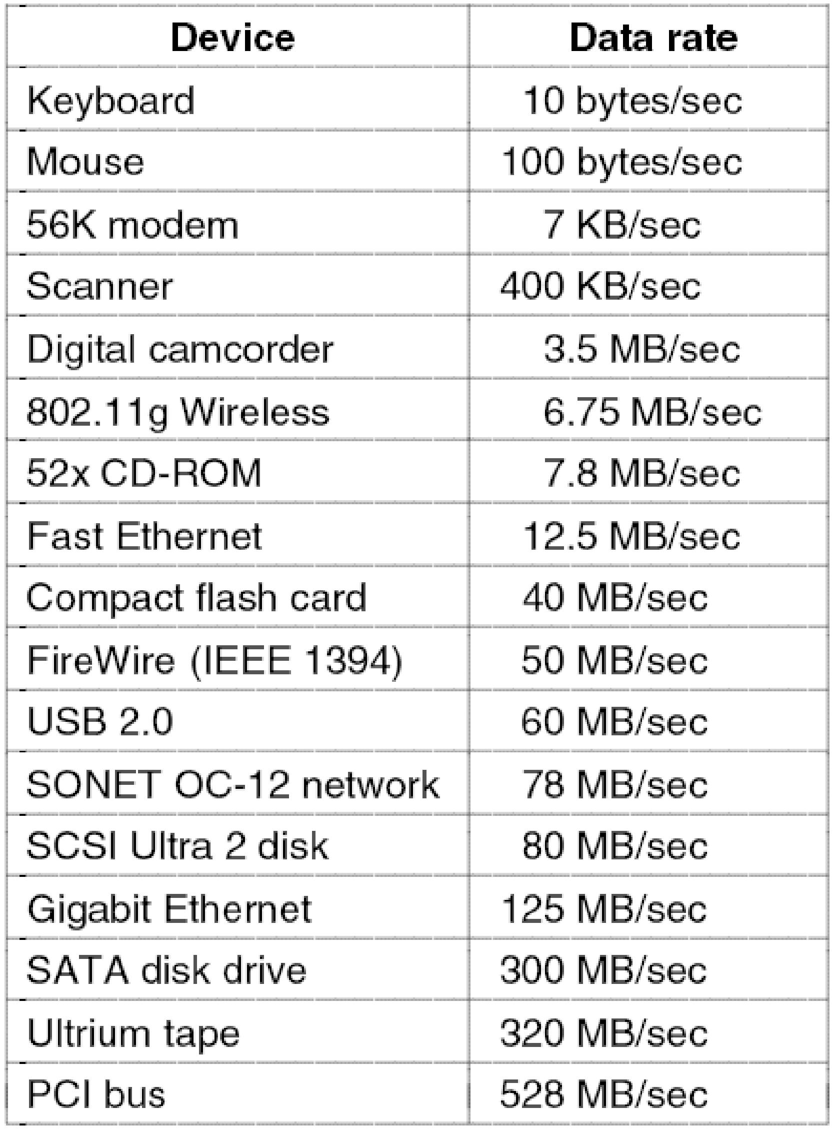
MODERN OPERATING SYSTEMS

Third Edition
ANDREW S. TANENBAUM

Chapter 5
Input/Output

I/O Devices & Data Rate



Device	Data rate
Keyboard	10 bytes/sec
Mouse	100 bytes/sec
56K modem	7 KB/sec
Scanner	400 KB/sec
Digital camcorder	3.5 MB/sec
802.11g Wireless	6.75 MB/sec
52x CD-ROM	7.8 MB/sec
Fast Ethernet	12.5 MB/sec
Compact flash card	40 MB/sec
FireWire (IEEE 1394)	50 MB/sec
USB 2.0	60 MB/sec
SONET OC-12 network	78 MB/sec
SCSI Ultra 2 disk	80 MB/sec
Gigabit Ethernet	125 MB/sec
SATA disk drive	300 MB/sec
Ultrium tape	320 MB/sec
PCI bus	528 MB/sec

Some typical device, network, and bus data rates.

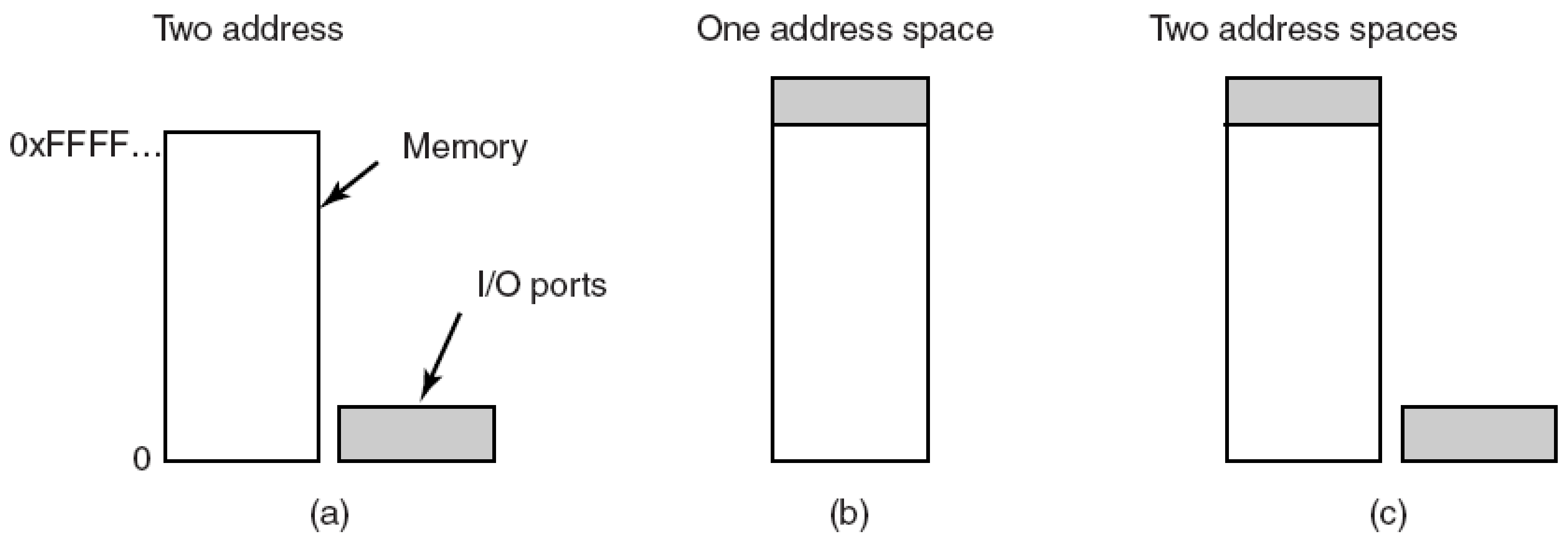
Block Device & Character Device

- Block Device
  - Read/Write Unit: Block
  - Each block has an address
  - Typical Operation: Read/Write N_{th} block, Seek
  - Ex: Hard Drive, CD-ROM, USB
- Character Device
  - Read/Write Unit: Char
  - Each char has no address
  - Ex: Mouse, Printer, Modem

Device Controllers

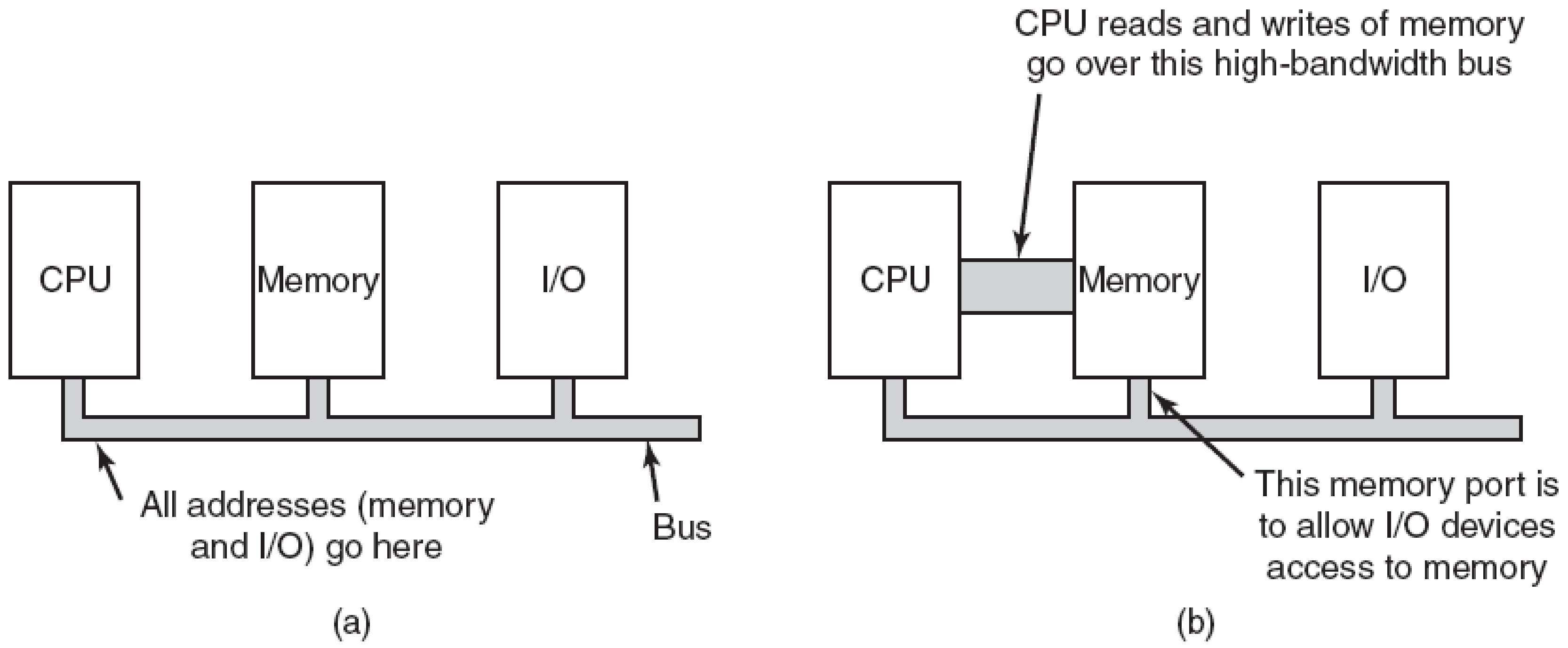
- I/O Devices have
  - Mechanical component
  - Electronic component
- The electronic component: device controller
  - May be able to handle multiple devices
- Controller’s tasks
  - Convert bit stream to block of bytes
  - Perform error correction as necessary
  - Decode command register to low level signal
  - Send an interrupt to CPU
  - Manage buffers within it

Memory-Mapped I/O



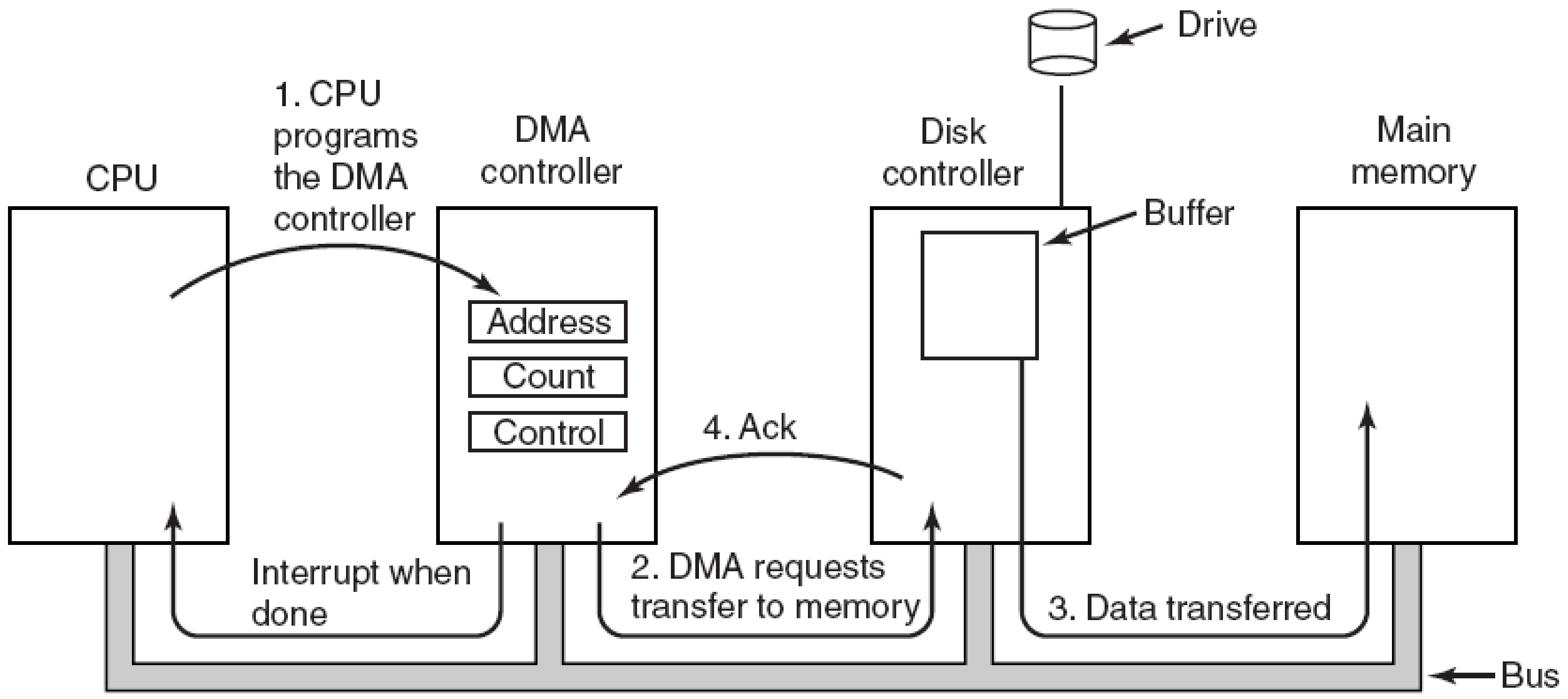
- Separate I/O and memory space
- Memory-mapped I/O
- Hybrid.

Memory Bus Architecture


CPU reads and writes of memory go over this high-bandwidth bus
CPU
Memory
I/O
All addresses (memory and I/O) go here
Bus
(a)
CPU
Memory
I/O
This memory port is to allow I/O devices access to memory
(b)

- A single-bus architecture
- A dual-bus memory architecture.

Direct Memory Access (DMA)

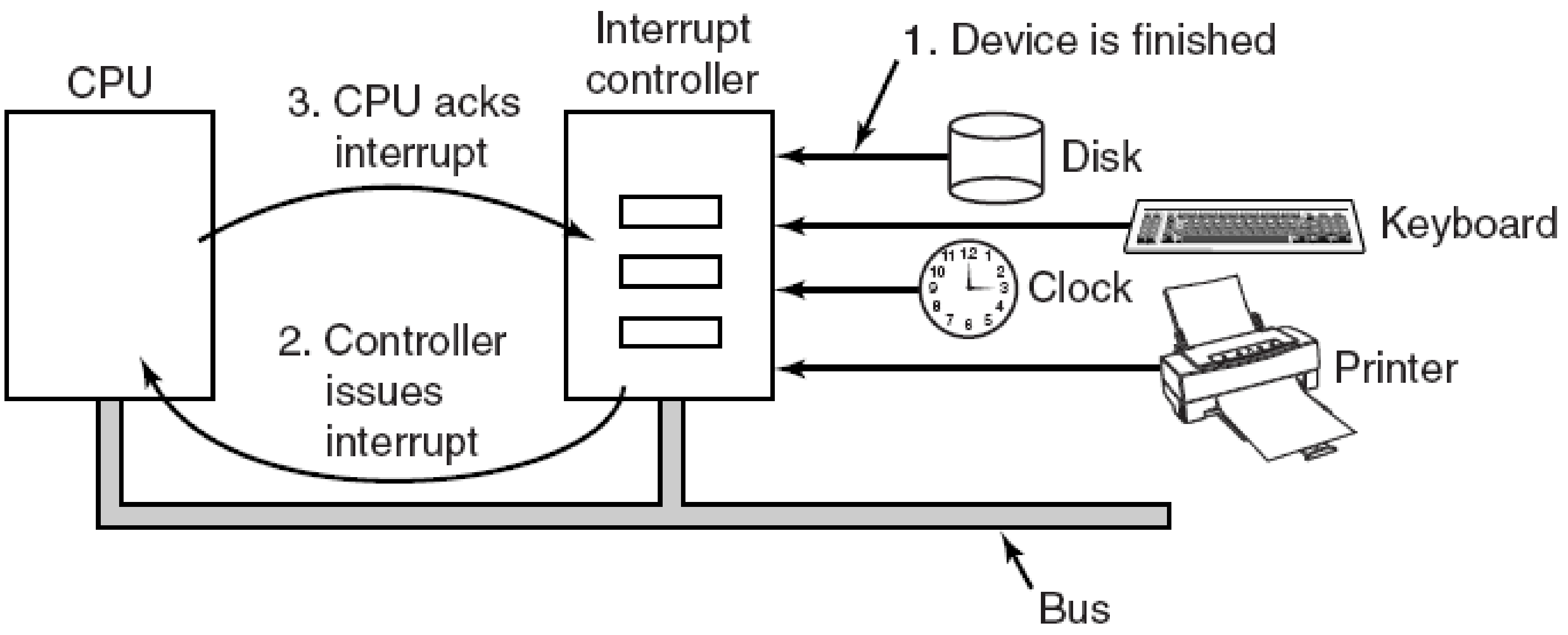


Operation of a DMA transfer.

DMA (Direct Memory Access)

- Bus transfer mode
  - Word-at-a-time mode
    - DMA uses Cycling Stealing
  - Burst Mode
- Data transfer method of DMA
  - Fly-by-mode
  - Two bus requests to transfer data from device controller to memory or vice versa

Interrupts Revisited



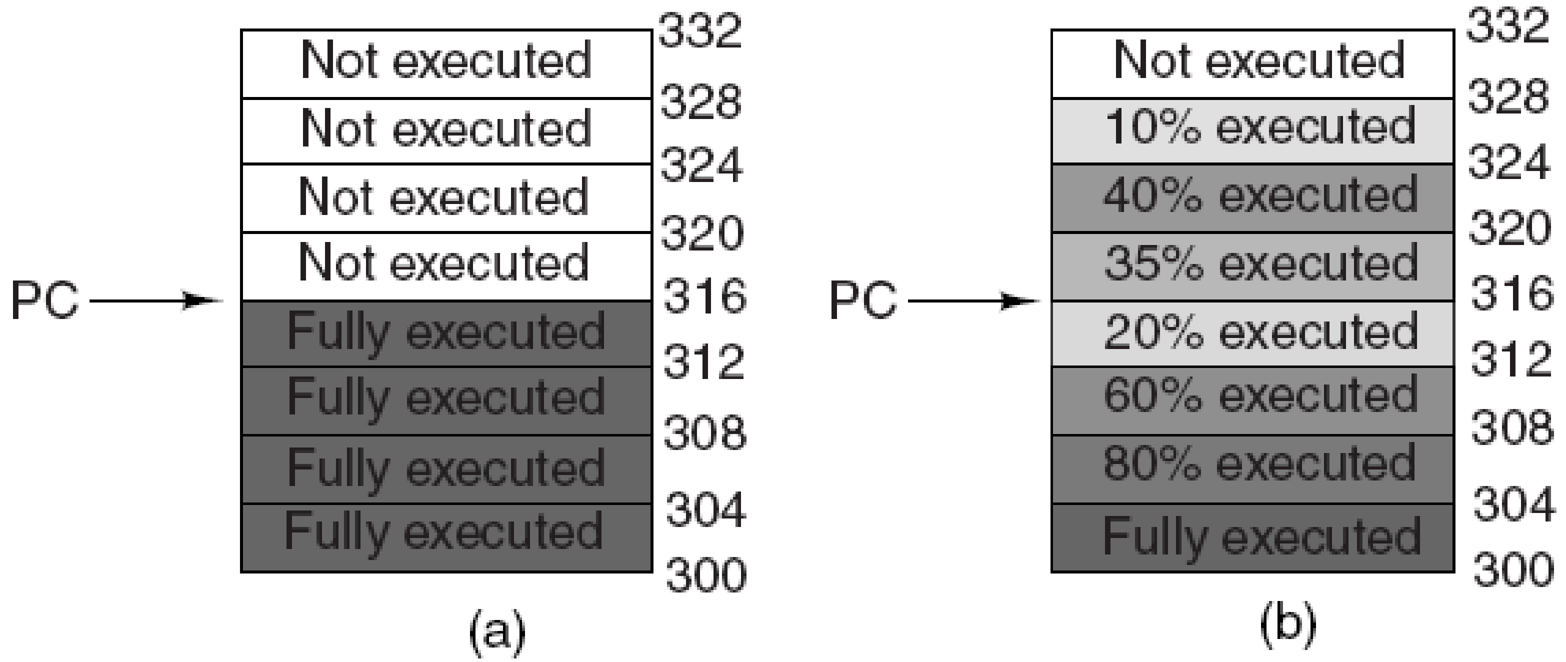
How an interrupt happens. The connections between the devices and the interrupt controller actually use interrupt lines on the bus rather than dedicated wires.

