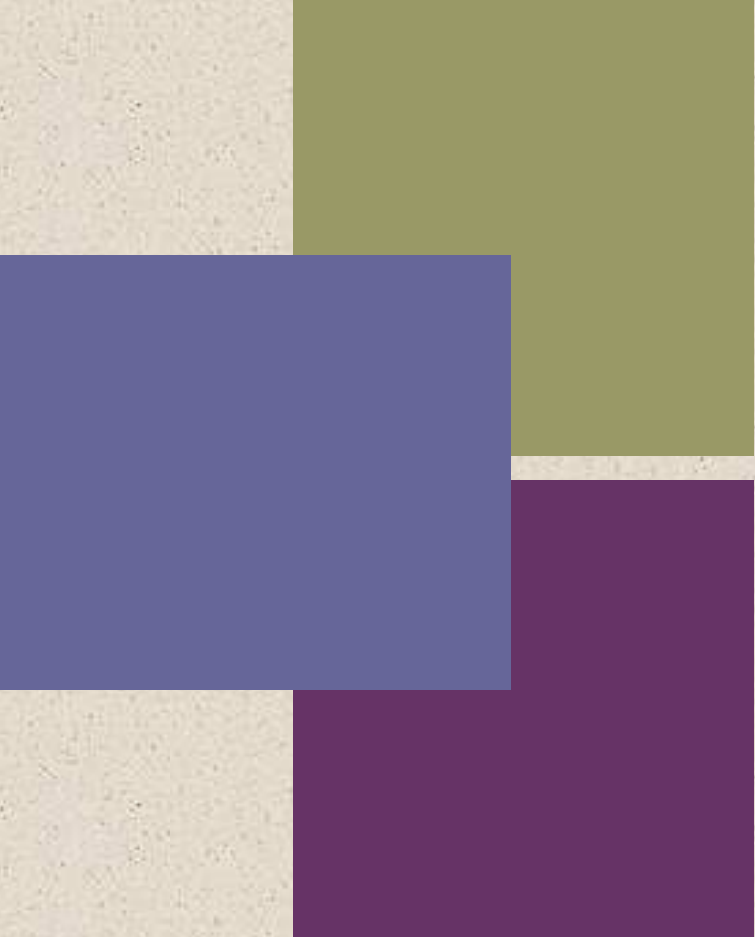


**William Stallings
Computer Organization
and Architecture
10th Edition**



+ Chapter 8

Operating System Support

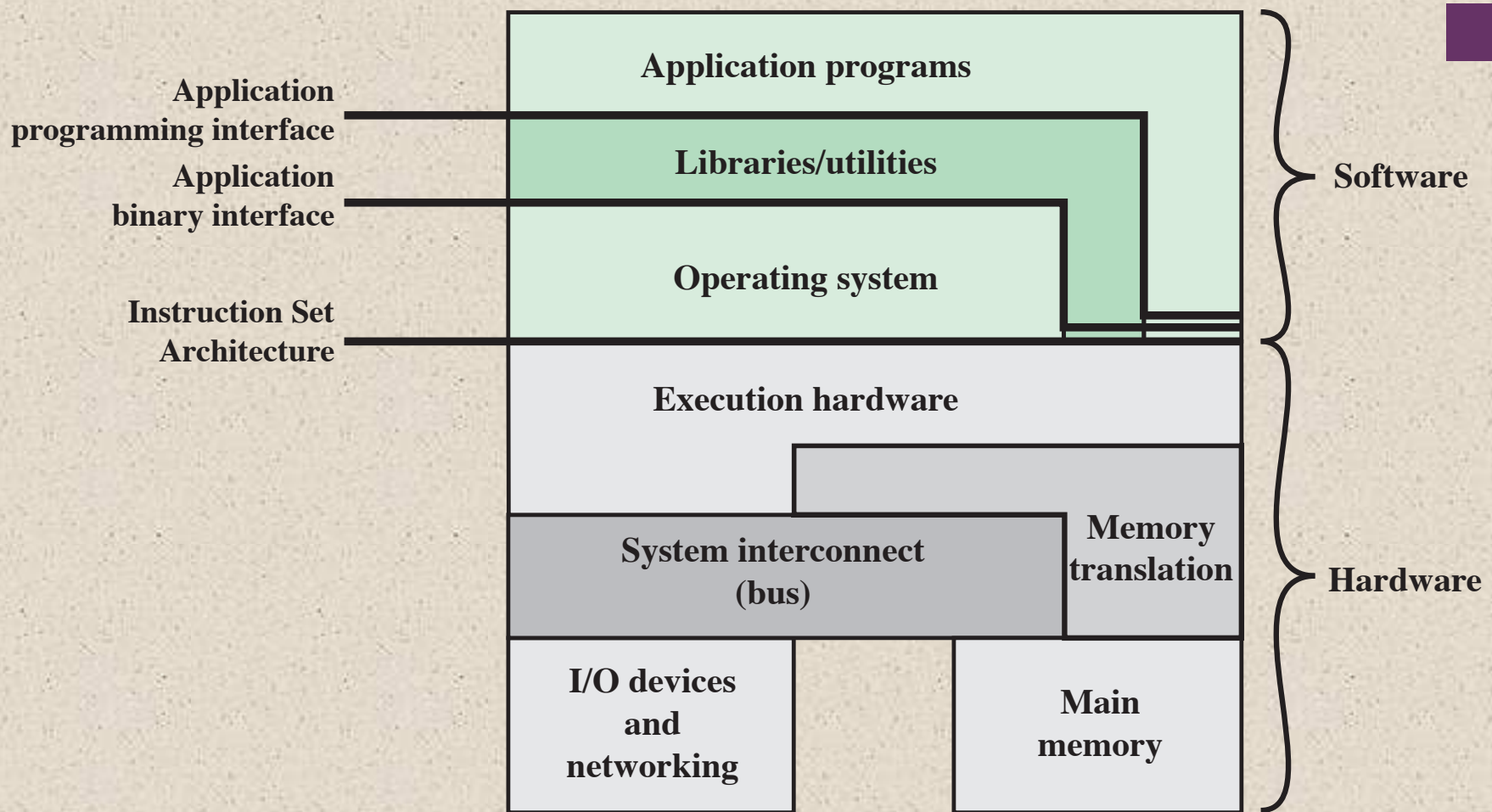


Figure 8.1 Computer Hardware and Software Structure



Operating System (OS) Services

- The most important system program
- Masks the details of the hardware from the programmer and provides the programmer with a convenient interface for using the system
- The OS typically provides services in the following areas:
 - Program creation
 - Program execution
 - Access to I/O devices
 - Controlled access to files
 - System access
 - Error detection and response
 - Accounting



Interfaces

- Key interfaces in a typical computer system:

Instruction set architecture (ISA)

Defines the machine language instructions that a computer can follow

Boundary between hardware and software

Application binary interface (ABI)

Defines a standard for binary portability across programs

Defines the system call interface to the operating system and the hardware resources and services available in a system through the user ISA

Application programming interface (API)

Gives a program access to the hardware resources and services available in a system through the user ISA supplemented with high-level language (HLL) library calls

Using an API enables application software to be ported easily to other systems that support the same API



Operating System as Resource Manager

A computer is a set of resources for the movement, storage, and processing of data and for the control of these functions

- The OS is responsible for managing these resources

The OS as a control mechanism is unusual in two respects:

- The OS functions in the same way as ordinary computer software – it is a program executed by the processor
- The OS frequently relinquishes control and must depend on the processor to allow it to regain control



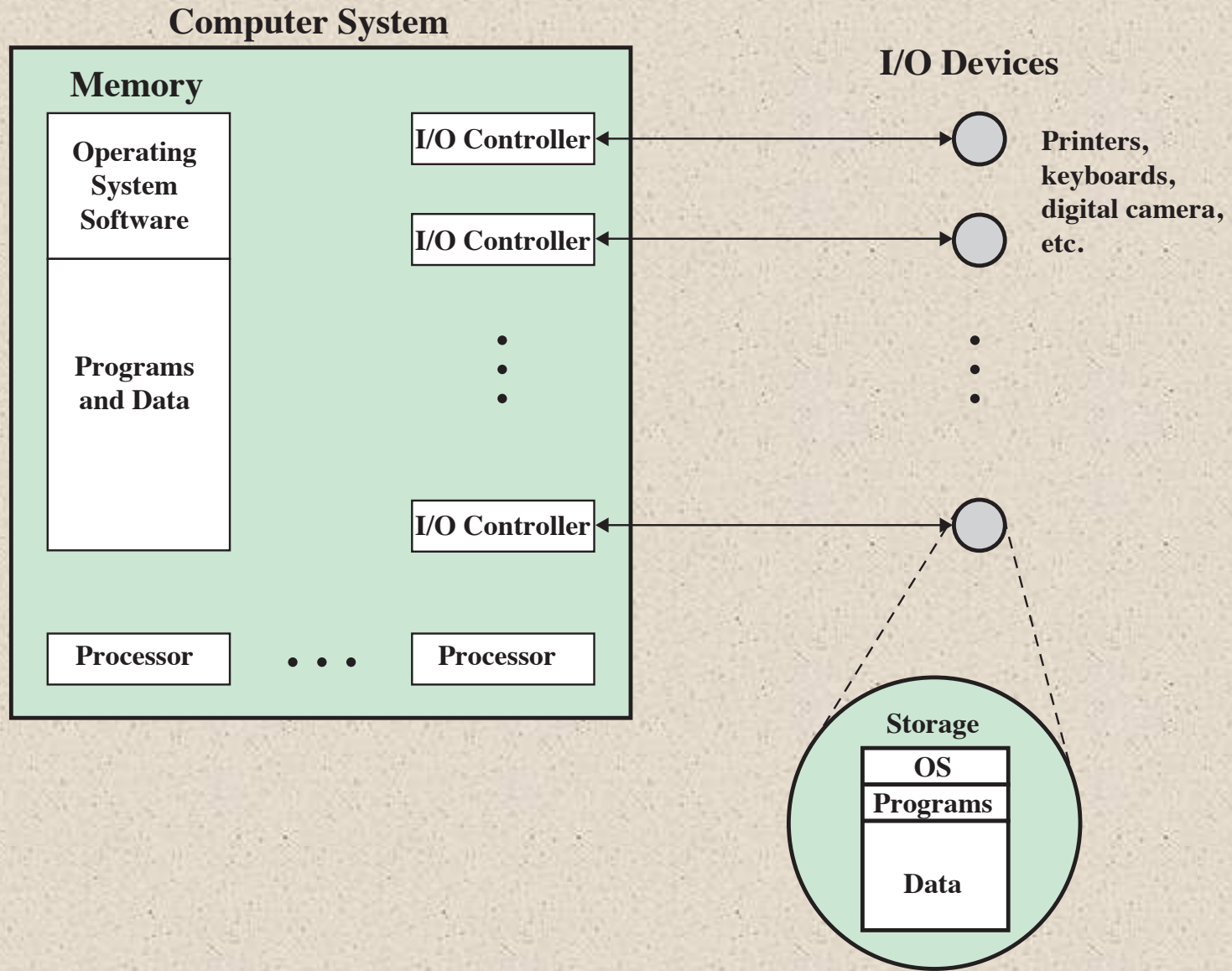
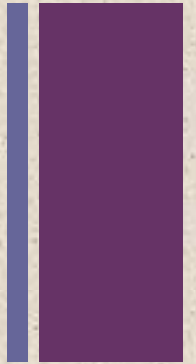


