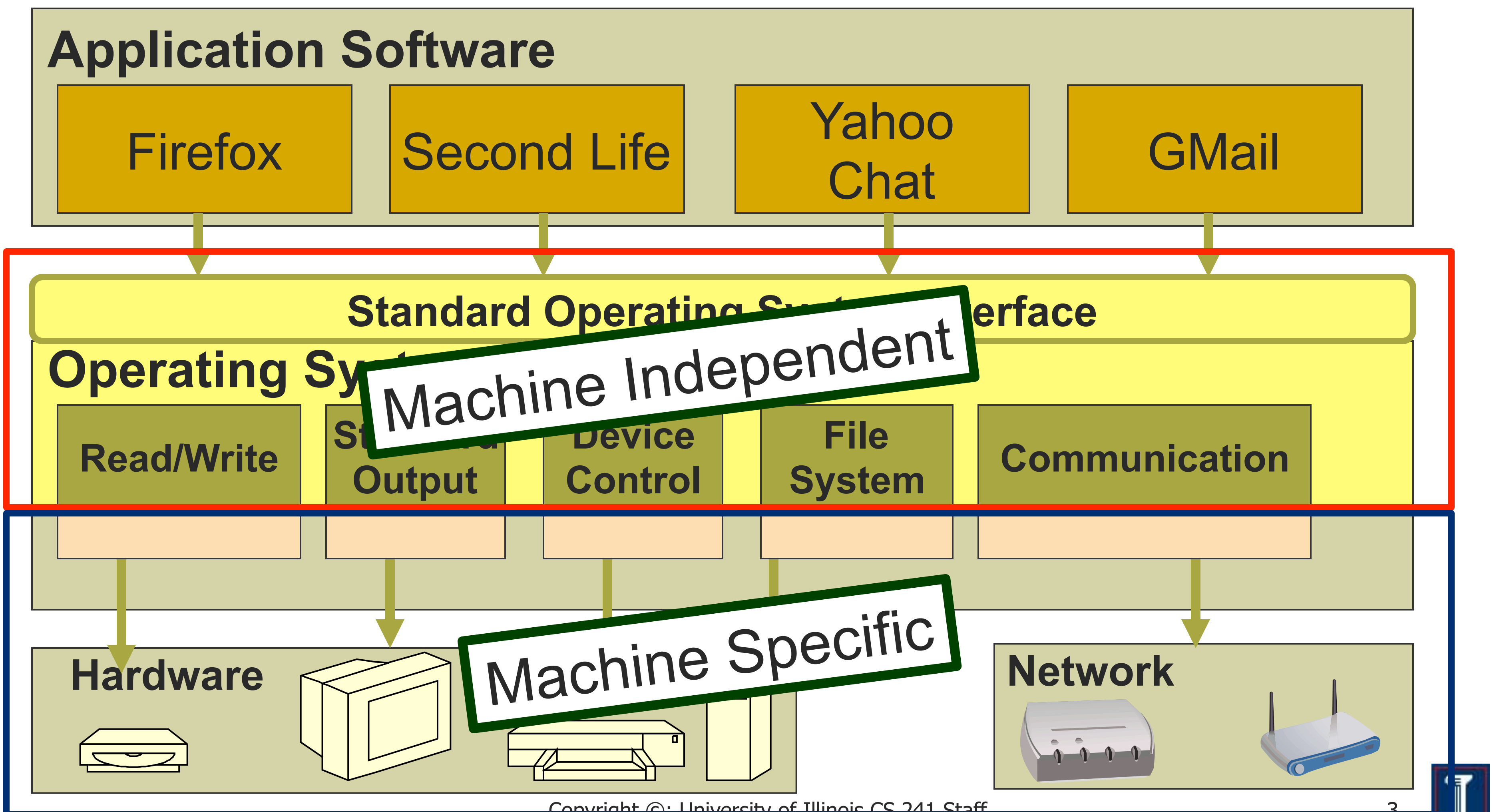
Operating Systems Orientation

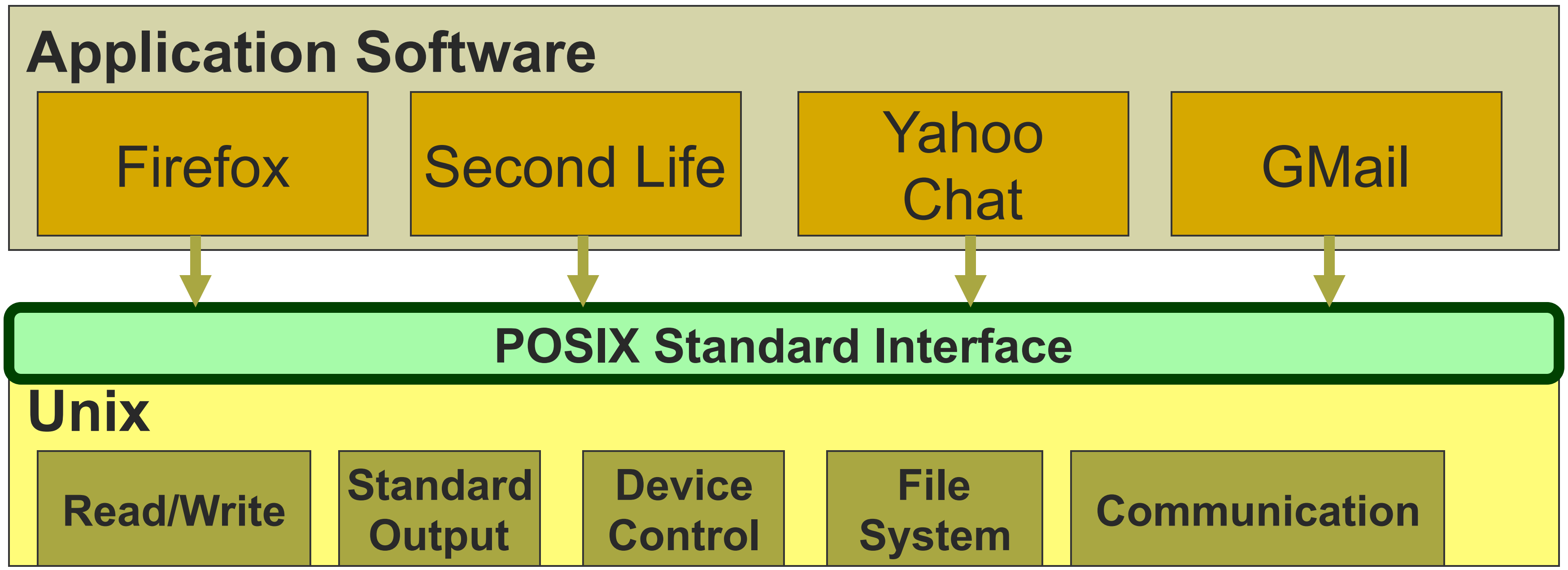
Objectives

- Explain the main purpose of operating systems
- Explain the POSIX standard (UNIX specification)
- Explain fundamental machine concepts
  - Instruction processing
  - Memory hierarchy
  - Interrupts
  - I/O

