



# 作业管理之进程调度

## 进程的调度

- ◆ 进程调度概述
- ◆ 进程调度算法

# 作业管理之进程调度

## 进程的调度

进程调度是指计算机通过决策决定哪个就绪进程可以获得CPU使用权

### 多道程序设计

- ◆ 保留旧进程的运行信息，请出旧进程（收拾包袱）
- ◆ 选择新进程，准备运行环境并分配CPU（新进驻）

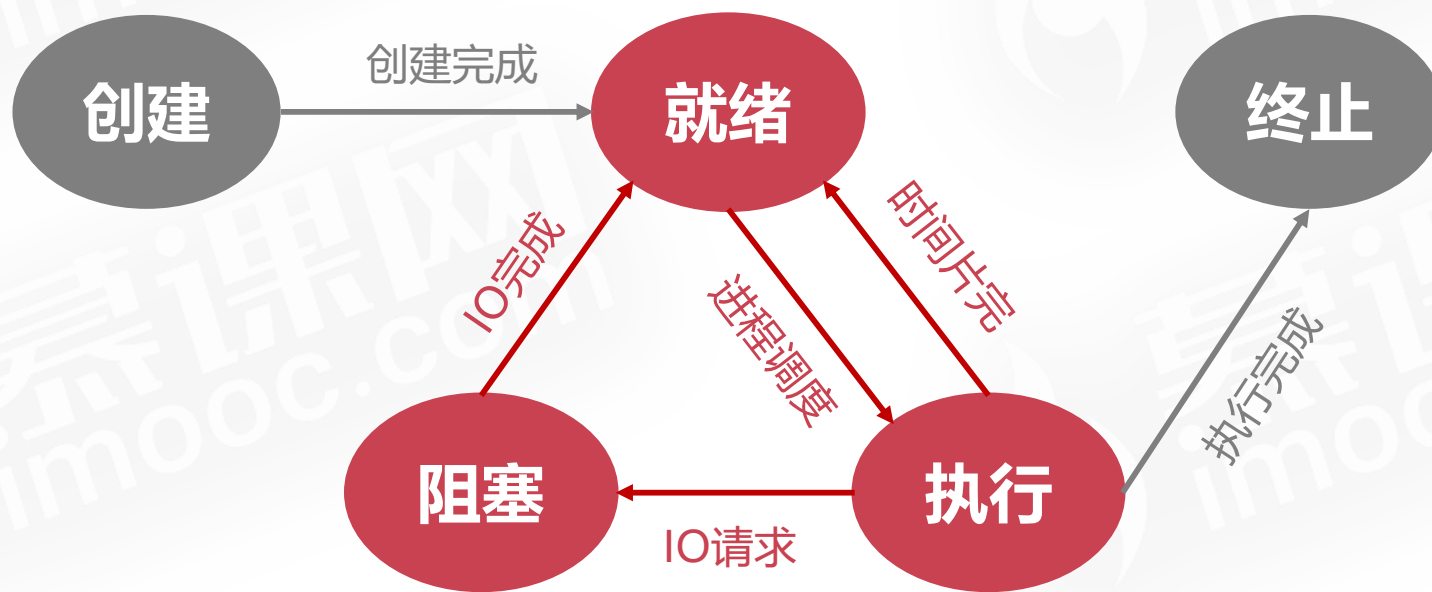
# 作业管理之进程调度

## 进程的调度

- ◆ 就绪队列的排队机制
- ◆ 选择运行进程的委派机制
- ◆ 新老进程的上下文切换机制

# 作业管理之进程调度

## 进程的调度



进程的五状态模型

# 作业管理之进程调度

## 进程的调度



将就绪进程按照一定的方式排成队列，以便调度程序可以最快找到就绪进程

就绪队列的排队机制

# 作业管理之进程调度

## 进程的调度



调度程序以一定的策略选择就绪进程，将CPU资源分配给它

选择运行进程的委派机制

# 作业管理之进程调度

## 进程的调度



保存当前进程的上下文信息，装入被委派执行进程的运行上下文

新老进程的上下文切换机制



# 作业管理之进程调度

## 进程的调度

- ◆ 非抢占式的调度
- ◆ 抢占式的调度

老进程还没执行完呢？



# 作业管理之进程调度

## 进程的调度

- ◆ 处理器一旦分配给某个进程，就让该进程一直使用下去
- ◆ 调度程序不以任何原因抢占正在被使用的处理器
- ◆ 直到进程完成工作或因为IO阻塞才会让出处理器

非抢占式的调度

# 作业管理之进程调度

## 进程的调度

- ◆ 允许调度程序以一定的策略暂停当前运行的进程
- ◆ 保存好旧进程的上下文信息，分配处理器给新进程

抢占式的调度

# 作业管理之进程调度

## 进程的调度

	抢占式调度	非抢占式调度
系统开销	频繁切换，开销大	切换次数少，开销小
公平性	相对公平	不公平
应用	通用系统	专用系统

# 作业管理之进程调度

## 进程的调度

- ◆ 进程调度概述
- ◆ 进程调度算法

# 作业管理之进程调度

## 进程的调度

- ◆ 先来先服务调度算法
- ◆ 短进程优先调度算法
- ◆ 高优先权优先调度算法
- ◆ 时间片轮转调度算法

# 作业管理之进程调度

## 进程的调度



先来先服务调度算法

# 作业管理之进程调度

## 进程的调度

- ◆ 调度程序优先选择就绪队列中估计运行时间最短的进程
- ◆ 短进程优先调度算法不利于长作业进程的执行

短进程优先调度算法



# 作业管理之进程调度

## 进程的调度

- ◆ 进程附带优先权，调度程序优先选择权重高的进程
- ◆ 高优先权优先调度算法使得紧迫的任务可以优先处理

前台进程/  
后台进程

高优先权优先调度算法



@咚咚呛

# 作业管理之进程调度

## 进程的调度

- ◆ 按先来先服务的原则排列就绪进程
- ◆ 每次从队列头部取出待执行进程，分配一个时间片执行
- ◆ 是相对公平的调度算法，但不能保证及时响应用户

时间片轮转调度算法

# 作业管理之进程调度

## 进程的调度

- ◆ 先来先服务调度算法
- ◆ 短进程优先调度算法
- ◆ 高优先权优先调度算法
- ◆ 时间片轮转调度算法

# 作业管理之进程调度

## 进程的调度

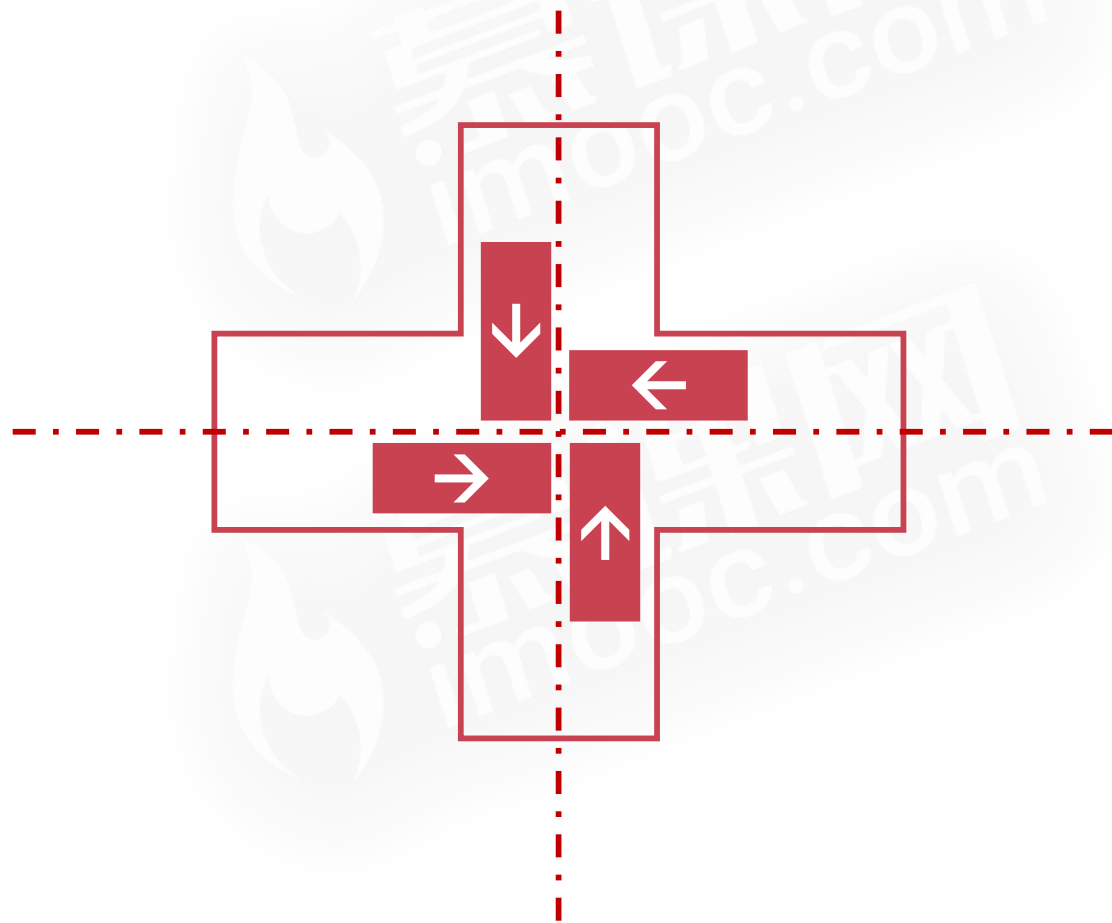
- ◆ 进程调度概述
- ◆ 进程调度算法



# 作业管理之死锁

## 死锁

死锁是指两个或两个以上的进程在执行过程中，由于竞争资源或者由于彼此通信而造成的一种阻塞的现象，若无外力作用，它们都将无法推进下去。此时称系统处于死锁状态或系统产生了死锁，这些永远在互相等待的进程称为死锁进程。



