计算机考研系列书课包

玩转操作系统

主讲人 刘财政

第四讲　内存管理

本讲内容

考点一: 内存管理概述 ★

考点二: 内存连续分配管理方式 ★★★

考点三: 基本分页内存管理方式 ★★★★★

考点四: 虚拟存储器 ★★★★★

考点五: 带快表的两级系统的地址翻译过程 ★★★★★

考点一：

内存管理概述

考点框架

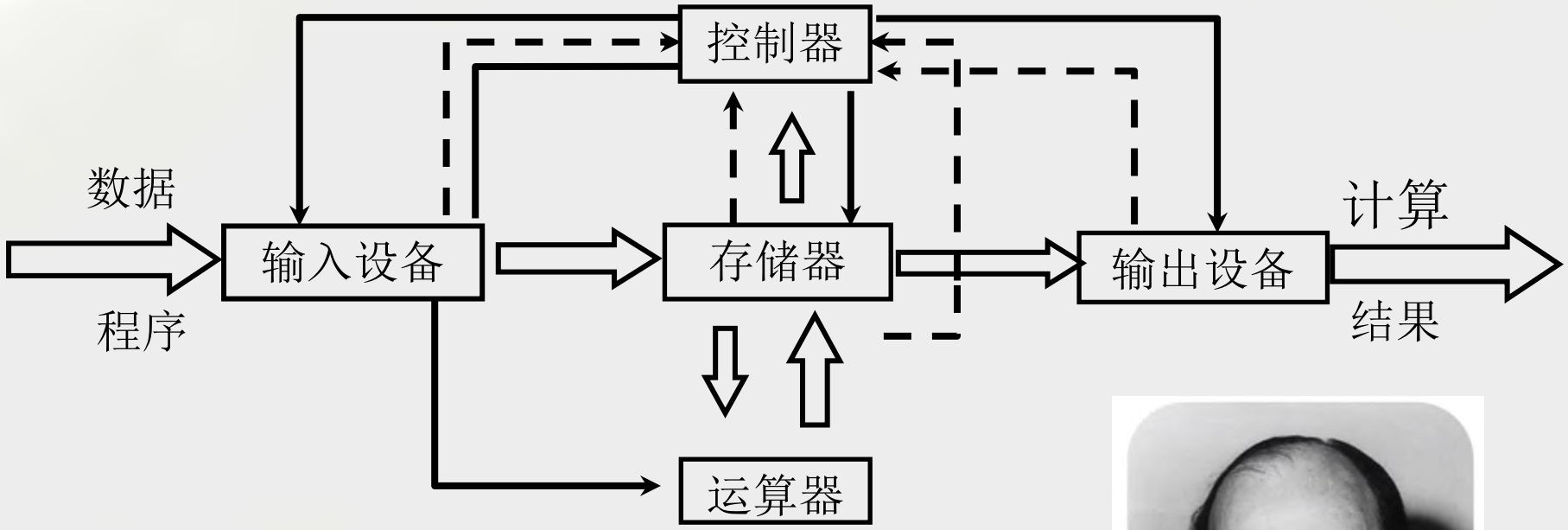
- 内存管理的概念
- 内存管理的基本功能
- 程序的装入
- 程序的链接
- 程序的运行过程

考点一: 内存管理概述

内存管理的概念

考点一: 内存管理概述

冯·诺依曼计算机



- 存储程序思想：指令和数据以同等地位存于存储器；指令和数据用二进制表示
- 存储器用来存放数据和程序

冯.诺依曼

考点一: 内存管理概述

内存的组织结构

按编号存取包裹

考点一: 内存管理概述

内存的组织结构

主存储器的编址单元是字节

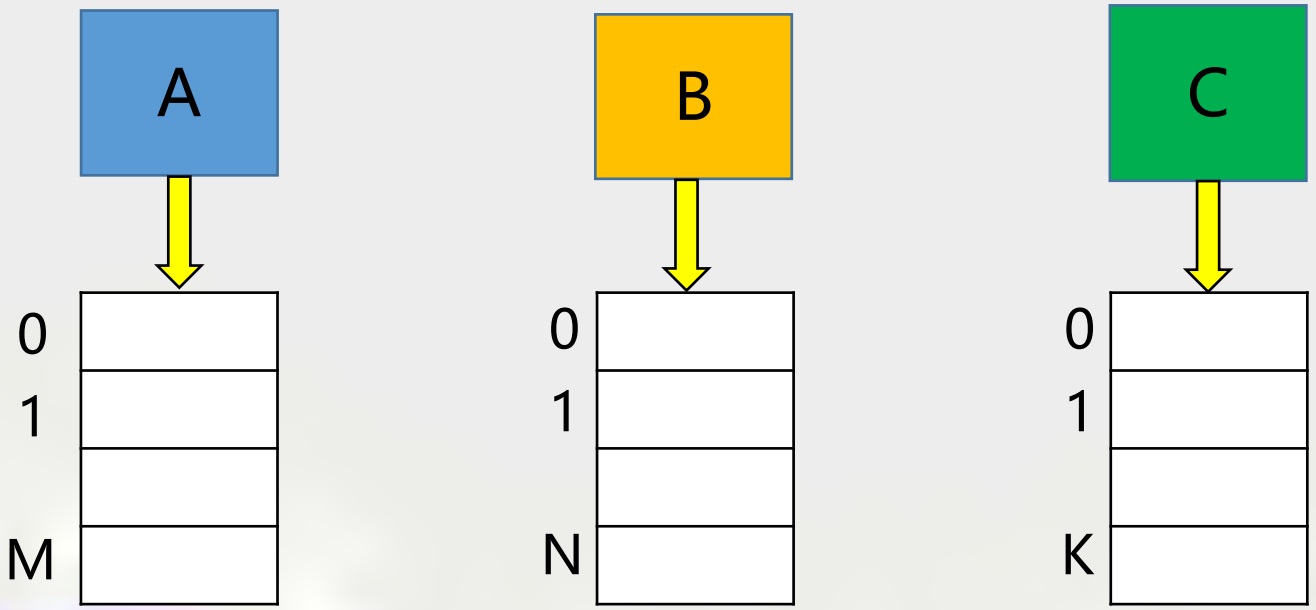
0	1	2	3	4	5	6	7
8	9	10	11	12	13	14	15
16	17	18	19	20	21	22	23
24	25	26	27	28	29	30	31
32	33	34	35	36	37	38	39
40	41	42	43	44	45	46	47

内存地址是也叫物理地址

考点一: 内存管理概述

逻辑地址（logical Address： LA）

- 源程序经过汇编或编译后，形成目标代码，每个目标代码都是以0为基址顺序进行编址的，原来用符号名访问的单元用具体的数据——单元号取代。
- 这样生成的目标程序占据一定的地址空间，称为作业的逻辑地址空间，简称逻辑空间。



考点一: 内存管理概述

物理地址(physical address： PA)

- 物理内存，真实存在的插在主板内存槽上的内存条的容量的大小.内存是由若干个存储单元组成的，每个存储单元有一个编号，可唯一标识一个存储单元，称为内存地址（或物理地址）。

0	1	2	3	4	5	6	7
8	9	10	11	12	13	14	15
16	17	18	19	20	21	22	23
24	25	26	27	28	29	30	31
32	33	34	35	36	37	38	39
40	41	42	43	44	45	46	47

考点一: 内存管理概述

物理地址(physical address： PA)

- 把内存看成一个从0字节一直到内存最大容量逐字节编号的存储单元数组，即每个存储单元与内存地址的编号相对应。

0	1	2	3	4	5	6	7
8	9	10	11	12	13	14	15
16	17	18	19	20	21	22	23
24	25	26	27	28	29	30	31
32	33	34	35	36	37	38	39
40	41	42	43	44	45	46	47

考点一: 内存管理概述

- 外部碎片，是由于大量信息由于先后写入、置换、删除而形成的空间碎片。

- 由于这样的原因形成的空间碎片，我们称之为外部碎片。

考点一: 内存管理概述

- 外部碎片，是由于大量信息由于先后写入、置换、删除而形成的空间碎片。

- 由于这样的原因形成的空间碎片，我们称之为外部碎片。

考点一: 内存管理概述

- 内部碎片，是由于存量信息容量与最小存储空间单位不完全相符而造成的空间碎片。

考点一: 内存管理概述

内存管理的基本功能

考点一: 内存管理概述

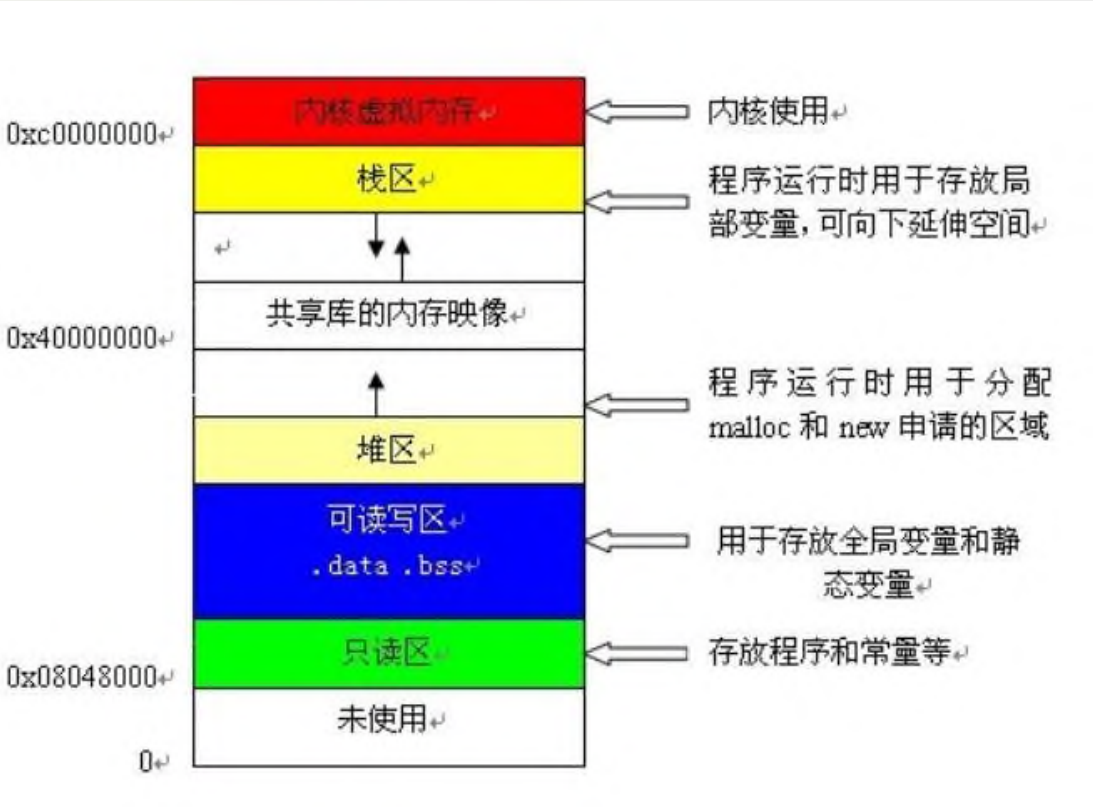
内存的基本功能

- 内存分配和回收
- 地址转换
- 内存空间的扩充
- 存储保护

考点一: 内存管理概述

内存的基本功能

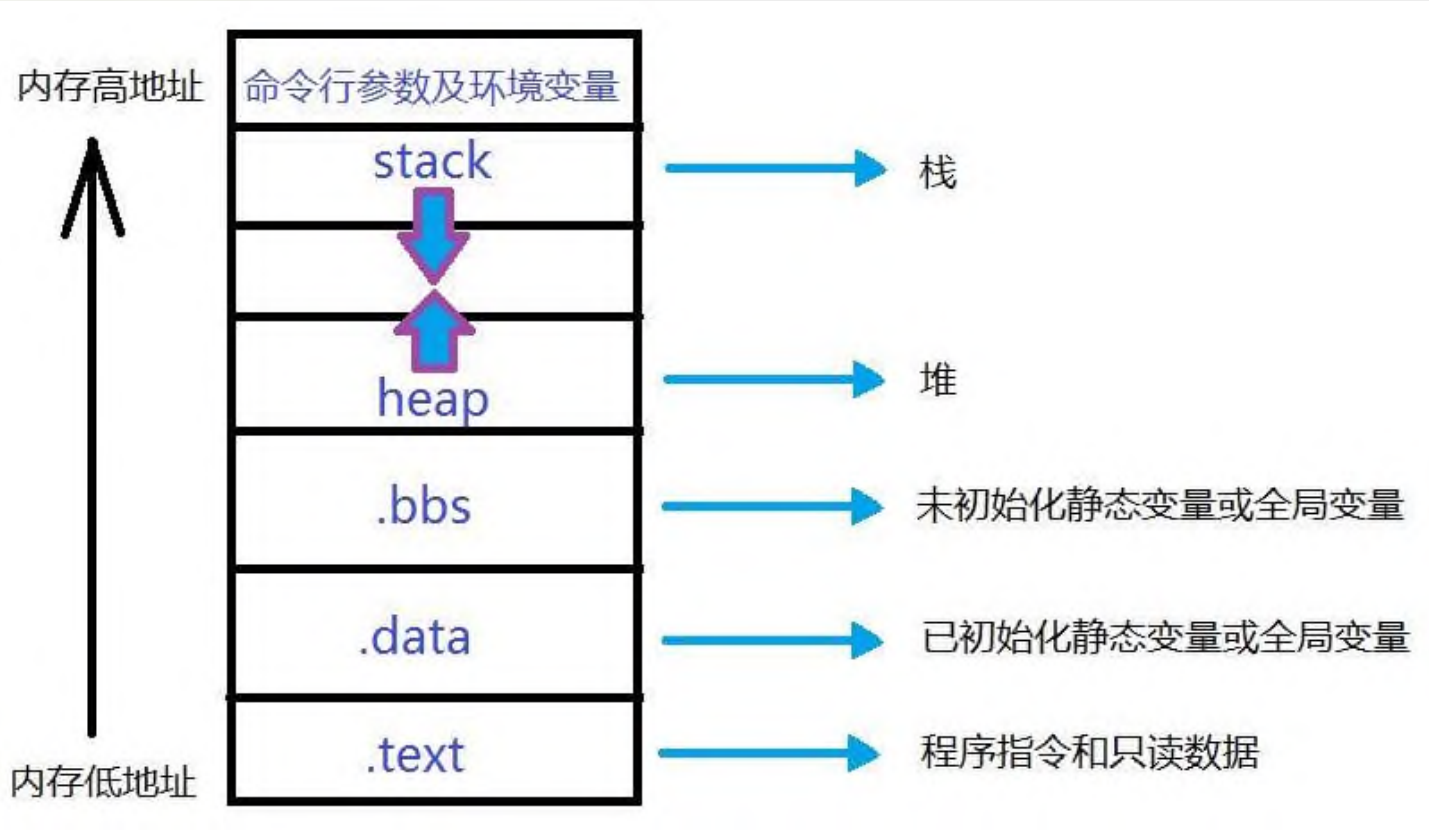
内存分配和回收



考点一: 内存管理概述

内存的基本功能

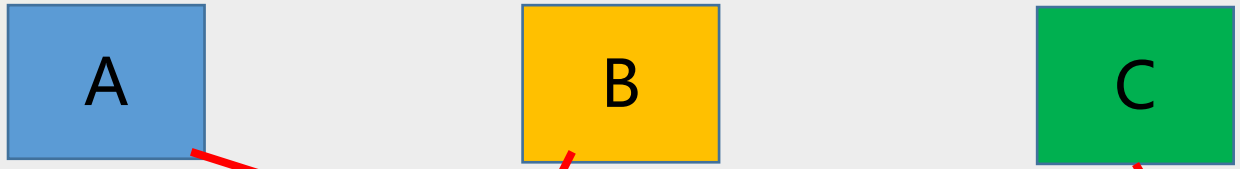
- 内存分配和回收



考点一: 内存管理概述

内存的基本功能

- 地址转换:　逻辑地址 → 物理地址

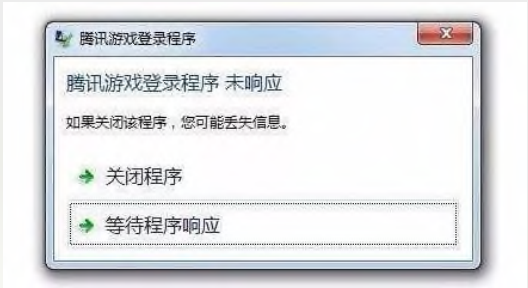


0	1	2	3	4	5	6	7
8	9	10	11	12	13	14	15
16	17	18	19	20	21	22	23
24	25	26	27	28	29	30	31
32	33	34	35	36	37	38	39
40	41	42	43	44	45	46	47

