

第十一章、文件系统的实现

- 1. 存储和访问文件的有关问题**
- 2. 分配磁盘空间的问题**
- 3. 空闲空间的管理问题**

目录

1. 文件系统结构
2. 文件系统实现
3. 目录实现方法
4. 磁盘空间的分配方法
5. 空闲空间管理
6. 效率与性能
7. 恢复
8. 基于日志结构的文件系统

11.1 文件系统结构

- I. 磁盘的逻辑单元为块儿 (block)
- II. 内存与磁盘之间的传输是以块儿为单位

■ 磁盘的特点

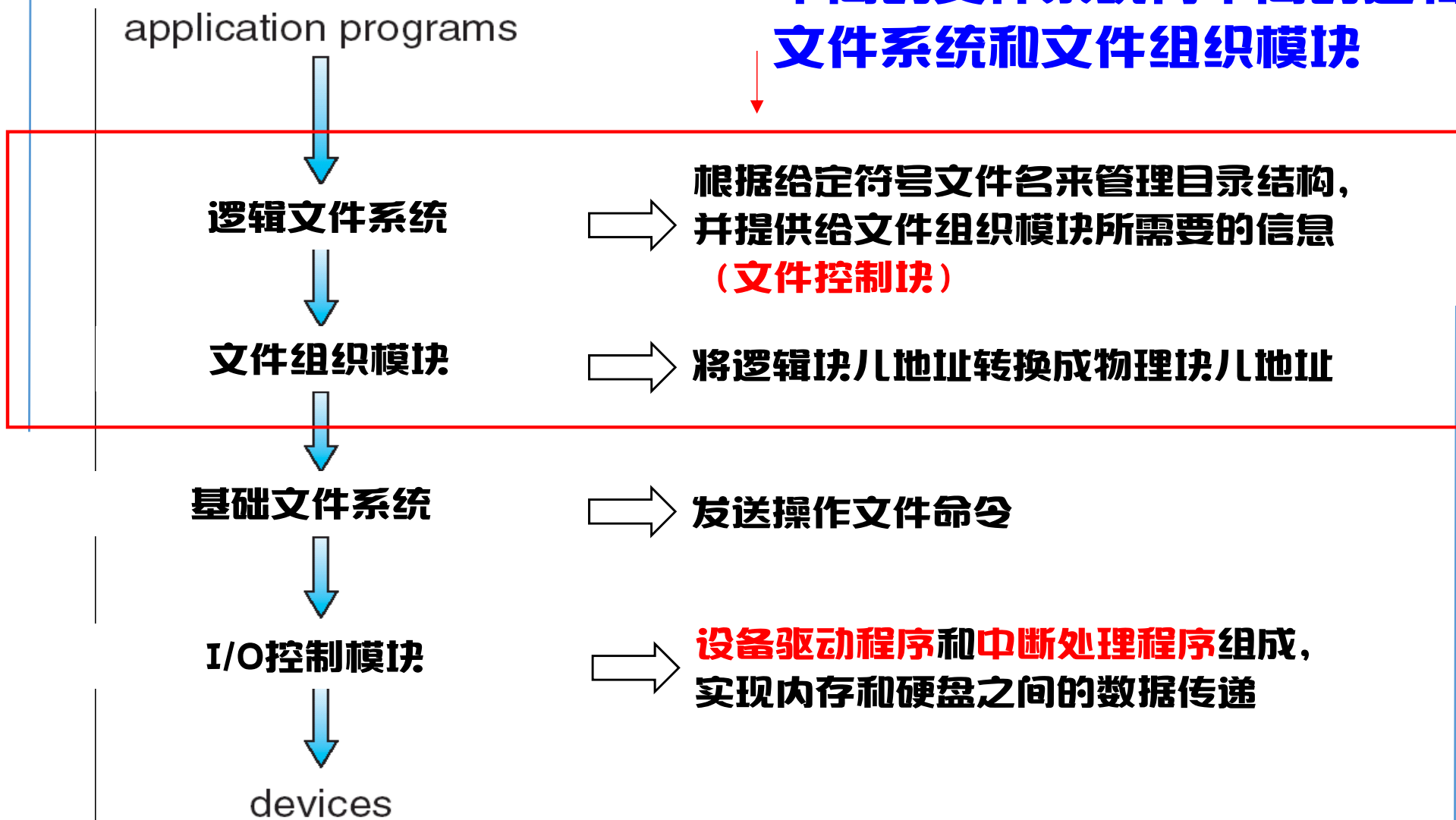
- 1. 可以原地重写，可以从磁盘上读一块儿，修改该块儿，并将它写回到原来的位置
- 2. 可以直接访问磁盘上的任意一块。因此，可以方便地按顺序或随机访问文件。

文件系统结构

- **文件系统需要提供的机制**
 - **存储文件**
 - **访问文件**
- **文件系统有两个不同的设计问题**
 1. **访问问题：如何定义文件系统对用户的接口**
 2. **存储问题：如何把逻辑文件系统映射到物理外存设备**
 - **需要创建数据结构和算法**
 - **通常分层次设计文件系统**

文件系统的层次结构

不同的文件系统有不同的逻辑
文件系统和文件组织模块



逻辑文件系统：文件控制块

文件控制块儿：File Control Block: FCB

file permissions
file dates (create, access, write)
file owner, group, ACL
file size
file data blocks or pointers to file data blocks

文件系统

大多数操作系统支持多种不同的文件系统， 举例：

- **CD-ROM ISO9660 文件格式**
- **Unix 文件系统 (Unix File System)**
- **Windows**
 - **FAT (File Allocation Table) , FAT32, NTFS (Windows NT File System)**
- **Linux**
 - **可扩展文件系统 (Extended file system)**
 - **Ext2, ext3, ext4 ...**
 - **分布式文件系统* (Distributed File System)**

文件控制块

- 文件控制块儿: File Control Block (FCB)
- 在Linux系统中, 文件控制块儿叫做 i-node (index node)

文件管理表 → I-node 表

Linux 系统当中的每个文件都有唯一的 i-node

- “\$ls -l” shows the **i-node number** of the files
- “\$stat file_name”

i-node(FCB) in linux

i-node number

```
root@hbpark-OptiPlex-390: /home/hbpark
root@hbpark-OptiPlex-390:/home/hbpark# ls -il
total 84
7471117 drwxr-xr-x 2 hbpark hbpark 4096 Mar 13 17:19 Desktop
7471121 drwxr-xr-x 2 hbpark hbpark 4096 Mar  6 16:32 Documents
7471118 drwxr-xr-x 4 hbpark hbpark 4096 Mar  6 17:09 Downloads
7471108 -rw-r--r-- 1 hbpark hbpark 8445 Mar 26 2013 examples.desktop
7471122 drwxr-xr-x 2 hbpark hbpark 4096 Mar 26 2013 Music
7471123 drwxr-xr-x 2 hbpark hbpark 4096 Mar 26 2013 Pictures
7471120 drwxr-xr-x 2 hbpark hbpark 4096 Mar 26 2013 Public
7471119 drwxr-xr-x 2 hbpark hbpark 4096 Mar 26 2013 Templates
7471392 -rwxrwxr-x 1 hbpark hbpark 7366 Mar 17 11:38 test
7471232 -rw-rw-r-- 1 hbpark hbpark 102 Mar 17 11:30 test.c
7471390 -rw-rw-r-- 1 hbpark hbpark 17960 Mar 17 11:33 test.i
7471391 -rw-rw-r-- 1 hbpark hbpark 1040 Mar 17 11:37 test.o
7471353 -rw-rw-r-- 1 hbpark hbpark 548 Mar 17 11:35 test.s
7471124 drwxr-xr-x 2 hbpark hbpark 4096 Mar 26 2013 Videos
root@hbpark-OptiPlex-390:/home/hbpark#
```

i-node(FCB) in linux

```
root@hbpark-OptiPlex-390: /home/hbpark
Access: 2014-04-23 20:18:11.664668166 +0800
Modify: 2014-03-17 11:33:13.992234715 +0800
Change: 2014-03-17 11:33:13.992234715 +0800
Birth: -
root@hbpark-OptiPlex-390:/home/hbpark#
root@hbpark-OptiPlex-390:/home/hbpark#
root@hbpark-OptiPlex-390:/home/hbpark#
root@hbpark-OptiPlex-390:/home/hbpark# stat test.i
File: `test.i'
Size: 17960          Blocks: 40          IO Block: 4096   regular file
Device: 801h/2049d   Inode: 7471390    Links: 1
Access: (0664/-rw-rw-r--)  Uid: ( 1000/   hbpark)   Gid: ( 1000/   hbpark)
Access: 2014-04-23 20:18:11.664668166 +0800
Modify: 2014-03-17 11:33:13.992234715 +0800
Change: 2014-03-17 11:33:13.992234715 +0800
Birth: -
root@hbpark-OptiPlex-390:/home/hbpark#
root@hbpark-OptiPlex-390:/home/hbpark#
root@hbpark-OptiPlex-390:/home/hbpark#
```

i-node(FCB) in linux

\$ df -i (report file system disk space usage)

```
root@hbpark-OptiPlex-390:/home/hbpark# df -i
Filesystem          Inodes   IUsed   IFree IUse% Mounted on
/dev/sda1            14745600 195788 14549812    2% /
udev                 198559    484   198075    1% /dev
tmpfs                202436    396   202040    1% /run
none                 202436     4   202432    1% /run/lock
none                 202436     6   202430    1% /run/shm
none                 202436    20   202416    1% /run/user
root@hbpark-OptiPlex-390:/home/hbpark#
```

