CS341: Operating System

Course Files:

https://u.pcloud.link/publink/show?code=kZITAQXZCxtf3rGMIhpo90X7HMyGRXe3cBOk

Credit Some Slide: UCB CS162

Operating System

Process Management: process; thread; scheduling. Concurrency: mutual exclusion; synchronization; semaphores; monitors; Deadlocks: characterization; prevention; avoidance; detection. Memory Management: allocation; hardware support; paging; segmentation. Virtual Memory: demand paging; replacement; allocation; thrashing. File Systems and Implementation. Secondary Storage: disk structure; disk scheduling; disk management. (Linux will be used as a running example, while examples will drawn also from Windows NT/7/8.); Advanced Topics: Distributed Systems. Security. Real-Time Systems.

Prerequisite & Text Book

- Prerequisite: **PDS & Computer Architecture**

- Text Books

- A. Silberschatz, P. B. Galvin and G. Gagne, Operating System Concepts, 8th Ed, John Wiley & Sons, 2010.

- A. S. Tenenbaum, Modern Operating Systems, 2nd Ed, Prentice Hall of India, 2001.

- H. M. Deitel, P. J. Deitel and D. R. Choffness, Operating Systems, 3rd Ed, Prentice Hall, 2004.

- W. Stallings, Operating Systems: Internal and Design Principles, 5th Ed, Prentice Hall, 2005.

- M. J. Bach, The Design of the UNIX Operating System, Prentice Hall of India, 1994.

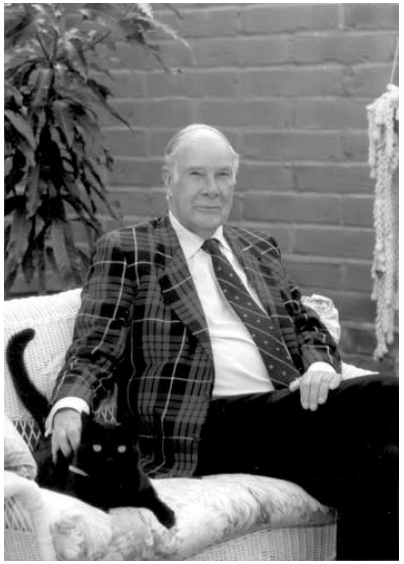
- M. K. McKusick et al, The Design and Implementation of the 4.4 BSD Operating System, Addison Wesley, 1996.

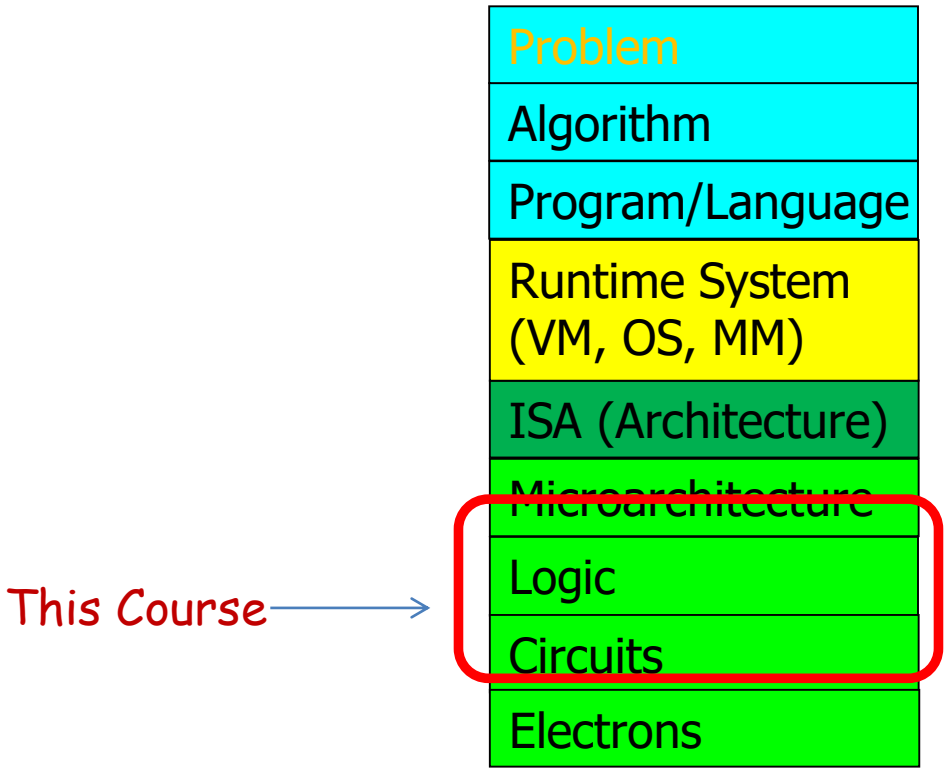
Evaluation Policy(CS341)

- Assignments
  & Quizzes
- Mid-Sem
- End-Sem
- Mini Project

Levels of Transformation

"The purpose of computing is insight" (*Richard Hamming*)
We gain and generate insight by solving problems
How do we ensure problems are solved by electrons?



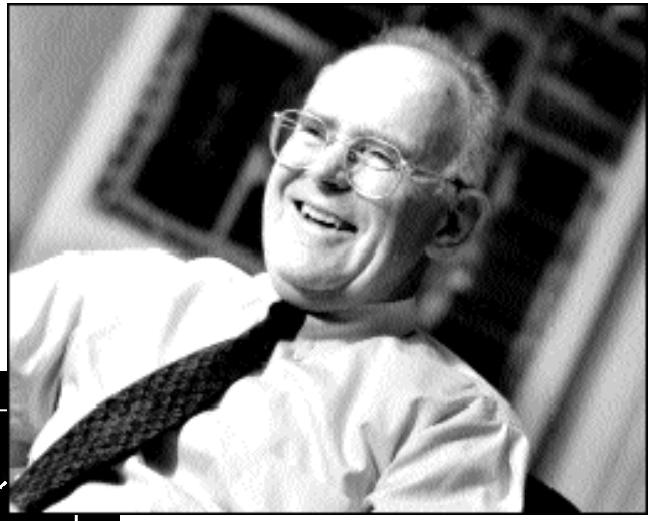


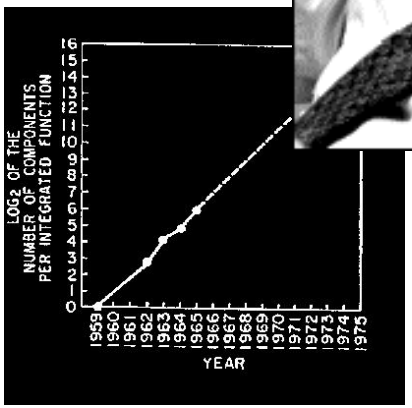
The Power of Abstraction

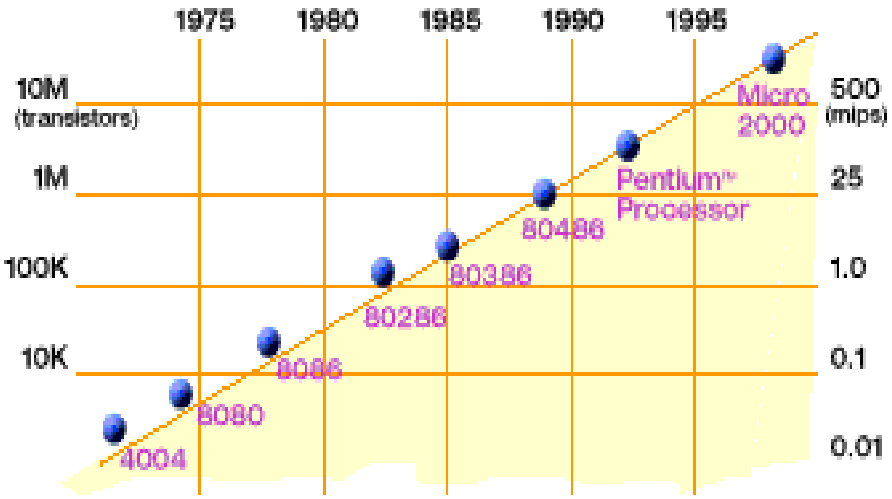
- Levels of transformation create abstractions
  - Abstraction: A higher level only needs to know about the interface to the lower level, not how the lower level is implemented
  - E.g., high-level language programmer does not really need to know what the ISA is and how a computer executes instructions

- Abstraction improves productivity
  - No need to worry about decisions made in underlying levels
  - E.g., programming in Java vs. C vs. assembly vs. binary vs. by specifying control signals of each transistor every cycle

Technology Trends: Moore's Law







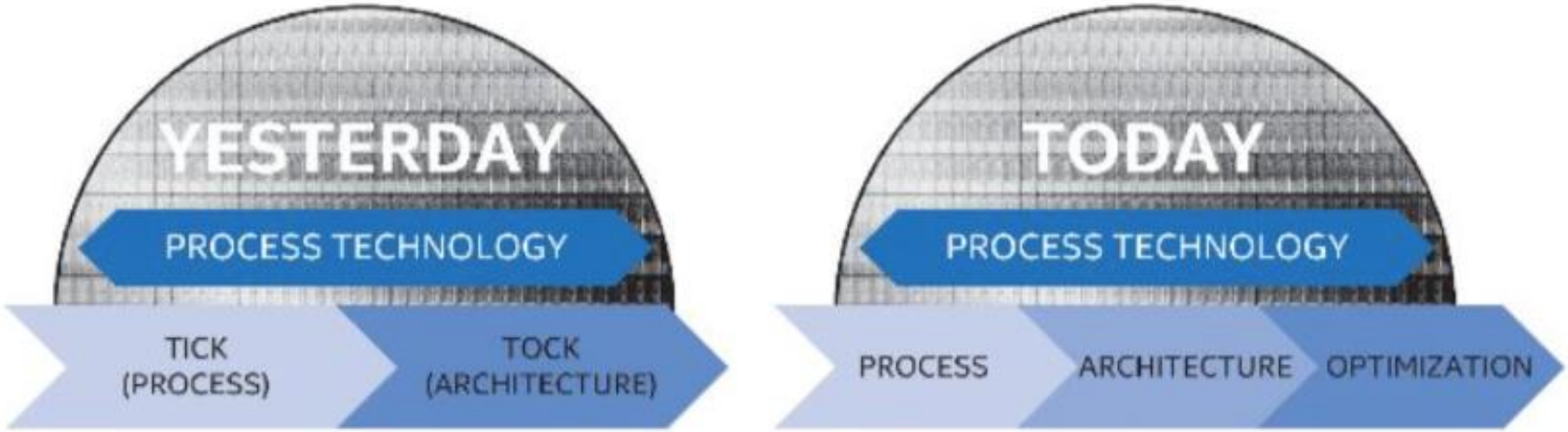
2X transistors/Chip Every 1.5 years

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.

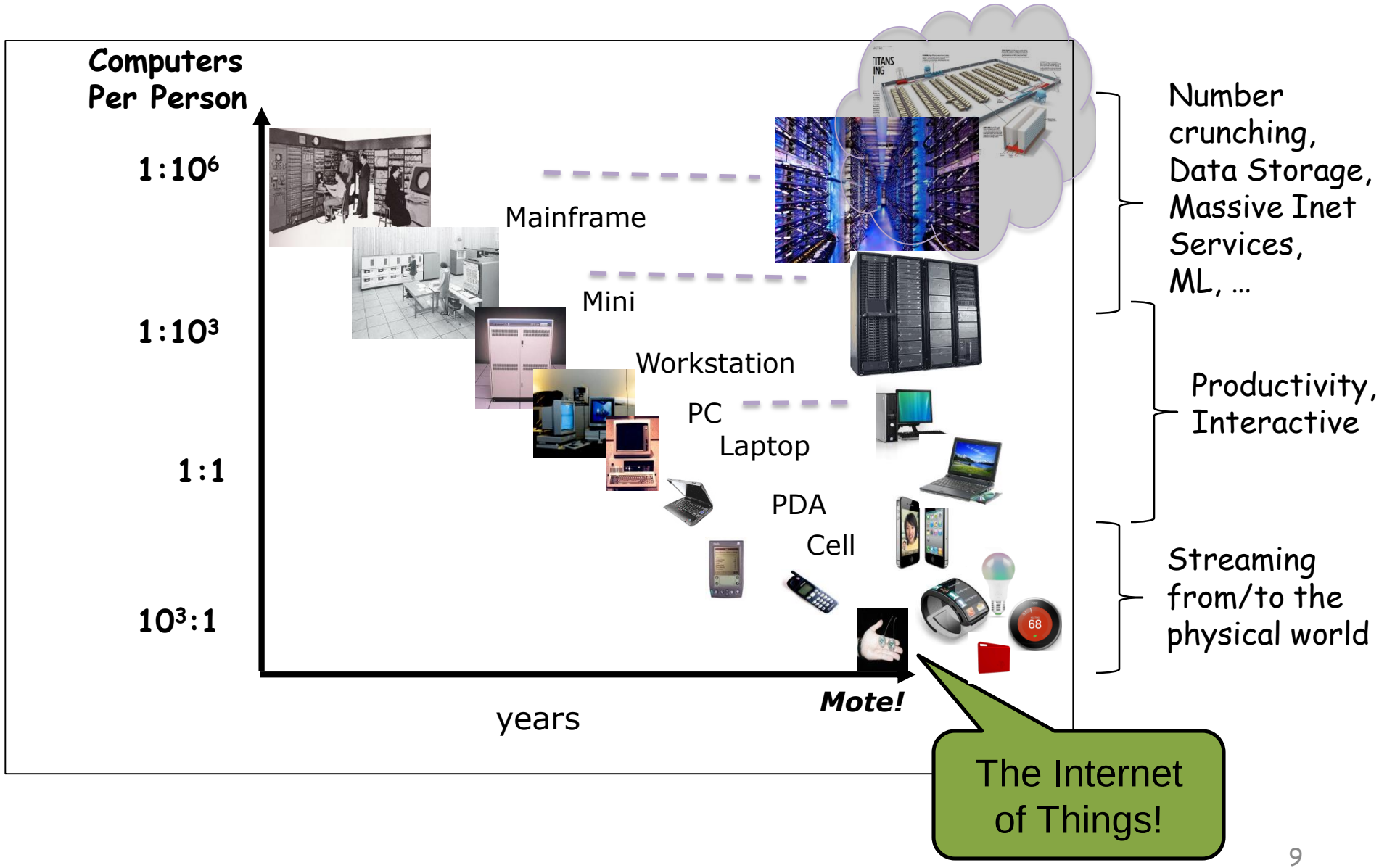
Microprocessors have become smaller, denser, and more powerful.

But then Moore's Law Ended…

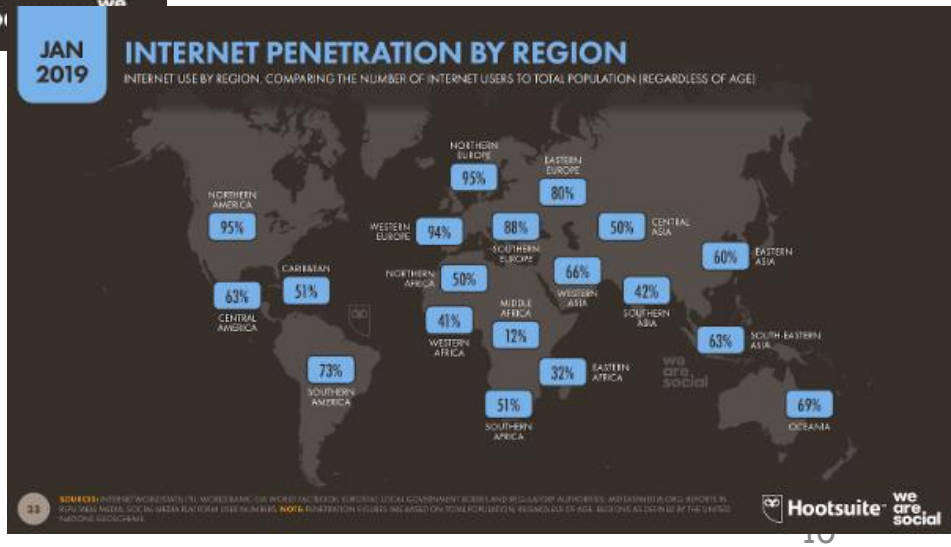


- Moore's Law has (officially) ended -- Feb 2016
  - No longer getting 2 x transistors/chip every 18 months…
  - or even every 24 months
- May have only 2-3 smallest geometry fabrication plants left:
  - Intel and Samsung and/or TSMC
- Vendors moving to 3D stacked chips
  - More layers in old geometries

Bell's Law: new computer class per 10 years

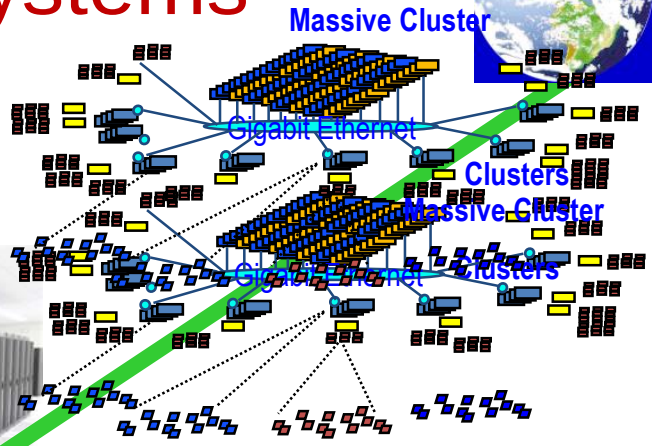


Society connected



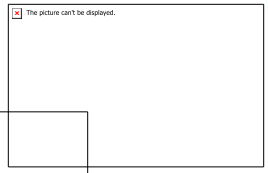
Societal Scale Information Systems

- The world is a large distributed system
  - Microprocessors in everything
  - Vast infrastructure behind them



Internet Connectivity

Scalable, Reliable, Secure Services



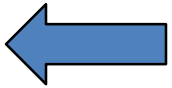
Databases
Information Collection
Remote Storage
Online Games
Commerce
…

MEMS for Sensor Nets

Range of Timescales

Jeff Dean: "Numbers Everyone Should Know"

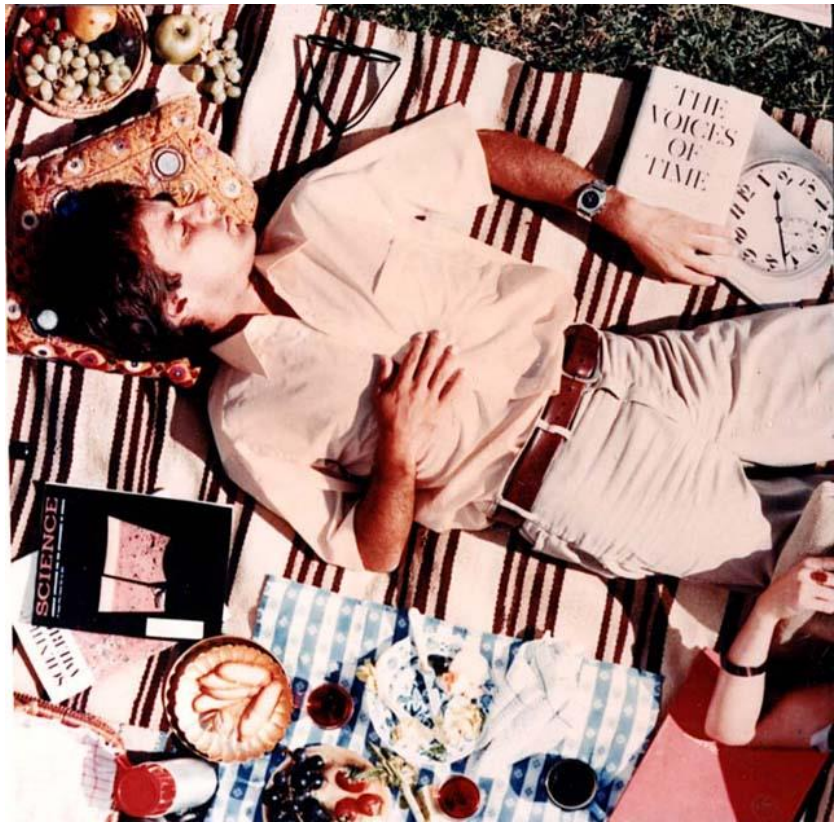
L1 cache reference	0.5 ns
Branch mispredict	5 ns
L2 cache reference	7 ns
Mutex lock/unlock	25 ns
Main memory reference	100 ns
Compress 1K bytes with Zippy	3,000 ns
Send 2K bytes over 1 Gbps network	20,000 ns
Read 1 MB sequentially from memory	250,000 ns
Round trip within same datacenter	500,000 ns
Disk seek	10,000,000 ns
Read 1 MB sequentially from disk	20,000,000 ns
Send packet CA->Netherlands->CA	150,000,000 ns



Key Stroke / Click
100 ms

Orders of Magnitude

10^0



10^1

Source: UMASS

Orders of Magnitude

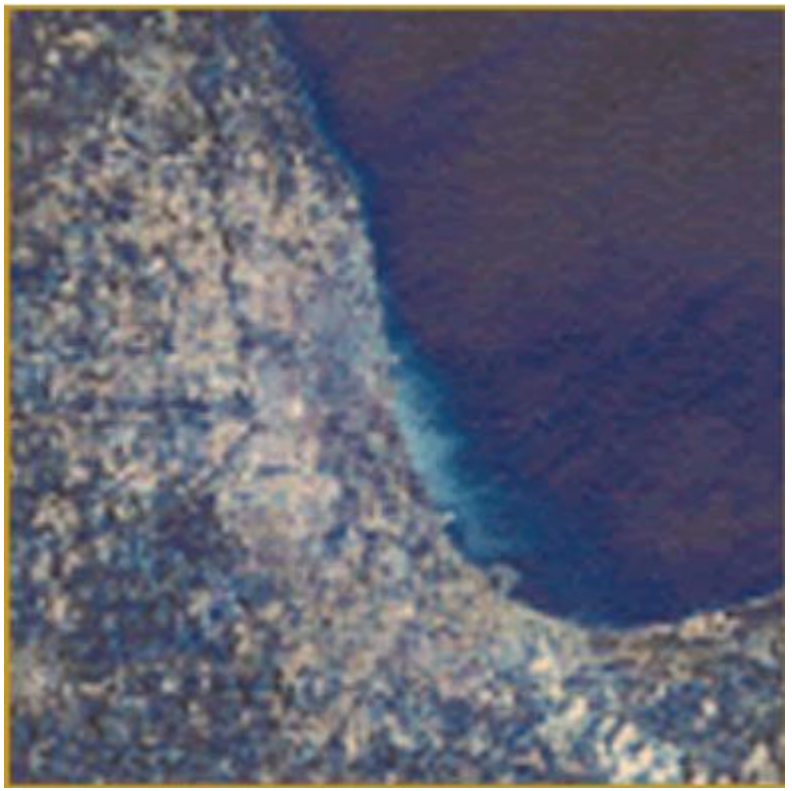
10^2

10^3

Orders of Magnitude

10^4

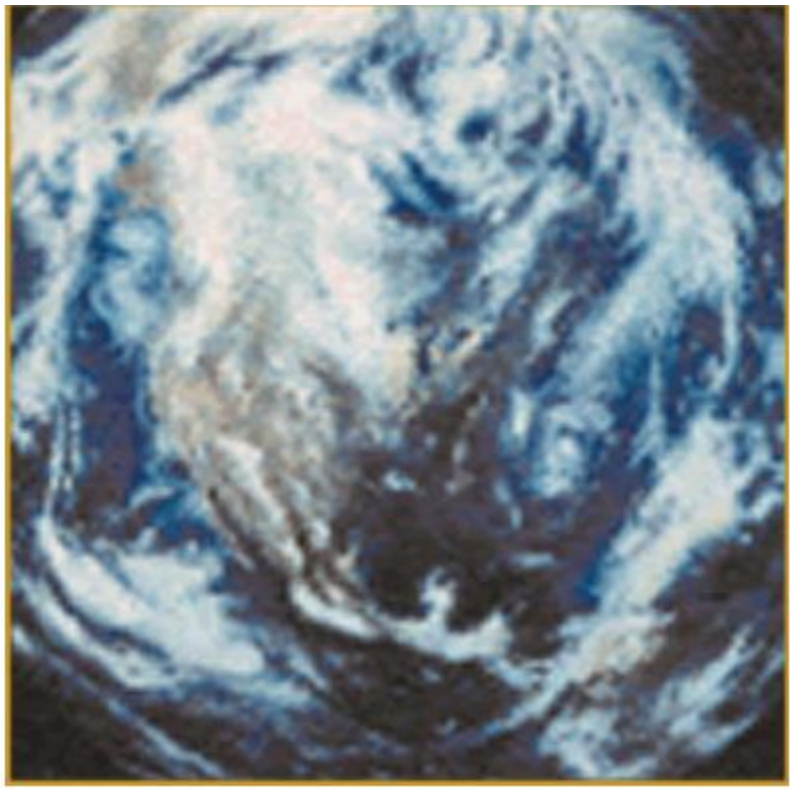
10^5



Orders of Magnitude

10^6

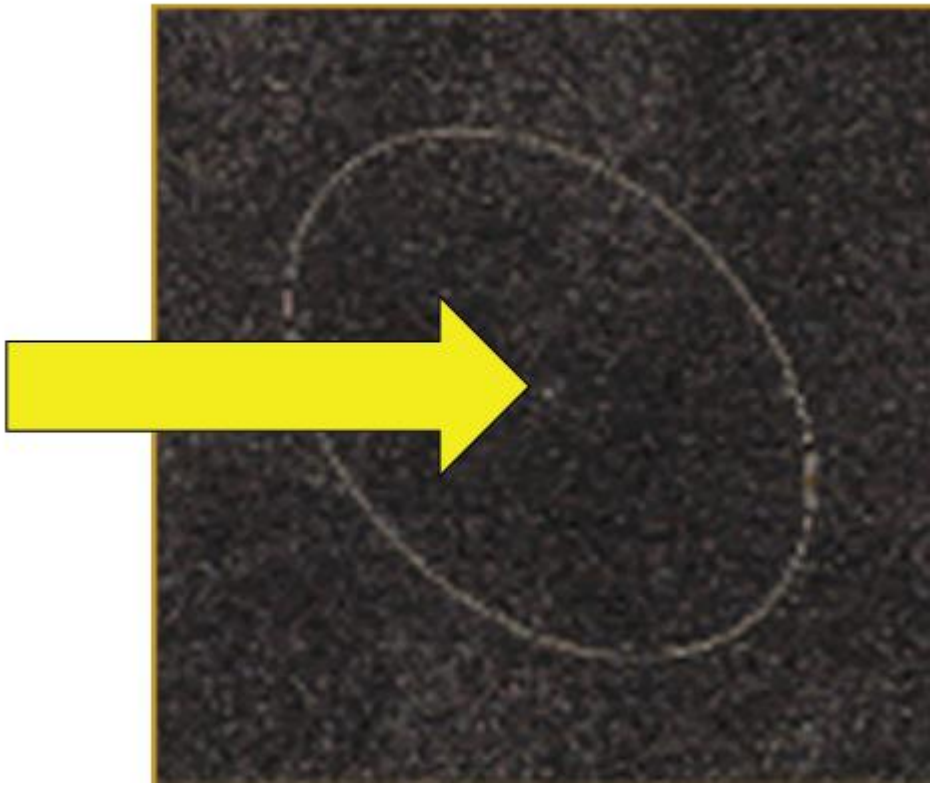
10^7



Orders of Magnitude

10^8

10^9



Why CS341?

