Universitas Esa Unggul

Smart, Creative and Entrepreneurial

CCJ-123-DASAR PENGEMBANGAN PERANGKAT LUNAK (PERTEMUAN-13)

www.esaunggul.ac.id

Dosen Pengampu :

5165-Kundang K Juman,

Prodi Teknik Informatika Fakultas Ilmu Komputer

Operating Systems

File System Implementation

A. Frank - P. Weisberg

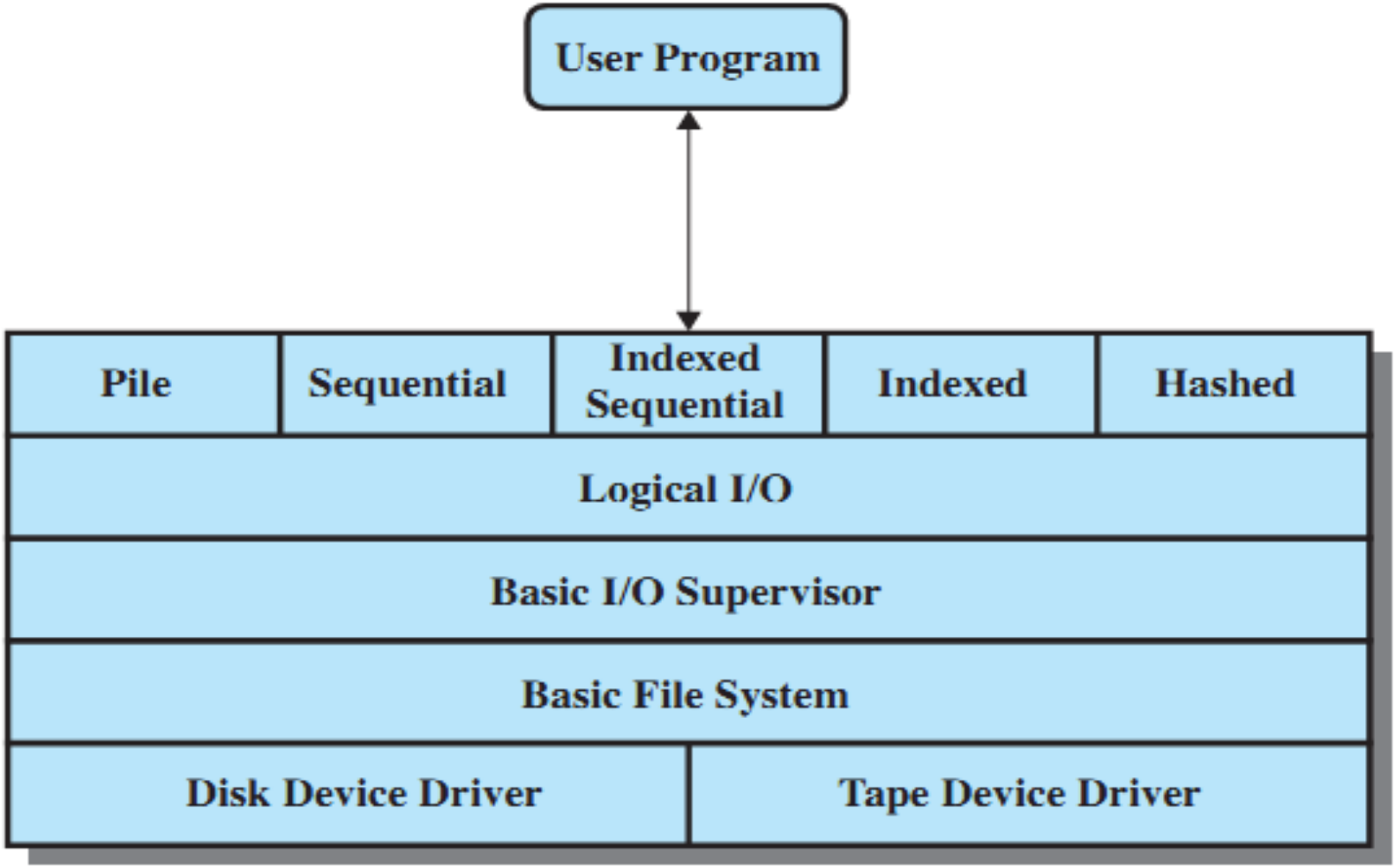
File System Implementation

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

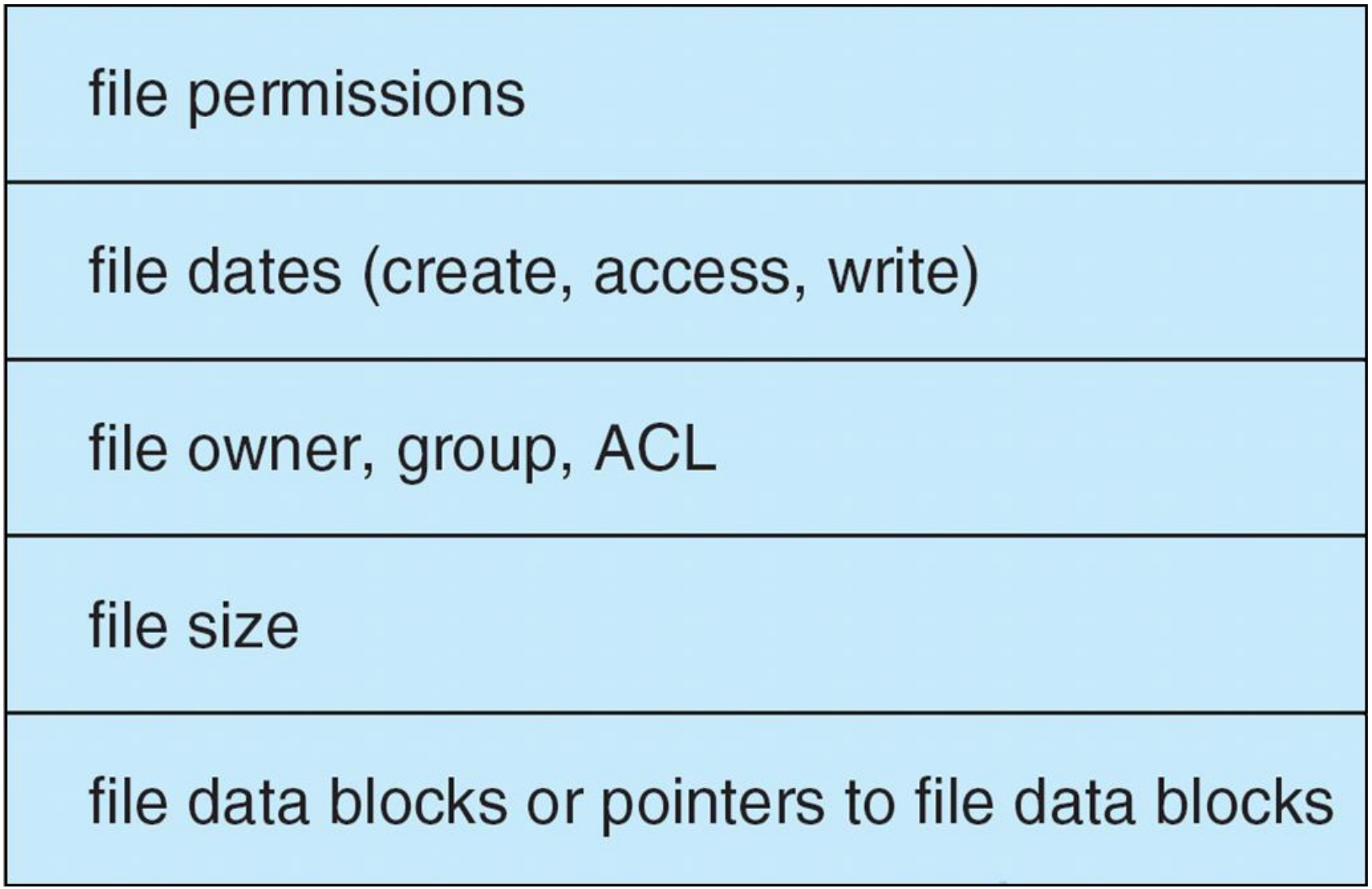
File-System Structure

- File structure:
  - Logical storage unit
  - Collection of related information
- 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 in-place 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.

