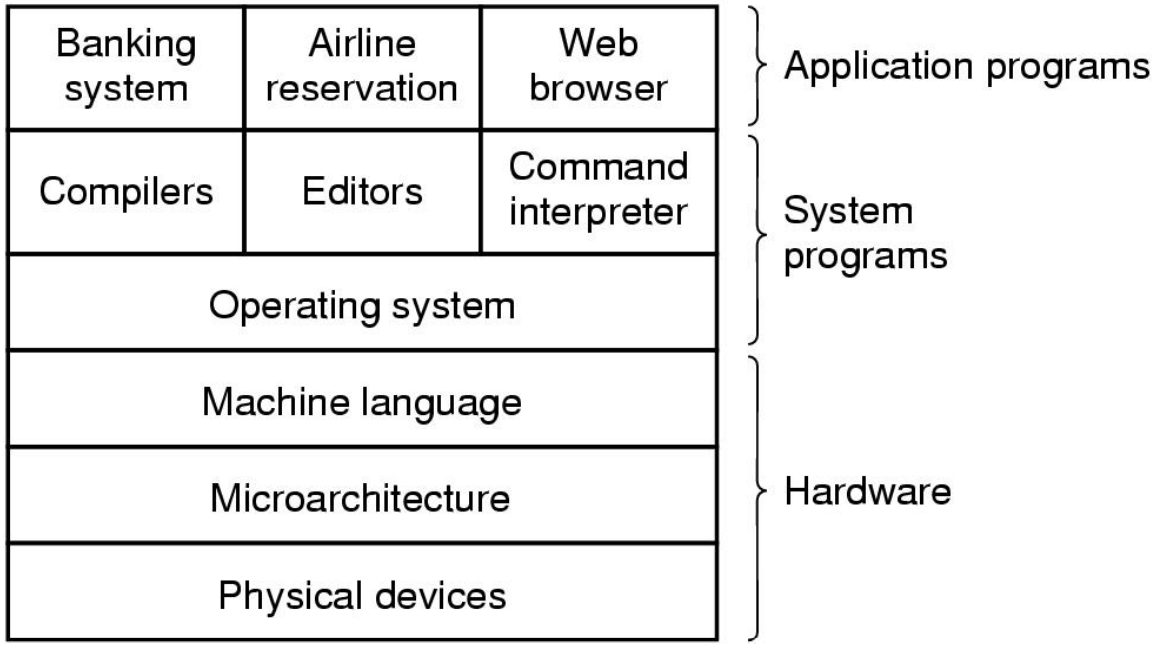
Computer and OS System Overview

Introduction



- A computer system consists of
  - hardware
  - system programs
  - application programs

Operating System

- Provides a set of services to system users (collection of service programs)
- Shield between the user and the hardware
- Resource manager:
  - CPU(s)
  - memory and I/O devices
- A control program
  - Controls execution of programs to prevent errors and improper use of the computer

Operating System Definition (Cont)

- No universally accepted definition
- "Everything a vendor ships when you order an operating system" is good approximation
  - But varies wildly
- "The one program running at all times on the computer" is the **kernel**. Everything else is either a system program (ships with the operating system) or an application program

Computer system overview: starting from 0

Basic functionality of a computer system: Instruction Cycle

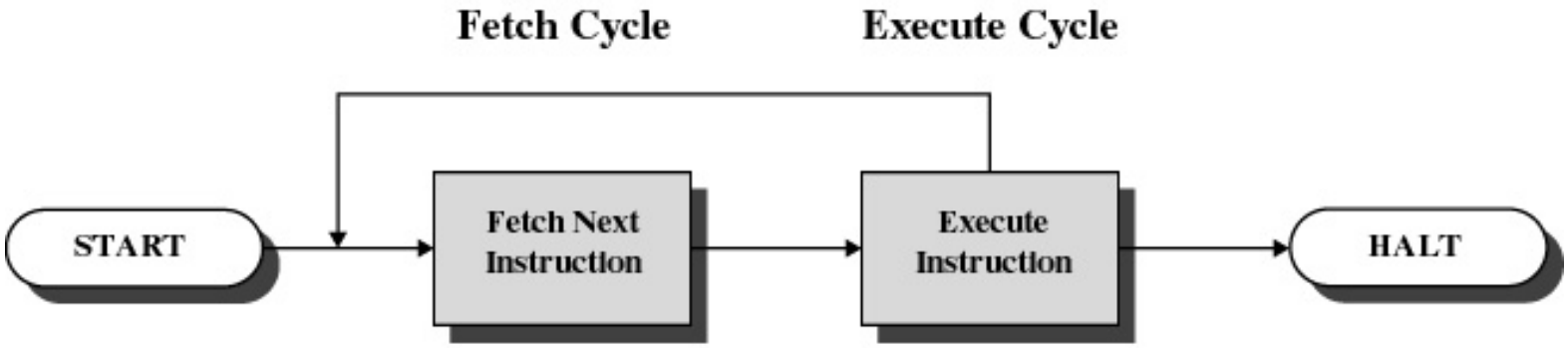
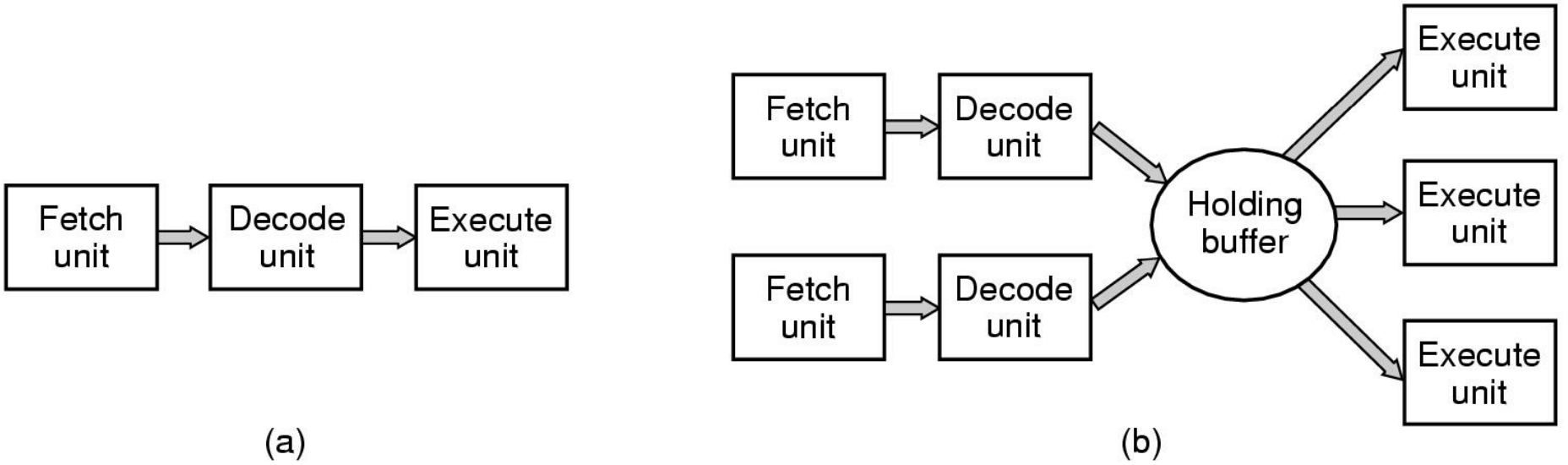


Figure 1.2 Basic Instruction Cycle

Parenthesis:
A closer-to-reality-view of todays' processors



(a) A three-stage pipeline
(b) A superscalar CPU
(c) Multicore CPUs

Basic Elements of a Computer System

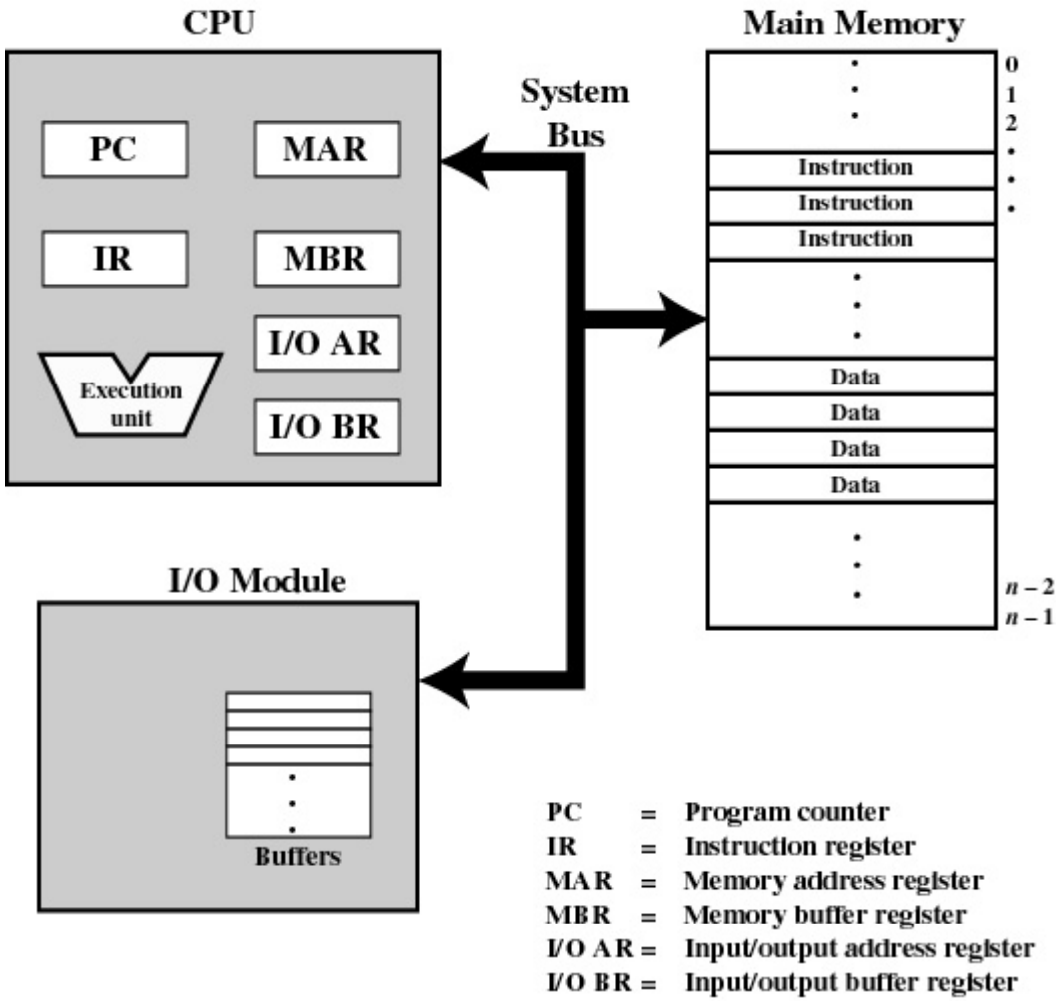


Figure 1.1 Computer Components: Top-Level View

- Processor + registers
- Main Memory ("real" or primary memory)
  - volatile
- I/O modules
  - secondary memory devices
  - communications equipment
  - terminals
- System bus
  - communication among processors, memory, and I/O modules

Registers: 1. User-Visible

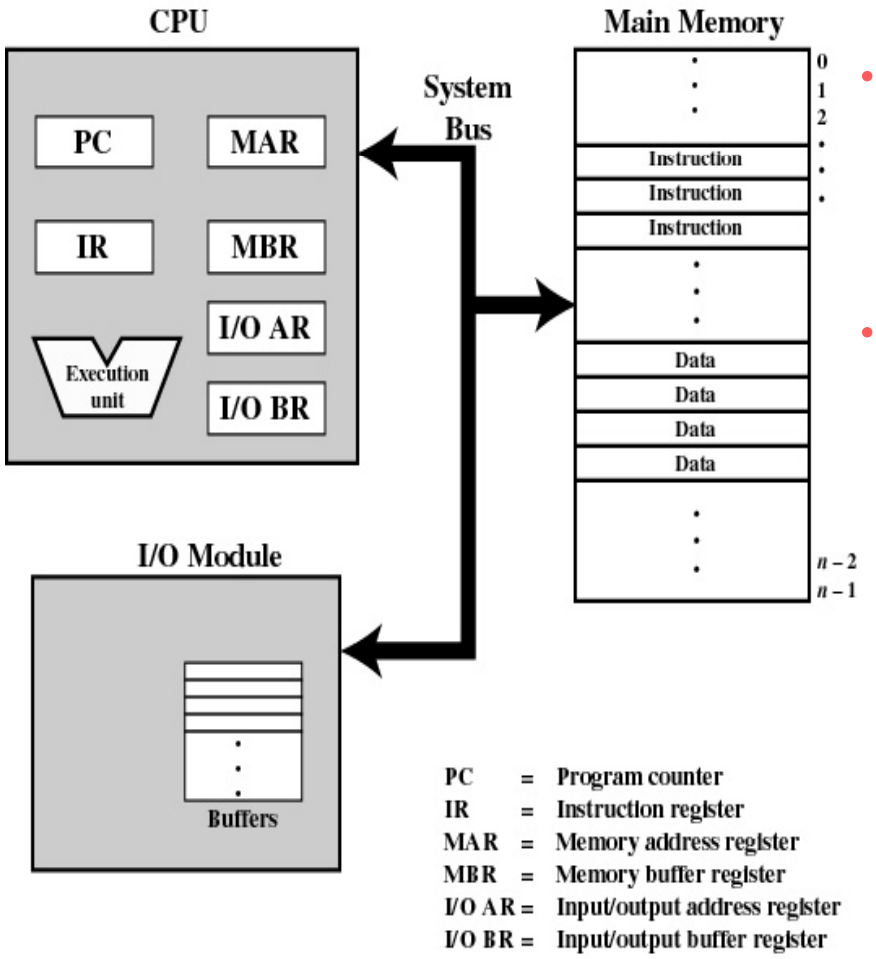


Figure 1.1 Computer Components: Top-Level View

- May be referenced by machine lang.
  - By both application and system programs
- Enable programmer to minimize main-memory references by optimizing register use
- Types of user-visible registers
  - Data
  - Address
    - Index: for indexed addressing, offset
    - Stack pointer: for procedure calling

Registers: 2. Control and Status

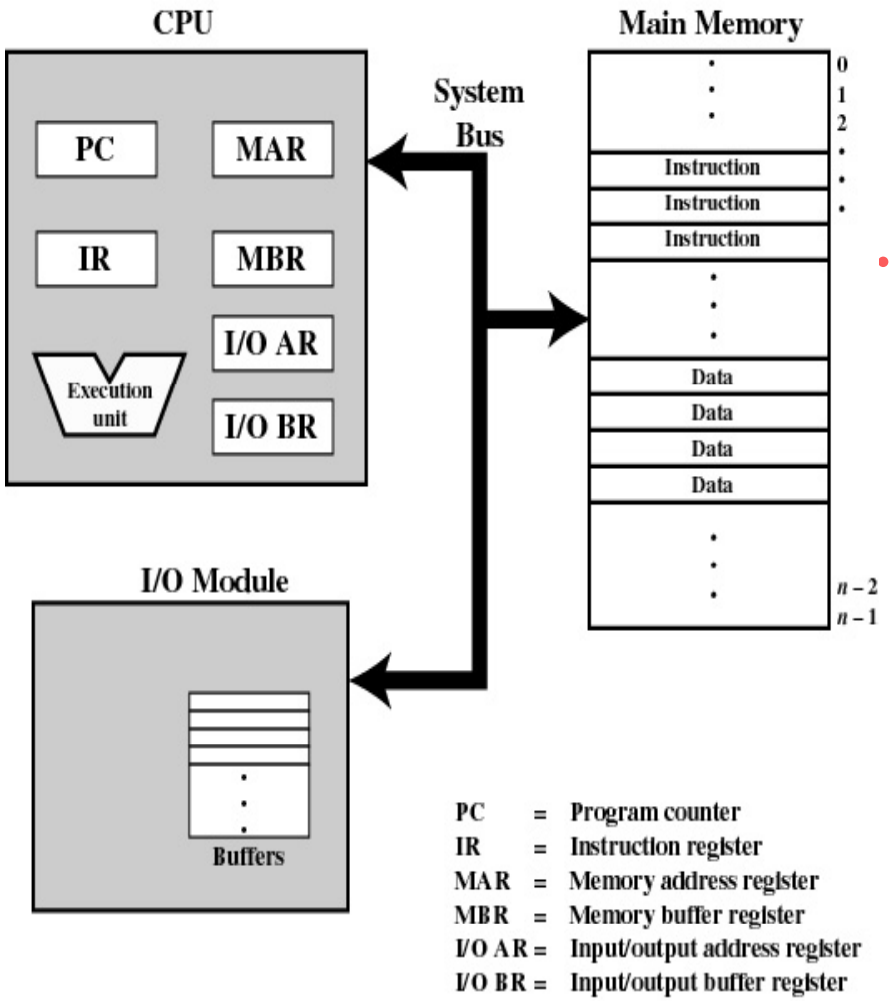


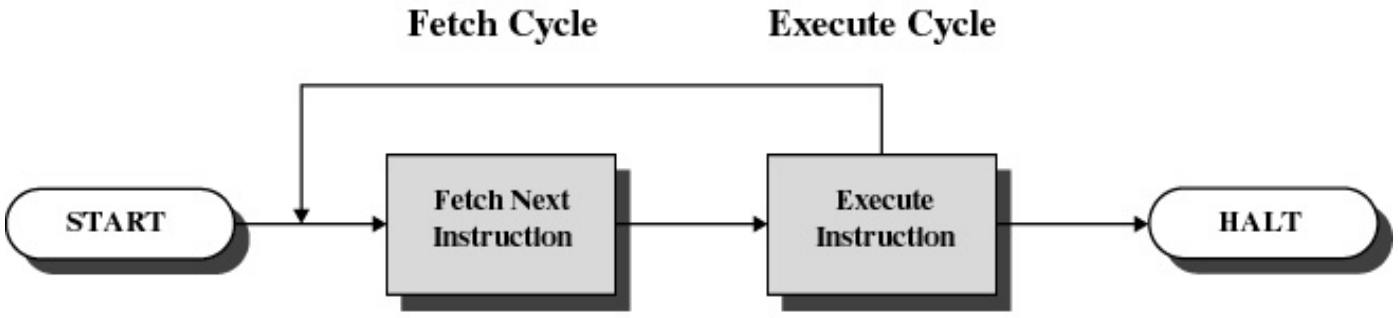
Figure 1.1 Computer Components: Top-Level View

- Used by
  - processor to control execution
  - operating-system to control the execution of programs
- Basic C&S registers:
  - Program Counter (PC)
    - Contains the address of an instruction to be fetched
  - Instruction Register (IR)
    - Contains the instruction most recently fetched
  - Program Status Word (PSW)
    - condition codes (positive/negative/ zero result, overflow, …)
    - Interrupt enable/disable
    - Supervisor/user mode

