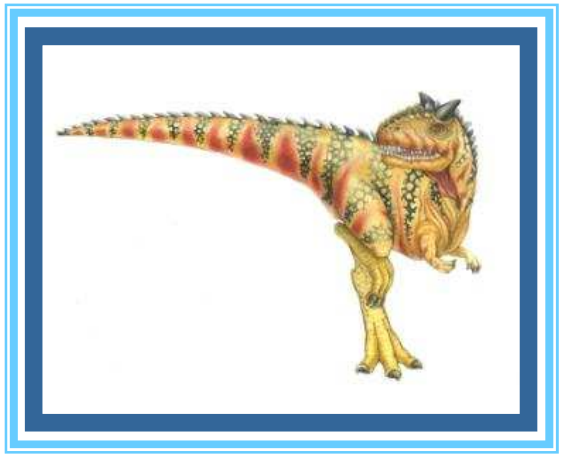
Topic 1: Introduction



- Chapter 1: Introduction
- Chapter 2: System Structures

Silberschatz, Galvin and Gagne ©2014

Rudowsky ©2005

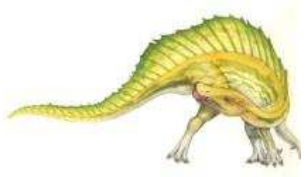
Walpole ©2010

Kubiatowicz ©2010

Stallings ©2015

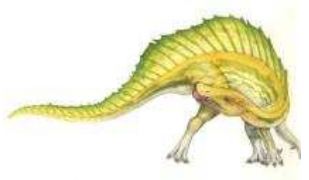
Operating Systems

http://conocea.com/blog/sistemas-operativos/



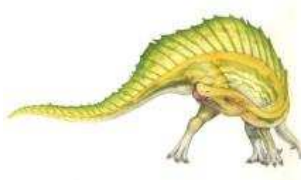
Contents

- What Operating Systems Do
- History of Operating Systems
- Operating Systems Services/Functions
- Operating Systems Structure and Implementation
- Hardware Protection/Support



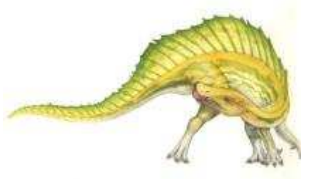
Contents

- **What Operating Systems Do**
- History of Operating Systems
- Operating Systems Services/Functions
- Operating Systems Structure and Implementation
- Hardware Protection/Support



Four Components of a Computer System

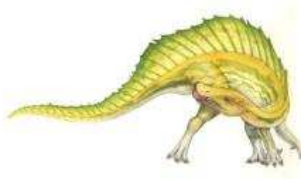
- Computer system can be divided into four components
  - Hardware – provides basic computing resources
    - CPU, memory, I/O devices
  - Operating system
    - Controls and coordinates use of hardware among various applications and users
  - Application programs – define the ways in which the system resources are used to solve the computing problems of the users
    - Word processors, compilers, web browsers, database systems, video games
  - Users
    - People, machines, other computers



What is an Operating System?

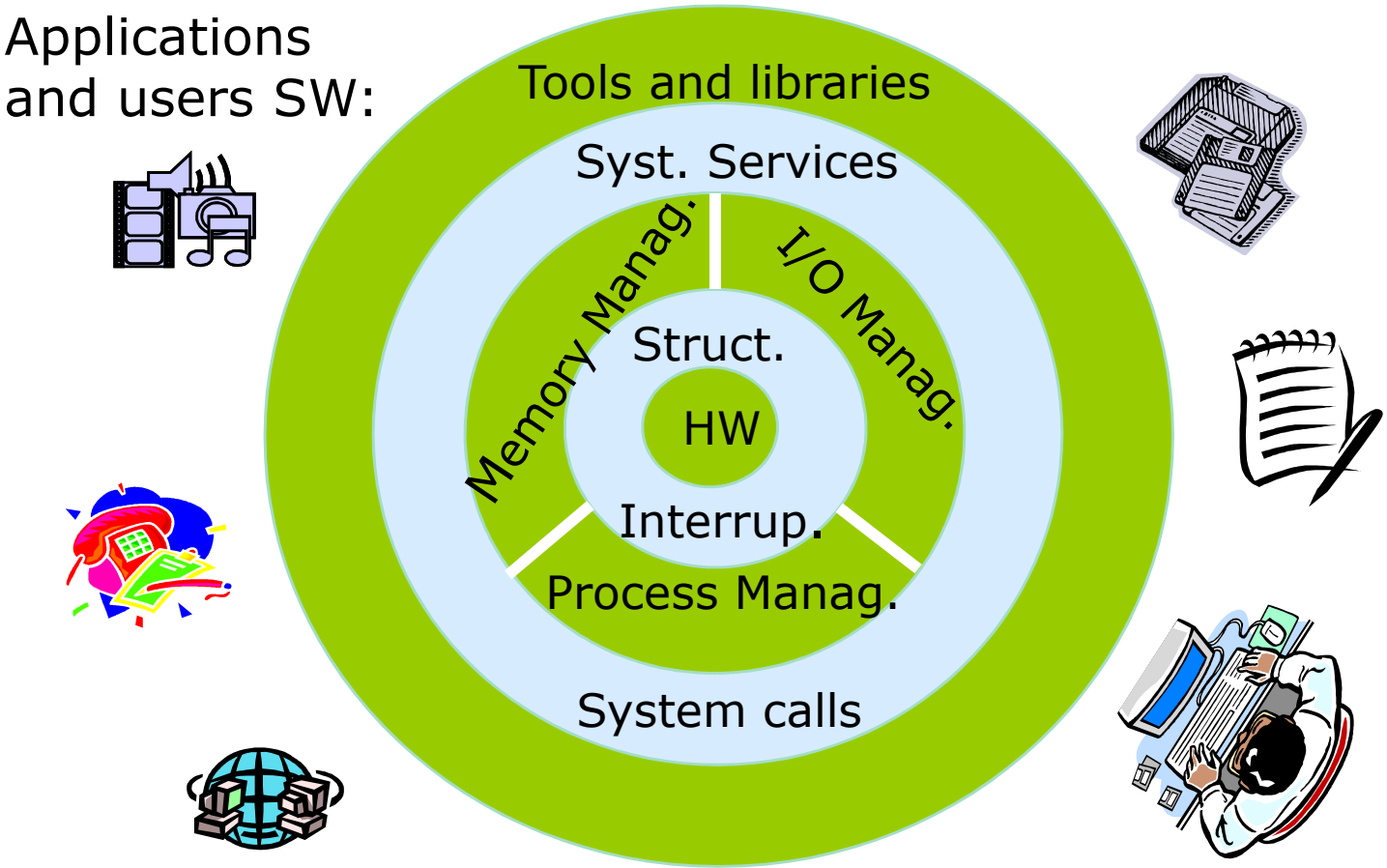
DEFINITIONS

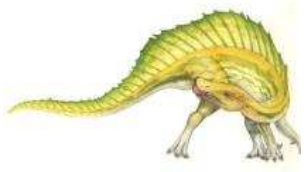
- A program that acts as an intermediary between a user of a computer and the computer hardware
- Group of programs implemented as software or firmware that allow to use the computer HW to all the users, reaching good features
- Program having a set of functions which goal is to simplify the management and use of the computer, doing it secure and efficient
- Program that manages the machine resources (CPU, memory, I/O devices, disks, network, etc.)



Context

- Where is the OS in a computer system?





Computer System Structure

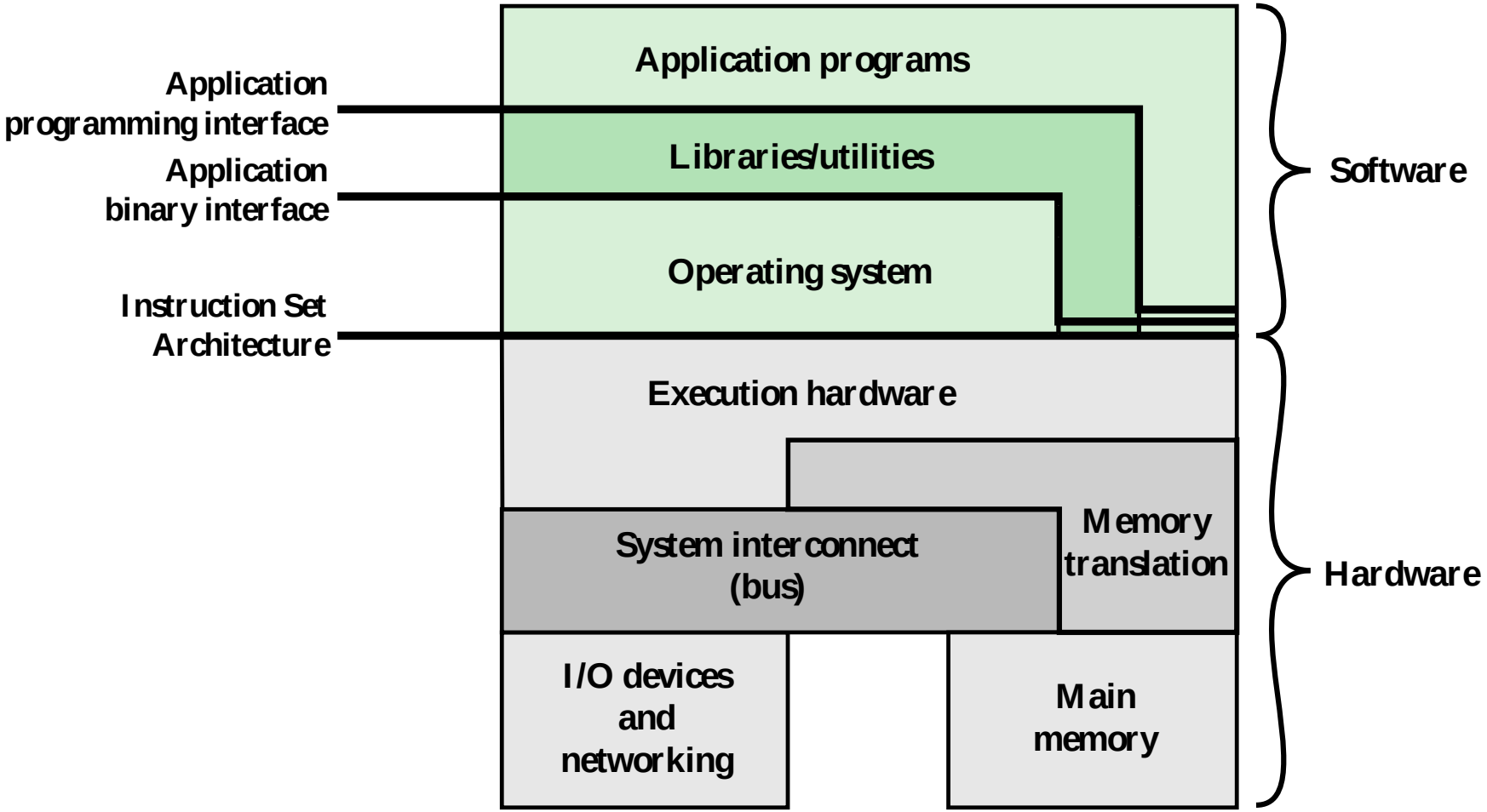
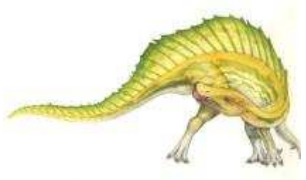


Figure 2.1 Computer Hardware and Software Structure

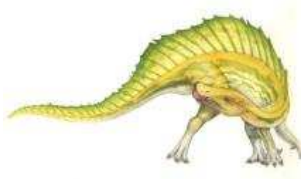
Source: Operating Systems. W. Stallings



Operating System goals

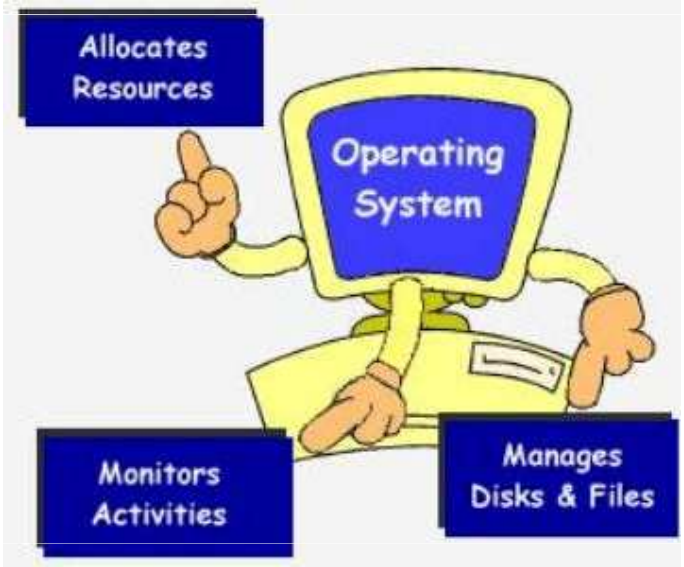
- Operating system goals:
  - Execute user programs and make solving user problems easier
  - Make the computer system convenient to use
  - Use the computer hardware in an efficient manner

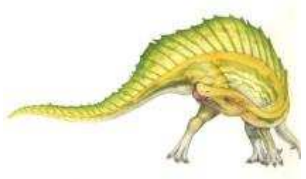
- ***User* goals** and ***System* goals**
  - User goals – operating system should be convenient to use, easy to learn, reliable, safe, and fast
  - System goals – operating system should be easy to design, implement, and maintain, as well as flexible, reliable, error-free, and efficient



Operating System Services

- Services/Functions:
  - **Resource Manager**
    - Memory Management
    - I/O Management
    - CPU Scheduling
  - **Abstract Machine**
    - Operating System Services: system calls/API
  - **Interface between User and Machine**
    - User interface: to control of the machine on the user's end, and to receive feedback from the machine
      - Graphics User Interface (GUI) or Command-Line (CLI)

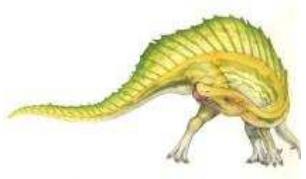




What if we didn't have an OS?

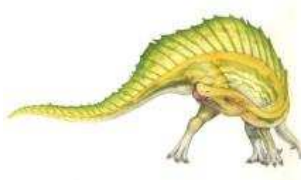
- Source Code⇒Compiler⇒Object Code⇒Hardware
- How do you get object code onto the hardware?
- How do you print out the answer?
- Once upon a time, had to Toggle in program in binary and read out answer from LED's!

Altair 8080



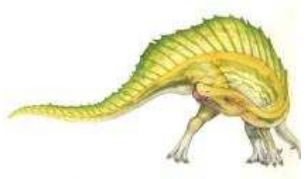
Contents

- What Operating Systems Do
- **History of Operating Systems**
- Operating Systems Services/Functions
- Operating Systems Structure and Implementation
- Hardware Protection/Support



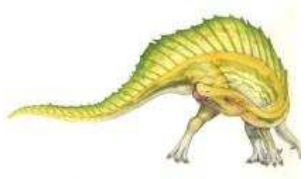
Why to study history of O.S.

