

第四讲物理内存管理：连续内存分配

第 1 节计算机体系结构和内存层次

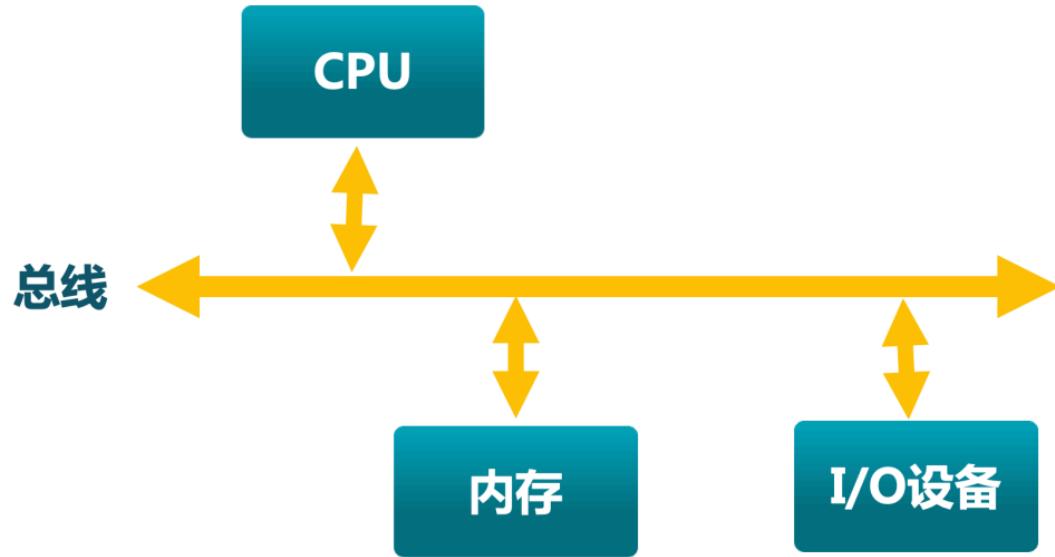
向勇、陈渝

清华大学计算机系

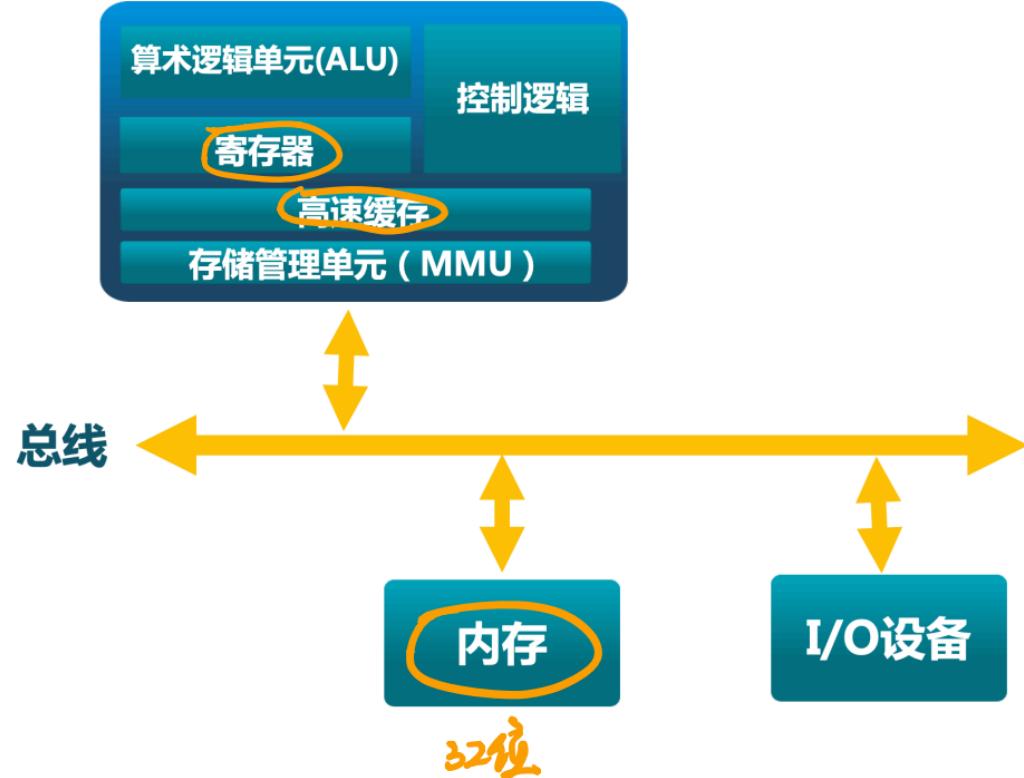
xyong,yuchen@tsinghua.edu.cn

2020 年 5 月 5 日

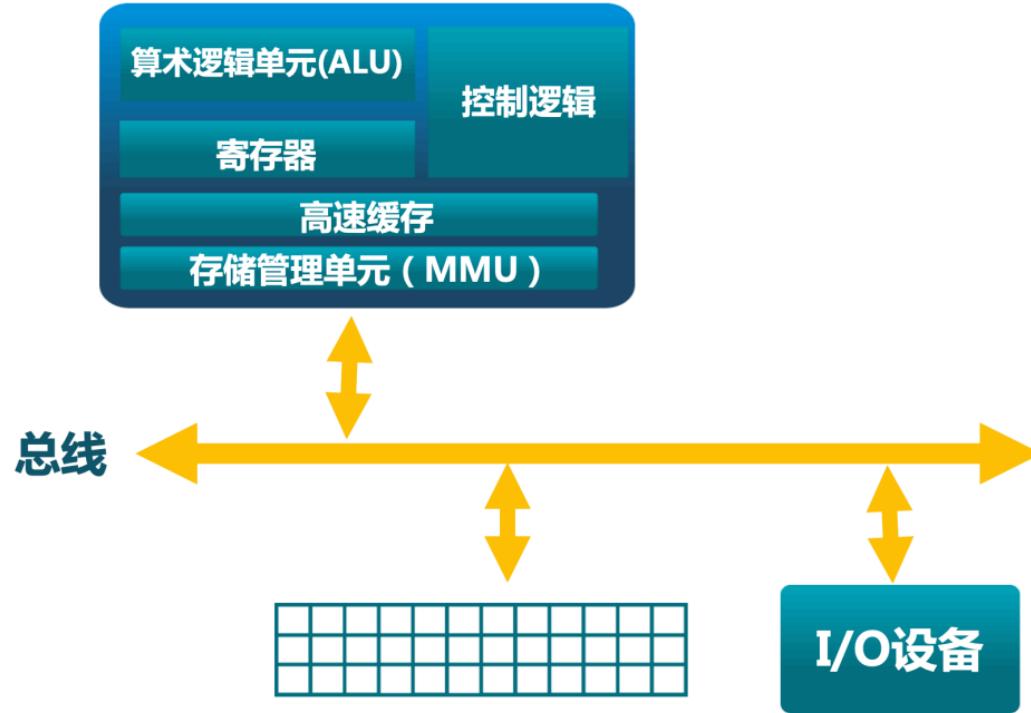
计算机体系结构



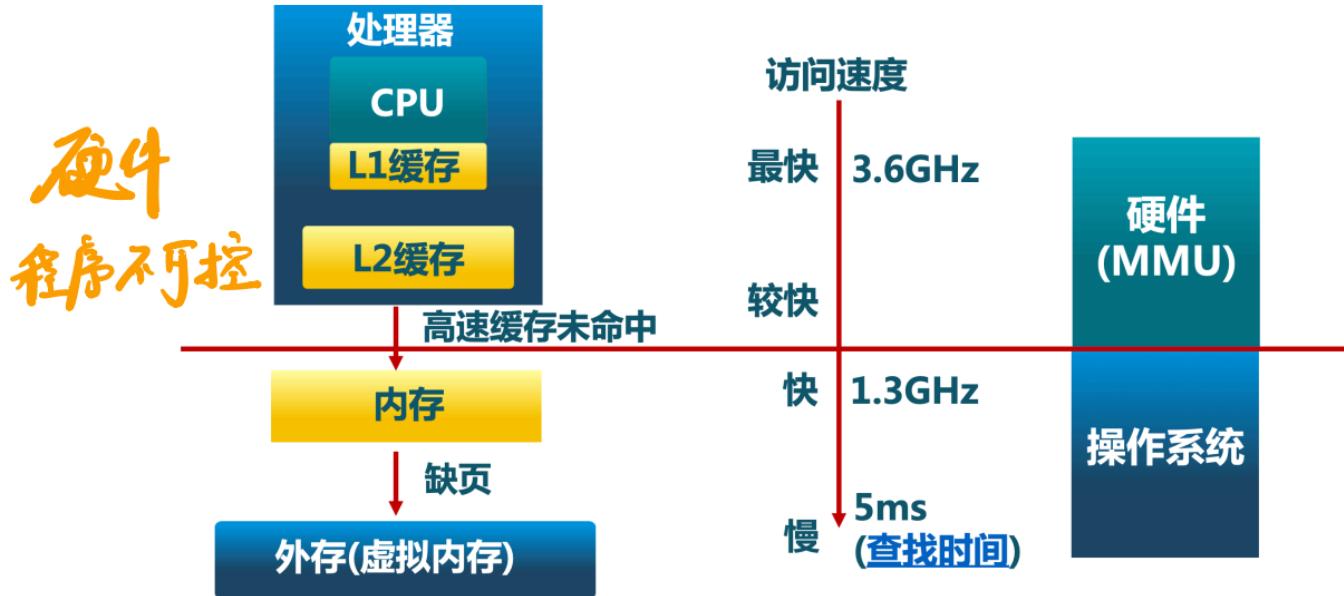
计算机体系结构



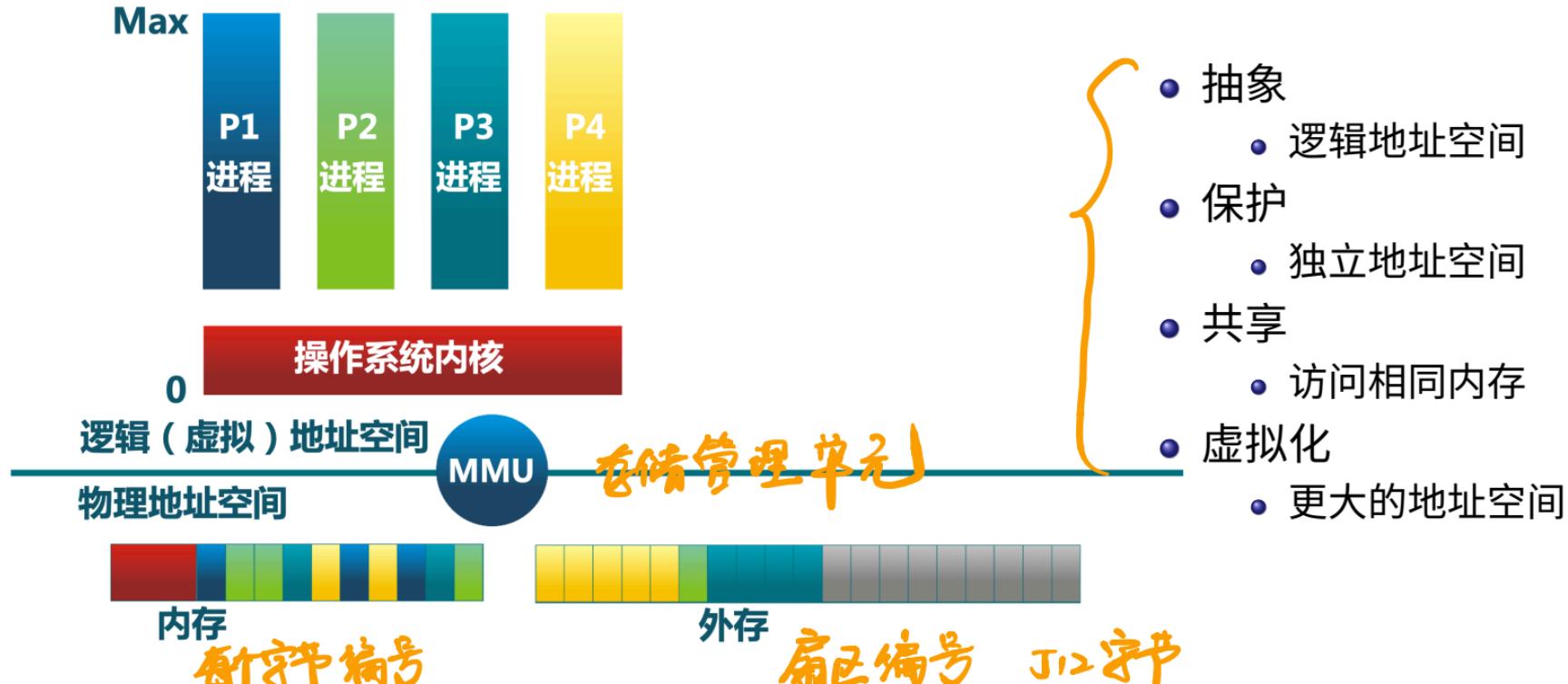
计算机体系结构



内存层次



操作系统的内存管理



操作系统的内存管理方式

- 操作系统中采用的内存管理方式
 - 重定位 (relocation)
