5

INPUT/OUTPUT

Device	**Data rate**
Keyboard	10 bytes/sec
Mouse	100 bytes/sec
56K modem	7 KB/sec
Telephone channel	8 KB/sec
Dual ISDN lines	16 KB/sec
Laser printer	100 KB/sec
Scanner	400 KB/sec
Classic Ethernet	1.25 MB/sec
USB (Universal Serial Bus)	1.5 MB/sec
Digital camcorder	4 MB/sec
IDE disk	5 MB/sec
40x CD-ROM	6 MB/sec
Fast Ethernet	12.5 MB/sec
ISA bus	16.7 MB/sec
EIDE (ATA-2) disk	16.7 MB/sec
FireWire (IEEE 1394)	50 MB/sec
XGA Monitor	60 MB/sec
SONET OC-12 network	78 MB/sec
SCSI Ultra 2 disk	80 MB/sec
Gigabit Ethernet	125 MB/sec
Ultrium tape	320 MB/sec
PCI bus	528 MB/sec
Sun Gigaplane XB backplane	20 GB/sec

Fig. 5-1. Some typical device, network, and bus data rates.

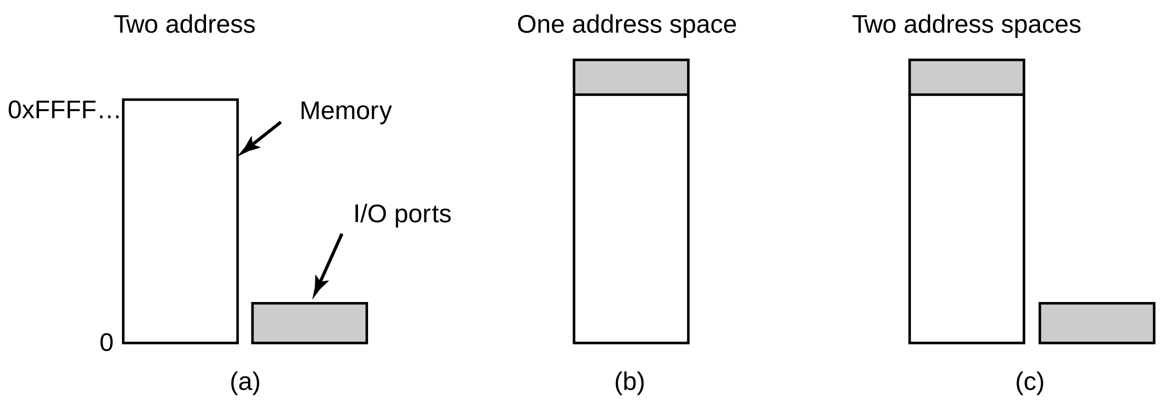


Fig. 5-2. (a) Separate I/O and memory space. (b) Memory-mapped I/O. (c) Hybrid.

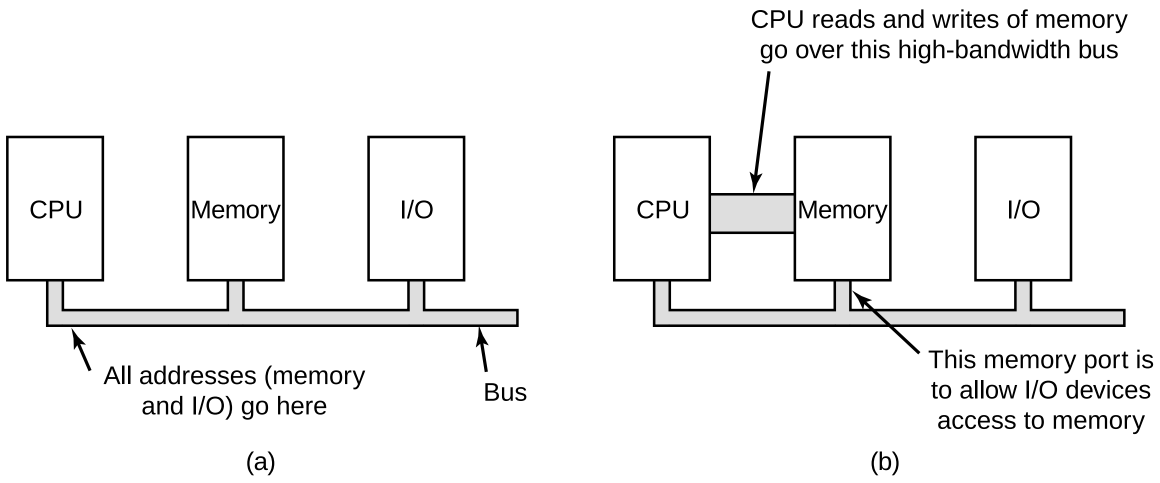

CPU
Memory
I/O
All addresses (memory and I/O) go here
Bus
(a)
CPU reads and writes of memory go over this high-bandwidth bus
CPU
Memory
I/O
This memory port is to allow I/O devices access to memory
(b)

Fig. 5-3. (a) A single-bus architecture. (b) A dual-bus memory architecture.

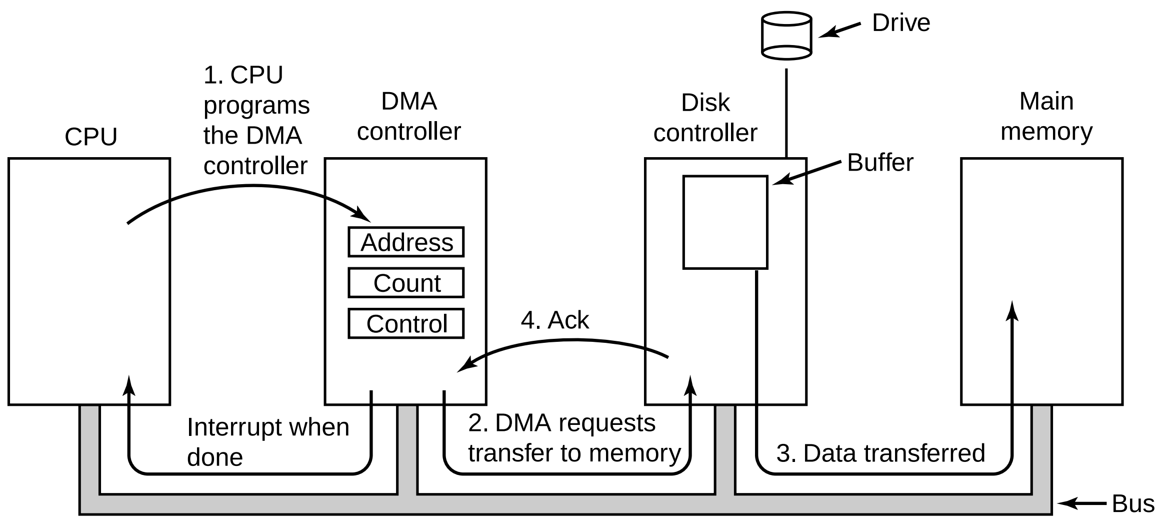


Fig. 5-4. Operation of a DMA transfer.

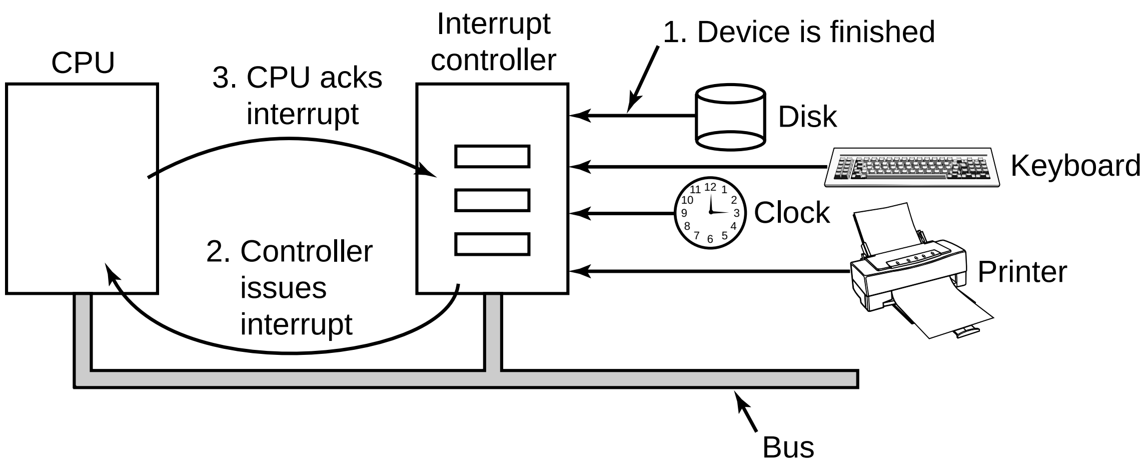


Fig. 5-5. How an interrupt happens. The connections between the devices and the interrupt controller actually use interrupt lines on the bus rather than dedicated wires.

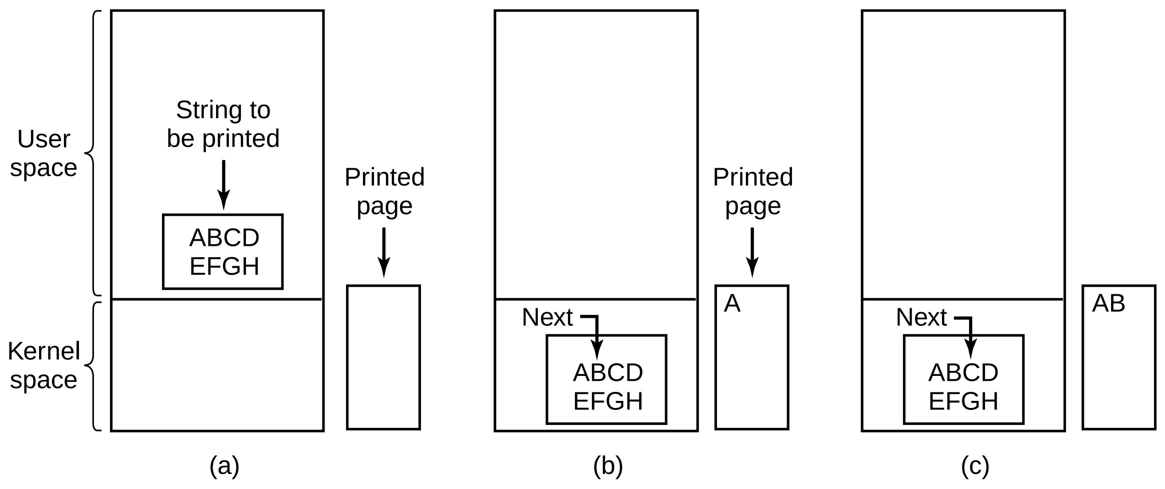


Fig. 5-6. Steps in printing a string.

```
copy_from_user(buffer, p, count);              /* p is the kernel bufer */
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( );
```

Fig. 5-7. Writing a string to the printer using programmed 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)

Fig. 5-8. Writing a string to the printer using interrupt-driven I/O. (a) Code executed when the print system call is made. (b) Interrupt service procedure.

```
copy_from_user(buffer, p, count);    acknowledge_interrupt( );
set_up_DMA_controller( );            unblock_user( );
scheduler( );                        return_from_interrupt( );

              (a)                                (b)
```

Fig. 5-9. Printing a string using DMA. (a) Code executed when the print system call is made. (b) Interrupt service procedure.

