



# Operating Systems Overview

Operating Systems Design

# Contents

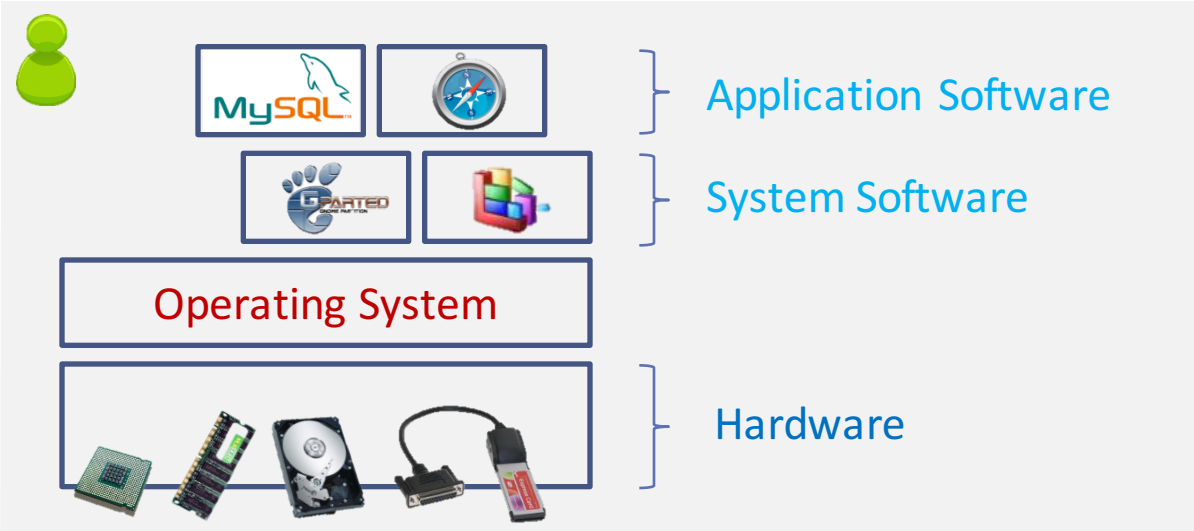
- What Operating Systems Do
- Computer-System Organization
- Computer-System Architecture
- Operating-System Structure
- Operating-System Operations
- Process Management
- Memory Management
- Storage Management
- Protection and Security

# Topics

1. **What an Operating System Is.**
2. What an Operating System does.
3. Operating System Structure.

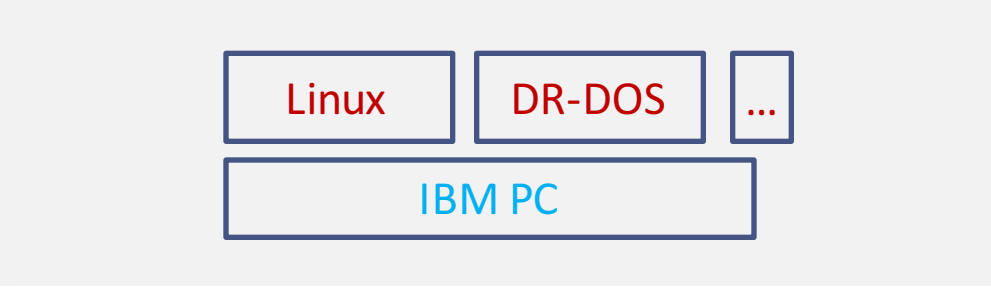
# What is an Operating System?

- A program that acts as an intermediary between a user of a computer and the computer hardware
- 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

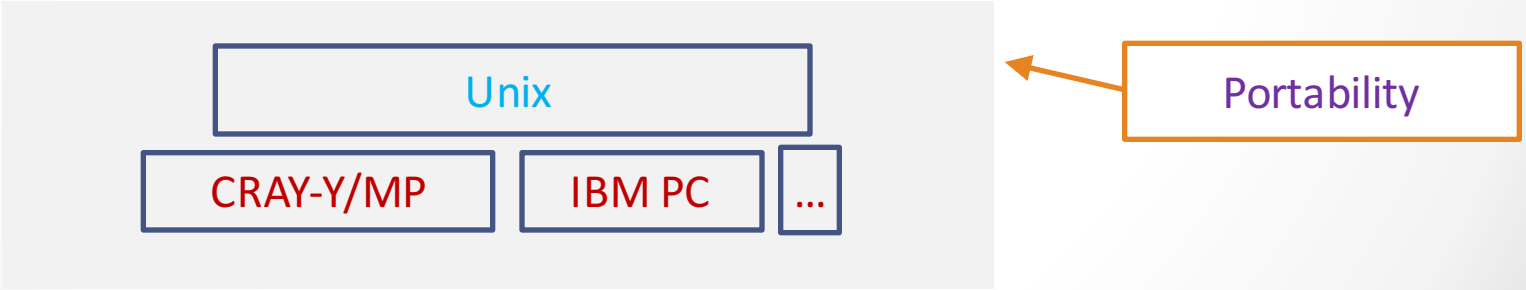


# Versatile

- Same machine, different Operating Systems: IBM PC



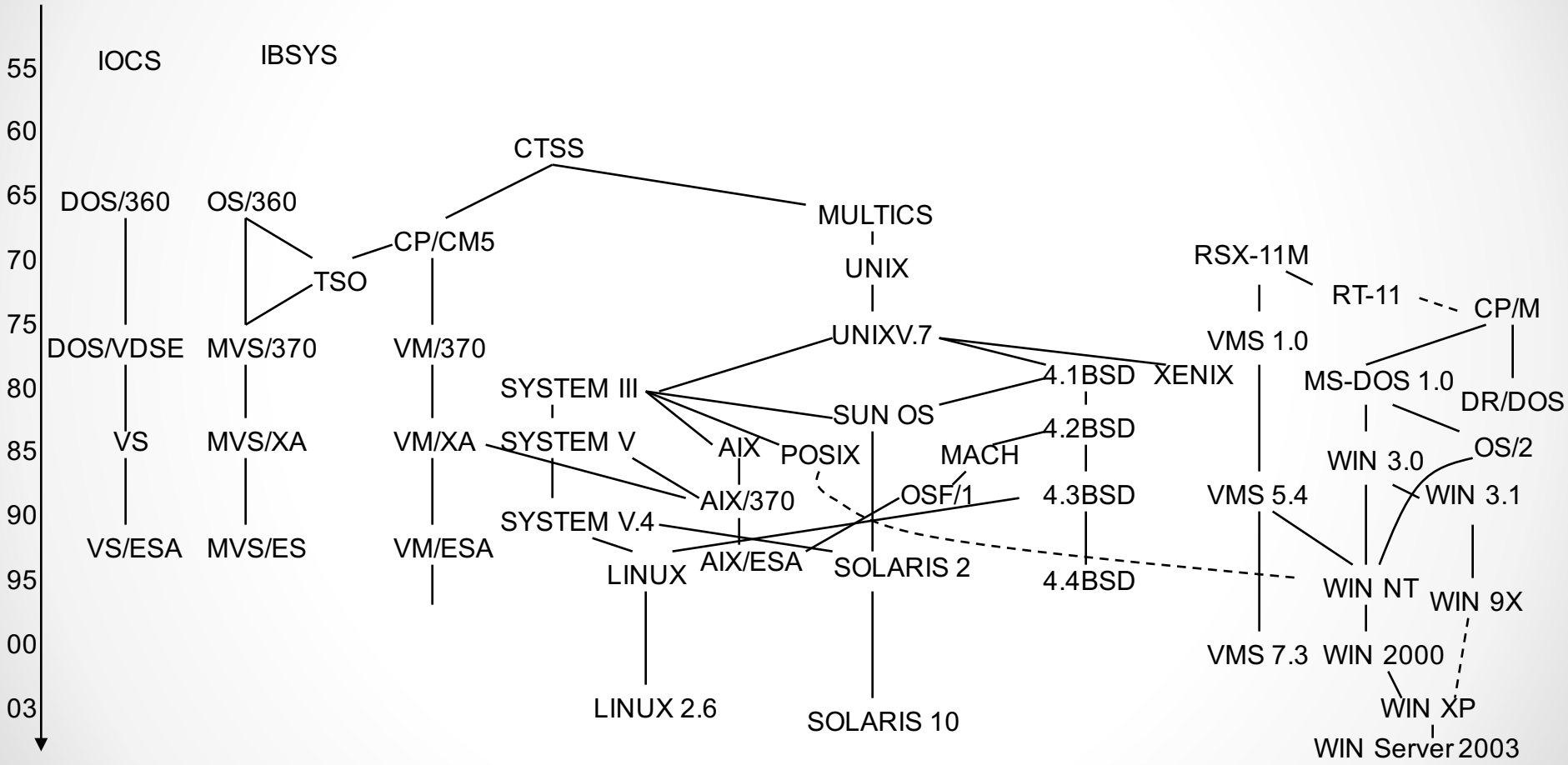

- Same Operating System, different machine: Unix



# Evolution: continuous changes have to be adapted

- To the new users demands:
  - Voice recognition, multi-touch input, etc.
- To the evolution (or new kind) of hardware:
  - Drivers for all type of new devices
  - Multicore systems, virtualization, etc.
- To integrate different environments solutions:
  - Batch processing, multiprogramming, time-sharing, etc.
  - Multiuser, collaborative work, etc.
  - Distributed systems, network services, etc.