考点一: 内存管理概述

内存的基本功能

- 内存空间的扩充



考点一: 内存管理概述

内存的基本功能

存储保护

内存分配前，需要保护操作系统不受用户进程的影响，同时保护用户进程不受其他用户进程的影响。

进程1

进程2

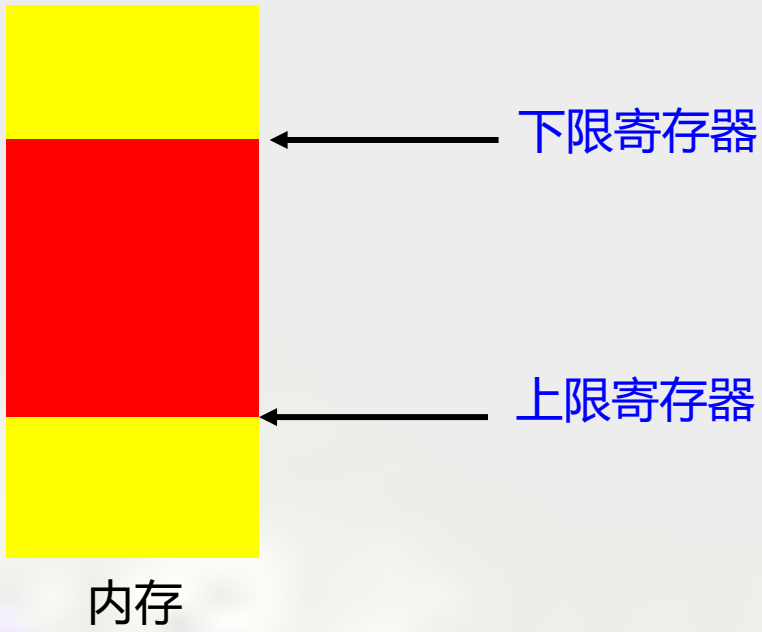
鸡犬之声相闻，
老死不相往来

考点一: 内存管理概述

内存的基本功能

- 存储保护

➢ 在CPU中设置一对下限寄存器和上限寄存器,存放正在执行的程序在主存中的下限和上限地址。

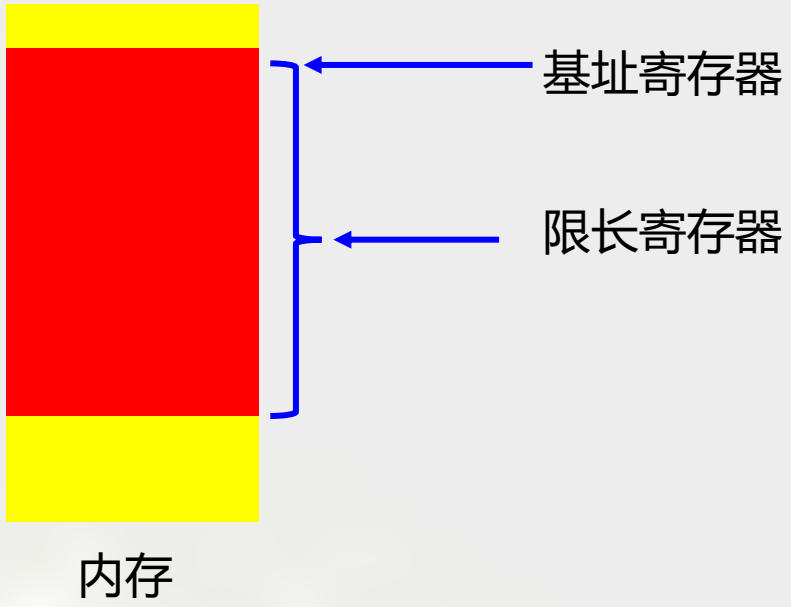


考点一: 内存管理概述

内存的基本功能

存储保护

- 可将一个寄存器作为基址寄存器，另一寄存器作为限长寄存器（指示存储区长度）



考点一: 内存管理概述

内存的基本功能

存储保护

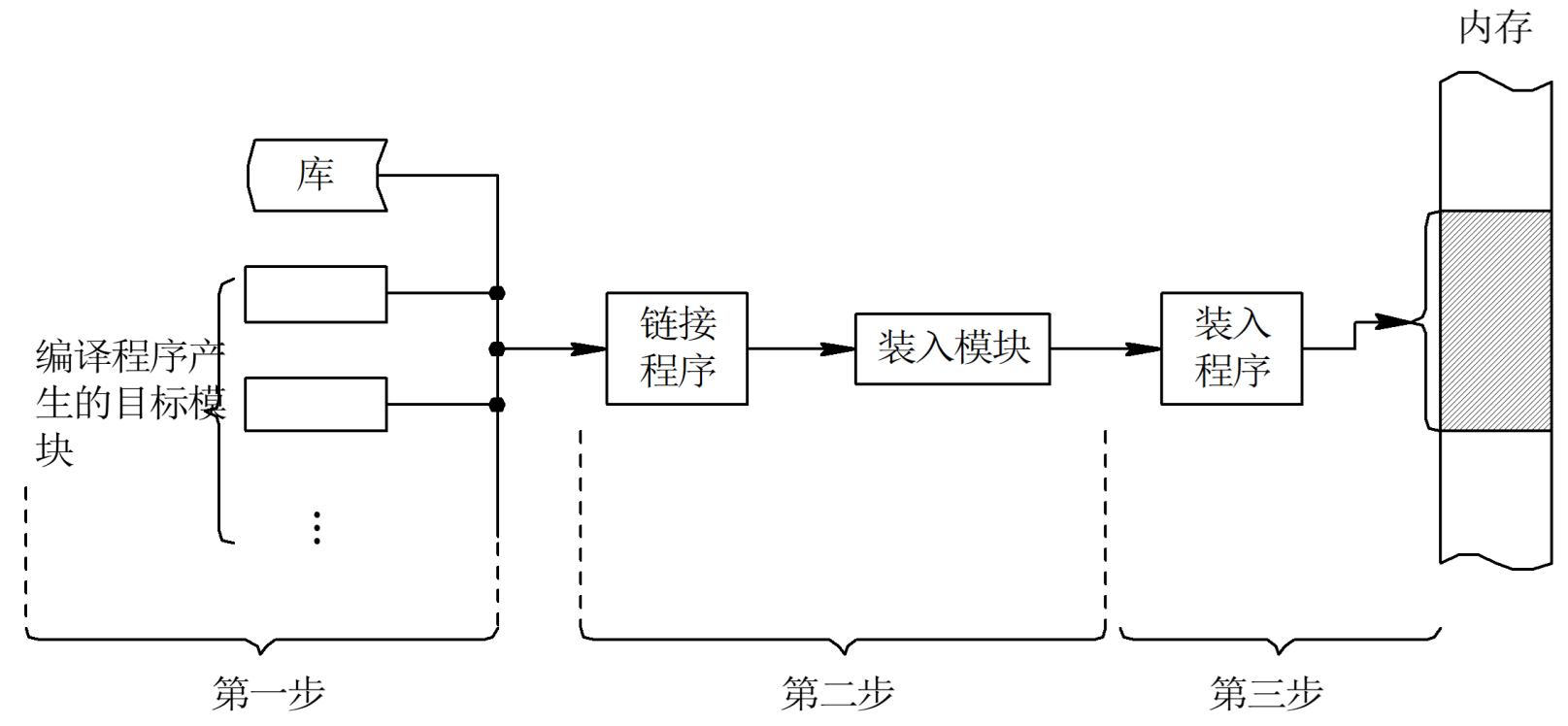
- 每当CPU要访问主存，硬件自动将被访问的主存地址与界限寄存器的内容进行比较，以判断是否越界。
- 如果未越界，则按此地址访问主存，否则将产生程序中断——越界中断（存储保护中断）。

考点一: 内存管理概述

程序的运行过程

考点一: 内存管理概述

程序的运行过程



考点一: 内存管理概述

程序的运行过程

程序运行之前必须为其建立进程，而创建进程的首要任务是将程序和数据装入内存。用户源程序执行流程如下：

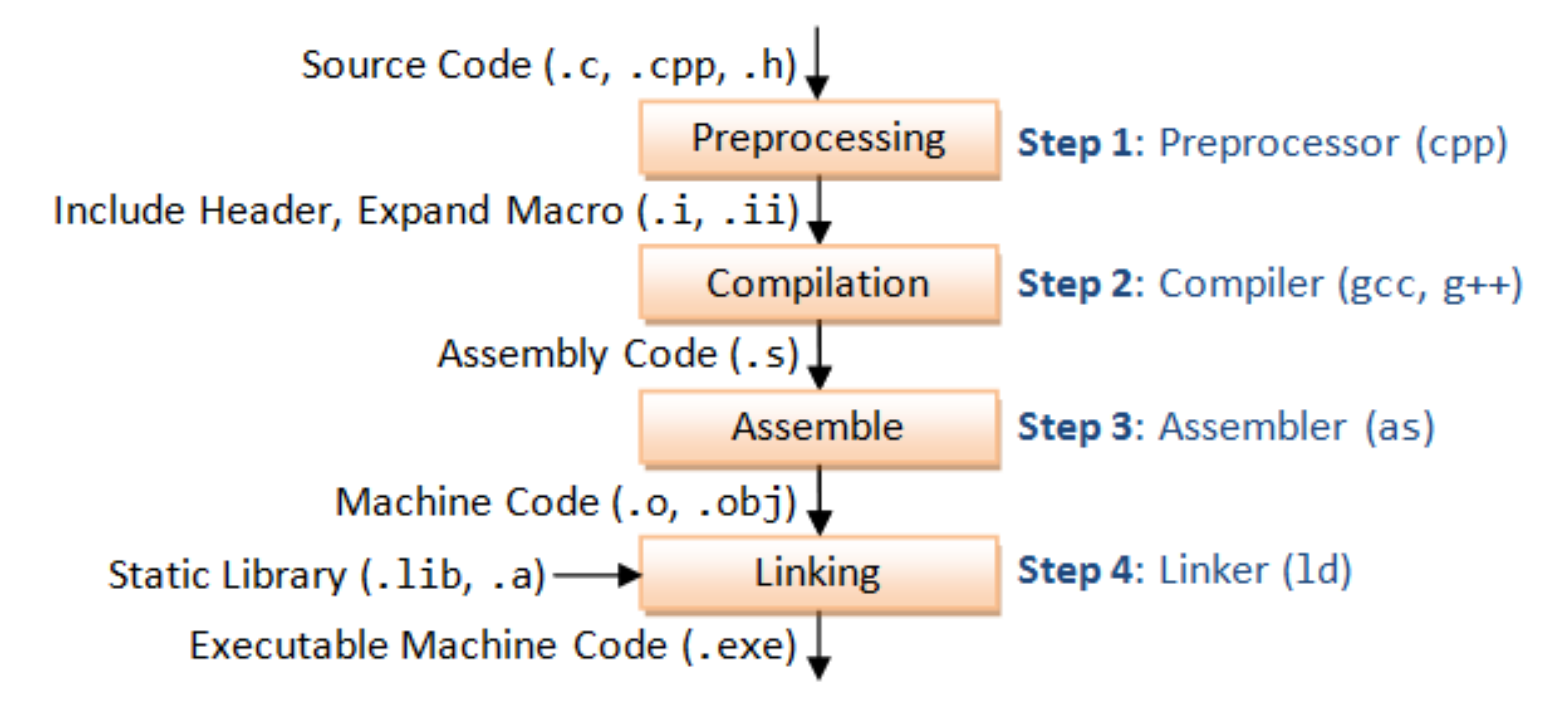
- **编译**：将用户源代码编译为目标代码的过程
- **链接**：将编译后形成的一组目标代码以及所需库函数链接在一起，形成完整的装入模块
- **装入**：将装入模块装入内存
- **执行**：运行内存中的可执行文件

考点一: 内存管理概述

程序的运行过程

过程可以分为4部分内容组成

预处理器->编译器->汇编器->链接器



考点一: 内存管理概述

程序的运行过程

```
├── test.c
└── inc
    ├── mymath.h
    └── mymath.c
```

```c
// test.c
#include <stdio.h>
#include "mymath.h"// 自定义头文件
int main(){
    int a = 2;
    int b = 3;
    int sum = add(a, b);
    printf("a=%d, b=%d, a+b=%d\n", a, b, sum);
}
```

```c
// mymath.h
#ifndef MYMATH_H
#define MYMATH_H
int add(int a, int b);
int sub(int a, int b);
#endif
```

```c
// mymath.c
int add(int a, int b){
        return a+b;
}
int sub(int a, int b){
        return a-b;
}
```

考点一: 内存管理概述

程序的运行过程

预处理器

(1)、处理所有的注释，以空格代替

(2)、讲所有的#define删除，并且展开所有的宏定义

(3)、处理条件编译指令#if，#ifdef、#elif，#else、#endif

(4)、处理#include，展开文件包含

(5)、保留编译器需要使用#pragma指令

考点一: 内存管理概述

程序的运行过程

- 编译： 编译不是指程序从源文件到二进制程序的全部过程，而是指将经过预处理之后的程序转换成特定汇编代码(assembly code)的过程。

```
// test.c汇编之后的结果test.s
    .file   "test.c"
    .section    .rodata
.LC0:
    .string "a=%d, b=%d, a+b=%d\n"
    .text
    .globl  main
    .type   main, @function
main:
.LFB0:
    .cfi_startproc
    pushl   %ebp
    .cfi_def_cfa_offset 8
    .cfi_offset 5, -8
    movl    %esp, %ebp
    .cfi_def_cfa_register 5
    andl    $-16, %esp
    subl    $32, %esp
    movl    $2, 20(%esp)
    movl    $3, 24(%esp)
    movl    24(%esp), %eax
    movl    %eax, 4(%esp)
    movl    20(%esp), %eax
    movl    %eax, (%esp)
    call    add
    movl    %eax, 28(%esp)
    movl    28(%esp), %eax
    movl    %eax, 12(%esp)
    movl    24(%esp), %eax
    movl    %eax, 8(%esp)
    movl    20(%esp), %eax
    movl    %eax, 4(%esp)
    movl    $.LC0, (%esp)
    call    printf
    leave
    .cfi_restore 5
    .cfi_def_cfa 4, 4
    ret
    .cfi_endproc
```

考点一: 内存管理概述

程序的运行过程

- 汇编(Assemble)：汇编过程将上一步的汇编代码转换成机器码(machine code)，这一步产生的文件叫做目标文件，是二进制格式。
- 这一步会为每一个源文件产生一个目标文件。

```
├── test.c
└── inc
    ├── mymath.h
    └── mymath.c
```

```
├── test.o
└── inc
    ├── mymath.o
    └── mymath.o
```

考点一: 内存管理概述

程序的运行过程

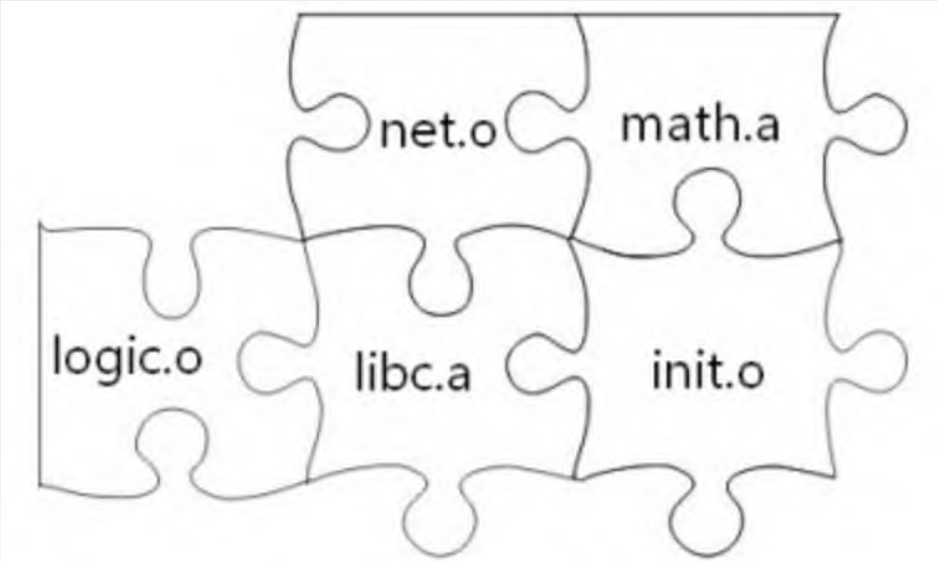
- 链接: 由汇编程序生成的目标文件并不能立即就被执行，其中可能还有许多没有解决的问题．例如，某个源文件中的函数可能引用了另一个源文件中定义的某个符号(如变量或者函数调用等);在程序中可能调用了某个库文件中的函数，等等。所有的这些问题，都需要经链接程序的处理方能得以解决。

```
├── test.o
└── inc
    ├── mymath.o
    └── mymath.o
```

考点一: 内存管理概述

程序的运行过程

- 链接：链接程序的主要工作就是将有关的目标文件彼此相连接，也即将在一个文件中引用的符号同该符号在另外一个文件中的定义连接起来，使得所有的这些目标文件成为一个能够让操作系统装入执行的统一整体。



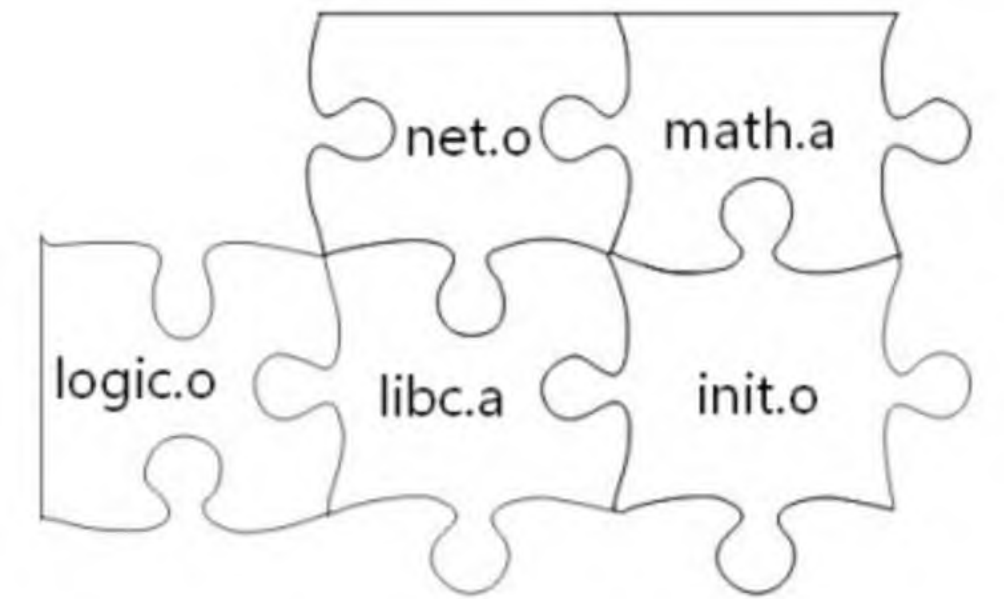
考点一: 内存管理概述

程序的链接

考点一: 内存管理概述

链接方式

- 链接程序的主要工作就是将有关的目标文件彼此相连接，也即将在一个文件中引用的符号同该符号在另外一个文件中的定义连接起来，使得所有的这些目标文件成为一个能够诶操作系统装入执行的统一整体。



考点一: 内存管理概述

- **静态链接：**程序执行前，将若干从 0 地址开始的目标模块及所需库函数链接为完整、从唯一“0”地址开始的装配模块

```
// test.c
#include <stdio.h>
#include "mymath.h “
int main(){
   return 0;
}
```

```
int sub(int a, int b){
        return a-b;
}

int add(int a, int b){
        return a+b;
}
```

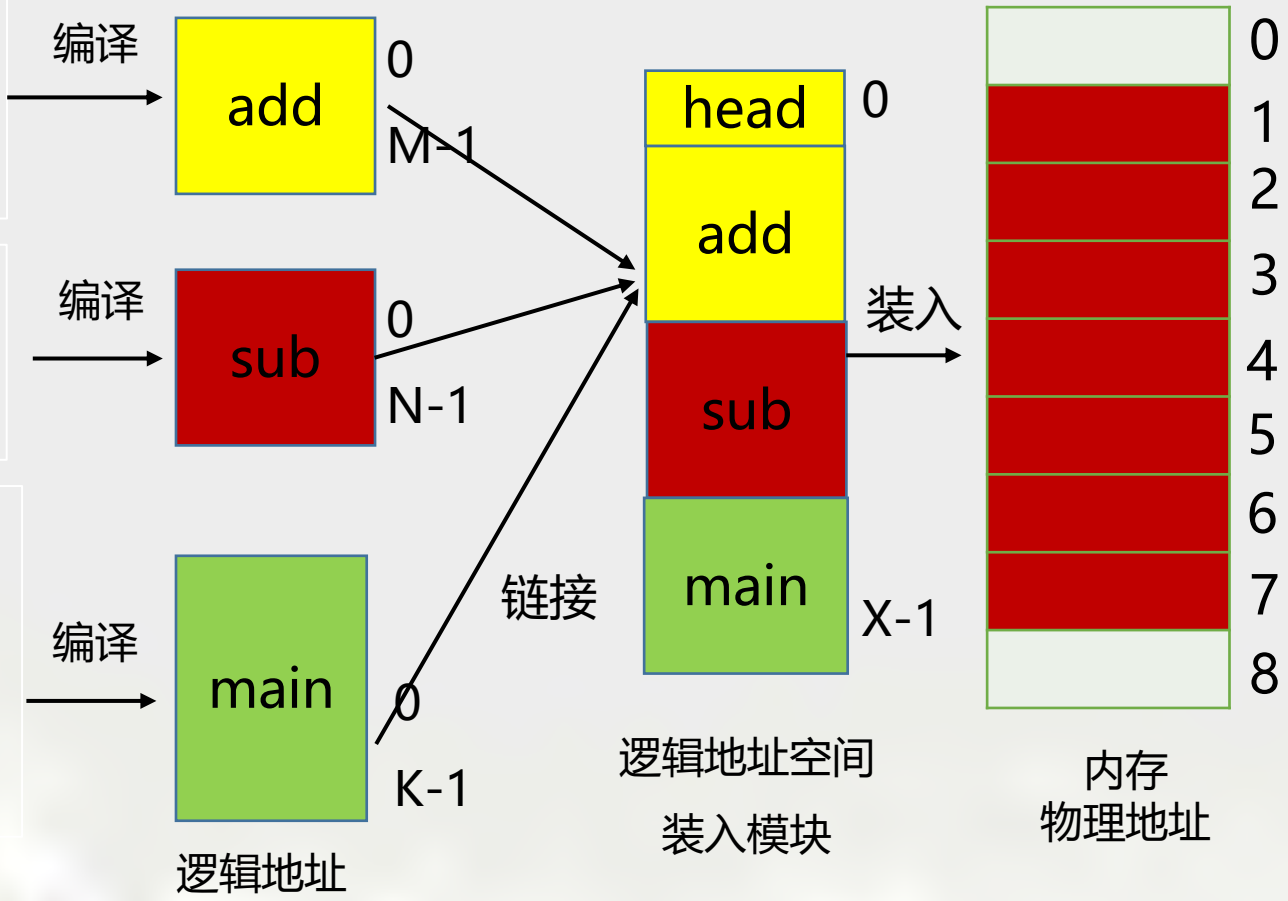
考点一: 内存管理概述

- **静态链接：**程序执行前，将若干从 0 地址开始的目标模块及所需库函数链接为完整、从唯一“0”地址开始的装配模块

```
int add(int a, int b){
    return a+b;
}
```

