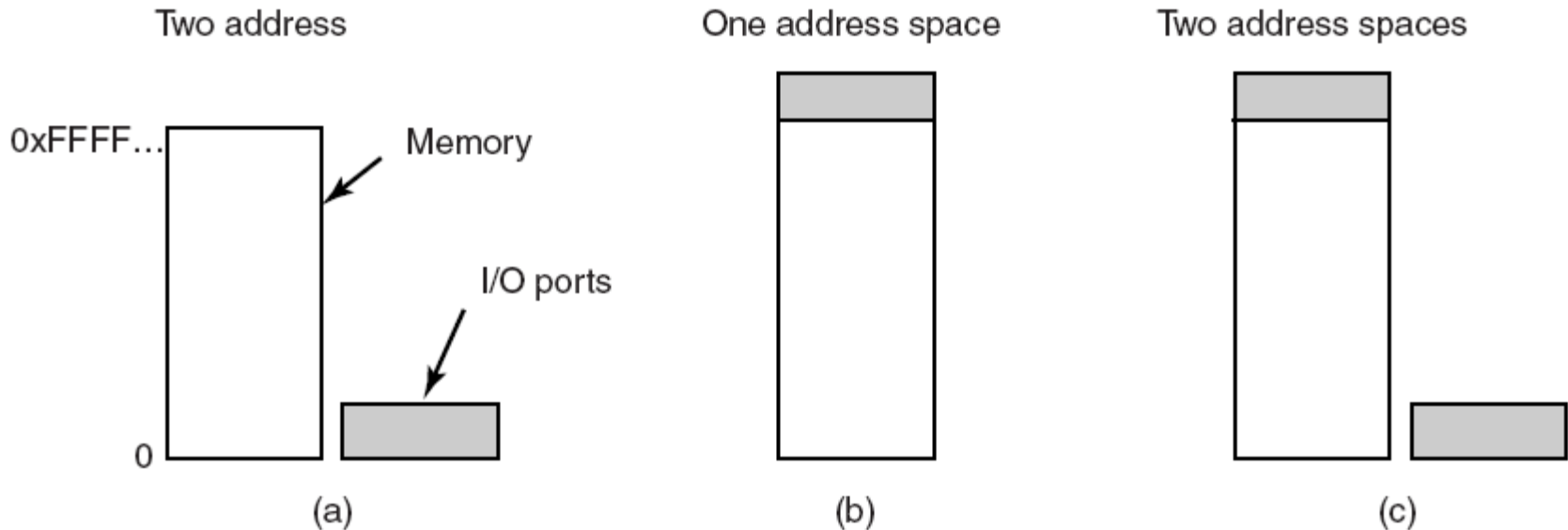


Input / Output

# I/O Devices

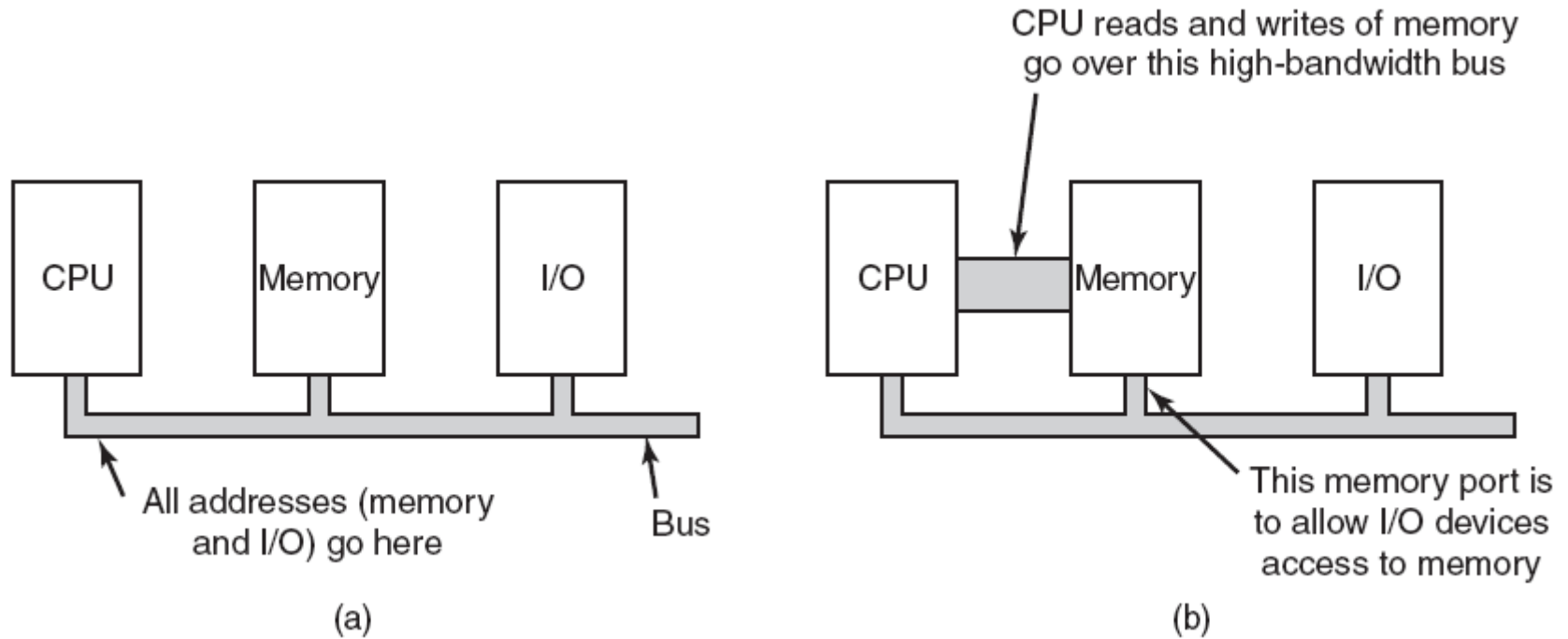
Device	Data rate
Keyboard	10 bytes/sec
Mouse	100 bytes/sec
56K modem	7 KB/sec
Scanner	400 KB/sec
Digital camcorder	3.5 MB/sec
802.11g Wireless	6.75 MB/sec
52x CD-ROM	7.8 MB/sec
Fast Ethernet	12.5 MB/sec
Compact flash card	40 MB/sec
FireWire (IEEE 1394)	50 MB/sec
USB 2.0	60 MB/sec
SONET OC-12 network	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

# Memory-Mapped I/O (1)



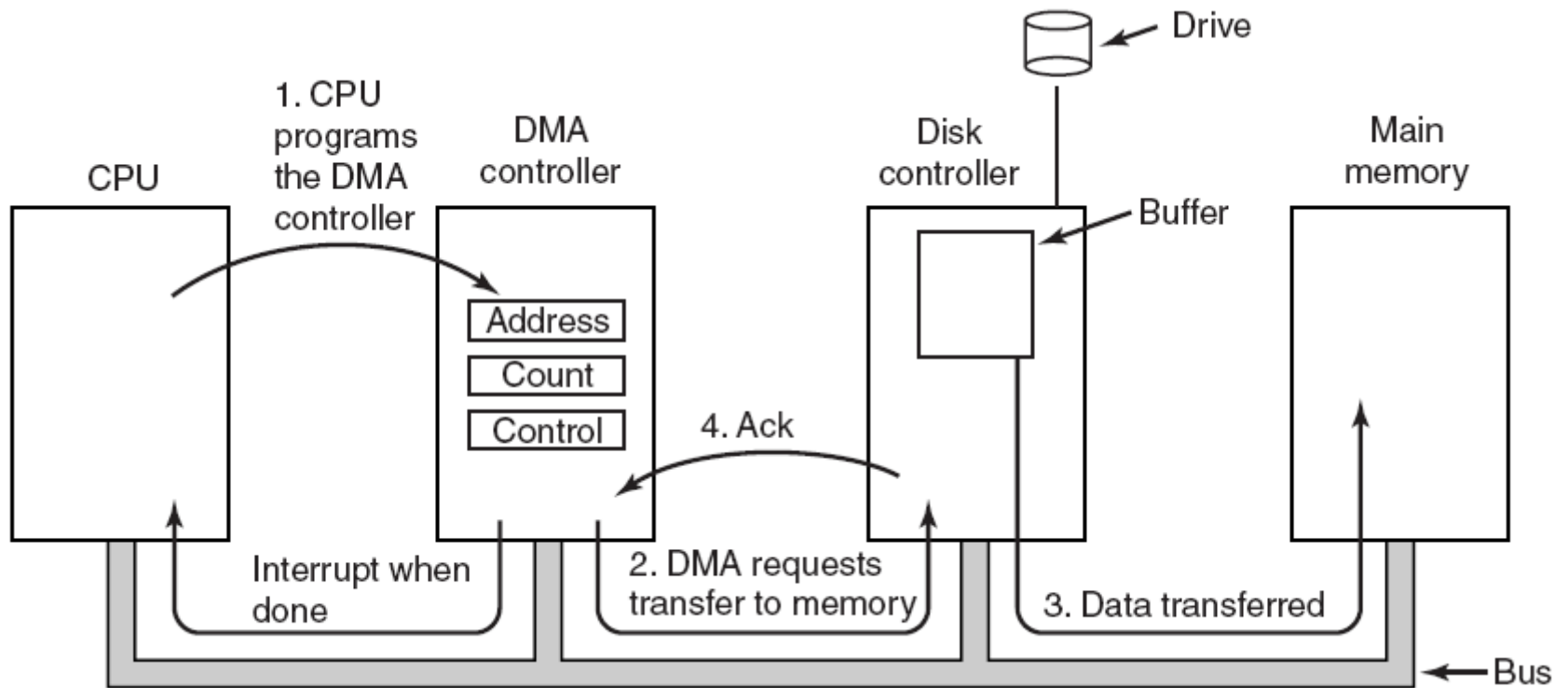
- (a) Separate I/O and memory space.
- (b) Memory-mapped I/O. (c) Hybrid.

# Memory-Mapped I/O (2)



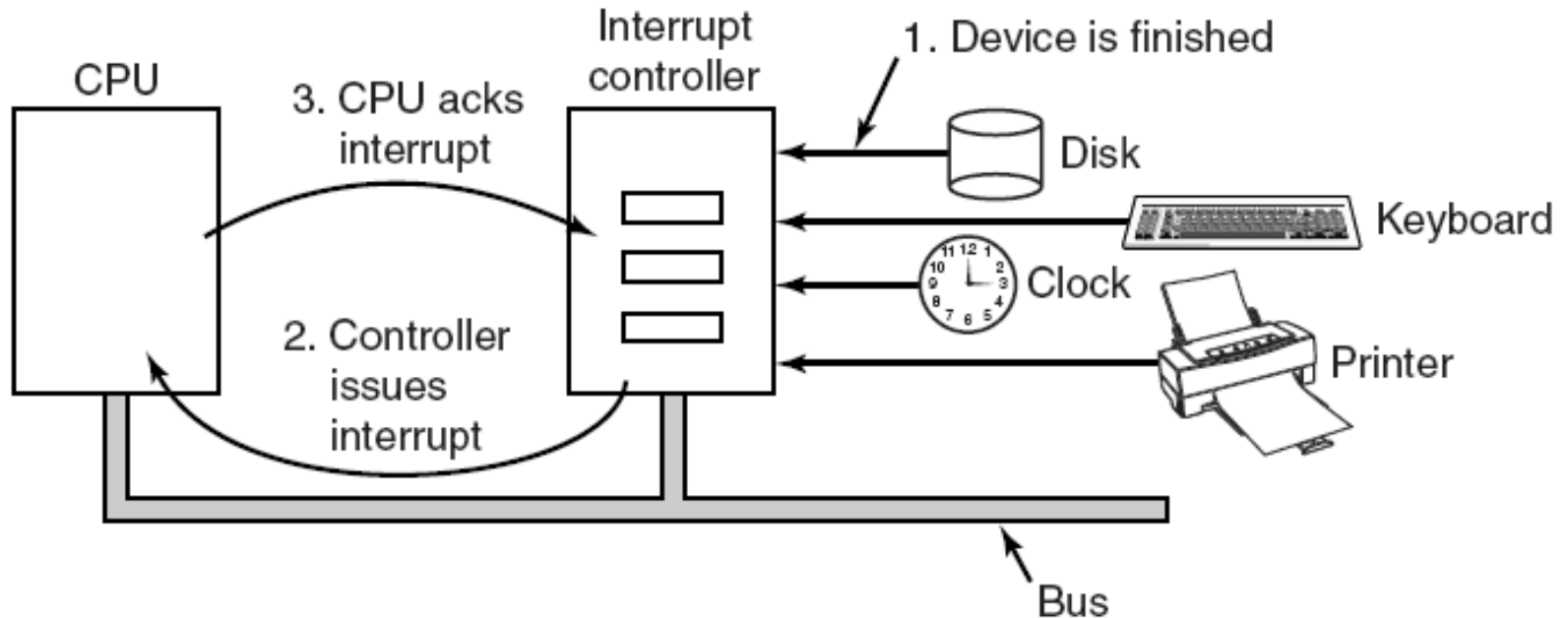
- (a) A single-bus architecture.
- (b) A dual-bus memory architecture.

# Direct Memory Access (DMA)



- Operation of a DMA transfer.

# Interrupts Revisited

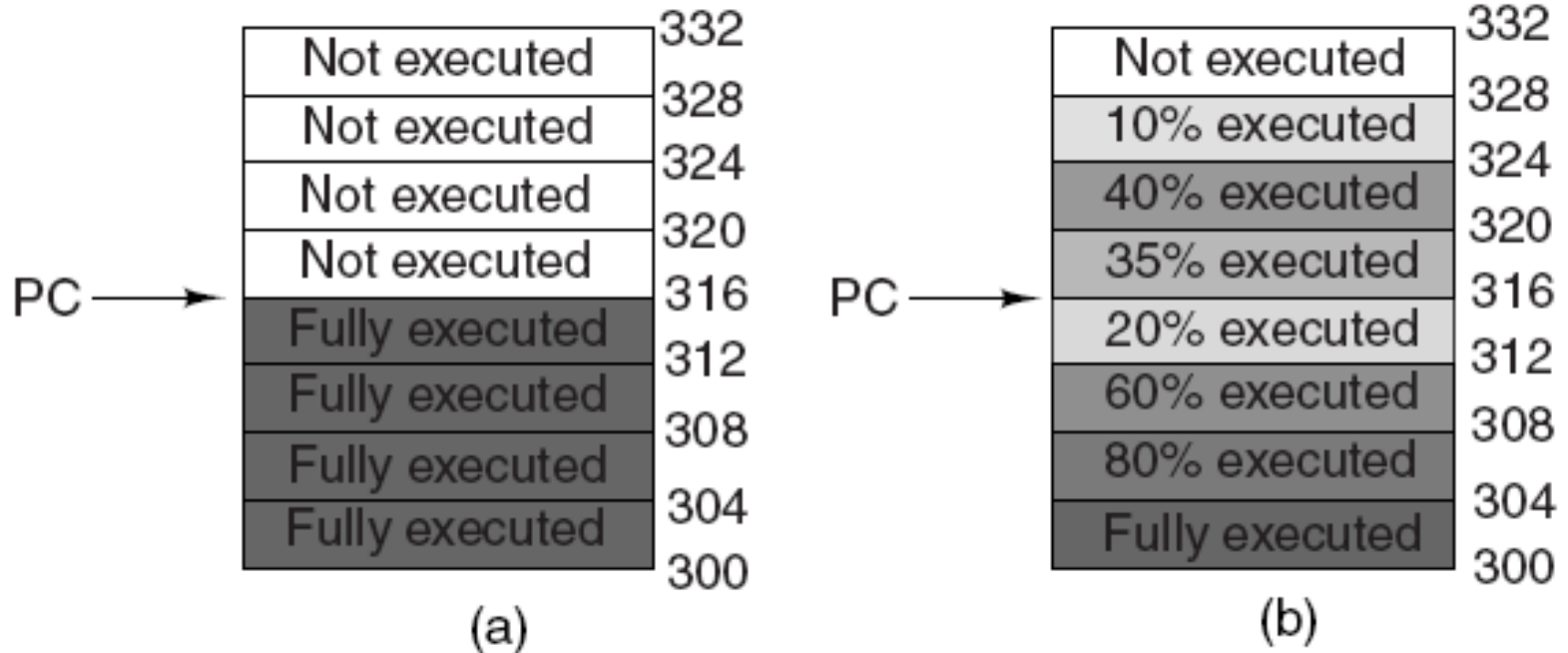


- How an interrupt happens. The connections between the devices and the interrupt controller actually use interrupt lines on the bus rather than dedicated wires.

