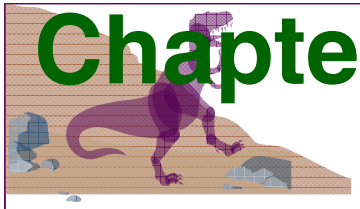
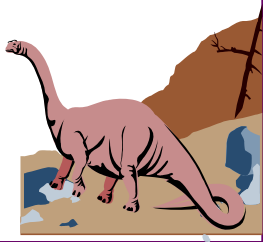
Chapter 2: Operating-System Structures

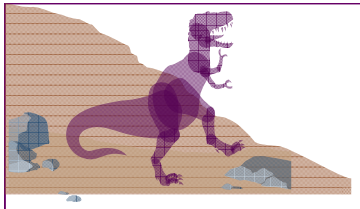
肖 卿 俊
办公室：计算机楼212室
电邮：csqjxiao@seu.edu.cn
主页：http://cse.seu.edu.cn/PersonalPage/csqjxiao
电话：025-52091022



Chapter 2: Operating-System Structures (7th Edition)

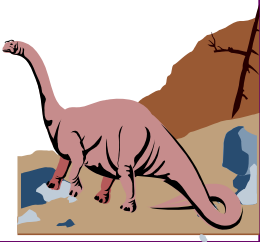
- Operating System Services
- User Operating System Interface
- System Calls
- Types of System Calls
- System Programs
- Operating System Design and Implementation
- Operating System Structure
- Virtual Machines
- Operating System Generation
- System Boot

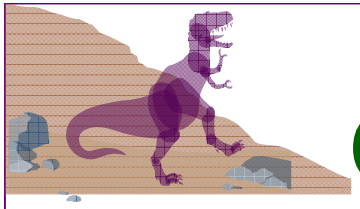




Chapter 3: Operating-System Structures (6th Edition)

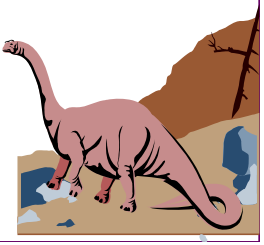
- System Components
- Operating System Services
- System Calls
- System Programs
- System Structure
- Virtual Machines
- System Design and Implementation
- System Generation

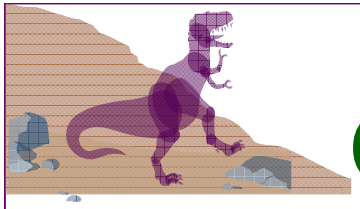




Question about OS Services

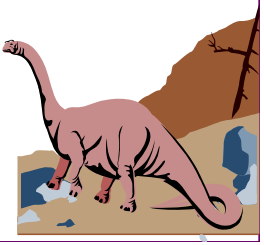
- Name (as many as you can) system services you expect from an operating system

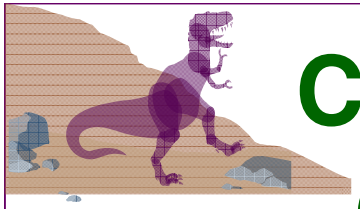




Question about OS Services

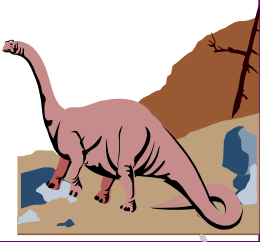
- Name (as many as you can) system services you expect from an operating system
  - Process scheduling (or job scheduling)
  - Inter-process communication (IPC)
  - Memory management
    - ✓Protection, sharing, demand paging
  - File system for organizing external storage
  - Access to I/O devices, e.g., microphones, speaker
  - Access to the networks

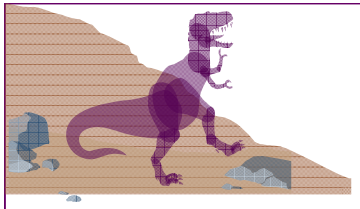




Common System Components (and Types of System Calls)

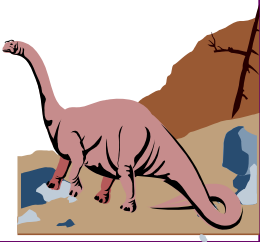
- Process Management
- Main Memory Management
- File Management
- I/O System Management
- Secondary-Storage Management
- Networking
- Protection System
- Command-Interpreter System

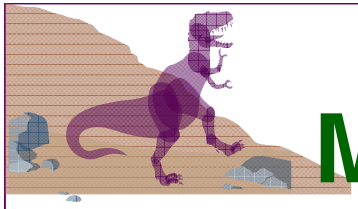




Process Management

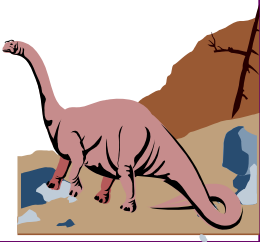
- A *process* is a program in execution.
- The operating system is responsible for the following activities in connection with process management.
  - Process creation and deletion.
  - Process suspension and resumption.
  - Provision of mechanisms for:
    - ✓Process synchronization
    - ✓Process communication
    - ✓Deadlock handling





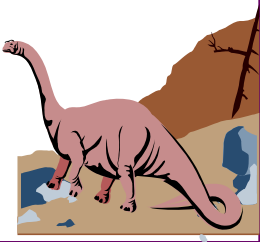
Main-Memory Management

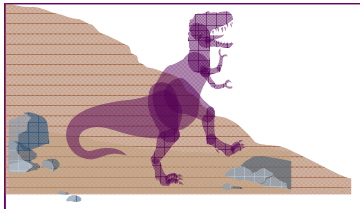
- Memory is a large array of words or bytes, each with its own address. It is a repository of quickly accessible data shared by the CPU and I/O devices.
- Main memory is a volatile storage device. It loses its contents in the case of system failure.



Main-Memory Management (Cont.)

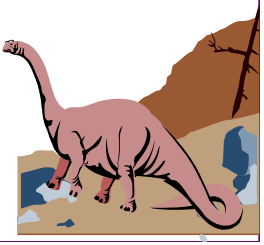
- The operating system is responsible for the following activities in connections with memory management:
  - Keep track of which parts of memory are currently being used and by whom.
  - Decide which processes to load when memory space becomes available.
  - Allocate and deallocate memory space as needed.

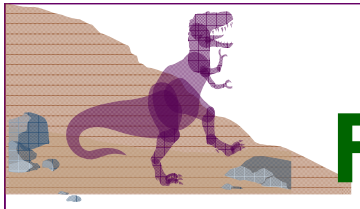




File Management

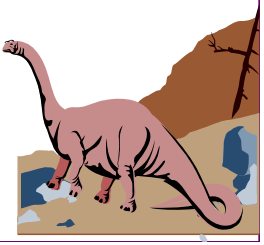
- There are different types of physical media to store information. Each of them has its own characteristics and physical organization
- Operating System provides a uniform logical view of information storage, i.e., file.
- A file is a collection of related information defined by its creator. Commonly, files represent programs (both source and object forms) and data.





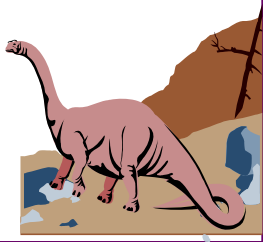
File Management (Cont.)

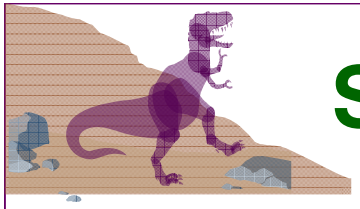
- The operating system is responsible for the following activities in connections with file management:
  - File creation and deletion.
  - Directory creation and deletion.
  - Support of primitives for manipulating files and directories.
  - Mapping files onto secondary storage.
  - File backup on stable (nonvolatile) storage media.



Secondary-Storage Management

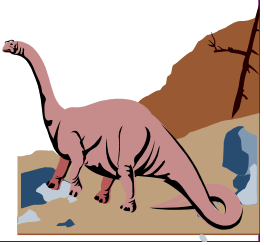
- Since main memory (*primary storage*) is volatile and too small to accommodate all data and programs permanently, the computer system must provide *secondary storage* to back up main memory.
- Most modern computer systems use disks as the principle on-line storage medium, for both programs and data.

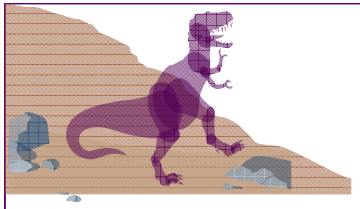




Secondary-Storage Management (Cont.)

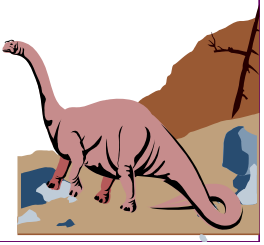
- The operating system is responsible for the following activities in connection with disk management:
  - Free space management
  - Storage allocation
  - Disk scheduling

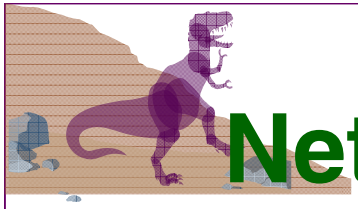




I/O System Management

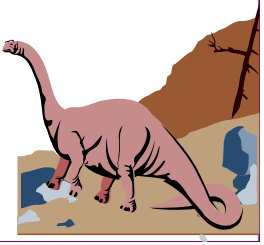
- The I/O subsystem consists of:
  - A buffer-caching system
  - A general device-driver interface
  - Drivers for specific hardware devices

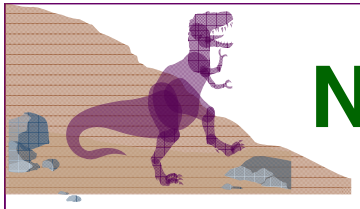




Networking (Distributed Systems)

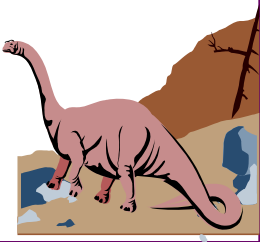
- A *distributed* system is a collection processors that do not share memory or a clock. Each processor has its own local memory.
- The processors in the system are connected through a communication network.
- Communication takes place using a *protocol.*

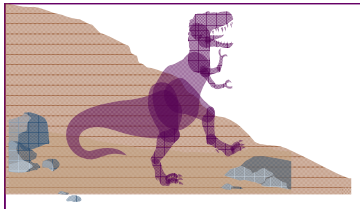




Networking (Distributed Systems) (Cont.)

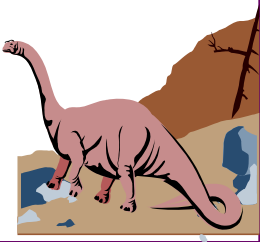
- A distributed system provides user access to various system resources.
- Access to a shared resource allows:
  - Computation speed-up
  - Increased data availability
  - Enhanced reliability

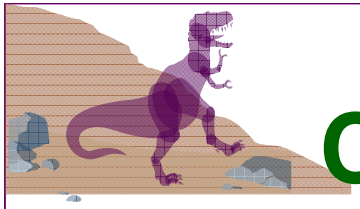




Protection System

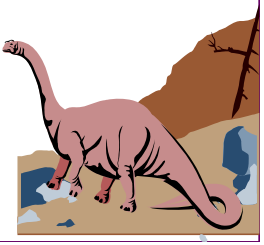
- *Protection* refers to a mechanism for controlling access by programs, processes, or users to both system and user resources.
- The protection mechanism must:
  - distinguish between authorized and unauthorized usage.
  - specify the controls to be imposed and means for enforcement.

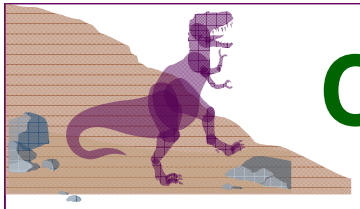




Command-Interpreter System

- Many commands are given to the operating system by control statements which deal with:
  - process creation and management
  - I/O handling
  - secondary-storage management
  - main-memory management
  - file-system access
  - protection
  - networking

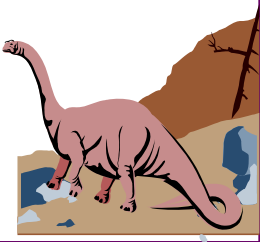


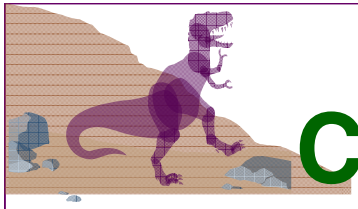


Command-Interpreter System (Cont.)

- The program that reads and interprets control statements is called variously:
  - command-line interpreter
  - shell (in UNIX)

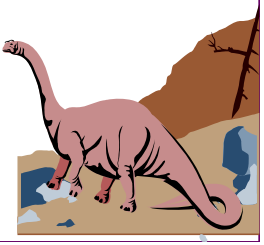
Its function is to get and execute the next command statement.

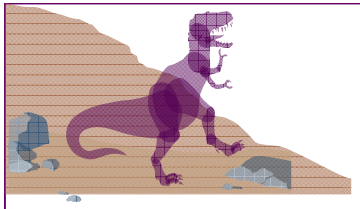




Common System Components

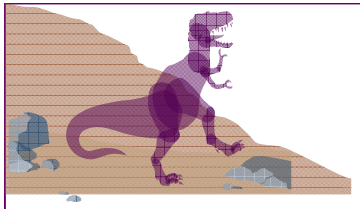
- Process Management
- Main Memory Management
- File Management
- I/O System Management
- Secondary-Storage Management
- Networking
- Protection System
- Command-Interpreter System





Operating System Services

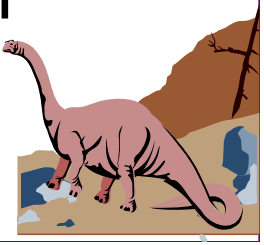
- Program execution – system capability to load a program into memory and to run it.
- I/O operations – since user programs cannot execute I/O operations directly, the operating system must provide some means to perform I/O.
- File-system manipulation – program capability to read, write, create, and delete files.
- Communications – exchange of information between processes
- Error detection – ensure correct computing by detecting errors in the CPU and memory hardware, in I/O devices, or in user programs.

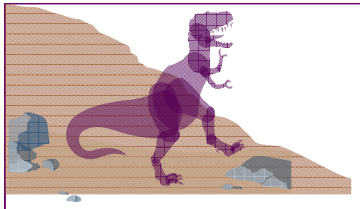


Additional Operating System Functions

Additional functions exist not for helping the user, but rather for ensuring efficient system operations.

- Resource allocation – allocating resources to multiple users or multiple jobs running at the same time.
- Accounting – keep track of and record which users use how much and what kinds of computer resources for account billing or for accumulating usage statistics.
- Protection – ensuring that all access to system resources is controlled.

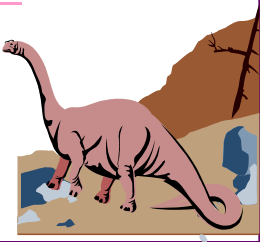


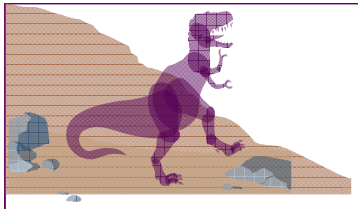


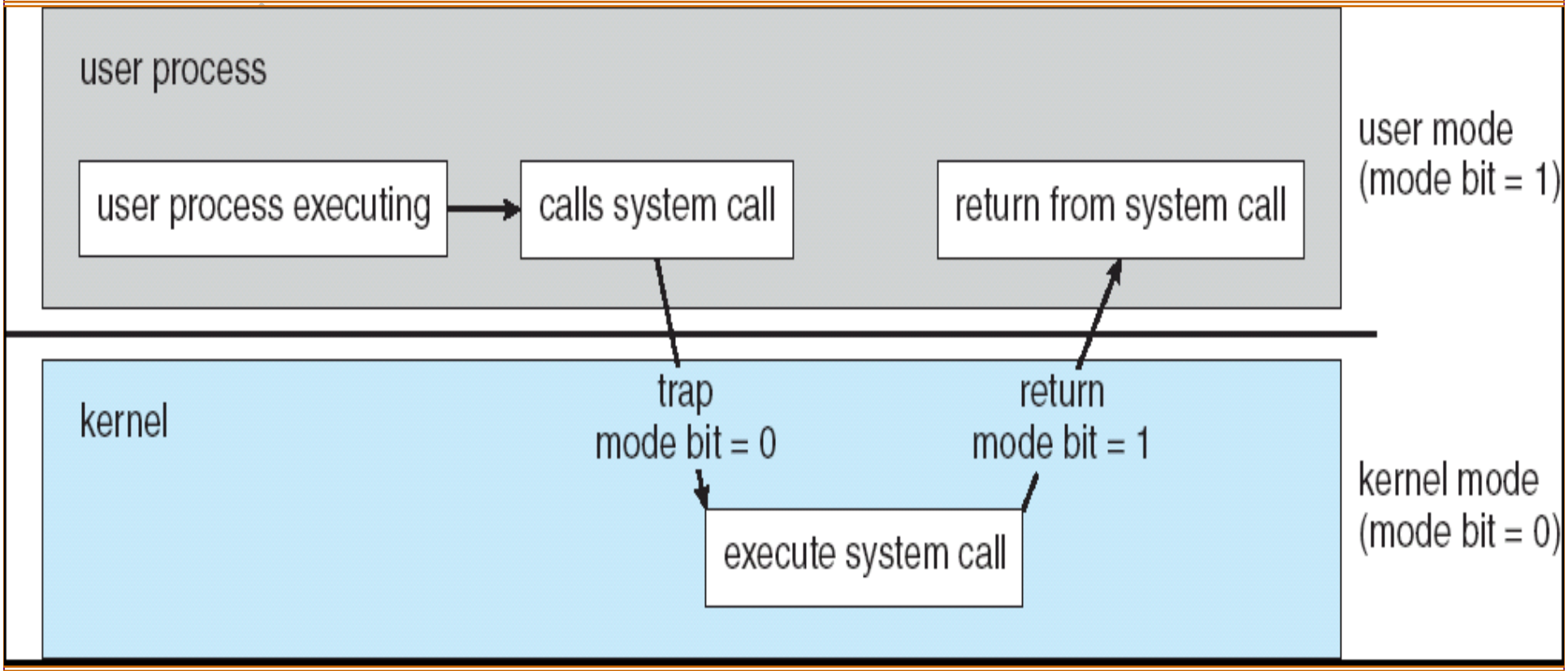
System Calls

- System calls provide the interface between a running program and the operating system.
  - Generally available as assembly-language instructions.
  - Languages defined to replace assembly language for systems programming allow system calls to be made directly (e.g., C, C++)

