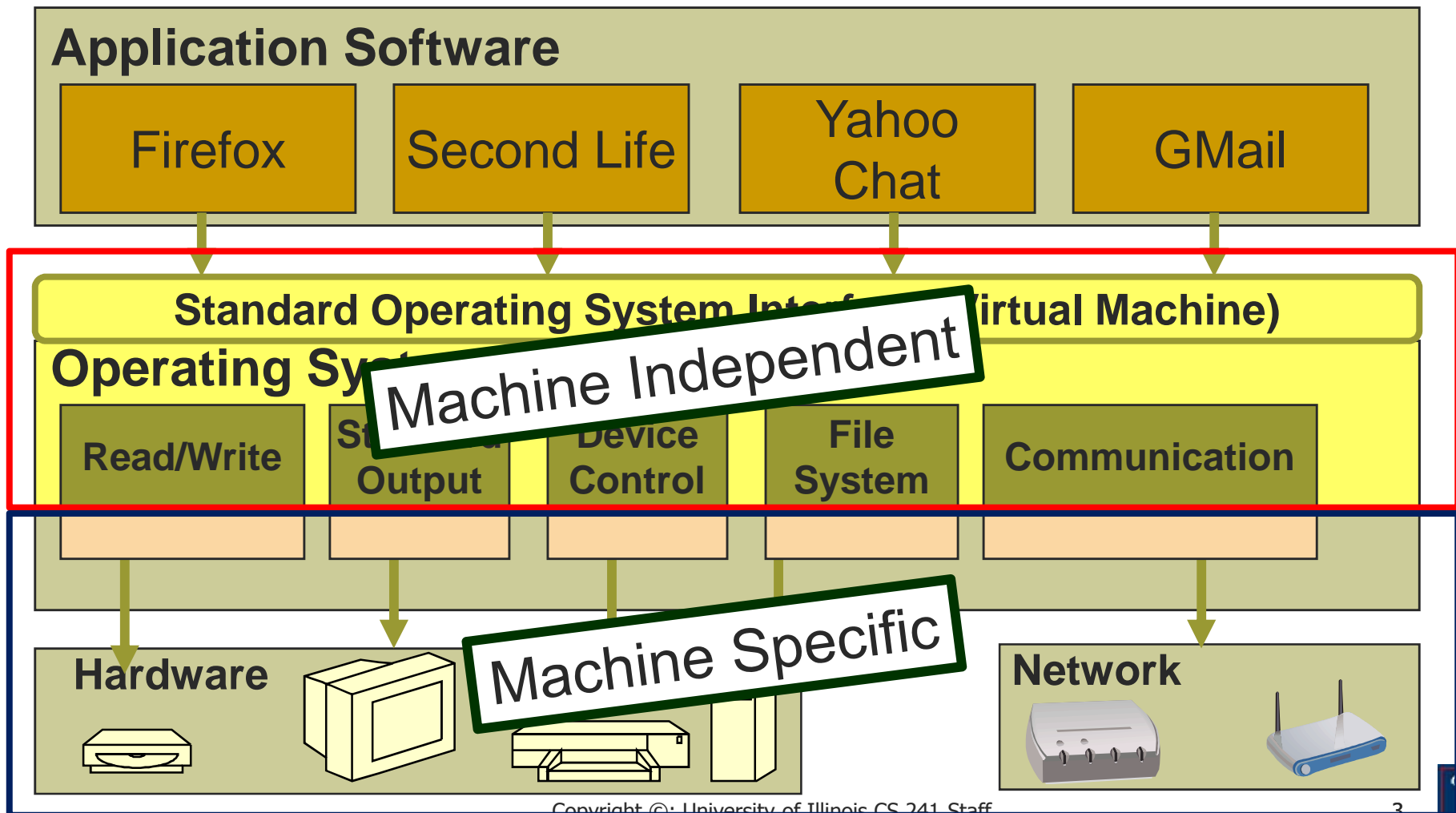# Operating Systems Orientation

Objectives

- Explain the main purpose of operating systems and describe milestones of OS evolution

- Explain fundamental machine concepts
  - Instruction processing
  - Memory hierarchy
  - Interrupts
  - I/O

- Explain fundamental OS concepts
  - System calls
  - Processes
  - Synchronization
  - Files

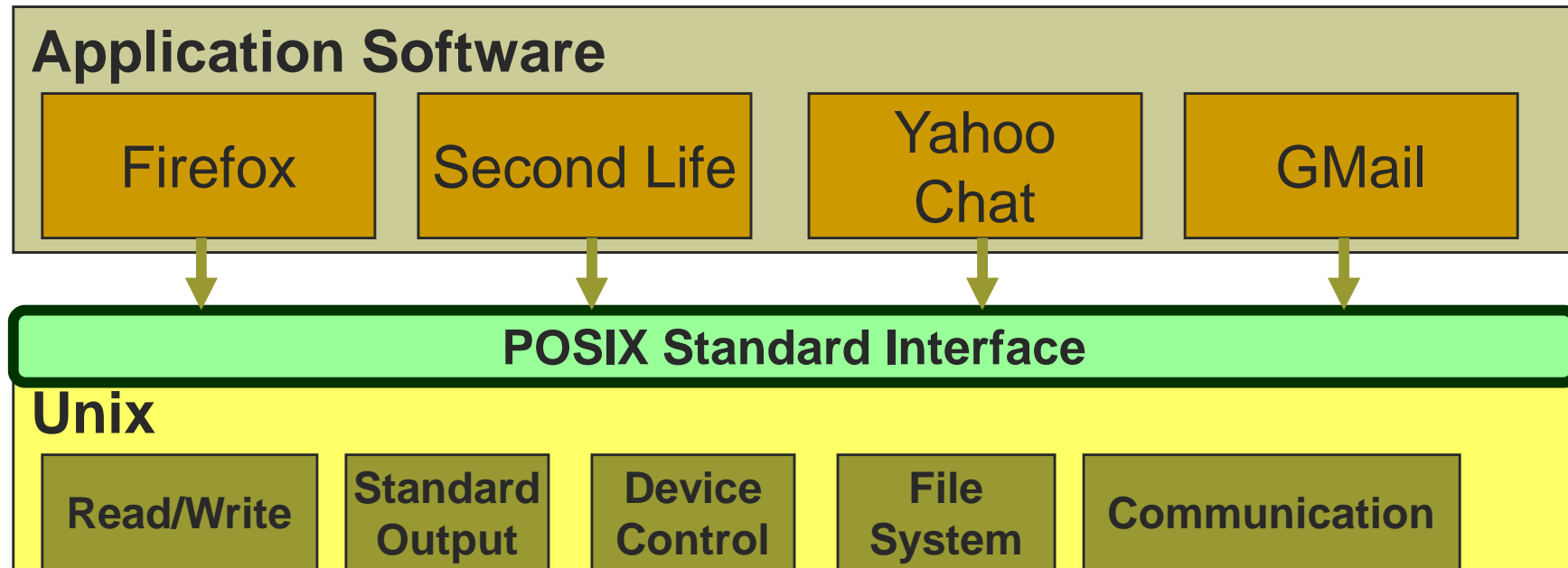- Explain the POSIX standard (UNIX specification)

OS Structure

Application Software

| Firefox | Second Life | Yahoo Chat | GMail |

Standard Operating System Interface (Virtual Machine)