# 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.

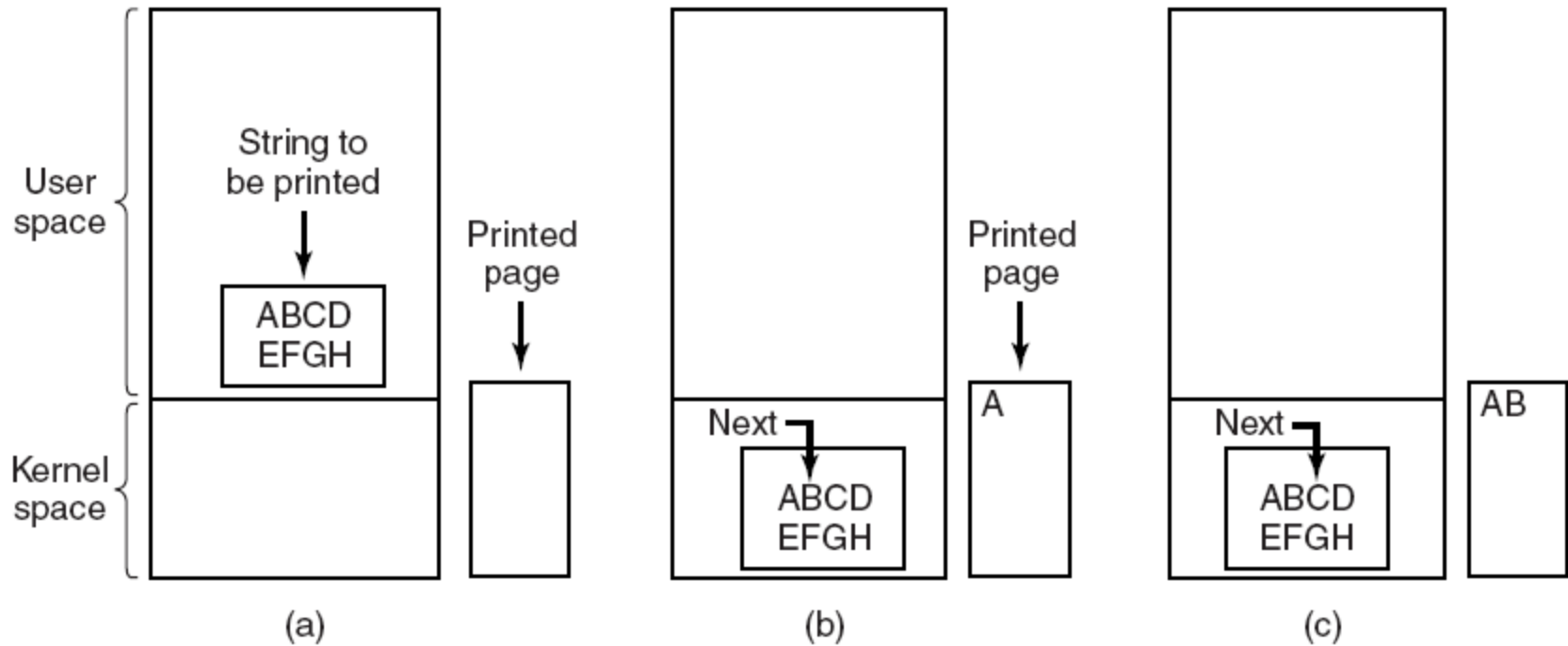
# Precise and Imprecise Interrupts (2)



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



# 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.
- (a) Code executed at the time the print system call is made.
- (b) Interrupt service procedure for the printer.

# I/O Using DMA

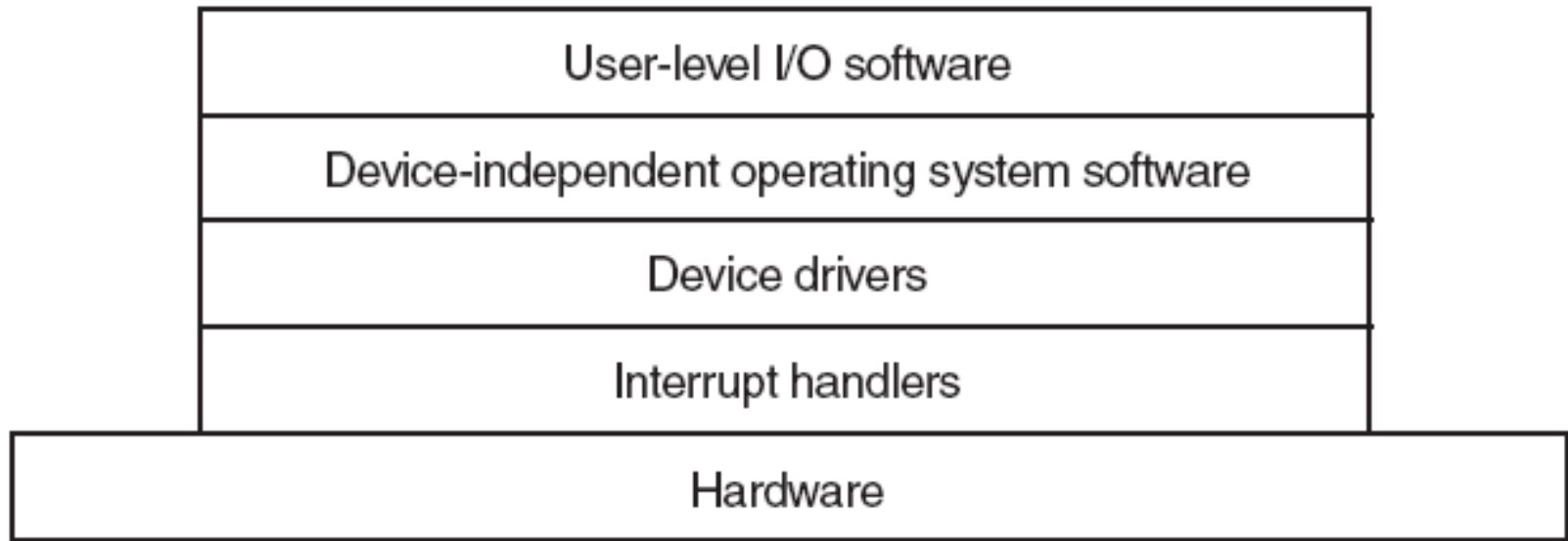
```
copy_from_user(buffer, p, count);  
set_up_DMA_controller( );  
scheduler( );
```

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

(b)

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

# I/O Software Layers



- Layers of the I/O software system.

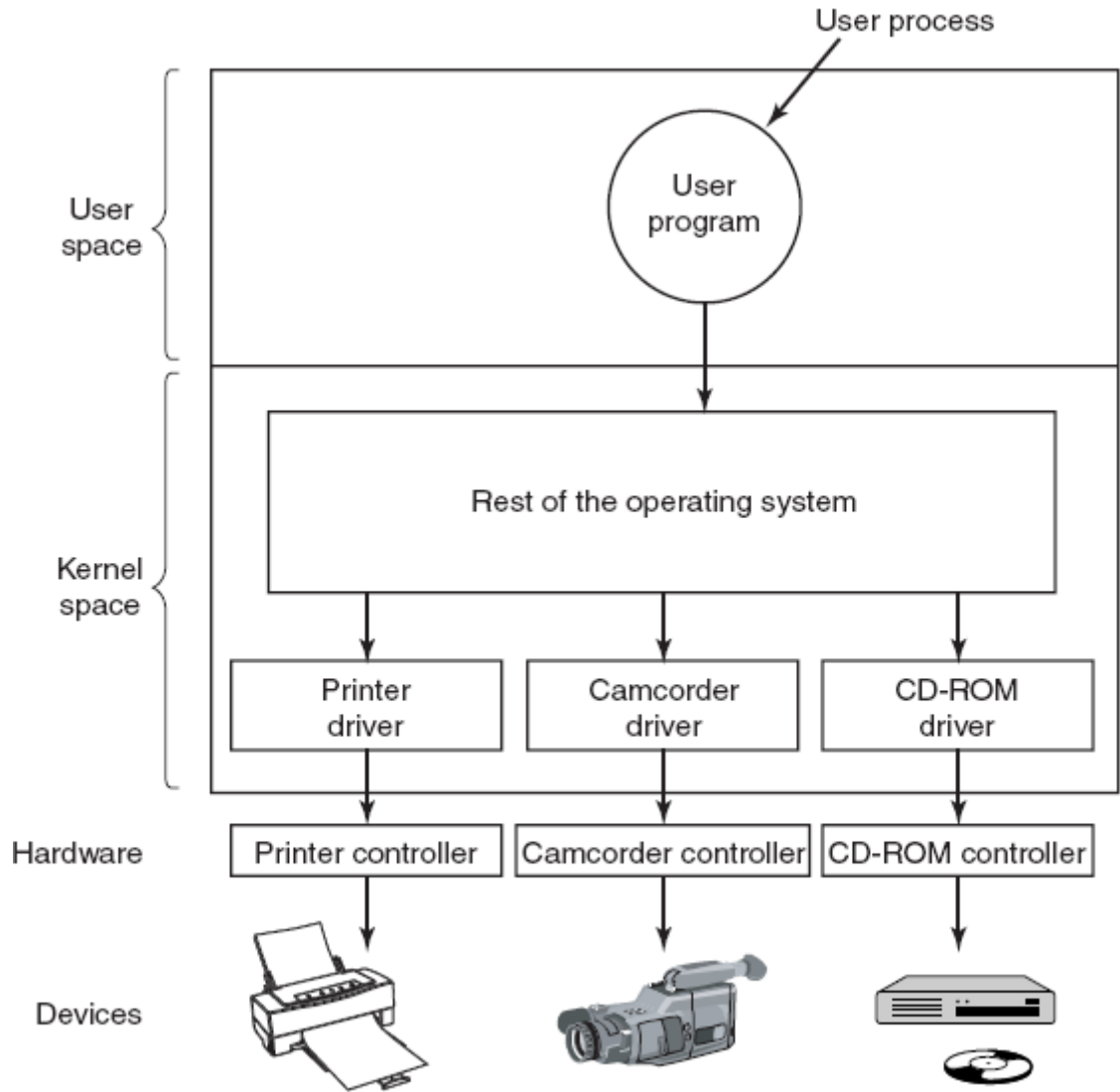
# Interrupt Handlers (1)

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, reenale 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. In reality all communication between drivers and device controllers goes over the bus.

