# Operating System

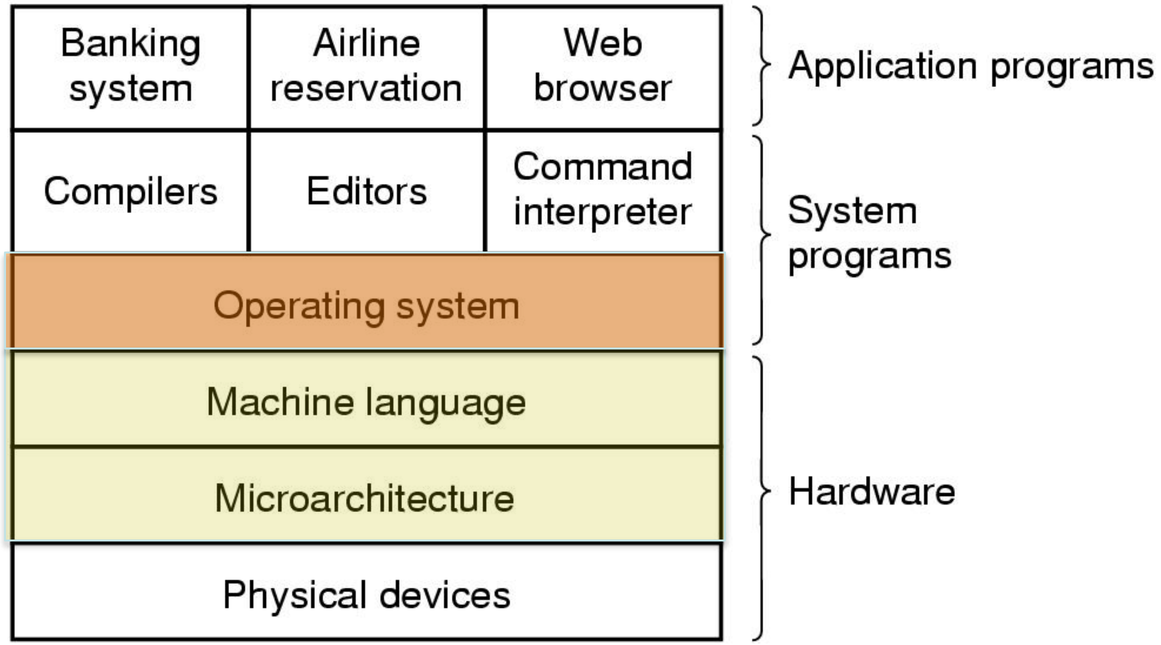
Virendra Singh
Computer Architecture and Dependable Systems Lab
Department of Electrical Engineering
Indian Institute of Technology Bombay
http://www.ee.iitb.ac.in/~viren/
E-mail: viren@ee.iitb.ac.in

*EE-309: Microprocessors*

Lecture 42 (03 Nov 2015)

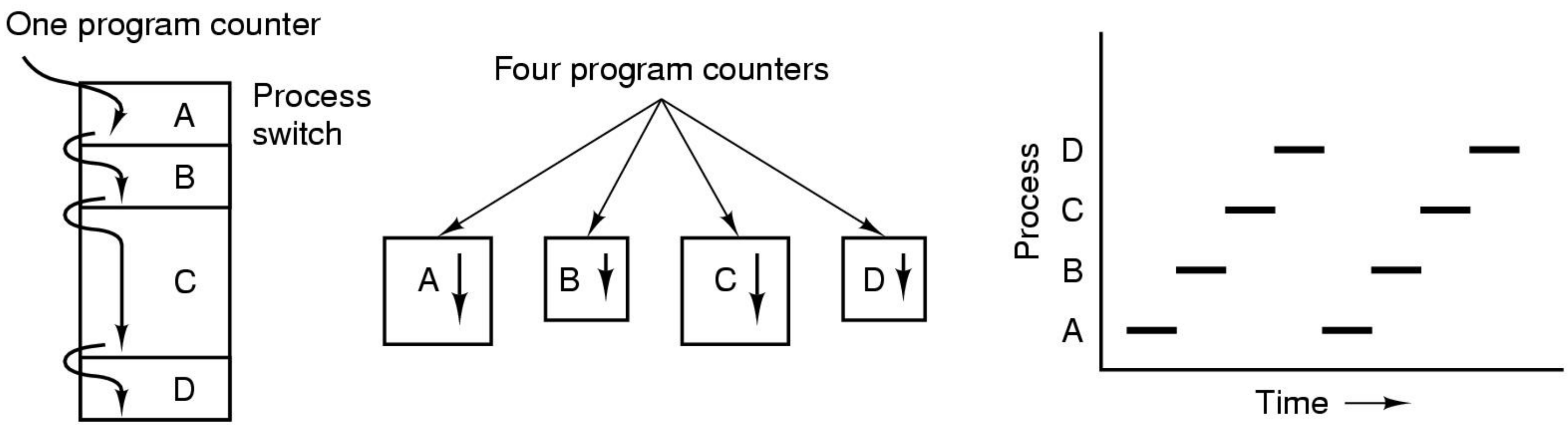
CADSL

# Introduction




- A computer system consists of
  - Hardware
  - System programs
  - Application programs

# Processes: The Process Model




(a) (b) (c)

- Multiprogramming of four programs
- Conceptual model of 4 independent, sequential processes
- Only one program active at any instant

# Process Creation

Principal events that cause process creation

1. System initialization
2. Execution of a process creation system
3. User request to create a new process
4. Initiation of a batch job