Precise and Imprecise Interrupts (1)

Properties of a *precise interrupt*

1. PC (Program Counter) is saved in a known place.
2. All instructions before the one pointed to by the PC have fully executed.
3. No instruction beyond the one pointed to by the PC has been executed.
4. Execution state of the instruction pointed to by the PC is known.

Precise and Imprecise Interrupts (2)



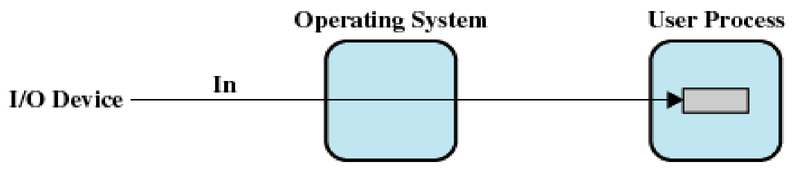
- (a) A precise interrupt
- (b) An imprecise interrupt

Goal of I/O Software (1)

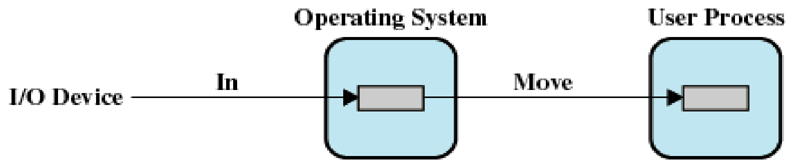
- Device Independency
  - Program can access any I/O devices with the same way
  - Ex) HDD, CD-ROM, or USB stick
  - Ex) Sort < input > output
- Uniform Naming
  - Name of a file or device is a string or an integer
  - Mount a file system of a device to root file system
- Error Handling
  - Handle errors as close to the hardware as possible

Goal of I/O Software (2)

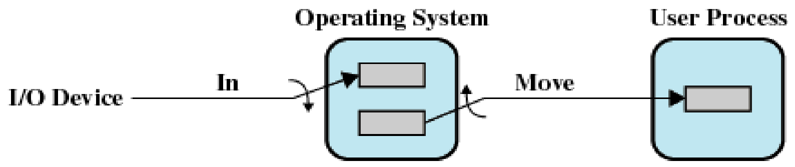
- Synchronous vs asynchronous
  - Blocked transfer vs. interrupt-driven
- Buffering
  - Single Buffering
  - Double Buffering
- Sharable vs. dedicated devices
  - Disks are sharable
  - Tape drivers would not be



(a) No buffering

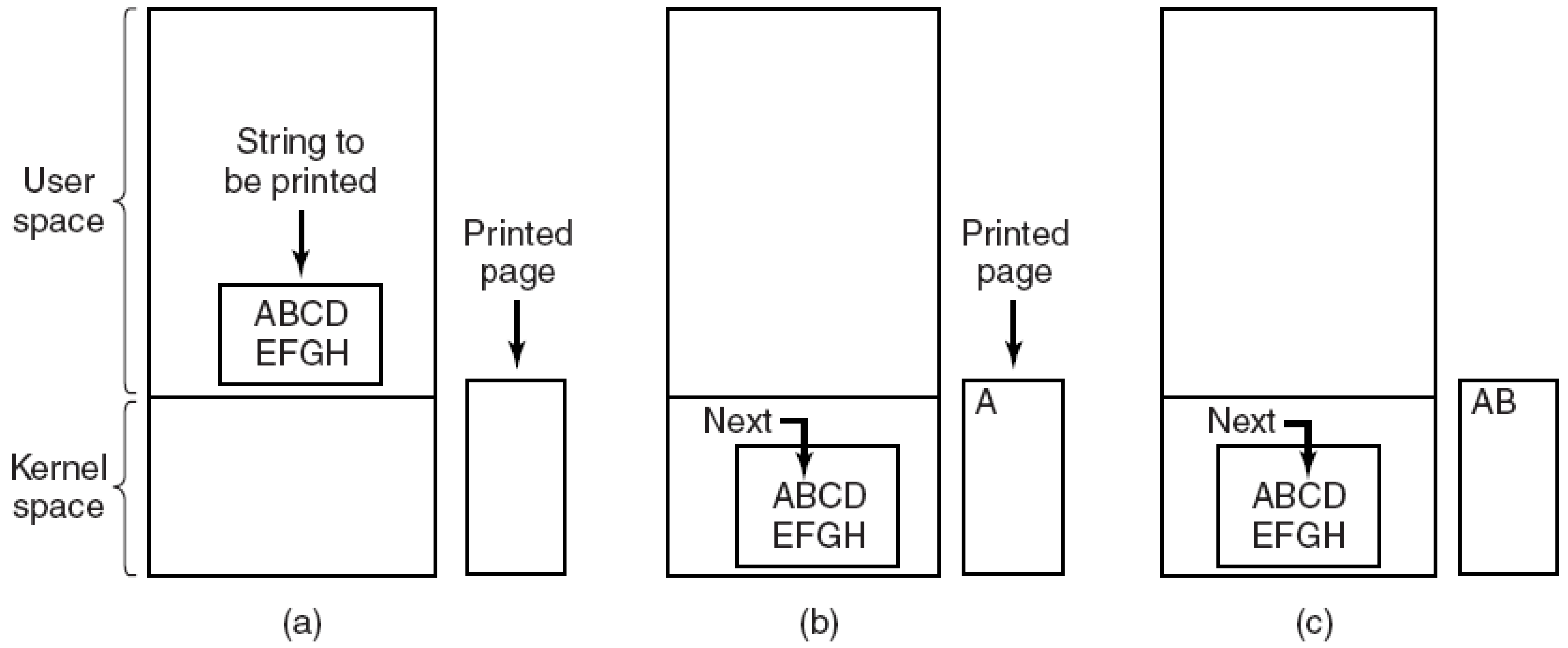


(b) Single buffering



(c) Double buffering

Programmed I/O (1)



Steps in printing a string.

Programmed I/O (2)

```
copy_from_user(buffer, p, count);                    /* p is the kernel buffer */
for (i = 0; i < count; i++) {                        /* loop on every character */
     while (*printer_status_reg != READY) ;       /* loop until ready */
     *printer_data_register = p[i];                  /* output one character */
}
return_to_user( );
```

Writing a string to the printer using programmed I/O.

Interrupt-Driven I/O

```
copy_from_user(buffer, p, count);
enable_interrupts( );
while (*printer_status_reg != READY) ;
*printer_data_register = p[0];
scheduler( );
```

(a)

```
if (count == 0) {
      unblock_user( );
} else {
      *printer_data_register = p[i];
      count = count – 1;
       i = i + 1;
}
acknowledge_interrupt( );
return_from_interrupt( );
```

(b)

Writing a string to the printer using interrupt-driven I/O.

- Code executed at the time the print system call is made.
- Interrupt service procedure for the printer.

I/O Using DMA

```
copy_from_user(buffer, p, count);
set_up_DMA_controller( );
scheduler( );
```

(a)

```
acknowledge_interrupt( );
unblock_user( );
return_from_interrupt( );
```

(b)

Printing a string using DMA.

- Code executed when the print system call is made.
- Interrupt service procedure.

I/O Software Layers

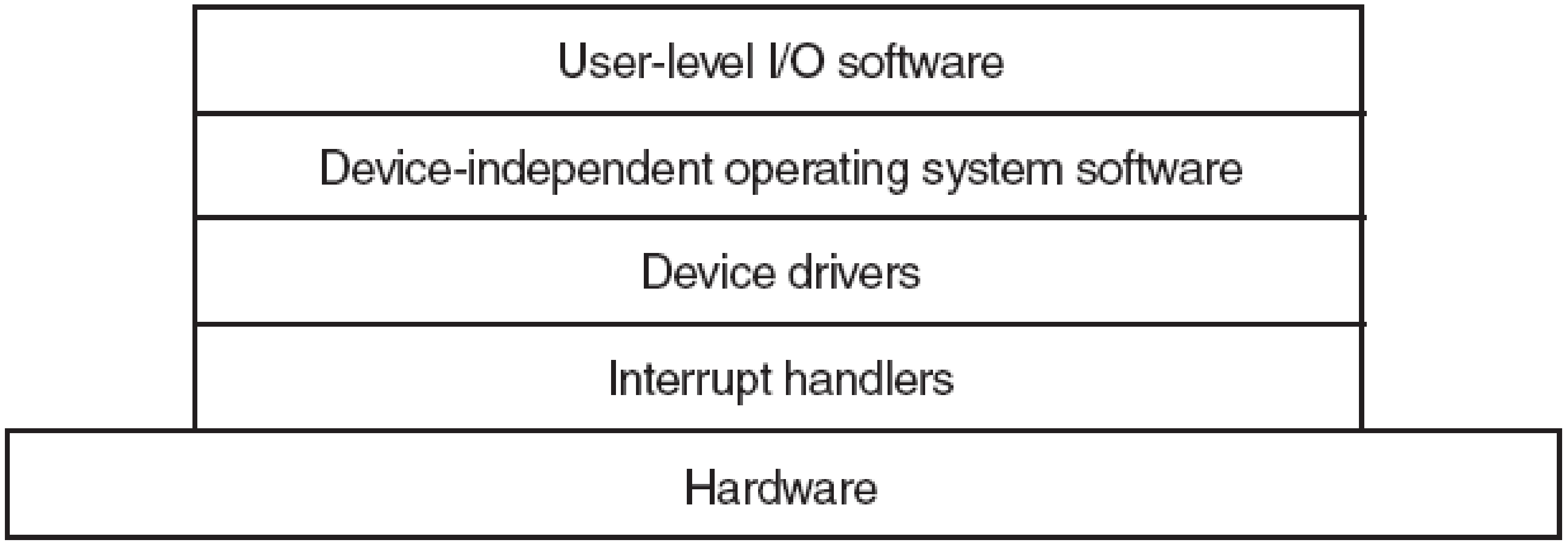


Figure 5-11. Layers of the I/O software system.

Interrupt Handlers (2)

- Interrupt handlers are best hidden
  - Have driver starting an I/O operation block until interrupt notifies of completion
- Interrupt procedure does its task
  - Then unblock driver that started it
- Steps must be performed in software after interrupt completed
  - Save regs not already saved by interrupt hardware
  - Set up context for interrupt service procedure

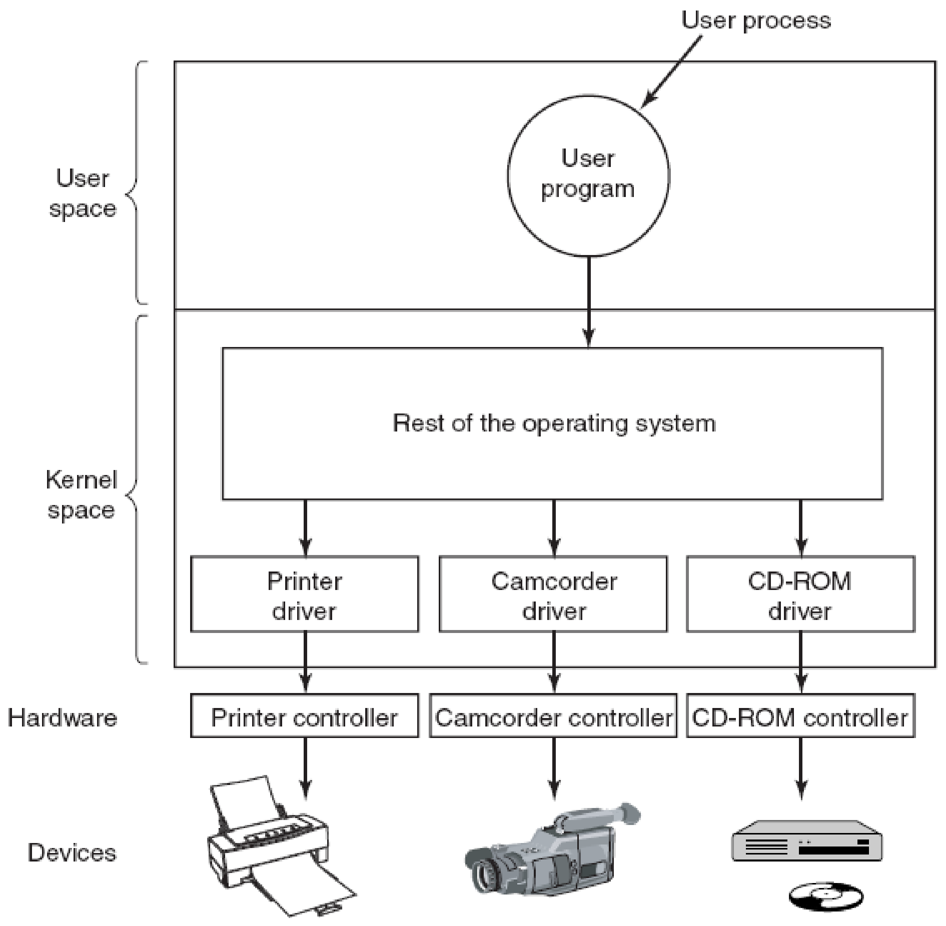
Interrupt Handlers (2)

1. Save registers not already been saved by interrupt hardware.
2. Set up a context for the interrupt service procedure.
3. Set up a stack for the interrupt service procedure.
4. Acknowledge the interrupt controller. If there is no centralized interrupt controller, reenable interrupts.
5. Copy the registers from where they were saved to the process table.

Interrupt Handlers (2)

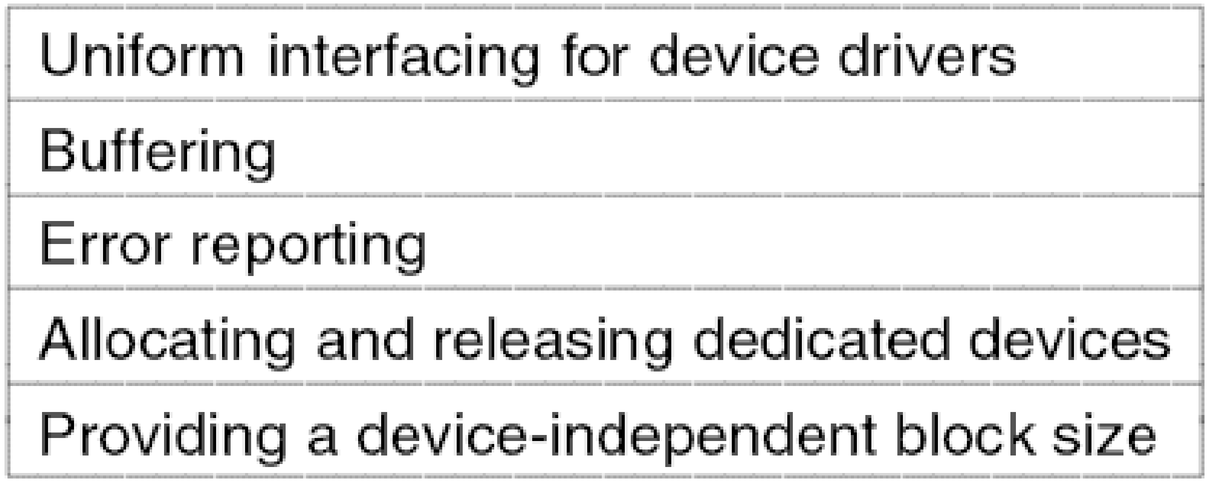
6. Run the interrupt service procedure.
7. Choose which process to run next.
8. Set up the MMU context for the process to run next.
9. Load the new process’ registers, including its PSW.
10. Start running the new process.

Device Drivers



- Logical positioning of device drivers is shown.
- Communications between drivers and device controllers goes over the bus.

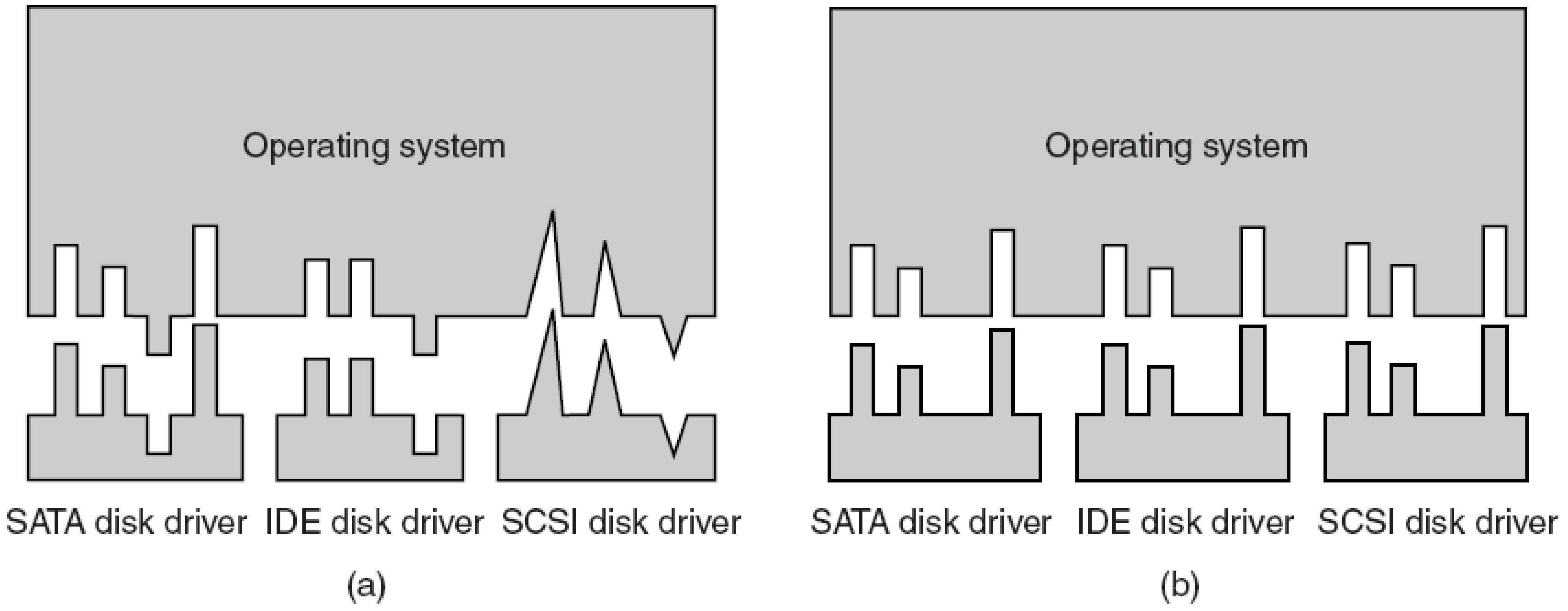
Device-Independent I/O Software



Uniform interfacing for device drivers
Buffering
Error reporting
Allocating and releasing dedicated devices
Providing a device-independent block size

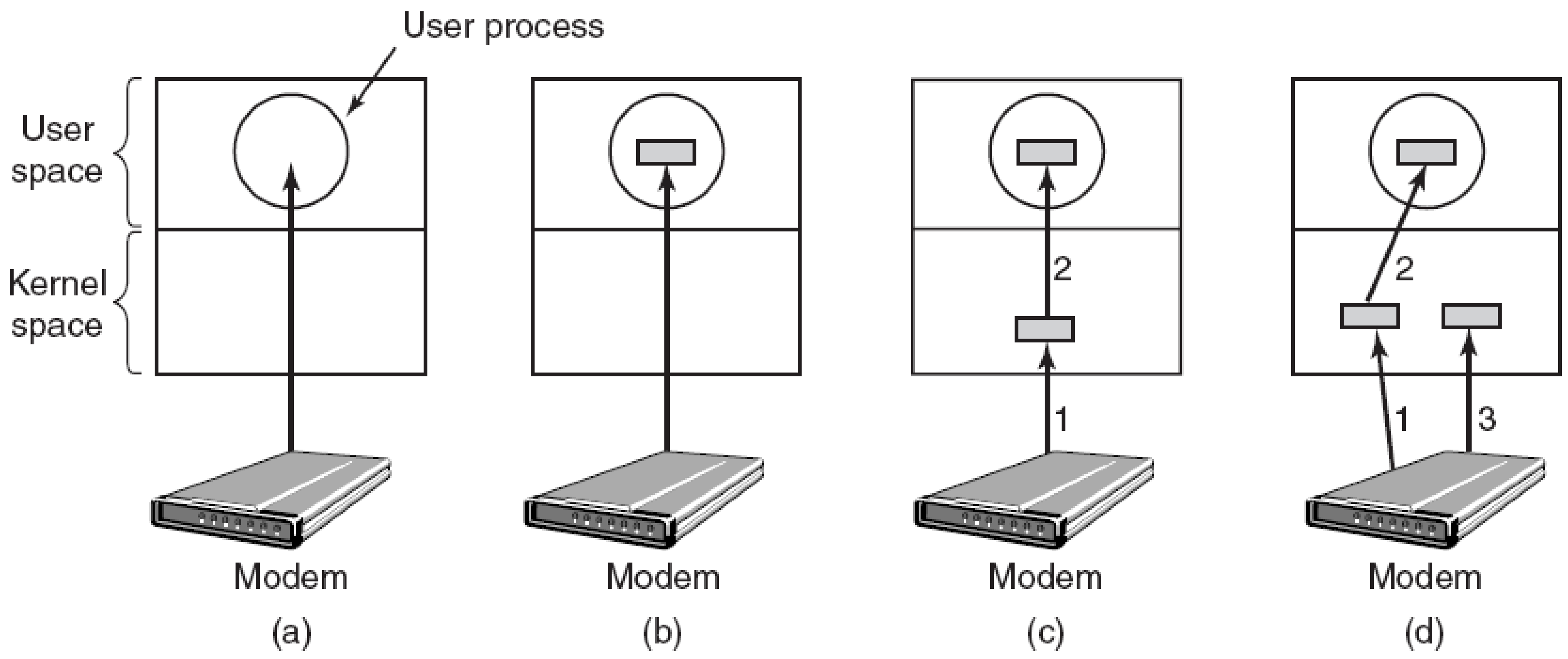
Functions of the device-independent I/O software.