- To learn how computers work
- To learn how to manage complexity through appropriate abstractions

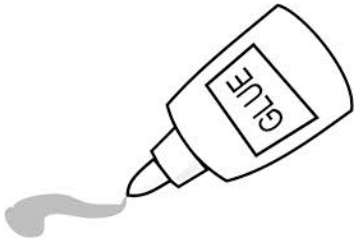
  -infinite CPU, infinite memory, files, semaphores, etc.
- To learn about system design

  -performance vs. simplicity, HW vs. SW, etc

- Because OSs are everywhere!

What is an Operating System?

- Referee
  - Manage sharing of resources, Protection, Isolation
    - Resource allocation, isolation, communication
- Illusionist
  - Provide clean, easy to use abstractions of physical resources
    - Infinite memory, dedicated machine
    - Higher level objects: files, users, messages
    - Masking limitations, virtualization



- Glue
  - Common services
    - Storage, Window system, Networking
    - Sharing, Authorization
    - Look and feel

What is an Operating System?

- Manager
  - Manage sharing of resources, Protection, Isolation
    - Resource allocation, isolation, communication
- Poet
  - Provide clean, easy to use abstractions of physical resources
    - Infinite memory, dedicated machine
    - Higher level objects: files, users, messages
    - Masking limitations, virtualization
- Coordinator
  - Common services
    - Storage, Window system, Networking
    - Sharing, Authorization
    - Look and feel

What is an Operating System?

- Definition has changed over years
  - – Originally, very bare bones
  - – Now, includes more and more
- Operating System (OS)

  -interface between the user and the architecture

  -Implements a virtual machine that is easier to program than raw hardware.

Summary of Operating System Principles

OS as juggler: providing the illusion of a dedicated machine with infinite memory and CPU.

- **OS as government:** protecting users from each other, allocating resources efficiently and fairly, and providing secure and safe communication.

- **OS as complex system:** keeping OS design and implementation as simple as possible is the key to getting the OS to work.

- **OS as history teacher:** learning from past to predict the future, i.e., OS design tradeoffs change with technology.

Abstraction vs. Arbitration

Abstraction

software that hides lower level details and provides a set of high-level functions

Reasons

The code needed to control peripheral devices is not standardized

Performs operations on behalf of programs. Example: Input/Output

The operating system introduces new functions as it abstracts the hardware.

Operation systems introduce the file abstraction so that programs don't need to have a disk to operate.

The operating system transforms the computer hardware into multiple virtual computers.

-Each running program is called a process.

The operating system can enforce security through abstraction.

Abstraction Vs. Arbitration

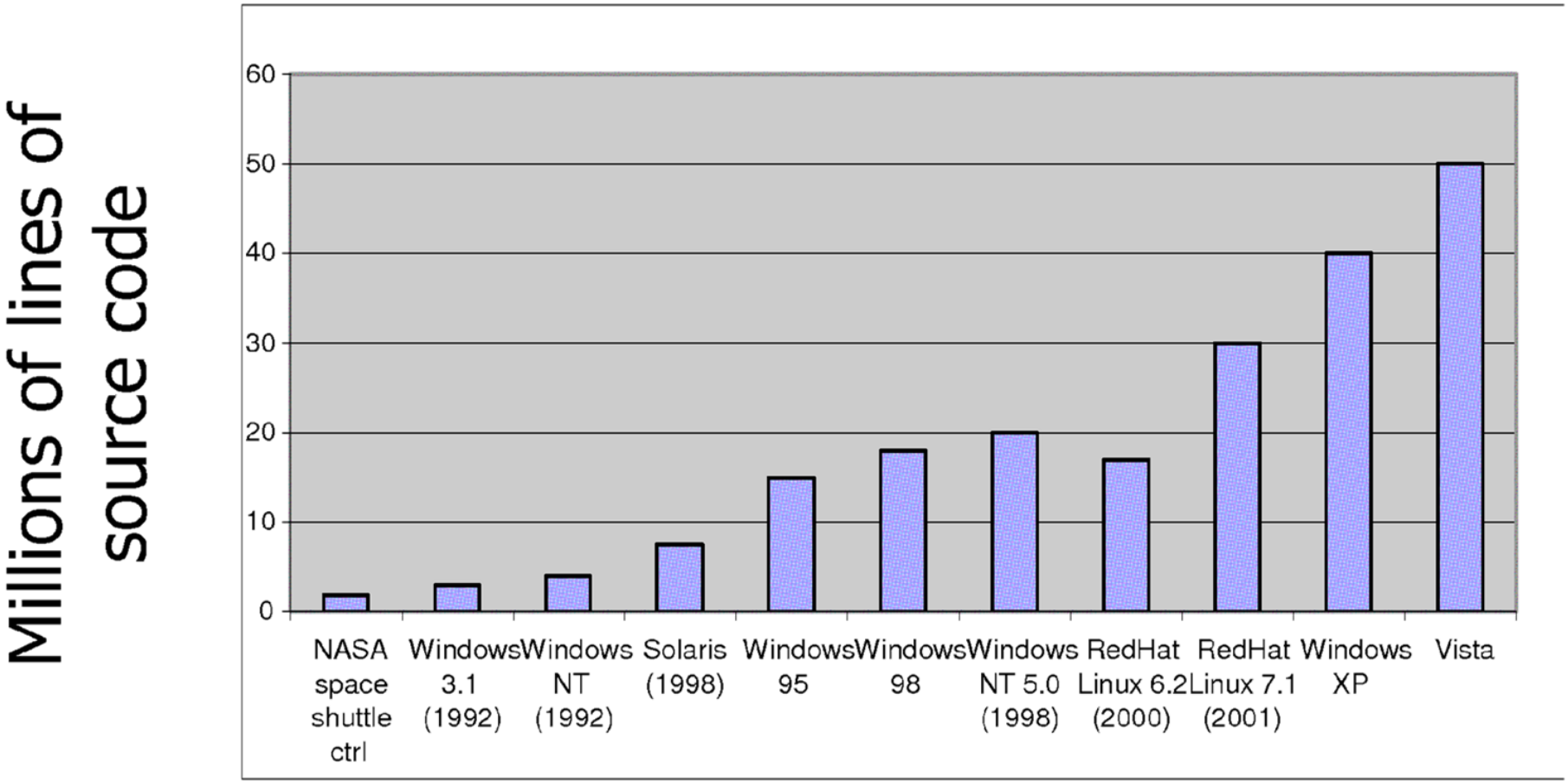
Arbitration

The set of rules in a computer's operating system for allocating the resources of the computer.

Reasons
Allows its peripheral devices or memory to be run by more than one user.
Must be able to handle the potential access conflicts between processors.
-Can result in data corruption.

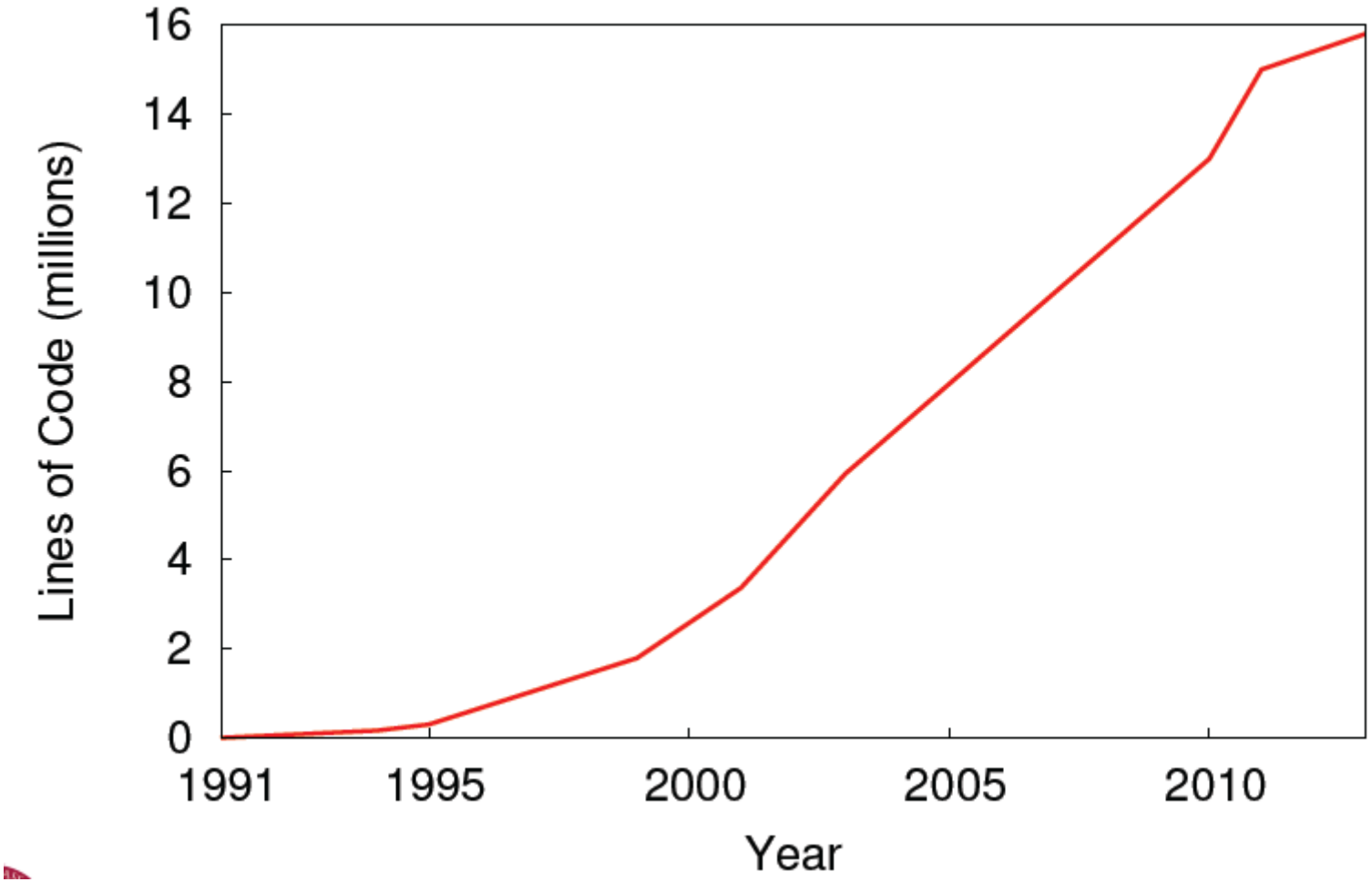
Arbitration failure occurs when multiple request signals arrive at the same location at the same time.

Increasing Software Complexity

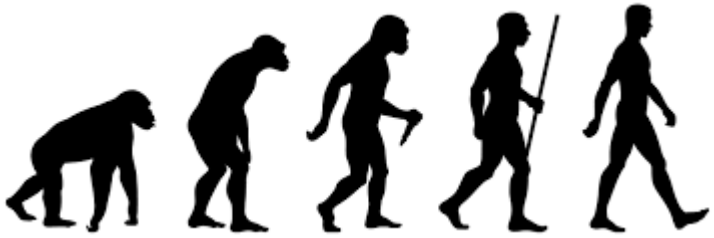


From MIT's 6.033 course

Linux Kernel Size

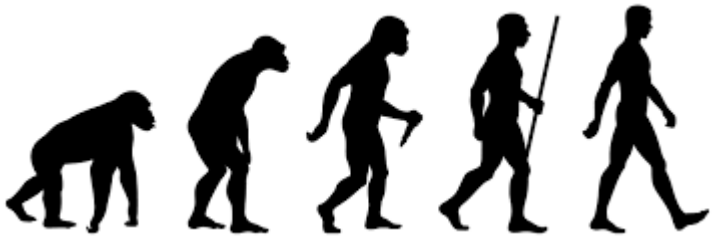


History of Operating Systems



- The first generation (1945–55) vacuum tubes
- The second generation (1955–65) transistors and batch systems
- The third generation (1965–1980) ICs and multiprogramming
- The fourth generation (1980–present) personal computers
- The fifth generation (1990–present) mobile computers

History of OS

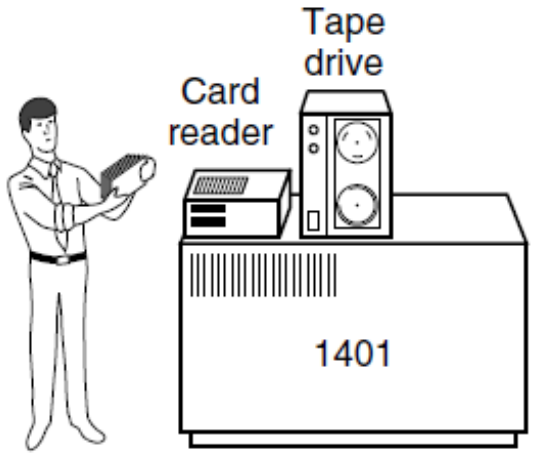


History Phase 1: Hardware expensive, humans cheap

Computers cost millions of $ optimize to make most efficient use of hardware

1) User at console – one user at a time; OS is a subroutine library (Literally a stack of cards you pulled off a shelf to, say, do a matrix multiply)

Problem – have to wait between jobs while user enters next job (innovations make job entry faster: binary switches-> keyboard-> card reader ->tape reader)



History of Operating Systems

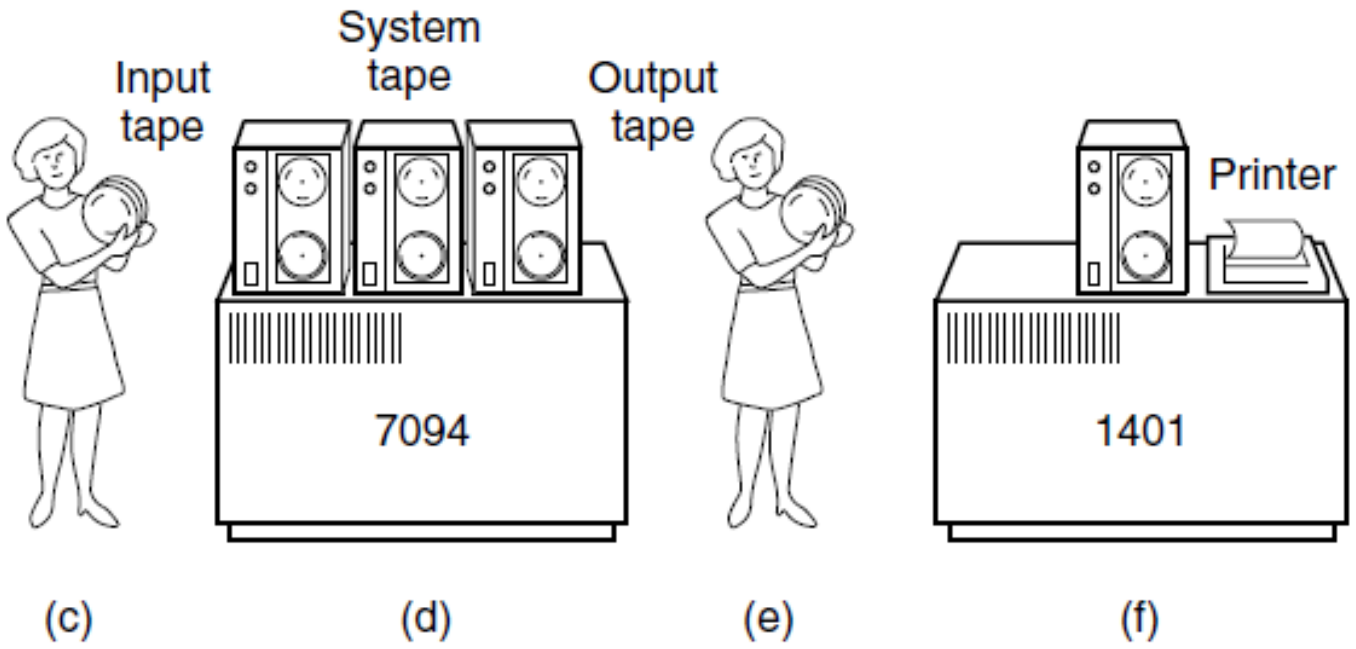
2) Batch monitor – load program, run print

Advantage – can load next job immediately as previous one finishes

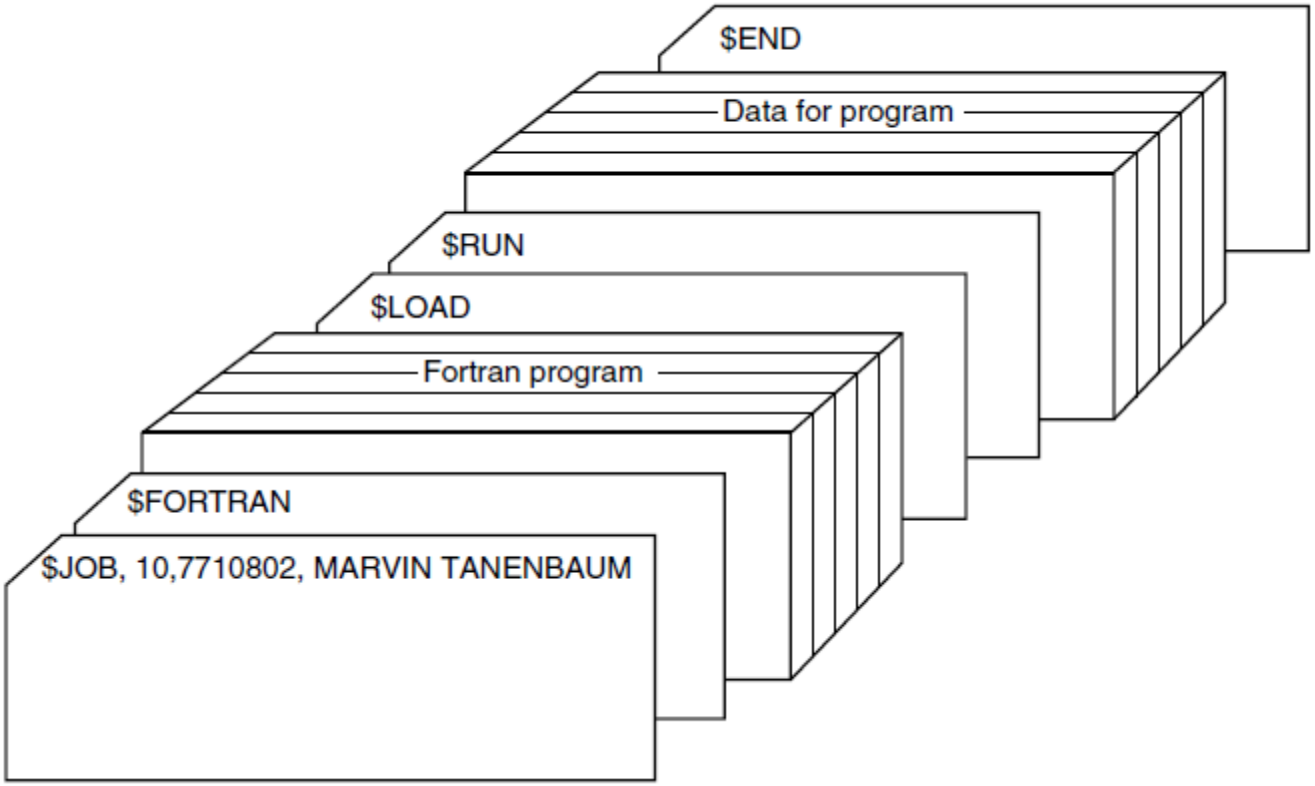
2 problems

no protection – what if program has a bug and crashes the batch monitor waste time rebooting

computer idle during I/O



Batch Systems



Structure of a typical job.

History of Operating Systems

3) Data channels, interrupts: overlap I/O and computation

DMA – direct memory access for I/O devices.
OS requests I/O, goes back to computing, gets interrupt when I/O device finishes

4) **Memory protection + relocation**

Multiprogramming – several programs run at same time; users share the system

Multiprogramming benefits

- Small jobs not delayed by large jobs
- more overlap between I/O and CPU

History of Operating Systems

Multiprogramming requires memory protection to keep bugs in one program from crashing the system or corrupting other programs

Bad news: OS must manage all these interactions between programs.

Each step seems logical, but at some point – just gets too complicated

- Multics – announced in 1963; ran in 1969
- OS360 released with 1000 bugs

UNIX based on multics, but simplified so they could get it to work!

Multics (Multiplexed Information and Computing Service)

History of Operating Systems

History Phase 2: Hardware cheap, humans expensive

5) **interactive time sharing**

Use cheap terminals to let multiple users interact with the system at the same time.

Sacrifice CPU time to get better response time for users

OS does timesharing to give illusion of each user has own computer

History Phase 3: Hardware very cheap, humans very expensive

6) Personal computing

Computers are cheap, so give everyone a computer.

Initially, OS became a subroutine library again (MSDos, MacOS)

Since then, adding back in memory protection, multiprogramming, etc. (when humans are expensive, don't waste their time by letting programs crash each other)

History phase 4: Distributed systems

Computers so cheap – give people a bunch of them (PC at home, 2 in office, a portable, and share some machines in a lab)

how do I coordinate a bunch of machines?