# 作业管理之死锁

## 死锁

- ◆ 死锁的产生
- ◆ 死锁的处理

# 作业管理之死锁

## 死锁

- ◆ 竞争资源
- ◆ 进程调度顺序不当

死锁的产生



# 作业管理之死锁

## 死锁

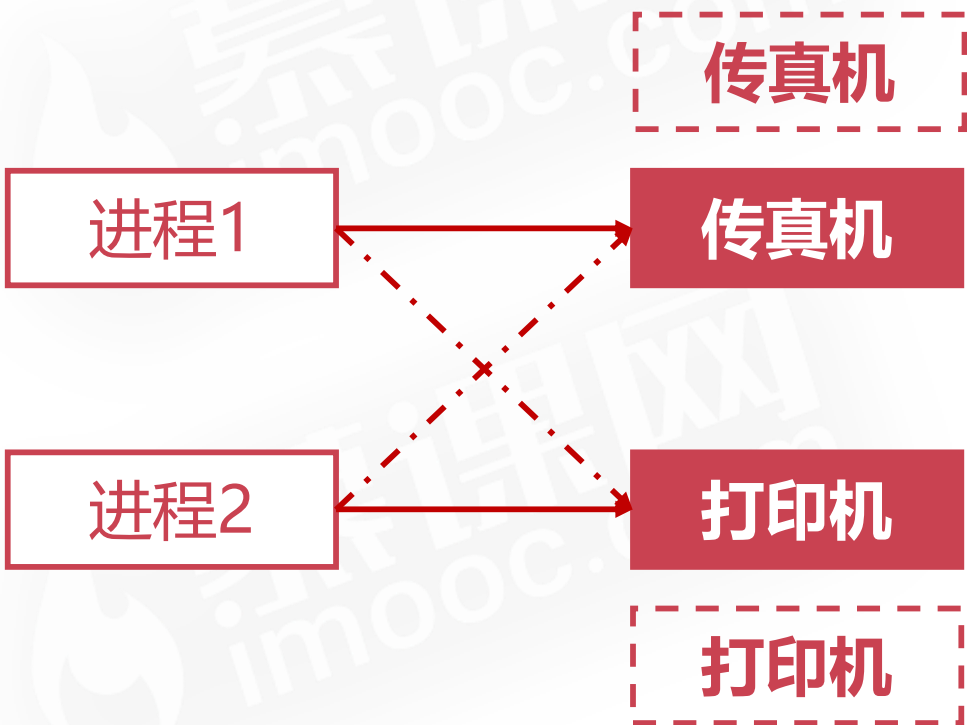
- ◆ 共享资源数量不满足各个进程需求
- ◆ 各个进程之间发生资源进程导致死锁

死锁的产生

竞争资源

# 作业管理之死锁

## 死锁



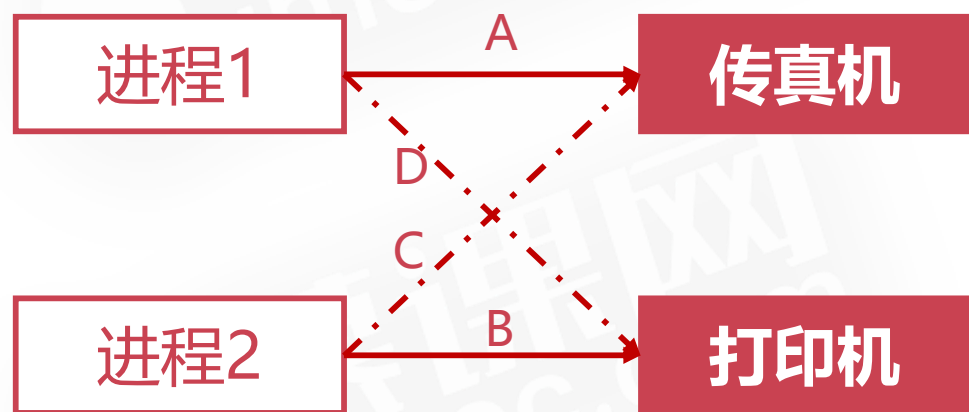
- ◆ 等待请求的资源被释放
- ◆ 自身占用资源不释放

死锁的产生

竞争资源

# 作业管理之死锁

## 死锁



$A \rightarrow B \rightarrow C \rightarrow D$

$A \rightarrow D \rightarrow B \rightarrow C$

死锁的产生

进程调度顺序不当

# 作业管理之死锁

## 死锁

- ◆ 互斥条件
- ◆ 请求保持条件
- ◆ 不可剥夺条件
- ◆ 环路等待条件

死锁的四个必要条件

# 作业管理之死锁

## 死锁

- ◆ 进程对资源的使用是排他性的使用
- ◆ 某资源只能由一个进程使用，其他进程需要使用只能等待

互斥条件

# 作业管理之死锁

## 死锁

- ◆ 进程至少保持一个资源，又提出新的资源请求
- ◆ 新资源被占用，请求被阻塞
- ◆ 被阻塞的进程不释放自己保持的资源

请求保持条件

# 作业管理之死锁

## 死锁

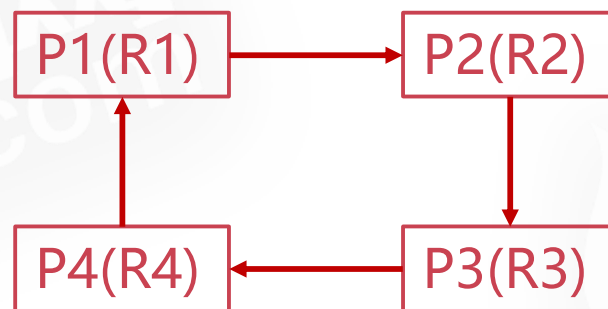
- ◆ 进程获得的资源在未完成使用前不能被剥夺
- ◆ 获得的资源只能由进程自身释放

