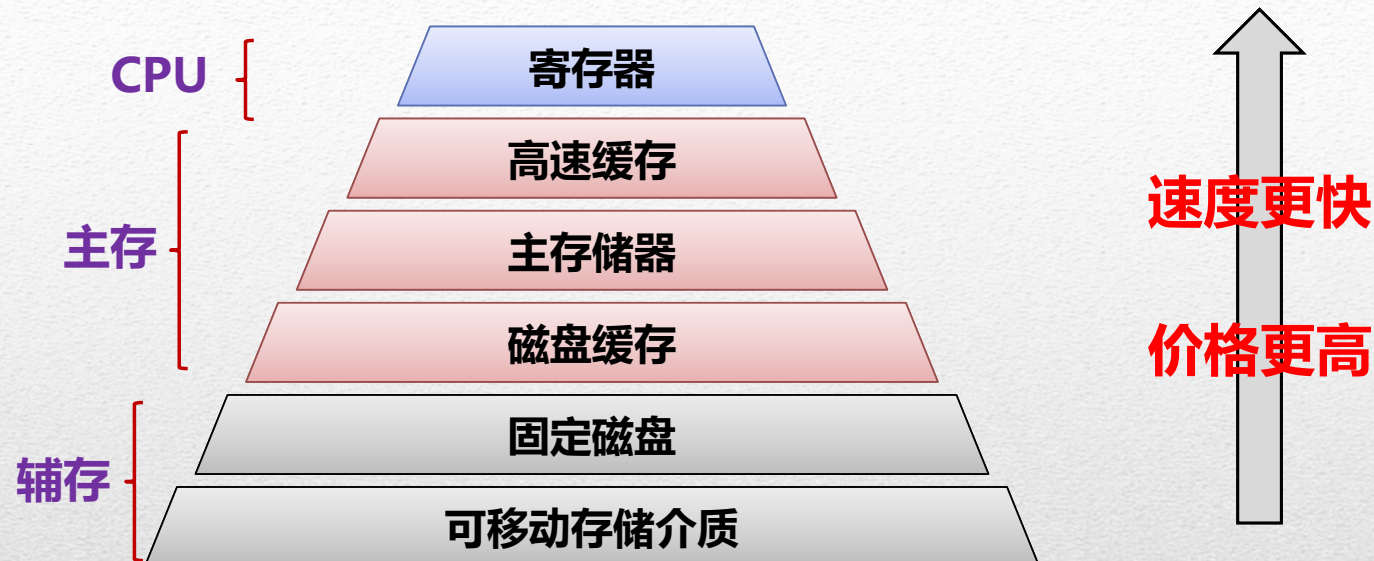


# 第三章

## 存储管理

方 钰

---



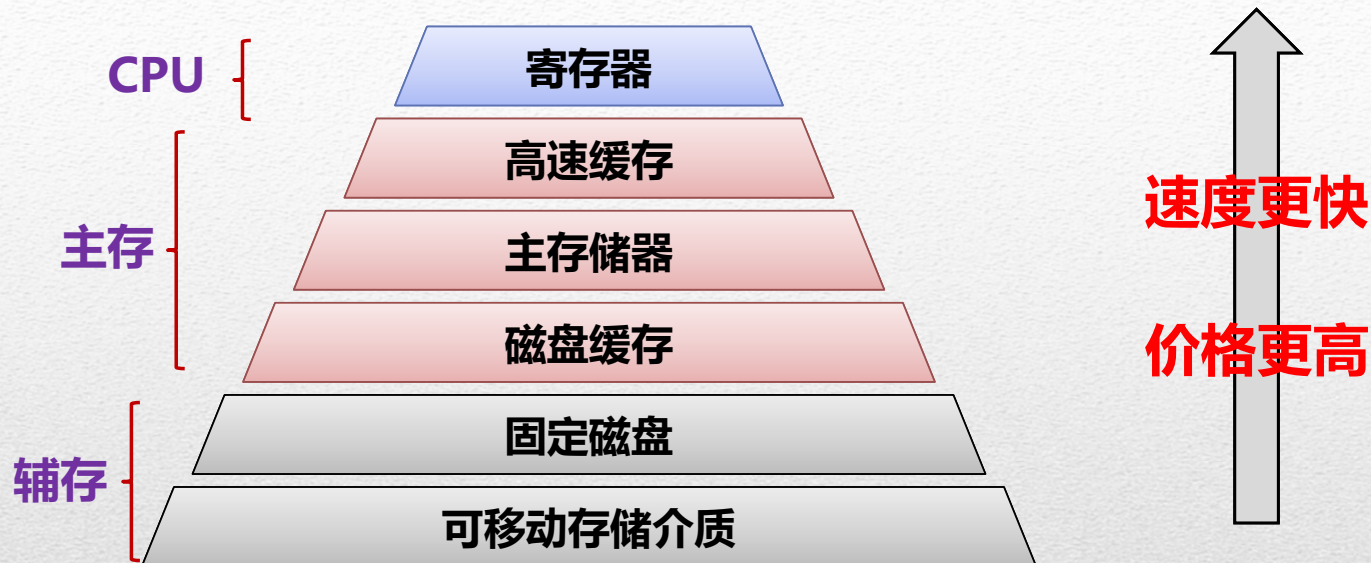
**寄存器：** 速度最快，能与CPU协调工作。

**高速缓存：** 根据程序执行的局部性原理建立。可多级。

**主存储器：** 用于保存程序运行的指令和数据，可执行存储器。

**磁盘缓存：** 利用主存空间，缓存磁盘信息。



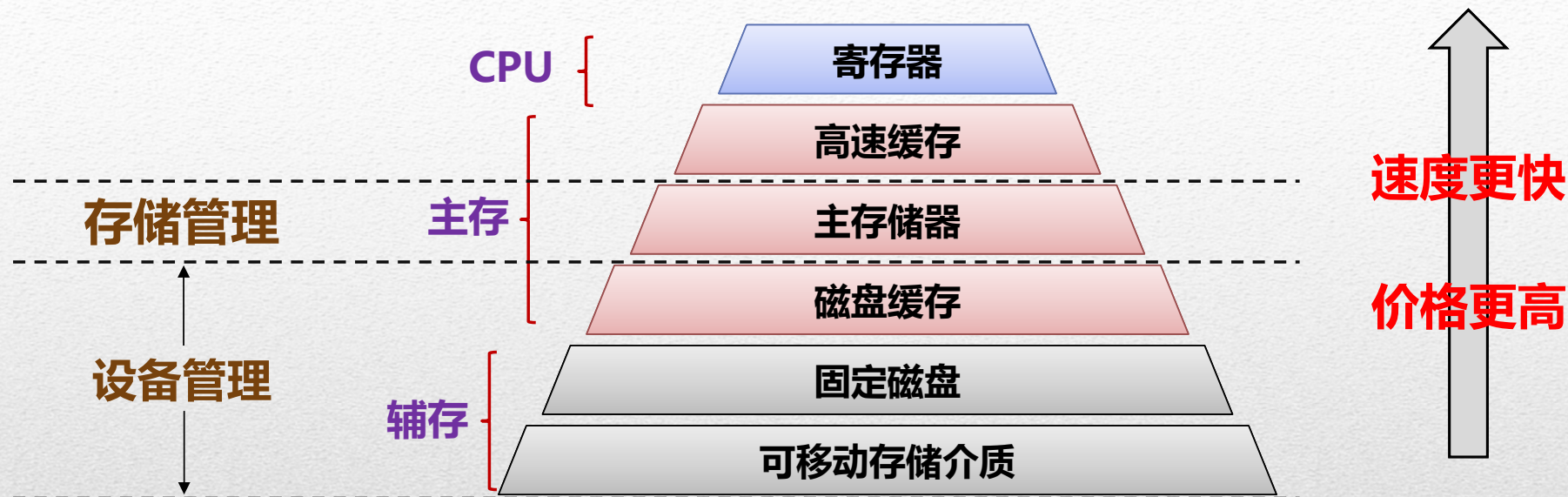


**寄存器：** 速度最快，能与CPU协调工作。

**高速缓存：** 根据程序执行的局部性原理建立。可多级。

**主存储器：** 用于保存程序运行的指令和数据，可执行存储器。

**磁盘缓存：** 利用主存空间，缓存磁盘信息。



**寄存器:** 速度最快，能与CPU协调工作。

**高速缓存:** 根据程序执行的局部性原理建立。可多级。

**主存储器:** 用于保存程序运行的指令和数据，可执行存储器。

**磁盘缓存:** 利用主存空间，缓存磁盘信息。

**存储管理目标:**  
将各级存储组织  
成一个整体





# 主要内容

## 3.1 存储管理的主要任务

## 3.2 连续分配方式

## 3.3 页式存储管理

## 3.4 UNIX 存储管理



# 任务之一：主存空间的管理

## 1. 空闲空间的管理

索引号	起始地址/KB	大小/KB
1	20	10
2	32	15
3	50	20

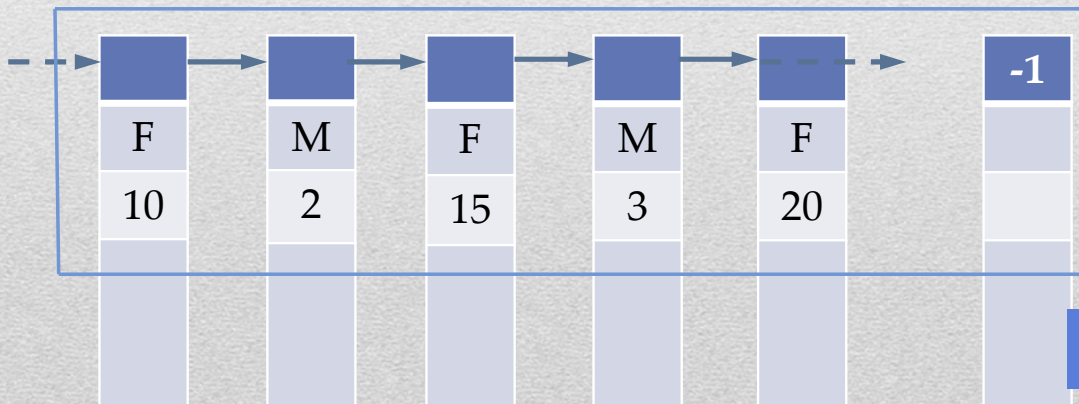
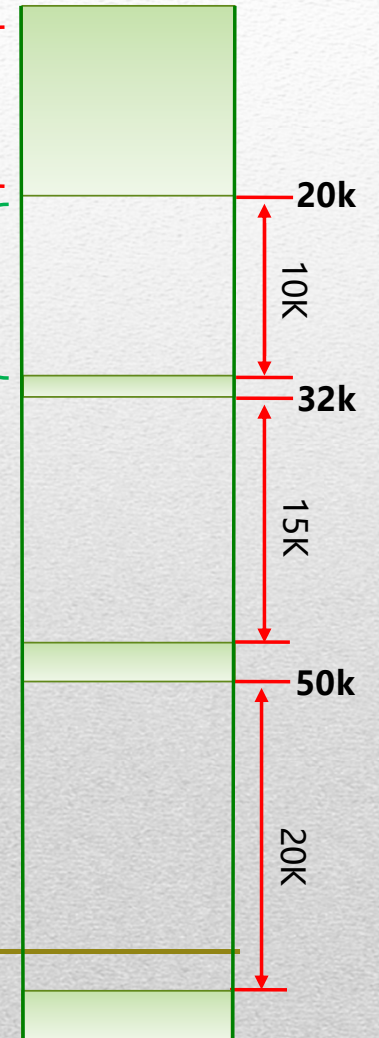
索引表

如果选择1K大小的固定分配单元

7	6	5	4	3	2	1	0
1	1	1	1	1	1	1	1
1	1	1	1	1	1	1	1
0	0	0	0	1	1	1	1
1	1	0	0	0	0	0	0
0	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0
.....	.....	.....	.....	.....	.....	.....	.....