- A feature that once run only on huge systems may have made its way into very small systems
  - Many features once available only on mainframes have been adopted for microcomputers
- The same O.S. concepts are appropriate for various classes of computers: mainframes, minicomputers, microcomputers, and handhelds.
- Ej. MULTICS (mainframe) → UNIX (minicomputer) → Unix like (microcomputers) → Windows, Linux (desktop) → Andorid, iOS (handheld)

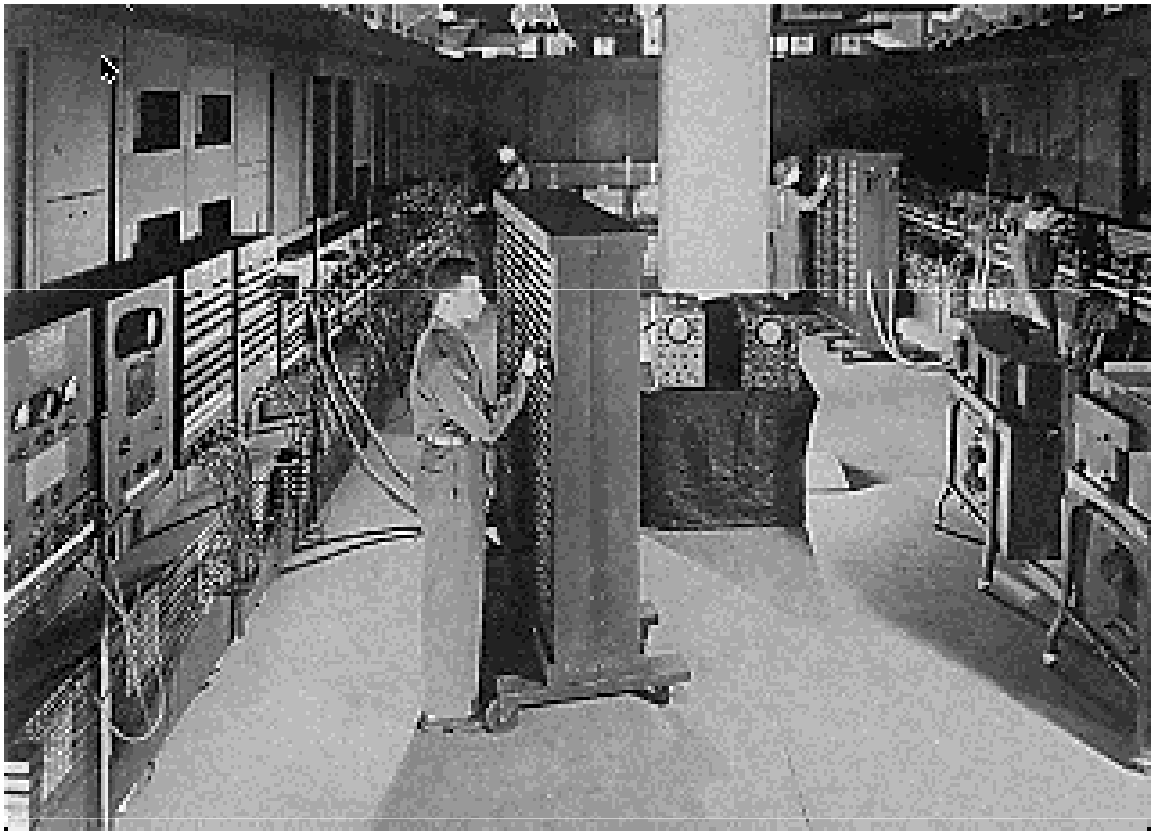


History of Operating Systems

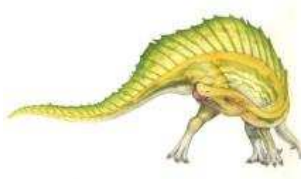
- Pre-electronic
  - Charles Babbage (1792-1871) “analytical machine”
  - Purely mechanical, failed because technology could not produce the required wheels, cog, gears to the required precision
- First generation 1945 - 1955
  - Aiken, von Neumann, Eckert, Mauchley and Zuse
  - programming done via plugboards, **no OS or language**
  - vacuum tubes



ENIAC: (1945—1955)

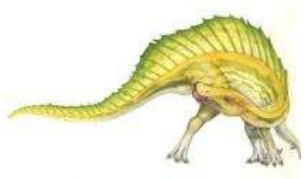


- “The machine designed by Drs. Eckert and Mauchly was a monstrosity. When it was finished, the ENIAC filled an entire room, weighed thirty tons, and consumed two hundred kilowatts of power.”
- http://ei.cs.vt.edu/~history/ENIAC.Richey.HTML



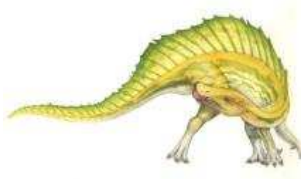
Second generation 1955 - 1965

- Transistors more reliable than vacuum tubes



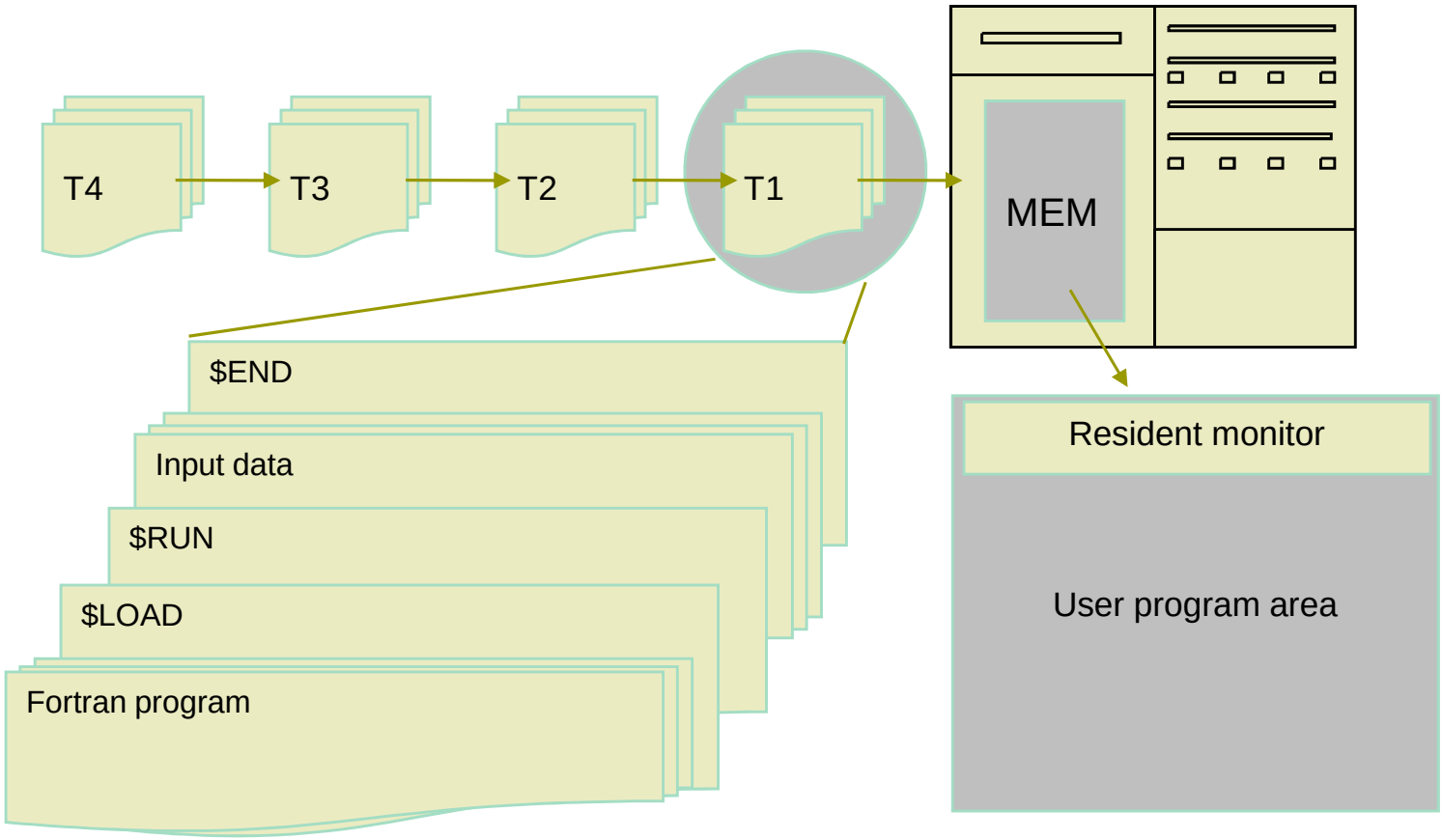
Second generation 1955 - 1965

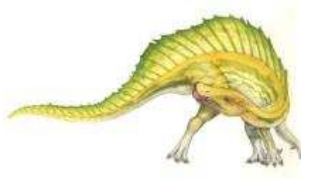
- Computers cost millions of $'s: optimize for more efficient use of the hardware
  - **batch systems** introduced to reduce wasted time in setting up and running jobs
  - Lack of interaction between user and computer
    - When user thinking at console, computer idle⇒BAD!
    - Feed computer batches and make users wait
- Compilers appear (Fortran, Cobol..)
- First rudimentary OS for automatic job sequencing: resident monitor
  - Always in memory
  - load program, run, print
  - Implements device drivers both for system and application programming



Second generation 1955 - 1965

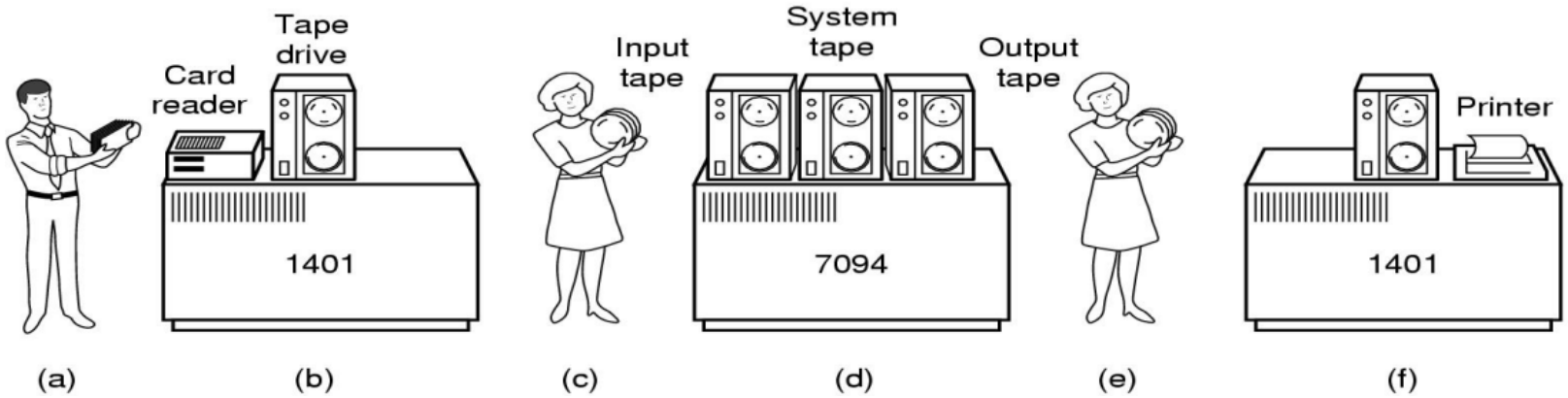
- Jobs read in via punched cards:
  - Data, program for a job
  - Control cards: directives to the resident monitor indicating what program to run



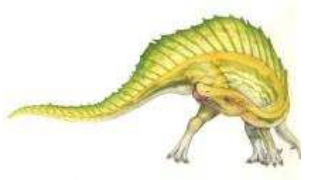


Off-line execution

- Even with automatic job sequencing the expensive CPU is often idle
  - The mechanical I/O devices are intrinsically slower
    - Solution: overlapped I/O:
- Off-line execution: slow card readers (input) and line printers (output) replaced by magnetic-tape units: separate devices for input and output



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

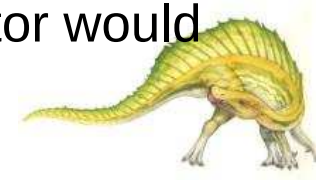


Third generation 1965 – 1980

- Use of integrated circuits provided major price/performance advantage over 2nd generation

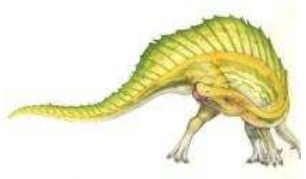
A Multics System

- The 6180 at MIT IPC, skin doors open, circa 1976:
  - "We usually ran the machine with doors open so the operators could see the AQ register display, which gave you an idea of the machine load, and for convenient access to the EXECUTE button, which the operator would push to enter BOS if the machine crashed."
  - [http://www.multicians.org/multics-stories.html]



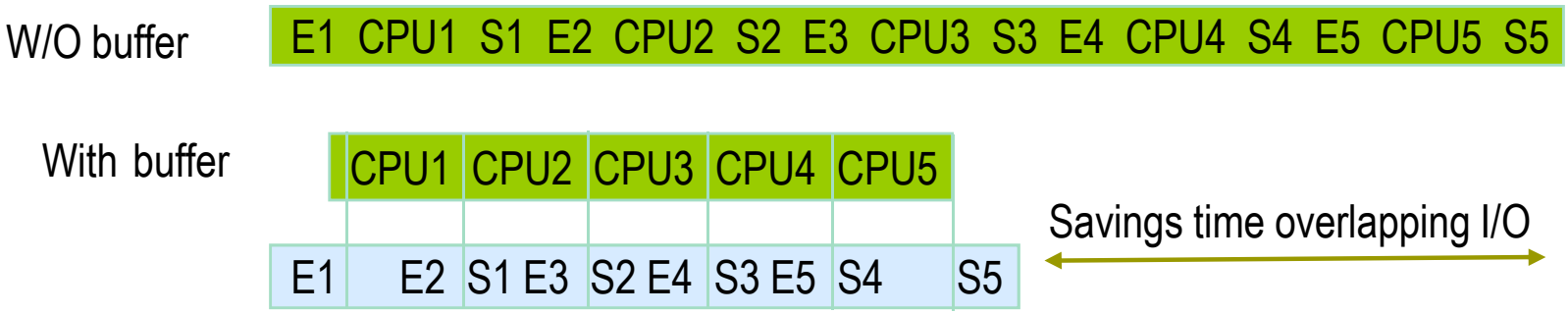
Third generation 1965 – 1980

- Computers available for tens of thousands of dollars instead of millions
- Complexity gets out of hand:
  - Multics: announced in 1963, ran in 1969
  - OS 360: released with 1000 known bugs (APARs)
    - "Anomalous Program Activity Report"
- OS finally becomes an important science:
  - How to deal with complexity???
  - UNIX based on Multics, but vastly simplified

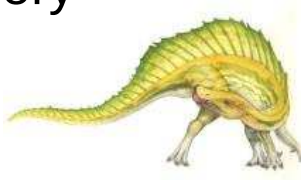


Third generation 1965 – 1980

- Data channels, Interrupts: overlap I/O and compute
  - DMA – Direct Memory Access for I/O devices
  - I/O can be completed asynchronously
- Goal: Reduce I/O time $\Rightarrow$ new techniques such as:
  - Buffering: storing data temporarily while it is being transferred (through DMA)

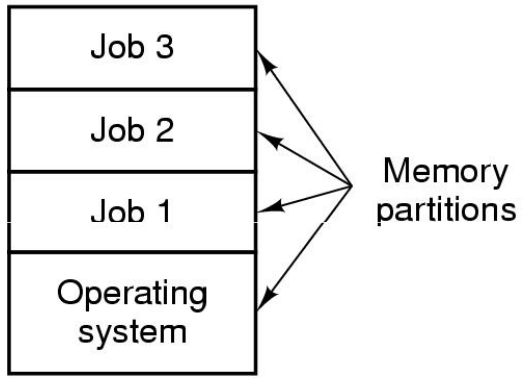


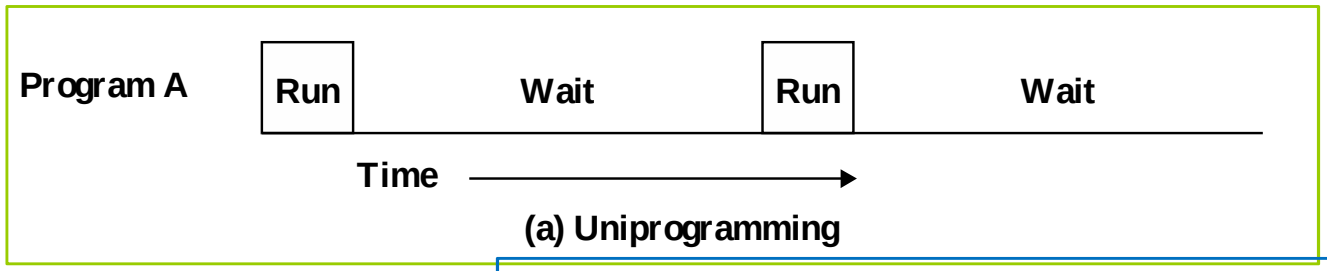
  - Spooling: read jobs from cards to disk ready to load into memory and queue output to disk for printing

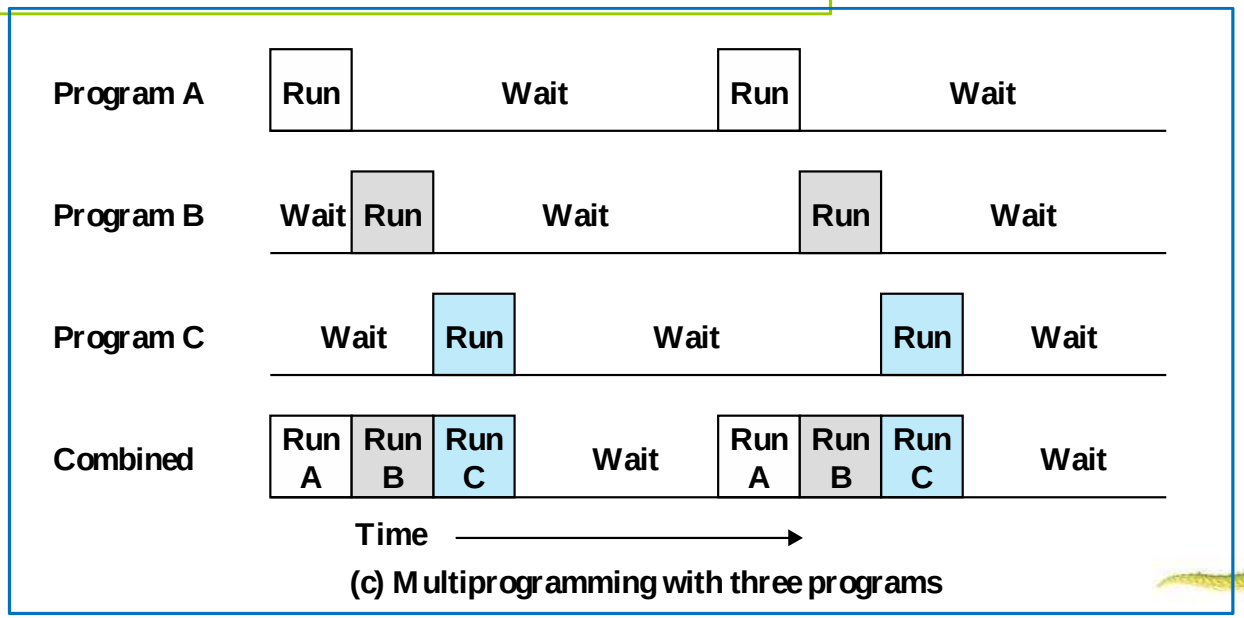


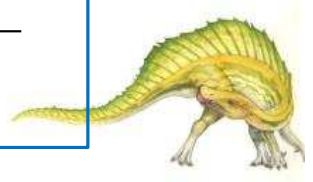
Third generation 1965 – 1980

- Introduced multiprogramming to make most efficient use of CPU: several programs run simultaneously
  - Small jobs not delayed by large jobs
  - More overlap between I/O and CPU
  - Need memory protection between programs and/or OS





