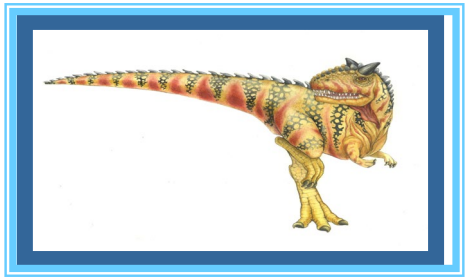
Chapter 11: File System Implementation



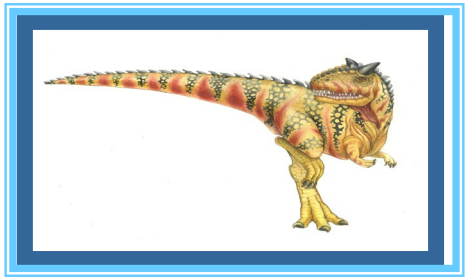
File System Implementation

- 11.1 File-System Structure
- 11.2 File-System Implementation
- 11.3 Directory Implementation
- 11.4 Allocation Methods
- 11.5 Free-Space Management
- 11.6 Efficiency and Performance
- 11.7 Recovery
- 11.8 NFS

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

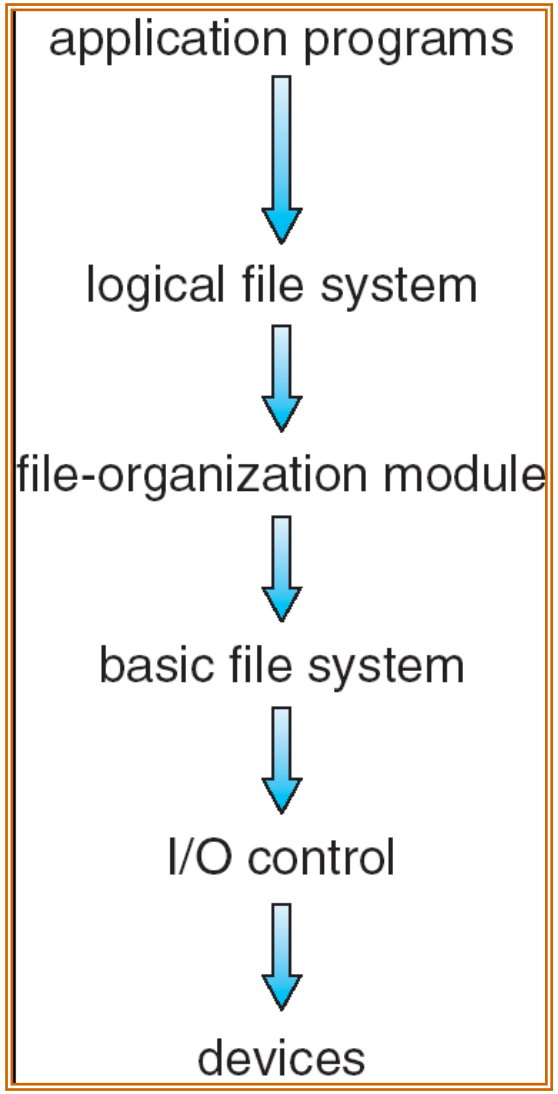
11.1 File-System(organized) Structure



File-System(organized) Structure

- File structure
  - Logical storage unit
  - Collection of related information
- File system organized into layers
- File system resides on secondary storage (disks)
  - Provides efficient and convenient access to disk by allowing data to be stored, located retrieved easily
  - 文件系统是操作系统中以文件方式管理计算机软件资源的软件和被管理的文件和数据结构（如目录和索引表等）的集合。
- File control block（文件控制块）– storage structure consisting of information(Attributes) about a file
- Device driver（设备驱动）controls the physical device

分层设计的文件系统



- **Application Programs:** The code that's making a file request
- **逻辑文件系统**
  - 管理**元数据**：文件系统的所有结构数据，而不包括实际数据（或文件内容）
  - 根据给定符号文件名来管理目录结构
  - 逻辑文件系统通过**文件控制块(FCB)**来维护文件结构
- **文件组织模块**
  - 知道文件及其逻辑块和物理块。
  - 空闲空间管理器
- **基本文件系统**
  - 向合适的设备驱动程序发送一般命令就可对磁盘上的物理块进行读写
- **I/O控制**
  - 由**设备驱动程序**和**中断处理程序**组成，实现内存与磁盘之间的信息转移

File System Types

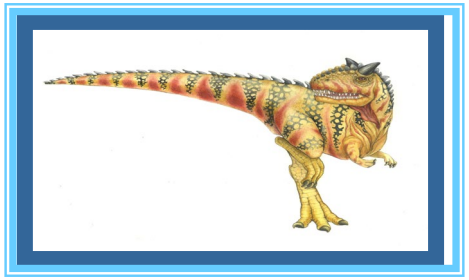
- FAT（MS-DOS文件系统）
- FAT32（VFAT）(win98文件系统)
- **NTFS（NT文件系统）**
- S51K/S52K（AT&T UNIX sysv）
- ext (minix文件系统)
- **ext2、ext3、ext4（linux文件系统、Android）**
- proc、sysfs（linux虚拟文件系统）
- yaffs (Yet Another Flash File System)
- ReiserFS (Linux一种日志文件系统)
- **HFS+ (Mac OS，iOS文件系统)**
- HPFS（OS/2高性能文件系统）
- UFS(BSD UNIX的一种文件系统)

File System Types

- **iso9660 (通用的光盘文件系统)**
- NFS (网络文件系统)
- VFS (Linux虚拟文件系统)

  VFS是物理文件系统与服务之间的一个接口，它屏蔽各类文件系统的差异，给用户和程序提供一个统一的接口
- ZFS （Open Solaris文件系统）
- **LTFS(Liner Tape File System，线性磁带文件系统) 为磁带提供了一种通用、开放的文件系统。**
- **APFS（Apple File System）苹果新一代的文件系统，MAC os、iOS**
- ….

11.2 File-System Implementation

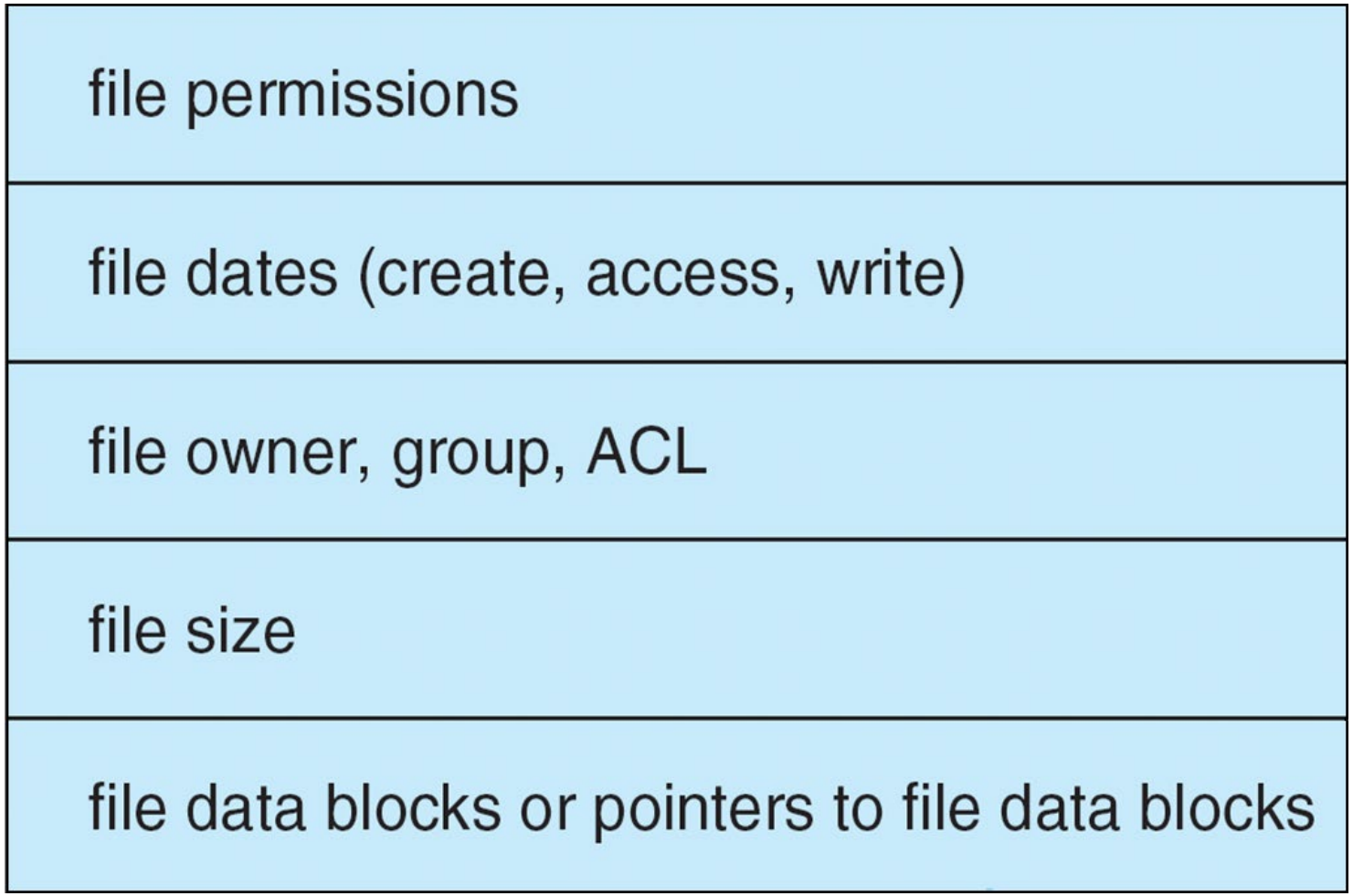


File-System Implementation

1. On-Disk file system structures:

- 在磁盘上，文件系统可能包括如下信息：
  - 如何启动所存储的操作系统
  - 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
- 磁盘结构包括
  - Boot control block contains info needed by system to boot OS from that volume
  - Volume（卷） control block contains volume details
  - Directory structure organizes the files
  - Per-file File Control Block (FCB，文件控制块) contains many details about the file
- 实例：Linux ext2

A Typical File Control Block



In-Memory File System Structures

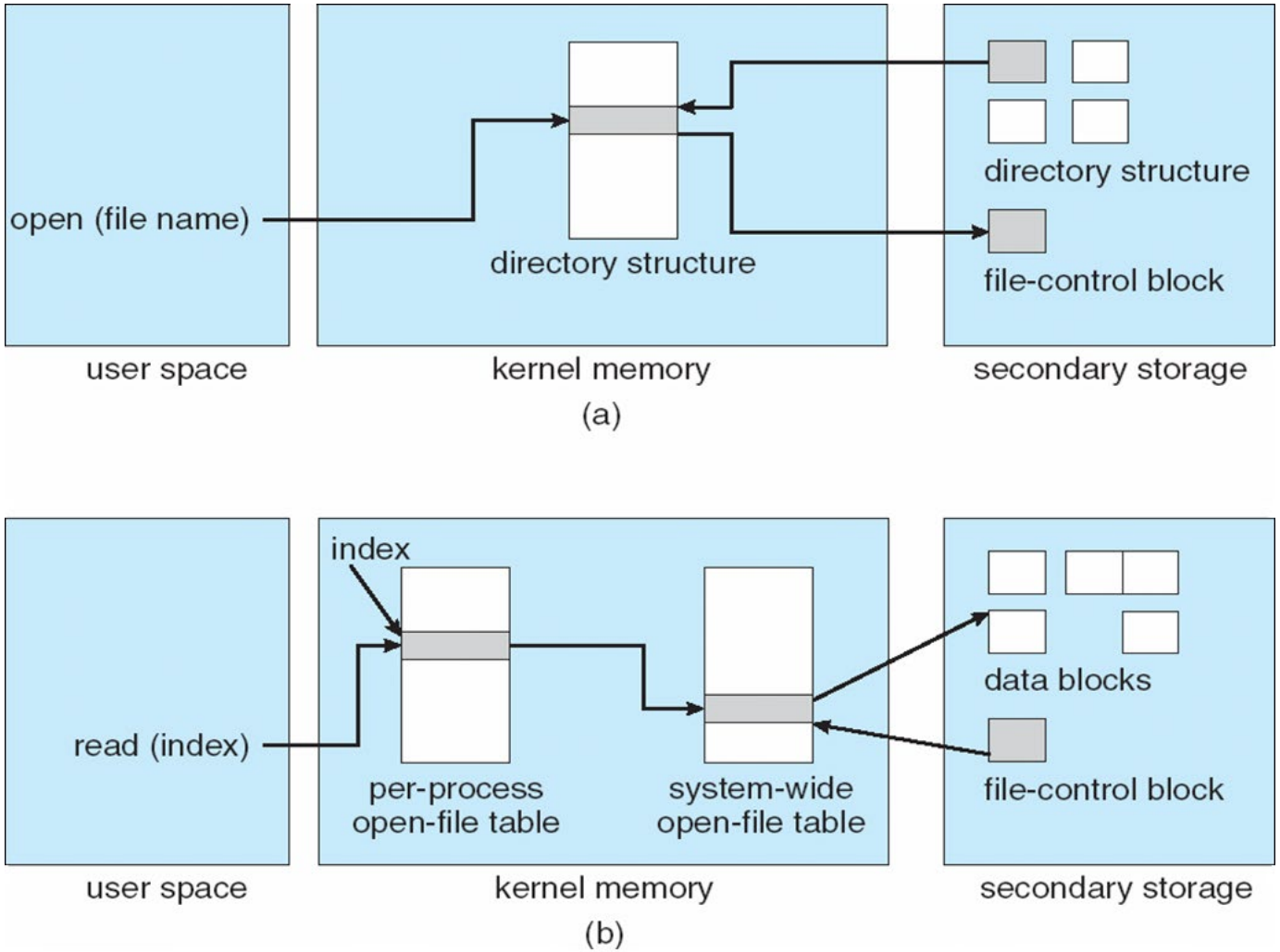
2. In-Memory File System Structures:

- An in-memory partition table(分区表)
- An in-memory directory structure（目录结构）
- The system-wide open-file table （系统打开文件表）
- The per-process open-file table （进程打开文件表）

  - Figure 12-3(a) refers to opening a file.
  - Figure 12-3(b) refers to reading a file.

- 实例：Linux vfs

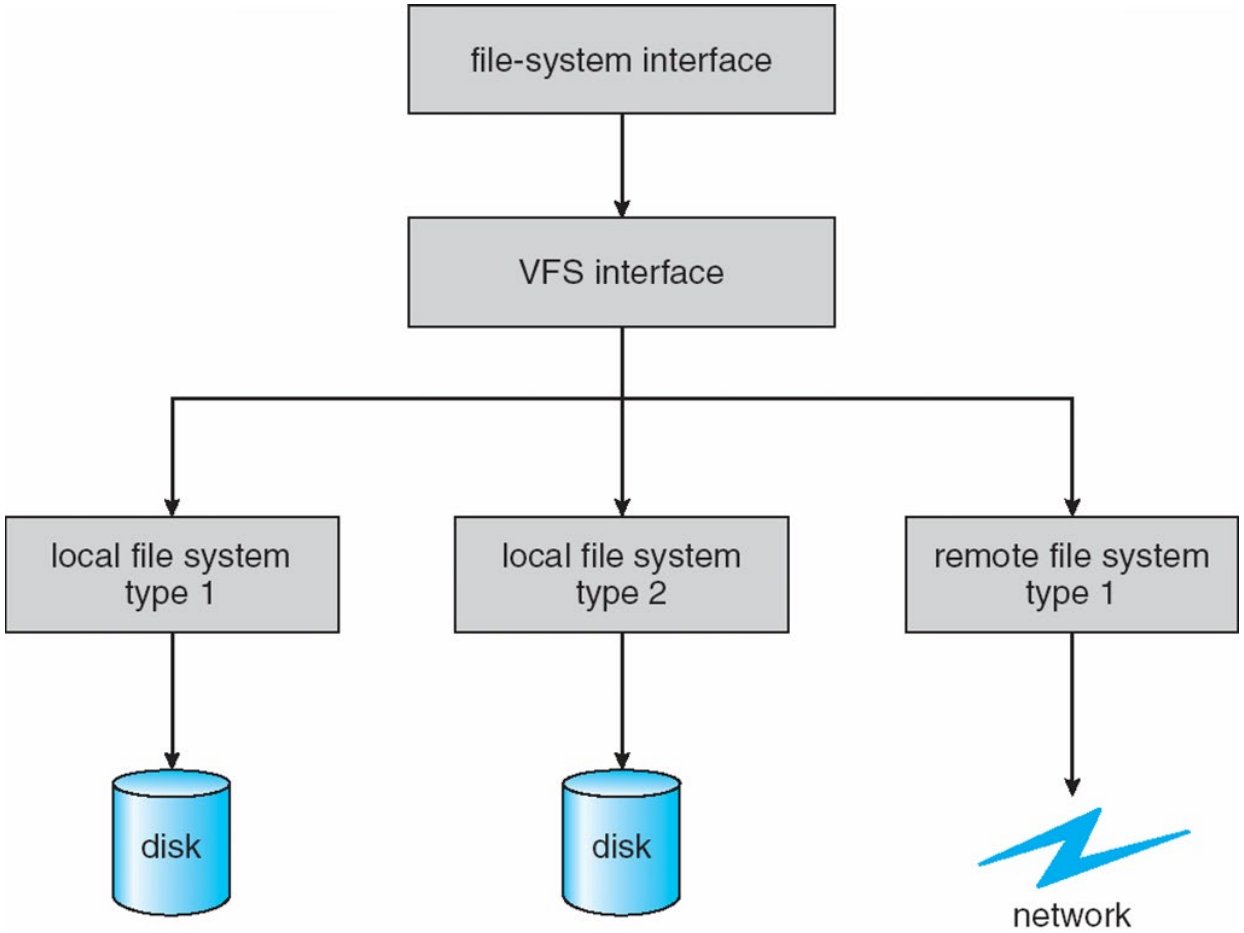
In-Memory File System Structures



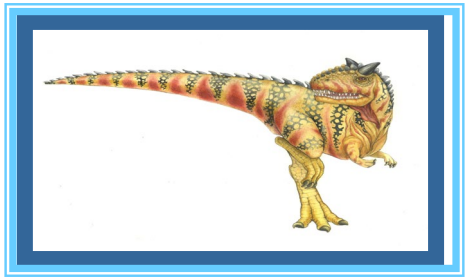
Virtual File Systems

- 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.

Schematic View of Virtual File System

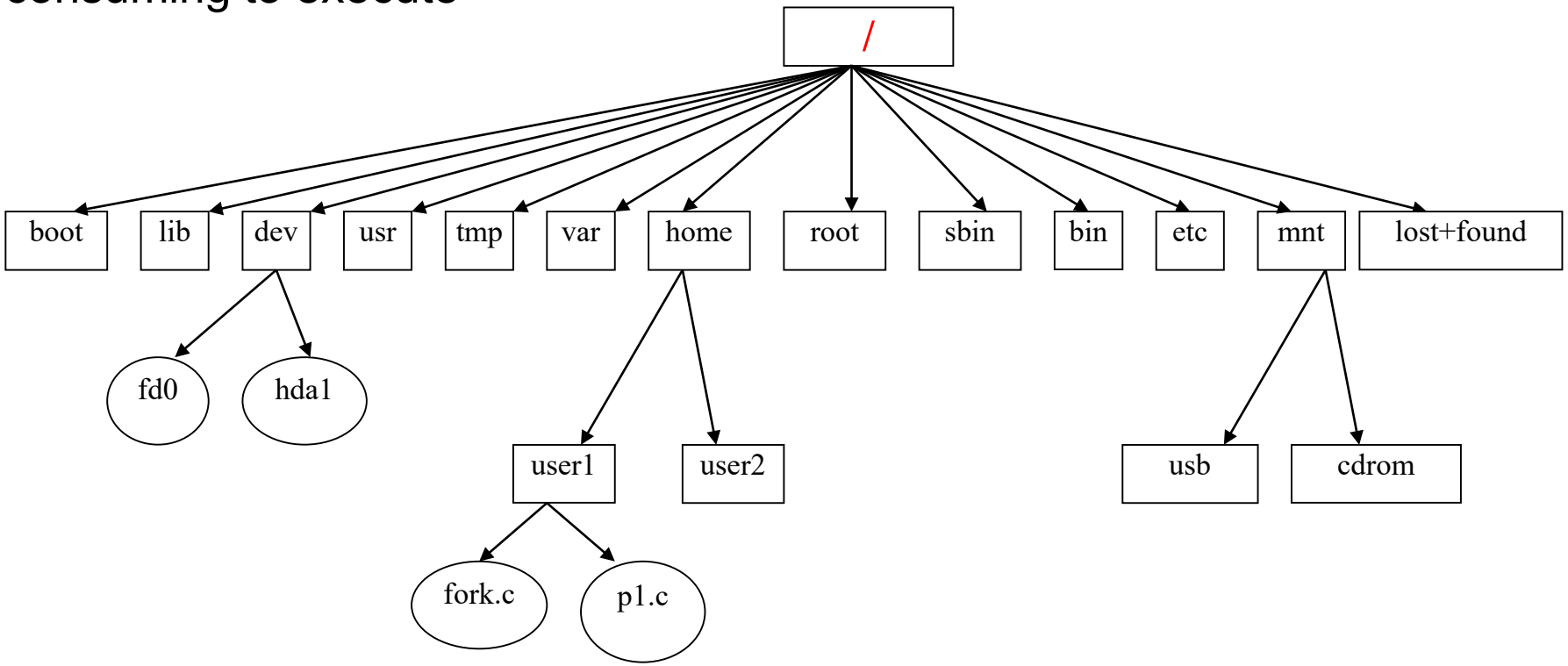


11.3 Directory Implementation



Directory Implementation

- **Linear list**（线性检索法）of file names with pointer to the data blocks.
  - simple to program
  - time-consuming to execute

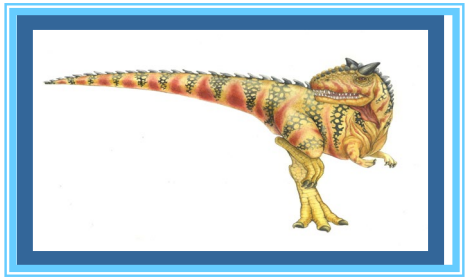


Directory Implementation

- **Hash Table**（哈希表） – linear list with hash data structure.
  - decreases directory search time
  - **collisions** – situations where two file names hash to the same location
  - fixed size

- 索引

11.4 Allocation Methods，文件物理结构



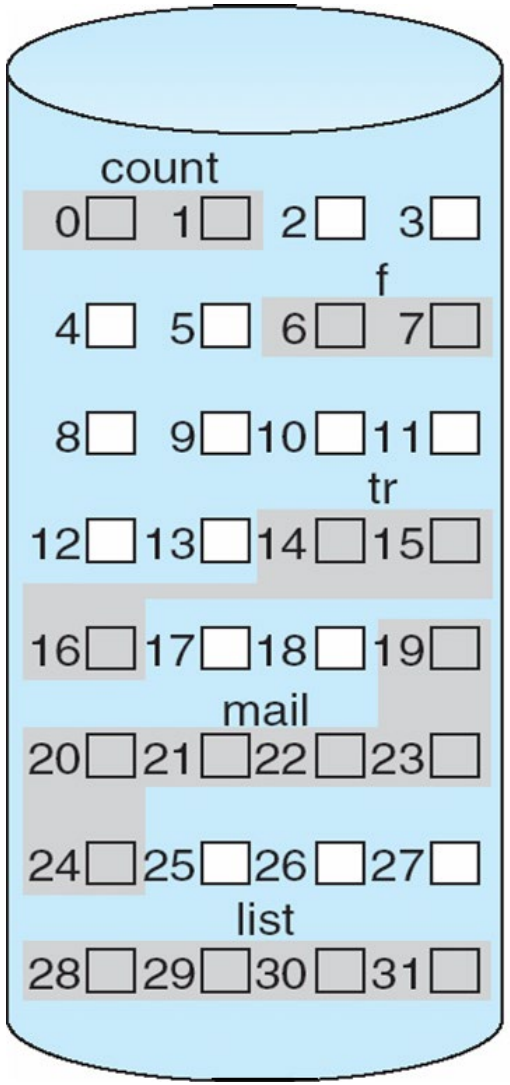
Allocation Methods，文件物理结构

- An allocation method refers to how disk blocks are allocated for files:
  - Contiguous allocation（连续分配）
  - Linked allocation （链接分配）
  - Indexed allocation（索引分配）
  - Unix、Linux直接间接混合分配方法

