Chapter 3: Operating Systems

Daniel Merkle

Based on Slides by J. Glenn Brookshear and DM510 Slides

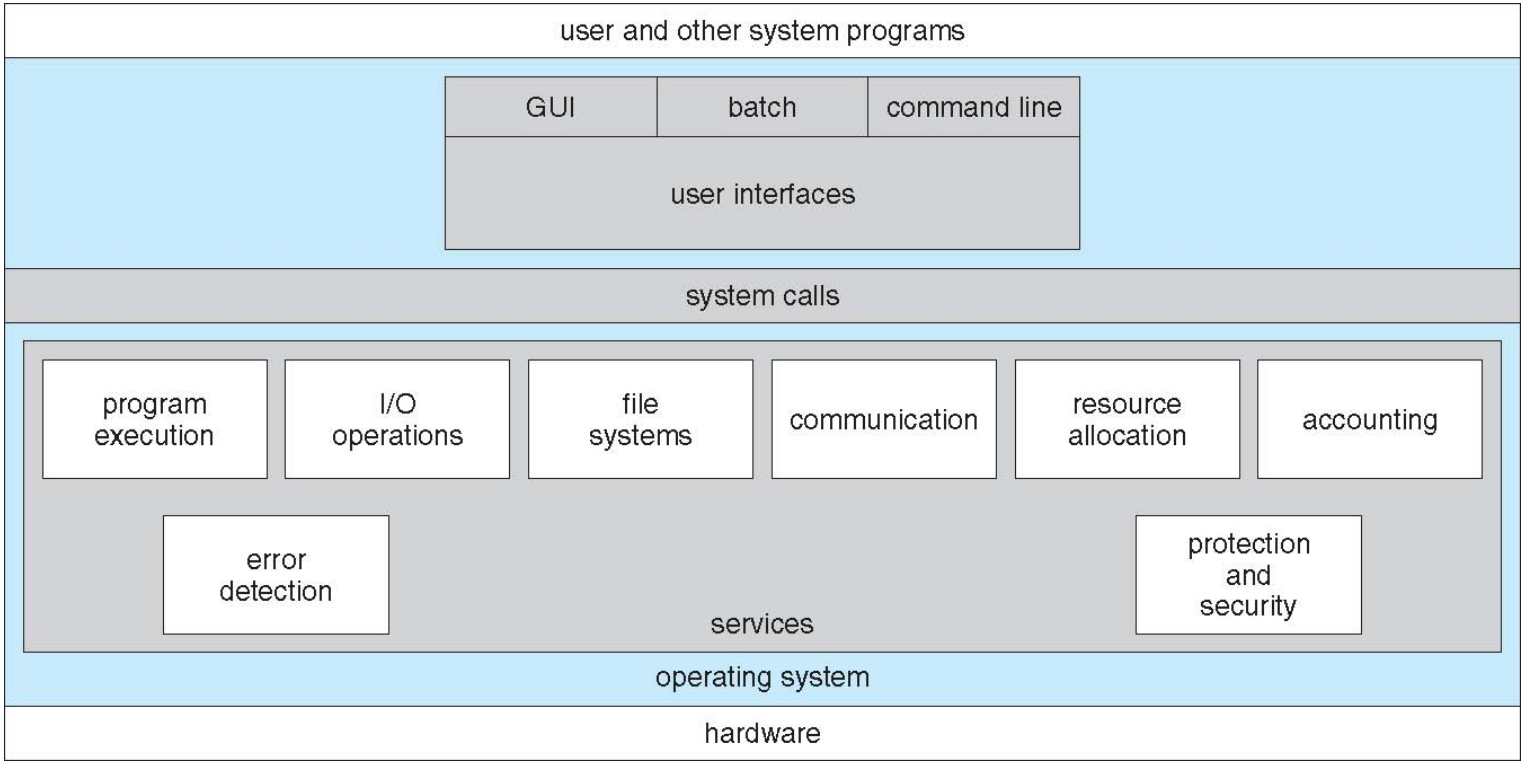
Functions of Operating Systems

- Control overall operation of computer
  - Store and retrieve files
  - Schedule programs for execution
  - Coordinate the execution of programs
  - ...

What is an Operating System?

- A program that acts as an intermediary between a user of a computer and the computer hardware
- Operating system goals:
  - Execute user programs and make solving user problems easier
  - Make the computer system convenient to use
  - Use the computer hardware in an efficient manner

A View of Operating System Services



→ Show Linux Kernel Map: http://www.makelinux.net/kernel_map

Chapter 3: Operating Systems

- 3.1 The History of Operating Systems
- 3.2 Operating System Architecture
- 3.3 Coordinating the Machine's Activities
- 3.4 Handling Competition Among Processes
- 3.5 Security

Evolution of Shared Computing

- Batch processing
- Interactive processing
- Multitasking is a method by which multiple tasks, also known as processes, share common processing resources such as a CPU.
- Scheduling Strategies:
  - Multiprogramming
  - Time-sharing
  - Real-time (strict deadlines)
- Multiprocessor machines