OS Structure



POSIX
The UNIX Interface Standard

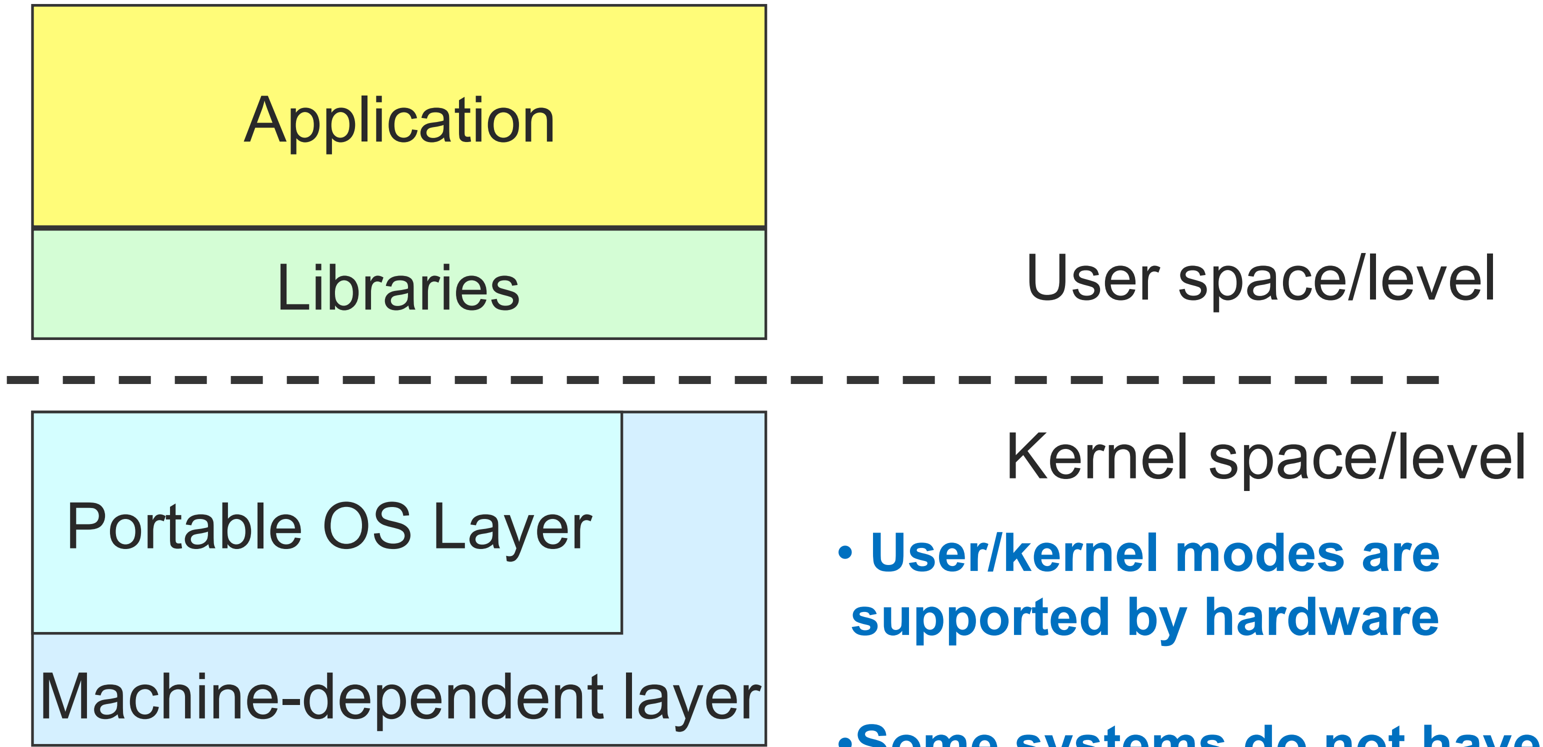


What is an Operating System?

- It is an *extended machine*
  - Hides the messy details that must be performed