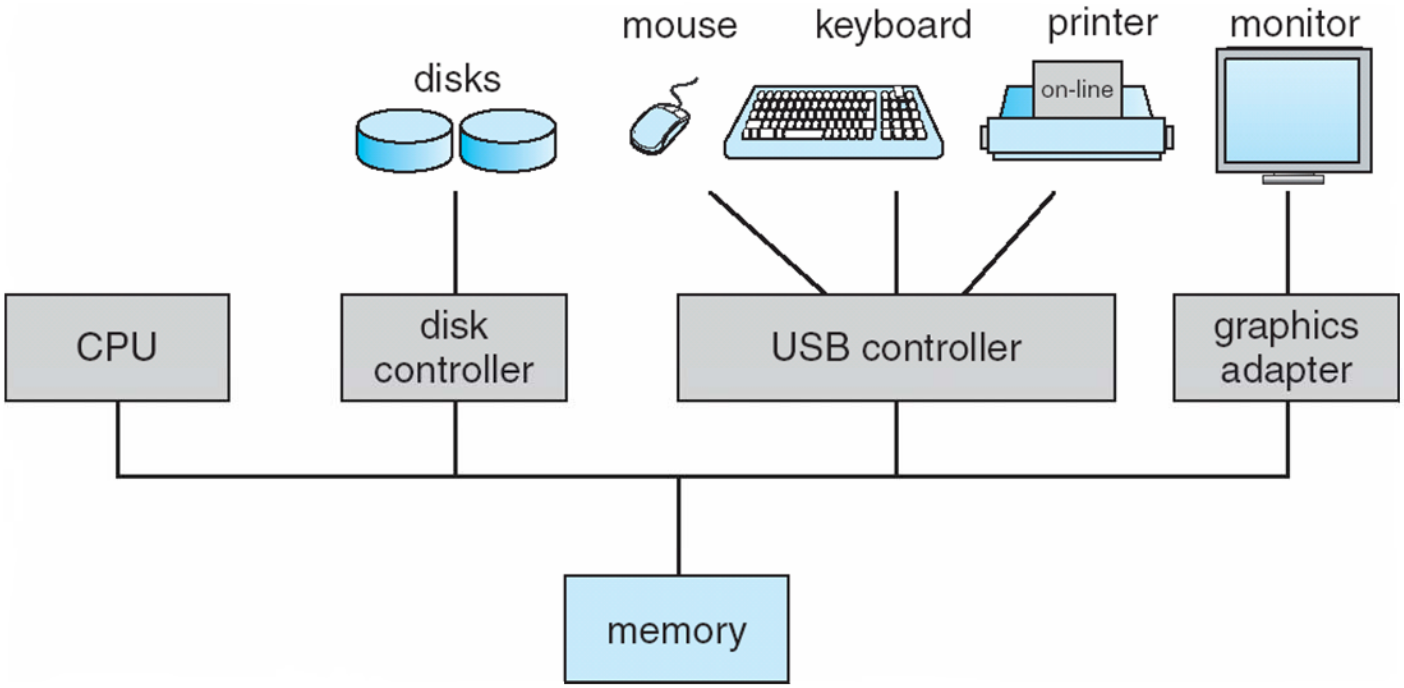
Instruction Cycle revisit

- Processor fetches instruction from memory
  - Program counter (PC) holds address of instruction to be fetched next; PC is incremented after each fetch
  - Fetched instruction is placed in the instruction register

- Types of instructions
  - Processor-memory
  - Processor-I/O
  - Data processing
  - Control: alter sequence of execution



Is that enough for ....



- Components of a simple personal computer

Interrupts!

- An interruption of the normal sequence of execution! Why?
  - Something went wrong (div. by 0, reference outside user's memory space, hardware failure,…)
  - Timer
  - I/O
- and then?
  - Interrupt handler takes control:
    - a program that determines the nature of the interrupt and performs whatever actions are needed
    - generally part of the operating system

Interrupt Cycle

- Processor checks for interrupts
- If no interrupts fetch the next instruction for the current program
- If an interrupt is pending, suspend execution of the current program, and execute the interrupt handler

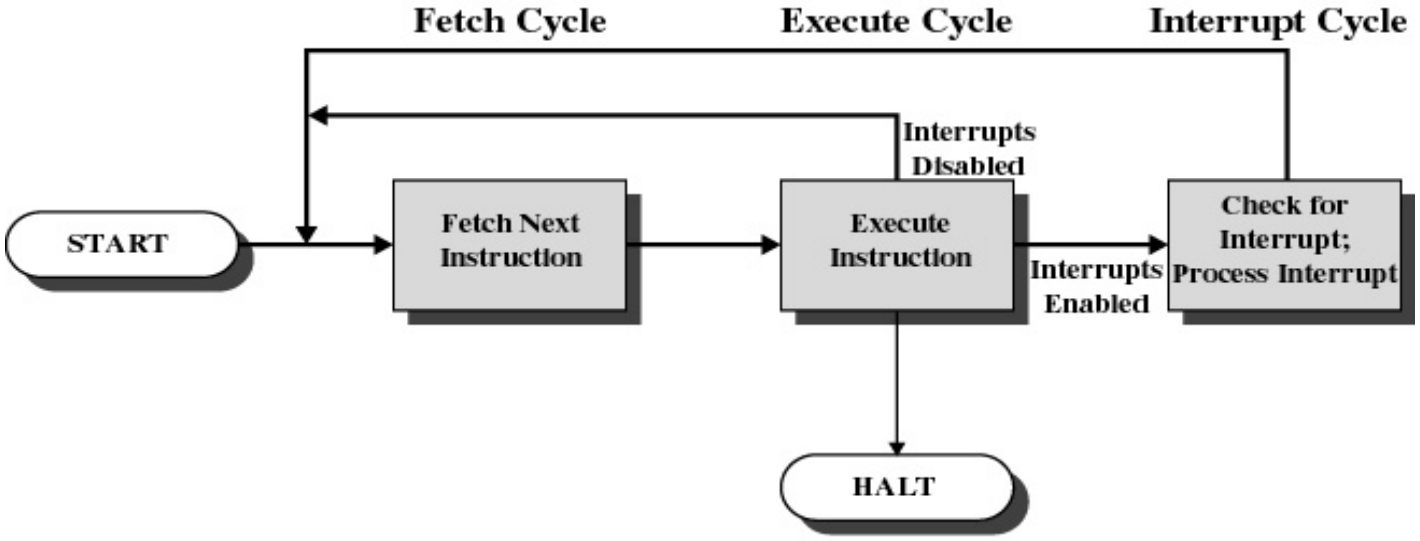


Figure 1.7 Instruction Cycle with Interrupts

Control flow with interrupts

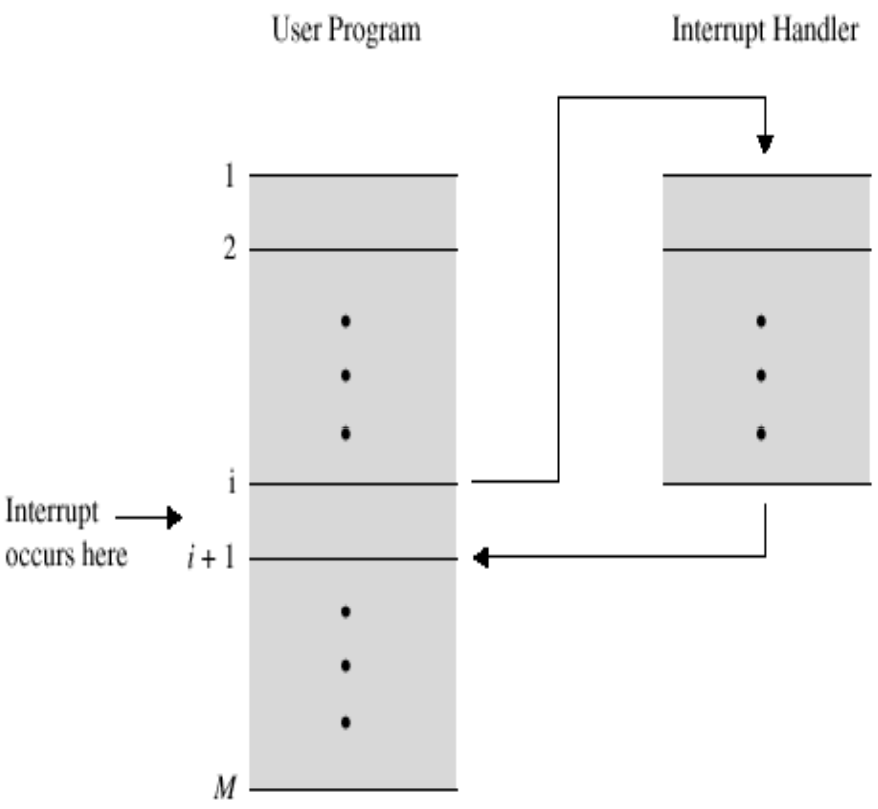


Figure 1.6 Transfer of Control via Interrupts

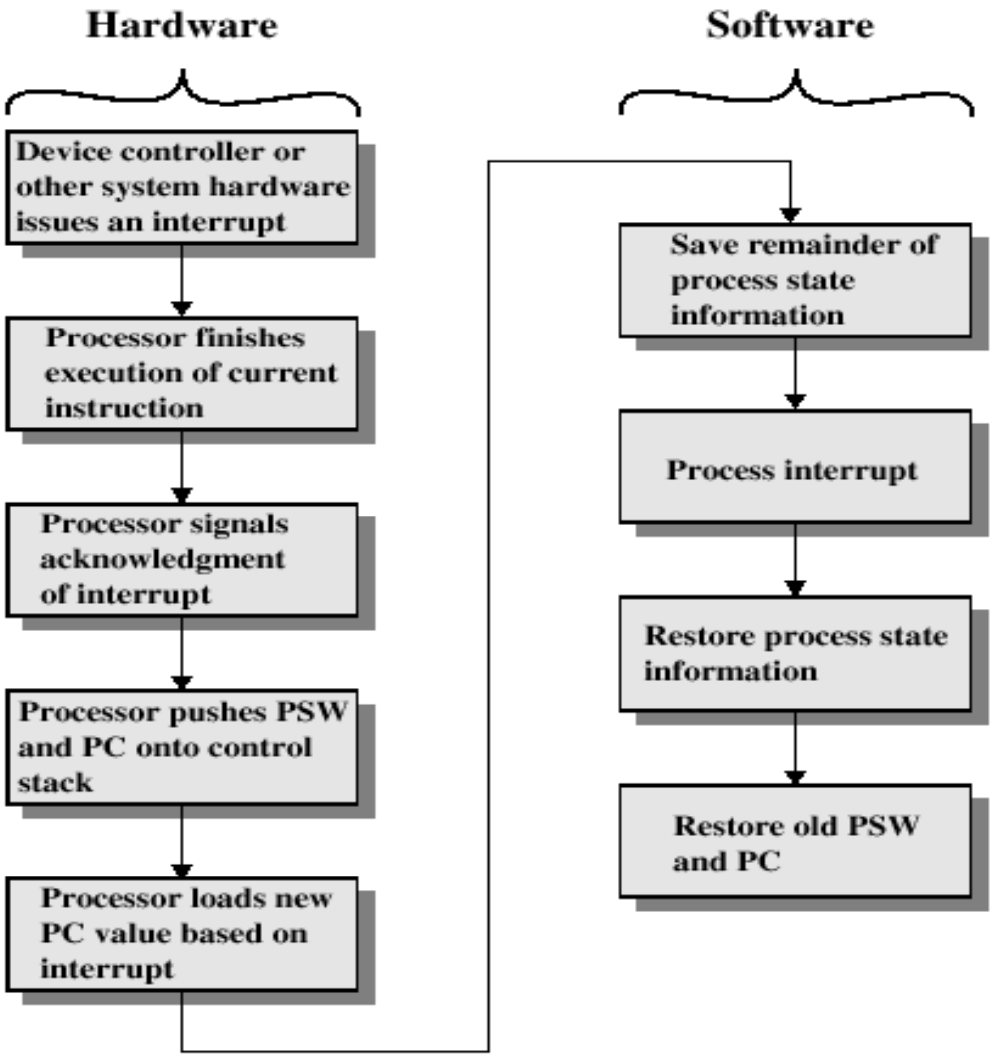
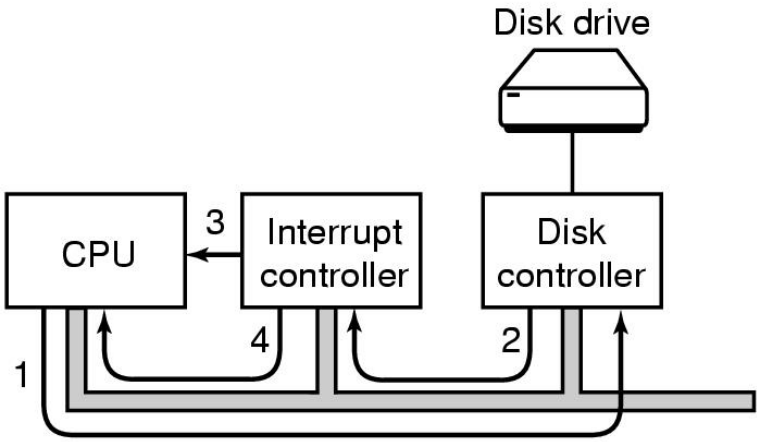
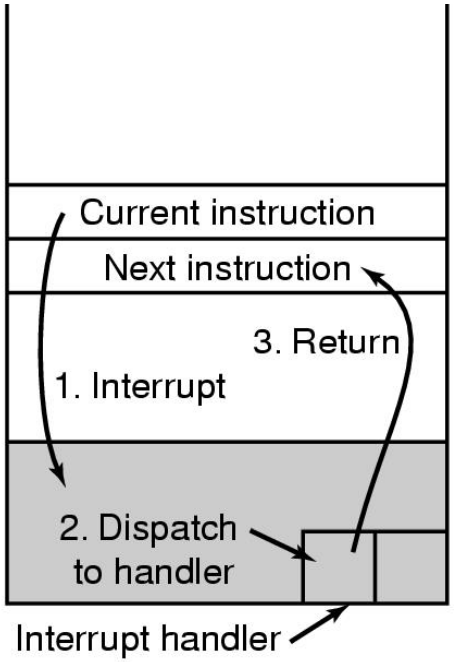


Figure 1.10 Simple Interrupt Processing

What happens



(a)



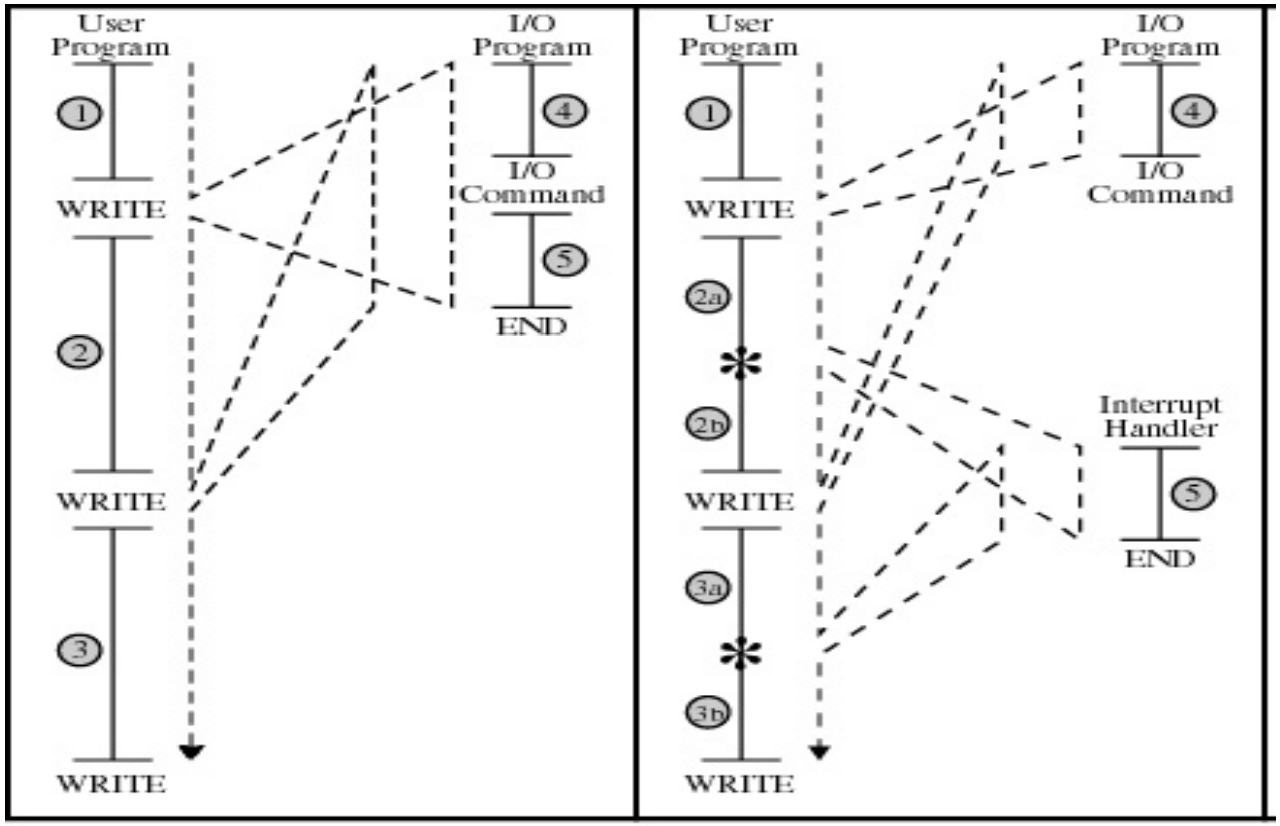
(b)

(a) Steps in starting an I/O device and getting interrupt
(b) How the CPU is interrupted

Interrupts as support for I/O

Note:

- Interrupts allow the processor to execute other instructions while an I/O operation is in progress
- Improve processing efficiency