Operating System

| Read/Write | Standard Output | Device Control | File System | Communication |

Machine Independent

Machine Specific

Hardware

Network

POSIX
The UNIX Interface Standard

Application Software

| Firefox | Second Life | Yahoo Chat | GMail |

POSIX Standard Interface

Unix

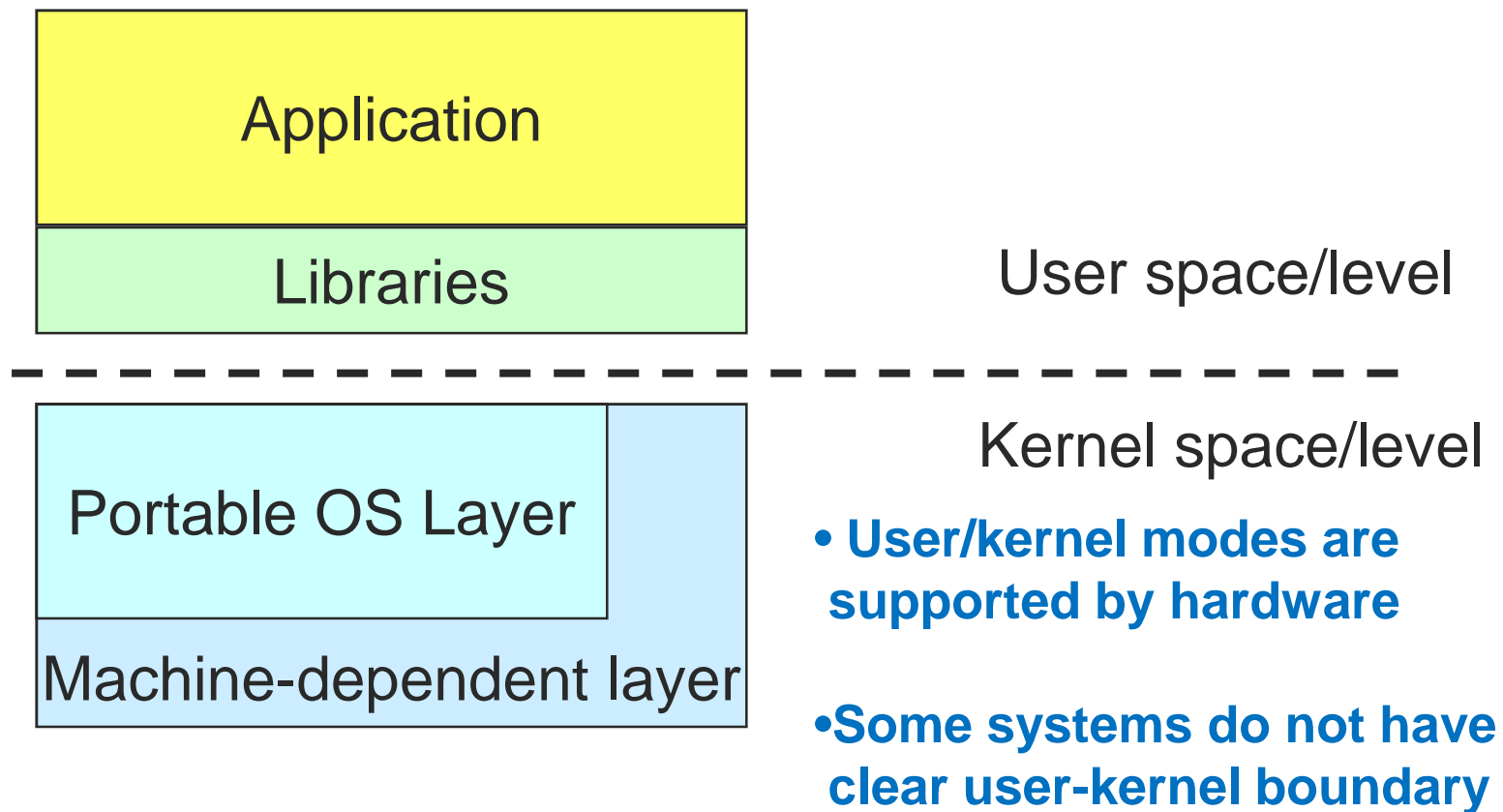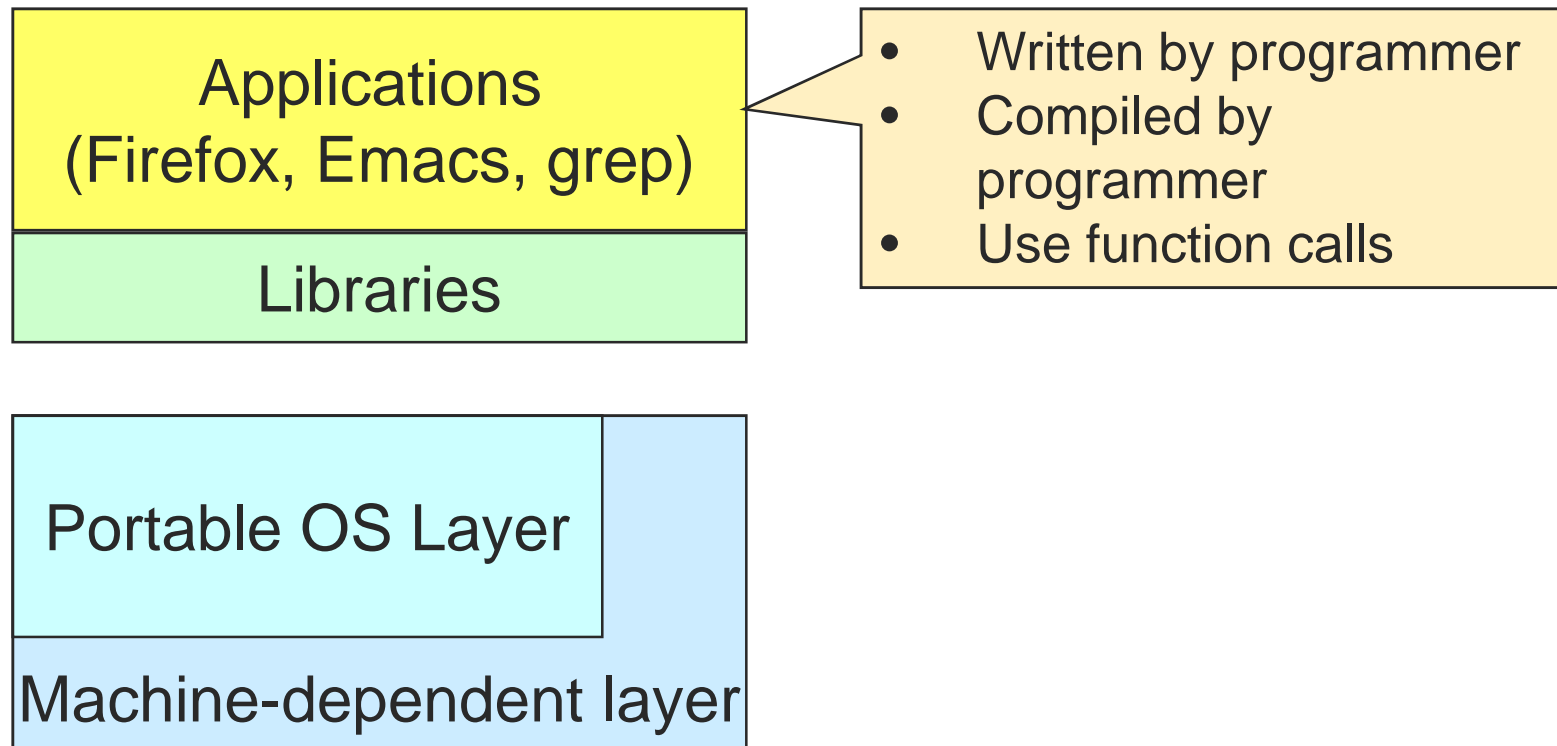| Read/Write | Standard Output | Device Control | File System | Communication |

What is an Operating System?