位示图

内存



占用前几个字节的空间  
F: 空闲; M: 占用

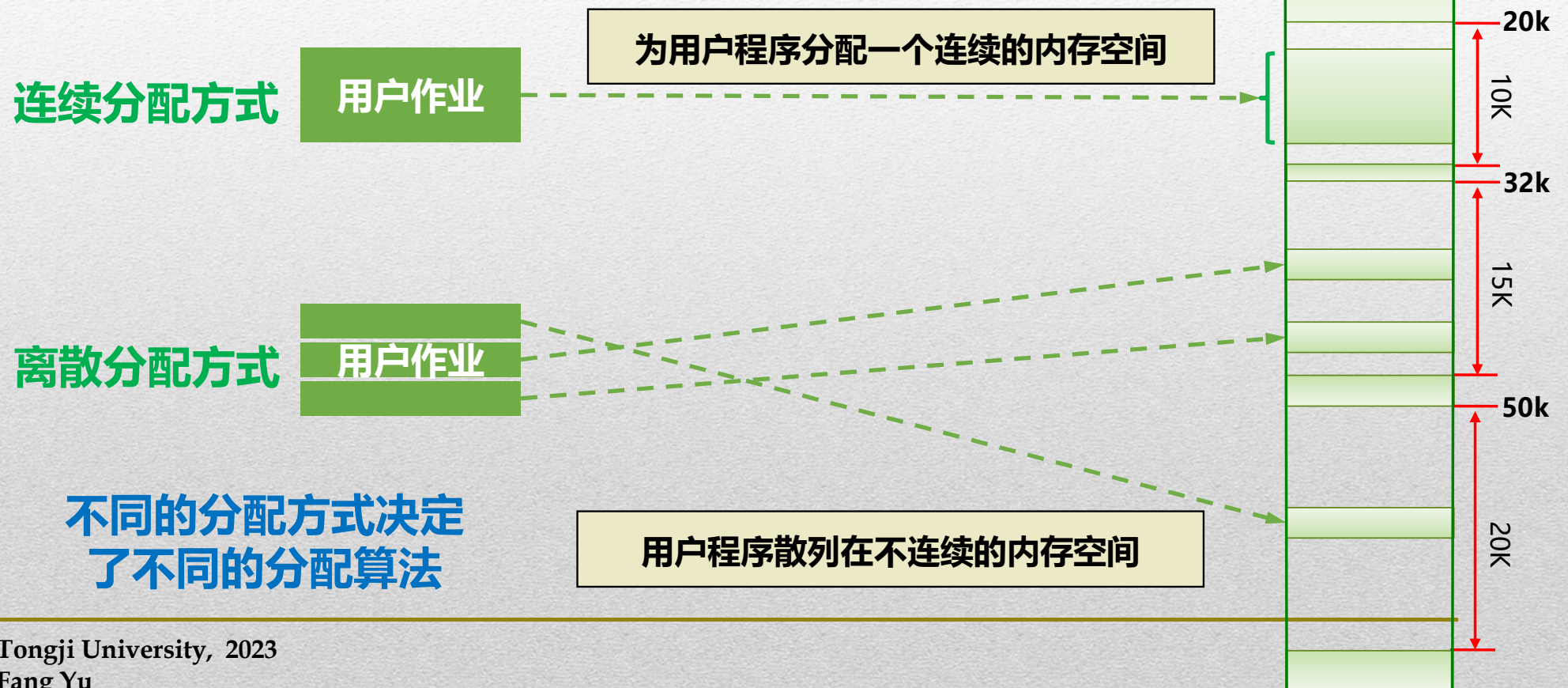
链式队列





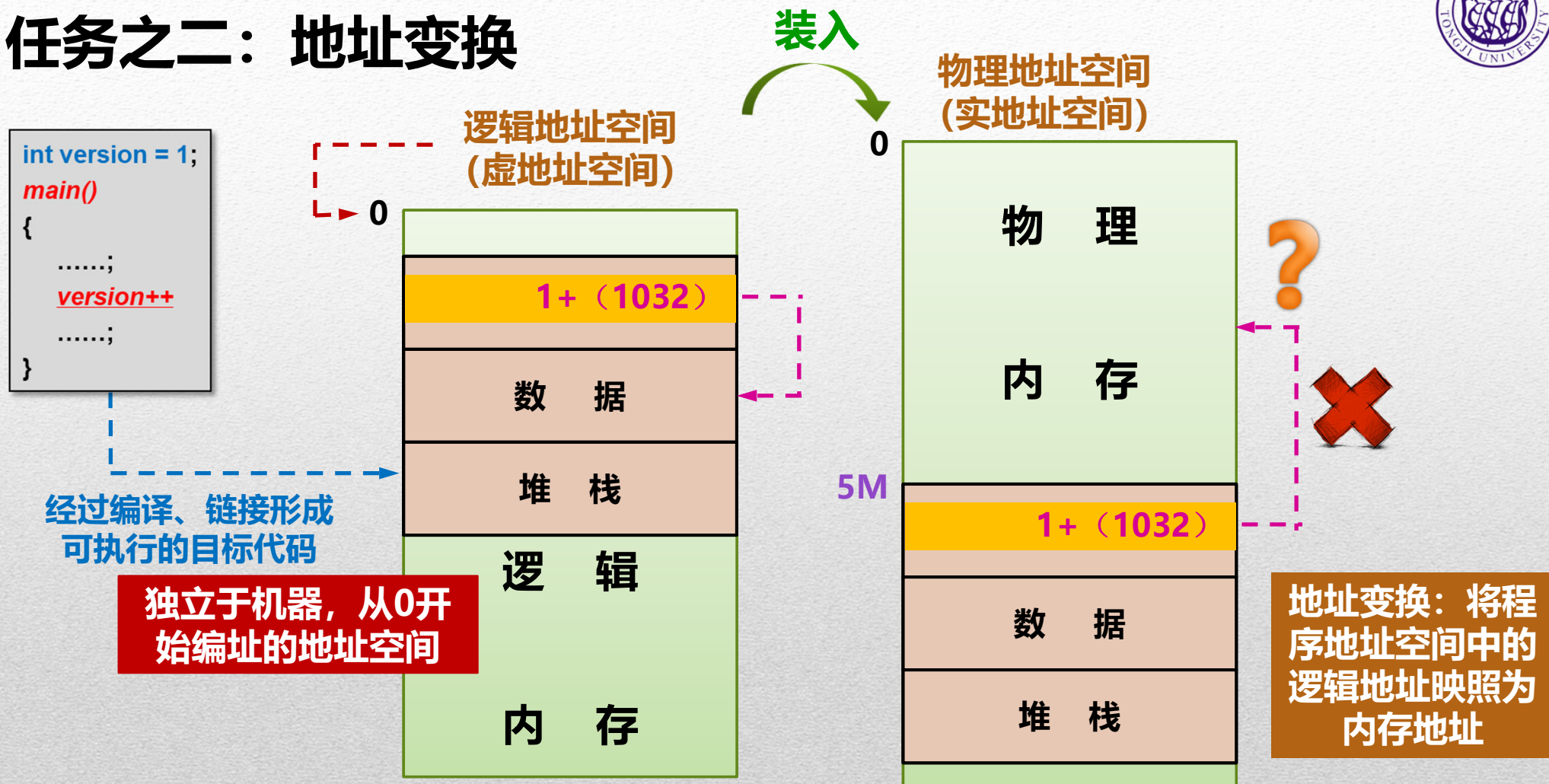
# 任务之一：主存空间的管理

## 2. 空闲空间的分配





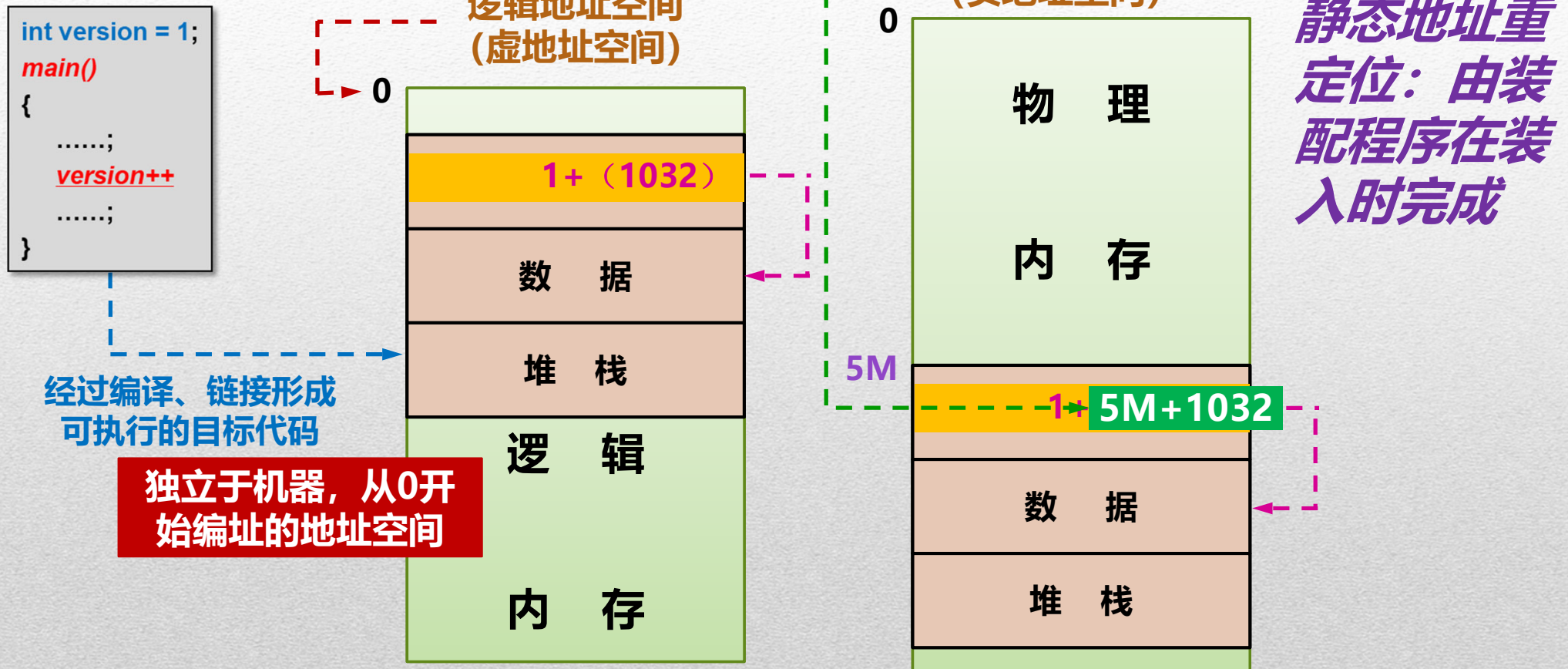
## 任务之二：地址变换





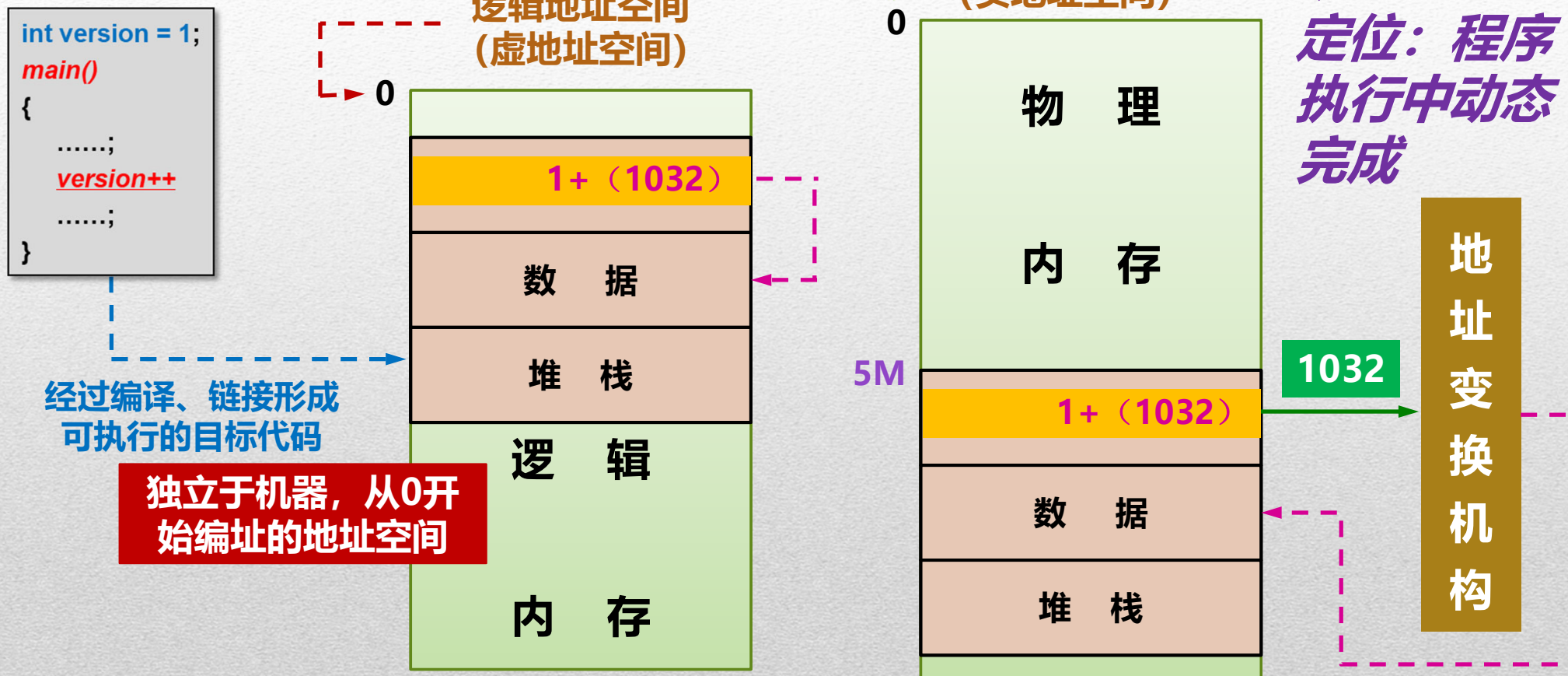


## 任务之二：地址变换





## 任务之二：地址变换







## 任务之二：地址变换

静态重定位	动态重定位
不需要硬件支持	需要专门的硬件支持
软件算法简单	软件算法比较复杂
程序必须占用连续的内存空间	程序不必占用连续的内存空间
程序执行过程中不能移动	程序执行过程可以移动

**如果程序的逻辑地址在执行过程中可能发生改变，则必须使用动态地址重定位。**



## 任务之二：地址变换

静态重定位	动态重定位
不需要硬件支持	需要专门的硬件支持
软件算法简单	软件算法比较复杂
程序必须占用连续的内存空间	程序不必占用连续的内存空间
程序执行过程中不能移动	程序执行过程可以移动

