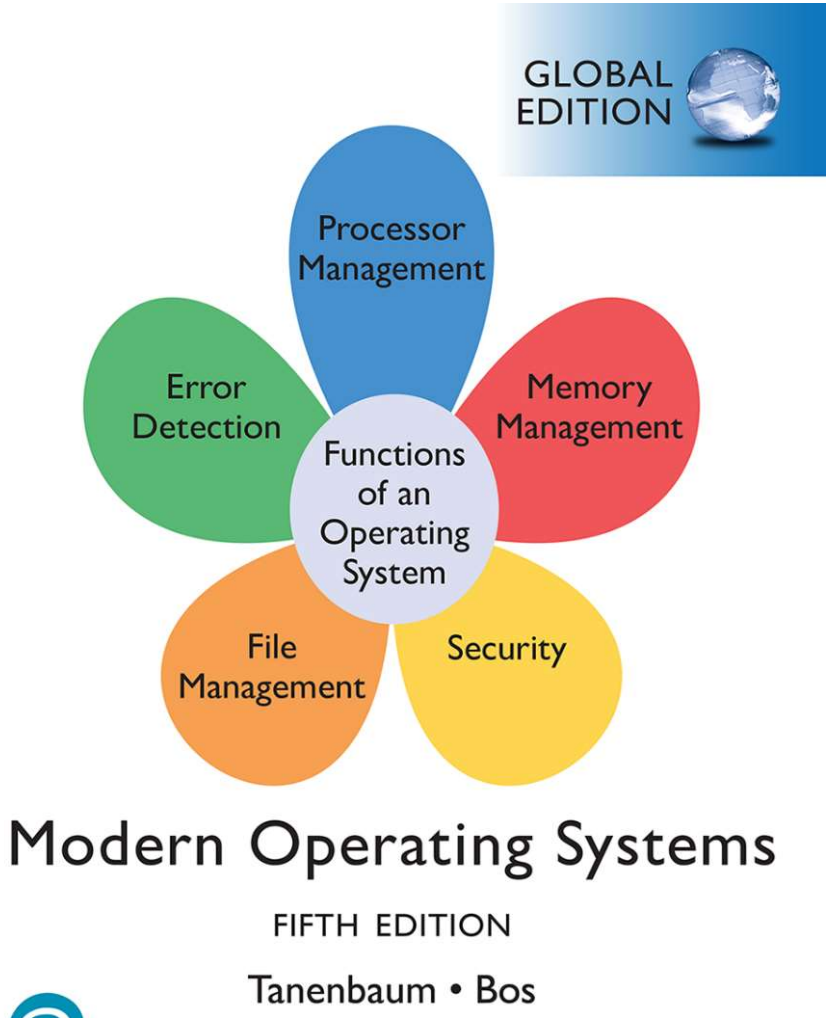
Modern Operating Systems

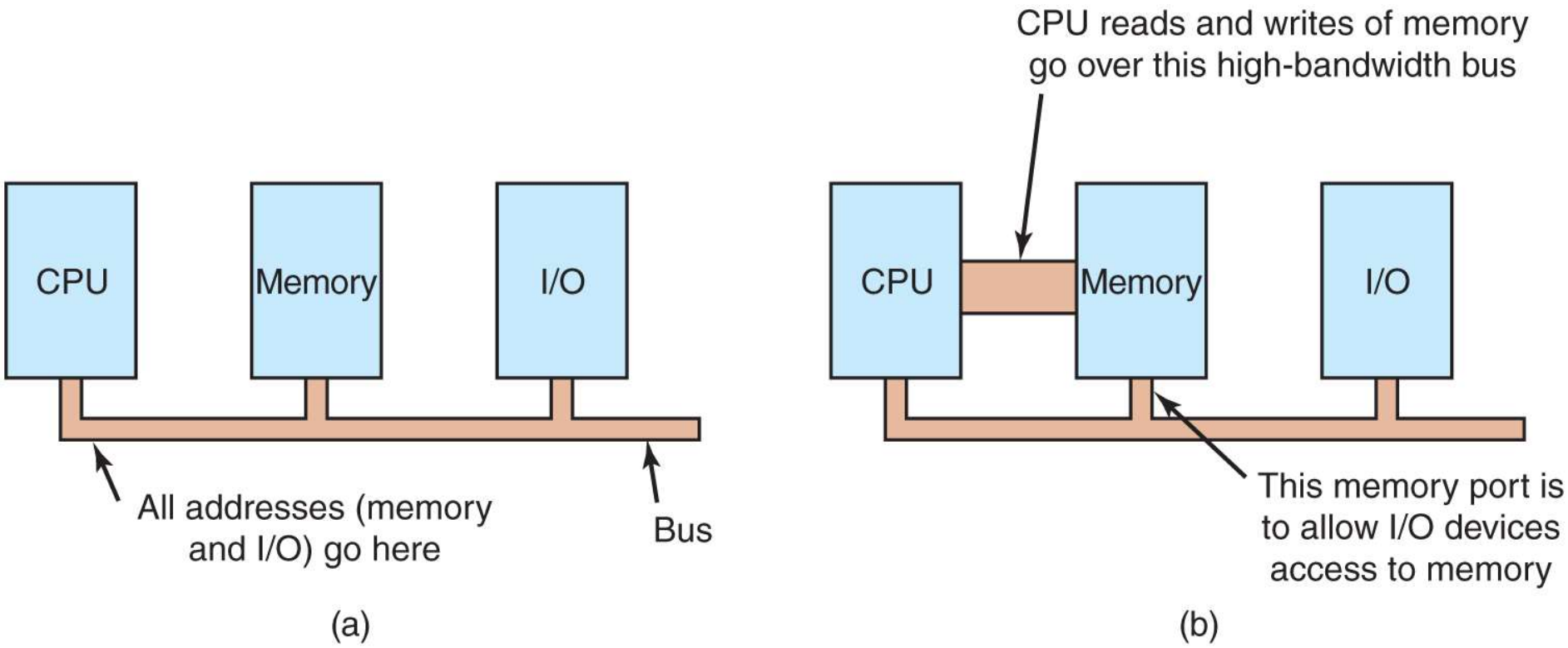
Fifth Edition, Global Edition



Chapter 5

Input/Output

System vs Peripheral Bus



a) A single-bus architecture

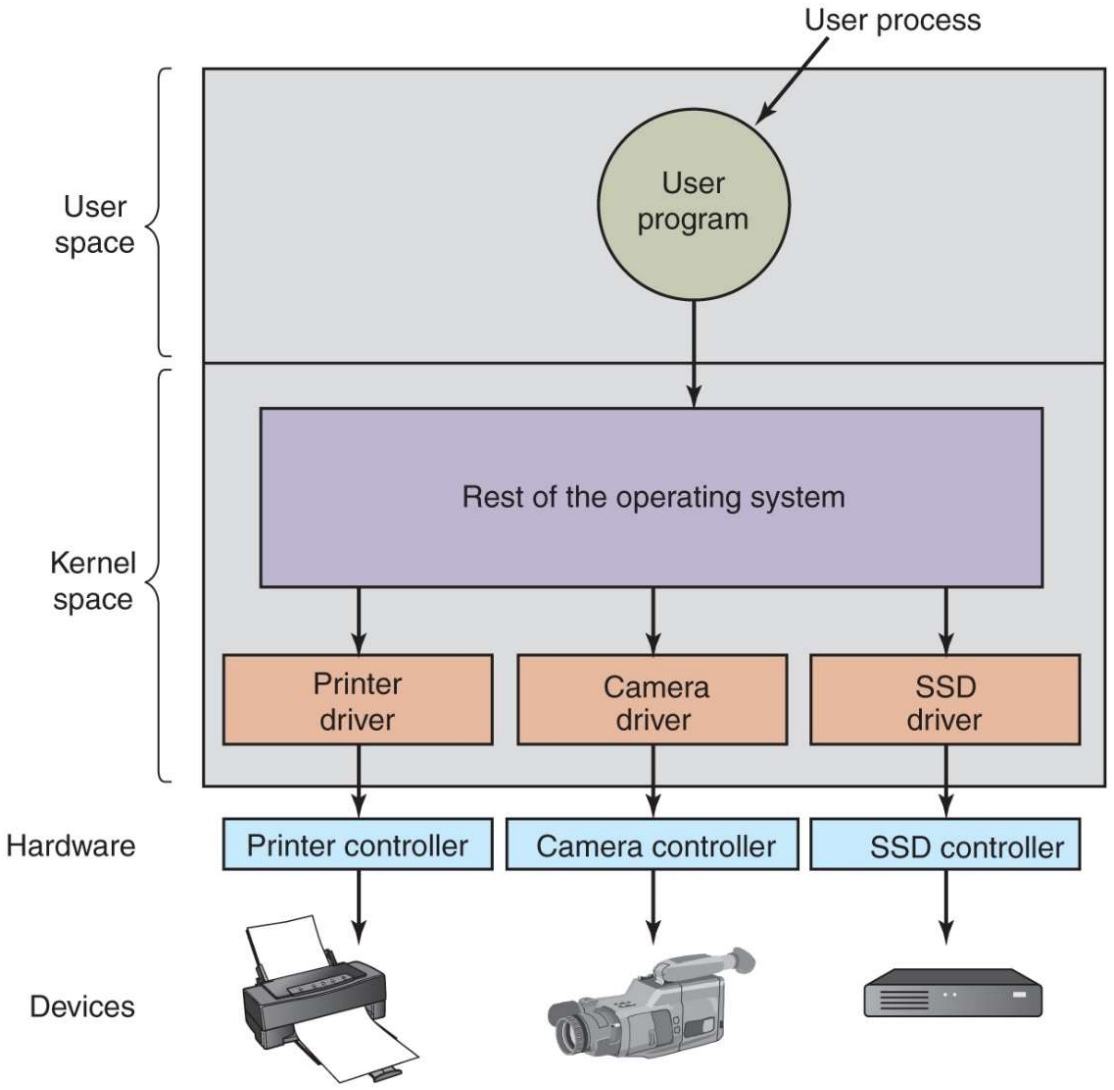
b) (b) A dual-bus memory architecture.

Device Drivers

Logical positioning of device drivers from app to device via:

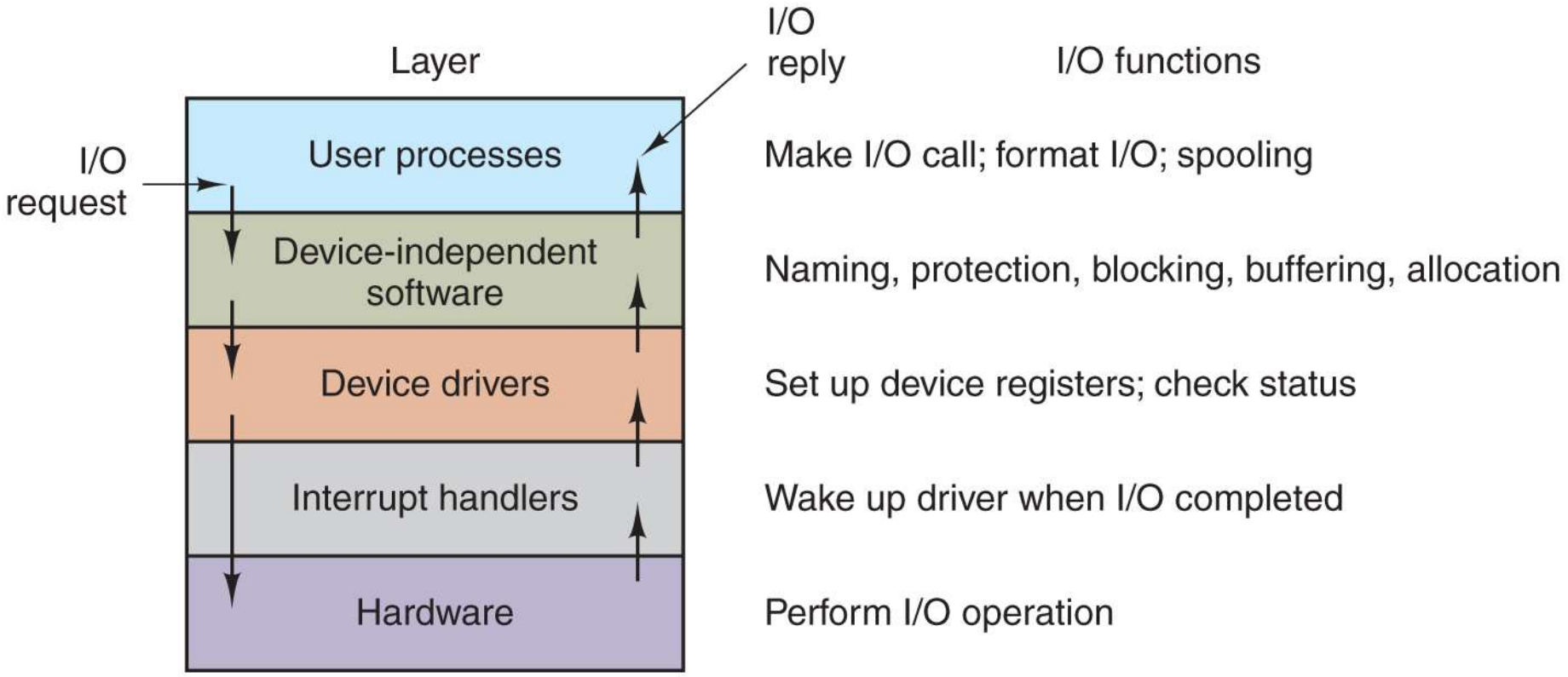
- system call
- device-independent i/o
- device drivers
- controllers

Most communication between drivers & device controllers goes over a peripheral bus



Device Drivers

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



Device-Independent I/O SW

- Provide a uniform device driver I/F to “similar” hw devices
  - Hide device-specific details from user
- Handle buffering and caching
- Error handling and reporting
  - Detect, recover from, and report I/O errors
- Device allocation and release

Types of Errors & Handling

User errors

- invalid buffer/address
- bad parameter

Device errors

- is another try likely to fix the problem?
- if unable to fix, report error to OS

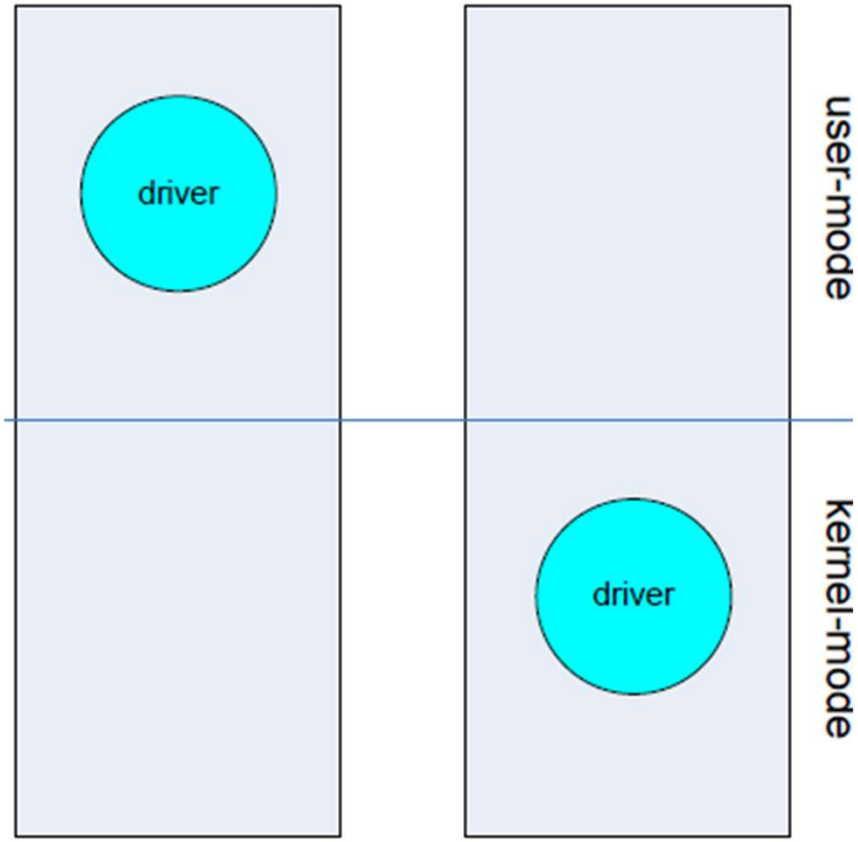
Error handling

- handle at lowest layer possible?
- many errors are transient

Device Drivers

Where do device drivers live?