# Operating system evolution

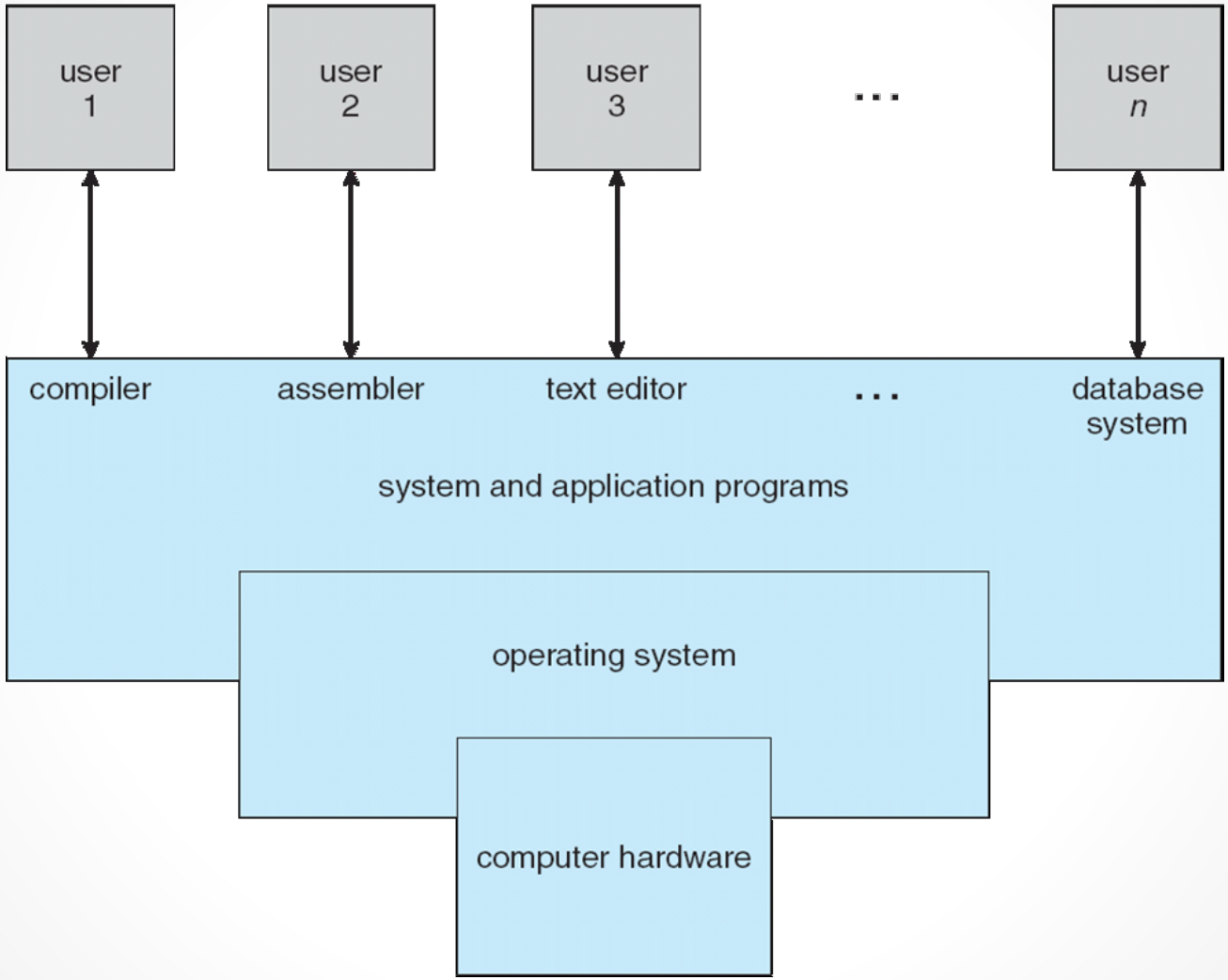



https://www.microsoft.com/resources/sharedsource/windowsacademic/curriculumresourcekit.mspx

# Computer System Structure

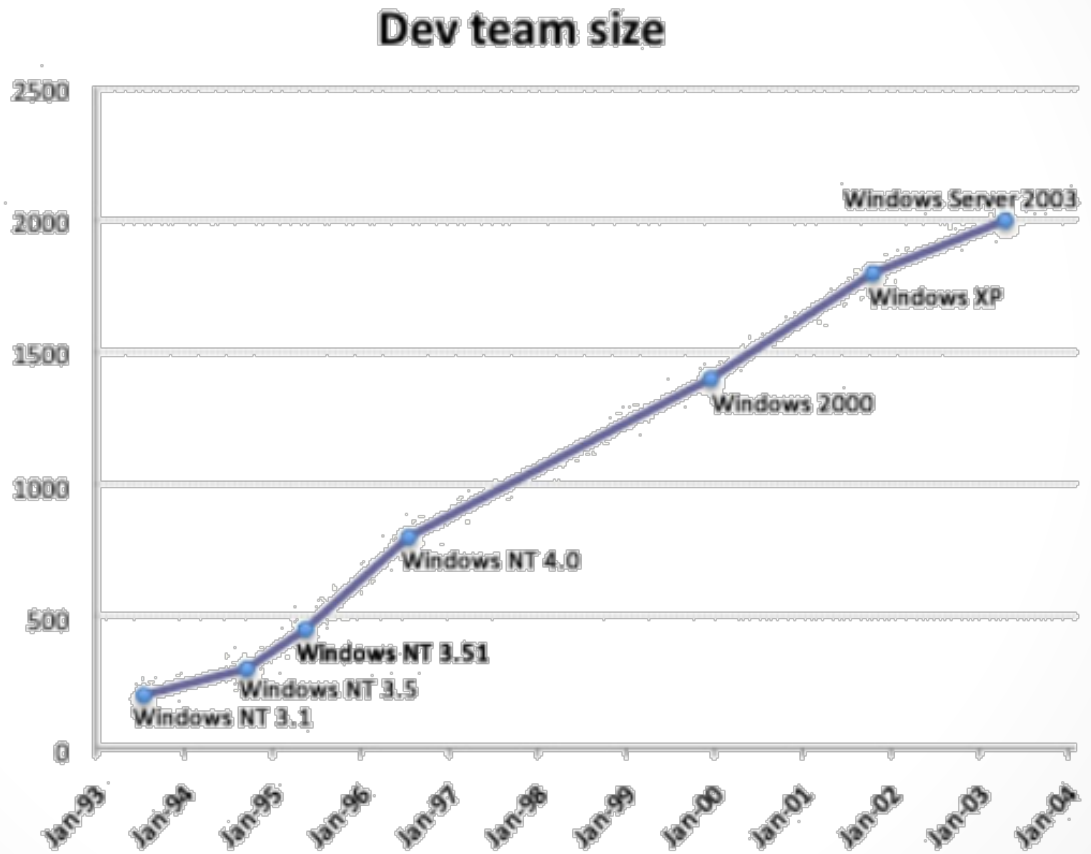
- Computer system can be divided into four components:
  - Hardware – provides basic computing resources
    - CPU, memory, I/O devices
  - Operating system
    - Controls and coordinates use of hardware among various applications and users
  - Applications – define the ways in which the system resources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

# Four Components of a Computer System



# Complexity of the Operating Systems

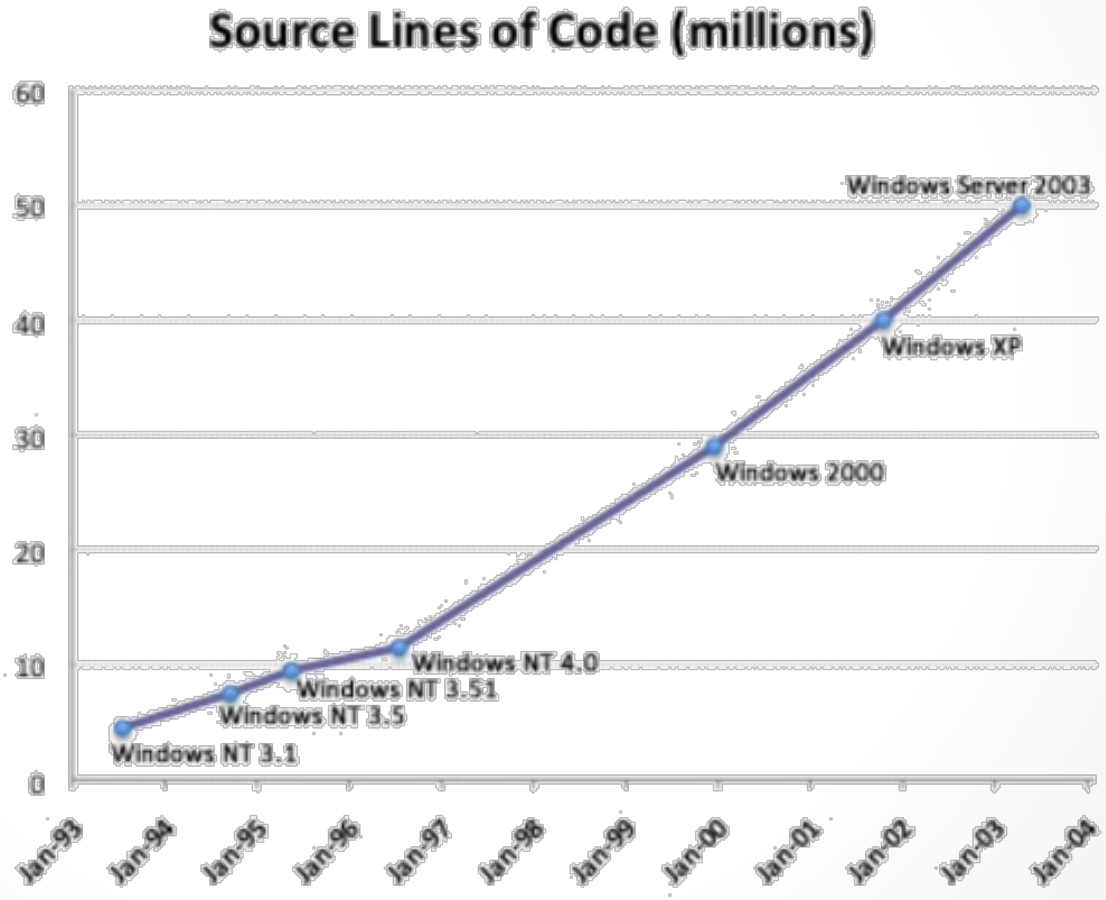
## Windows




http://www.zdnet.co.uk/reviews/desktop-os/2010/11/20/a-quarter-century-of-windows-40090900/5/#top

# Complexity of the Operating Systems

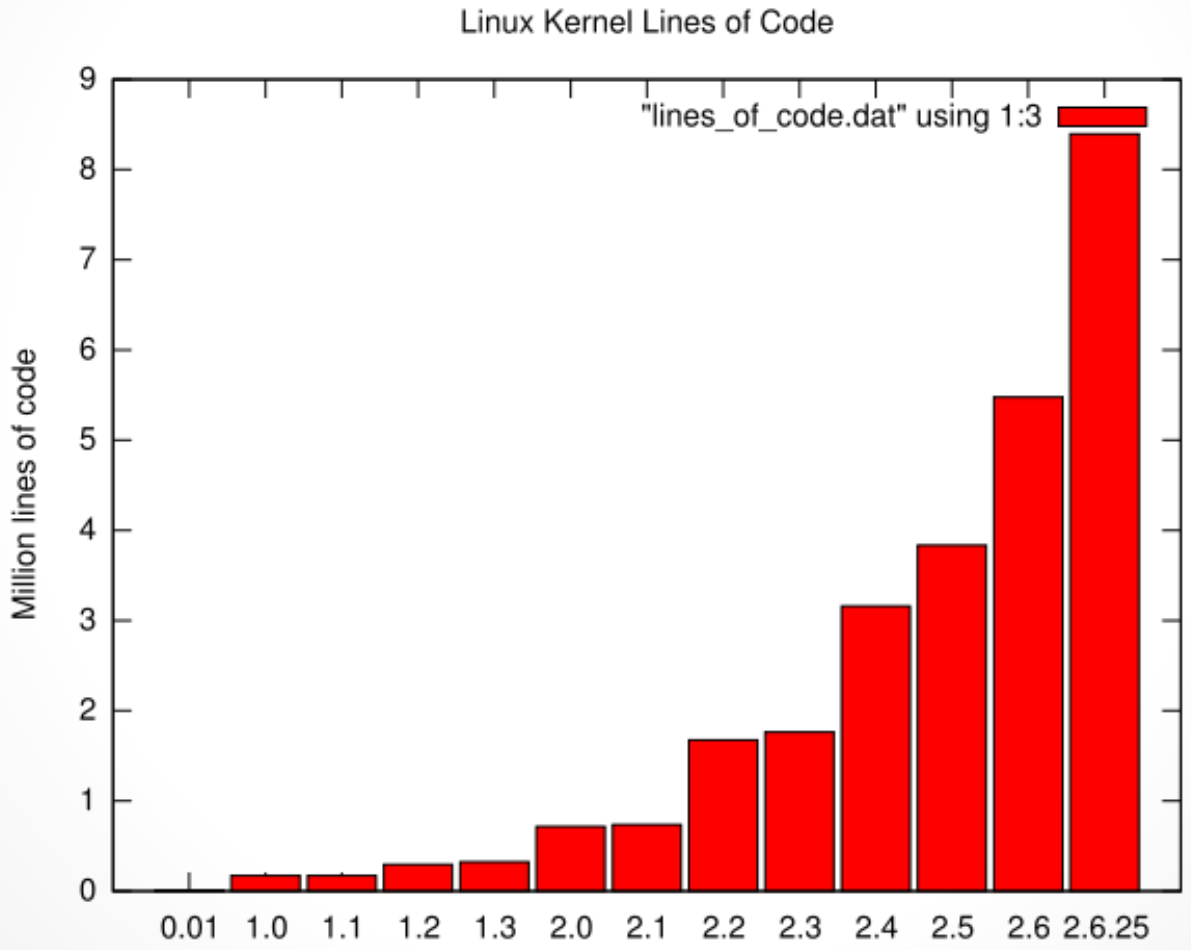
## Windows




http://www.zdnet.co.uk/reviews/desktop-os/2010/11/20/a-quarter-century-of-windows-40090900/5/#top

