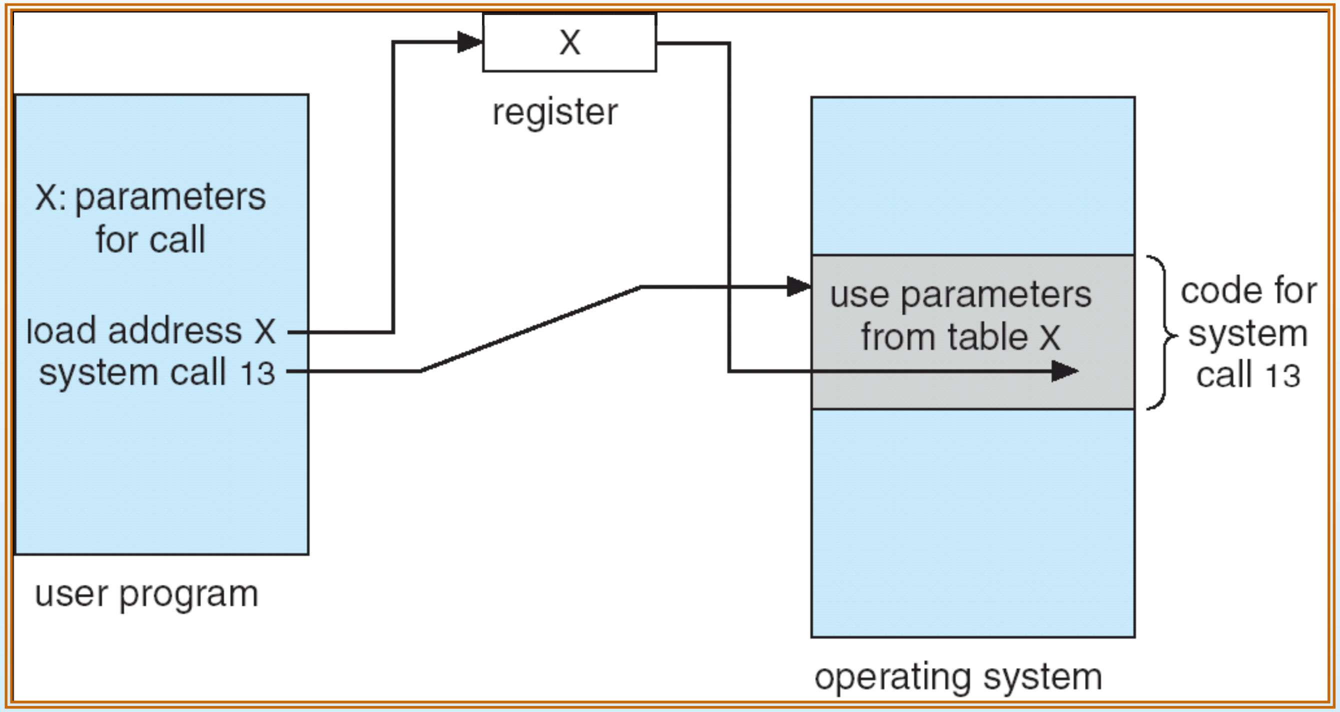
System Call Parameter Passing

- Often, more information is required when designing system call
  - Information varies according to OS and types of system call
- Three general methods used to pass parameters to the OS
  - Simplest: pass the parameters in *registers*
    - In some cases, may be more parameters than registers
  - Parameters placed, or *pushed,* onto the *stack* by the program and *popped* off the stack by the operating system
  - Parameters are stored in other places, such as a block or table, in memory, and the address of block or table is passed as a parameter in a register (next slide)
    - This approach taken by Linux and Solaris

Parameter Passing via Table



X is not the actual value but the address.