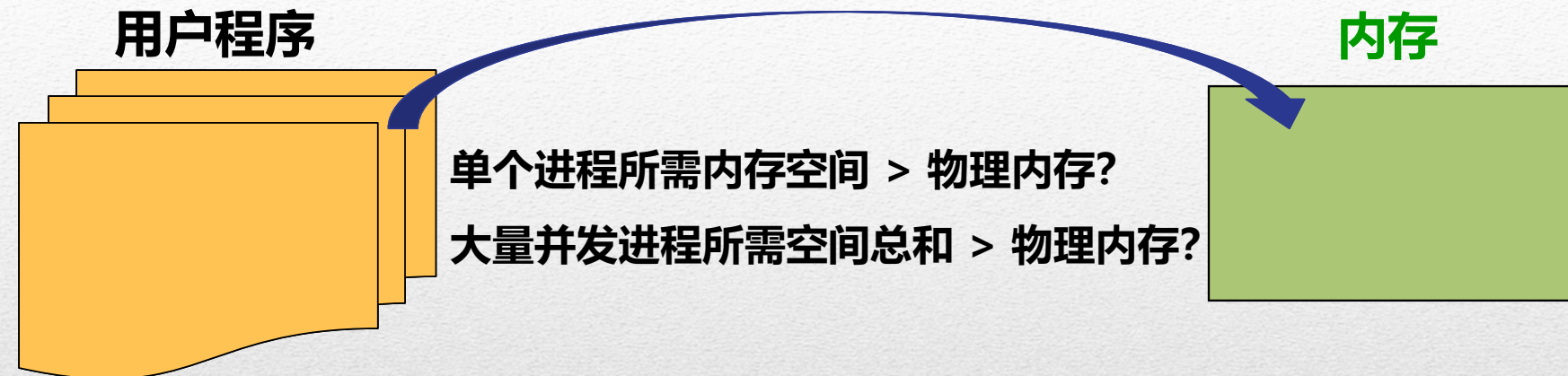
现代操作系统采用  
的地址重定位方式

如果程序的逻辑地址在执行过程中可能发生改变，则必须使用动态地址重定位。





## 任务之三：内存扩充



- 一次性
- 驻留性

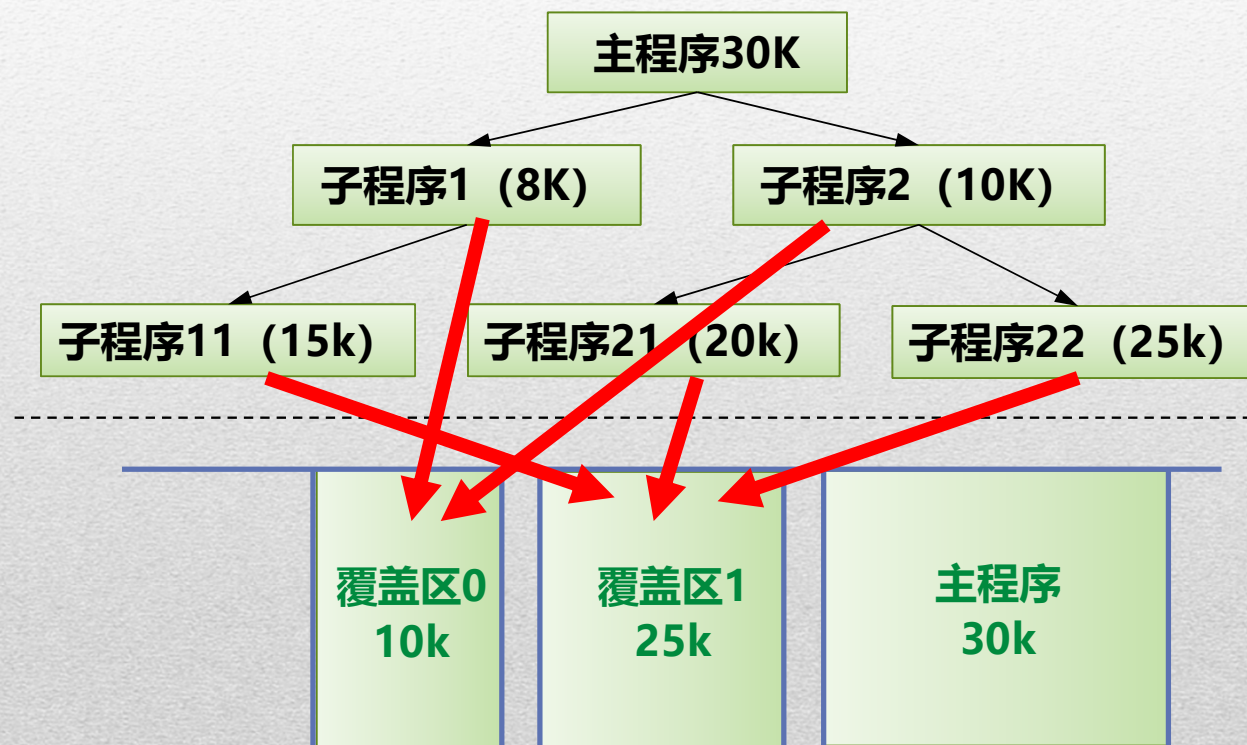
1. 导致程序无法运行
2. 大量暂时不用的程序占用大量内存空间，一些需要运行的作业无法装入

**内存扩充：在逻辑上扩充内存容量，使得小内存满足大需求。**



## 任务之三：内存扩充

### 1. 覆盖：一个作业的若干程序段共享一段主存空间



优点：程序无需一次性全部装入。

缺点：程序员提供模块之间的覆盖关系。

打破一次性!!!





## 任务之三：内存扩充

### 2. 交换：进程的全部或部分图象可在内外存间移动

#### 以进程的全部图象为单位

把内存中暂时不能运行的进程的全部进程图象调出到外存，腾出足够的内存空间，在把具备运行条件的进程调入内存。

UNIX V6++ 采用的内存扩充方式

#### 以进程的部分图象为单位

如：以“页”或“段”为单位。是请求分页和请求分段式存储管理的基础，其目的是为了支持虚拟存储系统。

程序的物理地址在执行的过程中可能发生改变

打破驻留性!!!



## 任务之三：内存扩充

### 3. 虚拟存储器：从逻辑上扩充内存

与覆盖类似的是：一个作业在执行时，**只需一部分地址空间在主存**，另一部分在辅存。在辅存的地址空间将来通过请求调入功能，陆续进入主存时，并有可能覆盖主存中的地址空间（**对程序员透明**）。

与交换类似的是：一个作业在执行时，部分地址空间可以经由置换功能，在内存和外存之间移动（**以“页”或“段”为单位，对程序员透明**）。

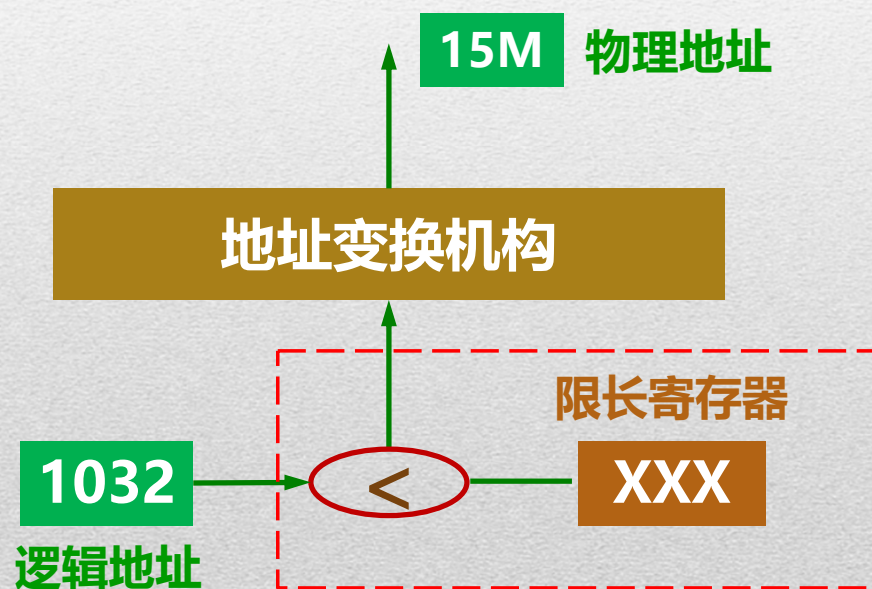
**虚拟存储器**：具有请求调入和置换功能，从逻辑上对内存容量加以扩充。其逻辑容量由内存容量和外存容量之和决定，速度接近于内存速度，成本接近于外存。





## 任务之四：内存共享与保护

- **信息共享**：多个进程可以共享一块内存空间
- **信息保护**：进程只能在分配给它的存储区内活动



软硬件方式实现：

- 限长寄存器
- 上下界寄存器



# 主要内容

## 3.1 存储管理的主要任务

## 3.2 连续分配方式

## 3.3 页式存储管理

## 3.4 UNIX 存储管理

对主存空间的组织与管理

地址变换方式

内存扩充方式

内存保护方式





## 连续分配方式

一个用户程序占用一个连续的内存空间

连续分配方式

用户作业

为一个用户程序分配  
一个连续的内存空间

### 1. 单一连续分配

最简单，只适用于单用户、单任务的系统



地址变换

静态地址重定位

内存保护

越界保护或没有



## 连续分配方式

一个用户程序占用一个连续的内存空间

### 3. 固定分区分配

#### 分配方式

将内存用户空间划分为若干个固定大小的区域，在每个分区中装入一道作业

#### 内存扩充

可采用覆盖和交换技术实现内存扩充

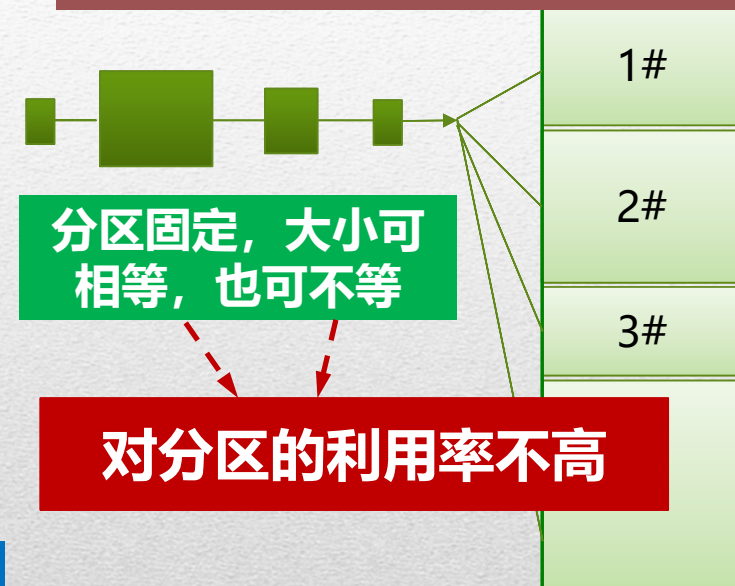
#### 地址变换

静态/动态地址重定位

#### 内存保护

利用界限寄存器实现越界保护

最简单的一种可运行多道程序的存储管理方式





## 连续分配方式

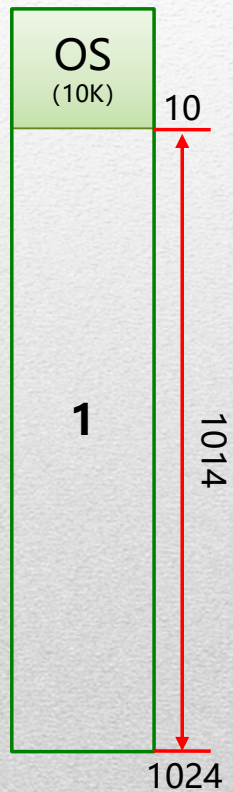
一个用户程序占用一个连续的内存空间



## 3. 可变分区分配

空闲分区表  
(动态变化)