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
- **Random access**（随机存取）
- Wasteful of space (dynamic storage-allocation problem)
- Files cannot grow

Contiguous Allocation of Disk Space

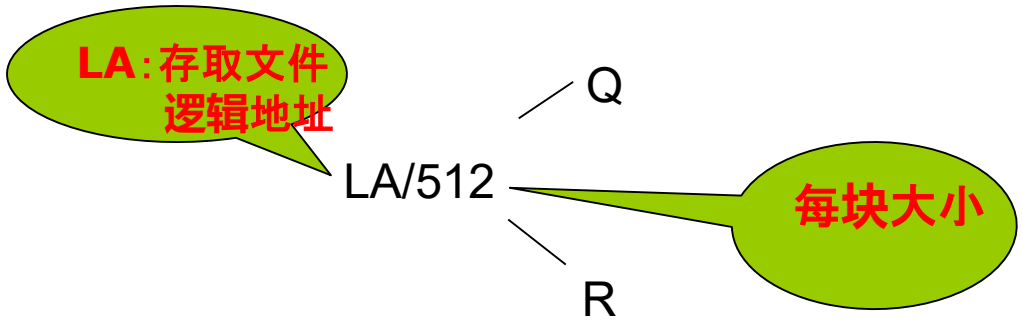


directory

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

Contiguous Allocation

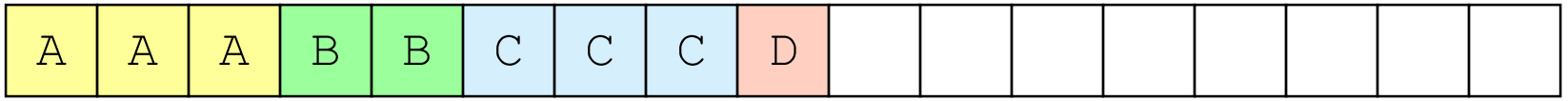
- Mapping from logical to physical



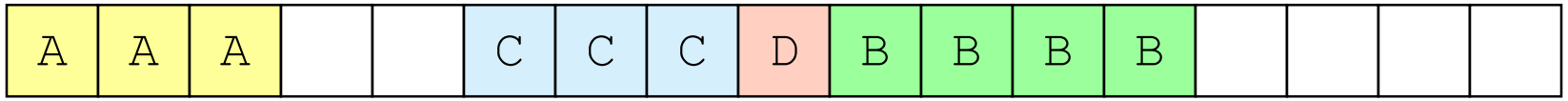
Block to be accessed = Q + starting address
Displacement into block = R

Contiguous Allocation of Disk Space

- Four files allocated contiguously to disk:

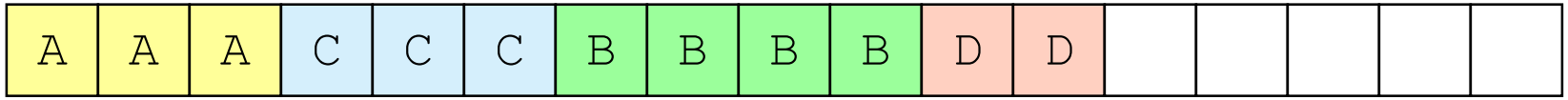


- File B outgrows its space and is reallocated:



- File D outgrows its space and is reallocated:

- Defragmentation combines free disk space:

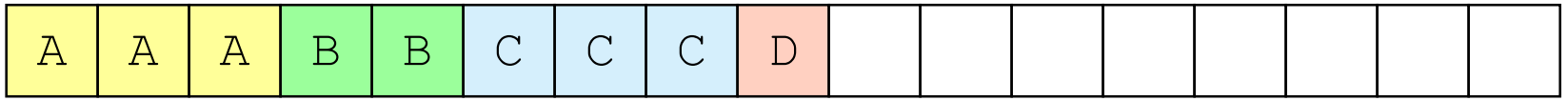


Extent-Based Systems （基于长度系统）

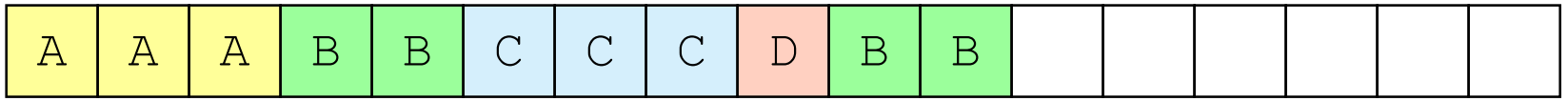
- Many newer file systems (I.e. Veritas File System) use a modified contiguous allocation scheme
- Extent-based file systems allocate disk blocks in extents
- An extent is a contiguous block of disks
  - Extents are allocated for file allocation
  - A file consists of one or more extents

Extent-Based Systems

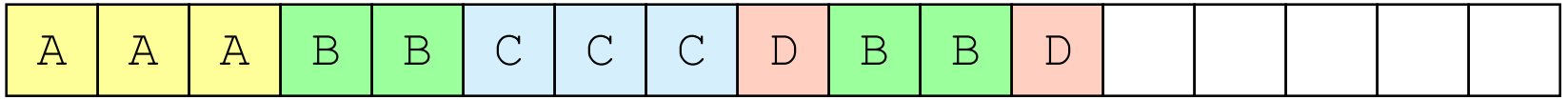
- Four files allocated contiguously to disk:



- File B outgrows its space and a cluster is allocated:



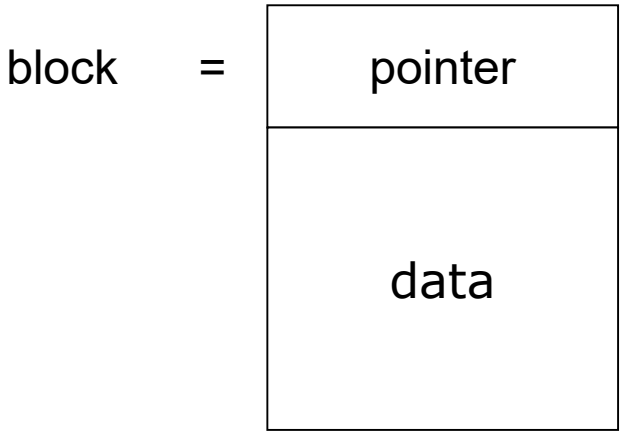
- File D outgrows its space and a cluster is allocated:



- Need for defragmentation is slightly reduced

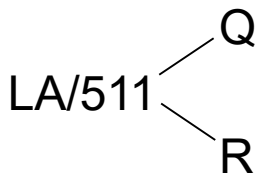
Linked Allocation

- Each file is a linked list of disk blocks: blocks may be scattered anywhere on the disk.



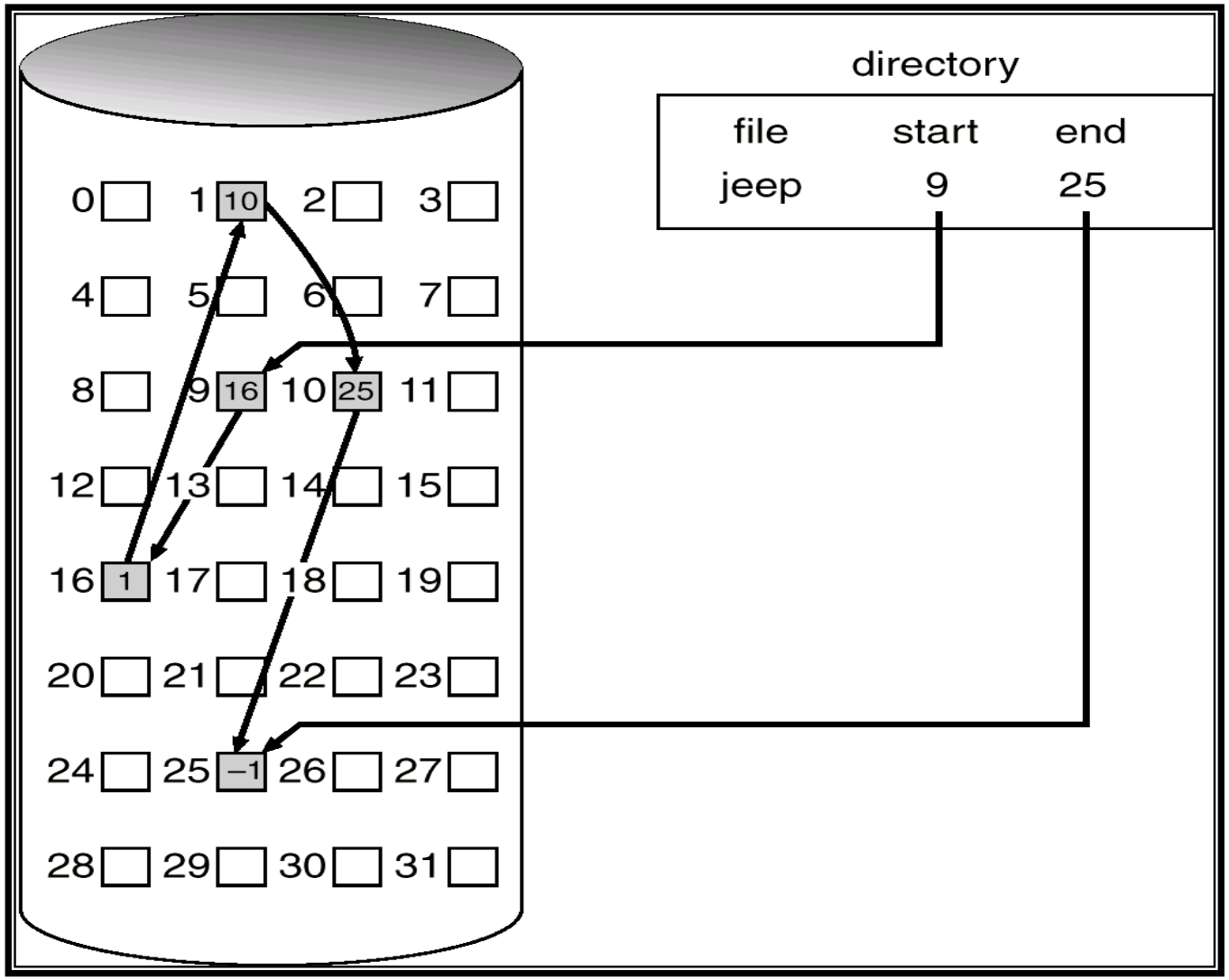
Linked Allocation (Cont.)

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



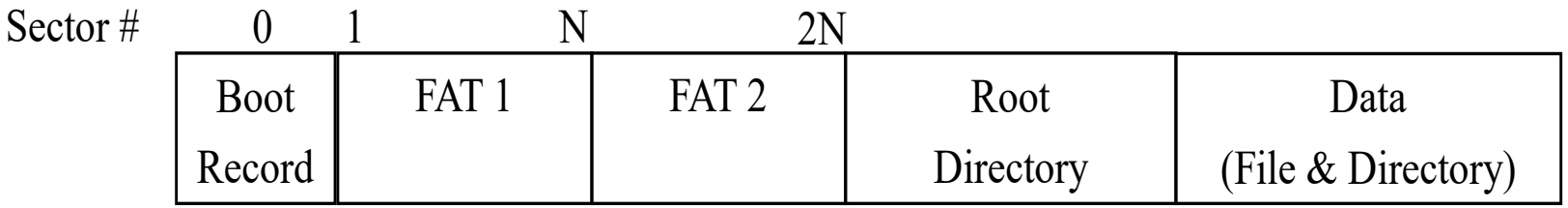
Block to be accessed is the Qth block in the linked chain of blocks representing the file.
Displacement into block = R + 1