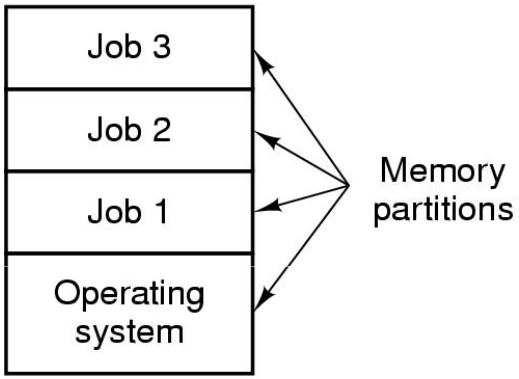




Source: Operating Systems. W. Stallings Fig. 2.5

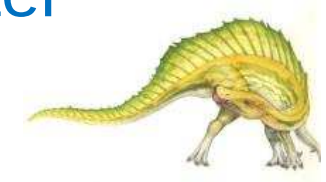
Aside: Degree of multiprogramming

- Definition:
  - Degree of multiprogramming is the number of processes in memory (except for the OS)
- Example:



Degree of multiprogramming= 3

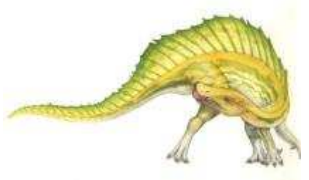
Greater degree of multiprogramming, the greater system performance



Multiprogramming Example

	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

Table 2.1 Sample Program Execution Attributes



Source: Operating Systems. W. Stallings

Multiprogramming Example

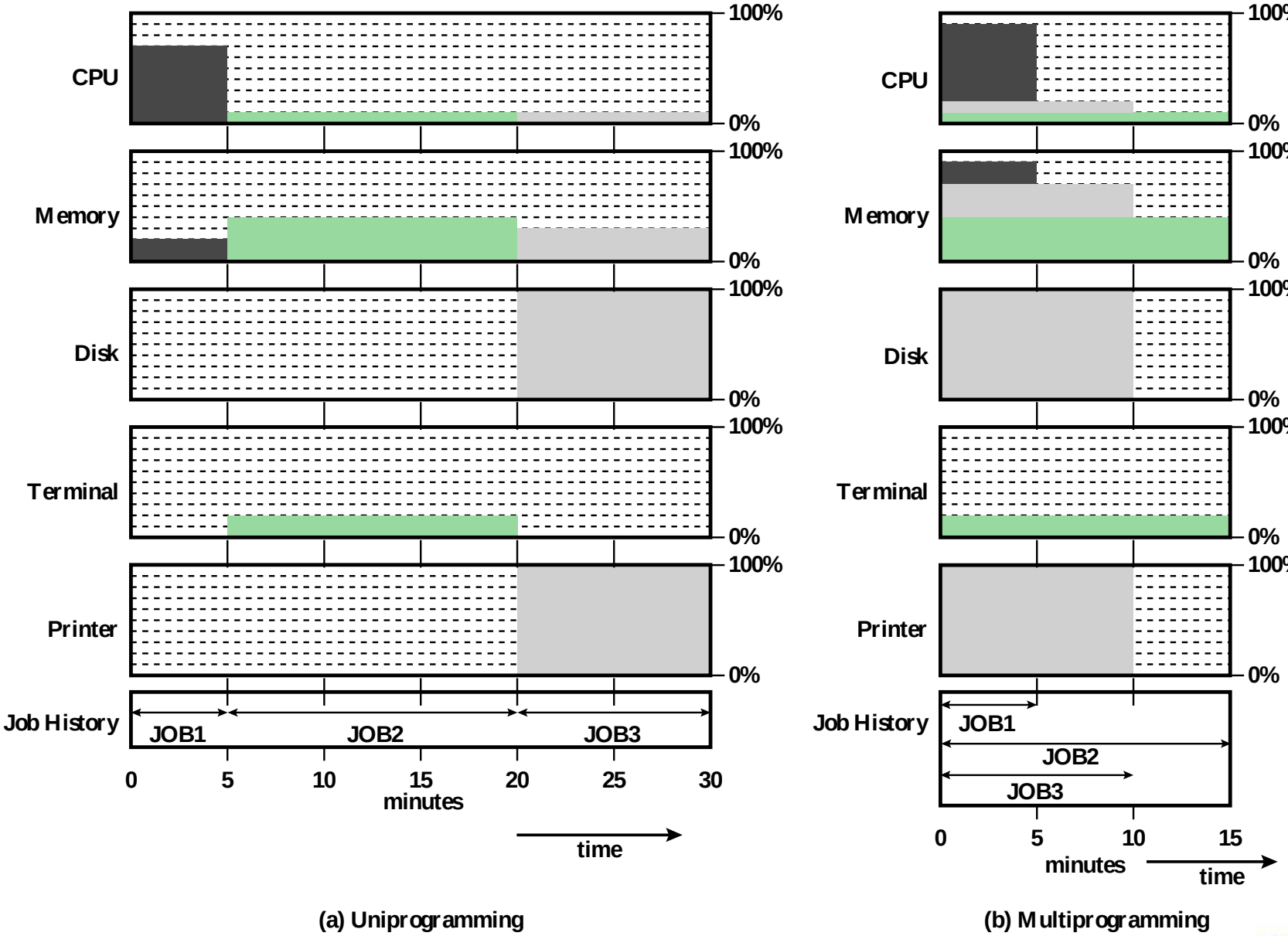
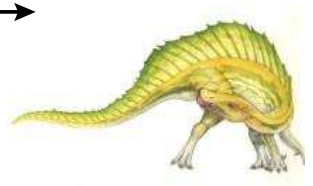


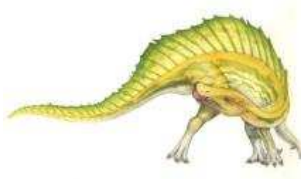
Figure 2.6 Utilization Histograms

Source: Operating Systems. W. Stallings



Third generation 1965 – 1980

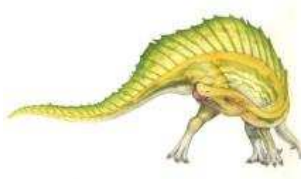
- Timesharing (a variant of multiprogramming) provides for user interaction with the computer system
  - Use cheap terminals (~$1000) to let multiple users interact with the system at the same time
  - On-line communication between the user and the system is provided; when the operating system finishes the execution of one command, it seeks the next “control statement” from the user’s keyboard.
  - The CPU is multiplexed among several jobs that are kept in memory and on disk (the CPU is allocated to a job only if the job is in memory)
    - switch occurs so frequently that the user can interact with each program as it is running
    - each user is given the impression that the entire system is dedicated to his use
    - Batch jobs could be running in background



Problems an OS must solve

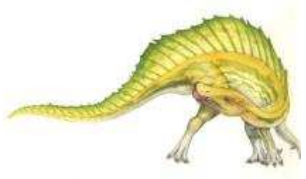
- Time sharing the CPU among applications
- Space sharing the memory among applications
- Space sharing the disk among users
- Time sharing access to the disk
- Time sharing access to the network
- Protection
  - of applications from each other
  - of user data from other users
  - of hardware/devices
  - of the OS itself!

- Timesharing: user authentication and protection; File System



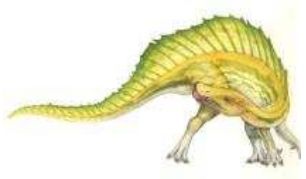
Operating System Structure

- **Multiprogramming** needed for efficiency
  - Single user cannot keep CPU and I/O devices busy at all times
  - Multiprogramming organizes jobs (code and data) so CPU always has one to execute
  - A subset of total jobs in system is kept in memory
  - One job selected and run via **job scheduling**
  - When it has to wait (for I/O for example), OS switches to another job
- **Timesharing (multitasking)** is logical extension in which CPU switches jobs so frequently that users can interact with each job while it is running, creating **interactive** computing
  - **Response time** should be < 1 second
  - Each user has at least one program executing in memory ⇨**process**
  - If several jobs ready to run at the same time ⇨ **CPU scheduling**
  - If processes don't fit in memory, **swapping** moves them in and out to run
  - **Virtual memory** allows execution of processes not completely in memory



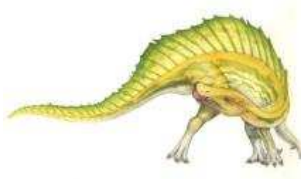
Fourth generation 1980 – present

- Large Scale Integrated chips
- Computer costs $1K, Programmer costs $100K/year
  - If you can make someone 1% more efficient by giving them a computer, it's worth it!
  - Use computers to make people more efficient
- Personal computers:
  - computers cheap, so give everyone a PC



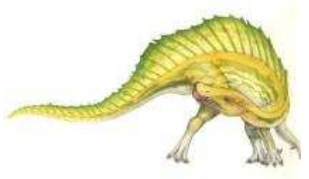
Fourth generation 1980 – present

- Limited Hardware Resources Initially:
  - OS becomes a subroutine library
  - One application at a time (MSDOS, CP/M, …)
- Eventually PCs become powerful:
  - OS regains all the complexity of a “big” OS
  - multiprogramming, memory protection, etc (NT,OS/2)



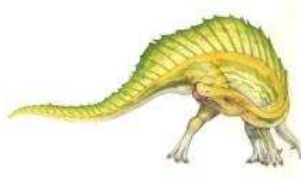
Fourth generation 1980 – present

- Distributed OS
  - OS over a collection of independent networked computers
  - Appear to the users of the system as a more powerful single computer
  - Examples: Mach, Amoeba
- Middleware
  - SW layer executed over a network of computers
  - Each computer has its own OS
  - Examples: Corba, DCOM (Microsoft)



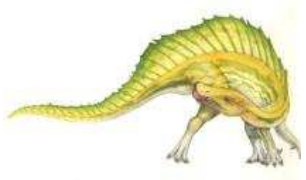
Fourth generation 1980 – present

- Multiprocessor OS
  - Parallel systems provide speed and fault tolerance
  - OS with Asymmetric Multiprocessing support
    - A CPU runs the OS and the remaining ones runs the process
  - OS with Symmetric Multiprocessing (SMP) support
    - Any CPU can run the OS
    - Multithreading OS (Linux, Windows) support SMP
- Real time OS
  - Systems with temporal constrains: response time bound
  - No HD nor virtual mem. OS stored in ROM
  - Hard RT systems: response time very short and fixed
    - Response time with deadline
  - Soft RT systems: critical process have high priority
    - Priority scheduling



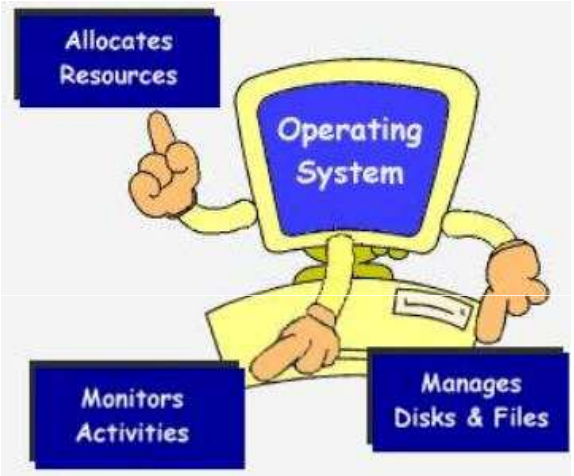
Contents

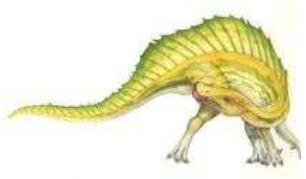
- What Operating Systems Do
- History of Operating Systems
- **Operating Systems Services/Functions**
- Operating Systems Structure and Implementation
- Hardware Protection/Support



Operating System Services

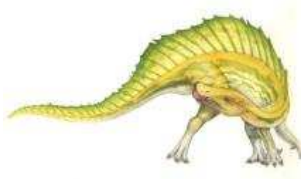
- Services/Functions:
  - **Resource Manager**
    - Memory Management
    - I/O Management
    - CPU Scheduling
  - **Abstract Machine**
    - Operating System Services: system calls/API
  - **Interface between User and Machine**
    - User interface: to control of the machine on the user's end, and to receive feedback from the machine
      - Graphics User Interface (GUI) or Command-Line (CLI).





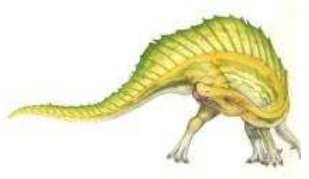
Operating System Services

- OS is a **Resource Manager**
  - Controls access to all shared resources:
    - Time management: CPU and disk transfer scheduling
    - Space management: main and secondary storage allocation
    - Synchronization and deadlock handling: IPC, critical section, coordination
    - Accounting and status information: resource usage tracking
  - Decides between conflicting requests for efficient and fair resource use



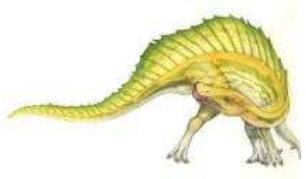
Operating System Services

- One set of OS functions exists for ensuring the **efficient operation of the system** itself via resource sharing
  - **Resource allocation** - When multiple users or multiple jobs running concurrently, resources must be allocated to each of them
    - Many types of resources - Some (such as CPU cycles, main memory, and file storage) may have special allocation code, others (such as I/O devices) may have general request and release code
  - **Accounting** - To keep track of which users use how much and what kinds of computer resources



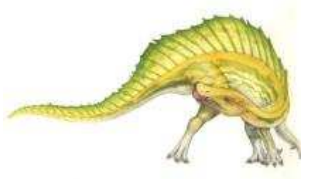
Operating System Services (Cont)

- One set of OS functions exists for ensuring the **efficient operation of the system** itself via resource sharing
  - **Protection and security** - The owners of information stored in a multiuser or networked computer system may want to control use of that information, concurrent processes should not interfere with each other
    - **Protection** involves ensuring that all access to system resources is controlled
    - **Security** of the system from outsiders requires user authentication, extends to defending external I/O devices from invalid access attempts
    - If a system is to be protected and secure, precautions must be instituted throughout it. A chain is only as strong as its weakest link



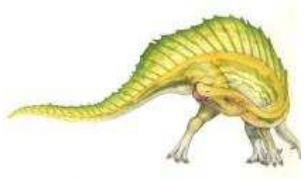
Operating System Services (Cont)

- Another set of operating-system services provides functions that are helpful to the **user**:
  - **Program execution** - The system must be able to load a program into memory and to run that program, end execution, either normally or abnormally (indicating error)
  - **I/O operations** - A running program may require I/O, which may involve a file or an I/O device
  - **File-system manipulation** - The file system is of particular interest. Obviously, programs need to read and write files and directories, create and delete them, search them, list file Information, permission management.

