COS 318: Operating Systems

Overview

Kai Li
Computer Science Department
Princeton University

(http://www.cs.princeton.edu/courses/cos318/)

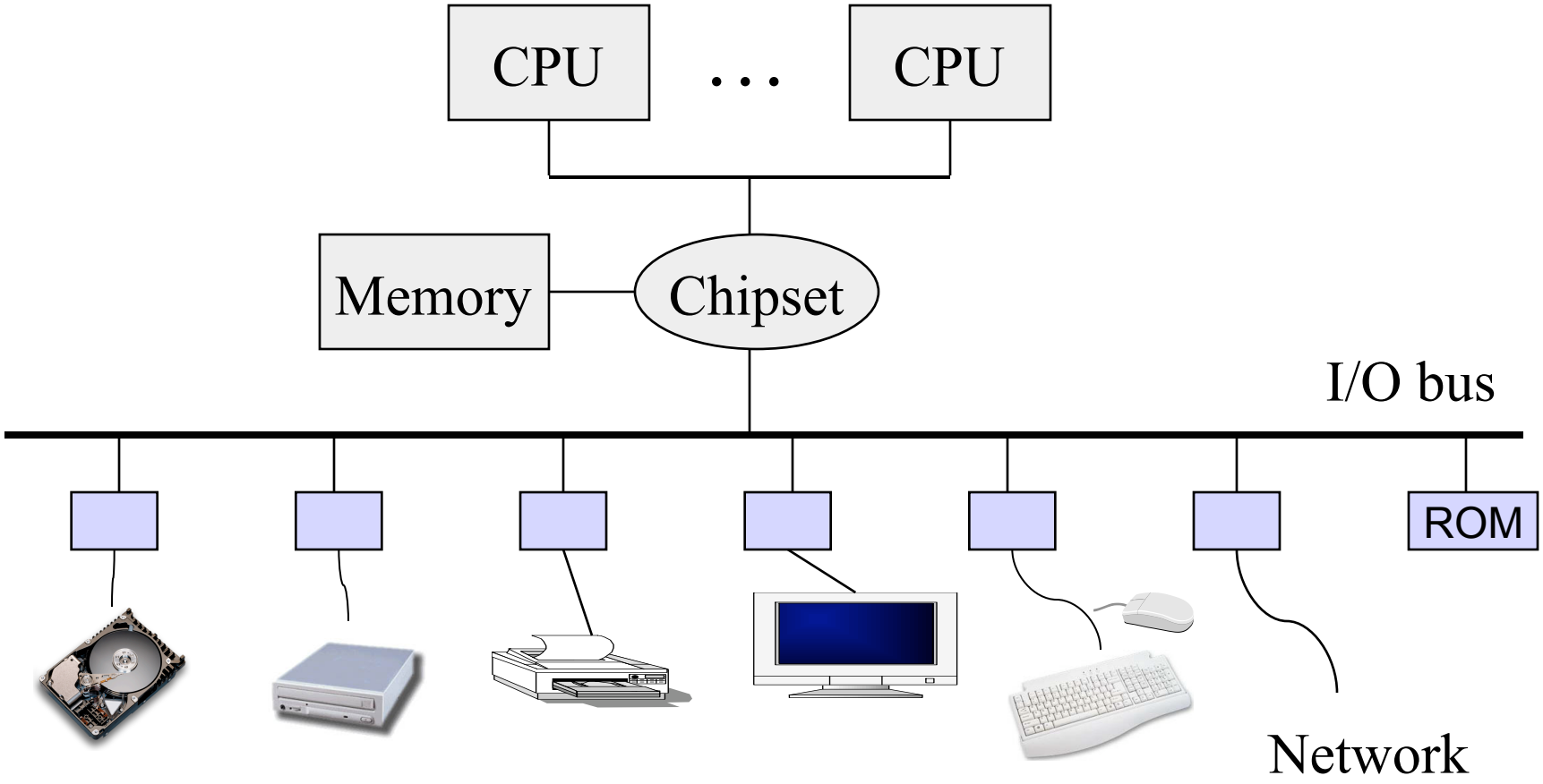
Logistics

- Precepts:
  - Tue, Wed: TBD, 105 CS building
- Design review:
  - 9/28 during 6-10pm, 010 Friends center
- Project 1 due:
  - 10/5 at 11:59pm
- Reminder:
  - Subscribe to the cos318 mailing list today!

Today

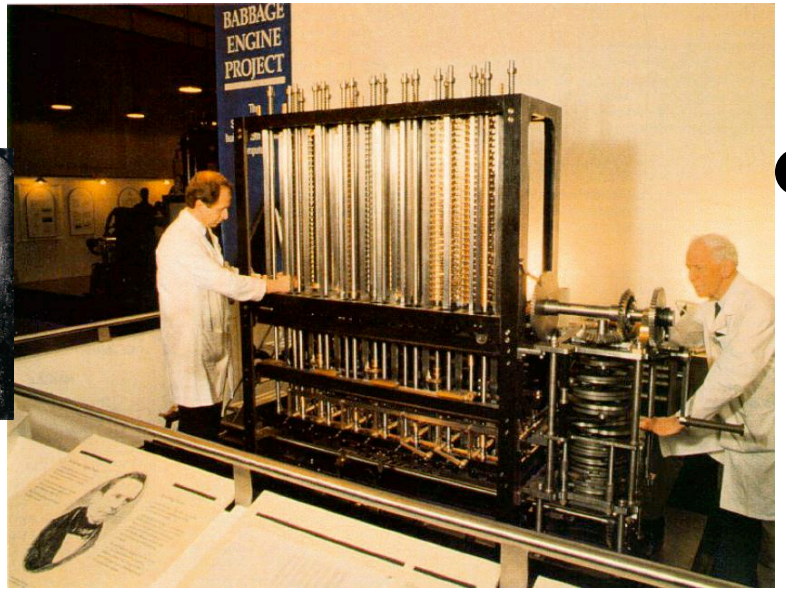
- Overview of OS structure
- Overview of OS components

