Chapter 6

File Systems

6.1 Files
6.2 Directories
6.3 File system implementation
6.4 Example file systems

sun@hit.edu.cn

Long-term Information Storage

- Must store large amounts of data
- Information stored must survive the termination of the process using it
- Multiple processes must be able to access the information concurrently

sun@hit.edu.cn

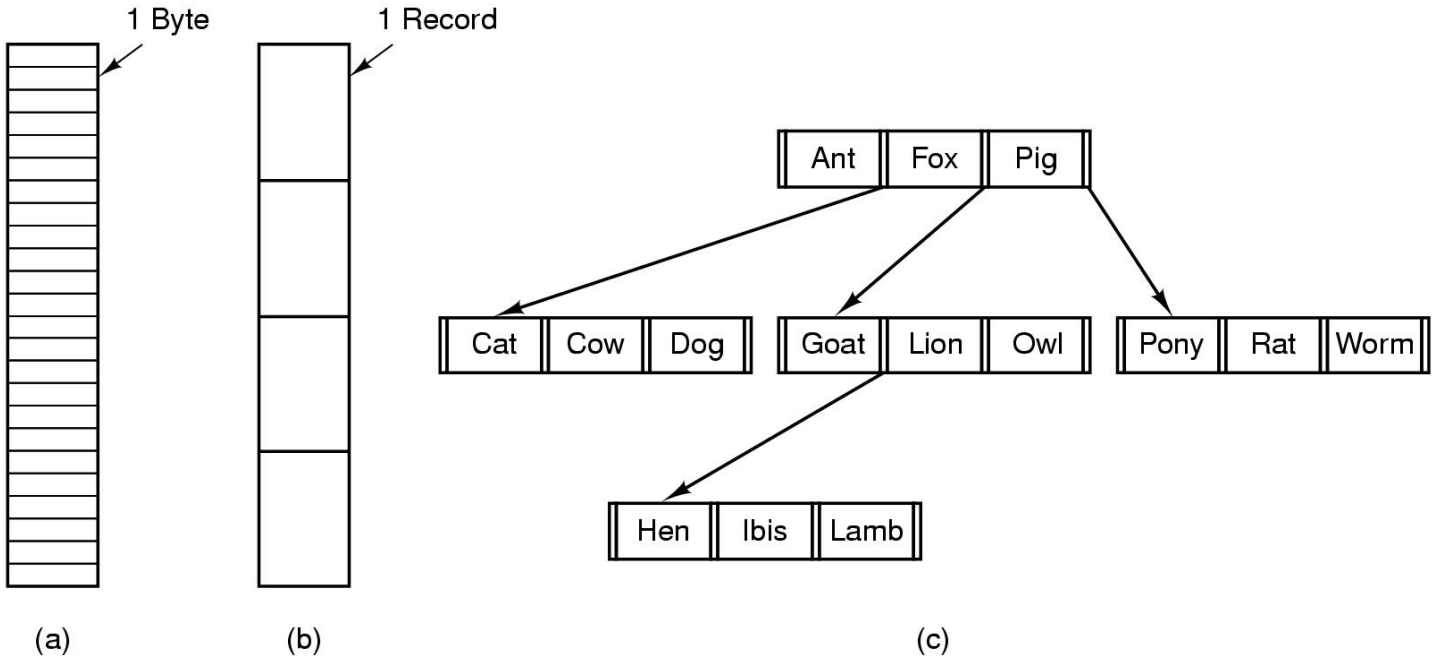
File Naming

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

Typical file extensions.

sun@hit.edu.cn

File Structure



- **Three kinds of files**
  - **byte sequence**
  - **record sequence**
  - **tree**

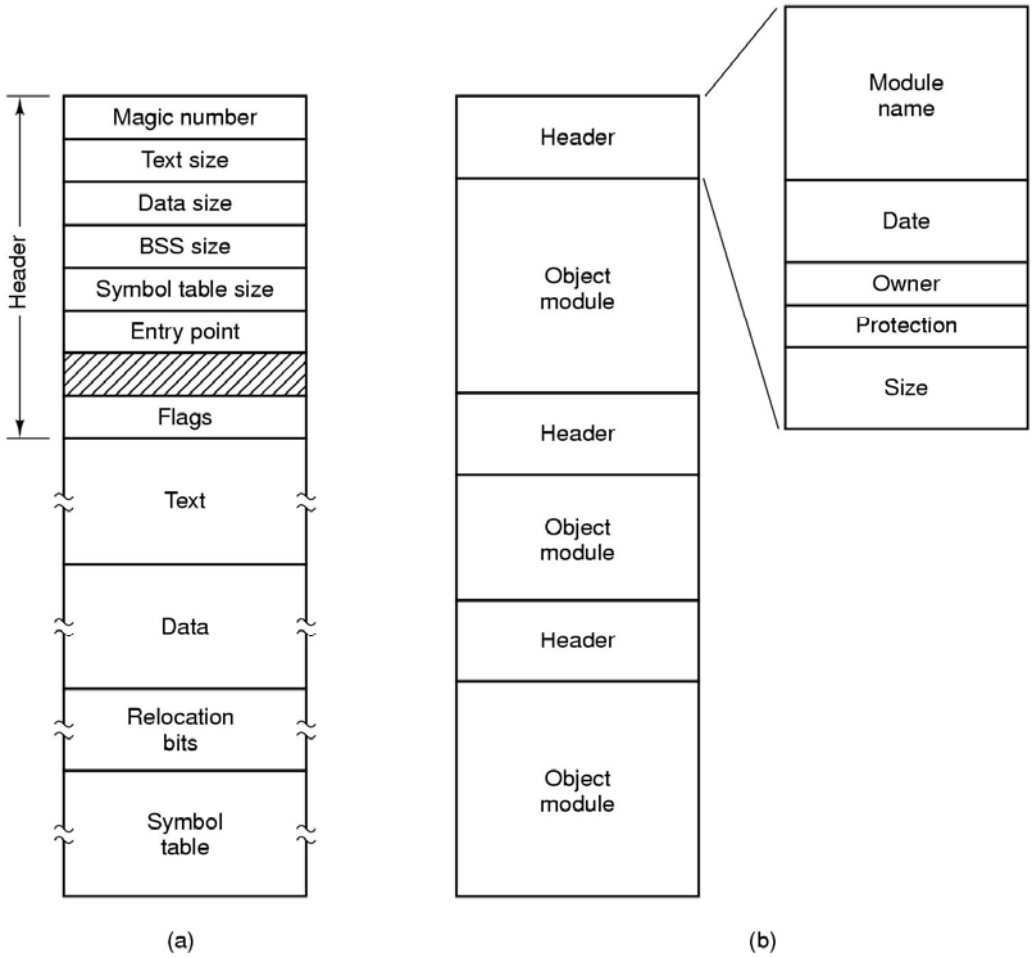
sun@hit.edu.cn

File Types

- **Regular files**
  - **the ones that contain user information**
  - **ASCII files and binary files**
- **Directories**
  - **system files for maintaining the structure of the file system**

sun@hit.edu.cn

File Types



(a) An executable file (b) An archive

sun@hit.edu.cn

File Access

- **Sequential access**
  - read all bytes/records from the beginning
  - cannot jump around, could rewind or back up
  - convenient when medium was magnetic tape
- **Random access**
  - bytes/records read in any order
  - essential for database systems
  - read can be …
    - move file marker (seek), then read or …
    - read and then move file marker

sun@hit.edu.cn

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 has last changed
Current size	Number of bytes in the file
Maximum size	Number of bytes the file may grow to

Possible file attributes

sun@hit.edu.cn

File Operations

1. Create
2. Delete
3. Open
4. Close
5. Read
6. Write
7. Append
8. Seek
9. Get attributes
10. Set Attributes
11. Rename

sun@hit.edu.cn

An Example 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 */
```

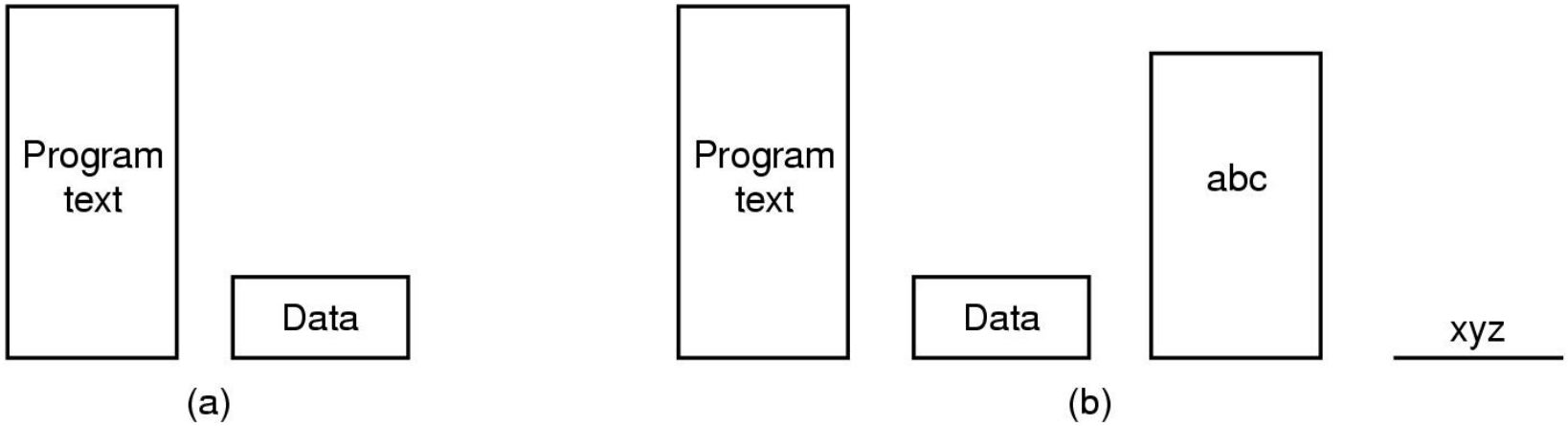
An Example Program Using File System Calls (2/2)

