File System Implementation

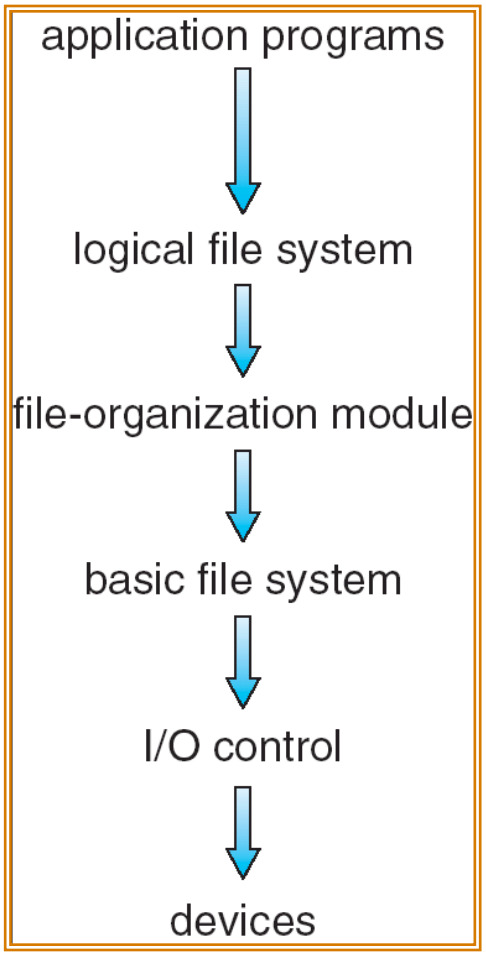
Chapter 11: File System Implementation

- File-System Structure
- File-System Implementation
- Directory Implementation
- Allocation Methods

File-System Structure

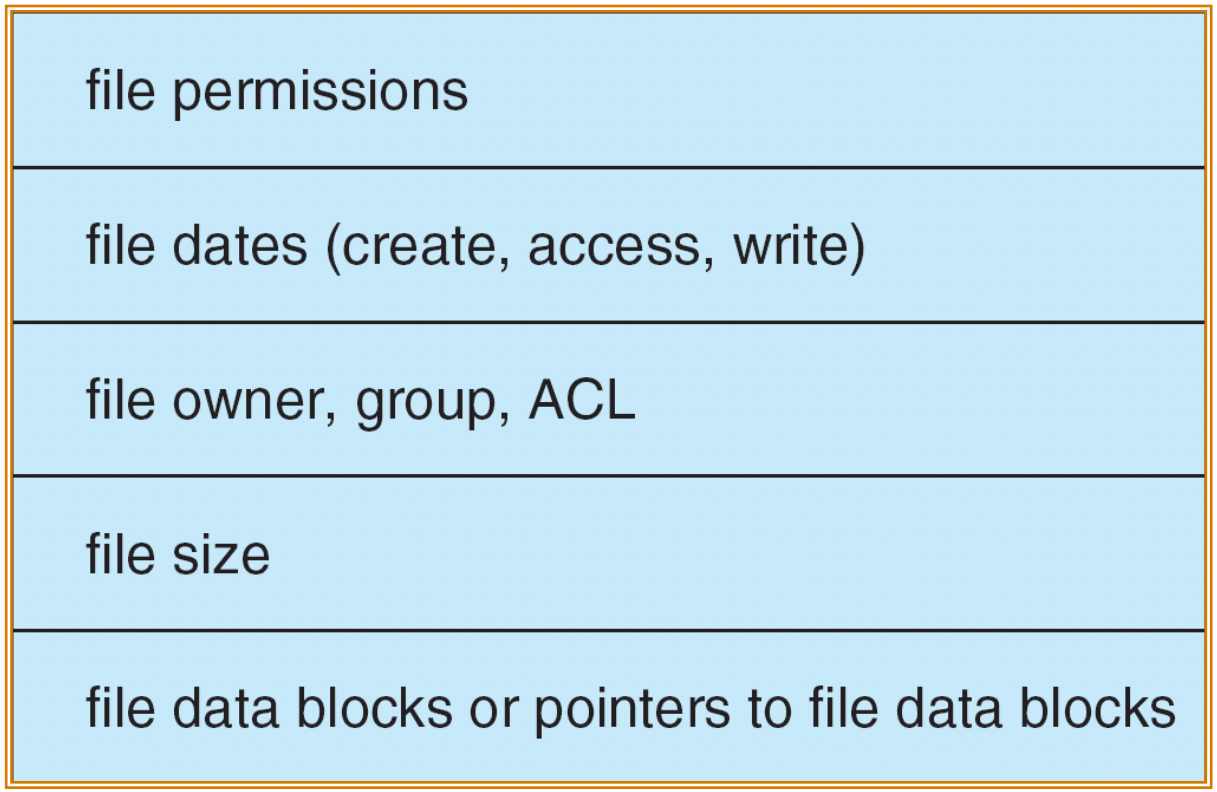
- File structure
  - Logical storage unit
  - Collection of related information
- File system resides on secondary storage (disks)
- File system organized into layers
- **File control block** – storage structure consisting of information about a file

Layered File System



- I/O Control: device drivers and interrupt handlers that talk to the disk
- Basic file system: Generic block reads and writes
  - e.g., read cylinder 73, track 2, sector 10
- File organization: Files and logical blocks
  - Translate logical blocks to physical
  - Manage free space
- Logical file system: Metadata information
  - e.g., owner, permissions, size, etc.

