

操作系统A

Principles of Operating System

北京大学计算机科学技术系 陈向群

Department of computer science
and Technology, Peking University

2020 Autumn

存储管理——大纲

□ 重要概念

- 存储体系、存储保护、地址重定位

□ 物理内存管理

- 数据结构（位示图、空闲区表、空闲区链表）
- 分配算法（首先适配、最佳适配、最差适配 ……）

□ 各种存储管理方案

- 单一连续区、固定分区、可变分区、页式、段式、段页式

□ 虚拟存储管理

- 硬件、页表、页错误处理
- 软件策略：读取策略、放置策略、置换策略、驻留集策略、清除策略、装载控制策略

重要概念

地址转换
地址变换
地址翻译
地址映射

地址重定位

单一连续区

虚拟内存

驻留集

逻辑地址
物理地址

固定分区

虚拟
存储空间

工作集

存储保护

可变分区

虚拟地址
物理地址

页面置换算法

存储共享

页式

页表/页表项

清除策略

覆盖技术

段式

快表TLB

页缓冲技术

交换技术

段页式

Page Fault

加载控制

地址重定位、地址保护、.....

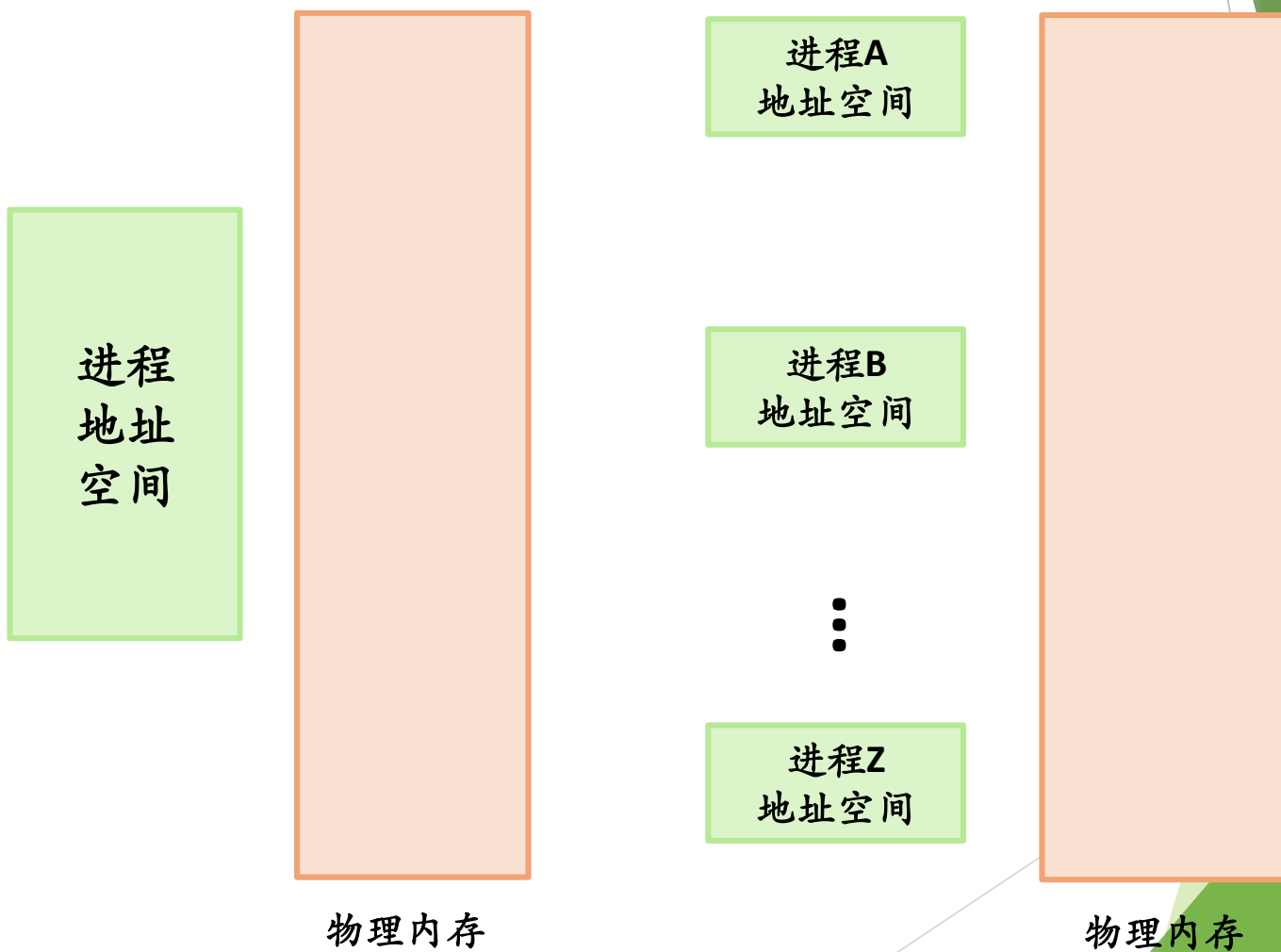
基本概念

……已经了解的

- ▶ 程序装载到内存才可以运行
程序以文件形式（**可执行文件格式**）保存在磁盘上
- ▶ 多道程序设计模型
允许多个程序同时进入内存
- ▶ 每个进程有自己独立的地址空间
 - ▶ 一个进程执行时不能访问另一个进程的地址空间
 - ▶ 不能执行不适当的操作
- ▶ 局部性原理
- ▶ 存储体系



要解决的问题



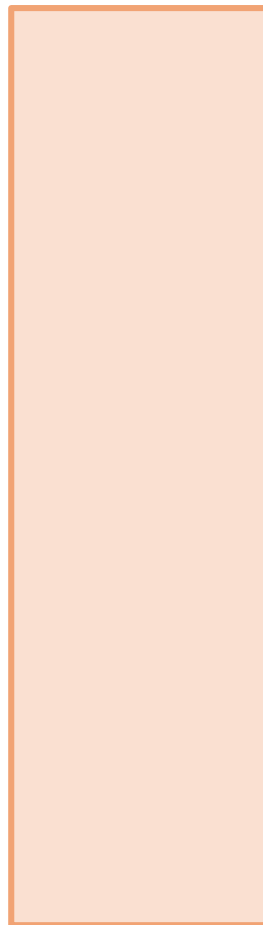
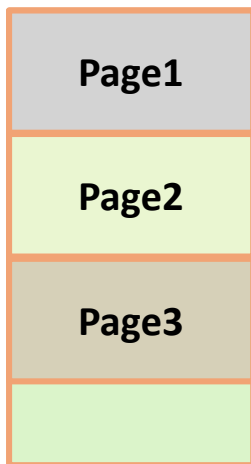
多道程序设计与内存管理

同时有多个
用户程序在
内存中

- ▶ 内存管理要做更多的工作
 - ▶ 需要支持地址重定位
 - ▶ 程序中的地址不一定是最终的物理地址
 - ▶ 在程序运行前无法计算出物理地址
因为不能确定程序被加载到内存什么地方
- ▶ 需要支持地址保护
 - ▶ 进程间地址空间不能互访问

要解决的问题

连续性 — 离散性
驻留性 — 交换性
一次性 — 多次性



物理内存

地址空间

程序发出的地址应与物理内存地址无关

0xffffffff	内核虚拟存储器
0xc0000000	用户栈 (运行时创建的)
	共享库的存储器 映射区域
0x04000000	
	运行时堆 (在运行时由malloc创建的)
	读/写数据
	只读的代码和数据
0x08048000	
0	

