Day after SNOWDAY 2020!

Chapter 2 and SEEsh!

Double pointers?!?!

See Iain's post!
- Questions on tutorial

C is pass by value…

Administration

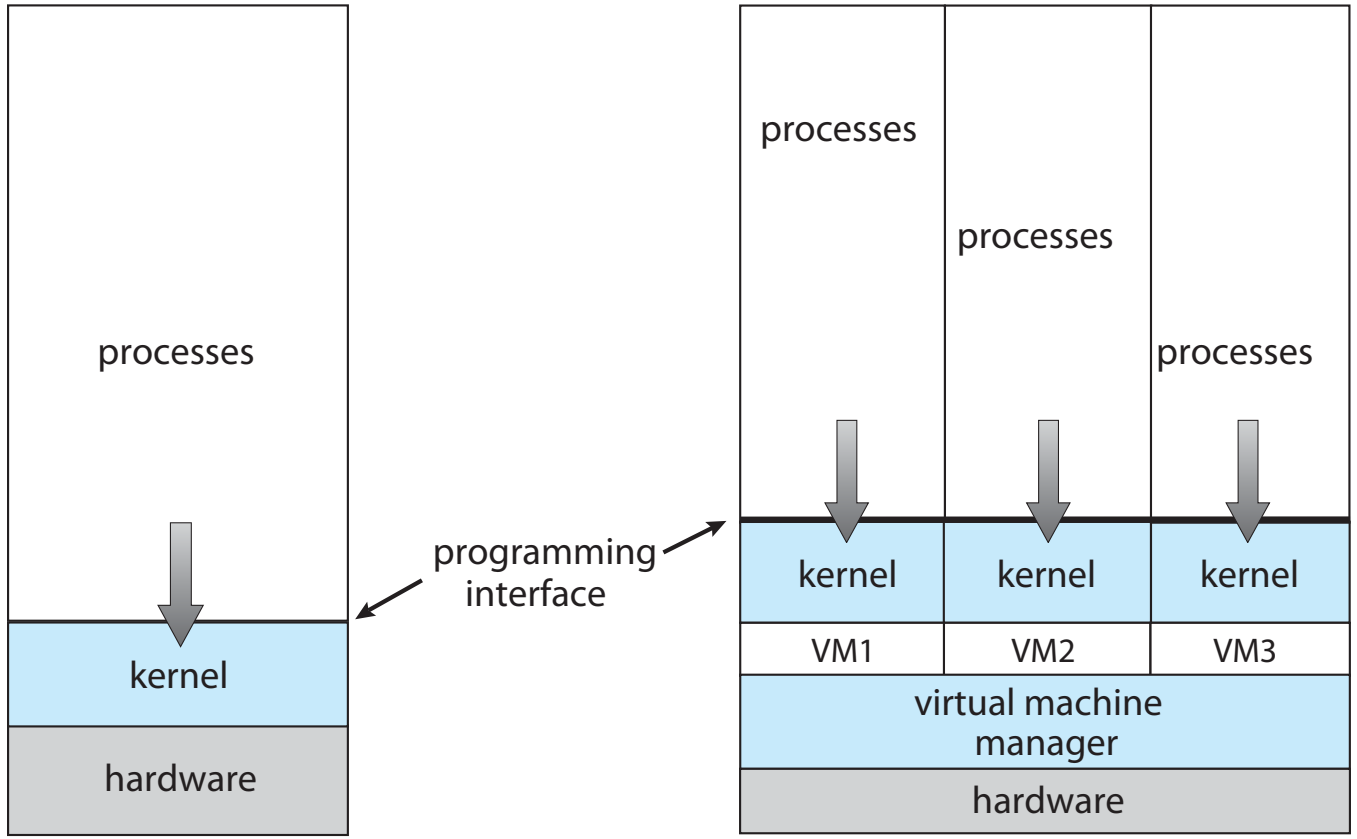
- Great stuff on the forums!
  - Several people are noticing their code is “freezing” or ”pausing”?!
- Posted info on GDB
- Posted material from Tutorials
  - Iain video, Iain and Jon on forum
- Posted Assignment 1 Part 2: SEEsh!
  - Due Jan 31
- Extended Assignment 1 Part 1
  - ALSO now due Jan 31, **with part 2**!

SEEsh

- you will find LOADS AND LOADS of examples out there!

