File Systems

Operating Systems

Assoc. Prof. Milos Jovanovik, PhD

File Systems

- What is necessary for a long-term information store?
  - Big quantity of information must be stored.
  - The information must exist even after the execution of the process that uses it.
  - Multiple processes should be able to simultaneously access the information.

File Systems

- Imagine the disk as a linear sequence of fixed-size blocks, which supports reading and writing data on the blocks.
- Questions:
  - How do you find information?
  - How do you keep one user from reading another user's data?
  - How do you know which blocks are free?

Files

Files

- Processes (threads), address spaces and files are the most important concepts in OS
- Files are logical information units, created by a process
  - Similar to address spaces
- File system
  - Has the task of file management: how the files are structured, accessed, named, used, protected, implemented, etc.

File Names

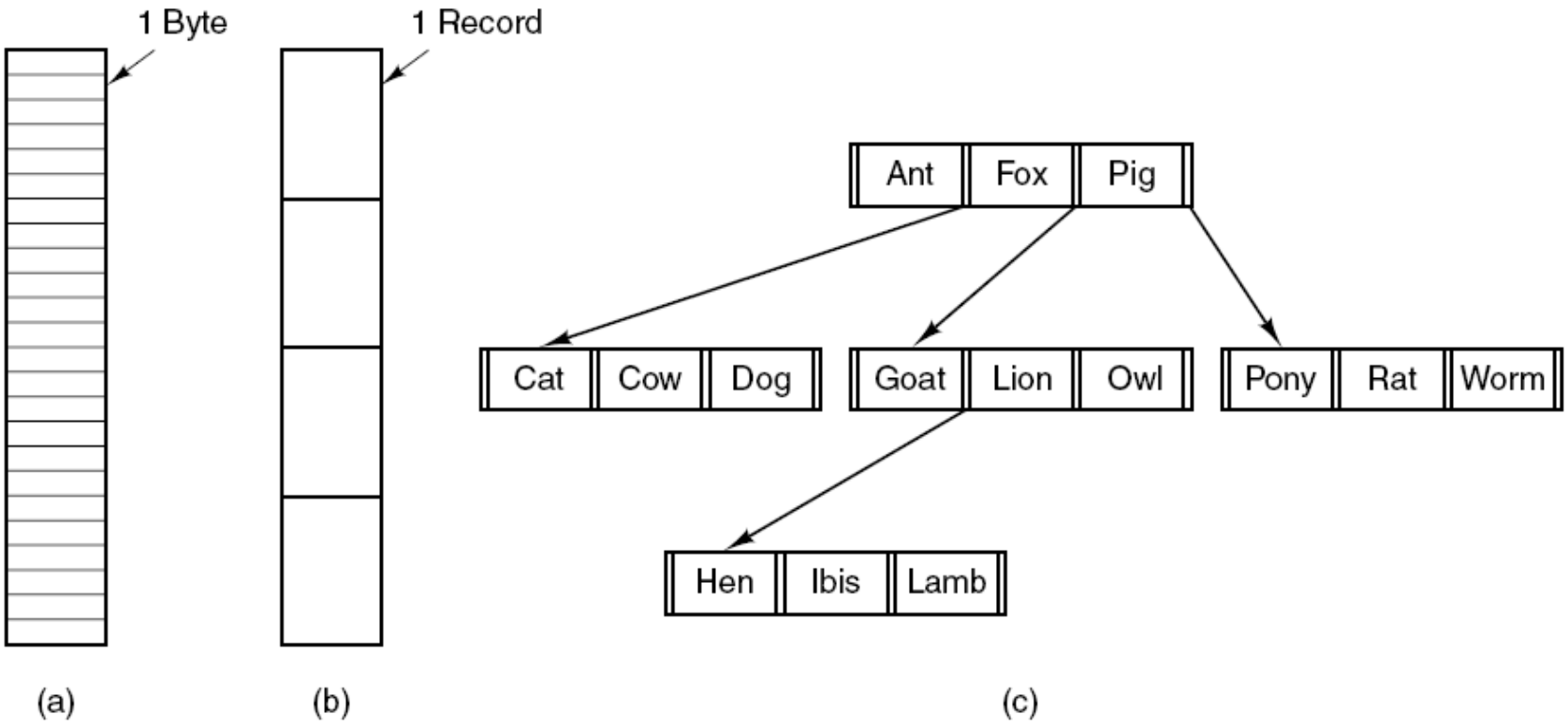
- Files are an abstract mechanism
  - In order to save the data on the disk and read it later
  - When one process creates a file, it must name it in order to access it later
- The name consists of two parts:
  - **Name**: the file name
  - **Extension**: the file characteristics
    - In UNIX, it is just a convention; the C compiler is an exception
    - In Windows, with the extension we mark which program can this file be opened with; a double click starts the corresponding program

File Names

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

Figure 4-1. Some typical extensions.

File Structure



Three types of files:

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

File Types

- Regular:
  - ASCII or binary files
  - ASCII files contain text, which can be shown and printed as-is, and can be edited with any text editor
  - Binary files have an internal structure known to the programs that use them
- Directories
  - Files in which other files are organized

File Types

- Character special files (files for character devices)
  - Connect to I/O and model serial I/O devices
    - Terminals, printers and networks
- Block special files (files for block devices)
  - They model disks