- Traditionally part of kernel (built-in or loadable)
  - device-independent kernel I/F
- User-space I/O, user-mode drivers
  - Libraries
    - abstractions of system calls
    - Examples: scanf/printf/cout
  - Spoolers using daemons
    - Prioritize accesses to devices
    - Examples: printer (CUPS/LP), mail, Samba



Device Drivers

Clean abstractions (uniform I/F, names)

- driver API as generic as possible

Security

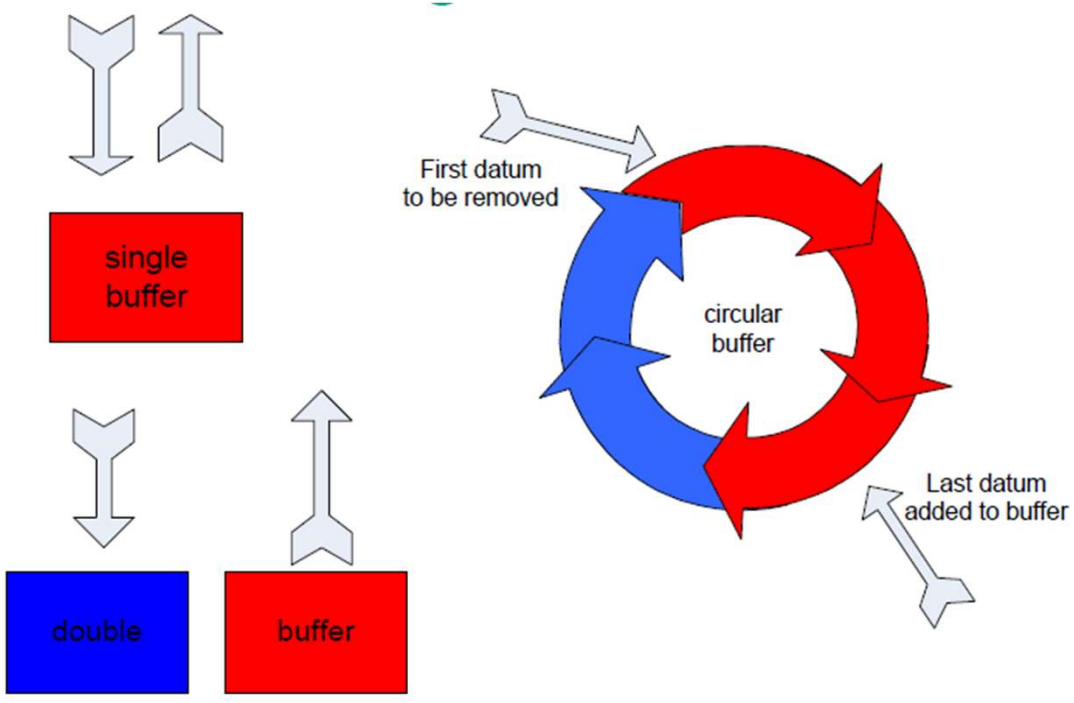
- Who is allowed to access the device? Permissions?

Robustness

- check for valid parameters
- interrupts that occur within an interrupt?

Other support

- hot plug
- suspend/hibernate
- Buffering?
- where to put data? size? latency?

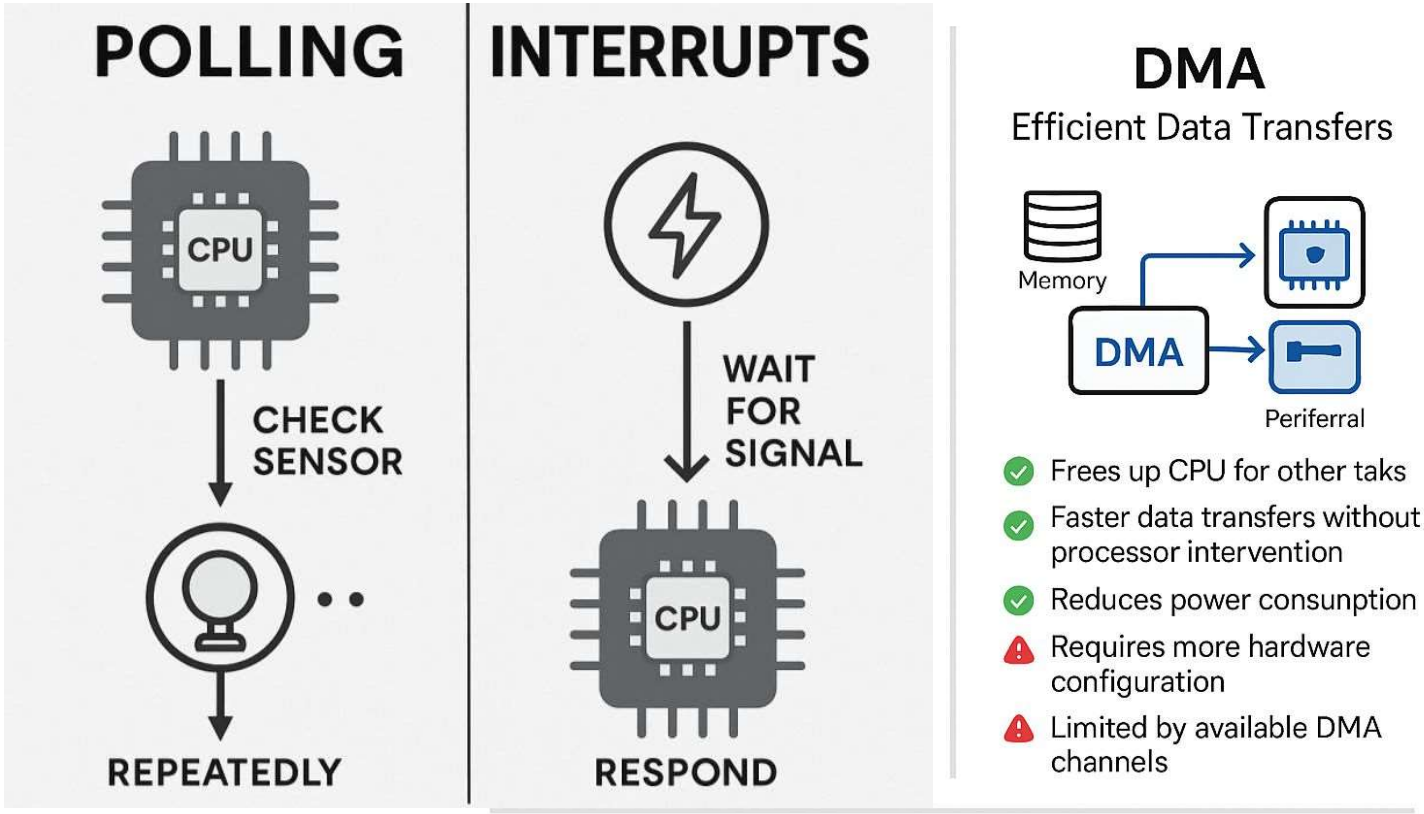


Kernel Modules & Device Drivers

- Character Devices
- Block Devices
- Network Devices

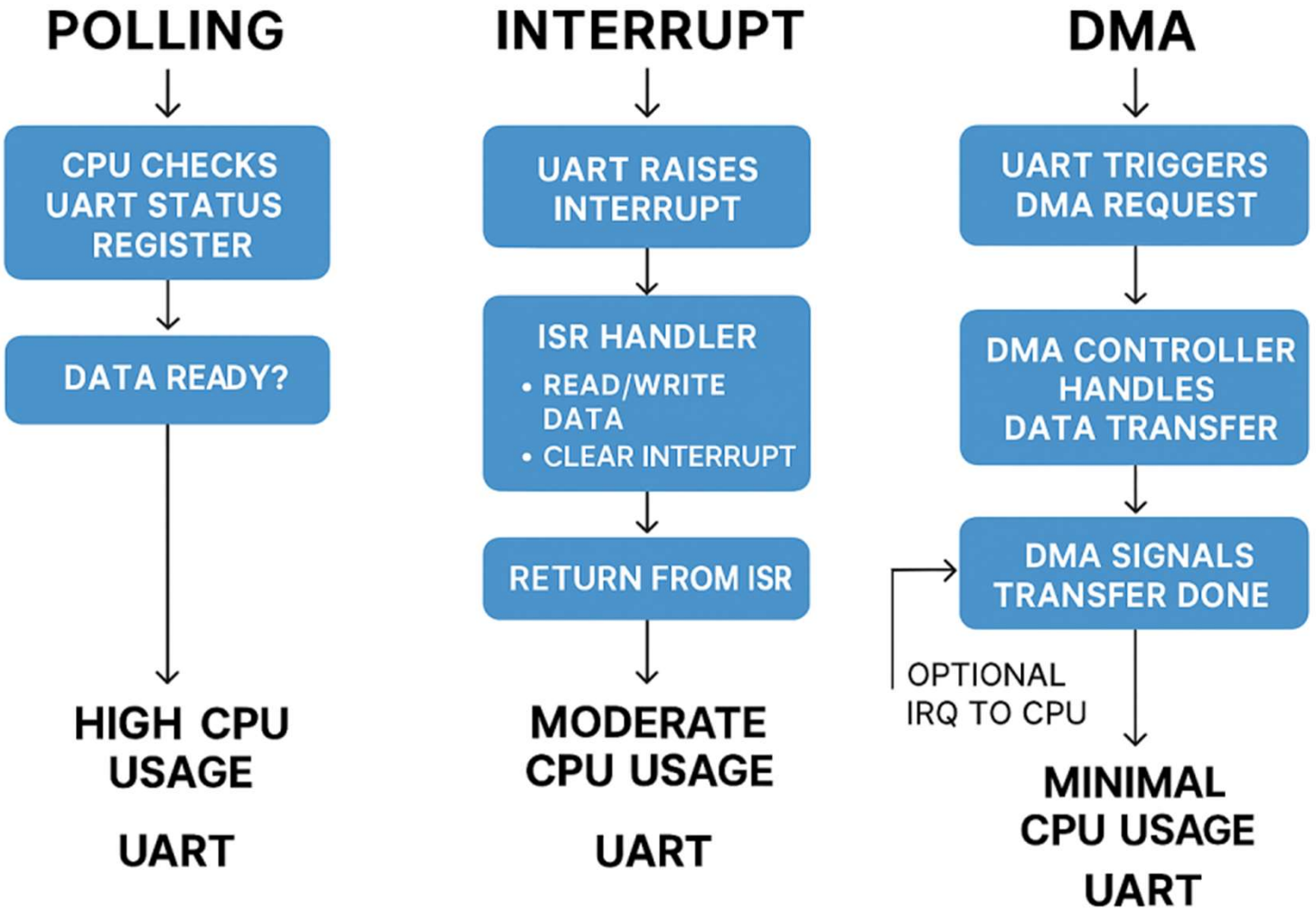
Device	Data rate
Keyboard	10 bytes/sec
Mouse	100 bytes/sec
56K modem	7 KB/sec
Bluetooth 5 BLE	256 KB/sec
Scanner at 300 dpi	1 MB/sec
Digital video recorder	3.5 MB/sec
802.11n Wireless	37.5 MB/sec
USB 2.0	60 MB/sec
16x Blu-ray disc	72 MB/sec
Gigabit Ethernet	125 MB/sec
SATA 3 disk drive	600 MB/sec
USB 3.0	625 MB/sec
Single-lane PCIe 3.0 bus	985 MB/sec
802.11ax Wireless	1.25 GB/sec
PCIe Gen 3.0 NVMe M.2 SSD (reading)	3.5 GB/sec
USB 4.0	5 GB/sec
PCI Express 6.0	126 GB/sec