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

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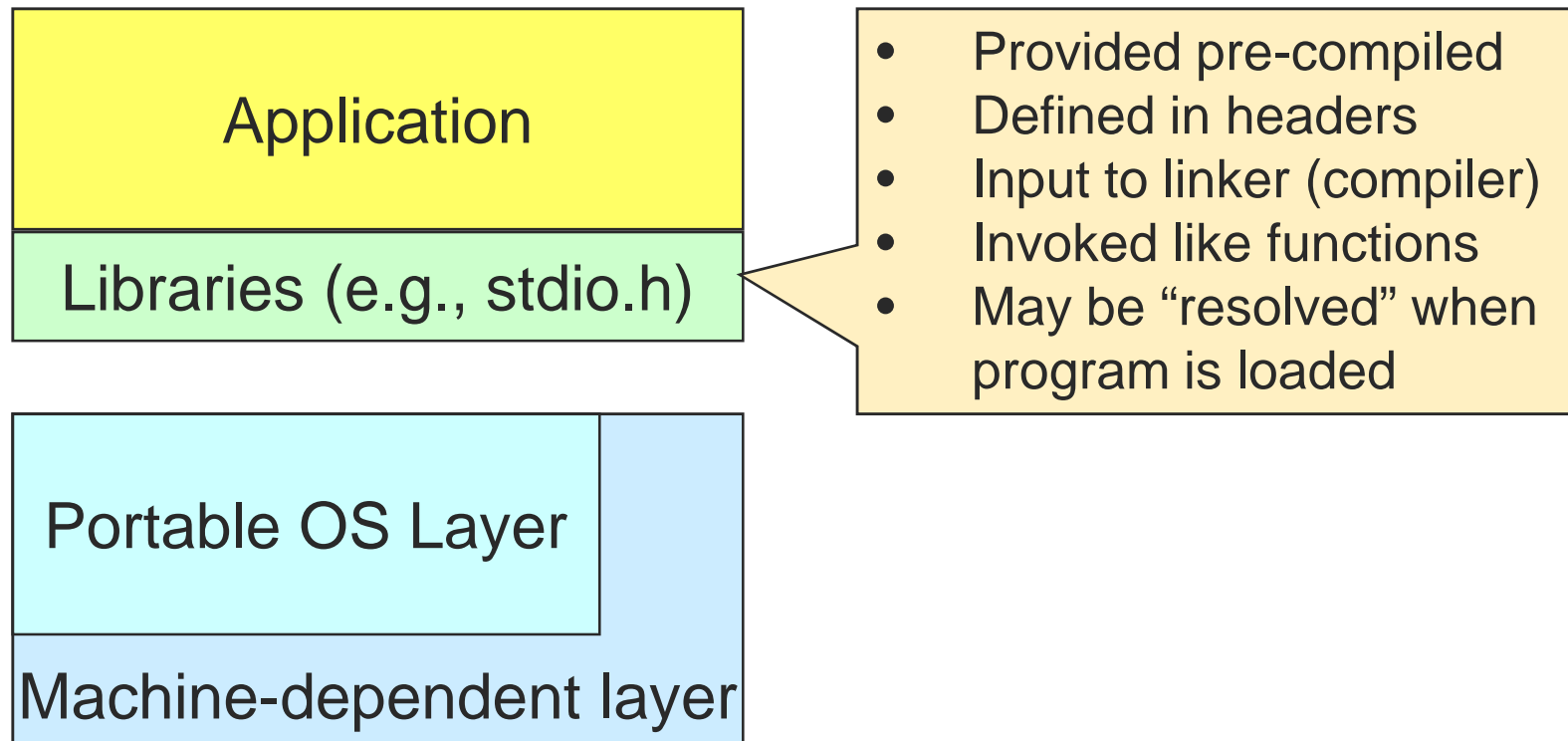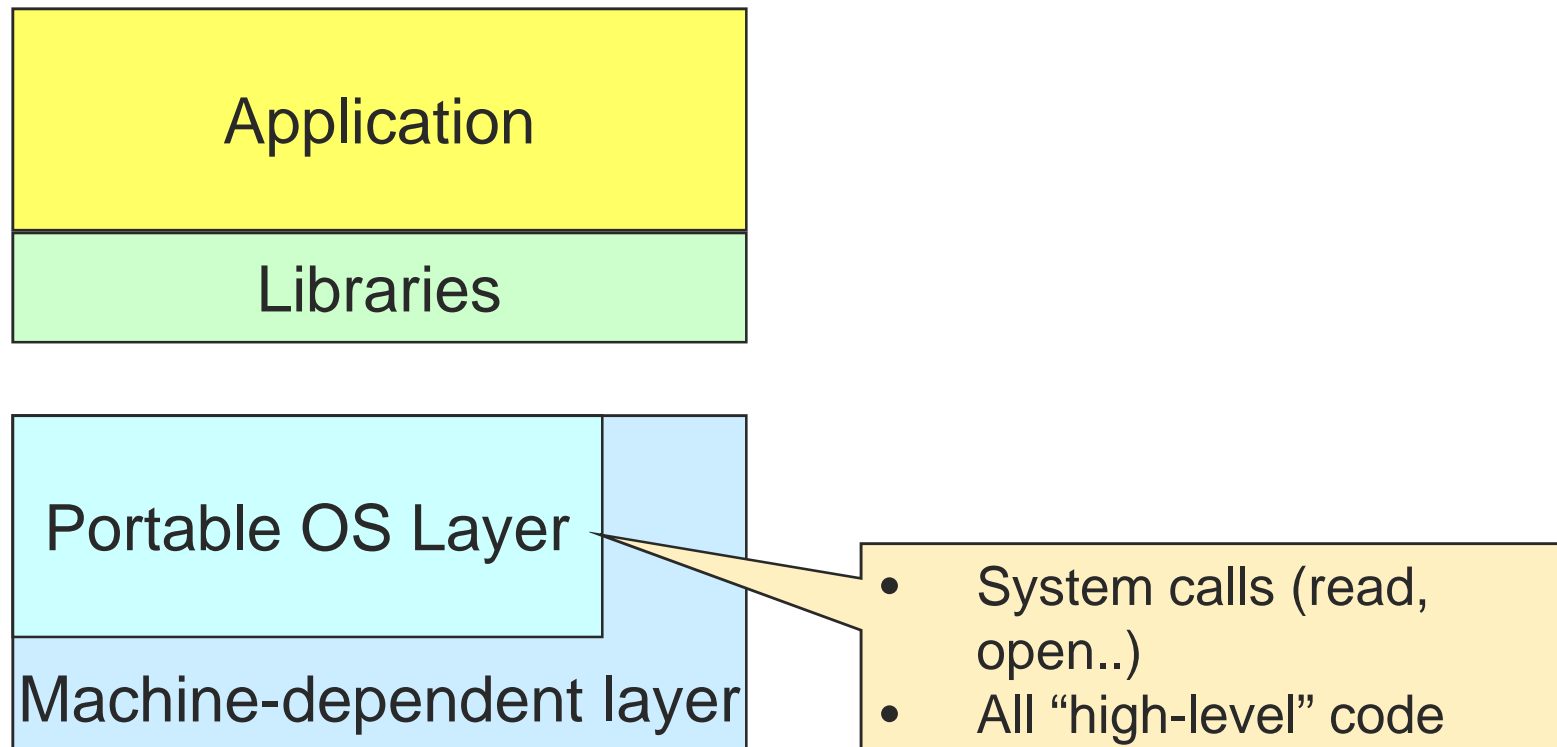 - Each program gets space on the resource

A Peek into Unix

Application
Libraries

User space/level

Kernel space/level

Portable OS Layer
Machine-dependent layer

- **User/kernel modes are supported by hardware**

- **Some systems do not have clear user-kernel boundary**

Application

Applications (Firefox, Emacs, grep)
Libraries

- Written by programmer
- Compiled by programmer
- Use function calls

Portable OS Layer
Machine-dependent layer

Unix: Libraries

Application
Libraries (e.g., stdio.h)

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

Portable OS Layer
Machine-dependent layer

Typical Unix OS Structure

Application
Libraries

Portable OS Layer
Machine-dependent layer

- System calls (read, open..)
- All "high-level" code

Typical Unix OS Structure

Application

Libraries

Portable OS Layer

Machine-dependent layer

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

History of Computer Generations

- Pre-computing generation 1792 - 1871
  - Babbage's "Analytical Engine"
  - Purely mechanical --- but never worked
  - A man before his time
    - When this works, we'll need software!
    - First programming language

History of Computer Generations

- Pre-computing generation 1792 – 1871

- First generation 1945 – 1955
  - Vacuum tubes, relays, plug boards
  - Seconds per operation!
    - Focus on numerical calculations
  - No programming language
    - Everything done using pure machine language or wiring electrical circuits!
  - No operating system
    - Sign up for your time slot!
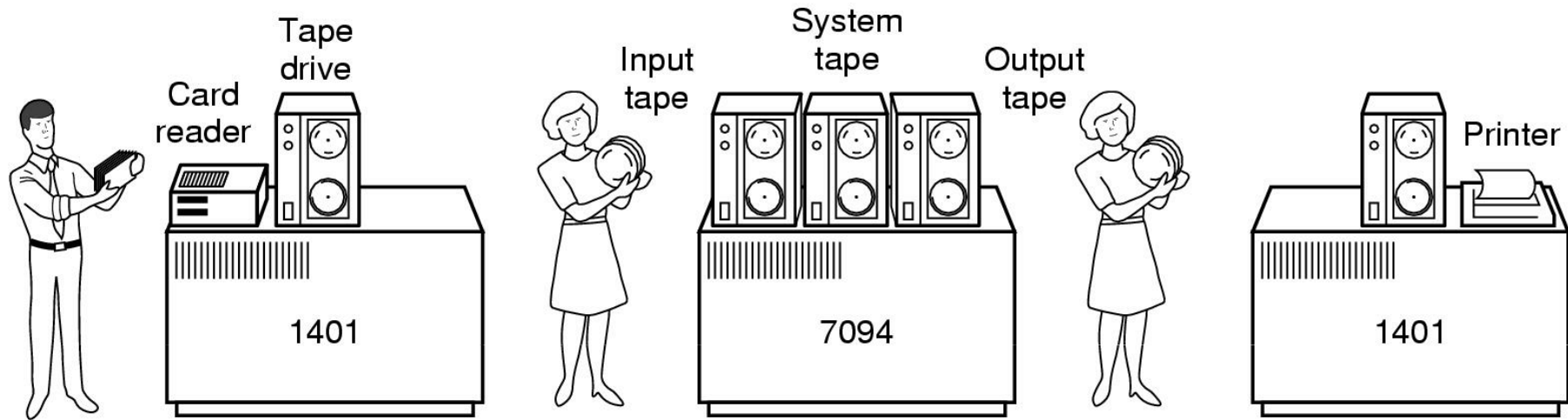 - Progress: Punch cards!

