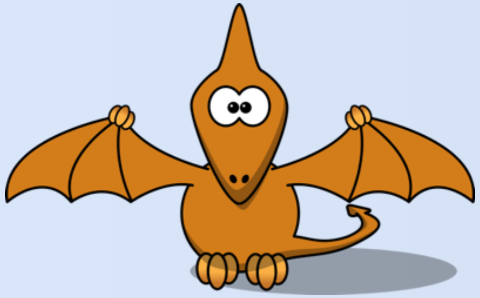
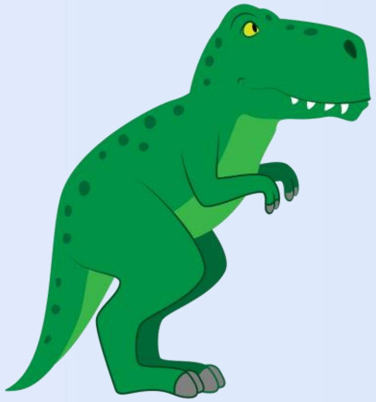


Chapter 11-15.

Storage Management



Operating System Concepts (10th Ed.)

Ch.11 Mass-Storage Structure

- Mass-Storage
  - non-volatile, secondary storage system of a computer
  - usually, HDD(Hard Disk Drive) or NVM(Non-Volatile Memory).
  - sometimes, magnetic tapes, optical disks, or cloud storage.
    - using the structure of RAID systems.

- Hard Disk Drives:

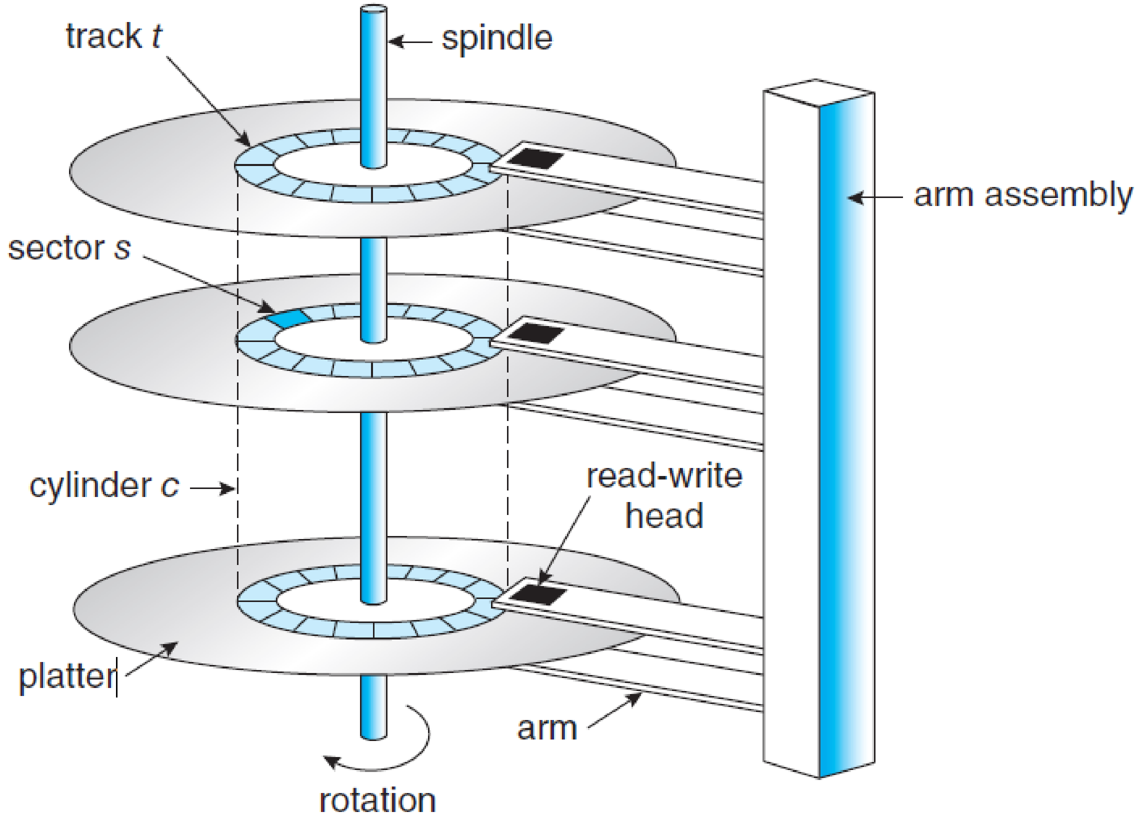


Figure 11.1 *HDD moving-head disk mechanism.*

- HDD Scheduling:
  - to *minimize* the access time (or *seek time*)
  - to *maximize* data transfer *bandwidth*.

  - ***seek time***:
    - the time for the device arm to move the heads to the cylinder containing the desired sector,
    - and the rotational latency is the additional time for the platter to rotate the desired sector to the head
  - ***disk bandwidth***:
    - the total number of bytes transferred, divided by the total time
    - between the first request and the completion of the last transfer.

- FIFO Scheduling:
  - intrinsically fair, but generally does not provide the fastest services.
  - total *head movement* of 640 *cylinders*.

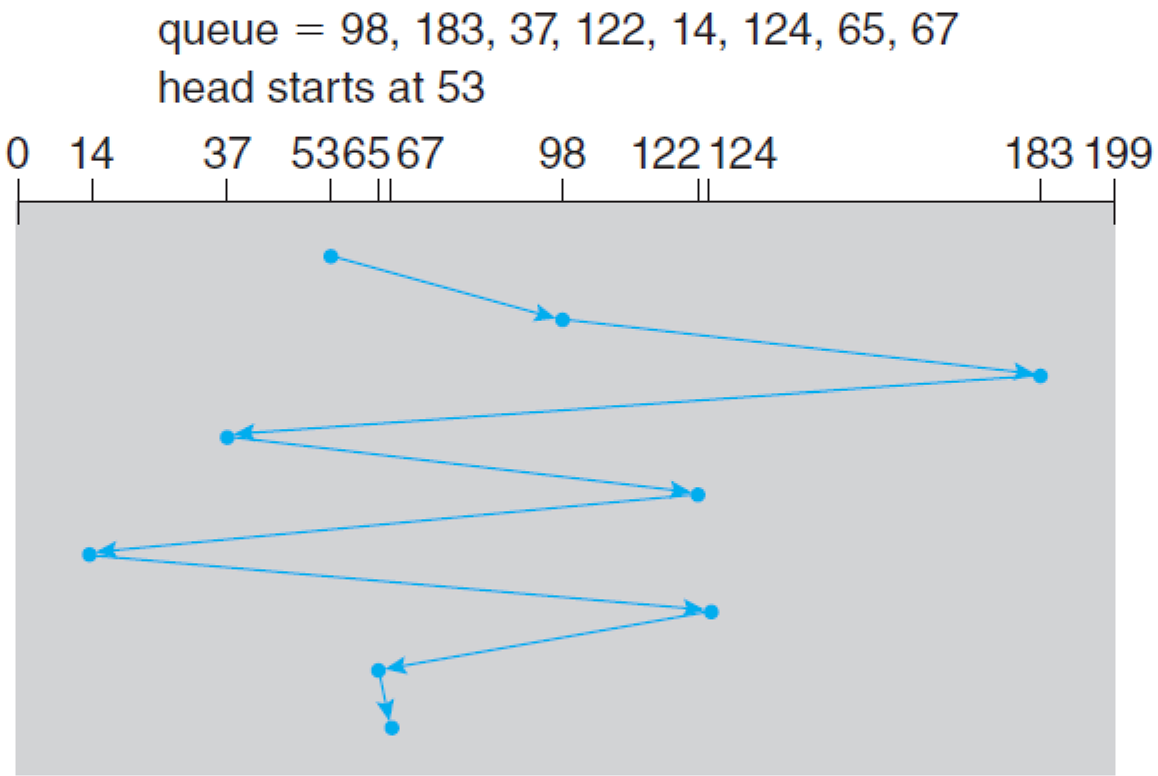


Figure 11.6 *FCFS disk scheduling.*

- SCAN Scheduling:
  - the disk arm *starts* *at one end* of the disk and moves *toward the other end*,
    - servicing requests as it reaches each cylinder,
    - until it gets to the other end of the disk.
  - At the other end,
    - the *direction* of head movement is *reversed*, and continue.

SCAN Scheduling:

- if the direction of head movement is moving towards 0.
- total *head movement* of 236 *cylinders*.