  - Presents user with a virtualized and simplified abstraction of the machine, easier to use

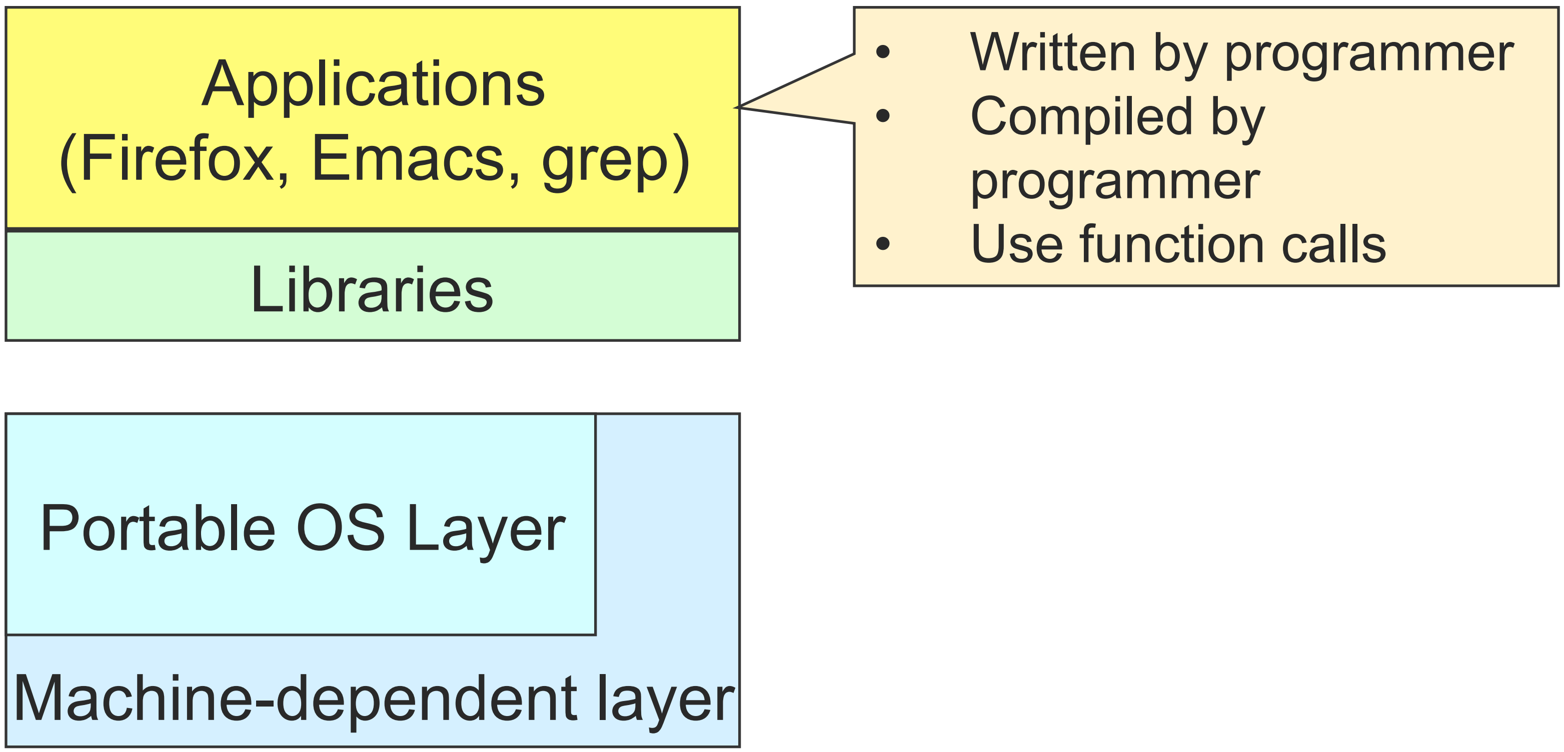
- It is a *resource manager*
  - Each program gets time with the resource
  - Each program gets space on the resource

