Operating Systems 2024/2025

T Class 02 – Introduction to Operating Systems

Vasco Pereira (vasco@dei.uc.pt)
Dep. Eng. Informática da Faculdade de Ciências e Tecnologia da Universidade de Coimbra

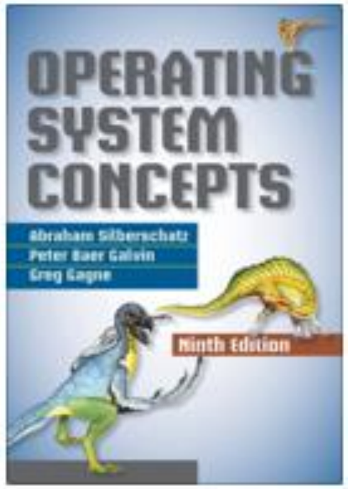
```
operating system
noun
the collection of software that directs a computer's operations, controlling and scheduling the execution of other
programs, and managing storage, input/output, and communication resources.

Abbreviation:  OS

                                                                                          Source: Dictionary.com
```

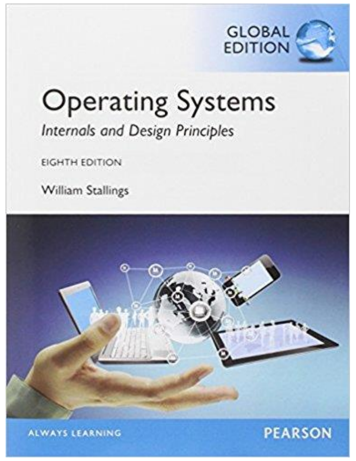
Updated on
12 February 2025

Disclaimer

- This slides and notes are based on the companion material [Silberschatz13]. The original material can be found at:
  - http://codex.cs.yale.edu/avi/os-book/OS9/slide-dir/
- In some cases, material from [Stallings15] may also be used. The original material can be found at:
  - http://williamstallings.com/OS/OS5e.html
  - http://williamstallings.com/OperatingSystems/
- The respective copyrights belong to their owners.

Note: Some slides are also based on previous versions from Bruno Cabral, Paulo Marques and Luis Silva (Operating Systems classes of DEI-FCTUC).

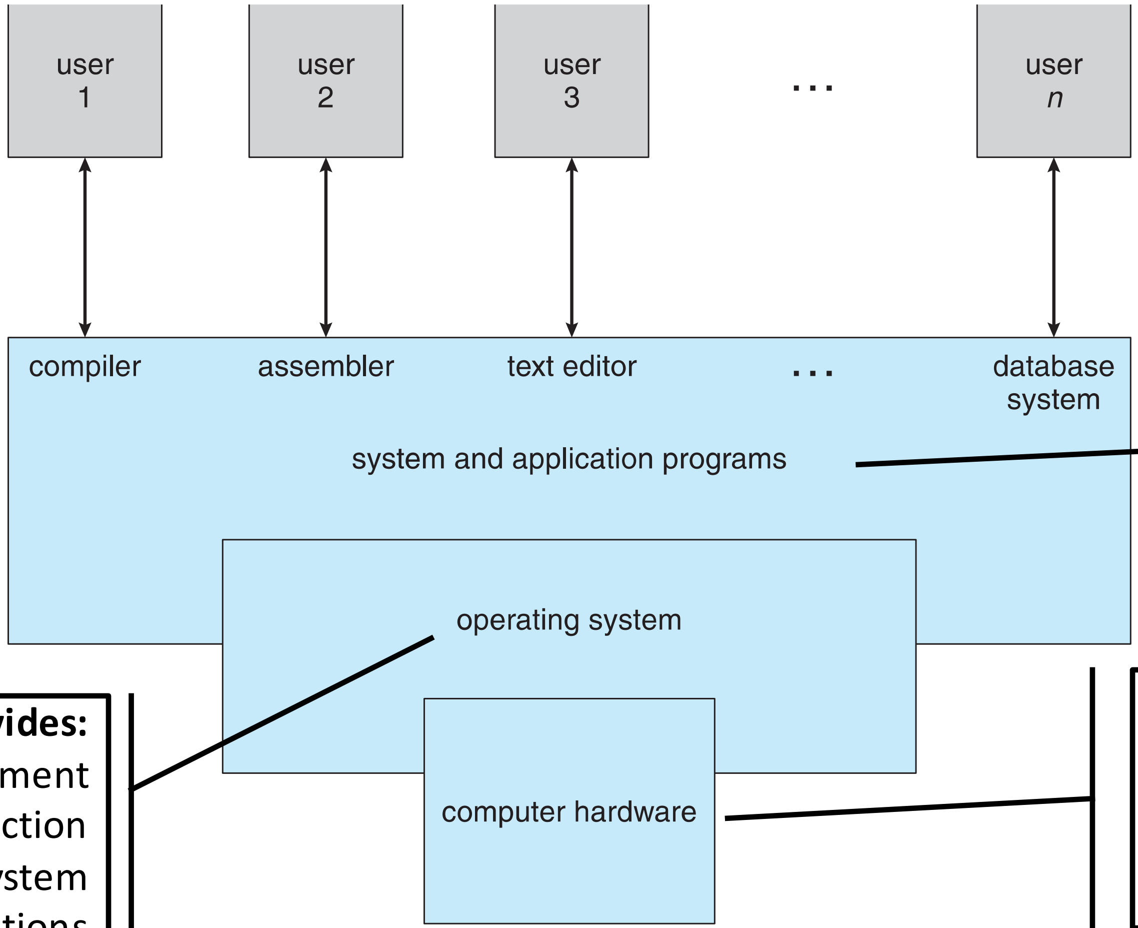




Introduction to Operating Systems
Outlines

- Describe the basic organization of a computer system
- Overview of OS components
- OS boot
- Overview of different types of computing environments
- OS services to users, processes and other systems

Operating system in a computer system

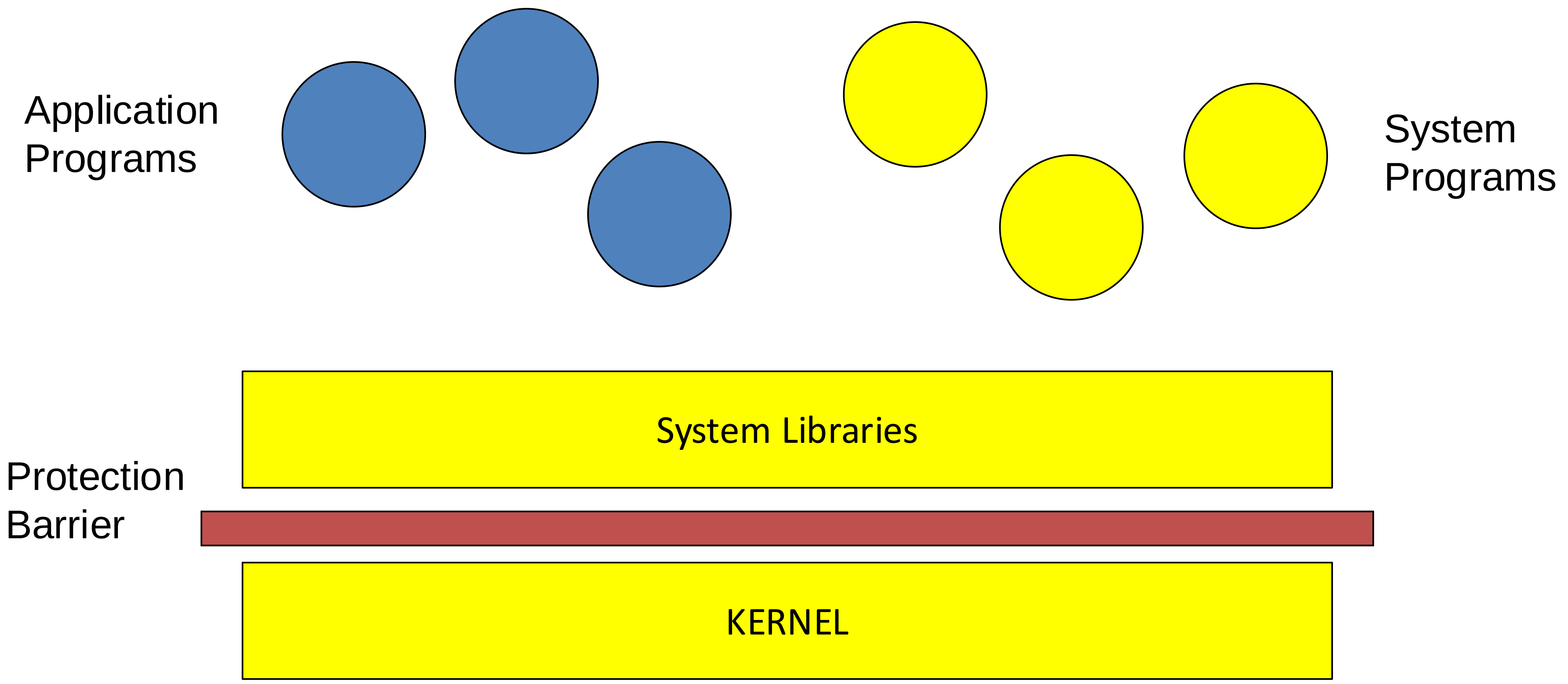


System programs: Associated with the operating system, but not part of its core
Application programs: Programs not associated with the operating system

Provides:
Device Management
Security & Protection
File System
Network communications
Process Management

Computing resources: CPU; Memory; I/O devices (Video Card, Monitor, Disk, Printer, Network card, etc.).

Simple Organization of an Operating System



Operating system in a computer system

- Many different operating systems exist, used in a wide variety of computing environments
  - Home appliances, cars, IoT, cloud servers, edge servers, smartphones, smartwatches, etc.

- Some computers have little or no user view (e.g., embedded computers)

What's an Operating System?

- The **Operating System** is a "special program" that allows the isolation of the hardware from the programs that run on the computer. It provides:
  - Memory management
  - Disk management
  - Peripherals management (keyboard, mouse, graphics card)
  - Management of users and programs, protecting the entire system
  - ... everything so that the programmer doesn't have to do it

What’s an Operating System?

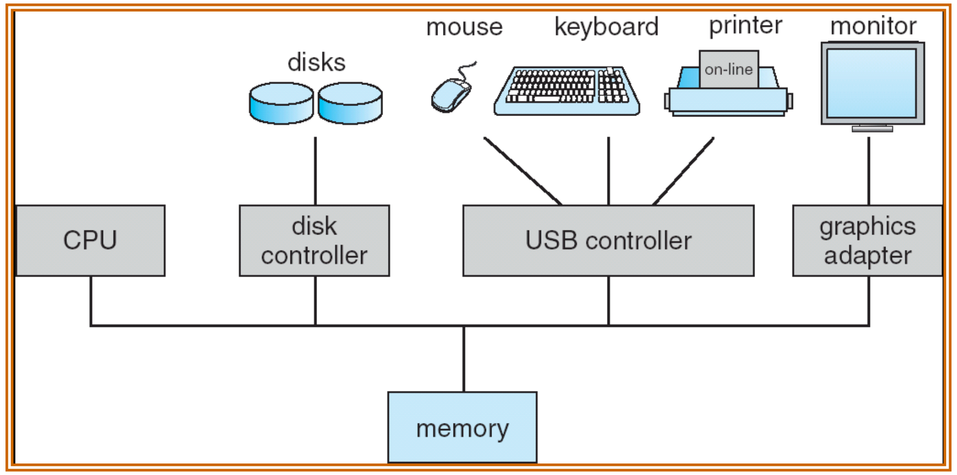
- OS is a control program
  - Controls execution of programs to prevent errors and improper use of the computer
- OS is a resource manager/allocator
  - Manages all resources
    - Process management
    - Memory management
    - File-system management
    - Mass-storage management
    - I/O management
  - Decides between conflicting requests for efficient and fair resource use

What's an Operating System?

- Different operating systems (OSs) may provide different functionalities
  - Some have 400 bytes (TinyOS), other GBytes!
  - Some have a basic text interface, other highly advanced touch screens
  - Some come with multiple system programs, other with just a small core of functionalities
- Usually, a general-purpose OS includes:
  - Kernel (the OS part that is always running in the machine)
  - Middleware frameworks (ease application development)
  - System programs that help to manage the system

What Operating Systems Do

- Sharing resources among applications
  - scheduling
  - allocation
- Making efficient use of limited resources
  - improving utilization
  - minimizing overhead
  - improving throughput/goodput
- Protecting applications from each other
  - enforcement of boundaries



Why study Operating Systems?

- Almost all code runs over on top of an operating system
- Knowledge about how an OS works is crucial to a proper, efficient, effective and secure programming
- And you can also one day create or modify one!

Classification of Operating Systems

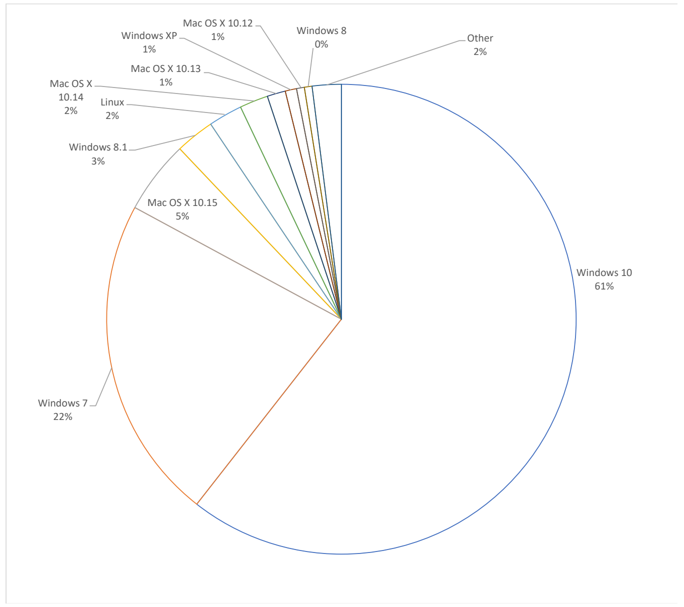
- Interface
  - CLI (Command-line interface)
  - Batch interface (commands are entered in files)
  - GUI (Graphical User Interface)
  - Voice recognition
- Users
  - Multi-user
  - Mono-user
- Tasks
  - Single-task - can support only a single task at any time; tasks are handled sequentially (e.g., MS-DOS)
  - Multi-task - executes more than one task at a time, by sharing resources
- Real-time Operating Systems (RTOS)
  - Operating systems that have strict time bounds and have a predictable behavior (e.g., VxWorks, VRTX, pSOS, LynxOS, …)

Classification of Operating Systems (cont.)