 - 分段 (segmentation)
 - 分页 (paging) **连续 最小单位**
 - 虚拟存储 (virtual memory)
 - 目前多数系统 (如 Linux) 采用按需页式虚拟存储
- 实现高度依赖硬件
 - 与计算机存储架构紧耦合
 - MMU(内存管理单元): 处理 CPU 存储访问请求的硬件

第四讲物理内存管理：连续内存分配

第 2 节再看程序的地址空间

向勇、陈渝

清华大学计算机系

xyong,yuchen@tsinghua.edu.cn

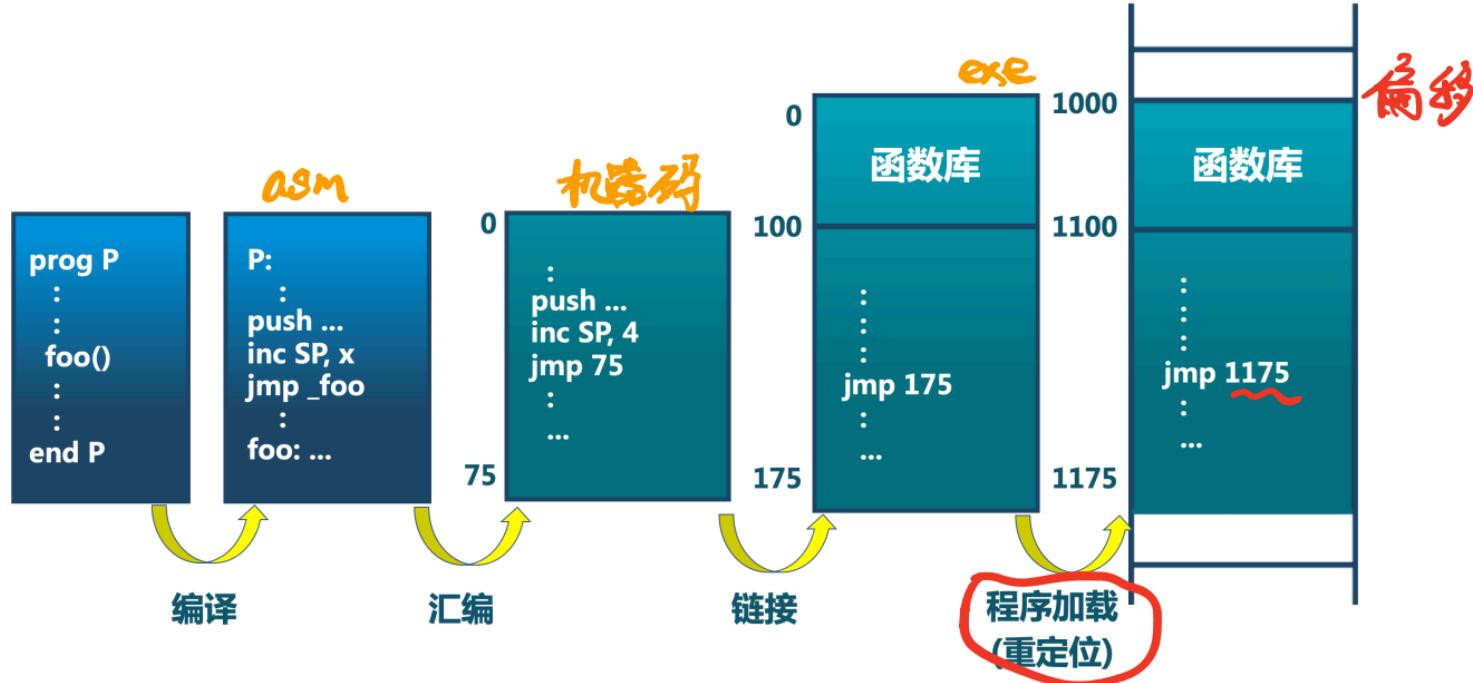
2020 年 5 月 5 日

地址空间定义



- 物理地址空间—硬件支持的地址空间
 - 起始地址 0，直到 MAX_{sys}
- 逻辑地址空间—在 CPU 运行的进程看到的地址
 - 起始地址 0，直到 MAX_{prog}

