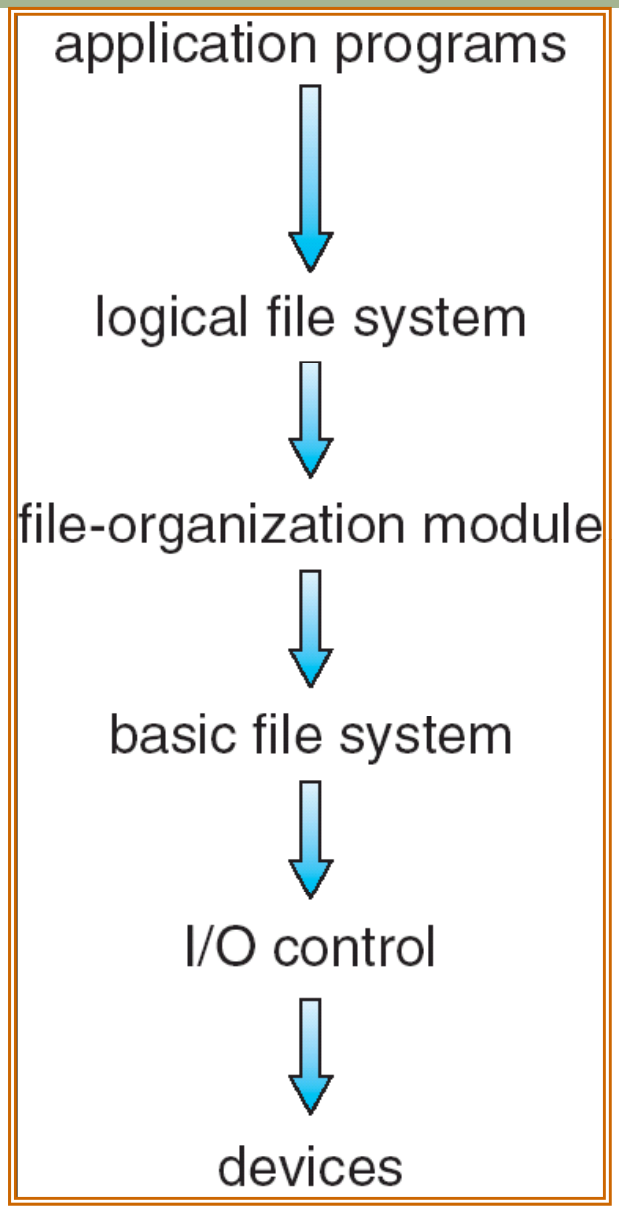
by I.-C. Lin, Dept. CS, NCTU. Textbook: Operating System Principles 7ed

# CHAPTER 11: IMPLEMENTING FILE SYSTEMS (CONCISE)

# File-System Structure

- File structure
  - Logical storage unit
  - Collection of related information
- File system resides on secondary storage (disks)
- File system organized into layers
- **File control block** – storage structure consisting of information about a file

# Layered File System



# Layered File System (cont.)

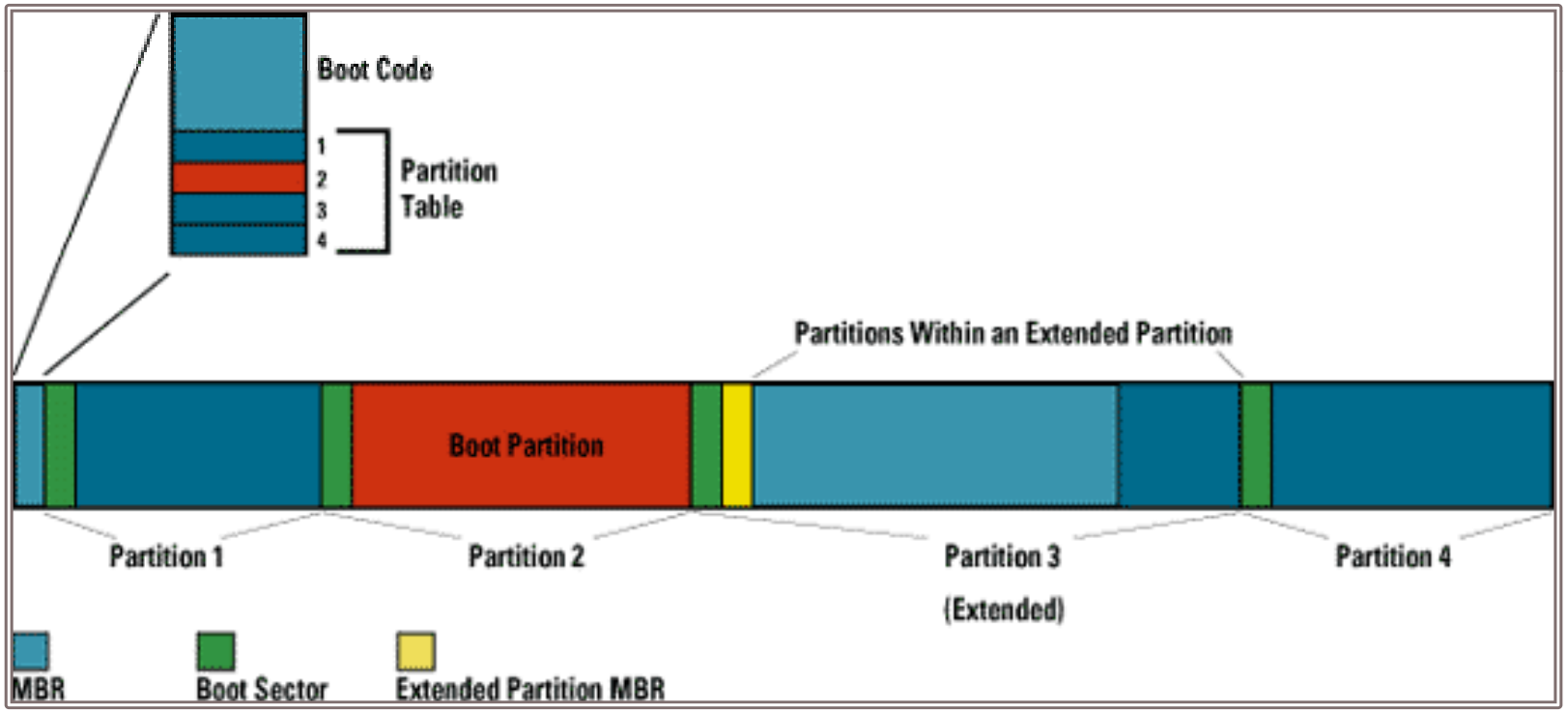
- I/O control
  - Device drivers and interrupt handlers.
  - Translate high-level commands to hardware-specific instructions.
- Basic file system
  - Issue generic commands to device drivers.
  - with physical blocks identified by its address (e.g. drive 1, cylinder 73, track2, sector 10)
- File-organization module
  - Logical blocks -> physical blocks
  - Free-space management
- Logical file system
  - Manage metadata. (all except the contents)
  - File-control block (FCB)