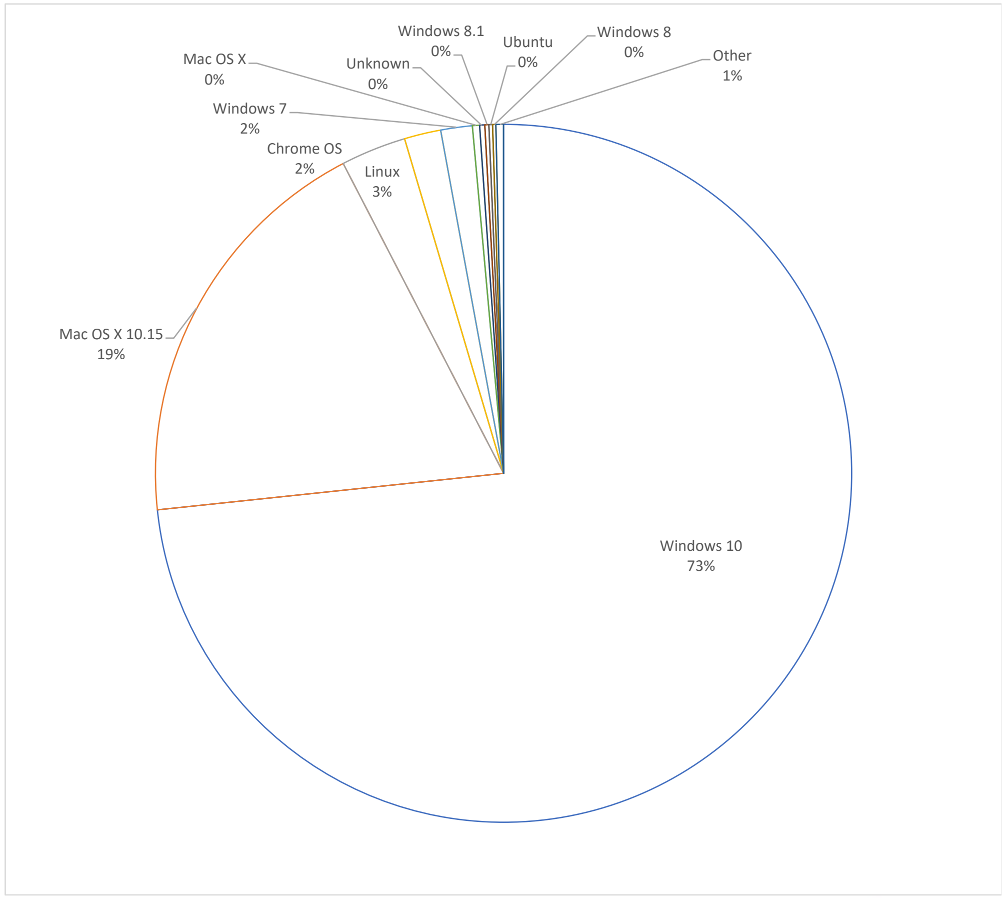
- Open/Closed Operating Systems:
  - Proprietary
  - Open systems
    - Have standardized programming interfaces and peripheral interconnects
  - Open-source systems (GPL license)
    - Operating systems made available in source-code format rather than just binary closed-source
    - Started by Free Software Foundation (FSF), which has “copyleft” GNU Public License (GPL)
      - (GNU = GNU's Not Unix!)
    - Examples include GNU/Linux and BSD UNIX (included in the core of Mac OS X), and many more

Desktop Operating System Market Share

August 2020



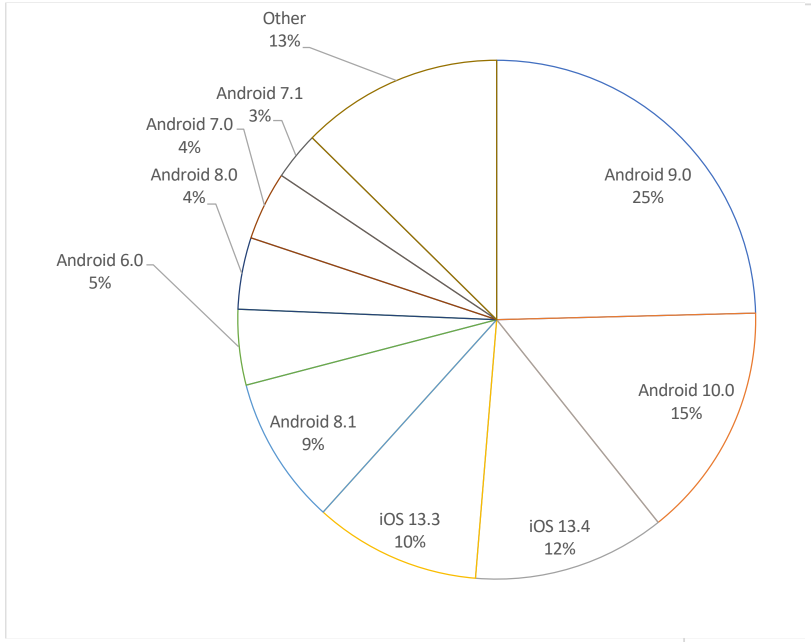
August 2024



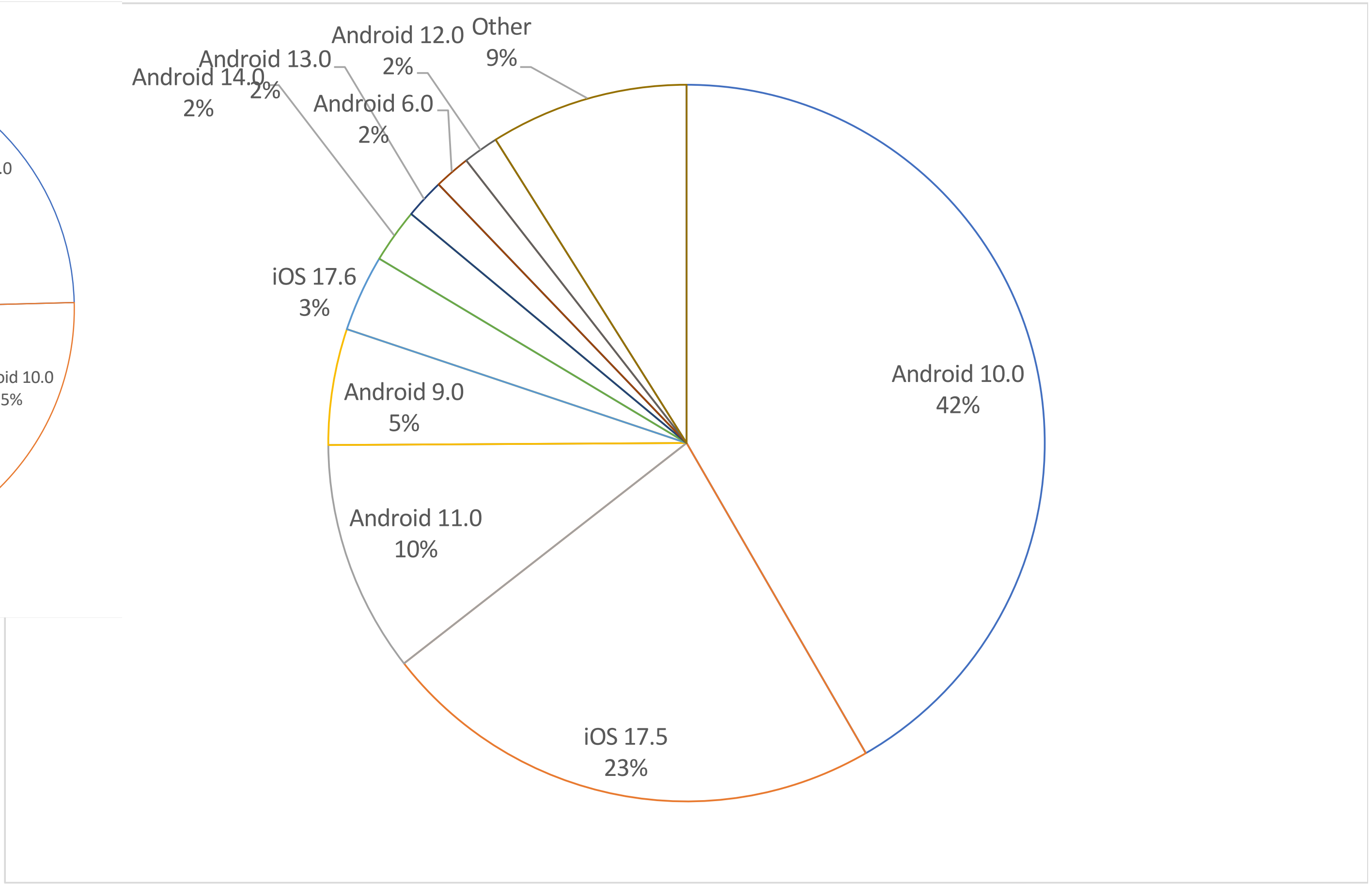
Source:
http://www.netmarketshare.com/operating-system-market-share.aspx

Mobile/tablet Operating System Market Share

August 2020



August 2024



Source:
http://www.netmarketshare.com/operating-system-market-share.aspx

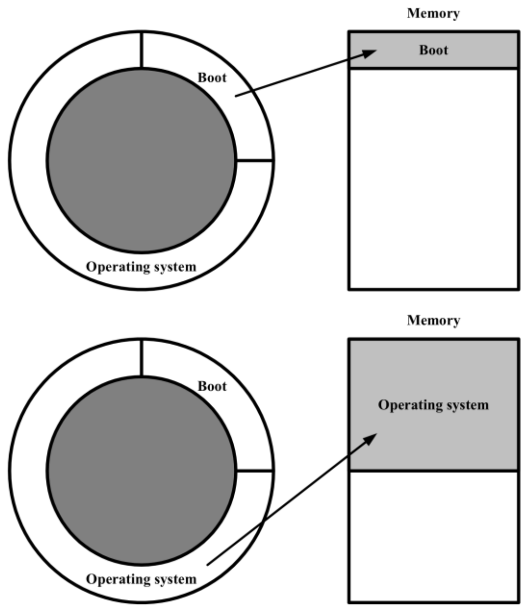
OS: Startup

- How does the hw know where the kernel is and how to load it?
- Starting a computer and loading the kernel is called booting.
- Most computers have a multistage boot

- Power up the computer (IBM - compatible PC)
  - Test operation (power-up diagnostics) :
    - In the power-up (and at reset) the CPU executes a jump instruction to a pre-defined address where an initial boot loader located in nonvolatile firmware (BIOS or UEFI) is run;
      - BIOS = Basic Input Output System
      - UEFI = Unified Extensible Firmware Interface
    - The initial boot loader program executes tests for verifying the correct operation of the CPU (subset of instructions) and RAM (basic tests)
      - Power-on self-test, also known as POST
    - If OK , a second boot loader is started ...

OS: Startup (2)

- Sequence startup (boot) OS :
  - A second boot loader is started, normally from the first sector of the boot device. This second boot loader may be able to start the OS, but normally it only knows the location of the remainder of the **bootstrap loader** program, which in turn locates the kernel, loads it into main memory, and starts its execution. .
    - Common bootstrap loader for Linux, GRUB, allows selection of kernel from multiple disks, versions, kernel options



BIOS vs UEFI

- UEFI (Unified Extensible Firmware Interface)
  - Appeared in 2007
  - Supports all the functionalities of the BIOS
  - Can be saved in any support (disk, flash memory, …)
  - Faster than BIOS
  - Allows for bigger disks
  - Better graphic interface
  - Allows Secure Boot

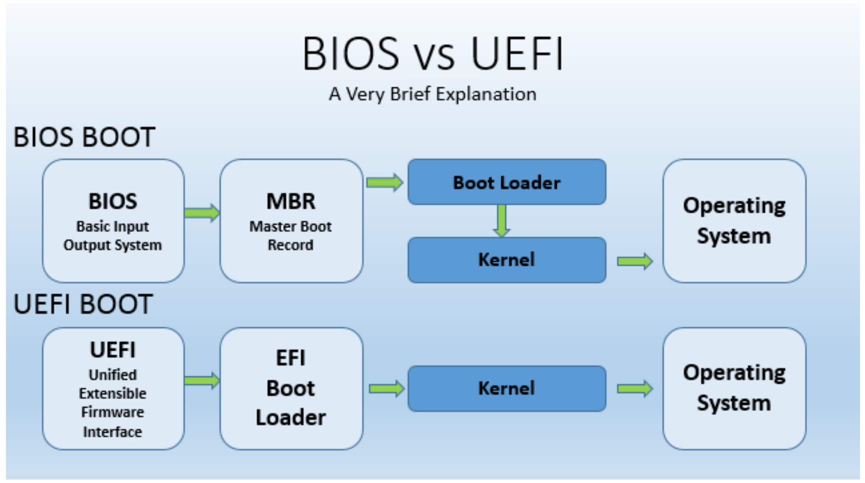


Image from:
https://passmoz.com/

OS: Initialization

- OS performs its boot “routines”:
  - hardware - Initialization of registers of the controllers , the system interrupts, the interrupt vector ;
  - software - Initialization of data structures that represent the various system resources and algorithms that support the management of these resources .
- Creation of the first cases :
  - The system auxiliary processes
    - login , file system , network ...
  - the interpreter commands
    - command line (sh , bash, csh, command.com) or windows
- "Wait for Interrupt "