History of Computer Generations

- Pre-computing generation 1792 – 1871

- First generation 1945 – 1955

- Second generation 1955 - 1965
  - Transistors, mainframes
  - Large human component
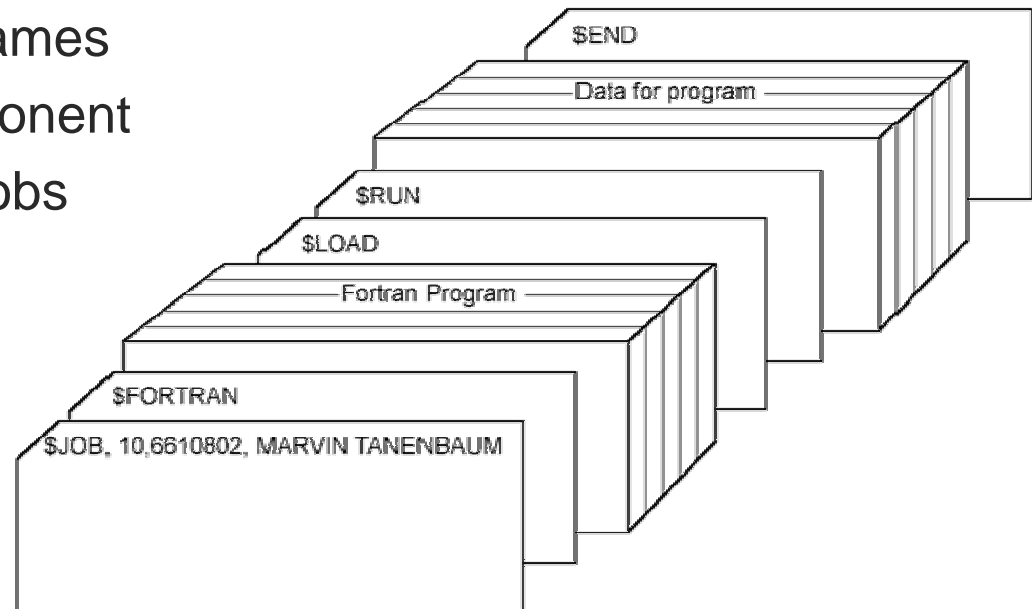
History of Operating Systems

- Early systems
  - bring cards to 1401
  - read cards to tape
  - put tape on 7094 which does computing
  - put tape on 1401 which prints output

History of Computer Generations

- Pre-computing generation 1792 – 1871

- First generation 1945 – 1955

- Second generation 1955 - 1965
  - Transistors, mainframes
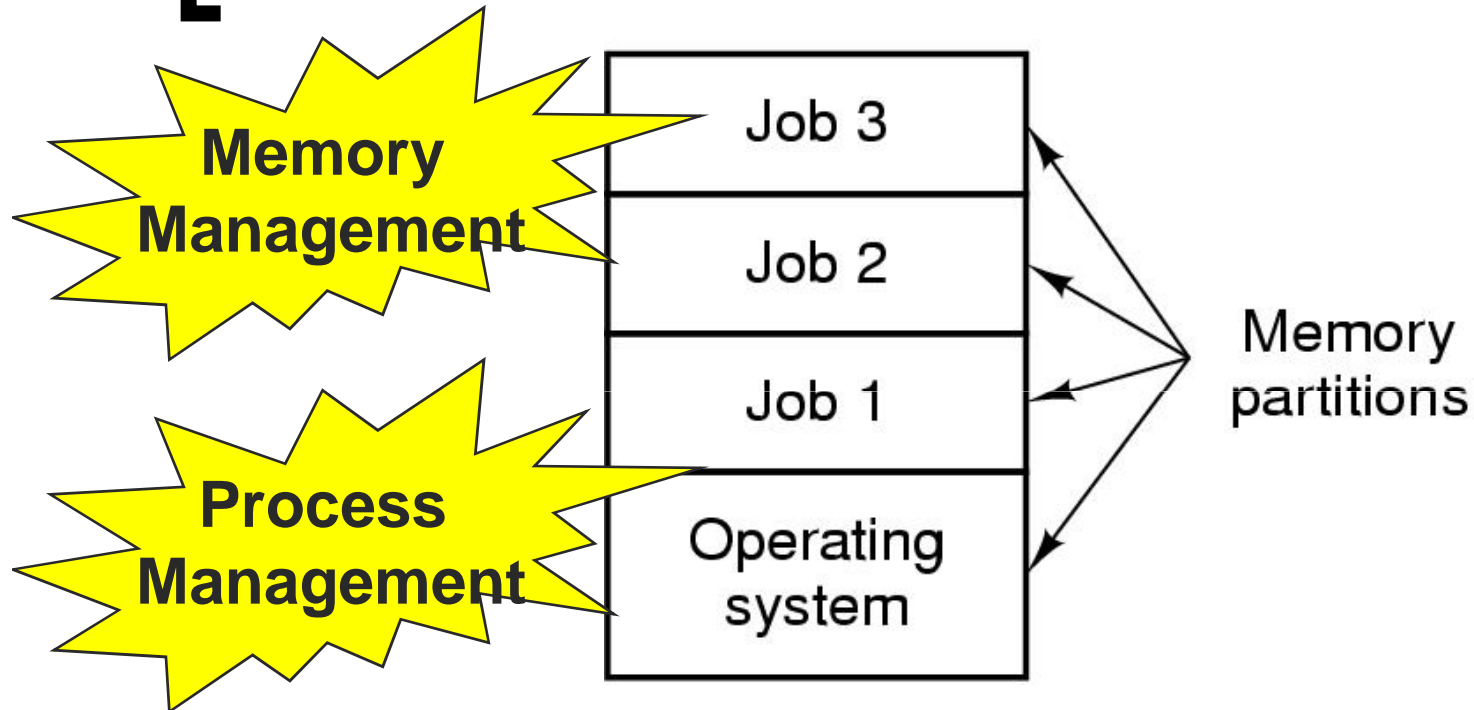 - Large human component
  - Solution: Batched jobs

History of Computer Generations

- Pre-computing generation 1792 – 1871

- First generation 1945 – 1955

- Second generation 1955 – 1965

- Third generation 1965 – 1980
  - Integrated circuits and multiprogramming
  - IBM's New model: all software and OS must work on all platforms
    - A beast!
  - Progress: Multiprogramming
    - Keep the CPU busy

History of Operating Systems

Memory Management

Process Management

Job 3
Job 2
Job 1
Operating system

Memory partitions

- Multiprogramming/timesharing system
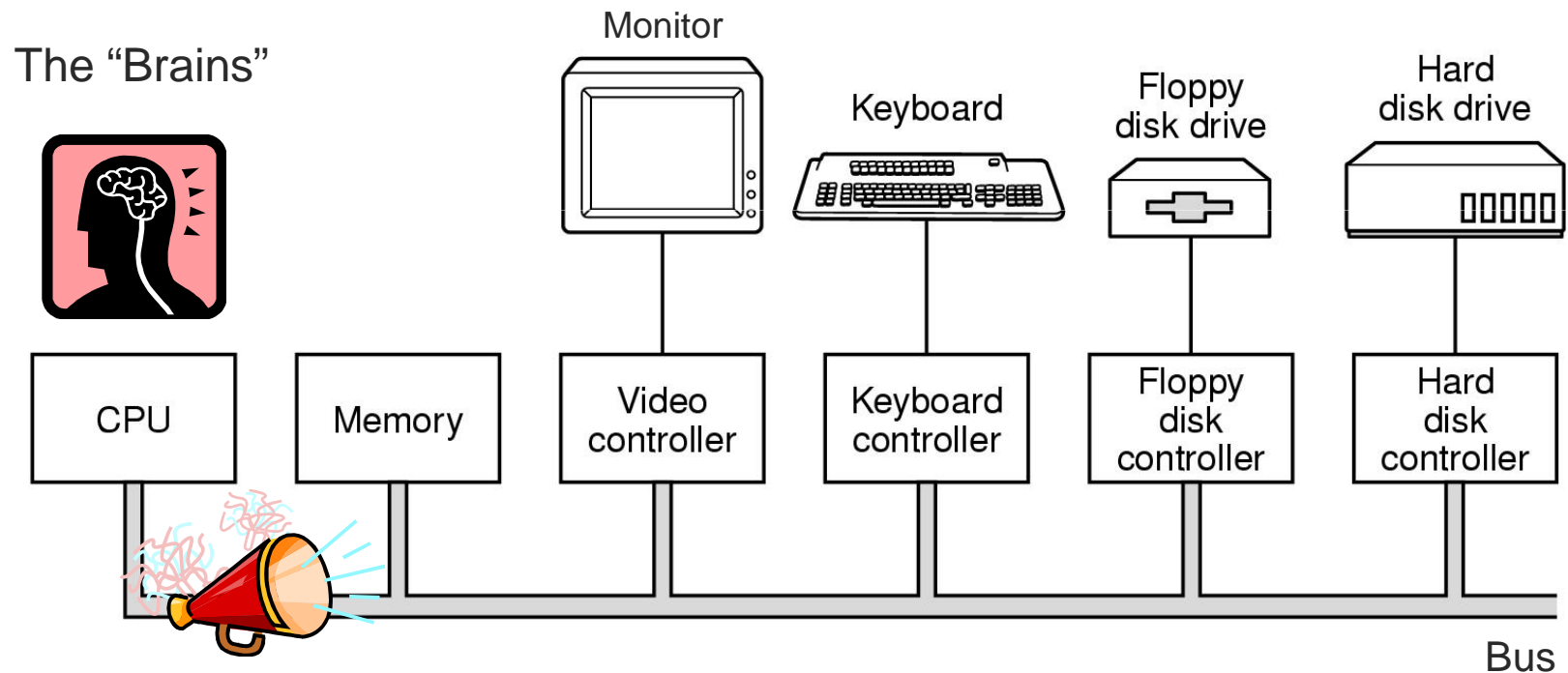 - Three jobs in memory – 3rd generation