Uniform Interfacing for Device Drivers



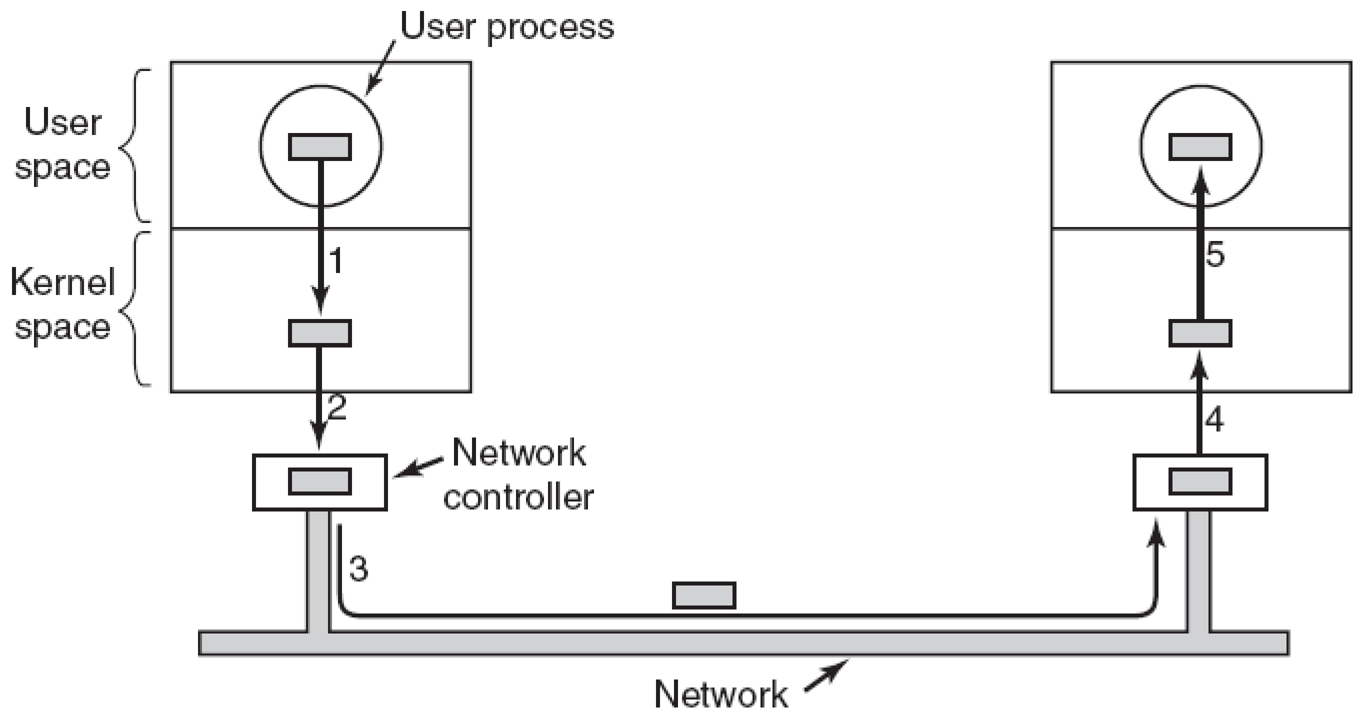
(a) Without a standard driver interface.
(b) With a standard driver interface.

Buffering (1)



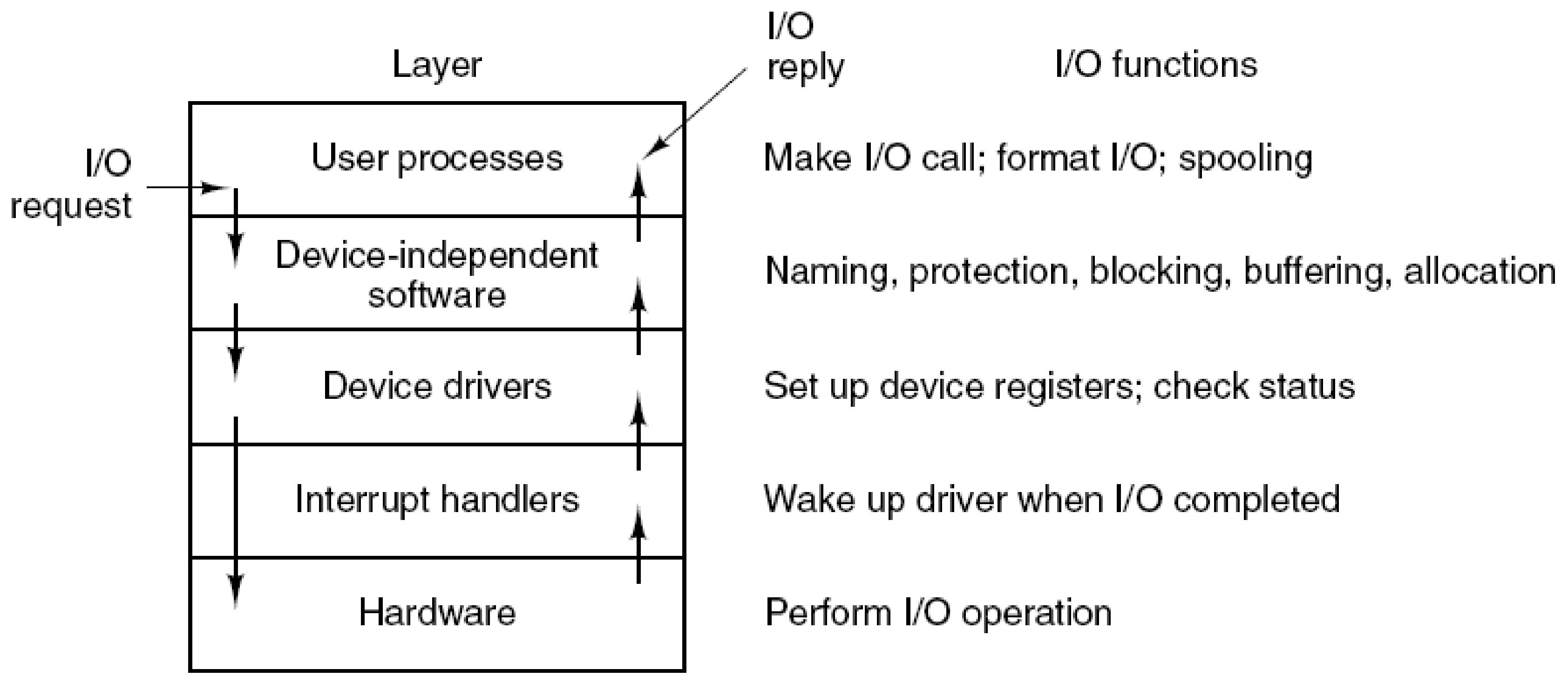
(a) Unbuffered input.
(b) Buffering in user space.
(c) Buffering in the kernel followed by copying to user space.
(d) Double buffering in the kernel.

Buffering (2)



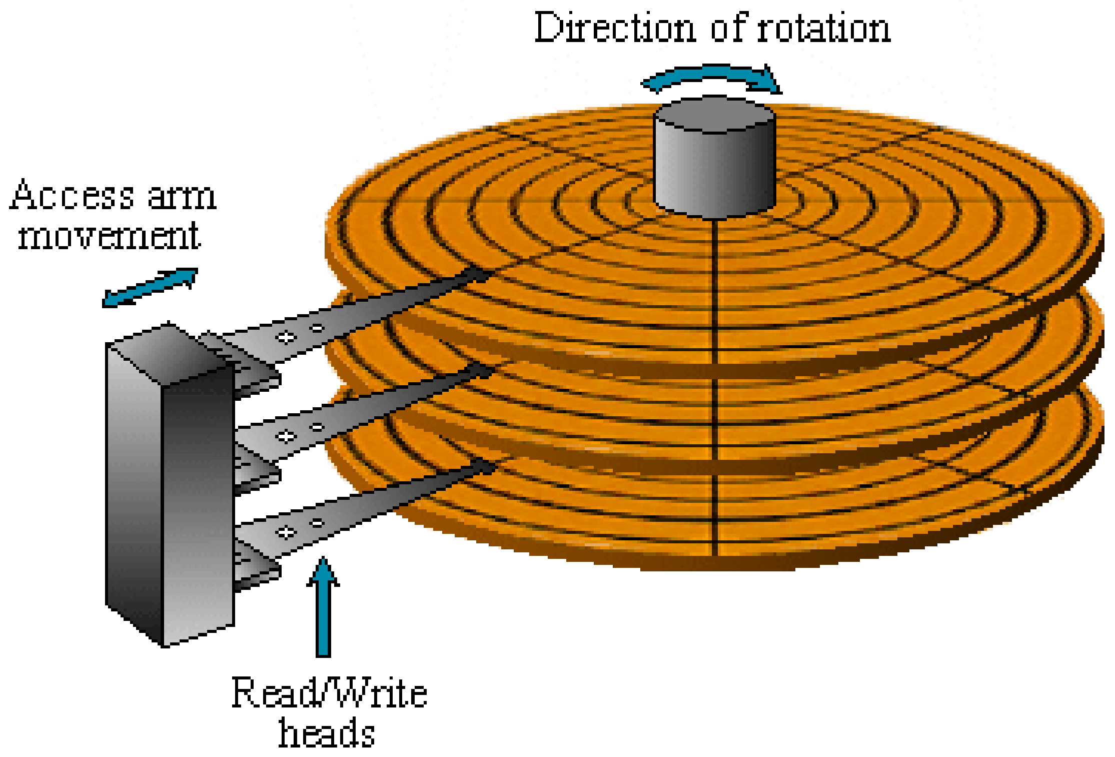
Networking may involve many copies of a packet.

User-Space I/O Software

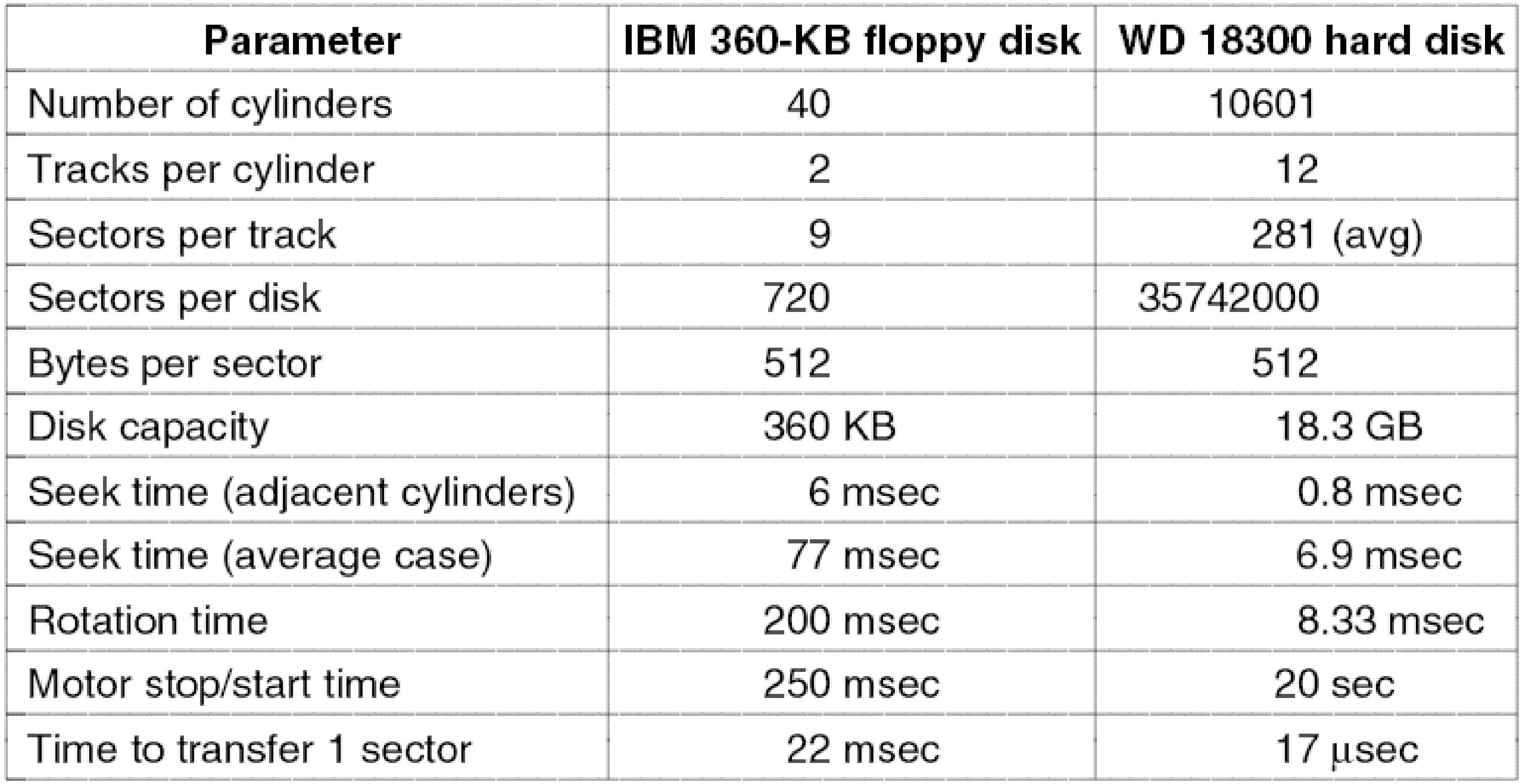

Layer
I/O reply
I/O functions
I/O request
User processes
Device-independent software
Device drivers
Interrupt handlers
Hardware
Make I/O call; format I/O; spooling
Naming, protection, blocking, buffering, allocation
Set up device registers; check status
Wake up driver when I/O completed
Perform I/O operation

Layers of the I/O system and the main functions of each layer.

Magnetic Disks Overview



Magnetic Disks (1)



Parameter	IBM 360-KB floppy disk	WD 18300 hard disk
Number of cylinders	40	10601
Tracks per cylinder	2	12
Sectors per track	9	281 (avg)
Sectors per disk	720	35742000
Bytes per sector	512	512
Disk capacity	360 KB	18.3 GB
Seek time (adjacent cylinders)	6 msec	0.8 msec
Seek time (average case)	77 msec	6.9 msec
Rotation time	200 msec	8.33 msec
Motor stop/start time	250 msec	20 sec
Time to transfer 1 sector	22 msec	17 μsec

Figure 5-18. Disk parameters for the original IBM PC 360-KB floppy disk and a Western Digital WD 18300 hard disk.

Magnetic Disks (2)

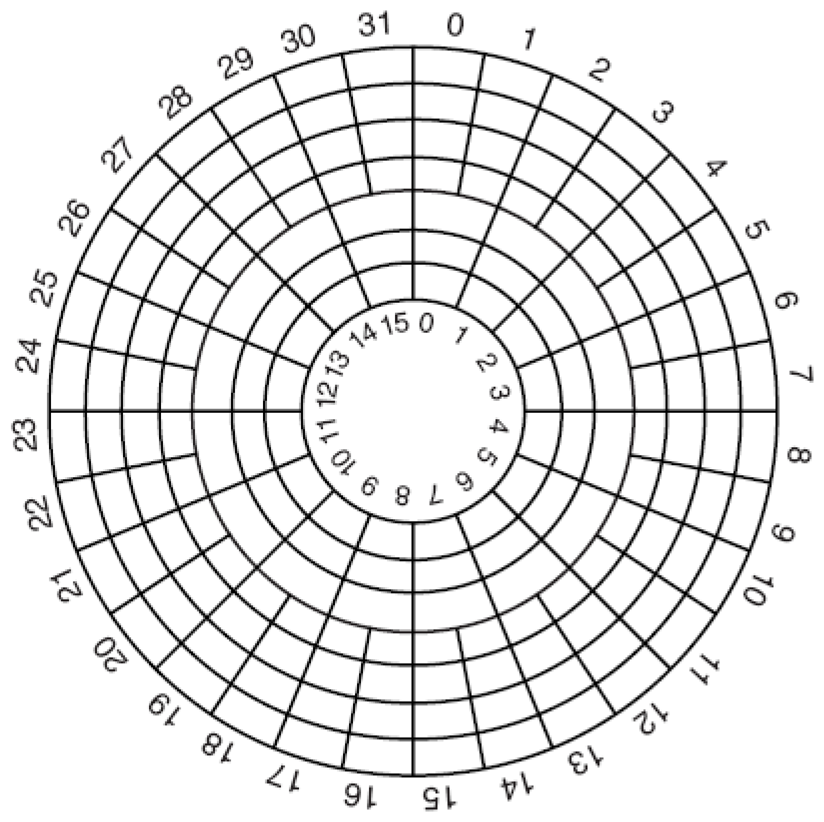
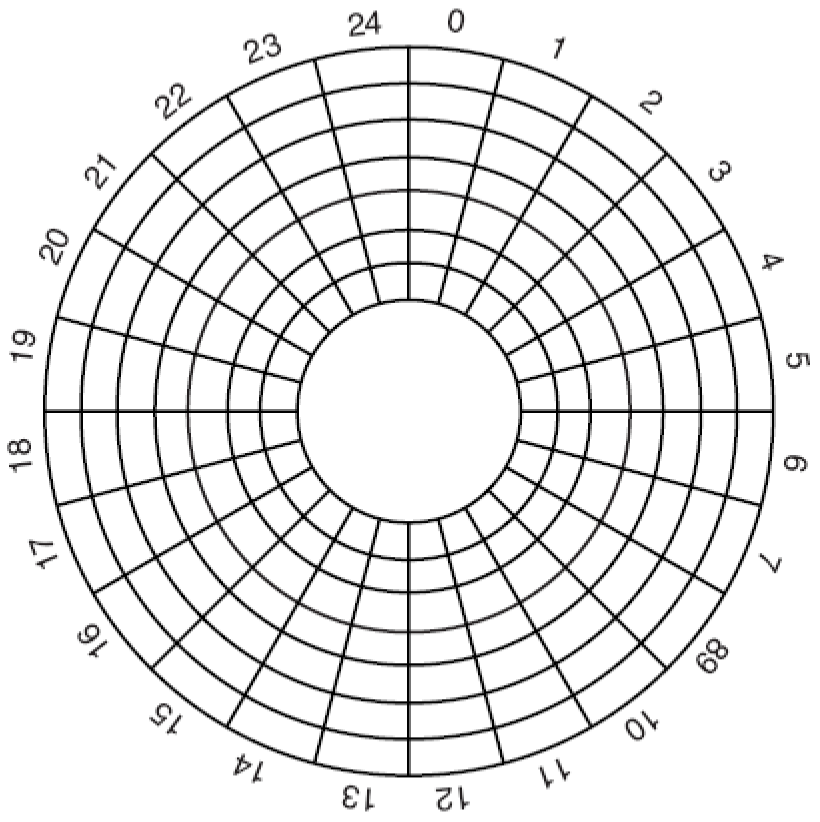
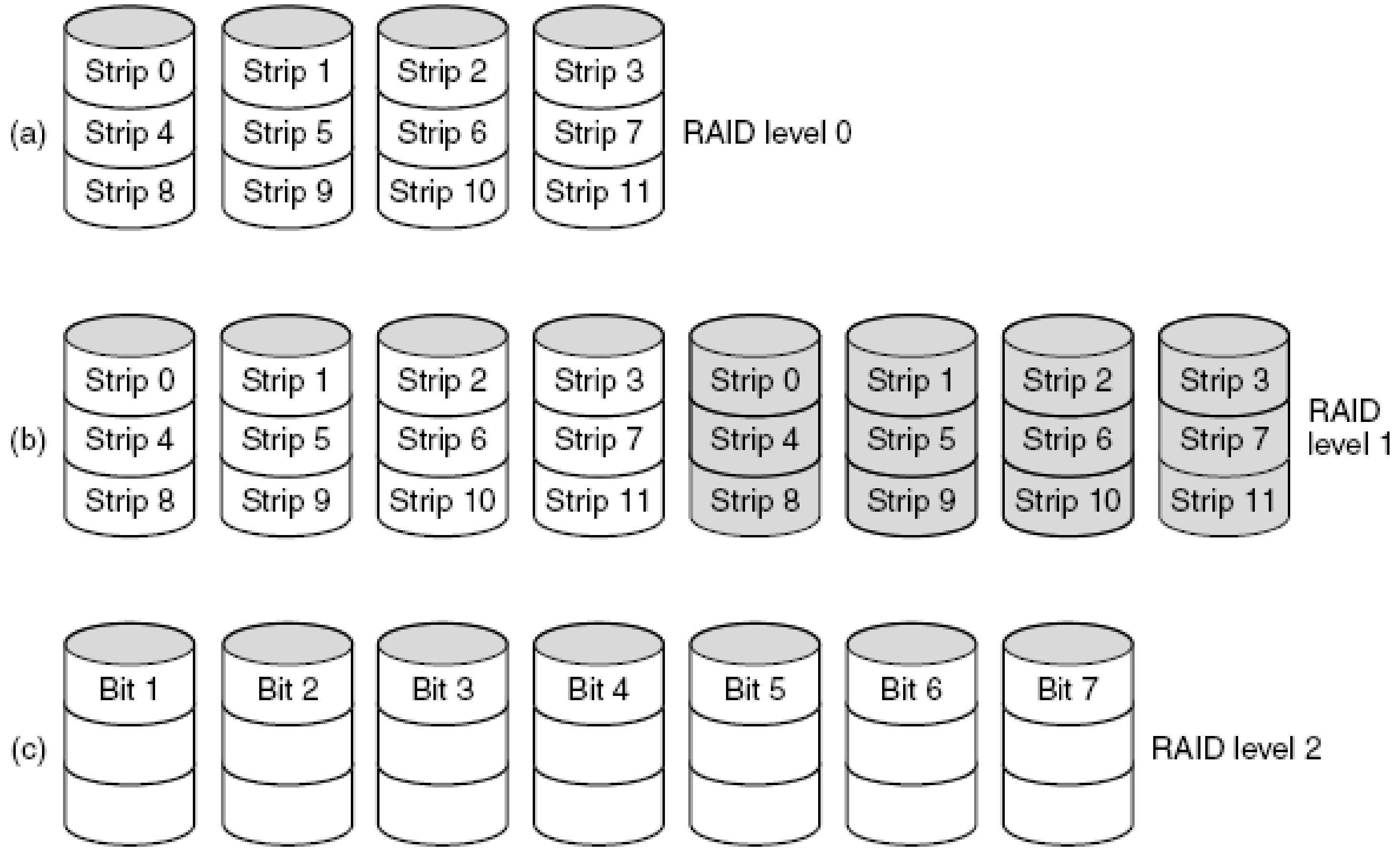



