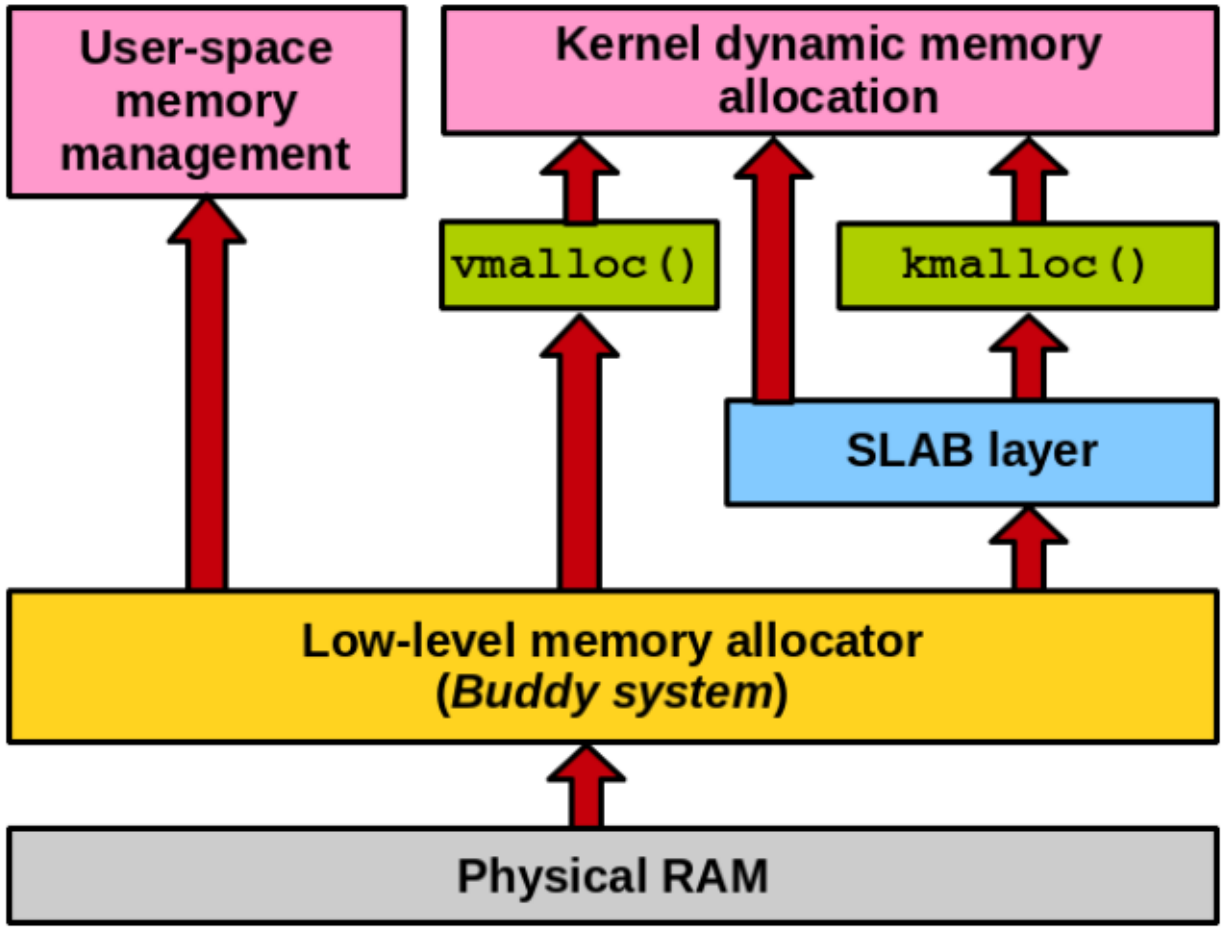
内核代码中执行的内存申请通常小于1页

为此，内核内存分配器应具备高效分配大内存和小内存的能力。

[1]http://books.gigatux.nl/mirror/kerneldevelopment/0672327201/ch03lev1sec1.html

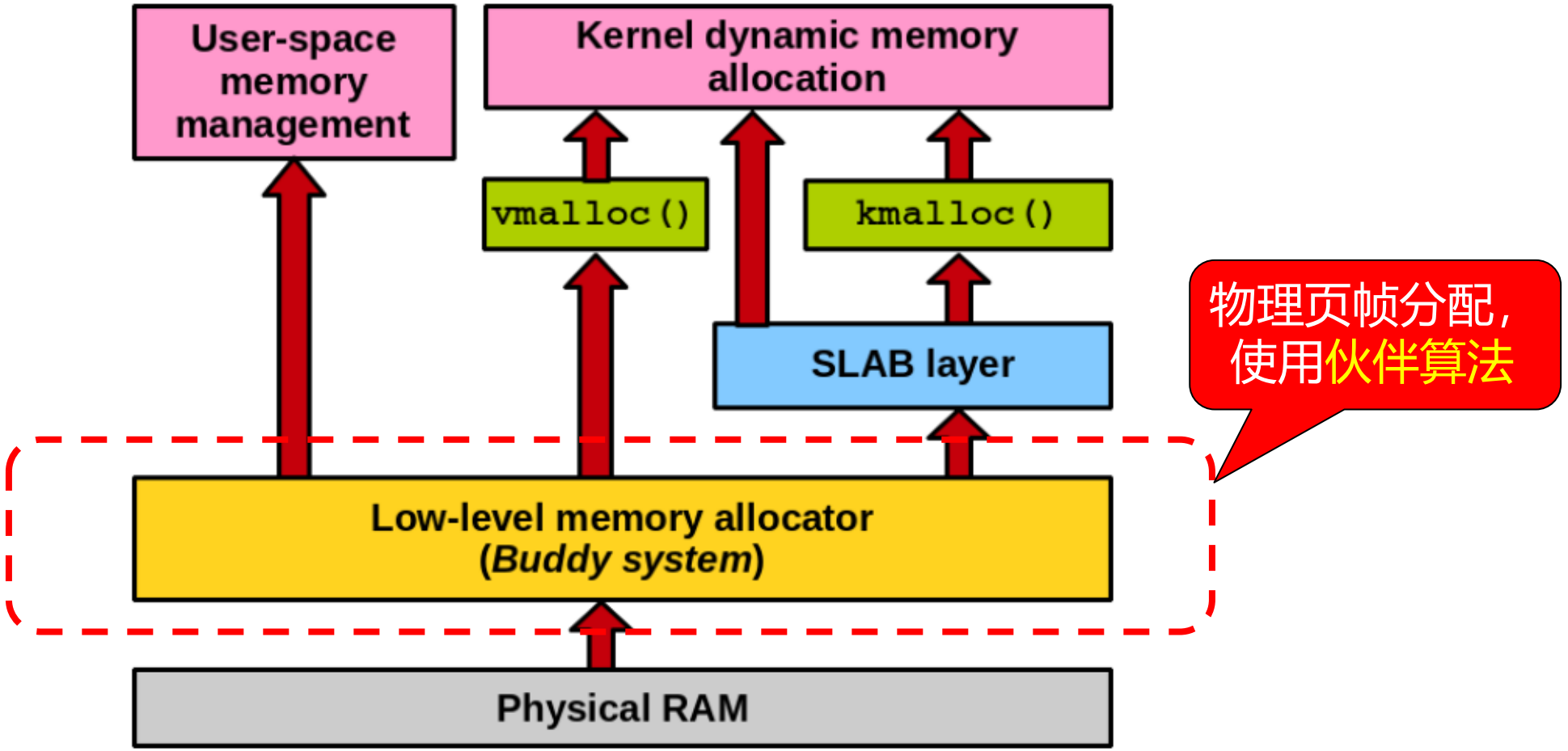
内核内存分配简介

以Linux系统为主，介绍其内存体系中的内核内存分配机制



伙伴算法

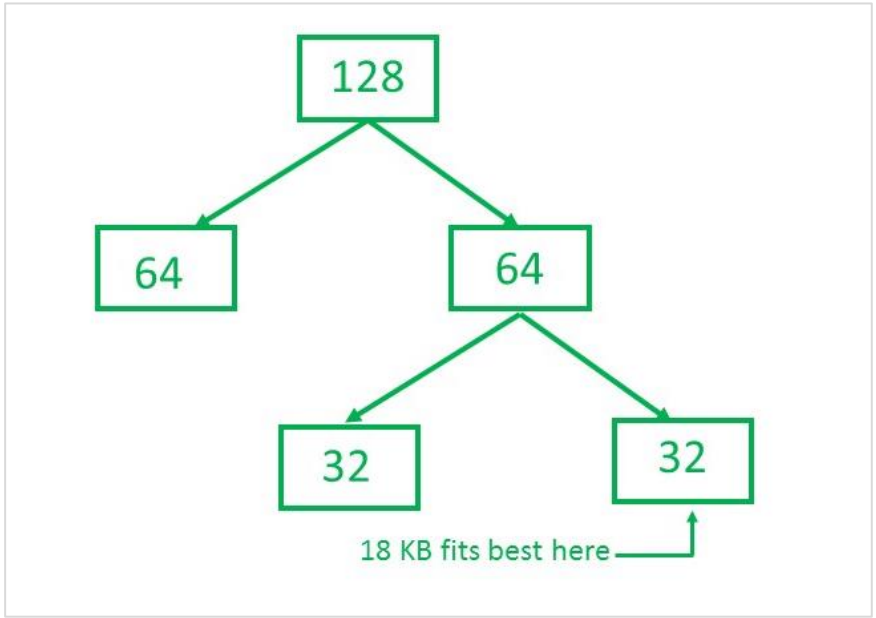
以Linux系统为主，介绍其内存体系中的内核内存分配机制



伙伴算法简介