Linked Allocation



FAT文件系统

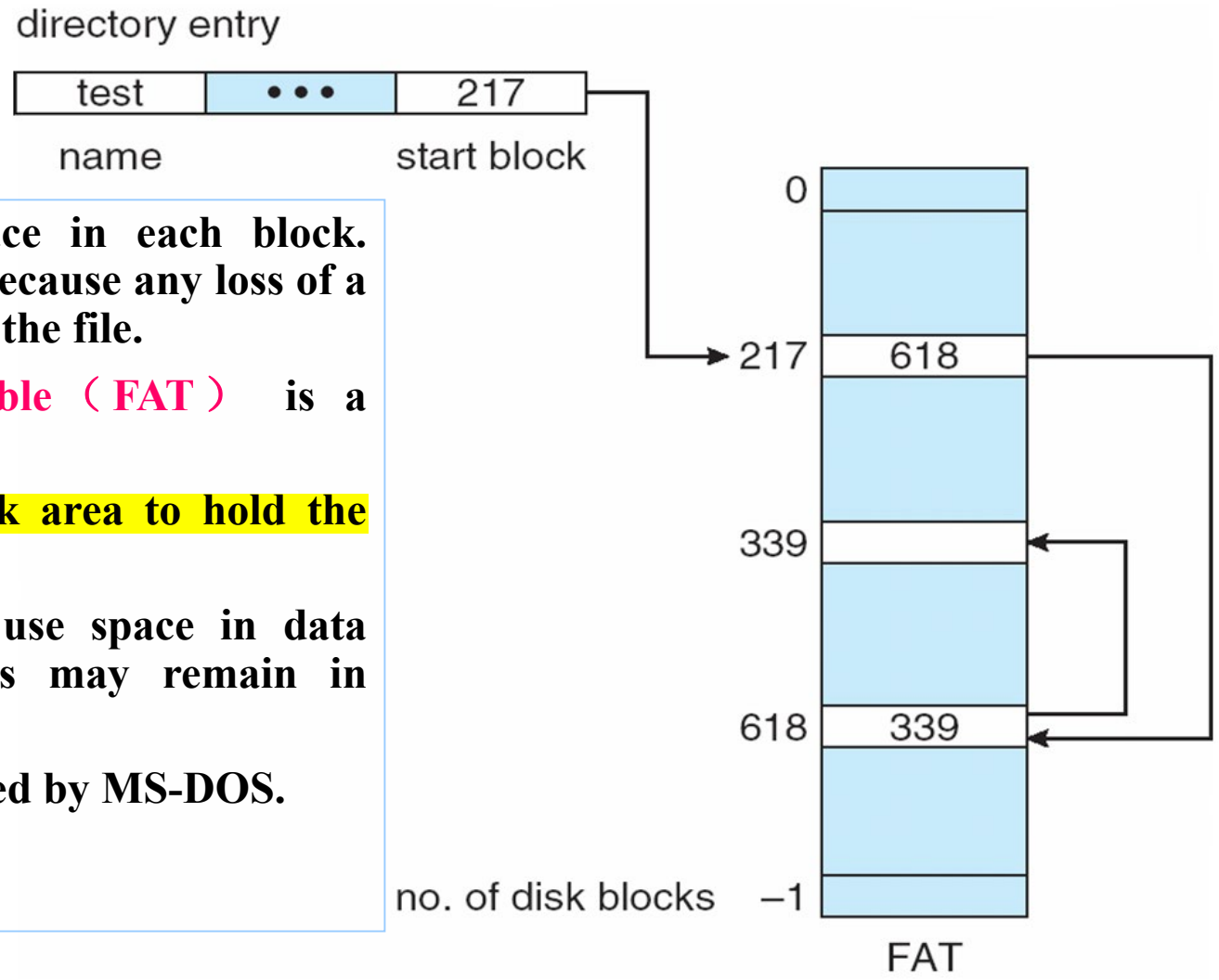
- File-allocation table (FAT) – disk-space allocation used by MS-DOS and OS/2.
- FAT12、FAT16、FAT32



Volume Structure in MS DOS

- FAT32引导区记录被扩展为包括重要数据结构的备份，根目录为一个普通的簇链，其目录项可以放在文件区任何地方。

File-Allocation Table



- Pointers use up space in each block. Reliability is not high because any loss of a pointer loses the rest of the file.
- A File Allocation Table（FAT） is a variation of this.
- It uses a separate disk area to hold the links.
- This method doesn't use space in data blocks. Many pointers may remain in memory.
- A FAT file system is used by MS-DOS.

FAT32磁盘的结构

- 主引导记录MBR是主引导区的第一个扇区，它由二部分组成：
  - 第一部分主引导代码，占据扇区的前446个字节，磁盘标识符（FD 4E F2 14）位于这段代码的末尾。
  - 第二部分是分区表，分区表中每个条目有16字节长，分区表最多有4个条目，第一个分区条目从扇区的偏移量位置是0x01BE。
- 扩展引导记录与主引导记录类同，如该扩展分区未装操作系统则第一部分主引导代码为0，标签字也标记一个扩展分区引导区和分区引导区的结束。
- 计算机系统启动时，首先执行的是BIOS引导程序，完成自检，并加载主引导记录和分区表，然后执行主引导记录，由它引导激活分区引导记录，再执行分区引导记录，加载操作系统，最后执行操作系统，配置系统。

FAT32目录项结构

- FAT的每个目录项为32个字节
- FAT32长文件名的目录项由几个32B表项组成。
- 用一个表项存放短文件名和其他属性（包括簇号、文件大小，最后修改时间和最后修改日期、创建时间、创建日期和最后存取日期），短文件名的属性是0x20。
- 用连续若干个表项存放长文件名，每个表项存放13个字符（使用Unicode编码，每个字符占用2个字节。）
- 长文件名的表项首字节的二进制数低5位值，分别为00001 、00010 、00011 、……，表示它们的次序，左起第2位为1（也就是在低5位基础上加40H）表示该表项是最后一项。最后项存放13个字符位置多余时，先用2个字节0表示结束，再用FFH填充。长文件名的属性是0x0F。长文件名项的第13、27、28字节为0x00，第14字节为短文件名校验和。

FAT32目录结构

- 长文件名The quick brown.fox（短文件名为THEQUI~1.FOX）
  目录项格式如下：

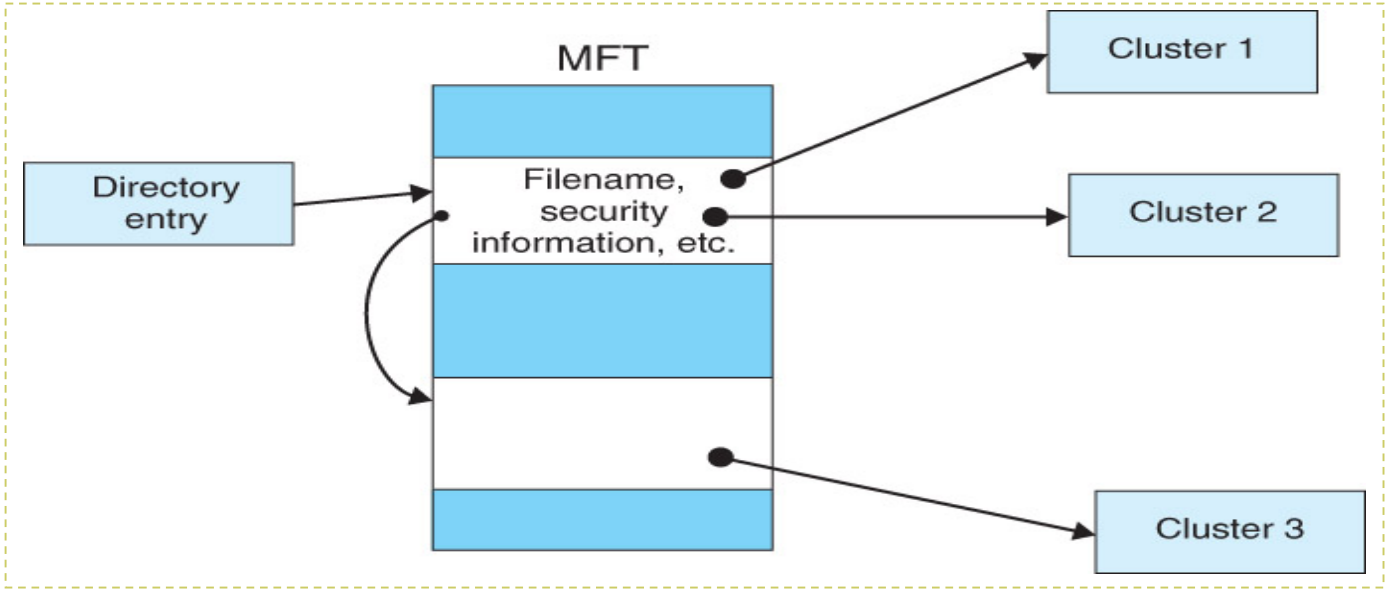
2	42	w 77 00	n 6E 00	2E 00	f 66 00	o 6F 00	属性 0F	00	校验和 07	X 78 00	
	00 00	FF FF	FF FF	FF FF	FF FF	FF FF	00 00	FF FF	FF FF		
1	01	T 54 00	h 68 00	e 65 00	20 00	q 71 00	属性 0F	00	校验和 07	U 75 00	
	i 69 00	c 63 00	k 6B 00	00	b 62 00	00 00	r 72 00	O 6F 00			
0	短 文 件 名 T H E Q U I ~ 1				扩展名 F O X	属性 20		创建时间			
	创建日期	最后存取日期	00 00	最后修改时间	最后修改日期	第一簇号	文 件 大 小				

File System (NTFS)

- **NTFS卷布局：**

引导扇区	主文件表(MFT)（前16个为系统文件）	文件区	MFT前16个元数据文件备份	文件区

- Each partition has a **Master File Table (MFT)**:



MFT用数据库记录形式组织，每条记录（MFT表项）长度1K

File System (NTFS)

- MFT由一个个MFT项（也称为文件记录）组成，每个MFT项占用1024字节的空间。
- MFT前16个记录用来存放元数据文件的信息，它们占有固定的位置。
- 每个MFT项的前部几十个字节有着固定的头结构，用来描述本MFT项的相关信息。后面的字节存放着文件属性等。
- 每个文件或目录的信息都包含在MFT中，每个文件或目录至少有一个MFT项。

小文件的内容就可以直接放在这里了 不占用其他空间

MFT表项信息

Indexed Allocation

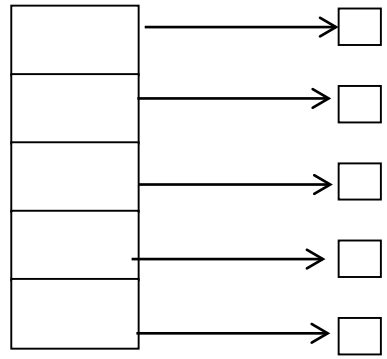
- Brings all pointers together into the index block
- Logical view
- Each file uses an index block on disk to contain addresses of other disk blocks used by the file.
- When the i th block is written, the address of a free block is placed at the i th position in the index block.

- Method suffers from wasted space since, for small files, most of the index block is wasted. What is the optimum size of an index block?

- If the index block is too small, we can:

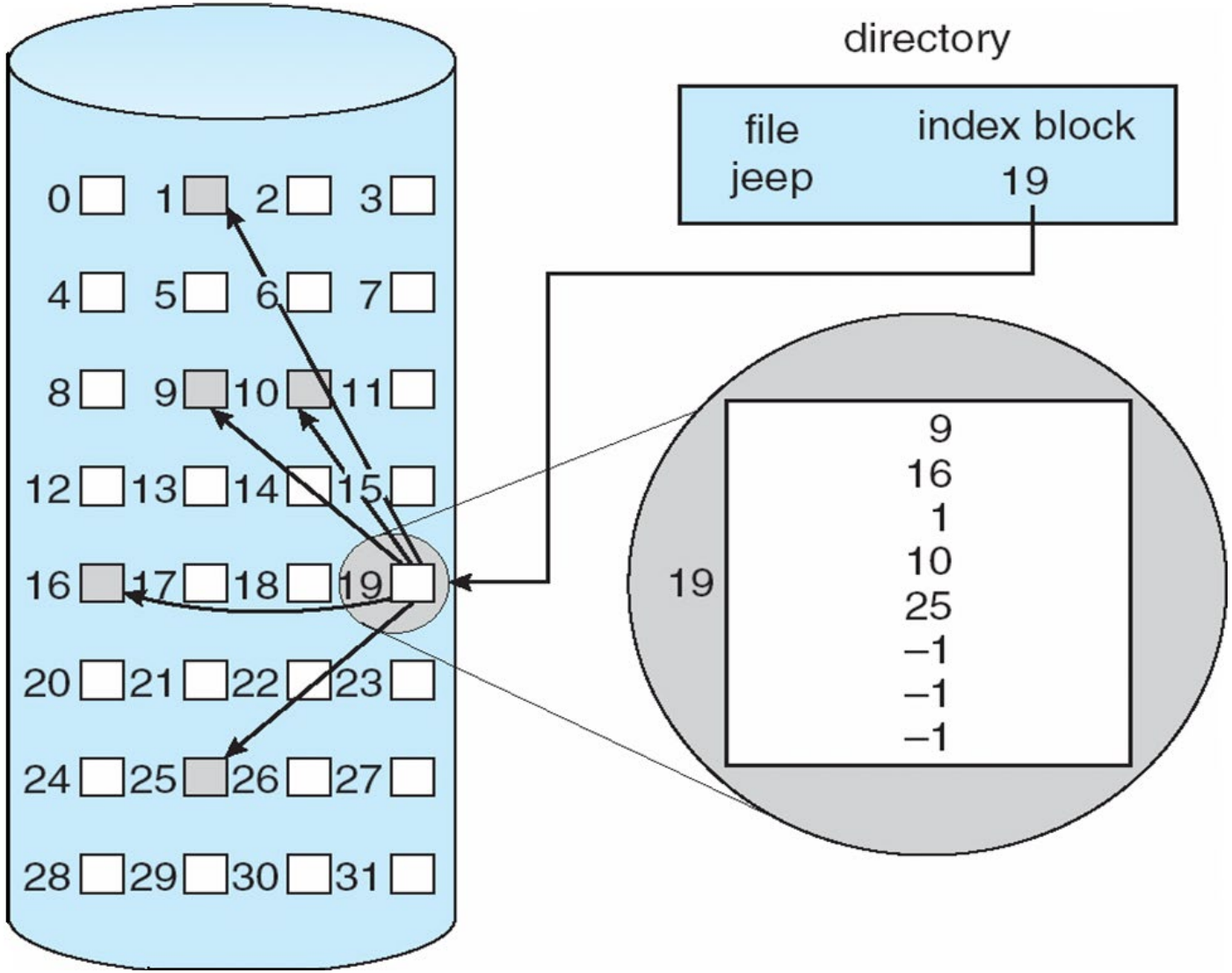
  Link several together

  Use a multilevel index



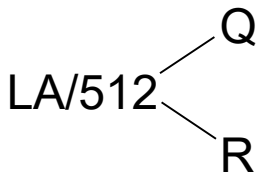
index table

Example of Indexed Allocation



Indexed Allocation (Cont.)

- Need index table
- Random access
- Dynamic access without external fragmentation, but have overhead of index block
- Mapping from logical to physical in a file of maximum size of 256K words and block size of 512 words. We need only 1 block for index table



Q = displacement into index table
R = displacement into block

Indexed Allocation – Mapping (Cont.)

- Mapping from logical to physical in a file of unbounded length (block size of 512 words)
- Linked scheme （链接索引）– Link blocks of index table (no limit on size)

LA / (512 x 511) → Q_1, R_1

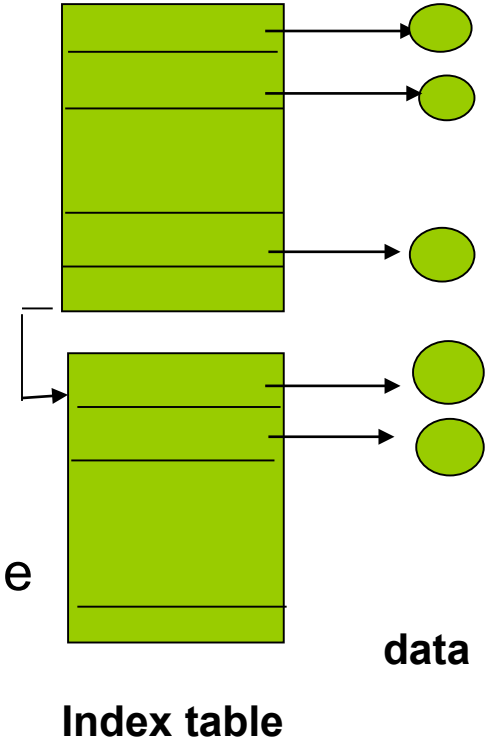
Q_1 = block of index table

R_1 is used as follows:

R_1 / 512 → Q_2, R_2

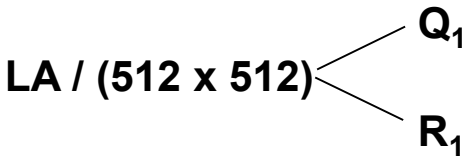
Q_2 = displacement into block of index table

R_2 displacement into block of file:

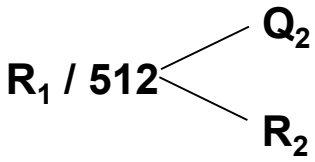


Indexed Allocation – Mapping (Cont.)

- Two-level index (maximum file size is 512^3) 二级索引

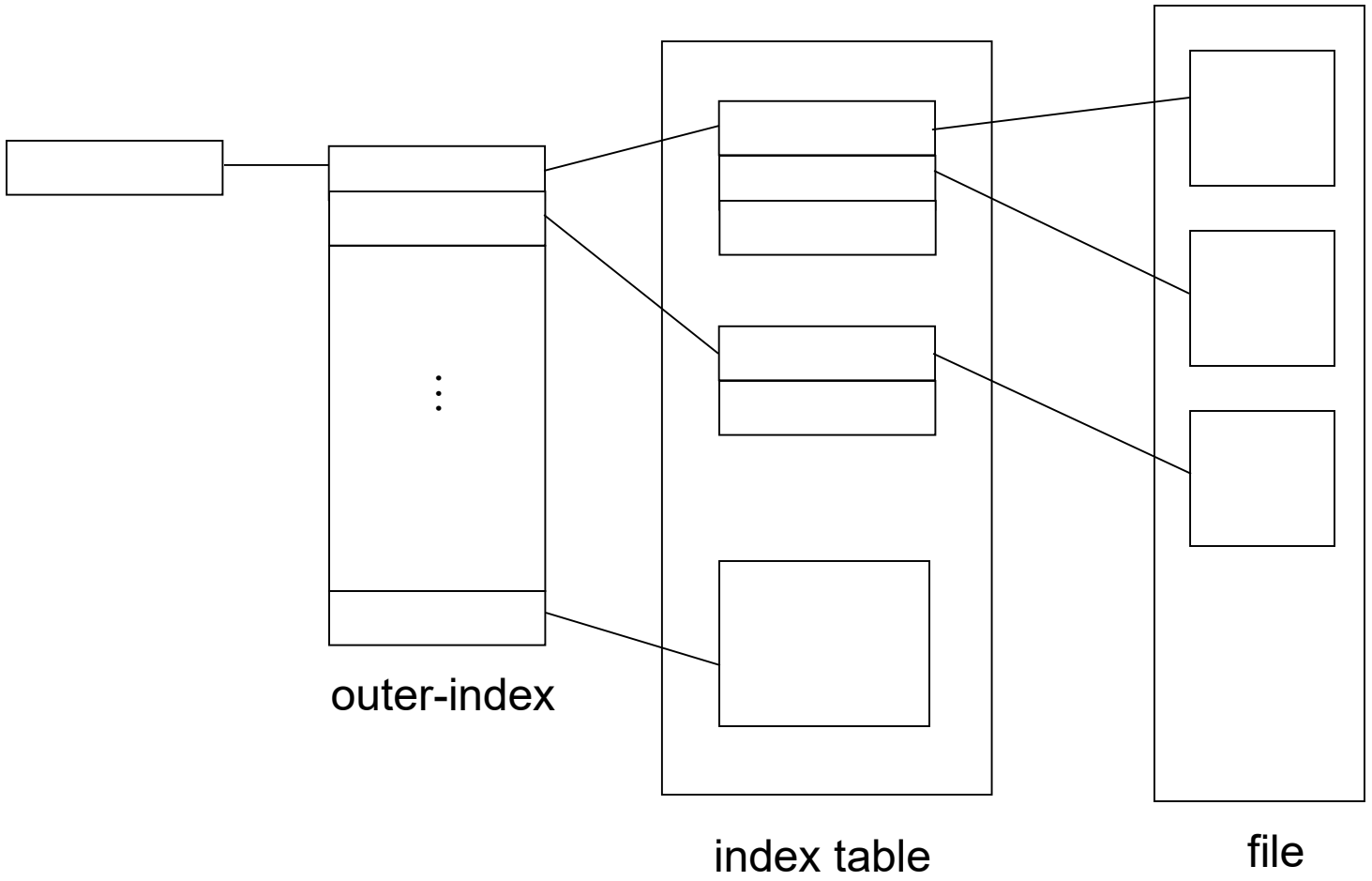


Q_1 = displacement into outer-index
R_1 is used as follows:



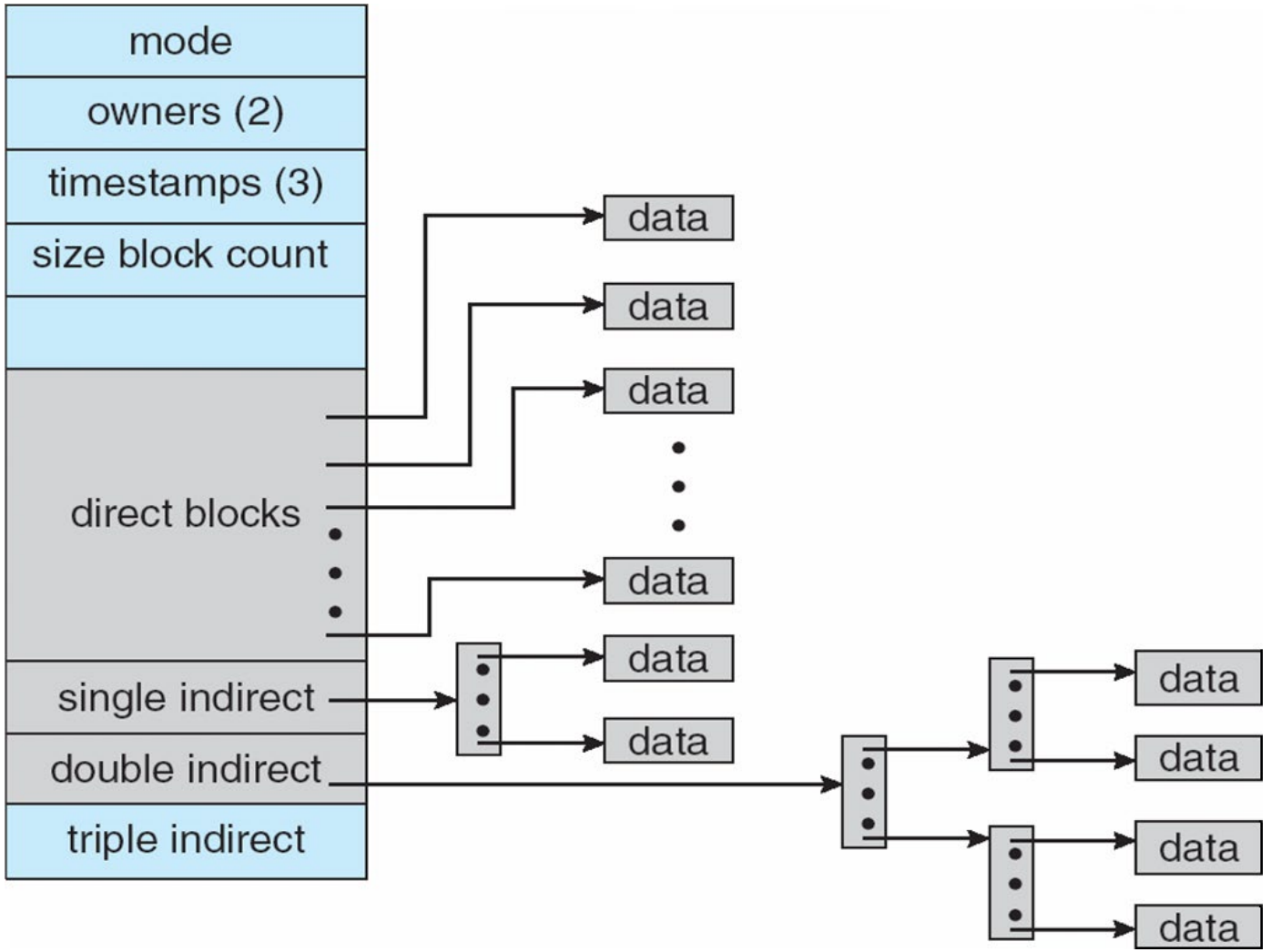
Q_2 = displacement into block of index table
R_2 displacement into block of file:

Indexed Allocation – Mapping (Cont.)