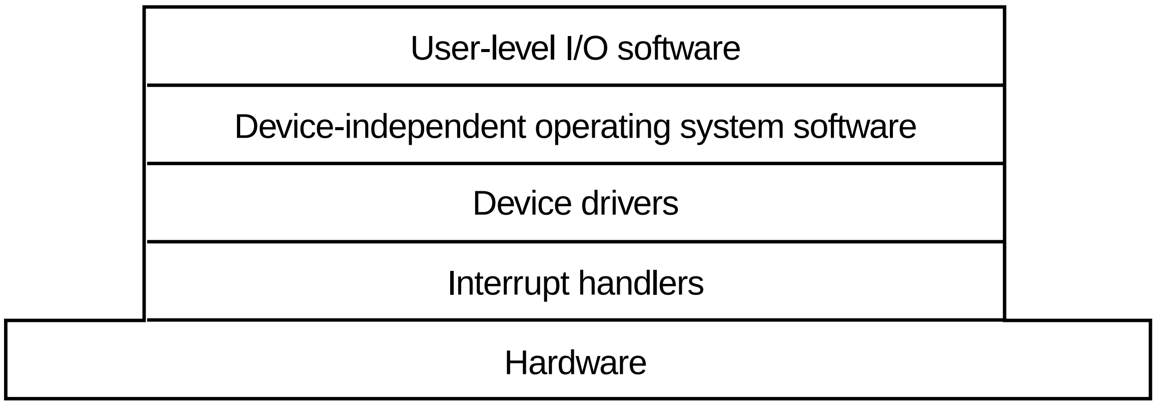


Fig. 5-10. Layers of the I/O software system.

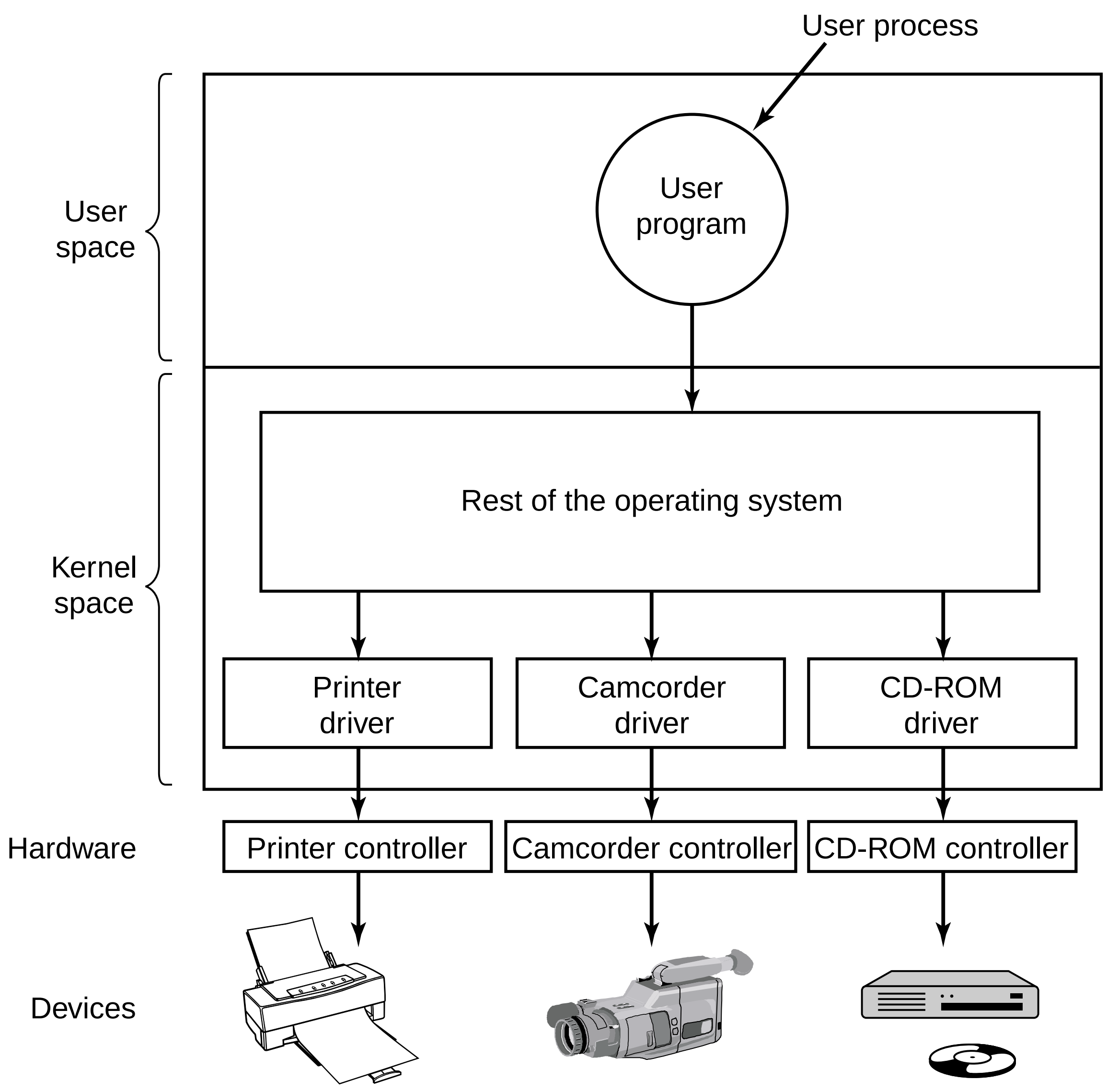


Fig. 5-11. Logical positioning of device drivers. In reality all communication between drivers and device controllers goes over the bus.

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

Fig. 5-12. Functions of the device-independent I/O software.

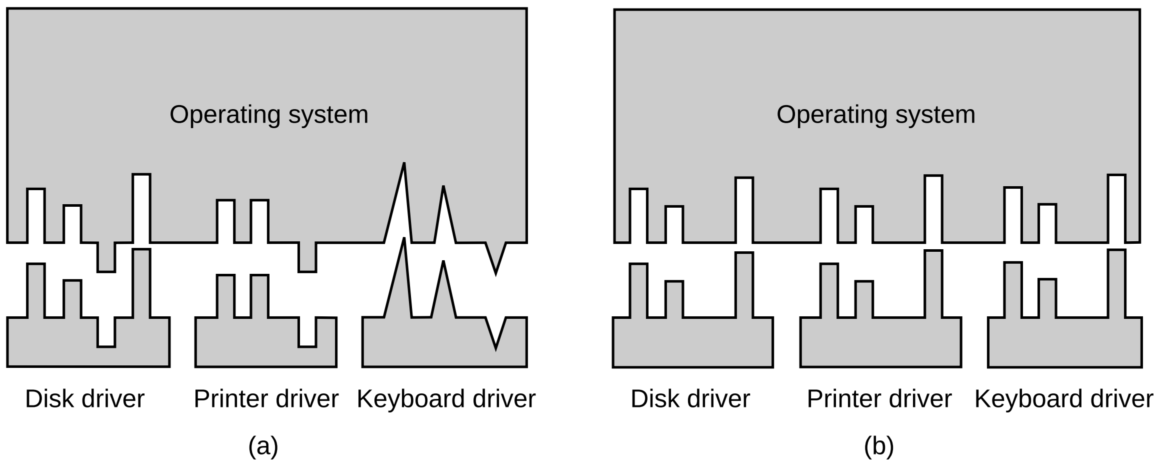


Fig. 5-13. (a) Without a standard driver interface. (b) With a standard driver interface.

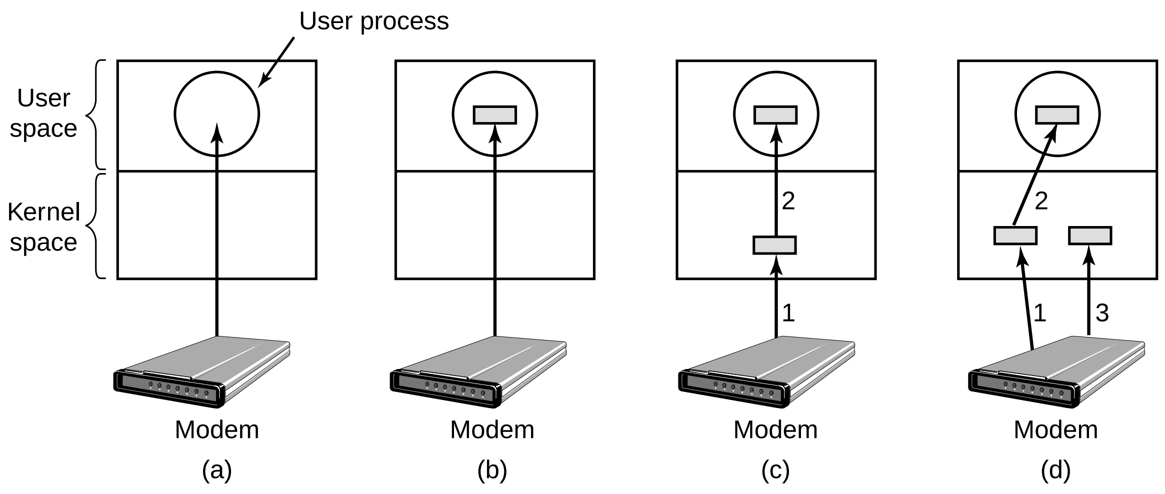


Fig. 5-14. (a) Unbuffered input. (b) Buffering in user space.
(c) Buffering in the kernel followed by copying to user space. (d) Double buffering in the kernel.