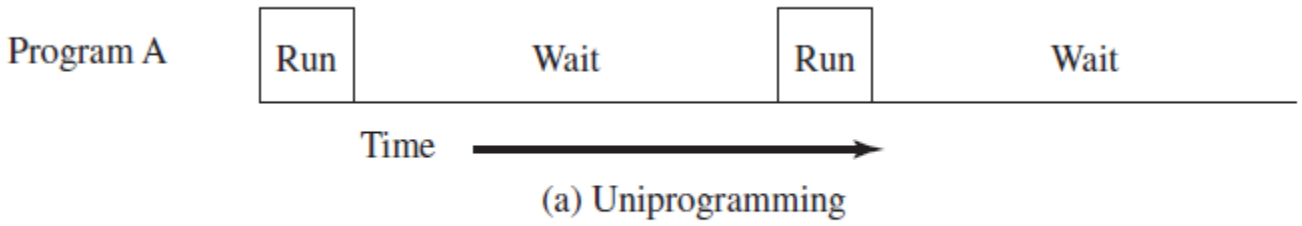
Networks fast – allow machines to share resources and data easily
Networks cheap – allow geographically distributed machines to
interact

Question: What does all this mean to OS?

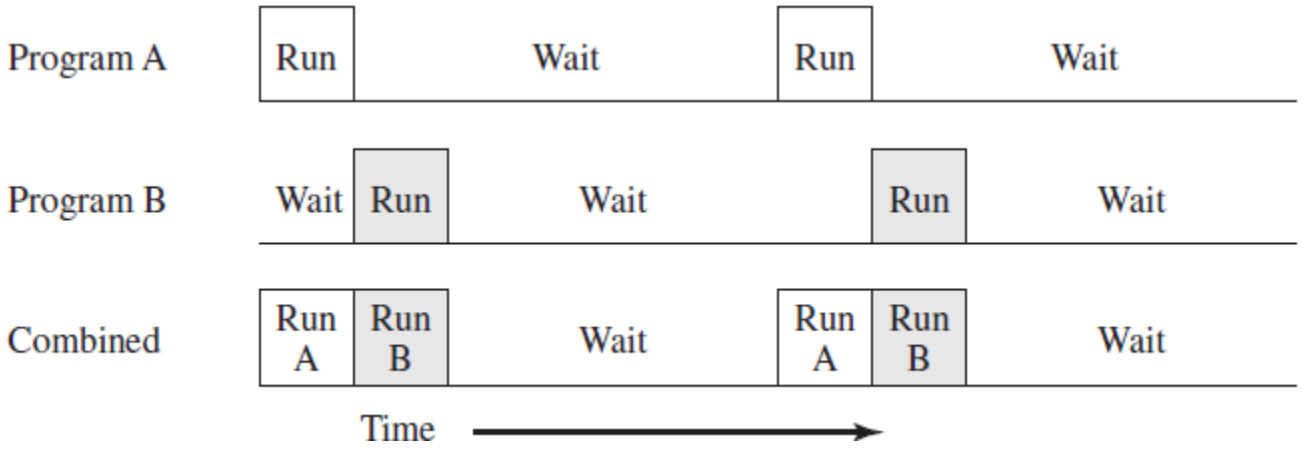
Key Features of Classes of Operating Systems

OS class	Period	Prime concern	Key concepts
Batch processing	1960s	CPU idle time	Automate transition between jobs
Multiprogramming	1960s	Resource utilization	Program priorities, preemption
Time-sharing	1970s	Good response time	Time slice, round-robin scheduling
Real time	1980s	Meeting time constraints	Real-time scheduling
Distributed	1990s	Resource sharing	Distributed control, transparency

System Utilization



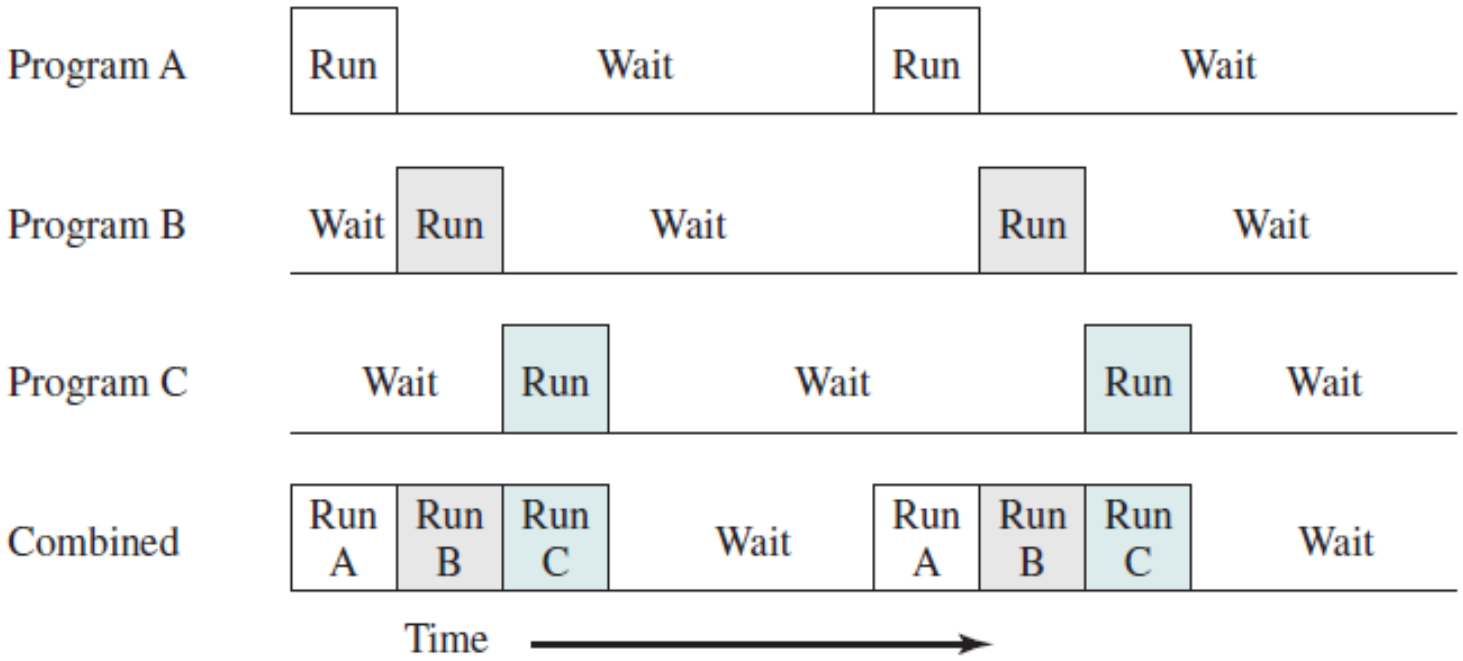
(a) Uniprogramming



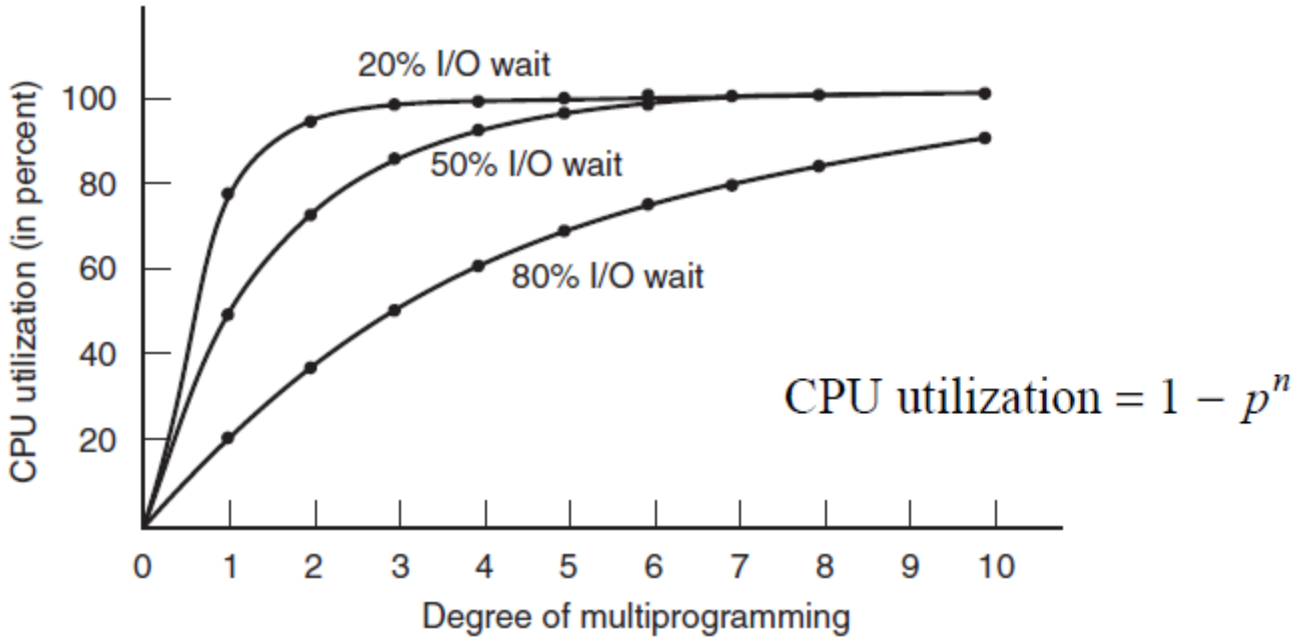
Read one record from file	15 μs
Execute 100 instructions	1 μs
Write one record to file	15 μs
Total	31 μs

$$\text{Percent CPU utilization} = \frac{1}{31} = 0.032 = 3.2\%$$

Multiprogramming with three programs



Modeling Multiprogramming



CPU utilization as a function of the number of processes in memory.

Summary

1) Key ideas: coordination (resource management, isolation), abstraction

2) Application of ideas changes

a. Over time
b. Across applications, services (not just kernels)

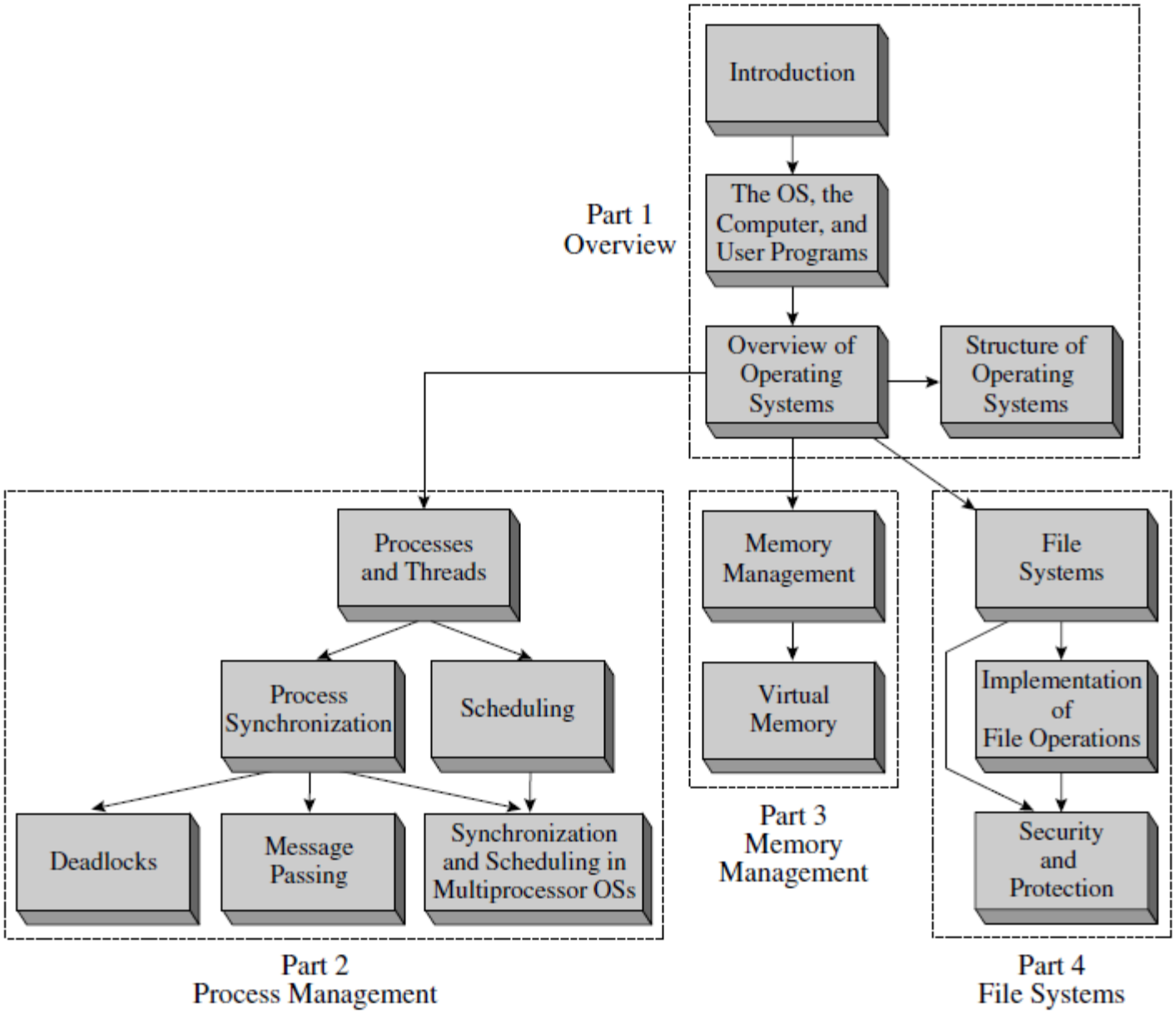
Point is: have to change with changing technology
OS's today are enormous, complex things

small OS – 100K lines
big OS – 50M lines
100 - 1000 people-year

Key aspect of this course, understand fundamentals OS design

Computer Architecture Basics

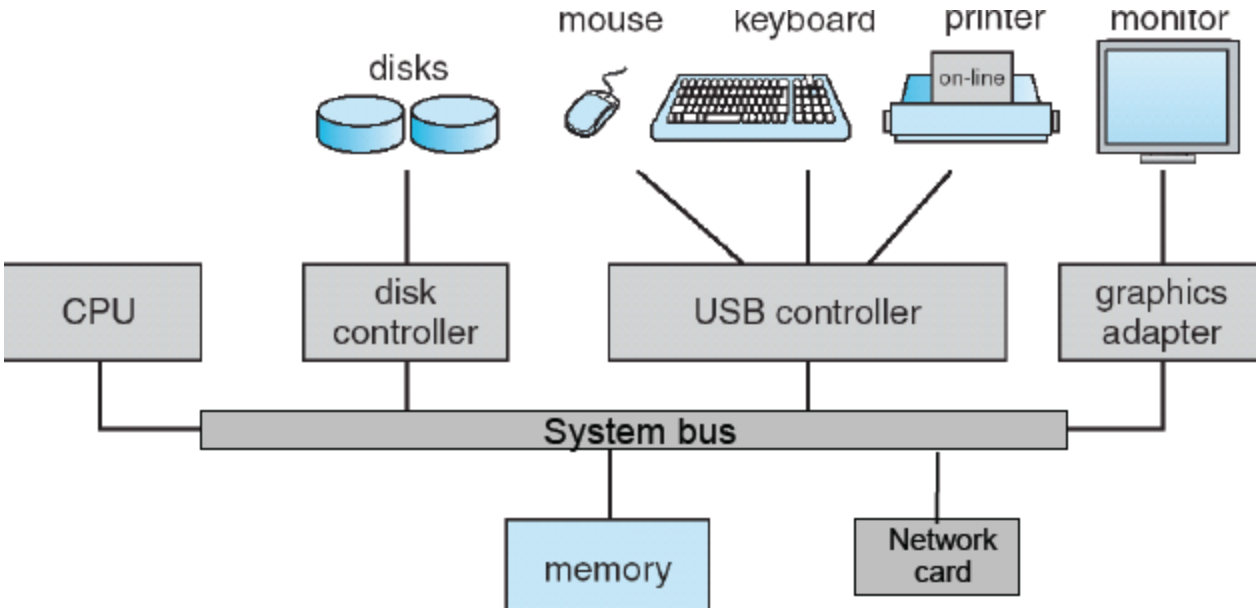
Overview


Introduction
The OS, the Computer, and User Programs
Part 1 Overview
Overview of Operating Systems
Structure of Operating Systems
Processes and Threads
Process Synchronization
Scheduling
Deadlocks
Message Passing
Synchronization and Scheduling in Multiprocessor OSs
Part 2 Process Management
Memory Management
Virtual Memory
Part 3 Memory Management
File Systems
Implementation of File Operations
Security and Protection
Part 4 File Systems

Architectural Features Motivated by OS Services

OS Service	Hardware Support
Protection	Kernel/user mode, protected instructions, base/limit registers
Interrupts	Interrupt vectors
System calls	Trap instructions and trap vectors
I/O	Interrupts and memory mapping
Scheduling, error recovery, accounting	Timer
Synchronization	Atomic instructions
Virtual memory	Translation look-aside buffers

Generic Computer Architecture



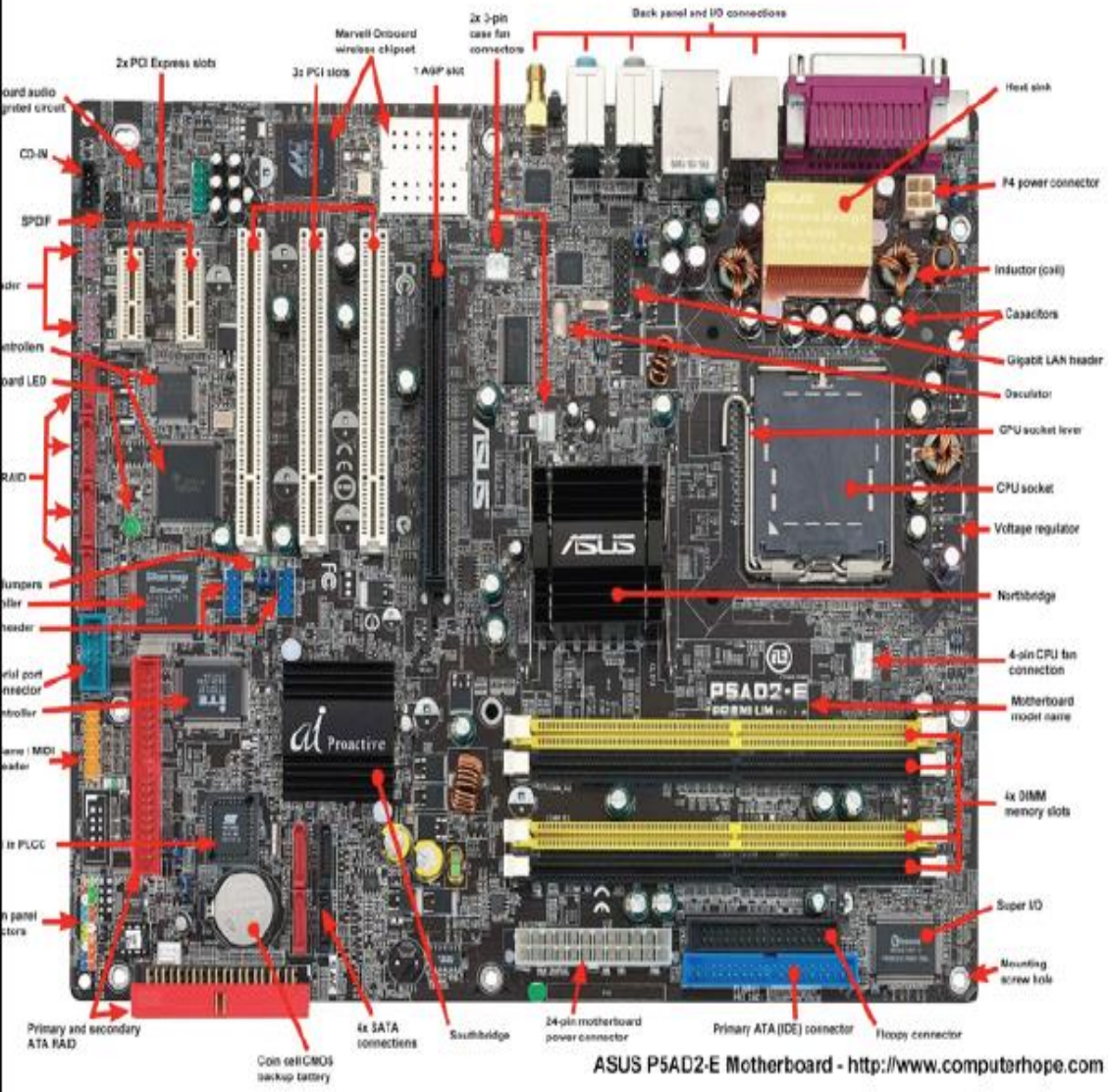
CPU: the processor that performs the actual computation

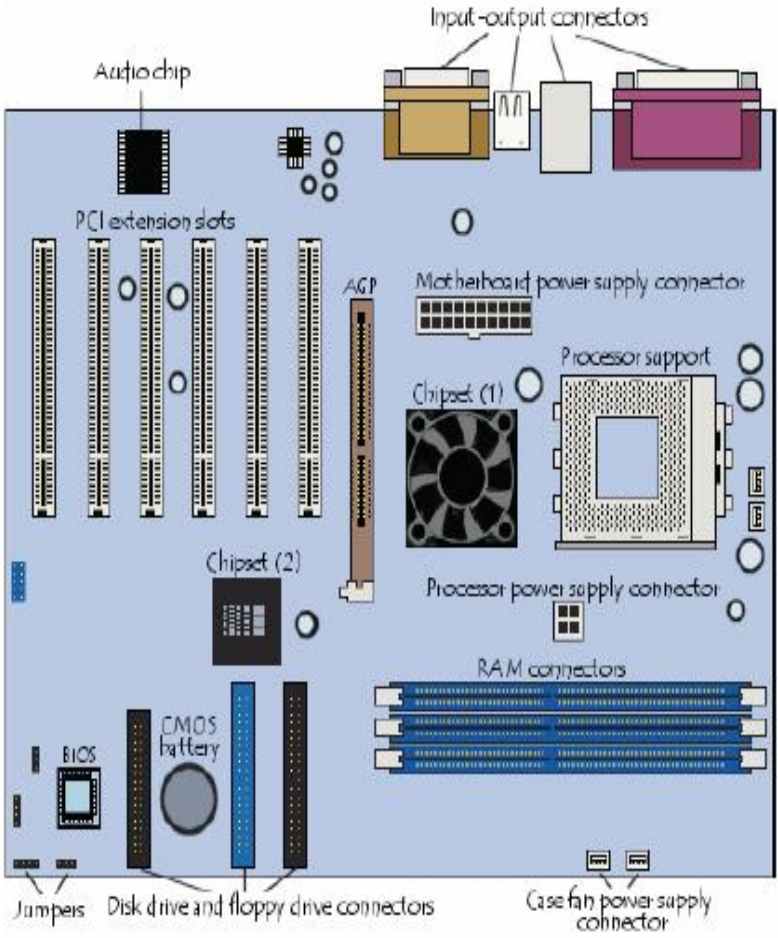
– Multiple "cores" common in today's processors

- **I/O devices:** terminal, disks, video board, printer, etc.