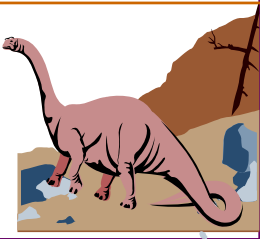
https://en.wikipedia.org/wiki/System_call#Typical_implementations
http://docs.cs.up.ac.za/programming/asm/derick_tut/syscalls.html

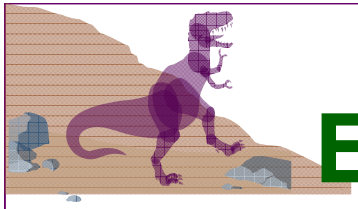






http://www.linux.it/~rubini/docs/ksys/ksys.html



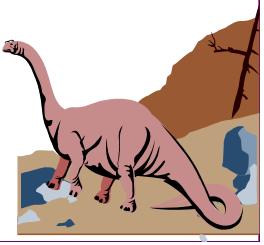


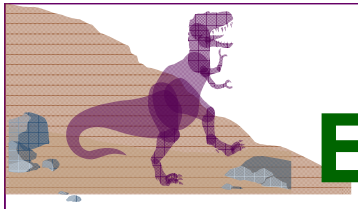
Example of System Calls (1/2)

- **cp** is the command line tool in Linux, which makes a copy of your files or directories.
  - For instance, let's say you have a file named **picture.jpg** in your working directory, and you want to make a copy of it called **picture-02.jpg**. You would run the command:

cp picture.jpg picture-02.jpg

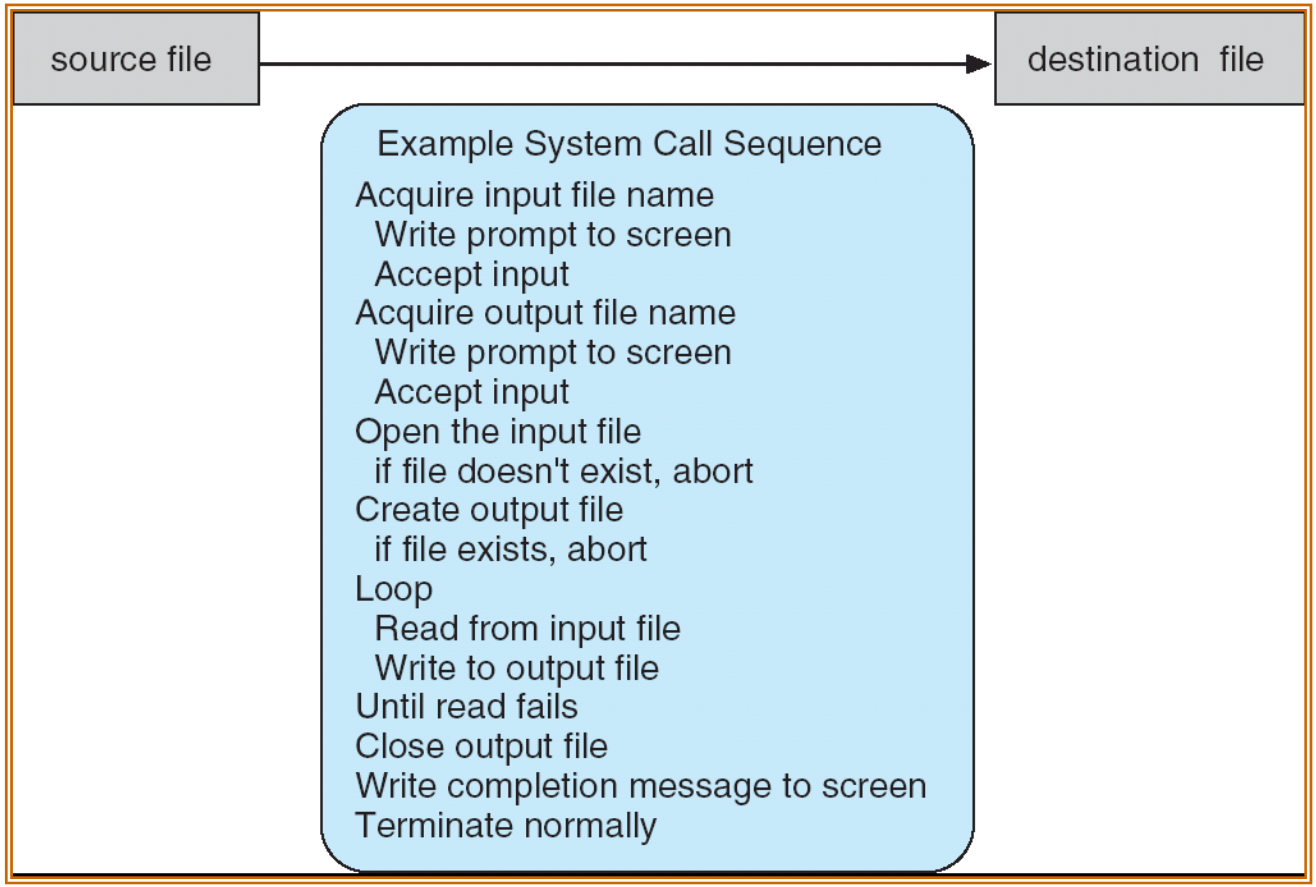
- System call sequence to copy the contents of one file to another file

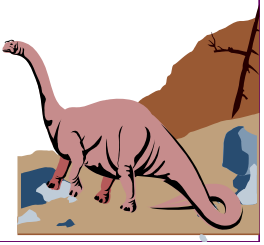


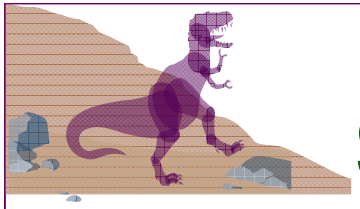


Example of System Calls (2/2)

- System call sequence to copy the contents of one file to another file

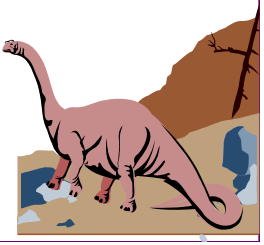


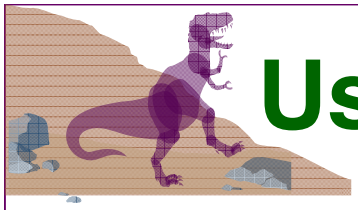




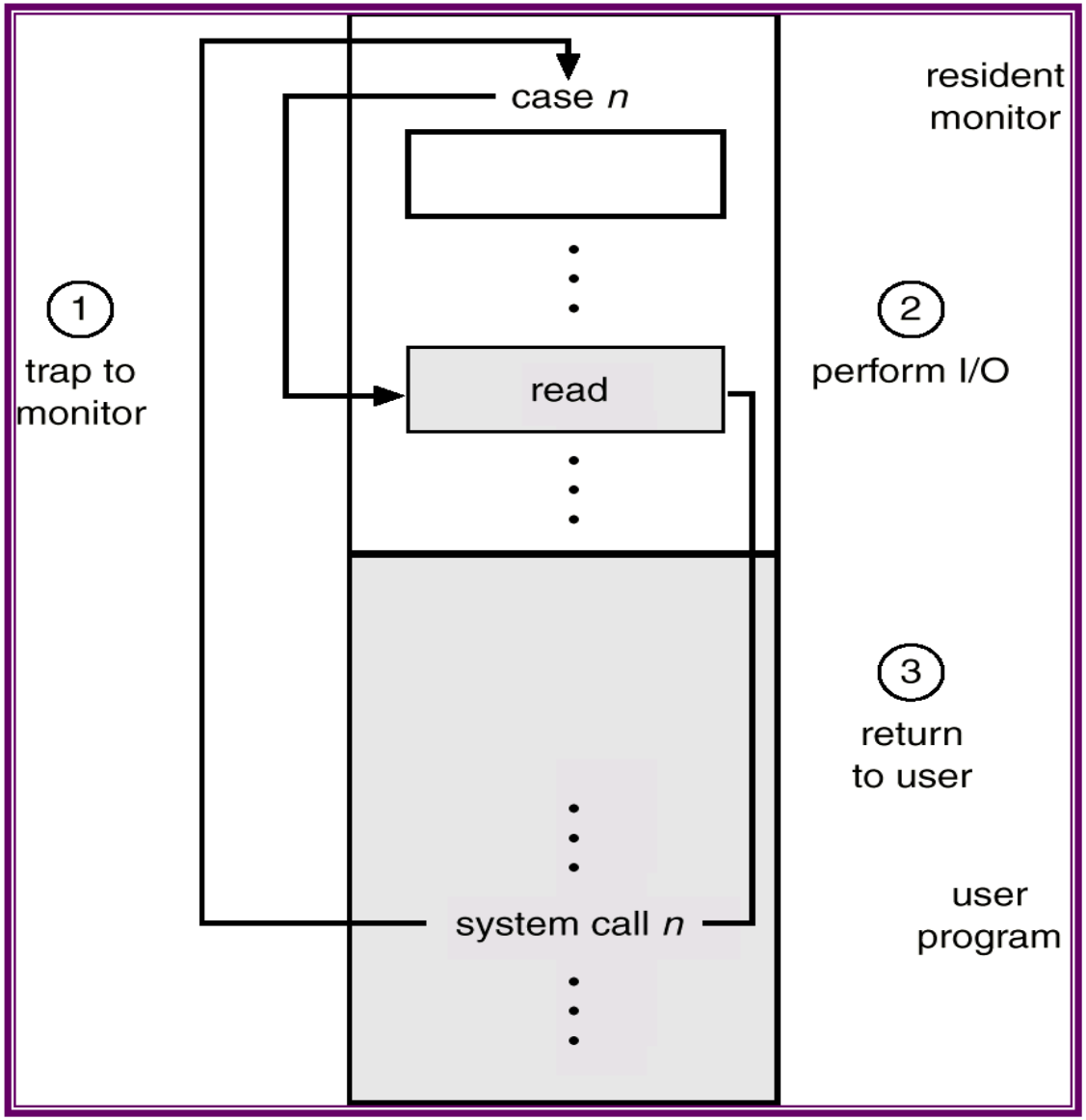
System Call Implementation

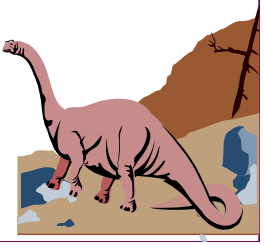
- Typically, a number associated with each system call
  - System-call interface maintains a table indexed according to these numbers
- The system call interface invokes intended system call in OS kernel and returns status of the system call and any return values

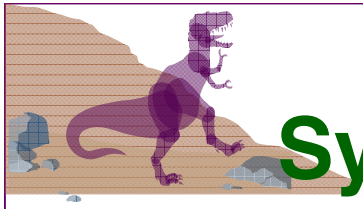




Use of A System Call to Perform I/O

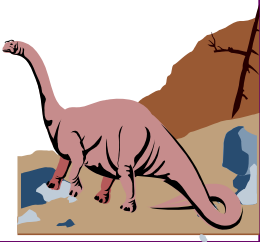


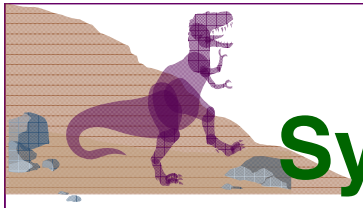




System Call Implementation (Cont.)

- The caller need know nothing about how the system call is implemented
  - Just needs to obey API and understand what OS will do as a result call
  - Most details of OS interface hidden from programmer by API
    - ✓Managed by run-time support library (set of functions built into libraries included with compiler)

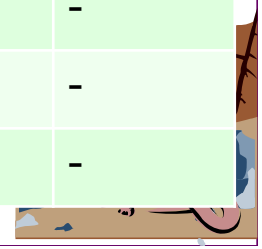


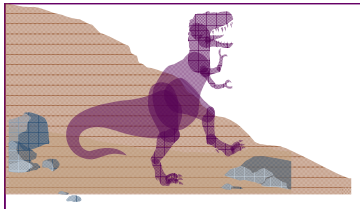


System Call Implementation (Cont.)

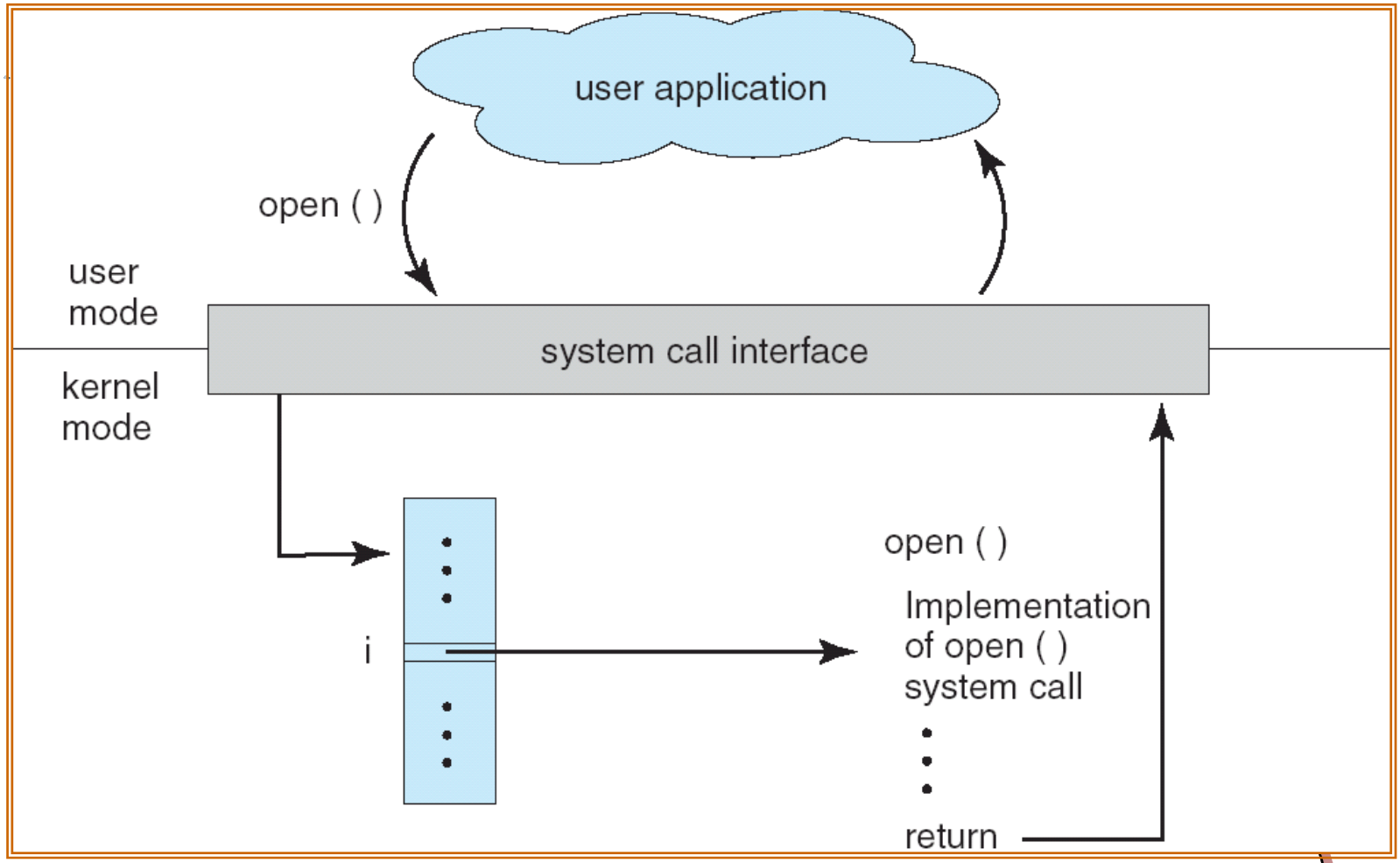
- The system calls for the Linux 2.2 kernel
  - On the left are the numbers of the system calls. This number will be put in register %eax.
  - On the right are the types of values to be put into the remaining registers before calling the trap 'int 0x80'.
  - After each syscall, an integer is returned in %eax.

