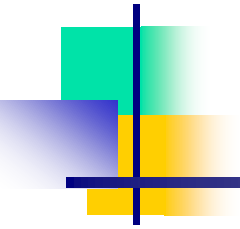
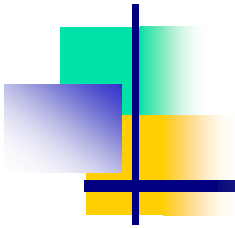


Operating Systems

Lecture1 - Introduction

Golestan University

Hossein Momeni
momeni@iust.ac.ir



Outline

- What is an operating system?
- Operating systems history
- Operating system concepts
- System calls
- Operating system structure
  - User interface to the operating system
  - Anatomy of a system call

Samples of Operating Systems

- IBSYS (IBM 7090)
- OS/360 (IBM 360)
- TSS/360 (360 mod 67)
- Michigan Terminal System
- CP/CMS & VM 370
- MULTICS (GE 645)
- Alto (Xerox PARC)
- Pilot (Xerox STAR)
- CP/M
- IRIX
- Solaris
- MVS
- VxWorks
- MACH
- Apollo DOMAIN
- Unix (System V & BSD)
- Apple Mac (v. 1– v. 9)
- MS-DOS
- Windows NT, 2000, XP, 7
- Novell Netware
- Linux
- FreeBSD
- PalmOS
- PocketPC
- VxWorks

Samples of Operating Systems (continue…)

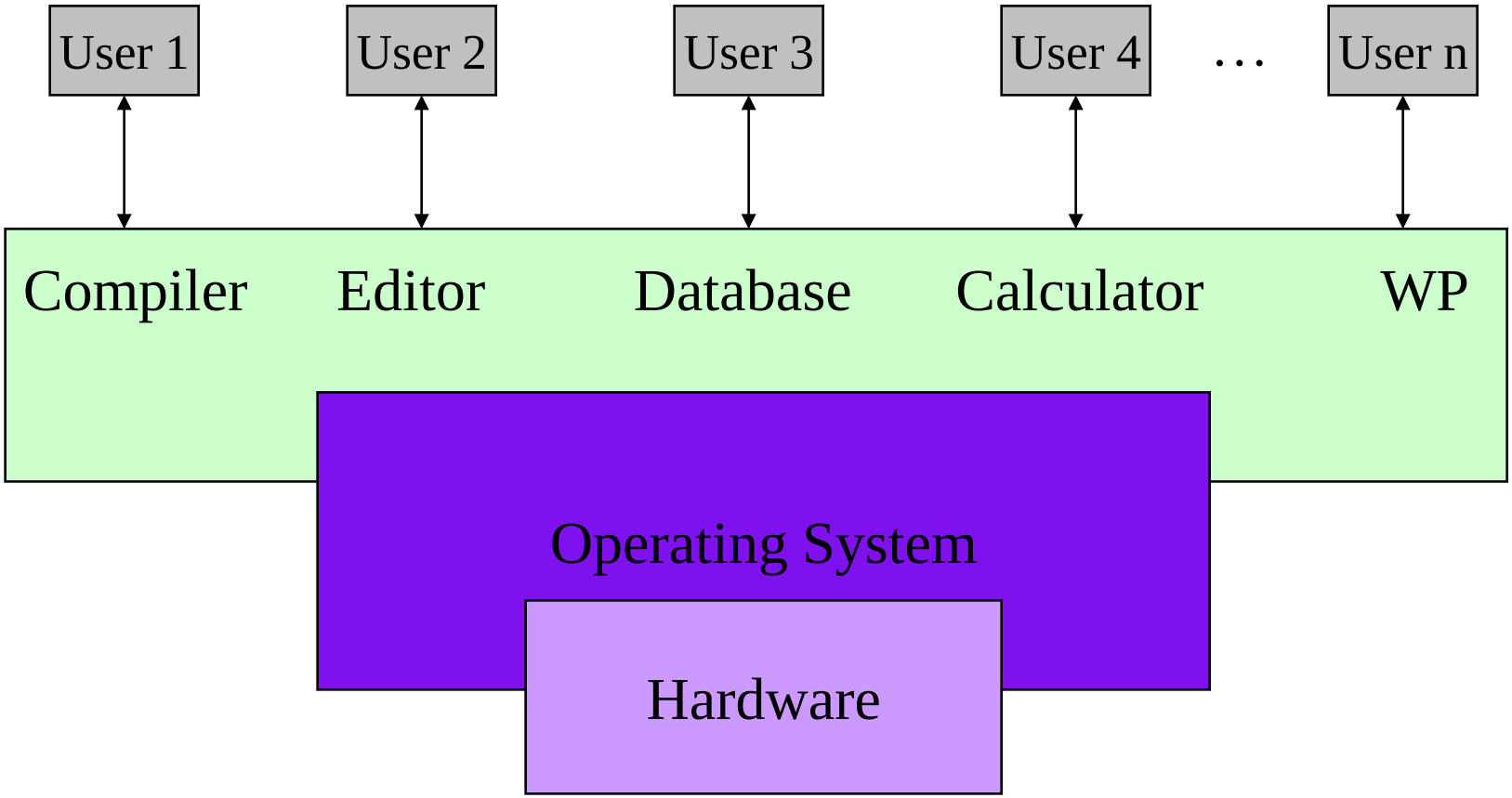
What is an Operating System?

- An Operating System is a program that acts as an intermediary/interface between a user of a computer and the computer hardware.

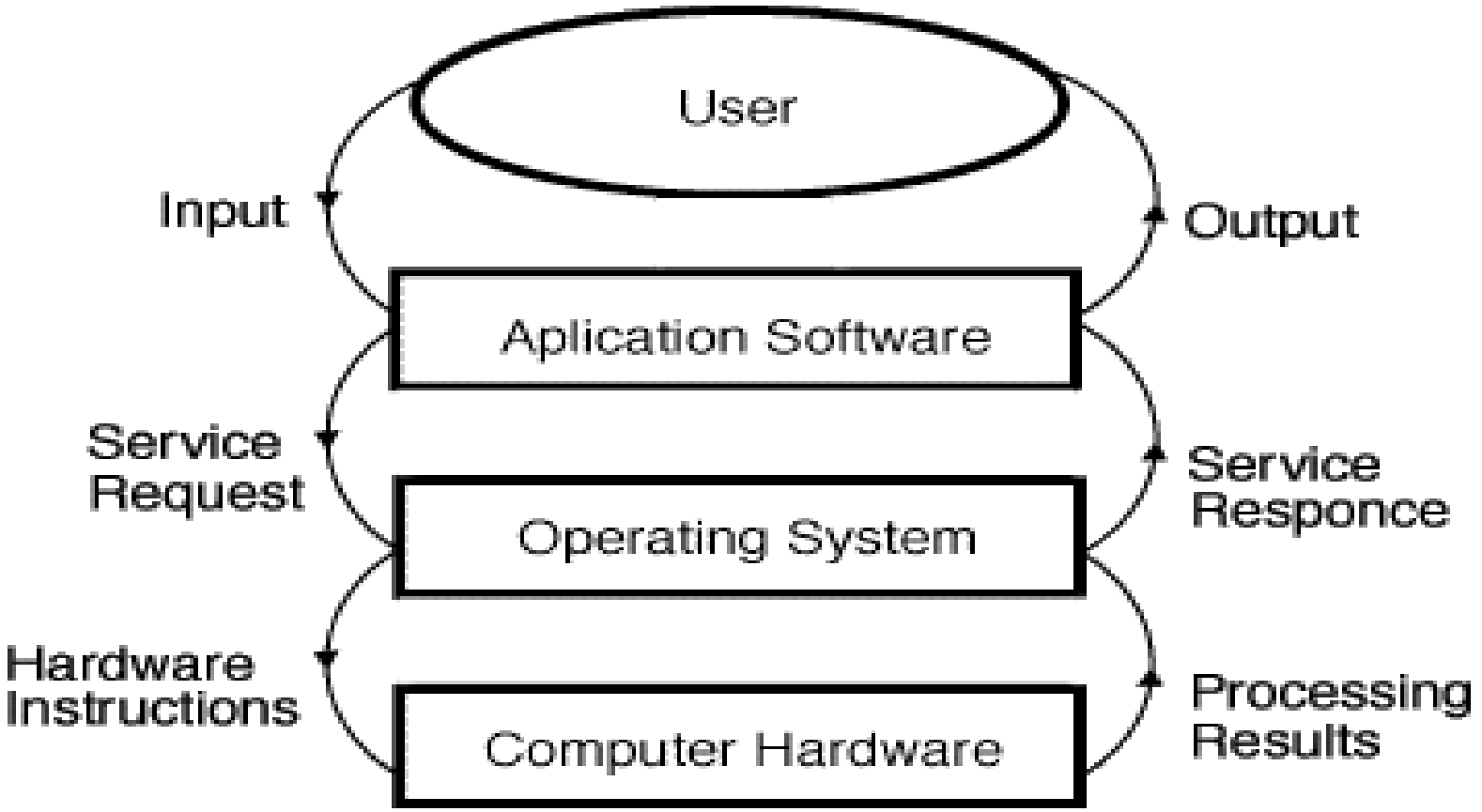
- It is an extended machine
  - Hides the messy details which must be performed
  - Presents user with a virtual machine, easier to use

