

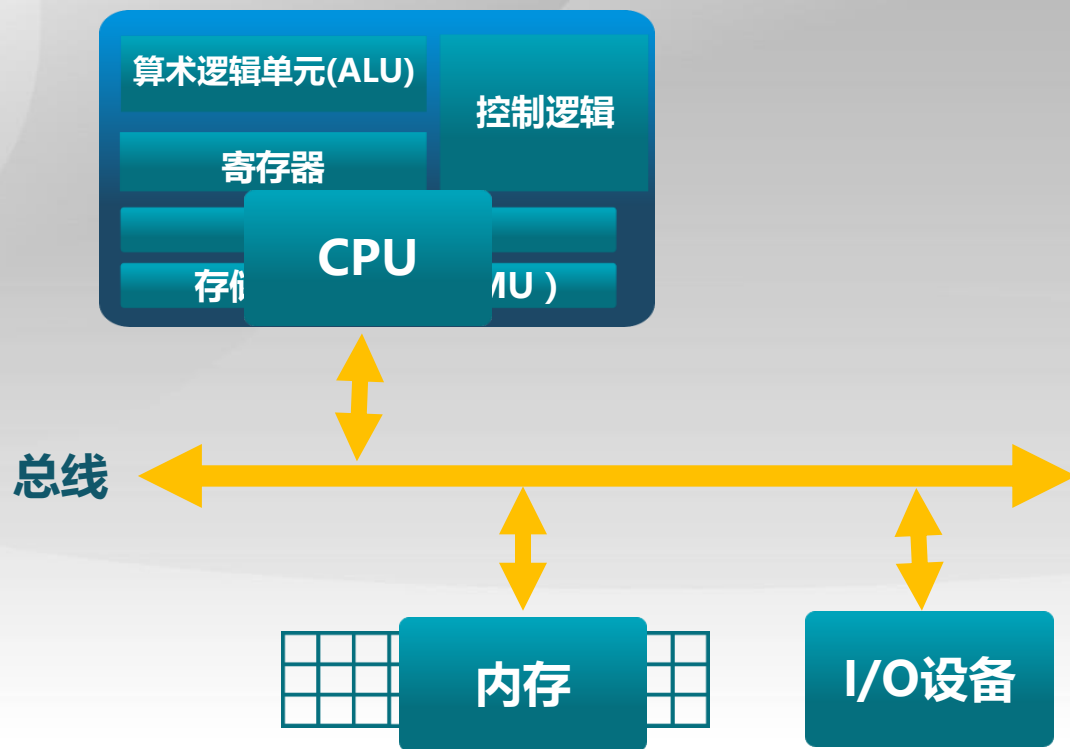


操作系统

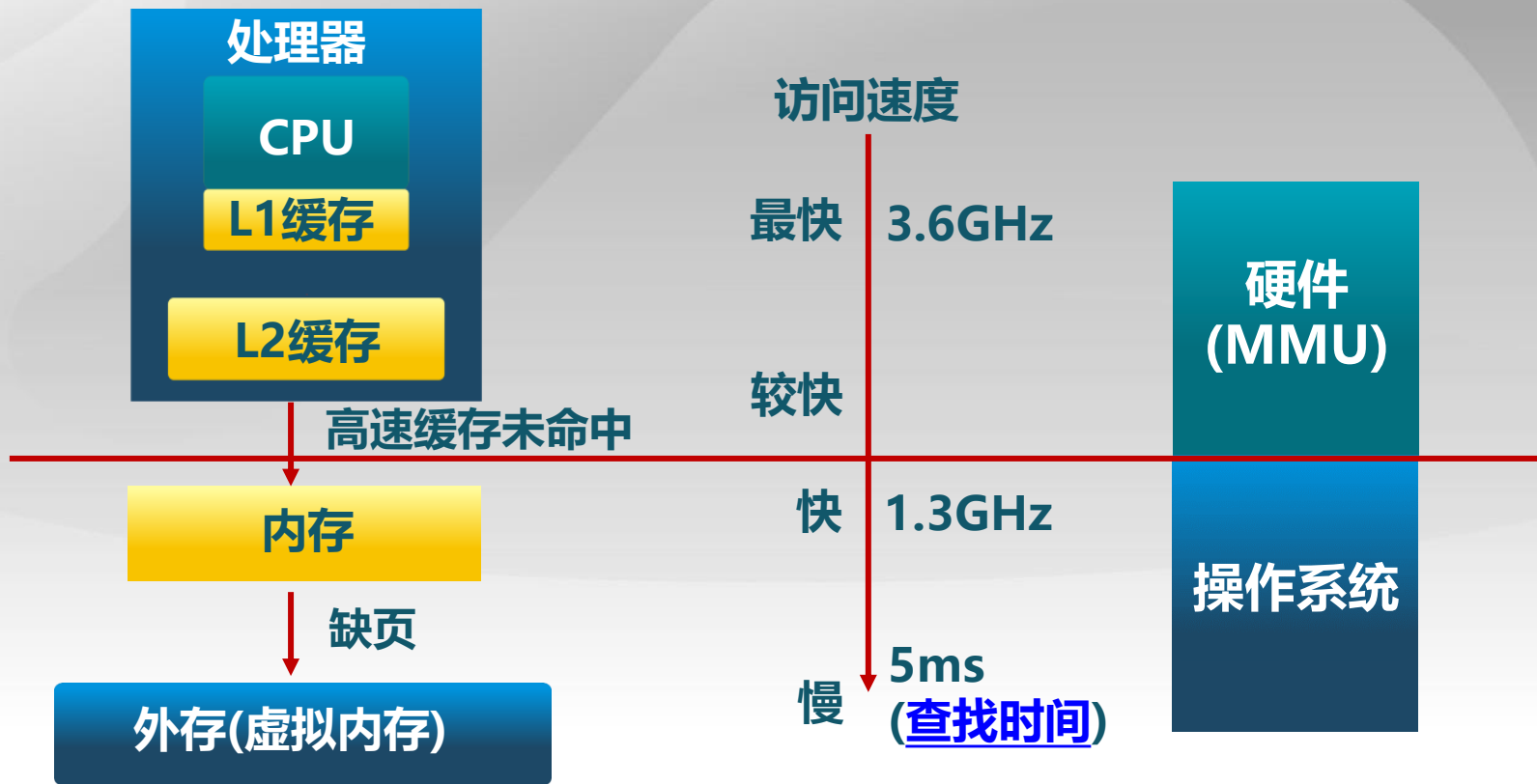
Operating System

Lec5 物理内存管理: 连续内存分配
清华大学计算机系

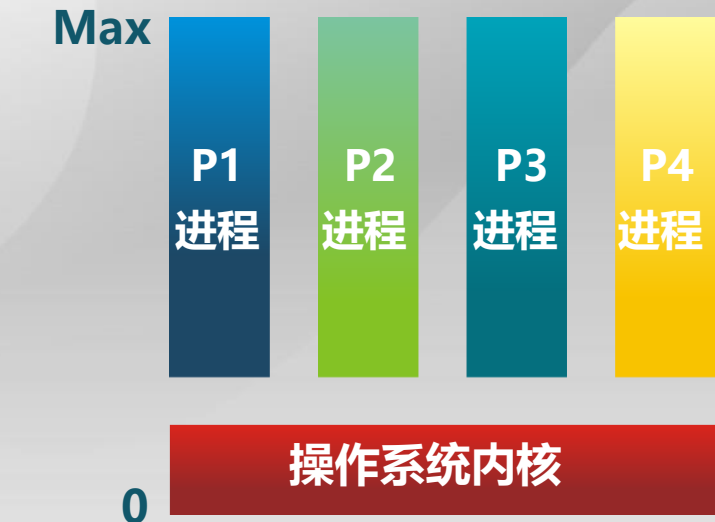
计算机体系结构



内存层次



操作系统的内存管理



逻辑（虚拟）地址空间

MMU

物理地址空间



内存



外存

- 抽象
 - ▣ 逻辑地址空间
- 保护
 - ▣ 独立地址空间
- 共享
 - ▣ 访问相同内存
- 虚拟化
 - ▣ 更大的地址空间

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)
- ▣ 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