Hardware of A Typical Computer

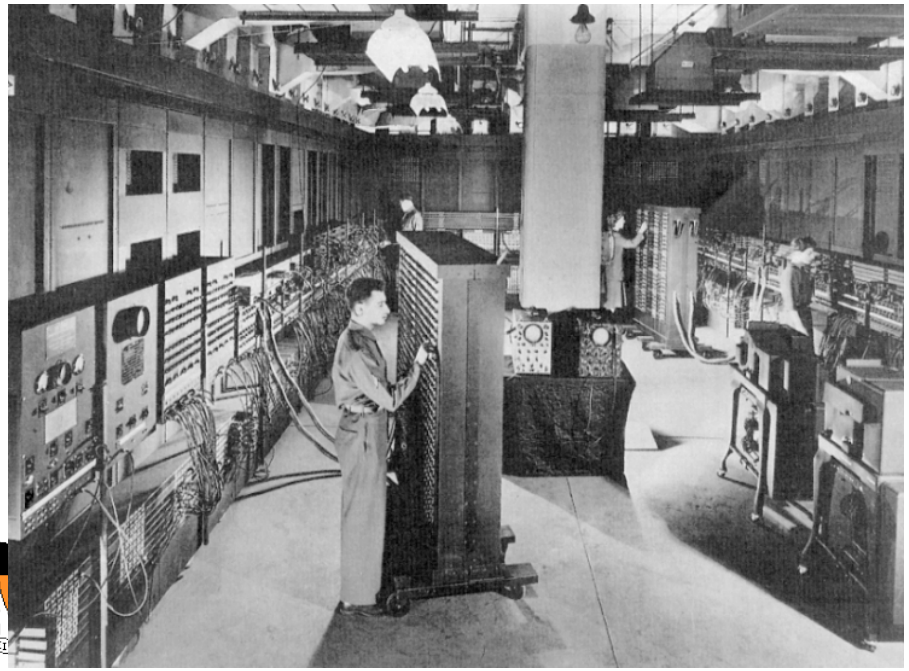


Computing machinery

Analytical Engine (~1850) Charles Babbage

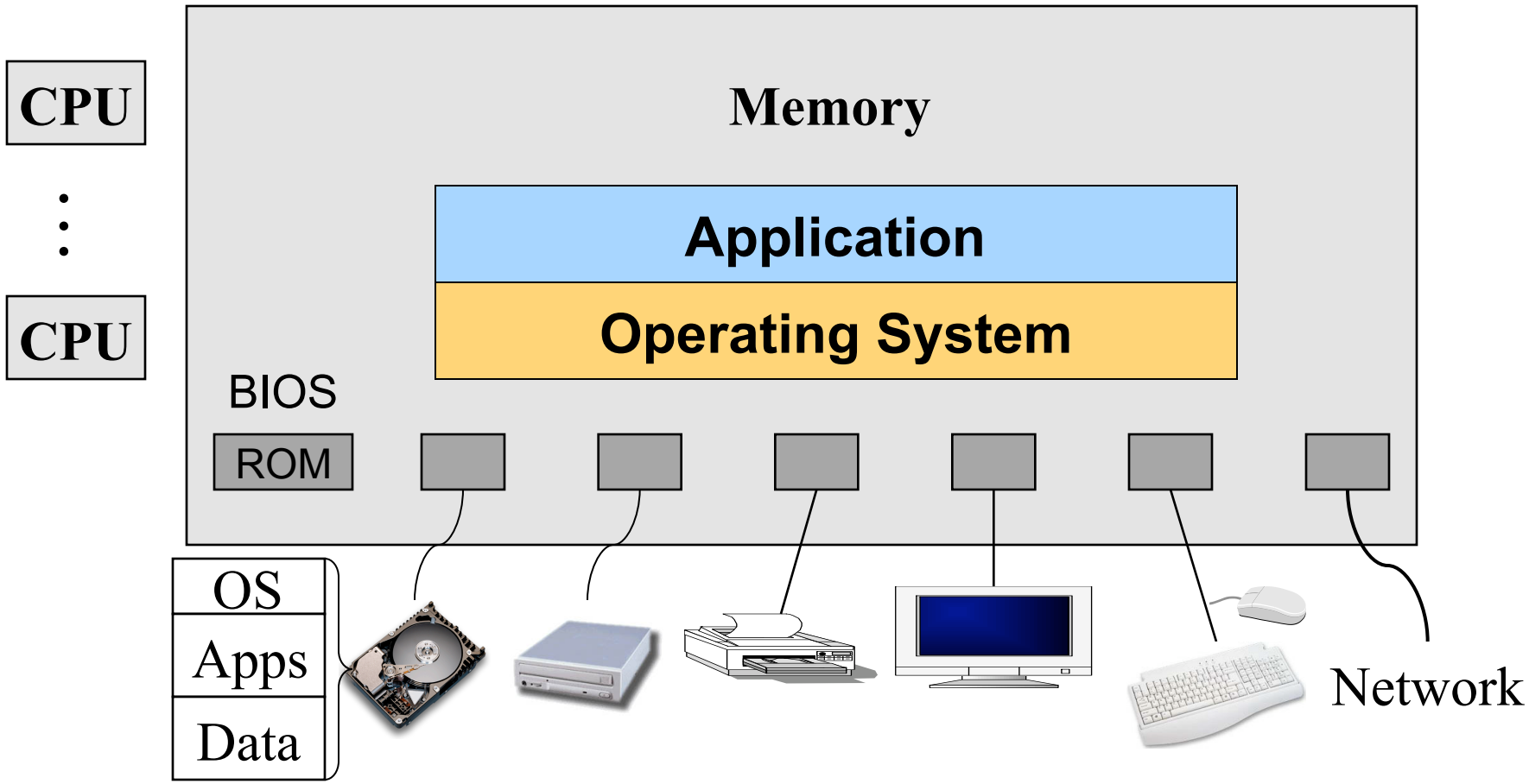


ENIAC (~1946) Eckert & Mauchly, UPenn



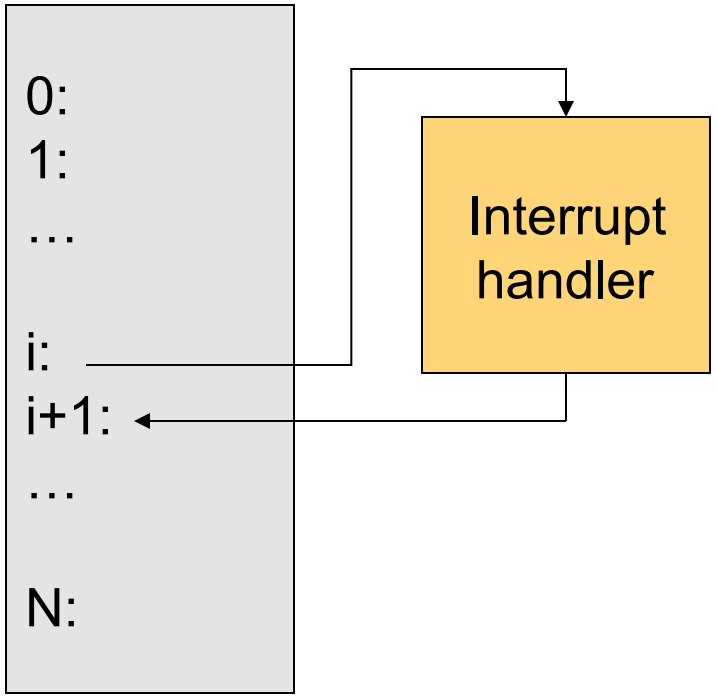
Johnniac (~1953) von Neumann, IAS