不可剥夺条件

# 作业管理之死锁

## 死锁

- ◆ 发生死锁时，必然存在进程-资源环形链



环路等待条件



# 作业管理之死锁

## 死锁

- ◆ 死锁的产生
- ◆ 死锁的处理

# 作业管理之死锁

## 死锁

- ◆ 预防死锁的方法
- ◆ 银行家算法

# 作业管理之死锁

## 死锁

◆ 互斥条件

◆ 请求保持~~条件~~

◆ 不可~~剥夺~~条件

◆ 环路等待~~条件~~

预防死锁的方法

# 作业管理之死锁

## 死锁

- ◆ 系统规定进程运行之前，一次性申请所有需要的资源
- ◆ 进程在运行期间不会提出资源请求，从而摒弃请求保持条件

预防死锁的方法

摒弃请求保持条件

# 作业管理之死锁

## 死锁

- ◆ 当一个进程请求新的资源得不到满足时，必须释放占有的资源
- ◆ 进程运行时占有的资源可以被释放，意味着可以被剥夺

预防死锁的方法

摒弃不可剥夺条件

# 作业管理之死锁

## 死锁

- ◆ 可用资源线性排序，申请必须按照需要递增申请
- ◆ 线性申请不再形成环路，从而摒弃了环路等待条件

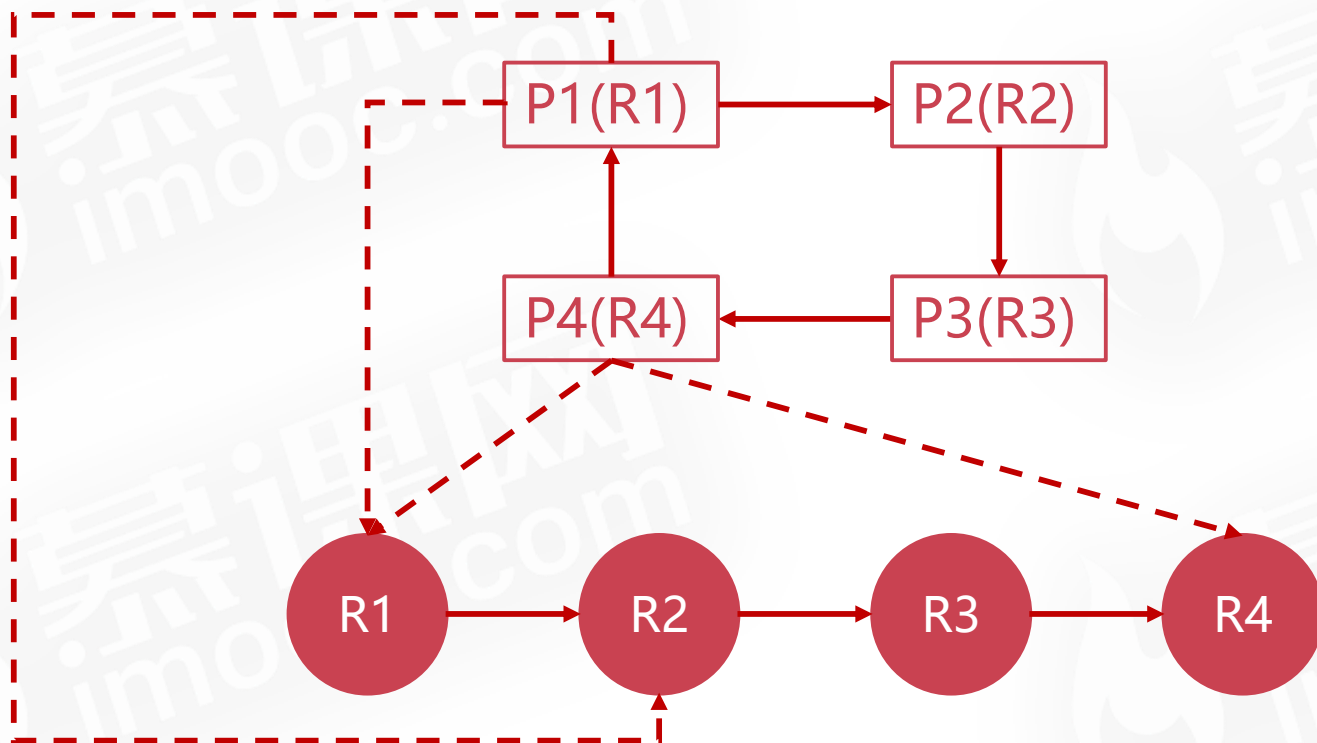


预防死锁的方法

摒弃环路等待条件

# 作业管理之死锁

死锁



预防死锁的方法

摒弃环路等待条件

# 作业管理之死锁

## 死锁

- ◆ 预防死锁的方法
- ◆ 银行家算法



# 作业管理之死锁

## 死锁

- ◆ 是一个可操作的著名的避免死锁的算法
- ◆ 以银行借贷系统分配策略为基础的算法

银行家算法

# 作业管理之死锁

## 死锁

- ◆ 客户申请的贷款是有限的，每次申请需声明最大资金量
- ◆ 银行家在能够满足贷款时，都应该给用户贷款
- ◆ 客户在使用贷款后，能够及时归还贷款

银行家算法

# 作业管理之死锁

## 死锁

	A	B	C	D
P1	0	0	1	4
P2	1	4	3	2
P3	1	3	5	4
P4	1	0	0	0

已分配资源表

	A	B	C	D
P1	0	6	5	6
P2	1	9	4	2
P3	1	3	5	6
P4	1	7	5	0

所需资源表

A	B	C	D
1	5	2	0

可分配资源表

# 作业管理之死锁

	A	B	C	D
P1	0	6	5	6
P2	1	9	4	2
P3	1	3	5	6
P4	1	7	5	0

所需资源表

A	B	C	D
1	5	2	0

可分配资源表

	A	B	C	D
P1	0	0	1	4
P2	1	4	3	2
P3	1	3	5	4
P4	1	0	0	0

已分配资源表

	A	B	C	D
P1	0	6	4	2
P2	0	5	1	0
P3	0	0	0	2
P4	0	7	5	0

还需分配资源表

# 作业管理之死锁

	A	B	C	D
P1	0	6	5	6
P2	1	9	4	2
P3	1	3	5	6
P4	1	7	5	0

所需资源表

A	B	C	D
1	5	2	0

可分配资源表

	A	B	C	D
P1	0	6	4	2
P2	0	5	1	0
P3	0	0	0	2
P4	0	7	5	0

还需分配资源表

	A	B	C	D
P1	0	0	1	4
P2	1	4	3	2
P3	1	3	5	4
P4	1	0	0	0

已分配资源表

# 作业管理之死锁

	A	B	C	D
P1	0	6	5	6
P2	1	9	4	2
P3	1	3	5	6
P4	1	7	5	0

所需资源表

A	B	C	D
1	5	2	0

可分配资源表

	A	B	C	D
P1	0	6	4	2
P2	0	5	1	0
P3	0	0	0	2
P4	0	7	5	0

还需分配资源表

	A	B	C	D
P1	0	0	1	4
P2	1	4	3	2
P3	1	3	5	4
P4	1	0	0	0

已分配资源表

# 作业管理之死锁

	A	B	C	D
P1	0	6	5	6
P2	1	9	4	2
P3	1	3	5	6
P4	1	7	5	0

所需资源表

A	B	C	D
1	5	2	0

可分配资源表

	A	B	C	D
P1	0	6	4	2
P2	0	5	1	0
P3	0	0	0	2
P4	0	7	5	0

还需分配资源表

	A	B	C	D
P1	0	0	1	4
P2	1	4	3	2
P3	1	3	5	4
P4	1	0	0	0

已分配资源表

# 作业管理之死锁

	A	B	C	D
P1	0	6	5	6
P2	1	9	4	2
P3	1	3	5	6
P4	1	7	5	0

所需资源表

A	B	C	D
1	5	2	0

可分配资源表

	A	B	C	D
P1	0	6	4	2
P2	0	5	1	0
P3	0	0	0	2
P4	0	7	5	0

还需分配资源表

	A	B	C	D
P1	0	0	1	4
P2	1	4	3	2
P3	1	3	5	4
P4	1	0	0	0

已分配资源表



# 作业管理之死锁

## 死锁

- ◆ 死锁的产生
- ◆ 死锁的处理