I/O Transfer Mode

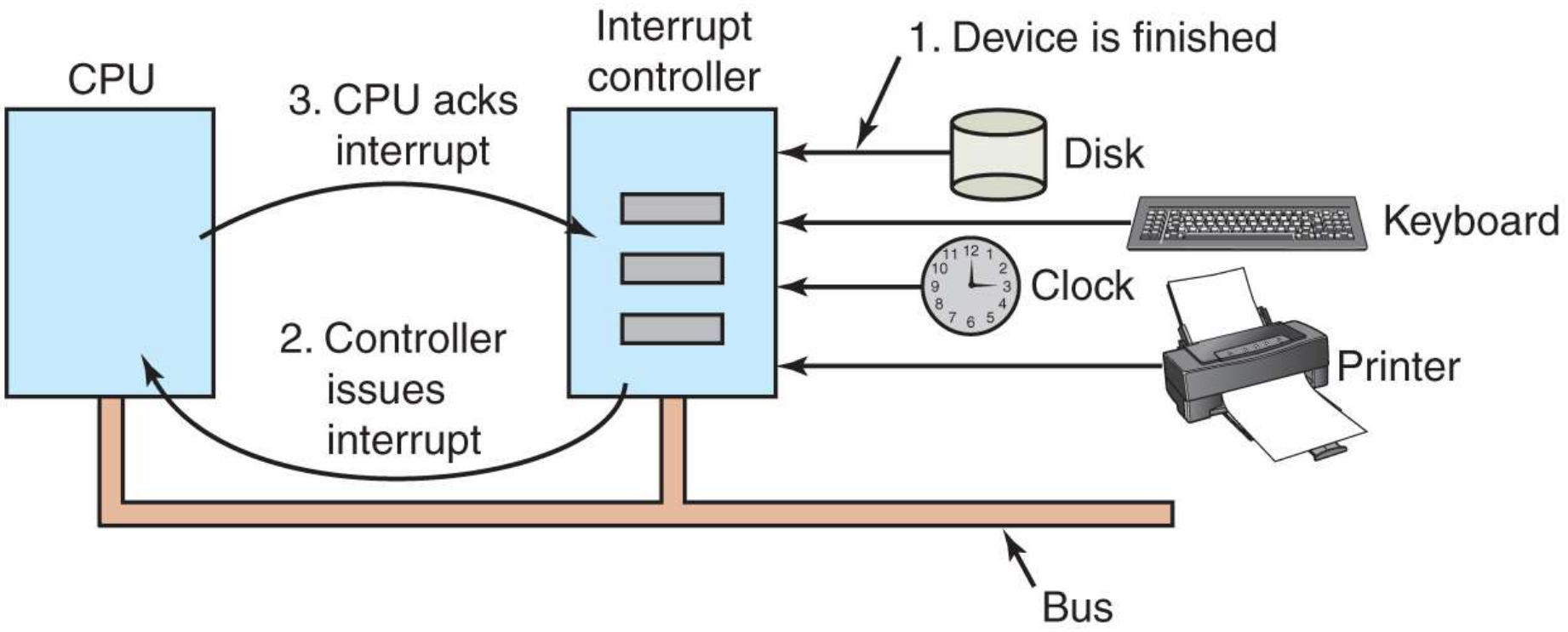


- Programmed I/O (polling, busy-waiting)
- Interrupt-driven I/O
- Direct memory access (DMA)
  - single interrupt per operation (burst mode)

Example: UART



Interrupt



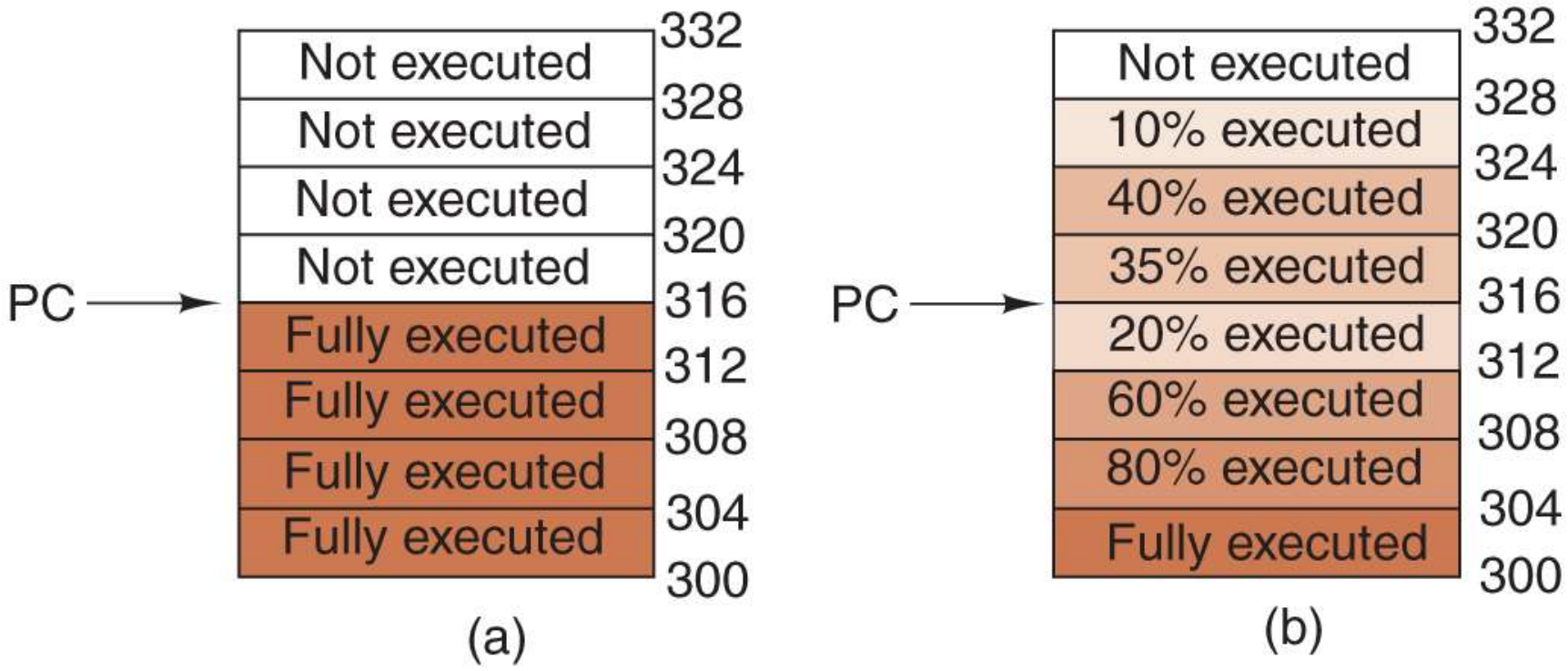
The device driver configures the device, and when the device if finished its controller uses interrupt lines on the bus to issue an IRQ, which is serviced by the interrupt handler (IRS)

Interrupt Operation

1. Application requests I/O, e.g., read/write/open/send
   - asynchronous/unsolicited/event-driven interrupts use no application request
   - (Examples: key press, mouse movement, incoming network packet, timers)
2. OS invokes the device driver to control the device
3. Device driver programs the device controller
   - write hw registers using memory-mapped I/O or port I/O
   - Buffer/memory addresses, transfer size, command bits, interrupt enable bits …
4. Device controller starts the operation on the hw device. Upon completion (or when requiring attention), the device controller issues an interrupt (IRQ)
5. CPU receives the IRQ and transfers control to the interrupt handler, which executes the Interrupt Service Routine (ISR) in kernel mode
6. ISR handles the interrupt
   - checks status, schedules transfer, clears IRQ flags, and possibly starts the next job
7. OS notifies the application if needed

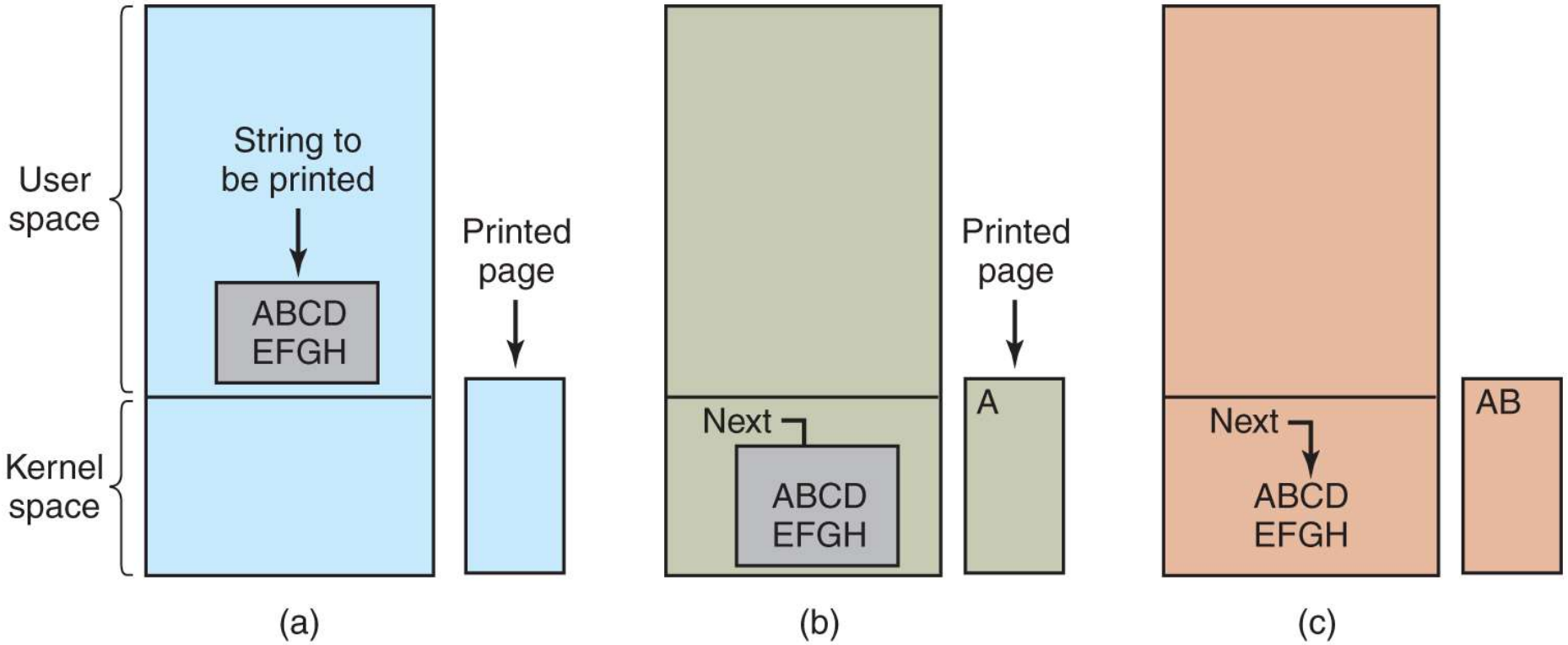
Precise & Imprecise Interrupts

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



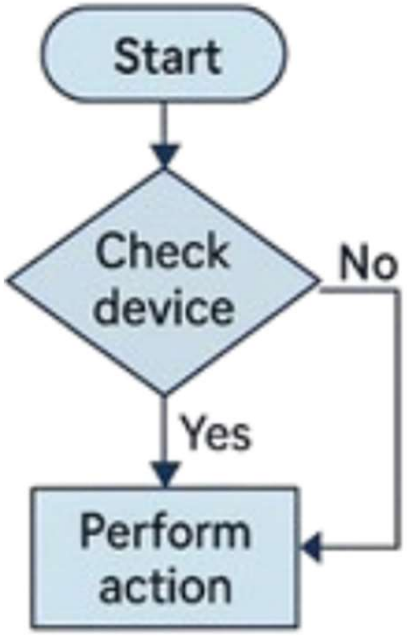
I/O Example: Printing a String

Steps:



Programmed I/O: Writing to a Printer

```
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( );
```



