CSE 451: Operating Systems
Winter 2026

Module 3
Components and Structure

Gary Kimura

But first, a debugging aid

- Printf is your friend.
- OS work and C in particular is memory/address oriented. It is often useful to print out the address of a value in addition to its content

```
printf("[%016x] %x", (int)&p->Field, p->Field);
```

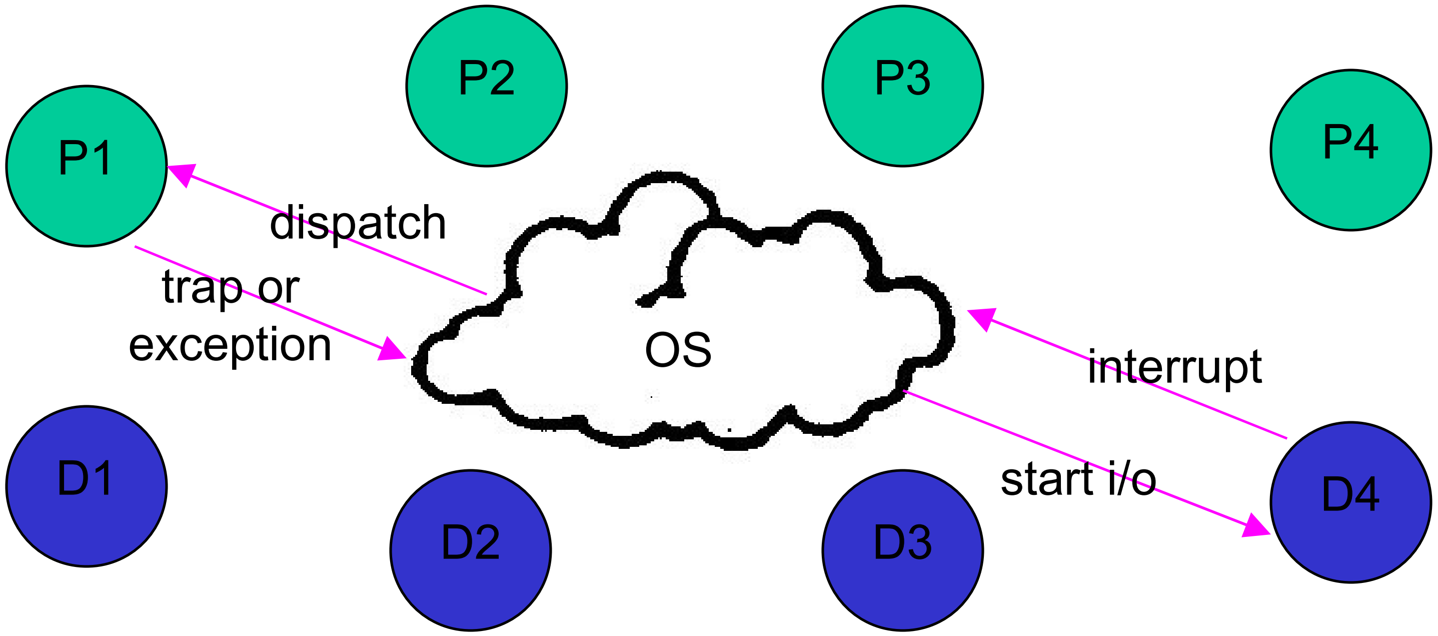
- And oftentimes I wondered where the printf was located

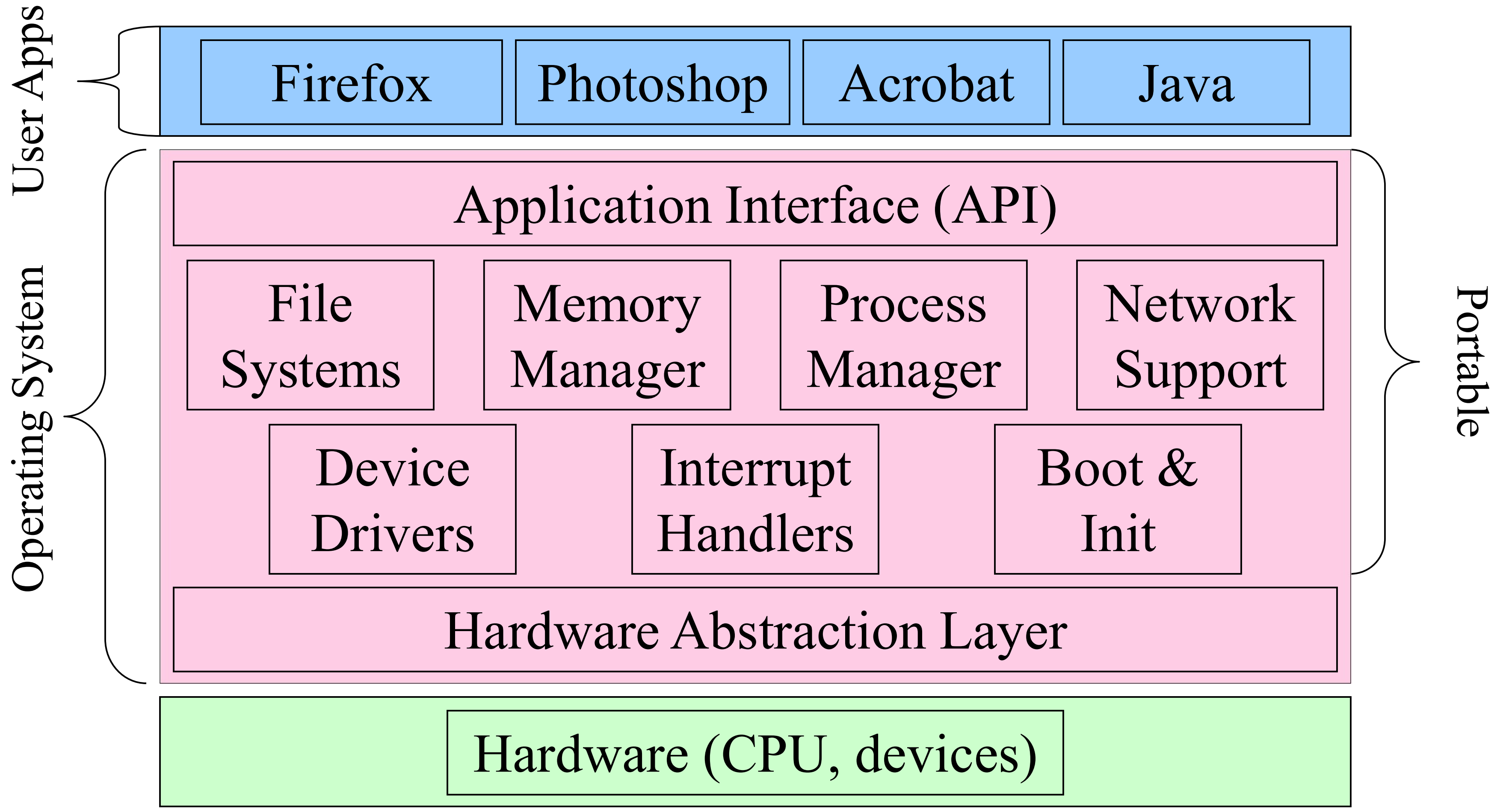
```
printf("[%s:%d]",__FILE__, __LINE__);
```