逻辑地址生成

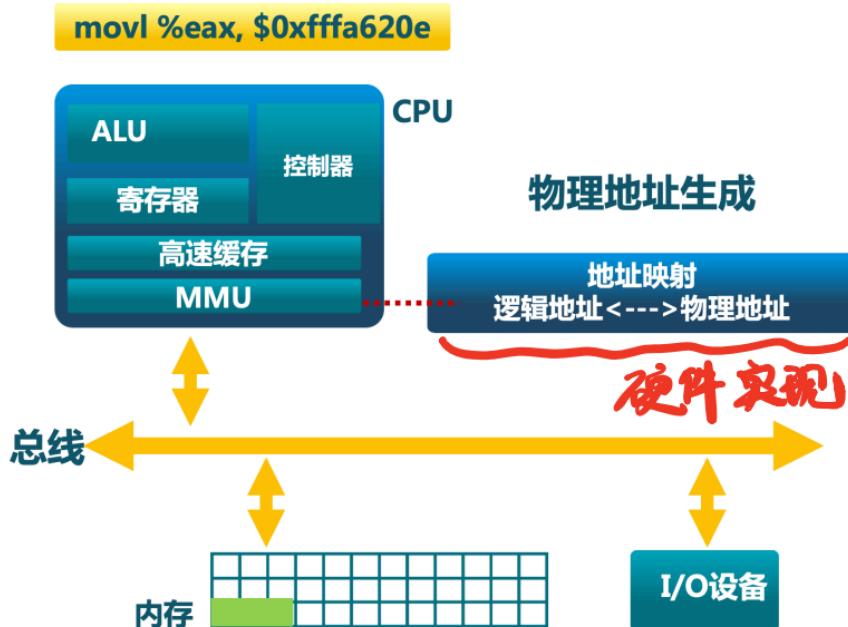


地址生成时机和限制

- 编译时
 - 假设起始地址已知
 - 如果起始地址改变，必须重新编译
- 加载时
 - 如编译时起始位置未知，编译器需生成可重定位的代码 (relocatable code)
 - 加载时，生成绝对地址
- 执行时
 - 执行时代码可移动
 - 需地址转换 (映射) 硬件支持