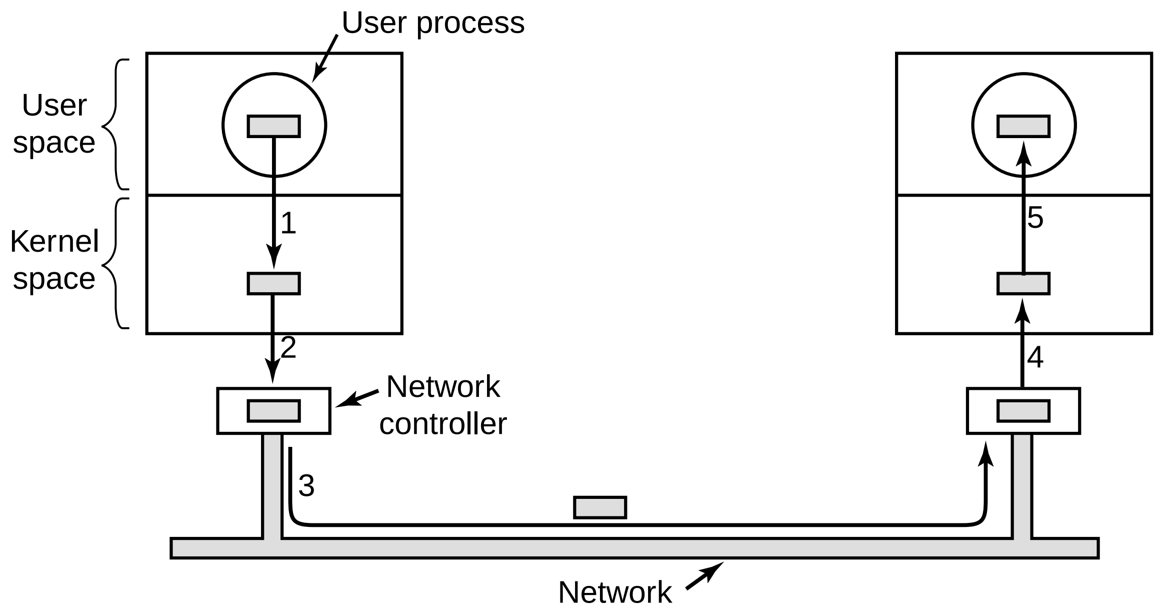


Fig. 5-15. Networking may involve many copies of a packet.

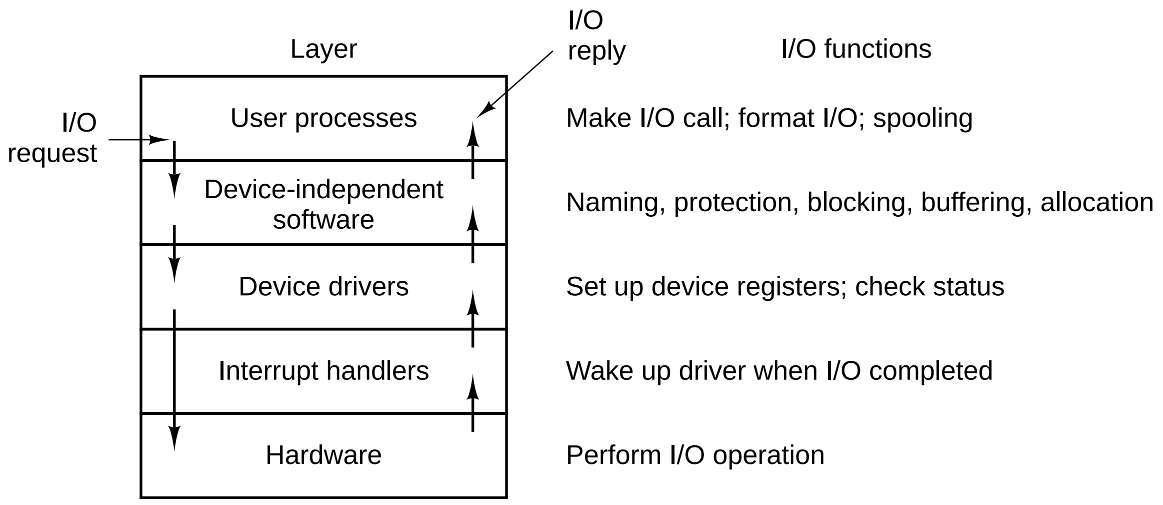

Layer
I/O reply
I/O functions
I/O request
User processes
Make I/O call; format I/O; spooling
Device-independent software
Naming, protection, blocking, buffering, allocation
Device drivers
Set up device registers; check status
Interrupt handlers
Wake up driver when I/O completed
Hardware
Perform I/O operation

Fig. 5-16. Layers of the I/O system and the main functions of each layer.

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

Fig. 5-17. Disk parameters for the original IBM PC 360-KB floppy disk and a Western Digital WD 18300 hard disk.

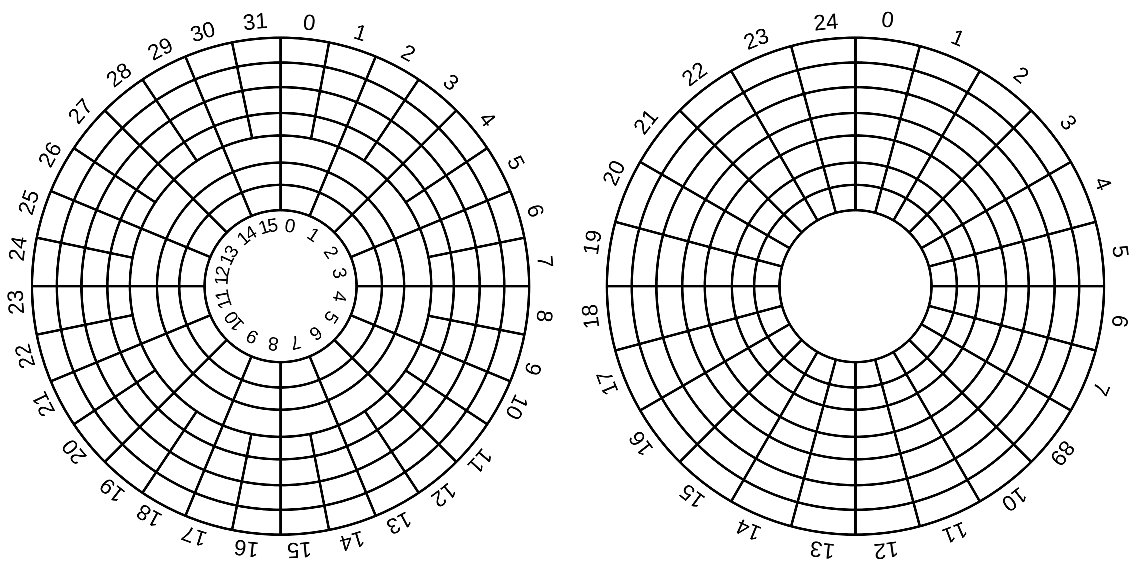


Fig. 5-18. (a) Physical geometry of a disk with two zones. (b) A possible virtual geometry for this disk.

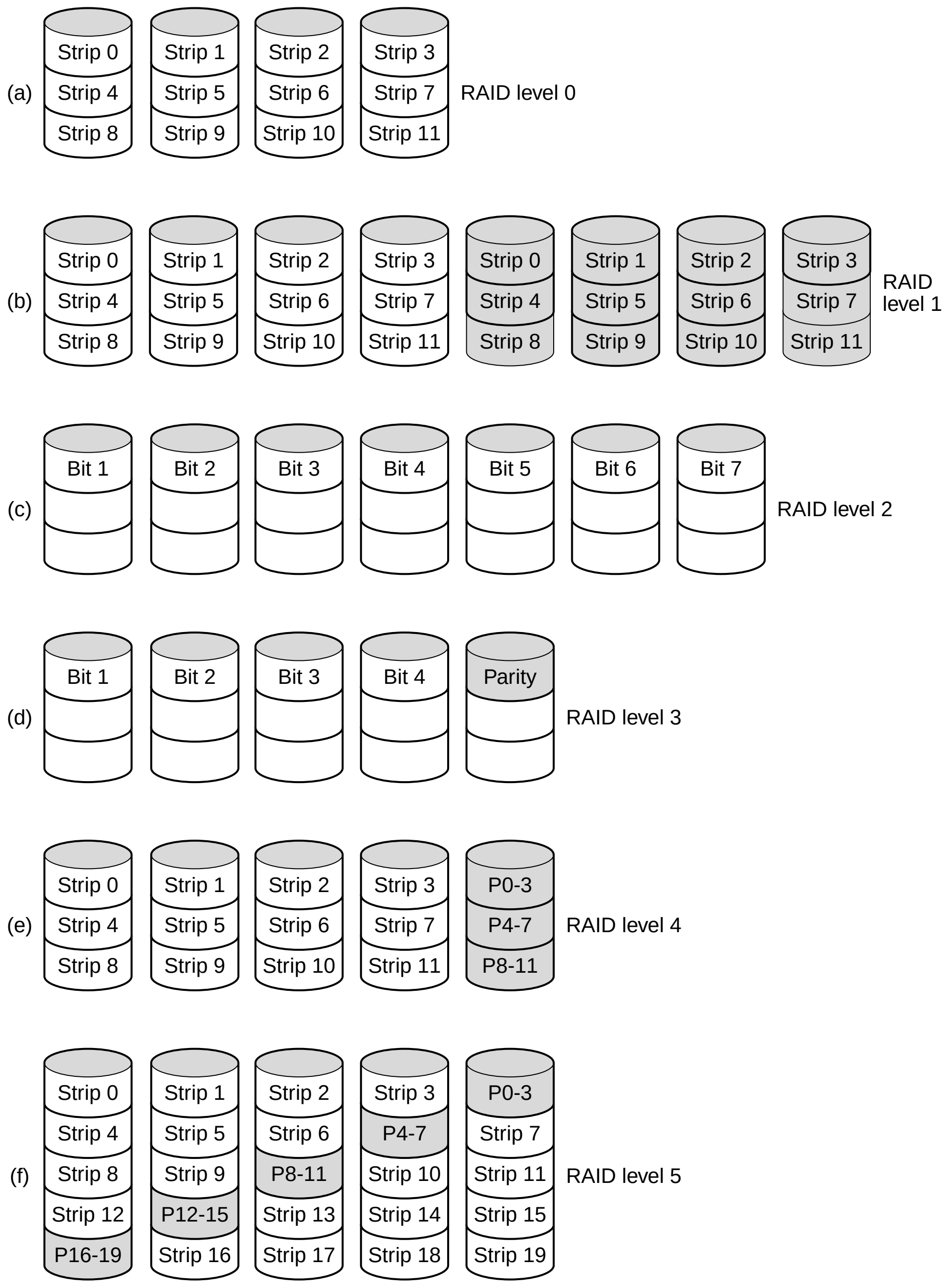


Fig. 5-19. RAID levels 0 through 5. Backup and parity drives are shown shaded.

