

**William Stallings**  
**Computer Organization**  
**and Architecture**  
**6<sup>th</sup> Edition**

---

**Chapter 8**  
**Operating System Support**

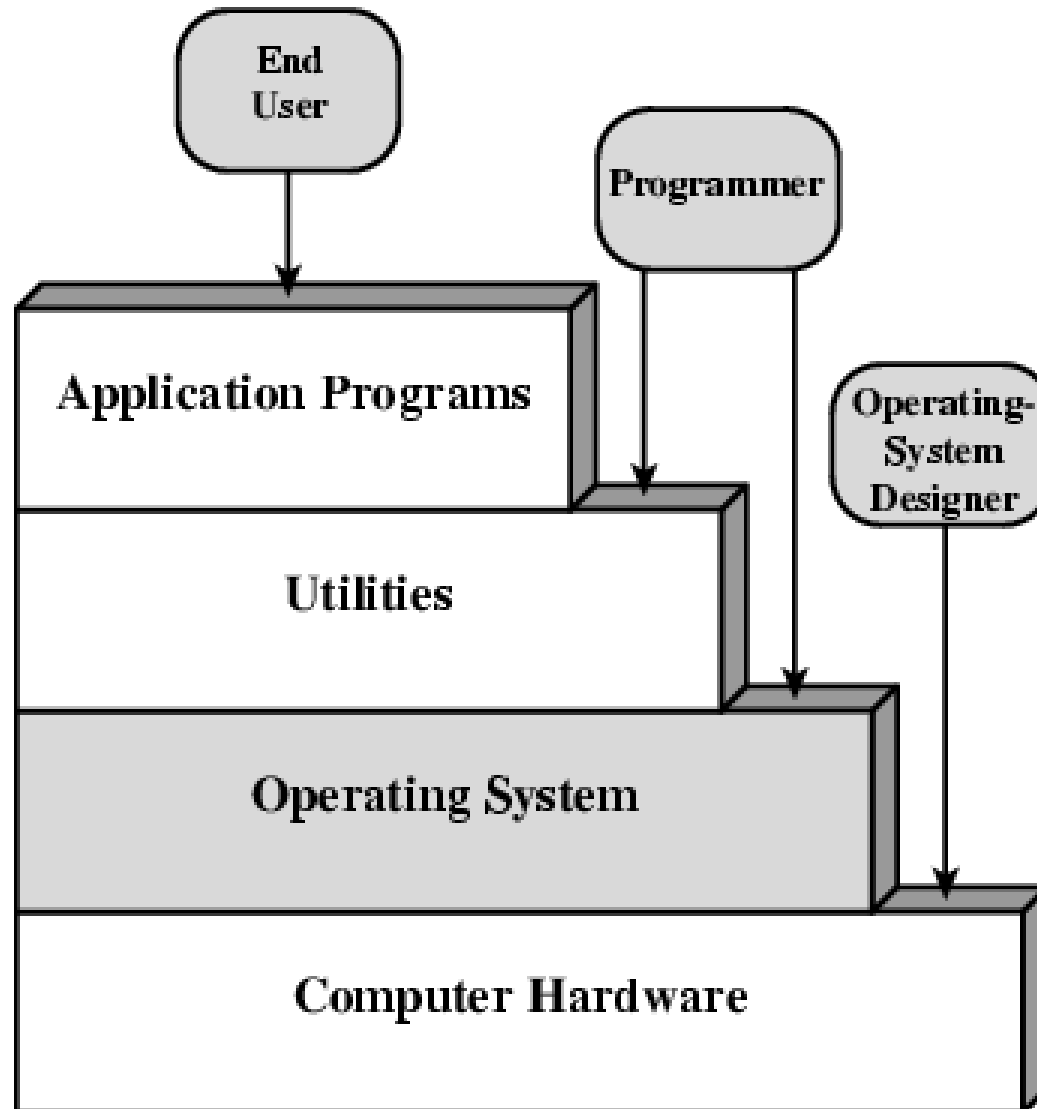
**Dr. Muazzam A. Khan**

# Objectives and Functions

---

- Convenience
  - Making the computer easier to use
- Efficiency
  - Allowing better use of computer resources

# Layers and Views of a Computer System



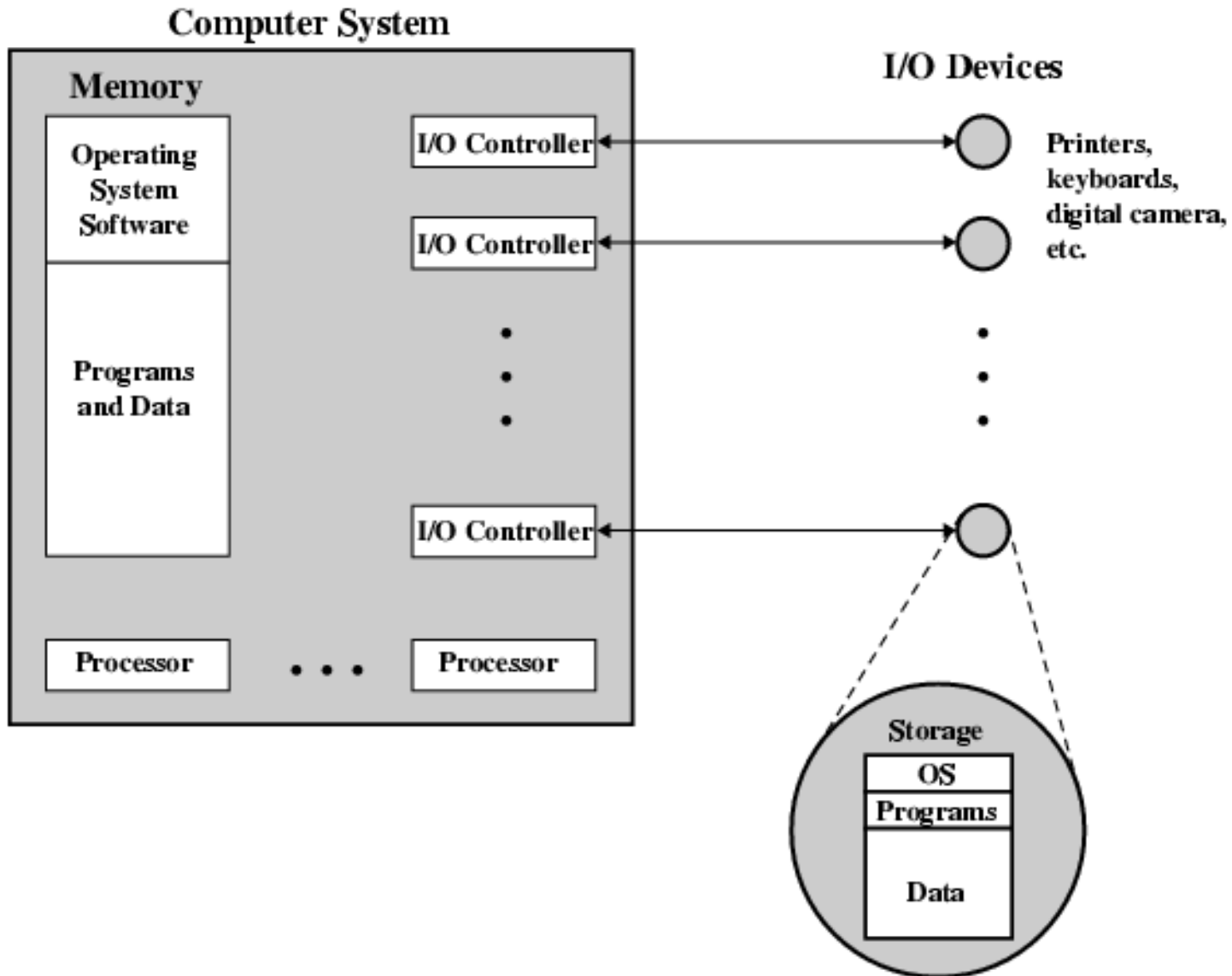
# Operating System Services

---

- Program creation
- Program execution
- Access to I/O devices
- Controlled access to files
- System access
- Error detection and response
- Accounting

# O/S as a Resource Manager

---



# **Types of Operating System**

---

- Interactive
- Batch
- Single program (Uni-programming)
- Multi-programming (Multi-tasking)

# Early Systems

---

- Late 1940s to mid 1950s
- No Operating System
- Programs interact directly with hardware
- Two main problems:
  - Scheduling
  - Setup time