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

Static View of System Components



Dynamic View of System Components



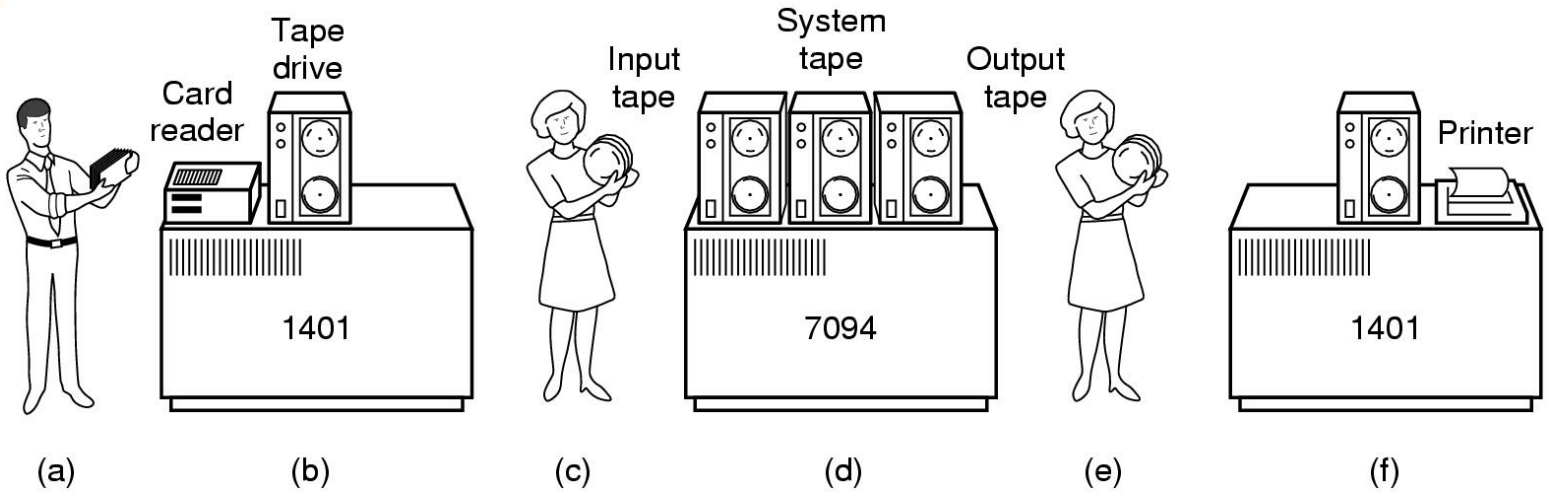
History of Operating Systems

- First generation: 1945 – 1955
  - Vacuum tubes, Plug boards
- Second generation: 1955 – 1965
  - Transistors, Batch systems
- Third generation: 1965 – 1980
  - Integrated circuits, Multiprogramming
- Fourth generation: 1980 – present
  - Large scale integration, Personal computers
- Next generation: ???
  - Systems connected by high-speed networks?
  - Wide area resource management?

First generation: direct input

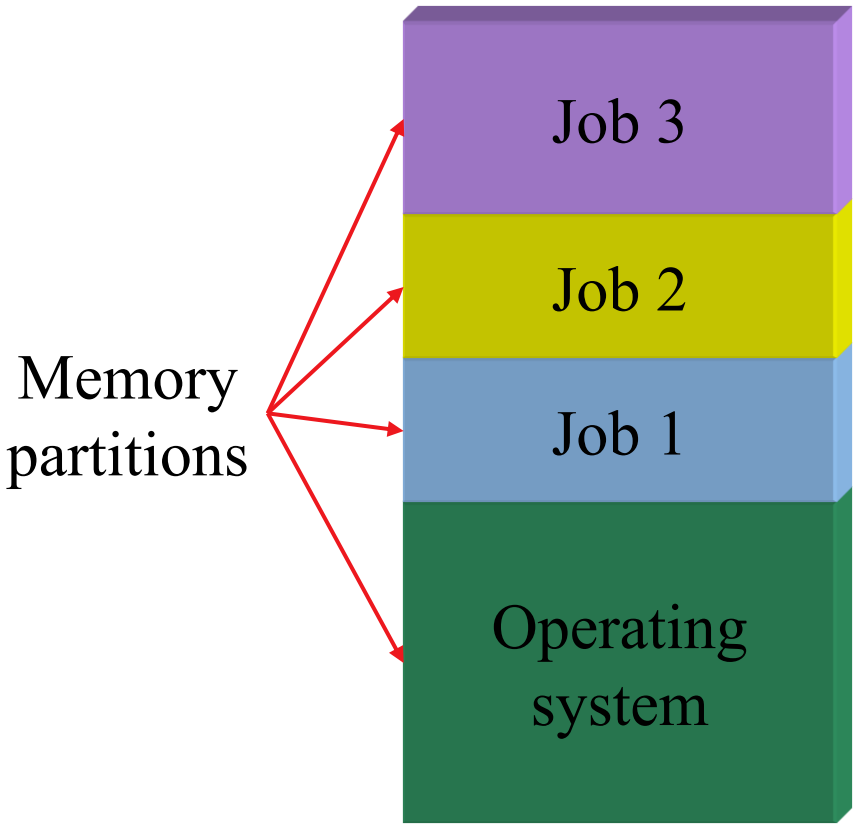
- Run one job at a time
  - Enter it into the computer (might require rewiring!)
  - Run it
  - Record the results
- No Operating System
- Problem: lots of wasted computer time!
  - Computer was idle during first and last steps
  - Computers were ***very*** expensive!
- Goal: make better use of an expensive commodity: computer time

Second generation: batch systems

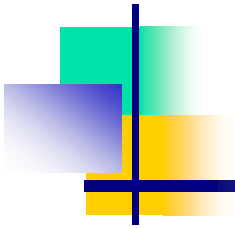


- Bring cards to 1401
- Read cards onto input tape
- Put input tape on 7094
- Perform the computation, writing results to output tape
- Put output tape on 1401, which prints output
- Offline Spooling
- High Turnaround time, No interaction, No overlapping I/O and CPU

Third generation: multiprogramming



- Multiple jobs in memory
  - Protected from one another
- Operating system protected from each job as well
- Resources (time, hardware) split between jobs
- Online Spooling
- Still not interactive
  - User submits job
  - Computer runs it
  - User gets results minutes (hours, days) later



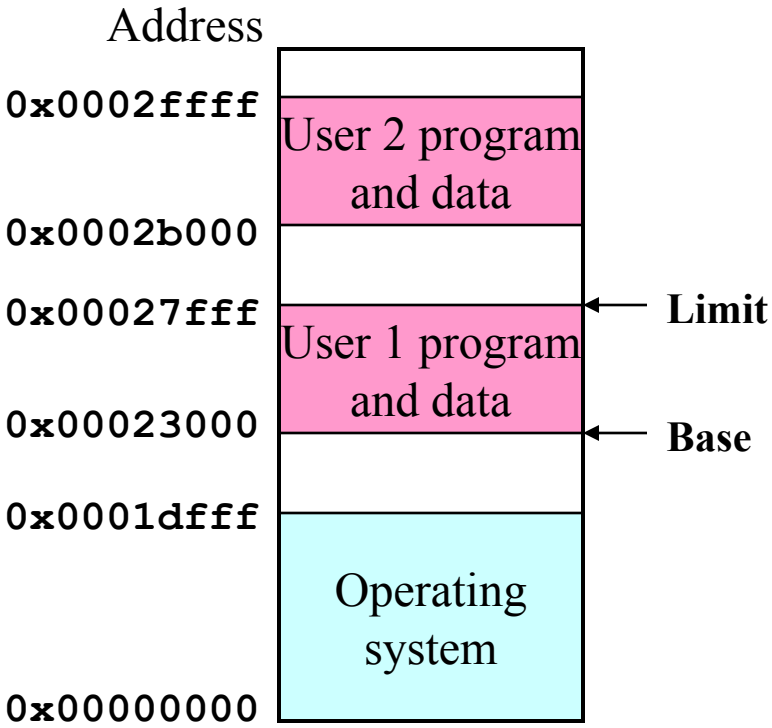
Timesharing

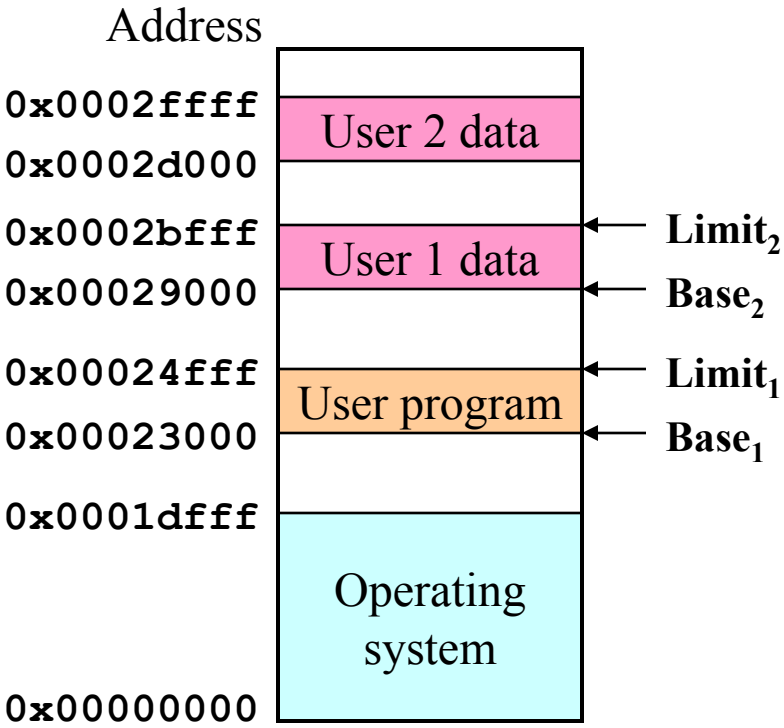
- Multi tasking
- Share CPU time between n Jobs
- Time Slice (Quantum)
- Timer Interrupt
- Multiprogramming allowed several jobs to be active at one time
  - Initially used for batch systems
  - Cheaper hardware terminals → interactive use
- Computer use got much cheaper and easier
  - Quick turnaround meant quick fixes for problems