A Typical Computer System

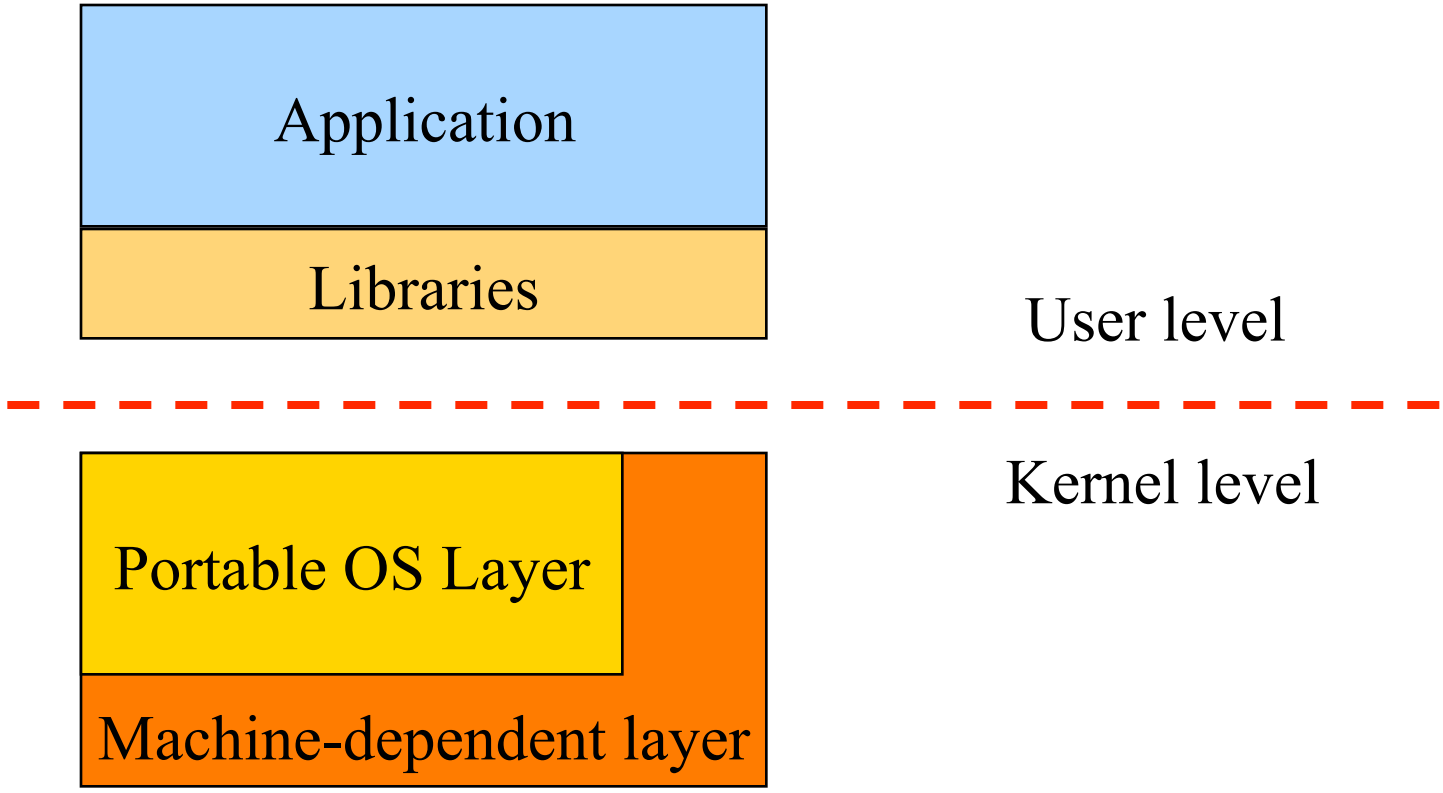


Interrupts

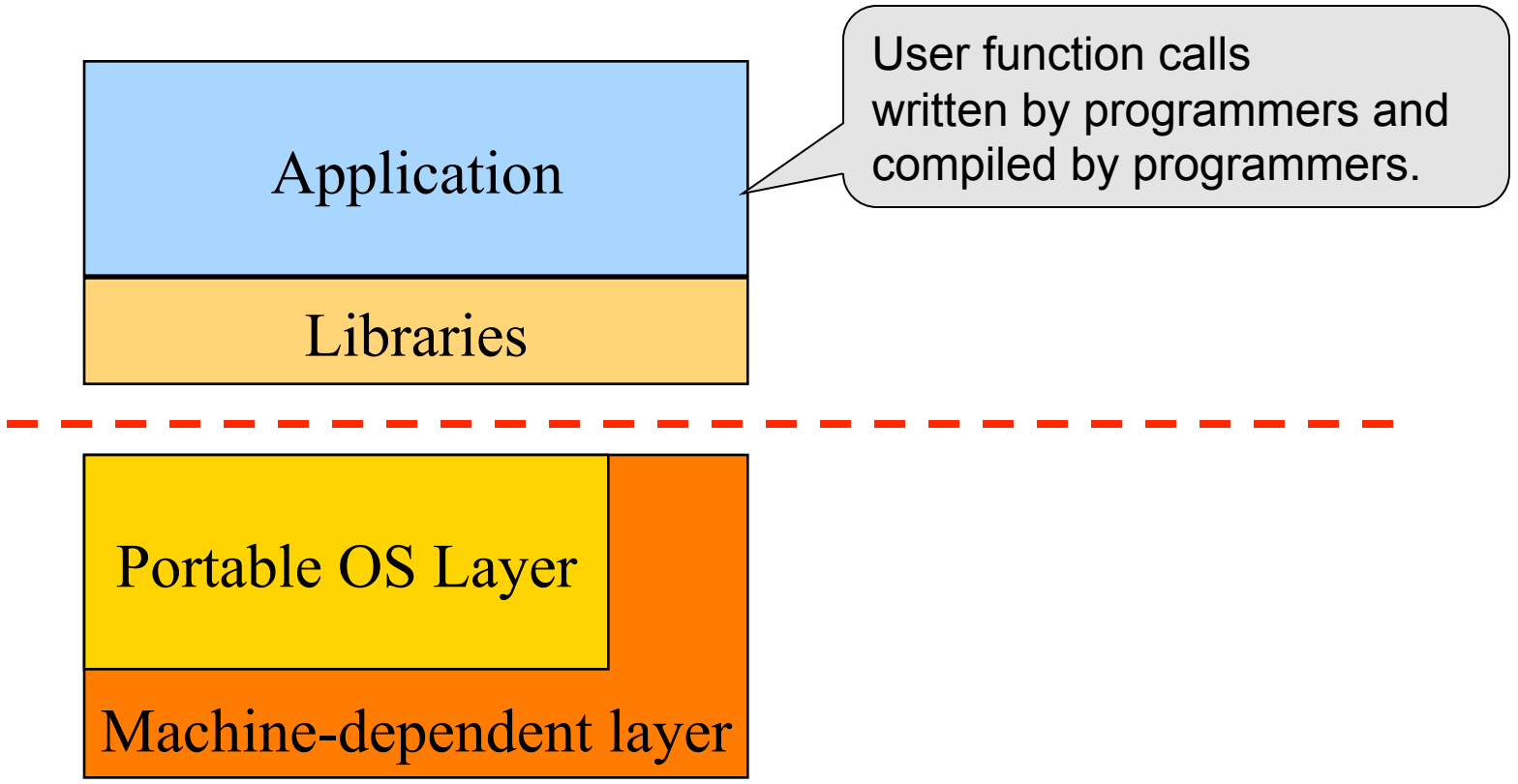
- Raised by external events
- Interrupt handler is in the kernel
  - Switch to another process
  - Overlap I/O with CPU
  - …
- Eventually resume the interrupted process