# Simple Batch Systems

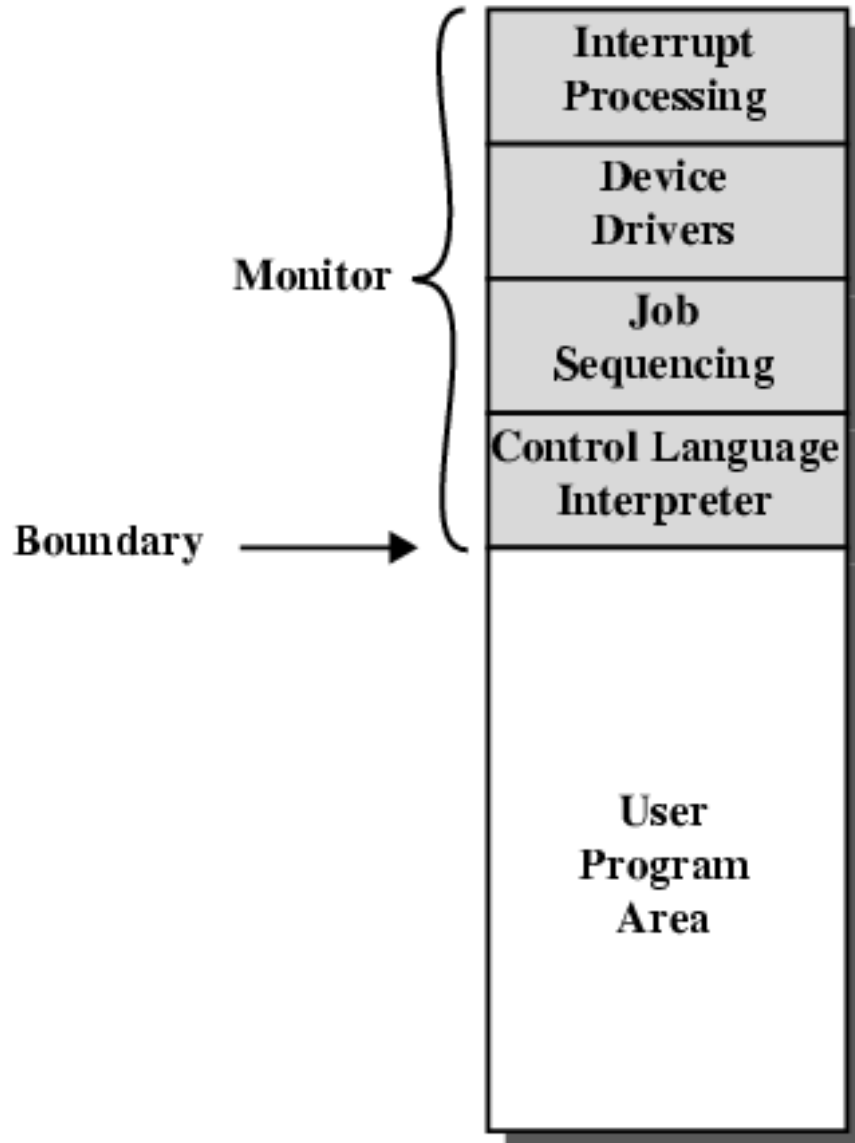
---

- Resident Monitor program
- Users submit jobs to operator
- Operator batches jobs
- Monitor controls sequence of events to process batch
- When one job is finished, control returns to Monitor which reads next job
- Monitor handles scheduling



# Memory Layout for Resident Monitor

---



# Job Control Language

---

- Instructions to Monitor
- Usually denoted by \$
- e.g.
  - \$JOB
  - \$FTN
  - ...      Some Fortran instructions
  - \$LOAD
  - \$RUN
  - ...      Some data
  - \$END

# Other Desirable Hardware Features

---

- Memory protection
  - To protect the Monitor
- Timer
  - To prevent a job monopolizing the system
- Privileged instructions
  - Only executed by Monitor
  - e.g. I/O
- Interrupts
  - Allows for relinquishing and regaining control

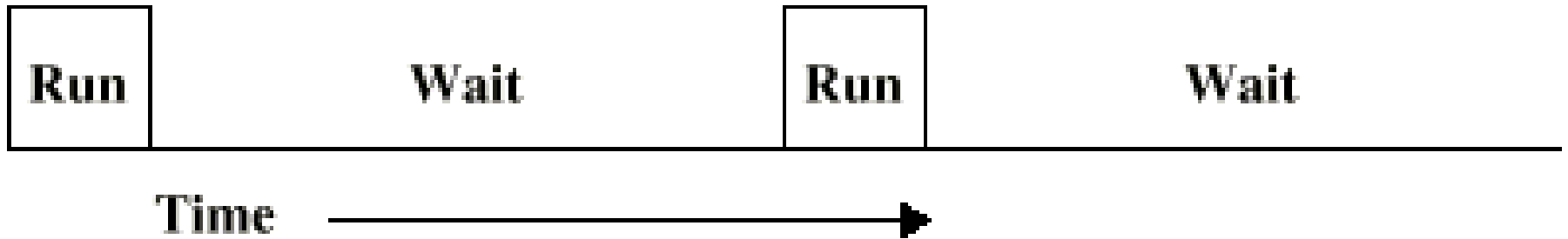
# **Multi-programmed Batch Systems**

---

- I/O devices very slow
- When one program is waiting for I/O, another can use the CPU