History of Computer Generations

- Pre-computing generation 1792 – 1871

- First generation 1945 – 1955

- Second generation 1955 – 1965

- Third generation 1965 – 1980
  - Integrated circuits and multiprogramming
  - IBM's New model: all software and OS must work on all platforms
  - Progress: Multiprogramming and timesharing
  - Progress: Spooling
    - Always have something ready to run
  - MULTICS + minicomputers == UNIX!

History of Computer Generations

- Pre-computing generation 1792 – 1871

- First generation 1945 – 1955

- Second generation 1955 – 1965

- Third generation 1965 – 1980

- Fourth generation 1980 – present
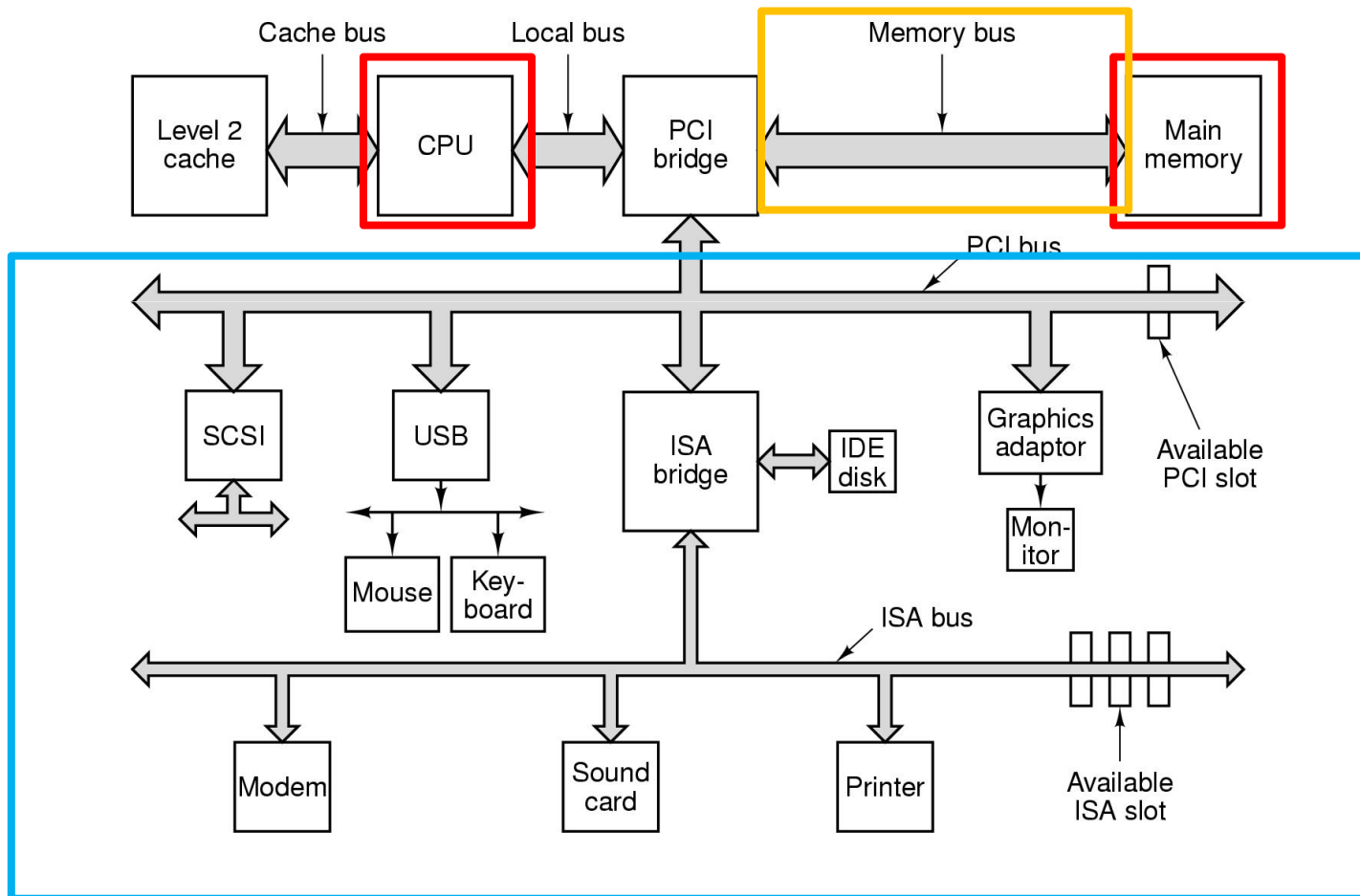 - Personal computers
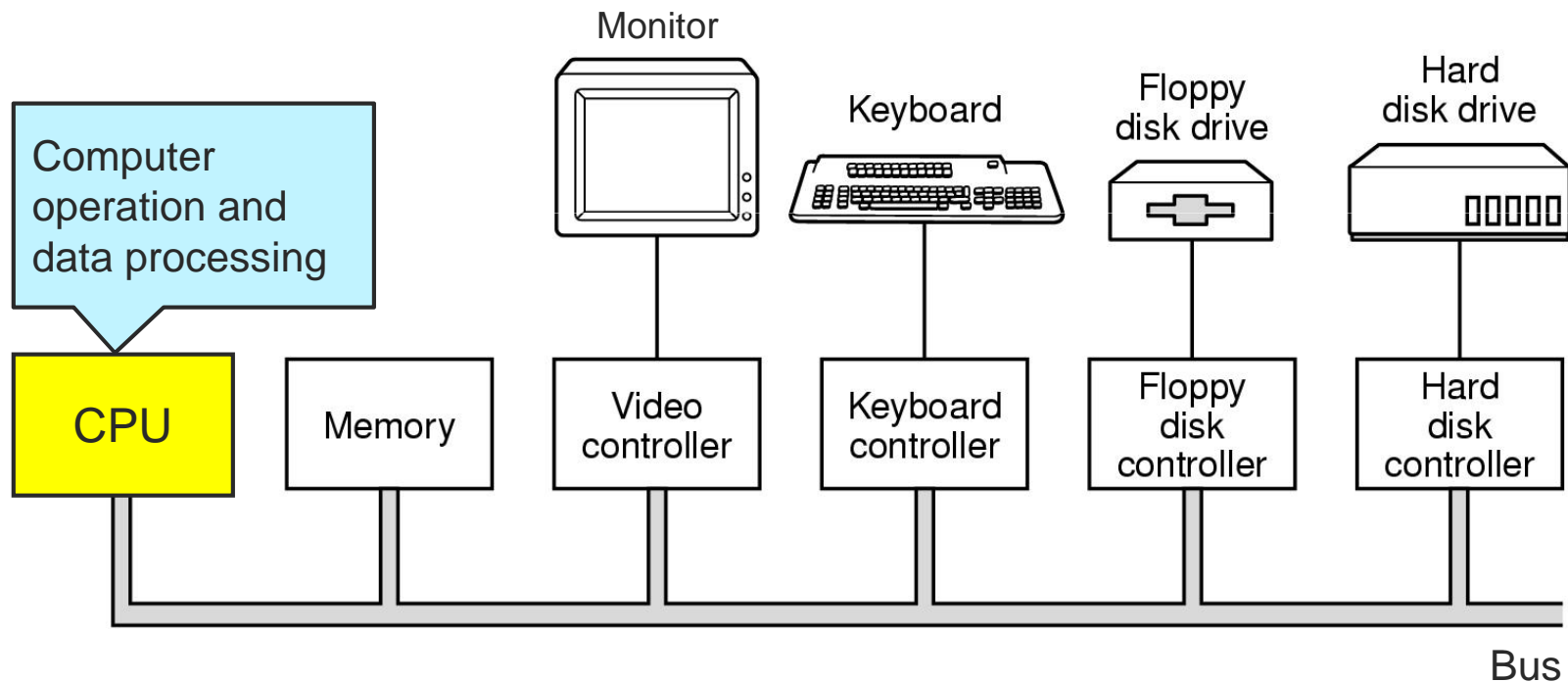 - Multi-processors
  - Phones
  - …