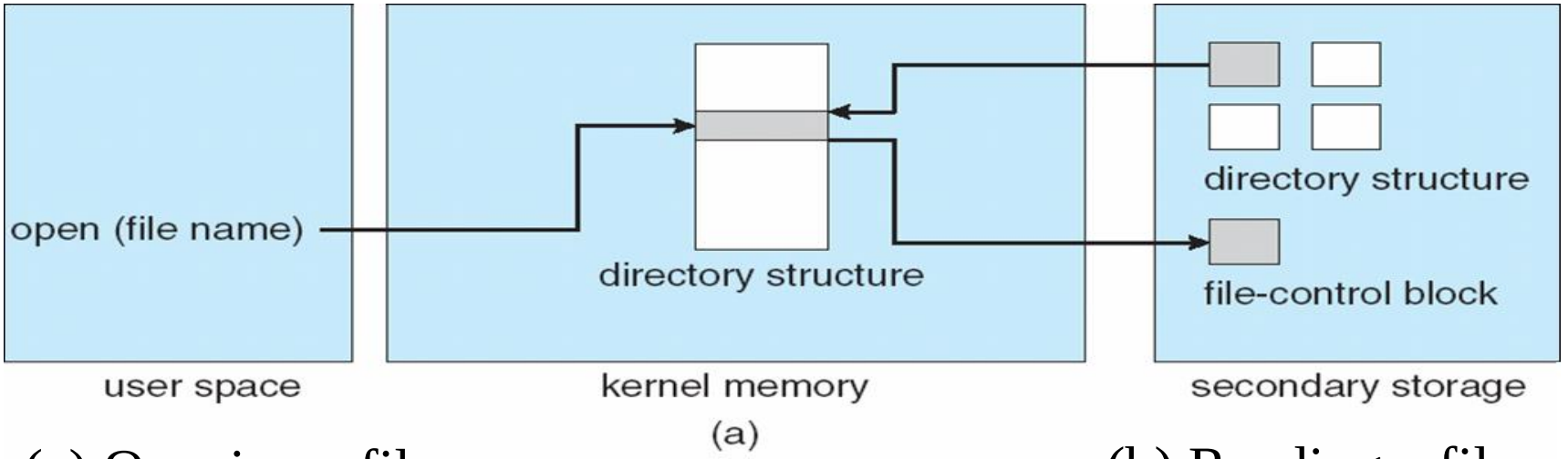
File System Software Architecture



A Typical File Control Block (FCB)

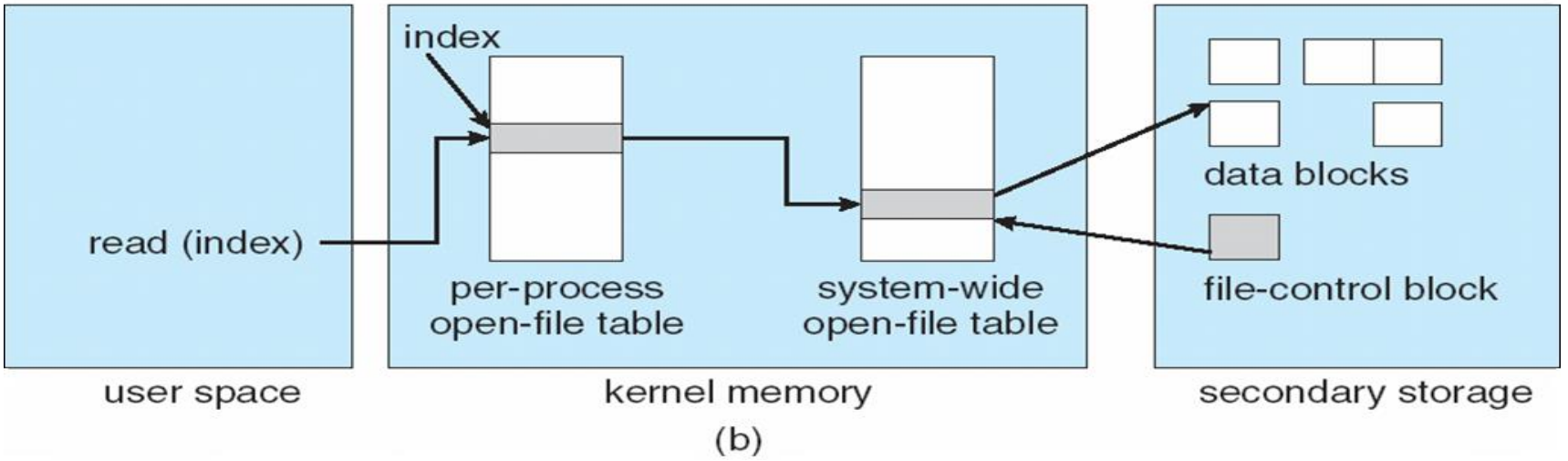


In-Memory File System Structures



(a) Opening a file

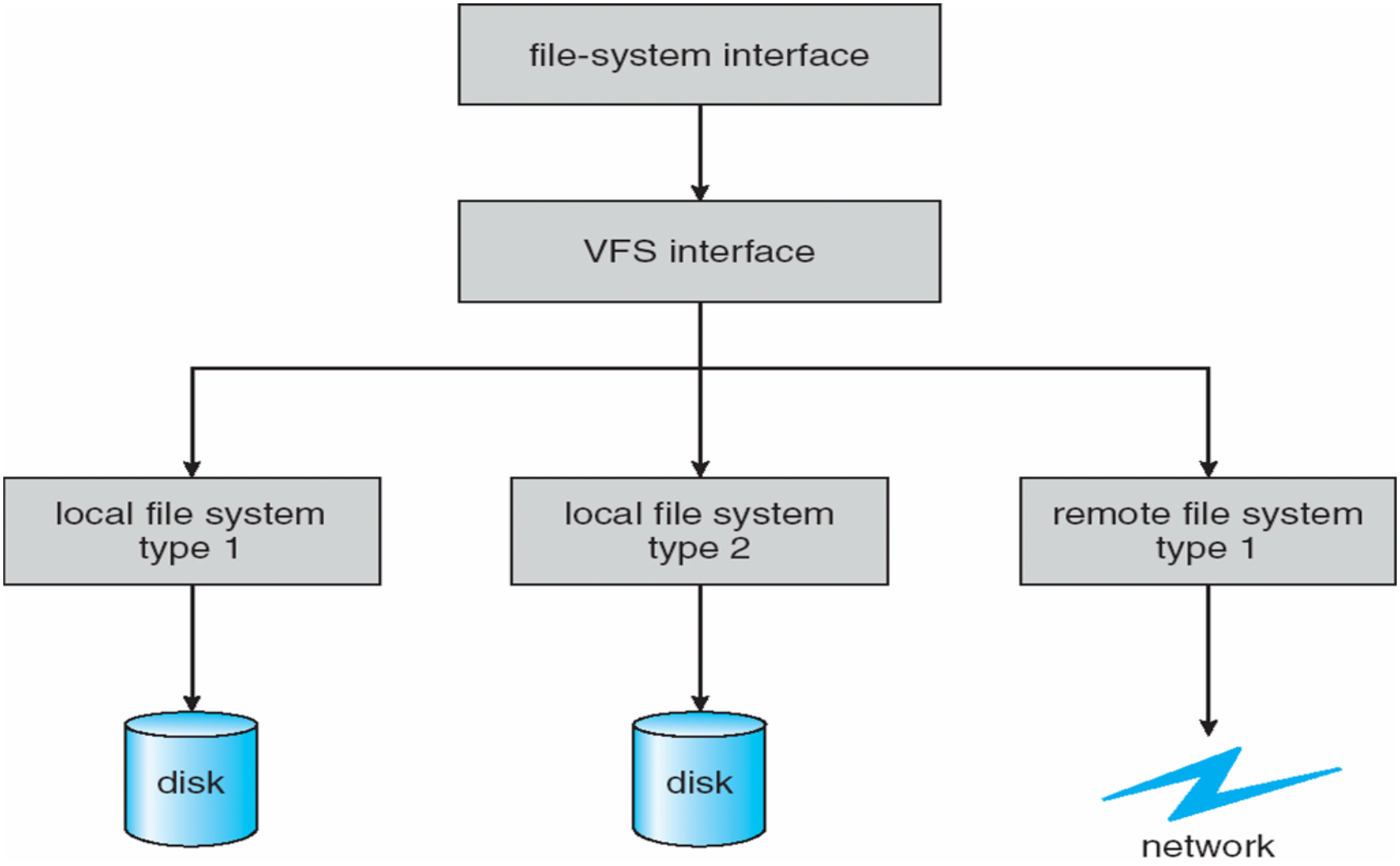
(b) Reading a file



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.