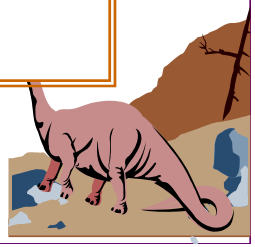
%eax	Name	Source	%ebx	%ecx	%edx	%esx	%edi
1	sys_exit	kernel/exit.c	int	-	-	-	-
2	sys_fork	arch/i386/kernel/process.c	struct pt_regs	-	-	-	-
3	sys_read	fs/read_write.c	unsigned int	char *	size_t	-	-
4	sys_write	fs/read_write.c	unsigned int	const char *	size_t	-	-
5	sys_open	fs/open.c	const char *	int	int	-	-
6	sys_close	fs/open.c	unsigned int	-	-	-	-
7	sys_waitpid	kernel/exit.c	pid_t	unsigned int *	int	-	-

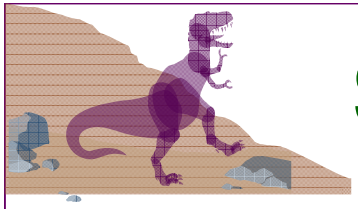




API – System Call – OS Relationship

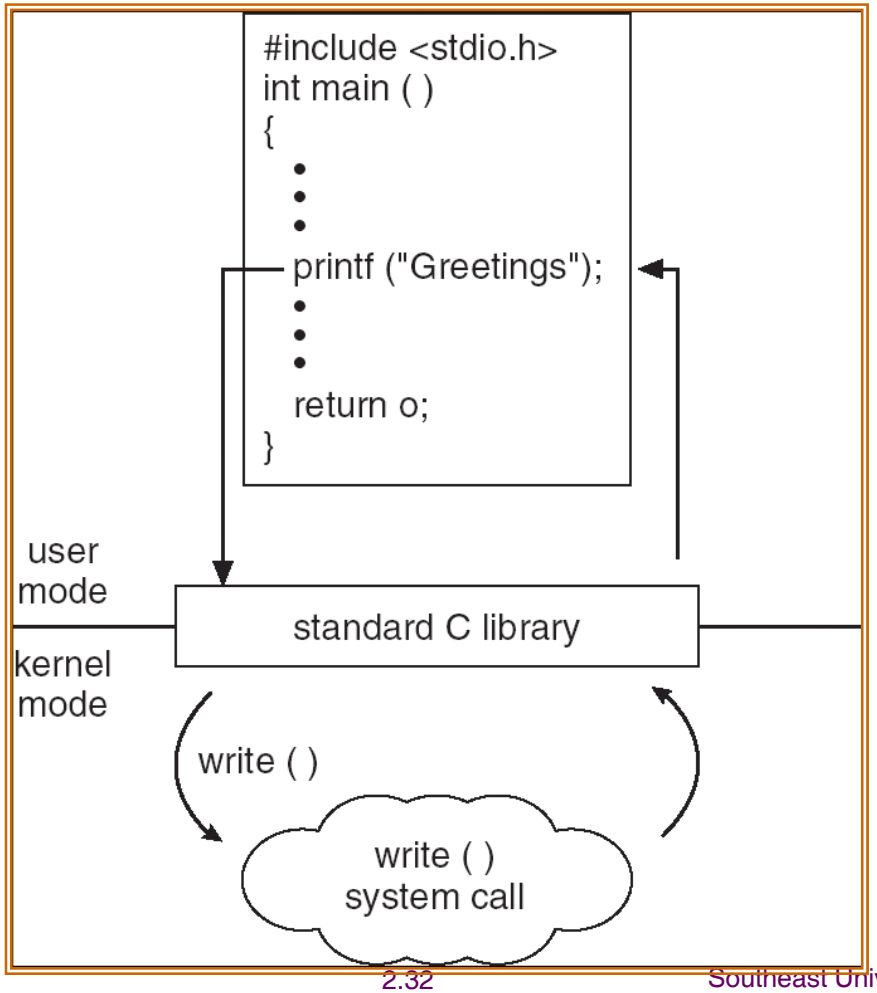


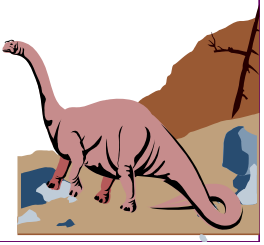


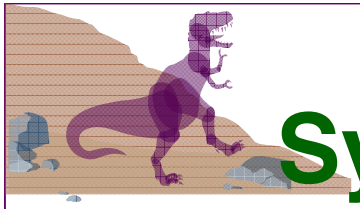


Standard C Library Example

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

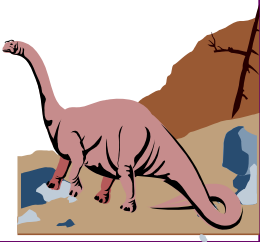


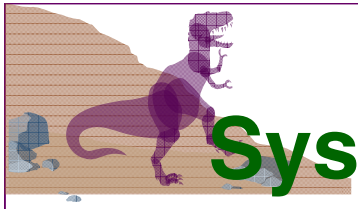




System Call Parameter Passing

- Often, more information is required than simply identity of desired system call
  - Exact type and amount of information vary according to OS and call
- Three general methods used to pass parameters to the OS

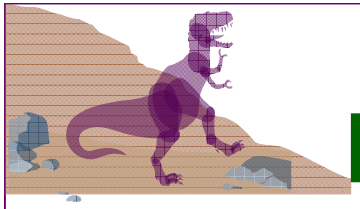




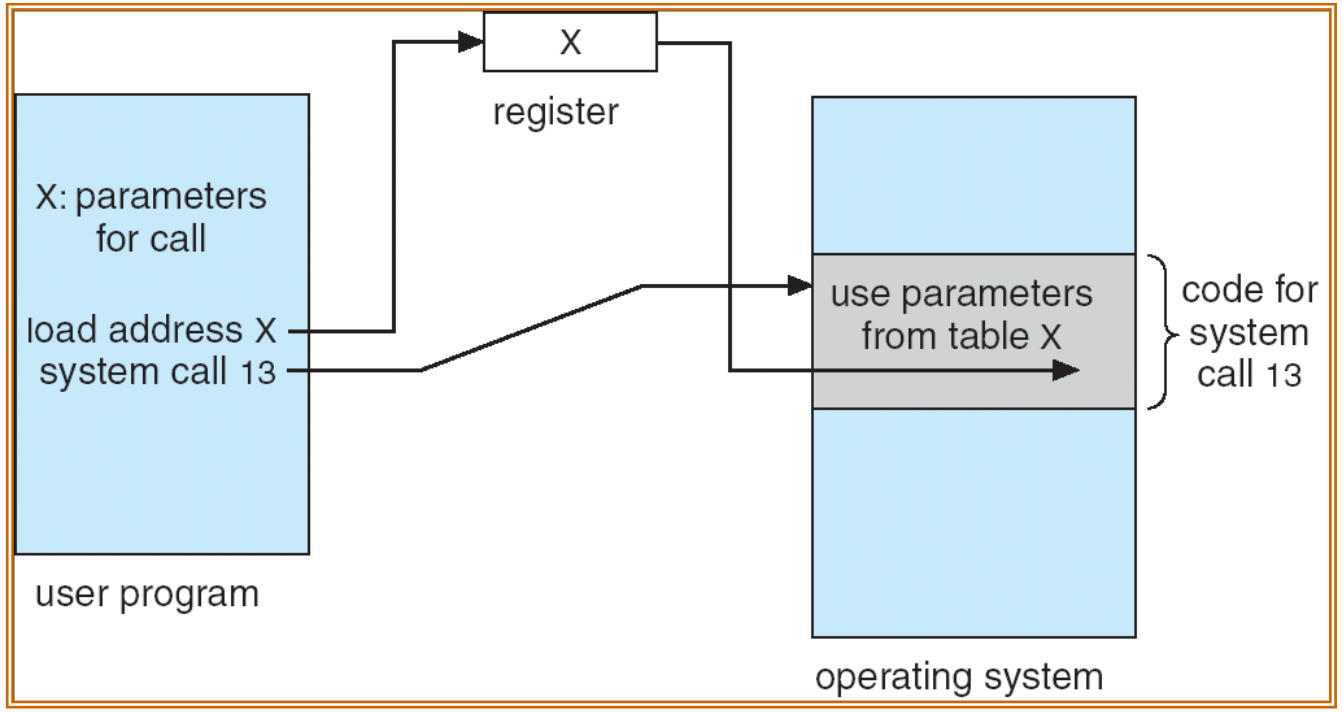
System Call Parameter Passing (Cont.)

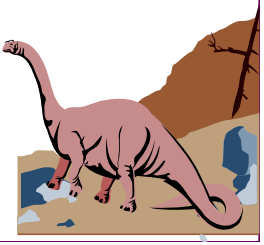
- Simplest: pass the parameters in *registers*
  - ✓ In some cases, may be more parameters than registers
- Parameters stored in a *block,* or table, in memory, and address of block passed as a parameter in a register
  - ✓ This approach taken by Linux and Solaris
- Parameters placed, or *pushed,* onto the *stack* by the program and *popped* off the stack by the operating system

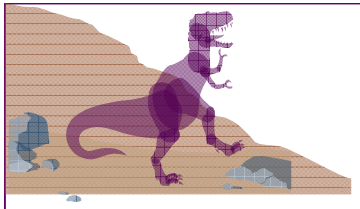
- Block and stack methods do not limit the number or length of parameters being passed



Parameter Passing via Table

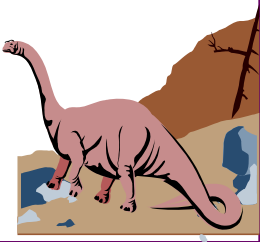


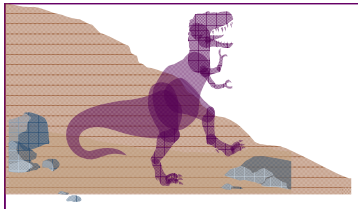




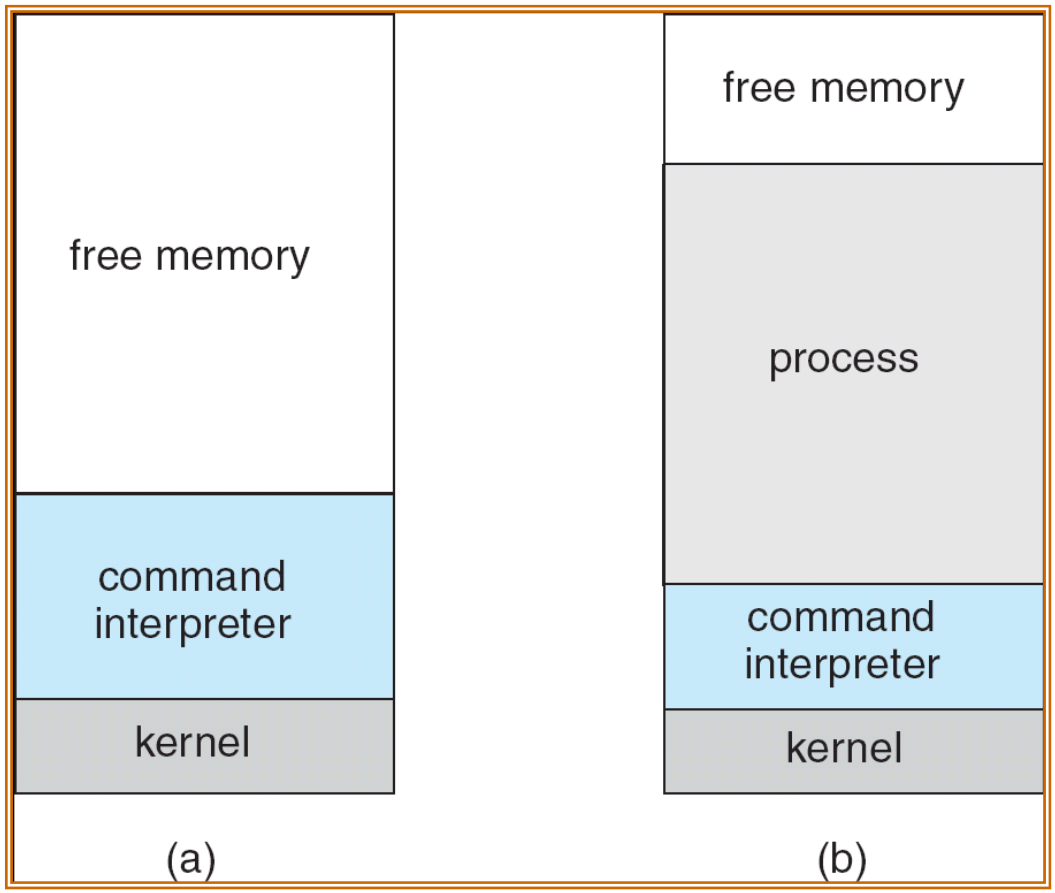
Types of System Calls

- Process control
- File management
- Device management
- Information maintenance
- Communications

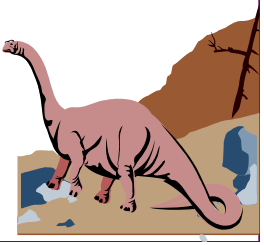


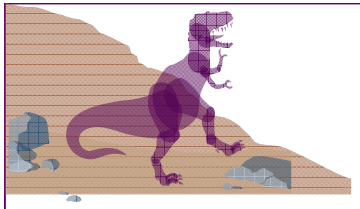


MS-DOS execution (Single Program)

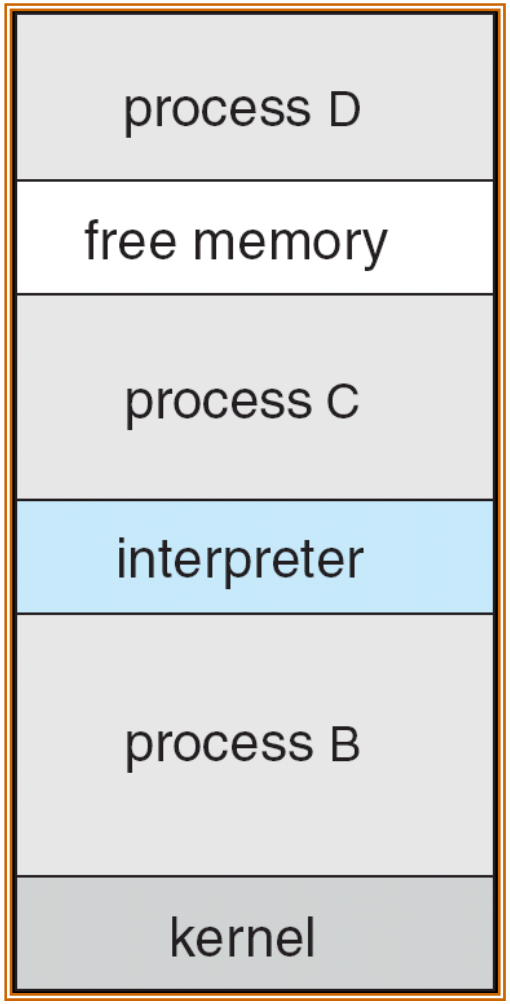


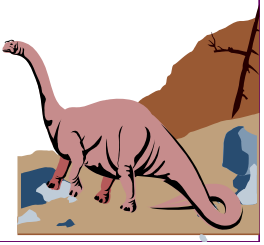
(a) At system startup (b) running a program

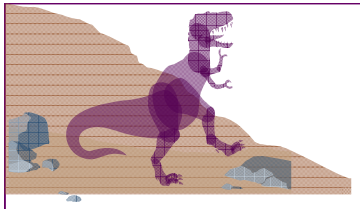




FreeBSD Running Multiple Programs

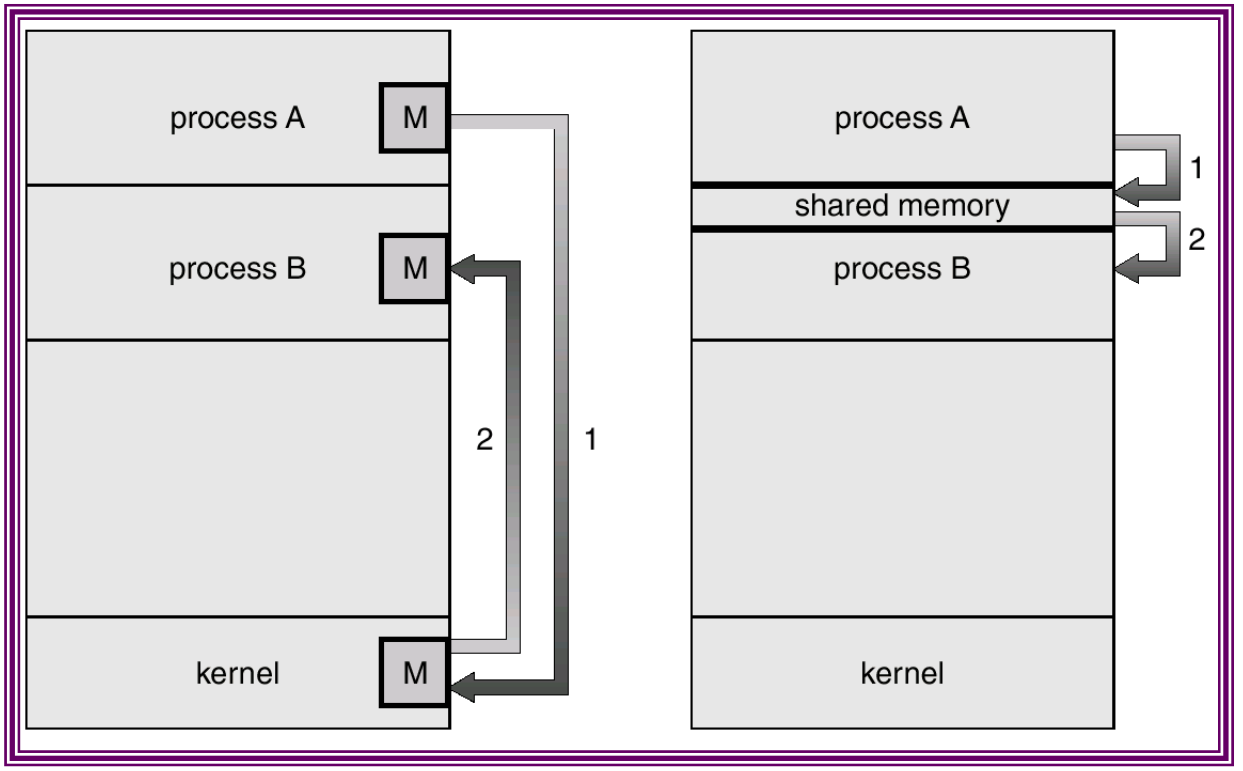


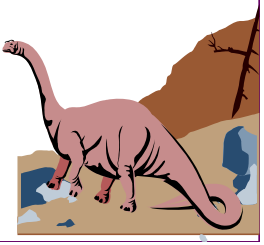


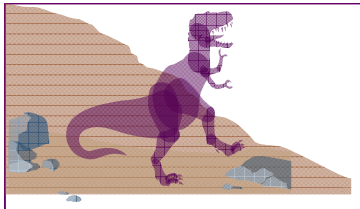


Communication Models

- Communication may take place using either message passing or shared memory.

