File System Implementation

Unit-IV

Session Objectives

- To describe the details of implementing local file systems and directory structures
- To describe the implementation of remote file systems
- To discuss block allocation and free-block algorithms and trade-offs

Session Outcomes

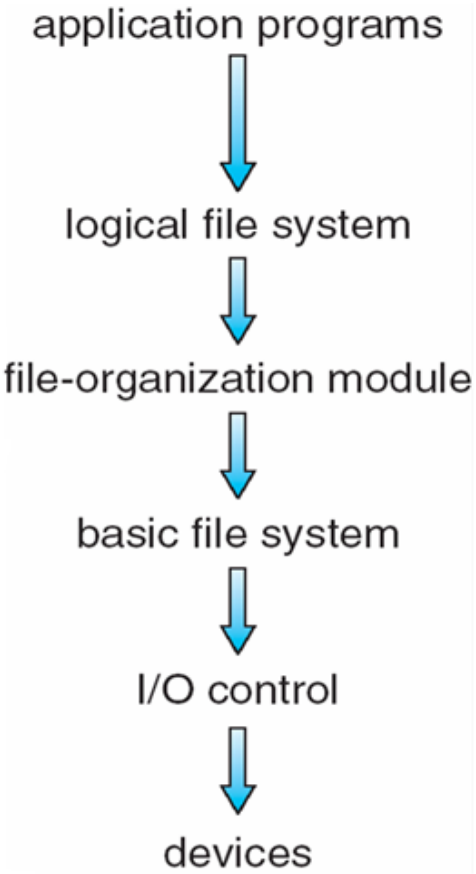
At the end of this session, participants will be able to

- Discuss
  File-System Structure
  File-System Implementation
  Directory Implementation
  Allocation Methods
  Free-Space Management
  Efficiency and Performance
  Recovery

File-System Structure

- File structure
- File system resides on secondary storage (disks)
  - Provided user interface to storage, mapping logical to physical
  - Provides efficient and convenient access to disk by allowing data to be stored, located retrieved easily
- Disk provides rewrite and random access
  I/O transfers performed in blocks of sectors (usually 512 bytes)
- 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



File System Layers

- Device drivers manage I/O devices at the I/O control layer Given commands like **read drive1, cylinder 72, track 2, sector 10, into memory location 1060** outputs low-level hardware specific commands to hardware controller
- Basic file system given command like **retrieve block 123** translates to device driver
- Also manages memory buffers and caches (allocation, freeing, replacement)
  – Buffers hold data in transit
  – Caches hold frequently used data
- **File organization module** understands files, logical address, and physical blocks
- Translates **logical block to physical block**
- Manages free space, disk allocation