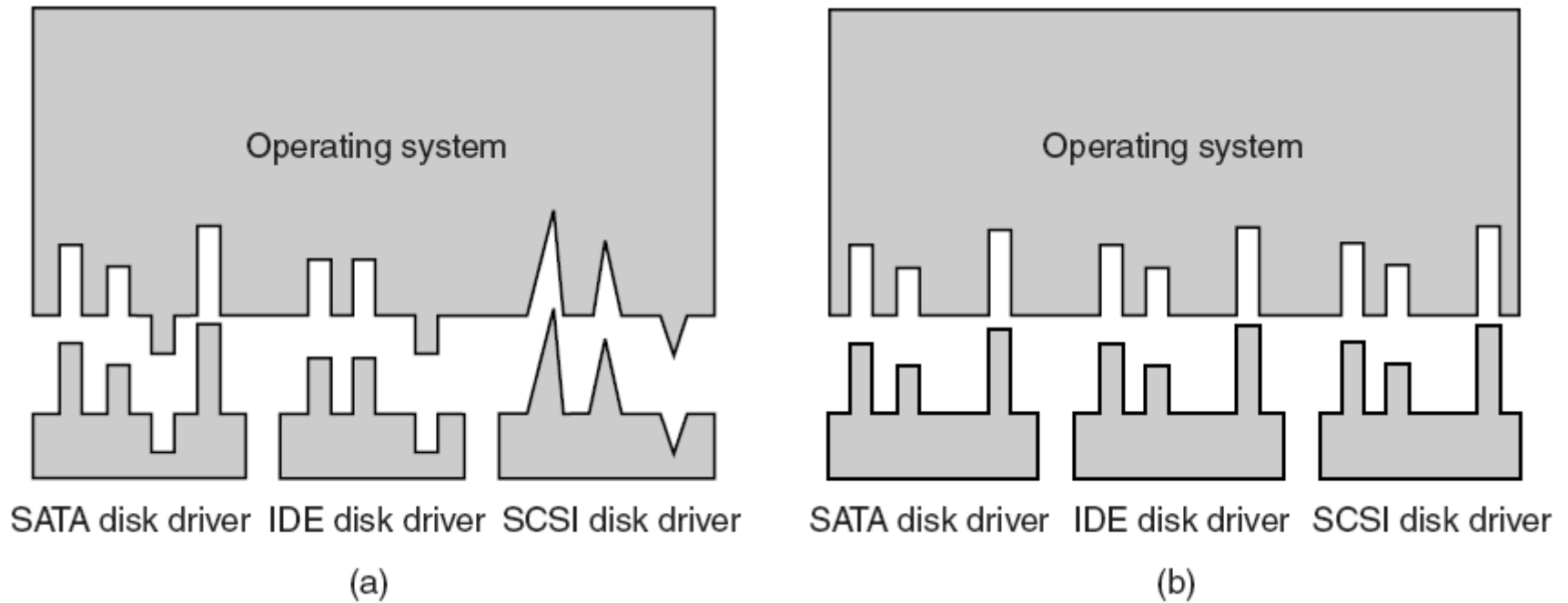


# Device-Independent I/O Software

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

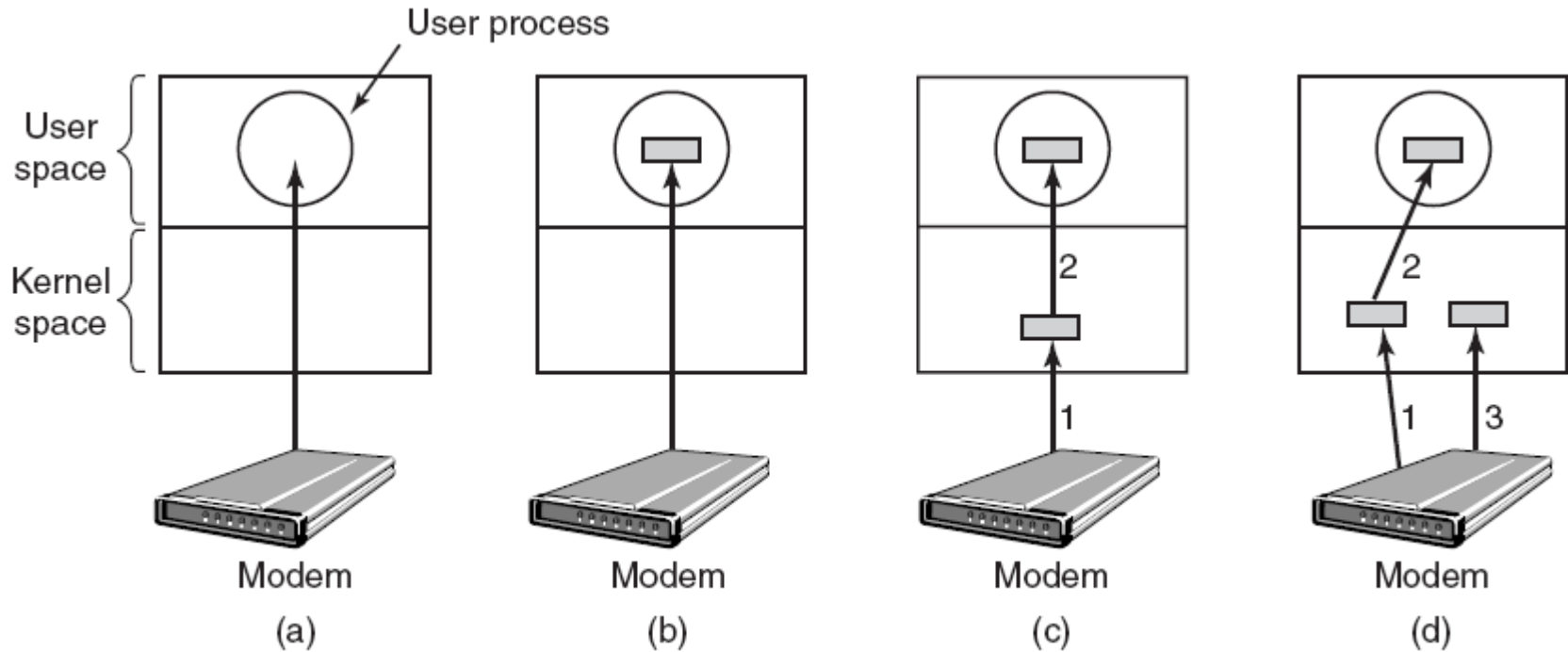
- Functions of the device-independent I/O software.

# Uniform Interfacing for Device Drivers



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

# Buffering (1)



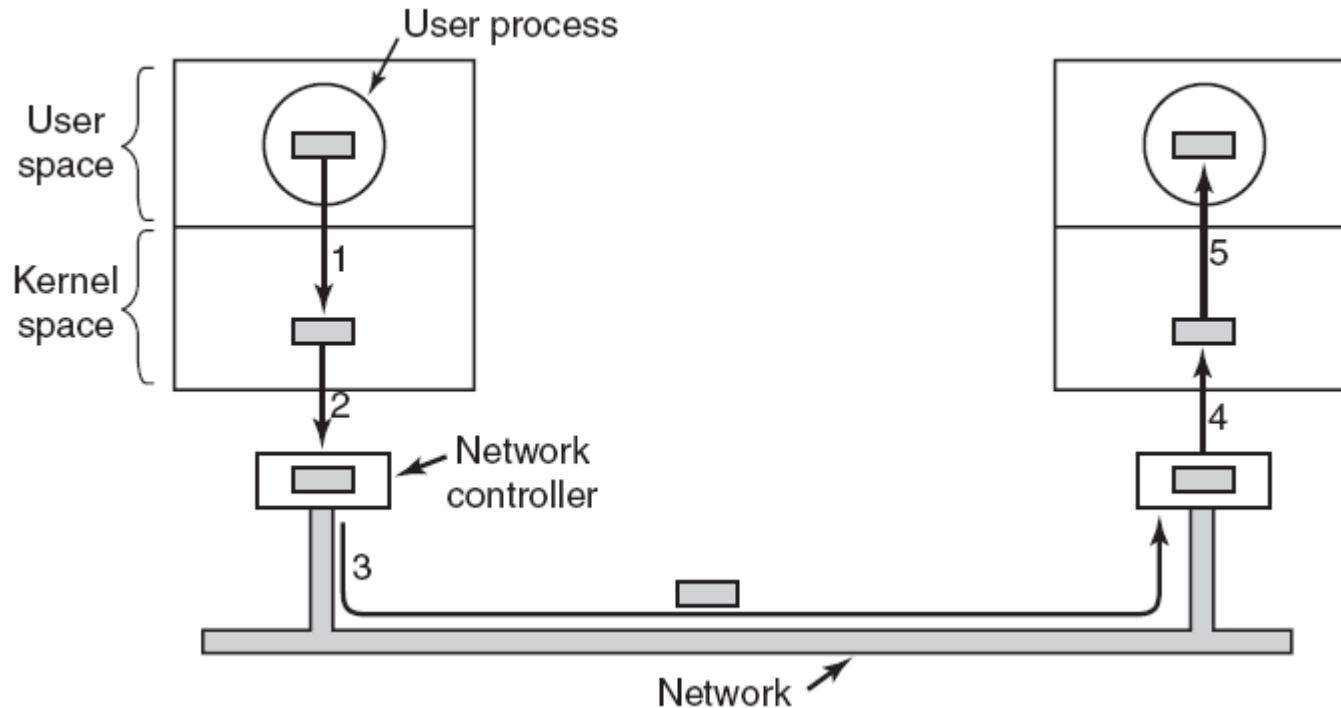
(a) Unbuffered input.

(b) Buffering in user space.

(c) Buffering in the kernel followed by copying to user space.

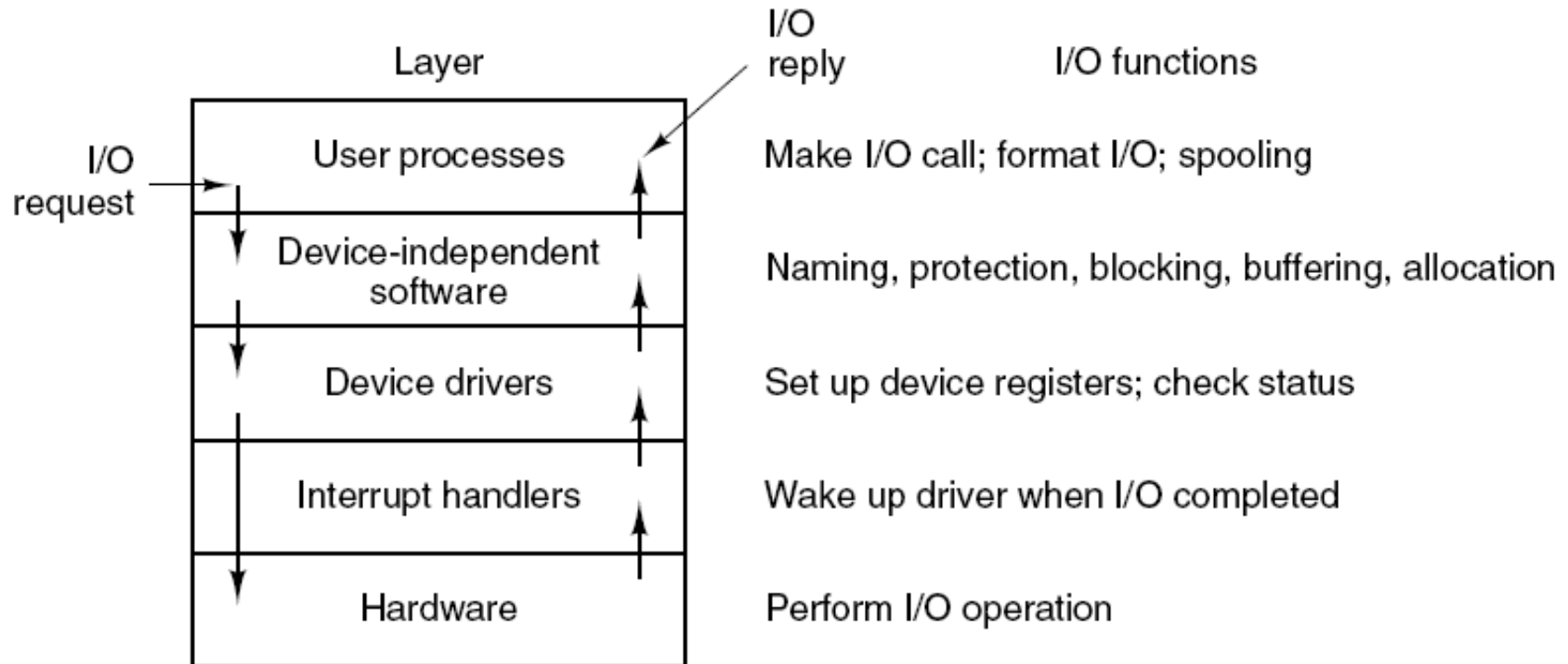
(d) Double buffering in the kernel.

# Buffering (2)



- Networking may involve many copies of a packet.

# User-Space I/O Software



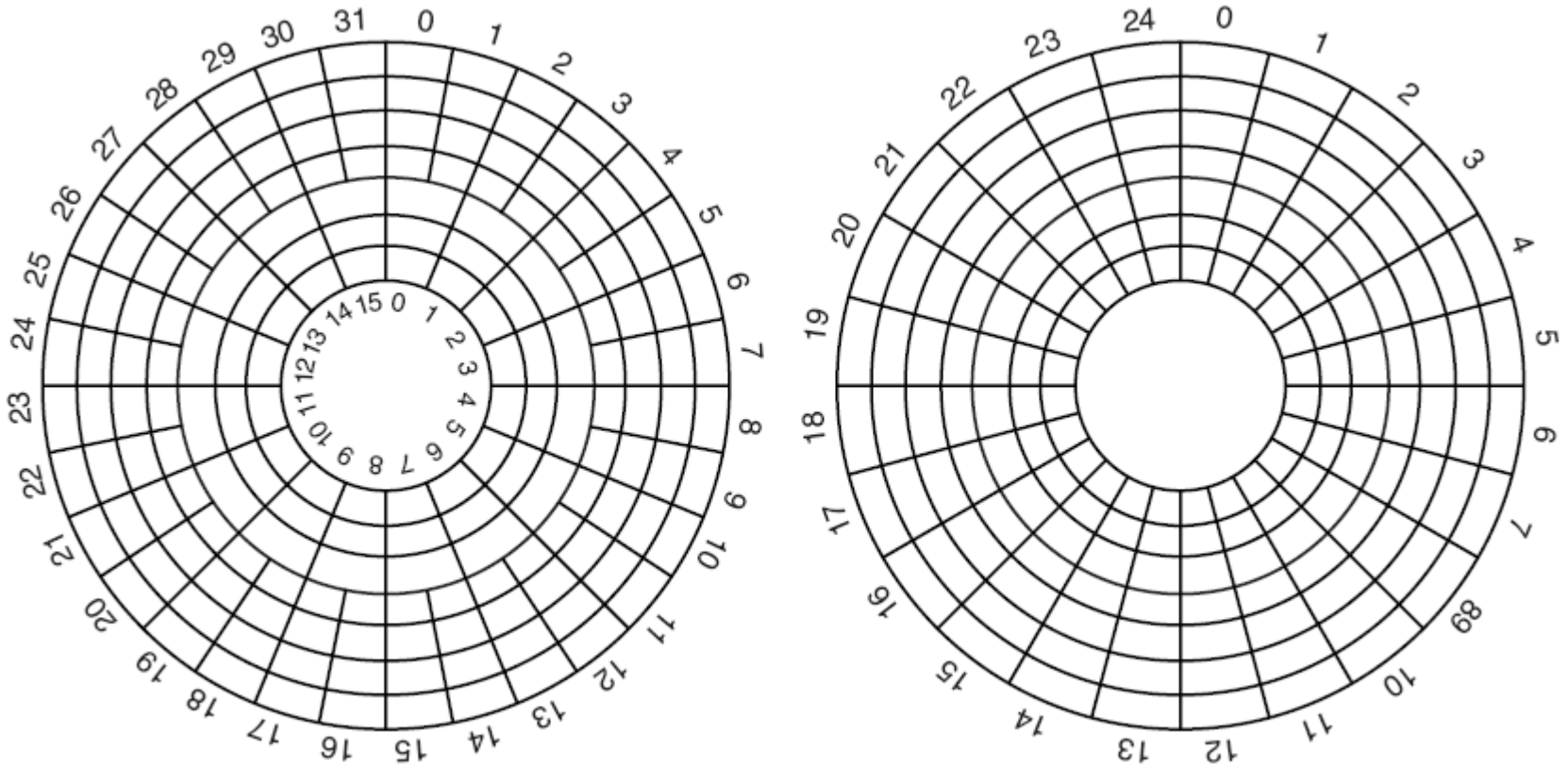
- Layers of the I/O system and the main functions of each layer.