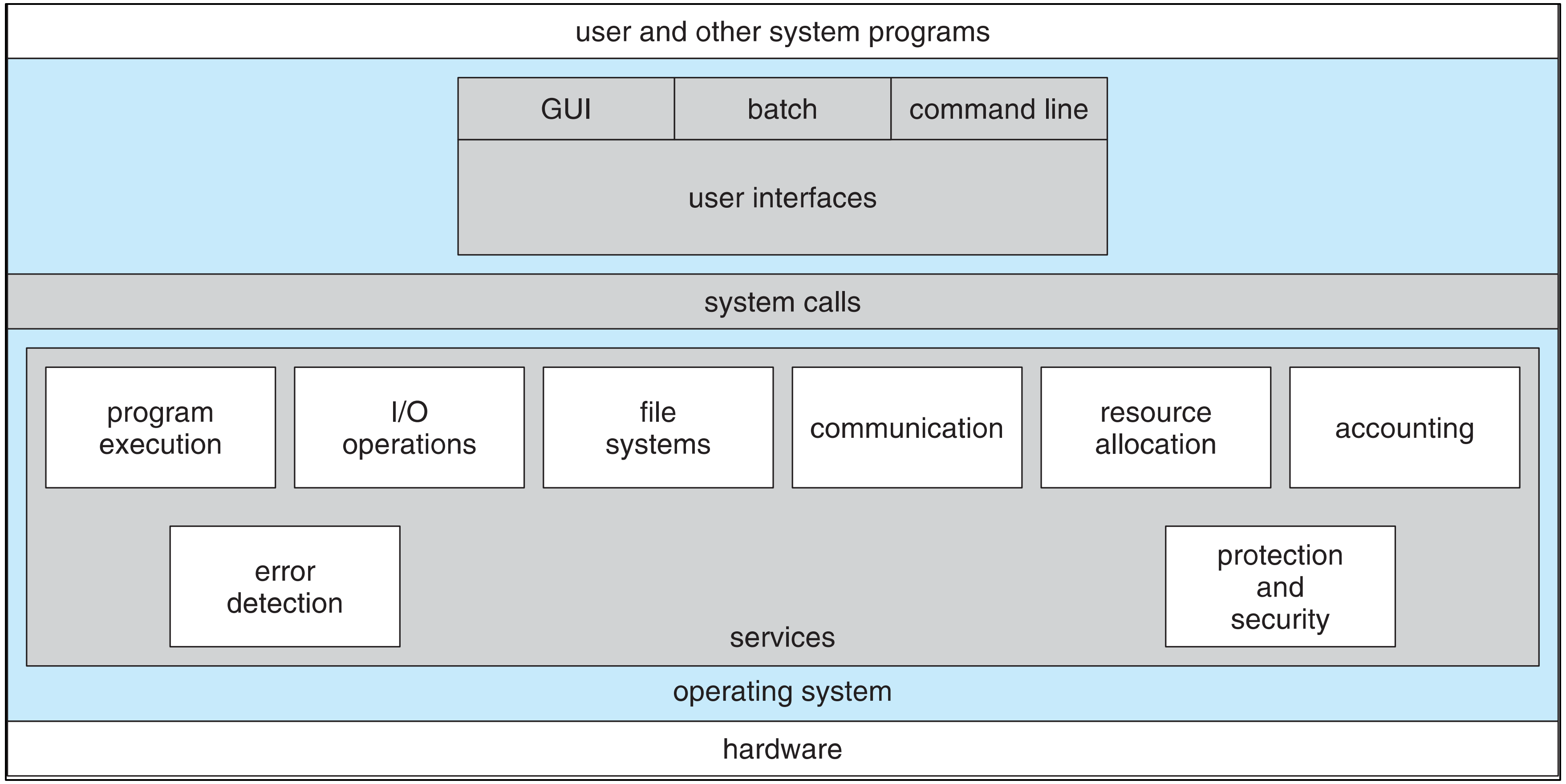
Wait for interrupt

- After booting, the OS is “quiet" waiting for work and ... there is a hardware interrupt
  - OS performs a task that handles the interruption .
    - Ex : CPU clock interrupts , OS updates the time
- The OS is " quiet" and ... a program requests a service - through a software interrupt or trap:
  - OS triggers a task that runs the service (this interruption)
    - Ex : Print a string on the screen (in MS -DOS)

```
mov dx , <address of string>
mov ah, 09H
int 21H
```

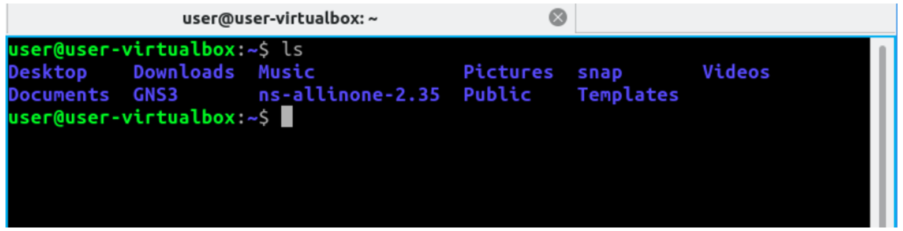
Operating System Provided Services

- The specific services provided to depend on the specific OS…



User and OS interface

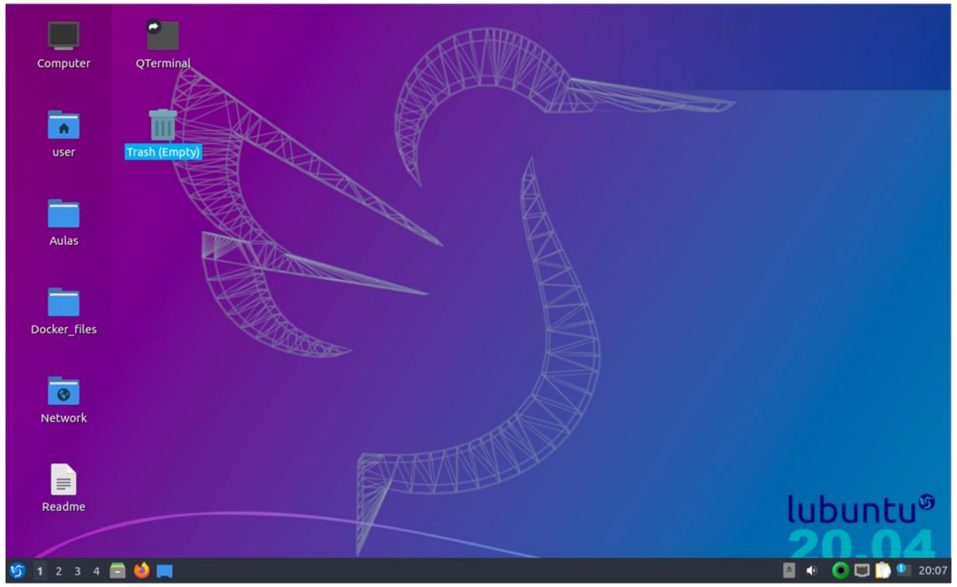
- Command interpreters
  - Shells



```
user@user-virtualbox:~$ ls
Desktop    Downloads  Music             Pictures  snap       Videos
Documents  GNS3       ns-allinone-2.35  Public    Templates
user@user-virtualbox:~$
```

Qterminal, Lubuntu 20.04

- Graphical User Interface (GUI)

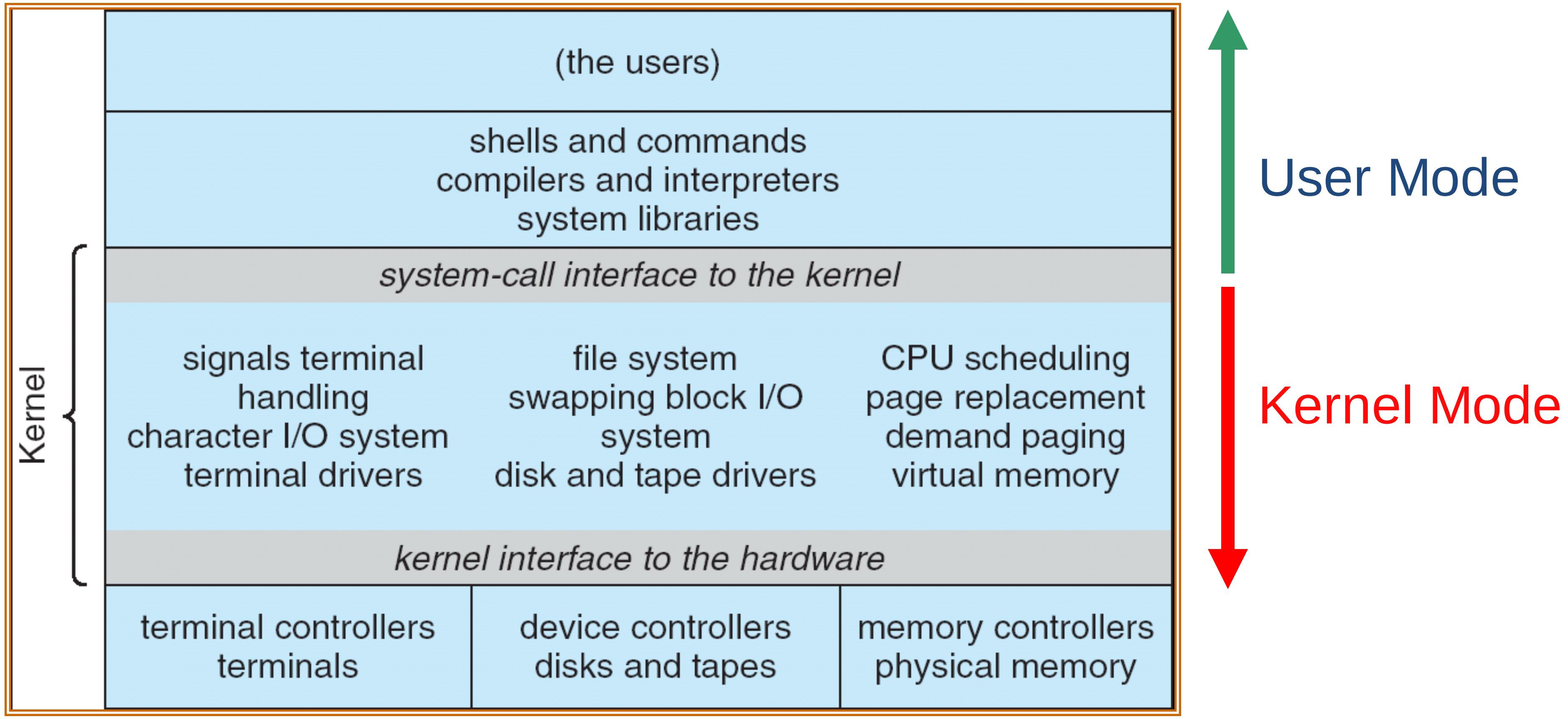


LUbuntu 20.04

Protection provided by the OS

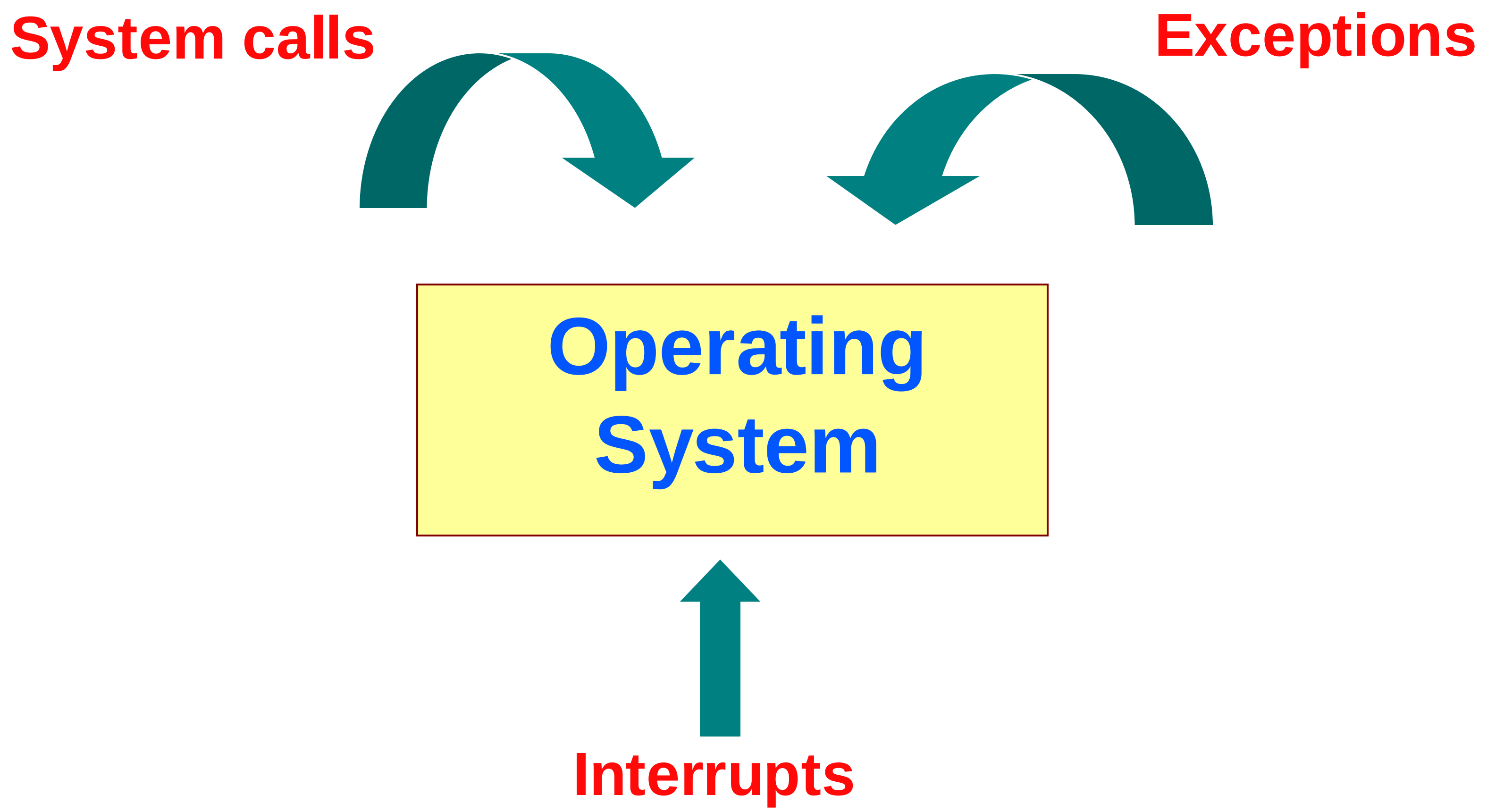
- Dual Mode of Operation
  - "User Mode" and "Kernel Mode"
  - In "User Mode" only common instructions can be performed (e.g. arithmetic and logic). In "Kernel Mode" anything can be done.
  - A mode bit contains the current mode of operation
  - RTI: Return from interruption
  - Kernel mode assumed on traps and on interrupts
- I/O Protection
  - All I/O Instructions are privileged.
    For doing I/O a trap into the operating system must be generated.
- Memory Protection
  - Each program can only access its memory
- CPU Protection
  - The CPU must remain in control of the Operating System, even if the applications does not want to let go.