Types of Regular Files

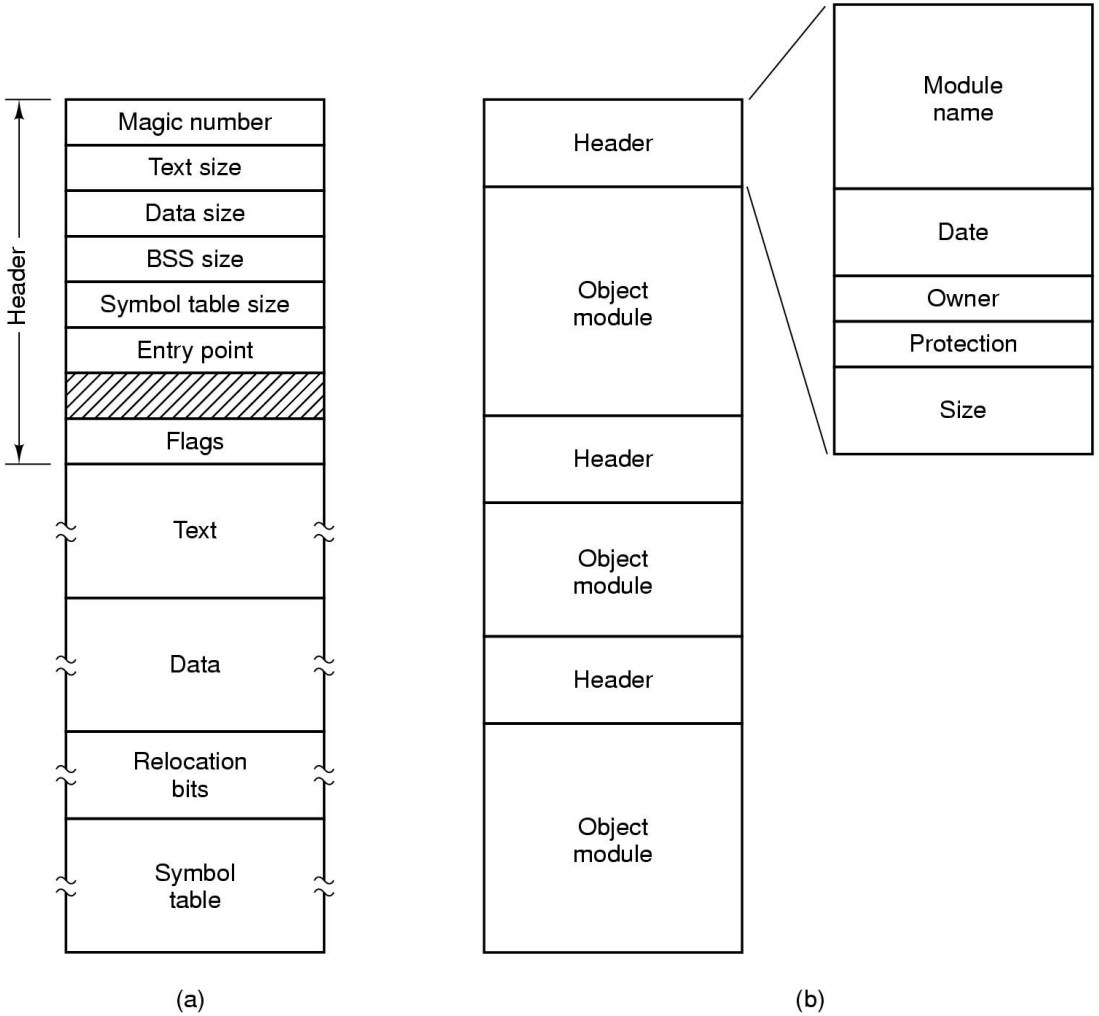


Figure 4-3. (a) Executable file. (b) Archive.

File Access

- File descriptor
  - A small integer which represents an object managed by the kernel, in/from which the process can write/read
  - Each process has a private space of file descriptors, starting from 0
  - Usually, 0 is standard input (stdin), 1 is standard output (stdout), and 2 is standard error (stderr)
- System calls: read() and write() to read from or write in the files associated with the file descriptors

File Access

- Sequential access
  - All the bytes / records are read from the beginning until the end of the file
  - There is no random access / jump to an arbitrary location
  - Magnetic tapes
- Random access
  - Bytes/records are read arbitrarily
  - Necessary for databases
  - Hard disks

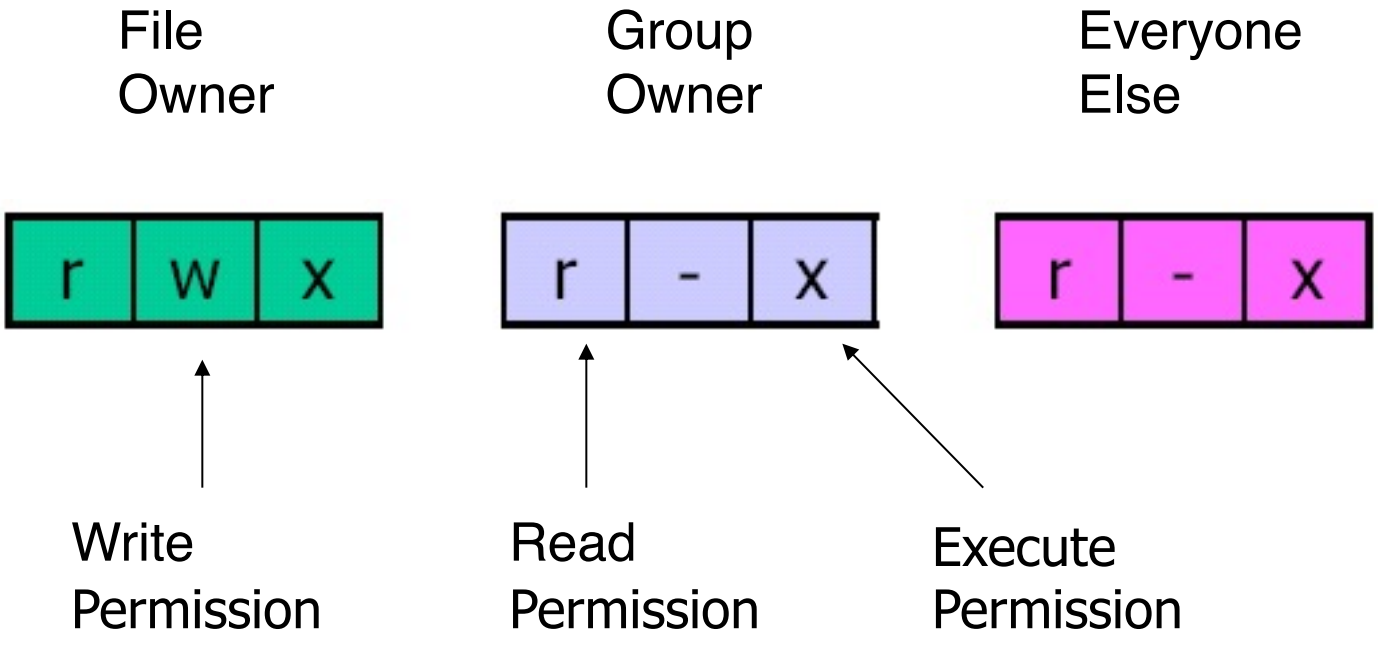
File Attributes (metadata)