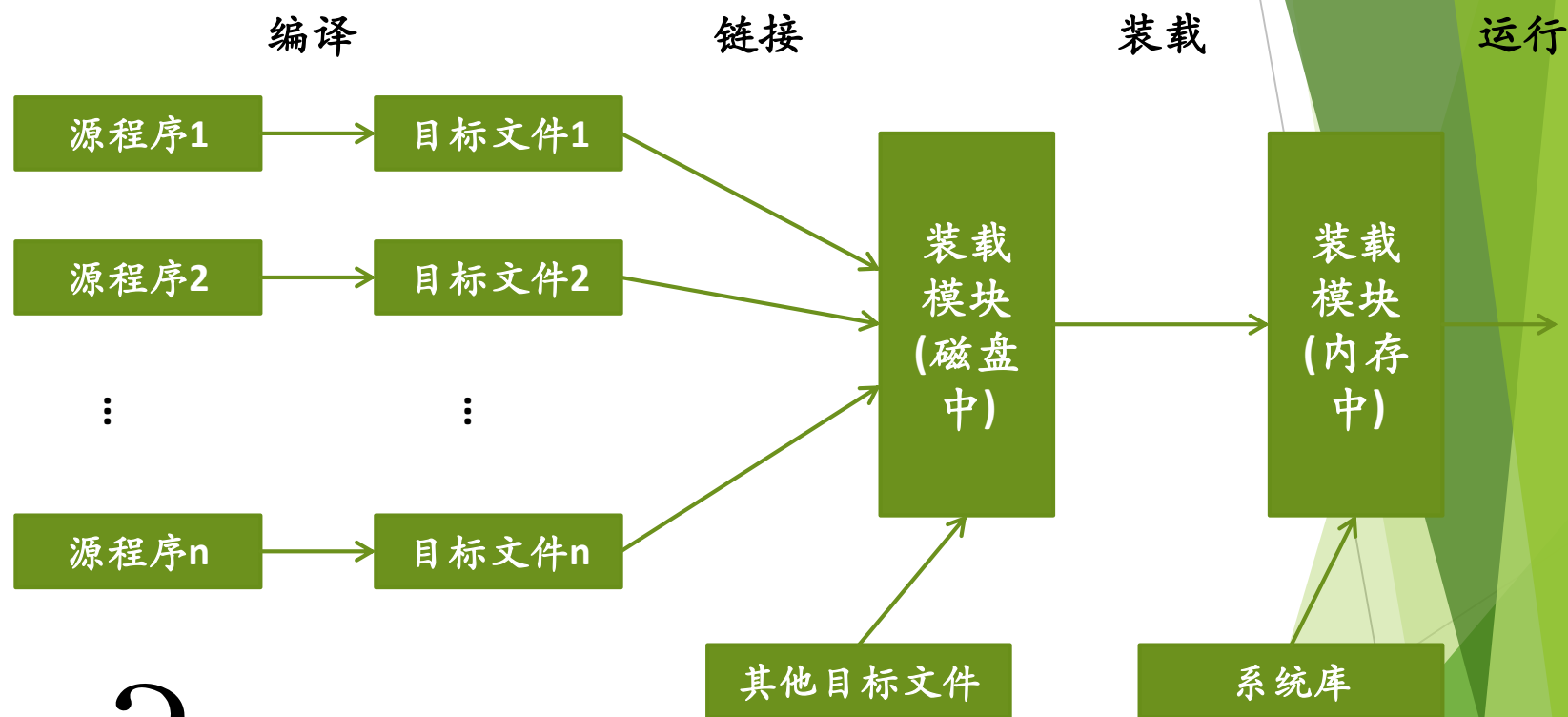
► 地址空间

有什么内容？在哪里？具体的位置？

► 活跃内容 or 工作集 or 常驻集

► 空闲空间组织、分配（放置）、回收、分割、合并

程序执行前的准备过程



?

