CMPT 300

Introduction to Operating Systems

Contact Information

- Instructor: Arrvindh Shriraman
- Office: Surrey: Room 4172
  - Hours: Tue/Thu 3:00-4:00 pm (or drop email)
    - Email: ashriram@cs.sfu.ca
- TA: Arun Bharadwaj
    - (Email: arun_bharadwaj@sfu.ca)
    - Hours: Tue/Thu 3:00-4:00 pm

Meet your instructor

- Joined SFU Faculty in January 2011

- Areas of research
  - Software for Multicore processors
  - Energy-Smart OSs/Apps for SmartPhones
  - Energy-Smart Datacentric Systems

- Interactive! Ask lot of questions
- Learn by hacking the real linux kernel.

Web-site

- All the information discussed today and more can always be found on the class web-site
- Class web site go to http://www.cs.sfu.ca/~ashriram/courses/CS300/
- Google groups : cs300@googlegroups.com

Topics

- History, Evolution, and Philosophies
- The User's View of Operating System
- Tasking and Processes
- Inter-process Communication, Concurrency Control and Resource Allocation
- Scheduling and Dispatch
- Physical and Virtual Memory Organization
- File Systems and I/O
- Security and Protection

What is an OS ?

Hardware and Software

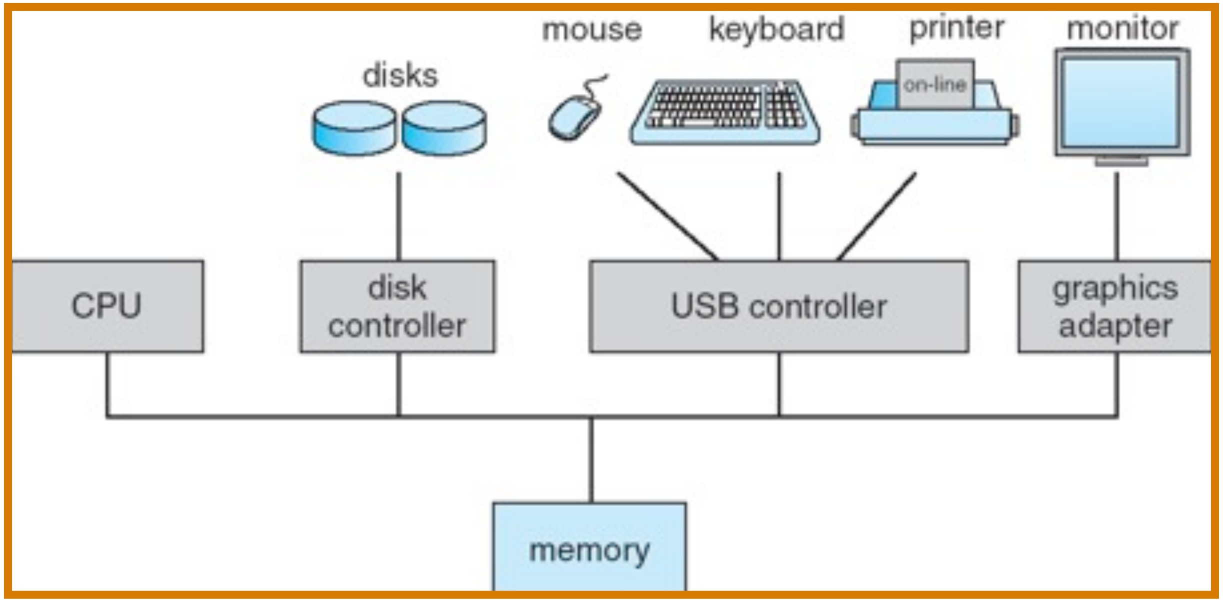
- A **computer** is a machine designed to perform operations specified with a set of instructions called a **program.**

- **Hardware** refers to the computer equipment.
  - keyboard, mouse, terminal, hard disk, printer, CPU
- **Software** refers to the programs that describe the steps we want the computer to perform.

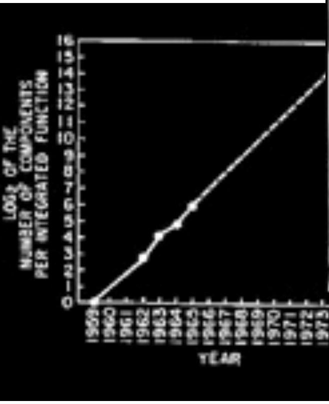
- **Operating system :** software that
  - manages the hardware
  - shares the hardware between applications

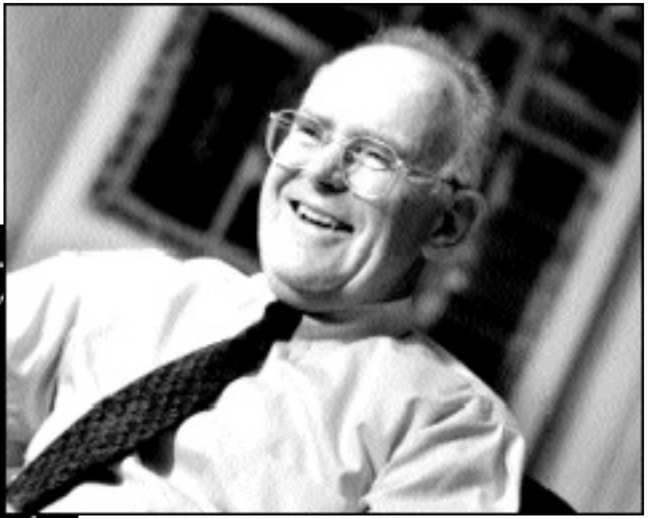
Computer Hardware

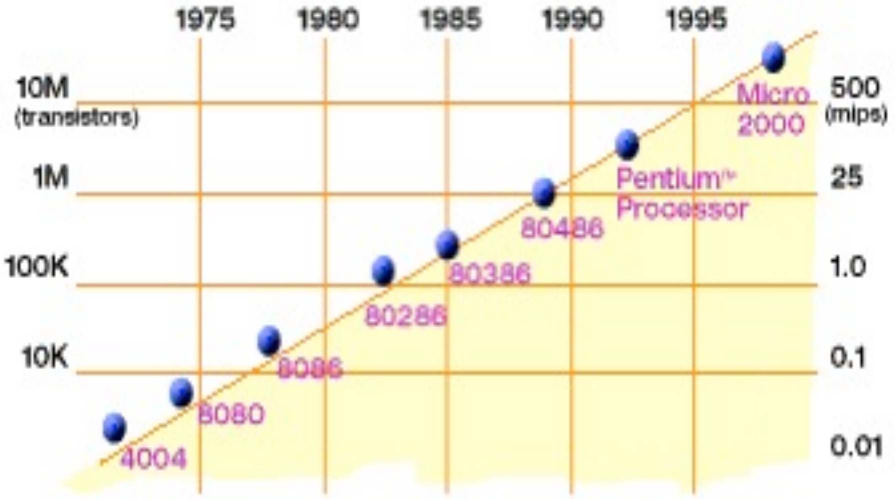
- Computer-system operation
  - One or more CPUs, device controllers connected by bus
  - Concurrent execution of CPUs and devices
  - Shared Memory



Technology Trends







2X transistors/Chip Every 1.5 years

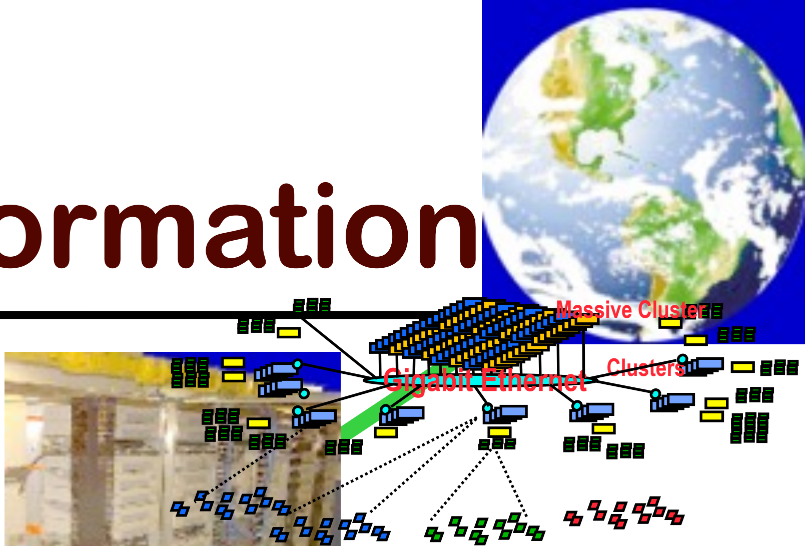
Called "Moore's Law"

Gordon Moore (co-founder of Intel) predicted in 1965 that the transistor density of semiconductor chips would double roughly every 18 months.

Microprocessors have become smaller, denser, and more powerful.