# Process Termination

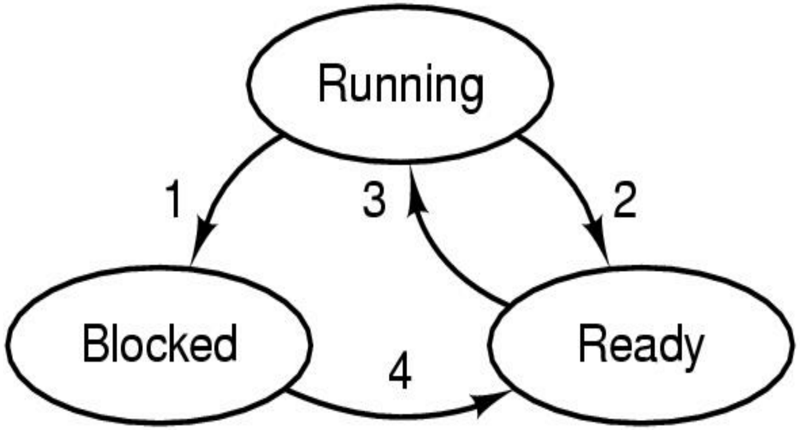
Conditions which terminate processes

1. Normal exit (voluntary)
2. Error exit (voluntary)
3. Fatal error (involuntary)
4. Killed by another process (involuntary)

# Process Hierarchies

- Parent creates a child process, child processes can create its own process
- Forms a hierarchy
  - UNIX calls this a "process group"
- Windows has no concept of process hierarchy
  - all processes are created equal

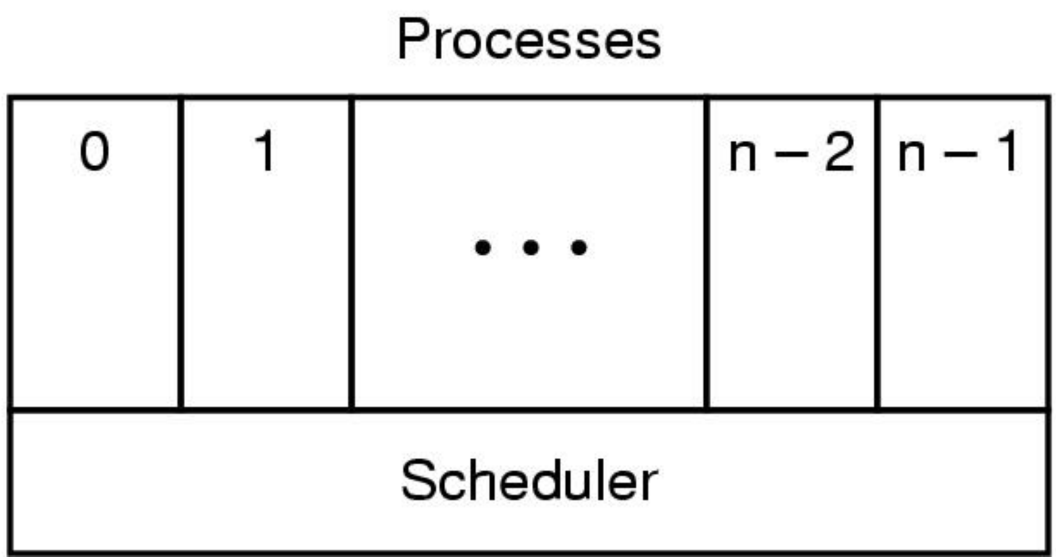
# Process States




1. Process blocks for input
2. Scheduler picks another process
3. Scheduler picks this process
4. Input becomes available

- Possible process states
  - Running
  - Blocked
  - Ready
- Transitions between states shown

# Process States




- Lowest layer of process-structured OS
  - ➢ Handles interrupts, scheduling
- Above that layer are sequential processes

# Implementation of Processes

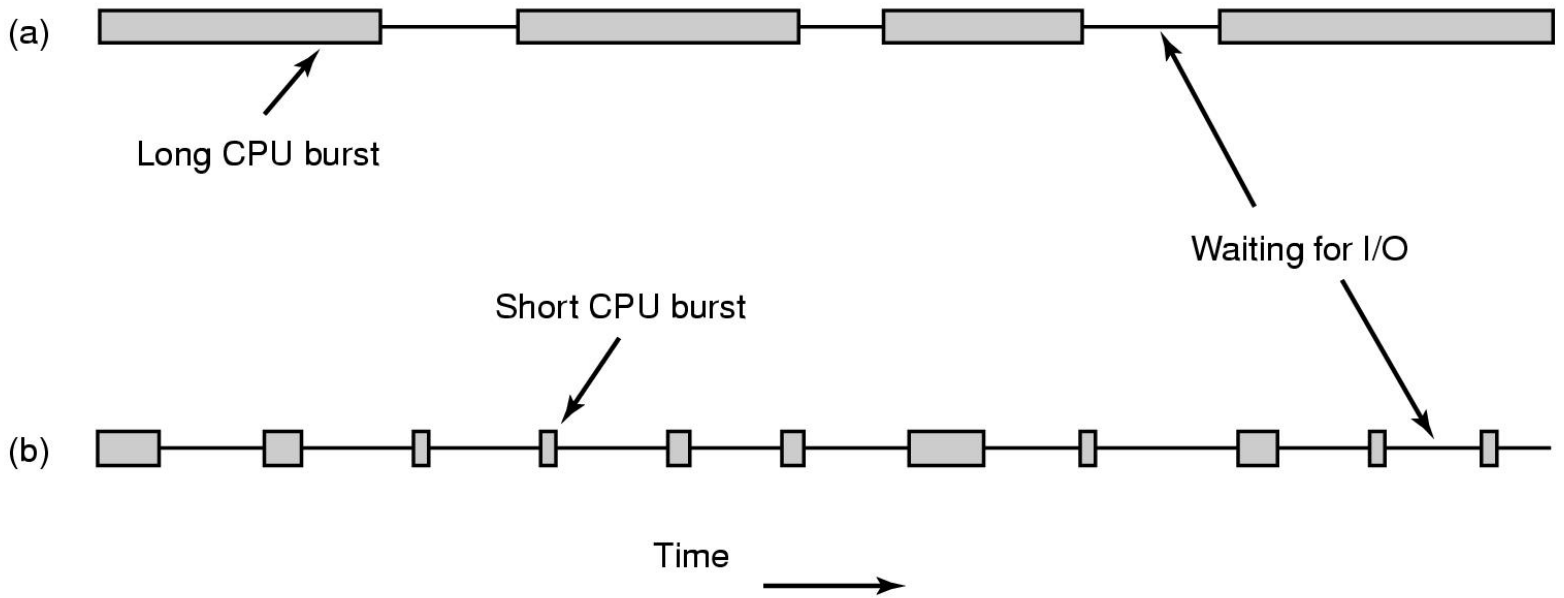
| Process management | Memory management | File management |
|---|---|---|
| Registers<br>Program counter<br>Program status word<br>Stack pointer<br>Process state<br>Priority<br>Scheduling parameters<br>Process ID<br>Parent process<br>Process group<br>Signals<br>Time when process started<br>CPU time used<br>Children's CPU time<br>Time of next alarm | Pointer to text segment<br>Pointer to data segment<br>Pointer to stack segment | Root directory<br>Working directory<br>File descriptors<br>User ID<br>Group ID |