– Network card is a key component, but also an I/O device

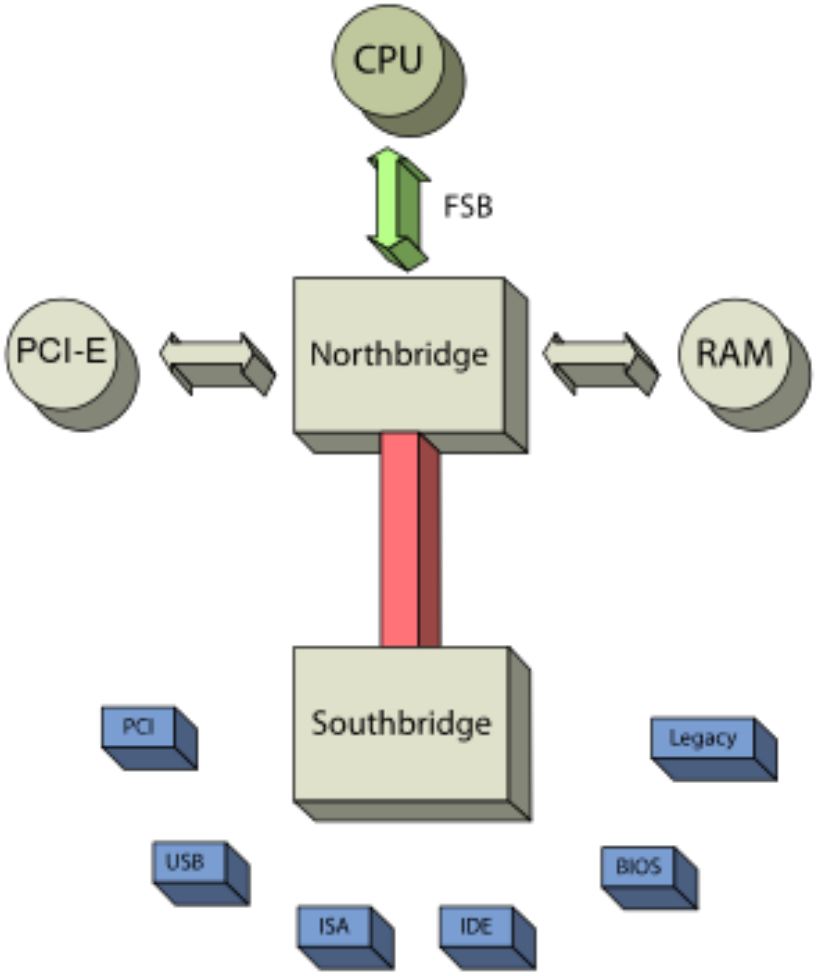
- **Memory:** RAM containing data and programs used by the CPU
- **System bus:** communication medium between CPU, memory, and peripherals





- Picture of a motherboard/logic board

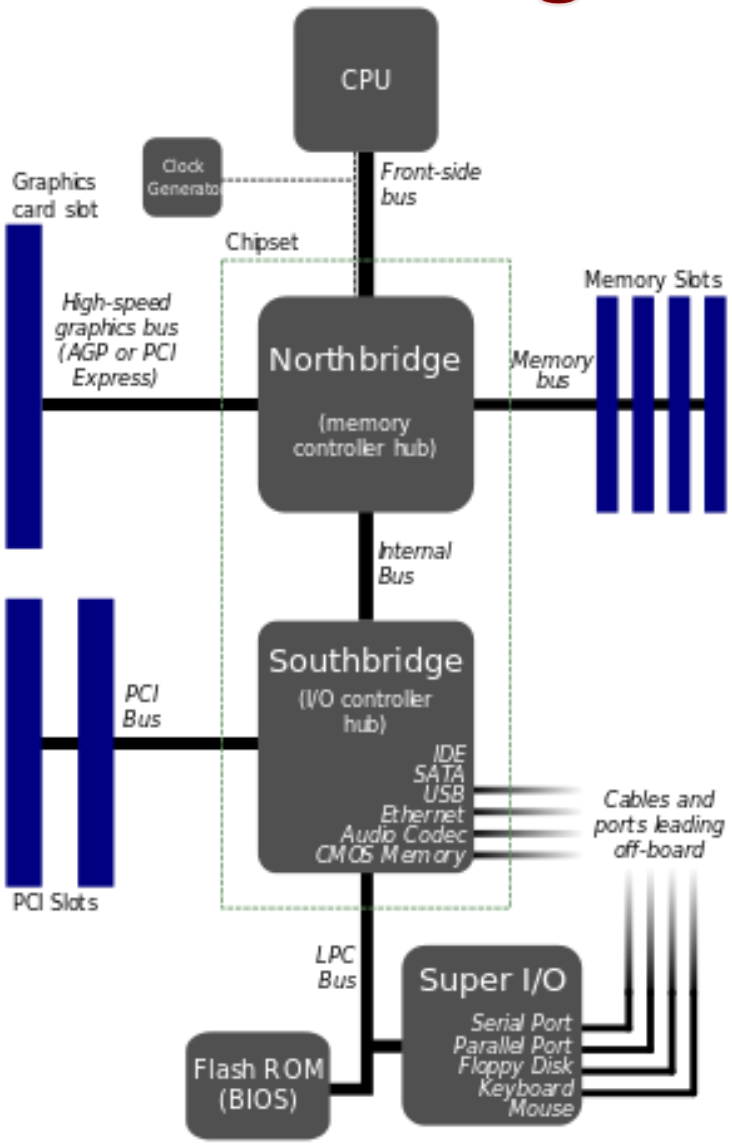
Typical PC Architecture: **northbridge** or **host bridge**



front-side bus (FSB)

I/O controller hub

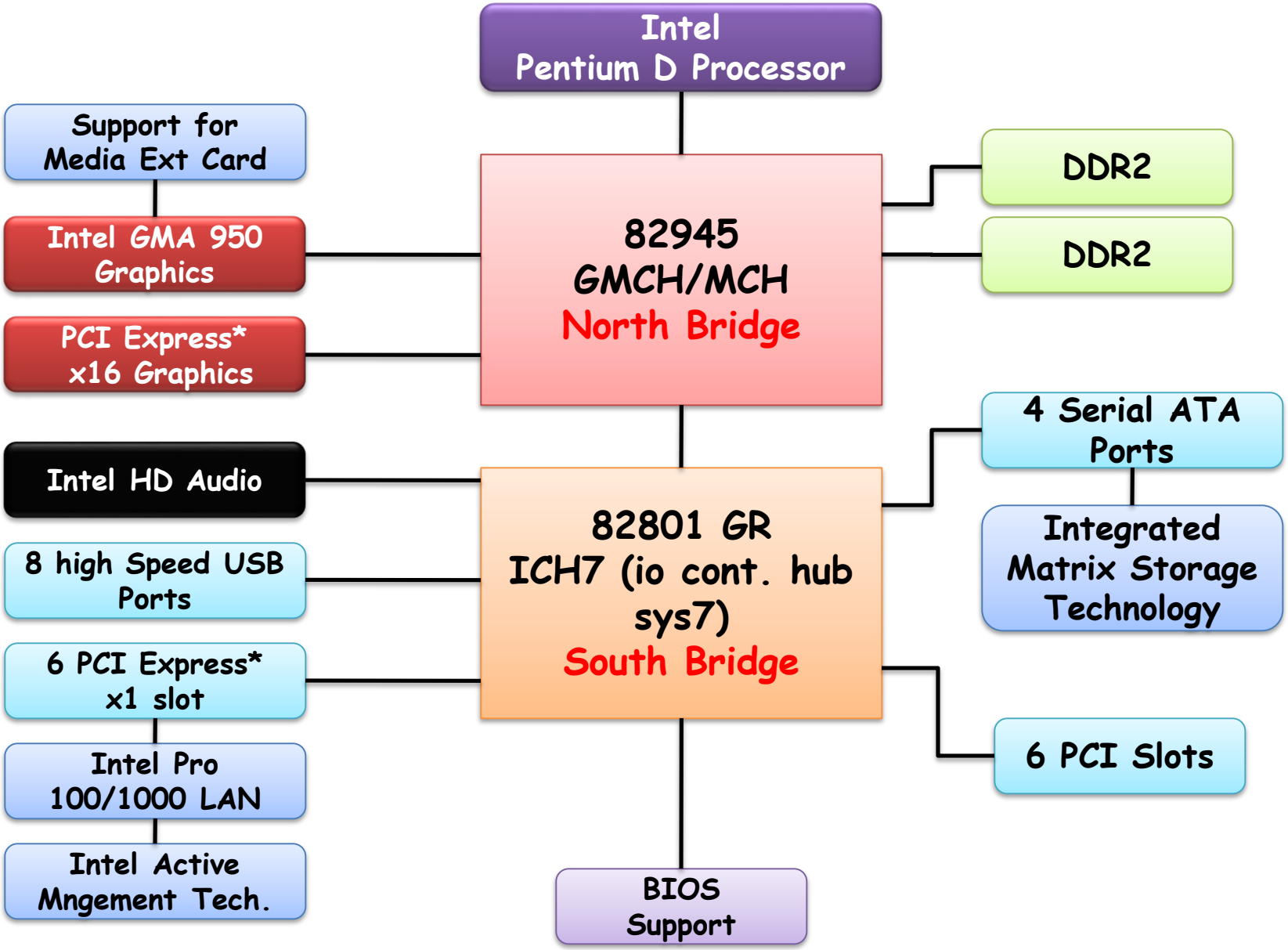
(Typical PC Architecture)
northbridge or **host bridge**



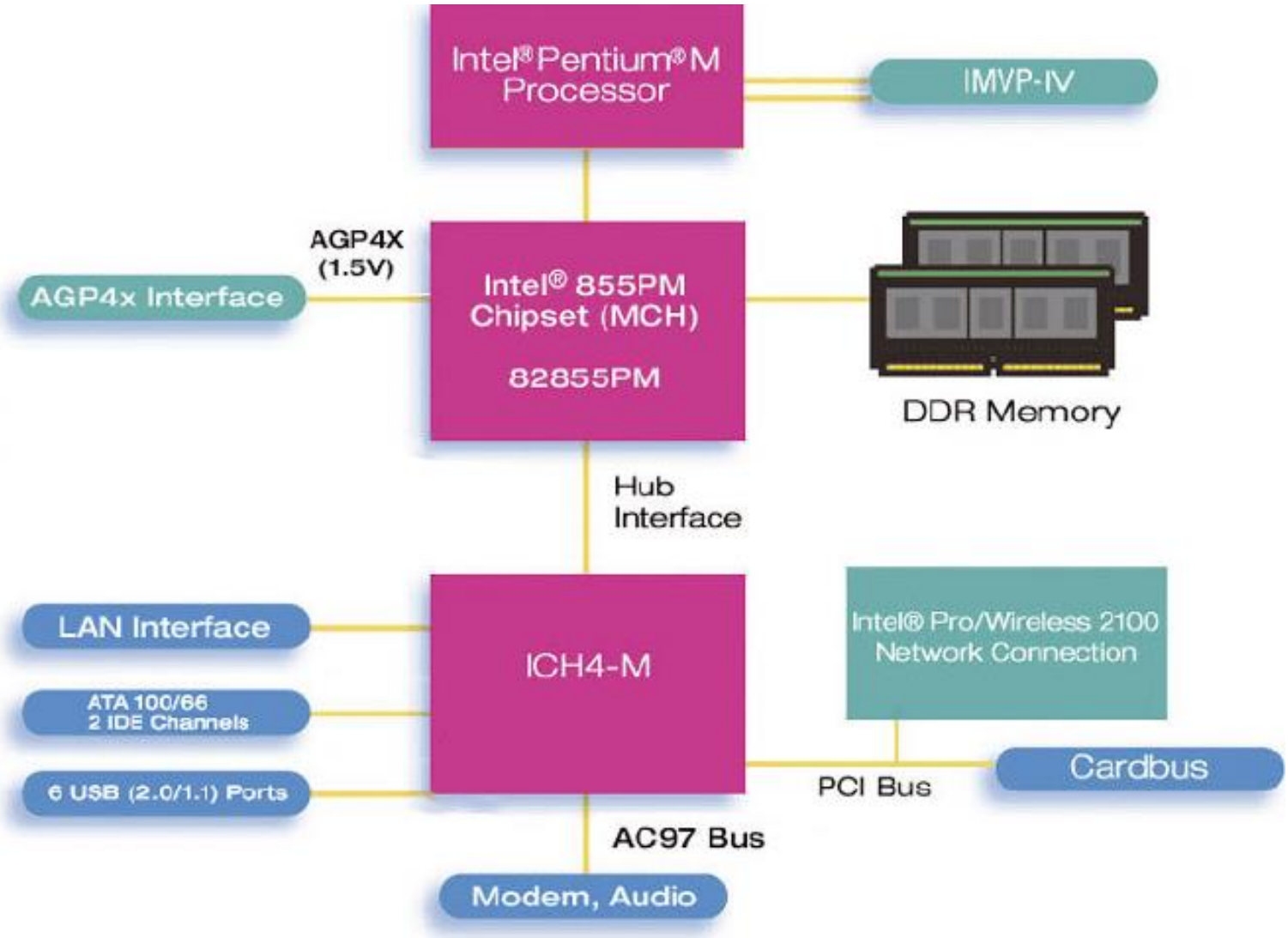
Graphics and Memory Controller Hu
(GMCH)

I/O Controller Hub (ICH)

Intel 945 Express Chipset

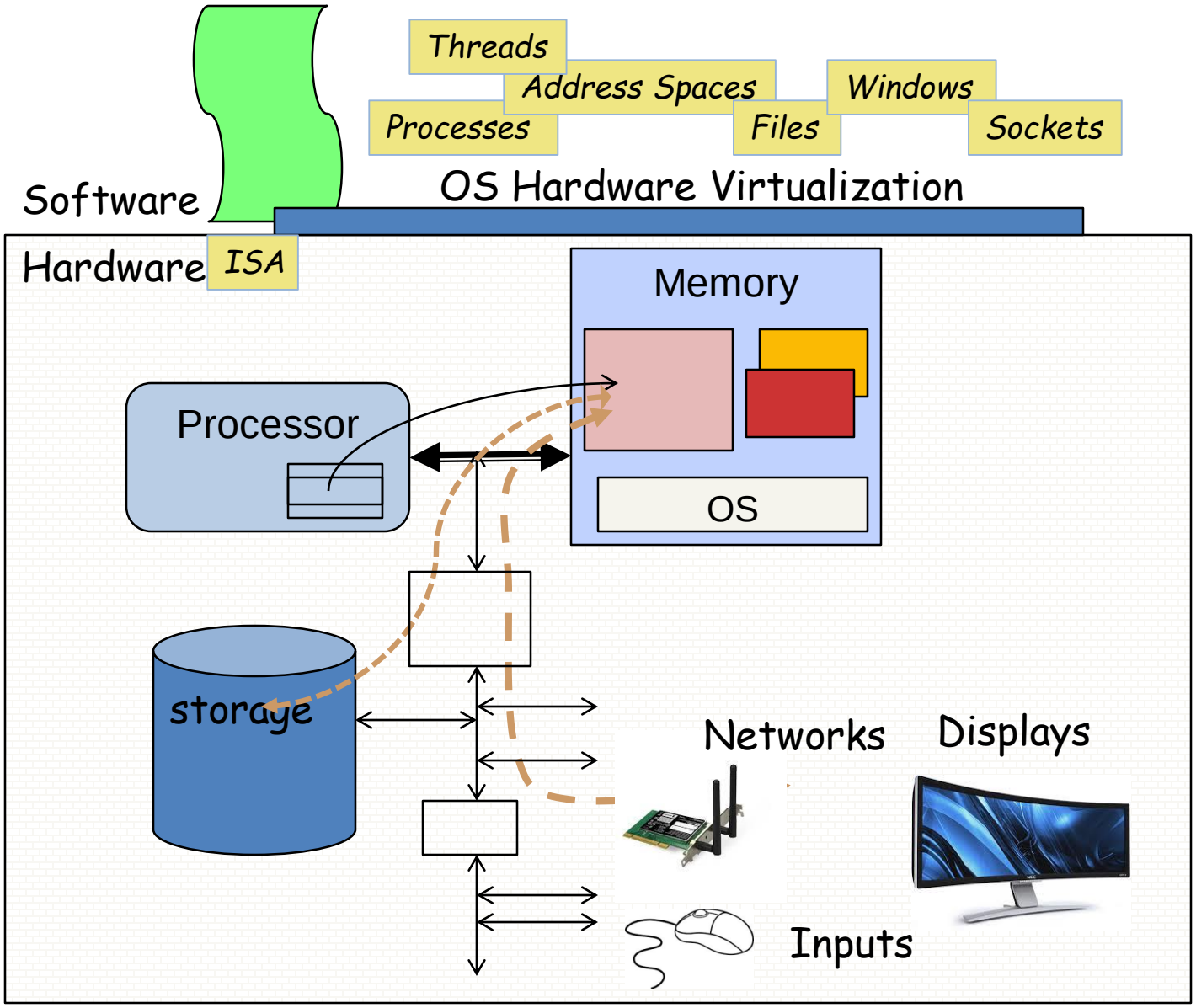


What is A Computer?



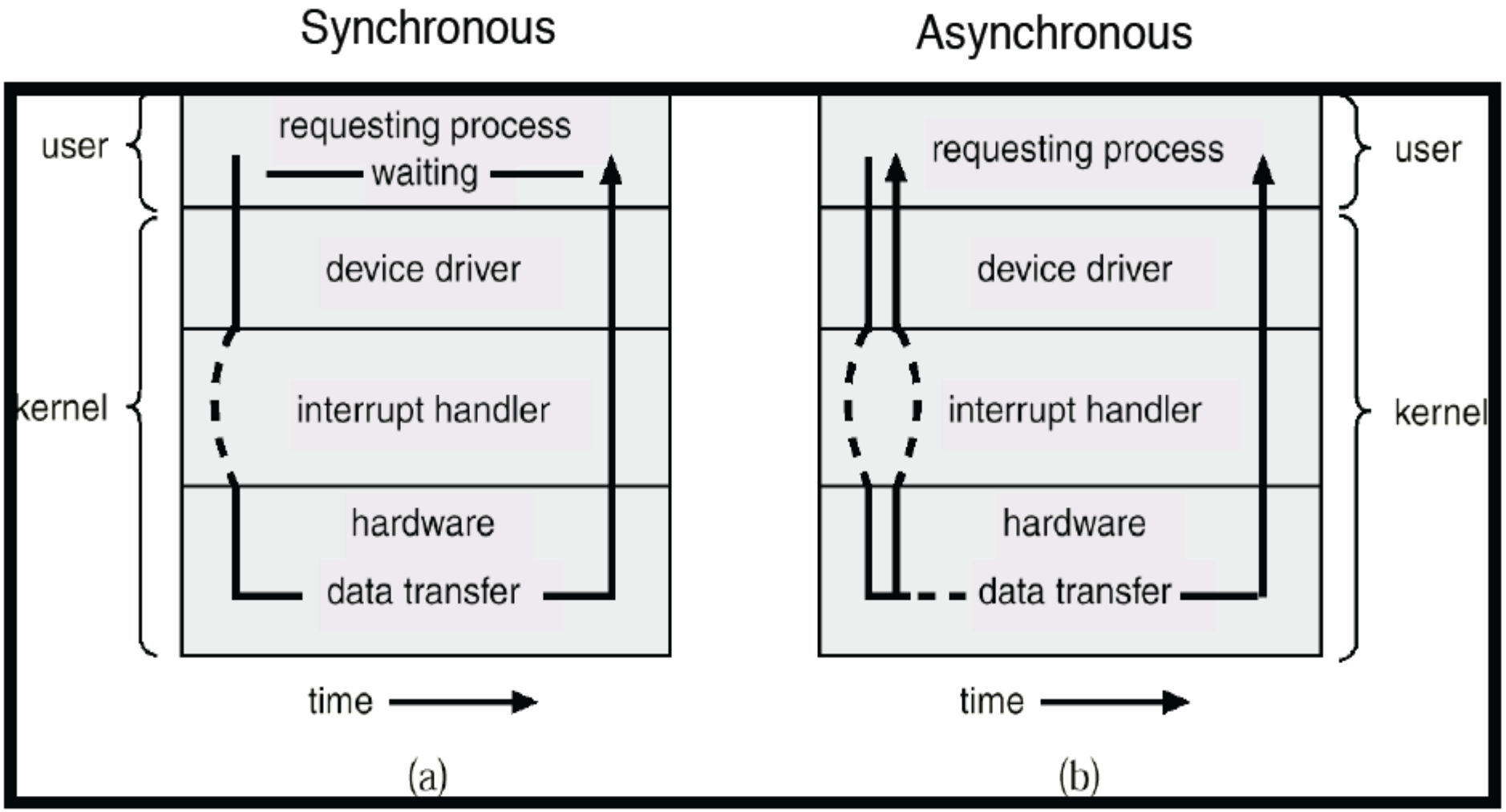
Courtesy, Intel

OS Basics: I/O


Threads
Address Spaces
Windows
Processes
Files
Sockets
Software
OS Hardware Virtualization
Hardware
ISA
Memory
Processor
OS
storage
Networks
Displays
Inputs

Three I/O Methods

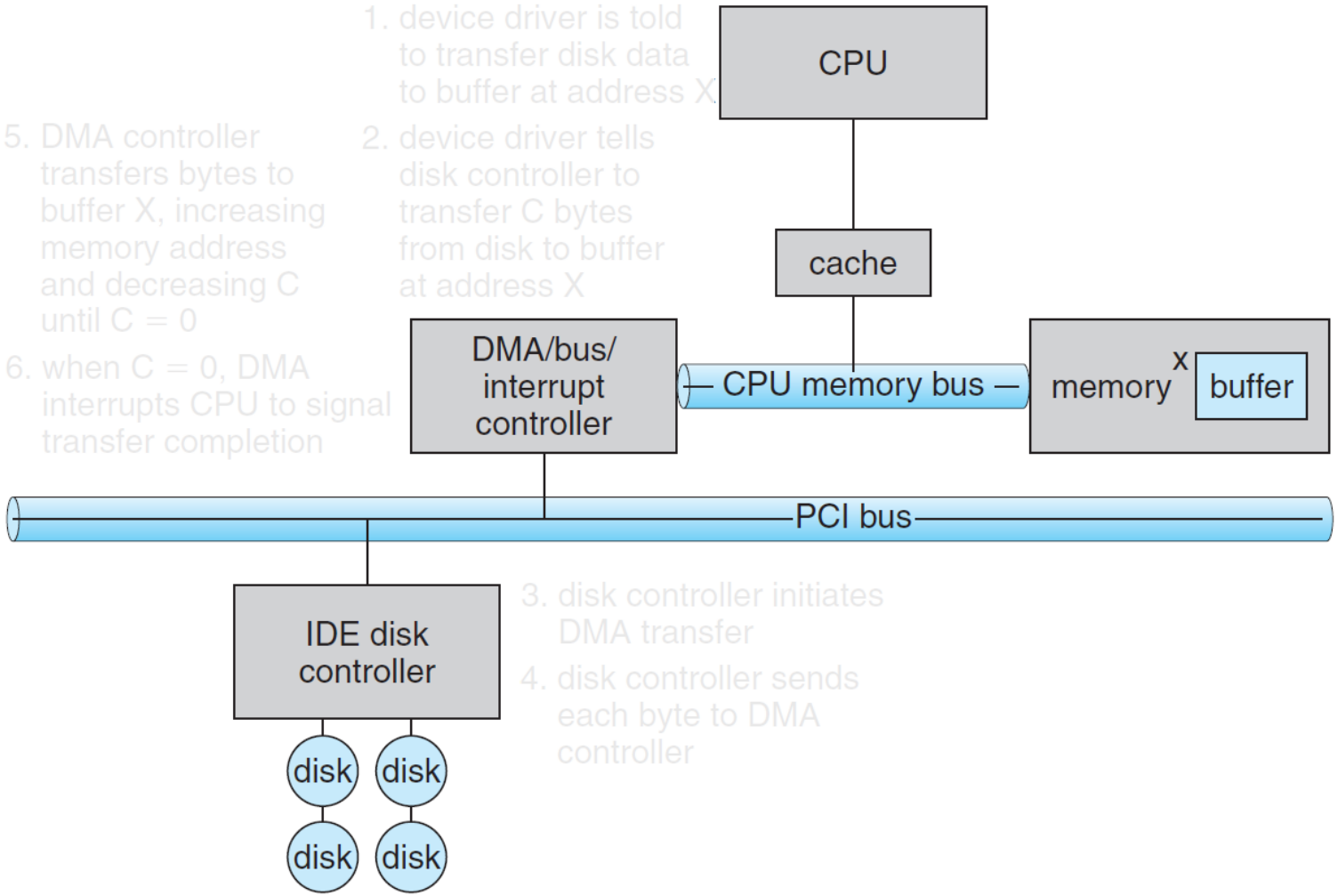
- Synchronous, asynchronous, memory-mapped



Memory-Mapped I/O

- Enables direct access to I/O controller (vs. being required to move the I/O code and data into memory)
- PCs reserve a part of the (physical) memory and put the device manager in that memory (e.g., all the bits for a video frame for a video controller).
- Access to the device then becomes almost as fast and convenient as writing the data directly into memory.

DMA transfer.