伙伴算法是一种动态存储器管理算法。该算法通过不断地平分较大的空闲内存块来获得较小的空闲内存块，直到获得所需要的内存块，当内存释放时，该算法尽可能地合并空闲块。

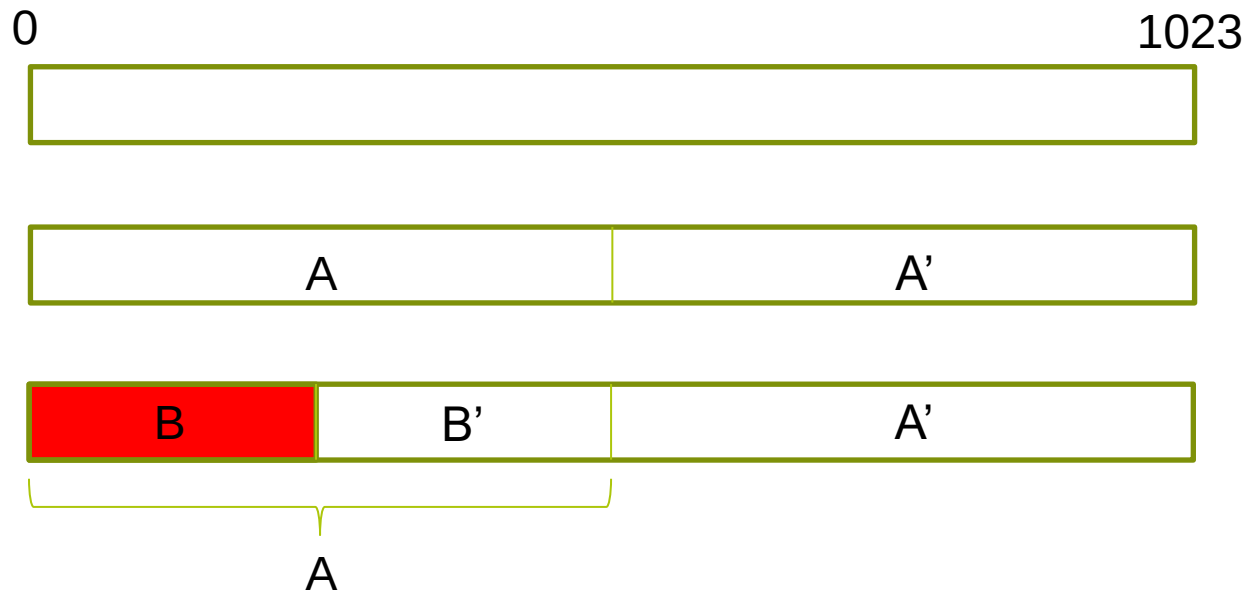
其中，在分配和合并内存块时都是以2的次幂为单位，即1,2,4,8,16,32,64,128等。所谓“伙伴”，就是指在空闲块被分裂时，由同一个大块内存分裂出来的两个小块内存就互称“伙伴”。



Linux利用buddy算法思想来进行物理页帧的分配

伙伴算法：分配

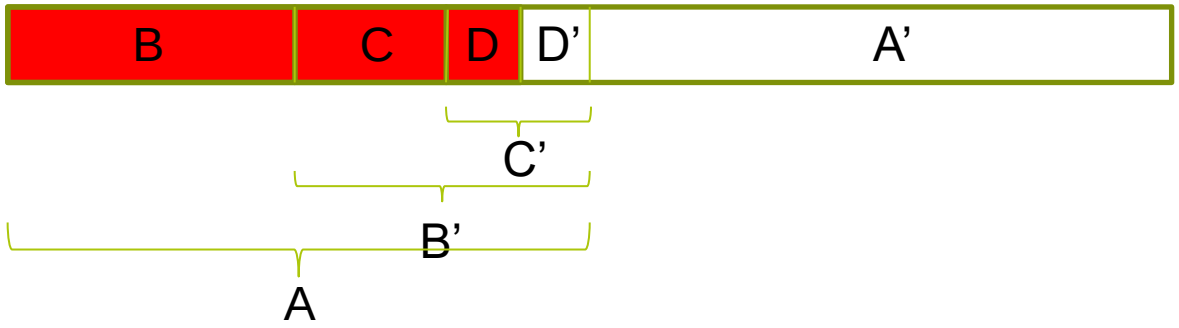
Allocate(256)



伙伴算法：分配

Allocate(256)

→ Allocate(128) → Allocate(64)