(a) No interrupts

(b) Interrupts; short I/O wait

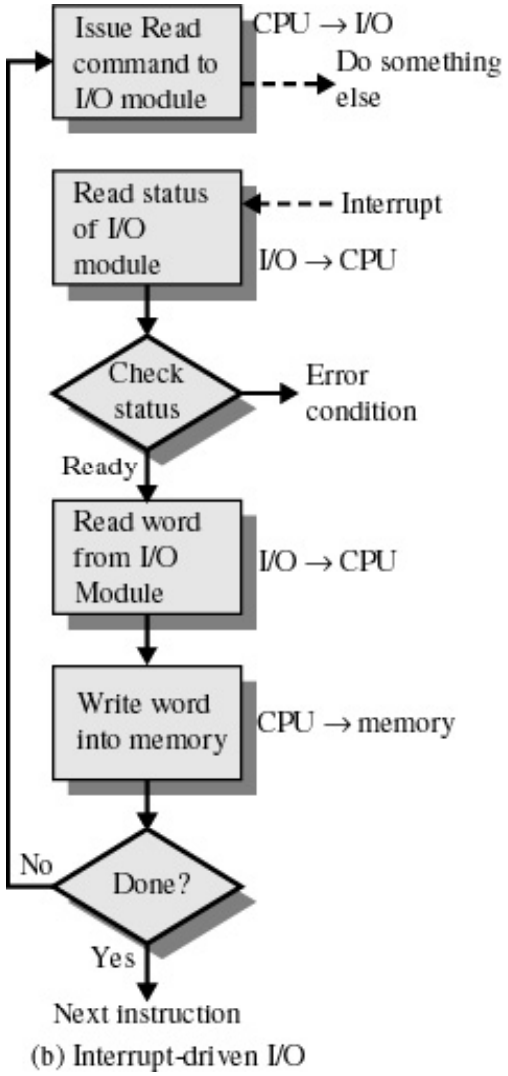
Interrupt-Driven I/O

- Processor is interrupted when I/O module ready to exchange data
- Processor is free to do other work
- No needless waiting

BUT:

- Still consumes a lot of processor time because every word read or written passes through the processor

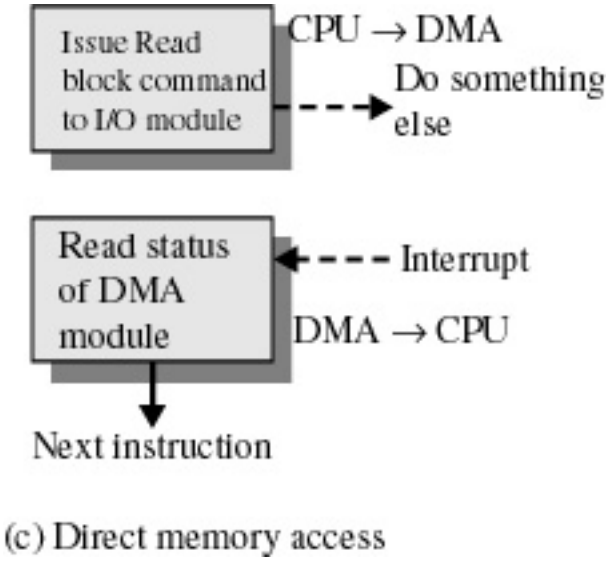
How about using DMA …



(b) Interrupt-driven I/O

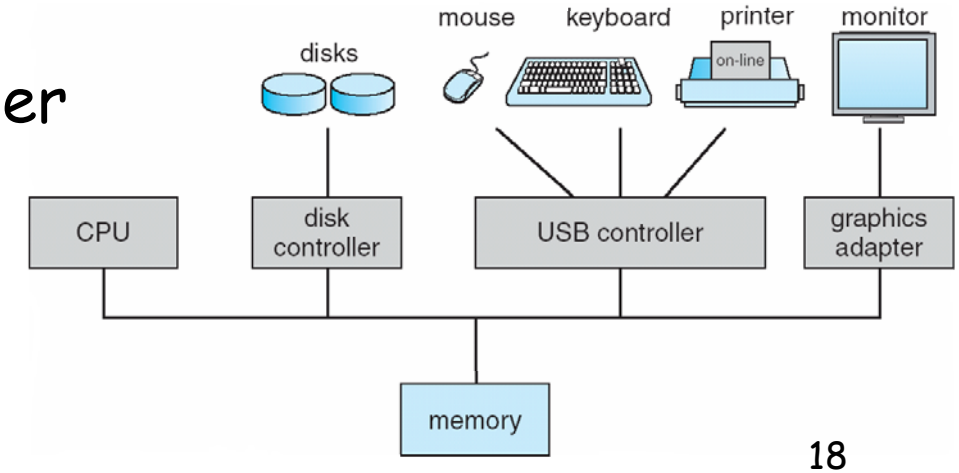
I/O using Direct Memory Access (DMA)

- The processor is only involved at the beginning and end of the transfer
  - Processor grants I/O (DMA) module authority to read from or write to memory a block of data
  - An interrupt is sent when the task is complete



(c) Direct memory access

- Processor is free to do other things



Multiple Interrupts

Q: To interrupt an interrupt?

1. Sequential Order: after interrupt handler completes, processor checks for additional interrupts

2. Priorities: High priority interrupts:
   - cause lower-priority interrupts to wait
   - cause a lower-priority interrupt handler to be interrupted
   - Example: when input arrives from communication line, it needs to be absorbed quickly to make room for more input

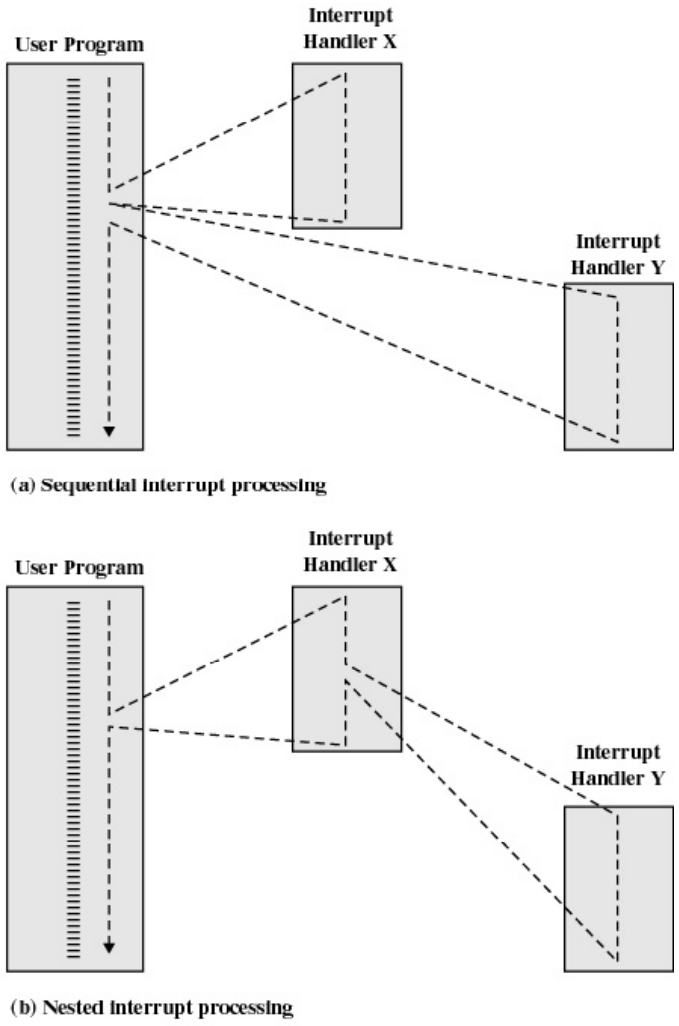


Figure 1.12 Transfer of Control with Multiple Interrupts

Cache Memory

- Increase the speed of memory
  - Processor speed is higher than memory speed

Hit: the information was in cache; else, miss

- Invisible to operating system

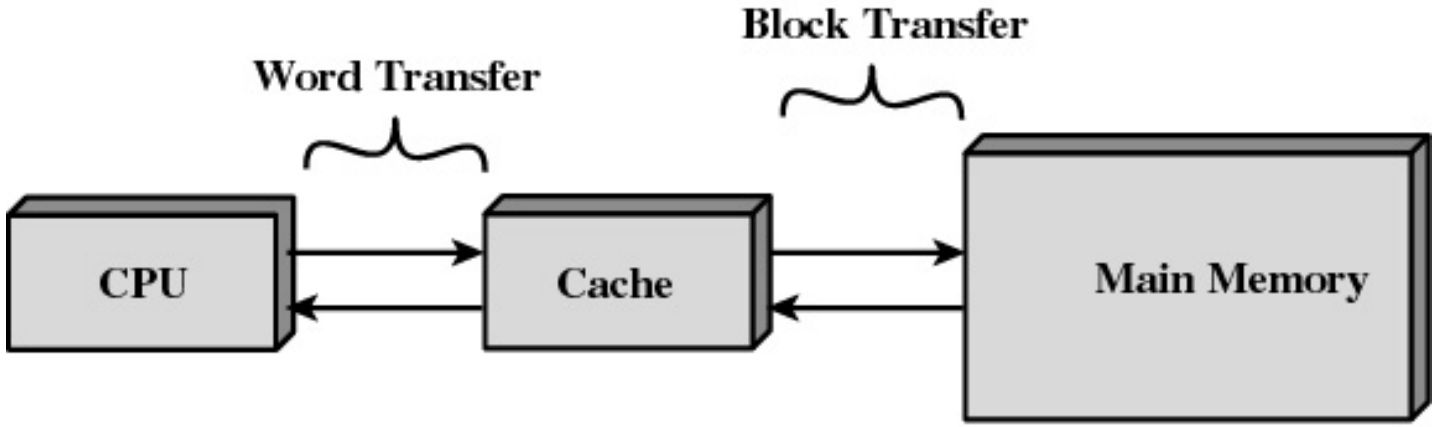


Figure 1.16 Cache and Main Memory

Cache Design: Important issues

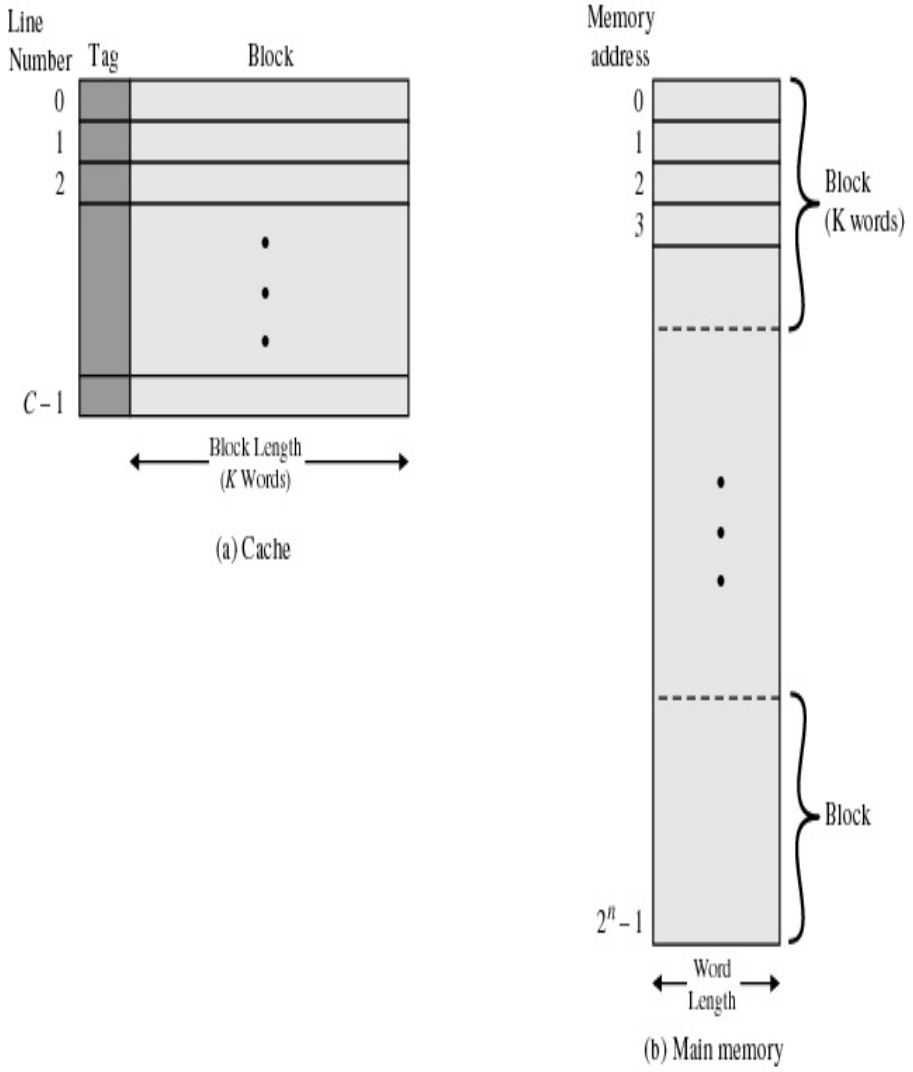


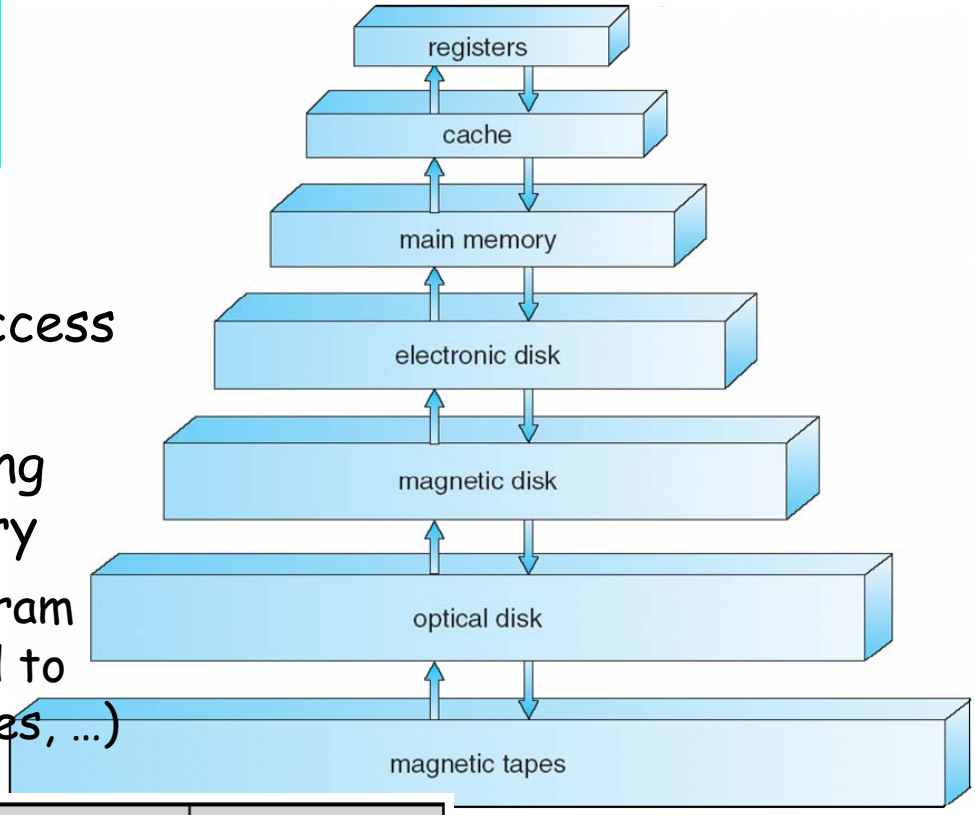
Figure 1.17 Cache/Main-Memory Structure

1. Cache size
2. Block size
3. Mapping function
   - determines which cache location the block will occupy
4. Replacement algorithm
   - determines which block to replace (e.g. Least-Recently-Used (LRU) algorithm)
5. Write policy
   - Can occur every time block is updated
   - Can occur only when block is replaced
     - Minimizes memory operations
     - Leaves memory in an obsolete state

Memory Hierarchy

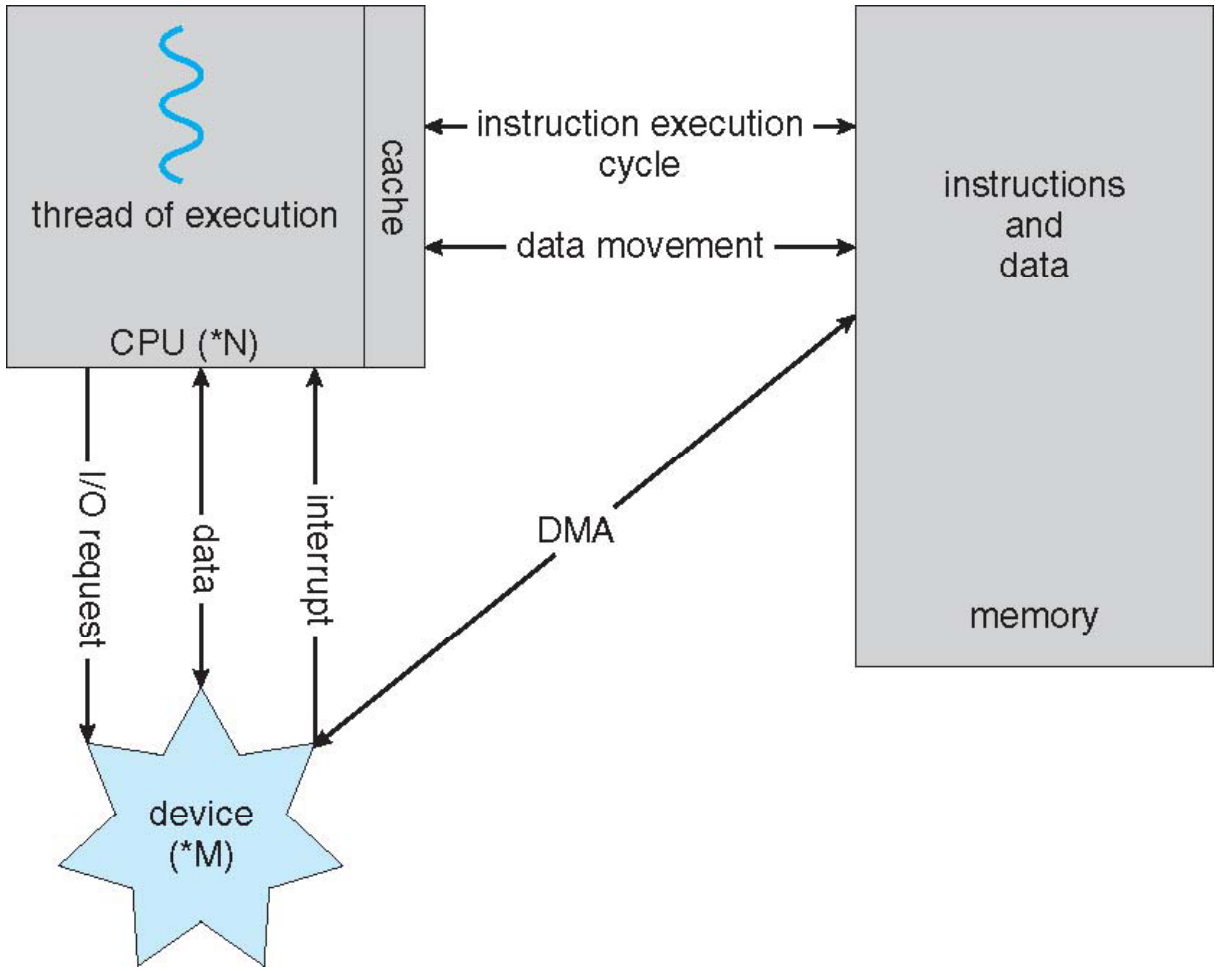
Going Down the Hierarchy

- Increasing capacity, Increasing access time
- Decreasing cost per bit, Decreasing frequency of access of the memory
  - locality of reference: during program execution memory addresses tend to cluster (iteration loops, subroutines, …)