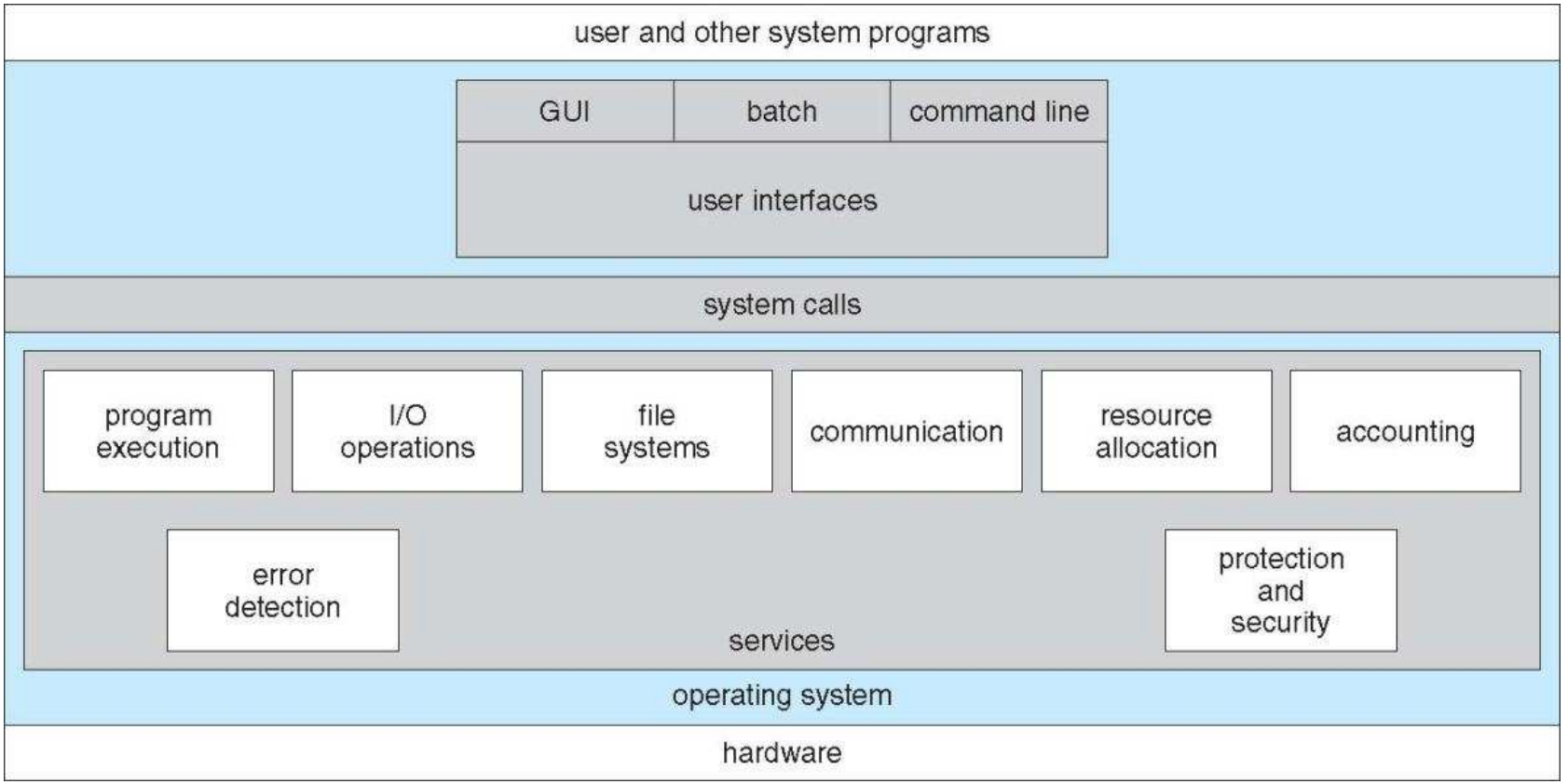


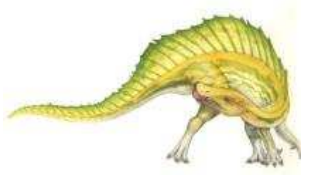
Operating System Services (Cont)

- Another set of operating-system services provides functions that are helpful to the **user**:
  - **Communications** – Processes may exchange information, on the same computer or between computers over a network
    - Communications may be via shared memory or through message passing (packets moved by the OS)
  - **Error detection** – OS needs to be constantly aware of possible errors
    - May occur in the CPU and memory hardware, in I/O devices, in user program
    - For each type of error, OS should take the appropriate action to ensure correct and consistent computing



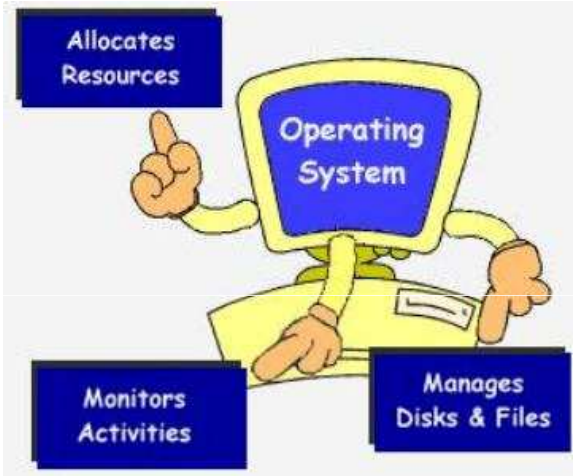
A View of Operating System Services

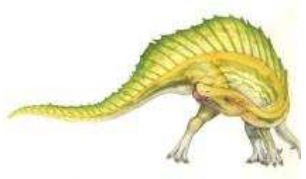




Operating System Services

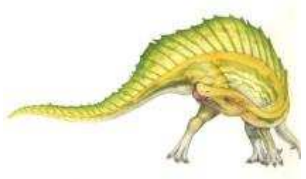
- Services/Functions:
  - **Resource Manager**
    - Memory Management
    - I/O Management
    - CPU Scheduling
  - **Abstract Machine**
    - Operating System Services: system calls/API
  - **Interface between User and Machine**
    - User interface: to control of the machine on the user's end, and to receive feedback from the machine
      - Graphics User Interface (GUI) or Command-Line (CLI).



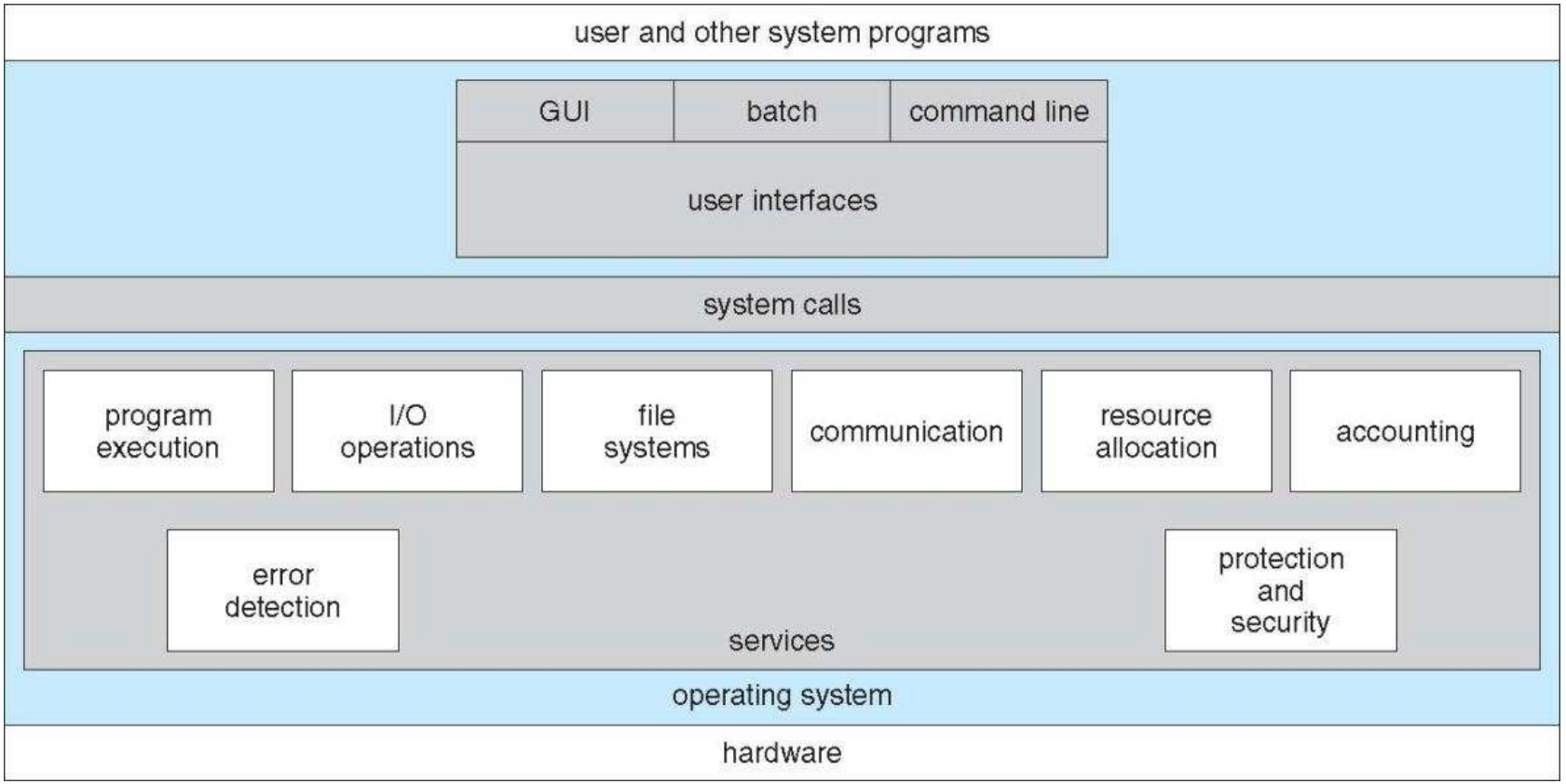


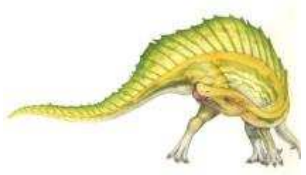
Operating System Services

- OS is an **Abstract Machine**
  - OS layer *transforms bare hardware machine into higher level abstractions:*
    - Hides complex details of the underlying hardware
    - Provides common API to applications and services
    - Simplifies application writing



A View of Operating System Services

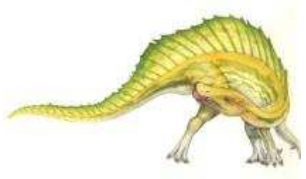




Source: Operating System Concepts. A. Silberschatz. Fig. 2.1

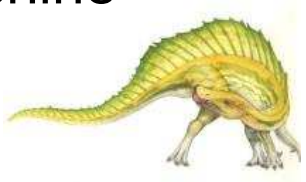
Why is abstraction important?

- Without OSs and abstract interfaces, application writers must program all device access directly
  - load device command codes into device registers
  - handle initialization, recalibration, sensing, timing etc for physical devices
  - understand physical characteristics and layout
  - control motors
  - interpret return codes … etc
- **Applications** suffer from
  - very complicated maintenance and upgrading
  - no portability
  - writing this code once, and sharing it, is how OS began!



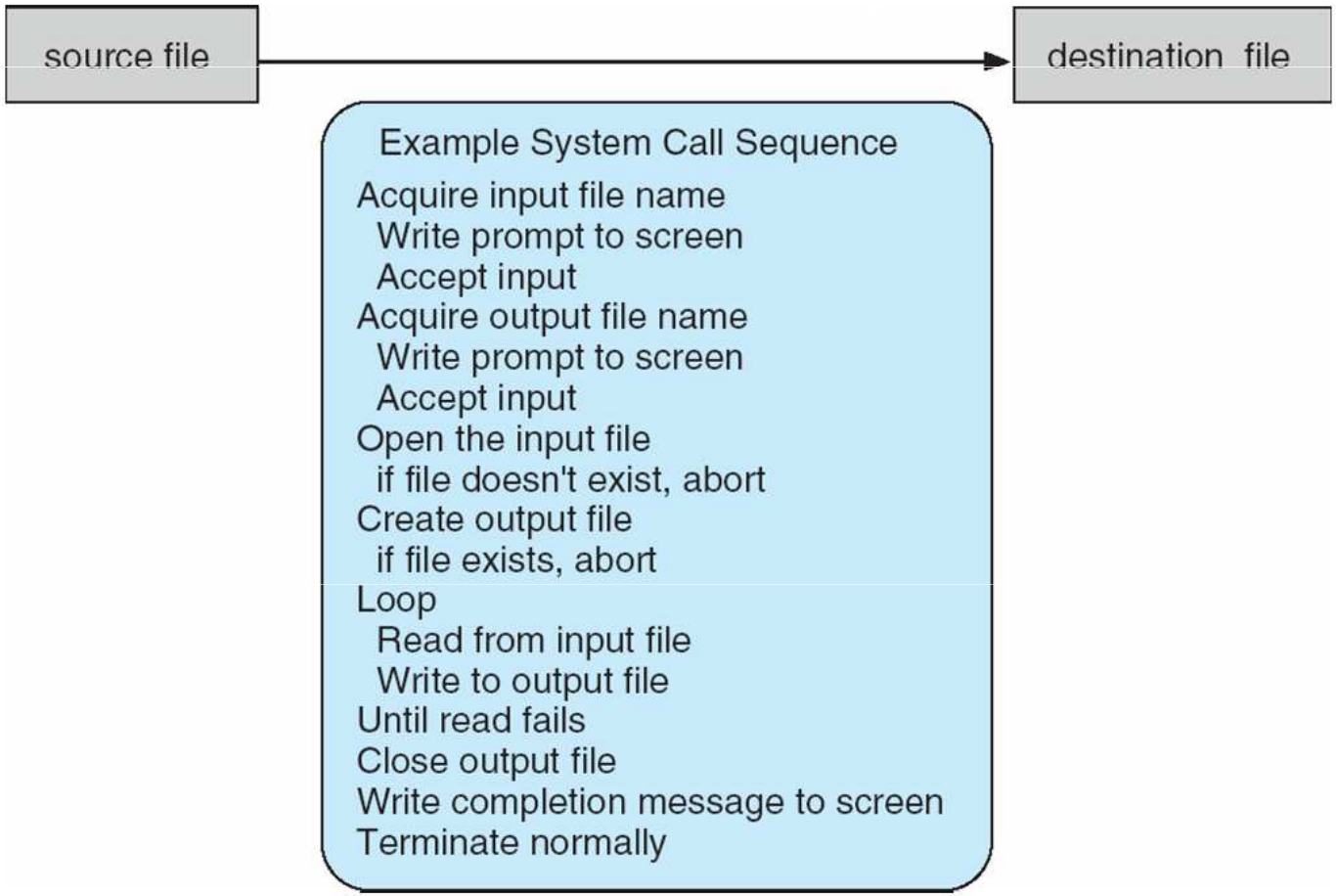
System Calls

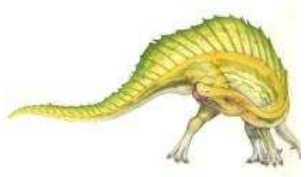
- System calls provide the interface between a running program and the operating system.
  - For example – open input file, create output file, print message to console, terminate with error or normally
  - Generally available as routines written in C and C++
  - Certain low-level tasks (direct hardware access) may be written in assembly-language
- Mostly accessed by programs via a high-level **Application Program Interface (API)** rather than direct system call use
  - Provides portability (underlying hardware handled by OS)
  - Hides the detail from the programmer
- Three most common APIs are Win32 API for Windows, POSIX API for POSIX-based systems (including virtually all versions of UNIX, Linux, and Mac OS X), and Java API for the Java virtual machine (JVM)



Example of System Calls

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





Source: Operating System Concepts. A. Silberschatz. Fig. 2.5

Example of Standard API

EXAMPLE OF STANDARD API

As an example of a standard API, consider the `read()` function that is available in UNIX and Linux systems. The API for this function is obtained from the `man` page by invoking the command

```
man read
```

on the command line. A description of this API appears below:

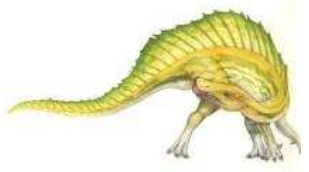
```
#include <unistd.h>

ssize_t    read(int fd, void *buf, size_t count)
```

return value — function name — parameters

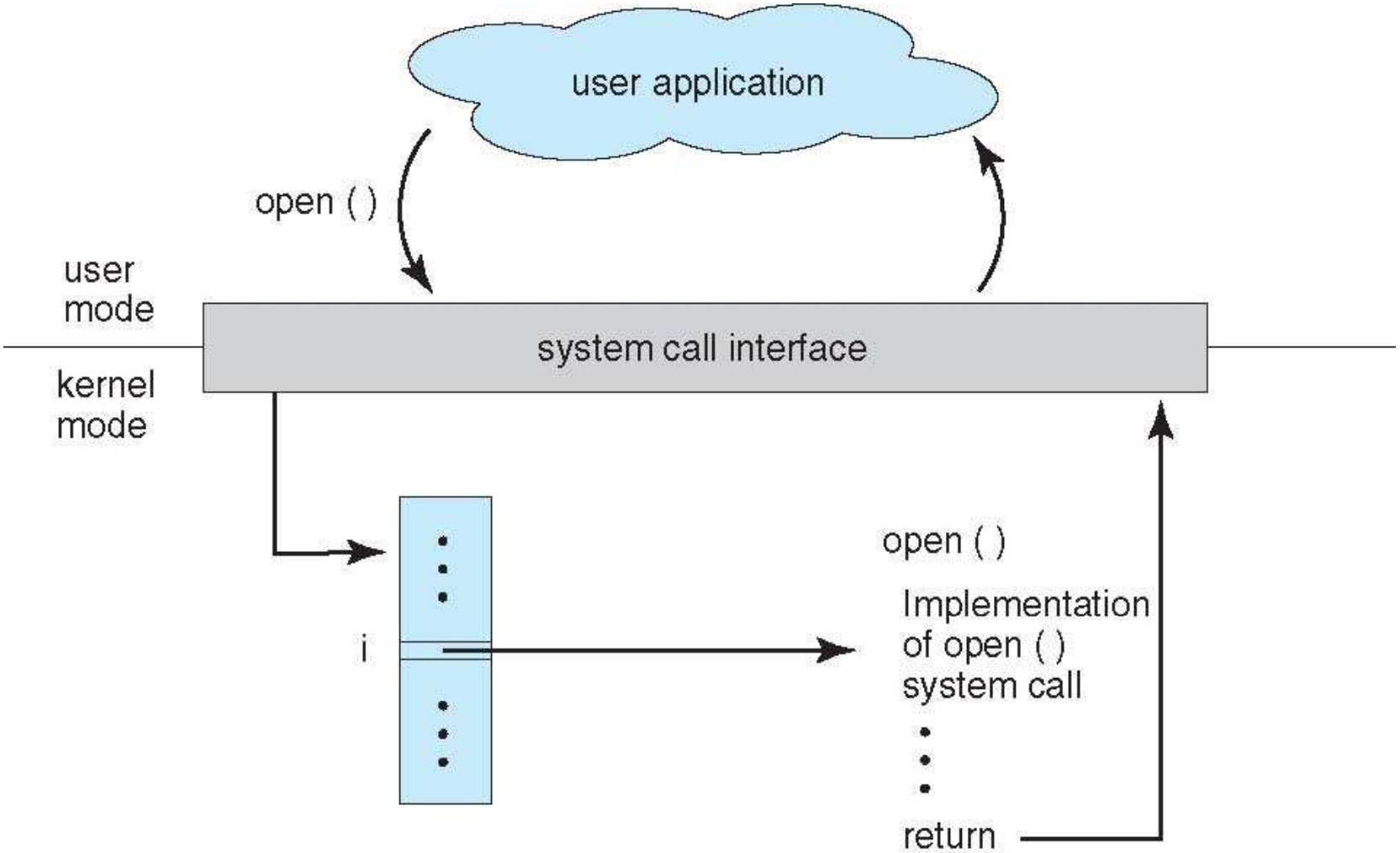
A program that uses the `read()` function must include the `unistd.h` header file, as this file defines the `ssize_t` and `size_t` data types (among other things). The parameters passed to `read()` are as follows:

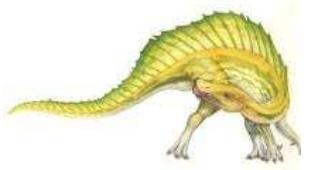
- `int fd`—the file descriptor to be read
- `void *buf`—a buffer where the data will be read into
- `size_t count`—the maximum number of bytes to be read into the buffer

On a successful read, the number of bytes read is returned. A return value of 0 indicates end of file. If an error occurs, `read()` returns −1.



API – System Call – OS Relationship

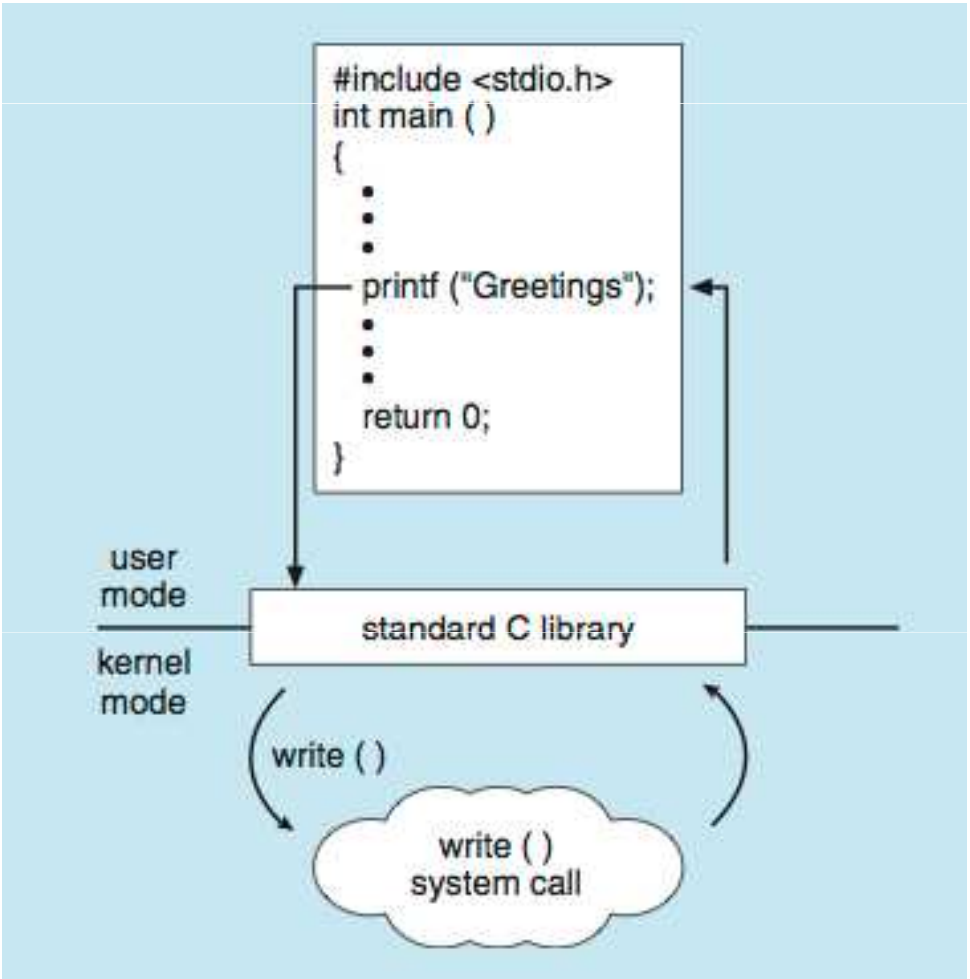


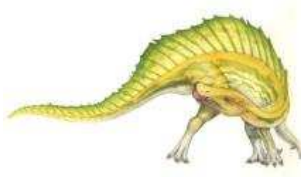


Source: Operating System Concepts. A. Silberschatz. Fig. 2.6

Standard C Library Example

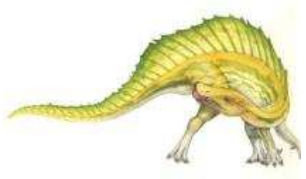
- C program invoking printf() library call, which calls write() system call





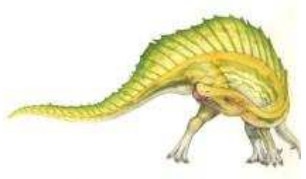
Types of System Calls

- File management
  - create file, delete file
  - open, close file
  - read, write, reposition
  - get and set file attributes
- Device management
  - request device, release device
  - read, write, reposition
  - get device attributes, set device attributes
  - logically attach or detach devices



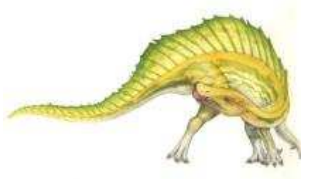
Types of System Calls (Cont.)

- Information maintenance
  - get time or date, set time or date
  - get system data, set system data
  - get and set process, file, or device attributes
- Communications
  - create, delete communication connection
  - send, receive messages if **message passing model** to **host name** or **process name**
    - From **client** to **server**
  - **Shared-memory model** create and gain access to memory regions
  - transfer status information
  - attach and detach remote devices



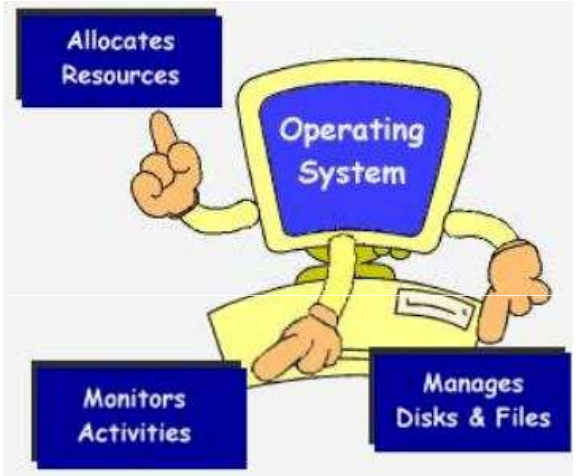
Types of System Calls (Cont.)

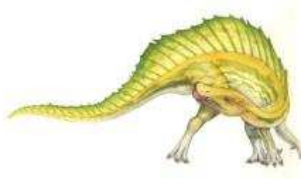
- Protection
  - Control access to resources
  - Get and set permissions
  - Allow and deny user access



Operating System Services

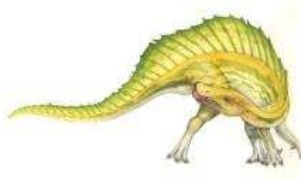
- Services/Functions:
  - **Resource Manager**
    - Memory Management
    - I/O Management
    - CPU Scheduling
  - **Abstract Machine**
    - Operating System Services: system calls/API
  - **Interface between User and Machine**
    - User interface: to control of the machine on the user's end, and to receive feedback from the machine
      - Graphics User Interface (GUI) or Command-Line (CLI).





User Operating System Interface

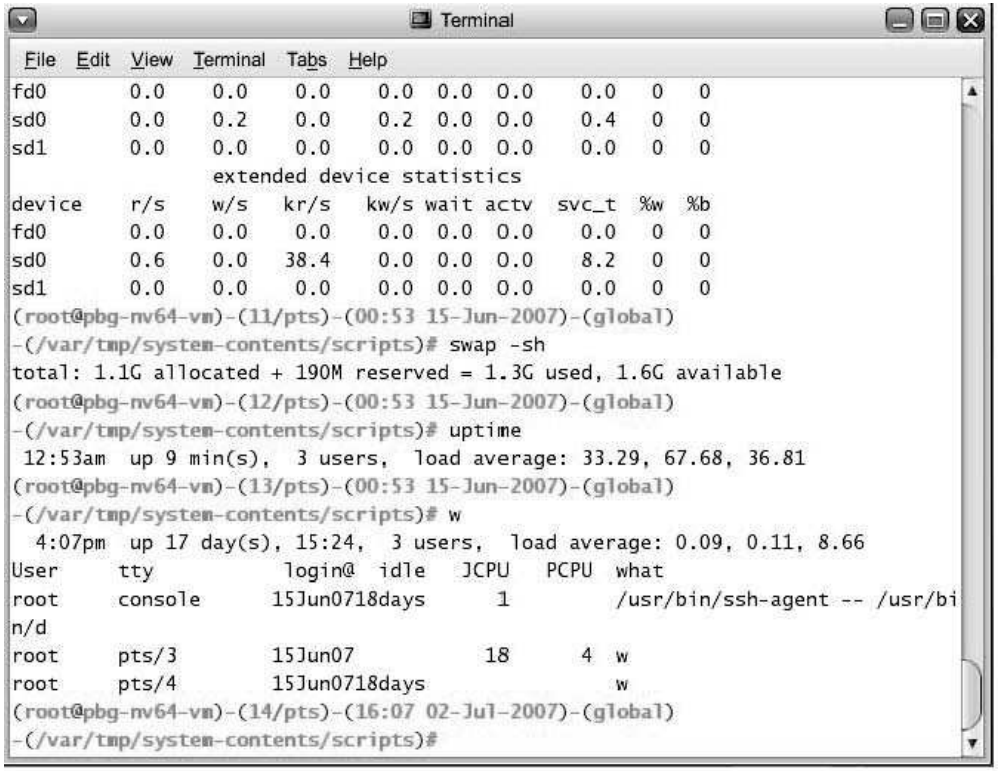
- Serves as the interface between the user and the OS
  - Graphics User Interface (GUI). User friendly, mouse based windows environment in the Macintosh and in Microsoft Windows
  - Command-Line (CLI). In MS-DOS and UNIX, commands are typed on a keyboard and displayed on a screen or printing terminal with the Enter or Return key indicating that a command is complete and ready to be executed
- 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

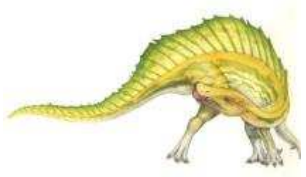


User Operating System Interface - CLI

Command Line Interface (CLI) or command interpreter allows direct command entry:

- Sometimes implemented in kernel, sometimes by systems program
- Sometimes multiple flavors implemented – shells
- Primarily fetches a command from user and executes it
- Sometimes commands built-in, sometimes just names of programs
  - If the latter, adding new features doesn't require shell modification

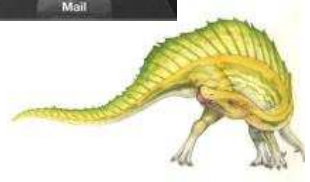




Source: Operating System Concepts. A. Silberschatz. Fig. 2.2

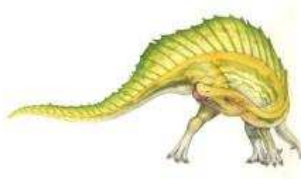
Touchscreen Interfaces

- Touchscreen devices require new interfaces
  - Mouse not possible or not desired
  - Actions and selection based on gestures
  - Virtual keyboard for text entry
- Voice commands.



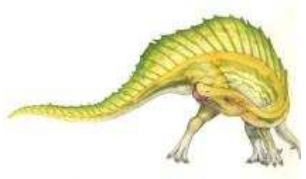
Contents

- What Operating Systems Do
- History of Operating Systems
- Operating Systems Services/Functions
- **Operating Systems Structure and Implementation**
- Hardware Protection/Support



Implementation

- Much variation
  - Early OSes in assembly language
  - Then system programming languages like Algol, PL/1
  - Now C, C++
- Actually usually a mix of languages
  - Lowest levels in assembly
  - Main body in C
  - Systems programs in C, C++, scripting languages like PERL, Python, shell scripts
- More high-level language easier to **port** to other hardware
  - But slower



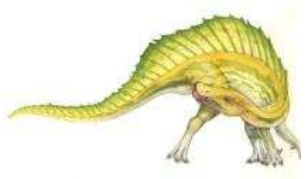
Operating System Structure

- General-purpose OS is very large program
- Various ways to structure them
  - Simple structure – MS-DOS
  - More complex -- UNIX

  } Monolithic approach

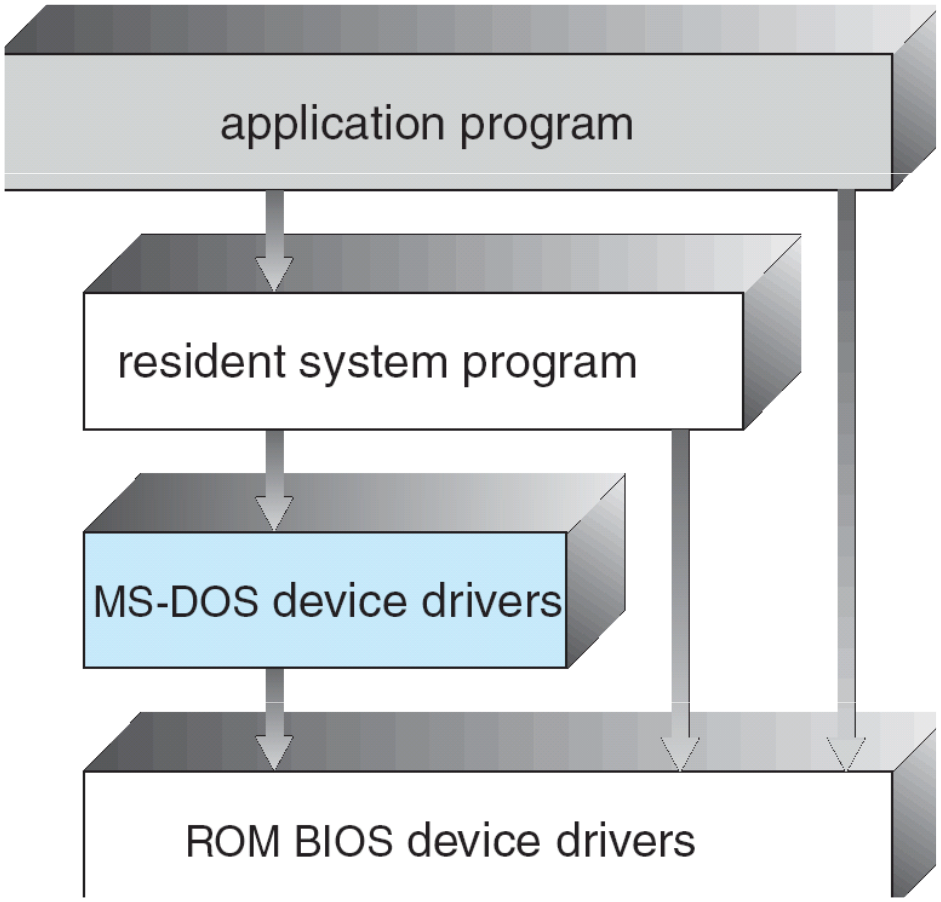
  - Layered – an abstraction
  - Microkernel -Mach

