Operating System Principles

操作系统原理

File System

李旭东

leexudong@nankai.edu.cn

Nankai University

Long-Term Information Storage

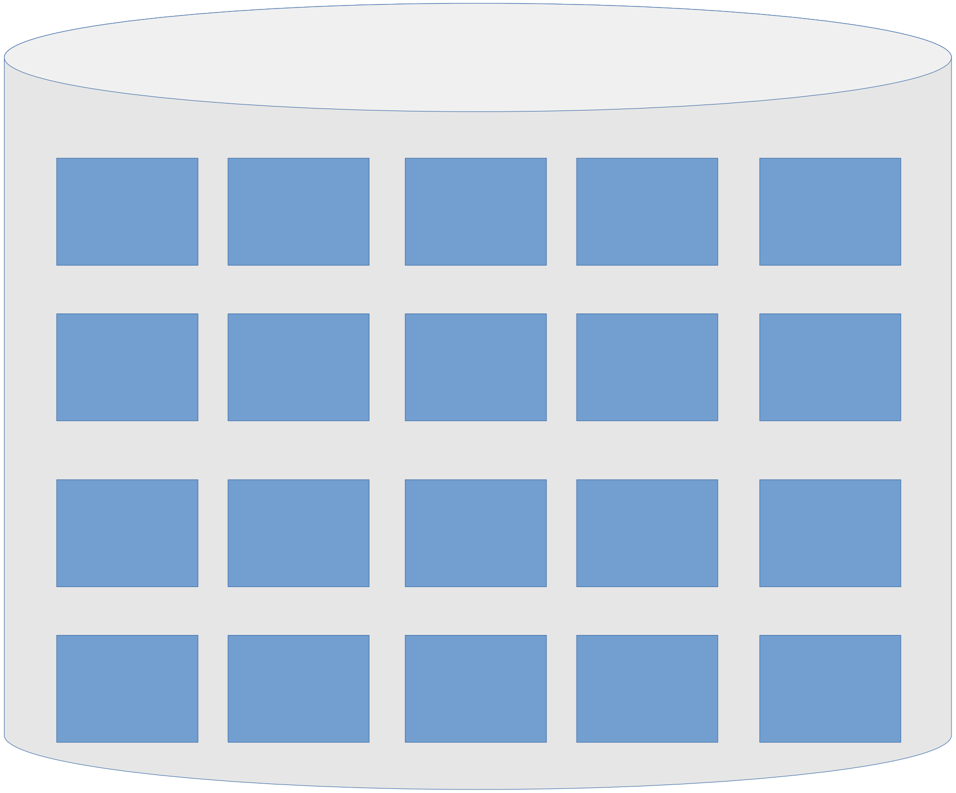
- Essential requirements
  - It must be possible to store a very large amount of information
  - The information must survive the termination of the process using it
  - Multiple processes must be able to access the information concurrently
- Solution
  - ???

Long-Term Information Storage

- Solution
  - Store information on disk and other external media in units called files.
  - Processes can then read them and write new one if need be
  - Persistent
  - File, Directory, File System

Long-Term Information Storage

- Storage with Block Device
  - Magnetic Disks
  - Magnetic Tape
  - Optical disc
  - Flash memory
- Block-Level Operations
  - Read i-th Block
  - Write i-th Block



Long-Term Information Storage

- Retrieve Information Quickly
  - 1. How do you find information?
  - 2. How do you keep one user from reading another user's data?
  - 3. How do you know which blocks are free?

Objectives

- File Concept
- Directory Concept
- File Share & Protection
- File System Implementation
- File System Reliability
- File System Performance
- File System Cases

File Concept

- File
  - A logical units of information
  - A byte stream
- File Size
- File Name
- File Logical Structure (File Content)
- File Type
- File Access
- File Attributes
- File Operations
- File Physical Structure

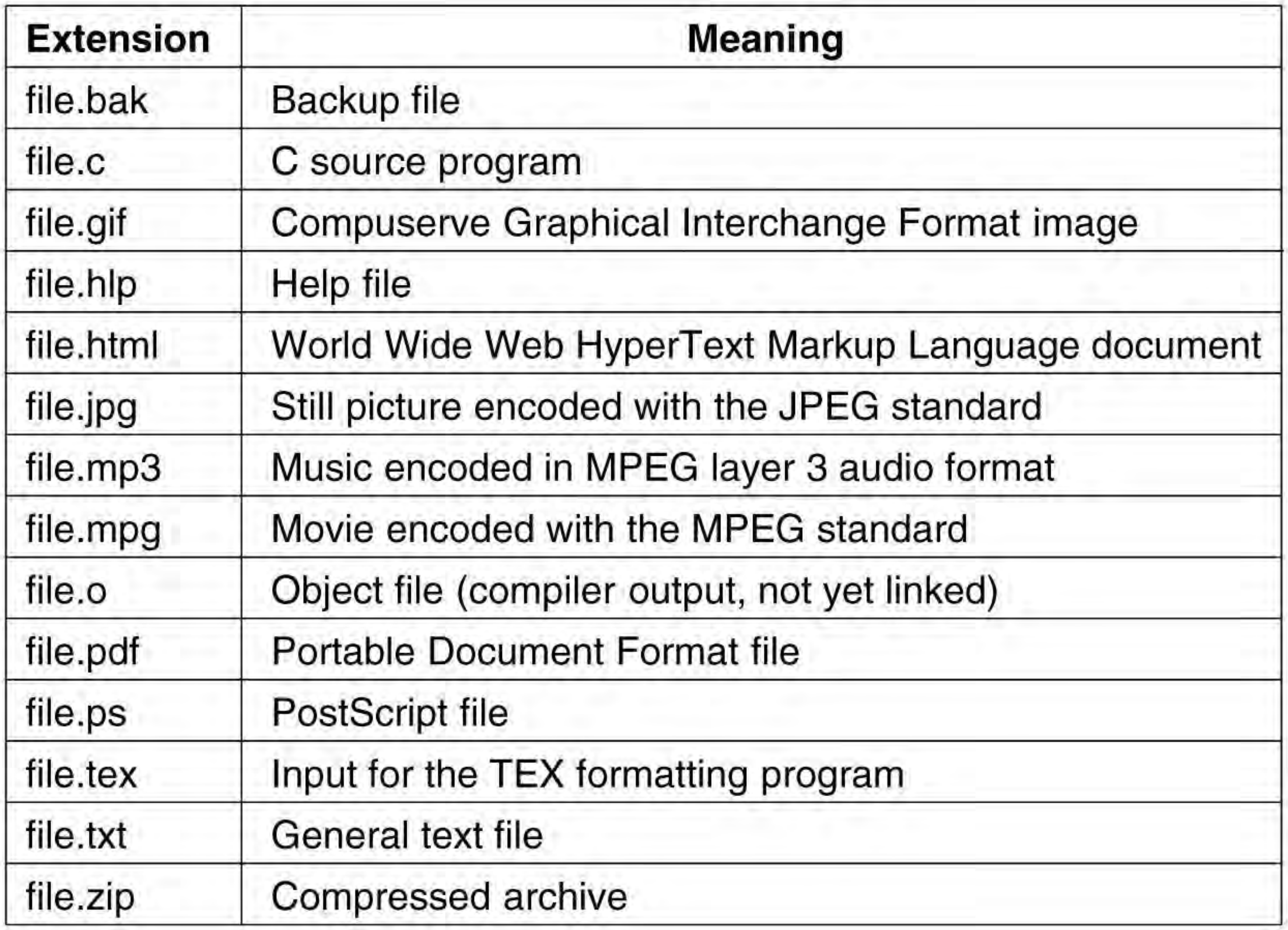
File Size

- Zero Byte
- ~~1 Bit~~
- 1 Byte
- 2 Bytes
- 1 KB（千）
- 1 MB（兆）
- …

- GB（吉）
- TB（太）
- PB（拍）
- EB（艾）
- ZB（泽）
  - 1.2ZB
- YB（尧）
- …

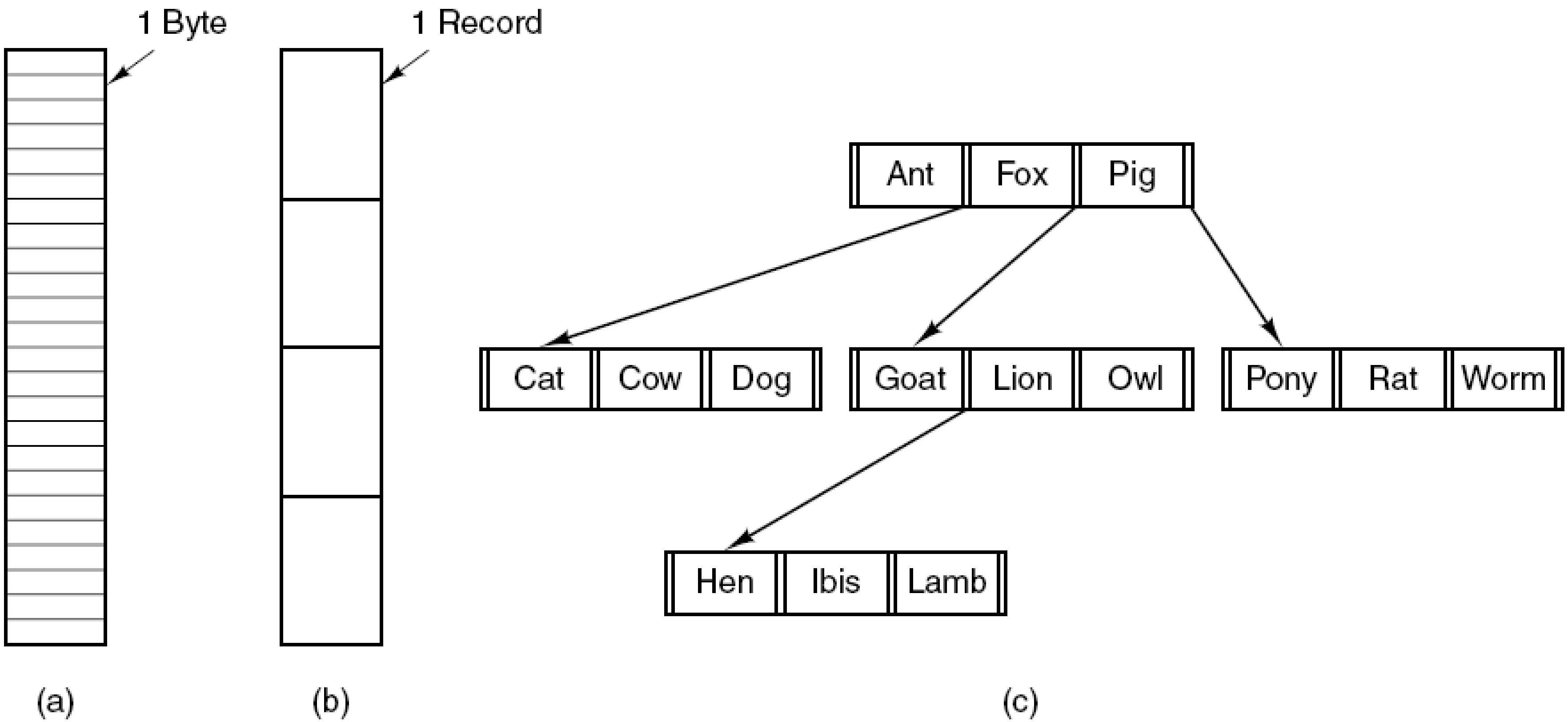
File Naming

- File extension
- File Name Charset



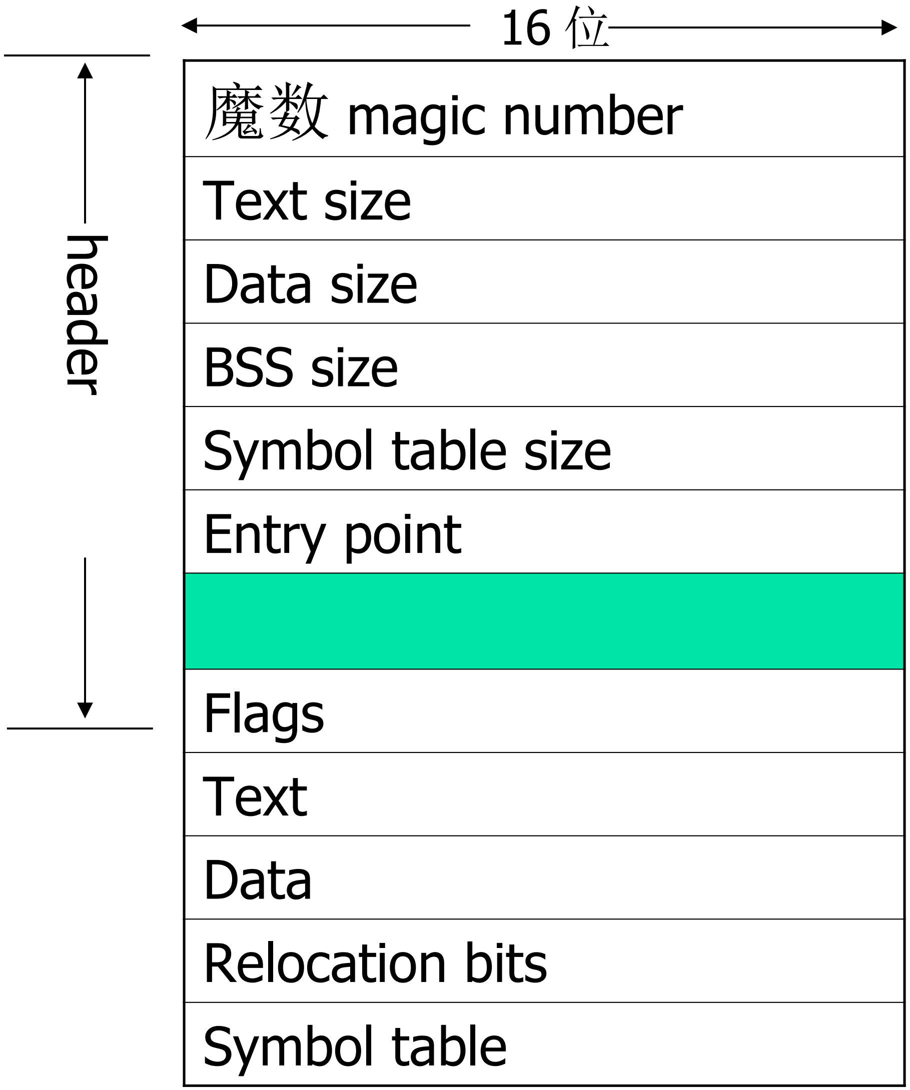
Extension	Meaning
file.bak	Backup file
file.c	C source program
file.gif	Compuserve Graphical Interchange Format image
file.hlp	Help file
file.html	World Wide Web HyperText Markup Language document
file.jpg	Still picture encoded with the JPEG standard
file.mp3	Music encoded in MPEG layer 3 audio format
file.mpg	Movie encoded with the MPEG standard
file.o	Object file (compiler output, not yet linked)
file.pdf	Portable Document Format file
file.ps	PostScript file
file.tex	Input for the TEX formatting program
file.txt	General text file
file.zip	Compressed archive

File Logical Structure



(a) Byte sequence. (b) Record sequence. (c) Tree.

File Logical Structure:
Executable Program File



16 位

header

魔数 magic number

Text size

Data size

BSS size

Symbol table size

Entry point

Flags

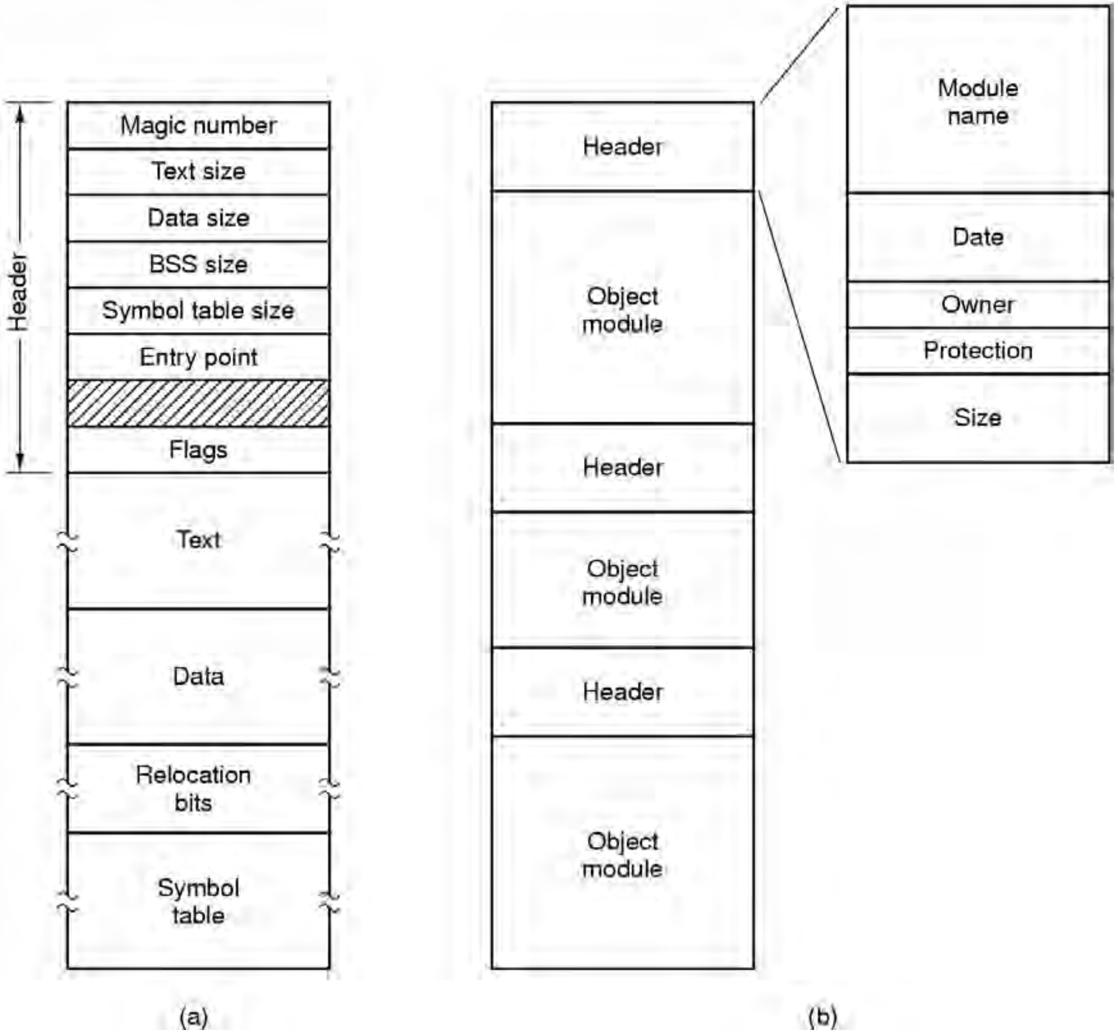
Text

Data

Relocation bits

Symbol table

File Types



(a) An executable file. (b) An archive.

