Operating Systems: Internals and Design Principles

Chapter 2
Operating System Overview

Seventh Edition

By William Stallings

Operating System

- An interface between applications and computer
- A program that controls the execution of application programs and the allocation of system resources

Main objectives of an OS:

- Convenience
- Efficiency
- Ability to evolve

Computer Hardware and Software Infrastructure

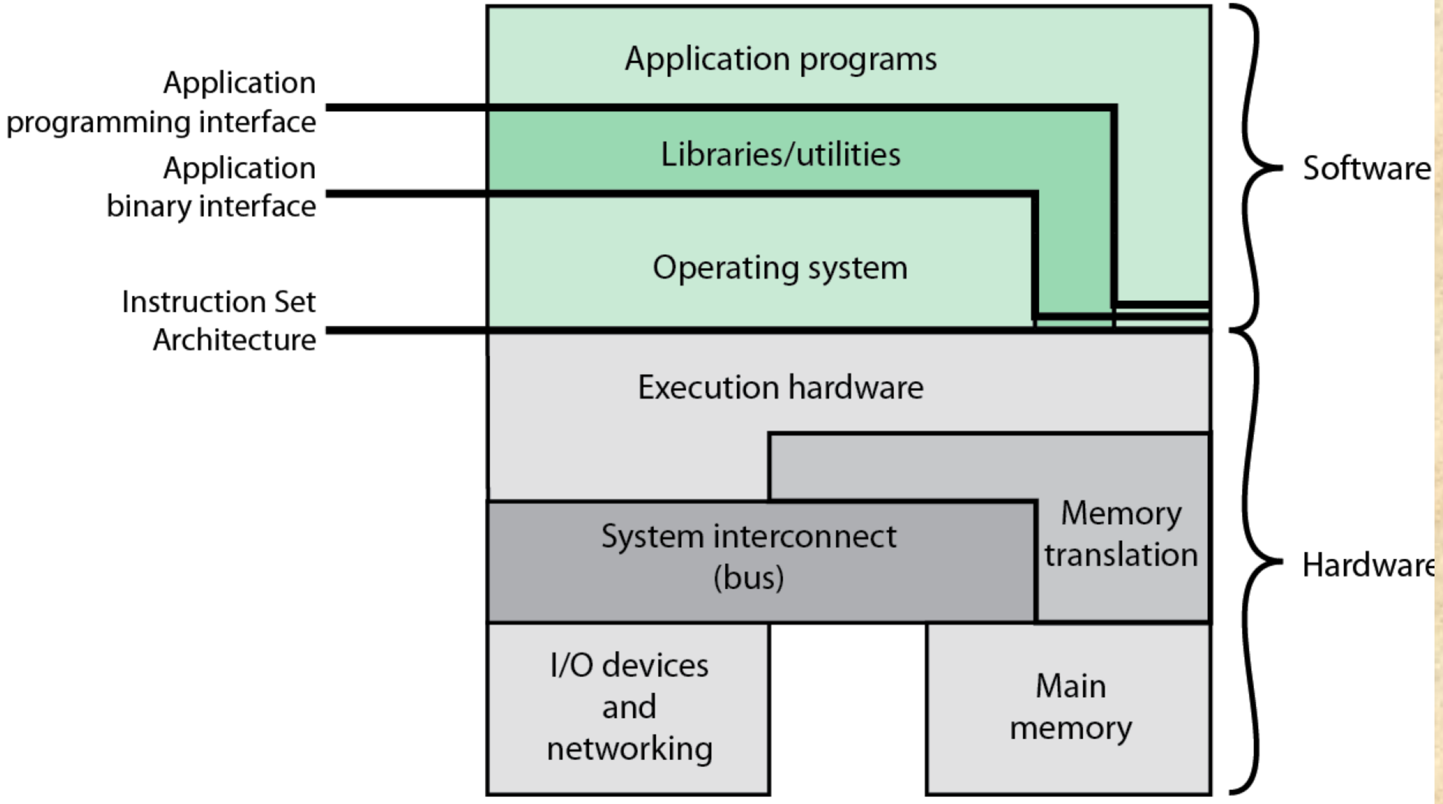


Figure 2.1 Computer Hardware and Software Infrastructure

Efficiency:The Operating System As a Resource Manager

- A computer is a set of resources for moving, storing, & processing data
- The OS is responsible for managing these resources
- The OS exercises its control through software

Operating System as Software

- Functions in the same way as ordinary computer software
- Program, or suite of programs, executed by the processor
- Frequently relinquishes control and must be able to regain control to decide on the next thing the processor should do.