# 存储管理之内存分配与回收

早期计算机编程并不需要过多的存储管理

随着计算机和程序越来越复杂，存储管理成为必要

- ◆ 确保计算机有足够的内存处理数据
- ◆ 确保程序可以从可用内存中获取一部分内存使用
- ◆ 确保程序可以归还使用后的内存以供其他程序使用

# 存储管理之内存分配与回收

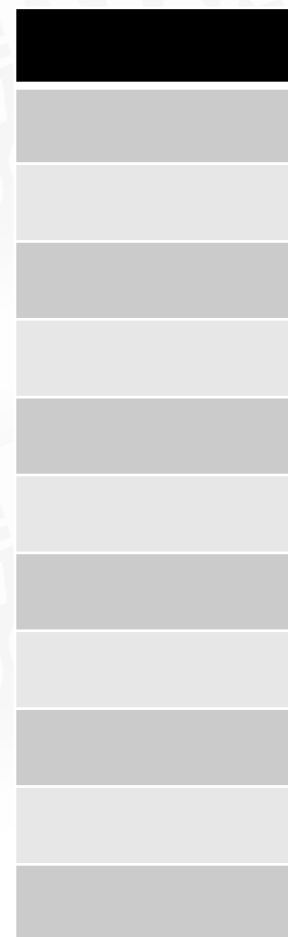
- ◆ 内存分配的过程
- ◆ 内存回收的过程

# 存储管理之内存分配与回收

## 内存分配的过程

- ◆ 单一连续分配是最简单的内存分配方式
- ◆ 只能在单用户、单进程的操作系统中使用

单一连续分配



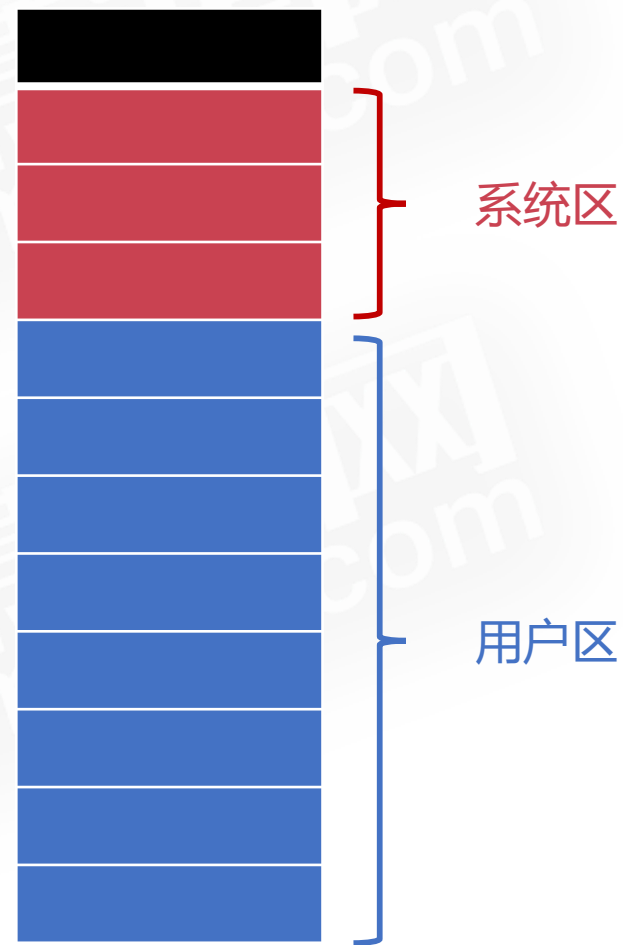
主存

# 存储管理之内存分配与回收

## 内存分配的过程

- ◆ 单一连续分配是最简单的内存分配方式
- ◆ 只能在单用户、单进程的操作系统中使用

单一连续分配

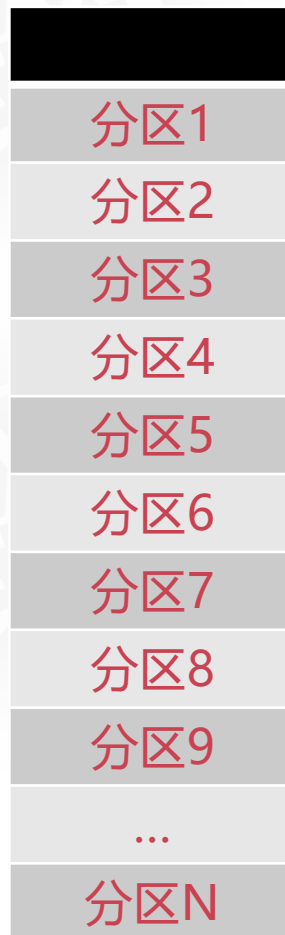


# 存储管理之内存分配与回收

## 内存分配的过程

- ◆ 固定分区分配是支持多道程序的最简单存储分配方式
- ◆ 内存空间被划分为若干固定大小的区域
- ◆ 每个分区只提供给一个程序使用，互不干扰

固定分区分配



# 存储管理之内存分配与回收

## 内存分配的过程

- ◆ 根据进程实际需要，动态分配内存空间
- ◆ 相关数据结构、分配算法

动态分区分配



# 存储管理之内存分配与回收

## 内存分配的过程

分区	1	2	3	4	5	6	7	8	9	10	11
标记	0	1	0	0	1	1	0	0	1	1	0

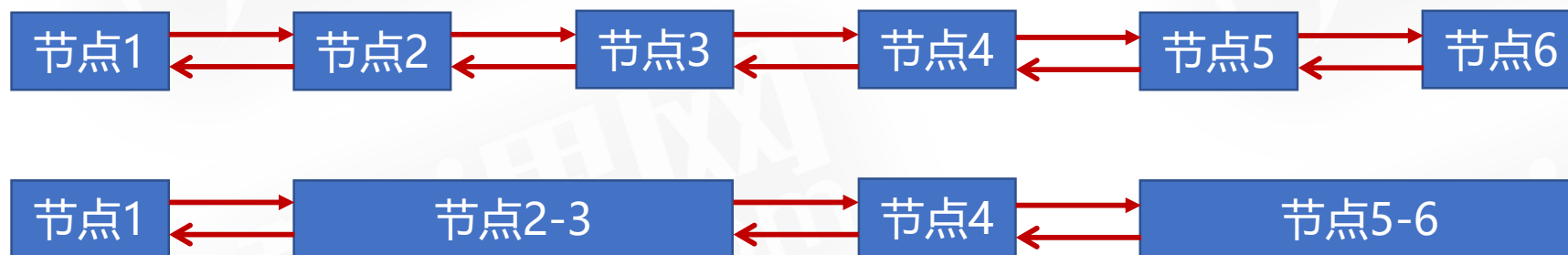


动态分区分配

动态分区空闲表数据结构

# 存储管理之内存分配与回收

## 内存分配的过程



◆ 节点需记录可存储的容量

动态分区分配

动态分区空闲链数据结构



# 存储管理之内存分配与回收

## 内存分配的过程

- ◆ 首次适应算法(FF算法)
- ◆ 最佳适应算法(BF算法)
- ◆ 快速适应算法(QF算法)

动态分区分配

动态分区分配算法

# 存储管理之内存分配与回收

## 内存分配的过程

- ◆ 分配内存时从开始顺序查找适合内存区
- ◆ 若没有合适的空闲区，则该次分配失败



动态分区分配

动态分区分配算法

首次适应算法

# 存储管理之内存分配与回收

## 内存分配的过程

- ◆ 分配内存时从开始顺序查找适合内存区
- ◆ 若没有合适的空闲区，则该次分配失败



动态分区分配

动态分区分配算法

首次适应算法

# 存储管理之内存分配与回收

## 内存分配的过程

- ◆ 分配内存时从开始顺序查找适合内存区
- ◆ 若没有合适的空闲区，则该次分配失败
- ◆ 每次从头部开始，使得头部地址空间不断被划分



动态分区分配

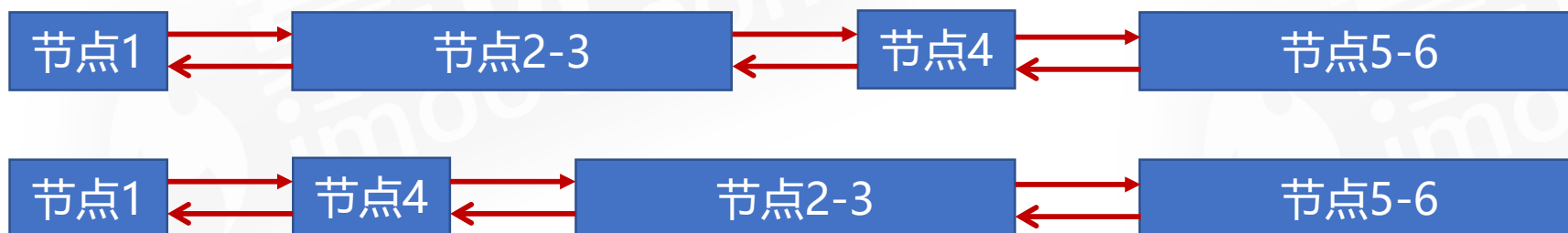
动态分区分配算法

首次适应算法

# 存储管理之内存分配与回收

## 内存分配的过程

- ◆ 最佳适应算法要求空闲区链表按照容量大小排序
- ◆ 遍历空闲区链表找到最佳合适空闲区



动态分区分配

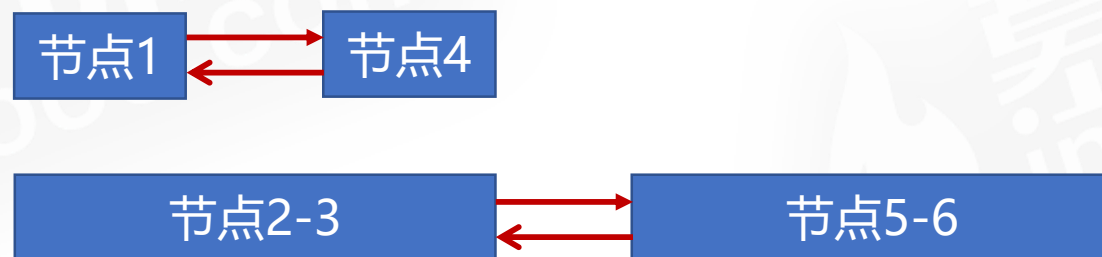
动态分区分配算法

最佳适应算法

# 存储管理之内存分配与回收

## 内存分配的过程

- ◆ 快速适应算法要求有多个空闲区链表
- ◆ 每个空闲区链表存储一种容量的空闲区



动态分区分配

动态分区分配算法

快速适应算法



# 存储管理之内存分配与回收

- ◆ 内存分配的过程
- ◆ 内存回收的过程

# 存储管理之内存分配与回收

## 内存回收的过程



# 存储管理之内存分配与回收

## 内存回收的过程

- ◆ 不需要新建空闲链表节点
- ◆ 只需要把空闲区1的容量增大为空闲区即可



# 存储管理之内存分配与回收

## 内存回收的过程

- ◆ 将回收区与空闲区合并
- ◆ 新的空闲区使用回收区的地址



# 存储管理之内存分配与回收

## 内存回收的过程

- ◆ 将空闲区1、空闲区2和回收区合并
- ◆ 新的空闲区使用空闲区1的地址



# 存储管理之内存分配与回收

## 内存回收的过程

- ◆ 为回收区创建新的空闲节点
- ◆ 插入到相应的空闲区链表中去



# 存储管理之内存分配与回收

- ◆ 内存分配的过程
- ◆ 内存回收的过程

慕课网  
imooc.com

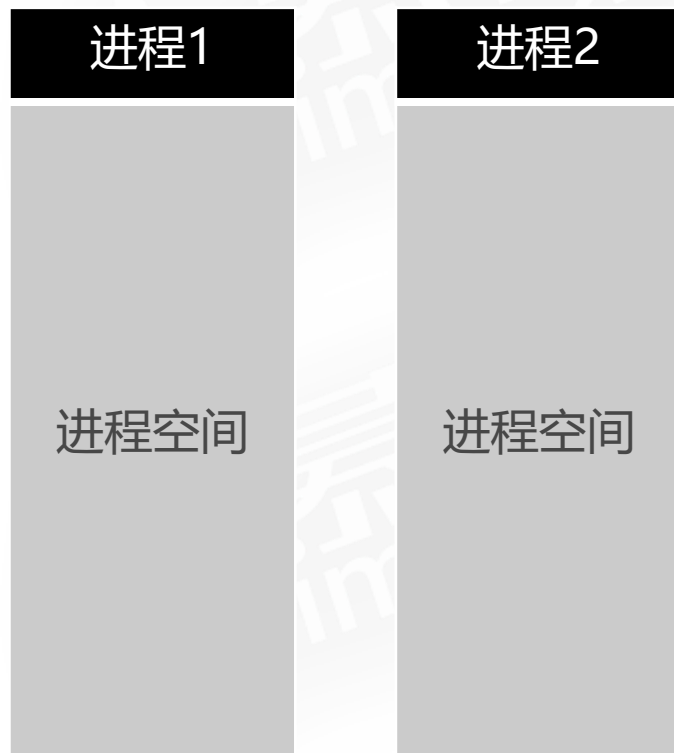
慕课网  
imooc.com

慕课网  
imooc.com

慕课网  
imooc.com



# 存储管理之段页式存储管理



操作系统是如何  
管理进程的空间  
的呢?