Fourth generation: Personal Computer

- Large Scale IC
- Multi Programming
- Multi Users
- Low Hardware Cost
- Distributed Systems, Parallel Systems

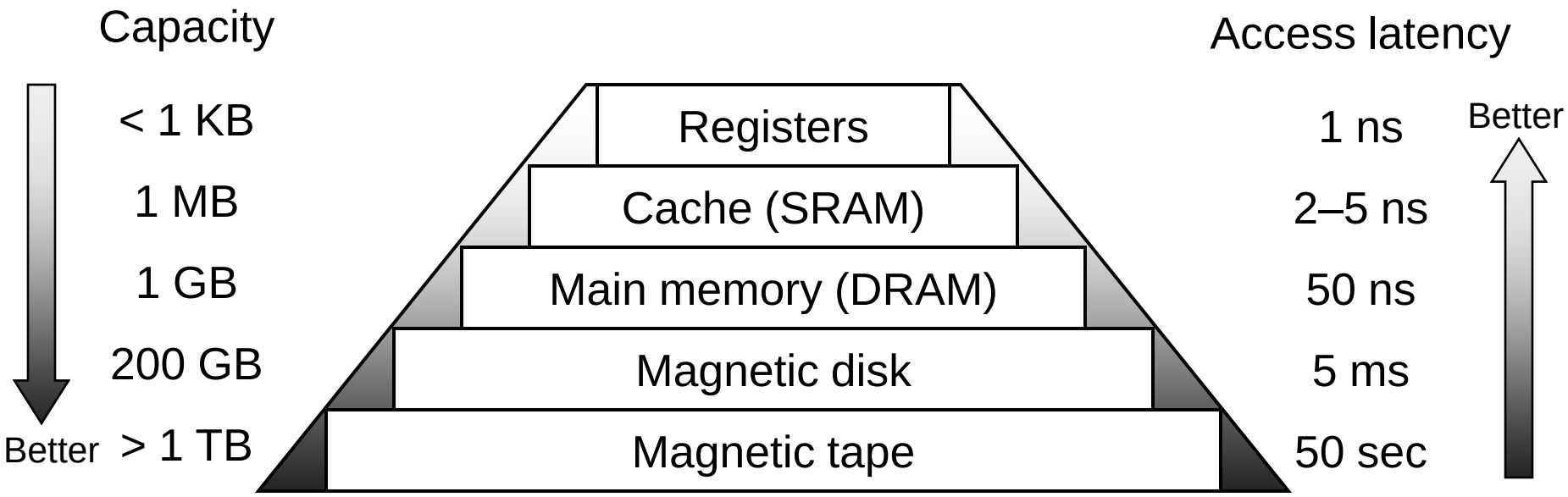
Memory





- Single base/limit pair: set for each process
- Two base/limit registers: one for program, one for data

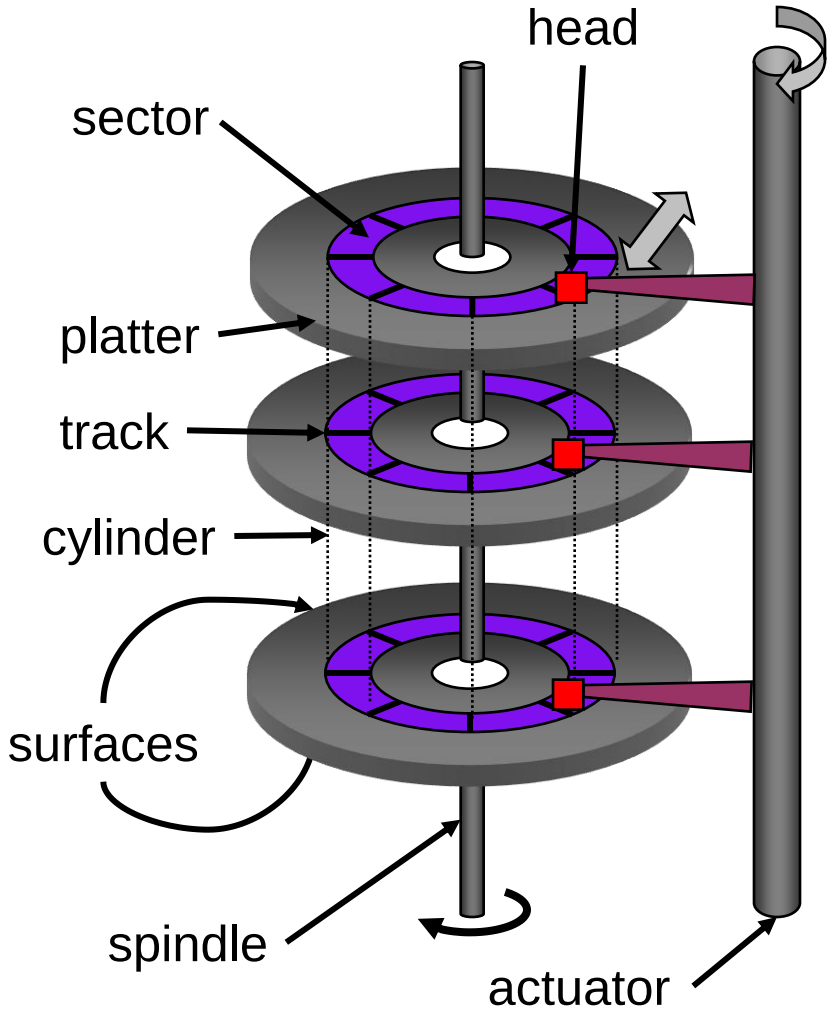
Storage pyramid



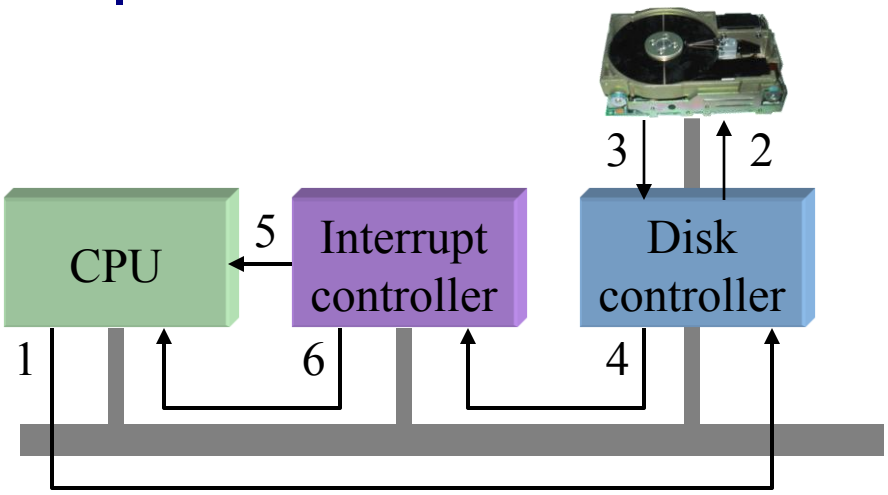
- Goal: really large memory with very low latency
  - Latencies are smaller at the top of the hierarchy
  - Capacities are larger at the bottom of the hierarchy
- Solution: move data between levels to create illusion of large memory with low latency