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

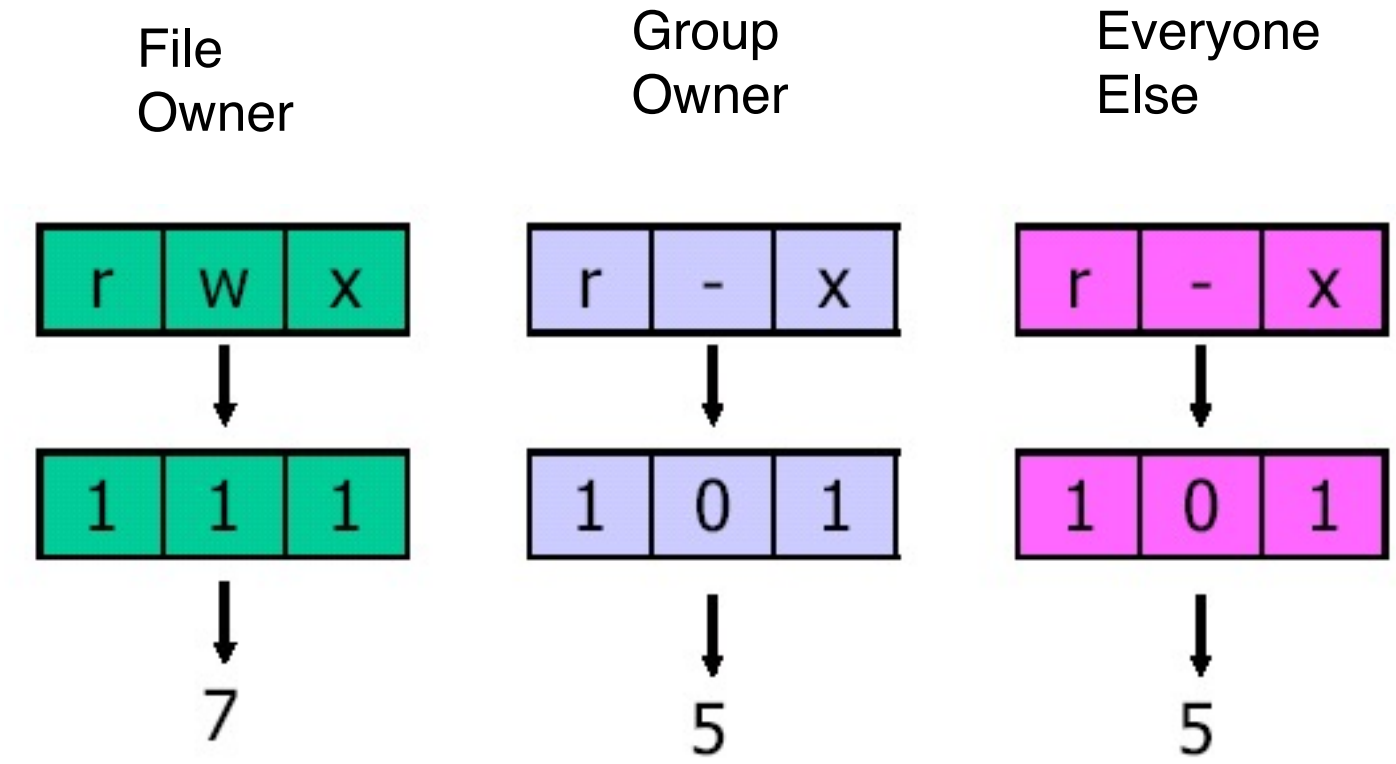
File Attributes

```
drwxr-xr-x 2 root   root   4096 Sep 24  2017 Unit2
drwxr-xr-x 2 root   root   4096 Feb 26 19:21 a
-rwxr-xr-x 1 root   root  10930 Mar  5 22:49 a.out
-rwxrwx--- 1 root   root     81 Aug  2  2017 a.txt
-rwxr-x--- 1 root   root     81 Jan 26 19:20 b.txt
```

File Attributes



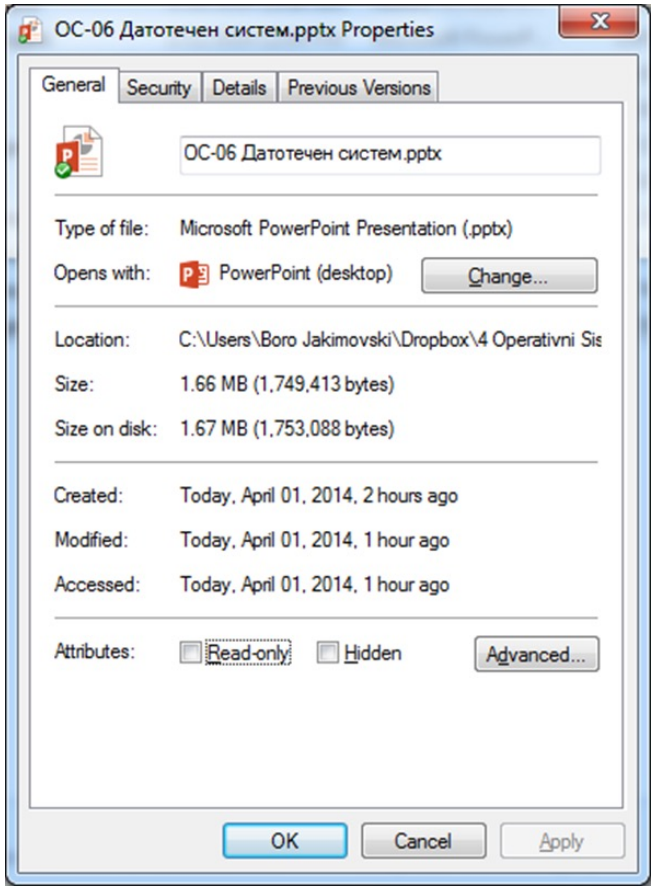
File Attributes



stat in Linux

```
$stat /etc/passwd
  File: `/etc/passwd'
  Size: 119417          Blocks: 248        IO Block: 4096   regular file
Device: 803h/2051d      Inode: 2696882     Links: 1
Access: (0644/-rw-r--r--)  Uid: (    0/    root)   Gid: (    0/    root)
Access: 2014-04-01 11:04:17.000000000 +0200
Modify: 2014-03-30 09:52:09.000000000 +0200
Change: 2014-03-30 09:52:09.000000000 +0200
```

File Properties in Windows



File Operations

- Create
- Delete
- Open
  - Read all the attributes and list of disk addresses in the main memory in order to speed up the access
- Close
- Read
- Write
- Append
  - Restricted write form

File Operations

- Seek
  - For random access files
- Get Attributes
  - Used in `make` in UNIX
- Set Attributes
  - Access protection in files, `chmod`
- Rename

System Calls Example …

```
#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 */

        /* 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 */
        . . .
```