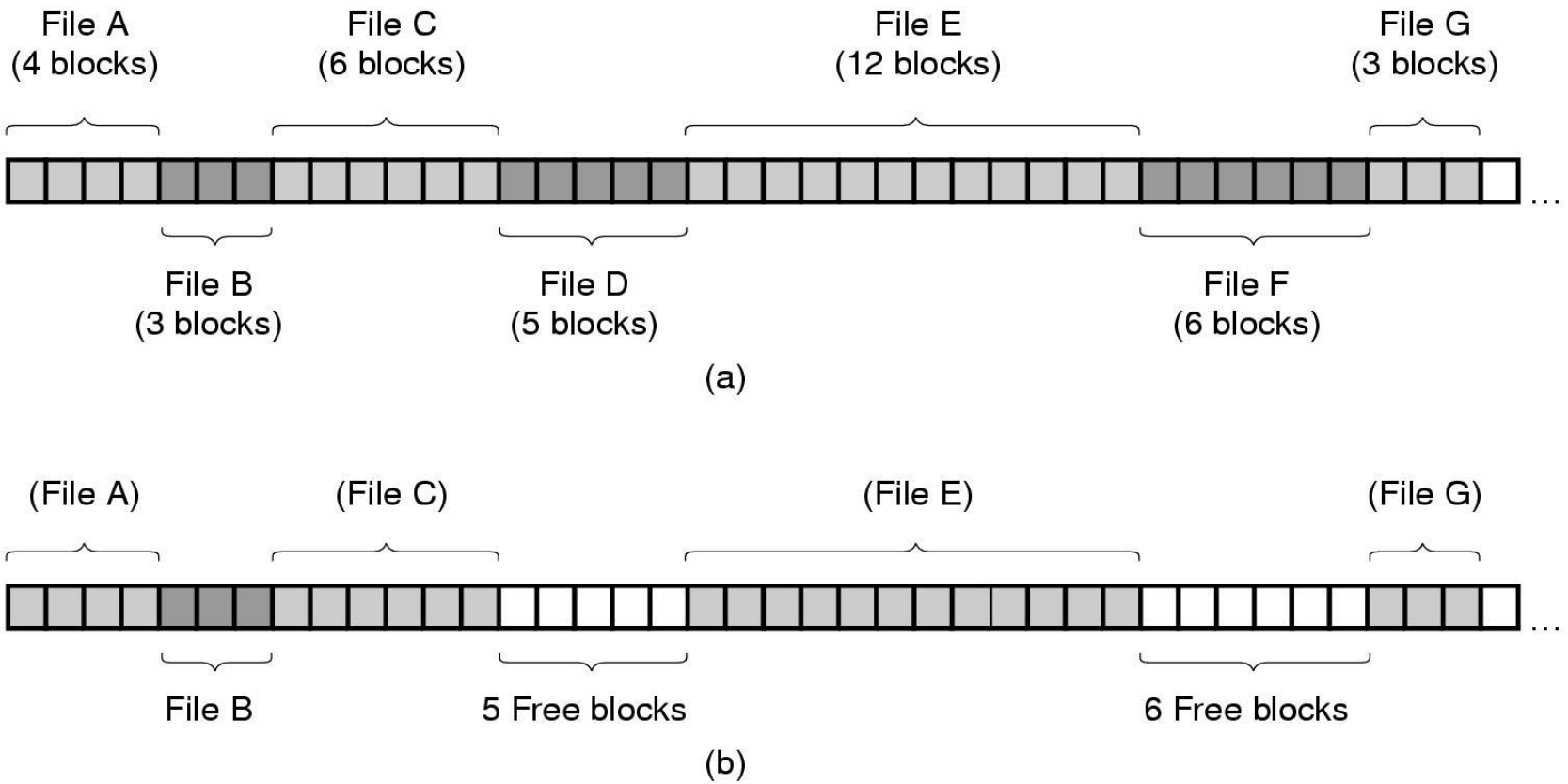
Schematic View of Virtual File System



Allocation Methods

- An allocation method refers to how disk blocks are allocated for files:
  1. Contiguous allocation
  2. Chained/Linked allocation
  3. Indexed allocation
  4. Combined schemes

Contiguous Allocation Example

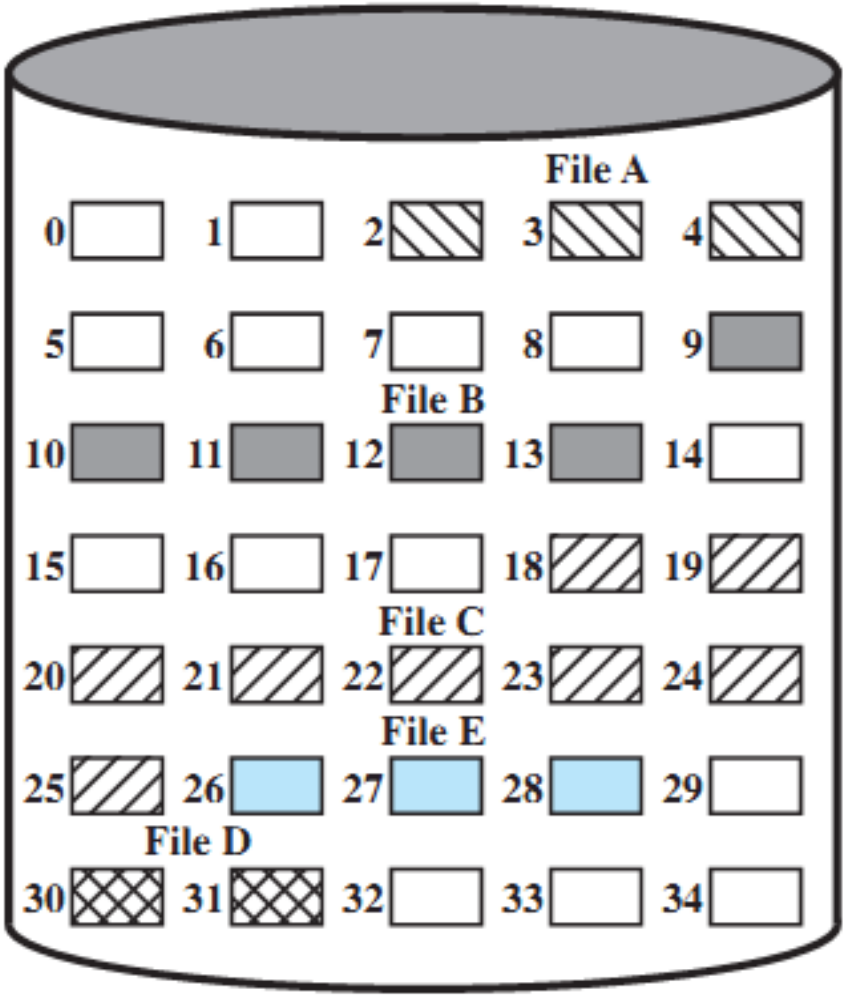


(a) Contiguous allocation of disk space for 7 files.
(b) The state of the disk after files D and F have been removed.

Contiguous Allocation

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

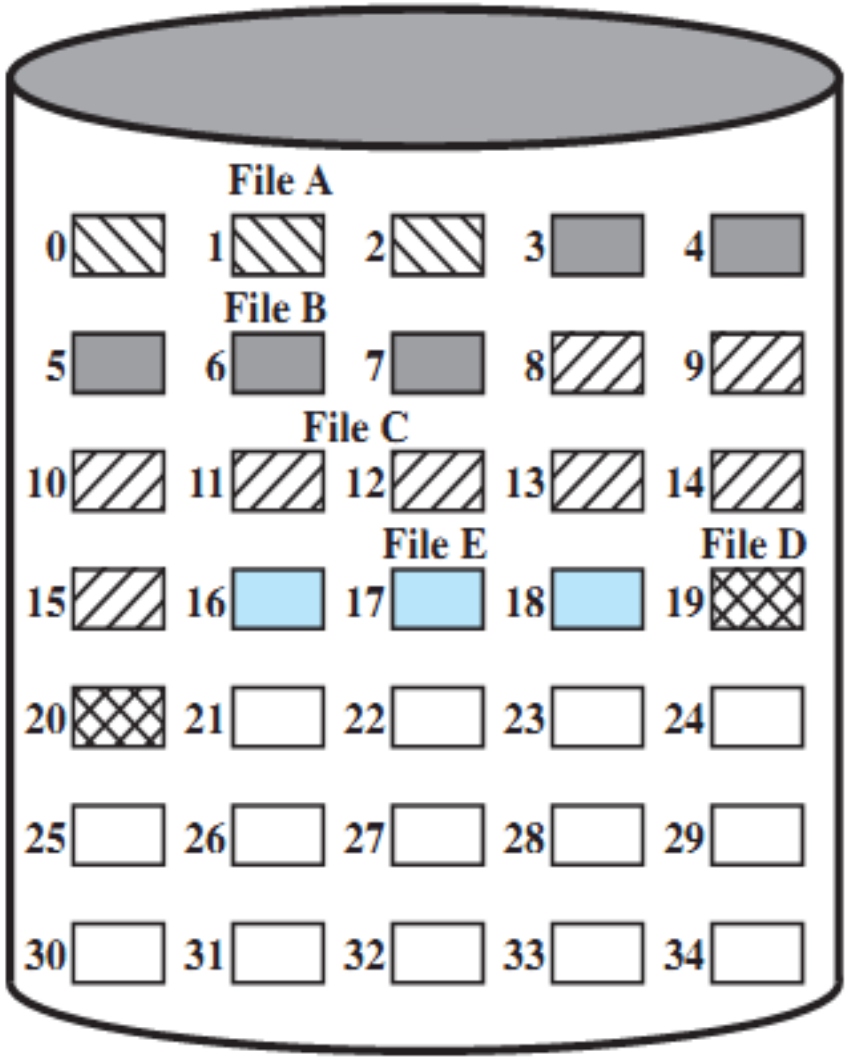
Contiguous File Allocation Example



File Allocation Table

File Name	Start Block	Length
File A	2	3
File B	9	5
File C	18	8
File D	30	2
File E	26	3

Contiguous file allocation (after compaction)