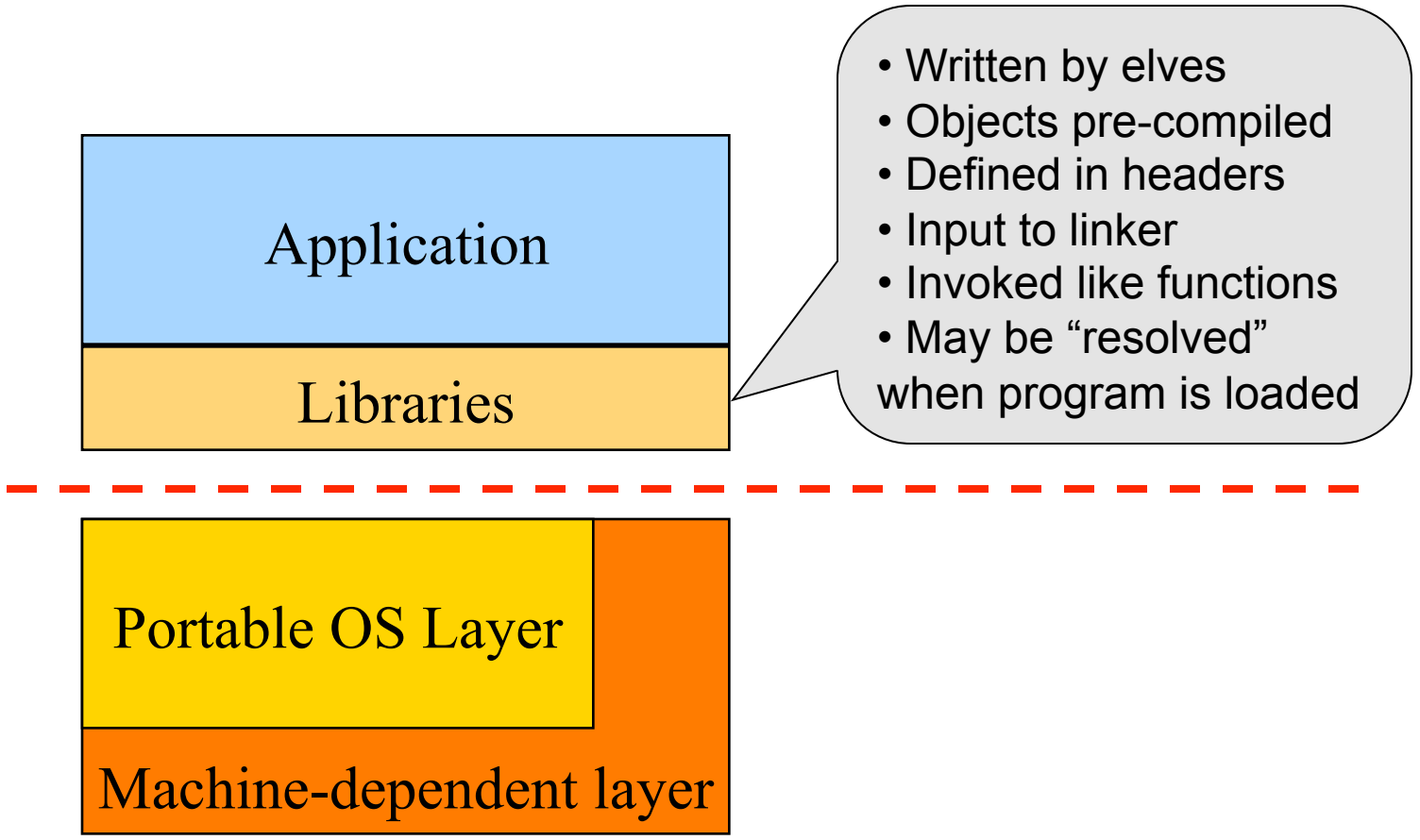
Typical Unix OS Structure



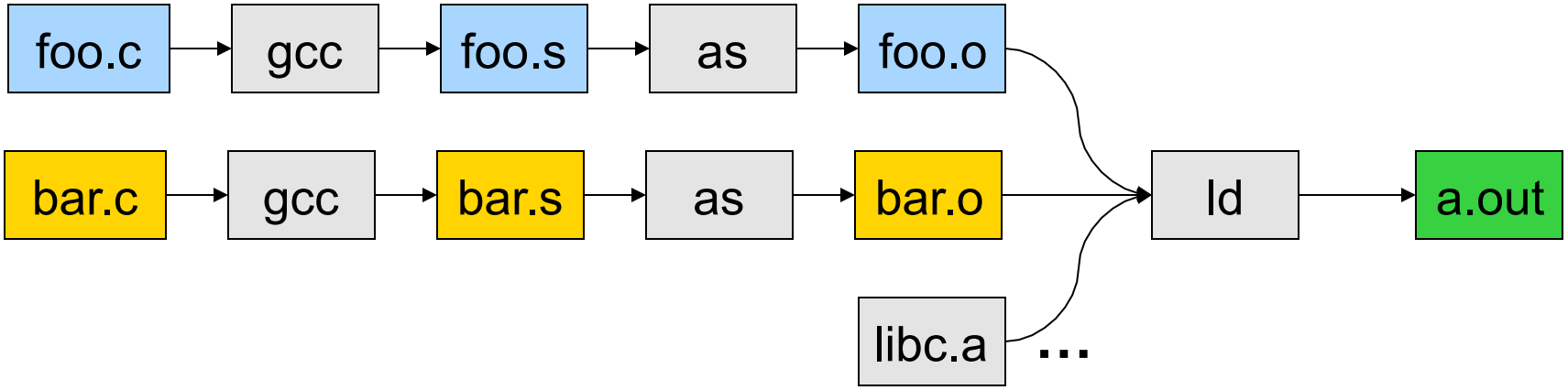
Typical Unix OS Structure



Typical Unix OS Structure



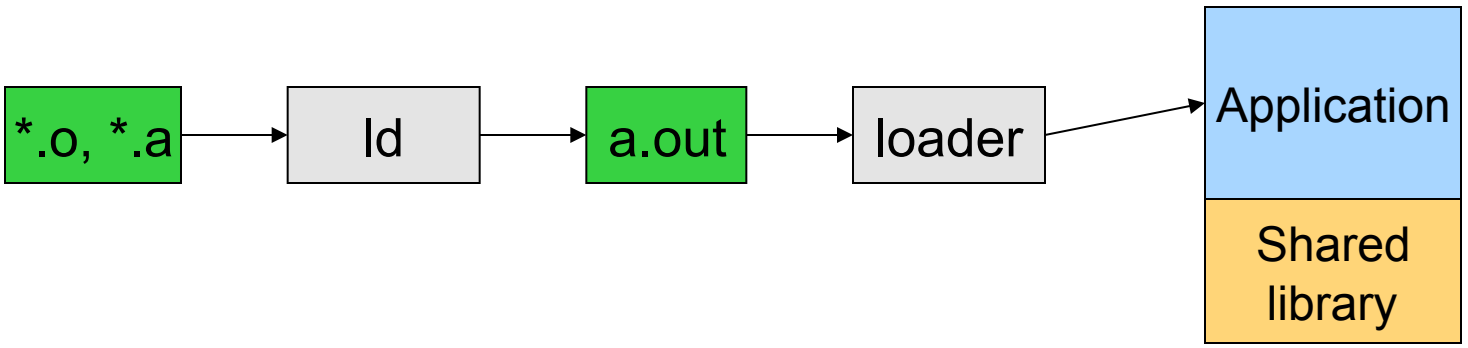
Pipeline of Creating An Executable File



- gcc can compile, assemble, and link together
- Compiler (part of gcc) compiles a program into assembly
- Assembler compiles assembly code into relocatable object file
- Linker links object files into an executable
- For more information:
  - Read man page of a.out, elf, ld, and nm