A Typical File Control Block



- FCB has all meta-information about a file
  - Linux calls these ***i-nodes***

Implementing File Operations (1)

- **Create a file:**
  - Find space in the file system, add directory entry.
- **Open a file:**
  - System call specifying name of file.
  - System searches directory structure to find file.
  - System keeps ***current file position pointer*** to the location where next write/read occurs
  - System call returns ***file descriptor*** (a handle) to user process
- **Reading a file:**
  - System call specifying file descriptor and number of bytes to read (and possibly where in memory to stick contents).

Implementing File Operations (2)

- **Writing in a file:**
  - System call specifying file descriptor and information to be written
  - Writes information at location pointed by the files current pointer
- **Repositioning within a file:**
  - System call specifying file descriptor and new location of current pointer
  - (also called a file ***seek*** even though does not interact with disk)
- **Closing a file:**
  - System call specifying file descriptor
  - Call removes current file position pointer and file descriptor associated with process and file
- **Deleting a file:**
  - Search directory structure for named file, release associated file space and erase directory entry
- **Truncating a file:**
  - Keep attributes the same, but reset file size to 0, and reclaim file space.

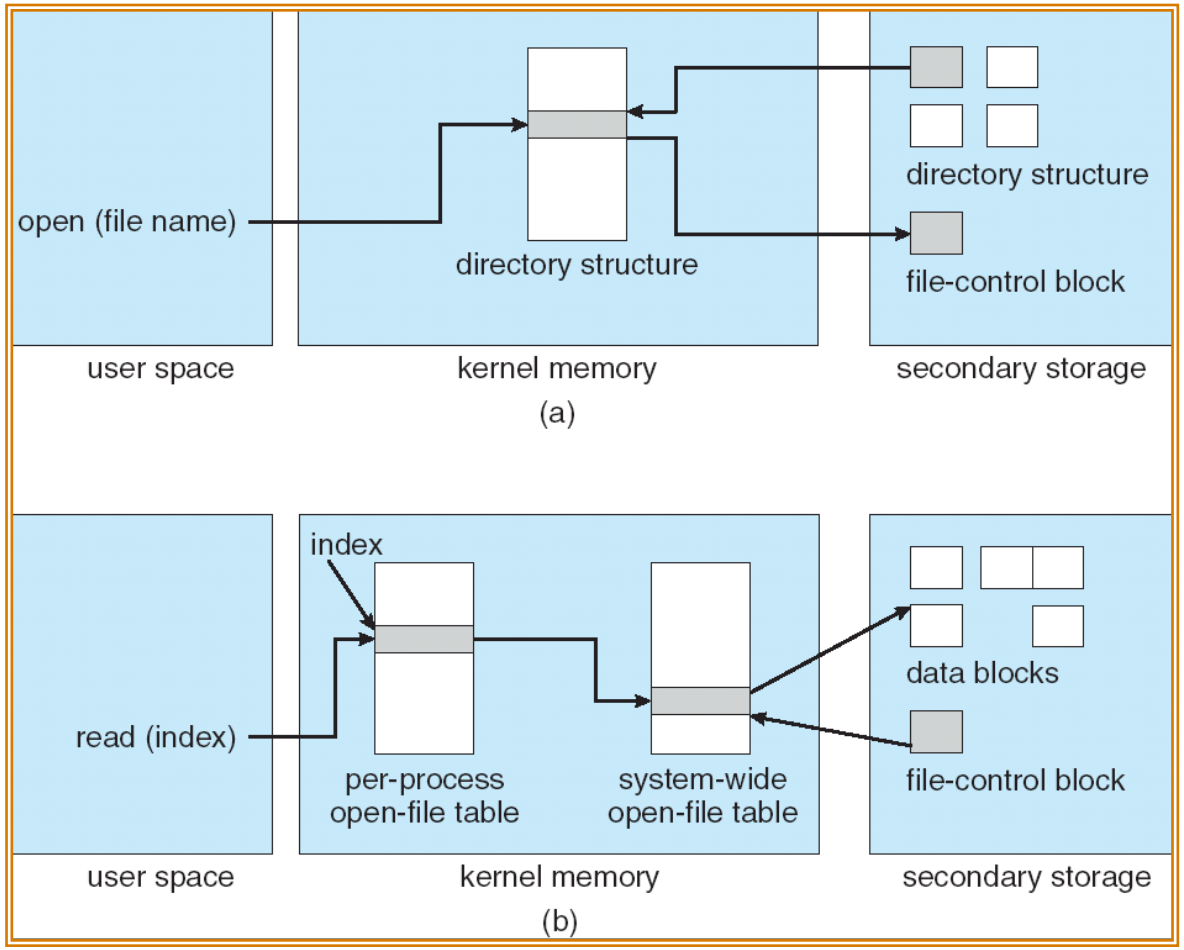
Other File Operations

- Most FS require an open() system call before using a file.
- OS keeps an in-memory table of open files, so when reading a writing is requested, they refer to entries in this table via a file descriptor.
- On finishing with a file, a close() system call is necessary. (creating & deleting files typically works on closed files)