http://en.wikipedia.org/wiki/Computer_multitasking

Figure 3.1 **Batch processing**

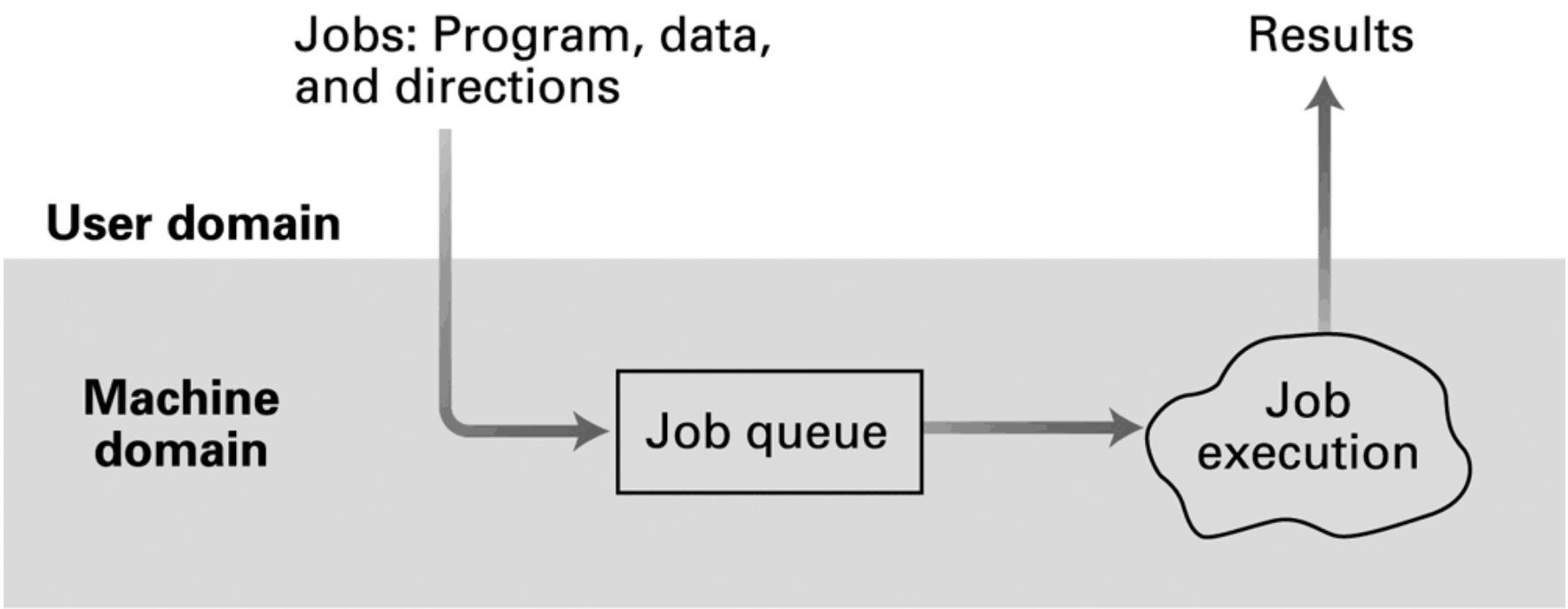
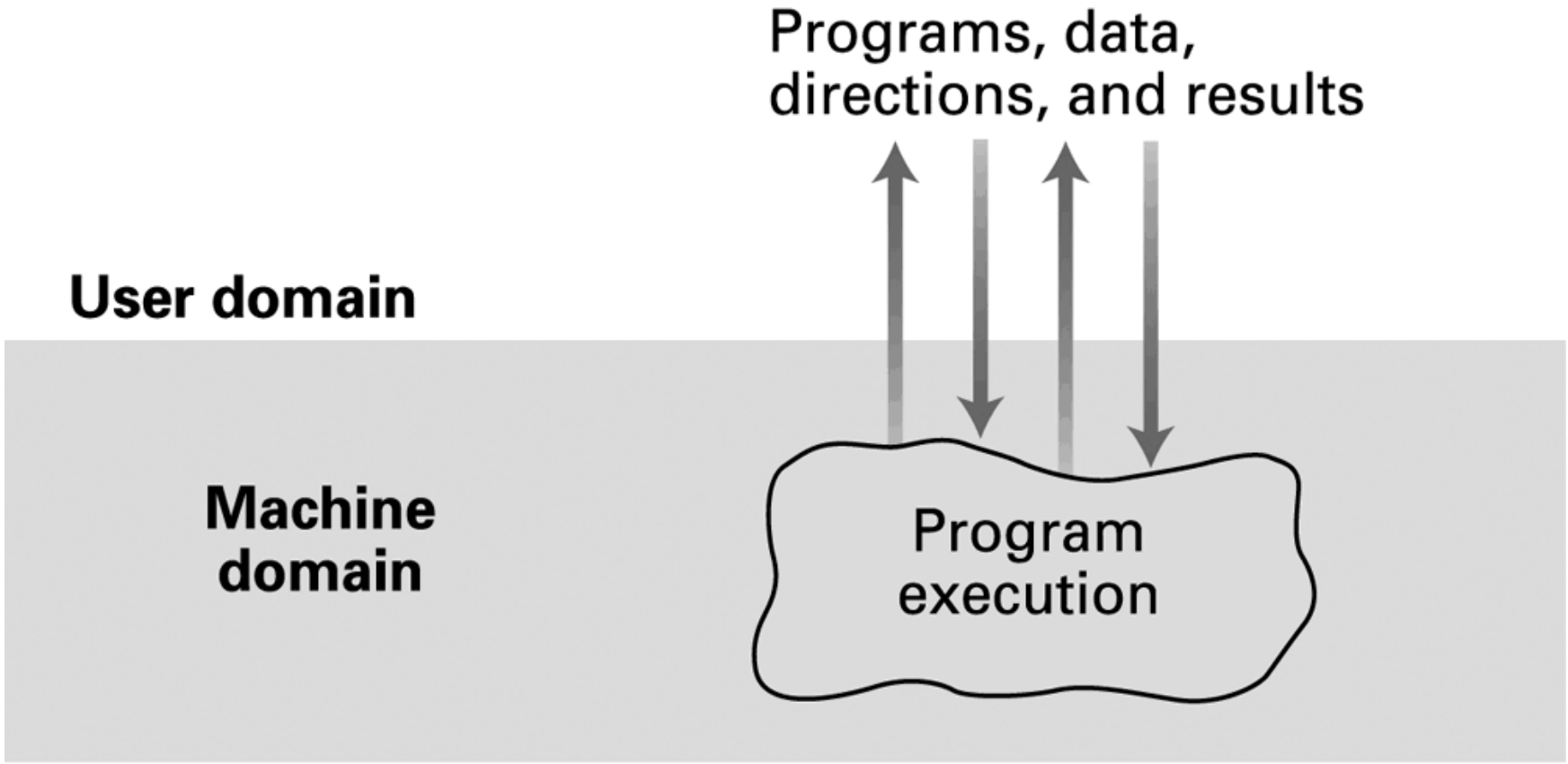
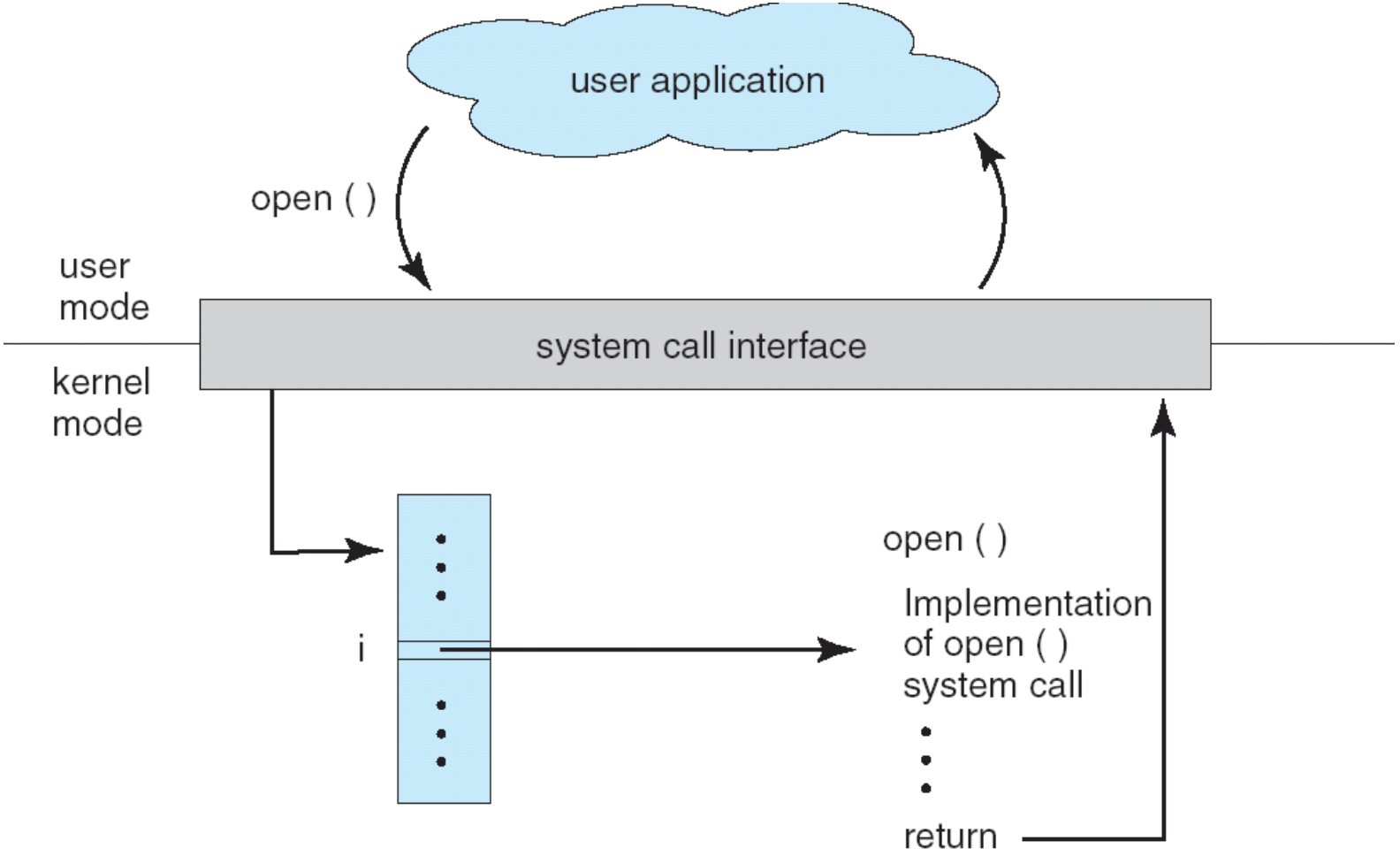


Figure 3.2 Interactive processing



System Call


user application
open ( )
user mode
system call interface
kernel mode
i
open ( )
Implementation of open ( ) system call
return

NERSC Franklin

- Used in DM818 – Parallel Computing
- NERSC Franklin massively parallel processing (MPP) system

  http://www.nersc.gov/nusers/systems/franklin/about.php

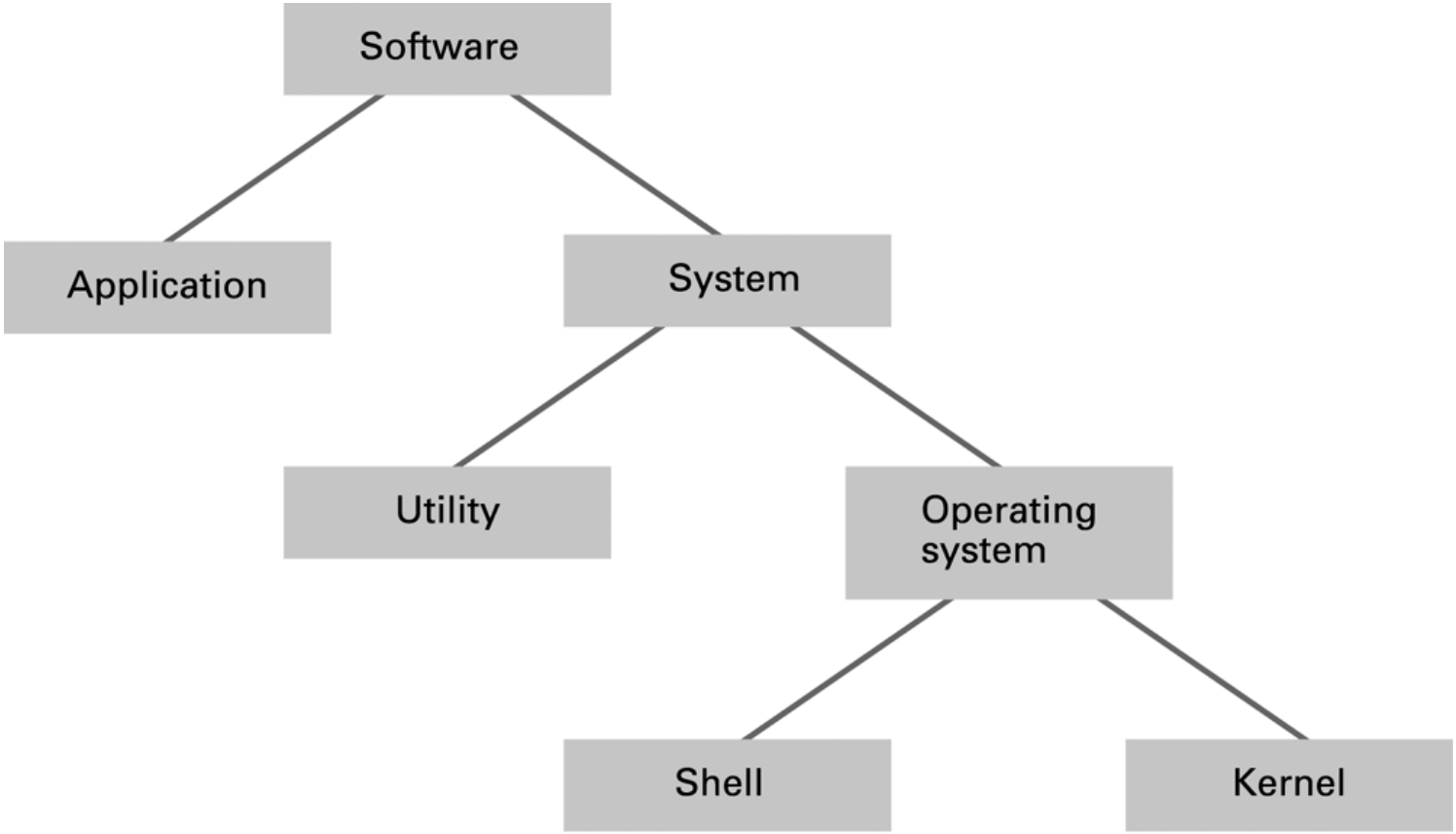
  http://www.top500.org/list/2009/06/100

- Batch Processing! Why?

