COS 318: Operating Systems

Overview

Kai Li
Computer Science Department
Princeton University

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

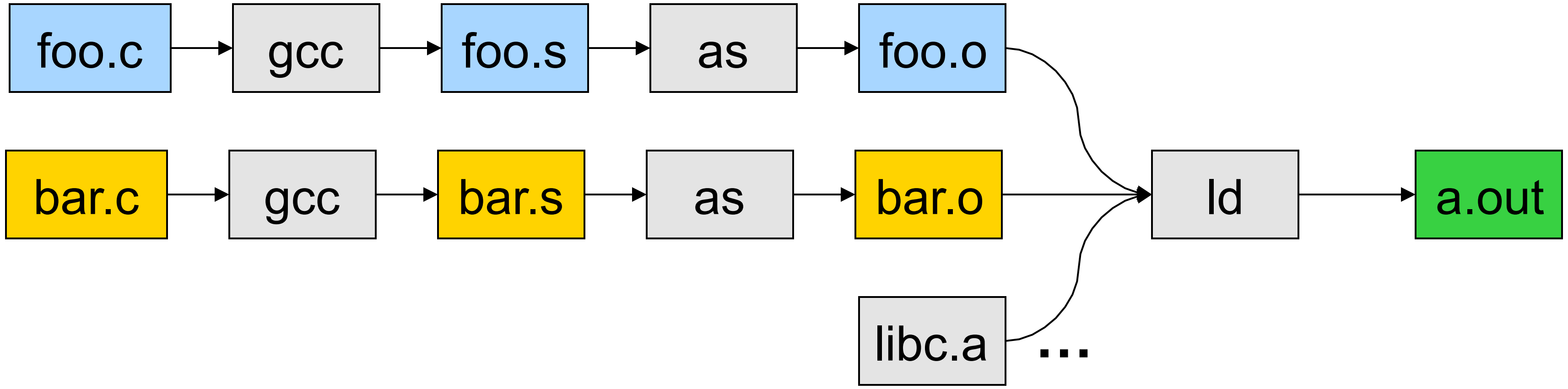
Important Times

- Lectures
  - 9/20 Lecture is here
  - Other lectures in Bowen Hall 222
- Precepts:
  - Tue: 7:30-8:20pm, 104 CS building
  - Thu (9/20): 7:30-8:20pm, 104 CS building
    - Tutorial of Assembly programming and kernel debugging
- Project 1
  - Design review:
    - 9/24: 10:30am – 10:30pm (Signup online), 010 Friends center
  - Project 1 due: 9/30 at 11:59pm
- To do:
  - Lab partner? Enrollment?