Figure 8.2 The Operating System as Resource Manager



Types of Operating Systems

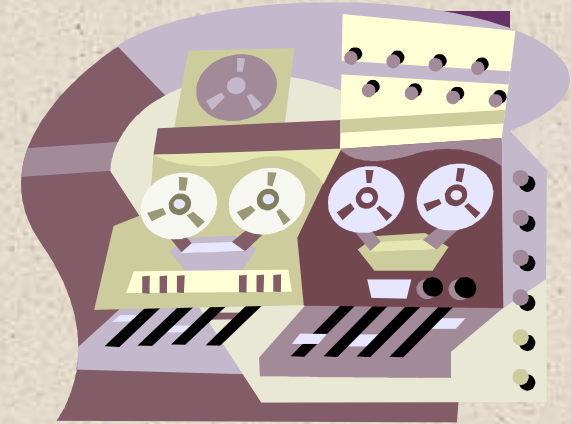


- **Interactive system**
 - The user/programmer interacts directly with the computer to request the execution of a job or to perform a transaction
 - User may, depending on the nature of the application, communicate with the computer during the execution of the job

- **Batch system**
 - Opposite of interactive
 - The user's program is batched together with programs from other users and submitted by a computer operator
 - After the program is completed results are printed out for the user



Early Systems



- From the late 1940s to the mid-1950s the programmer interacted directly with the computer hardware – there was no OS
 - Processors were run from a console consisting of display lights, toggle switches, some form of input device and a printer

- Problems:
 - Scheduling
 - Sign-up sheets were used to reserve processor time
 - This could result in wasted computer idle time if the user finished early
 - If problems occurred the user could be forced to stop before resolving the problem
 - Setup time
 - A single program could involve
 - Loading the compiler plus the source program into memory
 - Saving the compiled program
 - Loading and linking together the object program and common functions

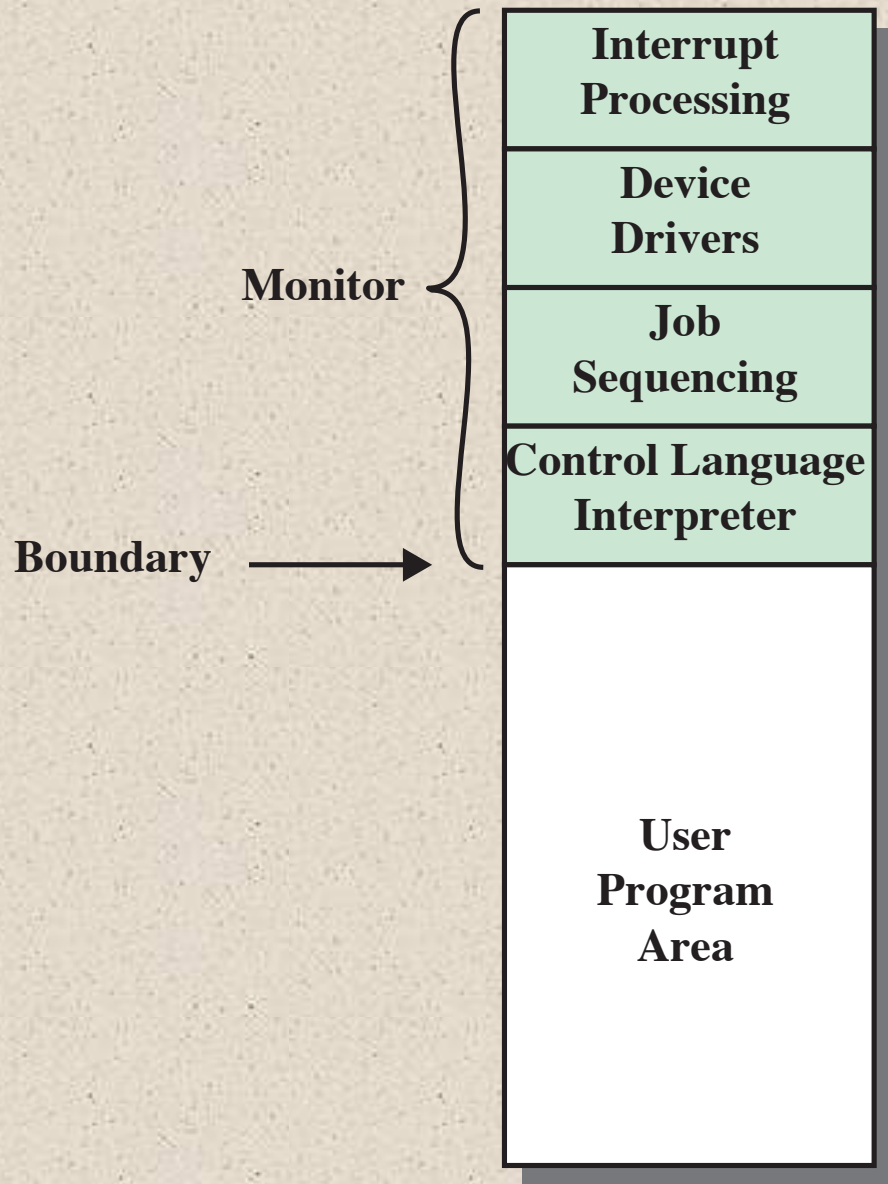


Figure 8.3 Memory Layout for a Resident Monitor

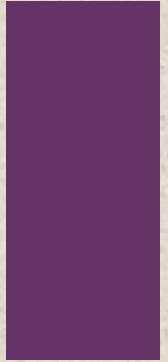
+ From the View of the Processor . . .

- Processor executes instructions from the portion of main memory containing the monitor
 - These instructions cause the next job to be read in another portion of main memory
 - The processor executes the instruction in the user's program until it encounters an ending or error condition
 - Either event causes the processor to fetch its next instruction from the monitor program
- The monitor handles setup and scheduling
 - A batch of jobs is queued up and executed as rapidly as possible with no idle time
- Job control language (JCL)
 - Special type of programming language used to provide instructions to the monitor
- Example:
 - \$JOB
 - \$FTN
 - ... Some Fortran instructions
 - \$LOAD
 - \$RUN
 - ... Some data
 - \$END
- Monitor, or batch OS, is simply a computer program
 - It relies on the ability of the processor to fetch instructions from various portions of main memory in order to seize and relinquish control alternately

**Each FORTRAN instruction and each item of data is on a separate punched card or a separate record on tape. In addition to FORTRAN and data lines, the job includes job control instructions, which are denoted by the beginning "\$".



Desirable Hardware Features



■ Memory protection

- User program must not alter the memory area containing the monitor
- The processor hardware should detect an error and transfer control to the monitor
- The monitor aborts the job, prints an error message, and loads the next job

■ Timer


- Used to prevent a job from monopolizing the system
- If the timer expires an interrupt occurs and control returns to monitor

■ Privileged instructions

- Can only be executed by the monitor
- If the processor encounters such an instruction while executing a user program an error interrupt occurs
- I/O instructions are privileged so the monitor retains control of all I/O devices

■ Interrupts

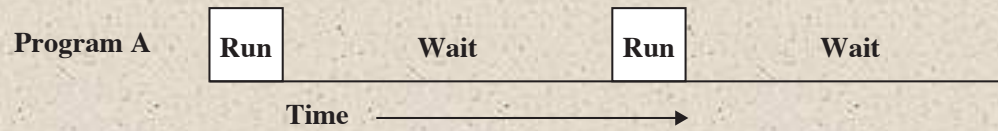
- Gives the OS more flexibility in relinquishing control to and regaining control from user programs



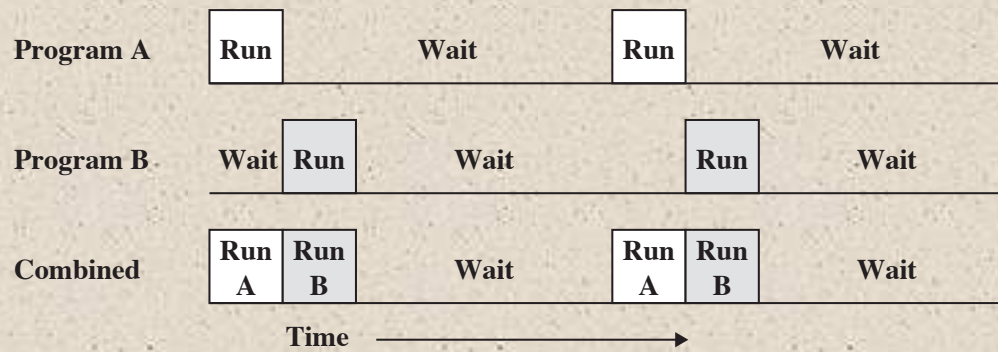
Read one record from file	15 μ s
Execute 100 instructions	1 μ s
Write one record to file	<u>15 μs</u>
TOTAL	31 μ s