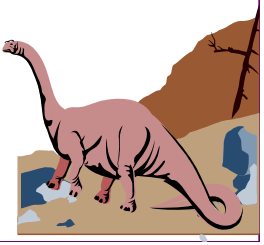


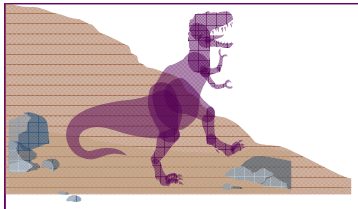




System Programs

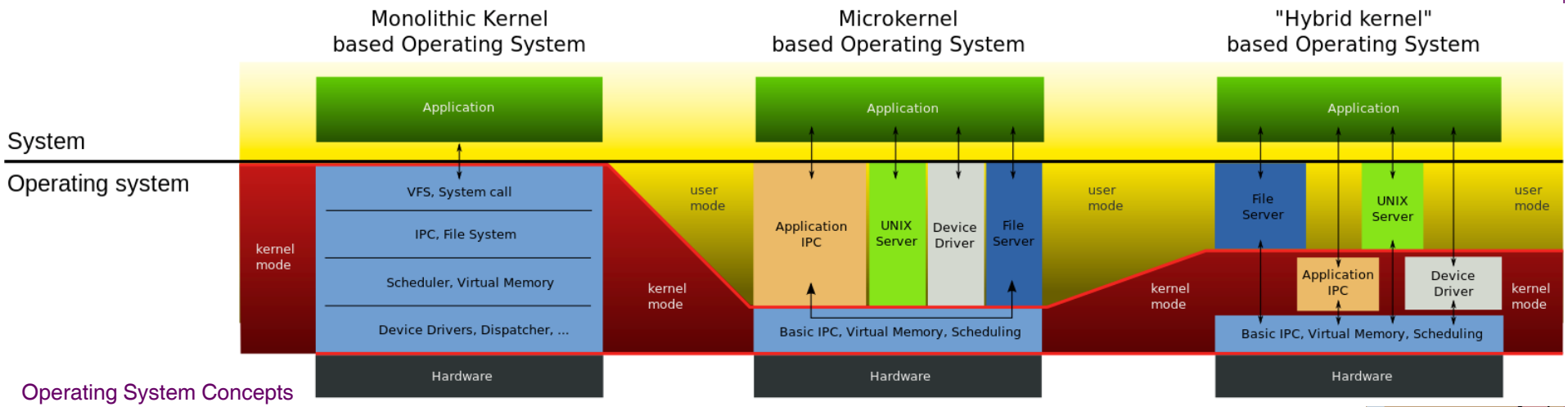
- System programs provide a convenient environment for program development and execution. The can be divided into:
  - File manipulation
  - Status information
  - File modification
  - Programming language support
  - Program loading and execution
  - Communications

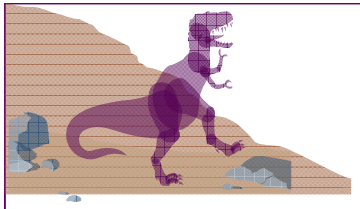




What is OS Structure?

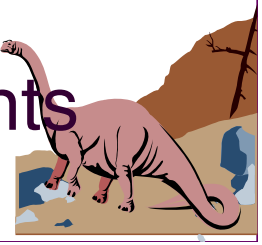
- The way the OS software is organized with respect to the applications that it serves and the underlying hardware that it manages
  - Monolithic kernel
  - Microkernel system structure
  - Hybrid kernel, and Monolithic kernel with modules

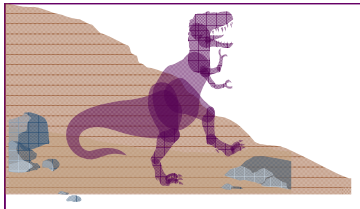




Goal of OS Structure

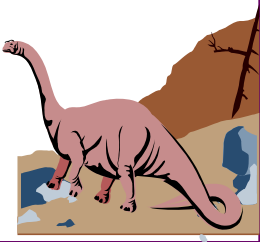
- Protection: within and across users + the OS itself
- Performance: time taken to perform the services
- Flexibility: Extensibility => Not one size fits all
- Scalability: performance↑ if hardware resources↑
- Agility: adapting to changes in application needs and/or resource availability
- Responsiveness: reaching to the external events

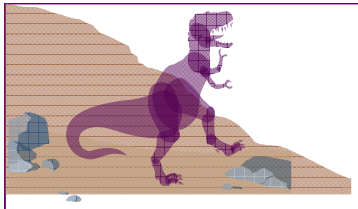




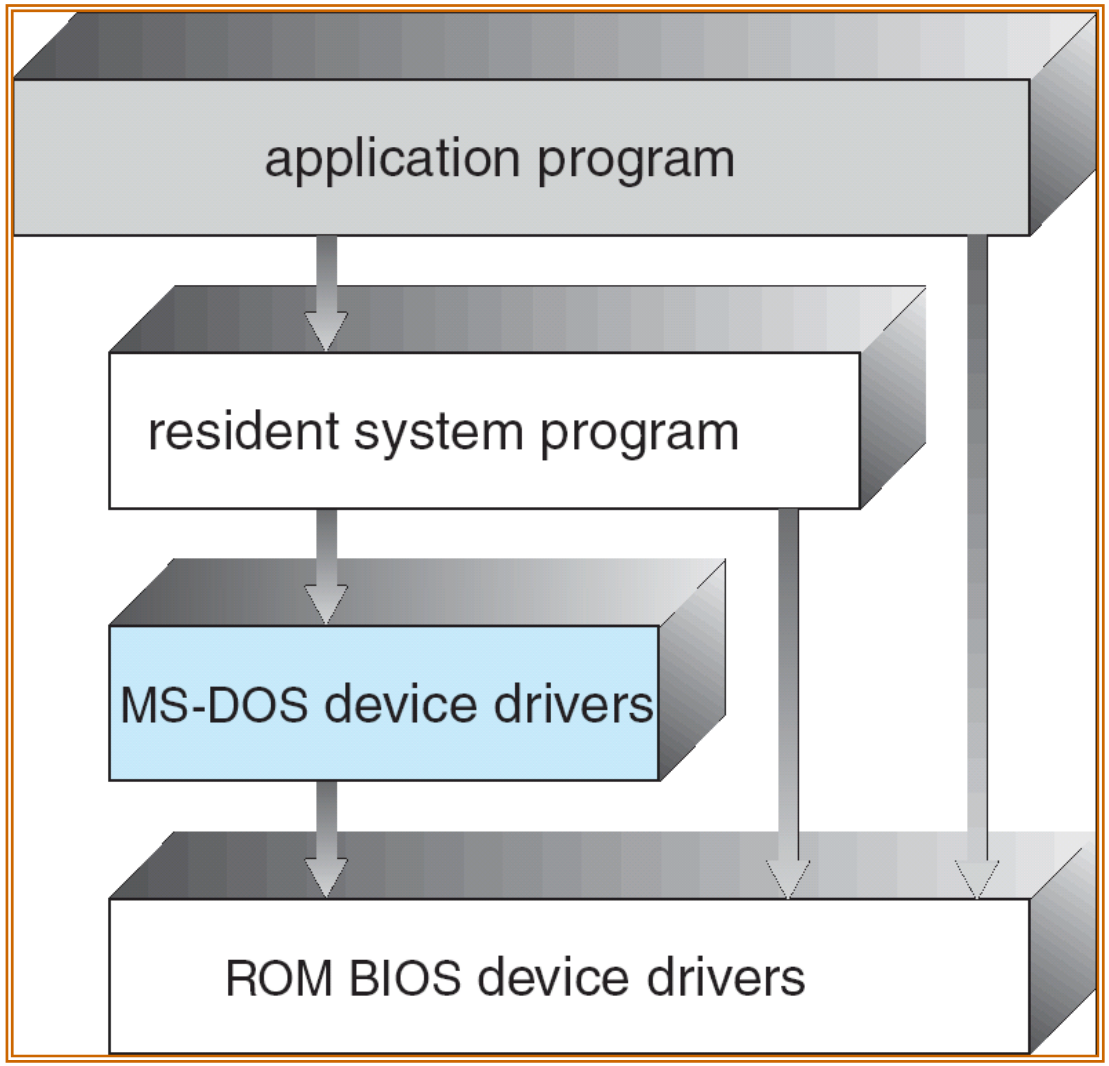
DOS-like Structure

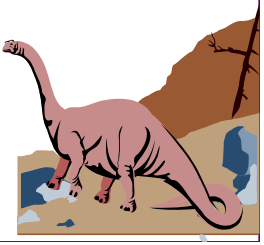
- MS-DOS – written to provide the most functionality in the least space
  - Performance: Access to system services is like a procedure call
  - Although MS-DOS has some structure, its interfaces and levels of functionality are not well separated
  - Bad Protection: an error of application can corrupt the OS
  - Not divided into modules

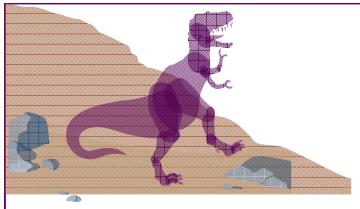




MS-DOS Layer Structure







DOS-like Structure (cont.)

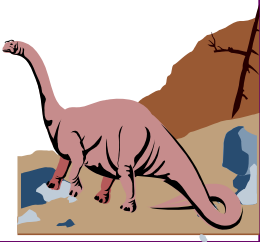
App 1 App 2 …… App n

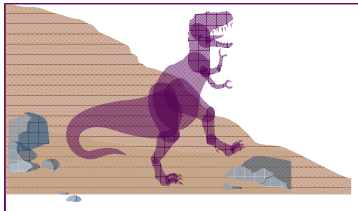
OS
Services and
Device Drivers

Hardware

No protection between the applications and the OS, which are in the same address space

Managed by the OS



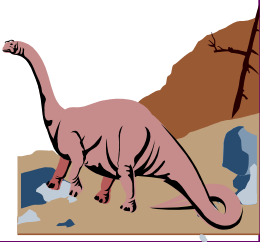


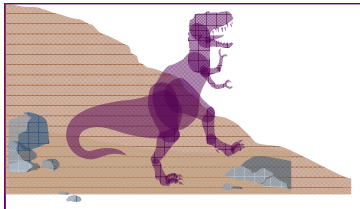
Monolithic Structure

App 1 App 2 …… App n ⇨ Each App in its own hardware address space

OS Services and Device Drivers ⇨ OS in its own hardware address space

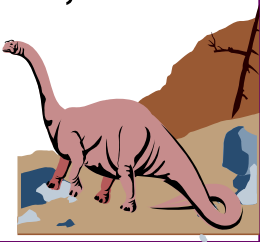
Hardware ⇨ Managed by the OS

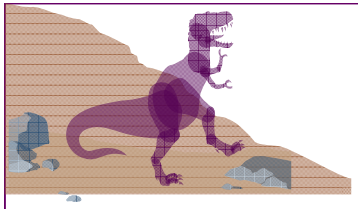




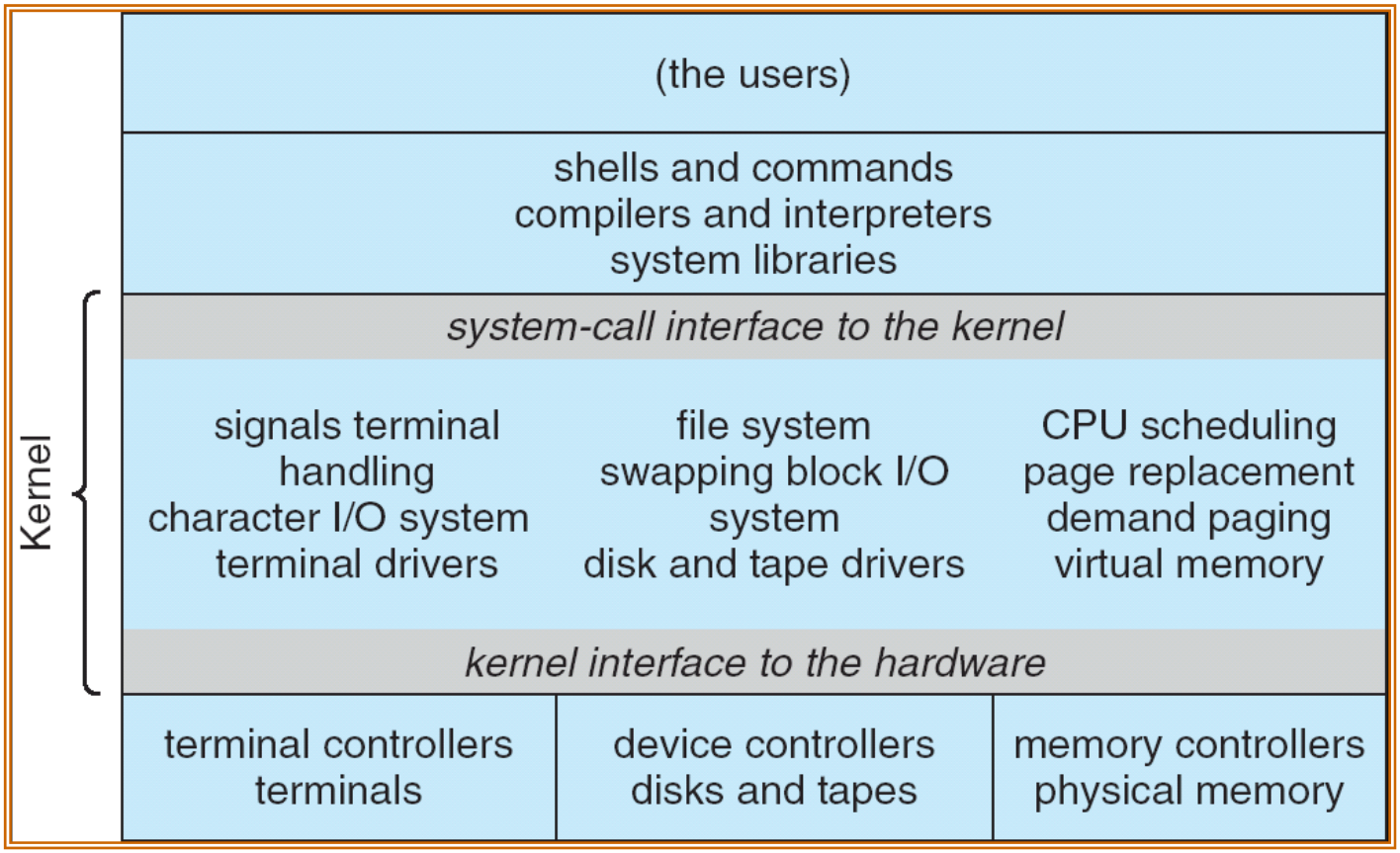
Monolithic Architecture Example: UNIX

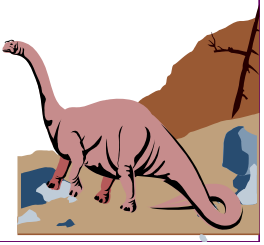
- UNIX – limited by hardware functionality, the original UNIX operating system had limited structuring. The UNIX OS consists of two separable parts
  - Systems programs
  - The kernel
    - ✓ Consists of everything below the system-call interface and above the physical hardware
    - ✓ Provides the file system, CPU scheduling, memory management, and other operating-system functions; a large number of functions for one level