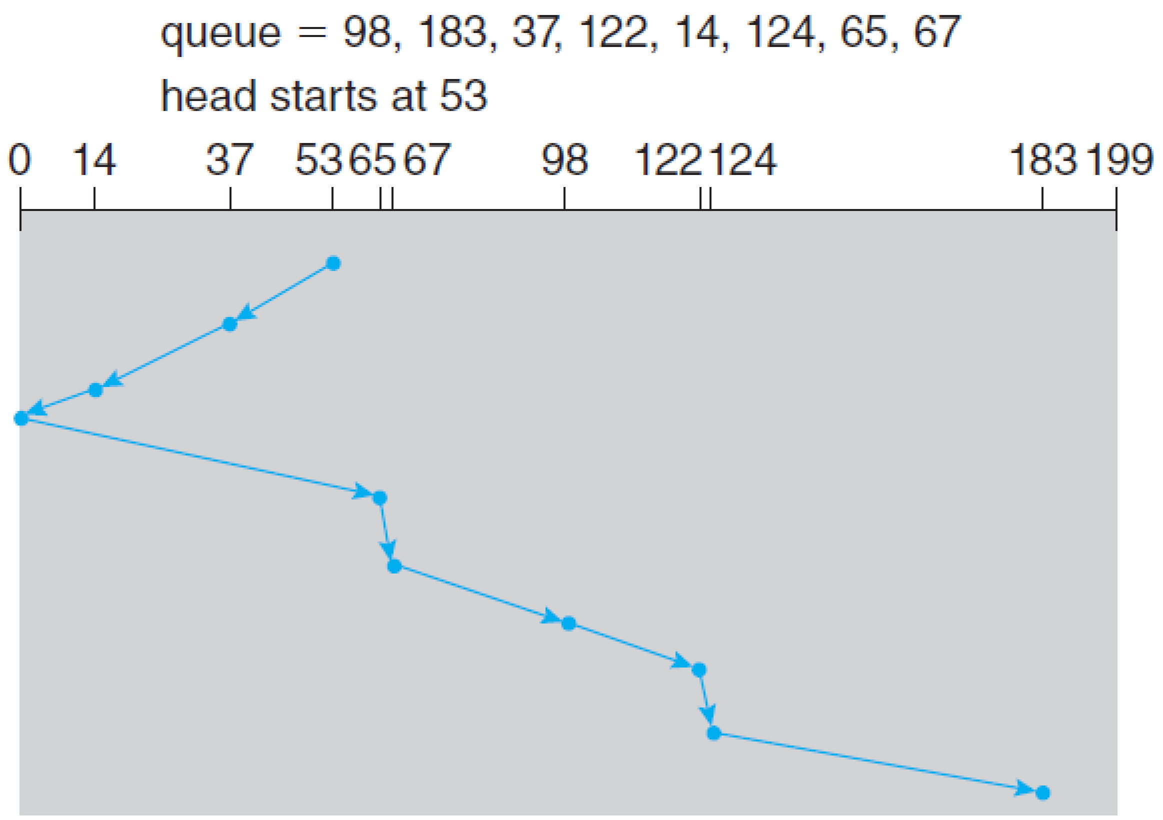


Figure 11.7 *SCAN disk scheduling.*

- C-SCAN (Circular-SCAN) Scheduling:
  - a variant of SCAN designed to provide a *more uniform wait time*.
  - moves the head from one end of the disk to the other,
    - servicing requests along the way.
  - however, when the head reaches the other end,
    - *returns immediately to* the *beginning* of the disk
    - *without servicing* any requests on the return trip.
  - treats the cylinders as a *circular list*
    - that wraps around from the final cylinder to the first one.

C-SCAN Scheduling:

- total *head movement* of *183 cylinders*. (ignore from 199 to 0)

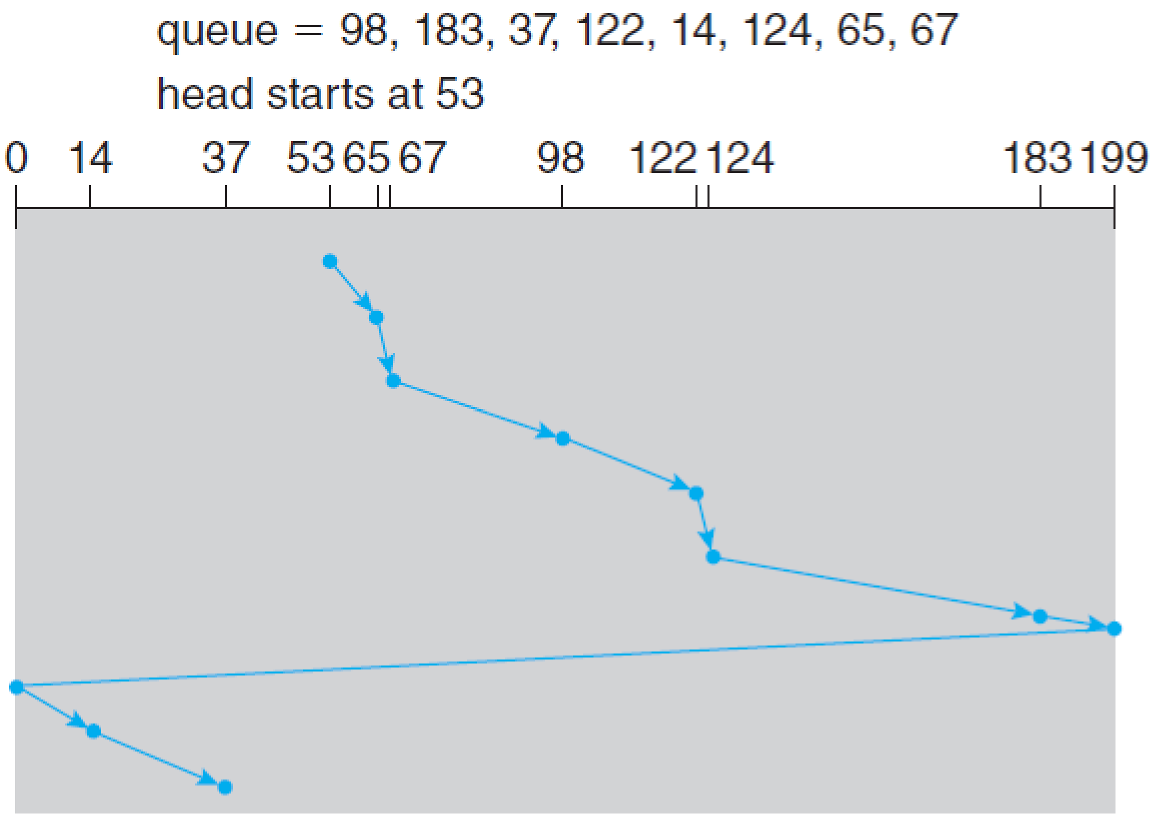


Figure 11.8 *C-SCAN disk scheduling.*

- Boot Block
  - For a computer to start running, when it is powered up,
    - it must have an initial program to run.
  - A bootstrap loader is stored in NVM flash memory,
    - and mapped to a known memory location.

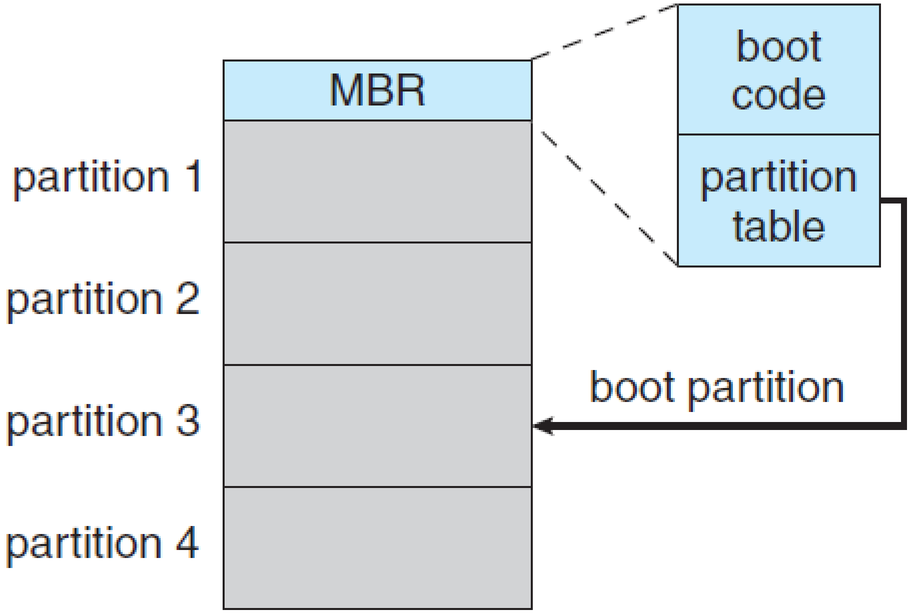


Figure 11.10 *Booting from a storage device in Windows.*

- RAID: Redundant Arrays of Independent Disks
  - a collection of a variety of disk-organization techniques.
  - to improve the *rate* at which data can be read or written.
    - if the drives are operated ***in parallel***.
  - to improve the ***reliability*** of data storage,
    - because *redundant information* can be stored on multiple devices.
  - thus, *failure* of *one drive* does *not lead to loss of data*.

Redundancy: Improvement of Reliability

- The *chance of fail* in a set of *N disks*
  - is much *greater* than the chance in a *single* disk.
- Suppose that *mean time between failures* (MTBF) = 100,000 hours.
  - MTBF in an array of 100 disks = 100,000 / 100 = 1,000 hours.
- The solution the problem of reliability: ***redundancy***.
  - the simplest: *mirroring* (duplicate all the drives)

- *Parallelism*: Improvement in Performance
  - With multiple drives, we can improve the transfer rate
    - by *striping* data across the drives.
  - ***bit-level striping***: splitting the *bits* of each *byte.*
    - if we have 8 drives, we can write i bit of each byte to drive i.
  - *block-level striping*: generalization to a number of drives.