1. i-node size

\$dumpe2fs -h /dev/sda1 | grep "Inode size"

i-node size = 256

2. i-node struct

Linux source file /include/linux/fs.h

```

root@hbpark-OptiPlex-390: /home/hbpark
tune2fs 1.42.5 (29-Jul-2012)
Filesystem volume name:   <none>
Last mounted on:         /
Filesystem UUID:          11e3e6b6-a711-4322-bdf9-9348fe49322b
Filesystem magic number:  0xEF53
Filesystem revision #:    1 (dynamic)
Filesystem features:      has_journal ext_attr resize_inode dir_index filetype n
eeds_recovery extent flex_bg sparse_super large_file huge_file uninit_bg dir_nli
nk extra_isize
Filesystem flags:          signed_directory_hash
Default mount options:    user_xattr acl
Filesystem state:         clean
Errors behavior:          Continue
Filesystem OS type:       Linux
Inode count:              14745600
Block count:              58978816
Reserved block count:    2948940
Free blocks:              53671887
Free inodes:              14549848
First block:              0
Block size:               4096
Fragment size:            4096
Reserved GDT blocks:      1009
Blocks per group:         32768
Fragments per group:      32768
Inodes per group:         8192

```

`$tune2fs -l /dev/sda1`

List the contents of the filesystem superblock

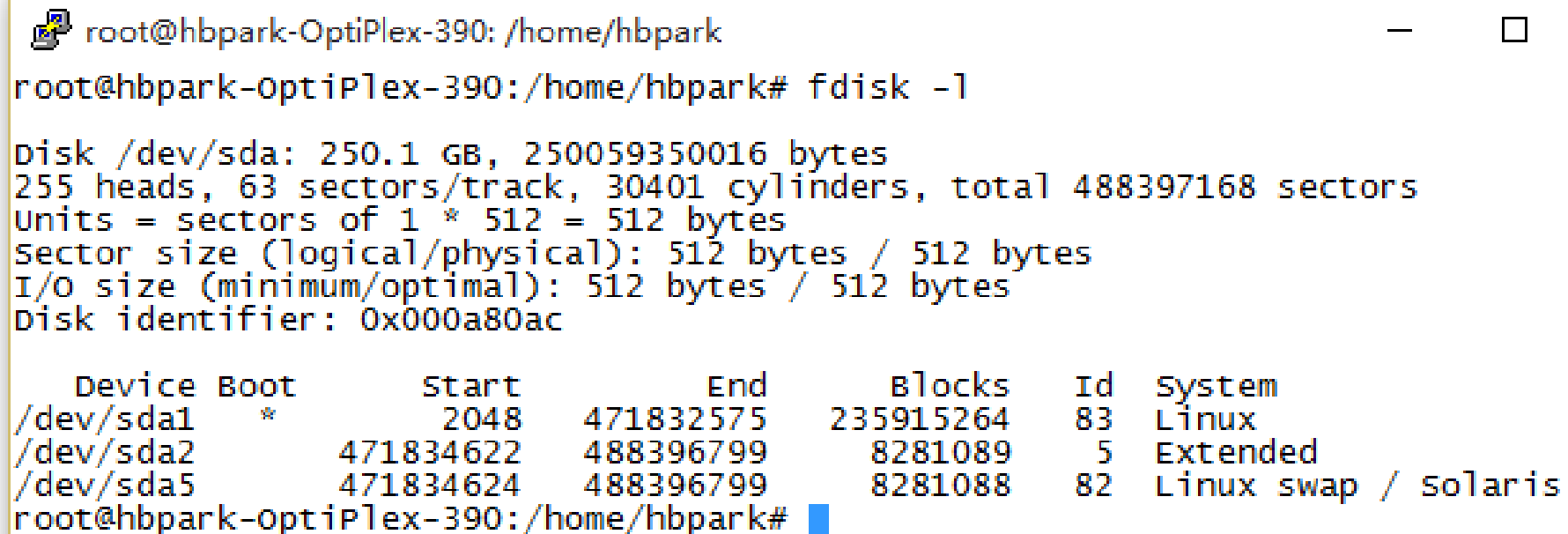
```

Lifetime writes:          110 GB
Reserved blocks uid:      0 (user root)
Reserved blocks gid:      0 (group root)
First inode:              11
Inode size:               256
Required extra isize:     28
Desired extra isize:      28
Journal inode:            8
First orphan inode:       7477603
Default directory hash:   half_md4
Directory Hash Seed:      0bf59e1d-7171-4da9-b129-57d72768f322
Journal backup:           inode blocks
root@hbpark-OptiPlex-390: /home/hbpark#

```

查看分区 in linux

\$ fdisk -l



```
root@hbpark-OptiPlex-390: /home/hbpark
root@hbpark-OptiPlex-390:/home/hbpark# fdisk -l

Disk /dev/sda: 250.1 GB, 250059350016 bytes
255 heads, 63 sectors/track, 30401 cylinders, total 488397168 sectors
Units = sectors of 1 * 512 = 512 bytes
Sector size (logical/physical): 512 bytes / 512 bytes
I/O size (minimum/optimal): 512 bytes / 512 bytes
Disk identifier: 0x000a80ac

   Device Boot      Start         End      Blocks   Id  System
/dev/sda1  *          2048     471832575     235915264    83   Linux
/dev/sda2             471834622     488396799      8281089      5   Extended
/dev/sda5             471834624     488396799      8281088     82   Linux swap / solaris
root@hbpark-OptiPlex-390:/home/hbpark#
```

11.2 文件系统的实现

在磁盘上，文件系统包括的信息有如何**启动操作系统**、**总的块数**、**空闲块的数目和位置**、**目录结构**以及**各个具体文件** 等

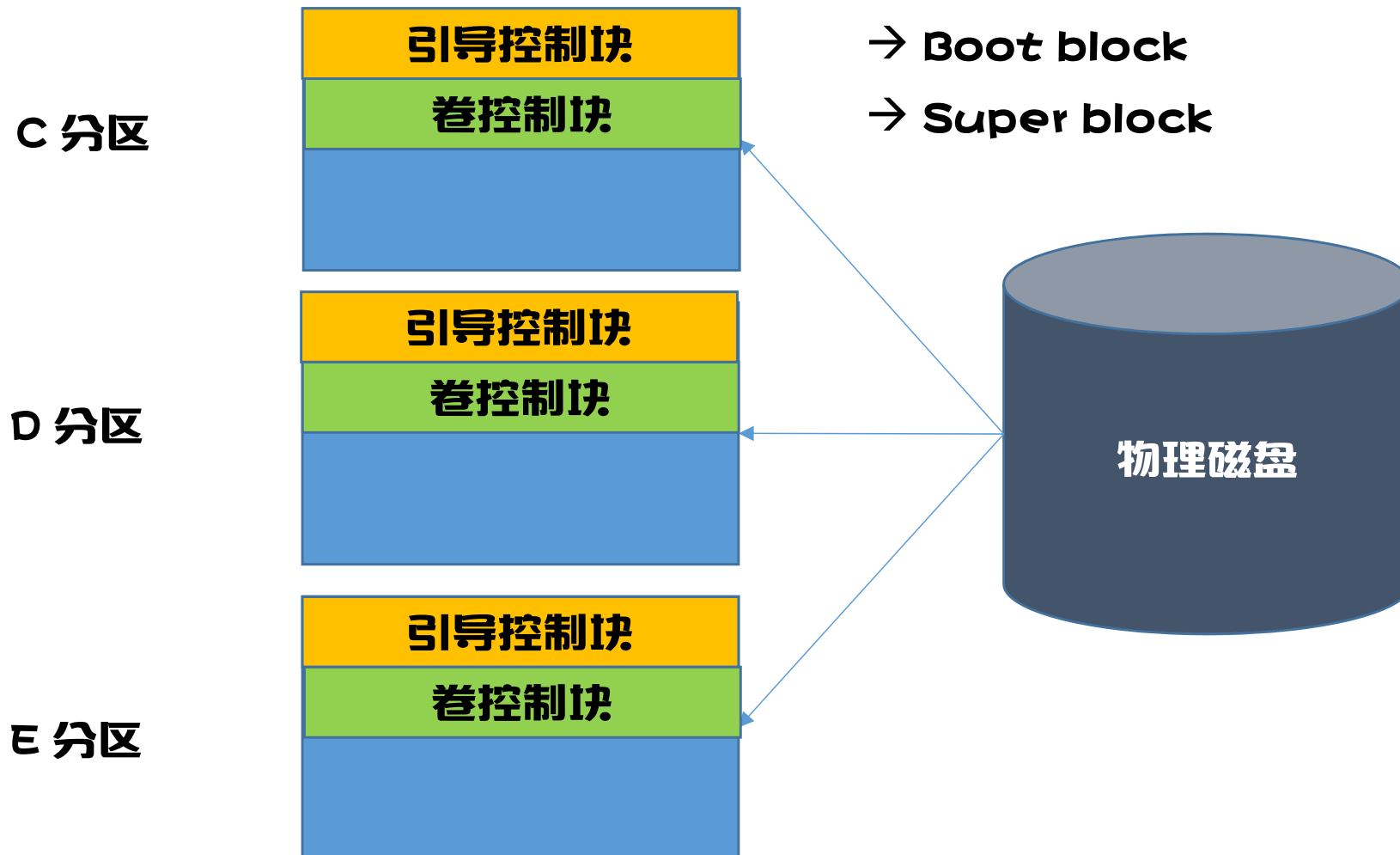
1. 每个卷的**引导控制块**

引导操作系统所需要的信息。UFS称为**引导块** (boot block)，NFS称为**分区引导扇区** (partition boot sector)

2. 每个卷的**卷控制块**

卷的详细信息（分区的块数、块的大小、空闲块的数量和指针、空闲FCB的数量和指针 等）。UFS称为**超级块儿** (super block)，NTFS主控文件表 (master boot sector)