Interrupt-Driven I/O: Writing to a Printer

```
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

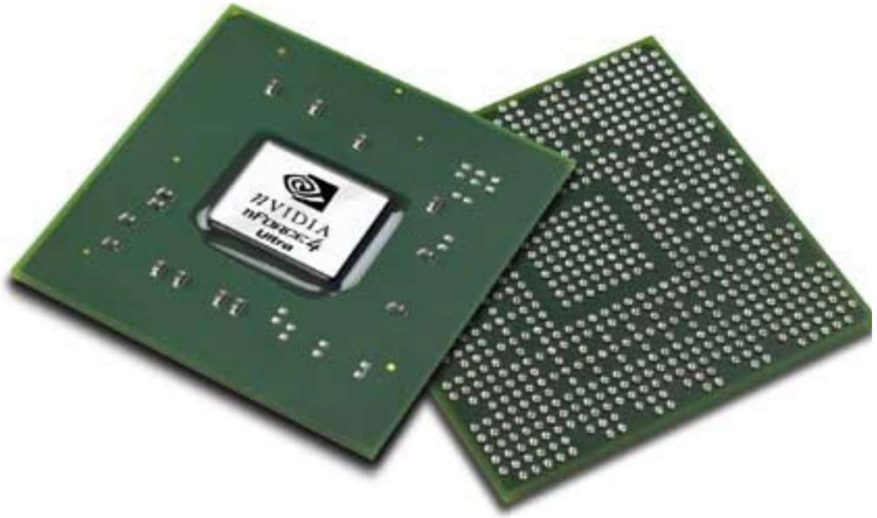
a) Code executed at the time the print system call is made

b) Interrupt service procedure for the printer

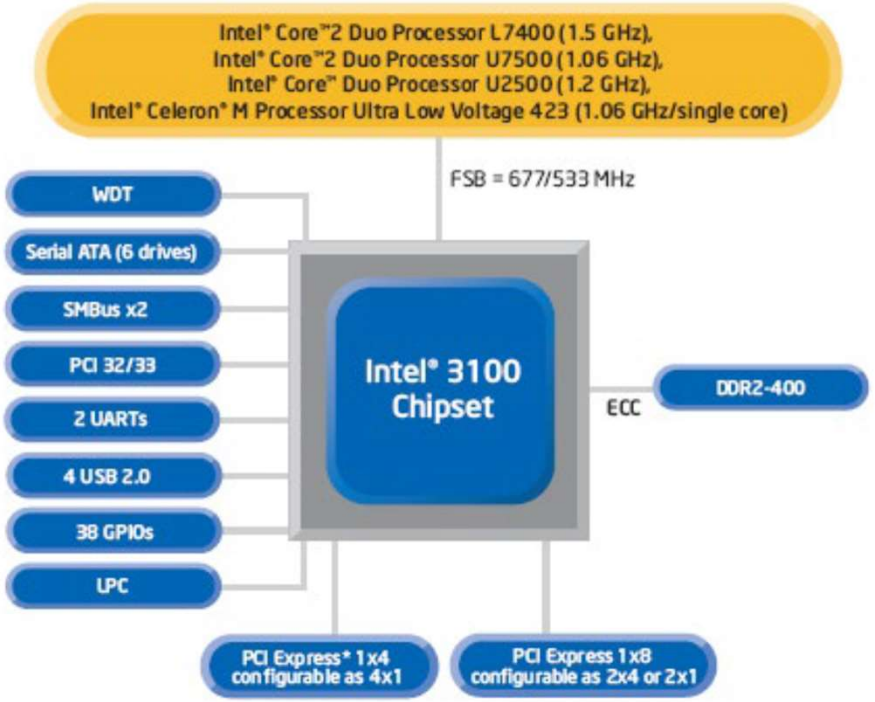
Direct Memory Access (DMA)

DMA controller

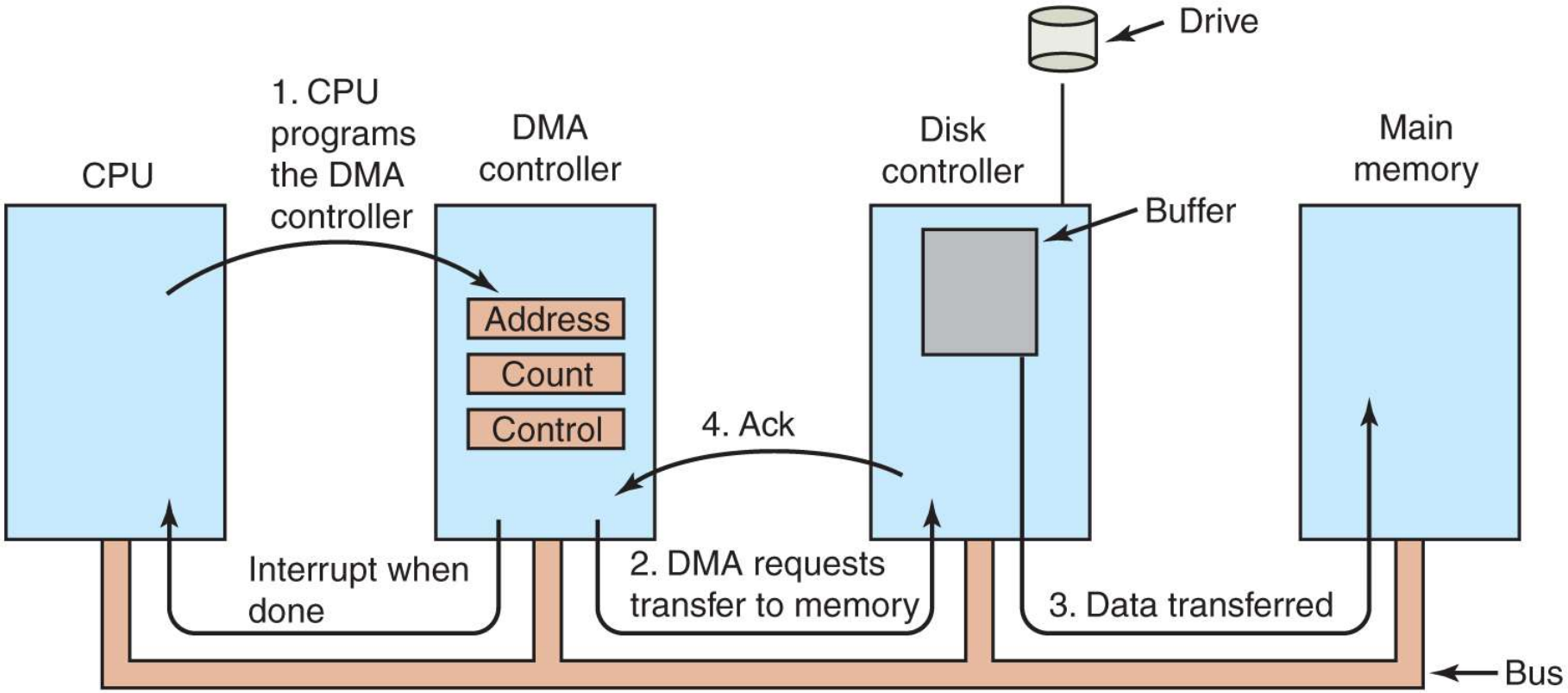
- manages transfer between device and RAM
- integrated with device controller or separate device on motherboard



nVIDIA nForce4 integrated chipset

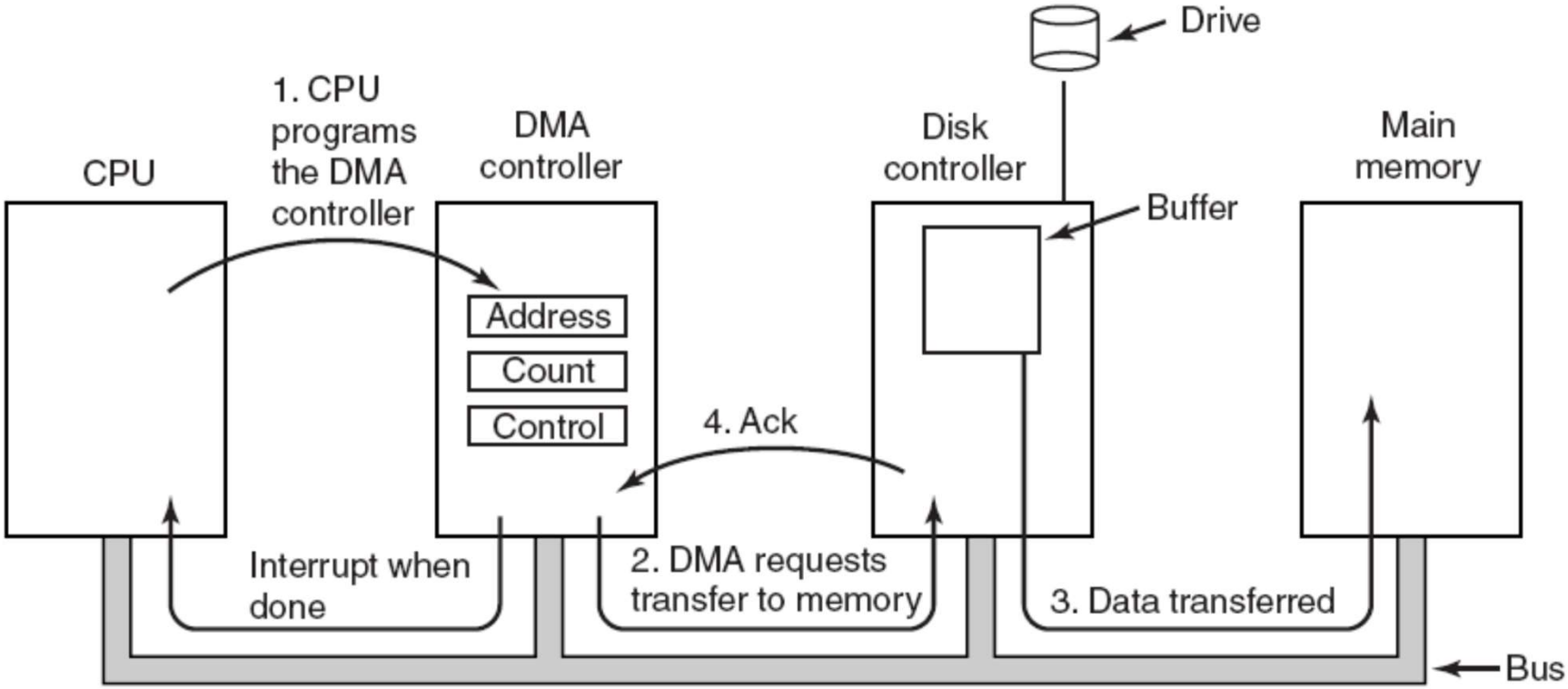


DMA Transfers



- Step 3 uses either
- Cycle stealing: Acquire bus and transfer a bus cycle or two, or
- Burst mode: Acquire bus and complete the transfer
- Bus acquisition takes time

Direct Memory Access (DMA)



DMA Example

Printing a string using DMA

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

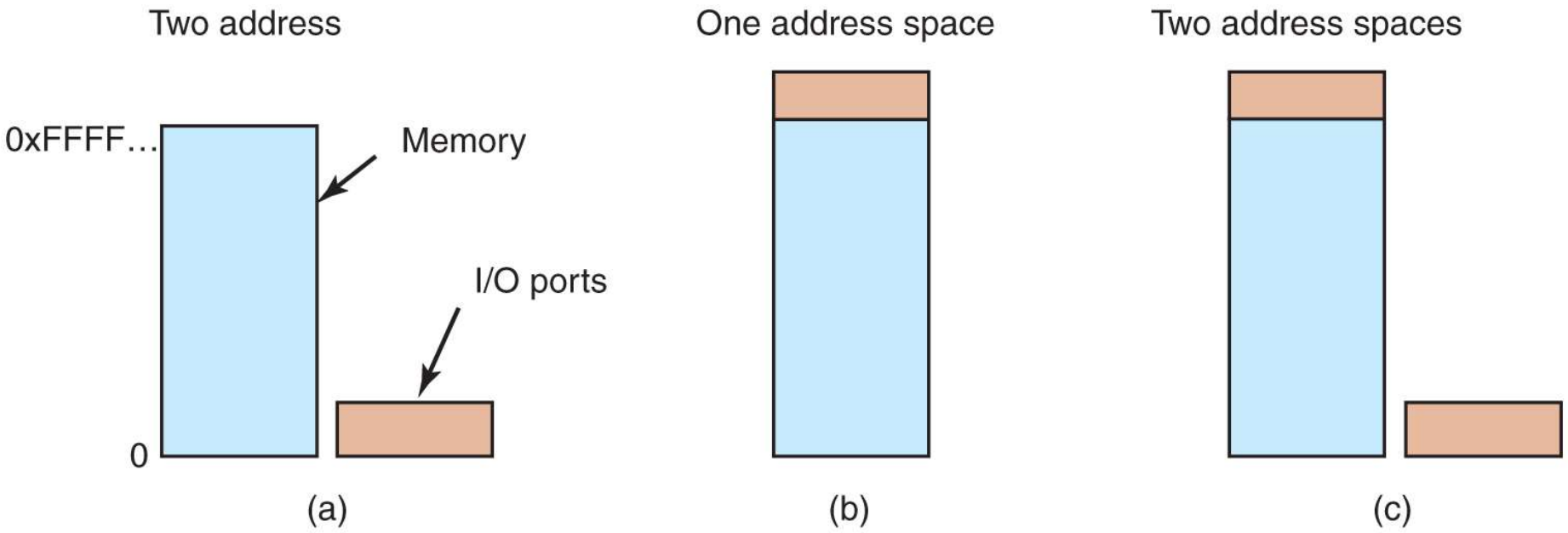
```
copy_from_user(buffer, p, count);
set_up_DMA_controller( );
scheduler( );
```

(a)

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

(b)

I/O Addressing of Real HW



(a) I/O Ports: Separate I/O and memory space

(b) Memory-mapped I/O

(c) Hybrid

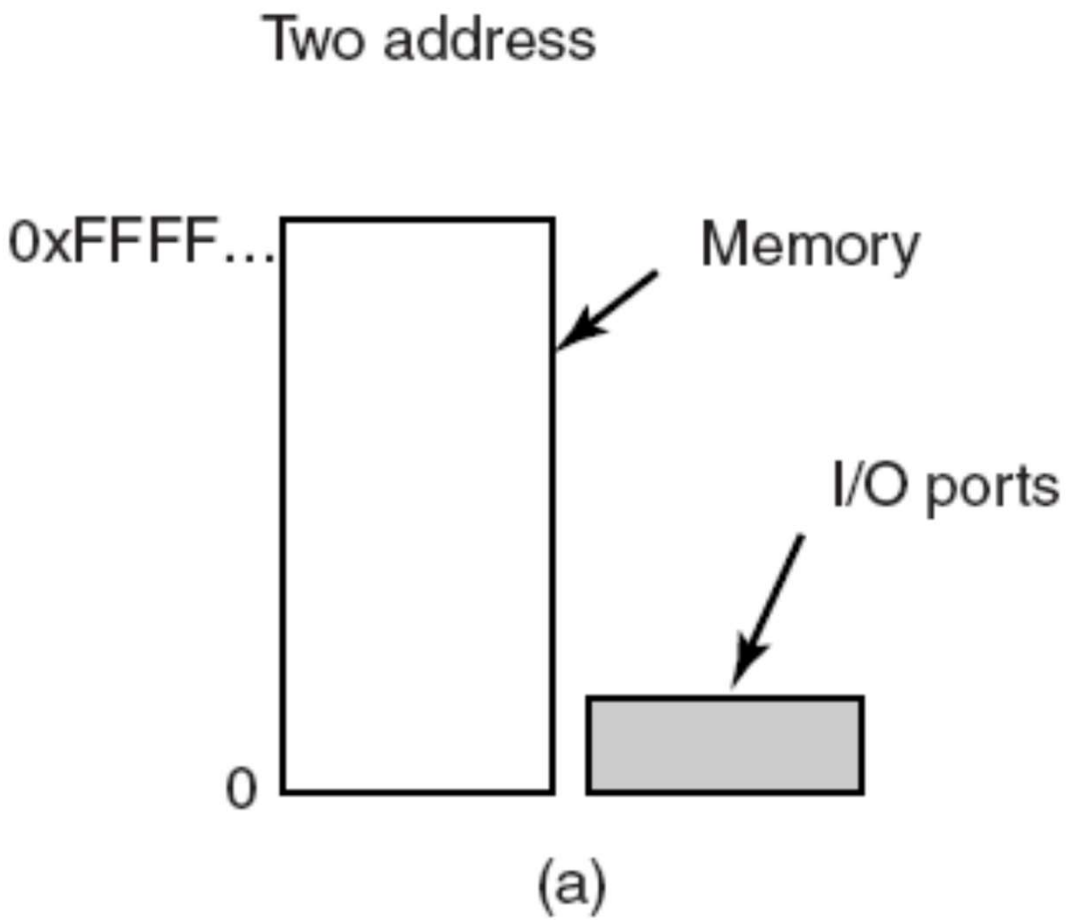
I/O Ports

IN/OUT instructions

Port types

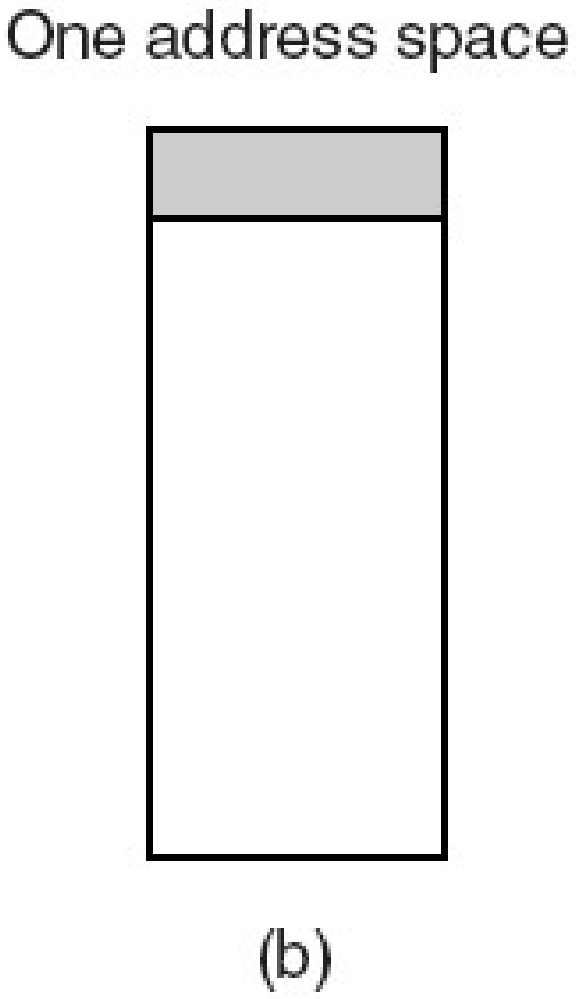
- control
- data

- Sample syntax
- In Reg, Port
- Out Reg, Port



Memory Mapped Devices

- I/O address space map
- Control and data registers are assigned system addresses



Examples of Device I/O

Memory mapped