索引号	大小/KB	起始地址/KB
1	1014	10



## 连续分配方式

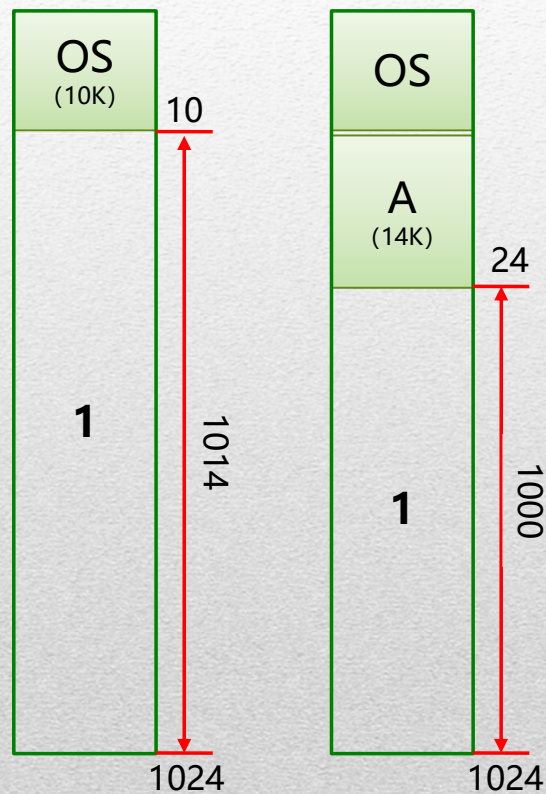
一个用户程序占用一个连续的内存空间



## 3. 可变分区分配

空闲分区表  
(动态变化)

索引号	大小/KB	起始地址/KB
1	1000	24



创建A



## 连续分配方式

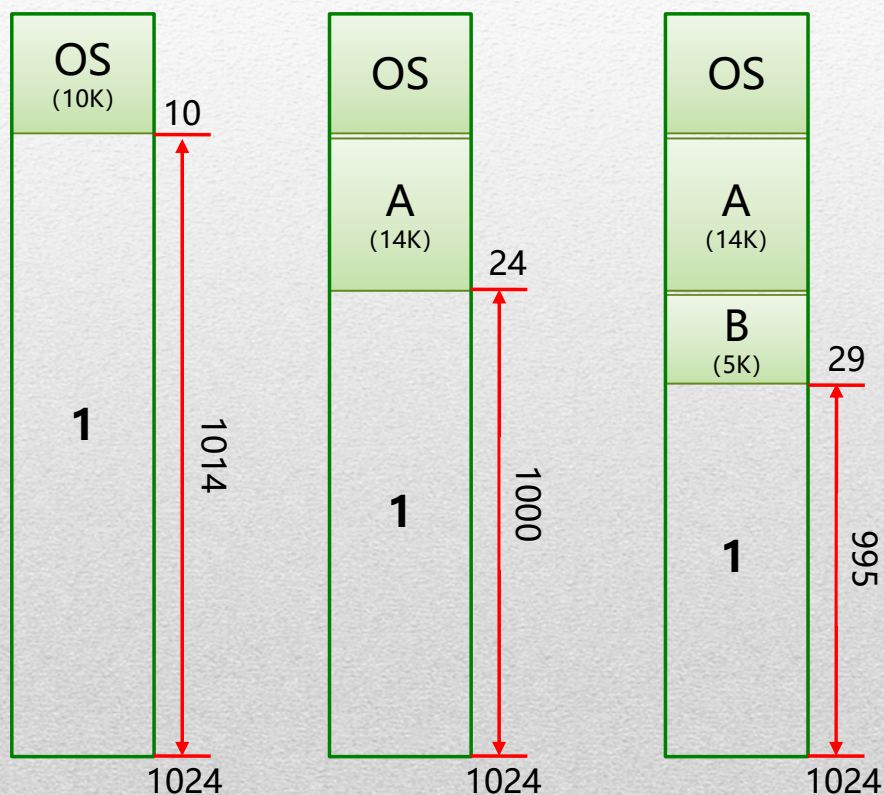
一个用户程序占用一个连续的内存空间



## 3. 可变分区分配

空闲分区表  
(动态变化)

索引号	大小/KB	起始地址/KB
1	995	29



创建A

创建B



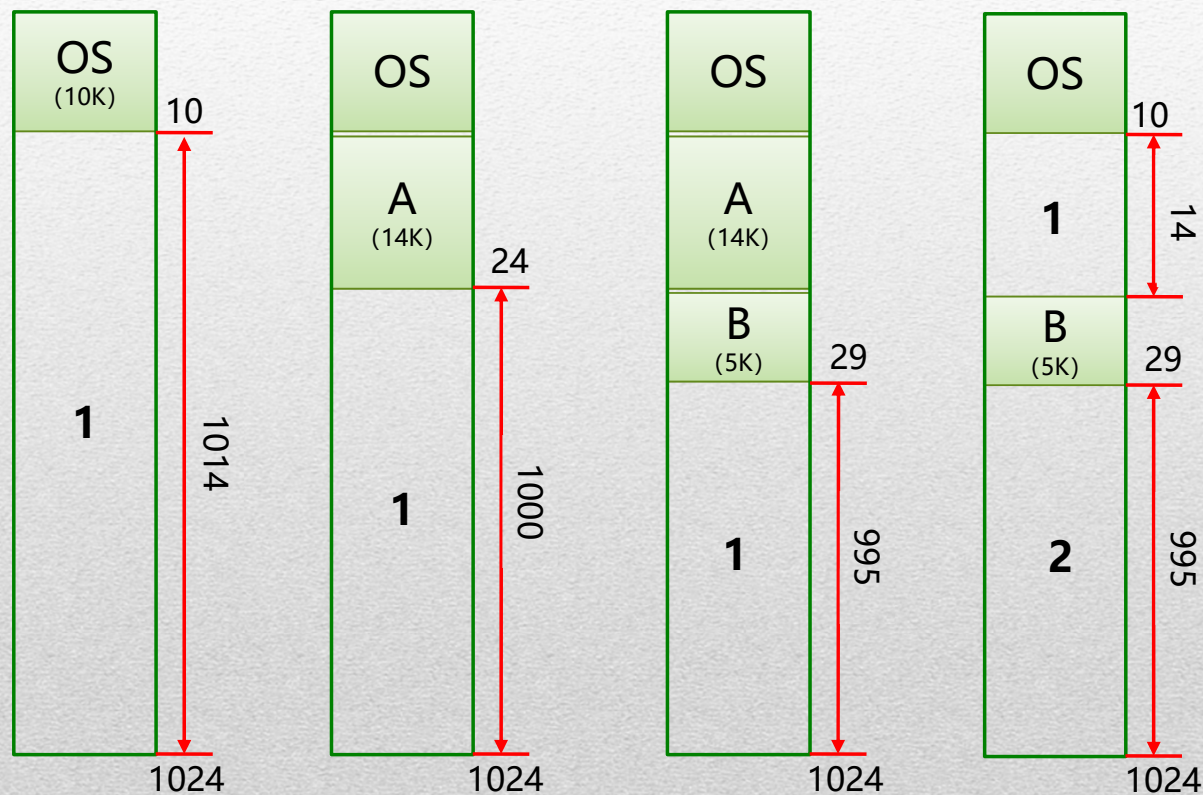
# 连续分配方式

一个用户程序占用一个连续的内存空间

## 3. 可变分区分配

空闲分区表  
(动态变化)