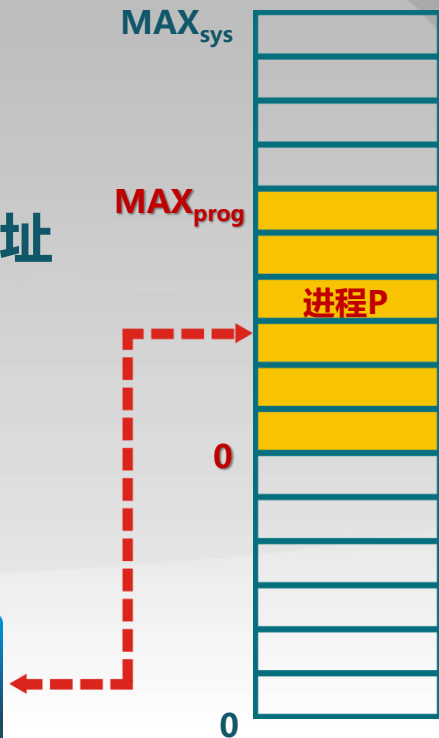


操作系统

Operating System

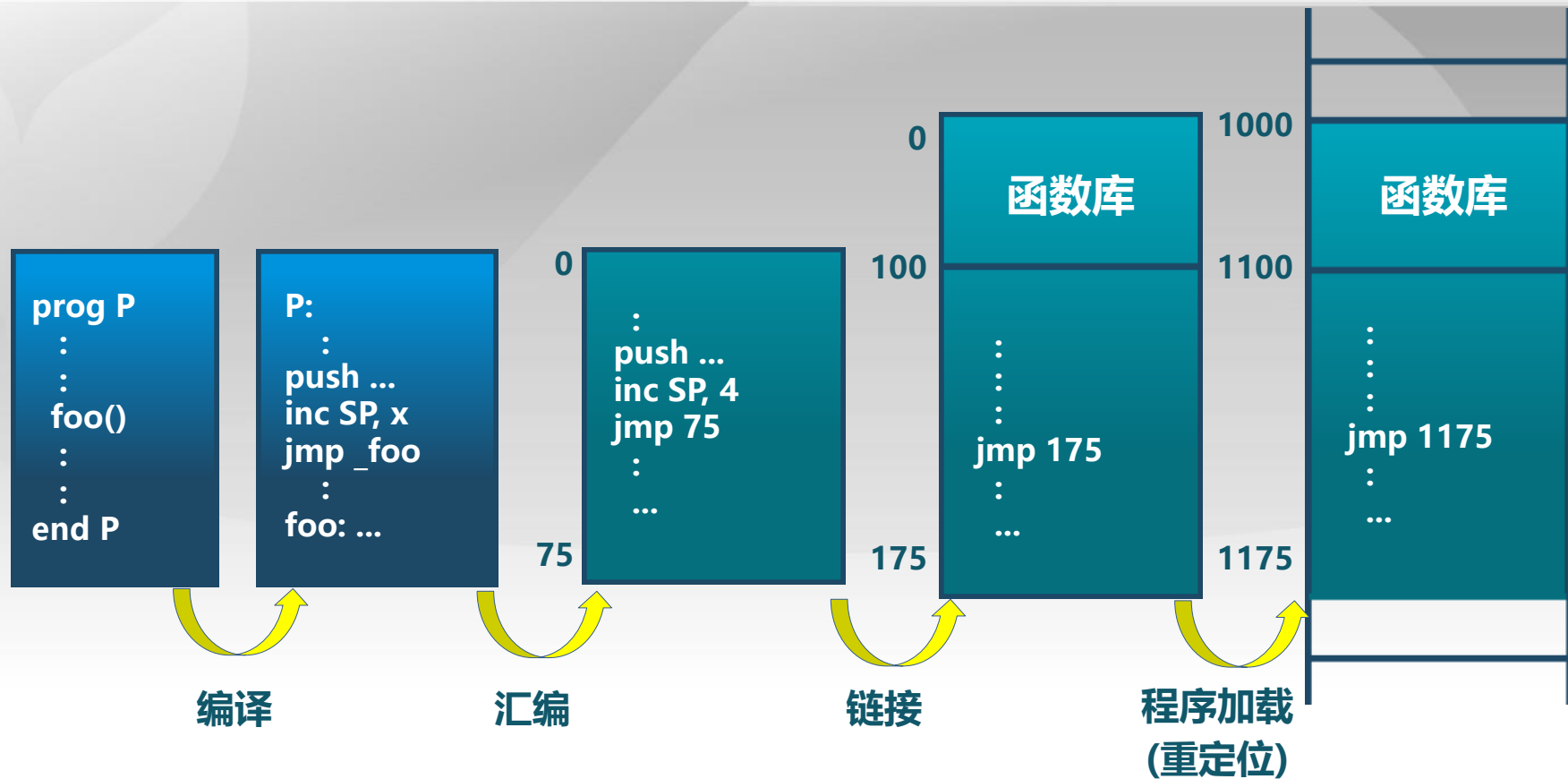
地址空间定义

- 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}



地址(address)是从哪里来的?