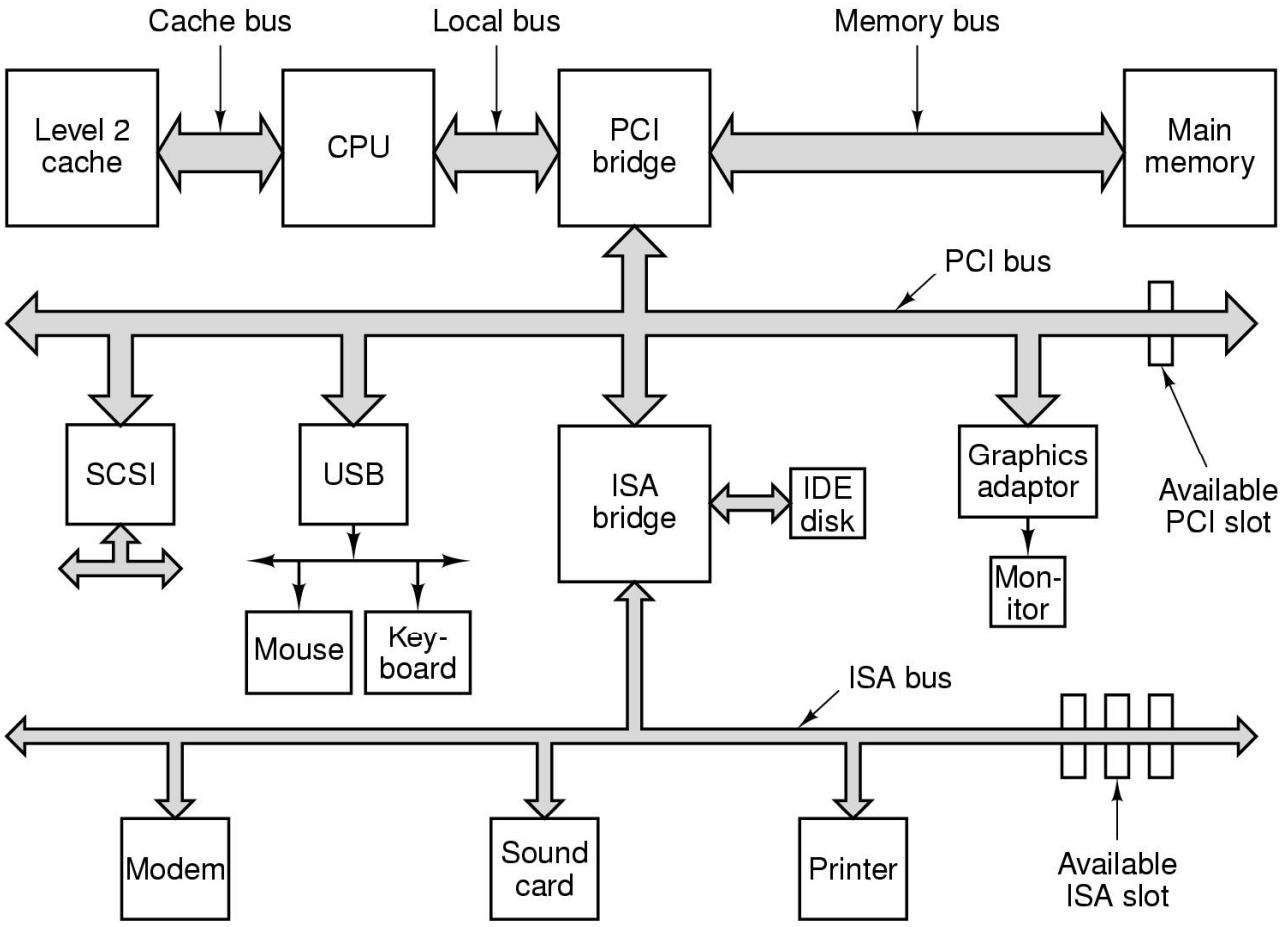


Level	1	2	3	4
Name	registers	cache	main memory	disk storage
Typical size	< 1 KB	> 16 MB	> 16 GB	> 100 GB
Implementation technology	custom memory with multiple ports, CMOS	on-chip or off-chip CMOS SRAM	CMOS DRAM	magnetic disk
Access time (ns)	0.25 – 0.5	0.5 – 25	80 – 250	5,000.000
Bandwidth (MB/sec)	20,000 – 100,000	5000 – 10,000	1000 – 5000	20 – 150
Managed by	compiler	hardware	operating system	operating system
Backed by	cache	main memory	disk	CD or tape

How a Modern Computer Works

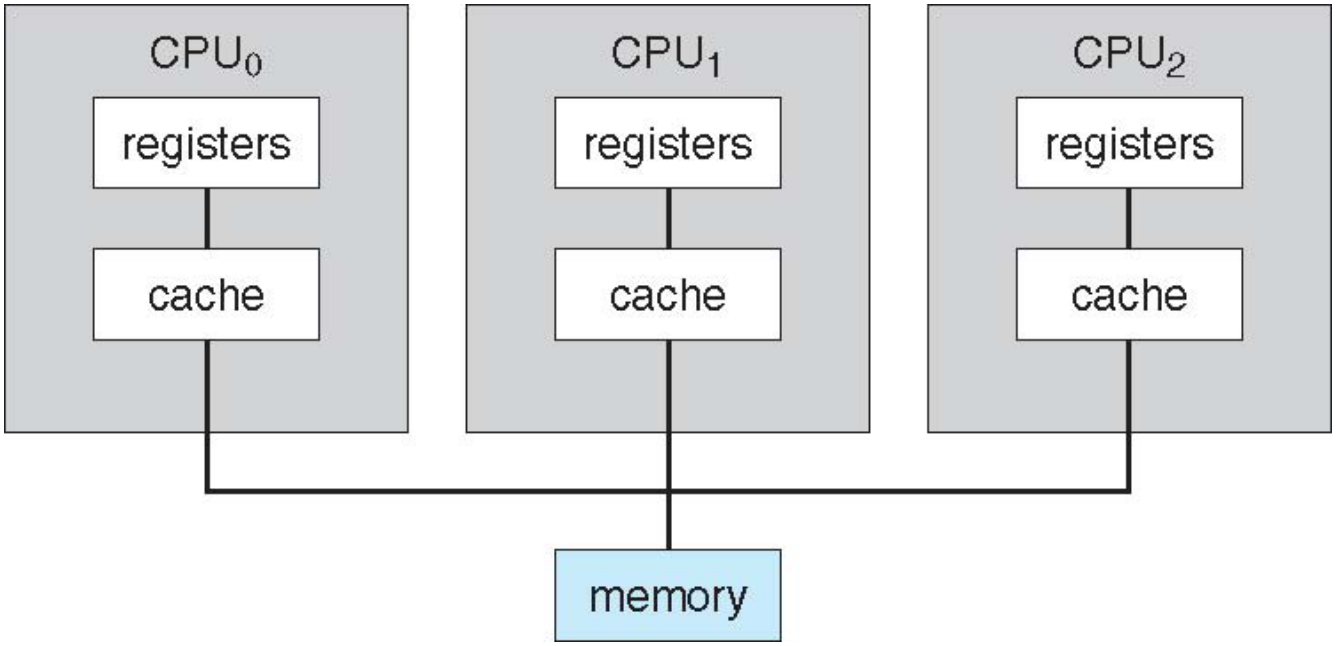


Computer Hardware Review

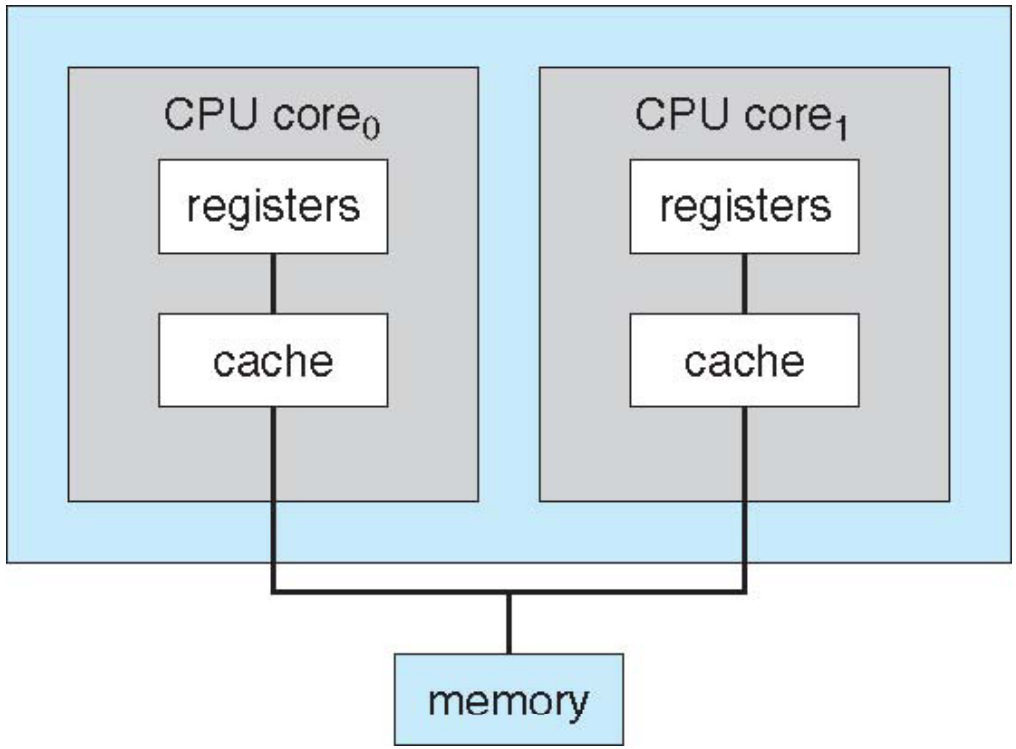


Structure of a large Pentium system (Fig, from Modern OS, A. Tanenbaum)

Symmetric Multiprocessing Architecture



A Dual-Core Design



Operating System Overview

Layers of Computer System

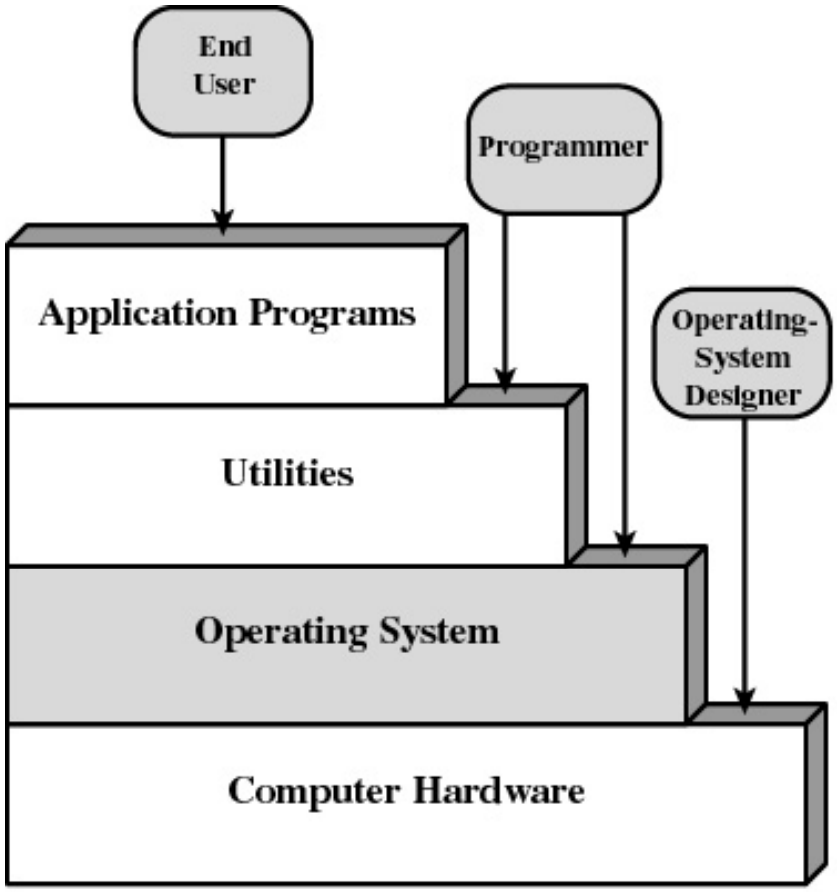


Figure 2.1 Layers and Views of a Computer System

Operating System – OS objectives

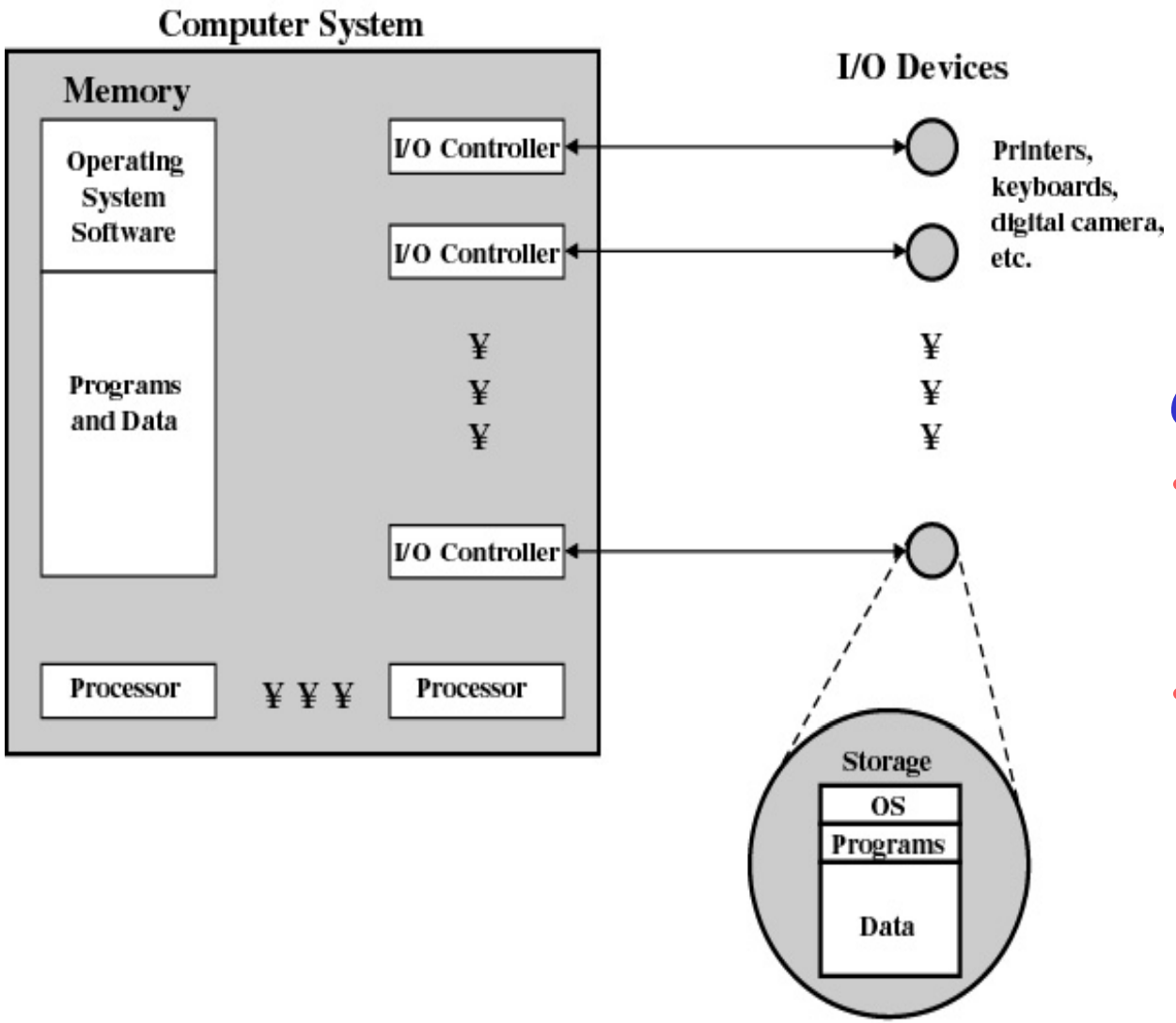
- Provides services to system users
- Shield between the user and the hardware
- Resource manager:
  - CPU(s)
  - memory and I/O devices
- A control program
  - Controls execution of programs to prevent errors and improper use of the computer
- Convenience
  - Makes the computer more convenient to use
- Efficiency
  - Allows computer system resources to be used in an efficient manner
- Ability to evolve
  - Permit introduction of new system functions without interfering with service

Services Provided by the Operating System

- Program execution:
  - CPU scheduling, resource (memory) allocation and management, synchronization
- Access to I/O devices
  - Uniform interfaces, hide details, optimise resources (disk scheduling)
- Controlled access to files
  - And structure of data
- System/resource access
  - Authorization, protection, allocation

- Utilities, e.g. for program development
  - Editors, compilers, debuggers
- Error detection and response, when, e.g.
  - hardware, software errors
  - operating system cannot grant request of application
- Monitoring, accounting

Operating System: … (roughly) it is a program …



- relinquishes control of the processor to execute other programs

OS Kernel:

- (roughly) portion of OS that is in main memory
- Contains most-frequently used functions

Basic OS structures: intro in historical order

- Hardware upgrades, new types of hardware, enabled features
- New services, new needs