Multiple Users of a File

- OS typically keeps two levels of internal tables:
- **Per-process table**
  - Information about the use of the file by the user (e.g. current file position pointer)
- **System-wide table**
  - Gets created by first process which opens the file
  - Location of file on disk
  - Access dates
  - File size
  - Count of how many processes have the file open (used for deletion)

In-Memory File System Structures



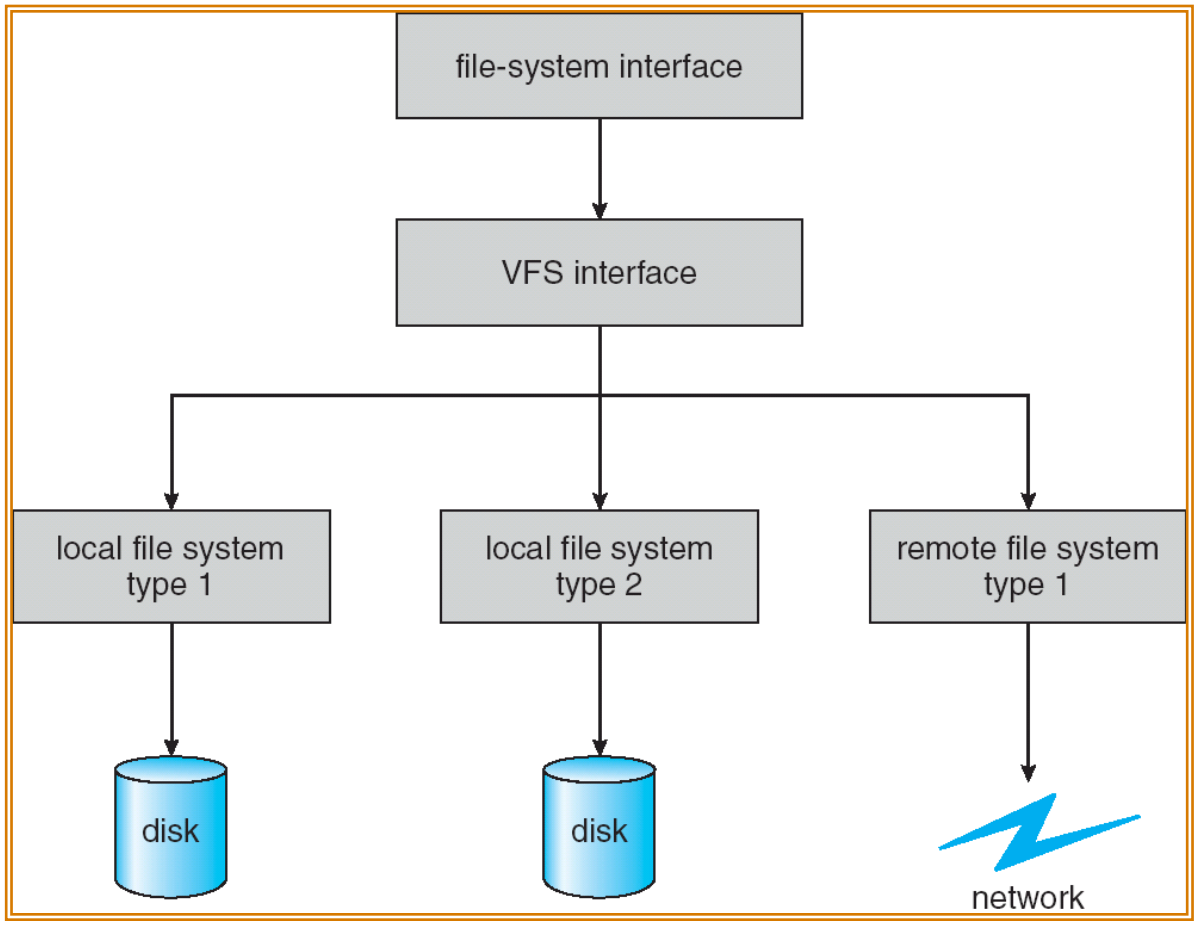
opening a file

reading a file

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



Allocating and Storing Files

- An allocation method refers to how disk blocks are allocated for files:

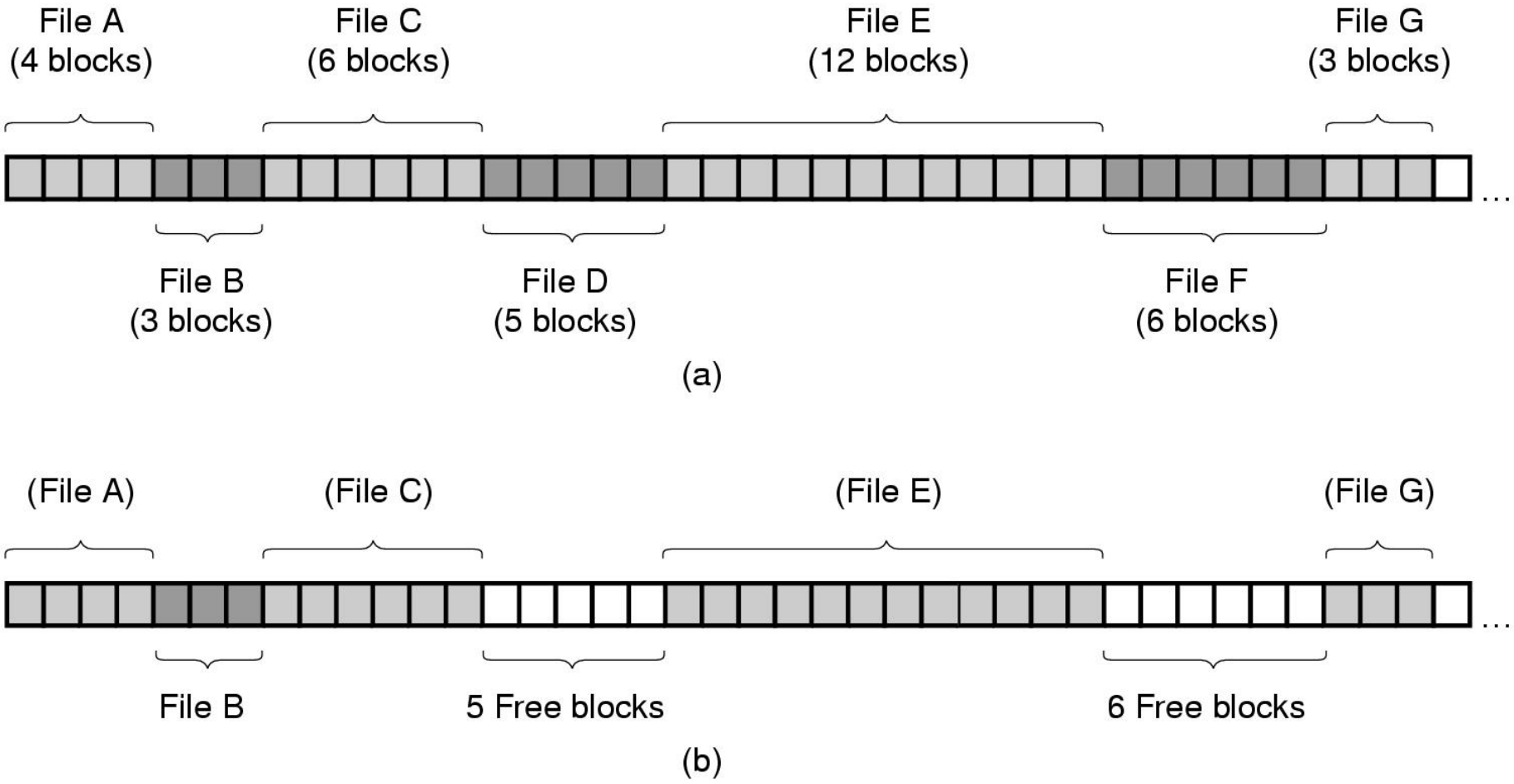
- **Contiguous allocation**
  - All bytes together, in order

- **Linked allocation**
  - Each block points to the next block

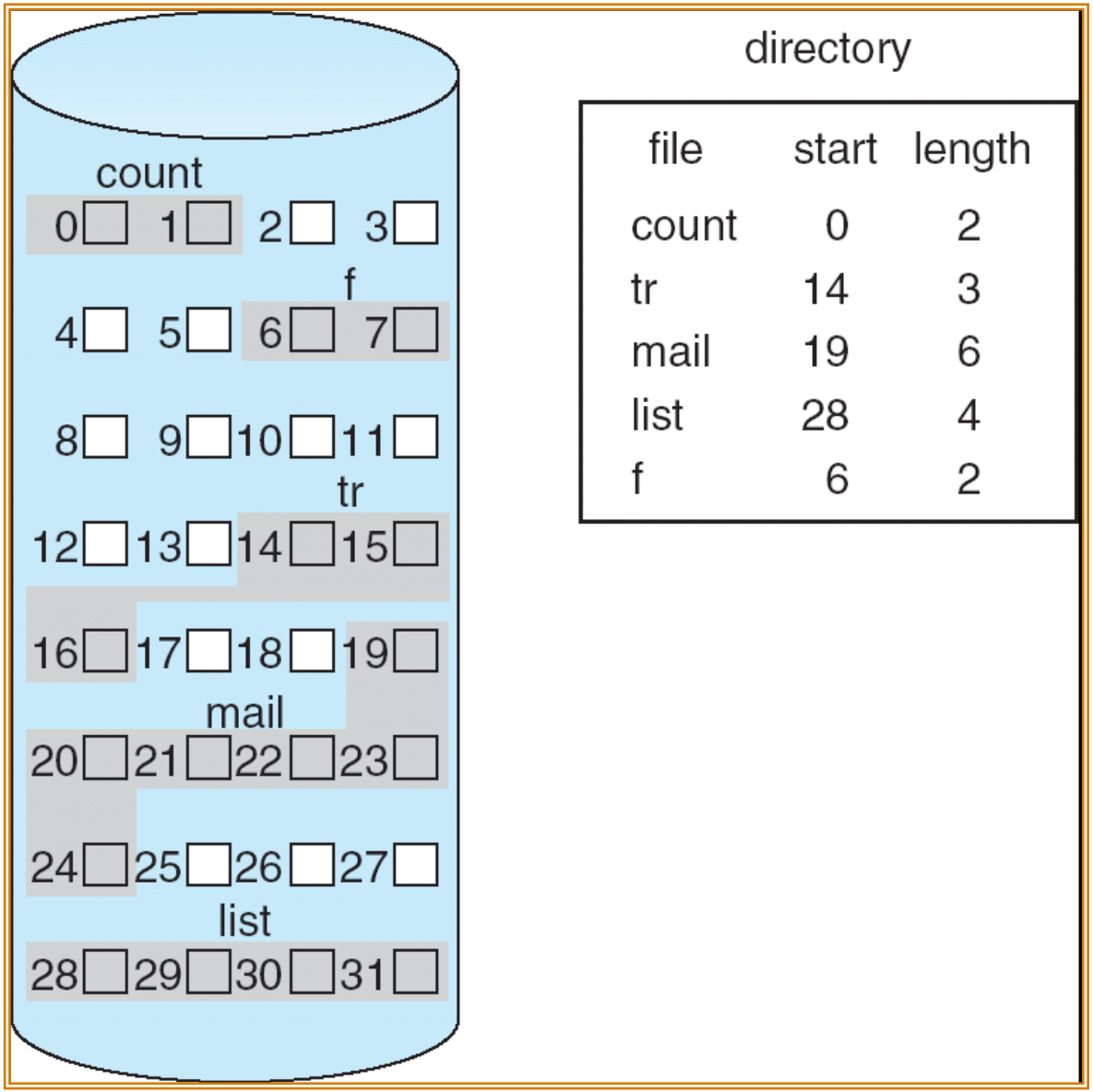
- **Indexed allocation**
  - An index block contains pointers to many other blocks