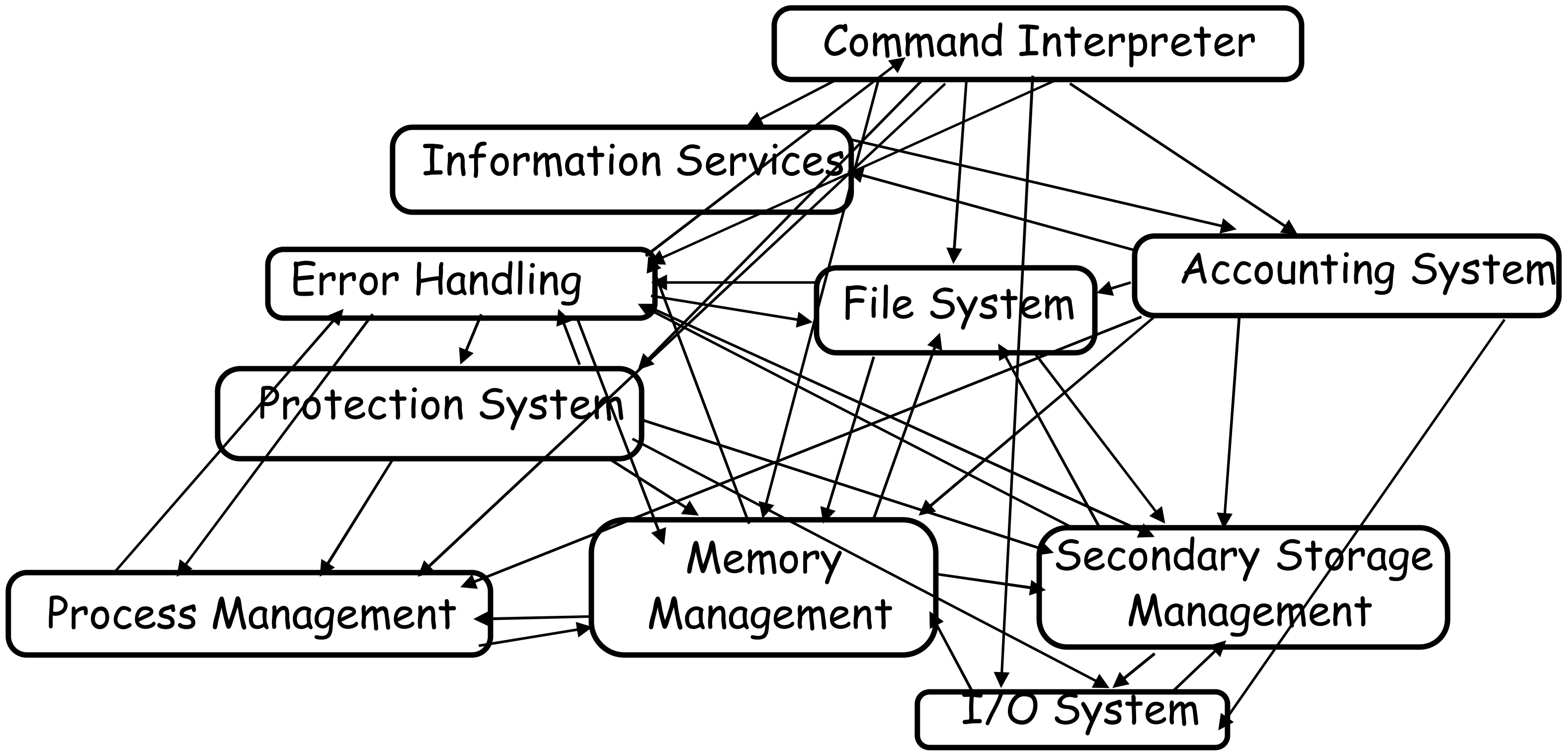
- Your mileage may vary

OS structure

- The OS sits between application programs and the hardware
  - it mediates access and abstracts away ugliness
  - programs request services via traps or exceptions
  - devices request attention via interrupts




User Apps
Firefox
Photoshop
Acrobat
Java
Operating System
Application Interface (API)
File Systems
Memory Manager
Process Manager
Network Support
Device Drivers
Interrupt Handlers
Boot & Init
Portable
Hardware Abstraction Layer
Hardware (CPU, devices)


Command Interpreter
Information Services
Error Handling
File System
Accounting System
Protection System
Memory Management
Process Management
Secondary Storage Management
I/O System

Major OS components

- processes/threads
- memory
- I/O
- secondary storage
- file systems
- protection
- shells (command interpreter, or OS UI)
- GUI
- networking

Process management

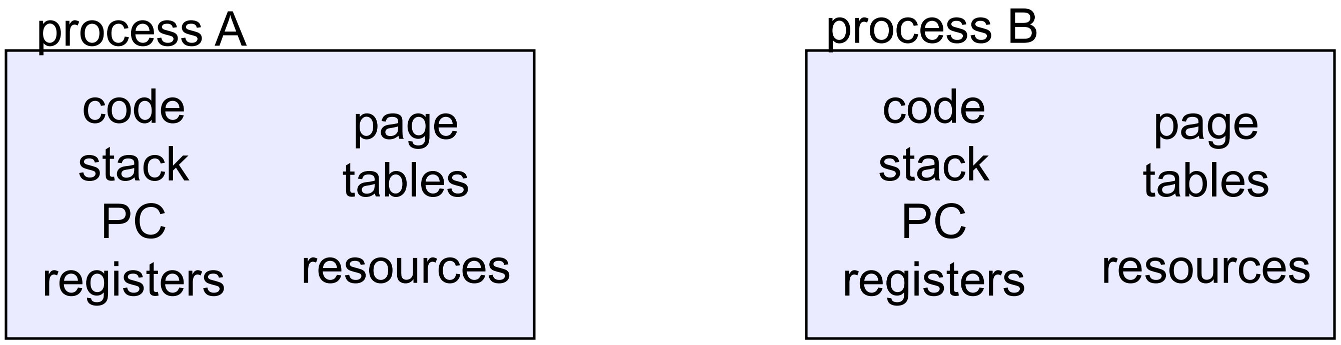
- An OS executes many kinds of activities:
  - users' programs
  - batch jobs or scripts
  - system programs
    - print spoolers, name servers, file servers, network daemons, …
- Each of these activities is encapsulated in a process
  - a process includes the execution context
    - PC, registers, VM, OS resources (e.g., open files), etc…
    - plus the program itself (code and data)
  - the OS's process module manages these processes
    - creation, destruction, scheduling, …