Operating System as Resource Manager

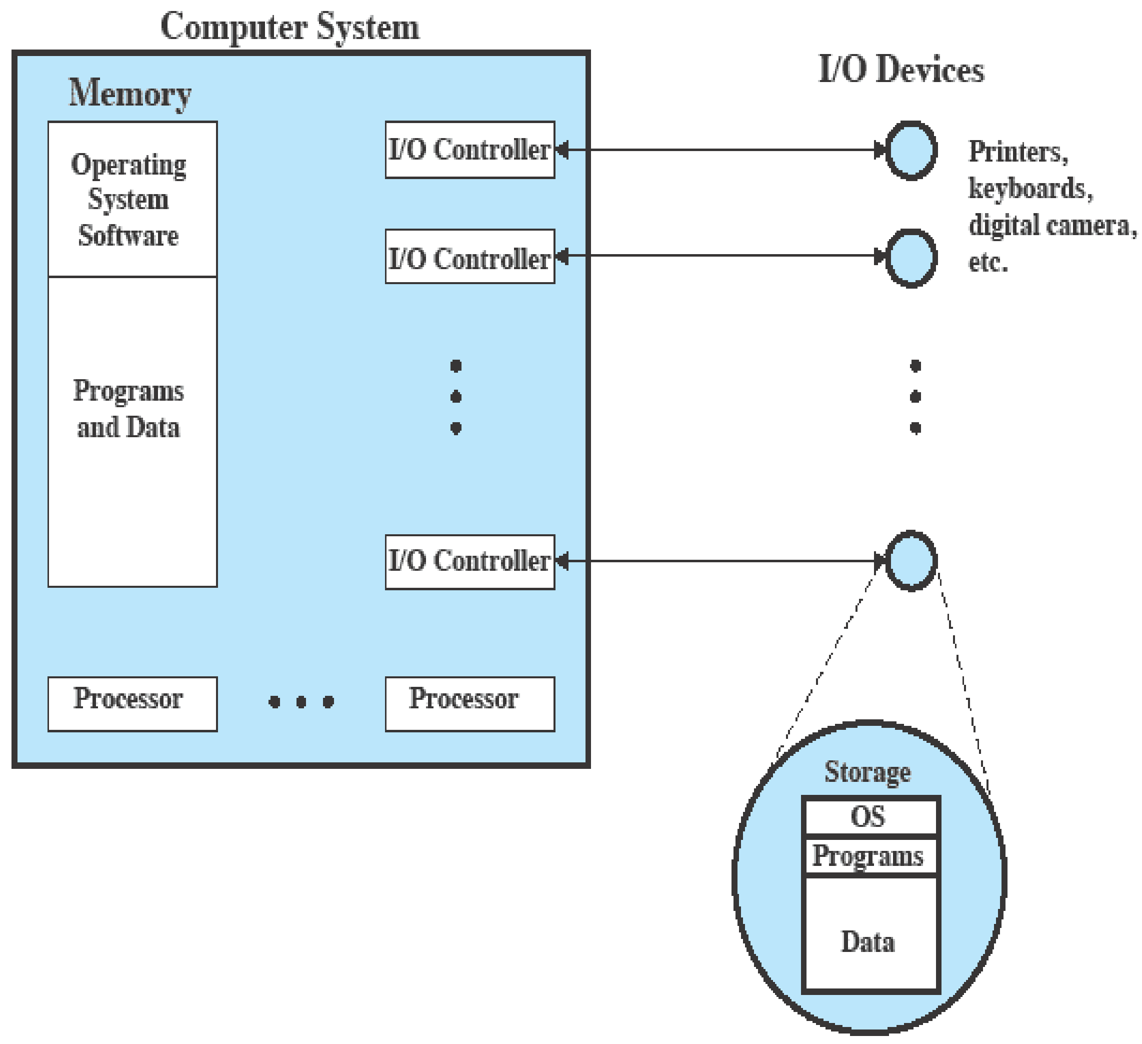
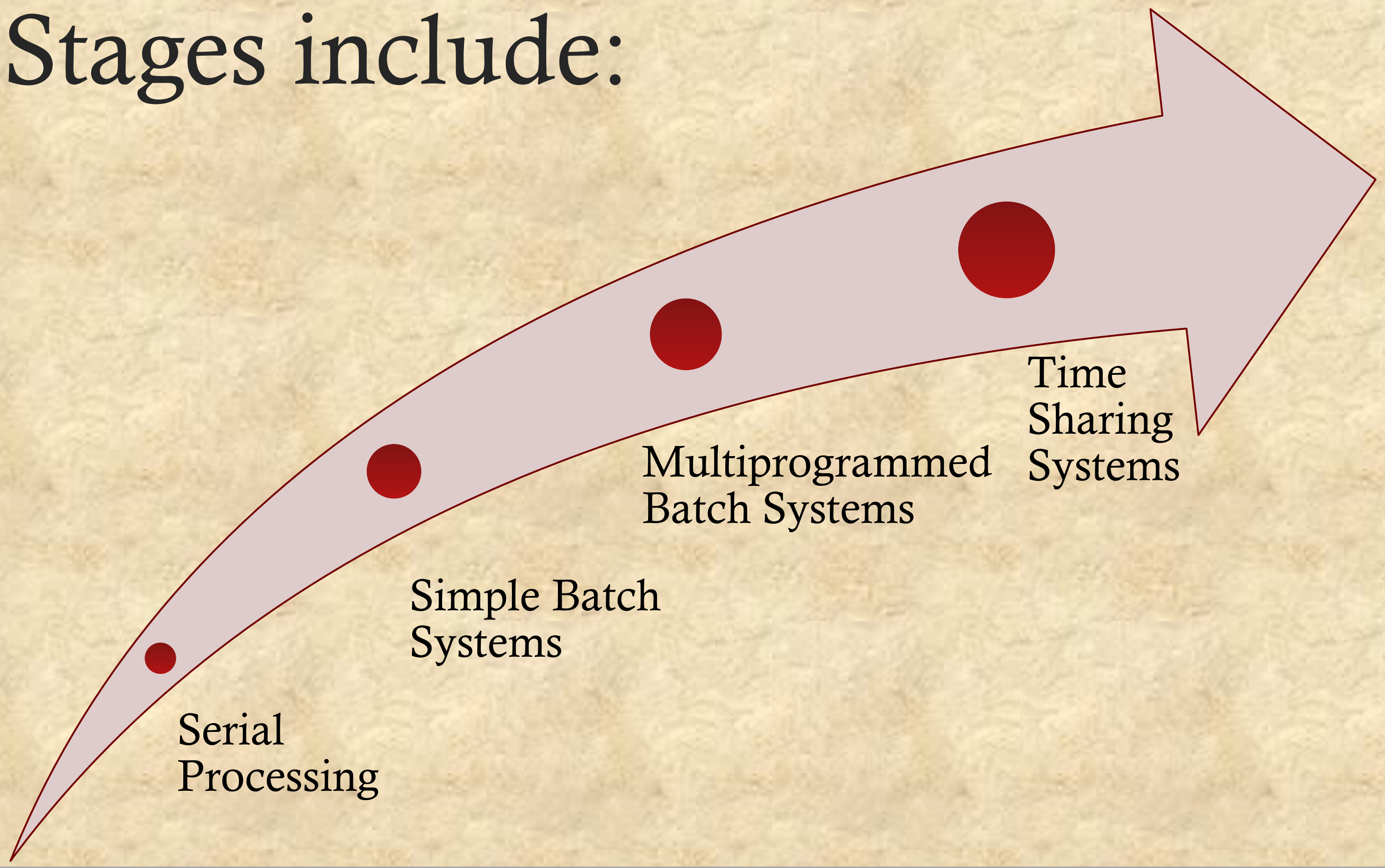
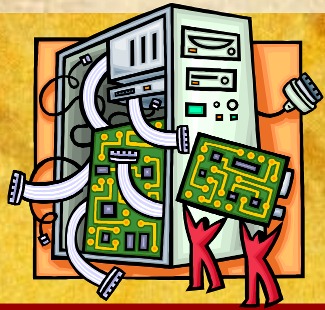