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

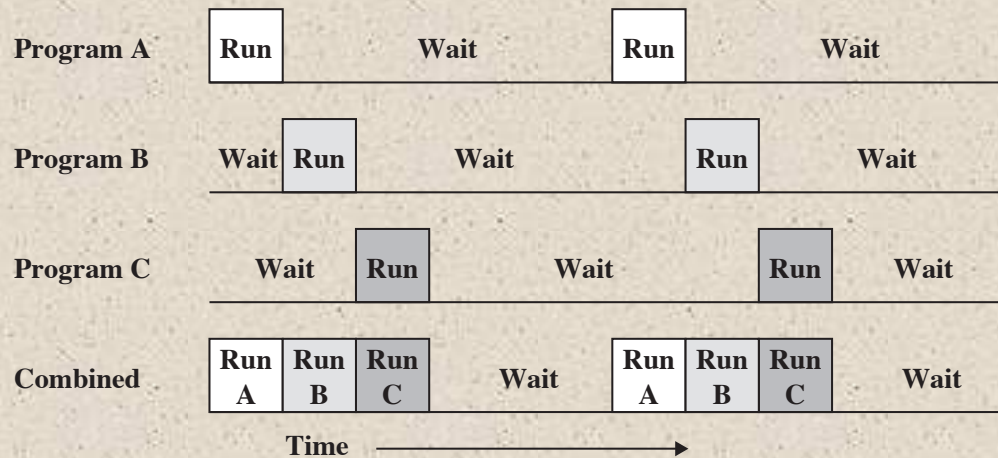
Figure 8.4 System Utilization Example



(a) Uniprogramming



(b) Multiprogramming with two programs



(c) Multiprogramming with three programs

Figure 8.5 Multiprogramming Example



Table 8.1
Sample Program Execution Attributes

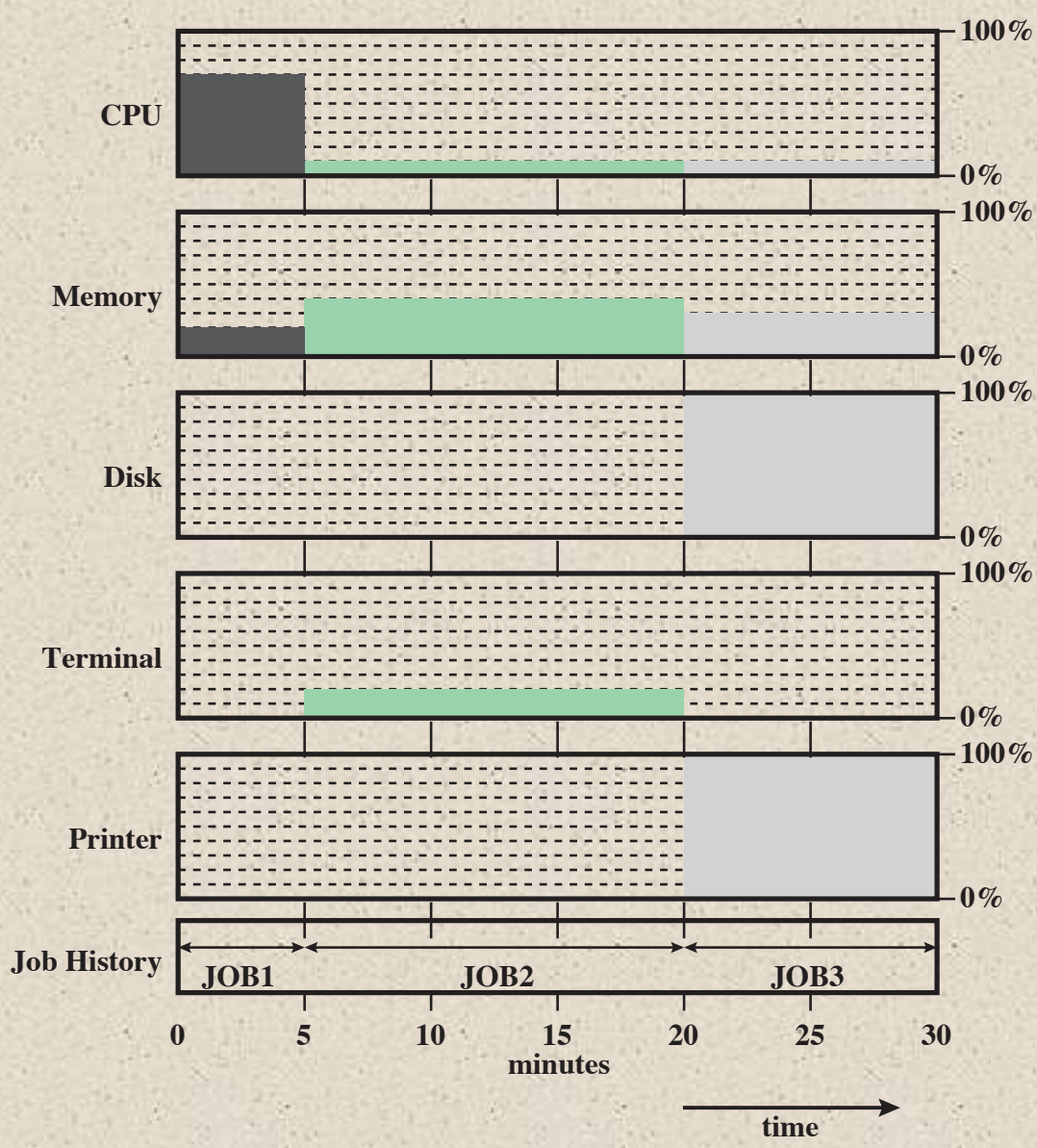
	JOB1	JOB2	JOB3
Type of job	Heavy compute	Heavy I/O	Heavy I/O
Duration	5 min	15 min	10 min
Memory required	50 M	100 M	80 M
Need disk?	No	No	Yes
Need terminal?	No	Yes	No
Need printer?	No	No	Yes



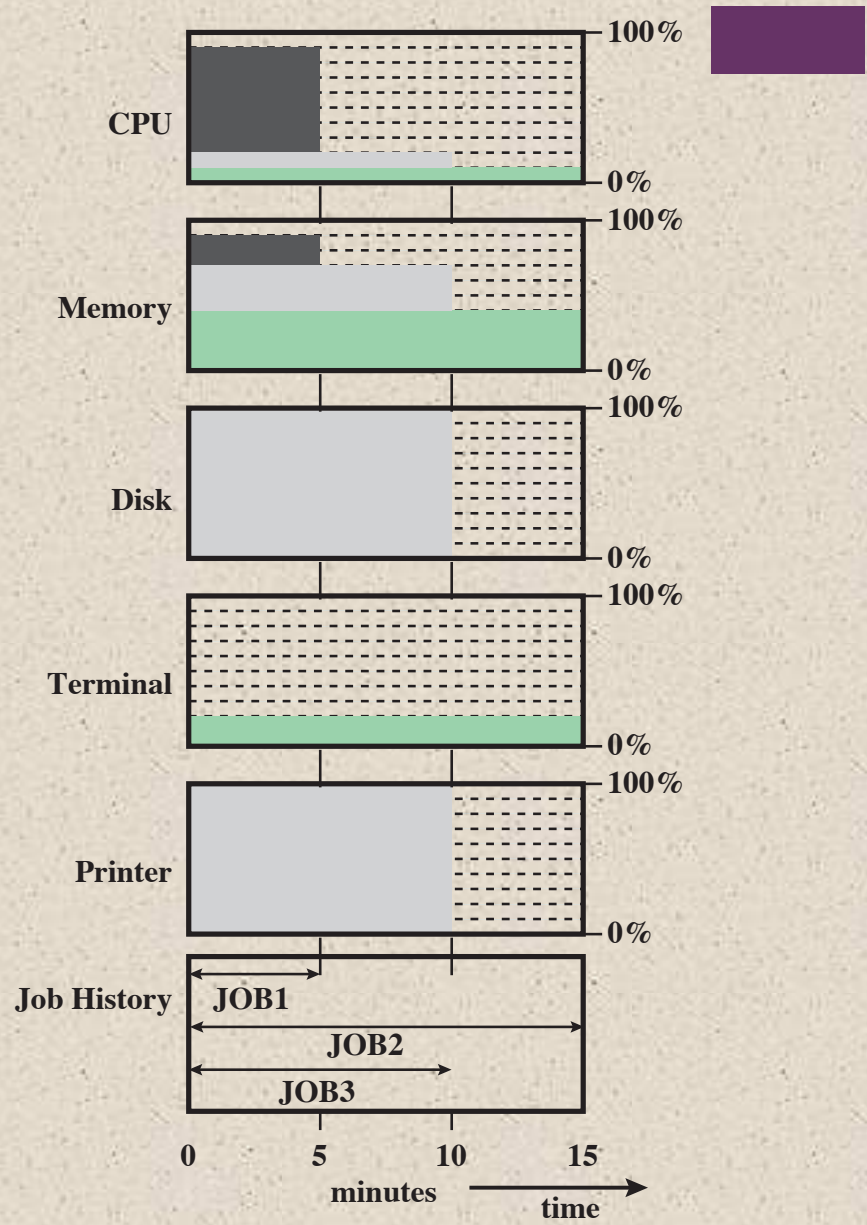
Table 8.2

Effects of Multiprogramming on Resource Utilization

	Uniprogramming	Multiprogramming
Processor use	20%	40%
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



(a) Uniprogramming



(b) Multiprogramming

Figure 8.6 Utilization Histograms



Time Sharing Systems



- Used when the user interacts directly with the computer
- Processor's time is shared among multiple users
- Multiple users simultaneously access the system through terminals, with the OS interleaving the execution of each user program in a short burst or quantum of computation
- Example:
 - If there are n users actively requesting service at one time, each user will only see on the average $1/n$ of the effective computer speed



Table 8.3

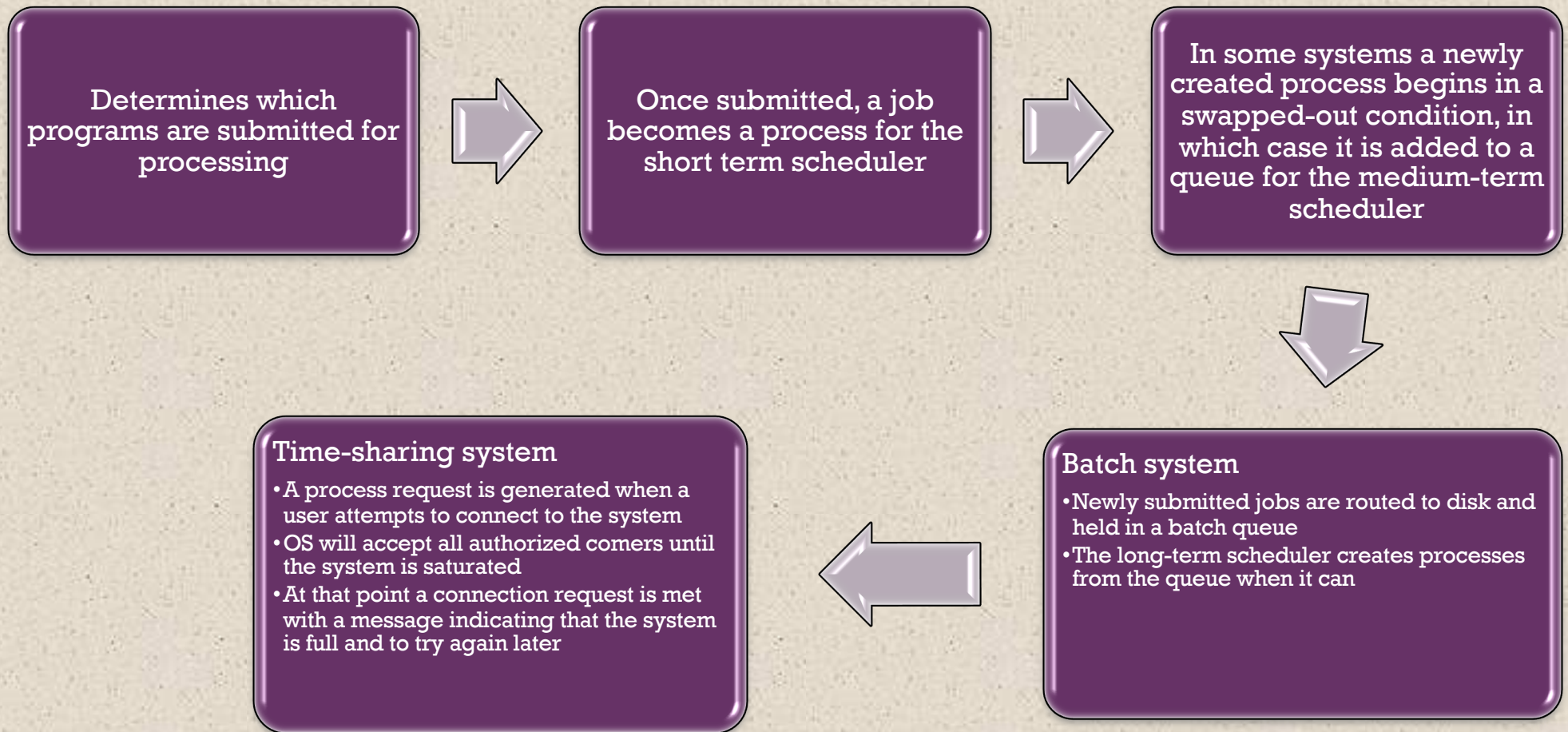
Batch Multiprogramming versus Time Sharing

	Batch Multiprogramming	Time Sharing
Principal objective	Maximize processor use	Minimize response time
Source of directives to operating system	Job control language commands provided with the job	Commands entered at the terminal

Table 8.4 Types of Scheduling

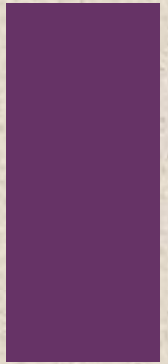
Long-term scheduling	The decision to add to the pool of processes to be executed
Medium-term scheduling	The decision to add to the number of processes that are partially or fully in main memory
Short-term scheduling	The decision as to which available process will be executed by the processor
I/O scheduling	The decision as to which process's pending I/O request shall be handled by an available I/O device

Long Term Scheduling





Medium-Term Scheduling and Short-Term Scheduling



Medium-Term

- Part of the swapping function
- Swapping-in decision is based on the need to manage the degree of multiprogramming
- Swapping-in decision will consider the memory requirements of the swapped-out processes

Short-Term

- Also known as the dispatcher
- Executes frequently and makes the fine-grained decision of which job to execute next