  } Layer/module approach

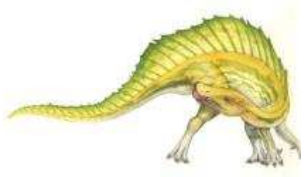


Simple Structure

- ***MS-DOS*** – written to provide the most functionality in the least space
  - Not well divided into modules
  - Designed without realizing it was going to become so popular
  - Although MS-DOS has some structure, its interfaces and levels of functionality are not well separated
  - Vulnerable to system crashes when a user program fails
  - Written for Intel 8088 which had no dual mode or hardware protection

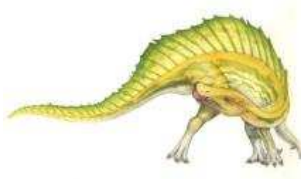


MS-DOS layer structure

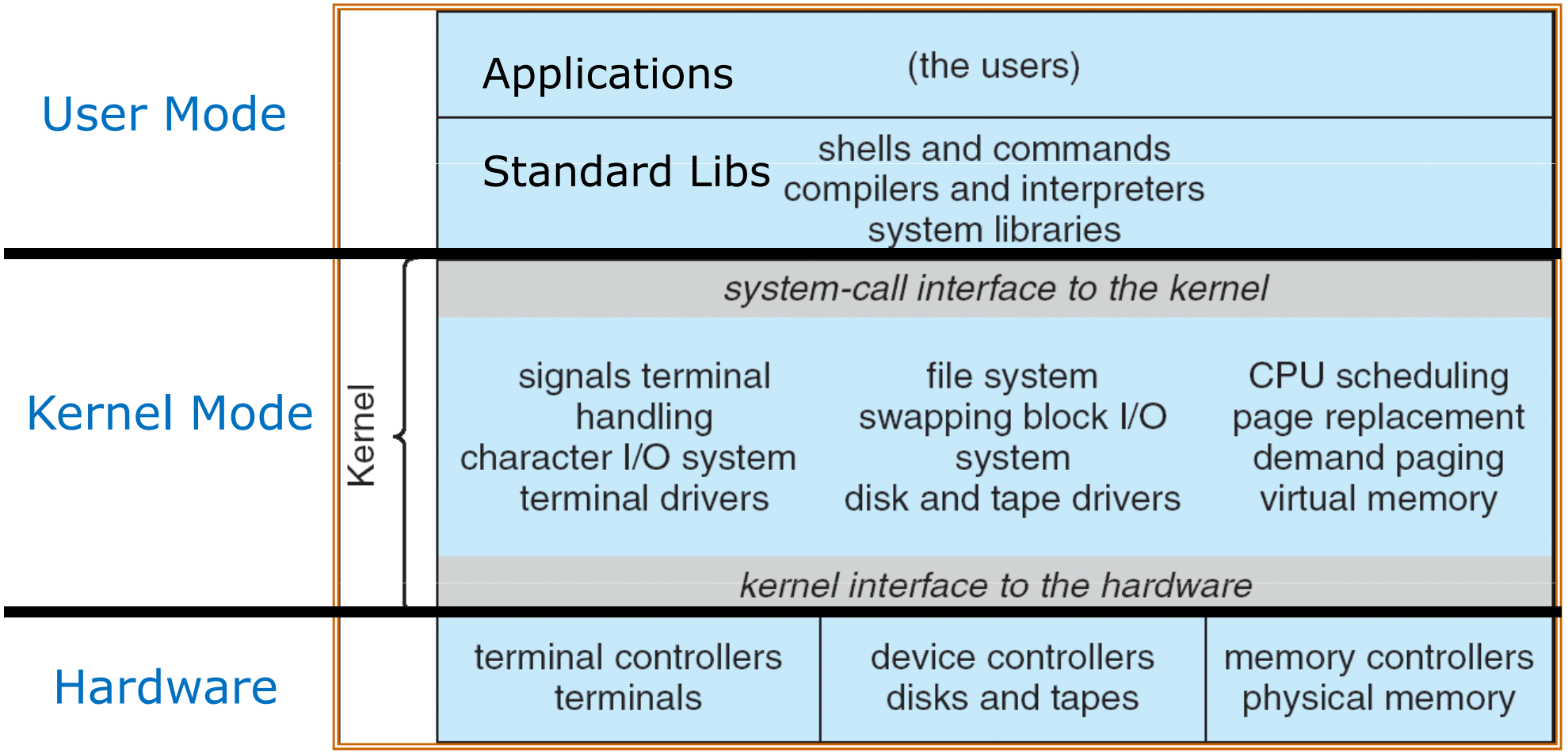


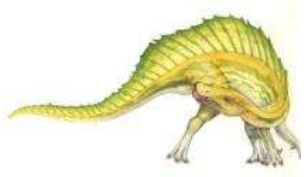
Operating System Structure

- ***UNIX*** – the original UNIX operating system had limited structuring due to limited hardware functionality
- The UNIX OS consists of two separable parts
  - Systems programs
  - The kernel – series of interfaces and device drivers
    - 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
    - All this functionality in one level makes UNIX difficult to enhance



Traditional UNIX System Structure

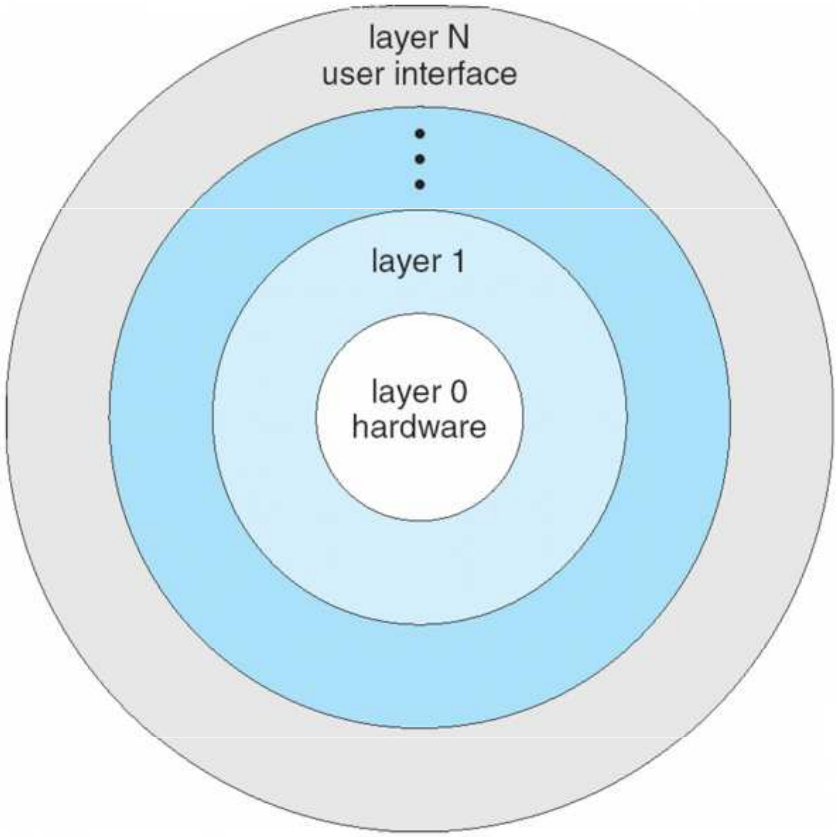




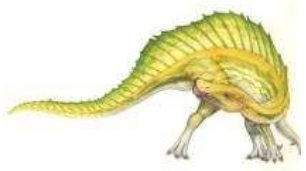
Source: Operating System Concepts. A. Silberschatz. Fig. 2.12

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

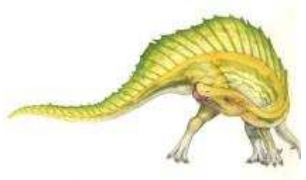


An operating system layer

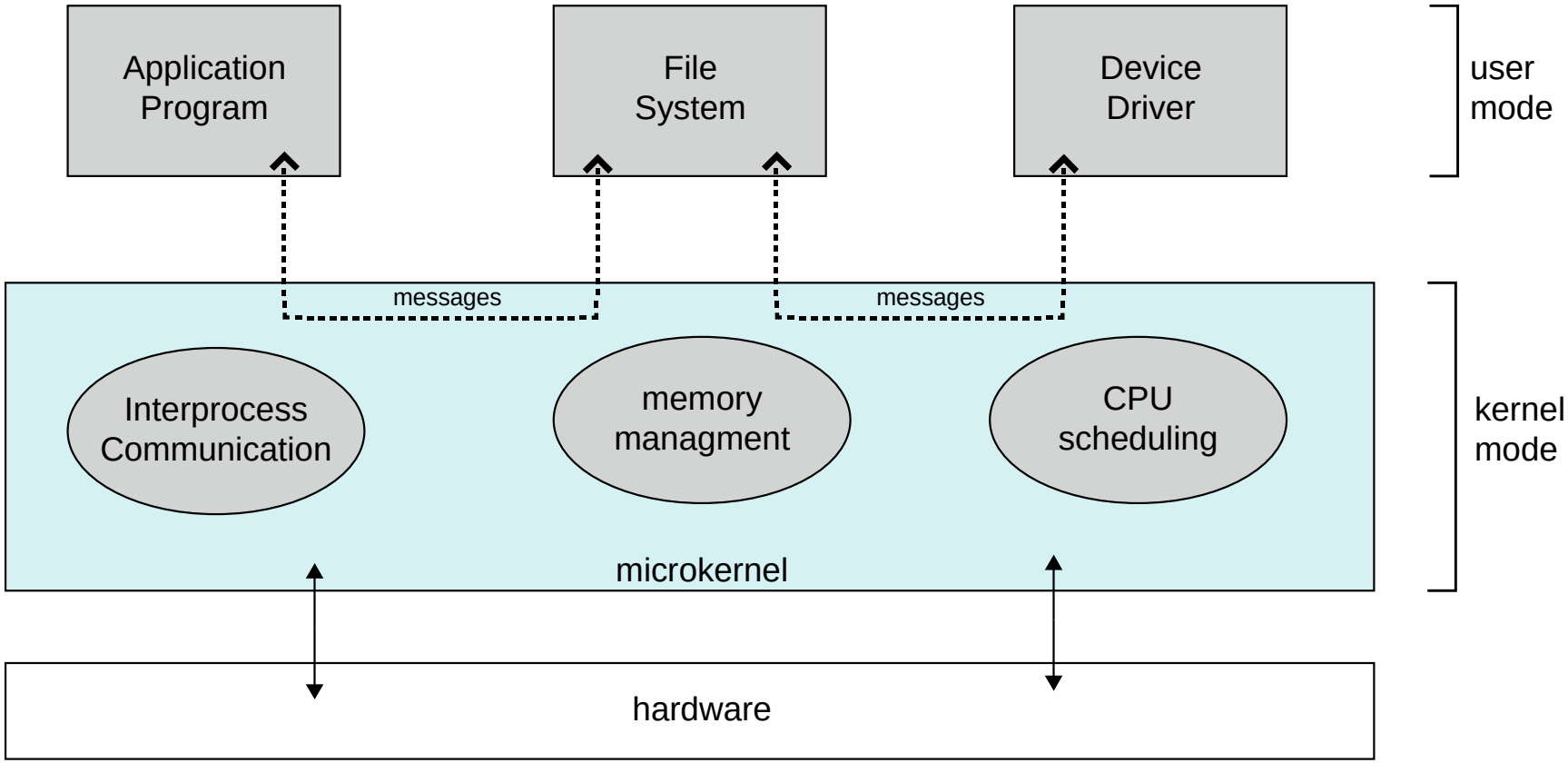


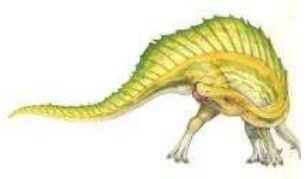
Microkernel System Structure

- Moves as much from the kernel into user space
- **Mach** example of **microkernel**
  - Mac OS X kernel (**Darwin**) partly based on Mach
- Communication takes place between user modules using **message passing**
- 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



Microkernel System Structure

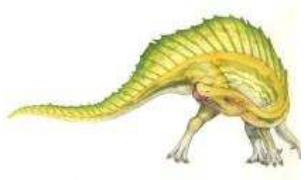




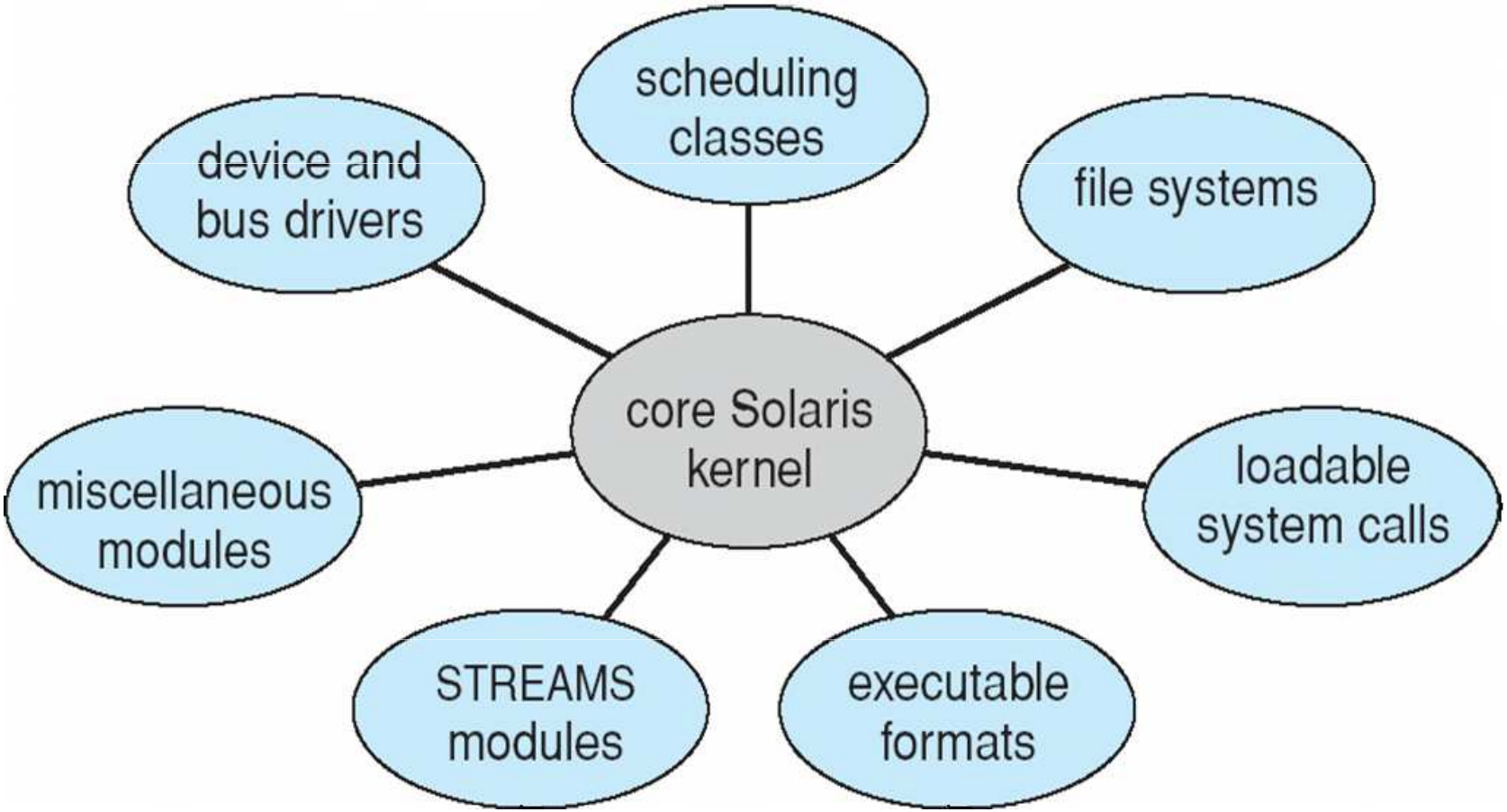
Source: Operating System Concepts. A. Silberschatz. Fig. 2.14

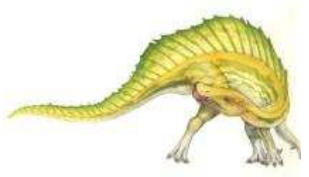
Modules

- Many modern operating systems implement **loadable kernel modules**
  - 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 flexibility
  - Linux, Solaris, etc