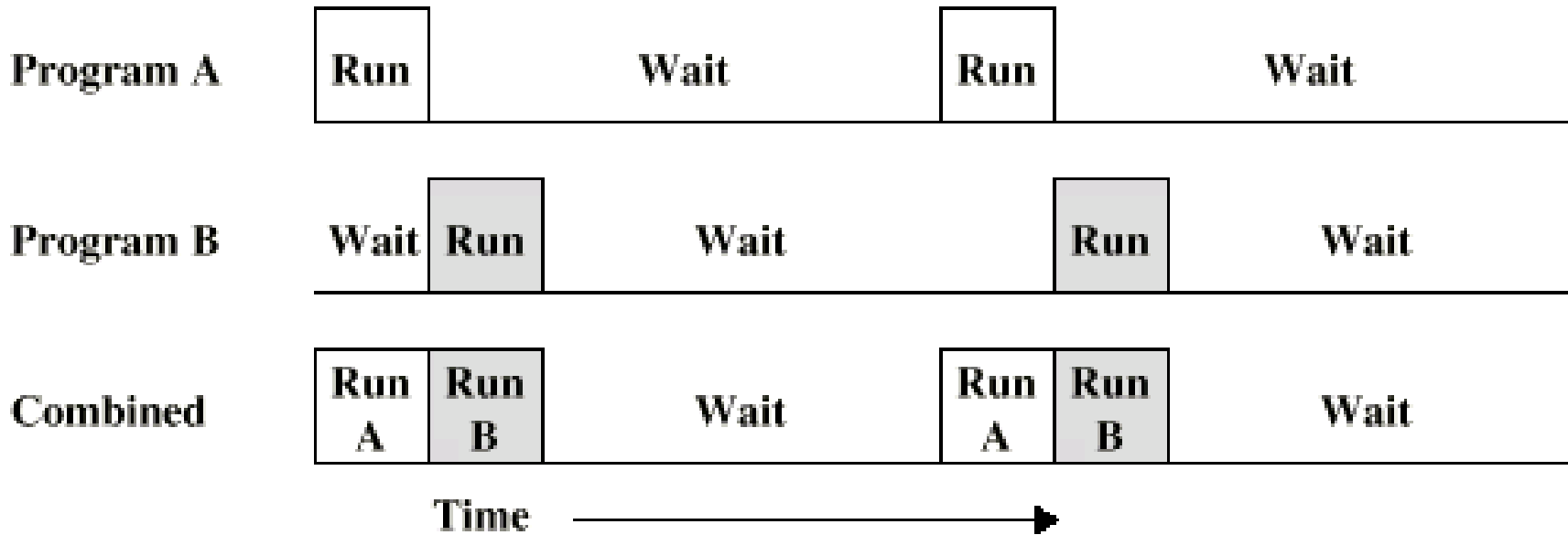
# Single Program

---



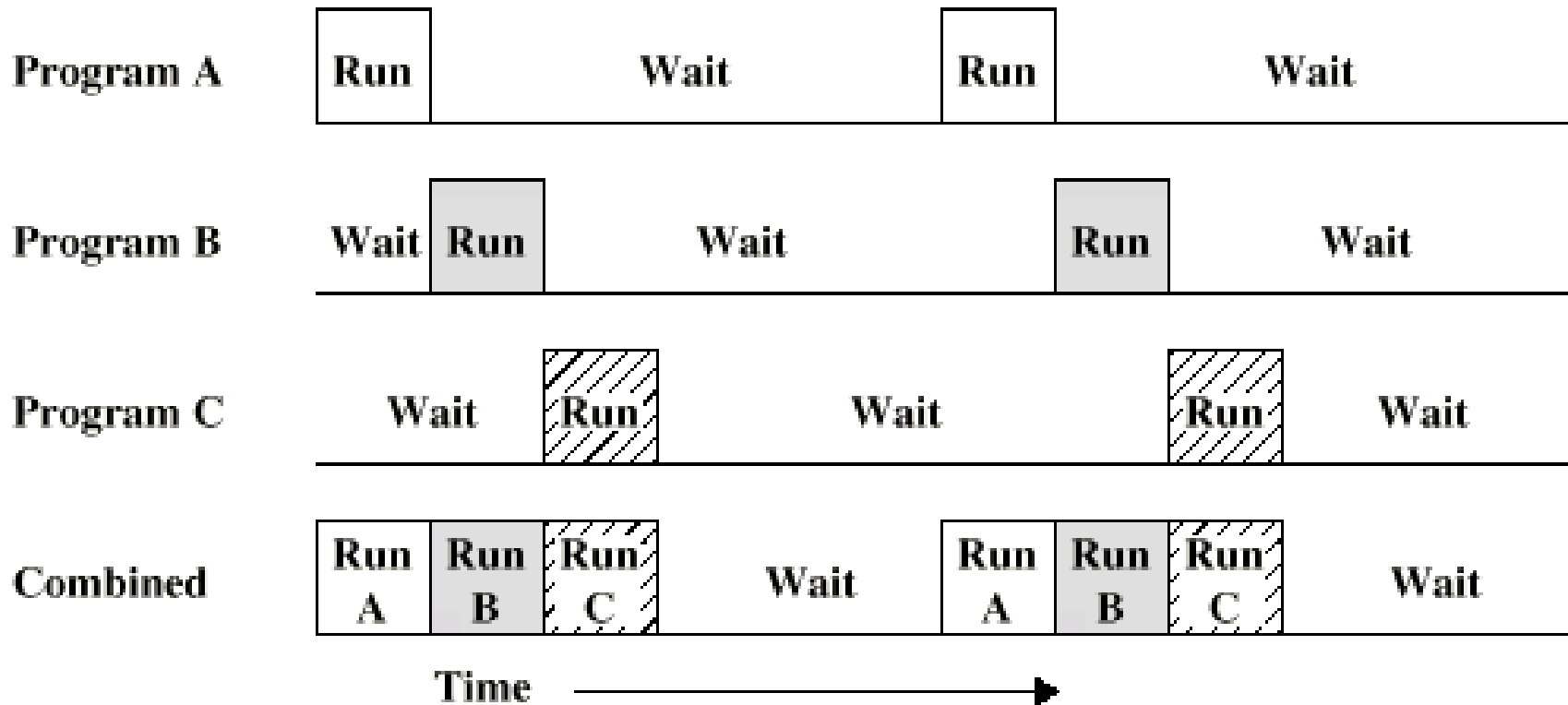
# Multi-Programming with Two Programs

---



# Multi-Programming with Three Programs

---



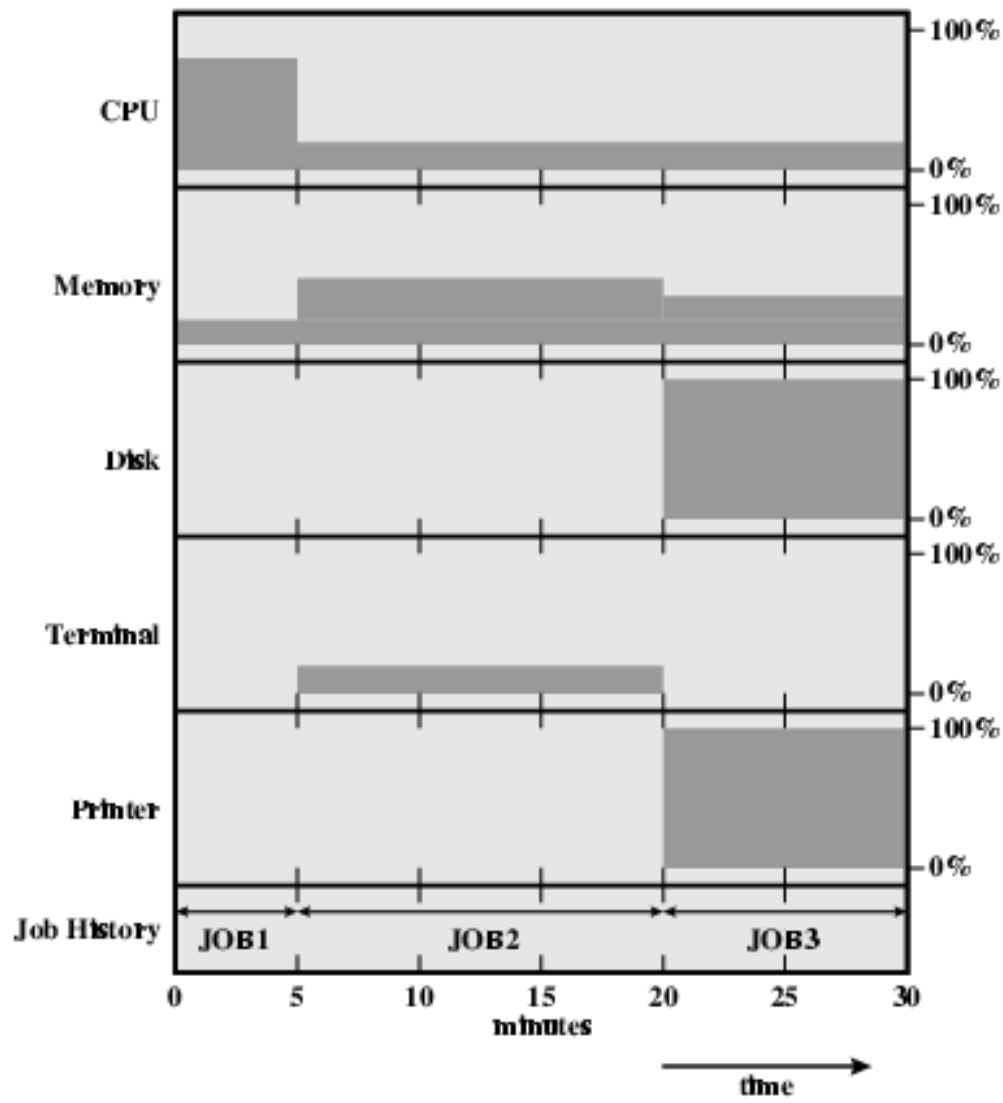
# Sample Program Execution Attributes

---

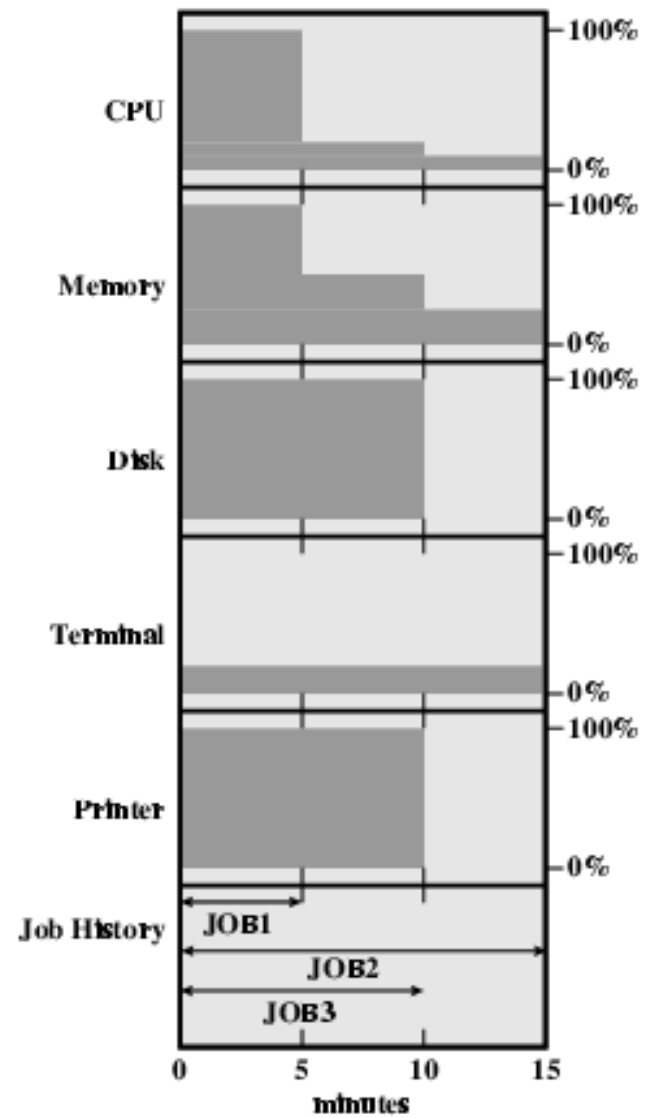
	JOB1	JOB2	JOB3
<b>Type of job</b>	Heavy compute	Heavy I/O	Heavy I/O
<b>Duration</b>	5 min	15 min	10 min
<b>Memory required</b>	50 K	100 K	80 K
<b>Need disk?</b>	No	No	Yes
<b>Need terminal?</b>	No	Yes	No
<b>Need printer?</b>	No	No	Yes



# Utilization



(a) Uniprogramming



(b) Multiprogramming

# Effects of Multiprogramming on Resource Utilization

---

	Uniprogramming	Multiprogramming
Processor use	22%	43%
Memory use	33%	67%
Disk use	33%	67%
Printer use	33%	67%
Elapsed time	30 min	15 min
Throughput rate	6 jobs/hr	12 jobs/hr
Mean response time	18 min	10 min

# Time Sharing Systems

---

- Allow users to interact directly with the computer
  - i.e. Interactive
- Multi-programming allows a number of users to interact with the computer

# Scheduling

---

- Key to multi-programming
- Long term
- Medium term
- Short term
- I/O

# Long Term Scheduling

---

- Determines which programs are submitted for processing
- i.e. controls the degree of multi-programming
- Once submitted, a job becomes a process for the short term scheduler
- (or it becomes a swapped out job for the medium term scheduler)

# Medium Term Scheduling

---

- Part of the swapping function (more later...)
- Usually based on the need to manage multi-programming
- If no virtual memory, memory management is also an issue