- RAID Levels:
  - *mirroring*: highly reliable, however, too *expensive*.
  - *striping*: highly efficient, however, *not* related to *reliability*.
  - How about ***parity bit***?
    - set to 1: the number of bits in the byte is *even*.
    - set to 0: the number of bits in the byte is *odd*.
    - It enable us to detect an error if f one of the bits is damaged.
  - *RAID levels*: classify these schemes
    - according to *different cost-performance trade-offs*

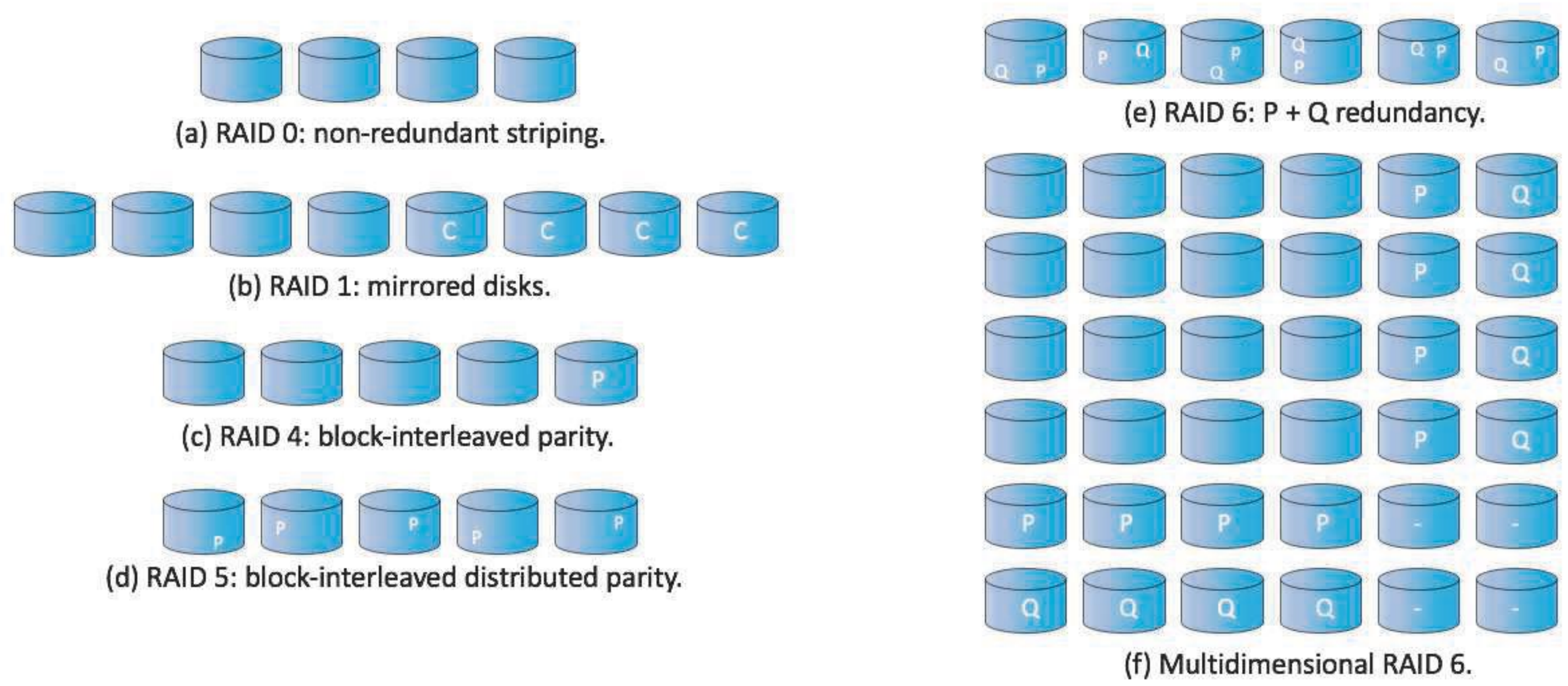


Figure 11.15 *RAID levels.*

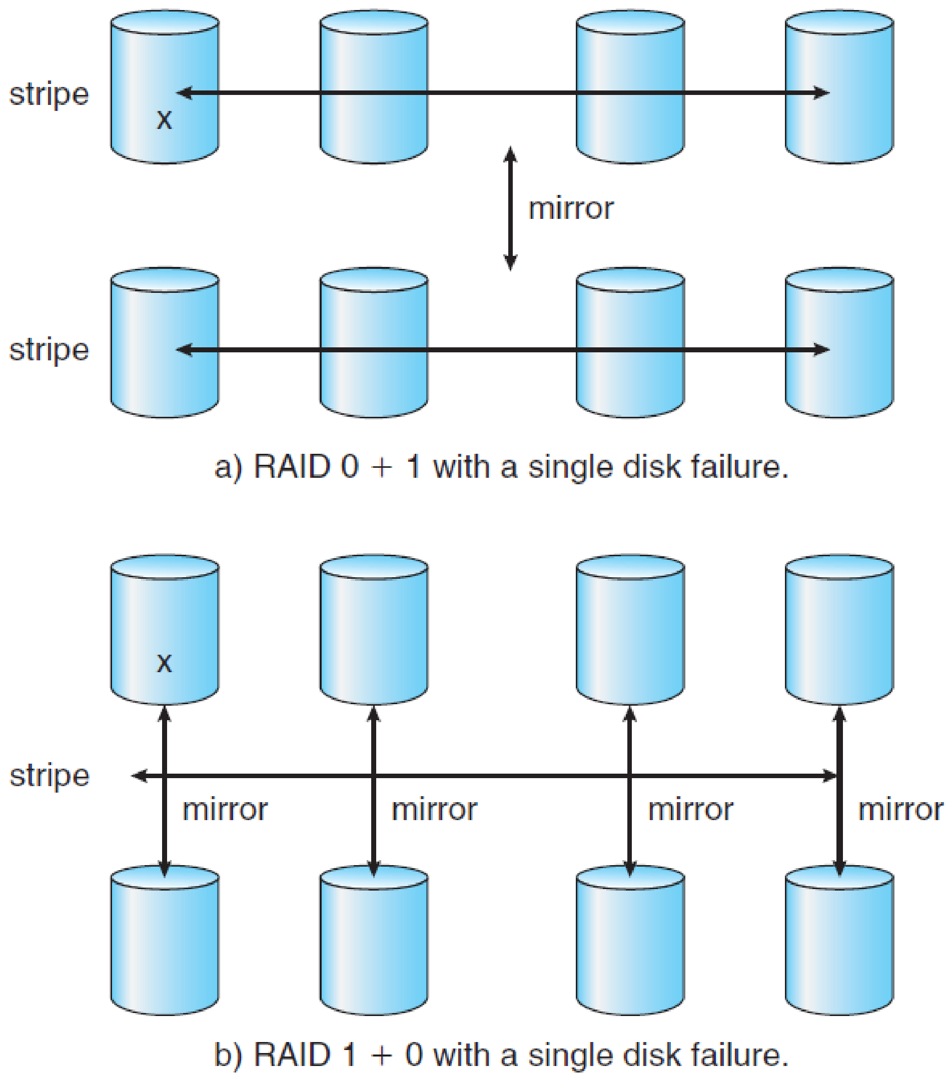


Figure 11.16 *RAID 0 + 1 and 1 + 0 with a single disk failure.*

- Two main jobs of a computer: I/O and computing.
  - In many cases, the main job is I/O,
    - for instance, web browsing, file editing, youtube, game, and so force.
  - The role of Operating System in I/O is
    - to manage and control I/O operations and I/O devices.

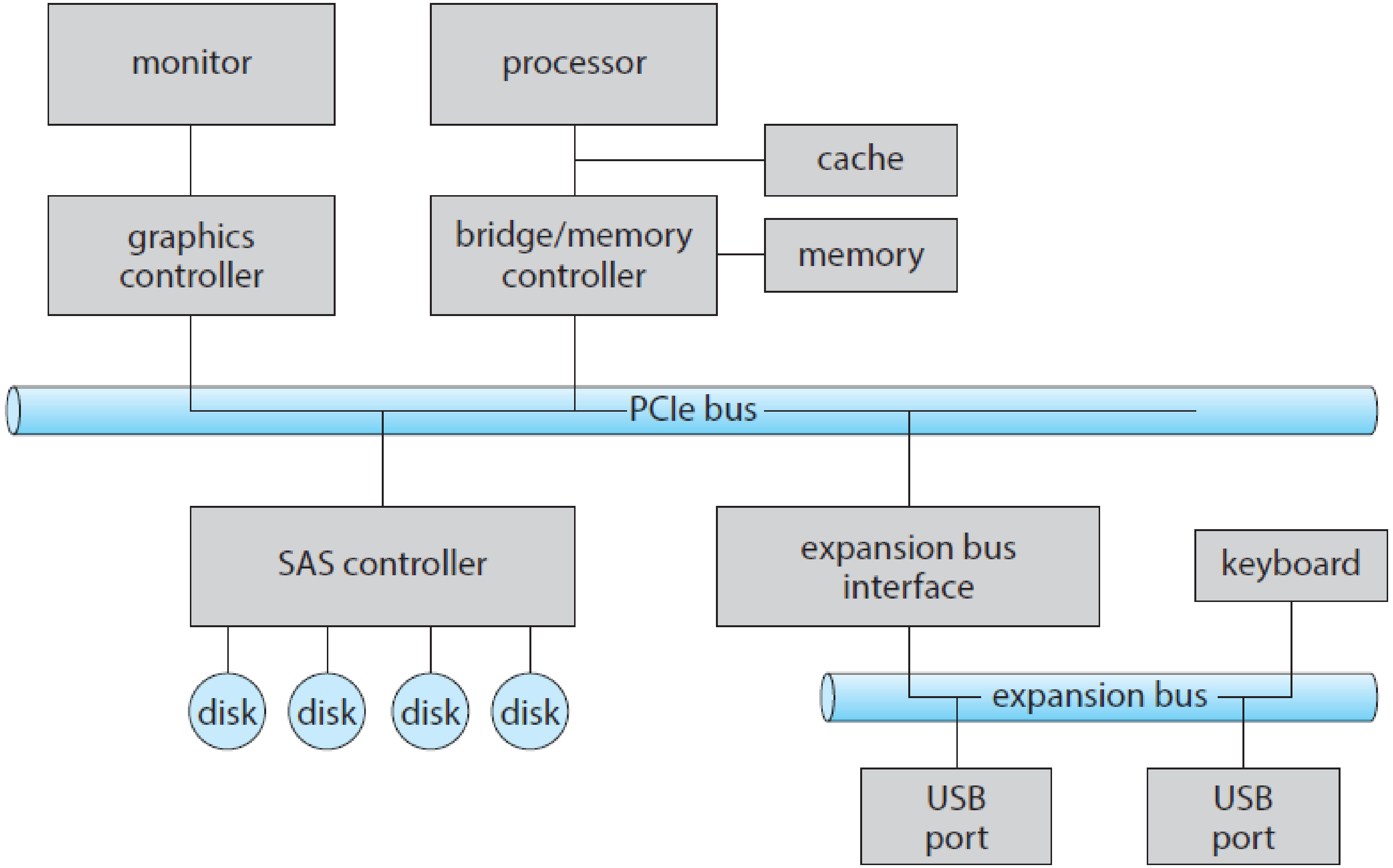


Figure 12.1 *A typical PC bus architecture.*

- Memory-Mapped I/O
  - How does the processor give commands and data to a controller to accomplish an I/O transfer?
  - The controller has one or more registers for data and control signals.
    - *data-in* register: is read by the host to get input.
    - *data-out* register: is written by the host to send output.
    - *status* register: contains bits that can be read by the host.
    - *control* register: can be written by the host to start a command or to change the mode of a device.