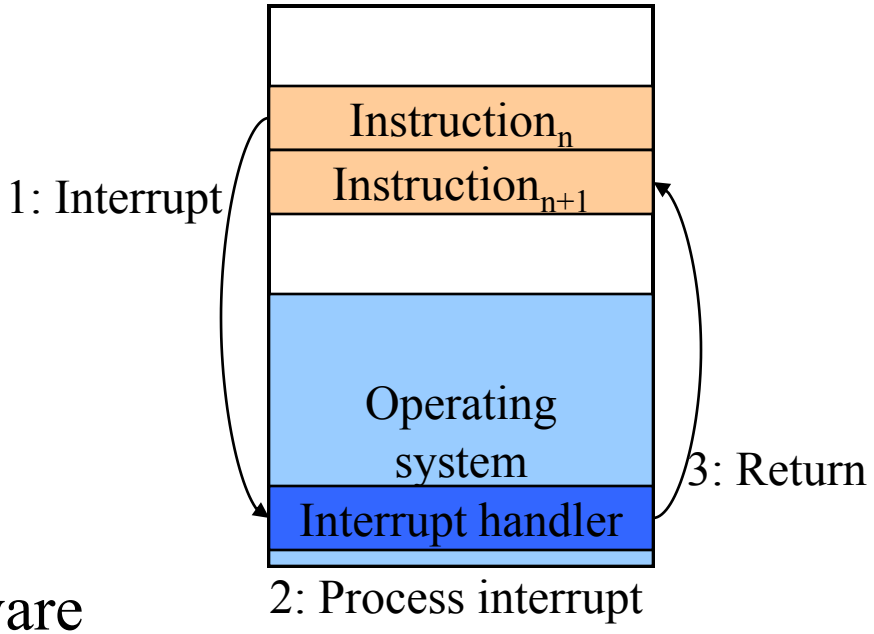
Disk drive structure

- Data stored on surfaces
  - Up to two surfaces per platter
  - One or more platters per disk
- Data in concentric tracks
  - Tracks broken into sectors
    - 256B-1KB per sector
  - Cylinder: corresponding tracks on all surfaces
- Data read and written by heads
  - Actuator moves heads
  - Heads move in unison



Anatomy of a device request



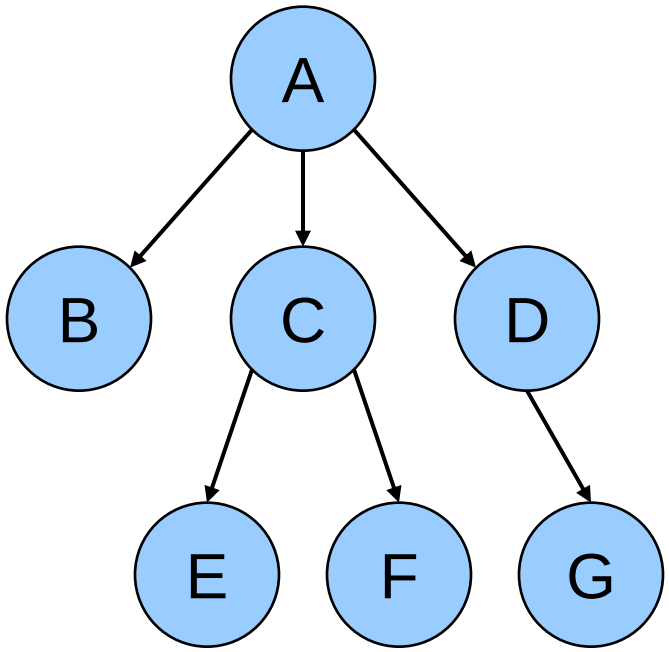


- Left: sequence as seen by hardware
  - Request sent to controller, then to disk
  - Disk responds, signals disk controller which tells interrupt controller
  - Interrupt controller notifies CPU
- Right: interrupt handling (software point of view)

Operating system Components

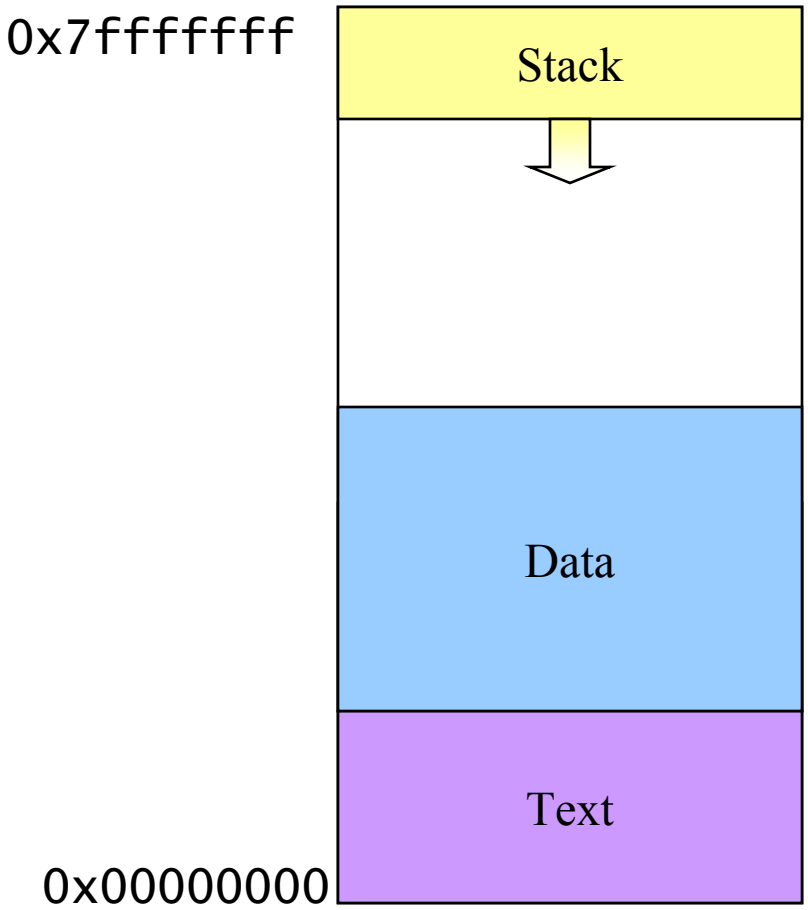
- Process Management
- Memory Management
- File Management
- I/O Management
- Process Scheduler
- Inter Process Communication
- Network Unit
- Command Interpreter
- Security Management

Processes



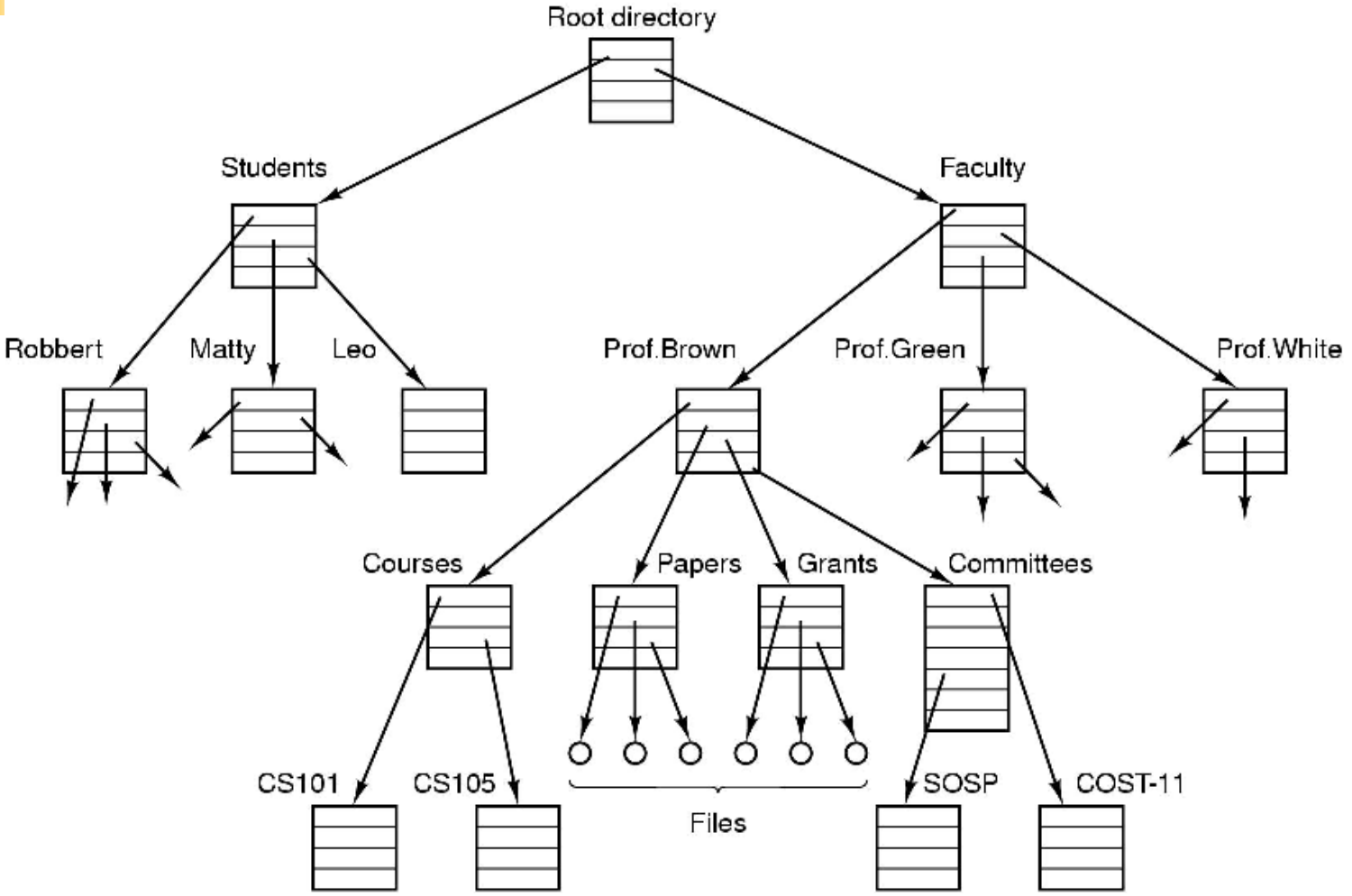
- Process: program in execution
  - Address space (memory) the program can use
  - State (registers, including program counter & stack pointer)
- OS keeps track of all processes in a *process table*
- Processes can create other processes
  - Process tree tracks these relationships
  - A is the *root* of the tree
  - A created three child processes: B, C, and D
  - C created two child processes: E and F
  - D created one child process: G