Types of System Calls

- Categories are similar to categories of library functions

- Process control
  - Create, terminate, allocate/free memory
- File management
  - Create/delete, open/close, read/write
- Device management
  - Request/release, read/write
- Information maintenance
  - Get/set the time/date, get system data
- Communications
  - Create/connect communication connection, send/receive

Types of System Calls

	Windows	Unix
Process Control	CreateProcess() ExitProcess() WaitForSingleObject()	fork() exit() wait()
File Manipulation	CreateFile() ReadFile() WriteFile() CloseHandle()	open() read() write() close()
Device Manipulation	SetConsoleMode() ReadConsole() WriteConsole()	ioctl() read() write()
Information Maintenance	GetCurrentProcessID() SetTimer() Sleep()	getpid() alarm() sleep()
Communication	CreatePipe() CreateFileMapping() MapViewOfFile()	pipe() shmget() mmap()
Protection	SetFileSecurity() InitlializeSecurityDescriptor() SetSecurityDescriptorGroup()	chmod() umask() chown()

System Program

- Provides a convenient environment of program development and execution
  - Some of them are simply user interfaces to system calls; others are considerably more complex
- The followings are six system program categories
- File management - Create, delete, copy, rename, print, dump, list, and generally manipulate files and directories
- Status information
  - Some ask the system information - date, time, amount of available memory, disk space, number of users
  - Others provide detailed performance, logging, and debugging information
  - Typically, these programs provide formatted output and print the output to the terminal or other output devices
  - Some systems implement a registry - used to store and retrieve configuration information

System Program (cont'd)

- File modification
  - Text editors may be available to create and modify files
  - Special commands to search contents of files or perform transformations of the text to other types
- Programming-language support - Compilers, assemblers, debuggers and interpreters are sometimes provided
- Program loading and execution- Absolute loaders, relocatable loaders, linkage editors, and overlay-loaders, debugging systems for higher-level and machine language
- Communications - Provide the mechanism for creating virtual connections among processes, users, and computer systems
  - Allow users to send messages to another's screens, browse web pages, send electronic-mail messages, log in remotely, transfer files from one machine to another

Operating System Design and Implementation

- Design and Implementation of OS is not "solvable", but some approaches have proven "successful"
- Internal structure of different Operating Systems can vary widely
- Start by defining goals and specifications
- Affected by the 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 efficient, flexible, reliable, and error-free

Operating System Design and Implementation (Cont.)

- Important principle to separate

  Policy: What will be done?
  Mechanism: How to do it?

  → Flexibility!

- Mechanisms determine how to do something, policies decide what will be done
  - The separation of policy from mechanism is a very important principle, it allows maximum **flexibility**.
- Example
  - CPU timer construction
    - Policy : How long is the timer to be set?
    - Mechanism : Detailed various timer construction methods
  - Priority Management
    - Policy : Priority decision
    - Mechanism : Detailed Control method for handling priority