# Complexity of the Operating Systems

## Linux




Fedora Core 9 => ~200

https://secure.wikimedia.org/wikipedia/de/wiki/Datei:Linux_kernel_loc.svg
http://blogs.computerworld.com/204_5_million_lines_of_code_equals_one_great_linux_distribution

# Complex Multidisciplinary Software

- Multidisciplinary Software:
  - User Interface, System Software, Artificial Intelligence, etc.
- Complex Software:
  - Lots of lines of code
  - Lots of working groups required
- Sensitive Software:
  - A driver failure (software for a device) can hang up all the system.
  - Management of several application data from several users without losing anything or move data to incorrect user.

# Topics

1. What an Operating System Is.
2. **What an Operating System does.**
3. Operating System Structure.

# Operating System Definition

- OS is a **resource allocator**
  - Manages all resources
  - Decides between conflicting requests for efficient and fair use of resources
- OS is a **control program**
  - Controls execution of programs to prevent errors and improper use of the computer

# Operating System Definition (Cont'd)

- No universally accepted definition
- "Everything a vendor ships when you order an operating system" is good approximation
  - But varies widely

- "The kernel is a continuously-running computer program that constitutes the central core of a OS. Everything else is either a system program (ships with the operating system) or an application program

# Operating system main tasks

User

Operating System

- User interface (system calls).
- Extended machine:
  - Services, programmer interface, etc.
- Resource manager:
  - CPU, memory, etc.

Hardware

# User interface

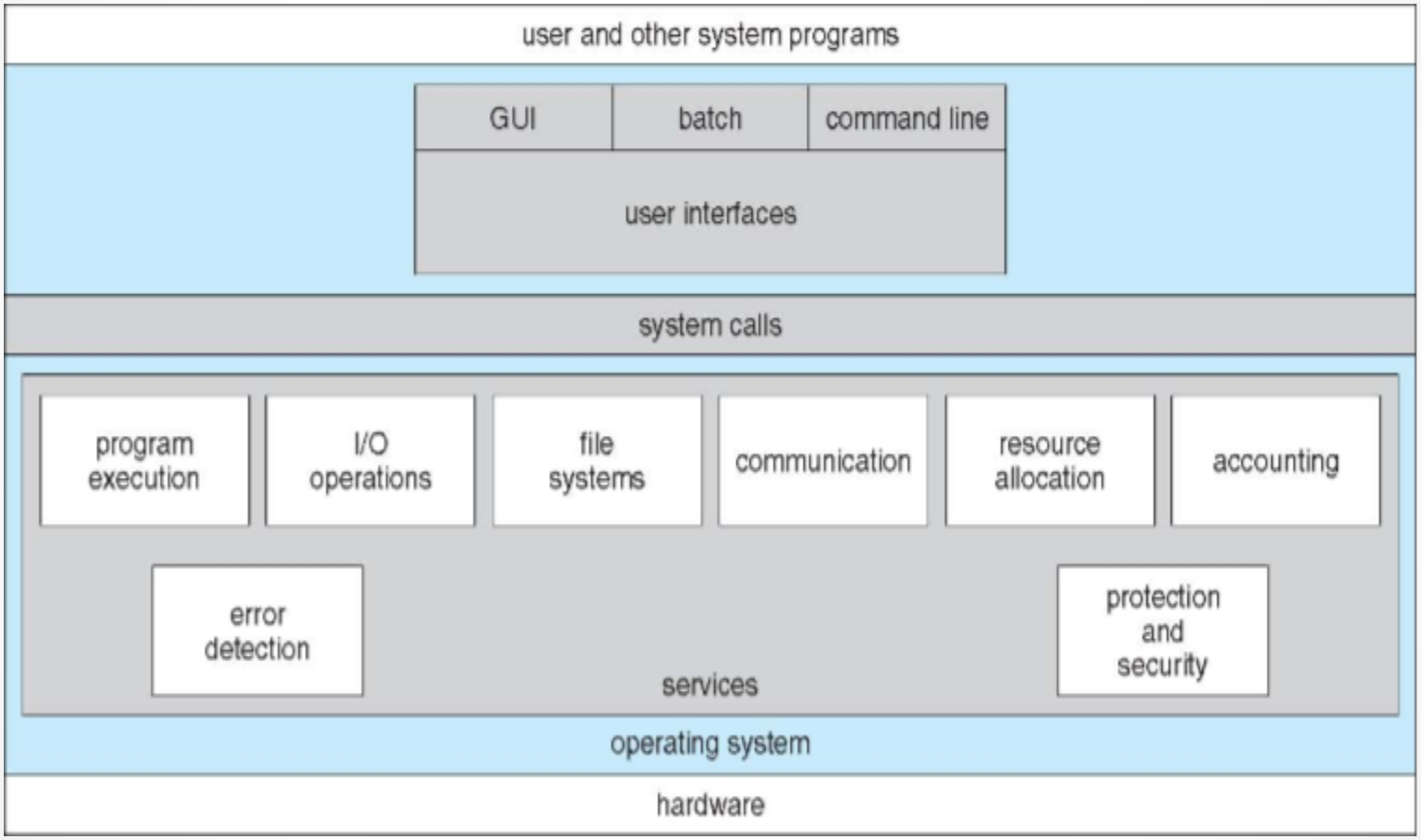
- Programmer interface:
  Extended machine interface.
  - Through system calls.

- User interface:
  User/OS interaction.
  - Through *shell*:
    - Command Line Interface or CLI
    - Graphic User Interface or GUI

# Main operating system areas

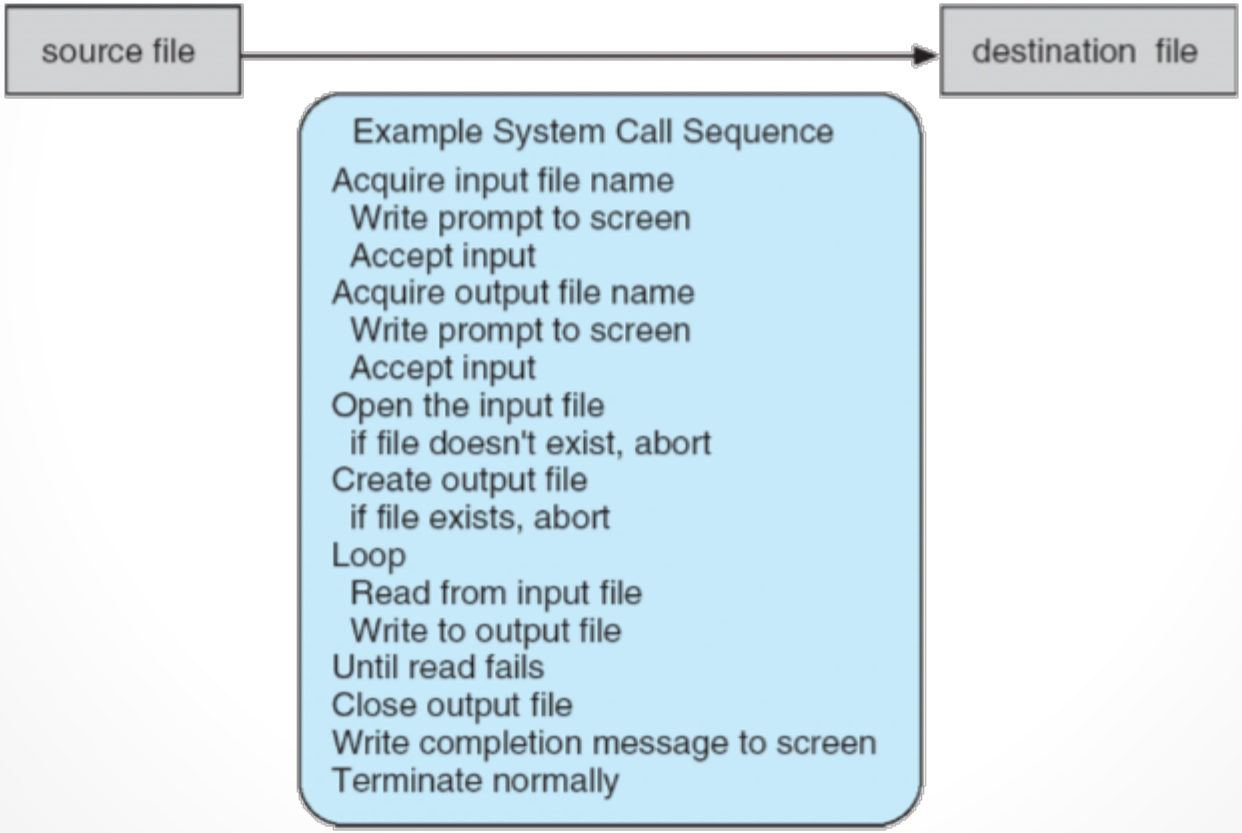
- Process management – Scheduling
  - Scheduling
  - Priorities, multiuser
- Memory management
  - To split the memory among the process, with protection and sharing
- Storage management – File systems
  - To offer an unified logical vision for user and programs that is independent of the physical medium
- Devices management
  - Covering the hardware dependencies
  - Concurrent access management support