```
/* 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 */
}
```

Memory-Mapped Files



(a) Segmented process before mapping files into its address space

(b) Process after mapping

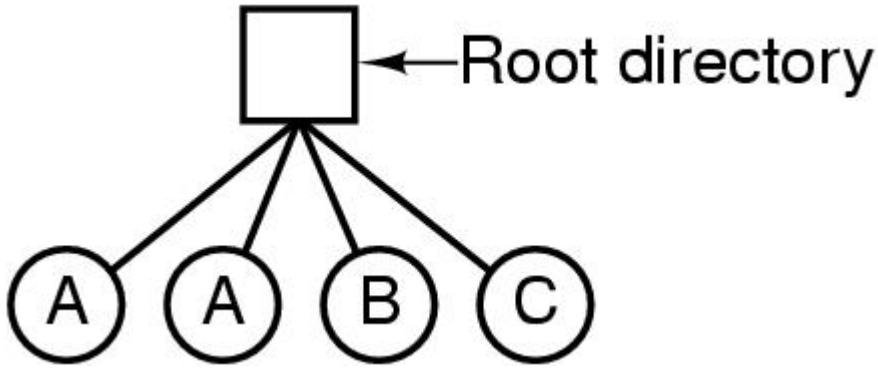
existing file *abc* into one segment

creating new segment for *xyz*

sun@hit.edu.cn

Directories

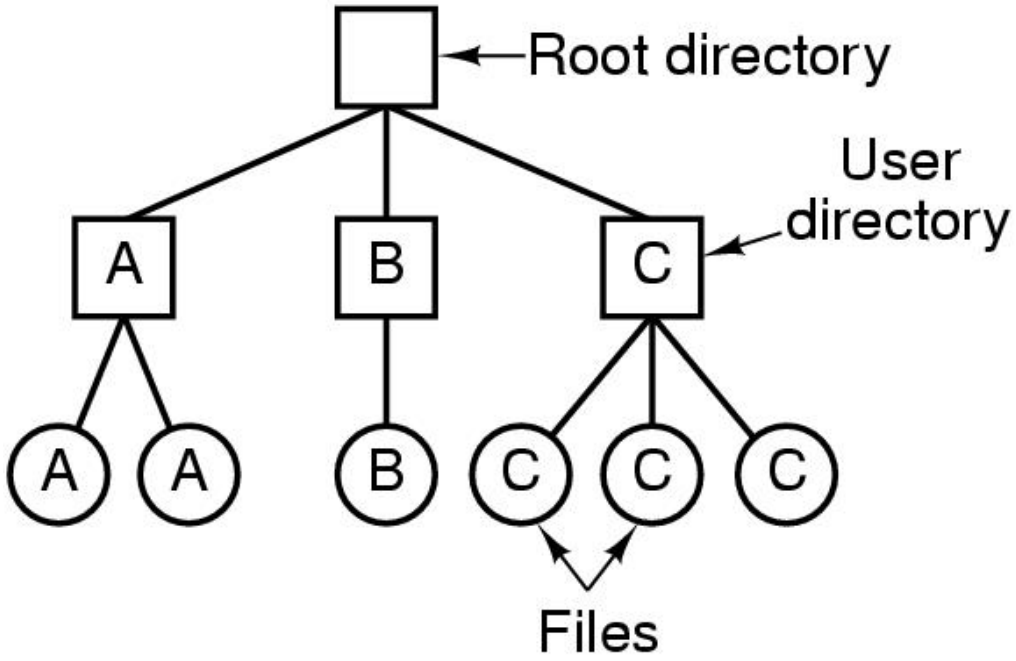
Single-Level Directory Systems



- **A single level directory system**
  - **contains 4 files**
  - **owned by 3 different people, A, B, and C**

sun@hit.edu.cn

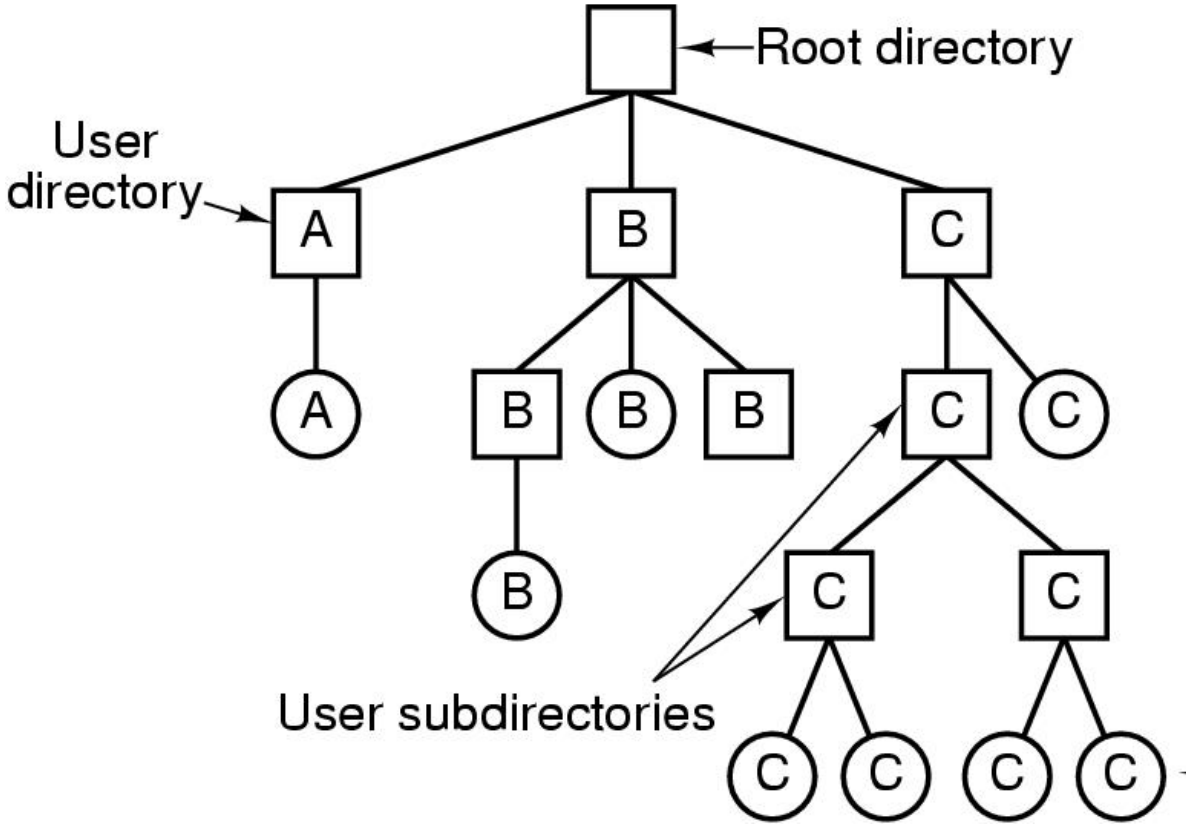
Two-level Directory Systems



Letters indicate *owners* of the directories and files

sun@hit.edu.cn

Hierarchical Directory Systems



A hierarchical directory system

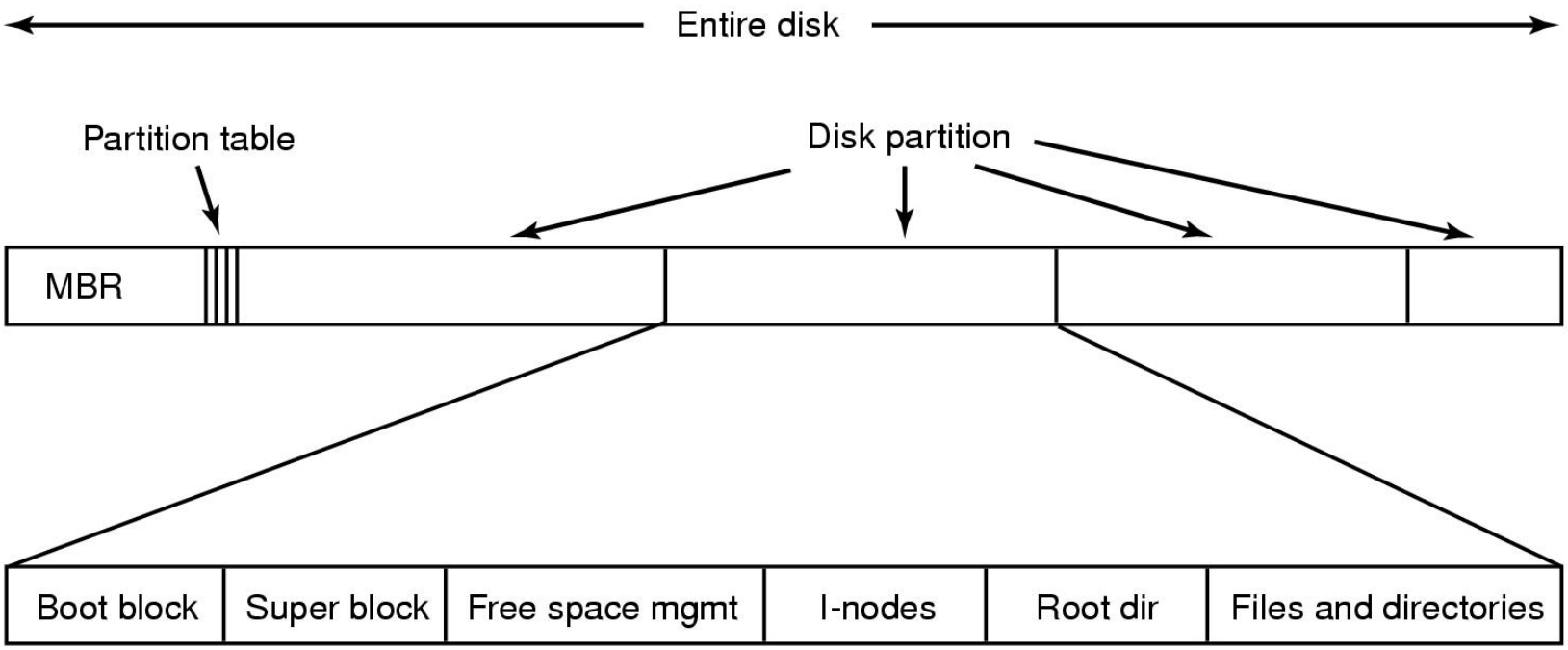
sun@hit.edu.cn

Directory Operations

1. Create
2. Delete
3. Opendir
4. Closedir
5. Readdir
6. Rename
7. Link
8. Unlink

sun@hit.edu.cn

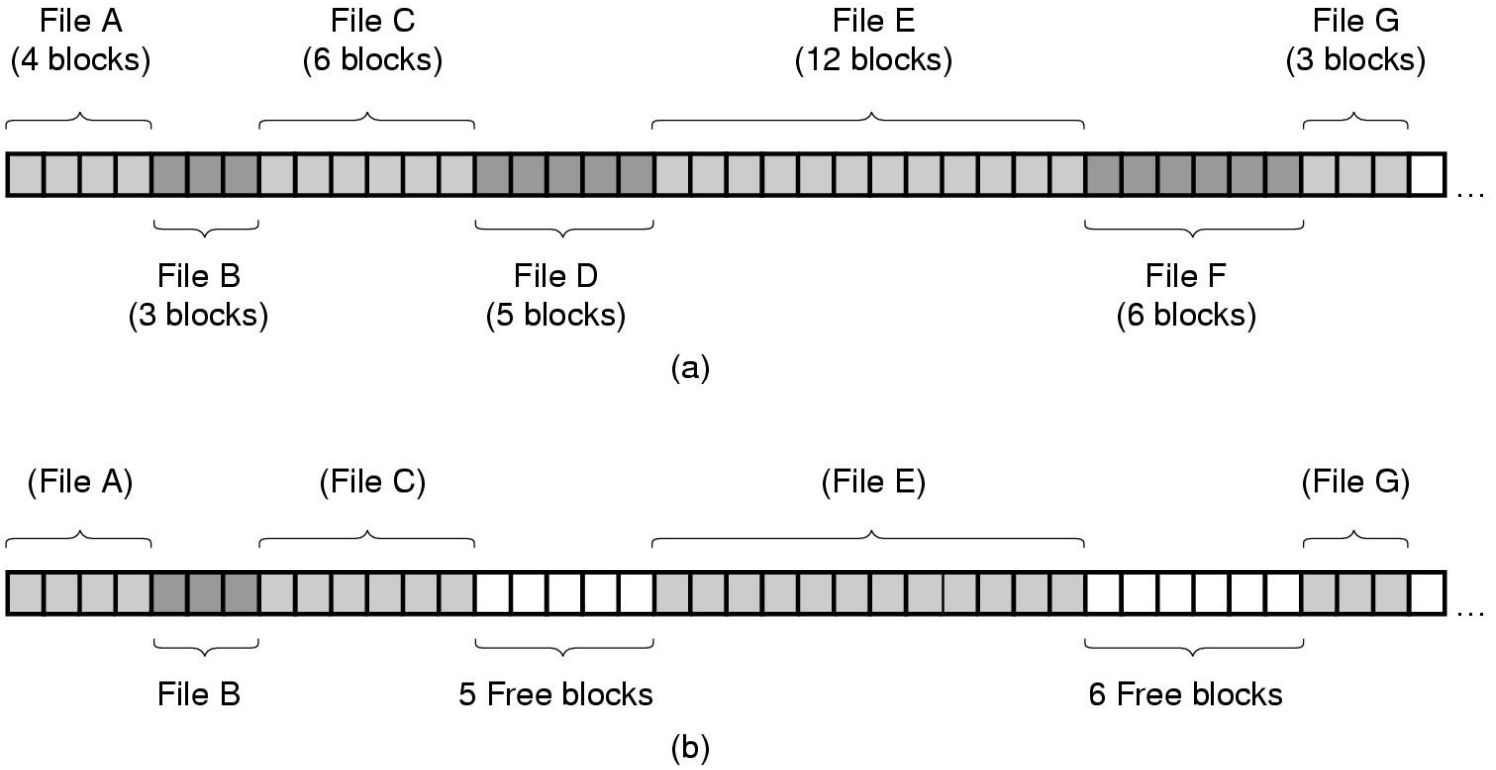
File System Implementation



A possible file system layout

sun@hit.edu.cn

Implementing Files (1)