Figure 2.2 The Operating System as Resource Manager

Evolution of Operating Systems

- Stages include:





Serial Processing

Earliest Computers:

- No operating system
  - programmers interacted directly with the computer hardware
- Computers ran from a console with display lights, toggle switches, some form of input device, and a printer
- Users had access to the computer in "series"

Problems:

- Scheduling:
  - most installations used a hardcopy sign-up sheet to reserve computer time
    - time allocations could run short or long, resulting in wasted computer time
- Setup time
  - a considerable amount of time was spent just on setting up the program to run

Simple Batch Systems

- Early computers were very expensive
  - important to maximize processor utilization
- Monitor (primitive operating system)
  - user no longer has direct access to processor
  - job is submitted to computer operator who batches them together and places them on an input device

Monitor Point of View

- Monitor controls the sequence of events
- *Resident Monitor* is software always in memory
- Monitor reads in job and gives control
- Job returns control to monitor

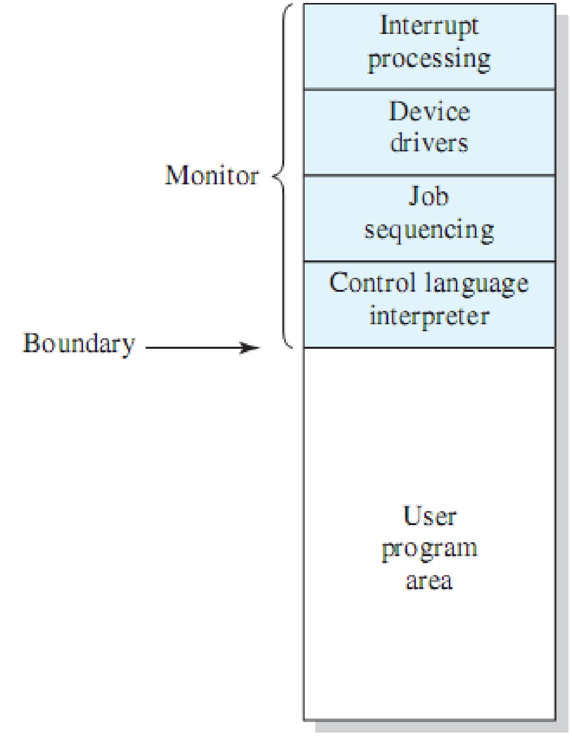


Figure 2.3 **Memory Layout for a Resident Monitor**

Desirable Hardware Features

Memory protection for monitor

- while the user program is executing, it must not alter the memory area containing the monitor

Timer

- prevents a job from monopolizing the system

Privileged instructions

- can only be executed by the monitor

Interrupts

- gives OS more flexibility in controlling user programs

Modes of Operation

User Mode

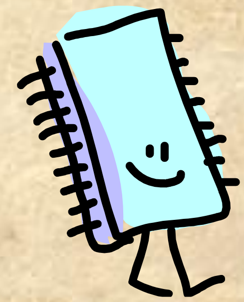
- user program executes in user mode
- certain areas of memory are protected from user access
- certain instructions may not be executed

Kernel Mode

- monitor executes in kernel mode
- privileged instructions may be executed
- protected areas of memory may be accessed

Simple Batch System Overhead

- Processor time alternates between execution of user programs and execution of the monitor
- Sacrifices:
  - some main memory is now given over to the monitor
  - some processor time is consumed by the monitor
- Despite overhead, the simple batch system improves utilization of the computer.