A Peek into Unix

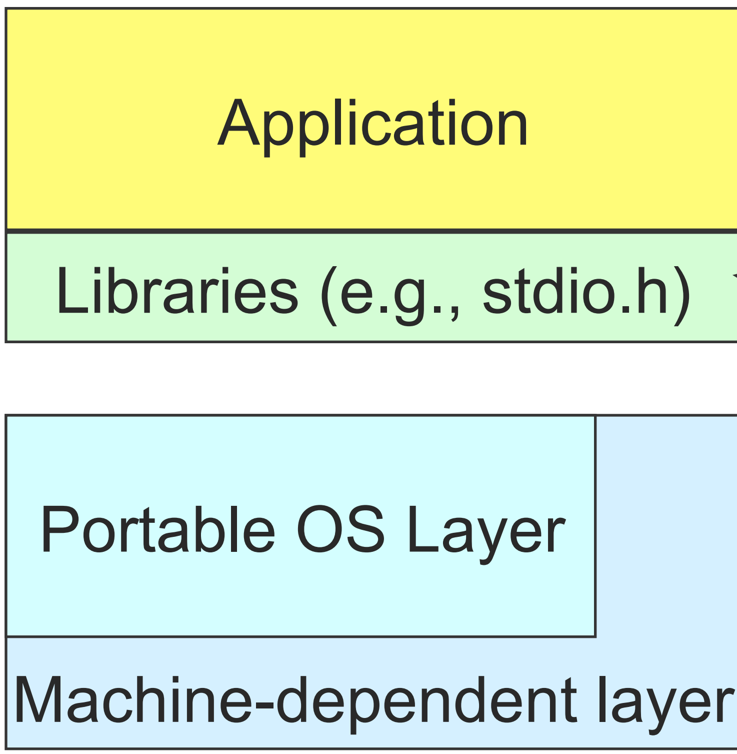


- **User/kernel modes are supported by hardware**
- **Some systems do not have clear user-kernel boundary**