Today

- Overview of OS functionalities
- Overview of OS components

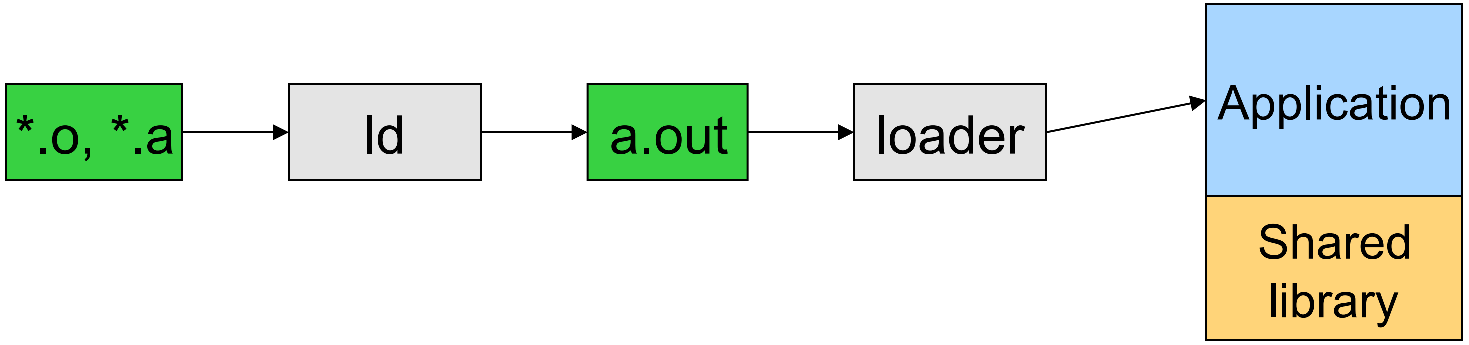
User’s View: Create An Executable File



- 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

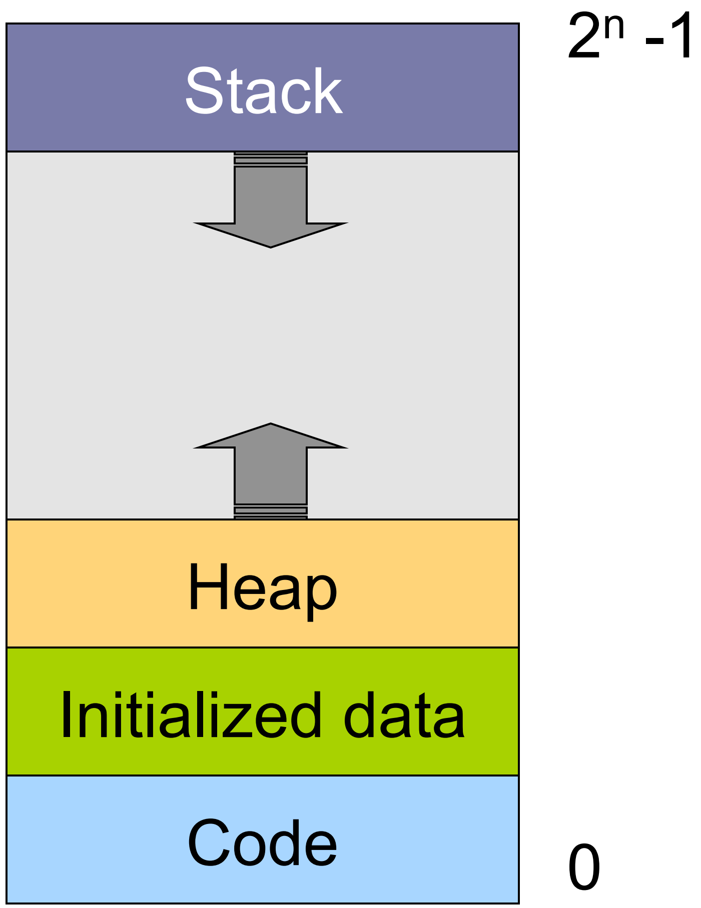
Run An Application

- 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

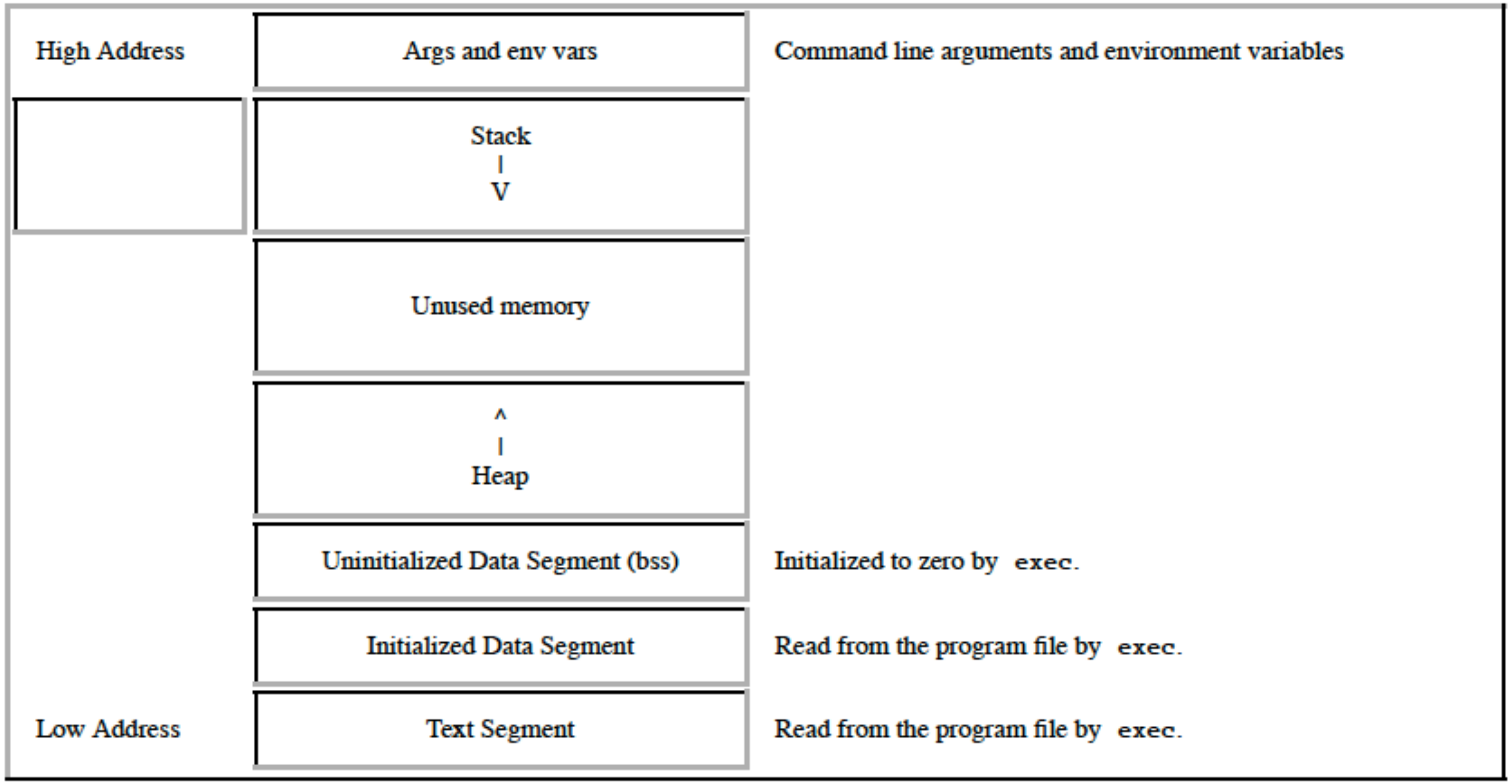


What's An Application?

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



In More Detail

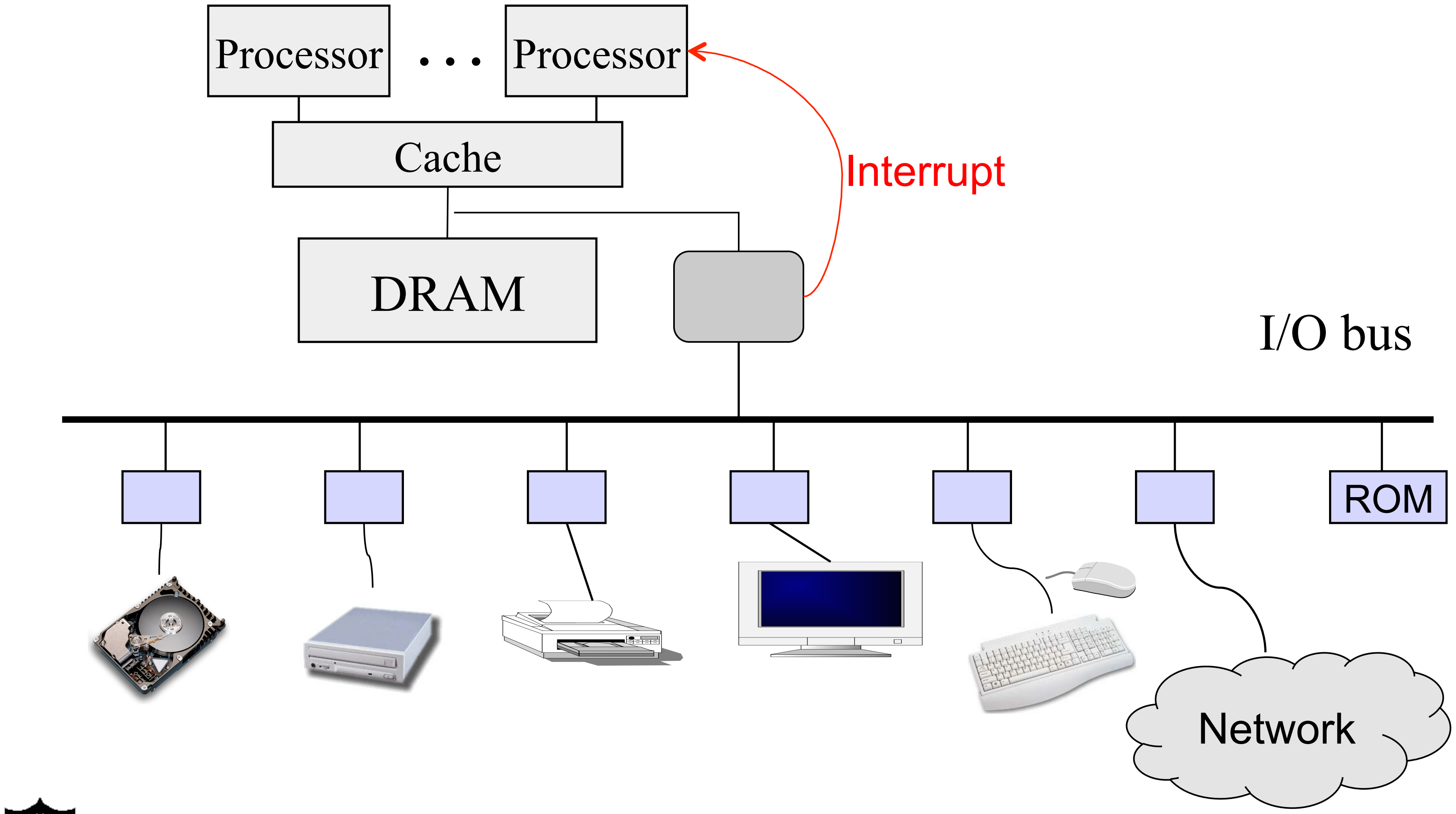


High Address	Args and env vars	Command line arguments and environment variables
	Stack | V	
	Unused memory	
	^ | Heap	
	Uninitialized Data Segment (bss)	Initialized to zero by `exec`.
	Initialized Data Segment	Read from the program file by `exec`.
Low Address	Text Segment	Read from the program file by `exec`.