Fields of a process table entry

# Process Scheduling

- Which process to run?
- Efficient use of CPU
  - Switching is expensive
- Switch to kernel mode
- Save current process state
  - Registers
  - Process table
  - Memory map

# Process Scheduling




- Bursts of CPU usage alternate with periods of I/O wait
  - a CPU-bound process
  - an I/O bound process

# Process Scheduling

**All systems**

Fairness - giving each process a fair share of the CPU
Policy enforcement - seeing that stated policy is carried out
Balance - keeping all parts of the system busy

**Batch systems**

Throughput - maximize jobs per hour
Turnaround time - minimize time between submission and termination
CPU utilization - keep the CPU busy all the time

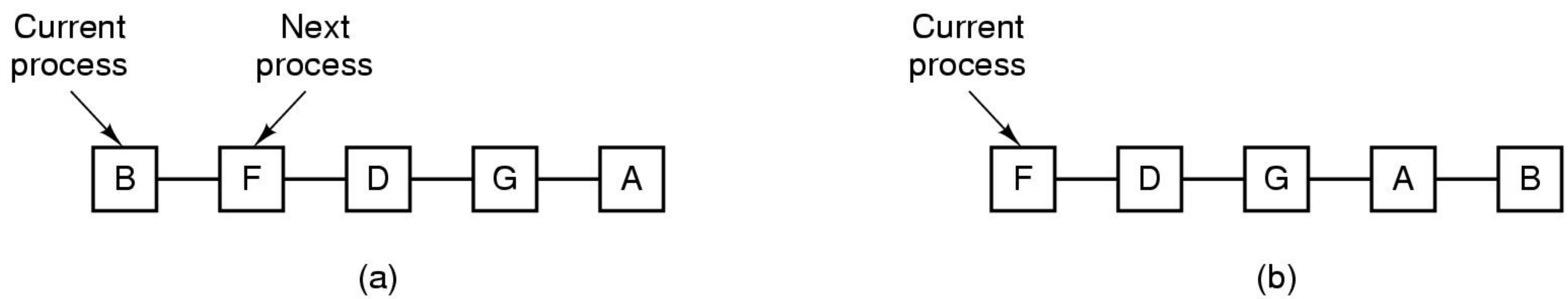
**Interactive systems**

Response time - respond to requests quickly
Proportionality - meet users' expectations