何时将指令、数据绑定到内存地址

地址重定位

► 逻辑地址（相对地址，虚拟地址）

用户程序经过编译、汇编后形成目标代码，目标代码通常采用相对地址的形式，其首地址为0，其余指令中的地址都相对于首地址而编址

不能用逻辑地址在内存中读取信息

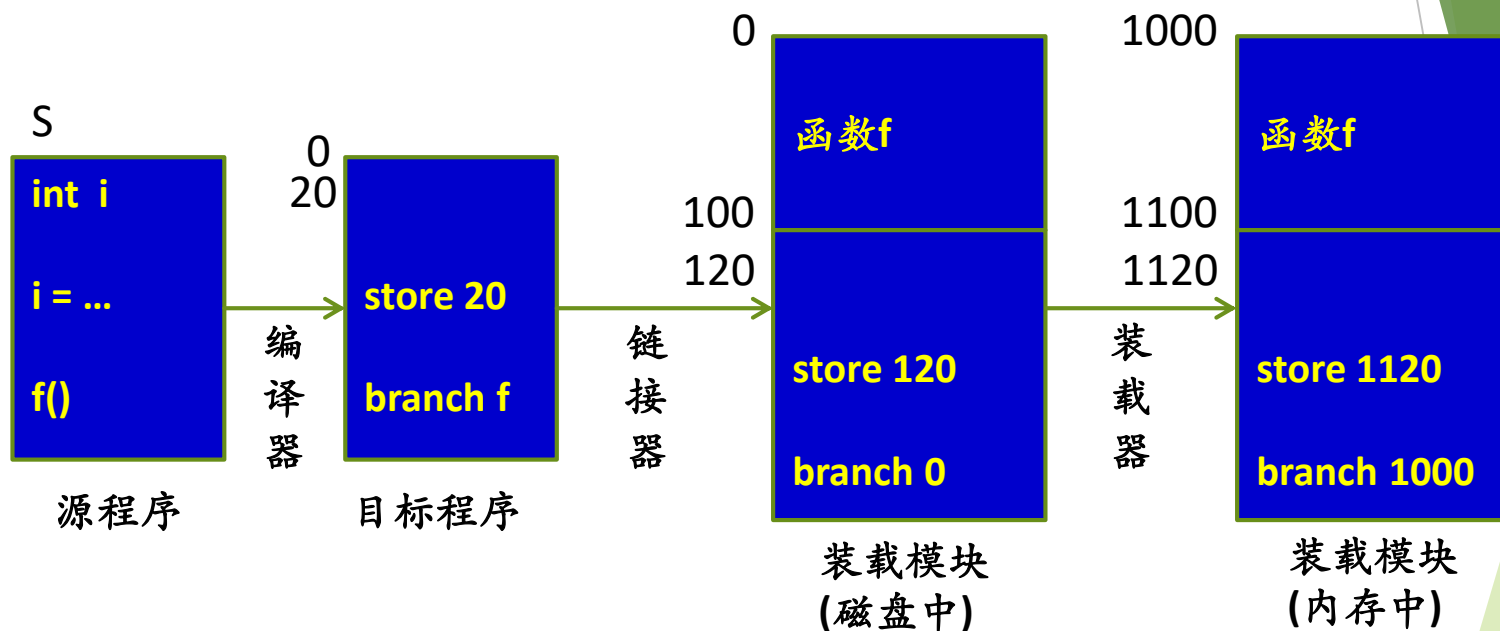
► 物理地址（绝对地址，实地址）

内存中存储单元的地址，可直接寻址

地址重定位：将用户程序中的逻辑地址转换为运行时可由机器直接寻址的物理地址的过程

目的：保证CPU执行指令时可正确访问内存单元

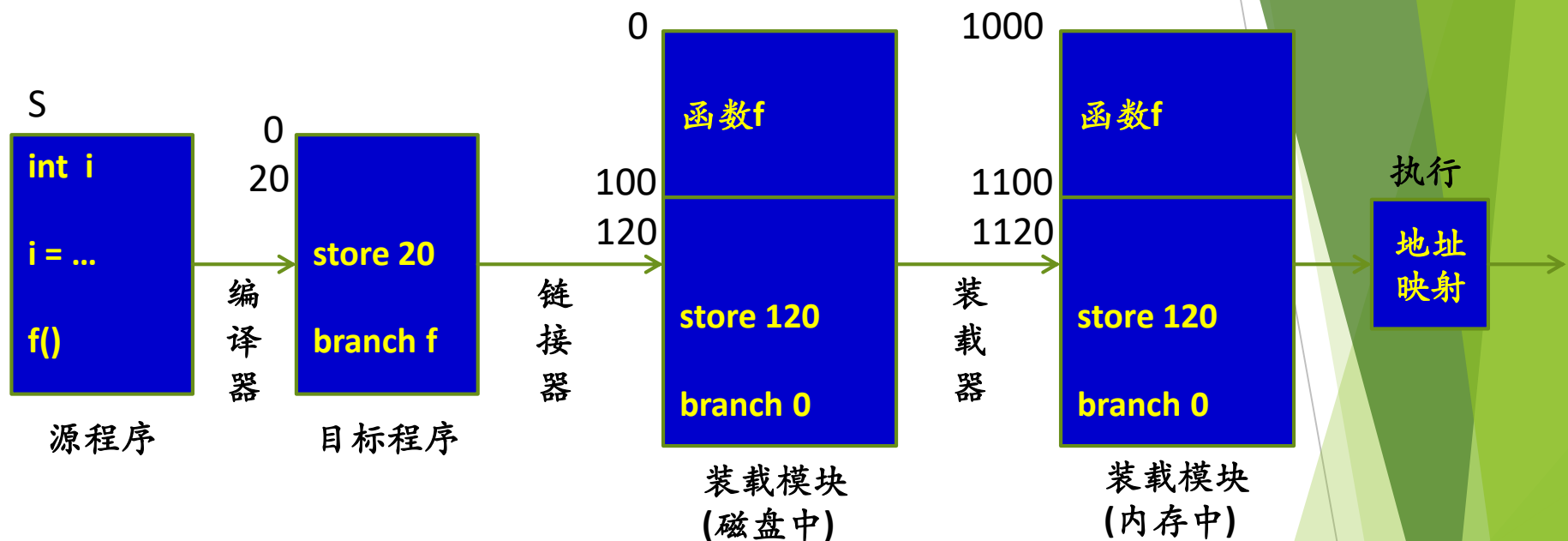
静态地址重定位



当用户程序加载到内存时，一次性实现逻辑地址到物理地址的转换

■ 一般可以由软件完成

动态地址重定位

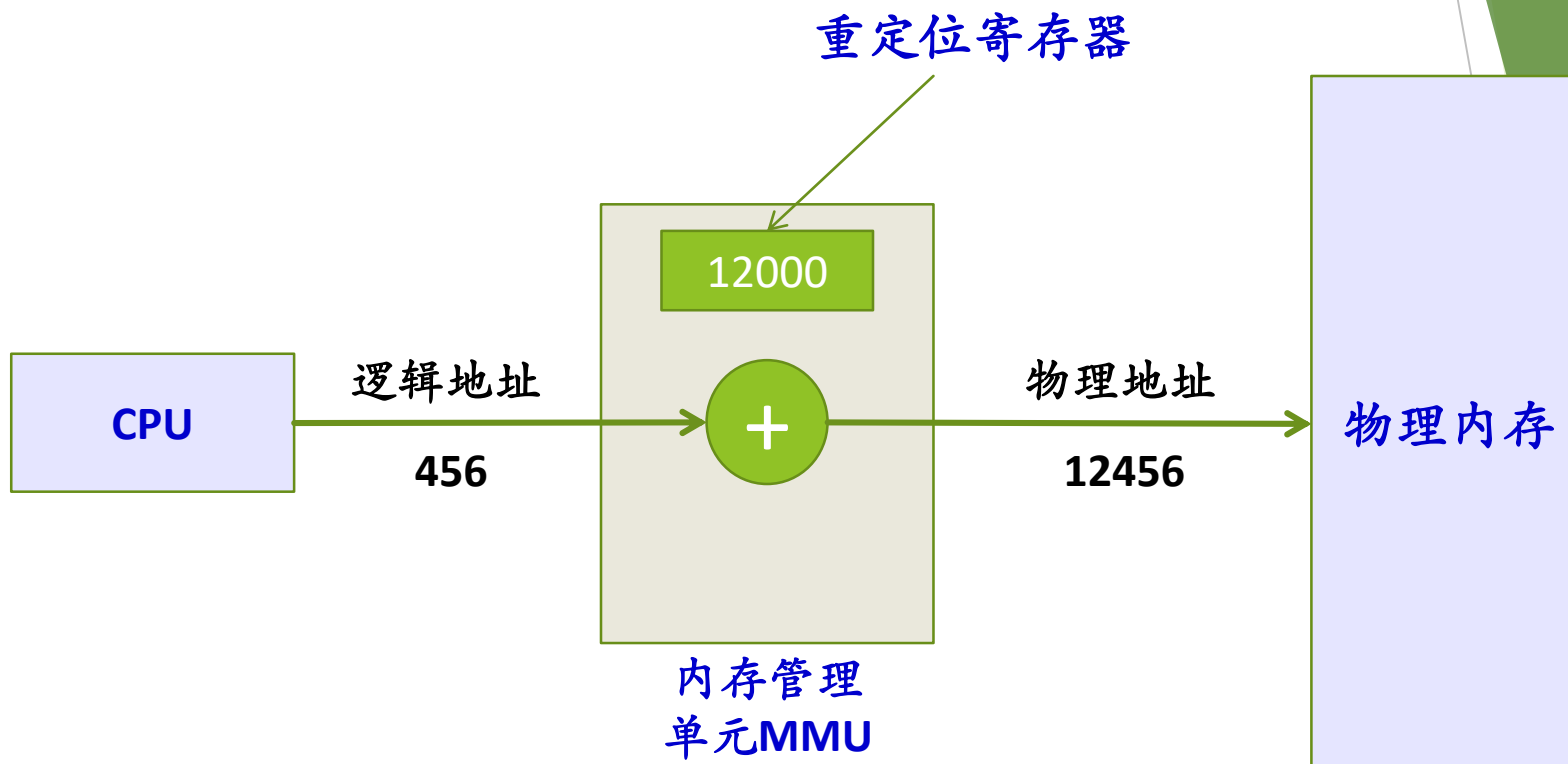


在进程执行过程中进行地址变换

→ 即逐条指令执行时完成地址映射

■ 需要硬件支持?

动态重定位实现——MMU

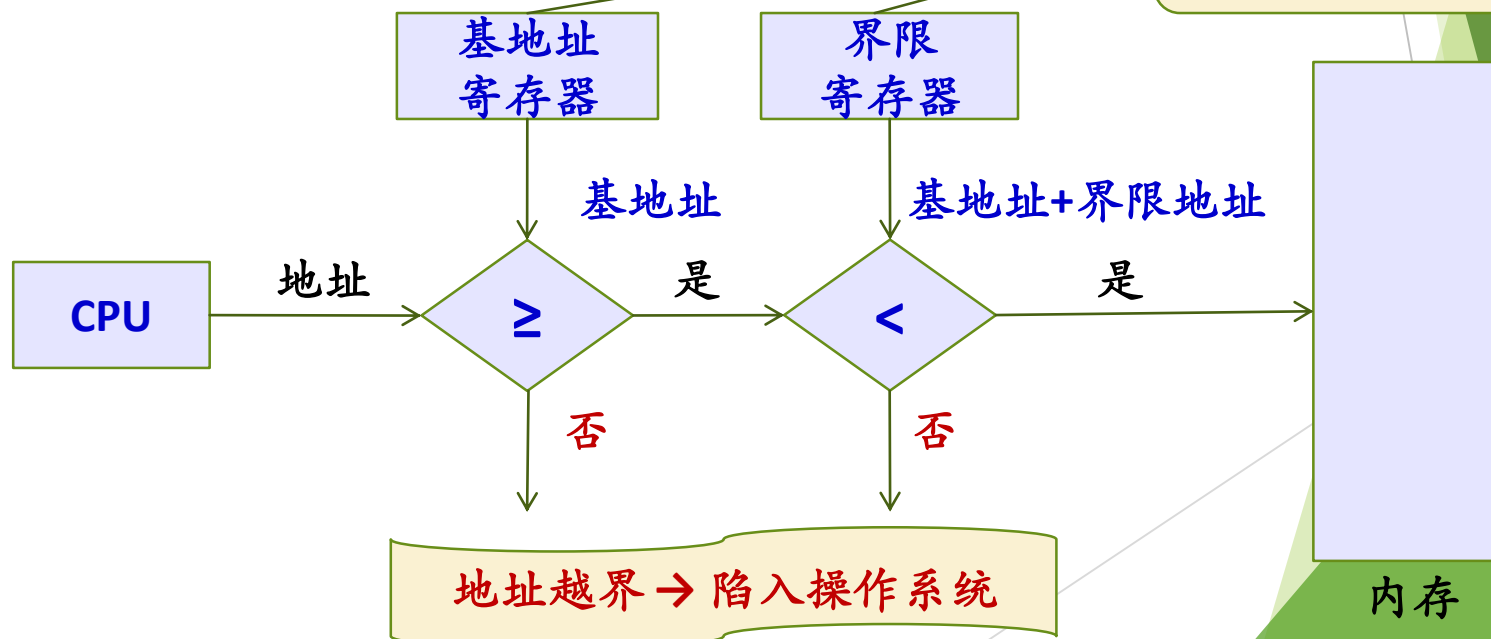


地址保护

防止地址越界

- 确保每个进程可访问的合法地址的范围，以确保进程只访问其合法地址

操作系统通过特殊的特权指令加载



存储管理的基本目标

- ▶ 给进程分配内存——地址空间
- ▶ 往内存加载内容——映射进程地址空间到物理内存
- ▶ 存储保护——地址越界、权限
- ▶ 管理共享的内存
- ▶ 最小化存储访问时间