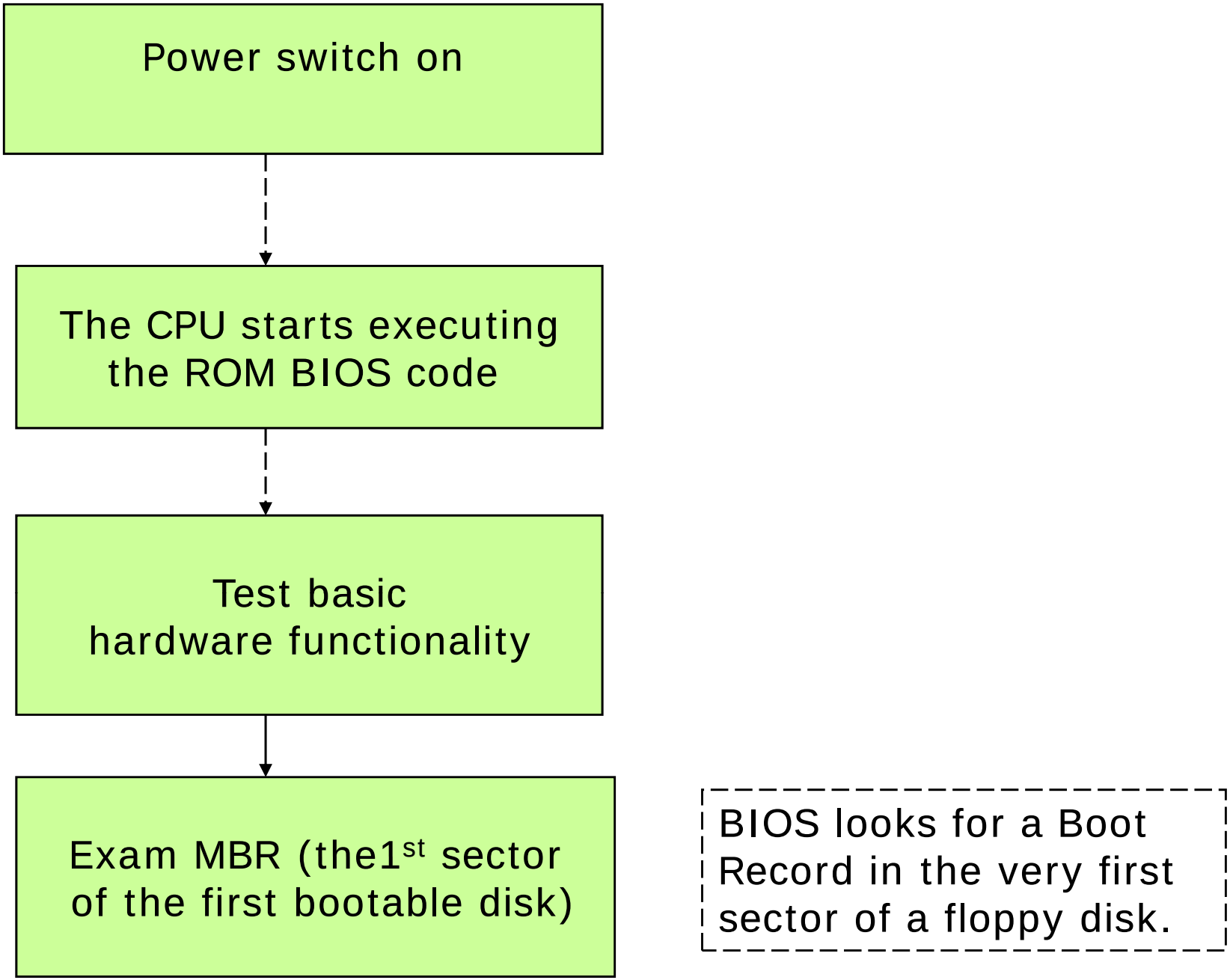
# Layered File System (cont.)

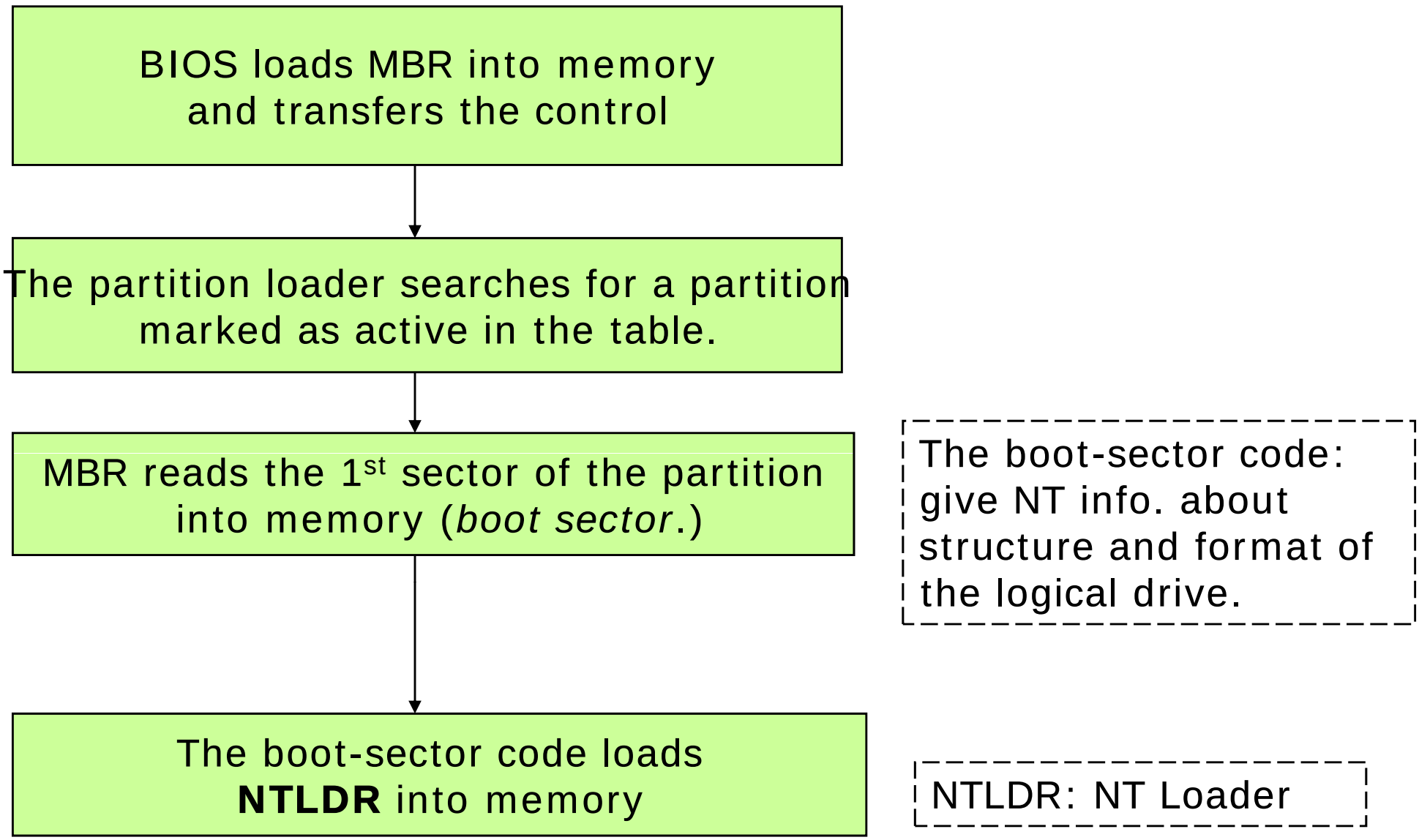
- Layered structure
  - Flexibility
  - Minimize duplicate codes to support multiple file systems.
  - E.g.
    - CD-ROM: ISO 9660
    - Unix: UNIX file system (UFS)
    - Windows: FAT, FAT32, NTFS (Windows NT File System)
    - Linux: Extended file system (ext2, ext3…)