Important: Processes vs. Threads

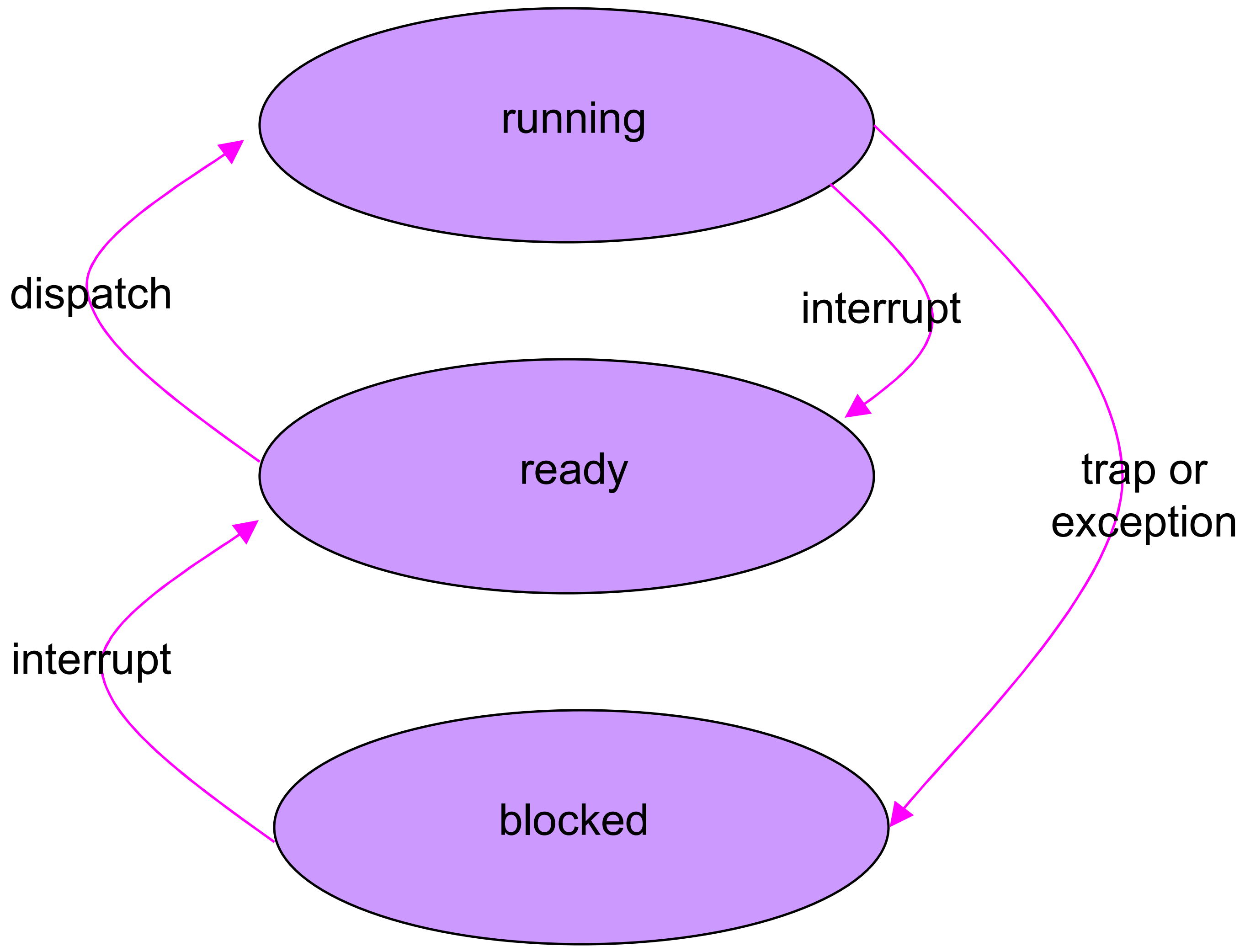
- Soon, we will separate the “thread of control” aspect of a process (program counter, call stack) from its other aspects (address space, open files, owner, etc.). And we will allow each {process / address space} to have multiple threads of control.
- But for now – for simplicity and for historical reasons – consider each {process / address space} to have a single thread of control.

Program/processor/process

- Note that a program is totally passive
  - just bytes on a disk that encode instructions to be run
- A process is an instance of a program being executed by a (real or virtual) processor
  - at any instant, there may be many processes running copies of the same program (e.g., an editor); each process is separate and (usually) independent
  - Linux: `ps -auwwx` to list all processes



States of a user process



Process operations

- The OS provides the following kinds operations on processes (i.e., the process abstraction interface):
  - create a process
  - delete a process
  - suspend a process
  - resume a process
  - clone a process
  - inter-process communication
  - inter-process synchronization
  - create/delete a child process (subprocess)

Memory management

- The primary memory is the directly accessed storage for the CPU
  - programs must be resident in memory to execute
  - memory access is fast
  - but memory doesn't survive power failures
- OS must:
  - allocate memory space for programs
  - deallocate space when needed by rest of system
  - maintain mappings from physical to virtual memory
    - through page tables
  - decide how much memory to allocate to each process
    - a policy decision
  - decide when to remove a process from memory
    - also policy

I/O

- A big chunk of the OS kernel deals with I/O
  - hundreds of thousands of lines in Windows, Unix, etc.
- The OS provides a standard interface between programs (user or system) and devices
  - file system (disk), sockets (network), frame buffer (video)
- Device drivers are the routines that interact with specific device types
  - encapsulates device-specific knowledge
    - e.g., how to initialize a device, how to request I/O, how to handle interrupts or errors
    - examples: SCSI device drivers, Ethernet card drivers, video card drivers, sound card drivers, …
- Note: Windows has ~35,000 device drivers!

Secondary storage

- Secondary storage (disk, FLASH, tape) is persistent memory
  - often magnetic media, survives power failures (hopefully)
- Routines that interact with disks are typically at a very low level in the OS
  - used by many components (file system, VM, …)
  - handle scheduling of disk operations, head movement, error handling, and often management of space on disks
- Usually independent of file system
  - although there may be cooperation
  - file system knowledge of device details can help optimize performance
    - e.g., place related files close together on disk

File systems

- Secondary storage devices are crude and awkward
  - e.g., “write a 4096 byte block to sector 12”
- File system: a convenient abstraction
  - defines logical objects like files and directories
    - hides details about where on disk files live
  - as well as operations on objects like read and write
    - read/write byte ranges instead of blocks
- A file is the basic unit of long-term storage
  - file = named collection of persistent information
- A directory is just a special kind of file
  - directory = named file that contains names of other files and metadata about those files (e.g., file size)
- Note: Sequential byte stream is only one possibility!