Contiguous Allocation

- Allocate files contiguously on disk



Contiguous Allocation of Disk Space

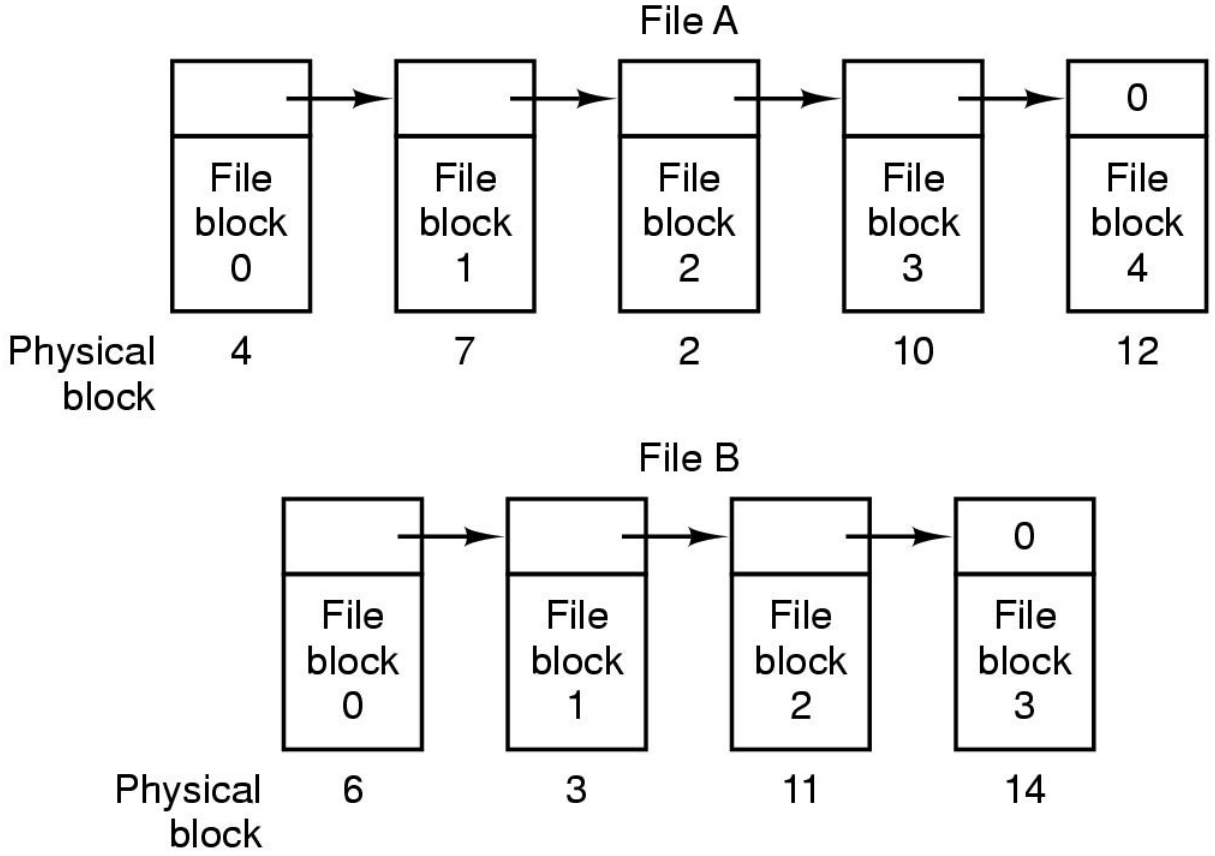


Contiguous Allocation

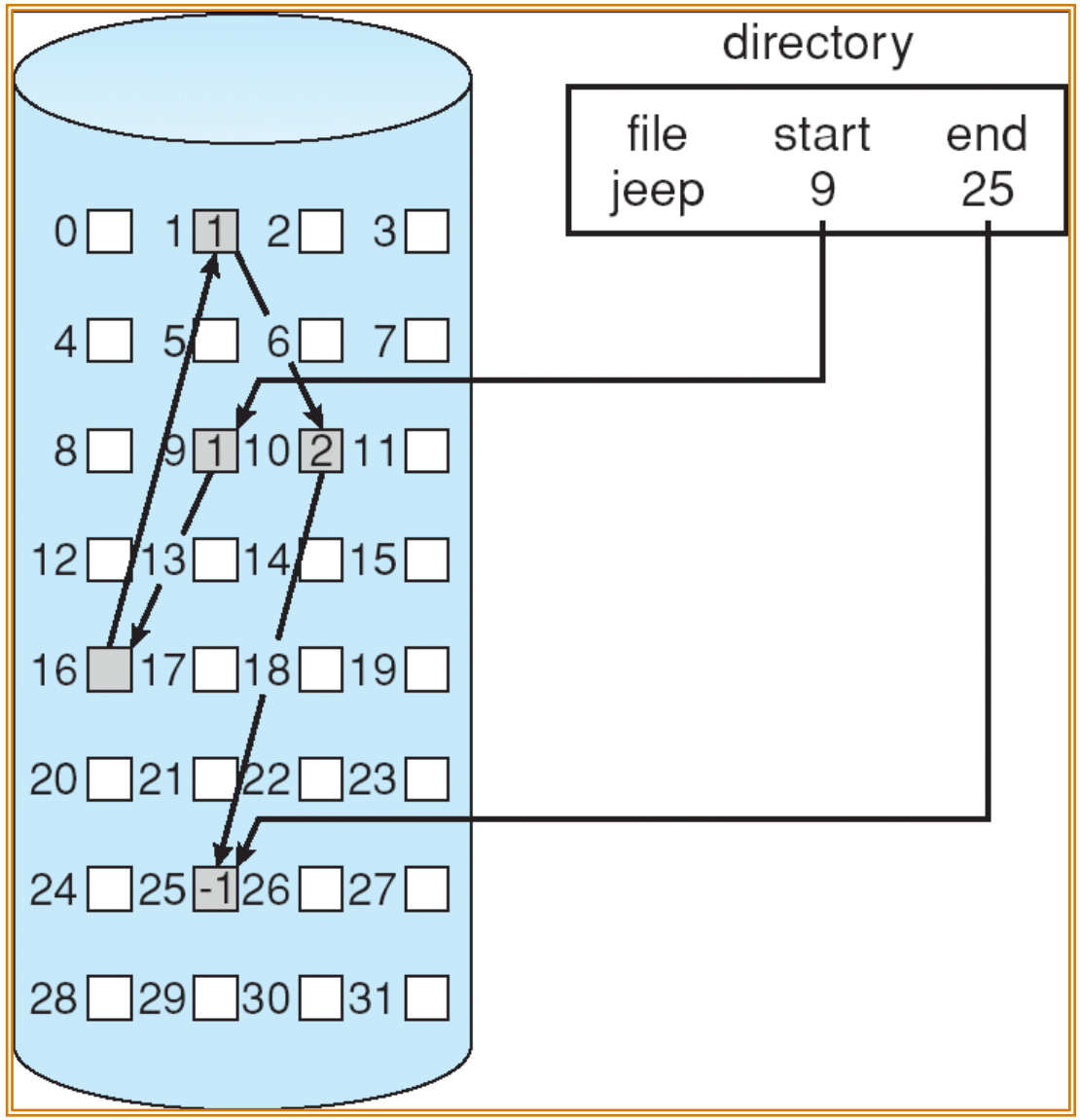
- **Pros:**
  - Simple: state required per file is start block and size
  - Performance: entire file can be read with one seek
- **Cons:**
  - Files can't grow
  - Fragmentation: external frag is bigger problem
  - Wastes space
- Used in CDROMs, DVDs

Linked List Allocation

- Each file is stored as linked list of blocks
  - First word of each block points to next block
  - Rest of disk block is file data