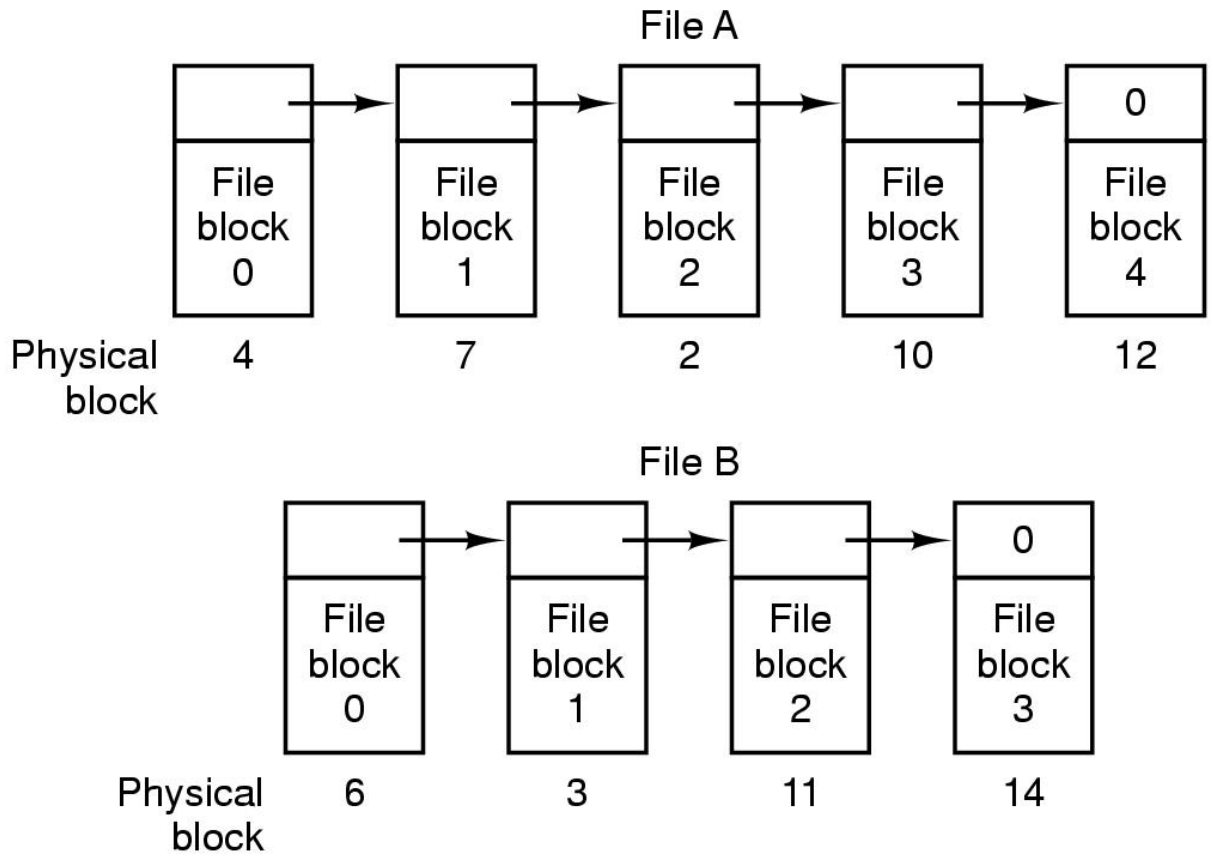


(a) Contiguous allocation of disk space for 7 files

(b) State of the disk after files *D* and *E* have been removed

sun@hit.edu.cn

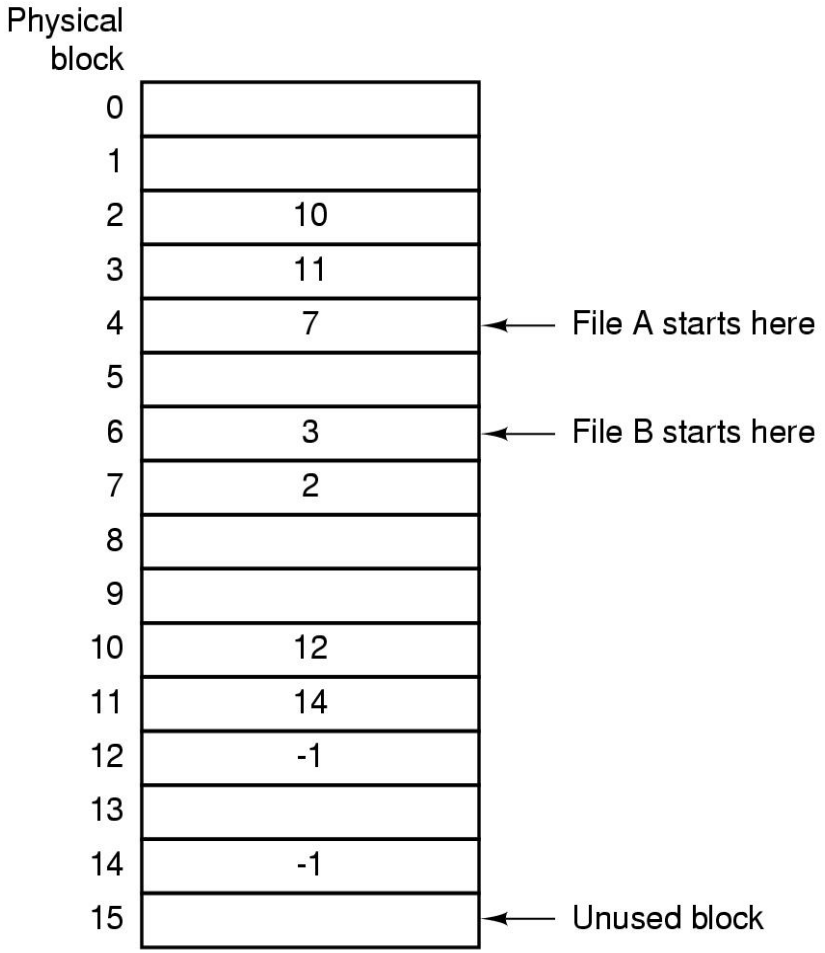
Implementing Files (2)



Storing a file as a linked list of disk blocks

sun@hit.edu.cn

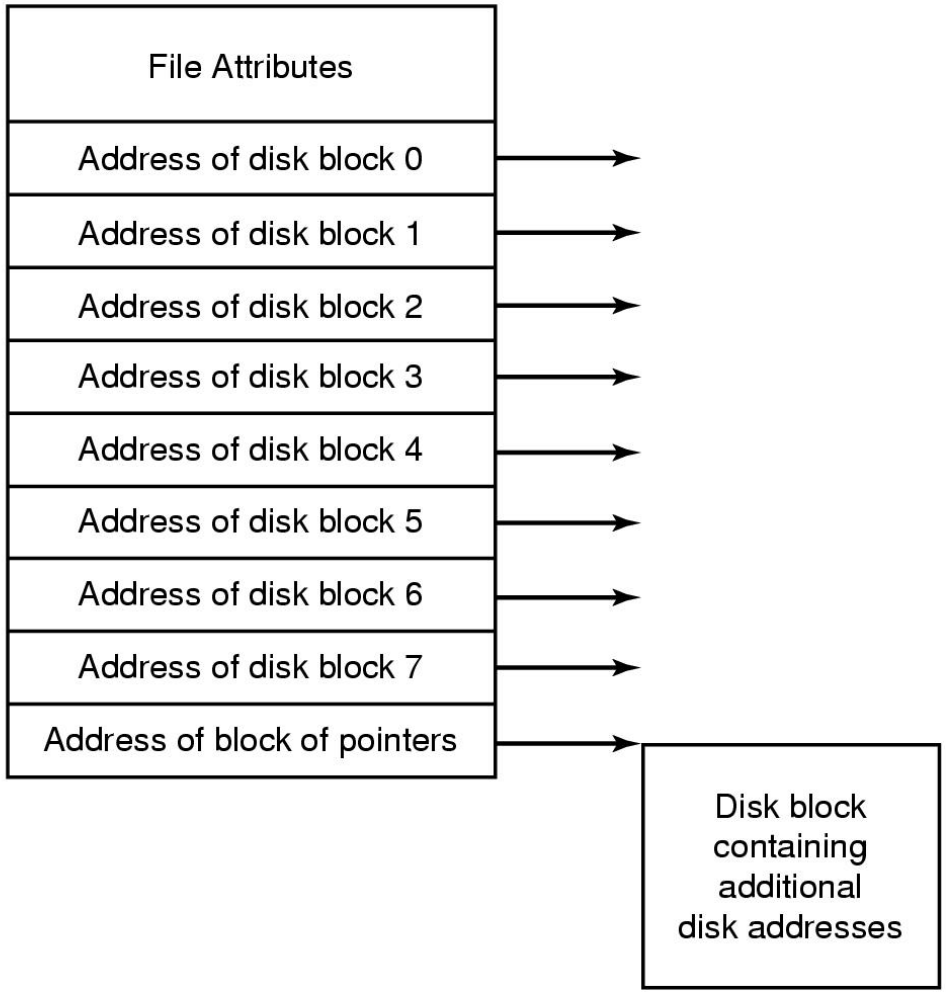
Implementing Files (3)



Linked list allocation using a File Allocation Table in RAM

sun@hit.edu.cn

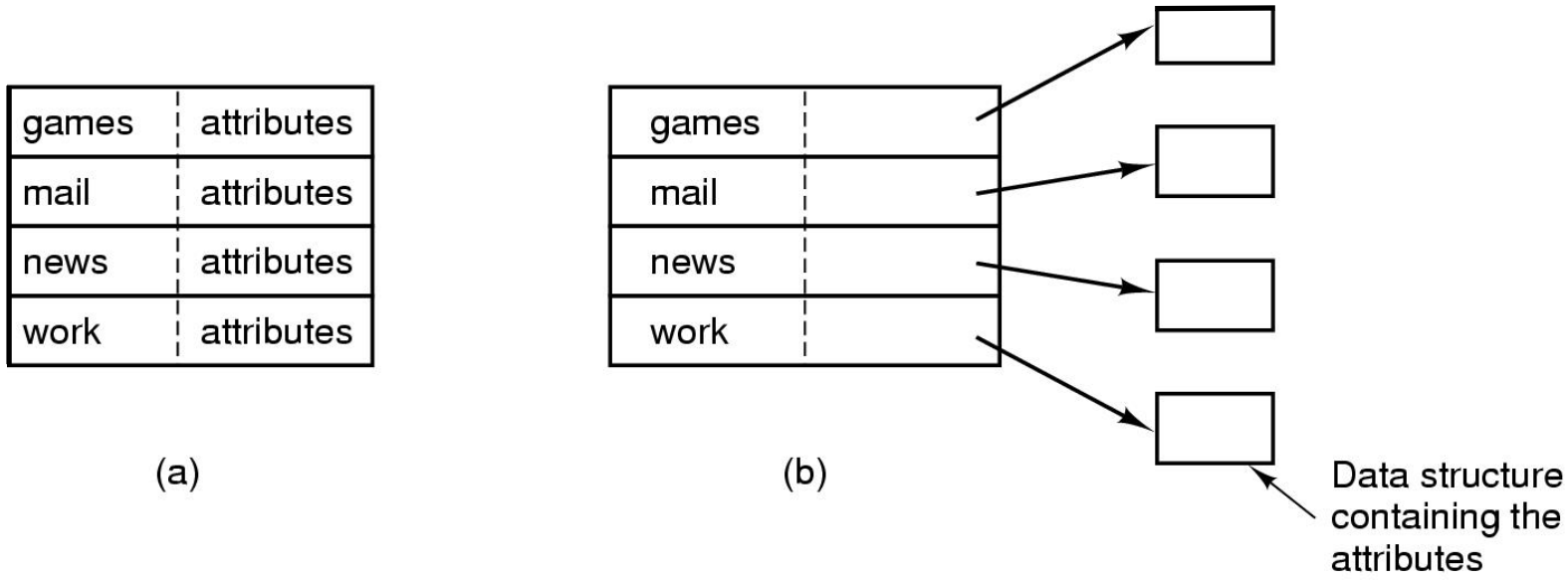
Implementing Files (4)



An example i-node

sun@hit.edu.cn

Implementing Directories (1)



(a) A simple directory

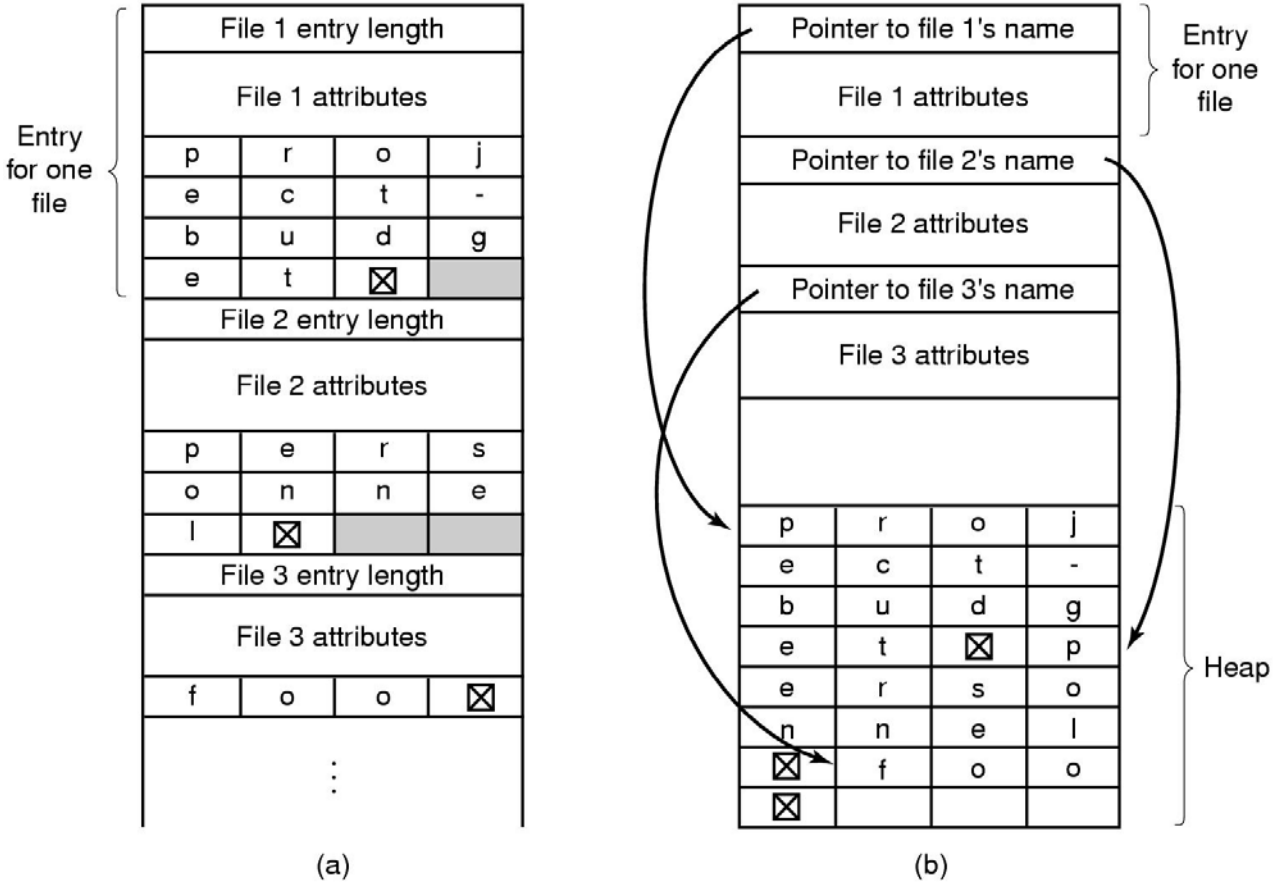
fixed size entries

disk addresses and attributes in directory entry

(b) Directory in which each entry just refers to an i-node

sun@hit.edu.cn

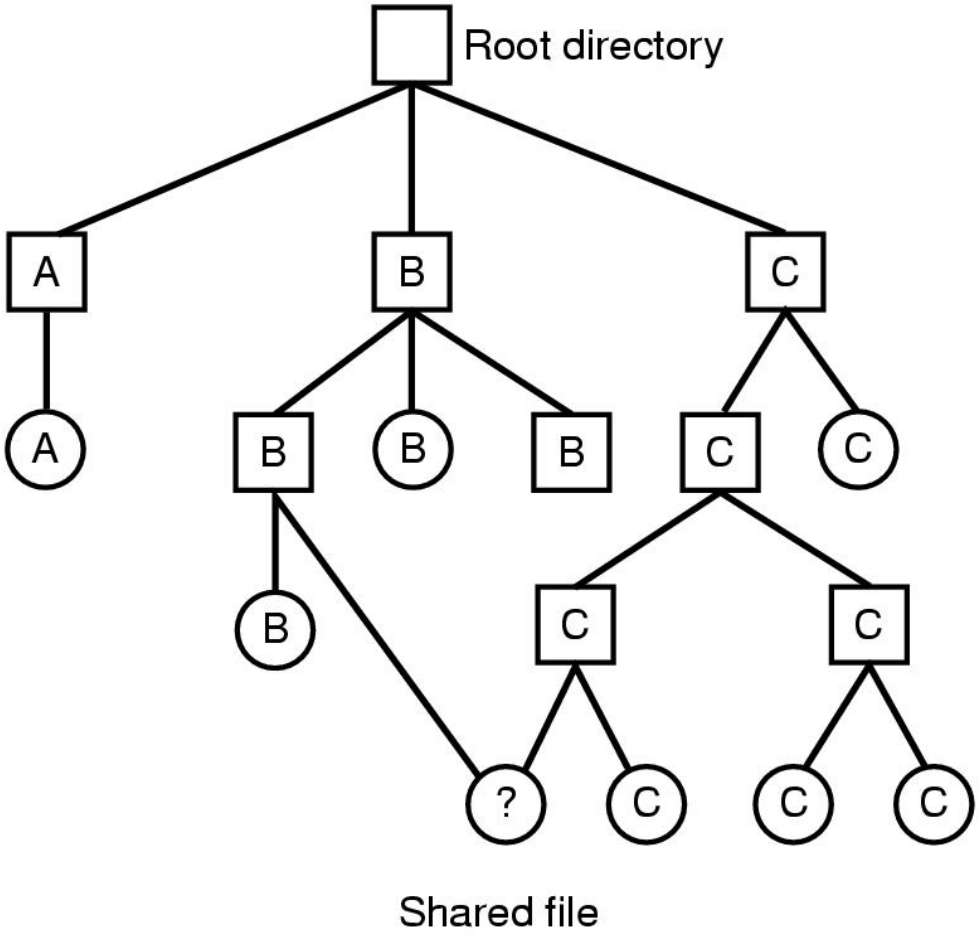
Implementing Directories (2)



- **Two ways of handling long file names in directory**
  - **(a) In-line**
  - **(b) In a heap**

sun@hit.edu.cn

Shared Files (1)



Shared file

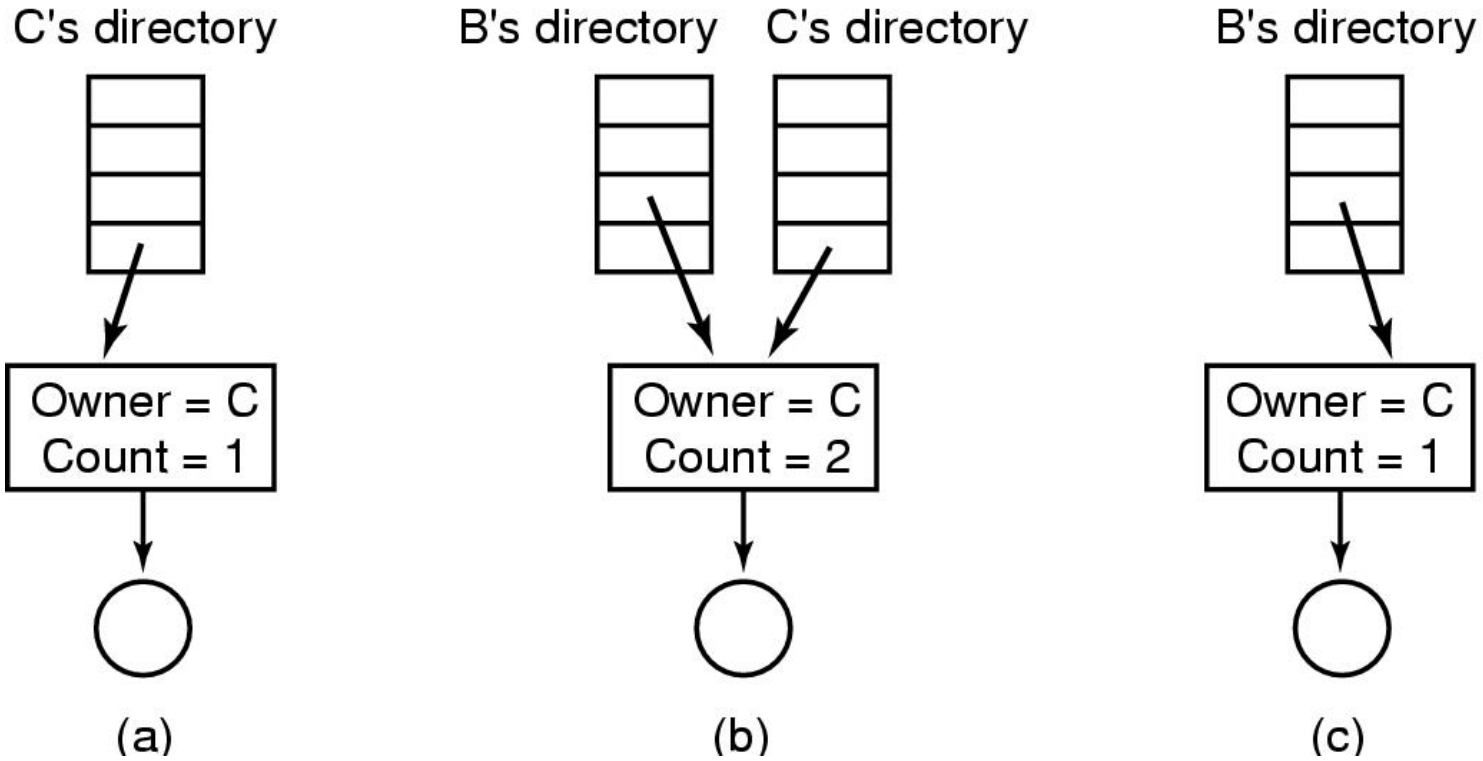
File system containing a shared file