Multiprogrammed Batch Systems

Read one record from file	15 µs
Execute 100 instructions	1 µs
Write one record to file	15 µs
TOTAL	31 µs

$$\text{Percent CPU Utilization} = \frac{1}{31} = 0.032 = 3.2\%$$

Figure 2.4 System Utilization Example

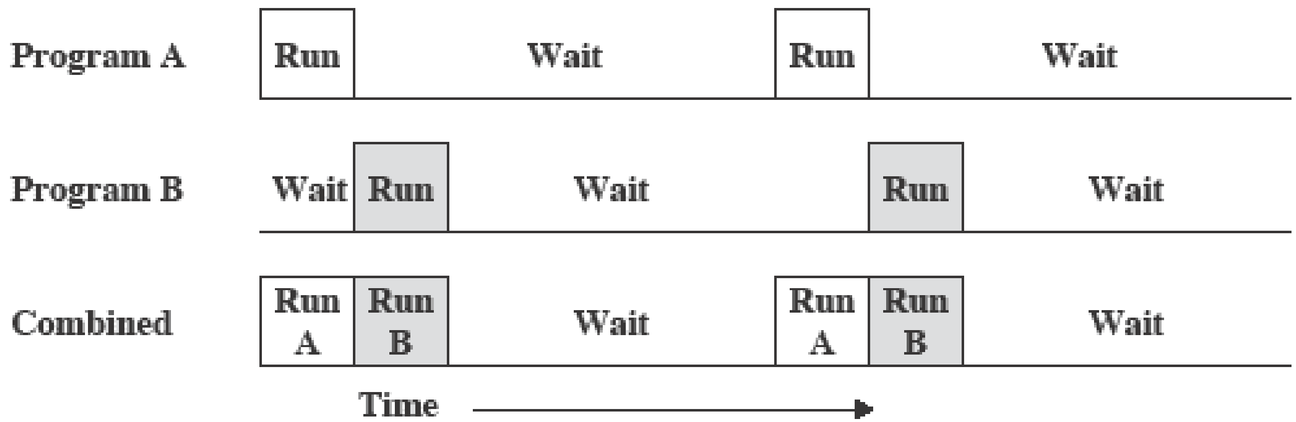
- Processor is often idle
  - even with automatic job sequencing
  - I/O devices are slow compared to processor

Uniprogramming

(a) Uniprogramming

- The processor spends a certain amount of time executing, until it reaches an I/O instruction; it must then wait until that I/O instruction concludes before proceeding

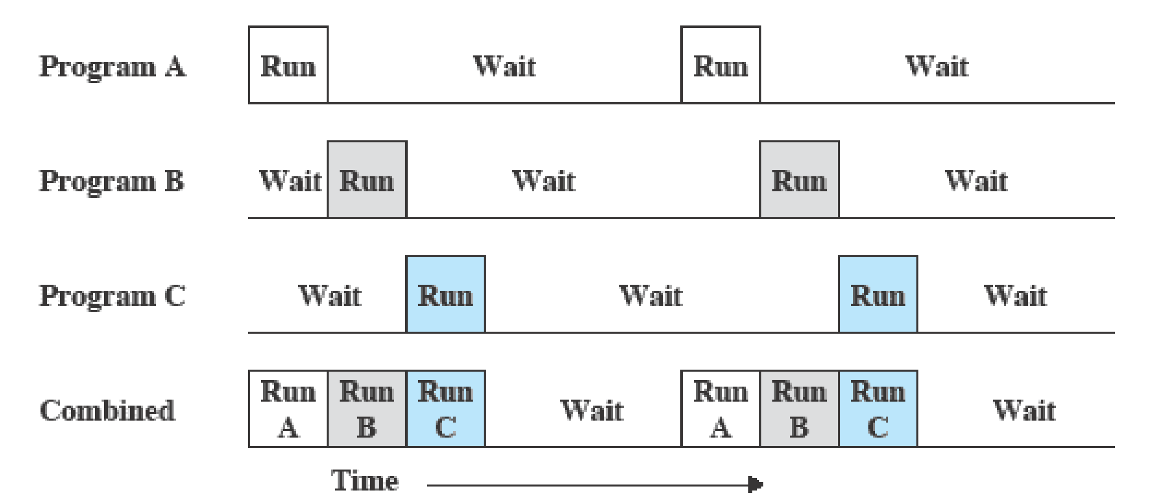
Multiprogramming



(b) Multiprogramming with two programs

- What if there's enough memory to hold the OS (resident monitor) and *two* user programs.
- When one job needs to wait for I/O, the processor can switch to the other job, which may not be waiting.