文件系统的实现



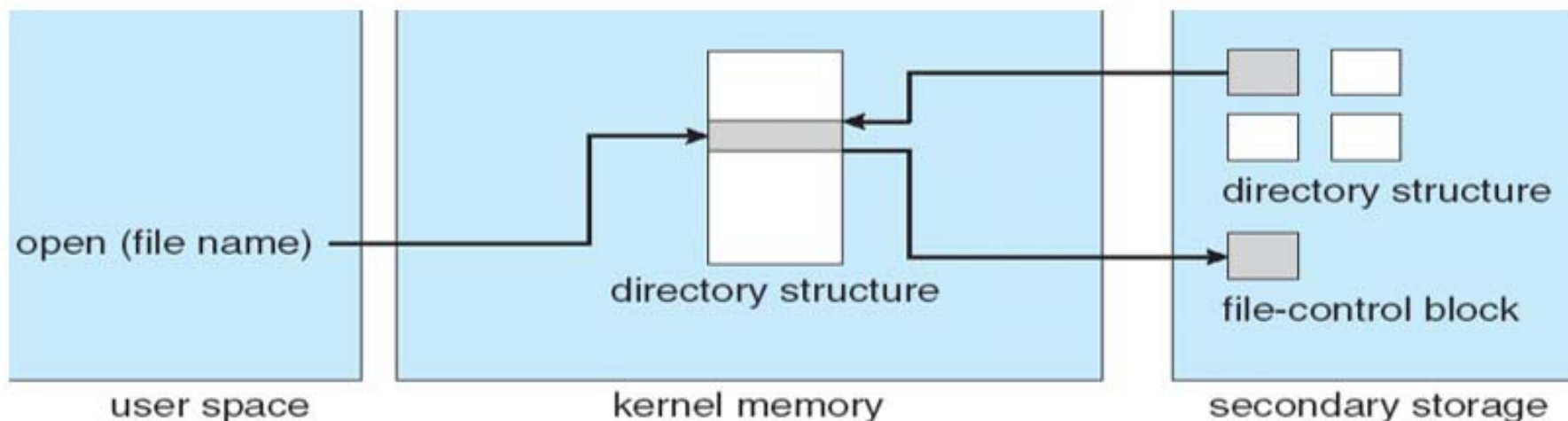
内存中的文件系统结构

创建一个新文件

1. 分配一个新的FCB
2. 相应的目录信息读入内存
3. 更新目录结构和FCB
4. 将结果写入磁盘

打开文件

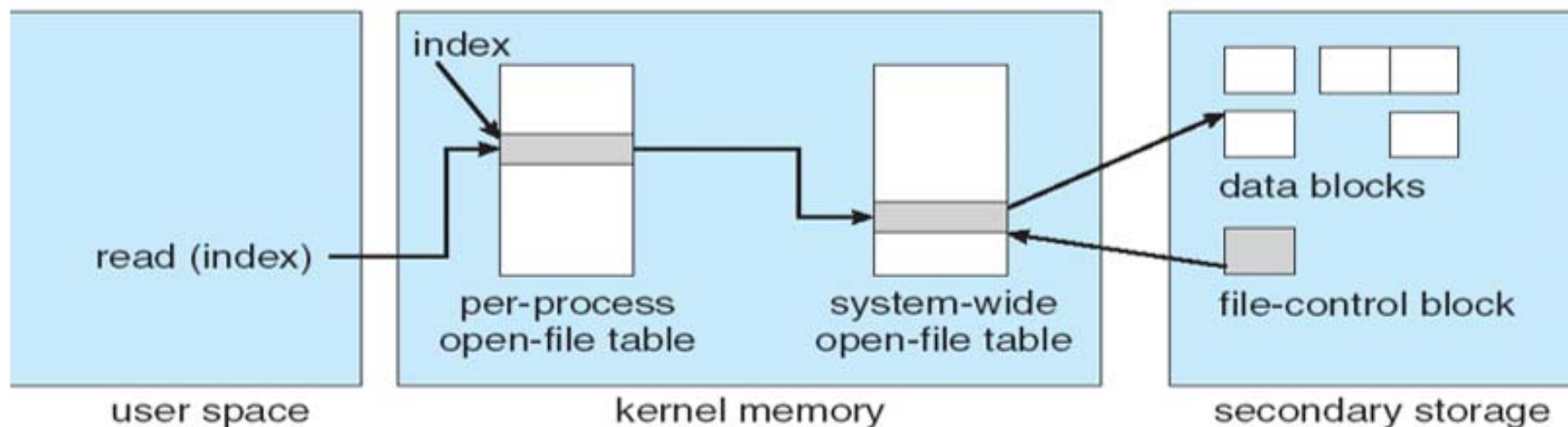
1. 根据给定的文件名来搜索目录结构
2. 如果有，就将其FCB复制到系统范围内的打开文件表
3. 如果没有，就创建文件



内存中的文件系统结构

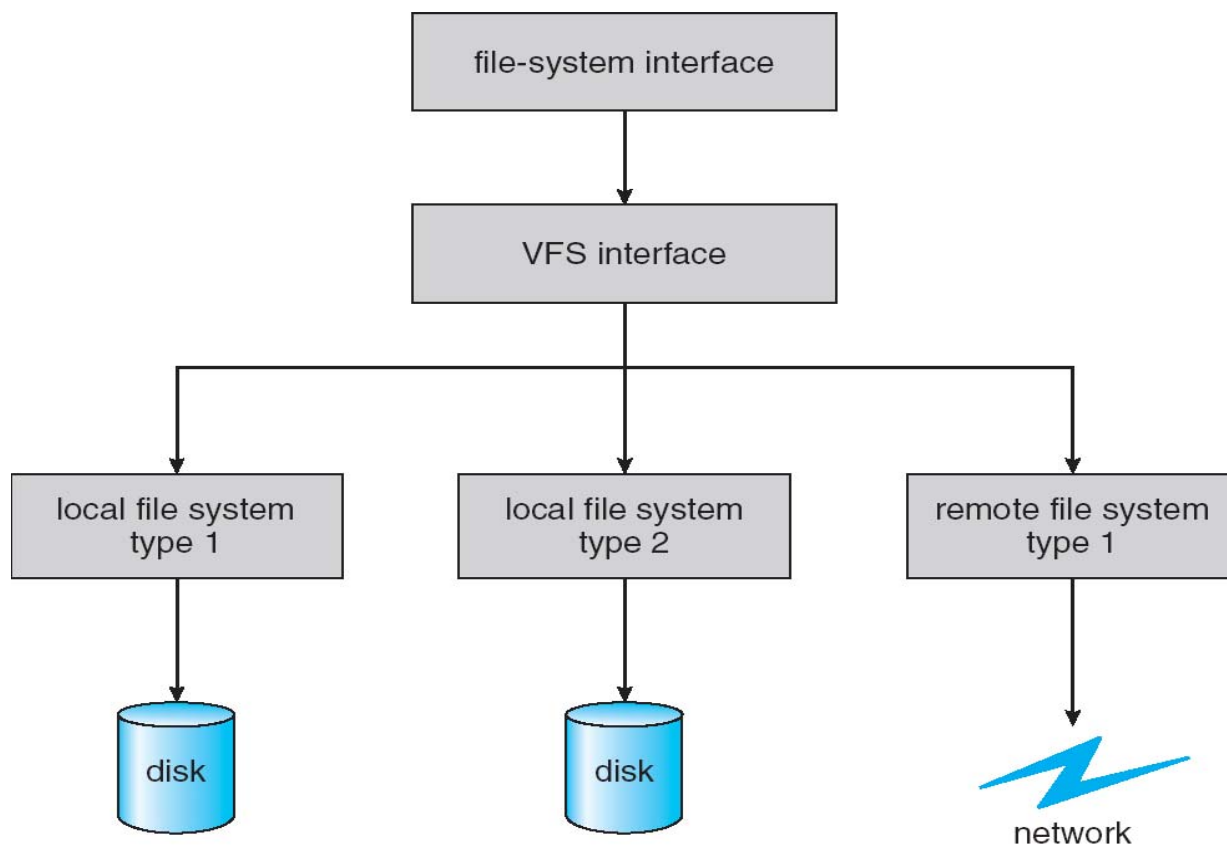
1. **单个进程打开文件表**：包括指向系统范围内已打开文件表中合适条目的指针和其他信息
2. **系统范围内的打开文件表**：包括每个打开文件的FCB副本和其他信息