**Real-time systems**

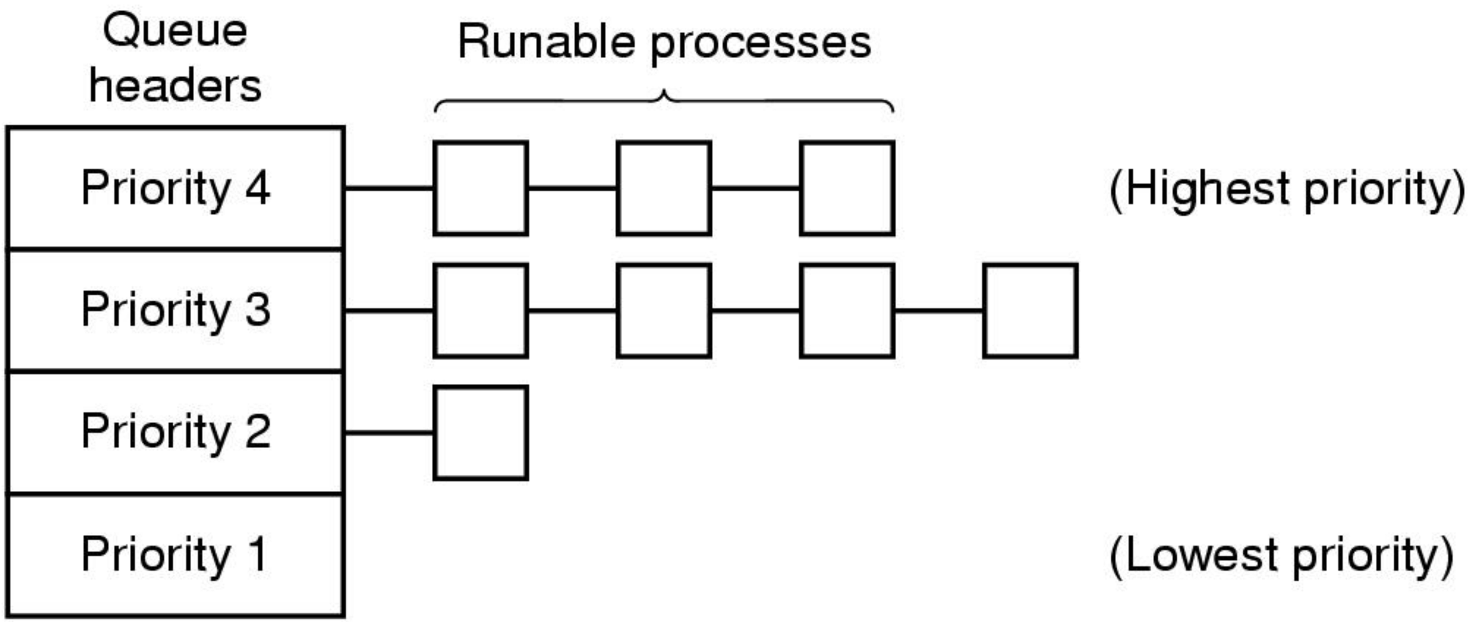
Meeting deadlines - avoid losing data
Predictability - avoid quality degradation in multimedia systems

# Scheduling in Interactive Systems




- Round Robin Scheduling
  - list of runnable processes
  - list of runnable processes after B uses up its quantum

# Scheduling in Interactive Systems




A scheduling algorithm with four priority classes

# Scheduling in Real-Time Systems

Schedulable real-time system

- Given
  - *m* periodic events
  - event *i* occurs within period $P_i$ and requires $C_i$ seconds
- Then the load can only be handled if

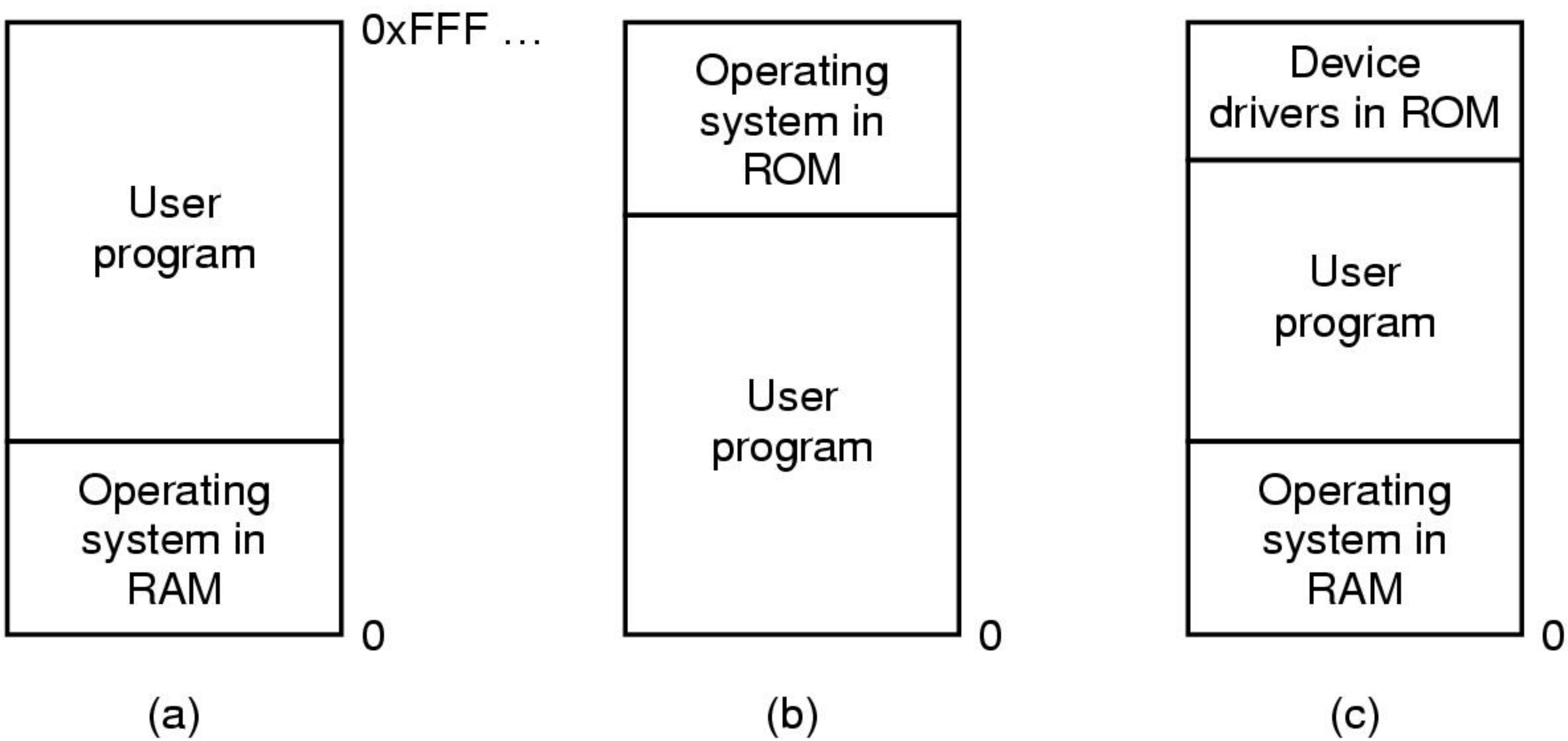
$$\sum_{i=1}^{m} \frac{C_i}{P_i} \leq 1$$