Basic OS structures: intro in historical order
1. before the stone age

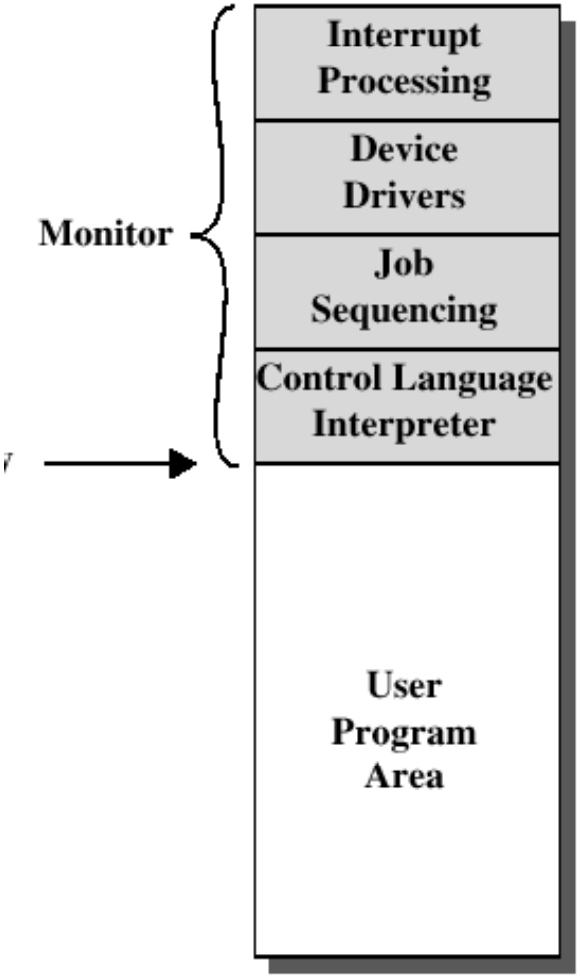
Serial Processing

- No operating system
- Machines run from a console with display lights and toggle switches, input device, and printer
- Schedule tome
- Setup included
  - loading the compiler, source program,
  - saving compiled program
  - loading
  - linking

Basic OS structures: intro in historical order
2. first "tools" appear

Simple Batch Systems:

- Monitors
  - Software that controls the running programs
  - Batch jobs together
  - Program branches back to monitor when finished
  - Resident monitor is in main memory and available for execution
- Job Control Language (JCL)
  - Provides instruction to the monitor
    - what compiler to use
    - what data to use



ure 2.3 Memory Layout for a Resident Moni

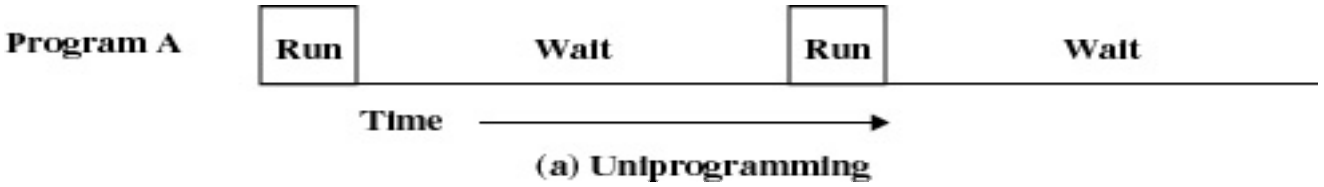
H/W features which made the first tools possible:

- Memory protection
  - do not allow the memory area containing the monitor to be altered
- Priviledged instructions
  - Only for monitor, e.g. for interface with I/O devices
- Interrupts
  - Mechanisms for the OS to relinquish control and regain it
- Timer
  - prevents a job from monopolizing the system

Basic OS structures: intro in historical order
2. Uni/multi-programming

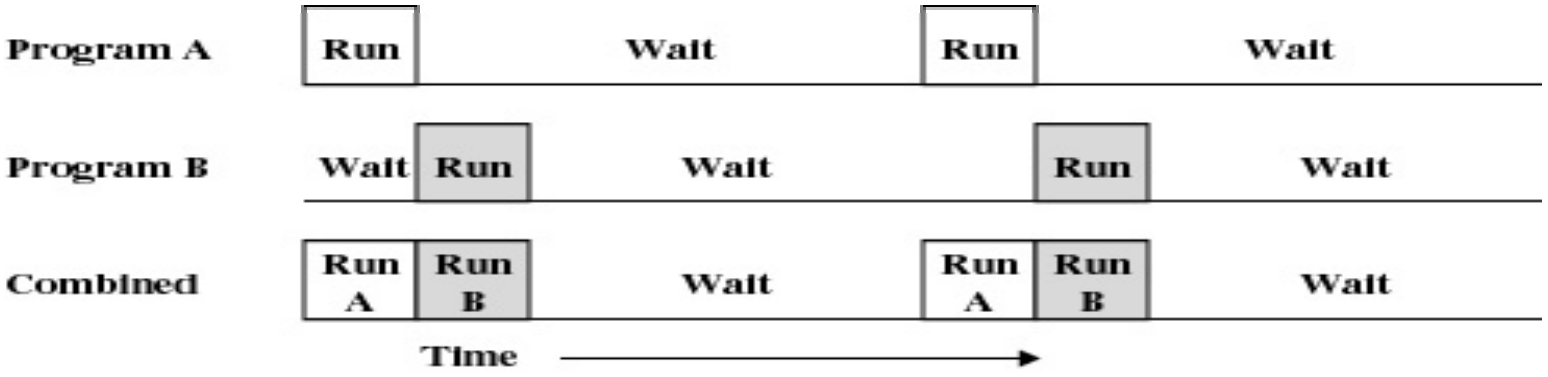
from uniprogramming….

Processor must wait for I/O instruction to complete before proceeding

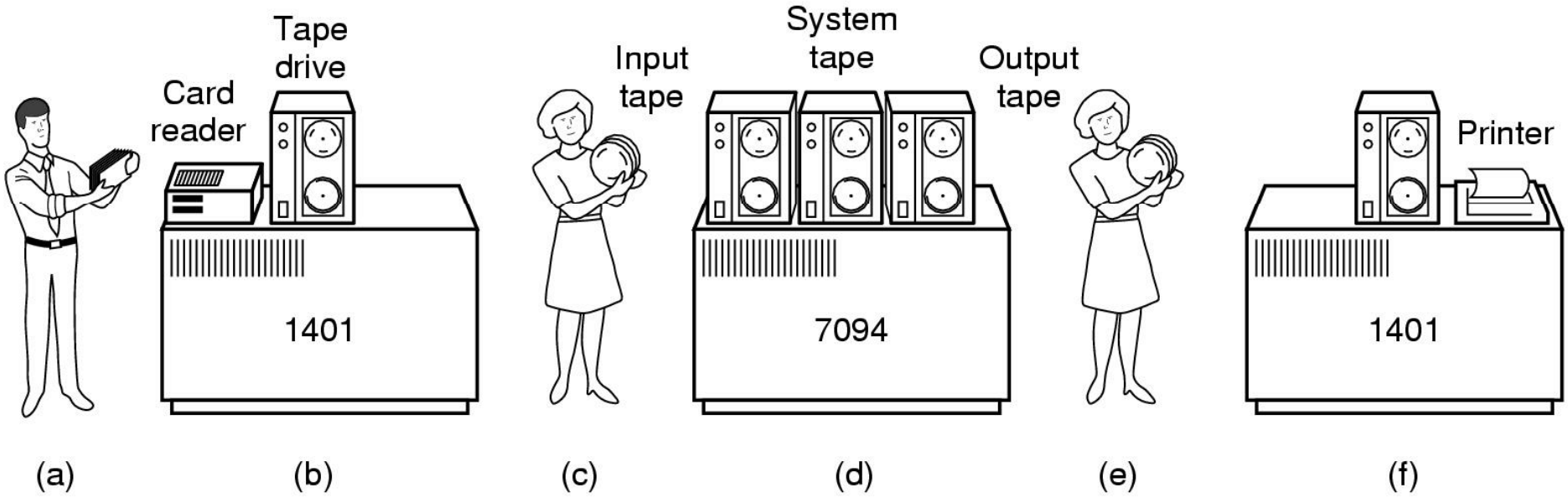


… to Multiprogramming

When one job needs to wait for I/O, the processor can switch to the other job



Early batch system



- bring cards to 1401
- read cards to tape
- put tape on 7094 which does computing
- put tape on 1401 which prints output

Basic OS structures: intro in historical order
2,5: Multiprogramming, Time Sharing

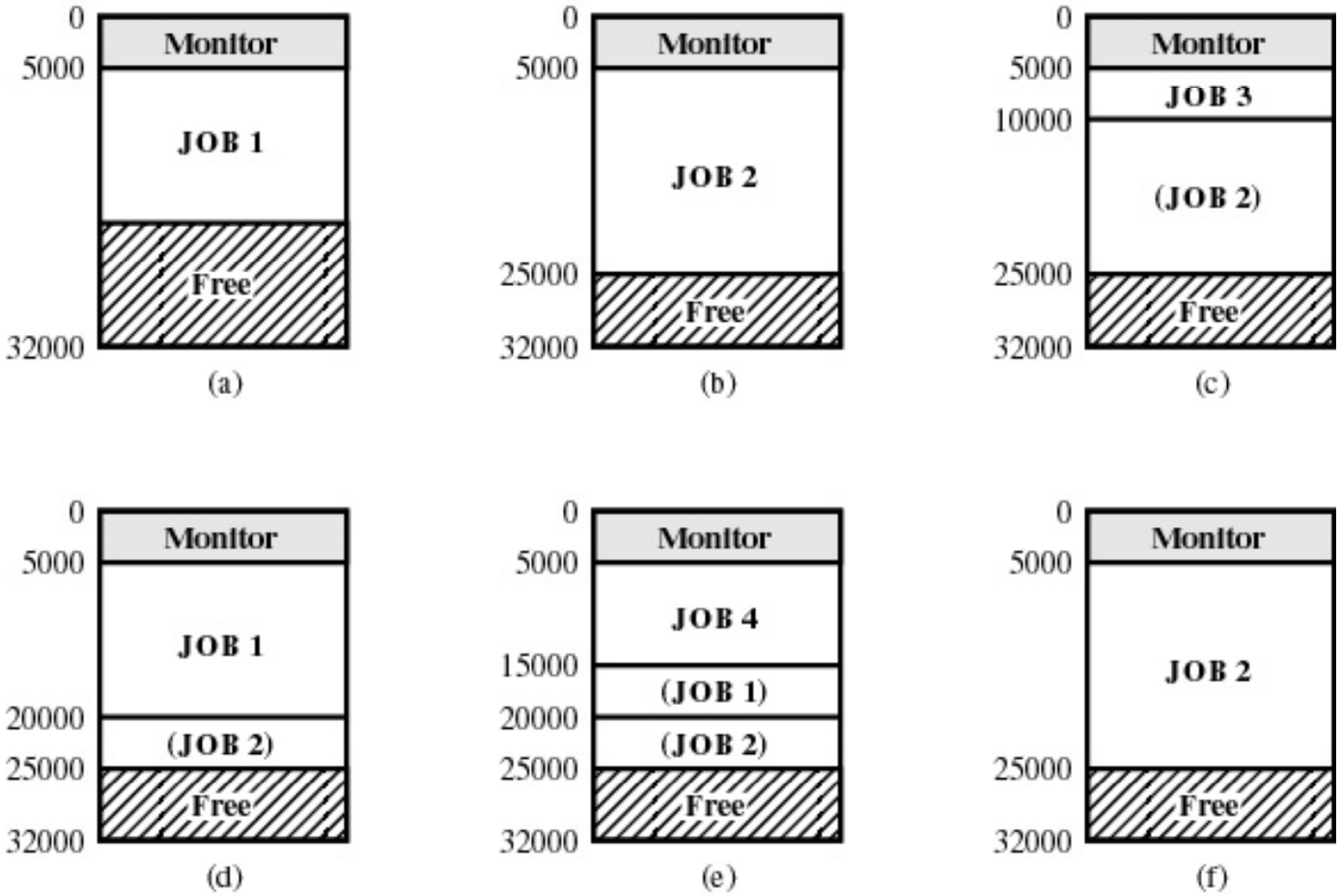
Time sharing systems use multiprogramming to handle multiple interactive jobs

- Processor's time is shared among multiple users
- Multiple users simultaneously access the system through terminals

	Batch Multiprogramming	**Time Sharing**
Principal objective	Maximize processor use	Minimize response time
Source of directives to operating system	Job control language commands provided with the job	Commands entered at the terminal

Basic OS structures: intro in historical order (2 & 2,5) multiprogramming needs ...

- ... memory management!



Summary: evolution

- First generation 1945 - 1955
  - vacuum tubes, plug boards
- Second generation 1955 - 1965
  - transistors, batch systems
- Third generation 1965 – 1980
  - ICs and multiprogramming
- Fourth generation 1980 – present
  - personal computers
- More contemporary present:
  - Personal computers become parallel, portable/embedded OS's

Basic OS structures: intro from service point of view
Zooming in a few key services

The main job of OS is to:

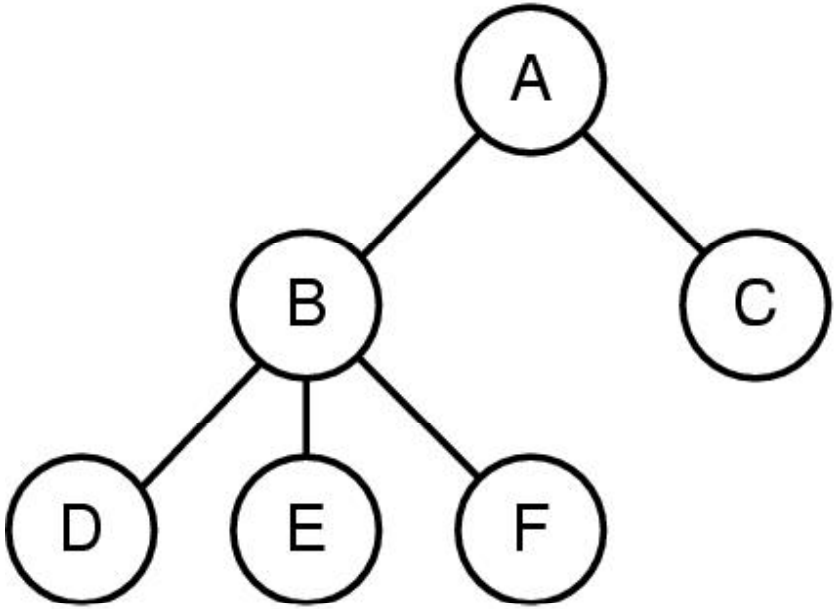
run processes! ...

Process:the concept

Process = a program in execution

- Example processes:
  - OS kernel
  - OS shell
  - Program executing after compilation
  - www-browser
- What do processes do? What do they need?

Processes create other processes



- A process tree
  - A (e.g. shell) created two child processes, B (e.g. browser) and C (print)
  - B created three child processes, D, E, and F (various browser services/windows)

Processes need… to get memory

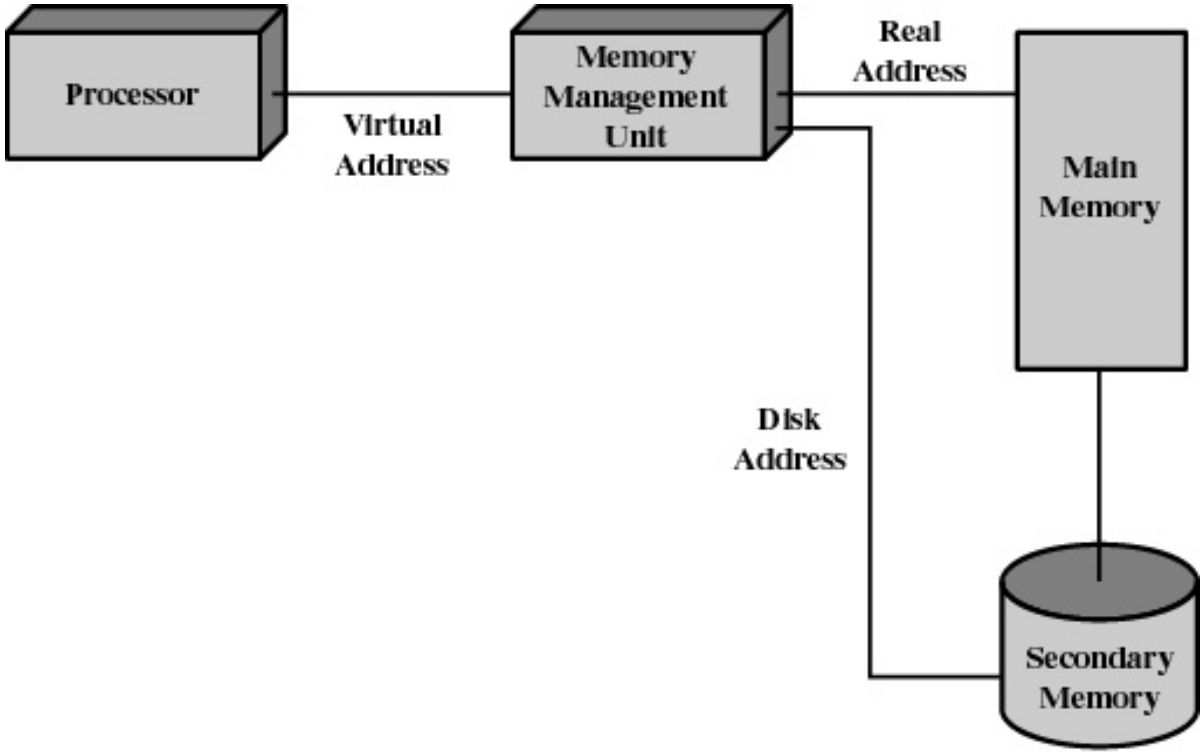
Memory Management

Important issues:

- Process isolation
  - Prevent interference between different processes
  - Protection and access control when sharing memory
- Allocation
  - Create, destroy modules dynamically
  - Support for modular programming
  - Efficiency, good utilization

Virtual Memory

- Allows programmers to address memory from a logical point of view (without worrying about the physical availability/location)
- Transfer memory ↔disk: transparent to the processes

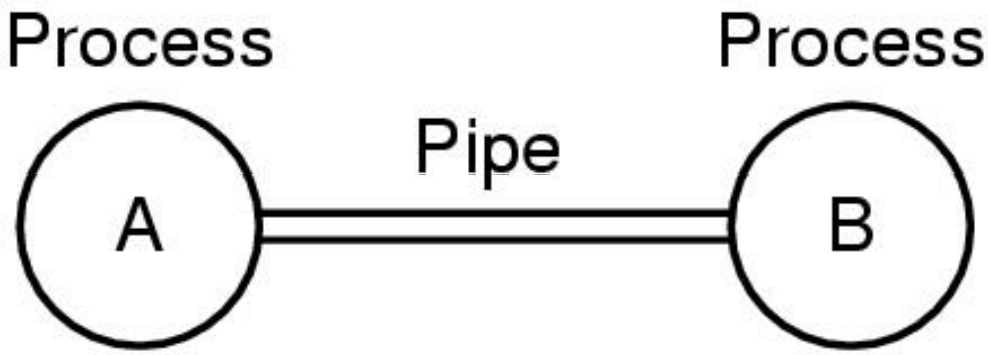


(processes also need) …, to get CPU time and other resources, …

Goals when allocating resources to processes:

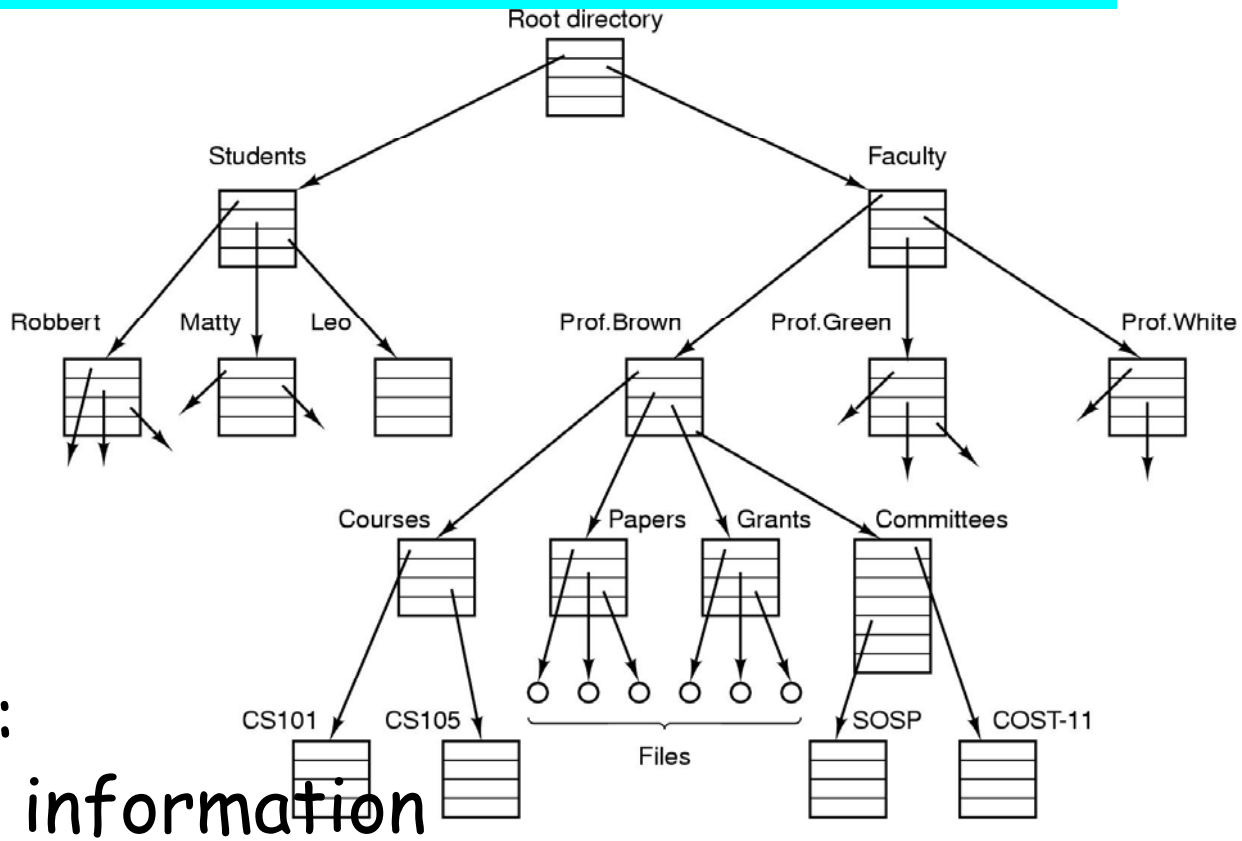
- Fairness and Differential responsiveness
  - give fair access to all processes
  - Allocate to different classes of jobs accordingly
- Efficiency
  - maximize throughput, minimize response time, and accommodate as many users as possible

... synchronization, communication...



More: mutual exclusion, producer-consumer, signal upon dependent tasks, dealing with/preventing/avoiding deadlocks ...

..., to do IO, access files, ...



Important issues:

- Organization of information
- Efficient access
- Memory management of IO
- drivers, interfaces

Protection and Security

- **Protection** – controlling access of processes or users to resources defined by the OS
- **Security** – defense of the system against internal and external attacks
  - Huge range, including denial-of-service, worms, viruses, identity theft, theft of service
- Systems generally first distinguish among users, to determine who can do what
  - User identities (**user IDs**, security IDs) include name and associated number, one per user
  - User ID then associated with all files, processes of that user to determine access control
  - Group identifier (**group ID**) allows set of users to be defined and controls managed, then also associated with each process, file
  - **Privilege escalation** allows user to change to effective ID with more rights

Process: Implementation

Consists of three components

- An executable program
- Associated data needed by the program
- Execution context of the program
  - All the book-keeping information the system needs to manage the process

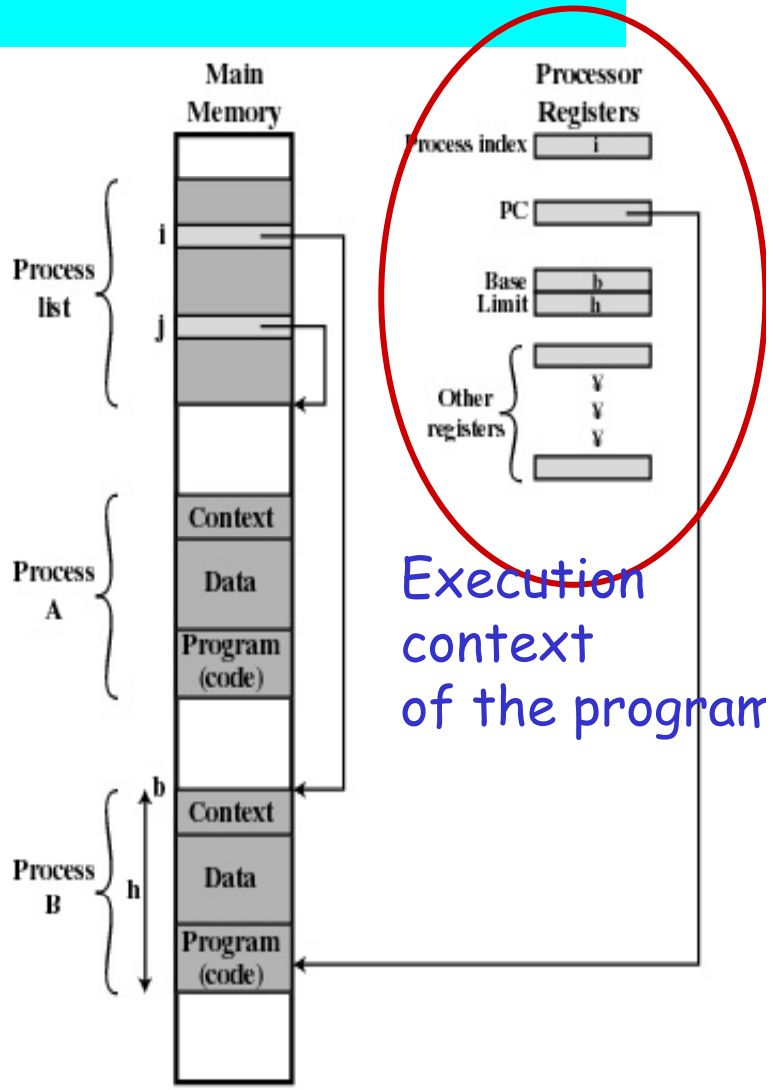
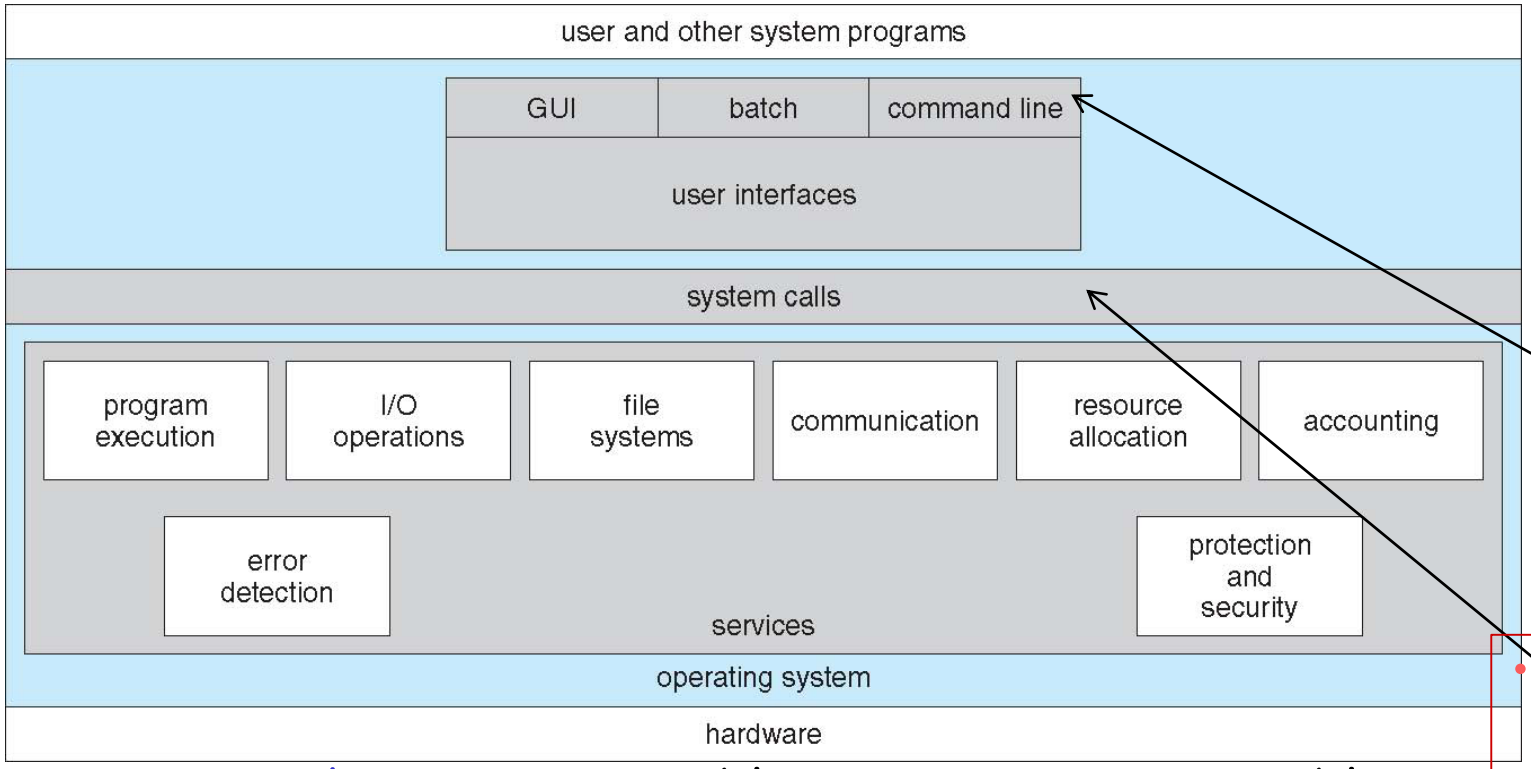


Figure 2.8 Typical Process Implement

Major Elements of an Operating System



OS = very large piece of software!

- User interface

- User or system programs make direct use of the OS via system calls

Components: decompose a problem into more manageable subproblems (process manager, file manager, etc)

- Bootstrap program activates OS kernel (permanent system process)
  - Shell (≠ kernel): program to let the user initiate processes

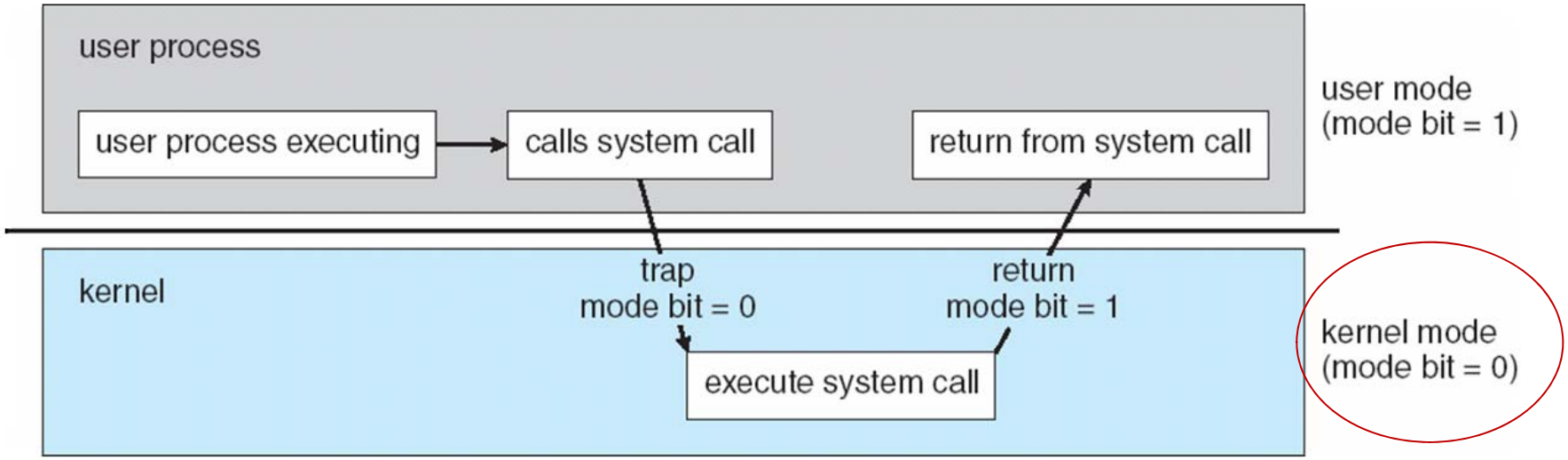
User Operating System Interface

Command Line Interface (CLI) or command interpreter

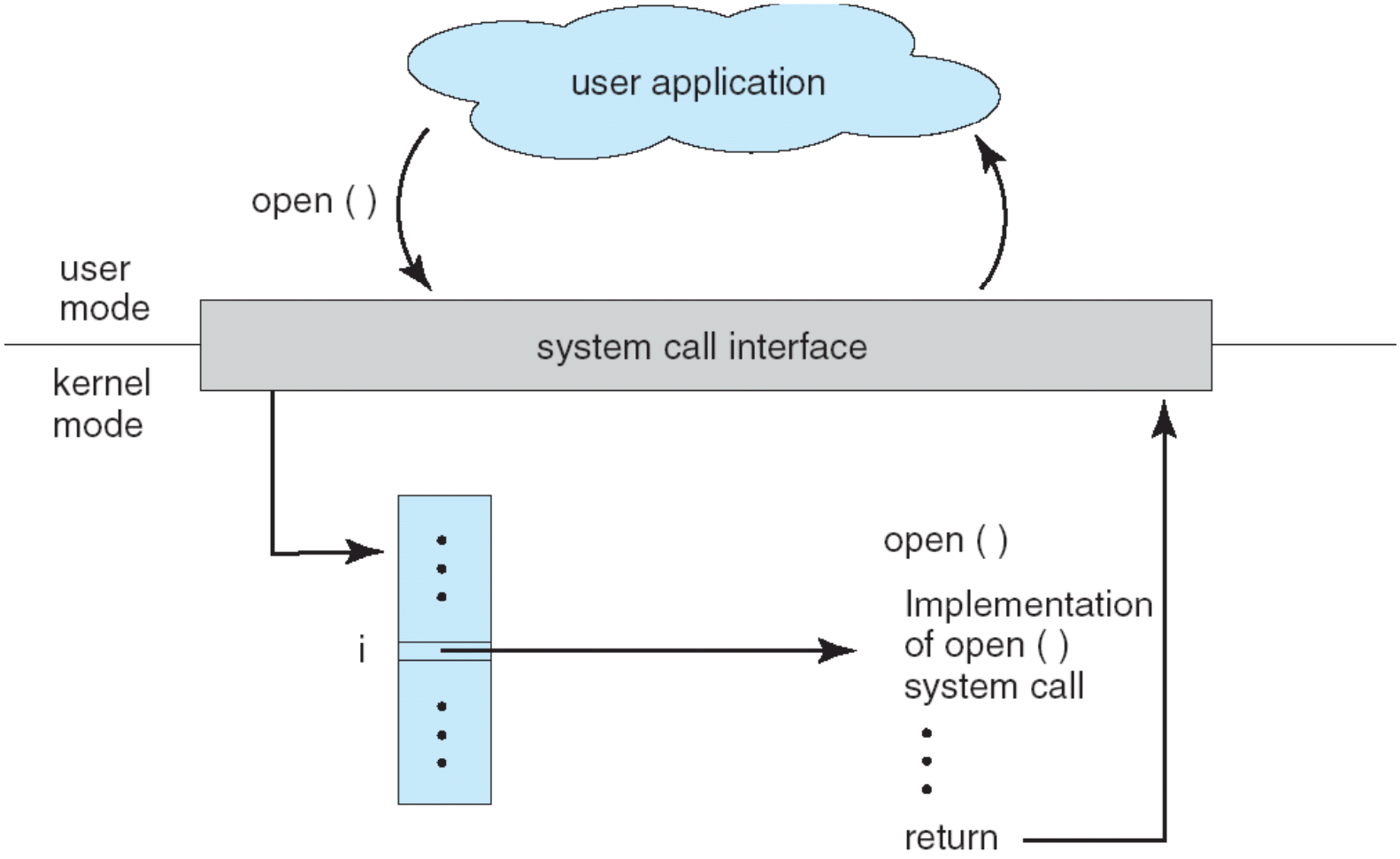
- Can be implemented in kernel, sometimes by systems program
- Sometimes multiple flavors implemented – shells
- Primarily fetches a command from user and executes it

- Graphical User Interface: User-friendly desktop metaphor interface
  - Usually mouse, keyboard, and monitor
  - Icons represent files, programs, actions, etc
  - Invented at Xerox PARC
- Many systems now include both CLI and GUI interfaces
  - Microsoft Windows is GUI with CLI "command" shell
  - Apple Mac OS X as "Aqua" GUI interface with UNIX kernel underneath and shells available
  - Solaris is CLI with optional GUI interfaces (Java Desktop, KDE)