索引号	大小/KB	起始地址/KB
1	14	10
2	995	29







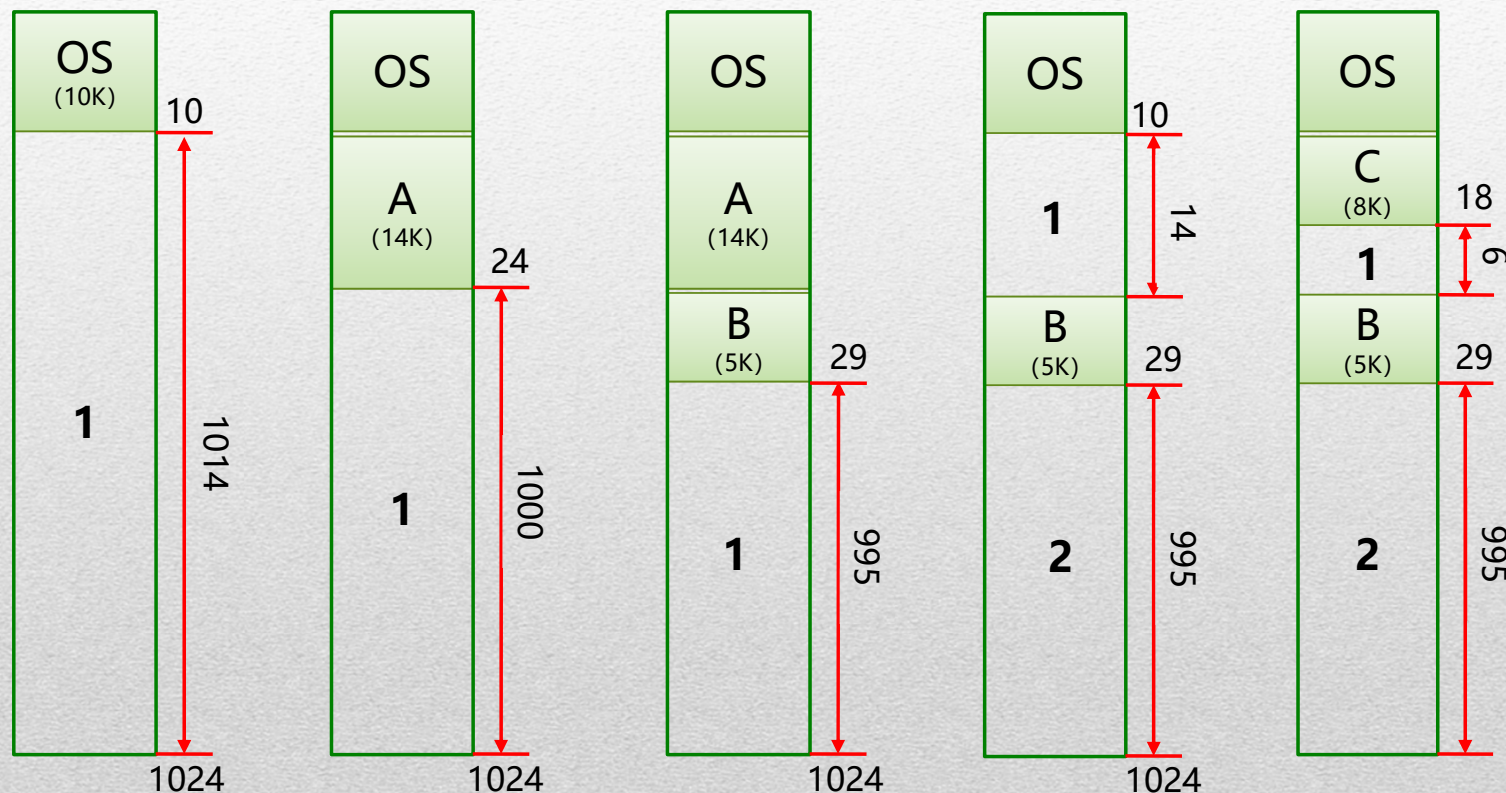
# 连续分配方式

一个用户程序占用一个连续的内存空间

## 3. 可变分区分配

空闲分区表  
(动态变化)

索引号	大小/KB	起始地址/KB
1	6	18
2	995	29





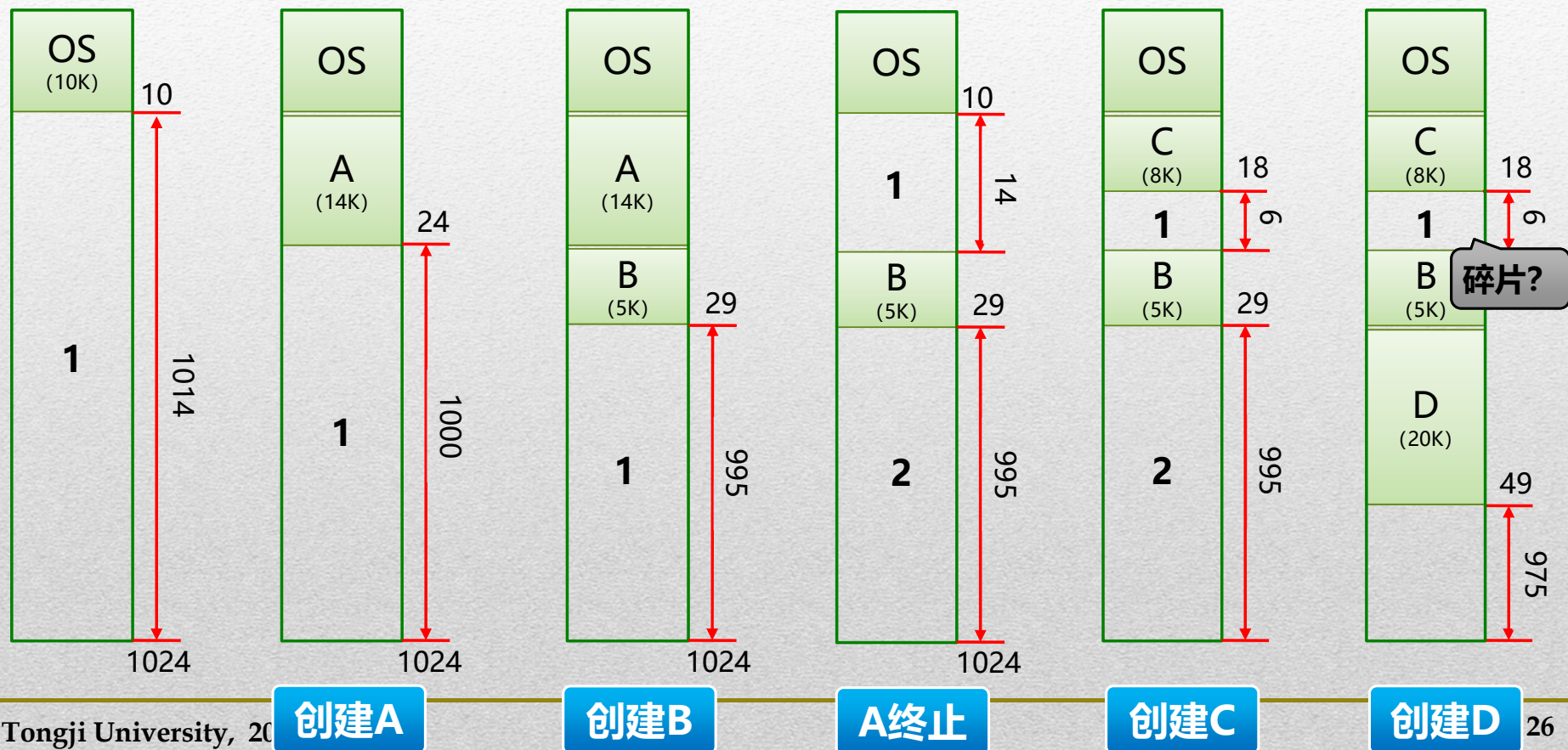
# 连续分配方式

一个用户程序占用一个连续的内存空间

## 3. 可变分区分配

空闲分区表  
(动态变化)

索引号	大小/KB	起始地址/KB
1	6	18
2	975	49







## 连续分配方式

一个用户程序占用一个连续的内存空间

### 3. 可变分区分配

#### 分配算法

##### 首次适应

- 从第1个空闲区开始顺序找, 直至找到一个大小满足要求的空闲分区
- 从该分区中划一块给请求者, 余下的仍留空闲分区表中

##### 循环首次适应

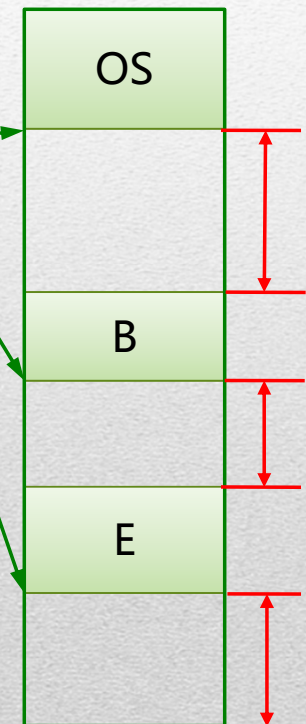
- 从上次找到空闲分区的下一个空闲分区开始查找 (采用循环查找方式)
- 从该分区中划一块给请求者, 余下的仍留空闲分区表中

按空闲分区首地址  
升序排列

索引号	大小	起始地址
1	150	1000
2	70	2000
3	100	3000

矛盾和碎片集中在低地址部分

缺乏大的空闲分区





## 连续分配方式

一个用户程序占用一个连续的内存空间

### 3. 可变分区分配

例：系统中的空闲分区表如下，现有三个作业依次申请内存空间100K、30K及7K。给出分别按**首次适应算法**和**循环首次适应算法**的内存分配情况及分配后空闲分区表。

索引号	大小	起始地址
1	32	20
2	8	52
3	120	60
4	331	180

首次适应算法

索引号	大小	起始地址
1	2	50
2	1	59
3	20	160
4	331	180

循环首次适应算法

索引号	大小	起始地址
1	25	27
2	8	52
3	20	160
4	301	210





## 连续分配方式

一个用户程序占用一个连续的内存空间

### 3. 可变分区分配

#### 分配算法

##### 最佳匹配法

- 总把能满足要求，又最小的空闲区分配给进程

避免大材小用，但碎片小

##### 最坏匹配法

- 总把能满足要求，又最大的空闲区分配给进程，产生碎片的几率最小。

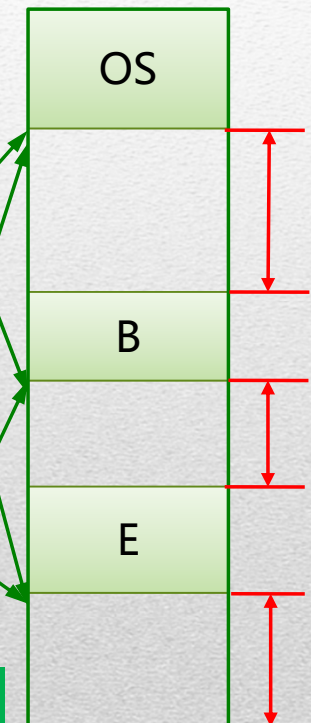
查找效率高，但缺大分区

按空闲分区大小升序排列

索引号	大小	起始地址
1	70	2000
2	100	3000
3	150	1000

索引号	大小	起始地址
1	150	1000
2	100	3000
3	70	2000

按空闲分区大小降序排列





## 连续分配方式

一个用户程序占用一个连续的内存空间

### 3. 可变分区分配

例：系统中的空闲分区表如下，现有三个作业依次申请内存空间100K、30K及7K。给出分别按**最佳匹配算法**和**最坏匹配算法**的内存分配情况及分配后空闲分区表。