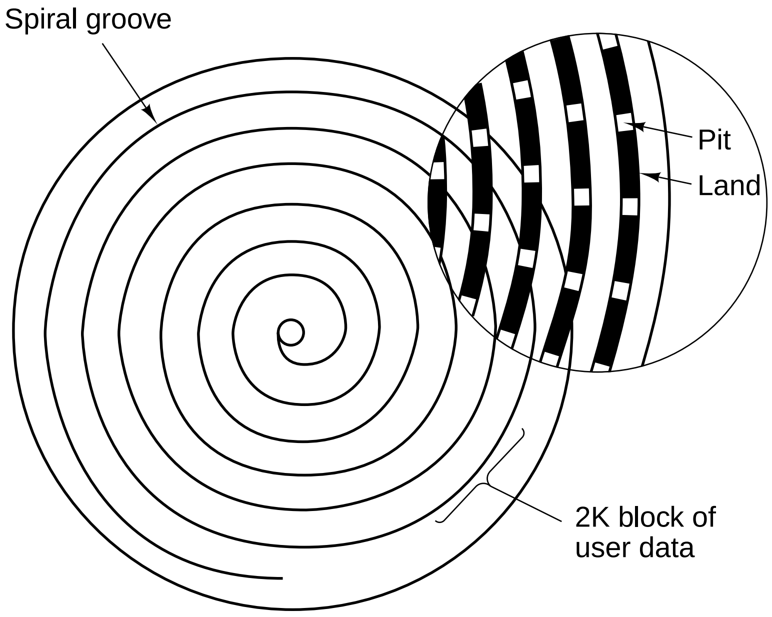


Fig. 5-20. Recording structure of a compact disc or CD-ROM.

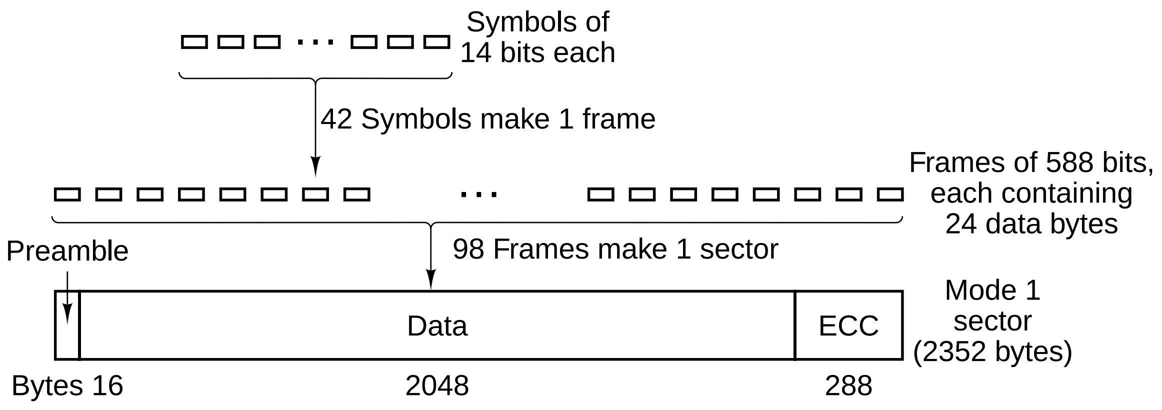


Fig. 5-21. Logical data layout on a CD-ROM.

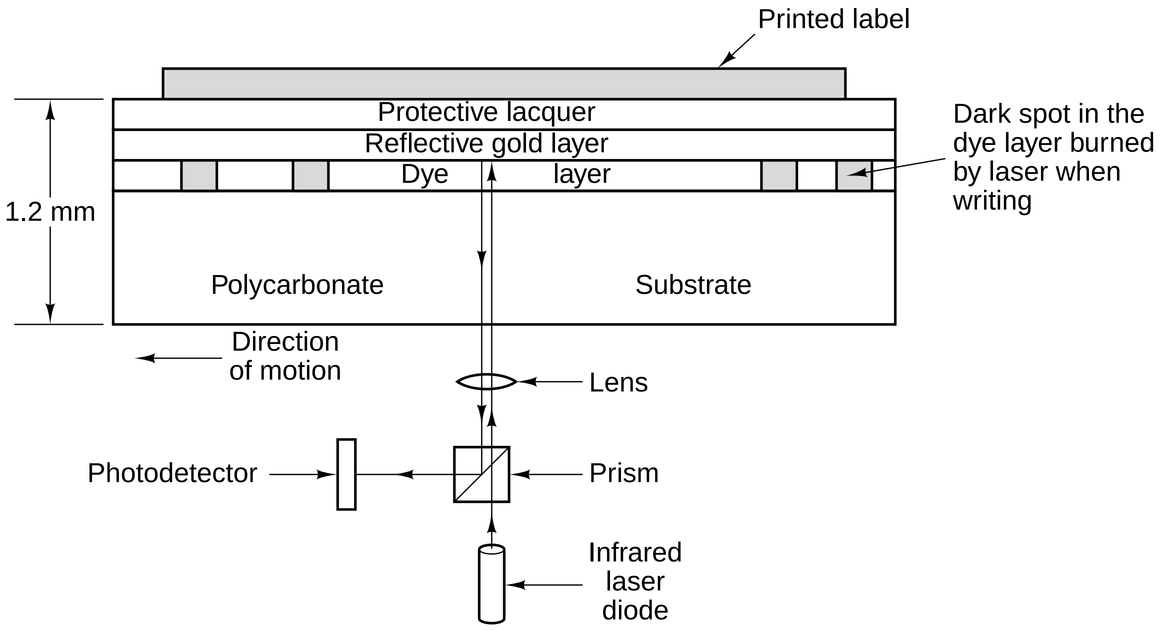


Fig. 5-22. Cross section of a CD-R disk and laser (not to scale). A silver CD-ROM has a similar structure, except without the dye layer and with a pitted aluminum layer instead of a gold layer.

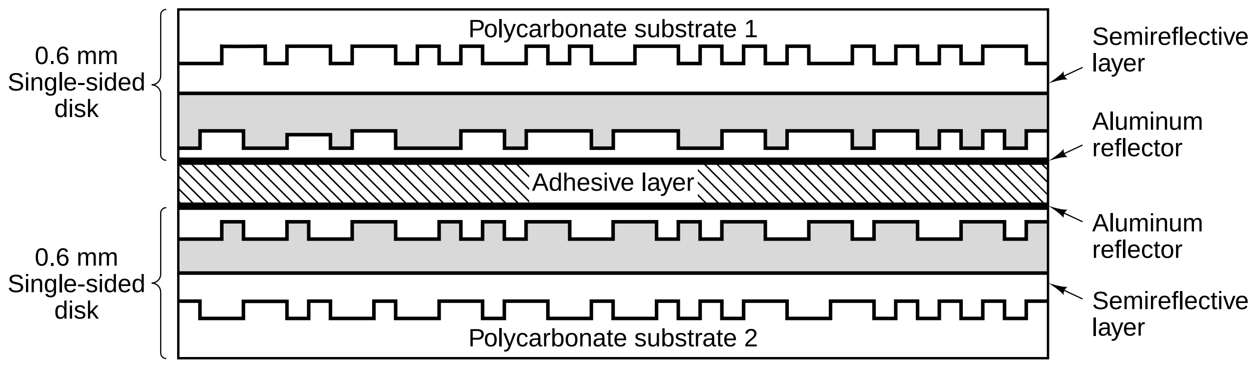


Fig. 5-23. A double-sided, dual layer DVD disk.

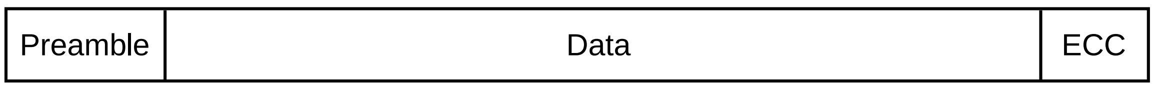


Fig. 5-24. A disk sector.

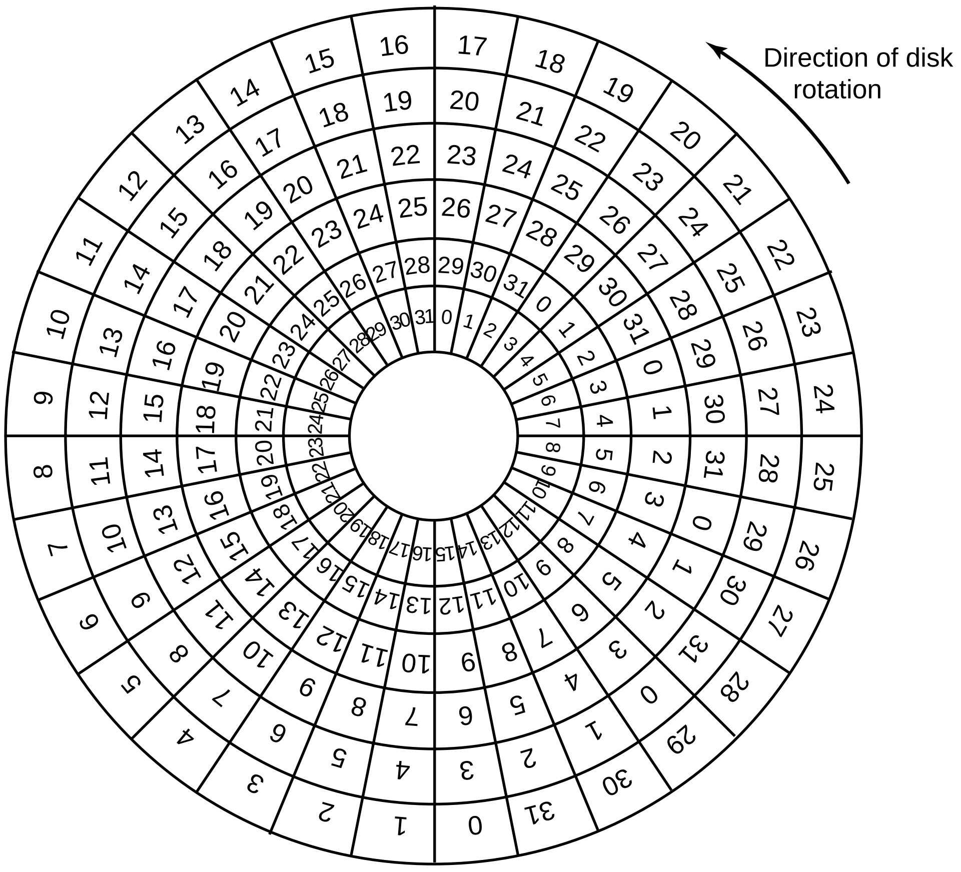


Fig. 5-25. An illustration of cylinder skew.