```
// byte register
// mapped to location 0x1200

uint8 *char = 0x1200;
uint8 value;

// Read register
value = *(char);
// Assign register
*(char) = 0x7;
```

Port mapped

```
// x86 assembler
// byte register mapped
// to I/O port 0x400

// Read to byte (AL register)
//
IN AL, 0x400

// Write to register
MOV AL, 0x7
OUT 0x400, AL
```

Hybrid: I/O Ports & Memory Mapped I/O

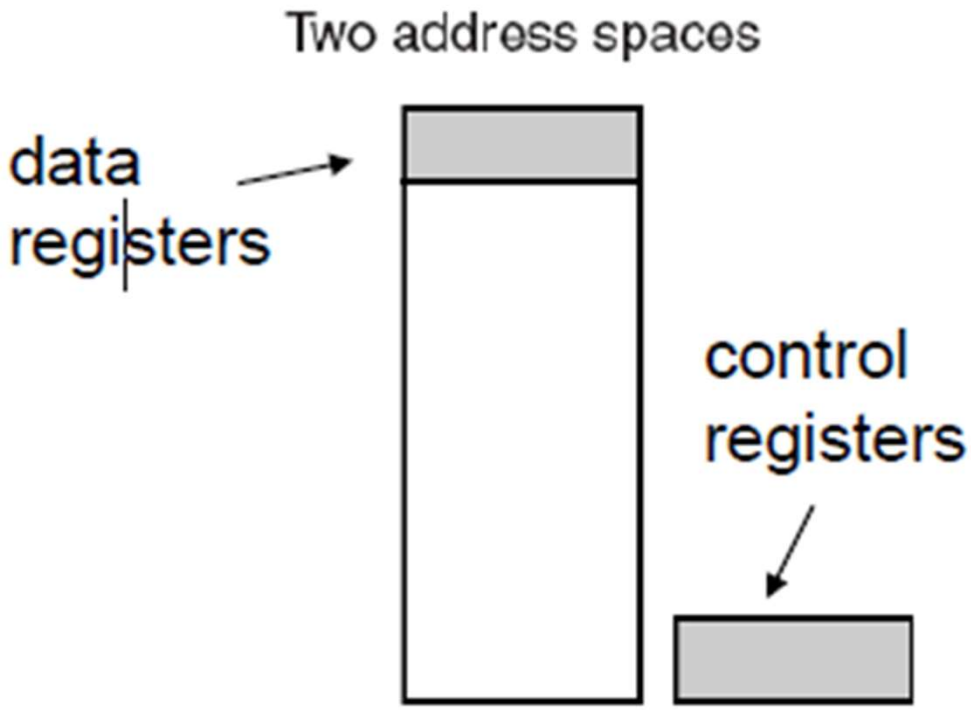
- Combination Modern Intel architectures support both port and memory-mapped I/O

I/O Ports

- Extra address line
- Use of assembler IN/OUT

Memory mapped

- Conceptually easier
- Cahe & Bus issues



Character Devices: Keyboard

Keyboards & scan code

- Most modern keyboard have ≤ 128 keys
  - 7 bits sufficient to encode a key
  - 1 bit to encode press/release
- Each key press produces a scan code, with the high order bit indicating if the key has been pressed or released

Examples

- {press k, release k, press i, release i, press s, release s, press s, release s} ⇒ device driver maps this to word: kiss
- {Shift press, k press, k release, shift release} or {Shift press, k press, shift release, k release} ⇒ device driver maps this to capital K

Key Presses & Scan Codes

	Key event	Report	Comment
1	Press (only) “H”	00 00 0b 00 00 00 00 00	Scancode for “H” is 0x0b
2	Press “J” without releasing “H”	00 00 0b 0d 00 00 00 00	Scancode for “J” is 0x0d
3	Press “B” without releasing “HJ”	00 00 0b 0d 05 00 00 00	Scancode for “B” is 0x05
4	Release “J”, still pressing “HB”	00 00 0b 05 00 00 00 00	No press is 0x00
5	Press “O” without releasing “HB”	00 00 0b 05 12 00 00 00	Scancode for “O” is 0x12

Scan codes are bytes sent by a USB keyboard when a user presses and releases different keys

Earlier key presses are encoded by the bytes toward the left

Terminals & Display Devices

Terminals handle keycode conversions to a coding system, such as ASCII or unicode

- Non-Canonical mode: character by character input
- Canonical mode: deliver a line at a time
  - Examples: text editors, text-UIs, games, shells doing advanced input
- Raw vs Cooked mode related to the amount of processing
  - often raw connected to Non-Canonical and Cooked related to Canonical mode
- Frequently echo keystrokes to display devices (escape characters & sequences)

Terminals & Display Devices

- Control codes: tab, newline, etc. can vary between devices
  - termcap/terminfo (database of available terminal capabilities)
- vt100 terminal
  - text terminal (no framebuffer graphics devices)
  - single font (bitmap size, green), inverse, blink, uline, bold
- {vt102, vt220, vt320, vt420, vt520, …}
  - configurable fonts, Unicode, 256-colors, truecolor, emoji, resizing, glyph ligatures, transparency, images (in some terminals), GPU rendering
- Ncurses is a library for full-screen, text UIs
  - It uses terminfo & targets vt100/ANSI-like terminal environments
  - It is used by: htop, less, nano, dialog nmtui, installers/configuration tools
- vim, emacs, tmux implement their own screen/rendering engtine & target VT100/ANSI-like terminal behavior

Canonical Mode: Escape Characters

Characters that are handled specially in canonical mode

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	Line feed (unchangeable)

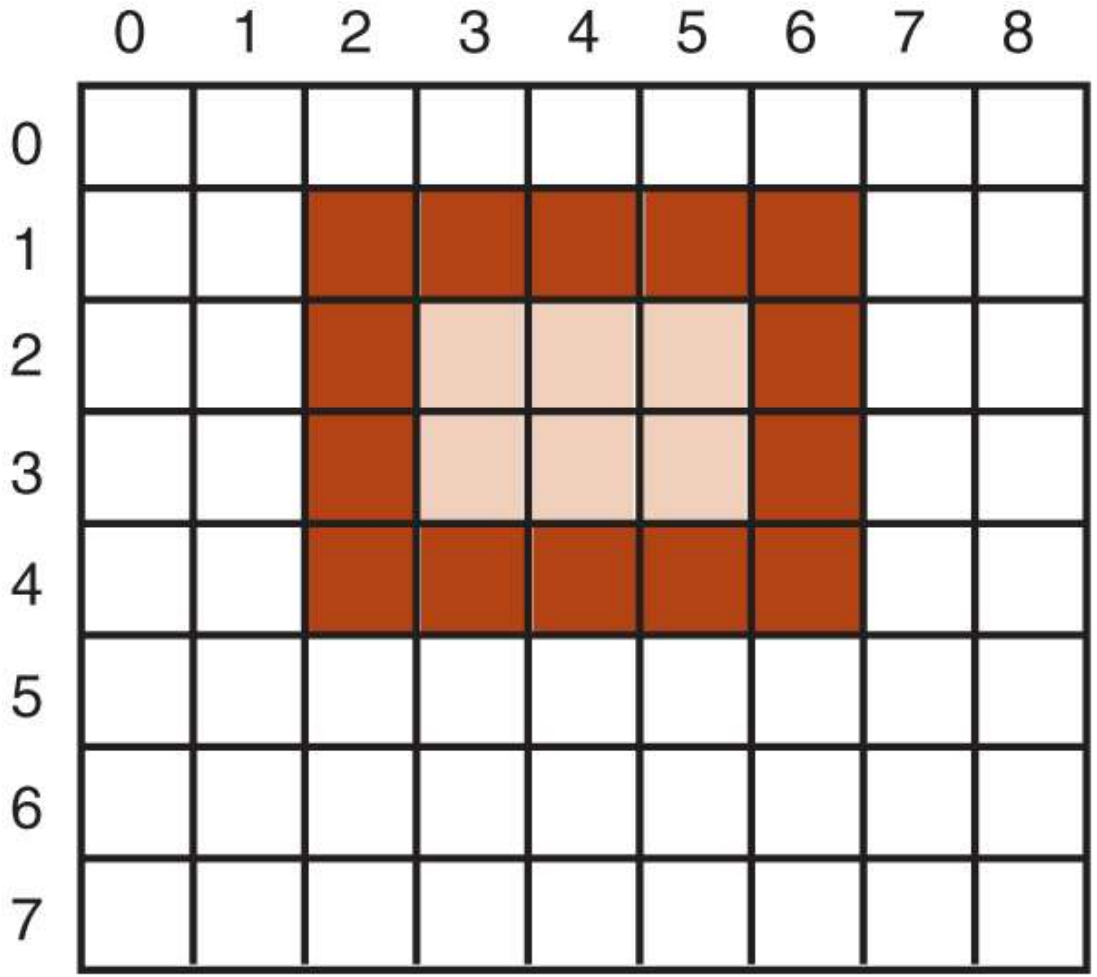