`movl %eax, $0xfffa620e`

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

地址生成过程

■ CPU

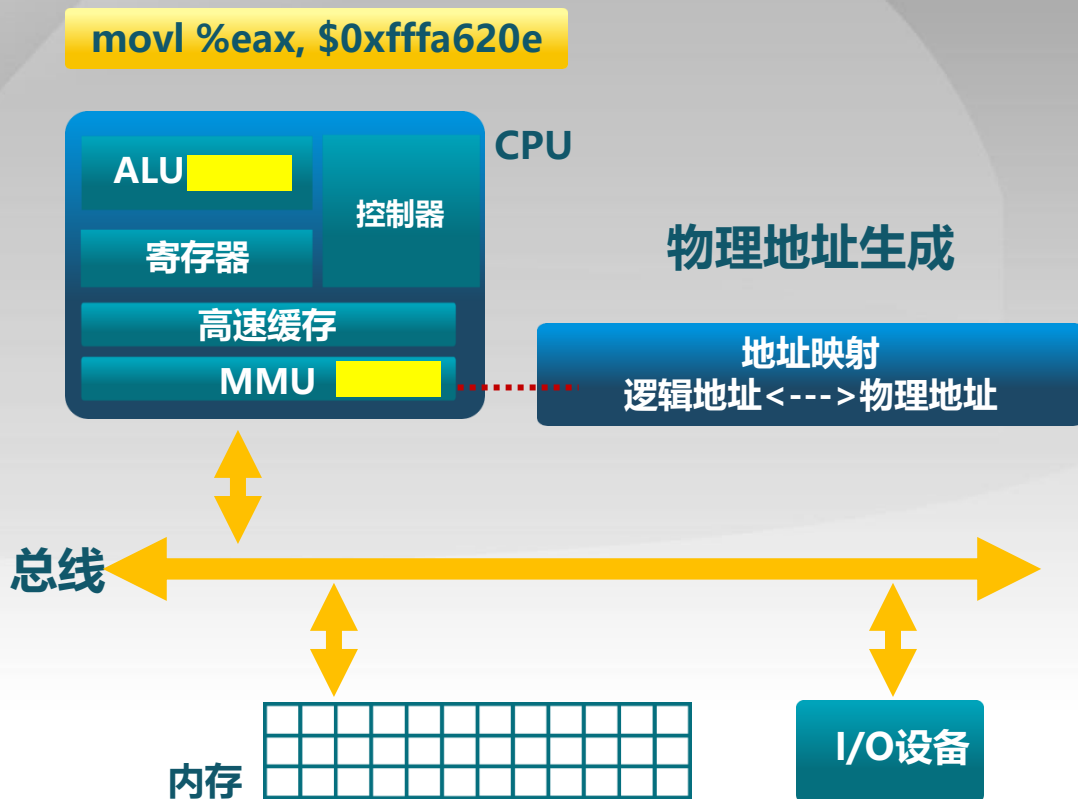
- ▶ ALU：需要逻辑地址的内存内容
- ▶ MMU：进行逻辑地址和物理地址的转换
- ▶ CPU控制逻辑：给总线发送物理地址请求