Memory image of a Unix process



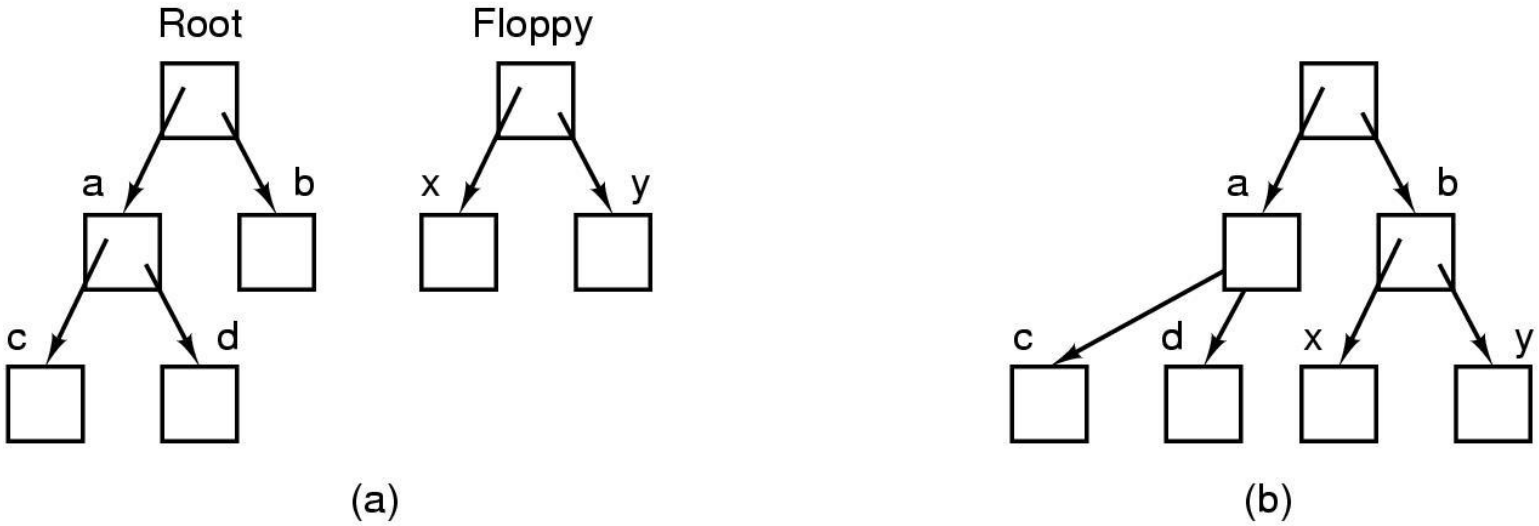
- Processes have three segments
  - Text: program code
  - Data: program data
    - Statically declared variables
    - Areas allocated by malloc() or new (heap)
  - Stack
    - Automatic variables
    - Procedure call information
- Address space growth
  - Text: doesn't grow
  - Data: grows "up"
  - Stack: grows "down"

Hierarchical file systems



File system for a university department

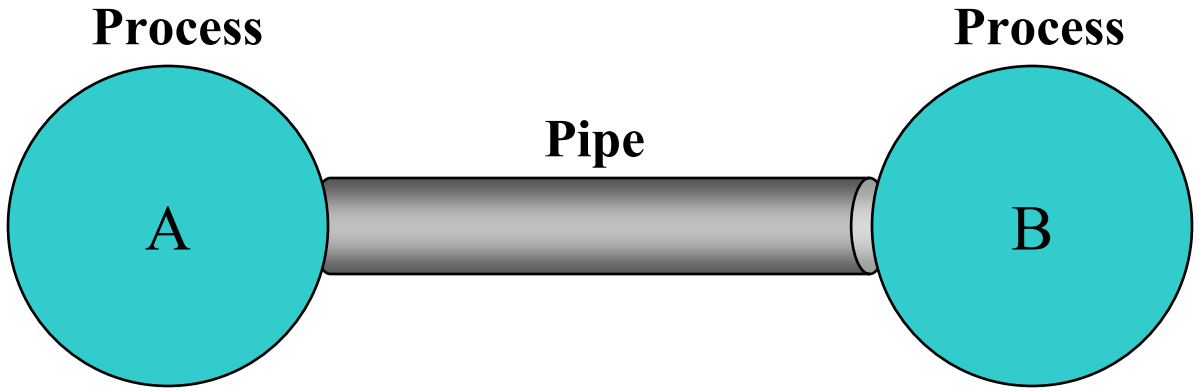
Mounting



- Before mounting,
  - files on floppy are inaccessible
- After mounting floppy on b,
  - files on floppy are part of file hierarchy

Inter-process communication

- Processes may want to exchange information with each other
- Many ways to do this, including
  - Network
  - Pipe (special file): A writes into pipe, and B reads from it