Dual Mode of Operation (at least) - to assure security



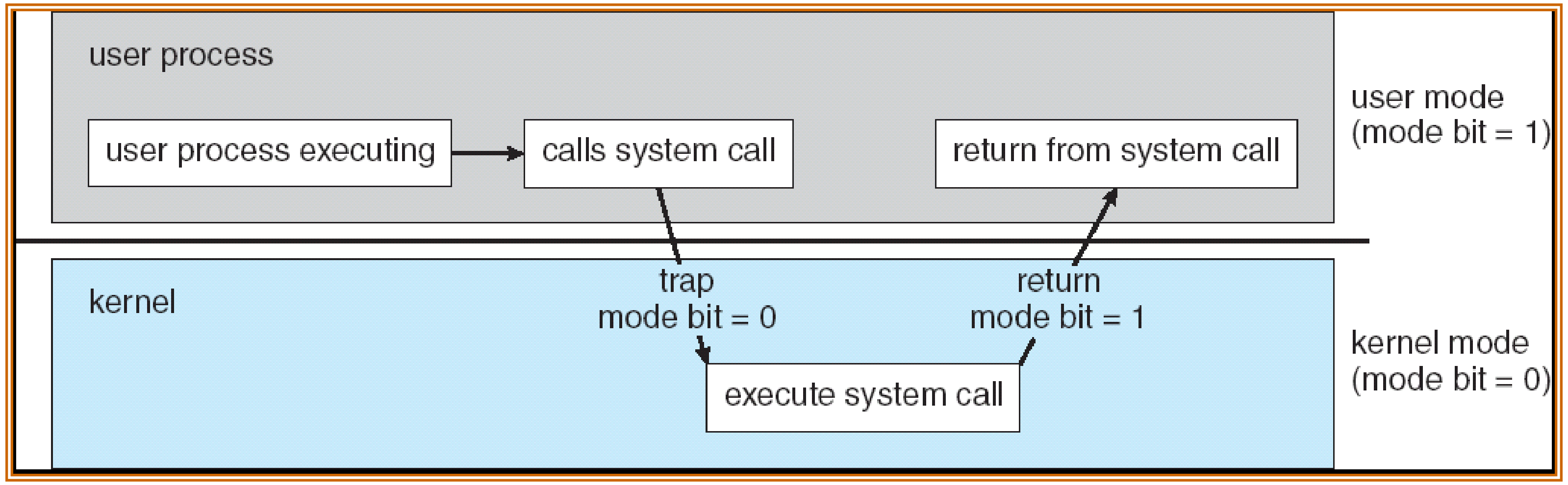
UNIX Operating System Structure

System Calls, Exceptions and Interrupts

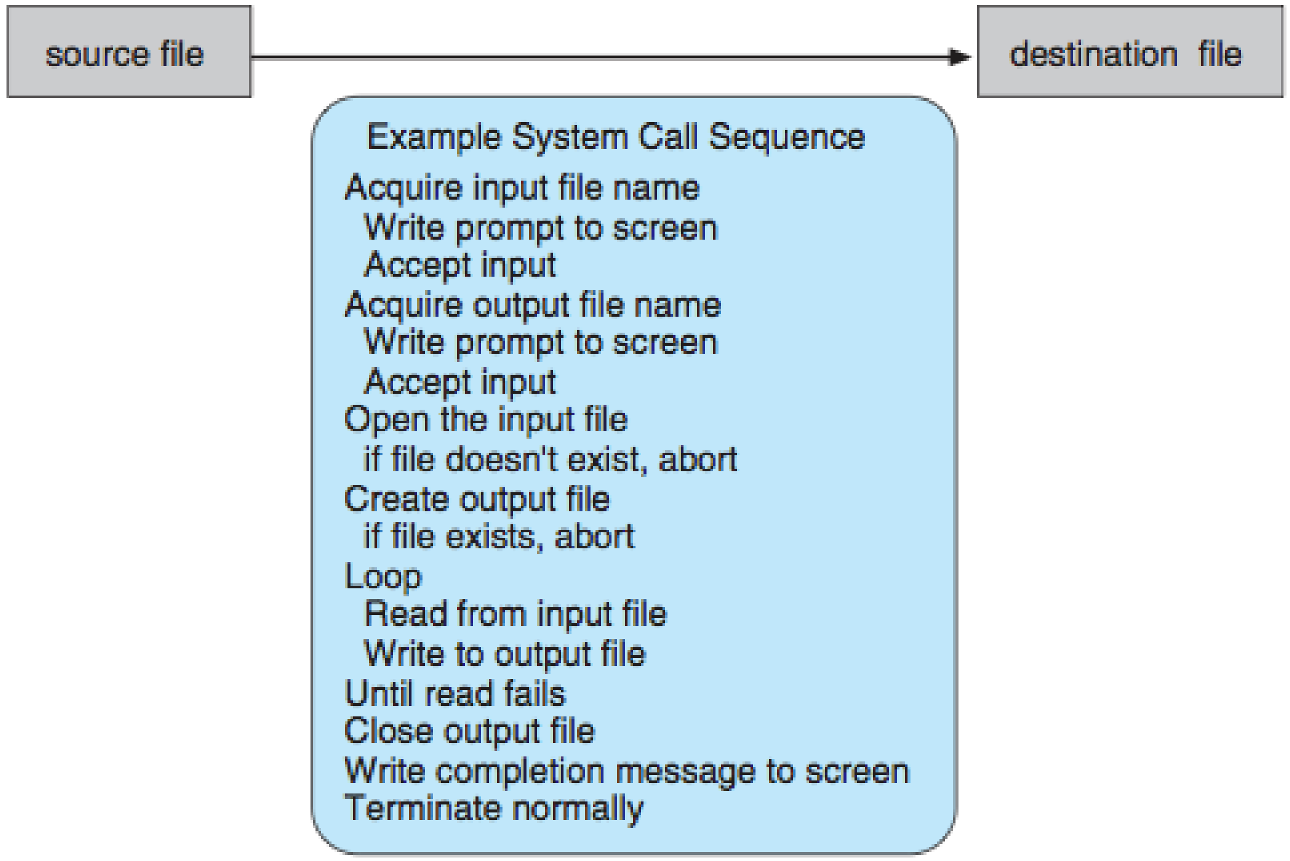


System Calls

Dual Mode of Operation



System Calls
A file copy operation

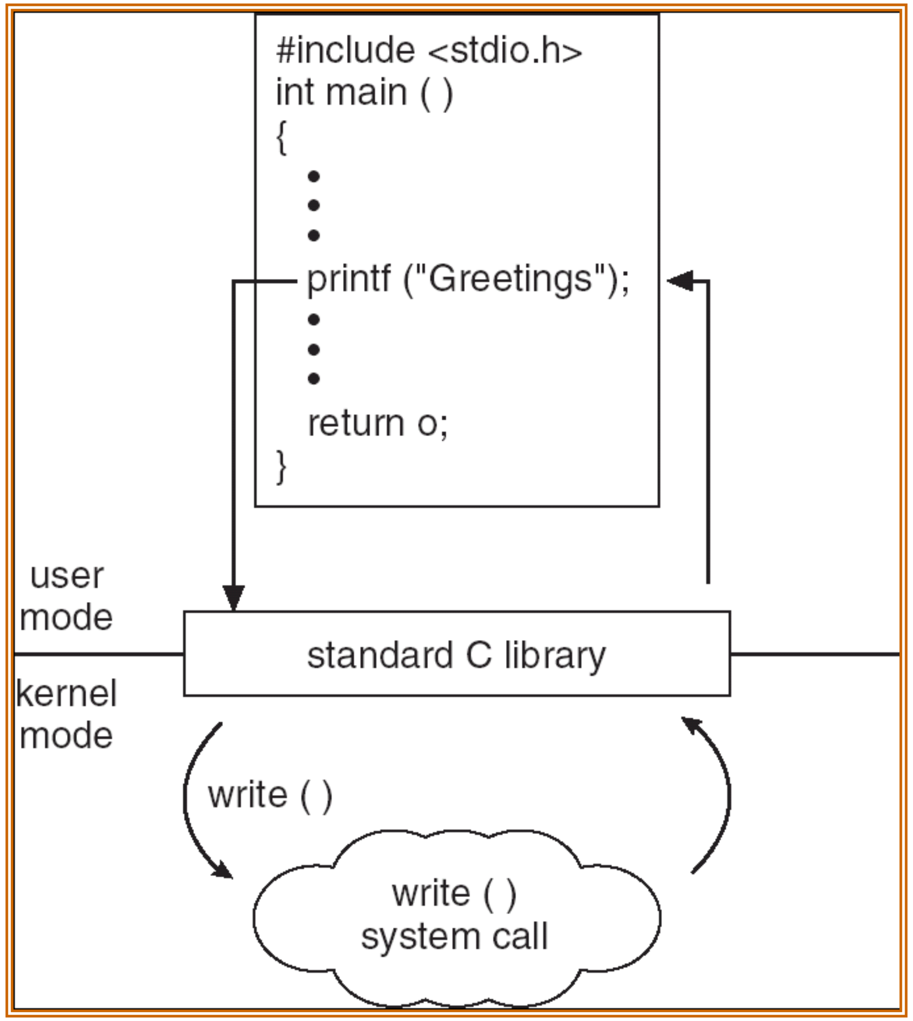


```
Example System Call Sequence
Acquire input file name
  Write prompt to screen
  Accept input
Acquire output file name
  Write prompt to screen
  Accept input
Open the input file
  if file doesn't exist, abort
Create output file
  if file exists, abort
Loop
  Read from input file
  Write to output file
Until read fails
Close output file
Write completion message to screen
Terminate normally
```

Even a simple program creates multiple system calls!

System Calls

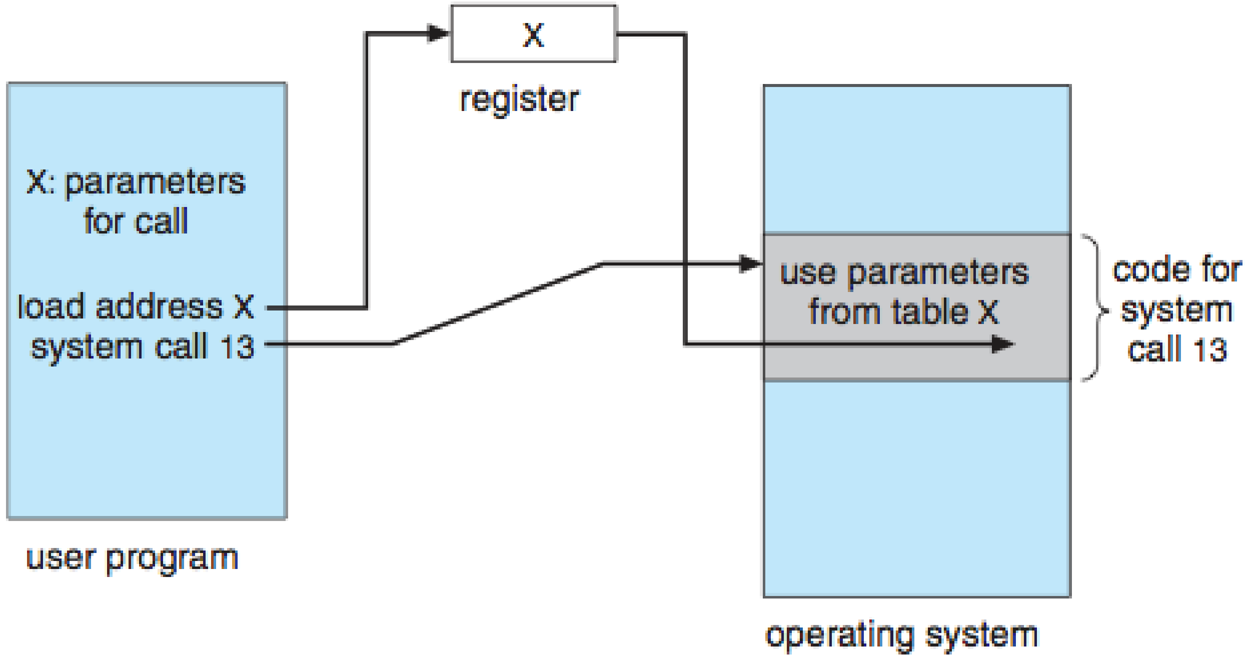
Structure



```
#include <stdio.h>
int main ( )
{
   •
   •
   •
   printf ("Greetings");
   •
   •
   •
   return 0;
}
```

System Calls

Passing of parameters as a table



System Calls
Example with a "Hello World" Application!

```
#include <stdio.h>

int main() {
  printf("Hello World\n");
  return 0;
}
```

- The "strace" program is very useful:
  - It allows you to see what system calls are made and with what parameters!