Societal Scale Information Systems

The world is a parallel system
- Microprocessors in everything

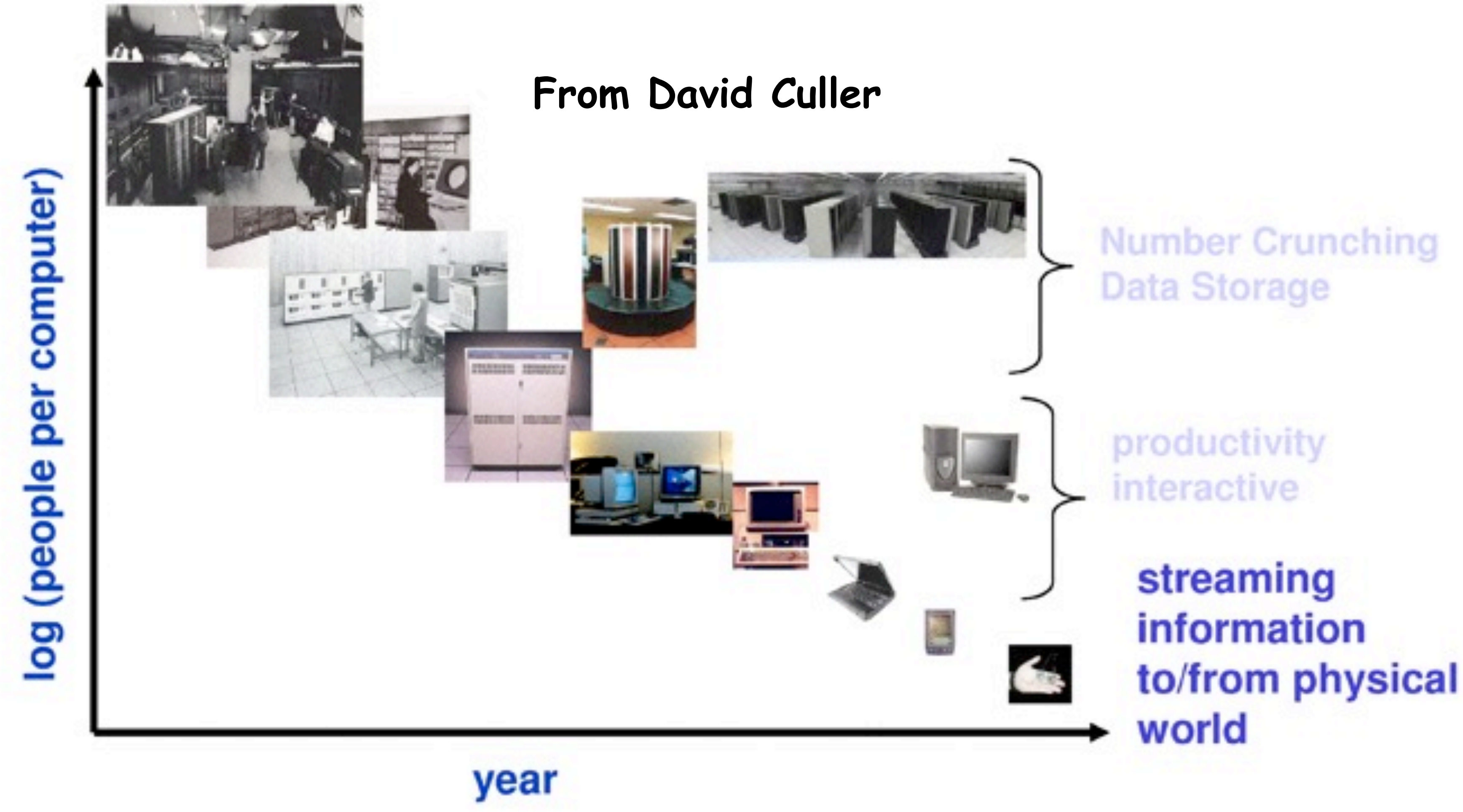


Scalable, Reliable
Secure Services

Databases
Information Collection
Remote Storage
Online Games
Commerce
…

MEMS for
Sensor Nets

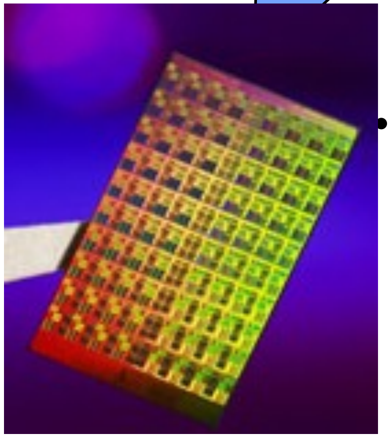
People to Computer Ratio

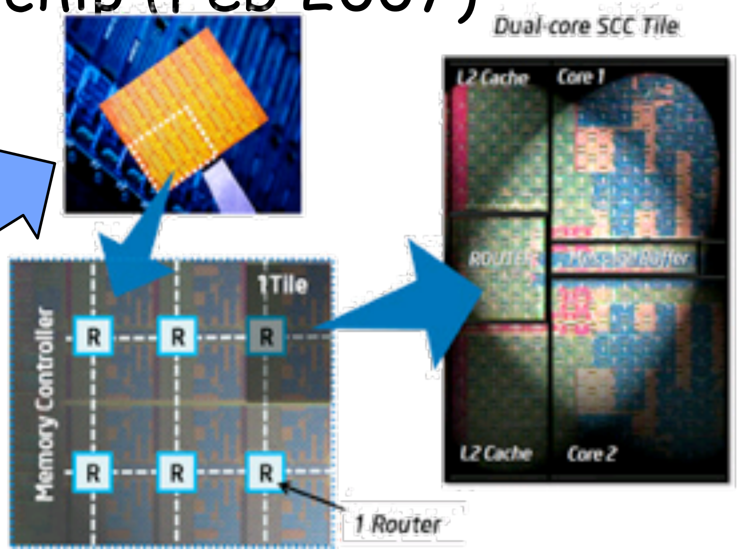


From David Culler

The World is parallel

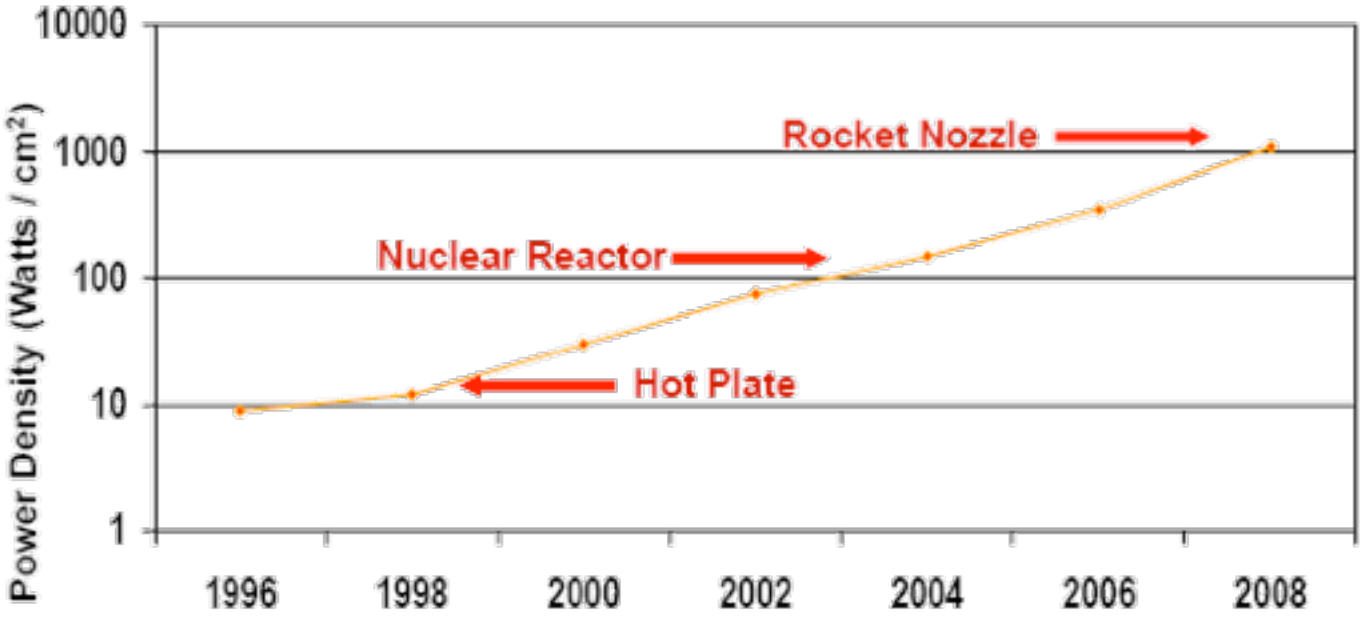
- Intel 80-core multicore chip (Feb 2007)
  - 80 simple cores
  - 100 million transistors
  - 65nm feature size
- Intel Cloud Chip Computer (August 2010)
  - 24 "tiles" with two cores/tile
  - 4 DDR3 memory controllers
  -





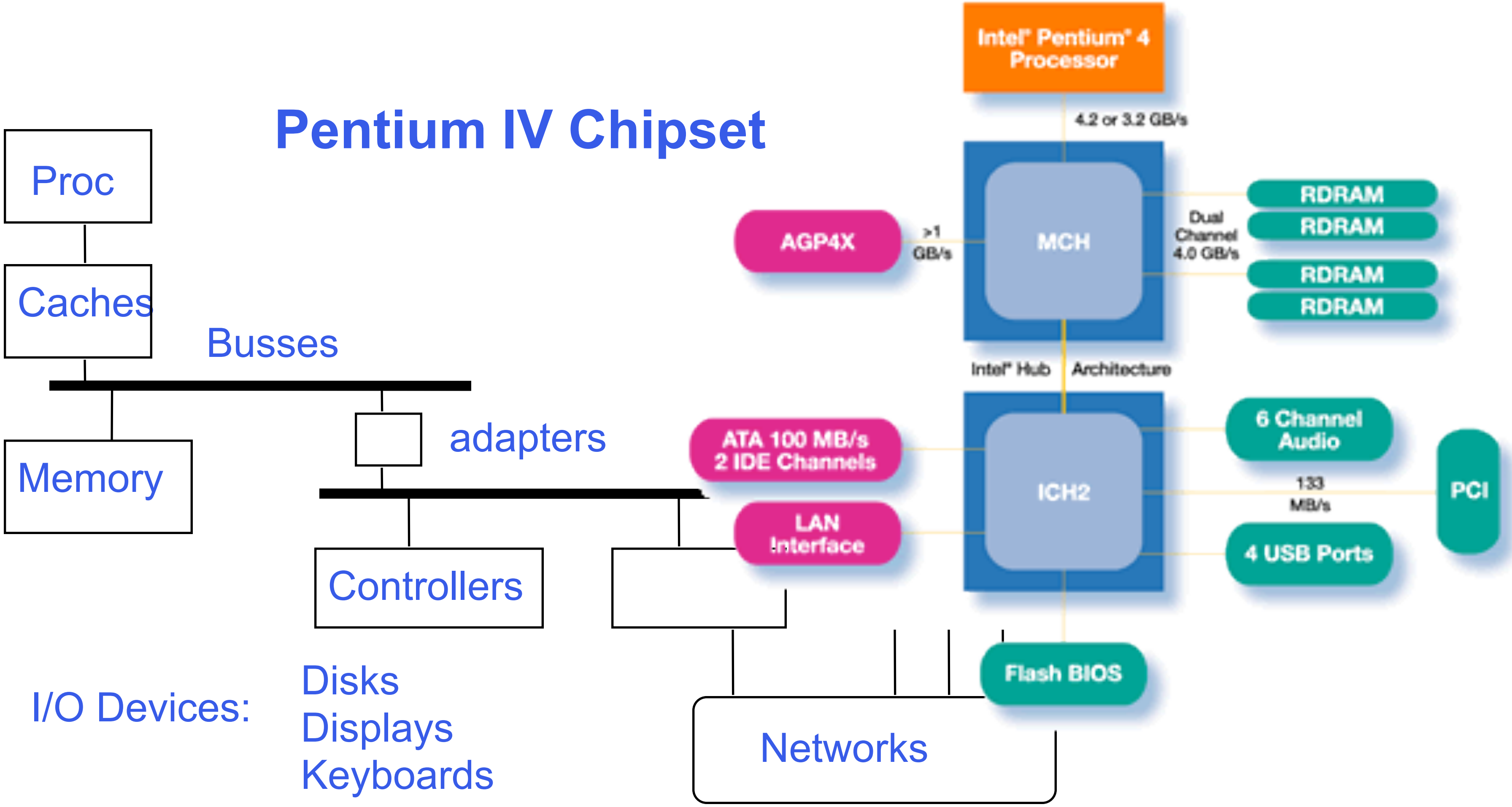
- "ManyCore" refers to many processors/chip
  - 64? 128? Hard to say exact boundary
- How to program these?
  - Use 2 CPUs for video/audio
  - Use 1 for word processor, 1 for browser, 76 for virus checking???

Power Challenge



Power Density Becomes Too High to Cool Chips Inexpensively

Computers are Complex



Pentium IV Chipset

Proc

Caches

Busses

Memory

adapters

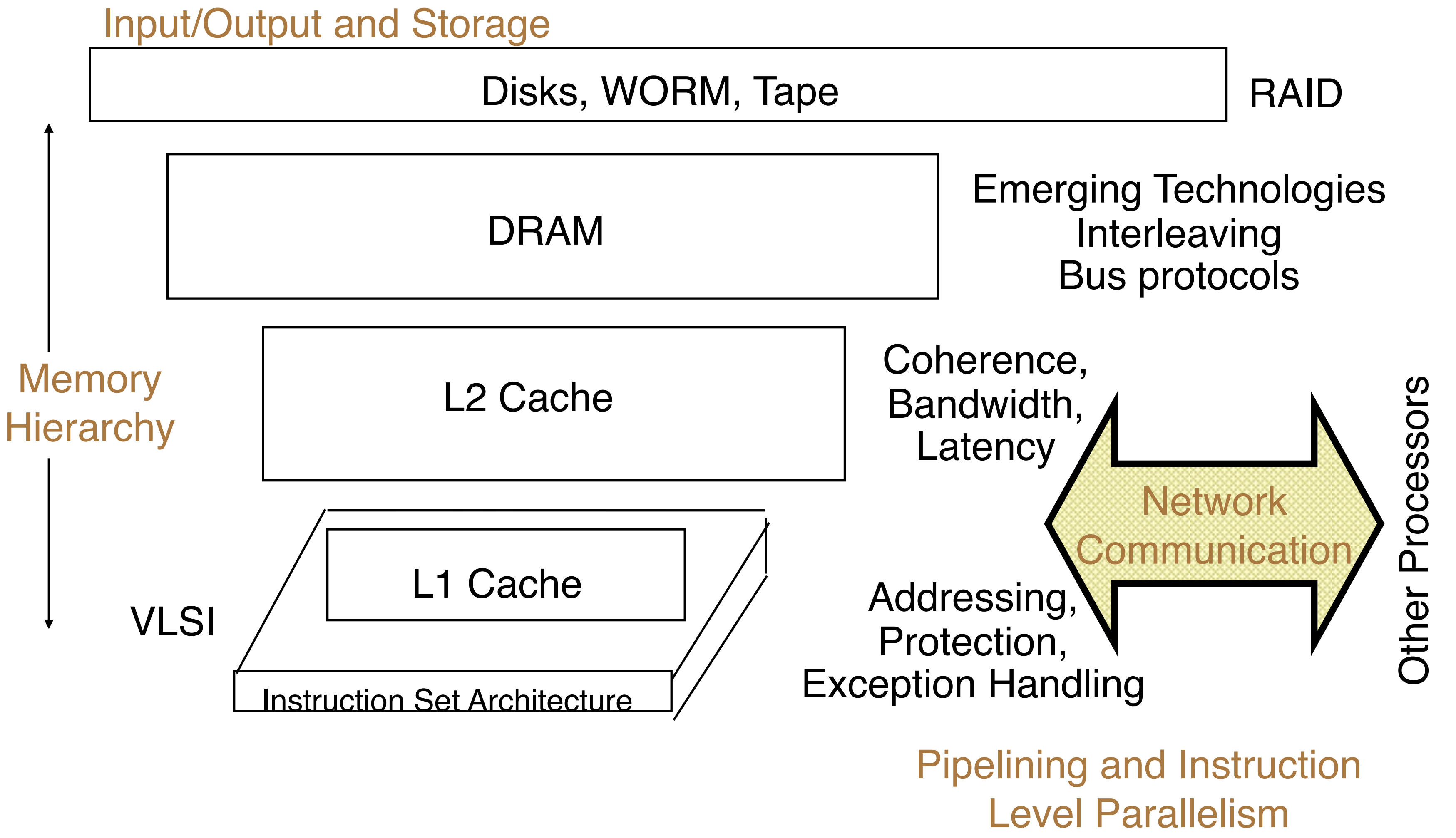
Controllers

I/O Devices:

Disks
Displays
Keyboards

Networks

Sample of Computer Architecture



Input/Output and Storage

Disks, WORM, Tape

RAID

DRAM

Emerging Technologies
Interleaving
Bus protocols

L2 Cache

Coherence,
Bandwidth,
Latency

Memory
Hierarchy

Network
Communication

Other Processors

L1 Cache

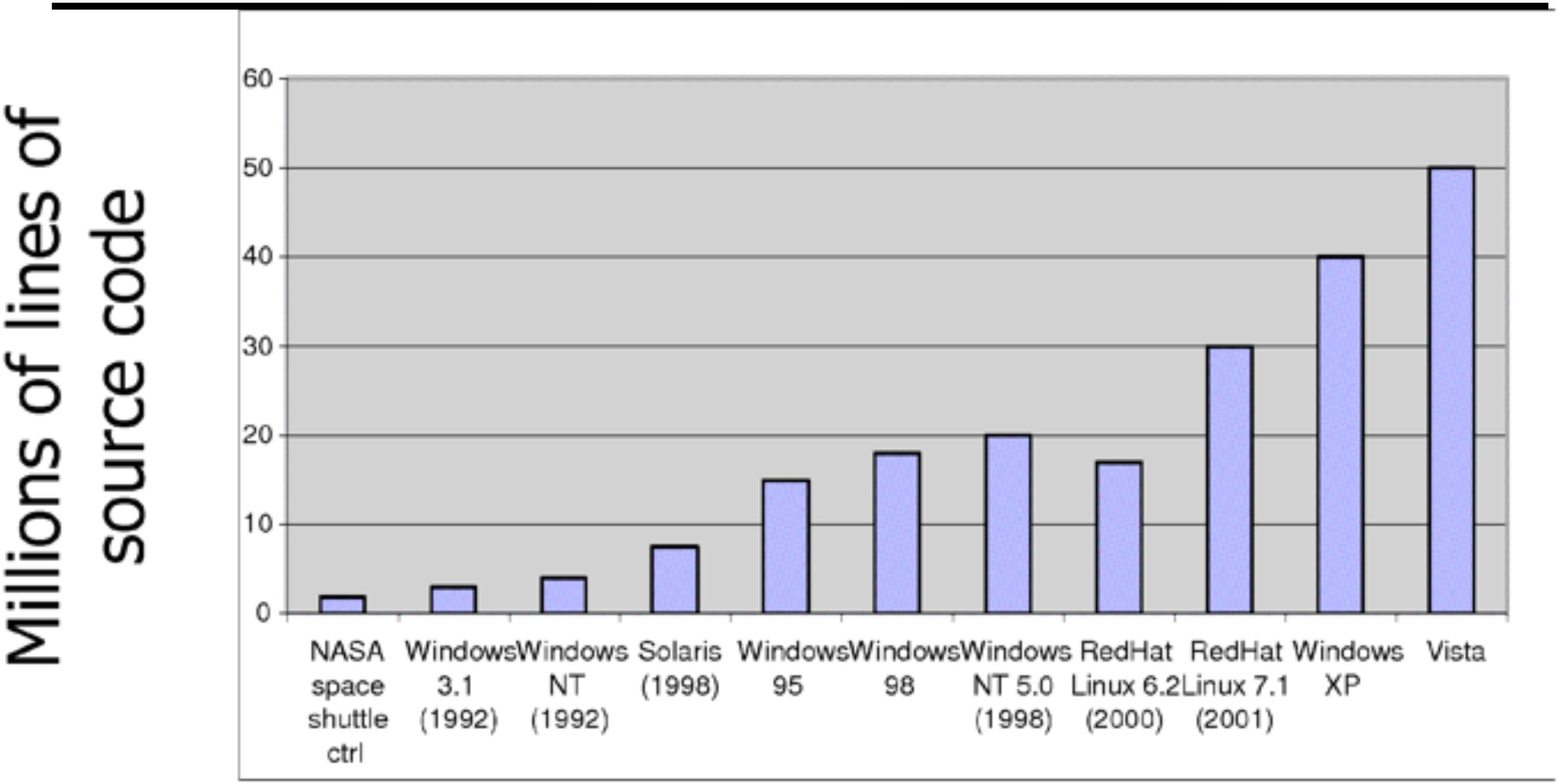
VLSI

Instruction Set Architecture

Addressing,
Protection,
Exception Handling

Pipelining and Instruction
Level Parallelism

Increasing Software Complexity



From MIT’s 6.033 course

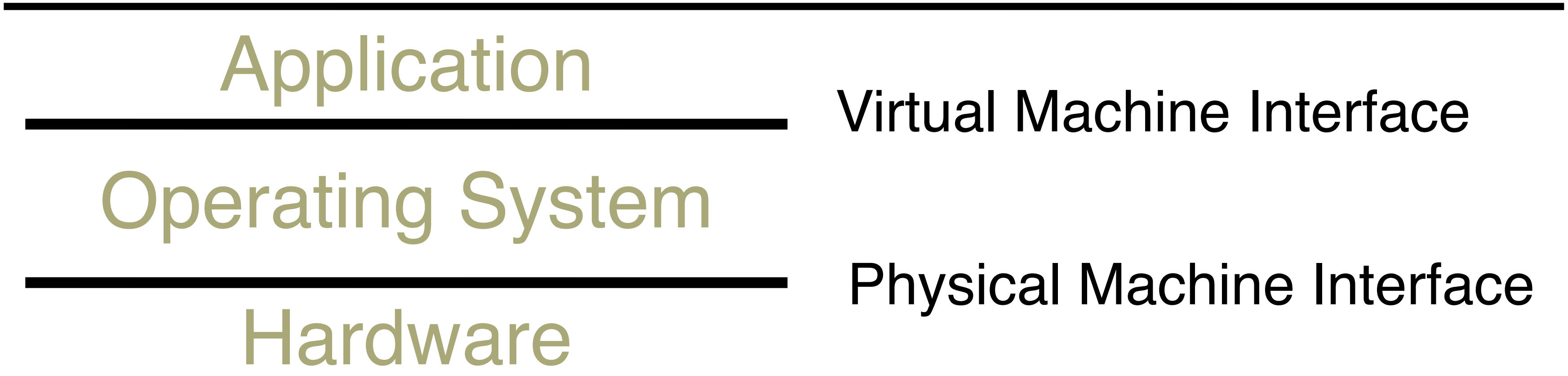
Mars Rover Program

- Pathfinder hardware limitations/complexity:
  - 20Mhz processor, 128MB of DRAM, VxWorks OS
  - cameras, scientific instruments, batteries, solar panels, and locomotion equipment
- Can't hit reset button very easily!
  - Must reboot itself
- Individual Programs must not interfere
  - Better not crash antenna positioning software!
- Further, all software may crash occasionally
  - Automatic restart with diagnostics sent to Earth
  - Periodic checkpoint of results saved?
- Certain functions critical:
  - Need to stop before hitting something

How do we tame complexity?

- Every piece of computer hardware different
  - Different CPU (Pentium, PowerPC, ColdFire, ARM, MIPS….)
  - Different amounts of memory, disk, …
  - Different types of devices
    - Mice, Keyboards, Sensors, Fingerprint readers, touch screen
  - Different networking environment
    - Cable, Wireless, Firewalls,…
- Questions:
  - Programmer need to write a single program that performs many independent activities?
  - Very program need to be moded for every hardware?
  - Faulty program crash everything?
  - Program have access to all hardware?