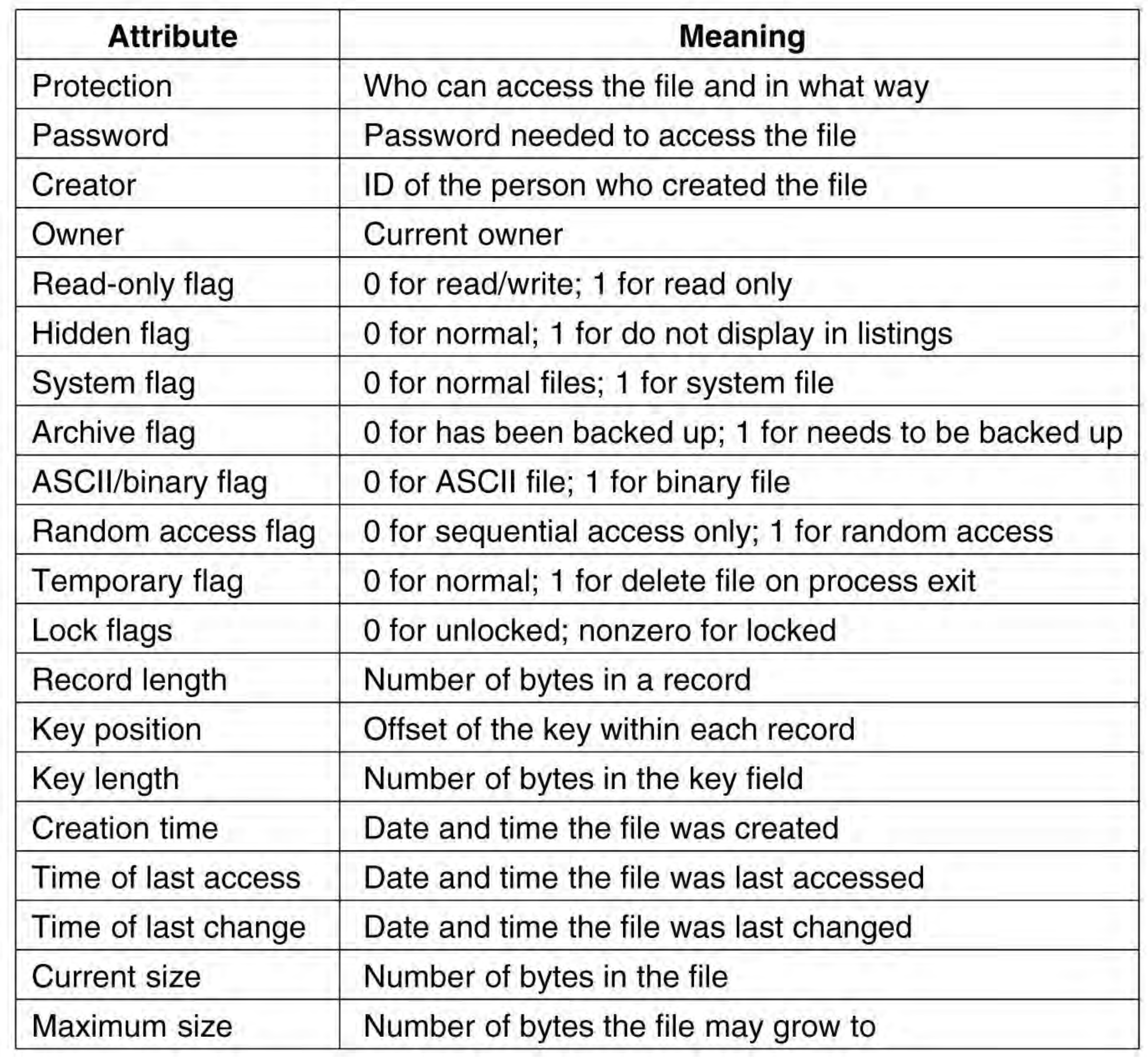
File Types

- T1
  - Regular files
  - Directory files
- T2
  - Character special files
  - Block speicial files
- T3
  - ASCII files
  - Binary files
- ...

File Access

- Sequential access
- Random access
- Key-value access

File Attributes



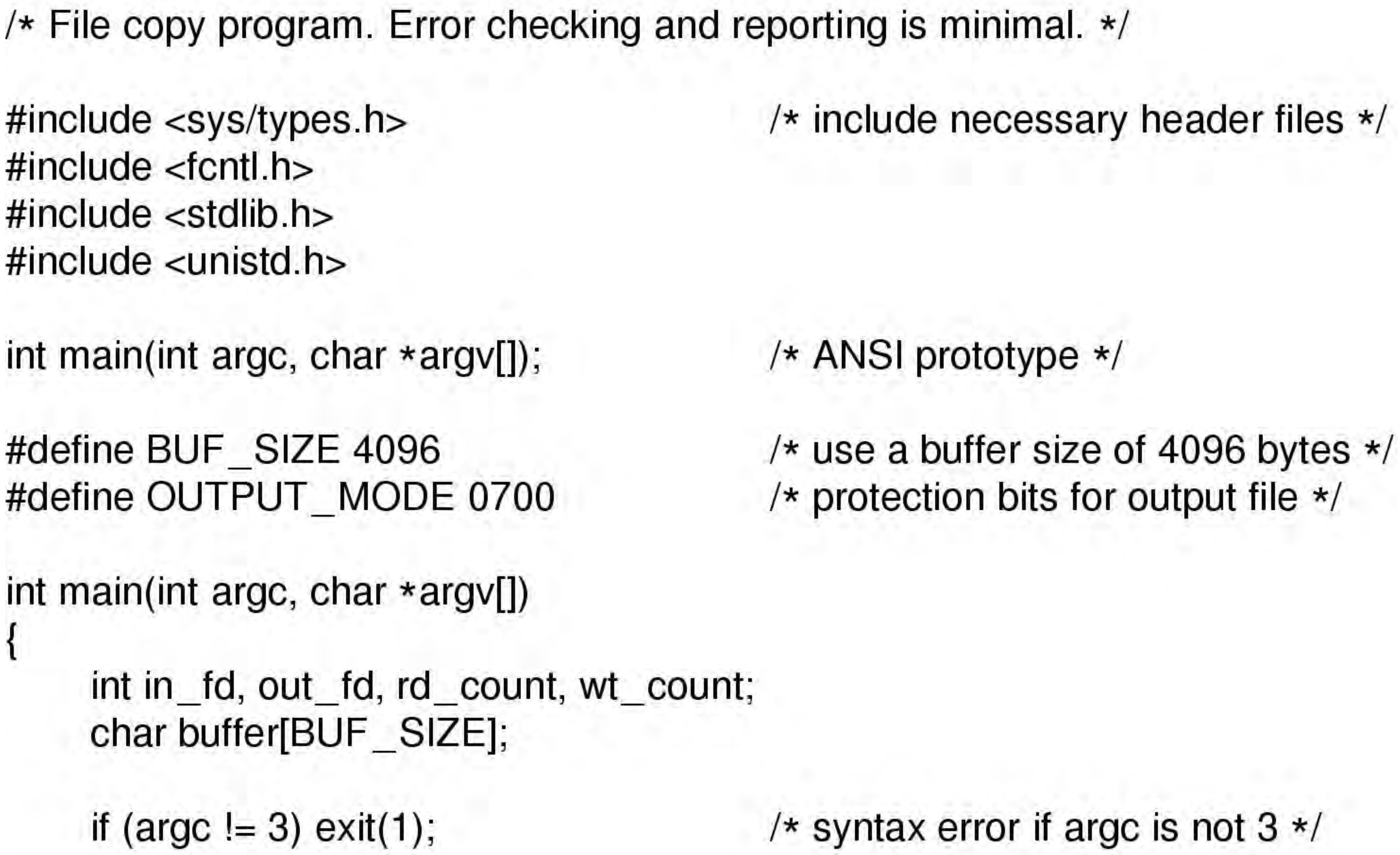
Attribute	Meaning
Protection	Who can access the file and in what way
Password	Password needed to access the file
Creator	ID of the person who created the file
Owner	Current owner
Read-only flag	0 for read/write; 1 for read only
Hidden flag	0 for normal; 1 for do not display in listings
System flag	0 for normal files; 1 for system file
Archive flag	0 for has been backed up; 1 for needs to be backed up
ASCII/binary flag	0 for ASCII file; 1 for binary file
Random access flag	0 for sequential access only; 1 for random access
Temporary flag	0 for normal; 1 for delete file on process exit
Lock flags	0 for unlocked; nonzero for locked
Record length	Number of bytes in a record
Key position	Offset of the key within each record
Key length	Number of bytes in the key field
Creation time	Date and time the file was created
Time of last access	Date and time the file was last accessed
Time of last change	Date and time the file was last changed
Current size	Number of bytes in the file
Maximum size	Number of bytes the file may grow to

File Operations

The most common system calls relating to files:

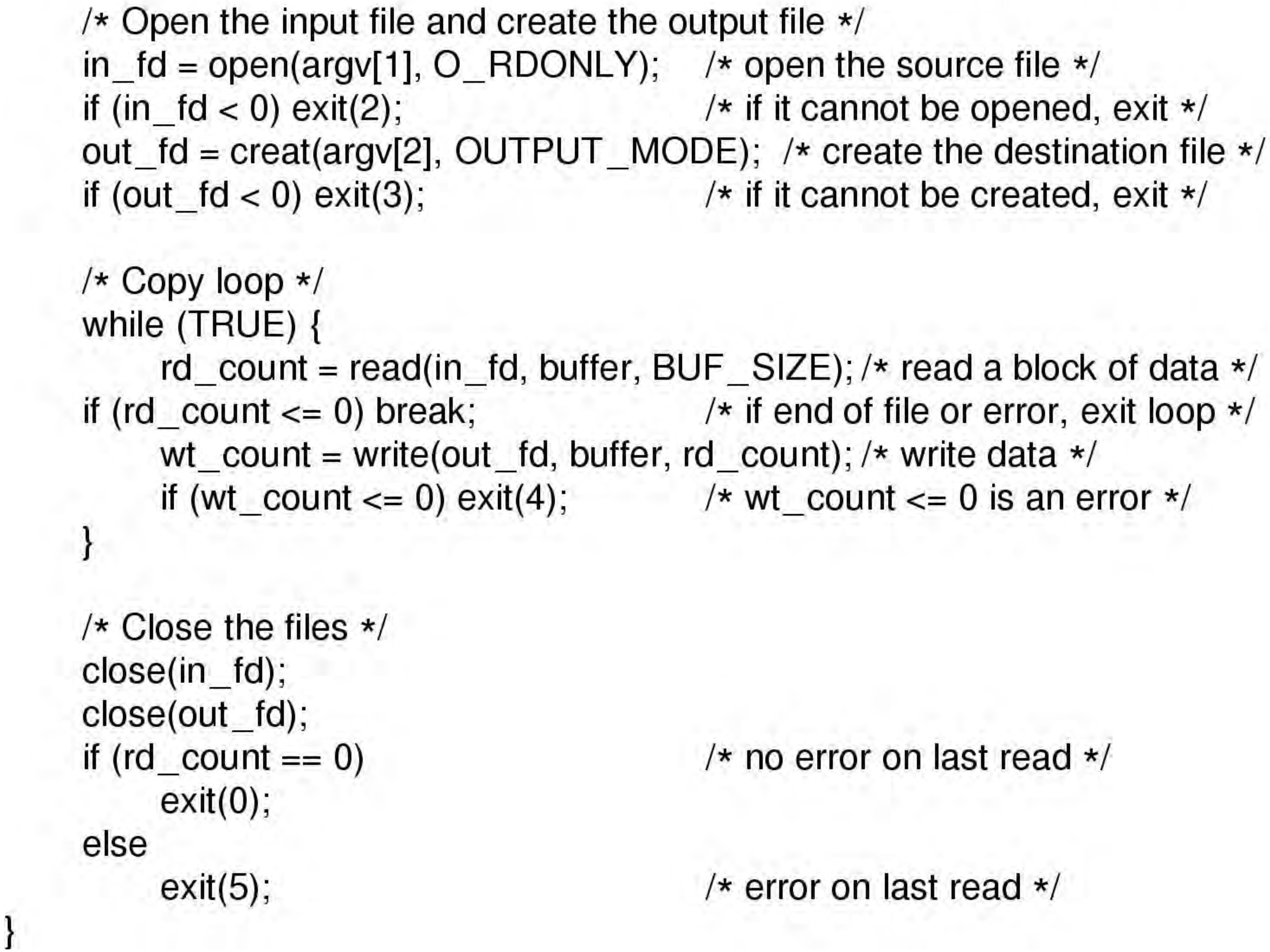
- Create
- Delete
- Open
- Close
- Read
- Write
- Append
- Seek
- Get Attributes
- Set Attributes
- Rename

CP Program:Using File System Calls (1/2)



```
/* File copy program. Error checking and reporting is minimal. */

#include <sys/types.h>                       /* include necessary header files */
#include <fcntl.h>
#include <stdlib.h>
#include <unistd.h>

int main(int argc, char *argv[]);            /* ANSI prototype */

#define BUF_SIZE 4096                        /* use a buffer size of 4096 bytes */
#define OUTPUT_MODE 0700                     /* protection bits for output file */

int main(int argc, char *argv[])
{
     int in_fd, out_fd, rd_count, wt_count;
     char buffer[BUF_SIZE];

     if (argc != 3) exit(1);                 /* syntax error if argc is not 3 */
```

CP Program:Using File System Calls (2/2)



```
/* Open the input file and create the output file */
in_fd = open(argv[1], O_RDONLY);     /* open the source file */
if (in_fd < 0) exit(2);               /* if it cannot be opened, exit */
out_fd = creat(argv[2], OUTPUT_MODE);  /* create the destination file */
if (out_fd < 0) exit(3);              /* if it cannot be created, exit */

/* Copy loop */
while (TRUE) {
     rd_count = read(in_fd, buffer, BUF_SIZE); /* read a block of data */
if (rd_count <= 0) break;             /* if end of file or error, exit loop */
     wt_count = write(out_fd, buffer, rd_count); /* write data */
     if (wt_count <= 0) exit(4);      /* wt_count <= 0 is an error */
}

/* Close the files */
close(in_fd);
close(out_fd);
if (rd_count == 0)                    /* no error on last read */
     exit(0);
else
     exit(5);                         /* error on last read */
}
```

File Physical Structure

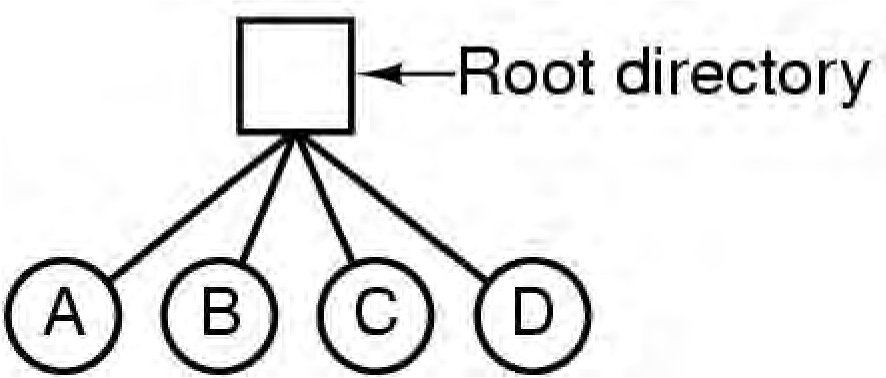
- How to store in block storage
  - ???

Directory Concept

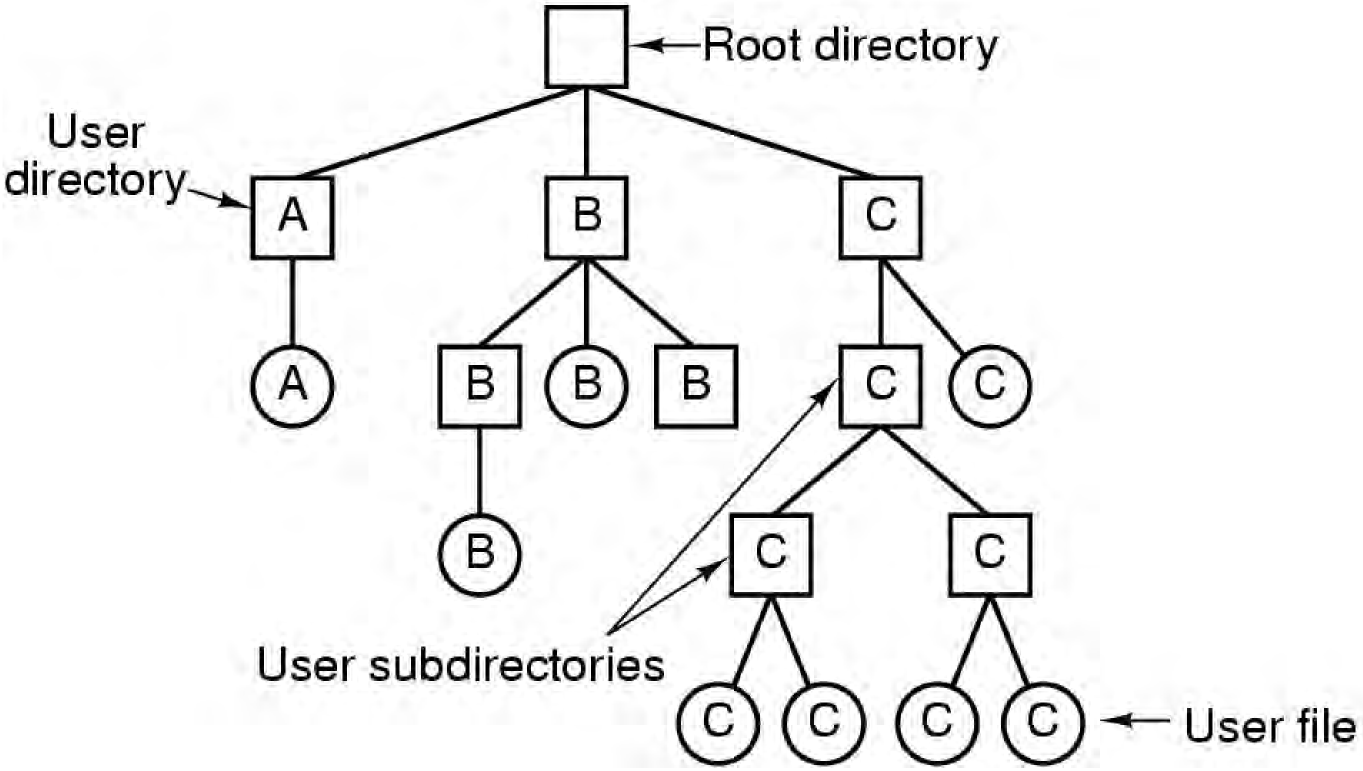
- Directory
    - Folder, Which contains files
- Single-level Directory System
- Hierarchical Directory System
- Path Names
- Directory Operations
- Implement of Directory

Single-level Directory System

Root directory

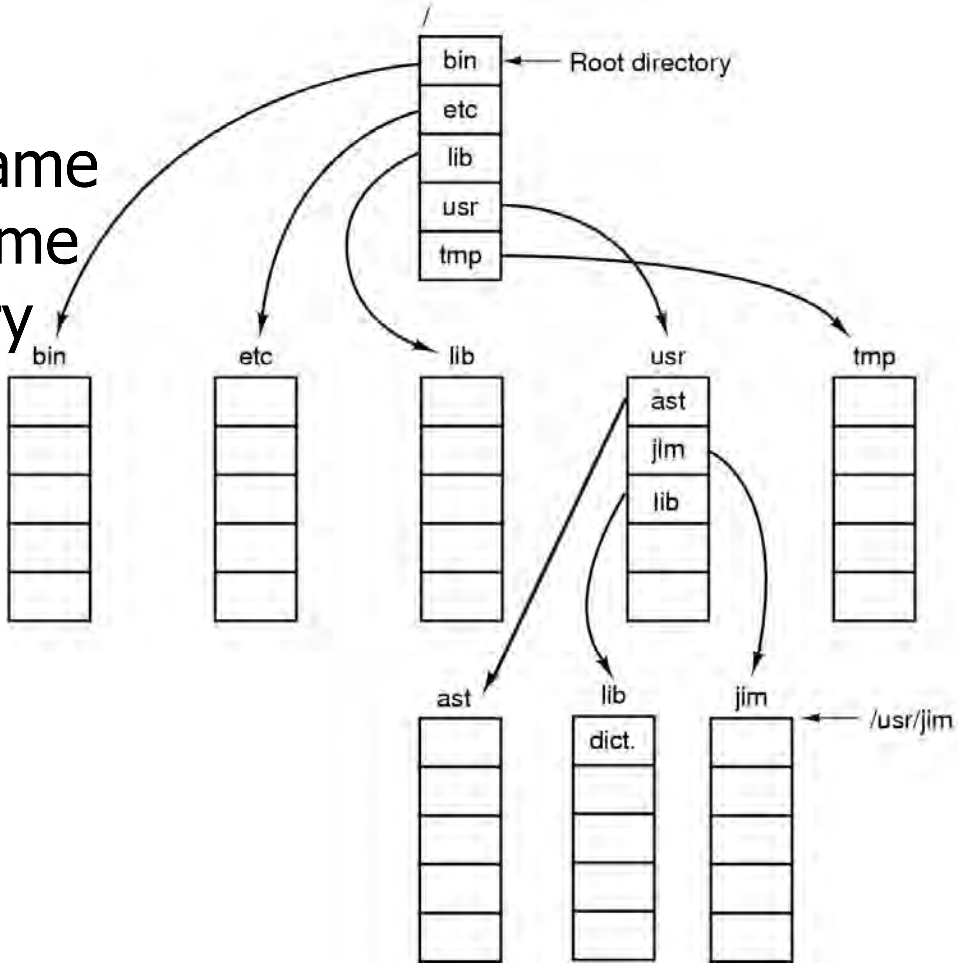


Hierarchical Directory System



Path Names

- Absolute path name
- Relative path name
- Working directory



A UNIX directory tree

Directory Oprations

System calls for managing directories:

- Create
- Delete
- Opendir
- Closedir
- Readdir
- Rename
- Link
- Unlink

- Link
  - Hard link
  - Symbolic link

Implement of Directory

- How to store in block storage
  - ???