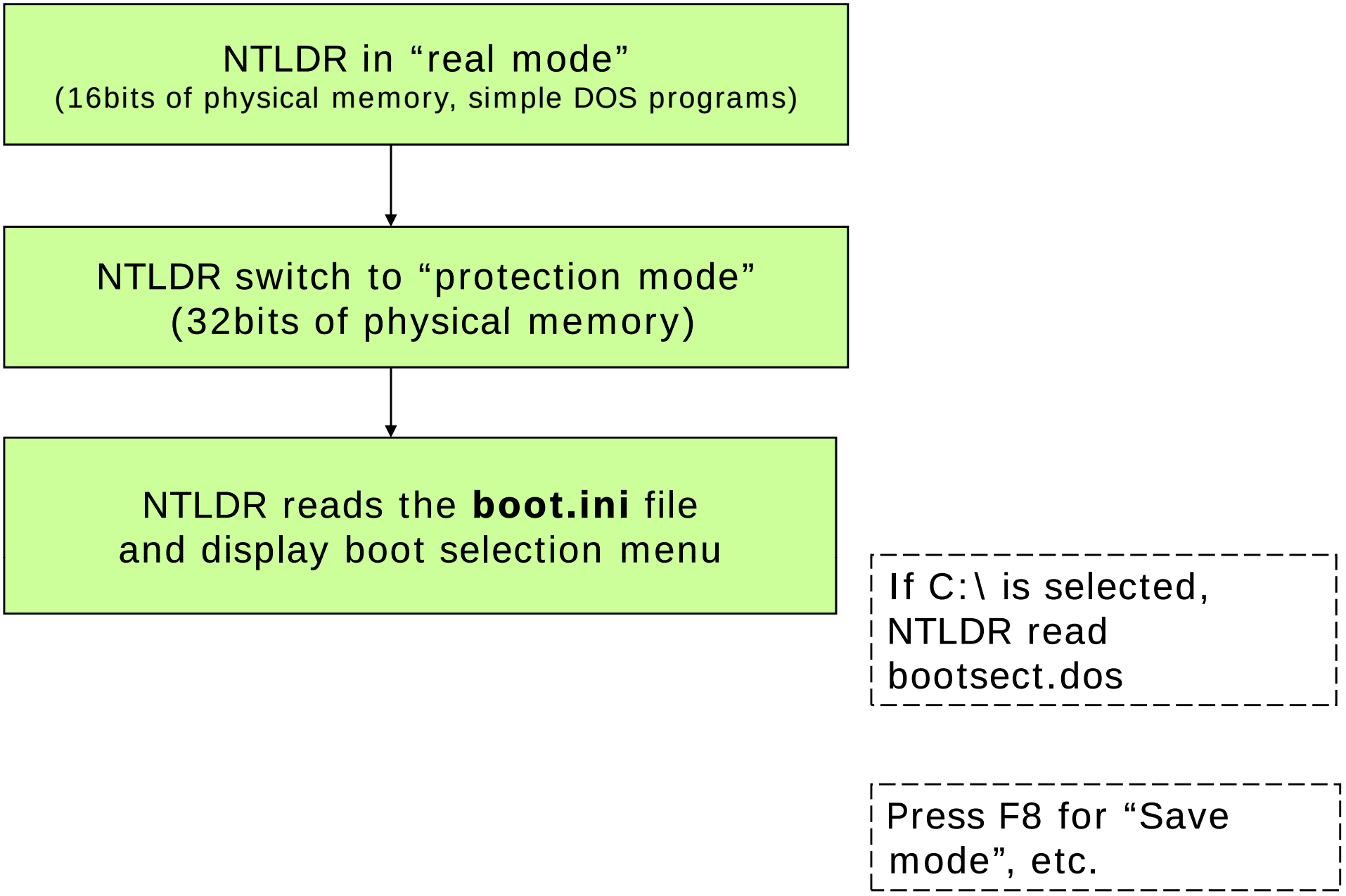
# On-disk Structure

- On-disk structure
- Boot control block
  - UFS: boot block
  - NTFS: partition boot sector




Power switch on
The CPU starts executing the ROM BIOS code
Test basic hardware functionality
Exam MBR (the1st sector of the first bootable disk)
BIOS looks for a Boot Record in the very first sector of a floppy disk.


BIOS loads MBR into memory
and transfers the control
The partition loader searches for a partition
marked as active in the table.
MBR reads the 1st sector of the partition
into memory (*boot sector*.)
The boot-sector code loads
**NTLDR** into memory
The boot-sector code:
give NT info. about
structure and format of
the logical drive.
NTLDR: NT Loader


NTLDR in “real mode”
(16bits of physical memory, simple DOS programs)
NTLDR switch to “protection mode”
(32bits of physical memory)
NTLDR reads the **boot.ini** file
and display boot selection menu
If C:\ is selected,
NTLDR read
bootsect.dos
Press F8 for “Save
mode”, etc.

# E.g. BOOT.INI

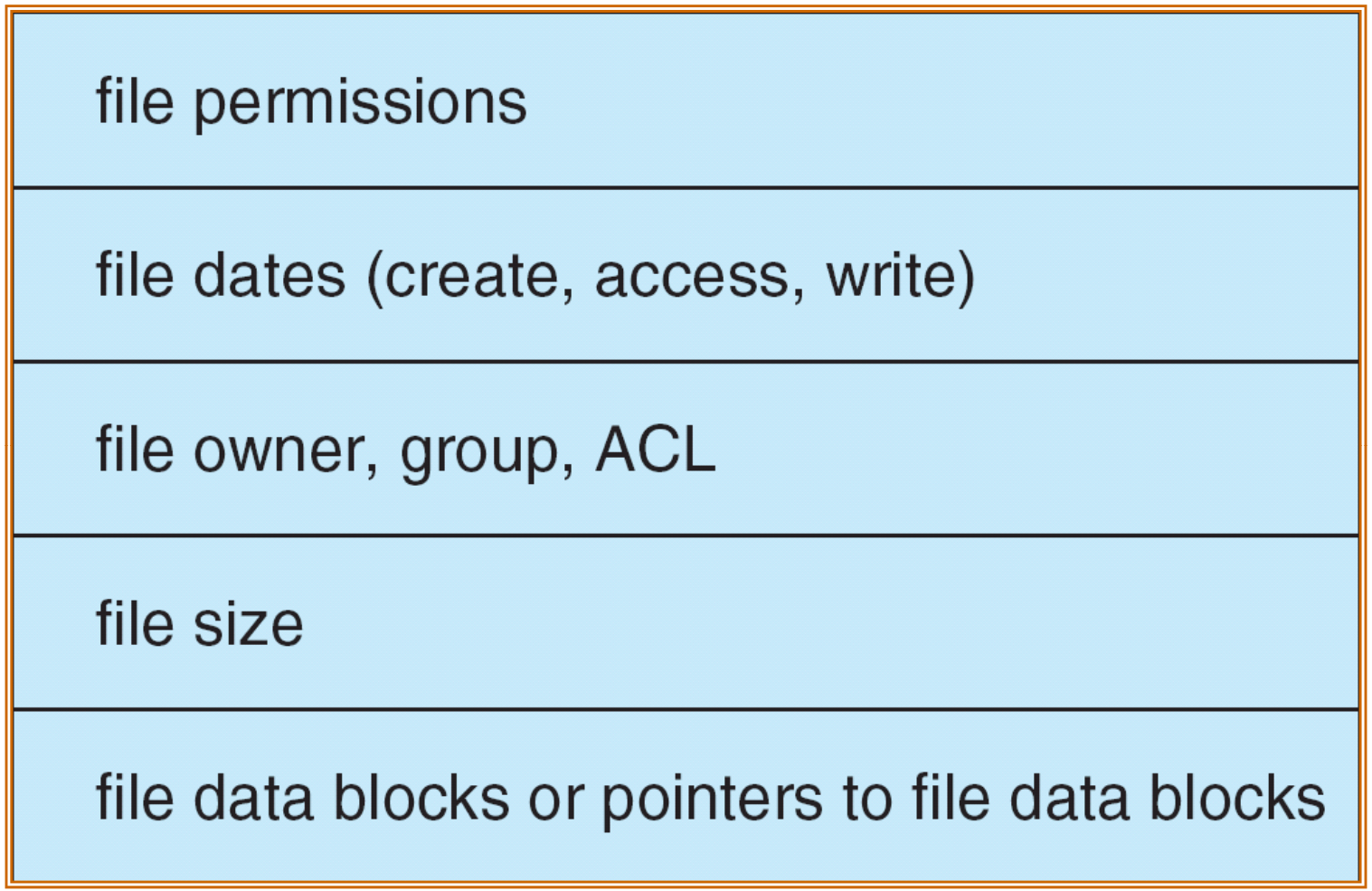
```
[boot loader]
timeout=30
default=multi(0)disk(0)rdisk(0)partition(1)\WINDOWS

[operating systems]
multi(0)disk(0)rdisk(0)partition(1)\WINDOWS="Microsoft
    Windows XP Home Edition" /fastdetect /NoExecute=OptIn
multi(0)disk(0)rdisk(0)partition(2)\WINNT="Windows 2000
    Professional" /fastdetect
C:\="MS Windows"
```