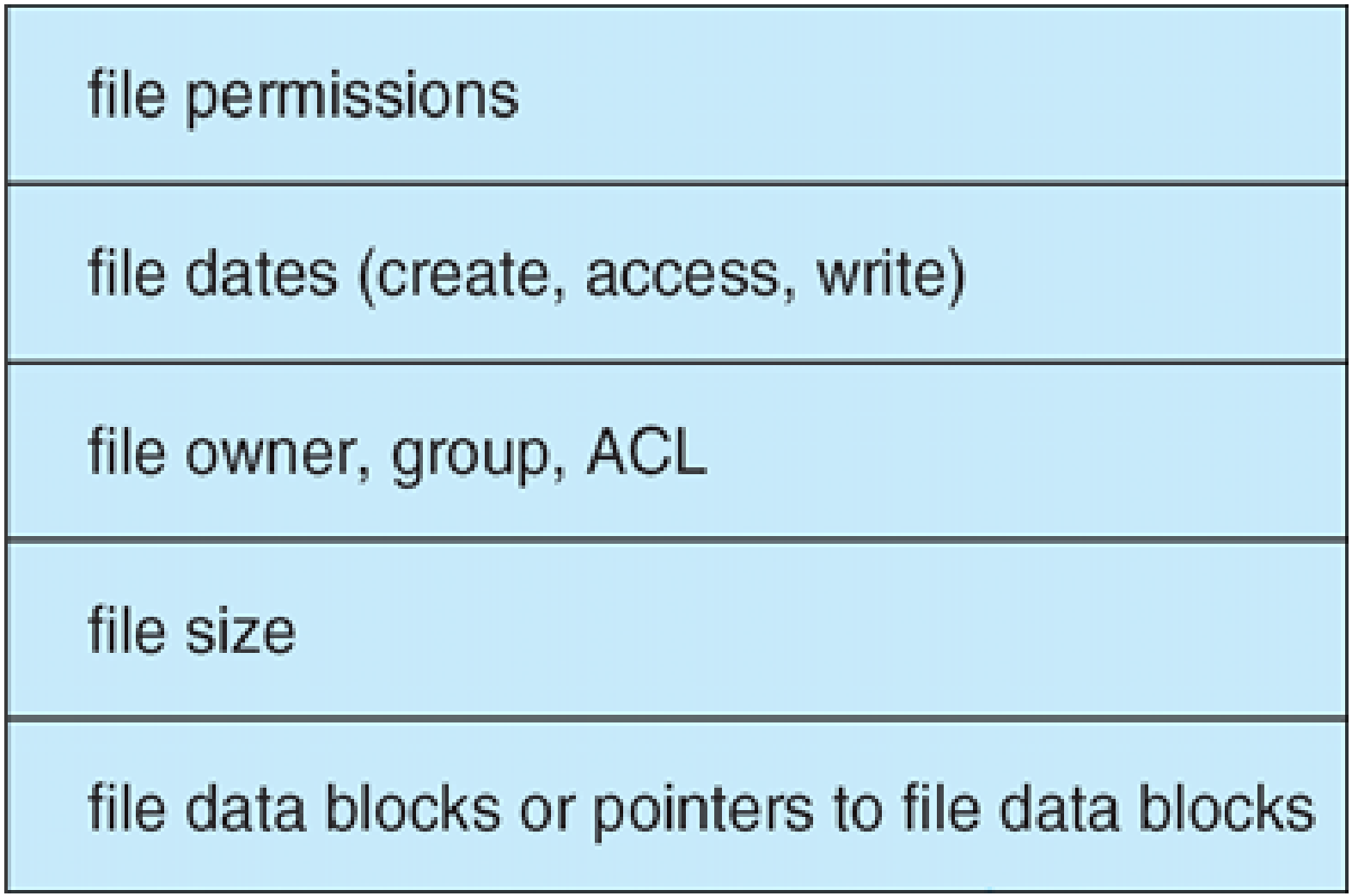
File System Layers

- Logical file system manages metadata information
  - Translates file name into file number, file handle, location by maintaining file control blocks (inodes in UNIX)
  - Directory management
  - Protection
- Layering useful for reducing complexity and redundancy, but adds overhead and can decrease performance
- Translates file name into file number, file handle, location by maintaining file control blocks (inodes in UNIX)