Operating System Structure

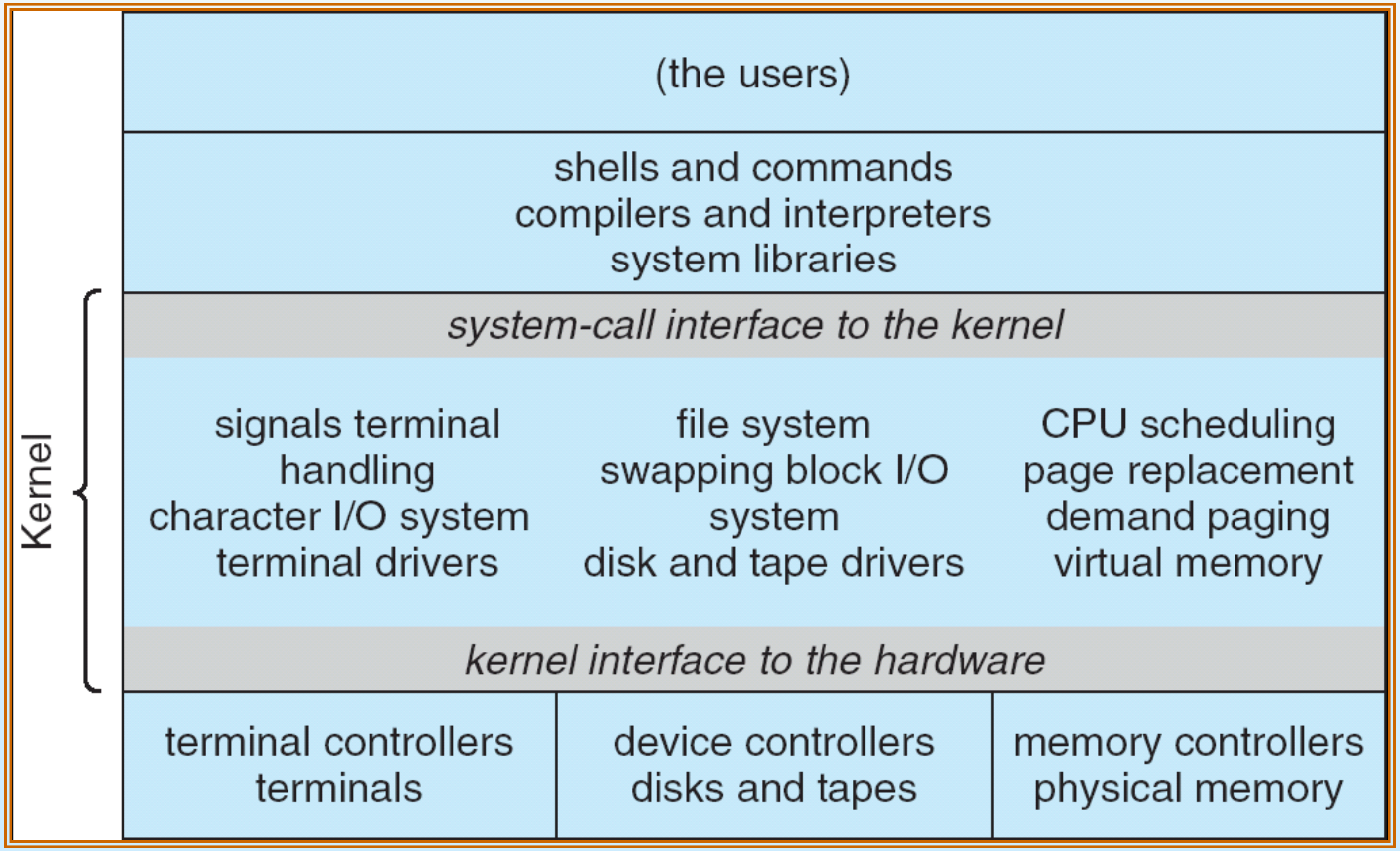
- Simple Structure
- Layered Approach
- Microkernel
- Monolithic
- Module

Operating System Structure

- Simple Structure
  - MS-DOS – written to provide the most functionality in the least space
    - Not divided into modules
    - Although MS-DOS has some structure, its interfaces and levels of functionality are not well separated
- Layered Approach
  - The operating system is divided into a number of layers (levels)
  - Each is built on top of the nearest lower layers.
  - The bottom layer (layer 0), is the hardware; the highest (layer N) is the user interface.