sun@hit.edu.cn

Shared Files (2)

- **Hard link**
  - **the directories point to the little data structure associated with the file**
- **Symbolic link**
  - **create a new LINK file which contains just the path name of the file to which it is linked**

Shared Files (3)



(a) Situation prior to linking

(b) After the link is created

(c) After the original owner removes the file

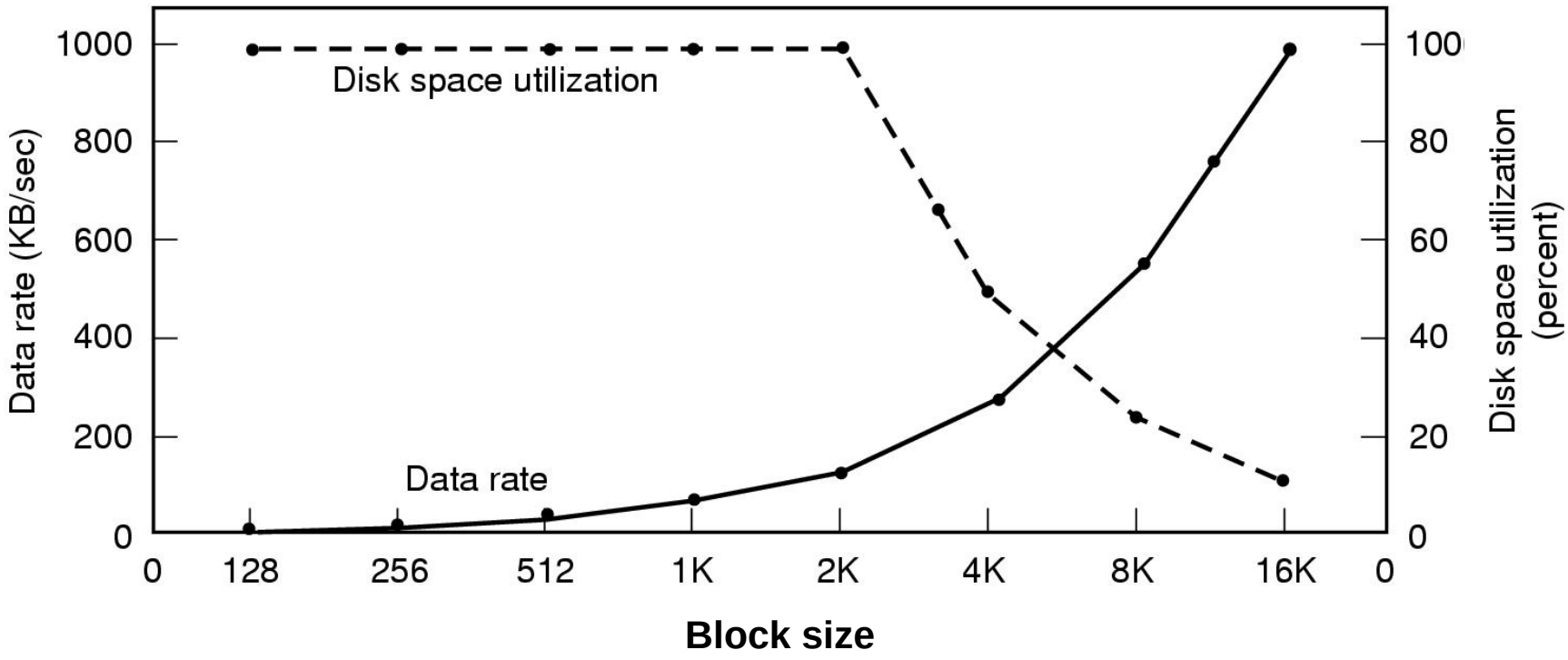
sun@hit.edu.cn

Disk Space Management
Block Size

- **An example**
  - **consider a disk with 131,072 bytes per track, a rotation time of 8.33ms, and an average seek time of 10ms. The time to read a block of *k* bytes is then the sum of the *seek*, *rotational delay*, and *transfer times*.**
- **Assume all files are 2KB**

sun@hit.edu.cn

Block Size



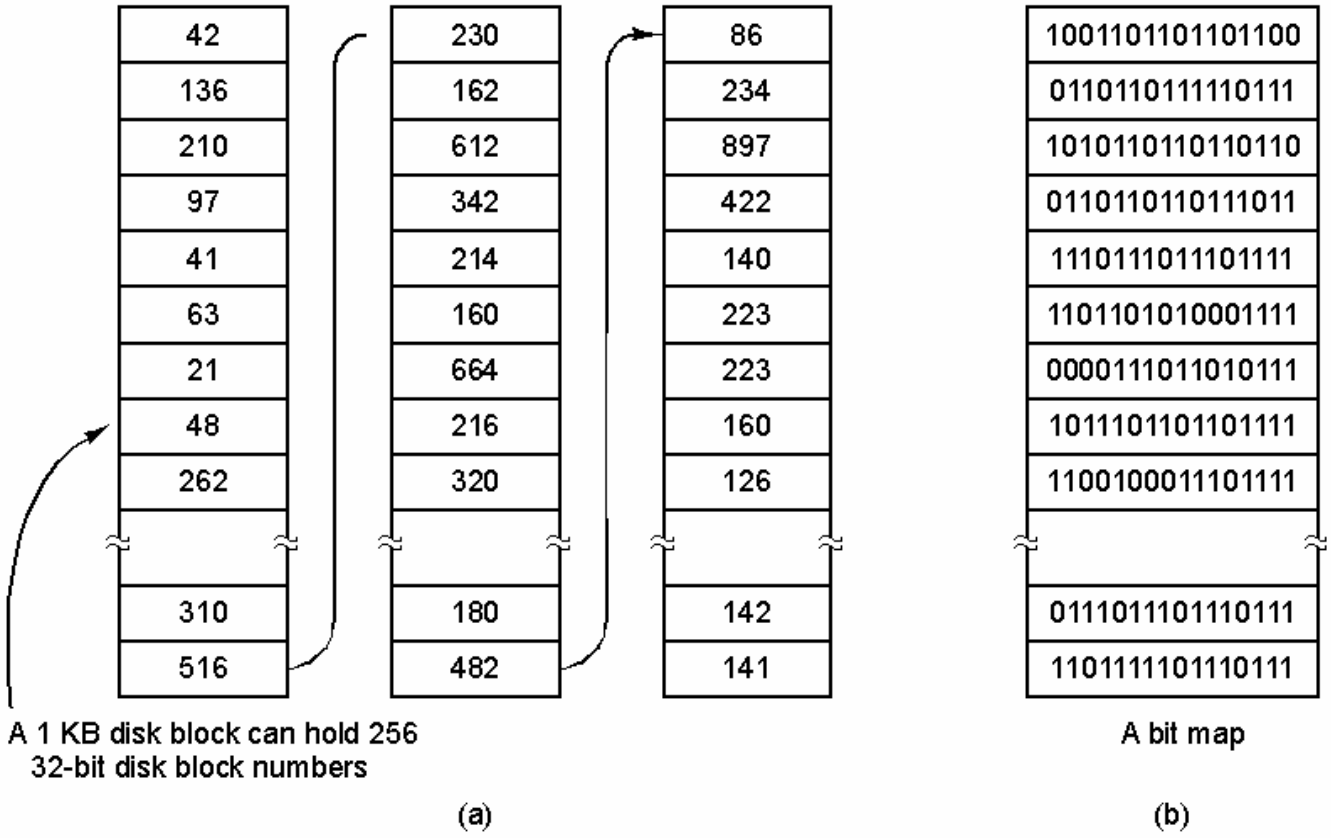
- Dark line (left hand scale) gives data rate of a disk
- Dotted line (right hand scale) gives disk space efficiency
- All files 2KB

sun@hit.edu.cn

Block Size

- For UNIX, 1 KB block is commonly used