System Calls
“strace” command

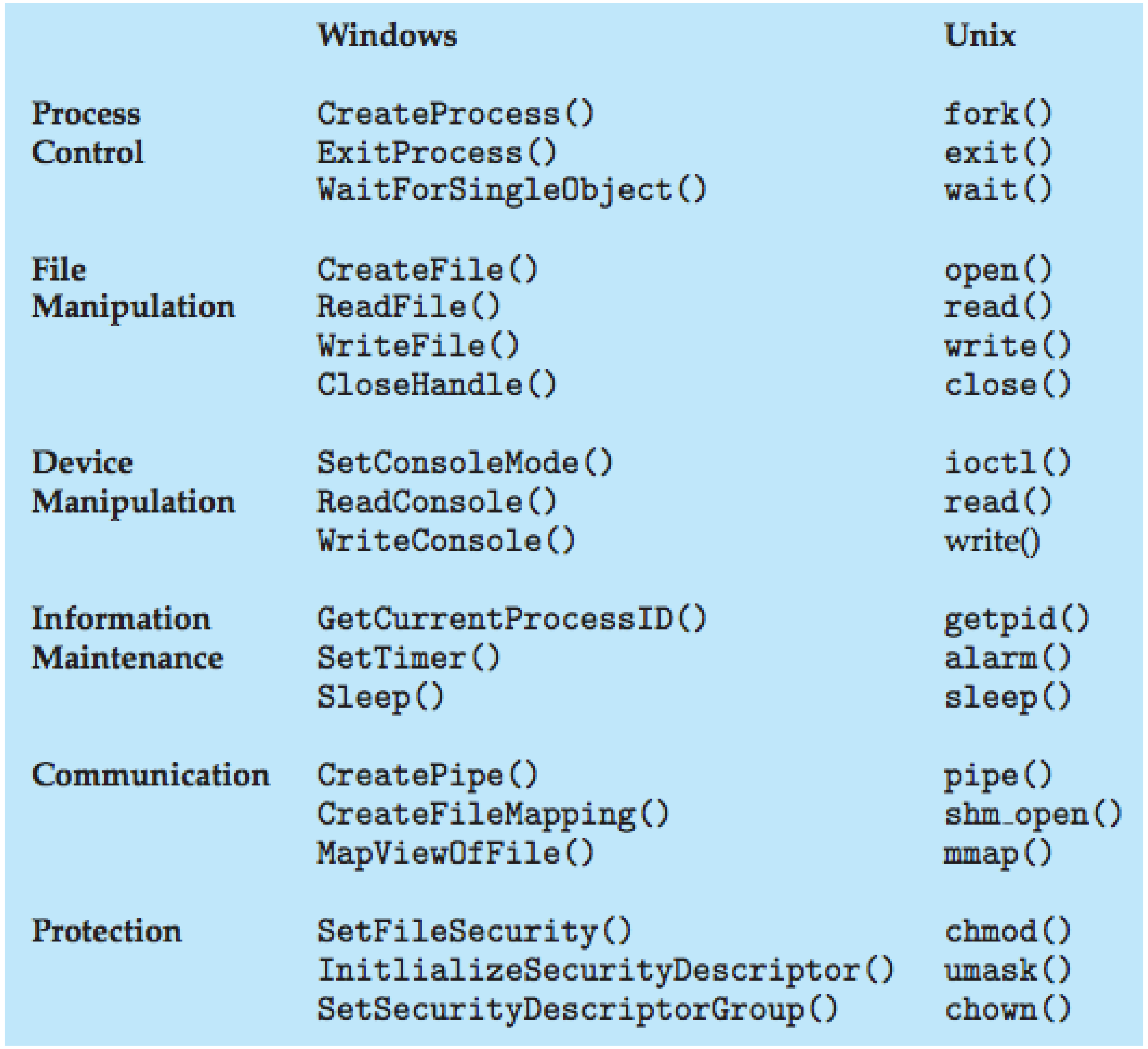
$strace ./hello

```
[vasco@student2]$ ./hello
Hello world
[vasco@student2]$ strace ./hello
execve("./hello", ["./hello"], [/* 28 vars */]) = 0
brk(0)                                  = 0x1bf9000
mmap(NULL, 4096, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) = 0x7fe594bb5000
access("/etc/ld.so.preload", R_OK)      = -1 ENOENT (No such file or directory)
open("/etc/ld.so.cache", O_RDONLY)      = 3
fstat(3, {st_mode=S_IFREG|0644, st_size=53205, ...}) = 0
mmap(NULL, 53205, PROT_READ, MAP_PRIVATE, 3, 0) = 0x7fe594ba8000
close(3)                                = 0
open("/lib64/libc.so.6", O_RDONLY)      = 3
read(3, "\177ELF\2\1\1\3\0\0\0\0\0\0\0\0\3\0>\0\1\0\0\0000\356\201f=\0\0\0"..., 832) = 832
fstat(3, {st_mode=S_IFREG|0755, st_size=1928936, ...}) = 0
mmap(0x3d66800000, 3750184, PROT_READ|PROT_EXEC, MAP_PRIVATE|MAP_DENYWRITE, 3, 0) = 0x3d66800000
mprotect(0x3d6698a000, 2097152, PROT_NONE) = 0

(…)

write(1, "Hello world\n", 12Hello world
)           = 12
exit_group(0)                           = ?
+++ exited with 0 +++
[vasco@student2]$
```

System Calls

Examples of WINDOWS and UNIX

6 categories of system calls

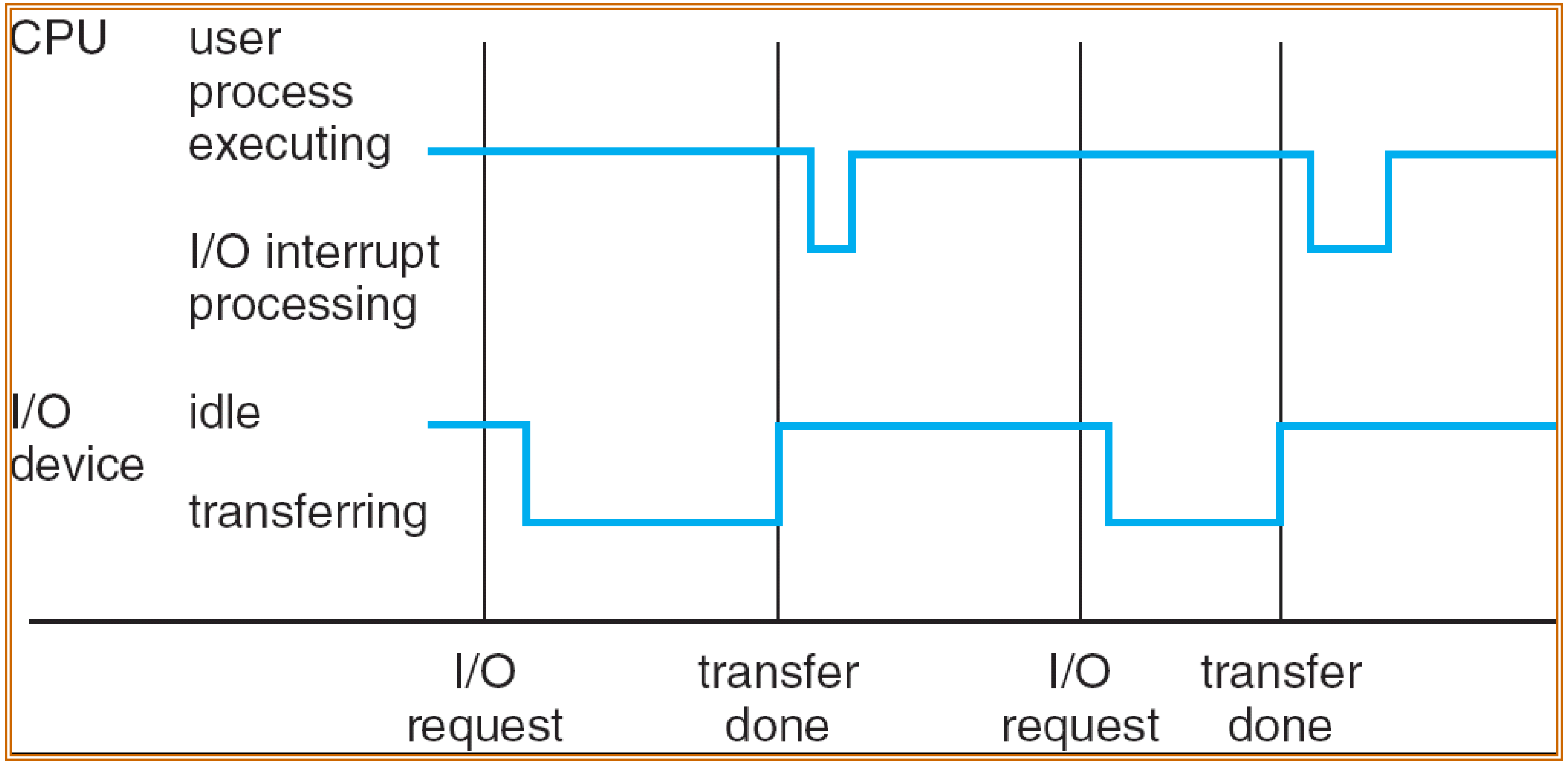


	Windows	Unix
Process Control	CreateProcess()	fork()
	ExitProcess()	exit()
	WaitForSingleObject()	wait()
File Manipulation	CreateFile()	open()
	ReadFile()	read()
	WriteFile()	write()
	CloseHandle()	close()
Device Manipulation	SetConsoleMode()	ioctl()
	ReadConsole()	read()
	WriteConsole()	write()
Information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

Input/Output versus CPU Usage

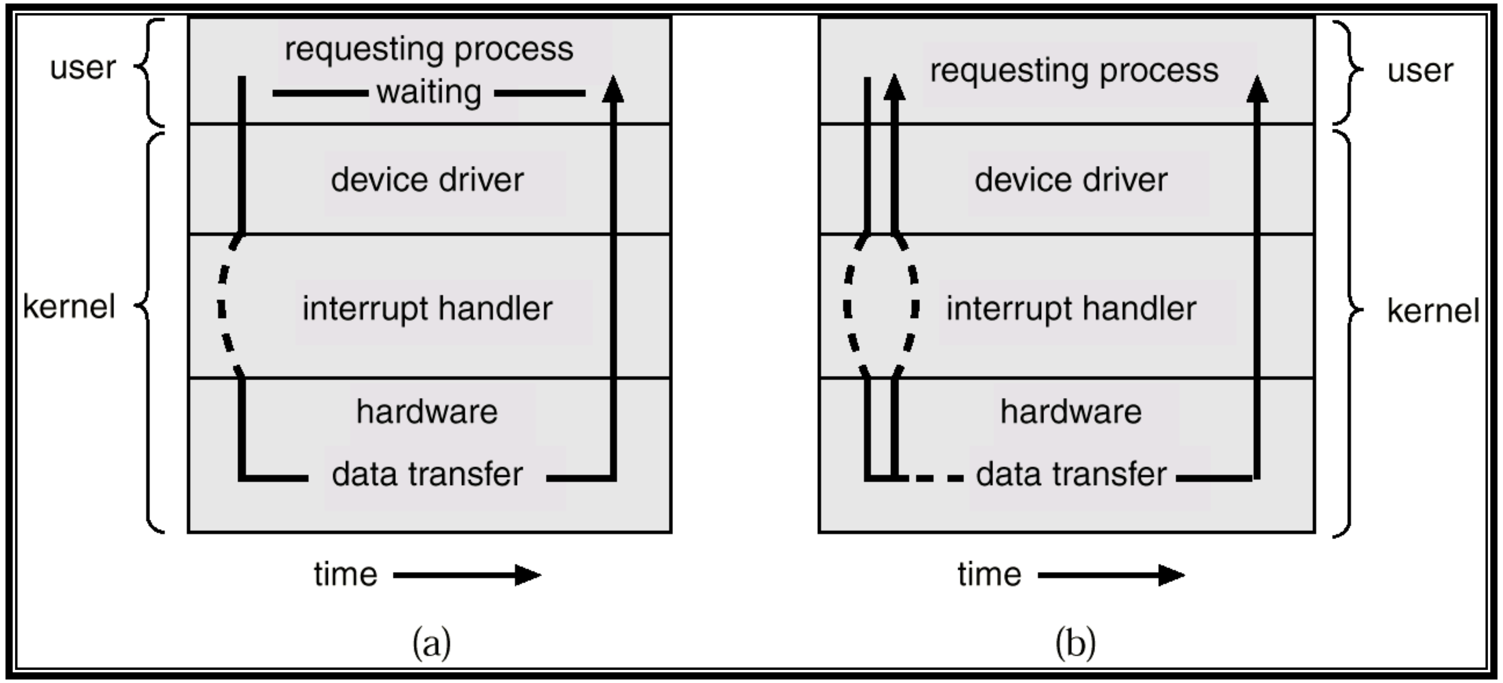
- I/O devices and the CPU can execute concurrently
- Each device controller oversees a particular device
- Each device controller has a local buffer
- CPU moves data from/to main memory to/from local buffers
- I/O is from the device to local buffer of controller
- Device controller informs CPU that it has finished its operation by causing an interrupt