Simple program for file copying

… System Calls Example

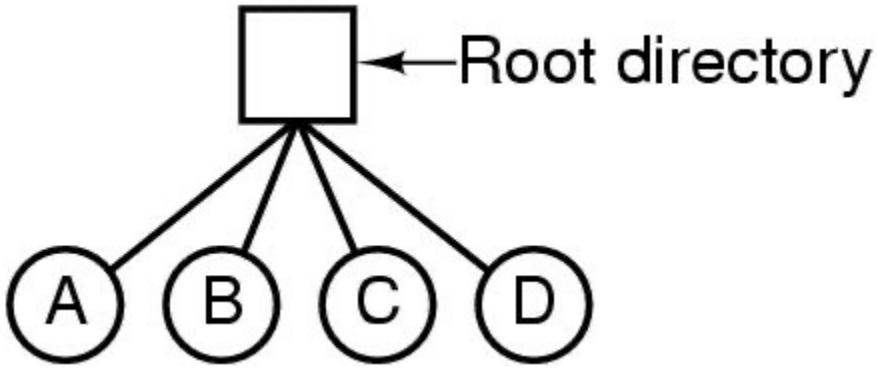
```
    /* Copy loop */
    while (TRUE) {
        rd_count = read(in_fd, buffer, BUF_SIZE); /* read a block of data */
        if (rd_count <= 0) break;                 /* if end of file or error, exit loop */
        wt_count = write(out_fd, buffer, rd_count); /* write data */
        if (wt_count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

Simple program for file copying

Directories

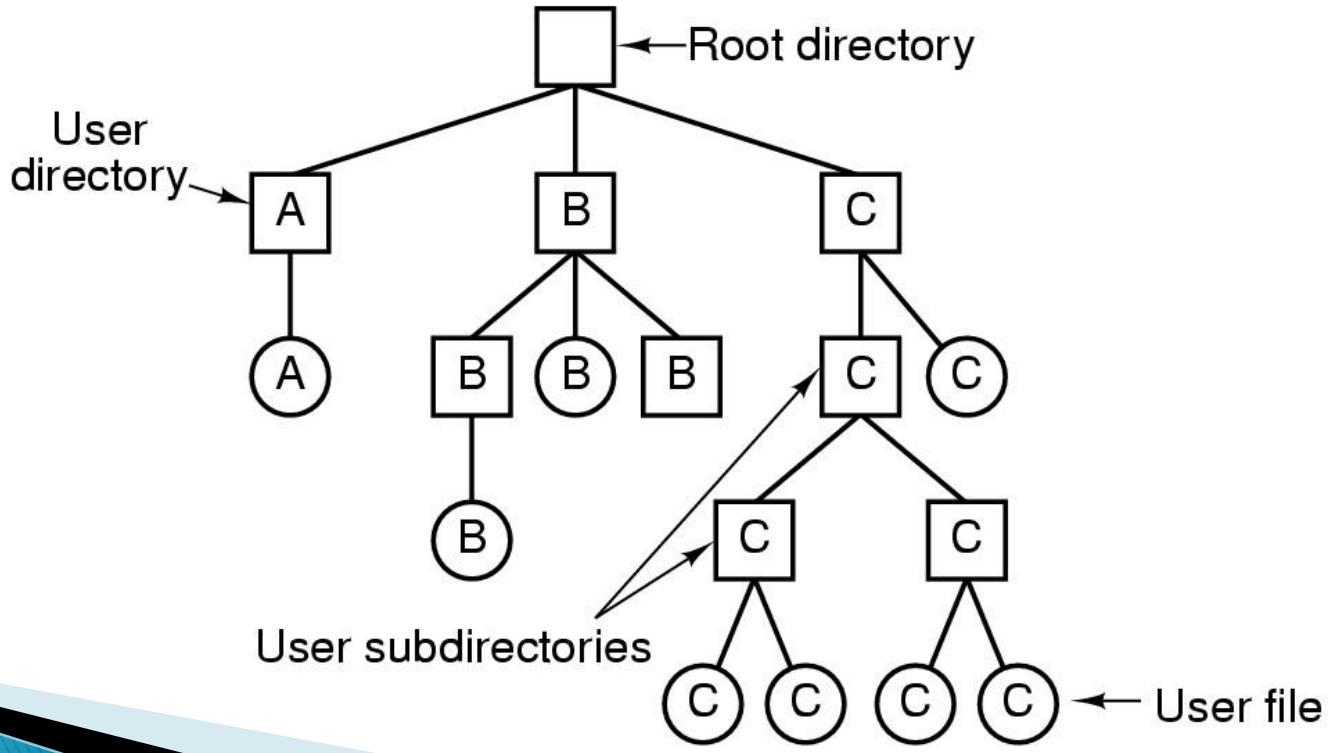
Single-Level Directory Systems

- Single-level structure
- Advantage:
  - Simpler software design
  - Fast and easy locating the files
- Disadvantage:
  - Forbidden use of files with same names

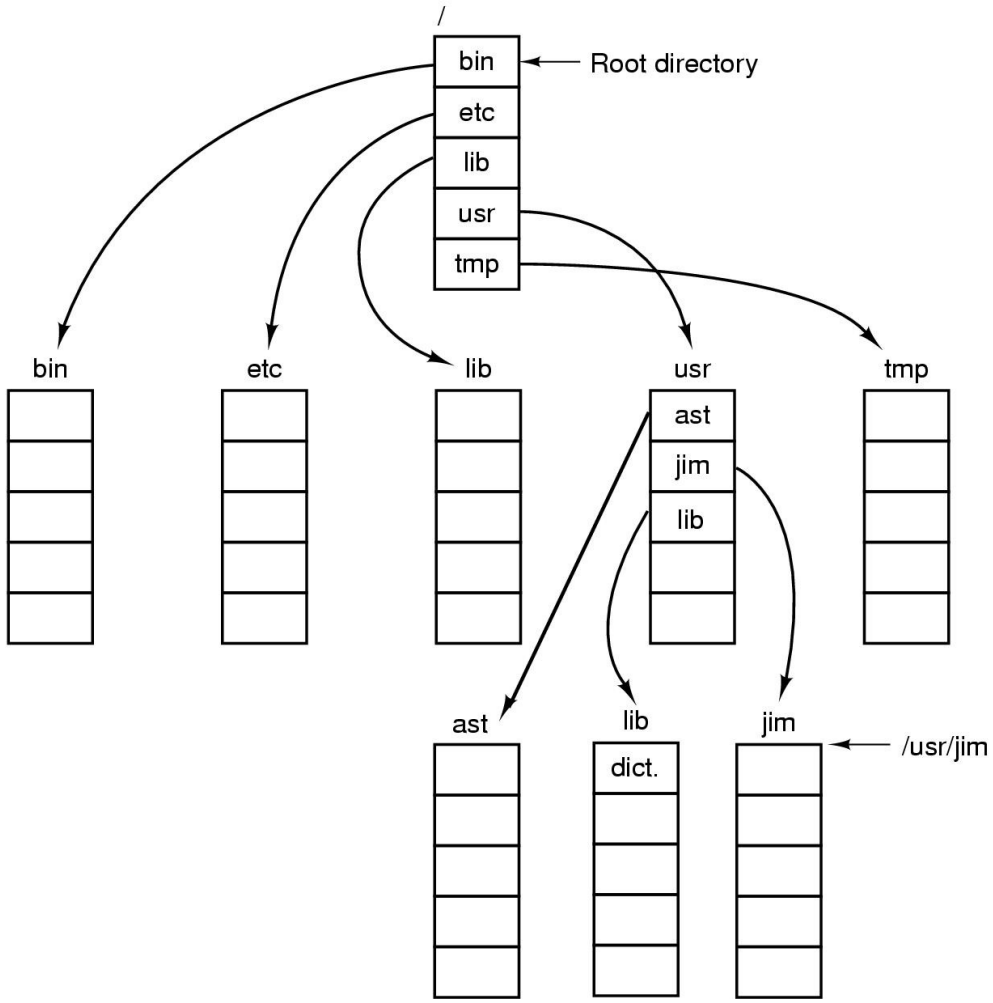


Hierarchical Directory Systems

- File grouping in a natural way
- A tree of directories
- Each user can have several directories



Path Names



Unix Directory Tree

Path Names

- Absolute path
  - Starts from the root and it is unique
    - /home/user
- Relative path
  - Concept of a current directory