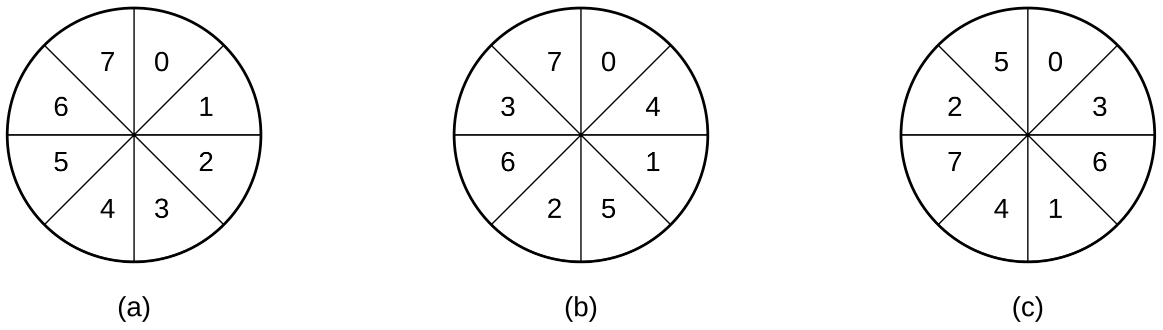


Fig. 5-26. (a) No interleaving. (b) Single interleaving. (c) Double interleaving.

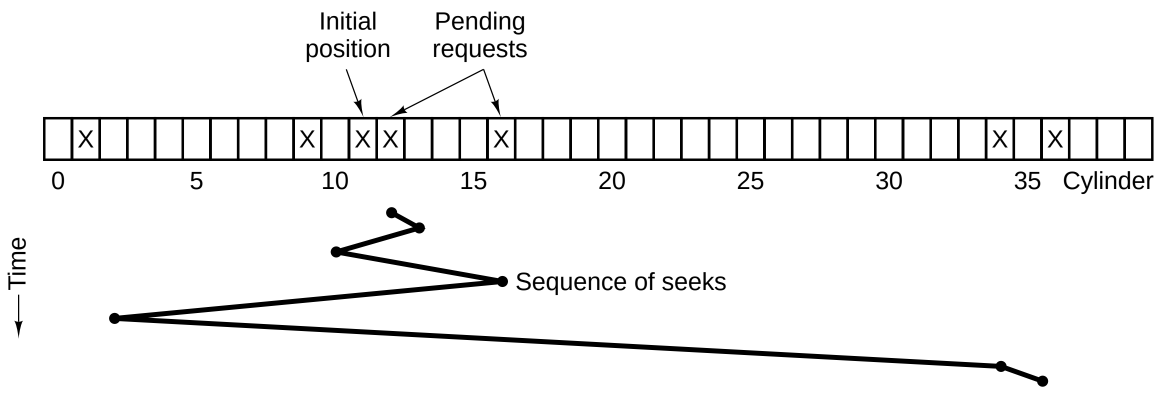


Fig. 5-27. Shortest Seek First (SSF) disk scheduling algorithm.

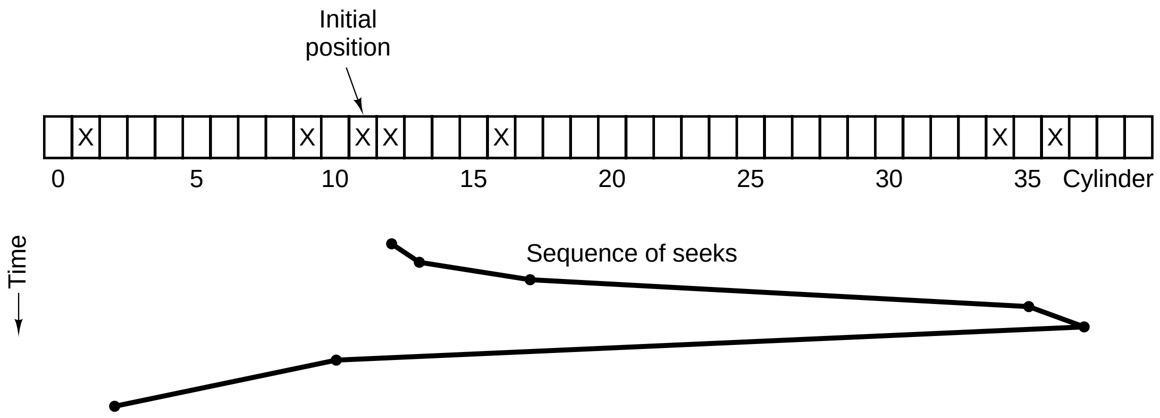


Fig. 5-28. The elevator algorithm for scheduling disk requests.

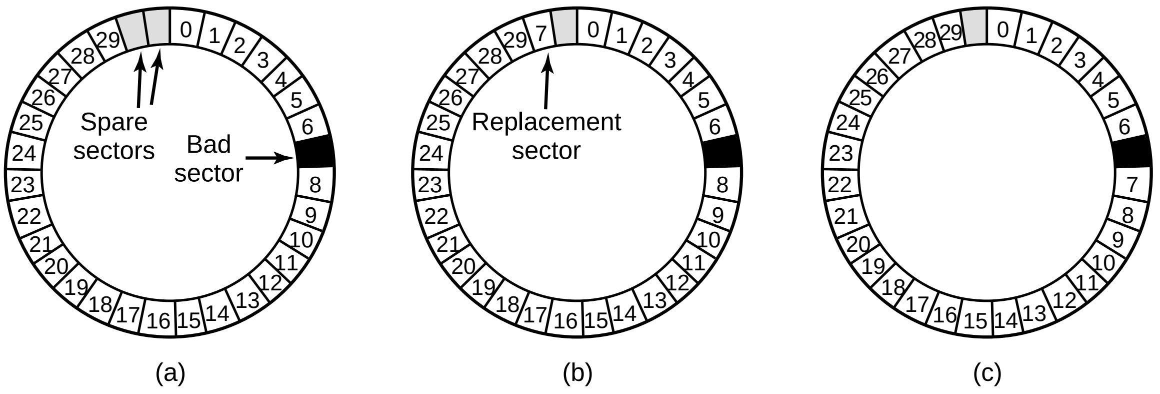


Fig. 5-29. (a) A disk track with a bad sector. (b) Substituting a spare for the bad sector. (c) Shifting all the sectors to bypass the bad one.

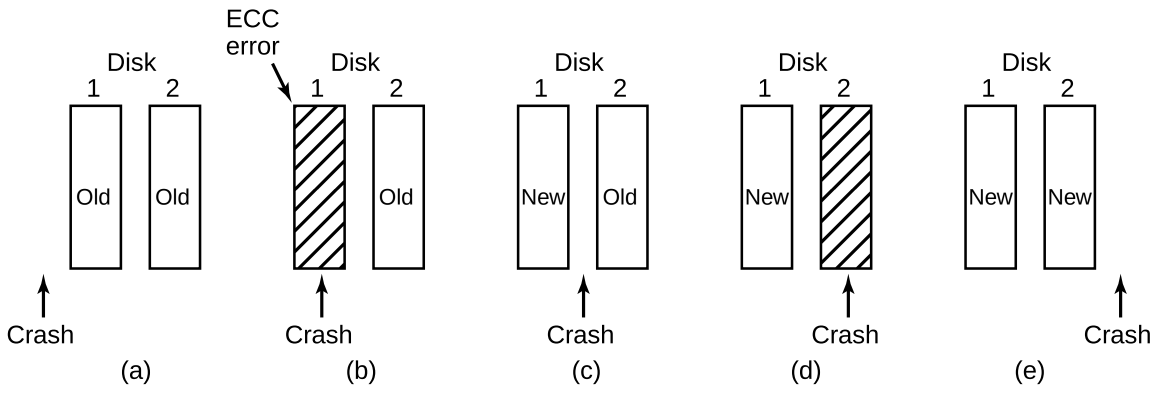


Fig. 5-30. Analysis of the influence of crashes on stable writes.

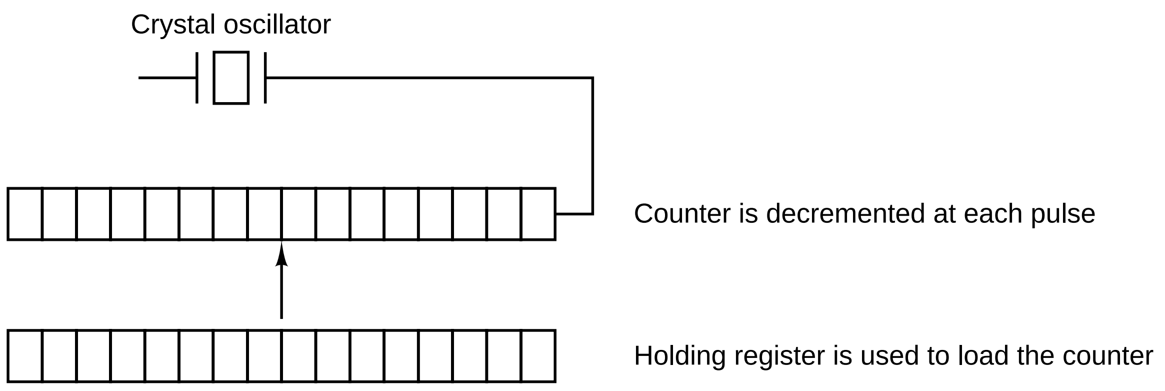

Crystal oscillator
Counter is decremented at each pulse
Holding register is used to load the counter

Fig. 5-31. A programmable clock.