  - Read the document of ELF

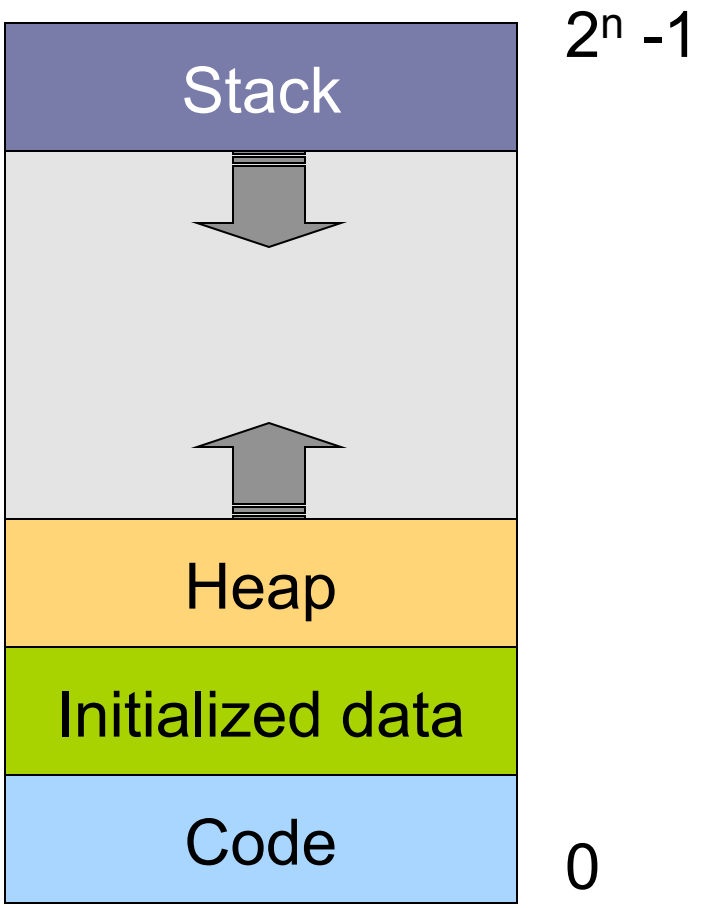
Execution (Run An Application)

- On Unix, “loader” does the job
  - Read an executable file
  - Layout the code, data, heap and stack
  - Dynamically link to shared libraries
  - Prepare for the OS kernel to run the application



What's An Application?

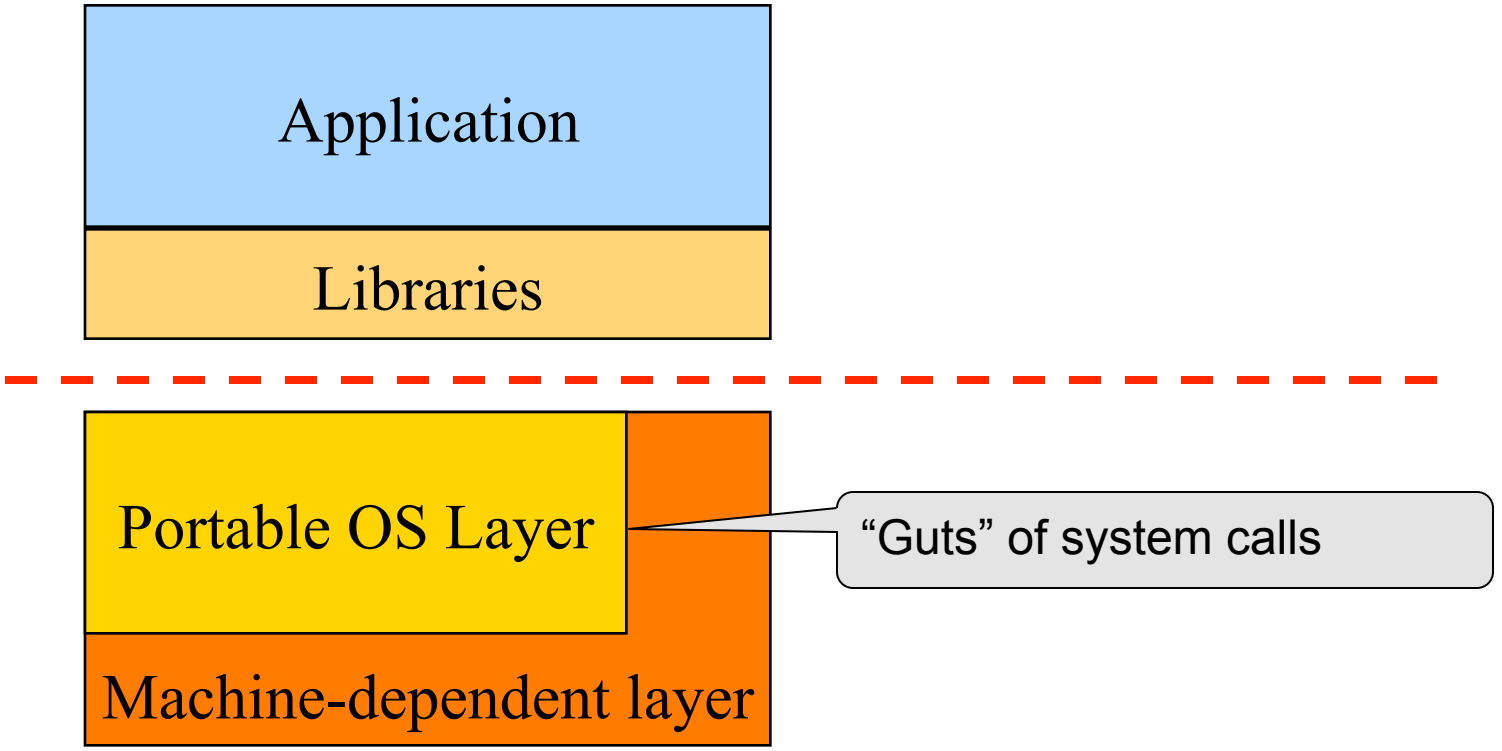
- Four segments
  - Code/Text – instructions
  - Data – initialized global variables
  - Stack
  - Heap
- Why?
  - Separate code and data
  - Stack and heap go towards each other