  - All paths are relative to the current directory
  - If the working directory is /usr/ast:
    - Instead of:
      - `cp /usr/ast/mailbox /usr/ast/mailbox.bak`
    - We can write:
      - `cp mailbox mailbox.bak`

File Names

- Special names:
  - “.” and “..”
  - Current and parent directory:
    - `cp /usr/lib/dictionary .`
    - `cp /usr/lib/dictionary ../dictionary`
    - `cp /usr/lib/dictionary /usr/ast/dictionary`

Directory System Calls

- Create
- Delete
- OpenDir
- CloseDir
- ReadDir
- Rename
- Link
- Unlink

Working with Directories

- Hard link
  - One file can be in several directories
  - There is a counter in the file attributes that keeps the information about how many directories have the file
- Symbolic link
  - A new file is created with a link (path) to the real file's location

File-System Implementation

File System Implementation: Two Aspects

- Users:
  - How to name files, which operations are supported, how the directory tree looks
- Implementators
  - How to keep the files and the directories, managing with the disk space, providing efficiency and reliability

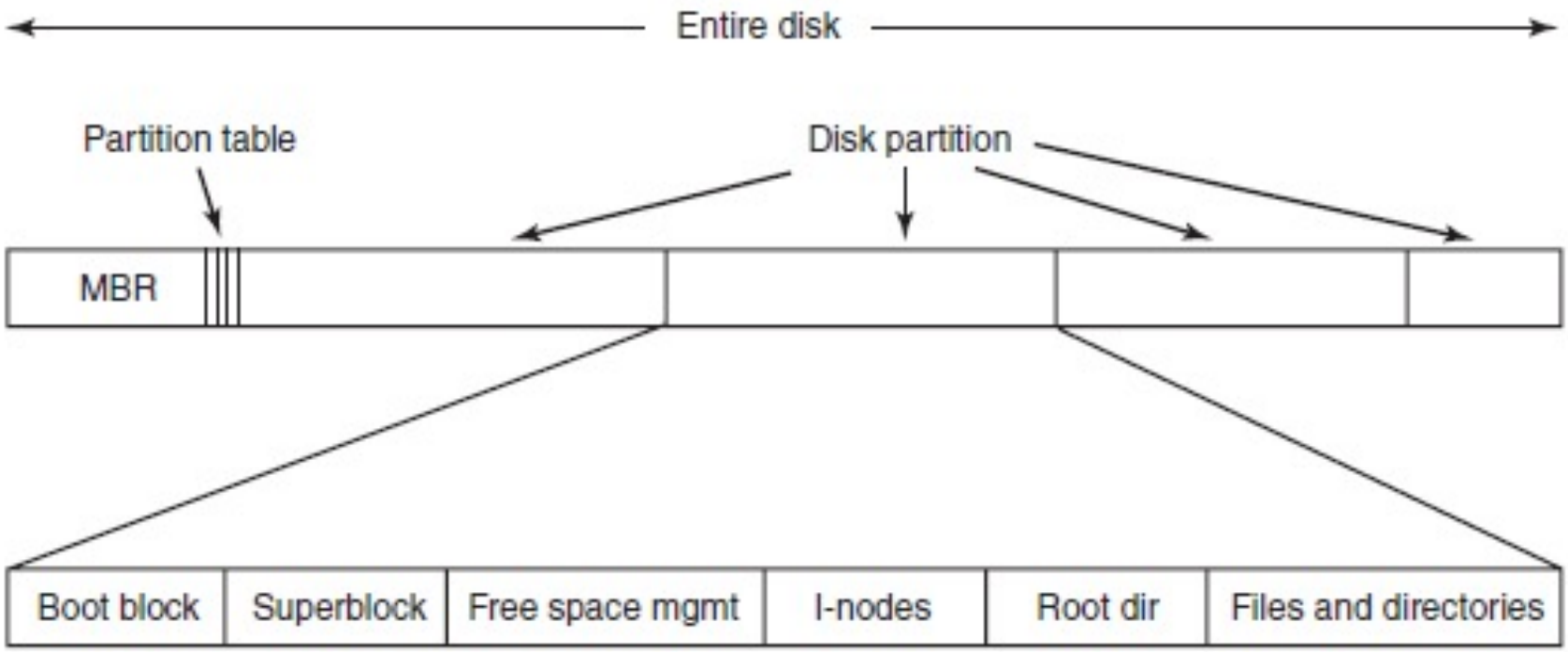