!!
虚拟存储
运行时 映射
obj. 可移动 · 互换

地址生成过程



- CPU

- ALU: 需要逻辑地址的内存内容
- MMU: 进行逻辑地址和物理地址的转换
- CPU 控制逻辑: 给总线发送物理地址请求

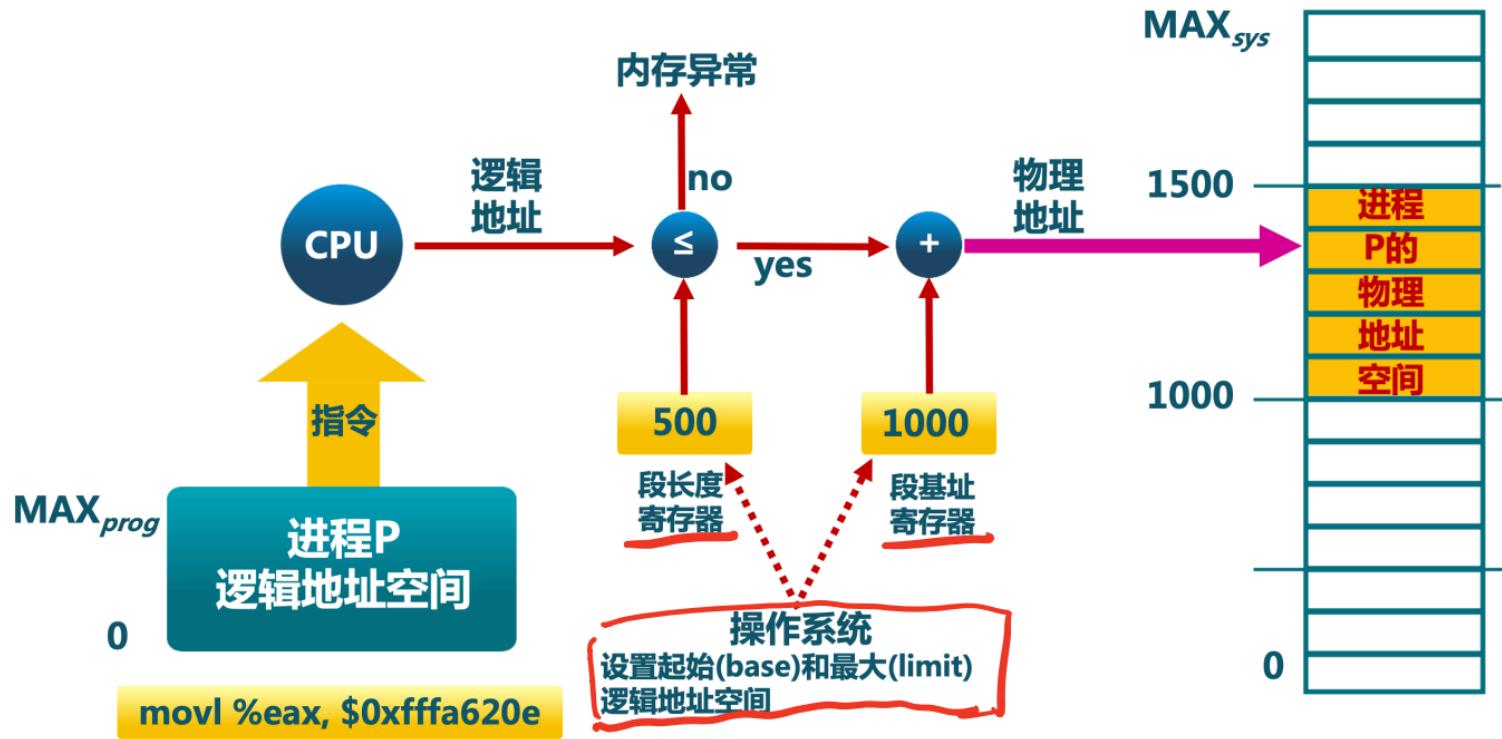
- 内存

- 发送物理地址的内容给 CPU
- 或接收 CPU 数据到物理地址

- 操作系统

- 建立逻辑地址 LA 和物理地址 PA 的映射

地址检查



第四讲物理内存管理：连续内存分配

第 3 节连续内存分配

向勇、陈渝

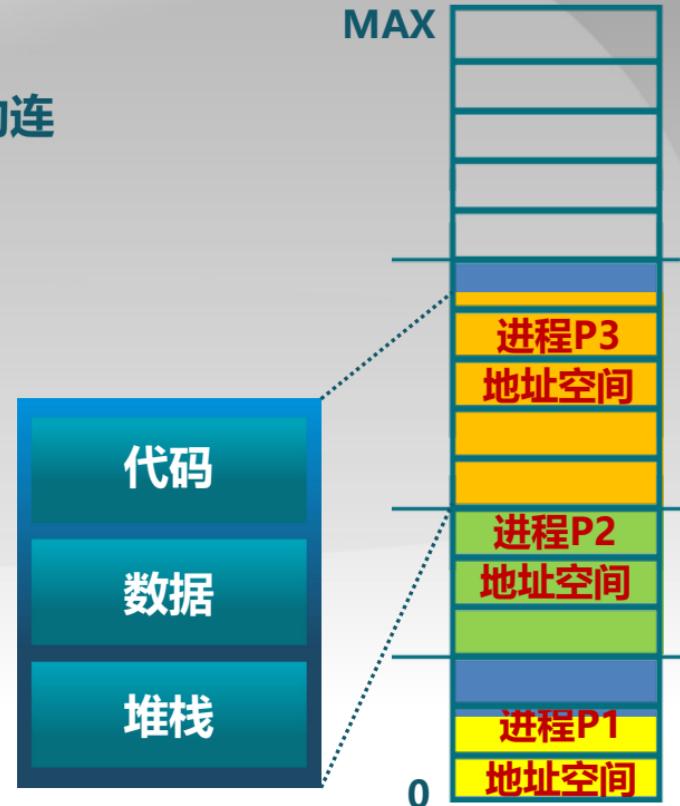
清华大学计算机系

xyong,yuchen@tsinghua.edu.cn

2020 年 5 月 5 日

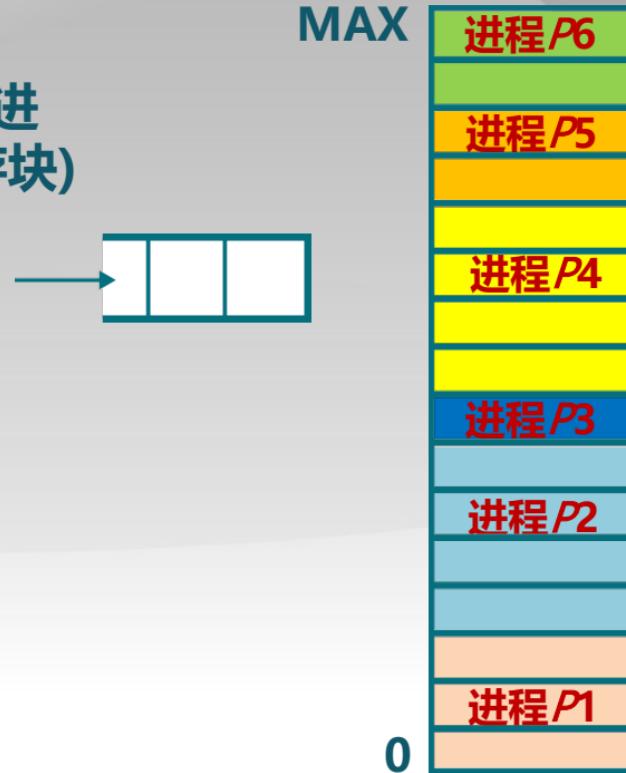
连续内存分配和内存碎片

- 连续内存分配
 - ▶ 给进程分配一块不小于指定大小的连续的物理内存区域
- 内存碎片
 - ▶ 空闲内存不能被利用
- 外部碎片
 - ▶ 分配单元之间的未被使用内存
- 内部碎片
 - ▶ 分配单元内部的未被使用内存
 - ▶ 取决于分配单元大小是否要取整



连续内存分配：动态分区分配

- 动态分区分配
 - ▶ 当程序被加载执行时，分配一个进程指定大小可变的分区(块、内存块)
 - ▶ 分区的地址是连续的
- 操作系统需要维护的数据结构
 - ▶ 所有进程的已分配分区
 - ▶ 空闲分区(Empty-blocks)
- 动态分区分配策略
 - ▶ 最先匹配(First-fit)
 - ▶ 最佳匹配(Best-fit)
 - ▶ 最差匹配(Worst-fit)



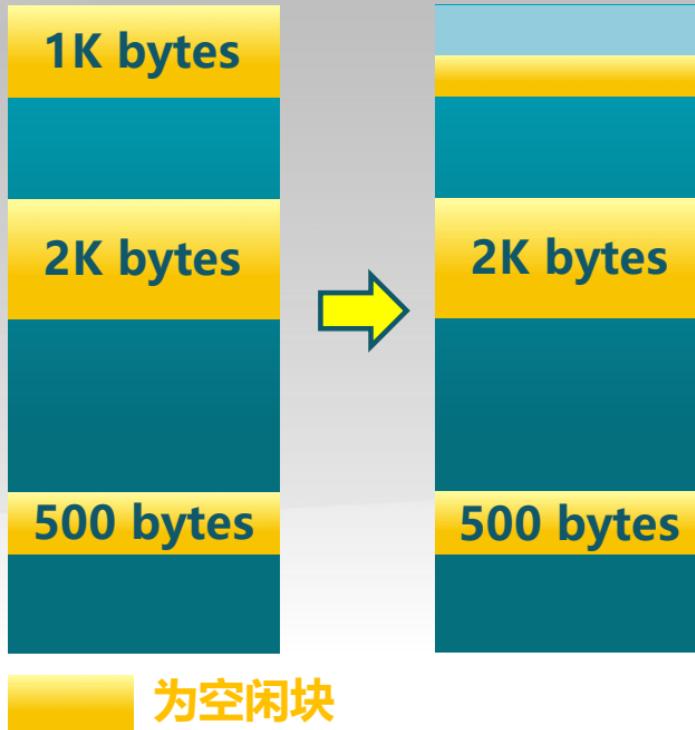
最先匹配(First Fit Allocation)策略

思路：

分配n个字节，使用第一个可用的空间比n大的空闲块。

示例：

分配400字节，使用第一个1KB的空闲块。



最先匹配(First Fit Allocation)策略

■ 原理 & 实现

- ▶ 空闲分区列表按地址顺序排序
- ▶ 分配过程时，搜索一个合适的分区
- ▶ 释放分区时，检查是否可与临近的空闲分区合并

■ 优点

- ▶ 简单
- ▶ 在高地址空间有大块的空闲分区

■ 缺点

- ▶ 外部碎片
- ▶ 分配大块时较慢

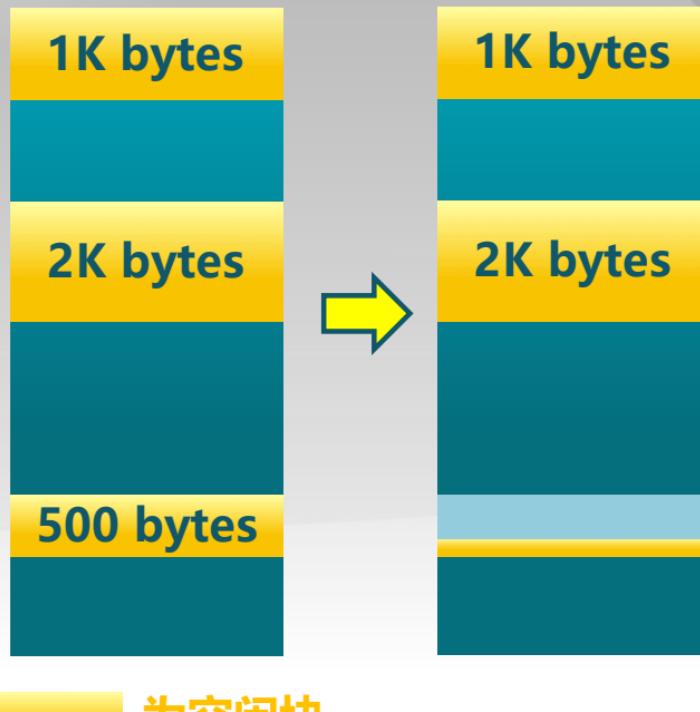
最佳匹配(Best Fit Allocation)策略

思路：

分配n字节分区时，查找并
使用不小于n的最小空闲分区

示例：

分配400字节， 使用第3个空
闲块(最小)