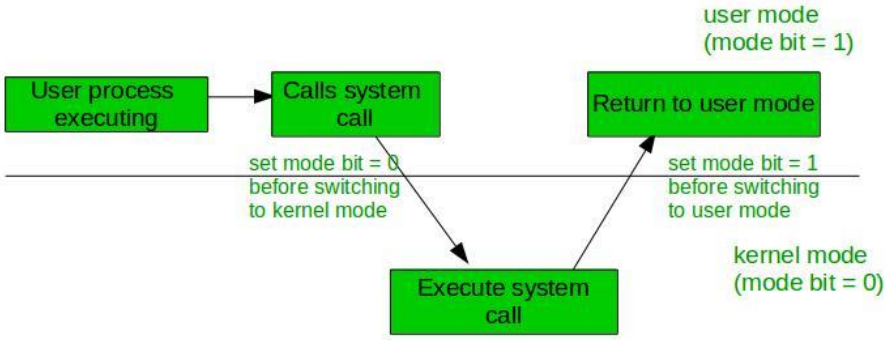


Nachos: Virtual OS Environment

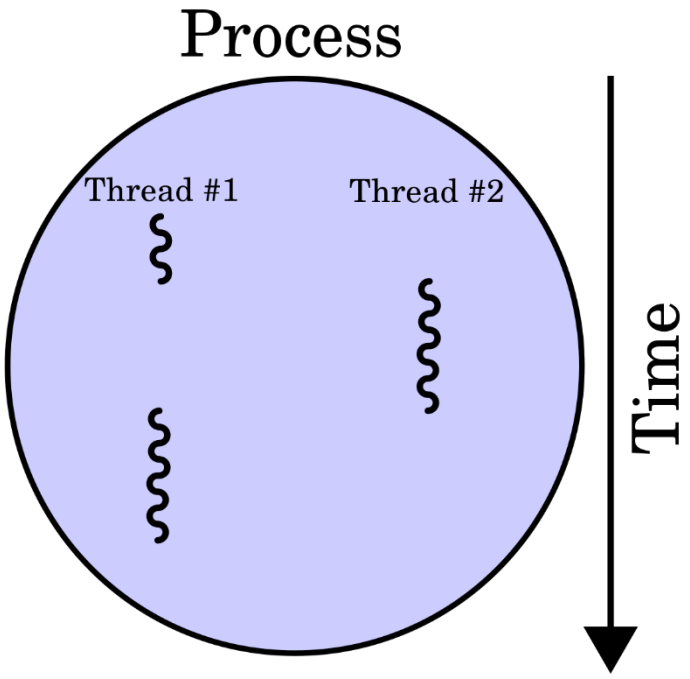
- Nachos
  - Designed by Thomas Anderson at UC Berkeley in 1992
  - Simulation environment: Hardware, interrupts, I/O
  - Execution of User Programs running on this platform

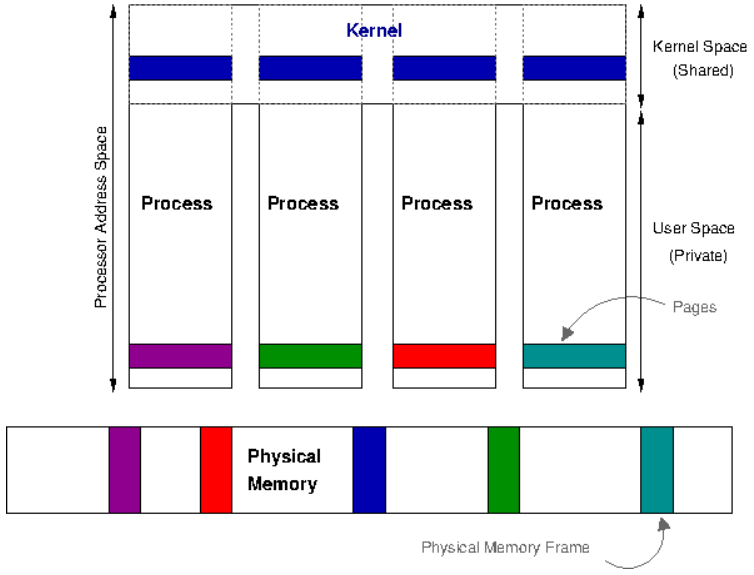
Educational OS that some components can be implemented by users

- Process management.
- CPU scheduling.
- Memory management.
- File system management.
- Networking.



Four Fundamental OS Concepts

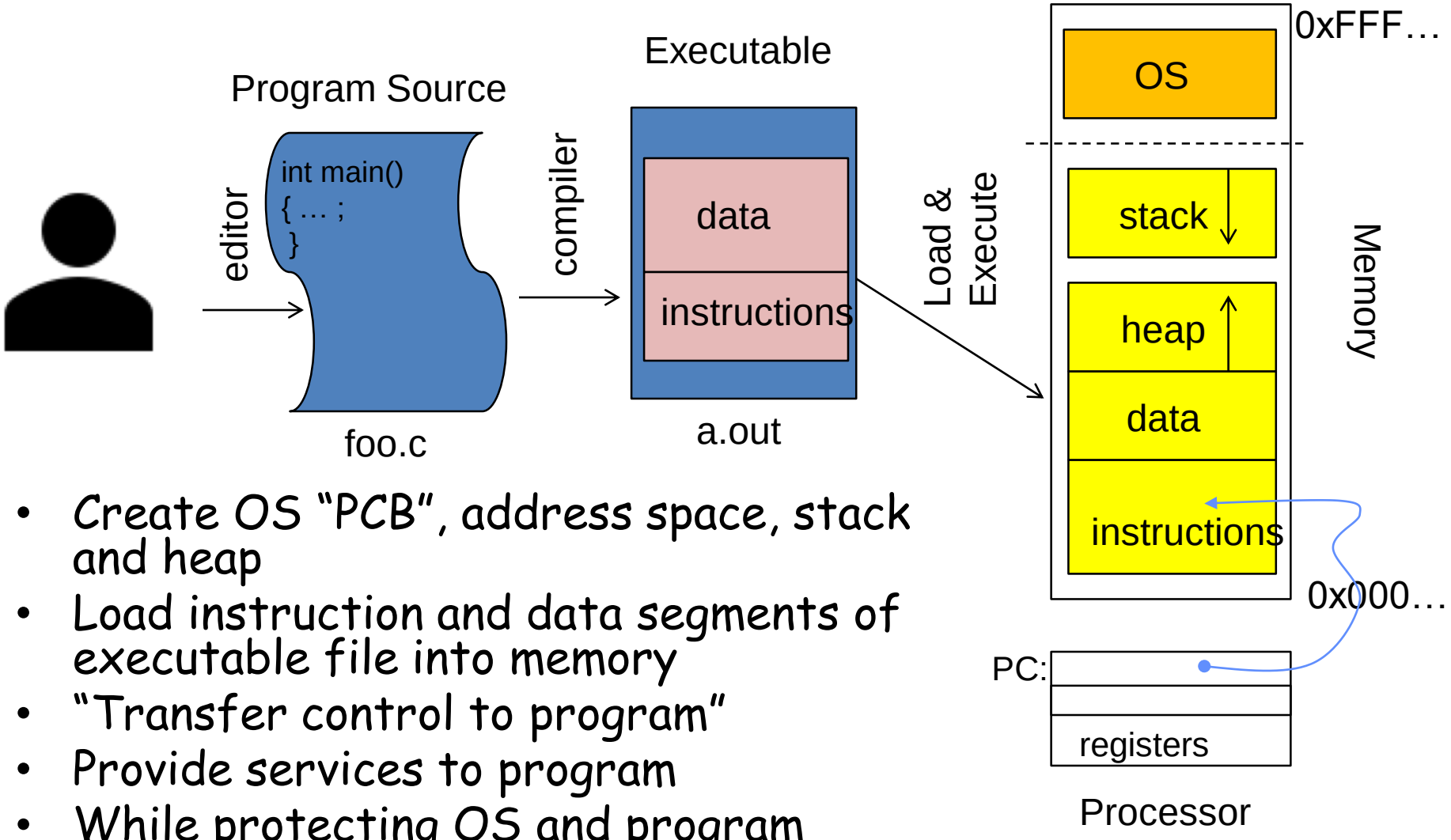




Four Fundamental OS Concepts

- **Thread: Execution Context**
  - Program Counter, Registers, Execution Flags, Stack
- **Address space**
  - Program's view of memory is distinct from physical machine
- **Process: an instance of a running program**
  - Address Space + One or more Threads + …
- **Dual mode operation / Protection**
  - Only the "system" can access certain resources
  - Combined with translation, isolates programs from each other

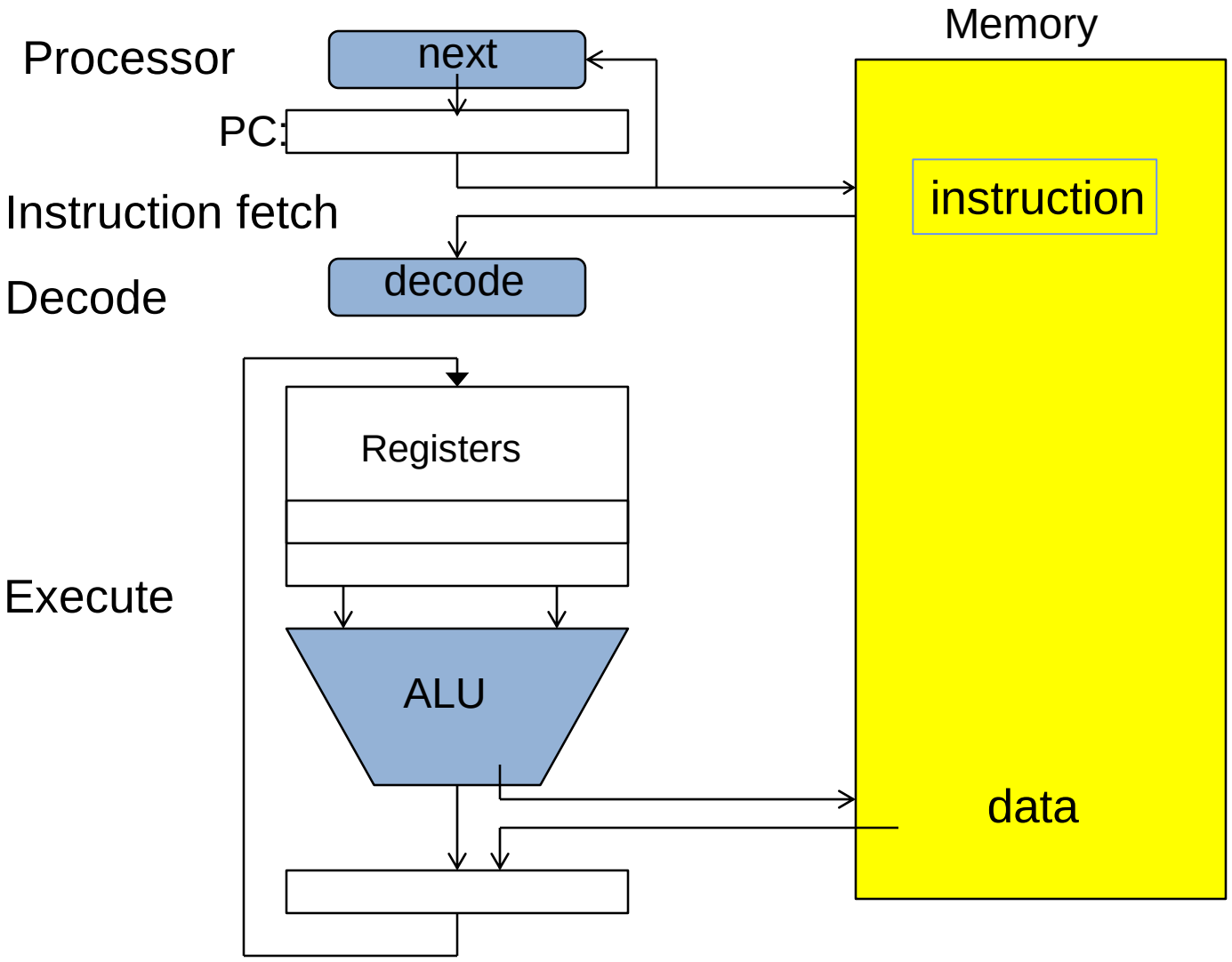
OS Bottom Line: Run Programs



- Create OS "PCB", address space, stack and heap
- Load instruction and data segments of executable file into memory
- "Transfer control to program"
- Provide services to program
- While protecting OS and program

Instruction Fetch/Decode/Execute

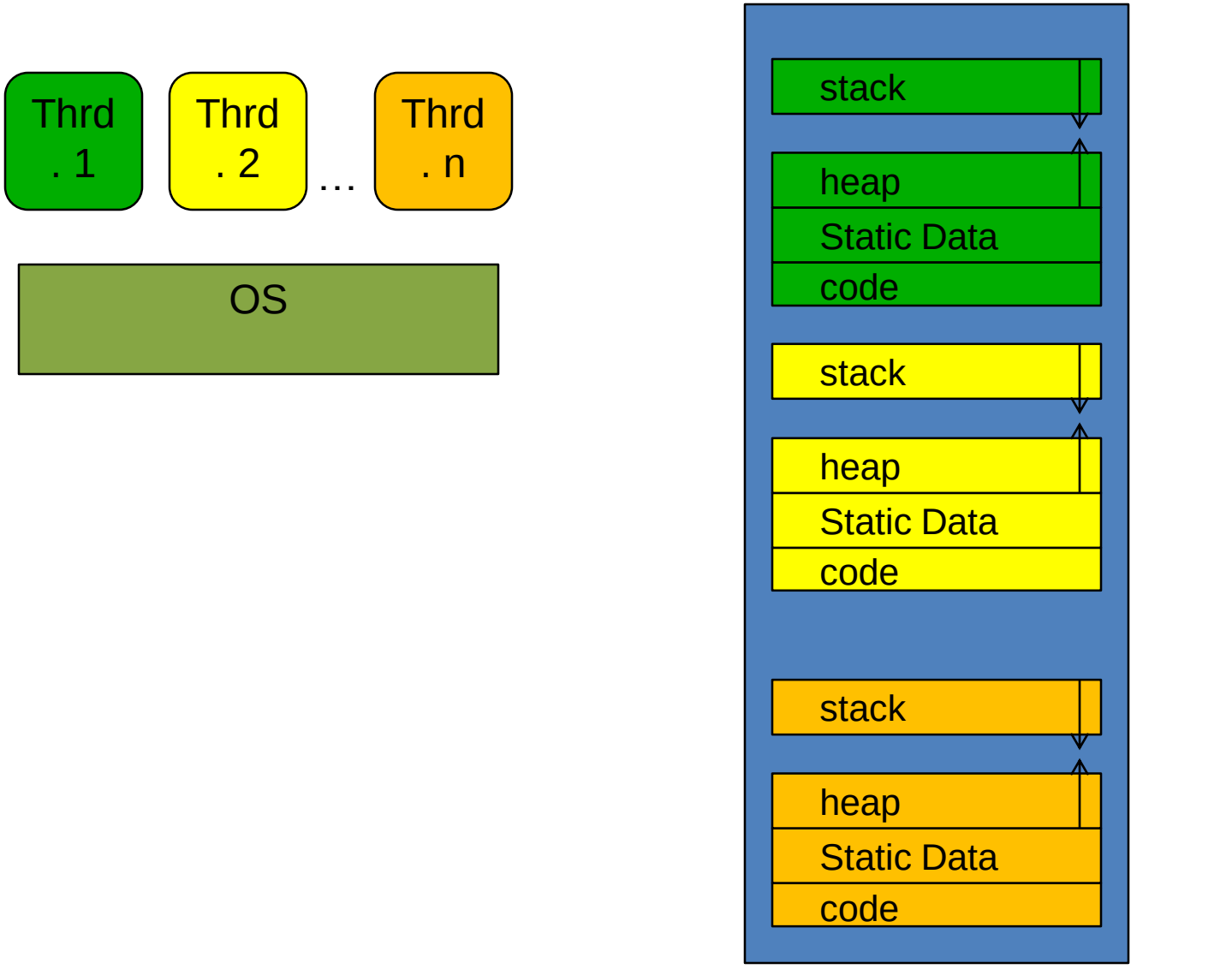
The instruction cycle



Thread

- Definition: **A single, unique execution context**
  - Program counter, registers, stack
- A thread is *executing* on a processor (core) when it is resident in that processor's registers
- Registers hold the root state of the thread:
  - The rest is "in memory"
  - Including program counter & currently executing instruction

Multiprogramming - Multiple Threads of Control

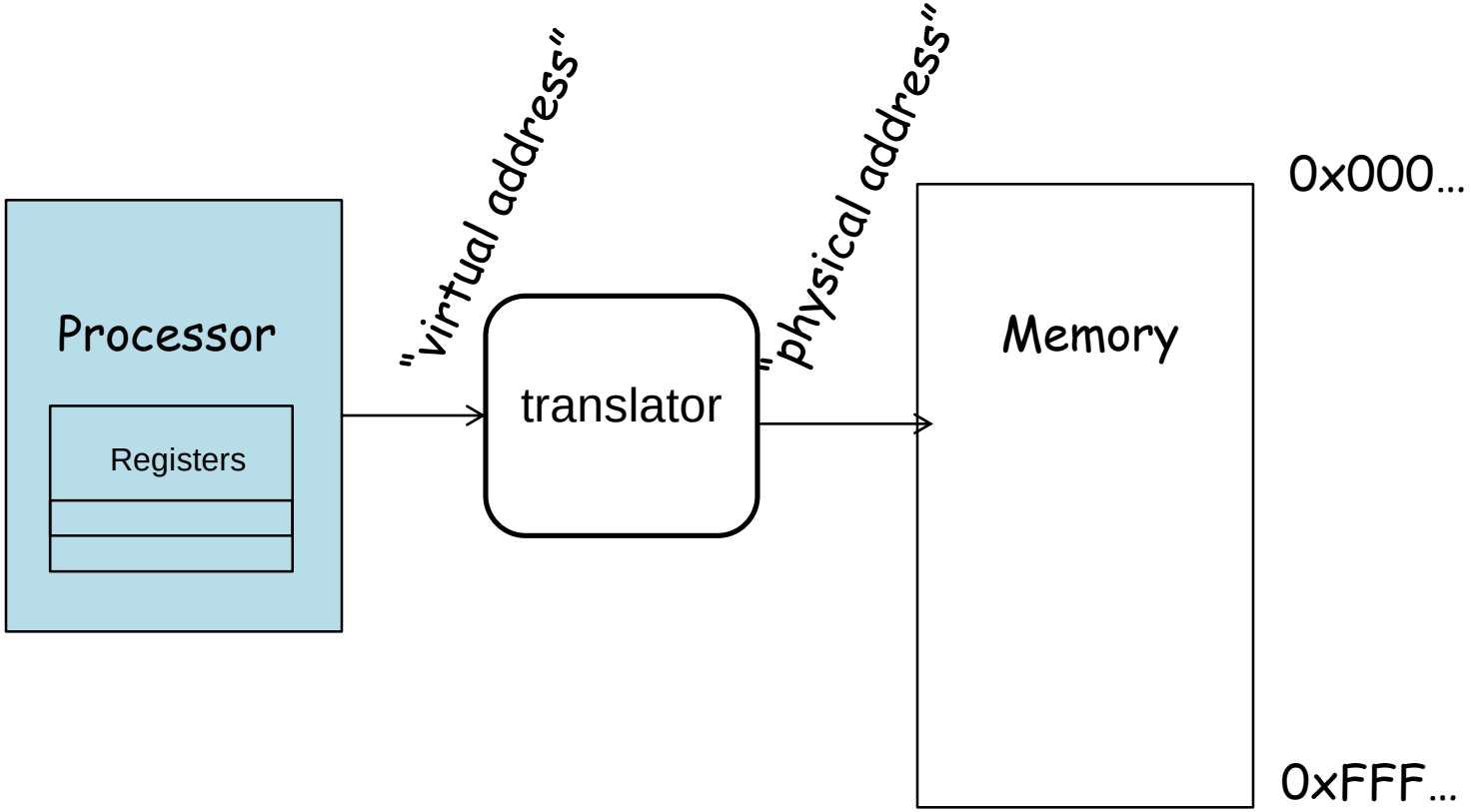


Four Fundamental OS Concepts

- **Thread: Execution Context**
  - Program Counter, Registers, Execution Flags, Stack
- **Address space** (with **translation**)
  - Program's view of memory is distinct from physical machine
- **Process: an instance of a running program**
  - Address Space + One or more Threads
- **Dual mode operation / Protection**
  - Only the “system” can access certain resources
  - Combined with translation, isolates programs from each other

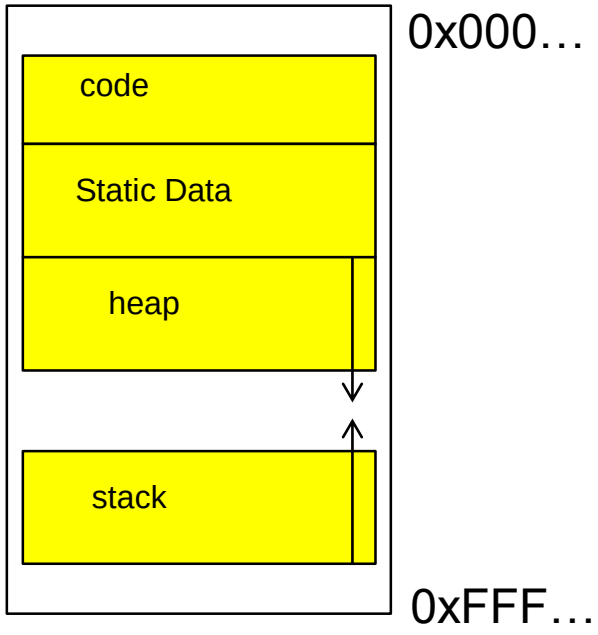
Key OS Concept: Address Space

- Program operates in an address space that is distinct from the physical memory space of the machine

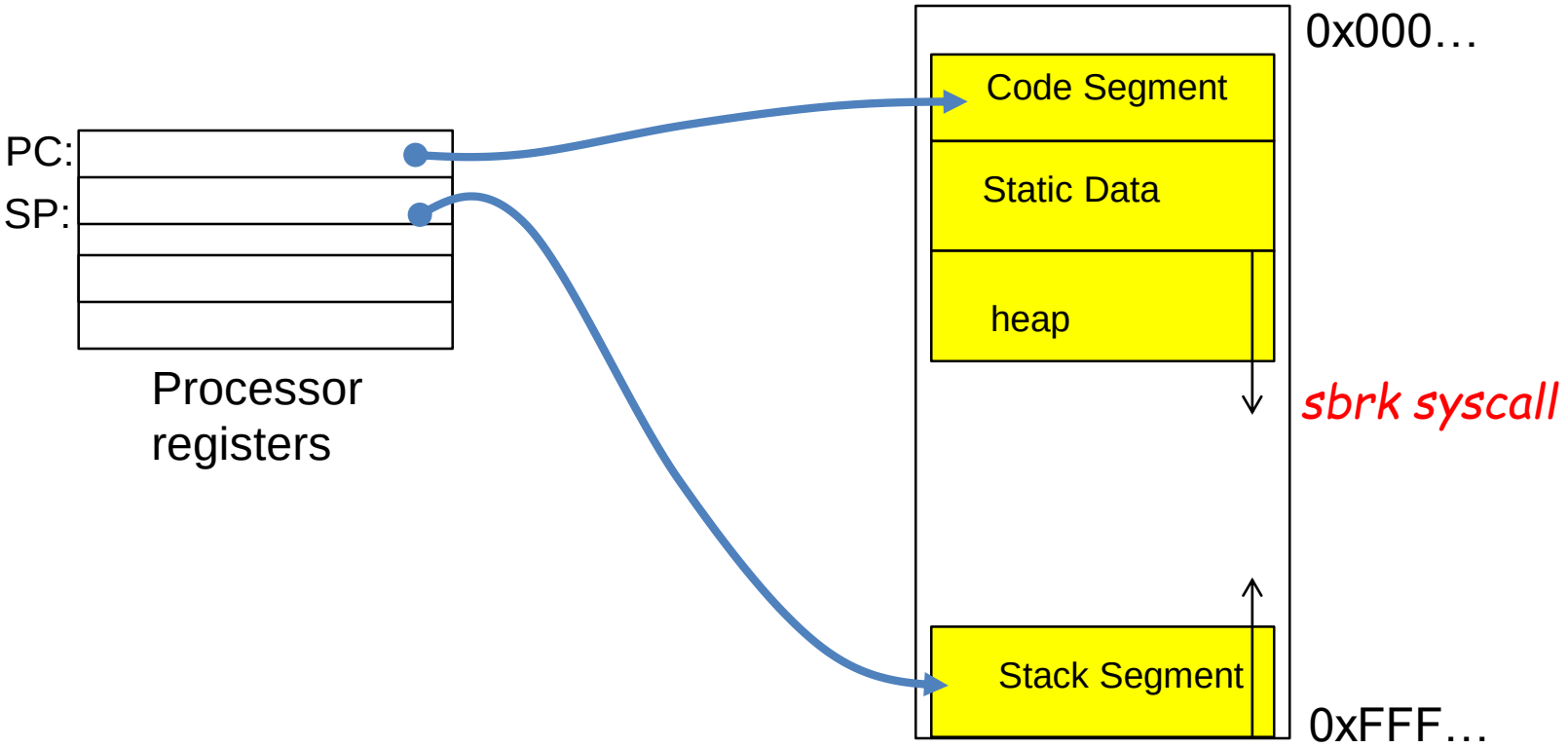


Address Space

- Definition: **Set of accessible addresses and the state associated with them**
  - 2^{32} = ~4 billion on a 32-bit machine
- What happens when you read or write to an address?
  - Perhaps acts like regular memory
  - Perhaps causes I/O operation
    - (Memory-mapped I/O)
  - Causes program to abort (segfault)?
  - Communicate with another program