Virtual Machine Abstraction



Application

Virtual Machine Interface

Operating System

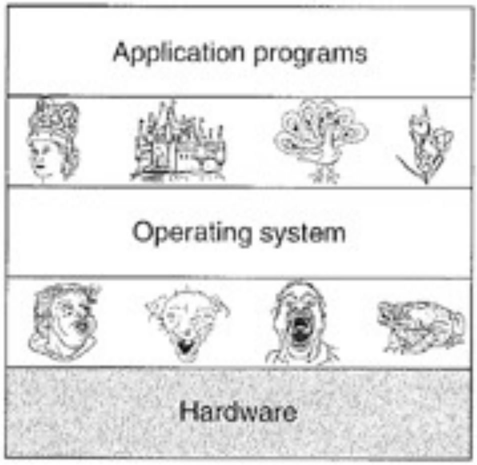
Physical Machine Interface

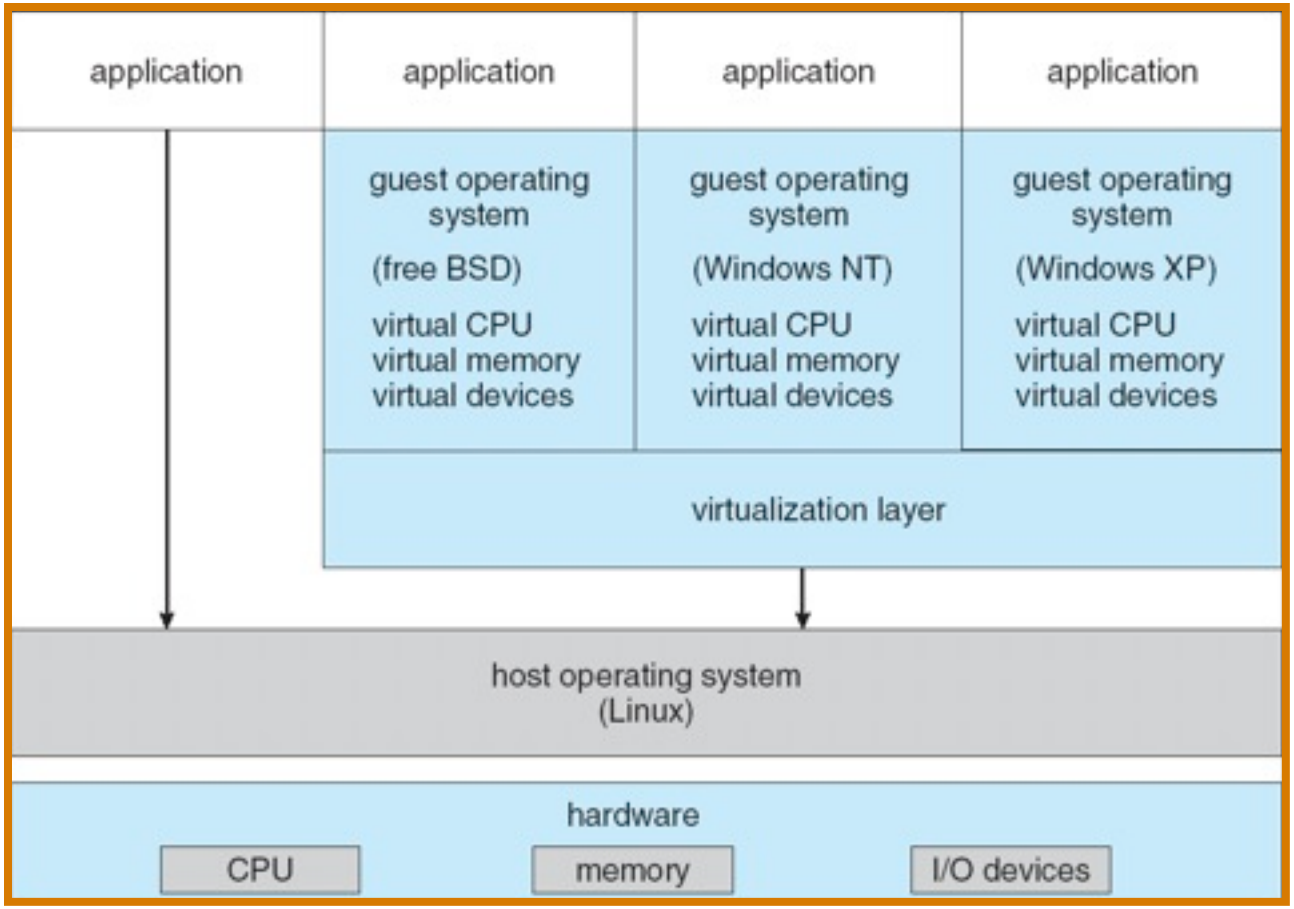
Hardware

- Software Engineering Problem:
  - Turn HW/SW quirks ⇒ what programmers want
  - Optimize for convenience, utilization, security, etc…
- For any OS area (e.g. sched, virtual memory, network):
  - What’s the hardware interface? (physical reality)
  - What’s the application interface? (nicer abstraction)

Virtual Machines

- Software emulation of an abstract machine
  - Make it look like hardware has features you want
  - Programs from one hardware & OS on another one
- Programming simplicity
  - Each process thinks it has all memory/CPU time
  - Different Devices appear to have same interface
  - Device Interfaces more powerful than raw hardware
    - Bitmapped display ⇒ windowing system
    - Ethernet card ⇒ networking (TCP/IP)
- Fault Isolation
  - Processes do not impact other processes
  - Bugs cannot crash whole machine




application
application
application
application
guest operating system
(free BSD)
virtual CPU
virtual memory
virtual devices
guest operating system
(Windows NT)
virtual CPU
virtual memory
virtual devices
guest operating system
(Windows XP)
virtual CPU
virtual memory
virtual devices
virtualization layer
host operating system
(Linux)
hardware
CPU
memory
I/O devices

What does an OS do?

- Silberschatz and Gavin: “An OS is Similar to a government”
  - Begs the question: does a government do anything useful by itself?

- Coordinator and Traffic Cop:
  - Manages all resources
  - Prevents errors and improper use of the computer

- Facilitator (“useful” abstractions):
  - Provides facilities/services; Standard Libraries
  - Make application programming easier, faster, less error-prone

- Some features reflect both tasks:
  - File system is needed by everyone (Facilitator) …
  - … but File system must be protected (Traffic Cop)

What is an Operating System,… Really?

- Most Likely:
  - Memory Management
  - I/O Management
  - CPU Scheduling
  - Synchronization / Mutual exclusion primitives
  - Communications? (Does Email belong in OS?)
  - Multitasking/multiprogramming?

- What about?
  - File System?
  - Multimedia Support?
  - User Interface?
  - Internet Browser? ☺

Operating System Definition (Cont’d)

- No universally accepted definition

- “*Everything a vendor ships when you order an operating system*” is good approximation
  - But varies wildly

- “*The one program running at all times on the computer*” is the OS kernel
  - Everything else is either a system program (ships with the operating system) or an application program

Summary

- Provides a virtual machine abstraction to handle diverse hardware
- Coordinate resources and protect users from each other
- Simplify application development by providing standard services and abstractions
- Provide an array of fault containment, fault tolerance, and fault recovery

Machine language

- Each type of processor (like Pentium 4, Athalon, Z80, …) has its own instruction set
- Each instruction in an instruction set does a single thing like access a piece of data, add two pieces of data, compare two pieces of data …
- Each instruction is represented by a unique number .This # may be different for different instruction sets, but no two instructions in the same instruction set should have the same #

Machine Language programs

- A machine language program is a list of instructions
    - Each instruction is represented by a number
    - Inside the memory of the computer, each number is represented in binary (as a string of 1’s and 0’s)
    - The long string of 0’s and 1’s is easy for the computer to understand, but difficult for a human to read or write

Assembly

- Assembly languages make it easier for the programmer.
  - Assembly is easier for humans to read/write
  - Use mnemonics like ADD, CMP, … to replace the numbers that identify each of the instructions in the instruction set
  - The code for an Assembly program is written into a text file, which is translated into machine language program and executed.

Computer Software: Languages

- Some Computer Languages
  - Machine language (machine instruction set)
  - assembly language
  - high level languages (Compilers/Interpreters)
    - C, C++, Ada, Fortran, Basic, Java
    - Do YOU know of any others?
    - mathematical computation tools (MATLAB, Mathematica, ...)

- Application software is written using computer languages.

- Operating systems are also written using computer languages (often C, some assembly)

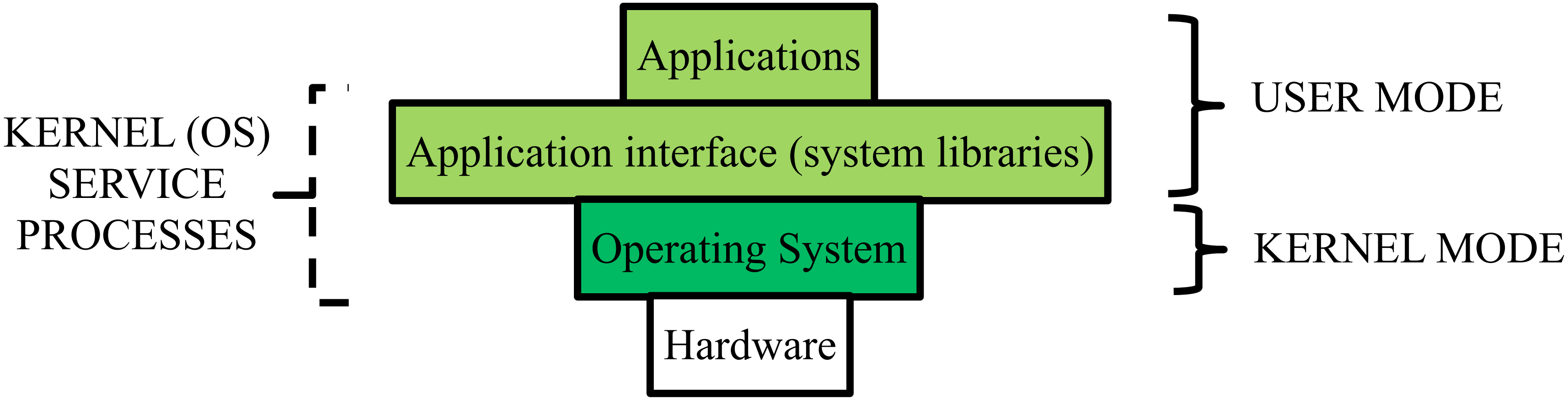
Computer Software: Applications

- Application Software (Software Tools)
  - Word processors (Microsoft Word, WordPerfect, ...)
  - Spreadsheet programs (Excel, Lotus1-2-3, ...)
  - Computer games
  - Communication software (email, chat, web browser…)
  - Telecommunication software (VOIP, …)
  - Integrated programming environments

User mode / kernel mode

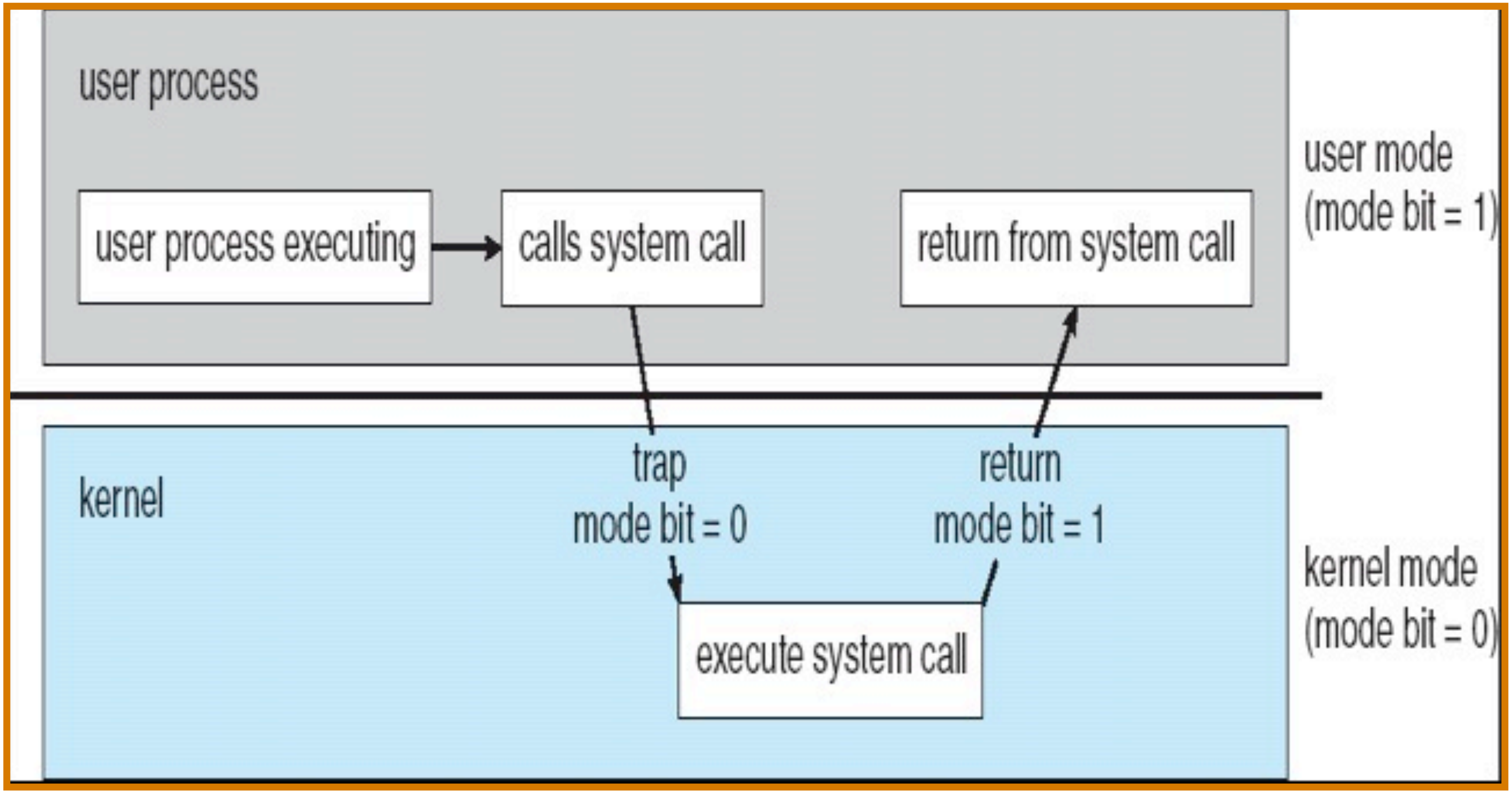
- Most application software runs in user mode. No access to direct hardware
- Operating systems run in kernel mode (supervisor mode) and have access to the complete instruction set,
- Application software running in user mode can used system calls to access hardware managed by OS
- User mode programs may perform duties for the OS