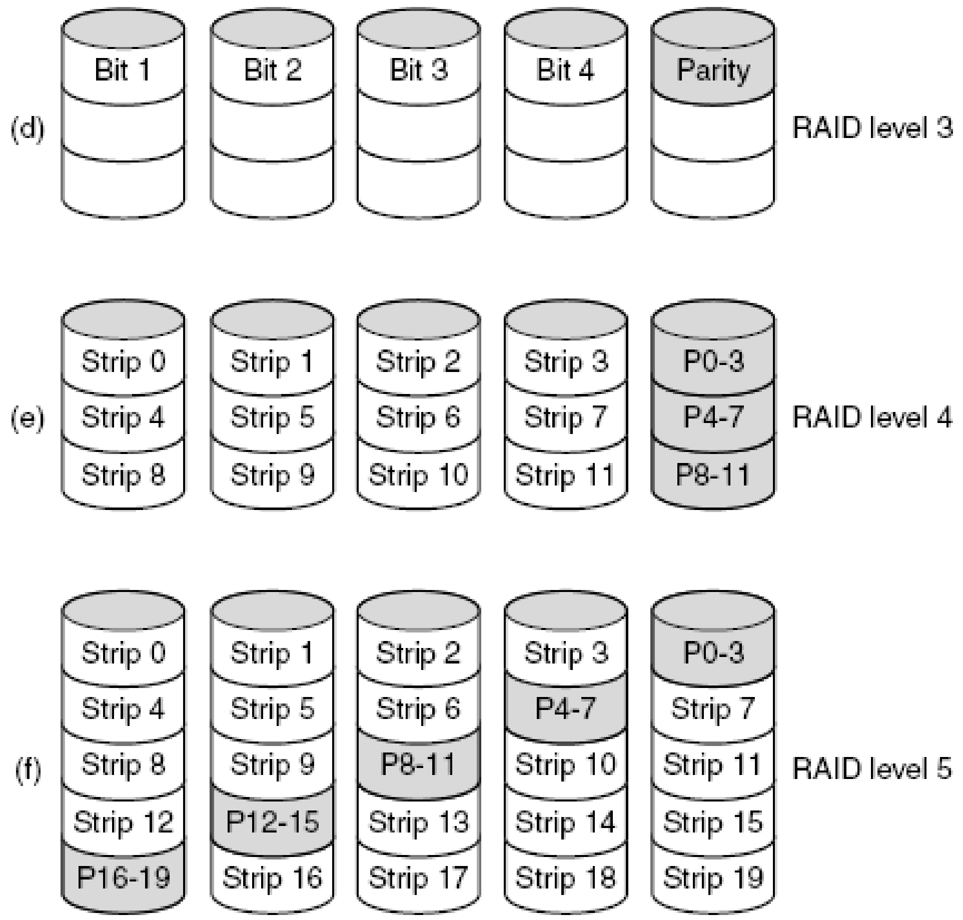
Figure 5-19. (a) Physical geometry of a disk with two zones.
(b) A possible virtual geometry for this disk.

RAID (1)



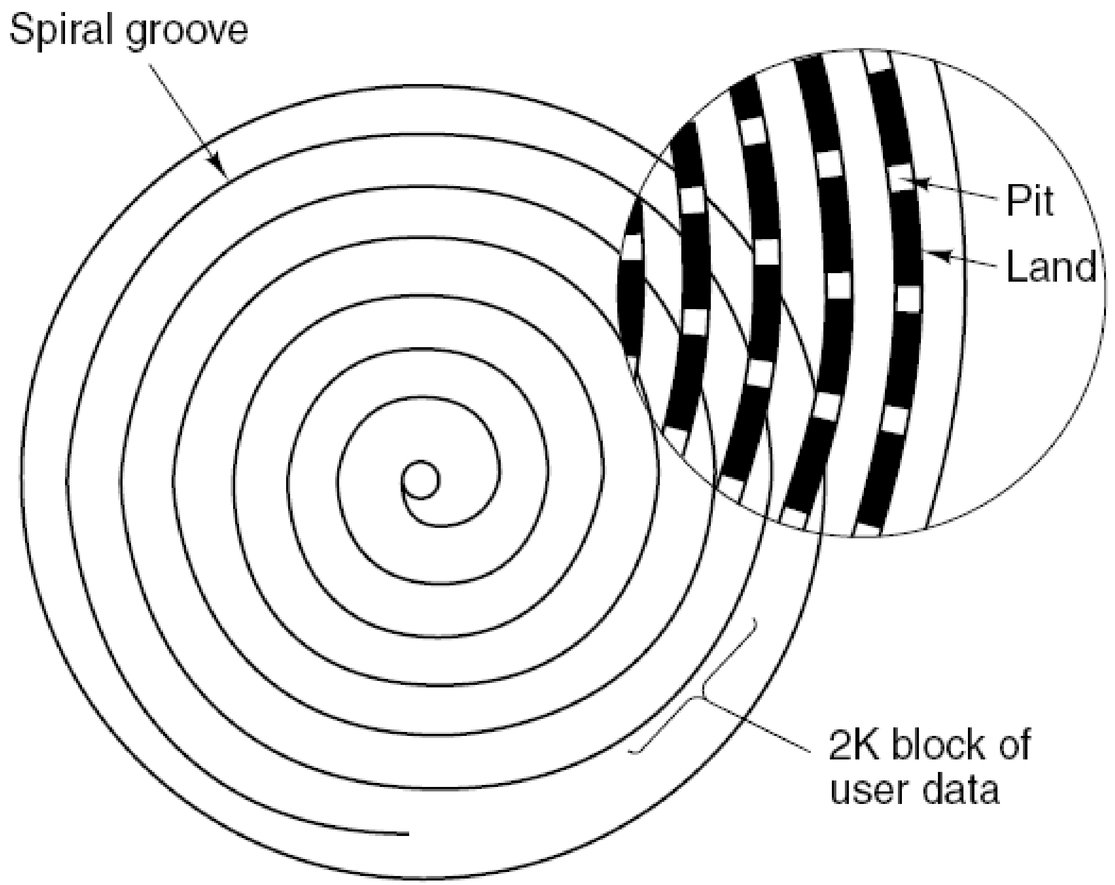
- RAID levels 0 through 2
- Backup and parity drives are shown shaded.

RAID (2)



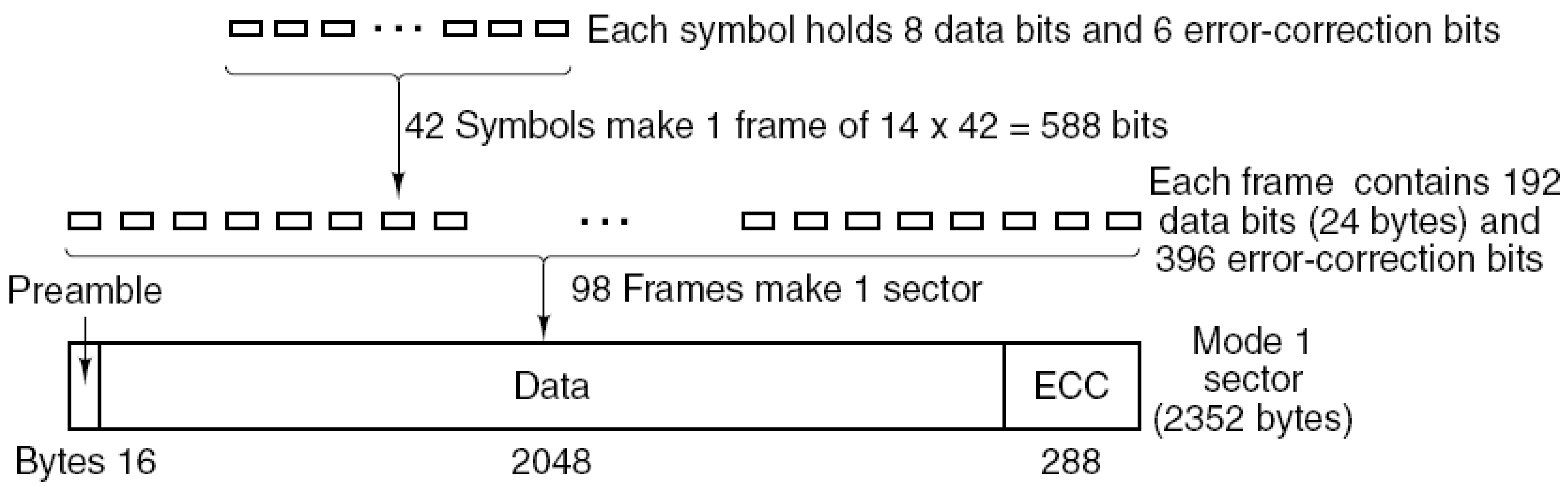
- RAID levels 2 through 5.
- Backup and parity drives are shown shaded.

CD-ROMs (1)



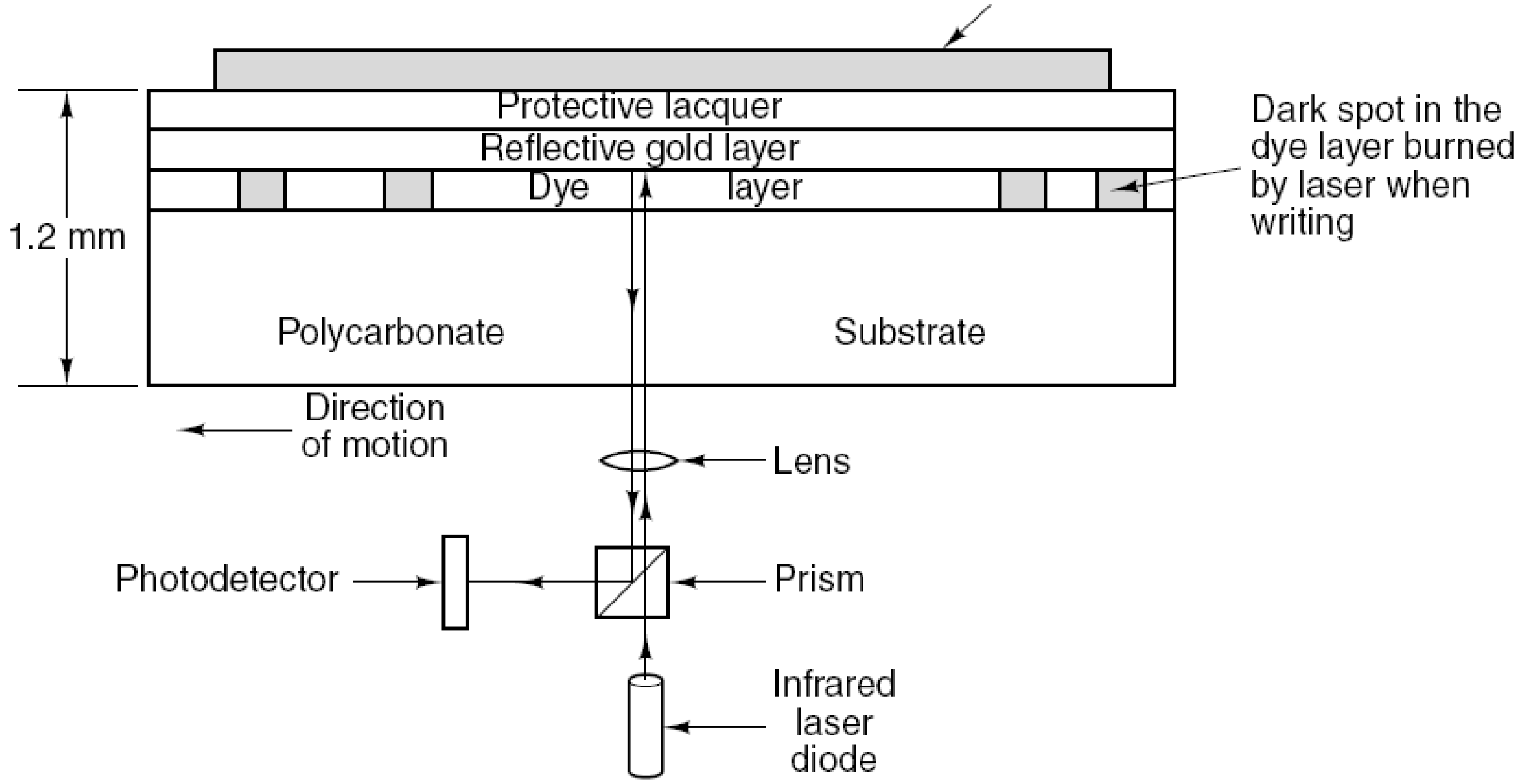
Recording structure of a compact disc or CD-ROM.

CD-ROMs (2)


Each symbol holds 8 data bits and 6 error-correction bits
42 Symbols make 1 frame of 14 x 42 = 588 bits
Each frame contains 192 data bits (24 bytes) and 396 error-correction bits
98 Frames make 1 sector
Preamble
Data
ECC
Mode 1 sector (2352 bytes)
Bytes 16
2048
288

Logical data layout on a CD-ROM.

CD-Recordables (1)



- Cross section of a CD-R disk and laser.
- A silver CD-ROM has similar structure
  - without dye layer
  - with pitted aluminum layer instead of gold layer

DVD (1)

DVD Improvements on CDs

1. Smaller pits
   (0.4 microns versus 0.8 microns for CDs).
2. A tighter spiral
   (0.74 microns between tracks versus 1.6 microns for CDs).
3. A red laser
   (at 0.65 microns versus 0.78 microns for CDs).

DVD (2)

DVD Formats

1. Single-sided, single-layer (4.7 GB).
2. Single-sided, dual-layer (8.5 GB).
3. Double-sided, single-layer (9.4 GB).
4. Double-sided, dual-layer (17 GB).

DVD (3)

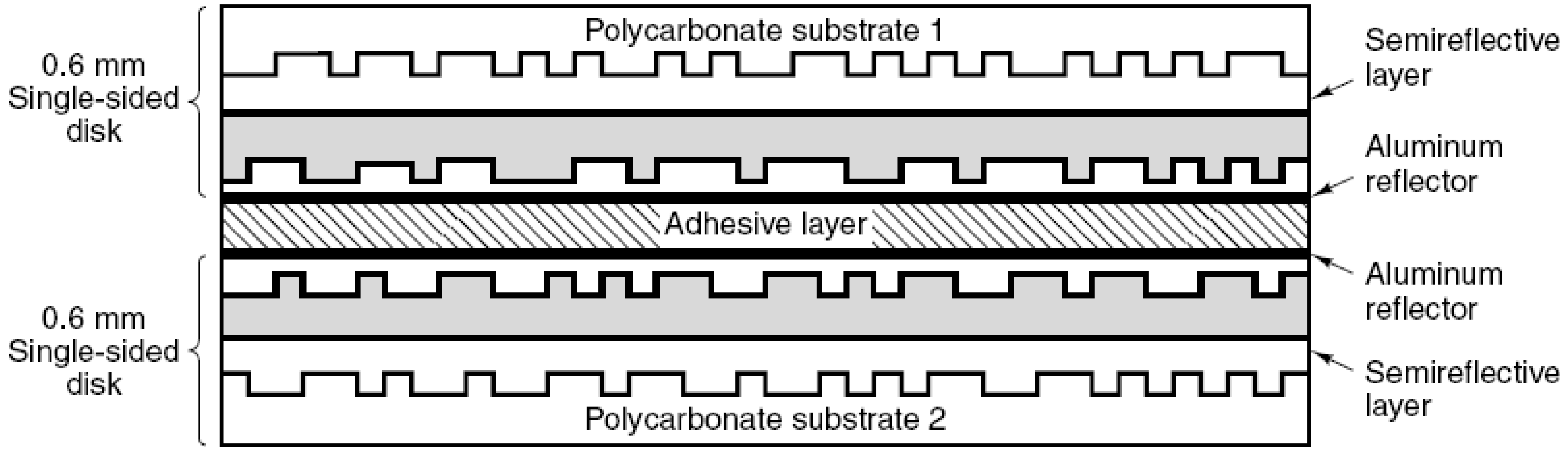
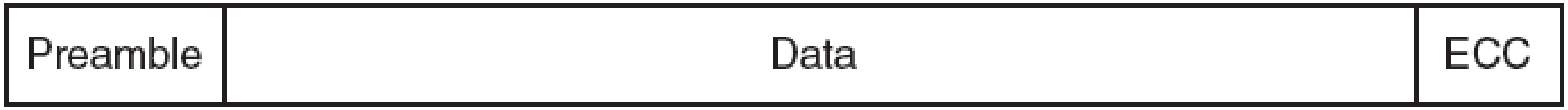