# On-disk Structure (cont.)

- Volume control block
  - About volume (“logical drive”) details
  - E.g. number of blocks, free block count…
  - UFS: superblock
  - NTFS: in master file table
- Directory structure
  - NTFS: in the master file table
- Per-file FCB
  - UFS: inode
  - NTFS: in the master file table

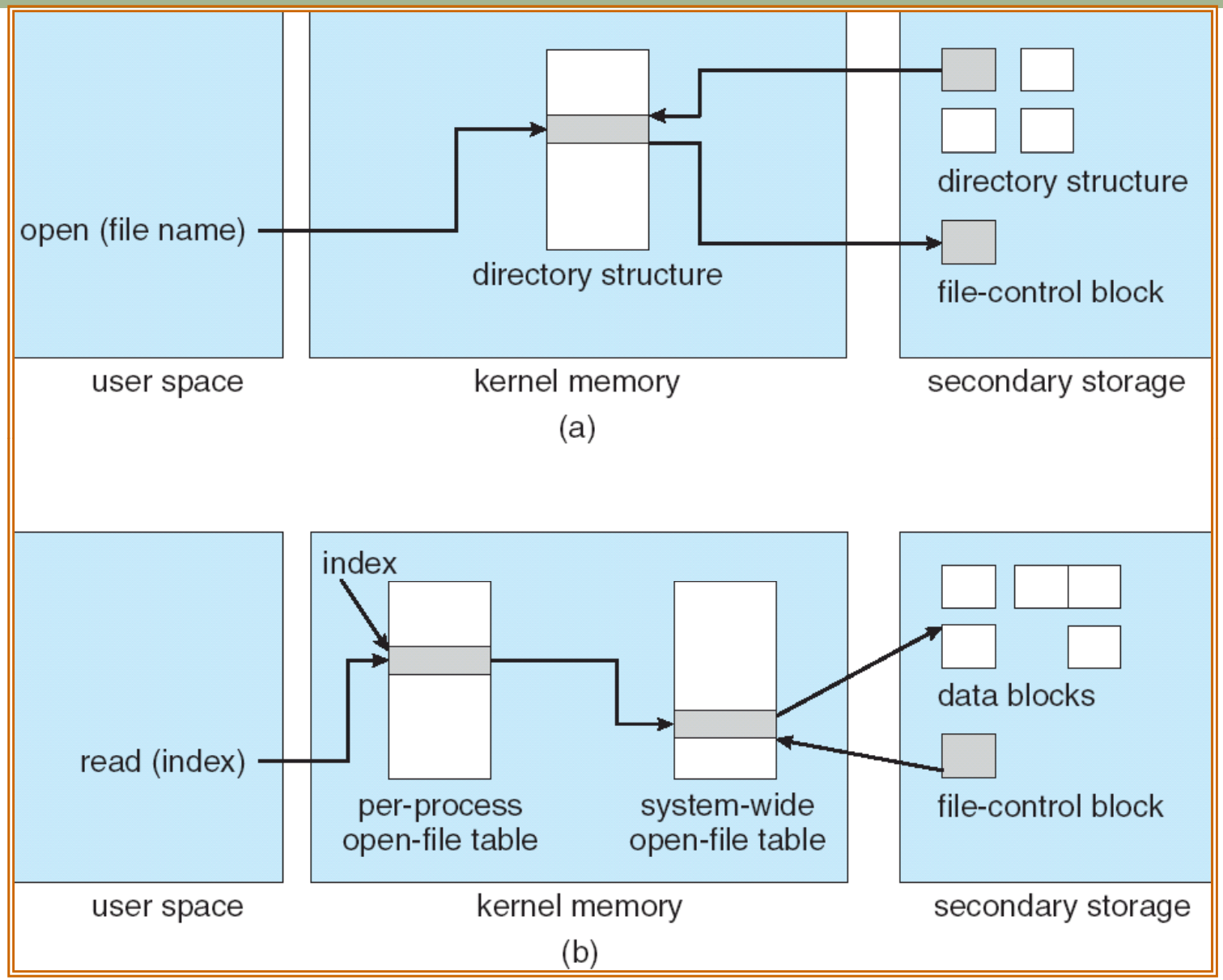
# A Typical File Control Block



# In-Memory File System Structures

- In-memory mounted table
- In-memory directory structure
- System-wide open-file table
  - Copy of the FCB of each open file, etc.
- Per-process open-file table
  - Per-process info. and a pointer to the system-wide open-file table

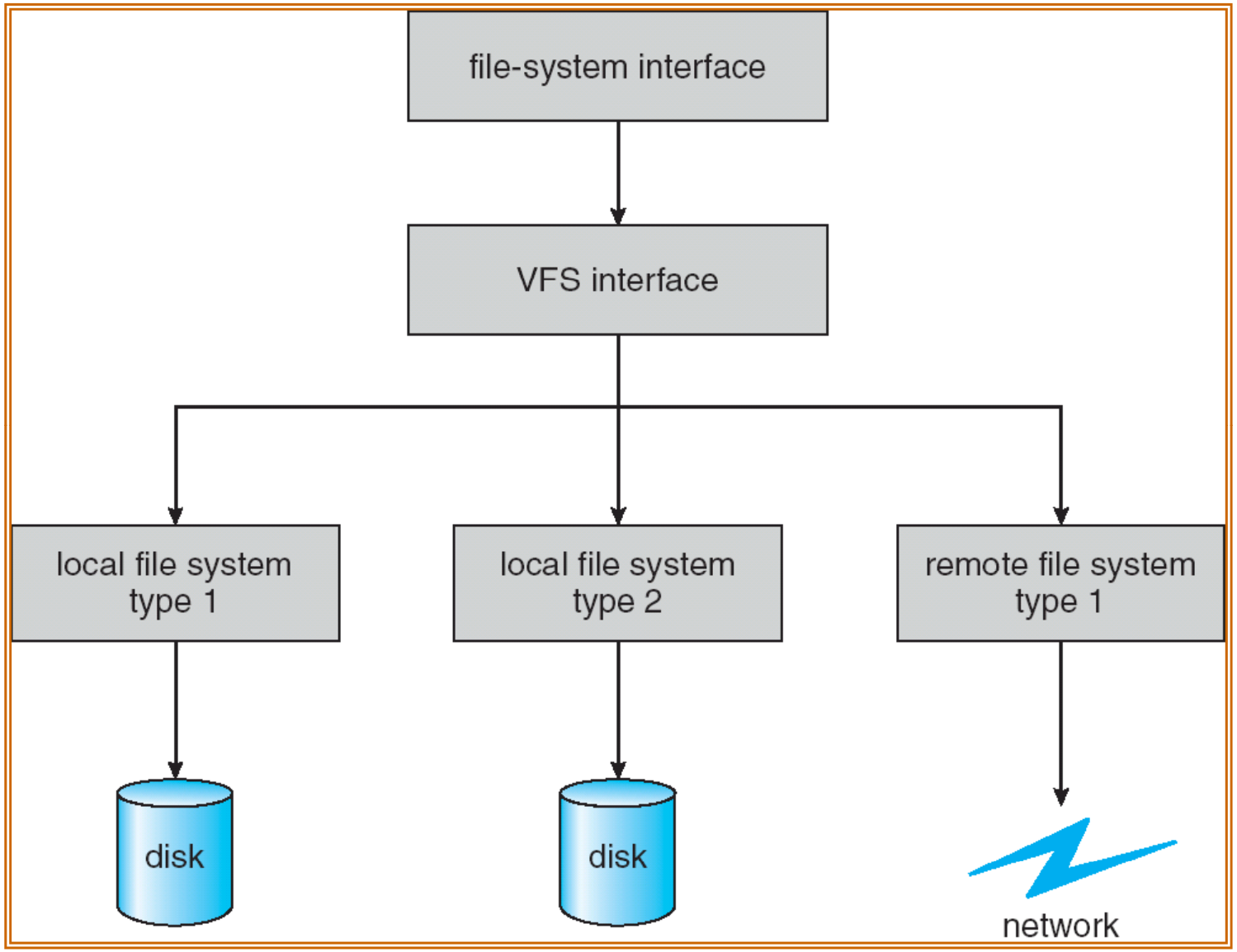
# In-Memory File System Structures



# Virtual File Systems

- Virtual File Systems (VFS) provide an object-oriented way of implementing file systems.
- VFS Concurrently supports multiple types of 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