Multiprogramming



(c) Multiprogramming with three programs

- Multiprogramming
  - also known as multitasking
  - memory is expanded to hold three, four, or more programs and switch among all of them

Multiprogramming Example

Table 2.1 Sample Program Execution Attributes

	JOB1	JOB2	JOB3
Type of job	Heavy compute	Heavy I/O	Heavy I/O
Duration	5 min	15 min	10 min
Memory required	50 M	100 M	75 M
Need disk?	No	No	Yes
Need terminal?	No	Yes	No
Need printer?	No	No	Yes

Effects on Resource Utilization

	Uniprogramming	Multiprogramming
Processor use	20%	40%
Memory use	33%	67%
Disk use	33%	67%
Printer use	33%	67%
Elapsed time	30 min	15 min
Throughput	6 jobs/hr	12 jobs/hr
Mean response time	18 min	10 min

Table 2.2 Effects of Multiprogramming on Resource Utilization

Time-Sharing Systems

- Can be used to handle multiple *interactive* jobs
- Processor time is shared among multiple users
- Origin: multiple users simultaneously access the system through terminals, with the OS interleaving the execution of each user program in a short burst or quantum of computation

Batch Multiprogramming vs. Time Sharing

	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

Table 2.3 Batch Multiprogramming versus Time Sharing

Compatible Time-Sharing Systems

CTSS

- One of the first time-sharing operating systems
- Developed at MIT by a group known as Project MAC for IBM 709/7094
- Ran on a computer with 32,000 *36*-bit words of main memory, with the resident monitor consuming 5000 of that
- To simplify both the monitor and memory management a program was always loaded to start at the location of the 5000th word

Time Slicing

- System clock generates interrupts at a rate of approximately one every 0.2 seconds
- At each interrupt OS regained control and could assign processor to another user
- At regular time intervals the current user would be preempted and another user loaded in
- Old user programs and data were written out to disk
- Old user program code and data were restored in main memory when that program was next given a turn

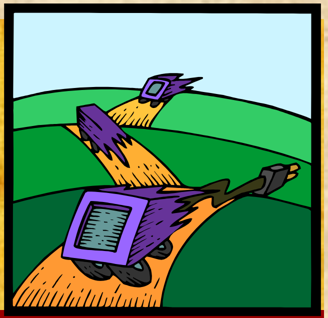
Major Advances

- Operating Systems are among the most complex pieces of software ever developed

Major advances in development include:

- Processes
- Memory management
- Information protection and security
- Scheduling and resource management
- System structure

Process



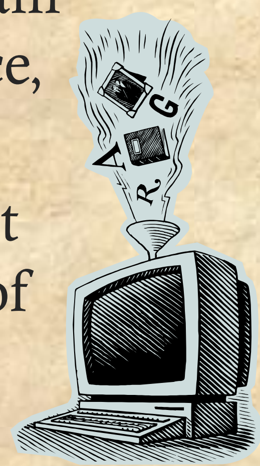
- Fundamental to the structure of operating systems

A *process* can be defined as:

- a program in execution
- an instance of a running program
- the entity that can be assigned to, and executed on, a processor
- a unit of activity characterized by a single sequential thread of execution, a current state, and an associated set of system resources

Components of a Process

- A process contains three components:
  - an executable program
  - the associated data needed by the program (variables, work space, buffers, etc.)
  - the execution context (or "process state") of the program



- The execution context is essential:
  - it is the internal data by which the OS is able to supervise and control the process
  - includes the contents of the various process registers
  - includes information such as the priority of the process and whether the process is waiting for the completion of a particular I/O event

Process Management

- The entire state of the process at any instant is contained in its context
- New features can be designed and incorporated into the OS by expanding the context to include any new information needed to support the feature

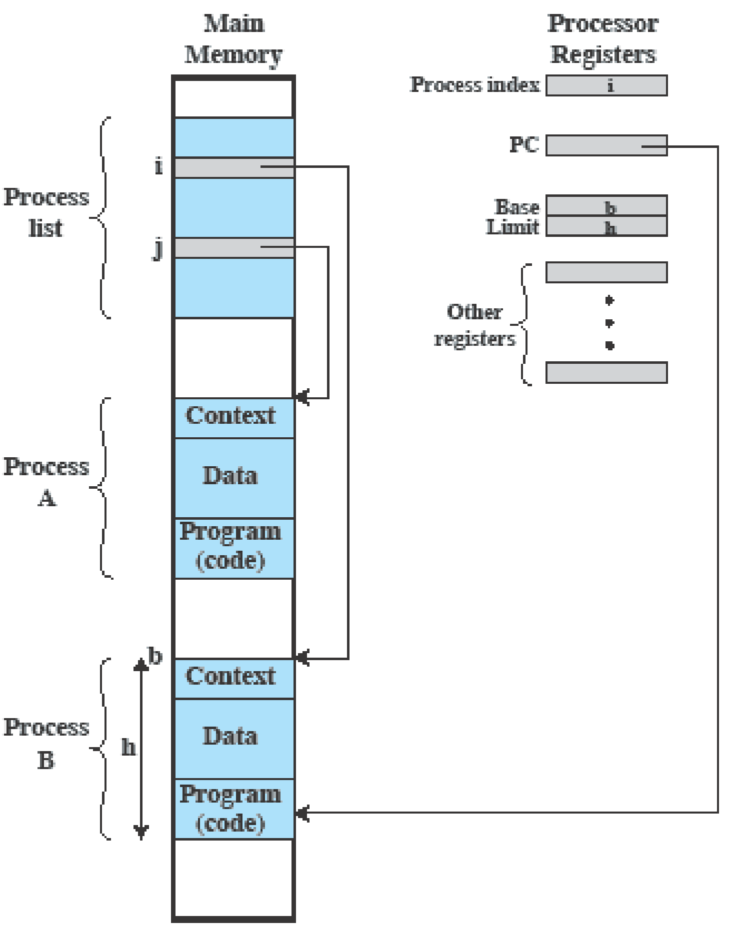


Figure 2.8 Typical Process Implementation

Multithreading

- Technique in which a process, executing an application, is divided into threads that can run concurrently