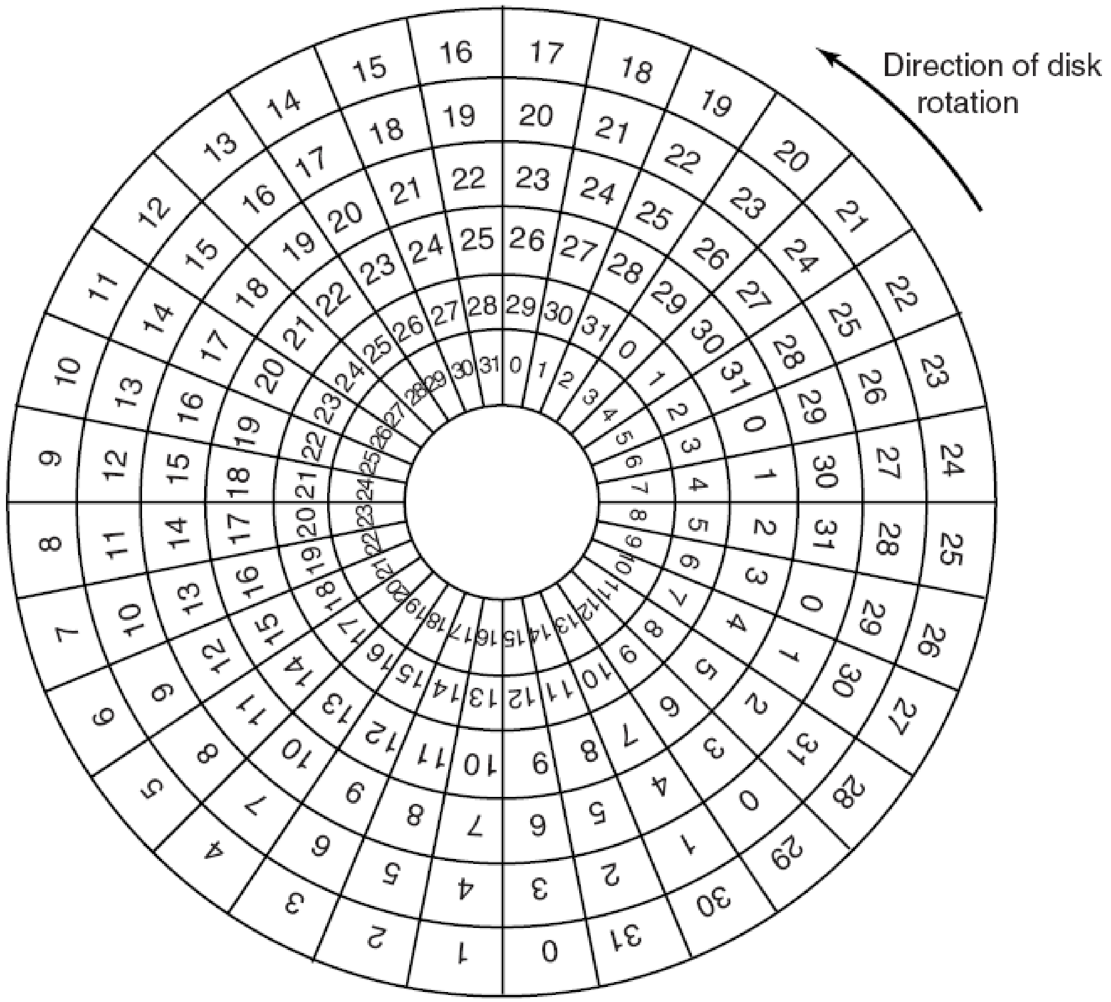
Figure 5-24. A double-sided, dual-layer DVD disk.

Disk Formatting (1)



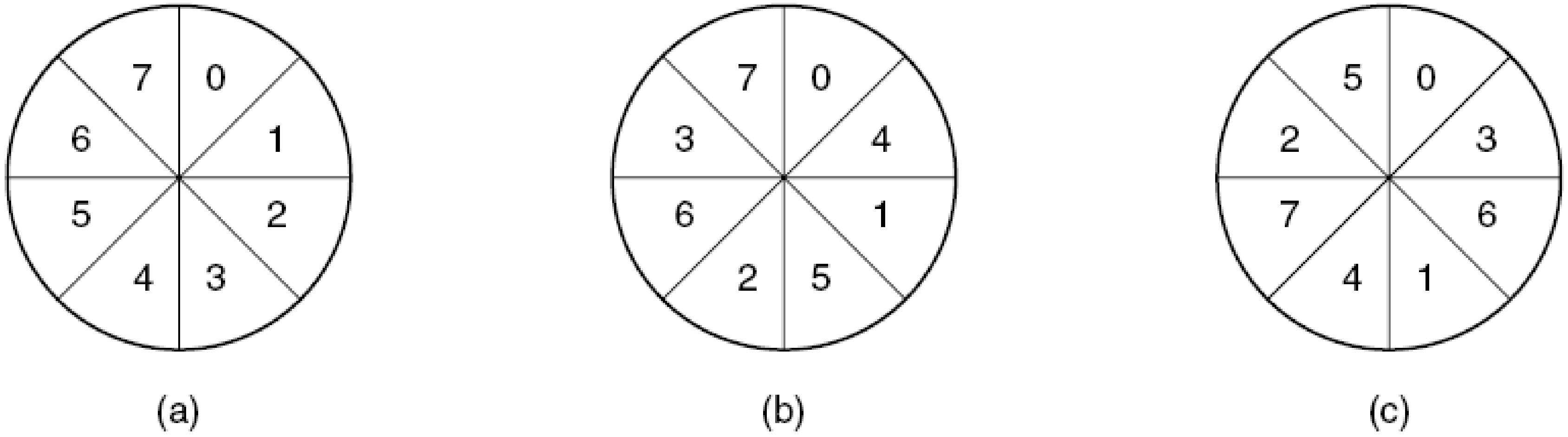
A disk sector.

Disk Formatting (2)



An illustration of cylinder skew.

Disk Formatting (3)



(a) No interleaving.
(b) Single interleaving.
(c) Double interleaving.

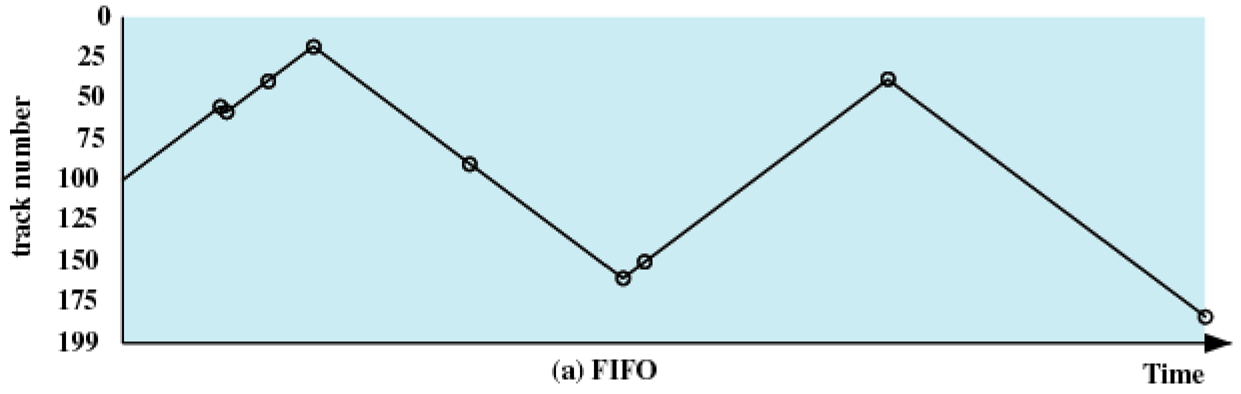
Disk Arm Scheduling Algorithms (1)

- Read/write time factors
  1. Seek time
  2. Rotational delay
  3. Actual data transfer time.
- Seek time dominates
- Error checking is done by controllers

Disk Arm Scheduling Algorithms (2)

- First-in, first-out (FIFO)
  - Process request sequentially
  - Fair to all processes
  - Approaches random scheduling in performance if there are many processes

Request order : 55 58 39 18 90 160 150 38 184

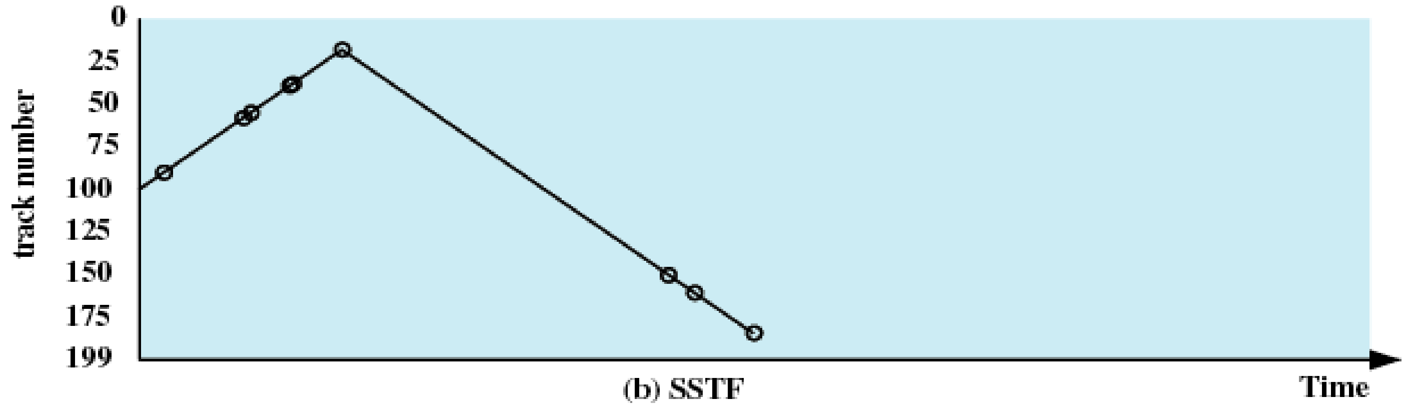


(a) FIFO

Disk Arm Scheduling Algorithms (3)

- Shortest Service Time First
  - Select the disk I/O request that requires the least movement of the disk arm from its current position
  - Always choose the minimum Seek time

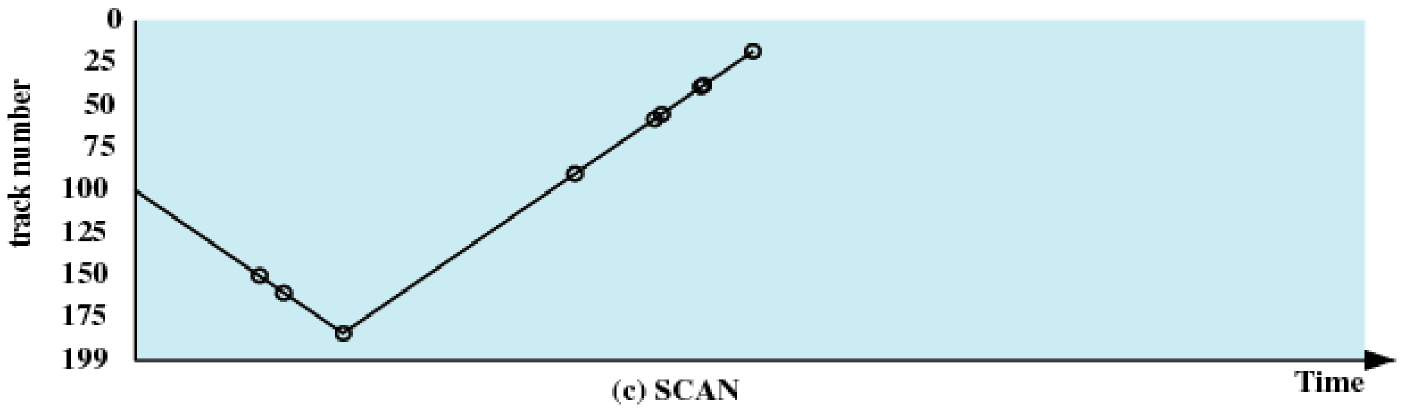
Request order : 55 58 39 18 90 160 150 38 184



Disk Arm Scheduling Algorithms (4)

- SCAN or elevator algorithm
  - Arm moves in one direction only, satisfying all outstanding requests until it reaches the last track in that direction
  - Direction is reversed

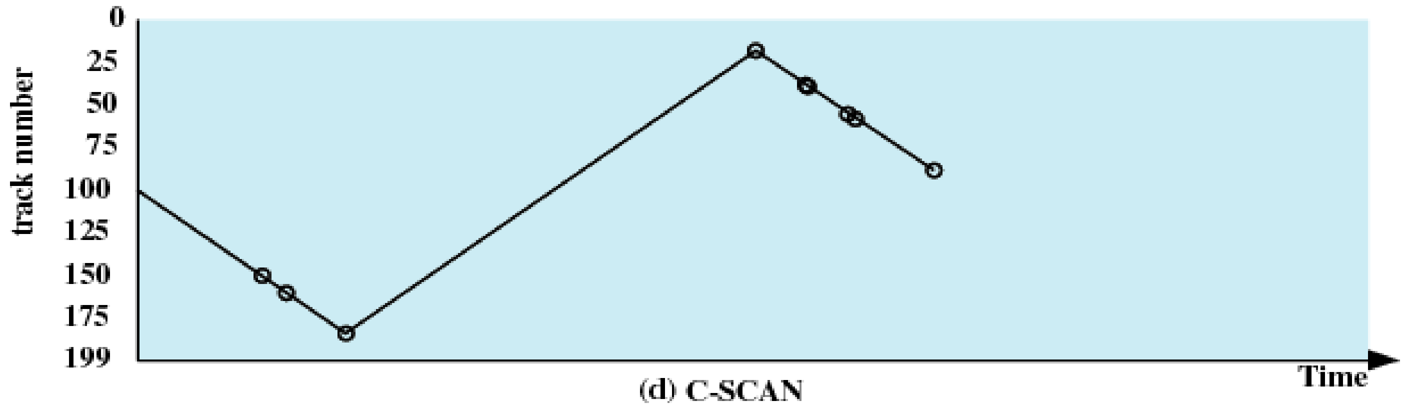
Request order : 55 58 39 18 90 160 150 38 184



Disk Arm Scheduling Algorithms (5)

- C-SCAN
  - Restricts scanning to one direction only
  - When the last track has been visited in one direction, the arm is returned to the opposite end of the disk and the scan begins again

Request order : 55 58 39 18 90 160 150 38 184

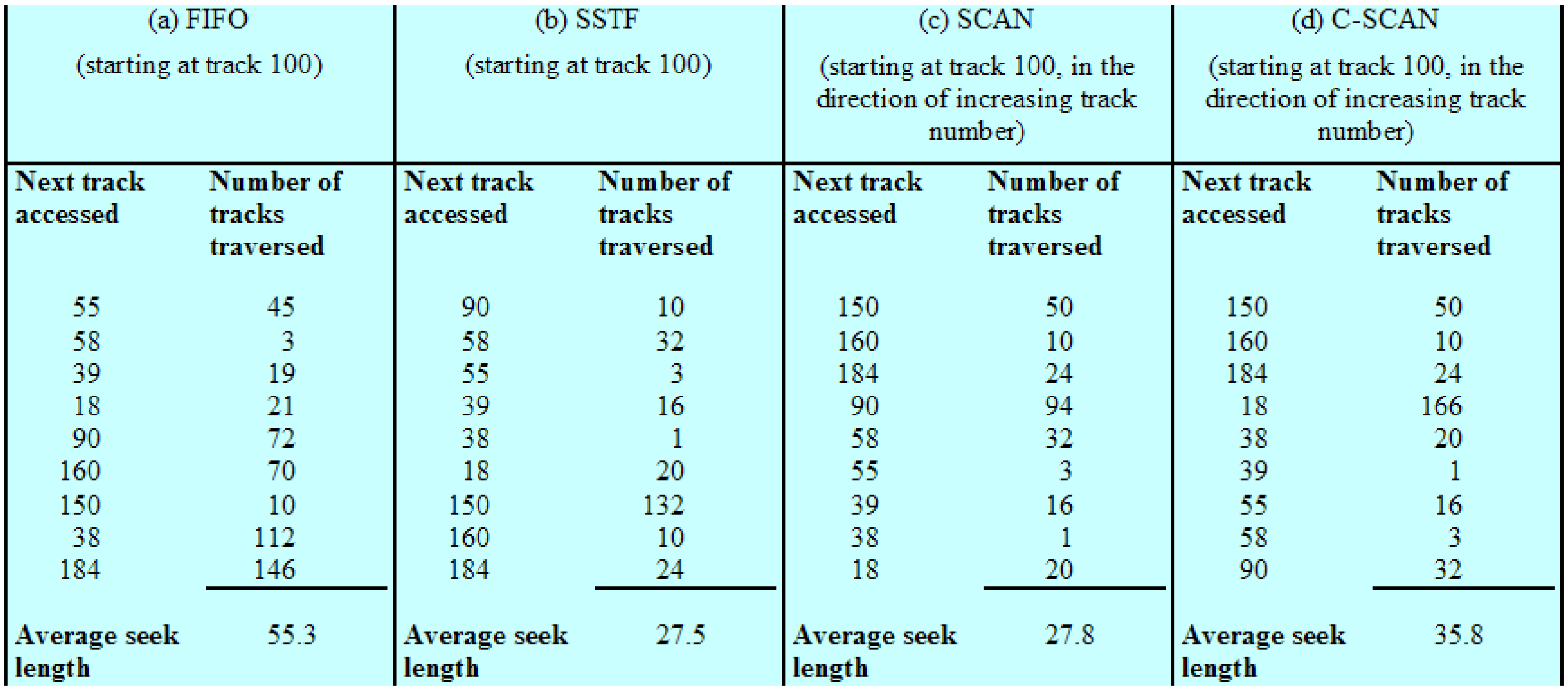


Disk Arm Scheduling Algorithms (6)

- N-step-SCAN
  - Segments the disk request queue into subqueues of length N
  - Subqueues are processed one at a time, using SCAN
  - New requests added to other queue when queue is processed
- FSCAN
  - Two queues
  - One queue is empty for new requests

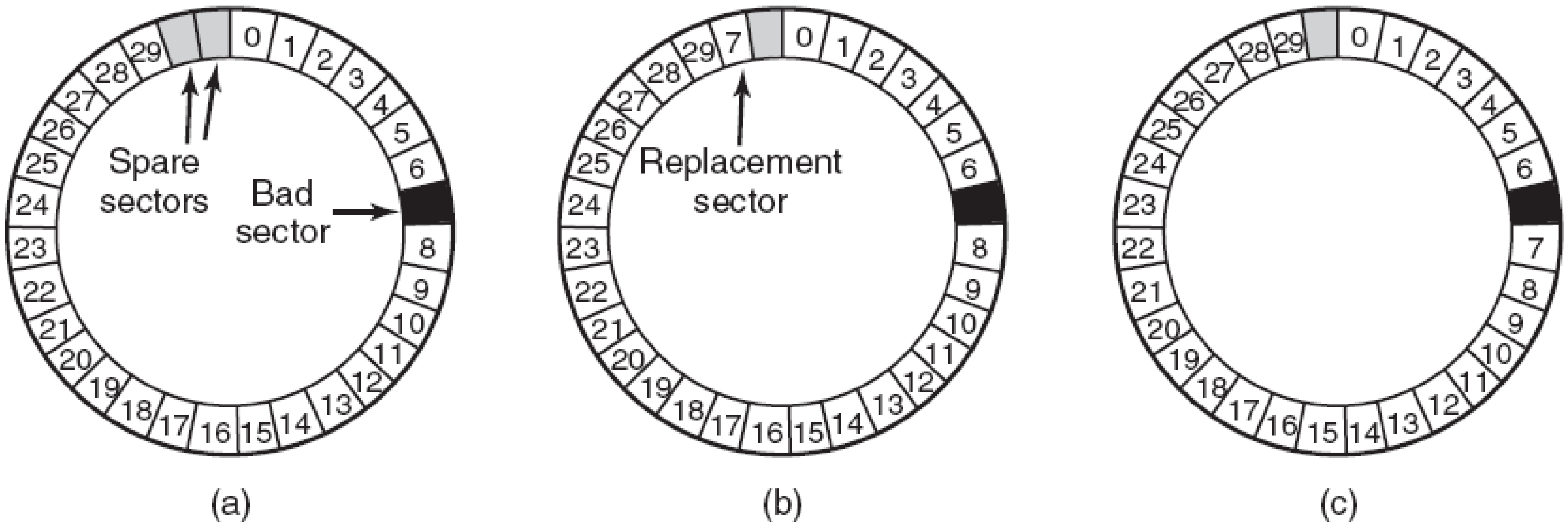
Comparison of disk arm scheduling algorithms

Table 11.2 Comparison of Disk Scheduling Algorithms



(a) FIFO (starting at track 100)		(b) SSTF (starting at track 100)		(c) SCAN (starting at track 100, in the direction of increasing track number)		(d) C-SCAN (starting at track 100, in the direction of increasing track number)	
Next track accessed	**Number of tracks traversed**	**Next track accessed**	**Number of tracks traversed**	**Next track accessed**	**Number of tracks traversed**	**Next track accessed**	**Number of tracks traversed**
55	45	90	10	150	50	150	50
58	3	58	32	160	10	160	10
39	19	55	3	184	24	184	24
18	21	39	16	90	94	18	166
90	72	38	1	58	32	38	20
160	70	18	20	55	3	39	1
150	10	150	132	39	16	55	16
38	112	160	10	38	1	58	3
184	146	184	24	18	20	90	32
Average seek length	55.3	**Average seek length**	27.5	**Average seek length**	27.8	**Average seek length**	35.8

Error Handling



(a) A disk track with a bad sector.