Memory-mapped I/O

- If the device supports the memory-mapped I/O,
  - control registers are mapped into the address space of the processor.
- The CPU executes I/O requests
  - using the standard data-transfer instructions to read and write the device-control registers at their mapped locations in physical memory.

I/O address range (hexadecimal)	device
000–00F	DMA controller
020–021	interrupt controller
040–043	timer
200–20F	game controller
2F8–2FF	serial port (secondary)
320–32F	hard-disk controller
378–37F	parallel port
3D0–3DF	graphics controller
3F0–3F7	diskette-drive controller
3F8–3FF	serial port (primary)

Figure 12.2 *Device I/O port locations on PCs (partial).*

- Three types of I/O:
  - ***polling***: or *busy-waiting*:
    - reading the status register repeatedly until the busy bit becomes clear.
  - ***interrupt***:
    - CPU has a wire called the interrupt-request line.
    - If CPU detects an interrupt,
      it jumps to an ISR(*interrupt service routine*) to handle an interrupt.
    - The addresses of ISRs is specified in the *interrupt vector table*.
  - ***DMA***: *Direct Memory Access*:
    - used to avoid programmed I/O (one byte at a time).
    - useful for handling large data transfer.

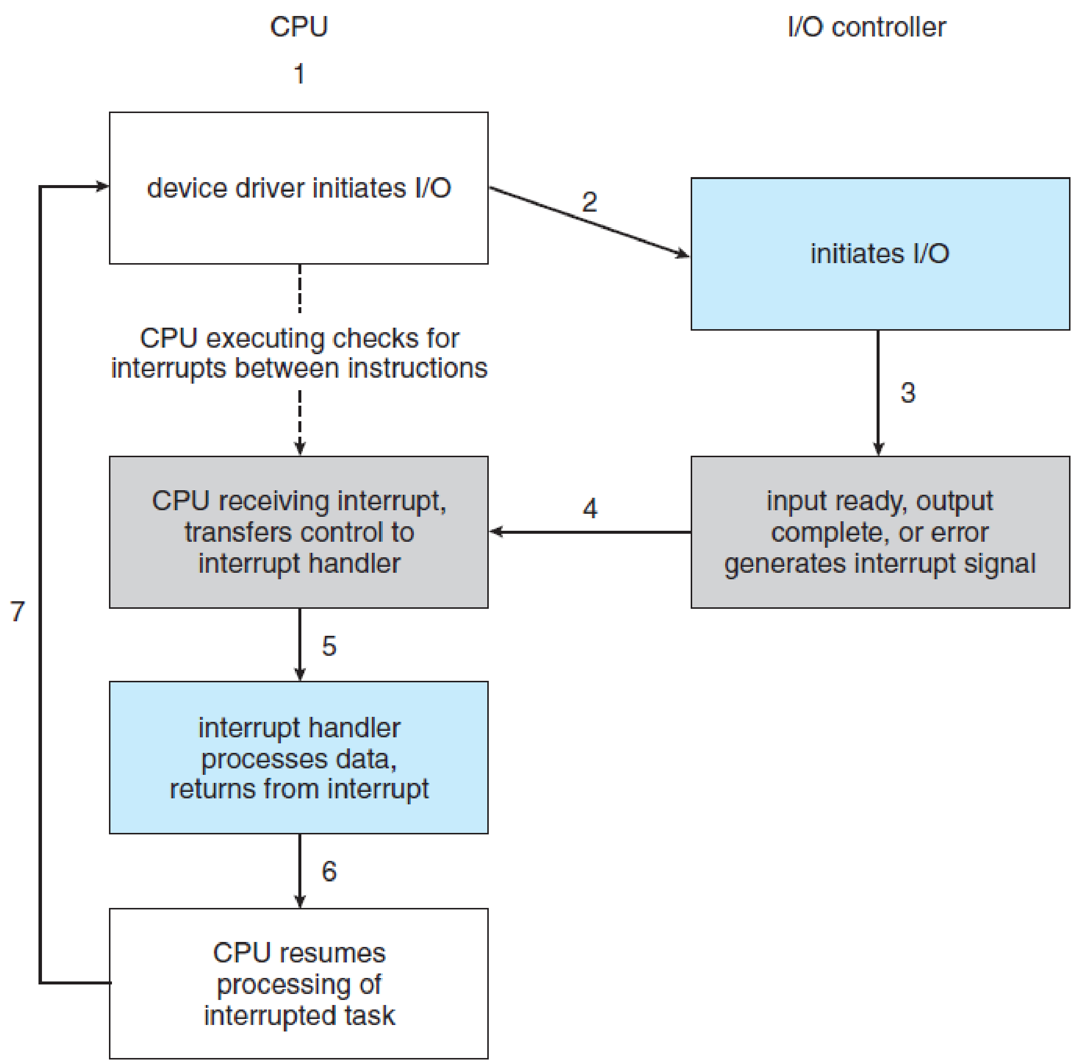


Figure 12.3 *Interrupt-driven I/O cycle.*

vector number	description
0	divide error
1	debug exception
2	null interrupt
3	breakpoint
4	INTO-detected overflow
5	bound range exception
6	invalid opcode
7	device not available
8	double fault
9	coprocessor segment overrun (reserved)
10	invalid task state segment
11	segment not present
12	stack fault
13	general protection
14	page fault
15	(Intel reserved, do not use)
16	floating-point error
17	alignment check
18	machine check
19–31	(Intel reserved, do not use)
32–255	maskable interrupts

Figure 12.5 *Intel Pentium processor event-vector table.*

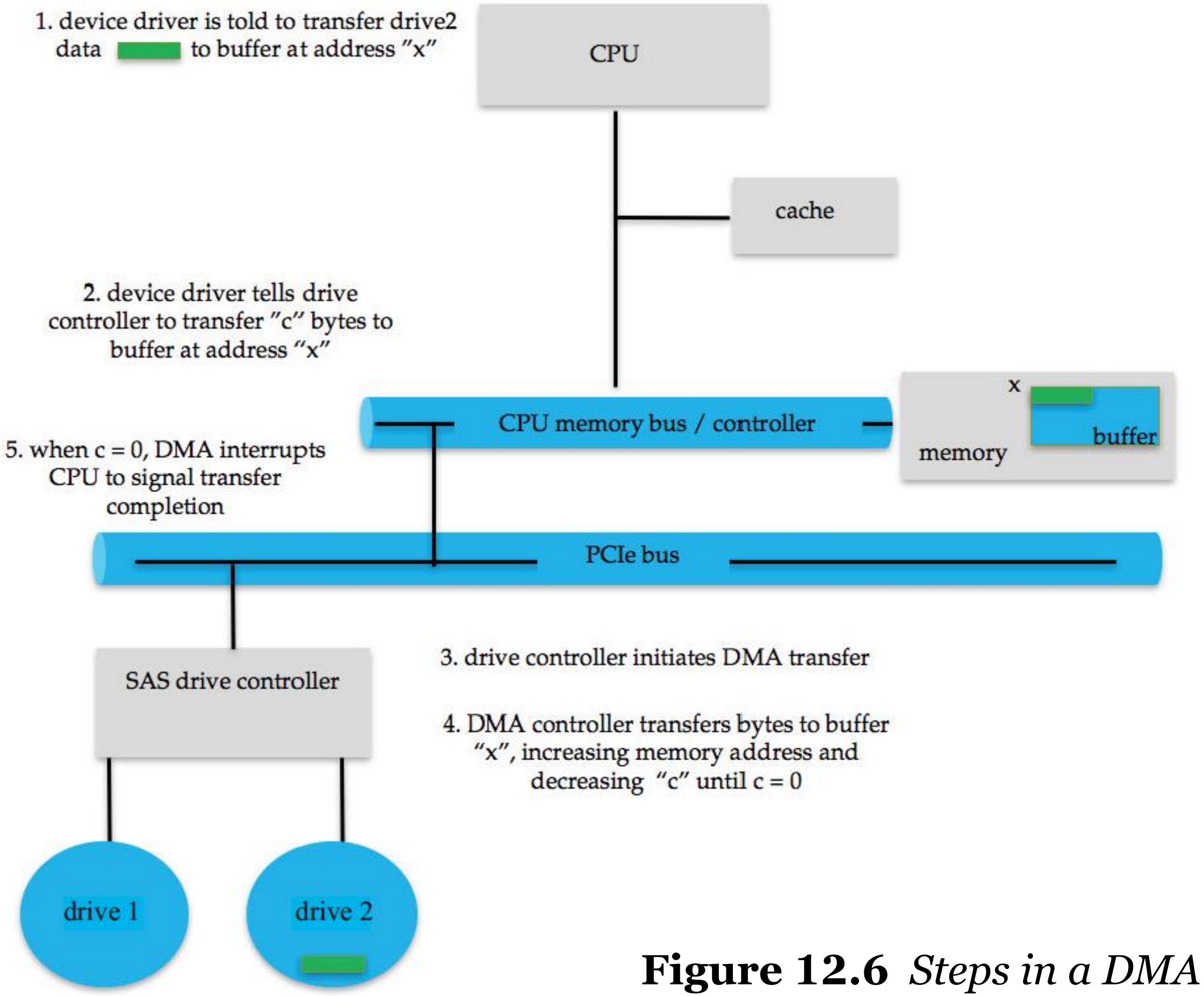


Figure 12.6 *Steps in a DMA transfer.*

- Blocking I/O .vs. Non-blocking I/O
  - *Blocking* I/O: a thread is suspended.
    - moved from *running* queue to *waiting* queue.
  - *Non-blocking* I/O: does not halt the execution of the thread.
    - e.g., receiving keyboard or mouse input in word processor.
    - returns as much as available.
  - *Asynchronous system call*: the thread continues to execute its code.