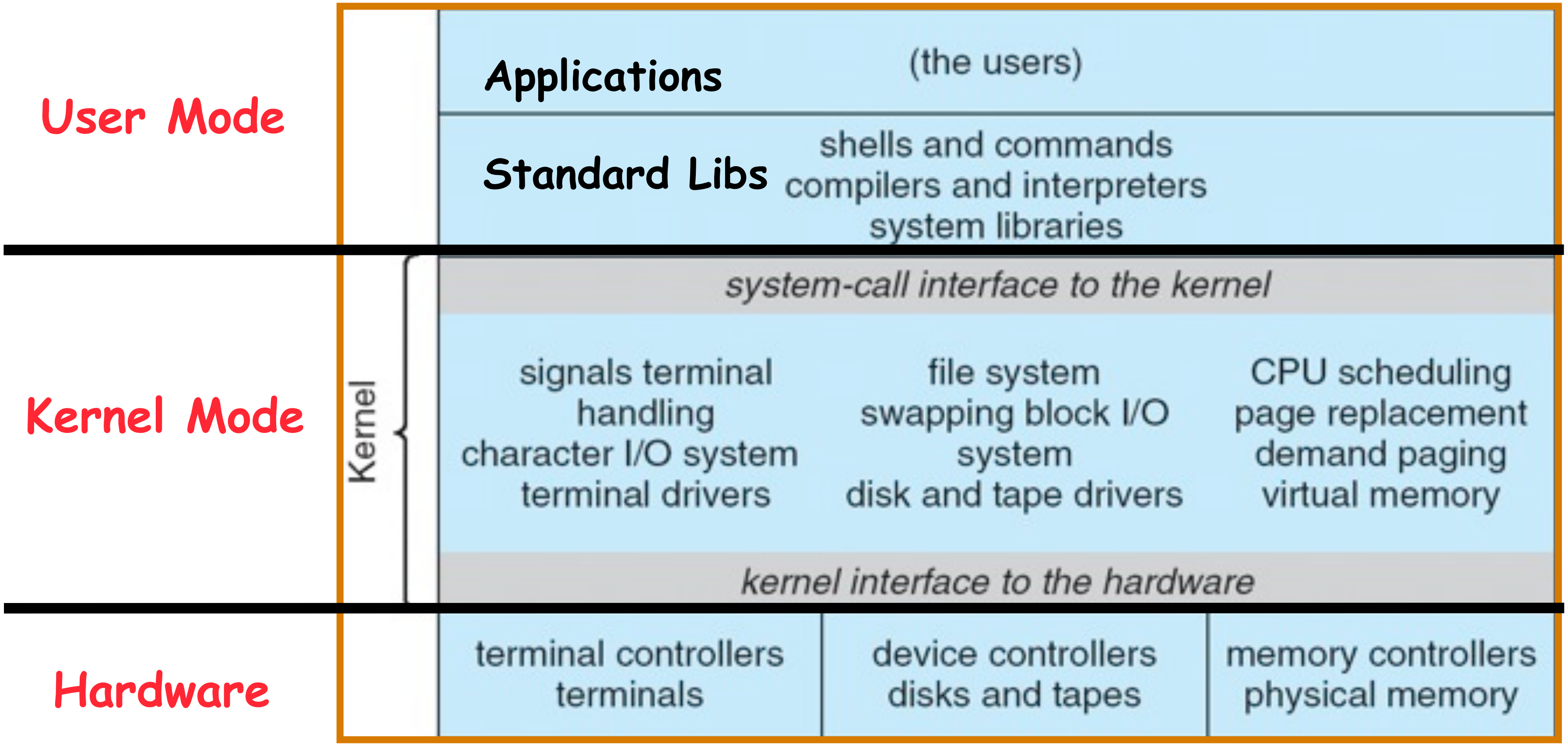
Modes



For some operating systems there may not be a separation between kernel mode and user mode (embedded systems, interpreted systems)

Dual-Mode operation





User Mode

Applications (the users)

Standard Libs
shells and commands
compilers and interpreters
system libraries

Kernel Mode

Kernel

system-call interface to the kernel

signals terminal handling
character I/O system
terminal drivers

file system
swapping block I/O system
disk and tape drivers

CPU scheduling
page replacement
demand paging
virtual memory

kernel interface to the hardware

Hardware

terminal controllers
terminals

device controllers
disks and tapes

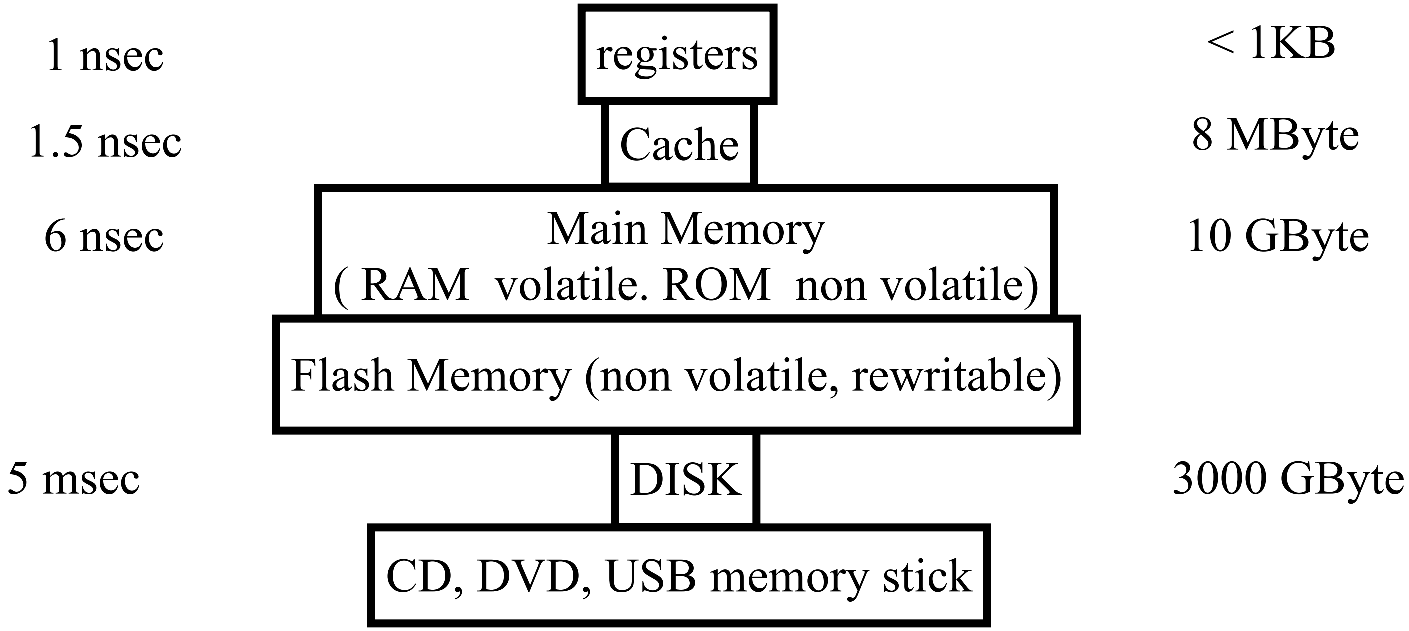
memory controllers
physical memory

Memory Hierarchy

- Different types of memory have different access speeds and costs
- Faster access speed implies higher cost
- Greater capacity often implies lower access speed
- From fastest access to slowest access
  - Registers
  - Cache
  - Memory
  - Disk
  - Tapes

Memory

- Modern computers use several kinds of storage



1 nsec registers < 1KB

1.5 nsec Cache 8 MByte

6 nsec Main Memory (RAM volatile. ROM non volatile) 10 GByte

Flash Memory (non volatile, rewritable)

5 msec DISK 3000 GByte

CD, DVD, USB memory stick

Memory Hierarchy

- As you go down the pyramid
  - Decreasing cost per bit, Increasing capacity
  - Increasing access time, Decreasing frequency of access

  - Note that the fastest memory, sometimes referred to as **primary memory**, is usually **volatile** (register, cache, main memory)

  - Non-volatile (continues to store information when the power is off) memory is usually slower. Referred to as **secondary** or **auxiliary memory**. e.g., Flash

Registers and cache

- Parts of the CPU
- Register access speed comparable to CPU clock speed
- Cache memory may be as fast or as much as several times slower
- Registers
  - Usually 64x64 for 64-bit machine, 32x32 for 32-bit machine
  - Usually < 1 Kbyte
- Cache
  - As much as 8Mbytes

Concept of Cache

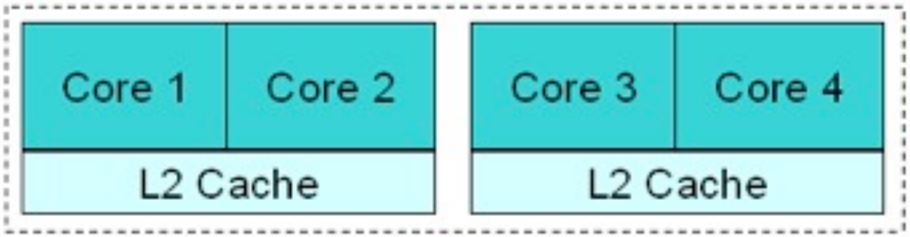
- Provide Hash-Table on the CPU
  - slower that the registers
  - cheaper and larger than registers
  - faster than main memory

Cache design

- Cache size and Cache line size
  - Determined to optimize access time

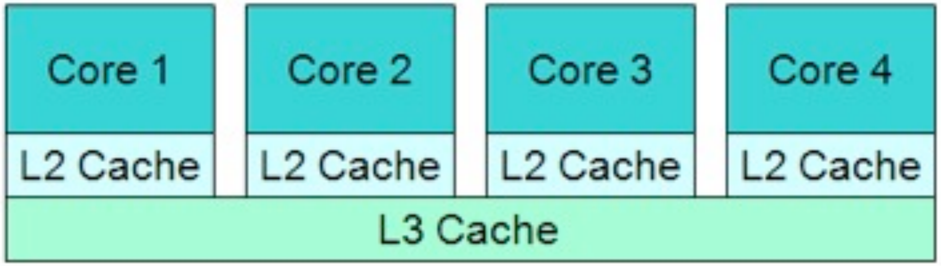
- Mapping function
  - Which cache lines may be loaded into which cache slots
    - can any line go in any slot, or is there a mapping function to define rules governing which line can be place in which slot
- Replacement algorithm
  - When is a cache line in a cache slot replaced by another cache line

Modern Cache Architectures



Core 2 Duo Core2 Quad

Shared cache requires more complicated cache controller



Nehalem (i5, i7) AMD K10 (Phenom 9)

Individual caches are more difficult to keep coherent (properly synchronized)

On-chip L1 caches are omitted from the figure

Memory

- Main memory is typically DRAM (Dynamic Random Access Memory)
- Cache is typically SRAM (Static Random Access Memory)
  - Smaller and faster than DRAM
- Both are volatile: contents lost when power is turned off

Disk

- Hard disk
- CD, DVD, Blu-Ray

Disk storage is much cheaper that memory

- (3GB memory or 2000GB disk about the same cost)
- Access time for disk is at least one order of magnitude slower than for memory

Input / Output

- Reading or writing data from a perepheral device is not simple
  - Each device has its own controller (hardware)
  - Each device is managed by a device driver (software to use the controller)
    - Device drivers are specific to hardware and to the operating system using the device
  - Input and output is SLOW in comparison to CPU operations.

Topic coverage

1 week	Fundamentals & History
1.5 weeks	Process Control and Threads
2.5 weeks	Synch. and Scheduling
2 weeks	Protection and Address Translation
1 week	Demand paging
1 week	File system
2.5 weeks	Network and Distributed Sys.
1 week	Protection and Security

Grading

- 5 Assignments (Total: 75%)
  - 3 / 5 linux kernel-based assignments
    - adding a syscall, thread scheduling, VM, I/O
  - Functional Shell
  - Synchronization

- Midterm and Final (10%)
- Class Participation (5%)

Computing

- All assignments will be Linux/C/gcc.
  - No Java, No C++
  - Real OSs use combination of C/C++
- All linux kernel assignments will be on QEMU (runs Linux on Linux)
- All assignments will be tested on undergrad lab machines.