# Directory Implementation

- **Linear list** of file names with pointer to the data blocks.
  - simple to program
  - time-consuming to execute

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

# Allocation Methods

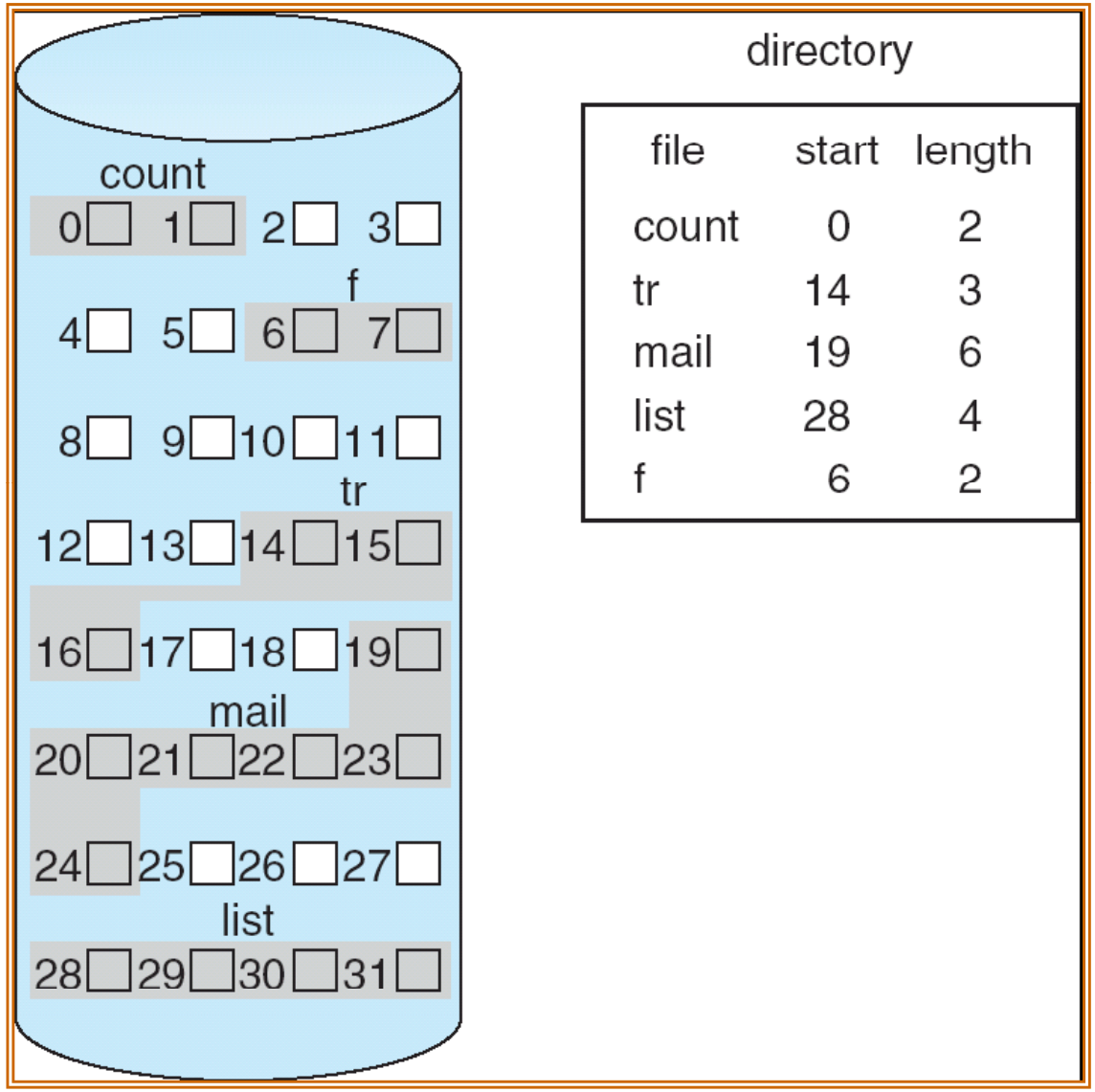
- An allocation method refers to how disk blocks are allocated for files:
- **Contiguous allocation**
- **Linked allocation**
- **Indexed allocation**

# Contiguous Allocation

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

How to deal with these problems?