Escape Sequences

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.

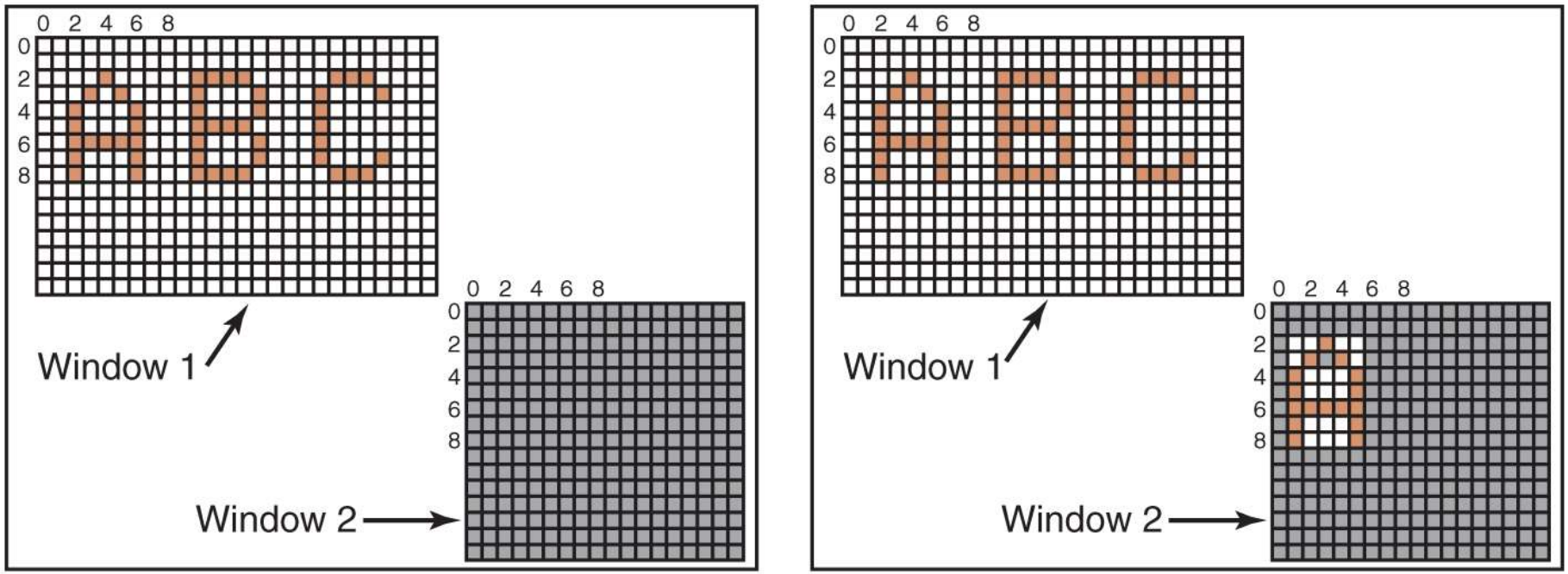
Escape sequence	Meaning
ESC [*n* A	Move up *n* lines
ESC [*n* B	Move down *n* lines
ESC [*n* C	Move right *n* spaces
ESC [*n* D	Move left *n* spaces
ESC [*m* ; *n* H	Move cursor to (*m*,*n*)
ESC [*s* J	Clear screen from cursor (0 to end, 1 from start, 2 all)
ESC [*s* K	Clear line from cursor (0 to end, 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

Font Representation: Bitmap

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



BitMap



Copying bitmaps using *BitBlt* function

a) Before

b) After

Vector Rendering

20 pt: abcdefgh

53 pt: abcdefgh

81 pt: abcdefgh

Bitmap: pixel-based → loses quality when enlarged

Vector: math-based (splines) → scales without losing quality

Pointing Devices: Mouse

Device provides

- delta x and delta y changes (in mickeys, usually 0.1cm)
- Buttons
- Delta wheel changes

Device driver

- determines single vs double click
- pointer speed

Rendering Engines (Two Domains)

Domain A: Graphics Stack

- Direct GPU APIs: Interfaces that talk directly to the GPU to produce frames
  - OpenGL, Direct3D, Vulkan, Metal
    - triangles, textures, shaders, pre-assume an external presentation surface: drawable, swapchain, or framebuffer target provided by OS or runtime
- 2D Rendering Engines: turn UI descriptions to pixels
  - Skia, Cairo (used in GTK)
    - 2D drawing, surface composition, text rendering, GPU-accelerated rasterization

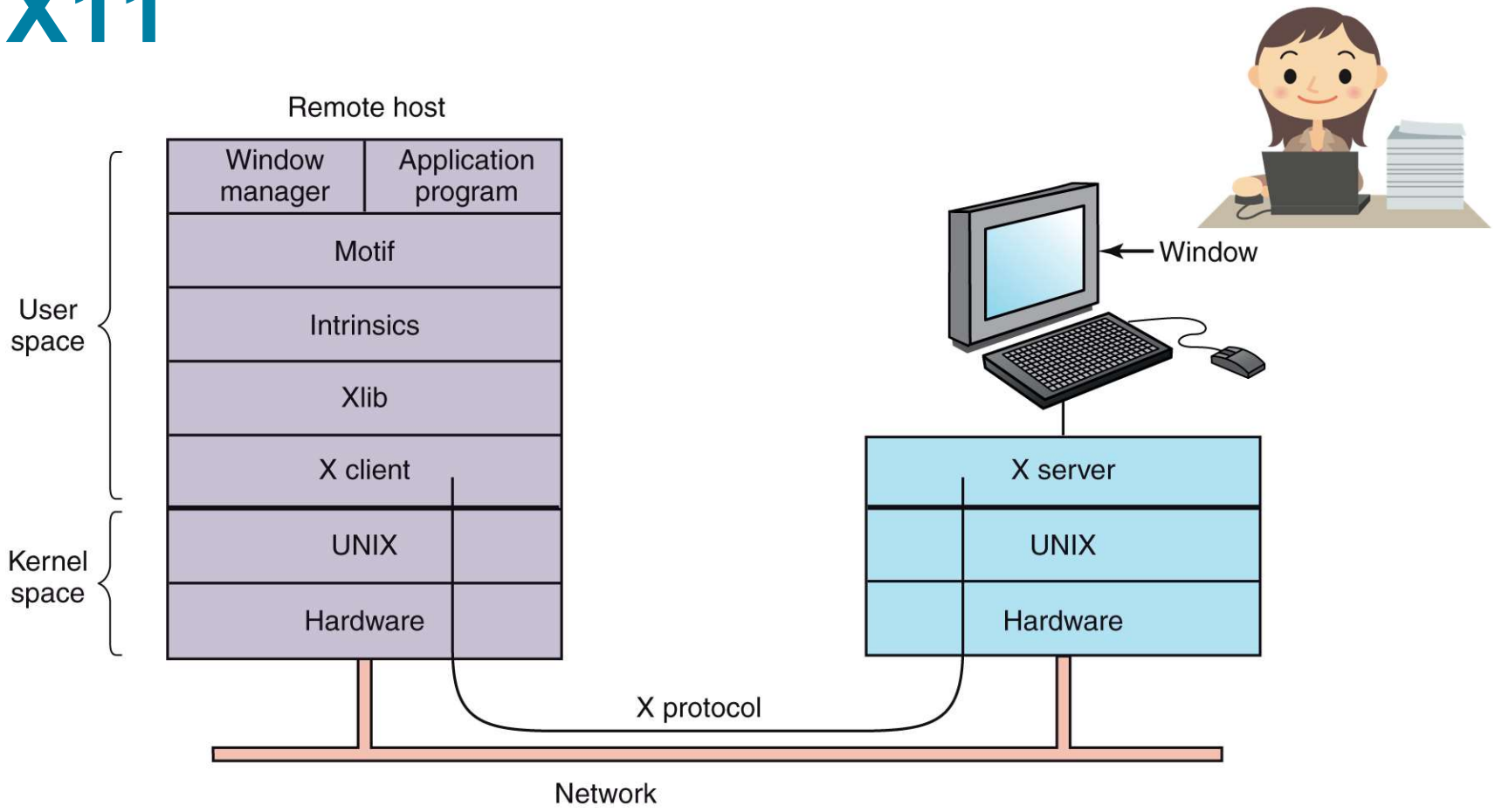
Domain B: Compute Stack

- General Purpose GPU Computation Layer
  - OpenCL/CUDA GPGPU compute layer
    - image processing, machine learning, vision pipelines
- Computer Vision Acceleration Layer
  - OpenCV for computer vision with OpenCL/CUDA acceleration

Windowing Platforms & Frameworks

- Platform Windowing Systems
  - X11 & Wayland client/server protocols
    - windows, input events, basic drawing surfaces, X11 is network-transparent
  - Windows Win32 GUI, macOS Cocoa
- Thin Graphics Systems: Multimedia Abstraction Libraries (C/C++-oriented)
  - SDL/SFML: X11/Wayland/Win32 + OpenGL/Vulkan or software
    - windows, input events, 2d surfaces/textures/audio context (OpenGL/Vulkan)
- GUI frameworks with standard UI components
  - GTK uses different backends X11/Wayland/… (used by Gnome)
    - widgets (buttons, menus, layouts, text), events, rendering, accessibility
  - Qt full application framework (used by KDE)
    - widgets, events, networking, custom rendering, OpenGL integration
- Heterogeneous UI Frameworks: X11/Wayland/Win32/Cocoa + own rendering
  - Electron (Chromium), Flutter (Skia), React Native, JavaFX/Swing
- End-User Applications: Browsers, Editors, Games, IDEs, OpenCV-based

X11

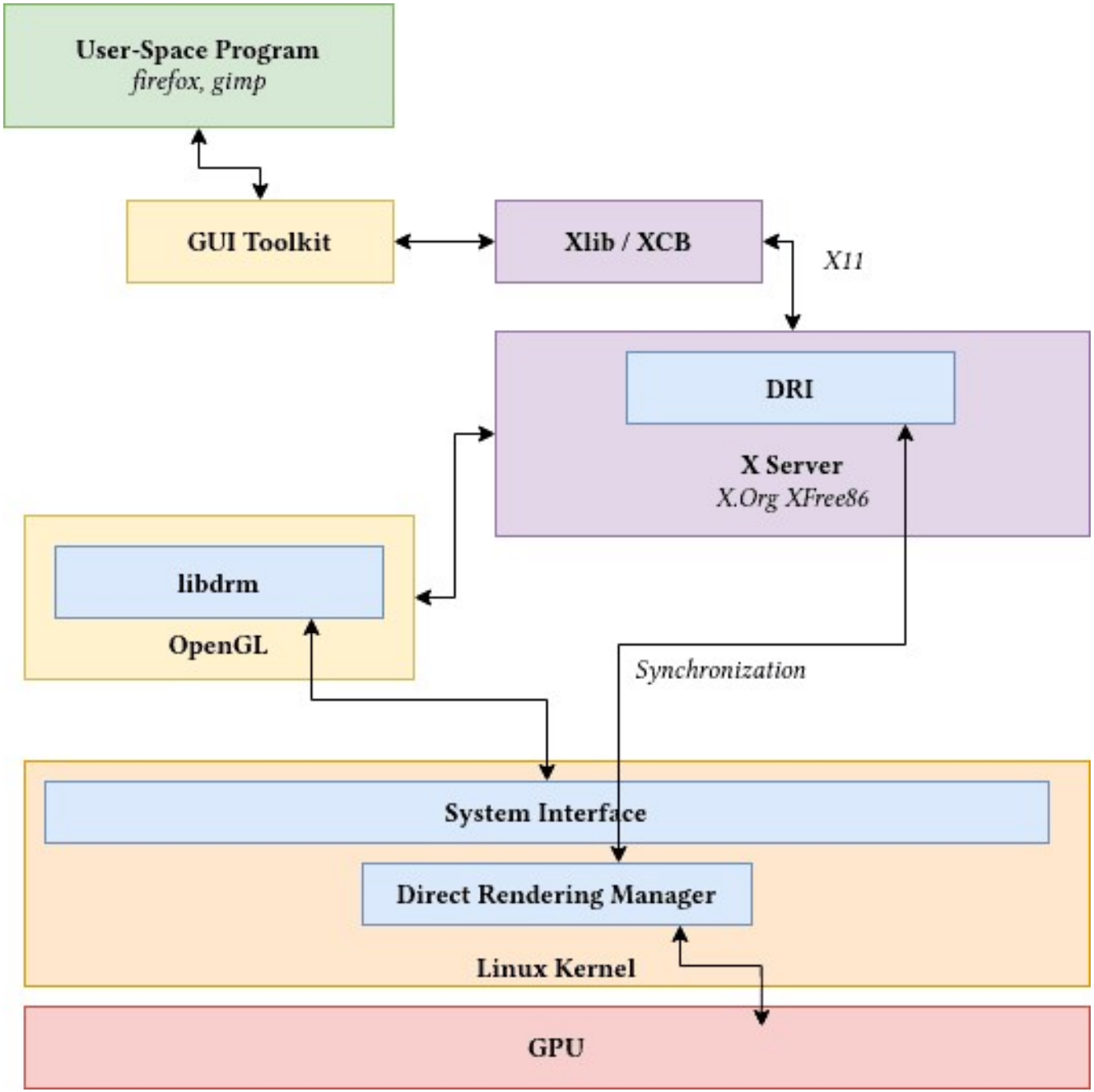


Clients and servers in the X Window System

Wayland improves latency (simpler rendering) & security

Recently, Ubuntu, OpenSUSE, and RedHat adopted Wayland

X11

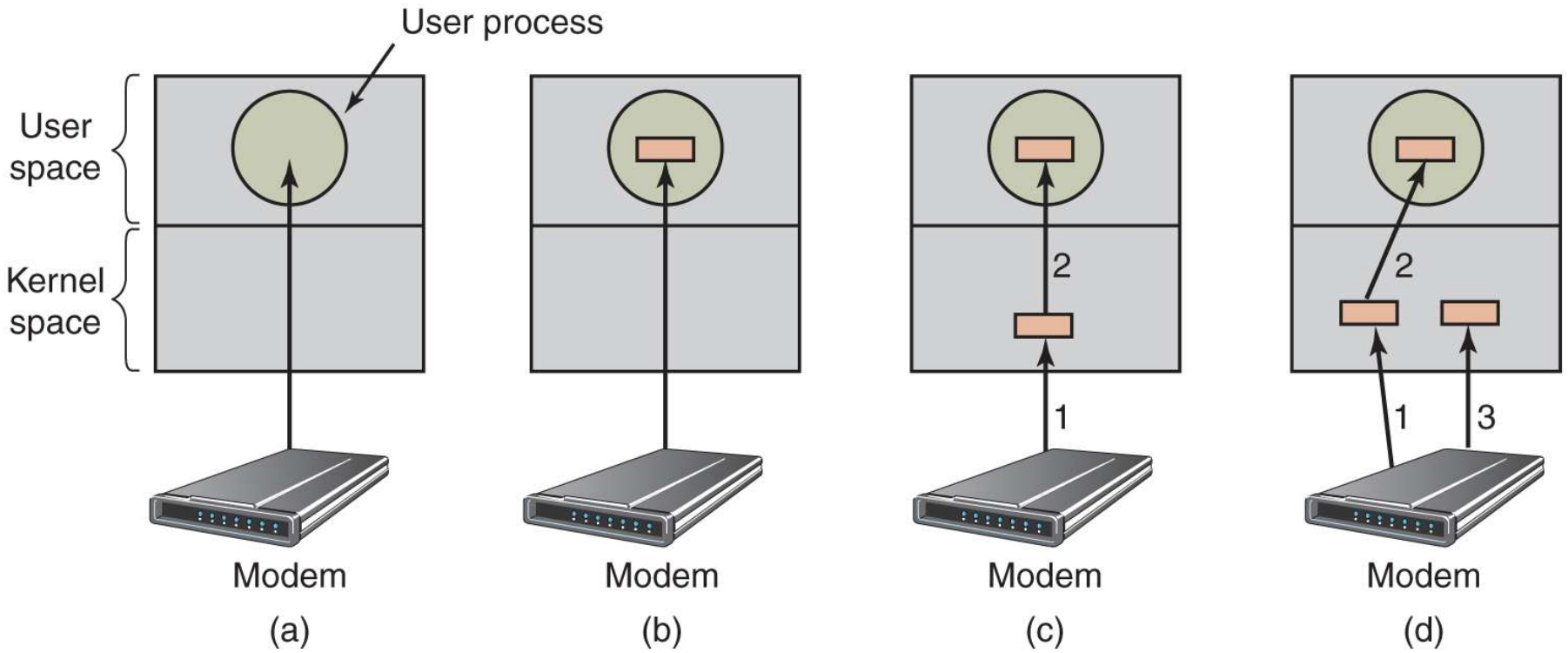

User-Space Program
firefox, gimp
GUI Toolkit
Xlib / XCB
X11
DRI
X Server
X.Org XFree86
libdrm
OpenGL
Synchronization
System Interface
Direct Rendering Manager
Linux Kernel
GPU

X11 Example

A skeleton of an X Window application program.