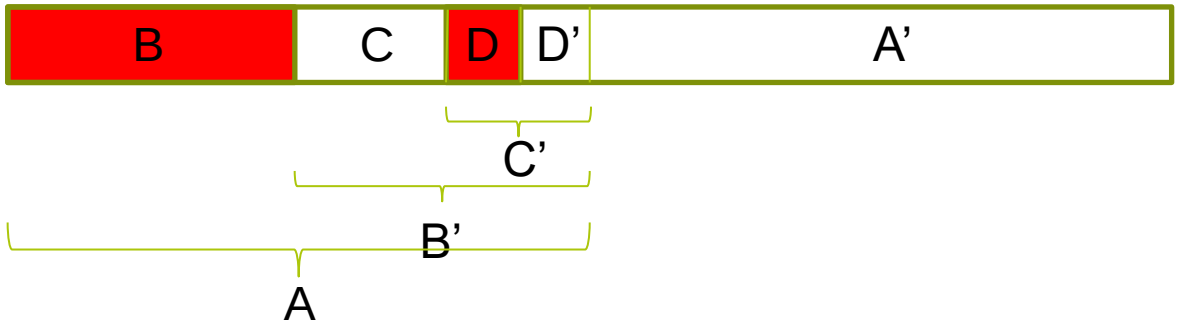


伙伴算法：释放

Allocate(256)

→ Allocate(128) → Allocate(64)

→ Release(C,128)



伙伴算法

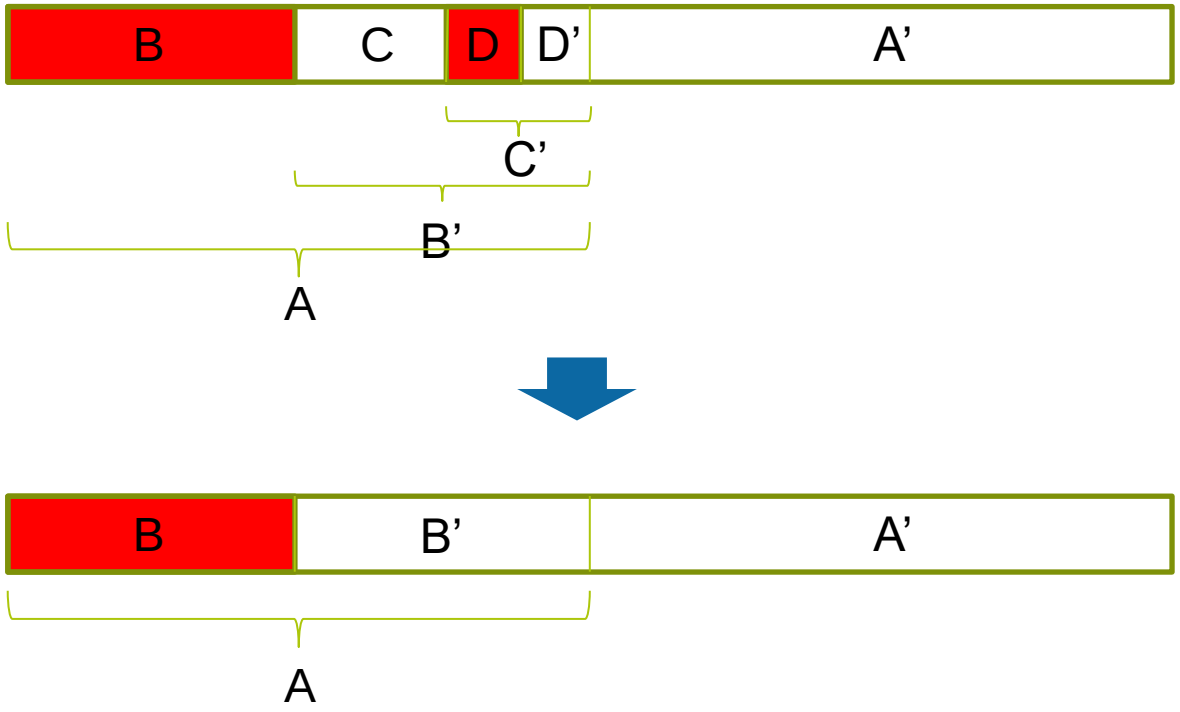
伙伴算法：释放

Allocate(256)

→ Allocate(128) → Allocate(64)

→ Release(C,128)

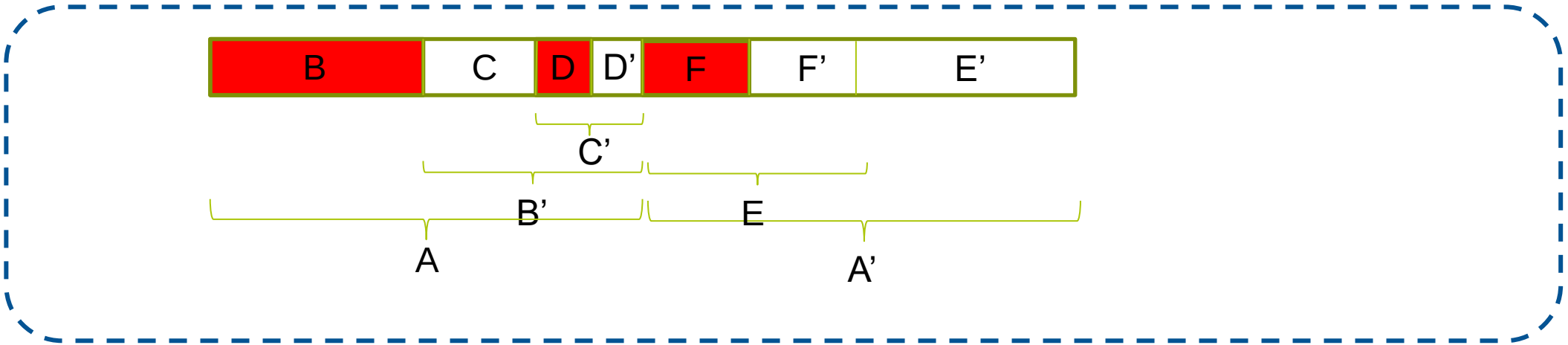
→ release(D,64)