- The diff. between *non-blocking* and *asynchronous* system call:
  - a *non-blocking* read() call
    - *returns immediately* with whatever data are available,
    - the full number of bytes requested, fewer, or none at all.
  - An *asynchronous* read() call
    - *requests a transfer* that will be performed in its entirety
    - but will complete at some future time.

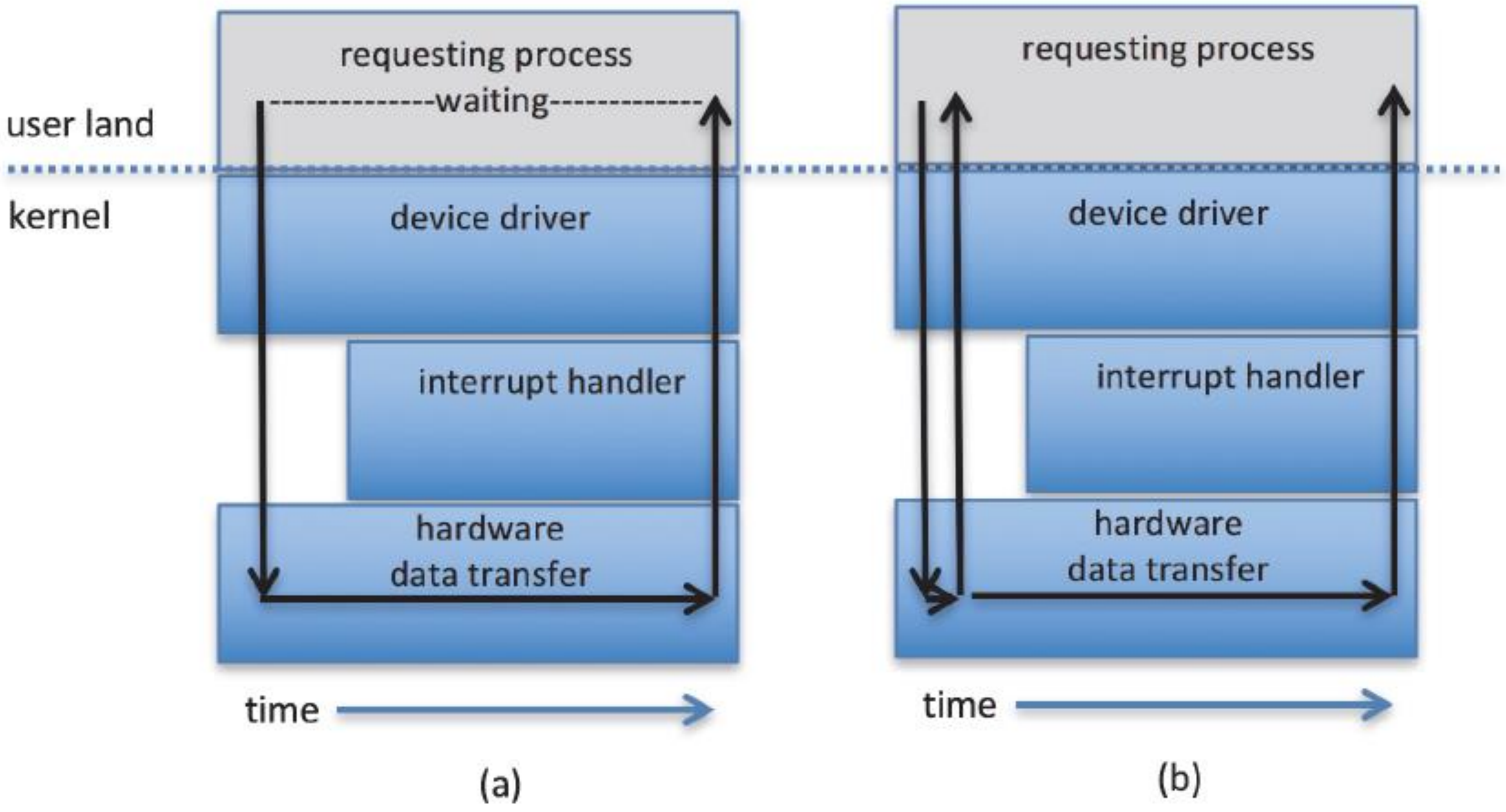


Figure 12.9 *Two I/O methods: (a) synchronous (b) asynchronous.*

- **File System**:
  - provides the mechanism for
    - on-line storage of and access to both *data* and *programs*
    - of the *operating system* and *all the users* of the computer system.
  - consists of two distinct parts
    - a collection of *files*, each storing related data
    - a *directory* structure, which organizes all the files in the system.

- Access Methods:
  - *sequential access*:
    - Information in the file is processed in order,
    - one record after the other.
  - *direct access*: relative access:
    - A file is made up of fixed-length logical records that
    - allow programs to read and write records rapidly in no particular order.

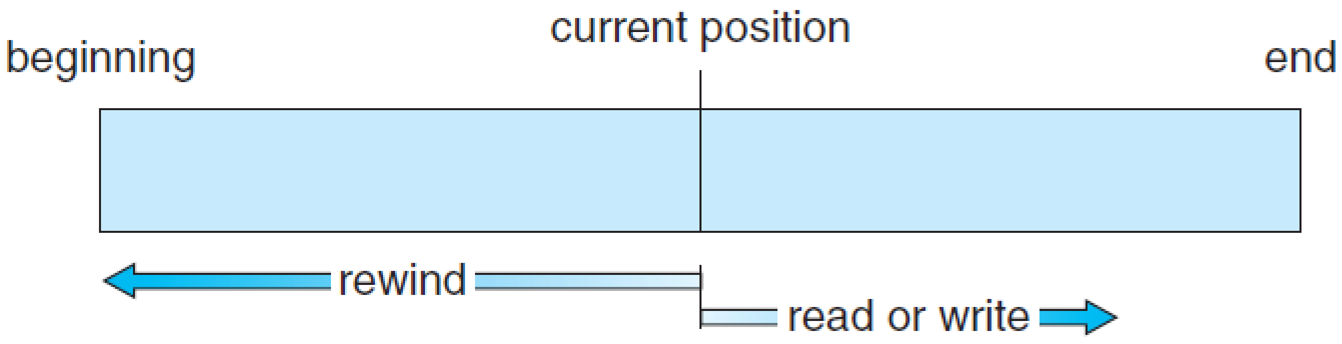


Figure 13.4 *Sequential-access file.*

sequential access	implementation for direct access
reset	cp = 0;
read_next	read cp; cp = cp + 1;
write_next	write cp; cp = cp + 1;

Figure 13.5 *Simulation of sequential access on a directed-access file.*

- Directory Structure:
  - The directory can be viewed as
    - a symbol table that translates file names into theirs file control blocks.
  - The ways of organizing the directory structure:
    - *Single-Level Directory*
    - *Two-Level Directory*
    - *Tree-Structured Directories*
    - *Acyclic-Graph Directories*