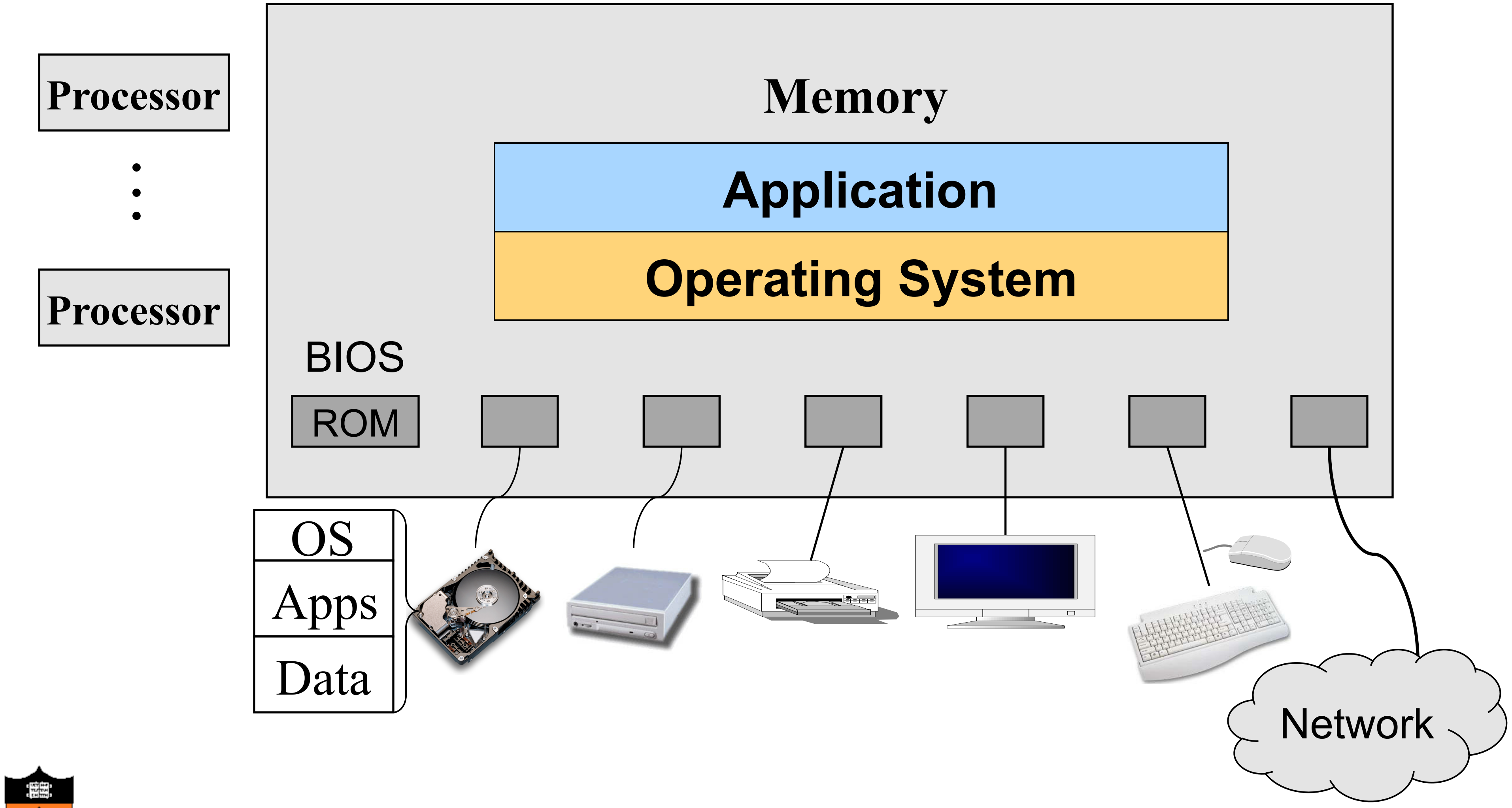
Responsibilities

- 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

Hardware View of A Typical Computer

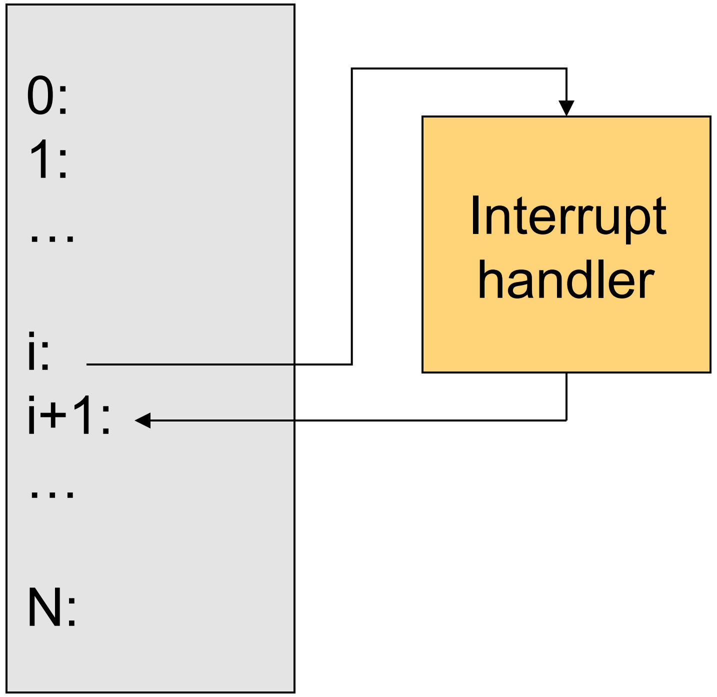


Software + Hardware View

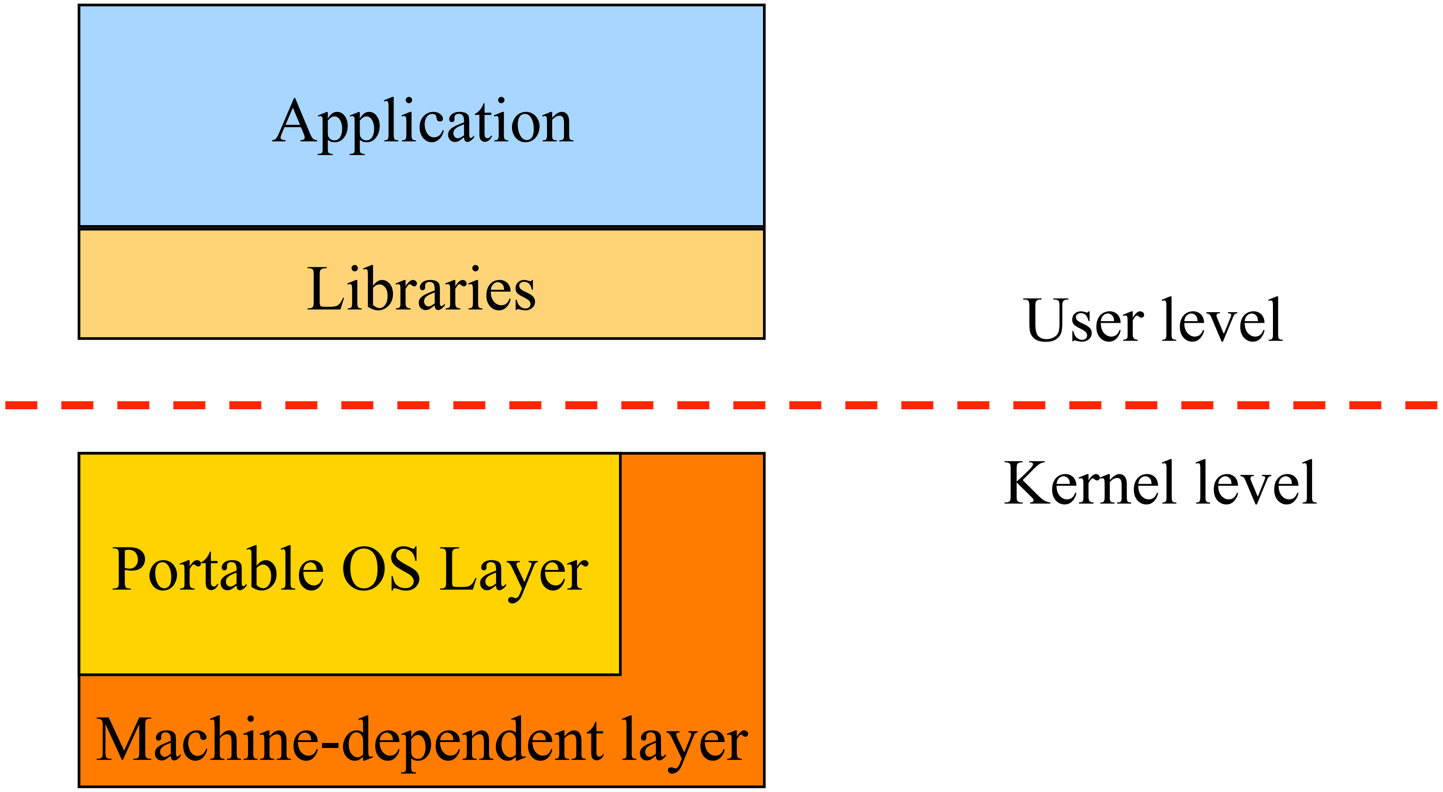