Responsibilities

- Stack
  - Layout by compiler
  - Allocate/deallocate by process creation (fork) and termination
  - Names are relative off of stack pointer and entirely local
- Heap
  - Linker and loader say the starting address
  - Allocate/deallocate by library calls such as malloc() and free()
  - Application program use the library calls to manage
- Global data/code
  - Compiler allocate statically
  - Compiler emit names and symbolic references
  - Linker translate references and relocate addresses
  - Loader finally lay them out in memory

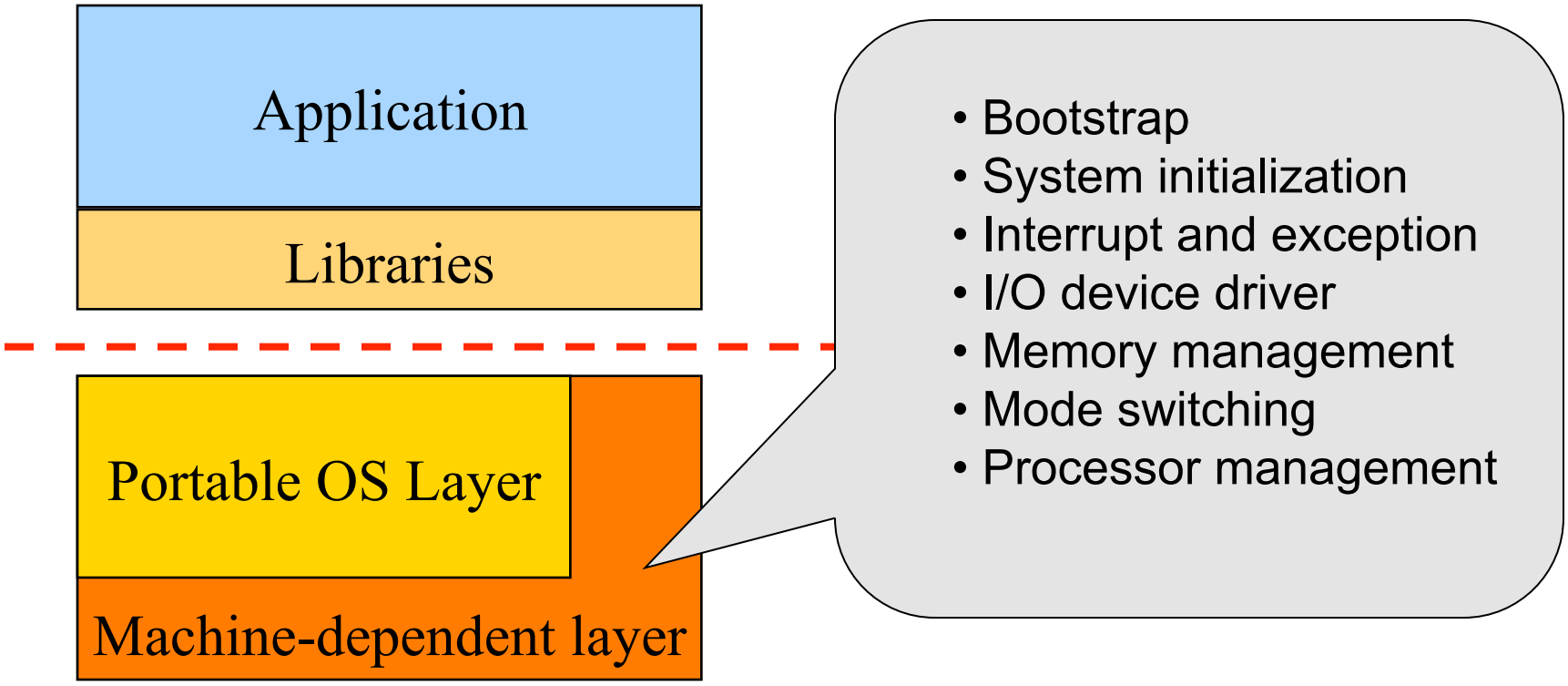
Typical Unix OS Structure



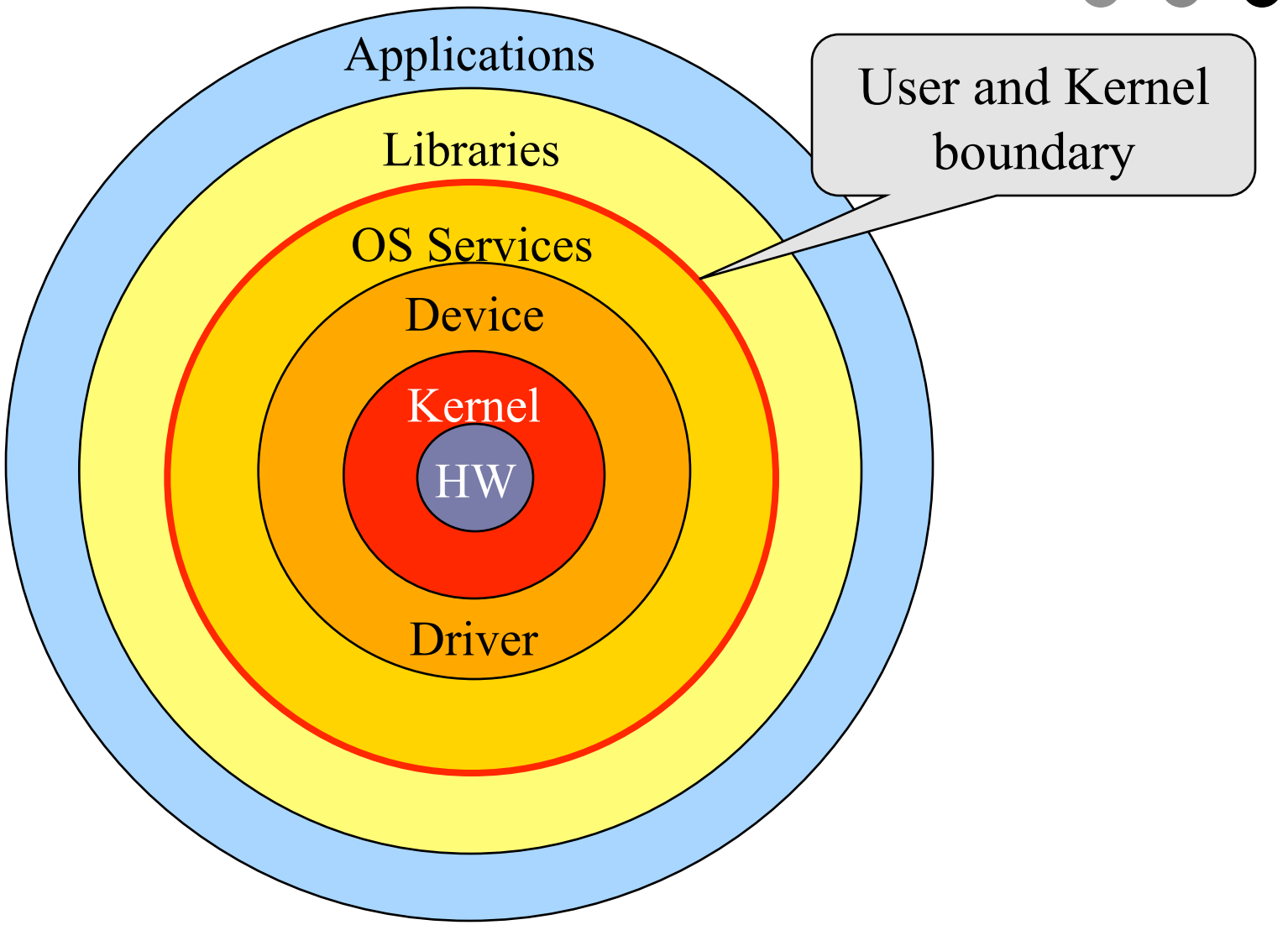
OS Service Examples

- Examples that are not provided at user level