# 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 $\mu$ sec

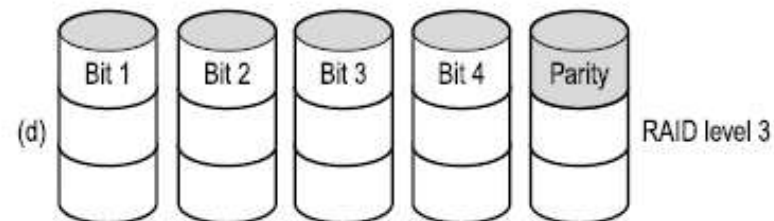
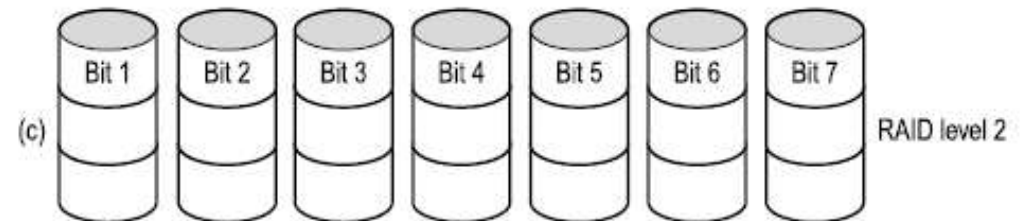
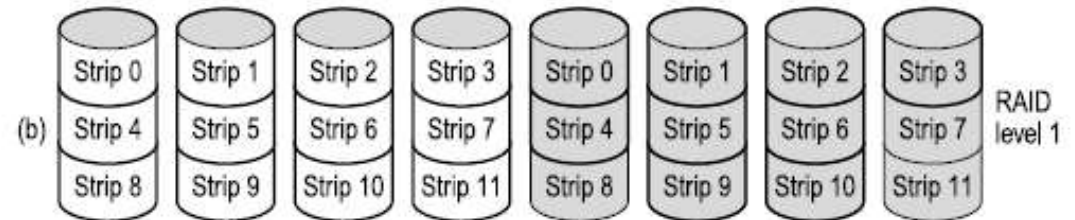
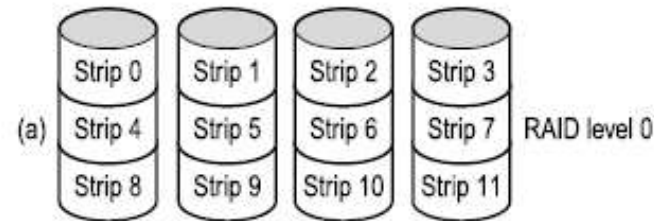
- Disk parameters for the original IBM PC 360-KB floppy disk and a Western Digital WD 18300 hard disk (18.2 GB, 1999)

# Magnetic Disks (2)



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

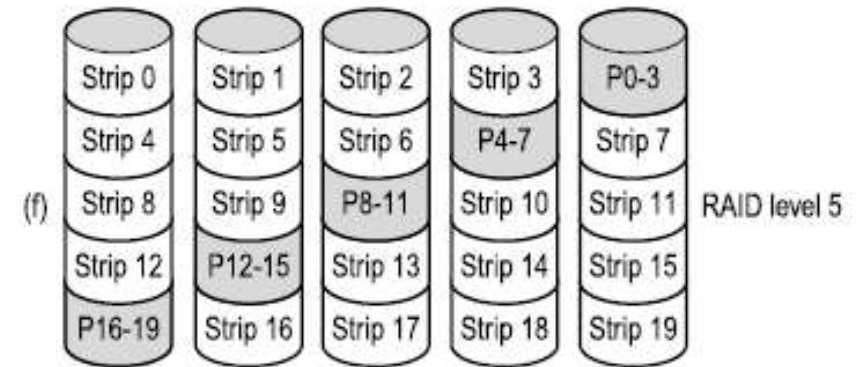
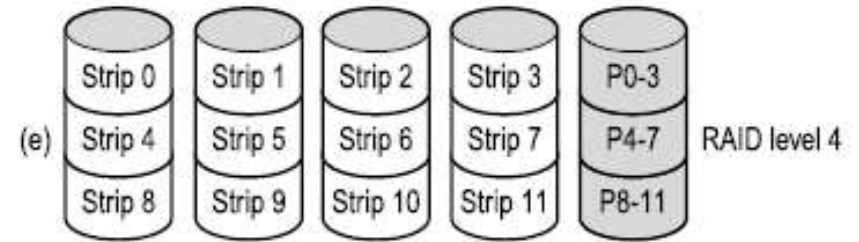
# RAID (1)



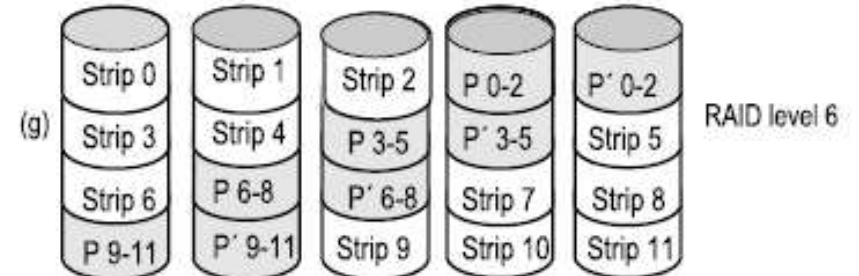
- RAID levels 0 through 3. Backup and parity drives are shown shaded.



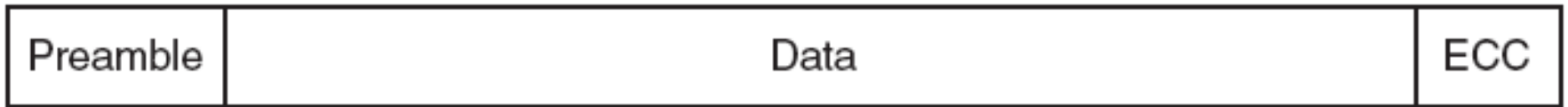
# RAID (2)



- RAID levels 4 through 6. Backup and parity drives are shown shaded.

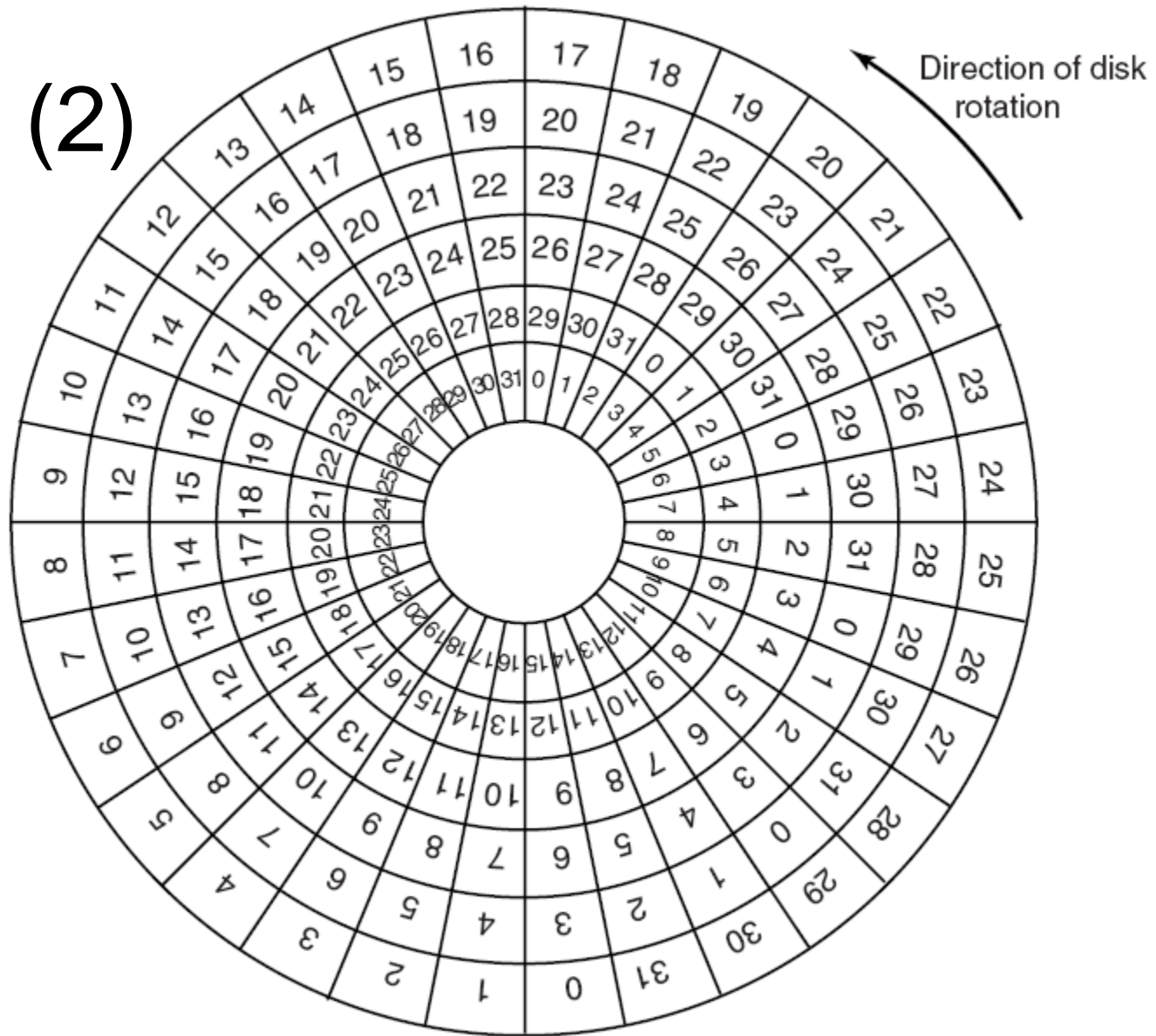


# Disk Formatting (1)



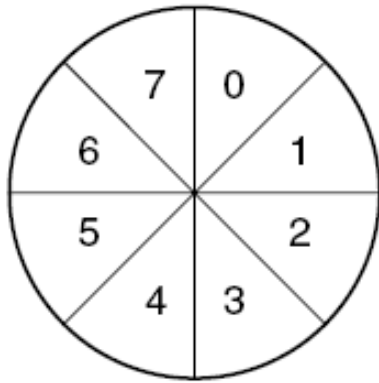
- A disk sector.

# Disk Formatting (2)

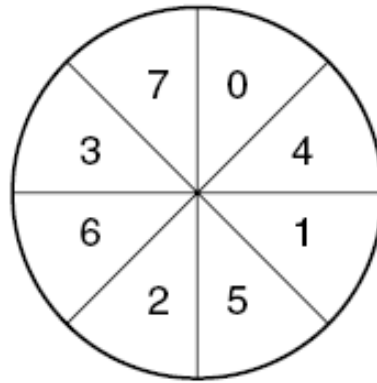


- An illustration of cylinder skew.

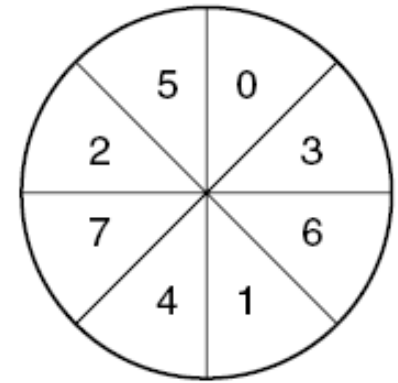
# Disk Formatting (3)



(a)



(b)



(c)

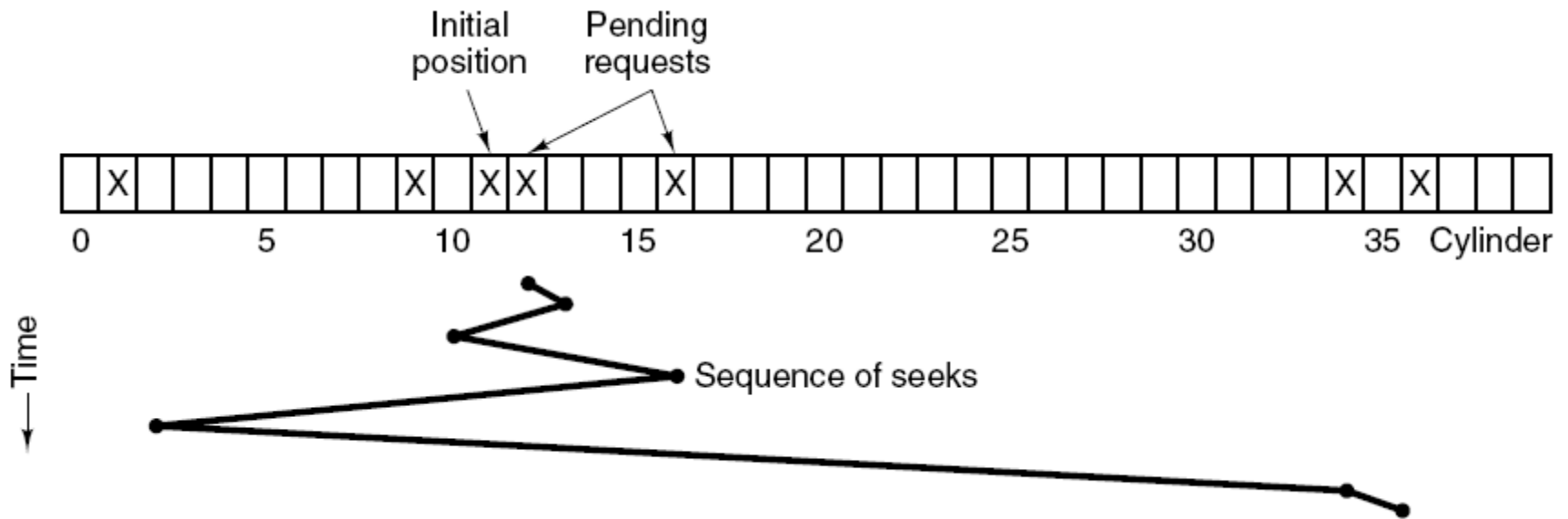
- (a) No interleaving. (b) Single interleaving.  
(c) Double interleaving.

# Disk Arm Scheduling Algorithms (1)

Read/write time factors

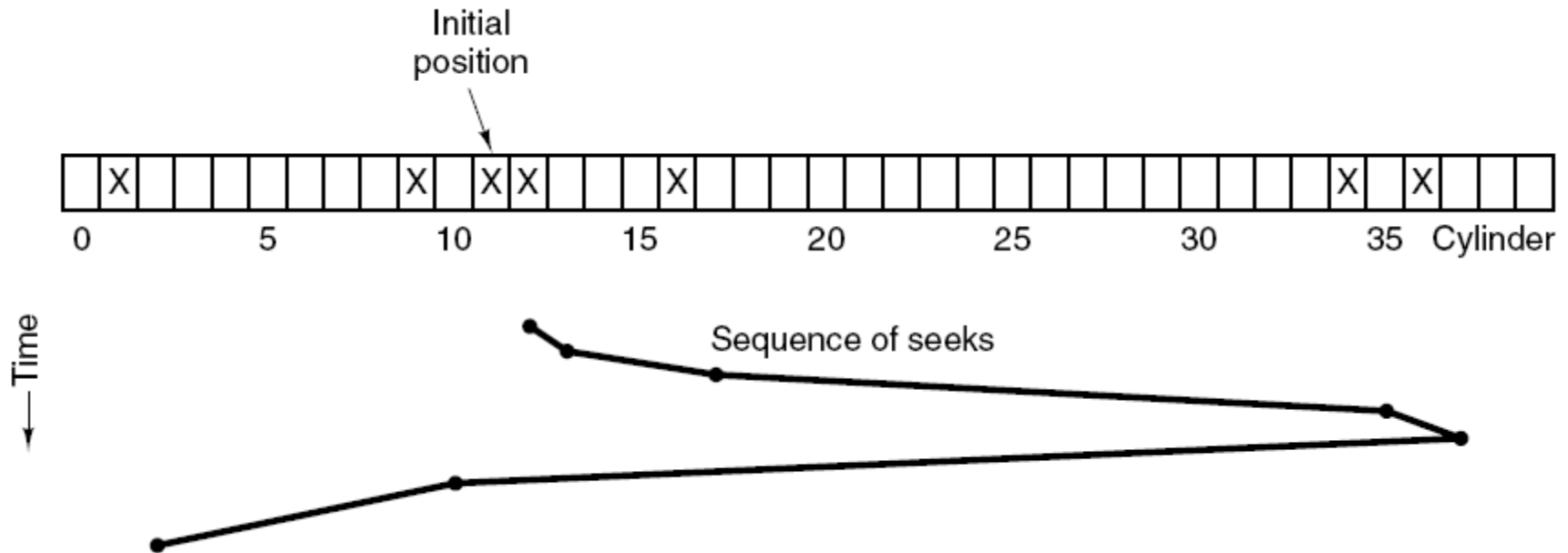
1. Seek time (the time to move the arm to the proper cylinder).
2. Rotational delay (the time for the proper sector to rotate under the head).
3. Actual data transfer time.