Software View of Interrupts

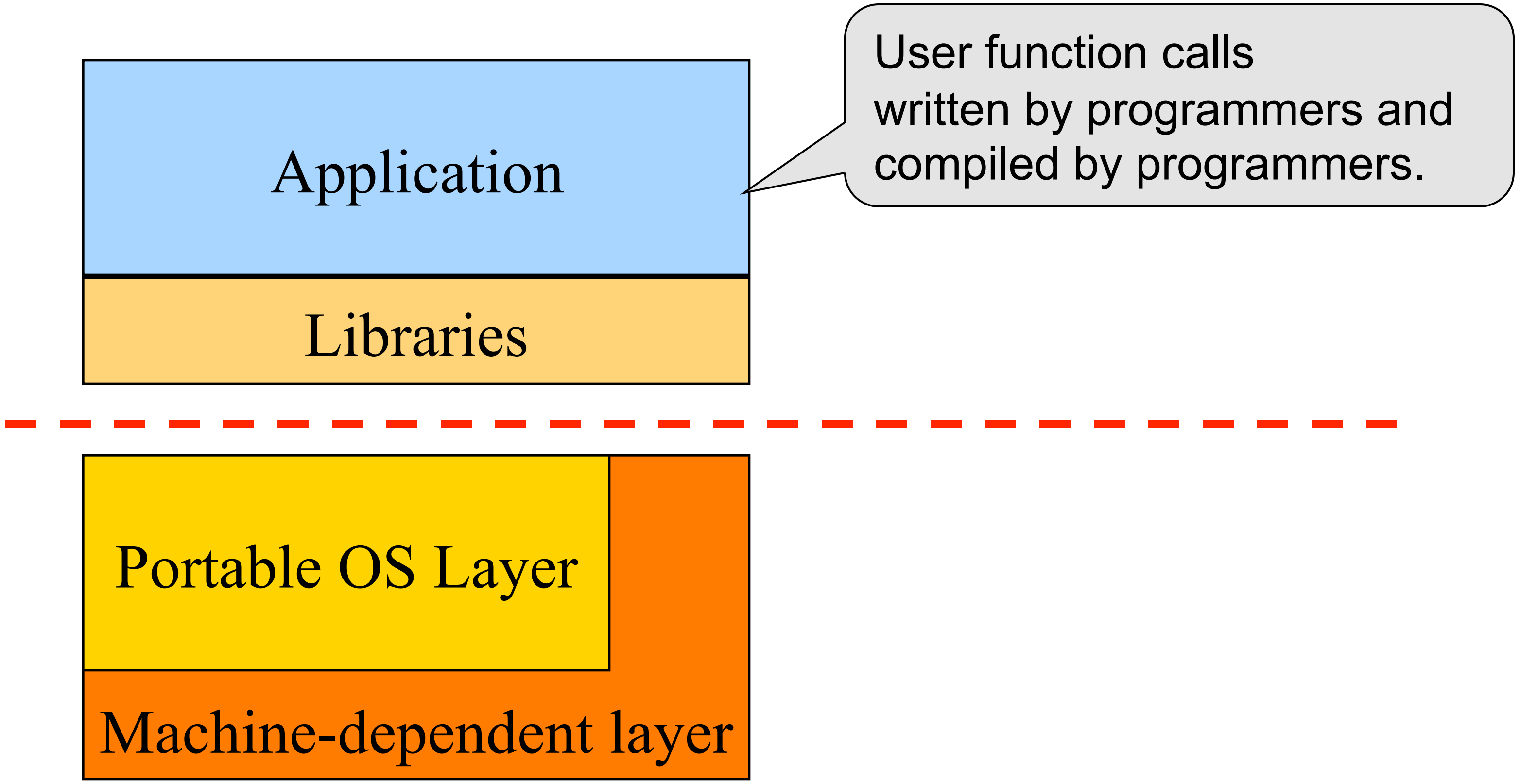
- Raised by external events
- Interrupt handler is in the kernel
  - Switch to another process
  - Overlap I/O with CPU
  - …
- Eventually resume the interrupted process