# Contiguous Allocation of Disk Space

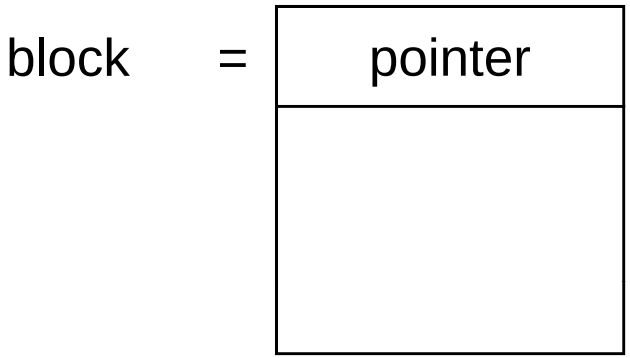


# Extent-Based Systems

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

# Linked Allocation

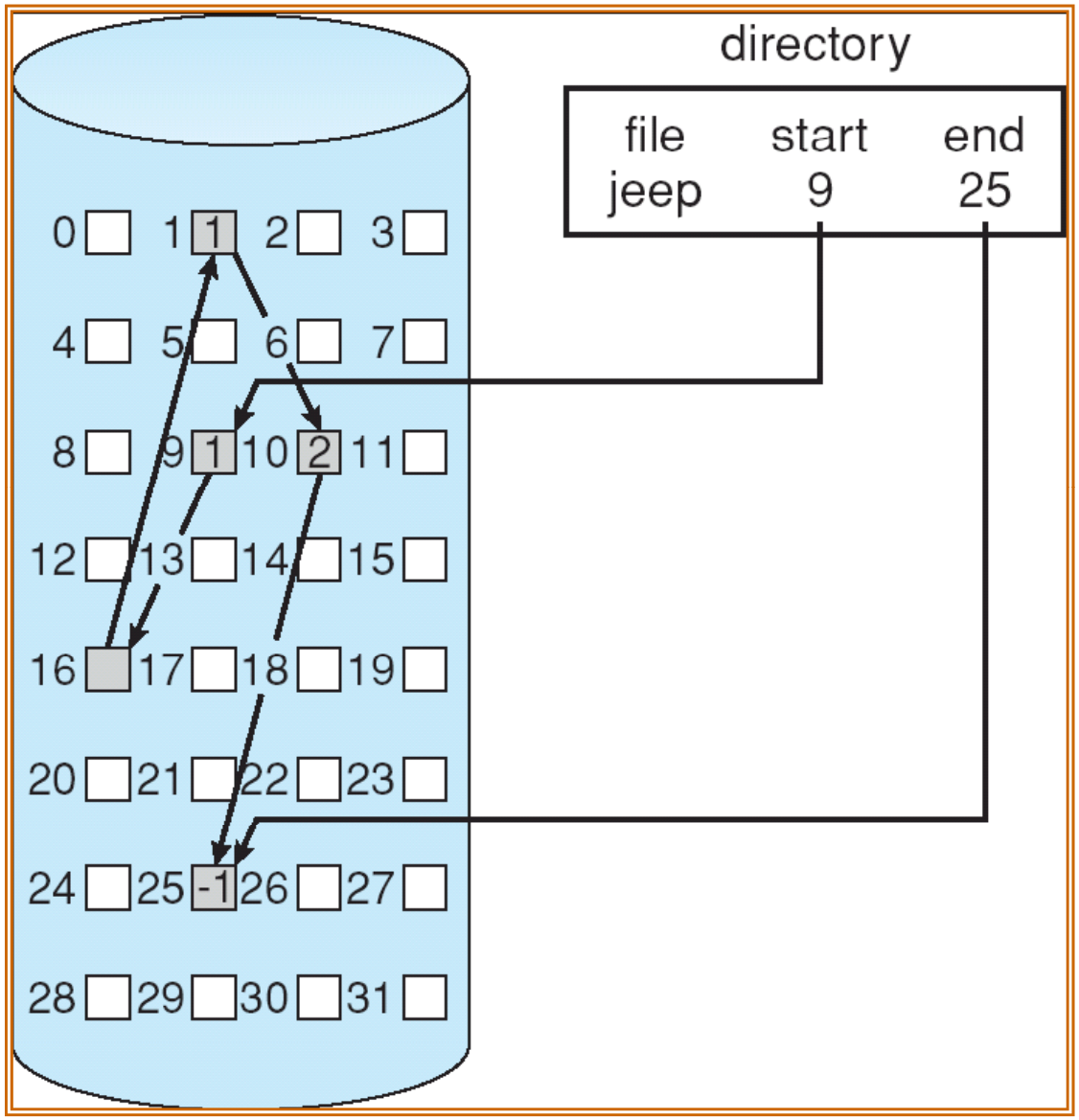
- Each file is a linked list of disk blocks: blocks may be scattered anywhere on the disk.



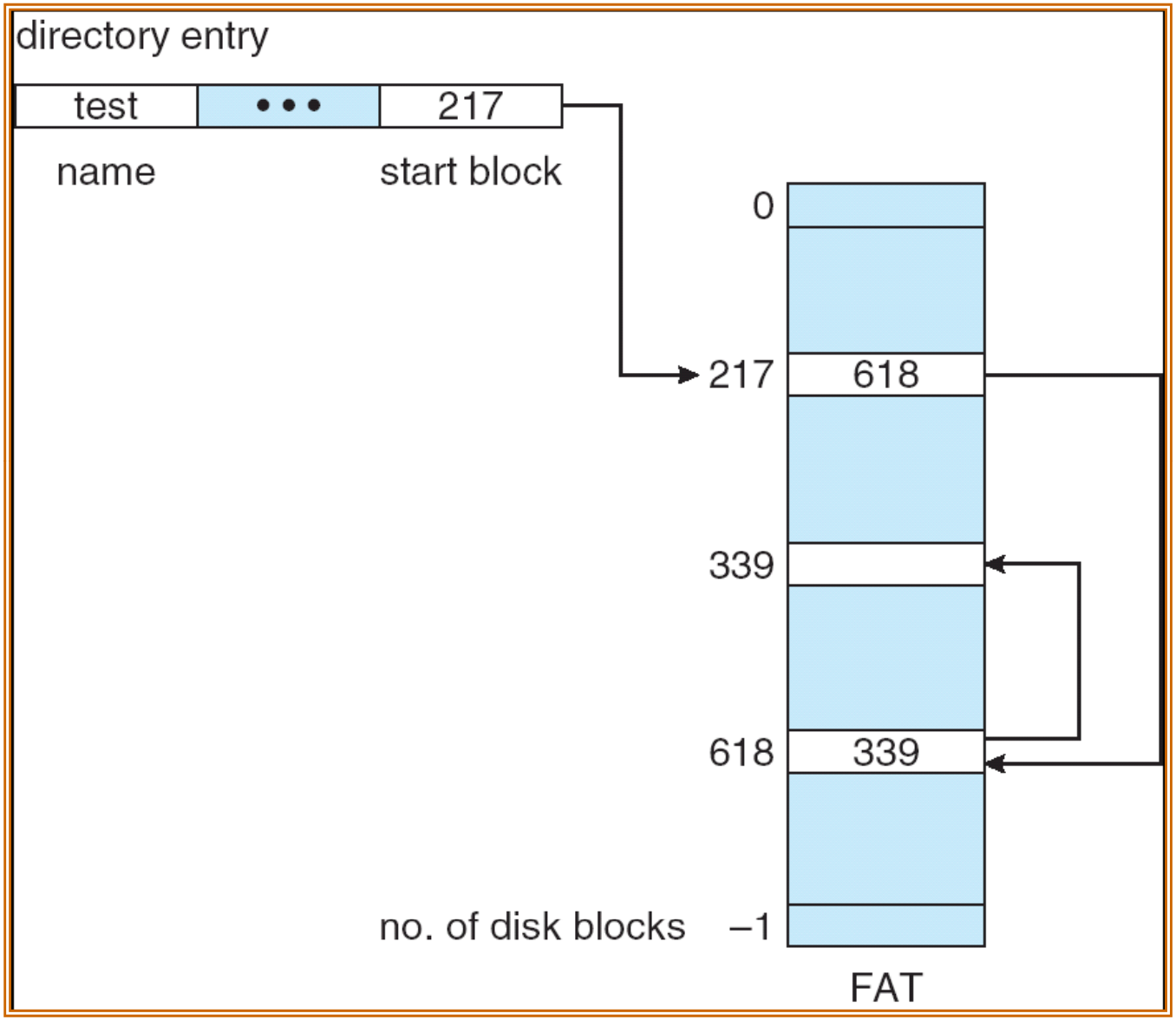
# Linked Allocation (Cont.)

- Simple – need only starting address
- Free-space management system
  - no waste of space
- No random access
- File-allocation table (FAT) – disk-space allocation used by MS-DOS and OS/2.