# Disk Arm Scheduling Algorithms (2)



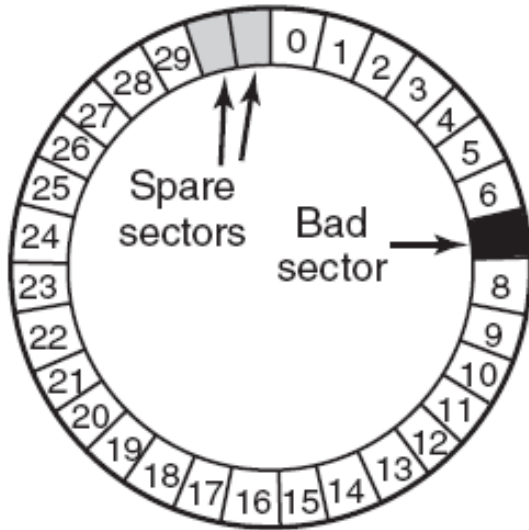
- Shortest Seek First (SSF) disk scheduling algorithm.

# Disk Arm Scheduling Algorithms (3)

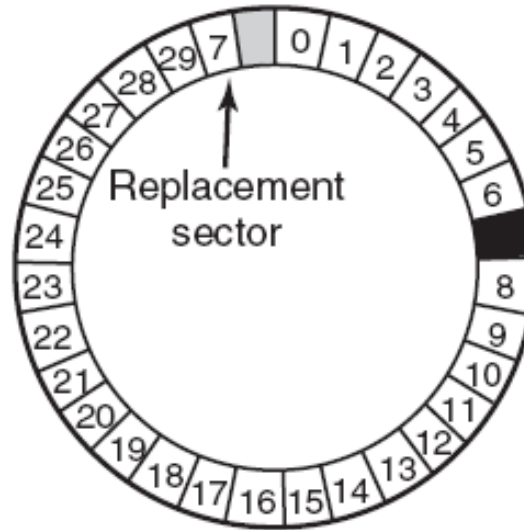


- The elevator algorithm for scheduling disk requests.

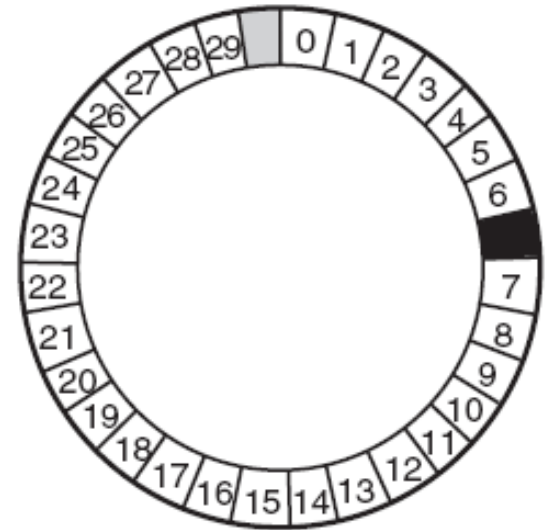
# Error Handling



(a)



(b)



(c)

- (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.

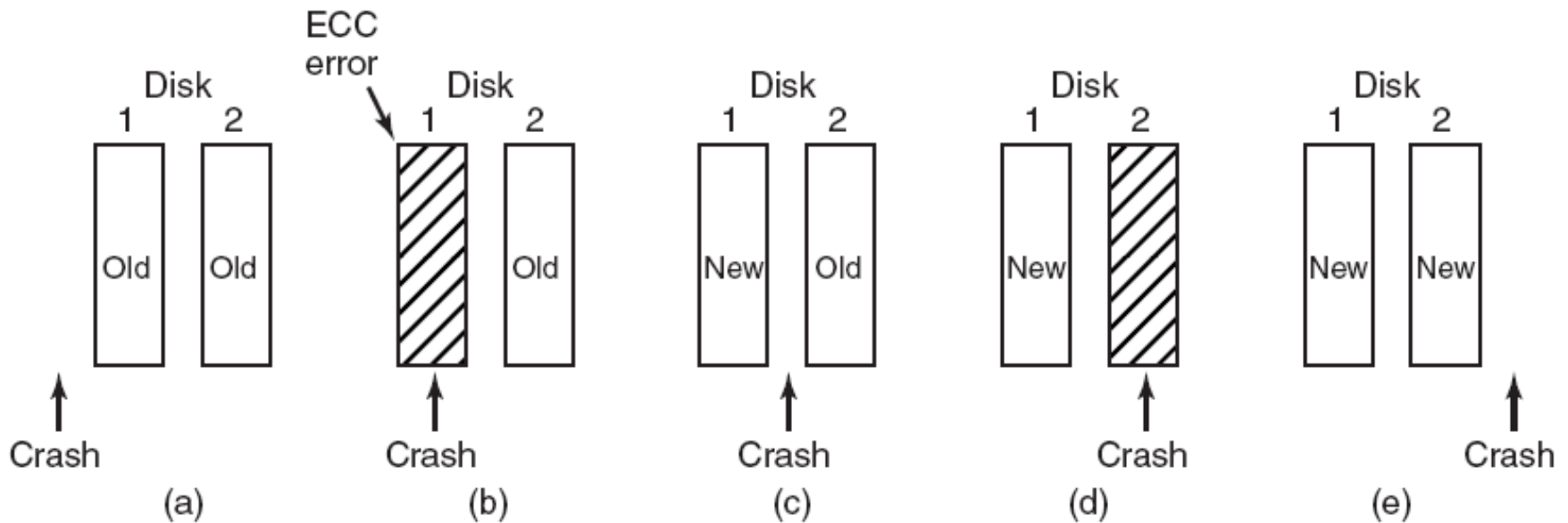


# Stable Storage (1)

Operations for stable storage using identical disks:

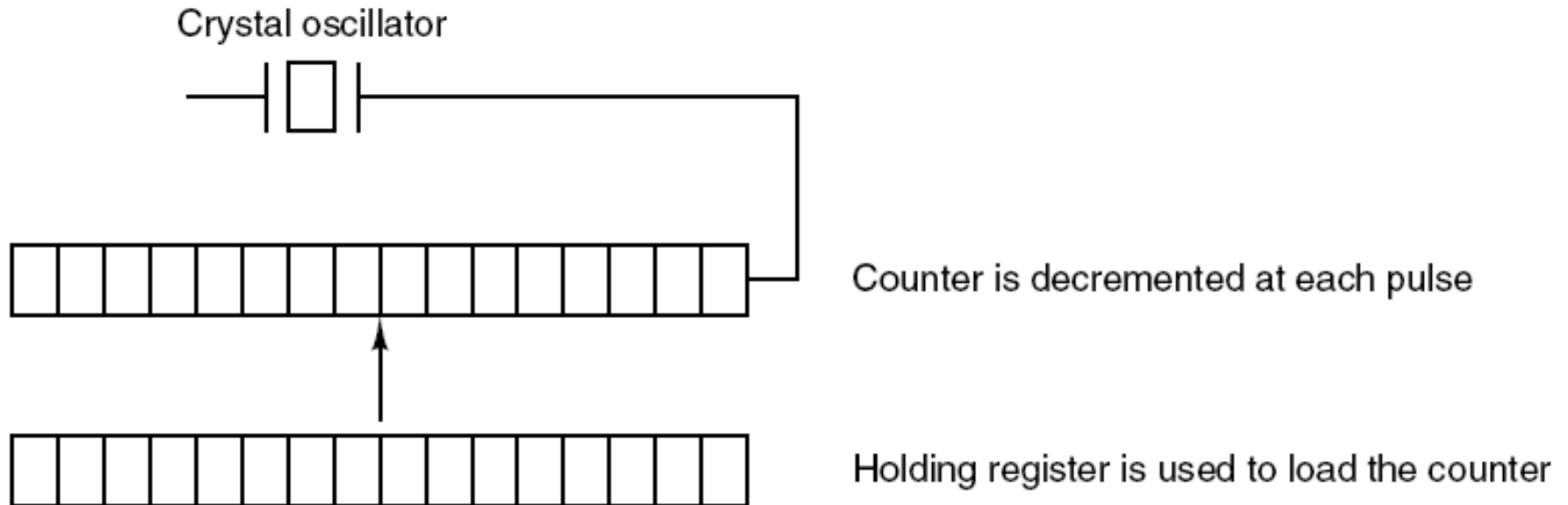
1. Stable writes
2. Stable reads
3. Crash recovery

# Stable Storage (2)



- Analysis of the influence of crashes on stable writes.

# Clock Hardware



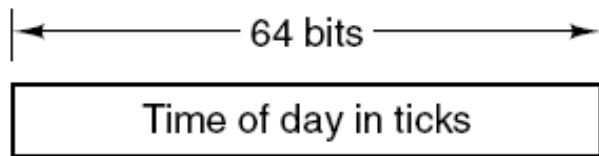
- A programmable clock.

# Clock Software (1)

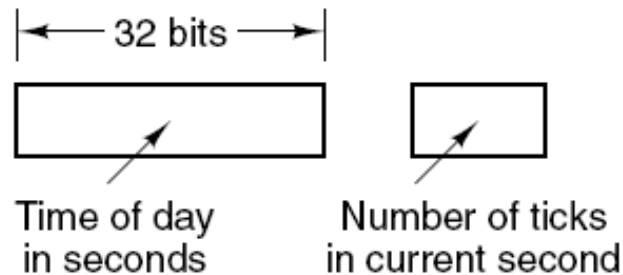
Typical duties of a clock driver

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.

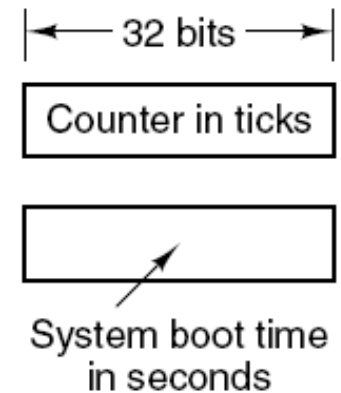
# Clock Software (2)



(a)



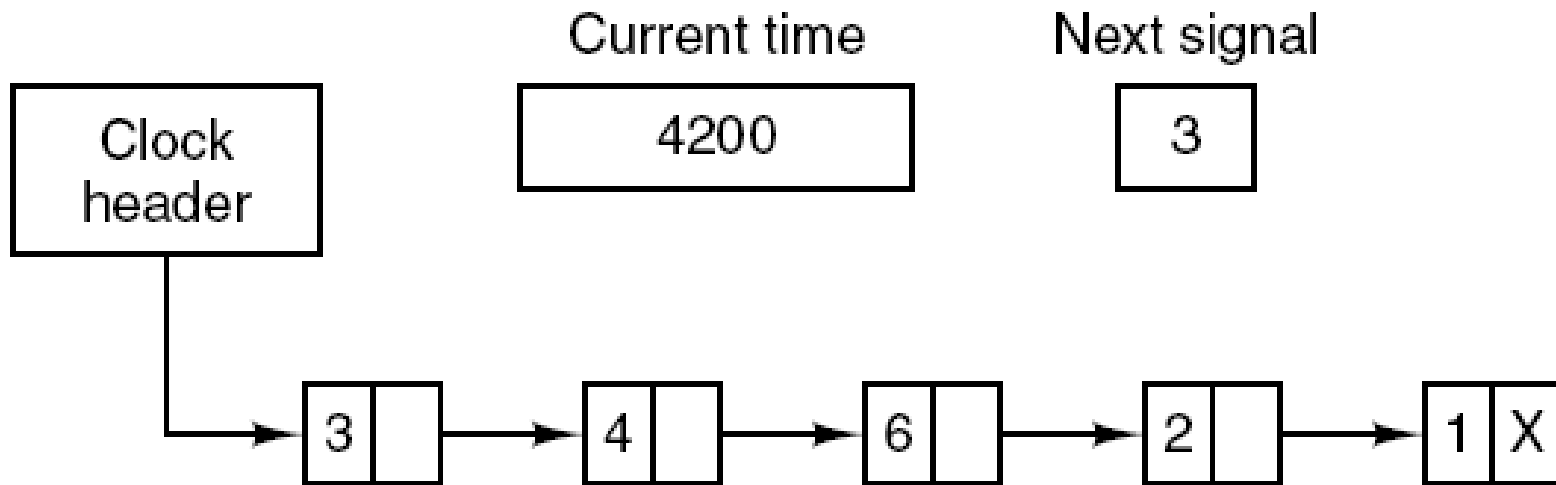
(b)



(c)

- Three ways to maintain the time of day.

# Clock Software (3)



- Simulating multiple timers with a single clock.

# Soft Timers

Soft timers succeed according to rate at which kernel entries are made because of:

1. System calls.
2. TLB misses.
3. Page faults.
4. I/O interrupts.
5. The CPU going idle.

# Keyboard Software

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.