Application

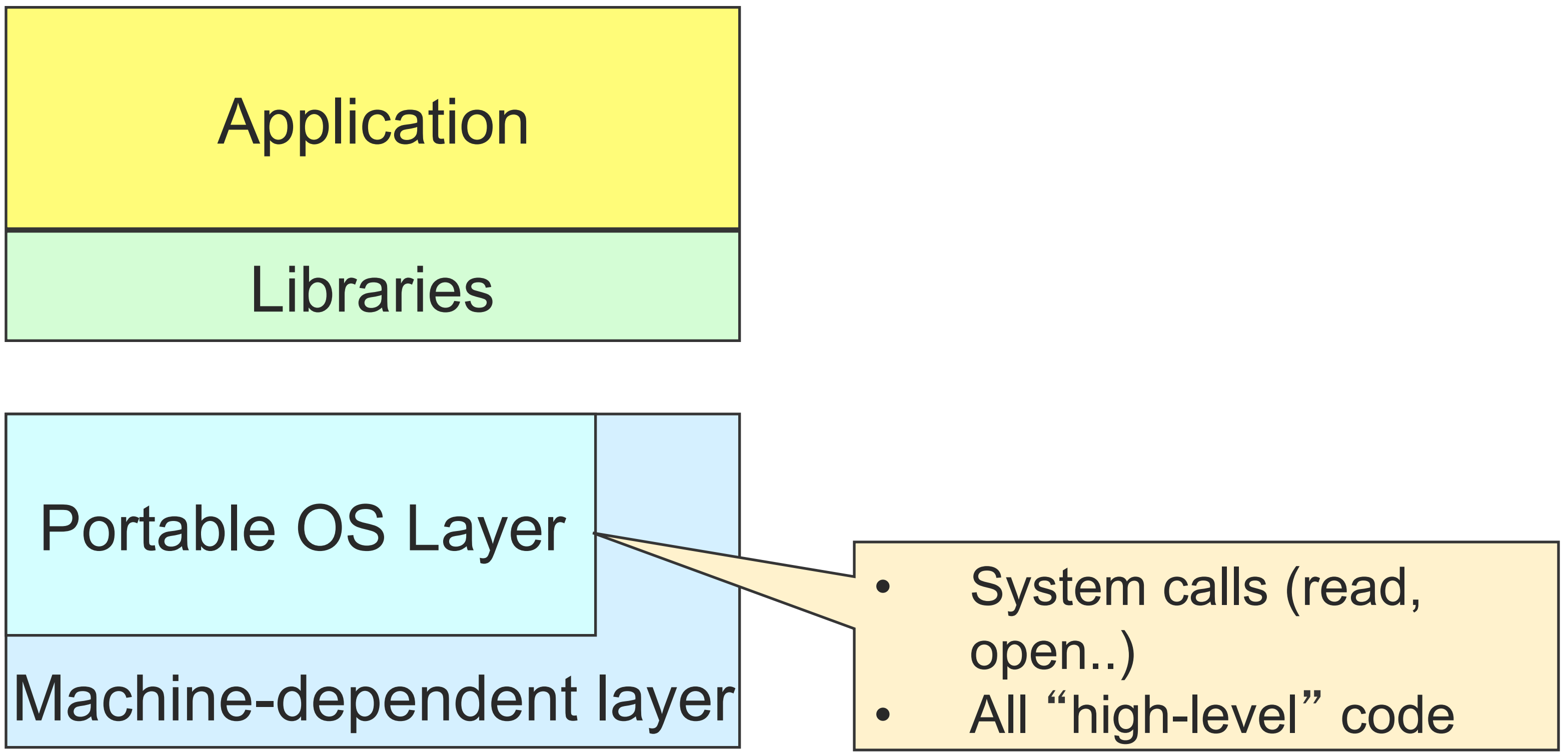


Unix: Libraries

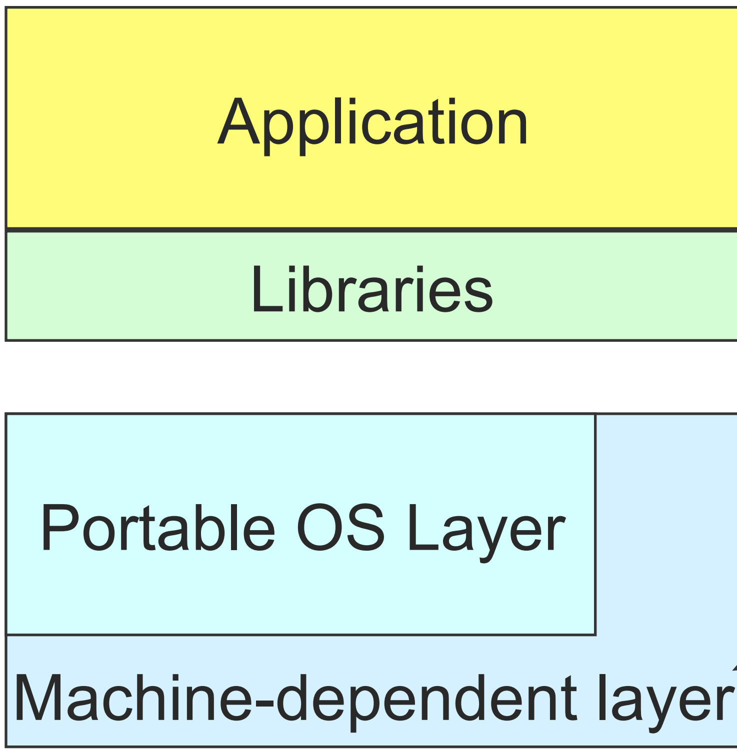


- Provided pre-compiled
- Defined in headers
- Input to linker (compiler)
- Invoked like functions
- May be “resolved” when program is loaded