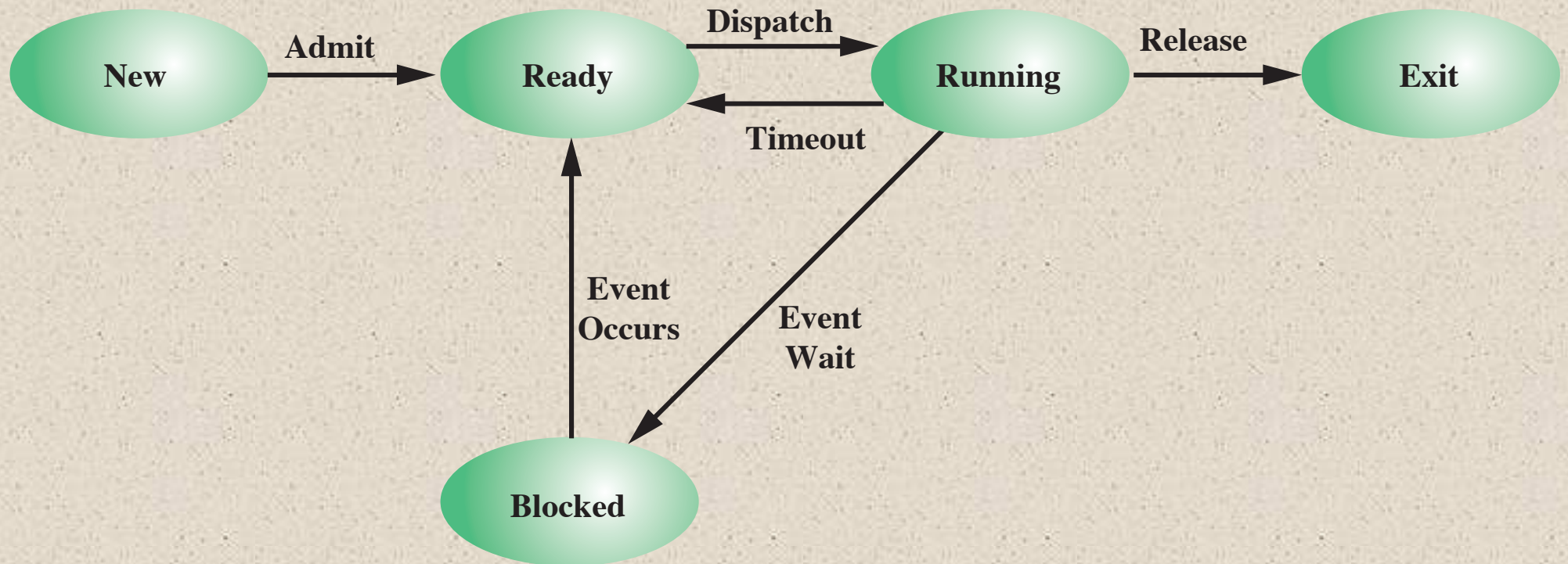


Figure 8.7 Five-State Process Model

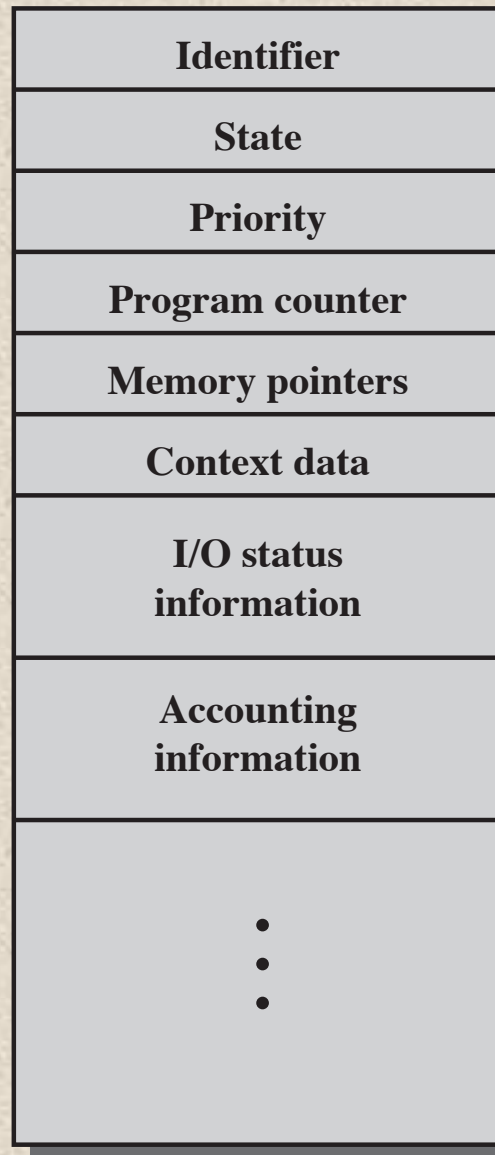


Figure 8.8 Process Control Block

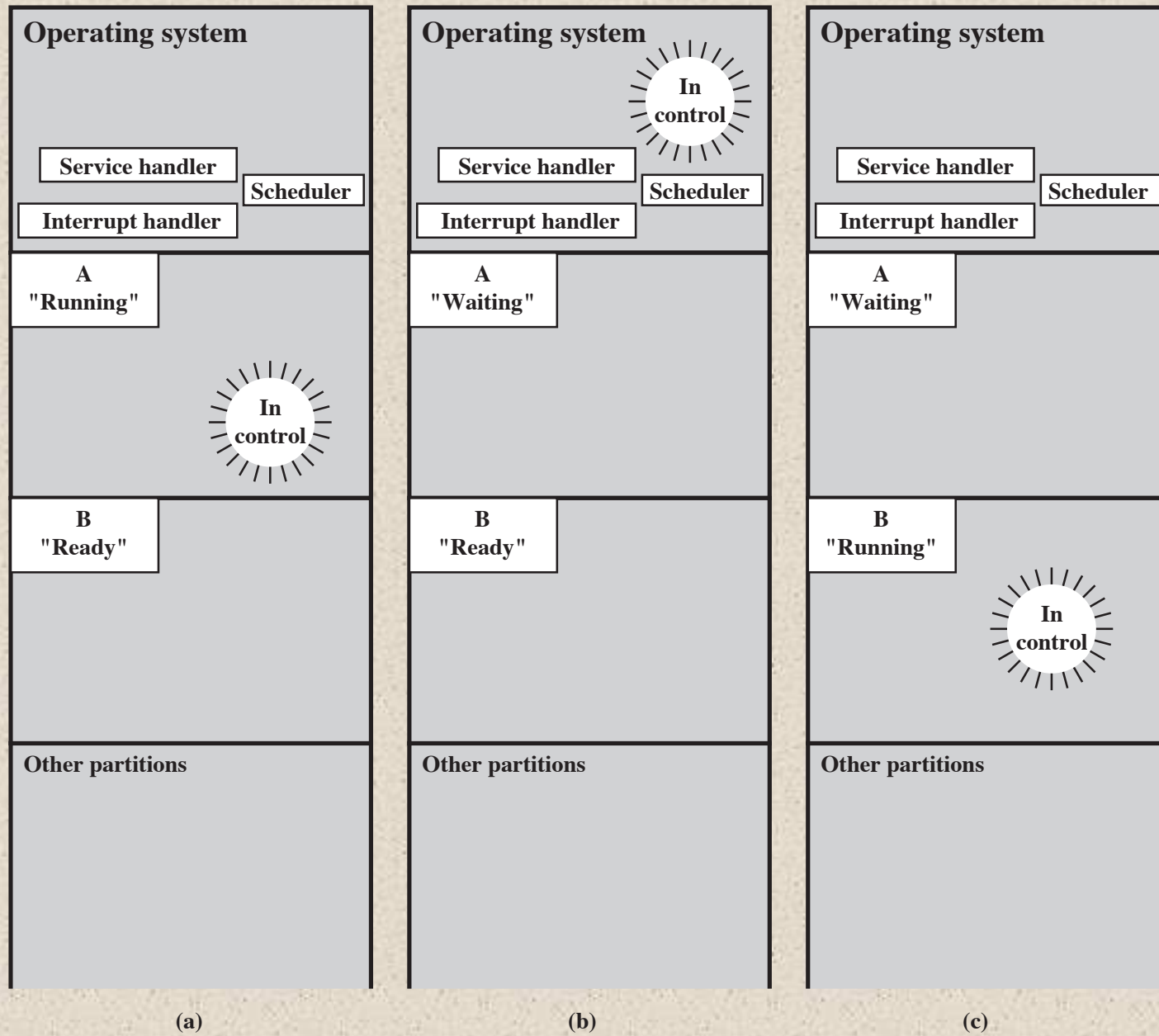


Figure 8.9 Scheduling Example

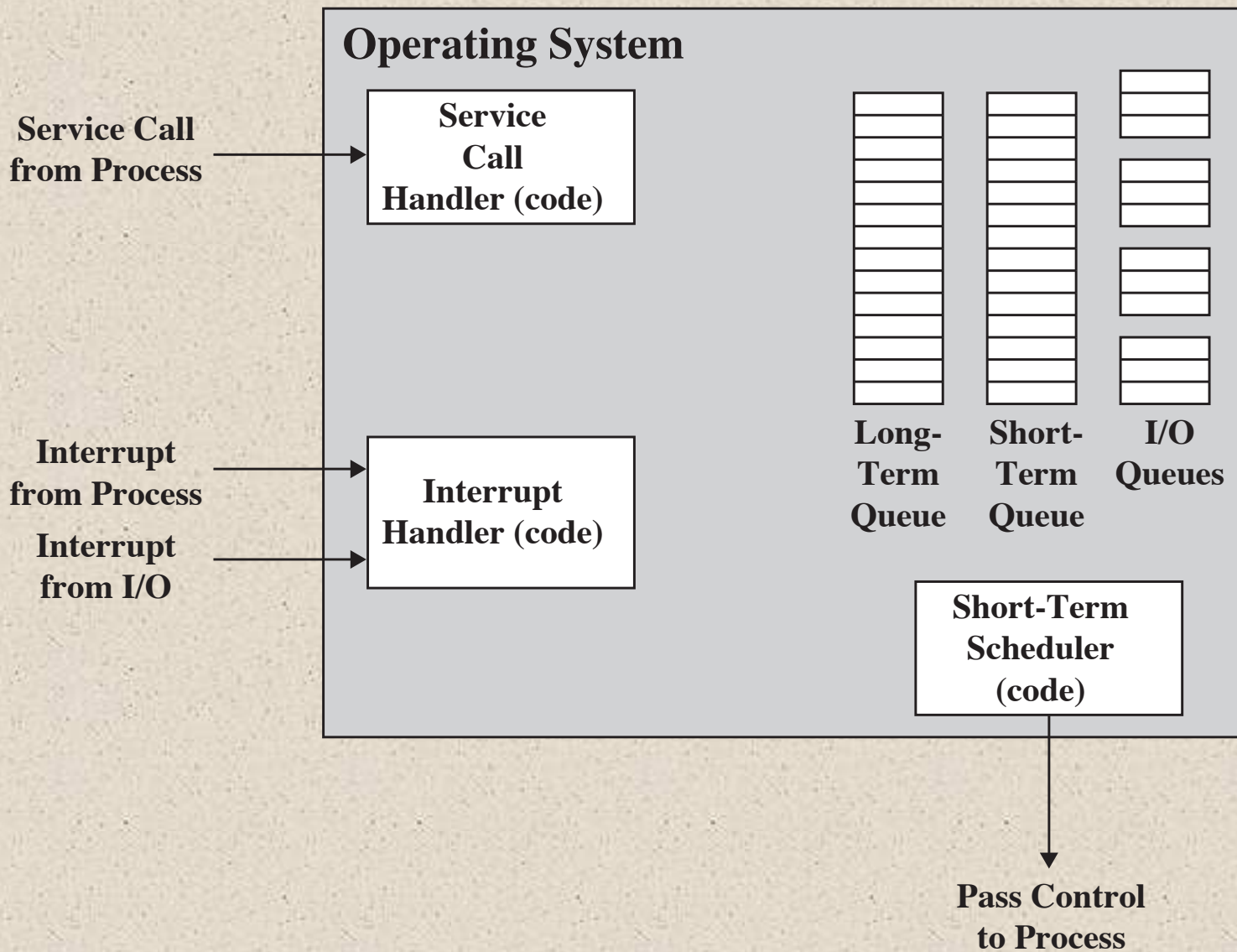


Figure 8.10 Key Elements of an Operating System for Multiprogramming

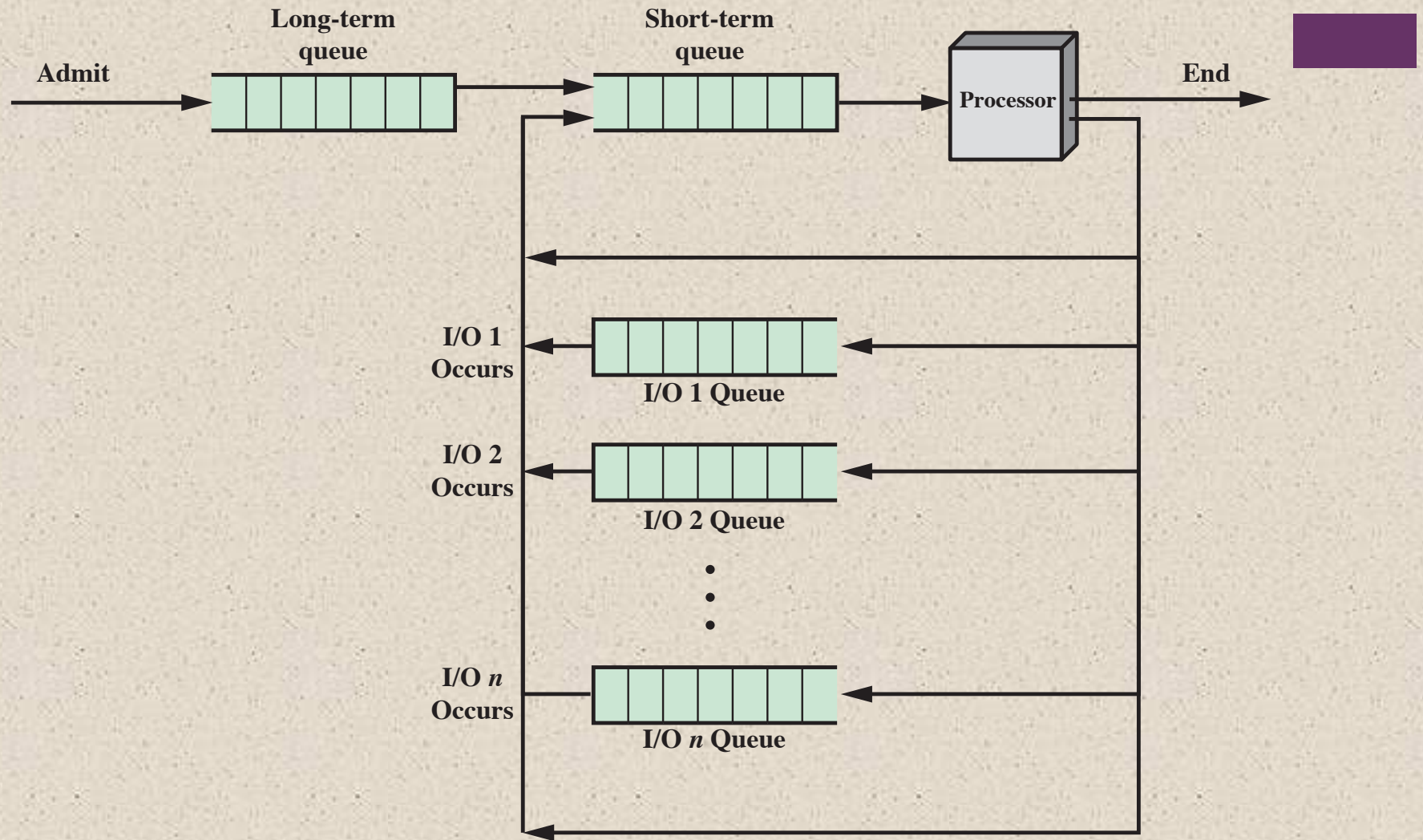
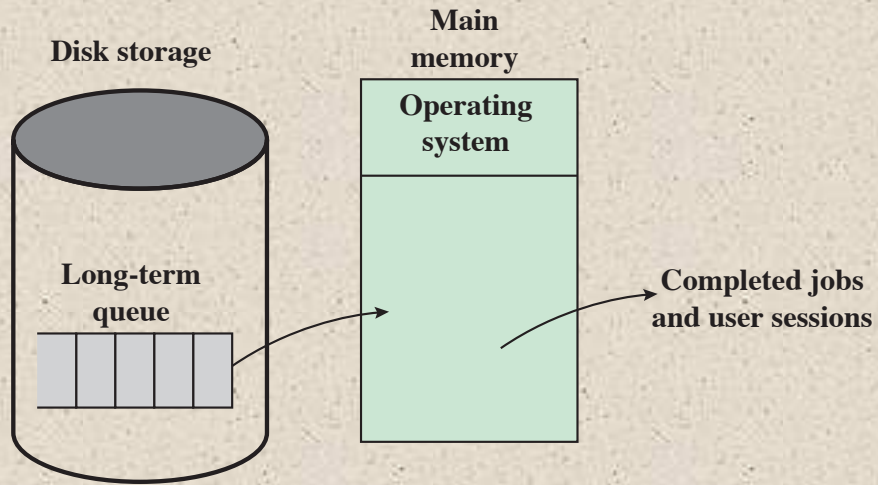