CMPT 300

History of Operating Systems

[RIP] : Dennis Ritchie
Creator of C and Unix

History of Operating Systems

- First generation 1945 - 1955
  - vacuum tubes, plug boards
- Second generation 1955 - 1965
  - transistors, batch systems
- Third generation 1965 – 1980
  - ICs and multiprogramming
- Fourth generation 1980 – present
  - personal computers

The earliest computers (1945-55)

- Built of relays, vacuum tubes
- Very large, Very slow by today’s standards
- Built, programmed and maintained by the same people
- Programmed by using switches, paper tape, etc)
- No operating system, single operation, single problem, sequential access

The next generation (1955-65)

- Transistor based, increased reliability
- The first commercial mainframes, still very large and very expensive
- Used assembler or even early high level languages like Fortran or ALGOL
- Rudimentary operating system, one program at a time, with control commands to compile, load, execute, terminate, basic compilers
- Input using cards, paper tape, magnetic tape …

History of Operating Systems: Phases

- Phase 1: Hardware is expensive, humans are cheap
  - ➡ User at console: single-user systems
  - ➡ Batching systems
  - ➡ Multi-programming systems
- Phase 2: Hardware is cheap, humans are expensive
  - ➡ Time sharing: Users use cheap terminals and share servers
- Phase 3: Hardware is very cheap, humans are very expensive
  - ➡ Personal computing: One system per user
  - ➡ Distributed computing: lots of systems per user
- Phase 4: Ubiquitous computing/Cloud computing
  - ➡ Cell phone, mp3 player, DVD player, TIVO, PDA, iPhone, eReader
  - ➡ Software as a service, Amazon’s elastic compute cloud

History of Operating Systems: Phases

- Phase 1: Hardware is expensive, humans are cheap
  - User at console: single-user systems
  - Batching systems
  - Multi-programming systems
- Phase 2: Hardware is cheap, humans are expensive
  - Time sharing: Users use cheap terminals and share servers
- Phase 3: Hardware is very cheap, humans are very expensive
  - Personal computing: One system per user
  - Distributed computing: lots of systems per user
- Phase 4: Ubiquitous computing

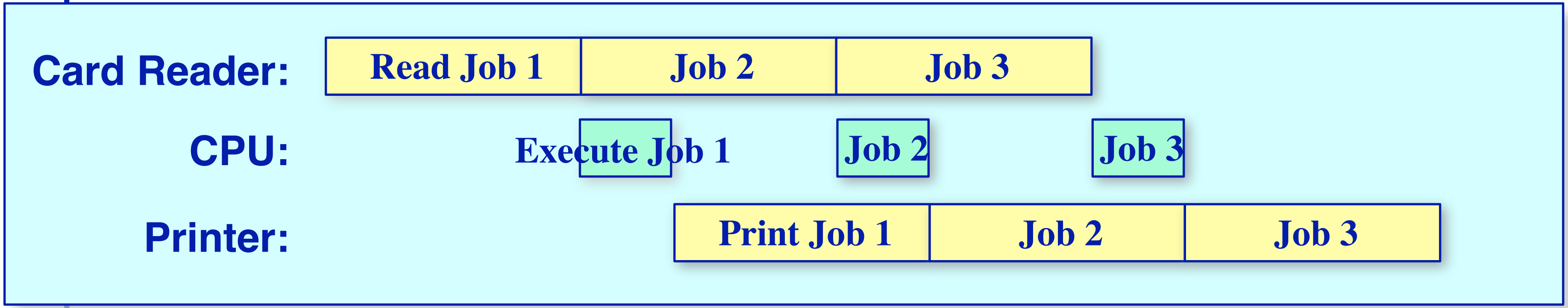
A Brief History of Operating Systems

Hand programmed machines ('45-'55)

- Single user systems
- OS = *loader* + *libraries of common subroutines*
- Problem: low *utilization* of expensive components

$$\frac{\textit{Execution time}}{\textit{Execution time} + \textit{Card reader time}} = \%\ \textit{utilization}$$

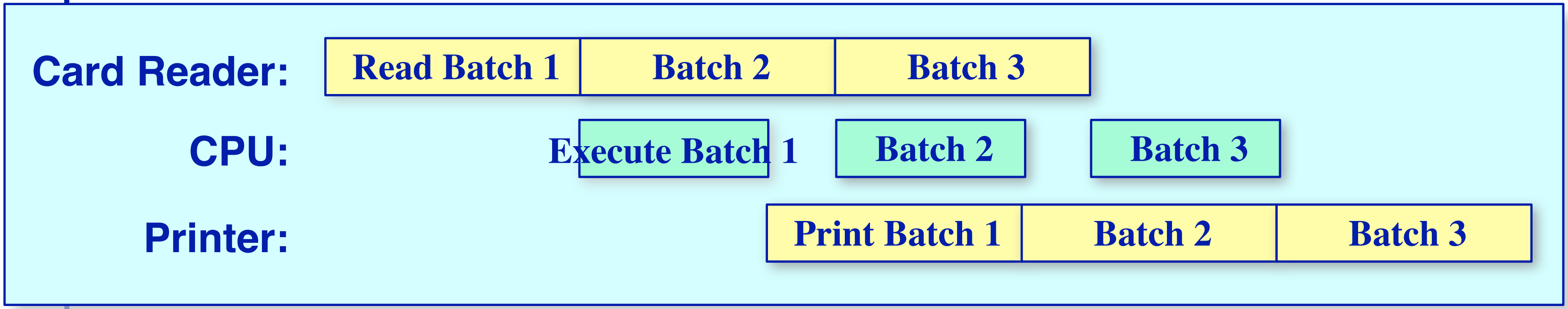
- Batching *v.* sequential execution of jobs



Card Reader: Read Job 1 | Job 2 | Job 3

CPU: Execute Job 1 | Job 2 | Job 3

Printer: Print Job 1 | Job 2 | Job 3

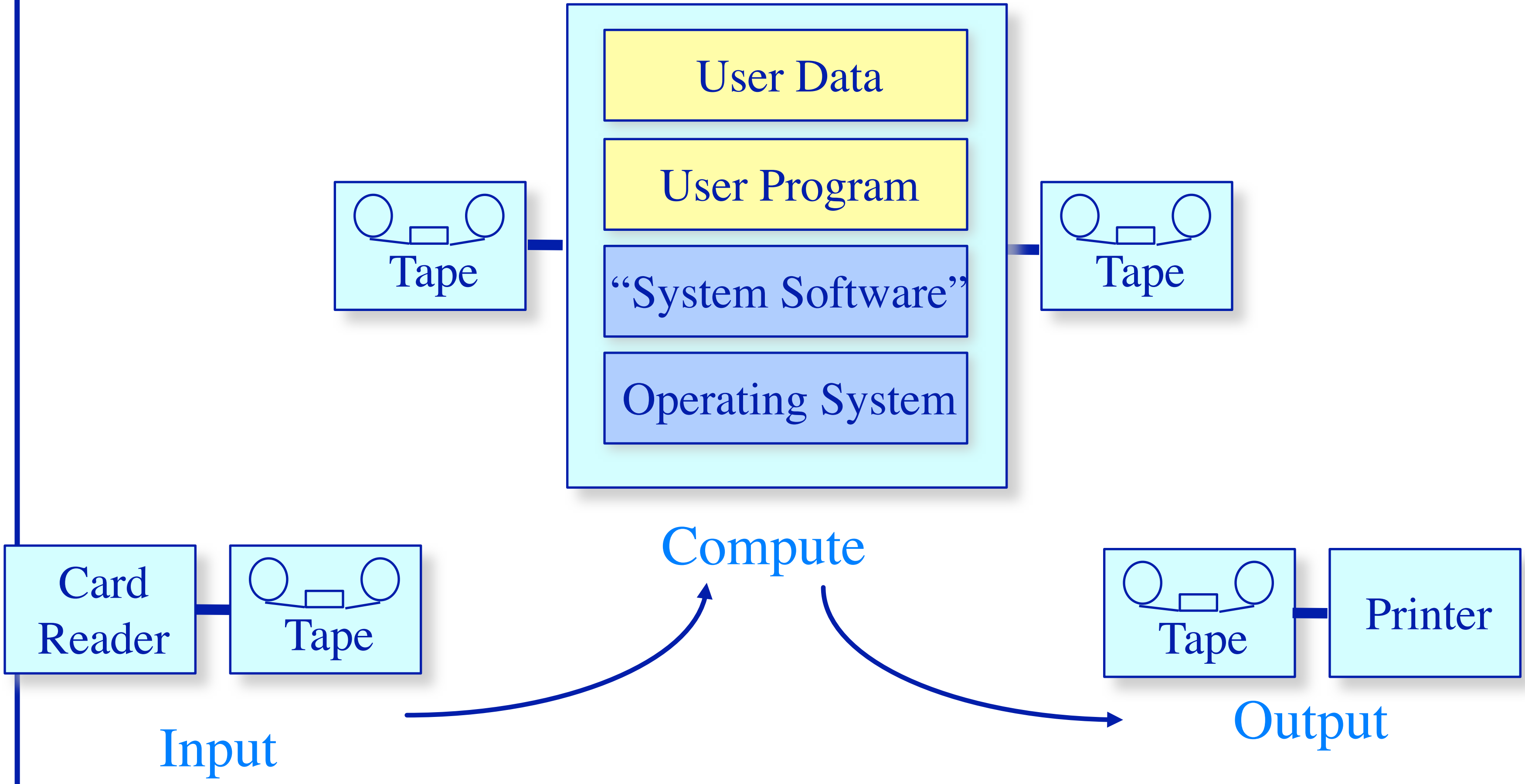


Card Reader: Read Batch 1 | Batch 2 | Batch 3

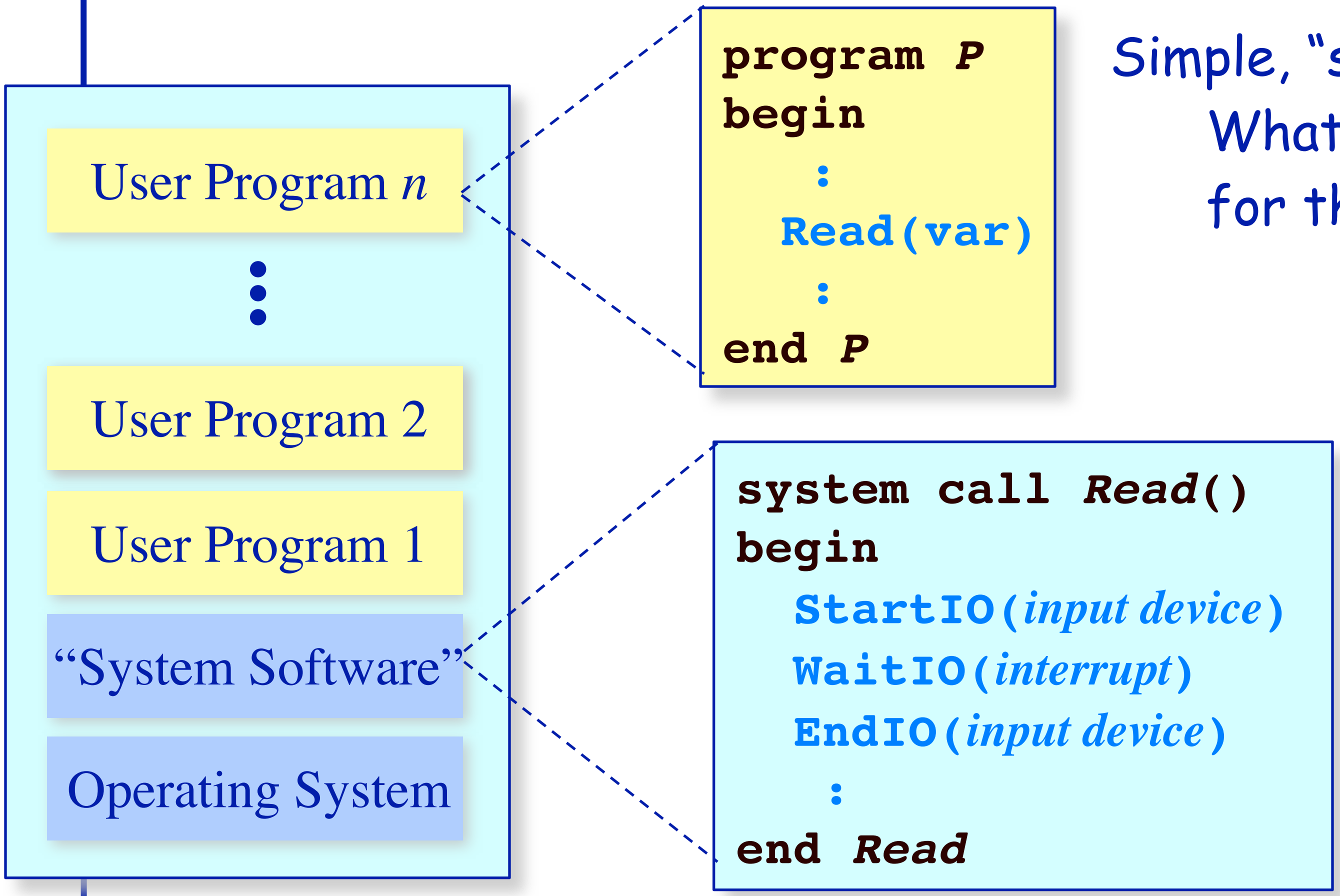
CPU: Execute Batch 1 | Batch 2 | Batch 3