Typical Unix OS Structure

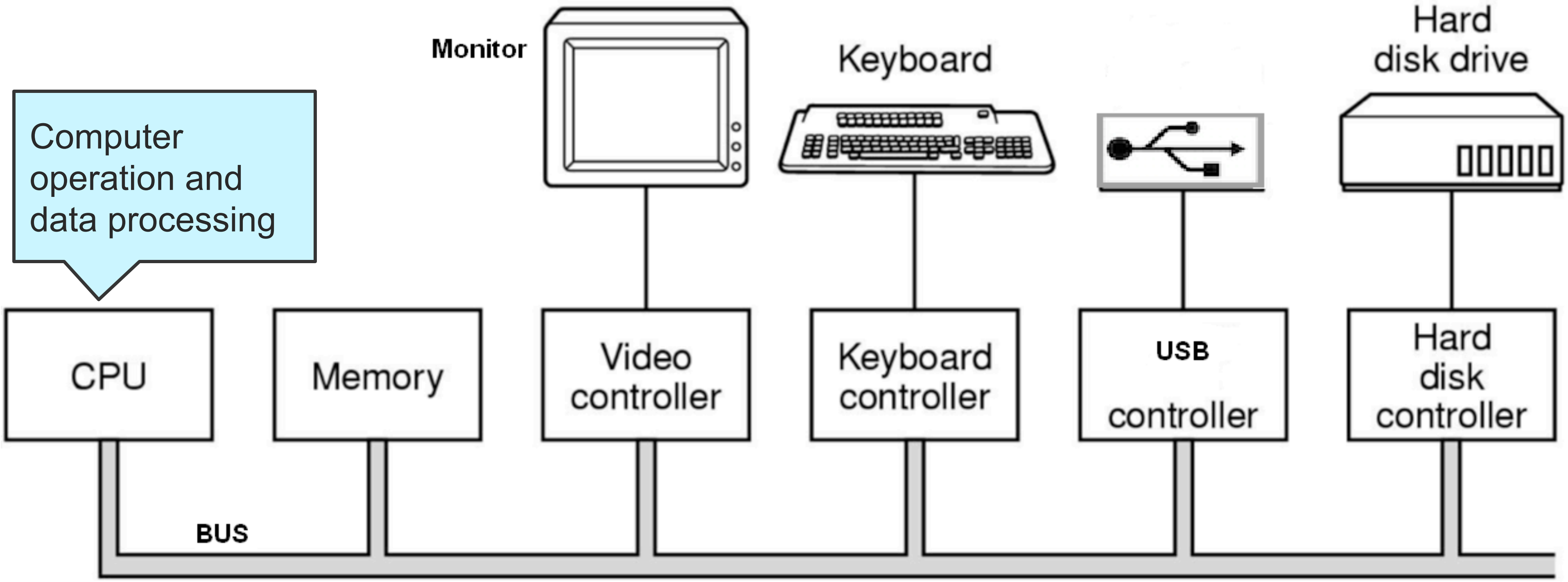


Typical Unix OS Structure



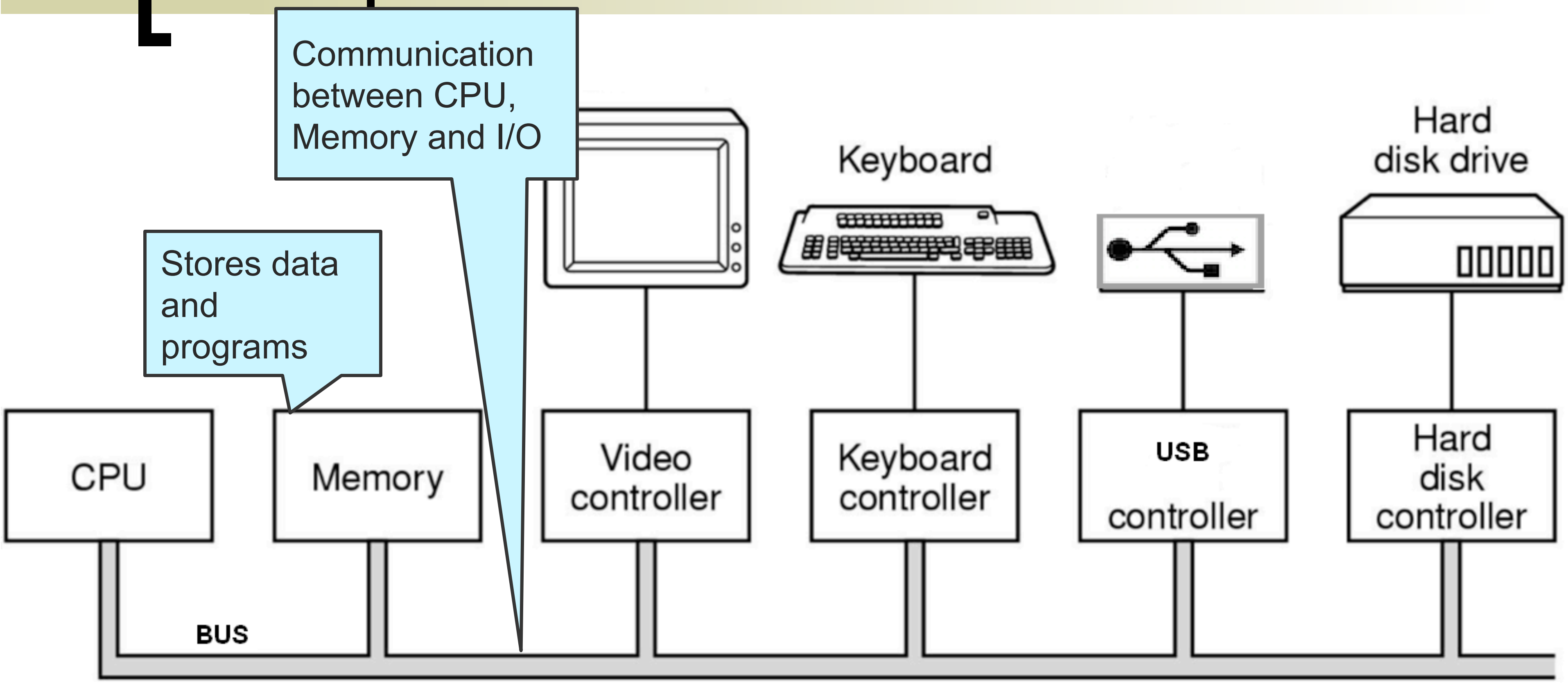
- Bootstrap
- System initialization
- Interrupt and exception
- I/O device driver
- Memory management
- Kernel/user mode switching
- Processor management