Programmer OS interface: making a system call

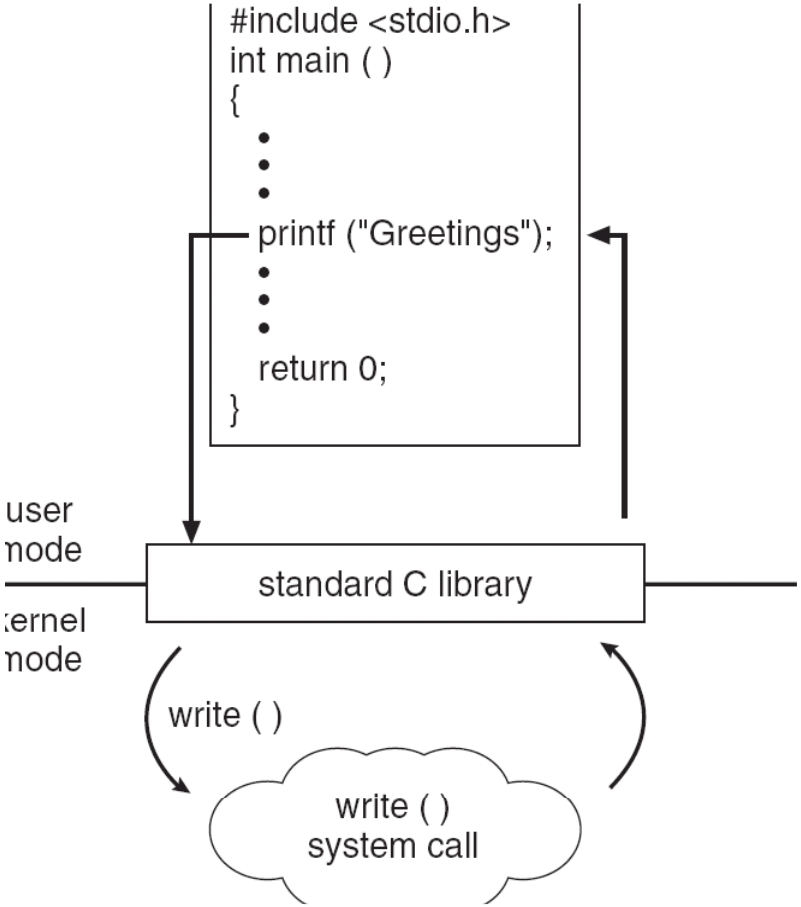


Programmer OS interface: making a system call, e.g:



Standard C Library Example

- C program invoking printf() library call, which calls write() system call



Some System Calls

Process management

Call	Description
pid = fork( )	Create a child process identical to the parent
pid = waitpid(pid, &statloc, options)	Wait for a child to terminate
s = execve(name, argv, environp)	Replace a process' core image
exit(status)	Terminate process execution and return status

File management

Call	Description
fd = open(file, how, ...)	Open a file for reading, writing or both
s = close(fd)	Close an open file
n = read(fd, buffer, nbytes)	Read data from a file into a buffer
n = write(fd, buffer, nbytes)	Write data from a buffer into a file
position = lseek(fd, offset, whence)	Move the file pointer
s = stat(name, &buf)	Get a file's status information

Miscellaneous

Call	Description
s = chdir(dirname)	Change the working directory
s = chmod(name, mode)	Change a file's protection bits
s = kill(pid, signal)	Send a signal to a process
seconds = time(&seconds)	Get the elapsed time since Jan. 1, 1970

A stripped down shell

```
while (TRUE) {                                  /* repeat forever */
    type_prompt( );                             /* display prompt */
    read_command (command, parameters)          /* input from terminal */

if (fork() != 0) {                              /* fork off child process */
    /* Parent code */
    waitpid( -1, &status, 0);                   /* wait for child to exit */
} else {
    /* Child code */
    execve (command, parameters, 0);            /* execute command */
 }
}
```

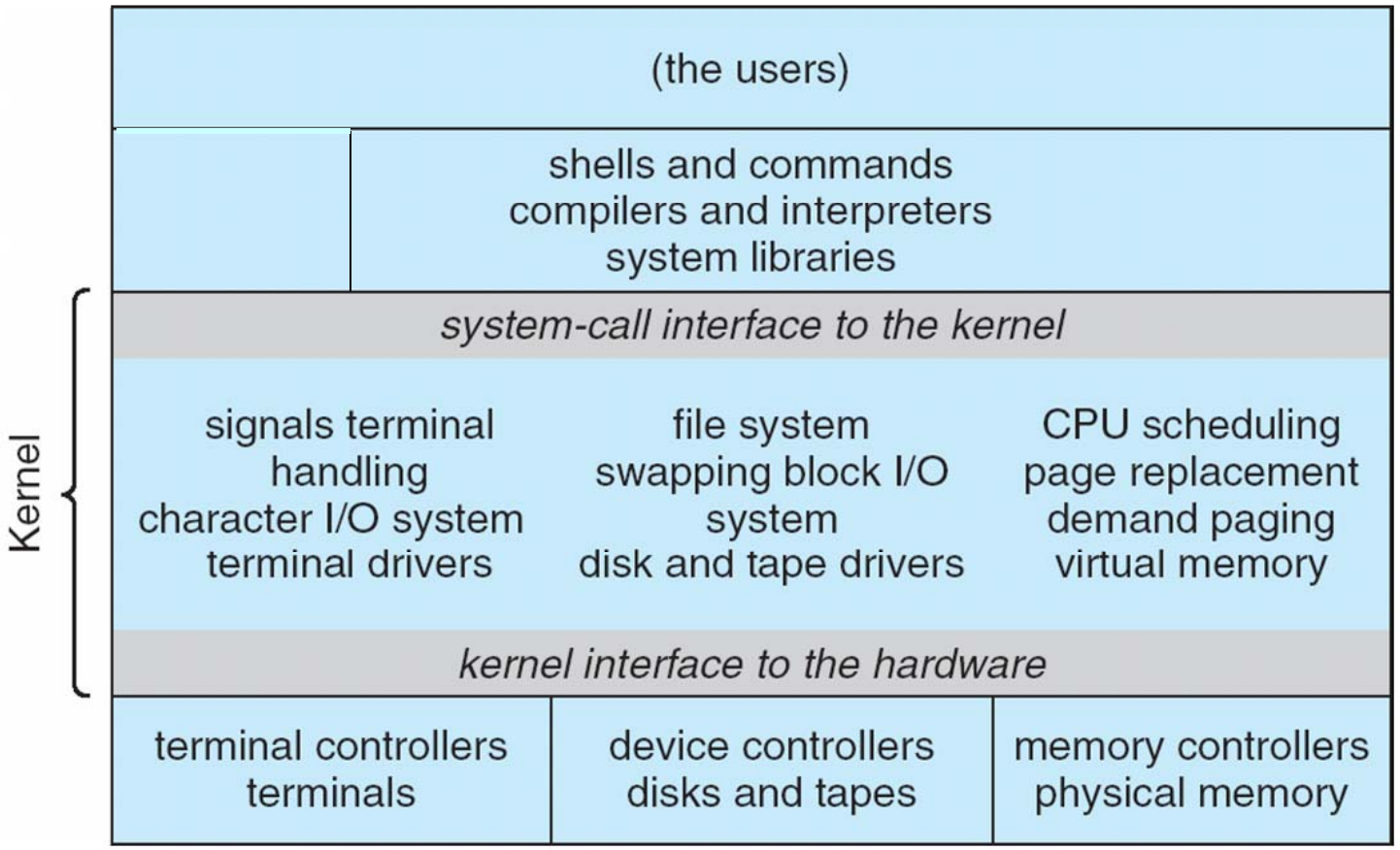
System Programs

- System programs provide a convenient environment for program development and execution. The can be divided into:
  - File manipulation
  - Status information
  - File modification
  - Programming language support
  - Program loading and execution
  - Communications
  - Application programs
- Most users' view of the operation system is defined by system programs, not the actual system calls

Operating System Design and Implementation

- Internal structure of different Operating Systems can vary widely
- Start by defining goals and specifications
- Affected by choice of hardware, type of system
- *User* goals and *System* goals
  - User goals – operating system should be convenient to use, easy to learn, reliable, safe, and fast
  - System goals – operating system should be easy to design, implement, and maintain, as well as flexible, reliable, error-free, and efficient

Traditional UNIX System Structure

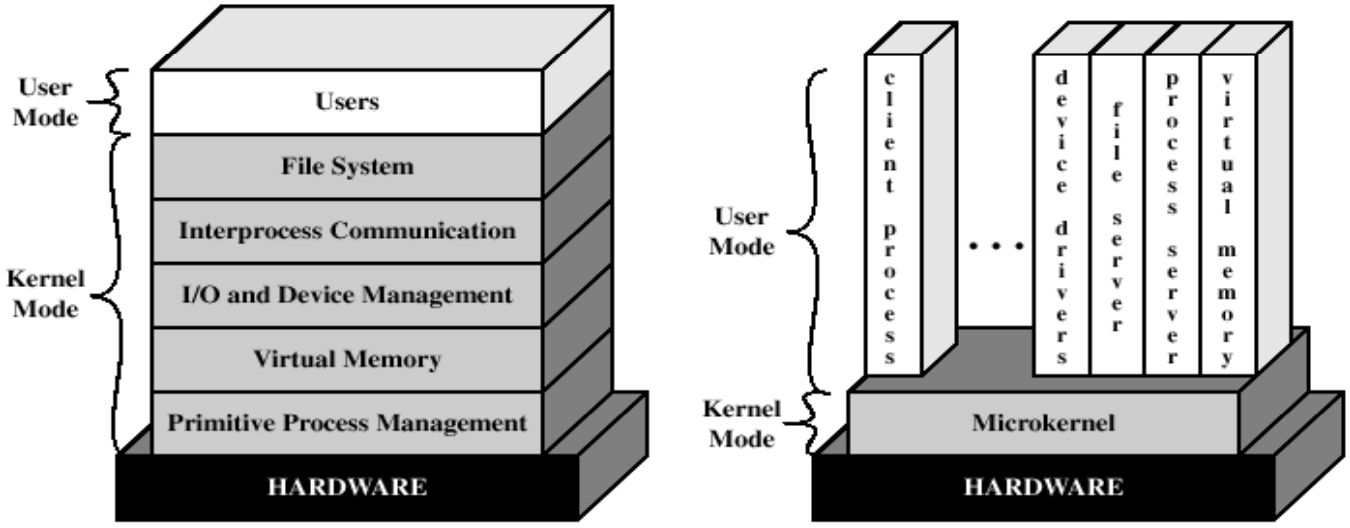


Issues in Modern Operating Systems

- Microkernel architecture
  - Only few essential functions in kernel; OO design
- Multithreading
  - A process may consist of several sequential threads of execution
- Concurrent computer systems
  - Symmetric multi-processor systems
  - Multi-threaded/multicore processors
  - Distributed systems
    - provide the illusion of a single main memory and single secondary memory space in cluster-based platforms
- Real-time Operating Systems
  - For time-critical applications, multimedia, …
- Embedded Operating Systems
  - Constraints: limited resources, special functionalities

Microkernel

- Small OS core; contains only essential OS functions:
  - Low-level memory management (address space mapping)
  - Process scheduling
  - I/O and interrupt management
- Many services traditionally included in the OS kernel are now external subsystems
  - device drivers, file systems, virtual memory manager, windowing system, security services



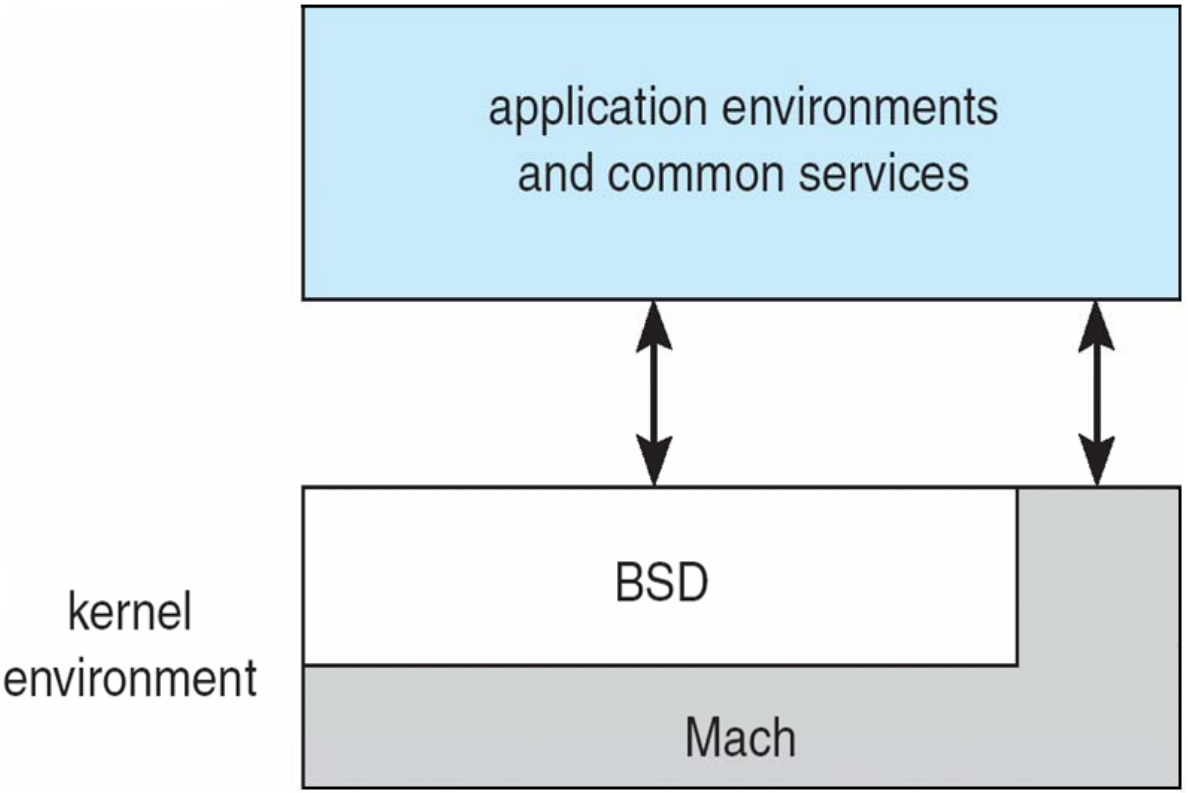
(a) Layered kernel

(b) Microkernel

Benefits of a Microkernel Organization

- Uniform interface on request made by a process
  - All services are provided by means of message passing
- Distributed system support
  - Message are sent without knowing what the target machine is
- Extensibility
  - Allows the addition/removal of services and features
- Portability
  - Changes needed to port the system to a new processor is changed in the microkernel - not in the other services
- Object-oriented operating system
  - Components are objects with clearly defined interfaces that can be interconnected
- Reliability
  - Modular design;
  - Small microkernel can be rigorously tested

Mac OS X Structure



Instantiation: Windows

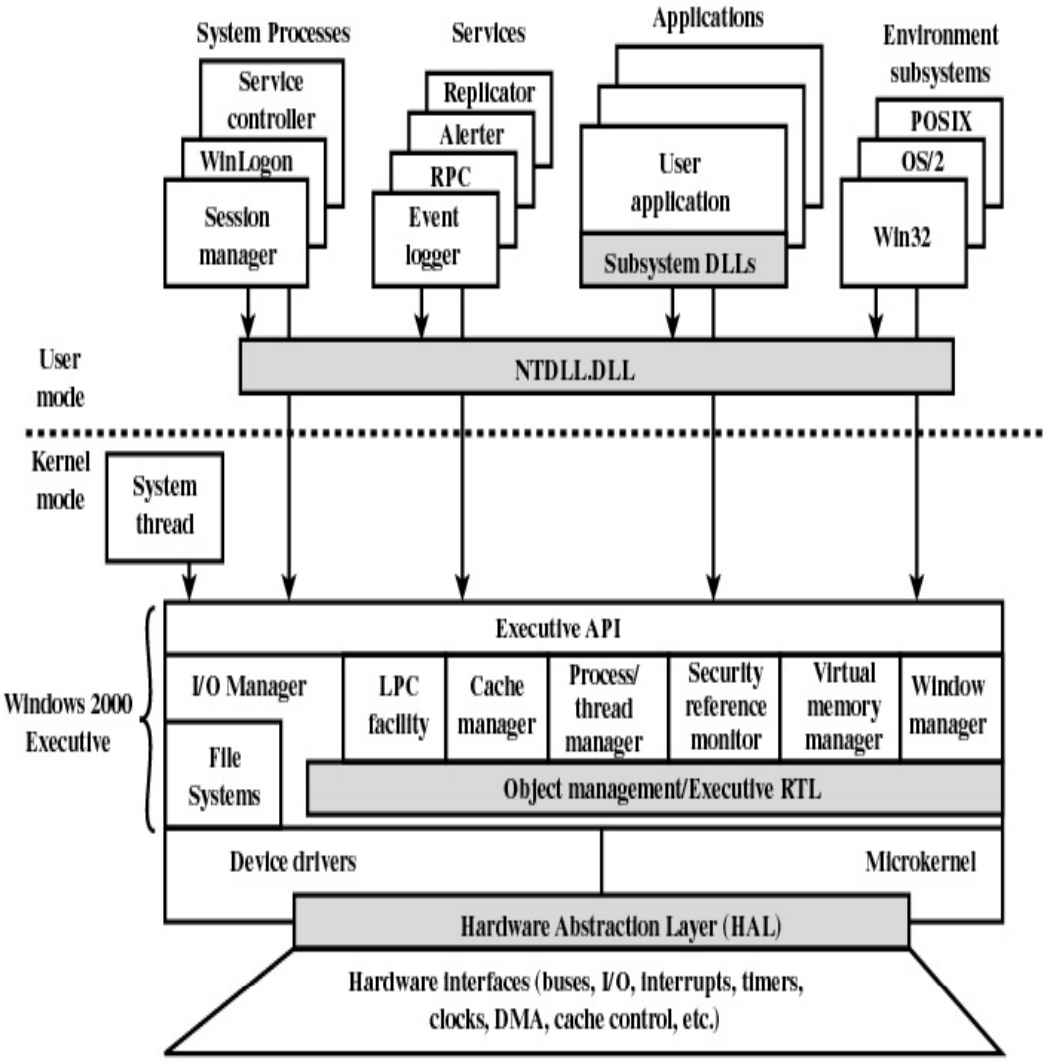
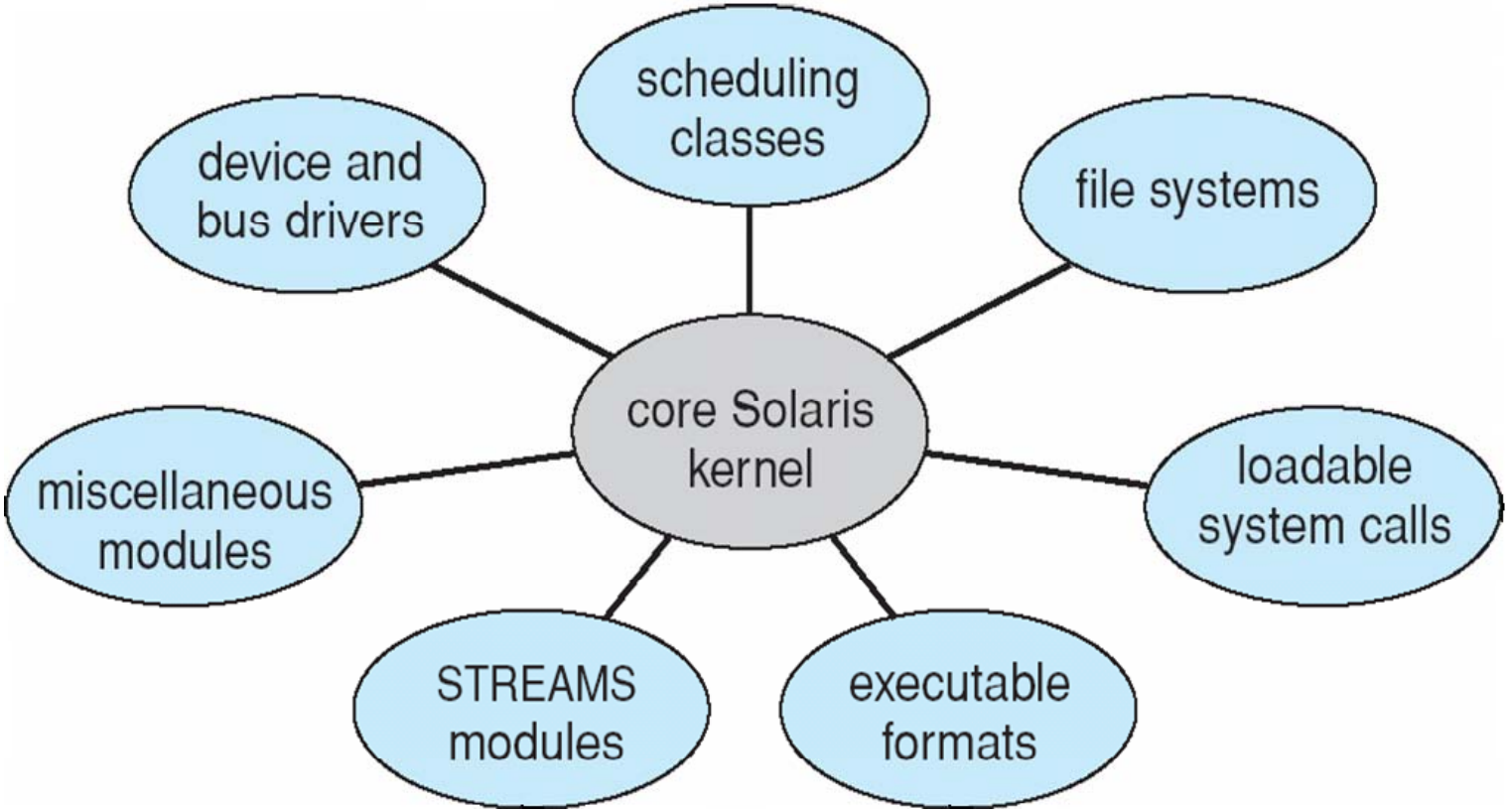