Printer: Print Batch 1 | Batch 2 | Batch 3

- Operating system = *loader* + *sequencer* + *output processor*



- Keep several jobs in memory and multiplex CPU between jobs

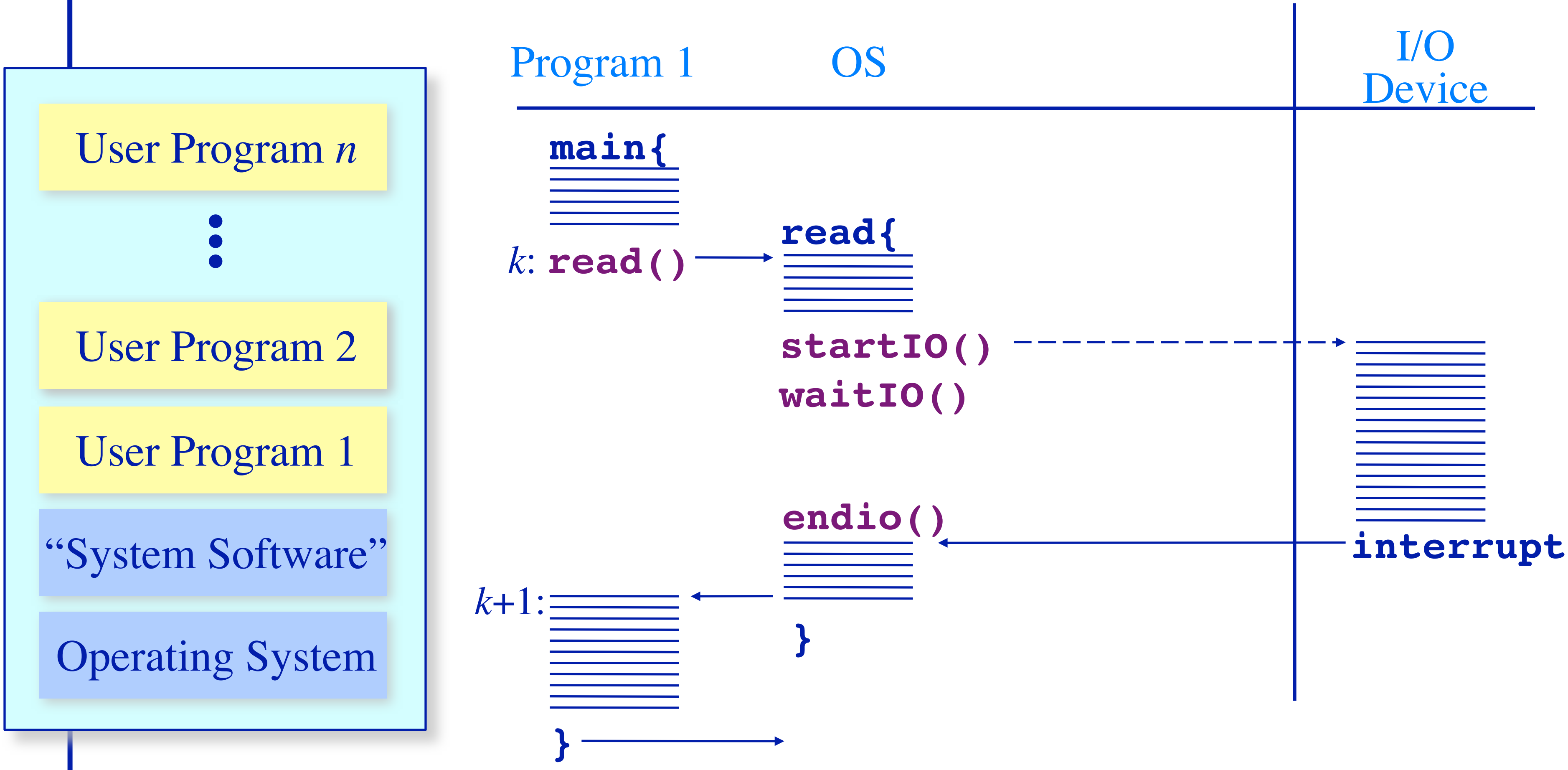


```
program P
begin
    :
   Read(var)
    :
end P
```

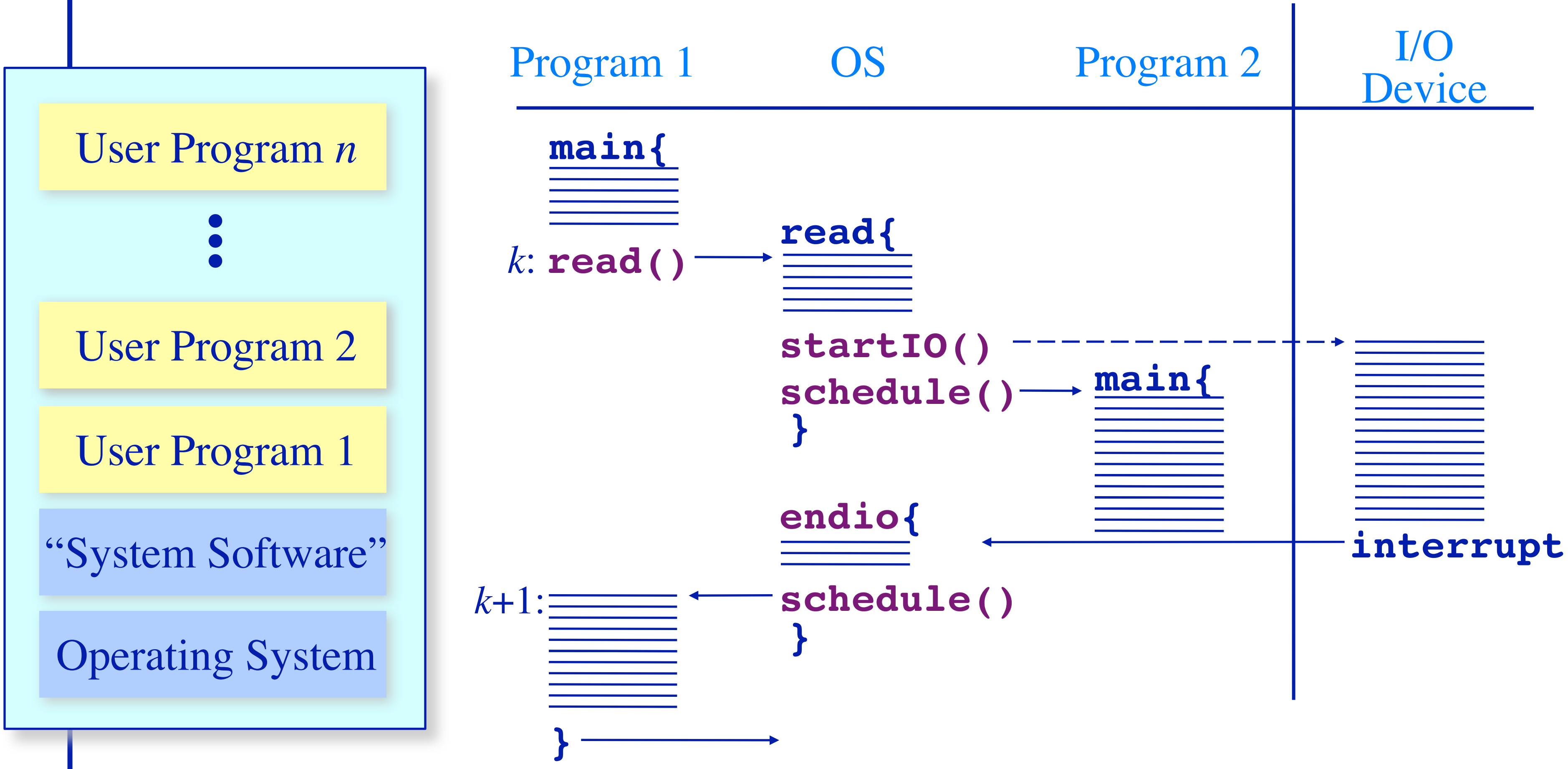
Simple, "synchronous" input:
What to do while we wait
for the I/O device?

```
system call Read()
begin
   StartIO(input device)
   WaitIO(interrupt)
   EndIO(input device)
    :
end Read
```

- Keep several jobs in memory and multiplex CPU between jobs



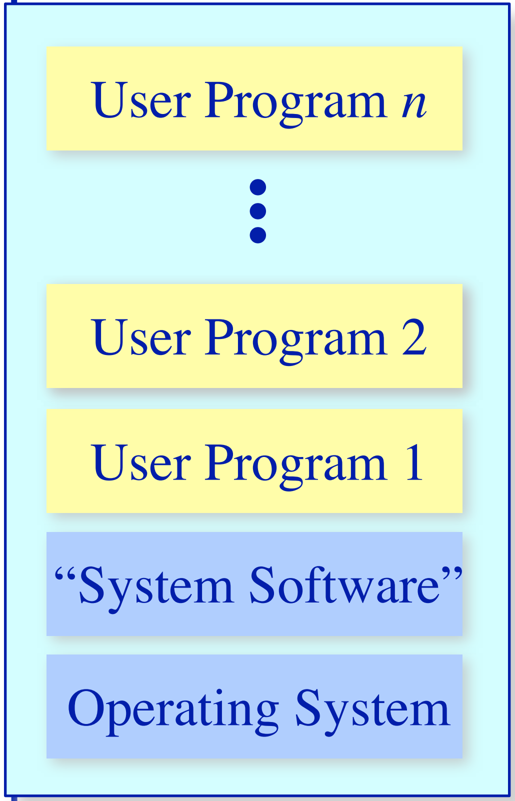
- Keep several jobs in memory and multiplex CPU between jobs



History of Operating Systems: Phases

- Phase 1: Hardware is expensive, humans are cheap
  - User at console: single-user systems
  - Batching systems
  - Multi-programming systems
- Phase 2: Hardware is cheap, humans are expensive
  - Time sharing: Users use cheap terminals and share servers
- Phase 3: Hardware is very cheap, humans are very expensive
  - Personal computing: One system per user
  - Distributed computing: lots of systems per user
- Phase 4: Ubiquitous computing