Layered Approach Example

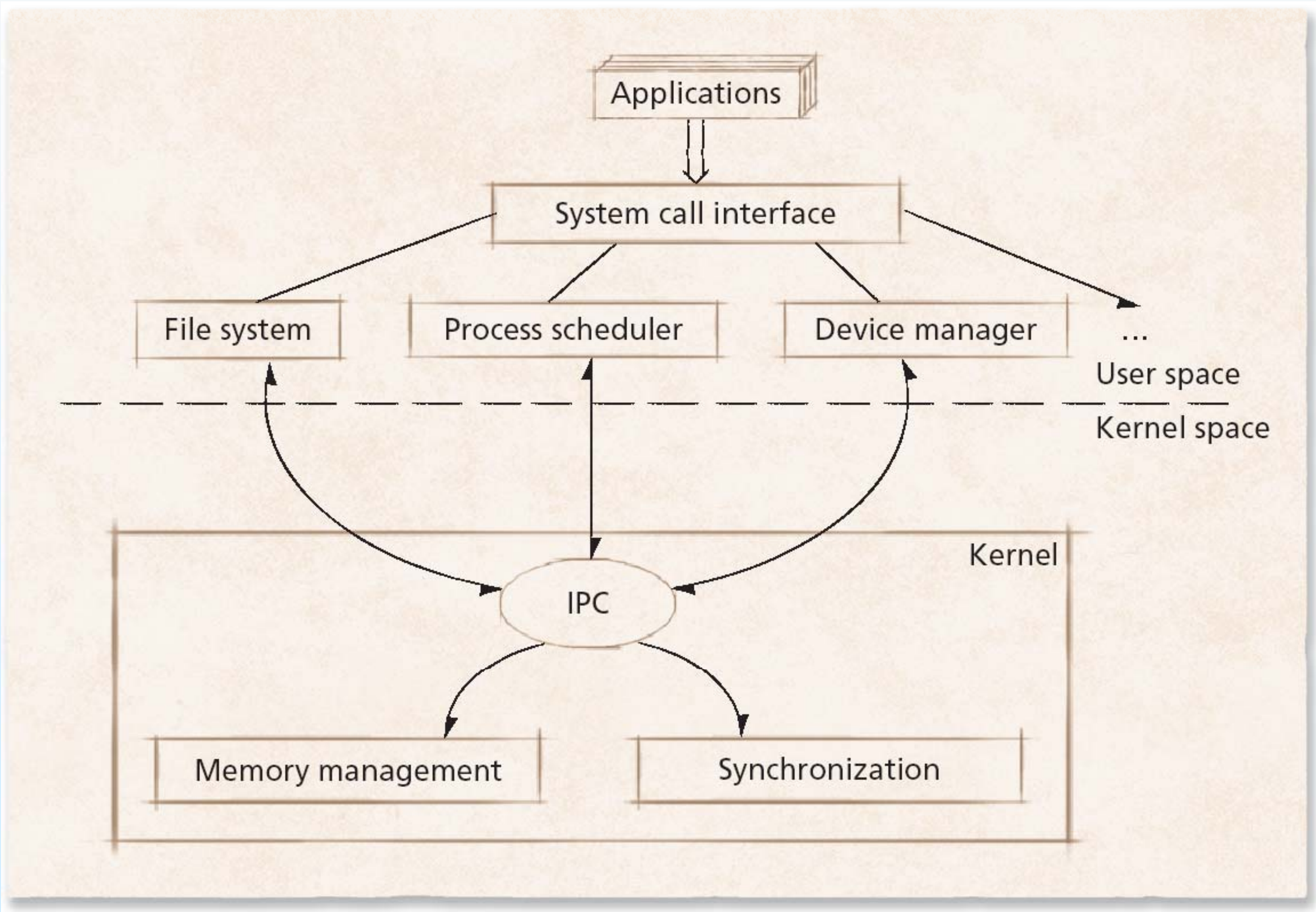
- UNIX – the original UNIX operating system had <u>limited structuring</u>. The UNIX OS <u>consists of separable layers</u>



Microkernel System Structure

- Moves as much as possible from kernel space into "*user*" space
  - It only implements hardware dependent functions or real-time facilities.
- Communication takes place between user modules using message passing
- Benefits:
  - Easier to extend or modify a microkernel
  - Easier to port the operating system to new architectures
  - More reliable (less code is running in kernel mode)
  - More secure
- Detriments:
  - Communication overhead
    - Performance overhead of user space to kernel space communication
- Example
  - Mach