# A View of Operating System Services

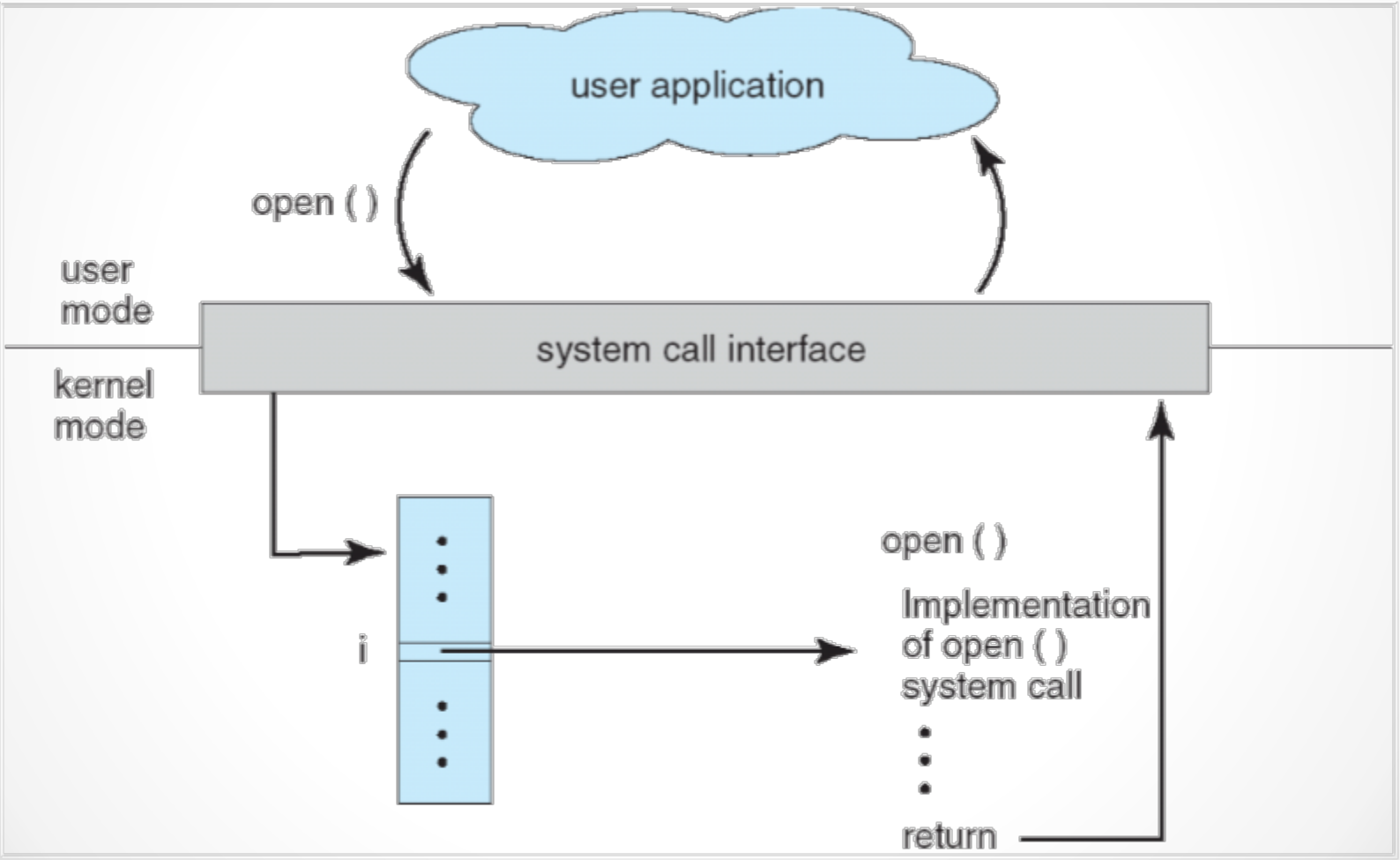


# Example of System Calls

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



# API – System Call – OS Relationship



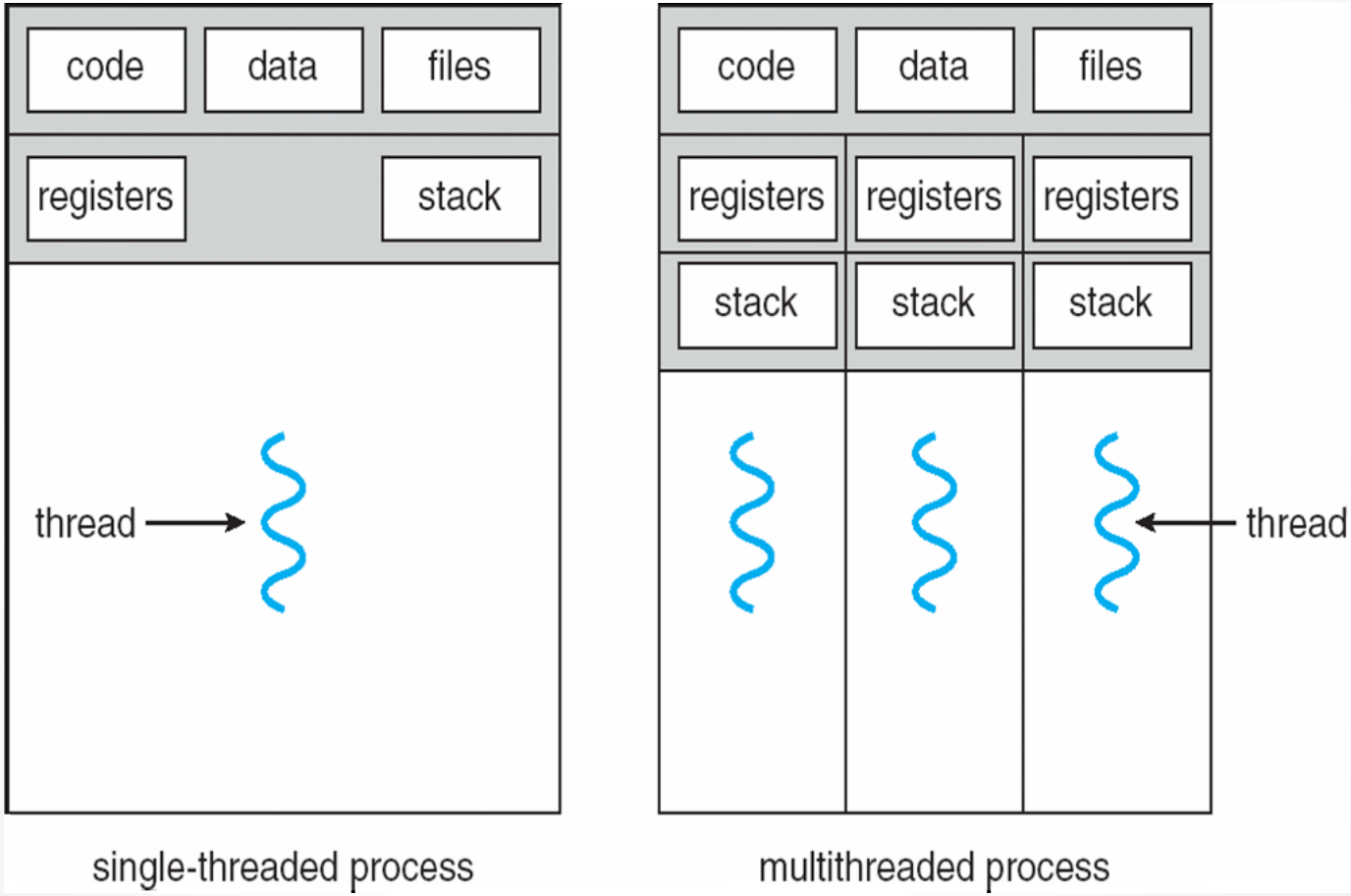
# Process Management

- A process is a program in execution. It is a unit of work within the system. Program is a *passive entity*, process is an *active entity*.
- Process needs resources to accomplish its task
  - CPU, memory, I/O, files
  - Initialization data
- Process termination requires reclaim of any reusable resources
- Single-threaded process has one **program counter** specifying location of next instruction to execute
  - Process executes instructions sequentially, one at a time, until completion
- Multi-threaded process has one program counter per thread
- Typically a system has many processes, some user, some operating system running concurrently on one or more CPUs
  - Concurrency by multiplexing the CPUs among the processes / threads

# Process Management Activities

- Creating and deleting user and system processes
- Suspending and resuming processes
- Providing mechanisms for process synchronization
- Providing mechanisms for process communication
- Providing mechanisms for deadlock handling

# Single and Multithreaded Processes



# Interprocess Communication

- Processes within a system may be **independent** or **cooperating**
- Cooperating process can affect or be affected by other processes, including sharing data
- Reasons for cooperating processes:
  - Information sharing
  - Computation speedup
  - Modularity
  - Convenience
- Cooperating processes need **interprocess communication** (**IPC**)
- Two models of IPC
  - Shared memory
  - Message passing

# Synchronization

- Race condition
- Critical section
- Locks
- Semaphores
- Deadlock
- Starvation
- Monitors

- Goal:  Avoiding DEADLOCKS!!

# Memory Management

- All data should be in memory before and after processing
- All instructions should be in memory in order to execute
- Memory management determines what is in memory when
  - Optimizing CPU utilization and computer response to users
- Memory management activities
  - Keeping track of which parts of memory are currently being used and by whom
  - Deciding which processes (or parts thereof) and data to move in and out of memory
  - Allocating and deallocating memory space as needed

# Storage Management

- OS provides uniform, logical view of information storage
  - Abstracts physical properties to logical storage unit - **file**
  - Each medium is controlled by a device (i.e., disk drive, tape drive)
    - Varying properties include access speed, capacity, data-transfer rate, access method (sequential or random)
- File-System management
  - Files are usually organized into directories
  - Access control on most systems to determine who can access what
  - OS activities include
    - Creating and deleting files and directories
    - Primitives to manipulate files and directories
    - Mapping files onto secondary storage
    - Backup files onto stable (non-volatile) storage media

# File Concept

- Storage abstractions
  - Contiguous logical address space
  - OS maps the logical space onto physical devices
  - Usually persistent
- Types:
  - Data
    - numeric
    - character
    - binary
  - Program
    - source
    - object
    - executable

# Mass-Storage Management

- Usually disks used to store data that does not fit in main memory or data that must be kept for a "long" period of time
- Proper management is of central importance
- OS activities
  - Free-space management
  - Storage allocation
  - Disk scheduling
- Some storage need not be fast
  - Tertiary storage includes optical storage, magnetic tape
  - Still must be managed
  - Varies between WORM (write-once, read-many-times) and RW (read-write)

# I/O Subsystem

- One purpose of the OS is to hide peculiarities of hardware devices from the user

- I/O subsystem responsible for
  - Memory management of I/O including buffering (storing data temporarily while it is being transferred), caching (storing parts of data in faster storage for performance), spooling (the overlapping of output of one job with input of other jobs)
  - General device-driver interface
  - Drivers for specific hardware devices

# Protection and Security

- **Protection** – any mechanism for controlling access of processes or users to resources defined by the OS
- **Security** – defense of the system against internal and external attacks
  - Huge range, including denial-of-service, worms, viruses, identity theft, theft of service

- Systems generally first distinguish among users, to determine who can do what
  - User identities (**user IDs**, security IDs) include name and associated number, one per user
  - User ID then associated with all files, processes of that user to determine access control
  - Group identifier (**group ID**) allows set of users to be defined and controls managed, then also associated with each process, file
  - **Privilege escalation** allows user to change to effective ID with more rights