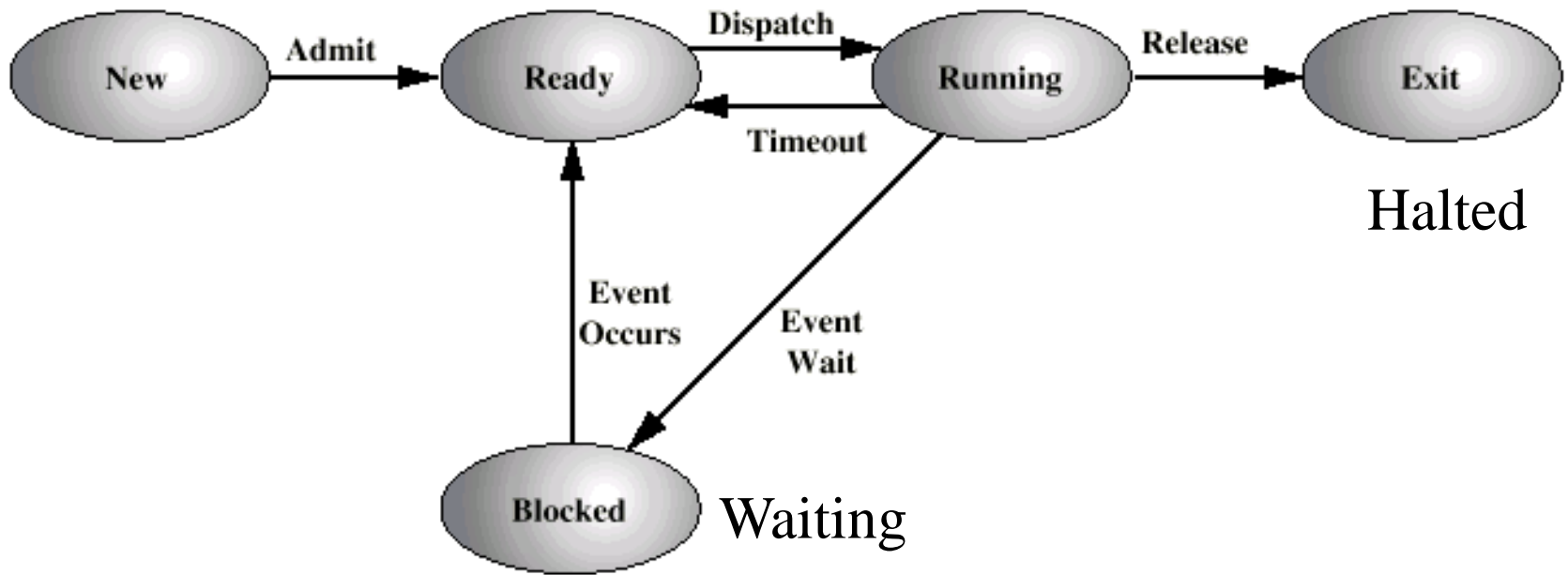
# Short Term Scheduler

---

- Dispatcher
- Fine grained decisions of which job to execute next
- i.e. which job actually gets to use the processor in the next time slot

# Five-State Process Model

---





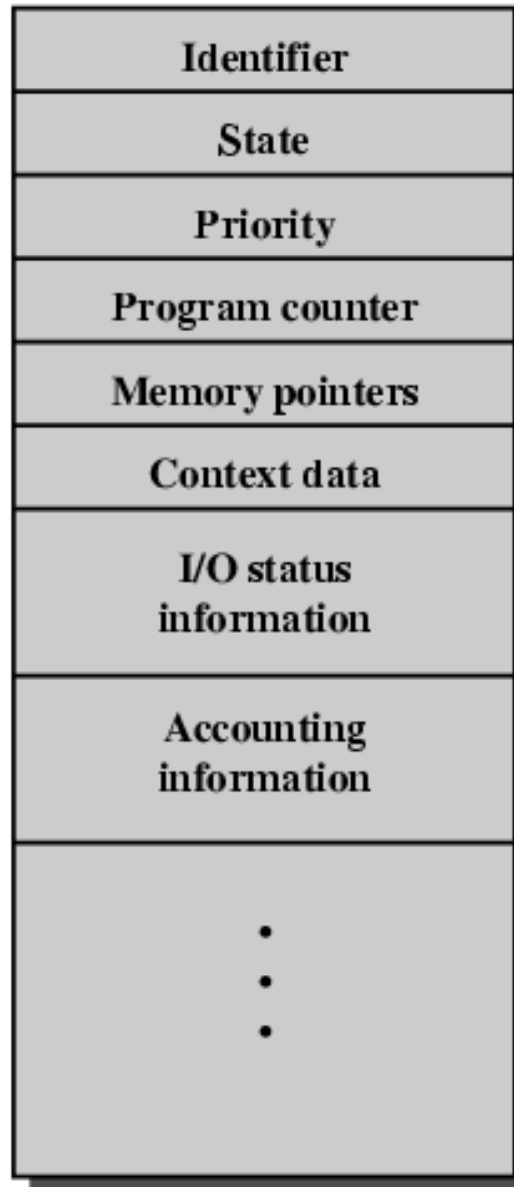
# Process Control Block

---

- Identifier
- State
- Priority
- Program counter
- Memory pointers
- Context data
- I/O status
- Accounting information