举例：Open() 返回值：文件描述符是一个非负整数。它是一进程打开文件表的索引值，指向系统范围内打开文件表相应条目



虚拟文件系统

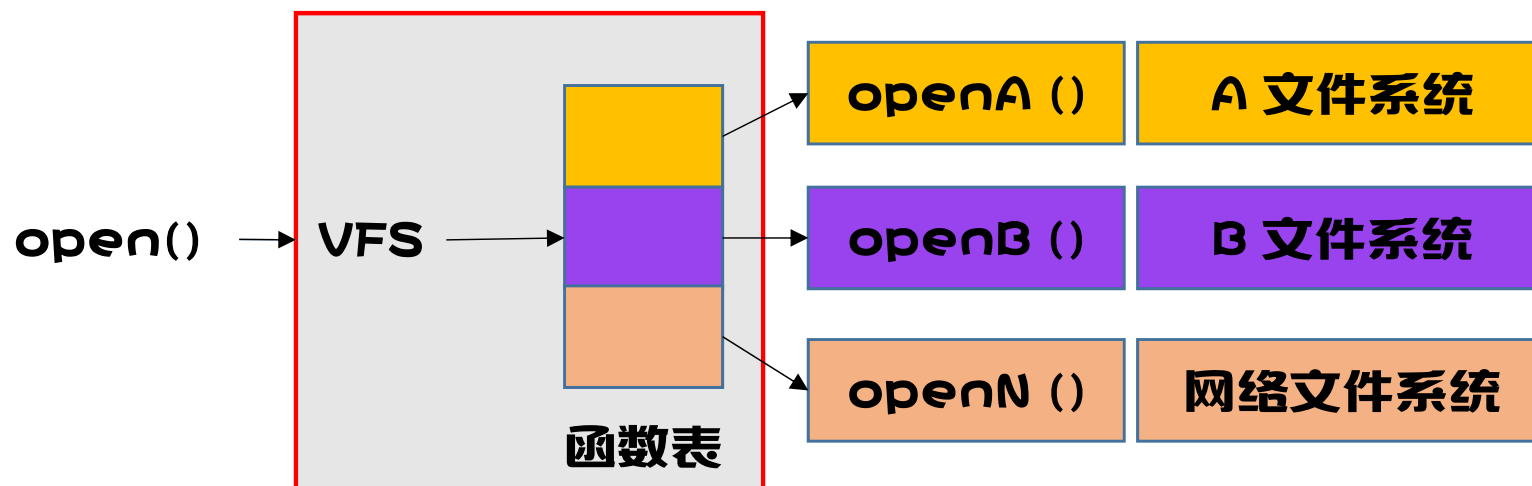
- Virtual File Systems (VFS)
- 给不同的文件系统类型提供**统一的系统调用接口**
- 提供面向对象技术来**实现文件管理**



虚拟文件系统示意图

虚拟文件系统

- 把文件系统的**通用操作**和**具体实现分开**
- 虚拟文件系统提供了在网络上唯一标识一个文件的机制。VFS基于 `vnode` 文件表示结构，它包含了一个数值标识符 → 表示位于整个网络范围内的唯一文件
 1. 能区分不同本地文件系统
 2. 能区分本地文件系统和远程文件系统



11.3 文件目录的实现

1. 线性列表：使用存储文件名和数据块指针

- **【文件名，数据块指针】**
- 优点：编程简单
- 缺点：因为需要搜索，运行较为费时

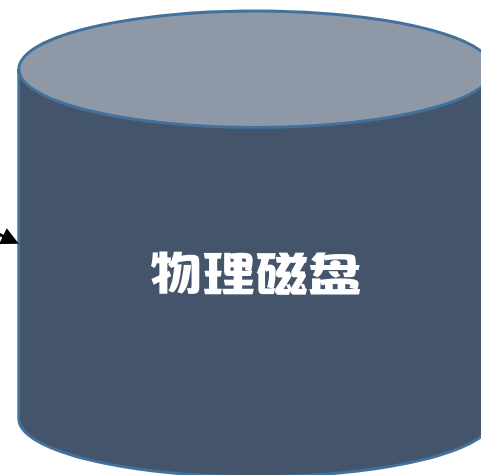
2. 哈希表：哈希表根据文件名得到一个值，并返回一个指向线性列表中元素的指针

- 优点：减少目录搜索时间
- 缺点：两个文件名哈希到相同的位置时可能发生冲突；因哈希表固定大小，创建文件需要哈希表重建时，比较麻烦。

文件目录的实现

文件名	数据块指针
文件名	数据块指针
文件名	数据块指针
文件名	数据块指针
...	...

线性列表

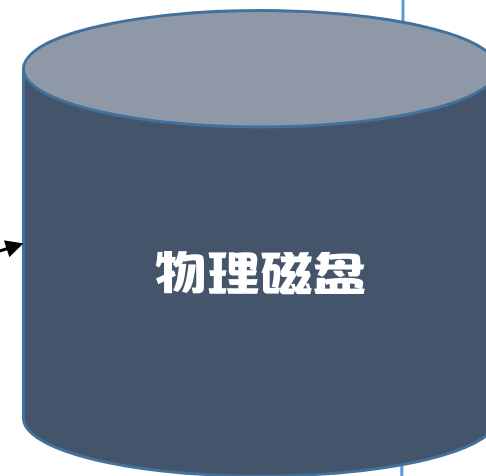


文件名	哈希值
文件名	哈希值
文件名	哈希值
文件名	哈希值
...	...

哈希表

文件名	数据块指针
文件名	数据块指针
文件名	数据块指针
文件名	数据块指针
文件名	数据块指针
...	...

线性列表



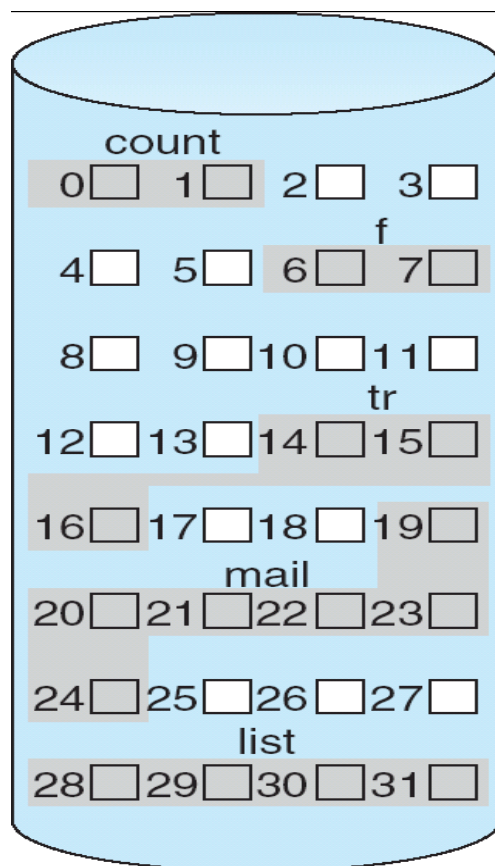
11.4 磁盘空间的分配方法

如何为文件分配磁盘空间



(1) 连续分配方式

每个文件在磁盘上占有一组连续的块儿， 文件用文件**第一块的磁盘地址**和**连续块的数量（即长度）**来定义



directory		
file	start	length
count	0	2
tr	14	3
mail	19	6
list	28	4
f	6	2

【第一块的磁盘地址， 长度】

连续分配方式

连续分配支持顺序访问和直接访问

会出现的问题

-当文件需要扩展，文件大小变大时会无法扩展

解决的方法

- I. 可以终止用户程序**
- II. 找一个更大的孔，复制文件内容到新空间**

连续分配方式问题举例

1	2	3	4	5
---	---	---	---	---

文件一 (1, 5)

6	7	8	9	10
---	---	---	---	----

文件二 (6, 5)

11	12	13	14	15
----	----	----	----	----

文件三 (11, 5)

--	--	--	--	--

--	--	--	--	--

--	--	--	--	--

基于扩展的连续分配方案

用以下参数来定义文件

I. 开始地址

II. 块儿数

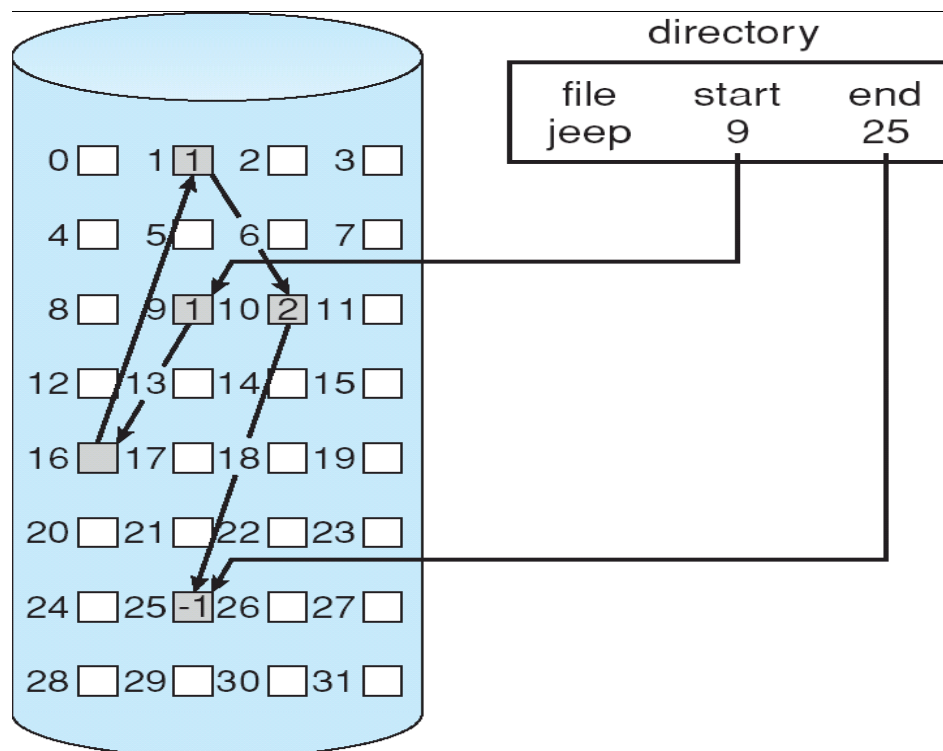
III. 指向下一个扩展块儿的指针（扩展块儿可以是多个）

定义格式：

文件【开始地址，块儿数，指向下一个扩展块儿的指针】

(2) 链接分配

- 每个文件是磁盘块儿的链表，磁盘块儿分布在磁盘的任何地方，文件有起始块和结束块来定义
- 定义格式：文件【起始块，结束块】，
同时，每个磁盘块都有指向下一个磁盘块的地址。



链接分配

优点:

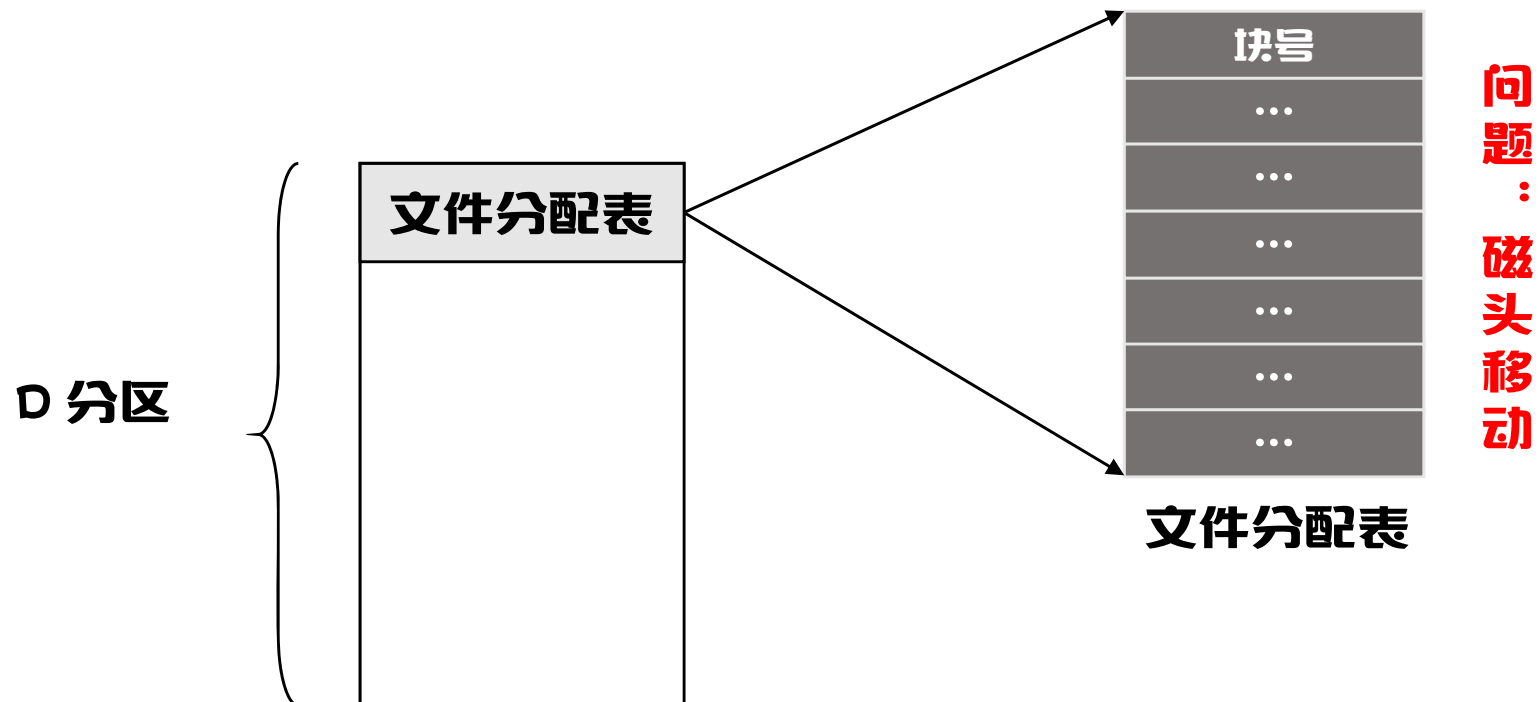
没有磁盘空间浪费

缺点:

- I. 不支持文件的直接访问**
- II. 需要更多的磁盘空间（指针）**

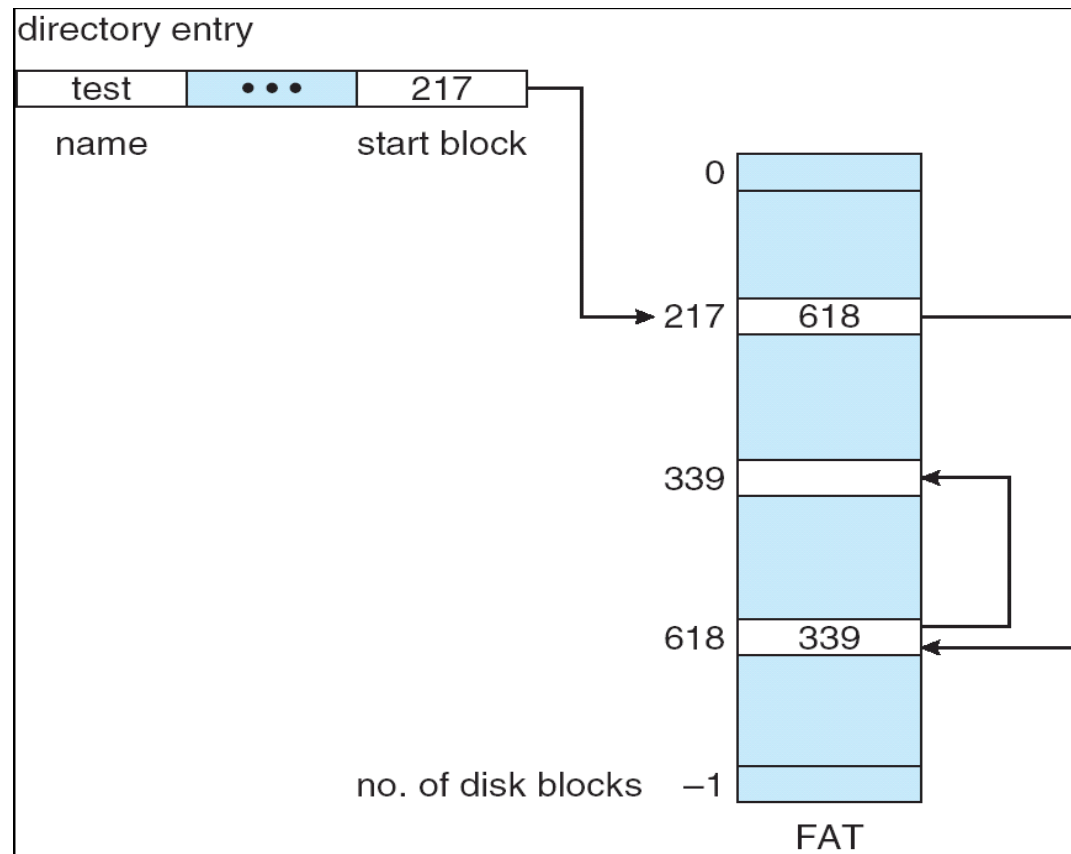
链接分配-文件分配表 (FAT)

- FAT是链接分配的变种，每个卷的开始部分用于存储文件分配表(File Allocation Table)
- 针对每个磁盘物理块都有一个FAT条目（未使用的块为0，使用的块包含下一个块儿号）