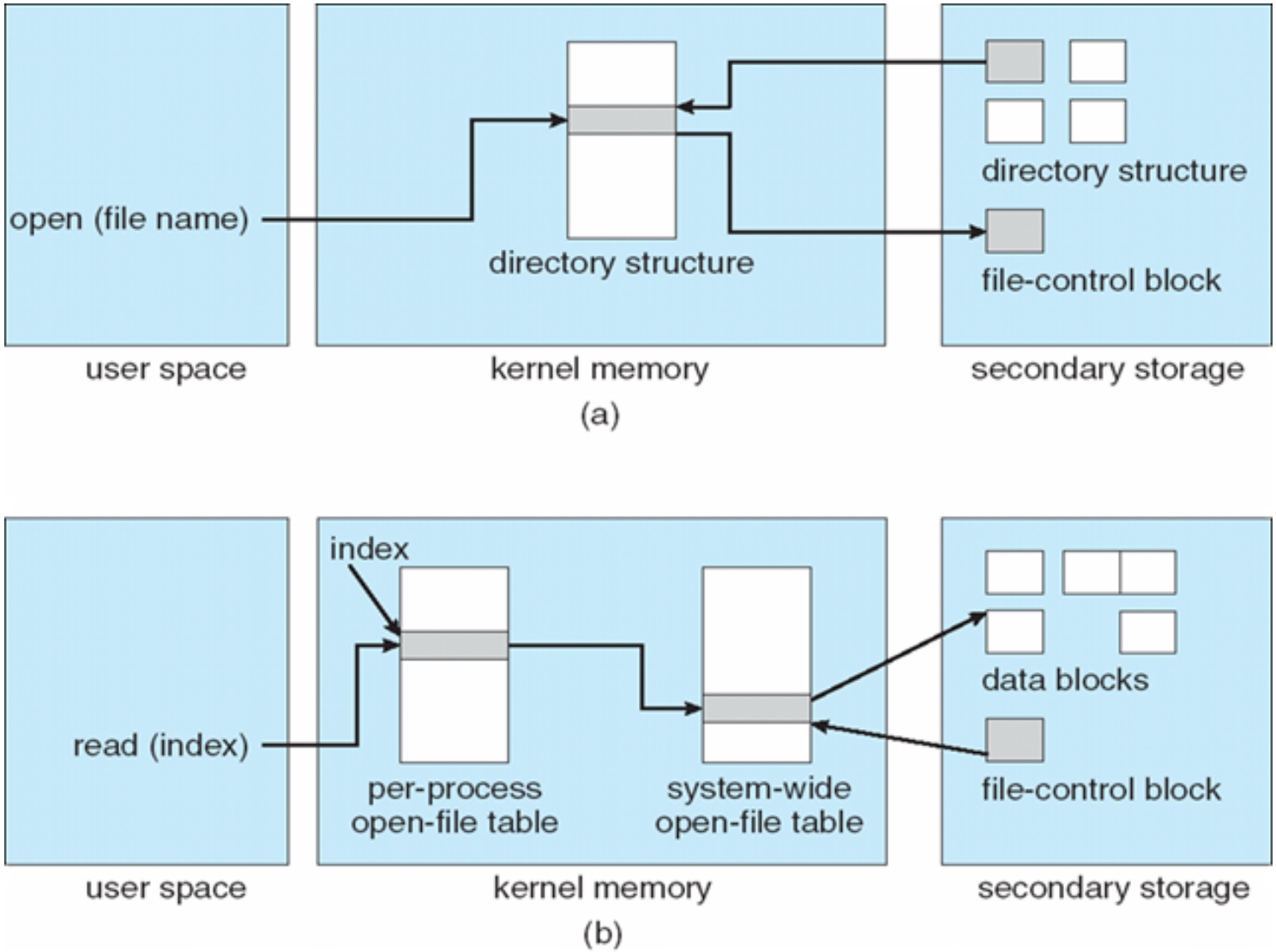
File-System Implementation

- **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 number of blocks, number of free blocks, block size, free block pointers or array
- Directory structure organizes the files
  Names and inode numbers, master file table
- Per-file **File Control Block (FCB)** contains many details about the file
  inode number, permissions, size, dates
  NTFS stores into in master file table using relational DB structures