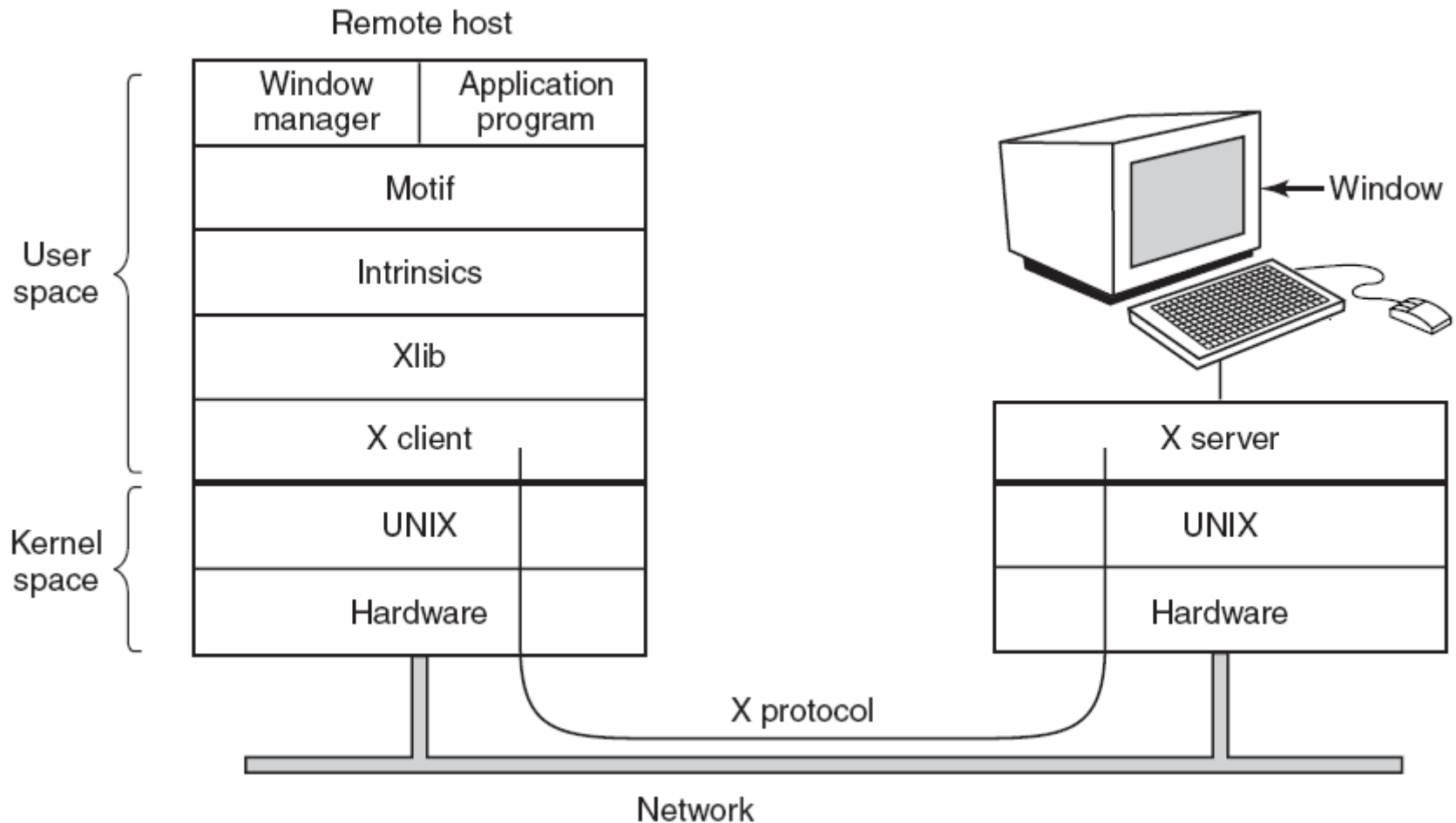


# The X Window System (1)

Escape sequence	Meaning
ESC [ <i>n</i> A	Move up <i>n</i> lines
ESC [ <i>n</i> B	Move down <i>n</i> lines
ESC [ <i>n</i> C	Move right <i>n</i> spaces
ESC [ <i>n</i> D	Move left <i>n</i> spaces
ESC [ <i>m</i> ; <i>n</i> H	Move cursor to ( <i>m</i> , <i>n</i> )
ESC [ <i>s</i> J	Clear screen from cursor (0 to end, 1 from start, 2 all)
ESC [ <i>s</i> K	Clear line from cursor (0 to end, 1 from start, 2 all)
ESC [ <i>n</i> L	Insert <i>n</i> lines at cursor
ESC [ <i>n</i> M	Delete <i>n</i> lines at cursor
ESC [ <i>n</i> P	Delete <i>n</i> chars at cursor
ESC [ <i>n</i> @	Insert <i>n</i> chars at cursor
ESC [ <i>n</i> m	Enable rendition <i>n</i> (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), and *n*, *m*, and *s* are optional numeric parameters.

# The X Window System (2)



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

# The X Window System (3)

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 */

    . . .
}
```

- A skeleton of an X Window application program.

# Graphical User Interfaces (2)

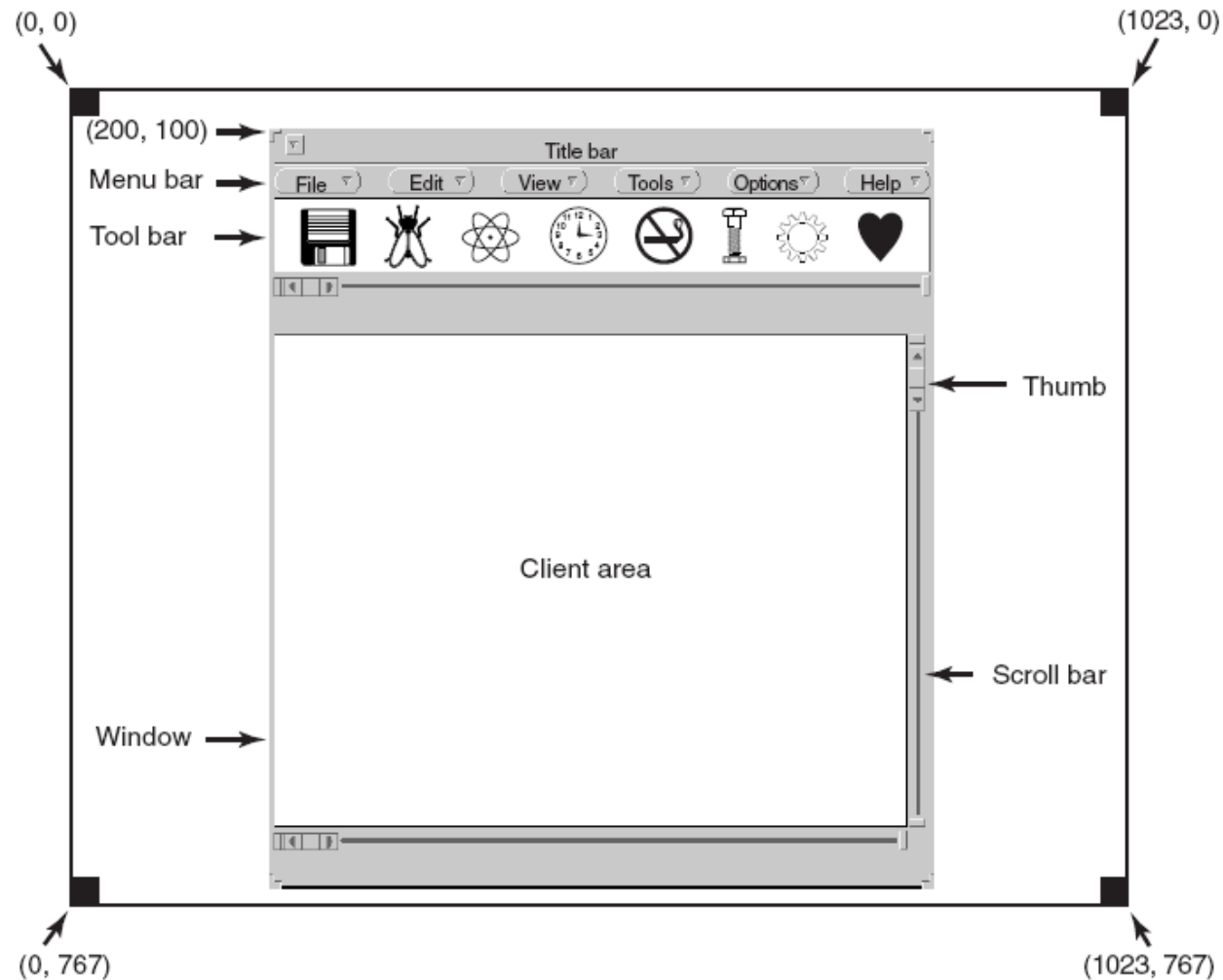
...

```
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 */
}
```

- 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);
}

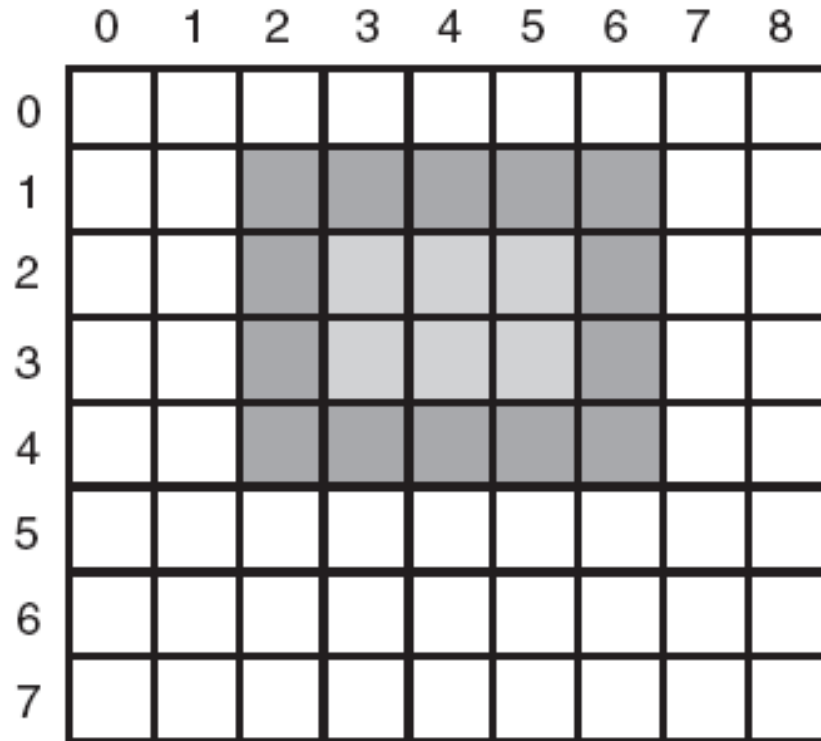
long CALLBACK WndProc(HWND hwnd, UINT message, UINT wParam, long lParam)
{
    /* Declarations go here. */

    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.

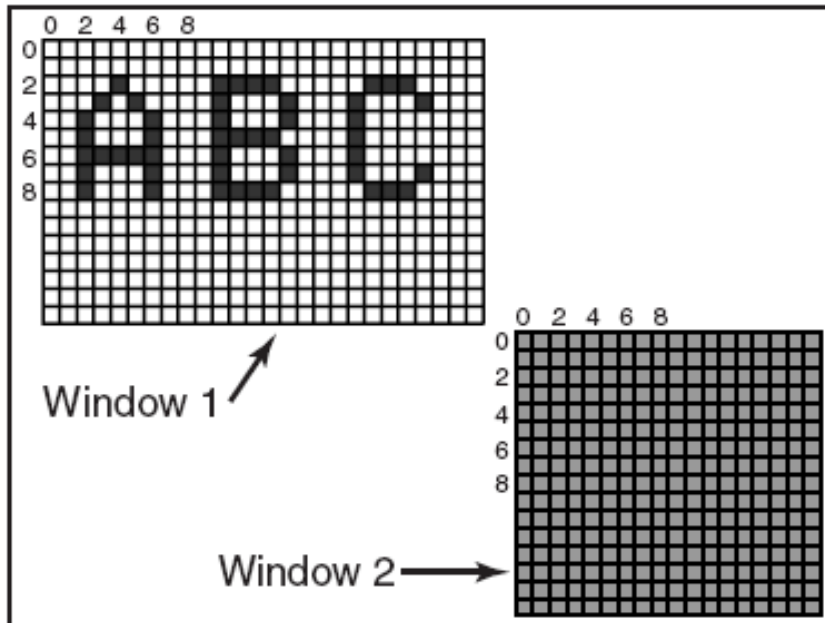


# Bitmaps (1)

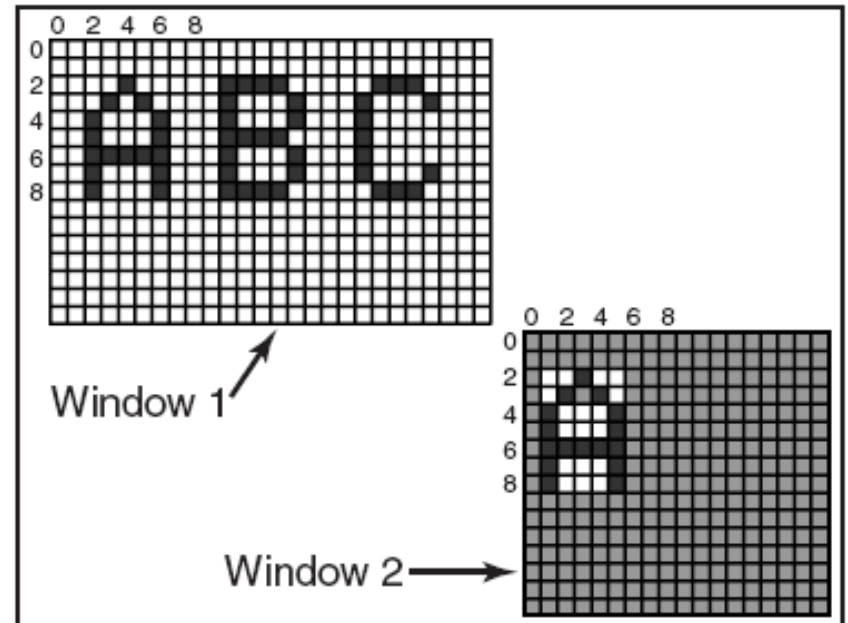


- An example rectangle drawn using Rectangle. Each box represents one pixel.

# Bitmaps (2)



(a)



(b)

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

# Thin Clients

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.

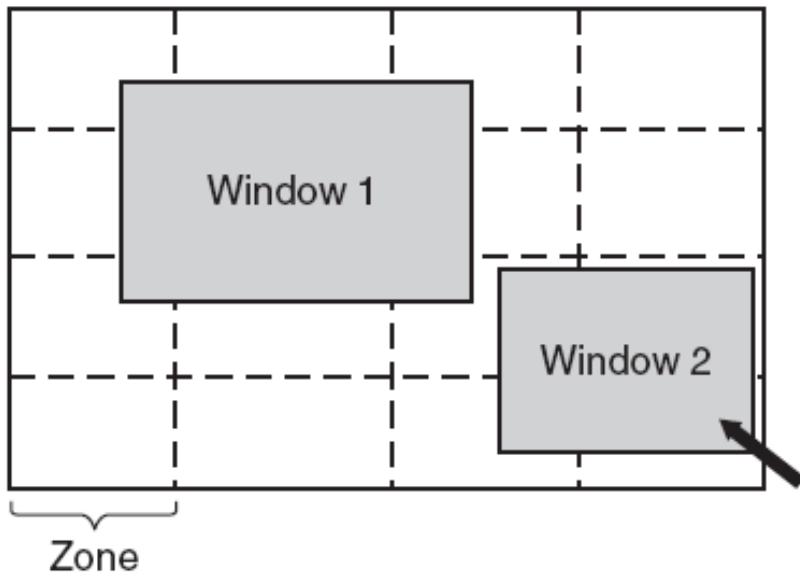
# Power Management Hardware Issues

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%

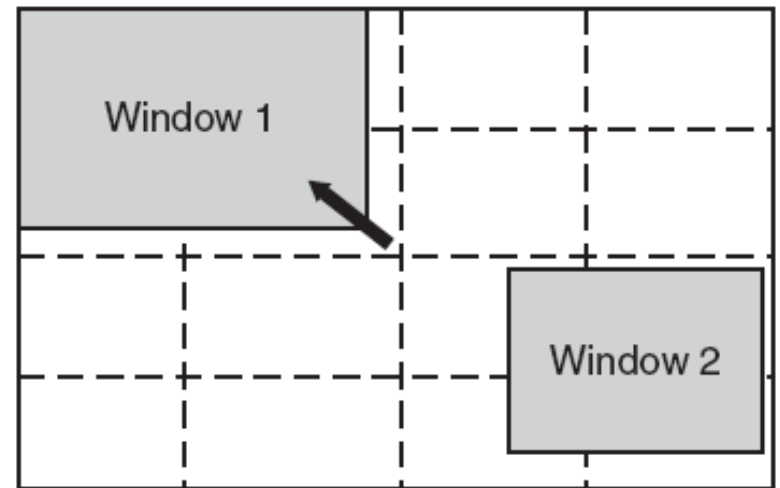
- Power consumption of various parts of a notebook computer.

# Power Management

## The Display



(a)

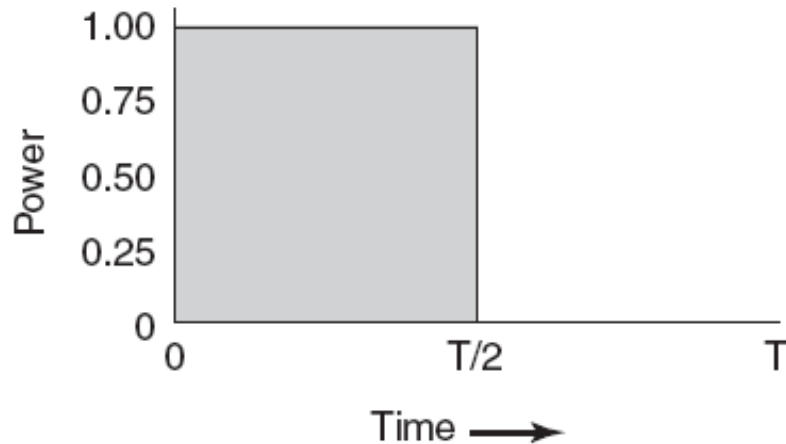


(b)

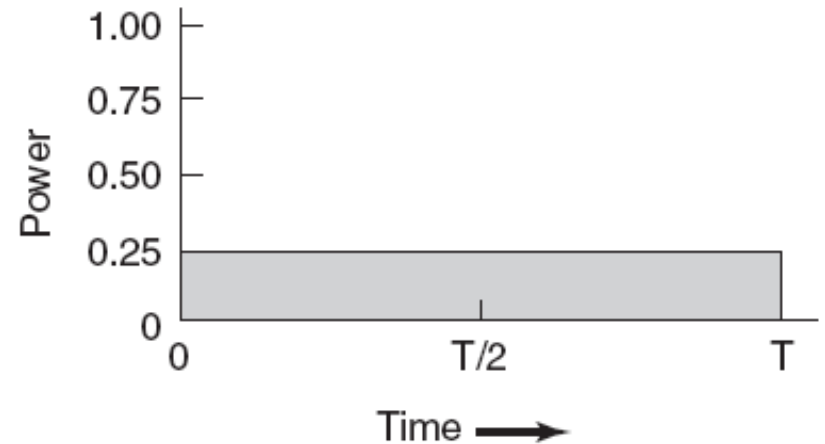
- The use of zones for backlighting the display.
- (a) When window 2 is selected it is not moved.
- (b) When window 1 is selected, it moves to reduce the number of zones illuminated.

# Power Management

## The CPU



(a)



(b)

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

# Deadlocks

# Preemptable and Nonpreemptable Resources

Sequence of events required to use a resource:

1. Request the resource.
2. Use the resource.
3. Release the resource.



# Resource Acquisition (1)