Linked Allocation

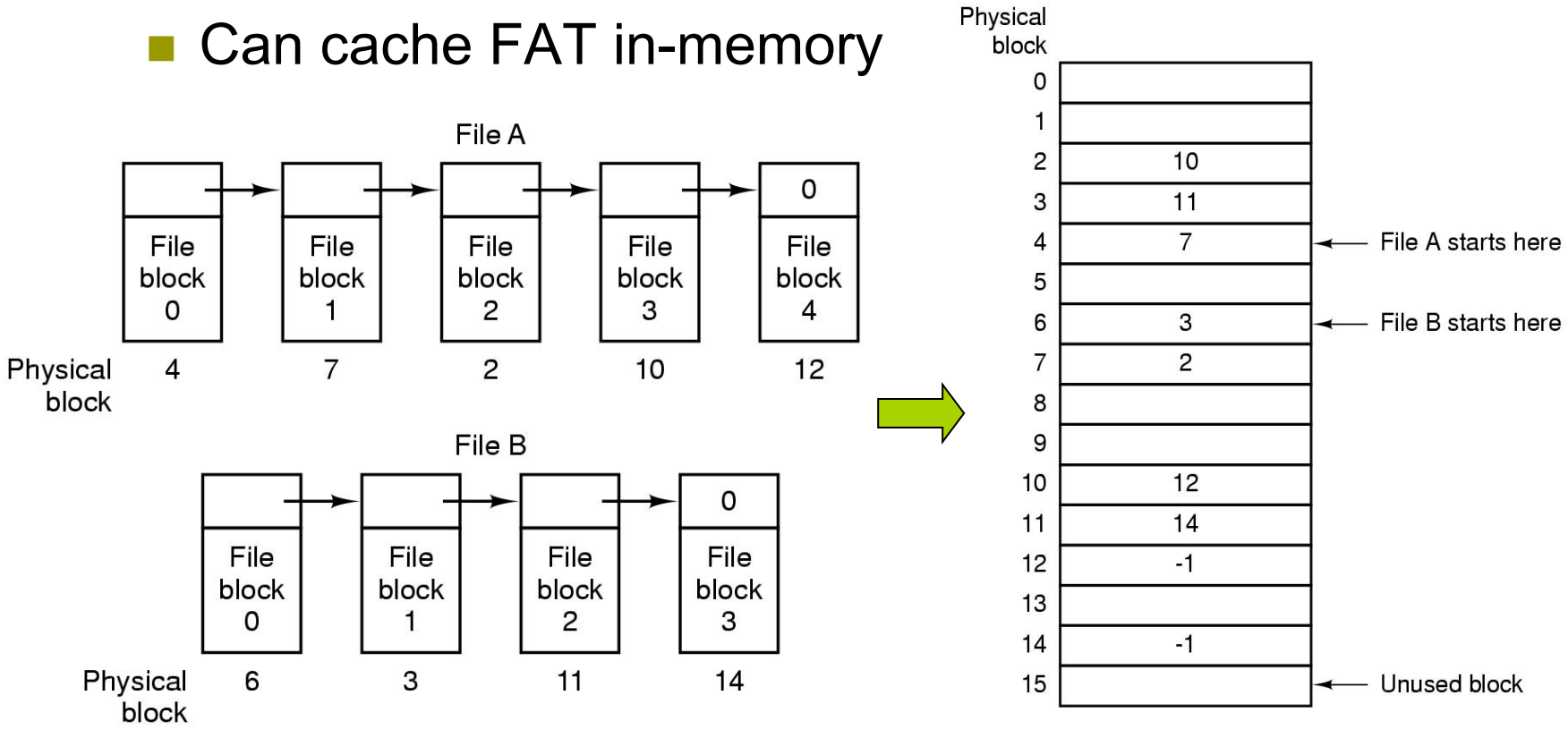


Linked List Allocation

- **Pros**:
  - No space lost to external fragmentation
  - Disk only needs to maintain first block of each file
- **Cons:**
  - Random access is costly
  - Overheads of pointers