File-System Implementation



In-Memory File System Structures



Partitions and Mounting

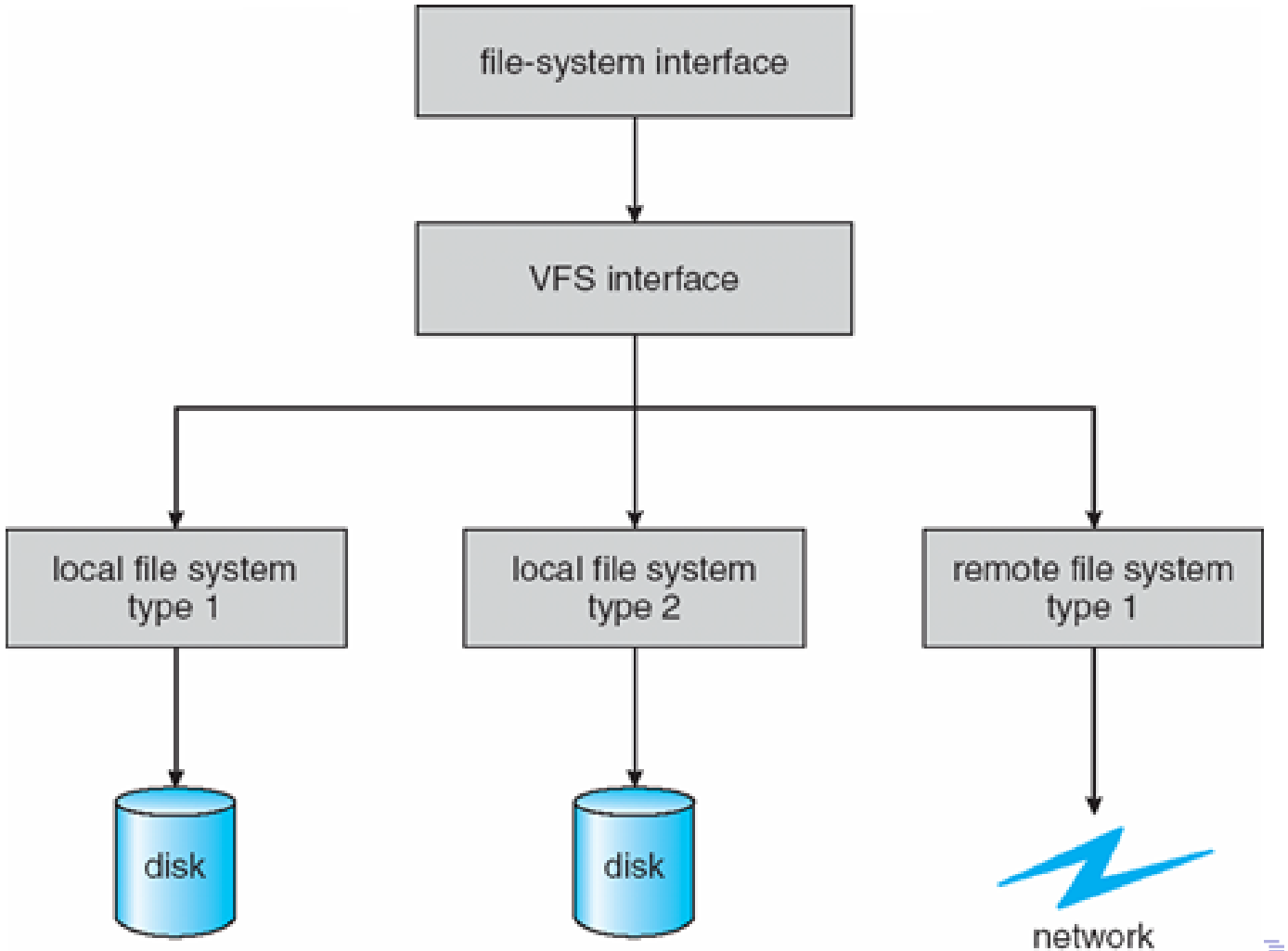
- Partition can be a volume containing a file system (“cooked”) or **raw** – just a sequence of blocks with no file system
- Boot block can point to boot volume or boot loader set of blocks that contain enough code to know how to load the kernel from the file system
  - Or a boot management program for 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

Virtual File Systems

- Virtual File Systems (VFS) on Unix 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
  - Separates file-system generic operations from implementation details
  - Implementation can be one of many file systems types, or network file system
    Implements vnodes which hold inodes or network file details
  - Then dispatches operation to appropriate file system implementation routines

Virtual File Systems

The API is to the VFS interface, rather than any specific type of file system



Directory Implementation

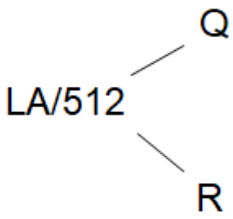
- Linear list of file names with pointer to the data blocks
  - Simple to program
  - Time-consuming to execute
  - Linear search time
  - Could keep ordered alphabetically via linked list
- Hash Table – linear list with hash data structure
  - Decreases directory search time
  - Collisions – situations where two file names hash to the same location
  - Only good if entries are fixed size, or use chained-overflow method

Allocation Methods - Contiguous

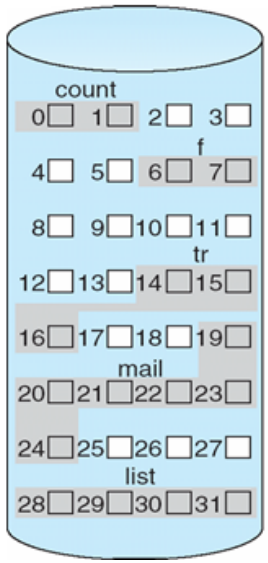
- An allocation method refers to how disk blocks are allocated for files:
- **Contiguous allocation** - each file occupies set of contiguous blocks
  - Best performance in most cases
  - Simple - only starting location (block no.) and length (number of blocks) are required
  - Problems include finding 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

