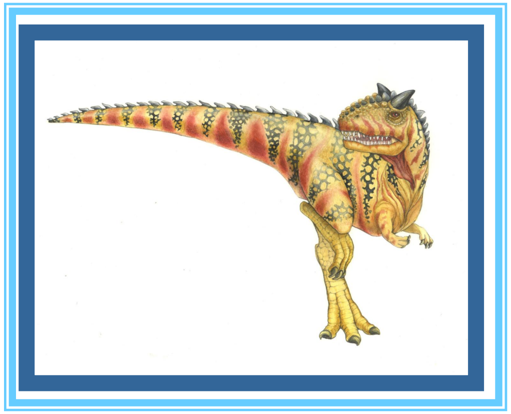
Chapter 11: File System Implementation



Chapter 11: File System Implementation

- File-System Structure
- File-System Implementation
- Directory Implementation
- Allocation Methods
- Free-Space Management
- Efficiency and Performance
- Recovery

Objectives

- To describe the details of implementing local file systems and directory structures
- To describe the implementation of remote file systems
- To discuss block allocation and free-block algorithms and trade-offs

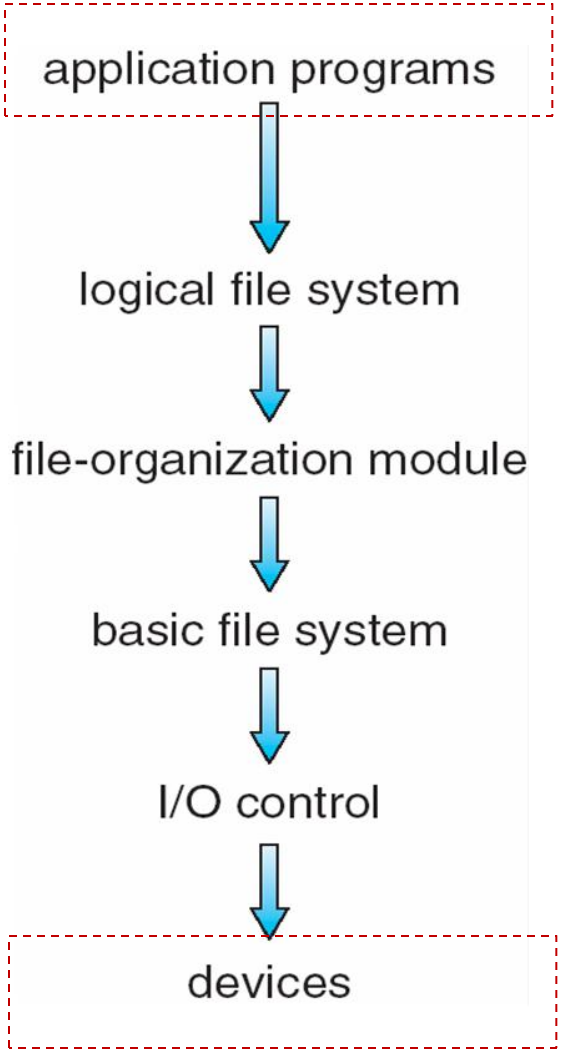
File-System Structure

- A file system includes:
  - File :Logical storage unit
    - Collection of related information
  - Directory
    - File control block – storage structure consisting of information about a file
  - Software

File-System Structure

- File system resides on secondary storage (disks)
  - Provides efficient and convenient access to disk by allowing data to be stored, located , retrieved easily
- File system organized into layers.
- Device driver controls the physical device
  - Transfer information between the main memory and the disk.

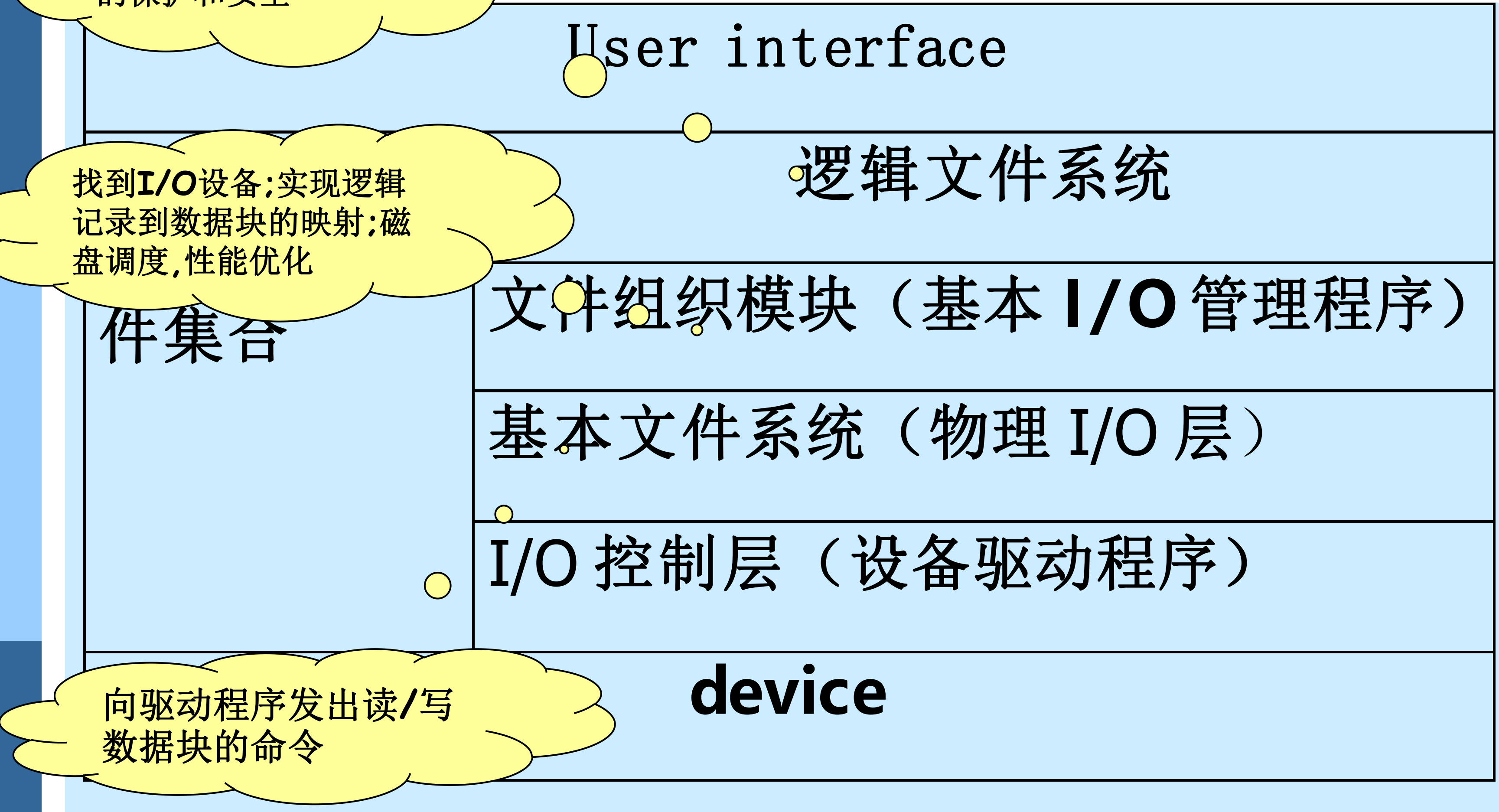
Layered File System



application programs	
logical file system	Provide information to file-organization module
file-organization module	Logical block address to physical block address
basic file system	Issue generic commands to the appropriate device driver to read and write physical blocks on the disk
I/O control	Consist of device drivers and interrupt handler
devices	