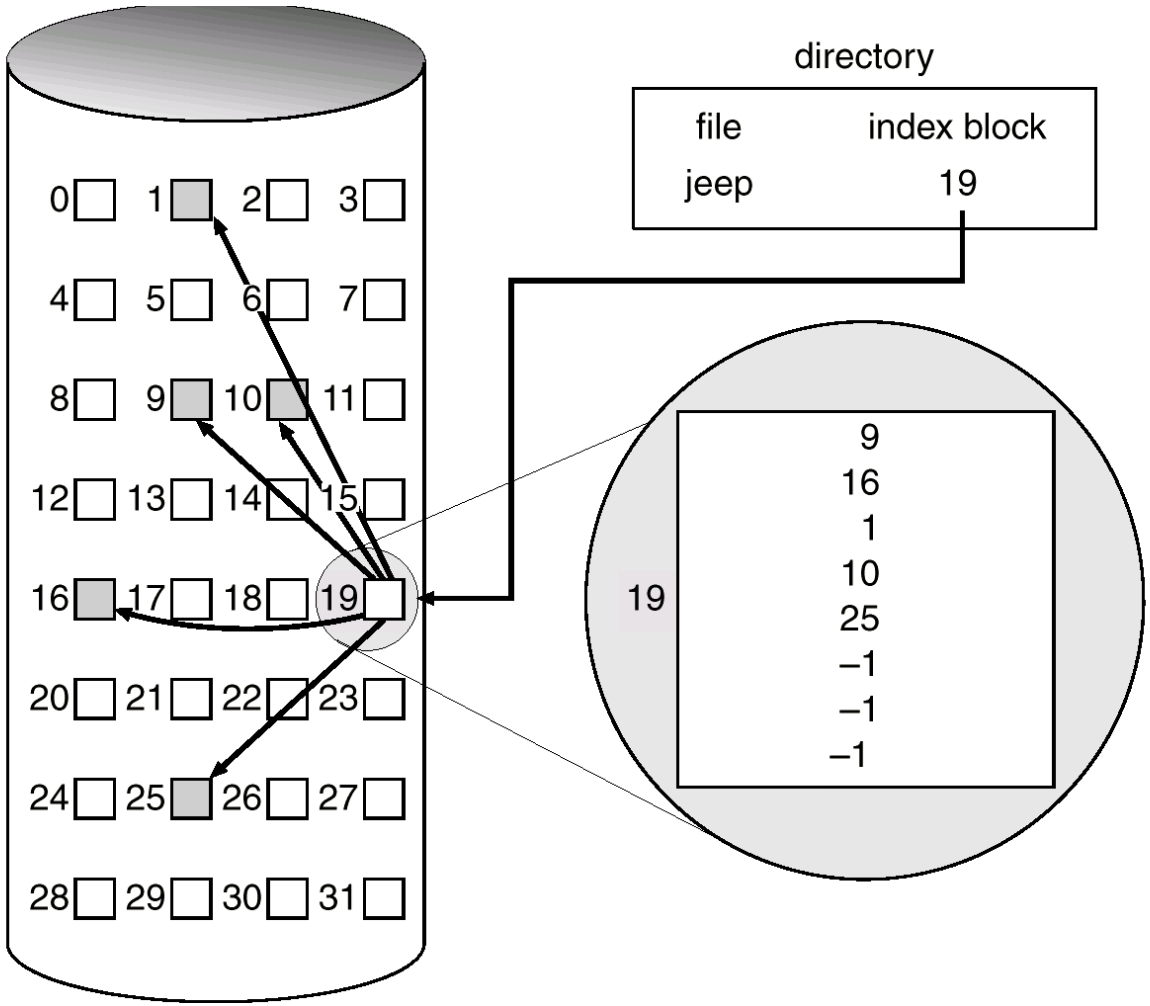
Example: MS-DOS File System

- Implement a linked list allocation using a table
  - Called File Allocation Table (FAT)
  - Take pointer away from blocks, store in this table
  - Can cache FAT in-memory



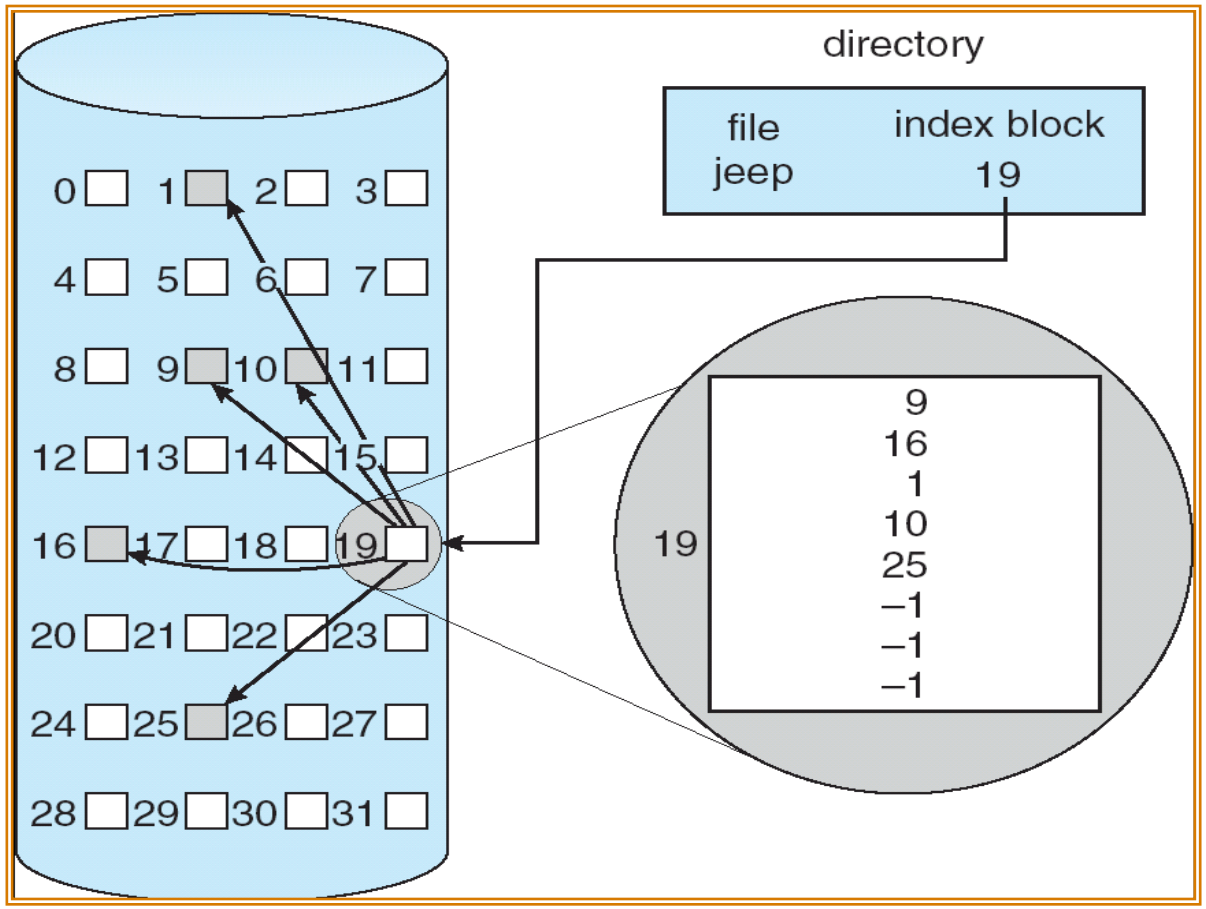
Indexed Allocation

- Index block contains pointers to each data block
- **Pros?**
  - Space (max open files * size per I-node)
- **Cons?**
  - what if file expands beyond I-node address space?



Indexed Allocation

- Brings all pointers to data blocks together into an *index block.*



Implementing Directories

- Directory: map ASCII file name to file attributes & location
- When a file is opened, OS uses path name to find dir
  - Directory has information about the file's disk blocks
    - Whole file (contiguous), first block (linked-list) or I-node (indexed allocation)
  - Directory also has attributes of each file
- 2 options: entries have all attributes, or point to file I-node