File System Implementation

Silberschatz, Galvin
Jul-Nov 2018
Moumita Patra

- **File system**- provides the mechanism for on-line storage and access to file contents, including data and programs
- File system resides in secondary storage

➔ Provide user interface to storage, mapping logical to physical

➔ File systems provide efficient and convenient access to disk

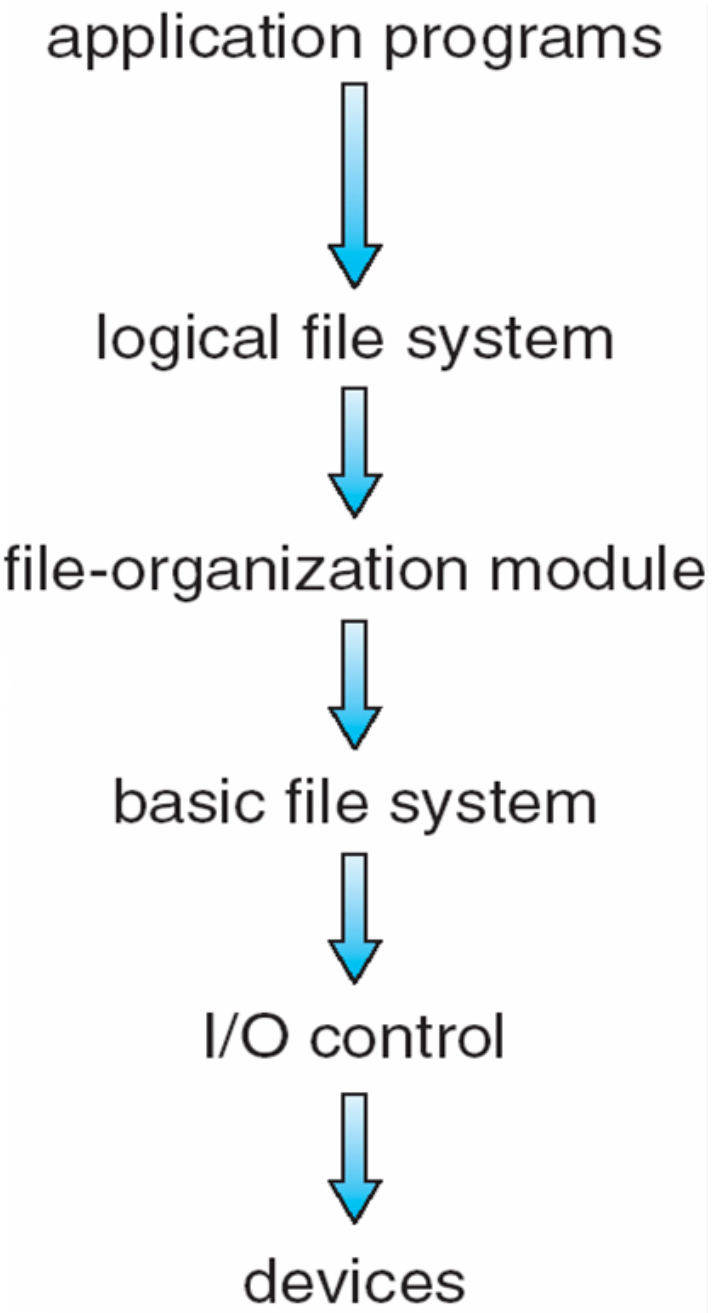
- Problems to handle:

➔ How the file system will look to the user

➔ Map the logical file system onto the physical secondary storage devices

- **File Control Block**- storage structure consisting of information about a file
- **Device driver** controls the physical device
- File system organized into layers

Layered File System



Layered File System

- I/O Control- consists of device drivers and interrupt handlers to transfer information between the main memory and the disk system
- Basic file system- issues generic commands to the appropriate device driver to read and write physical blocks on the disk
- File organization module- understands files, logical address and physical blocks
- Translates logical block # to physical block #
- Manages free space, disk allocation

- Logical file system- manages metadata information
- Translates file name into file number, file handle , location by maintaining FCBs
- Directory management
- Protection