Disk Space Organization

- Disks are divided in several partitions
- Sector 0 is called **MBR** (**master boot record**), and it is used to boot the computer
- At the end of this sector there is the **partition table** containing the start and the end address of each partition

Disk Space Organization

- The **BIOS** reads and executes the MBR, MBR finds the active partition
- Each partition has a **boot block** – used to load the OS
- After that we have:
  - Super block – file system parameters (magic number–FS type, number of blocks, etc.)
  - Free block information (bitmap or a list)
  - List of i-nodes, root directory and the rest of the files and directories

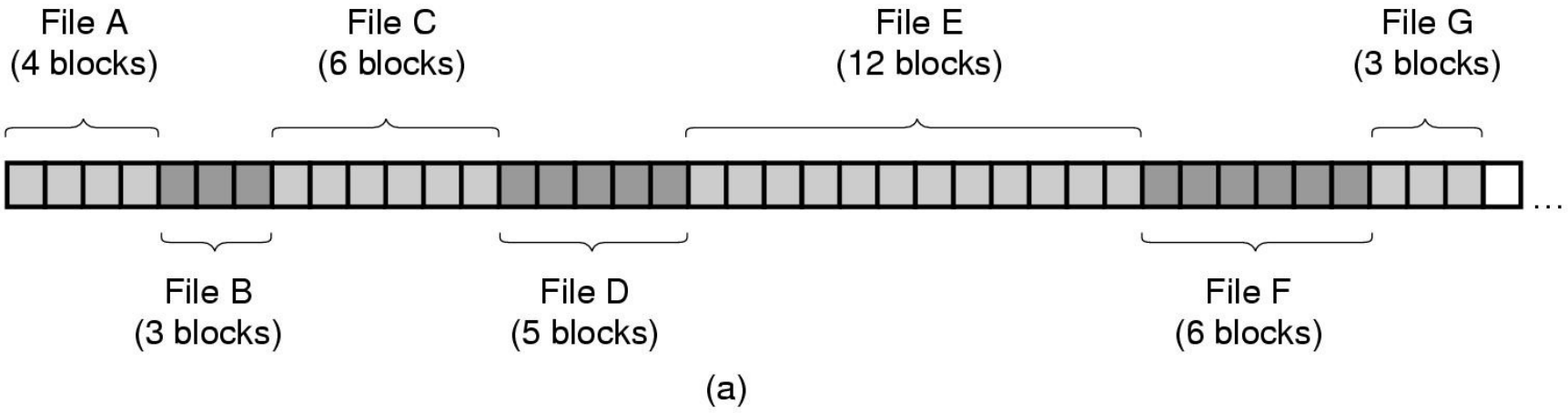
Disk Space Organization



File Implementation

- Files consist of several blocks
- How to allocate the blocks for a given file?
  - Sequential allocation
  - Allocation using a linked list
  - Allocation using a linked list and a table in memory
  - I-nodes

Sequential Allocation



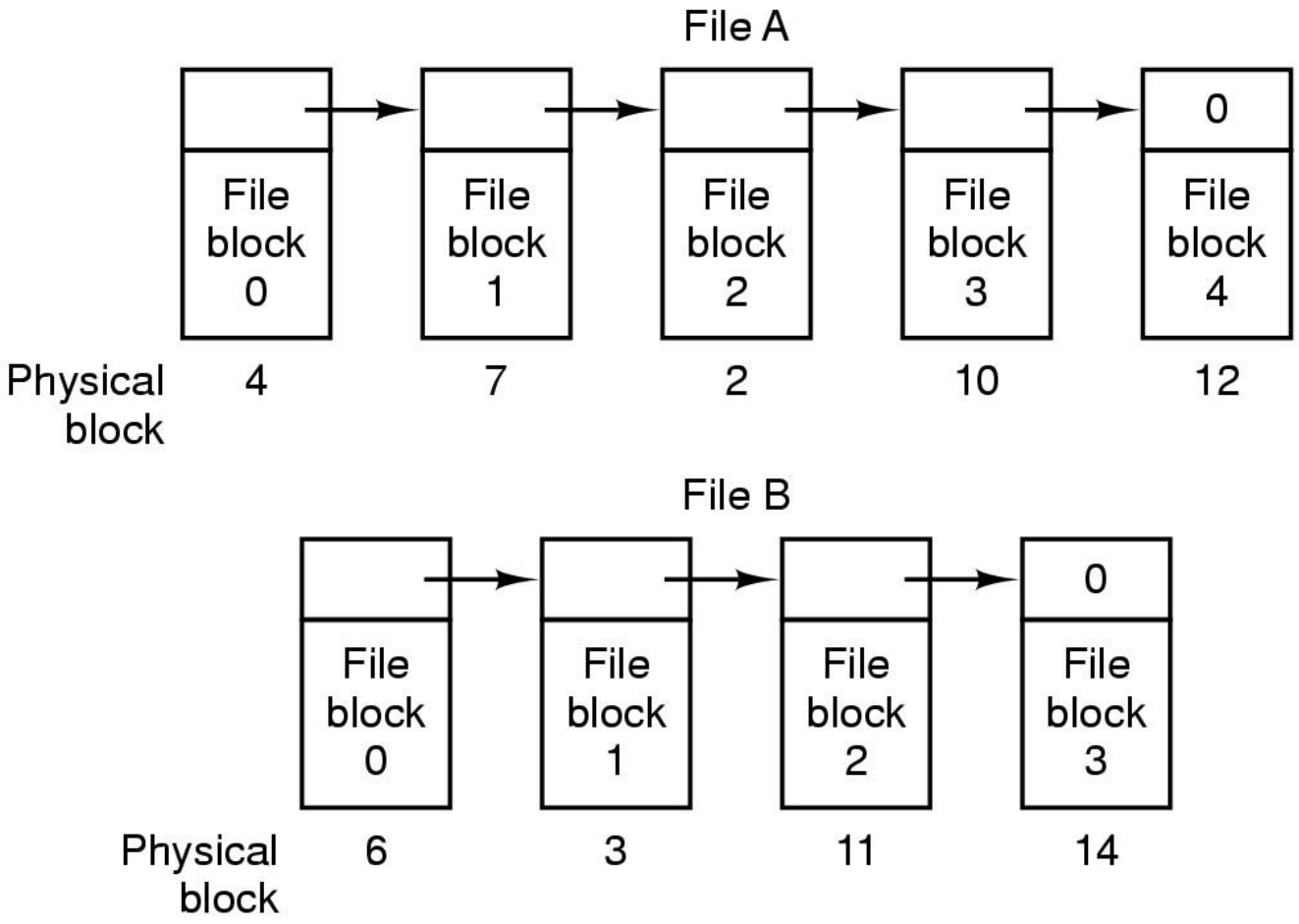
(a) Sequential allocation for 7 files