File system operations

- The file system interface defines standard operations:
  - file (or directory) creation and deletion
  - manipulation of files and directories (read, write, extend, rename, protect)
  - copy
  - lock
- File systems also provide higher level services
  - accounting and quotas
  - backup (must be incremental and online!)
  - (sometimes) indexing or search
  - (sometimes) file versioning

Protection

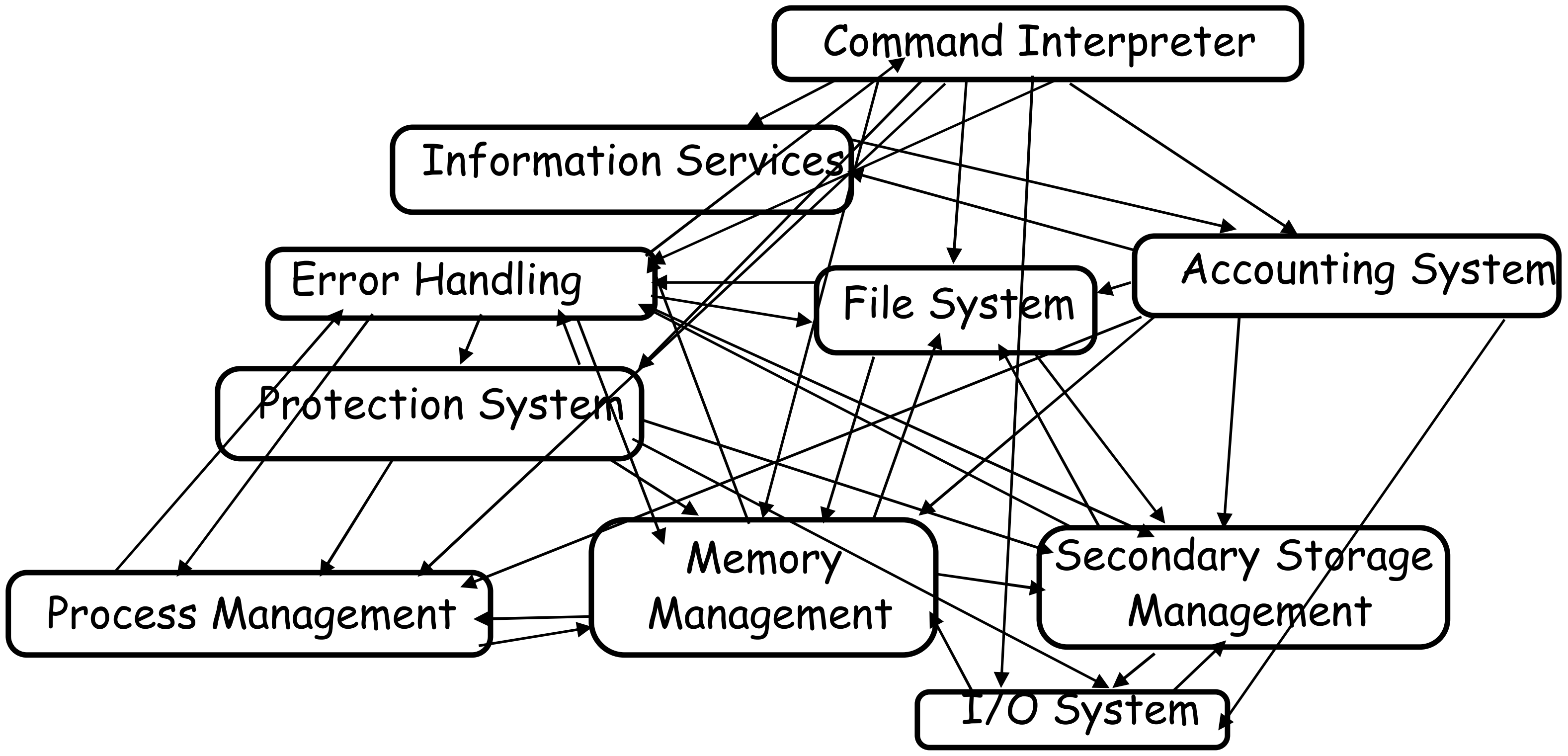
- Protection is a general mechanism used throughout the OS
  - all resources needed to be protected
    - memory
    - processes
    - files
    - devices
    - CPU time
    - …
  - protection mechanisms help to detect and contain unintentional errors, as well as preventing malicious destruction

Command interpreter (shell)

- A particular program that handles the interpretation of users' commands and helps to manage processes
  - user input may be from keyboard (command-line interface), from script files, or from the mouse (GUIs)
  - allows users to launch and control new programs
- On some systems, command interpreter may be a standard part of the OS (e.g., MS DOS, Apple II)
- On others, it's just non-privileged code that provides an interface to the user
  - e.g., bash/csh/tcsh/zsh on UNIX
- On others, there may be no command language
  - e.g., MacOS

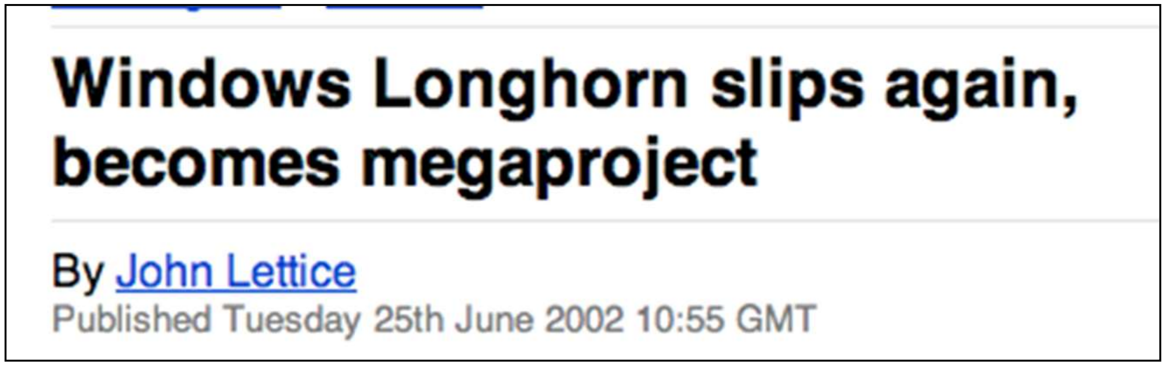
OS structure

- It's not always clear how to stitch OS modules together:

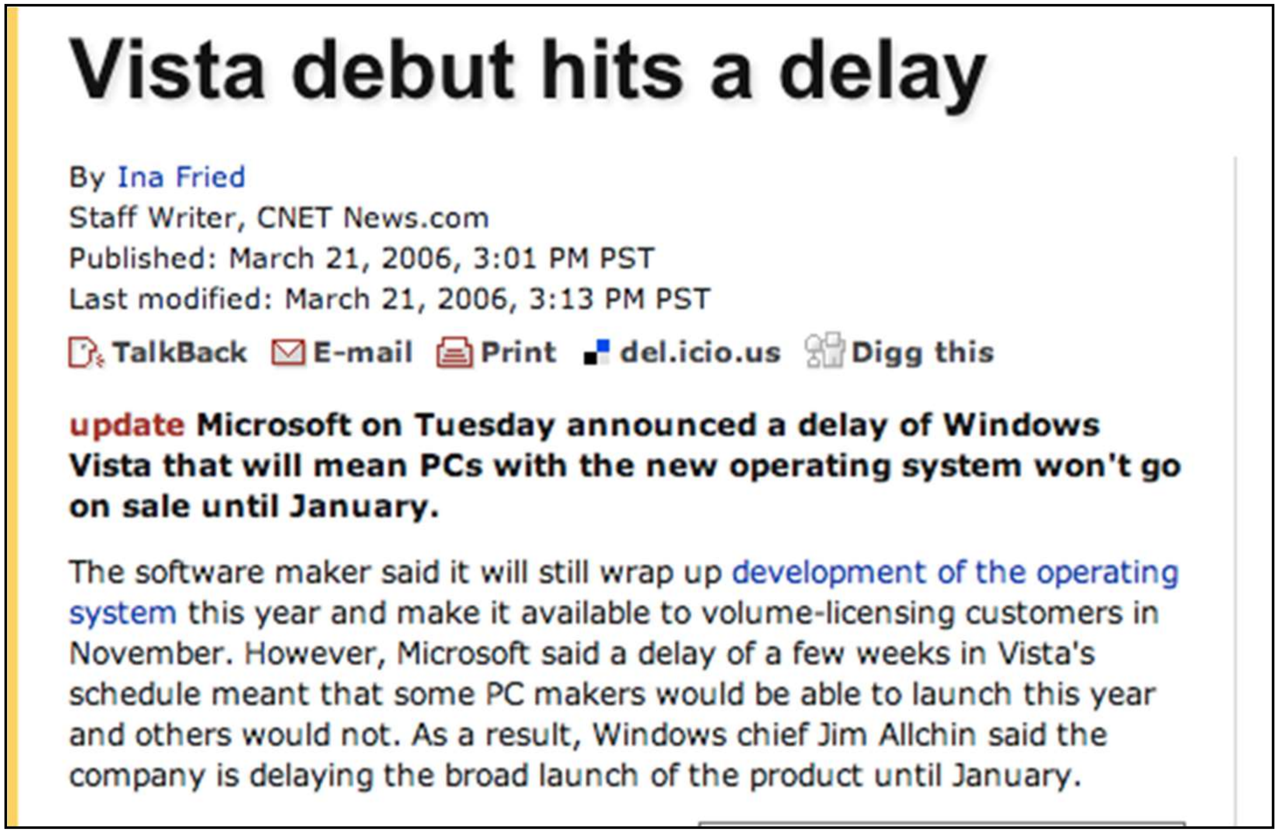


OS structure

- An OS consists of all of these components, plus:
  - many other components
  - system programs (privileged and non-privileged)
    - e.g., bootstrap code, the init program, …
- Major issue:
  - how do we organize all this?
  - what are all of the code modules, and where do they exist?
  - how do they cooperate?
- Massive software engineering and design problem
  - design a large, complex program that:
    - performs well, is reliable, is extensible, is backwards compatible, …


Windows Longhorn slips again, becomes megaproject

By John Lettice
Published Tuesday 25th June 2002 10:55 GMT


Vista debut hits a delay

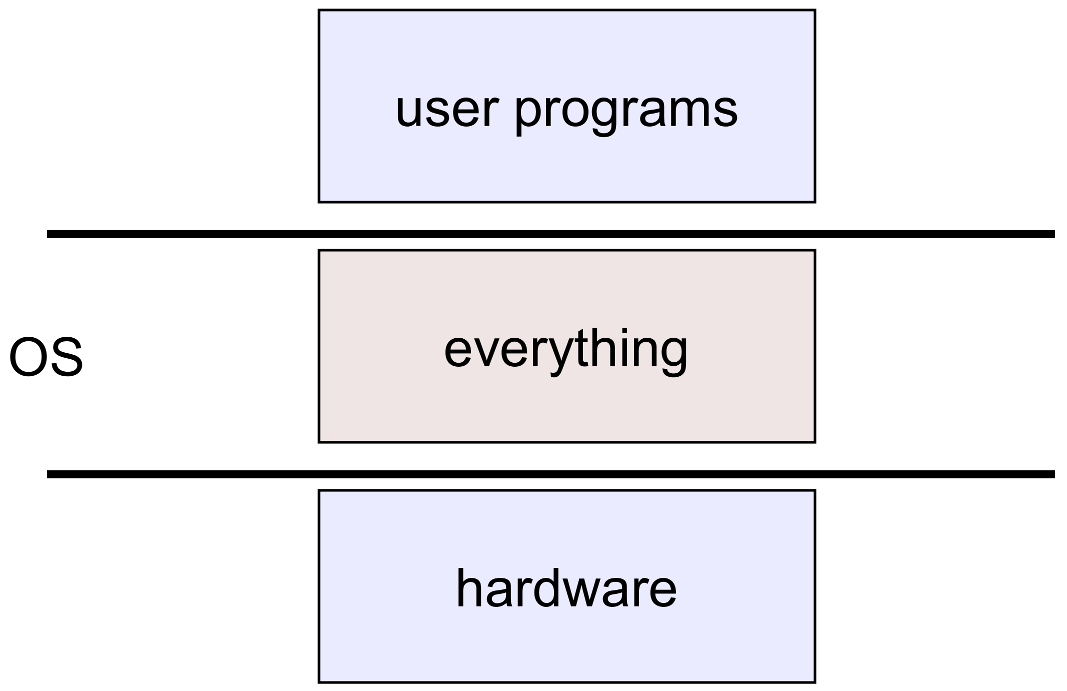
By Ina Fried
Staff Writer, CNET News.com
Published: March 21, 2006, 3:01 PM PST
Last modified: March 21, 2006, 3:13 PM PST

update Microsoft on Tuesday announced a delay of Windows Vista that will mean PCs with the new operating system won't go on sale until January.