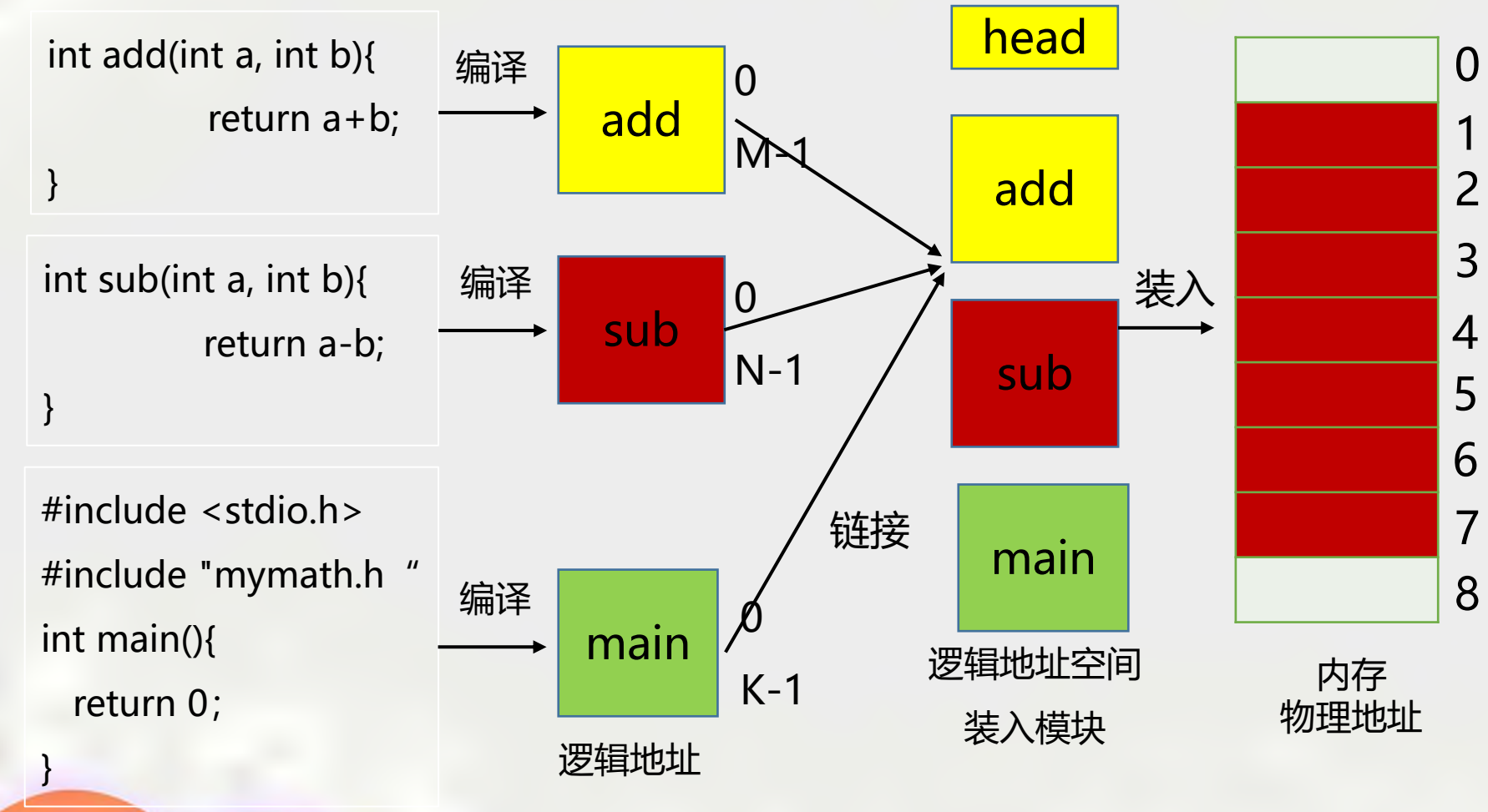
```
int sub(int a, int b){
    return a-b;
}
```

```
#include <stdio.h>
#include "mymath.h “
int main(){
  return 0;
}
```



考点一: 内存管理概述

- **装入时动态链接：**边装入内存边链接各目标模块



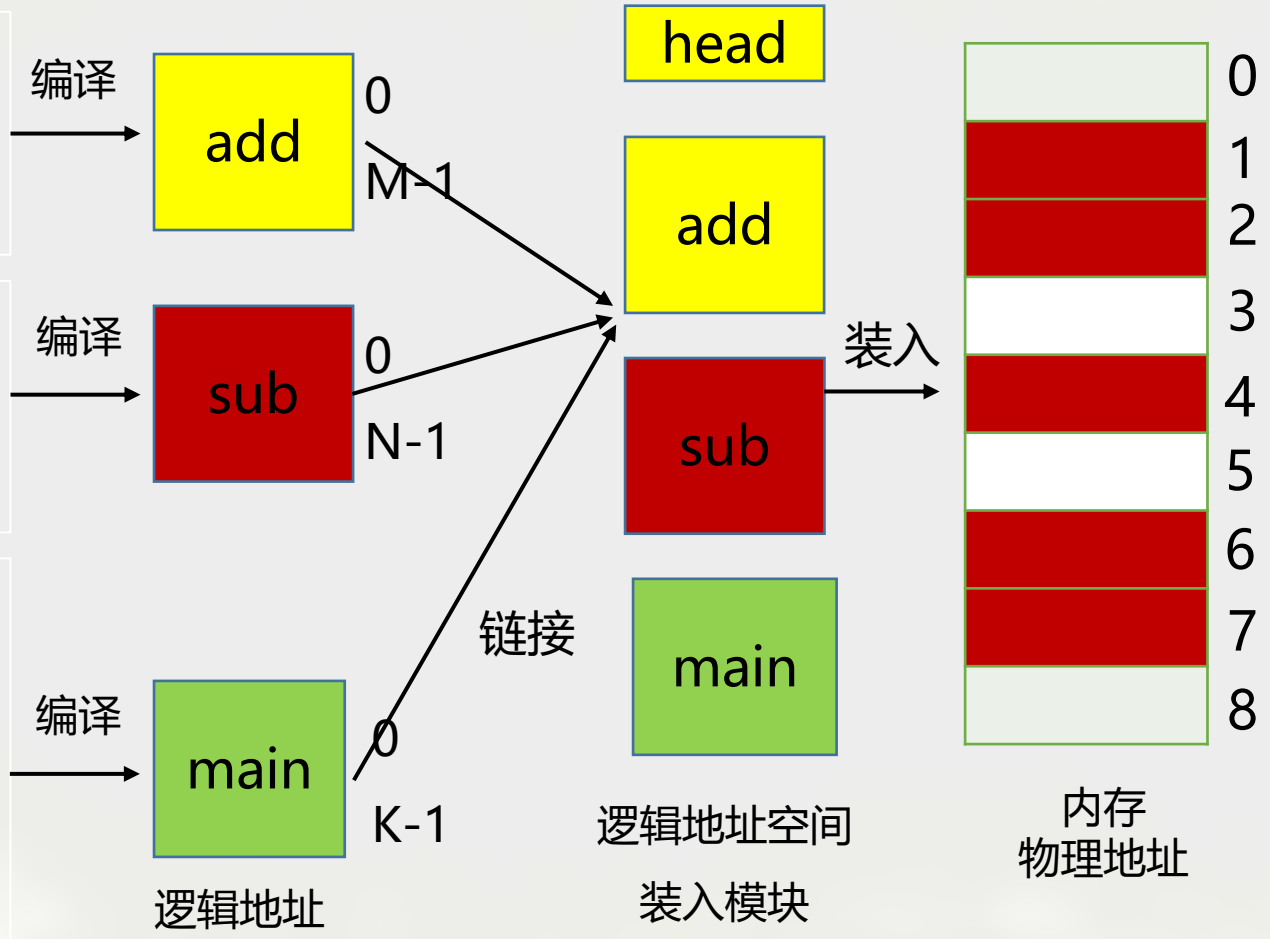
考点一: 内存管理概述

- **运行时动态链接：**程序执行过程中需要目标模块时，才对其进行链接

```
int add(int a, int b){
        return a+b;
}
```

```
int sub(int a, int b){
        return a-b;
}
```

```
#include <stdio.h>
#include "mymath.h "
int main(){
  return 0;
}
```




程序的装入和链接
步骤
编译
源程序 ->目标模块（Object modules）--------Compiler
由编译程序对用户源程序进行编译，形成若干个目标模块
链接
一组目标模块 ->装入模块 （Load Module）----------Linker
由链接程序将编译后形成的一组目标模板以及它们所需要的库函数链接在一起，形成一个完整的装入模块
装入
装入模块 ->内存 --------Loader
由装入程序将装入模块装入内存
程序的链接
静态链接方式(lib)
装入时动态链接
运行时动态链接(dll)

考点一: 内存管理概述

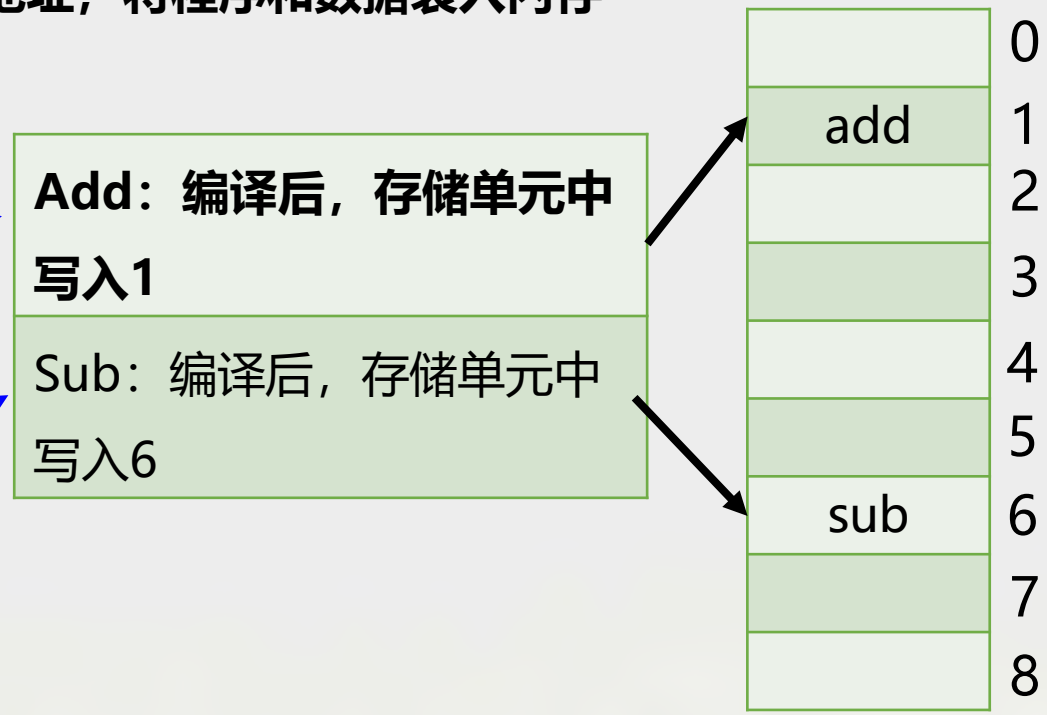
程序的装入

考点一: 内存管理概述

- **绝对装入方式：**编译程序产生**绝对地址的目标代码**，程序中的**逻辑地址**与实际**内存地址**完全**相同；在编译时，如果知道程序将放到内存中的哪个位置，编译程序将产生绝对地址的目标代码。**

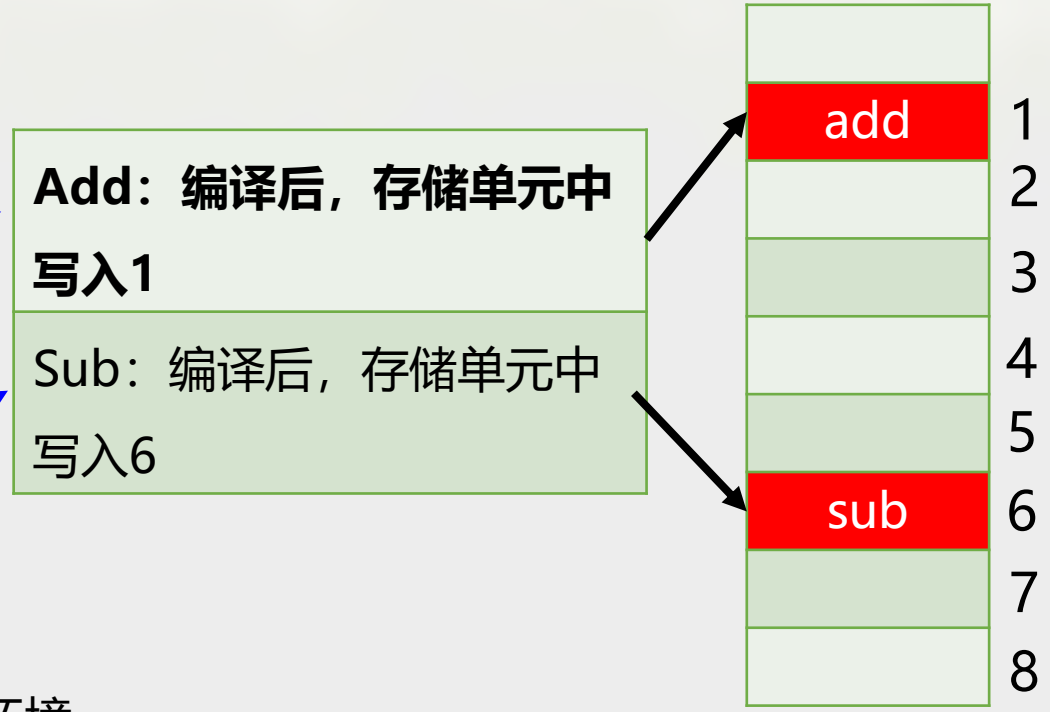
- **装入程序按照装入模块中的地址，将程序和数据装入内存**

```
// mymath.c
int add(int a, int b){
        return a+b;
}
int sub(int a, int b){
        return a-b;
}
```



考点一: 内存管理概述

```
// mymath.c
int add(int a, int b){
        return a+b;
}
int sub(int a, int b){
        return a-b;
}
```

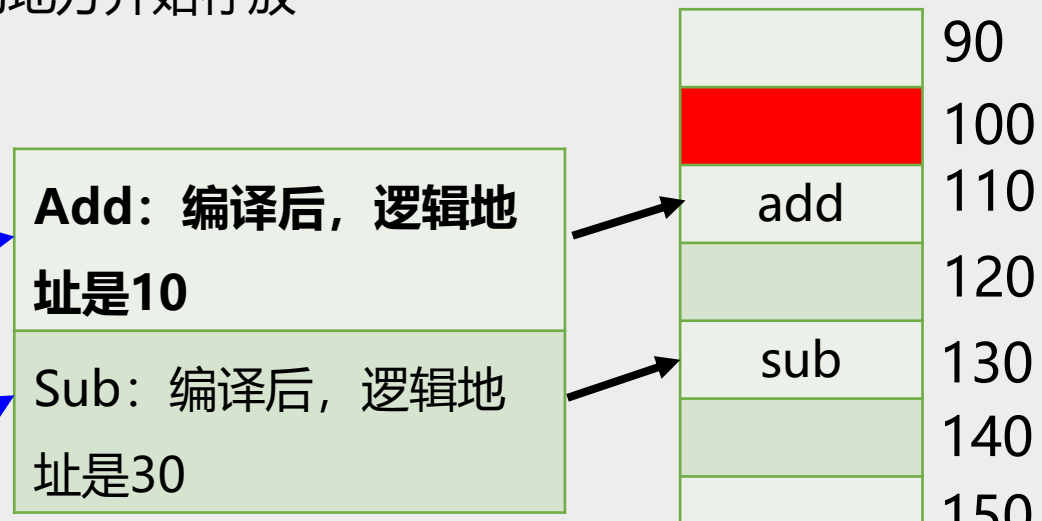


- 绝对装入只适用于单道程序环境。
- 程序中使用的绝对地址，可在编译或汇编时给出，也可由程序员直接赋予。通常情况下都是编译或汇编时再转换为绝对地址。

考点一: 内存管理概述

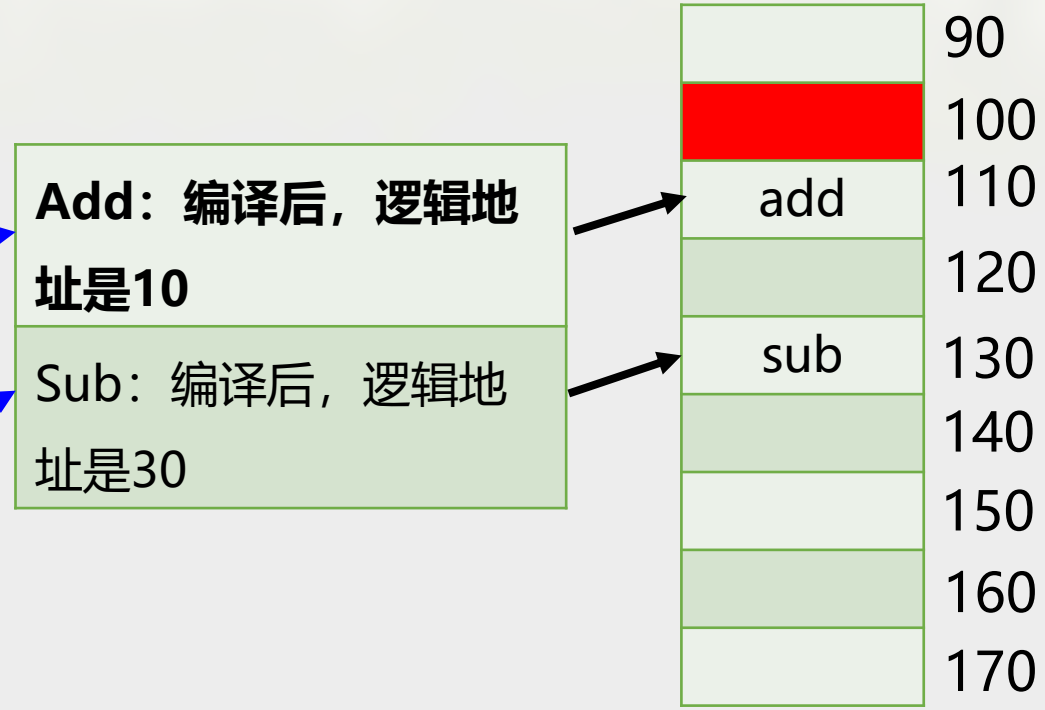
- **可重定位装入方式：多道**程序环境下，**装入**时**一次性**完成逻辑地址到物理**地址**的**变换**，并根据内存的当前情况，将装入模块装入到内存的适当的位置，又称**静态重定位**
- 装入模块要从地址为100的地方开始存放

```
// mymath.c
int add(int a, int b){
        return a+b;
}
int sub(int a, int b){
        return a-b;
}
```