File System Concept

- File System
  - It contains files and directories
- File System Types
- Implement of File System

File System Types

- Traditional: ext2
- Newest: ext3, ReiserFS, IBM JFS, xfs
- Other UNIX: minix, ext, xiafs
- FAT-12, FAT-16, FAT-32, VFAT, NTFS (read-only)
- HPFS (OS/2) read-only, HFS (Macintosh) read-only
- AFFS (Amiga), System V, Coherent, Xenix
- CD-ROM: **ISO 9660**
- UMSDOS (UNIX-like FS on MS-DOS)
- NFS (Network File System)
- SMBFS (Windows share), NCPFS (Novell Netware share)
- /proc (for kernel and process information)
- SHMFS (Shared Memory Filesystem)

File Systems Supported by Linux

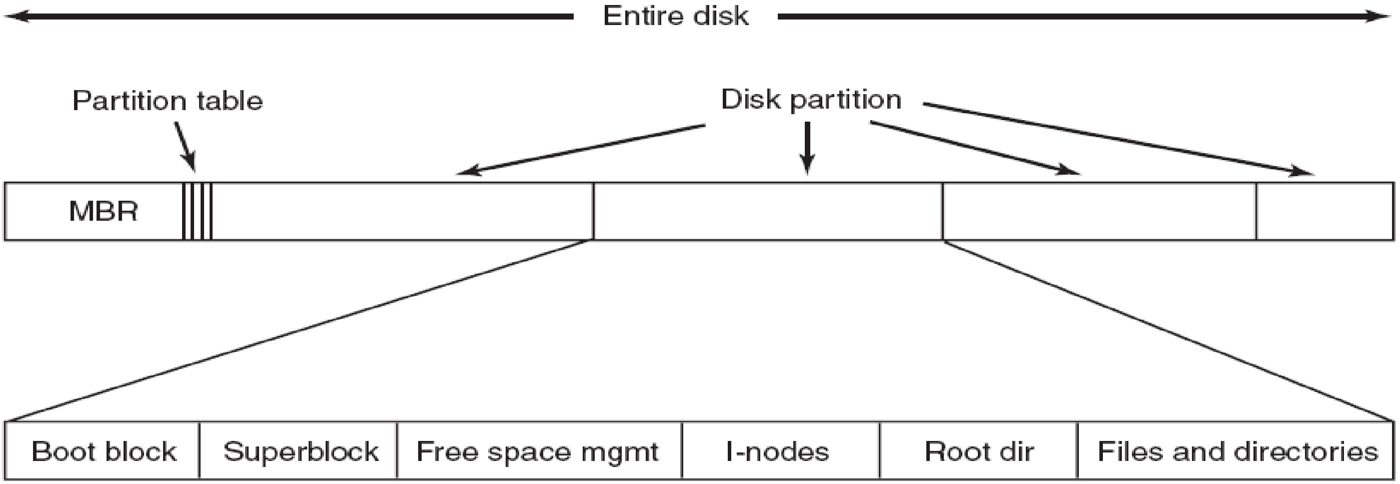
```
9p          configfs   freevxfs   jffs2              nls          smbfs
adfs        cramfs     fscache    jfs                notify       squashfs
affs        debugfs    fuse       Kconfig            ntfs         sysfs
afs         devpts     gfs2       Kconfig.binfmt     ocfs2        sysv
autofs      dlm        hfs        lockd              omfs         ubifs
autofs4     ecryptfs   hfsplus    logfs              openpromfs   udf
befs        efs        hostfs     Makefile           partitions   ufs
bfs         exofs      hpfs       minix              proc         xfs
btrfs       exportfs   hppfs      ncpfs              qnx4
cachefiles  ext2       hugetlbfs  nfs                quota
ceph        ext3       isofs      nfs_common         ramfs
cifs        ext4       jbd        nfsd               reiserfs
coda        fat        jbd2       nilfs2             romfs
```

Implement of File System

- File System Layout
- Implement of Files
- Implement of Directories
- Free Block Space Management

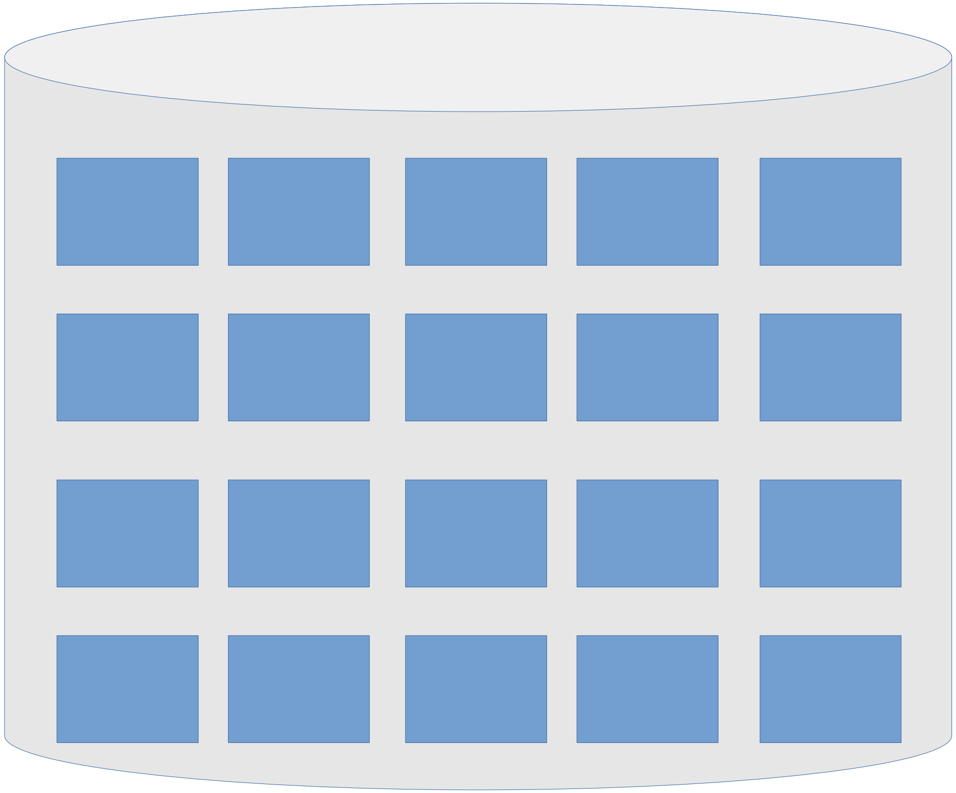
File System Layout

- Disk
- MBR (Master Boot Record)
  - Boot block
- Partitions
  - Superblock



Implement of Files

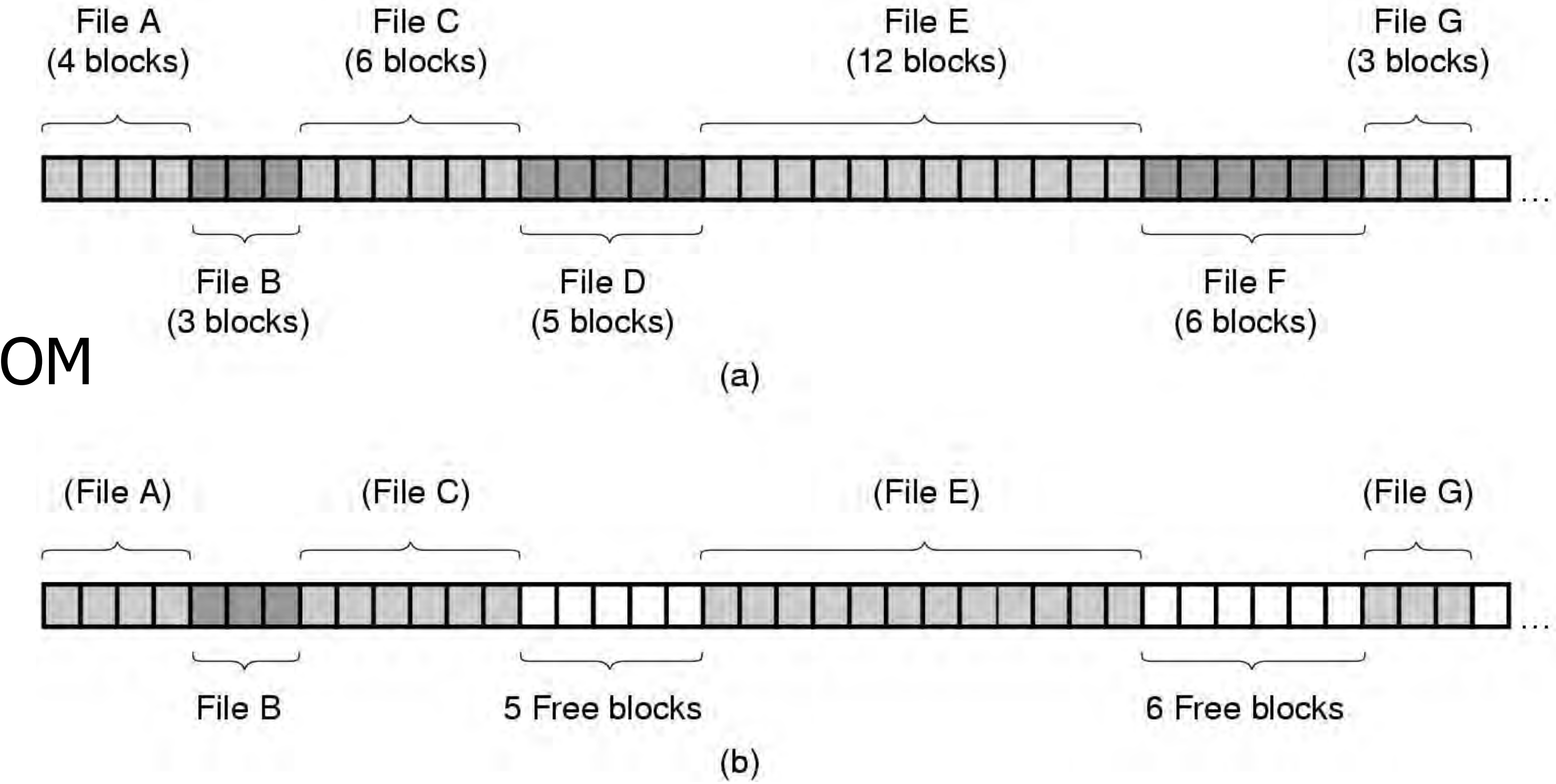
Where are File's Blocks?



Implement of Files

- Contiguous Allocation

CD-ROM

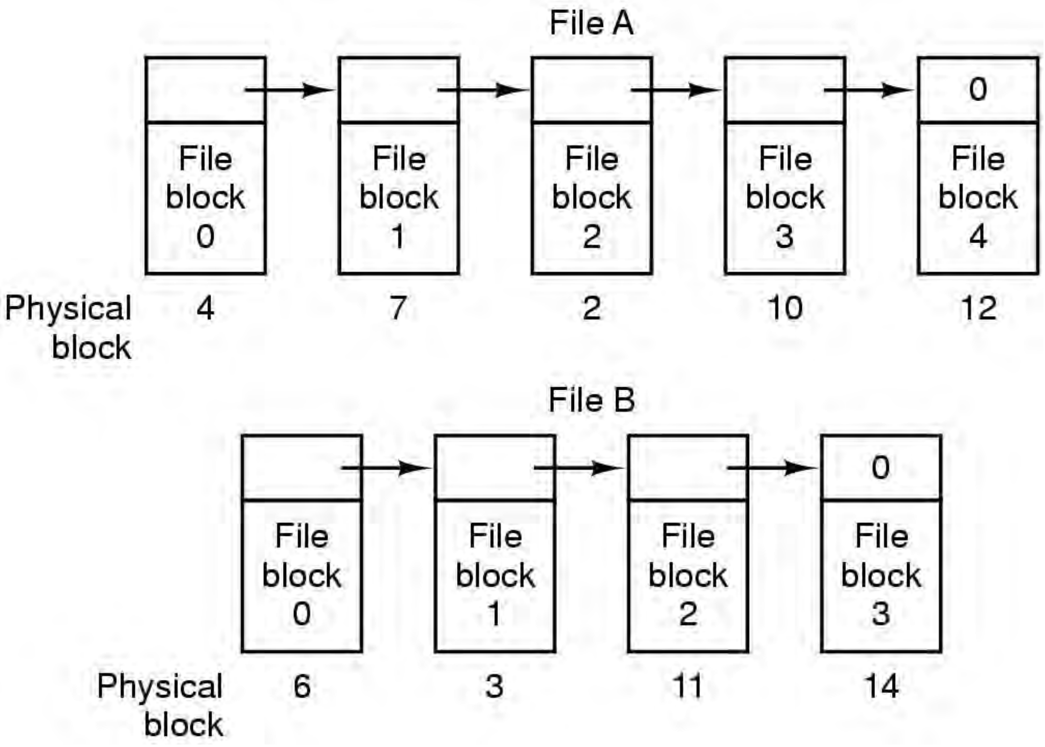


(a) Contiguous allocation of disk space for 7 files.

(b) The state of the disk after files D and F have been removed.

Implement of Files

- Linked List Allocation
  - The first word of each block is used as a pointer to the next one
  - The rest of the block is for data



- Internal fragmentation
- Random access is extremely slow

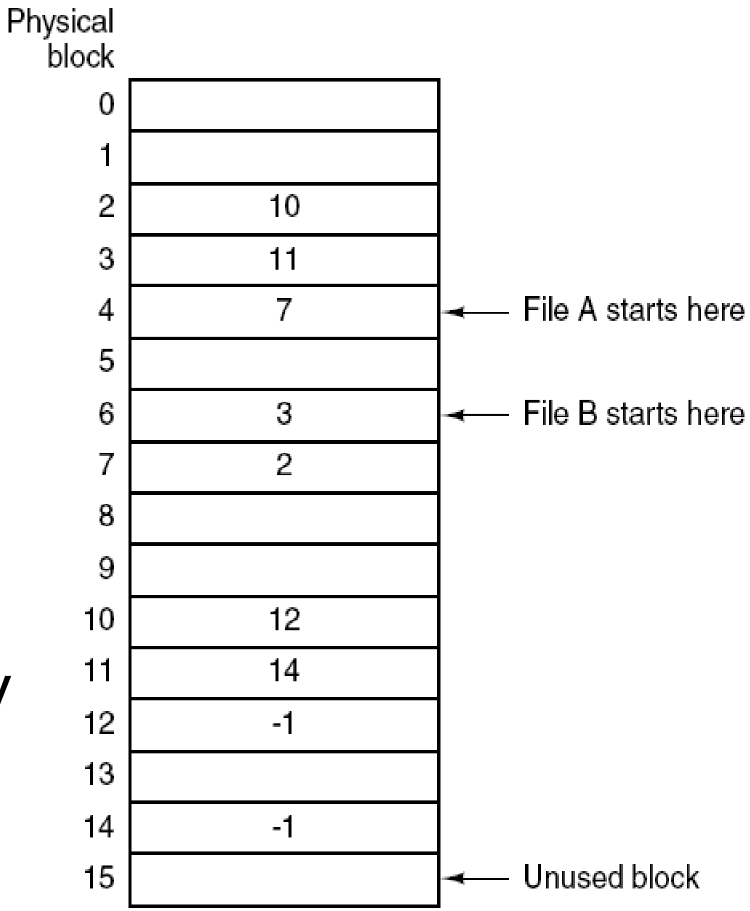
Storing a file as a linked list of disk blocks.

Implement of Files

- Linked List Allocation Using a Table in Memory
  - FAT: File Allocation Table
  - FAT12
  - FAT16
  - FAT32

- Random access is much easier

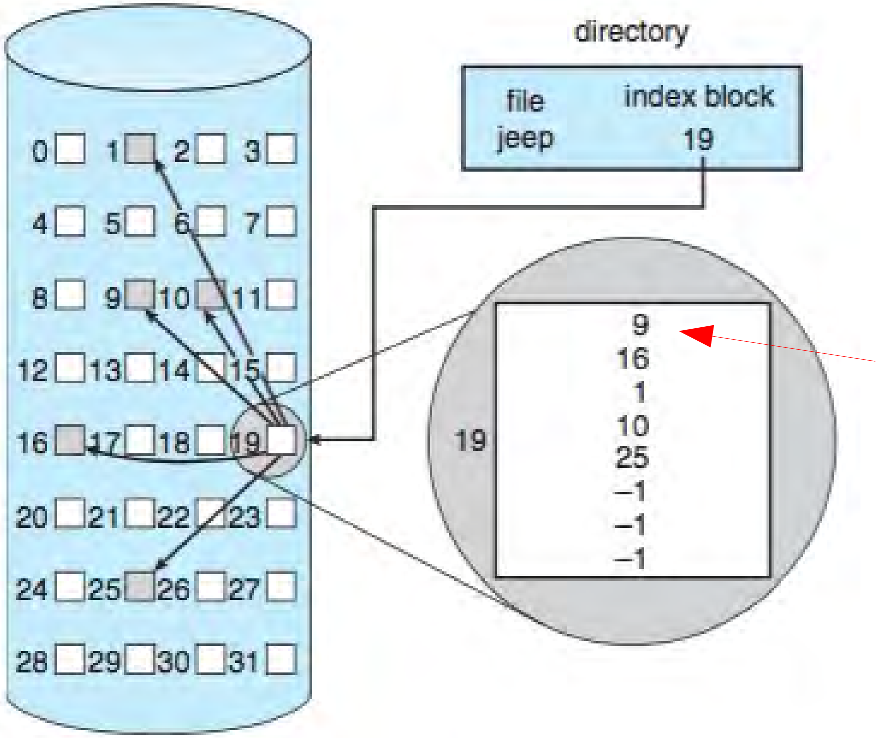
- ?Disadvantage
  - Too many table entries in memory



Physical block	
0	
1	
2	10
3	11
4	7
5	
6	3
7	2
8	
9	
10	12
11	14
12	-1
13	
14	-1
15	

Implement of Files

- I-nodes
  - Index-node, which lists the attributes and disk addresses of the files blocks