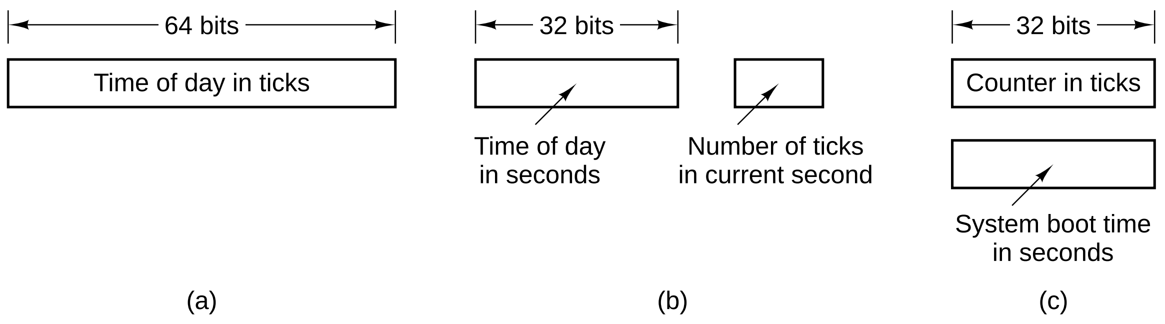


Fig. 5-32. Three ways to maintain the time of day.

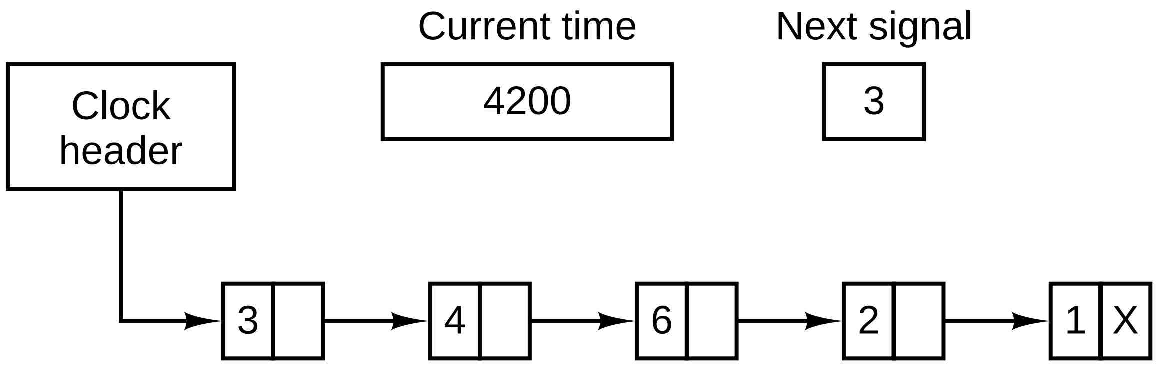


Fig. 5-33. Simulating multiple timers with a single clock.

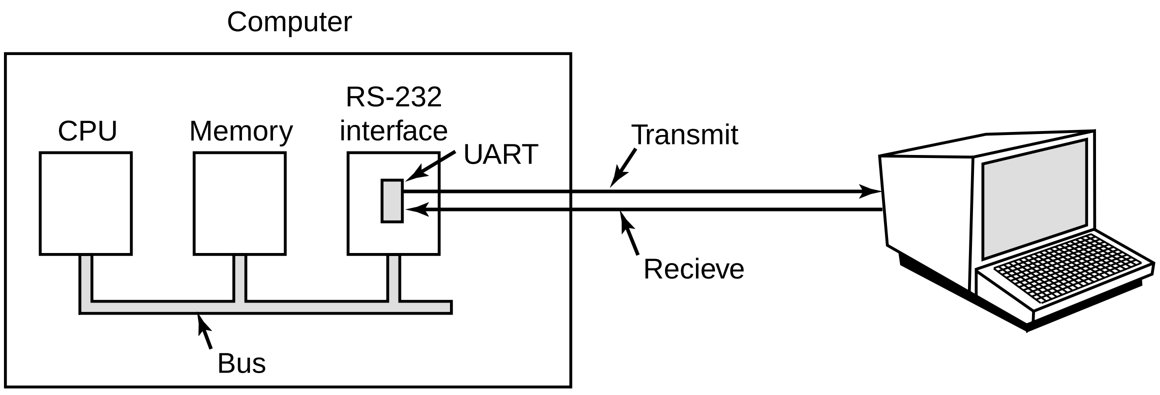


Fig. 5-34. An RS-232 terminal communicates with a computer over a communication line, one bit at a time.

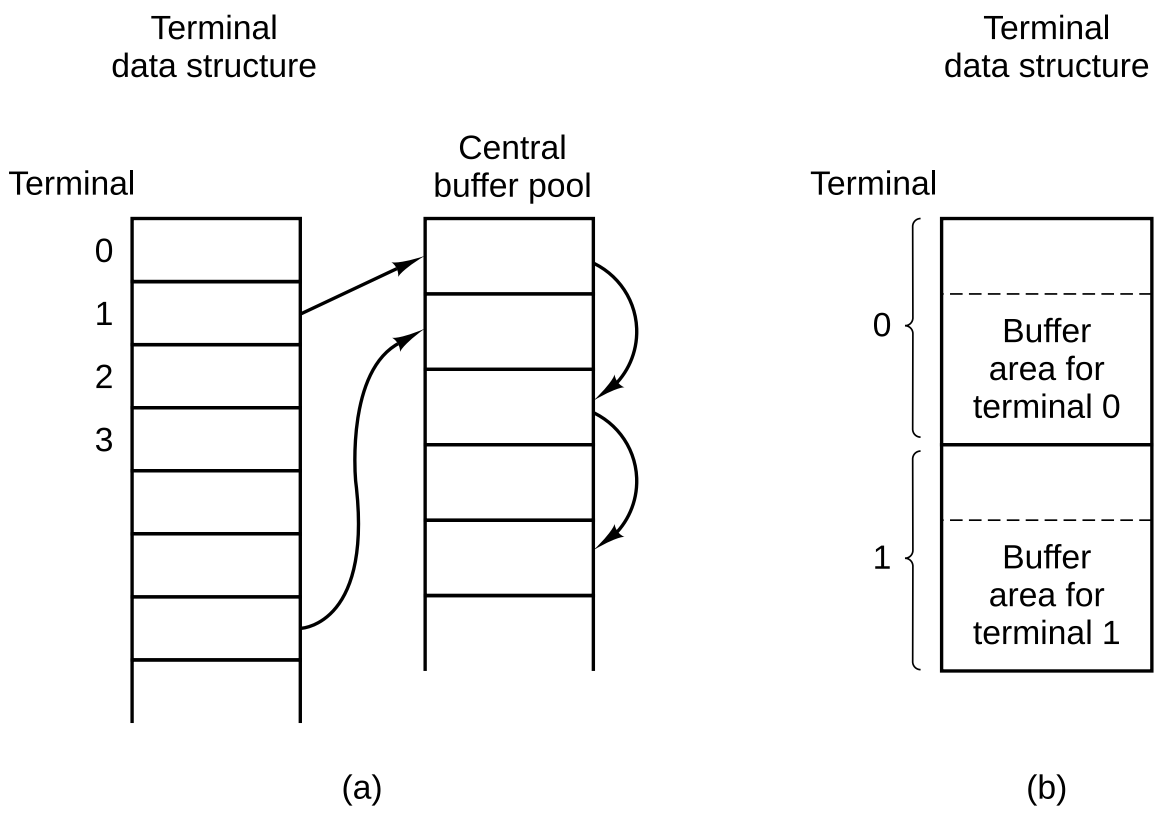


Fig. 5-35. (a) Central buffer pool. (b) Dedicated buffer for each terminal.

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)

Fig. 5-36. Characters that are handled specially in canonical mode.

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 from 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

Fig. 5-37. The ANSI escape sequences accepted by the terminal driver on output. ESC denotes the ASCII escape character (0x1B), and *n*, *m*, and *s* are optional numeric parameters.

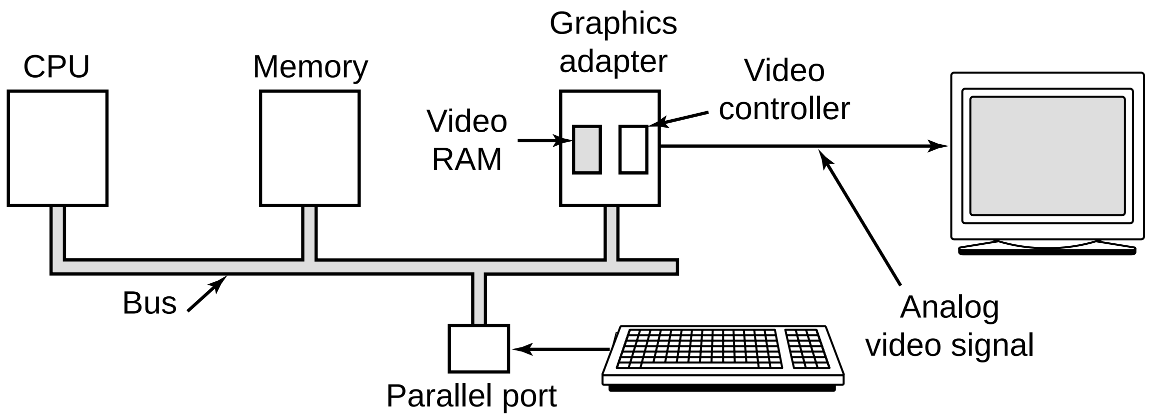


Fig. 5-38. With memory-mapped displays, the driver writes directly into the display's video RAM.