单一连续区、固定分区、可变分区、分页、分段、段页式

基本内存管理方案

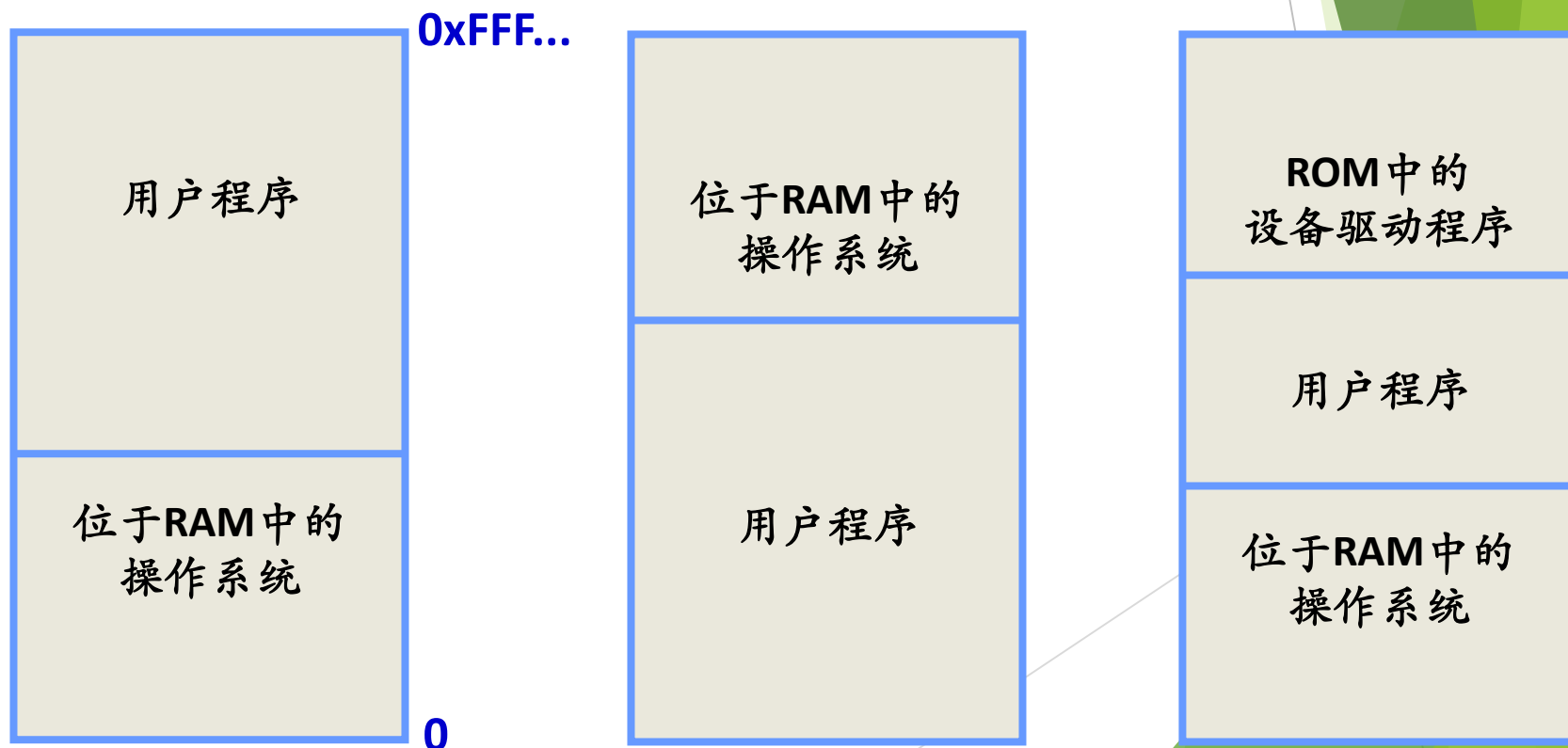
加载单位：进程

内存管理基本方案

单一连续区	每次只运行一个用户程序，用户程序独占内存，它总是被加载到同一个内存地址上
固定分区	把可分配的内存空间分割成若干个连续区域，每一区域称为分区。每个分区的大小可以相同也可以不同，分区大小固定不变，每个分区装一个且只能装一个进程
可变分区	根据进程的需求，把可分配的内存空间分割出一个分区，分配给该进程
页式	把用户程序地址空间划分成大小相等的部分，称为页。内存空间按页的大小划分为大小相等的区域，称为内存块（物理页面，页框，页帧）。以页为单位进行分配，逻辑上相邻的页，物理上不一定相邻
段式	用户程序地址空间按进程自身的逻辑关系划分为若干段，内存空间被动态的划分为若干个长度不相同的区域（可变分区）。以段为单位分配内存，每一段在内存中占据连续空间，各段之间可以不连续存放
段页式	用户程序地址空间：段式；内存空间：页式；分配单位：页

1. 单一用户(连续区)

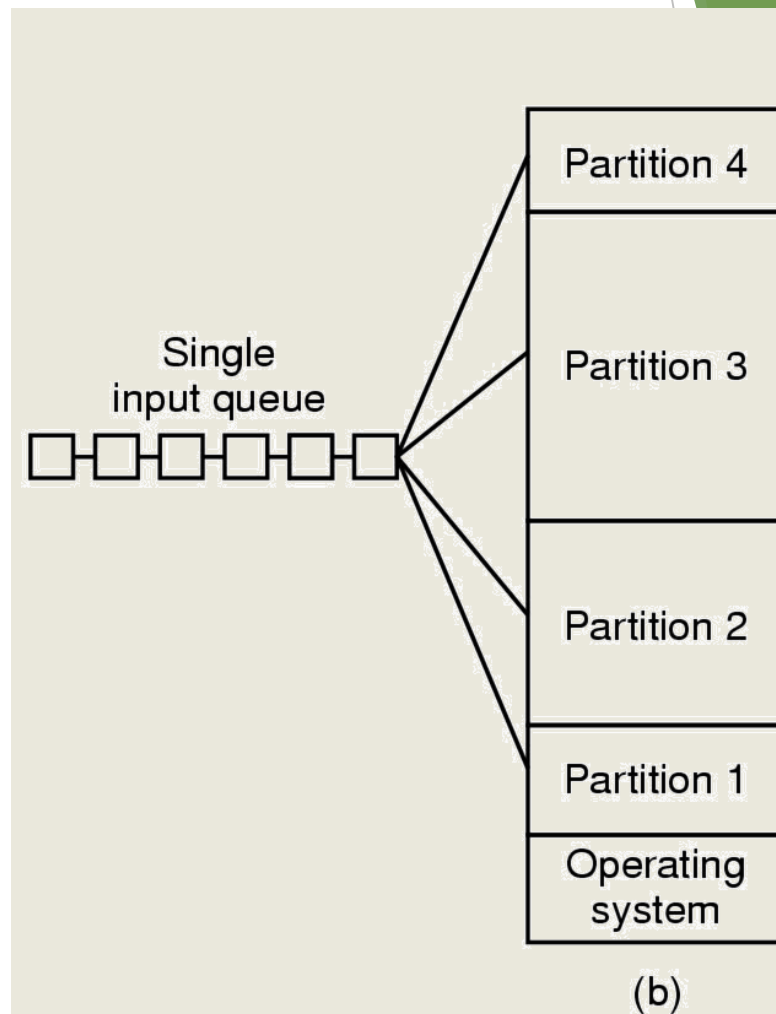
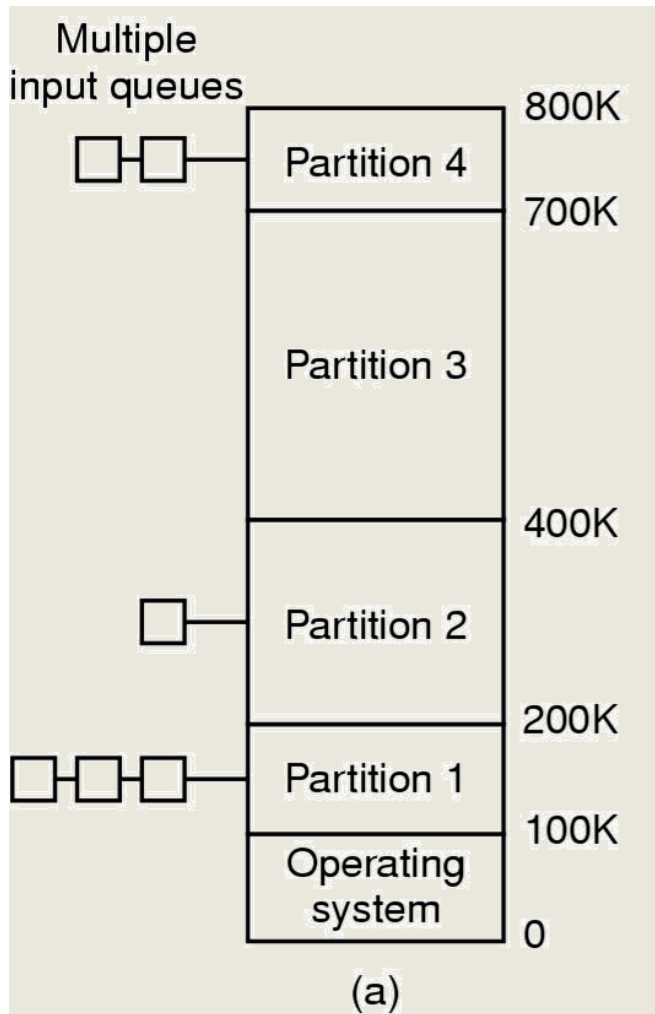
特点：一段时间内只有一个进程在内存
简单，内存利用率低