Thread

- dispatchable unit of work
- includes a processor context and its own data area to enable subroutine branching
- executes sequentially and is interruptible

Process

- a collection of one or more threads and associated system resources
- programmer has greater control over the modularity of the application and the timing of application related events

Symmetric Multiprocessing (SMP)

- Term that refers to a computer hardware architecture and also to the OS behavior that exploits that architecture
- Several processes can run in parallel
- Multiple processors are transparent to the user
  - these processors share same main memory and I/O facilities
  - all processors can perform the same functions
- The OS takes care of scheduling of threads or processes on individual processors and of synchronization among processors

SMP Advantages

Performance	more than one process can be running simultaneously, each on a different processor
Availability	failure of a single process does not halt the system
Incremental Growth	performance of a system can be enhanced by adding an additional processor
Scaling	vendors can offer a range of products based on the number of processors configured in the system

Multiprogramming

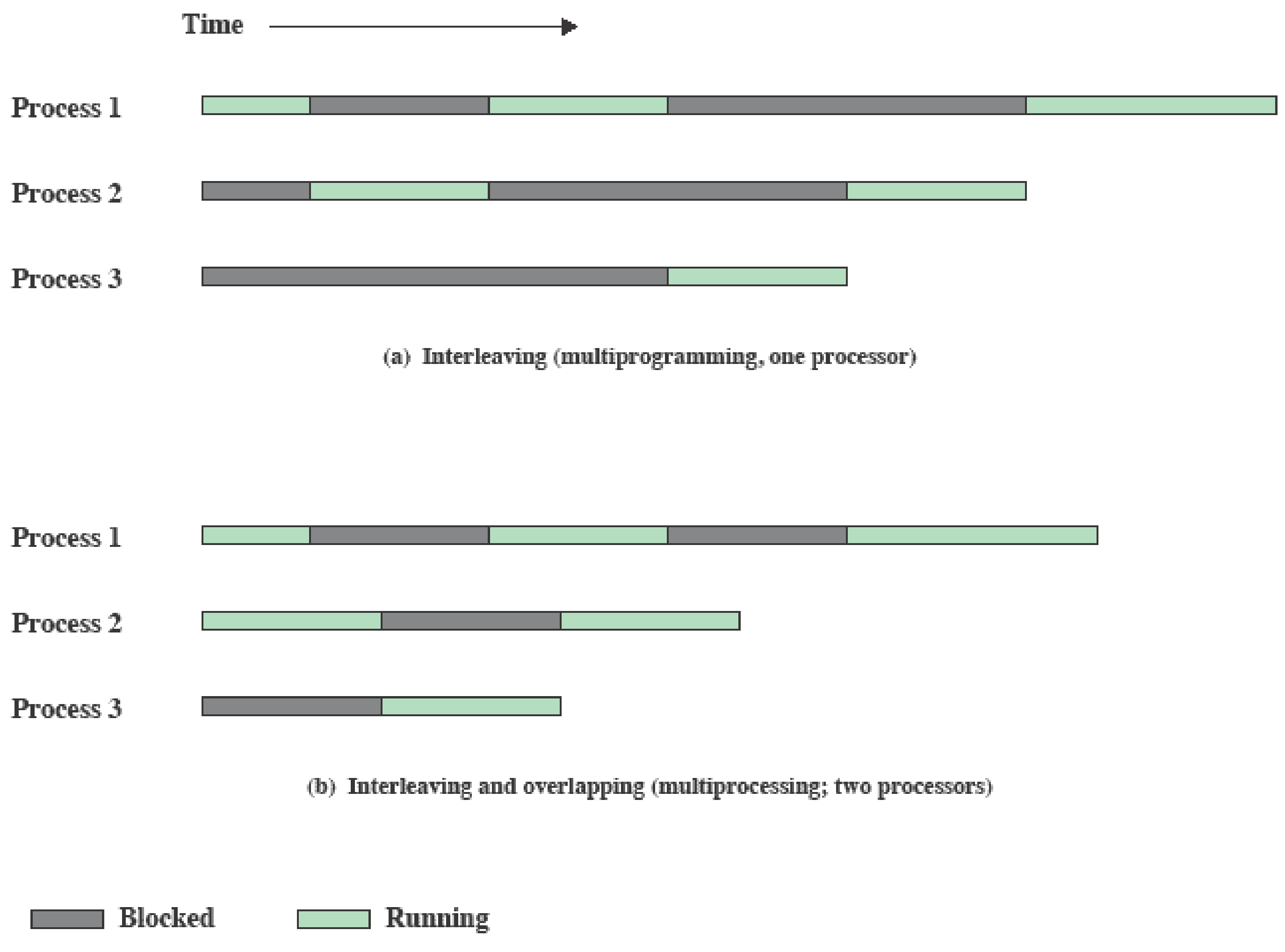


Figure 2.12 Multiprogramming and Multiprocessing

Virtual Machines and Virtualization

- Virtualization
  - enables a single PC or server to simultaneously run multiple operating systems or multiple sessions of a single OS
  - Each (guest) operating system runs in a **virtual machine (VM)**, and can execute multiple applications
  - Guest operating systems execute as if they were interacting directly with the hardware, but in fact they are interacting with a Virtual Machine Monitor (VMM) which runs directly on the hardware or on a **host** operating system