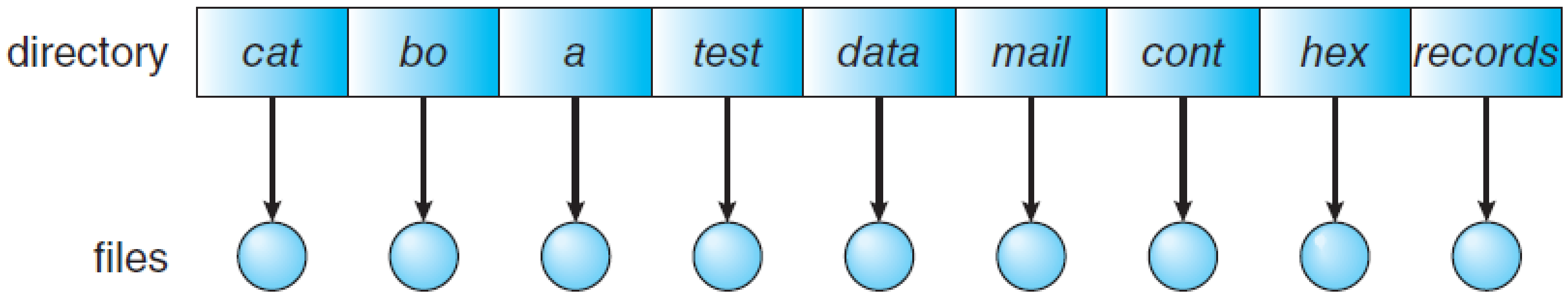


Figure 13.7 *Single-level directory.*

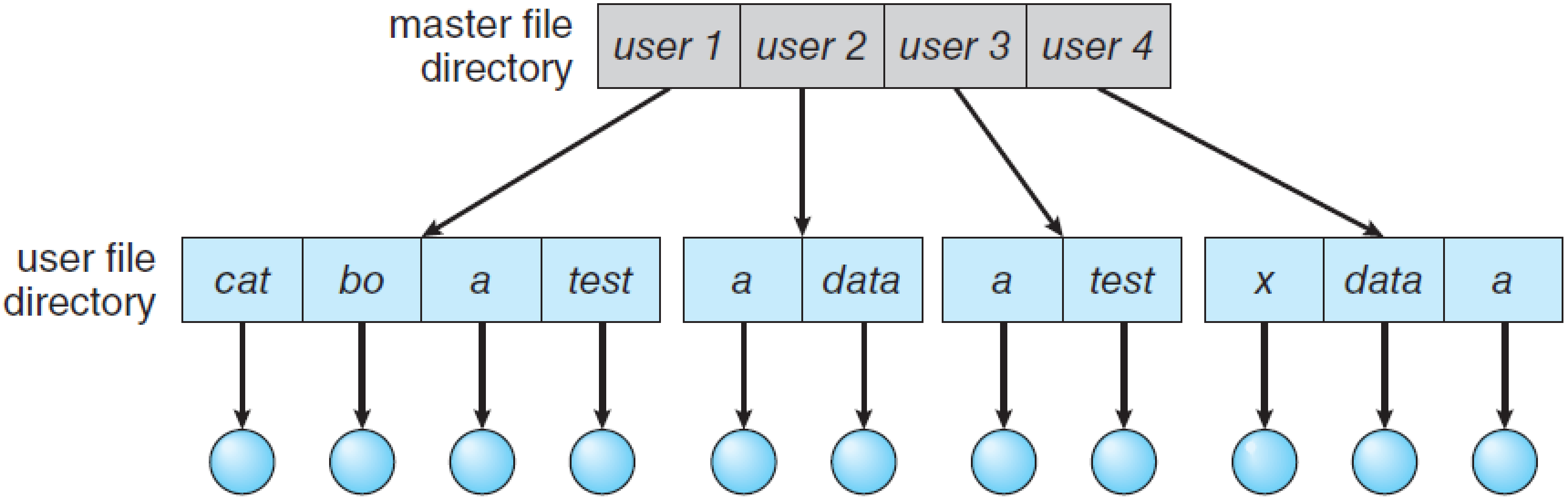


Figure 13.8 *Two-level directory.*

Ch.13 File-System Interface

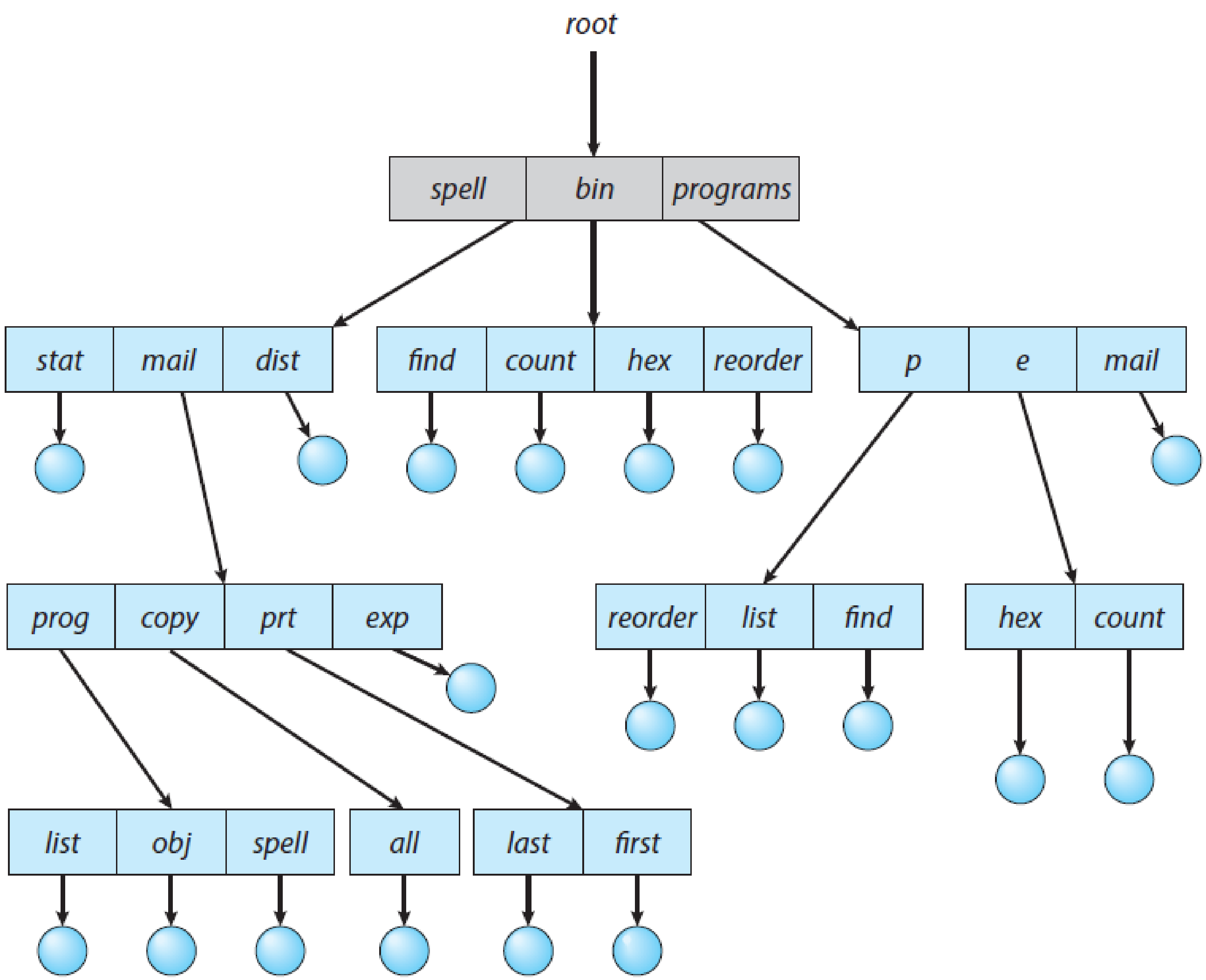


Figure 13.8 *Tree-structured directory structure.*

Ch.13 File-System Interface

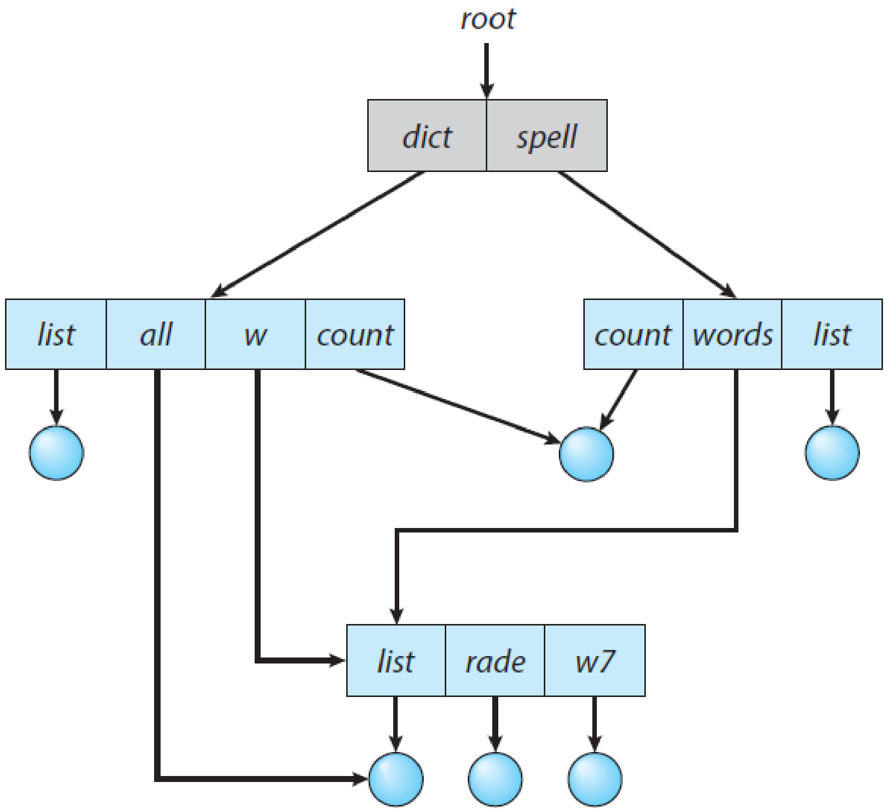


Figure 13.10 *Acyclic-graph directory structure.*

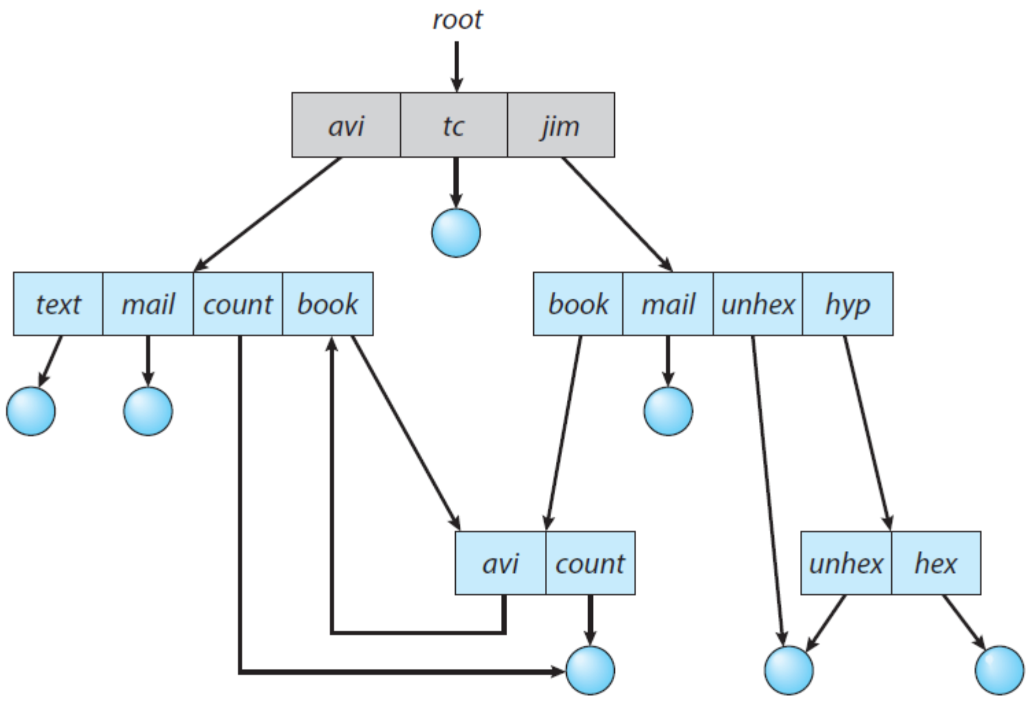


Figure 13.11 *General graph directory.*

- The file system itself
  - is generally composed of many different levels.