The software maker said it will still wrap up development of the operating system this year and make it available to volume-licensing customers in November. However, Microsoft said a delay of a few weeks in Vista's schedule meant that some PC makers would be able to launch this year and others would not. As a result, Windows chief Jim Allchin said the company is delaying the broad launch of the product until January.

Early structure: Monolithic

- Traditionally, OS's (like UNIX) were built as a monolithic entity:



Monolithic design

- Major advantage:
  - cost of module interactions is low (procedure call)
- Disadvantages:
  - hard to understand
  - hard to modify
  - unreliable (no isolation between system modules)
  - hard to maintain
- What is the alternative?
  - find a way to organize the OS in order to simplify its design and implementation

Layering

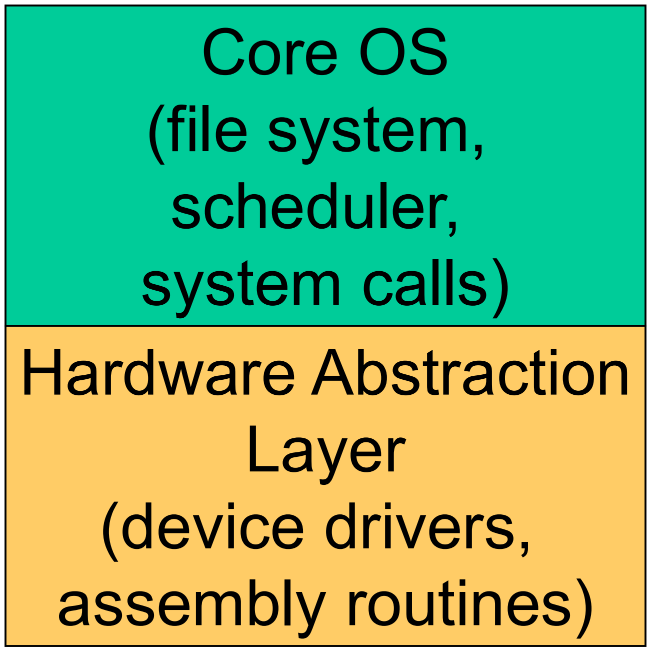
- The traditional approach is layering
  - implement OS as a set of layers
  - each layer presents an enhanced ‘virtual machine’ to the layer above
- The first description of this approach was Dijkstra’s THE system
  - Layer 5: Job Managers
    - Execute users’ programs
  - Layer 4: Device Managers
    - Handle devices and provide buffering
  - Layer 3: Console Manager
    - Implements virtual consoles
  - Layer 2: Page Manager
    - Implements virtual memories for each process
  - Layer 1: Kernel
    - Implements a virtual processor for each process
  - Layer 0: Hardware
- Each layer can be tested and verified independently

Problems with layering

- Imposes hierarchical structure
  - but real systems are more complex:
    - file system requires VM services (buffers)
    - VM would like to use files for its backing store
  - strict layering isn't flexible enough
- Poor performance
  - each layer crossing has overhead associated with it
- Disjunction between model and reality
  - systems modeled as layers, but not really built that way

Hardware Abstraction Layer

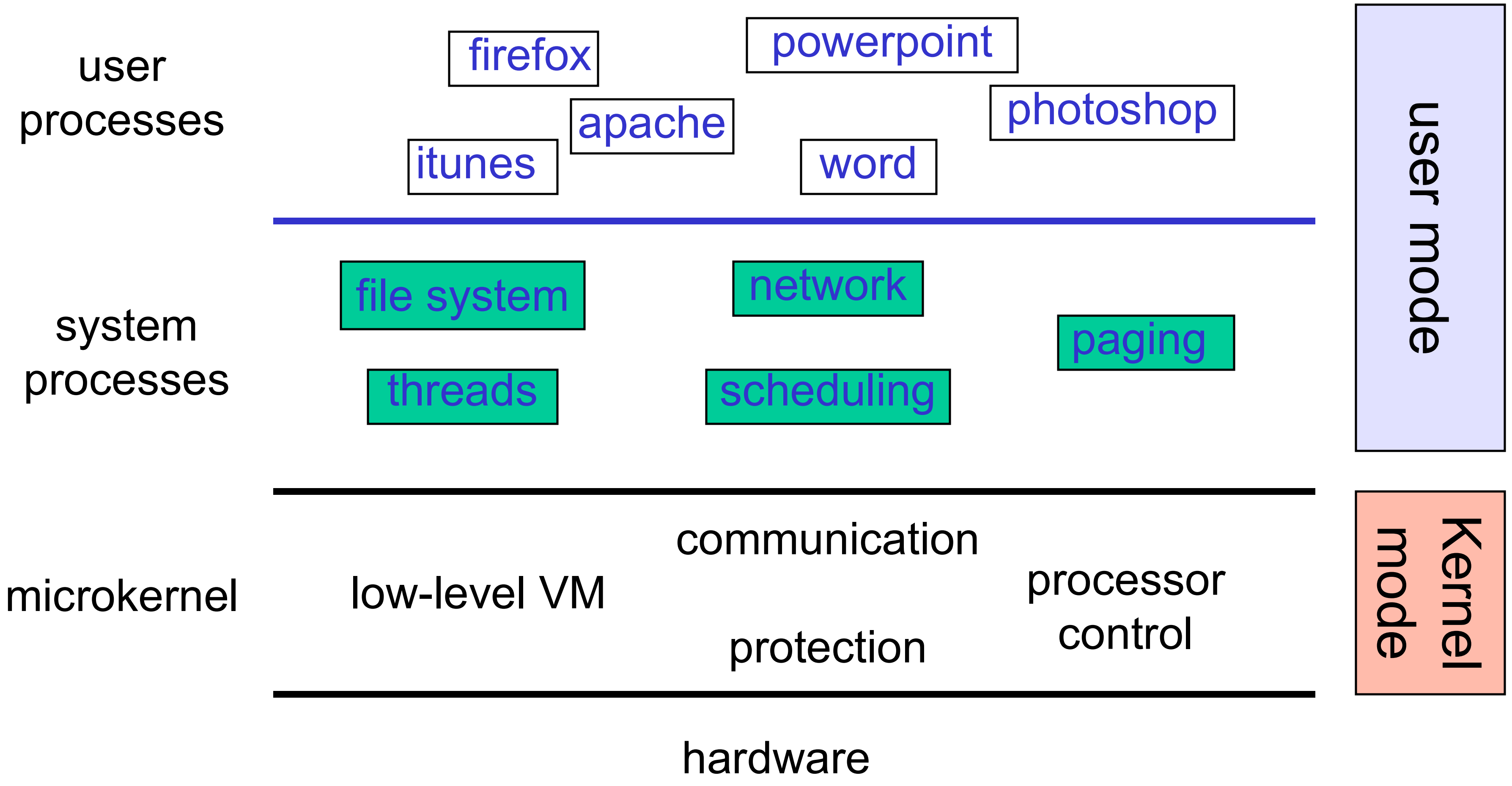
- An example of layering in modern operating systems
- Goal: separates hardware-specific routines from the “core” OS
  - Provides portability
  - Improves readability