```
typedef int semaphore;  
semaphore resource_1;
```

```
void process_A(void) {  
    down(&resource_1);  
    use_resource_1( );  
    up(&resource_1);  
}
```

(a)

```
typedef int semaphore;  
semaphore resource_1;  
semaphore resource_2;
```

```
void process_A(void) {  
    down(&resource_1);  
    down(&resource_2);  
    use_both_resources( );  
    up(&resource_2);  
    up(&resource_1);  
}
```

(b)

Using a semaphore to protect resources.  
(a) One resource. (b) Two resources.

# Resource Acquisition (2)

```
typedef int semaphore;  
    semaphore resource_1;  
    semaphore resource_2;  
  
void process_A(void) {  
    down(&resource_1);  
    down(&resource_2);  
    use_both_resources( );  
    up(&resource_2);  
    up(&resource_1);  
}  
  
void process_B(void) {  
    down(&resource_1);  
    down(&resource_2);  
    use_both_resources( );  
    up(&resource_2);  
    up(&resource_1);  
}
```

(a)

(a) Deadlock-free code.

# Resource Acquisition (3)

```
semaphore resource_1;  
semaphore resource_2;  
  
void process_A(void) {  
    down(&resource_1);  
    down(&resource_2);  
    use_both_resources( );  
    up(&resource_2);  
    up(&resource_1);  
}  
  