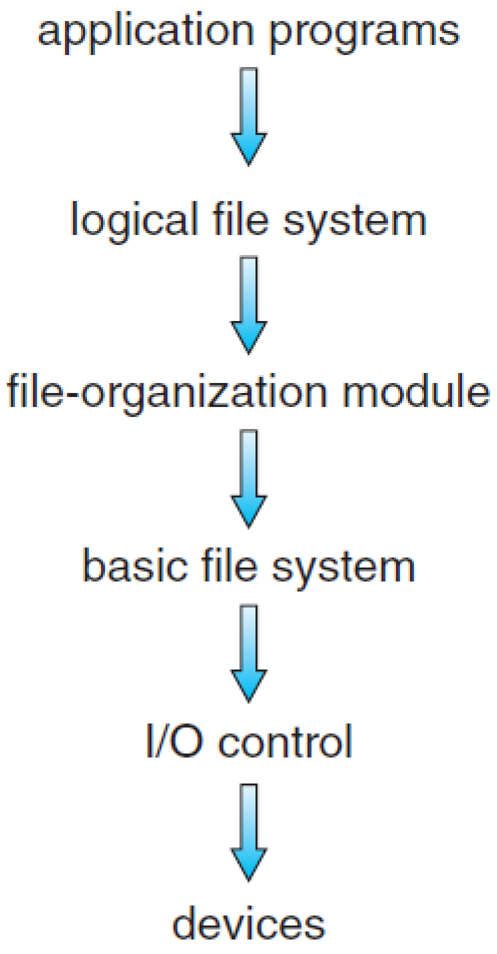
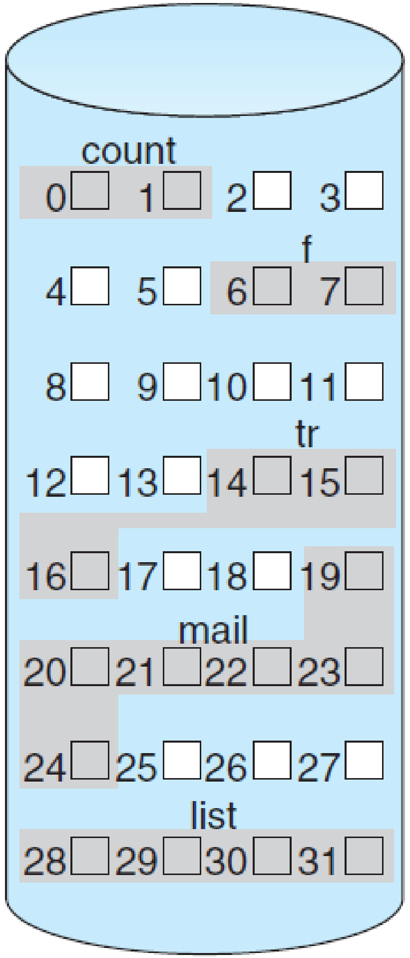


Figure 14.1 *Layered file system.*

- ***Allocation Method***:
  - The main problem of implementing the file system:
    - how to *allocate space to files*
    - so that storage space is *utilized effectively*
    - and files can be *accessed quickly*.
  - Three major methods in wise use:
    - *Contiguous Allocation*
    - *Linked Allocation*
    - ***Indexed Allocation***

- *Contiguous Allocation*:
  - requires that each file occupy a set of contiguous blocks on the device.



directory

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

Figure 14.4 *Contiguous allocation of disk space.*

- *Linked Allocation*:
  - The problems of contiguous allocation:
    - *external fragmentation*.
    - *need for compaction*.
  - Linked Allocation
    - solves all problems of contiguous allocation
    - each file is a linked list of storage blocks
    - the blocks may be scattered anywhere on the device.

Ch. 14 File-System Implementation

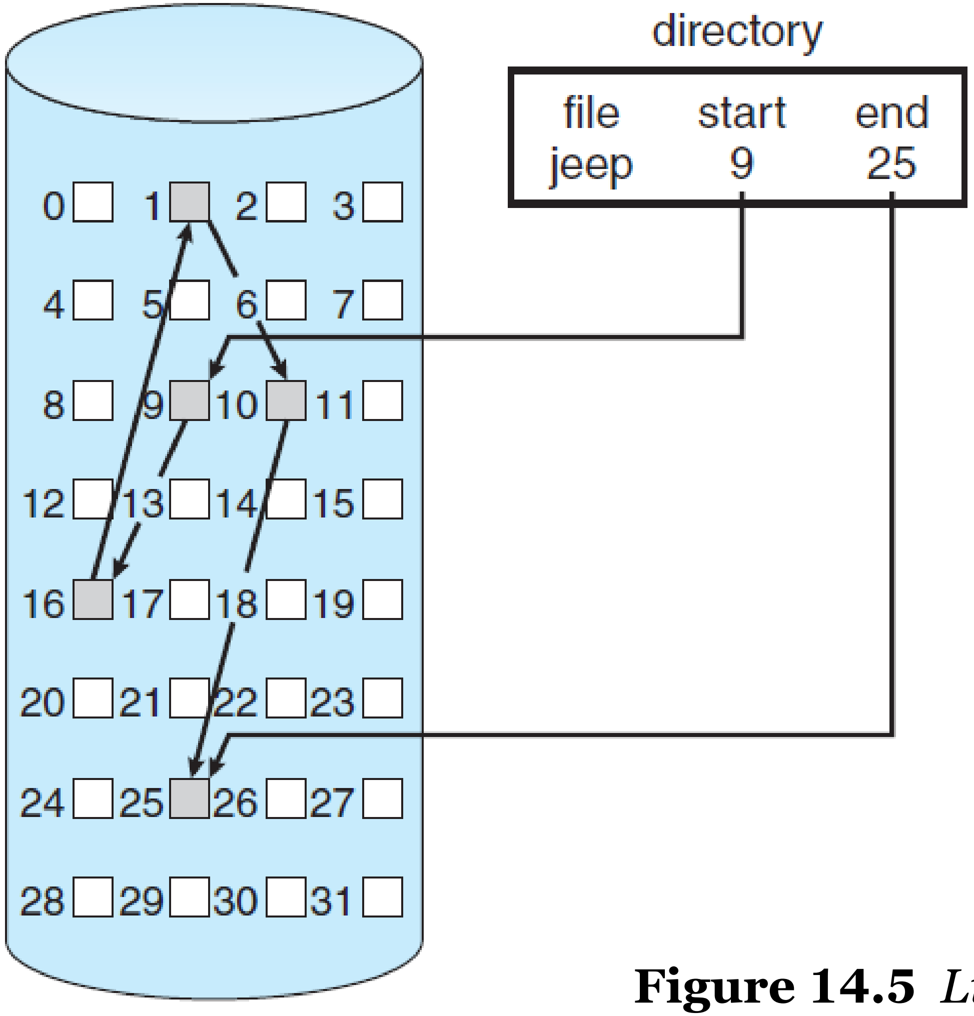


Figure 14.5 *Linked allocation of disk space.*

- *FAT*: *File Allocation Table*
  - The disadvantages of the linked allocation:
    - It can be used effectively only for sequential-access files
    - To find *i*-th block of a file, we must start at the beginning of that file.
  - FAT: an important variation on the linked allocation
    - simple but efficient method using a *file allocation table*.
  - A section of storage at the beginning of each volume
    - is set aside to contain the table.
    - the table has one entry for each block and is indexed by block number.