- For MS-DOS family, the block size can be any power of two from 512 bytes to 32 KB which is determined by the disk size

Disk Space Management
Keeping Track of Free Blocks

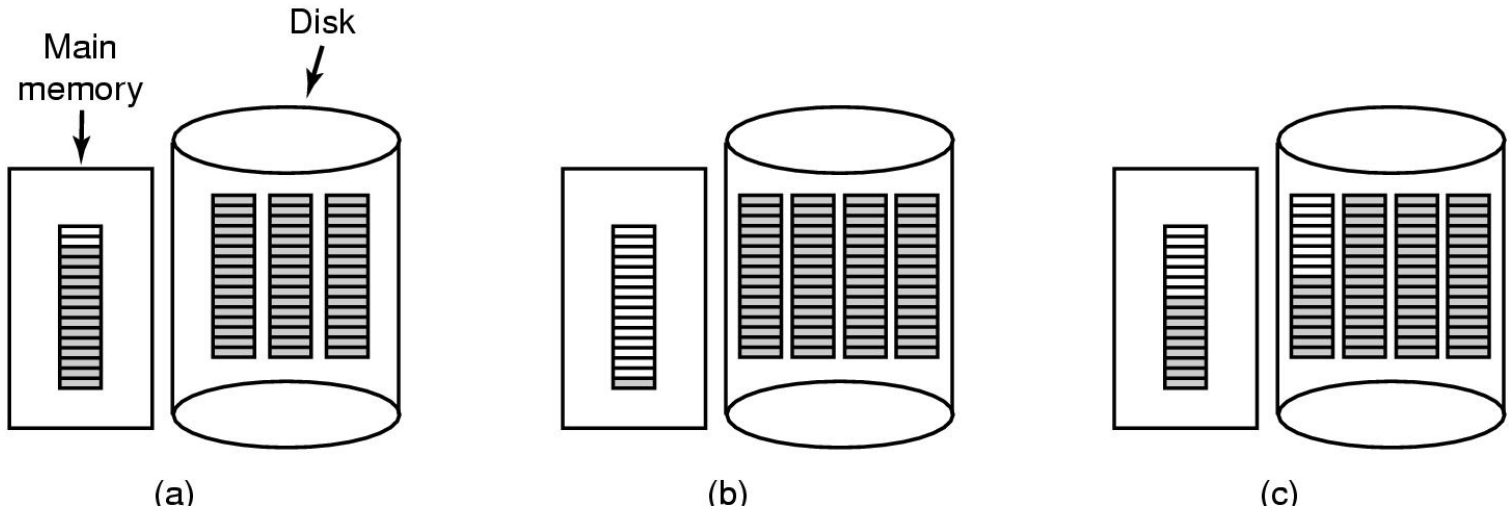


(a) Storing the free list on a linked list

(b) A bitmap

sun@hit.edu.cn

Linked List



(a) Almost-full block of pointers to free disk blocks in RAM

- three blocks of pointers on disk

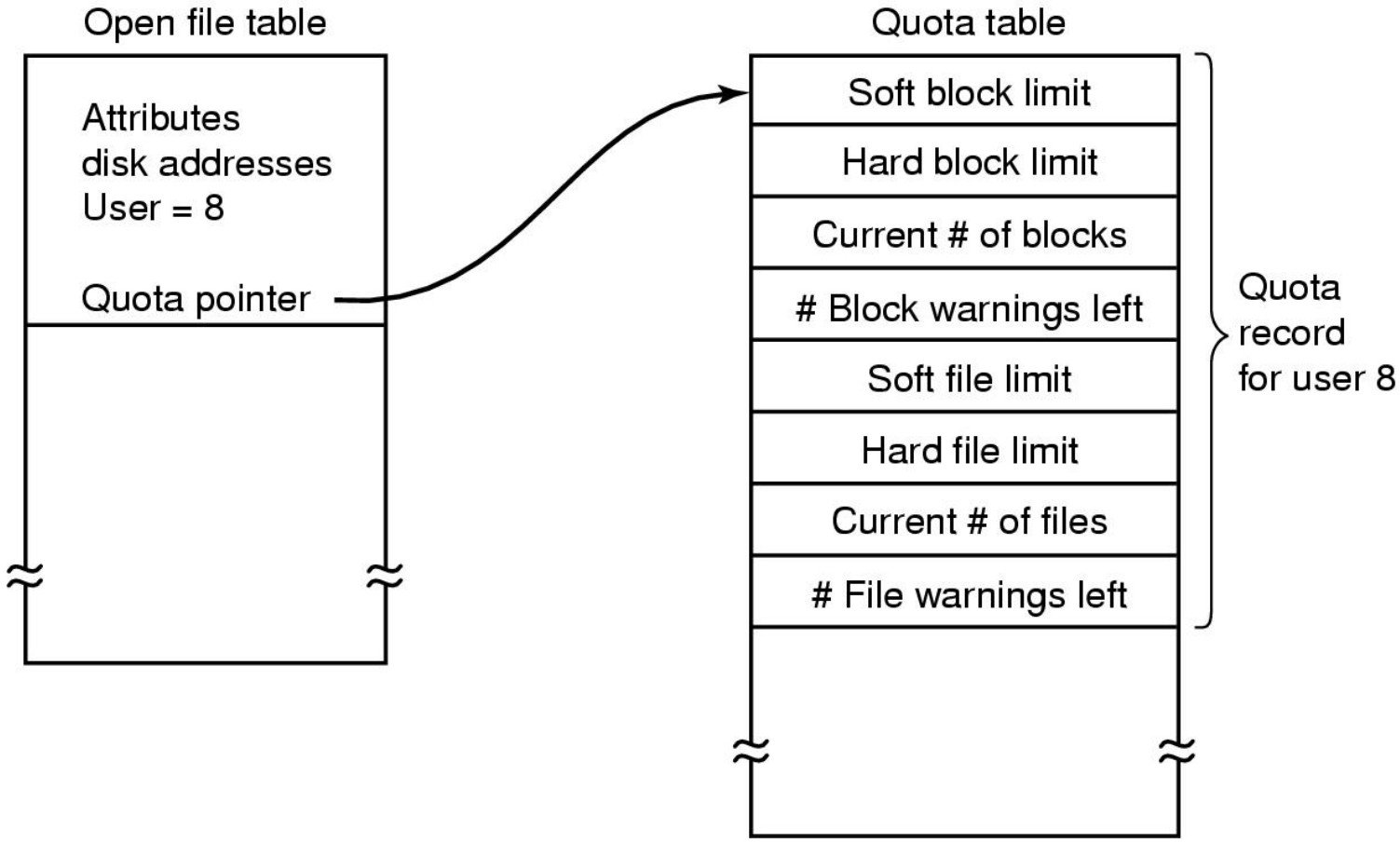
(b) Result of freeing a 3-block file

(c) Alternative strategy for handling 3 free blocks

- shaded entries are pointers to free disk blocks

sun@hit.edu.cn

Disk Space Management
Disk Quotas



Quotas for keeping track of each user's disk use

sun@hit.edu.cn

File System Reliability

- Data is more expensive than hardware
- Backup!
  - Recover from disaster
  - Recover from stupidity
- A backup takes a long time and occupies a large amount of space
  - backup only part of the file system
  - backup changed files
    - incremental dump
  - compression
  - difficult to backup an active file system
  - nontechnical problems

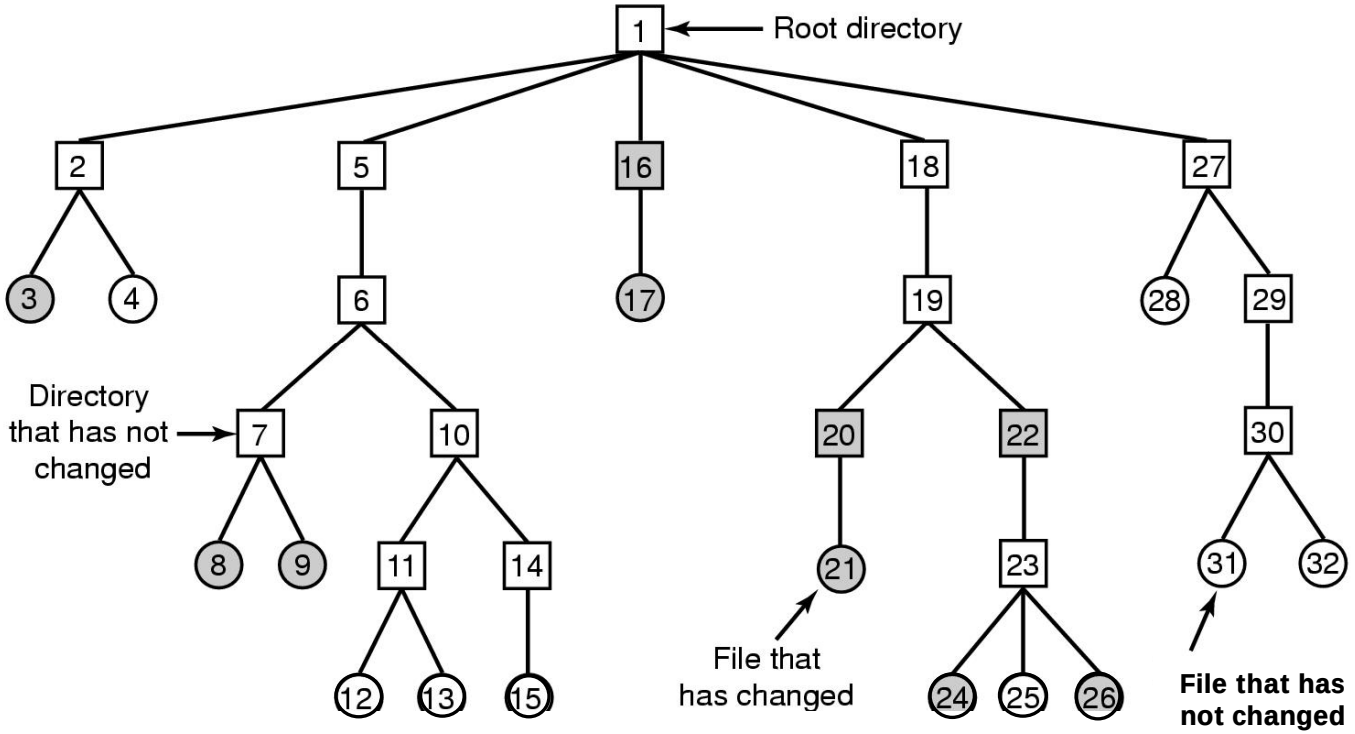
sun@hit.edu.cn