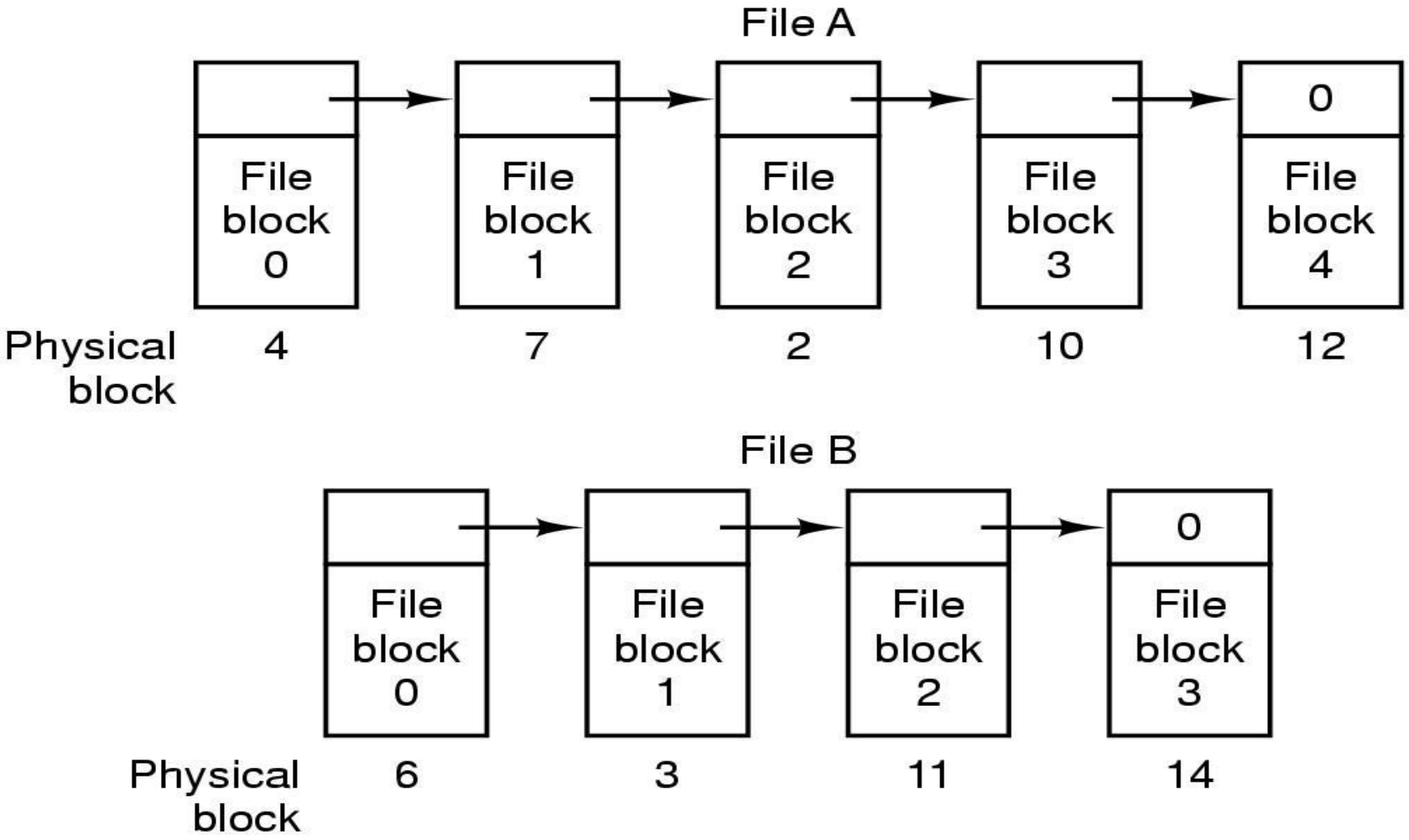
File Allocation Table

File Name	Start Block	Length
File A	0	3
File B	3	5
File C	8	8
File D	19	2
File E	16	3

Extent-Based Systems

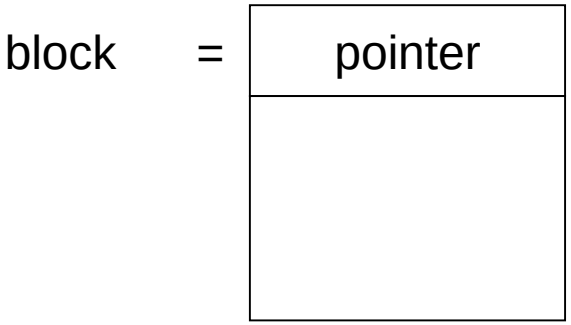
- Many newer file systems (i.e., Veritas File System) use a modified contiguous allocation scheme.
- Extent-based file systems allocate disk blocks in extents.
- An extent is a contiguous block of disks:
  - Extents are allocated for file allocation.
  - A file consists of one or more extents.

Chained/Linked Allocation Example



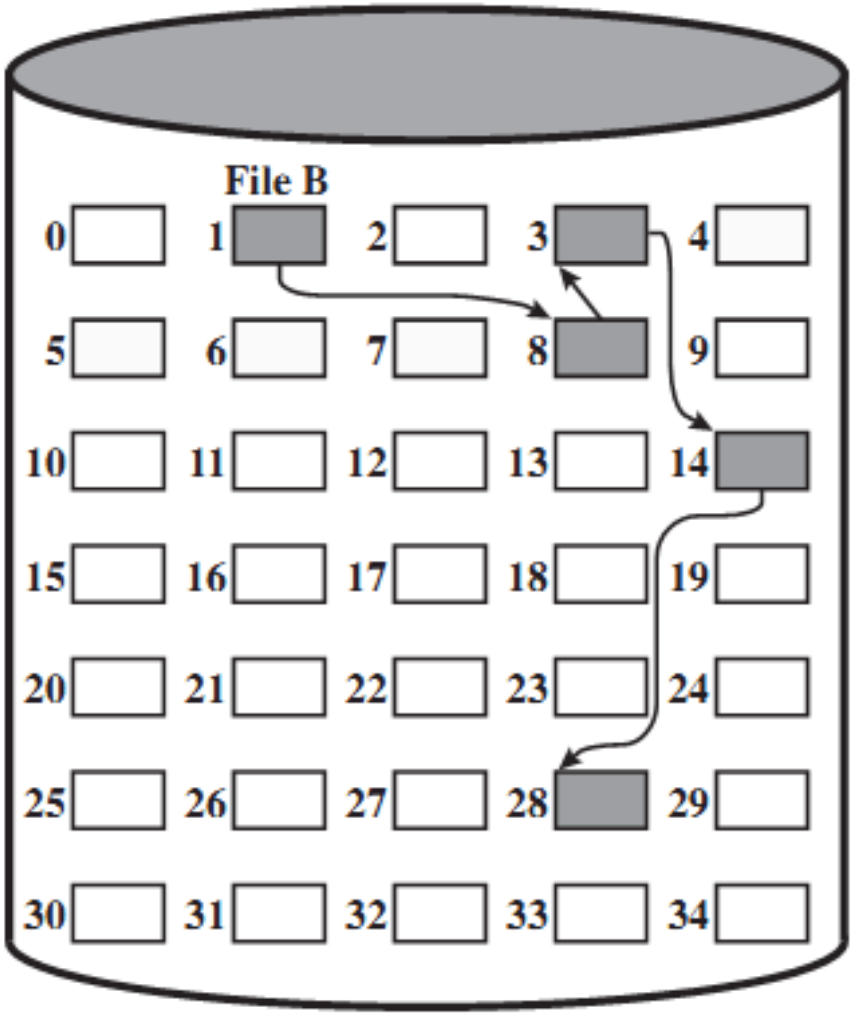
Chained Allocation

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



- Simple: need only starting address.
- Free-space management: no waste of space.
- No random access.