Layered sturture for file system helps in-

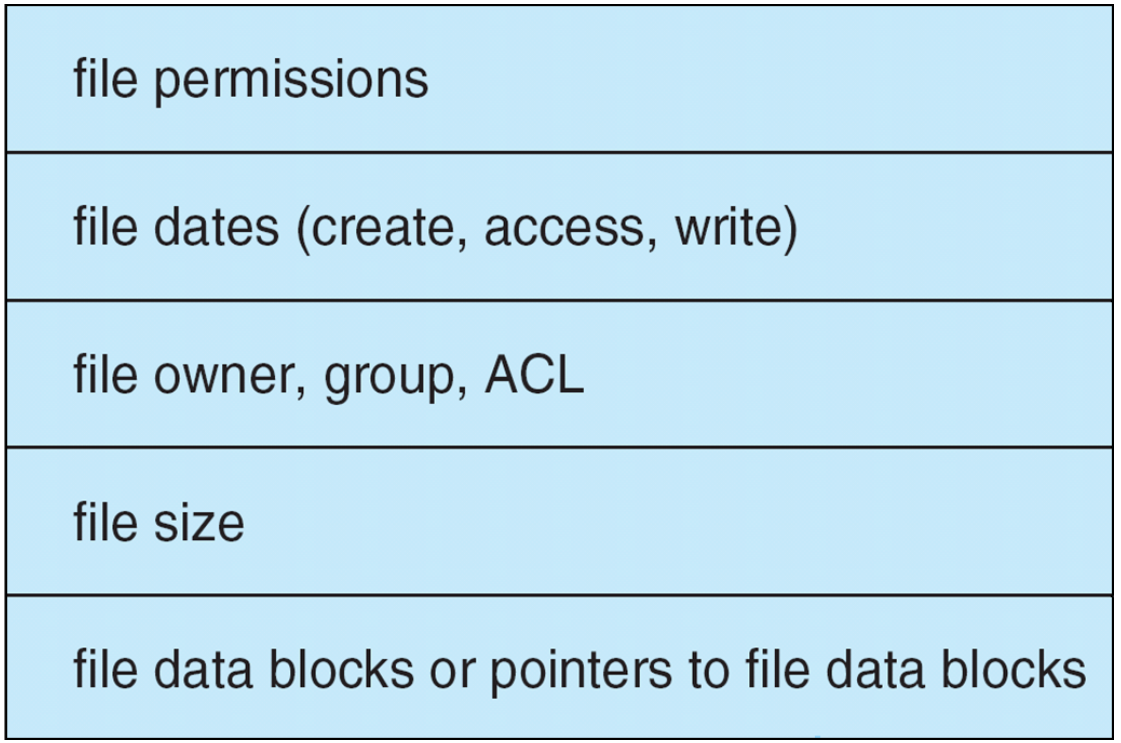
- Duplication of code is minimized
- Layering may introduce more OS overhead
- Challenges- how many layers to use? what each layer should do?

File-System Implementation

- We have system calls at the API level, but how do we implement their functions?
  - On-disk and in-memory structures
- **Boot control block** contains info needed by system to boot OS from that volume
  - Needed if volume contains OS, usually first block of volume
- **Volume control block (superblock, master file table)** contains volume details
  - Total # of blocks, # of free blocks, block size, free block pointers or array
- Directory structure organizes the files
  - Names and inode numbers, master file table

File-System Implementation (Cont.)