# 存储管理之段页式存储管理

- ◆ 页式存储管理
- ◆ 段式存储管理
- ◆ 段页式存储管理

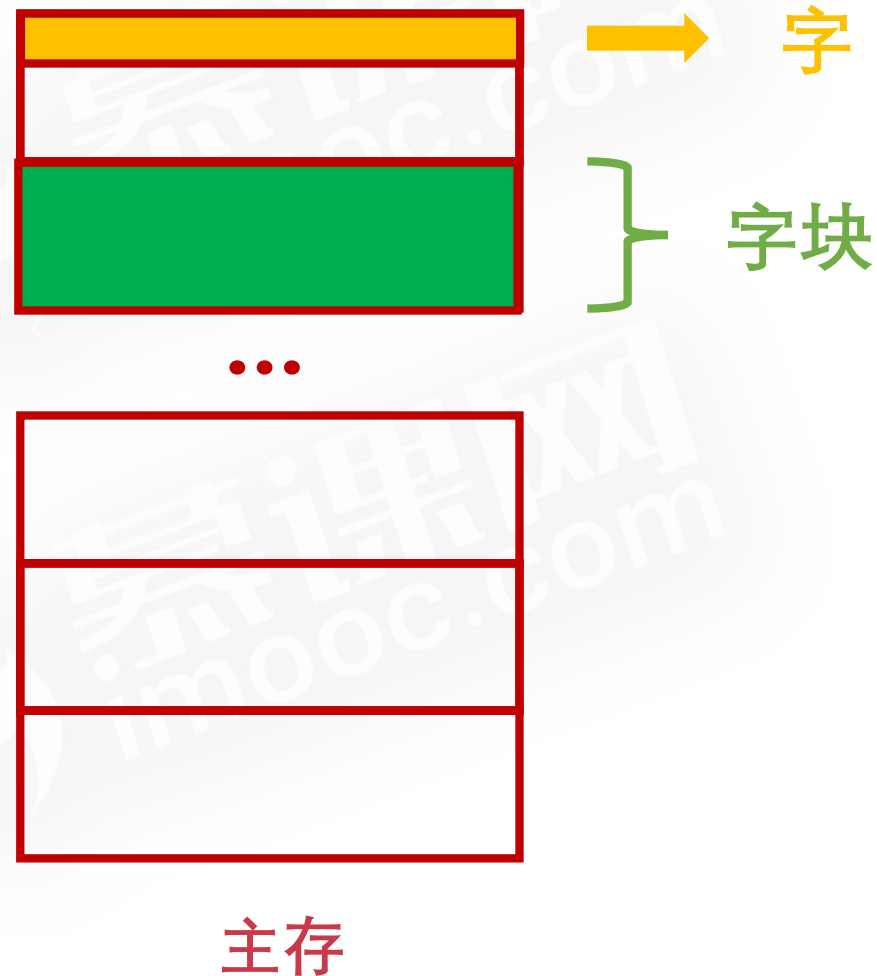
# 存储管理之段页式存储管理

## 页式存储管理

页面

◆ 字块是相对物理设备的定义

◆ 页面则是相对逻辑空间的定义



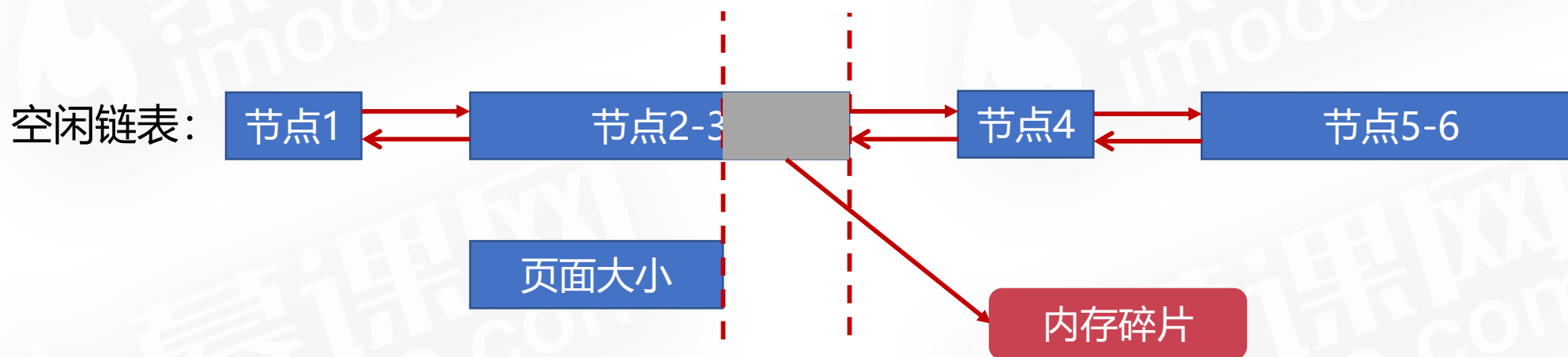
# 存储管理之段页式存储管理

## 页式存储管理

- ◆ 将进程逻辑空间等分成若干大小的页面
- ◆ 相应的把物理内存空间分成与页面大小的物理块
- ◆ 以页面为单位把进程空间装进物理内存中分散的物理块

# 存储管理之段页式存储管理

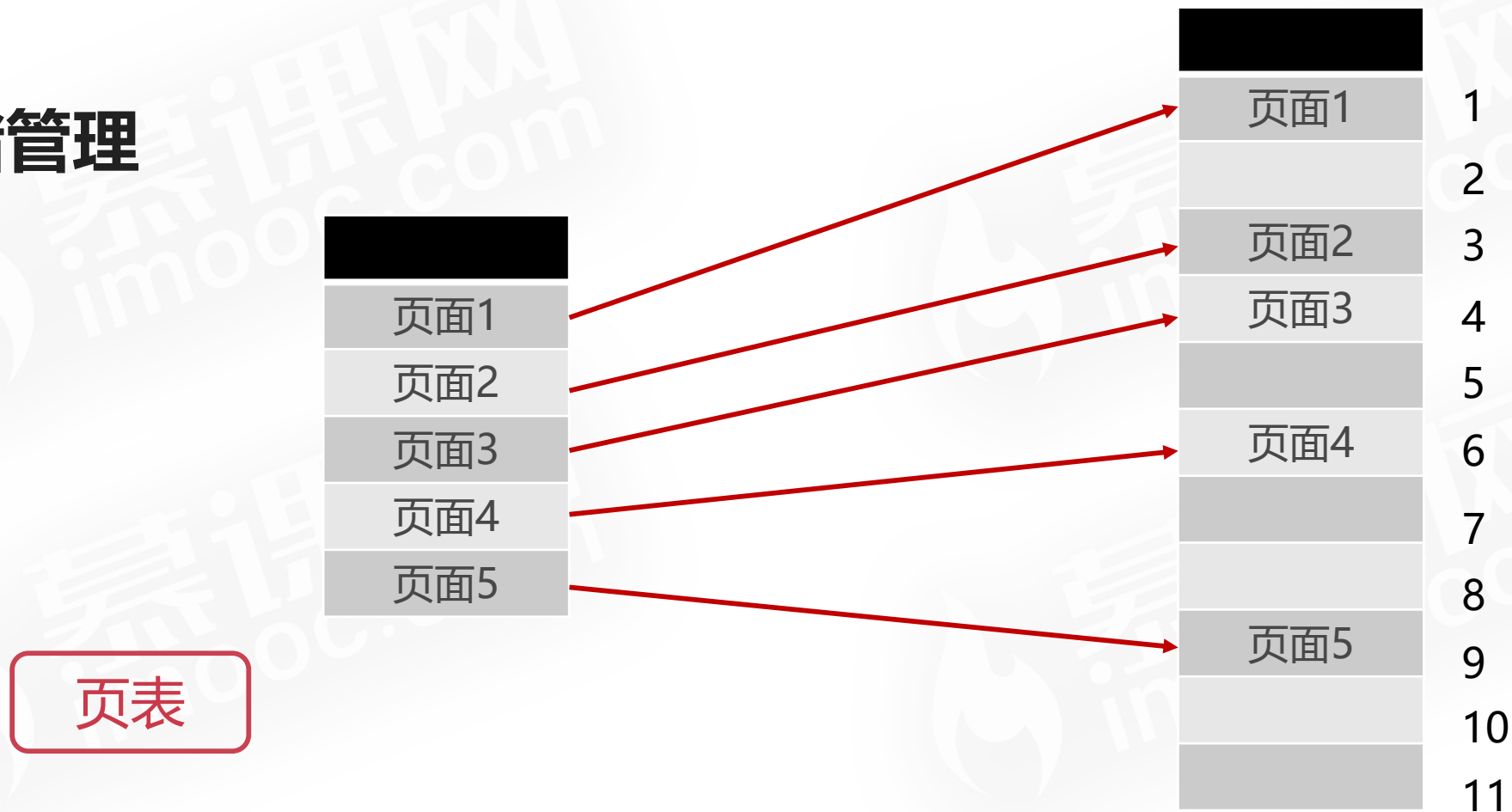
## 页式存储管理



- ◆ 页面大小应该适中，过大难以分配，过小内存碎片过多
- ◆ 页面大小通常是512B~8K

# 存储管理之段页式存储管理

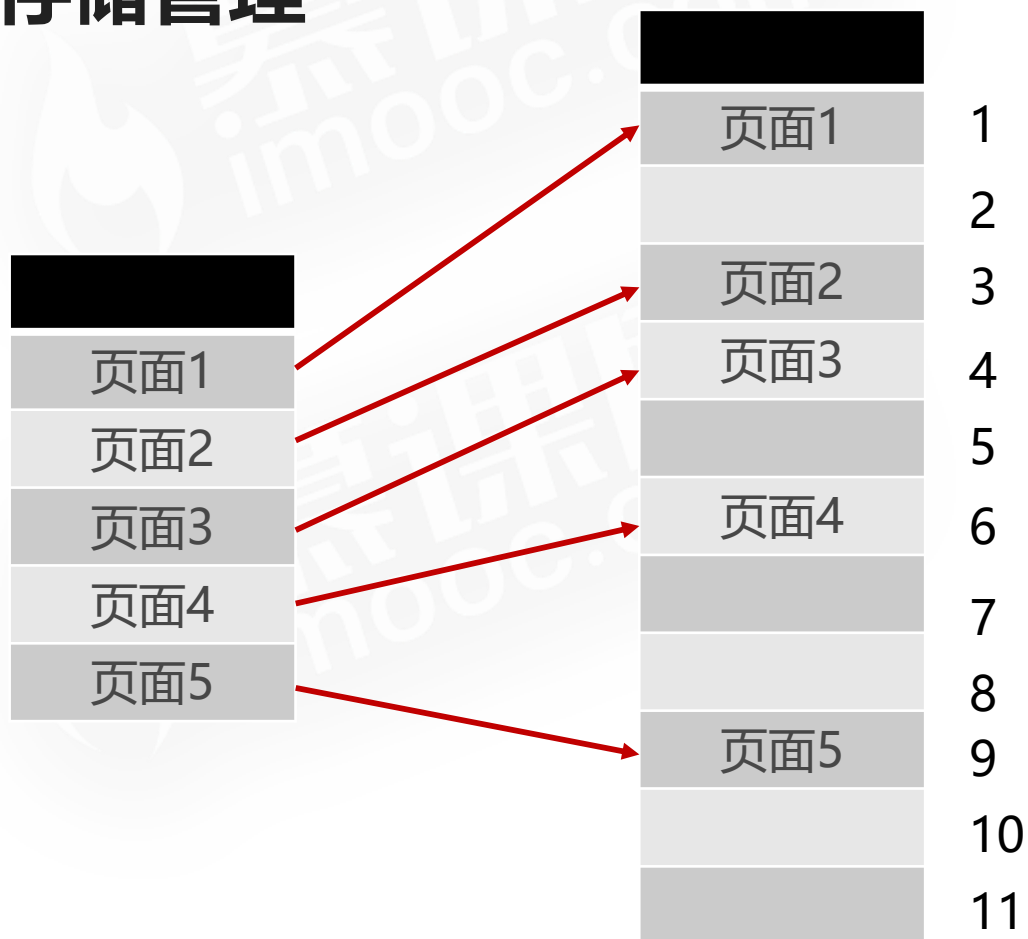
## 页式存储管理



◆ 页表记录进程逻辑空间与物理空间的映射

# 存储管理之段页式存储管理

## 页式存储管理



页面	字块
1	1
2	3
3	4
4	6
5	9

页表

页号	页内偏移
----	------

地址

# 存储管理之段页式存储管理

## 页式存储管理

现代计算机系统中，可以支持非常大的逻辑地址空间 ( $2^{32} \sim 2^{64}$ )，这样，页表就变得非常大，要占用非常大的内存空间，如，具有32位逻辑地址空间的分页系统，规定页面大小为4KB，则在每个进程页表中的页表项可达1M( $2^{20}$ )个，如果每个页表项占用1Byte，故每个进程仅仅页表就要占用1MB的内存空间。

32位系统进程的寻址空间为4G

$$4G/4KB = 2^{20}$$

多级页表



# 存储管理之段页式存储管理

## 页式存储管理

页面	字块
1	7
2	300
3	442
...	645
n	911

页面	字块
1	8
2	301
3	443
...	647
1024	913

页面	字块
1	90
2	302
3	445
...	649
1024	917

...



主存

# 存储管理之段页式存储管理

## 页式存储管理

- ◆ 将进程逻辑空间等分成若干大小的页面
- ◆ 相应的把物理内存空间分成与页面大小的物理块