考点一: 内存管理概述

```
// mymath.c
int add(int a, int b){
        return a+b;
}
int sub(int a, int b){
        return a-b;
}
```



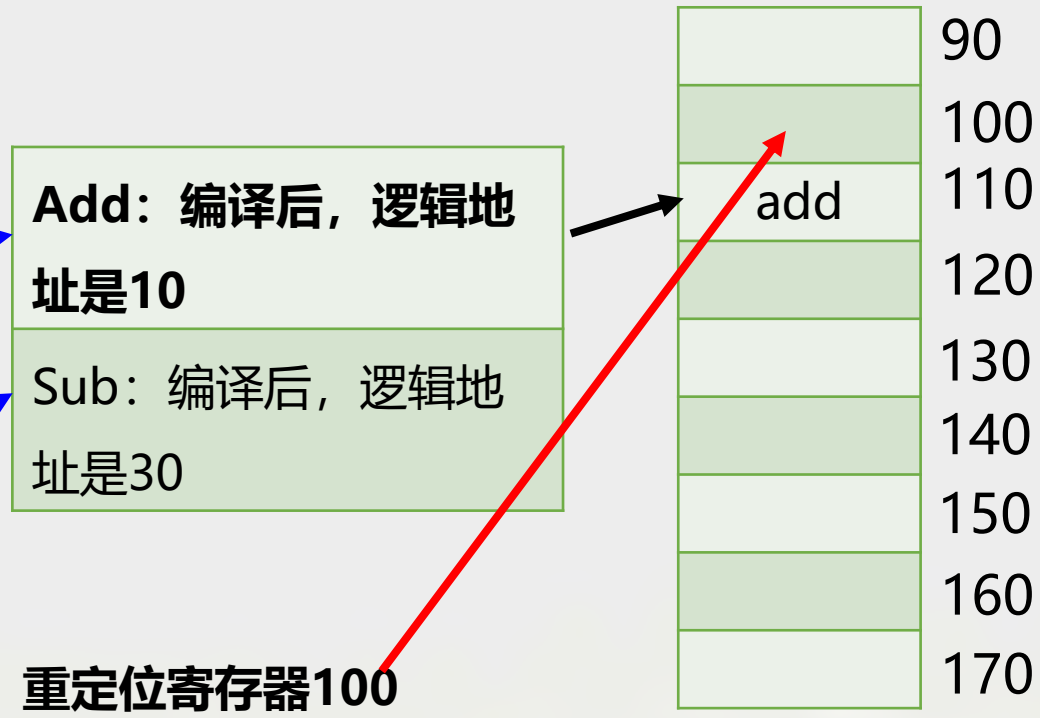
- 静态重定位的特点是在一个作业装入内存时，必须分配其要求的全部内存空间，如果没有足够的内存，就不能装入该作业。
- 作业一旦进入内存后，在运行期间就不能再移动，也不能再申请内存空间。

考点一: 内存管理概述

- **动态运行时装入方式：装入**内存后的所有地址都仍为**相对地址**，**地址变换**在程序**执行期间**、随着对每条指令的访问**自动**进行，又称**动态重定位**

- Add装入模块要从地址为100的地方开始存放

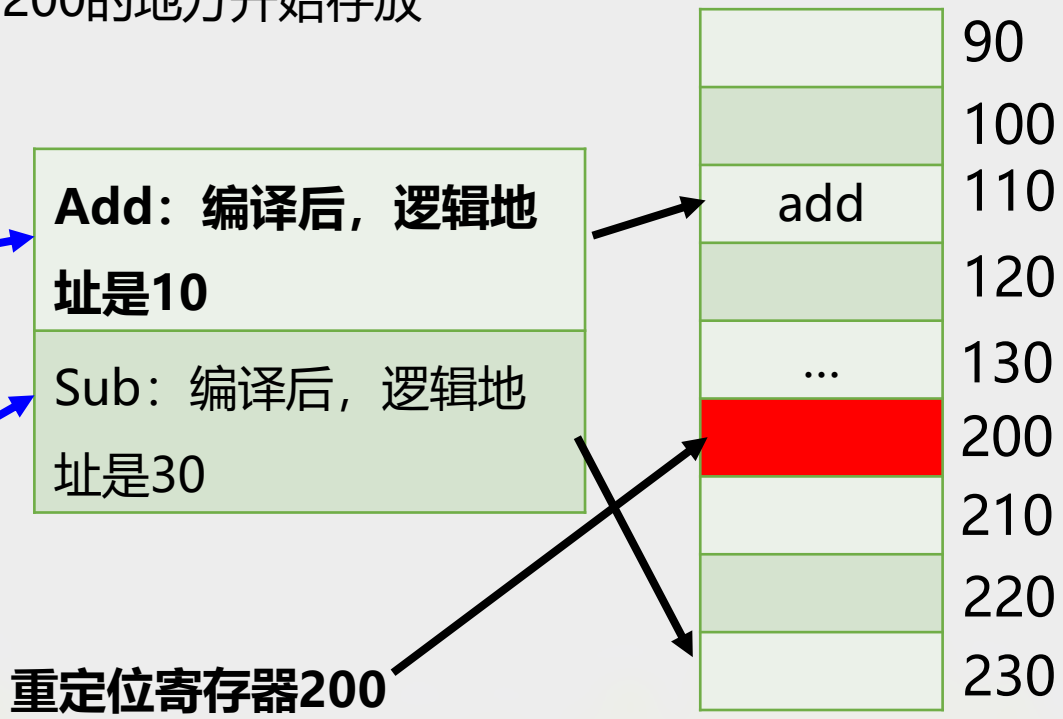
```
// mymath.c
int add(int a, int b){
        return a+b;
}
int sub(int a, int b){
        return a-b;
}
```



考点一: 内存管理概述

- **动态运行时装入方式：装入**内存后的所有地址都仍为**相对地址**，**地址变换**在程序**执行期间**、随着对每条指令的访问**自动**进行，又称**动态重定位**
- Sub装入模块要从地址为200的地方开始存放

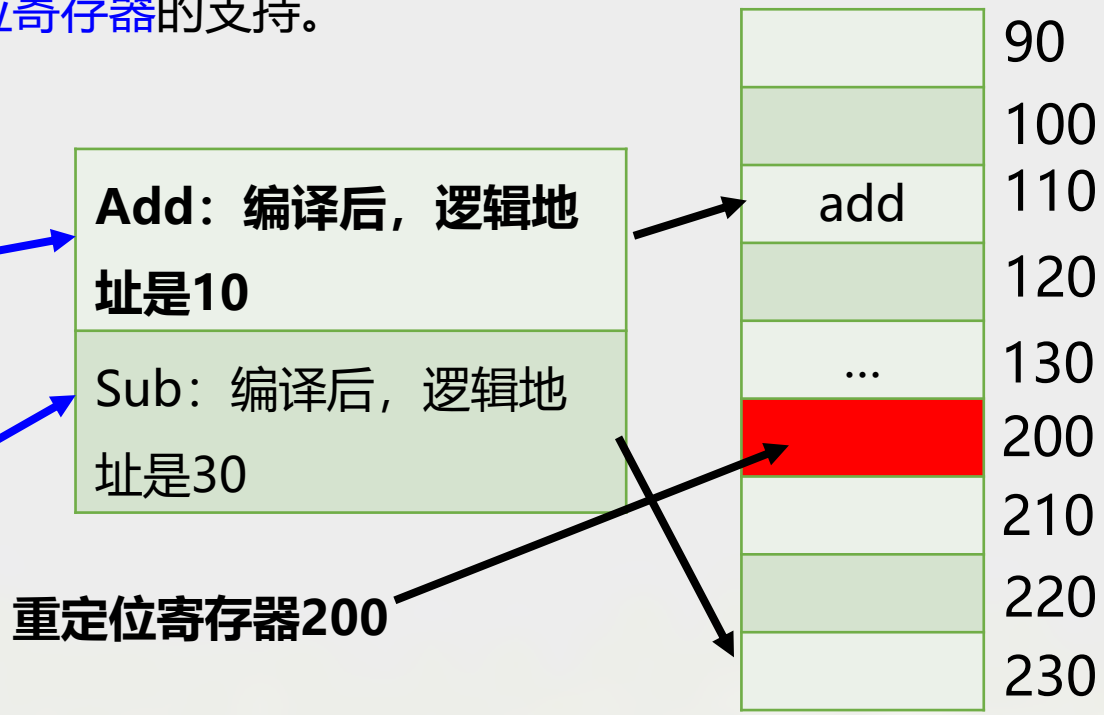
```
// mymath.c
int add(int a, int b){
        return a+b;
}
int sub(int a, int b){
        return a-b;
}
```



考点一: 内存管理概述

- 编译、链接后的装入模块的地址都是从0开始的。装入程序把装入模块装入内存后，并不会立即把逻辑地址转换为物理地址，而是把地址转换推迟到程序真正要执行时才进行。因此装入内存后所有的地址依然是逻辑地址。
- 这种方式需要一个重定位寄存器的支持。

```
// mymath.c
int add(int a, int b){
        return a+b;
}
int sub(int a, int b){
        return a-b;
}
```

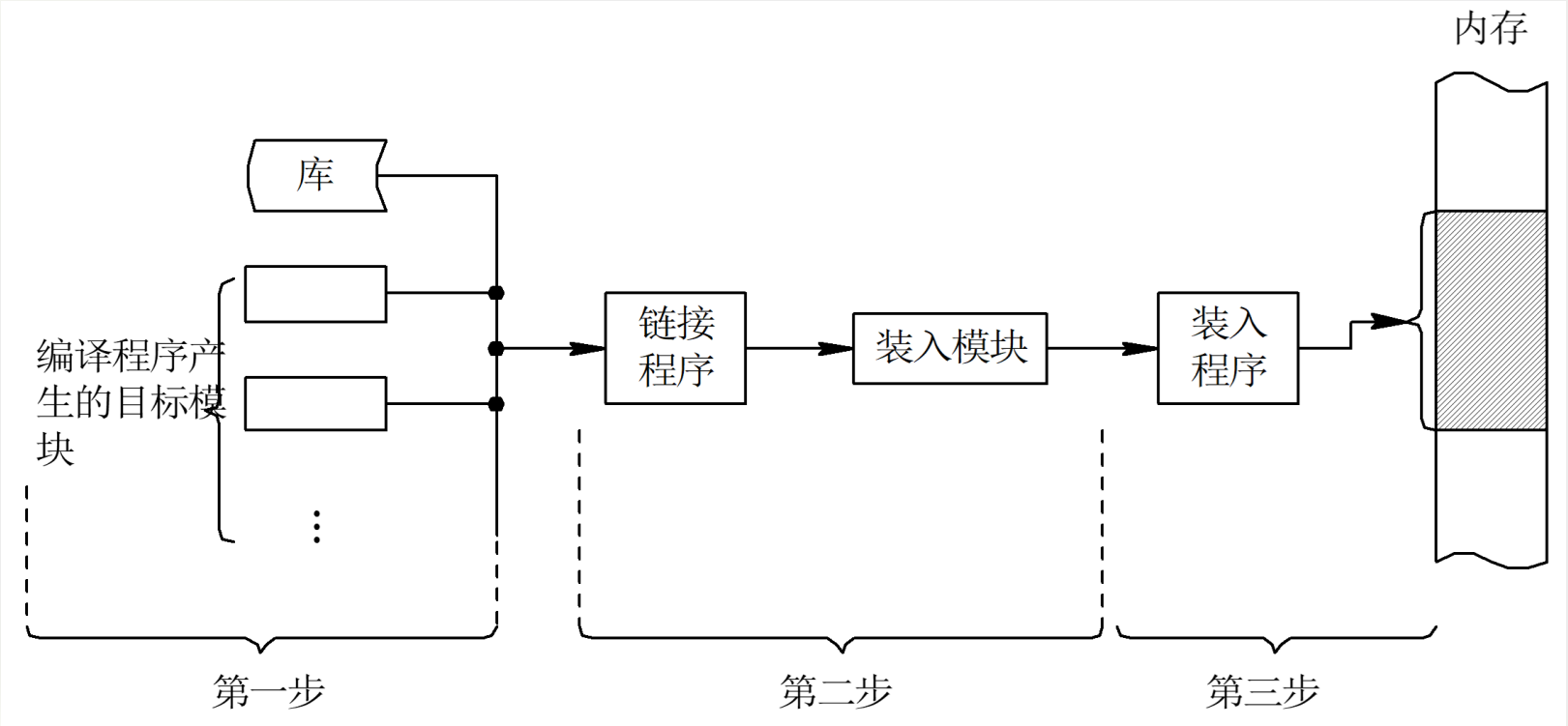



程序的装入
绝对装入方式
在编译时，如果知道程序将驻留在内存中指定的位置。编译程序将产生绝对地址的目标代码。
可重定位装入方式
在可执行文件中，列出各个需要重定位的地址单元和相对地址值。当用户程序被装入内存时，一次性实现逻辑地址到物理地址的转换，以后不再转换(一般在装入内存时由软件完成)。
优点：不需硬件支持，可以装入有限多道程序。
缺点：一个程序通常需要占用连续的内存空间，程序装入内存后不能移动。不易实现共享。
动态运行时的装入方式
动态运行时的装入程序在把装入模块装入内存后，并不立即把装入模块中的逻辑地址转换为物理地址，而是把这种地址转换推迟到程序真正要执行时才进行
优点：
OS可以将一个程序分散存放于不连续的内存空间，可以移动程序，有利用实现共享。
能够支持程序执行中产生的地址引用，如指针变量（而不仅是生成可执行文件时的地址引用）。
缺点：需要硬件支持，OS实现较复杂。
它是虚拟存储的基础。

【牛刀小试】

1．【汕头大学　2016】在虚拟内存管理中，地址变换机构将逻辑地址变为物理地址，形成该逻辑地址的阶段是(　)。

A．编辑　　B．编译　　C．链接　　D．装载



1．【汕头大学　2016】在虚拟内存管理中，地址变换机构将逻辑地址变为物理地址，形成该逻辑地址的阶段是(　)。

A．编辑　　B．编译　　C．链接　　D．装载

B【**解析**】在经过编译后会形成目标模块，目标模块的地址可以称为逻辑地址。因此选择B。

2．【南京理工大学　2017】采用运行时动态装入的进程，在其整个生命周期中允许(　)将其在物理内存中移动。

A．用户有条件地　　B．用户无条件地

C．操作系统有条件地　　D．操作系统无条件地

D【**解析**】动态重定位的特点是可以将程序分配到不连续的存储区中，在程序运行之前只装入它的部分代码即可投入运行，然后在程序运行期间，根据需要动态申请分配内存，因此采用运行时动态装入的进程，在其整个生命周期中允许操作系统无条件地将其在物理内存中移动，因此选择D。

3．【北京交通大学 2016】关于程序链接，如下说法正确的是(　)。

A．根据目标模块大小和链接次序对相对地址进行修改

B．根据装入位置，把目标模块中的相对地址转换为绝对地址

C．把每个目标模块中的相对地址转换为外部调用符号

D．采用静态链接方式更有利于目标模块的共享

C【解析】对相对地址进行修改，在编译程序所产生的所有目标模块中，使用的都是相对地址，起始地址都为0，每个模块中的地址都是相对于起始地址计算的，所以就需要对后面模块的地址加上它自身的长度，再后面的模块依次累加，A错误；

3．【北京交通大学 2016】关于程序链接，如下说法正确的是(　)。

A．根据目标模块大小和链接次序对相对地址进行修改

B．根据装入位置，把目标模块中的相对地址转换为绝对地址

C．把每个目标模块中的相对地址转换为外部调用符号

D．采用静态链接方式更有利于目标模块的共享

C【解析】根据装入位置，把目标模块中的相对地址转换为绝对地址属于程序的装入，B错误；

3．【北京交通大学 2016】关于程序链接，如下说法正确的是(　)。

A．根据目标模块大小和链接次序对相对地址进行修改

B．根据装入位置，把目标模块中的相对地址转换为绝对地址

C．把每个目标模块中的相对地址转换为外部调用符号

D．采用静态链接方式更有利于目标模块的共享

C【解析】变换外部调用符号，将每个模块中所用的外部调用符号也都变换为相对地址，C正确；

3．【北京交通大学 2016】关于程序链接，如下说法正确的是(　)。

A．根据目标模块大小和链接次序对相对地址进行修改

B．根据装入位置，把目标模块中的相对地址转换为绝对地址

C．把每个目标模块中的相对地址转换为外部调用符号

D．采用静态链接方式更有利于目标模块的共享

C【解析】采用装入时动态链接方式便于实现对目标模块的共享，采用静态链接方式时，每个应用模块都必须含有其目标模块的拷贝，无法实现对目标模块的共享，D错误。因此选择C。

4．【广东工业大学 2014】外存（如磁盘）上存放的程序和数据(　)。

A．可由CPU直接访问　　B．必须在CPU访问之前移入内存

C．是必须由文件系统管理的　　D．必须由进程调度程序管理

B【**解析**】CPU访问的数据，需要先调入内存，然后再访问，不能直接读取磁盘上的数据；外存（如磁盘）上存放的程序和数据与操作系统共同管理，因此选择B。

谢谢大家

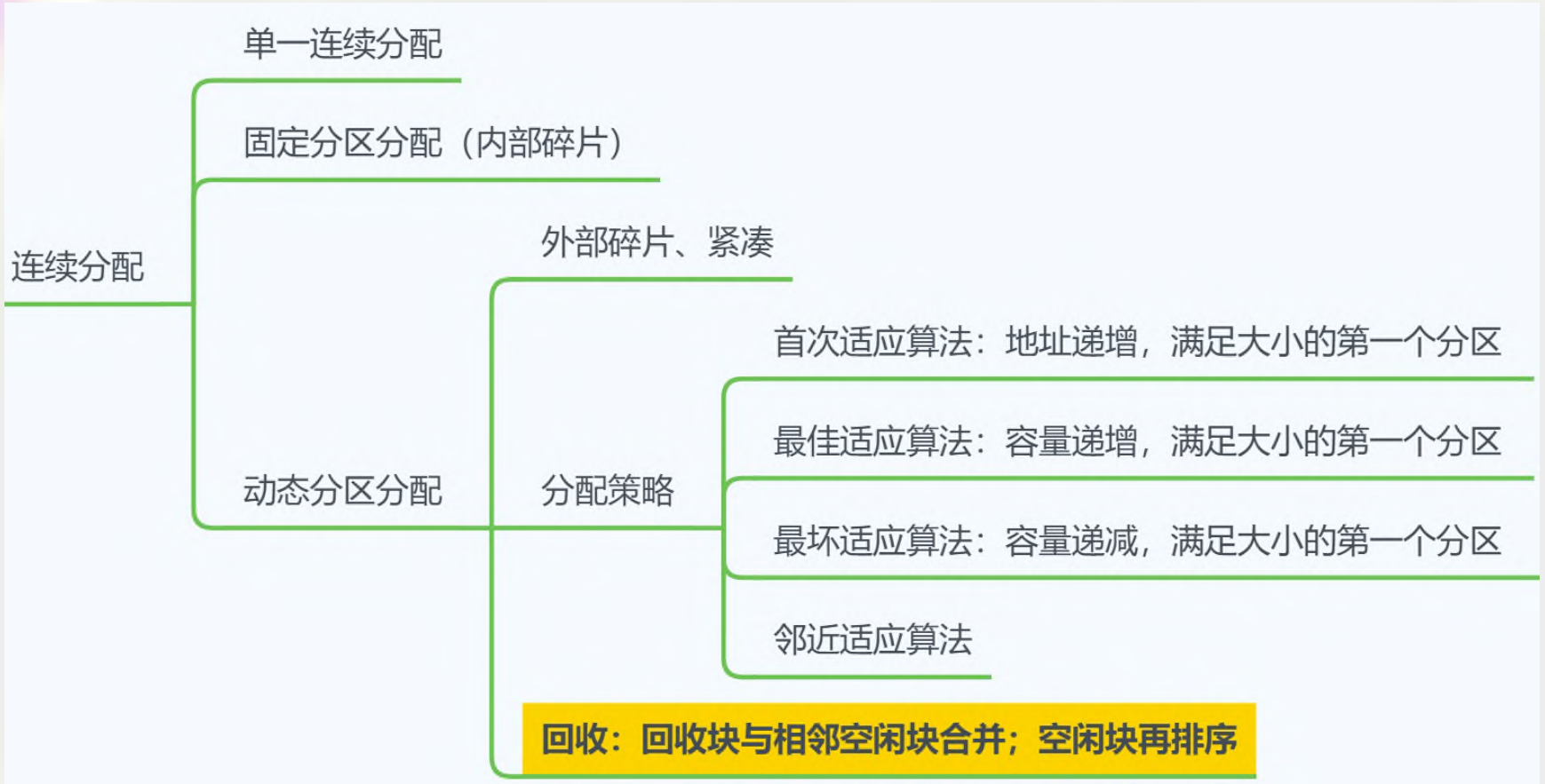
考点二:

内存连续分配管理方式

考点框架

- 单一连续分配
- 固定分区分配
- 动态分区分配
- 内存回收

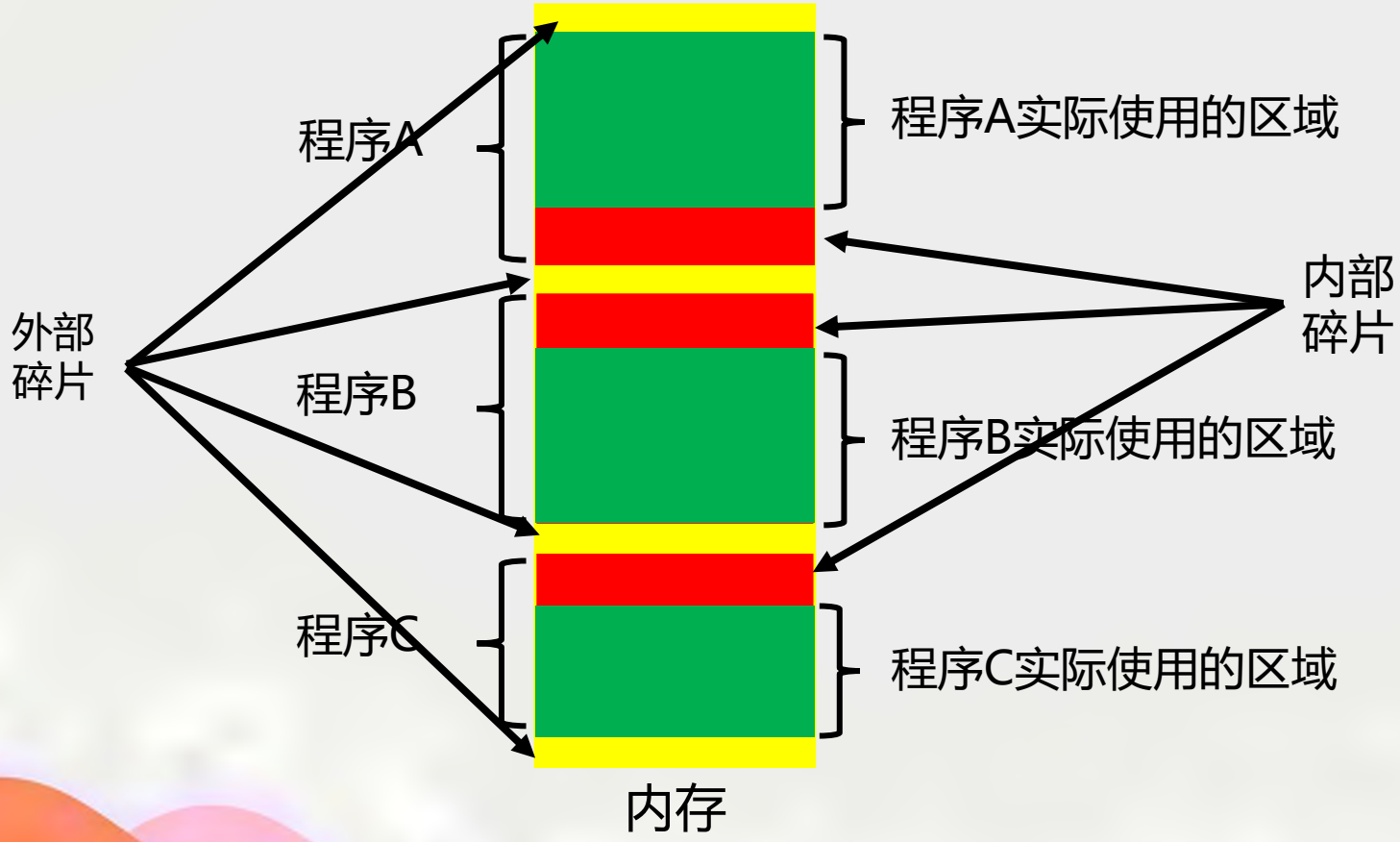
考点二: 内存连续分配管理方式



考点二: 内存连续分配管理方式

两个概念:

- 内碎片：占用分区之内未被利用的空间
- 外碎片：占用分区之间难以利用的空闲分区（通常是小空闲分区）



考点二: 内存连续分配管理方式

单一连续分配

考点二: 内存连续分配管理方式

- 这是最简单的一种存储管理方式，但只能用于单用户、单任务的操作系统中。
- 采用这种存储管理方式时，可把内存分为系统区和用户区两部分，系统区仅提供给OS使用，通常是放在内存的低址部分；
- 用户区是指除系统区以外的全部内存空间， 提供给用户使用。

考点二: 内存连续分配管理方式

- 优点：实现简单；无外部碎片；可以采用覆盖技术扩充内存；不一定需要采取内存保护
- 缺点：只能用于单用户、单任务的操作系统中；有内部碎片；存储器利用率极低。用户区浪费：小作业

考点二: 内存连续分配管理方式

固定分区分配

考点二: 内存连续分配管理方式

- 为了能在内存中装入多道程序，且这些程序之间又不会相互干扰，于是将整个用户空间划分为若干个固定大小的分区，在每个分区中只装入一道作业，这样就形成了最早的、最简单的一种可运行多道程序的内存管理方式。

(1) 分区大小相等

(2) 分区大小不等

管理数据结构：分区使用表

分区号	大小(KB)	起始地址(K)	状态
1	16	10	已分配
2	32	26	未分配
3	64	58	已分配
4	128	122	未分配
5	256	250	未分配

考点二: 内存连续分配管理方式

- 操作系统需要建立一个数据结构——分区说明表，来实现各个分区的分配与回收。每个表项对应一个分区，通常按分区大小排列。每个表项包括对应分区的大小、起始地址、状态（是否已分配）。

管理数据结构：分区使用表

分区号	大小(KB)	起始地址(K)	状态
1	16	10	已分配
2	32	26	未分配
3	64	58	已分配
4	128	122	未分配
5	256	250	未分配

- 当某用户程序要装入内存时，由操作系统内核程序根据用户程序大小检索该表，
- 从中找到一个能满足大小的、未分配的分区，将之分配给该程序，然后修改状态为“已分配”。

考点二: 内存连续分配管理方式

- 这种分区方式存在两个问题：
  - 一是程序可能太大而放不进任何一个分区中，这时用户不得不使用覆盖技术来使用内存空间；
  - 二是主存利用率低，当程序小于固定分区大小时，也占用了一个完整的内存分区空间，这样分区内部有空间浪费，这种现象称为内部碎片。

考点二: 内存连续分配管理方式

- 固定分区是可用于多道程序设计最简单的存储分配，无外部碎片，
- 不能实现多进程共享一个主存区，所以存储空间利用率低。
- 固定分区分配很少用于现在通用的操作系统中，但在某些用于控制多个相同对象的控制系统中仍发挥着一定的作用。

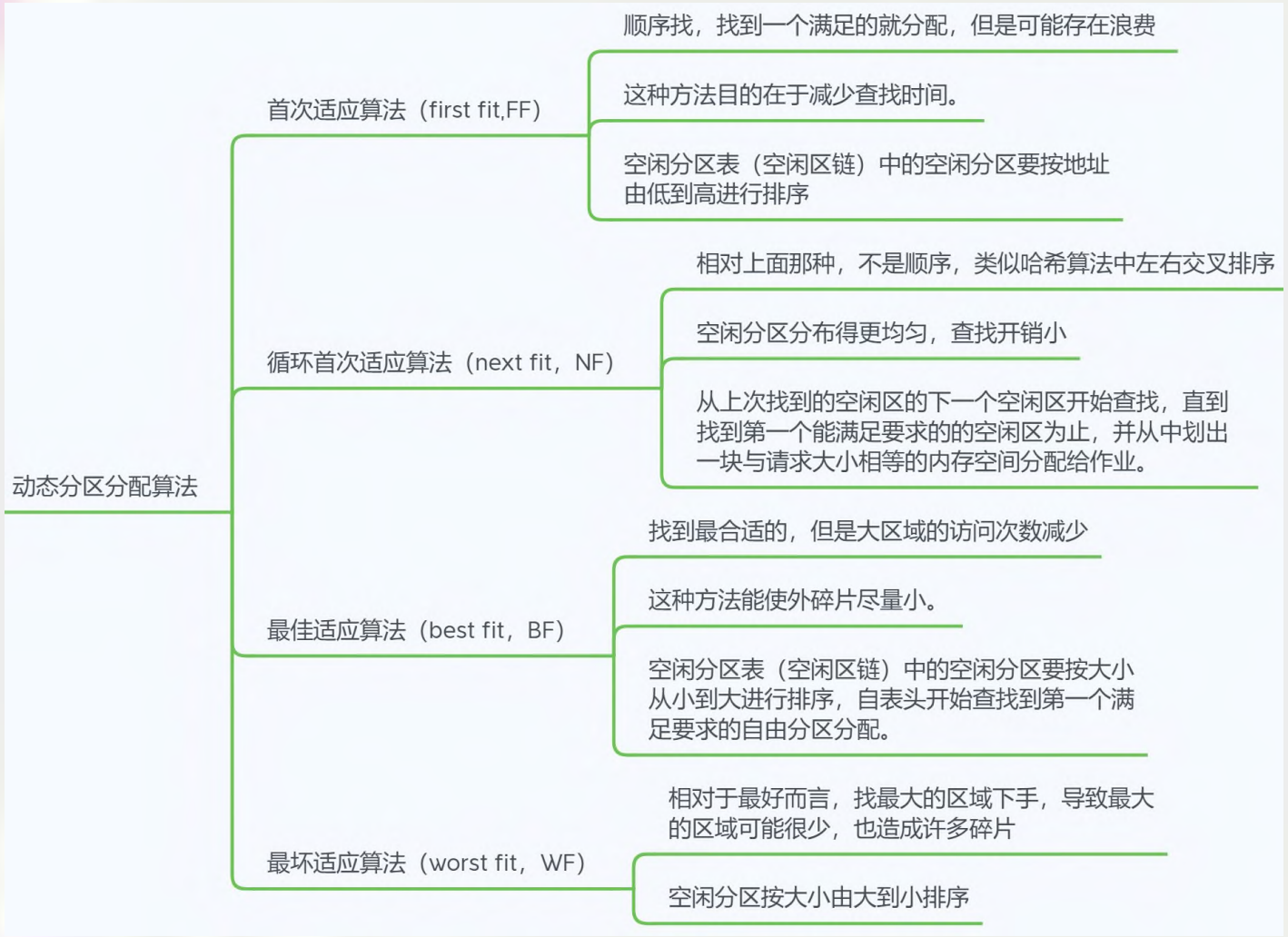
考点二: 内存连续分配管理方式

动态分区分配

考点二: 内存连续分配管理方式

- 动态分区分配又称为可变分区分配。这种分配方式不会预先划分内存分区，而是在进程装入内存时，根据进程的大小动态地建立分区，并使分区的大小正好适合进程的需要。因此系统分区的大小和数目是可变的。

考点二: 内存连续分配管理方式



考点二: 内存连续分配管理方式

首次适应算法（first fit,FF）

- 空闲分区以地址递增的次序链接。分配内存时顺序查找，找到大小能满足要求的第一个空闲分区, 从该分区中划出一块内存分配给请求者，余下的空闲分区仍留在空闲分区链中

考点二: 内存连续分配管理方式

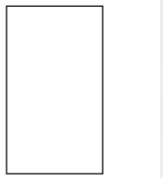
首次适应算法（first fit,FF）

- 某操作系统采用可变分区分配存储管理方法，用户区为512K且始址为0。若分配采用分配空闲区低地址部分的方案，对下述申请序列：申请310K，申请100K，释放310K，申请150K，申请40K，申请30K。采用首次适应算法，回答下列问题：

①给出每一步的已分配空间、空闲分区（给出始址，大小）？

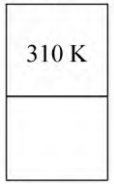
①用户区大小为512K,且从0开始,首次适应算法让空闲分区按地址递增,按照操作顺序,得其操作过程与分区表如图所示

初始



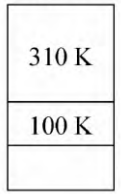
编号	大小	基址
0	512 K	0

申请310 K



编号	大小	基址
0	202 K	310 K

申请100 K



编号	大小	基址
0	102 K	410 K

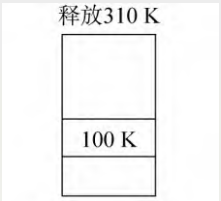
考点二: 内存连续分配管理方式

首次适应算法（first fit,FF）

- 某操作系统采用可变分区分配存储管理方法，用户区为512K且始址为0。若分配采用分配空闲区低地址部分的方案，对下述申请序列：申请310K，申请100K，释放310K，申请150K，申请40K，申请30K。采用首次适应算法，回答下列问题：

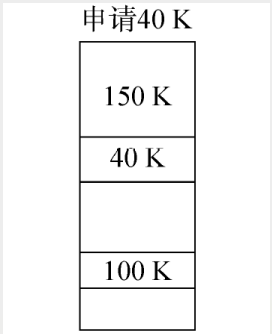
①给出每一步的已分配空间、空闲分区（给出始址，大小）？

①用户区大小为512K,且从0开始,首次适应算法让空闲分区按地址递增,按照操作顺序,得其操作过程与分区表如图所示

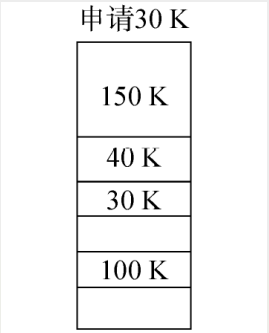


编号	大小	基址
0	310 K	0
1	102 K	410 K

编号	大小	基址
0	160 K	150 K
1	102 K	410 K



编号	大小	基址
0	120 K	190 K
1	102 K	410 K



编号	大小	基址
0	90 K	220 K
1	102 K	410 K

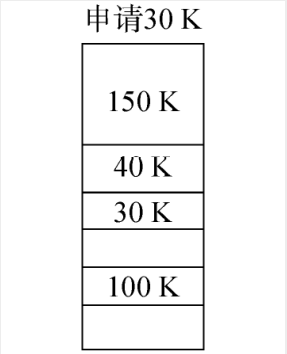
考点二: 内存连续分配管理方式

首次适应算法（first fit,FF）

- 某操作系统采用可变分区分配存储管理方法，用户区为512K且始址为0。若分配采用分配空闲区低地址部分的方案，对下述申请序列：申请310K，申请100K，释放310K，申请150K，申请40K，申请30K。采用首次适应算法，回答下列问题：

②若再申请120K，还能分配这120K存储空间吗?

再申请120K时,剩余的2个空闲分区大小为90K与102K,发现不够分配



编号	大小	基址
0	90 K	220 K
1	102 K	410 K

考点二: 内存连续分配管理方式

首次适应算法（first fit,FF）

- 特点：该算法倾向于使用内存中低地址部分的空闲区，在高地址部分的空闲区很少被利用，从而保留了高地址部分的大空闲区。显然为以后到达的大作业分配大的内存空间创造了条件。
- 缺点：低地址部分不断被划分，留下许多难以利用、很小的空闲区，
  - 每次查找又都从低地址部分开始，会增加查找的开销。

考点二: 内存连续分配管理方式

循环首次适应(next fit，NF)算法

- 空闲分区以地址递增的顺序排列
- 循环首次适应算法在为进程分配内存空间时，不再是每次都从链首开始查找，而是从上次找到的空闲分区的下一个空闲分区开始查找，直至找到一个能满足要求的空闲分区，从中划出一块与请求大小相等的内存空间分配给作业。

考点二: 内存连续分配管理方式

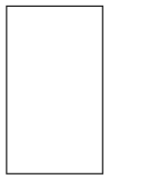
循环首次适应(next fit，NF)算法

- 某操作系统采用可变分区分配存储管理方法，用户区为512K且始址为0。若分配采用分配空闲区低地址部分的方案，对下述申请序列：申请310K，申请100K，释放310K，申请150K，申请40K，申请30K。采用循环首次适应算法，回答下列问题：

①给出每一步的已分配空间、空闲分区（给出始址，大小）？

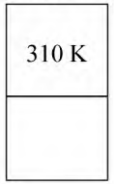
①用户区大小为512K,且从0开始,循环首次适应算法让空闲分区按地址递增,按照操作顺序,得其操作过程与分区表如图所示

初始



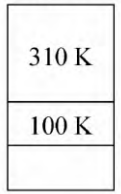
编号	大小	基址
0	512 K	0

申请310 K



编号	大小	基址
0	202 K	310 K

申请100 K



编号	大小	基址
0	102 K	410 K

考点二: 内存连续分配管理方式

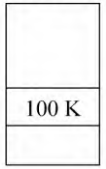
循环首次适应(next fit，NF)算法

- 某操作系统采用可变分区分配存储管理方法，用户区为512K且始址为0。若分配采用分配空闲区低地址部分的方案，对下述申请序列：申请310K，申请100K，释放310K，申请150K，申请40K，申请30K。采用循环首次适应算法，回答下列问题：

①给出每一步的已分配空间、空闲分区（给出始址，大小）？

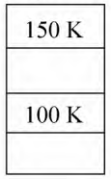
①用户区大小为512K,且从0开始,循环首次适应算法让空闲分区按地址递增,按照操作顺序,得其操作过程与分区表如图所示

释放310 K



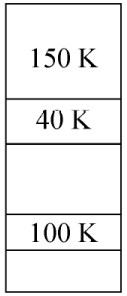
编号	大小	基址
0	310 K	0
1	102 K	410 K

申请150 K



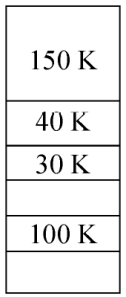
编号	大小	基址
0	160 K	150 K
1	102 K	410 K

申请40 K



编号	大小	基址
0	120 K	190 K
1	102 K	410 K

申请30 K



编号	大小	基址
0	90 K	220 K
1	102 K	410 K

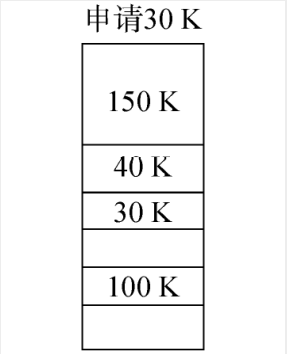
考点二: 内存连续分配管理方式

循环首次适应(next fit，NF)算法

- 某操作系统采用可变分区分配存储管理方法，用户区为512K且始址为0。若分配采用分配空闲区低地址部分的方案，对下述申请序列：申请310K，申请100K，释放310K，申请150K，申请40K，申请30K。采用循环首次适应算法，回答下列问题：

②若再申请120K，还能分配这120K存储空间吗?