Computer Hardware Review



- Components of a simple personal computer

Computer Hardware Review



- Components of a simple personal computer

CPU & Registers

- Fetch instruction from code memory
- Fetch operands from data memory
- Perform operation (and store result)
- Check interrupt line
- Go to next instruction

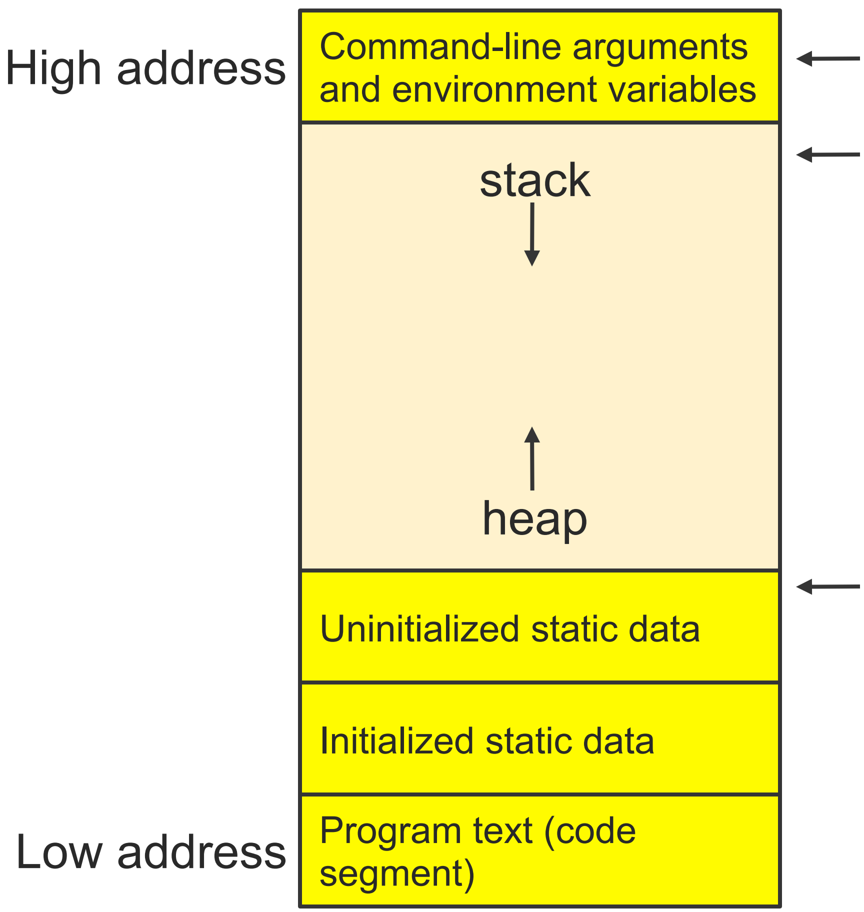
➔ CPU must maintain certain state information stored in internal registers