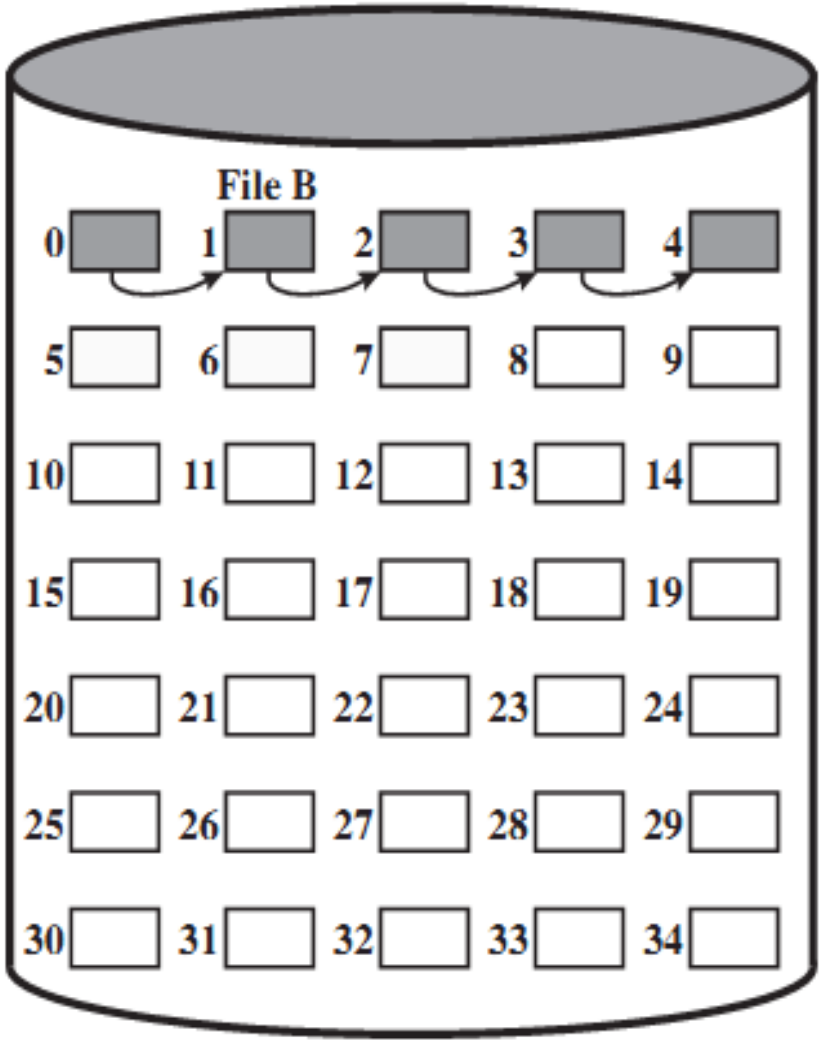
Chained File Allocation Example



File Allocation Table

File Name	Start Block	Length
• • •	• • •	• • •
File B	1	5
• • •	• • •	• • •

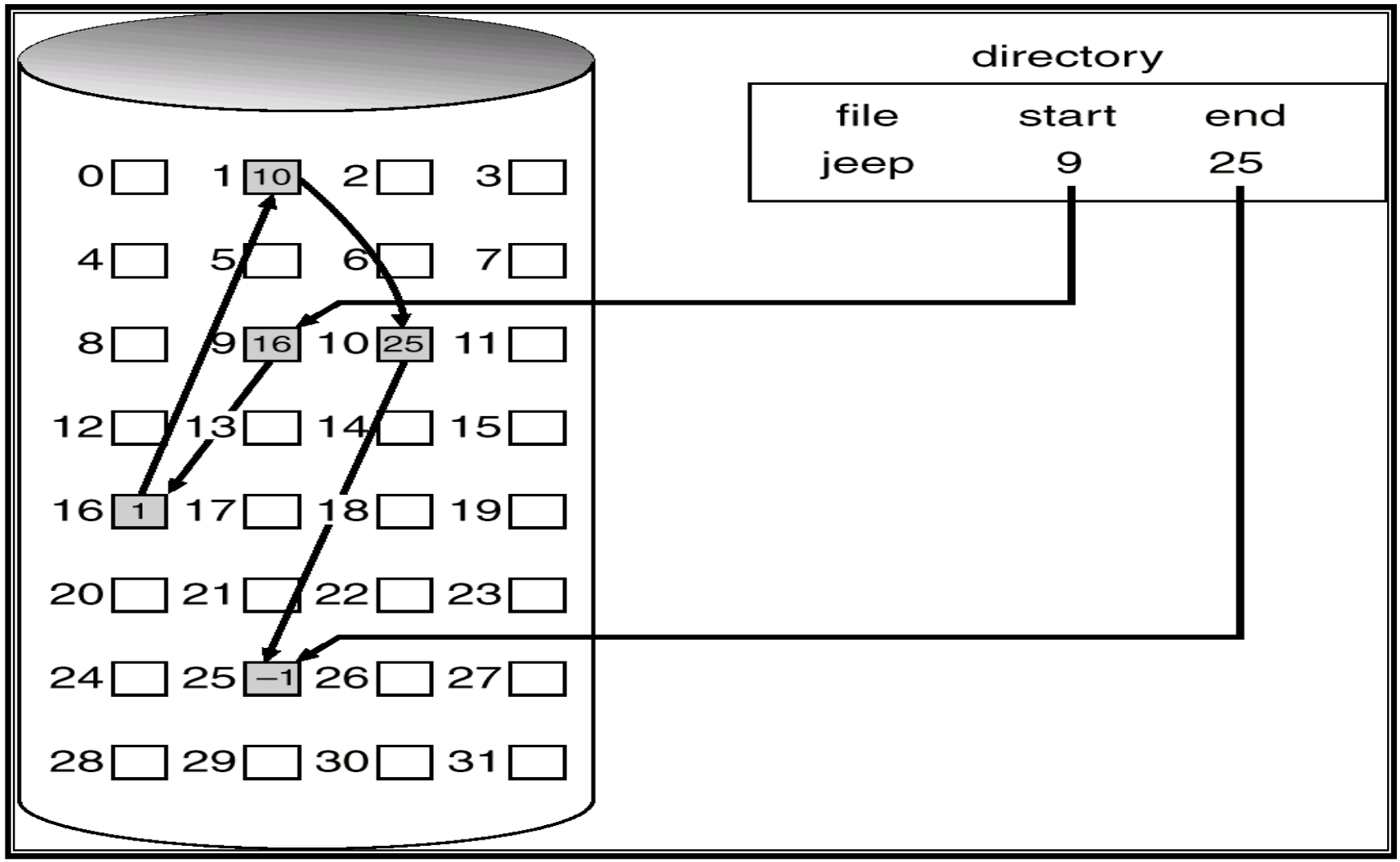
Chained File Allocation (after consolidation)