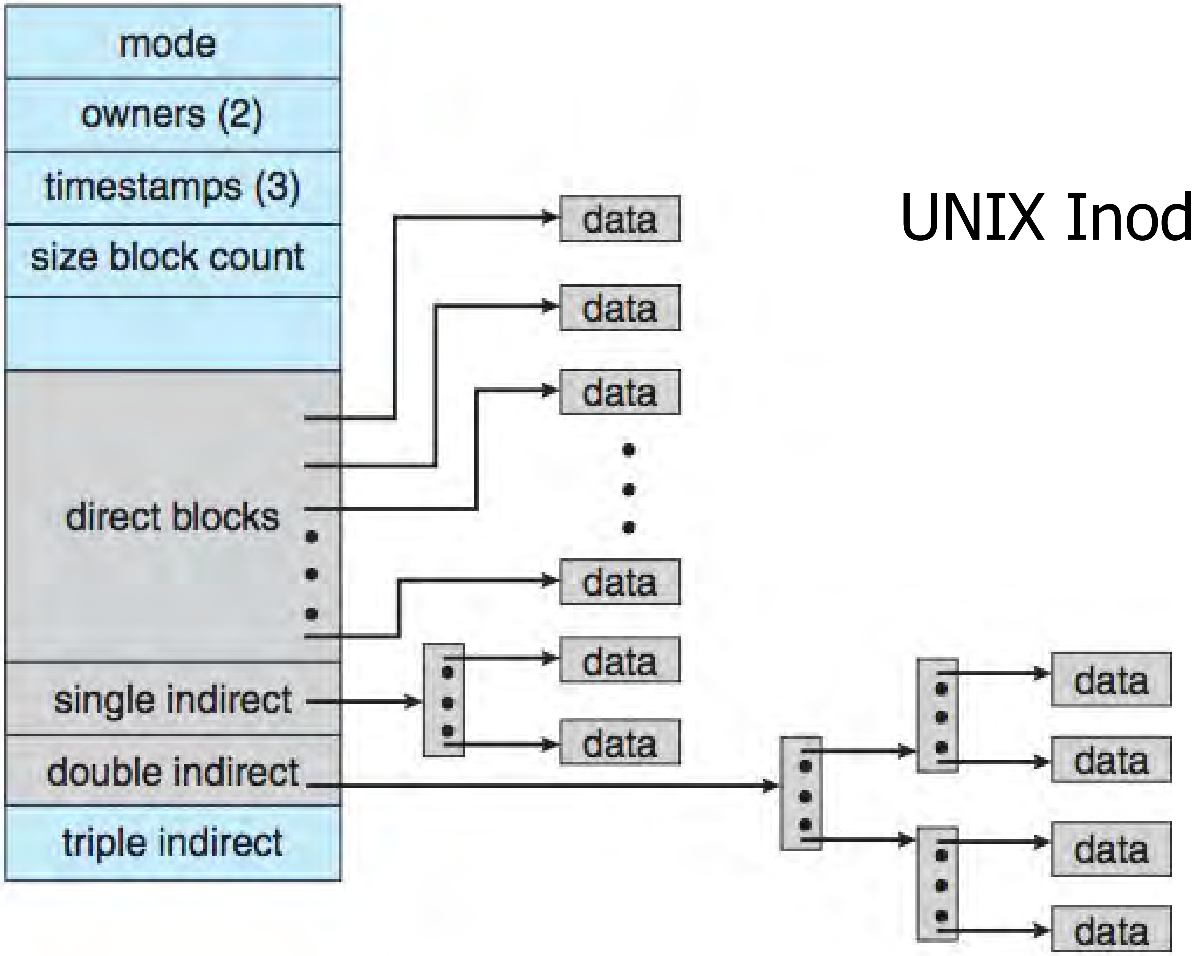


file	index block
jeep	19

9
16
1
10
25
–1
–1
–1

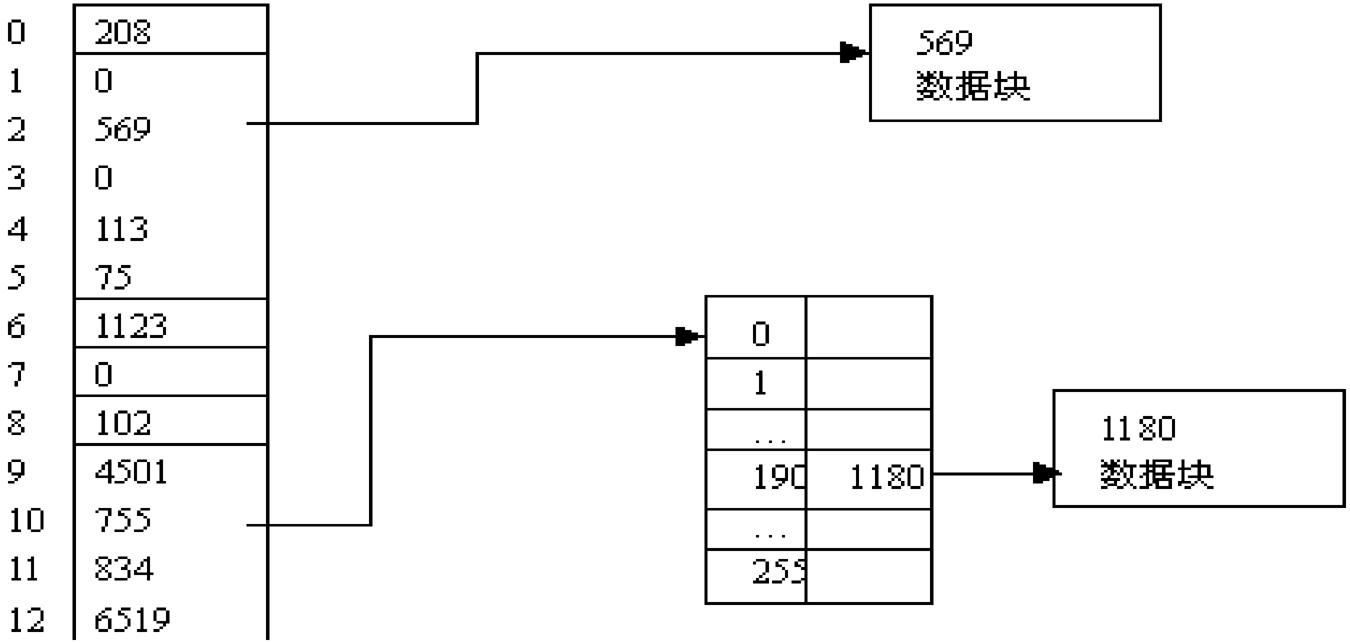
Index-block entry

I-Nodes



UNIX Inode

Quiz



0	208
1	0
2	569
3	0
4	113
5	75
6	1123
7	0
8	102
9	4501
10	755
11	834
12	6519

569
数据块

0	
1	
...	
190	1180
...	
255	

1180
数据块

How to read the 204803th data of file A, which physical block index has 4 bytes?

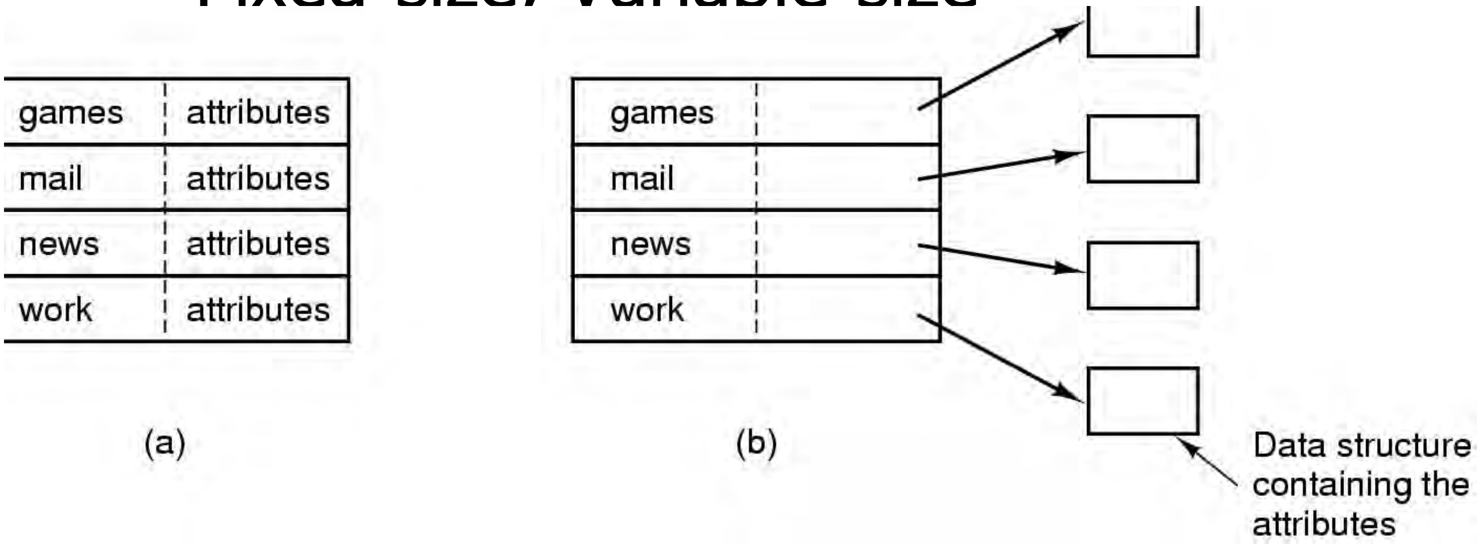
Question 1. when the size of physical block is 1024?

Question 2. when the size of physical block is 2048?

Implement of Directories

- Directory entry
  - File Control Block, FCB
  - Fixed-size, Variable-size

- File
  - FCB
  - Body



games	attributes
mail	attributes
news	attributes
work	attributes

(a)

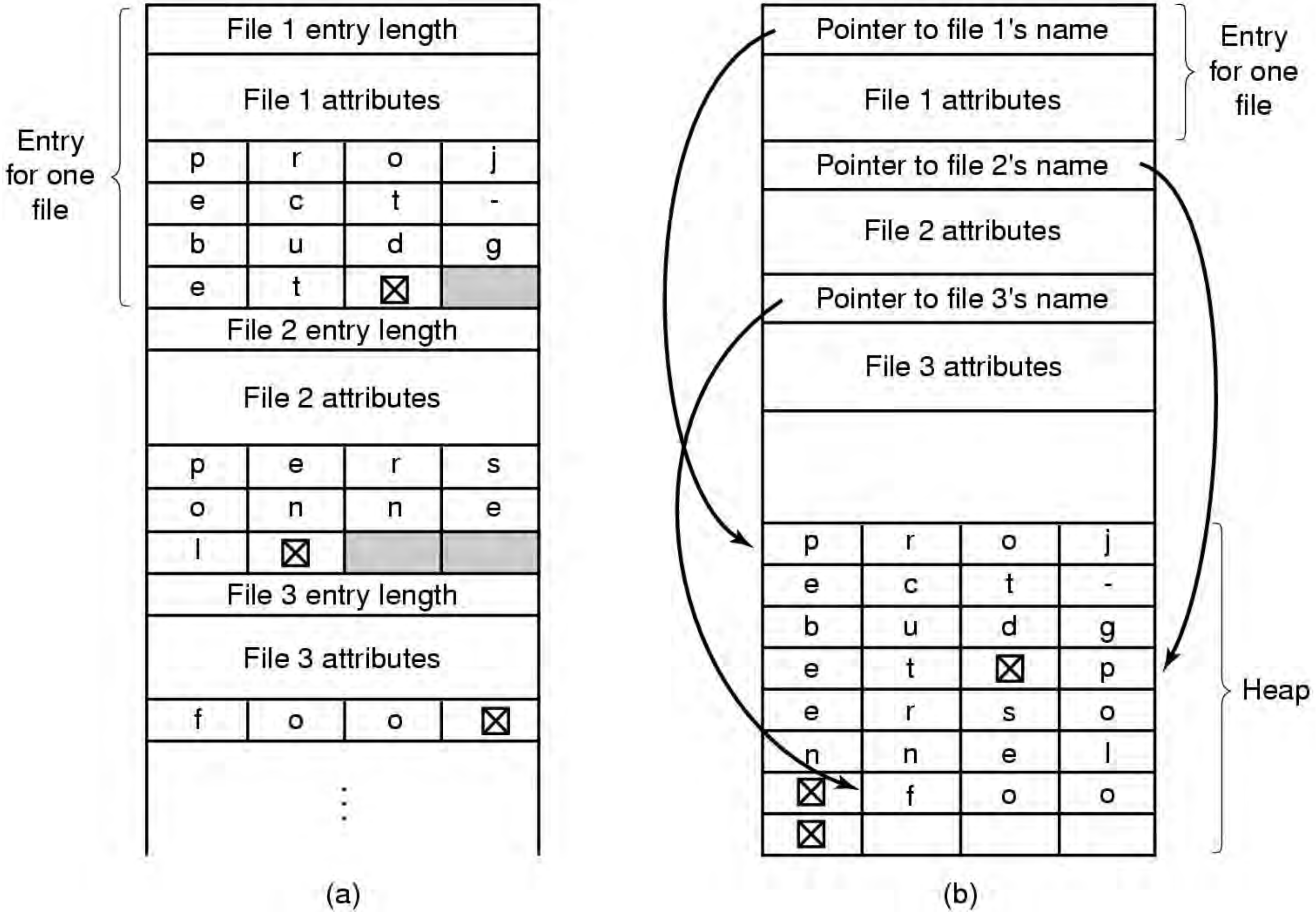
games
mail
news
work

(b)

(a) A simple directory containing fixed-size entries with the disk addresses and attributes in the directory entry.

(b) A directory in which each entry just refers to an i-node.

Implement of Directories

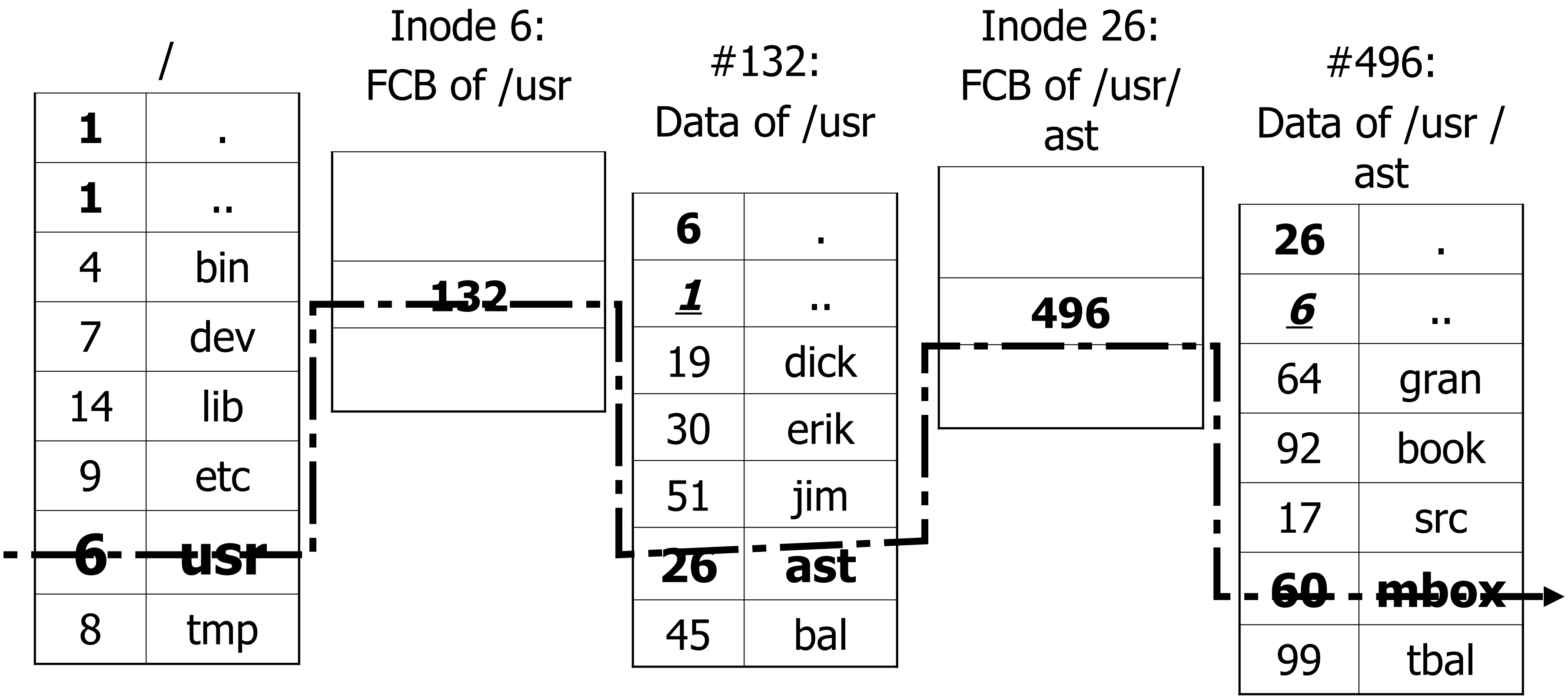


Two ways of handling long file names in a directory

(a) In-line. (b) In a heap.

Implement of Directories

- Search file by pathname
  - /usr/ast/mbox



/

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

Inode 6:
FCB of /usr

132

#132:
Data of /usr

6	.
1	..
19	dick
30	erik
51	jim
26	**ast**
45	bal

Inode 26:
FCB of /usr/
ast

496

#496:
Data of /usr /
ast

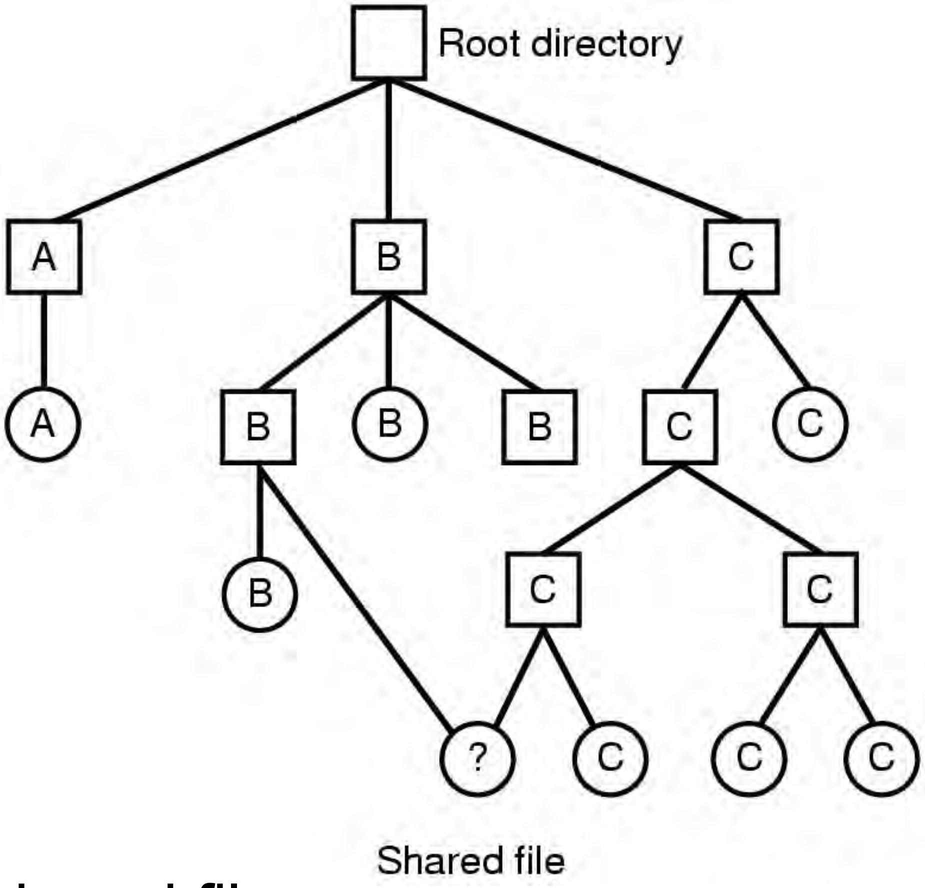
26	.
6	..
64	gran
92	book
17	src
60	**mbox**
99	tbal