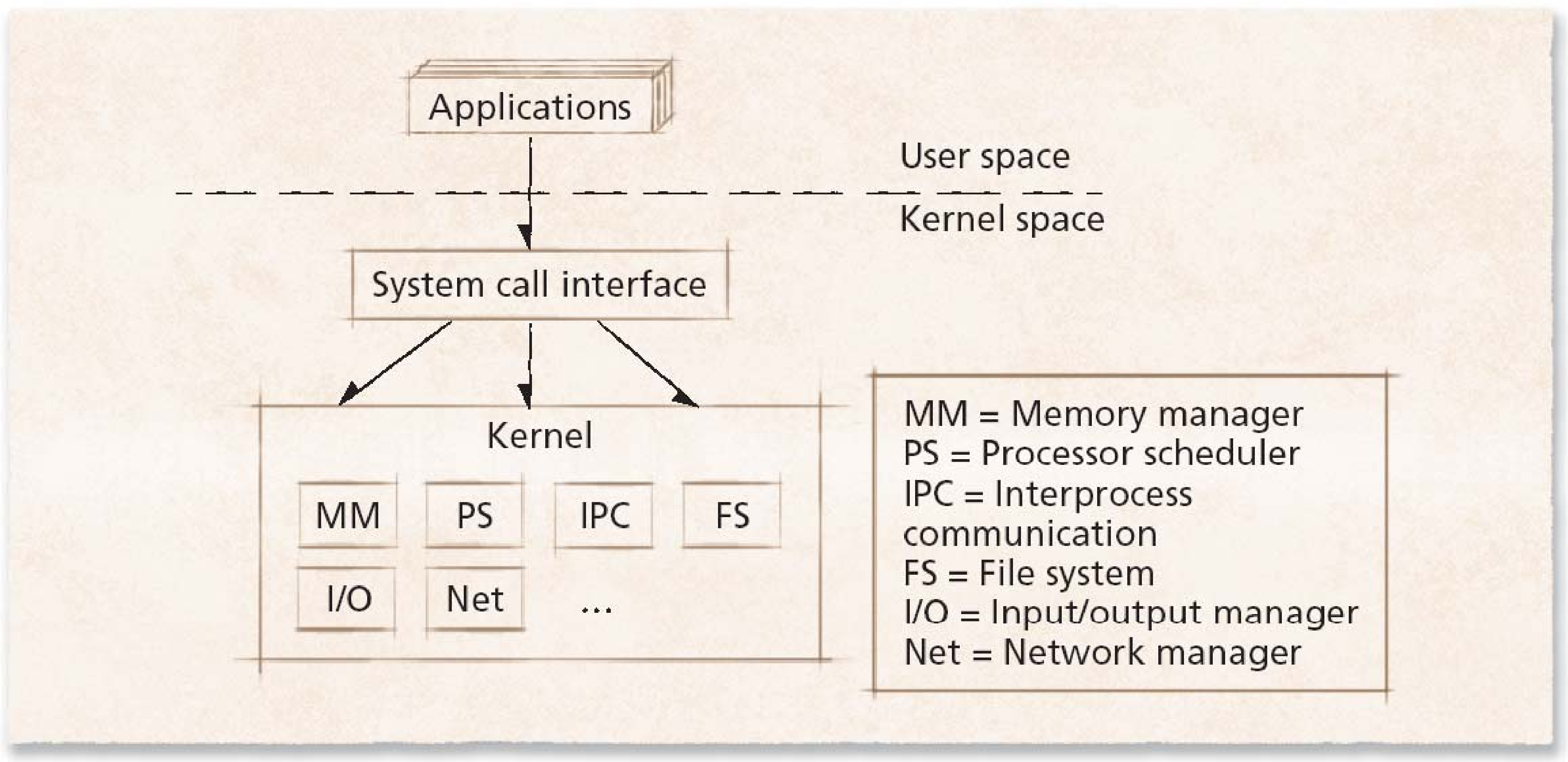
Microkernel System Structure



Monolithic System Structure

- Kernel controls all system functionalities
- Compared to microkernel structure, it is fast and efficient to manage resources.
- Detriments
  - Size overhead
    - All functionalities are resided in memory
  - Recompile and rebooting is necessary when modified
- Example
  - Unix (BSD Family), Solaris