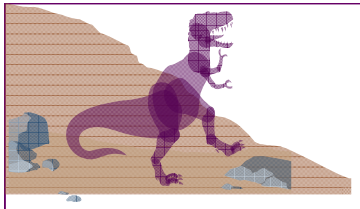




UNIX System Structure

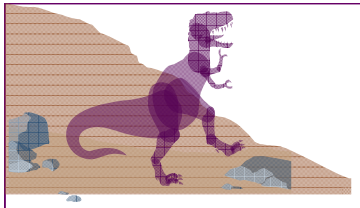






Gain and Loss of Monolithic Structure

- Loss of Protection in DOS-like Structure
  - Unacceptable for a general-purpose OS
- Monolithic Structure
  - Reduce performance loss by consolidation
- But …
  - Monolithic structure => no customization
- Need for customization
  - Applications of video game and computing prime numbers may have different needs for CPU scheduling, file access or memory management



Microkernel OS Structure

Communication takes place between user modules using IPC-based message passing

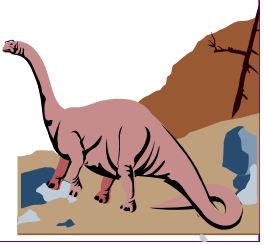
App 1 App 2 …… App n ⇨ Each App in its own hardware address space

OS Services

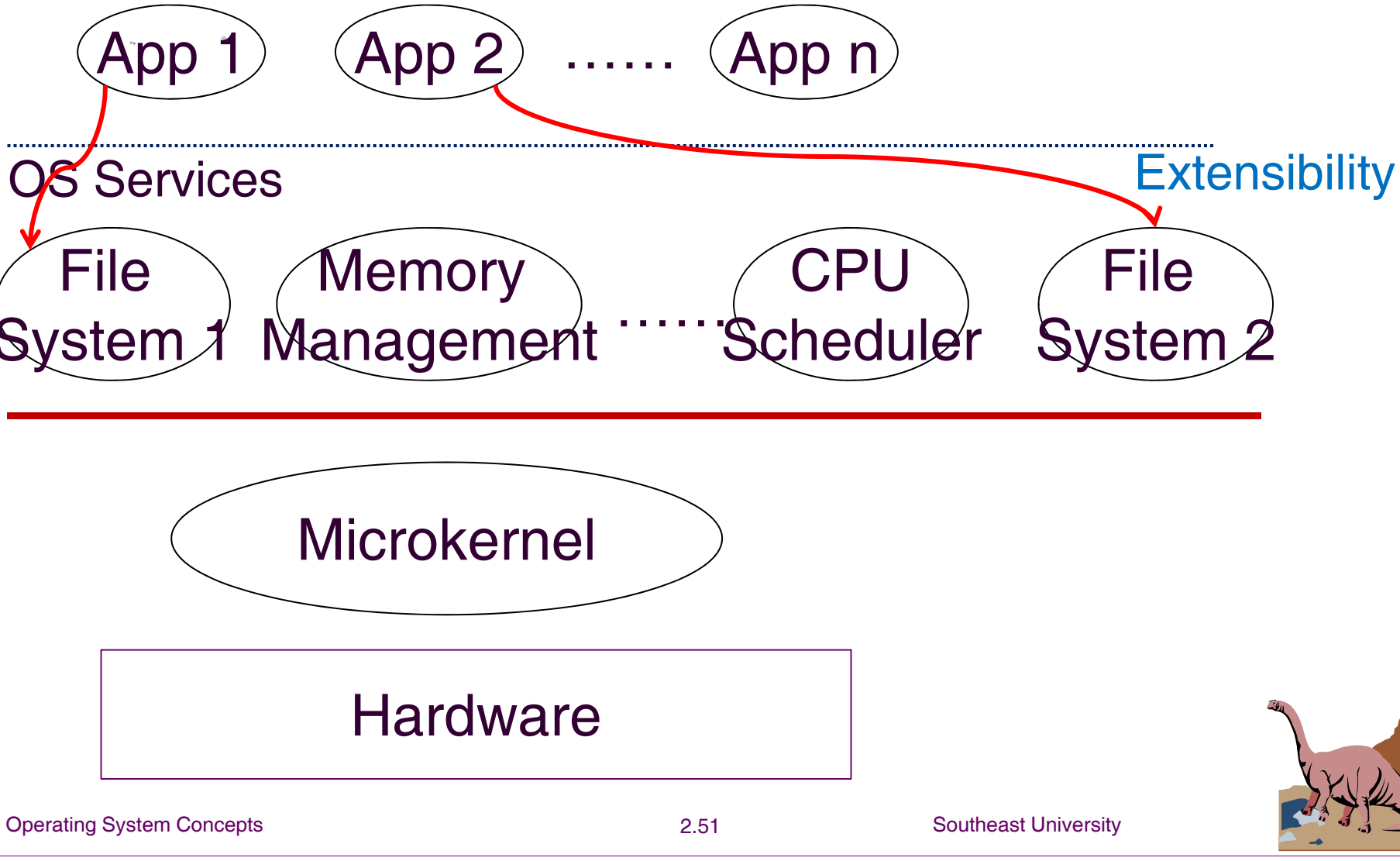
File System Memory Management …… CPU Scheduler ⇨ Each service in its own address space

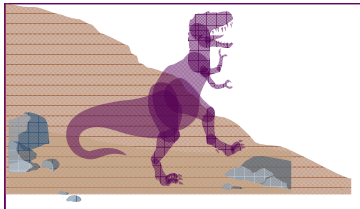
Microkernel ⇨ Simple abstraction
- Address space
- IPC

Hardware



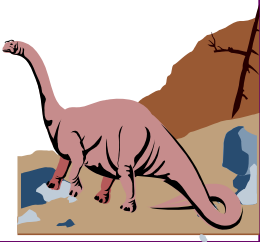
Advantage of Microkernel-based Design

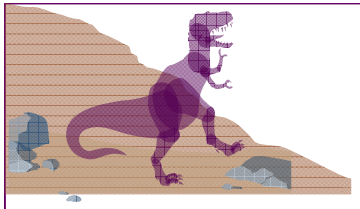




Pros and Cons of Microkernel System Structure

- Benefits:
  - Easier to extend a microkernel
  - Easier to port the operating system to new architectures
  - More reliable (less code is running in kernel mode)
  - More secure
- Detriments:
  - Performance overhead of user space to kernel space communication

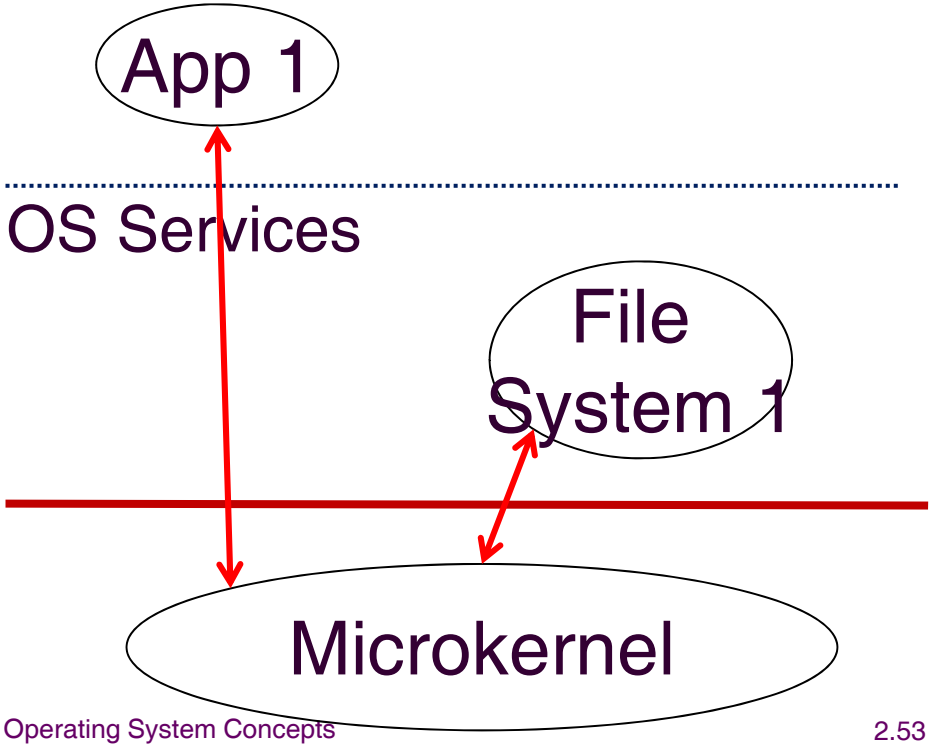




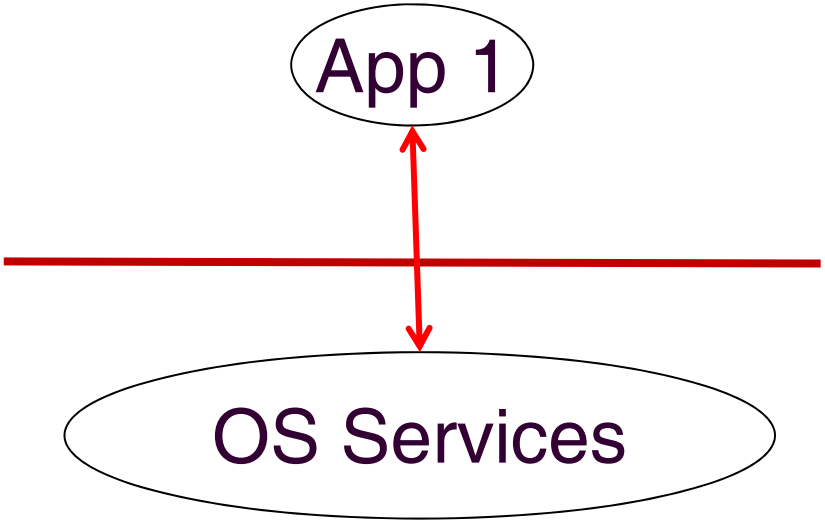
Why Performance Loss

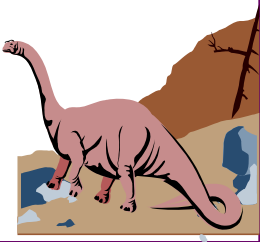
- Border Crossing
  - Change in locality, e.g., memory address space
  - Copy data between user and system spaces

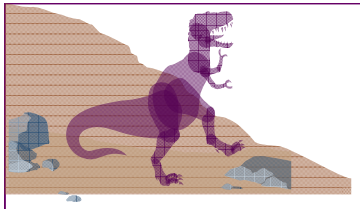
Microkernel



Monolithic



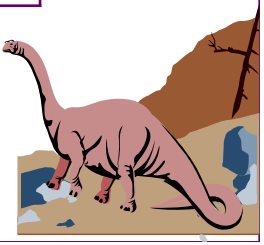


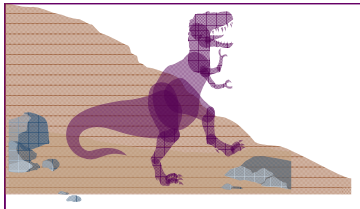


Question

- Based on discussion thus far ….

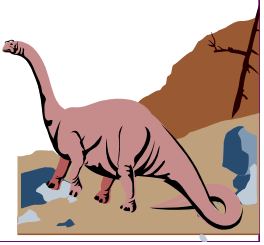
Feature	DOS-like OS	Monolithic OS	Microkernel OS
Extensibility	√		√
Protection		√	√
Performance	√	√	



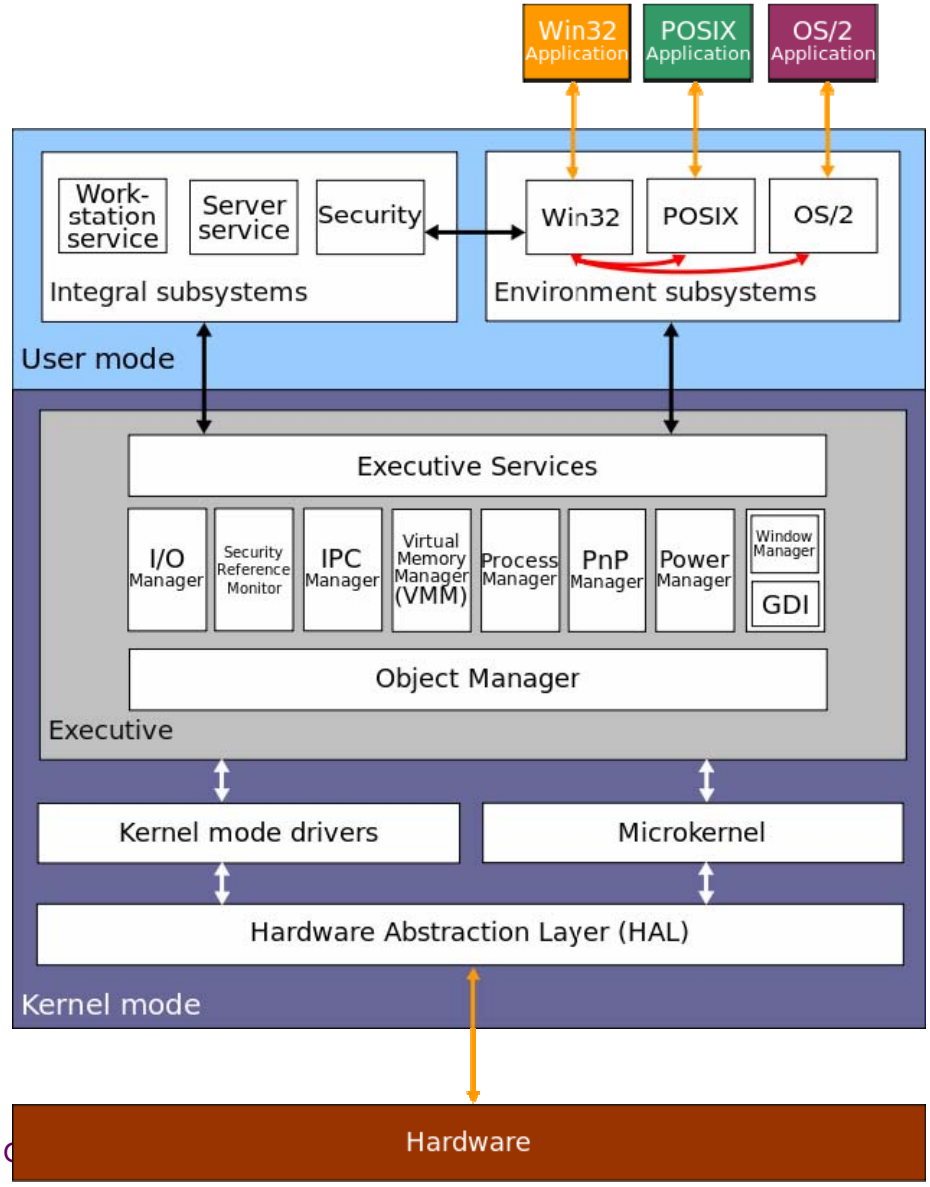


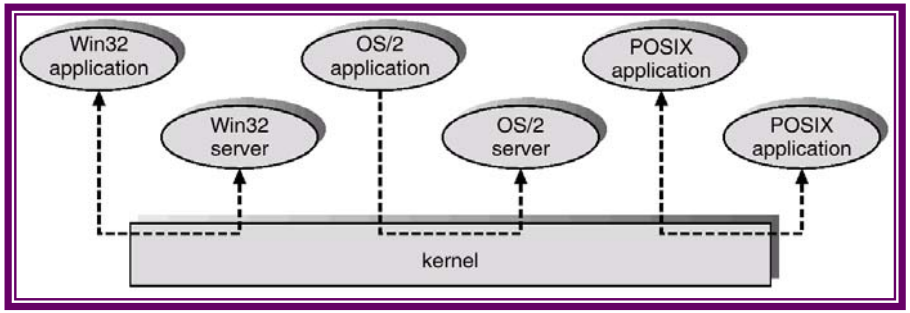
Hybrid Kernel Approach

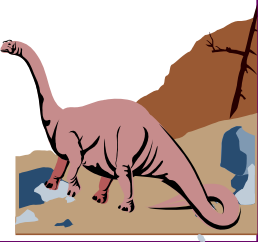
- The idea behind a hybrid kernel is to have a kernel structure similar to that of a microkernel, but to implement that structure in the manner of a monolithic kernel.
- In contrast to a microkernel, all (or nearly all) operating system services in a hybrid kernel are still in kernel space.