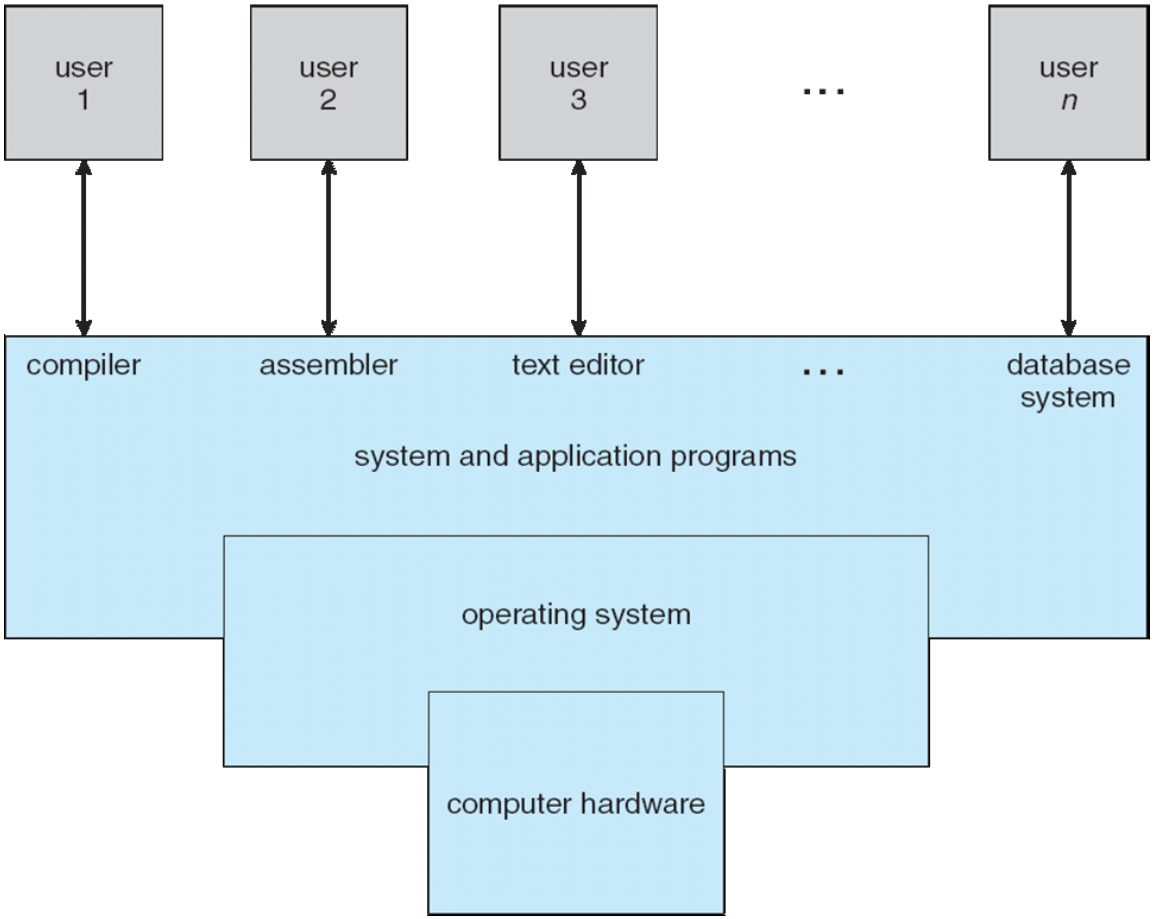
Types of Software

- Application software
  - Performs specific tasks for users
- System software
  - Provides infrastructure for application software
  - Consists of operating system and utility software

Figure 3.3 **Software classification**



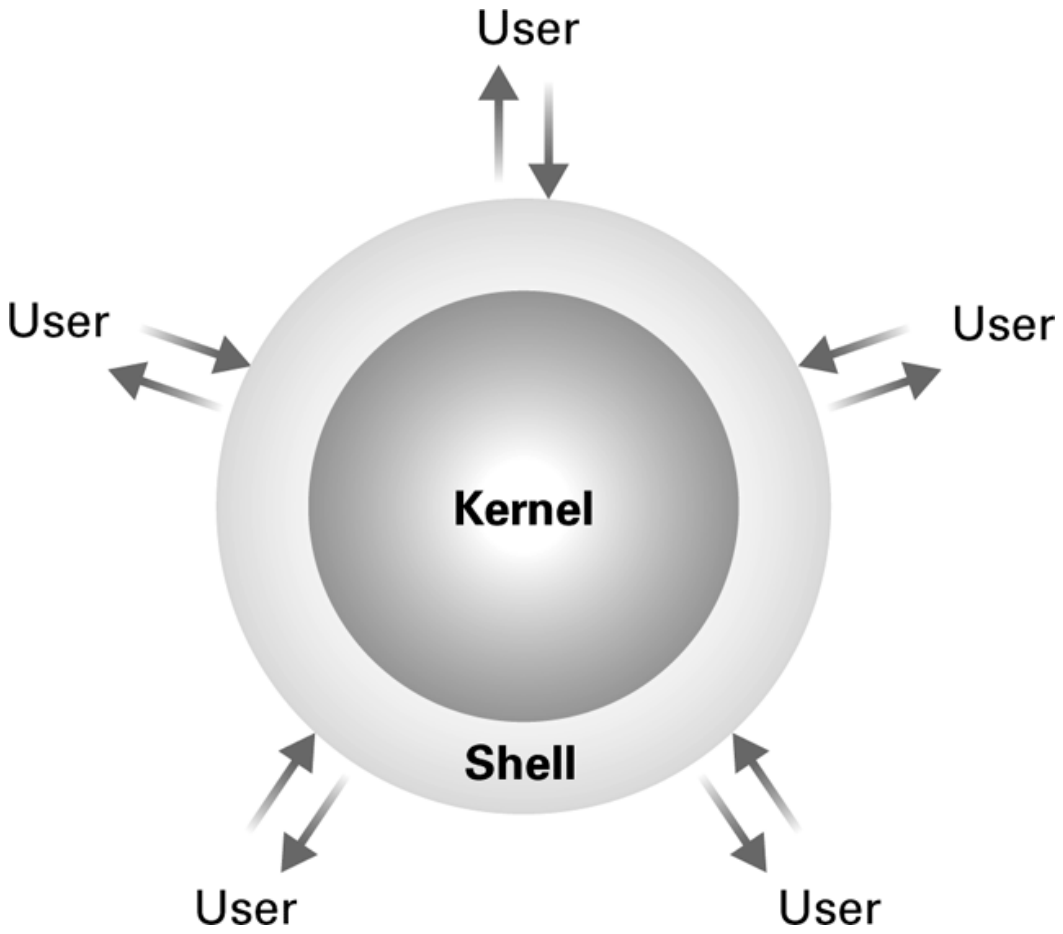
Four Components of a Computer System



Operating System Components

- **Shell:** Communicates with users, provides access to the services of a kernel
  - Text based
  - Graphical user interface (GUI)
- **Kernel:** Performs basic required functions
  - File manager
  - Device drivers
  - Memory manager
  - Scheduler and dispatcher

Figure 3.4 The shell as an interface between users and the operating system



File Manager

- **Directory** (or **Folder**): A user-created bundle of files and other directories (subdirectories)
- **Directory Path:** A sequence of directories within directories

Memory Manager

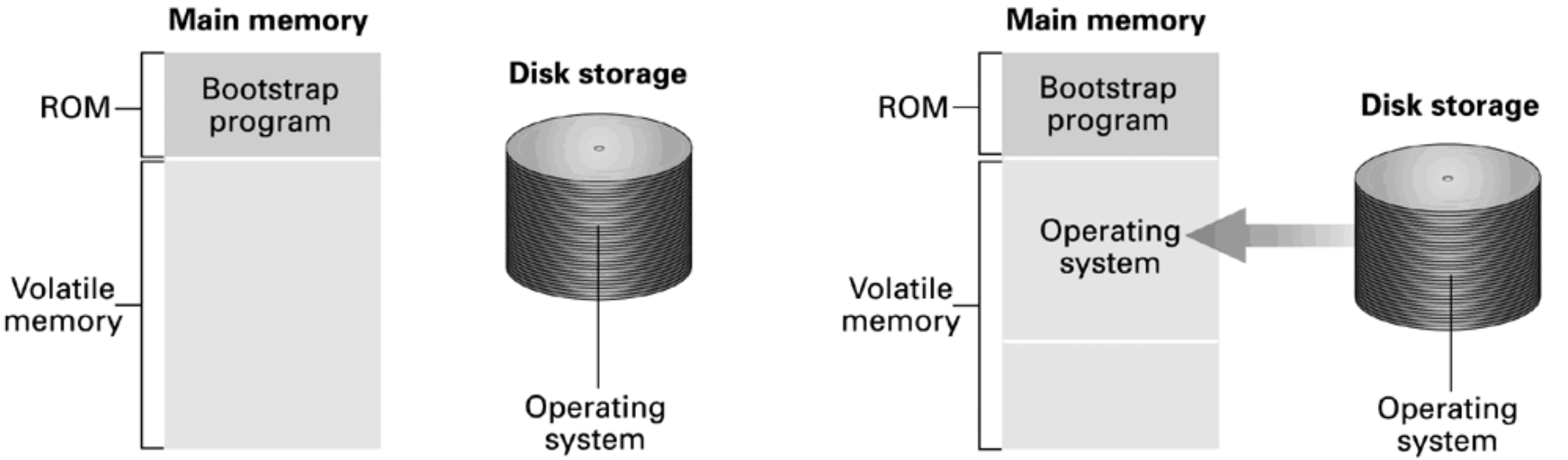
- Allocates space in main memory
- May create the illusion that the machine has more memory than it actually does (**virtual memory**) by moving blocks of data (**pages**) back and forth between main memory and mass storage

Getting it Started (Bootstrapping)

- **Bootstrap:** Program in ROM (example of firmware)
  - Run by the CPU when power is turned on
  - Transfers operating system from mass storage to main memory
  - Executes jump to operating system

  - The term Bootstrapping is often attributed to Rudolf Erich Raspe's story "The Surprising Adventures of Baron Münchausen", where the main character pulls himself out of a swamp, though it's disputed whether it was done by his hair or by his bootstraps.

Figure 3.5 The booting process



Step 1: Machine starts by executing the bootstrap program already in memory. Operating system is stored in mass storage.

Step 2: Bootstrap program directs the transfer of the operating system into main memory and then transfers control to it.

3.3 Coordinating the Machine's Activity

- **Process:** The activity of executing a program
- **Process State:** Current status of the activity (saved in the Process Control Block)
  - Program counter
  - General purpose registers
  - ...