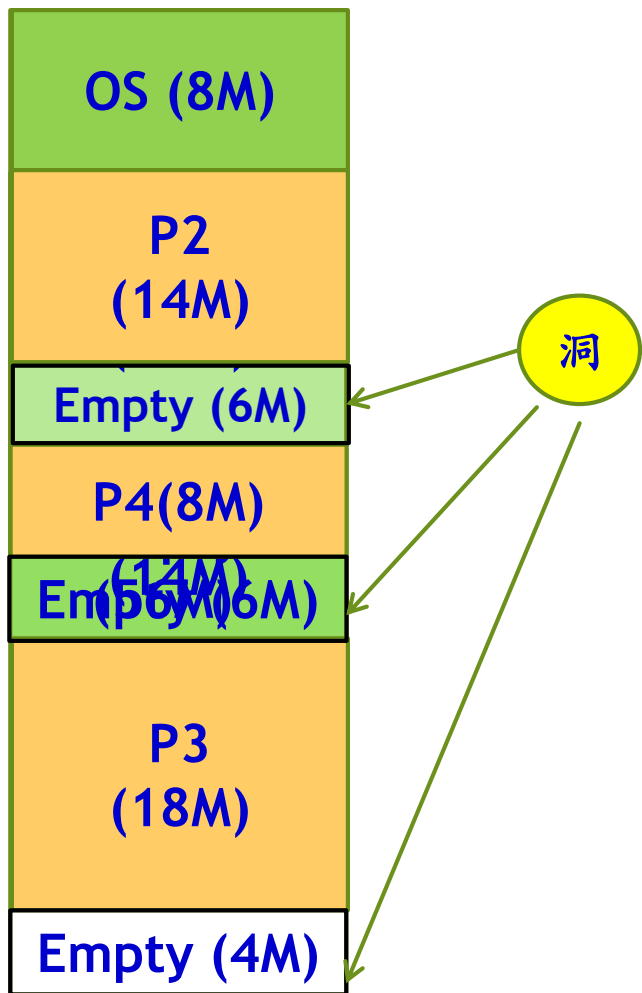
2. 固定分区

- ▶ 把内存空间分割成若干区域，称为分区
- ▶ 每个分区的大小可以相同也可以不同
- ▶ 分区大小固定不变
- ▶ 每个分区装一个且只能装一个进程

固定分区示例



3. 可变分区



- ▶ 根据进程的需要，把内存空闲空间分割出一个分区，分配给该进程
- ▶ 剩余部分成为新的空闲区



问题及解决方案

- ▶ **碎片** → 很小的、不易利用的空闲区
→ 导致内存利用率下降
- ▶ **解决方案** → **紧缩技术 (memory compaction)**
在内存移动程序，将所有小的空闲区合并为较大的空闲区
又称：压缩技术，紧致技术，搬家技术
- ▶ **紧缩时要考虑的问题**
系统开销？ 移动时机？

4. 页式存储管理

划分是由系统自动完成的，对用户是透明的

► 设计思想

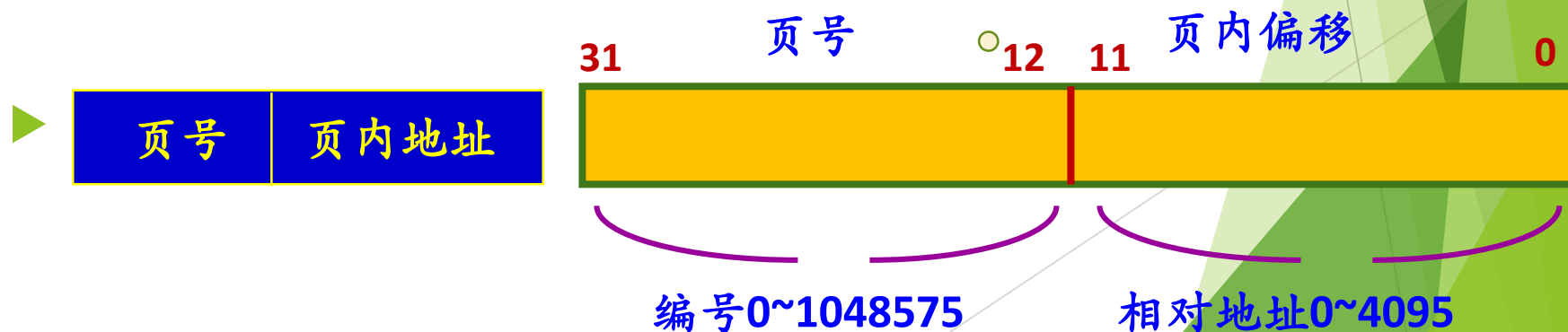
✓ 用户程序地址空间划分

划分为大小相等的区域—页 page

✓ 内存空间按页大小划分为大小相等的区域，称为内存块（物理页面，页框，页帧）page frame

✓ 内存分配规则：以页为单位进行分配，并按进程需要的页数来分配；逻辑上相邻的页，物理上不一定相邻

► 典型的页面尺寸：4K或4M

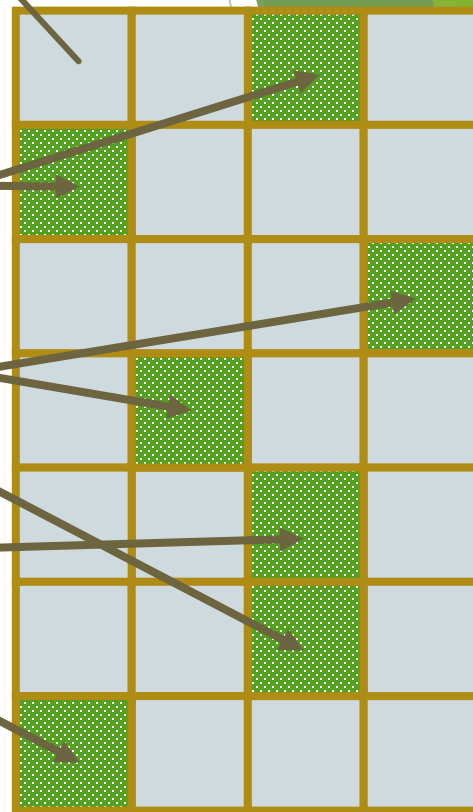


内存分配

页框 page frame
(页帧、物理页面、
内存块)