# Policy versus Mechanism

- Separate what is allowed to be done with how it is done
  - a process knows which of its children threads are important and need priority
- Scheduling algorithm parameterized
  - mechanism in the kernel
- Parameters filled in by user processes
  - policy set by user process

# Memory Management

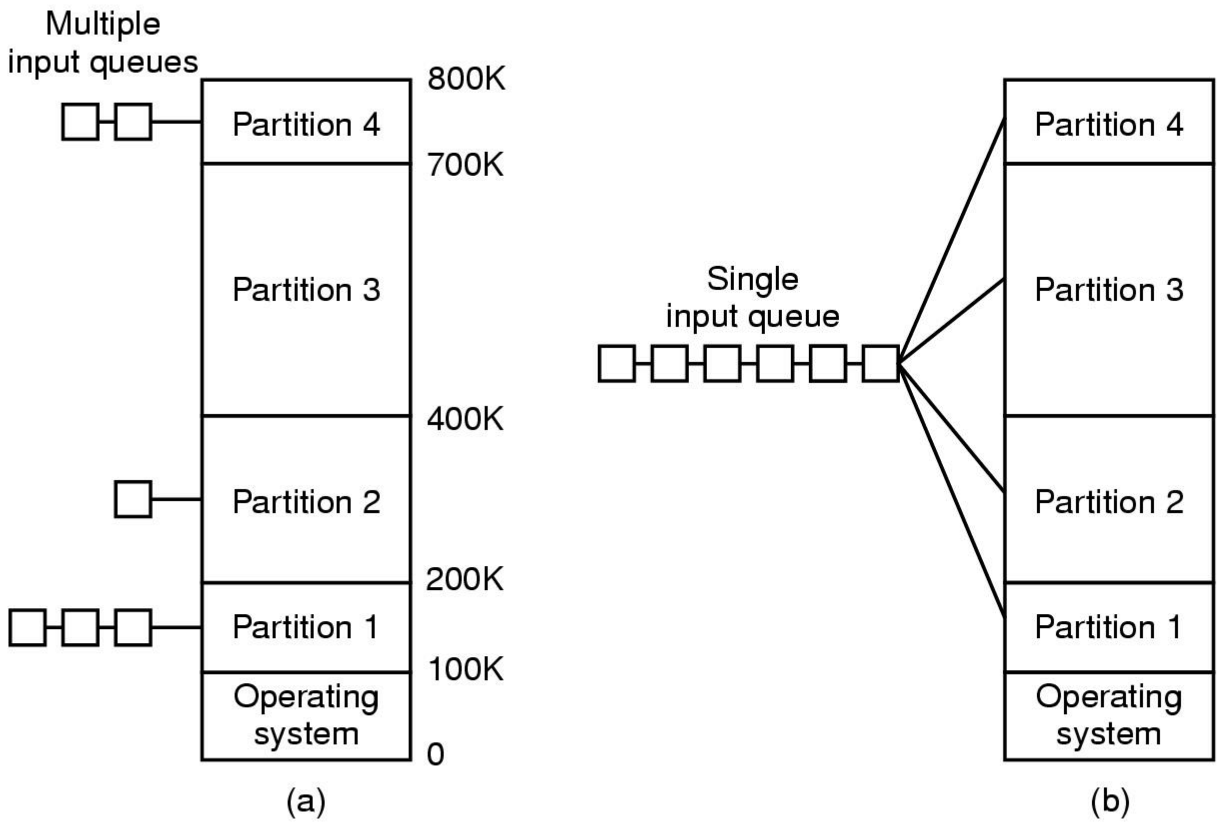
# Basic Memory Management
## Monoprogramming without Swapping or Paging