Typical Unix OS Structure

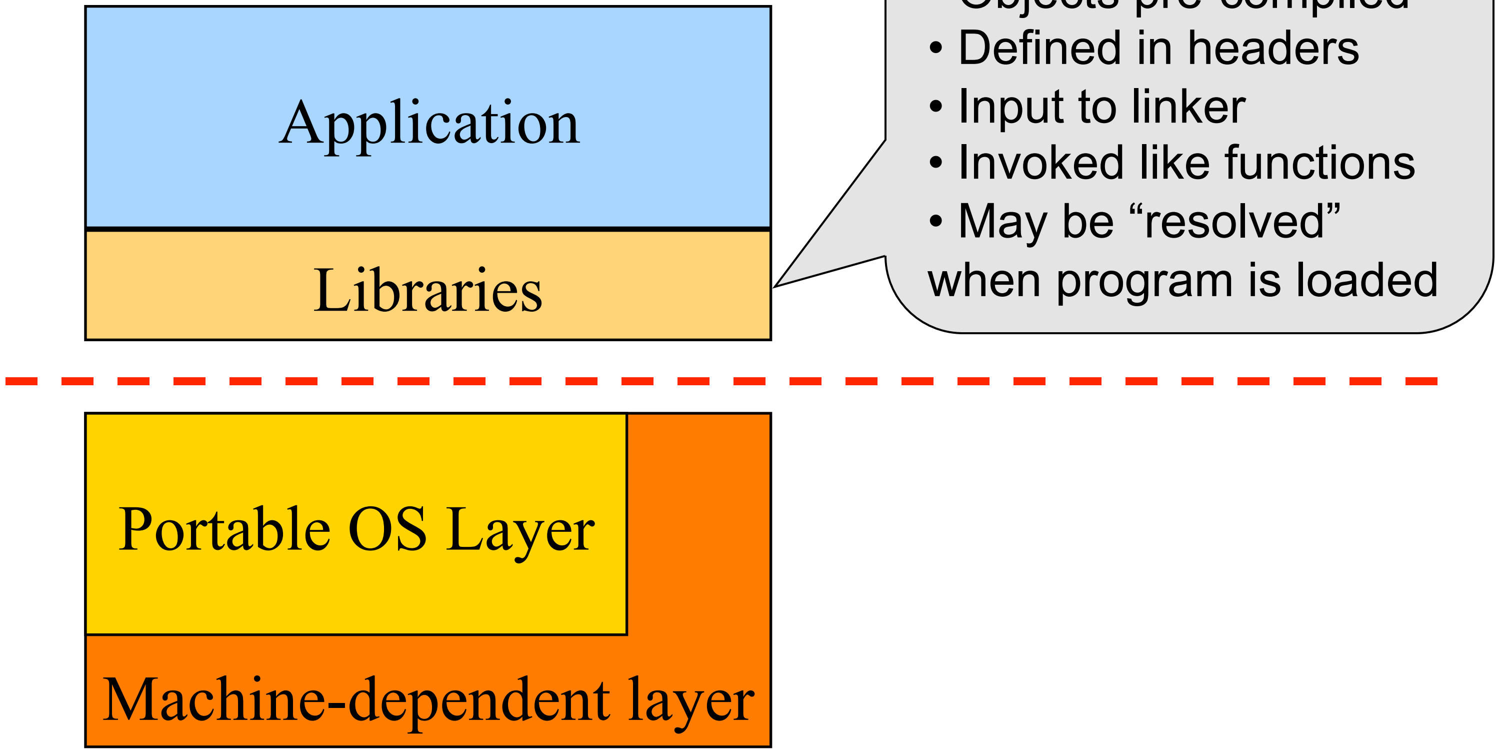


Typical Unix OS Structure



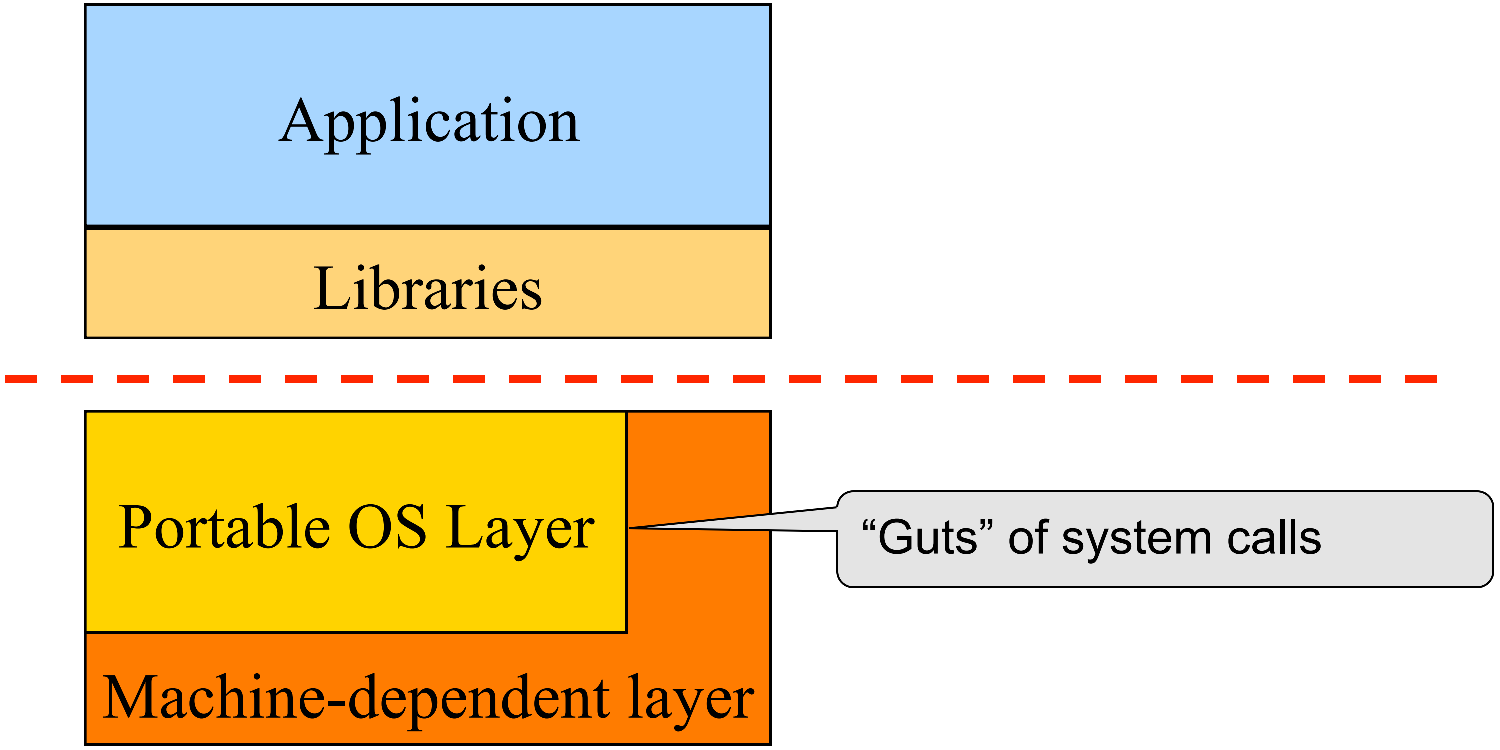
User function calls written by programmers and compiled by programmers.

Typical Unix OS Structure



- Written by elves
- Objects pre-compiled
- Defined in headers
- Input to linker
- Invoked like functions
- May be “resolved” when program is loaded