链接分配-文件分配表

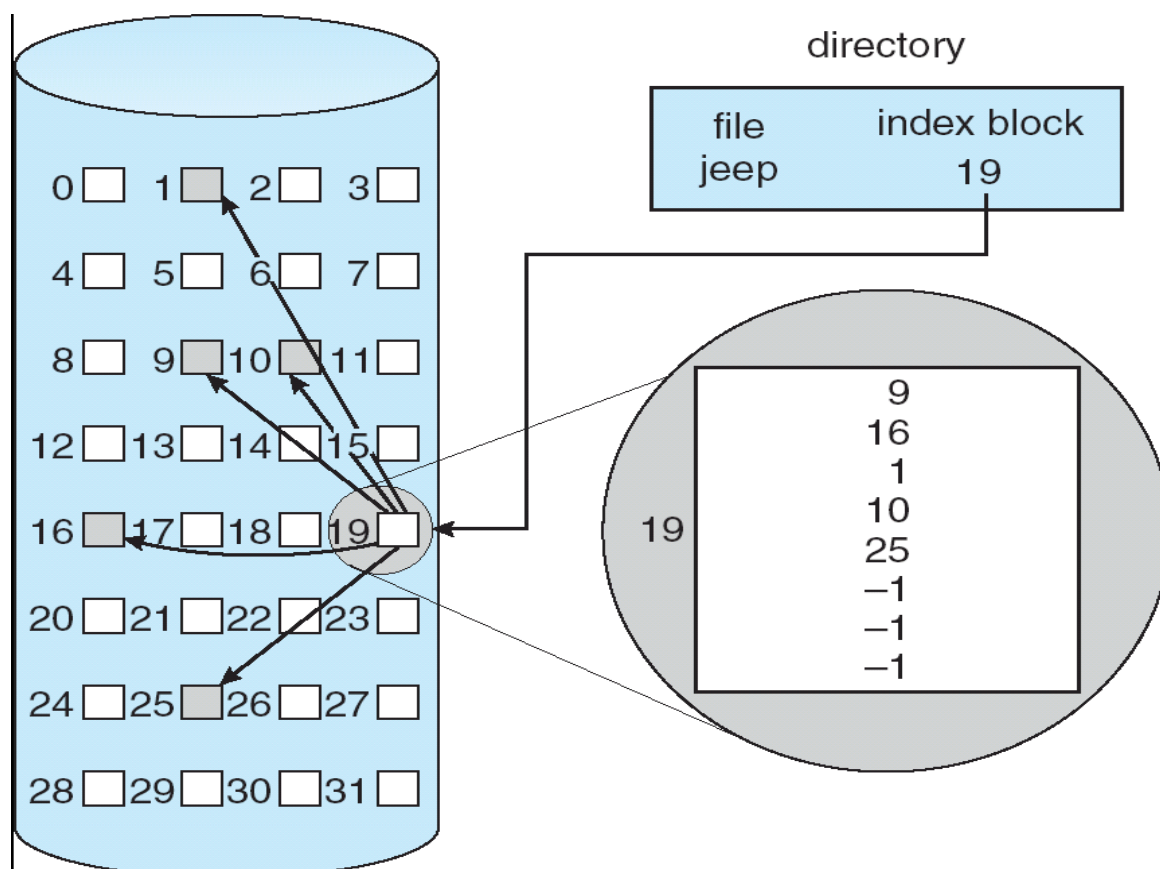
- 目录条目含有文件首块号码
- 块号索引的FAT条目包含文件下一块的号码



问题：
磁头移动

(3) 索引分配

文件用索引块来定义，每个文件有其索引块，索引块是一个磁盘块地址的数组。



索引分配

对每个文件必须分配一个完整的索引块



索引块的大小的如何决定？

如果大？ 如果小？

索引块指针的开销大。。

**如，一个文件索引块儿，能存储10个文件块儿，
需要20个文件块儿的文件，用2个文件索引块儿。**

索引分配

索引块的管理机制

1. 链接方案

- 为了处理大文件，可以将多个索引块链接起来

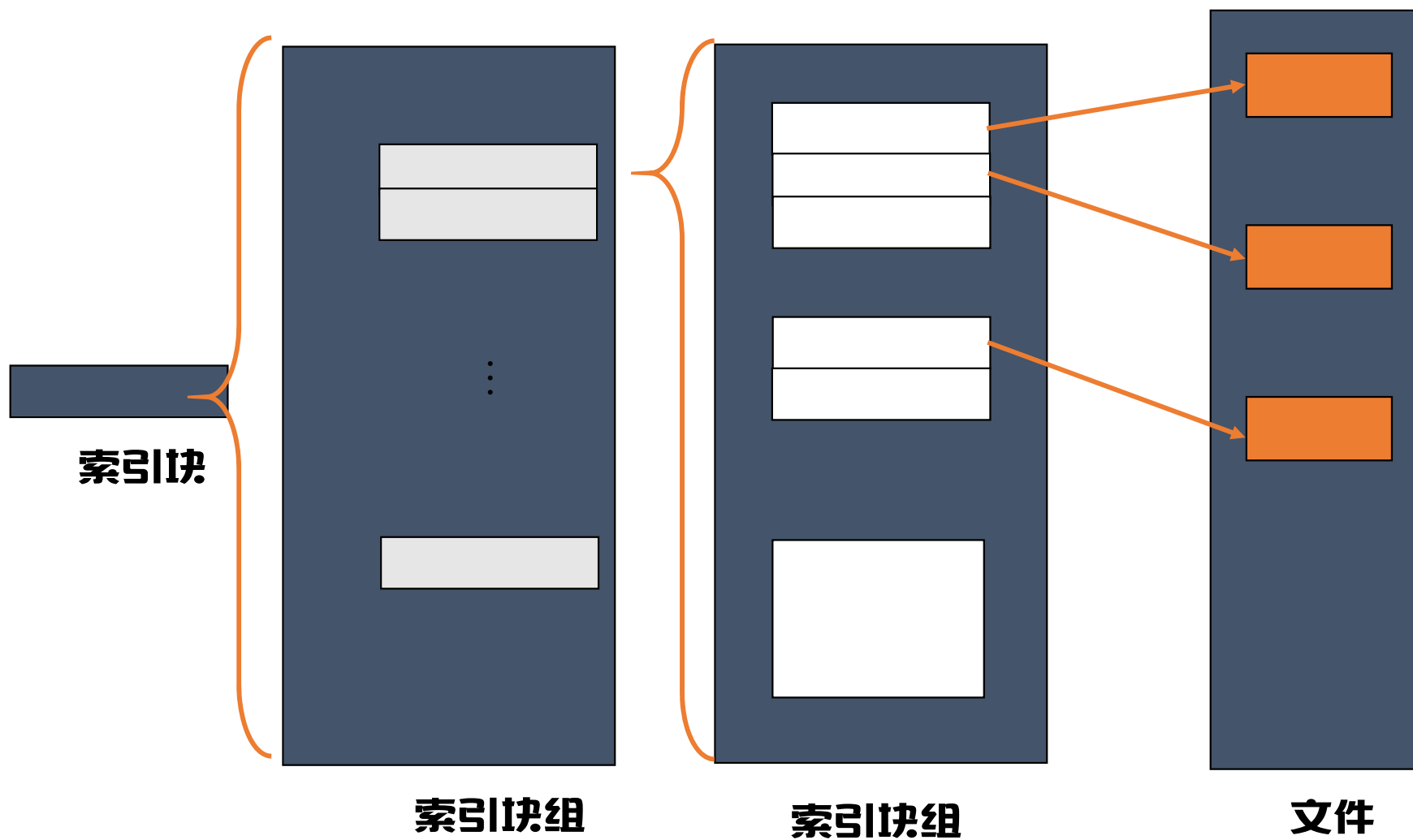
2. 多层次索引

- 用第一层索引块指向一组第二层的索引块，第二层索引块再指向文件块

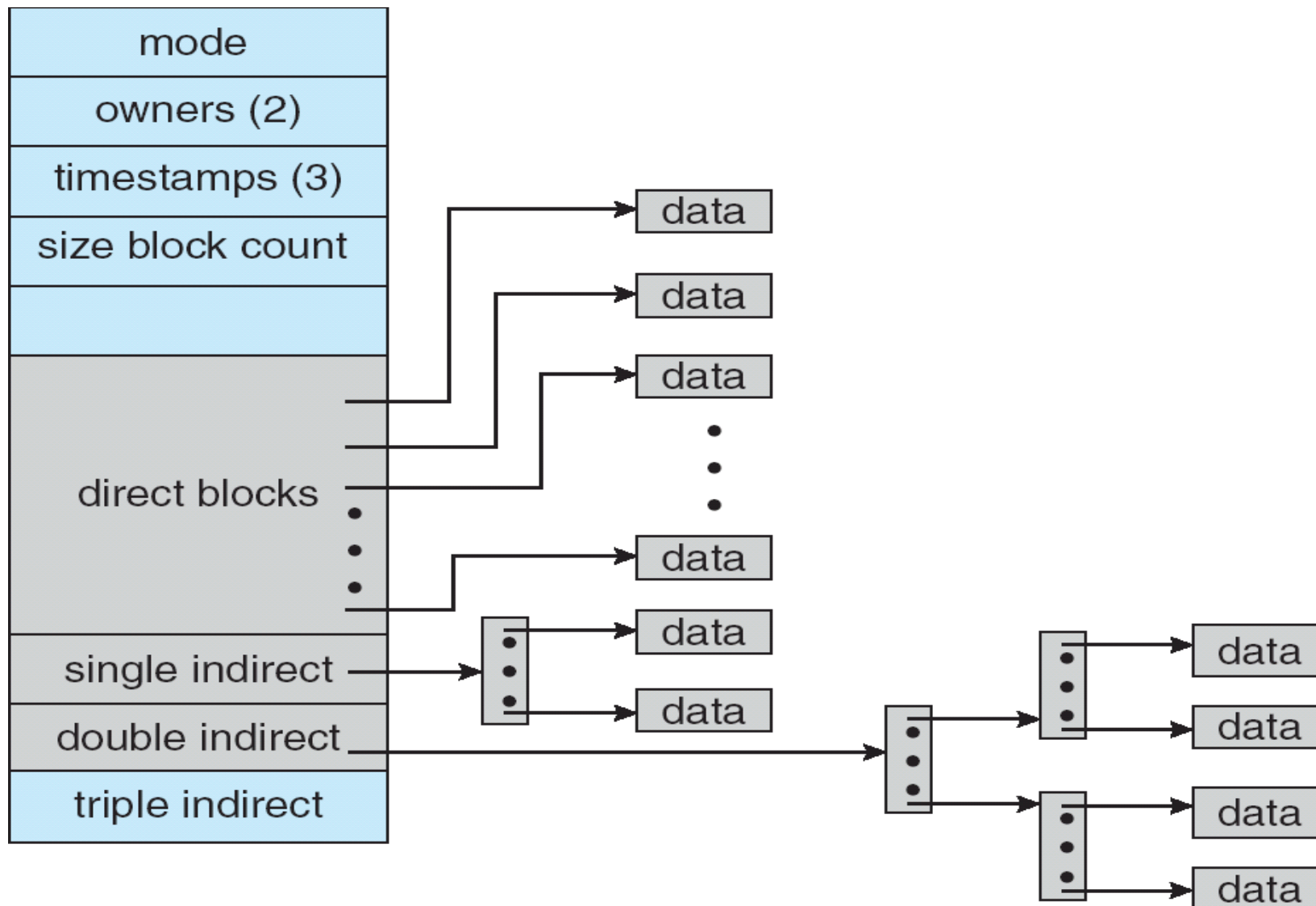
3. 组合方案

- 将索引块的前15个指针存储在文件的i-node中。其中，前12个指针指向直接块，剩下3个指针指向间接块
- 用于UFS (Unix File System)

多层次索引



组合方案



11.5 磁盘空闲空间的管理

实现方法有以下几种：



(1) 位向量 (bitmap)