(b) Substituting a spare for the bad sector.

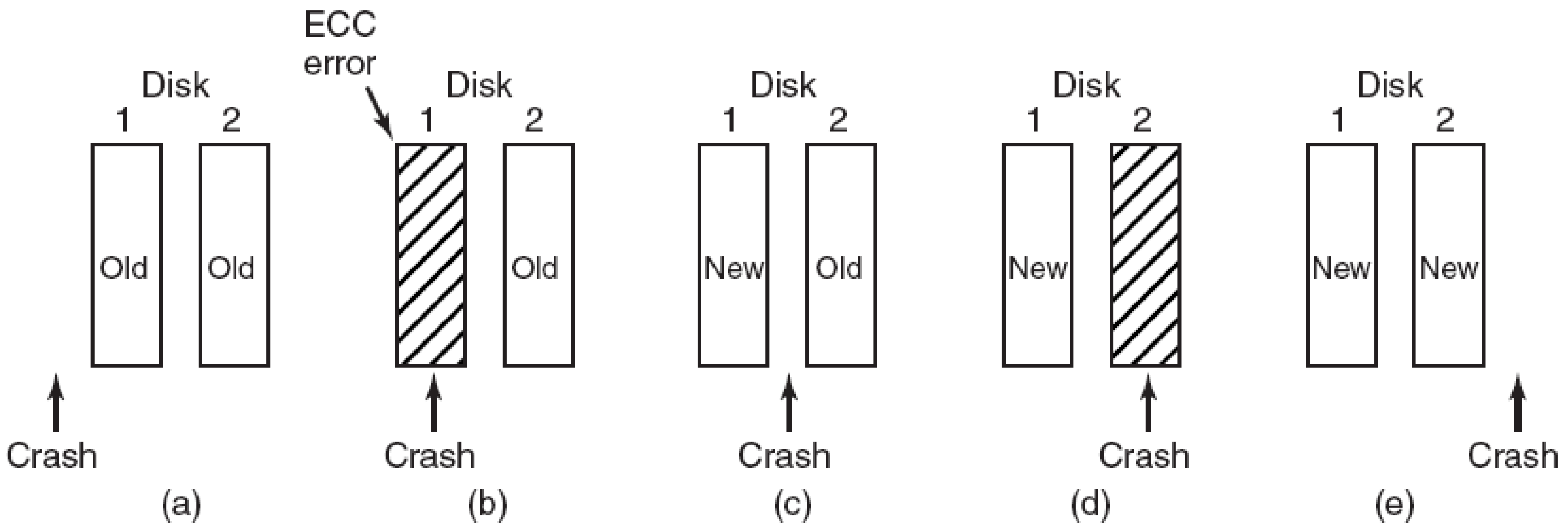
(c) Shifting all the sectors to bypass the bad one.

Stable Storage (1)

Operations for stable storage using identical disks:

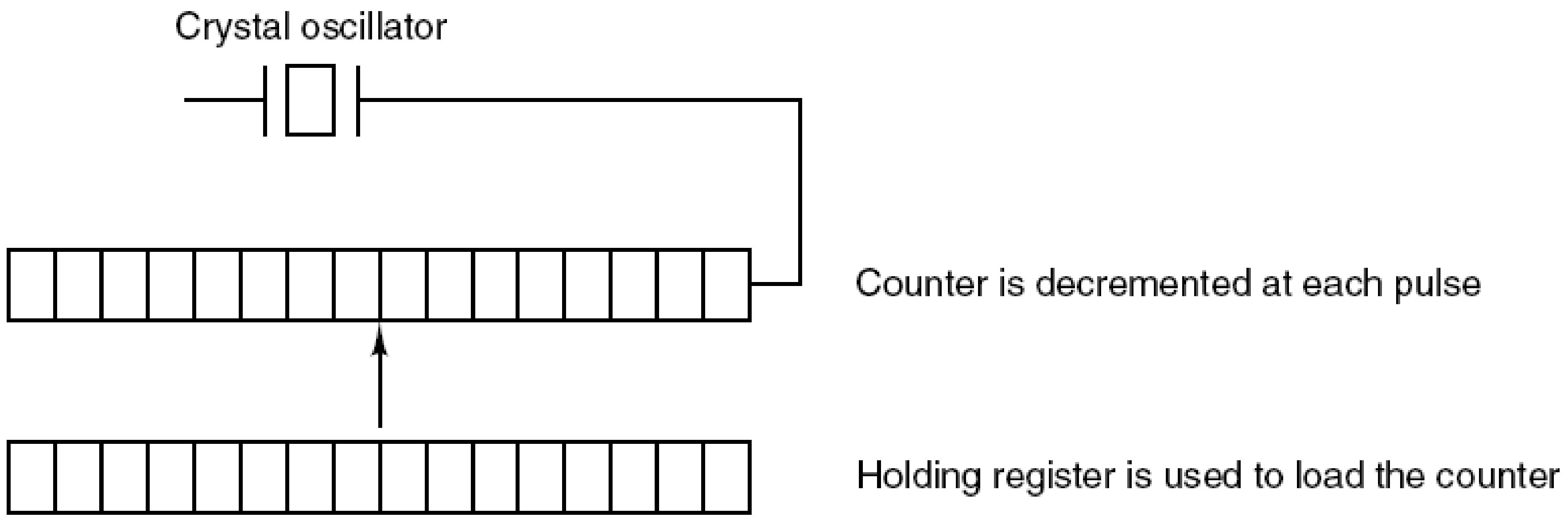
1. Stable writes
2. Stable reads
3. Crash recovery

Stable Storage (2)



Analysis of the influence of crashes on stable writes.

Clock Hardware


Crystal oscillator
Counter is decremented at each pulse
Holding register is used to load the counter

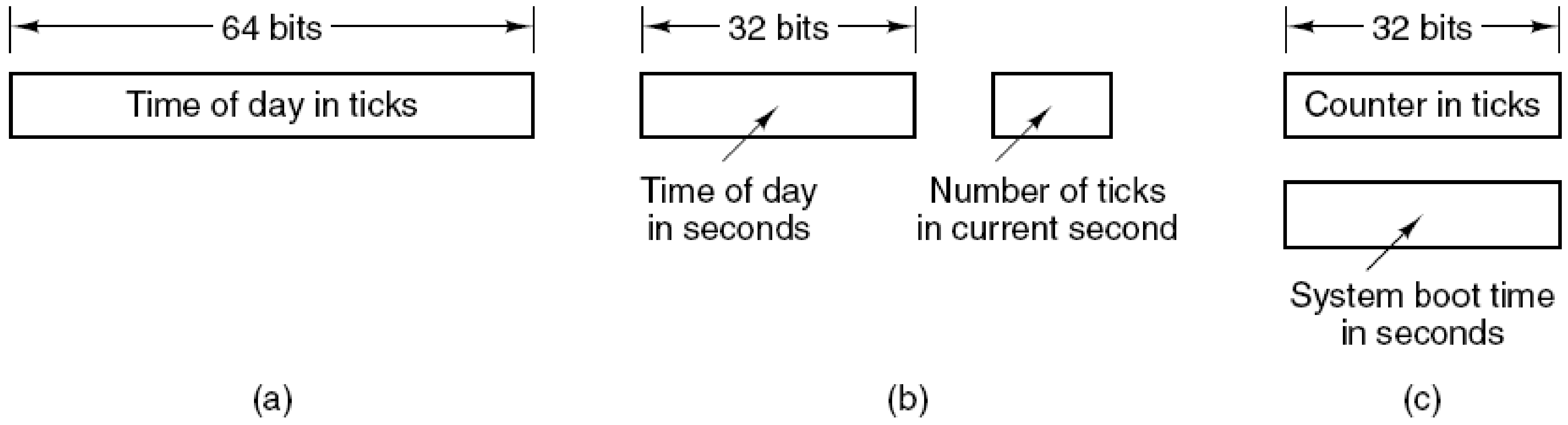
A programmable clock.

Clock Software (1)

Typical duties of a clock driver

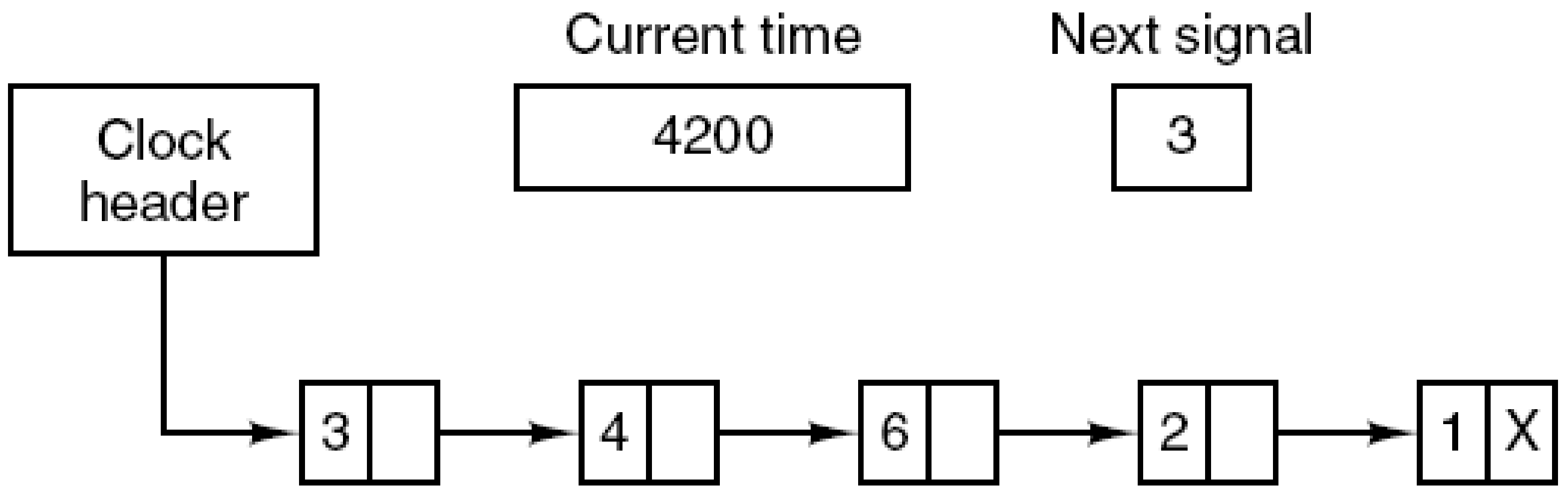
1. Maintaining the time of day.
2. Preventing processes from running longer than they are allowed to.
3. Accounting for CPU usage.
4. Handling alarm system call made by user processes.
5. Providing watchdog timers for parts of the system itself.
6. Doing profiling, monitoring, statistics gathering.

Clock Software (2)



Three ways to maintain the time of day.

Clock Software (3)

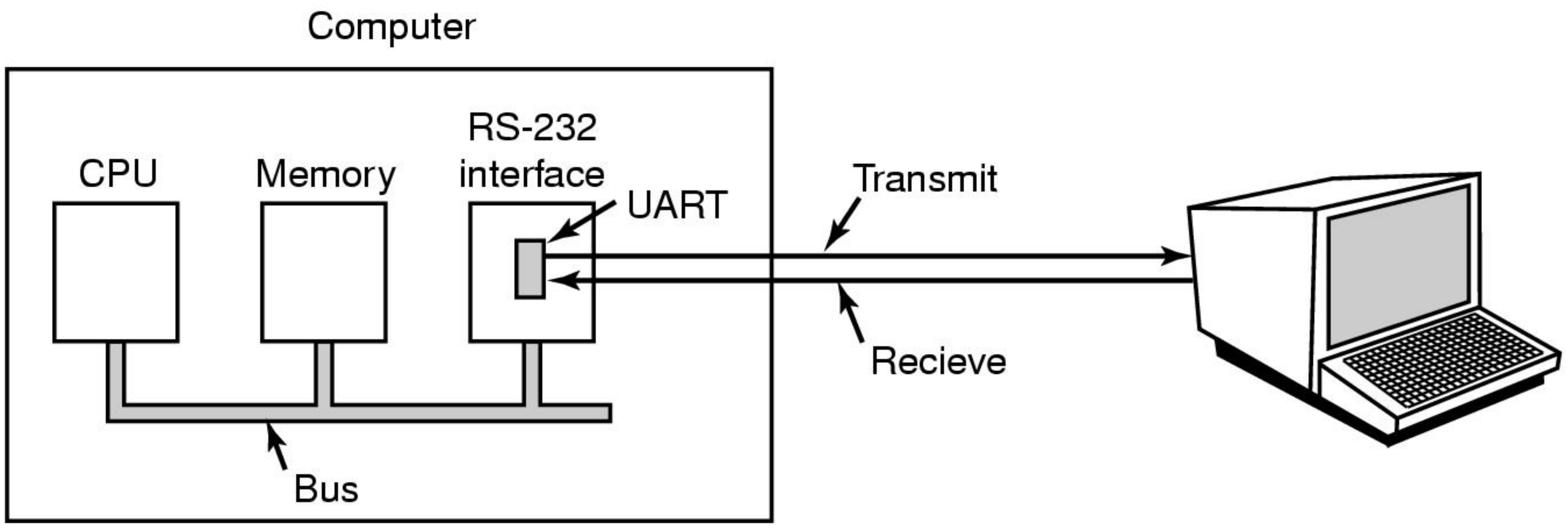


Simulating multiple timers with a single clock.

Soft Timers

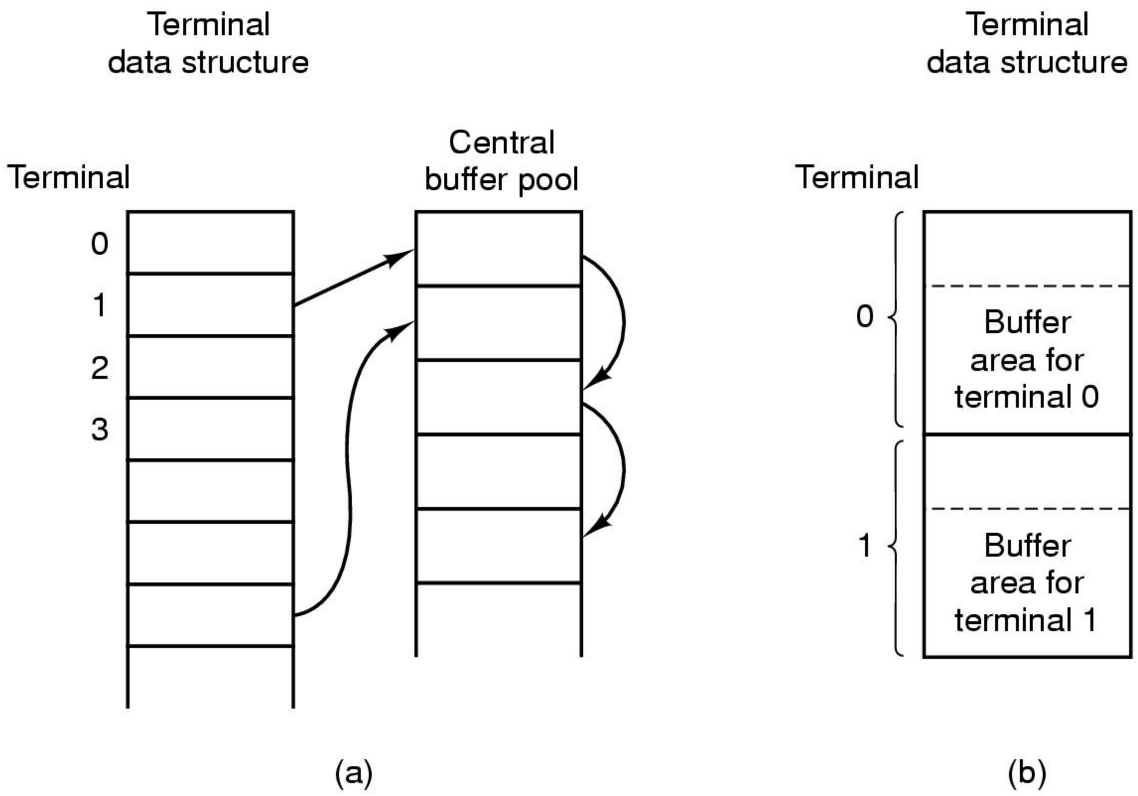
- Main system timer is used by OS
- Second clock may be used by applications to generate timer interrupts
  - No problems if interrupt frequency is low
- Soft timers avoid interrupts
  - Kernel check for soft timer expiration before it exits to user mode
  - How well this works depends on rate of kernel entries

Character Oriented Terminals
RS-232 Terminal Hareware



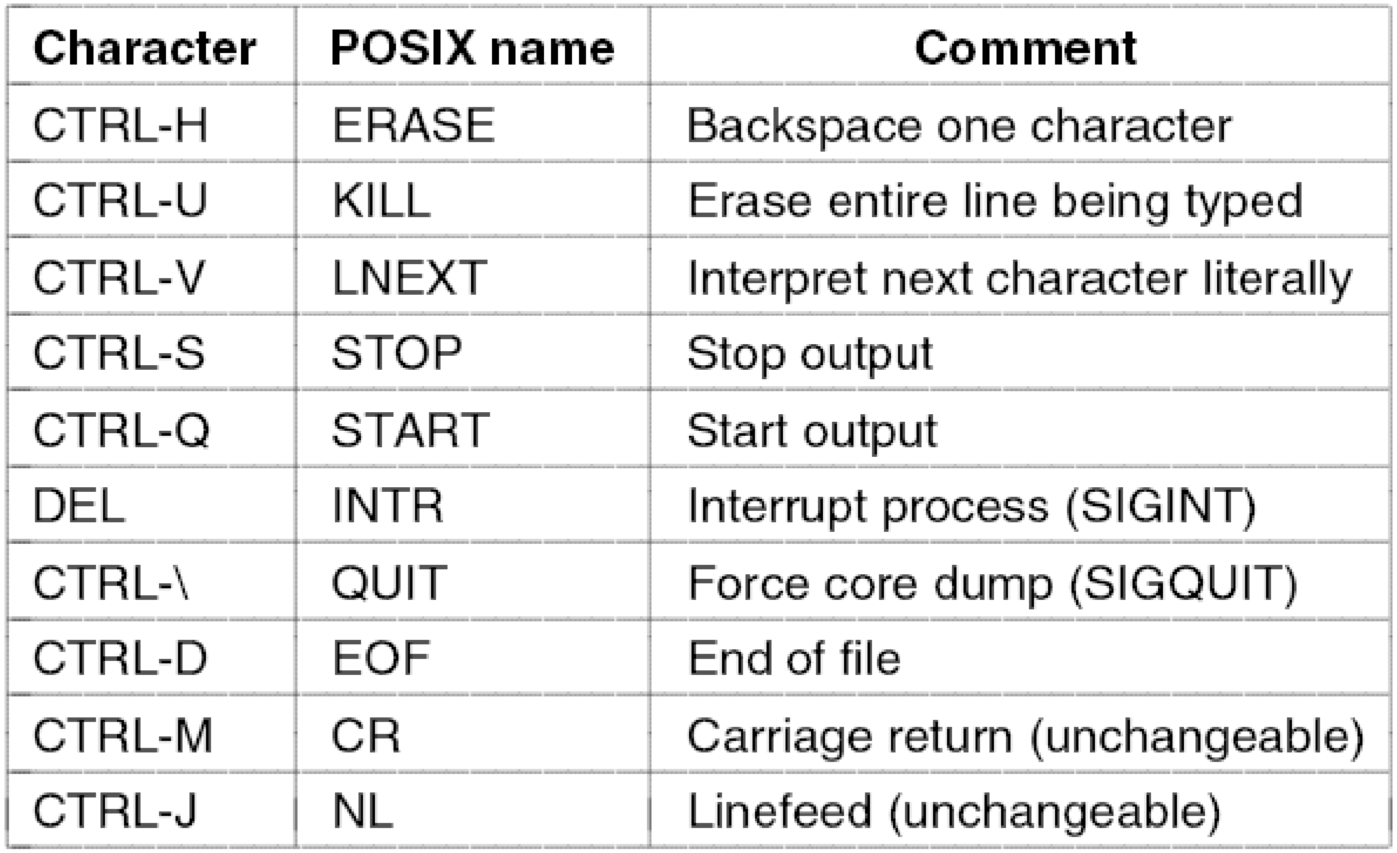
- An RS-232 terminal communicates with computer 1 bit at a time
- Called a serial line – bits go out in series, 1 bit at a time
- Windows uses COM1 and COM2 ports, first to serial lines
- Computer and terminal are completely independent

Keyboard Software (1)



- Central buffer pool
- Dedicated buffer for each terminal

Keyboard Software (2)



Character	POSIX name	Comment
CTRL-H	ERASE	Backspace one character
CTRL-U	KILL	Erase entire line being typed
CTRL-V	LNEXT	Interpret next character literally
CTRL-S	STOP	Stop output
CTRL-Q	START	Start output
DEL	INTR	Interrupt process (SIGINT)
CTRL-\	QUIT	Force core dump (SIGQUIT)
CTRL-D	EOF	End of file
CTRL-M	CR	Carriage return (unchangeable)
CTRL-J	NL	Linefeed (unchangeable)

Characters that are handled specially in canonical mode.