Dump Strategies

- **Physical Dump**
  - **easy**
  - **fast**
  - **not flexible**
- **Logical Dump**
  - **starts at one or more specified directories and recursively dumps all files and directories found there than have changed since some given base date**

sun@hit.edu.cn

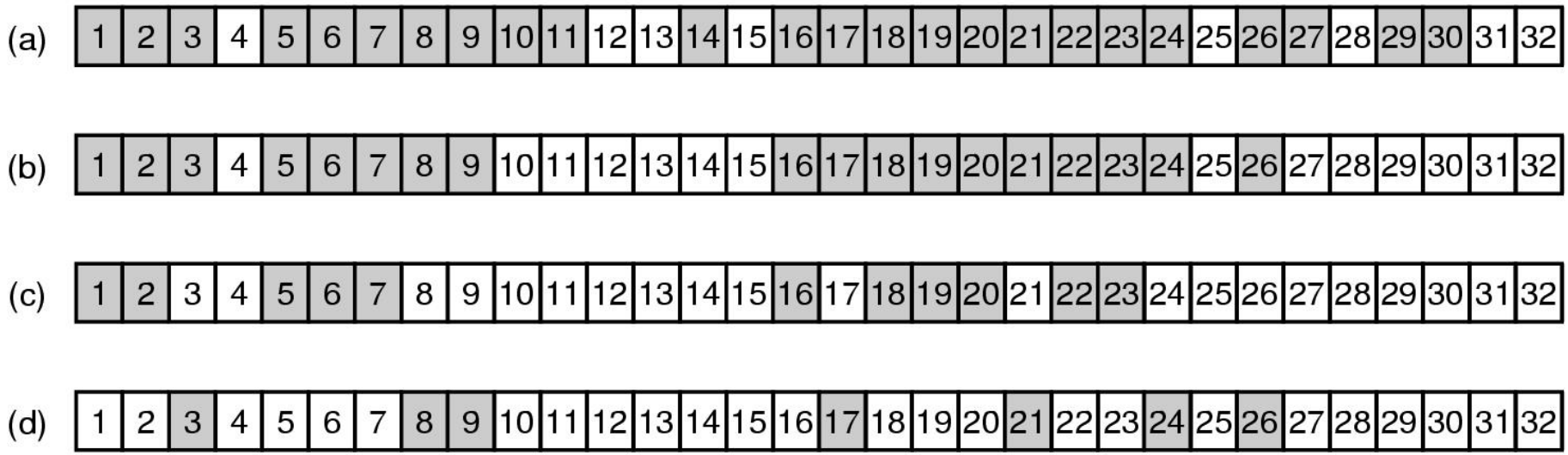
An Algorithm about Logical Dump



- **A file system to be dumped**
  - **squares are directories, circles are files**
  - **shaded items, modified since last dump**
  - **each directory & file labeled by i-node number**

sun@hit.edu.cn

An Algorithm about Logical Dump



Bit maps used by the logical dumping algorithm

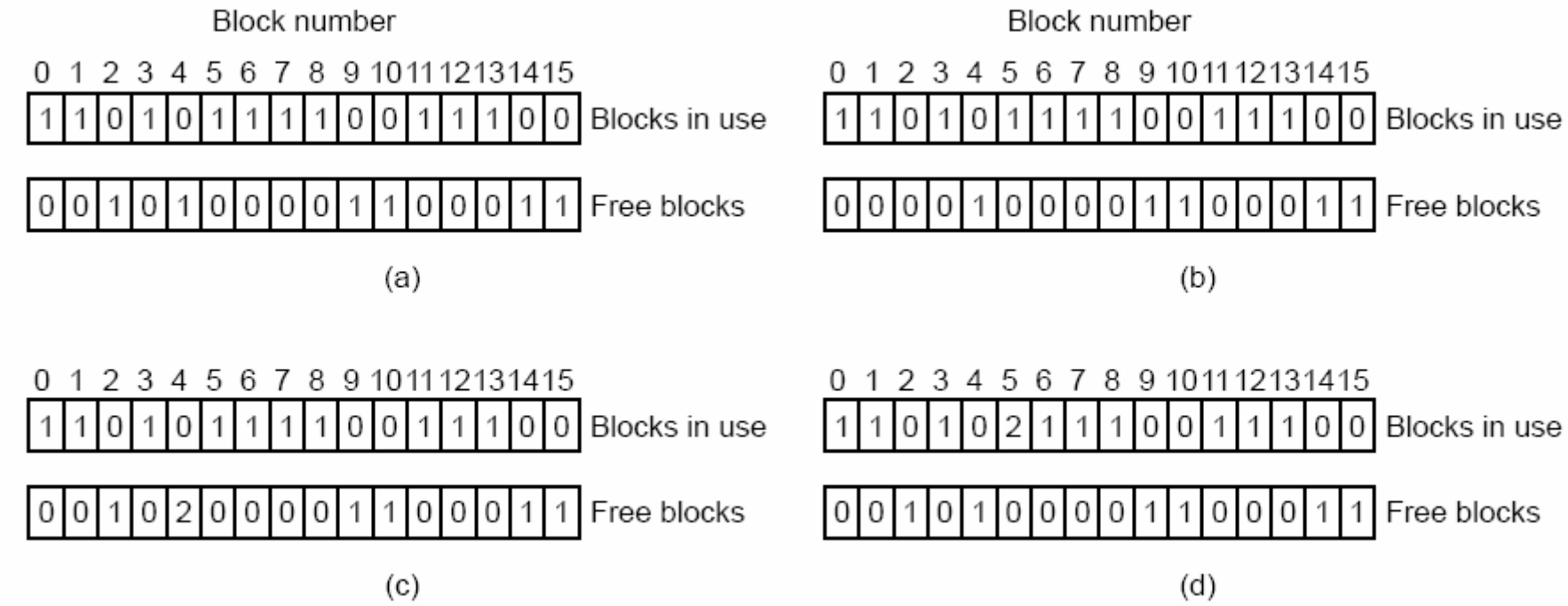
sun@hit.edu.cn

An Algorithm about Logical Dump

- **Restore**
  - create an empty file system
  - restore the most recent full dump
    - directories are all restored first
  - the files are restored
  - repeated this process with the following incremental dump

sun@hit.edu.cn

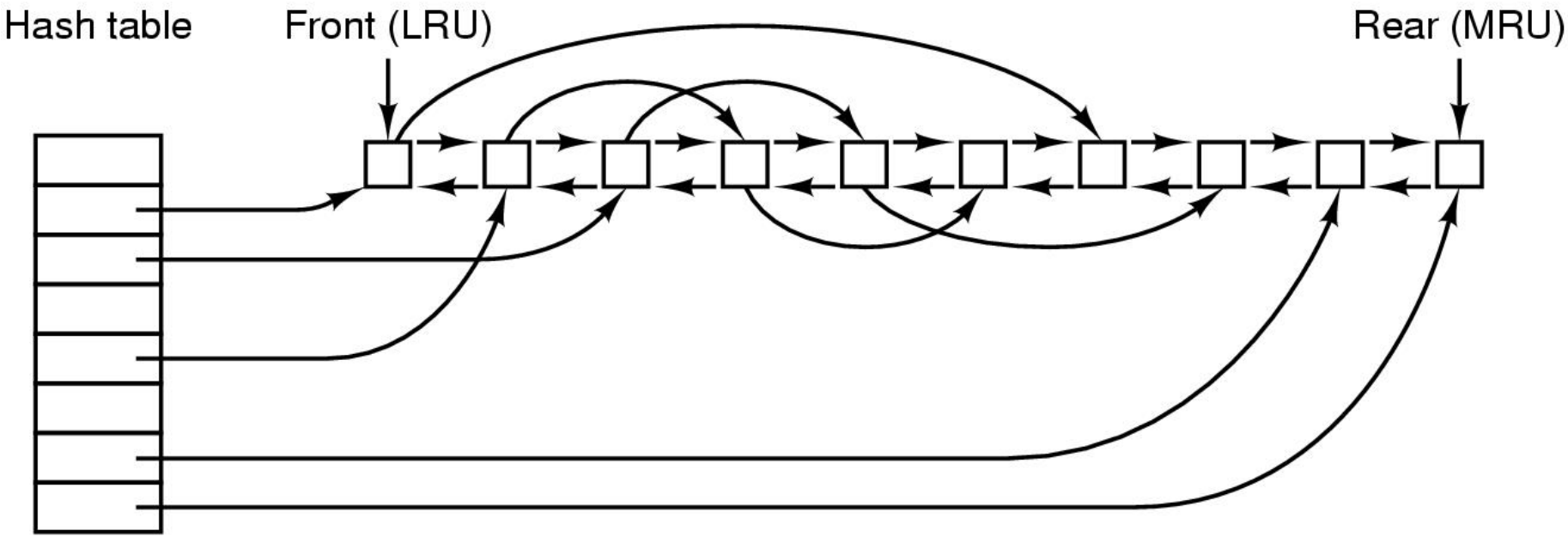
File System Consistency



- **File system states**
  - **(a) consistent**