Interrupt Timeline



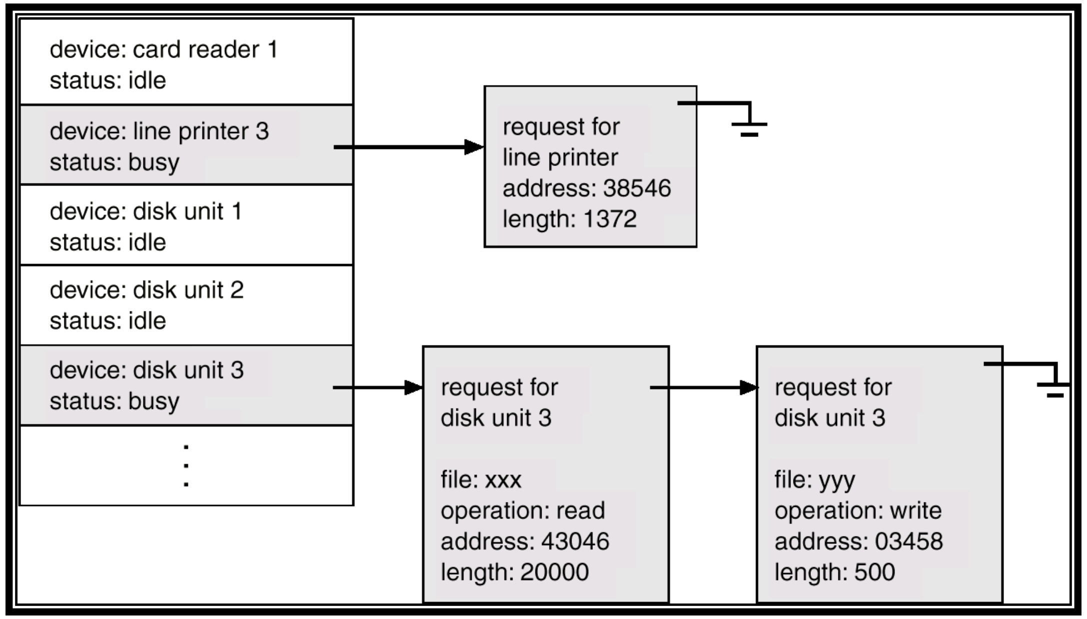
Different Types of I/O

- Synchronous (a) versus asynchronous I/O (b)
  - In (a) the application blocks on the system call
  - In (b) the application resumes execution after issuing the system call and later receives the answer



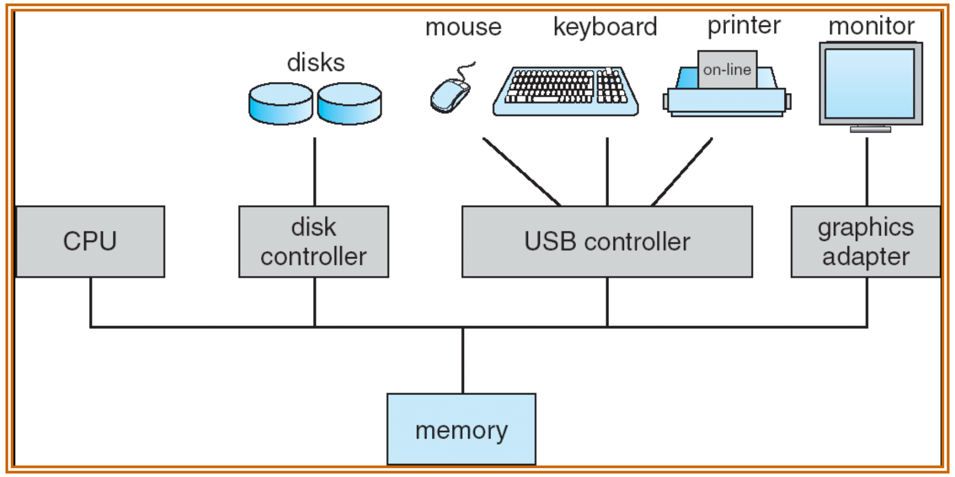
Device Status Table

- Each table entry indicates the device's type, address, and state (not functioning, idle, or busy). If the device is busy with a request, the type of request and other parameters will be stored in the table entry for that device.
- If the kernel supports asynchronous I/O, it also keeps track of many I/O requests at the same time.

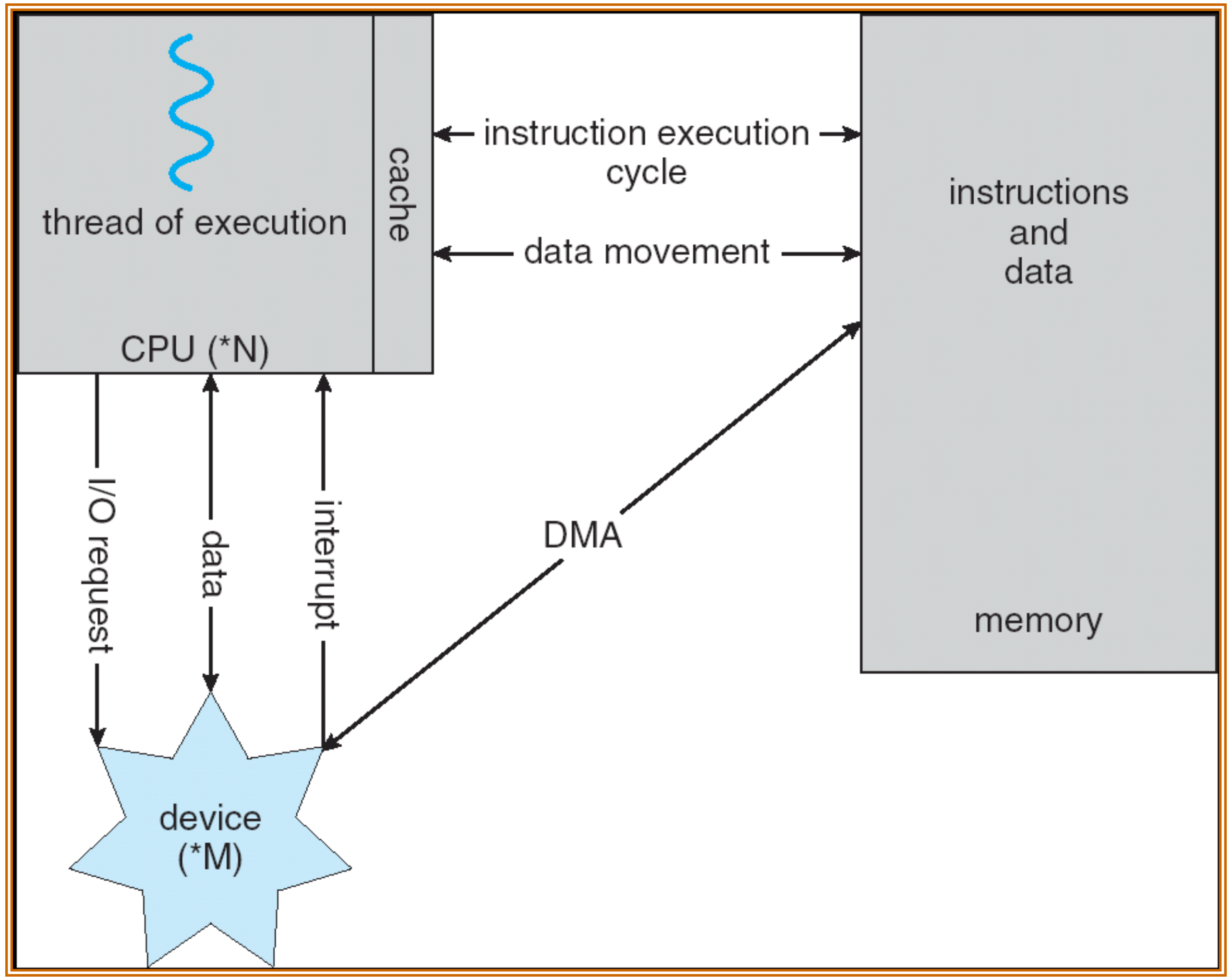


Direct Memory Access (DMA)

- 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
- Implies the capability of arbitrating the system bus
- Only one interrupt is generated per block, rather than one interrupt per byte

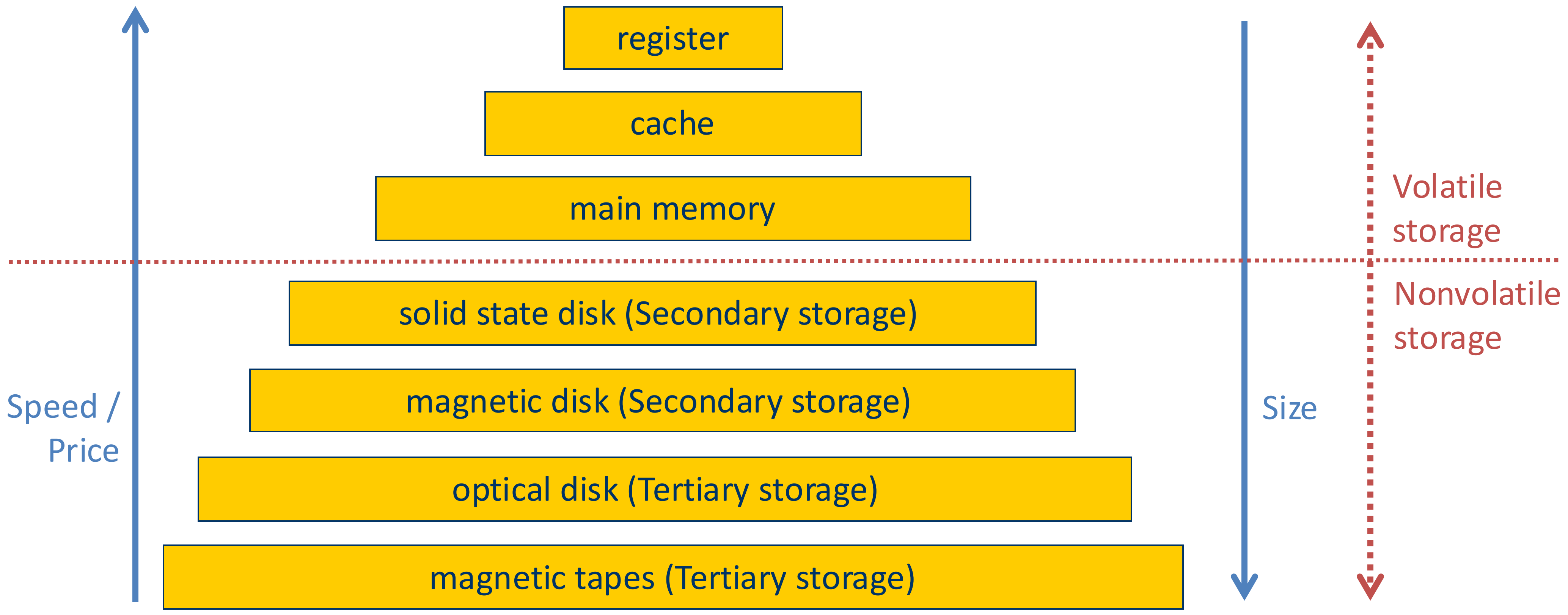


DMA (2)


thread of execution
CPU (*N)
cache
instruction execution cycle
data movement
instructions and data
memory
I/O request
data
interrupt
DMA
device (*M)

Storage Hierarchy

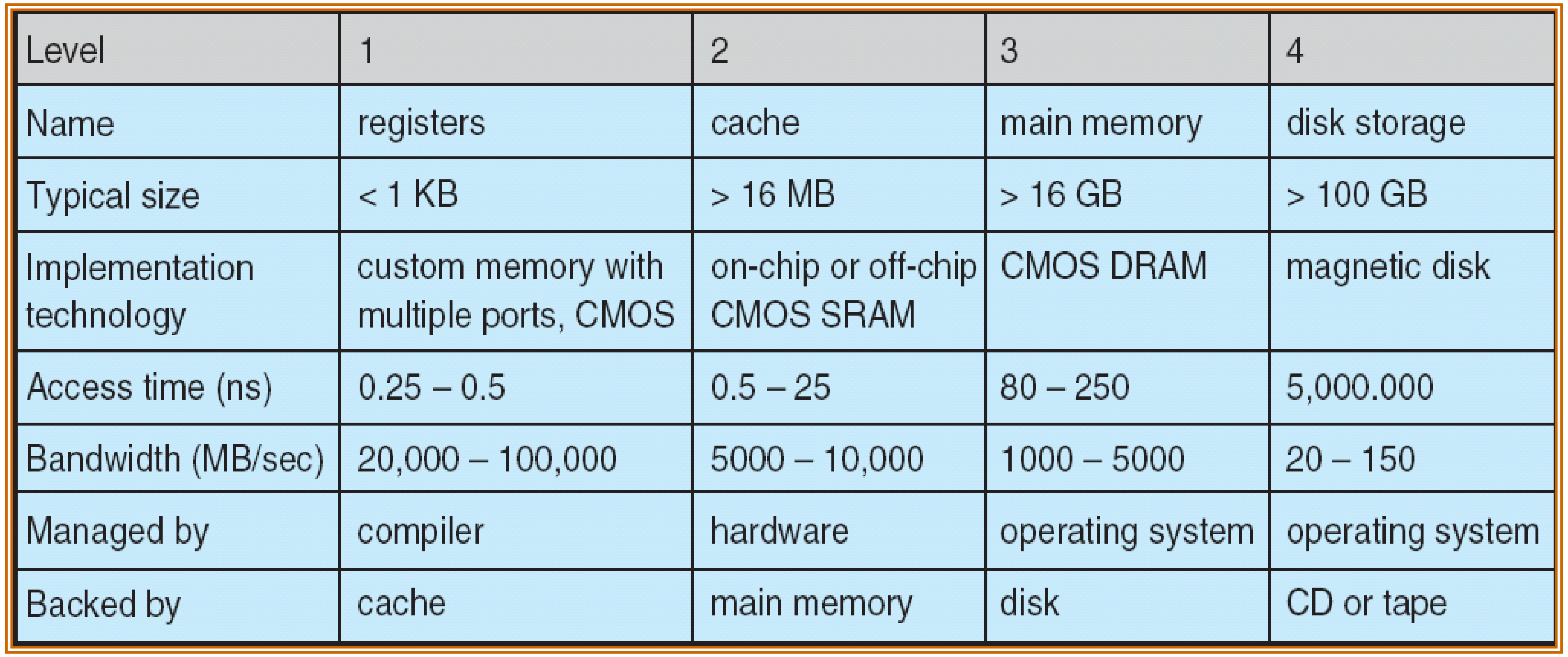
- Caches, Buffers and a Storage Hierarchy are Essential


register
cache
main memory
solid state disk (Secondary storage)
magnetic disk (Secondary storage)
optical disk (Tertiary storage)
magnetic tapes (Tertiary storage)
Speed / Price
Size
Volatile storage
Nonvolatile storage

Storage Hierarchy

- A great part of this hierarchy is controlled by the Operating System
  - Cache Management Policies
  - Buffer Management Policies
  - Memory Management Policies