Computer Hardware Review

The "Brains"

Monitor

Keyboard

Floppy disk drive

Hard disk drive

| CPU | Memory | Video controller | Keyboard controller | Floppy disk controller | Hard disk controller |

Bus

- Components of a simple personal computer

Early Pentium system

Computer Hardware Review

■ Components of a simple personal computer

Computer Hardware Review

Communication between CPU, Memory and I/O

Stores data and programs

| CPU | Memory | Video controller | Keyboard controller | Floppy disk controller | Hard disk controller |

Monitor

Keyboard

Floppy disk drive

Hard disk drive

Bus
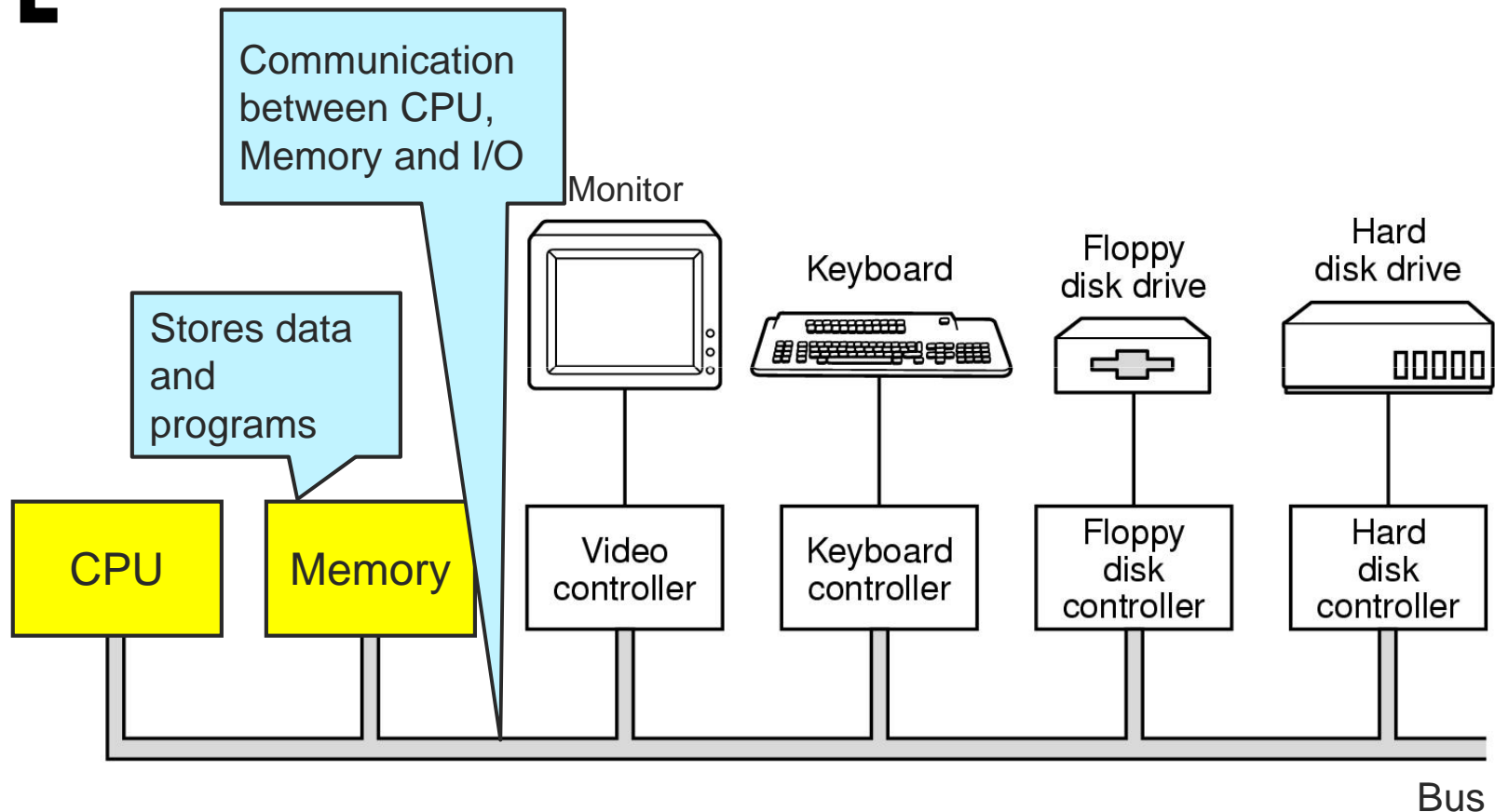
- Components of a simple personal computer

CPU, From CS231

- Fetch instruction from code memory

- Fetch operands from data memory

- Perform operation (and store result)

- (Check interrupt line)

- Go to next instruction

- 'Conventional CPU'
  (Ignore pipeline, optimization complexities)

CPU Registers

- Fetch instruction from code memory

- Fetch operands from data memory

- Perform operation (and store result)

- Go to next instruction

- Note: CPU must maintain certain state
  - Current instructions to fetch (program counter)
  - Location of code memory segment
  - Location of data memory segment

CPU Register Examples

- Hold instruction operands
- Point to start of
  - Code segment
  - Data segment
  - Stack segment
- Point to current position of
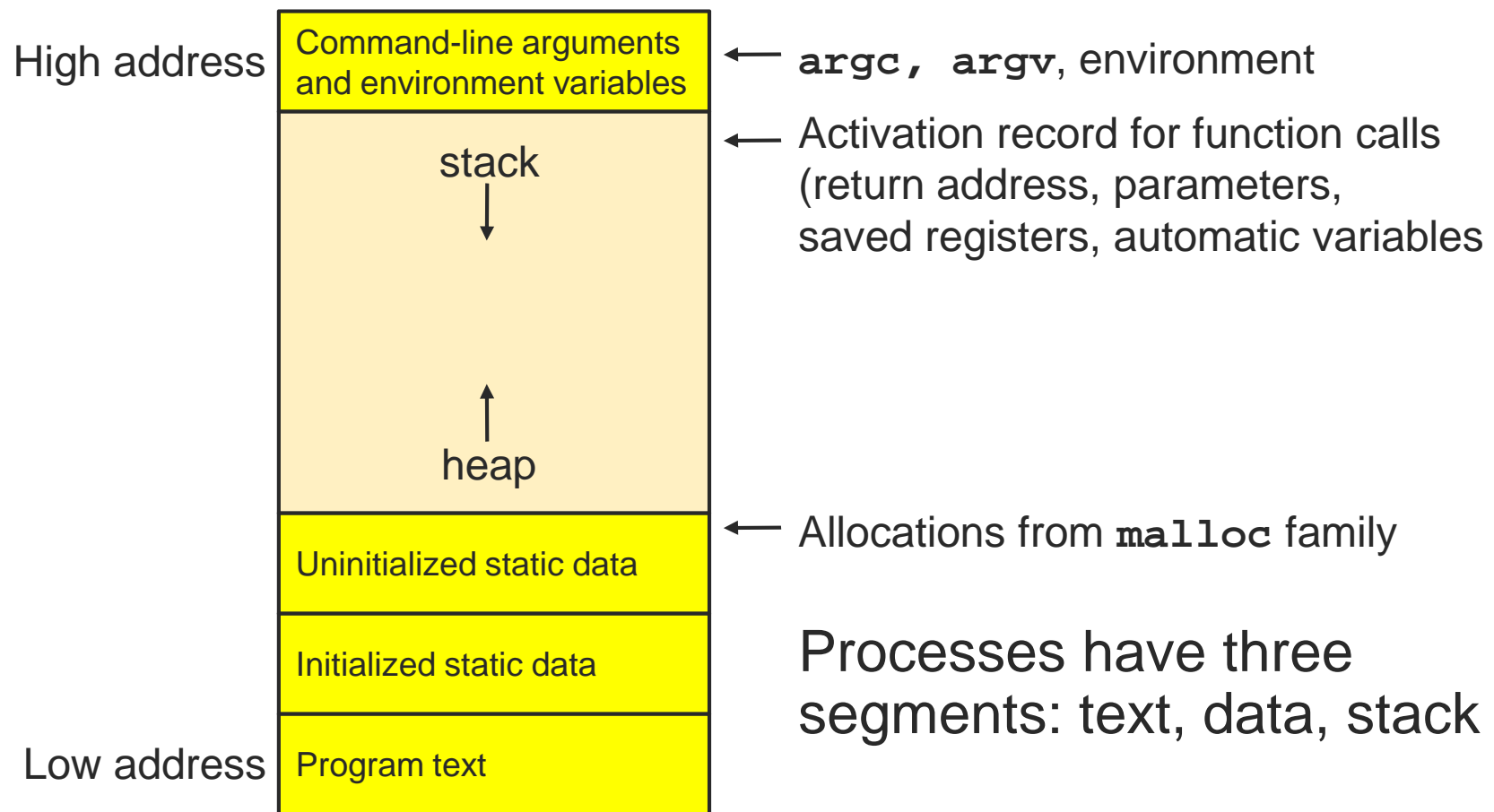 - Instruction pointer
  - Stack pointer