Process Control Block (PCB)

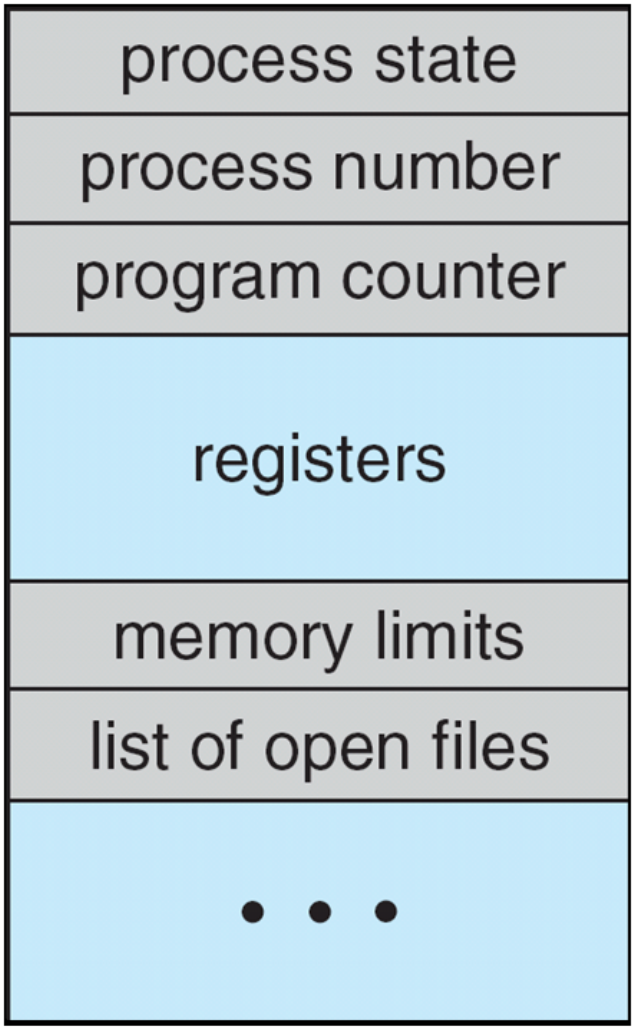
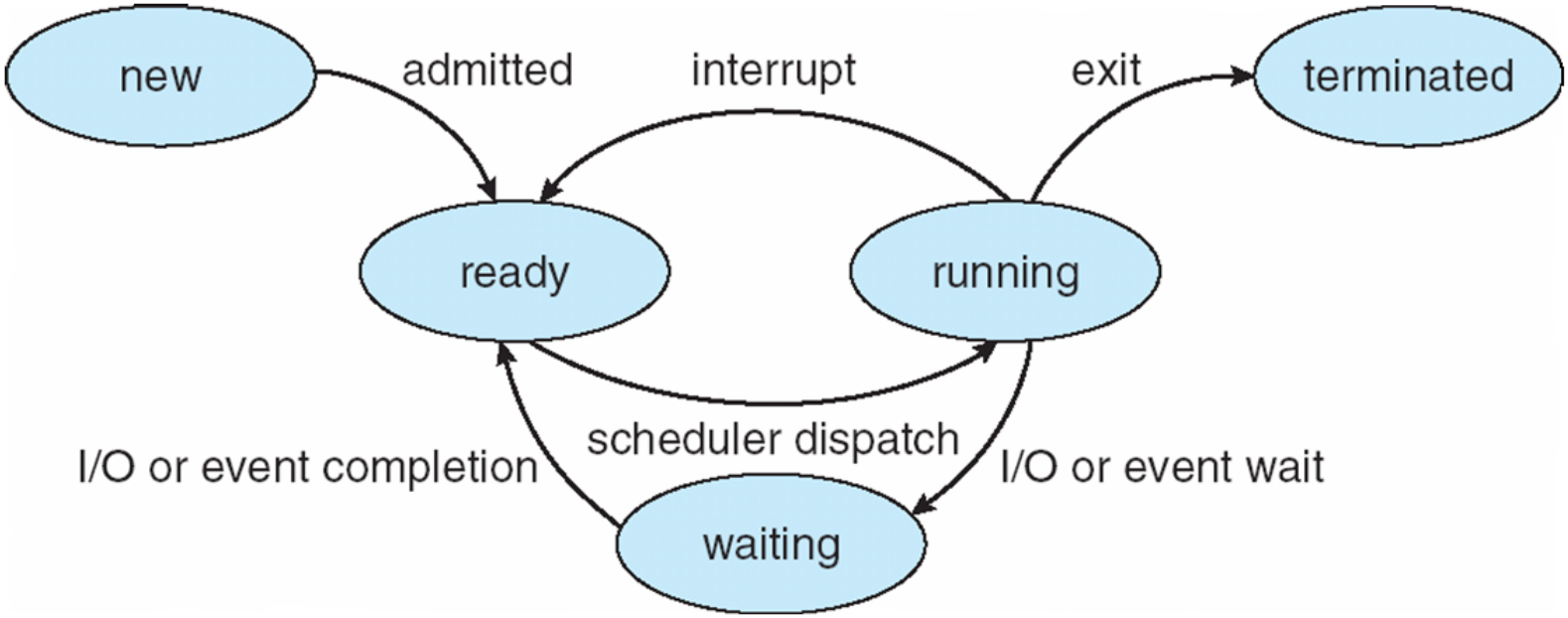


Diagram of Process State

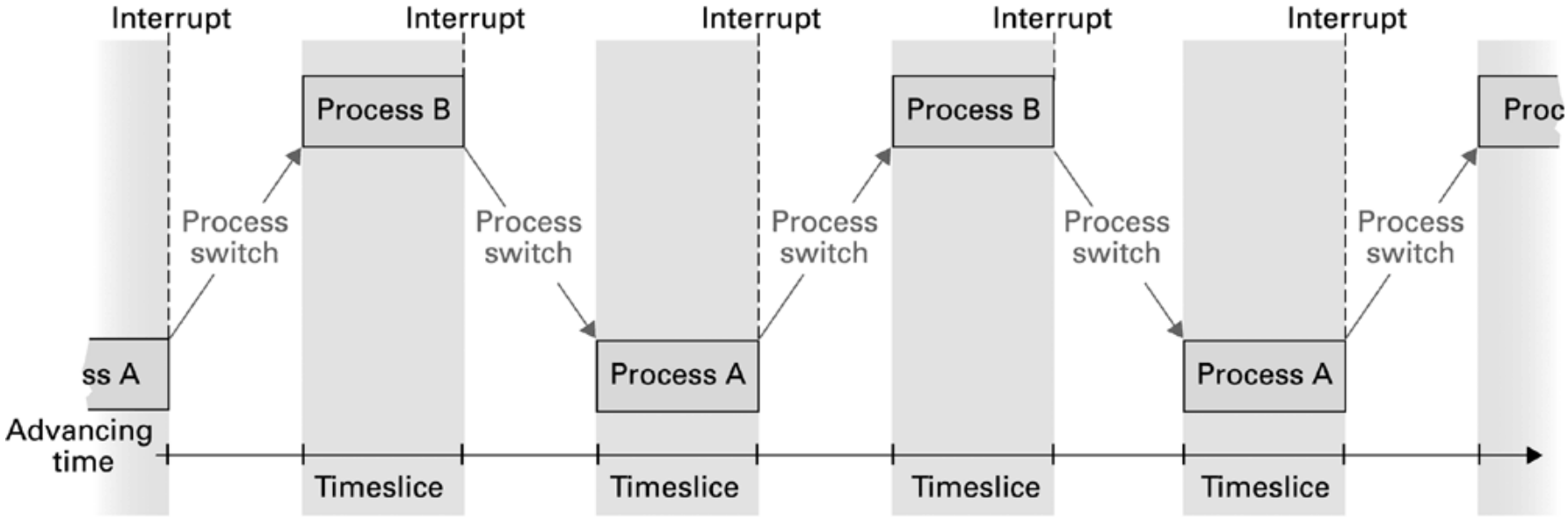


→ Show http://www.it.uom.gr/teaching/opsysanimation/animations/PROCESS.SWF

Process Administration

- **Scheduler:** Adds new processes to the process table and removes completed processes from the process table
- **Dispatcher:** Controls the allocation of time slices to the processes in the process table
  - The end of a time slice is signaled by an interrupt.
- Note that other definitions of Scheduler / Dispatcher exist (closer to reality).

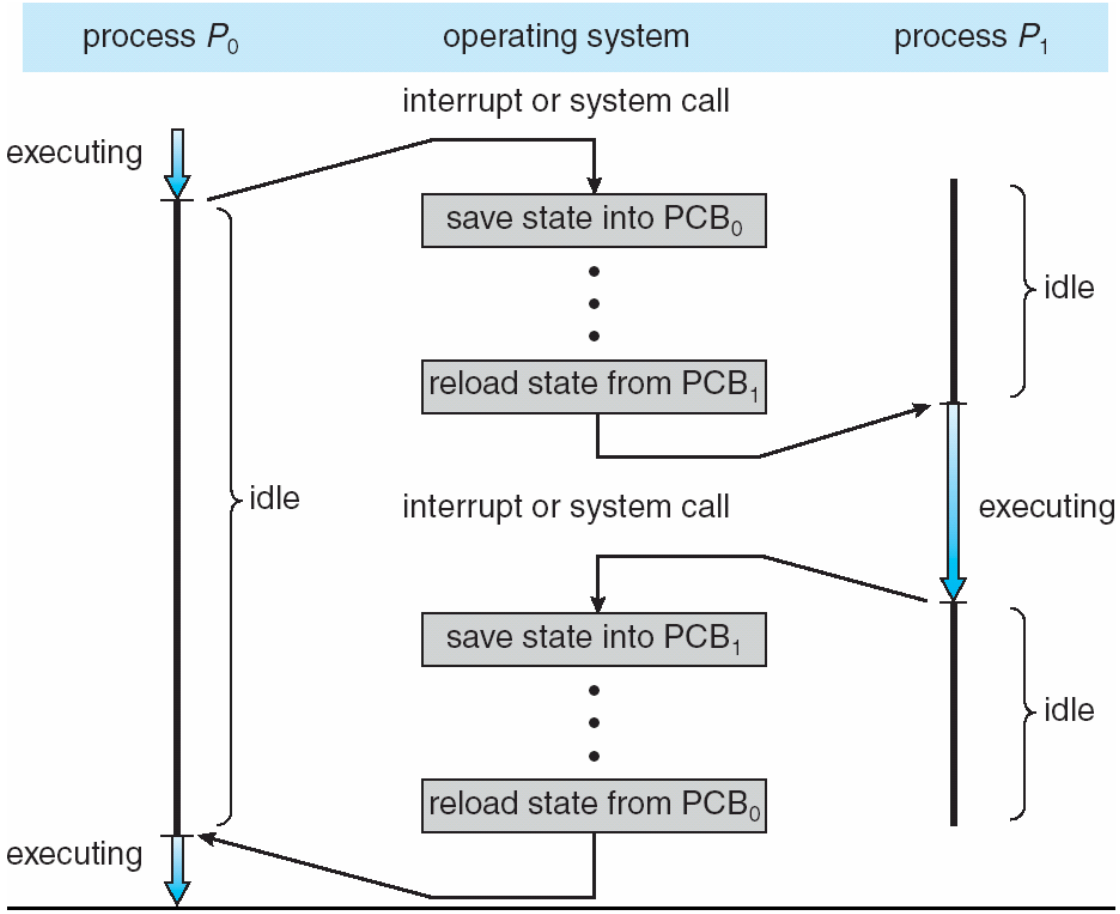
Figure 3.6 Time-sharing between process A and process B