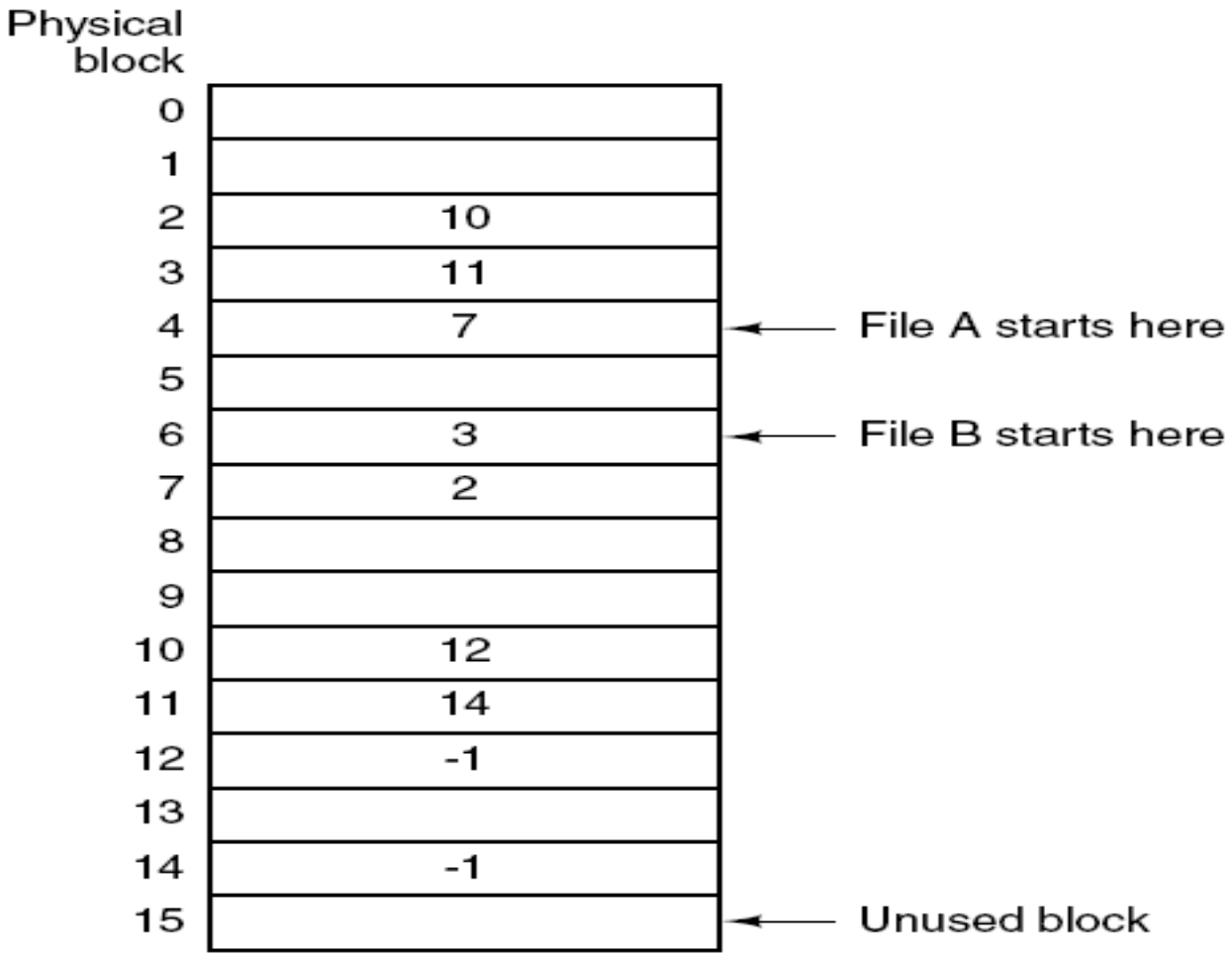
File Allocation Table

File Name	Start Block	Length
• • •	• • •	• • •
File B	0	5
• • •	• • •	• • •

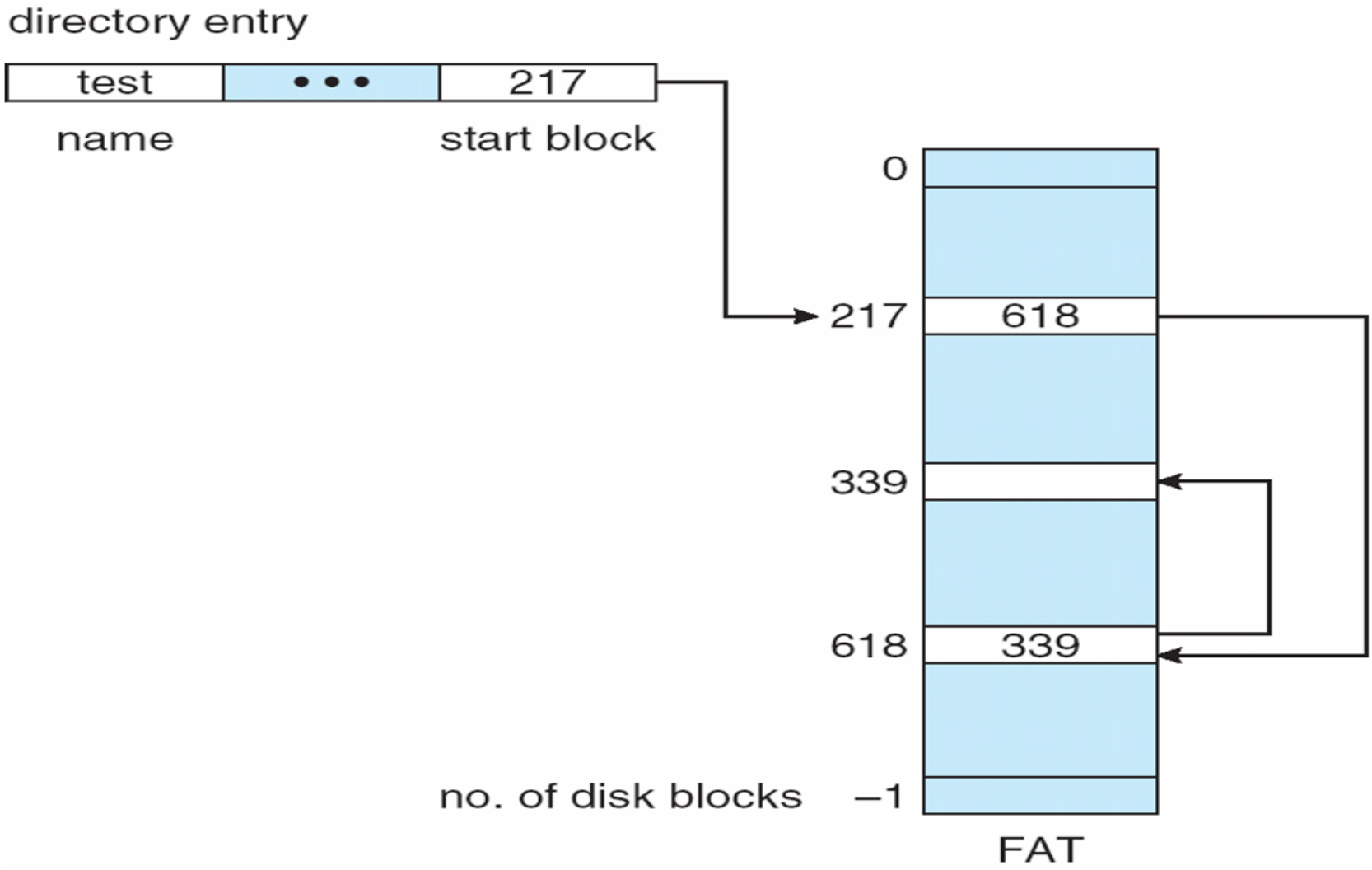
Another Chained Allocation Example



Linked List Allocation Using a Table in Memory

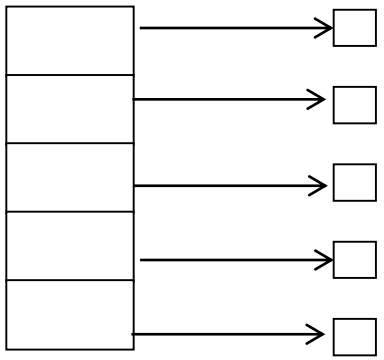


File-Allocation Table (FAT) Example

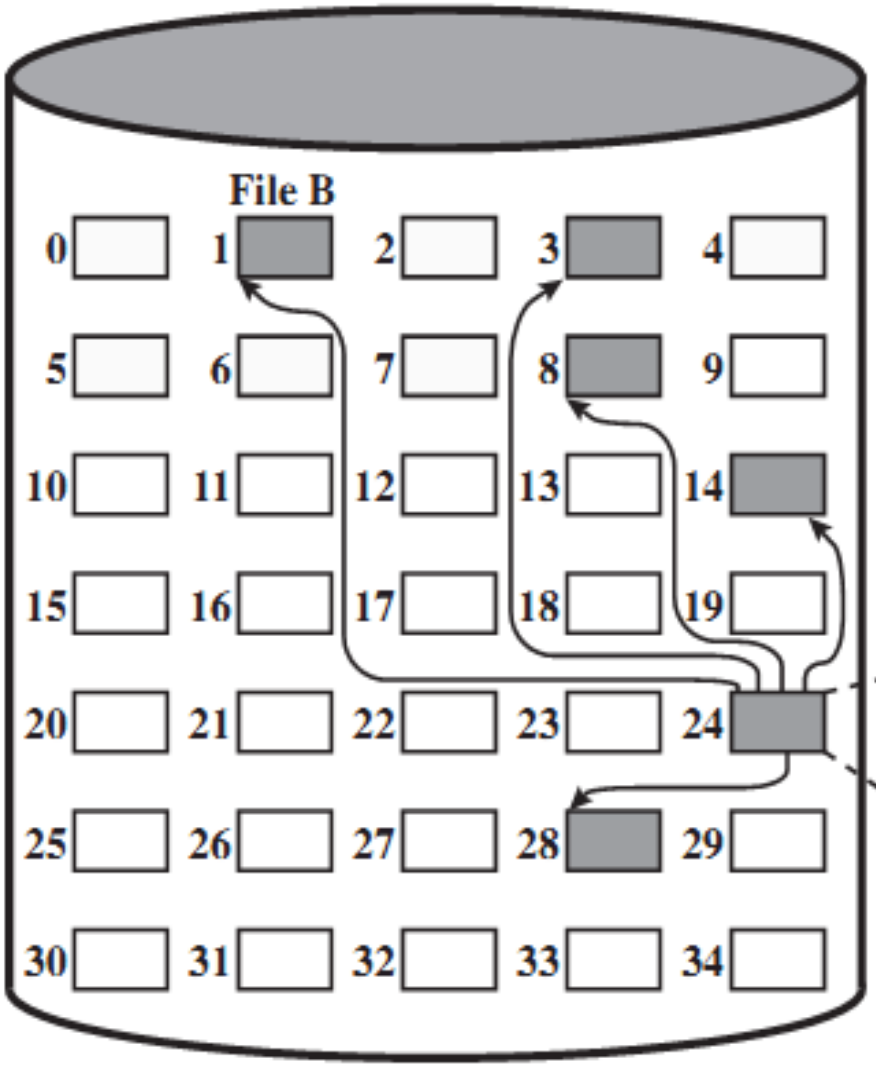


Indexed Allocation

- Brings all pointers together into the *index block.*
- It’s a logical view.
- Need index table.
- Provides random access.
- Dynamic access without external fragmentation, but have overhead of index block.

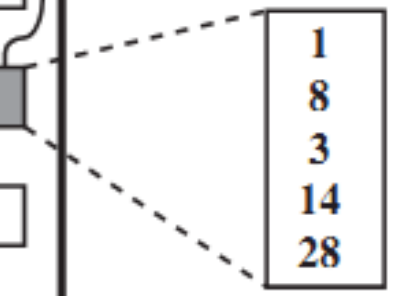


Indexed File Allocation Example

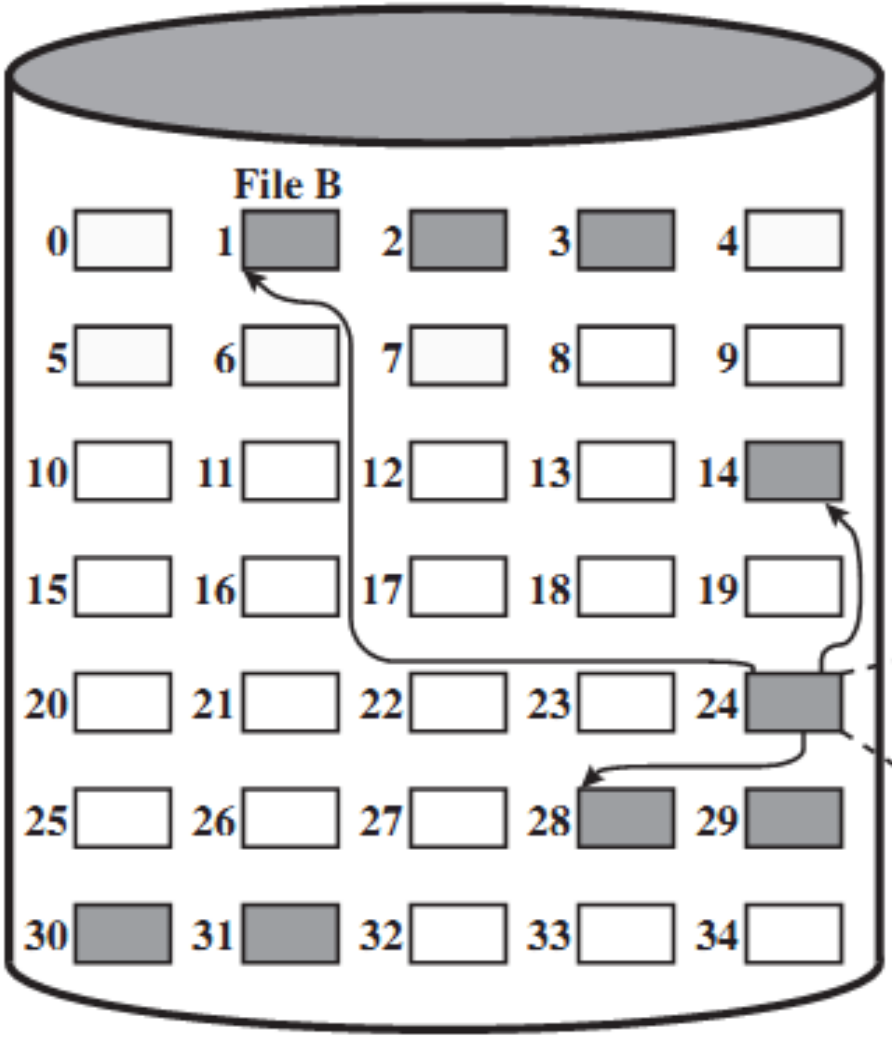


File Allocation Table

File Name	Index Block
• • •	• • •
File B	24
• • •	• • •



Indexed File Allocation (variable-size)