System Structure

管理文件目录,根据文件名得到该文件的相关信息,提供给下一层;文件的保护和安全



User interface

逻辑文件系统

件集合

文件组织模块（基本 I/O 管理程序）

基本文件系统（物理 I/O 层）

I/O 控制层（设备驱动程序）

device

找到I/O设备;实现逻辑记录到数据块的映射;磁盘调度,性能优化

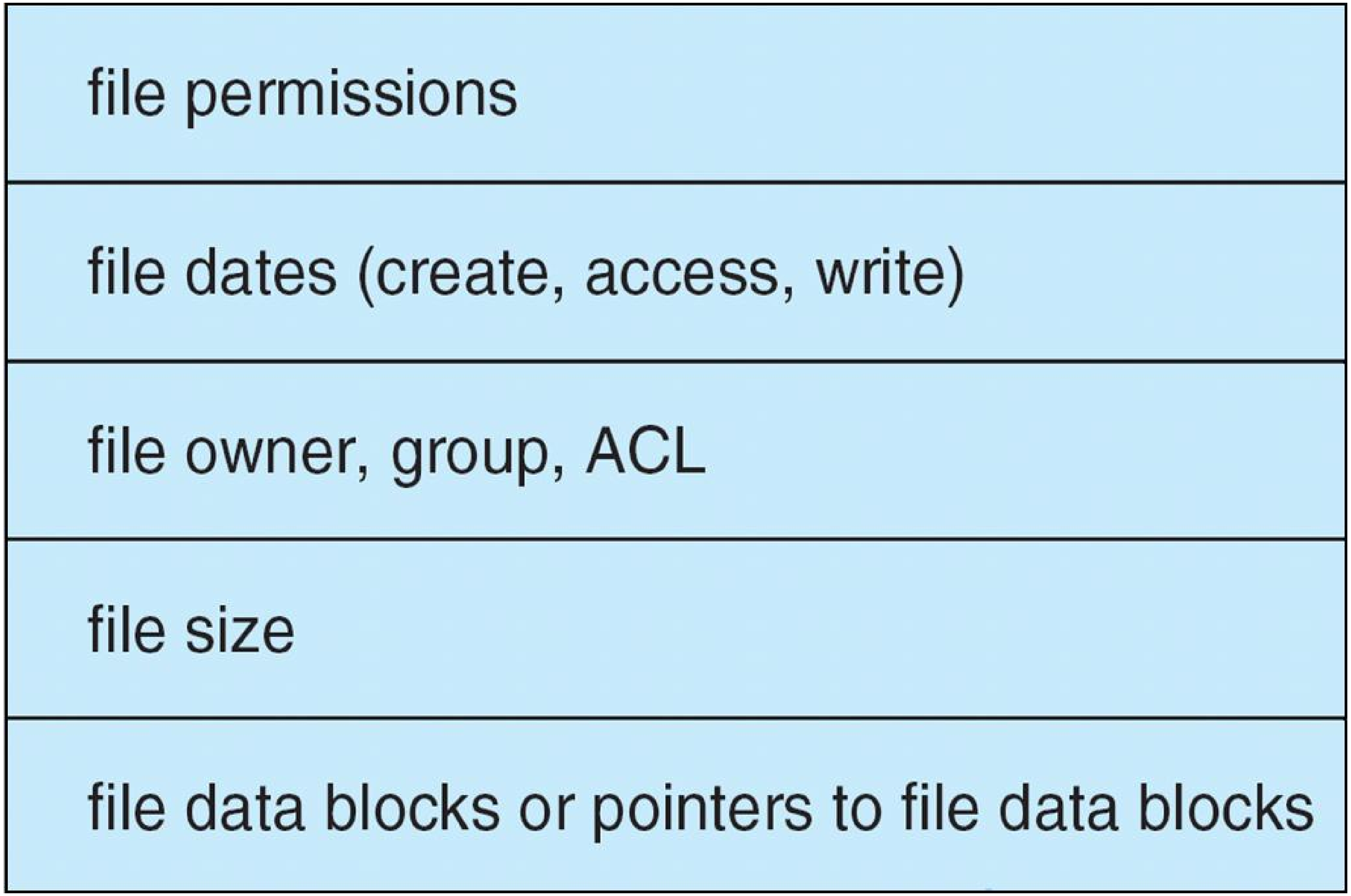
向驱动程序发出读/写数据块的命令

文件系统实现时需要哪些结构的支持？

File-System Implementation

- Several on-disk and in-memory structures are used to implement a file system
- On-disk structures includes:
  - Boot control block contains info needed by system to boot OS from that volume
  - Volume control block contains volume details
    - 分区块的数量、块大小、空闲块数量和指针、空闲的FCB数量和指针
    - UNIX FS: Super block（超级块）
    - NTFS: Master file table（主控文件表）
  - Directory structure organizes the files
  - Per-file File Control Block (FCB) contains many details about the file

A Typical File Control Block



file permissions

file dates (create, access, write)

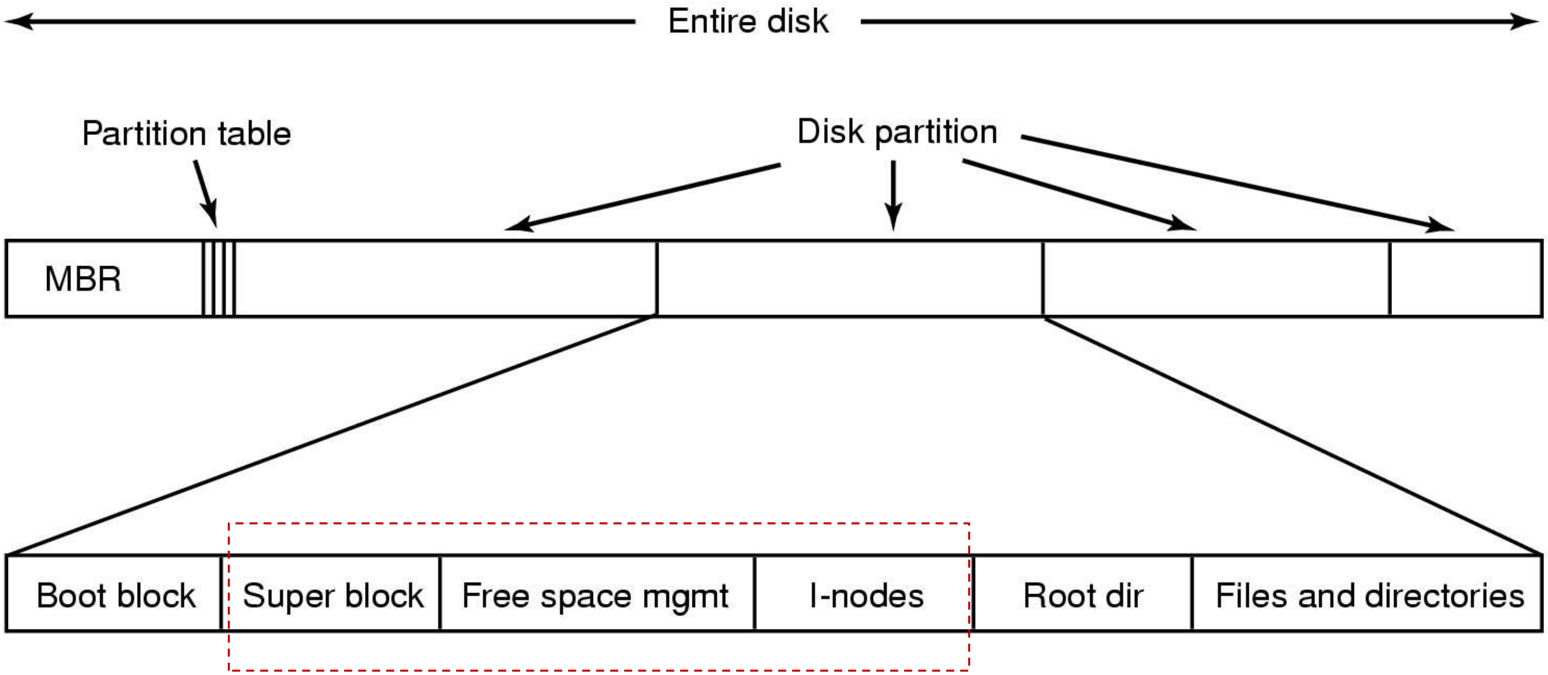
file owner, group, ACL

file size

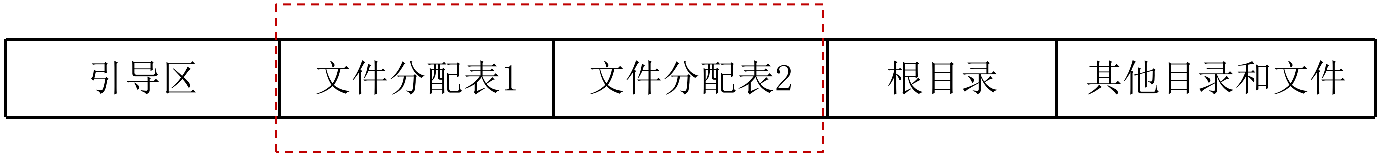
file data blocks or pointers to file data blocks

File System Layout

Unix file system layout



Window FAT file system layout



引导区	文件分配表1	文件分配表2	根目录	其他目录和文件

FAT（File Allocation Table）

File System Layout

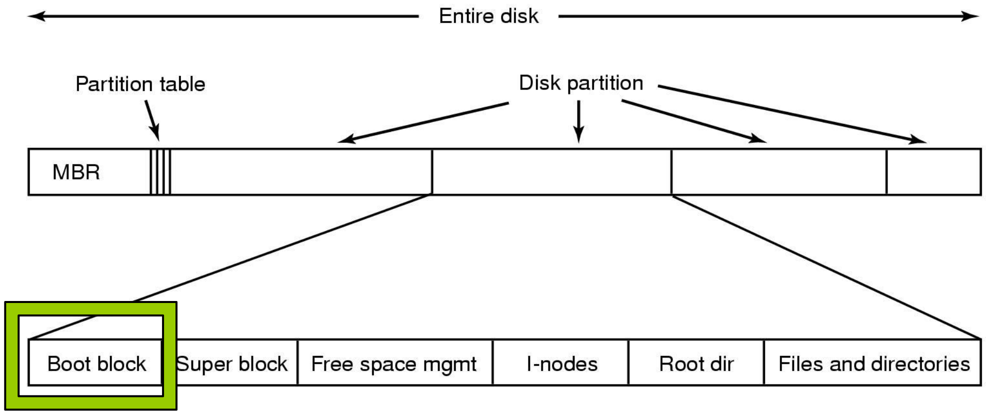
- Sector 0 of a disk is called MBR
  - Master Boot Record（主引导记录）
- MBR is used to boot the computer
- The end of MBR contains the partition table
  - Giving starting and ending addresses of each partition
  - One of the partitions is marked as active in the table

File System Layout

- At boot time, BIOS reads in the MBR
- MBR then locates the active partition
  - and reads in the first block, the boot block （引导块）, and execute it
- Boot block in turns loads the OS in the partition

File System Layout

- boot block后面的内容就因系统而异
- boot block后面是一个超级块(分区控制块)
  - 存放该文件系统的各种参数:文件系统类型,数据块尺寸,空闲块的数量和指针,空闲FCB的数量和指针等等
- 超级块后面是磁盘自由空间
- 再后面是I-NODE区(UNIX),FAT文件系统无此区
- 再后面是根目录区
- 最后是其他目录和用户文件区



In-Memory File System Structures

- The in-memory structures may include the ones as below:
  - An mount-table: contains info about each mounted volume
  - An directory-structure cache: holds the directory info of recent accessed directories
  - The system-wide open-file table
  - The per-process open-file table