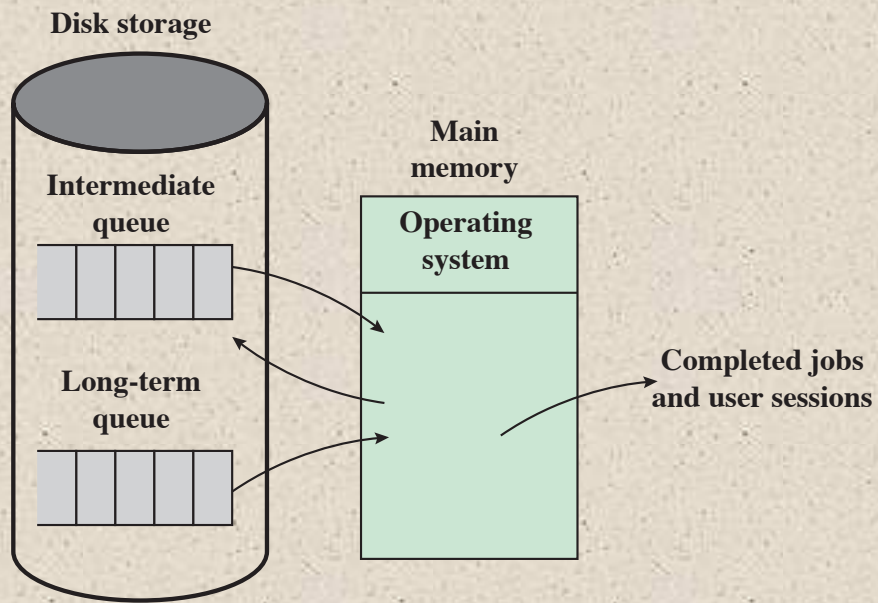


Figure 8.11 Queuing Diagram Representation of Processor Scheduling

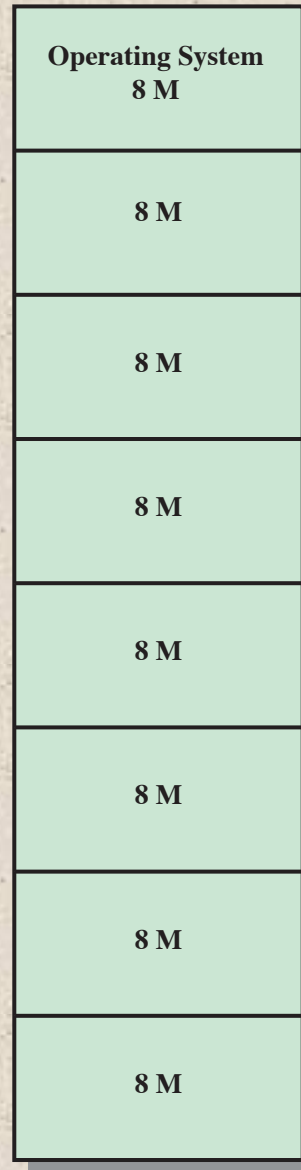


(a) Simple job scheduling

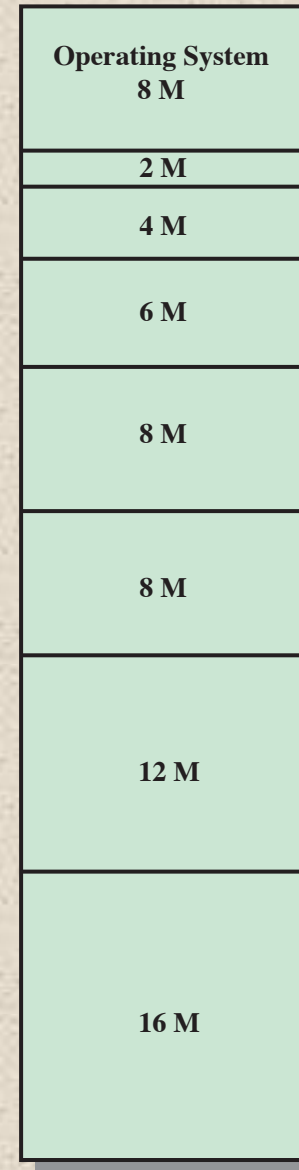


(b) Swapping

Figure 8.12 The Use of Swapping

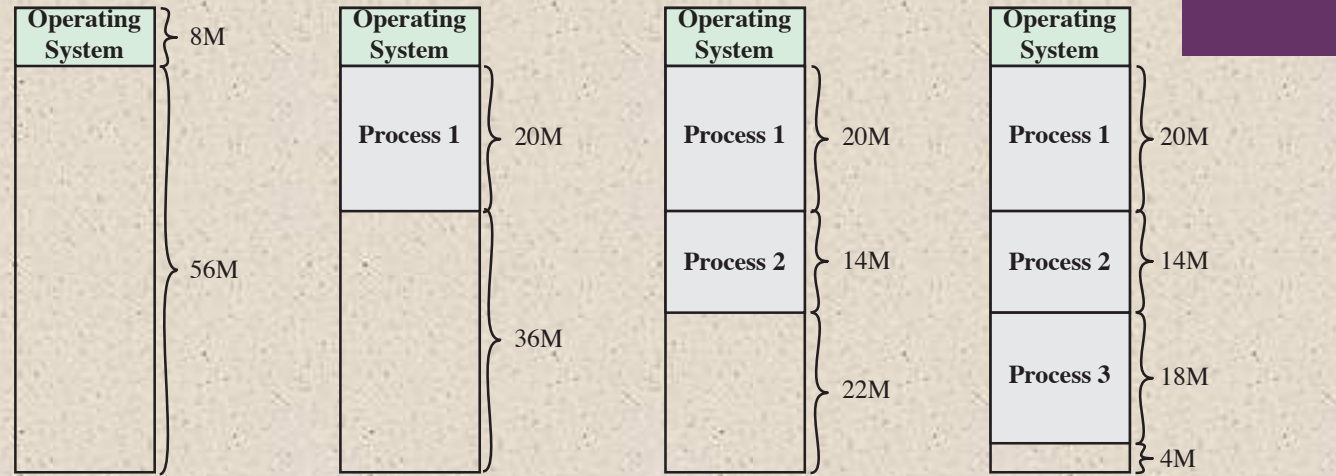


(a) Equal-size partitions



(b) Unequal-size partitions

Figure 8.13 Example of Fixed Partitioning of a 64-Mbyte Memory



(a)

(b)

(c)

(d)