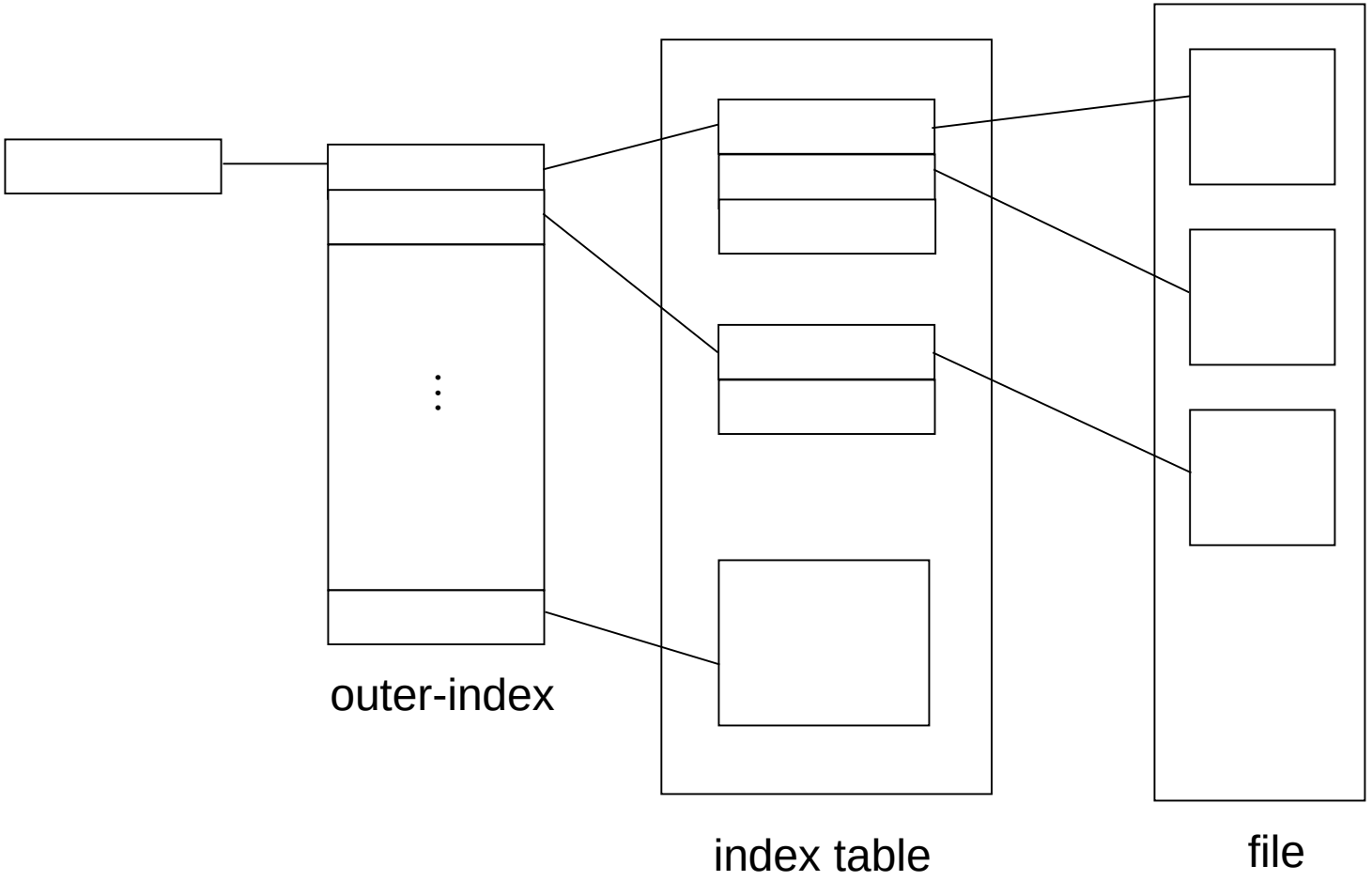


File Allocation Table

File Name	Index Block
• • •	• • •
File B	24
• • •	• • •

Start Block	Length
1	3
28	4
14	1

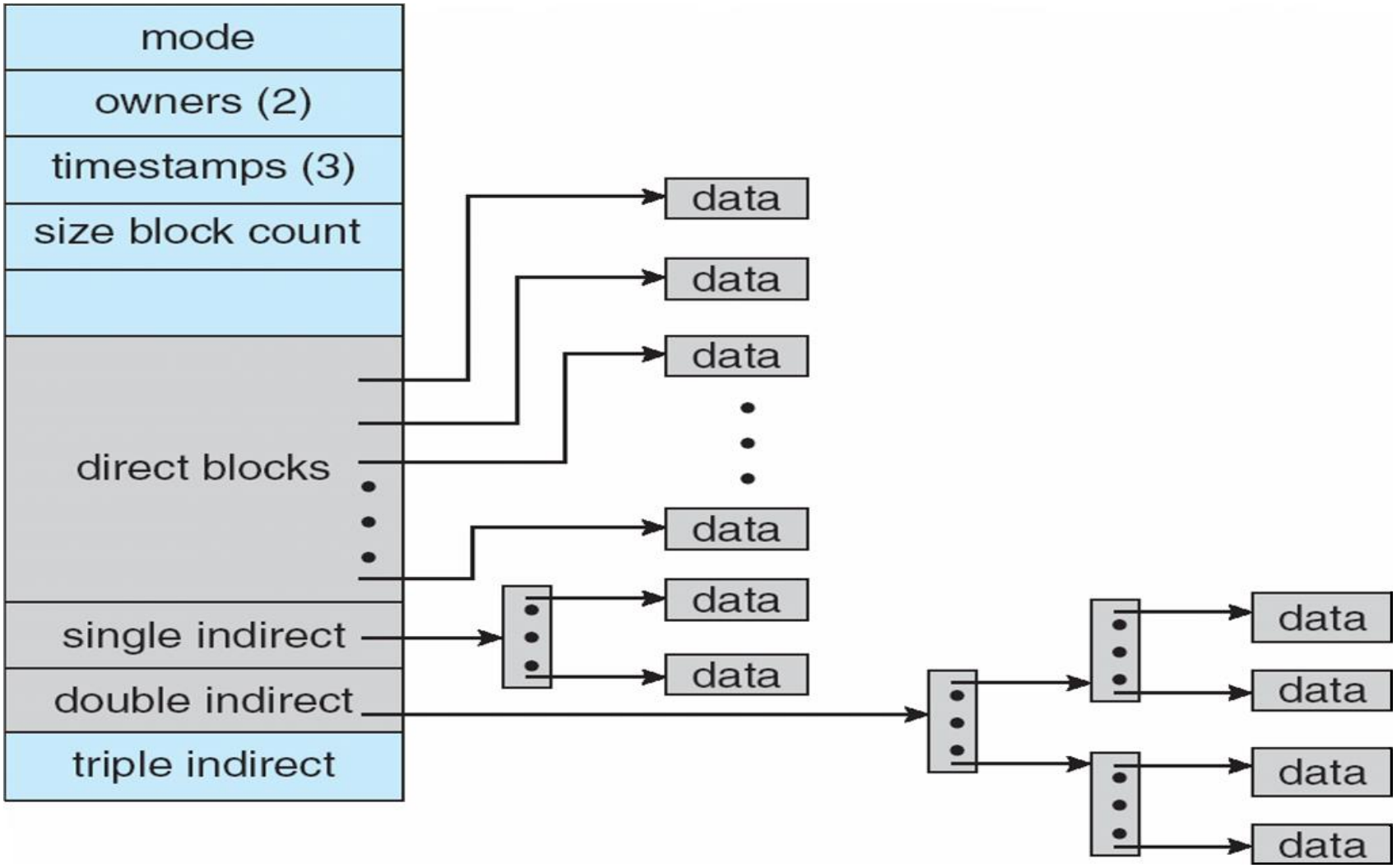
Indexed Allocation Mapping Example



Comparison of file allocation methods

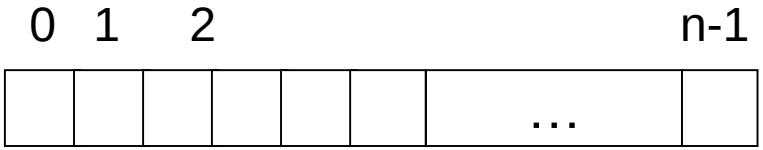
	Contiguous	Chained	Indexed	
Preallocation?	Necessary	Possible	Possible	
Fixed or variable size portions?	Variable	Fixed blocks	Fixed blocks	Variable
Portion size	Large	Small	Small	Medium
Allocation frequency	Once	Low to high	High	Low
Time to allocate	Medium	Long	Short	Medium
File allocation table size	One entry	One entry	Large	Medium

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



Free-Space Management (1)

- Bit vector (*n* blocks)



bit[*i*] = { 0 ⇒ block[*i*] free
1 ⇒ block[*i*] occupied

Block number calculation

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

Free-Space Management (2)

- Bit map requires extra space. For example:

  block size = 2^{12} bytes

  disk size = 2^{30} bytes (1 gigabyte)