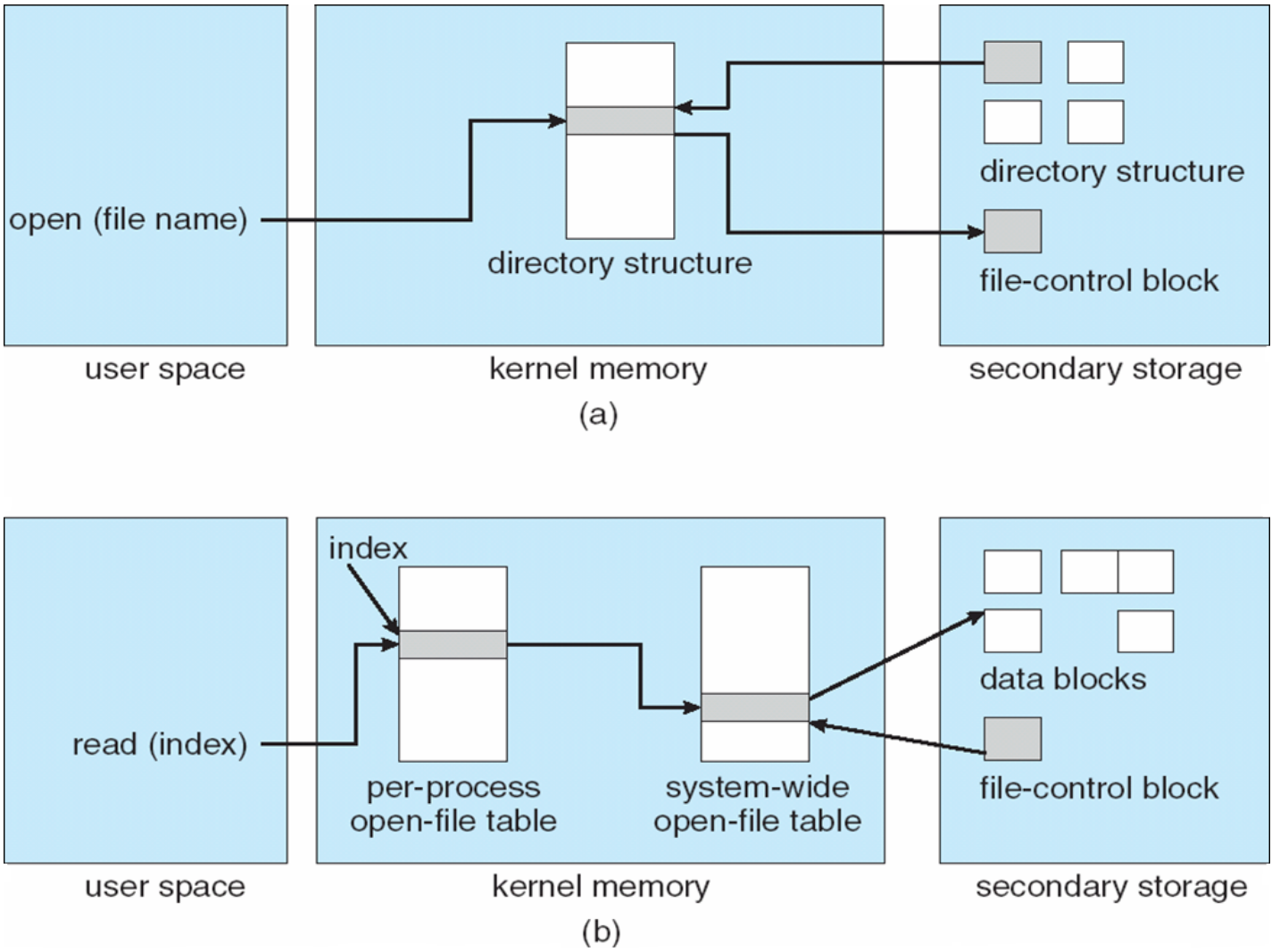
- Per-file **File Control Block (FCB)** contains many details about the file
  - inode number, permissions, size, dates
  - NTFS stores info in master file table using relational DB structures



In-Memory File Information

- In-memory information- used for file system management and performance improvement via caching
- In-memory structures:
- Mount table- contains information about each mounted volume
- In-memory directory structure cache- holds directory info of recently accessed directories
- System-wide open file table- contains a copy of the FCB of each open file as well as other info
- Per-process open-file table- contains a pointer to the appropriate entry in the system-wide open-file table
- Buffers- hold file-system blocks when they are being read/written from/to disks.