再申请120K时,剩余的2个空闲分区大小为90K与102K,发现不够分配



编号	大小	基址
0	90 K	220 K
1	102 K	410 K

考点二: 内存连续分配管理方式

循环首次适应(next fit，NF)算法

- 优点: 内存空闲分区分布均匀,减少了查找系统开销
- 缺点: 会使内部碎片分散到所有空间中

考点二: 内存连续分配管理方式

最佳适应(best fit，BF)算法

- 该算法要求将所有的空闲区按其大小排序后，以递增顺序形成一个空白链。

  这样每次找到的第一个满足要求的空闲区，必然是最优的。

考点二: 内存连续分配管理方式

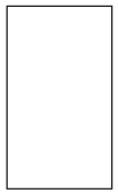
最佳适应(best fit，BF)算法

- 某操作系统采用可变分区分配存储管理方法，用户区为512K且始址为0。若分配采用分配空闲区低地址部分的方案，对下述申请序列：申请310K，申请100K，释放310K，申请150K，申请40K，申请30K。采用最佳适应算法，回答下列问题：

①给出每一步的已分配空间、空闲分区（给出始址，大小）？

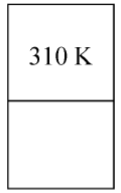
①用户区大小为512K,且从0开始,最佳适应算法让空闲分区按容量递增且优先分配空间最接近的,按照操作顺序可得其操作过程与分区表如图所示。

初始



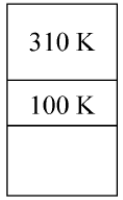
编号	大小	基址
0	512 K	0

申请310 K



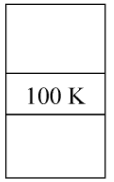
编号	大小	基址
0	202 K	310 K

申请100 K



编号	大小	基址
0	102 K	410 K

释放310 K



编号	大小	基址
0	102 K	410 K
1	310 K	0

考点二: 内存连续分配管理方式

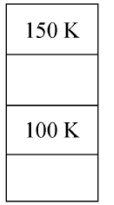
最佳适应(best fit，BF)算法

- 某操作系统采用可变分区分配存储管理方法，用户区为512K且始址为0。若分配采用分配空闲区低地址部分的方案，对下述申请序列：申请310K，申请100K，释放310K，申请150K，申请40K，申请30K。采用最佳适应算法，回答下列问题：

①给出每一步的已分配空间、空闲分区（给出始址，大小）？

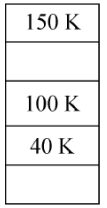
①用户区大小为512K,且从0开始,最佳适应算法让空闲分区按容量递增且优先分配空间最接近的,按照操作顺序可得其操作过程与分区表如图所示。

申请150 K



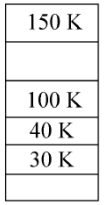
编号	大小	基址
0	102 K	410 K
1	160 K	150 K

申请40 K



编号	大小	基址
0	62 K	450 K
1	160 K	150 K

申请30 K



编号	大小	基址
0	32 K	480 K
1	160 K	150 K

考点二: 内存连续分配管理方式

最佳适应(best fit，BF)算法

- 某操作系统采用可变分区分配存储管理方法，用户区为512K且始址为0。若分配采用分配空闲区低地址部分的方案，对下述申请序列：申请310K，申请100K，释放310K，申请150K，申请40K，申请30K。采用最佳适应算法，回答下列问题：

②若再申请120K，还能分配这120K存储空间吗?

②若再申请120KB,剩余的2个空闲区为32K与160K,可以进行分配

分配后剩余的两个空闲区[基址,大小],[270 K,40K],[480 K,32 K]

编号	大小	基址
0	32 K	480 K
1	160 K	150 K

考点二: 内存连续分配管理方式

最佳适应(best fit，BF)算法

- 优点: 每次分配给文件的都是最合适该文件大小的分区。
- 缺点:内存中留下许多难以利用的小的空闲区。

考点二: 内存连续分配管理方式

最坏适应(worst fit，WF)算法

- 又称最大适应(Largest Fit)算法，空闲分区以容量递减的次序链接。找到第一个能满足要求的空闲分区，也就是挑选出最大的分区。

考点二: 内存连续分配管理方式

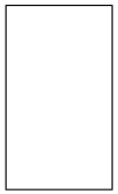
最坏适应(worst fit，WF)算法

- 某操作系统采用可变分区分配存储管理方法，用户区为512K且始址为0。若分配采用分配空闲区低地址部分的方案，对下述申请序列：申请310K，申请100K，释放310K，申请150K，申请40K，申请30K。采用最坏适应算法，回答下列问题：

①给出每一步的已分配空间、空闲分区（给出始址，大小）？

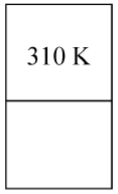
①用户区大小为512K,且从0开始,最坏适应算法让空闲区按容量递减,且优先分配空间最大的分区,按照操作顺序,可得其操作过程与分区表如图所示。

初始



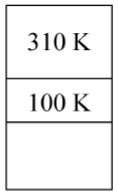
编号	大小	基址
0	512 K	0

申请310 K



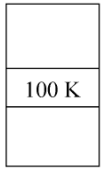
编号	大小	基址
0	202 K	310 K

申请100 K



编号	大小	基址
0	102 K	410 K

释放310 K



编号	大小	基址
0	310 K	0
1	102 K	410 K

考点二: 内存连续分配管理方式

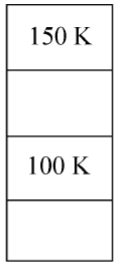
最坏适应(worst fit，WF)算法

- 某操作系统采用可变分区分配存储管理方法，用户区为512K且始址为0。若分配采用分配空闲区低地址部分的方案，对下述申请序列：申请310K，申请100K，释放310K，申请150K，申请40K，申请30K。采用最坏适应算法，回答下列问题：

①给出每一步的已分配空间、空闲分区（给出始址，大小）？

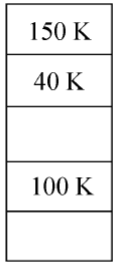
①用户区大小为512K,且从0开始,最坏适应算法让空闲区按容量递减,且优先分配空间最大的分区,按照操作顺序,可得其操作过程与分区表如图所示。

申请150 K



编号	大小	基址
0	160 K	150 K
1	102 K	410 K

申请40 K



编号	大小	基址
0	120 K	190 K
1	102 K	410 K

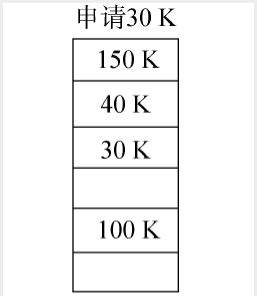
考点二: 内存连续分配管理方式

最坏适应(worst fit，WF)算法

- 某操作系统采用可变分区分配存储管理方法，用户区为512K且始址为0。若分配采用分配空闲区低地址部分的方案，对下述申请序列：申请310K，申请100K，释放310K，申请150K，申请40K，申请30K。采用最坏适应适应算法，回答下列问题：

②若再申请120K，还能分配这120K存储空间吗?

②再申请120KB,剩余的2个分区大小为 102 K与90K，不可以进行分配。



编号	大小	基址
0	102 K	410 K
1	90 K	220 K

考点二: 内存连续分配管理方式

最坏适应(worst fit，WF)算法

- 优点: 可以减少难以利用的小碎片
- 缺点: 每次都选最大的分区进行分配,虽然可以让分配后留下的空闲区更大、更可用, 但是这种方式会导致较大的连续空闲区被迅速用完, 如果之后有“大进程”到达,就没有内存分区可用了,
  - 需要排序,算法开销大

考点二: 内存连续分配管理方式

动态分区分配算法总结

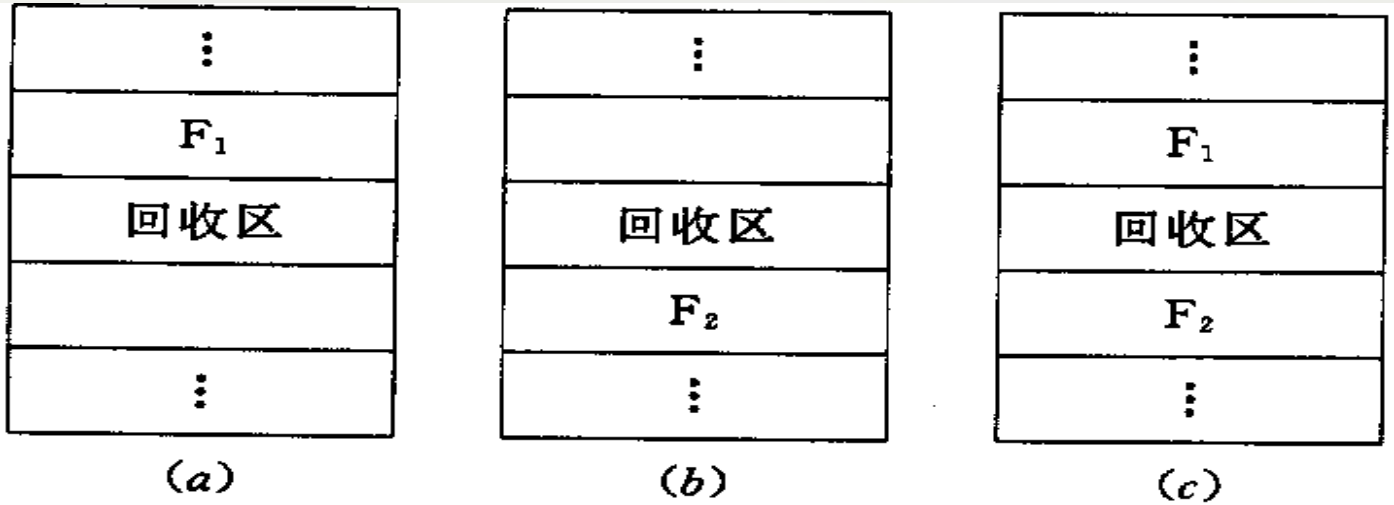
算法	算法思想	分区排列顺序	优点	缺点
首次适应	从头到尾找到一个合适的分区	地址递增排列	综合看性能最好。算法开销小，回收分区后一般不需要对空闲分区队列重新排序。	每次均从低地址开始查找，查找效率较低
最佳适应	优先使用更小的分区，以保留更多大分区	容量递增排列	会有更多的大分区被保留下来，更能满足大进程需求	会产生很多太小的、难以利用的碎片；算法开销大，回收分区后可能需要对空闲分区队列重新排序
最坏适应	优先使用更大的分区，以防止产生太小的不可用的碎片	容量递减排列	可以减少难以利用的小碎片	大分区容易被用完，不利于大进程；算法开销大
临近适应	由首次适应演变而来，每次从上次查找结束位置开始查找	地址递增排成循环链表	不用每次都从低地址的小分区开始检索。算法开销小	会使高地址的大分区也被用完

考点二: 内存连续分配管理方式

内存回收

考点二: 内存连续分配管理方式

- 就是要把空闲区和回收区合并起来



- 情况1：空闲区节点中，地址不变，大小修改即可。要把回收区的大小加进来。
- 情况2：该节点地址要变，变成回收区的地址，然后大小也要修改。
- 情况3：把回收区和空闲区加起来，这里会导致空闲区少一个。

【政哥点拨】

1．某计算机按字节编址，其动态分区内存管理采用最佳适应算法，每次分配和回收内存后都对空闲分区链重新排序。当前空闲分区信息如表4-2所示。

表4-2 当前空闲分区信息

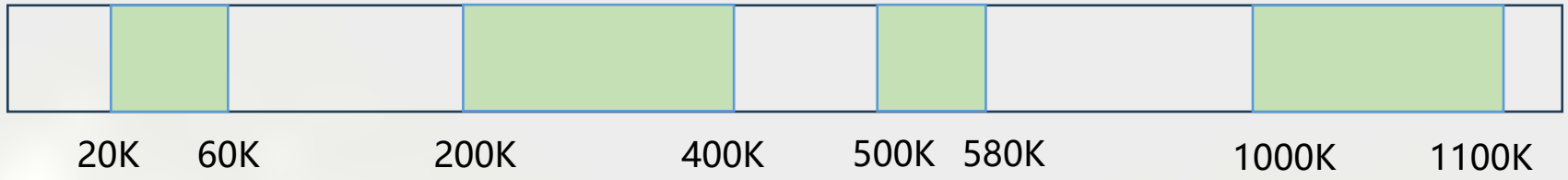
分区起始地址	20K	500K	1000K	200K
分区大小	40KB	80KB	100KB	200KB

回收起始地址为60 K、大小为140 KB的分区后，系统中空闲分区的数量、空闲分区链第一个分区的起始地址和大小分别是(　)。

A．3, 20 K, 380 KB　　B．3, 500 K, 80 KB

C．4, 20 K, 180 KB　　D．4, 500 K, 80 KB

B【解析】根据题意，得到回收前的内存分配情况如下所示：【深色区域是空闲区】



1．某计算机按字节编址，其动态分区内存管理采用最佳适应算法，每次分配和回收内存后都对空闲分区链重新排序。当前空闲分区信息如表4-2所示。

表4-2 当前空闲分区信息

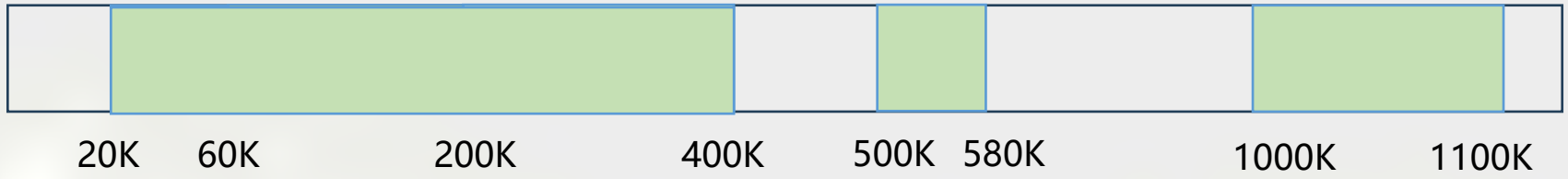
分区起始地址	20K	500K	1000K	200K
分区大小	40KB	80KB	100KB	200KB

回收起始地址为60 K、大小为140 KB的分区后，系统中空闲分区的数量、空闲分区链第一个分区的起始地址和大小分别是(　)。

A．3, 20 K, 380 KB　　B．3, 500 K, 80 KB

C．4, 20 K, 180 KB　　D．4, 500 K, 80 KB

B【解析】根据题意，得到回收前的内存分配情况如下所示：【深色区域是空闲区】



回收起始地址为60 K、大小为140 KB的分区，

第一个分区和第四个分区合并，成为起始地址为20 K、大小为380 KB的分区，剩余3个空闲分区。

1．某计算机按字节编址，其动态分区内存管理采用最佳适应算法，每次分配和回收内存后都对空闲分区链重新排序。当前空闲分区信息如表4-2所示。

表4-2 当前空闲分区信息

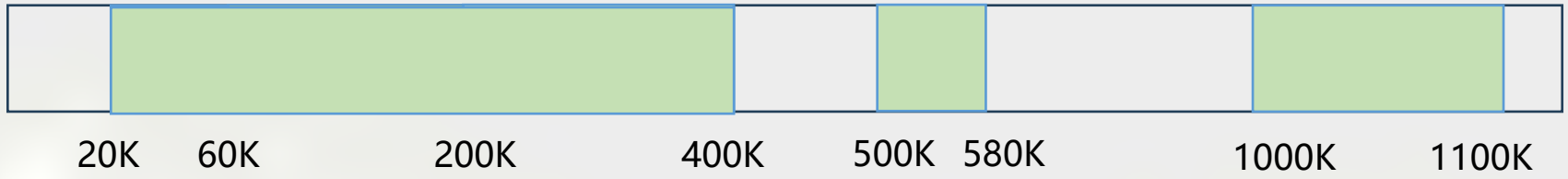
分区起始地址	20K	500K	1000K	200K
分区大小	40KB	80KB	100KB	200KB

回收起始地址为60 K、大小为140 KB的分区后，系统中空闲分区的数量、空闲分区链第一个分区的起始地址和大小分别是(　)。

A．3, 20 K, 380 KB　　B．3, 500 K, 80 KB

C．4, 20 K, 180 KB　　D．4, 500 K, 80 KB

B【解析】根据题意，得到回收前的内存分配情况如下所示：【深色区域是空闲区】

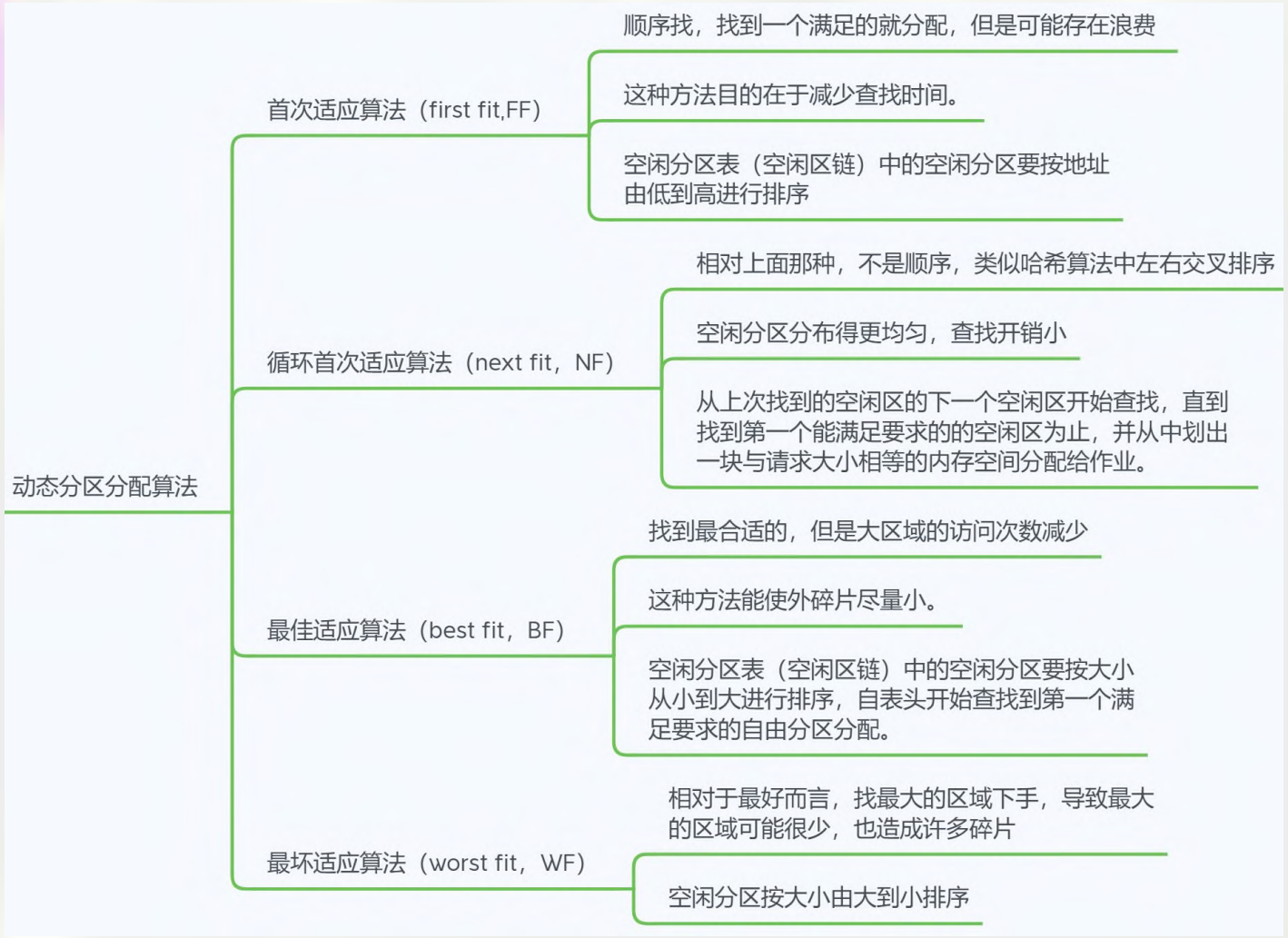


在回收内存后，算法会对空闲分区链按分区大小从小到大进行排序，

分区起始地址	500K	1000K	20K
分区大小	80KB	100KB	380KB

2．在空白表中，空白区按其长度由小到大进行查找的算法称为(　)算法。

A．最佳适应　　B．最差适应　　C．最先适应　　D．先进先出


动态分区分配算法
首次适应算法（first fit,FF）
顺序找，找到一个满足的就分配，但是可能存在浪费
这种方法目的在于减少查找时间。
空闲分区表（空闲区链）中的空闲分区要按地址由低到高进行排序
循环首次适应算法（next fit，NF）
相对上面那种，不是顺序，类似哈希算法中左右交叉排序
空闲分区分布得更均匀，查找开销小
从上次找到的空闲区的下一个空闲区开始查找，直到找到第一个能满足要求的的空闲区为止，并从中划出一块与请求大小相等的内存空间分配给作业。
最佳适应算法（best fit，BF）
找到最合适的，但是大区域的访问次数减少
这种方法能使外碎片尽量小。
空闲分区表（空闲区链）中的空闲分区要按大小从小到大进行排序，自表头开始查找到第一个满足要求的自由分区分配。
最坏适应算法（worst fit，WF）
相对于最好而言，找最大的区域下手，导致最大的区域可能很少，也造成许多碎片
空闲分区按大小由大到小排序