伙伴算法：伙伴的判定

“伙伴”应当满足以下三个条件：

- 两个块大小相同
- 两个块地址连续
- 两个块必须是同一个大块中分离出来的



Linux中buddy system实现

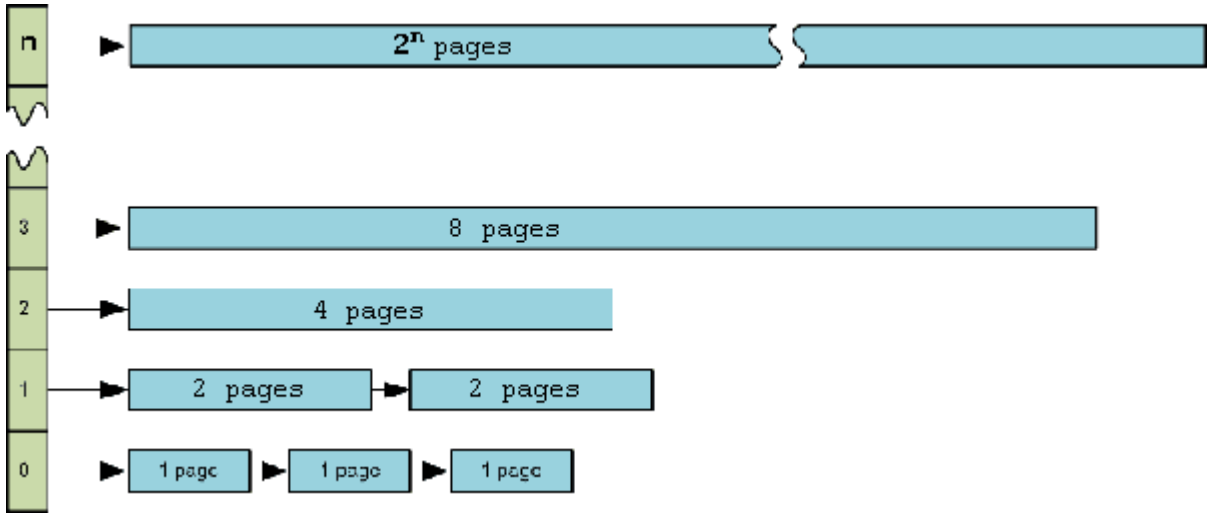
Linux defines at compile time 11 orders (see "include/linux/mmzone.h") and the ***default page size is*** *4 KiB.*

This means that the size of blocks of order 11 (the biggest ones) is 4 KiB * 2^{10} = 4 MiB.

in a system equipped with with 2 GiB of RAM

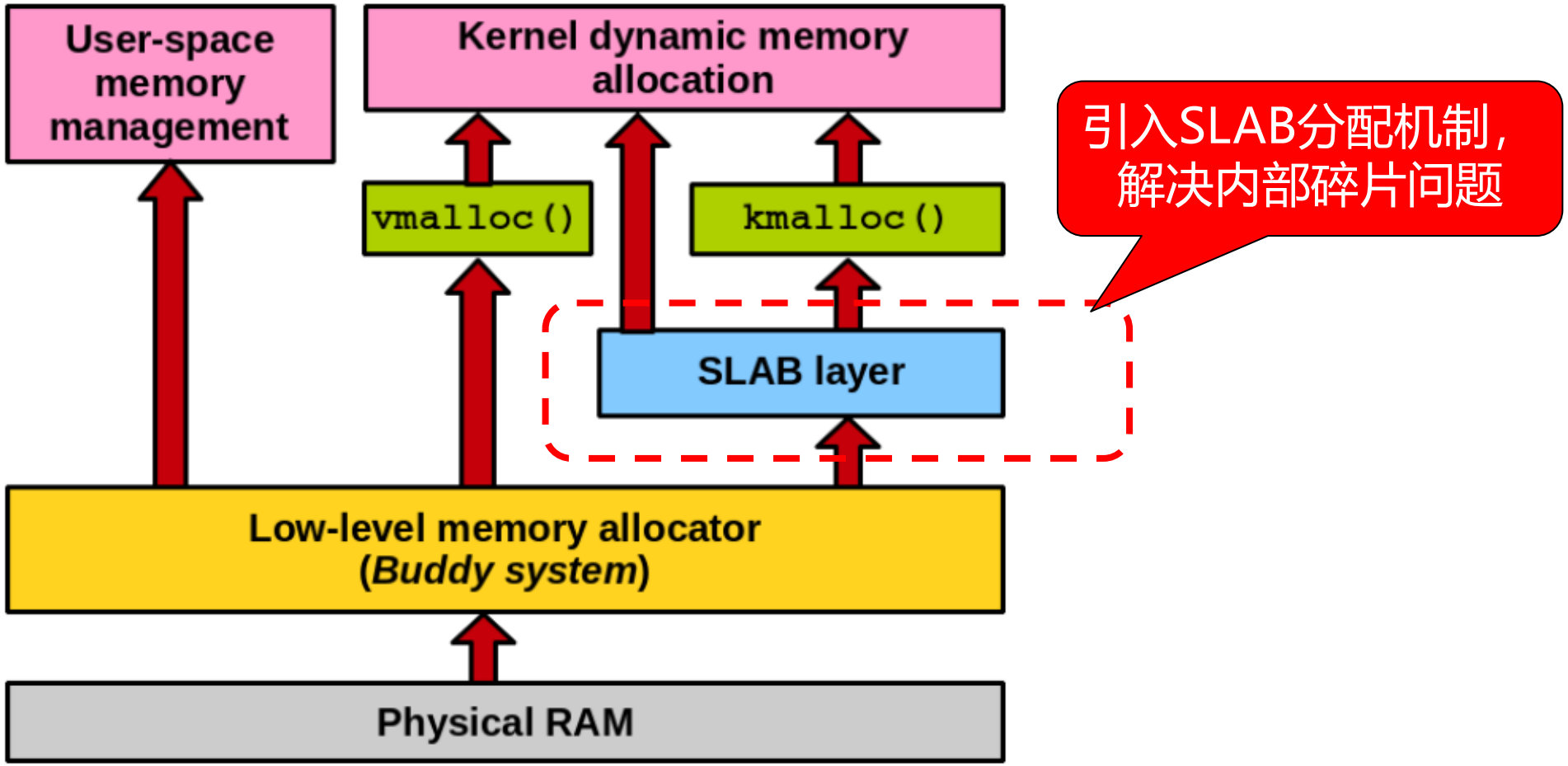
at boot has a little few than 512 order 11 companion buddies:

2GB/4MB = 512



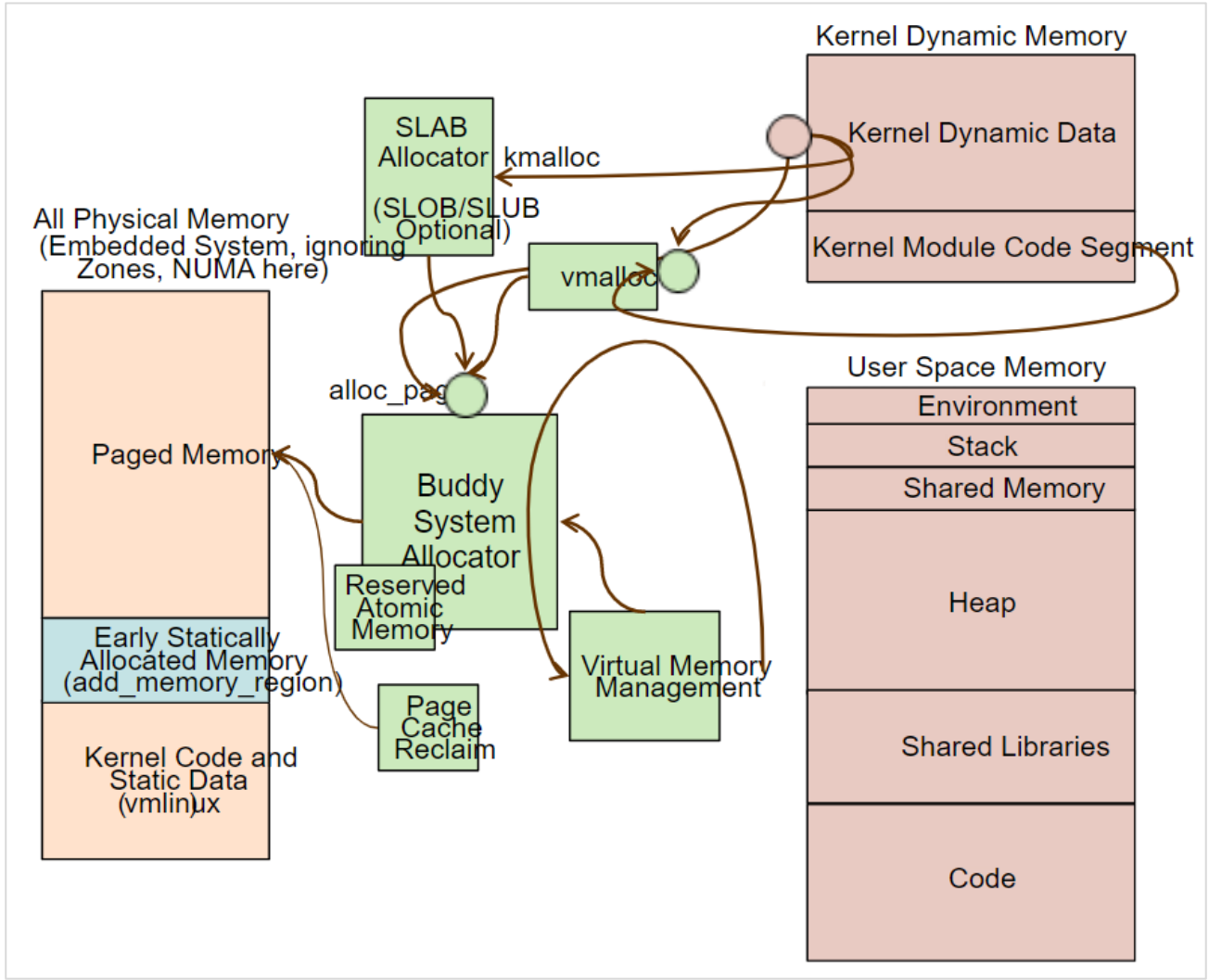
SLAB分配

以Linux系统为主，介绍其内存体系中的内核内存分配机制



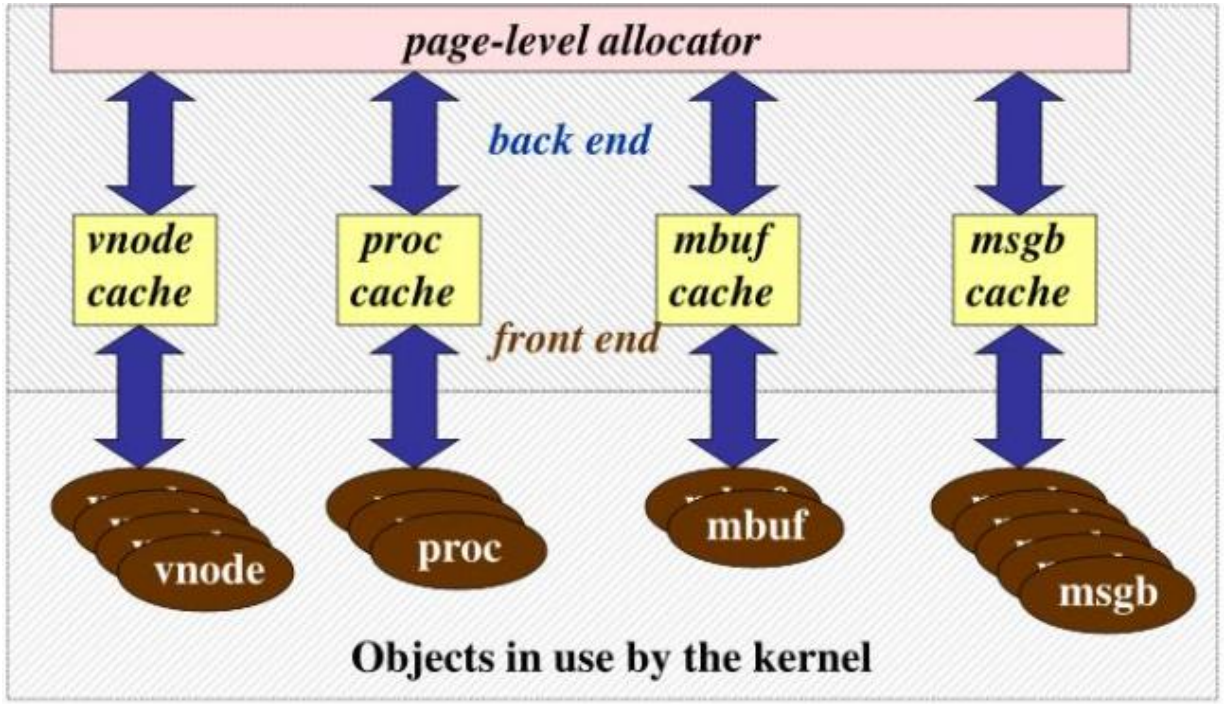
SLAB分配

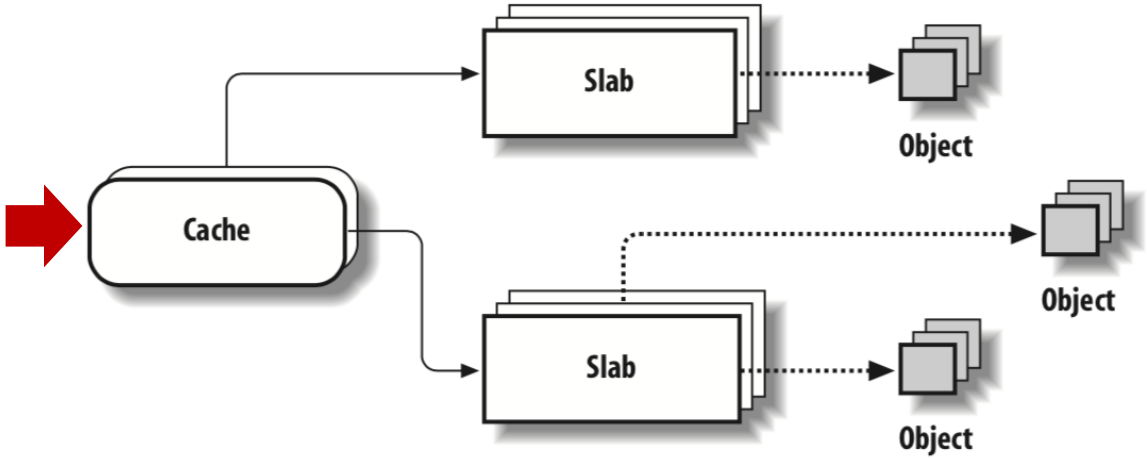
Linux内存管理总体图



SLAB分配

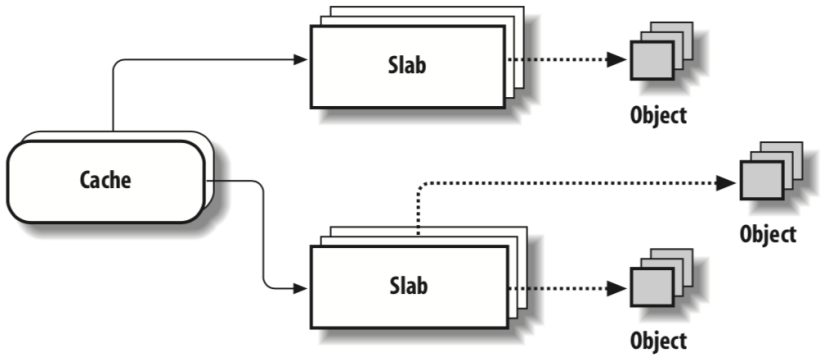
SLAB分配方法示意图

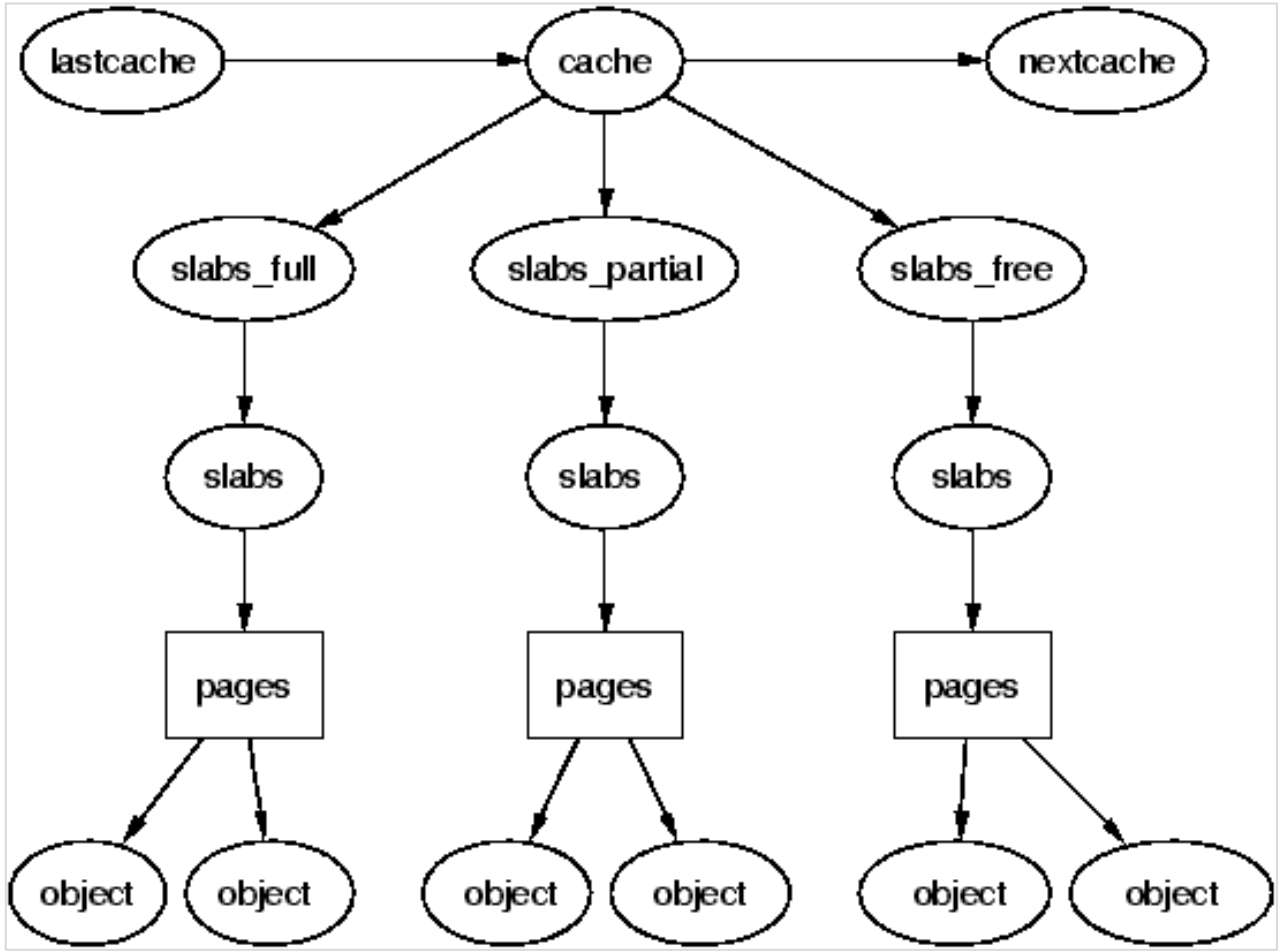




SLAB分配

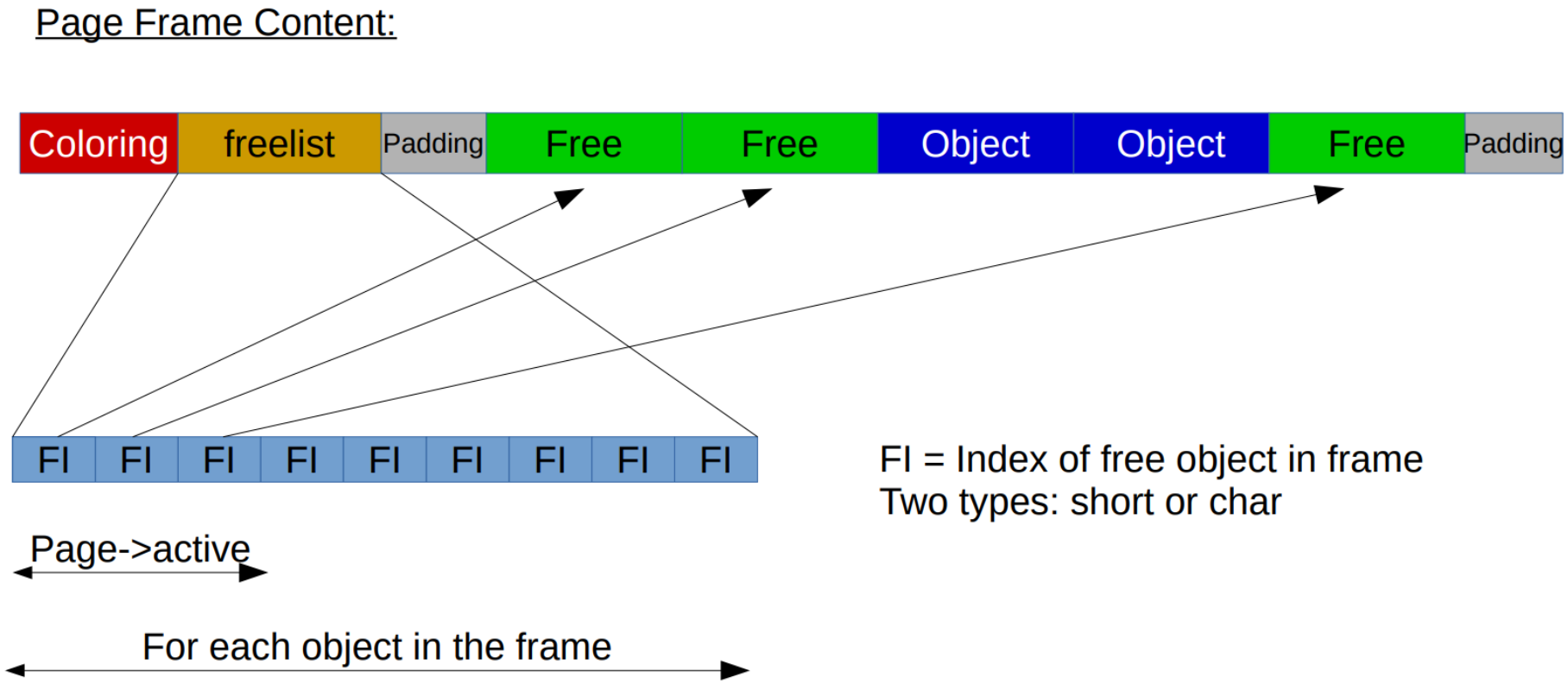
SLAB分配：Cache组织结构



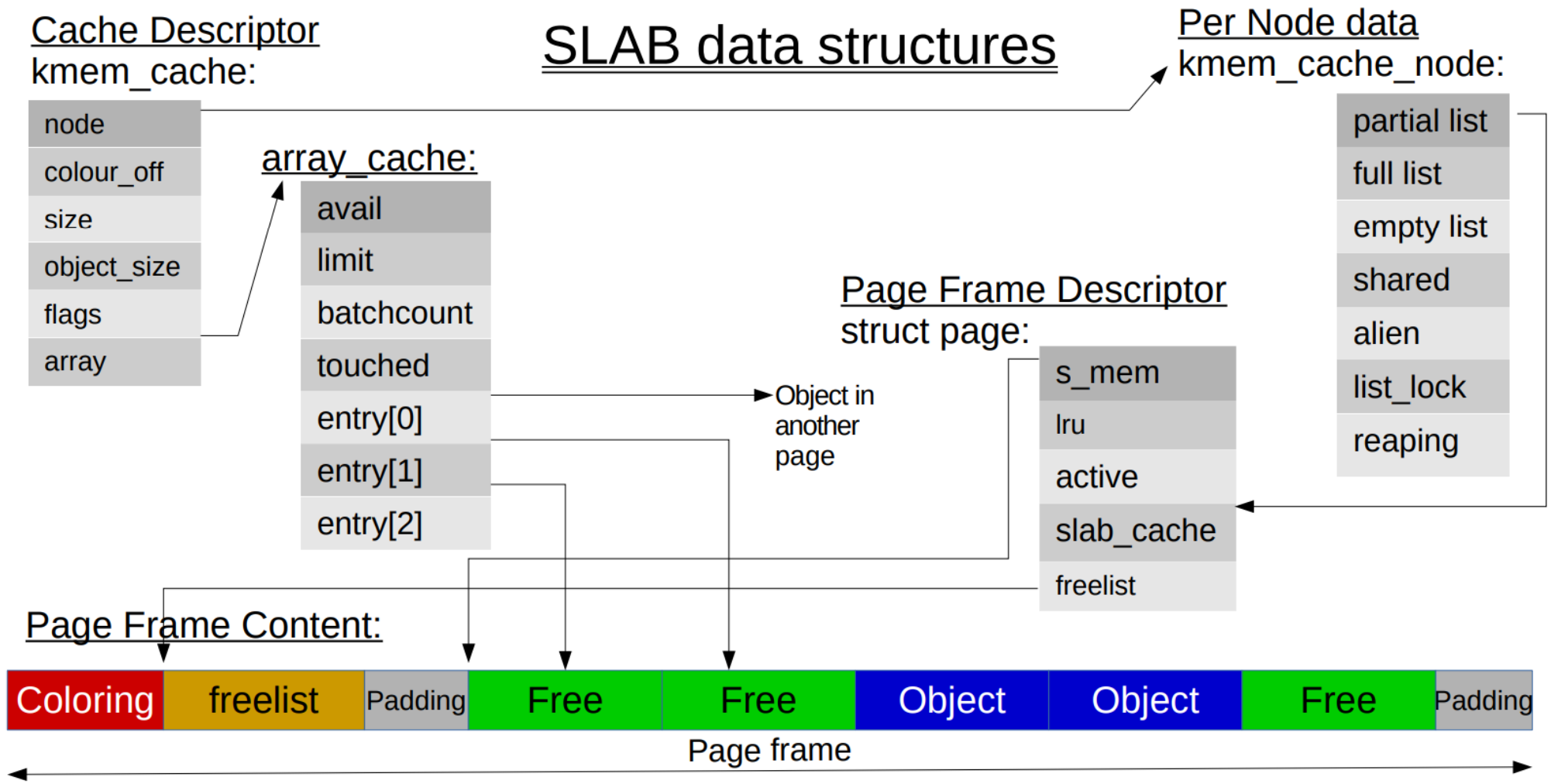


SLAB分配

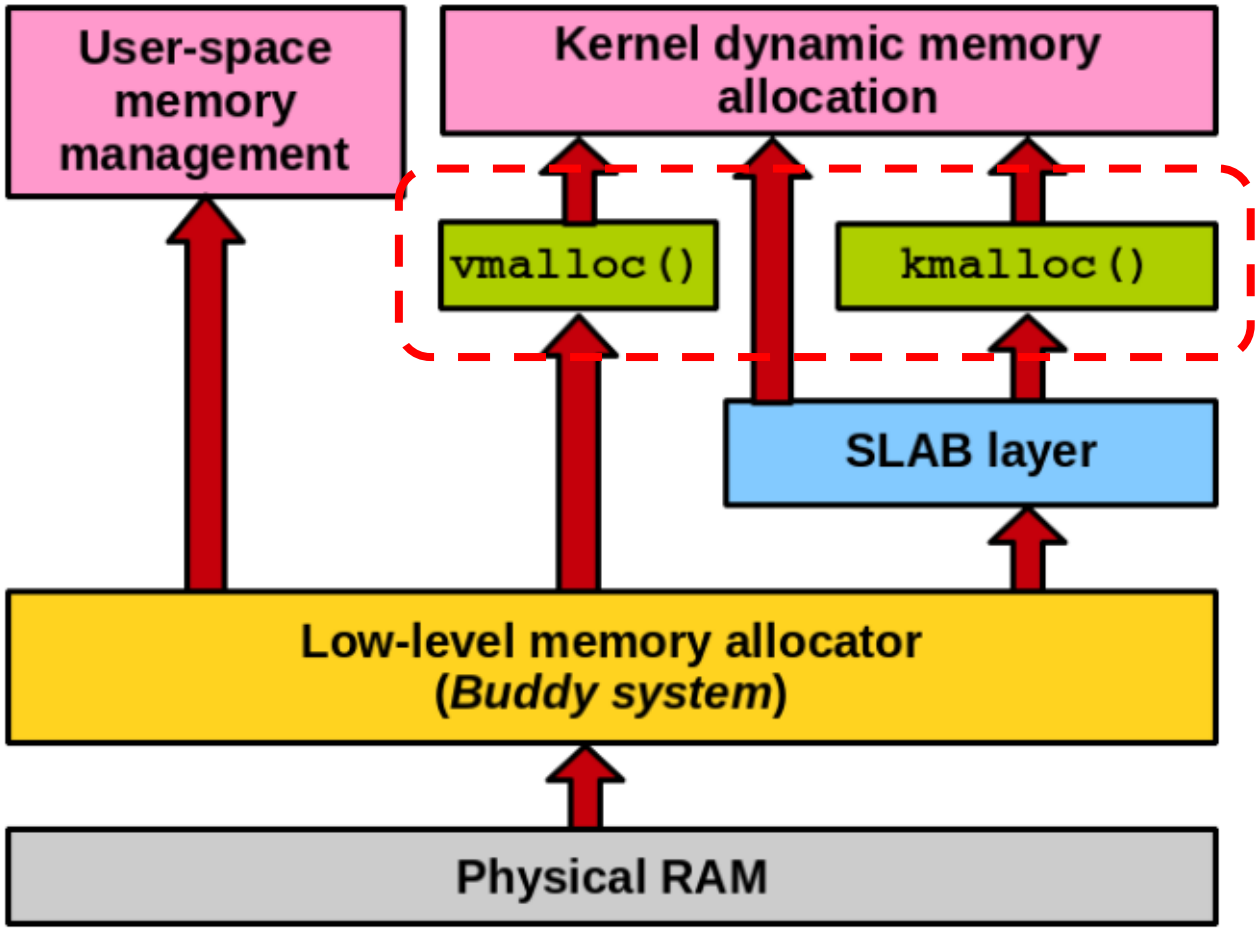
SLAB分配：每个页框中的对象管理结构



SLAB分配：整体数据结构



内核内存分配



Object level management

- Finer granularity, higher performance overhead
- By default, prefer the `kmalloc()` version over the `vmalloc()` version, for performance

Slab allocation