- One nice step by step tutorial is by Stephan Brennan
  - https://brennan.io/2015/01/16/write-a-shell-in-c/

Computing Environments - Virtualization

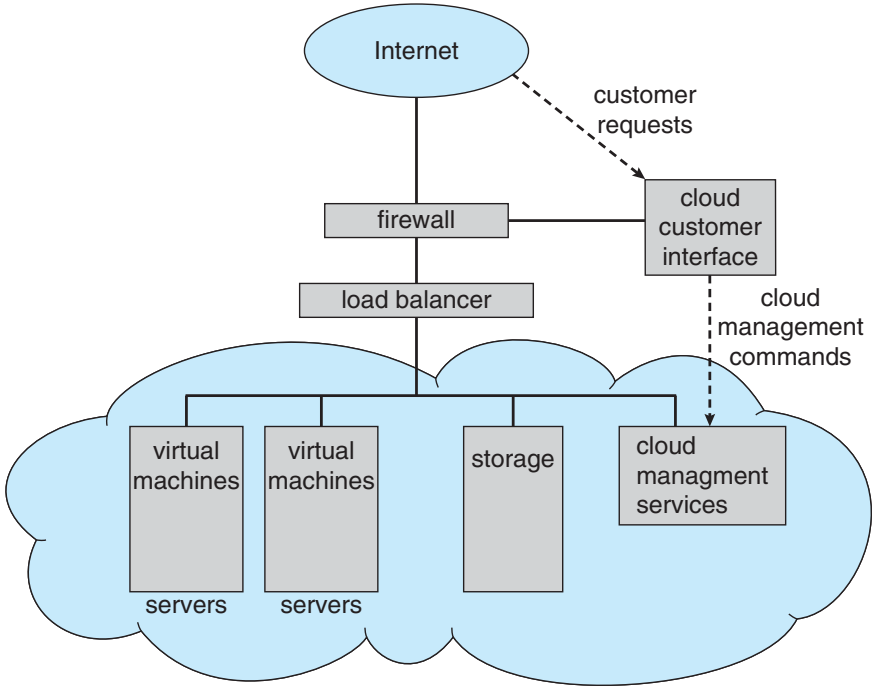


Computing Environments – Cloud Computing

- Delivers computing, storage, even apps as a service across a network
- Logical extension of virtualization because it uses virtualization as the base for it functionality.
  - Amazon **EC2** has thousands of servers, millions of virtual machines, petabytes of storage available across the Internet, pay based on usage
- Many types
  - **Public cloud** – available via Internet to anyone willing to pay
  - **Private cloud** – run by a company for the company's own use
  - **Hybrid cloud** – includes both public and private cloud components

Computing Environments – Cloud Computing

- Cloud computing environments composed of traditional OSes, plus VMMs, plus cloud management tools
  - Internet connectivity requires security like firewalls
  - Load balancers spread traffic across multiple applications



Computing Environments – Real-Time Embedded Systems

- Real-time embedded systems most prevalent form of computers
  - special purpose, limited purpose OS, **real-time OS**
  - Use expanding
- Many other special computing environments as well
  - Some have OSes, some perform tasks without an OS
- Real-time OS has well-defined fixed time constraints
  - Processing ***must*** be done within constraint
  - Correct operation only if constraints met
- TAKE 460 NEXT SPRING!!!!! ☺