➔ ‘Simplified CPU’
(Ignore pipeline, optimization complexities)

CPU Registers Example

- Hold instruction operands
- Point to start of
  - Code segment (executable instructions)
  - Data segment (static/global variables)
  - Stack segment (execution stack data)
- CPU must also maintain certain state:
  - Current instruction to execute (program counter)
  - Stack pointer

Sample Layout for program image in main memory



argc, argv, environment

Activation record for function calls (return address, parameters, saved registers, automatic variables

Allocations from **malloc** family

Processes have three segments: text, data, stack

Memory Hierarchy

- Leverage locality of reference

1. Decreasing cost per bit
2. Increasing capacity
3. Increasing access time
4. Decreasing frequency of access

Typical access time		Typical capacity
~0.5 nsec	Registers	<1 KB
~5 nsec	Cache	~4 MB
~20 nsec	Main memory	~4 GB
~10 msec	Magnetic disk	~1TB
seconds	Magnetic tape	several TB

I/O Device Access

- System Calls
  - Application makes a system call
  - OS translates to specific driver
  - Driver starts I/O
  - Polls device for completion or blocks user process until device controller generates interrupt
- Interrupts
  - Application requests an I/O operation
  - OS activates I/O device and asks for an interrupt upon completion
  - OS blocks application (i.e., blocking I/O system call)
  - When data available, I/O device controller generates interrupt

Operating System Concepts

- Shared resources
  - I have B GB of memory, but need N*B GB
  - I have N processes trying to access the network, disk, etc. at the same time
  - How would you control access to resources?
- Challenges
  - Who gets to use the resources?
  - How do you control fair use of the resources over time?
  - How to avoid deadlock?

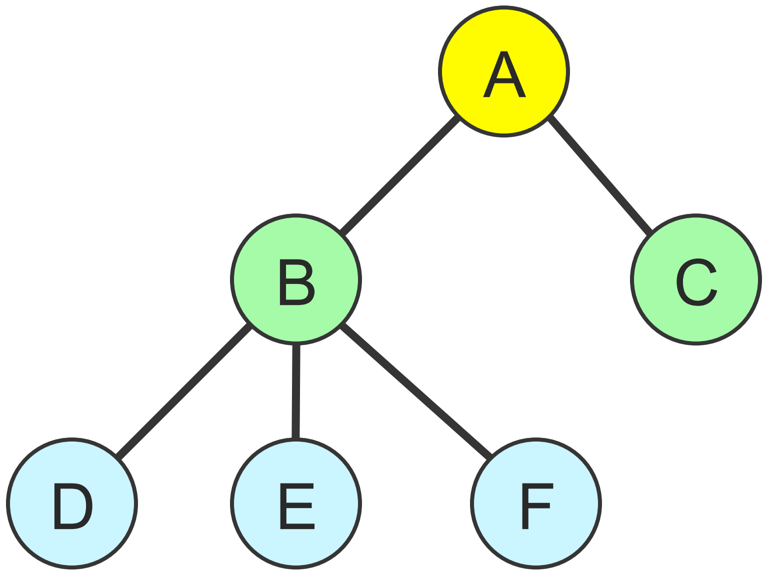
Operating System Concepts

- Process
  - An executable instance of a program
  - Only one process can use a (single-core) CPU at a time

- How is a program different from a process?
  - a program is a passive collection of instructions;
  - a process is the actual execution of those instructions; each process has a state to keep track of its execution
  - Several processes may be associated with the same program and share the same read-only code segment; for example, opening up several instances of the same program (like terminal) often means more than one process is being executed.

Operating System Concepts

- A process tree
  - A created two child processes, B and C
  - B created three child processes, D, E, and F



Operating System Concepts

- How would you switch CPU execution from one process to another?
- Solution: Context Switching
  - Store/restore state on CPU, so execution can be resumed from same point later in time
  - Triggers: multitasking, interrupt handling, user/kernel mode switching
  - Involves: Saving/loading registers and other state into a “process control block” (PCB)
  - PCBs stored in kernel memory

Operating System Concepts

- Context Switching
  - What are the costs involved?

Item	Time	Scaled Time in Human Terms (2 billion times slower)
Processor cycle	0.5 ns (@ 2 GHz)	1 s
Cache access	~5 ns	10 s
Memory access	~20 ns	40 s
Context switch overhead (Linux)	~5,000 ns (5 usec)	167 min
System quanta	~2,500,000 ns (2.5 ms)	57 days
Disk access	~10,000,000 ns (10 ms)	230 days

Operating System Concepts

- Inter-process Communication
  - Now process A needs to exchange information with process B
  - How would you enable communication between processes?