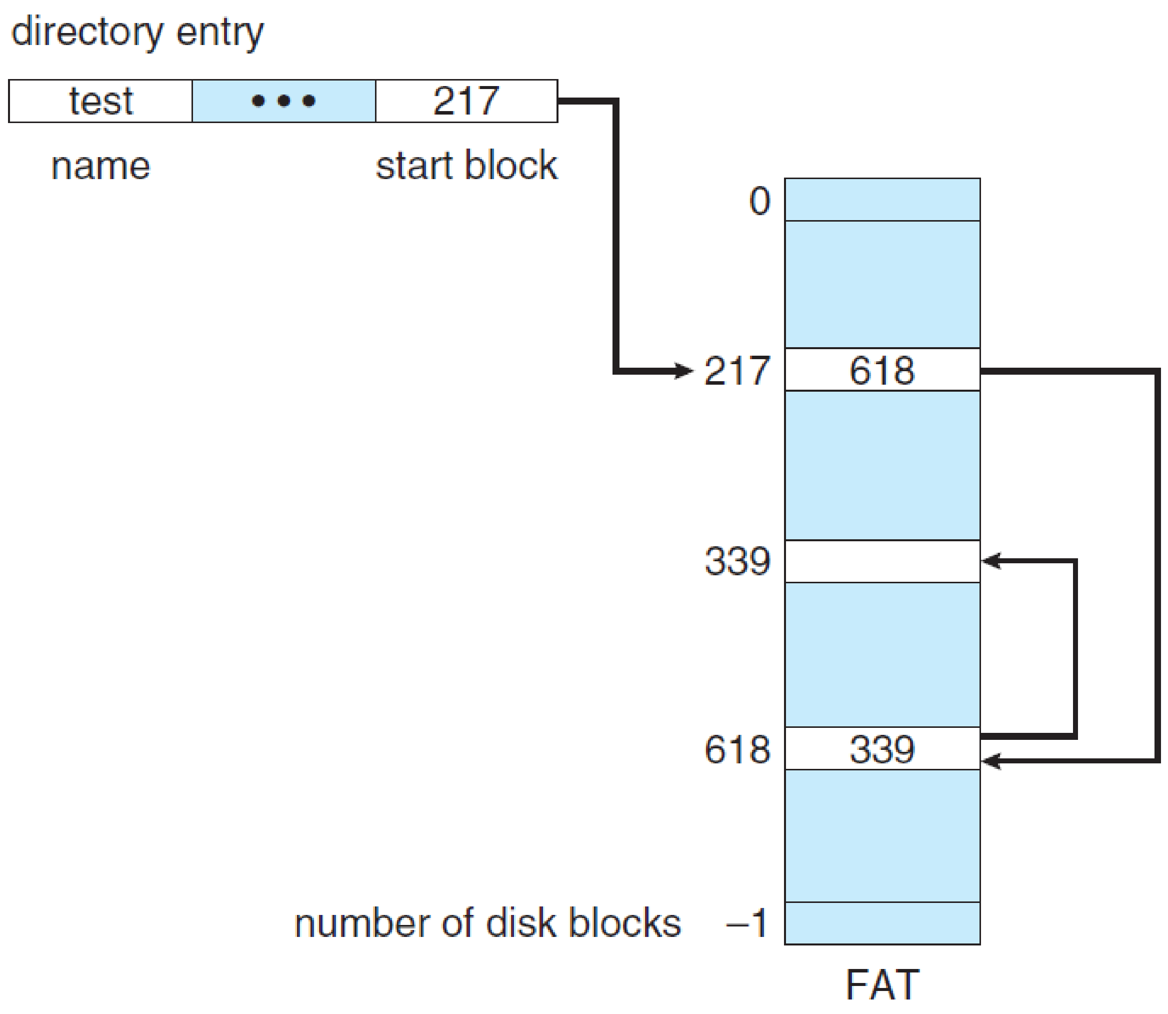


Figure 14.6 *File Allocation Table.*

- ***Indexed Allocation***:
  - The problems of linked allocation (without the FAT)
    - the pointers to the blocks are scattered with blocks
  - Indexed Allocation solves this problem
    - by bringing all the pointers together into the ***index block***.
  - Each file has its own index block,
    - which is an array of storage-block addresses.
    - the i-th entry in the index block points to the i-th block of the file.

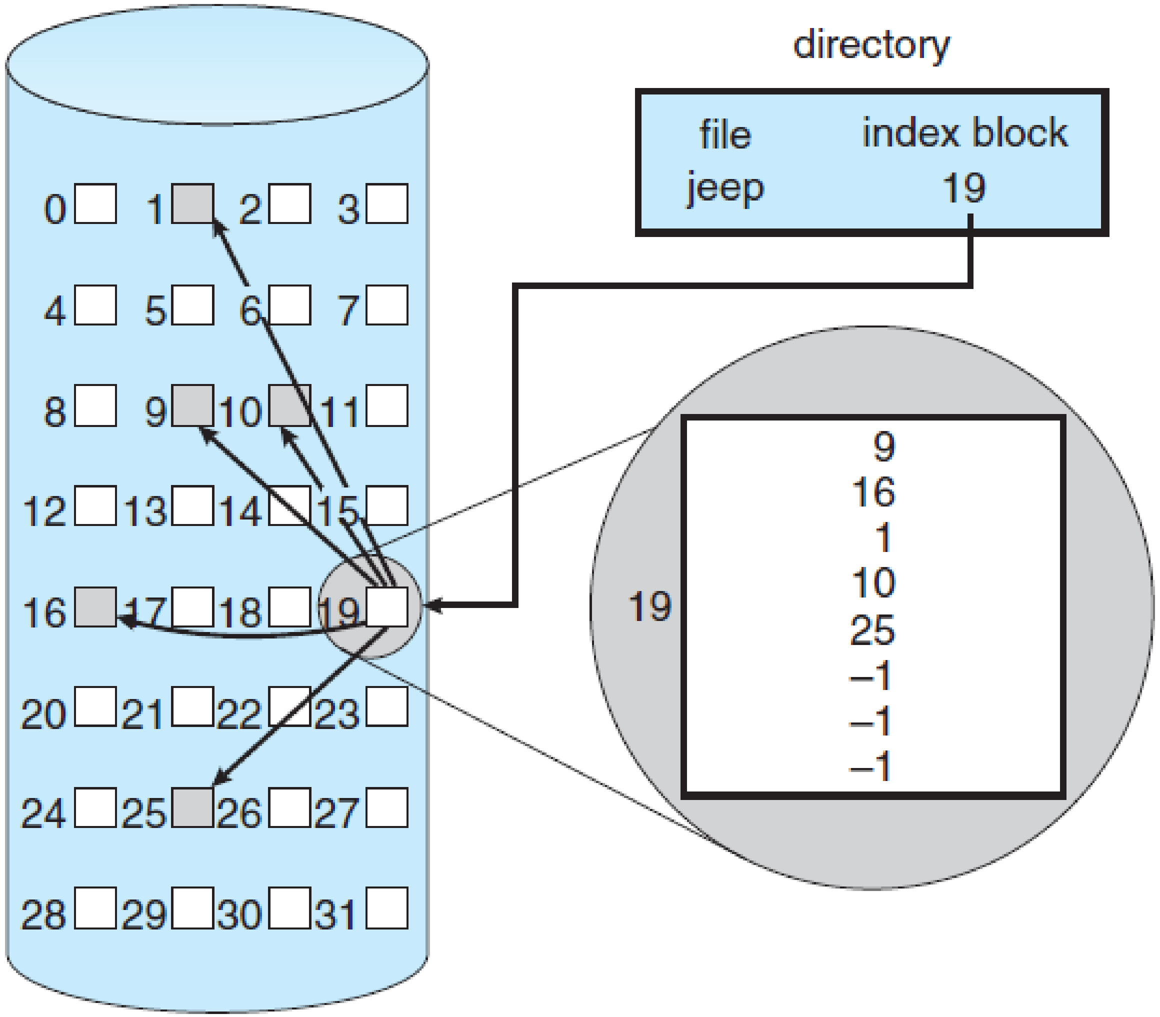


Figure 14.7 *Indexed allocation of disk space.*

- Free-Space Management
  - To keep track of free disk space,
    - the system maintains a *free-space list*.

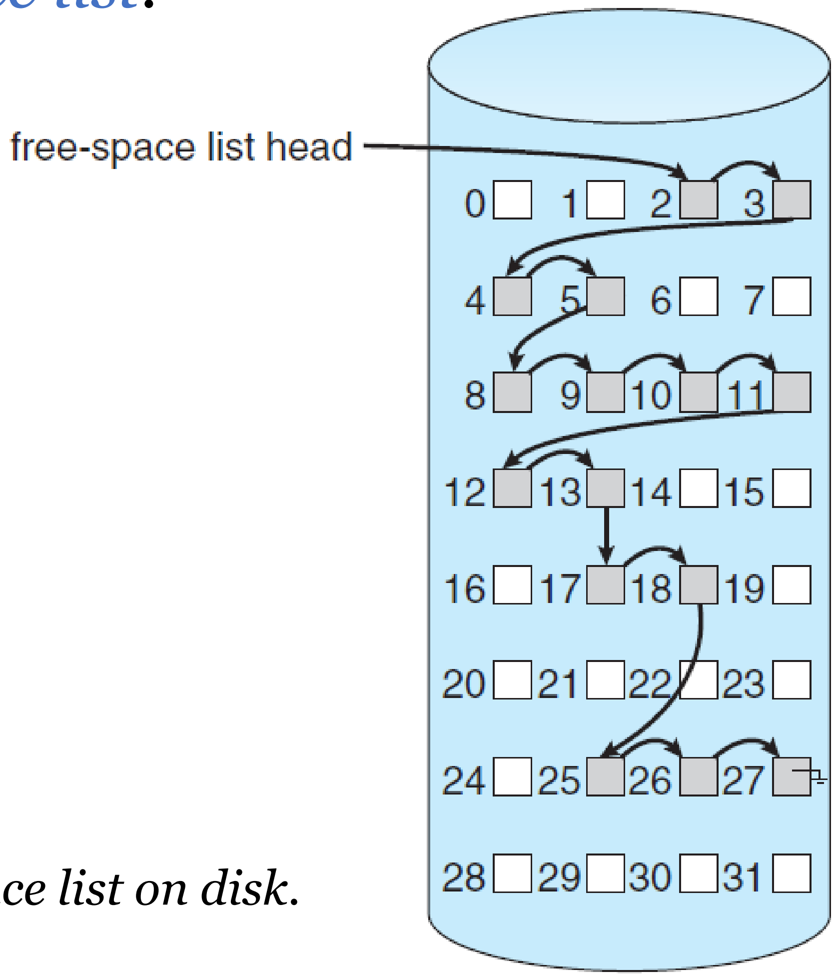


Figure 14.9 *Linked free-space list on disk.*

Any Questions?