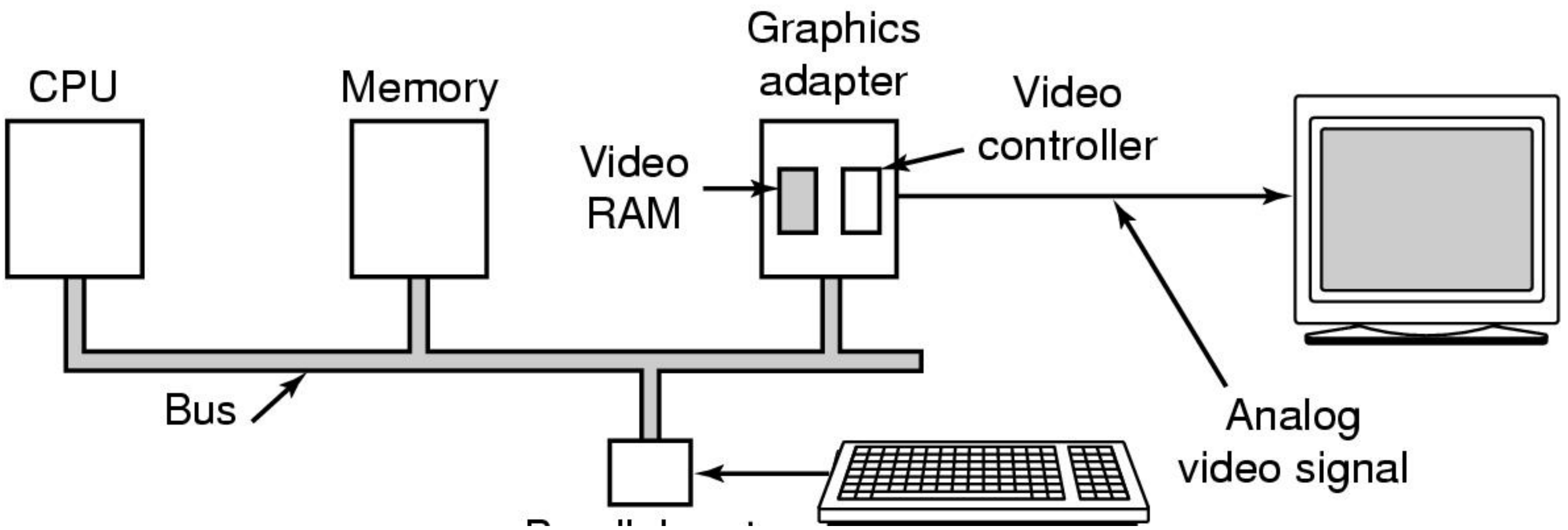
Keyboard Software (3)

Escape sequence	Meaning
ESC [*n* A	Move up *n* lines
ESC [*n* B	Move down *n* lines
ESC [*n* C	Move right *n* spaces
ESC [*n* D	Move left *n* spaces
ESC [*m* ; *n* H	Move cursor to (*m*,*n*)
ESC [*s* J	Clear screen from cursor (0 to end, 1 1from start, 2 all)
ESC [*s* K	Clear line from cursor (0 to end, 1 from start, 2 all)
ESC [*n* L	Insert *n* lines at cursor
ESC [*n* M	Delete *n* lines at cursor
ESC [*n* P	Delete *n* chars at cursor
ESC [*n* @	Insert *n* chars at cursor
ESC [*n* m	Enable rendition *n* (0=normal, 4=bold, 5=blinking, 7=reverse)
ESC M	Scroll the screen backward if the cursor is on the top line

The ANSI escape sequences accepted by the terminal driver on output.

- ESC denotes the ASCII escape character (0x1B)
- *n*, *m*, and *s* are optional numeric parameters.

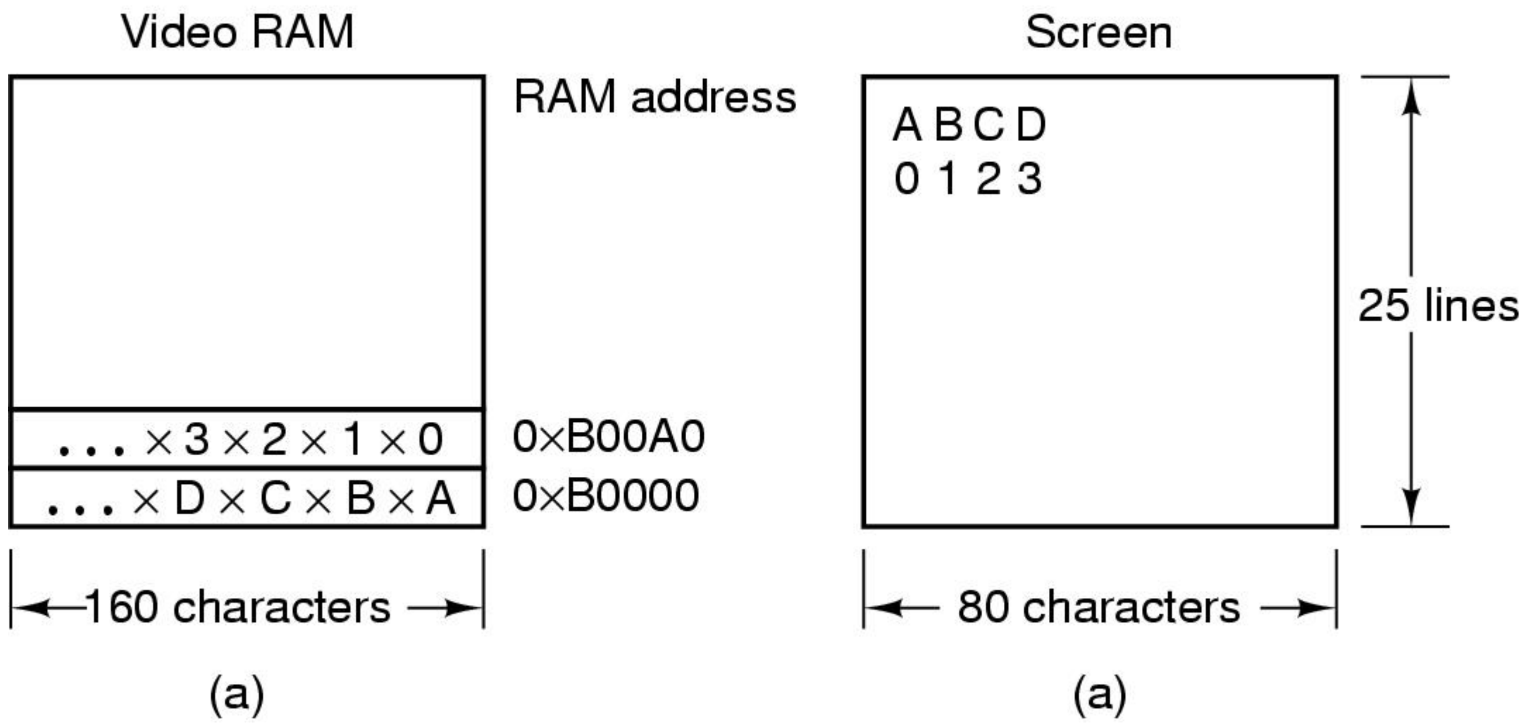
Display Hardware (1)



Memory-mapped displays

- Driver writes directly into display's video RAM

Display Hardware (1)



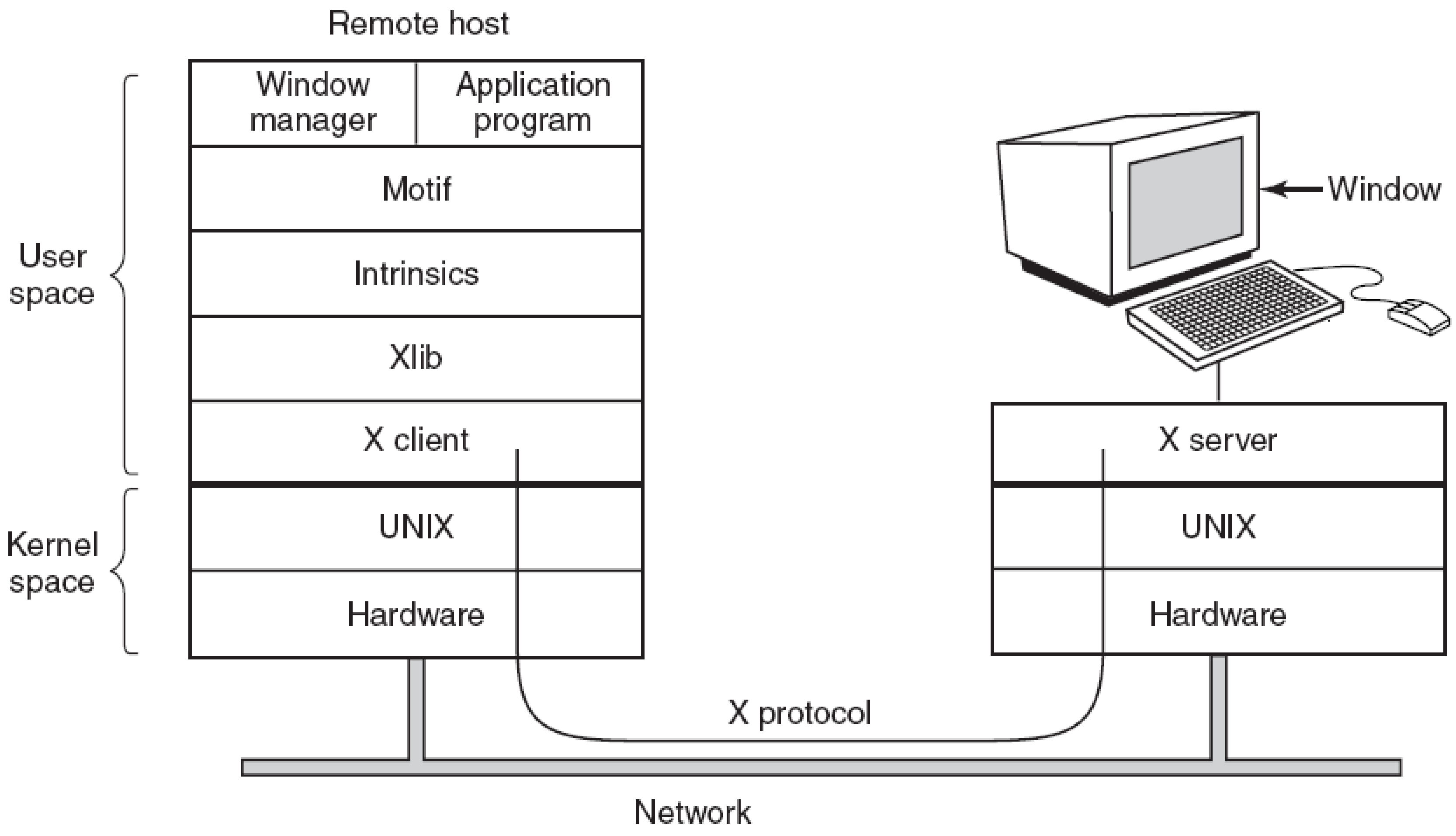
  - simple monochrome display
  - character mode
- Corresponding screen
  - the Xs are attribute bytes

Input Software

- Keyboard driver delivers a number
  – driver converts to characters
  – uses a ASCII table

- Exceptions, adaptations needed for other languages
  – many OS provide for loadable keymaps or code pages

The X Window System (1)



Clients and servers in the M.I.T. X Window System.

The X Window System (2)

Types of messages between client and server:

1. Drawing commands from the program to the workstation.
2. Replies by the workstation to program queries.
3. Keyboard, mouse, and other event announcements.
4. Error messages.

Graphical User Interfaces (1)

```
#include <X11/Xlib.h>
#include <X11/Xutil.h>

main(int argc, char *argv[])
{
        Display disp;                                   /* server identifier */
        Window win;                                     /* window identifier */
        GC gc;                                          /* graphic context identifier */
        XEvent event;                                   /* storage for one event */
        int running = 1;

        disp = XOpenDisplay("display_name");            /* connect to the X server */
        win = XCreateSimpleWindow(disp, ... );          /* allocate memory for new window */
        XSetStandardProperties(disp, ...);         /* announces window to window mgr */
        gc = XCreateGC(disp, win, 0, 0);           /* create graphic context */
        XSelectInput(disp, win, ButtonPressMask | KeyPressMask | ExposureMask);
        XMapRaised(disp, win);                     /* display window; send Expose event */

        . . .
```

Figure 5-38. A skeleton of an X Window application program.

Graphical User Interfaces (2)

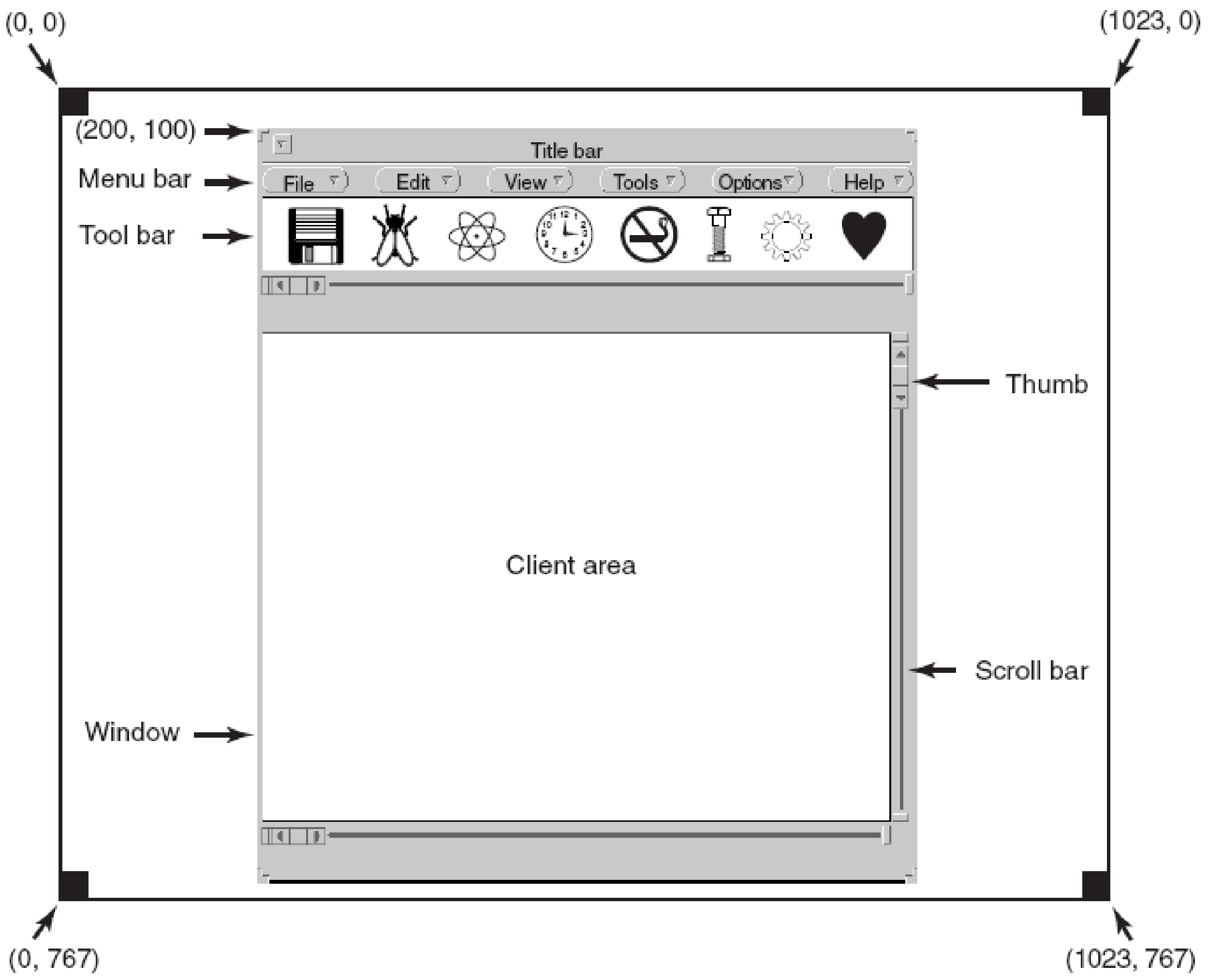
```
. . .

    while (running) {
            XNextEvent(disp, &event);          /* get next event */
            switch (event.type) {
                 case Expose:         ...;   break;       /* repaint window */
                 case ButtonPress:    ...;   break;       /* process mouse click */
                 case Keypress:       ...;   break;       /* process keyboard input */
            }
    }

    XFreeGC(disp, gc);                         /* release graphic context */
    XDestroyWindow(disp, win);                 /* deallocate window's memory space */
    XCloseDisplay(disp);                       /* tear down network connection */
}
```

Figure 5-38. A skeleton of an X Window application program.

Graphical User Interfaces (3)



A sample window located at (200, 100) on an XGA display.

Graphical User Interfaces (4)

```
#include <windows.h>

int WINAPI WinMain(HINSTANCE h, HINSTANCE, hprev, char *szCmd, int iCmdShow)
{
     WNDCLASS wndclass;                    /* class object for this window */
     MSG msg;                              /* incoming messages are stored here */
     HWND hwnd;                            /* handle (pointer) to the window object */

     /* Initialize wndclass */
     wndclass.lpfnWndProc = WndProc;       /* tells which procedure to call */
     wndclass.lpszClassName = "Program name";     /* Text for title bar */
     wndclass.hIcon = LoadIcon(NULL, IDI_APPLICATION);      /* load program icon */
     wndclass.hCursor = LoadCursor(NULL, IDC_ARROW);        /* load mouse cursor */

     RegisterClass(&wndclass);             /* tell Windows about wndclass */
     hwnd = CreateWindow ( ... )           /* allocate storage for the window */
     ShowWindow(hwnd, iCmdShow);           /* display the window on the screen */
     UpdateWindow(hwnd);                   /* tell the window to paint itself */
  . . .
```

A skeleton of a Windows main program.