Extent-Based Systems

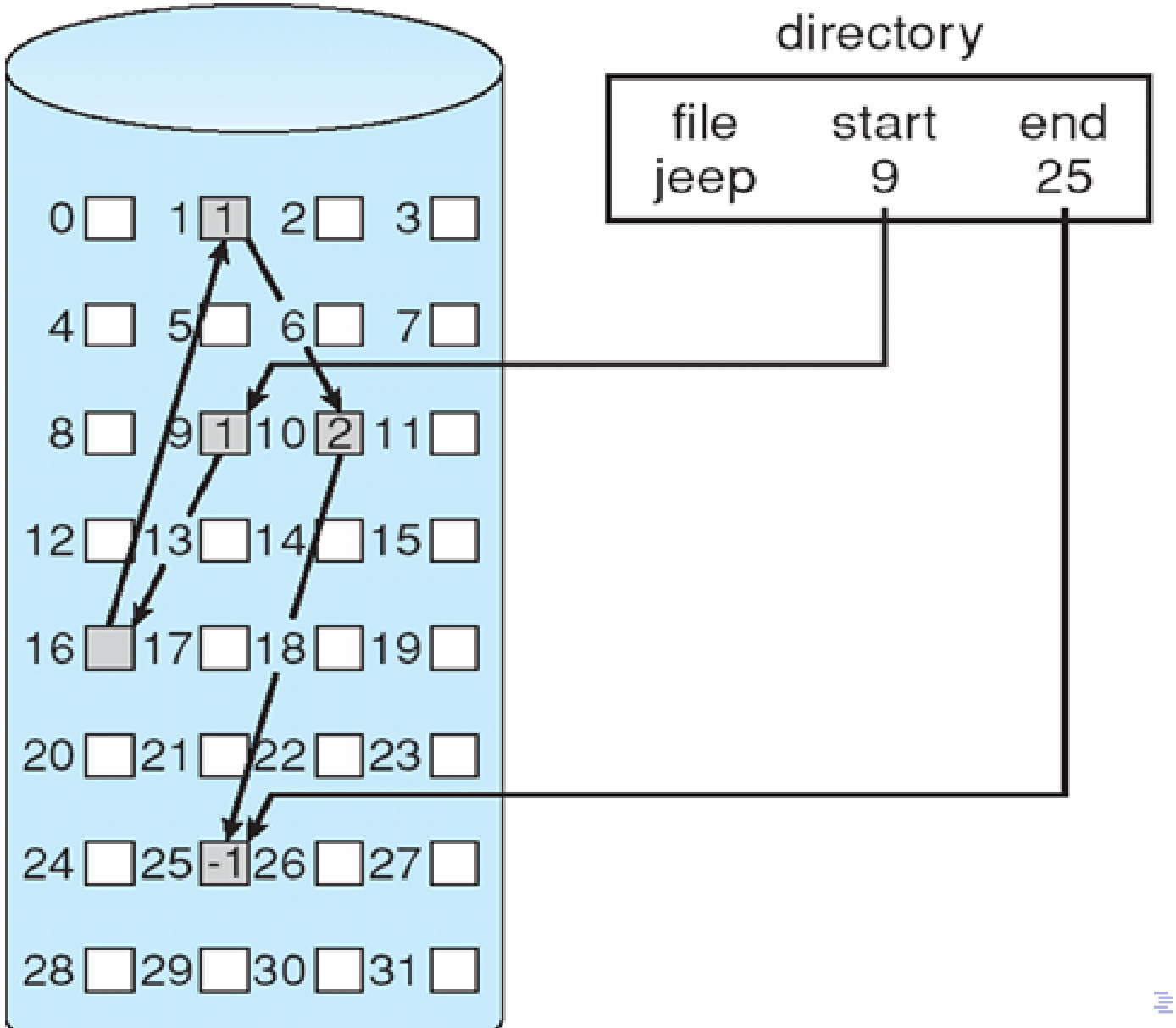
- Extent-based file systems allocate disk blocks in extents
- An extent is a contiguous block of disks
- Extents are allocated for file allocation
- A file consists of one or more extents