■ 内存

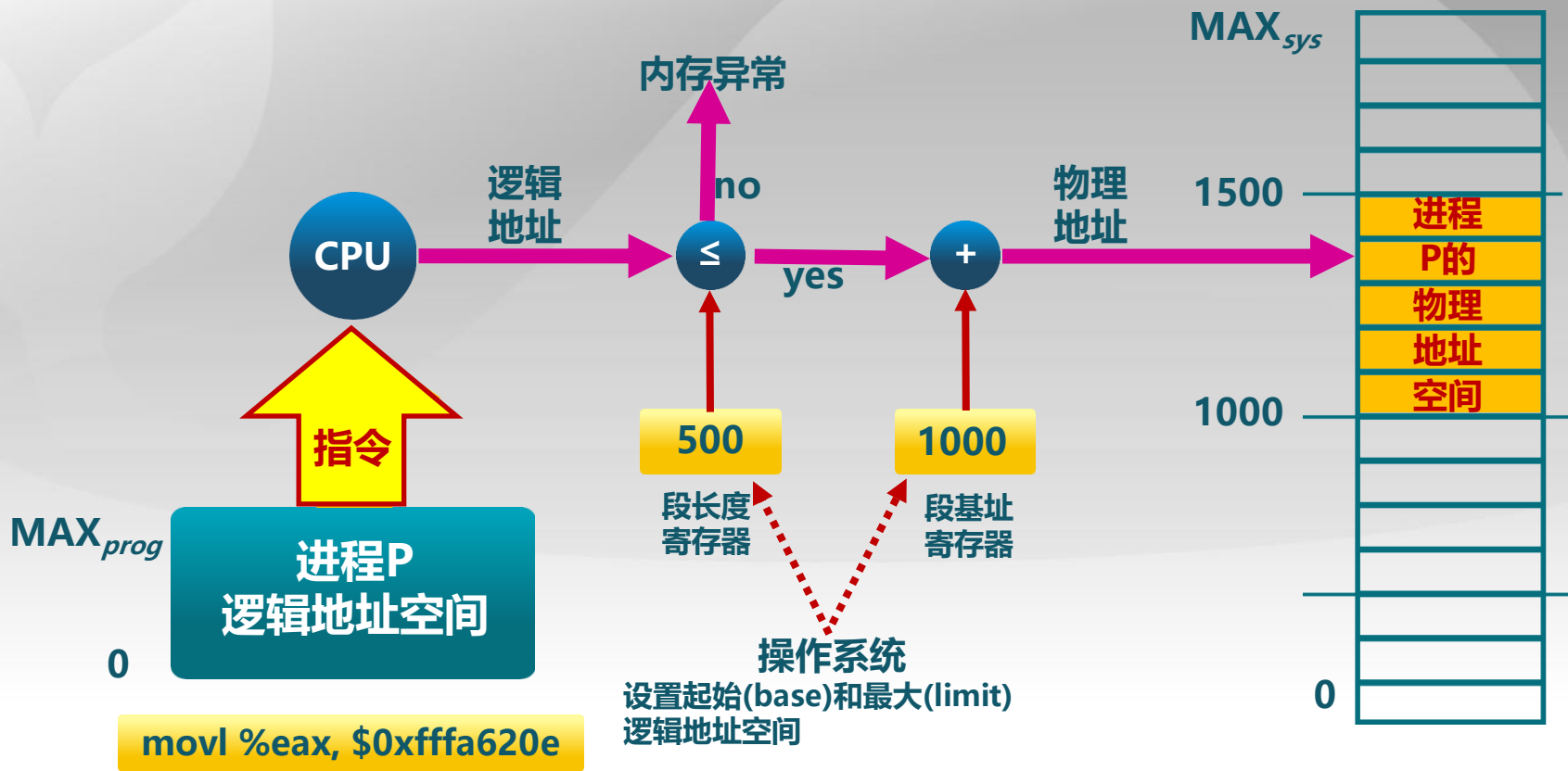
- ▶ 发送物理地址的内容给CPU
- ▶ 或接收CPU数据到物理地址

■ 操作系统

- ▶ 建立逻辑地址LA和物理地址PA的映射



地址检查



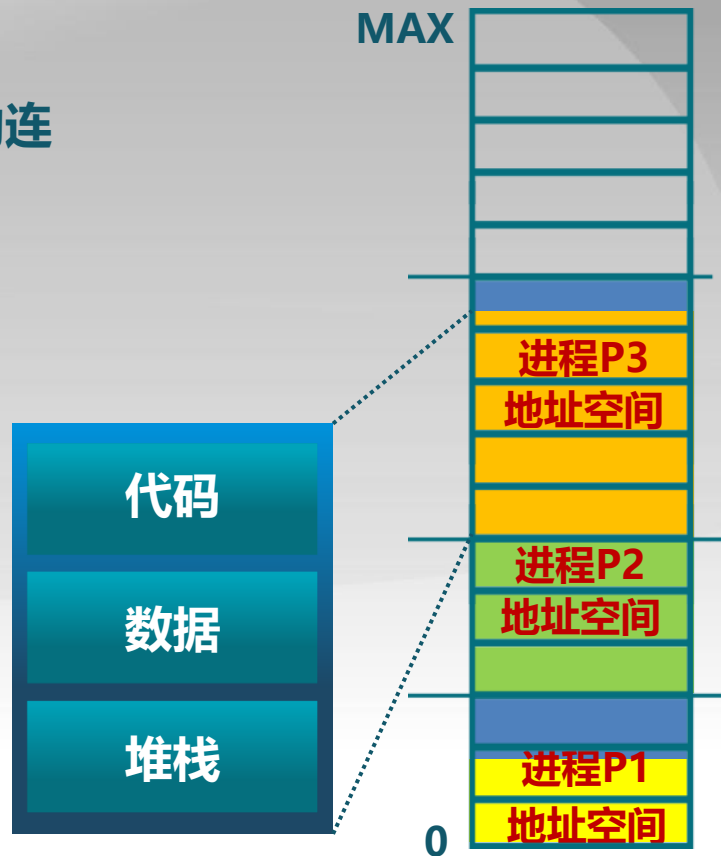


操作系统

Operating System

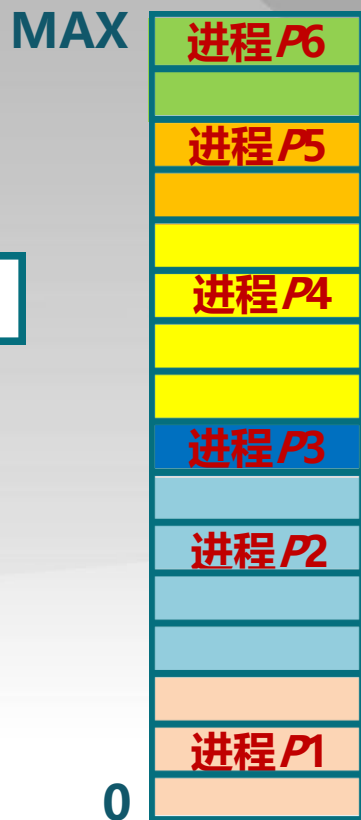
连续内存分配和内存碎片

- 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)