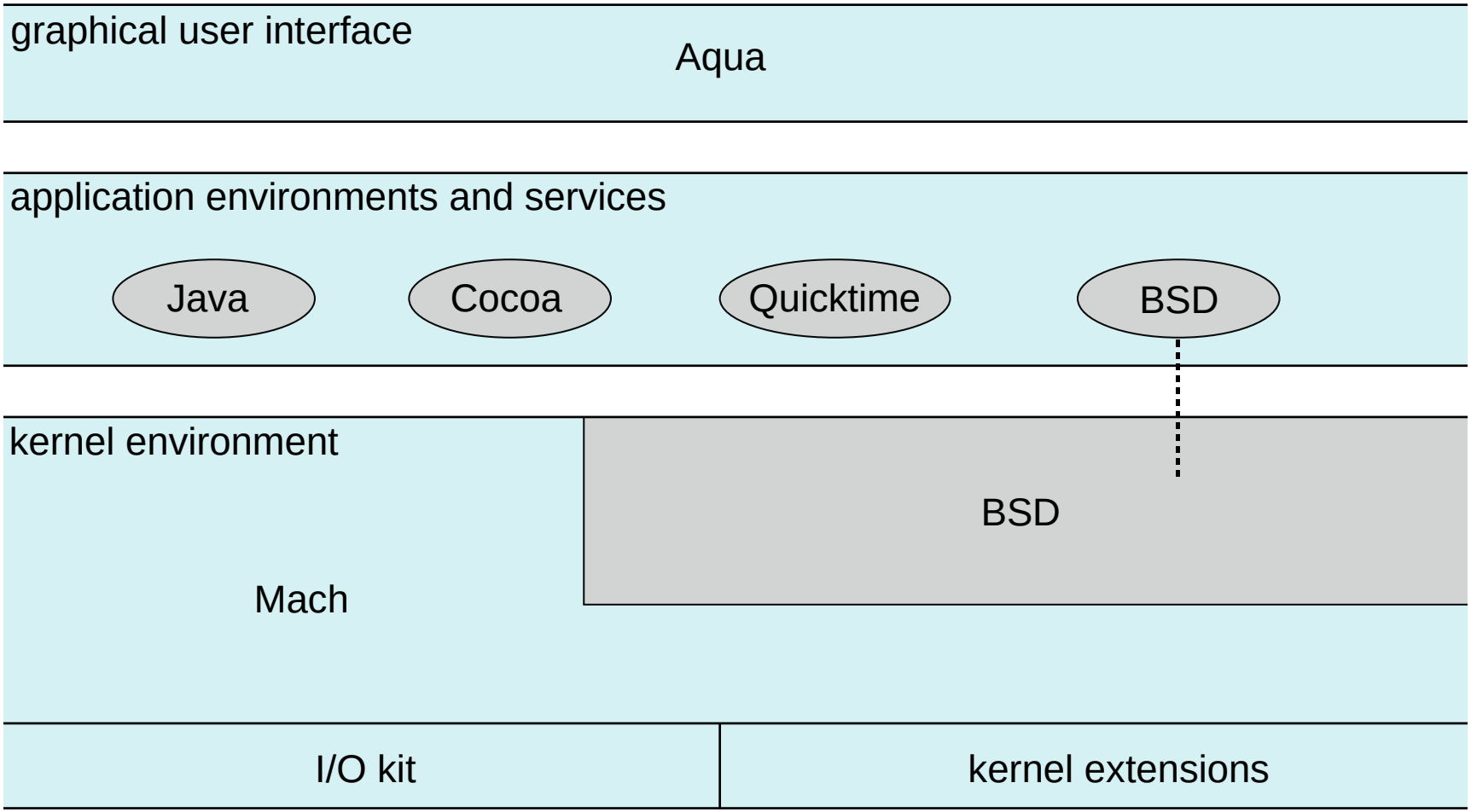
Solaris Modular Approach



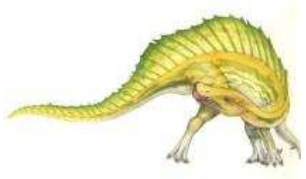


Source: Operating System Concepts. A. Silberschatz. Fig. 2.15

Mac OS X Structure



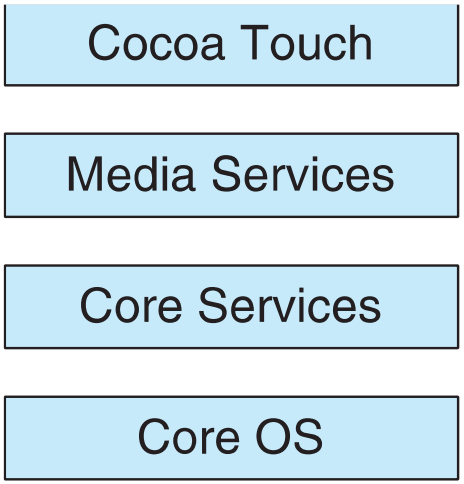
Hybrid structure



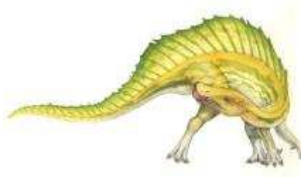
Source: Operating System Concepts. A. Silberschatz. Fig. 2.16

iOS

- Apple mobile OS for ***iPhone***, ***iPad***
  - Structured on Mac OS X, added functionality
  - Does not run OS X applications natively
    - Also runs on different CPU architecture (ARM vs. Intel)
  - **Cocoa Touch** Objective-C API for developing apps
  - **Media services** layer for graphics, audio, video
  - **Core services** provides cloud computing, databases
  - Core operating system, based on Mac OS X kernel



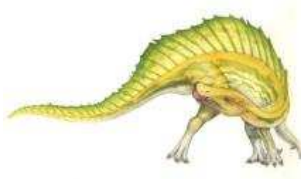
Architecture of Apple's iOS



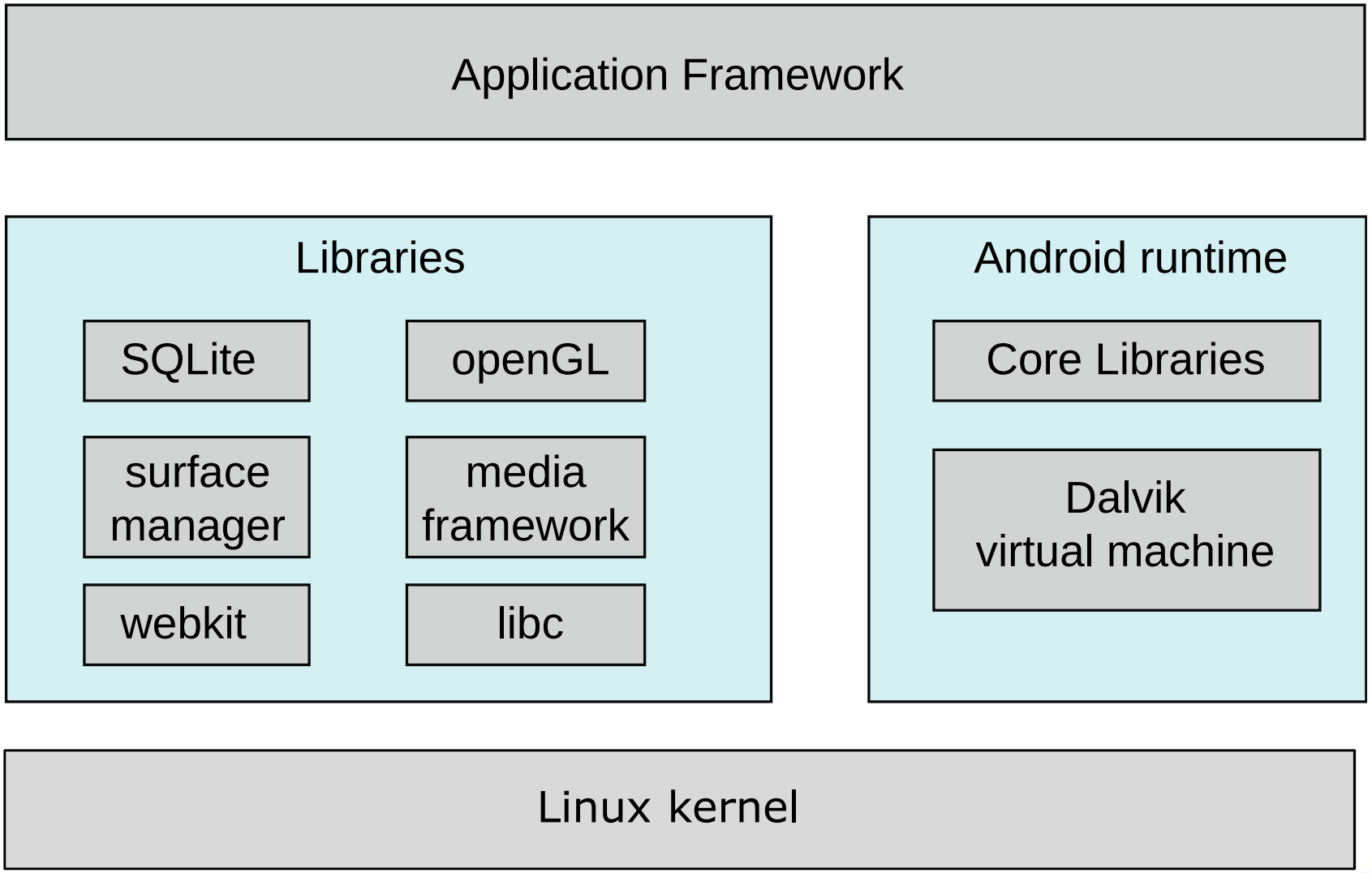
Source: Operating System Concepts. A. Silberschatz. Fig. 2.17

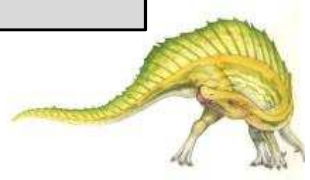
Android

- Developed by Open Handset Alliance (mostly Google)
  - Open Source
- Similar stack to IOS
- Based on Linux kernel but modified
  - Provides process, memory, device-driver management
  - Adds power management
- Runtime environment includes core set of libraries and Dalvik virtual machine
  - Apps developed in Java plus Android API
    - Java class files compiled to Java bytecode then translated to executable than runs in Dalvik VM
- Libraries include frameworks for web browser (webkit), database (SQLite), multimedia, smaller libc



Android Architecture

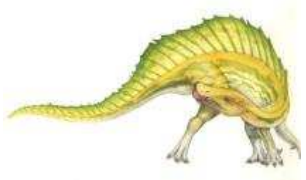




Source: Operating System Concepts. A. Silberschatz. Fig. 2.18

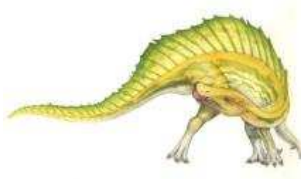
Contents

- What Operating Systems Do
- History of Operating Systems
- Operating Systems Services/Functions
- Operating Systems Structure and Implementation
- **Hardware Protection/Support**



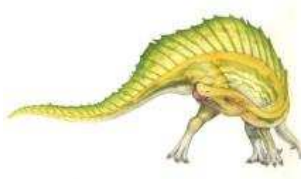
Hardware Protection/Support

- Most of the OS functions are implemented by SW
  - But some of them:
    - Impossible to implement by SW
    - Would be very slow if only implemented by SW
- OS and users share the hardware and software resources of the computing system:
  - process problems include infinite loop, processes modifying each other or the operating system
- Protection mechanisms:
  - Dual mode Operation
  - CPU protection
- HW support:
  - Interrupts
  - Direct Access Memory (DMA)
  - Memory Hierarchy



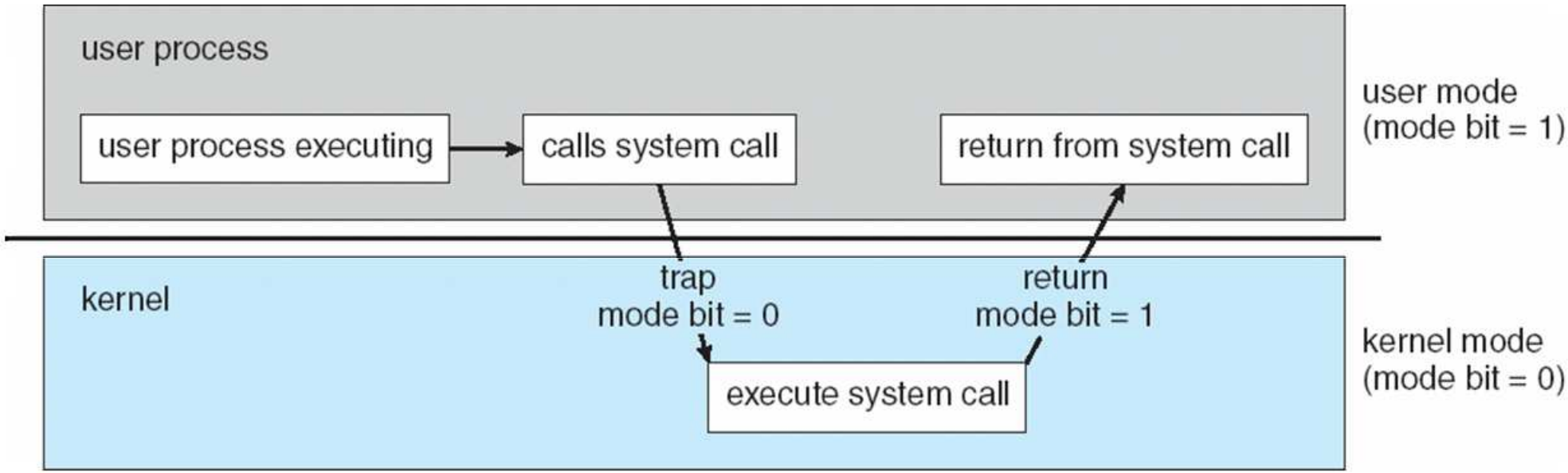
Dual-Mode Operation

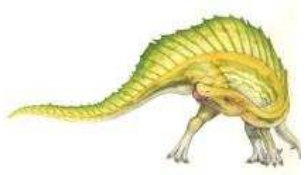
- Sharing system resources requires operating system to ensure that an incorrect program cannot cause other programs to execute incorrectly as well as to protect the OS from errant user programs
- Provide hardware support to differentiate between at least two modes of operations.
  - User mode – execution done on behalf of a user.
  - Monitor mode (also kernel mode or system mode) – execution done on behalf of operating system.
- MS-DOS for the Intel 8088 had no mode bit
  - User program can wipe out the OS
  - Multiple programs can write to a device simultaneously



Dual-Mode Operation

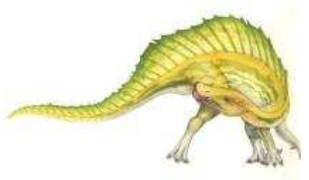
- Mode bit added to computer hardware to indicate the current mode:
  kernel (0) or user (1).
- When an interrupt or trap (program error) occurs hardware switches to monitor mode.





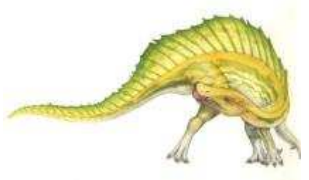
CPU Protection

- Timer – interrupts computer after specified period to ensure operating system maintains control
  - Timer is decremented every clock tick.
  - When timer reaches the value 0, an interrupt occurs.
- Timer commonly used to implement time sharing
  - interrupt every N milliseconds (a time slice) at which time a context switch occurs
    - Performs housekeeping on the program just stopped, saves registers, internal variables, buffers etc. and prepares for the next program to run
- Load-timer is a privileged instruction



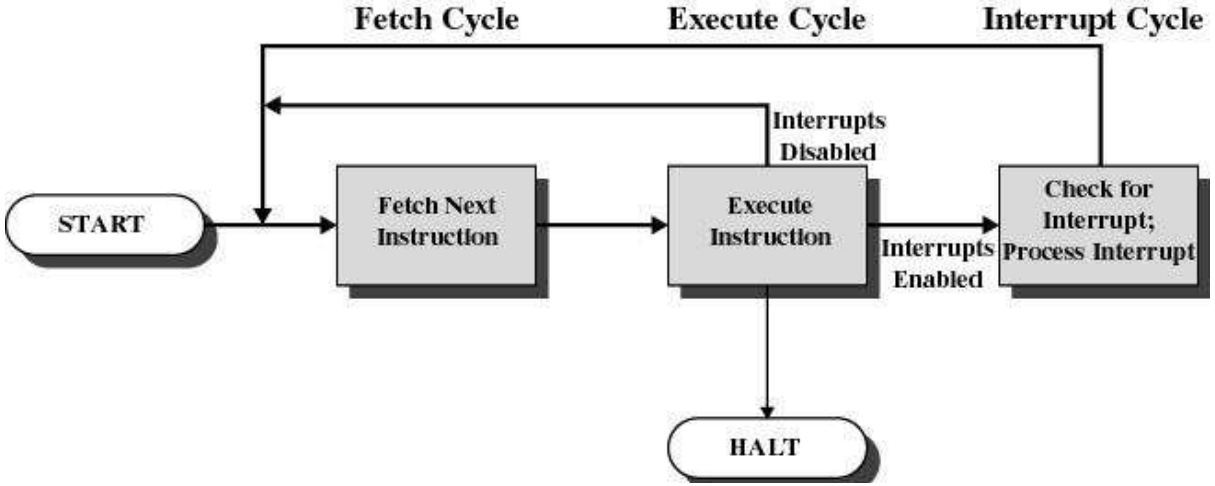
Interrupts

- Interrupts enable software to respond to signals from hardware
  - **Interrupt** driven by hardware
    - Key is pressed on a keyboard or a mouse is moved
  - **Interrupt** driven by software
    - May be initiated by a running process
    - Interrupt is called a trap – software generated caused by error or user request for an OS service (system call)
    - Dividing by zero or referencing protected memory
- Interrupts improve the system efficiency
  - I/O devices and the CPU can execute concurrently