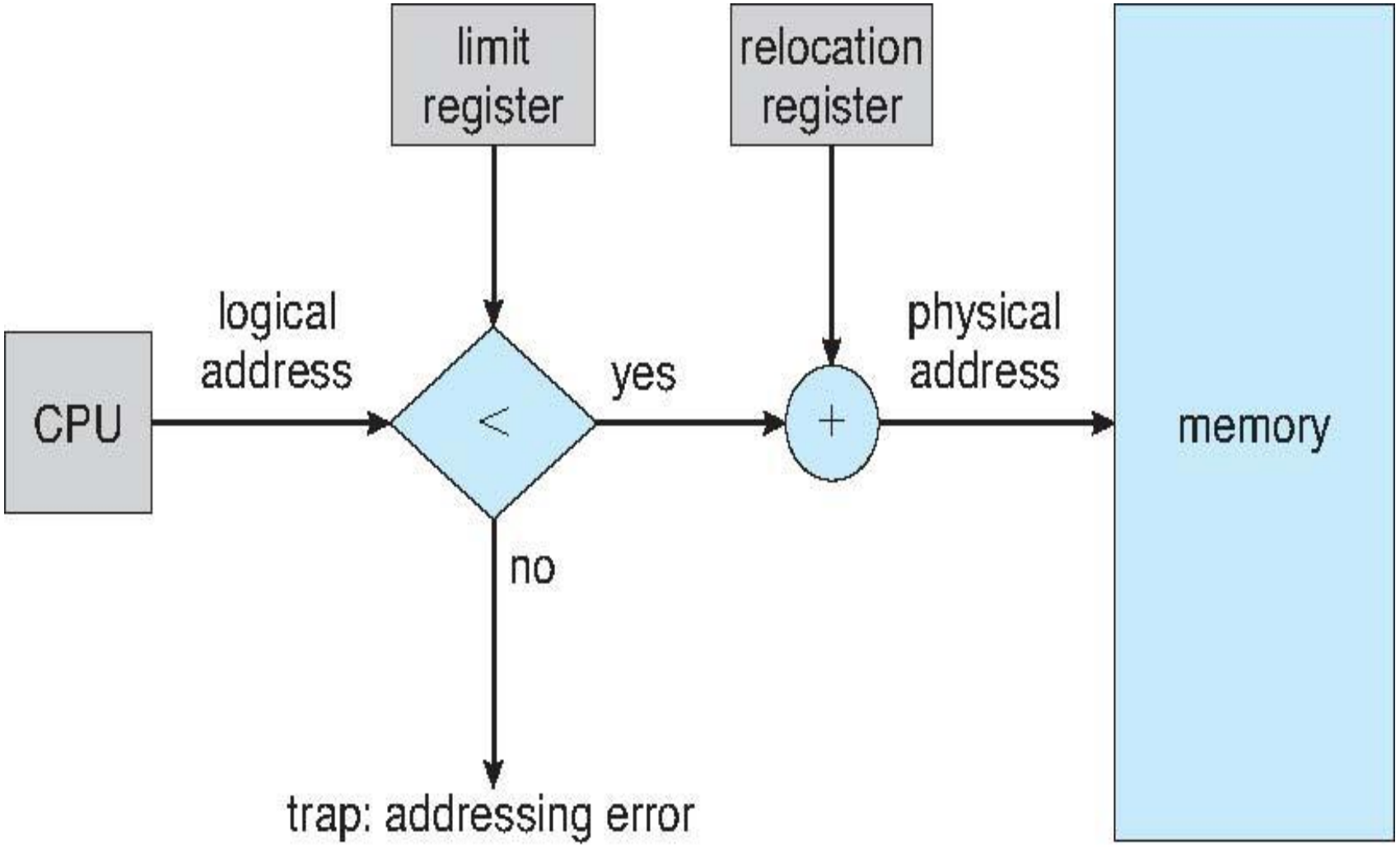
Address Space: typical structure



What can the hardware do to help the operating system protect itself from programs?

Programs from others?

Hardware Support for Relocation and Limit Registers

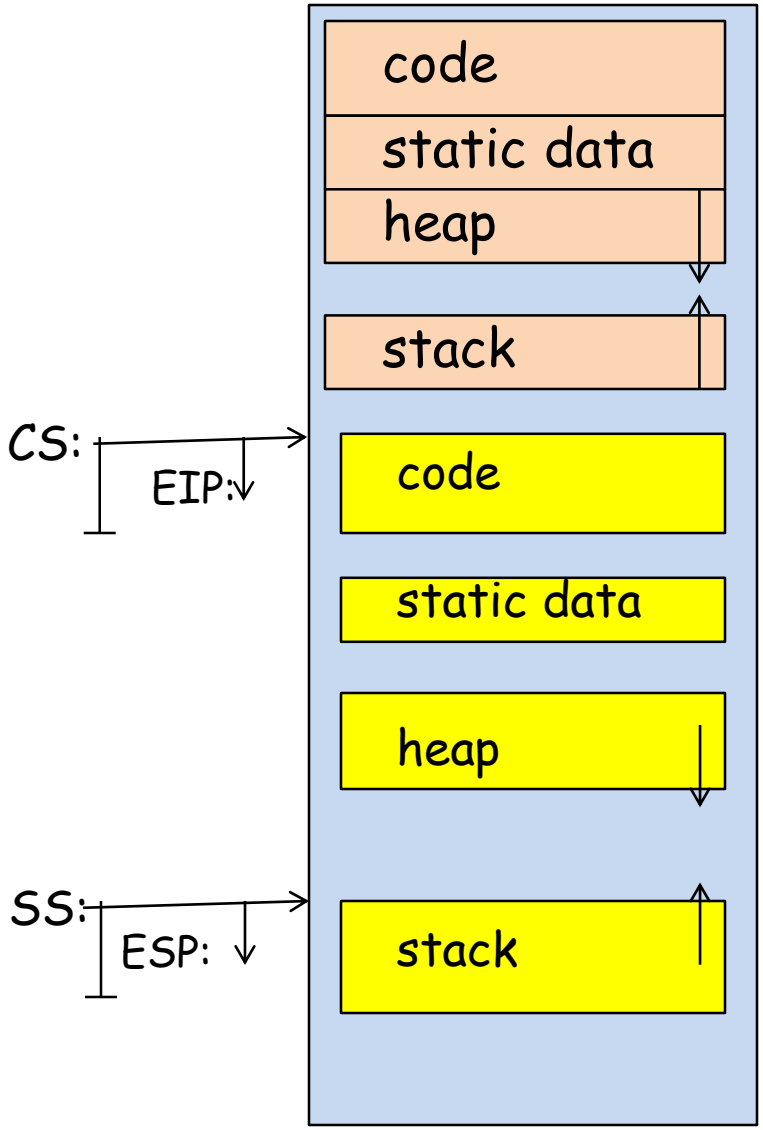


x86 – segments and stacks

Processor Registers

CS	EIP
SS	ESP
DS	EAX
ES	EBX
	ECX
	EDX
	ESI
	EDI

Start address, length and access rights associated with each segment register



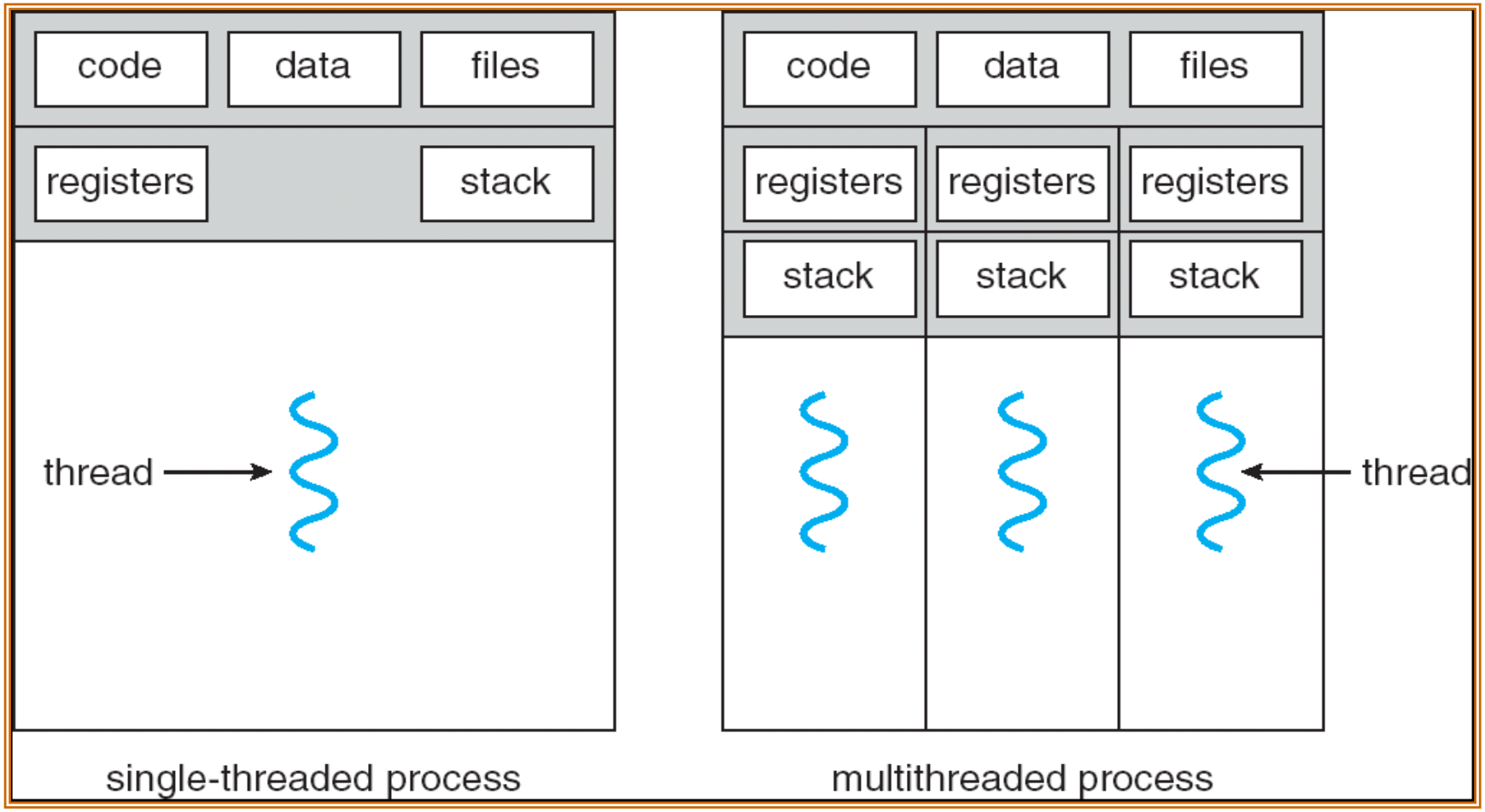
Four Fundamental OS Concepts

- **Thread: Execution Context**
  - Program Counter, Registers, Execution Flags, Stack
- **Address space** (with **translation**)
  - Program's view of memory is distinct from physical machine
- **Process: an instance of a running program**
  - Address Space + One or more Threads
- **Dual mode operation / Protection**
  - Only the "system" can access certain resources
  - Combined with translation, isolates programs from each other

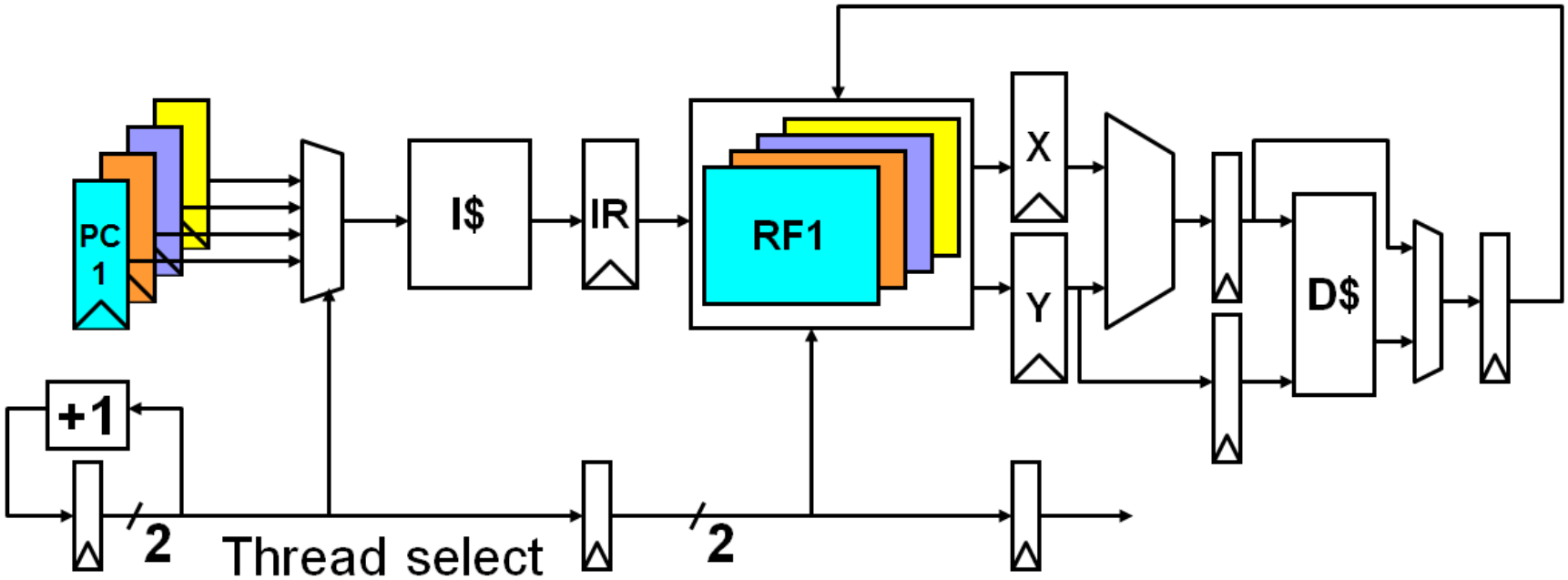
The Process

- Definition: **execution environment with restricted rights**
  - Address Space with One or More Threads
  - Owns memory (mapped pages)
  - Owns file descriptors, file system context, …
  - Encapsulates one or more threads sharing process resources
- Application program executes as a process
  - Complex applications can fork/exec child processes
- Why processes?
  - Protected from each other. OS Protected from them.
  - Execute concurrently
  - Basic unit OS deals with

Single and Multithreaded Processes



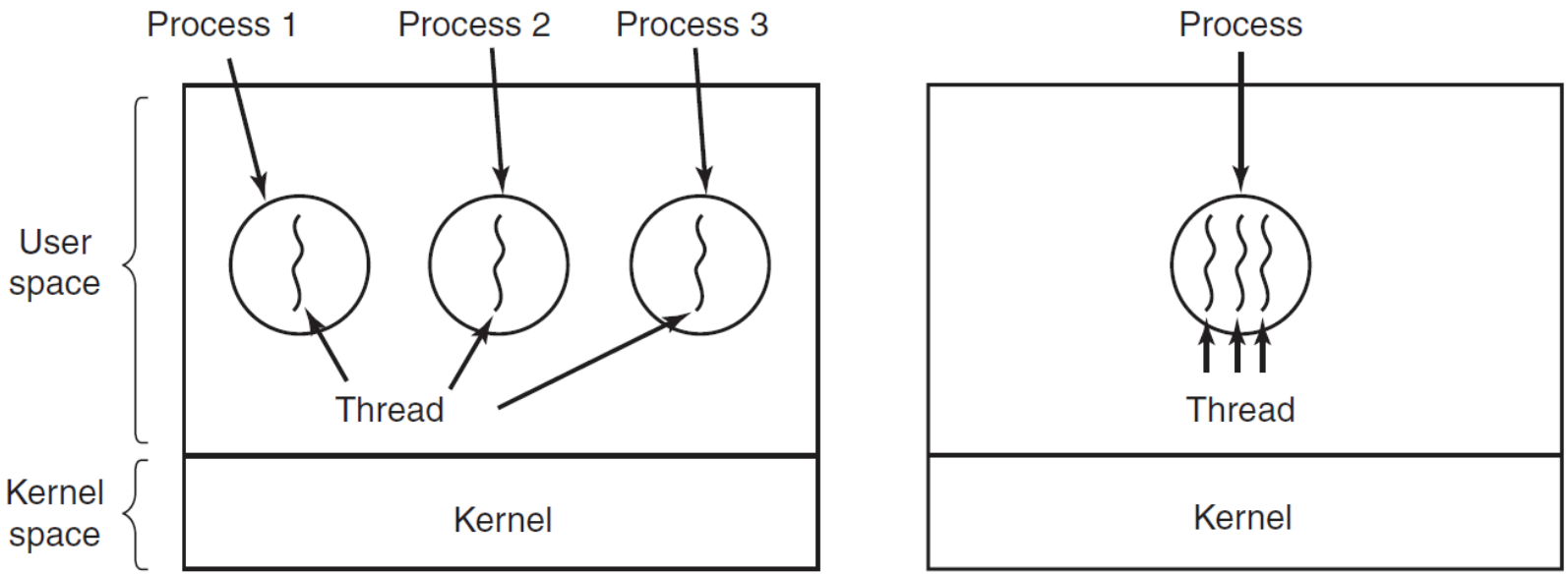
Thread Hardware



A thread switch should be much more efficient than a process switch, which typically requires hundreds to thousands of processor cycles.

Threads Overview

Having multiple threads running in parallel in one process is analogous to having multiple processes running in parallel in one computer.



Most applications running on modern computers are multithreaded.

An application is implemented as a separate process with several threads.

Example: Web browser has a thread displaying images or text. Another thread retrieves data from network. ■

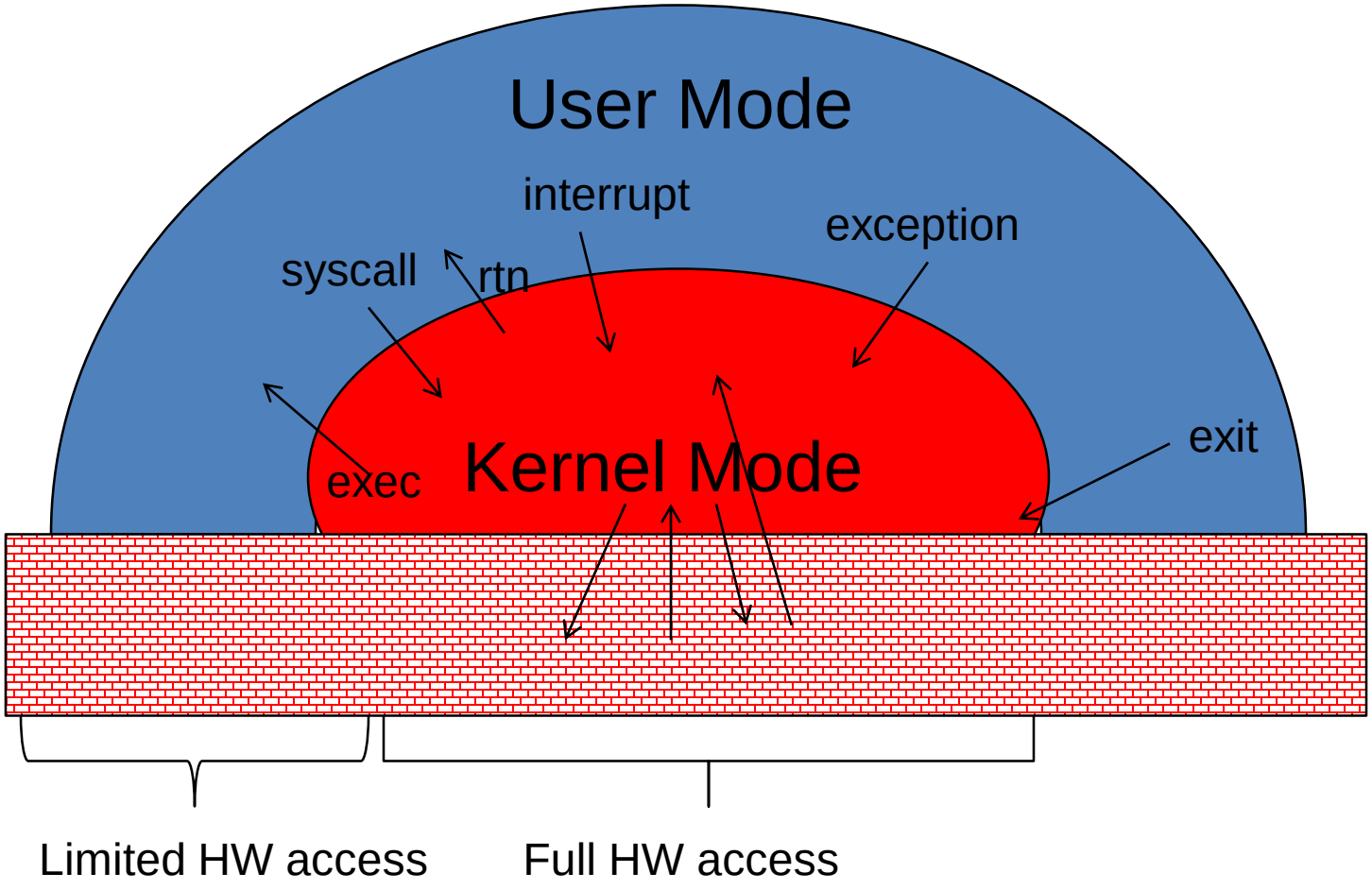
Example:Word processor has graphics display thread, another responding keystrokes, and thread performing background spelling and grammar checking. ■

CPU-intensive applications perform tasks in parallel to leverage processing capabilities on multicore systems.