- A timer interrupt is used to multiplex CPU among jobs



User Program *n*

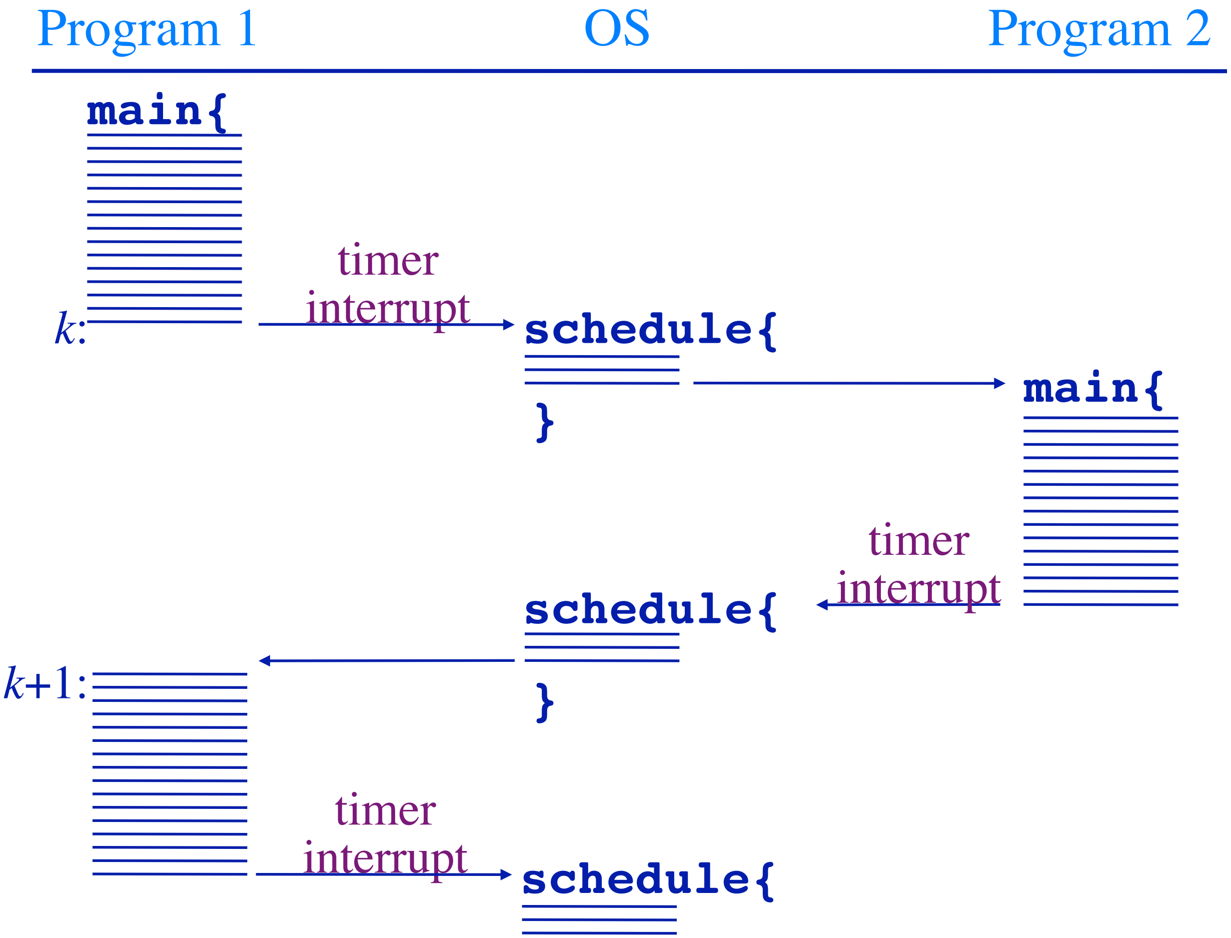
⋮

User Program 2

User Program 1

“System Software”

Operating System

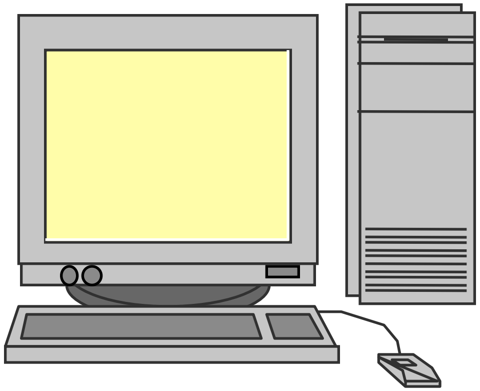


History of Operating Systems: Phases

- Phase 1: Hardware is expensive, humans are cheap
  - User at console: single-user systems
  - Batching systems
  - Multi-programming systems
- Phase 2: Hardware is cheap, humans are expensive
  - Time sharing: Users use cheap terminals and share servers
- Phase 3: Hardware is very cheap, humans are very expensive
  - Personal computing: One system per user
  - Distributed computing: lots of systems per user
- Phase 4: Ubiquitous computing

Operating Systems for PCs

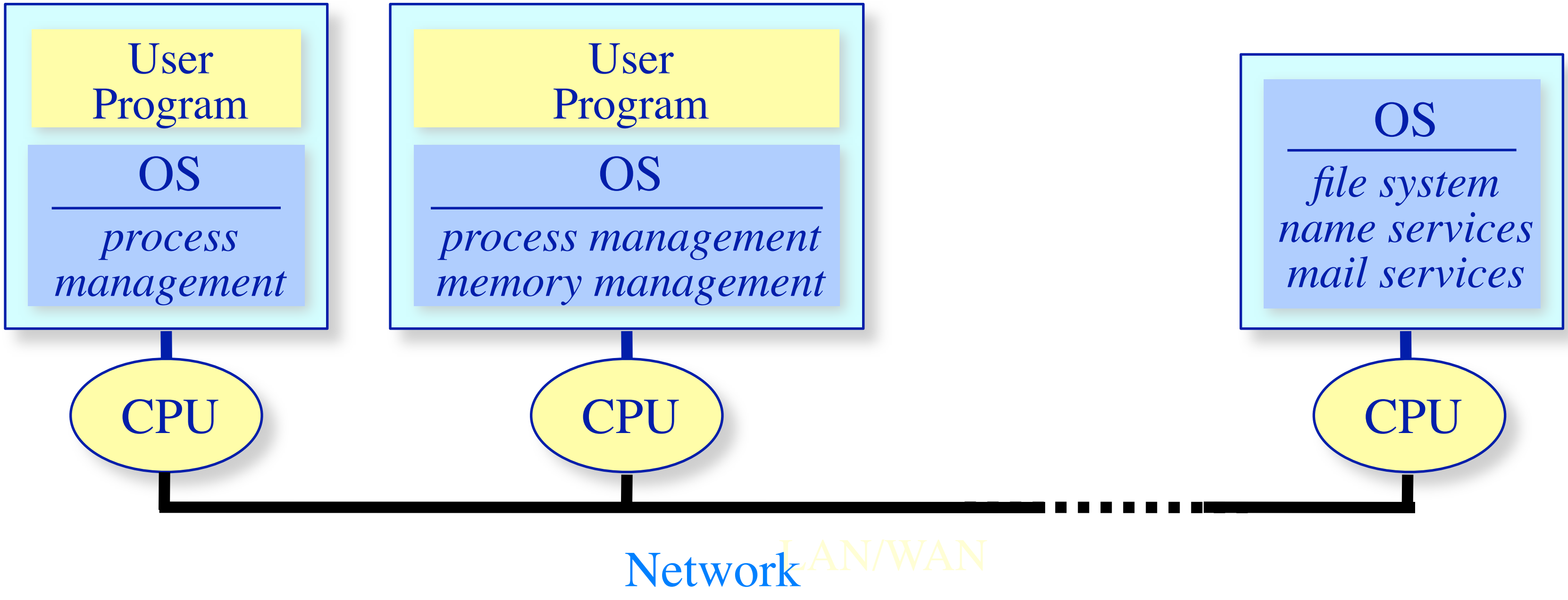
- Personal computing systems
  - Single user
  - Utilization is no longer a concern
  - Emphasis is on user interface and API
  - Many services & features not present



- Evolution
  - Initially: OS as a simple service provider (simple libraries)
  - Now: Multi-application systems with support for coordination and communication
  - Growing security issues (e.g., online commerce, medical records)

Distributed Operating Systems

- Typically support distributed services
  - Sharing of data and coordination across multiple systems
- Possibly employ multiple processors
  - Loosely coupled *v.* tightly coupled systems
- High availability & reliability requirements
  - Amazon, CNN



User Program

OS

process management

CPU

User Program

OS

process management
memory management

CPU

OS

file system
name services
mail services

CPU

Network LAN/WAN

History of Operating Systems: Phases

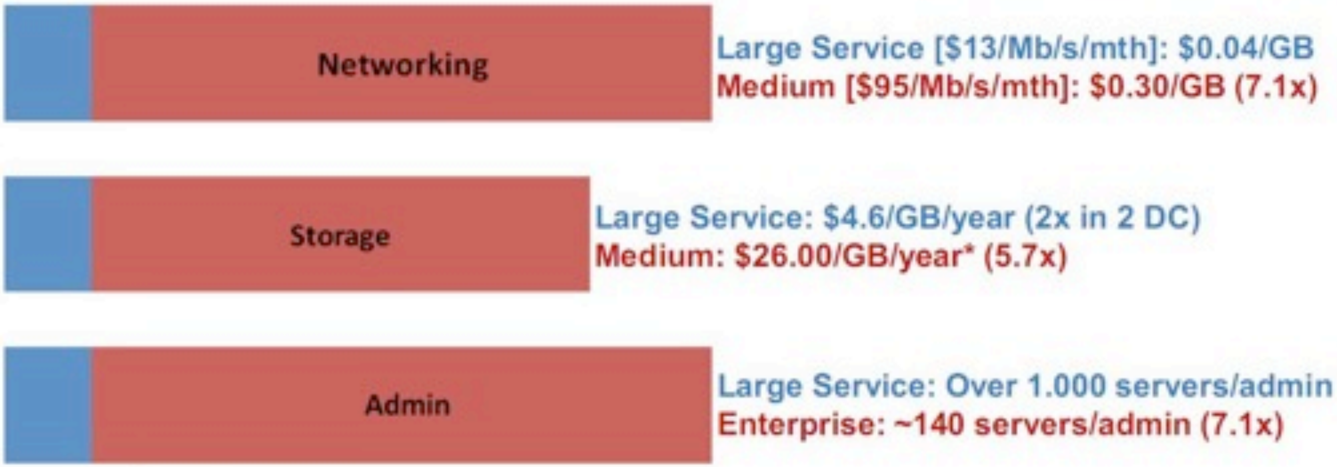
- Phase 1: Hardware is expensive, humans are cheap
  - User at console: single-user systems
  - Batching systems
  - Multi-programming systems
- Phase 2: Hardware is cheap, humans are expensive
  - Time sharing: Users use cheap terminals and share servers
- Phase 3: Hardware is very cheap, humans are very expensive
  - Personal computing: One system per user
  - Distributed computing: lots of systems per user
- Phase 4: Ubiquitous computing/Cloud computing
  - Everything will have computation, from pacemakers to toasters
  - Computing centralizing
  - "I think there is a world market for maybe five computers" – Tomas J. Watson, 1943 (president of IBM)

What is cloud computing?

- **Cloud computing** is where dynamically scalable and often virtualized resources are provided as a service over the Internet (thanks, wikipedia!)
- Infrastructure as a service (IaaS)
  - Amazon’s EC2 (elastic compute cloud)
- Platform as a service (PaaS)
  - Google gears
  - Microsoft azure
- Software as a service (SaaS)
  - gmail
  - facebook
  - flickr

Services Economies of Scale

- Substantial economies of scale possible
- 2006 comparison of very large service with small/mid-sized: (~1000 servers):



- High cost of entry
  - Physical plant expensive: 15MW roughly $200M
- Summary: significant economies of scale but at very high cost of entry
  - Small number of large players likely outcome

Thanks, James Hamilton, amazon

Richer Operating Systems

Intellectual property

- Copyrighted material is being disseminated in digital form without payment to copyright owners.
- Sue them (DMCA)
  - ➡ Napster (99-7/00)
  - ➡ RIAA lawsuits (9/03)
  - ➡ MPAA lawsuits against bittorrent operators (11/04)
- What is the future of file sharing?
  - ➡ Attempts to ban all file sharing at the university level.
  - ➡ Government tapping of IP networks.
- Can software stop intellectual property piracy?
  - ➡ Why not? The consumer controls the OS.
- What about adding hardware?
  - ➡ Intel's trusted execution technology. Who is trusted? Hint: Its not the owner of the computer…
- A PC is an open-ended system, not an appliance. For how much longer?

Richer Operating Systems

- Is it better to search for data (google), or organize it hierarchically (file folders)?
  - ➡Organization along a particular set of ideas (schema) might not be ideal for a different set of ideas.
  - ➡Gmail search vs. mail folders
- Integration of search in Vista and MacOS.
  - ➡Do you use My Documents folder, or do you maintain your own directories? use both a lot?