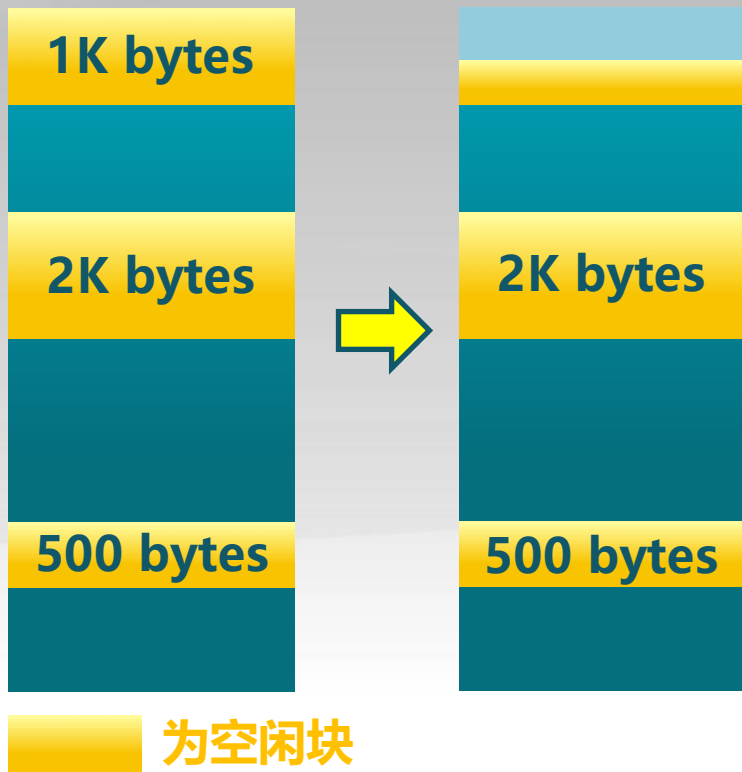
最先匹配(First Fit Allocation)策略

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

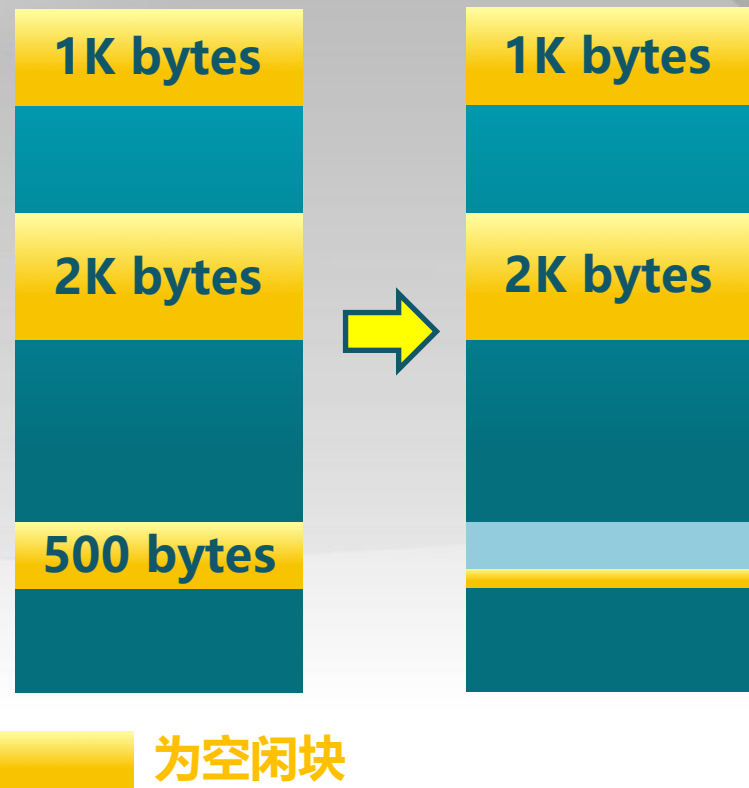
最佳匹配(Best Fit Allocation)策略

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

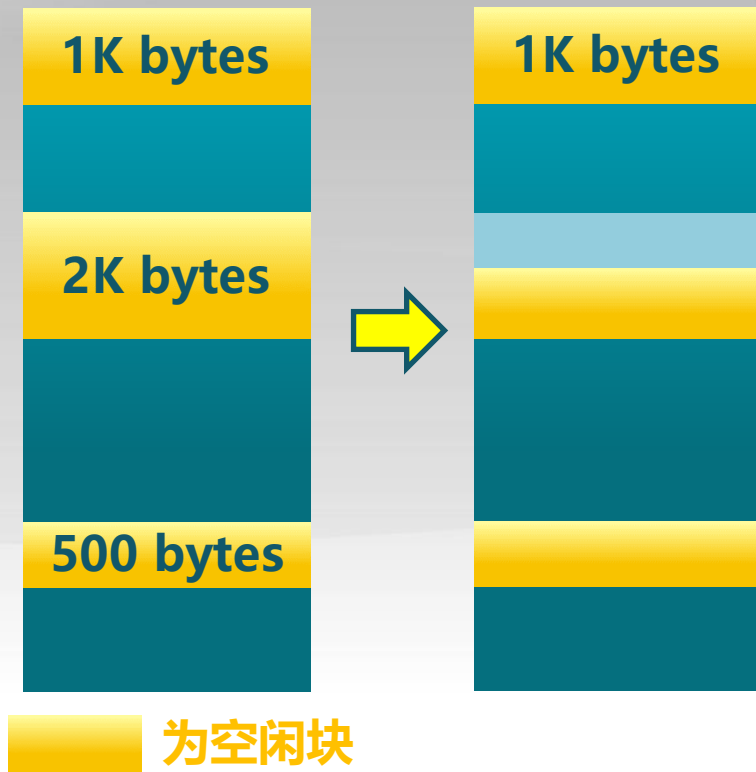
最差匹配(Worst Fit Allocation)策略

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



操作系统

Operating System

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

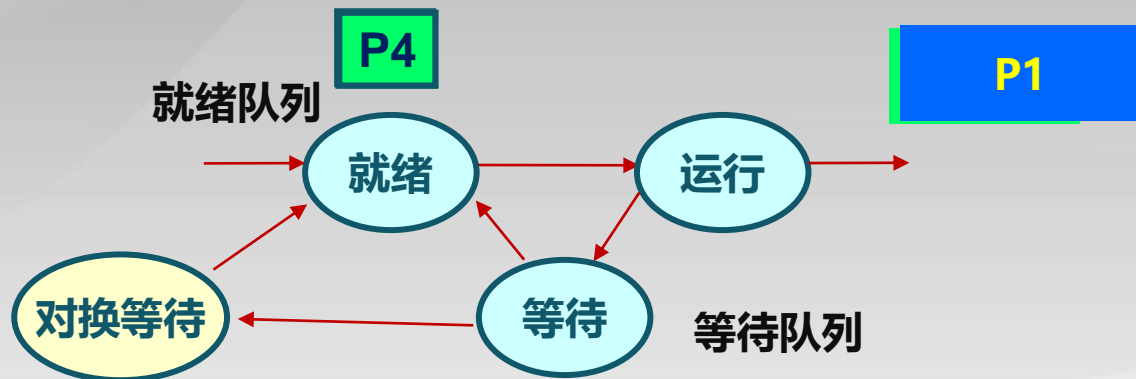