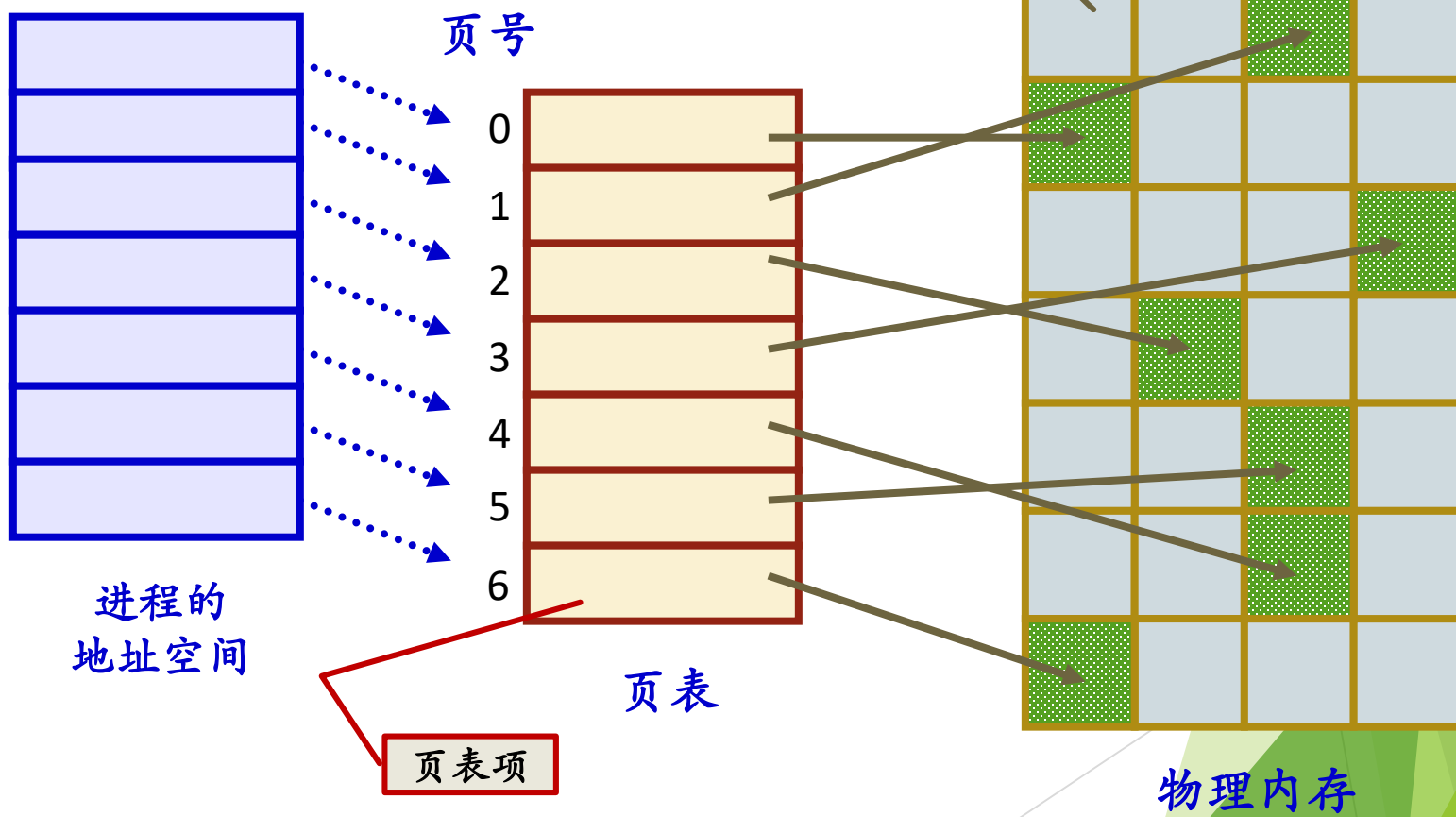


进程的
地址空间



物理内存

内存分配



相关数据结构及地址转换

► 页表

- **页表项**：记录了逻辑页号与页框号的对应关系
- 每个进程一个页表，存放在内存
- 页表起始地址保存在何处？

► 空闲内存管理

► 地址转换（硬件支持）



CPU取到逻辑地址，自动划分为页号和页内地址；用页号查页表，得到页框号，再与页内地址（页内偏移）拼接为物理地址

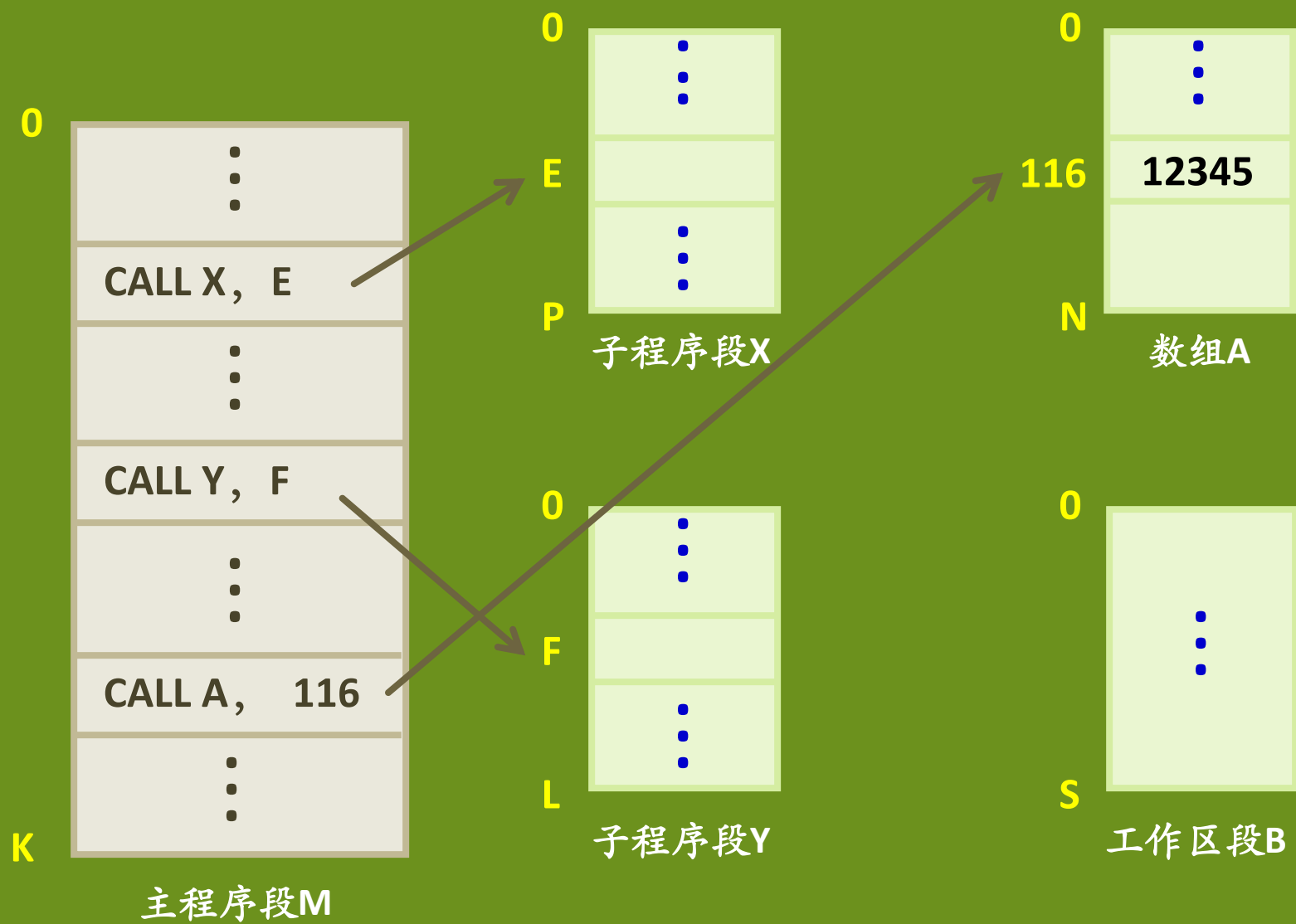
5. 段式存储管理

► 设计思想

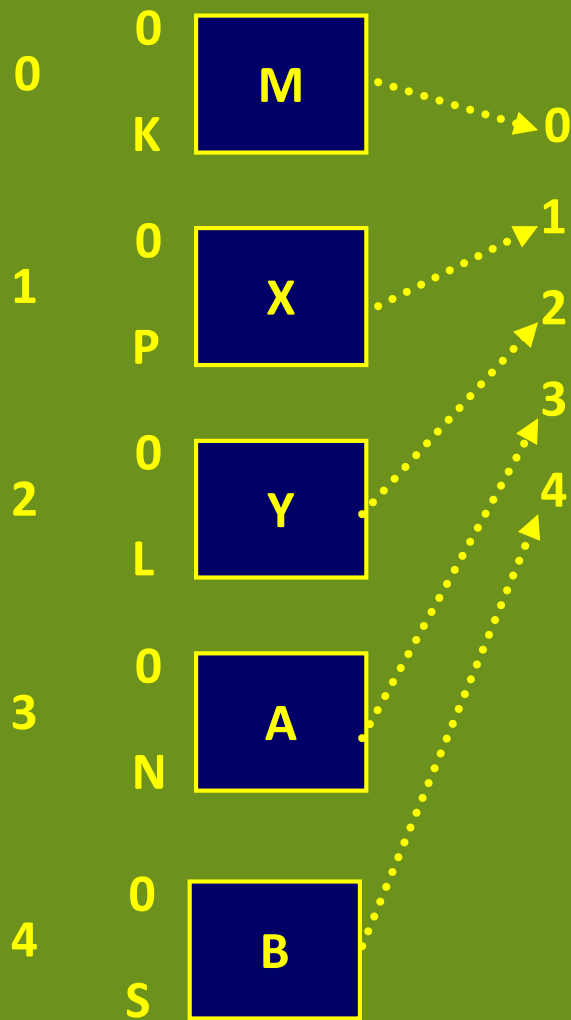
- 用户程序地址空间：按程序自身的逻辑关系划分为若干个程序段，每个程序段都有一个段名
- 内存空间被动态的划分为若干个长度不相同的区域，称为物理段，每个物理段由起始地址和长度确定
- 内存分配规则：以段为单位进行分配，每一个段在内存中占据连续空间，但各段之间可以不相邻

► 逻辑地址

段号	段内地址
----	------



逻辑段号



进程1的地址空间

长度 段地址

K	3200
P	1500
L	6000
N	8000
S	5000

段表

1000

3200

5000

6000

8000

操作系统

P

K

S

L

N

内存

相关数据结构及地址转换

► 段表

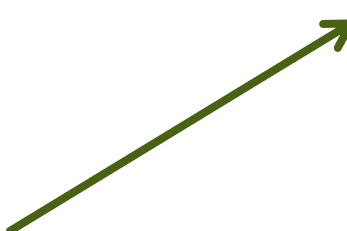
- 记录了段号、段首地址和段长度之间的关系
- 每个进程一个段表，存放在内存
- 段表起始地址保存在何处？

► 物理内存管理

- 同可变分区

► 地址转换（硬件）

CPU取到逻辑地址，用段号查段表，得到段的起始地址，再与段内偏移地址计算出物理地址



段号	段内地址
----	------

6. 段页式存储管理方案

► 产生背景

结合页式段式优点，克服二者的缺点

► 设计思想

用户程序划分：按段式划分（对用户来讲，按段的逻辑关系进行划分；对系统讲，按页划分每一段）

逻辑地址：

段号	段内地址	
	页号	页内地址

内存划分：按页式存储管理方案

内存分配：以页为单位进行分配

段页式存储管理

► 数据结构及有关操作

- 段表：记录了每一段的页表始址和页表长度
- 页表：记录了逻辑页号与内存块号的对应关系（每一段有一个，一个程序可能有多个页表）
- 空闲区管理：同页式管理
- 分配、回收：同页式管理