最佳匹配(Best Fit Allocation)策略

■ 原理 & 实现

- ▶ 空闲分区列表按照大小排序
- ▶ 分配时，查找一个合适的分区
- ▶ 释放时，查找并且合并临近的空闲分区（如果找到）

■ 优点

- ▶ 大部分分配的尺寸较小时，效果很好
 - 可避免大的空闲分区被拆分
 - 可减小外部碎片的大小
 - 相对简单

■ 缺点

- ▶ 外部碎片
- ▶ 释放分区较慢
- ▶ 容易产生很多无用的小碎片

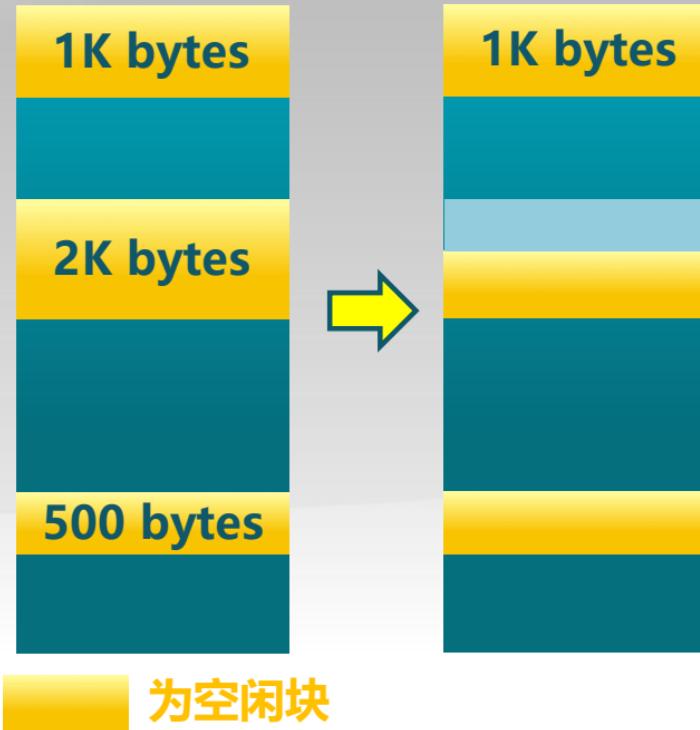
最差匹配(Worst Fit Allocation)策略

思路：

分配n字节，使用尺寸不
小于n的最大空闲分区

示例：

分配400字节，使用第
2个空闲块（最大）



最差匹配(Worst Fit Allocation)策略

■ 原理 & 实现

- ▶ 空闲分区列表按由大到小排序
- ▶ 分配时，选最大的分区
- ▶ 释放时，检查是否可与临近的空闲分区合并，进行可能的合并，并调整空闲分区列表顺序

■ 优点

- ▶ 中等大小的分配较多时，效果最好
- ▶ 避免出现太多的小碎片

■ 缺点

- ▶ 释放分区较慢
- ▶ 外部碎片
- ▶ 容易破坏大的空闲分区，因此后续难以分配大的分区

第四讲物理内存管理：连续内存分配

第 4 节碎片整理

向勇、陈渝

清华大学计算机系

xyong,yuchen@tsinghua.edu.cn

2020 年 5 月 5 日

碎片整理：紧凑(compaction)

■ 碎片整理

- ▶ 通过调整进程占用的分区位置来减少或避免分区碎片

■ 碎片紧凑

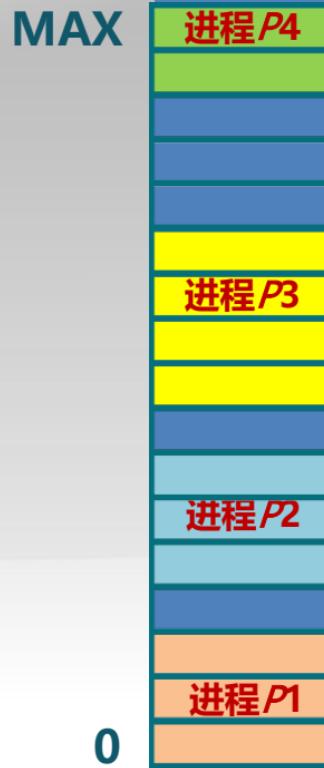
- ▶ 通过移动分配给进程的内存分区，以合并外部碎片

- ▶ 碎片紧凑的条件

- 所有的应用程序可动态重定位

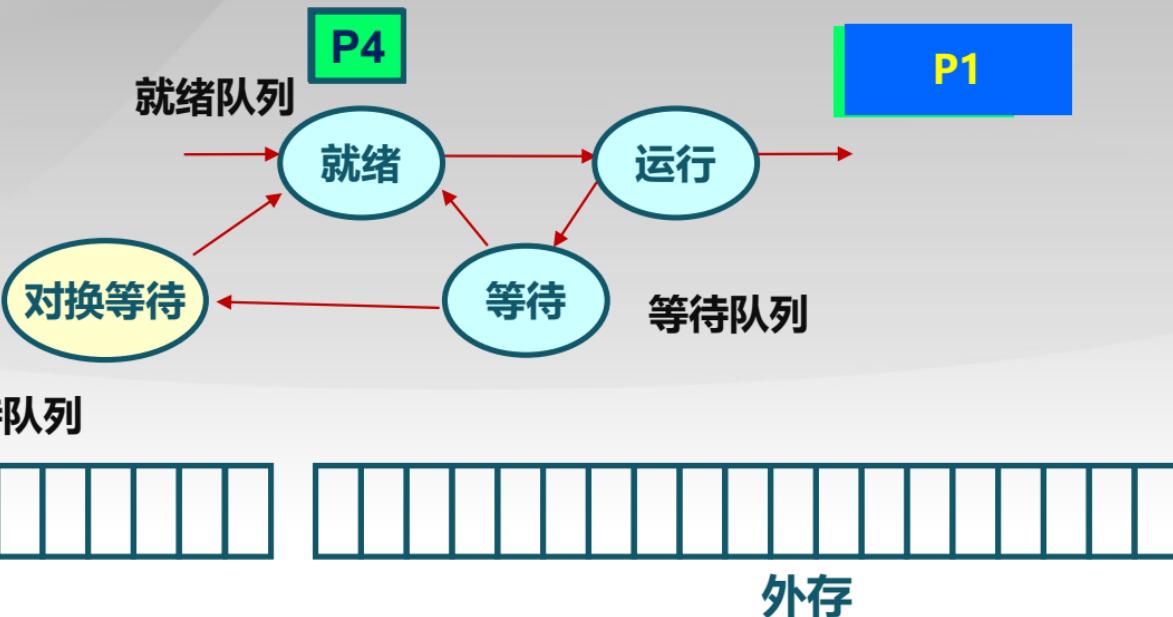
- ▶ 需要解决的问题

- 什么时候移动？
 - 开销

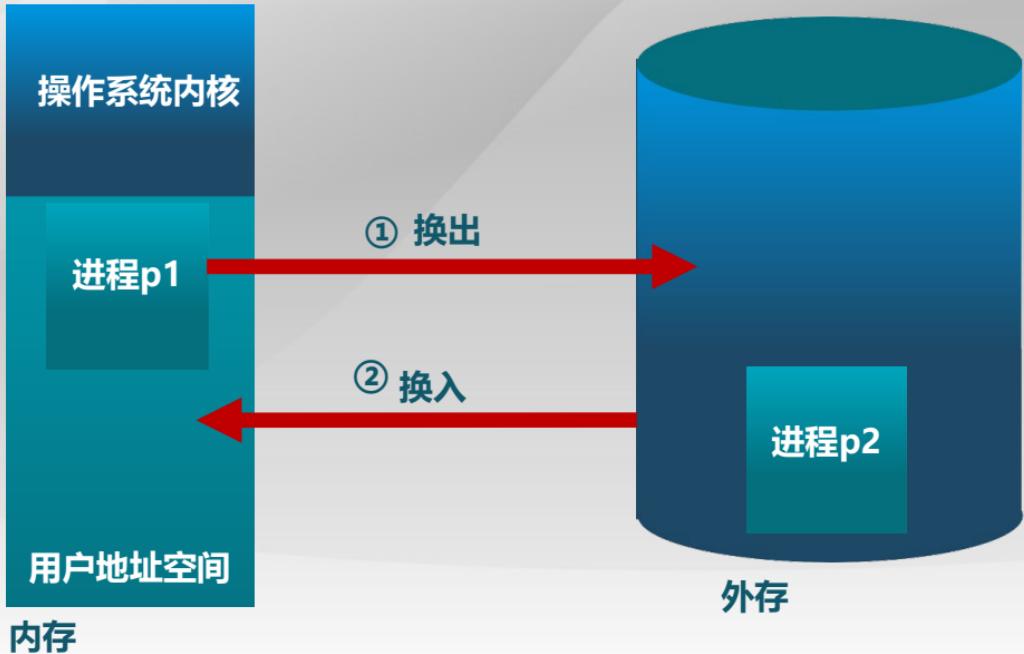


碎片整理：分区对换(Swapping in/out)

- 通过抢占并回收处于等待状态进程的分区，以增大可用内存空间



碎片整理：分区对换(Swapping in/out)