CPU Register Examples

- Hold instruction operands
- Point to start of
  - Code segment
  - Data segment
  - Stack segment
- Point to current position of
  - Instruction pointer
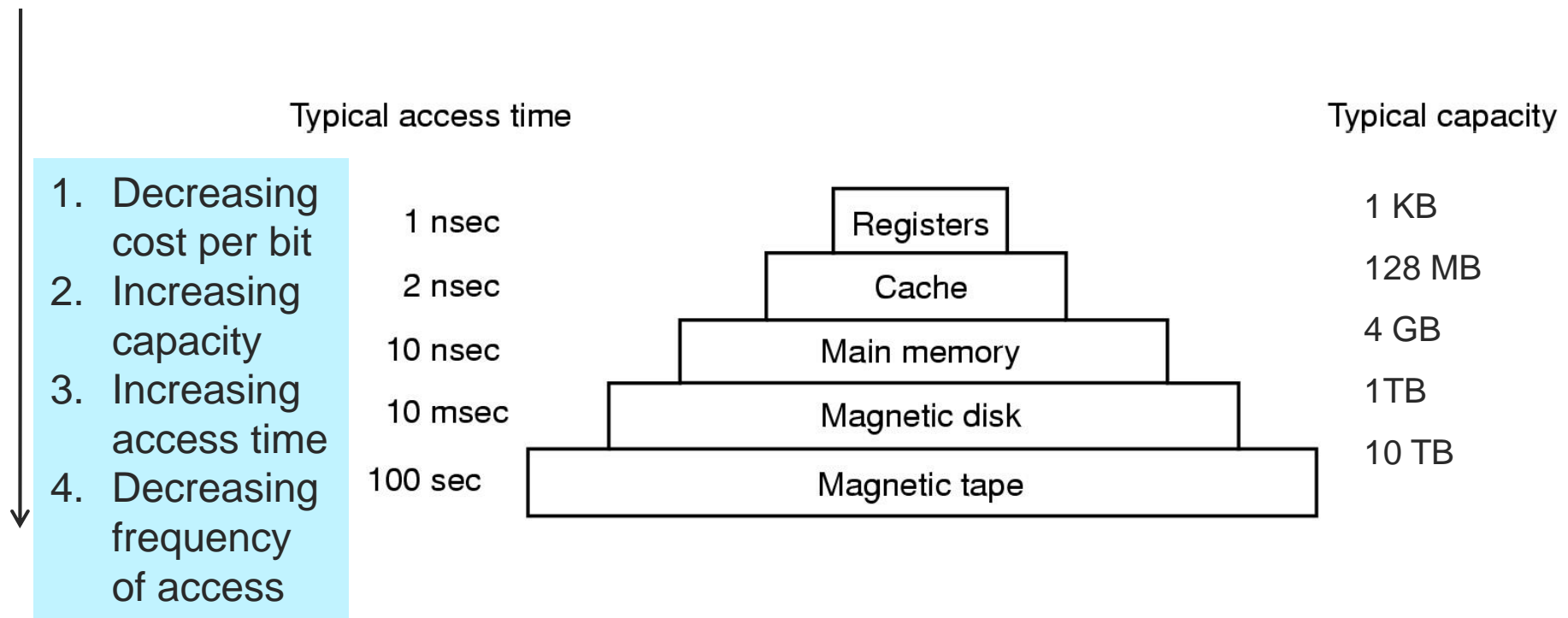 - Stack pointer
    - Why stack?
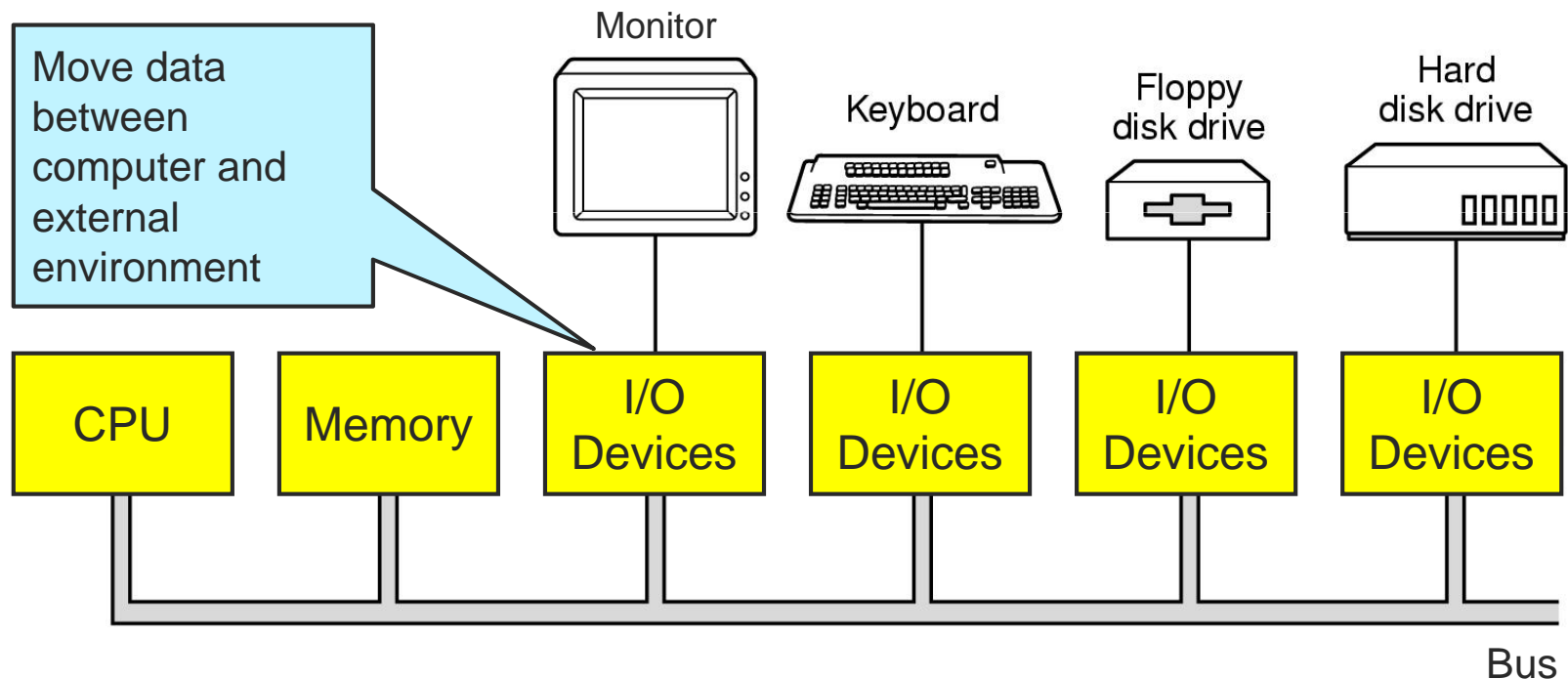
Sample Layout for program image in main memory

High address → Command-line arguments and environment variables ← **argc, argv**, environment

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

↓

↑

heap

Uninitialized static data ← Allocations from **malloc** family

Initialized static data

Low address → Program text

Processes have three segments: text, data, stack

Memory Hierarchy

- Locality of reference

Typical access time

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

Typical access time		Typical capacity
1 nsec	Registers	1 KB
2 nsec	Cache	128 MB
10 nsec	Main memory	4 GB
10 msec	Magnetic disk	1TB
100 sec	Magnetic tape	10 TB

Computer Hardware Review

Move data between computer and external environment

| Monitor | Keyboard | Floppy disk drive | Hard disk drive |

| CPU | Memory | I/O Devices | I/O Devices | I/O Devices | I/O Devices |

Bus

- Components of a simple personal computer

I/O Device Access

- **Systems Calls**
  - Application makes a system call
  - Kernel translates to specific driver
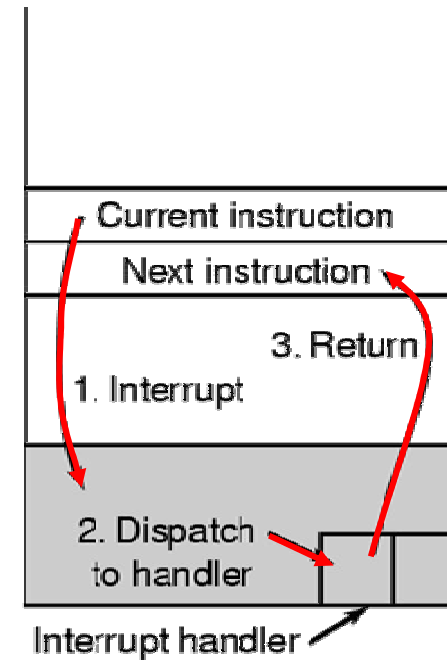 - Driver starts I/O
  - Polls device for completion