Implement of Directories

- Directory entry
  - Linear List
  - Hash Table

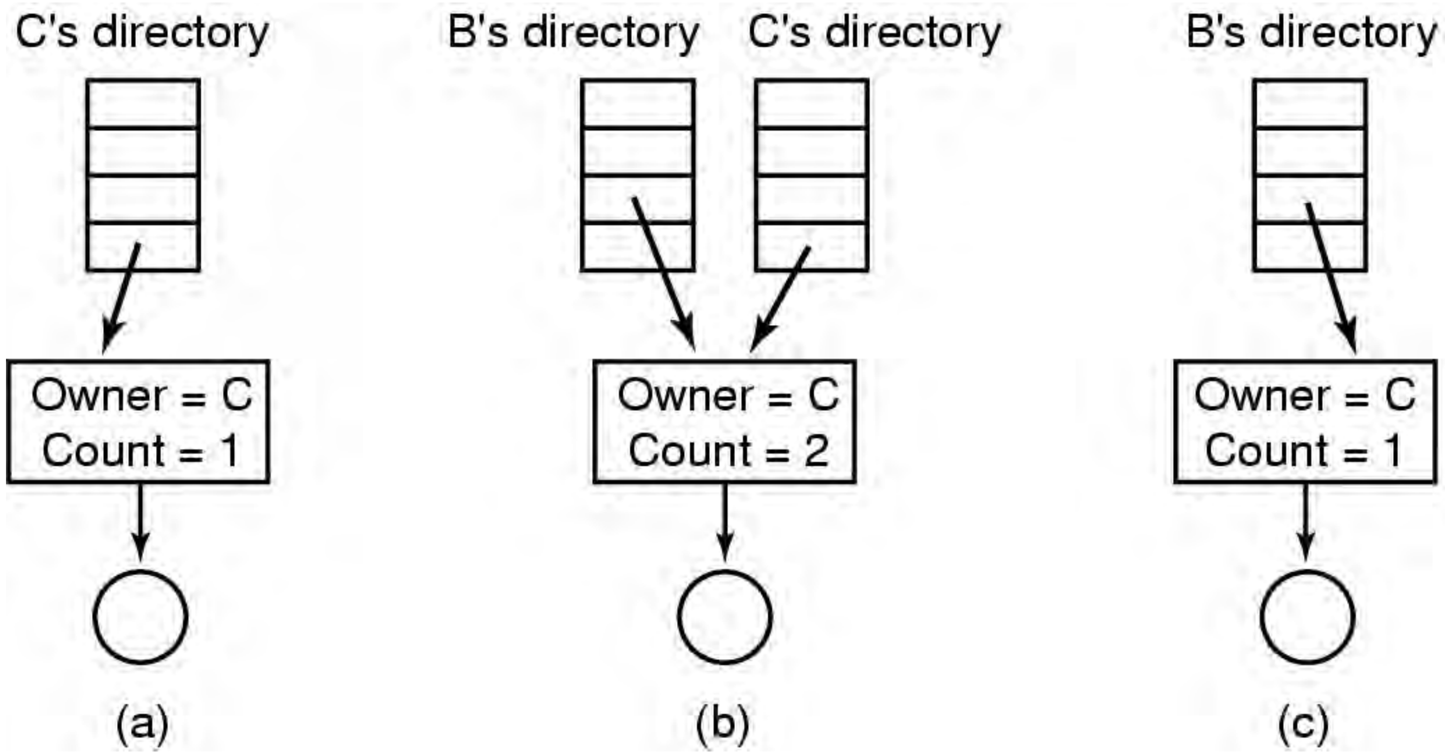
Shared Files

- Symbolic linking
- Hard linking



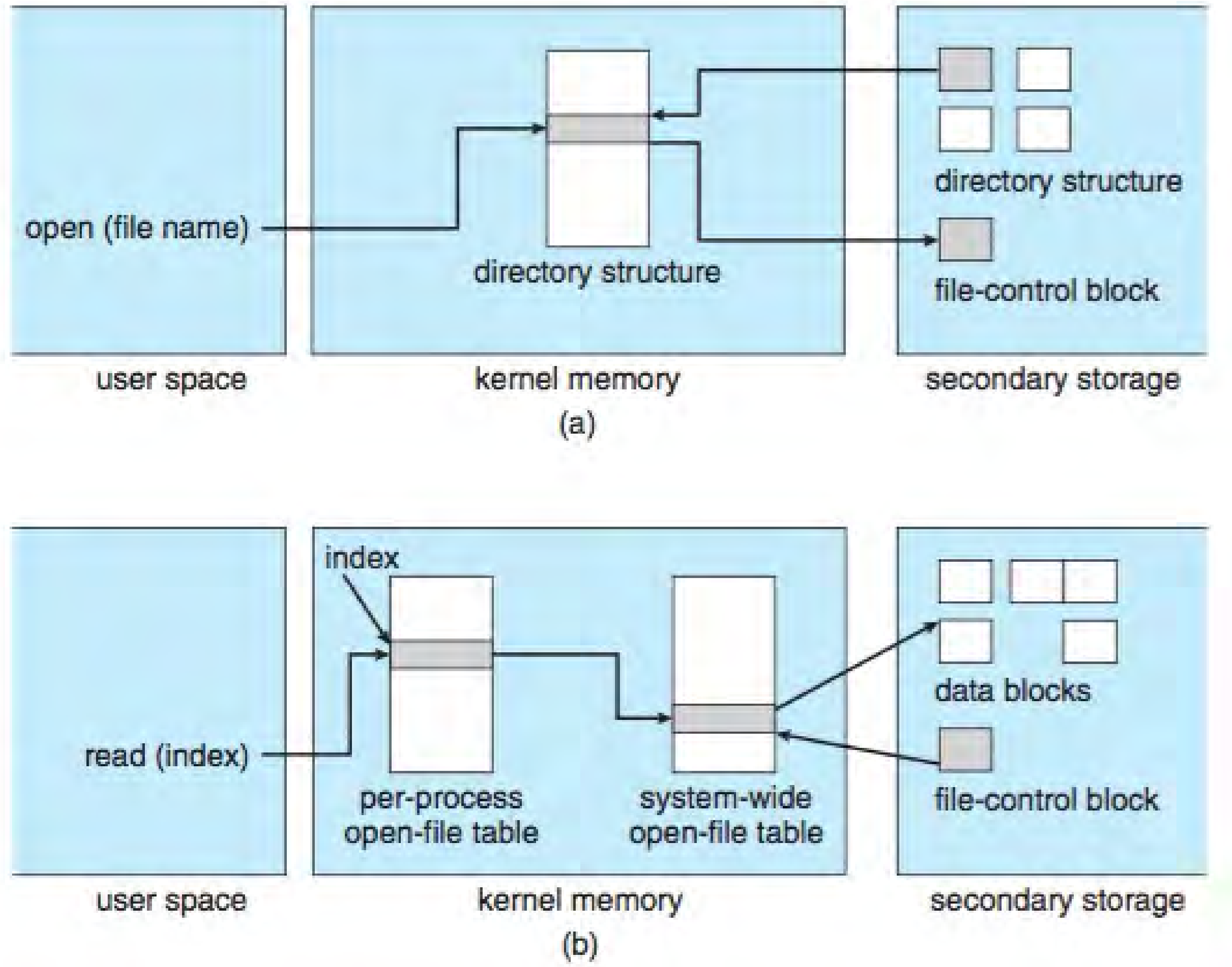
File system containing a shared file.

Shared Files



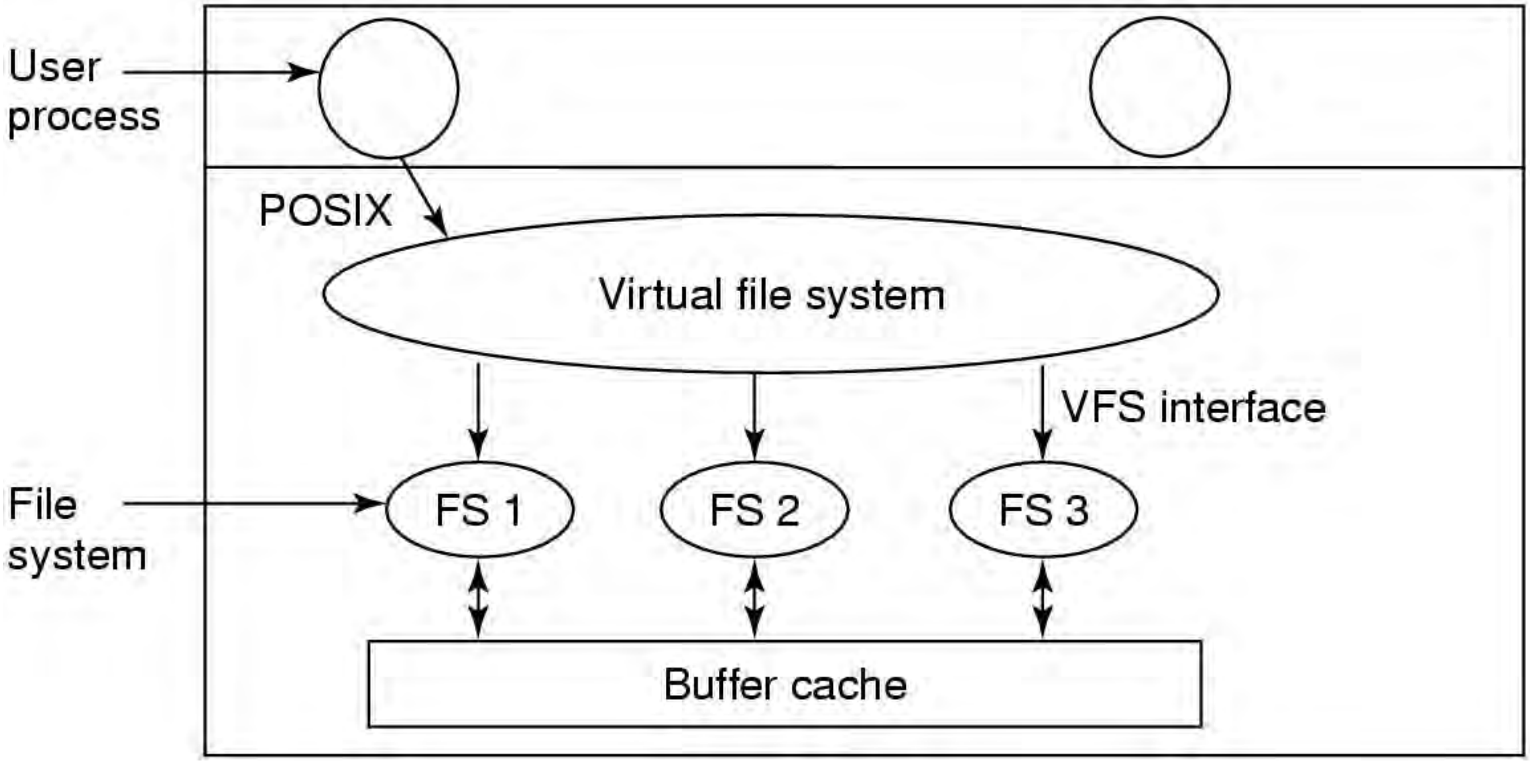
(a) Situation prior to linking. (b) After the link is created.
(c) After the original owner removes the file.

In-memory file-system structures

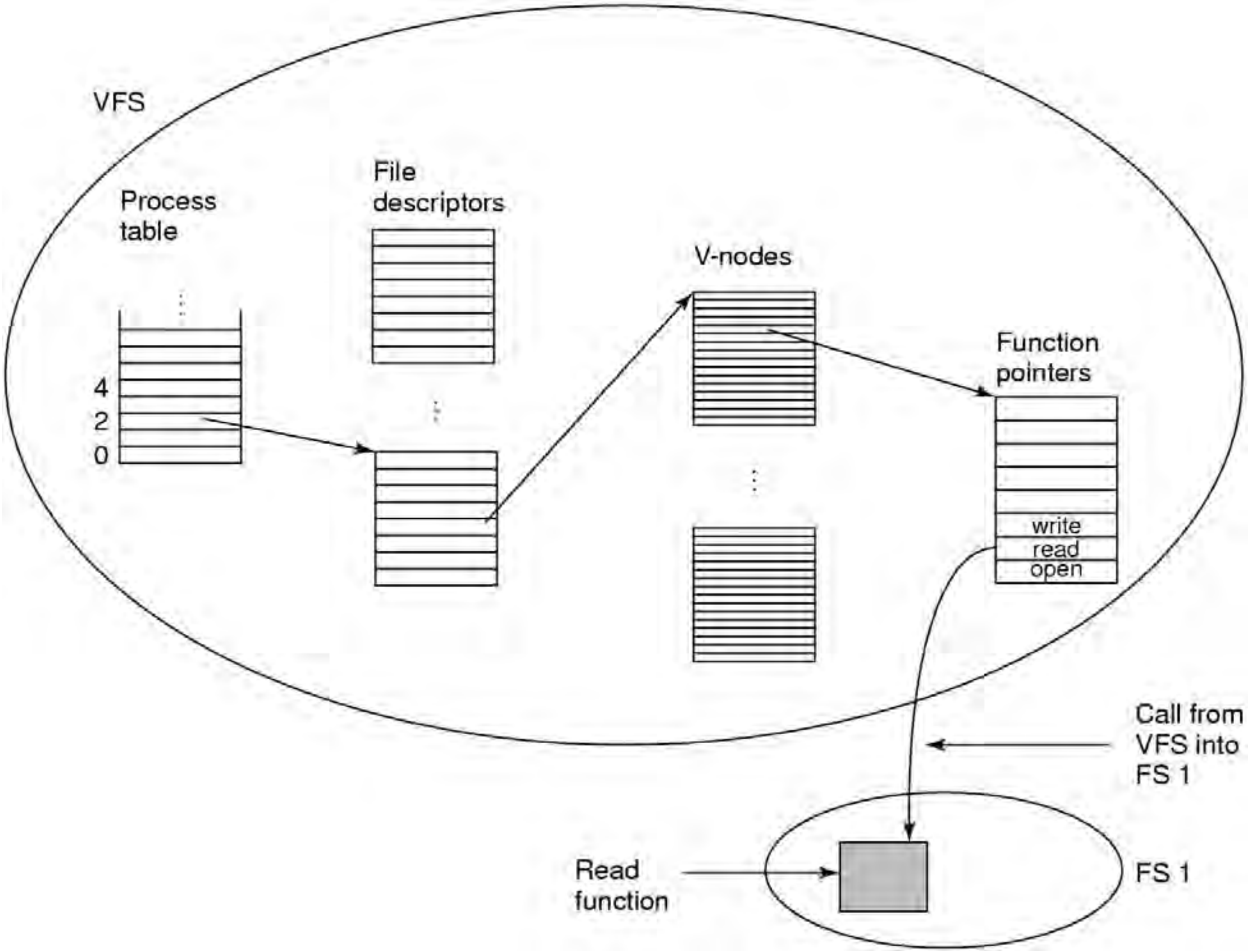


Virtual File Systems

- VFS Interface



Virtual File Systems



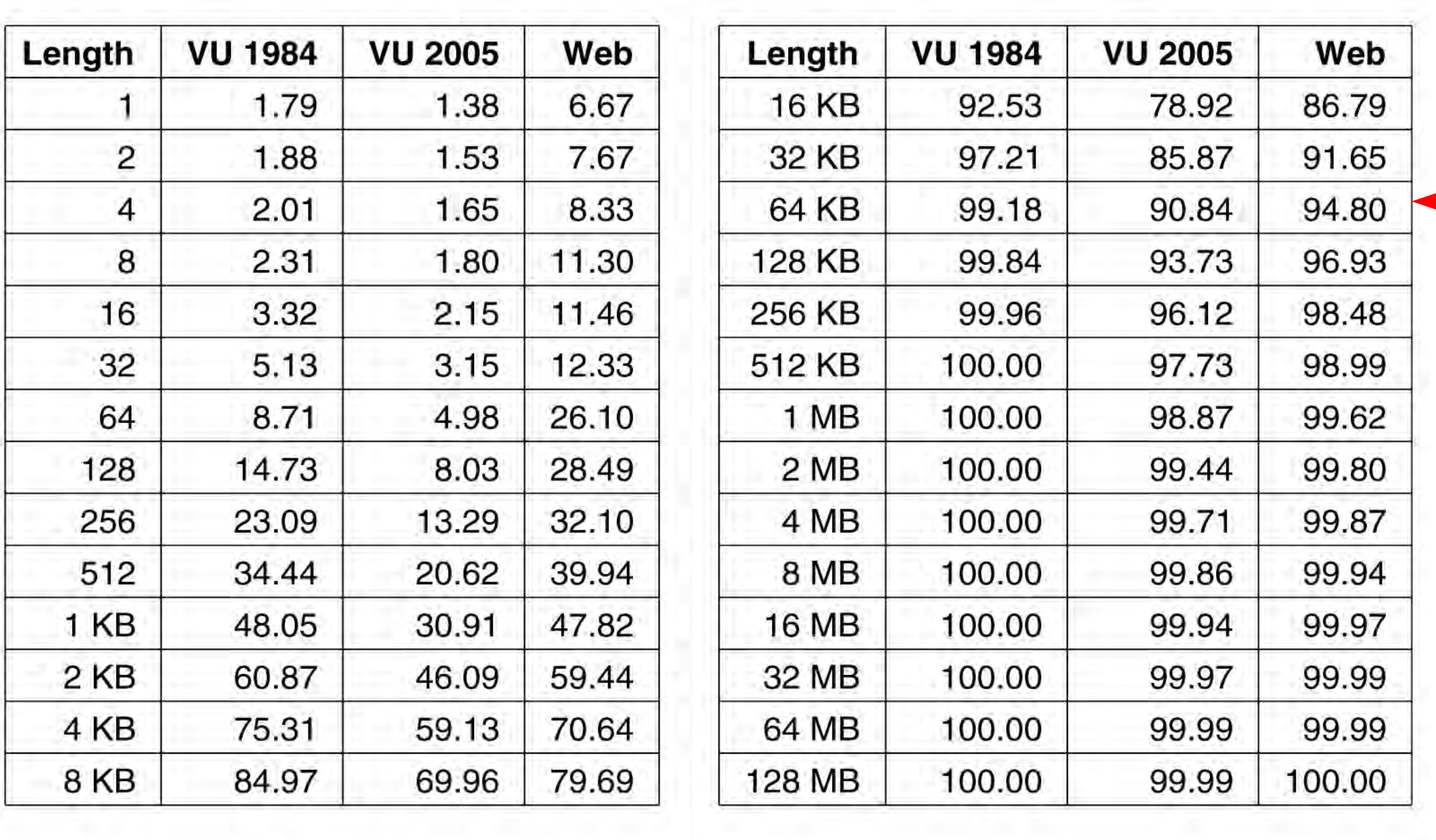
A simplified view of the data structures and code used by the VFS and concrete file system to do a read.

Disk Space Management

- Block Size
- Keeping Track of Free Blocks
- Disk Quotas
- File System Backups
- File System Consistency
- File System Performance
- Defragmenting Disks

Block Size

- Small ? Large?

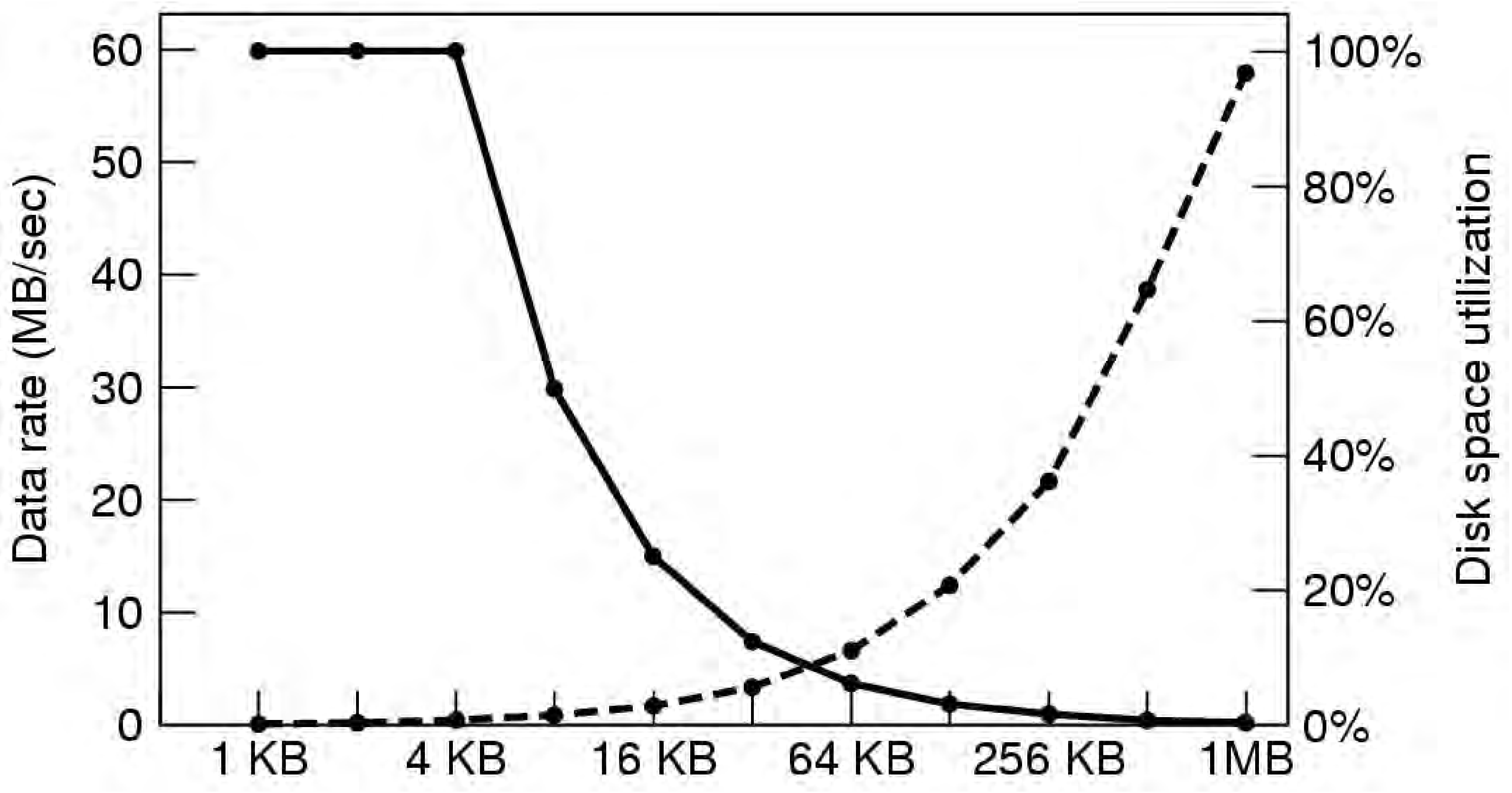


Length	VU 1984	VU 2005	Web
1	1.79	1.38	6.67
2	1.88	1.53	7.67
4	2.01	1.65	8.33
8	2.31	1.80	11.30
16	3.32	2.15	11.46
32	5.13	3.15	12.33
64	8.71	4.98	26.10
128	14.73	8.03	28.49
256	23.09	13.29	32.10
512	34.44	20.62	39.94
1 KB	48.05	30.91	47.82
2 KB	60.87	46.09	59.44
4 KB	75.31	59.13	70.64
8 KB	84.97	69.96	79.69

Length	VU 1984	VU 2005	Web
16 KB	92.53	78.92	86.79
32 KB	97.21	85.87	91.65
64 KB	99.18	90.84	94.80
128 KB	99.84	93.73	96.93
256 KB	99.96	96.12	98.48
512 KB	100.00	97.73	98.99
1 MB	100.00	98.87	99.62
2 MB	100.00	99.44	99.80
4 MB	100.00	99.71	99.87
8 MB	100.00	99.86	99.94
16 MB	100.00	99.94	99.97
32 MB	100.00	99.97	99.99
64 MB	100.00	99.99	99.99
128 MB	100.00	99.99	100.00

Percentage of files smaller than a given size (in bytes).