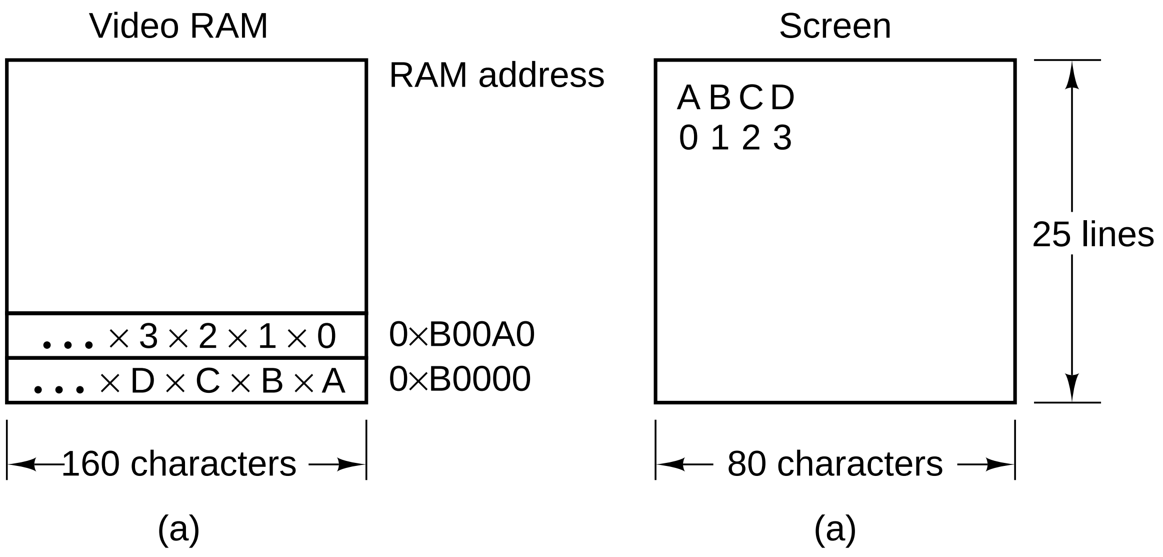


Fig. 5-39. (a) A video RAM image for a simple monochrome display in character mode. (b) The corresponding screen. The ×s are attribute bytes.

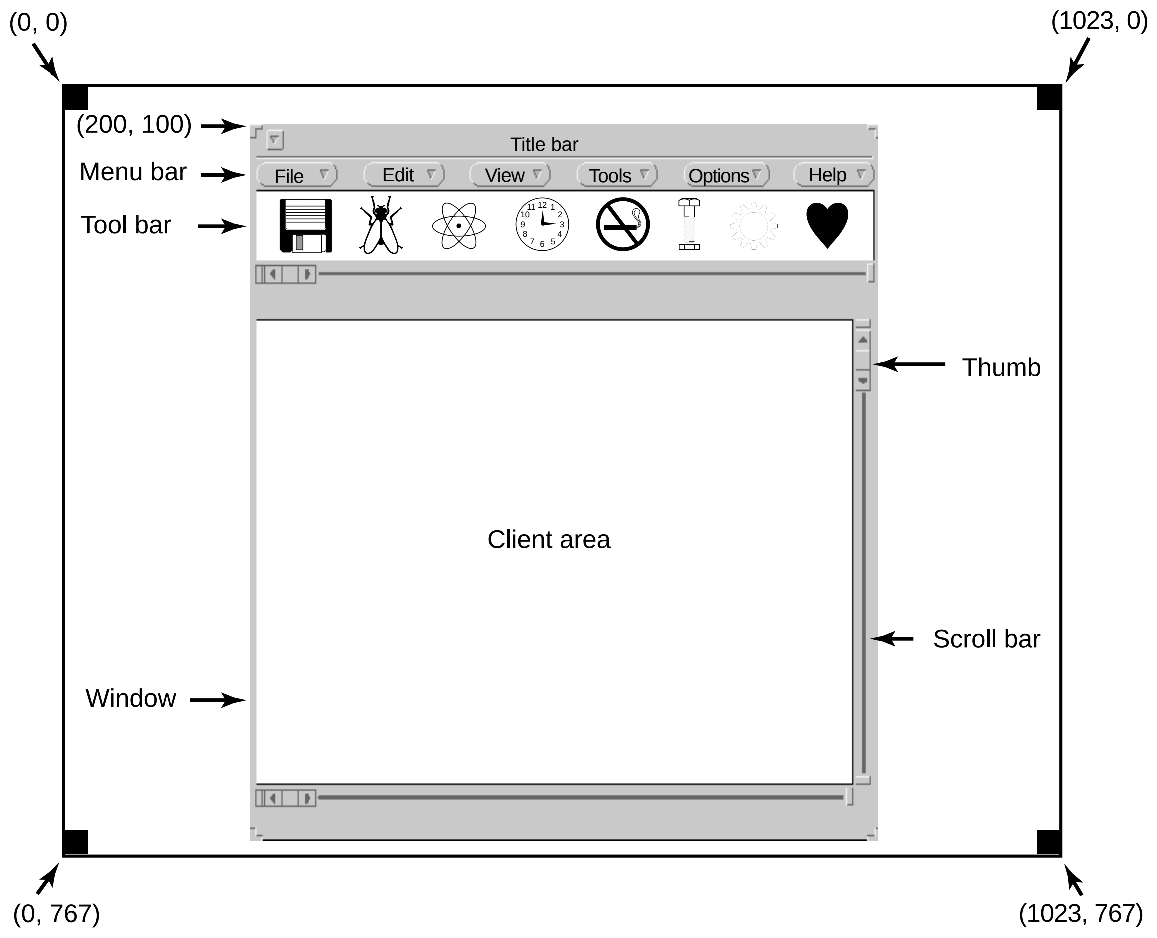


Fig. 5-40. A sample window located at (200, 100) on an XGA display.

```
#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 */

     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 */
}
```

Fig. 5-41. A skeleton of a Windows main program.

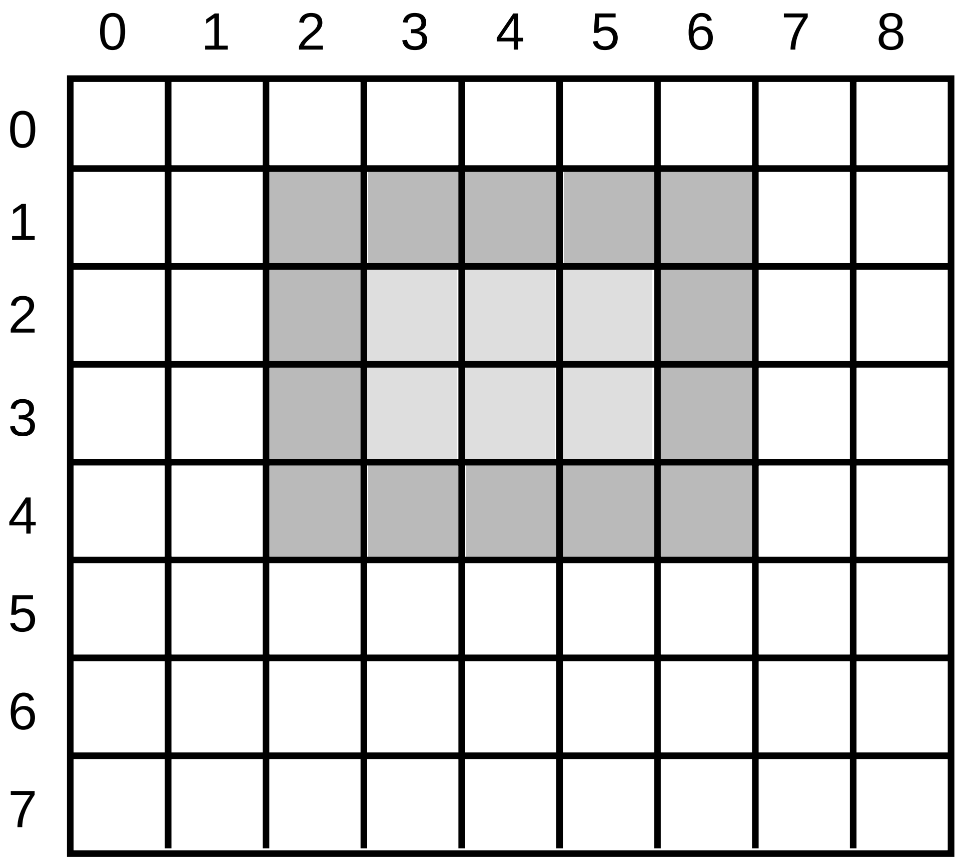
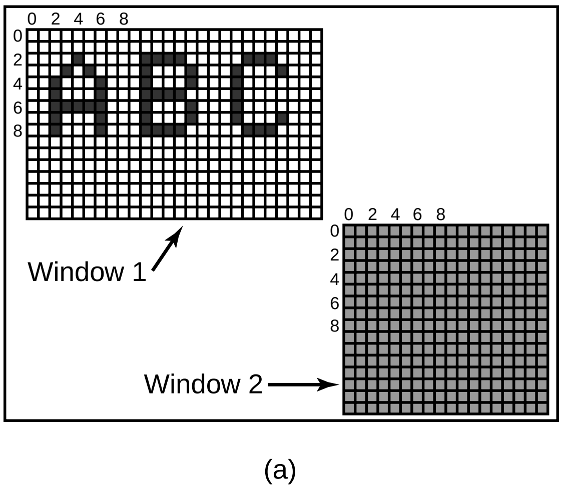


Fig. 5-42. An example rectangle drawn using *Rectangle*. Each box represents one pixel.



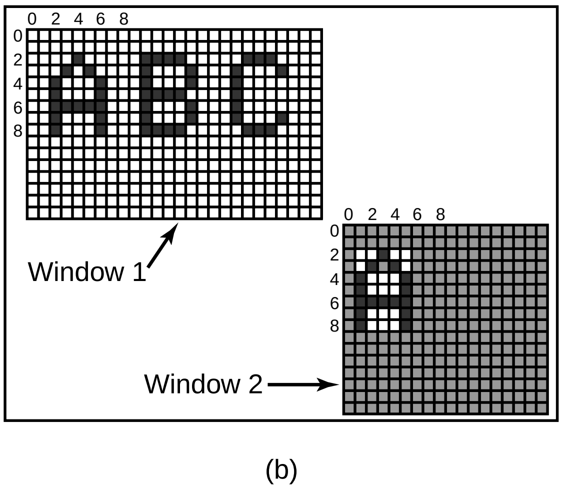


Fig. 5-43. Copying bitmaps using *BitBlt*. (a) Before. (b) After.