Principles of Locality:
- Temporal Locality
- Spatial Locality



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

Storage Hierarchy – a human perspective

	Computer time	Human time
Processor cycle time (for a 2GHz frequency)	0.5 ns	1 s
L2 cache access*	10 ns	20 s
Local DRAM*	80 ns	160 s (2.7min)
SSD access time*	100 µs	200,000 s (≈2.3 days)
Magnetic disk access time*	8,000,000 ns (8 ms)	16,000,000 s (≈185 days)
CD-ROM access time*	200 ms	400,000,000 s (≈12.7 years)

*Values change depending on the specific HW used

Memory Protection

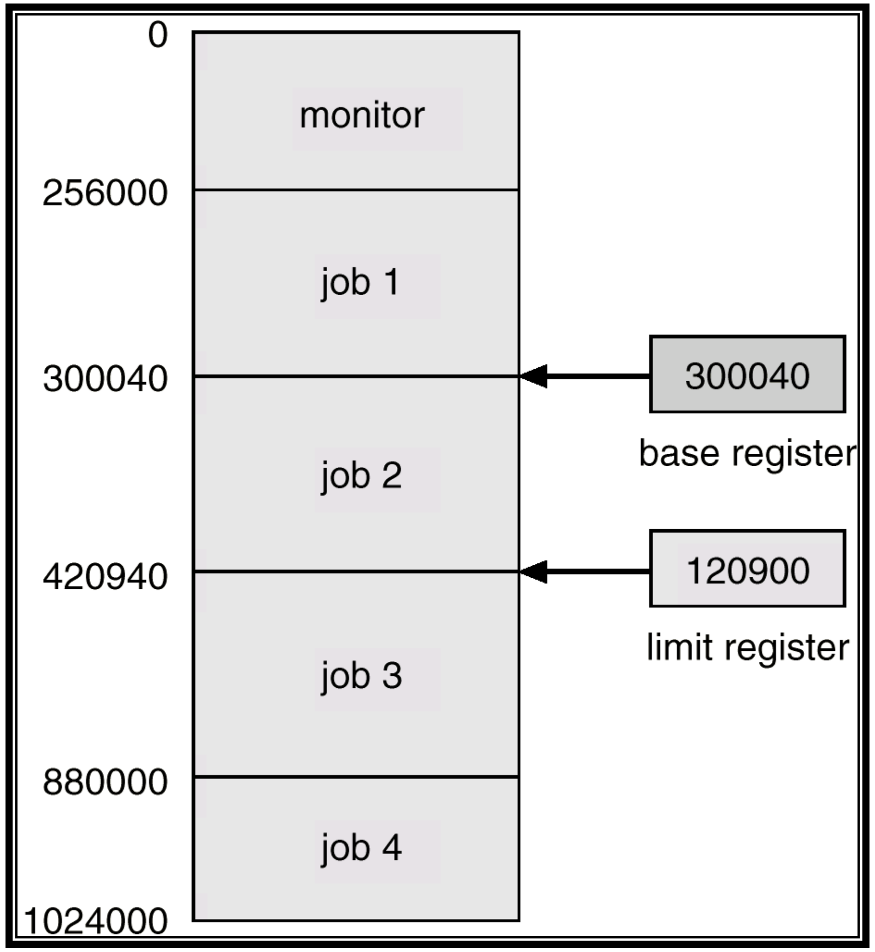
- Each program must only be able to access its own memory
  - Segmentation
  - Virtual Memory

- Segmentation is based on having a base pointer and a limit for each process running
  - Any access made outside of its "space" generates a trap and normally leads to the process being killed

- Virtual Memory is based on having a table that translates "virtual addresses" into real ones
  - Normally, there is such a table for each process
  - Each process sees all the address space
  - An access made to a non existing page generates a trap and normally leads to the process being killed

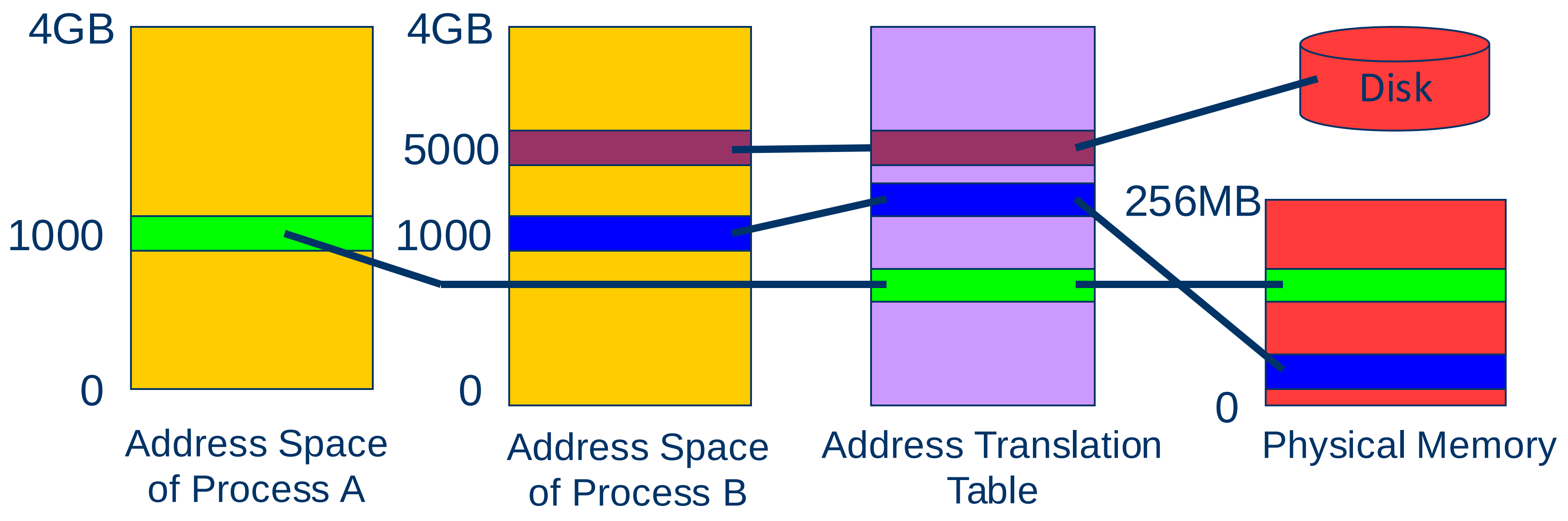
Memory Protection

Segmentation



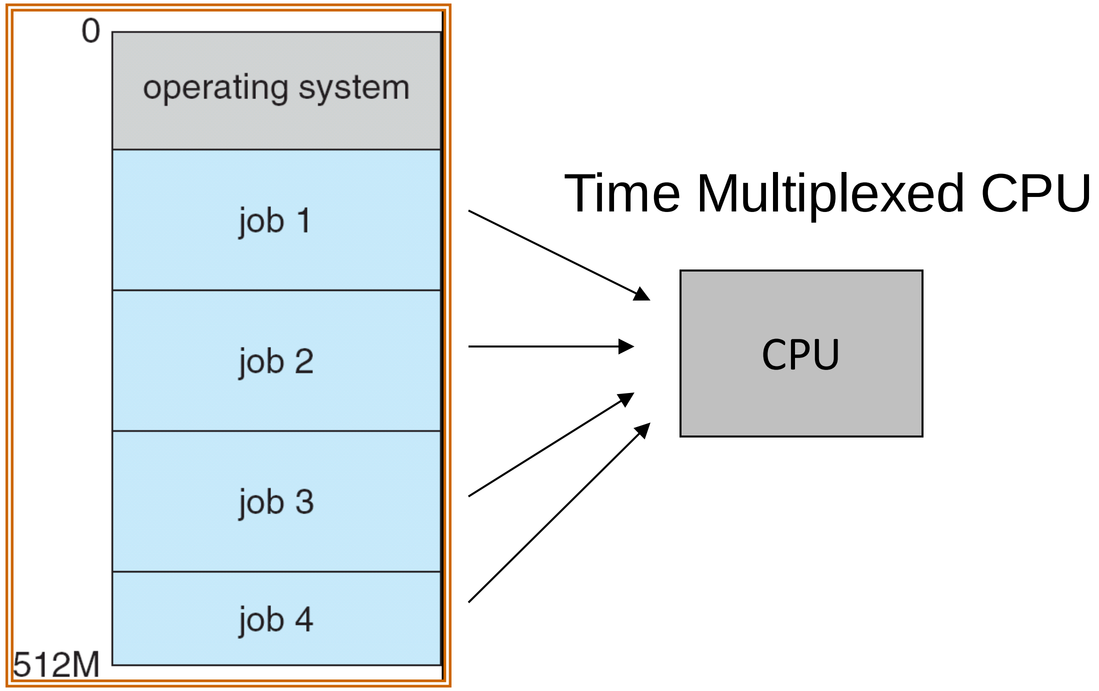
Memory Protection

Virtual Memory


4GB
1000
0
4GB
5000
1000
0
Disk
256MB
0
Address Space of Process A
Address Space of Process B
Address Translation Table
Physical Memory

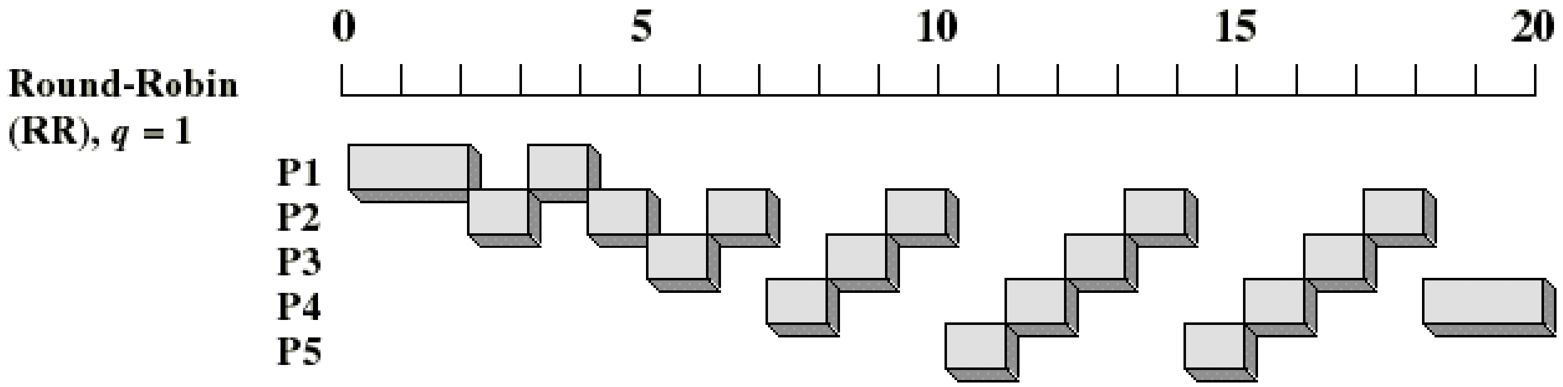
Multiprogramming

- Multiprogramming: maintaining several jobs in memory, active at the same time, so that when the current program can no longer run (for example because it’s blocked waiting for I/O), another is available to run
- Optimizes resource utilization: CPU and I/O



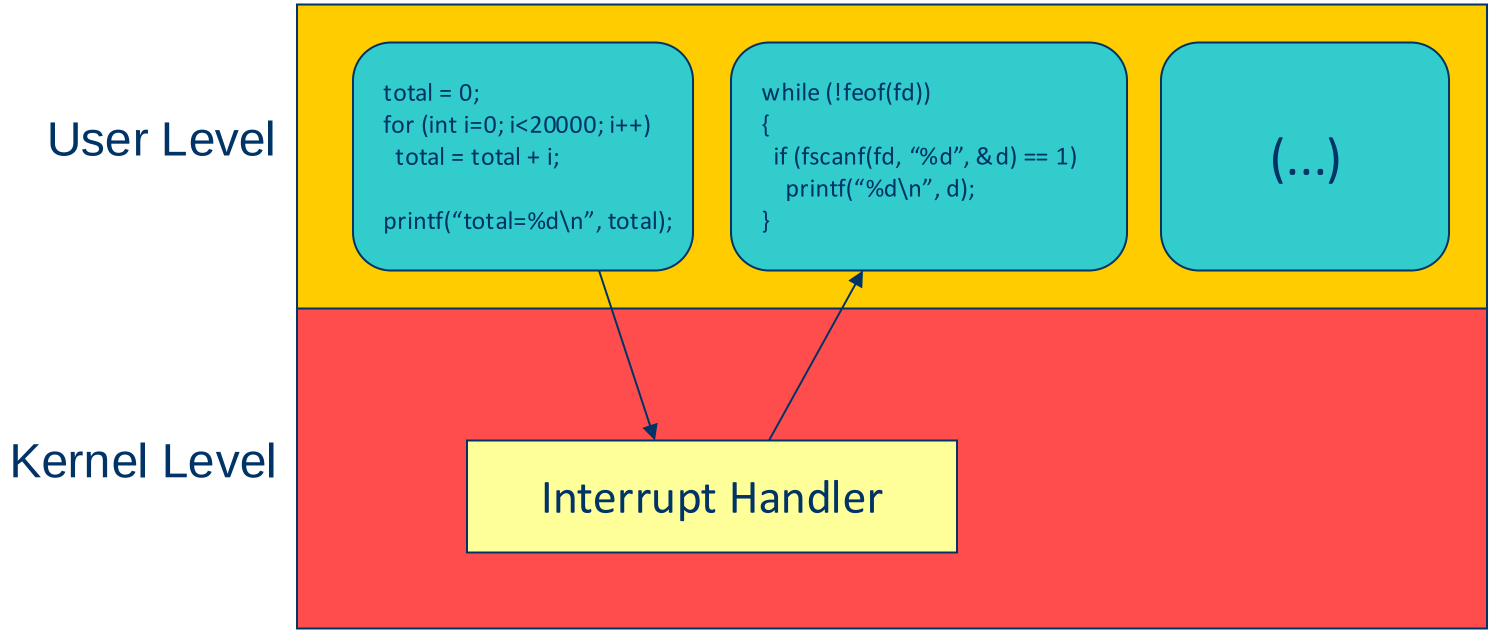
Multitasking

- Multitasking is an extension of multiprogramming in which the execution of the active programs is time-sliced:
  - Each program runs for a short period of time, then another is run.
  - When a program is running and is forcibly replaced by another, typically with a higher priority, it is said to have been preempted, thus the term Preemptive Operating Systems