  - System calls: file open, close, read and write
  - Control the CPU so that users won't stuck by running
    - while (1) ;
  - Protection:
    - Keep user programs from crashing OS
    - Keep user programs from crashing each other
- System calls are typically traps or exceptions
  - System calls are implemented in the kernel
  - When finishing the service, a system returns to the user code

Typical Unix OS Structure

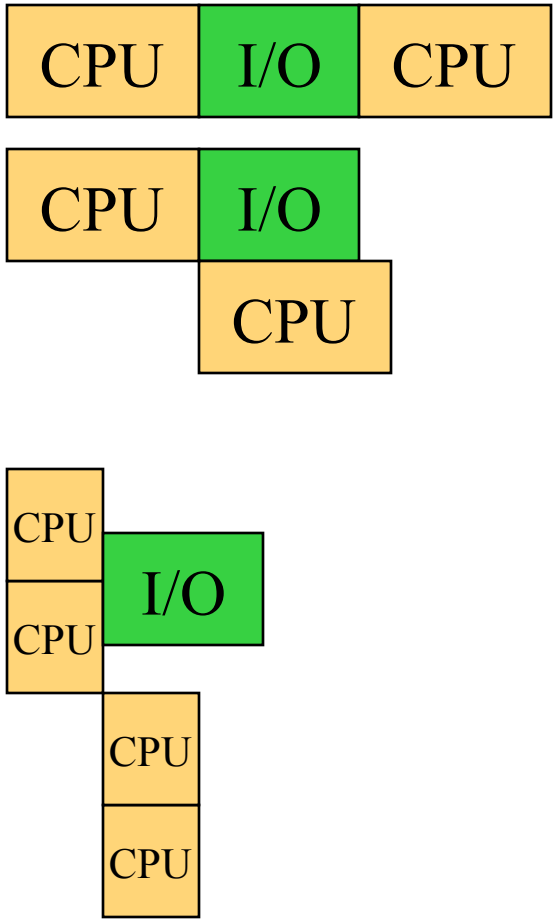


Software “Onion” Layers



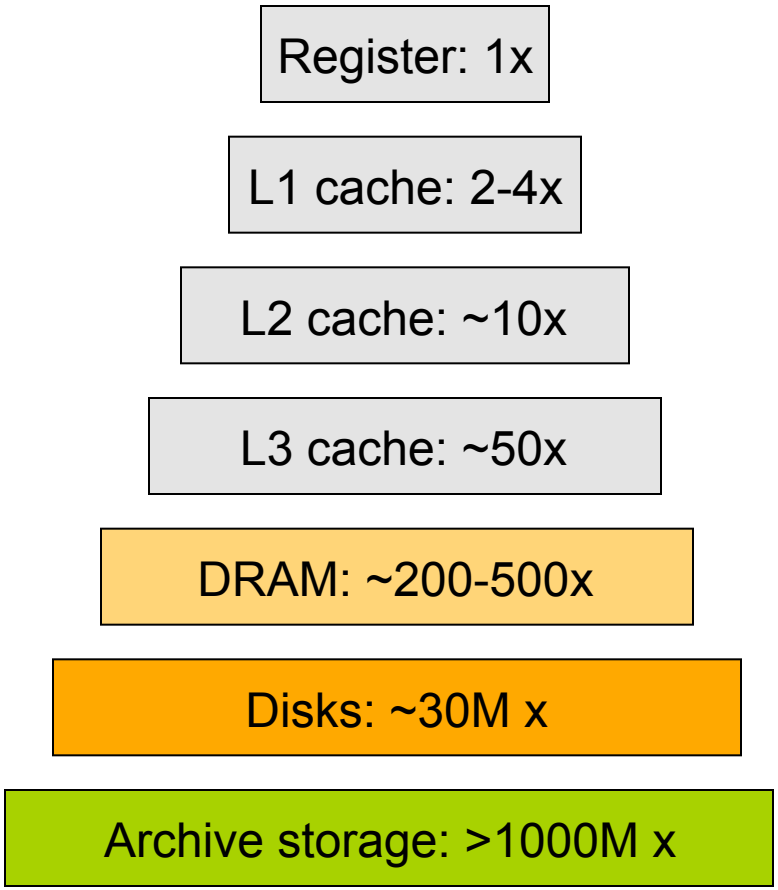
Processor Management

- Goals
  - Overlap between I/O and computation
  - Time sharing
  - Multiple CPU allocations
- Issues
  - Do not waste CPU resources
  - Synchronization and mutual exclusion
  - Fairness and deadlock free