Bourne Shell Command Interpreter

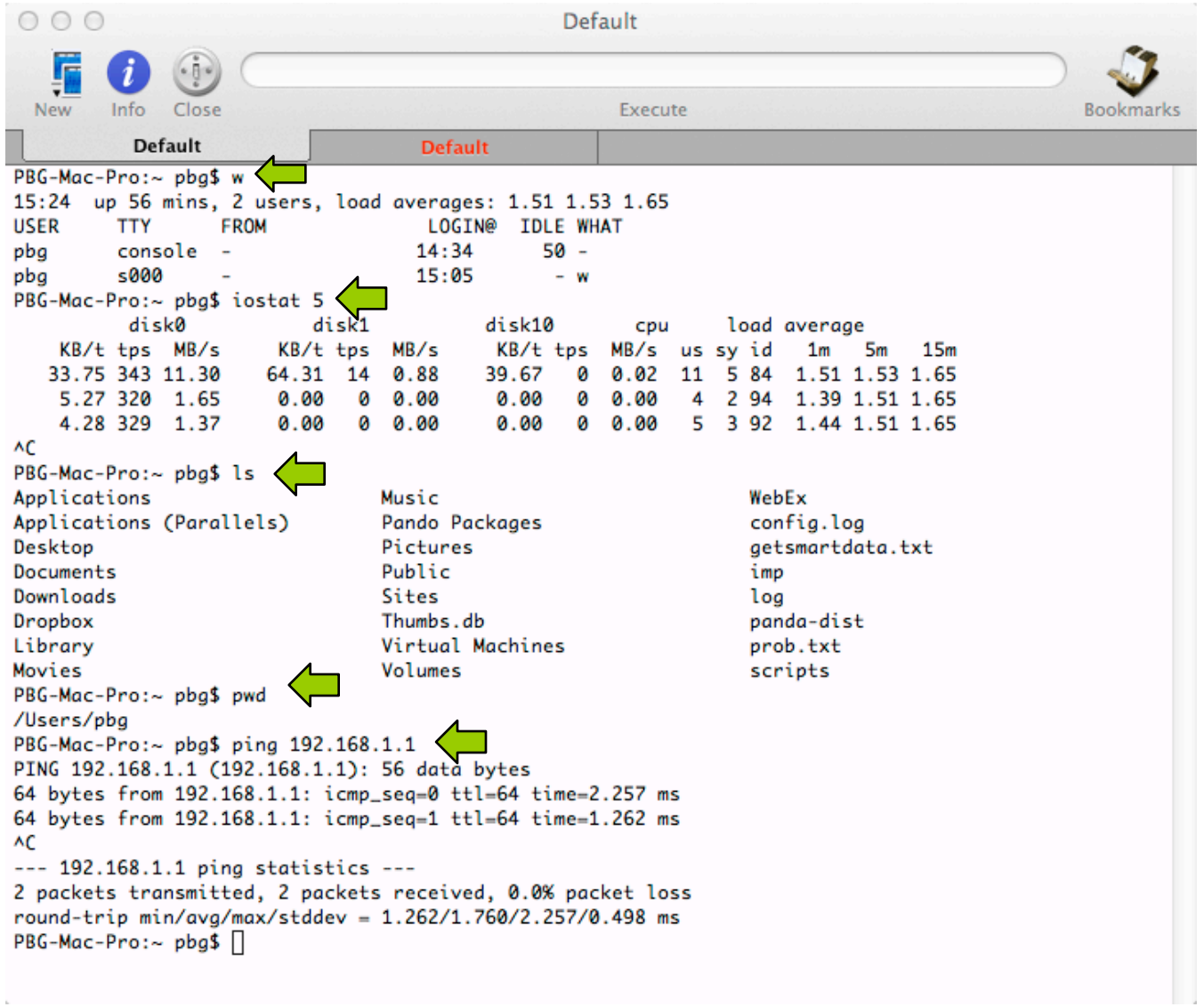
Default

New Info Close Execute Bookmarks

Default Default

```
PBG-Mac-Pro:~ pbg$ w
15:24  up 56 mins, 2 users, load averages: 1.51 1.53 1.65
USER     TTY      FROM              LOGIN@  IDLE WHAT
pbg      console  -                 14:34      50 -
pbg      s000     -                 15:05       - w
PBG-Mac-Pro:~ pbg$ iostat 5
          disk0           disk1           disk10       cpu     load average
    KB/t tps  MB/s     KB/t tps  MB/s     KB/t tps  MB/s  us sy id   1m   5m   15m
   33.75 343 11.30    64.31  14  0.88    39.67   0  0.02  11  5 84  1.51 1.53 1.65
    5.27 320  1.65     0.00   0  0.00     0.00   0  0.00   4  2 94  1.39 1.51 1.65
    4.28 329  1.37     0.00   0  0.00     0.00   0  0.00   5  3 92  1.44 1.51 1.65
^C
PBG-Mac-Pro:~ pbg$ ls
Applications                    Music                           WebEx
Applications (Parallels)        Pando Packages                  config.log
Desktop                         Pictures                        getsmartdata.txt
Documents                       Public                          imp
Downloads                       Sites                           log
Dropbox                         Thumbs.db                       panda-dist
Library                         Virtual Machines                prob.txt
Movies                          Volumes                         scripts
PBG-Mac-Pro:~ pbg$ pwd
/Users/pbg
PBG-Mac-Pro:~ pbg$ ping 192.168.1.1
PING 192.168.1.1 (192.168.1.1): 56 data bytes
64 bytes from 192.168.1.1: icmp_seq=0 ttl=64 time=2.257 ms
64 bytes from 192.168.1.1: icmp_seq=1 ttl=64 time=1.262 ms
^C
--- 192.168.1.1 ping statistics ---
2 packets transmitted, 2 packets received, 0.0% packet loss
round-trip min/avg/max/stddev = 1.262/1.760/2.257/0.498 ms
PBG-Mac-Pro:~ pbg$
```

Bourne Shell Command Interpreter



Operating System Design and Implementation

- Design and Implementation of OS not “solvable”, but some approaches have proven successful
- Internal structure of different Operating Systems can vary widely
- Start the design by defining goals and specifications
- Affected by choice of hardware, type of system
- **User** goals and **System** goals
  - User goals – operating system should be convenient to use, easy to learn, reliable, safe, and fast
  - System goals – operating system should be easy to design, implement, and maintain, as well as flexible, reliable, error-free, and efficient

Operating System Design and Implementation (Cont.)