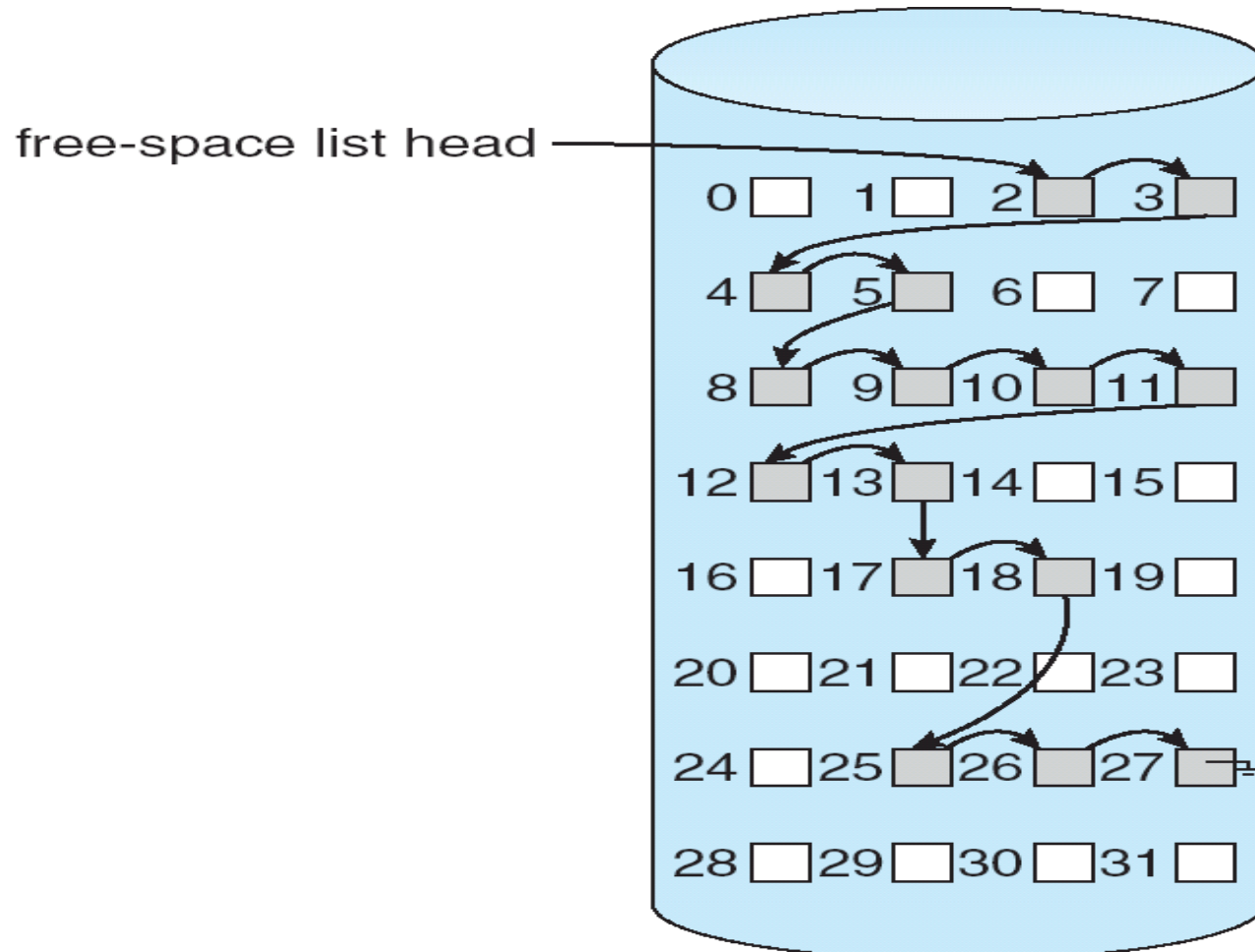
空闲空间表实现为**位图**, 或 **位向量**, 每块用一位 (bit) 表示



$$\text{bit}[k] = \begin{cases} 1 \Rightarrow \text{表示块儿空闲} \\ 0 \Rightarrow \text{表示块儿已分配} \end{cases}$$

(2) 链表管理空闲空间

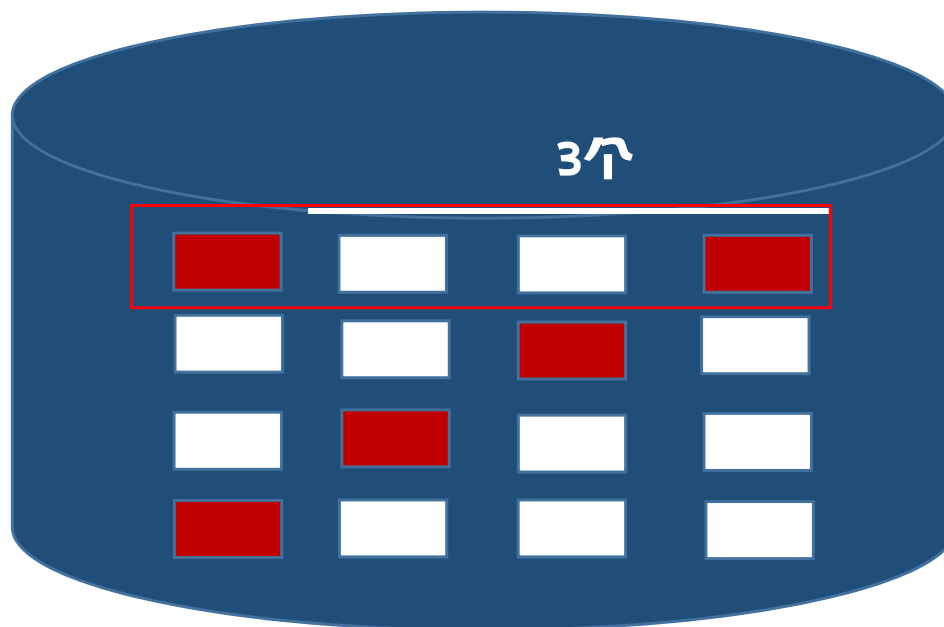
- 所有空闲块用链表链接起来，并将指向第一个空闲块儿的指针保存在特殊位置，同时缓存在内存
- 每个块儿含有下一个块儿的指针