# Topics

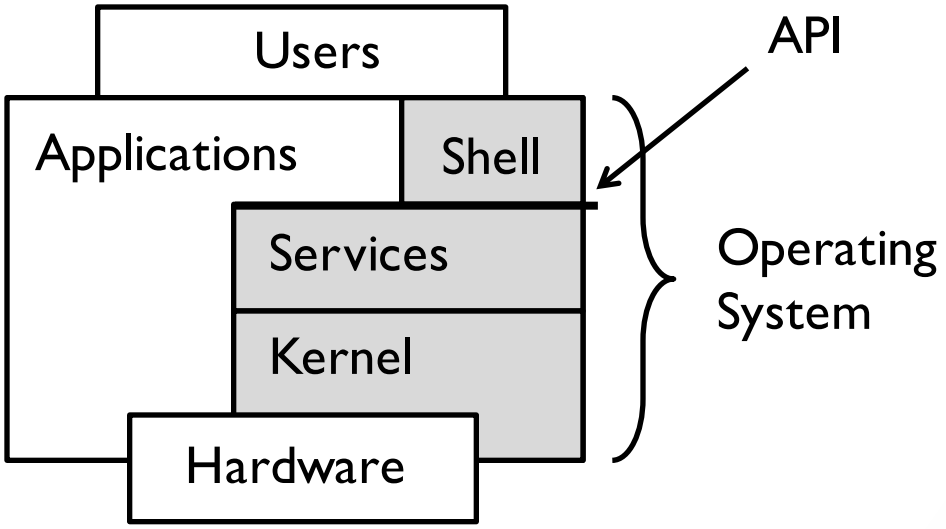
1. What an Operating System Is.
2. What an Operating System does.
3. **Operating System Structure.**

# 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

# Operating System general blocks

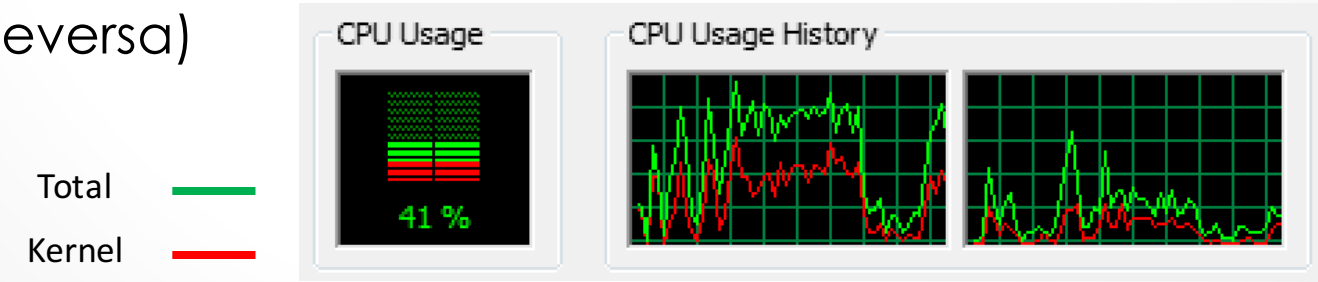


# What is the kernel?

- The operating system kernel is the core of the operating system.
- It is loaded on the machine's boot and it will stay resident until shutdown.
- The kernel activates the hardware devices and builds the first process (init) that will started the rest of the processes.
- Once loaded and initialized, it pass the control to the users processes that will execute until they need to perform some service request to the kernel (system call)
- The devices need to interrupt to the current process (or kernel) executing on CPU through hardware interruption; once the device has been treated, the kernel continues whatever was the interrupted activity.
- The kernel uses to include a small number of *kernel daemons*; there are processes that periodically execute kernel code and then block.

# Kernel and User Mode

- Modern systems use to use special hardware assistance in order to protect the kernel from malicious applications.
- There are some special bits (at least one) on the CPU named mode bits determining if the computer is executing in privilege (kernel mode) or in normal mode (user mode)
- The kernel code execute in kernel mode and the applications execute in user mode (kernel mode concept is different root privileges).
- The applications request services to the kernel through system calls that use controlled transfer from user mode to kernel mode (and viceversa)






http://www.codinghorror.com/blog/2008/01/understanding-user-and-kernel-mode.html

# Operating System Structure

- General types of operating system structures:
  - Monolithic
  - Subsystems
  - Layered
  - Microkernel

- Specific structures available:
  - Modules
  - Virtual machines

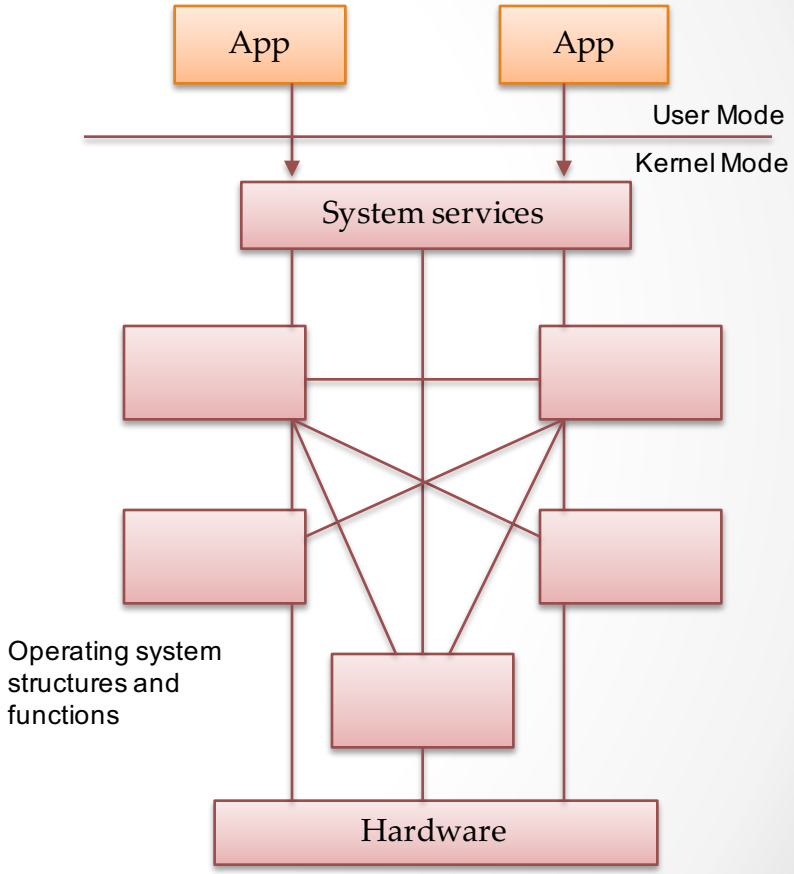
- Structures on some operating systems:
  - Linux
  - Windows 2000
  - Mac OS X
  - Minikernel

# Operating System Structure

## Monolithic (macrokernel)

- Monolithic system.
- Non-structured design.
- It is possible the access of any variable or function of the kernel from any kernel component.

- [I] very difficult maintenance and sensitive to errors

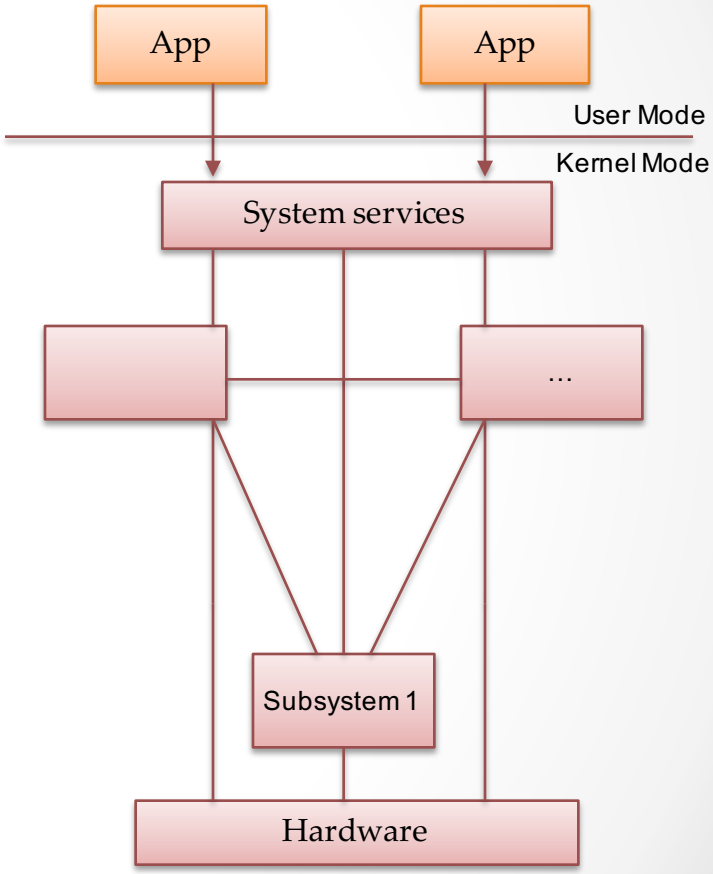


# Operating System Structure

## Subsystems

- Monolithic system composed of logical subsystems that offer well-defined interfaces as entry points.
- Grouped by related functions and structures.

- Example:
  - Linux

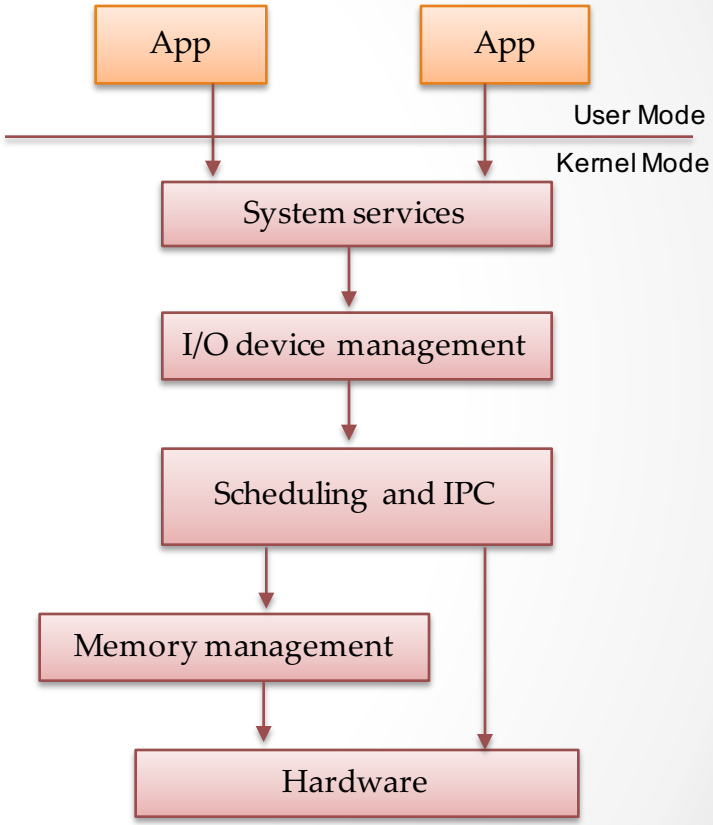


# Operating System Structure

## Layered

- Monolithic binary although designed with a logical layered structure
- Each layer only access to the interface of the inferior layer.