In-Memory File System Structures



Partitions and Mounting

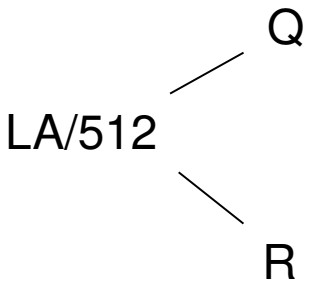
- Partition can be a volume containing a file system (“cooke d”) or **raw** – just a sequence of blocks with no file system
- Boot block can point to boot volume or boot loader set of blocks that contain enough code to know how to load the kernel from the file system
  - Or a boot management program for multi-os booting
- **Root partition** contains the OS, other partitions can hold other Oses, other file systems, or be raw
  - Mounted at boot time
  - Other partitions can mount automatically or manually
- At mount time, file system consistency checked
  - Is all metadata correct?
    - If not, fix it, try again
    - If yes, add to mount table, allow access

Allocation Methods - Contiguous

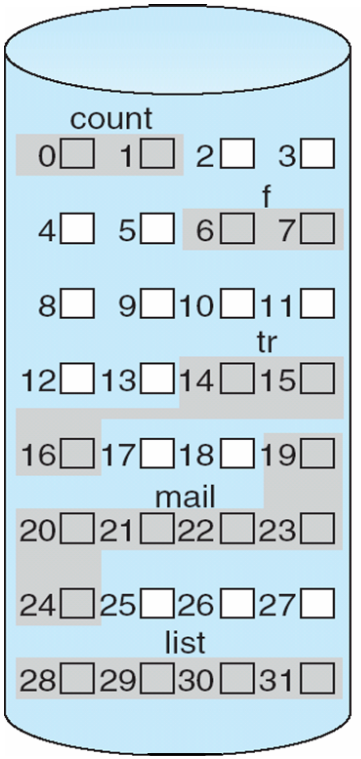
- An allocation method refers to how disk blocks are allocated for files:
- **Contiguous allocation** – each file occupies set of contiguous blocks
  - Best performance in most cases
  - Simple – only starting location (block #) and length (number of blocks) are required
  - Problems include finding space for file, knowing file size, external fragmentation, need for **compaction off-line** (**downtime**) or **on-line**

Contiguous Allocation

- Mapping from logical to physical



Block to be accessed = Q + starting address
Displacement into block = R



directory

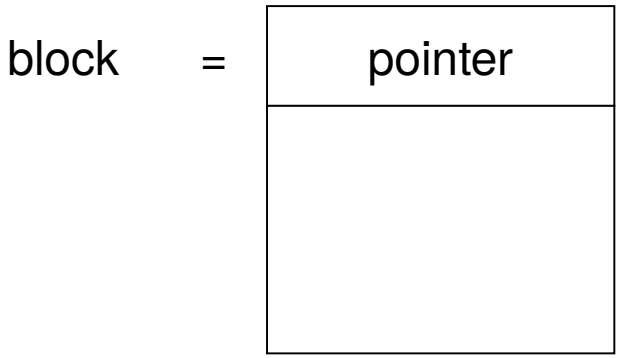
file	start	length
count	0	2
tr	14	3
mail	19	6
list	28	4
f	6	2

Allocation Methods - Linked

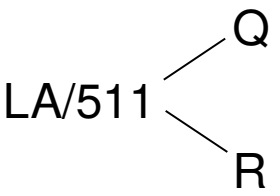
- **Linked allocation** – each file a linked list of blocks
  - File ends at null pointer
  - No external fragmentation
  - Each block contains pointer to next block
  - No compaction, external fragmentation
  - Free space management system called when new block needed
  - Improve efficiency by clustering blocks into groups but increases internal fragmentation
  - Reliability can be a problem
  - Locating a block can take many I/Os and disk seeks