► 地址转换

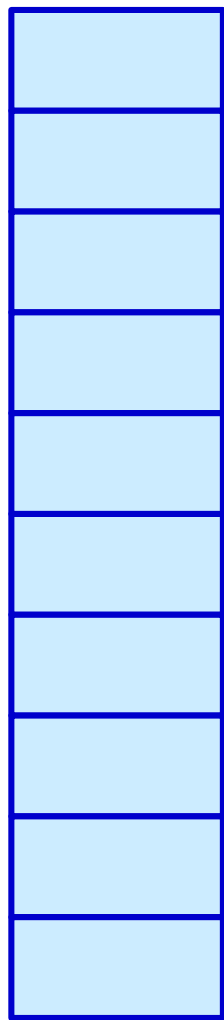
段号	段内地址	
	页号	页内地址

位图法、空闲区表、空闲区链表

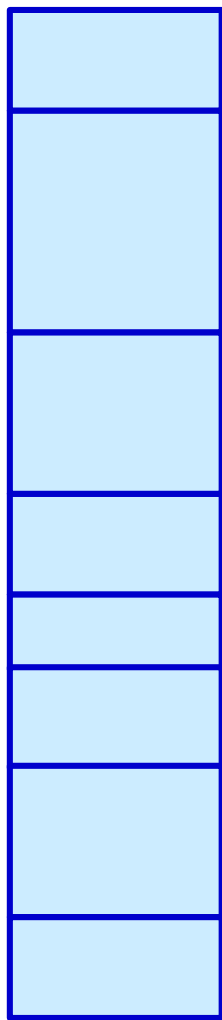
首先适配、最佳适配、最差适配

物理内存管理方案

空闲物理内存管理



等长划分



不等长划分

► 数据结构

► 位图

每个分配单元对应于位图中的一位，0表示空闲，1表示占用（或者相反）

► 空闲区表、已分配区表

表中每一项记录了空闲区（或已分配区）的起始地址、长度、标志

► 空闲块链表

内存分配算法

将该空闲区分为两部分，一部分供进程使用，另一部分形成新的空闲区

► 首次适配 first fit

在空闲区表中找到第一个满足进程要求的空闲区

► 下次适配 next fit

从上次找到的空闲区处接着查找

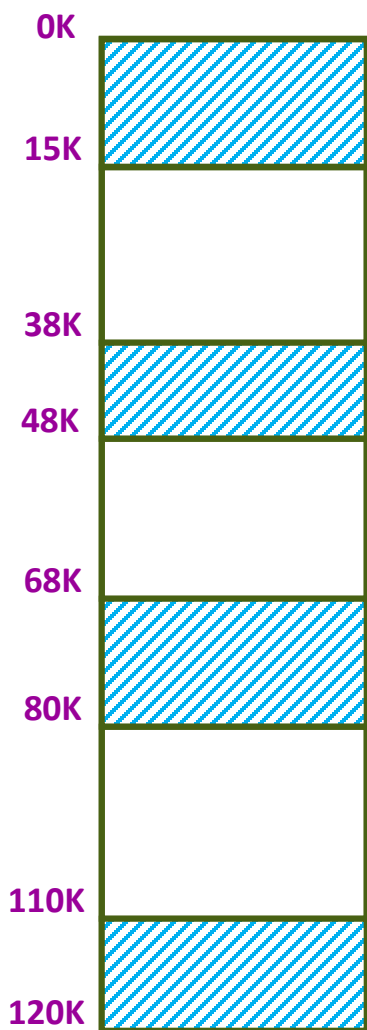
► 最佳适配 best fit

查找整个空闲区表，找到能够满足进程要求的最小空闲区

► 最差适配 worst fit

总是分配满足进程要求的最大空闲区

示例



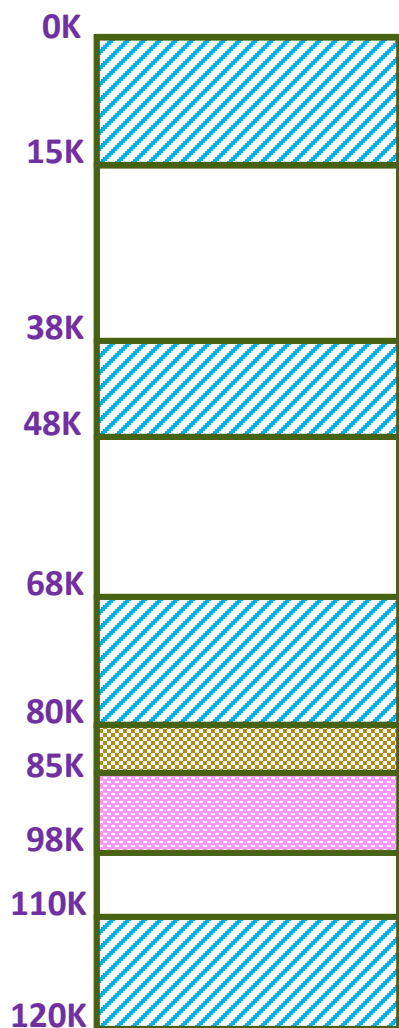
空闲区表

始址	长度	标志
15K	23K	未分配
48K	20K	未分配
80K	30K	未分配
		空项
		空项

已分配区表

始址	长度	标志
0K	15K	P ₁
38K	10K	P ₂
68K	12K	P ₃
110K	10K	P ₄
		空项
		空项

示例



空闲区表

始址	长度	标志
15K	23K	未分配
48K	20K	未分配
98K	12K	未分配
		空项
		空项

已分配区表

始址	长度	标志
0K	15K	P ₁
38K	10K	P ₂
68K	12K	P ₃
110K	10K	P ₄
80K	5K	P ₅
85K	13K	P ₆

回收问题

► 内存回收算法

► 当某一块归还后，前后空闲空间合并，修改内存空闲区表

► 四种情况

上相邻、下相邻、上下都相邻、上下都不相邻

伙伴系统

Linux底层内存管理采用

一种特殊的
“分离适配”
算法

- ▶ 一种经典的内存分配方案
- ▶ 主要思想：将内存按2的幂进行划分，组成若干空闲块链表；查找该链表找到能满足进程需求的最佳匹配块
- ▶ 算法：
 - ▶ 首先将整个可用空间看作一块： 2^U
 - ▶ 假设进程申请的空间大小为 s ，如果满足
$$2^{U-1} < s \leq 2^U$$
，则分配整个块
 - 否则，将块划分为两个大小相等的伙伴，大小为 2^{U-1}
 - ▶ 一直划分下去直到产生大于或等于 s 的最小块

伙伴系统的例子

1 MB 内存

1M

A 申请 100K

A=128K

128k

256K

512K

B 申请 240K

A=128K

128k

B=256K

512K

C 申请 64K

A=128K

C=64K

64K

B=256K

512K

D 申请 256K

A=128K

C=64K

64K

B=256K

D=256K

256K

释放 B

A=128K

C=64K

64K

256K

D=256K

256K

释放 A

128K

C=64K

64K

256K

D=256K

256K

E 申请 75K

E=128K

C=64K

64K

256K

D=256K

256K

释放 C

E=128K

128k

256K

D=256K

256K

释放 E

512K

D=256K

256K

释放 D

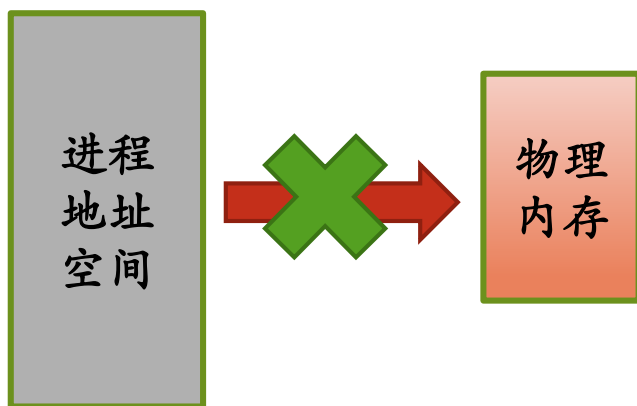
1M

覆盖技术、交换技术、.....

内存“扩充”

内存不足时如何管理?

- ▶ 内存紧凑（例如：可变分区）
- ▶ 覆盖技术
- ▶ 交换技术
- ▶ 虚存技术



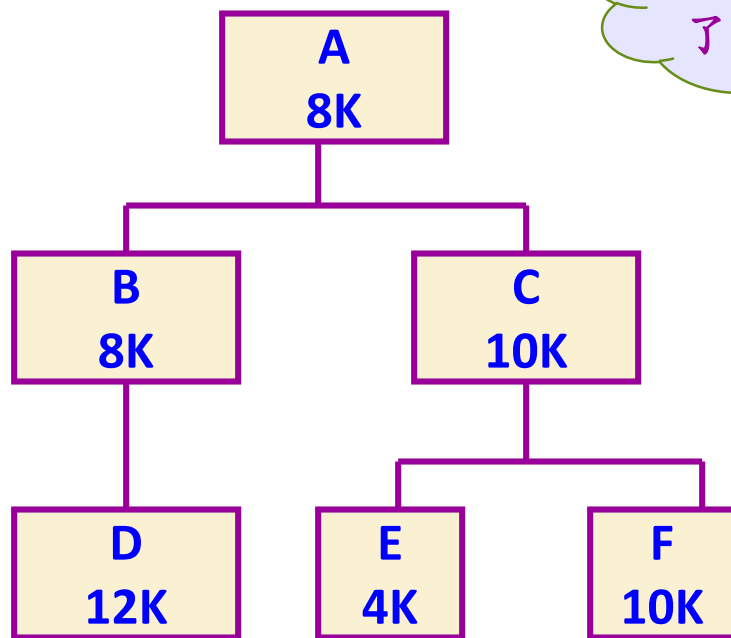
目标：解决在较小的存储空间中运行较大程序时遇到的矛盾

覆盖技术 (Overlaying)

- ▶ 解决的问题 → 程序大小超过物理内存总和
- ▶ 程序执行过程中，程序的不同部分在内存中相互替代
 - 按照其自身的逻辑结构将那些不会同时执行的程序段共享同一块内存区域
 - 要求程序各模块之间有明确的调用结构
- ▶ 程序员声明覆盖结构，操作系统完成自动覆盖