Virtual Machine Concept

Note: In some cases, servers in particular, the VMM runs directly on the hardware. This figure represents a *hosted* virtual machine

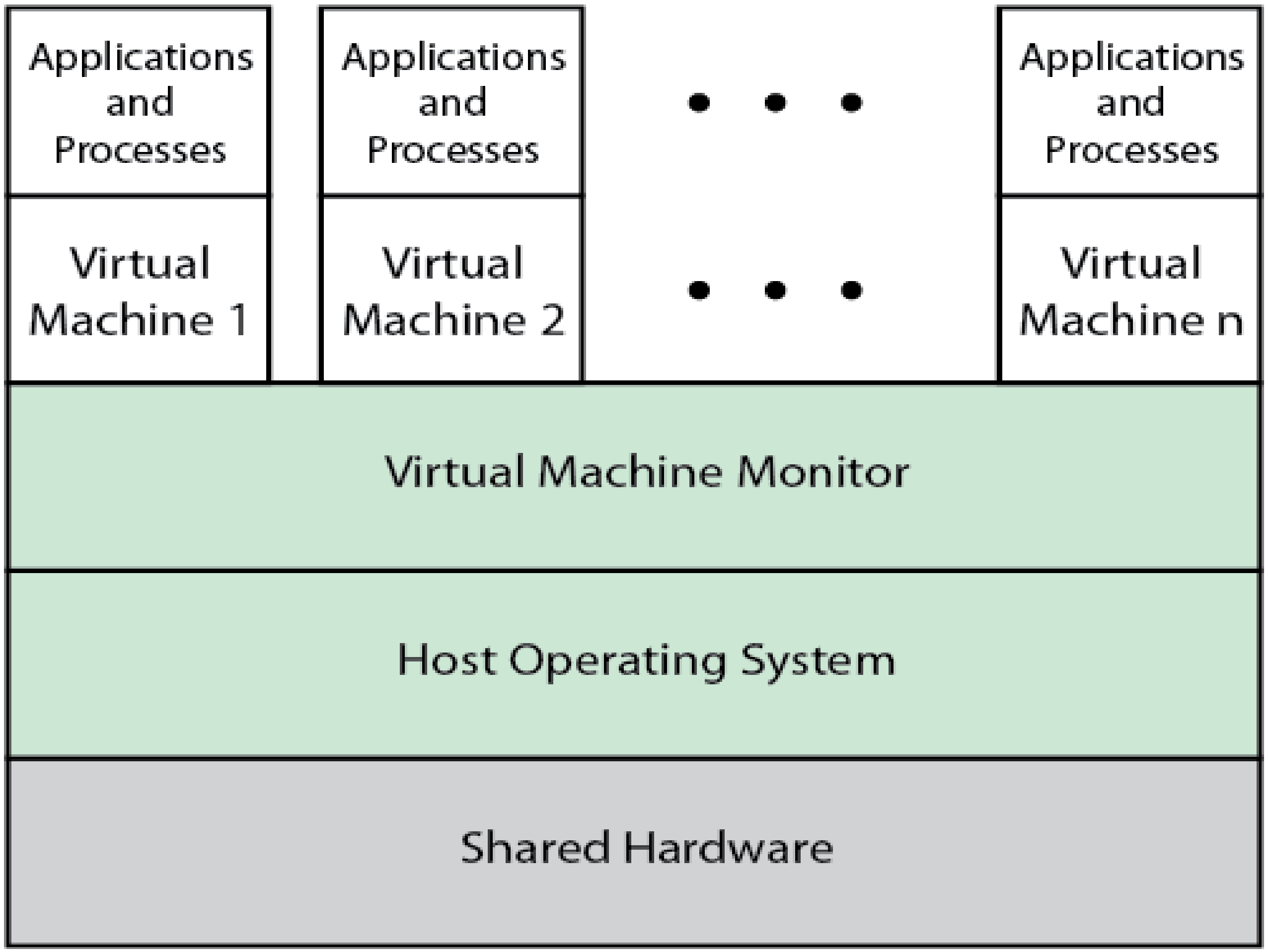
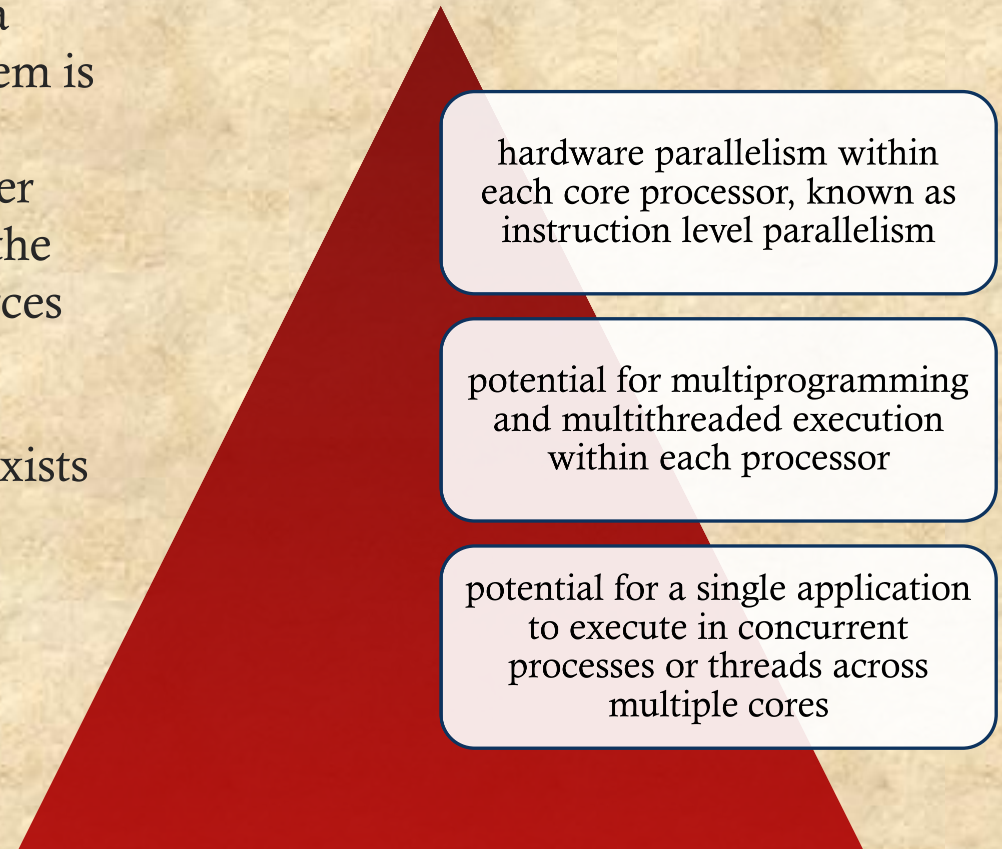