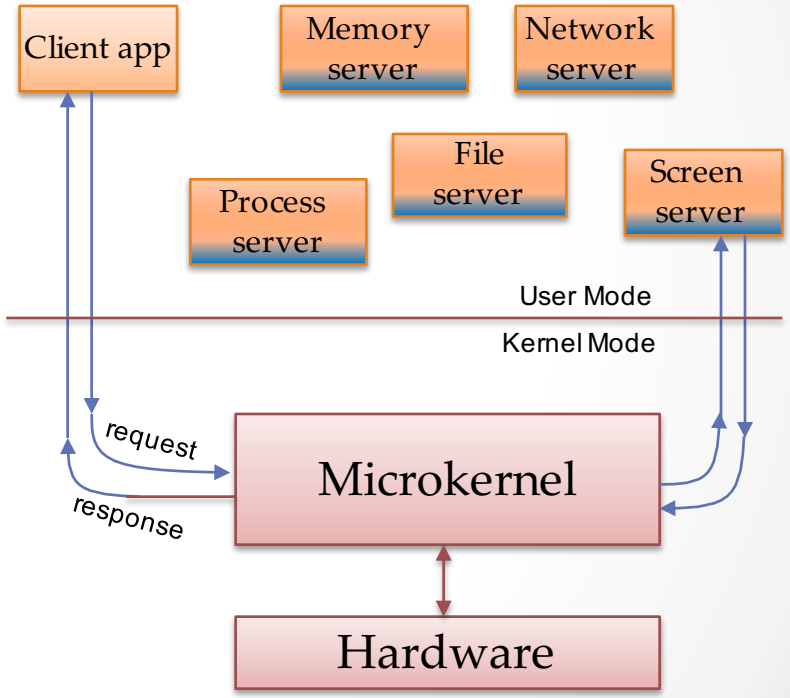
- Example:
  - THE (Dijkstra)
  - Multics, which added the notion of privileges rings



# Operating System Structure

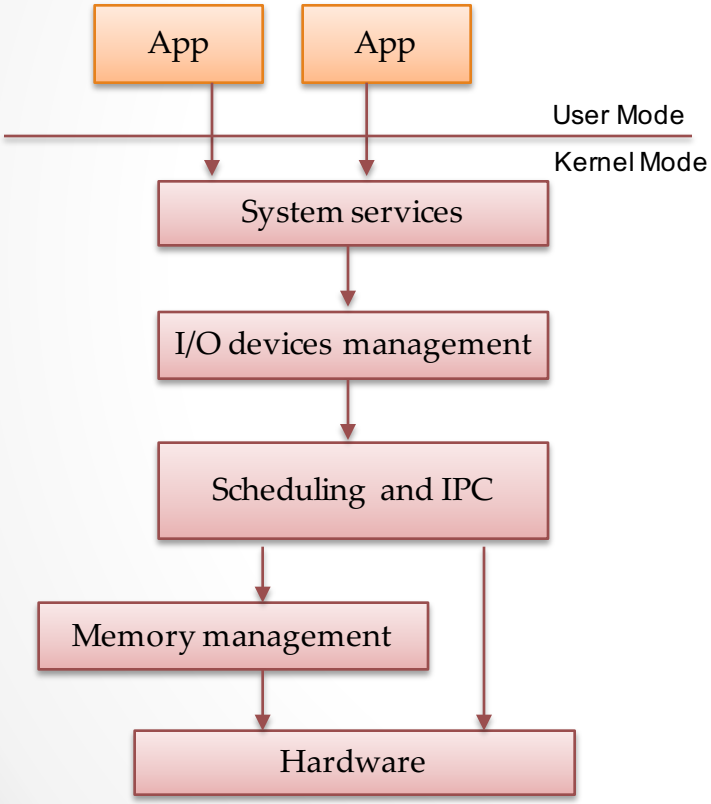
## Microkernel

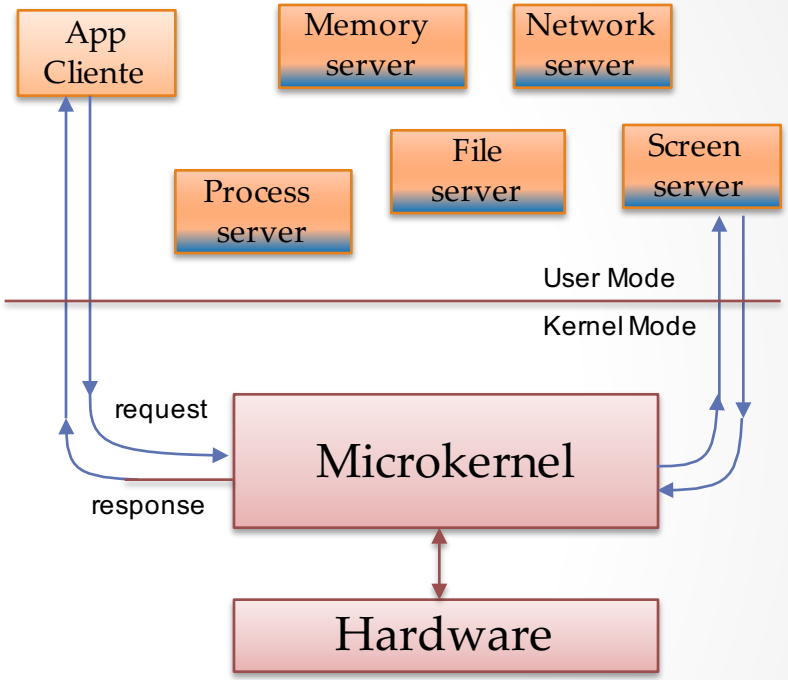
- Beyond structured design, the main components are executed as server process outside the kernel.
- The microkernel has:
  - Scheduling and process management.
  - Basic virtual memory management.
  - Basic process communication.
- Example:
  - Match, QNX, Minix, L4, etc.



# Operating System Structure

## Microkernel vs. Layered/Subsystems




Microkernel:

- [ok] more reliable and structured
- [ko] performance loss
  - Kernel mode to user mode change is expensive
- [ko] possible security problems
  - Eg.: *man in the middle*

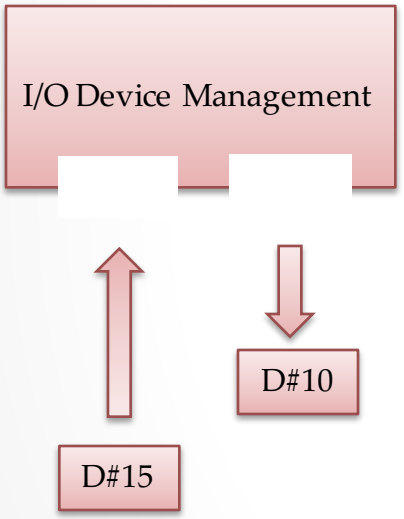
# Operating System Structure

- General types of operating system structures:
  - Monolithic
  - Subsystems
  - Layered
  - Microkernel

- Specific structures available:
  - Modules
  - Virtual machines

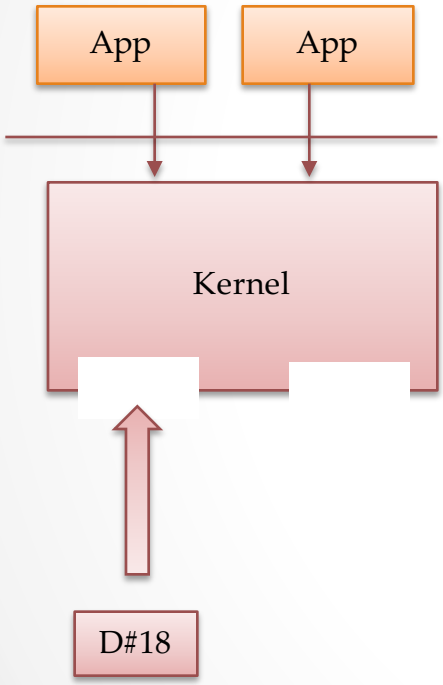
- Structures on some operating systems:
  - Linux
  - Windows 2000
  - Mac OS X
  - Minikernel

# Modules (1/2)



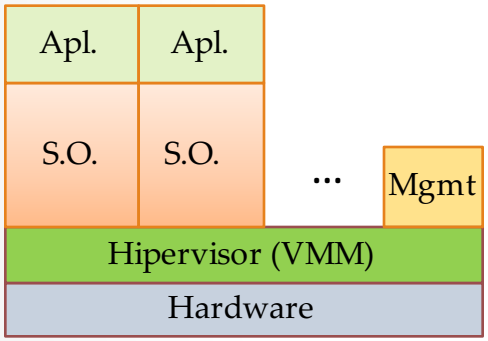

- The first kernels had to:
  - Include the code for all possible devices
  - Be recompiled for each new device added
- The modules initially were developed for the conditional inclusion of the device drivers
  - The modules can add dynamically the code for a pre-compiled driver.
  - Can be seen as the dynamic libraries for the kernel (kernel DLLs).
  - The module can be unloaded when the device is not longer used.

# Modules (2/2)




- The majority of the modern operating systems have a kernel that support the use of modules:
  - Linux, Solaris, BSD, Windows, etc.
- The modules are used nowadays not only for device drivers, but for adding other kinds of functionality:
  - The Linux kernel uses heavily for file systems, network protocols, system calls, etc.

# Virtual Machines




- The operating system provides some virtual hardware elements; why not everything virtual?
- IBM had used this idea into its mainframes since the beginning of the 70s. (1970 decade)
- An hypervisor virtualized all the computer so multiple operating system copies are executed simultaneously.
- The virtualization:
  - [ko] has some overload
  - [ok] provides us a excellent isolation among system and flexibility on the resource reservation, improving the costs especially on datacenters