(3) 组

将 n 个空闲块的地址保存在第一个空闲块中。

这些空闲块中的前 $n - 1$ 个为空，而最后一块包含另外 n 个空闲块的地址



(4) 计数

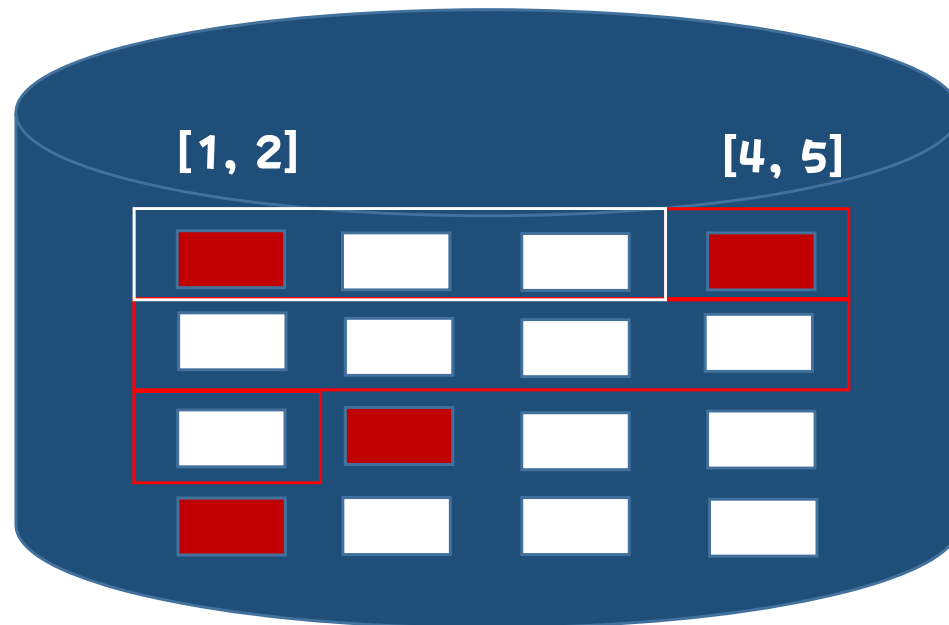
基于以下事实：

通常有多个连续块需要同时分配或释放，尤其是在使用连续分配时

- **记录第一块的地址和紧跟第一块的连续的空闲块的数量。**
- **空闲空间表的每个条目包括磁盘地址和数量。**

计数

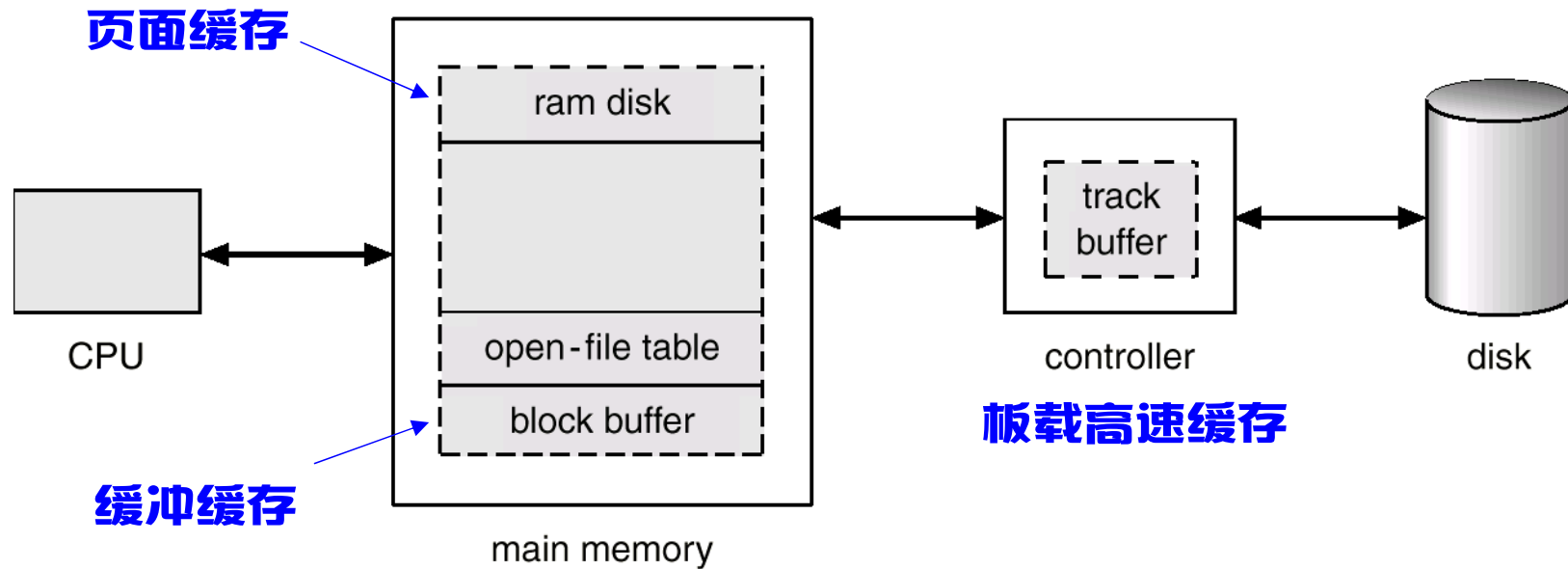
- 举例



11.6 效率和性能

- **磁盘空间的有效使用（效率）取决于**
 - **磁盘分配和目录管理算法**
 - **保留在文件目录条目中的数据类型**
- **改善性能 → 缓存**
 - **磁盘控制器的 板载高速缓存**
 - **缓冲缓存（buffer cache）**：一块独立内存，位于其中的块是马上需要使用的
 - **页面缓存**，将文件数据作为页而不是块来缓存。

不同的磁盘缓存位置



- 板载高速缓存，缓冲缓存，页面缓存 → 交换问题
- LRU（最近最少使用算法）是一个用于块或页置换的、合理且通用的算法。但是？

页面缓存

页面缓存使用虚拟内存技术，将文件数据作为页来缓存，比采用物理磁盘块来缓存更高效

文件的打开和访问

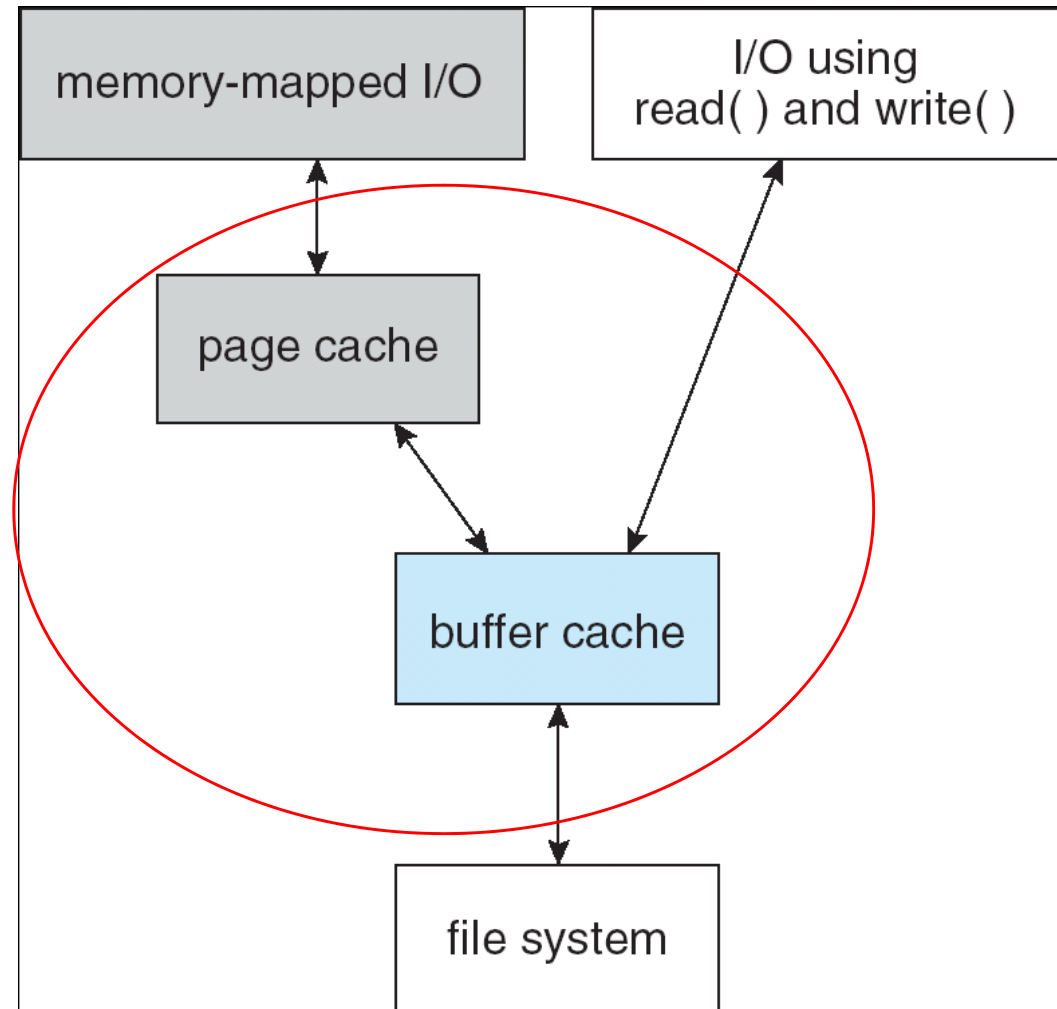
- 1. 内存映射 I/O 使用页缓存 (page cache)**
- 2. 标准 I/O 通过文件系统使用缓冲缓存(buffer cache)**

这种结果如下图所示.

缺少**统一缓冲缓存**的 I/O

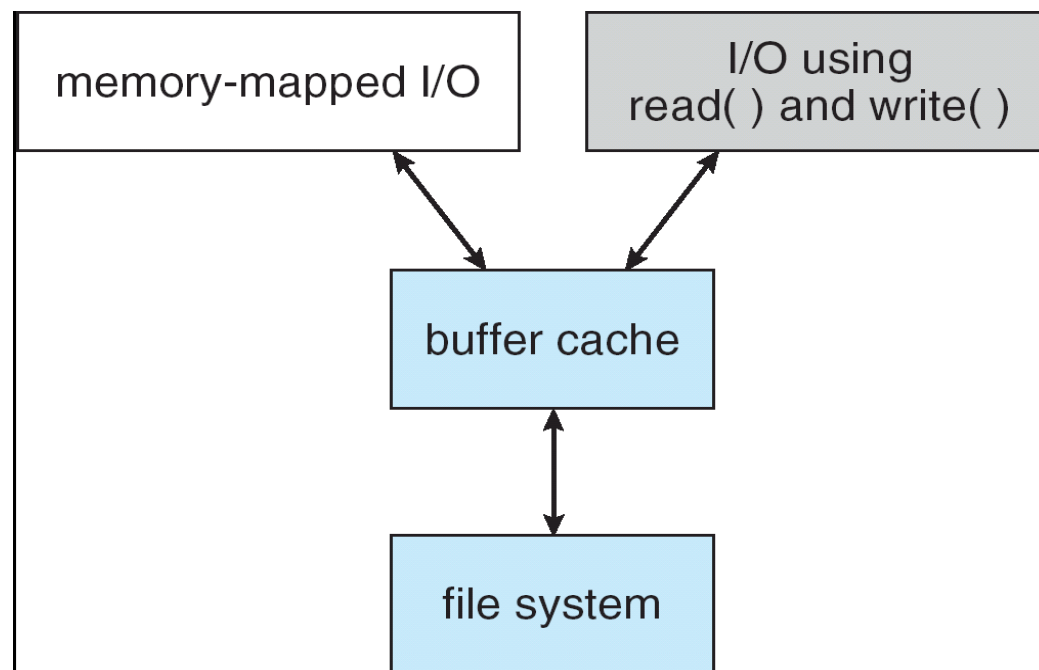
1. 内存映射先从文件系统中读入磁盘块, 并放入缓冲缓存
2. 缓冲缓存内的文件必须复制到页面缓存中

双重缓存



采用**统一缓冲缓存**的I/O

统一缓冲缓存：
统一使用页面缓存来缓
存进程页和文件数据



性能：页面置换

- 文件系统中的页面置换

- 特点：文件的读入或写出一般是按顺序进行。**所以，不适合采用LRU算法**

- 顺序访问可以通过**马上释放**和**预先读取**来加以优化

- I. 马上释放 (free-behind) : 请求下一页时，马上释放上一页
- II. 预先读取 (read-ahead) : 请求页之后的下一个页也一起读入

11.7 恢复

目录信息一般事先保存在内存中以加快访问，有时会导致目录结构中的数据和磁盘块中的数据不一致

1. 一致性检查：比较目录结构中的数据和磁盘块中的数据，尝试着去修正不一致

- **chkdsk (MS-DOS), fsck (unix)**

2. 备份&恢复：

**I. 备份 (backup) ： 利用系统程序来备份数据到其他
的存储设备，软盘，磁带**

**II. 恢复 (recovery) ： 通过从备份来恢复丢失的文件
或磁盘**

11.8 基于日志结构的文件系统

- **Log-based transaction-oriented, Journaling file system**
- 文件创建涉及到**目录结构修改**，**FCB分配**，**数据块分配** 等
- 元数据 (meta data) 的变化写入日志上- **提交事务**
- 提交了的事务，并不一定马上完成操作
 - I. 当实际完成时，就从日志中删除事务条目
- If the file system crashes, all remaining transactions in the log must still be performed



Q&A