Three simple ways of organizing memory

- - an operating system with one user process

# Multiprogramming with Fixed Partitions



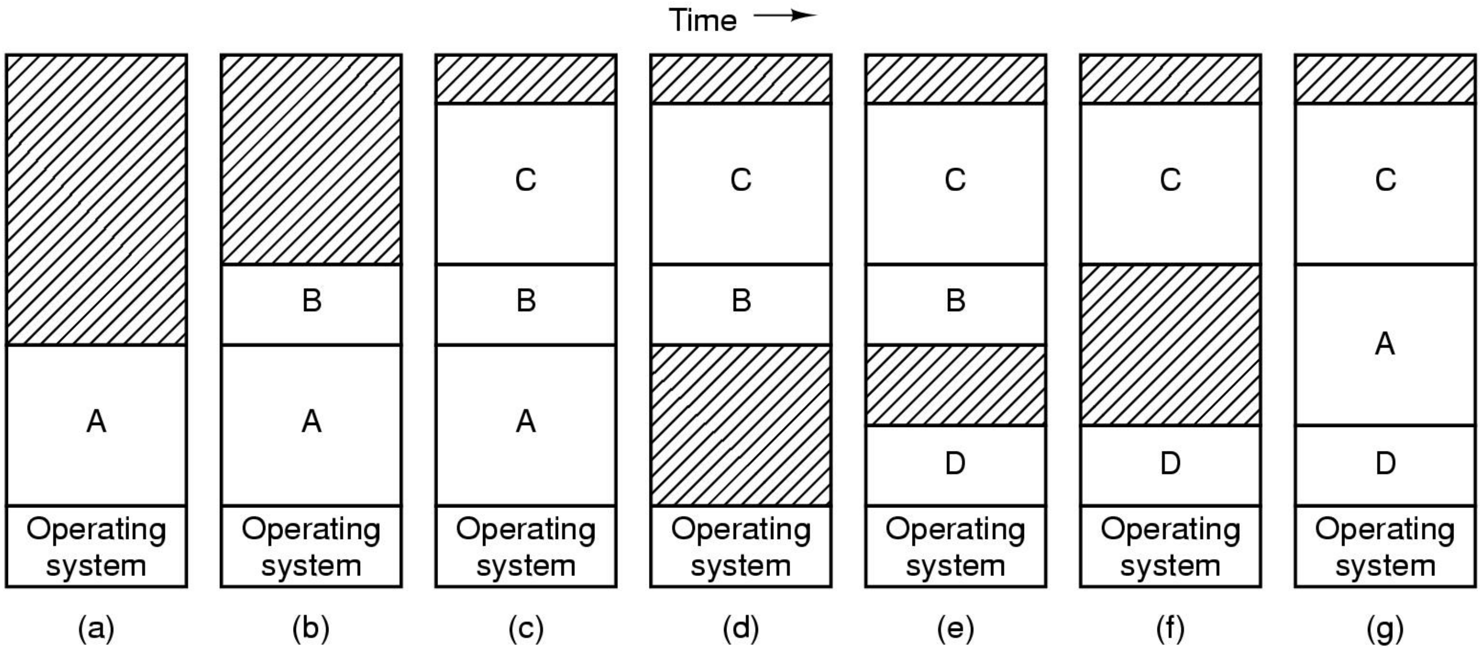

- Fixed memory partitions
  - ➢ separate input queues for each partition
  - ➢ single input queue

# Relocation and Protection

- Cannot be sure where program will be loaded in memory
  - address locations of variables,  code routines cannot be absolute
  - must keep a program out of other processes' partitions

- Use base and limit values
  - address locations added to base value to map to physical addr
  - address locations larger than limit value is an error