Figure 2.13 Virtual Memory Concept

Correction: should be Virtual MACHINE Concept

Multicore OS Considerations

- The design challenge for a many-core multicore system is to efficiently harness the multicore processing power and intelligently manage the substantial on-chip resources efficiently
- Potential for parallelism exists at three levels:



Microsoft Windows Overview

- **MS-DOS 1.0** released in 1981
  - 4000 lines of assembly language source code
  - ran in 8 Kbytes of memory
  - used Intel 8086 microprocessor
- **Windows 3.0** shipped in 1990
  - 16-bit
  - GUI interface
  - implemented as a layer on top of MS-DOS
- **Windows 95**
  - 32-bit version
  - led to the development of Windows 98 and Windows Me
- **Windows NT (3.1)** released in 1993
  - 32-bit OS with the ability to support older DOS and Windows applications as well as provide OS/2 support
- **Windows 2000**
  - included services and functions to support distributed processing
  - Active Directory
  - plug-and-play and power-management facilities
- **Windows XP** released in 2001
  - goal was to replace the versions of Windows based on MS-DOS with an OS based on NT
- **Windows Vista** shipped in 2007
- **Windows Server** released in 2008
- **Windows 7** shipped in 2009, as well as **Windows Server 2008 R2**
- **Windows Azure**
  - targets cloud computing

Threads and SMP

- Two important characteristics of Windows are its support for threads and for symmetric multiprocessing (SMP)
  - OS routines can run on any available processor, and different routines can execute simultaneously on different processors
  - Windows supports the use of multiple threads of execution within a single process. Multiple threads within the same process may execute on different processors simultaneously
  - Server processes may use multiple threads to process requests from more than one client simultaneously
  - Windows provides mechanisms for sharing data and resources between processes and flexible interprocess communication capabilities

Traditional UNIX Systems

- Were developed at Bell Labs and became operational on a PDP-7 in 1970
- Incorporated many ideas from Multics
- PDP-11was a milestone because it first showed that UNIX could be an OS for all computers
- Next milestone was rewriting UNIX in the programming language C
  - demonstrated the advantages of using a high-level language for system code
- Was described in a technical journal for the first time in 1974
- First widely available version outside Bell Labs was Version 6 in 1976
- Version 7, released in 1978 is the ancestor of most modern UNIX systems
- Most important of the non-AT&T systems was UNIX BSD (Berkeley Software Distribution)

LINUX Overview

- Started out as a UNIX variant for the IBM PC
- Linus Torvalds, a Finnish student of computer science, wrote the initial version
- Linux was first posted on the Internet in 1991
- Today it is a full-featured UNIX system that runs on several platforms
- Is free and the source code is available
- Key to success has been the availability of free software packages
- Highly modular and easily configured

Modular Monolithic Kernel

- Most UNIX systems are monolithic (includes virtually all of the OS functionality in one large block of code that runs as a single process with a single address space)
- All the functional components of the kernel have access to all of its internal data structures and routines
- Linux improves on this somewhat because it is structured as a collection of modules

Loadable Modules

- Relatively independent blocks
- A module is an object file whose code can be linked to and unlinked from the kernel at runtime
- A module is executed in kernel mode on behalf of the current process
- Have two important characteristics:
  - Dynamic linking
  - Stackable modules