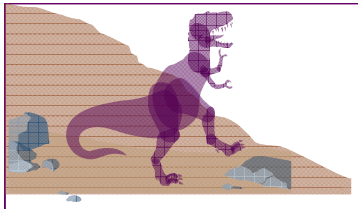


Hybrid Kernel Example: Windows NT Client-Server Structure

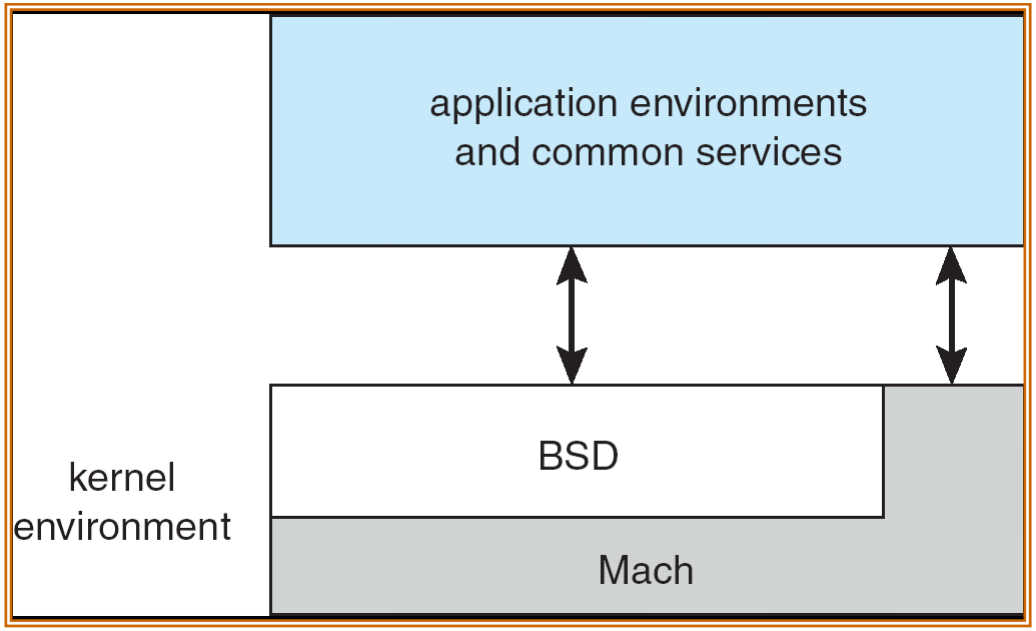


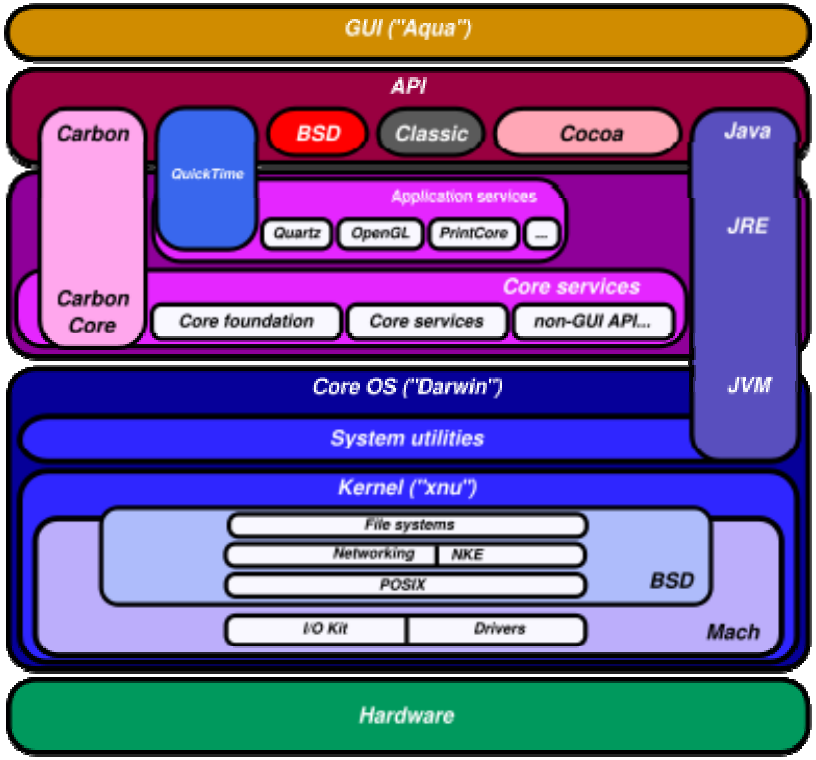


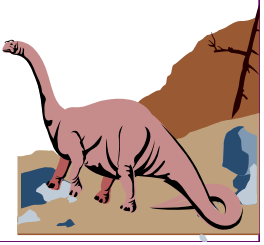


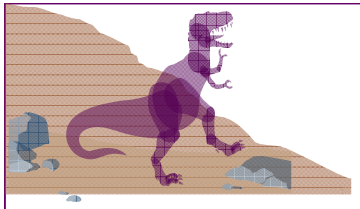


Hybrid Kernel Example: Mac OS X Structure

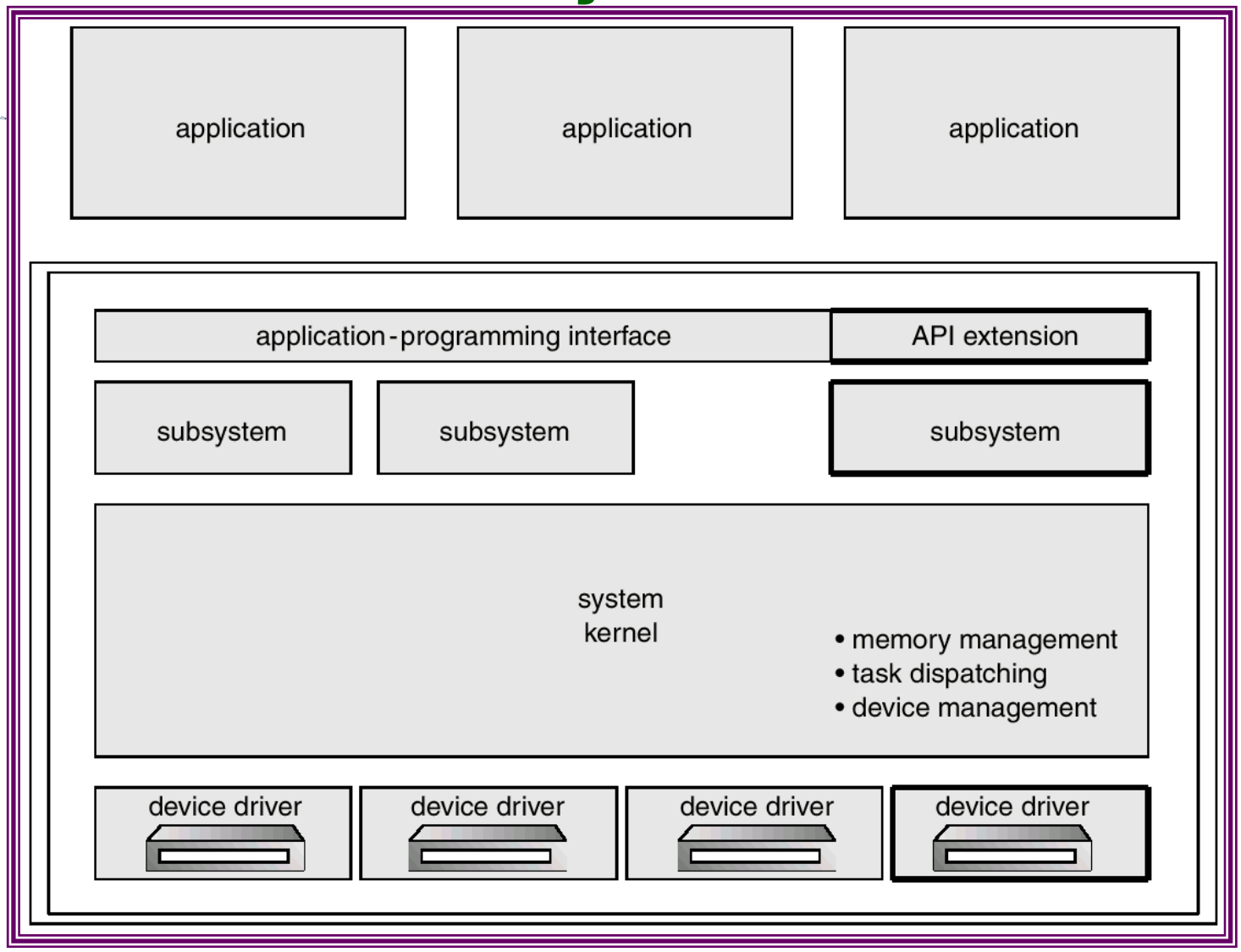




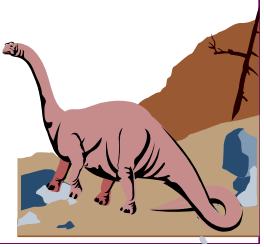


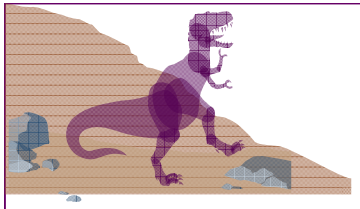


Hybrid Kernel Example: OS/2 Layer Structure



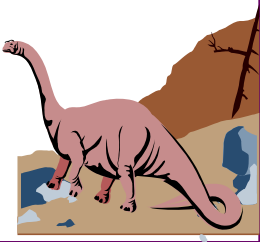
https://en.wikipedia.org/wiki/OS/2

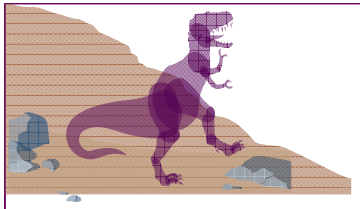




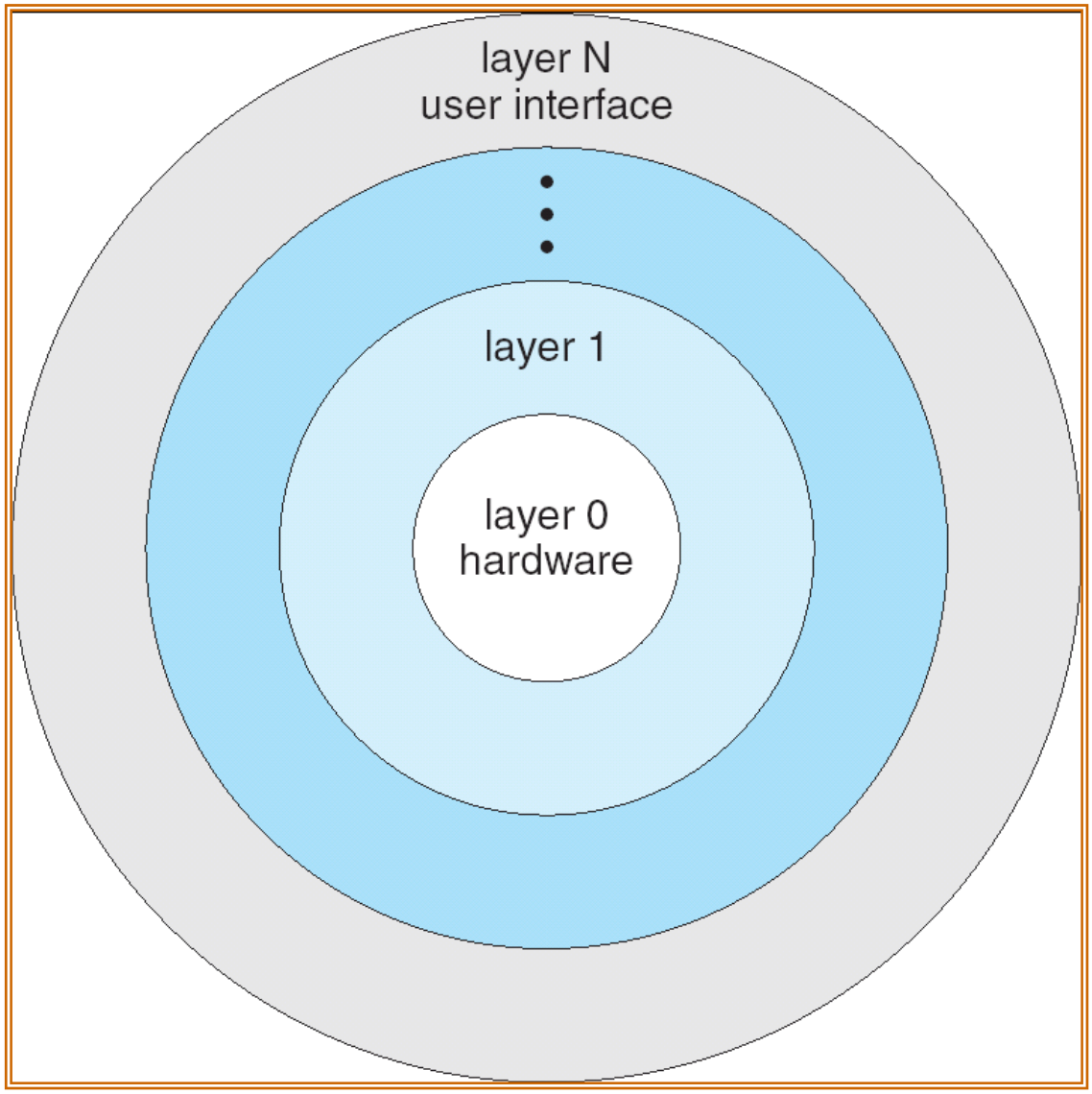
Design Philosophy: Layered Approach

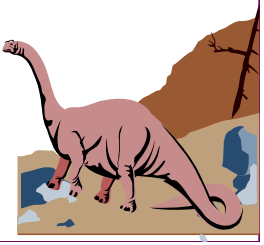
- The operating system is divided into a number of layers (levels), each built on top of lower layers. The bottom layer (layer 0), is the hardware; the highest (layer N) is the user interface.
- With modularity, layers are selected such that each uses functions (operations) and services of only lower-level layers

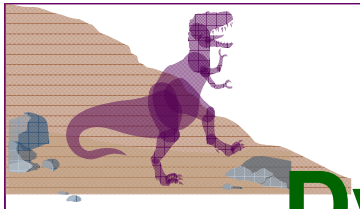




Layered Operating System

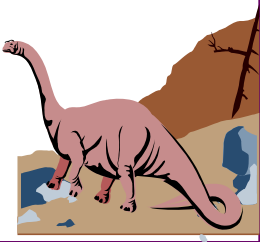


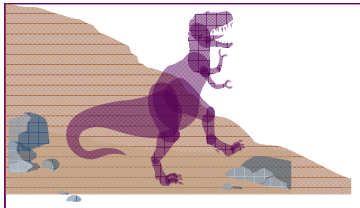




Monolithic Kernel with Dynamically Loadable Modules

- Most modern operating systems implement kernel modules
  - Uses object-oriented approach
  - Each core component is separate
  - Each talks to the others over known interfaces
  - Each is loadable as needed within the kernel
- Overall, similar to layers but with more flexible

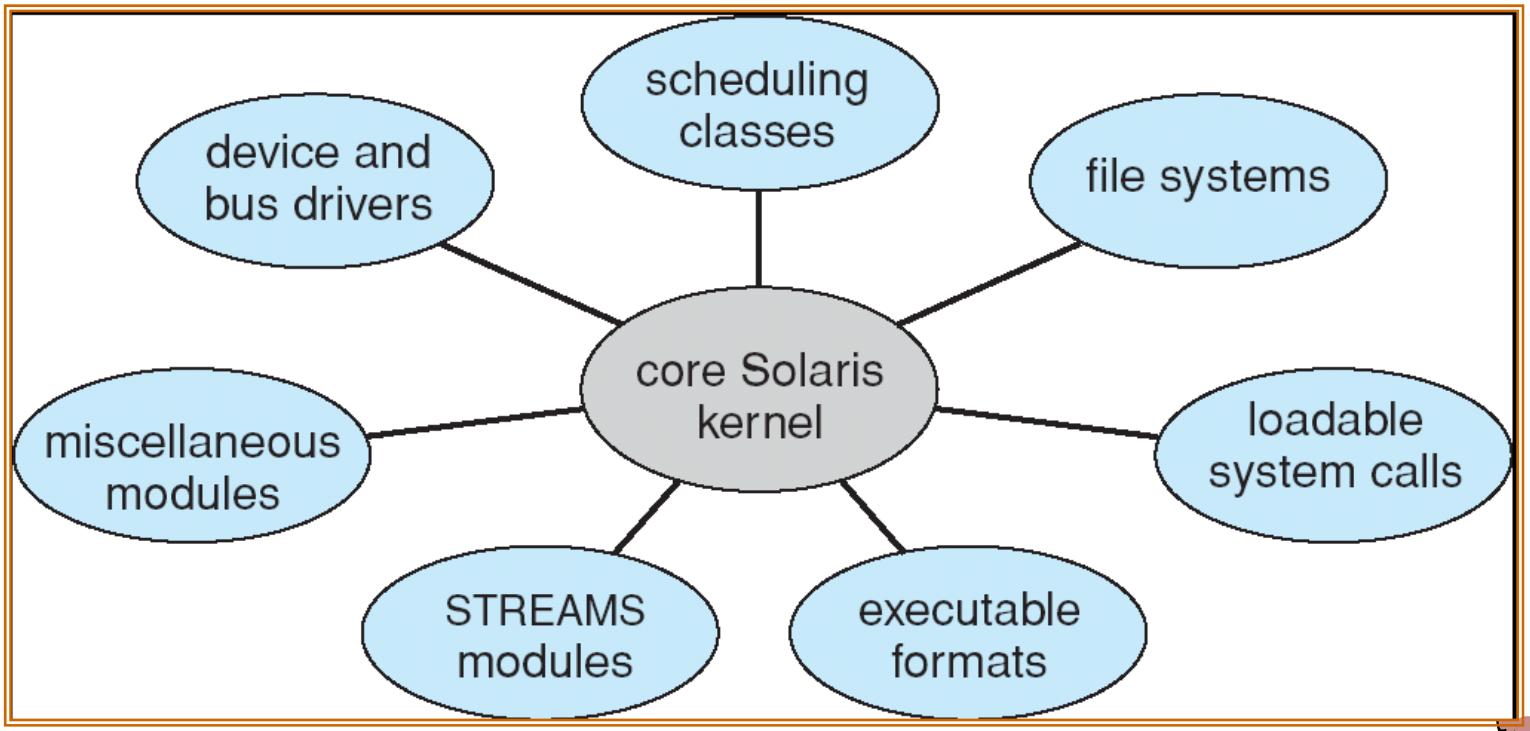




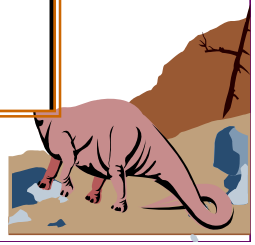
Solaris Modular Approach

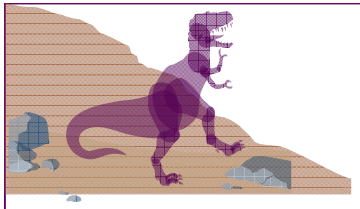
◆Solaris is a Unix operating system originally developed by Sun Microsystems.