- Important principle to separate

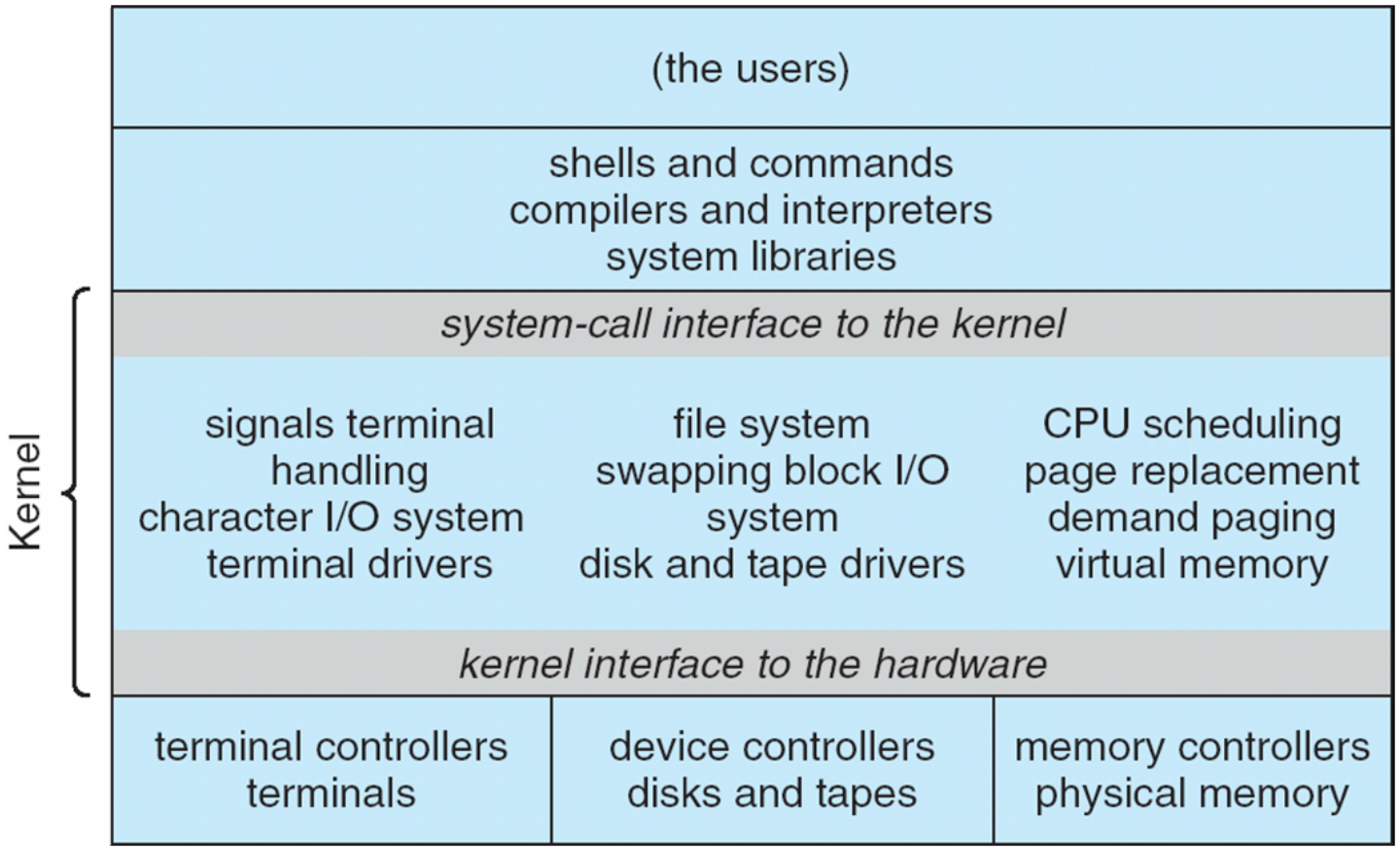
  Policy: ***What*** will be done?
  Mechanism: ***How*** to do it?

- Mechanisms determine how to do something, policies decide what will be done

- The separation of policy from mechanism is a very important principle, it allows maximum flexibility if policy decisions are to be changed later (example – timer)