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

- Linux ext2/ext3

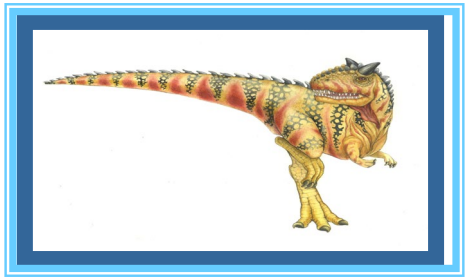


实例

- 一个文件系统中有一个20MB大文件和一个20KB小文件,当分别采用连续、链接、链接索引、二级索引和LINUX 分配方案时，每块大小为4096B,每块地址用4B表示，问:
  （1）各文件系统管理的最大的文件是多少?
  （2）每种方案对大、小两文件各需要多少专用块来记录文件的物理地址(说明各块的用途)？
  （3）如需要读大文件前面第5.5KB的信息和后面第（16M＋5.5KB）的信息，则每个方案各需要多少次盘I/O操作?

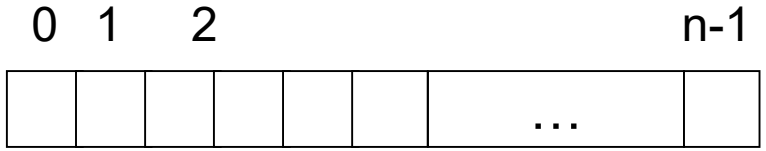
答案

11.5 Free-Space Management



Free-Space Management

1. Bit vector（位图）　(*n* blocks)



$$\text{bit}[i] = \begin{cases} 0 \Rightarrow \text{block}[i] \text{ free} \\ 1 \Rightarrow \text{block}[i] \text{ occupied} \end{cases}$$

Block number calculation

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

Free-Space Management (Cont.)

- Bit map requires extra space
  - Example:

    block size = 2^{12} bytes

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

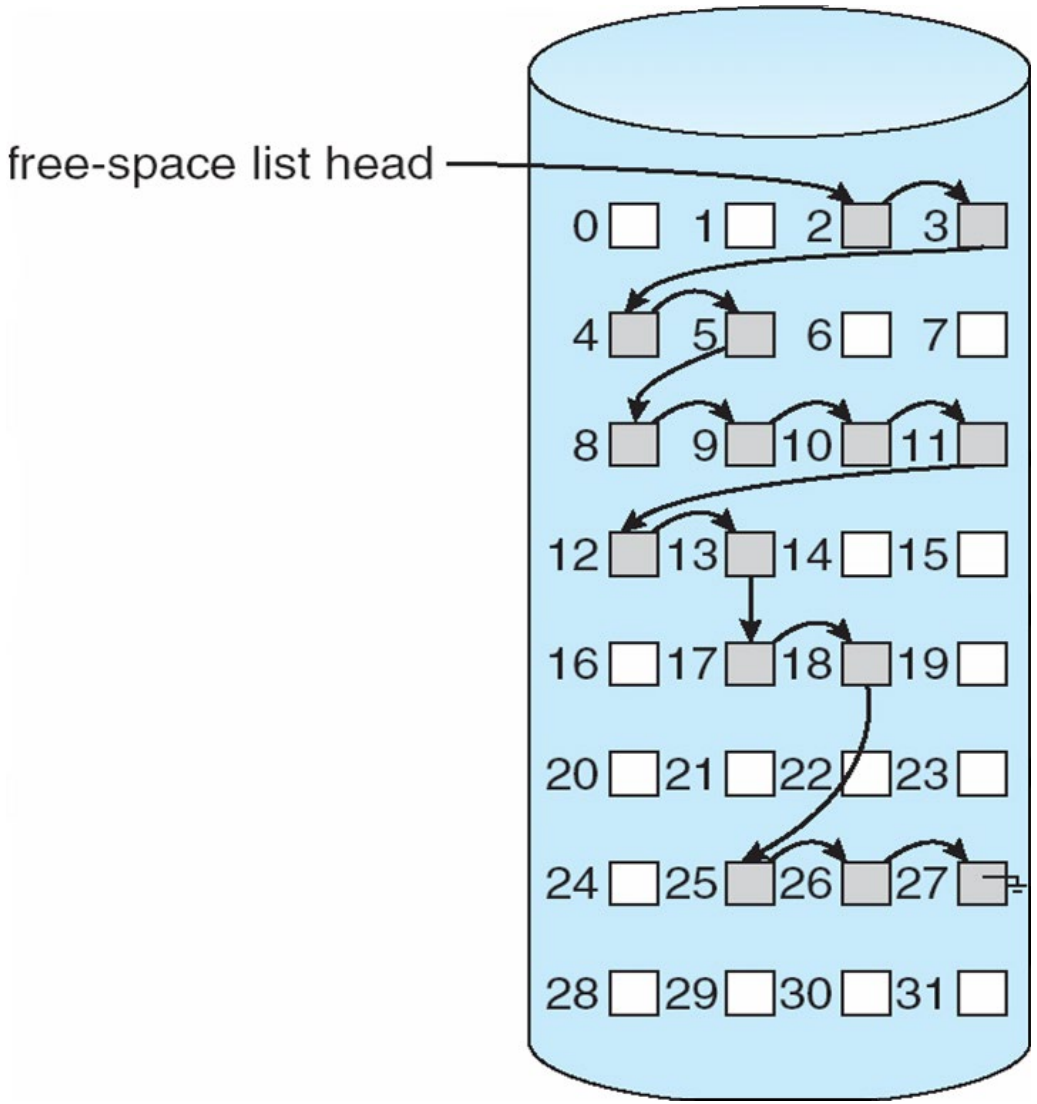
    $n = 2^{30}/2^{12} = 2^{18}$ bits (or 32K bytes)
- Easy to get contiguous files

- **Linked list (free list，空闲链表)**
  - Cannot get contiguous space easily
  - No waste of space
- **Grouping（分组）---Unix 成组连接法(next page)**
- **Counting（空闲表）**

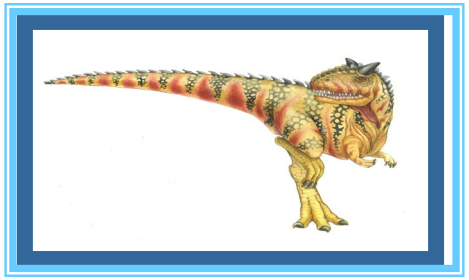
Free-Space Management (Cont.)

- Need to protect:
  - Pointer to free list
  - Bit map
    - Must be kept on disk
    - Copy in memory and disk may differ
    - Cannot allow for block[*i*] to have a situation where bit[*i*] = 1 in memory and bit[*i*] = 0 on disk
  - Solution:
    - Set bit[*i*] = 1 in disk
    - Allocate block[*i*]
    - Set bit[*i*] = 1 in memory

Linked Free Space List on Disk



11.6 Efficiency and Performance



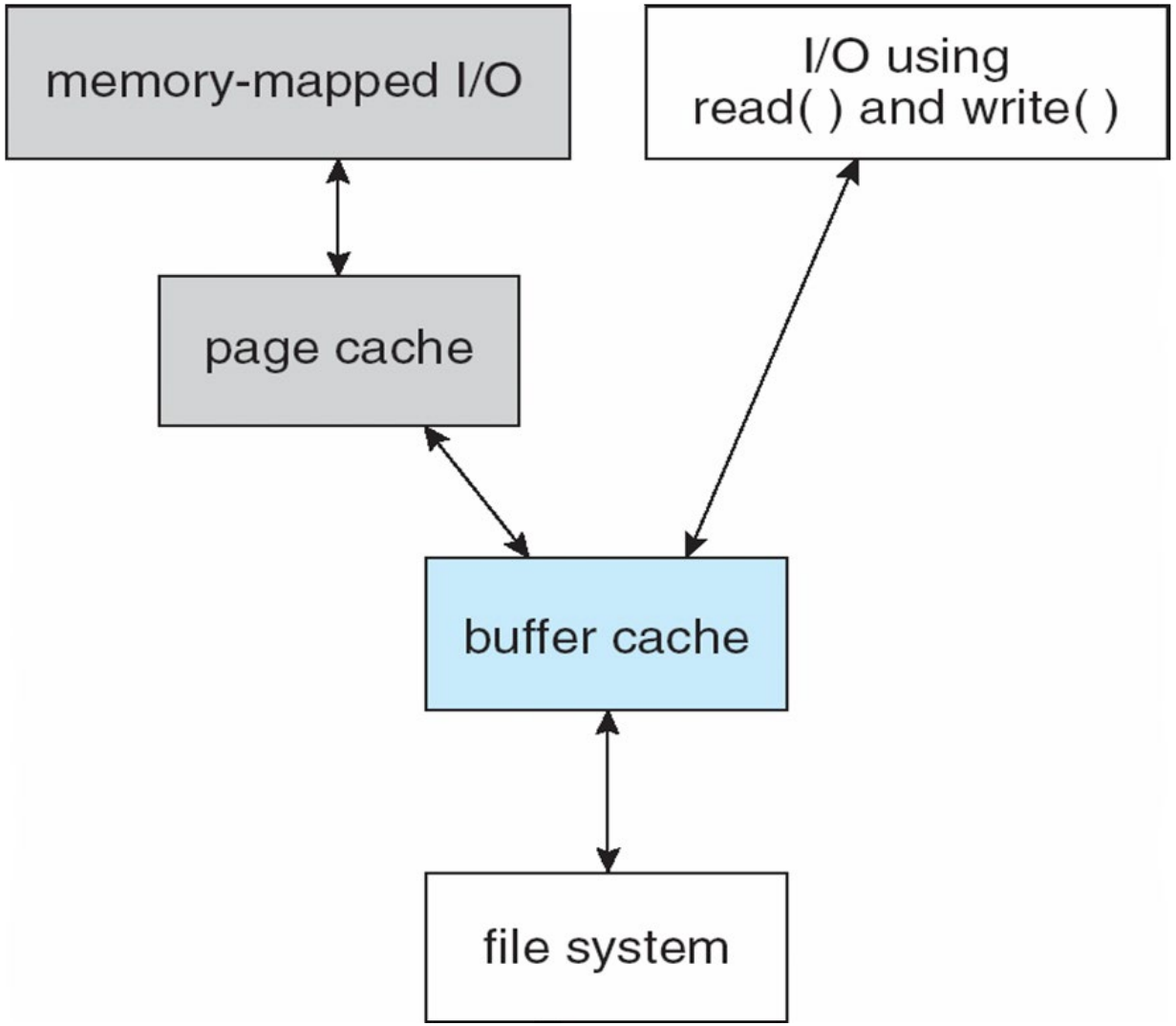
Efficiency and Performance

- Efficiency dependent on:
  - disk allocation and directory algorithms
  - types of data kept in file's directory entry
- Performance
  - disk cache – separate section of main memory for frequently used blocks
  - free-behind and read-ahead – techniques to optimize sequential access
  - improve PC performance by dedicating section of memory as virtual disk, or RAM disk