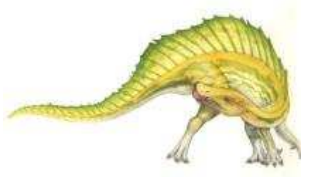


Interrupt Handling

- After receiving an interrupt, the processor completes execution of the current instruction, then pauses the current process:



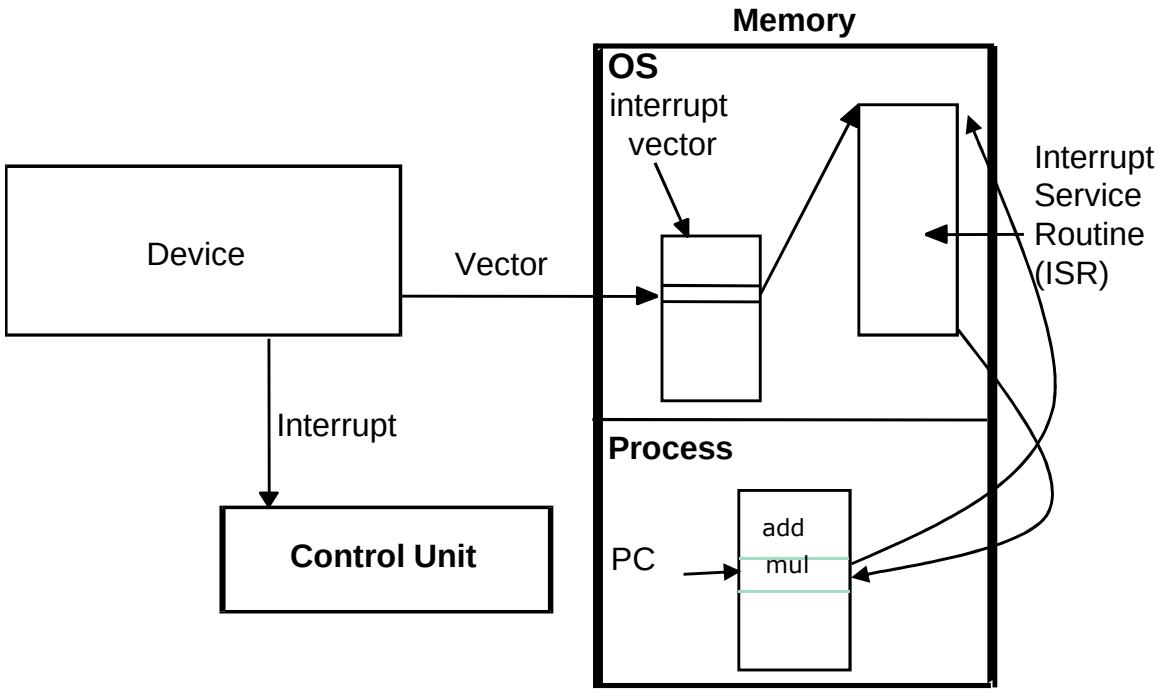
- Interrupt architecture must save the address of the interrupted instruction
- The processor will then transfer to a fixed location and executes the service routine for the interrupt:
  - The interrupt handler determines how the system should respond
- Determines which type of interrupt has occurred:
  - **vectored** interrupt system:
    - Interrupt transfers control to the interrupt service routine generally, through the **interrupt vector**, which contains the addresses of all the service routines

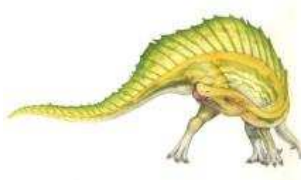


Source: Operating Systems. W. Stallings Fig. 1.7

Interrupt Handling

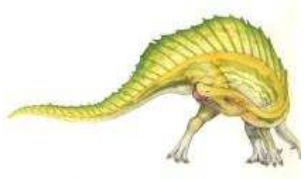
- Interrupt handlers are stored in an array of pointers called the interrupt vector
  - To handle the interrupt quickly, a table of pointers is generally stored in low memory which hold the addresses of the ISR for the various devices
  - This array, or interrupt vector, of addresses is then indexed by a unique device number to provide the address of the ISR for the interrupting device
- After the interrupt handler completes, the interrupted process is restored and execution continues from the address of the interrupted instruction (stored on stack) or the next process is executed





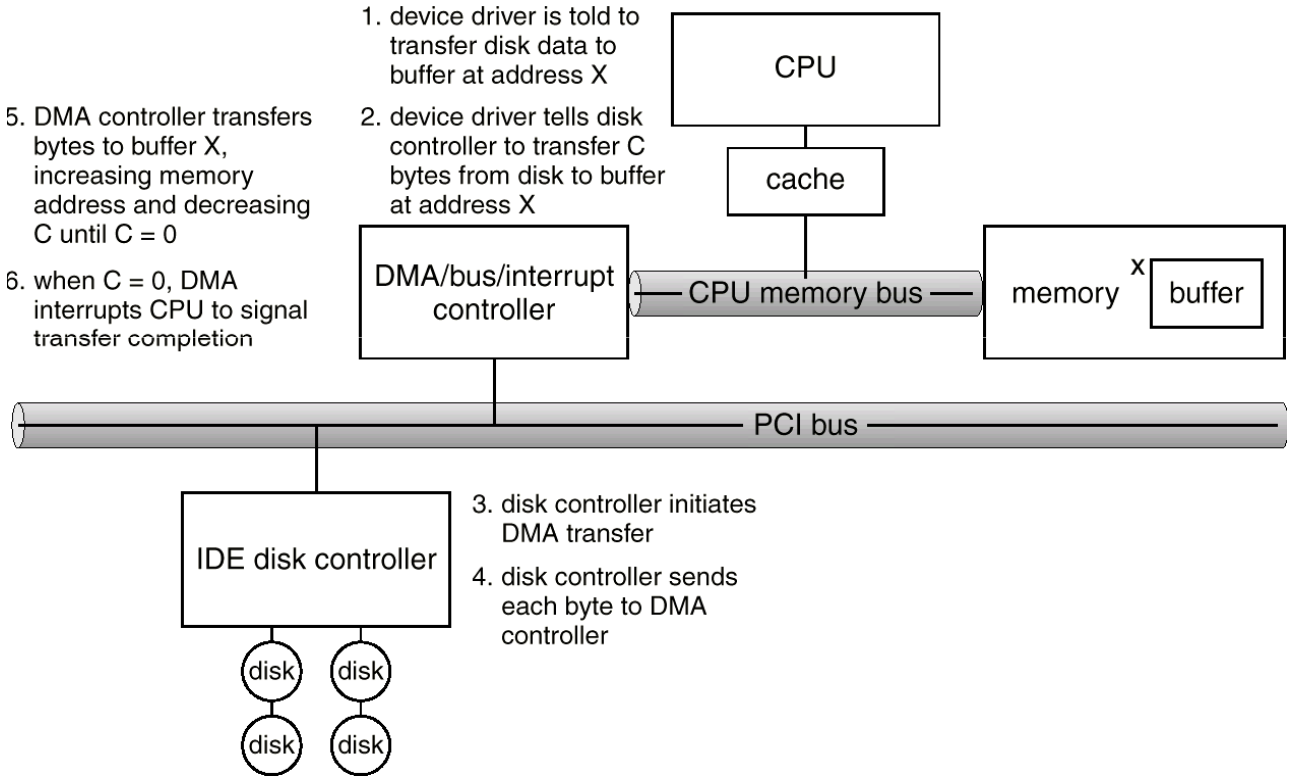
Direct Memory Access Structure

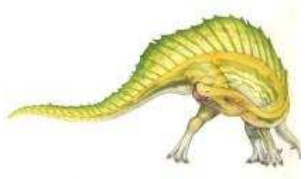
- DMA: Used for high-speed I/O devices able to transmit information at close to memory speeds
- Device controller transfers blocks of data from buffer storage directly to main memory without CPU intervention
  - Processor is free to do other things
  - Transfers a block of data directly to or from memory
- Only one interrupt is generated per block, rather than the one interrupt per byte
  - An interrupt is sent when the task is complete
- The processor is only involved at the beginning and end of the transfer



Direct Memory Access Structure

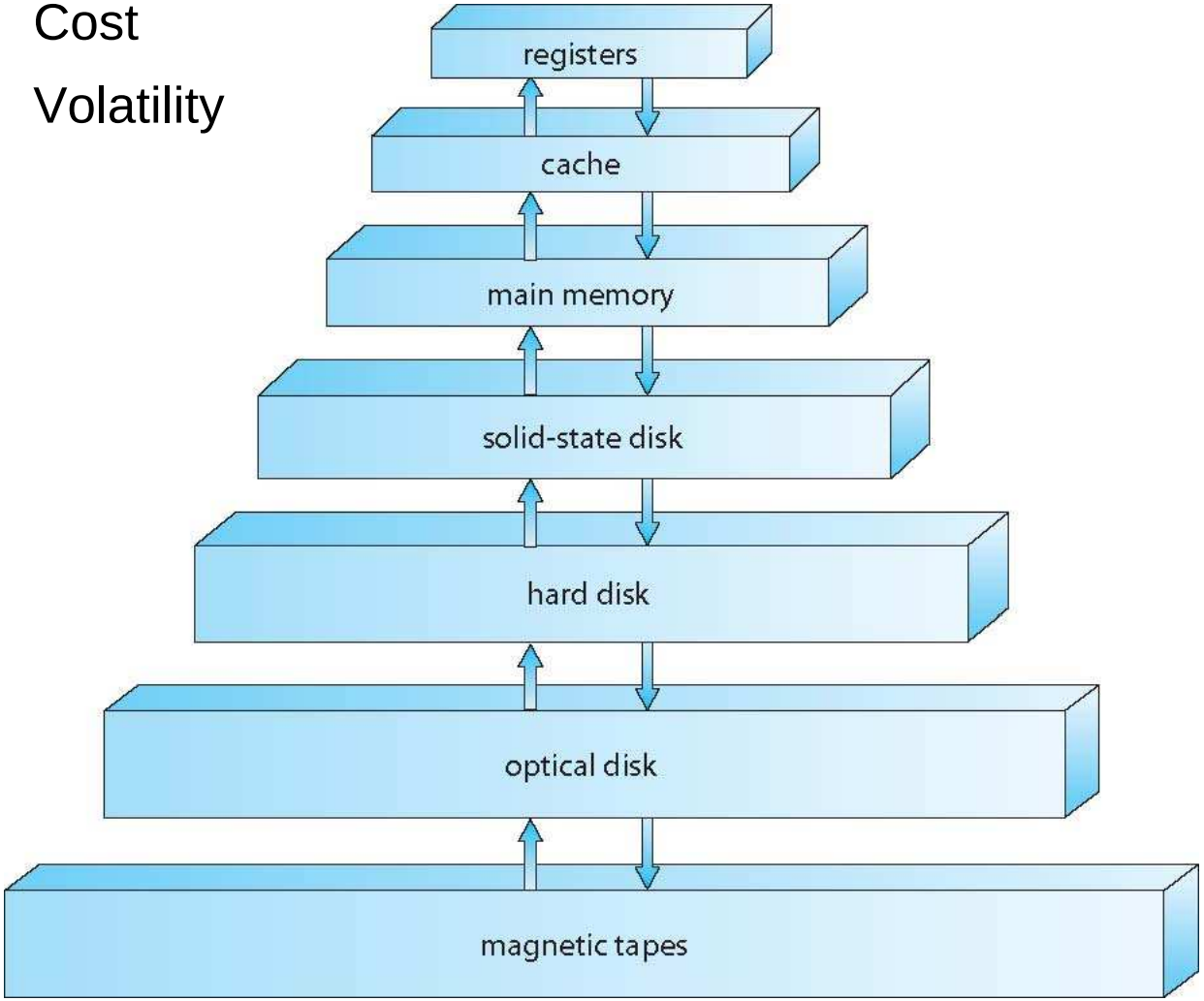
- Processor sends information to DMA to start and doesn't hear from DMA until transfer is completed at which time is receives an interrupt
- **While DMA is transferring, CPU is free to perform other tasks**
  - The DMA controller seizes the memory bus to transfer a word of data to memory, This is called "stealing" memory cycles from the CPU as the CPU can not access main memory at that time (though the primary and secondary cache are accessible). However, overall total system performance is improved.

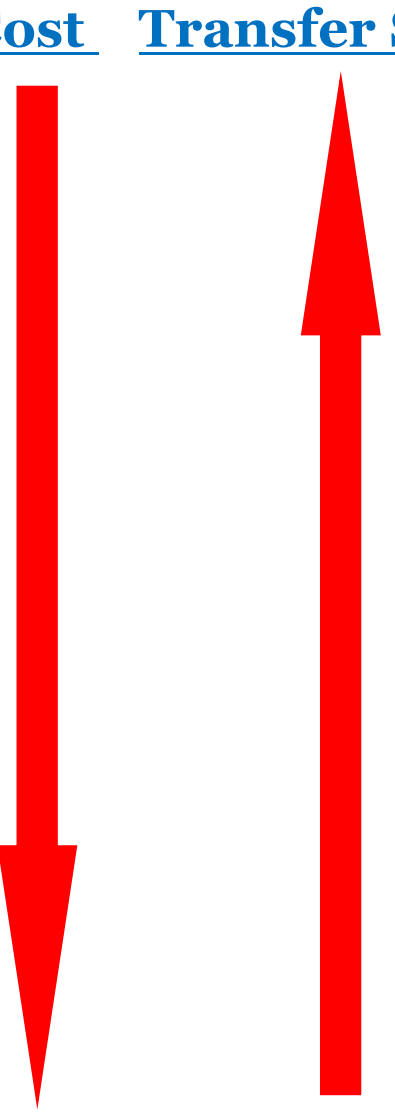


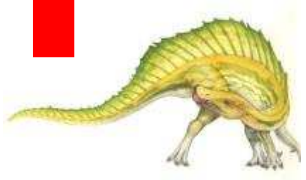


Storage-Device Hierarchy

- Storage systems organized in hierarchy:
  - Speed
  - Cost
  - Volatility

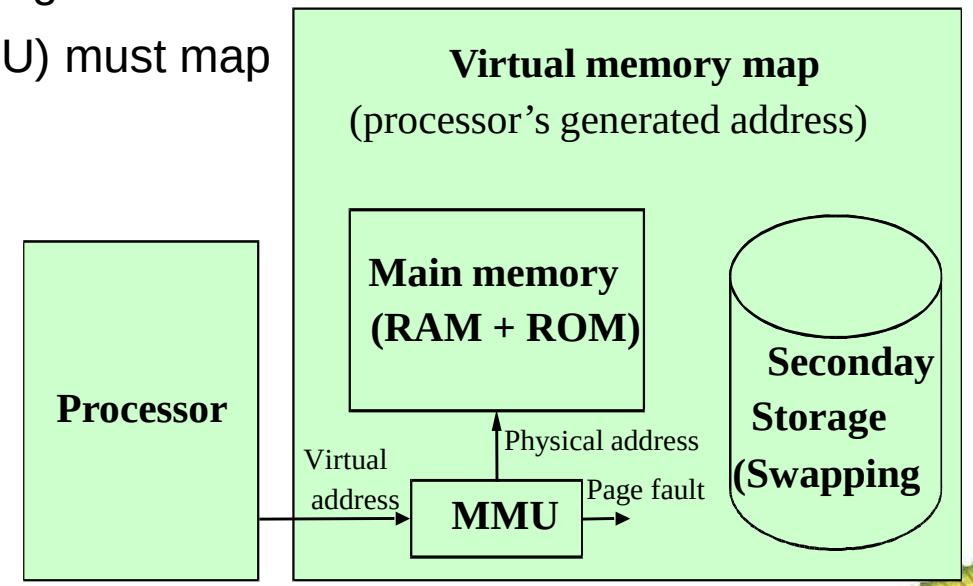


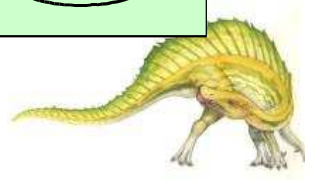




Virtual Memory

- **Virtual memory** allows execution of processes not completely in memory
- There is a separation of user logical memory from physical memory
  - Only part of the program needs to be in memory for execution
  - Logical address space can therefore be much larger than physical address space
  - More programs running concurrently
  - Physical memory organized in page frames
  - Memory Management Unit (MMU) must map logical to physical addresses





Bibliography

- J. CARRETERO, F. GARCÍA, P. DE MIGUEL, F. PÉREZ, **Sistemas Operativos. Una visión aplicada. 2ª Edición, Mc Graw-Hill, 2007.**
  - **Capítulos 1 y 2.**
- A. SILBERSCHATZ, P. GALVIN, G. GAGNE, **Operating System Concepts. 9ª Edición, Mc Graw Hill, 20014.**
  - **Capítulos 1 y 2.**
- A.S. TANEMBAUM, **Modern Operating Systems. 4ª Edición, Pearson, 2015.**
  - **Capítulo 1.**
- W. STALLINGS, **Operating Systems. 8ª Edición, Pearson, 2014.**