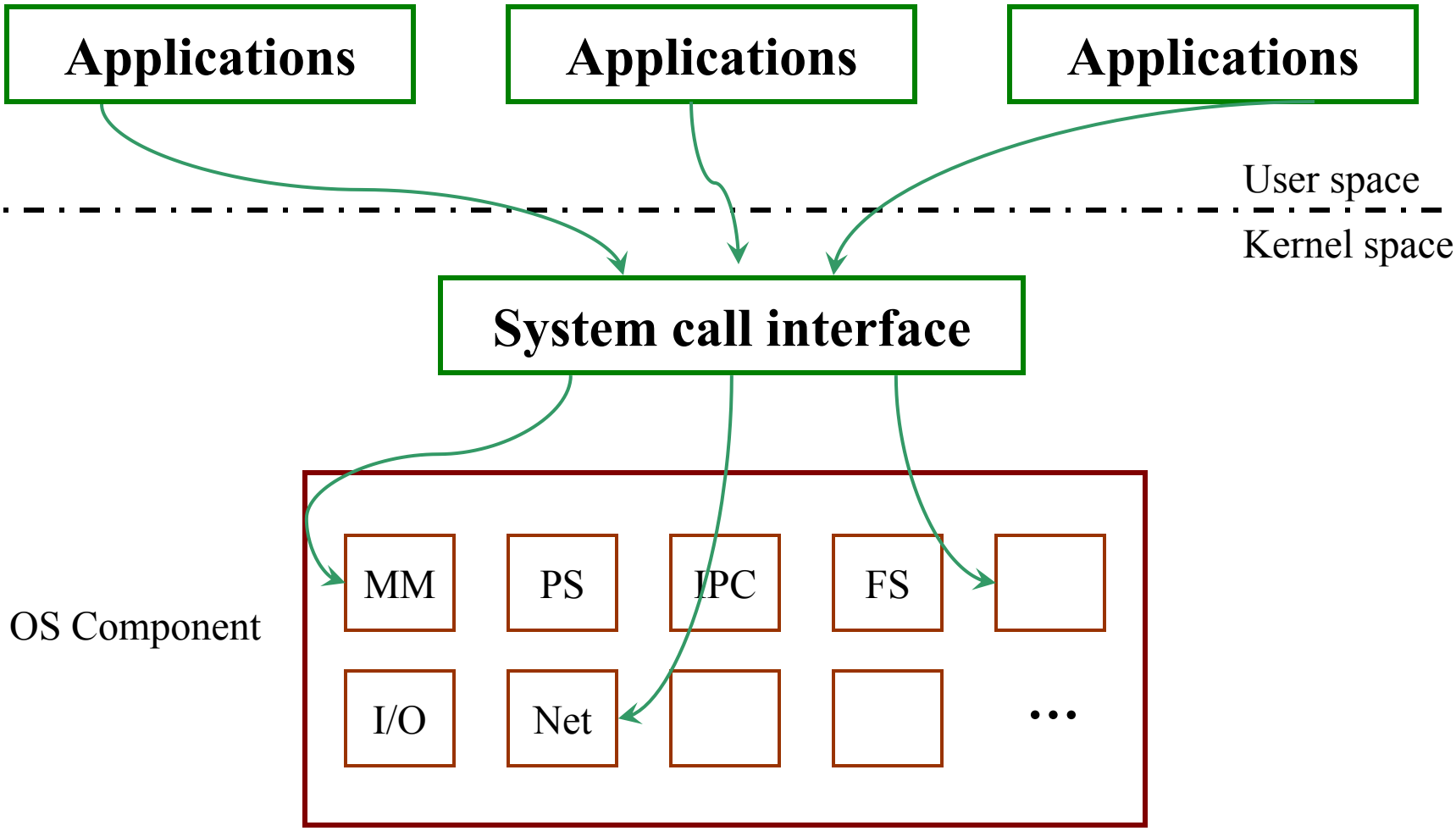


Two processes connected by a pipe

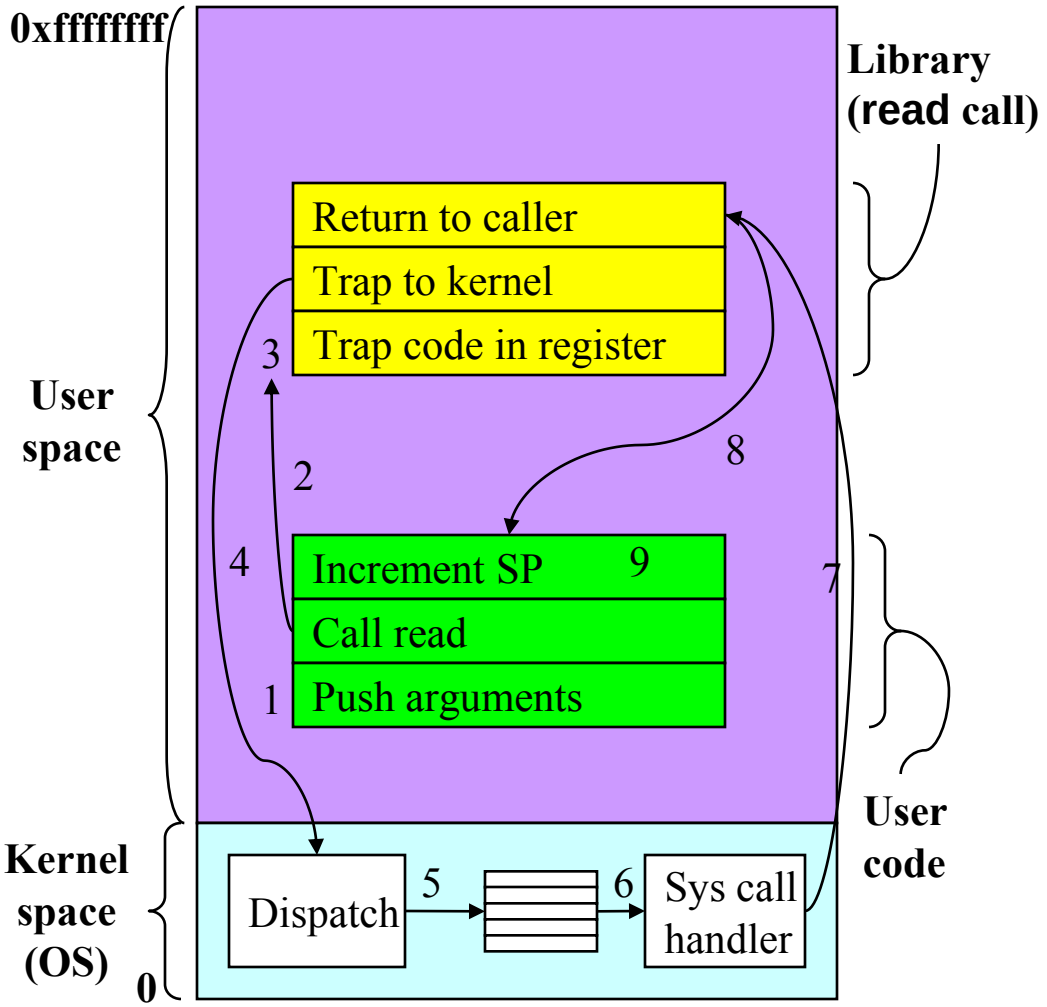
System calls

- Programs want the OS to perform a service
  - Access a file
  - Create a process
  - Others…
- Accomplished by system call
  - Program passes relevant information to OS
  - OS performs the service if
    - The OS is able to do so
    - The service is permitted for this program at this time
  - OS checks information passed to make sure it's OK
    - Don't want programs reading data into other programs' memory!

System calls…



Making a system call



- System call:
  read(fd,buffer,length)
- Program pushes arguments, calls library
- Library sets up trap, calls OS
- OS handles system call
- Control returns to library
- Library returns to user program
- There are 9 steps in making the system call

System calls for files & directories

Call	Description
fd = open(name,how)	Open a file for reading and/or writing
s = close(fd)	Close an open file
n = read(fd,buffer,size)	Read data from a file into a buffer
n = write(fd,buffer,size)	Write data from a buffer into a file
s = lseek(fd,offset,whence)	Move the “current” pointer for a file
s = stat(name,&buffer)	Get a file’s status information (in *buffer*)

MINIX System calls

More system calls

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
s = chdir(dirname)	Change the working directory
s = chmod(name,mode)	Change a file's protection bits
s = kill(pid,signal)	Send a signal to a process
seconds = time(&seconds)	Get the elapsed time since 1 Jan 1970

MINIX System calls

System Calls

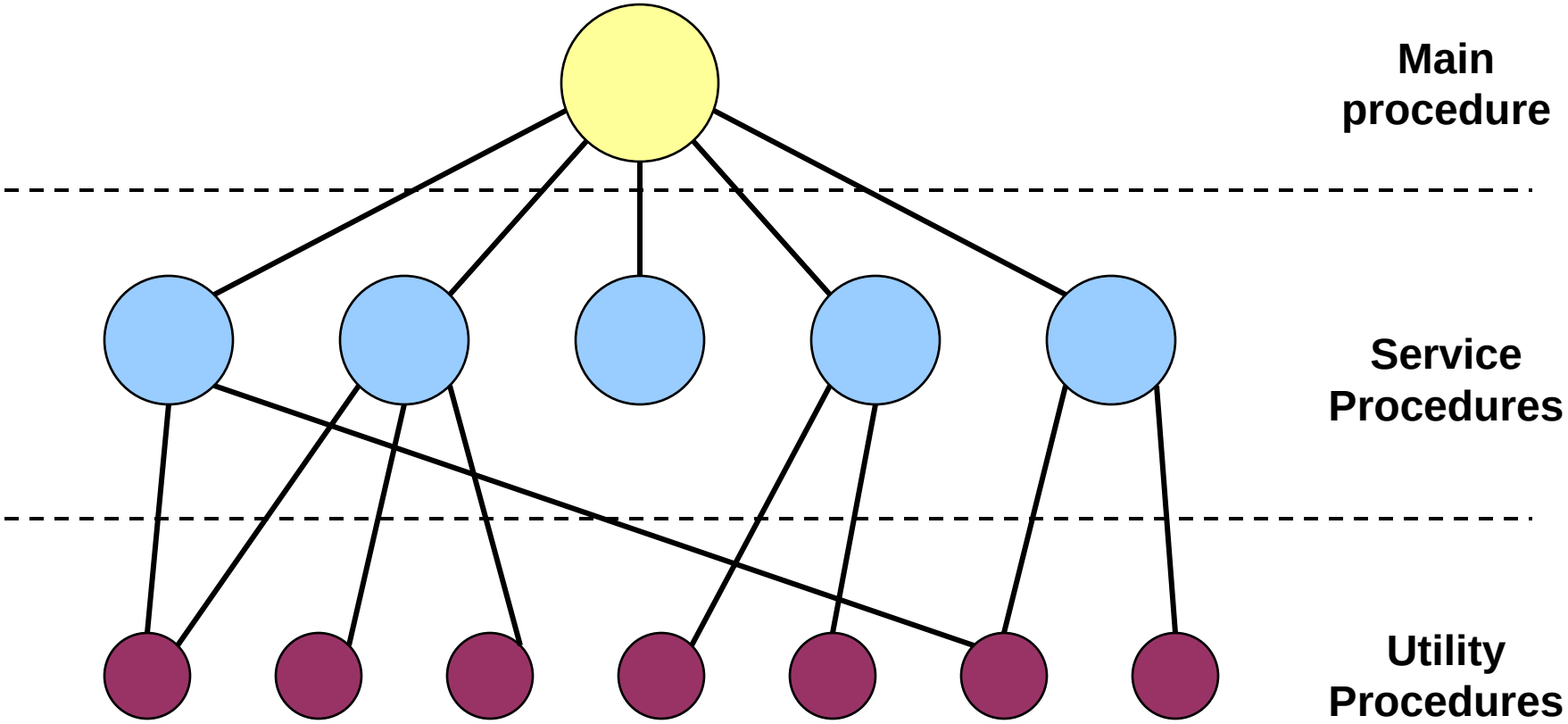
UNIX	Win32	Description
fork	CreateProcess	Create a new process
waitpid	WaitForSingleObject	Can wait for a process to exit
execve	(none)	CreateProcess = fork + execve
exit	ExitProcess	Terminate execution
open	CreateFile	Create a file or open an existing file
close	CloseHandle	Close a file
read	ReadFile	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