Four Fundamental OS Concepts

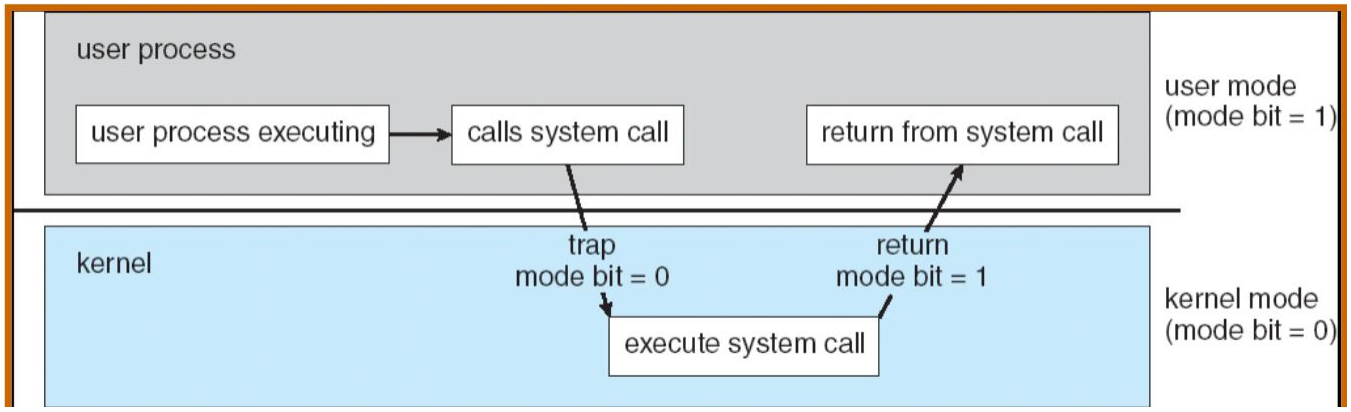
- **Thread: Execution Context**
  - Program Counter, Registers, Execution Flags, Stack
- **Address space** (with **translation**)
  - Program's view of memory is distinct from physical machine
- **Process: an instance of a running program**
  - Address Space + One or more Threads
- **Dual mode operation / Protection**
  - Only the "system" can access certain resources
  - Combined with translation, isolates programs from each other

User/Kernel (Privileged) Mode



Basic tool: Dual Mode Operation

- Hardware provides at least two modes:
  1. Kernel Mode (or "supervisor" / "protected" mode)
  2. User Mode
- Certain operations are **prohibited** when running in user mode
  - Changing the page table pointer
- Carefully controlled transitions between user mode and kernel mode
  - System calls, interrupts, exceptions



System Calls

Programming interface to the services provided by the OS

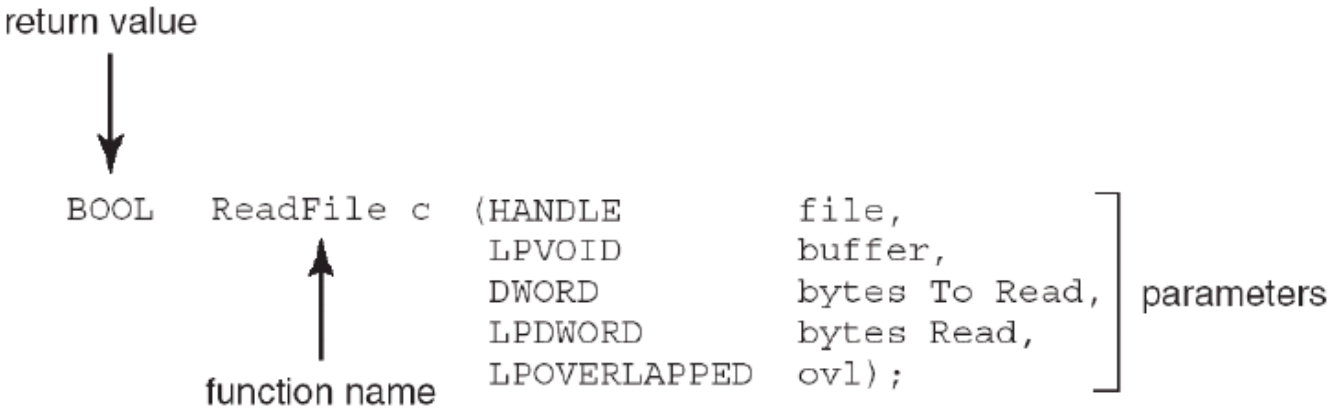
- Typically written in a high-level language (C or C++)

- Mostly accessed by programs via a high-level Application Program Interface (API) rather than direct system call use

- Three most common APIs are Win32 API for Windows, POSIX API for POSIX-based systems (including virtually all versions of UNIX, Linux, and Mac OS X), and Java API for the Java virtual machine (JVM)

Example of Standard API

- Consider the ReadFile() function in the
- Win32 API—a function for reading from a file



- A description of the parameters passed to ReadFile()
  - HANDLE file—the file to be read
  - LPVOID buffer—a buffer where the data will be read into and written from
  - DWORD bytesToRead—the number of bytes to be read into the buffer
  - LPDWORD bytesRead—the number of bytes read during the last read
  - LPOVERLAPPED ovl—indicates if overlapped I/O is being used

Example System calls

Process management

Call	Description
pid = fork( )	Create a child process identical to the parent
pid = waitpid(pid, &statloc, options)	Wait for a child to terminate
s = execve(name, argv, environp)	Replace a process' core image
exit(status)	Terminate process execution and return status

File management

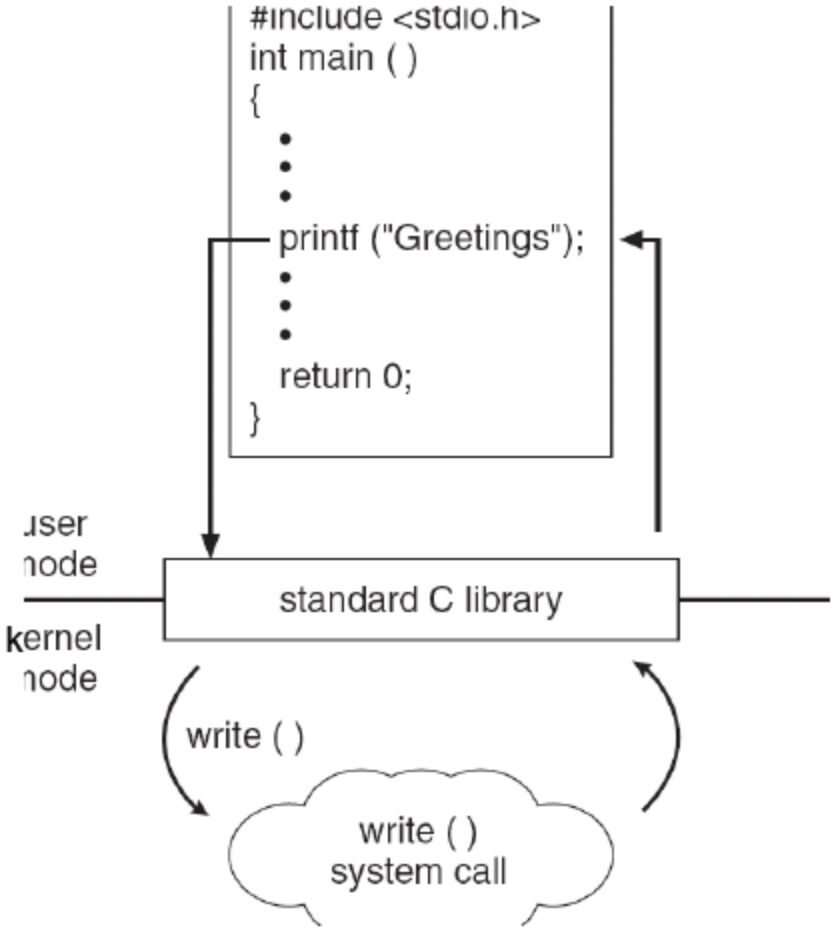
Call	Description
fd = open(file, how, ...)	Open a file for reading, writing or both
s = close(fd)	Close an open file
n = read(fd, buffer, nbytes)	Read data from a file into a buffer
n = write(fd, buffer, nbytes)	Write data from a buffer into a file
position = lseek(fd, offset, whence)	Move the file pointer
s = stat(name, &buf)	Get a file's status information

System Calls

UNIX	Win32	Description
fork	CreateProcess	Create a new process
waitpid	WaitForSingleObject	Can wait for a process to exit
execve	(none)	CreateProcess = fork + execve
exit	ExitProcess	Terminate execution
open	CreateFile	Create a file or open an existing file
close	CloseHandle	Close a file
read	ReadFile	Read data from a file
write	WriteFile	Write data to a file
lseek	SetFilePointer	Move the file pointer
stat	GetFileAttributesEx	Get various file attributes
mkdir	CreateDirectory	Create a new directory
rmdir	RemoveDirectory	Remove an empty directory
link	(none)	Win32 does not support links
unlink	DeleteFile	Destroy an existing file
mount	(none)	Win32 does not support mount
umount	(none)	Win32 does not support mount
chdir	SetCurrentDirectory	Change the current working directory
chmod	(none)	Win32 does not support security (although NT does)
kill	(none)	Win32 does not support signals
time	GetLocalTime	Get the current time

Standard C Library Example

C program invoking printf() library call, which calls write() system call



Protection

- CPU supports a set of assembly instructions

– MOV [address], ax
– ADD ax, bx
– MOV CRn (move control register)
– IN, INS (input string)
– HLT (halt)
– LTR (load task register)
– INT n (software interrupt)

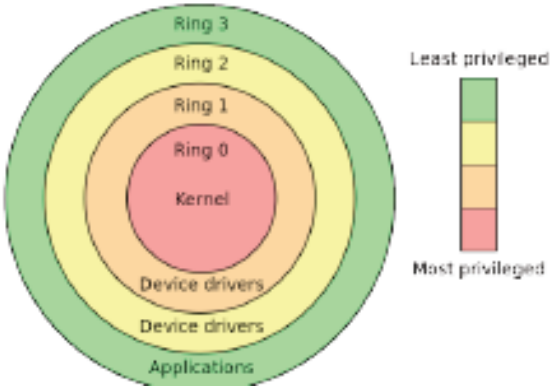
– Some instructions are sensitive or privileged

Protection

Kernel mode vs. User mode: To protect the system from aberrant users and processors, some instructions are restricted to use only by the OS. Users may not
– address I/O directly
– use instructions that manipulate the state of memory (page table pointers, TLB load, etc.)

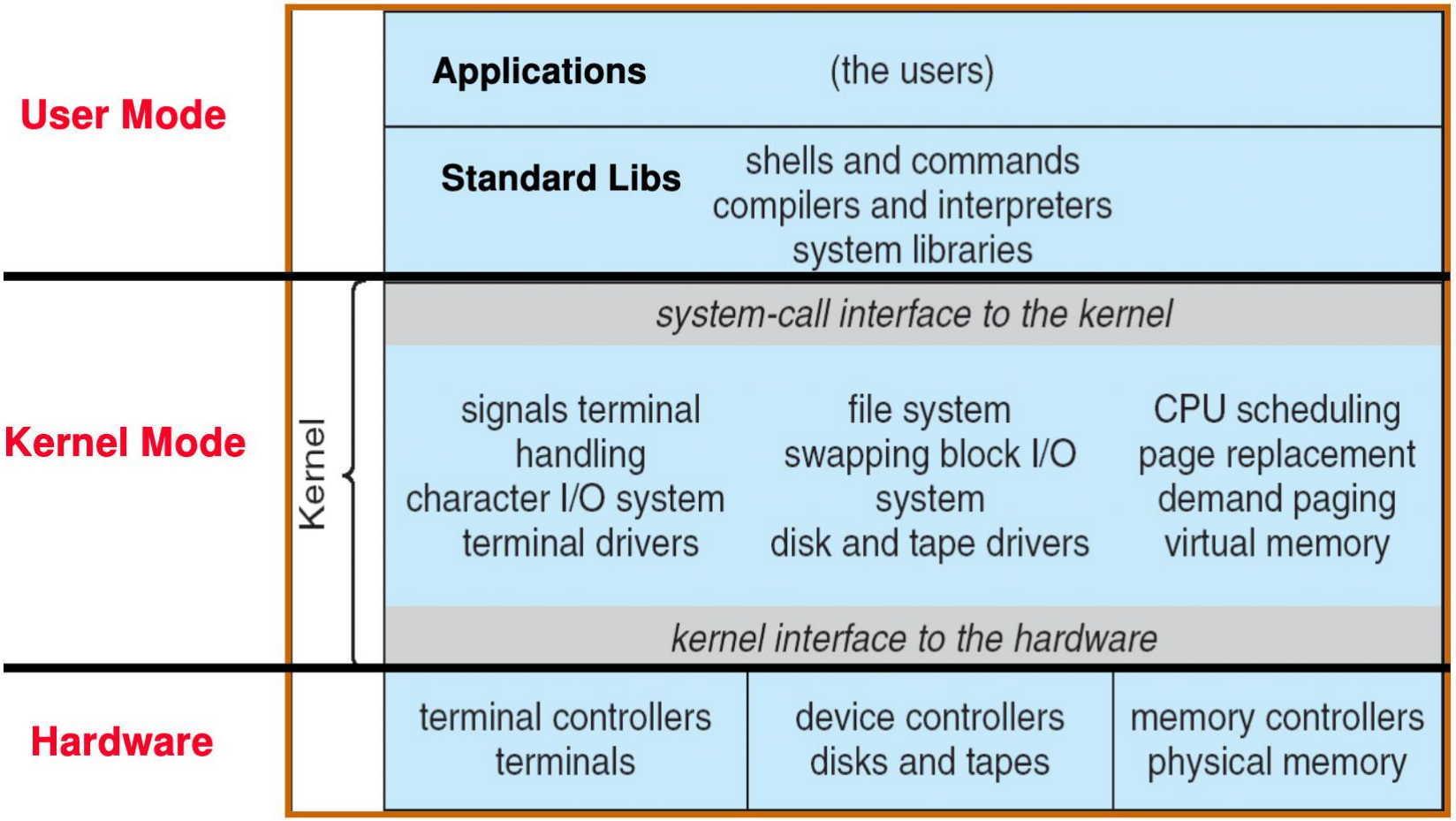
– set the mode bits that determine user or kernel mode
– disable and enable interrupts
– halt the machine

but in kernel mode, the OS can do all these things.



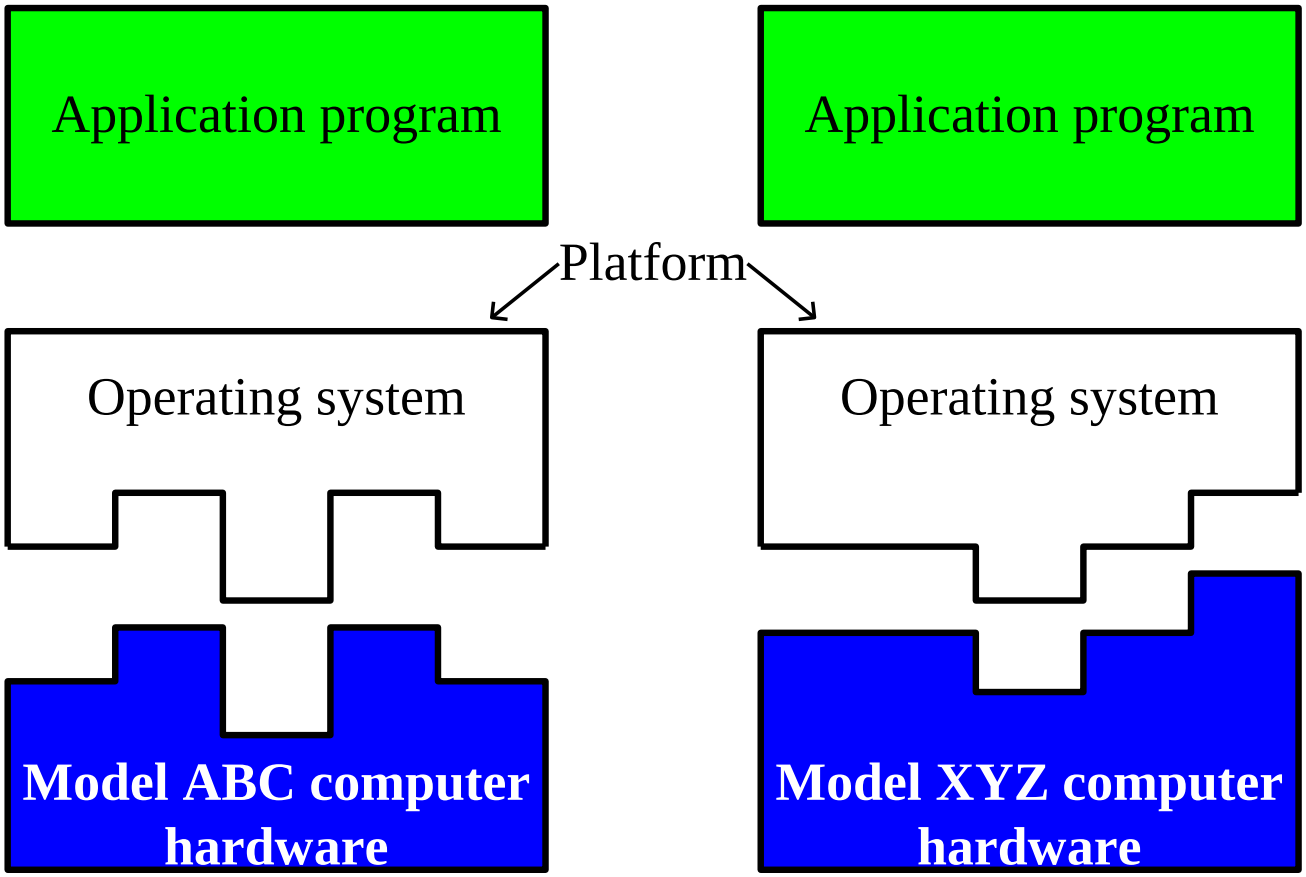
The hardware must support at least kernel and user mode.
– A status bit in a protected processor register indicates the mode.
– Protected instructions can only be executed in kernel mode

UNIX OS Structure



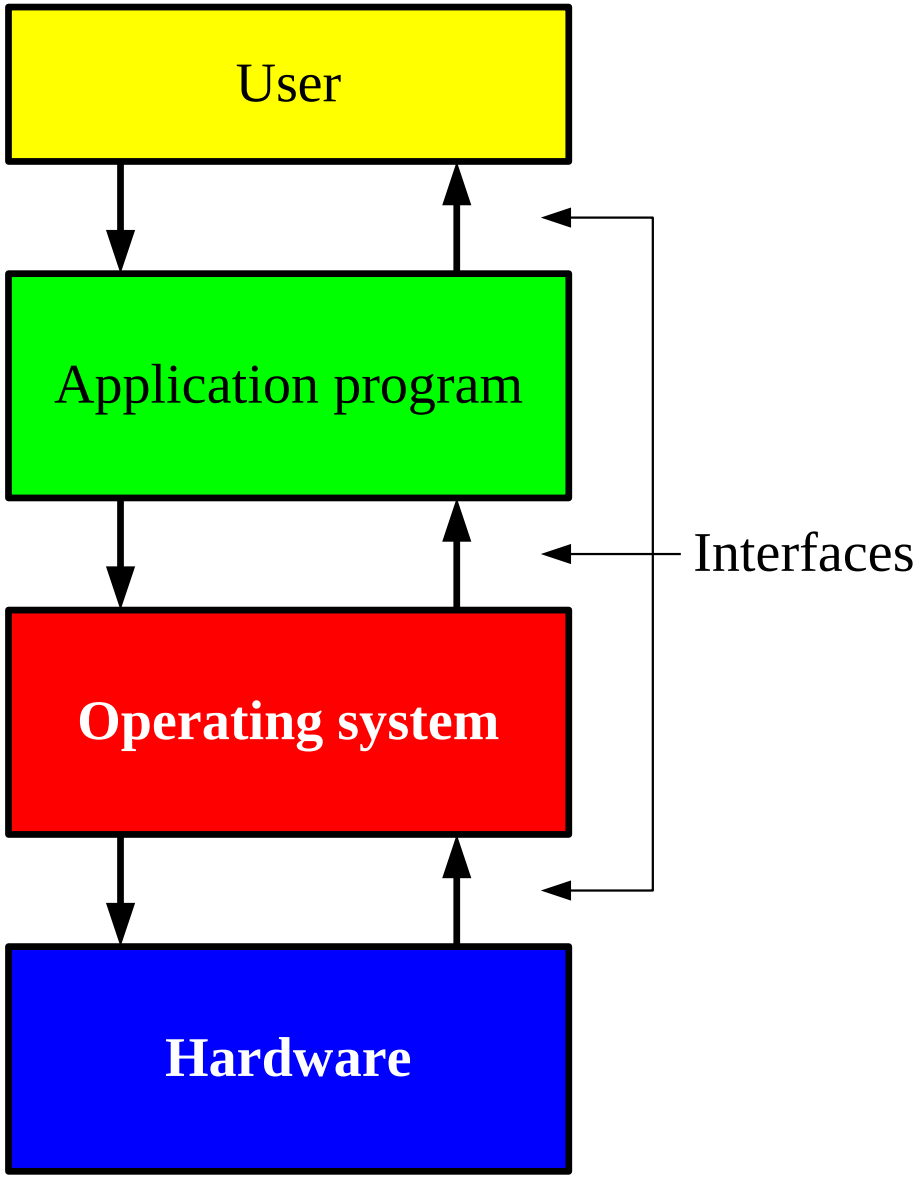
Architectural Features Motivated by OS Services

OS Service	Hardware Support
Protection	Kernel/user mode, protected instructions, base/limit registers
Interrupts	Interrupt vectors
System calls	Trap instructions and trap vectors
I/O	Interrupts and memory mapping
Scheduling, error recovery, accounting	Timer
Synchronization	Atomic instructions
Virtual memory	Translation look-aside buffers



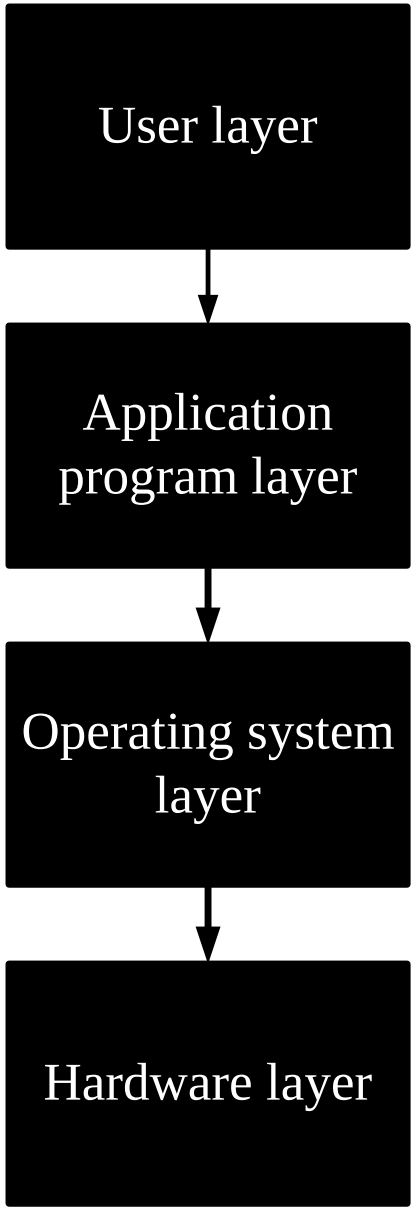
The operating system routines that interface with the application program represent a consistent platform for running application programs.

The operating system is a set of system software routines that sits between the application program and the hardware.



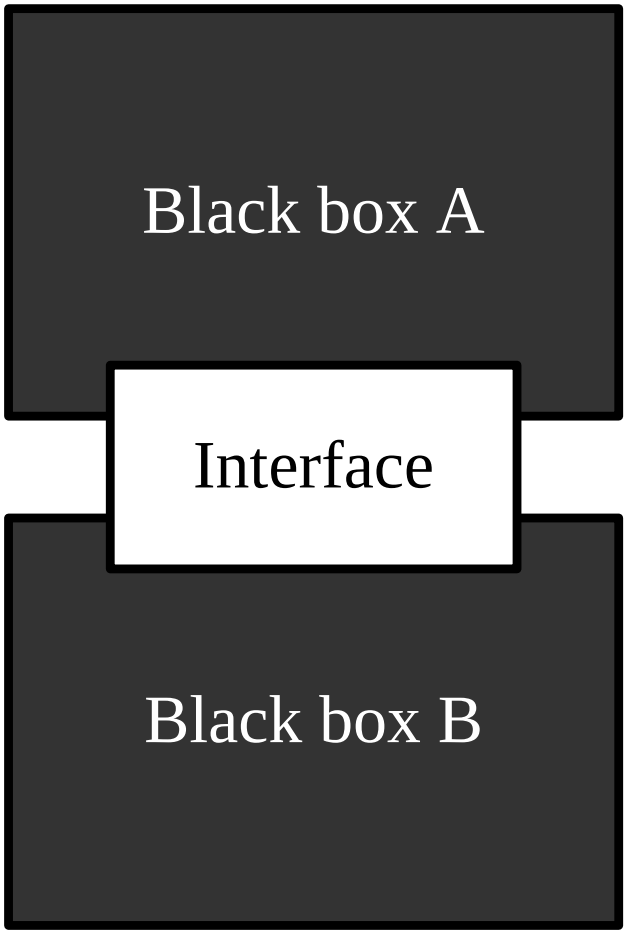
View each layer
of abstraction as a black box.

- Abstraction
  - A simplified view of an object that ignores the internal details.
- Layers of abstraction
- Black box
  - Inputs/outputs known
  - Contents hidden
  - Functionally independent

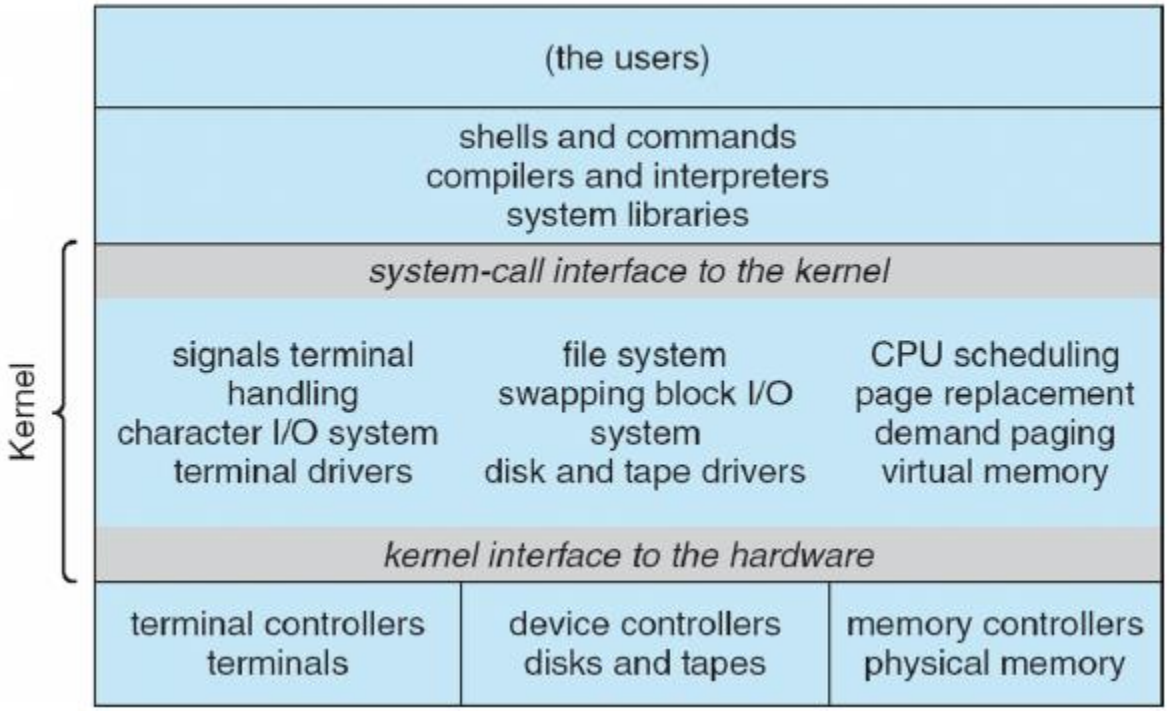


Two black boxes communicate through a shared interface.