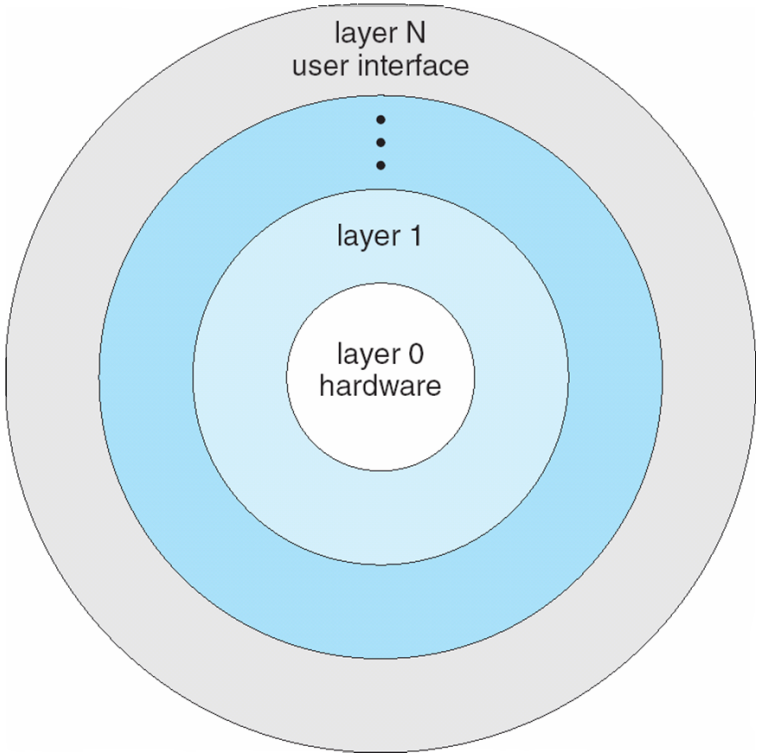
Traditional UNIX System Structure

Beyond simple but not fully layered



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



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

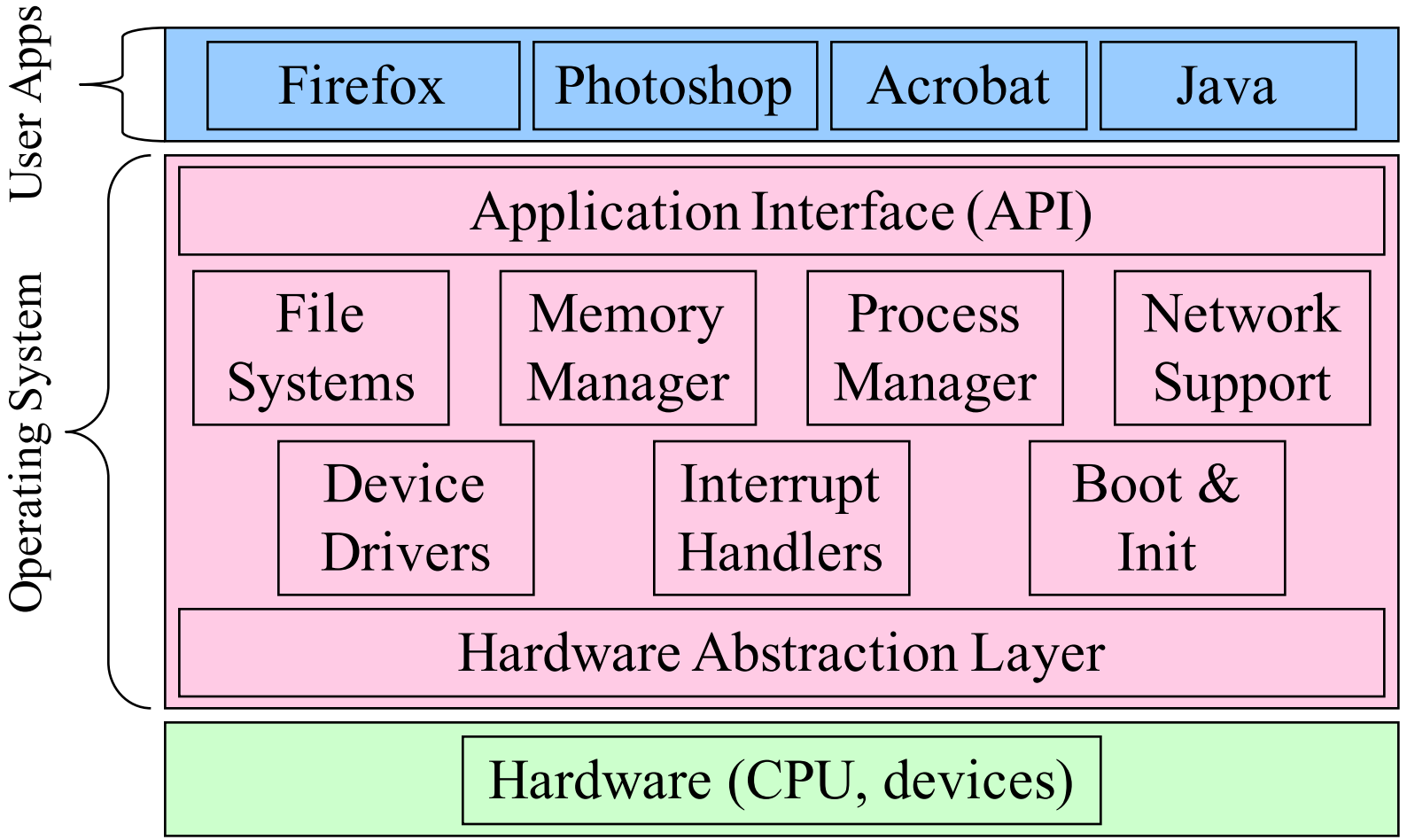
Operating-System Debugging

- **Debugging** is finding and fixing errors, or **bugs**
- OS generate **log files** containing error information
- Failure of an application can generate **core dump** file capturing memory of the process
- Operating system failure can generate **crash dump** file containing kernel memory
- Beyond crashes, performance tuning can optimize system performance
  - Sometimes using ***trace listings*** of activities, recorded for analysis
  - **Profiling** is periodic sampling of instruction pointer to look for statistical trends