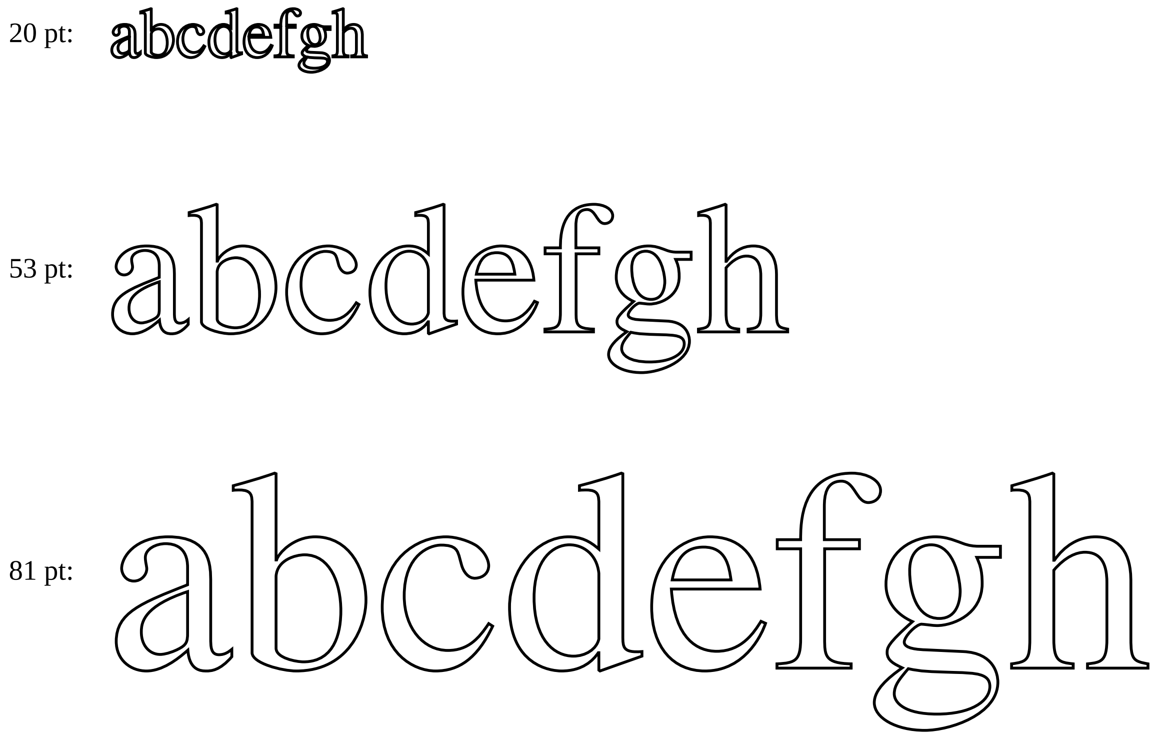


Fig. 5-44. Some examples of character outlines at different point sizes.

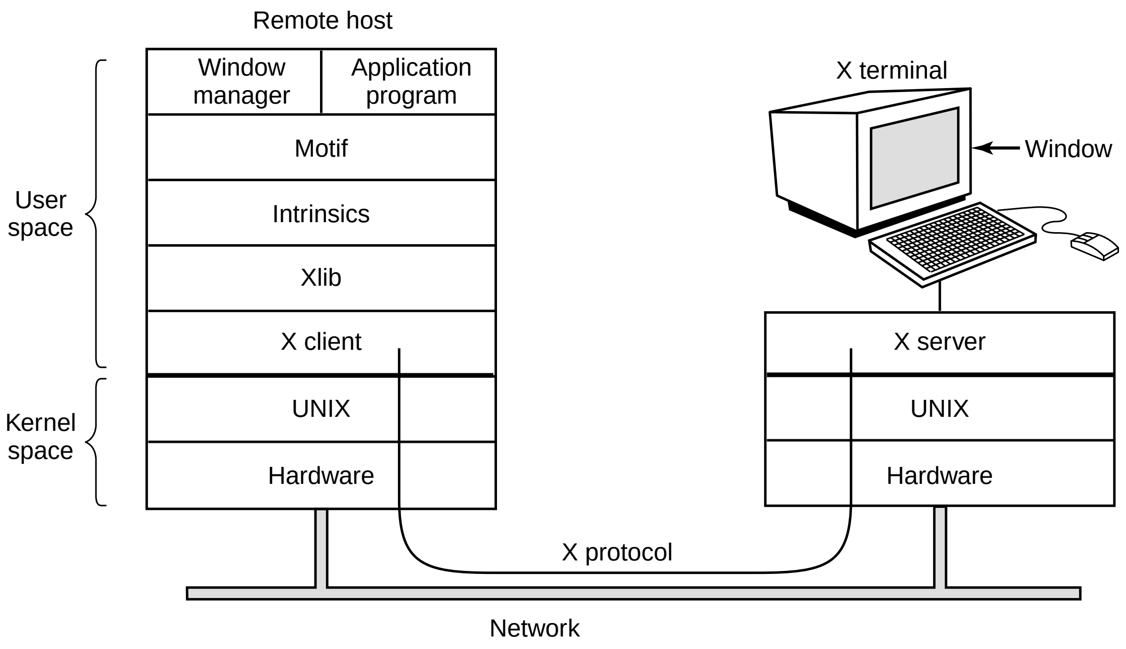


Fig. 5-45. Clients and servers in the M.I.T. X Window System.

```
#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 */

    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 */
}
```

Fig. 5-46. A skeleton of an X Window application program.

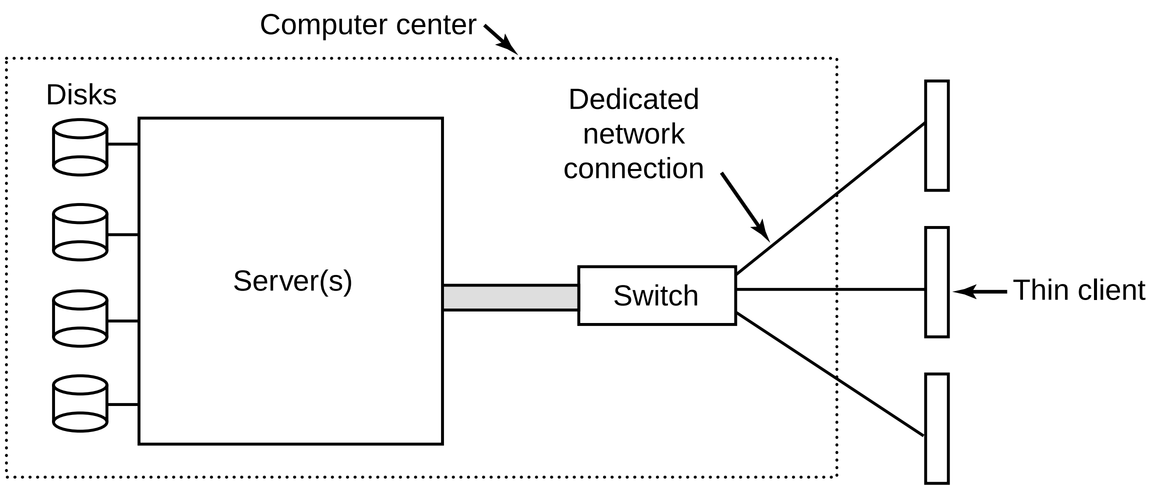


Fig. 5-47. The architecture of the SLIM terminal system.

Message	**Meaning**
SET	Update a rectangle with new pixels
FILL	Fill a rectangle with one pixel value
BITMAP	Expand a bitmap to fill a rectangle
COPY	Copy a rectangle from one part of the frame buffer to another
CSCS	Convert a rectangle from television color (YUV) to RGB

Fig. 5-48. Messages used in the SLIM protocol from the server to the terminals.

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%

Fig. 5-49. Power consumption of various parts of a laptop computer.

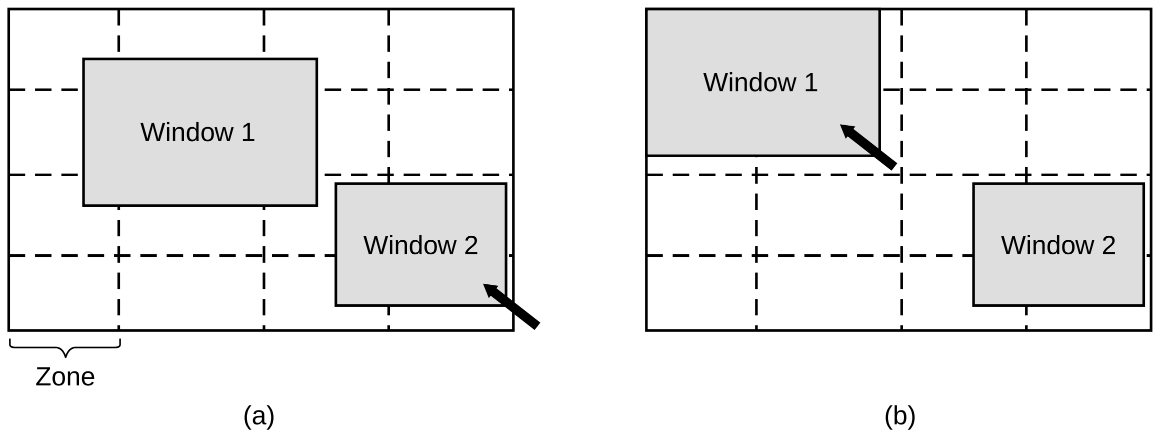


Fig. 5-50. The use of zones for backlighting the display. (a) When window 2 is selected it is not moved. (b) When window 1 is selected, it moves to reduce the number of zones illuminated.

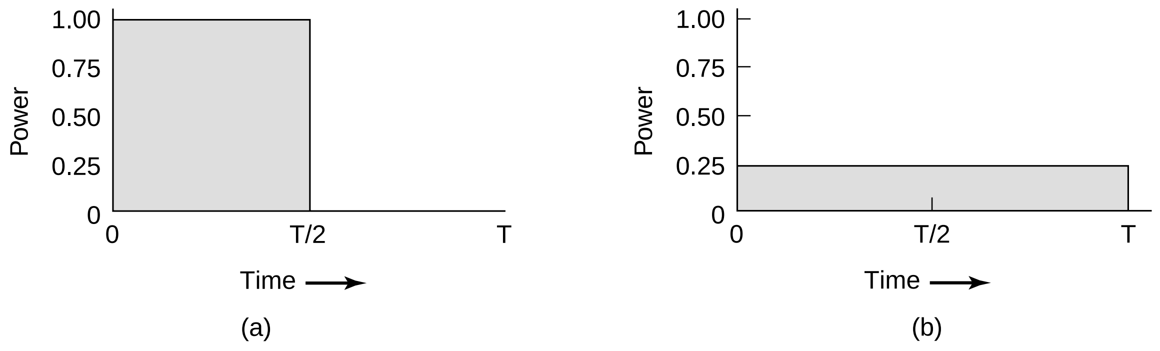


Fig. 5-51. (a) Running at full clock speed. (b) Cutting voltage by two cuts clock speed by two and power consumption by four.