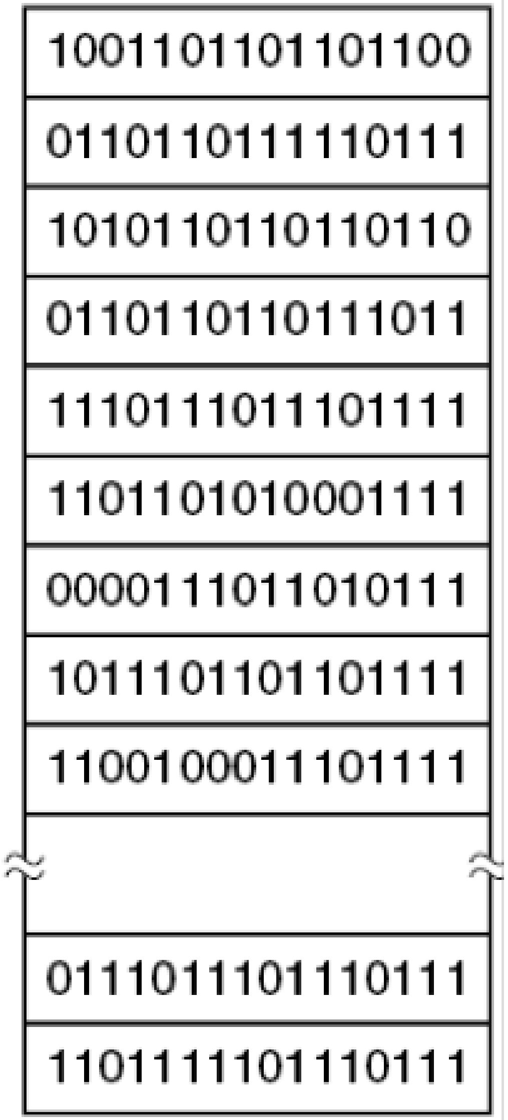
Block Size



The solid curve (left-hand scale) gives the data rate of a disk.
The dashed curve (right-hand scale) gives the disk space efficiency.
All files are 4 KB.

Keeping Track of Free Blocks

- Bit Vector
  - Bitmap

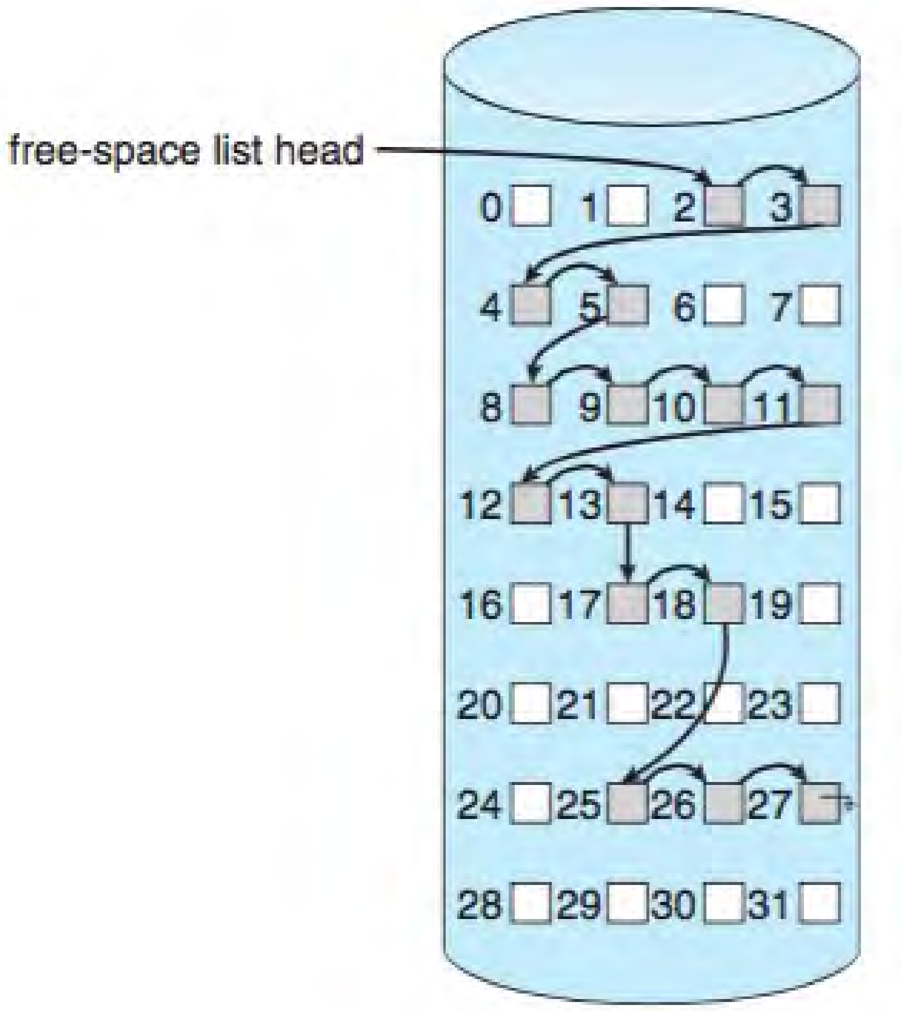


```
1001101101101100
0110110111110111
1010110110110110
0110110110111011
1110111011101111
1101101010001111
0000111011010111
1011101101101111
1100100011101111
≈                ≈
0111011101110111
1101111101110111
```

A bitmap

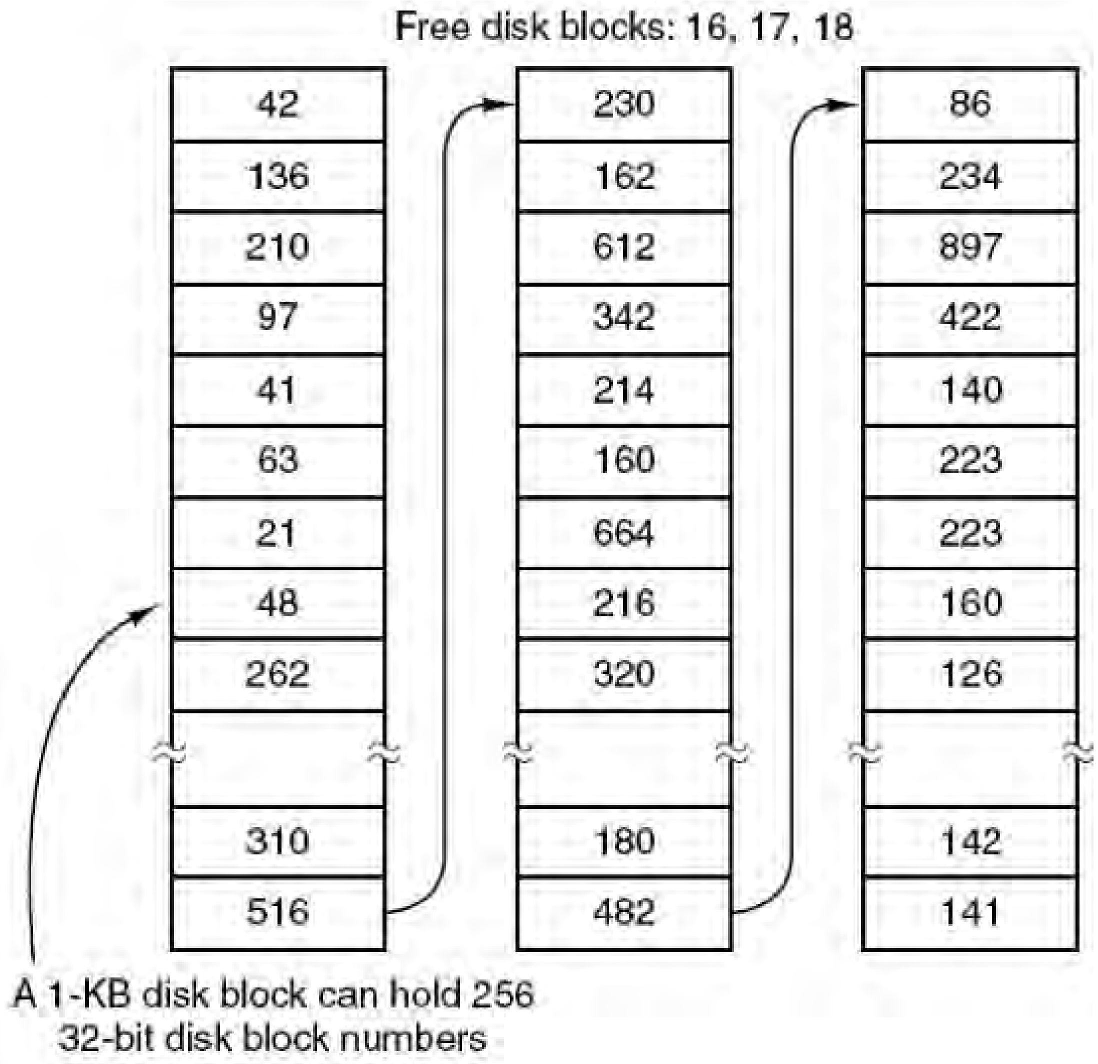
Keeping Track of Free Blocks

- Linked List



Keeping Track of Free Blocks

- Grouping Linked List: 成组连接法

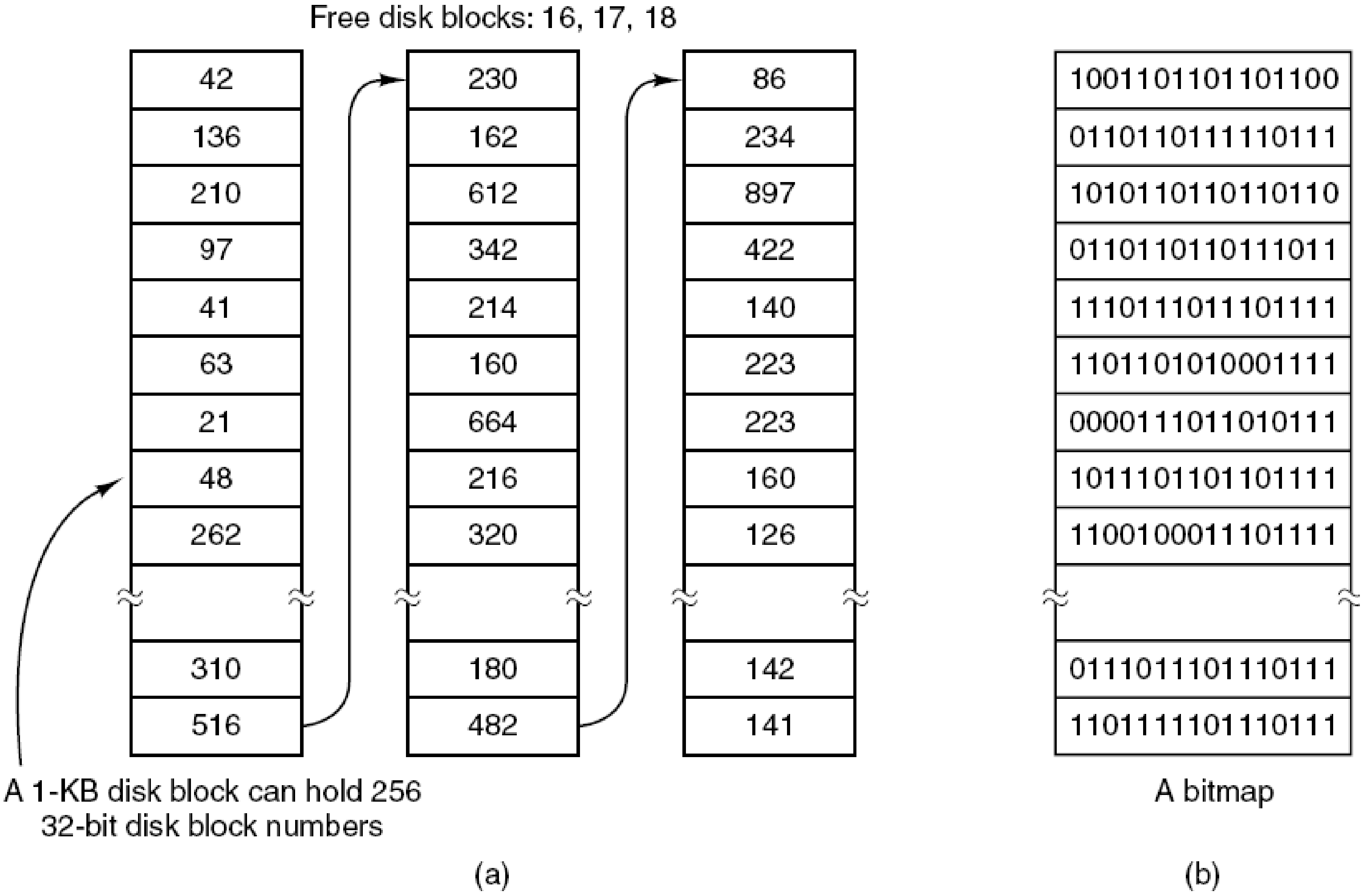


Free disk blocks: 16, 17, 18

42	230	86
136	162	234
210	612	897
97	342	422
41	214	140
63	160	223
21	664	223
48	216	160
262	320	126
310	180	142
516	482	141

A 1-KB disk block can hold 256 32-bit disk block numbers

Keeping Track of Free Blocks

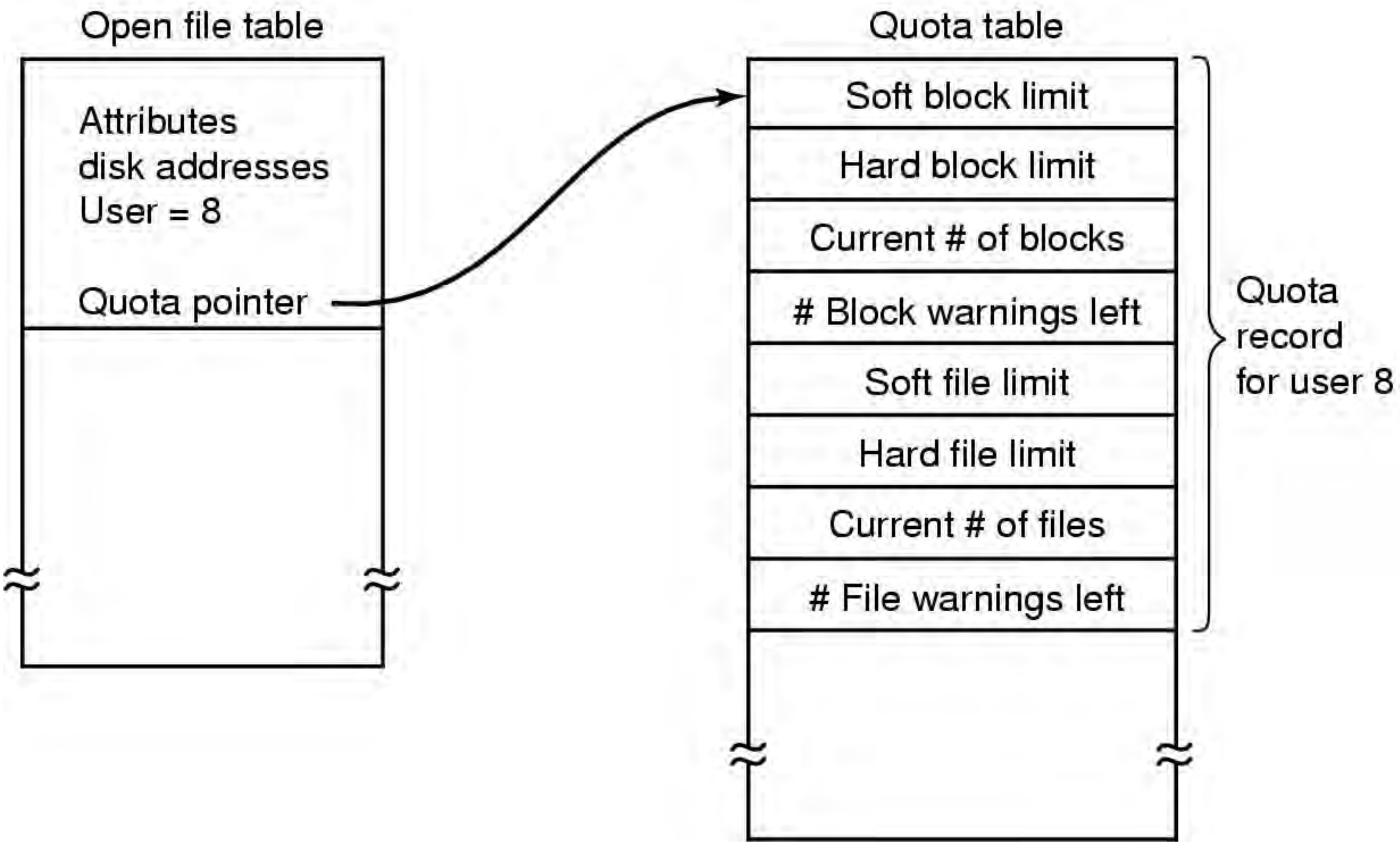


Free disk blocks: 16, 17, 18
A 1-KB disk block can hold 256 32-bit disk block numbers
A bitmap
(a)
(b)

42	230	86	1001101101101100
136	162	234	0110110111110111
210	612	897	1010110110110110
97	342	422	0110110110111011
41	214	140	1110111011101111
63	160	223	1101101010001111
21	664	223	0000111011010111
48	216	160	1011101101101111
262	320	126	1100100011101111
≈	≈	≈	≈
310	180	142	0111011101110111
516	482	141	1101111101110111

(a) Storing the free list on a linked list. (b) A bitmap.

Disk Quotas

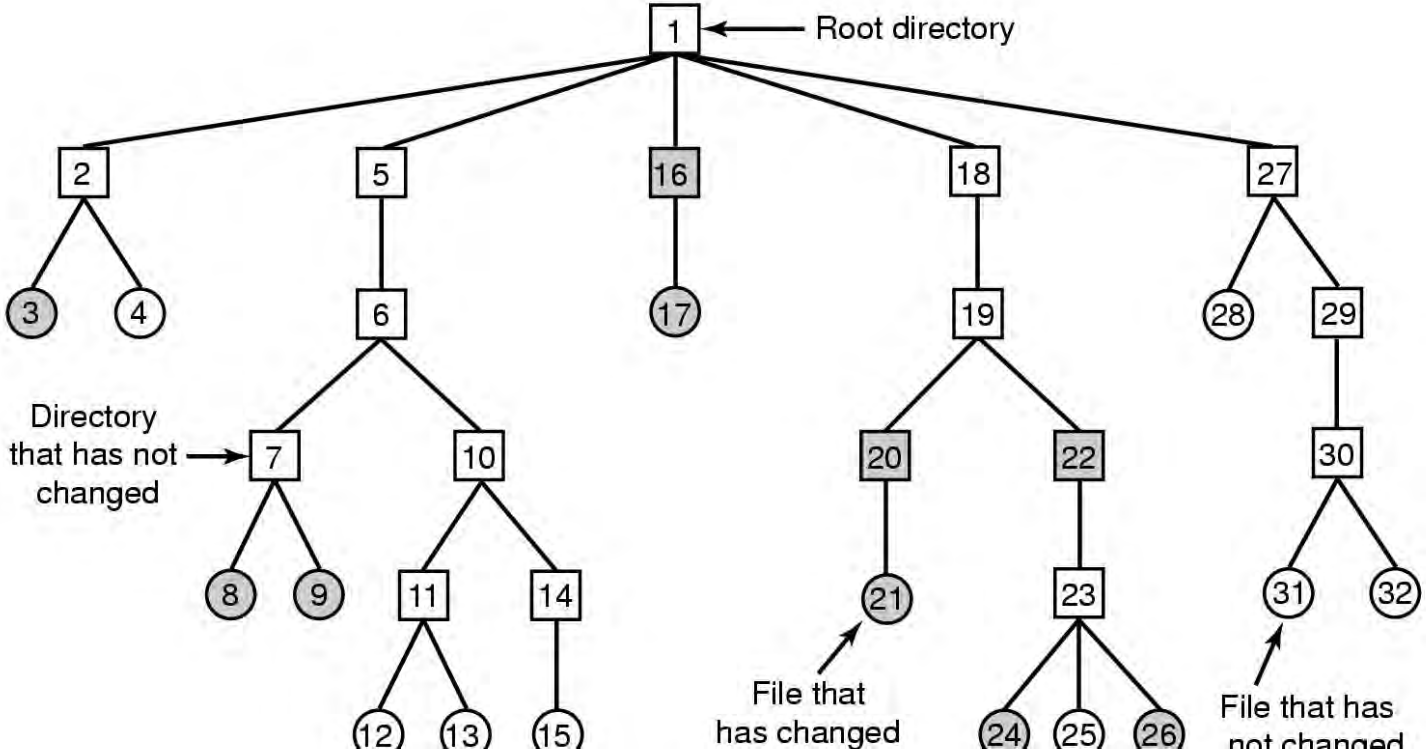

Open file table
Attributes
disk addresses
User = 8
Quota pointer
Quota table
Soft block limit
Hard block limit
Current # of blocks
# Block warnings left
Soft file limit
Hard file limit
Current # of files
# File warnings left
Quota
record
for user 8

File System Backups

- Backup v.s. Recover
- Physical dump v.s. logical dump

File System Backups

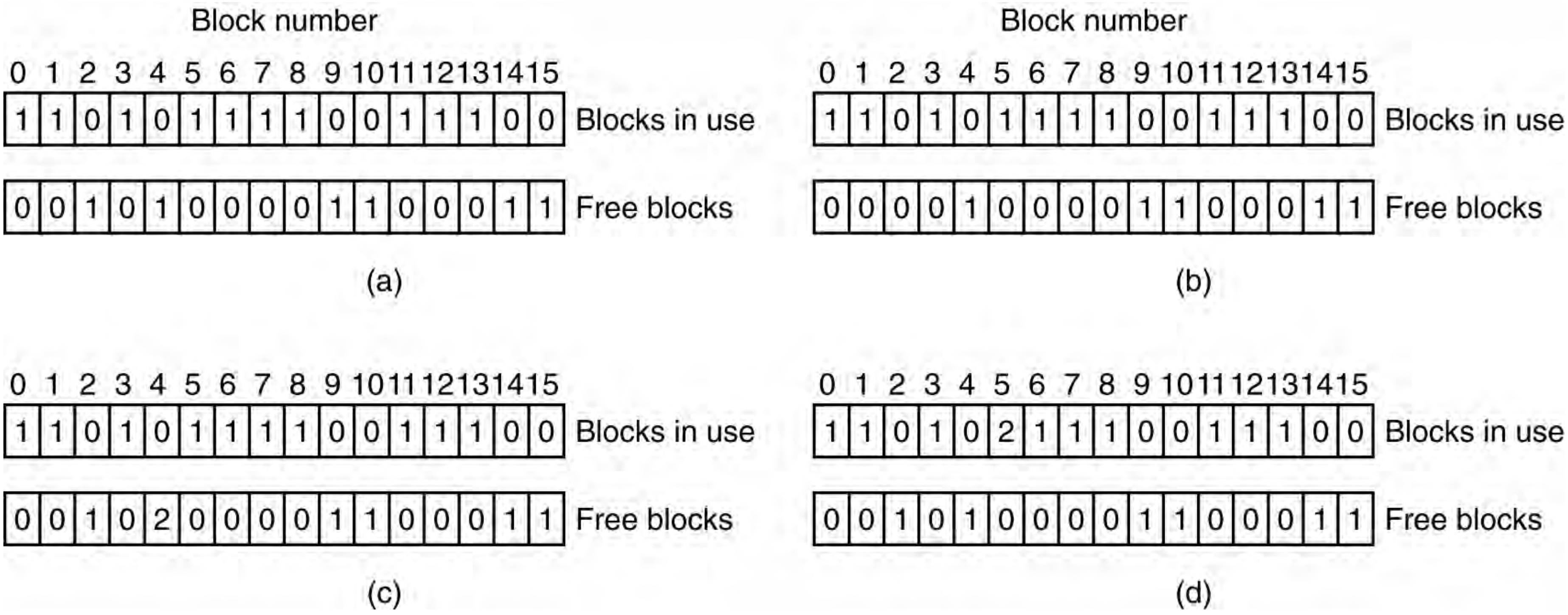
- Full dump v.s. incremental dump



A file system to be dumped. Squares are directories, circles are files. Shaded items have been modified since last dump. Each directory and file is labeled by its i-node number.