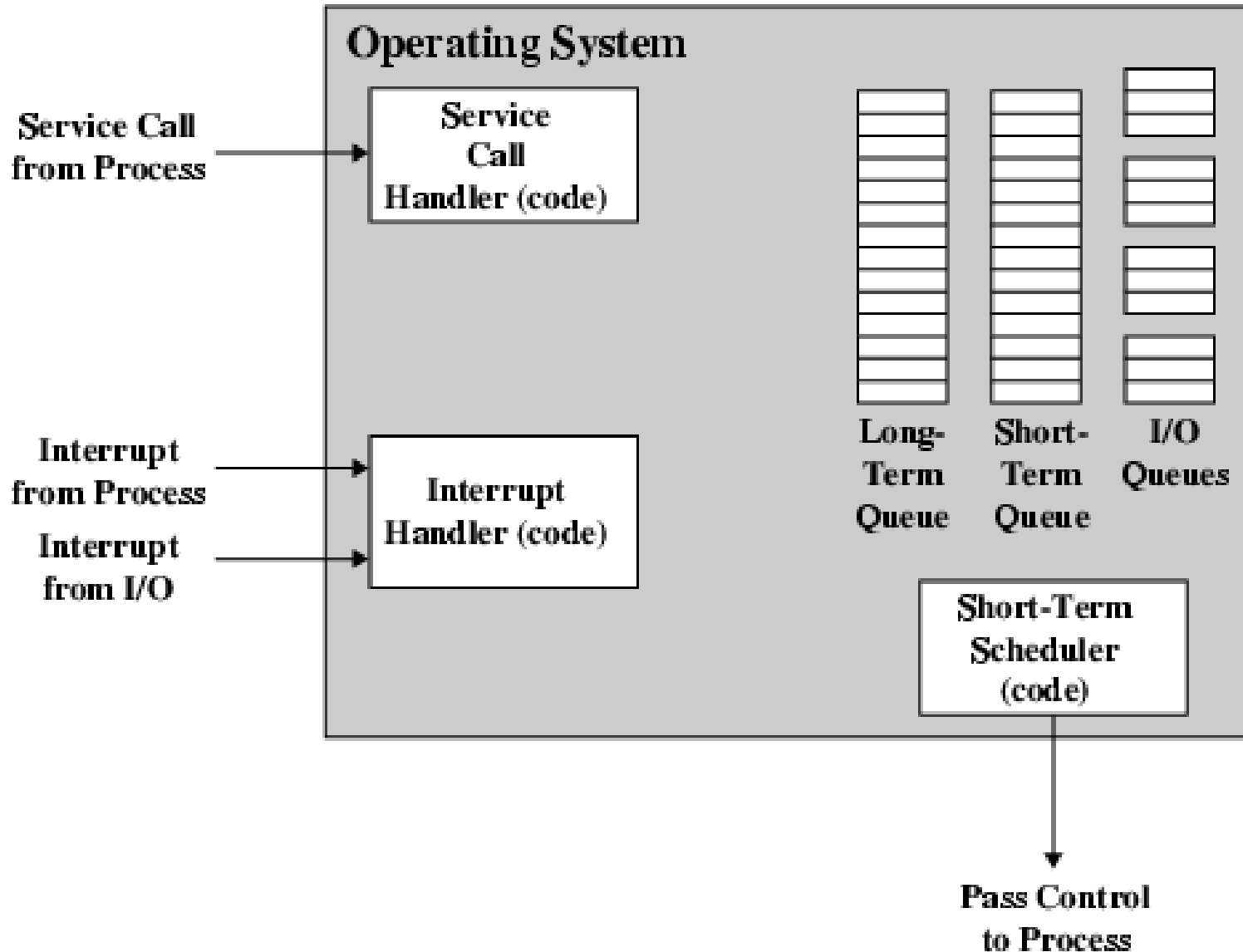
# PCB Diagram

---



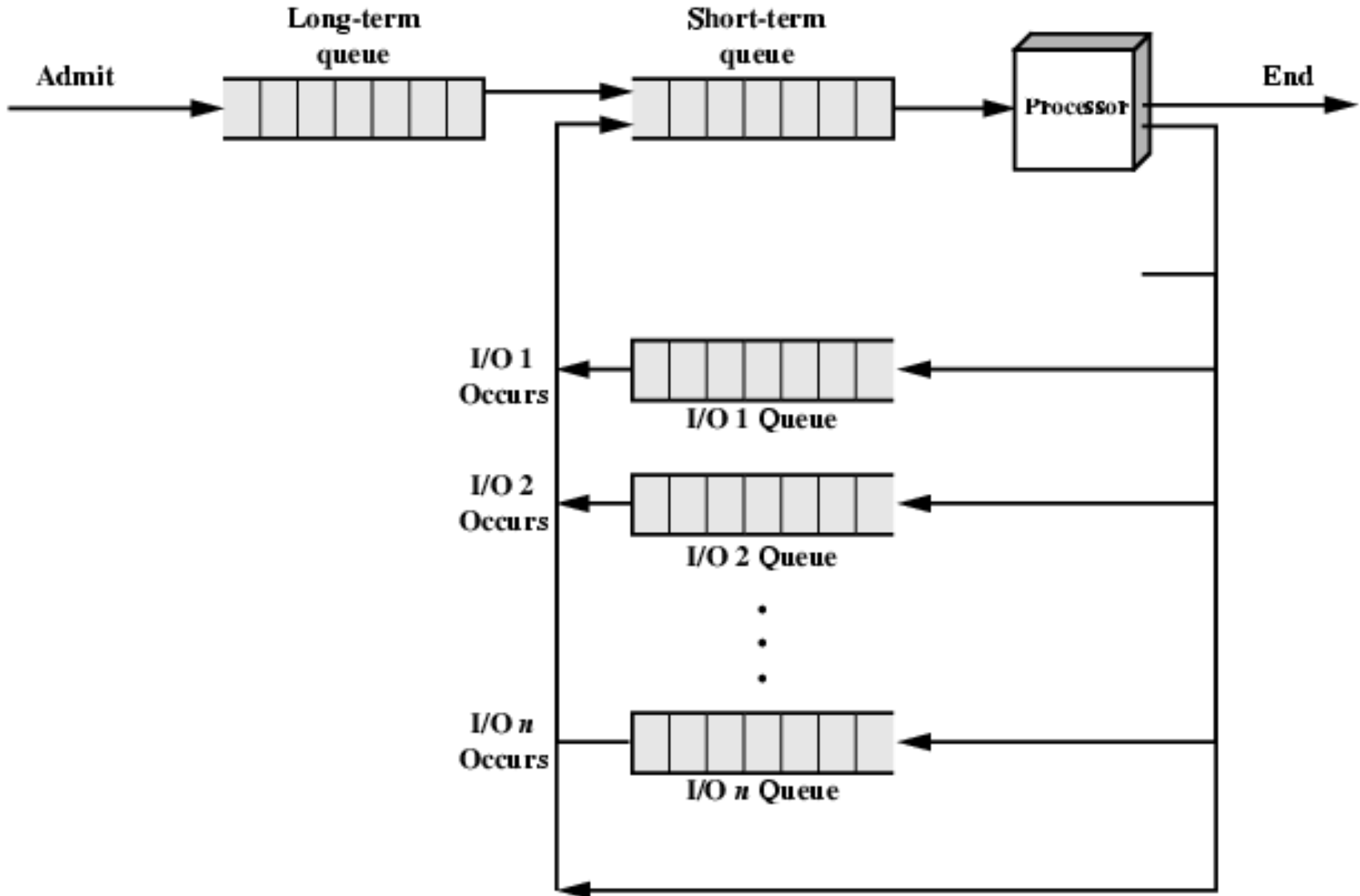
# Key Elements of O/S

---



# Process Scheduling

---



# Memory Management

---

- Uni-program
  - Memory split into two
  - One for Operating System (monitor)
  - One for currently executing program
- Multi-program
  - “User” part is sub-divided and shared among active processes

# Swapping

---

- Problem: I/O is so slow compared with CPU that even in multi-programming system, CPU can be idle most of the time
- Solutions:
  - Increase main memory
    - Expensive
    - Leads to larger programs
  - Swapping

# What is Swapping?

---

- Long term queue of processes stored on disk
- Processes “swapped” in as space becomes available
- As a process completes it is moved out of main memory
- If none of the processes in memory are ready (i.e. all I/O blocked)
  - Swap out a blocked process to intermediate queue
  - Swap in a ready process or a new process
  - But swapping is an I/O process...

# Partitioning

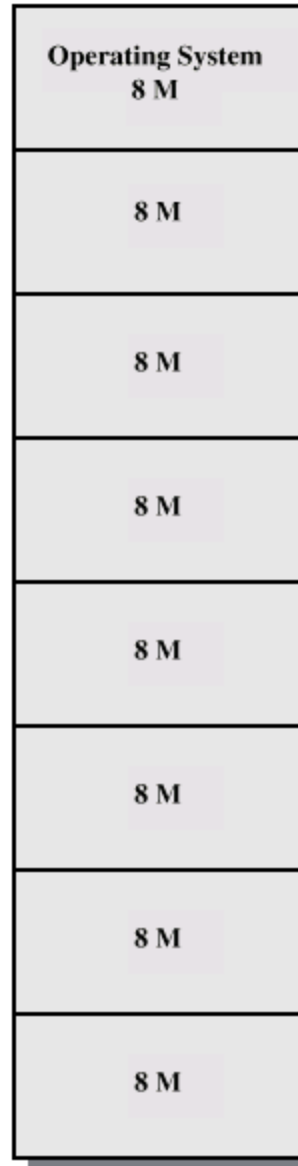
---

- Splitting memory into sections to allocate to processes (including Operating System)
- Fixed-sized partitions
  - May not be equal size
  - Process is fitted into smallest hole that will take it (best fit)
  - Some wasted memory
  - Leads to variable sized partitions

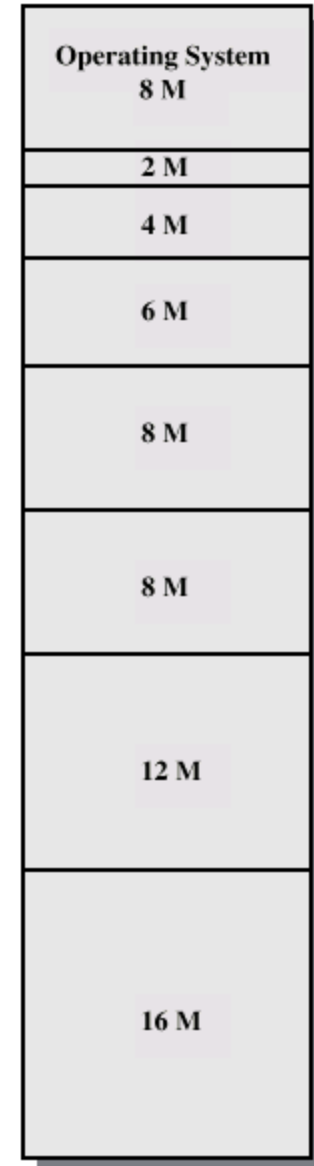


# Fixed Partitioning

---



(a) Equal-size partitions



(b) Unequal-size partitions

# Variable Sized Partitions (1)

---

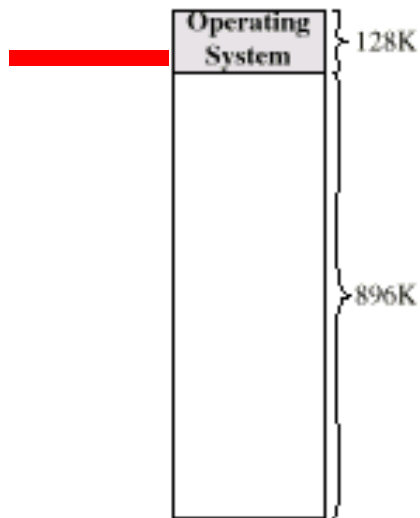
- Allocate exactly the required memory to a process
- This leads to a hole at the end of memory, too small to use
  - Only one small hole - less waste
- When all processes are blocked, swap out a process and bring in another
- New process may be smaller than swapped out process
- Another hole

## Variable Sized Partitions (2)

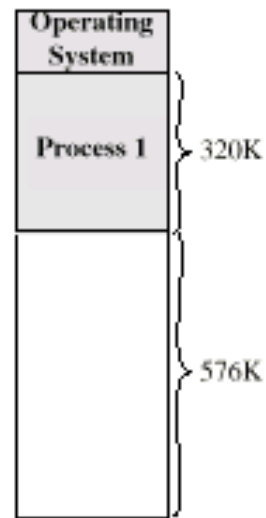
---

- Eventually have lots of holes (fragmentation)
- Solutions:
  - Coalesce - Join adjacent holes into one large hole
  - Compaction - From time to time go through memory and move all hole into one free block (c.f. disk de-fragmentation)

# Effect of Dynamic Partitioning



(a)



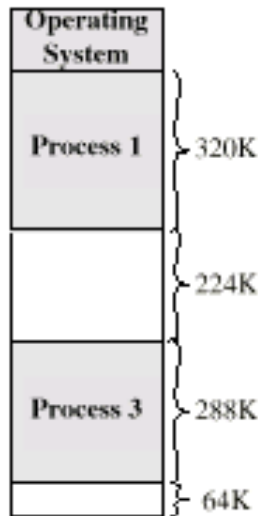
(b)



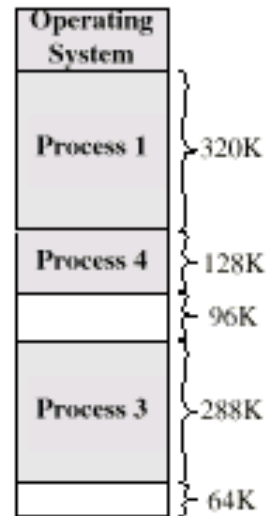
(c)



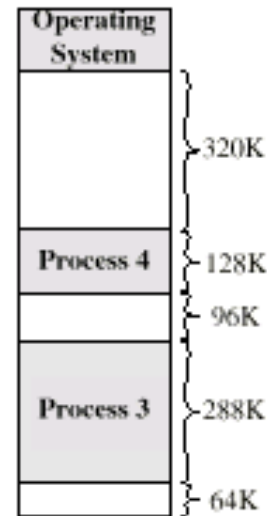
(d)



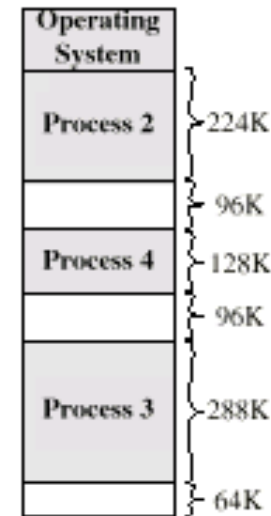
(e)



(f)



(g)



(h)