# Linked Allocation

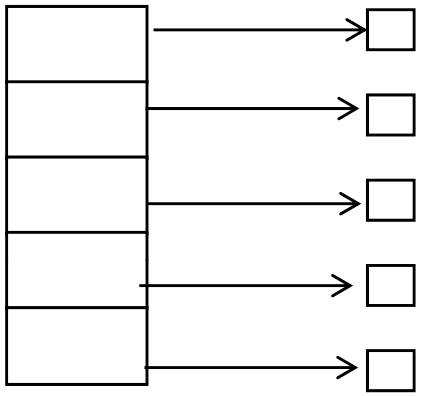


# File-Allocation Table



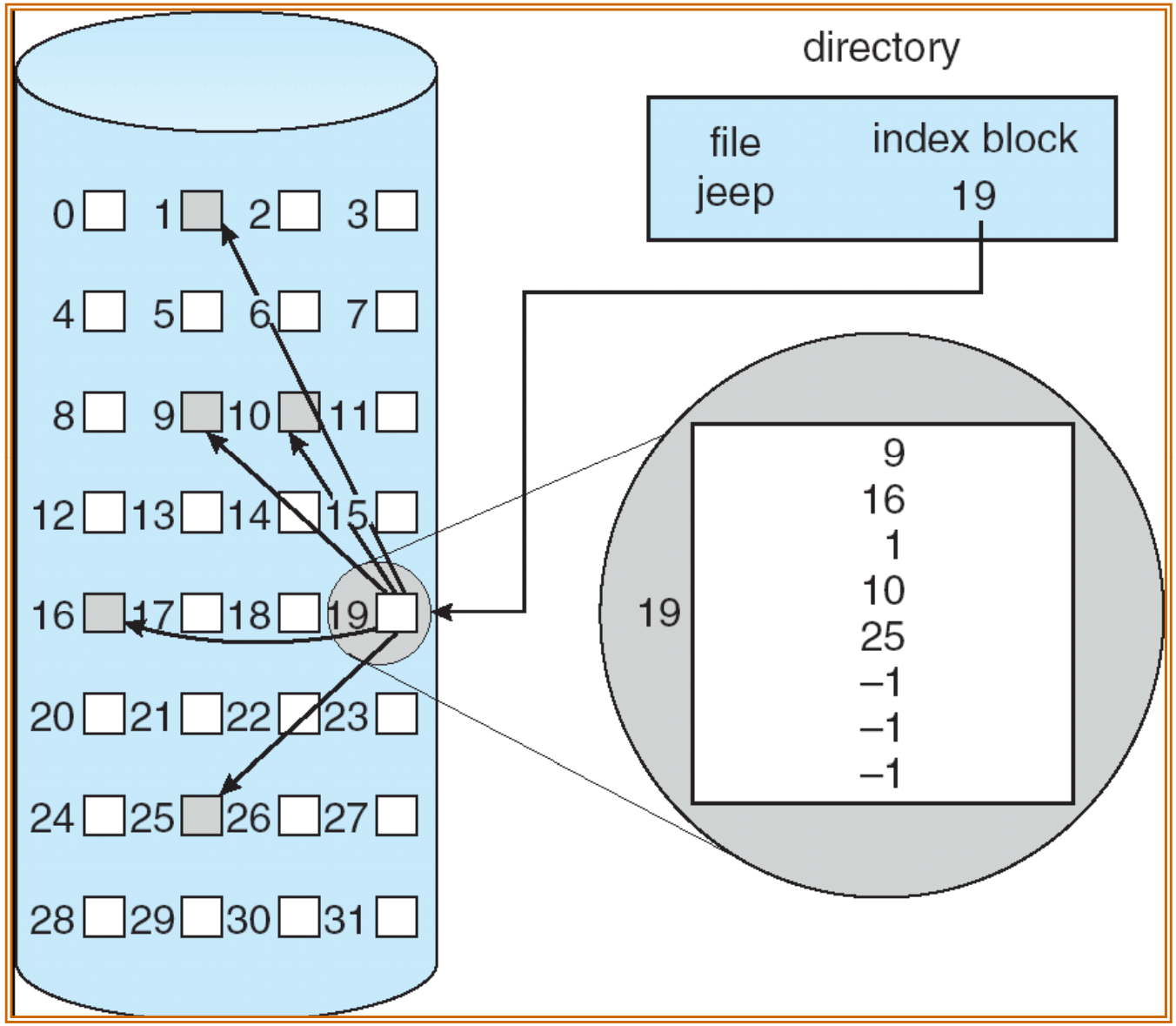
# Indexed Allocation

- Brings all pointers together into the *index block.*
- Logical view.