Some UNIX and Win32 API calls

Types of OS structures

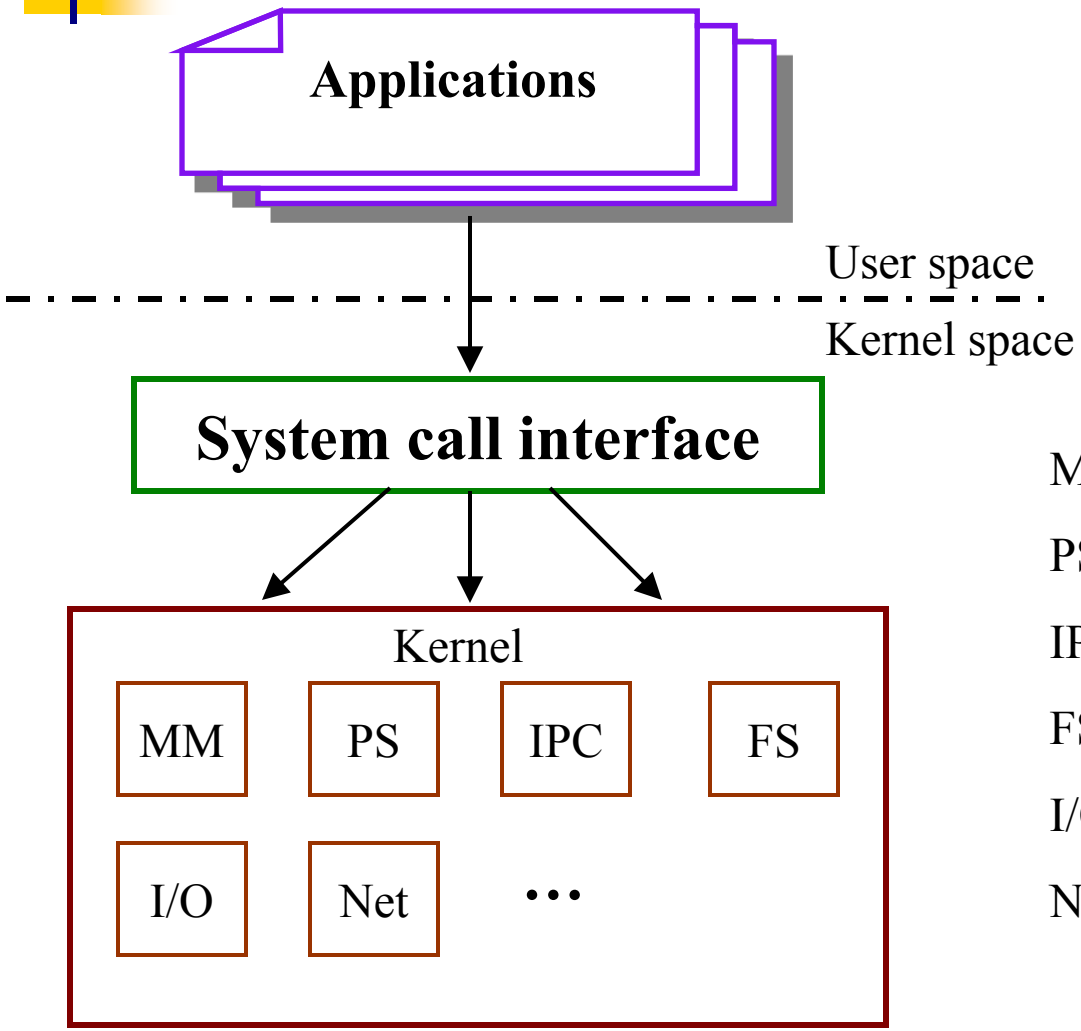
- Monolithic system
- Layered system
- Virtual machine
- Exokernel
- Client/Server