Logical address

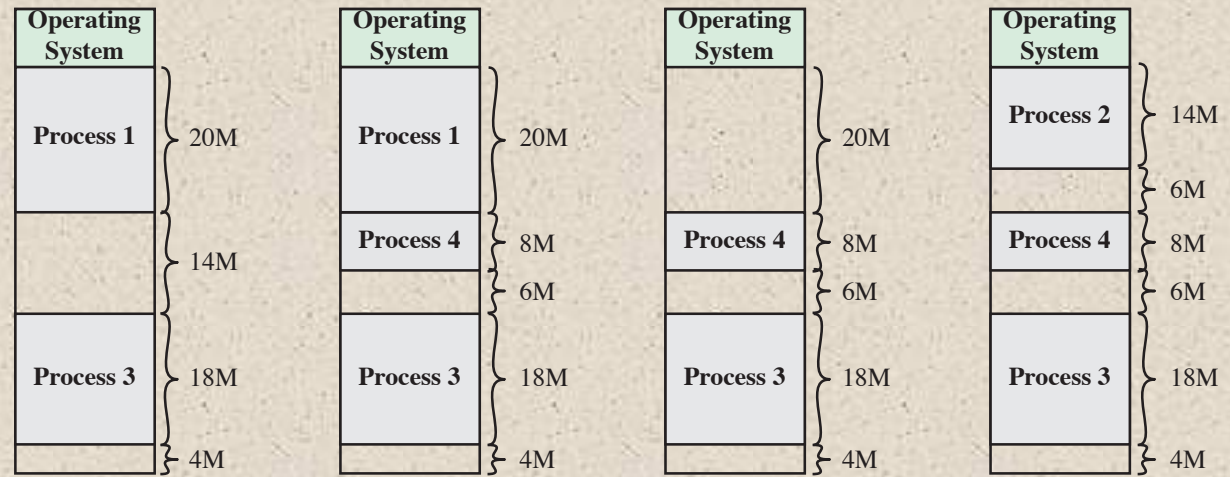
- expressed as a location relative to the beginning of the program

Physical address

- an actual location in main memory

Base address

- current starting location of the process



(e)

(f)

(g)

(h)

Figure 8.14 The Effect of Dynamic Partitioning

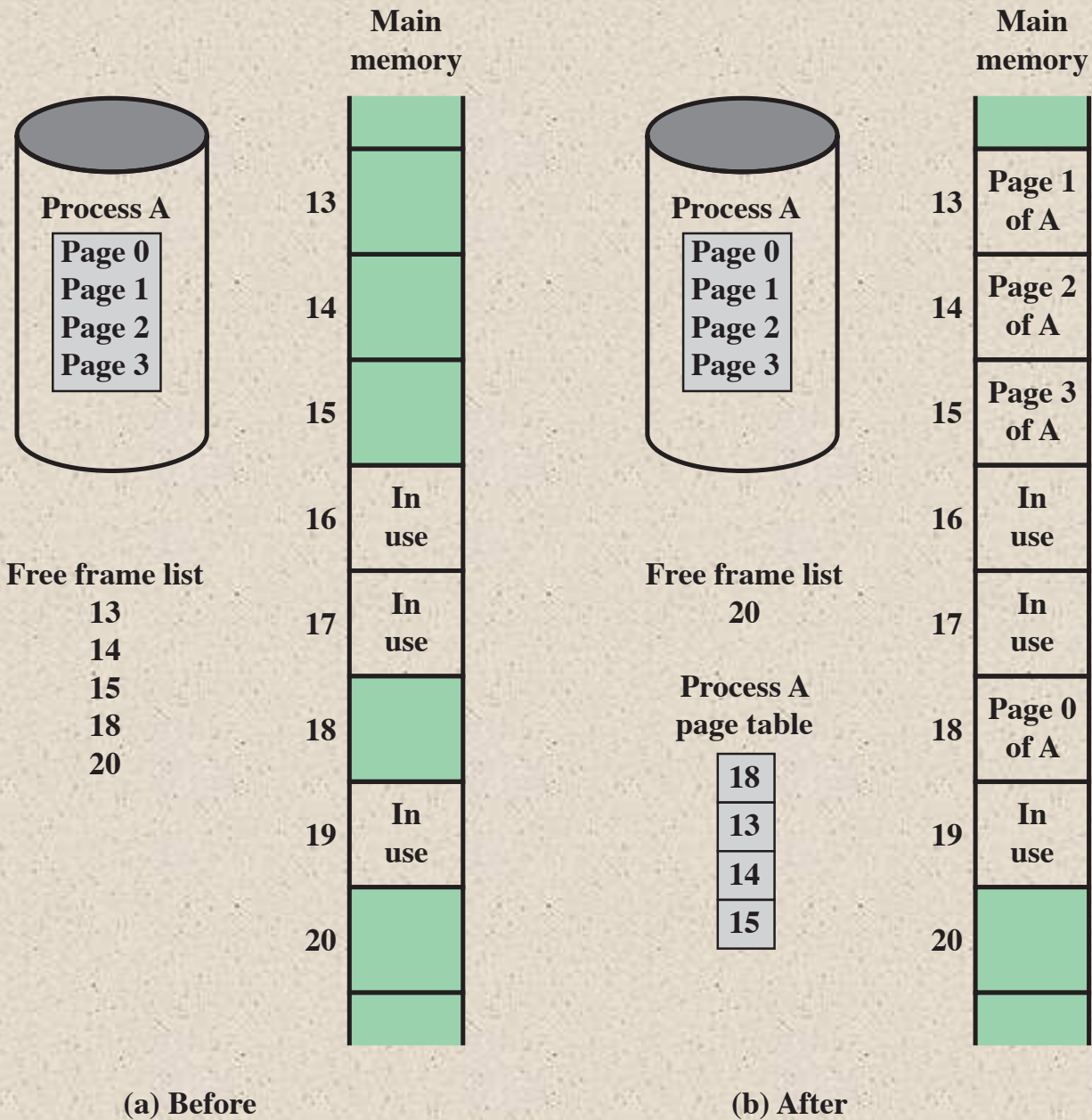


Figure 8.15 Allocation of Free Frames

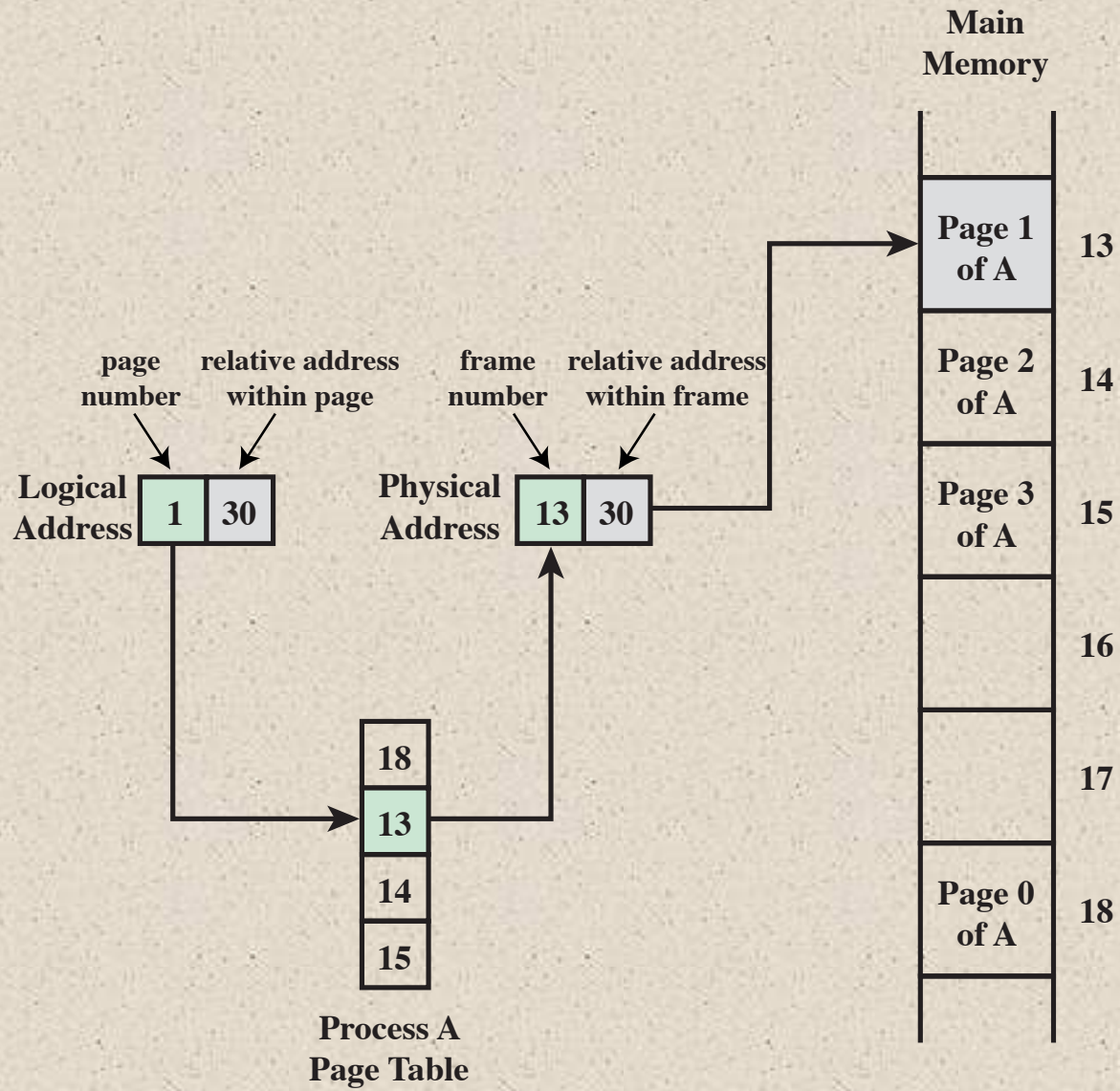


Figure 8.16 Logical and Physical Addresses

+ Virtual Memory

Demand Paging

- Each page of a process is brought in only when it is needed
- Principle of locality
 - When working with a large process execution may be confined to a small section of a program (subroutine)
 - It is better use of memory to load in just a few pages
 - If the program references data or branches to an instruction on a page not in main memory, a *page fault* is triggered which tells the OS to bring in the desired page
- Advantages:
 - More processes can be maintained in memory
 - Time is saved because unused pages are not swapped in and out of memory
- Disadvantages:
 - When one page is brought in, another page must be thrown out (*page replacement*)
 - If a page is thrown out just before it is about to be used the OS will have to go get the page again
 - *Thrashing*
 - When the processor spends most of its time swapping pages rather than executing instructions