- Optimizes performance if allocating and de-allocating large numbers of the same kind of object

Summary

小结：

- LRU算法的近似实现
- 物理页帧分配
- 内核内存分配

为何需要近似算法　　附加引用位算法　　二次机会算法

页帧分配策略　　工作集机制

内核内存分配需求　　Linux内存分配算法层次　　Buddy system

练习

习题1

内存保护需要(　)，以保证整个内存空间不被非法访问。

A. 由操作系统和内存硬件协同完成

B. 由操作系统或内存硬件独立完成

C. 由内存硬件独立完成

D. 由操作系统独立完成

练习

习题1

内存保护需要(　)，以保证整个内存空间不被非法访问。

A. 由操作系统和内存硬件协同完成

B. 由操作系统或内存硬件独立完成

C. 由内存硬件独立完成

D. 由操作系统独立完成

练习

习题2

在可变分区内存管理方案中，某一进程执行结束后，系统回收其主存空间并与相邻空闲分区合并，为此需修改空闲分区表，造成空闲分区数减1的情况是该回收分区（ ）。

A. 前后均无邻接空闲分区

B. 前后均有邻接空闲分区

C. 前有邻接空闲分区，但后无邻接空闲分区

D. 前无邻接空闲分区，但后有邻接空闲分区

练习

习题2

在可变分区内存管理方案中，某一进程执行结束后，系统回收其主存空间并与相邻空闲分区合并，为此需修改空闲分区表，造成空闲分区数减1的情况是该回收分区（ ）。

A. 前后均无邻接空闲分区

B. 前后均有邻接空闲分区

C. 前有邻接空闲分区，但后无邻接空闲分区