- System-wide open-file table
  - 整个系统一张
  - 放在内存：用于保存已打开文件的FCB

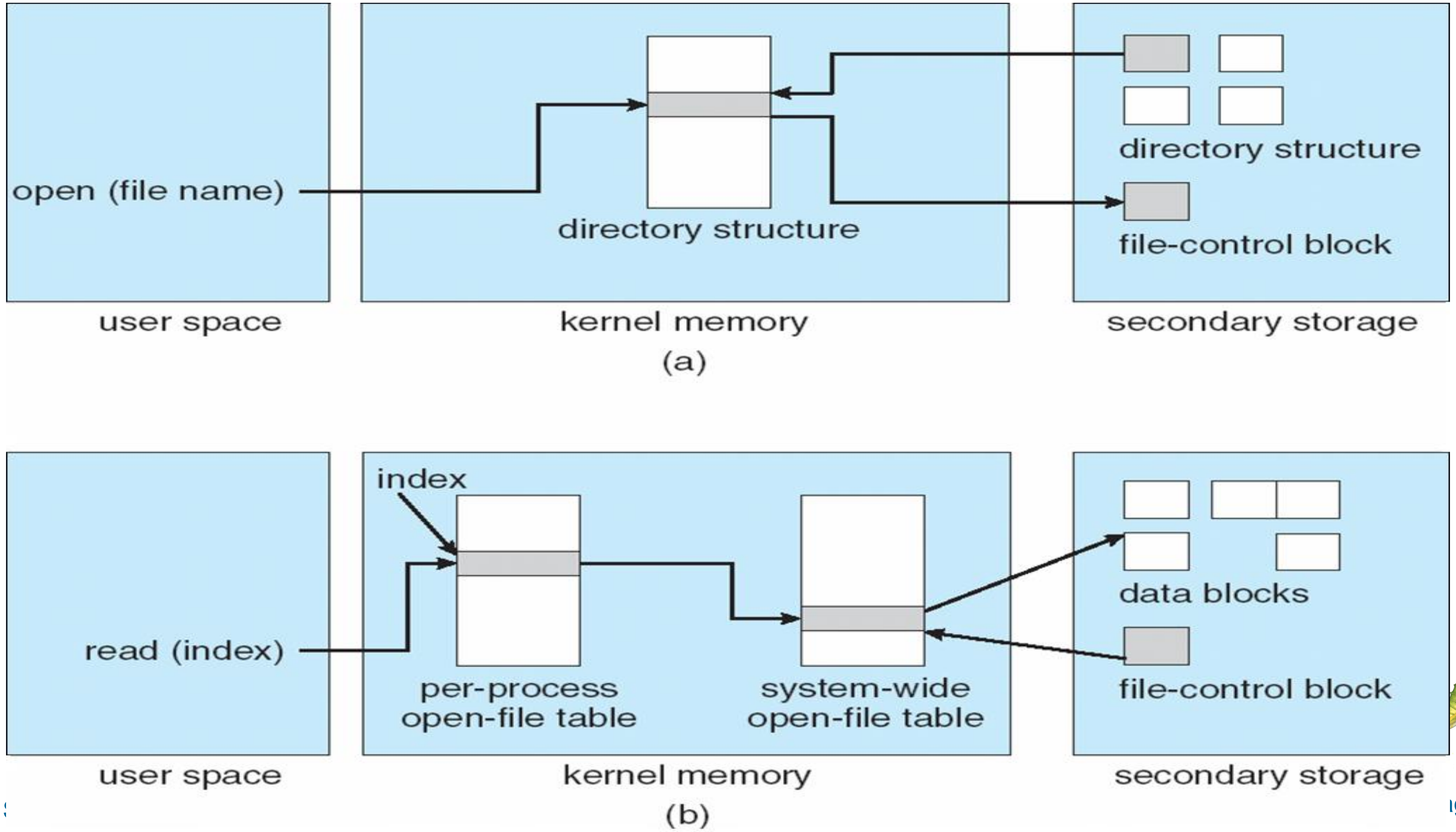
FCB(iNode)信息	引用次数	修改标记

- Per-process open-file table
  - 每个进程一张
  - 进程的FCB中记录了用户打开文件表的位置

文件描述符	打开方式	读写指针	系统打开文件表索引

In-Memory File System Structures

- The following figure illustrates the necessary file system structures provided by the operating systems.
  - Figure (a) refers to opening a file.
  - Figure (b) refers to reading a file.



(a)

(b)

Virtual File Systems

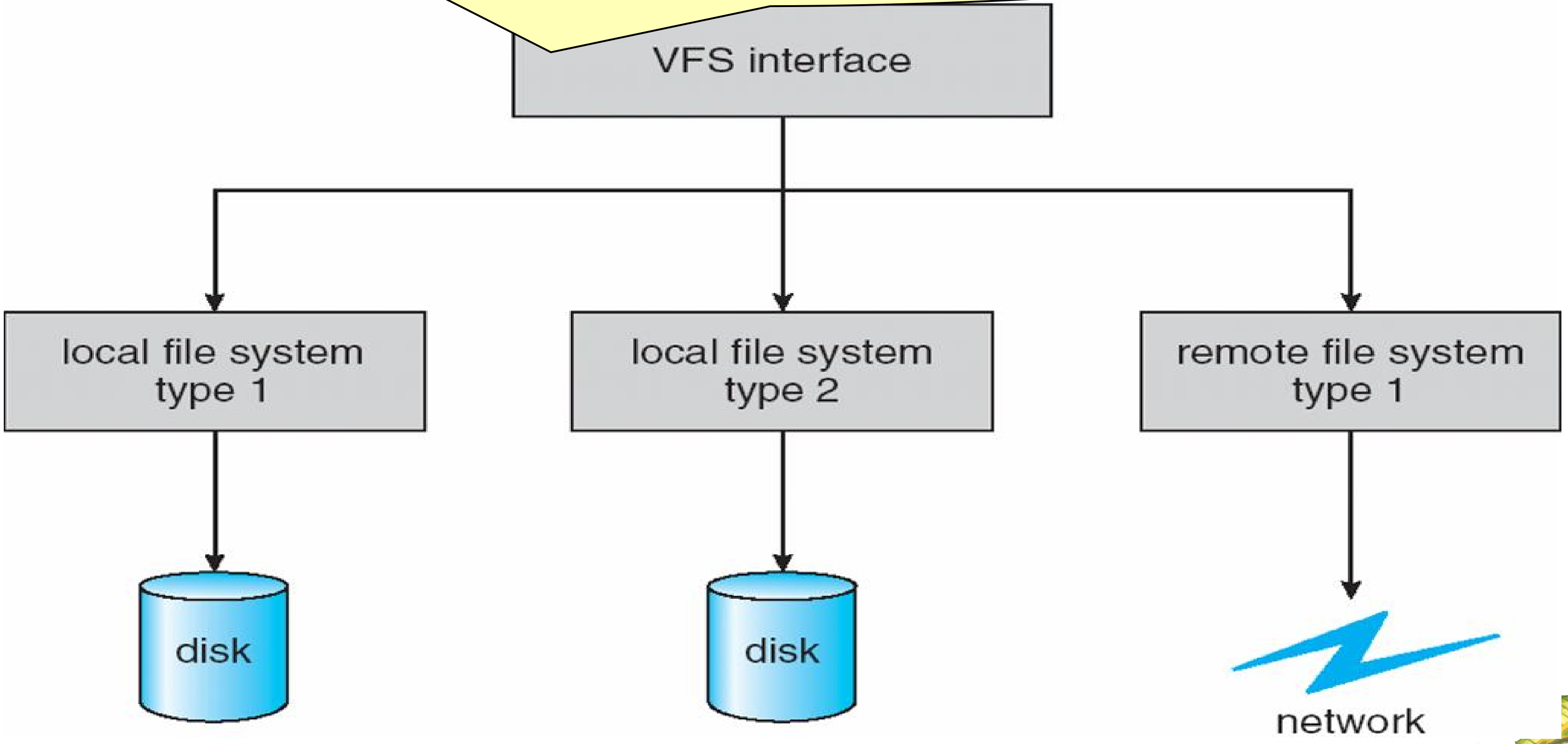
- Modern OS must support multiple type of file systems.
  - How to allow multiple types of file system to be integrated into a directory structure?
  - How to support users' seamless move between different file systems?
- Most OS, including UNIX, use object-oriented techniques to simplify, organize and modularize the implementation.

Virtual File Systems

- Virtual File Systems (VFS) provide an object-oriented way of implementing file systems.
- VFS allows the same system call interface (the API) to be used for different types of file systems.
- The API is to the VFS interface, rather than any specific type of file system.

Including open(), read(), write(), and close() calls and file descriptors.

Separating standard file-system Operations from their implementation ; providing a mechanism for uniquely representing a file throughout a network based on **vnode**



Directory Implementation

- **The selection of directory-allocation and directory-management algorithms significantly affects the efficiency, performance and reliability of the file system.**
- **Linear list** (线性列表) of file names with pointer to the data blocks.
  - simple to program
  - time-consuming to search
- **Hash Table** – linear list with hash data structure.
  - decreases directory search time
  - **collisions** – situations where two file names hash to the same location
  - fixed size

FCB

Name	**Location**	**Size**	**…**

文件目录改进

如何加快目录检索?

- 为加快目录检索可采用目录项分解法：把FCB分成两部分：
  - 符号目录项：
    - 文件名，文件号(iNode号)
  - 基本目录项（索引节点目录iNode）：
    - 除文件名外的所有字段

UNIX的目录结构

- UNIX采用文件名和文件说明分离的目录结构
  - 每个文件有一个存放在磁盘索引节点区的索引节点，称为磁盘索引节点，它包括以下内容：
    - 文件主标识符和同组用户标识符；
    - 文件类型：是普通文件、目录文件、符号连接文件或特别文件（又分**块设备**文件或字符**设备**文件）
    - 文件主人，同组用户和其它人对文件存取权限（读**R**、写**W**、执行**X**）
    - 文件的物理地址，用于UNIX直接、间接混合寻址的13个地址项di_addr[13]
    - 文件长度（字节数）di_size
    - 文件链接数di_nlink
    - 文件最近存取和修改时间等。

- 给定文件路径名为/usr/ast/mbox，检索过程如下

根目录

1	.
1	..
4	bin
7	dev
14	lib
9	etc
6	usr
8	tmp

查找根目录得到usr的iNode 6

iNode 6

132

记录了usr在132中

块132:
Usr目录文件

6	.
1	..
19	disk
30	erik
51	jim
26	ast
45	bat

usr/ast的iNode 26

iNode 26

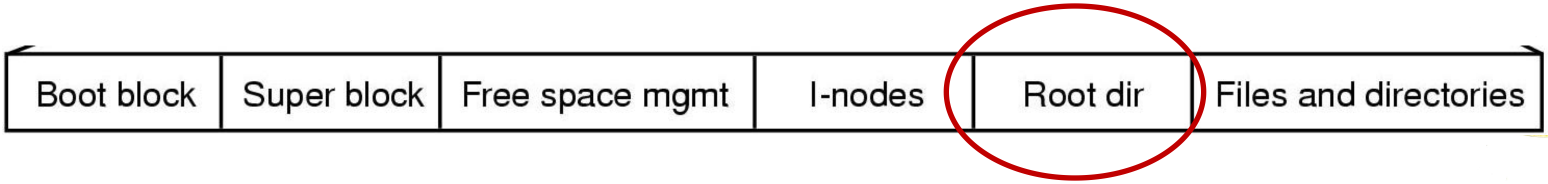
406

记录了/usr/ast在406中

块406:
usr/ast目录文件

26	.
6	..
64	grants
92	books
60	mbox
81	minix
17	src

usr/ast/mbox的iNode 60



习题

- 假设 一个FCB 占64个字节，物理块大小512字节。

符号目录项占10字节（文件名8字节，文件号2字节）。

基本目录项（iNode）占64-8=56字节

若一个目录文件有254个目录项，请分别给出分解前后查找文件的某一个FCB 的平均磁盘访问次数。

解答：

分解前： 占254*64/512=32块

分解后：符号文件占254*10/512=5块

查找一个文件的FCB的平均访盘次数：

分解前：16.5次 =（1+2+3+…+32）/32

分解后：4次 = (2+3+4+5+6)/5

文件的物理结构

- 文件的物理结构
  - 文件在存储介质上的存放方式
  - 主要解决两个问题：
  - 假设一个文件被划分为N块，这N块在磁盘上是怎么存放的?
  - 其地址（块号）在FCB中是怎么记录的？