  - **(b) missing block**
  - **(c) duplicate block in free list**
  - **(d) duplicate data block**

sun@hit.edu.cn

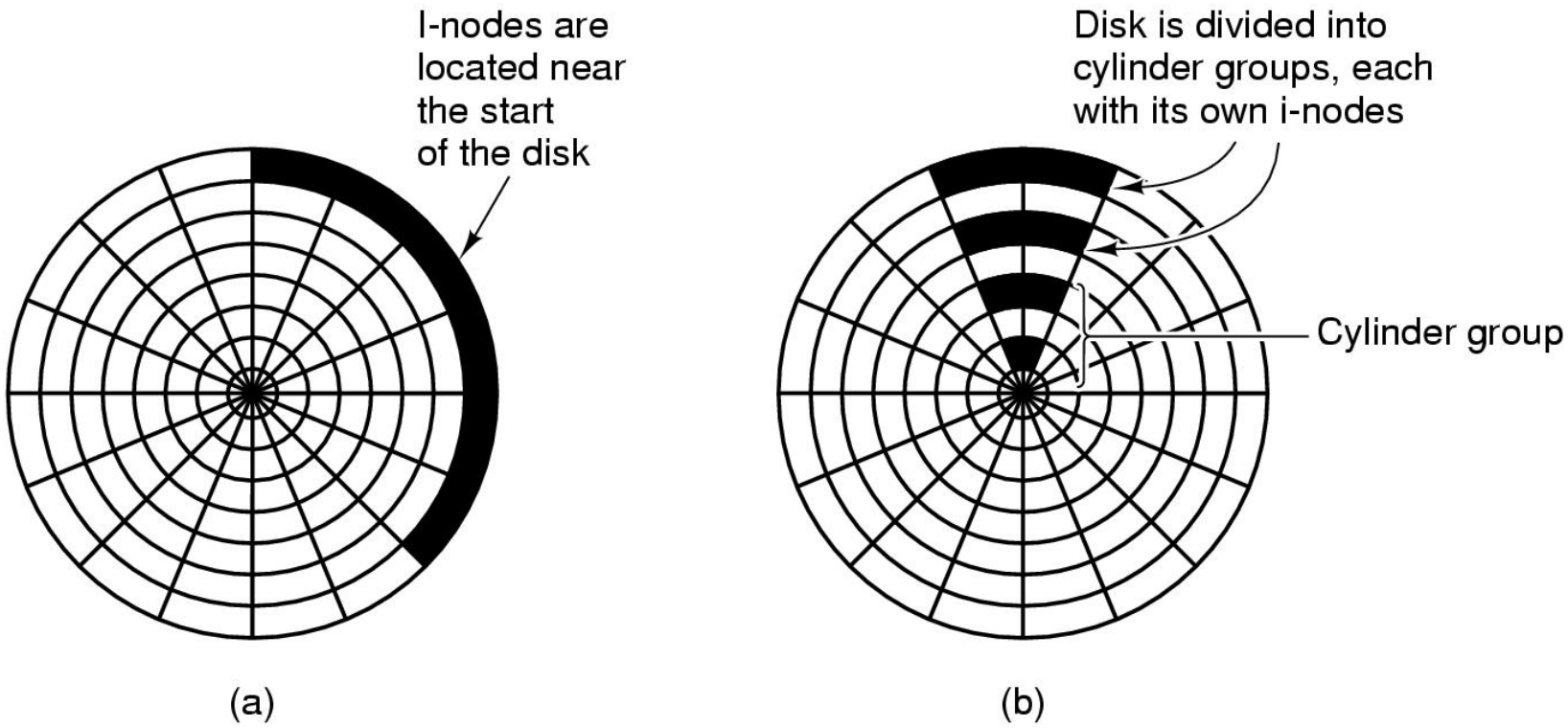
File System Performance
Caching



The block cache data structures

sun@hit.edu.cn

Reducing Disk Arm Motion



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

sun@hit.edu.cn

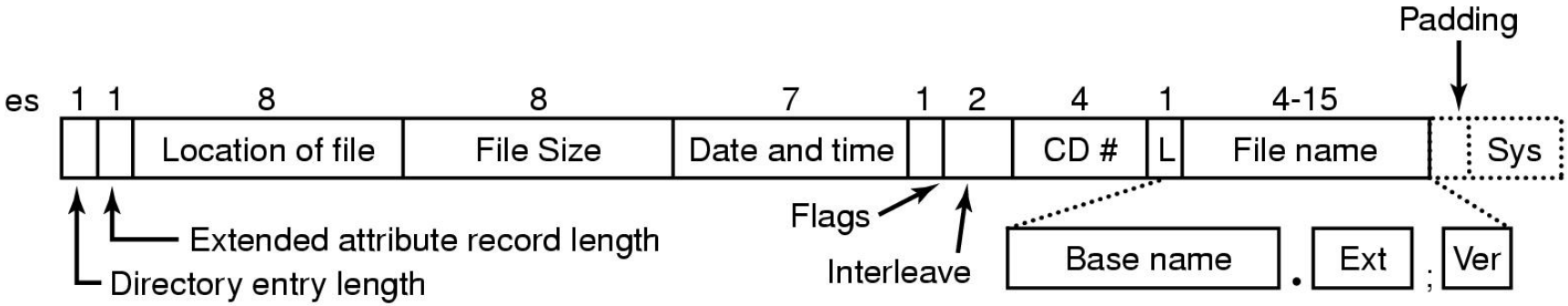
Log-Structured File Systems

- **With CPUs faster, memory larger**
  - **disk caches can also be larger**
  - **increasing number of read requests can come from cache**
  - **thus, most disk accesses will be writes**

- **LFS Strategy structures entire disk as a log**
  - **have all writes initially buffered in memory**
  - **periodically write these to the end of the disk log**
  - **when file opened, locate i-node, then find blocks**

sun@hit.edu.cn

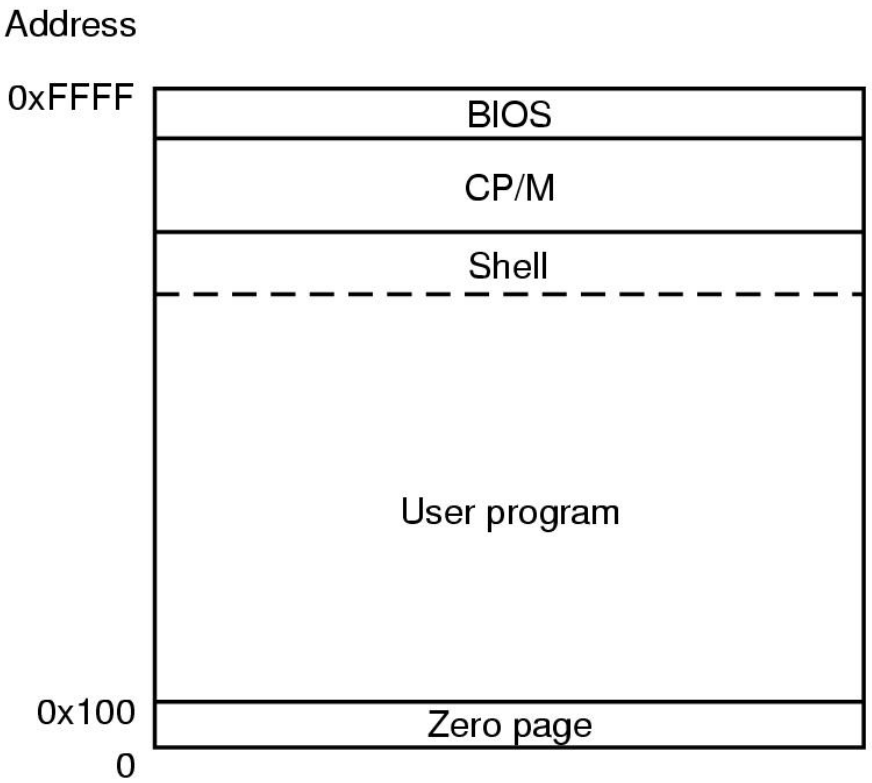
Example File Systems
CD-ROM File Systems



The ISO 9660 directory entry

sun@hit.edu.cn

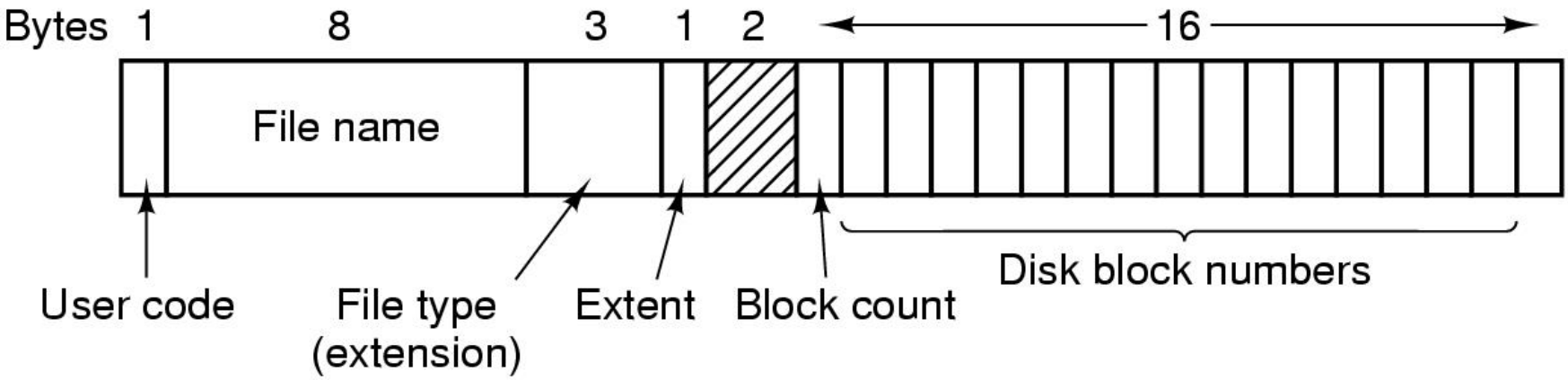
The CP/M File System (1)



Memory layout of CP/M

sun@hit.edu.cn