D. 前无邻接空闲分区，但后有邻接空闲分区

练习

习题3

动态重定位是在程序的（ ）过程中进行的。

A. 链接

B. 装入

C. 执行

D. 编译

练习

习题3

动态重定位是在程序的（ ）过程中进行的。

A. 链接

B. 装入

C. 执行

D. 编译

练习

习题4

下面关于内存管理的叙述，正确的是（ ）。

A. 存储保护的目的是限制内存的分配

B. 在内存大小为M、有N个用户的分时系统中，每个用户占用M/N大小的内存空间

C. 在虚拟内存系统中，只要磁盘空间无限大，进程就能拥有任意大的编址空间

D. 实现虚拟内存管理必须要有相应的硬件支持

练习

习题4

下面关于内存管理的叙述，正确的是（ ）。

A. 存储保护的目的是限制内存的分配

B. 在内存大小为M、有N个用户的分时系统中，每个用户占用M/N大小的内存空间

C. 在虚拟内存系统中，只要磁盘空间无限大，进程就能拥有任意大的编址空间

D. 实现虚拟内存管理必须要有相应的硬件支持

练习

习题5

下面的内存管理方案中，（ ）内存管理方式适宜采用静态重定位。

A. 固定分区

B. 分页

C. 分段

D. 动态重定位分区

练习

习题5

下面的内存管理方案中，（ ）内存管理方式适宜采用静态重定位。

A. 固定分区

B. 分页

C. 分段

D. 动态重定位分区

练习

习题6

连续分区分配和分页管理的主要区别是（ ）。

A. 连续分区分配中的块比分页管理中的页要小

B. 连续分区管理有地址映射而分区管理没有

C. 连续分区管理有存储保护而分区管理没有

D. 连续分区管理要求分配连续的空间，而分页管理没有这种要求