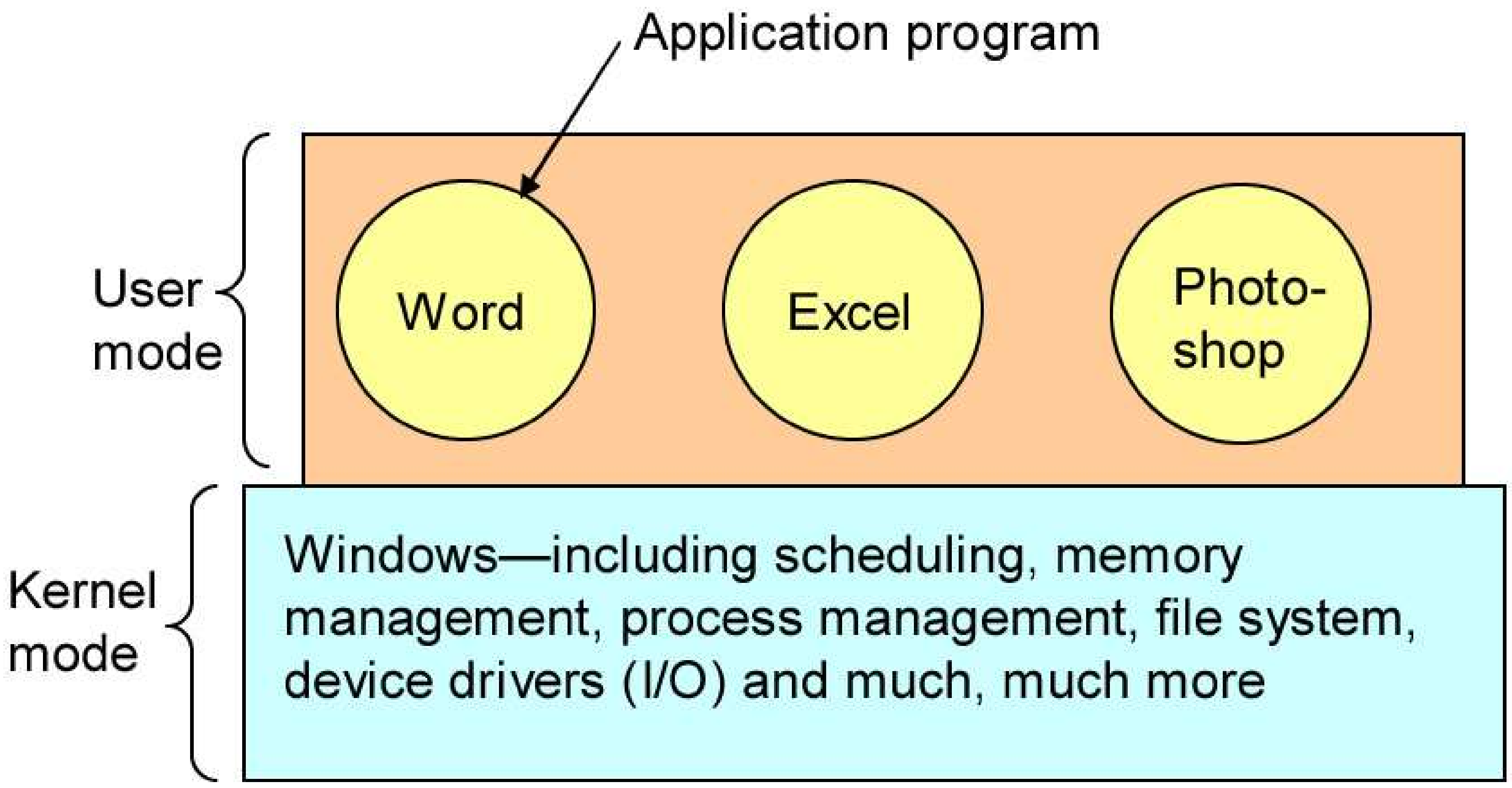
Microkernels

- Introduced in the late 80's, early 90's
  - recent resurgence of popularity
- Goal:
  - minimize what goes in kernel
  - organize rest of OS as user-level processes
- This results in:
  - better reliability (isolation between components)
  - ease of extension and customization
  - poor performance (user/kernel boundary crossings)
- First microkernel system was Hydra (CMU, 1970)
  - Follow-ons: Mach (CMU), Chorus (French UNIX-like OS), OS X (Apple)