# Relocation

---

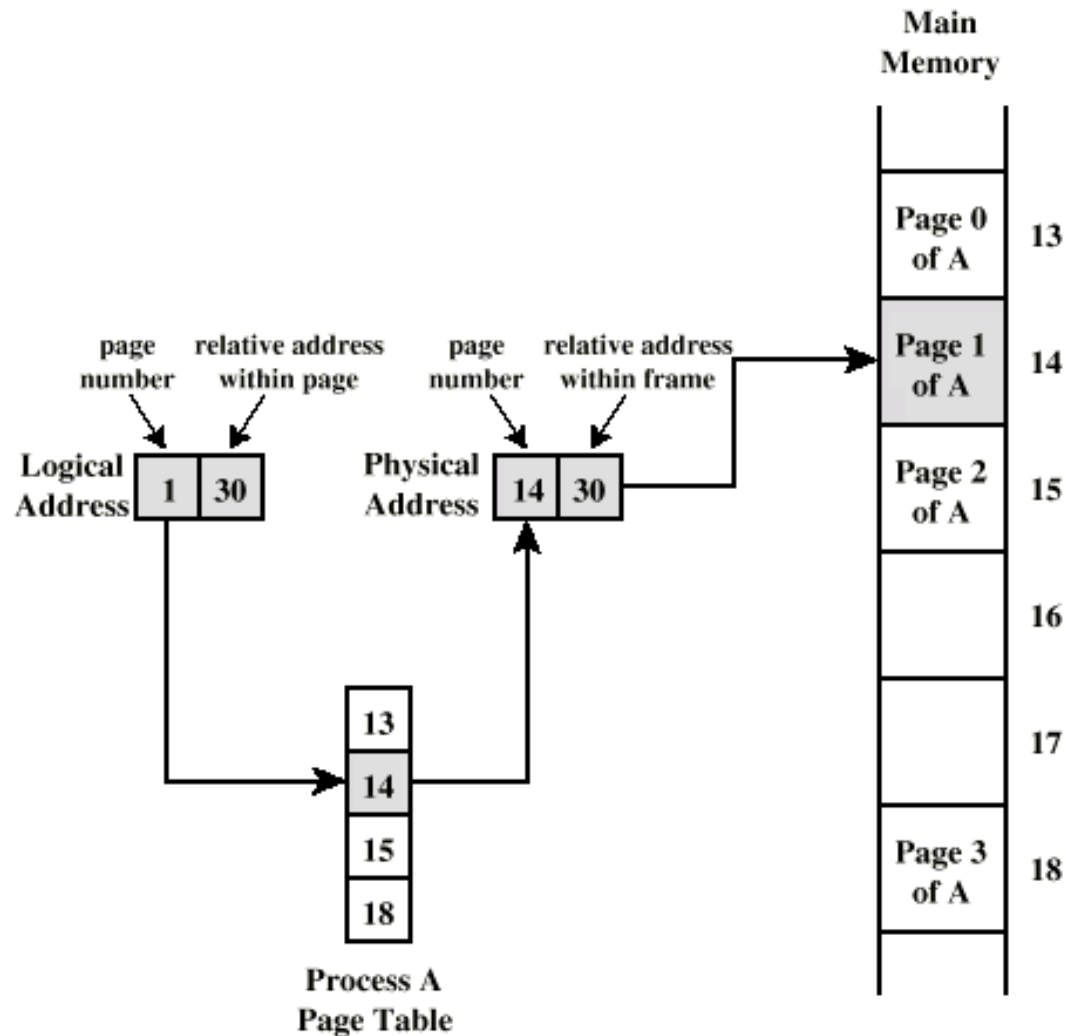
- No guarantee that process will load into the same place in memory
- Instructions contain addresses
  - Locations of data
  - Addresses for instructions (branching)
- Logical address - relative to beginning of program
- Physical address - actual location in memory (this time)
- Automatic conversion using base address

# Paging

---

- Split memory into equal sized, small chunks - page frames
- Split programs (processes) into equal sized small chunks - pages
- Allocate the required number page frames to a process
- Operating System maintains list of free frames
- A process does not require contiguous page frames
- Use page table to keep track

# Logical and Physical Addresses - Paging



# Virtual Memory

---

- Demand paging
  - Do not require all pages of a process in memory
  - Bring in pages as required
- Page fault
  - Required page is not in memory
  - Operating System must swap in required page
  - May need to swap out a page to make space
  - Select page to throw out based on recent history



# Thrashing

---

- Too many processes in too little memory
- Operating System spends all its time swapping
- Little or no real work is done
- Disk light is on all the time
  
- Solutions
  - Good page replacement algorithms
  - Reduce number of processes running
  - Fit more memory

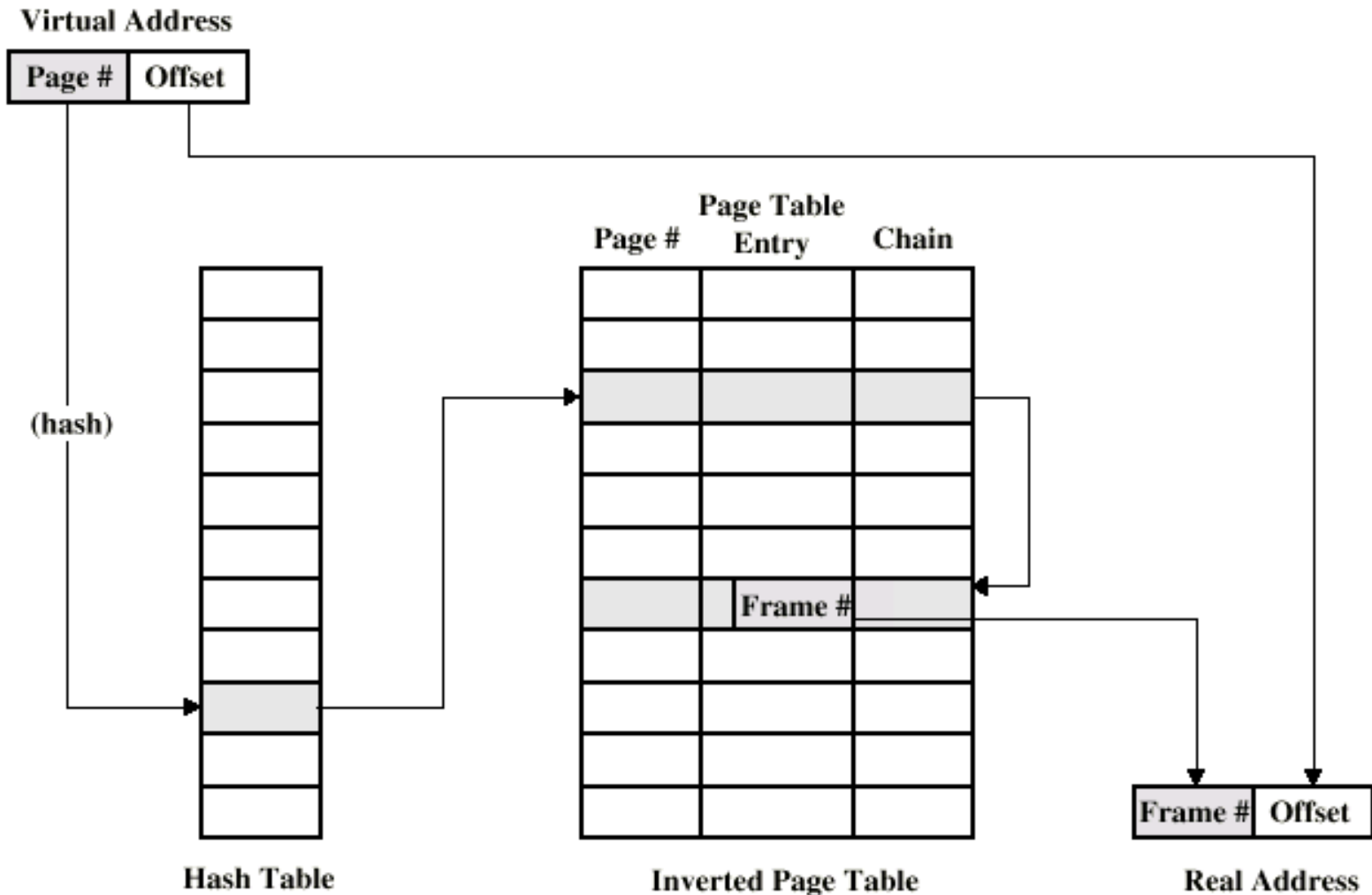
# Bonus

---

- We do not need all of a process in memory for it to run
- We can swap in pages as required
- So - we can now run processes that are bigger than total memory available!
  