Context Switch

- When CPU switches to another process, the system must save the state of the old process and load the saved state for the new process via a context switch
- Context of a process represented in the PCB
- Context-switch time is overhead; the system does no useful work while switching, therefore it has to be fast

CPU Switch From Process to Process

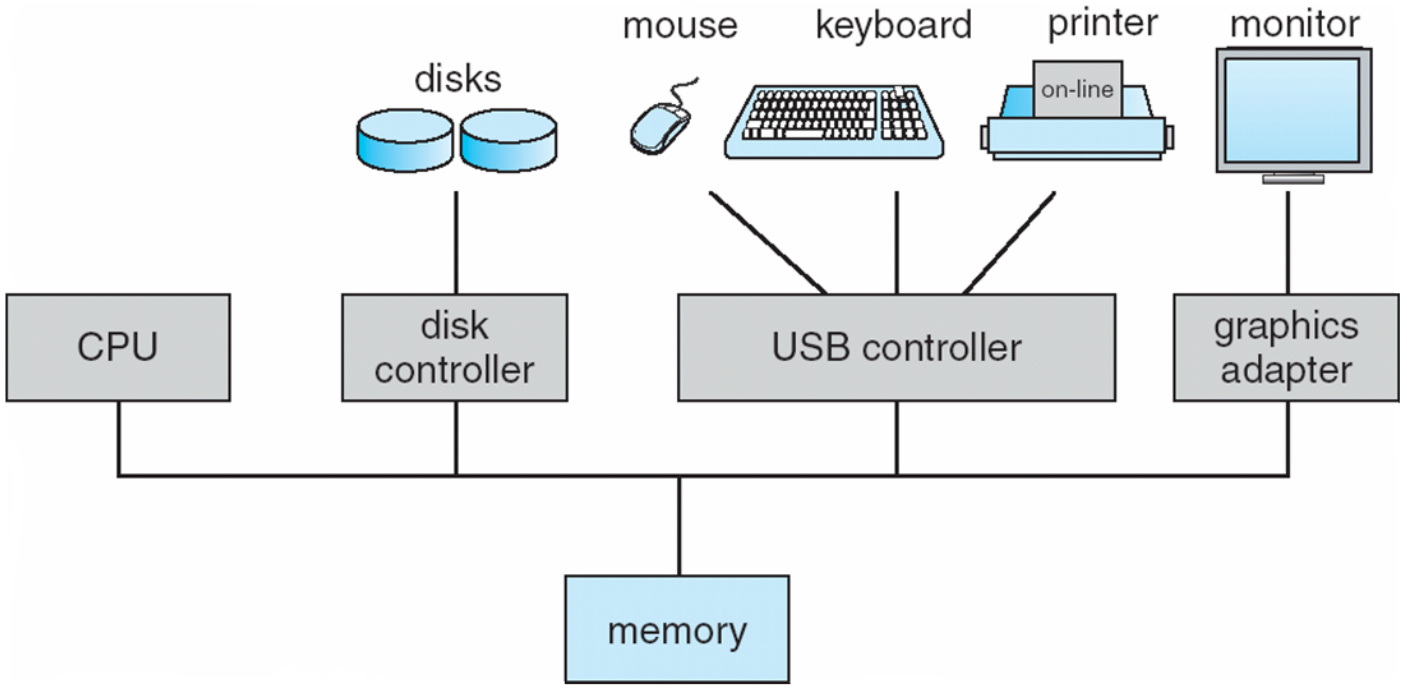


Interrupts


"PLEASE FEEL FREE TO INTERRUPT
IF YOU HAVE A QUESTION."
TOM
©1995 Tom Swanson

Computer System Organization

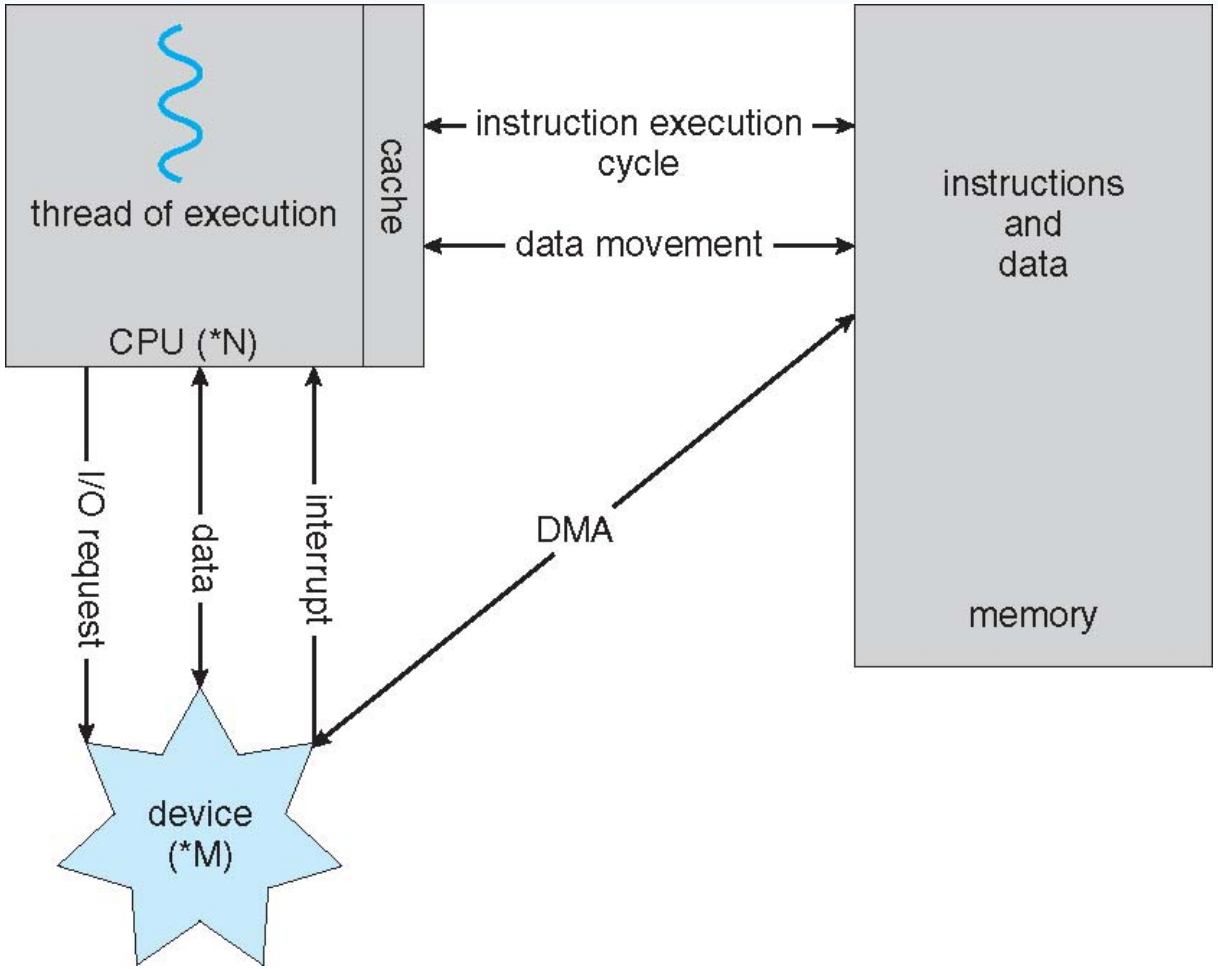
- Computer-system operation
  - One or more CPUs, device controllers connect through common bus providing access to shared memory
  - Concurrent execution of CPUs and devices competing for memory cycles