练习

习题6

连续分区分配和分页管理的主要区别是（ ）。

A. 连续分区分配中的块比分页管理中的页要小

B. 连续分区管理有地址映射而分区管理没有

C. 连续分区管理有存储保护而分区管理没有

D. 连续分区管理要求分配连续的空间，而分页管理没有这种要求

练习

习题7

采用两级页表的页式存储管理中，进程执行中CPU首次访问某逻辑地址时，需访问主存的次数是（ ）。

A. 1次

B. 2次

C. 3次

D. 4次

练习

习题7

采用两级页表的页式存储管理中，进程执行中CPU首次访问某逻辑地址时，需访问主存的次数是（ ）。

A. 1次

B. 2次

C. 3次

D. 4次

练习

习题8

段页式存储管理汲取了页式管理和段式管理的长处，其实现原理结合了页式和段式管理的基本思想，即（ ）。

A. 用分段方法来分配和管理物理存储空间，用分页方法来管理用户地址空间

B. 用分段方法来分配和管理用户地址空间，用分页方法来管理物理存储空间

C. 用分段方法来分配和管理主存空间，用分页方法来管理辅存空间

D. 用分段方法来分配和管理辅存空间，用分页方法来管理主存空间

练习

习题8

段页式存储管理汲取了页式管理和段式管理的长处，其实现原理结合了页式和段式管理的基本思想，即（ ）。

A. 用分段方法来分配和管理物理存储空间，用分页方法来管理用户地址空间

B. 用分段方法来分配和管理用户地址空间，用分页方法来管理物理存储空间

C. 用分段方法来分配和管理主存空间，用分页方法来管理辅存空间

D. 用分段方法来分配和管理辅存空间，用分页方法来管理主存空间

谢 谢!
Thank you!