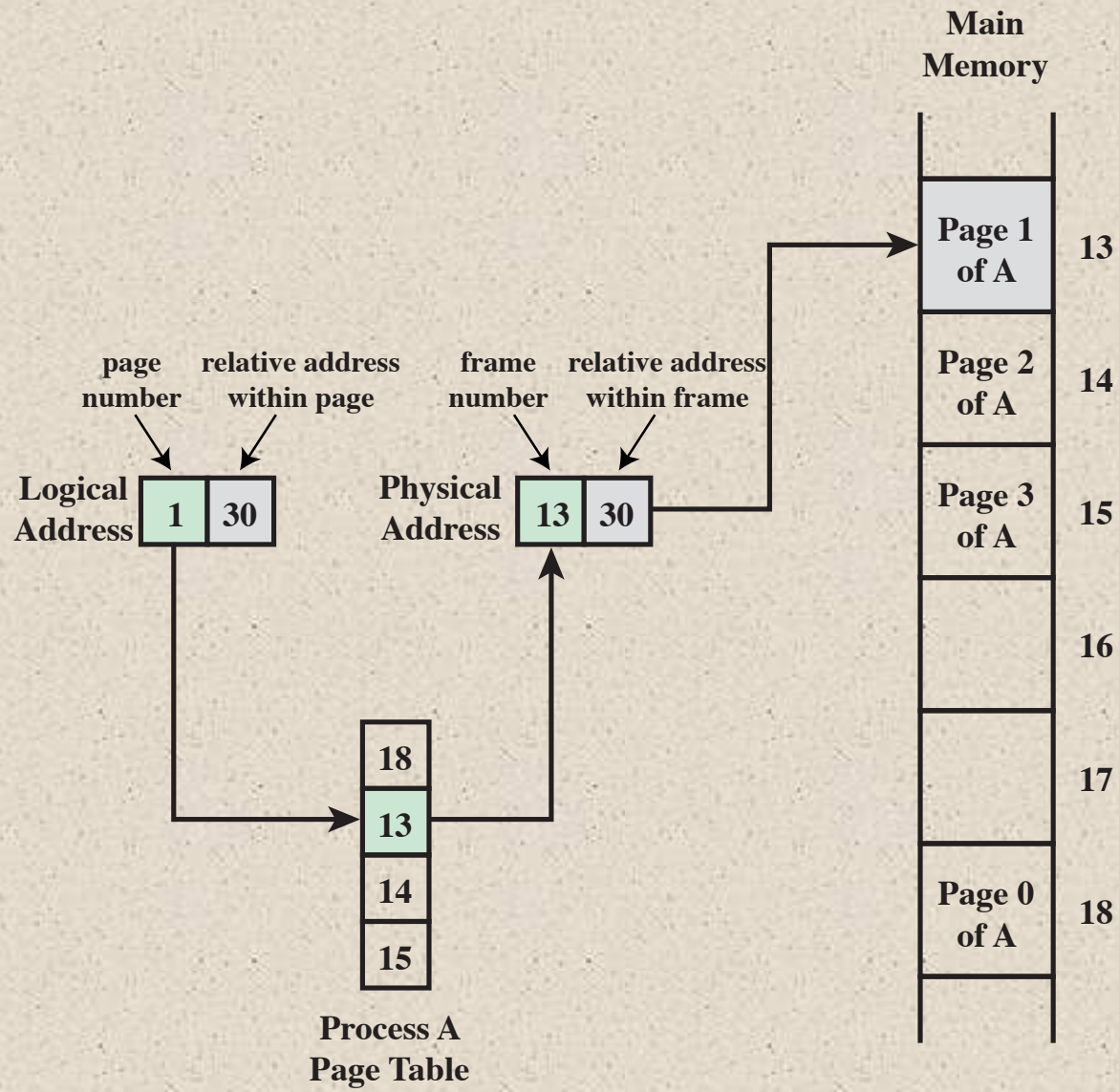


Figure 8.16 Logical and Physical Addresses

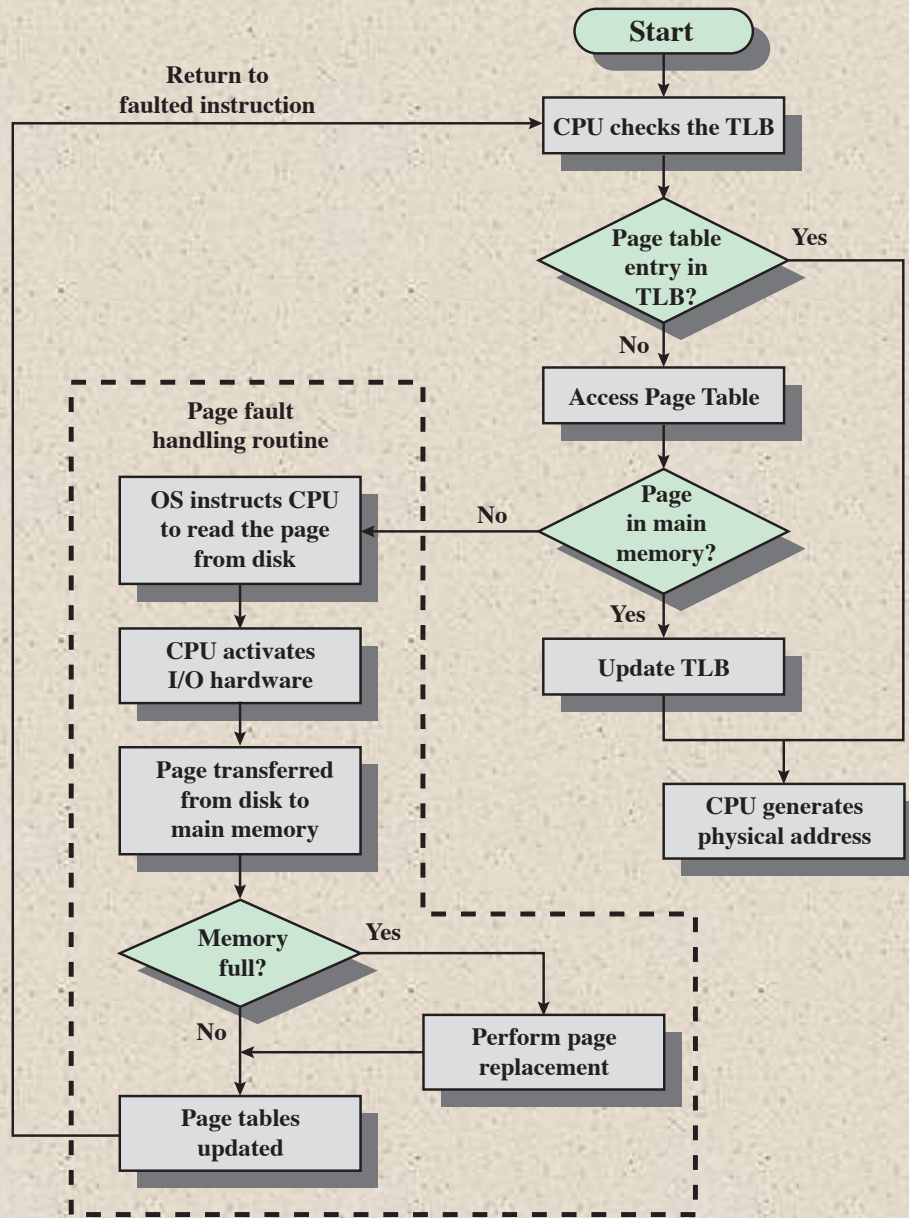


Figure 8.18 Operation of Paging and Translation Lookaside Buffer (TLB) [FURH87]

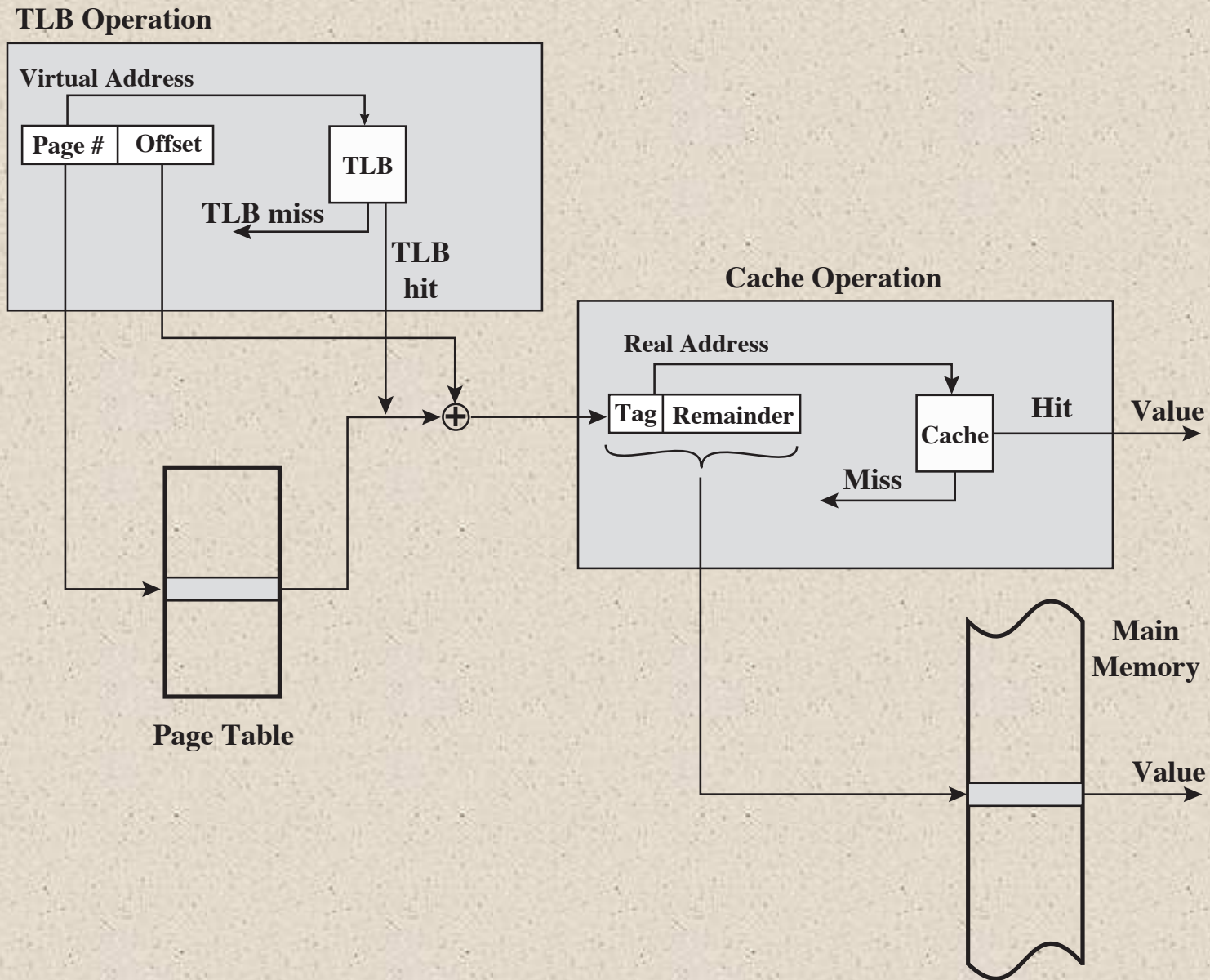


Figure 8.19 Translation Lookaside Buffer and Cache Operation



Segmentation



- Usually visible to the programmer
 - Provided as a convenience for organizing programs and data and as a means for associating privilege and protection attributes with instructions and data
 - Allows the programmer to view memory as consisting of multiple address spaces or segments
- Advantages:
 - Simplifies the handling of growing data structures
 - Allows programs to be altered and recompiled independently without requiring that an entire set of programs be re-linked and re-loaded
 - Lends itself to sharing among processes
 - Lends itself to protection

- Includes hardware for both segmentation and paging
- Unsegmented unpagged memory
 - Virtual address is the same as the physical address
 - Useful in low-complexity, high performance controller applications
- Unsegmented paged memory
 - Memory is viewed as a paged linear address space
 - Protection and management of memory is done via paging
 - Favored by some operating systems
- Segmented unpagged memory
 - Memory is viewed as a collection of logical address spaces
 - Affords protection down to the level of a single byte
 - Guarantees that the translation table needed is on-chip when the segment is in memory
 - Results in predictable access times
- Segmented paged memory
 - Segmentation is used to define logical memory partitions subject to access control, and paging is used to manage the allocation of memory within the partitions
 - Operating systems such as UNIX System V favor this view

Intel
x86



Memory
Management



Segmentation



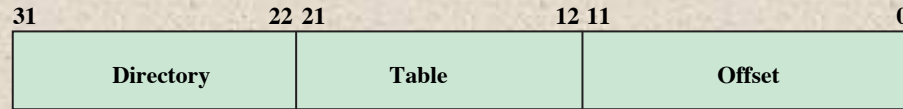
- Each virtual address consists of a 16-bit segment reference and a 32-bit offset
 - Two bits of segment reference deal with the protection mechanism
 - 14 bits specify segment
- Unsegmented virtual memory is $2^{32} = 4\text{Gbytes}$
- Segmented virtual memory is $2^{46} = 64\text{ terabytes (Tbytes)}$
- Physical address space employs a 32-bit address for a maximum of 4 Gbytes
- Virtual address space is divided into two parts
 - One-half is global, shared by all processors
 - The remainder is local and is distinct for each process

+ Segment Protection

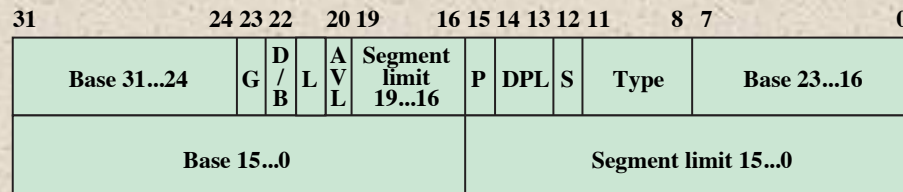
- Associated with each segment are two forms of protection:
 - Privilege level
 - Access attribute
- There are four privilege levels
 - Most protected (level 0)
 - Least protected (level 3)
- Privilege level associated with a data segment is its “classification”
- Privilege level associated with a program segment is its “clearance”
- An executing program may only access data segments for which its clearance level is lower than or equal to the privilege level of the data segment
- The privilege mechanism also limits the use of certain instructions