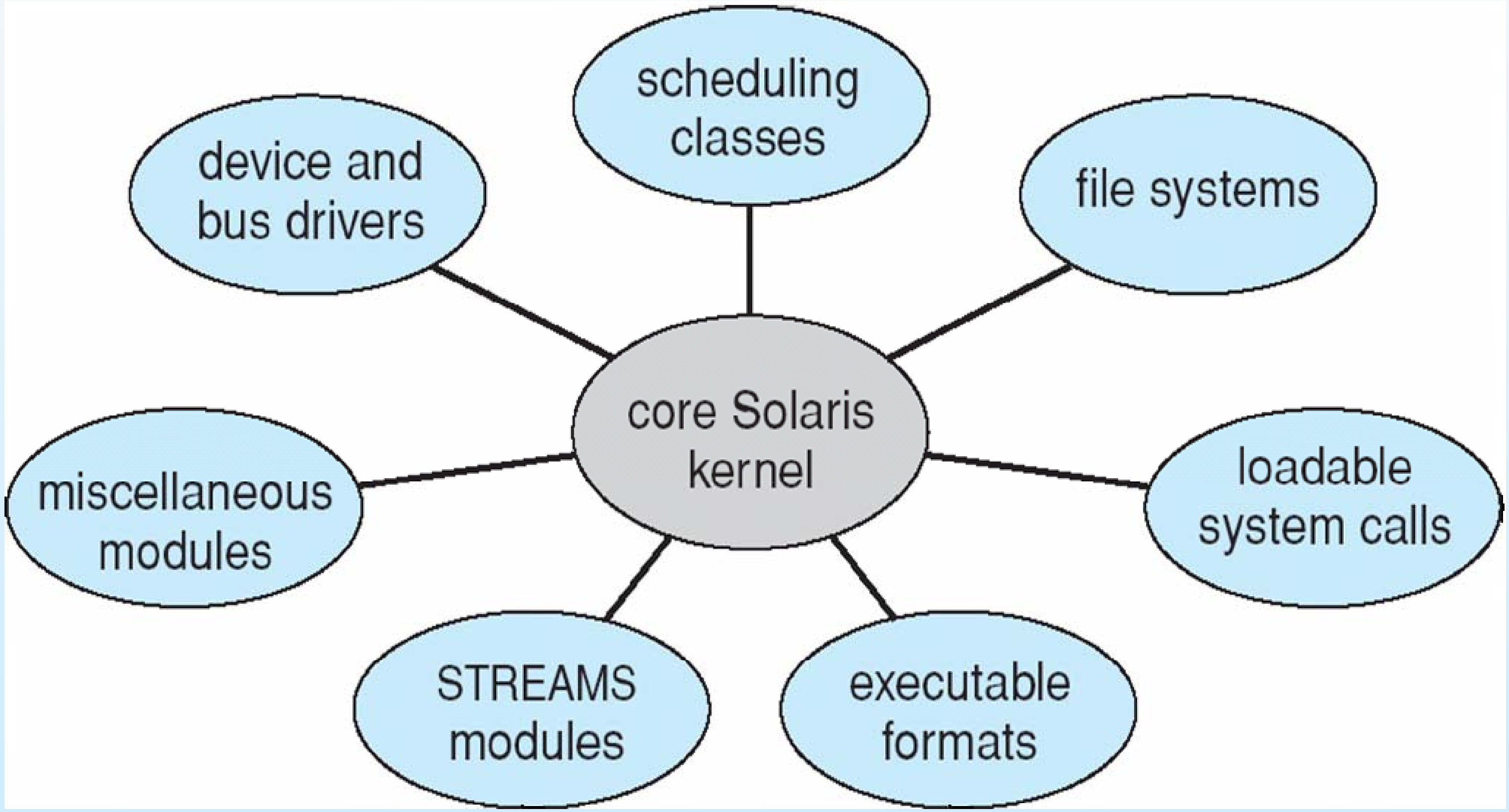
Monolithic System Structure



Monolithic with Module

- Monolithic architecture with Module
  - Kernel includes basic component
  - Non-frequently used parts are implemented by Module and it is loaded dynamically
  - Module is also useful to implement new device drivers or system calls
  - Example: Linux