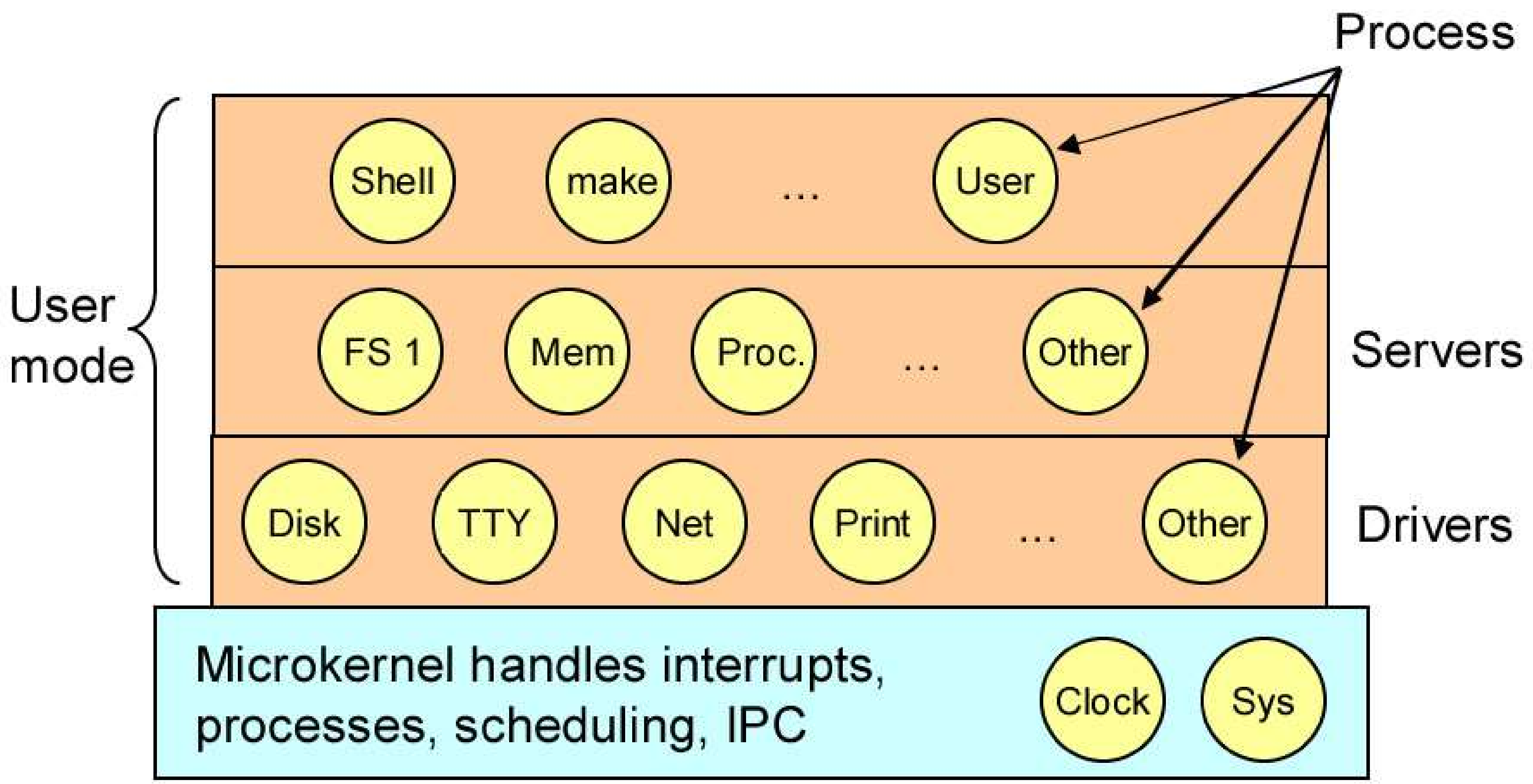
Microkernel structure illustrated



EXAMPLE: WINDOWS

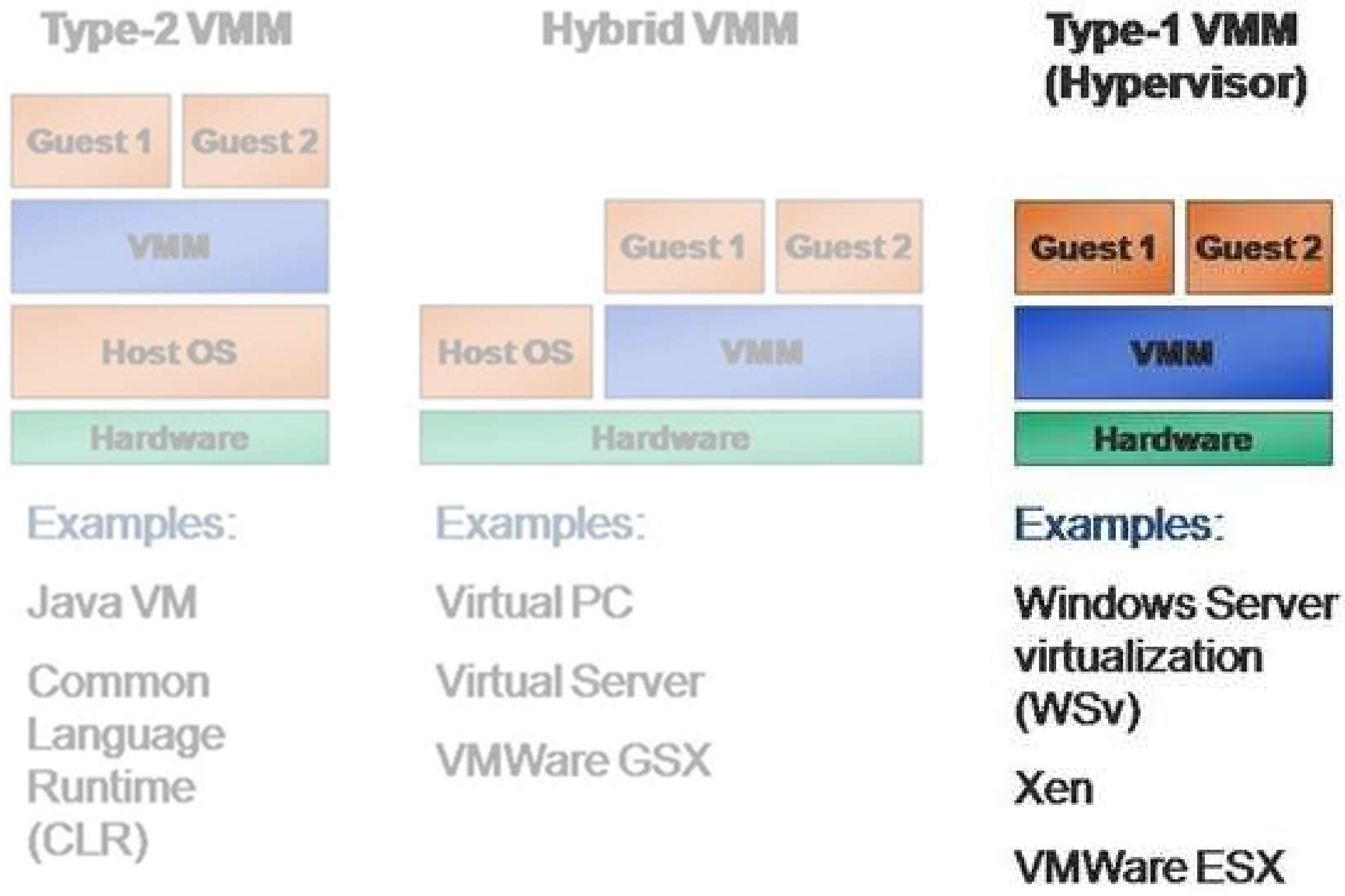

Application program
User mode
Word
Excel
Photo-shop
Kernel mode
Windows—including scheduling, memory management, process management, file system, device drivers (I/O) and much, much more

ARCHITECTURE OF MINIX 3



6

Virtual Machine Monitors



- Transparently implement “hardware” in software
- Voilà, you can boot a “guest OS”

Summary and Next Module

- Summary
  - OS design has been an evolutionary process of trial and error. Probably more error than success
  - Successful OS designs have run the spectrum from monolithic, to layered, to micro kernels, to virtual machine monitors
  - The role and design of an OS are still evolving
  - It is impossible to pick one “correct” way to structure an OS
- Next module
  - Processes, one of the most fundamental pieces in an OS
  - What is a process, what does it do, and how does it do it