Kernighan's Law: "*Debugging is twice as hard as writing the code in the first place. Therefore, if you write the code as cleverly as possible, you are, by definition, not smart enough to debug it.*"

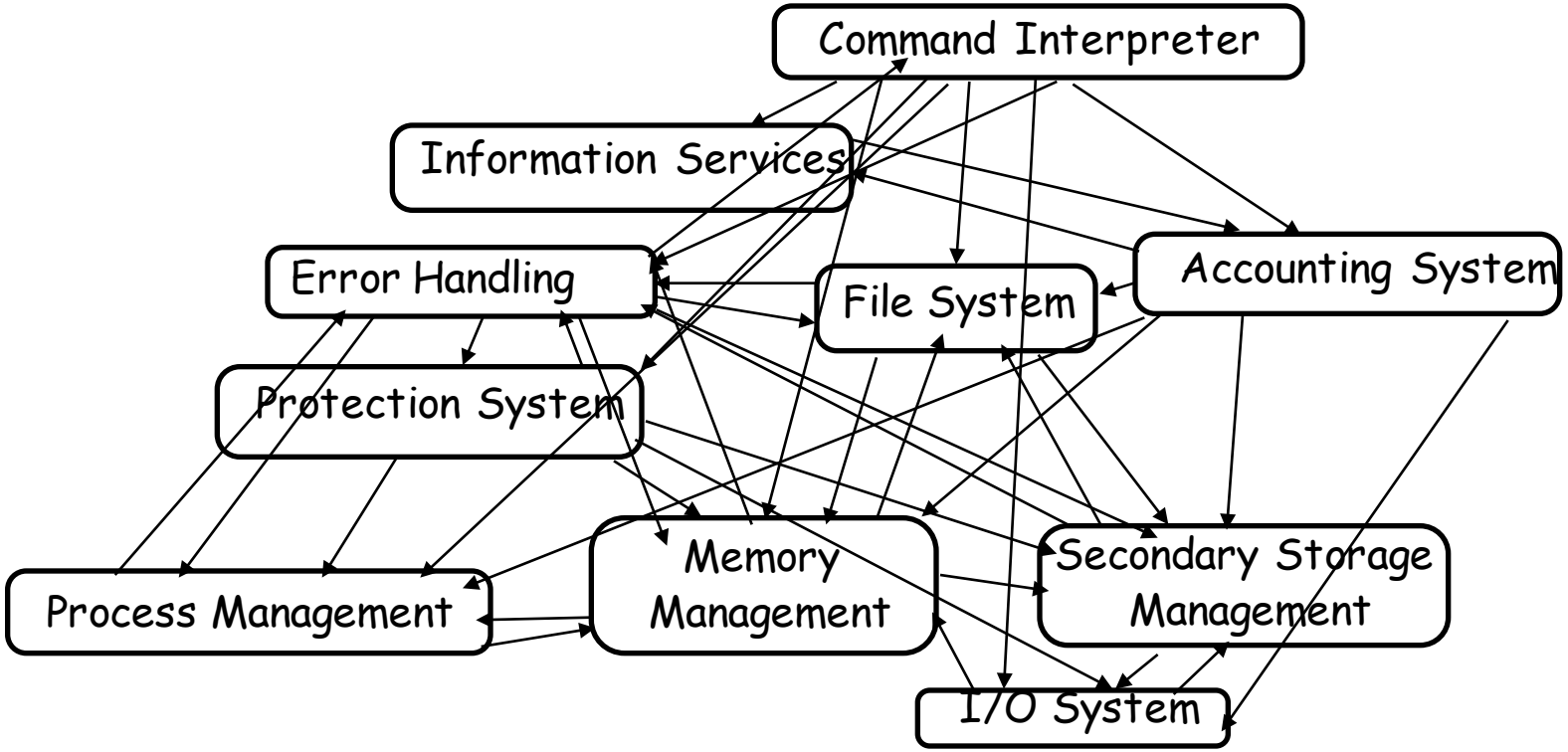
System Boot

- When power initialized on system, execution starts at a fixed memory location
  - Firmware ROM used to hold initial boot code
- Operating system must be made available to hardware so hardware can start it
  - Small piece of code – **bootstrap loader**, stored in **ROM** or **EEPROM** locates the kernel, loads it into memory, and starts it
  - Sometimes two-step process where **boot block** at fixed location loaded by ROM code, which loads bootstrap loader from disk
- Kernel loads and system is then **running**