Typical Unix OS Structure



“Guts” of system calls

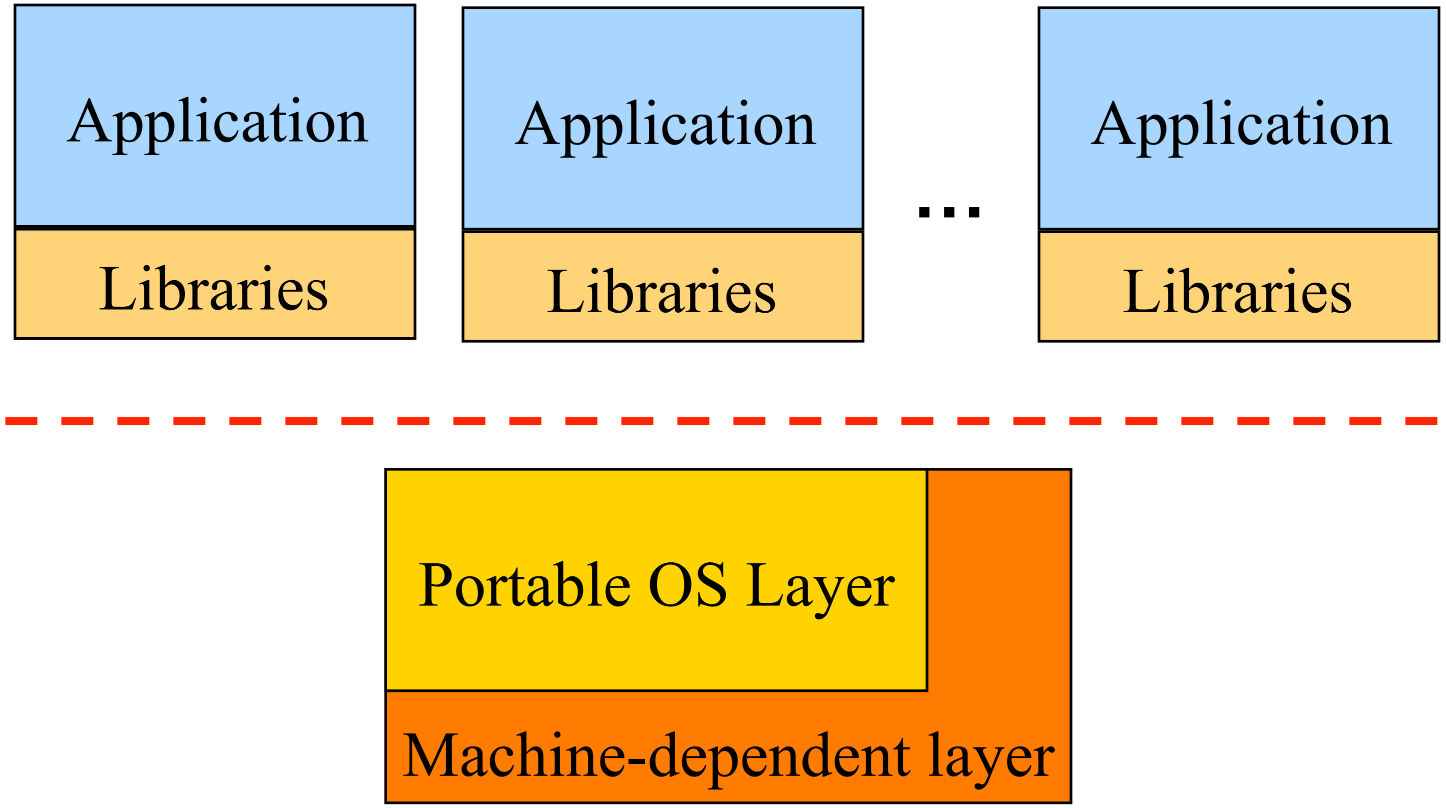
Run Multiple Applications

- Use multiple windows
  - Browser, shell, powerpoint, word, …

- Use command line to run multiple applications

```
% ls –al | grep ‘^d’
% foo &
% bar &
```