- ◆ 以页面为单位把进程空间装进物理内存中分散的物理块

有一段连续的逻辑分布在多个页面中，将大大降低执行效率

# 存储管理之段页式存储管理

- ◆ 页式存储管理
- ◆ 段式存储管理

# 存储管理之段页式存储管理

## 段式存储管理

- ◆ 将进程逻辑空间划分成若干段（非等分）
- ◆ 段的长度由连续逻辑的长度决定
- ◆ 主函数MAIN、子程序段X、子函数Y等

# 存储管理之段页式存储管理

## 段式存储管理



段号	基址	段长
1	10k	30k
2	40k	10k
3	50k	40k
...	...	...
n	...	...

段表

段号	段内偏移
----	------

段地址



主存

# 存储管理之段页式存储管理

## 段式存储管理

段式存储和页式存储都离散地管理了进程的逻辑空间

- ◆ 页是物理单位，段是逻辑单位
- ◆ 分页是为了合理利用空间，分段是满足用户要求
- ◆ 页大小由硬件固定，段长度可动态变化
- ◆ 页表信息是一维的，段表信息是二维的

# 存储管理之段页式存储管理

- ◆ 页式存储管理
- ◆ 段式存储管理
- ◆ 段页式存储管理

# 存储管理之段页式存储管理

## 段页式存储管理

- ◆ 分页可以有效提高内存利用率（虽然说存在页内碎片）
- ◆ 分段可以更好满足用户需求
- ◆ 两者结合，形成段页式存储管理



# 存储管理之段页式存储管理

## 段页式存储管理

◆ 先将逻辑空间按段式管理分成若干段

◆ 再把段内空间按页式管理等分成若干页

页号	页内偏移
----	------

页地址

段号	段内偏移
----	------

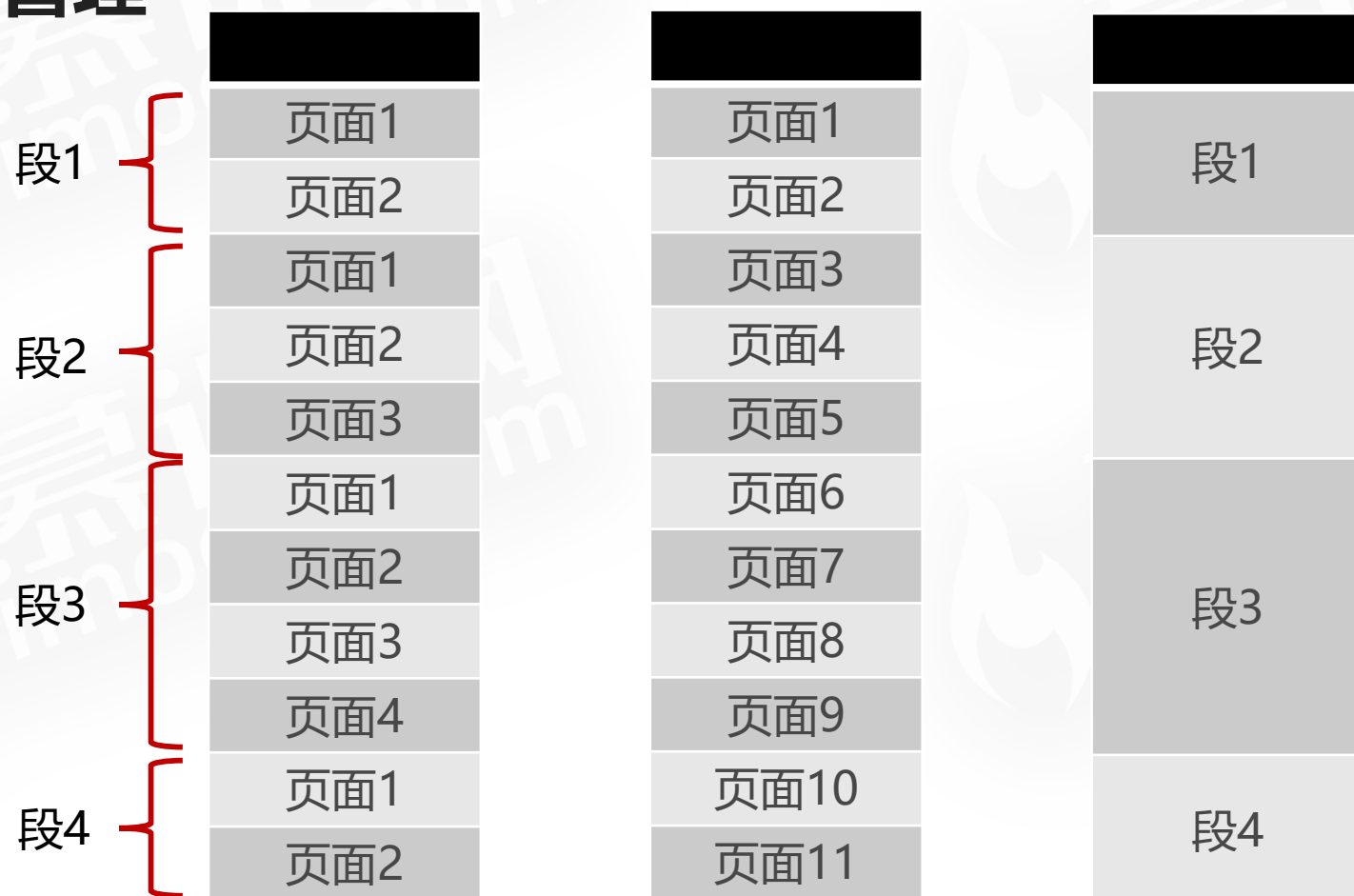
段地址

段号	段内页号	页内地址
----	------	------

段页地址

# 存储管理之段页式存储管理

## 段页式存储管理



# 存储管理之段页式存储管理

- ◆ 页式存储管理
- ◆ 段式存储管理
- ◆ 段页式存储管理



# 存储管理之虚拟内存

一个游戏十几G，物理内存只有4G，那这个游戏是怎么运行起来的？



# 存储管理之虚拟内存

- ◆ 虚拟内存概述
- ◆ 程序的局部性原理
- ◆ 虚拟内存的置换算法

# 存储管理之虚拟内存

## 虚拟内存概述

- ◆ 有些进程实际需要的内存很大，超过物理内存的容量
- ◆ 多道程序设计，使得每个进程可用物理内存更加稀缺
- ◆ 不可能无限增加物理内存，物理内存总有不够的时候