File System Consistency

- Blocks in use
- Free blocks



Block number

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	
1	1	0	1	0	1	1	1	1	0	0	1	1	1	0	0	Blocks in use
0	0	1	0	1	0	0	0	0	1	1	0	0	0	1	1	Free blocks

(a)

Block number

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	
1	1	0	1	0	1	1	1	1	0	0	1	1	1	0	0	Blocks in use
0	0	0	0	1	0	0	0	0	1	1	0	0	0	1	1	Free blocks

(b)

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	
1	1	0	1	0	1	1	1	1	0	0	1	1	1	0	0	Blocks in use
0	0	1	0	2	0	0	0	0	1	1	0	0	0	1	1	Free blocks

(c)

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	
1	1	0	1	0	2	1	1	1	0	0	1	1	1	0	0	Blocks in use
0	0	1	0	1	0	0	0	0	1	1	0	0	0	1	1	Free blocks

(d)

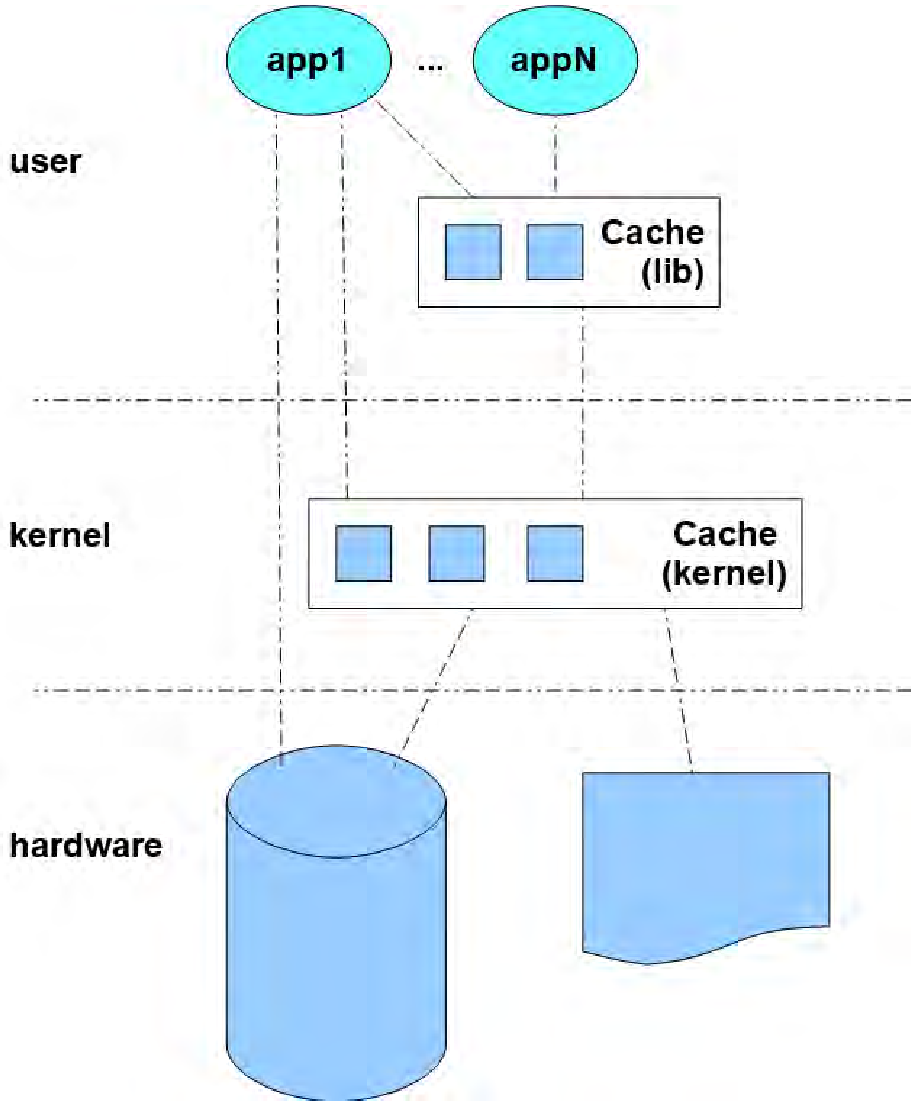
(a) Consistent. (b) Missing block.
(c) Duplicate block in free list. (d) Duplicate data block

File System Performance

- Caching
  - Block cache, buffer cache
- I-nodes layout
- Block Read Ahead
- Reducing Disk Arm Motion

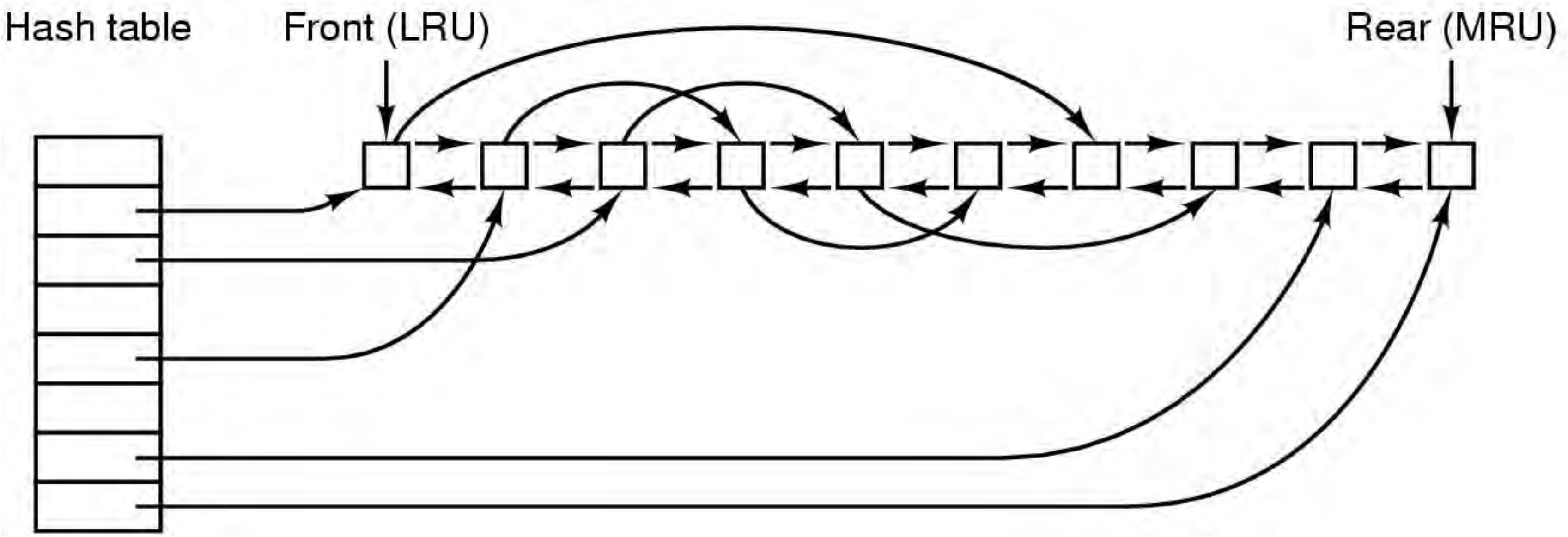
File System Performance: Cache

- Block cache
  - Buffer cache



File System Performance: Cache

- Block cache replacement algorithms
  - LRU, FIFO, ...



The buffer cache data structures

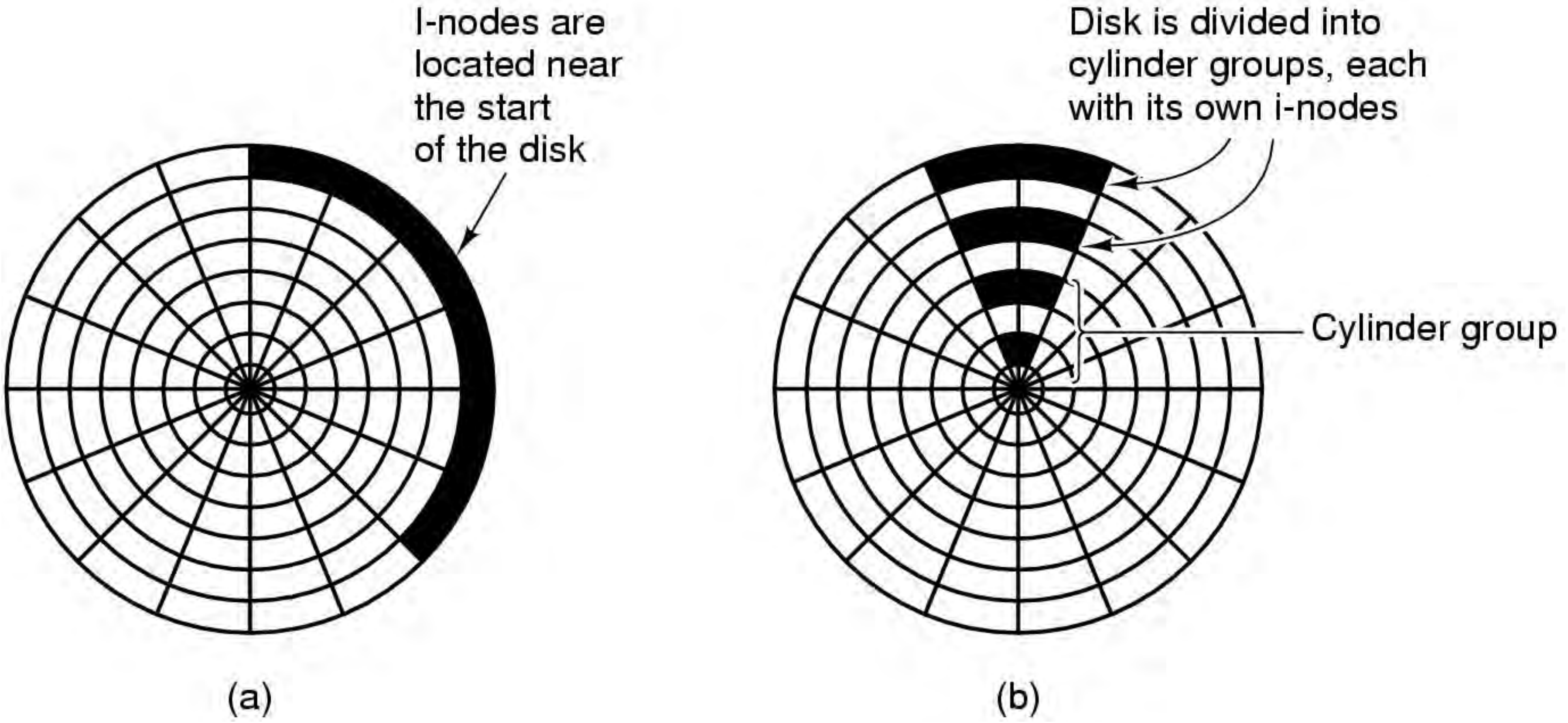
File System Performance: Cache

- Crash and File System Consistency?
- Write-back caches
- Write-through caches 通写高速缓存
- Sync, flushXXX

File System Performance

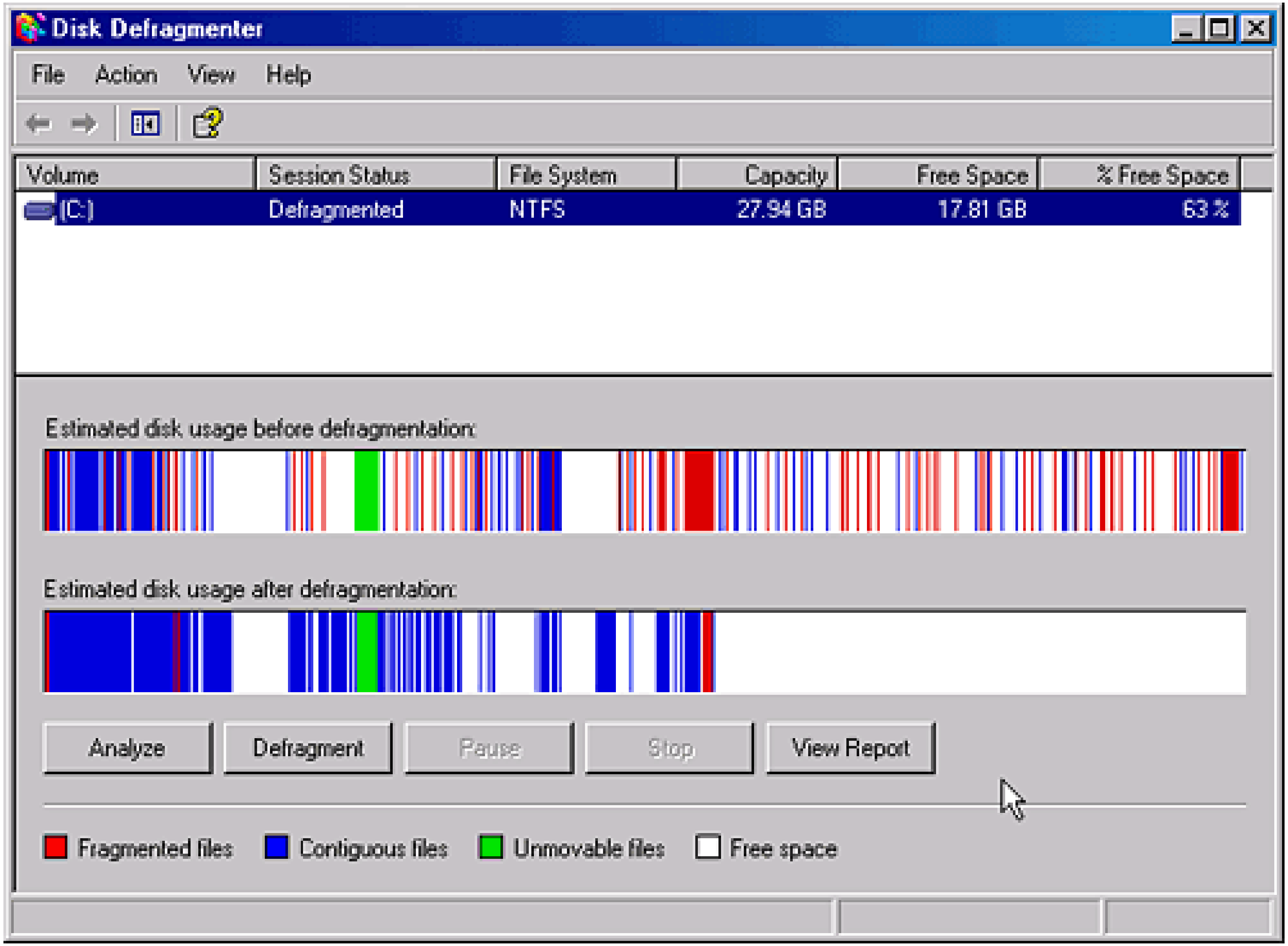
- Block Read Ahead
  - When executing a read from the disk, the disk arm moves the read/write head to (or near) the correct track, and after some settling time the read head begins to pick up bits.
  - Usually, the first sectors to be read are not the ones that have been requested by the operating system.
  - The disk's embedded computer typically saves these unrequested sectors in the disk buffer, in case the operating system requests them later.

File System Performance: I-nodes


I-nodes are located near the start of the disk
Disk is divided into cylinder groups, each with its own i-nodes
Cylinder group
(a)
(b)

- **I-nodes** placed at the start of the disk
- Disk divided into cylinder groups
  - each with its own blocks and i-nodes

Defragmenting Disks



Disk Defragmenter

Volume	Session Status	File System	Capacity	Free Space	% Free Space
(C:)	Defragmented	NTFS	27.94 GB	17.81 GB	63 %

Estimated disk usage before defragmentation:

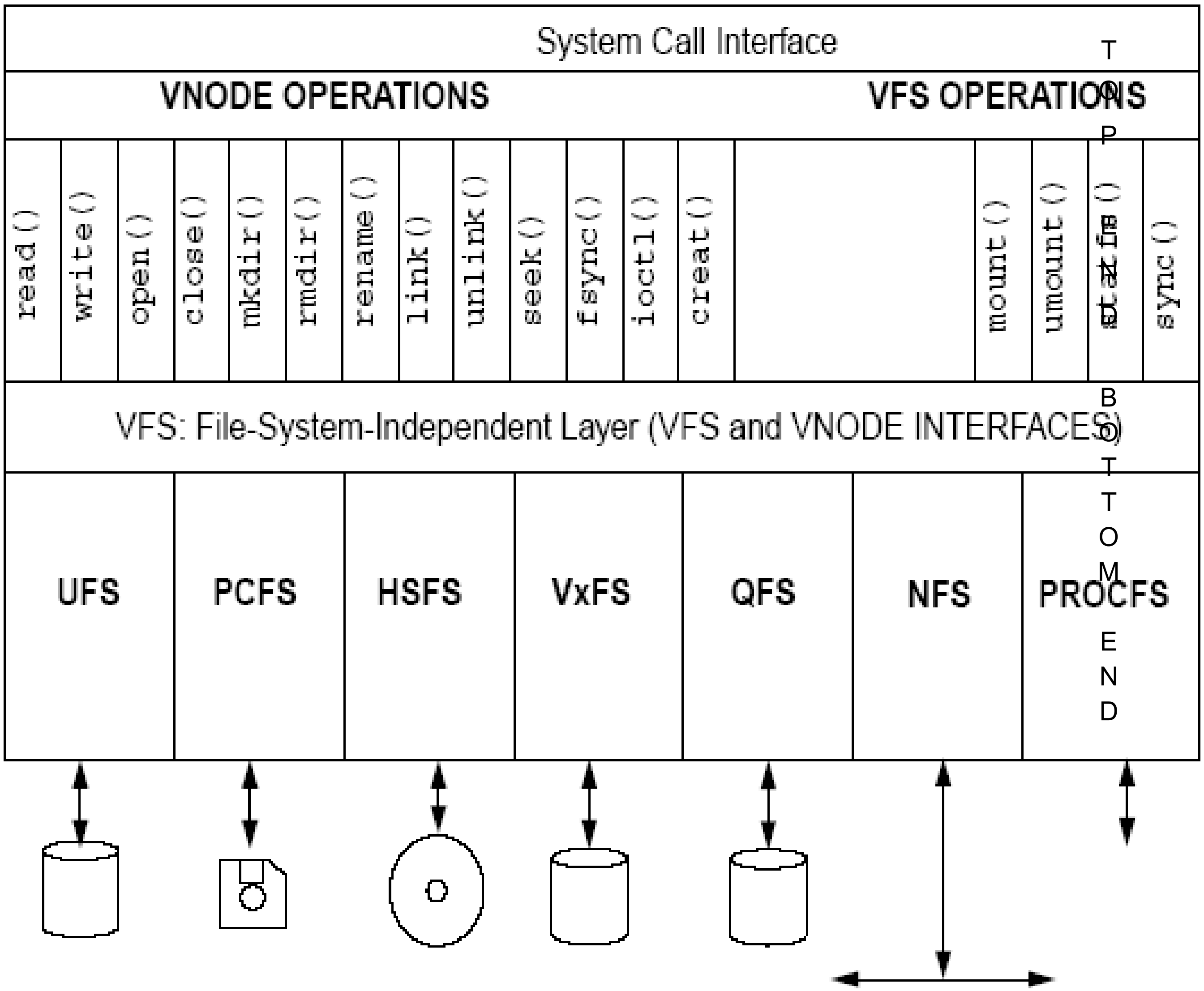
Estimated disk usage after defragmentation:

🟥 Fragmented files 🟦 Contiguous files 🟩 Unmovable files □ Free space

File System Framework Facilities

- **Loadable file system modules** are dynamically loaded at the time each file system type is first mounted.
- **The vnode/vfs framework** implementes file functions and file system management functions.
- **File system caching** implements caching interface with the HAT layer of the virtual memory system to map, unmap, and manage the memory used for caching.
- **Path-name management** converts paths into vnode pointers.
- **Directory name caching** provides a mechanism to cache pathname-to-vnode mappings.