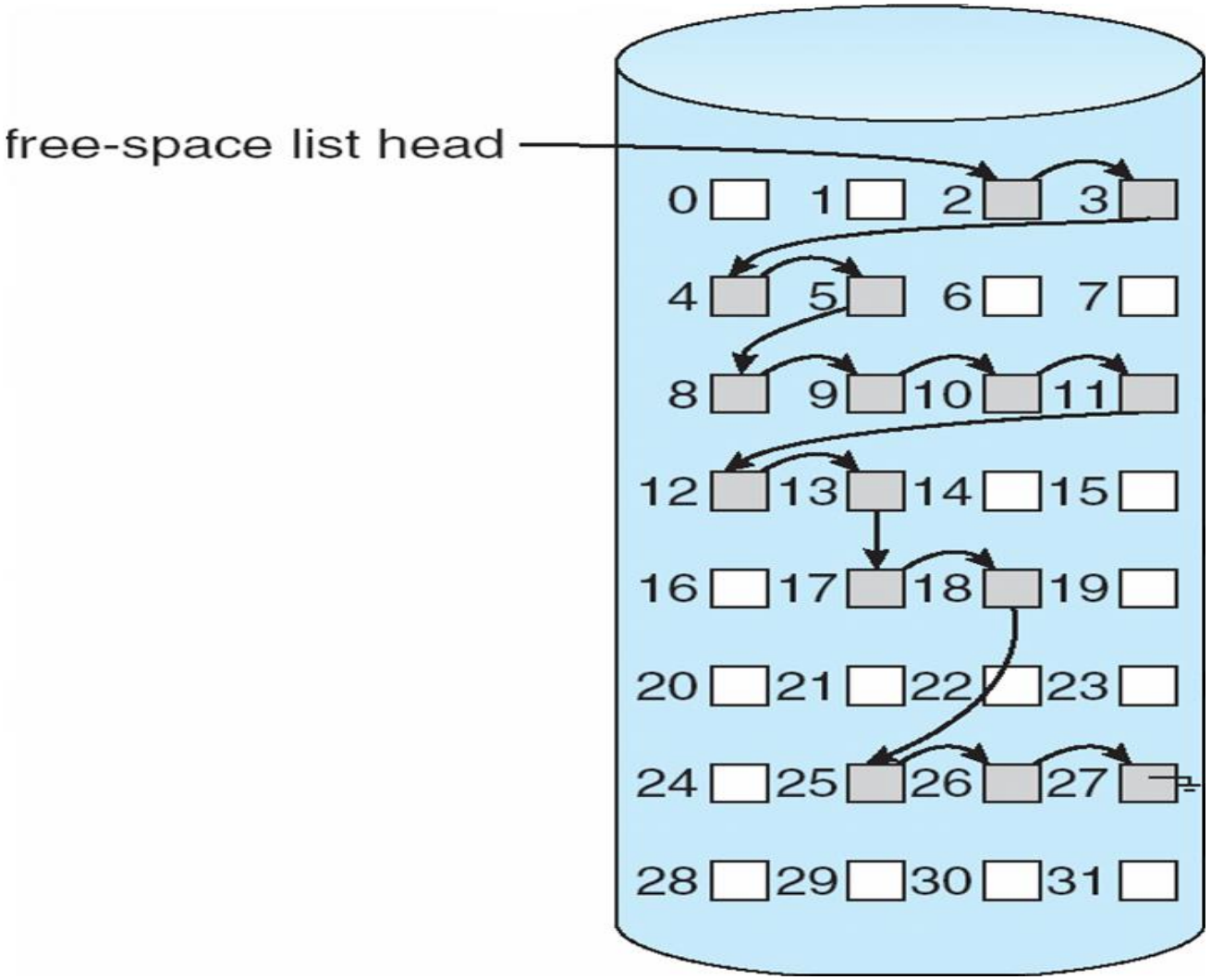
  $n = 2^{30}/2^{12} = 2^{18}$ bits (or 32K bytes)

- Easy to get contiguous files.
- Linked list (free list):
  - Cannot get contiguous space easily.
  - No waste of space.
- Grouping
- Counting

Free-Space Management (3)

- Need to protect:
  - Pointer to free list.
  - Bit map:
    - Must be kept on disk.
    - Copy in memory and disk may differ.
    - Cannot allow for block[*i*] to have a situation where bit[*i*] = 1 in memory and bit[*i*] = 0 on disk.
  - Solution:
    - Set bit[*i*] = 1 in disk.
    - Allocate block[*i*].
    - Set bit[*i*] = 1 in memory.

Linked Free Space List on Disk



Directory Implementation

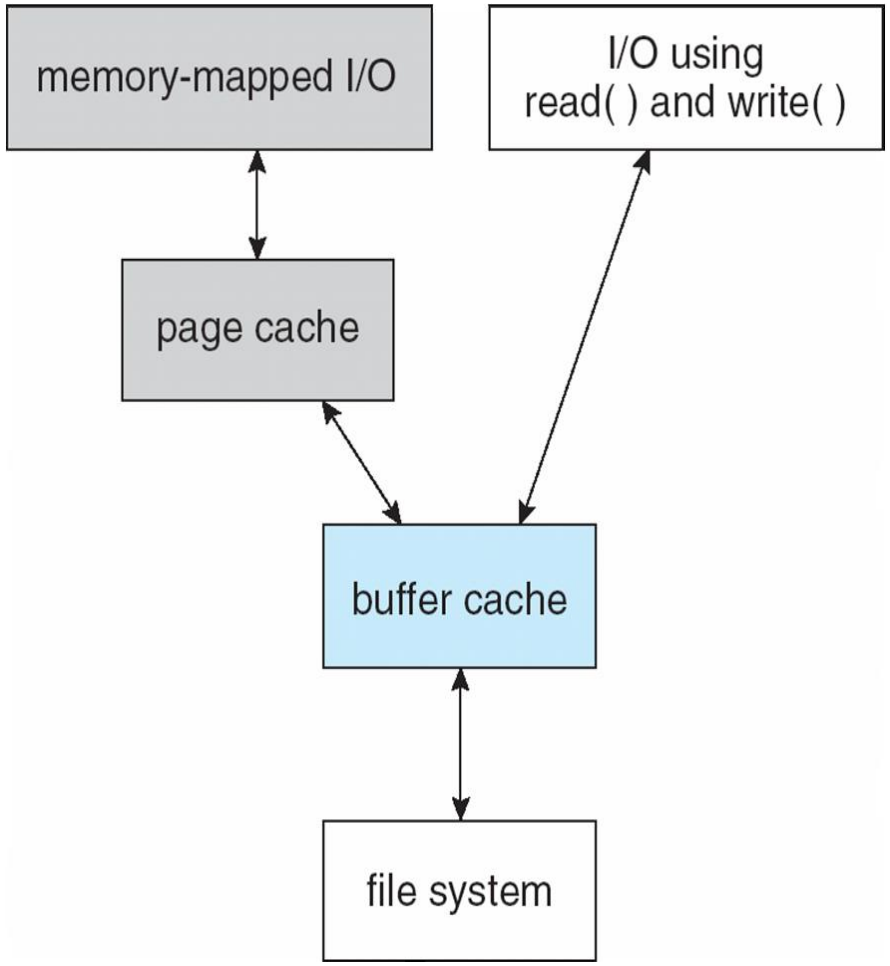
- Linear list of file names with pointer to the data blocks:
  - simple to program
  - time-consuming to execute
- Hash Table – linear list with hash data structure:
  - decreases directory search time
  - *collisions* – situations where two file names hash to the same location
  - fixed size

Efficiency and Performance

- Efficiency dependent on:
  - Disk allocation and directory algorithms.
  - Types of data kept in file's directory entry.
- Performance:
  - Disk cache: separate section of main memory for frequently used blocks.
  - Free-behind and read-ahead: techniques to optimize sequential access.
  - Improve PC performance by dedicating section of memory as virtual disk or RAM disk.

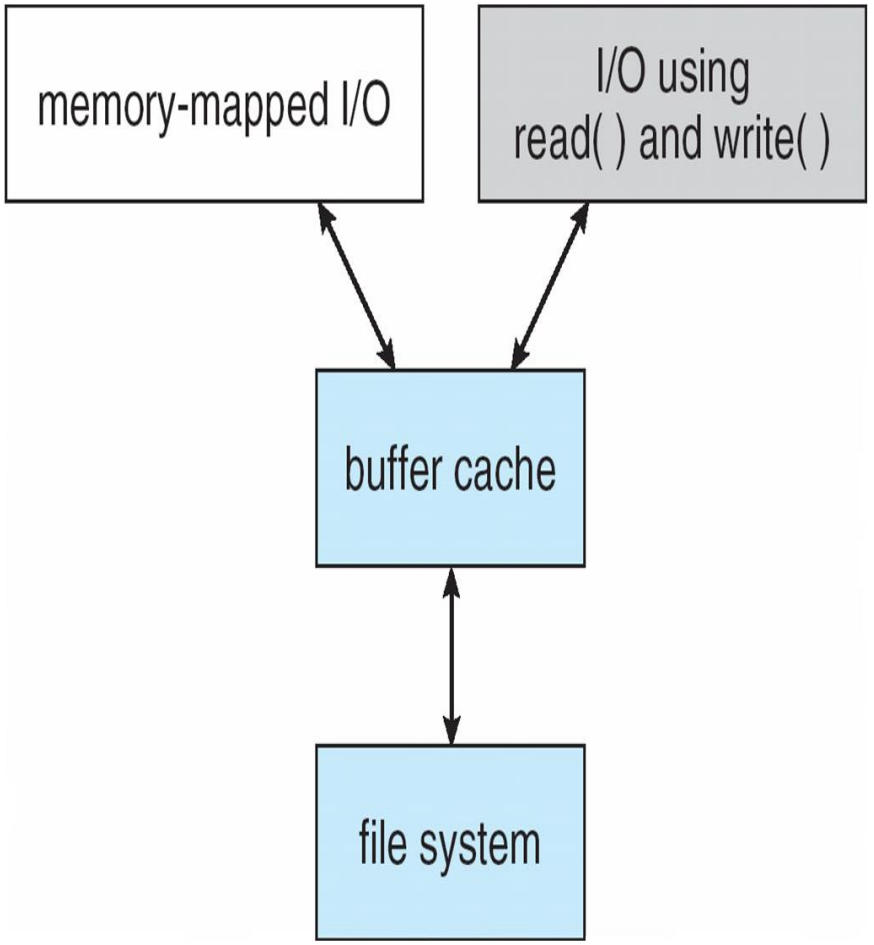
Page Cache

- A **page cache** caches pages rather than disk blocks using virtual memory techniques.
- Memory-mapped I/O uses a page cache.
- Routine I/O through the file system uses the buffer (disk) cache.



Unified Buffer Cache

- A unified buffer cache uses the same page cache to cache both memory-mapped pages and ordinary file system I/O.



Recovery

- Consistency checking – compares data in directory structure with data blocks on disk, and tries to fix inconsistencies.
- Use system programs to back up data from disk to another storage device (floppy disk, magnetic tape).
- Recover lost file or disk by restoring data from backup.

Storage Backup