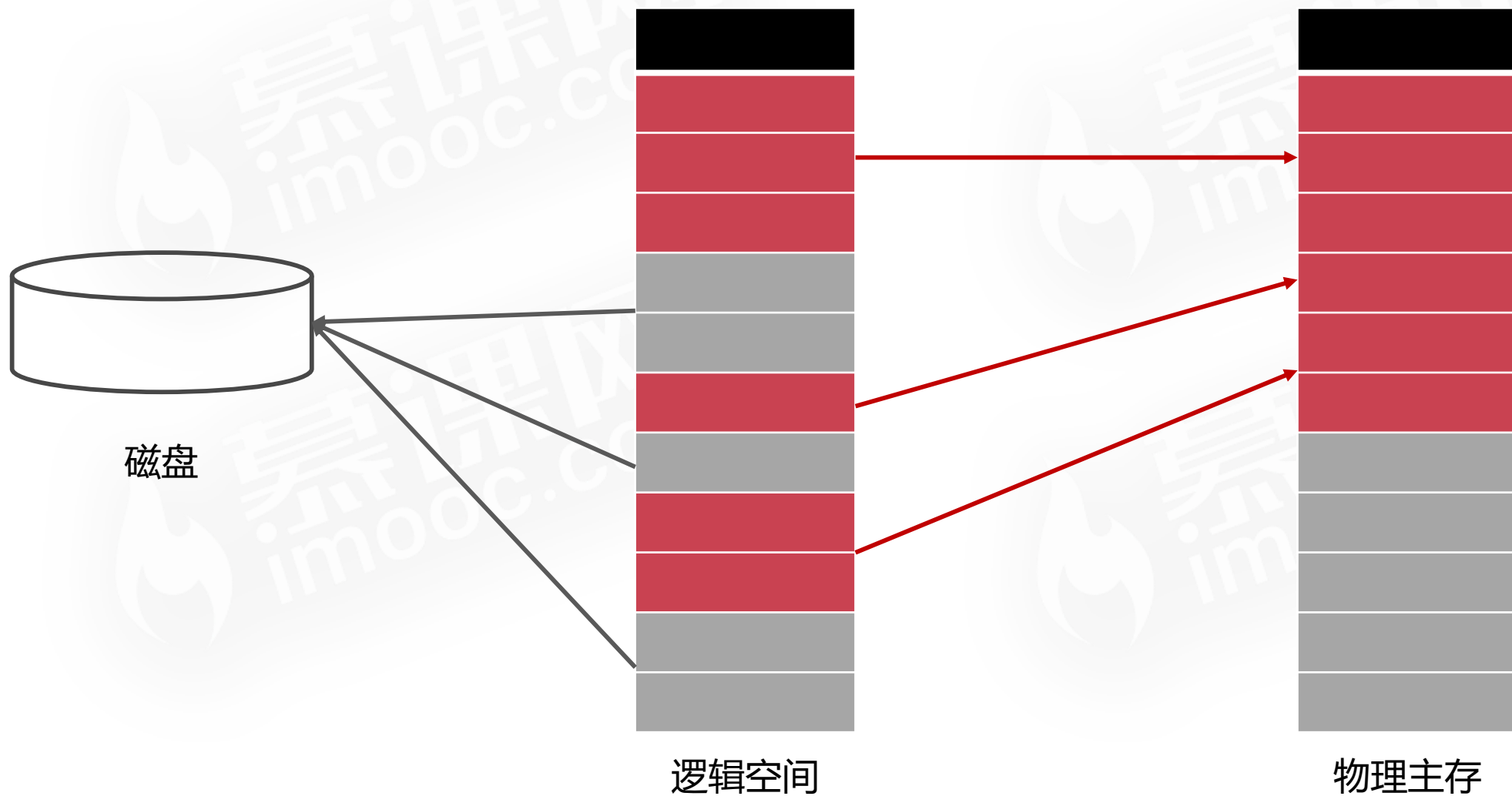
# 存储管理之虚拟内存

## 虚拟内存概述

- ◆ 虚拟内存是操作系统内存管理的关键技术
- ◆ 使得多道程序运行和大程序运行成为现实
- ◆ 把程序使用内存划分，将部分暂时不使用的内存放置在辅存



# 存储管理之虚拟内存



# 存储管理之虚拟内存

- ◆ 虚拟内存概述
- ◆ 程序的局部性原理

# 存储管理之虚拟内存

## 程序的局部性原理

局部性原理是指CPU访问存储器时，无论是存取指令还是存取数据，所访问的存储单元都趋于聚集在一个较小的连续区域中。

局部性原理

# 存储管理之虚拟内存

## 程序的局部性原理

- ◆ 程序运行时，无需全部装入内存，装载部分即可
- ◆ 如果访问页不在内存，则发出缺页中断，发起页面置换
- ◆ 从用户层面看，程序拥有很大的空间，即是虚拟内存

虚拟内存实际是对物理内存的补充，速度接近于内存，成本接近于辅存

# 存储管理之虚拟内存

- ◆ 虚拟内存概述
- ◆ 程序的局部性原理
- ◆ 虚拟内存的置换算法

# 存储管理之虚拟内存

## 虚拟内存的置换算法

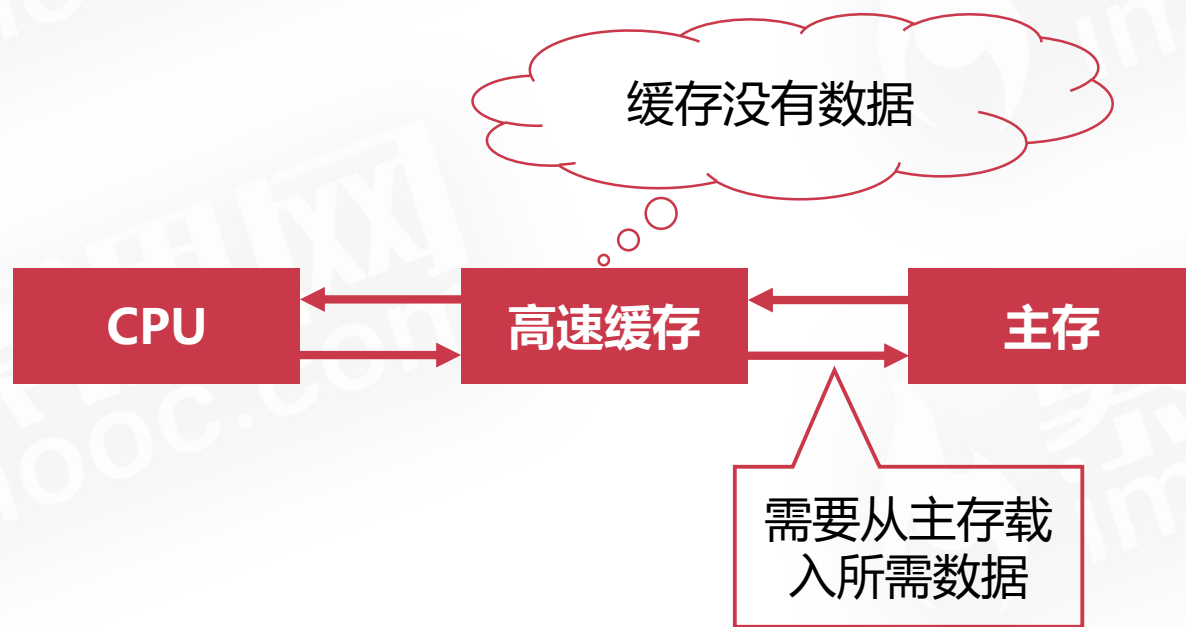
- ◆ 先进先出算法(FIFO)
- ◆ 最不经常使用算法(LFU)
- ◆ 最近最少使用算法(LRU)