内存

■ 需要解决的问题

► 交换哪个（些）程序？

第四讲物理内存管理：连续内存分配

第 5 节伙伴系统分配算法与实现

向勇、陈渝

清华大学计算机系

xyong,yuchen@tsinghua.edu.cn

2020 年 5 月 5 日



操作系统

Operating System

伙伴系统的实现

■ 数据结构

- ▶ 空闲块按大小和起始地址组织成二维数组
- ▶ 初始状态：只有一个大小为 2^{u} 的空闲块

■ 分配过程

- ▶ 由小到大在空闲块数组中找最小的可用空闲块
- ▶ 如空闲块过大，对可用空闲块进行二等分，直到得到合适的可用空闲块

伙伴系统中的内存分配



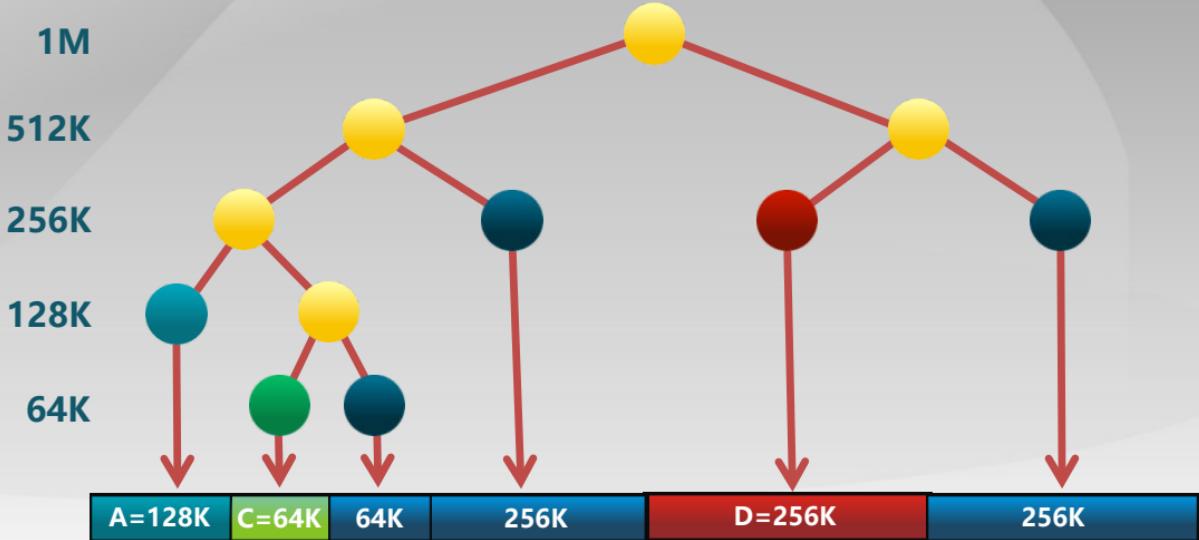
伙伴系统的实现

■ 释放过程

- ▶ 把释放的块放入空闲块数组
- ▶ 合并满足合并条件的空闲块

■ 合并条件

- ▶ 大小相同 2^i
- ▶ 地址相邻
- ▶ 低地址空闲块起始地址为 2^{i+1} 的位数



http://en.wikipedia.org/wiki/Buddy_memory_allocation

ucore中的物理内存管理

```
struct pmm_manager {  
    const char *name;  
    void (*init)(void);  
    void (*init_memmap)(struct Page *base, size_t n);  
    struct Page *(*alloc_pages)(size_t order);  
    void (*free_pages)(struct Page *base, size_t n);  
    size_t (*nr_free_pages)(void);  
    void (*check)(void);  
};
```

第四讲物理内存管理：连续内存分配

第 6 节 SLAB 分配器

向勇、陈渝

清华大学计算机系

xyong,yuchen@tsinghua.edu.cn

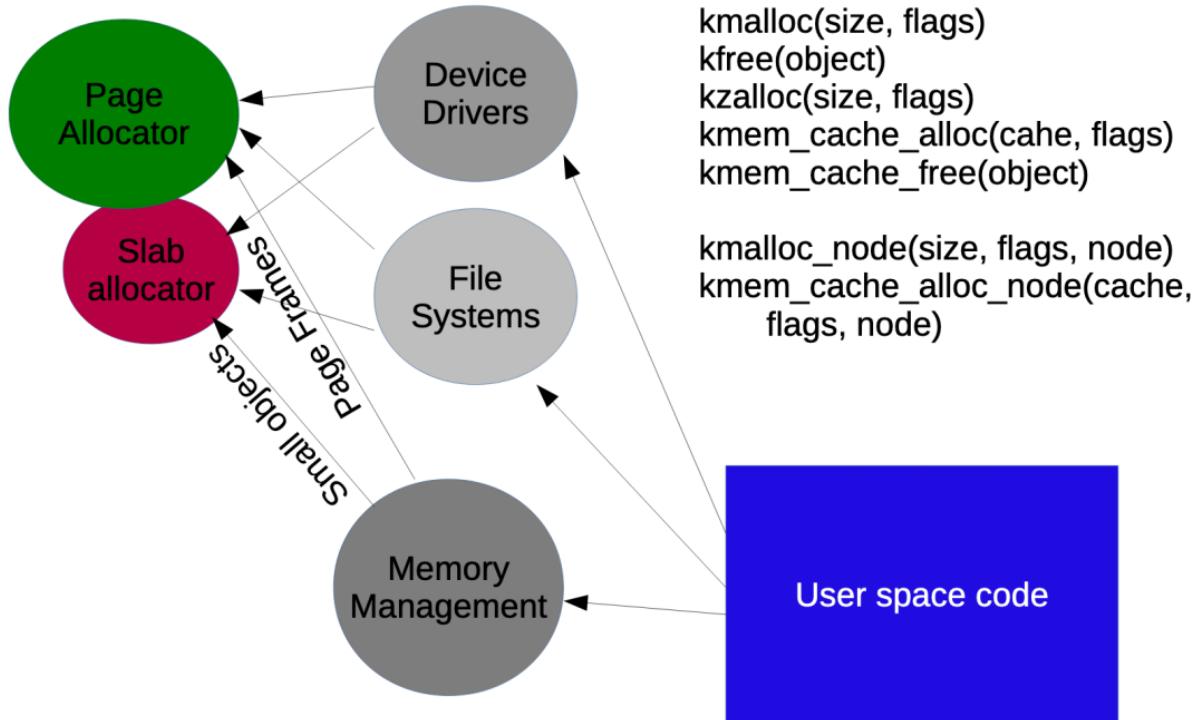
2020 年 5 月 5 日

rCore 中的物理内存管理

```
//Rust CODE
pub fn init(l: usize, r: usize)
pub fn init_allocator(l: usize, r: usize)
pub fn alloc_frame() -> Option<Frame>
pub fn alloc_frames(cnt: usize) -> Option<Frame>
pub fn deallocate_frame(f: Frame)
pub fn deallocate_frames(f: Frame, cnt: usize)
```

出处: [frame_allocator.rs](#)