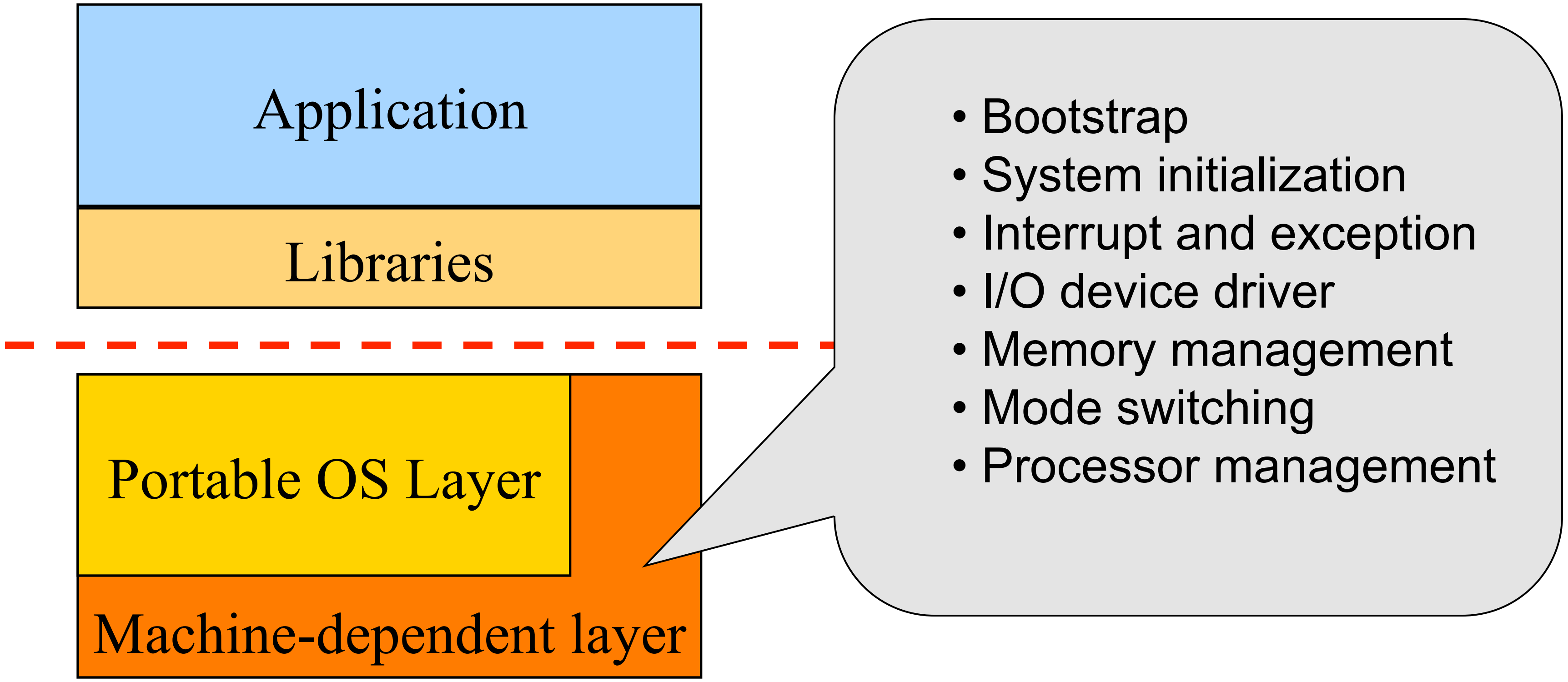
Support Multiple Processes



OS Service Examples

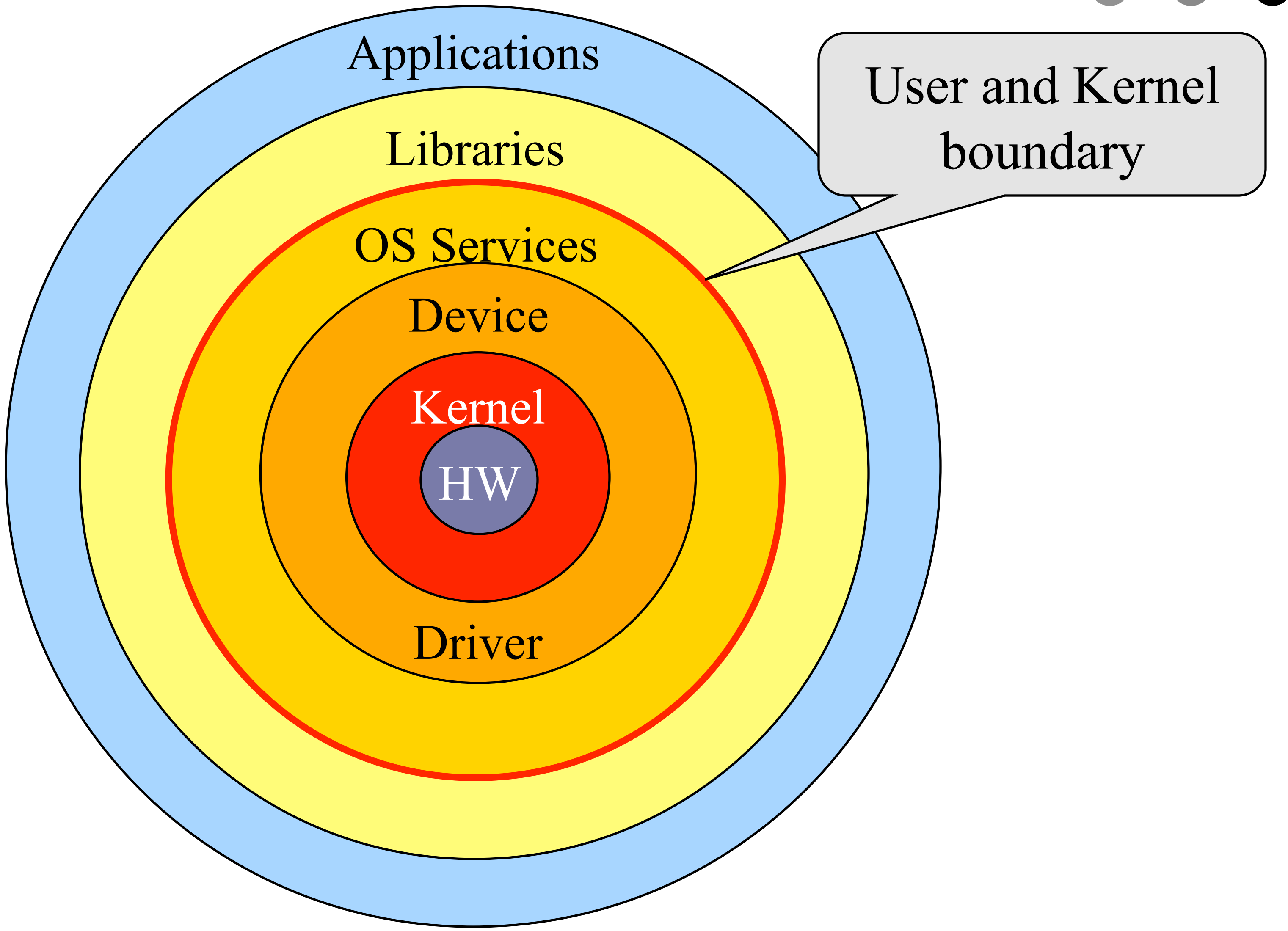
- Examples
  - System calls: fork, exec, exit, …
  - System calls: file open, close, read and write
  - Control the CPU so that users won’t stuck by running
    - while (1) ;
  - Protection:
    - Keep user programs from crashing OS
    - Keep user programs from crashing each other
- System calls are 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



- Bootstrap
- System initialization
- Interrupt and exception
- I/O device driver
- Memory management
- Mode switching
- Processor management

Software "Onion" Layers

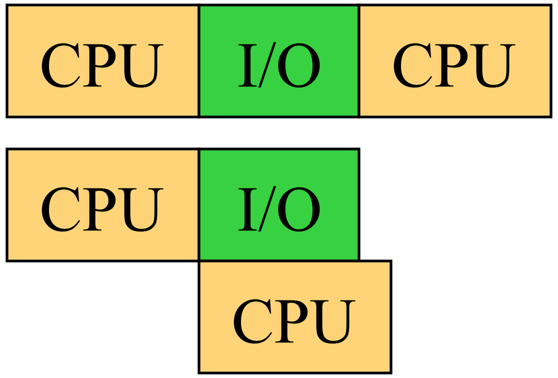


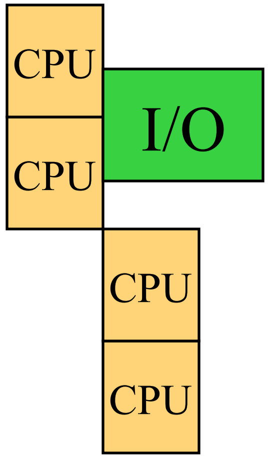
Today

- Overview of OS functionalities
- Overview of OS components

Processor Management

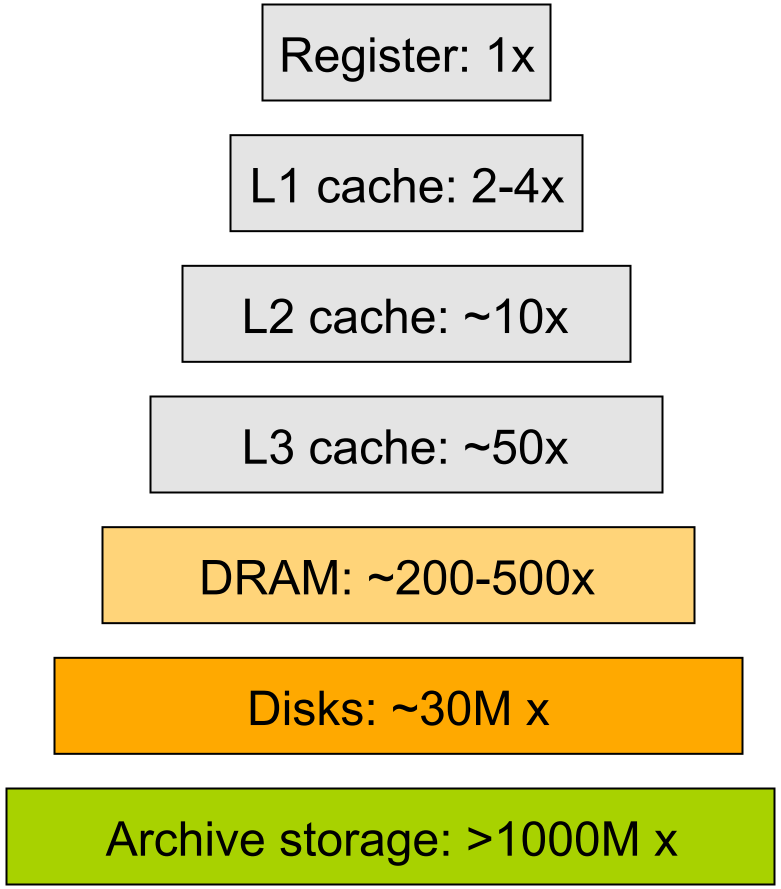
- Goals
  - Overlap between I/O and computation
  - Time sharing
  - Multiple CPU allocations
- Issues
  - Do not waste CPU resources
  - Synchronization and mutual exclusion
  - Fairness and deadlock free