http://www-128.ibm.com/developerworks/library/l-linuxvirt/index.html

# Operating System Structure

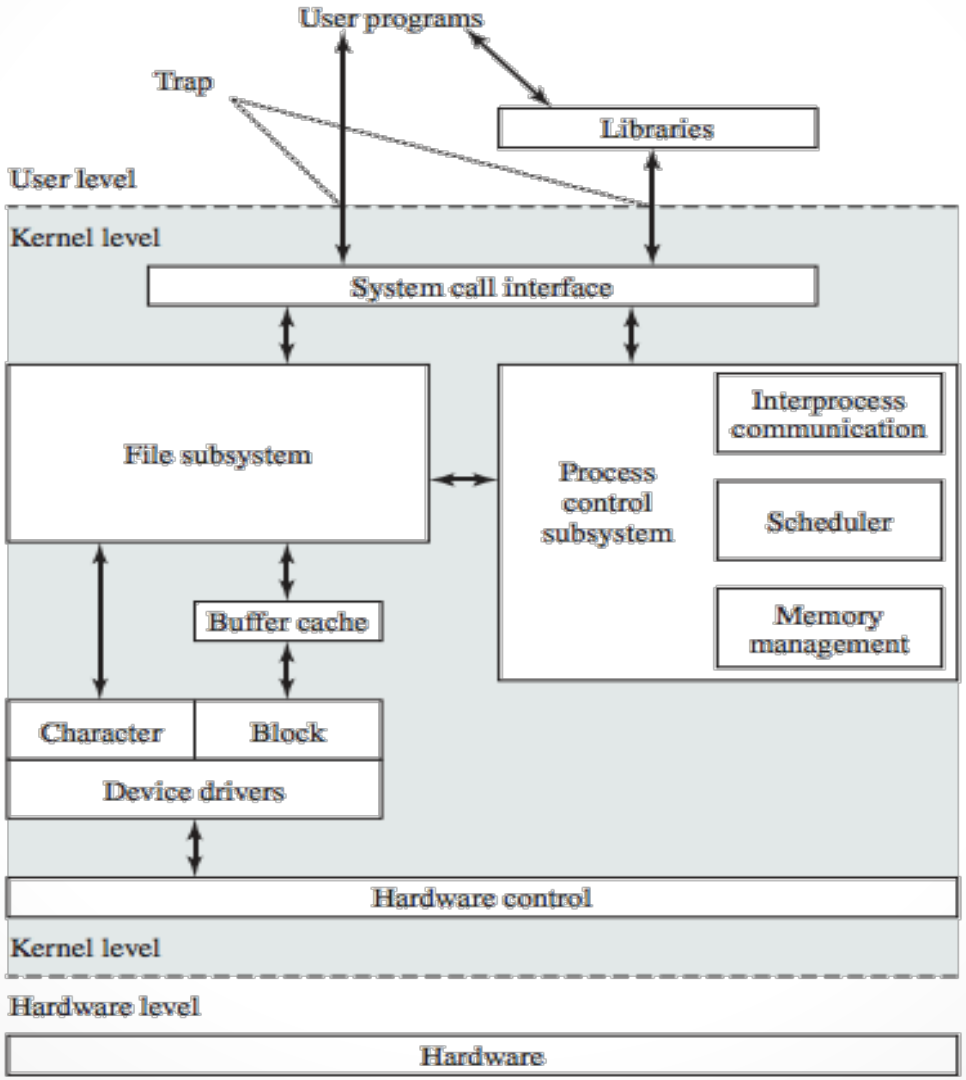
- General types of operating system structures:
  - Monolithic
  - Subsystems
  - Layered
  - Microkernel

- Specific structures available:
  - Modules
  - Virtual machines

- Structures on some operating systems:
  - Linux
  - Windows 2000
  - Mac OS X
  - Minikernel