void process_B(void) {  
    down(&resource_2);  
    down(&resource_1);  
    use_both_resources( );  
    up(&resource_1);  
    up(&resource_2);  
}
```

(b)

(b) Code with a potential deadlock.

# Introduction To Deadlocks

Deadlock can be defined formally as follows:

*A set of processes is deadlocked if each process in the set is waiting for an event that only another process in the set can cause.*

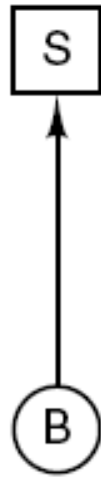
# Conditions for Resource Deadlocks

1. Mutual exclusion condition
2. Hold and wait condition.
3. No preemption condition.
4. Circular wait condition.

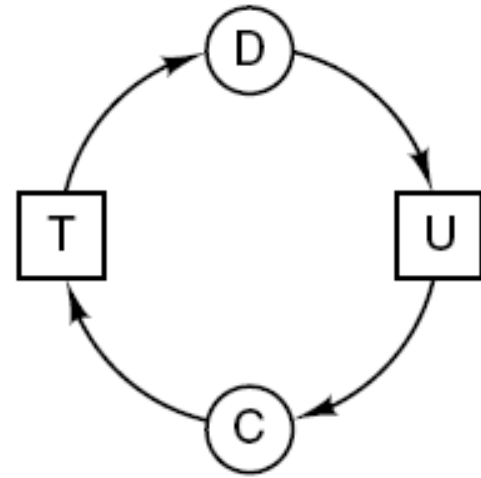
# Deadlock Modeling (1)



(a)



(b)



(c)

Resource allocation graphs. (a) Holding a resource.  
(b) Requesting a resource. (c) Deadlock.

# Deadlock Modeling (2)

A  
Request R  
Request S  
Release R  
Release S

(a)

B  
Request S  
Request T  
Release S  
Release T

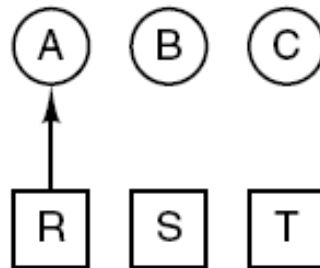
(b)

C  
Request T  
Request R  
Release T  
Release R

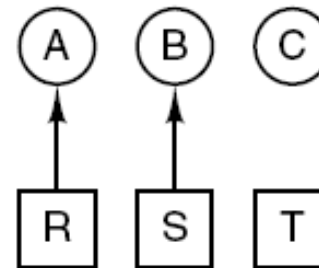
(c)

1. A requests R
2. B requests S
3. C requests T
4. A requests S
5. B requests T
6. C requests R  
deadlock

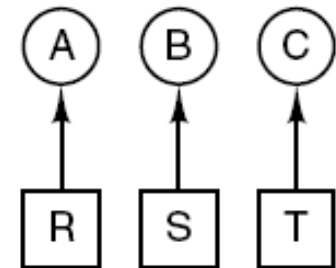
(d)



(e)



(f)



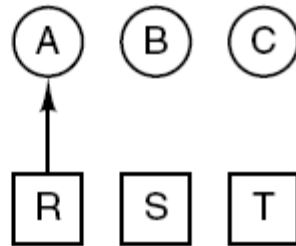
(g)

An example of how deadlock occurs and how it can be avoided.

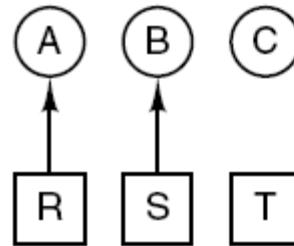
# Deadlock Modeling (3)

1. A requests R
2. B requests S
3. C requests T
4. A requests S
5. B requests T
6. C requests R  
deadlock

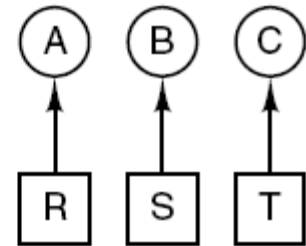
(d)



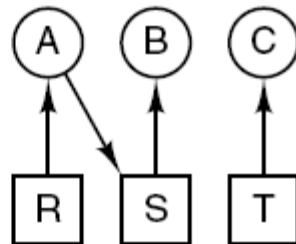
(e)



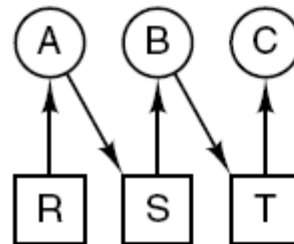
(f)



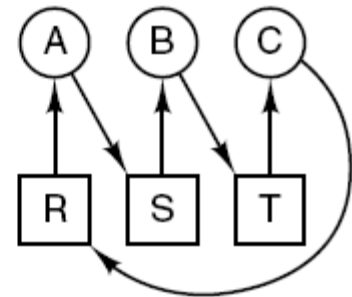
(g)



(h)



(i)



(j)

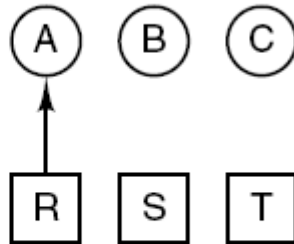
An example of how deadlock occurs and how it can be avoided.



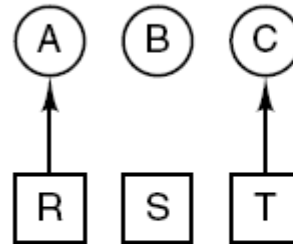
# Deadlock Modeling (4)

1. A requests R
2. C requests T
3. A requests S
4. C requests R
5. A releases R
6. A releases S  
no deadlock

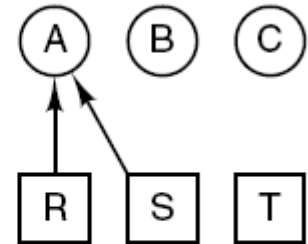
(k)



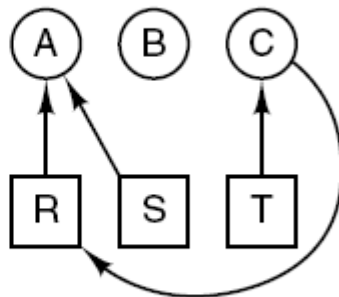
(l)



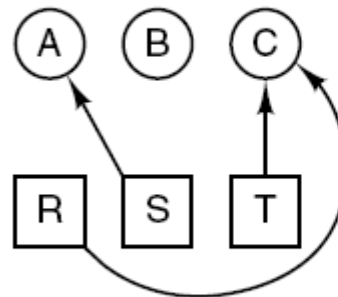
(m)



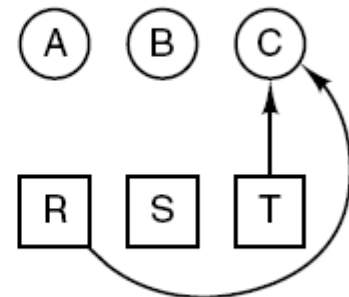
(n)



(o)



(p)



(q)

An example of how deadlock occurs and how it can be avoided.

# Deadlock Modeling (5)

Strategies for dealing with deadlocks:

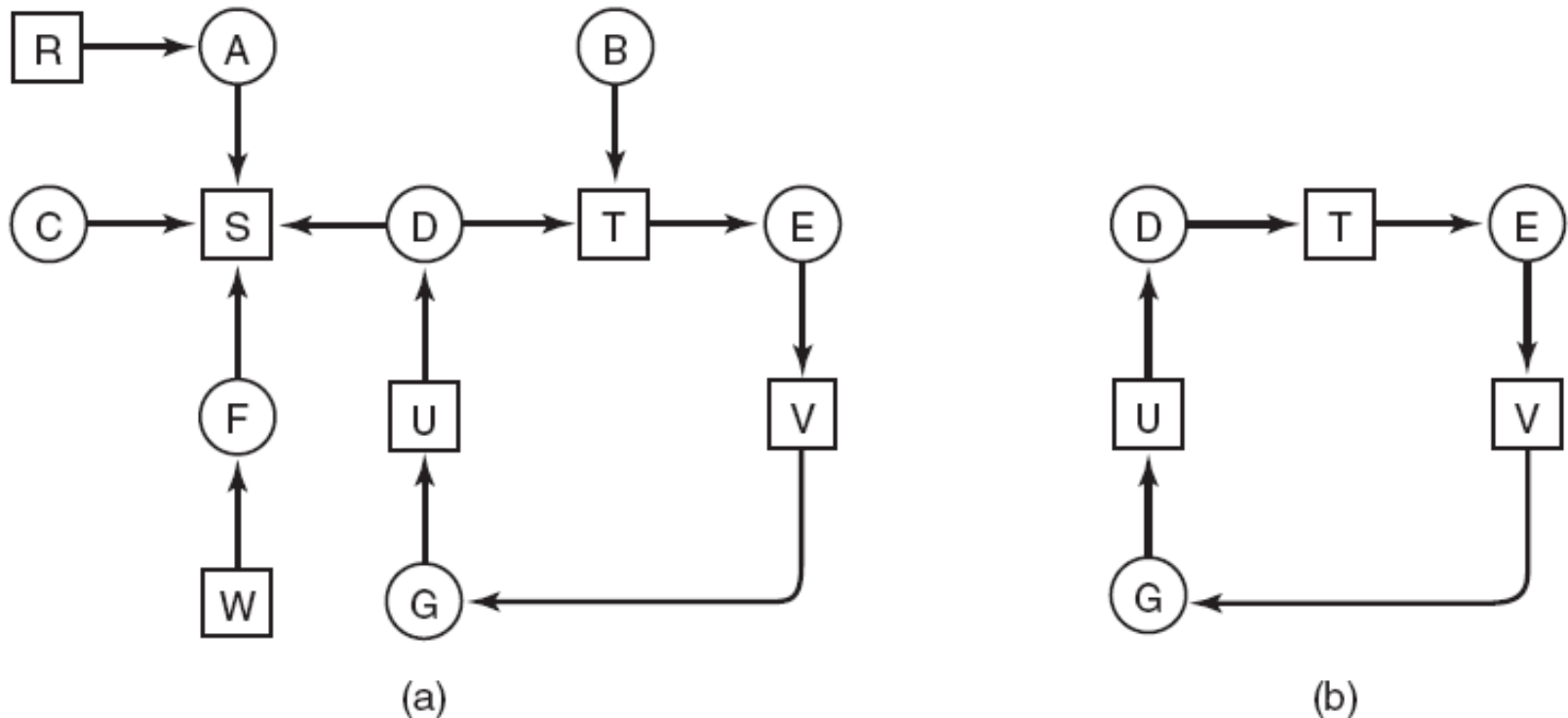
1. Just ignore the problem.
2. Detection and recovery. Let deadlocks occur, detect them, take action.
3. Dynamic avoidance by careful resource allocation.
4. Prevention, by structurally negating one of the four required conditions.

# Deadlock Detection with One Resource of Each Type (1)

Example of a system – is it deadlocked?

1. Process A holds R, wants S
2. Process B holds nothing, wants T
3. Process C holds nothing, wants S
4. Process D holds U, wants S and T
5. Process E holds T, wants V
6. Process F holds W, wants S
7. Process G holds V, wants U

# Deadlock Detection with One Resource of Each Type (1)



(a) A resource graph. (b) A cycle extracted from (a).

# Deadlock Detection with One Resource of Each Type (2)

Algorithm for detecting deadlock:

1. For each node, N in the graph, perform the following five steps with N as the starting node.
2. Initialize L to the empty list, designate all arcs as unmarked.
3. Add current node to end of L, check to see if node now appears in L two times. If it does, graph contains a cycle (listed in L), algorithm terminates.

# Deadlock Detection with One Resource of Each Type (3)


4. From given node, see if any unmarked outgoing arcs. If so, go to step 5; if not, go to step 6.
5. Pick an unmarked outgoing arc at random and mark it. Then follow it to the new current node and go to step 3.
6. If this is initial node, graph does not contain any cycles, algorithm terminates. Otherwise, dead end. Remove it, go back to previous node, make that one current node, go to step 3.

# Deadlock Detection with Multiple Resources of Each Type (1)

Resources in existence  
( $E_1, E_2, E_3, \dots, E_m$ )

Resources available  
( $A_1, A_2, A_3, \dots, A_m$ )


Current allocation matrix



$C_{11}$	$C_{12}$	$C_{13}$	$\dots$	$C_{1m}$
$C_{21}$	$C_{22}$	$C_{23}$	$\dots$	$C_{2m}$
$\vdots$	$\vdots$	$\vdots$		$\vdots$
$C_{n1}$	$C_{n2}$	$C_{n3}$	$\dots$	$C_{nm}$

Row n is current allocation  
to process n

Request matrix



$R_{11}$	$R_{12}$	$R_{13}$	$\dots$	$R_{1m}$
$R_{21}$	$R_{22}$	$R_{23}$	$\dots$	$R_{2m}$
$\vdots$	$\vdots$	$\vdots$		$\vdots$
$R_{n1}$	$R_{n2}$	$R_{n3}$	$\dots$	$R_{nm}$

Row 2 is what process 2 needs

The four data structures needed by the deadlock detection algorithm.

# Deadlock Detection with Multiple Resources of Each Type (2)

Deadlock detection algorithm:

1. Look for an unmarked process,  $P_i$ , for which the  $i$ -th row of  $R$  is less than or equal to  $A$ .
2. If such a process is found, add the  $i$ -th row of  $C$  to  $A$ , mark the process, and go back to step 1.
3. If no such process exists, the algorithm terminates.



# Deadlock Detection with Multiple Resources of Each Type (3)

$$E = \begin{pmatrix} 4 & 2 & 3 & 1 \end{pmatrix}$$

Tape drives  
Plotters  
Scanners  
CD Roms

$$A = \begin{pmatrix} 2 & 1 & 0 & 0 \end{pmatrix}$$

Tape drives  
Plotters  
Scanners  
CD Roms

Current allocation matrix

$$C = \begin{bmatrix} 0 & 0 & 1 & 0 \\ 2 & 0 & 0 & 1 \\ 0 & 1 & 2 & 0 \end{bmatrix}$$

Request matrix

$$R = \begin{bmatrix} 2 & 0 & 0 & 1 \\ 1 & 0 & 1 & 0 \\ 2 & 1 & 0 & 0 \end{bmatrix}$$

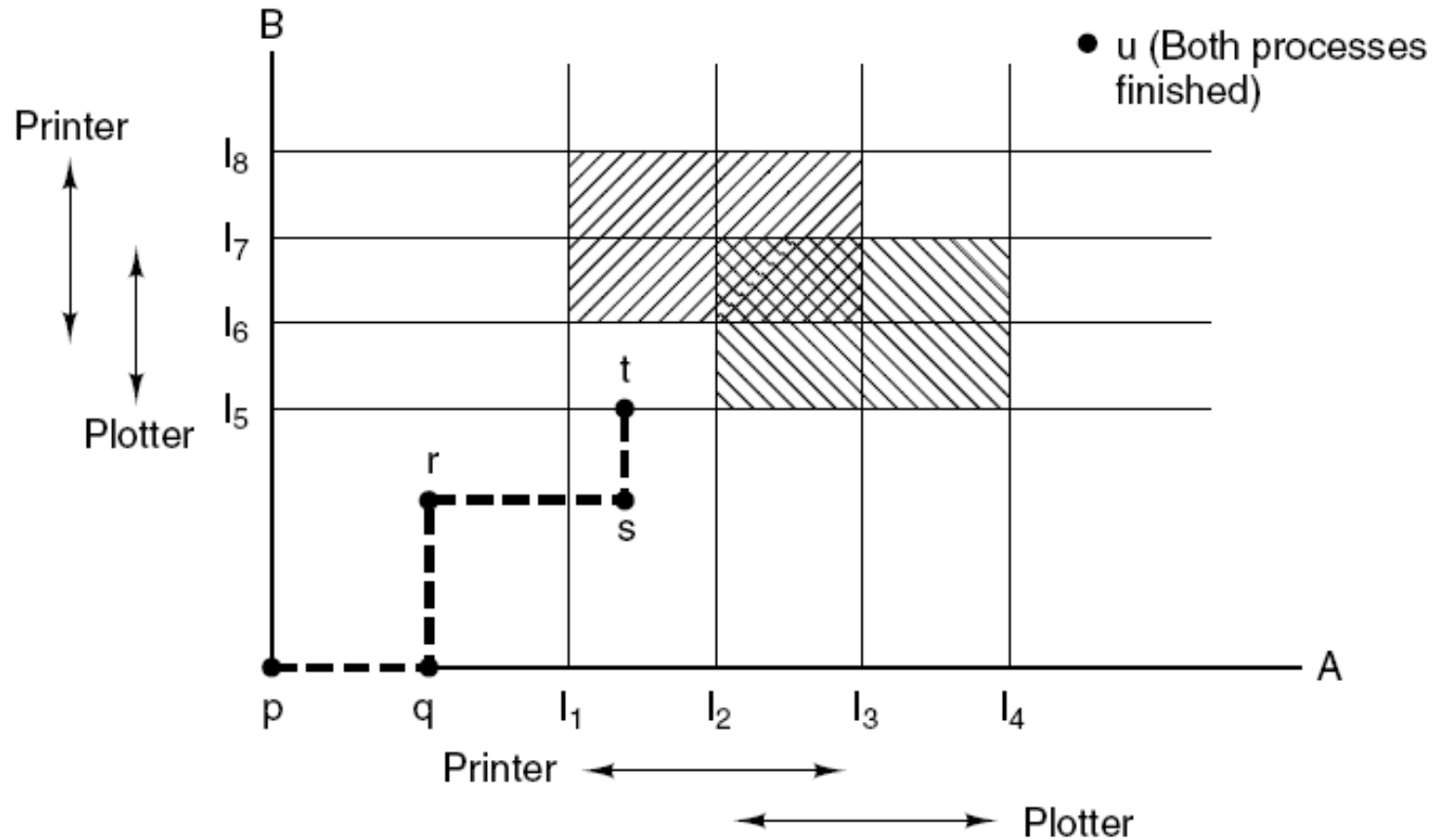


An example for the deadlock detection algorithm.

# Recovery from Deadlock

- Recovery through preemption
- Recovery through rollback
- Recovery through killing processes

# Deadlock Avoidance



Two process resource trajectories.

# Safe and Unsafe States (1)



Has Max

A	3	9
B	2	4
C	2	7

Free: 3  
(a)

Has Max

A	3	9
B	4	4
C	2	7

Free: 1  
(b)

Has Max

A	3	9
B	0	—
C	2	7

Free: 5  
(c)

Has Max

A	3	9
B	0	—
C	7	7

Free: 0  
(d)

Has Max

A	3	9
B	0	—
C	0	—

Free: 7  
(e)

Demonstration that the state in (a) is safe.

# Safe and Unsafe States (2)



Has Max		
A	3	9
B	2	4
C	2	7

Free: 3

(a)

Has Max		
A	4	9
B	2	4
C	2	7

Free: 2

(b)

Has Max		
A	4	9
B	4	4
C	2	7

Free: 0

(c)

Has Max		
A	4	9
B	—	—
C	2	7

Free: 4

(d)

Demonstration that the state in (b) is not safe.

# The Banker's Algorithm for a Single Resource

Has Max		
A	0	6
B	0	5
C	0	4
D	0	7

Free: 10

(a)

Has Max		
A	1	6
B	1	5
C	2	4
D	4	7

Free: 2

(b)



Has Max		
A	1	6
B	2	5
C	2	4
D	4	7

Free: 1

(c)



Three resource allocation states: (a) Safe. (b) Safe. (c) Unsafe.

# The Banker's Algorithm for Multiple Resources (1)

	Process	Tape drives	Plotters	Printers	CD ROMs
A	3	0	1	1	
B	0	1	0	0	
C	1	1	1	0	
D	1	1	0	1	
E	0	0	0	0	

Resources assigned



	Process	Tape drives	Plotters	Printers	CD ROMs
A	1	1	0	0	
B	0	1	1	2	
C	3	1	0	0	
D	0	0	1	0	
E	2	1	1	0	

Resources still needed



E = (6342)

P = (5322)

A = (1020)

The banker's algorithm with multiple resources.

# The Banker's Algorithm for Multiple Resources (2)

Algorithm for checking to see if a state is safe:

1. Look for row,  $R$ , whose unmet resource needs all  $\leq A$ . If no such row exists, system will eventually deadlock since no process can run to completion
2. Assume process of row chosen requests all resources it needs and finishes. Mark process as terminated, add all its resources to the  $A$  vector.
3. Repeat steps 1 and 2 until either all processes marked terminated (initial state was safe) or no process left whose resource needs can be met (there is a deadlock).



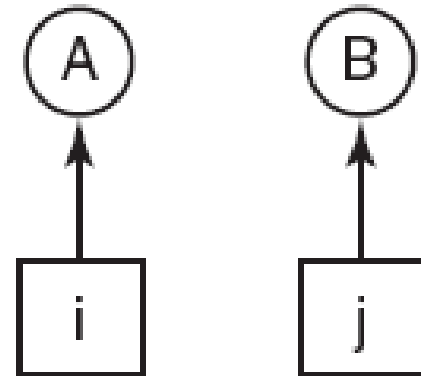
# Deadlock Prevention

- Attacking the mutual exclusion condition
- Attacking the hold and wait condition
- Attacking the no preemption condition
- Attacking the circular wait condition

# Attacking the Circular Wait Condition

1. Imagesetter
2. Scanner
3. Plotter
4. Tape drive
5. CD-ROM drive

(a)



(b)




(a) Numerically ordered resources. (b) A resource graph.

# Approaches to Deadlock Prevention

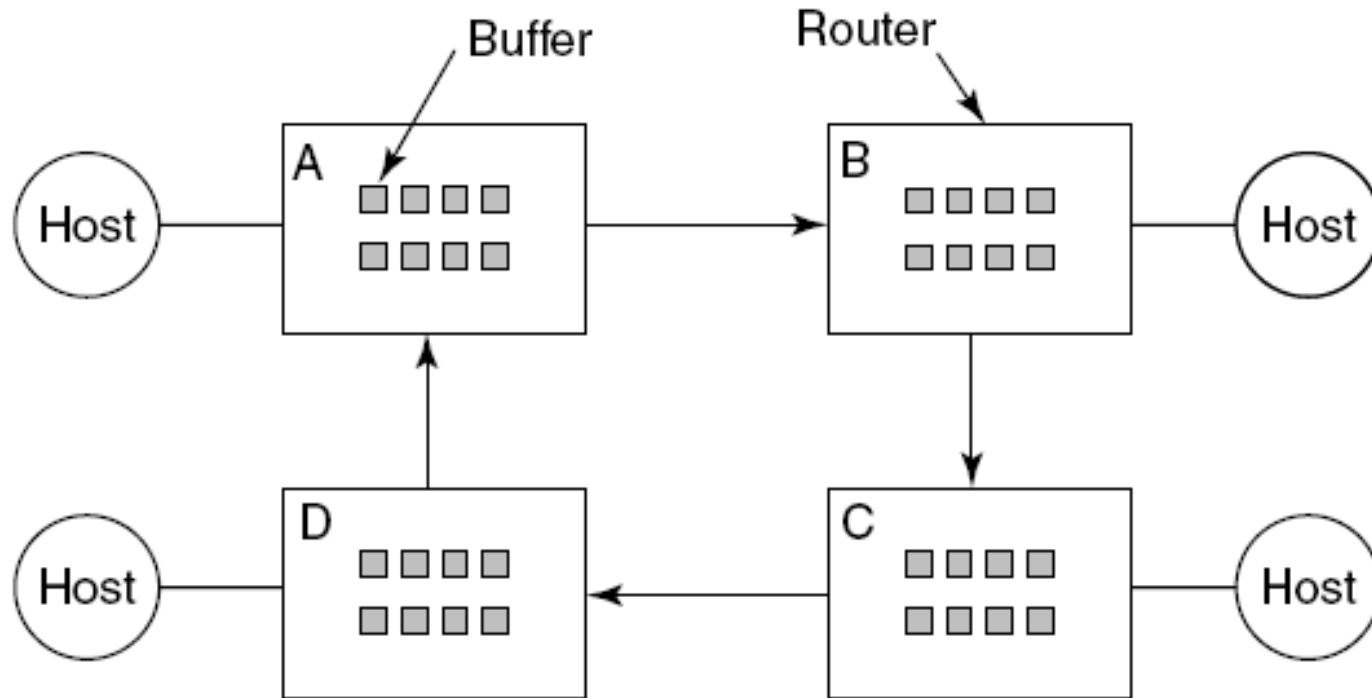
Condition	Approach
Mutual exclusion	Spool everything
Hold and wait	Request all resources initially
No preemption	Take resources away
Circular wait	Order resources numerically

Summary of approaches to deadlock prevention.

# Other Issues

- Two-phase locking 
- Communication deadlocks
- Livelock
- Starvation

# Communication Deadlocks



- A resource deadlock in a network.

# Livelock

```
void process_A(void) {  
    enter_region(&resource_1);  
    enter_region(&resource_2);  
    use_both_resources( );  
    leave_region(&resource_2);  
    leave_region(&resource_1);  
}
```

```
void process_B(void) {  
    enter_region(&resource_2);  
    enter_region(&resource_1);  
    use_both_resources( );  
    leave_region(&resource_1);  
    leave_region(&resource_2);  
}
```



- Busy waiting that can lead to livelock.