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

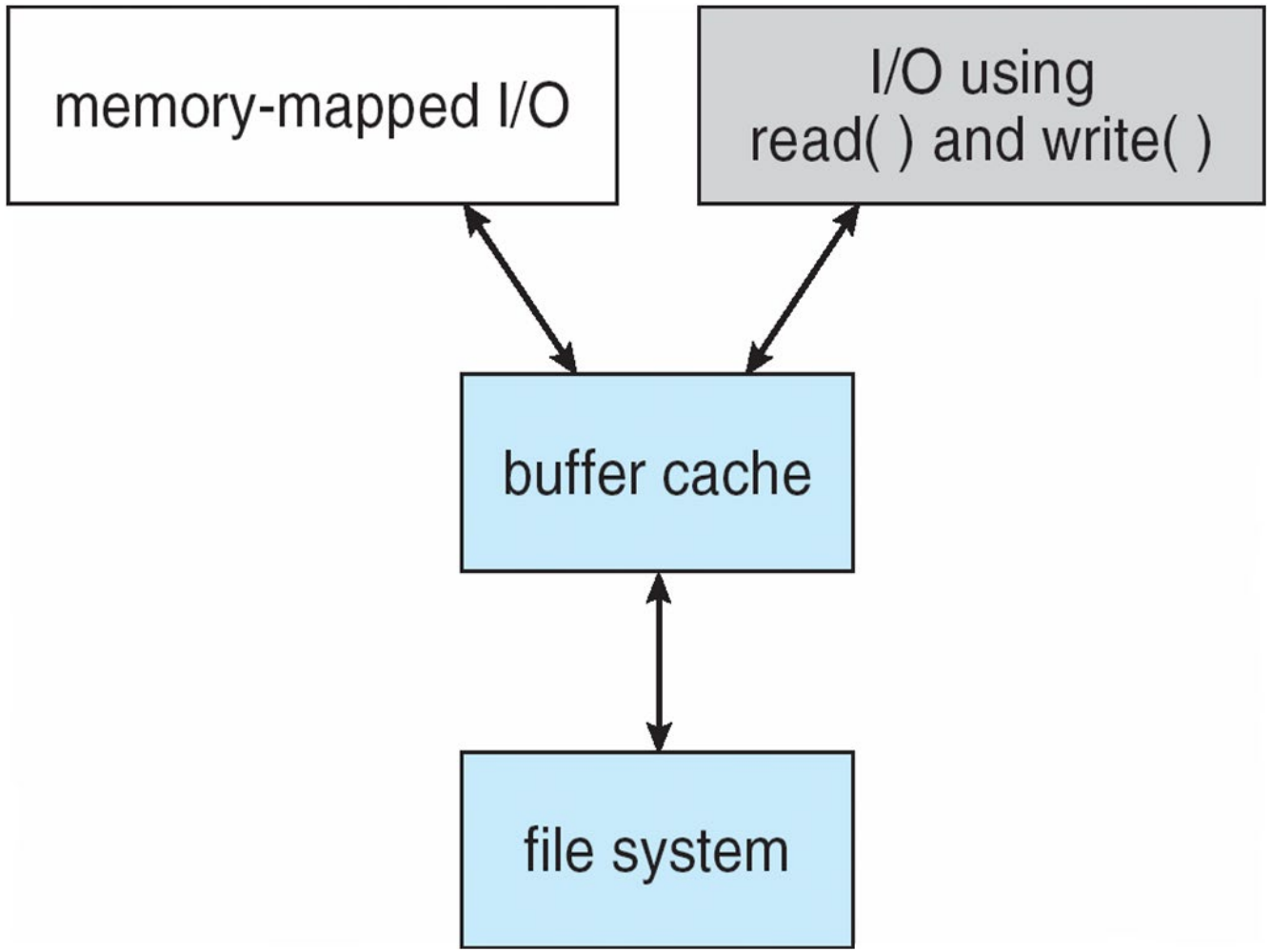
I/O Without a Unified Buffer Cache



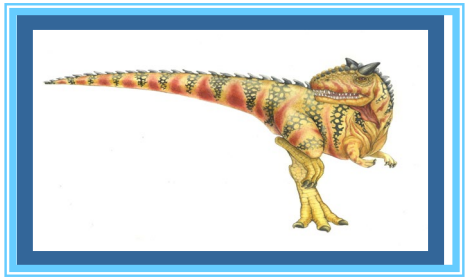
Unified Buffer Cache

- A unified buffer cache uses the same page cache to cache both memory-mapped pages and ordinary file system I/O

I/O Using a Unified Buffer Cache



11.7 Recovery



Recovery

- Consistency checking – compares data in directory structure with data blocks on disk, and tries to fix inconsistencies
- 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

11.8 Log Structured File Systems

- Log structured (or journaling) file systems record each update to the file system as a transaction
- 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

*11.9 The Sun Network File System (NFS)

- An implementation and a specification of a software system for accessing remote files across LANs (or WANs)
- The implementation is part of the Solaris and SunOS operating systems running on Sun workstations using an unreliable datagram protocol (UDP/IP protocol and Ethernet

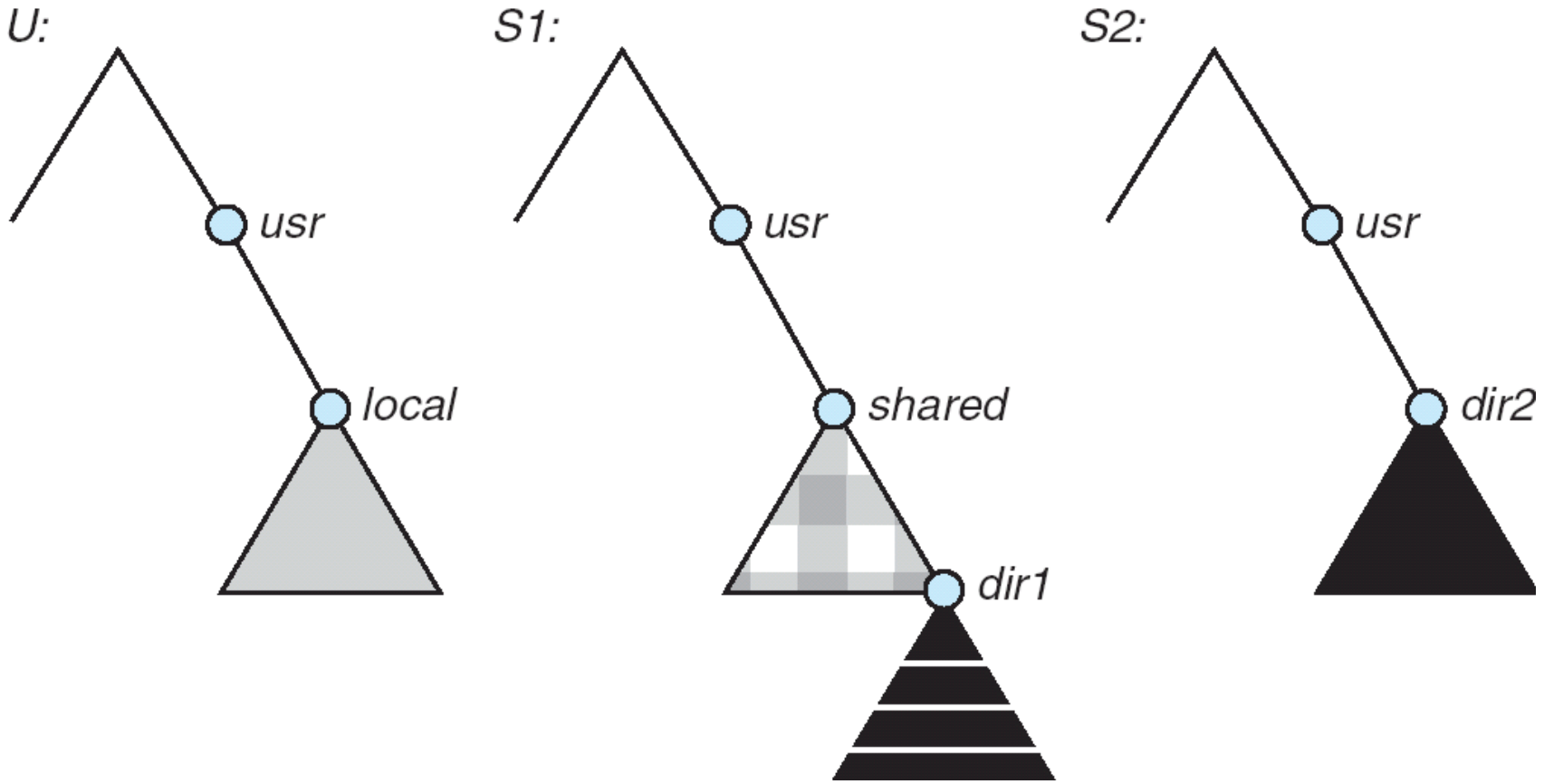
NFS (Cont.)

- Interconnected workstations viewed as a set of independent machines with independent file systems, which allows sharing among these file systems in a transparent manner
  - A remote directory is mounted over a local file system directory
    - The mounted directory looks like an integral subtree of the local file system, replacing the subtree descending from the local directory
  - Specification of the remote directory for the mount operation is nontransparent; the host name of the remote directory has to be provided
    - Files in the remote directory can then be accessed in a transparent manner
  - Subject to access-rights accreditation, potentially any file system (or directory within a file system), can be mounted remotely on top of any local directory

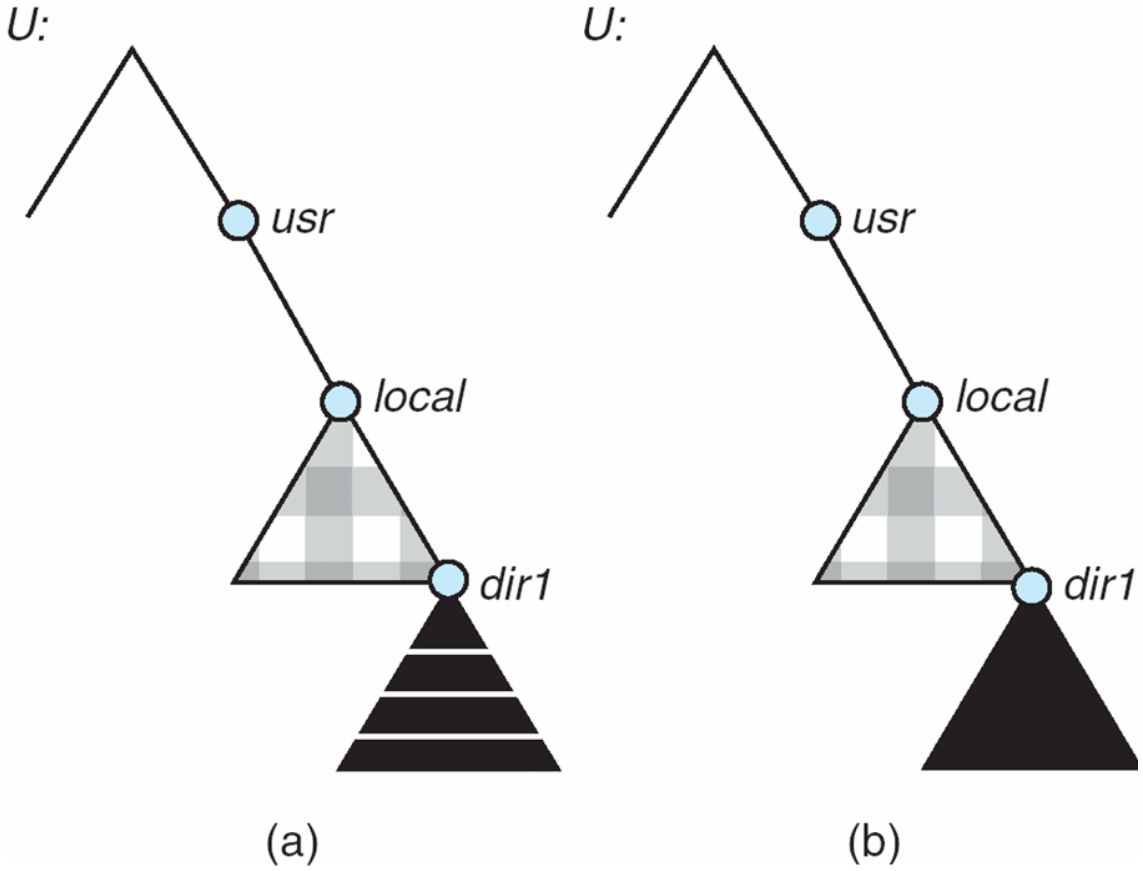
NFS (Cont.)