Linked Allocation

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



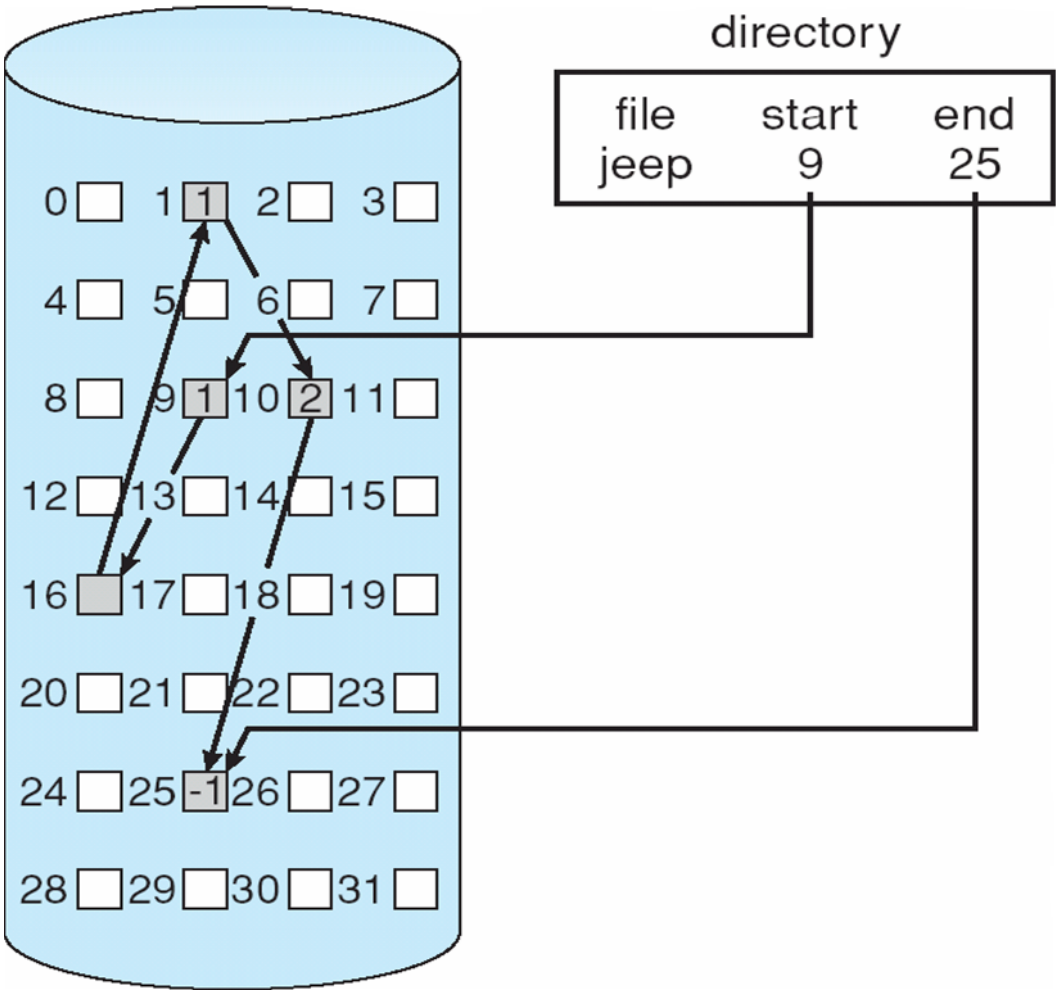
- Mapping