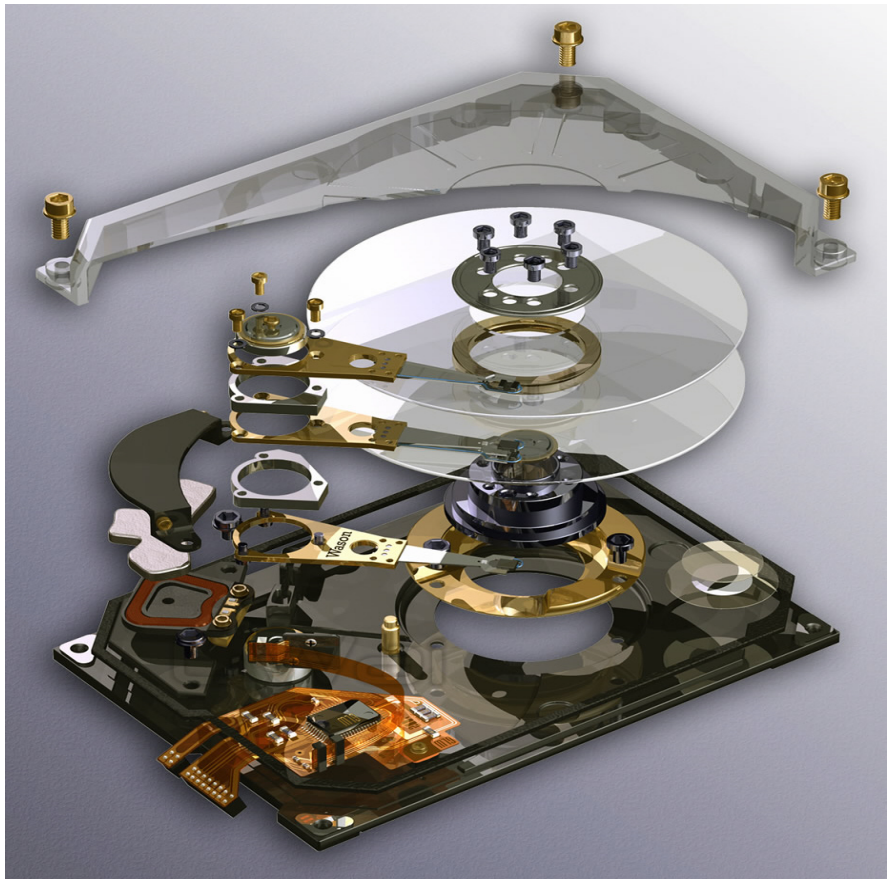
Computer-System Operation

- I/O (input/output) devices and the CPU can execute concurrently
- Each device controller is in charge of a particular device type
- Each device controller has a local buffer
- CPU moves data from/to main memory to/from local buffers
- I/O is from the device to local buffer of controller
- Device controller informs CPU that it has finished its operation by causing an *interrupt*

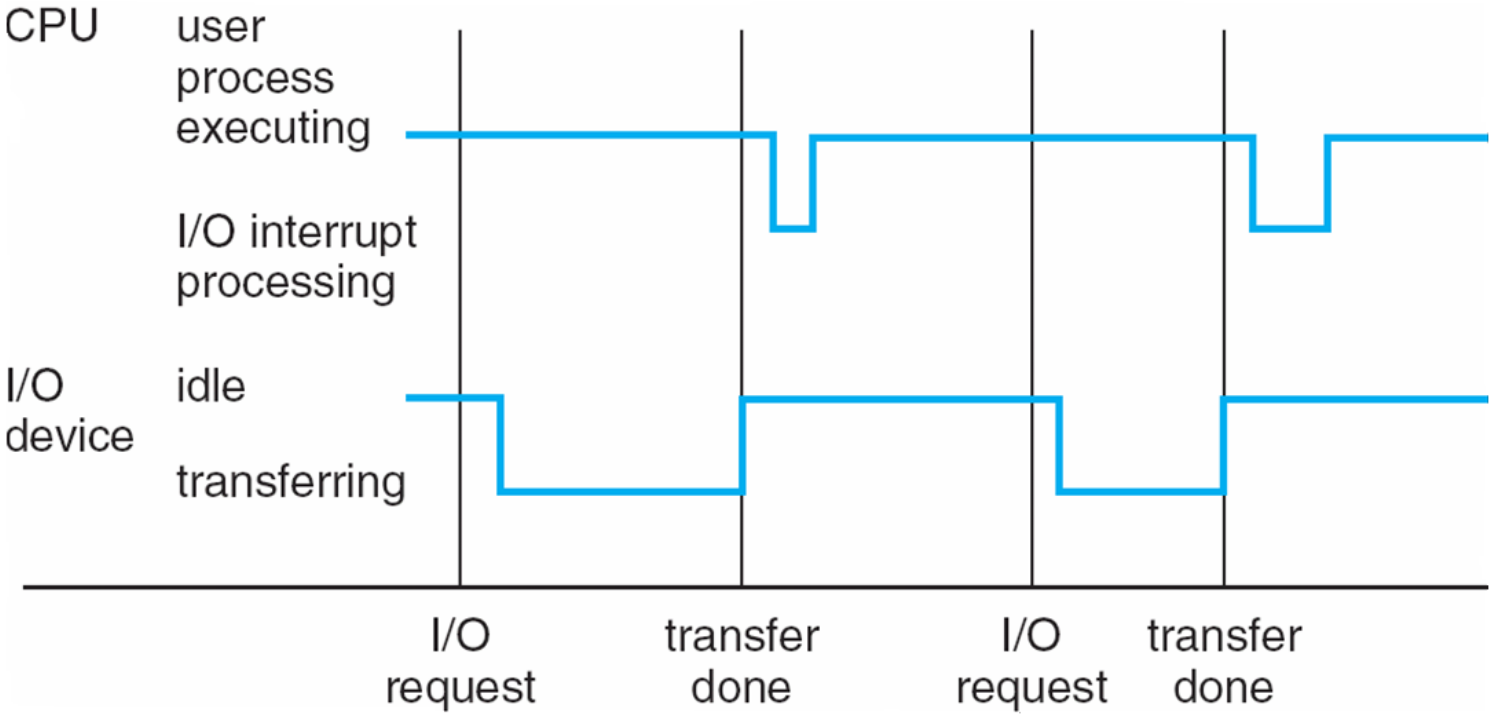
How a Modern Computer Works



Device Example: Hard Disk (+ Controller)



Interrupt Timeline



Interrupt timeline for a single process doing output

Scheduling Processes

- Select from among the processes in memory that are ready to execute, and allocates the CPU to one of them
- CPU scheduling decisions may take place when a process:
  1. Switches from running to waiting state
  2. Switches from running to ready state
  3. Switches from waiting to ready
  4. Terminates
- Scheduling under 1 and 4 is **nonpreemptive**
- All other scheduling is **preemptive**

→ Additional reading material in the Blackboard System

Scheduling Criteria

- **CPU utilization** – keep the CPU as busy as possible
- **Throughput** – # of processes that complete their execution per time unit
- **Turnaround time** – amount of time to execute a particular process
- **Waiting time** – amount of time a process has been waiting in the ready queue
- **Response time** – amount of time it takes from when a request was submitted until the first response is produced, not output (for time-sharing environment)

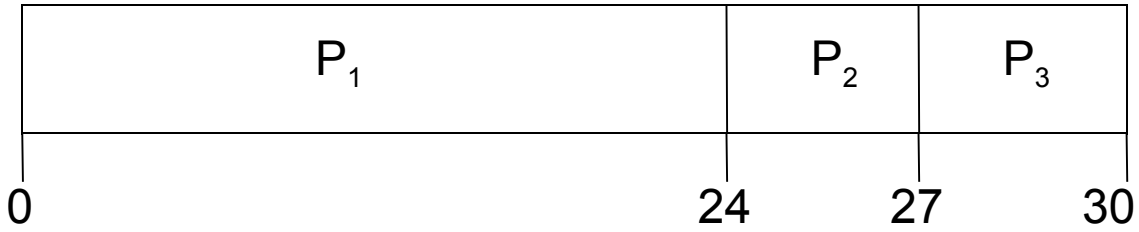
Scheduling Algorithm Optimization Criteria

- Max CPU utilization
- Max throughput
- Min turnaround time
- Min waiting time
- Min response time

First-Come, First-Served (FCFS) Scheduling

Process	Burst Time (Length)
P_1	24
P_2	3
P_3	3

- Suppose that the processes arrive in the order: P_1 , P_2 , P_3
  The Gantt Chart for the schedule is:



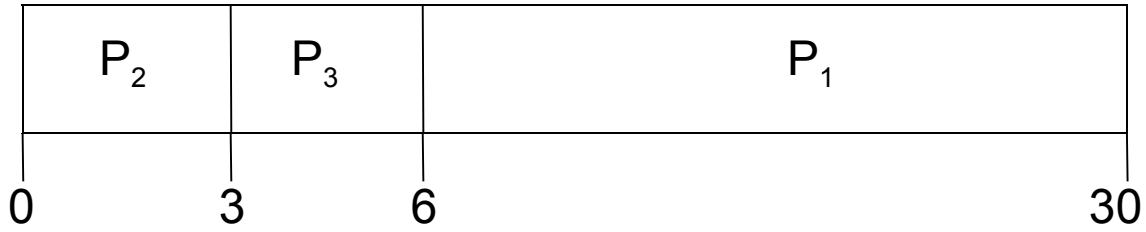
- Waiting time for P_1 = 0; P_2 = 24; P_3 = 27
- Average waiting time: (0 + 24 + 27)/3 = 17

FCFS Scheduling (Cont)

Suppose that the processes arrive in the order

P_2 , P_3 , P_1

- The Gantt chart for the schedule is:



- Waiting time for $P_1 = 6$; $P_2 = 0$, $P_3 = 3$
- Average waiting time: (6 + 0 + 3)/3 = 3
- Much better than previous case
- *Convoy effect* short process behind long process

Shortest-Job-First (SJF) Scheduling

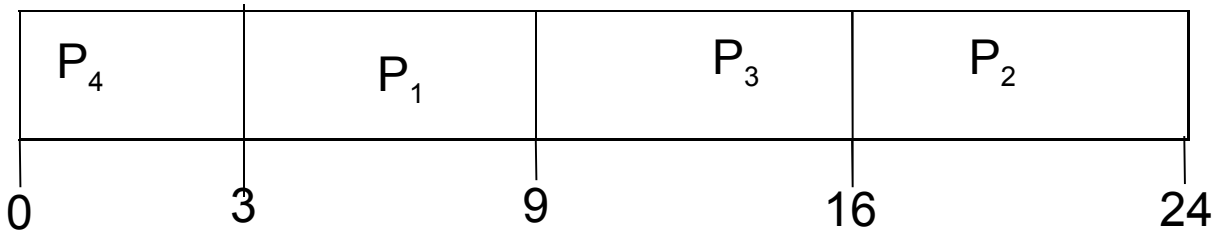
- Associate with each process the length of its next CPU burst. Use these lengths to schedule the process with the shortest time

- SJF is optimal – gives minimum average waiting time for a given set of processes
  - The difficulty is knowing the length of the next CPU request (not discussed in this lecture)

Example of SJF

Process	Burst Time
P_1	6
P_2	8
P_3	7
P_4	3

- SJF scheduling chart



- Average waiting time = (3 + 16 + 9 + 0) / 4 = 7

Priority Scheduling

- A priority number (integer) is associated with each process
- The CPU is allocated to the process with the highest priority (smallest integer ≡ highest priority)
  - Preemptive
  - nonpreemptive
- SJF is a priority scheduling where priority is the predicted next CPU burst time
- Problem ≡ **Starvation** – low priority processes may never execute
- Solution ≡ **Aging** – as time progresses increase the priority of the process