  - 连续结构
  - 链接结构
  - 索引结构

Allocation Methods

- An allocation method refers to how disk blocks are allocated for files:

- Contiguous allocation

- Linked allocation

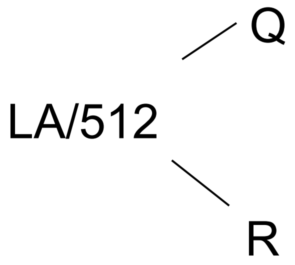
- Indexed allocation

Contiguous Allocation

- Each file occupies a set of contiguous blocks on the disk
- Simple – only starting location (block #) and length (number of blocks) are required
- Support sequential access and direct access
- Wasteful of space (dynamic storage-allocation problem) 碎片？ 外碎片
- Files cannot grow

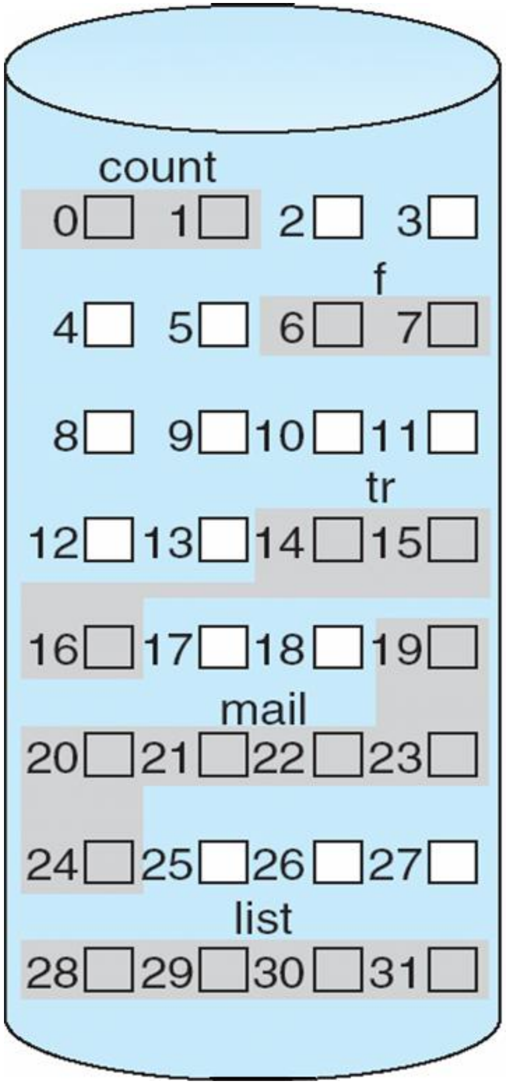
Contiguous Allocation

- Mapping from logical to physical



Block to be accessed = Q + starting address
Displacement into block = R

Contiguous Allocation of Disk Space



directory

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

FCB中如何记录文件地址？

第一块的块号 + 长度

Contiguous Allocation

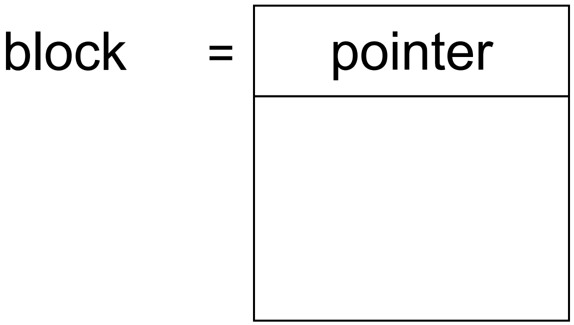
- 优点
  - 简单
  - 支持顺序存储和随机存取
  - 所需的磁盘寻道次数和寻道时间最少
  - 可以同时读入多个块，检索一个块也很容易
- 缺点
  - 文件不能动态增长
    - 预留空间：浪费　　或　重新分配和移动
  - 不利于文件插入和删除
  - 外部碎片：紧缩技术

Extent-Based Systems

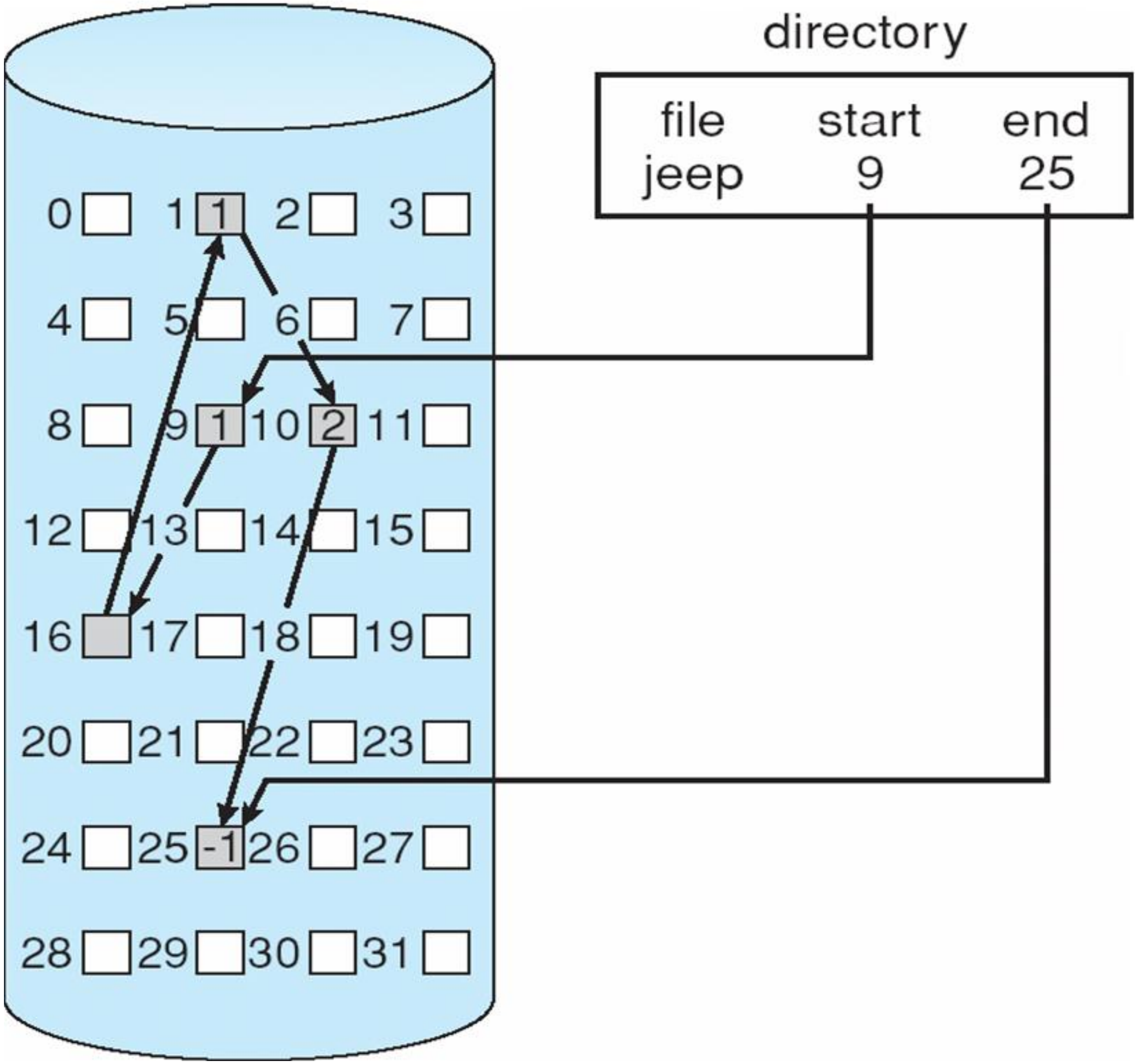
- Many newer file systems (I.e. Veritas File System) use a modified contiguous allocation scheme
- Extent-based file systems allocate disk blocks in extents
  - Initially, a contiguous chunk of space is allocated to the file
  - Then, if the amount proves to be not enough, another chunk of contiguous space—an extent—is added
  - The location of the file is recorded as a location and a block count, plus a link to the first block of the extent
- An extent is a contiguous block of disks
  - Extents are allocated for file allocation
  - A file consists of one or more extents

Linked Allocation

- Each file is a linked list of disk blocks: blocks may be scattered anywhere on the disk.
- The directory contains a pointer to the first and last blocks of the file.



Linked Allocation



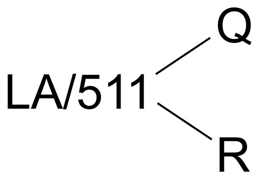
file	start	end
jeep	9	25

FCB中如何记录文件地址？

记录起始块号

Linked Allocation (Cont.)