```
#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 */
}
```

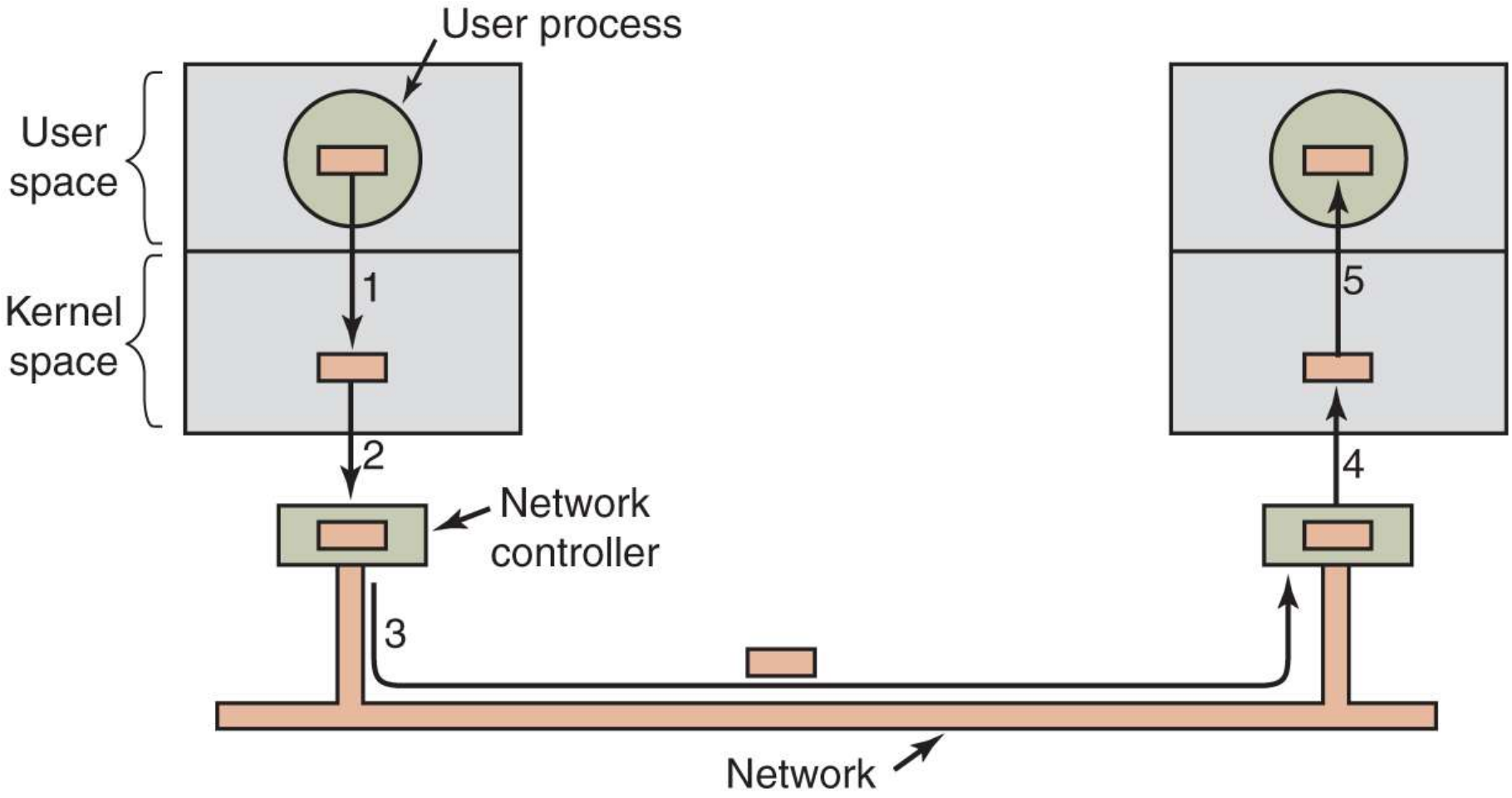
Network Devices



1. Unbuffered input
2. Buffering in user space
3. Buffering in the kernel followed by copying to user space
4. Double buffering in the kernel

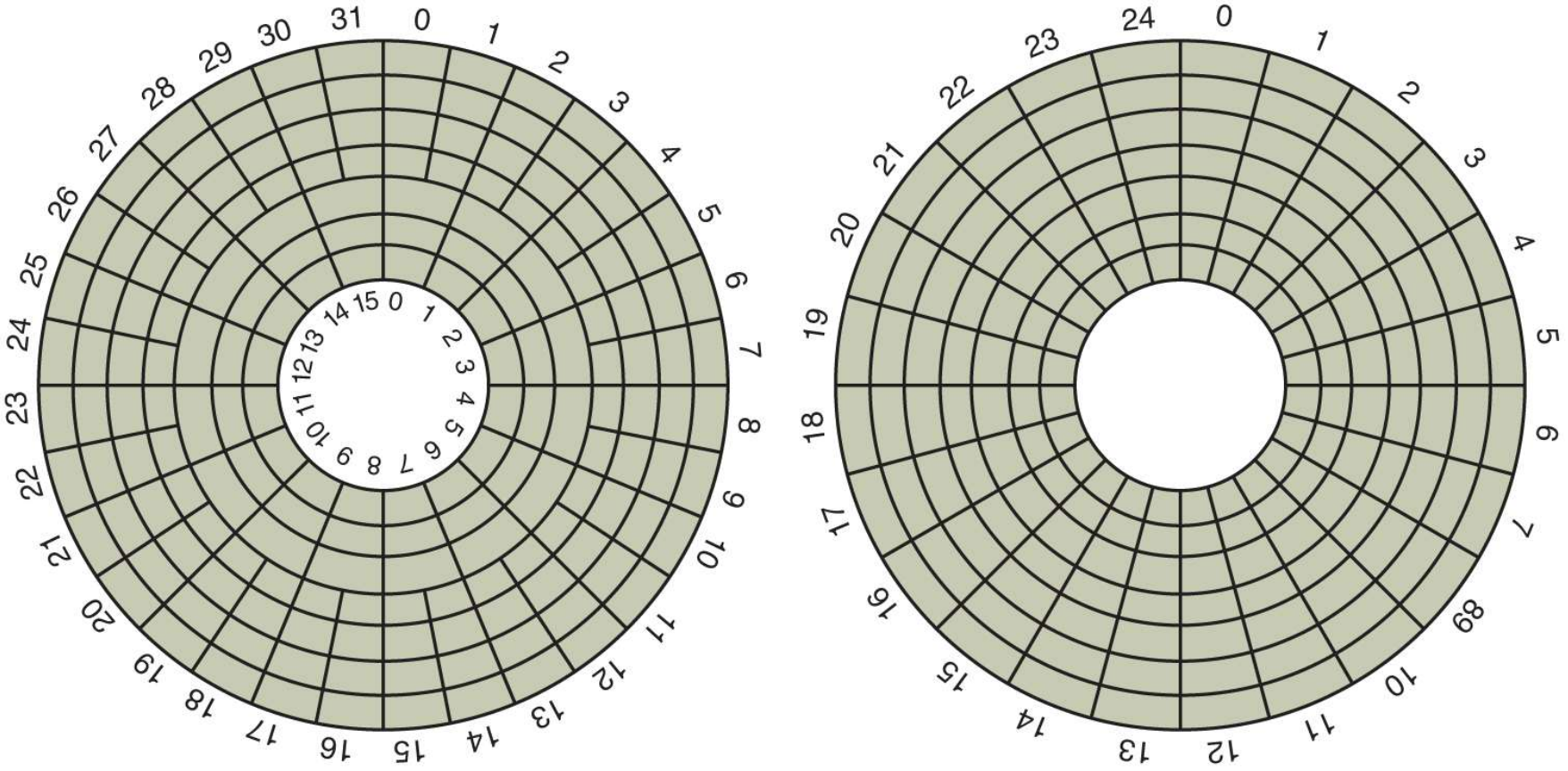
Socket Buffer - Network Controller

Networking may involve many copies of a packet.



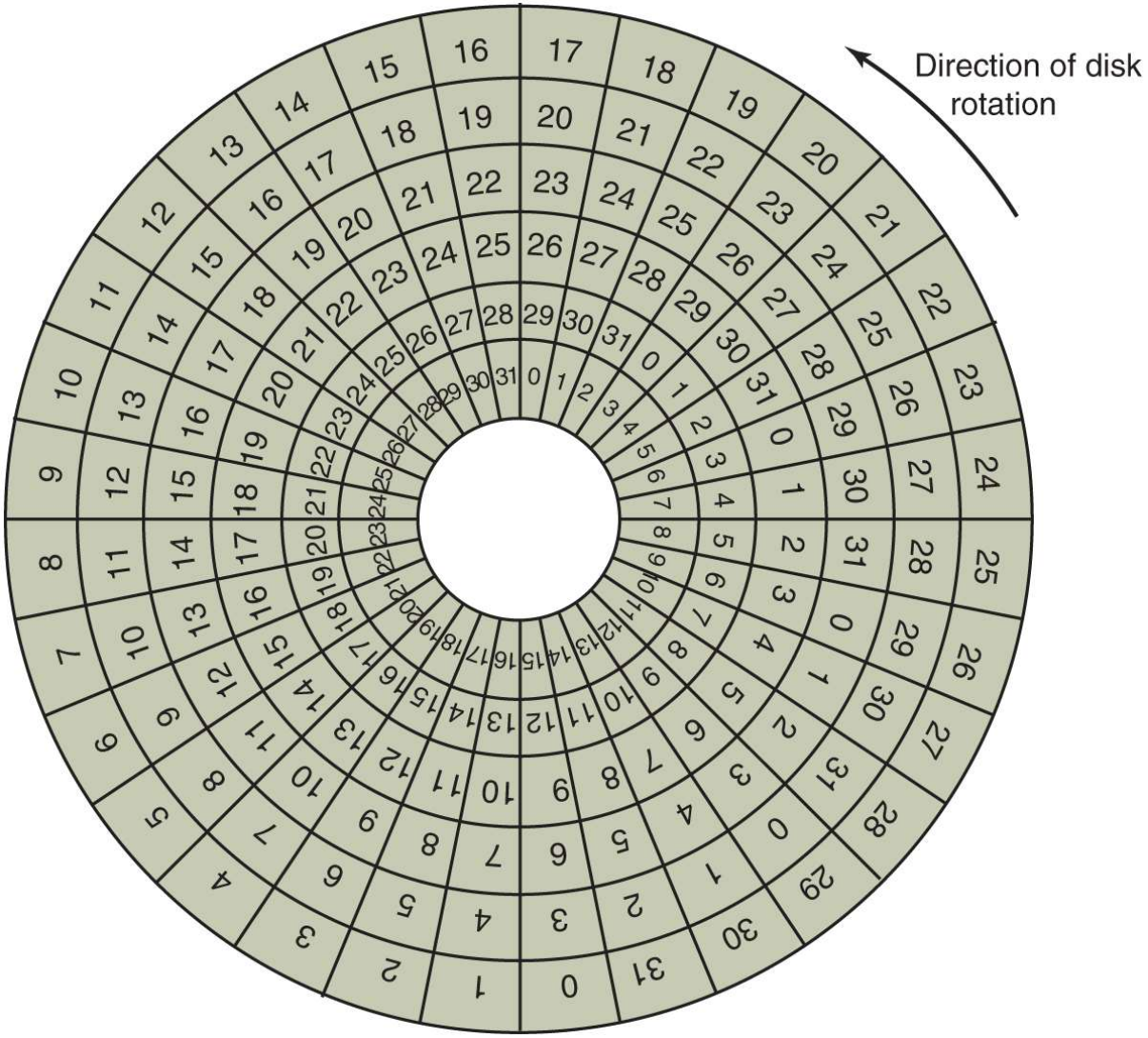
Disk Geometry

1. Physical geometry of a disk with two zones
2. A possible virtual geometry for this disk.



Disk Geometry

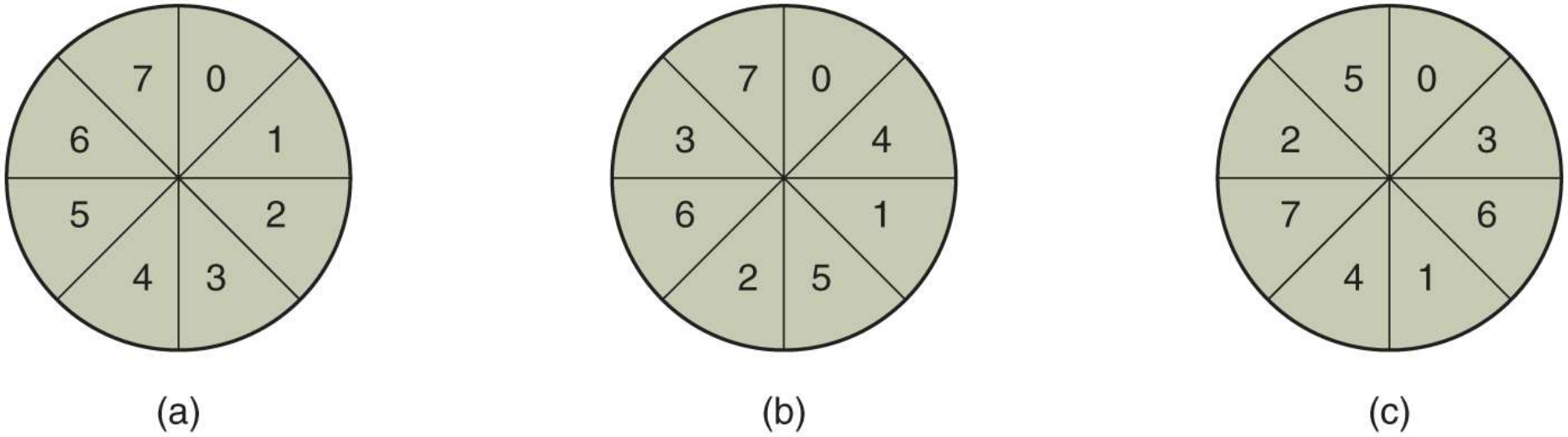
An illustration of cylinder skew.



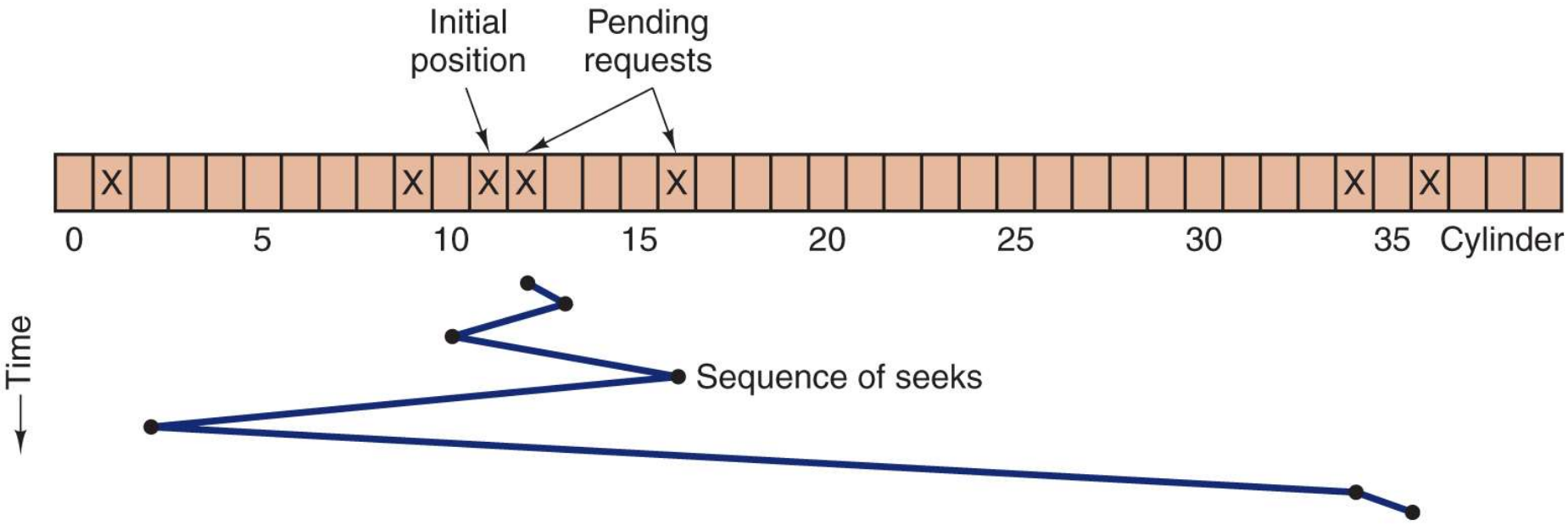
Disk Sector

Disk Sector Interleaving (Floppies)

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

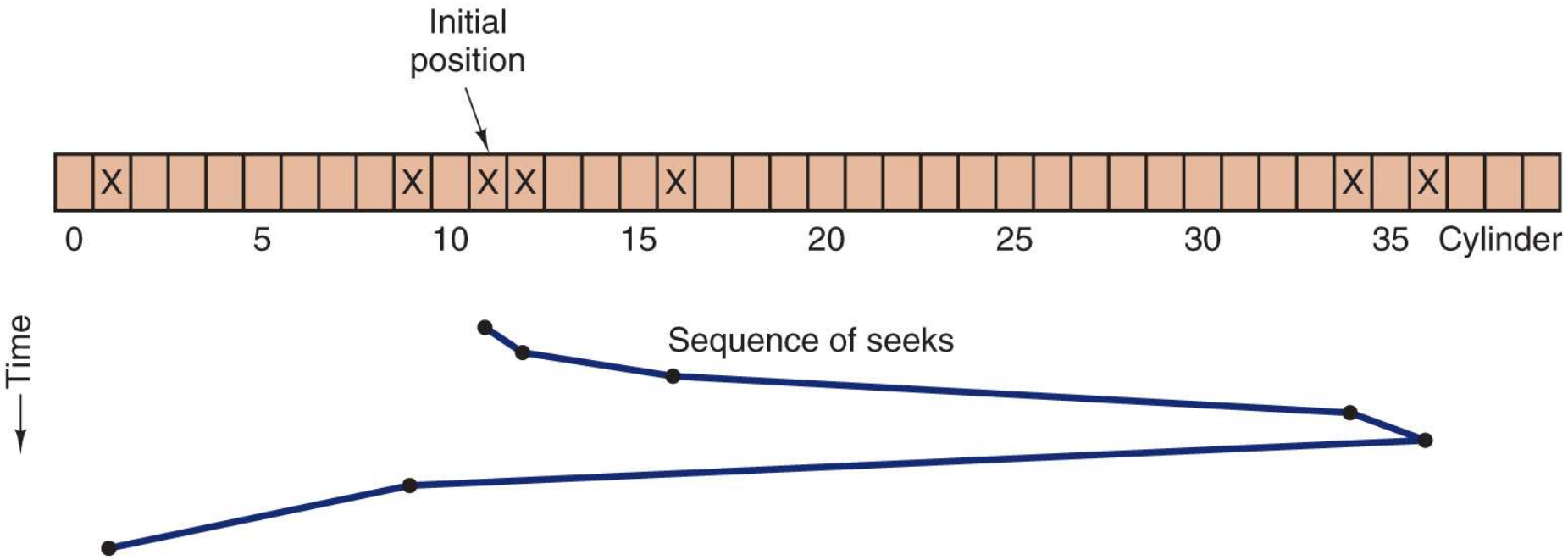


Disk Scheduling: Shortest Seek First



- Minimizes head movement
- Reduces seek time (HDDs), but
- Starvation (far requests may wait forever)?
- Fairness?

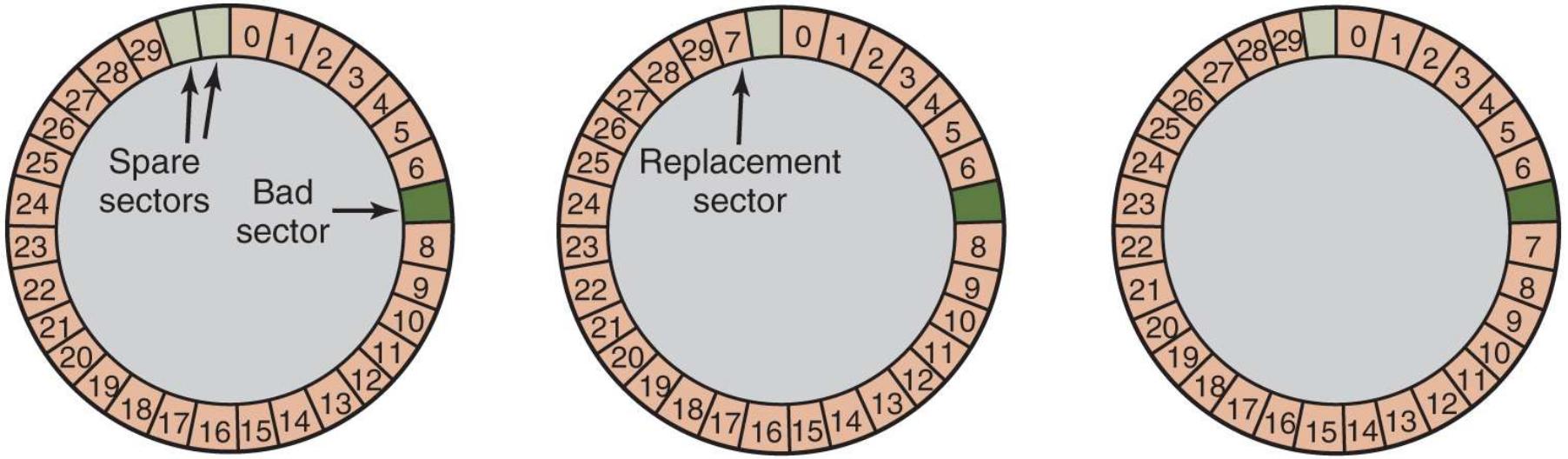
Elevator Algorithm



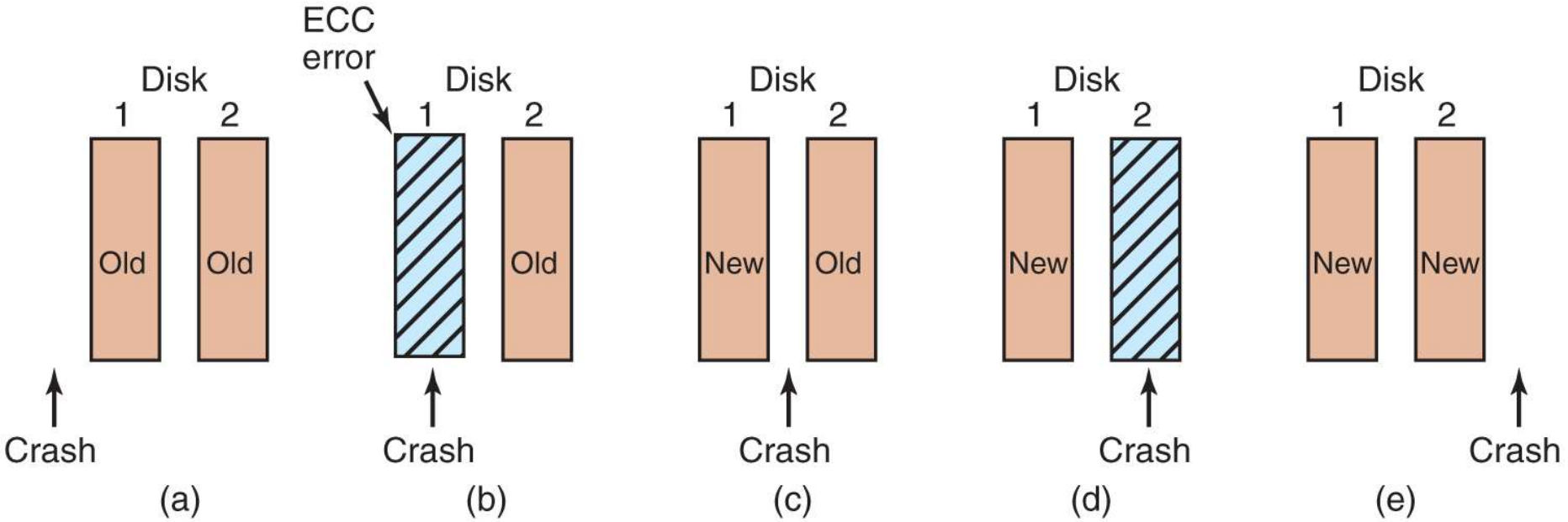
- Common variants: SCAN, LOOK, C-SCAN, C-LOOK
- Modern SSD schedulers (Kyber) focus on: fairness, latency, parallel I/O, queue depth