(b) Disk state after the removal of files D and F.

Sequential Allocation

- Advantages:
  - Easy to implement (address of the first block and the number of blocks)
  - Reading the file in one operation
- Disadvantages:
  - Disk fragmentation
  - The size of the files must be known in advance
- Implementation:
  - CD-ROM: file sizes are known in advance
  - DVD: several sequential *extents*, due to the use of UDF (Universal Disk Format)

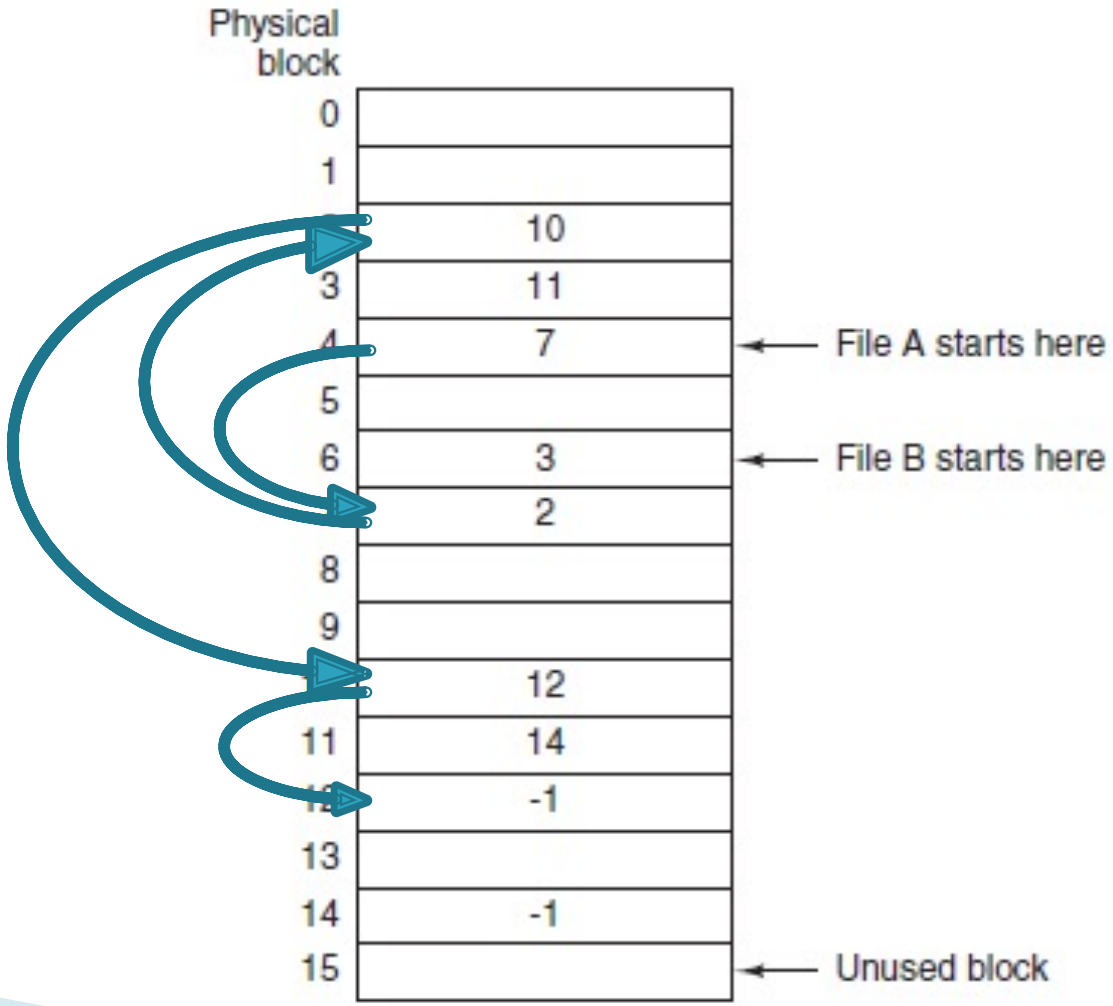
Linked List Allocation



Linked List Allocation

- Each block points to the next file block
- Advantages
  - No disk fragmentation
  - Only internal fragmentation, at the last block
  - It is enough to know the address of the first block
- Disadvantages
  - Slow seek: you need to read n-1 blocks to get to the n^{th} block
  - Part from the block is used as a pointer

Linked List Allocation with a Table in the Memory



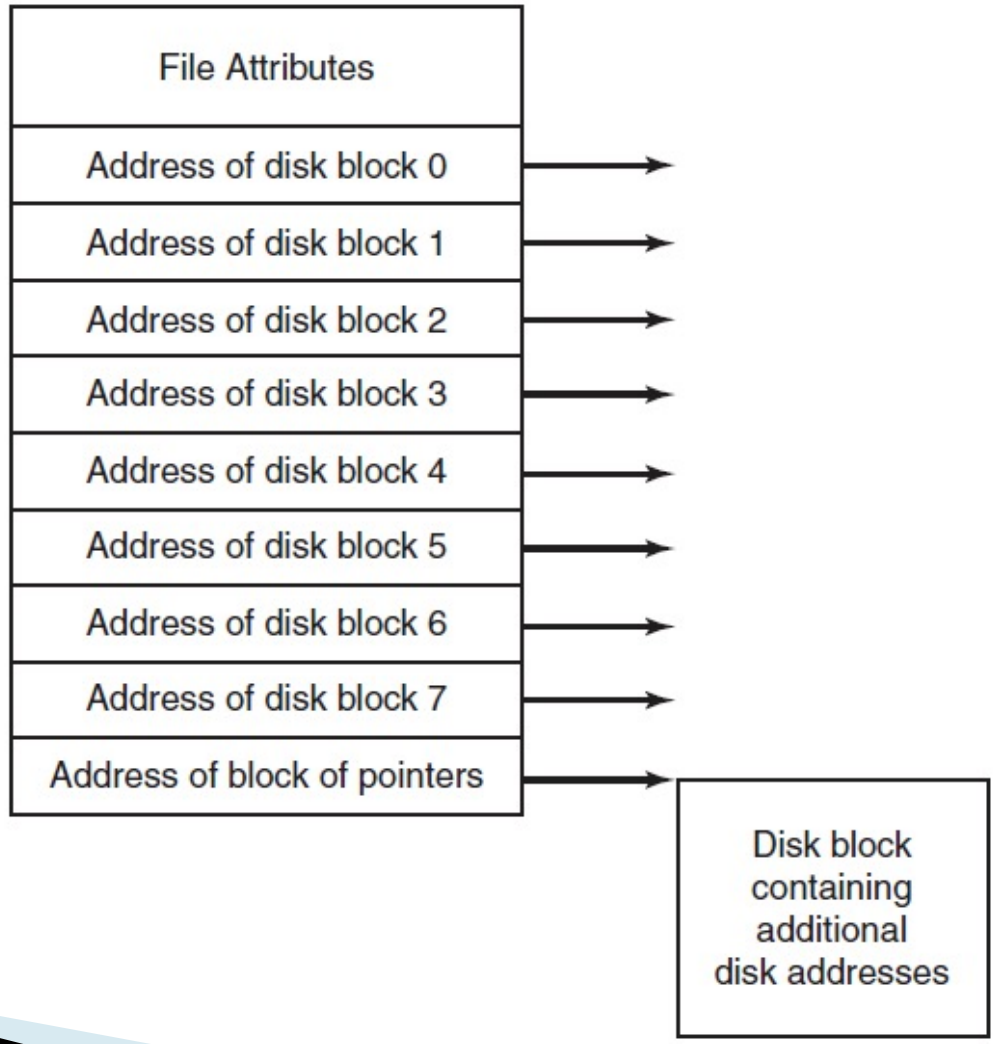
Linked List Allocation with a Table in the Memory

- The block pointers are kept in a table in memory: File Allocation Table (FAT)
- Advantages
  - The entire block is used for data