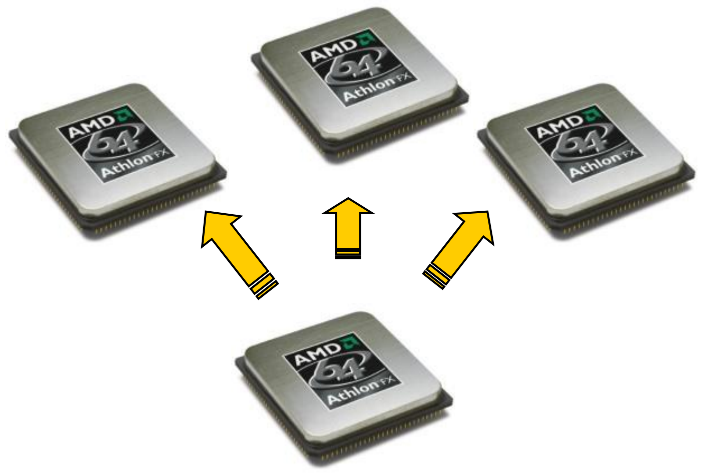
Multitasking

- For implementing multitasking, a non-maskable interrupt must connected to a hardware timer
  - Every few milliseconds (e.g., 100ms), the interrupt causes a task (or process) switch

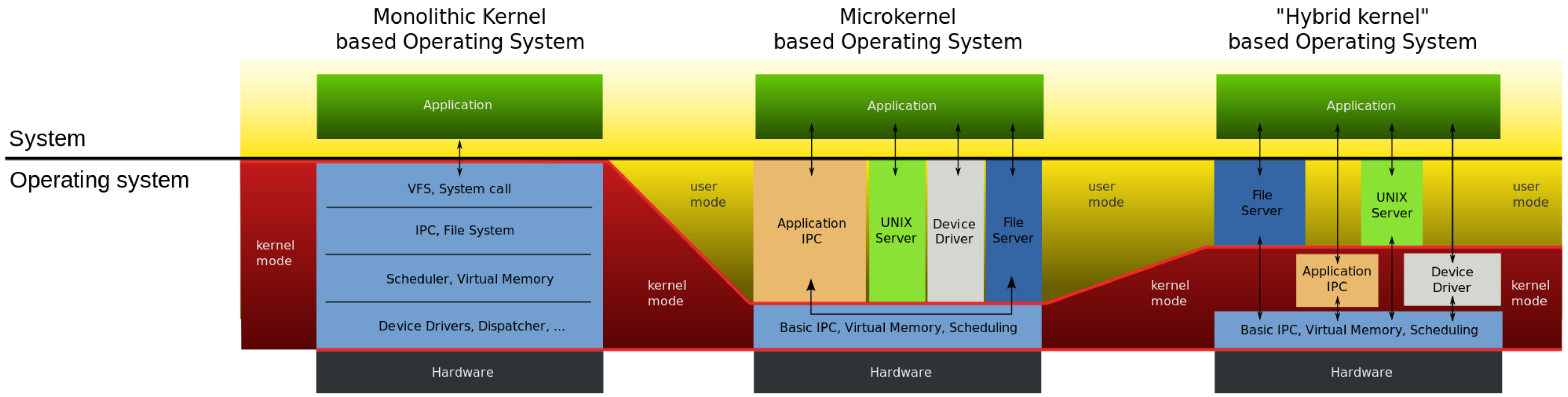


Nowadays…

- Since Windows XP, Linux 2.6.xx:
  - Preemptive multitasking operating systems using virtual memory
  - Each process thinks it has the whole computer for itself
    → "Virtual Machine"



Operating System Architecture and Structures


Monolithic Kernel based Operating System
Microkernel based Operating System
"Hybrid kernel" based Operating System
System
Operating system
Application
VFS, System call
IPC, File System
Scheduler, Virtual Memory
Device Drivers, Dispatcher, ...
kernel mode
user mode
Application IPC
UNIX Server
Device Driver
File Server
Basic IPC, Virtual Memory, Scheduling
Hardware

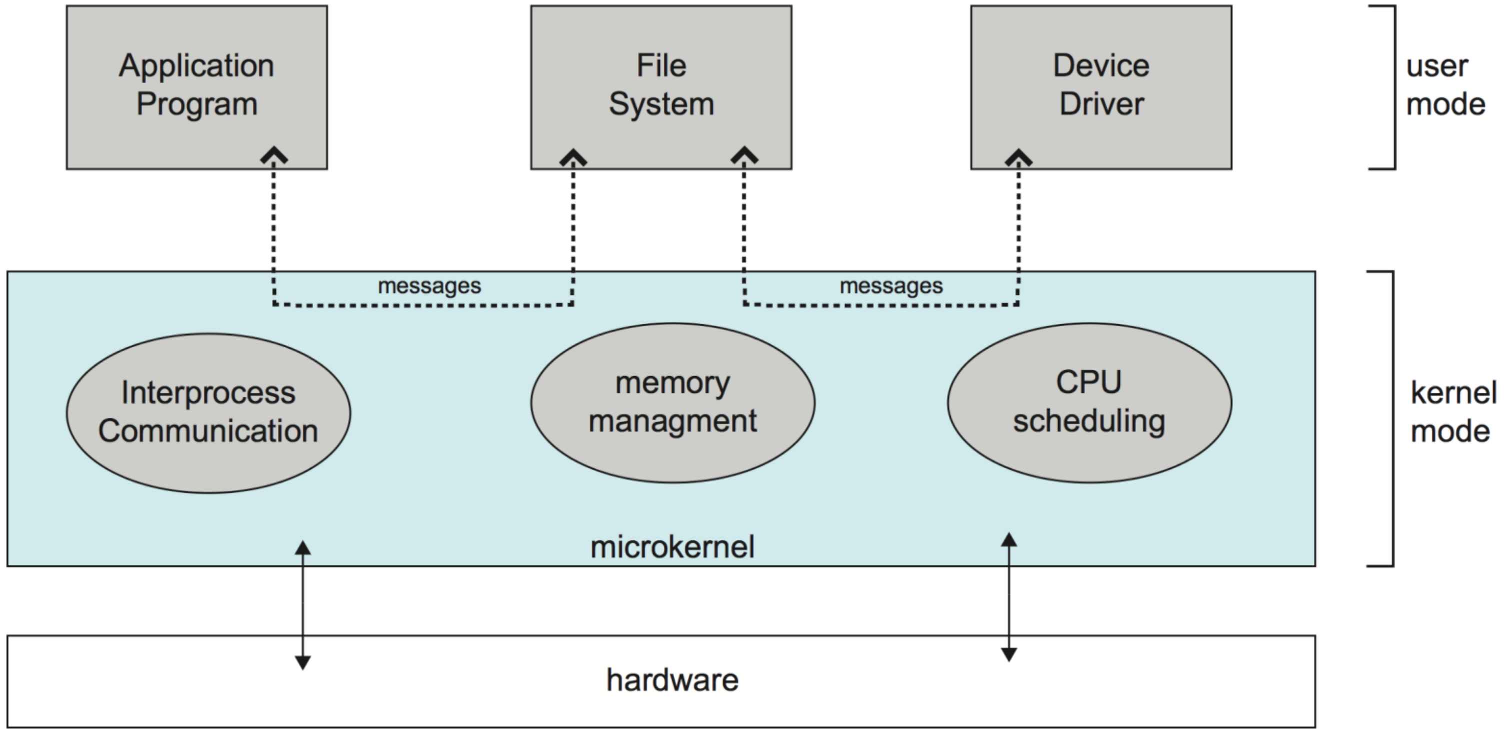
Source: http://en.wikipedia.org/wiki/File:OS-structure2.svg

Operating System Architecture and Structures

- **Microkernel**:
  - Moves as much as possible from the kernel into "user" space
  - The main function of the microkernel is to provide communication between the client program and the various services that are also running in user space. Communication is provided through message passing
  - e.g., Mach, XNU (used in OpenStep and Darwin, part of MAC OS X)
  - 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:
    - SLOW: Performance overhead of user space to kernel space communication

Operating System Architecture and Structures

- **Microkernel**:



Operating System Architecture and Structures

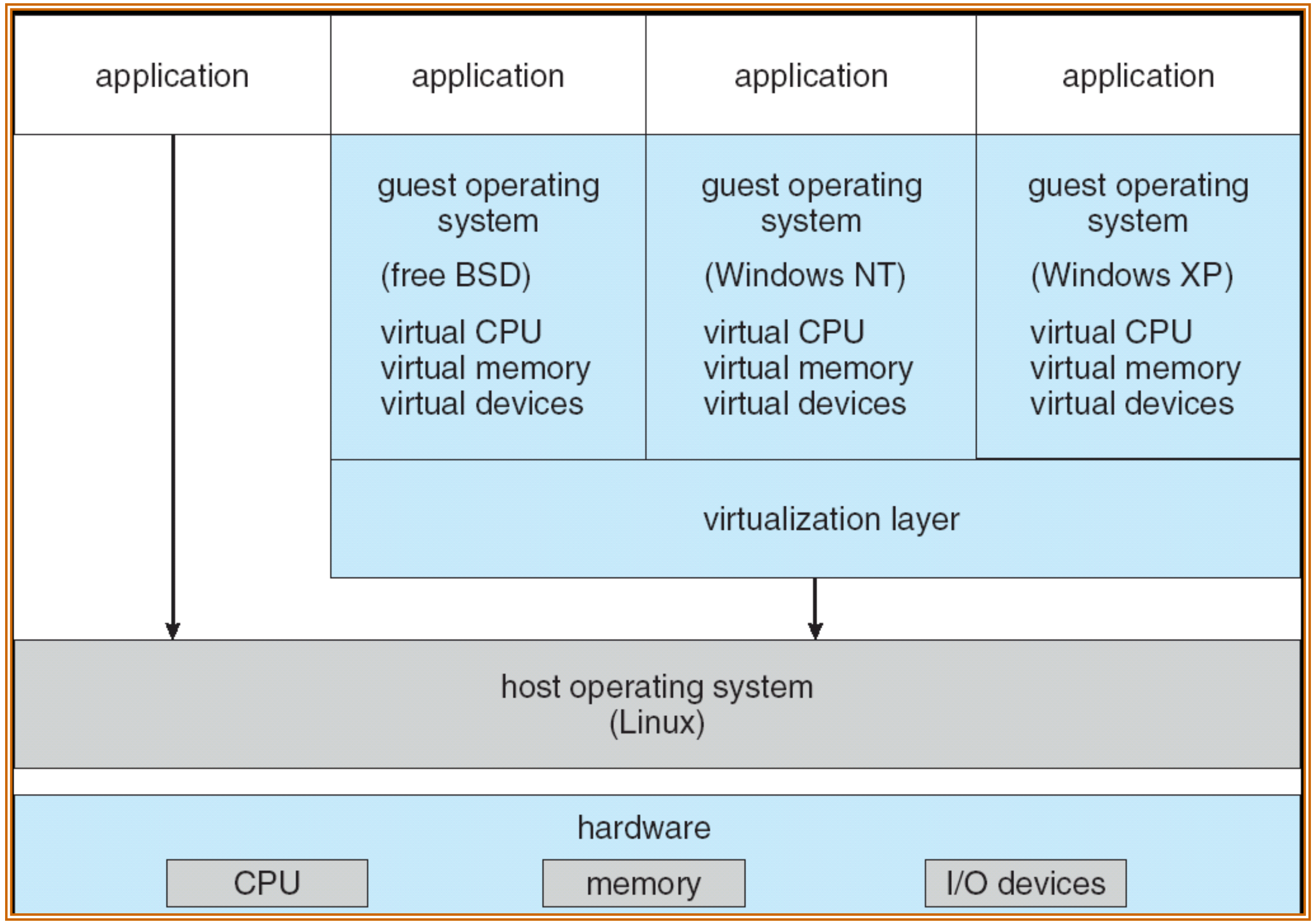
- **Monolithic**:
  - Everything is put below the system-call interface and above the physical hardware
  - Provides the file system, CPU scheduling, memory management, and other operating-system functions; a large number of functions for one level
  - e.g., Unix, Linux, MS-DOS, Windows 9* family
  - Benefits:
    - FAST!
  - Detriments:
    - Less reliable (more code is running in kernel mode)
    - Less secure
    - Not so easy to port to new architectures (… it depends)

Virtual Machines

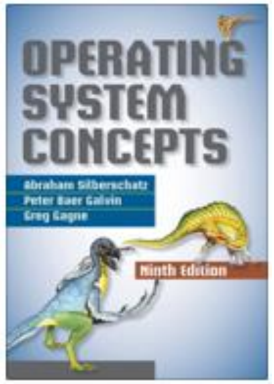
- Process Virtual Machines:
  - "Abstract Machine" that runs bytecode (e.g. Java, MS.NET)
  - Good for portability, allowing running the same unmodified application in different physical architectures
  - Designed to execute computer programs in a platform-independent environment

- System Virtual Machines:
  - Software that allows different operating systems to be run in parallel giving them the illusion of "having a whole machine"
    (e.g., VMWARE, XEN, VirtualBox, …)
  - Some Virtual Machines run on top of the hardware: IBM z/VM
  - Others run on top of the host OS: XEN, VMWARE, VirtualBox, etc
  - Allows for server consolidation, software testing, software fault tolerance, etc.

Virtual Machines
VMWARE Architecture



References



- [Silberschatz13]
  - Chapter 1: Introduction
    - 1.1, 1.2, 1.4, 1.5, 1.6, 1.7, 1.8, 1.9
  - Chapter 2: Operating-System Structures
    - All chapter 2!

Where to learn more?

- Some interesting videos in youtube:
  - Why Applications Are Operating-System Specific
  - How a Single Bit Inside Your Processor Shields Your Operating System's Integrity

Thank you! Questions?

I keep six honest serving men. They taught me all I knew. Their names are What and Why and When and How and Where and Who. —Rudyard Kipling