这个置换算法，  
好像有点熟悉？



# 存储管理之虚拟内存

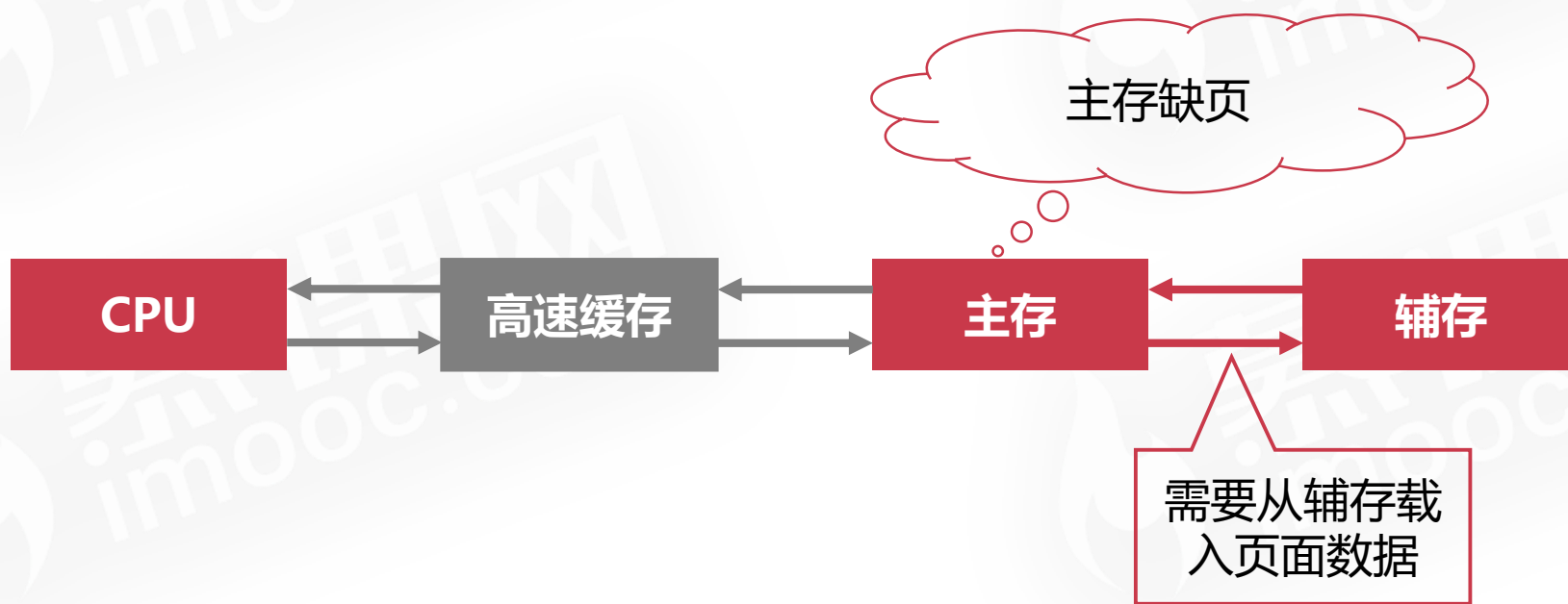
## 虚拟内存的置换算法



高速缓存的替换时机

# 存储管理之虚拟内存

## 虚拟内存的置换算法



主存页面的替换时机



# 存储管理之虚拟内存

## 虚拟内存的置换算法

- ◆ 替换策略发生在Cache-主存层次、主存-辅存层次
- ◆ Cache-主存层次的替换策略主要是为了解决速度问题
- ◆ 主存-辅存层次主要是为了解决容量问题

# 存储管理之虚拟内存

- ◆ 虚拟内存概述
- ◆ 程序的局部性原理
- ◆ 虚拟内存的置换算法



# Linux的存储管理

- ◆ Buddy内存管理算法
- ◆ Linux交换空间

# Linux的存储管理

## Buddy内存管理算法

- ◆ Buddy算法是经典的内存管理算法
- ◆ 算法基于计算机处理二进制的优势具有极高的效率
- ◆ 算法主要是为了解决内存外碎片的问题

# Linux的存储管理

## Buddy内存管理算法

内部碎片是已经被分配出去（能明确指出属于哪个进程）的内存空间大于请求所需的内存空间，不能被利用的内存空间就是内部碎片。

页内碎片

外部碎片是指还没有分配出去（不属于任何进程），但是由于大小而无法分配给申请内存空间的新进程的内存空闲块。

页外碎片

# Linux的存储管理

## Buddy内存管理算法



页内碎片



页外碎片

# Linux的存储管理

## Buddy内存管理算法

努力让内存分配与相邻内存合并能快速进行



# Linux的存储管理

## Buddy内存管理算法

◆ 向上取整为2的幂大小

◆ 70k→128k

◆ 129k→256k

◆ 666k→1024k

内存分配原则

# Linux的存储管理

## Buddy内存管理算法

- ◆ “伙伴” 指的是内存的 “伙伴”
- ◆ 一片连续内存的 “伙伴” 是相邻的另一片大小一样的连续内存

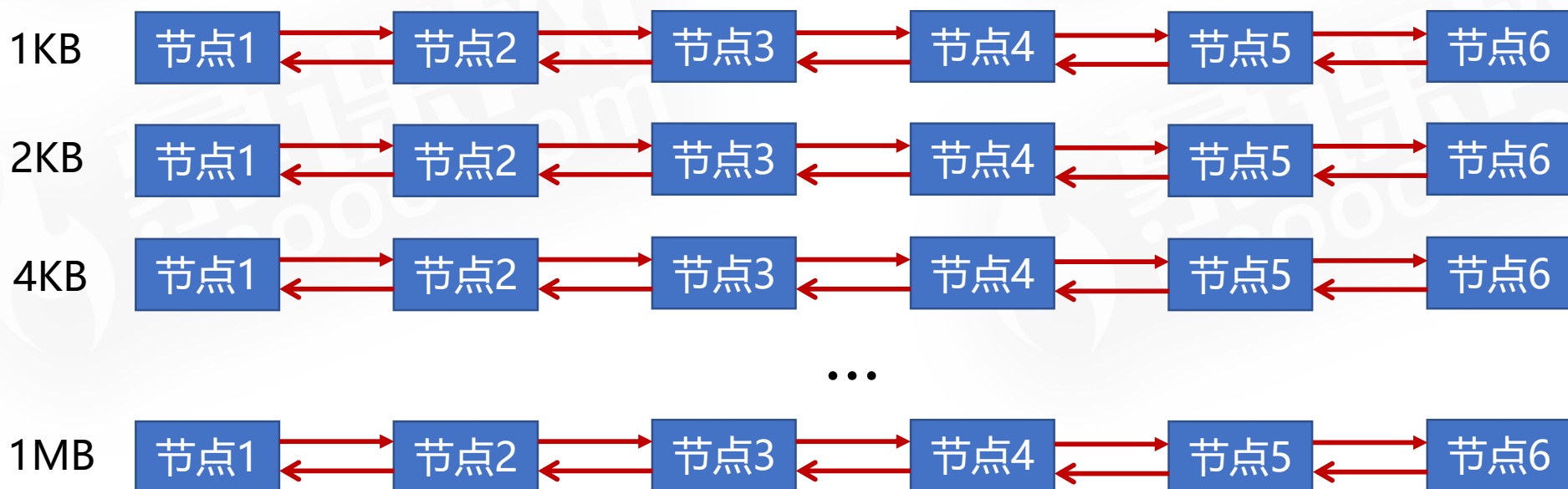
伙伴系统



# Linux的存储管理

## Buddy内存管理算法

- ◆ 创建一系列空闲块链表，每一种都是2的幂



# Linux的存储管理

## Buddy内存管理算法

- ◆ 假设存储空间有1M大小

1KB

NULL

2KB

NULL

4KB

NULL

...

1MB

1MB



# Linux的存储管理

## Buddy内存管理算法

### ◆ 分配100k内存

1. 100k向上取2的幂=128k
2. 查询是否有128k空闲内存块?
3. 没有! 查询是否有256k空闲内存块?
4. 没有! 查询是否有512k空闲内存块?
5. 没有! 查询是否有1M空闲内存块?

1KB	NULL
2KB	NULL
4KB	NULL
...	
1MB	1MB



# Linux的存储管理

## Buddy内存管理算法

### ◆ 分配100k内存

6. 有，摘下1M空闲内存块，分配出去
7. 拆下512k放在512k的空闲链表，其余的分配出去
8. 拆下256k放在256k的空闲链表，其余的分配出去
9. 拆下128k放在128k的空闲链表，其余的分配出去
10. 分配完毕

1KB	NULL
...	
128KB	节点1
256KB	节点1
512KB	节点1
1MB	NULL

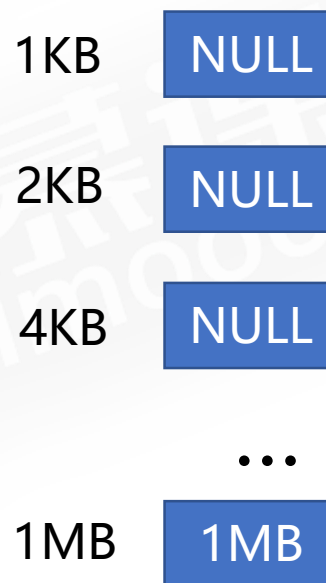


# Linux的存储管理

## Buddy内存管理算法

### ◆ 回收刚才分配的内存

1. 判断刚才分配的内存伙伴在空闲链表上吗？
2. 在！移除伙伴，合并为256k空闲内存，判断
3. 在！移除伙伴，合并为512k空闲内存，判断
4. 在！移除伙伴，合并为1M空闲内存
5. 插入1M空闲链表，回收完成



# Linux的存储管理

## Buddy内存管理算法

- ◆ Buddy算法是经典的内存管理算法
- ◆ 算法基于计算机处理二进制的优势具有极高的效率
- ◆ 算法主要是为了解决内存外碎片的问题

内存外碎片问题



内存内碎片问题



# Linux的存储管理

- ◆ Buddy内存管理算法
- ◆ Linux交换空间

# Linux的存储管理

## Linux交换空间

- ◆ 交换空间(Swap)是磁盘的一个分区
- ◆ Linux物理内存满时，会把一些内存交换至Swap空间
- ◆ Swap空间是初始化系统时配置的

# Linux的存储管理

## Linux交换空间

查看系统Swap空间→

# Linux的存储管理

## Linux交换空间

- ◆ 冷启动内存依赖
- ◆ 系统睡眠依赖
- ◆ 大进程空间依赖

# Linux的存储管理

## Linux交换空间

- ◆ Swap空间存在于磁盘
- ◆ Swap空间与主存发生置换
- ◆ Swap空间是操作系统概念
- ◆ Swap空间解决系统物理内存不足问题

Swap空间

VS

- ◆ 虚拟内存存在于磁盘
- ◆ 虚拟内存与主存发生置换
- ◆ 虚拟内存是进程概念
- ◆ 虚拟内存解决进程物理内存不足问题

虚拟内存

# Linux的存储管理

- ◆ Buddy内存管理算法
- ◆ Linux交换空间