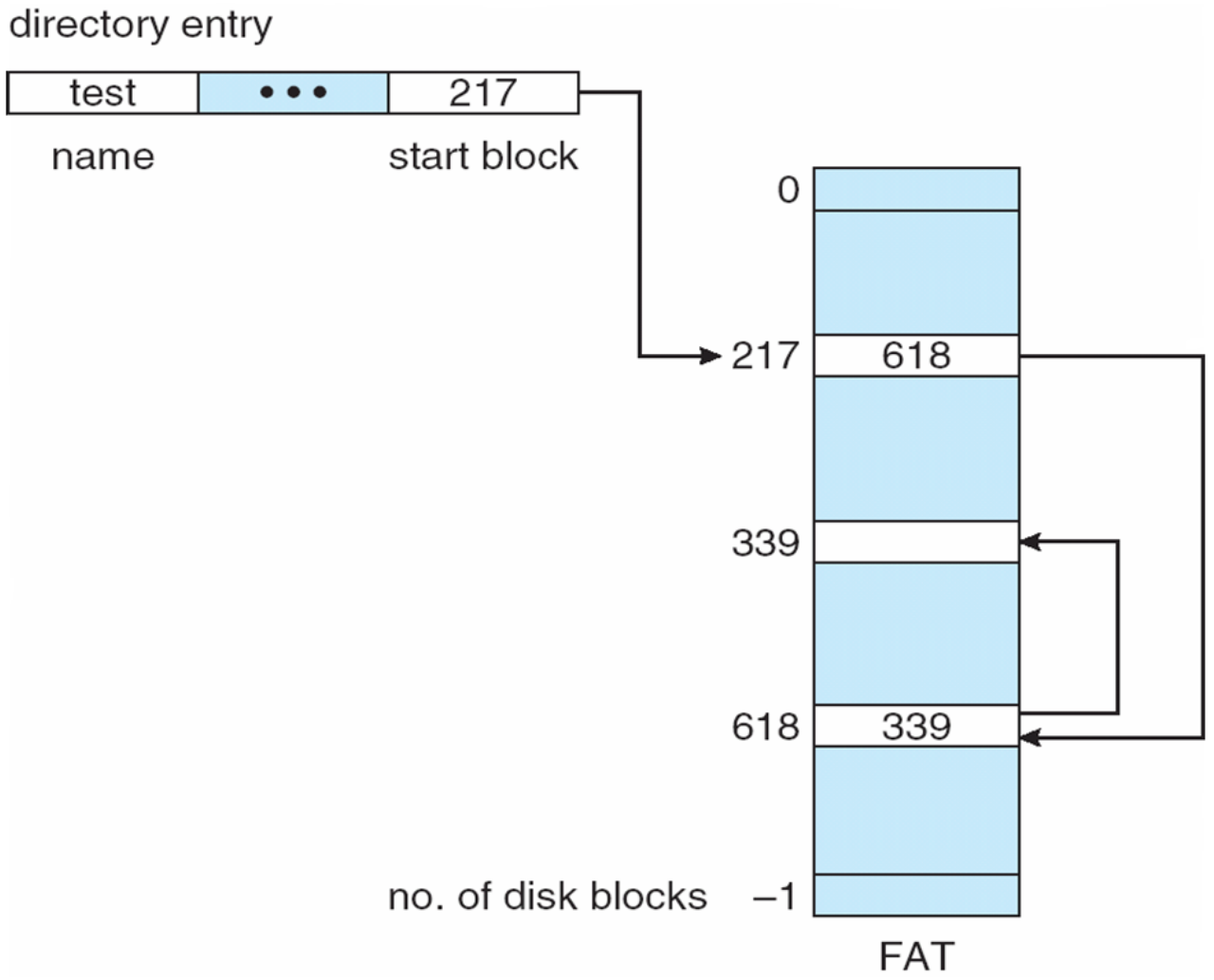
Block to be accessed is the Qth block in the linked chain of blocks representing the file.