与 SLAB 分配器相关的系统组成部件

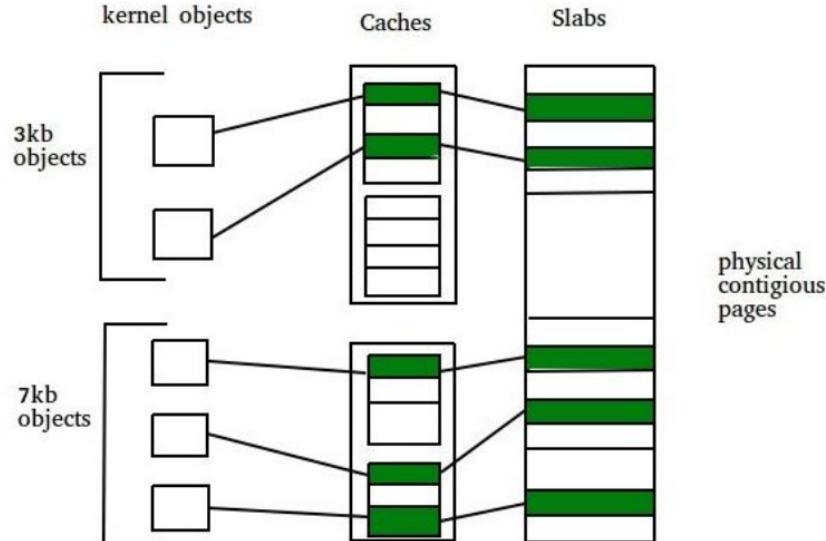


SLAB 分配器

SLAB 分配器源于 Solaris 2.4 的分配算法，工作于内存物理页分配算法之上，管理特定大小对象的缓存，进行快速高效的物理内存分配。

- 想解决的问题

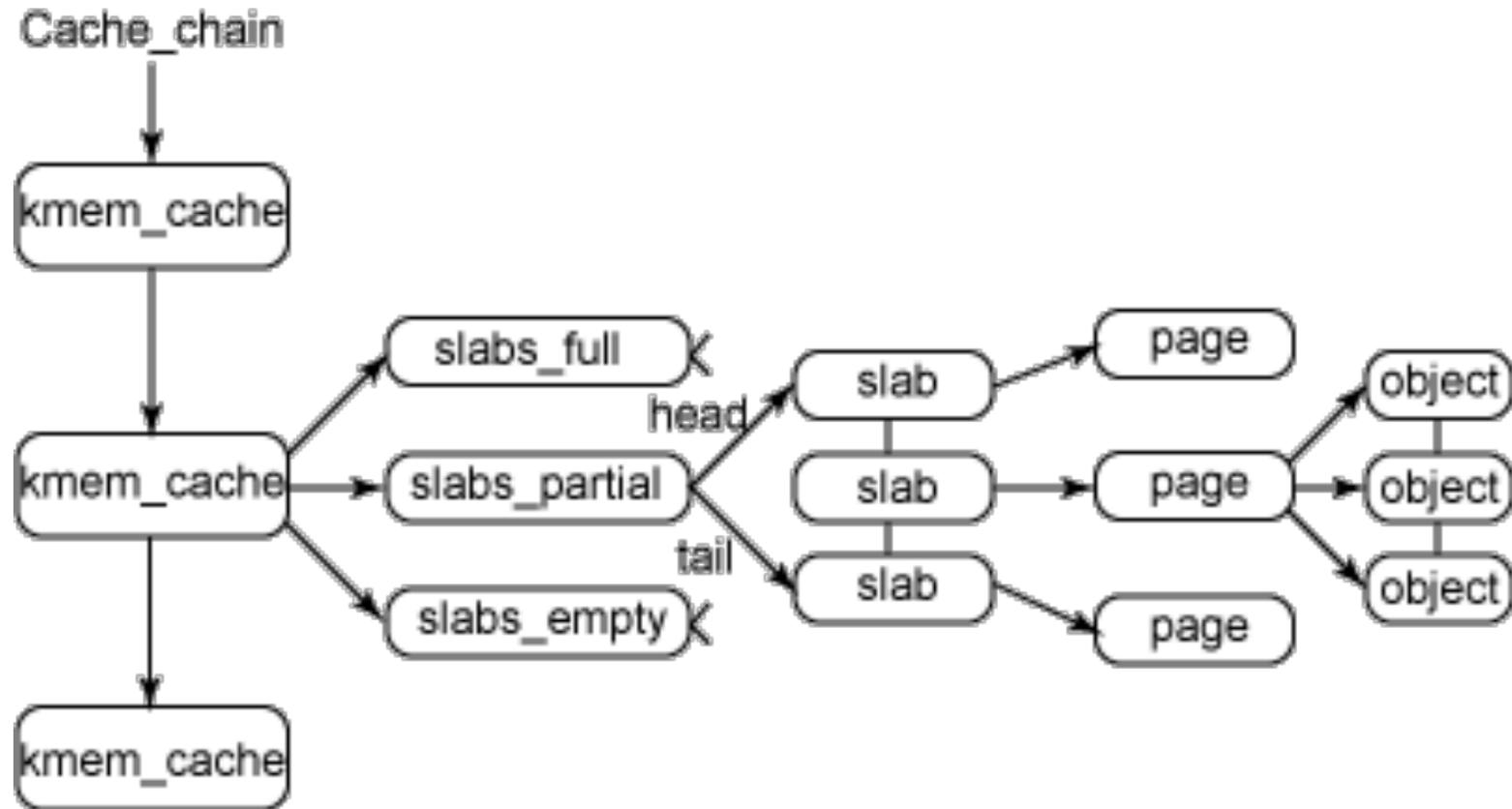
- 内核对象远小于页的大小
- 内核对象会被频繁的申请和释放
- 内核对象初始化时间超过分配和释放内存总时间



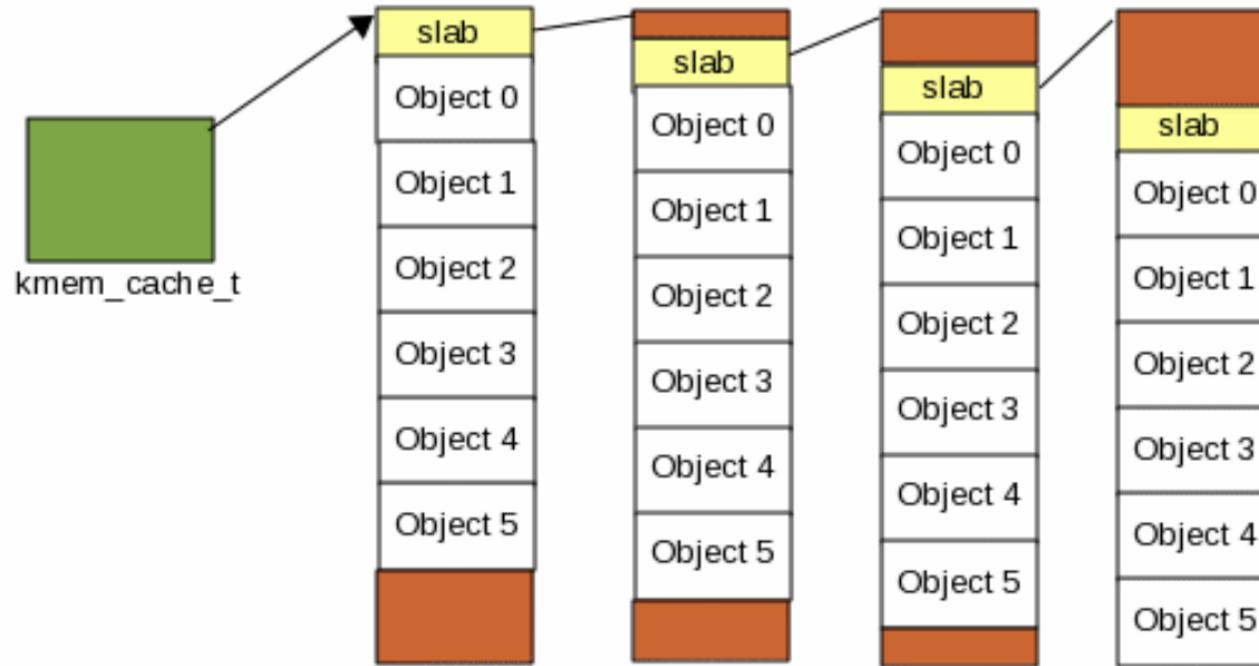
SLAB 分配器的特征

- 为每种使用的内核对象建立单独的缓冲区
- 按对象大小分组
- 两种 SLAB 对象状态：已分配或空闲
- 三类缓冲区队列：Full、Partial、Empty
- 优先从 Partial 队列中分配对象
- 缓冲区为每个处理器维护一个本地缓存