索引号	大小	起始地址	最佳匹配算法	索引号	大小	起始地址
1	8	52		1	1	59
2	32	20		2	2	50
3	120	60		3	20	160
4	331	180		4	331	180

索引号	大小	起始地址	最坏匹配算法	索引号	大小	起始地址
1	331	180		1	194	317
2	120	60		2	120	60
3	32	20		3	32	20
4	8	52		4	8	52



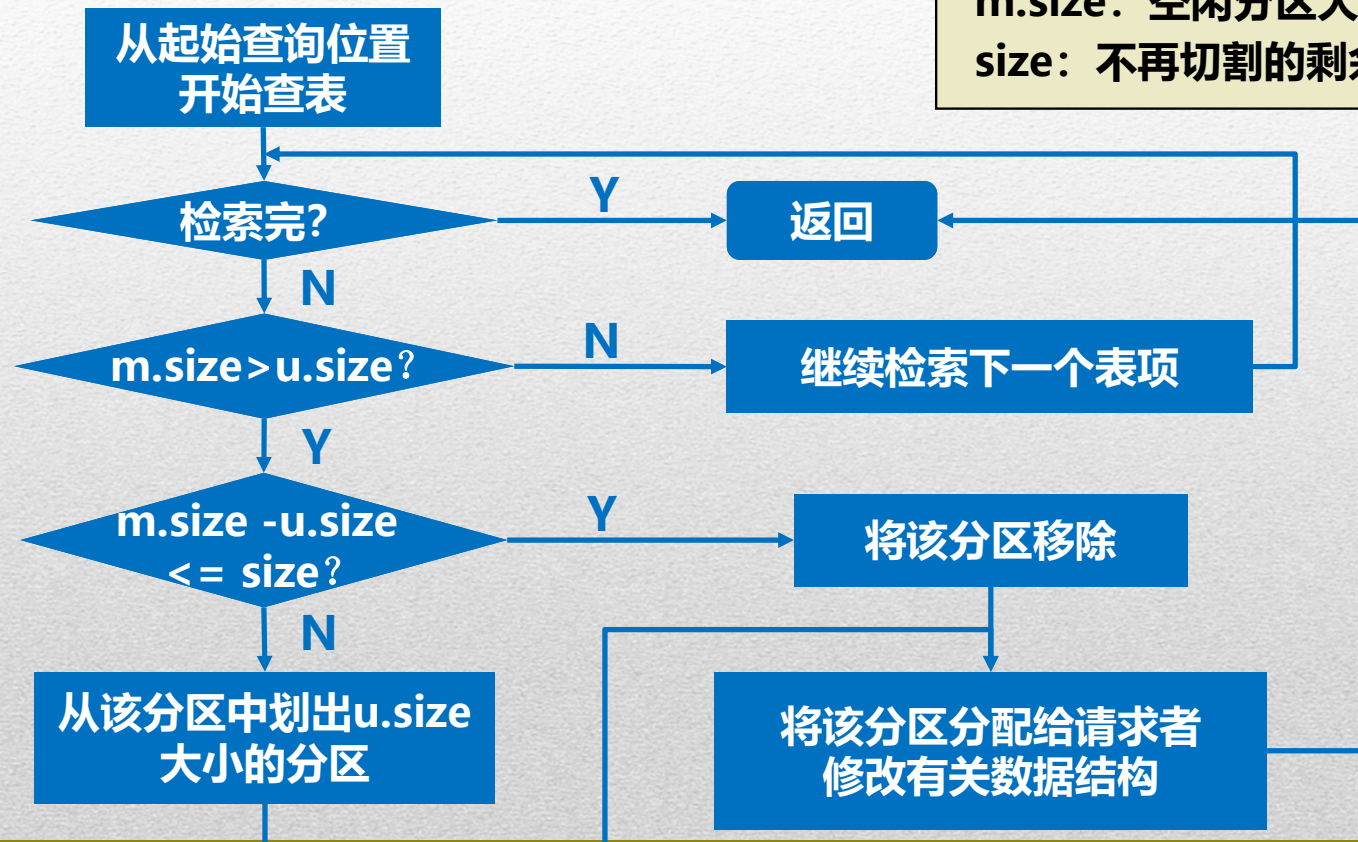


## 连续分配方式

一个用户程序占用一个连续的内存空间

## 3. 可变分区分配

分配操作



u.size: 请求分区大小

m.size: 空闲分区大小

size: 不再切割的剩余分区大小



# 连续分配方式

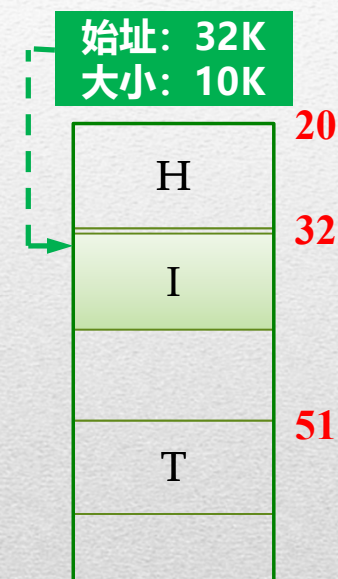
一个用户程序占用一个连续的内存空间

## 3. 可变分区分配

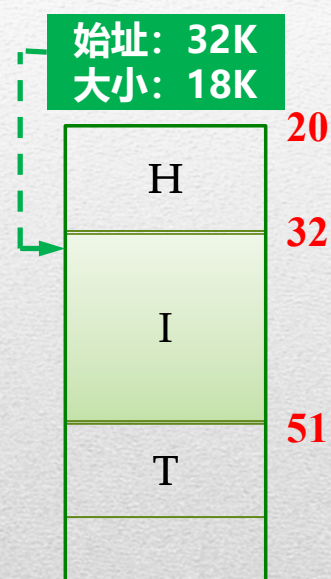
需要分四种情况考虑

索引号	大小	起始地址
H	12	20
T	20	51

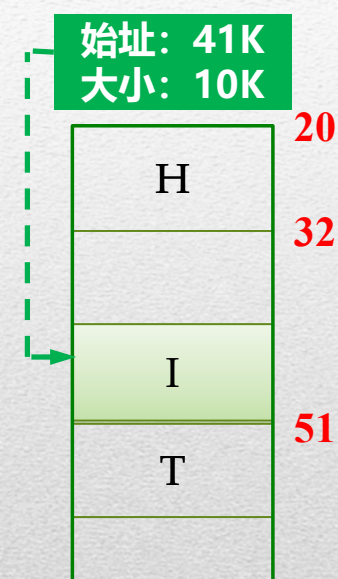
内存回收



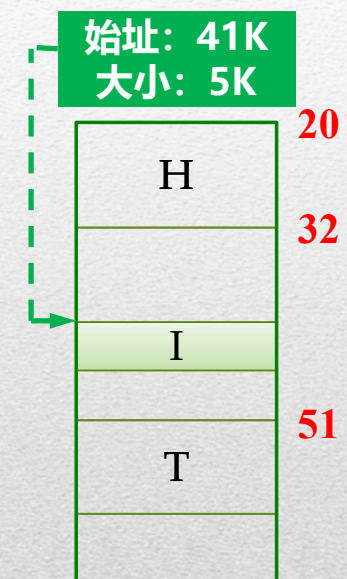
	大小	始址
H	22	20
T	20	51



	大小	始址
H	50	20
<del>T</del>	<del>20</del>	<del>51</del>



	大小	始址
H	12	20
T	30	41



	大小	始址
H	12	20
I	5	41
T	20	51