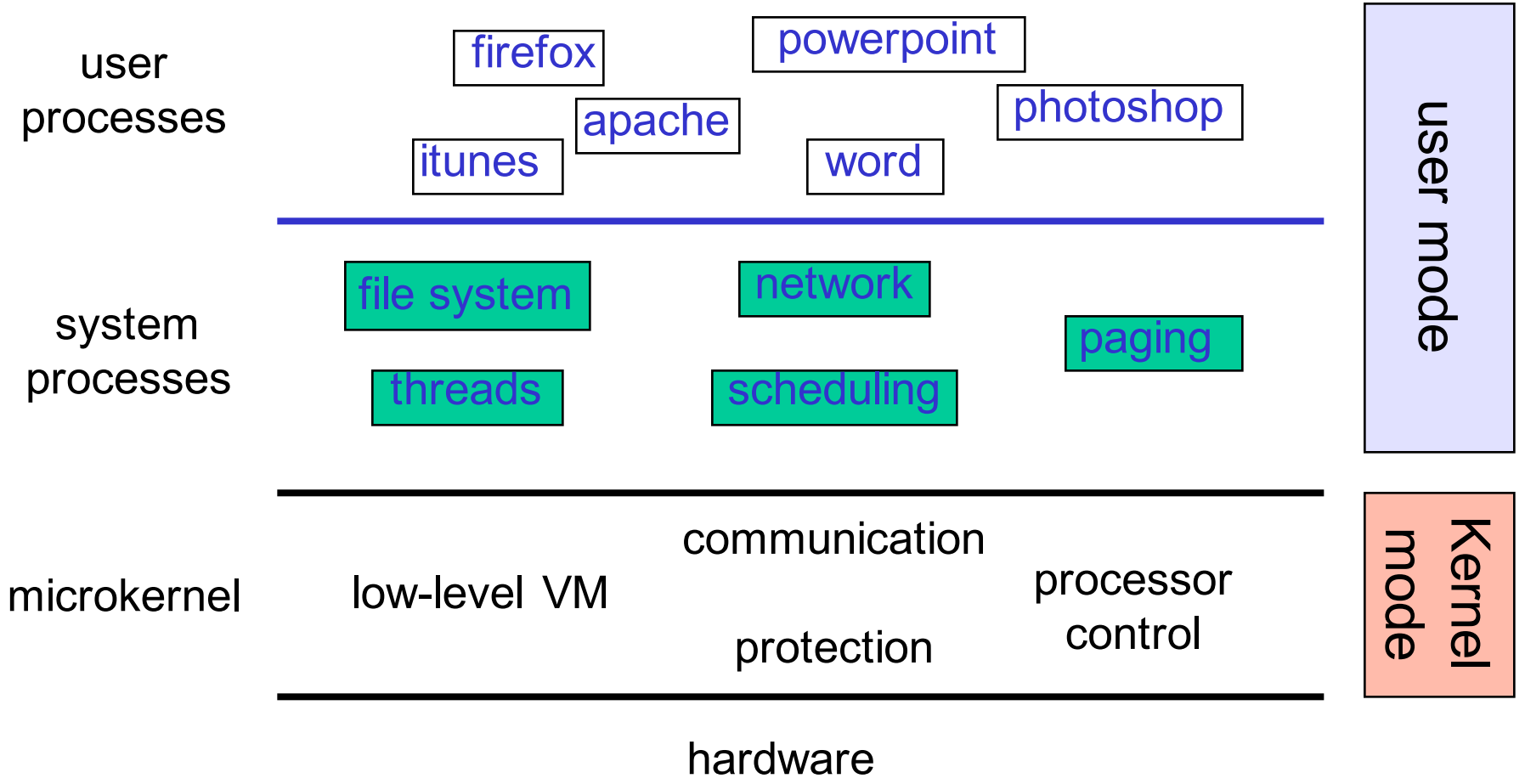

User Apps
Operating System
Firefox
Photoshop
Acrobat
Java
Application Interface (API)
File Systems
Memory Manager
Process Manager
Network Support
Device Drivers
Interrupt Handlers
Boot & Init
Hardware Abstraction Layer
Hardware (CPU, devices)