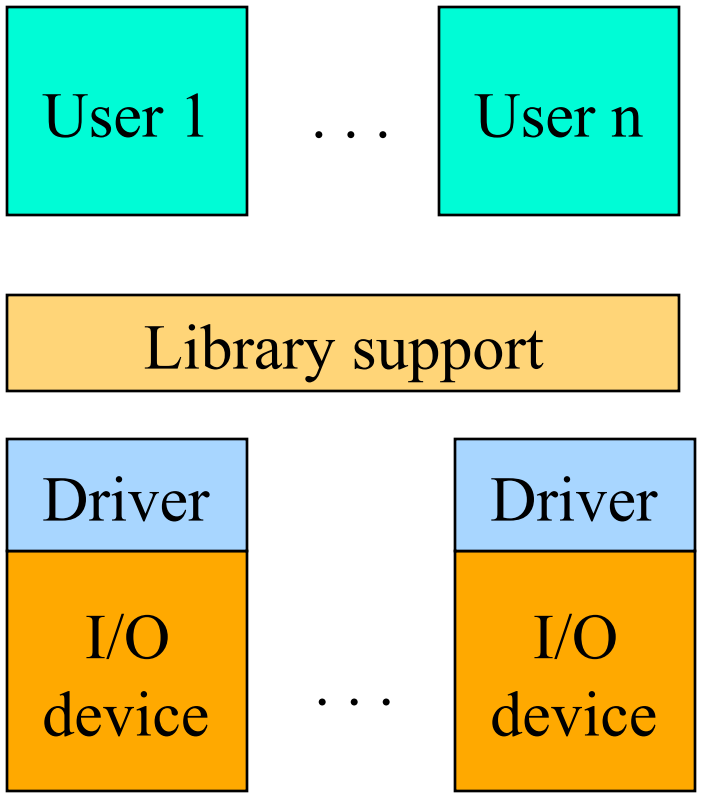
Memory Management

- Goals
  - Support programs to run
  - Allocation and management
  - Transfers from and to secondary storage
- Issues
  - Efficiency & convenience
  - Fairness
  - Protection



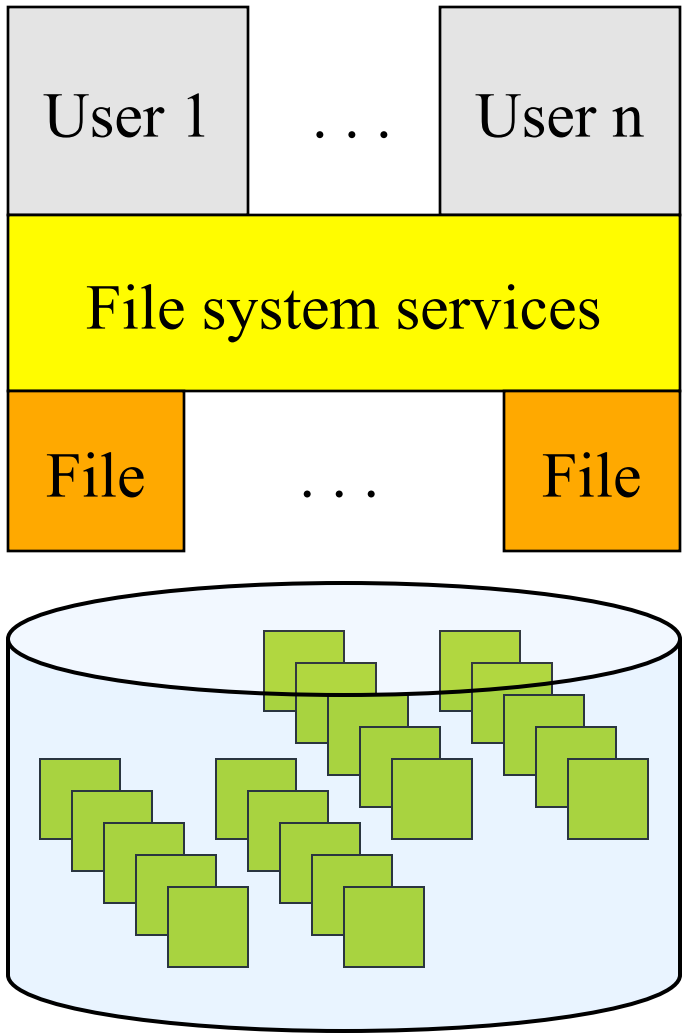
I/O Device Management

- Goals
  - Interactions between devices and applications
  - Ability to plug in new devices
- Issues
  - Efficiency
  - Fairness
  - Protection and sharing



File System

- Goals:
  - Manage disk blocks
  - Map between files and disk blocks
- A typical file system
  - Open a file with authentication
  - Read/write data in files
  - Close a file
- Issues
  - Reliability
  - Safety
  - Efficiency
  - Manageability

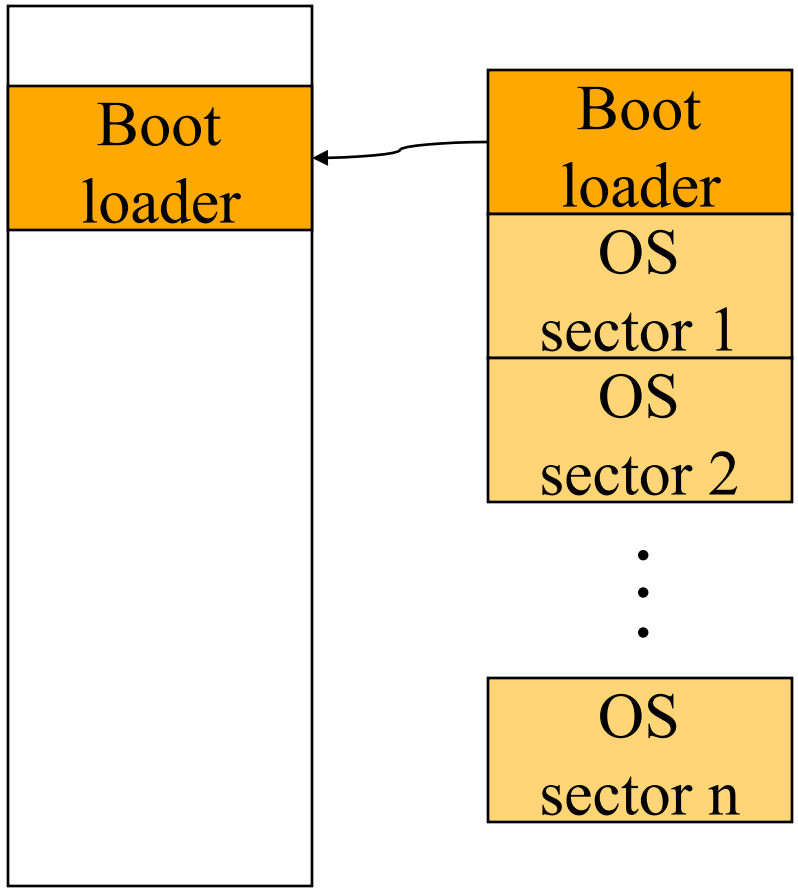


Window Systems

- Goals
  - Interacting with a user
  - Interfaces to examine and manage apps and the system
- Issues
  - Direct inputs from keyboard and mouse
  - Display output from applications and systems
  - Labor of division
    - All in the kernel (Windows)
    - All at user level
    - Split between user and kernel (Unix)

Bootstrap

- Power up a computer
- Processor reset
  - Set to known state
  - Jump to ROM code (BIOS is in ROM)
- Load in the boot loader from stable storage
- Jump to the boot loader
- Load the rest of the operating system
- Initialize and run
- Question: Can BIOS be on disk?



Ways to Develop An Operating System

- A hardware simulator
- A virtual machine
- A good kernel debugger
  - When OS crashes, always goes to the debugger
  - Debugging over the network
- Hire some smart programmers