■ 碎片紧凑

- ▶ 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



对换等待队列

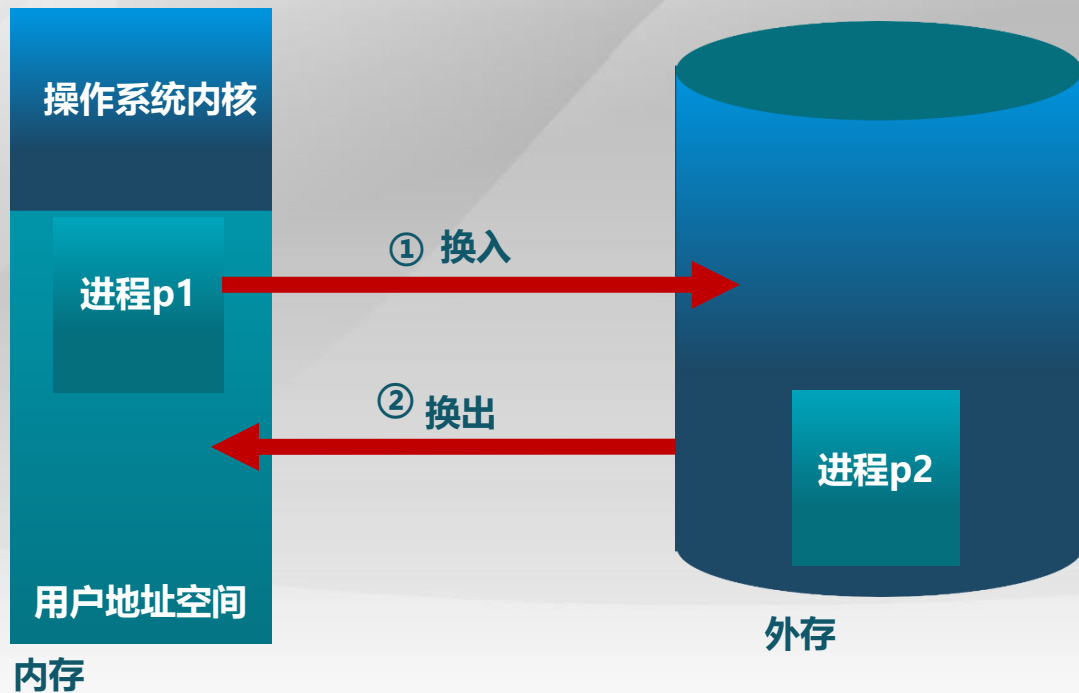


内存



外存

碎片整理：分区对换(Swapping in/out)



■ 需要解决的问题

▣ 交换哪个（些）程序？



操作系统

Operating System

伙伴系统(Buddy System)

- 整个可分配的分区大小 2^U
- 需要的分区大小为 $2^{U-1} < s \leq 2^U$ 时，把整个块分配给该进程；
 - ▣ 如 $s \leq 2^{i-1}$ ，将大小为 2^i 的当前空闲分区划分成两个大小为 2^{i-1} 的空闲分区
 - ▣ 重复划分过程，直到 $2^{i-1} < s \leq 2^i$ ，并把一个空闲分区分配给该进程

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