index table

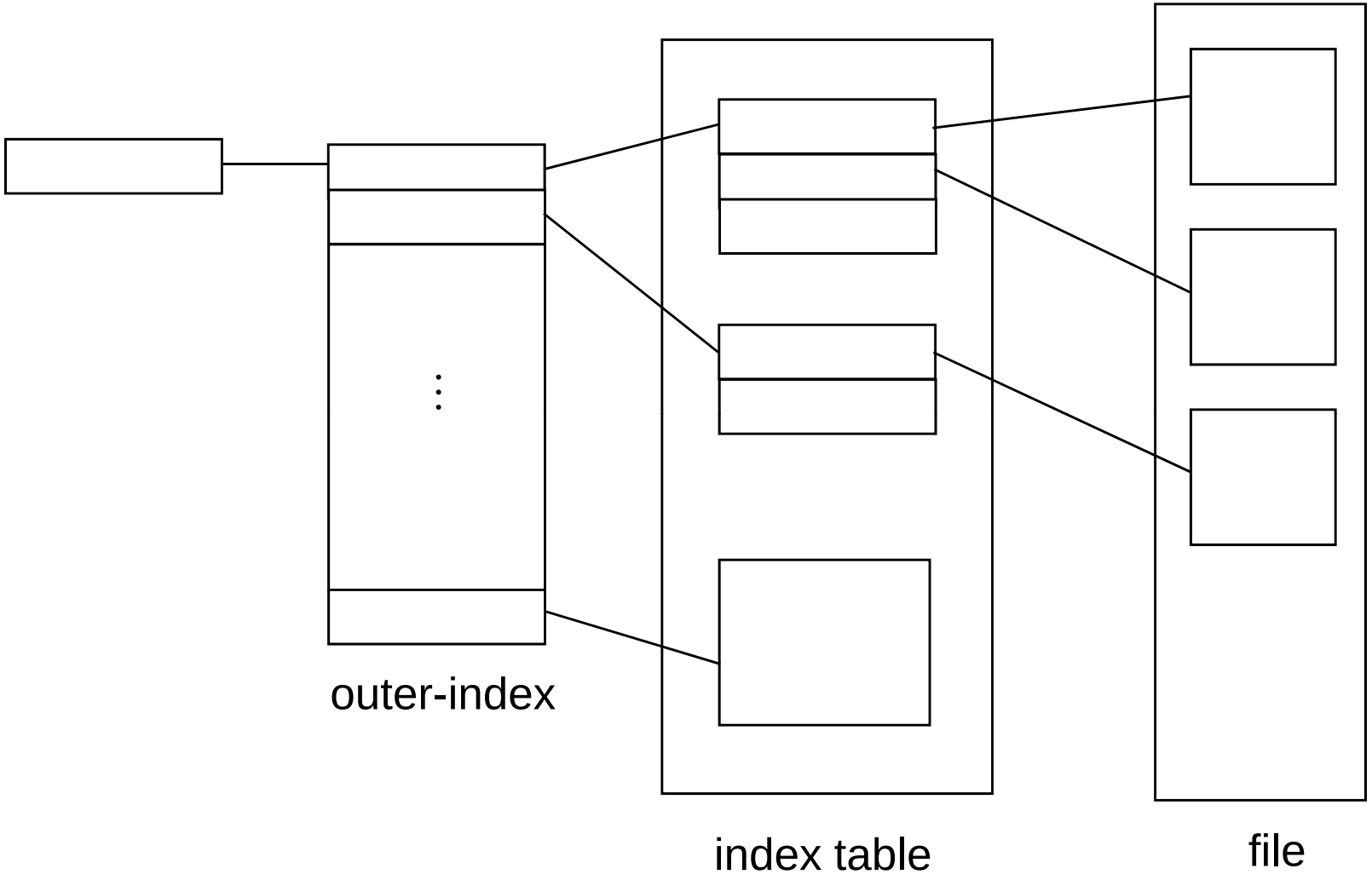
# Example of Indexed Allocation



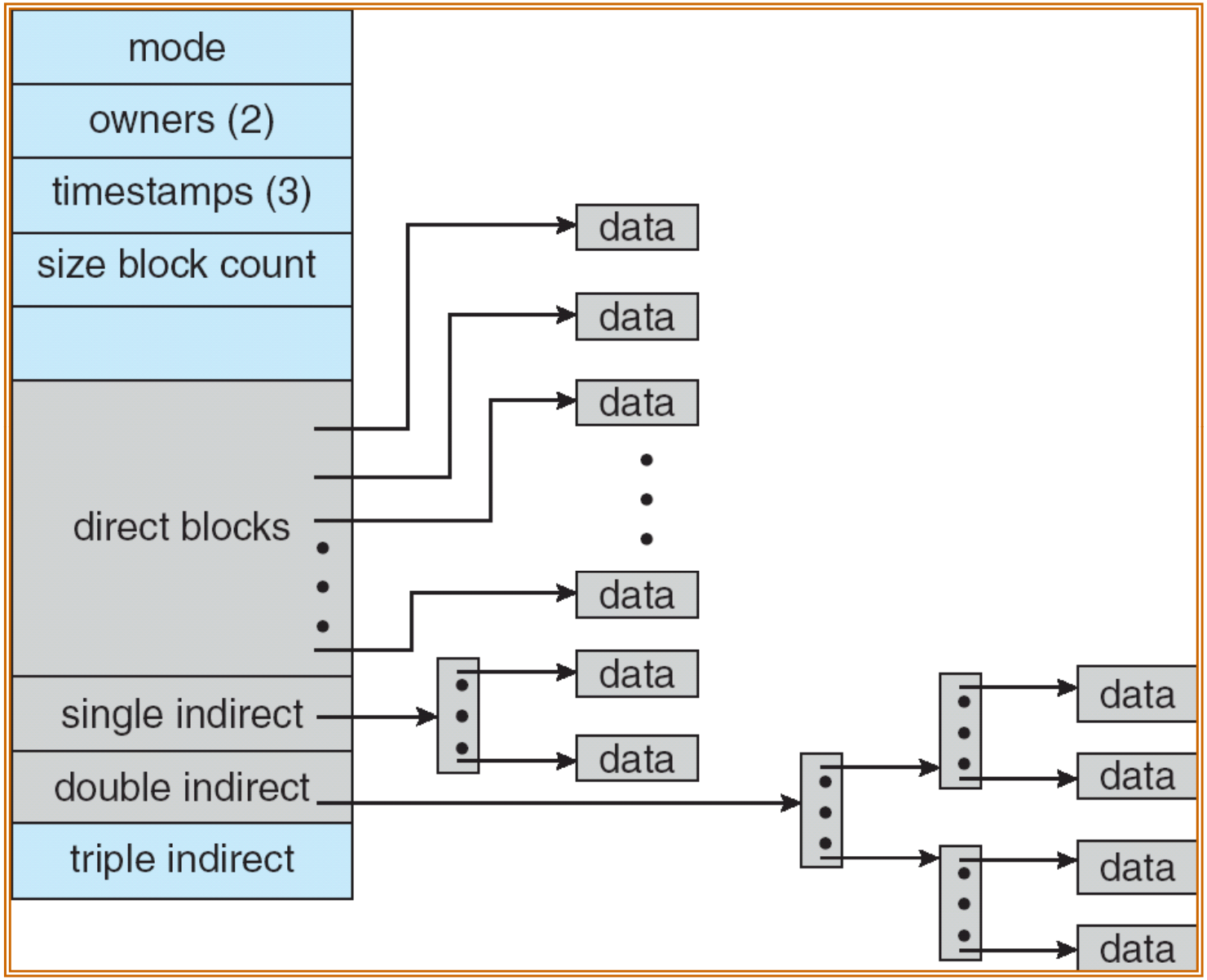
# Indexed Allocation (Cont.)

- Need index table
- Random access
- Dynamic access without external fragmentation, but have overhead of index block.
- How large should the index block be?
  - Linked scheme – Link blocks of index table (no limit on size).
  - Multilevel index

# Indexed Allocation – Mapping (Cont.)

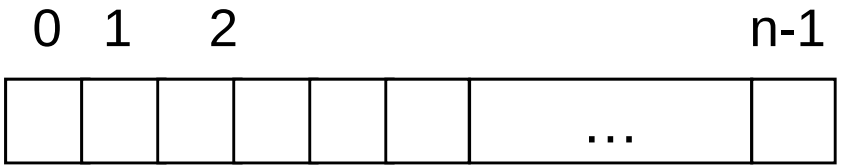


# Combined Scheme: UNIX (4K bytes per block)



# Free-Space Management

- Bit vector (*n* blocks)




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

Block number calculation

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

# Free-Space Management (Cont.)

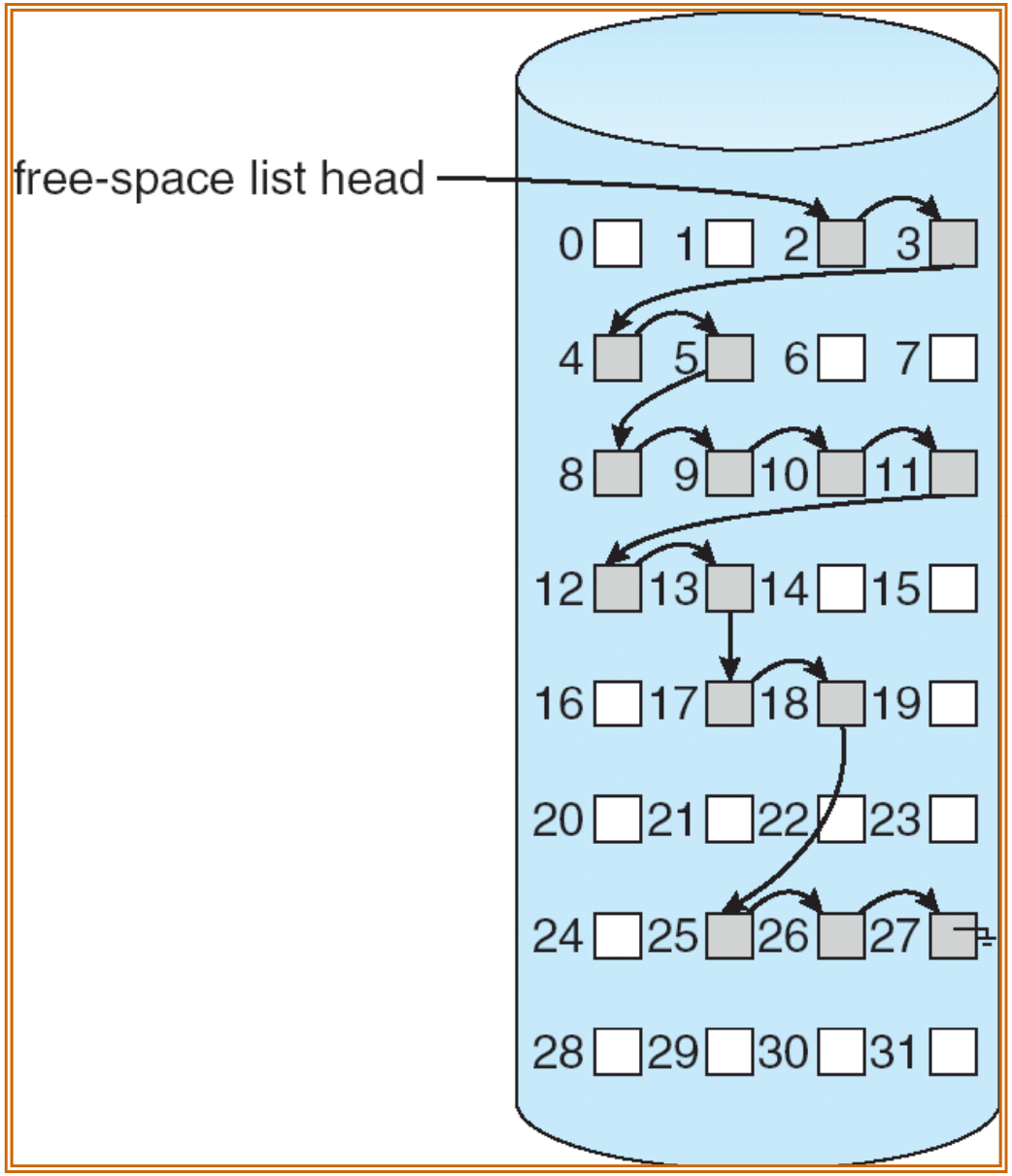
- Bit map requires extra space
  - Example:

    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)
  - Cannot get contiguous space easily
  - No waste of space
- Grouping
  - Modified free-list
- Counting
  - With a variable free-contiguous-block length

# Linked Free Space List on Disk



# END OF CHAPTER 11