Displacement into block = R + 1

Linked Allocation



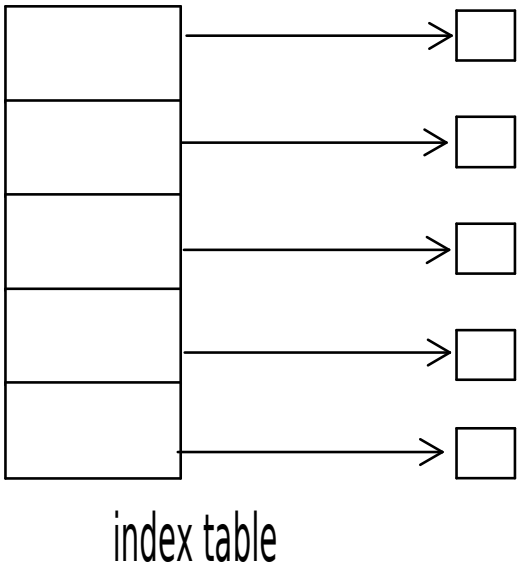
File-Allocation Table



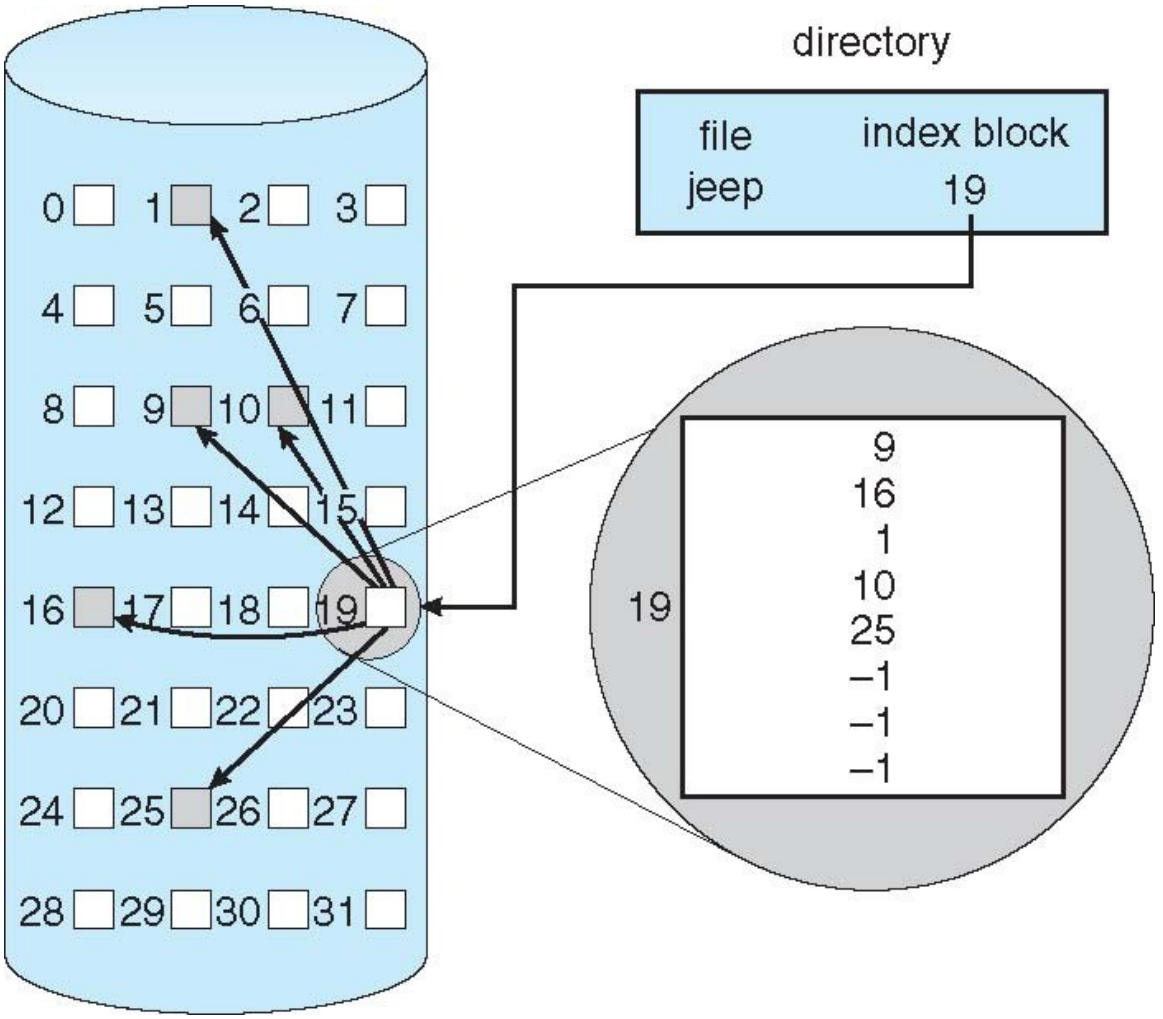
Allocation Methods - Indexed

- **Indexed allocation**
  - Each file has its own **index block**(s) of pointers to its data blocks

- Logical view

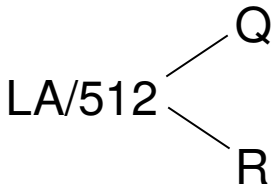


Example of Indexed Allocation



Indexed Allocation (Cont.)

- Need index table
- Random access
- Dynamic access without external fragmentation, but have overhead of index block
- Mapping from logical to physical in a file of maximum size of 256K bytes and block size of 512 bytes. We need only 1 block for index table



Q = displacement into index table
R = displacement into block