Figure 2.13 Windows 2000 Architecture

- Client/Server computing; base for distributed computing
- Modified microkernel architecture
  - not a pure microkernel: many system functions outside of the microkernel run in kernel mode
  - modules can be removed, upgraded, or replaced without rewriting the entire system

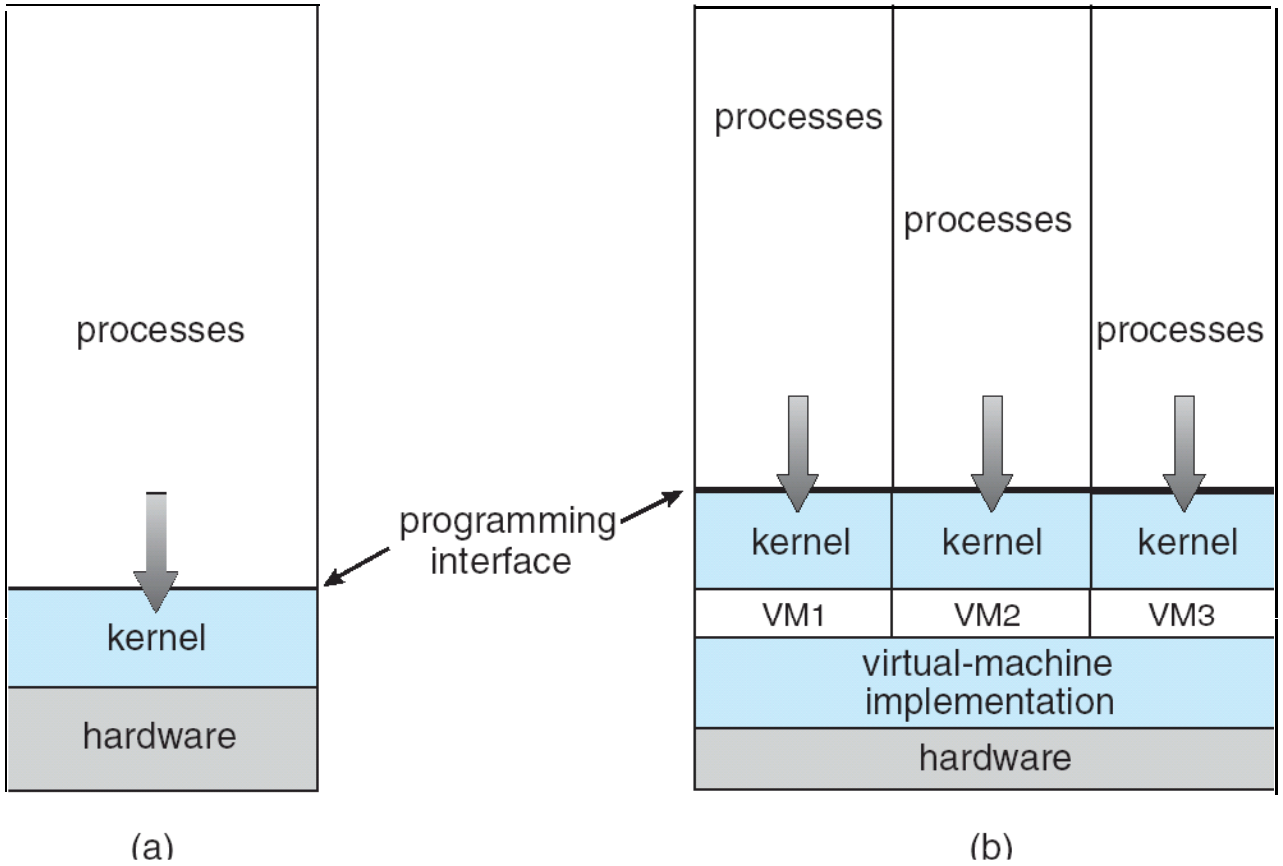
Solaris Modular Approach



Virtual Machine

- treats hardware and the operating system kernel as though they were all hardware
- provides an interface *identical* to the underlying bare hardware
- The operating system host creates the illusion that a process has its own processor and (virtual memory)
- Each guest provided with a (virtual) copy of underlying computer

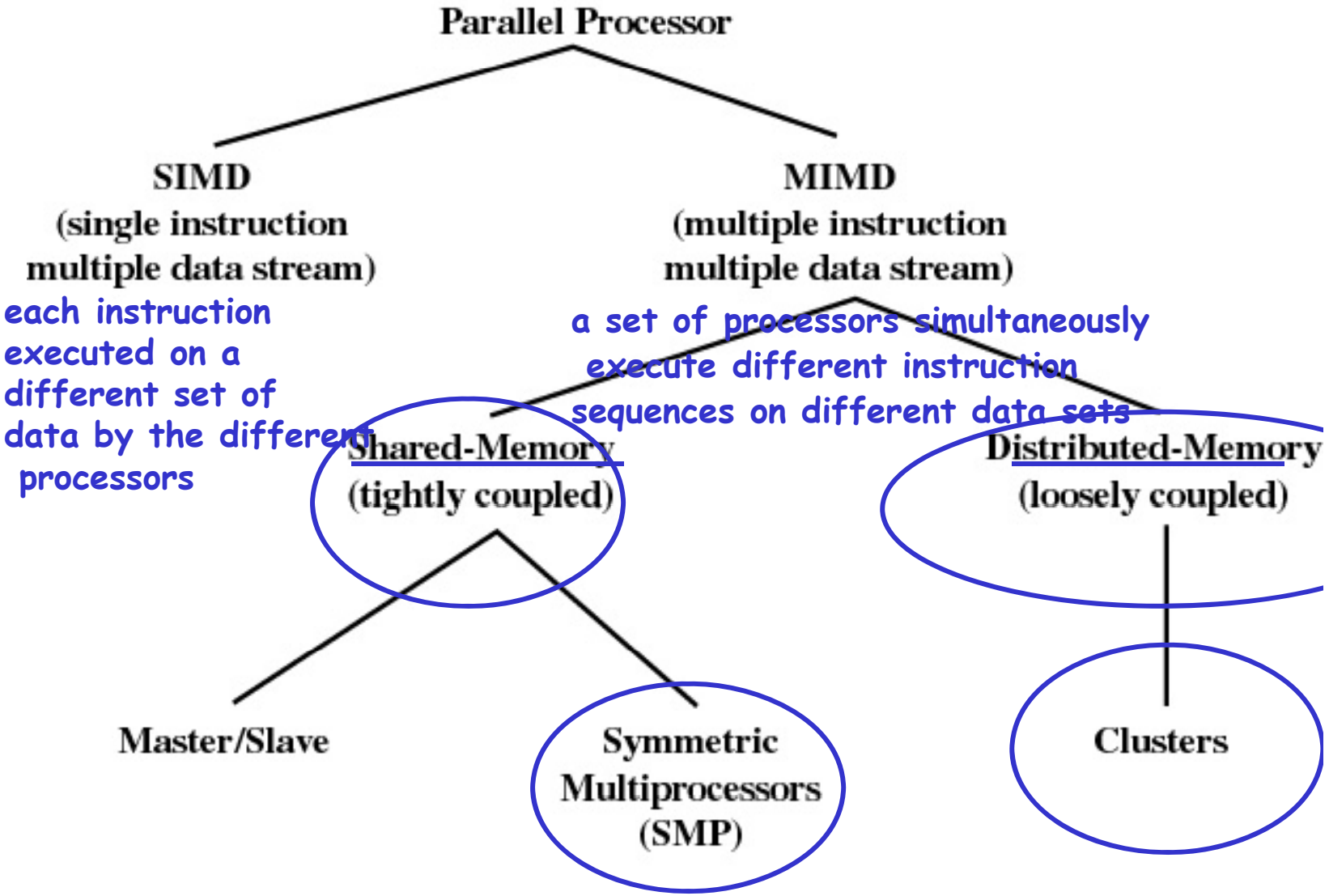
Virtual Machines (Cont)



Virtual Machines History and Benefits

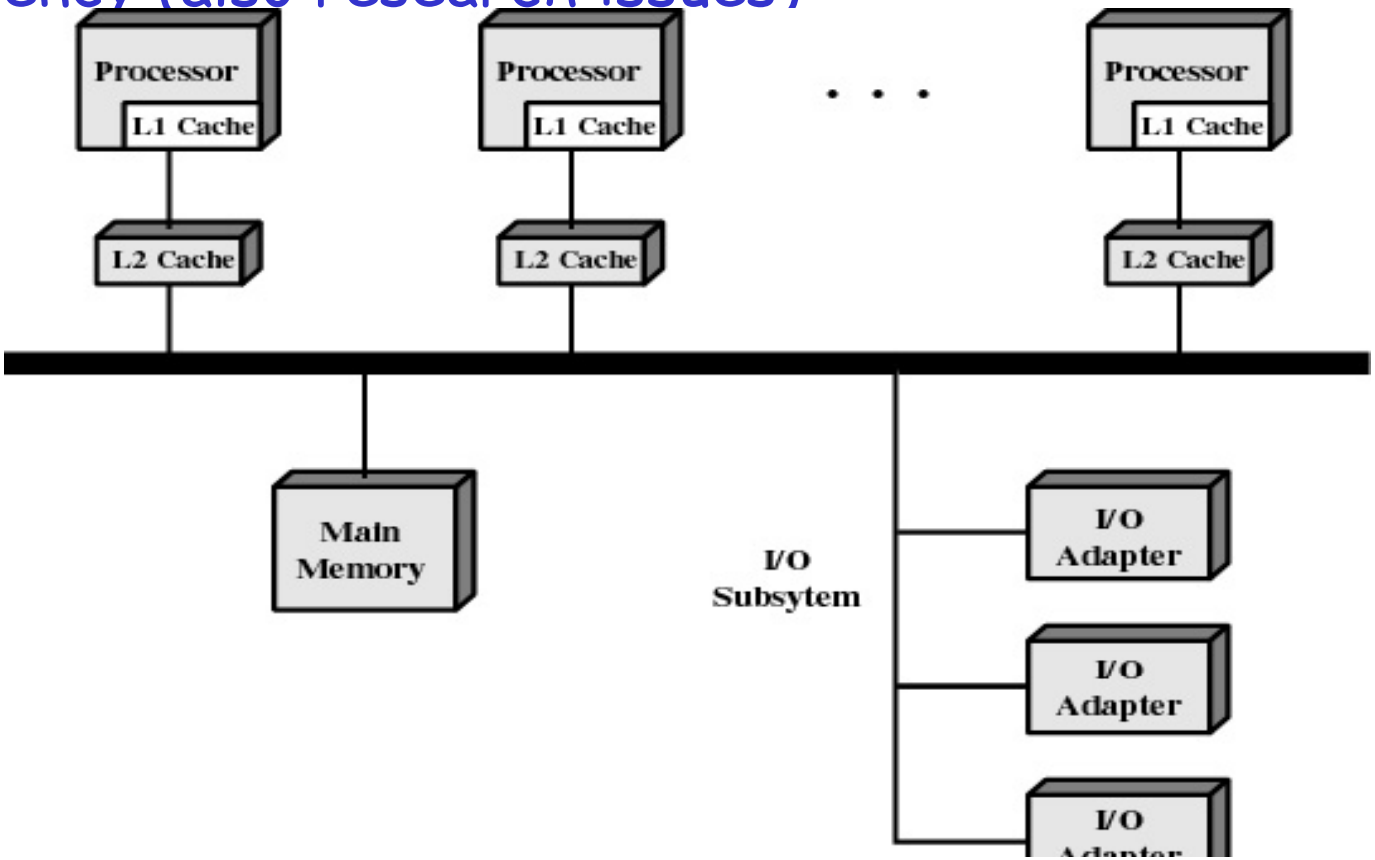
- commercially in IBM mainframes ,1972
- multiple execution environments (different OSs) share the same hardware, protect from each other
- Some file sharing permitted, controlled
- Commutate with each other + other physical systems via networking
- Useful for development, testing
- "Open Virtual Machine Format", standard format of virtual machines, allows a VM to run within many different virtual machine (host) platforms

Concurrent Computer Systems



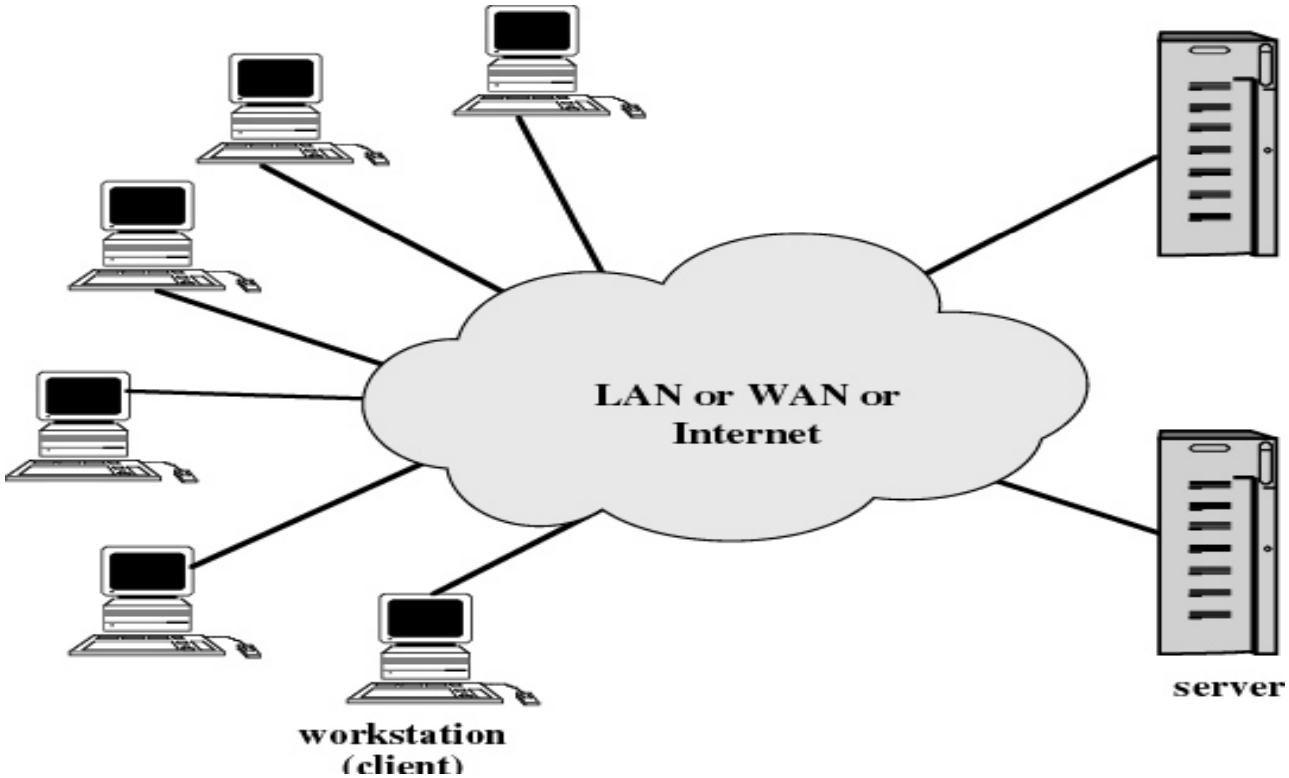
Symmetric Multiprocessors and multicores

- Processors share the same memory and I/Os
- Kernel can execute on any processor
- Scheduling & synchronization, memory management & consistency (also research issues)



Cluster Computer Platforms

- Network
- Middleware layer (part of OS) to provide
  - single-system image (synchronization, consistency, global states, file systems)
  - fault-tolerance, load balancing, parallelism



Open-Source Operating Systems

- Operating systems made available in source-code format rather than just binary closed-source
- Counter to the copy protection and Digital Rights Management (DRM) movement
- Started by Free Software Foundation (FSF), which has "copyleft" GNU Public License (GPL)
- Examples include GNU/Linux, BSD UNIX (including core of Mac OS X), and Sun Solaris

Summary

- OS: intermediary between user and hardware
  - Execute programs in convenient + efficient manner
  - Software that manages/interacts with the hardware
- Organization?
  - Define goals, find methods/strategies to reach them
  - Work piece by piece

- We saw "trailers" of the movies, we have context.
- Next: piece by piece focus