- Main memory is called real memory
- User/programmer sees much bigger memory - virtual memory

# ***Inverted Page Table Structure***

---

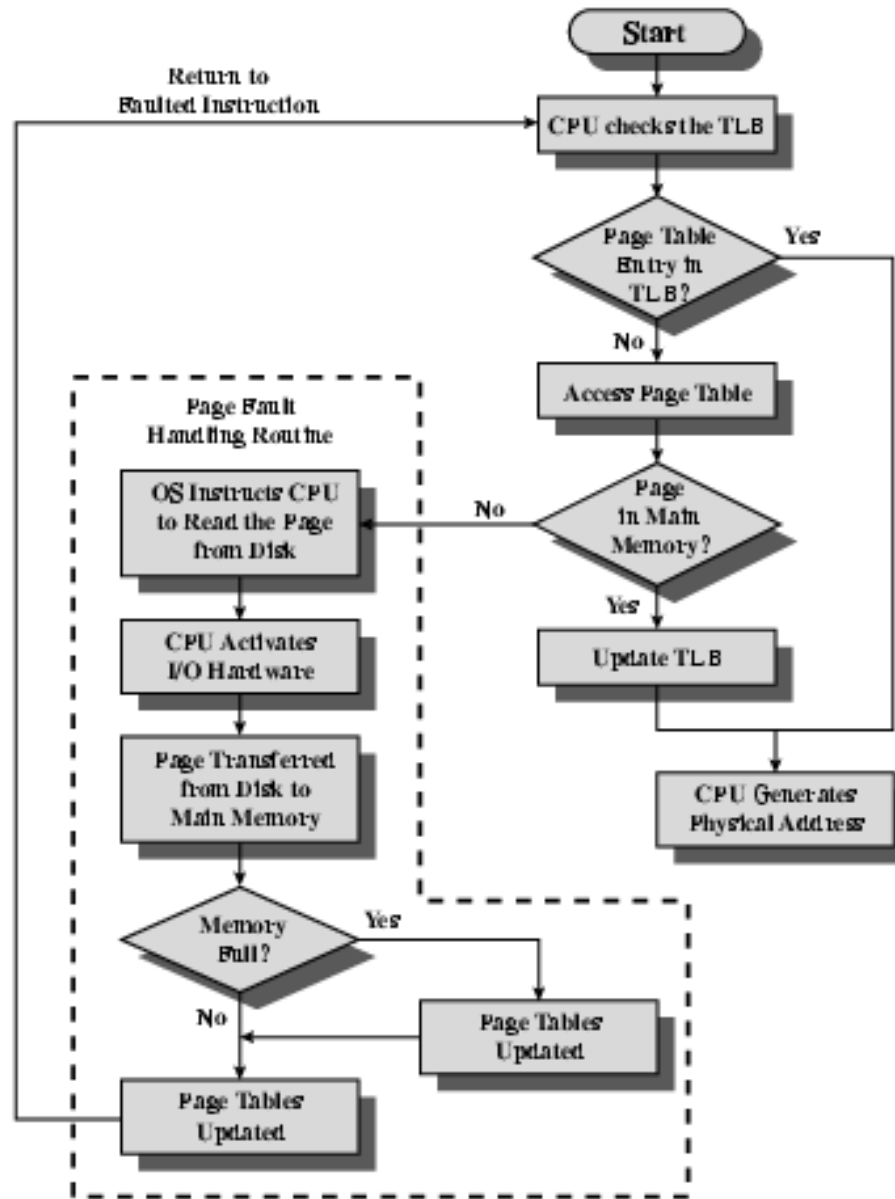


# Translation Lookaside Buffer

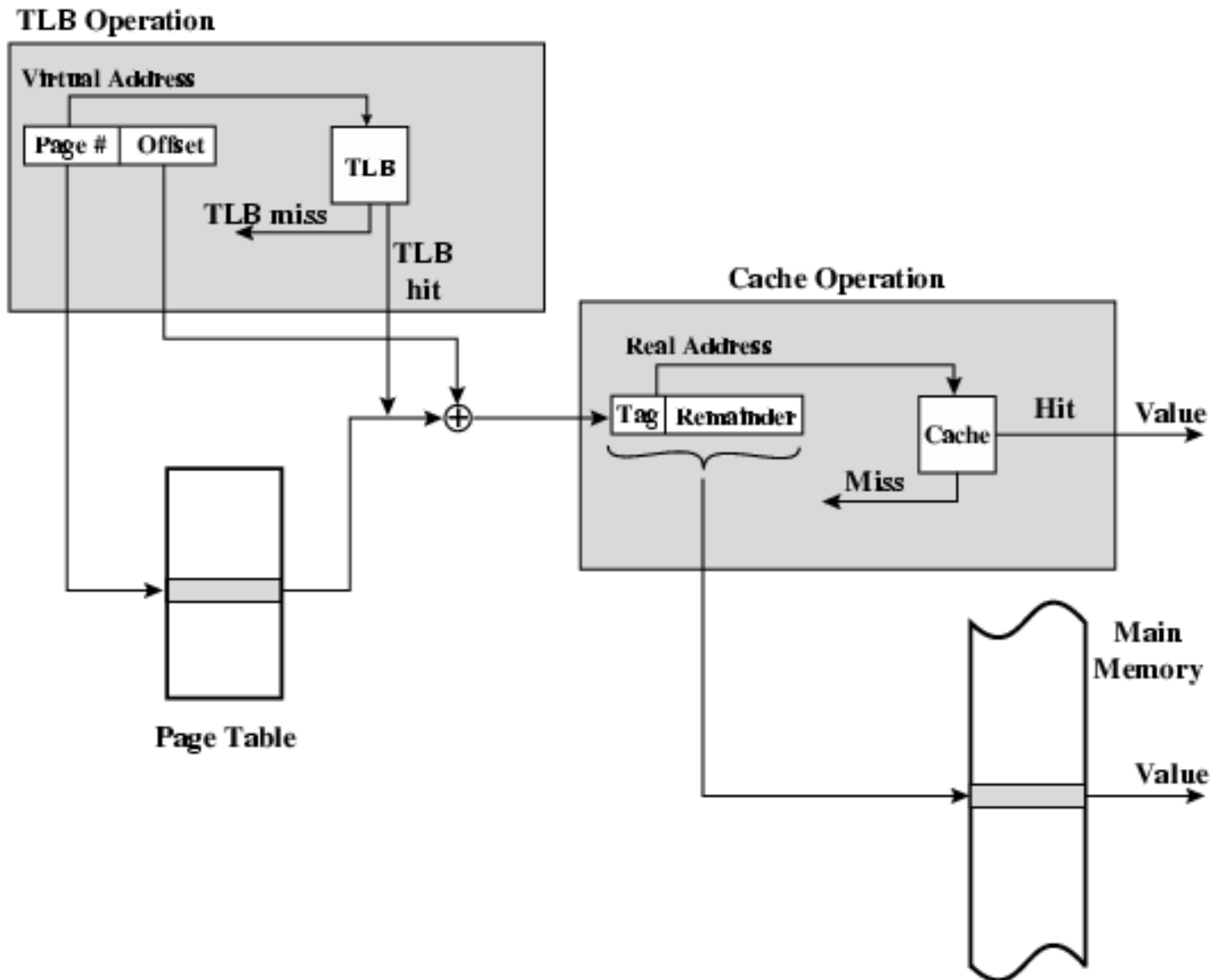
---

- Every virtual memory reference causes two physical memory access
  - Fetch page table entry
  - Fetch data
- Use special cache for page table
  - TLB

# TLB Operation



# TLB and Cache Operation



# Segmentation

---

- Paging is not (usually) visible to the programmer
- Segmentation is visible to the programmer
- Usually different segments allocated to program and data
- May be a number of program and data segments

# **Advantages of Segmentation**

---

- Simplifies handling of growing data structures
- Allows programs to be altered and recompiled independently, without re-linking and re-loading
- Lends itself to sharing among processes
- Lends itself to protection
- Some systems combine segmentation with paging