- **Interrupts**
  - Application starts device
  - Asks for an interrupt upon completion
  - OS blocks application
  - Device controller generates interrupt

I/O Interrupt Mechanism
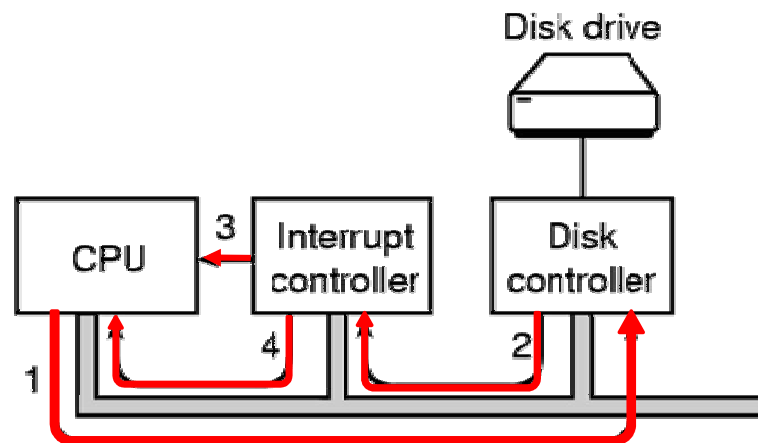
(a)   (b)

1. Application writes into device registers, Controller starts device
2. When done, device controller signals interrupt controller
3. Interrupt controller asserts pin on CPU
4. Interrupt controller puts I/O device number on bus to CPU

Operating System Concepts

- **Process**
  - An executable instance of a program
  - Only one process can use the CPU at a time

A   Z
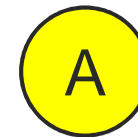
Operating System Concepts

- **Context Switching**
  - Now I have N processes and M processors?
  - How would you switch CPU execution from one process to another?
  - What are the costs involved?

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	1 ns (1 GHz)	2 s
Memory access	15 ns	30 s
Context switch	5,000 ns (5 micros)	167 m
Disk access	7,000,000 ns (7 ms)	162 days
System quanta	100,000,000 (100 ms)	6.3 years

Operating System Concepts

- **Shared resources**
  - Now I have B KB of memory, but need 2B KB
  - Now I have N processes trying to access the disk
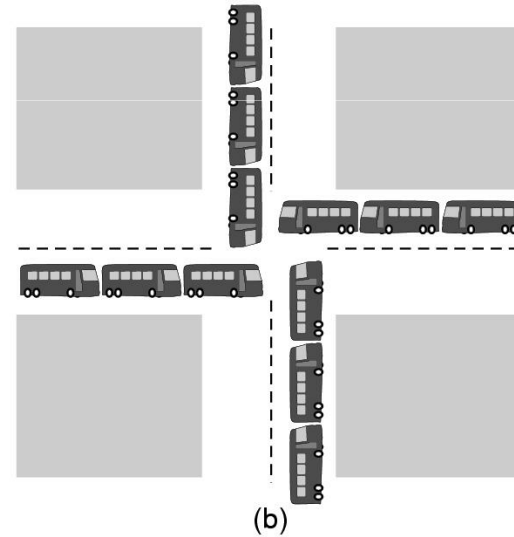 - How would you control access to resources?
  - What are the challenges?

Operating System Concepts

- Shared resources
  - What are the challenges?
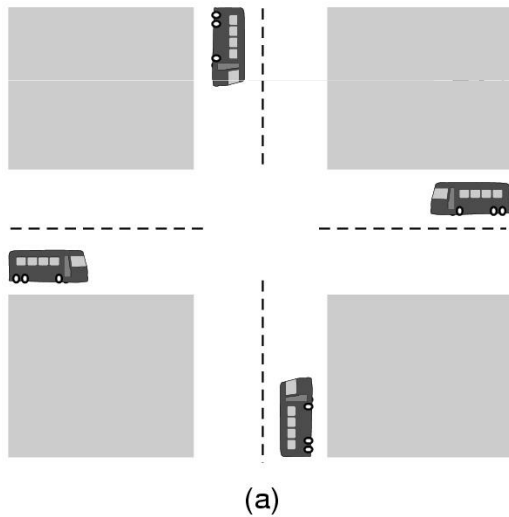
(a) A potential deadlock
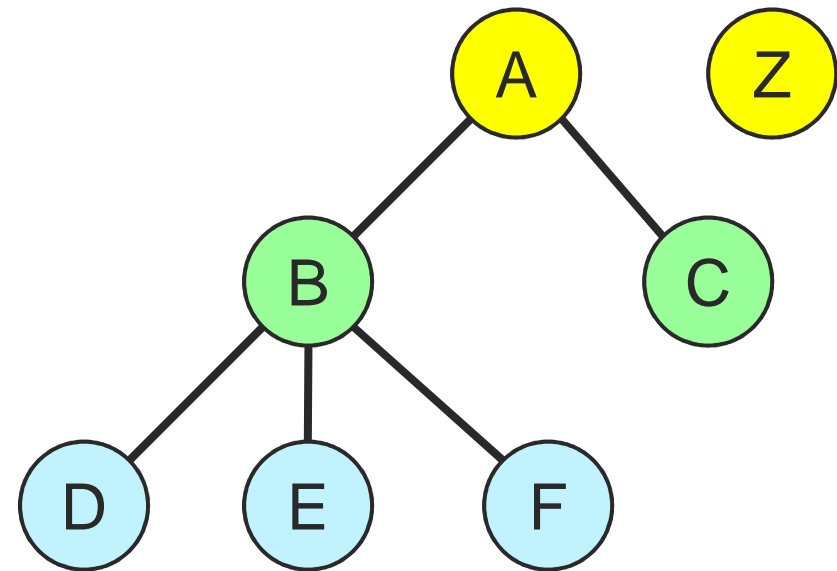
(b) An actual deadlock

Operating System Concepts

- ## Process
  - An executable instance of a program
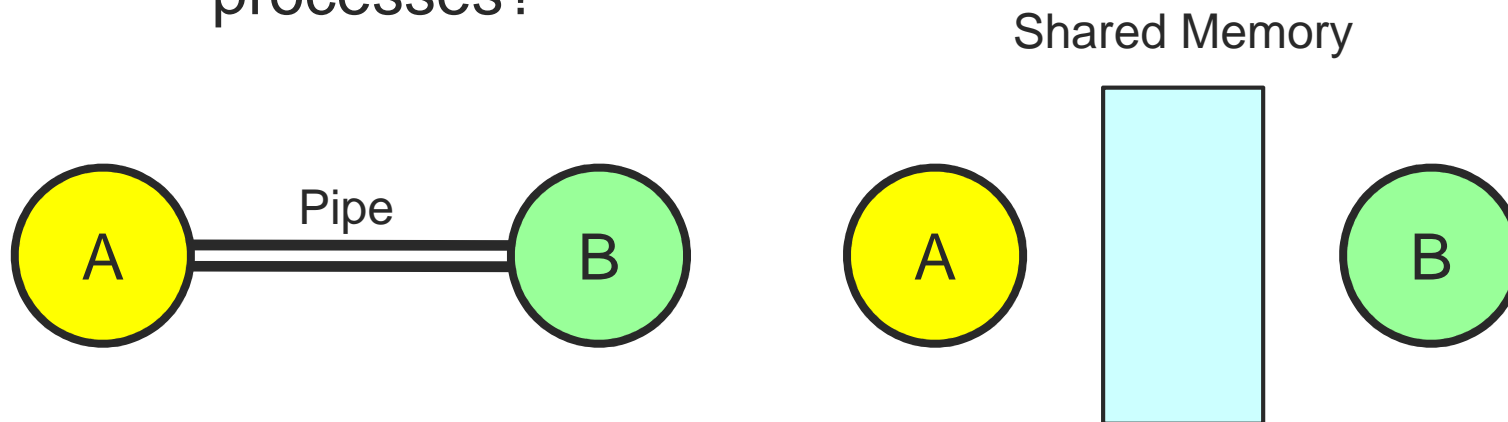 - Only one process can use the CPU at a time
- ## A process tree
  - A created two child processes, B and C
  - B created three child processes, D, E, and F

Operating System Concepts

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

Shared Memory

Pipe

A — B

A □ B

Summary

- Resource Manager
- Hardware independence
- Virtual Machine Interface
- POSIX
- Concurrency & Deadlock