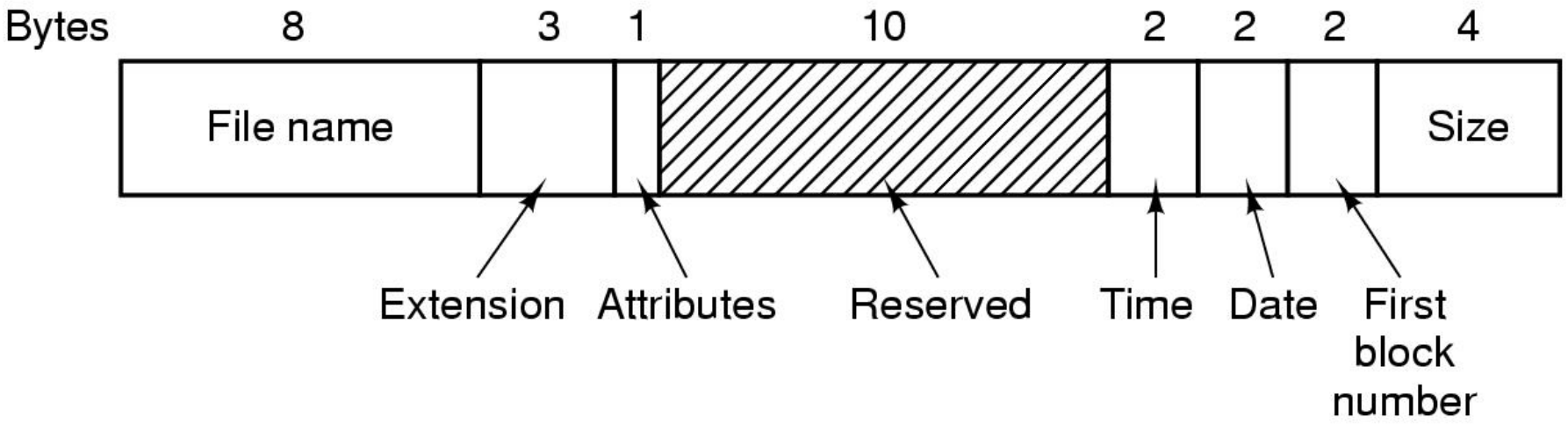
The CP/M File System (2)



The CP/M directory entry format

sun@hit.edu.cn

The MS-DOS File System (1)



The MS-DOS directory entry

sun@hit.edu.cn

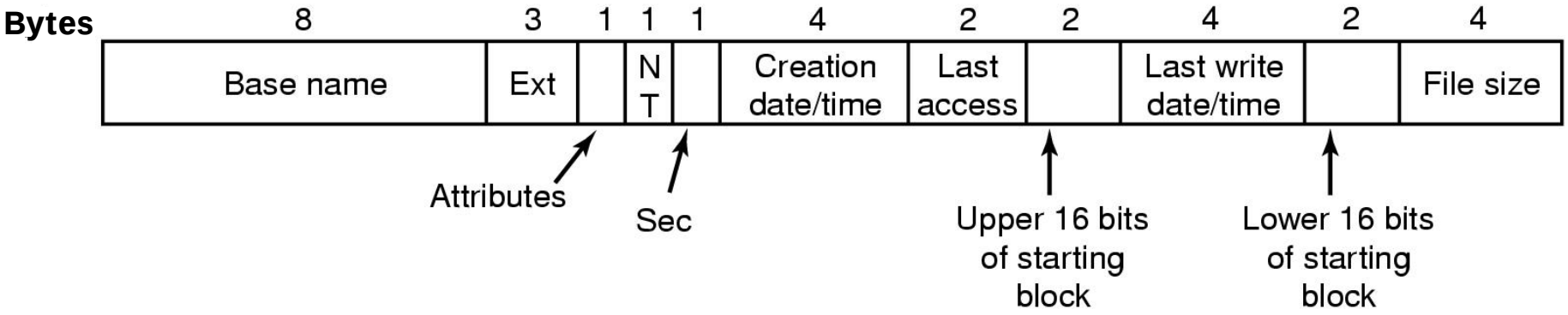
The MS-DOS File System (2)

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

sun@hit.edu.cn

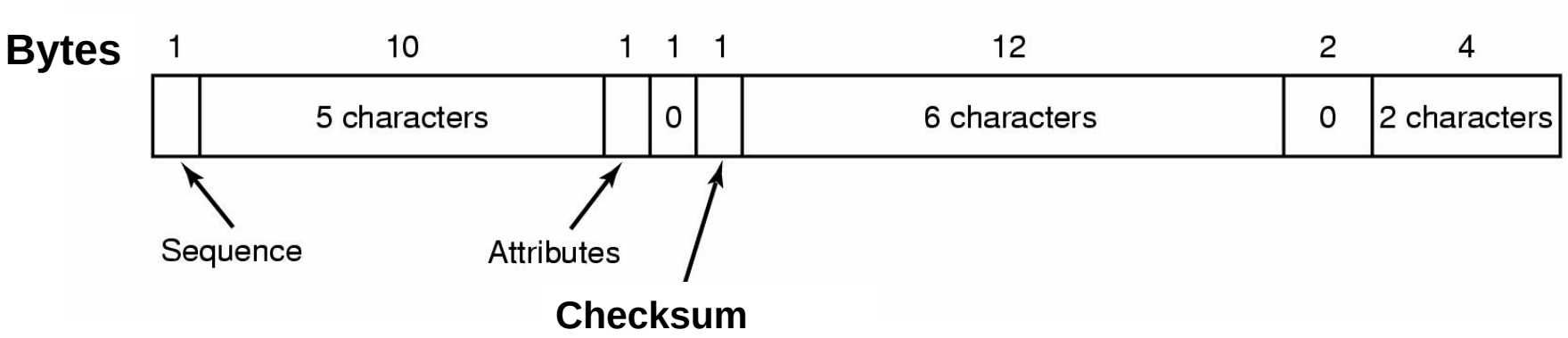
The Windows 98 File System (1)



The extended MOS-DOS directory entry used in Windows 98

sun@hit.edu.cn

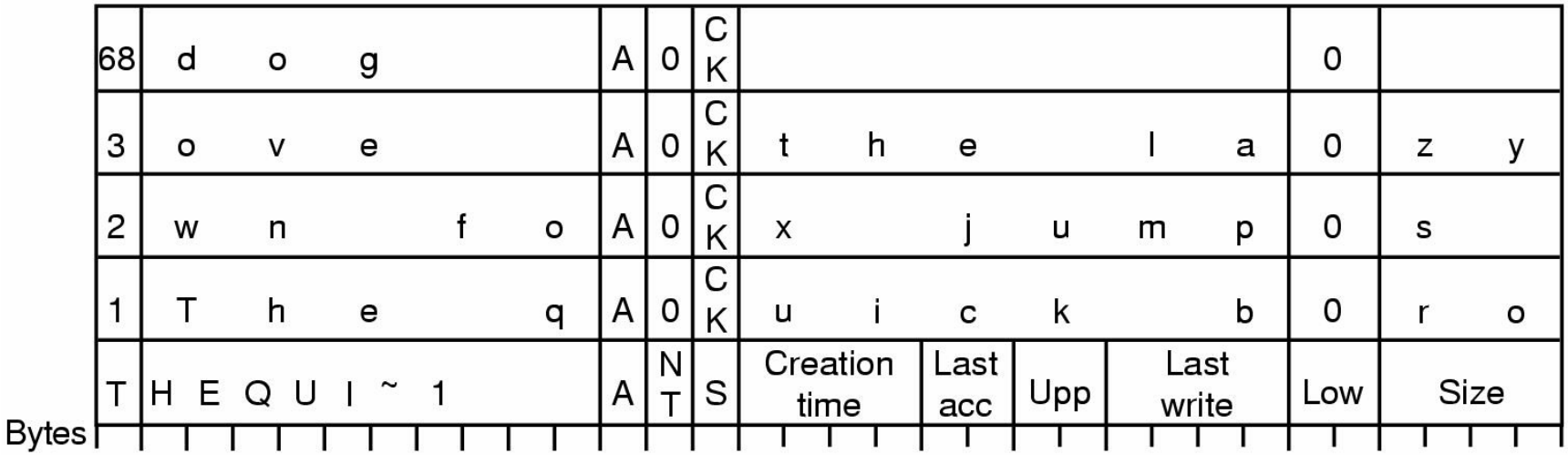
The Windows 98 File System (2)



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

sun@hit.edu.cn

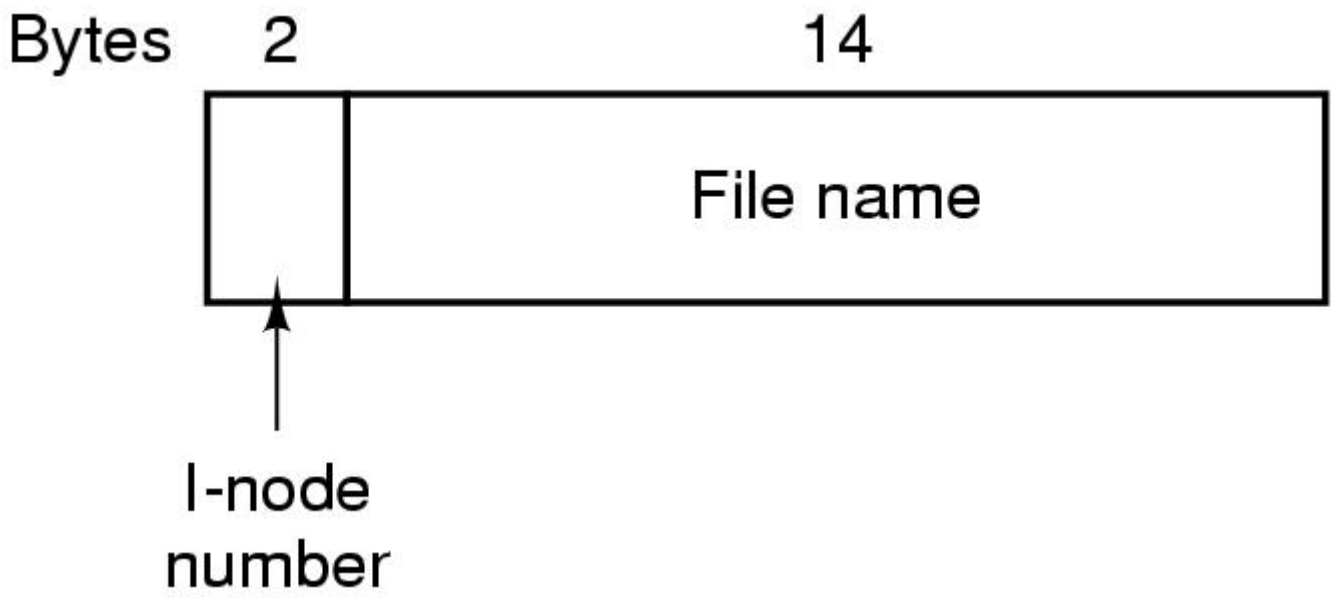
The Windows 98 File System (3)



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

sun@hit.edu.cn

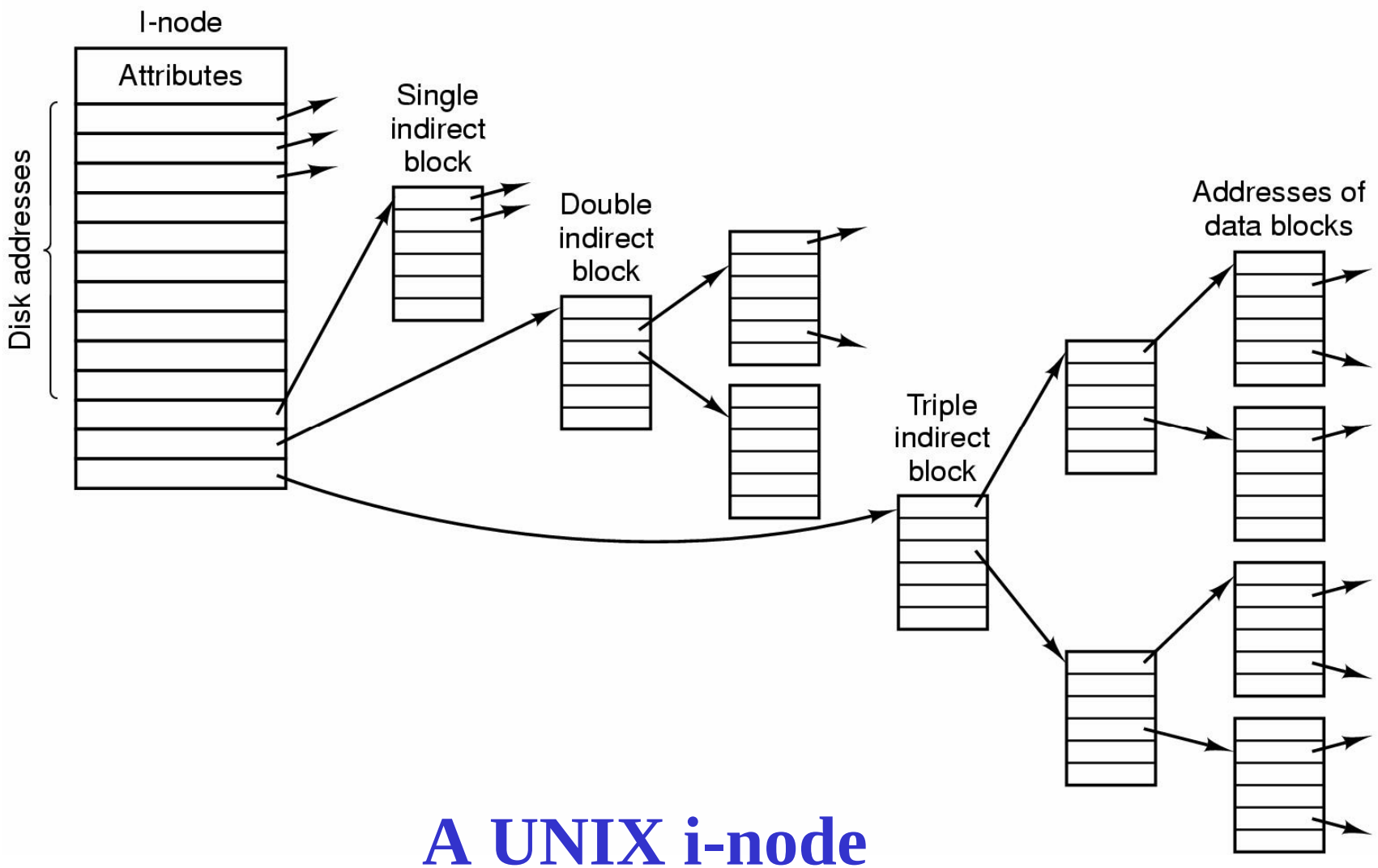
The UNIX V7 File System (1)



A UNIX V7 directory entry

sun@hit.edu.cn

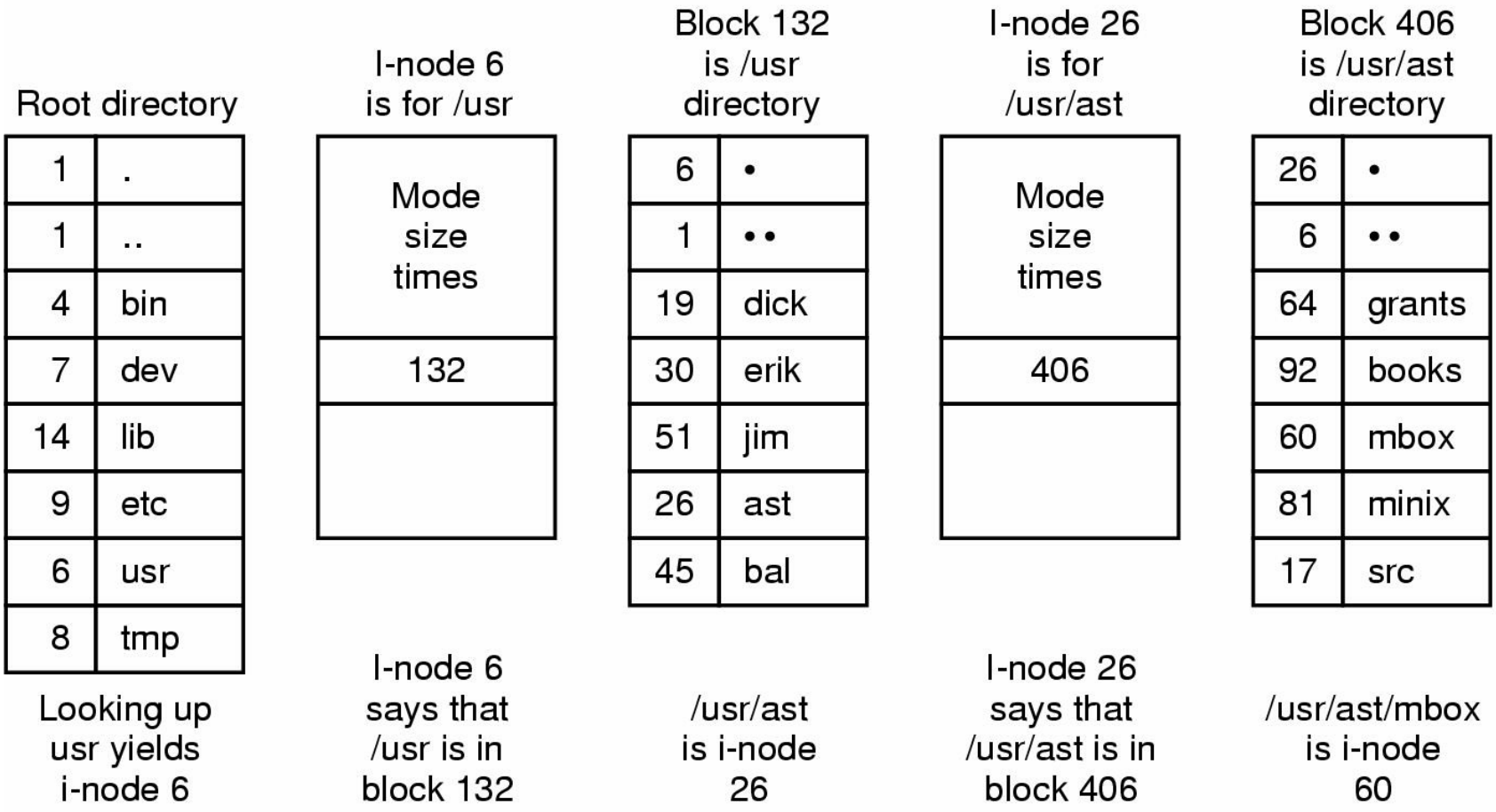
The UNIX V7 File System (2)



A UNIX i-node

sun@hit.edu.cn

The UNIX V7 File System (3)



The steps in looking up */usr/ast/mbox*

sun@hit.edu.cn