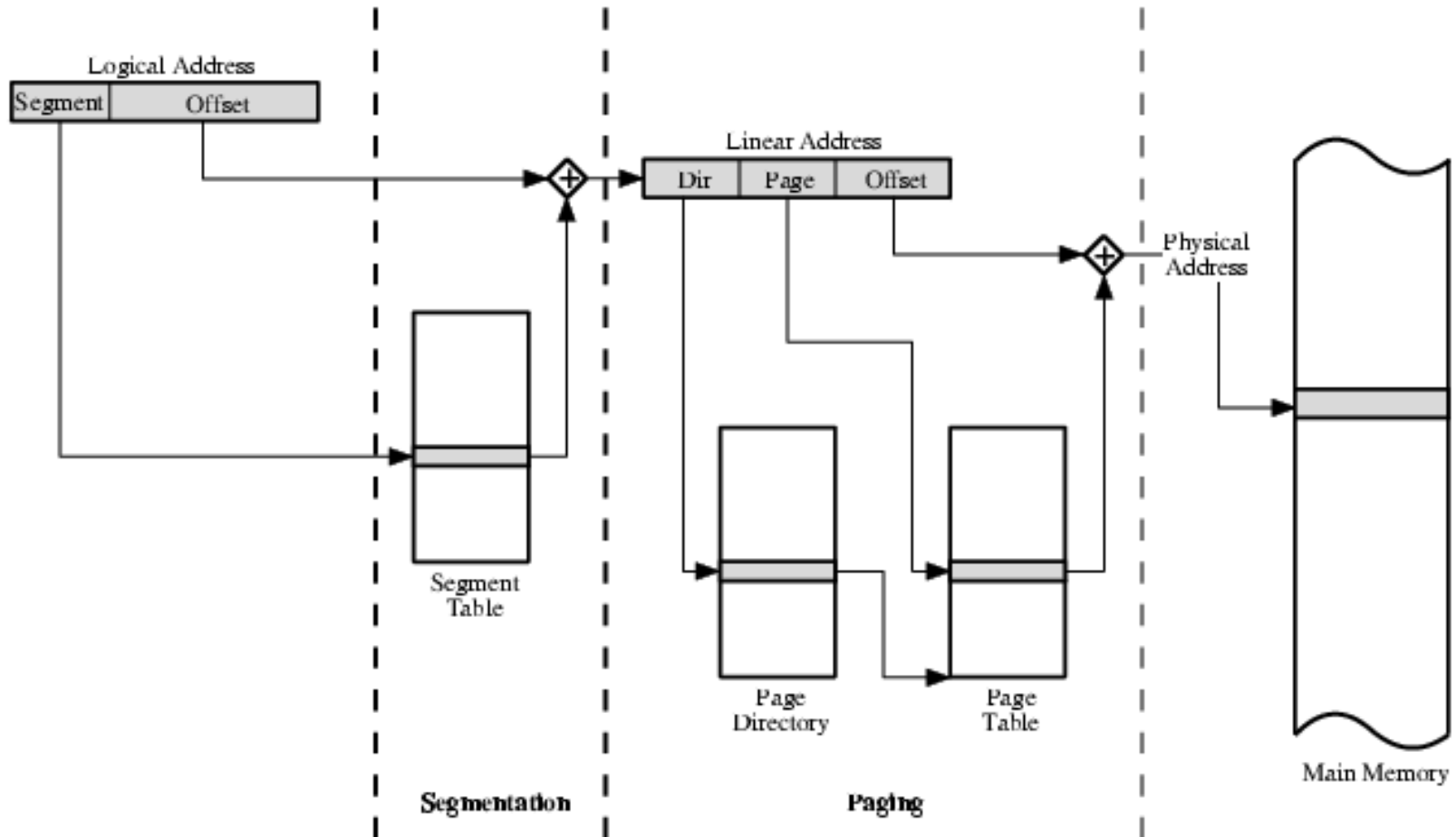


# Pentium II

---

- Hardware for segmentation and paging
- Unsegmented unpaged
  - virtual address = physical address
  - Low complexity
  - High performance
- Unsegmented paged
  - Memory viewed as paged linear address space
  - Protection and management via paging
  - Berkeley UNIX
- Segmented unpaged
  - Collection of local address spaces
  - Protection to single byte level
  - Translation table needed is on chip when segment is in memory
- Segmented paged
  - Segmentation used to define logical memory partitions subject to access control
  - Paging manages allocation of memory within partitions
  - Unix System V

# Pentium II Address Translation Mechanism



# Pentium II Segmentation

---

- Each virtual address is 16-bit segment and 32-bit offset
- 2 bits of segment are protection mechanism
- 14 bits specify segment
- Unsegmented virtual memory  $2^{32} = 4\text{Gbytes}$
- Segmented  $2^{46} = 64\text{ terabytes}$ 
  - Can be larger – depends on which process is active
  - Half (8K segments of 4Gbytes) is global
  - Half is local and distinct for each process

# Pentium II Protection

---

- Protection bits give 4 levels of privilege
  - 0 most protected, 3 least
  - Use of levels software dependent
  - Usually level 3 for applications, level 1 for O/S and level 0 for kernel (level 2 not used)
  - Level 2 may be used for apps that have internal security e.g. database
  - Some instructions only work in level 0

# Pentium II Paging

---

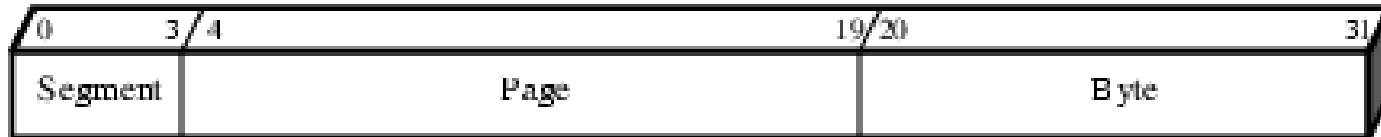
- Segmentation may be disabled
  - In which case linear address space is used
- Two level page table lookup
  - First, page directory
    - 1024 entries max
    - Splits 4G linear memory into 1024 page groups of 4Mbyte
    - Each page table has 1024 entries corresponding to 4Kbyte pages
    - Can use one page directory for all processes, one per process or mixture
    - Page directory for current process always in memory
  - Use TLB holding 32 page table entries
  - Two page sizes available 4k or 4M

# PowerPC Memory Management Hardware

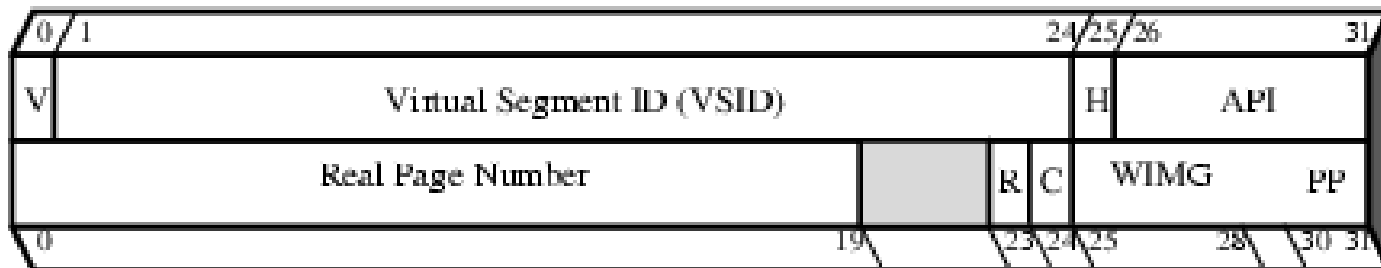
---

- 32 bit – paging with simple segmentation
  - 64 bit paging with more powerful segmentation
- Or, both do block address translation
  - Map 4 large blocks of instructions & 4 of memory to bypass paging
  - e.g. OS tables or graphics frame buffers
- 32 bit effective address
  - 12 bit byte selector
    - =4kbyte pages
  - 16 bit page id
    - 64k pages per segment
  - 4 bits indicate one of 16 segment registers
    - Segment registers under OS control

# PowerPC 32-bit Memory Management Formats



(a) Effective address



V = Entry valid bit  
 H = Hash function identifier  
 API = Abbreviated page index

R = Referenced bit  
 C = Changed bit  
 WIMG = Cache and storage access control bits  
 PP = Page protection bits

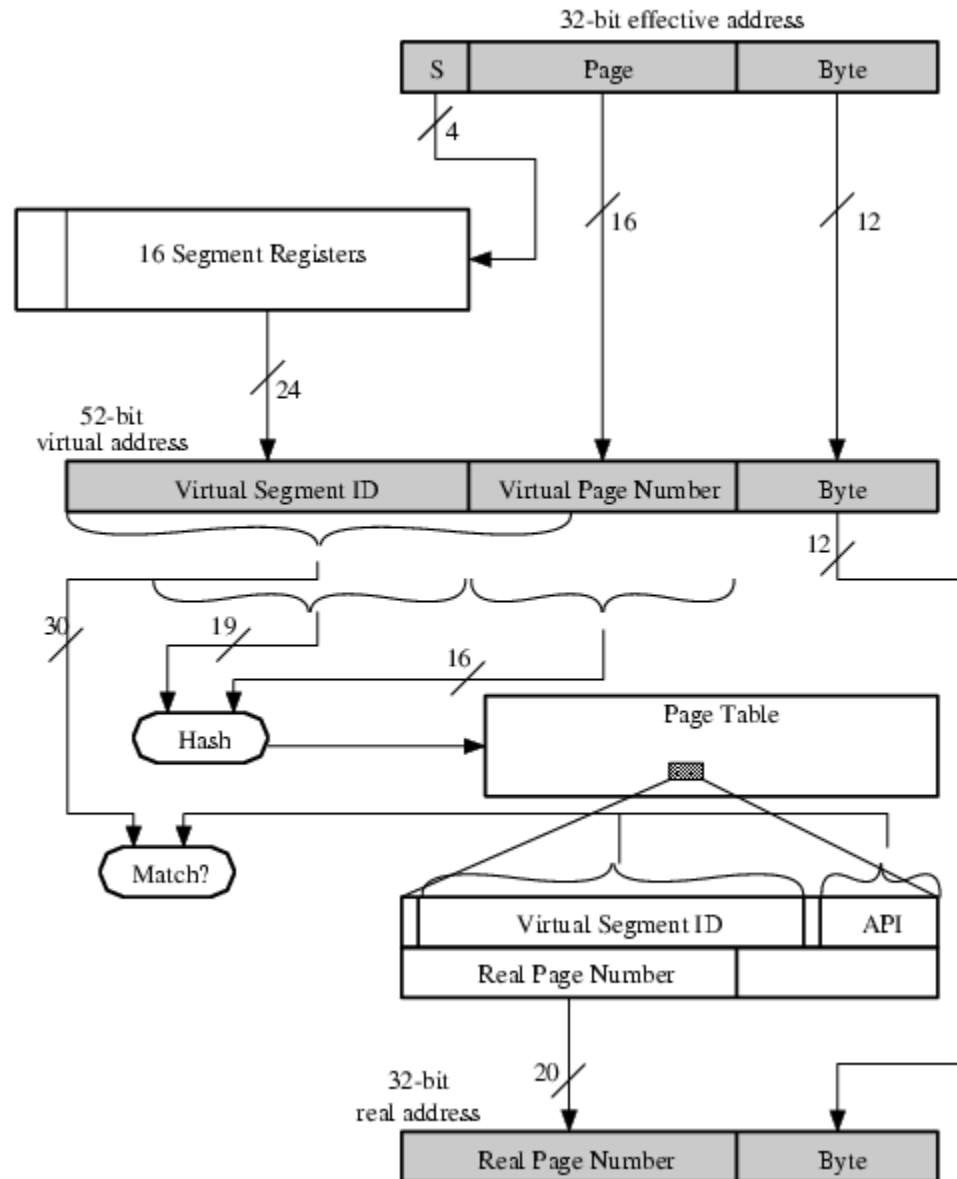
■ = reserved

(b) Page Table Entry



(c) Real address

# PowerPC 32-bit Address Translation





# Recommended Reading

---

- Stallings, W. Operating Systems, Internals and Design Principles, Prentice Hall 1998
- Loads of Web sites on Operating Systems