◆Kernel type is Monolithic with dynamically loadable modules

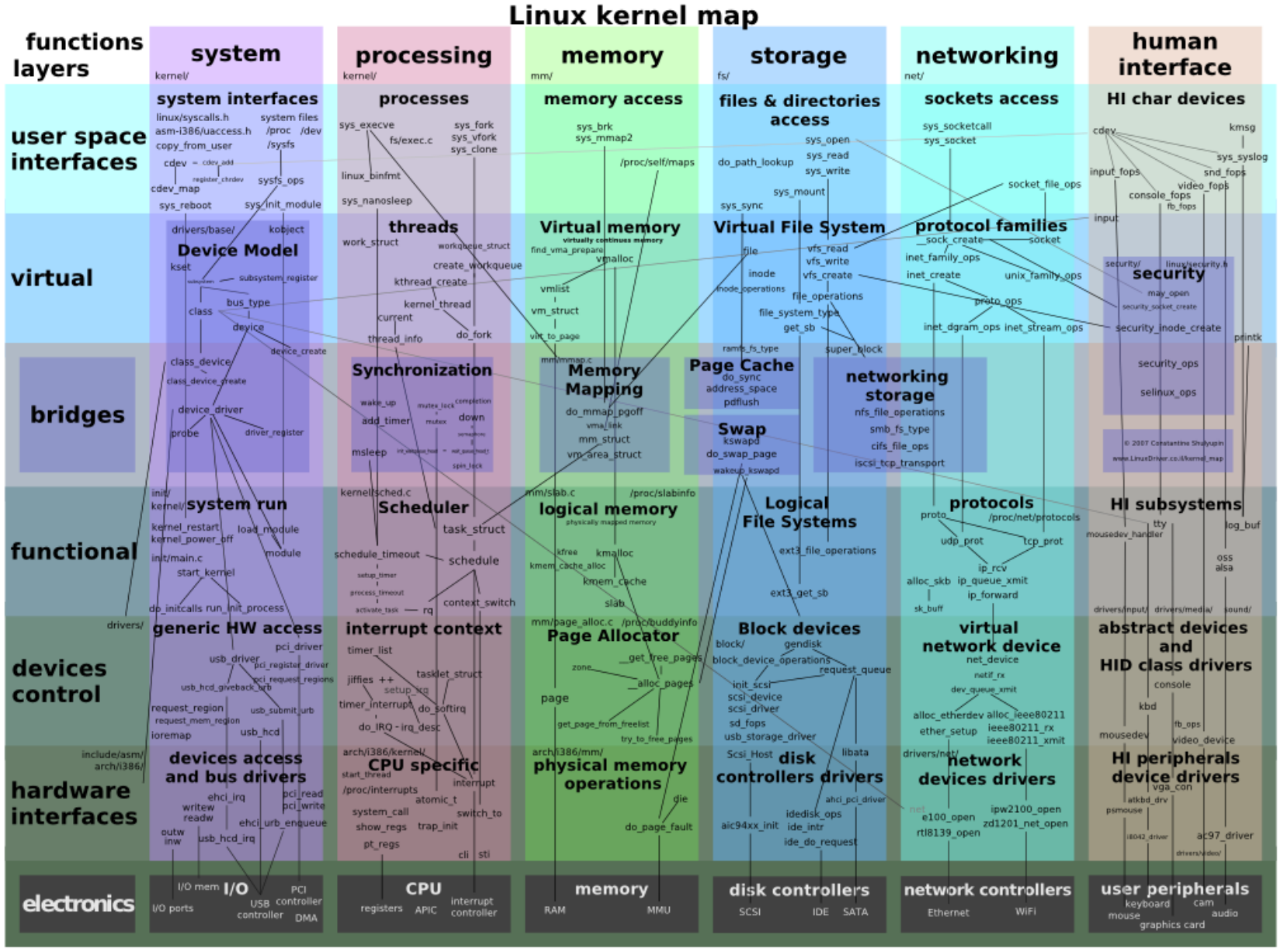


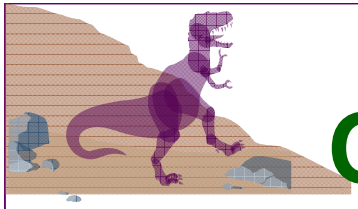
https://en.wikipedia.org/wiki/Solaris_(operating_system)





Linux is also a modular monolithic kernel

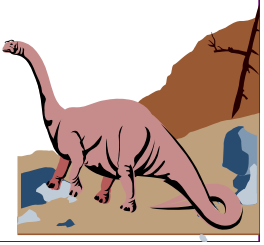


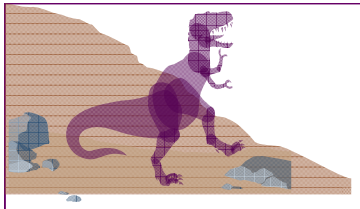


Question about Virtualization

- What comes to your mind when you hear about the word “virtualization”
  - Memory Systems
  - Data Centers
  - Java Virtual Machine
  - Virtual Box
  - IBM VM/370
  - Google Glass
  - Cloud Computing
  - Dalvik JVM
  - VMWare Workstation
  - The movie “Inception”

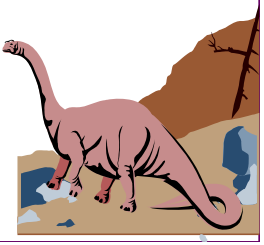
https://en.wikipedia.org/wiki/Virtualization
https://en.wikipedia.org/wiki/VirtualBox

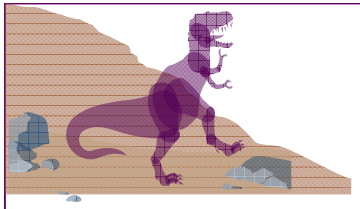




Concept of Virtualization

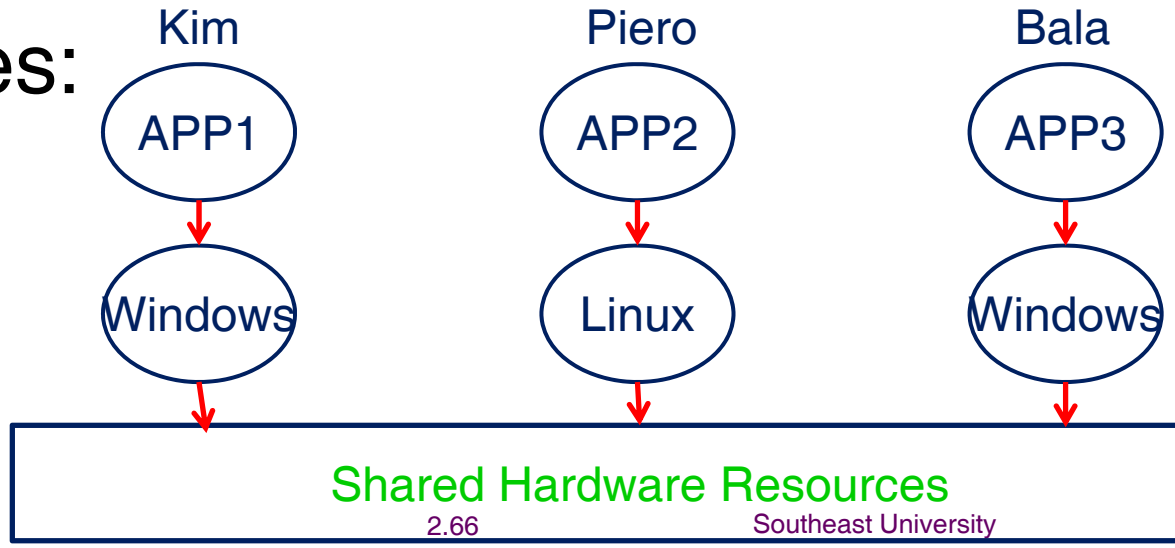
- Virtualization refers to the act of creating a virtual (rather than actual) version of something, including
  - hardware platform virtualization,
  - memory virtualization,
  - CPU virtualization,
  - storage virtualization,
  - network virtualization, etc.

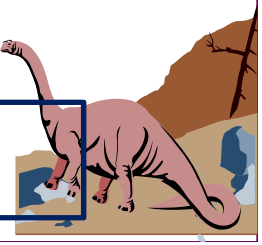


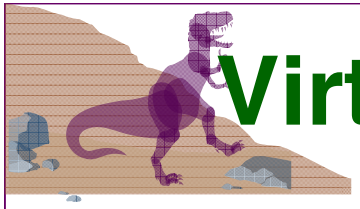


Virtual Machines

- The operating system creates the illusion of multiple processes, each executing on its own processor with its own (virtual) memory
- A *virtual machine* system provides an interface *identical* to the underlying bare hardware, creating an illusion of multiple (virtual) machines
- User Drives:

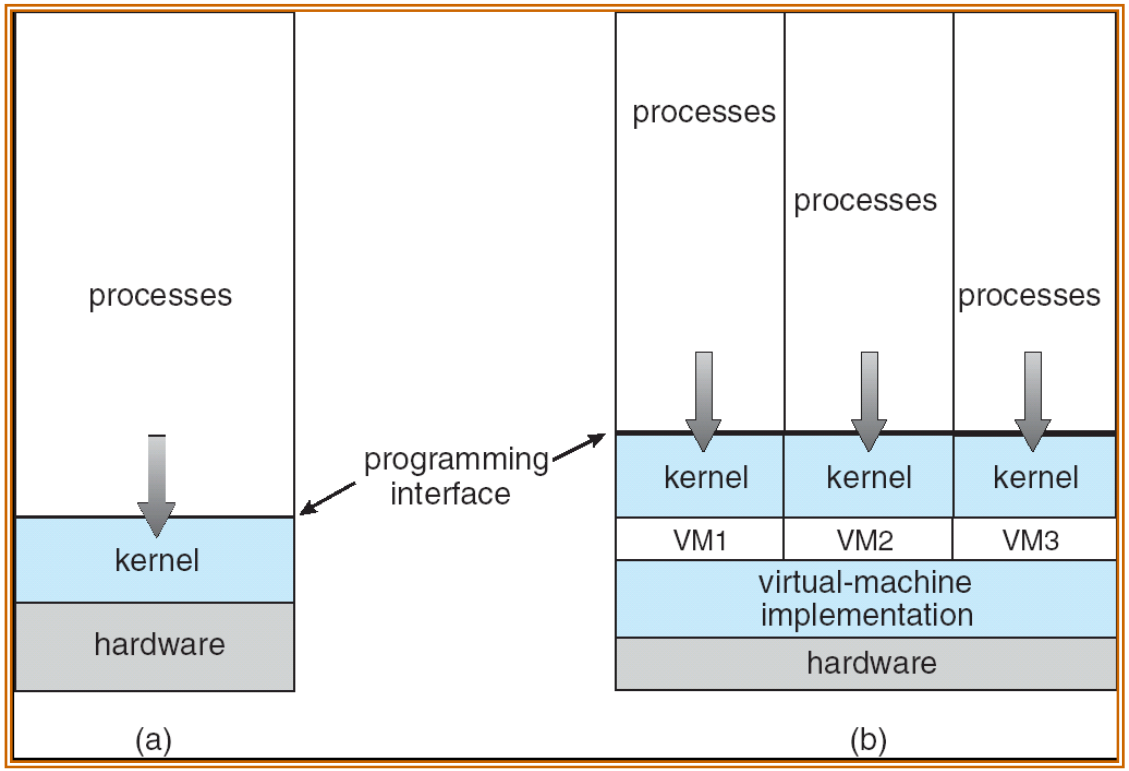






Virtual Machine Implementation 1: Native Hypervisor

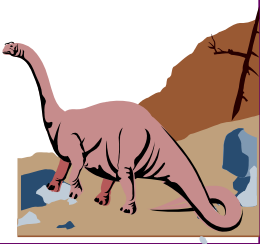
- A **hypervisor** or **virtual machine monitor** (**VMM**) is a piece of computer software, firmware or hardware that creates and runs virtual machines

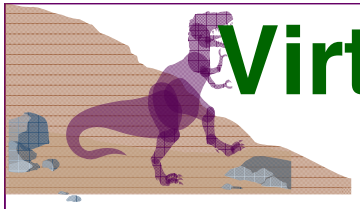


Non-virtual Machine

Virtual Machines over Native VMM

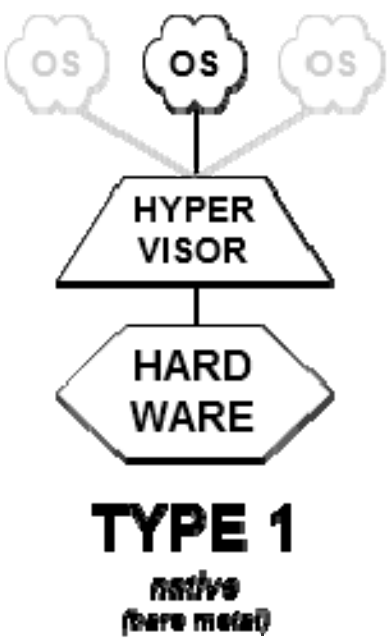
https://en.wikipedia.org/wiki/Hypervisor

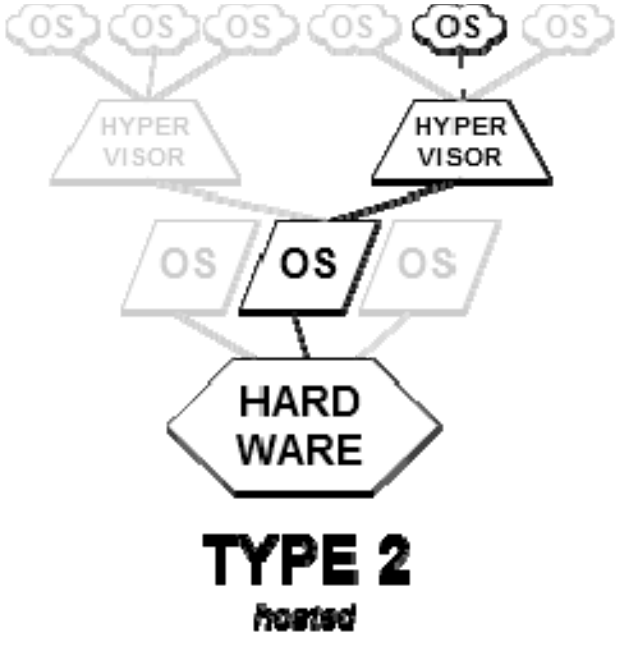


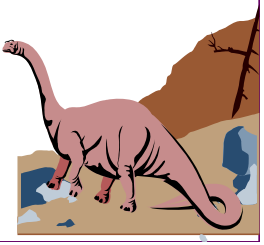


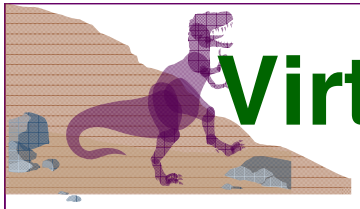
Virtual Machine Implementation 2: Hosted Hypervisor

- A hosted hypervisor takes the layered approach to its logical conclusion. It treats underlying hardware and the host operating system kernel as though they were all hardware.



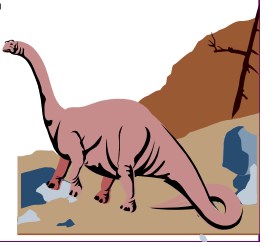


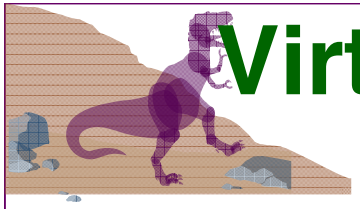




Virtual Machine Implementation 2: Hosted Hypervisor

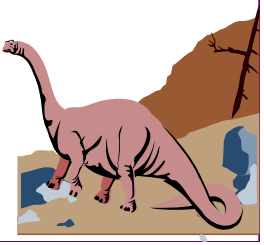
- This is how it works:
  - Your main OS runs like usual (Windows in LiLi's case). This OS is called "Host OS“.
  - A virtualization software (e.g., VirtualBox or VMWare) will launch a second OS on top of the first one.
  - The virtualization software will trick the second OS and give him some virtual hardware.
- Each OS, no matter virtual or host, is not aware of the other's existence.