Memory Management

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



Register: 1x

L1 cache: 2-4x

L2 cache: ~10x

L3 cache: ~50x

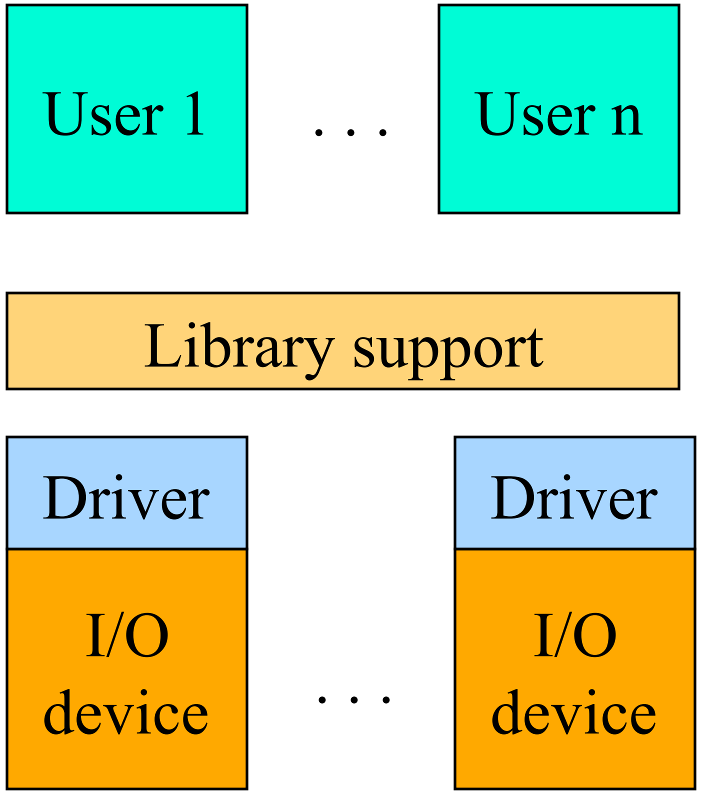
DRAM: ~200-500x

Disks: ~30M x

Archive storage: >1000M x

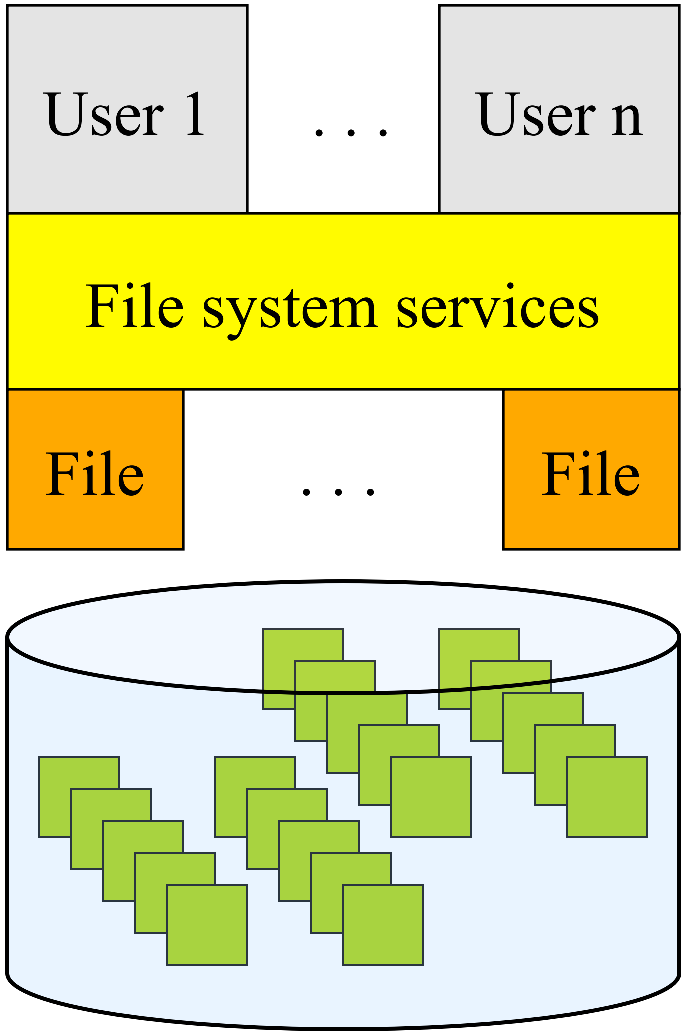
I/O Device Management

- 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



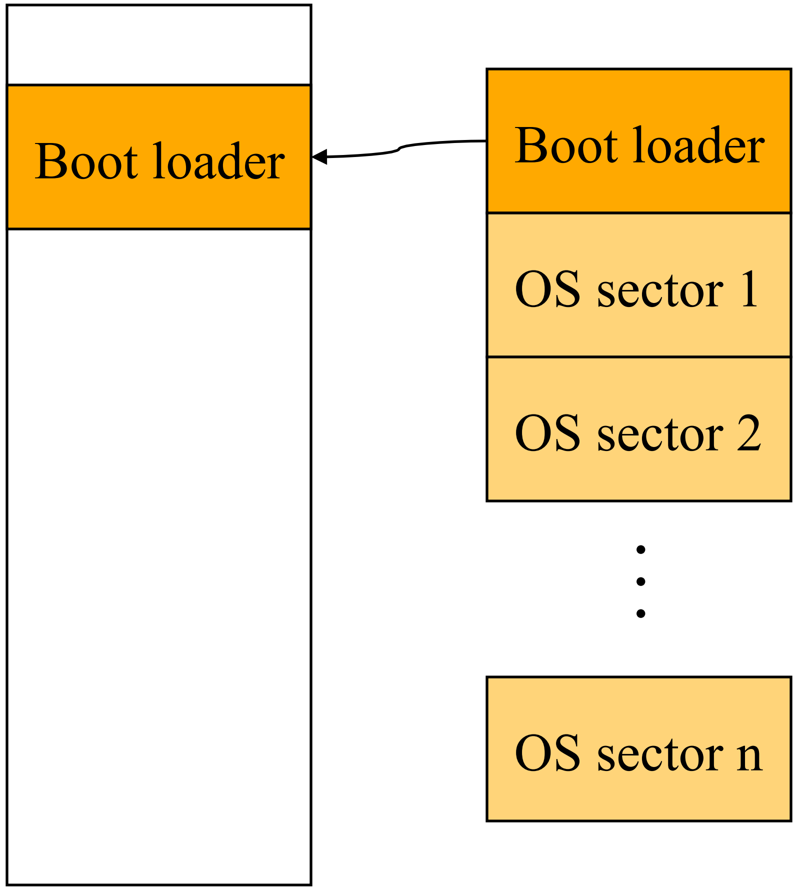
Window Systems

- Goals
  - Interacting with a user
  - Interfaces to examine and manage apps and the system
- Issues
  - Inputs from keyboard, mouse, touch screen, …
  - 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?



Develop An Operating System

- A hardware simulator
- A virtual machine
- A kernel debugger
  - When OS crashes, always goes to the debugger
  - Debugging over the network
- Smart people

1972

1998

Summary

- Overview of OS functionalities
  - Layers of abstractions
  - Services to applications
  - Manage resources
- Overview of OS components
  - Processor management
  - Memory management
  - I/O device management
  - File system
  - Window system
  - …