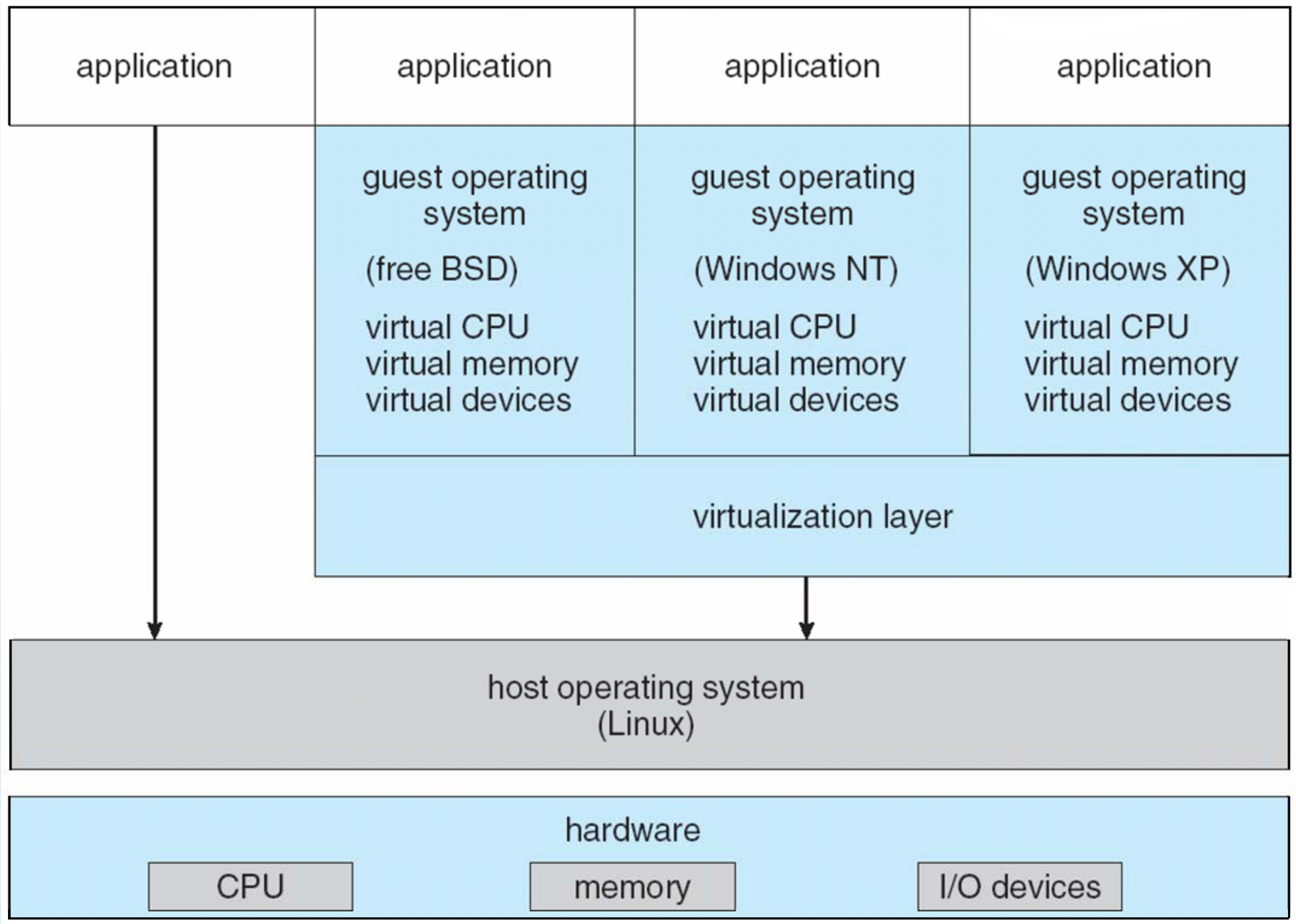
Solaris Loadable Module



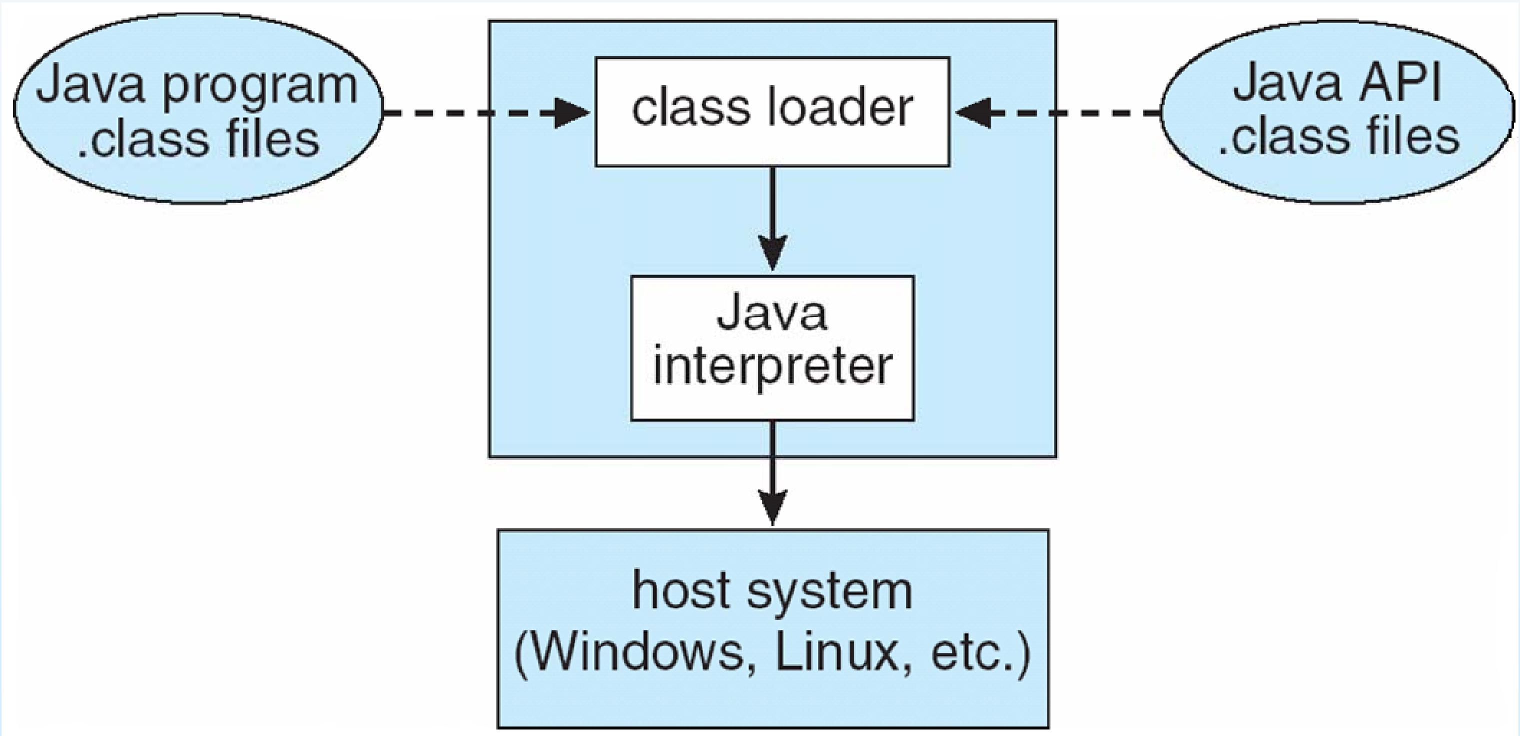
Virtual Machines

- A virtual machine takes the layered approach.
- It treats hardware and the operating system kernel as if they were all hardware
- A virtual machine provides an identical interface to the underlying bare hardware
- The operating system host creates the illusion that a process has its own processor and virtual memory
- Each guest is provided with a (virtual) copy of underlying computer
- Example
  - JAVA virtual machine
  - VMWare, VirtualBox, etc

VMware Architecture



The Java Virtual Machine



End of Chapter 2