2．在空白表中，空白区按其长度由小到大进行查找的算法称为(　)算法。

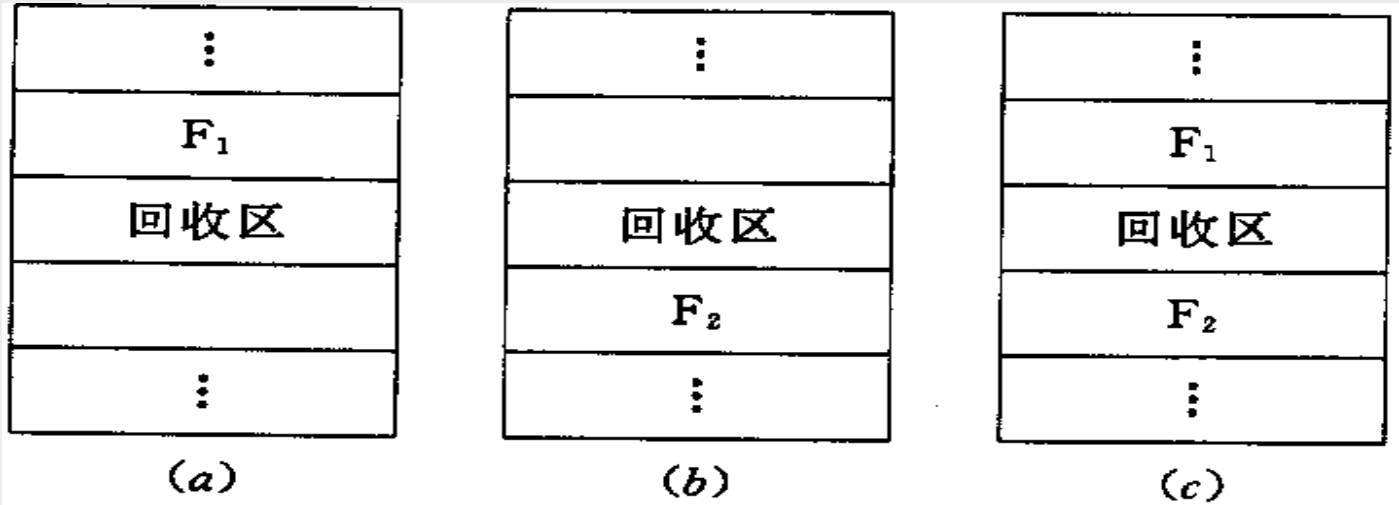
A．最佳适应　　B．最差适应　　C．最先适应　　D．先进先出

A【**解析**】所谓最佳适应算法是指每次为作业分配内存时，总是把能满足要求又是最小的空闲分区分配给作业，避免“大材小用”。为了加速寻找，该算法要求将所有的空闲分区按其容量以从小到大的顺序形成一个空闲分区链。这样第一次找到的能满足要求的空闲区必然是最佳的。

3．【广东工业大学 2017】在可变分区分配方案中，某一进程完成后，系统回收其主存空间，并与相邻空闲区合并，为此需修改空闲区表，造成空闲区数减1的情况是(　　)。

A．无上邻空闲区也无下邻空闲区　　B．有上邻空闲区但无下邻空闲区

C．有下邻空闲区但无上邻空闲区　　D．有上邻空闲区也有下邻空闲区



- 情况1：空闲区节点中，地址不变，大小修改即可。要把回收区的大小加进来。
- 情况2：该节点地址要变，变成回收区的地址，然后大小也要修改。
- 情况3：把回收区和空闲区加起来，这里会导致空闲区少一个。

3．【广东工业大学 2017】在可变分区分配方案中，某一进程完成后，系统回收其主存空间，并与相邻空闲区合并，为此需修改空闲区表，造成空闲区数减1的情况是(　　)。

A．无上邻空闲区也无下邻空闲区　　B．有上邻空闲区但无下邻空闲区

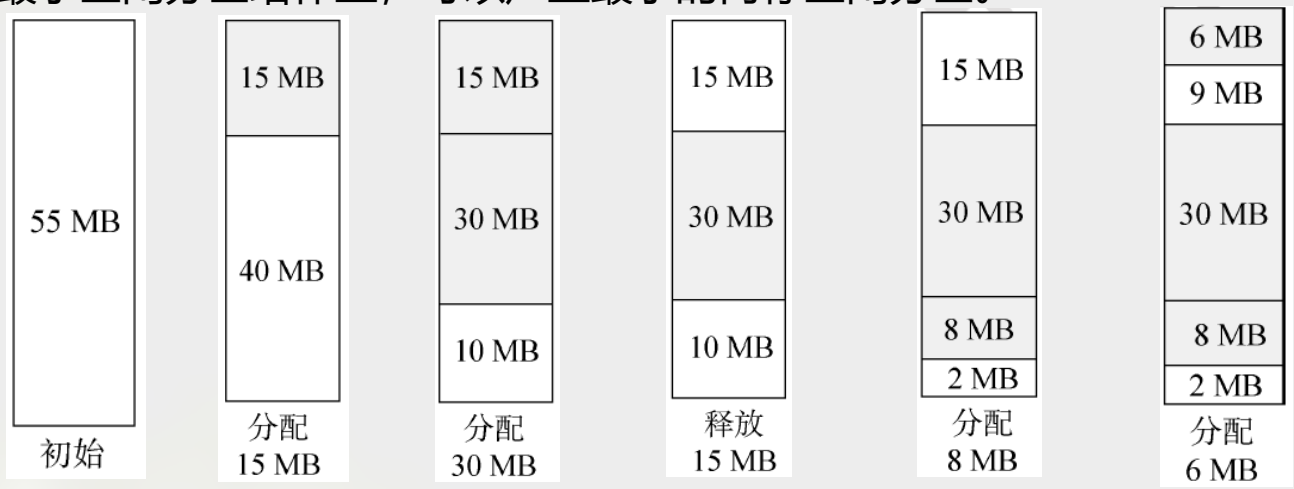
C．有下邻空闲区但无上邻空闲区　　D．有上邻空闲区也有下邻空闲区

D【**解析**】将上邻空闲区、下邻空闲区和回收区合并为一个空闲区，空闲区数反而减少一个。仅有上邻空闲区或下邻空闲区时，空闲区数不会减少。

4．【中国计量大学　2018】某基于动态分区存储管理的计算机，其主存容量为55 MB(初始为空)，采用最佳适配(Best Fit)算法，分配和释放的顺序为分配15 MB，分配30 MB，释放15 MB，分配8 MB，分配6 MB，此时主存中最大空闲分区的大小是(　)。

A．7 MB　　B．9 MB　　C．10 MB　　D．15 MB

B【**解析**】最佳适配算法是指每次为作业分配内存空间时，总是找到能满足空间大小需要的最小空闲分区给作业，可以产生最小的内存空闲分区。



图中灰色部分为分配的空间，白色部分为空闲区。容易发现，此时主存中最大空闲分区的大小为9 MB。

【牛刀小试】

1．【广东工业大学 2014】在循环首次适应算法中，要求空闲分区按(　　)顺序链接成空闲分区链。

A．空闲区首址递增　　B．空闲区首址递减

C．空闲区大小递增　　D．空闲区大小递减

A【**解析**】循环首次适应(NF)算法：将所有空闲分区按照地址递增的次序链接，在申请内存分配时，总是从上次找到的空闲分区的下一个空闲分区开始查找，将满足需求的第一个空闲分区分配给作业，因此选择A。

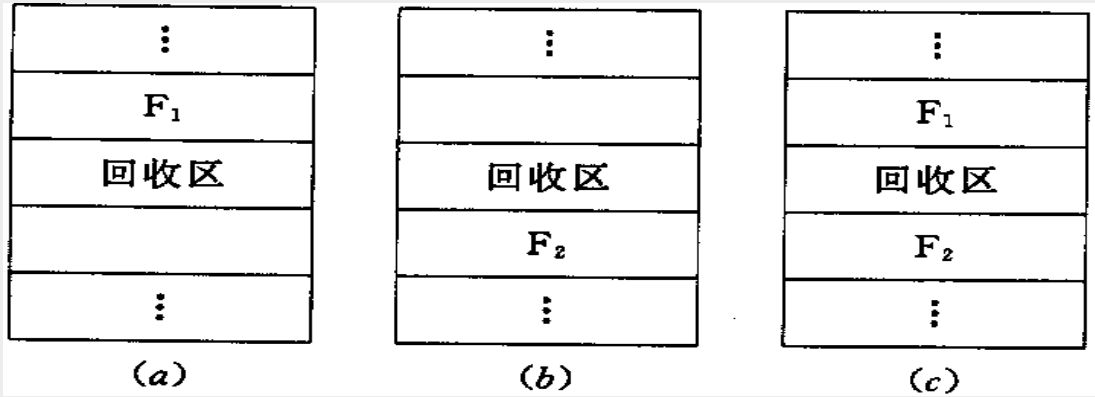
2．【广东工业大学　2014】在回收内存时如果出现下述情况：释放区与插入点的后一分区F2相邻接，此时应(　　)。

A．为回收区建立一分区表项，填上分区的大小和始址

B．以F1分区的表项作为新表项且不做任何改变

C．以F1分区的表项作为新表项，修改新表项的大小

D．以F2分区的表项作为新表项，同时修改新表项的大小和始址



- 情况1：空闲区节点中，地址不变，大小修改即可。要把回收区的大小加进来。
- 情况2：该节点地址要变，变成回收区的地址，然后大小也要修改。
- 情况3：把回收区和空闲区加起来，这里会导致空闲区少一个。

2．【广东工业大学　2014】在回收内存时如果出现下述情况：释放区与插入点的后一分区F2相邻接，此时应(　　)。

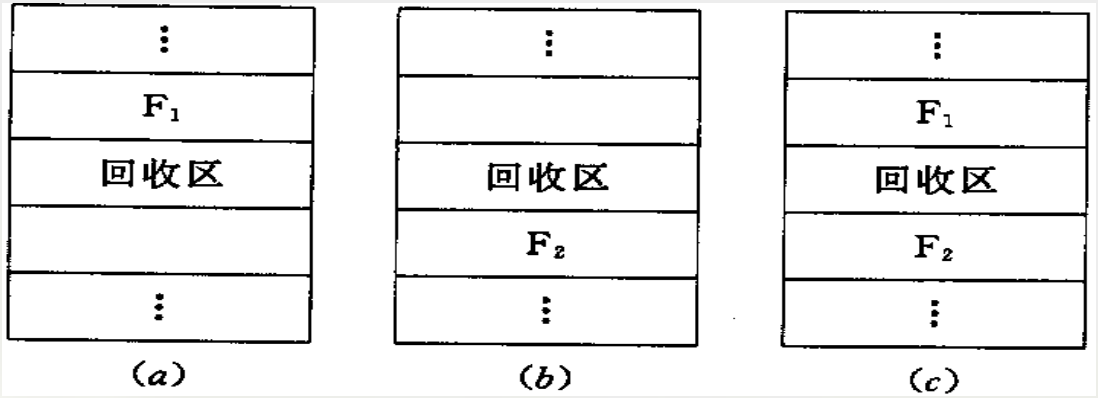
A．为回收区建立一分区表项，填上分区的大小和始址

B．以F1分区的表项作为新表项且不做任何改变

C．以F1分区的表项作为新表项，修改新表项的大小

D．以F2分区的表项作为新表项，同时修改新表项的大小和始址

D【**解析**】在回收内存时，当释放区与插入点后一分区F2相邻接，此时应以F2分区的表项作为新表项，同时修改新表项的大小和起始地址，因此选择D。



3．【南京理工大学 2018】采用可变分区存储管理的系统中，(　　)分区分配算法最有可能在内存高地址端保留大分区。

A．首次适应　　B．最佳适应

C．最差适应　　D．循环首次适应

A【**解析**】

首次适应算法每次从头开始按空闲区地址递增顺序分配，在高地址保留大的分区；

循环首次适应按空闲区地址递增顺序分配，每次从当前位置进行比较；

最佳适应按空闲区大小递增顺序分配；

最差适应按空闲区大小递减顺序分配，因此选择A。

4．【南京工业大学 2016】分区管理要求对每一个作业都分配(　)的内存单元。

A．地址连续　　B．若干地址不连续

C．若干连续的帧　　D．若干不连续的帧

A【**解析**】分区式存储管理方式是连续分配的方式，就是对每一个作业都分配一组地址连续的内存单元，因此选择A。

5．【北京交通大学 2016】对于基于空闲分区链的动态分区存储管理方式，当进程运行完毕释放内存时，系统应根据回收区的起始地址从空闲分区链中找到相应的插入位置，然后实施回收插入操作。如下处理方案中存在问题的是(　)。

A．检查确认，若回收区仅与插入位置前的空闲分区F_1相邻接，应将回收区与F_1合并，即修正F_1的大小（加上回收区大小）

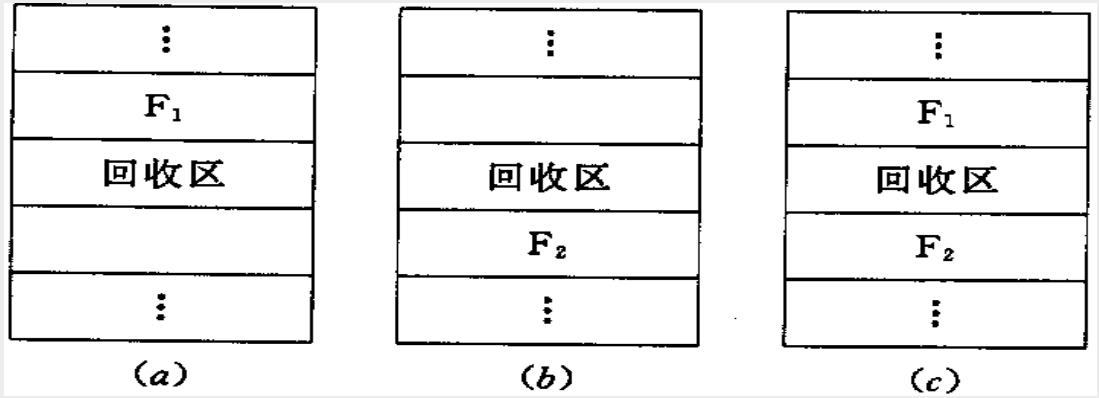
B．检查确认，若回收区仅与插入位置后的空闲分区F_2相邻接，应将回收区与F_2合并，即修正F_2的大小（加上回收区大小）

C．检查确认，若回收区同时与插入位置前、后的空闲分区F_1和F_2相邻接，应将回收区与F_1，F_2合并，即修正F_1的大小（加上回收区大小和F_2大小），并删除F_2表项节点

D．检查确认，若回收区与插入位置前、后的空闲分区F_1和F_2均不相邻接，应为回收区单独建立新表项，填写起始地址和大小，并插入到空闲分区链

5．【北京交通大学 2016】对于基于空闲分区链的动态分区存储管理方式，当进程运行完毕释放内存时，系统应根据回收区的起始地址从空闲分区链中找到相应的插入位置，然后实施回收插入操作。如下处理方案中存在问题的是(　)。

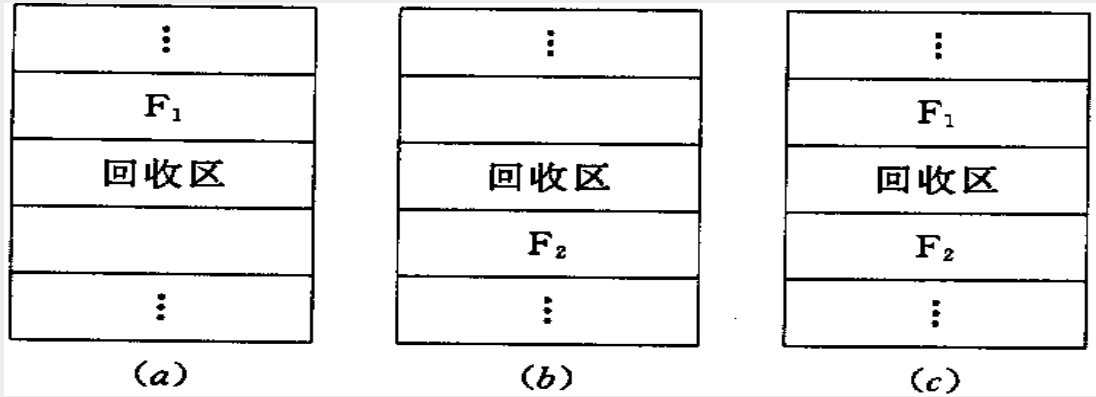
A．检查确认，若回收区仅与插入位置前的空闲分区F_1相邻接，应将回收区与F_1合并，即修正F_1的大小（加上回收区大小）



- 情况1：空闲区节点中，地址不变，大小修改即可。要把回收区的大小加进来。
- 情况2：该节点地址要变，变成回收区的地址，然后大小也要修改。
- 情况3：把回收区和空闲区加起来，这里会导致空闲区少一个。
- 在回收内存时，释放区与插入点前一分区F_1相邻接，此时应以F_1分区的表项为新表项，且修改新表项的大小，A正确；

5．【北京交通大学 2016】对于基于空闲分区链的动态分区存储管理方式，当进程运行完毕释放内存时，系统应根据回收区的起始地址从空闲分区链中找到相应的插入位置，然后实施回收插入操作。如下处理方案中存在问题的是(　)。