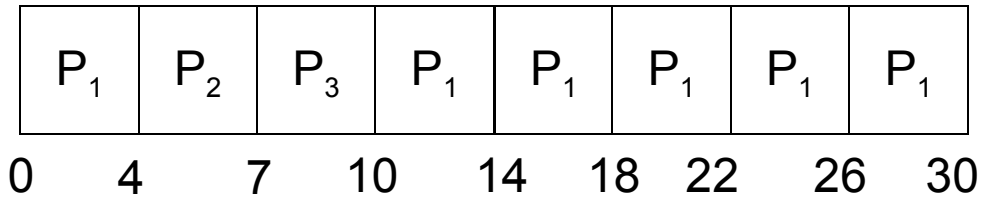
Round Robin (RR)

- Each process gets a small unit of CPU time (*time quantum*), usually 10-100 milliseconds. After this time has elapsed, the process is preempted and added to the end of the ready queue.
- If there are n processes in the ready queue and the time quantum is q, then each process gets $1/n$ of the CPU time in chunks of at most q time units at once. No process waits more than $(n-1)q$ time units.
- Performance
  - q large $\Rightarrow$ FIFO
  - q small $\Rightarrow$ q must be large with respect to context switch, otherwise overhead is too high

Example of RR with Time Quantum = 4

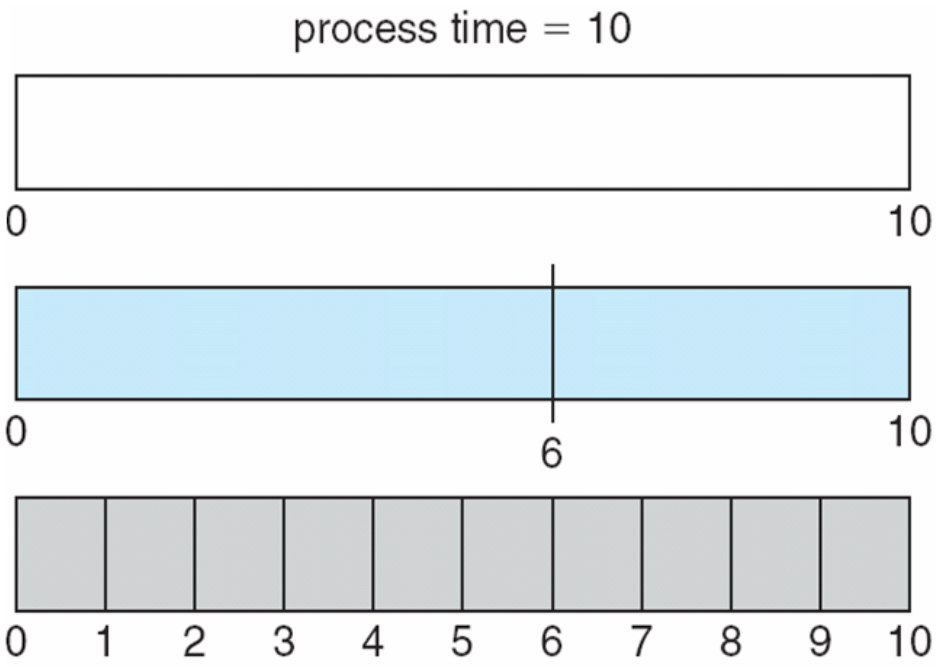
Process	Burst Time
P_1	24
P_2	3
P_3	3

- The Gantt chart is:



- Typically, higher average turnaround than SJF, but better *response*

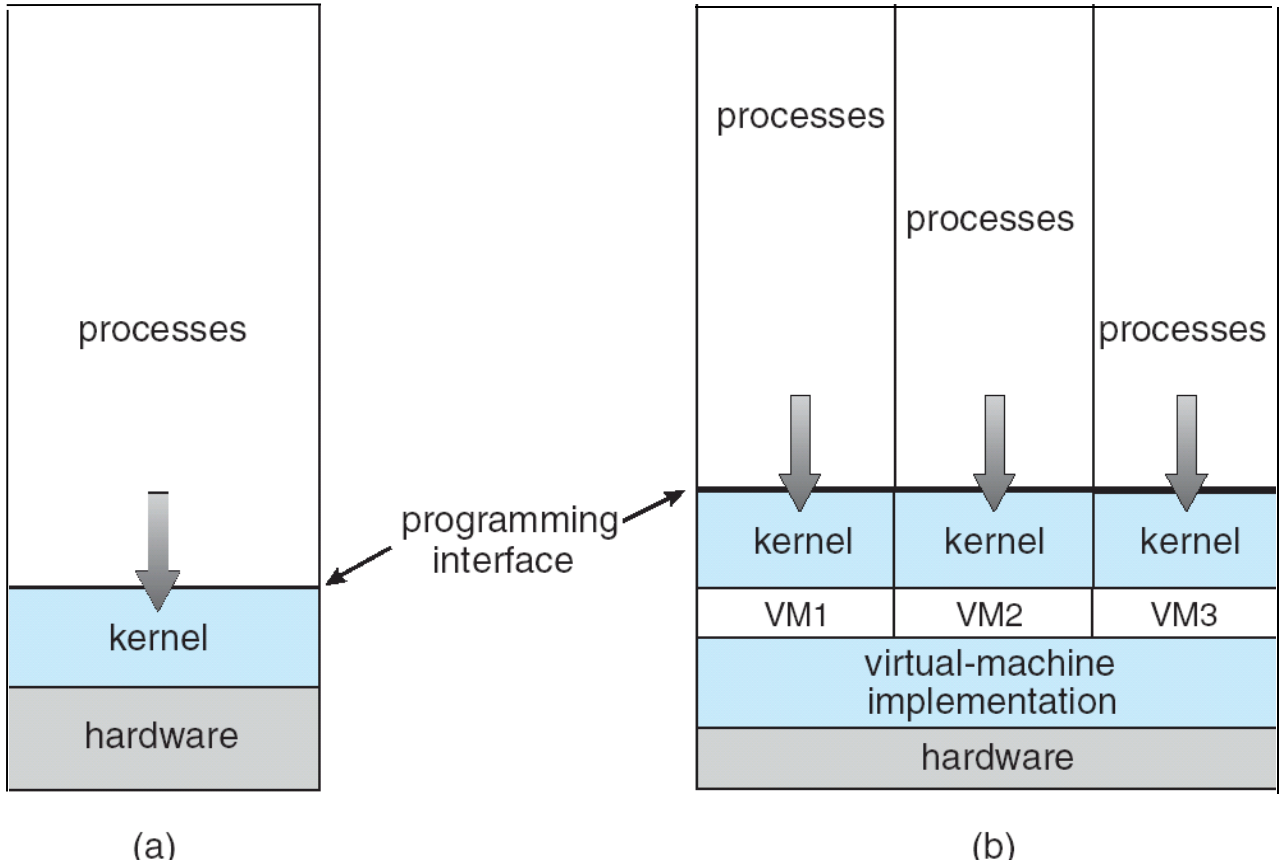
Time Quantum and Context Switch Time



quantum	context switches
12	0
6	1
1	9

→ Show Solaris

Excursus: Virtual Machines



(a) Nonvirtual machine (b) virtual machine

Example Solaris Dispatch Table

priority	time quantum	time quantum expired	return from sleep
0	200	0	50
5	200	0	50
10	160	0	51
15	160	5	51
20	120	10	52
25	120	15	52
30	80	20	53
35	80	25	54
40	40	30	55
45	40	35	56
50	40	40	58
55	40	45	58
59	20	49	59

→ show `dispadmin` in OpenSolaris

3.4 Handling Competition for Resources

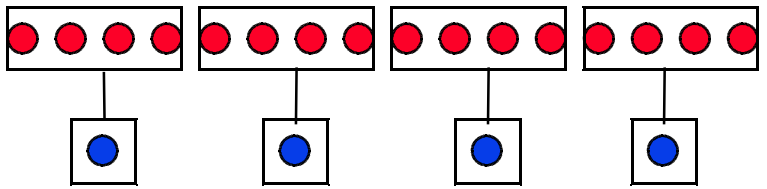
- **Semaphore:** A “control flag”
- **Critical Region:** A group of instructions that should be executed by only one process at a time
- **Mutual exclusion:** Requirement for proper implementation of a critical region

Simple Example

- Shared memory strategy:
  - small number p << n=size(A) processes
  - attached to same memory

$$\sum_{i=0}^{n-1} f(A[i])$$

- Assign n/p numbers to each process (or thread)
  - Each computes independent “private” results and partial sum.
  - Collect the p partial sums and compute a global sum.