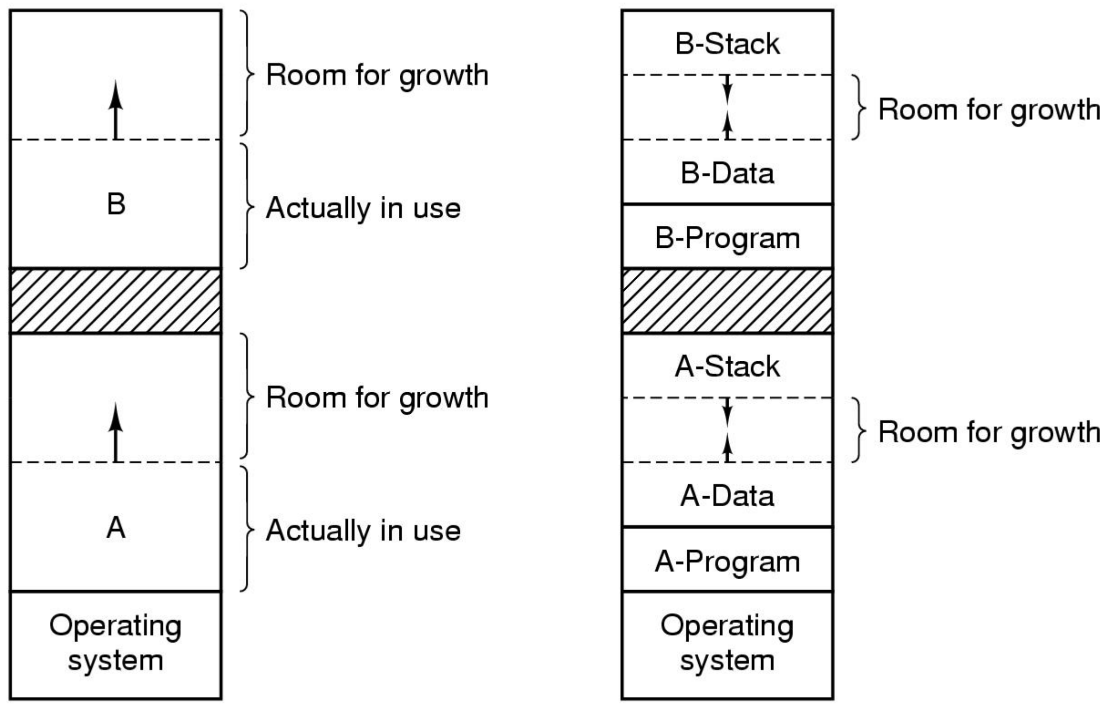
# Swapping




Memory allocation changes as

- processes come into memory
- leave memory

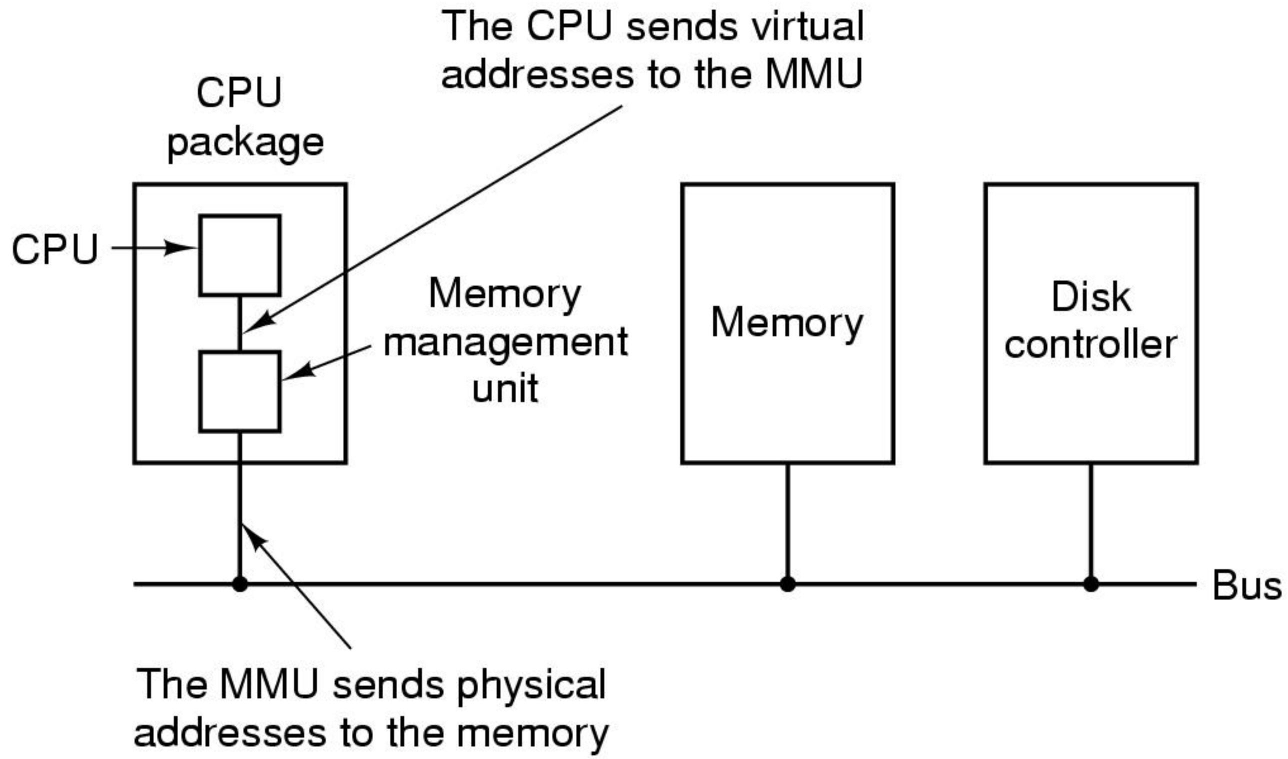
# Swapping




(a) (b)

- Allocating space for growing data segment
- Allocating space for growing stack & data segment