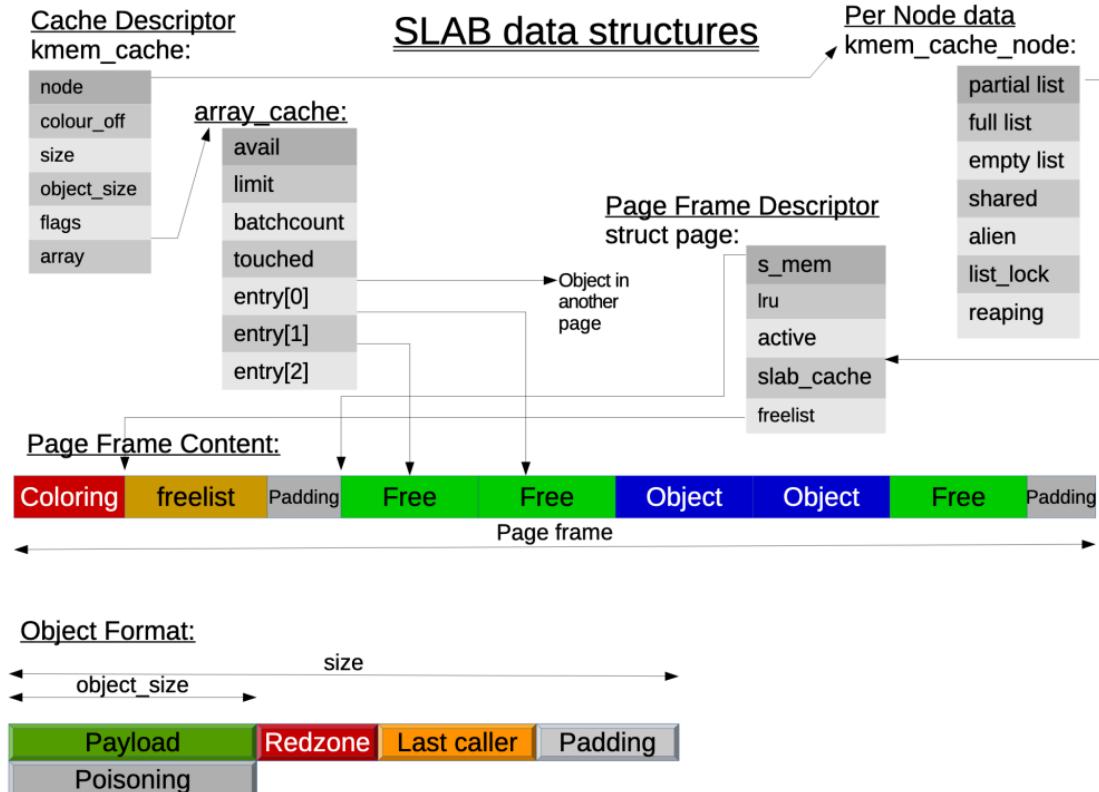
SLAB 分配器的结构



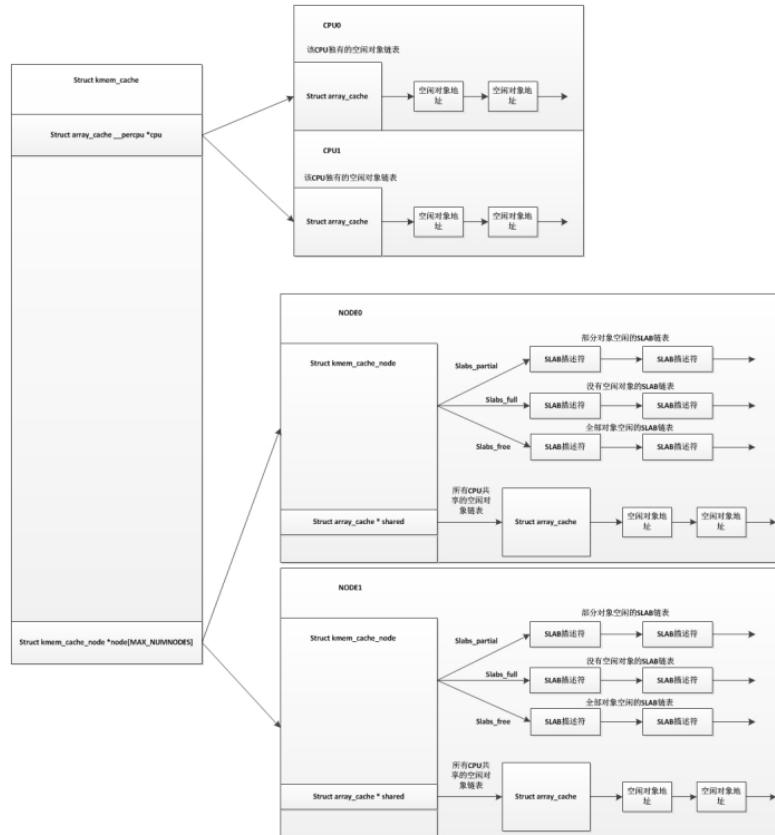
CPU 缓存着色与 SLAB



SLAB 的数据结构



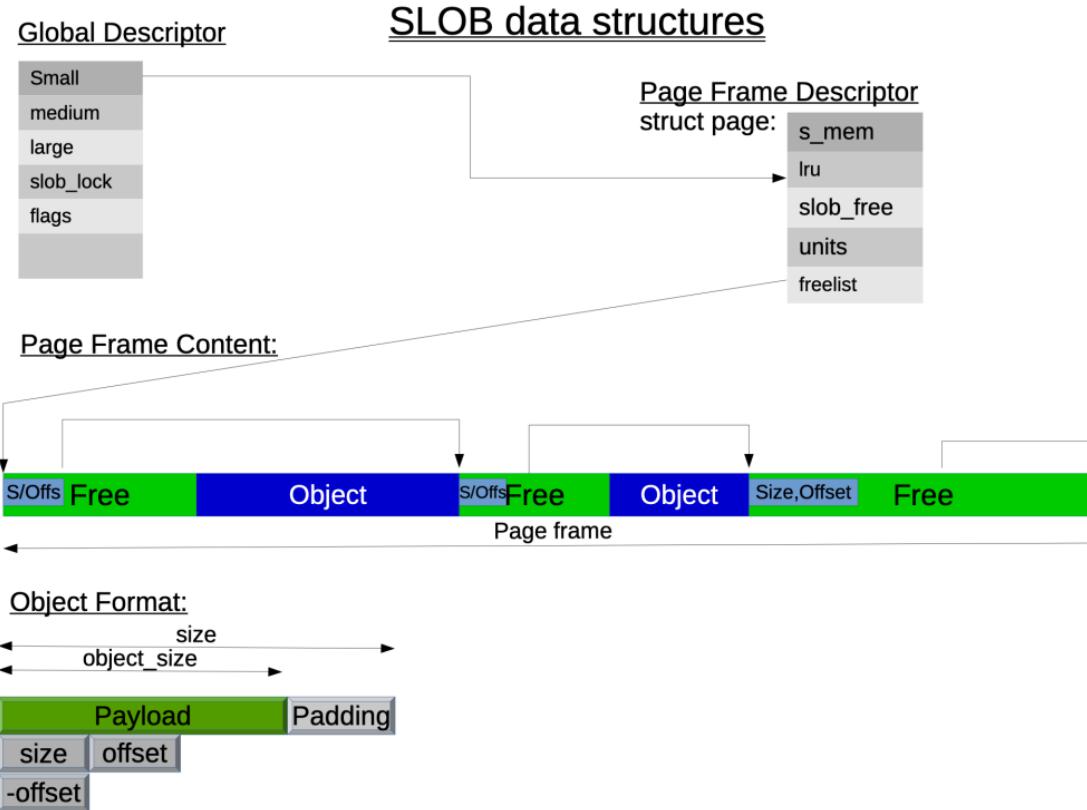
单个 SLAB 分配器结构



SLOB 分配器

- SLOB 分配器是针对嵌入式系统的 SLAB 简化版本
 - 没有本地 CPU 高速缓存和本地节点的概念
 - 只存在三个全局 partial free 链表
 - 链表按对象大小来划分

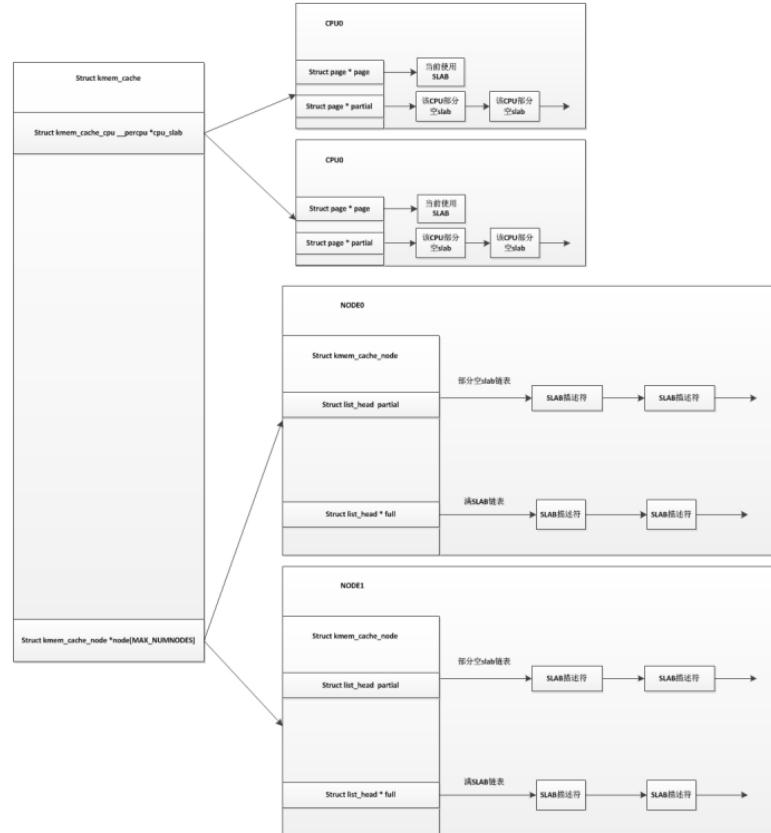
SLOB 分配器的数据结构



SLUB 分配器

- 目标
 - 简化设计理念
- 思路
 - 简化 SLAB 的结构：取消了大量的队列和相关开销
 - 一个 SLAB 是一组一个或多个页面，封装了固定大小的对象，内部没有元数据
 - 将元数据存储在页面相关的页结构
 - 没有单独的 Empty SLAB 队列

单个 SLUB 分配器结构



SLUB 分配器的数据结构