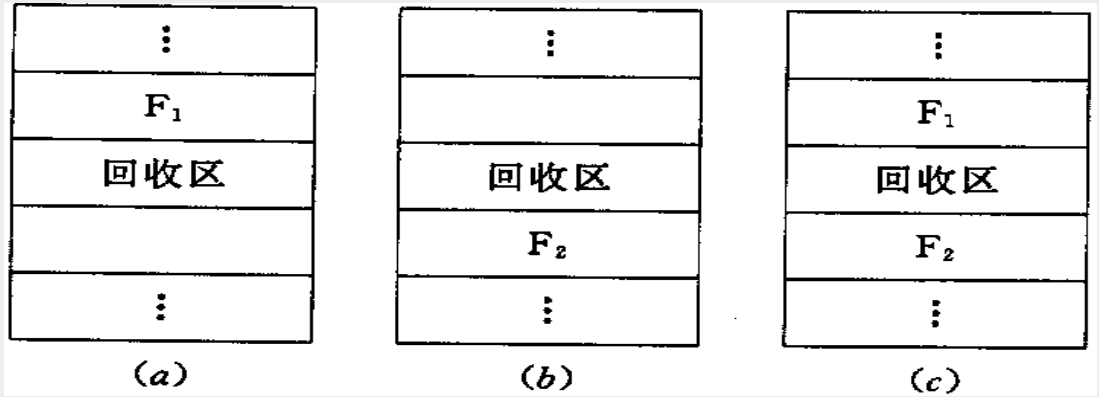
B．检查确认，若回收区仅与插入位置后的空闲分区F_2相邻接，应将回收区与F_2合并，即修正F_2的大小（加上回收区大小）



- 情况1：空闲区节点中，地址不变，大小修改即可。要把回收区的大小加进来。
- 情况2：该节点地址要变，变成回收区的地址，然后大小也要修改。
- 情况3：把回收区和空闲区加起来，这里会导致空闲区少一个。
- 释放区与插入点后一分区F_2相邻接，此时应以F_2分区的表项作为新表项，同时修改新表项的大小和始址，B错误

5．【北京交通大学 2016】对于基于空闲分区链的动态分区存储管理方式，当进程运行完毕释放内存时，系统应根据回收区的起始地址从空闲分区链中找到相应的插入位置，然后实施回收插入操作。如下处理方案中存在问题的是(　)。

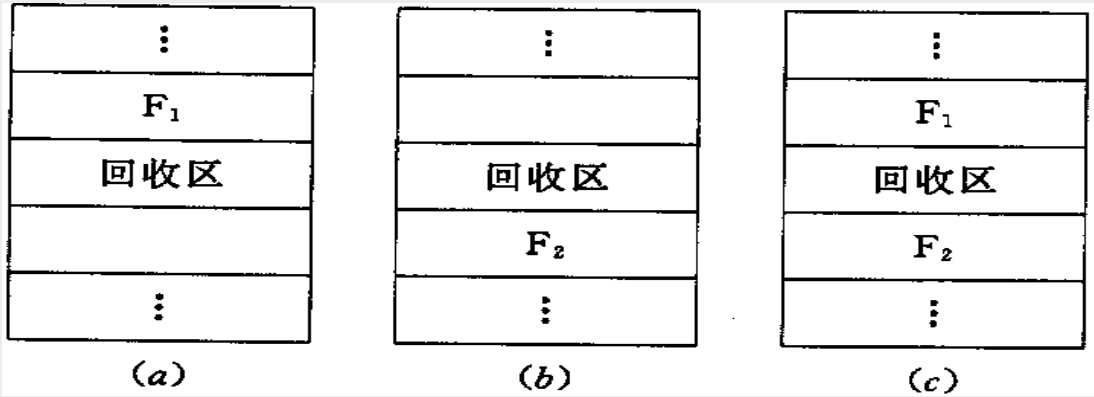
C．检查确认，若回收区同时与插入位置前、后的空闲分区F_1和F_2相邻接，应将回收区与F_1，F_2合并，即修正F_1的大小（加上回收区大小和F_2大小），并删除F_2表项节点



- 情况1：空闲区节点中，地址不变，大小修改即可。要把回收区的大小加进来。
- 情况2：该节点地址要变，变成回收区的地址，然后大小也要修改。
- 情况3：把回收区和空闲区加起来，这里会导致空闲区少一个。
- 释放区既与F_1相邻接，又与F_2相邻接，此时应以F_1分区的表项为新表项，且修改新表项的大小，并删除F_2所对应的表项，C正确；

5．【北京交通大学 2016】对于基于空闲分区链的动态分区存储管理方式，当进程运行完毕释放内存时，系统应根据回收区的起始地址从空闲分区链中找到相应的插入位置，然后实施回收插入操作。如下处理方案中存在问题的是(　)。

D．检查确认，若回收区与插入位置前、后的空闲分区F_1和F_2均不相邻接，应为回收区单独建立新表项，填写起始地址和大小，并插入到空闲分区链

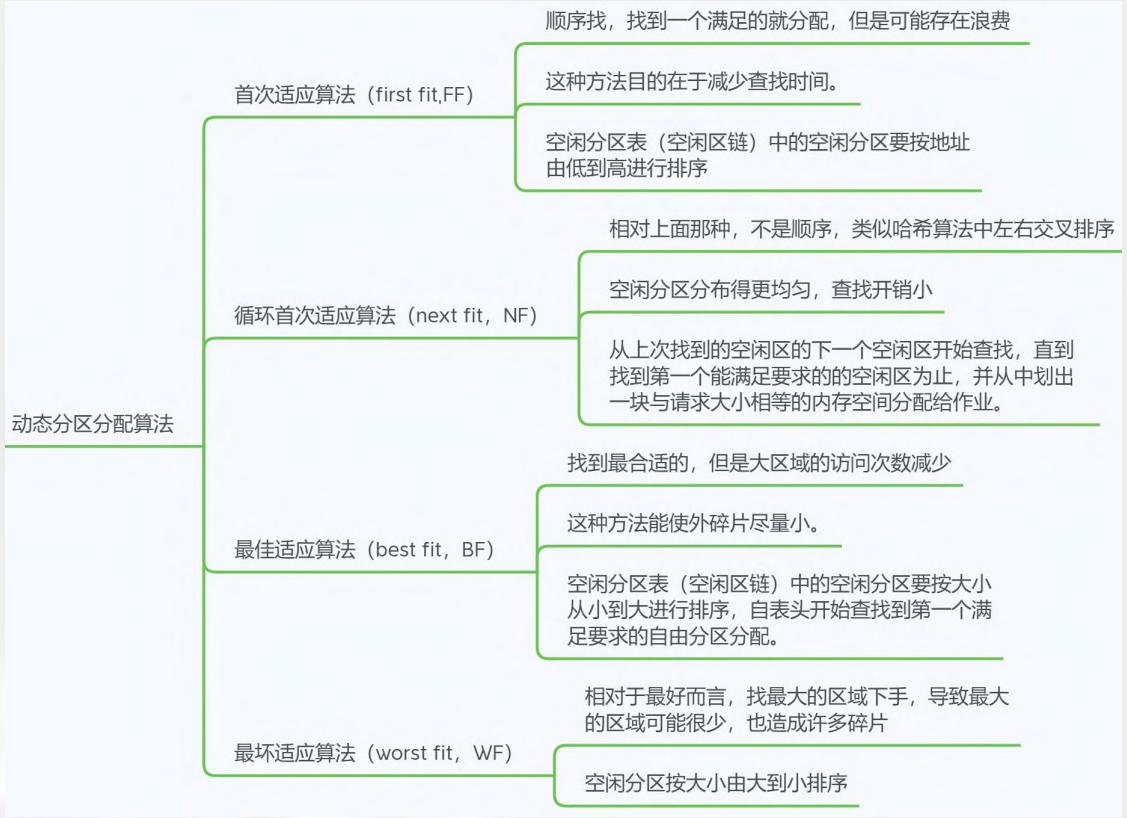


- 情况1：空闲区节点中，地址不变，大小修改即可。要把回收区的大小加进来。
- 情况2：该节点地址要变，变成回收区的地址，然后大小也要修改。
- 情况3：把回收区和空闲区加起来，这里会导致空闲区少一个。
- 释放区不与F_1和F_2相邻接，此时应为回收区建立一分区表项，填上分区的大小和始址，D正确，

6．分区管理中采用最佳适应分配算法时，把空闲区按(　)次序登记在空闲区表中。

A．长度递增　　B．长度递减　　C．地址递增　　D．地址递减

A【**解析**】分区管理中采用最佳适应分配算法时，把空闲区按长度递增次序登记在空闲区表中，因此选择A。



【经典总结】

(1)首次适应(FF)算法以空闲链的首地址递增顺序组织起来，当提出分配需求时，遍历组织好的空白链，找到第一个空间大于等于分配需求的空白分配块分配。

若遍历一遍都未找到满足需求的空白块，则分配失败。

- 优点：该算法倾向于优先利用低地址部分的空闲块，从而保留了高地址部分的空闲块，则高地址部分就有可能留有大容量的内存块，为大需求的作业创造了条件。
- 缺点：该算法每次都是从低地址找起，导致其低地址留下了许多无法使用的外部碎片，降低了后续查找的效率。

【经典总结】

(2)循环首次适应(NF)算法在FF算法的基础上，针对其查找效率低的缺点进行改进，不改变空白链的组织方式，只改变查找方式，每次查找的起始位置是从上次查找位置的下一个位置开始，而不是从头查找，当循环查找一遍之后仍未找到满足需求的空白内存块时分配失败，这样就避免了对外部碎片的查找浪费。

- 优点：使内存分配在内存中更加均匀，相对于首次适应算法来说查找效率更高。
- 缺点：由于分配均匀，使得内存中缺乏大的空闲内存块，当后续出现大内存需求的作业时无法满足。

【经典总结】

(3)最佳适应(BF)算法，“最佳”指的是大小最合适、最接近。空白链以容量大小的顺序组织，每次遍历空白链，查找第一个能满足需求的空白块进行分配，这样就一定程度上减少了外部碎片的大小，也避免了“大材小用”。

- 优点：每次分割剩余的空间总是最小的，减少了外部碎片的大小。
- 缺点：从空间利用率的角度来看，BF算法确实是物尽其用，但是每次分割留下的都是难以利用的外部碎片，降低了查找效率。

【经典总结】

(4)最坏适应(WF)算法是将空白区按照容量递减，在扫描整个空链表时，总是挑选一个最大空闲块，从中分割需求的内存块。实际上，这样的算法未必是最坏的。

- 优点：可使剩下的空闲区不至于太小，产生外部碎片的可能性更小，对中小作业有利。查找效率高，只找最大的，若最大的不满足，就直接失败。
- 缺点：可能会导致空白存储链中缺少大容量的空白内存块，当大容量作业进入时无法满足。

7．在动态分区式内存管理中，能使内存空间中空闲区分布较均匀的算法是(　)。

A．最佳适应算法　　B．最坏适应算法

C．首次适应算法　　D．循环首次适应算法

D【**解析**】循环首次适应算法按空闲区地址递增顺序分配，每次从当前位置开始。在动态分区式内存管理中，能使内存空间中空闲区分布较均匀的算法是循环首次适应算法，因此选择D。

8．某系统的空闲分区表如表所示。

分区号	大小 / KB	起始地址/KB
1	32	100
2	10	150
3	5	200
4	218	220
5	96	530

采用可变式分区管理策略，现有如下作业序列：96 KB，20 KB，200 KB。若用首次适应算法和最佳适应算法来处理这些作业序列，则(　　)该作业序列请求。

A．首次适应算法能满足，最佳适应算法不能满足

B．首次适应算法不能满足，最佳适应算法能满足

C．都能满足

D．都不能满足

B【**解析**】首次适应算法按空闲区地址递增顺序，96 KB申请分区4号，剩余122 KB，20 KB申请分区1号，剩余12KB

申请200 KB无法满足；

分区号	大小 / KB	起始地址/KB
1	12KB	100
2	10	150
3	5	200
4	122KB	220
5	96	530

8．某系统的空闲分区表如表所示。

分区号	大小 / KB	起始地址/KB
1	32	100
2	10	150
3	5	200
4	218	220
5	96	530

采用可变式分区管理策略，现有如下作业序列：96 KB，20 KB，200 KB。若用首次适应算法和最佳适应算法来处理这些作业序列，则(　　)该作业序列请求。

A．首次适应算法能满足，最佳适应算法不能满足

B．首次适应算法不能满足，最佳适应算法能满足

C．都能满足

D．都不能满足

B【**解析**】最佳适应算法按空闲区大小递增顺序，96 KB申请分区5号，恰好分配，

20 KB申请分区1号，剩余12 KB，

200 KB申请分区4号，剩余18 KB，

故首次适应算法不能满足，最佳适应算法能满足，

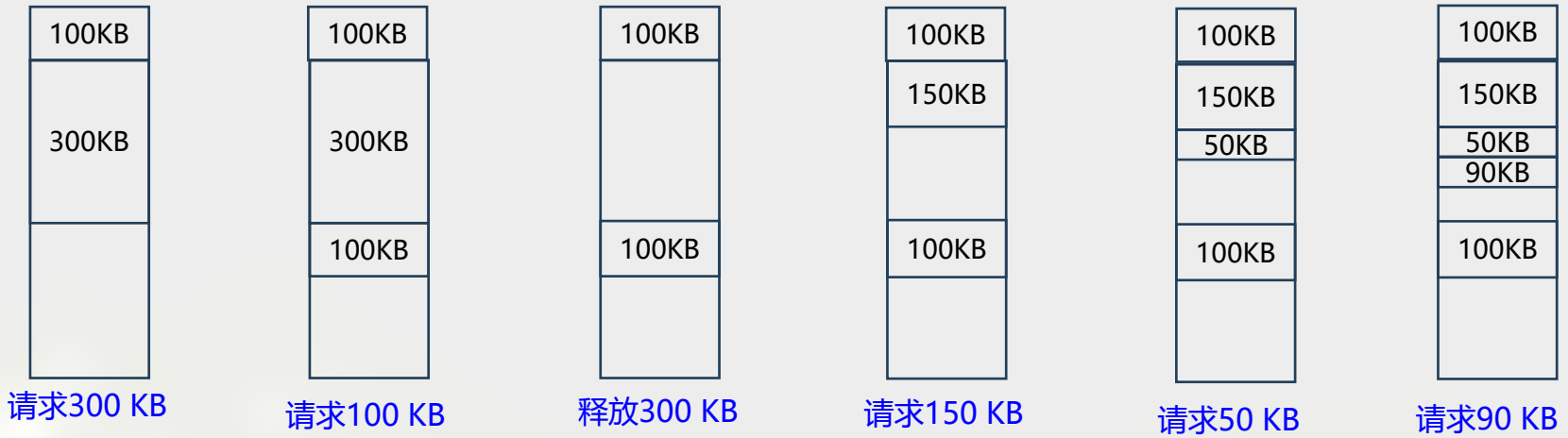
因此选择B。

分区号	大小 / KB	起始地址/KB
1	12KB	100
2	10	150
3	5	200
4	18KB	220
5	0KB	530

9．某操作系统采用动态分区存储管理技术。操作系统在低地址占用了100 KB的空间，用户区主存从100 KB处开始占用512 KB。初始时，用户区全部为空闲，分配时截取空闲分区的低地址部分作为已分配区。在执行以下申请、释放操作序列后：请求300 KB、请求100 KB、释放300 KB、请求150 KB、请求50 KB、请求90 KB，回答以下问题。

(1)采用首次适应算法时，主存中有哪些空闲分区？画出主存分布图，并指出空闲分区的首地址和大小。

(1)首次适应算法按空闲区地址递增顺序分配，主存分布图如图4-9所示。

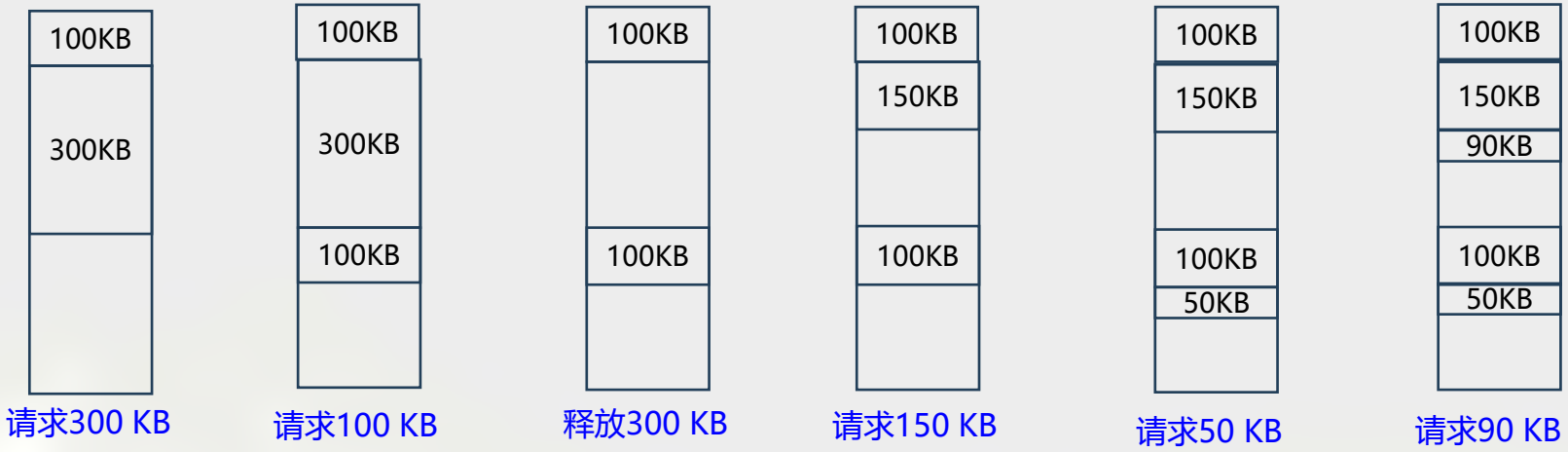


- 空闲区1的首地址为100 KB+150 KB+50 KB+90 KB=390 KB，大小为10 KB；
- 空闲区2的首地址为100 KB+290 KB+10 KB+100 KB=500 KB，大小为112KB。

9．某操作系统采用动态分区存储管理技术。操作系统在低地址占用了100 KB的空间，用户区主存从100 KB处开始占用512 KB。初始时，用户区全部为空闲，分配时截取空闲分区的低地址部分作为已分配区。在执行以下申请、释放操作序列后：请求300 KB、请求100 KB、释放300 KB、请求150 KB、请求50 KB、请求90 KB，回答以下问题。

(2)采用最佳适应算法时，主存中有哪些空闲分区？画出主存分布图，并指出空闲分区的首地址和大小。

(2)最佳适应算法按空闲区大小递增顺序分配，主存分布图如图所示。



- 空闲区1的首地址为100 KB+150 KB+90 KB=340 KB，大小为60 KB；
- 空闲区2的首地址为100 KB+150 KB+90 KB+60 KB+100 KB+50 KB=550 KB，大小为62 KB。

9．某操作系统采用动态分区存储管理技术。操作系统在低地址占用了100 KB的空间，用户区主存从100 KB处开始占用512 KB。初始时，用户区全部为空闲，分配时截取空闲分区的低地址部分作为已分配区。在执行以下申请、释放操作序列后：请求300 KB、请求100 KB、释放300 KB、请求150 KB、请求50 KB、请求90 KB，回答以下问题。

(3)若随后又要请求80 KB，针对上述两种情况会产生什么后果？说明了什么问题？

(3)首次适应算法：将112 KB空闲分区分配给80 KB，首次适应算法满足请求；

最佳适应算法：空闲区都比80 KB小，最佳适应算法无法满足请求。

说明首次适应算法在某些情况下比最佳适应算法效果好。

谢谢大家