File System Layers



System Call Interface

VNODE OPERATIONS

VFS OPERATIONS

read()
write()
open()
close()
mkdir()
rmdir()
rename()
link()
unlink()
seek()
fsync()
ioctl()
creat()

mount()
umount()
statfs()
sync()

VFS: File-System-Independent Layer (VFS and VNODE INTERFACES)

UFS
PCFS
HSFS
VxFS
QFS
NFS
PROCFS

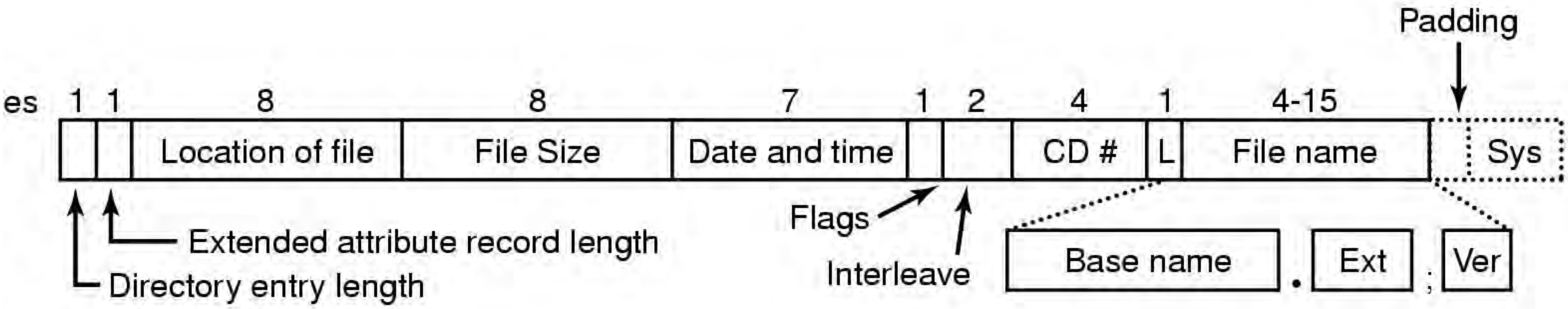
Topics of File Systems

- Log-structured File System
- Journaling File System
- Network File System: NFS

File Systems Cases

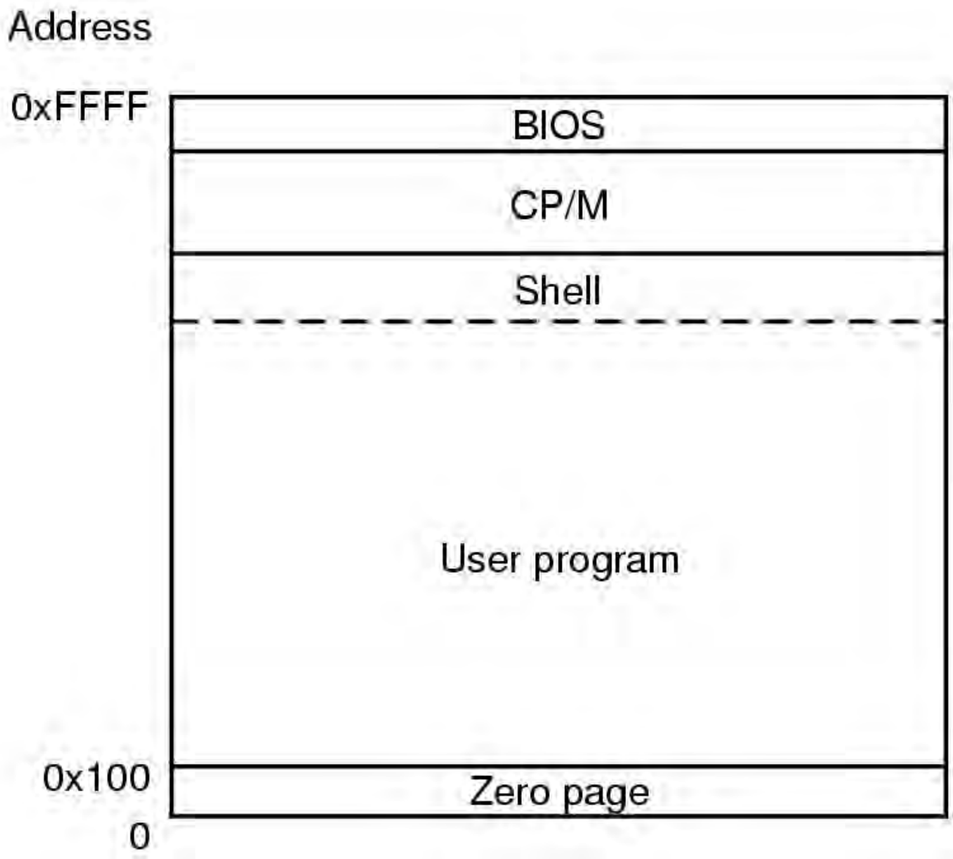
- CD-ROM: ISO9660
- CP/M File System
- FAT 16
- NTFS
- Ext4
- …

CD-ROM File Systems



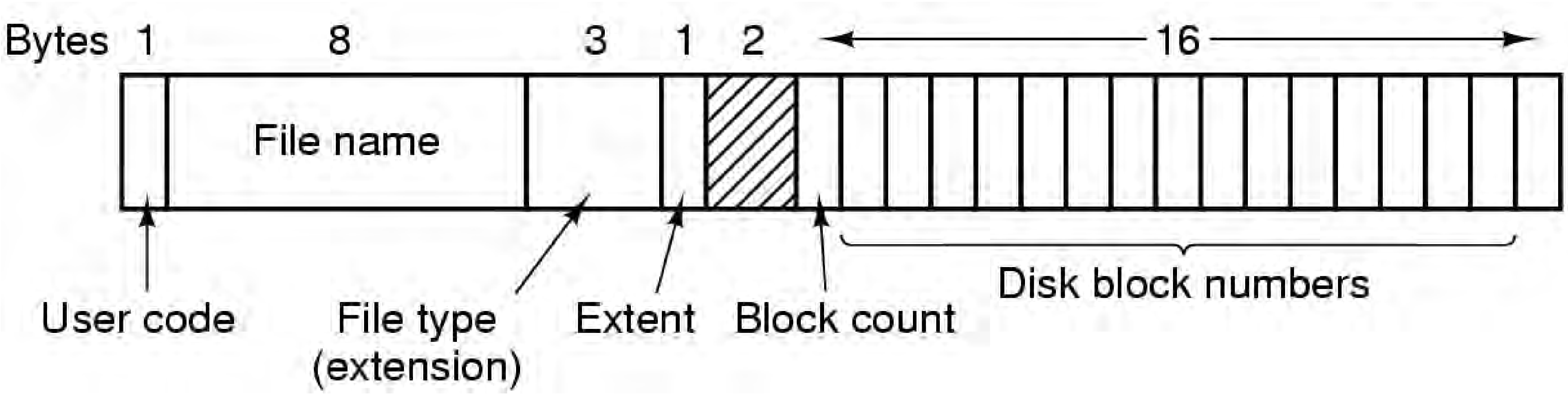
The ISO 9660 directory entry

The CP/M File System (1)



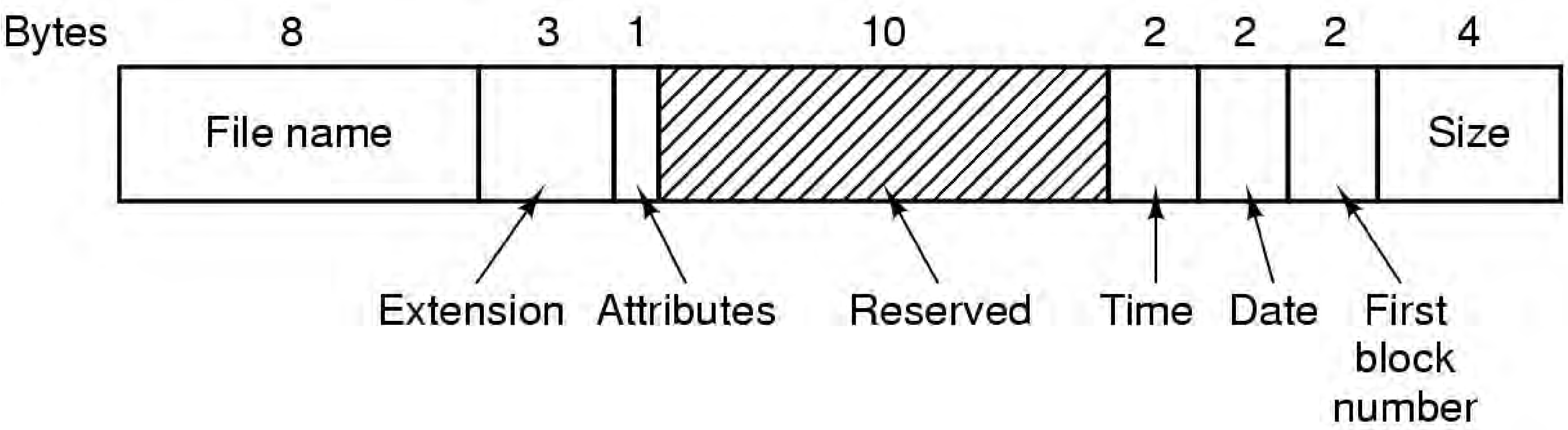
Memory layout of CP/M

The CP/M File System (2)



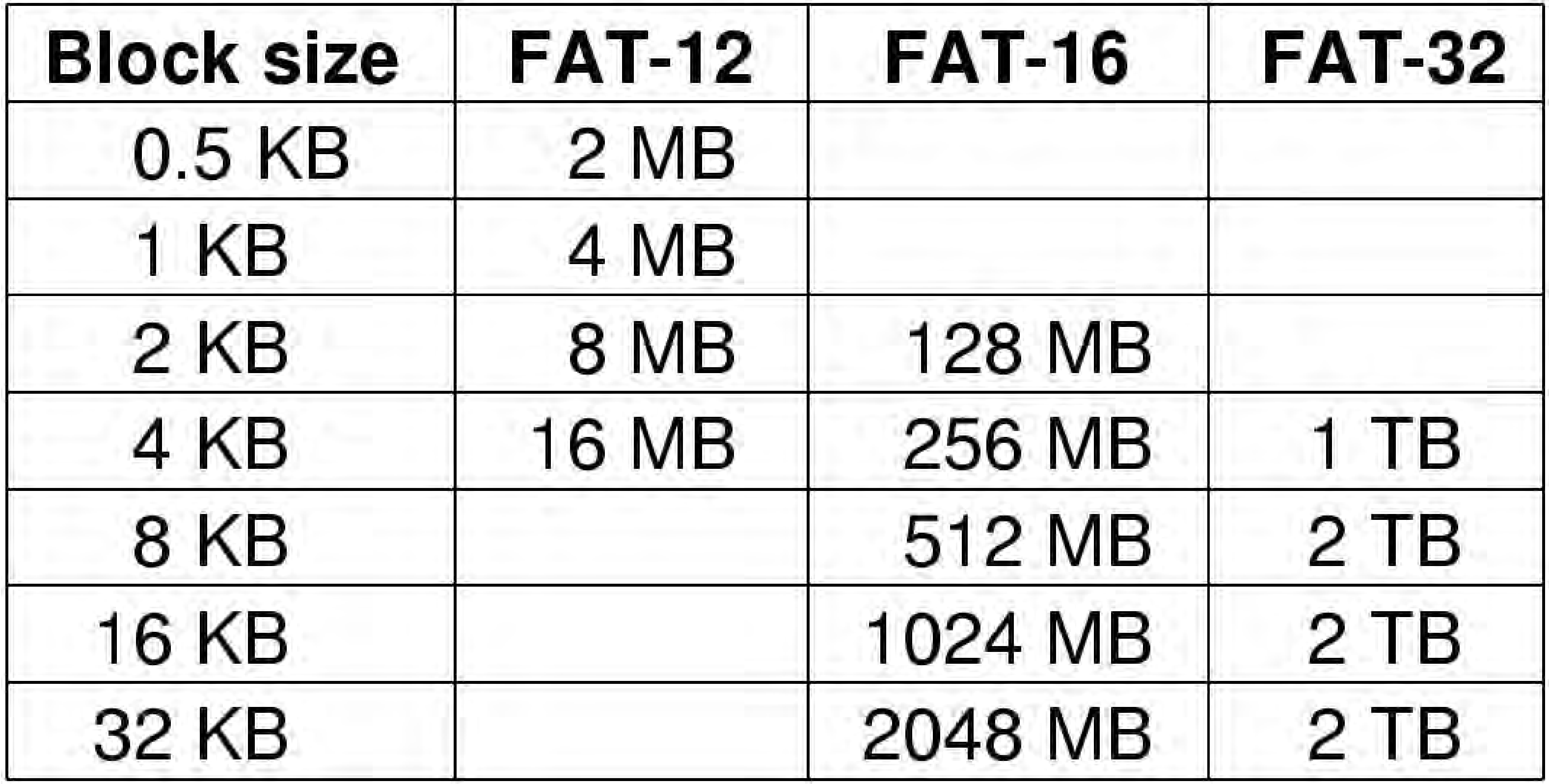
The CP/M directory entry format

The MS-DOS File System (1)



The MS-DOS directory entry

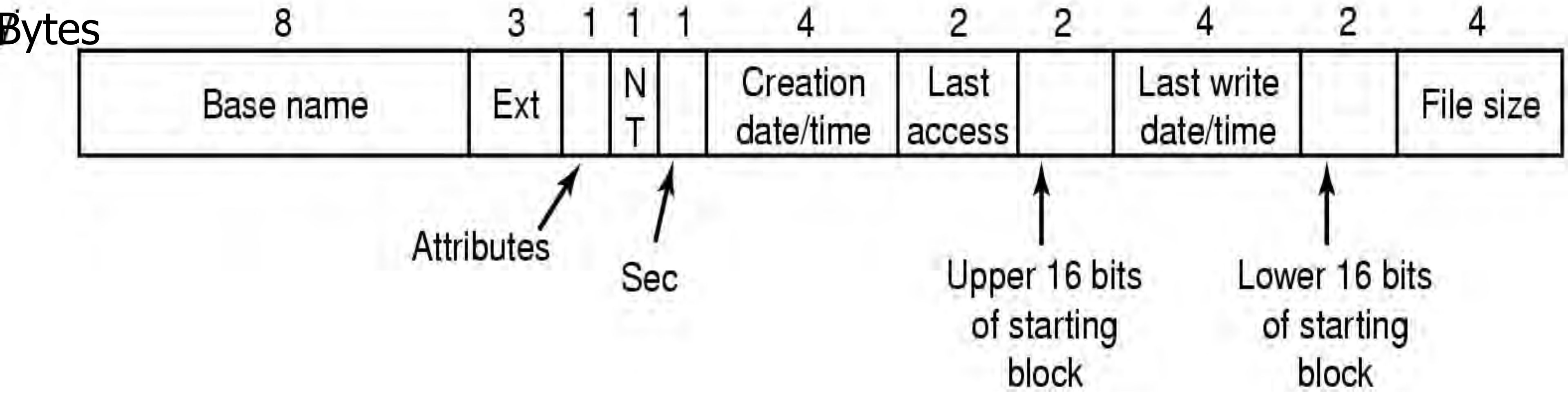
The MS-DOS File System (2)



Block size	FAT-12	FAT-16	FAT-32
0.5 KB	2 MB		
1 KB	4 MB		
2 KB	8 MB	128 MB	
4 KB	16 MB	256 MB	1 TB
8 KB		512 MB	2 TB
16 KB		1024 MB	2 TB
32 KB		2048 MB	2 TB

- Maximum partition for different block sizes
- The empty boxes represent forbidden combinations

The Windows 98 File System 1/3

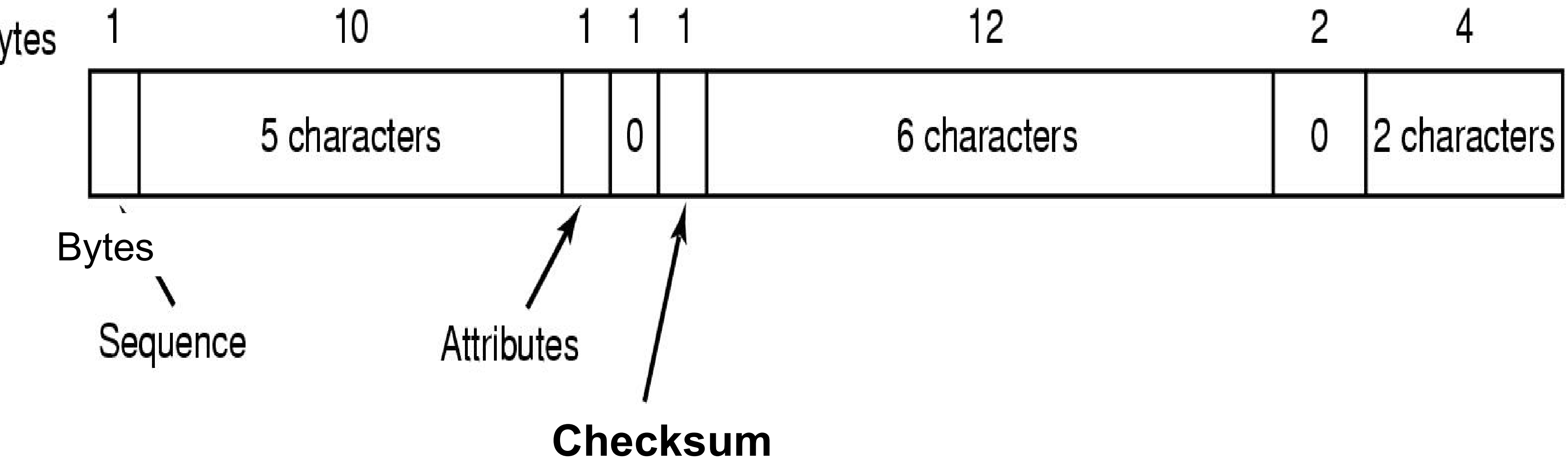


Bytes	8	3	1	1	1	4	2	2	4	2	4
	Base name	Ext		N T		Creation date/time	Last access		Last write date/time		File size

Attributes
Sec
Upper 16 bits of starting block
Lower 16 bits of starting block

The extended MOS-DOS directory entry used in Windows 98

The Windows 98 File System 2/3



An entry for (part of) a long file name in Windows 98

The Windows 98 File System 3/3

68	d o g	A	0	CK					0	
3	o v e	A	0	CK	t h e l a				0	z y
2	w n f o	A	0	CK	x j u m p				0	s
1	T h e q	A	0	CK	u i c k b				0	r o
T	H E Q U I ~ 1	A	NT	S	Creation time	Last acc	Upp	Last write	Low	Size

Bytes

An example of how a long name is stored in Windows 98

Summary

- File Concept
- Directory Concept
- File Share & Protection
- File System Implementation
- File System Reliability
- File System Performance
- File System Cases

Any Questions?