  - Fast random access
    - The 'chain' is in memory
- Disadvantages
  - The entire table must be in memory all the time
  - Poor scalability
    - Example: 1TB disk, 1KB block size --> 1 billion entries, each 3-4 bytes --> 2.4-3 GB of main memory

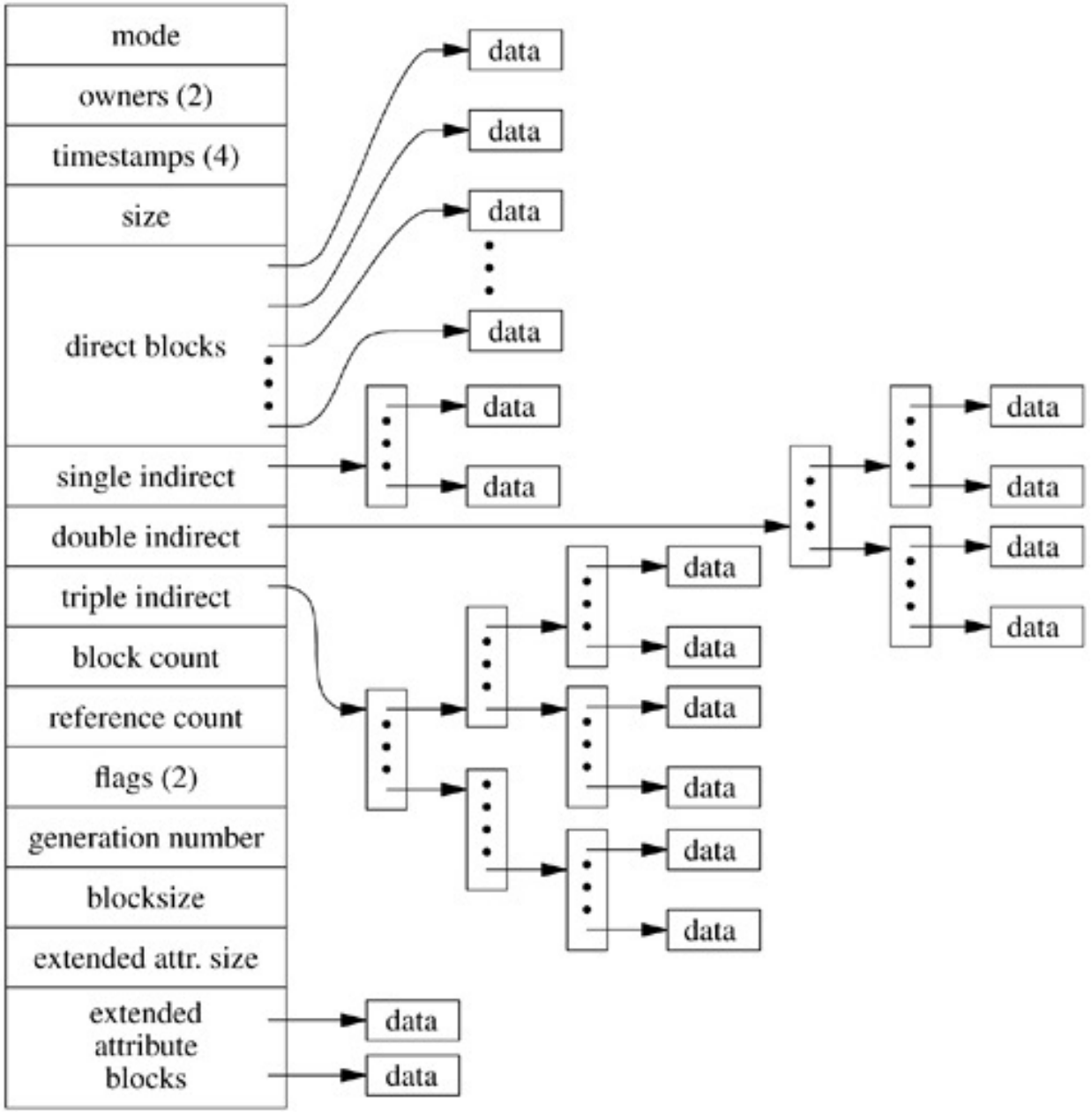
I-nodes

- Each file is associated with a data structure called **the index node** (i-node)
- The node lists the attributes and the disk addresses of the file's blocks
- Advantages
  - The i-node needs to be in memory only when the file is open
- Disadvantage
  - Fixed size

I-nodes


File Attributes
Address of disk block 0
Address of disk block 1
Address of disk block 2
Address of disk block 3
Address of disk block 4
Address of disk block 5
Address of disk block 6
Address of disk block 7
Address of block of pointers
Disk block containing additional disk addresses

I-nodes


mode
owners (2)
timestamps (4)
size
direct blocks
single indirect
double indirect
triple indirect
block count
reference count
flags (2)
generation number
blocksize
extended attr. size
extended attribute blocks
data

Questions?