Allocation Methods - Linked

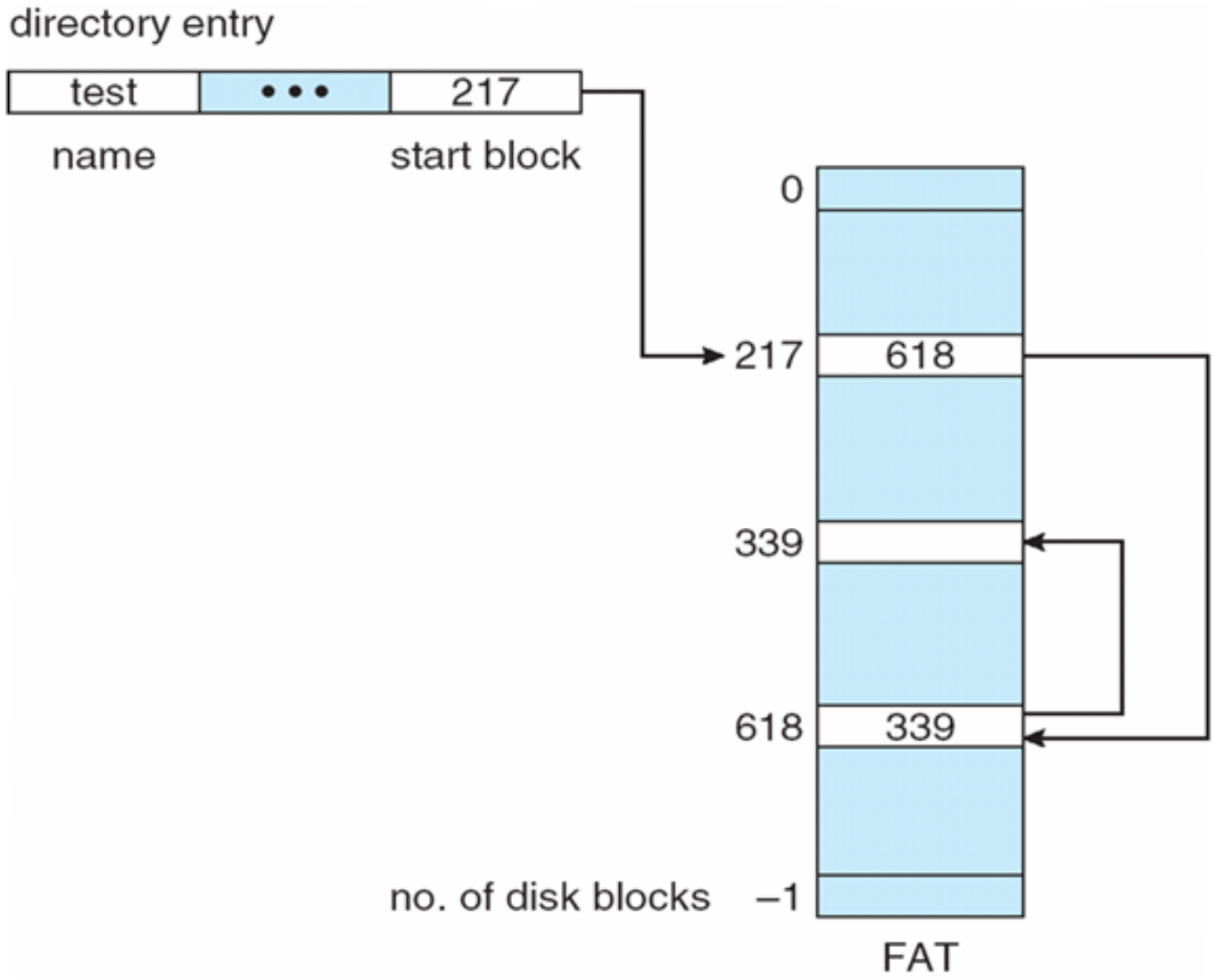
Linked allocation – each file a linked list of blocks