- Simple – need only starting address
- Free-space management system – no waste of space
- No random access
- Mapping



Block to be accessed is the Qth block in the linked chain of blocks representing the file.
Displacement into block = R + 1

Linked Allocation

- 优点
  - 提高了磁盘空间利用率，不存在外碎片问题
  - 有利于文件插入和删除
  - 有利于文件动态扩充
- 缺点
  - 存取速度慢，不适于随机存取
  - 可靠性问题，如指针出错
  - 更多的寻道次数和寻道时间
  - 链接指针占用一定的空间

File-Allocation Table

链接结构的一个变形

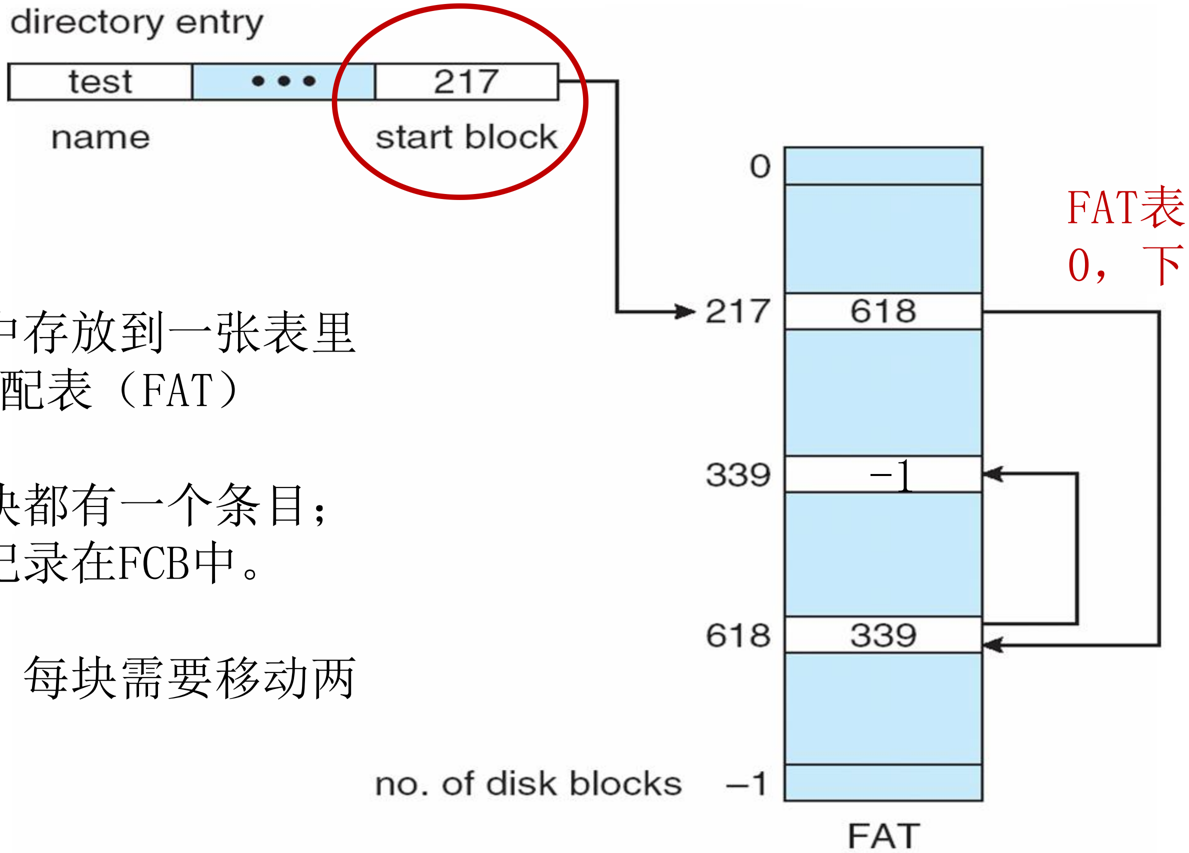
File-allocation table (FAT)：

disk-space allocation used by MS-DOS and OS/2.

思想：
将指针集中存放到一张表里
---文件分配表（FAT）

每个磁盘块都有一个条目；
起始块号记录在FCB中。

最坏情况，每块需要移动两次磁头。



FAT表项的值有3种：
0，下一块块号，-1

0表示空闲物理块

引导区	文件分配表1	文件分配表2	根目录	其他目录和文件

Indexed Allocation

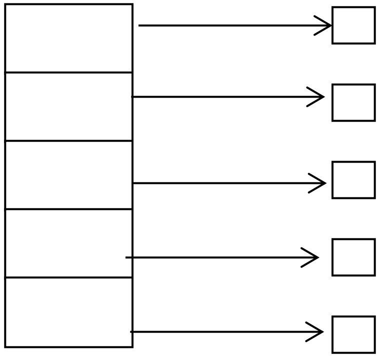
- Brings all pointers together into the index block
- Each file has its own index block, which is an array of disk-block addresses.

  索引表就是磁盘块地址数组，其中第i个条目指向文件的第i块

- The directory contains the address of the index block
- Logical view

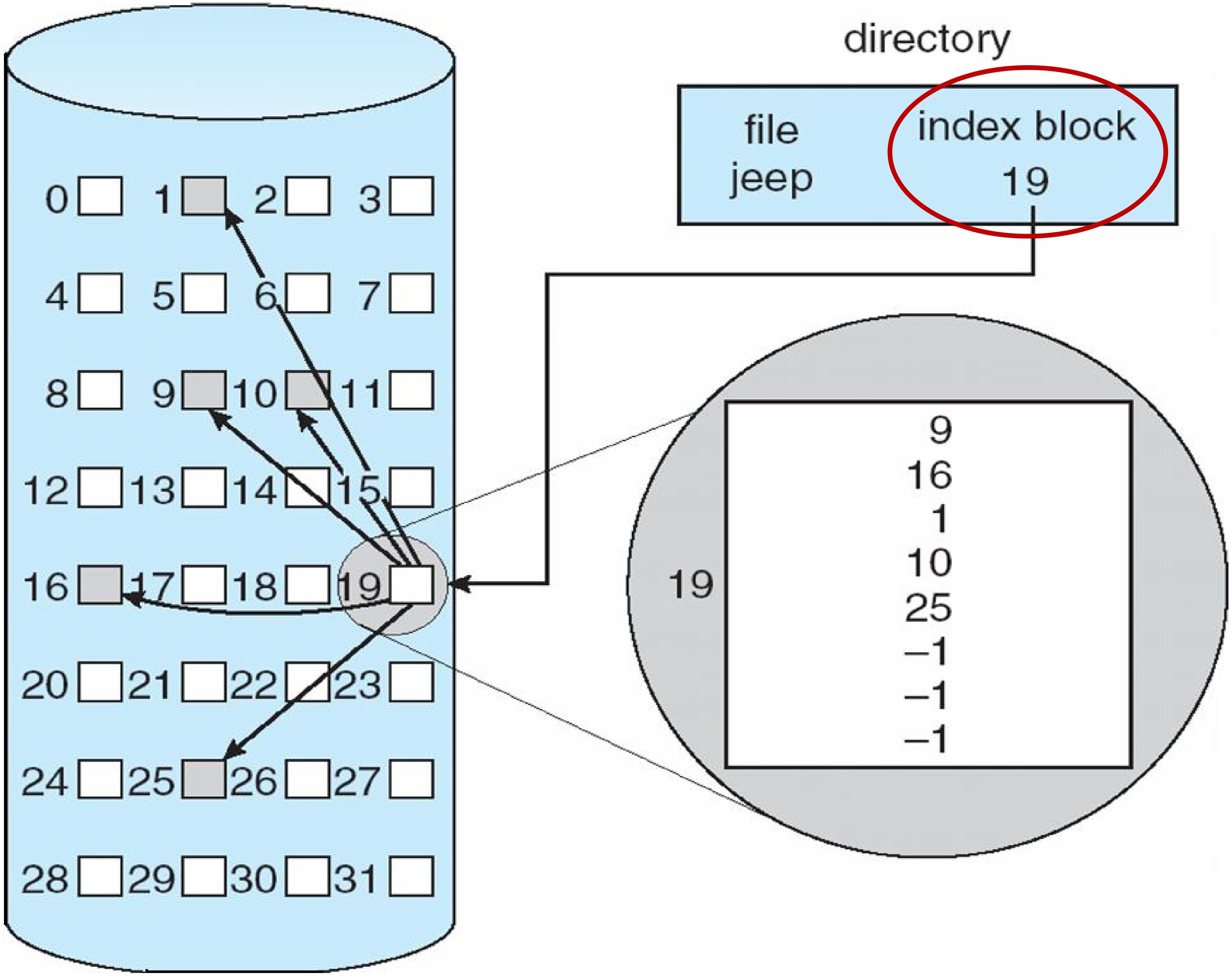
FCB中如何记录文件地址？

索引表存放在何处？



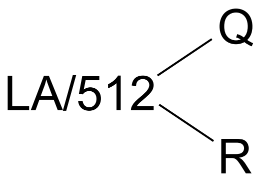
index table

Example of Indexed Allocation



Indexed Allocation (Cont.)

- Need index table
- Random access
- No external fragmentation, but have overhead of index block
- Mapping



Q = displacement into index table
R = displacement into block

Indexed Allocation

- 优点

  保持链接结构的优点，又克服了其缺点:

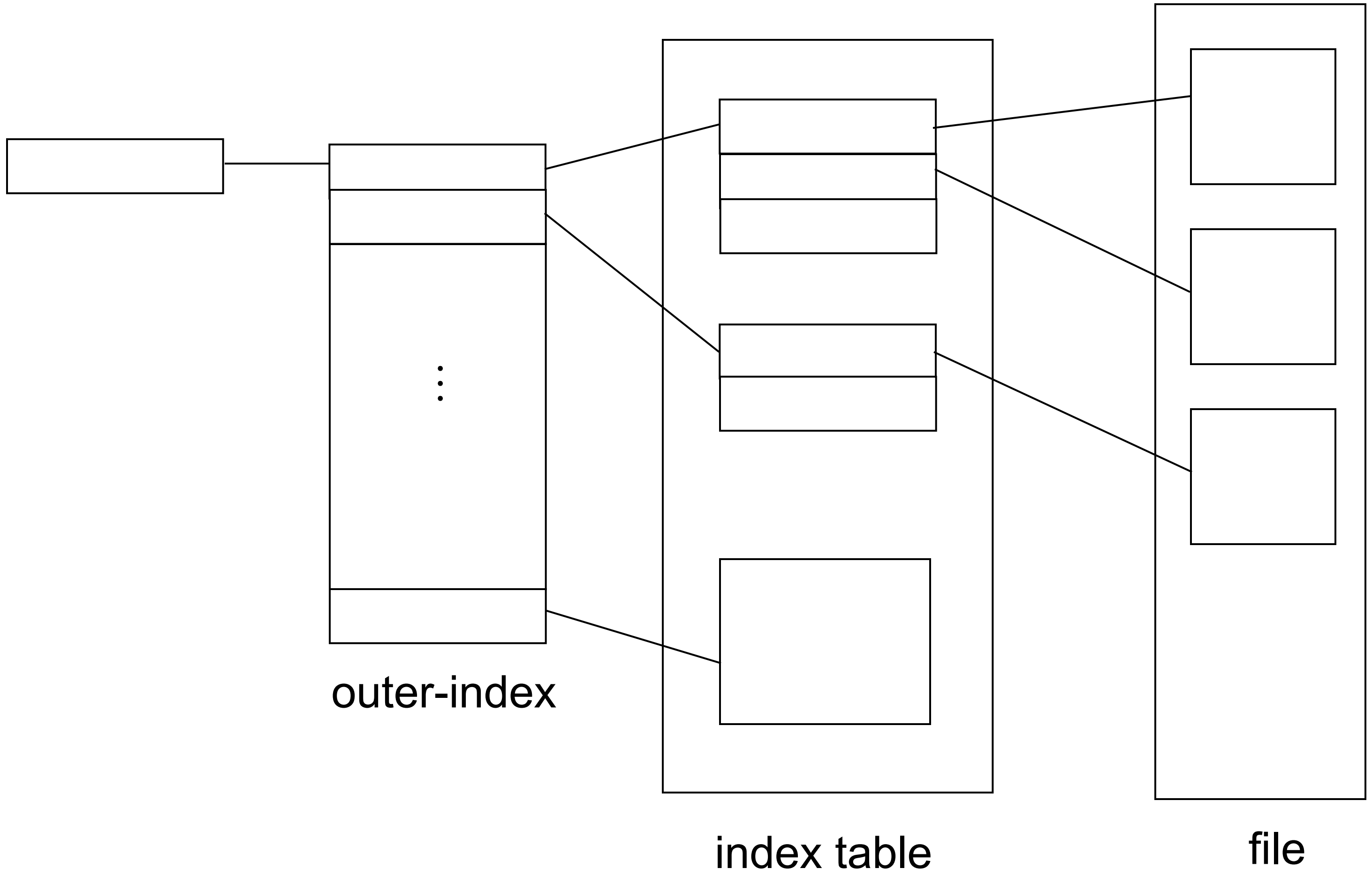
  - 既能顺序存取，又能随机存取
  - 满足了文件动态增长、插入删除要求
  - 能充分利用外存空间

- 缺点
  - 较多的寻道次数和寻道时间
  - 索引表本身带来了系统开销, 如：内外存空间，存取时间

Indexed Allocation (Cont.)

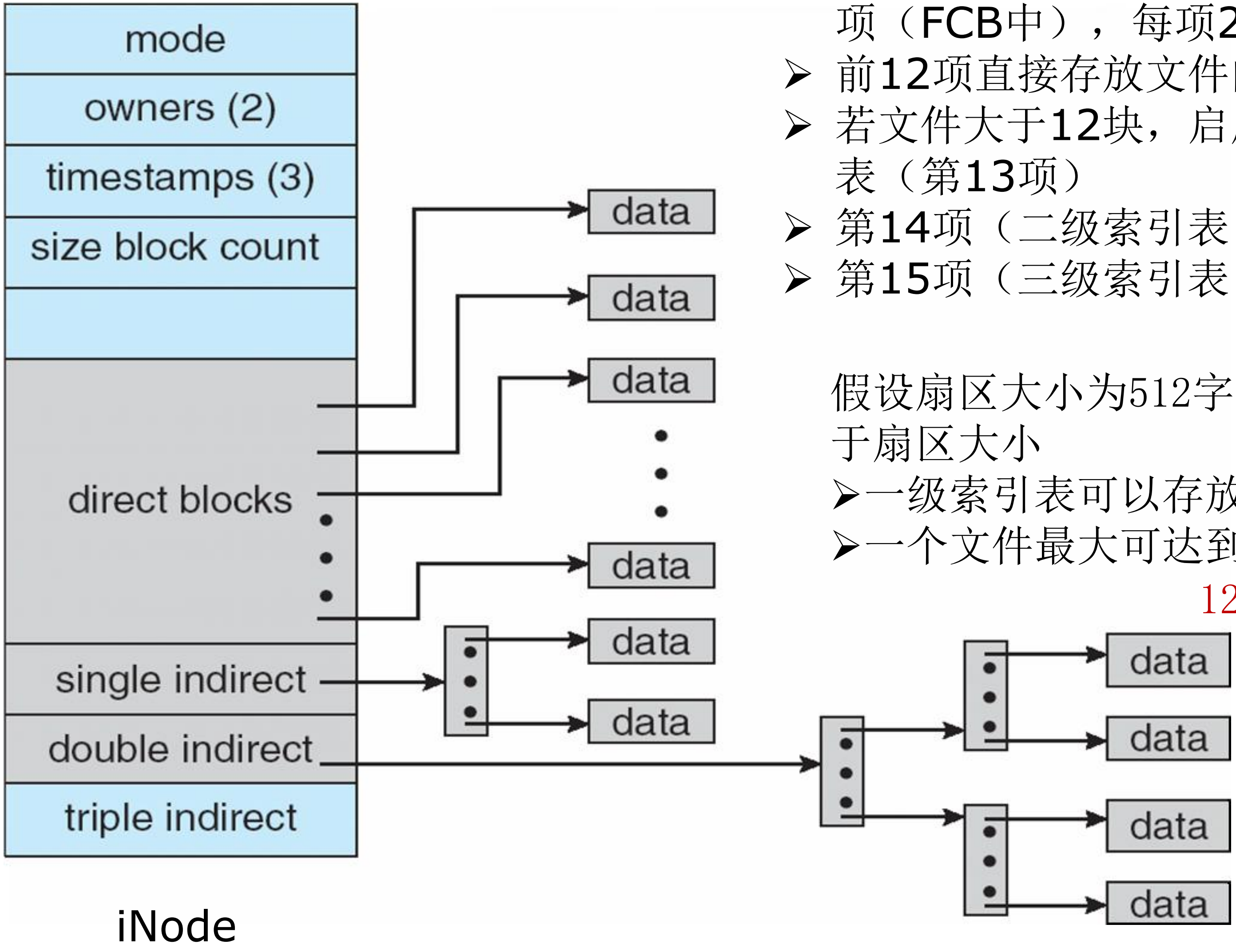
- A large file may have several index blocks, how to organize them? (block size of 512 words)
  - Linked scheme – Link blocks of index table (no limit on size)
  - Two-level index (maximum file size is 512^3)

Indexed Allocation – Mapping (Cont.)



Combined Scheme: UNIX UFS (4K bytes per block)

UNIX三级索引结构



- 每个文件的主索引表有15个索引项（FCB中），每项2个字节
- 前12项直接存放文件的物理块号
- 若文件大于12块，启用一级索引表（第13项）
- 第14项（二级索引表）
- 第15项（三级索引表）

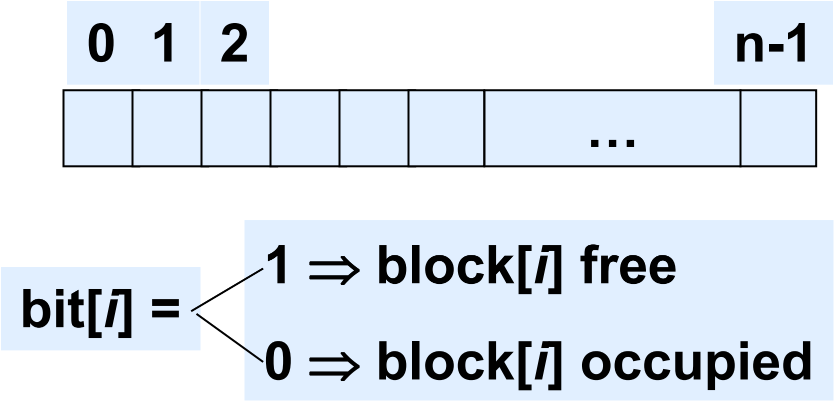
假设扇区大小为512字节，物理块等于扇区大小

- 一级索引表可以存放?个物理块号 256
- 一个文件最大可达到?个物理块

$12+256+256^2+256^3$

Free-Space Management

- Bit vector (*n* blocks)



bit[*i*] = 1 ⇒ block[*i*] free

bit[*i*] = 0 ⇒ block[*i*] occupied

- Advantage
  - Simple
  - Efficient in finding the first free block or n consecutive free blocks
- How to find the first free block
  - The first non-0 word is scanned for the first 1-bit, which is the location of the first free block.

(number of bits per word) * (number of 0-value words) + offset of first 1-bit

第一个空闲块号码（按字扫描）

Free-Space Management (Cont.)

- Bit map requires extra space
  - Example:

    block size = 2^{12} bytes

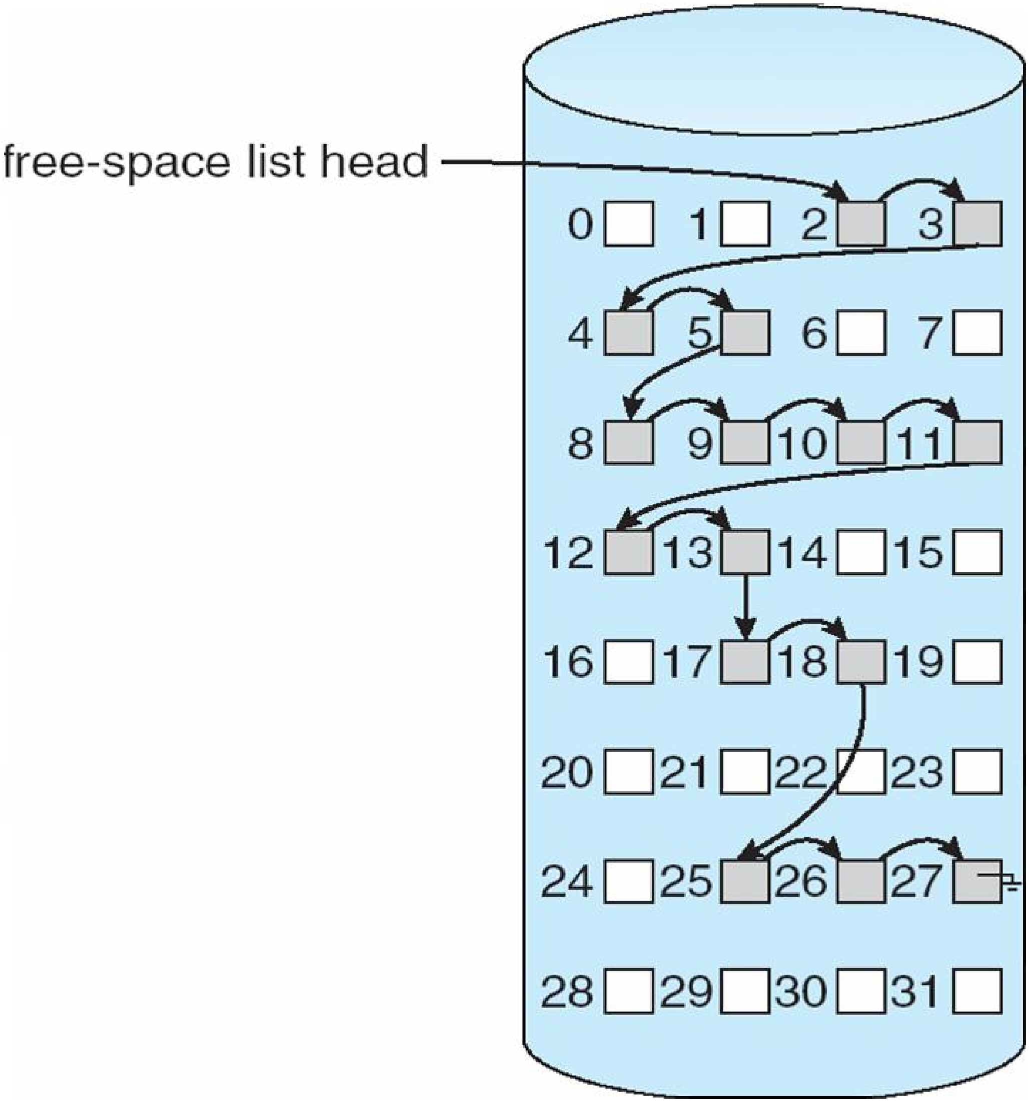
    disk size = 2^{30} bytes (1 gigabyte)

    $n = 2^{30}/2^{12} = 2^{18}$ bits (or 32K bytes)

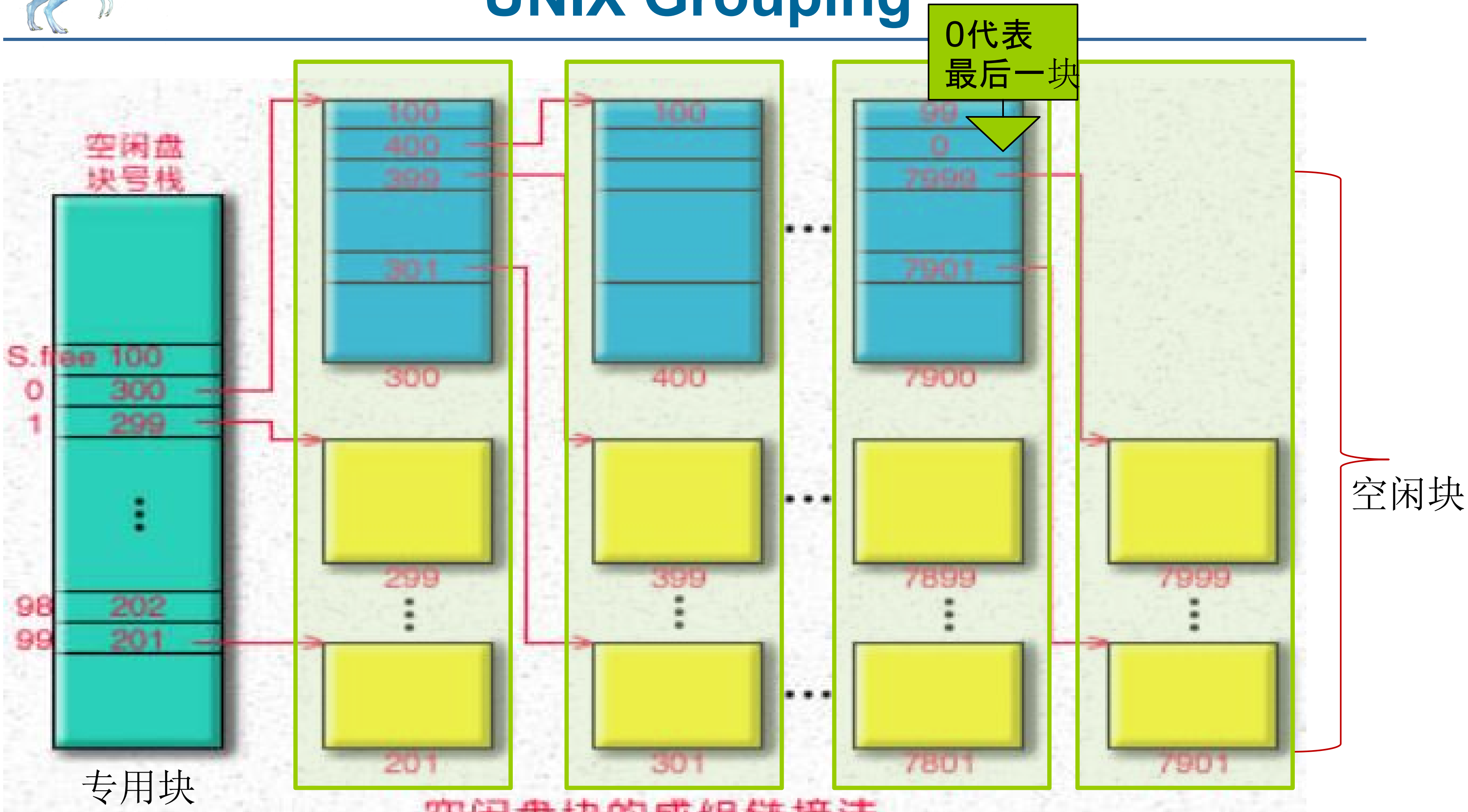
Free-Space Management (Cont.)

- Linked list (free list) 空闲块链表
  - Cannot get contiguous space easily
  - No waste of space
- Grouping 成组链接法
  - 在第一个空闲块中存储n个空闲块的地址
  - 前n-1为空，第n个指向另外n个空闲块的地址
- Counting 空闲块表
  - 空闲空间表中每个条目记录一组连续空闲块的起始地址和数量.

Linked Free Space List on Disk



UNIX Grouping



0代表
最后一块

空闲块

专用块

空闲盘块的成组链接法

第一行100表示：

第一组空闲块块数；

其他行表示块号。

先分配201

Efficiency and Performance

- 磁盘服务 --> 速度是系统性能的主要瓶颈之一
- 设计文件系统应尽可能减少磁盘访问次数

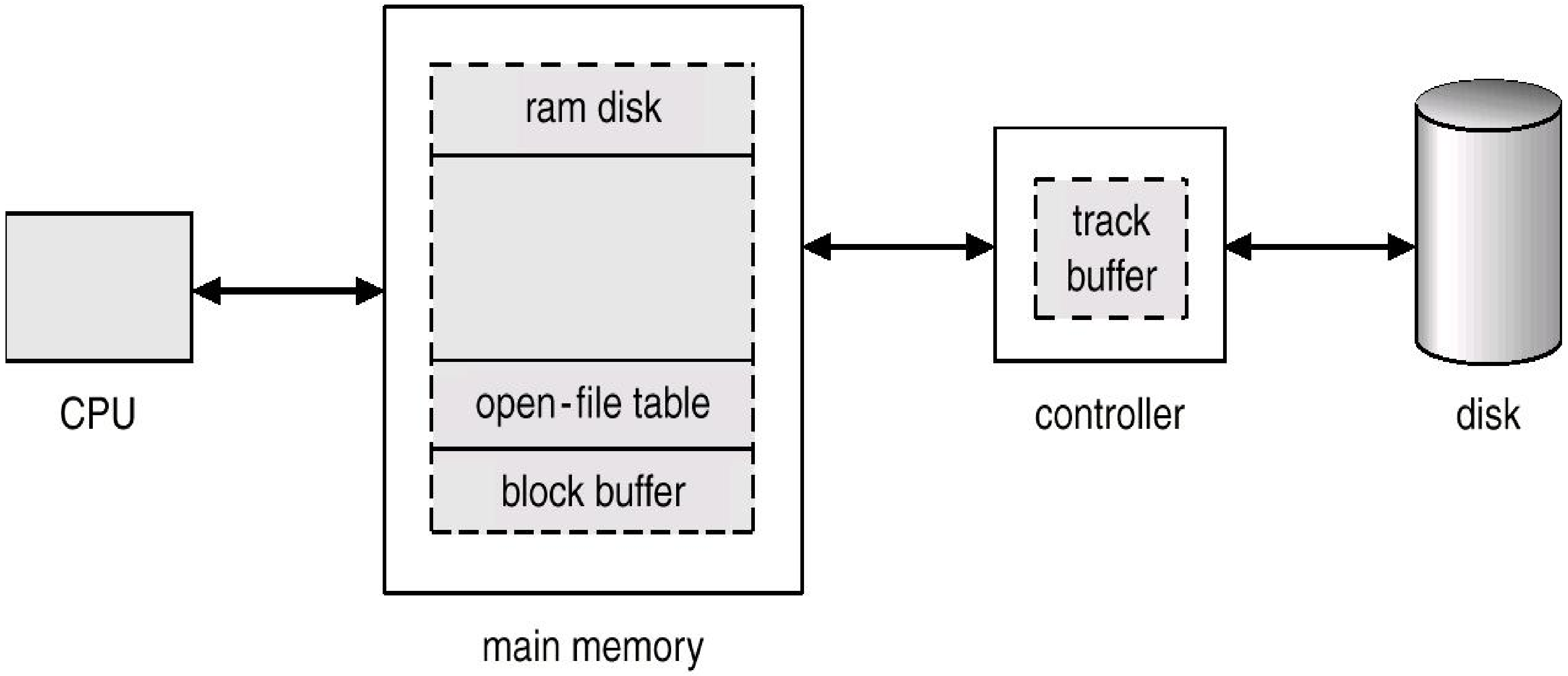
- 提高文件系统性能的方法：
  - 目录项（FCB）分解、当前目录、磁盘碎片整理、合理分配磁盘空间
  - 块高速缓存、提前读取
  - 磁盘调度、RAID技术等