Major OS components

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


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

Microkernel structure illustrated

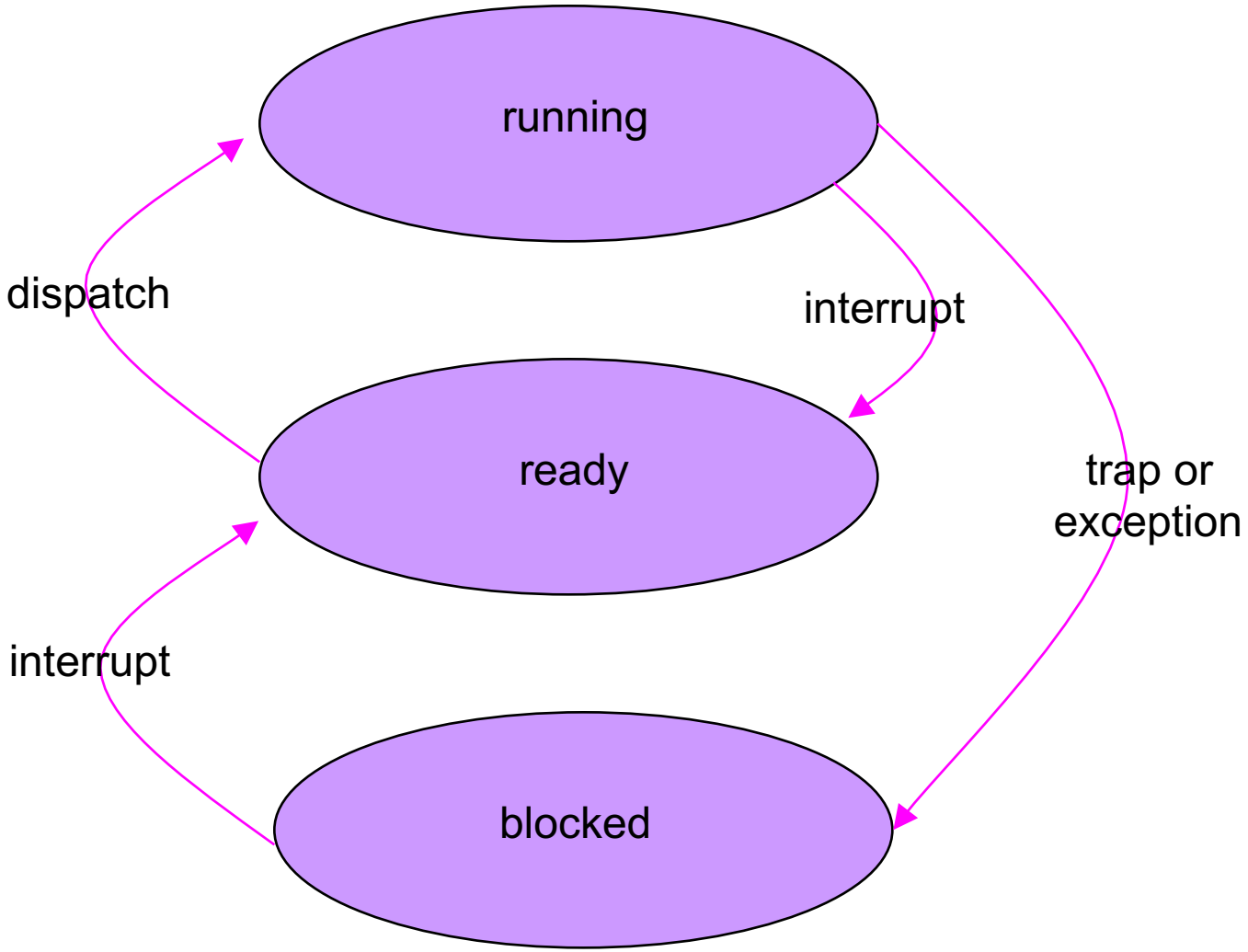


21

Process management

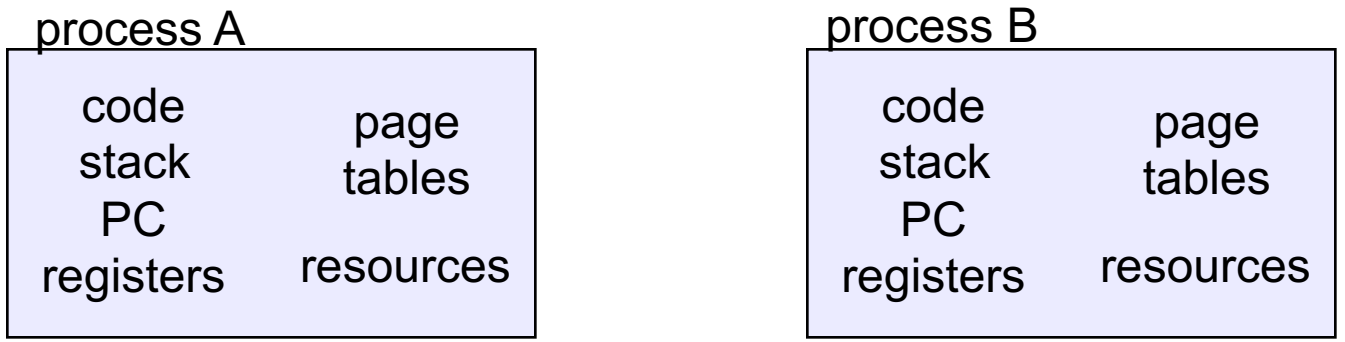
- An OS executes many kinds of activities:
  - users' programs
  - batch jobs or scripts
  - system programs
    - print spoolers, name servers, file servers, network daemons, …
- Each of these activities is encapsulated in a process
  - a process includes the execution context
    - PC, registers, VM, 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, …

States of a user process



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



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

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

- *SEEsh*!