主要用在早期的操作系统

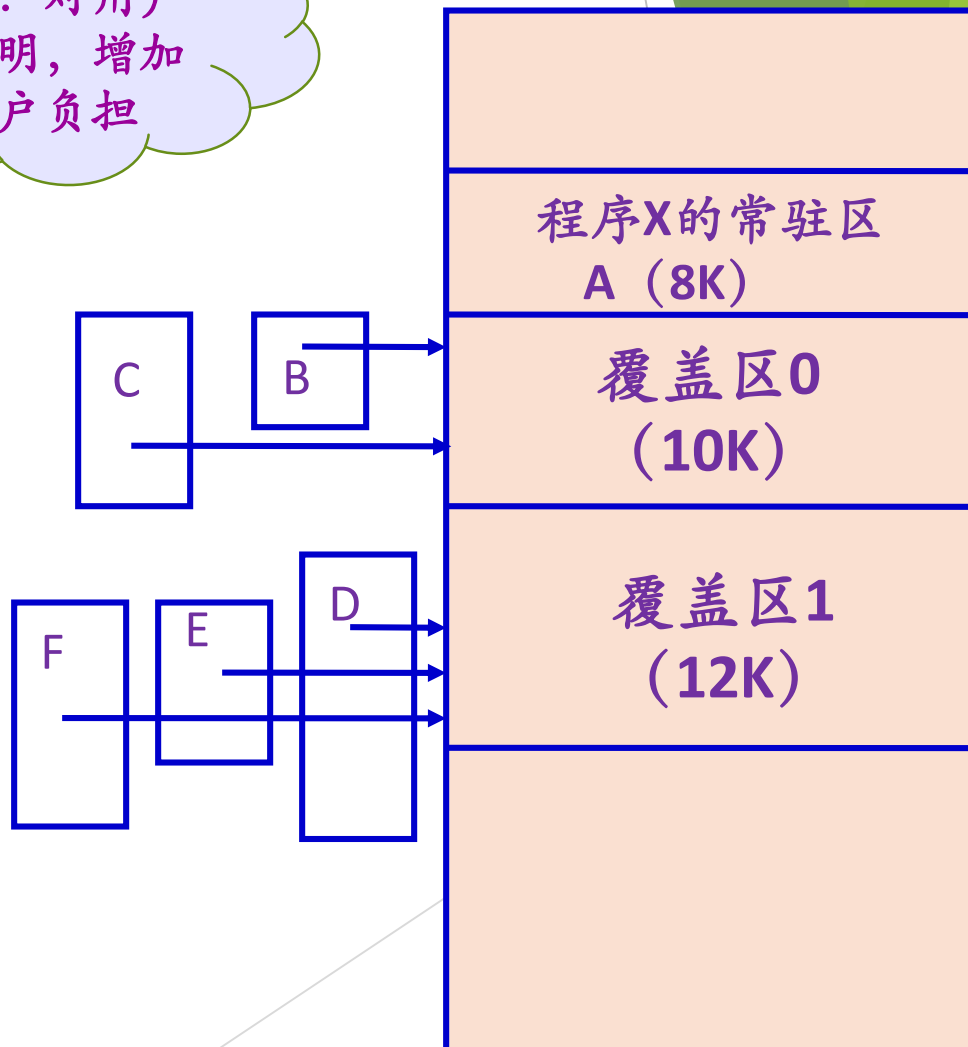
覆盖技术示例



程序X的调用结构：54K

缺点：对用户不透明，增加了用户负担

内存占用：30K



覆盖技术的不足

- ▶ 增加编程困难

- ▶ 需程序员划分功能模块，并确定模块间的覆盖关系
 - ▶ 增加了编程的复杂度

- ▶ 增加执行时间

- ▶ 从外存装入覆盖模块
 - ▶ 时间换空间

交换技术 (Swapping)

- ▶ 最早用于小型分时系统

roll in roll out

- ▶ 设计思想

内存空间紧张时，系统将内存中某些进程暂时移到外存，把外存中某些进程换进内存，占据前者所占用的区域（进程在内存与外存之间的动态调度）

- ▶ 讨论：实现时遇到的问题

- ✓ 进程的什么部分需要交换到磁盘？在磁盘的什么位置保存被换出的进程？
- ✓ 交换时机？
- ✓ 如何选择被换出的进程？
- ✓ 换出后再换入的进程是否回到原处？
- ✓ 如何处理进程空间增长？

问题讨论

swapper

- ▶ 运行时创建或修改的内容：栈和堆
- ▶ 交换区：一般系统会指定一块特殊的磁盘区域作为交换空间（swap space），包含连续的磁道，操作系统可以使用底层的磁盘读写操作对其高效访问
- ▶ 何时需发生交换？

只要不用就换出（很少再用）；内存空间不够或有不够的危险时换出

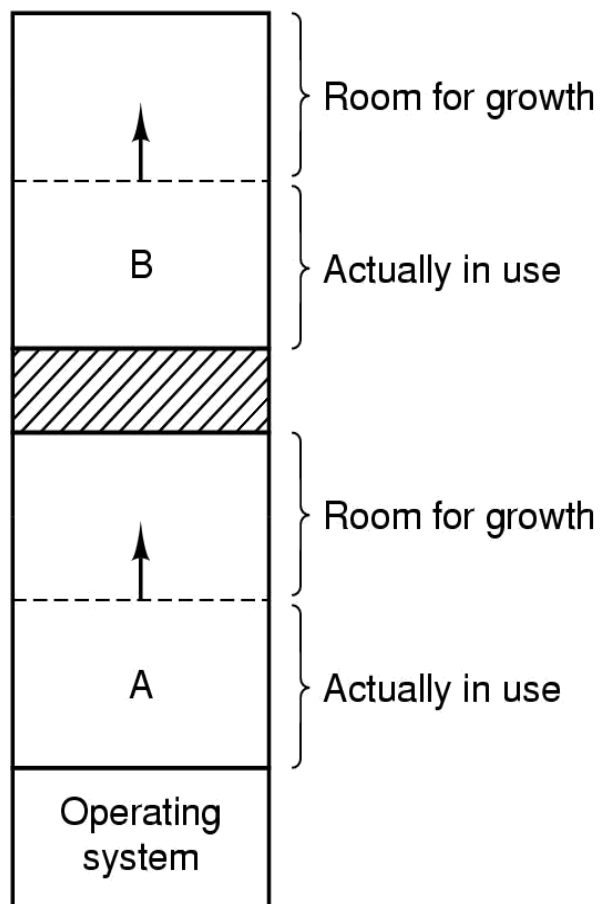
→ 与调度器结合使用

- ▶ 考虑进程的各种属性；不应换出处于等待I/O状态的进程
- ▶ 换出后又换入的进程不一定回到原处（采用动态重定位）

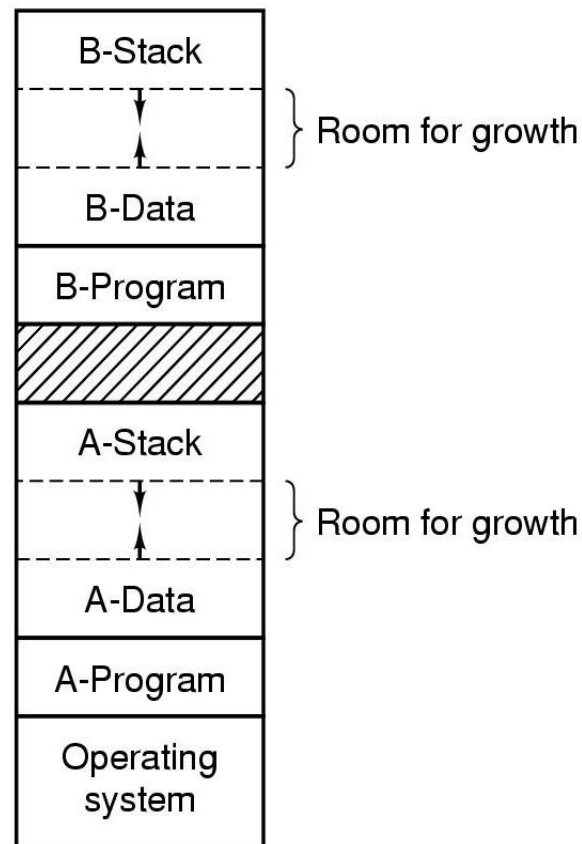
进程空间增长的困难及解决

► 数据段

► 栈段



(a)



(b)

重点小结

► 基本概念

- 存储体系
- 交换与覆盖技术
- 逻辑地址、物理地址、地址重定位
- 地址保护

► 各种存储管理方案

- 单一连续区、固定分区、可变分区、分页、分段、段页式
- 相关数据结构
- 地址转换过程

► 物理内存管理

- 位示图、空闲区表/已分配区表、空闲块链表、伙伴系统
- 分配与回收算法
- 碎片问题（内碎片、外碎片）

作业6

- 1、阅读Three easy pieces的15.1-15.3，概括总结主要思想。
- 2、总结交换技术的主要思想。

作业提交时间：
2020年11月29日 晚23:30

The background features abstract, overlapping green geometric shapes, primarily triangles and polygons, in various shades of green, creating a modern and dynamic look.

Thanks

The End