Shared Memory "Code" for Computing a Sum

```
static int s = 0;
```

```
Thread 1

  for i = 0, n/2-1
      s = s + f(A[i])
```

```
Thread 2

  for i = n/2, n-1
      s = s + f(A[i])
```

- Problem is a race condition on variable s in the program
- A race condition or data race occurs when:
  - two processors (or two threads) access the same variable, and at least one does a write.
  - The accesses are concurrent (not synchronized) so they could happen simultaneously

Shared Memory "Code" for Computing a Sum

A= | 3 | 5 | $f(x) = x^2$

```
static int s = 0;
```

Thread 1		Thread 2	
….		…	
compute f([A[i]) and put in reg0	9	compute f([A[i]) and put in reg0	25
reg1 = s	0	reg1 = s	0
reg1 = reg1 + reg0	9	reg1 = reg1 + reg0	25
s = reg1	9	s = reg1	25
…		…	

- Assume A = [3,5], $f(x) = x^2$, and s=0 initially
- For this program to work, s should be $3^2 + 5^2 = 34$ at the end
  - but it may be 34,9, or 25

→ show gcc -S example

Improved Code for Computing a Sum

```
static int s = 0;
static lock lk;
```

```
Thread 1

  local_s1= 0
  for i = 0, n/2-1
     local_s1 = local_s1 + f(A[i])
lock(lk);
  s = s + local_s1
unlock(lk);
```

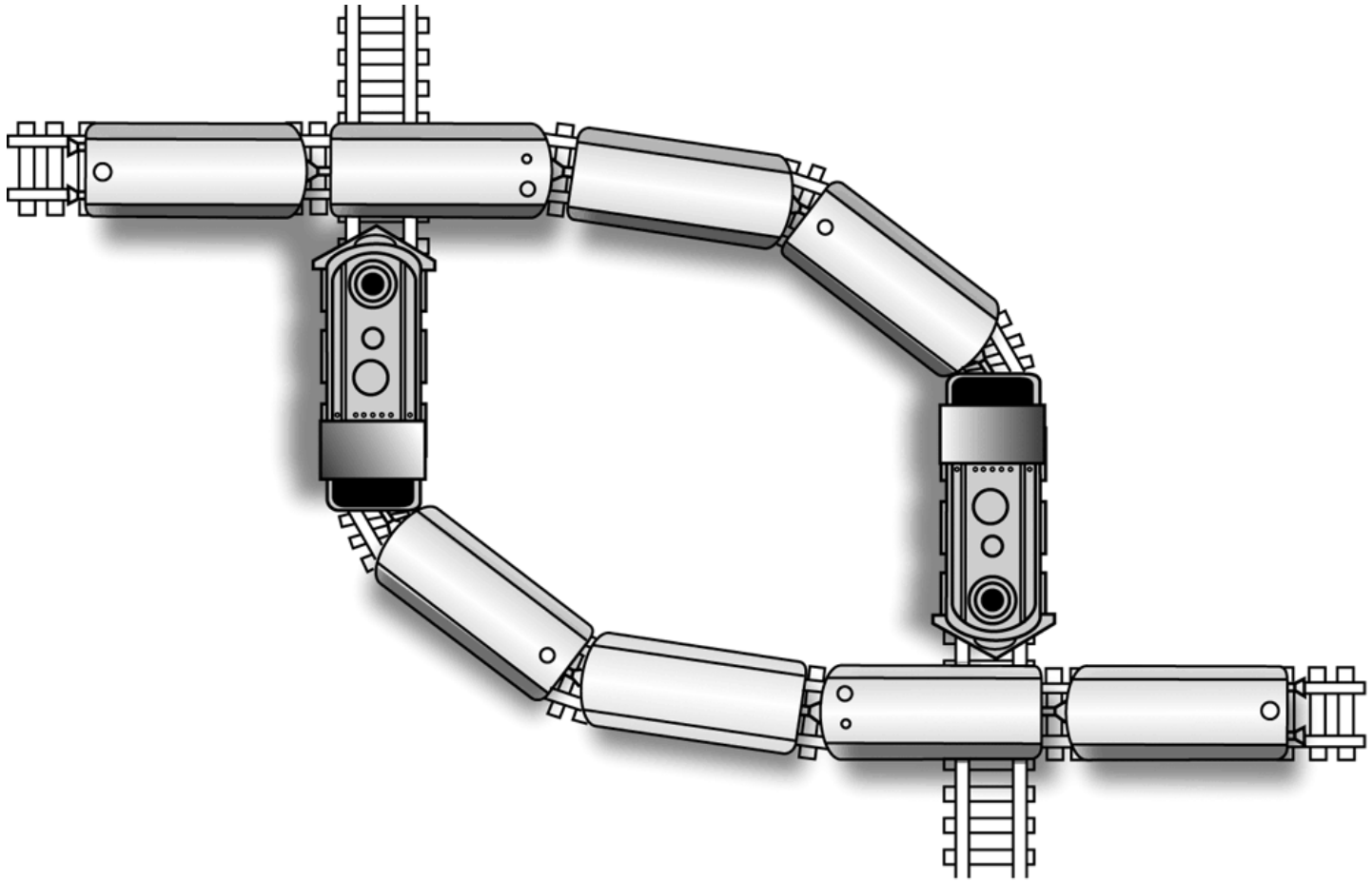
```
Thread 2

  local_s2 = 0
  for i = n/2, n-1
     local_s2= local_s2 + f(A[i])
lock(lk);
  s = s +local_s2
unlock(lk);
```

- Most computation is on private variables
  - Sharing frequency is also reduced, which might improve speed
  - But there is still a race condition on the update of shared s
  - The race condition can be fixed by adding locks (only one thread can hold a lock at a time; others wait for it)

Deadlock

- Processes block each other from continuing
- Conditions required for deadlock
  1. Competition for non-sharable resources
  2. Resources requested on a partial basis
  3. An allocated resource can not be forcibly retrieved

Figure 3.7 A deadlock resulting from competition for non-shareable railroad intersections



Dining-Philosophers Problem

The dining philosophers problem is summarized as five philosophers sitting at a table doing one of two things: eating or thinking. While eating, they are not thinking, and while thinking, they are not eating. The five philosophers sit at a circular table with a large bowl of spaghetti in the center. A fork is placed in between each pair of adjacent philosophers, and as such, each philosopher has one fork to his left and one fork to his right. As spaghetti is difficult to serve and eat with a single fork, it is assumed that a philosopher must eat with two forks. Each philosopher can only use the forks on his immediate left and immediate right. (Wikipedia)

→ show Philosopher Applet

Dining-Philosopher Problem

- If philosophers never speak to each other, this creates a dangerous possibility of deadlock when every philosopher holds a left fork and waits perpetually for a right fork (or vice versa).

- Starvation might also occur independently of deadlock if a philosopher is unable to acquire both forks because of a timing problem.

Solutions?

How to avoid deadlocks?

How to avoid starvation?

3.5 Security

- Attacks from outside
  - Problems
    - Insecure passwords
    - Sniffing software
  - Counter measures
    - Auditing software

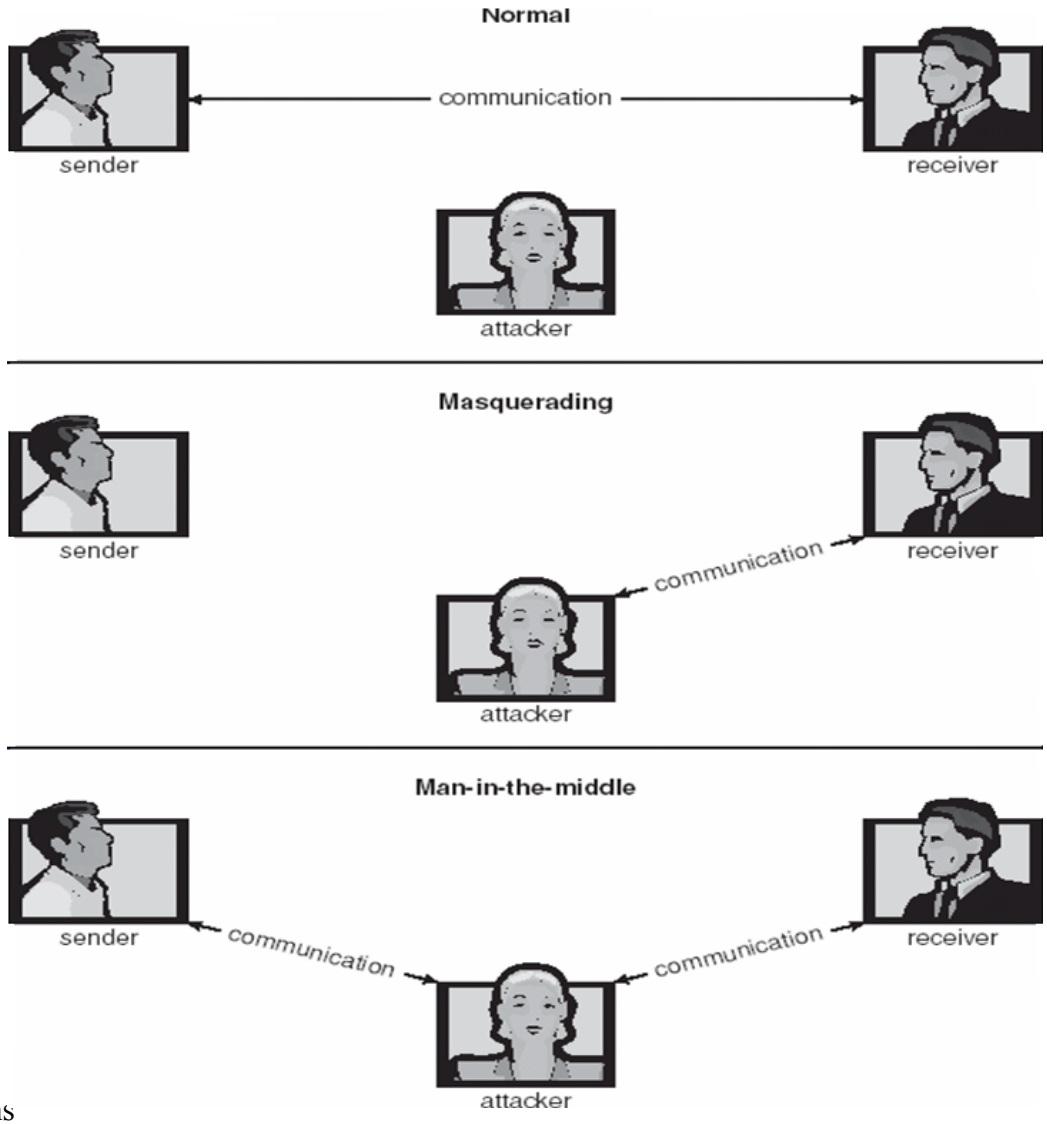
Security (continued)

- Attacks from within
  - Problem: Unruly processes
  - Counter measures: Control process activities via privileged modes and privileged instructions

Security Violations

- Categories
  - **Breach of confidentiality** (unauthorized reading of data)
  - **Breach of integrity** (unauthorized modification of data)
  - **Breach of availability** (unauthorized destruction of data)
  - **Theft of service** (unauthorized use of resources)
  - **Denial of service** (preventing legitimate use of a system)
- Methods
  - **Masquerading (breach authentication)**
  - **Replay attack**
    - **Message modification**
  - **Man-in-the-middle attack**
  - **Session hijacking**

Standard Security Attacks



Chapter 3: Operating Systems

- 3.1 The History of Operating Systems
- 3.2 Operating System Architecture
- 3.3 Coordinating the Machine's Activities
- 3.4 Handling Competition Among Processes
- 3.5 Security