Fault Tolerance

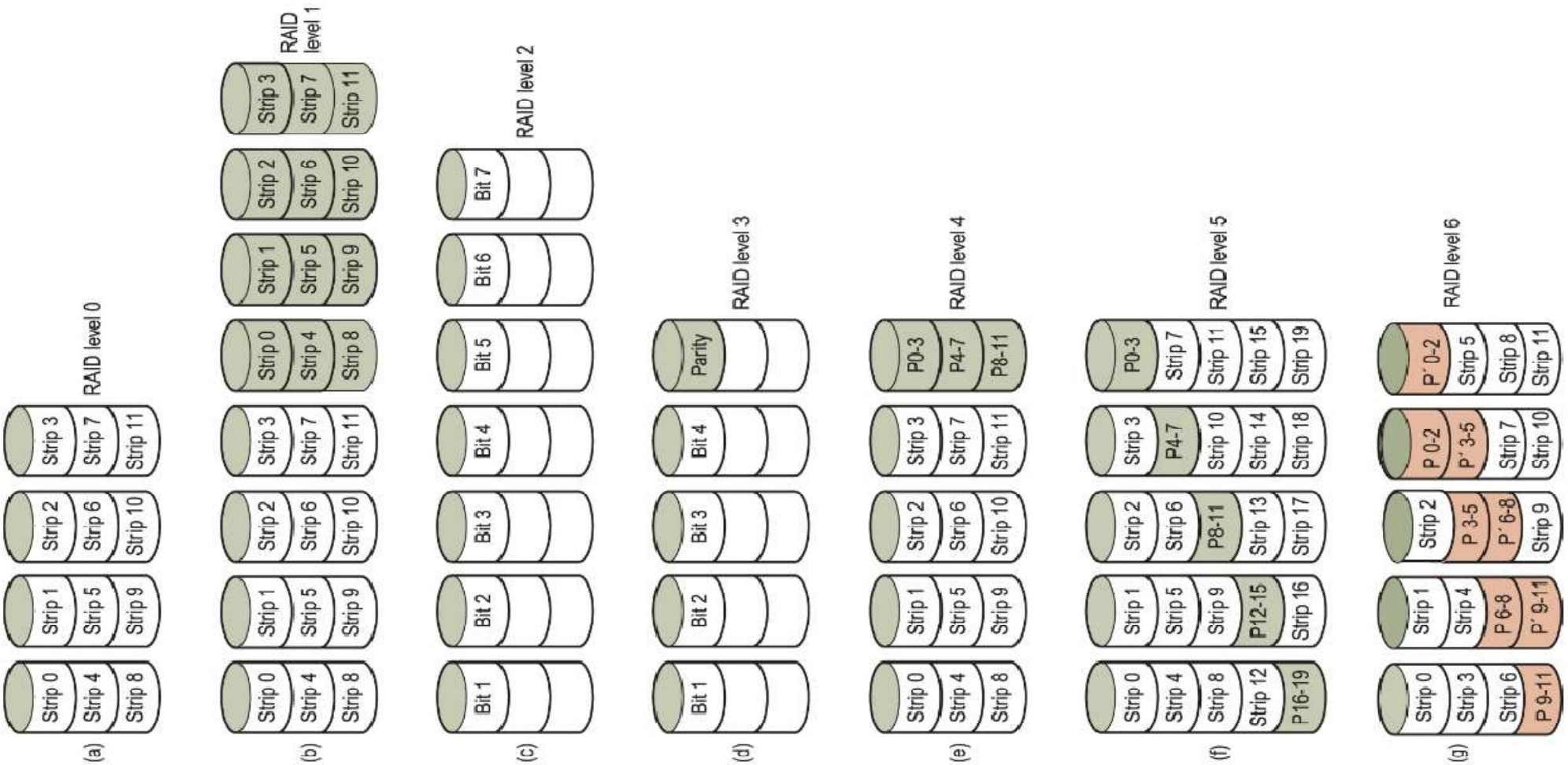
(a) A disk track with a bad sector. (b) Substituting a spare for the bad sector. (c) Shifting all the sectors to bypass the bad one.



Influence of Crashes on Stable Writes



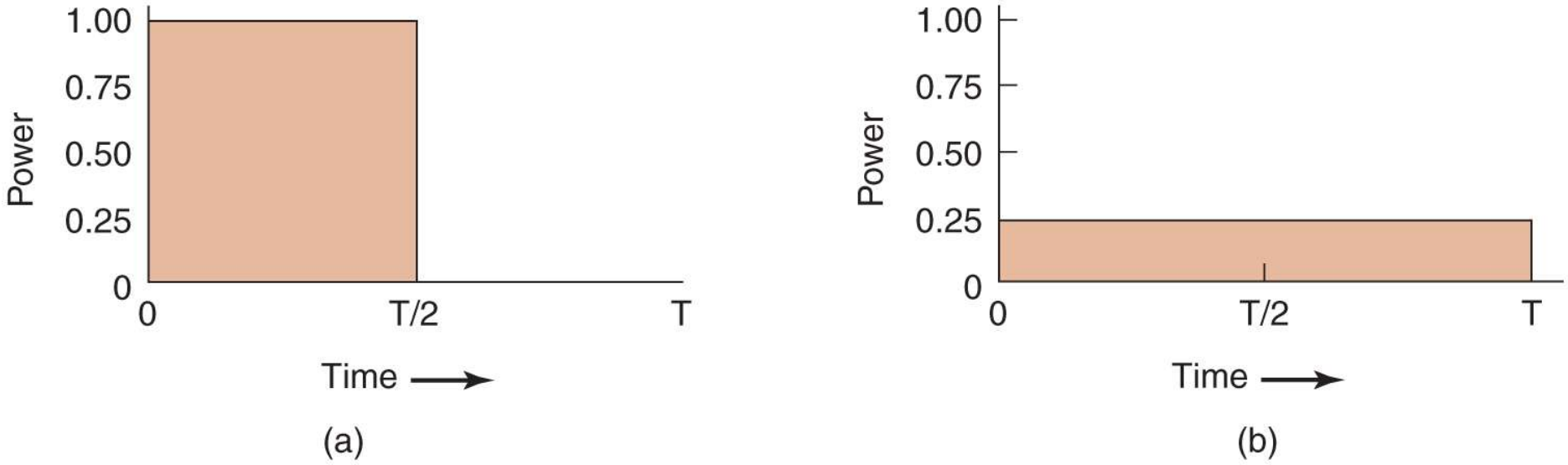
RAID



- RAID levels 0 through 6.
- Backup and parity drives are shown shaded.

DVFS & Clock Gating

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



C States

Example C-States in modern processors (based on Intel terminology). As these states are model specific, you may well find that they are different for the CPU inside your own computer.

State	Name	Meaning
C_0	Active	CPU executes instructions.
C_1	Auto Halt	CPU's core clock off , other components (e.g., bus interface and interrupt controller still run at full speed so processor can return to execution almost immediately.
C_2	Stop Clock	Core + bus clocks are off, but CPU maintains software-visible state.
C_3	Deep Sleep	Even the clock generator is off. CPU flushes internal caches. No snooping/cache coherence.
C_4	Deeper Sleep	Brilliant name for a state where the CPU voltage is reduced also.
C_n	...	(More C-states are possible.)