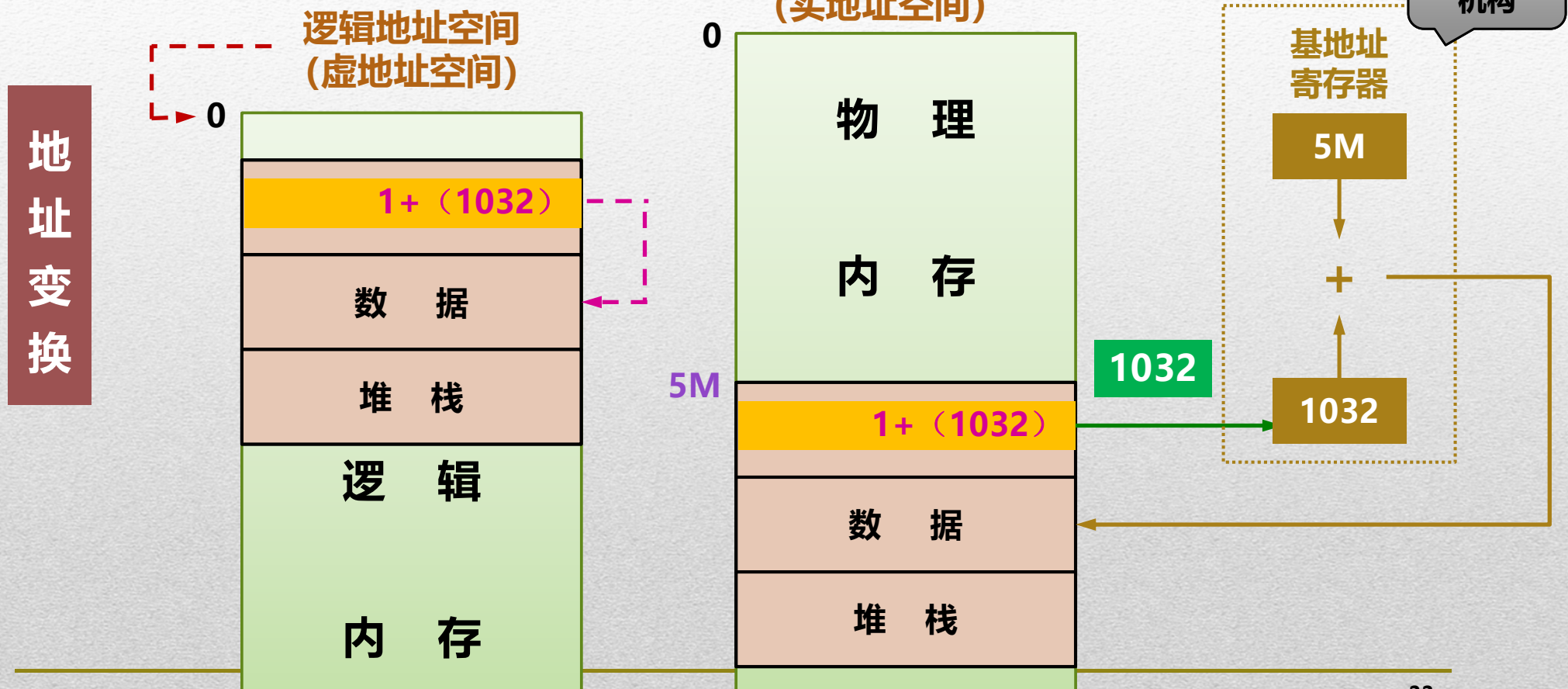




## 连续分配方式

一个用户程序占用一个连续的内存空间

## 3. 可变分区分配





## 连续分配方式

一个用户程序占用一个连续的内存空间

### 3. 可变分区分配

其他

#### 内存扩充

可采用覆盖和交换技术  
实现内存扩充

#### 内存保护

利用界限寄存器实现越  
界保护

对分配过程中造成的碎  
片如何处理???

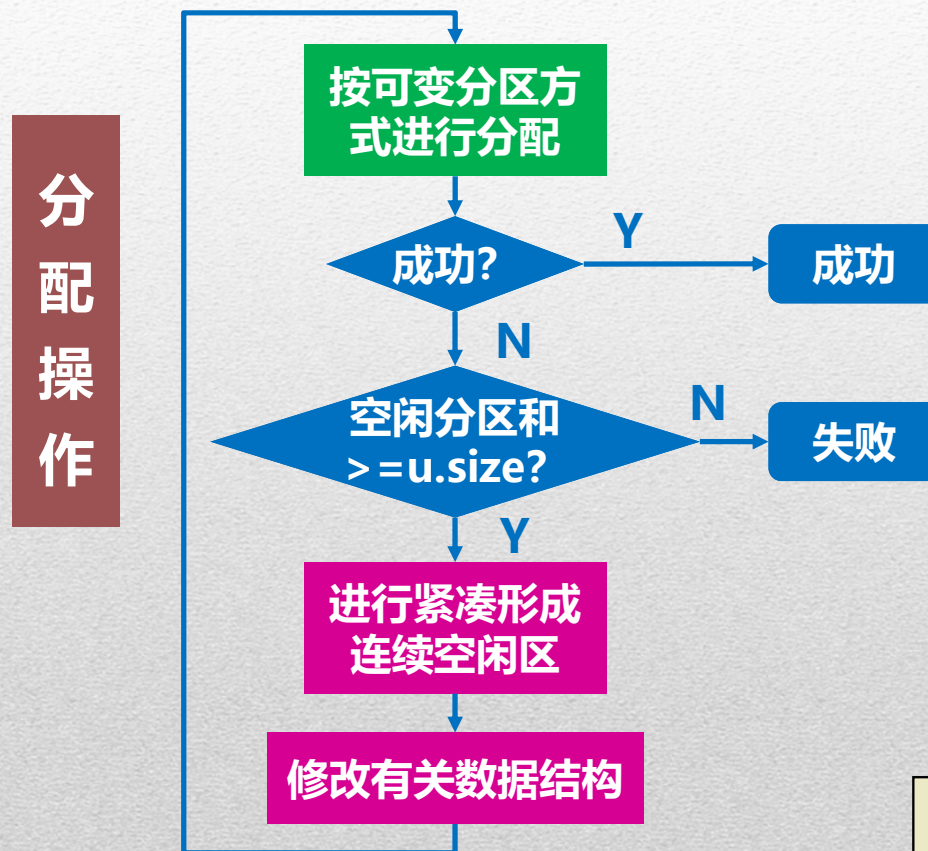




## 连续分配方式

一个用户程序占用一个连续的内存空间

### 4. 可重定位分区分配



紧凑后的用户程序在内存中的位置发生了变化，必须动态地址重定位！！



## 连续分配方式

一个用户程序占用一个连续的内存空间

	单一	固定	可变/可重定位
适用环境	单道	多道	
重定位	静态	静态/动态	
分配方式	静态分配 连续区		动态分配 连续区
释放	执行完后全部释放		
保护	没有	越界保护	
内存扩充	没有	交换与覆盖	

如果采用交换、覆盖  
或紧凑，则必须使用  
动态重定位

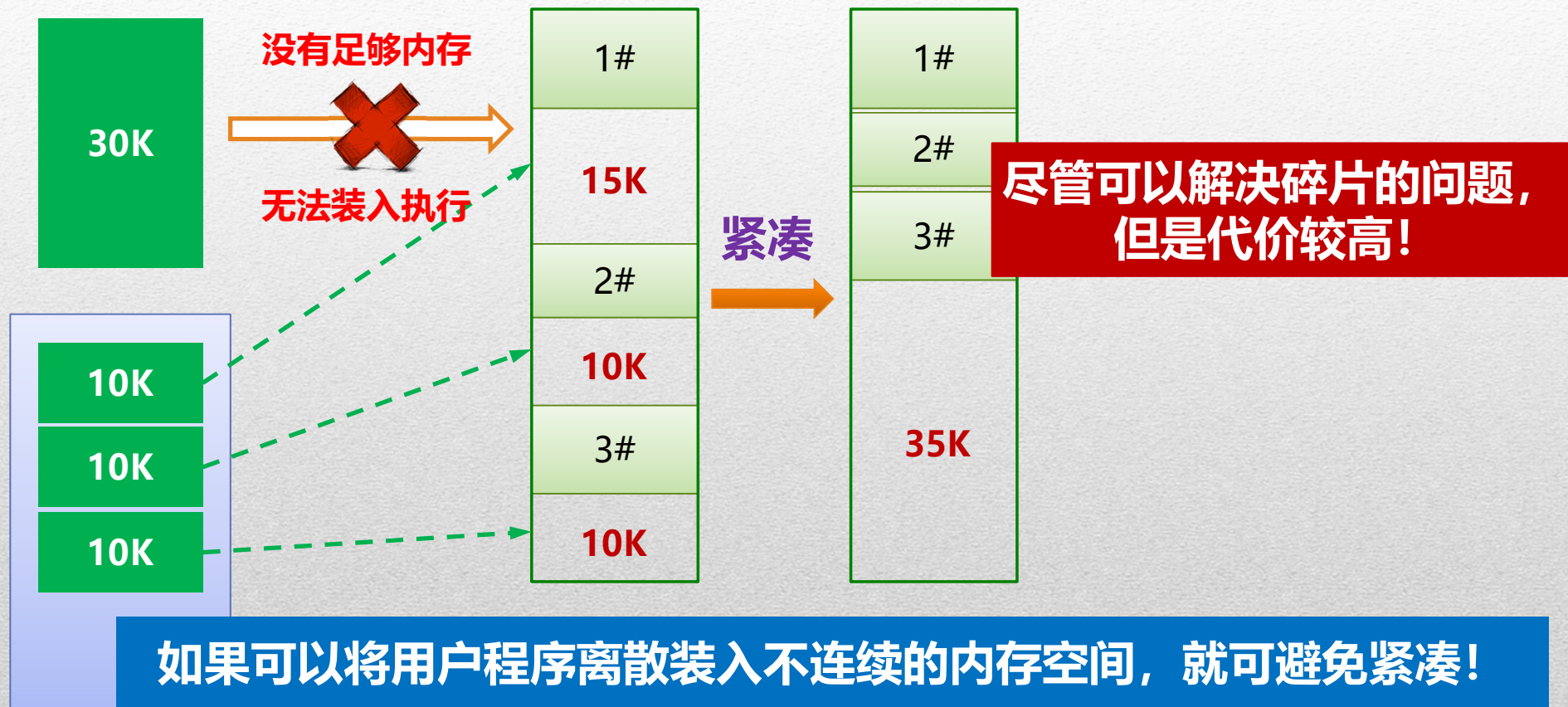




## 连续分配方式

一个用户程序占用一个连续的内存空间

实现简单，可用于单道或多道程序，但.....





## 本节小结:

- 1 两种内存分配方式: 连续分配方式和离散分配方式
- 2 三种内存空间管理方式: 索引表、链式队列与位示图
- 3 逻辑地址与物理地址的概念、静态与动态地址重定位
- 4 内存扩充: 交换与覆盖
- 5 连续分配方式的具体实现