- To use a black box, all you must know are its interface rules.



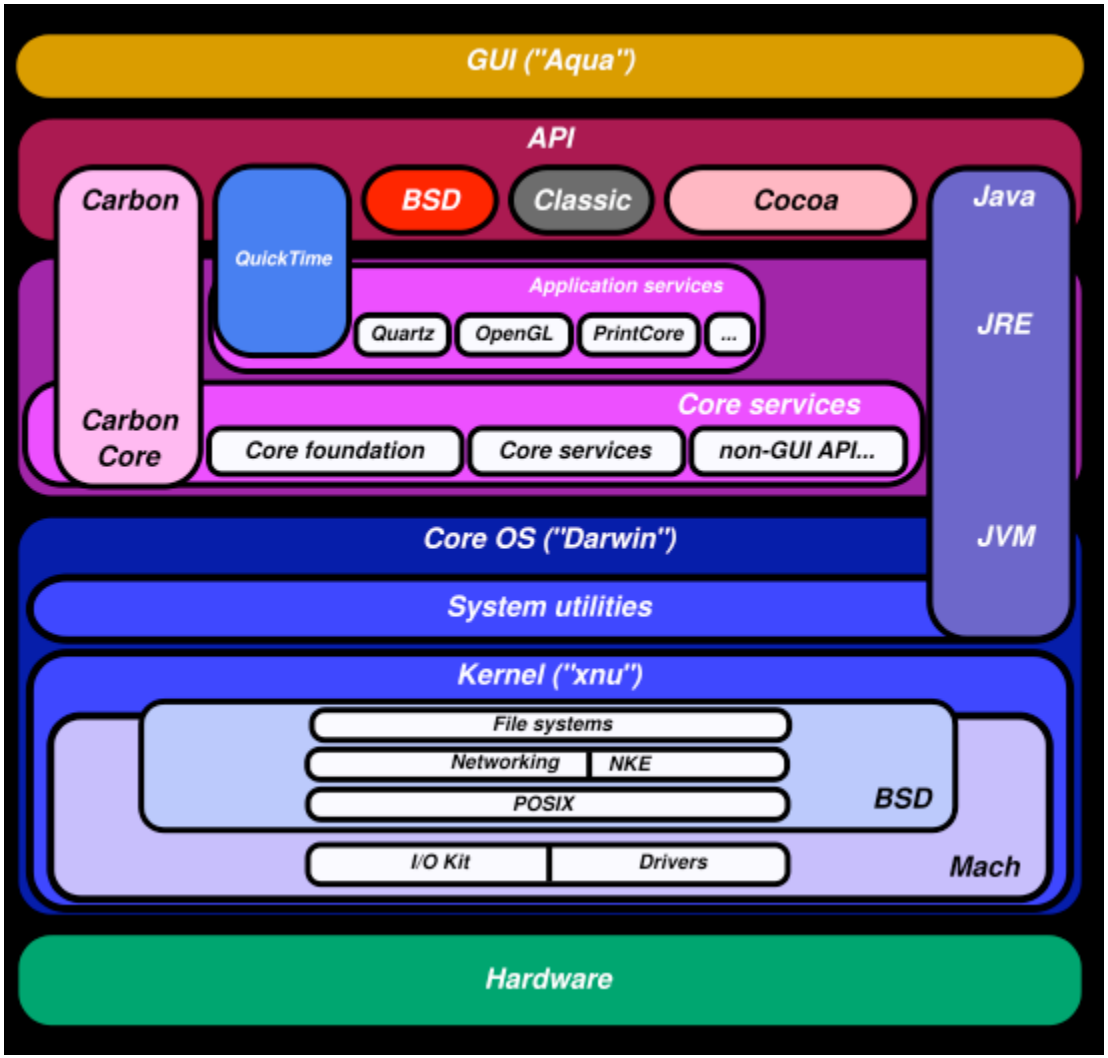
One Basic OS Structure



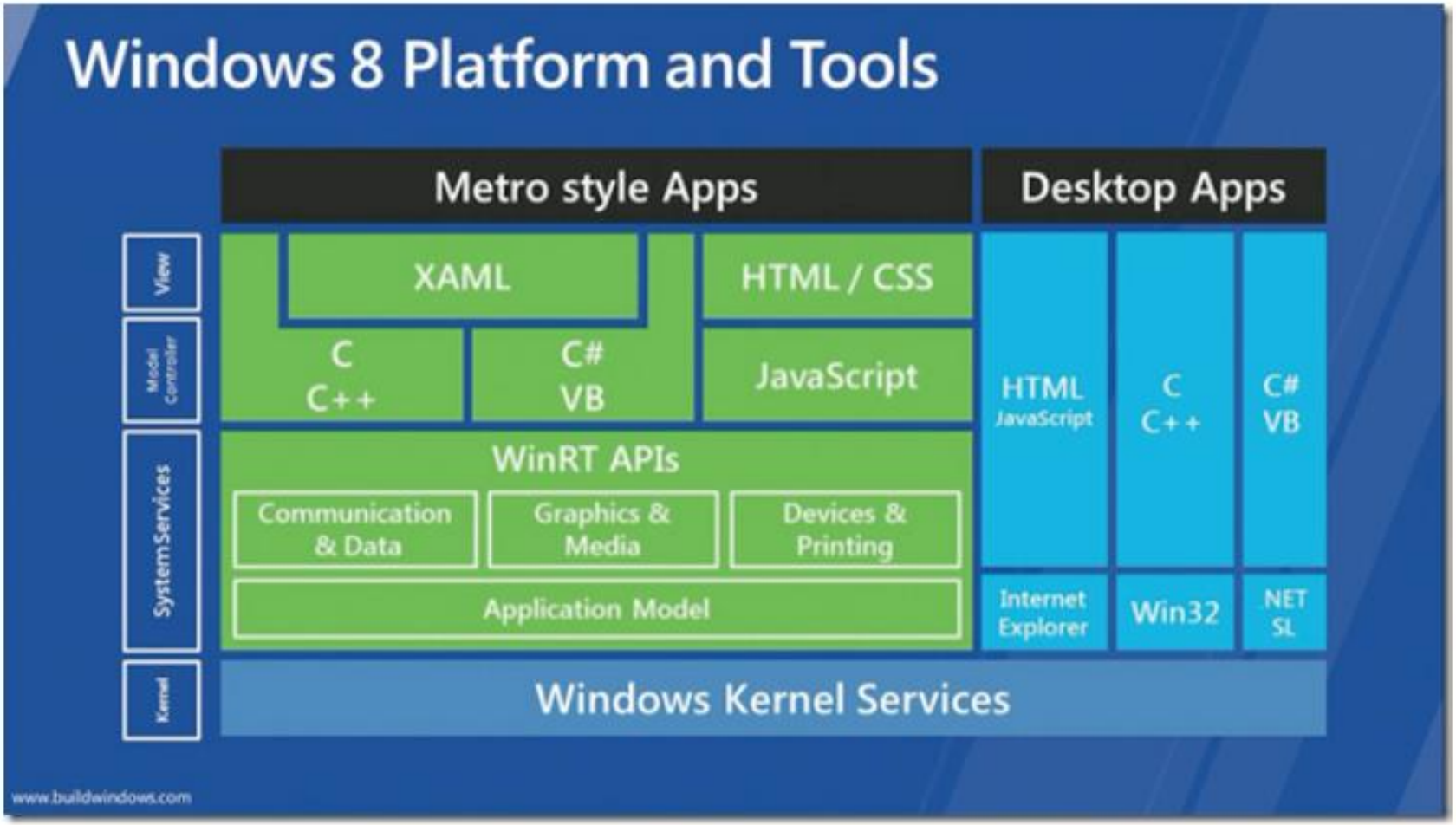
The *kernel* is the protected part of the OS that runs in kernel mode, protecting the critical OS data structures, and device registers from user programs.

• Debate about what functionality goes into the kernel (above figure: UNIX) - "monolithic kernels"

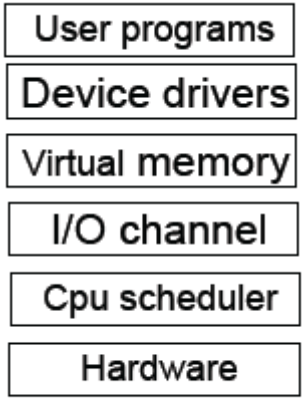
Mac OS X Architecture



Windows 8 Architecture

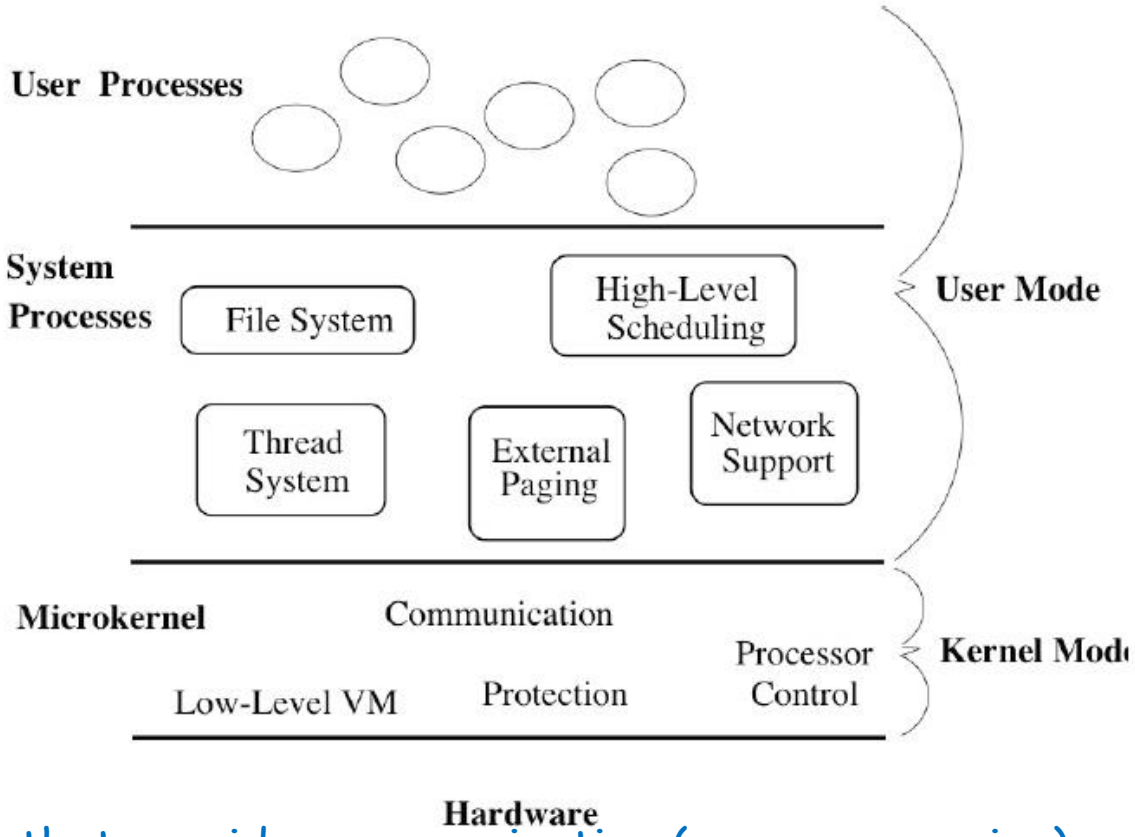


Layered OS design



- *Layer N: uses layer N-1* and provides new functionality to N+1
- •Advantages: modularity, simplicity, portability, ease of design/debugging
- Disadvantage - communication overhead between layers,
- extra copying, book-keeping, layer design

Microkernel



- Small kernel that provides communication (message passing) and other basic functionality
- other OS functionality implemented as user-space processes

Microkernel Features

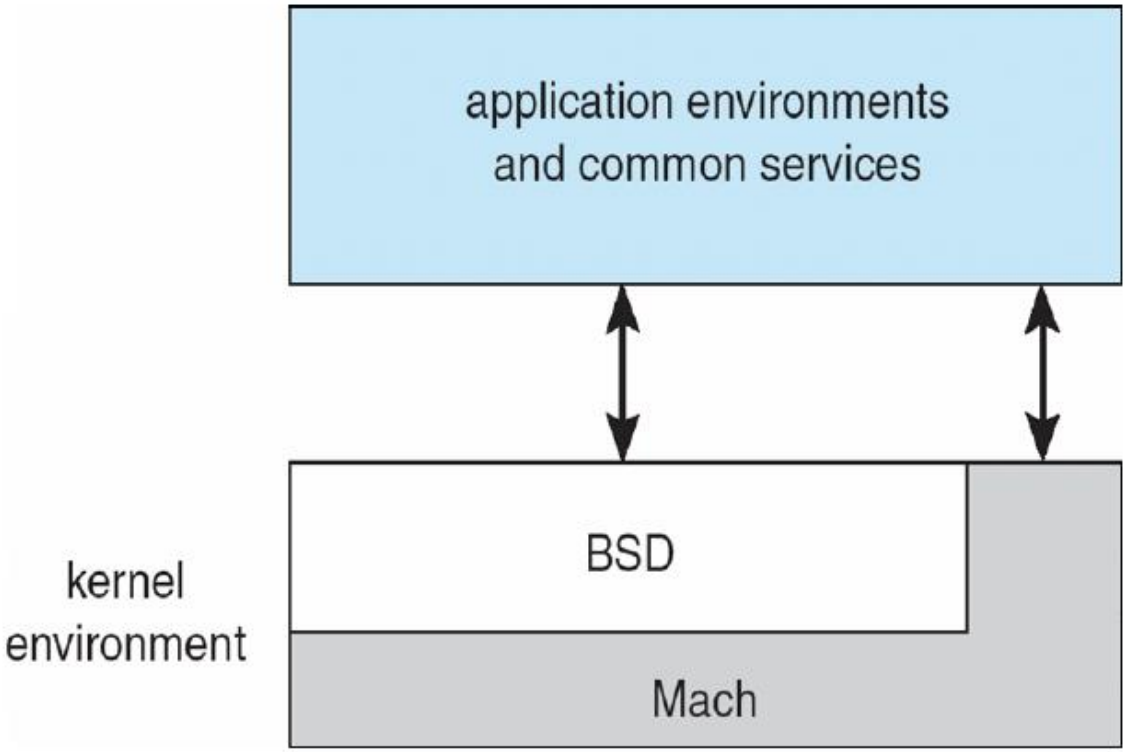
Goal: to minimize what goes in the kernel mechanism, no policy), implementing as much of the OS in User-Level processes as possible.

- Advantages

– better reliability, easier extension and customization
– mediocre performance (unfortunately)

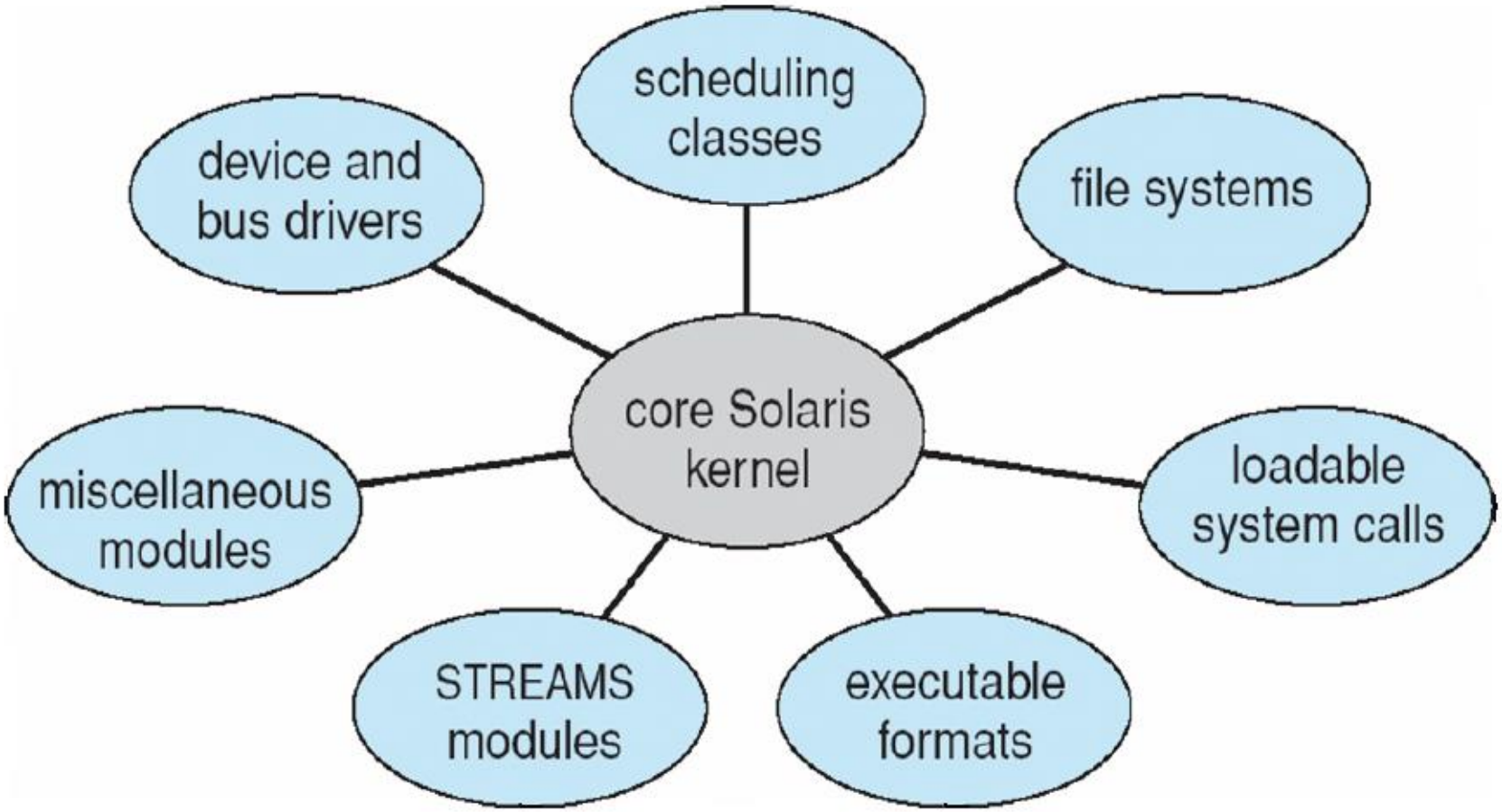
- First Microkernel was Hydra (CMU '70). Current systems include Chorus (France) and Mach (CMU).

Mac OS X - hybrid approach



Layered system: Mach microkernel (mem, RPC, IPC) + BSD (threads, CLI, networking, filesystem) + user-level services (GUI)

Solaris Modular Approach



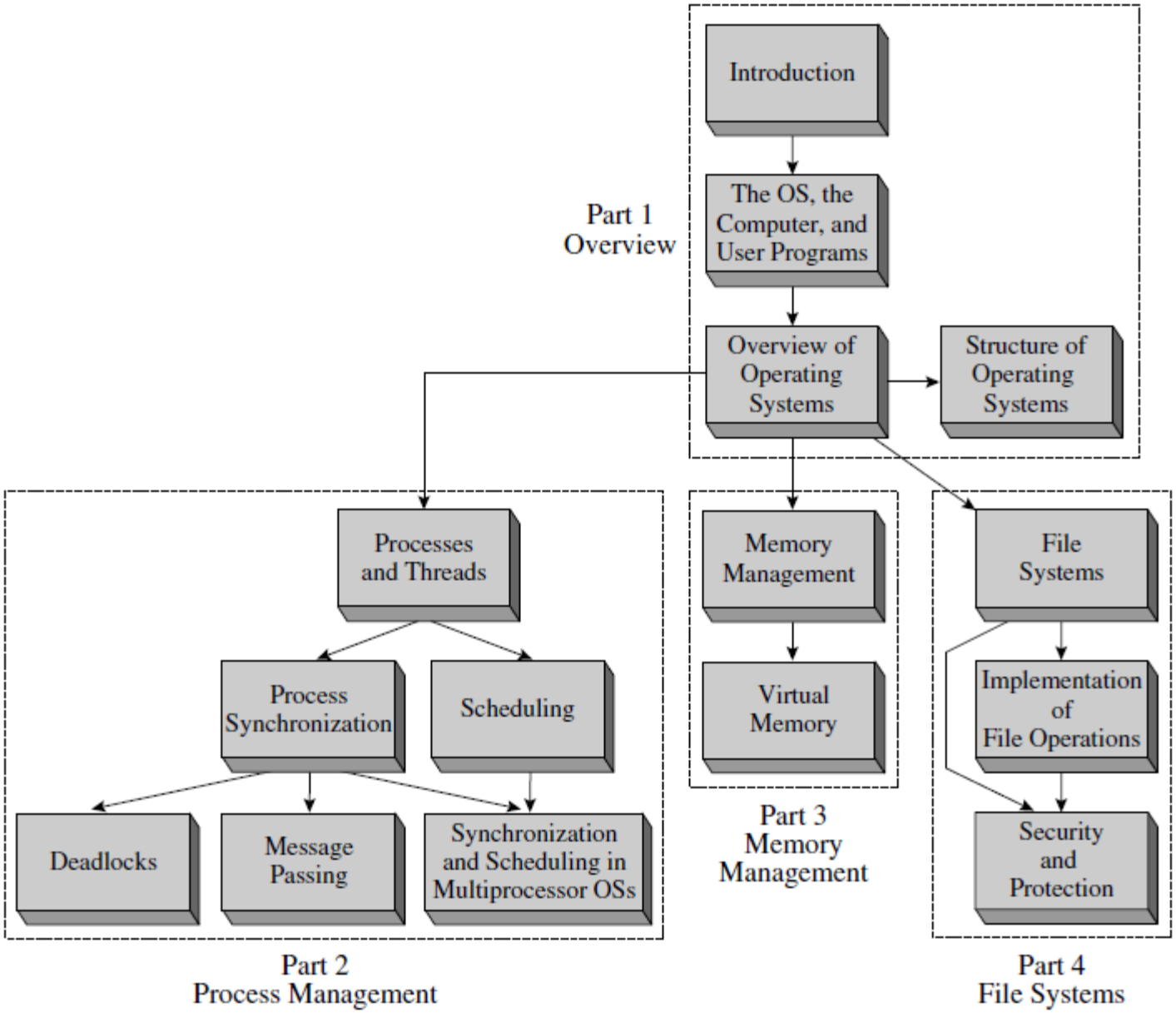
OS Structures- Summary

Big Design Issue: How do we make the OS efficient, reliable, and extensible?

• **General OS Philosophy**: The design and implementation of an OS involves a constant tradeoff between *simplicity* and *performance*. As a general rule, strive for simplicity except when you have a strong reason to believe that you need to make a particular component complicated to achieve acceptable performance (strong reason = simulation or evaluation study)

Process Management

Overview


Introduction
The OS, the Computer, and User Programs
Part 1 Overview
Overview of Operating Systems
Structure of Operating Systems
Processes and Threads
Process Synchronization
Scheduling
Deadlocks
Message Passing
Synchronization and Scheduling in Multiprocessor OSs
Part 2 Process Management
Memory Management
Virtual Memory
Part 3 Memory Management
File Systems
Implementation of File Operations
Security and Protection
Part 4 File Systems