Monolithic system



Simple structuring model for a monolithic system

Monolithic system…



MM = Memory Manager

PS = Processor Scheduler

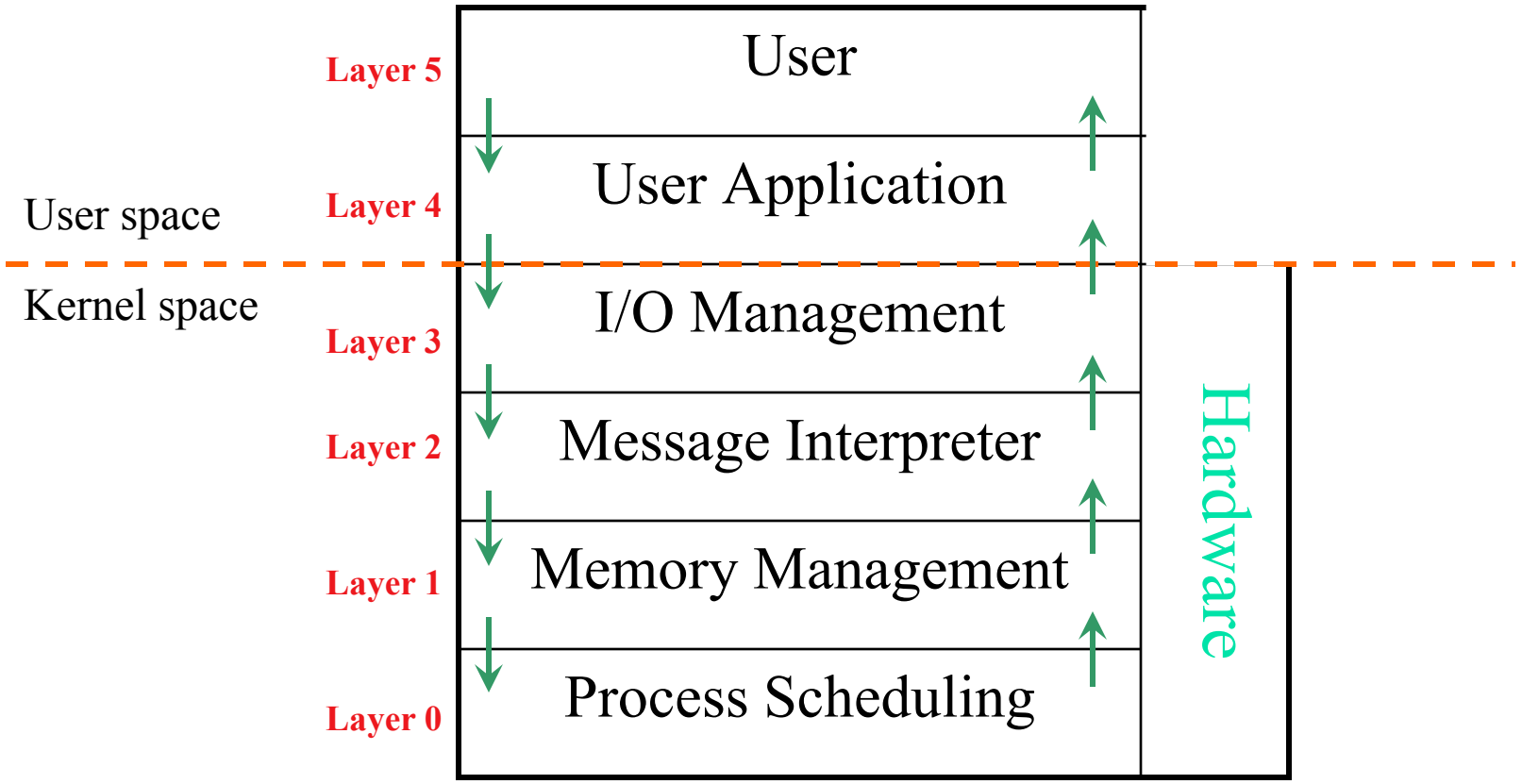
IPC = Inter Process Communication

FS = File System

I/O = Input/Output manager

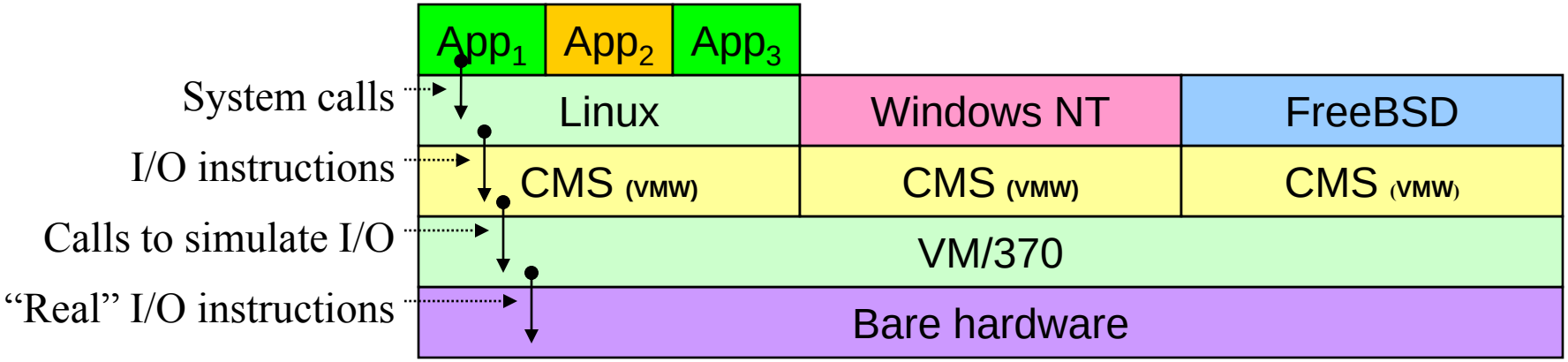
Net = Network manager

Layered system

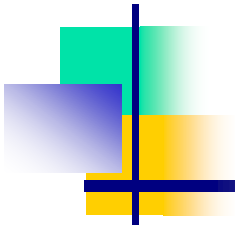


Structure of the THE operating system

Virtual machine



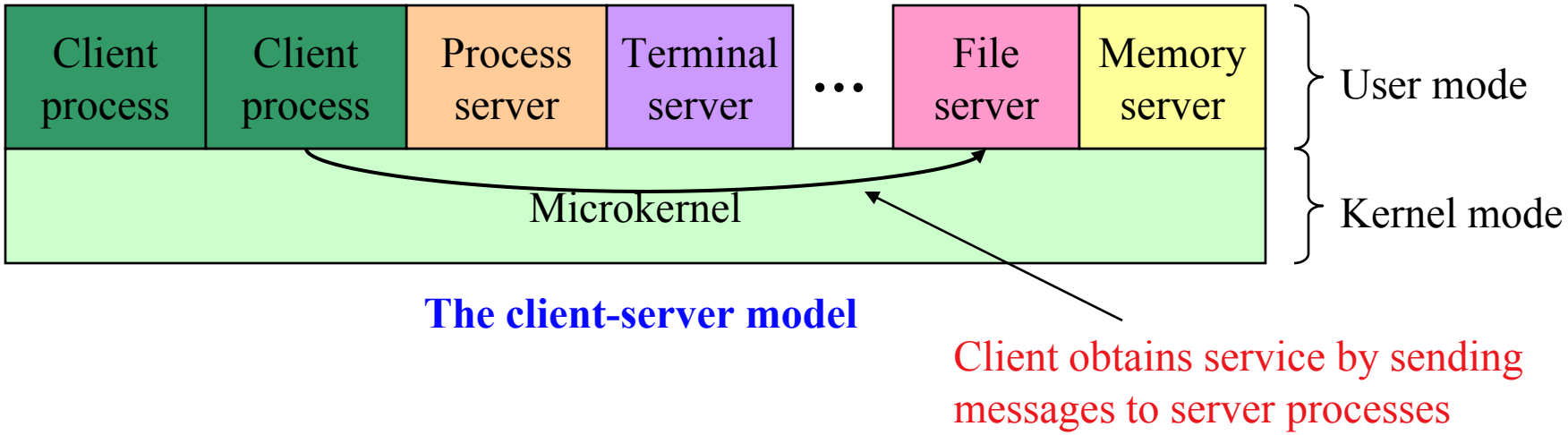
- First widely used in VM/370 with CMS (Conversational Monitor System)
- Available today in VMware
  - Allows users to run any x86-based OS on top of Linux or NT
- "Guest" OS can crash without harming underlying OS
  - Only virtual machine fails—rest of underlying OS is fine
- "Guest" OS can even use raw hardware
  - Virtual machine keeps things separated



Exokernel

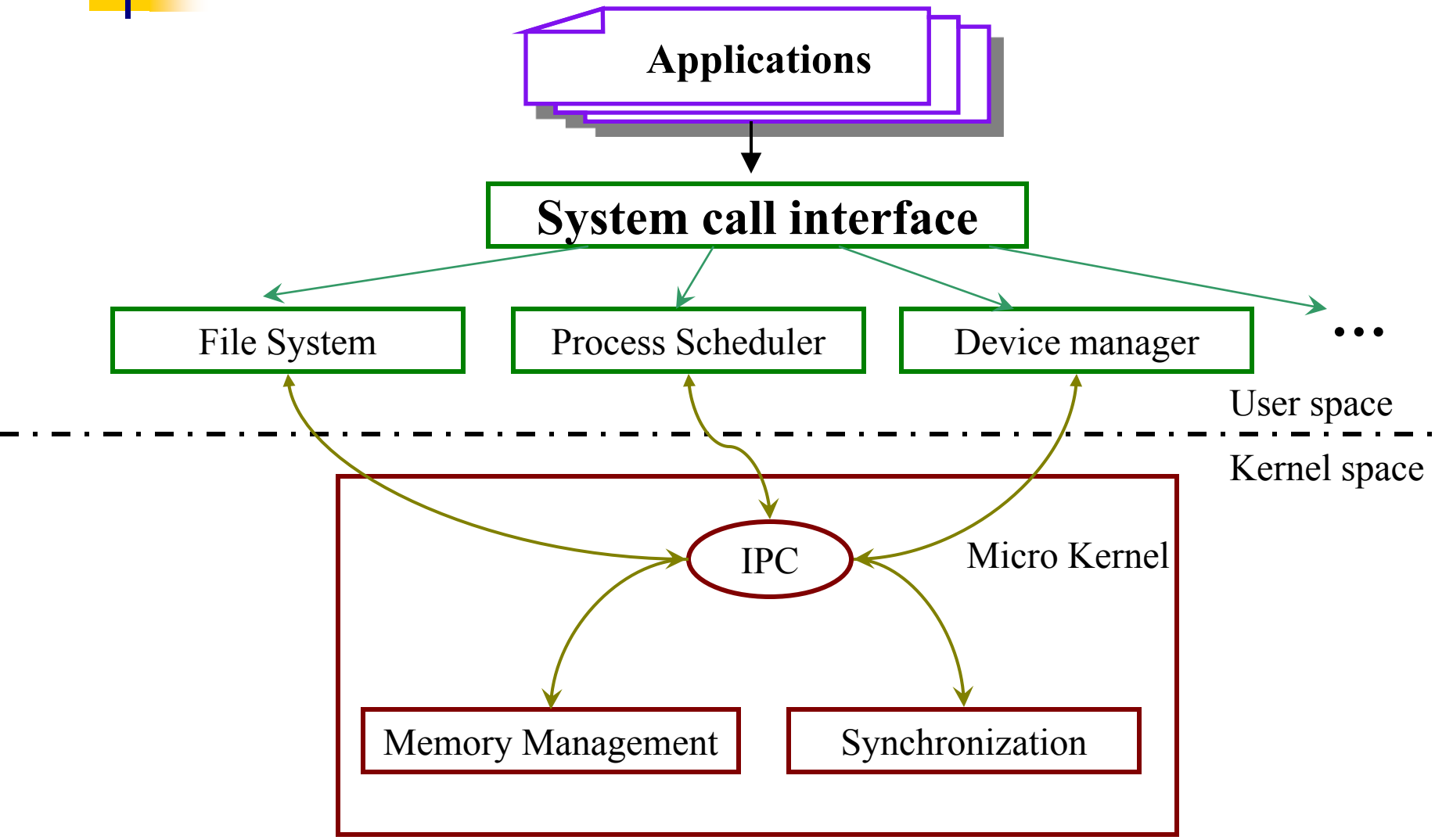
- Same as Virtual machine
- Assign one virtual computer to any user
- Allocate sub set of resource to any virtual machine

Client/Server

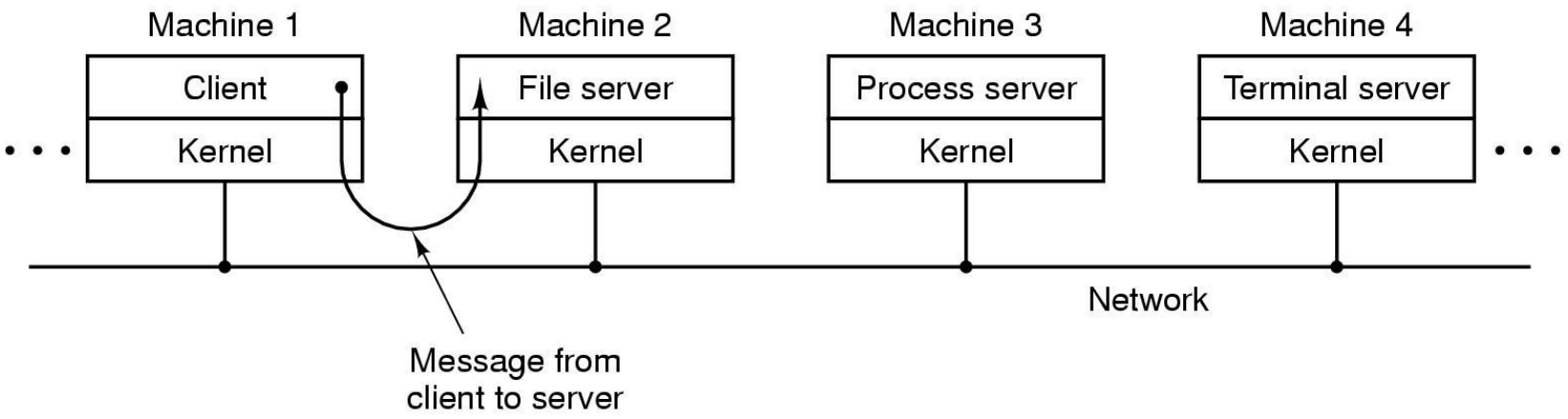


- Processes (clients and OS servers) don't share memory
  - Communication via message-passing
  - Separation reduces risk of "byzantine" failures
- Examples include Mach, QNX, early versions of Windows NT

Client/Server…



Client/Server…



The client-server model in a distributed system