Graphical User Interfaces (5)

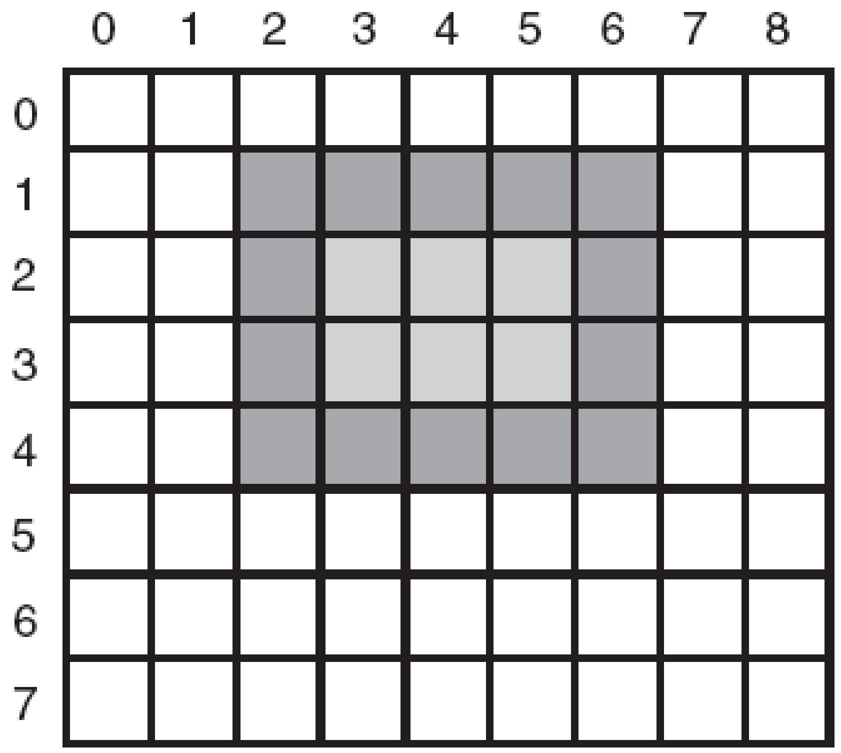
```
. . .
        while (GetMessage(&msg, NULL, 0, 0)) {          /* get message from queue */
                TranslateMessage(&msg);         /* translate the message */
                DispatchMessage(&msg);          /* send msg to the appropriate procedure */
        }
        return(msg.wParam);
}

long CALLBACK WndProc(HWND hwnd, UINT message, UINT wParam, long lParam)
{
        /* Declarations go here. */

        switch (message) {
                case WM_CREATE:     ... ;   return ... ;    /* create window */
                case WM_PAINT:      ... ;   return ... ;    /* repaint contents of window */
                case WM_DESTROY:    ... ;   return ... ;    /* destroy window */
        }
        return(DefWindowProc(hwnd, message, wParam, lParam));       /* default */
}
```

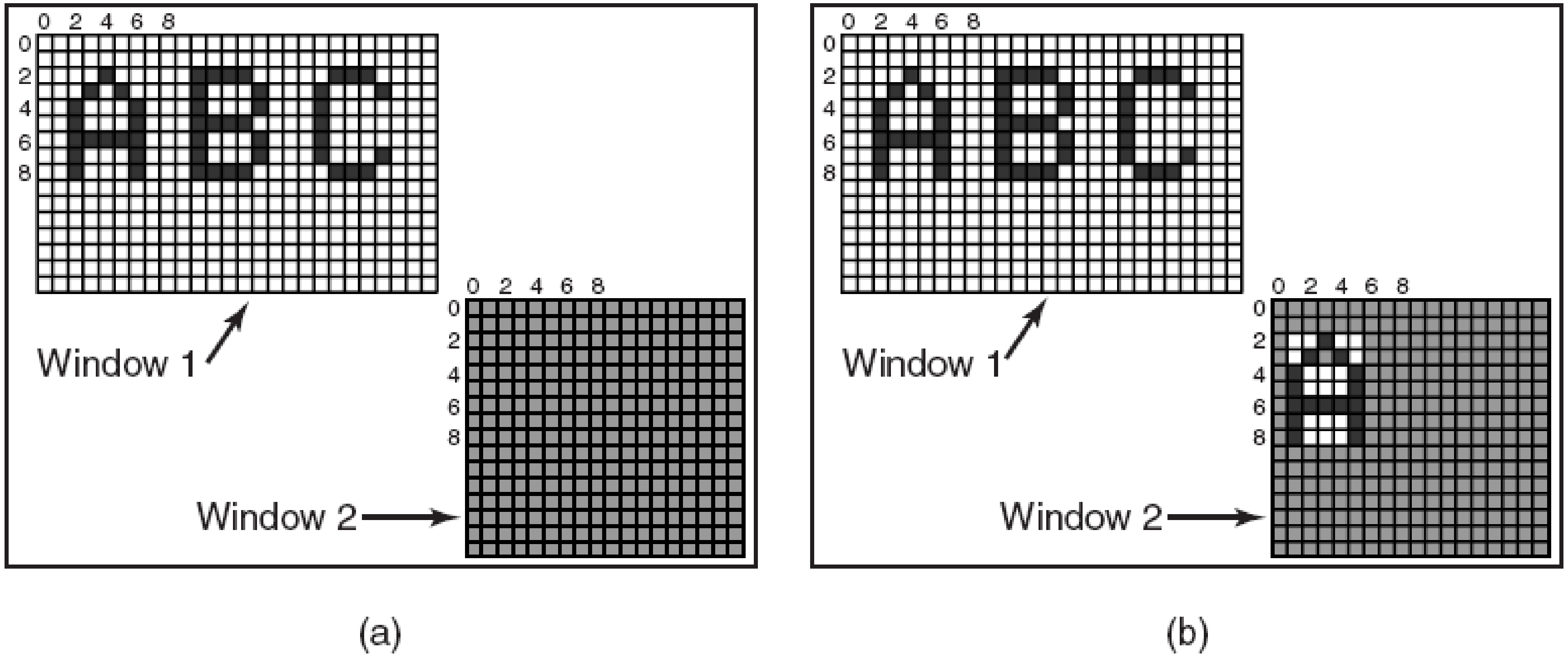
A skeleton of a Windows main program.

Bitmaps (1)



An example rectangle drawn using *Rectangle*. Each box represents one pixel.

Bitmaps (2)

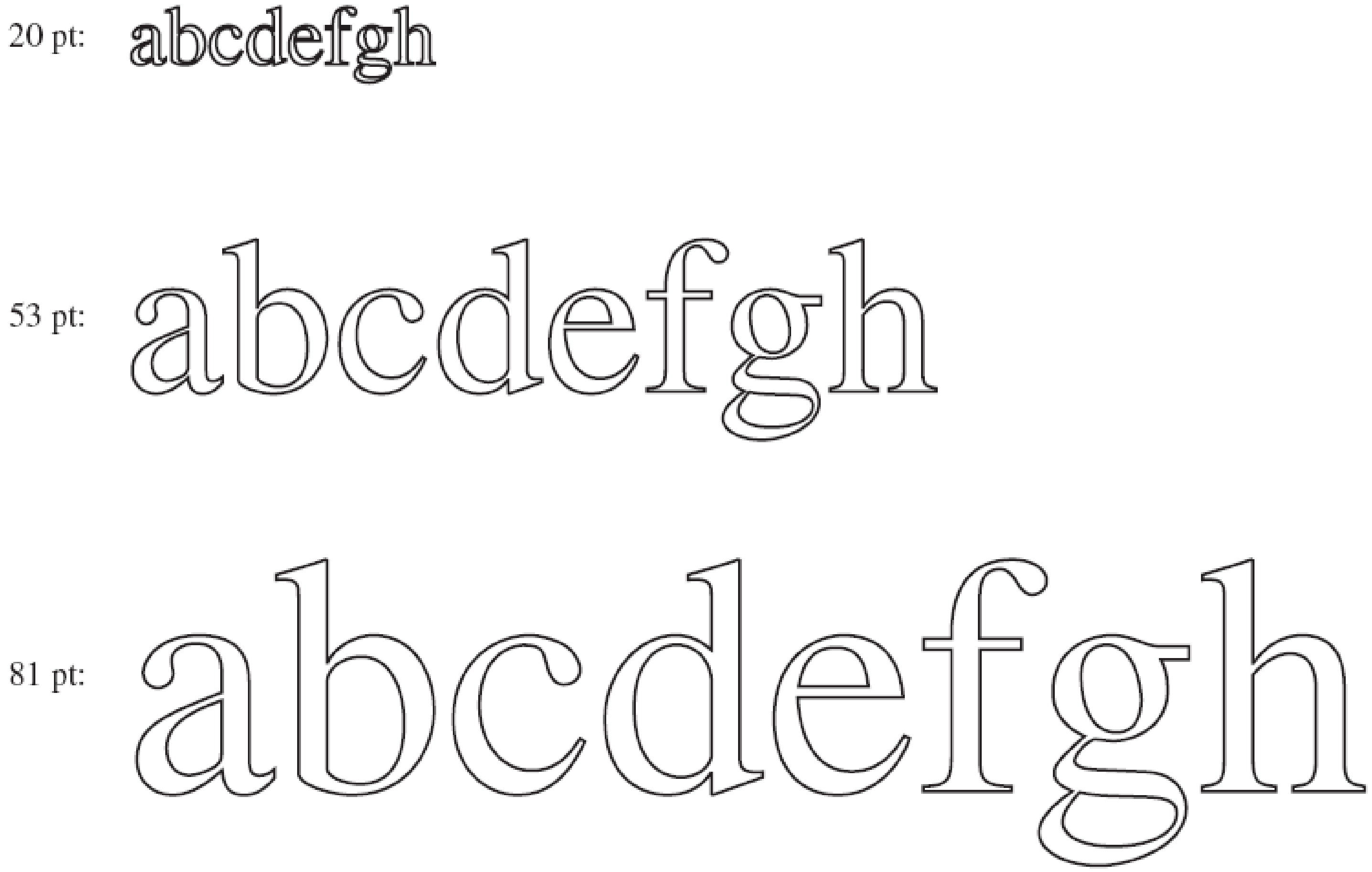


Copying bitmaps using *BitBlt*.

(a) Before.

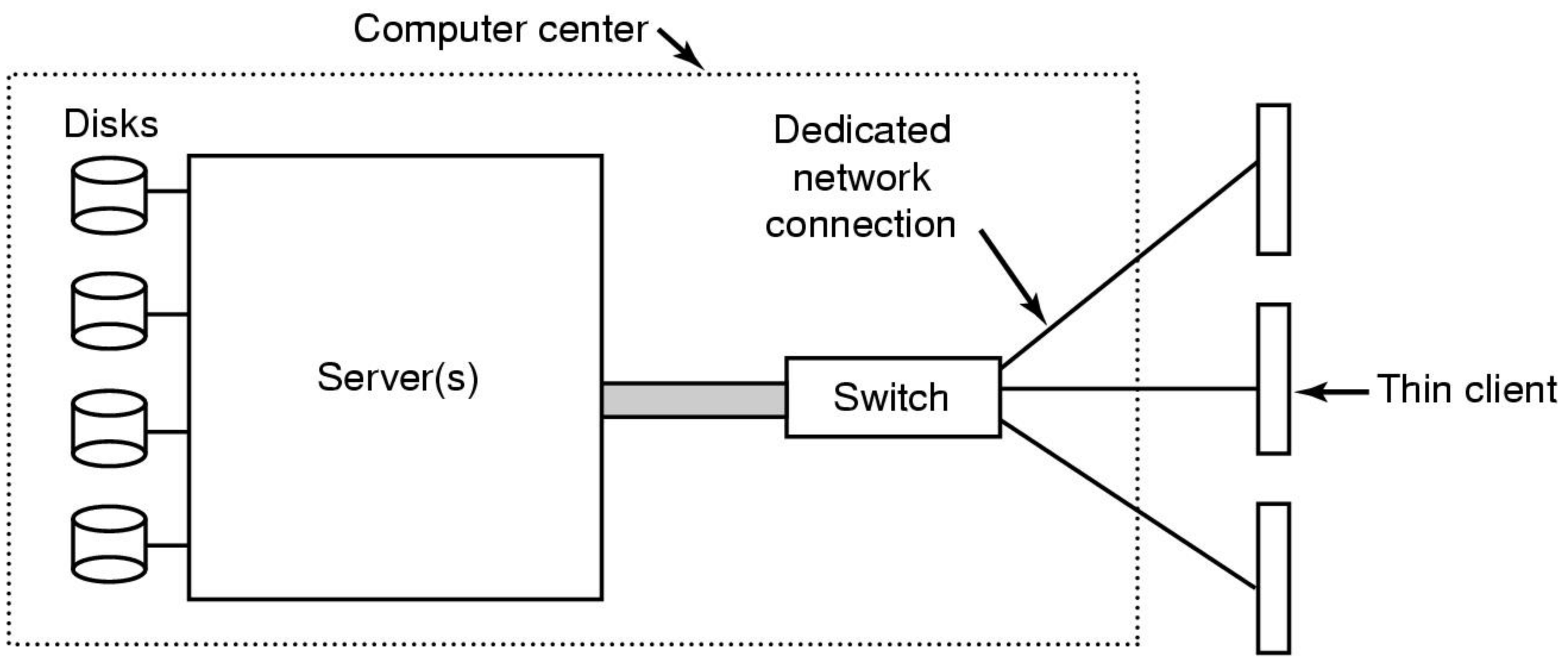
(b) After.

True Type Fonts



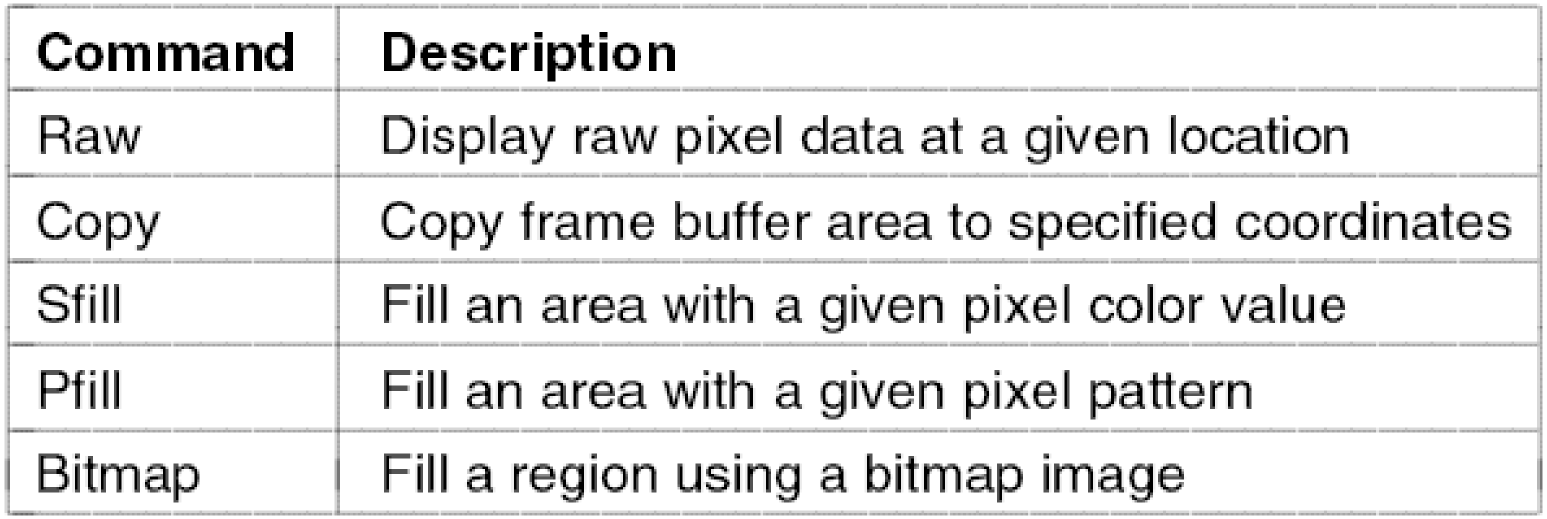
Some examples of character outlines at different point sizes.

Thin Clients (1)



The architecture of the thin clients system

Thin Clients (2)



Command	**Description**
Raw	Display raw pixel data at a given location
Copy	Copy frame buffer area to specified coordinates
Sfill	Fill an area with a given pixel color value
Pfill	Fill an area with a given pixel pattern
Bitmap	Fill a region using a bitmap image

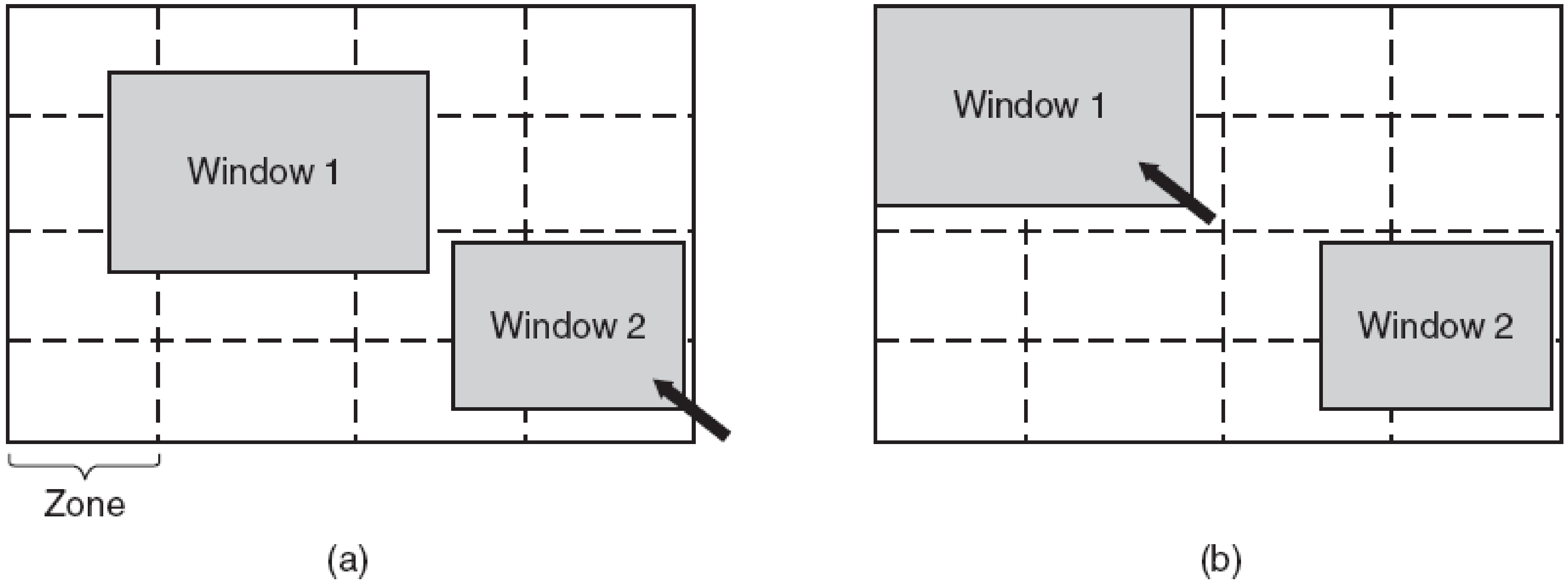
The THINC protocol display commands.

Power Management Hardware Issues

Device	Li et al. (1994)	Lorch and Smith (1998)
Display	68%	39%
CPU	12%	18%
Hard disk	20%	12%
Modem		6%
Sound		2%
Memory	0.5%	1%
Other		22%

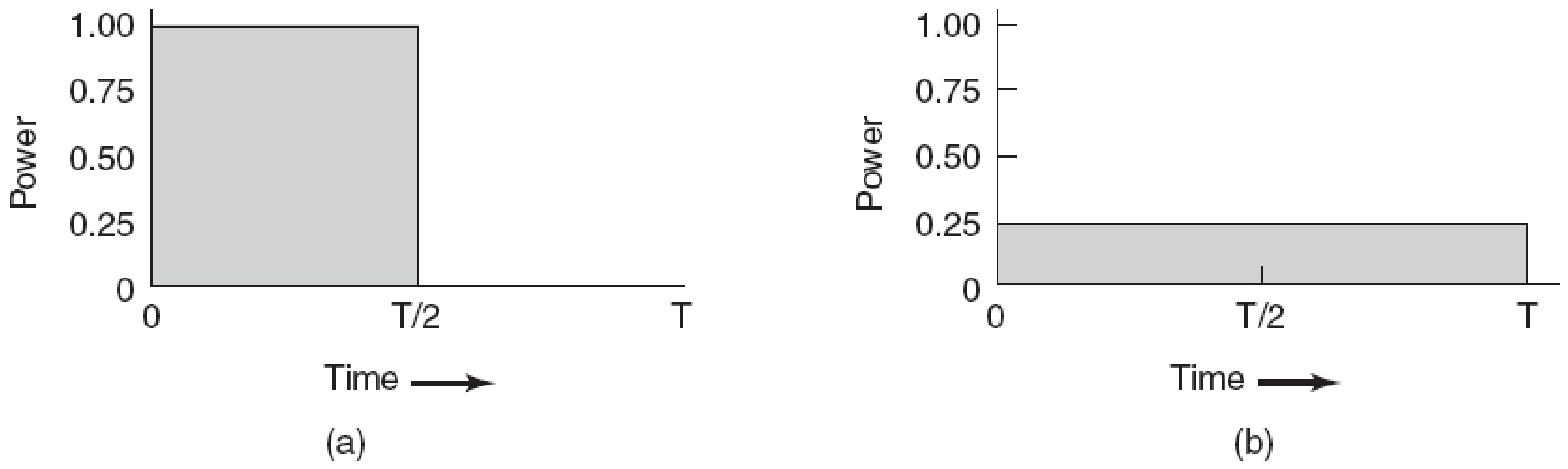
Power consumption of various parts of a notebook computer.

Power Management
The Display



The use of zones for backlighting the display.
(a) When window 2 is selected it is not moved.
(b) When window 1 is selected, it moves to reduce the number of zones illuminated.

Power Management
The CPU



(a) Running at full clock speed

(b) Cutting voltage by two

- cuts clock speed by two
- power consumption by four