- NFS is designed to operate in a heterogeneous environment of different machines, operating systems, and network architectures; the NFS specifications independent of these media
- This independence is achieved through the use of RPC primitives built on top of an External Data Representation (XDR) protocol used between two implementation-independent interfaces
- The NFS specification distinguishes between the services provided by a mount mechanism and the actual remote-file-access services

Three Independent File Systems



Mounting in NFS



Mounts

Cascading mounts

NFS Mount Protocol

- Establishes initial logical connection between server and client
- Mount operation includes name of remote directory to be mounted and name of server machine storing it
  - Mount request is mapped to corresponding RPC and forwarded to mount server running on server machine
  - Export list – specifies local file systems that server exports for mounting, along with names of machines that are permitted to mount them
- Following a mount request that conforms to its export list, the server returns a file handle—a key for further accesses
- File handle – a file-system identifier, and an inode number to identify the mounted directory within the exported file system
- The mount operation changes only the user's view and does not affect the server side

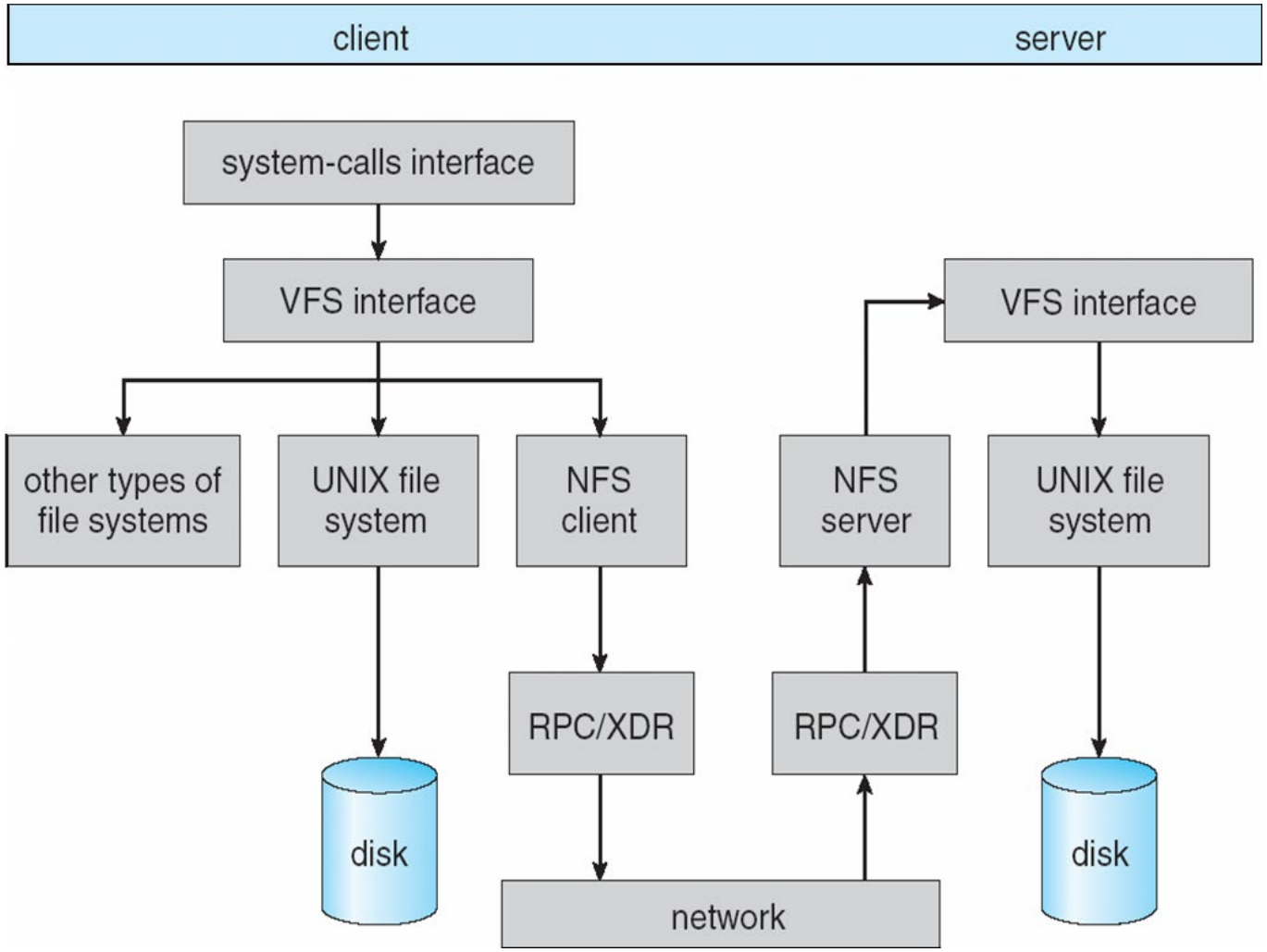
NFS Protocol

- Provides a set of remote procedure calls for remote file operations. The procedures support the following operations:
  - searching for a file within a directory
  - reading a set of directory entries
  - manipulating links and directories
  - accessing file attributes
  - reading and writing files
- NFS servers are **stateless**; each request has to provide a full set of arguments
  (NFS V4 is just coming available – very different, stateful)
- Modified data must be committed to the server's disk before results are returned to the client (lose advantages of caching)
- The NFS protocol does not provide concurrency-control mechanisms

Three Major Layers of NFS Architecture

- UNIX file-system interface (based on the **open, read, write**, and **close** calls, and **file descriptors**)
- *Virtual File System* (VFS) layer – distinguishes local files from remote ones, and local files are further distinguished according to their file-system types
  - The VFS activates file-system-specific operations to handle local requests according to their file-system types
  - Calls the NFS protocol procedures for remote requests
- NFS service layer – bottom layer of the architecture
  - Implements the NFS protocol

Schematic View of NFS Architecture



NFS Path-Name Translation

- Performed by breaking the path into component names and performing a separate NFS lookup call for every pair of component name and directory vnode

- To make lookup faster, a directory name lookup cache on the client's side holds the vnodes for remote directory names

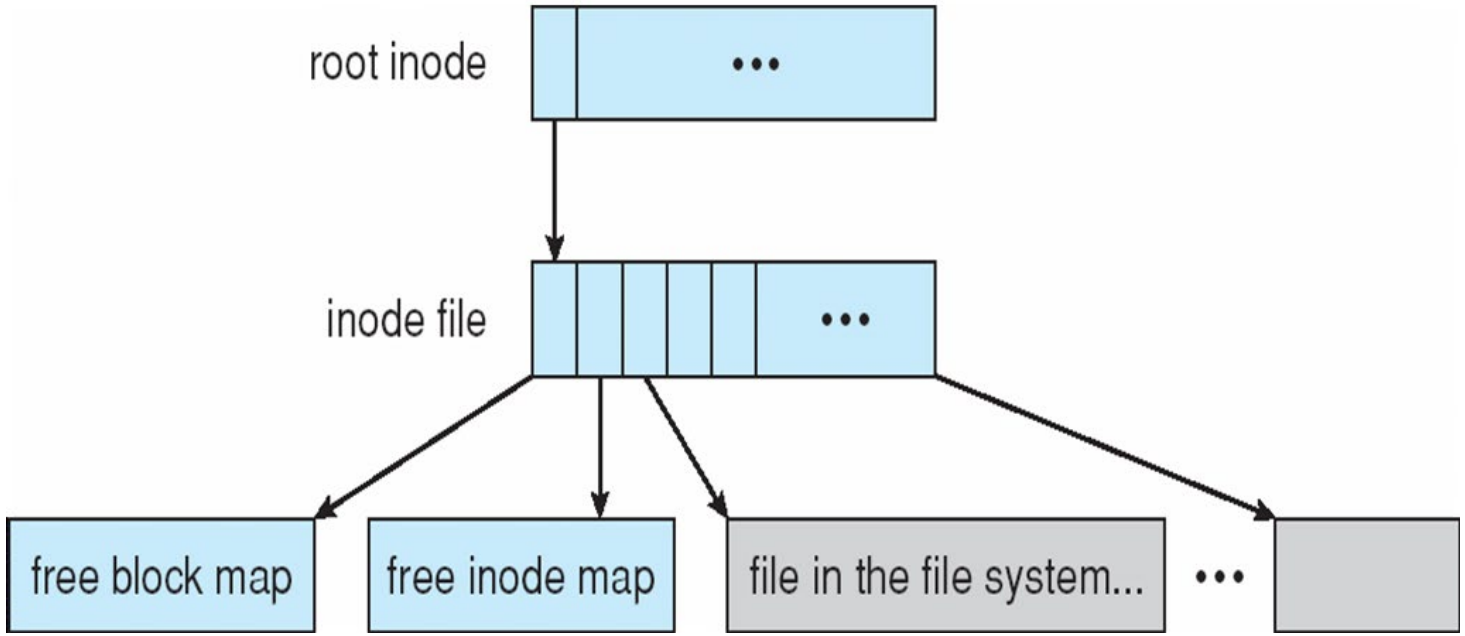
NFS Remote Operations

- Nearly one-to-one correspondence between regular UNIX system calls and the NFS protocol RPCs (except opening and closing files)
- NFS adheres to the remote-service paradigm, but employs buffering and caching techniques for the sake of performance
- File-blocks cache – when a file is opened, the kernel checks with the remote server whether to fetch or revalidate the cached attributes
  - Cached file blocks are used only if the corresponding cached attributes are up to date
- File-attribute cache – the attribute cache is updated whenever new attributes arrive from the server
- Clients do not free delayed-write blocks until the server confirms that the data have been written to disk

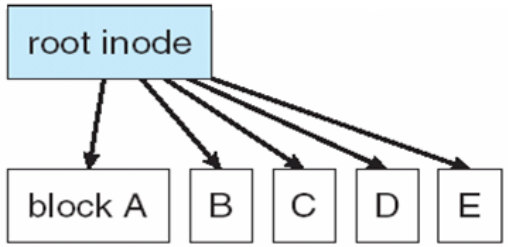
*11.10 Example: WAFL File System

- Used on Network Appliance “Filers” – distributed file system appliances
- “Write-anywhere file layout”
- Serves up NFS, CIFS, http, ftp
- Random I/O optimized, write optimized
  - NVRAM for write caching
- Similar to Berkeley Fast File System, with extensive modifications

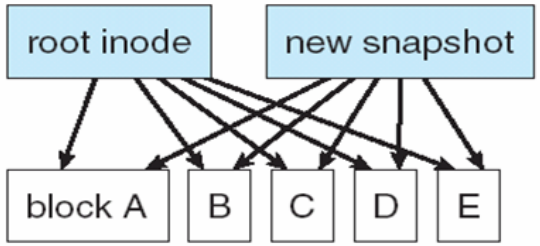
The WAFL File Layout



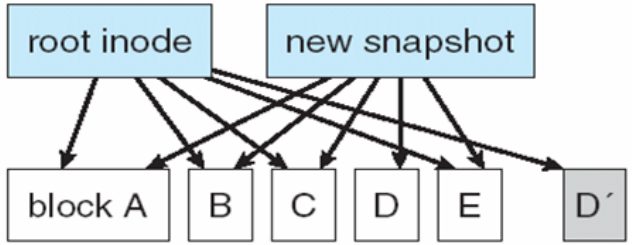
Snapshots in WAFL



(a) Before a snapshot.



(b) After a snapshot, before any blocks change.



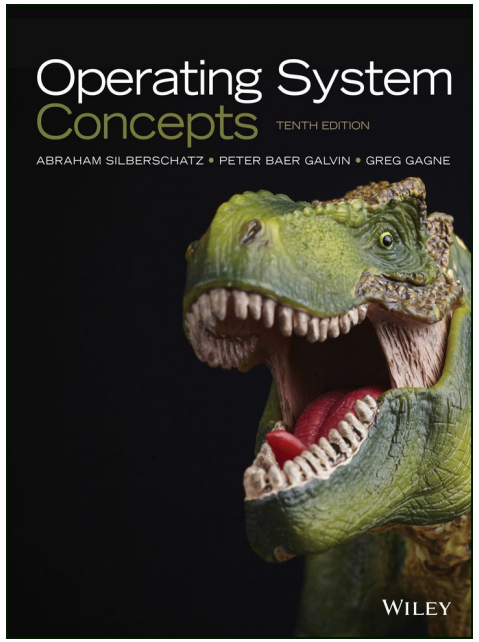
(c) After block D has changed to D´.

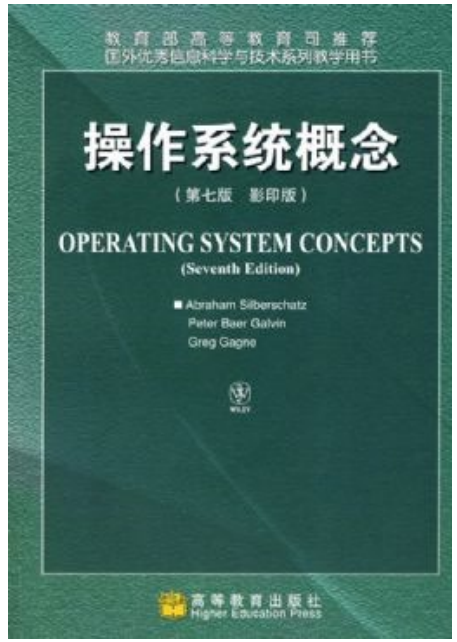
HOMEWORK

- 学在浙大
- 习题分析

Reading Assignments

- Read for this week:
  - Chapters 11
    of the text book:
- Read for next week:
  - Chapters 12
    of the text book:





End of Chapter 11