Efficiency and Performance

- Efficiency dependent on:
  - disk allocation and directory algorithms
  - types of data kept in file’s directory entry

- Performance
  - disk cache（文件缓存、磁盘高速缓存、块高速缓存、缓冲区高速缓存）– separate section of main memory for frequently used blocks 位于内核内存中
  - read-ahead（提前读取）and free-behind （随后释放） – techniques to optimize sequential access

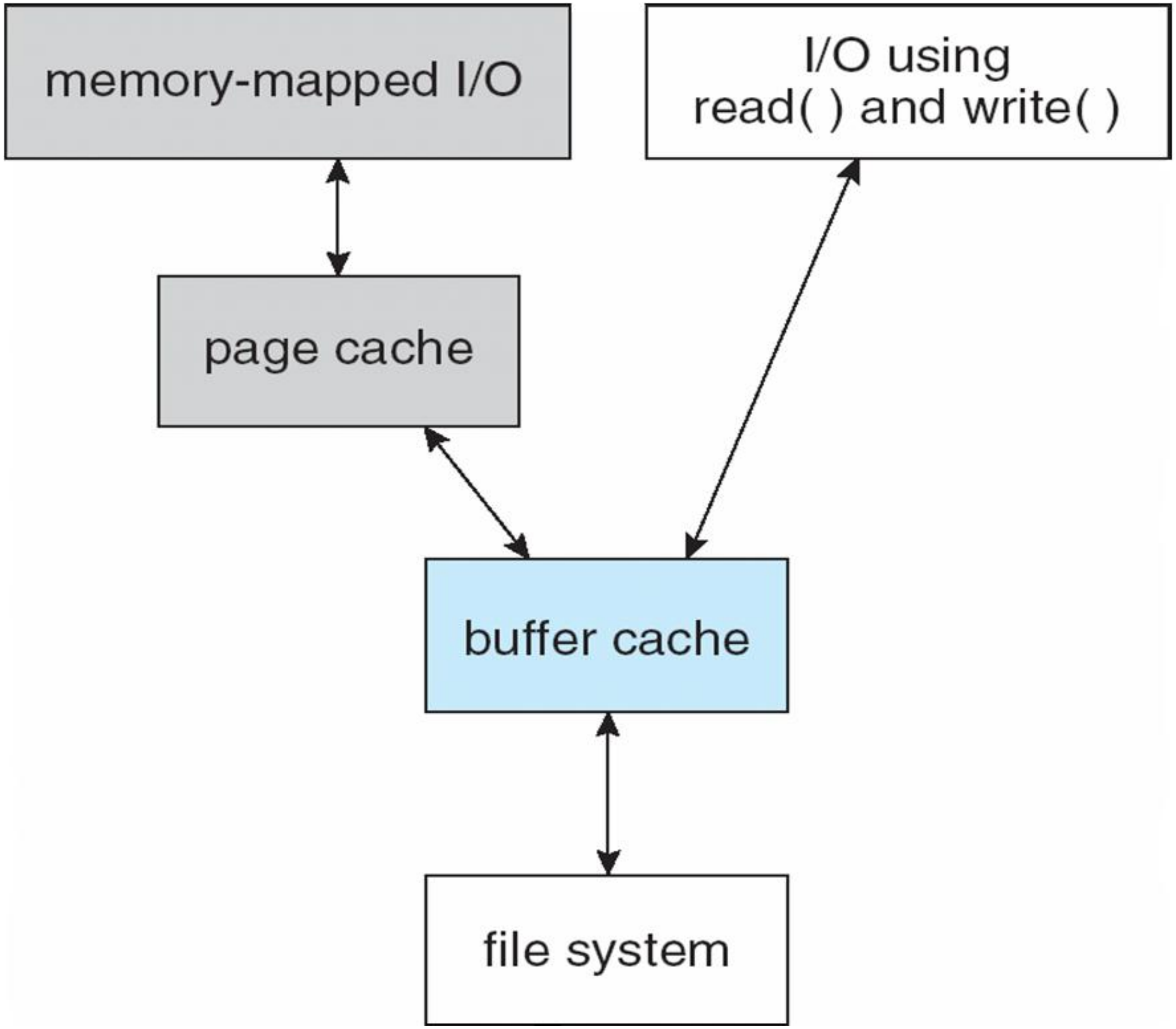
Various Disk-Caching Locations



Page Cache

- A page cache caches pages rather than disk blocks using virtual memory techniques
- Memory-mapped I/O uses a page cache
- Routine I/O through the file system uses the buffer (disk) cache
- This leads to the following figure

I/O Without a Unified Buffer Cache



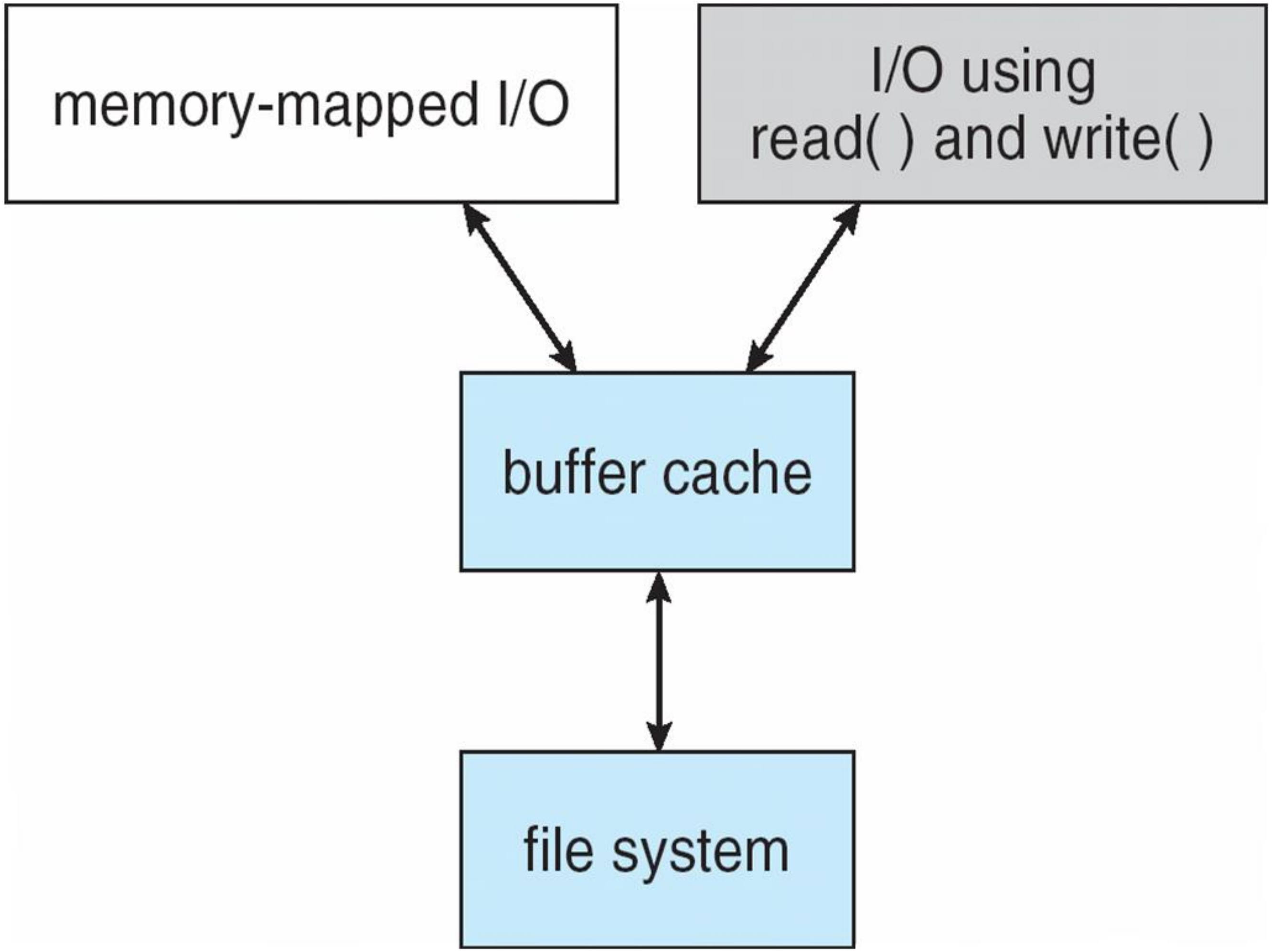
I/O Without a Unified Buffer Cache

- Memory-mapped I/O example:
  - Read in disk blocks from the file system and store them in the buffer cache;
  - Copy the blocks into the page cache
- This is called double caching
  - Waste memory
  - Waste CPU and I/O cycles
  - Inconsistencies between the two caches

Unified Buffer Cache

- A unified buffer cache uses the same page cache to cache both memory-mapped pages and ordinary file system I/O
- examples: Solaris, Linux, Windows NT, 2000, XP

I/O Using a Unified Buffer Cache



File System Consistency

- Multi-step updates cause problems for consistency
  - if crash happens in the middle
- E.g. transfer $100 from my account to Janet’s
  - 1. deduct $100 from my account
  - 2. add $100 to Janet’s account
- What happens if you crash between step 1 and 2?

File System Consistency

- E.g. Create a file
  - 1. allocate an I-NODE, and write it to disk
  - 2. write address of the I-NODE and the file name to file directory
- If crash between the 2 steps?
  - 文件在磁盘上, 但文件夹中却没有出现
  - 所谓的”孤儿”文件
- 如果将两步颠倒, 则可能会出现”魅影”文件

File System Consistency

- Consistency checking – compares data in directory structure with data blocks on disk, and tries to fix inconsistencies
  - Consistency checker—a system program running at reboot time
  - UNIX: fsck
  - MS-D0S: chkdsk
- Use system programs to back up data from disk to another storage device (magnetic tape, other magnetic disk, optical)
- Recover lost file or disk by restoring data from backup

Log Structured File Systems

- Log structured (or journaling) file systems record each update to the file system as a transaction(事务)
- All transactions are written to a log
  - A transaction is considered committed once it is written to the log
  - However, the file system may not yet be updated
- The transactions in the log are asynchronously written to the file system
  - When the file system is modified, the transaction is removed from the log
- If the file system crashes, all remaining transactions in the log must still be performed

End of Chapter 11