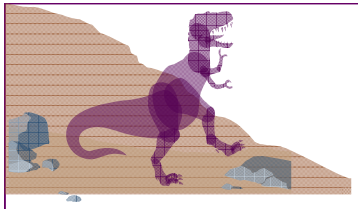




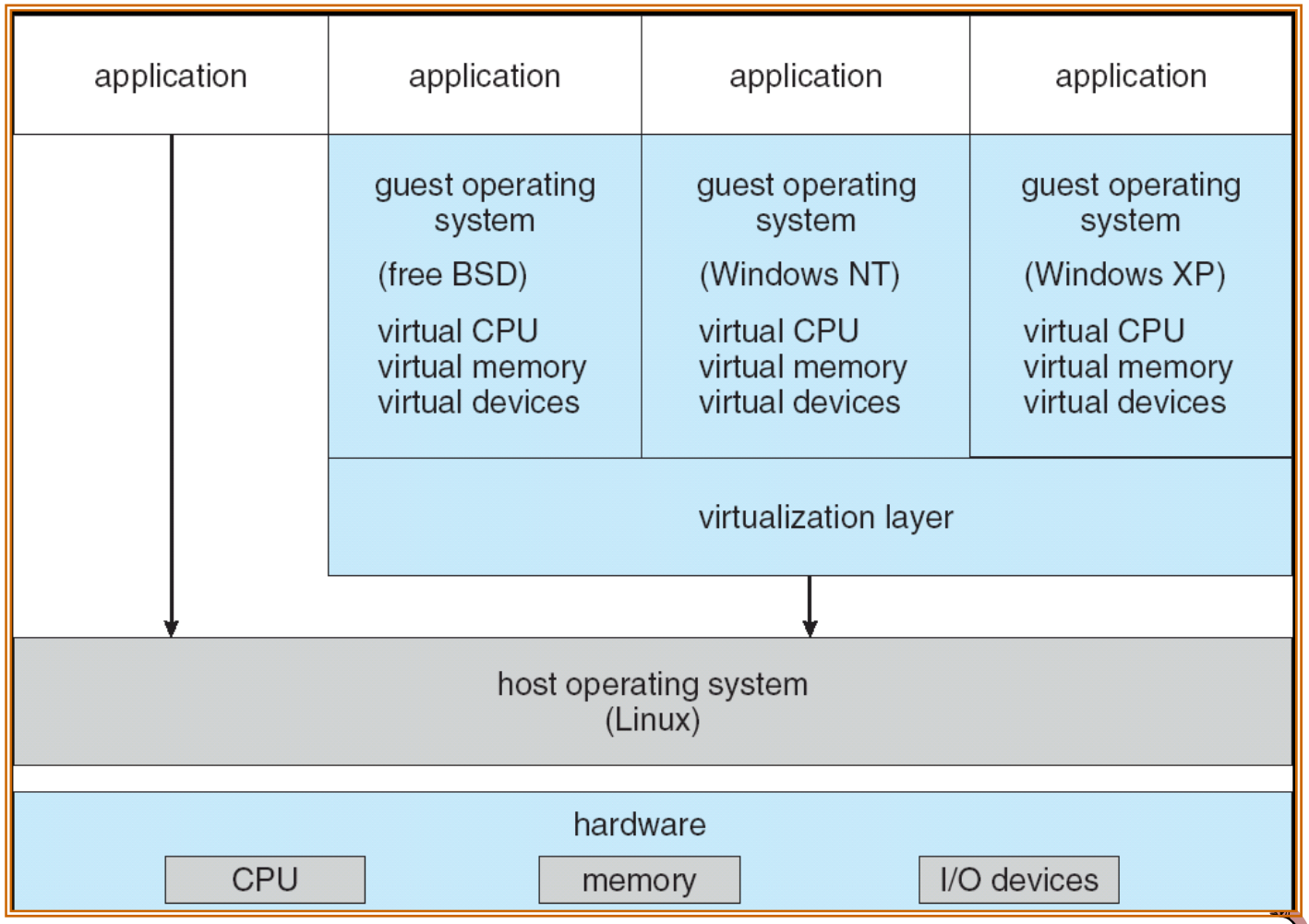
Virtual Machine Implementation 2: Hosted Hypervisor

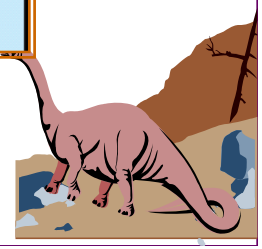
- The resources of the physical computer are shared to create the virtual machines
  - CPU scheduling can create the appearance that users have their own processor
  - Spooling and a file system can provide virtual disk, virtual card readers, and virtual line printers
  - A normal user time-sharing terminal serves as the virtual machine operator's console

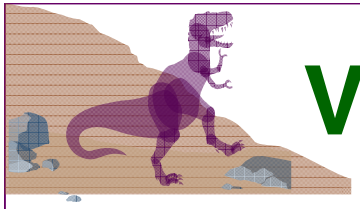




VMware Architecture

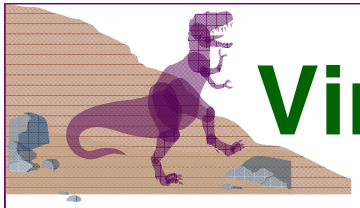






Virtual Machines Advantages

- The virtual-machine concept provides complete protection of system resources since each virtual machine is isolated from all other virtual machines. This isolation, however, permits no direct sharing of resources.
- A virtual-machine system is a perfect vehicle for operating-systems research and development. System development is done on the virtual machine, instead of on a physical machine and so does not disrupt normal system operation.

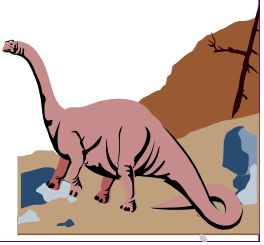


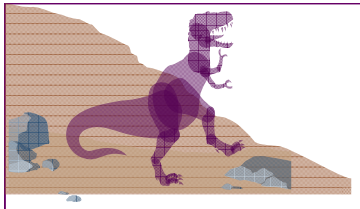
Virtual Machines Disadvantage

- The virtual machine concept is difficult to implement due to the effort required to provide an *exact* duplicate to the underlying machine

- Other lightweight virtualization mechanisms are available

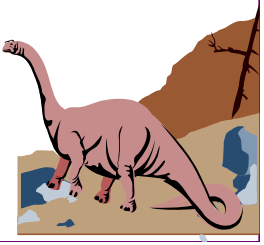
https://en.wikipedia.org/wiki/Virtualization

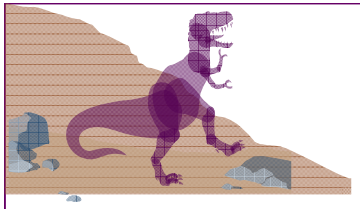




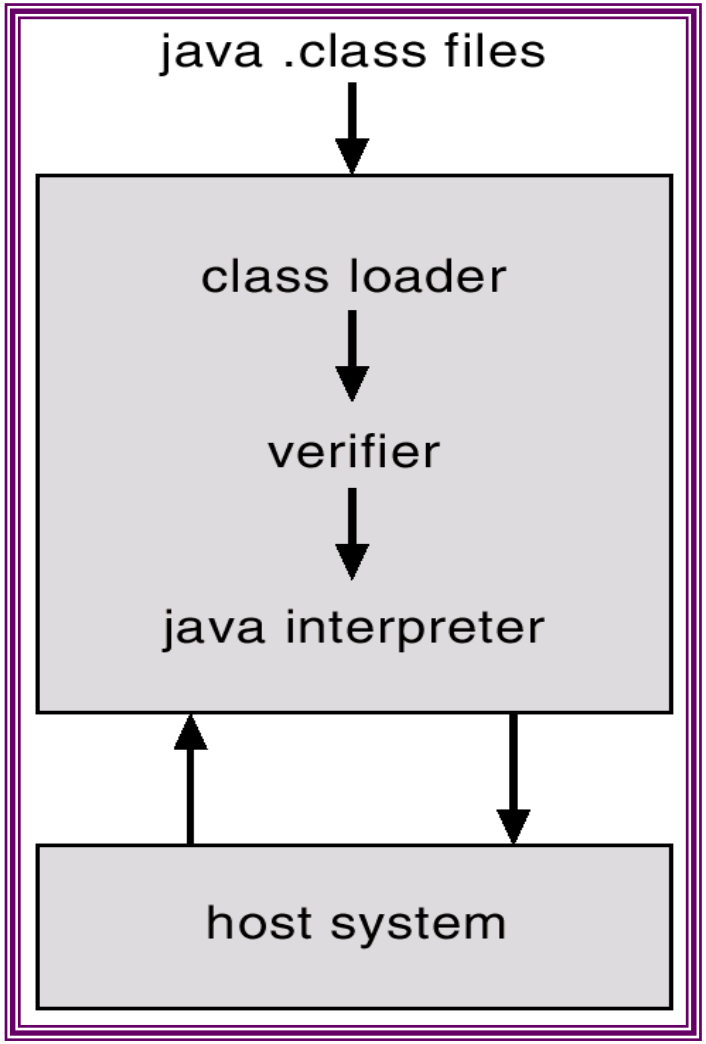
Java Virtual Machine

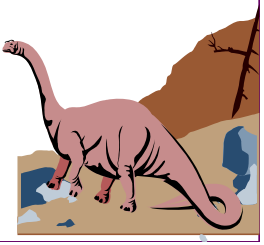
- Compiled Java programs are platform-neutral bytecodes executed by a Java Virtual Machine (JVM).
- JVM consists of
  - class loader
  - class verifier
  - runtime interpreter
- Just-In-Time (JIT) compilers increase performance

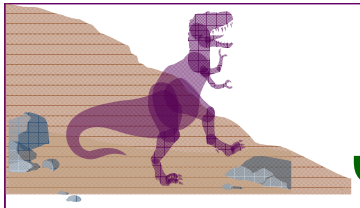




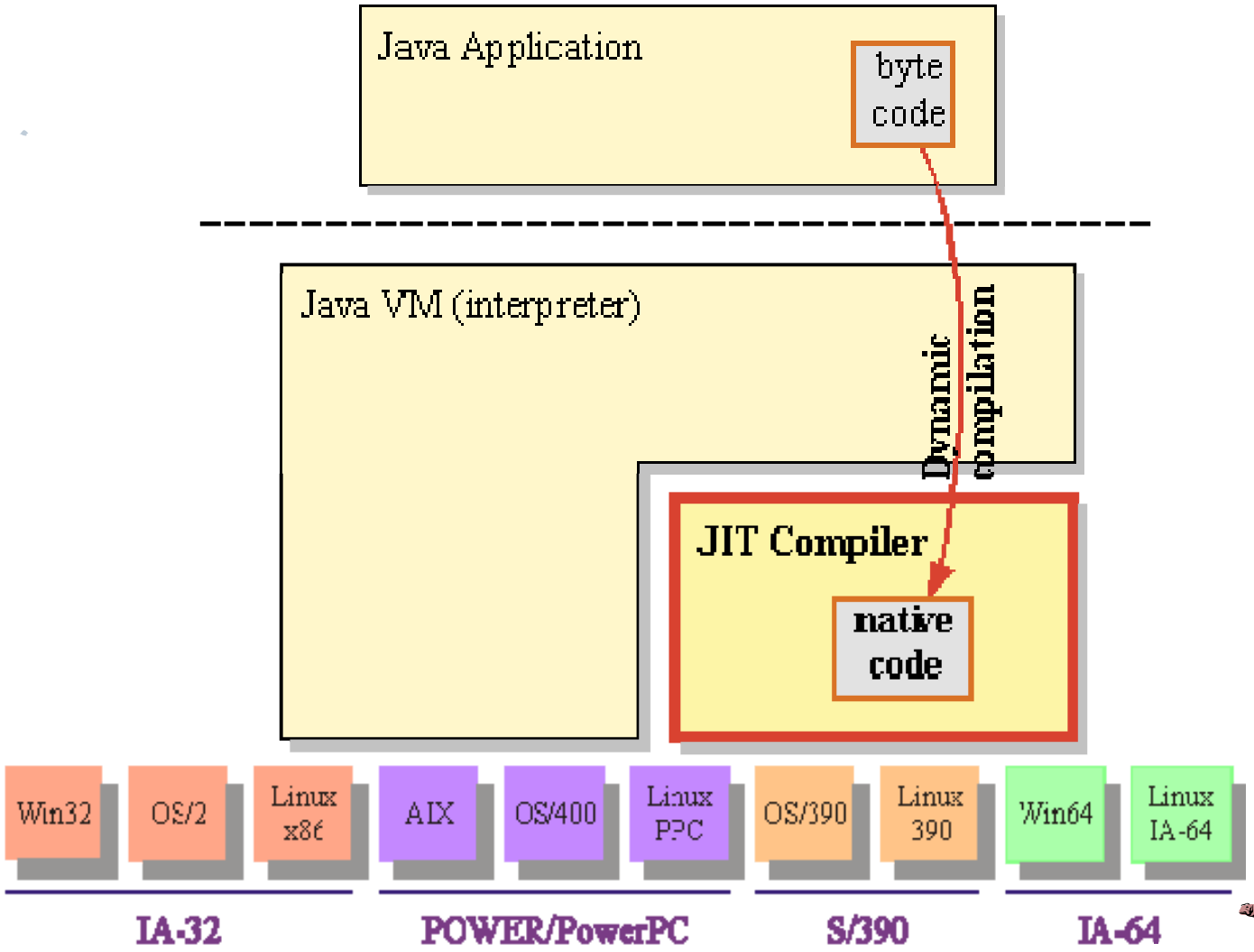
Java Virtual Machine

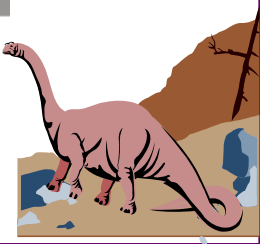


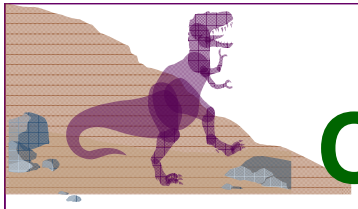




Java Virtual Machine (Cont.)

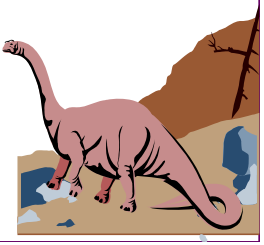






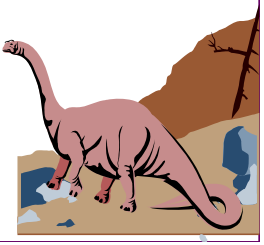
Operating System Generation

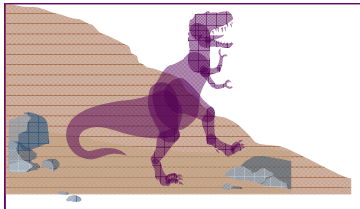
- Operating systems are designed to run on any of a class of machines; the system must be configured for each specific computer site
- SYSGEN program obtains information concerning the specific configuration of the hardware system



Operating System Generation (cont.)

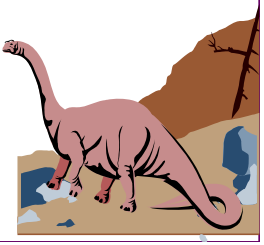
- *Booting* – starting a computer by loading the kernel
- *Bootstrap program* – code stored in ROM that is able to locate the kernel, load it into memory, and start its execution

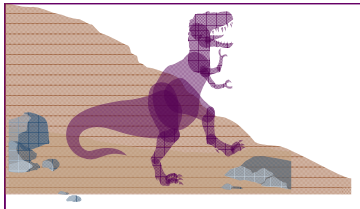




System Boot

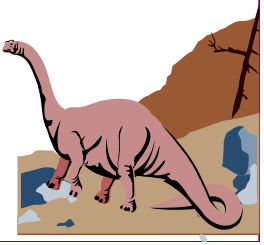
- Operating system must be made available to hardware so hardware can start it
  - Small piece of code – **bootstrap loader**, locates the kernel, loads it into memory, and starts it
  - Sometimes two-step process where **boot block** at fixed location loads bootstrap loader
  - When power is initialized on system, execution starts at a fixed memory location
    - ✓Firmware is used to hold the initial boot code

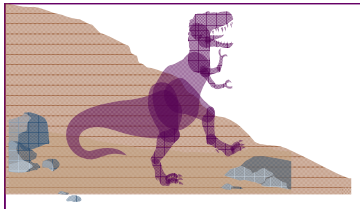




Operating System Design and Implementation

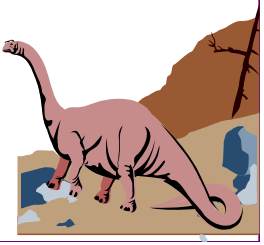
- Design and Implementation of OS not “solvable”, but some approaches have proven successful
- Internal structure of different operating systems can vary widely
- Start by defining goals and specifications

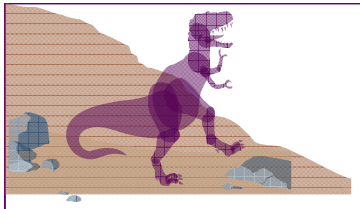




Operating System Design and Implementation (cont.)

- 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





Operating System Design and Implementation (cont.)

- Important principle to separate

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

- 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 if policy decisions are to be changed later

https://en.wikipedia.org/wiki/Separation_of_mechanism_and_policy