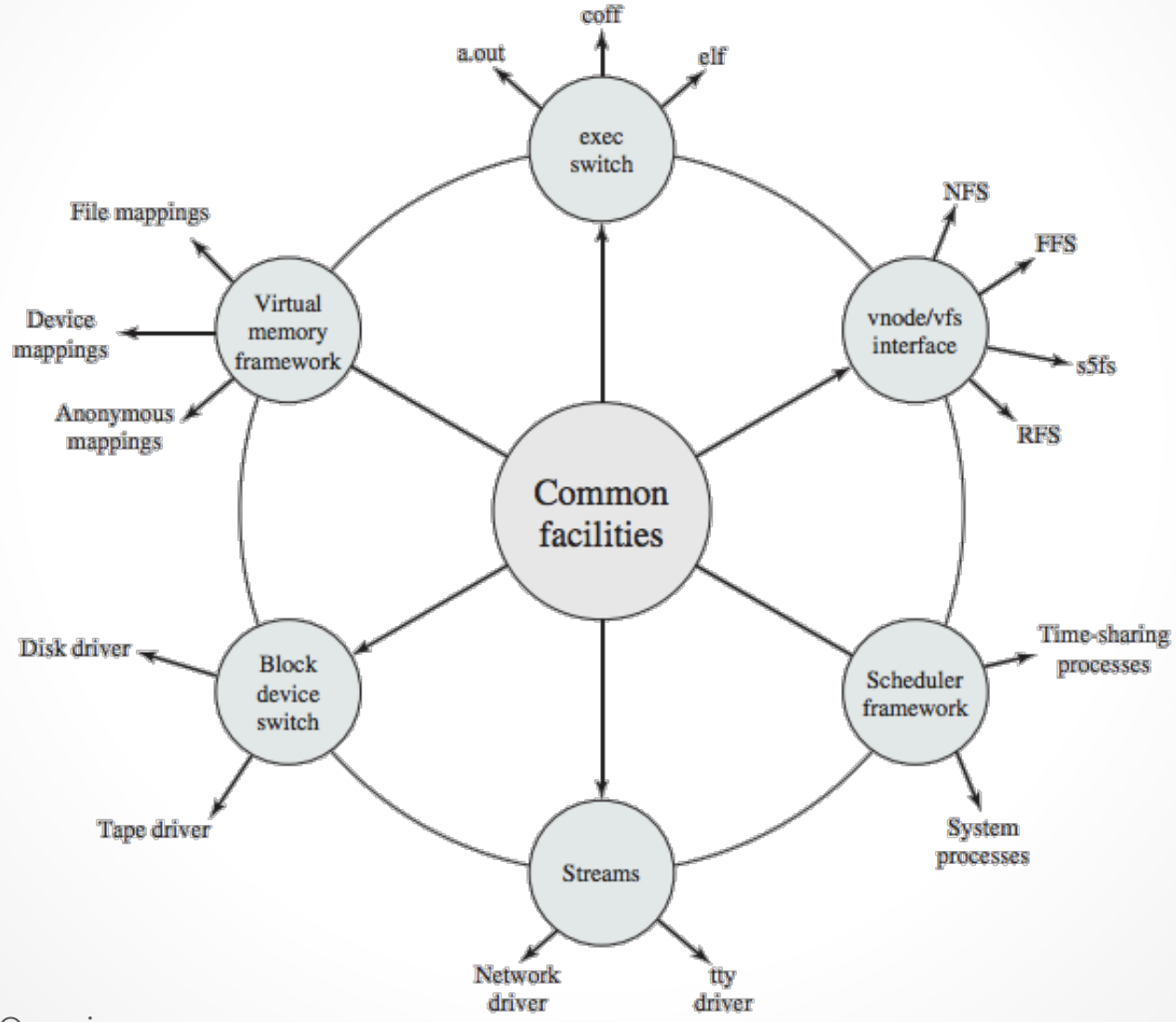
# Operating System Structure

## UNIX (old version)



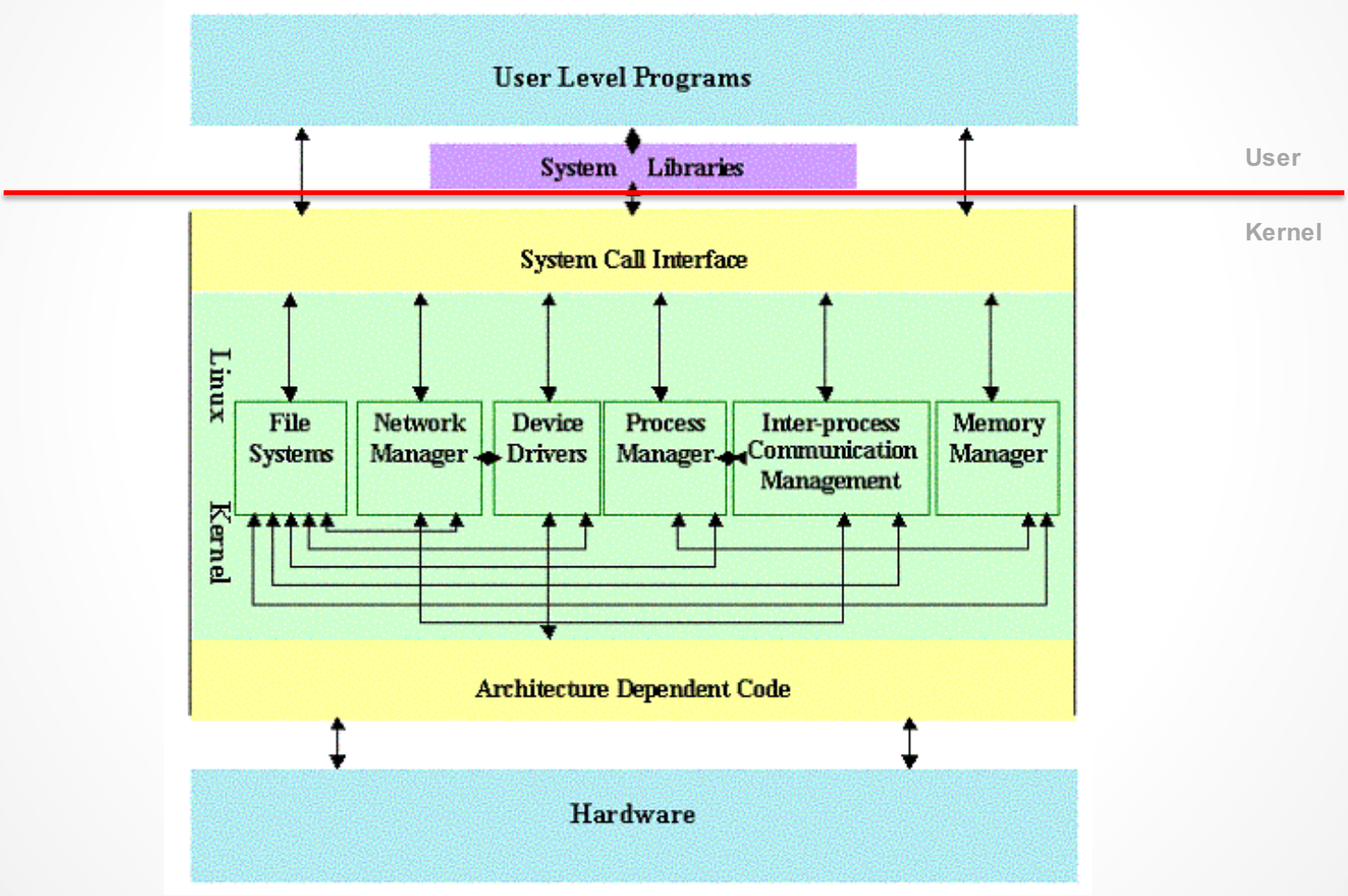
# Operating System Structure

## UNIX (modern version)



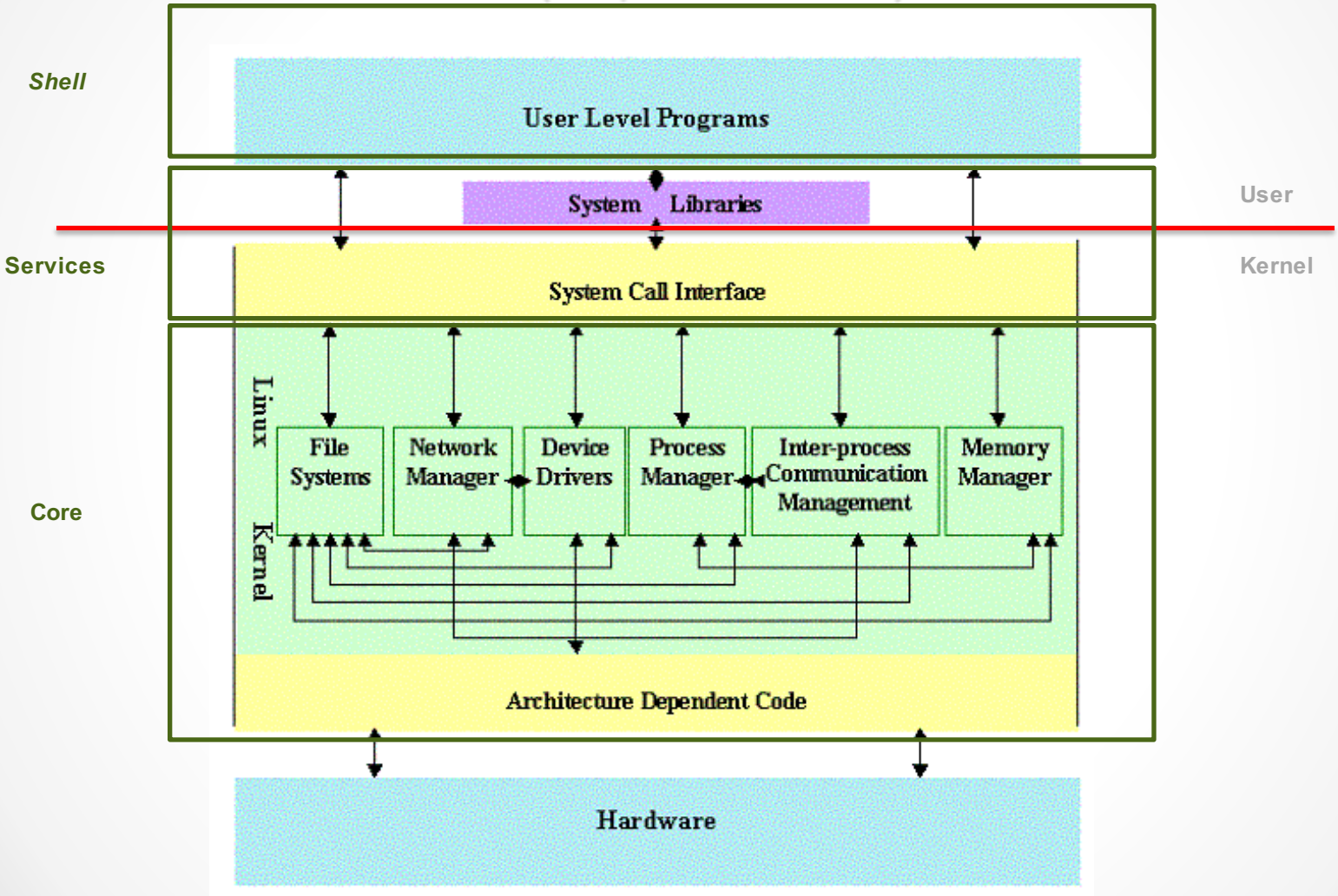
# Operating System Structure

## Linux (simplified version)

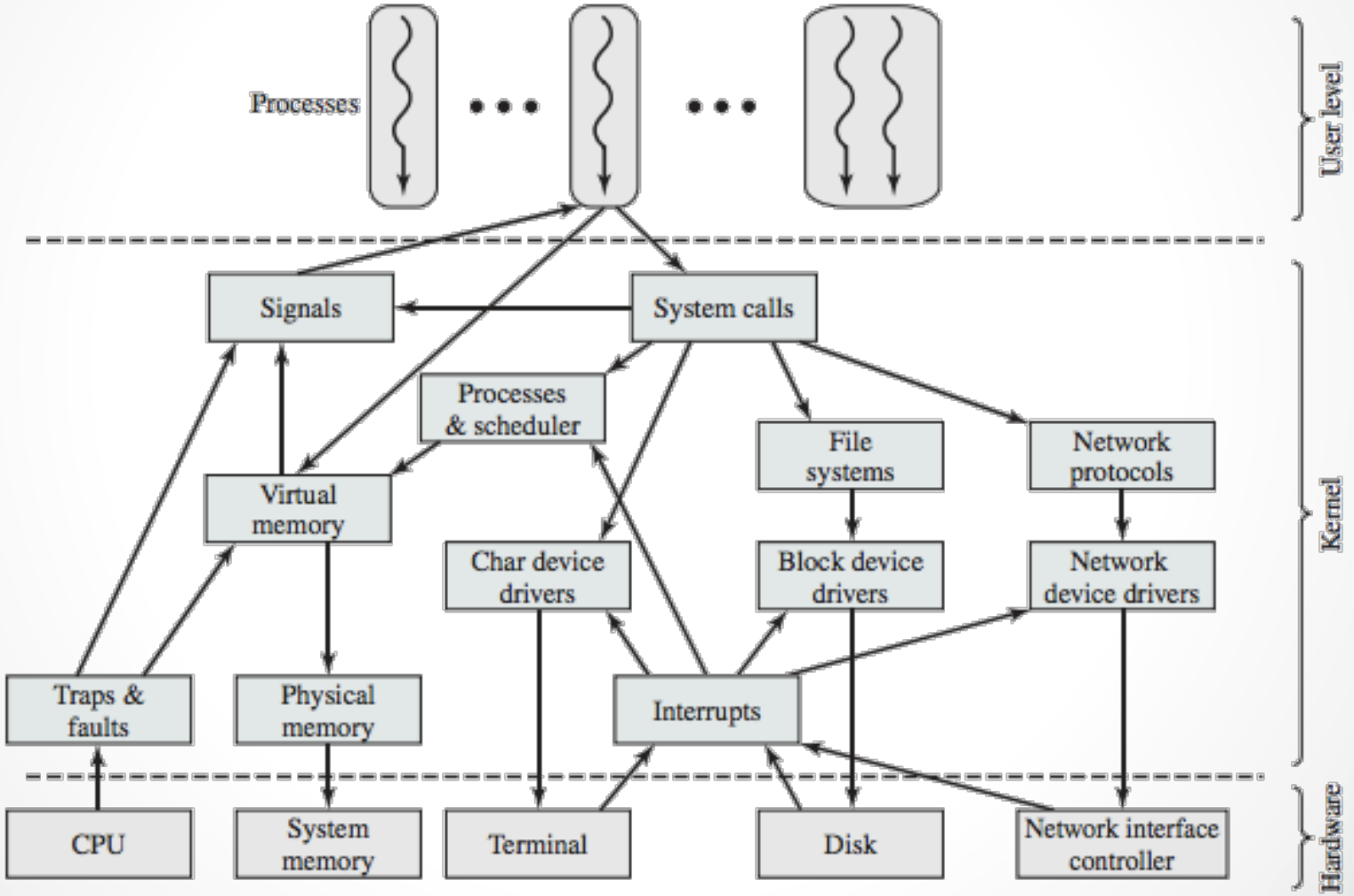


# Operating System Structure

## Linux (simplified version)

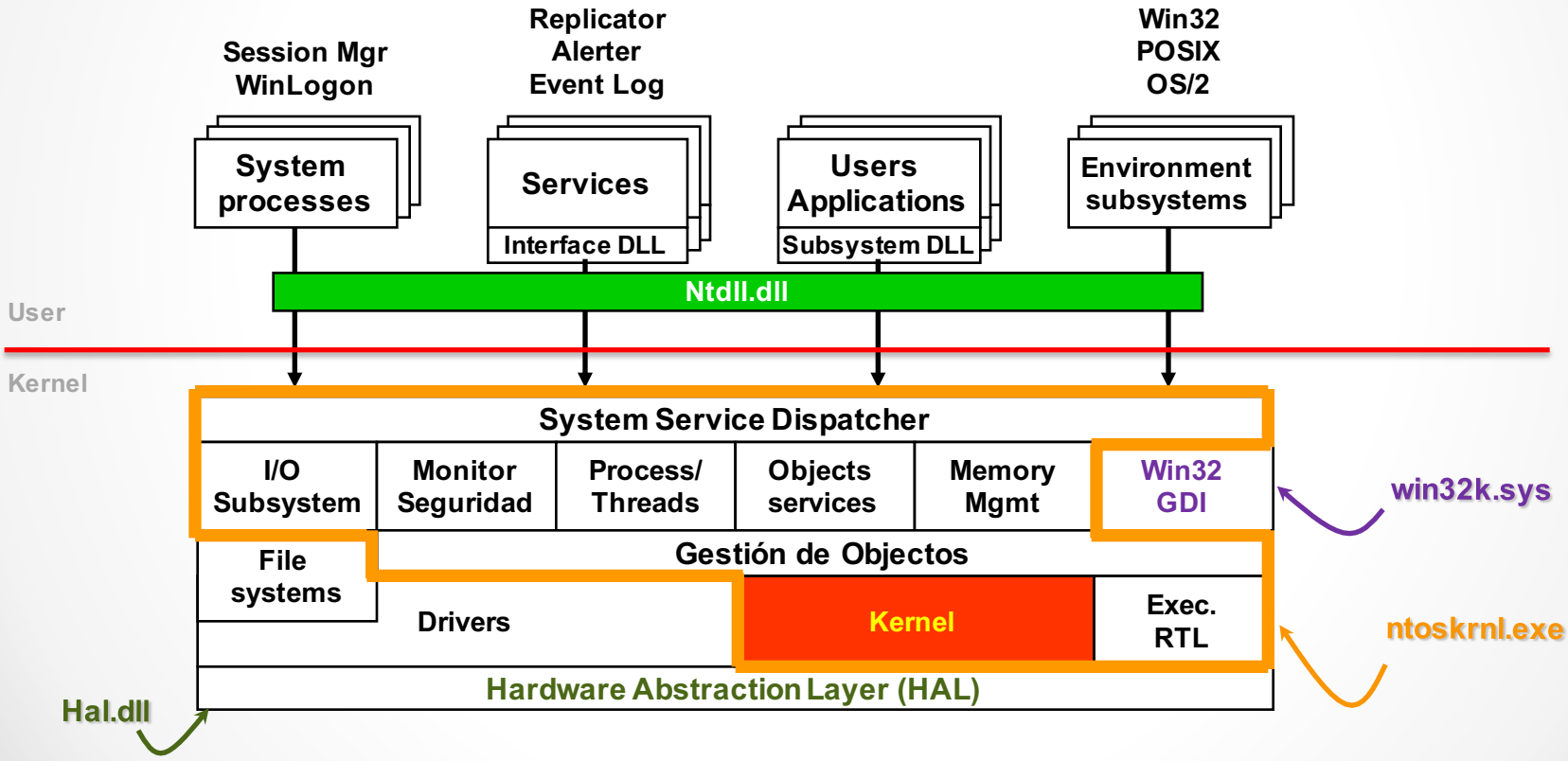


# Linux kernel components



# Operating System Structure

## Windows 2000



# Goals in the Operating System's design

- Performance: efficiency and speed
  - Low overhead,
    appropriated usage of resources
- Stability: robustness y resilience
  - Uptime, graceful degradation, reliability and integrity
- Capability: features, flexibility and compatibility
- Security and protection
  - Protect users from each others
  - System is secure for 'the bad guys'
- Portability
- Clarity
- Extensibility

# Operating System's design tradeoffs

- Butler Lampson:
  «choose any three design goals»
- Many goals are antagonistic:
  - Efficiency vs. protection
    - More checks, more overhead
  - Clarity vs. compatibility
    - Ugly implementation of "broken" standards (e.g. Unix's signal)
  - Flexibility vs. security
    - The more you can do, the more opportunities for security holes!
- But not all are antagonistic:
  - Portability tents to enhance code clarity

Grupo ARCOS

Operating Systems Design
Grado en Ingeniería Informática
Universidad Carlos III de Madrid