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

Linked Allocation

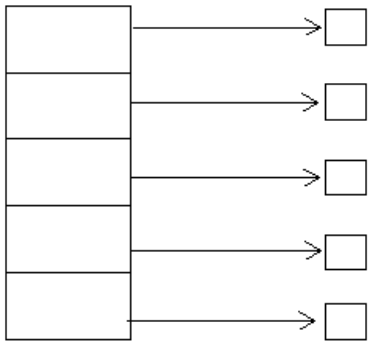


Linked Allocation



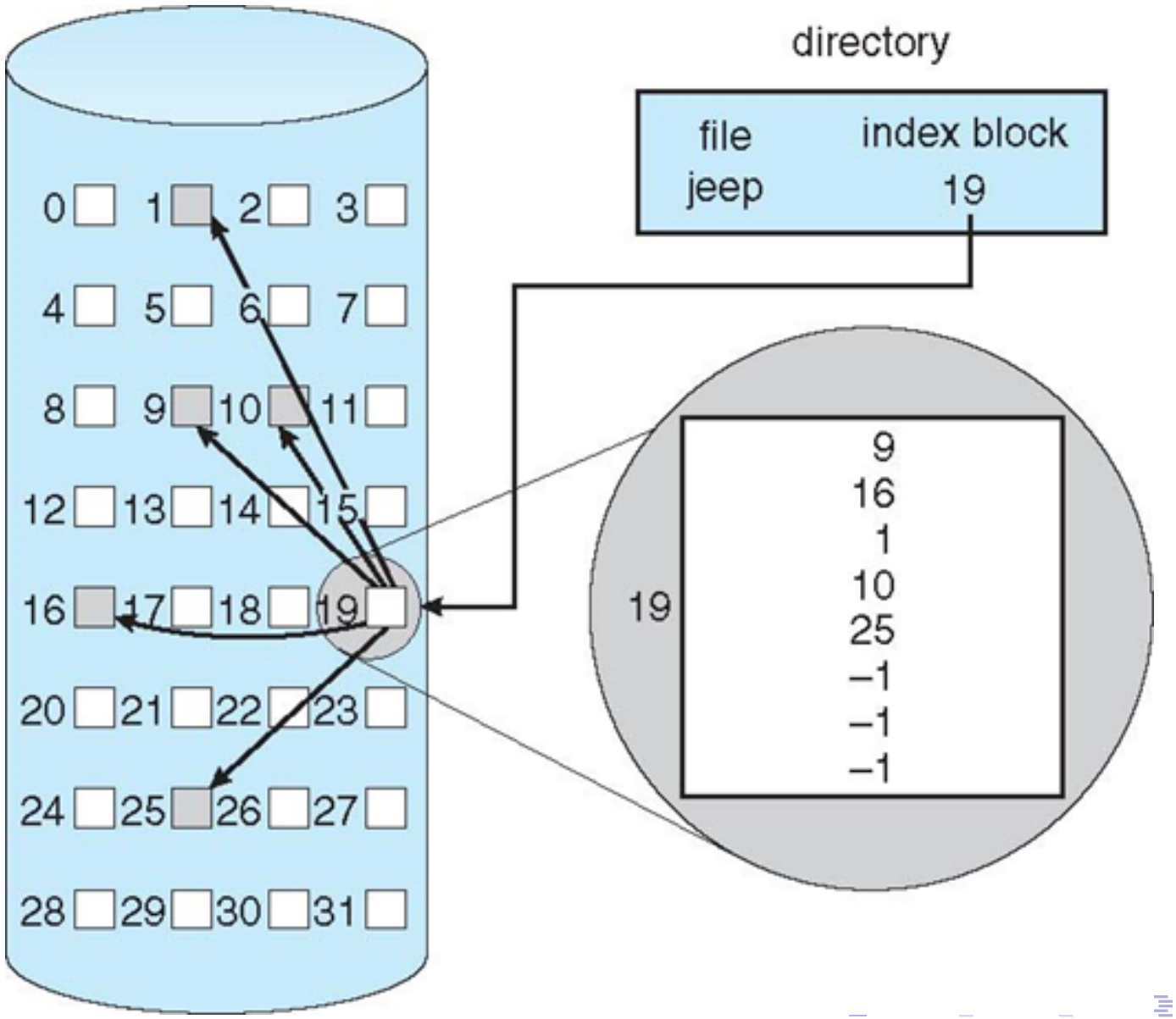
Indexed Allocation

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



index table

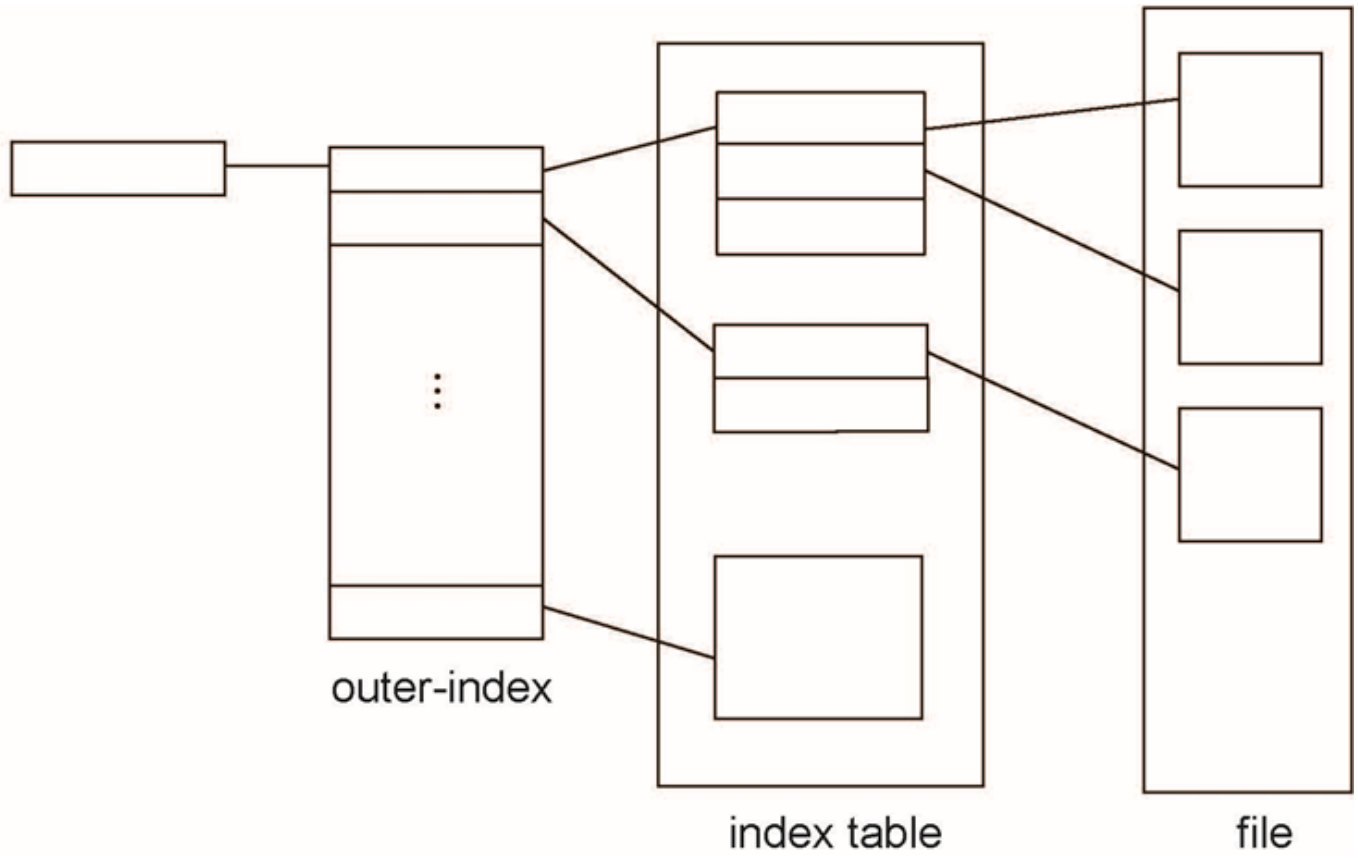
Indexed Allocation



Indexed Allocation

- 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
- Multilevel index

Indexed Allocation



Summary

Discussed

- Implementation of local file systems and directory structures
- Implementation of remote file systems
- Block allocation and free-block algorithms and trade-offs

Test Your Understanding

- File system resides on ———
- ———————translates logical block to physical block
- File Control Block (FCB) contains many details about ————-
- ———— separates file system generic operations from implementation details