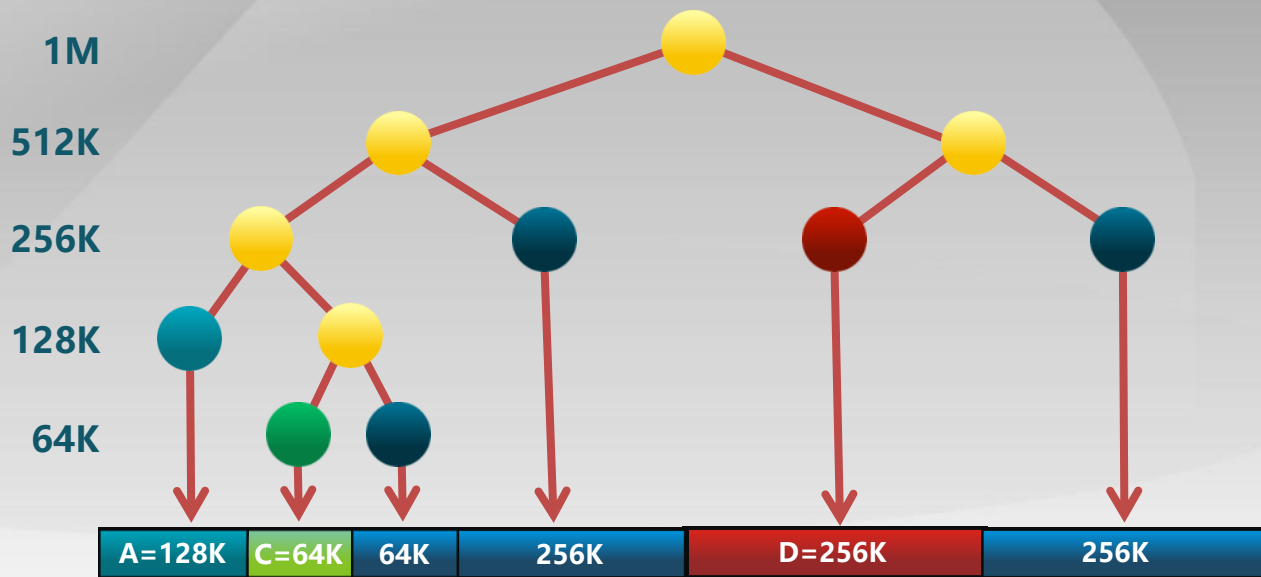
伙伴系统的实现

■ 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);  
};
```

ucore中的伙伴系统实现

```
const struct pmm_manager buddy_pmm_manager = {  
    .name = "buddy_pmm_manager",  
    .init = buddy_init,  
    .init_memmap = buddy_init_memmap,  
    .alloc_pages = buddy_alloc_pages,  
    .free_pages = buddy_free_pages,  
    .nr_free_pages = buddy_nr_free_pages,  
    .check = buddy_check,  
};
```



操作系统

Operating System