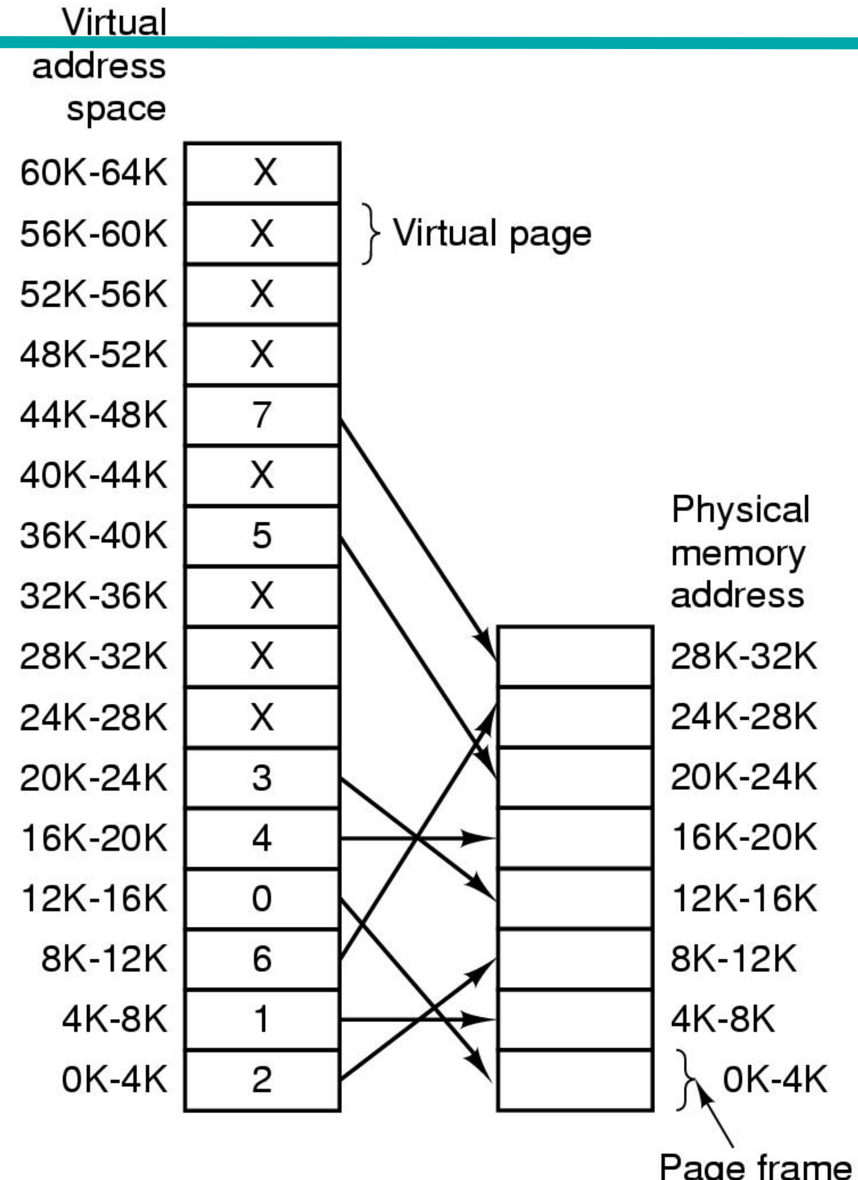
# Virtual Memory: Paging



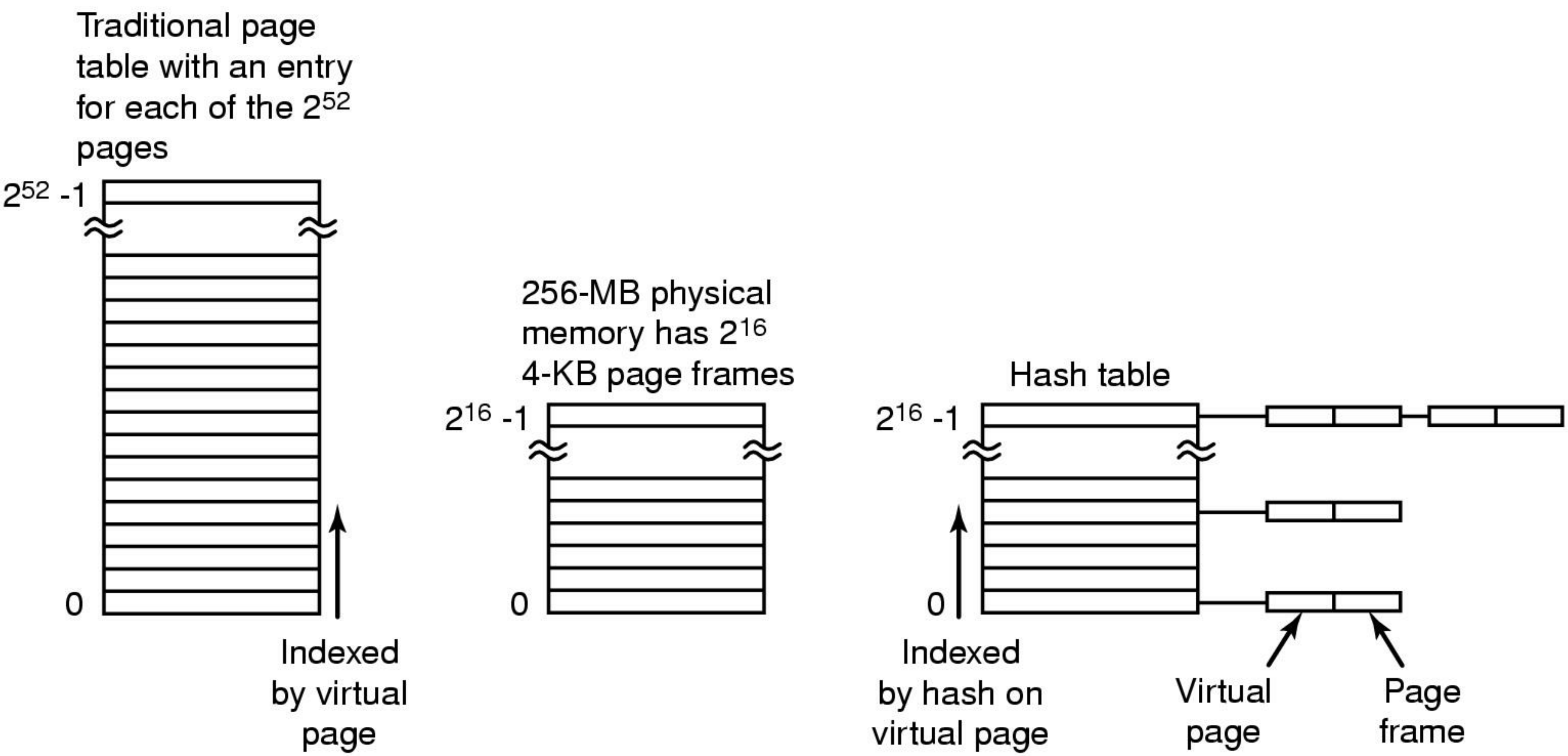

The position and function of the MMU

# Paging

The relation between virtual addresses and physical memory addresses given by page table



# Inverted Page Tables




Comparison of a traditional page table with an inverted page table

# Page Replacement Algorithms

- Page fault forces choice
  - which page must be removed
  - make room for incoming page
- Modified page must first be saved
  - unmodified just overwritten
- Better not to choose an often used page
  - will probably need to be brought back in soon

# Optimal Page Replacement Algorithm

- Replace page needed at the farthest point in future
  - Optimal but unrealizable

- Estimate by …
  - logging page use on previous runs of  process
  - although this is impractical

# Not Recently Used Page Replacement Algorithm

- Each page has Reference bit, Modified bit
  – bits are set when page is referenced, modified
- Pages are classified
  1. not referenced, not modified
  2. not referenced, modified
  3. referenced, not modified
  4. referenced, modified
- NRU removes page at random
  – from lowest numbered non empty class

# FIFO Page Replacement Algorithm

- Maintain a linked list of all pages
  - in order they came into memory
- Page at beginning of list replaced
- Disadvantage
  - page in memory the longest may be often used

# Least Recently Used (LRU)

- Assume pages used recently will used again soon
  - throw out page that has been unused for longest time
- Must keep a linked list of pages
  - most recently used at front, least at rear
  - update this list <u>every memory reference</u> !!
- Alternatively keep counter in each page table entry
  - choose page with lowest value counter
  - periodically zero the counter

# Thank You