TI — Table indicator
 RPL — Requestor privilege level
 (a) Segment selector

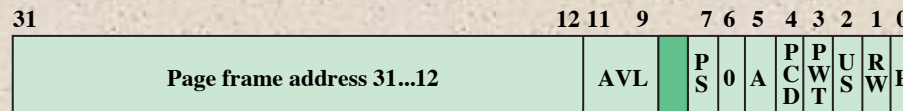


(b) Linear address



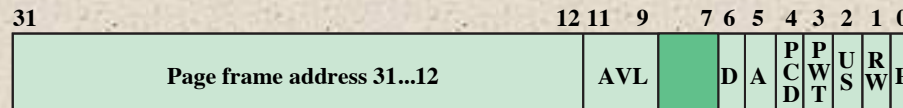
AVL — Available for use by system software L — 64-bit code segment (64-bit mode only)
 Base — Segment base address P — Segment present
 D/B — Default operation size Type — Segment type
 DPL — Descriptor privilege size S — Descriptor type
 G — Granularity

(c) Segment descriptor (segment table entry)



AVL — Available for systems programmer use PWT — Write through = reserved
 P — Page size US — User/supervisor
 A — Accessed RW — Read-write
 PCD — Cache disable P — Present

(d) Page directory entry



D — Dirty

(e) Page table entry

Figure 8.20 Intel x86 Memory Management Formats

Table 8.5

x86 Memory Management Parameters (page 1 of 2)

Segment Descriptor (Segment Table Entry)

Base

Defines the starting address of the segment within the 4-GByte linear address space.

D/B bit

In a code segment, this is the D bit and indicates whether operands and addressing modes are 16 or 32 bits.

Descriptor Privilege Level (DPL)

Specifies the privilege level of the segment referred to by this segment descriptor.

Granularity bit (G)

Indicates whether the Limit field is to be interpreted in units by one byte or 4 KBytes.

Limit

Defines the size of the segment. The processor interprets the limit field in one of two ways, depending on the granularity bit: in units of one byte, up to a segment size limit of 1 MByte, or in units of 4 KBytes, up to a segment size limit of 4 GBytes.

S bit

Determines whether a given segment is a system segment or a code or data segment.

Segment Present bit (P)

Used for nonpaged systems. It indicates whether the segment is present in main memory. For paged systems, this bit is always set to 1.

Type

Distinguishes between various kinds of segments and indicates the access attributes.

Table 8.5

x86 Memory Management Parameters (page 2 of 2)



Page Directory Entry and Page Table Entry

Accessed bit (A)

This bit is set to 1 by the processor in both levels of page tables when a read or write operation to the corresponding page occurs.

Dirty bit (D)

This bit is set to 1 by the processor when a write operation to the corresponding page occurs.

Page Frame Address

Provides the physical address of the page in memory if the present bit is set. Since page frames are aligned on 4K boundaries, the bottom 12 bits are 0, and only the top 20 bits are included in the entry. In a page directory, the address is that of a page table.

Page Cache Disable bit (PCD)

Indicates whether data from page may be cached.

Page Size bit (PS)

Indicates whether page size is 4 KByte or 4 MByte.

Page Write Through bit (PWT)

Indicates whether write-through or write-back caching policy will be used for data in the corresponding page.

Present bit (P)

Indicates whether the page table or page is in main memory.

Read/Write bit (RW)

For user-level pages, indicates whether the page is read-only access or read/write access for user-level programs.

User/Supervisor bit (US)

Indicates whether the page is available only to the operating system (supervisor level) or is available to both operating system and applications (user level).

+ 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 4 Gbyte linear memory into 1024 page groups of 4 Mbyte
 - Each page table has 1024 entries corresponding to 4 Kbyte 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

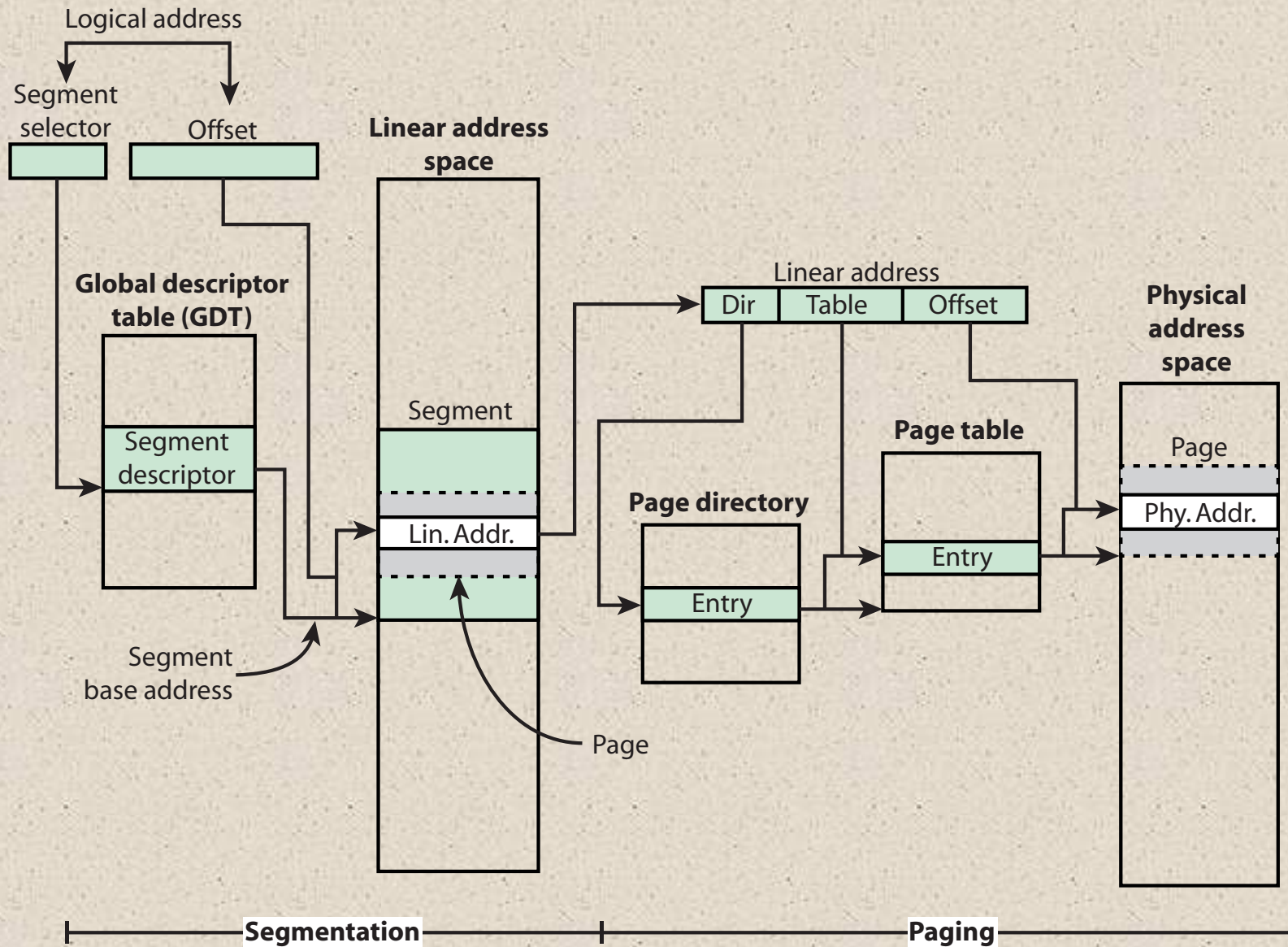


Figure 8.21 Intel x86 Memory Address Translation Mechanisms

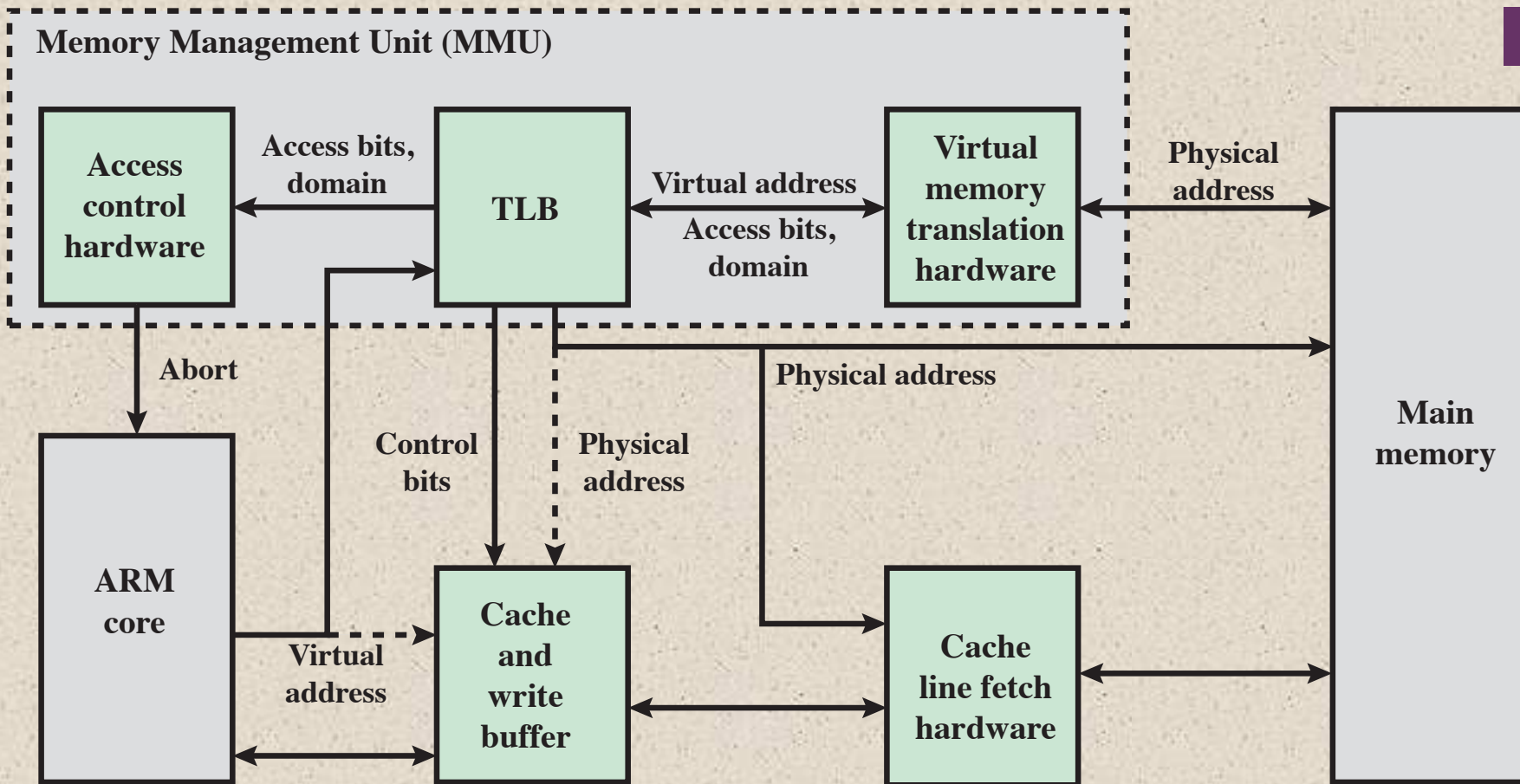


Figure 8.22 ARM Memory System Overview

+ Virtual Memory Address Translation

- The ARM supports memory access based on either sections or pages
- Supersections (optional)
 - Consist of 16-MB blocks of main memory
- Sections
 - Consist of 1-MB blocks of main memory
- Large pages
 - Consist of 64-kB blocks of main memory
- Small pages
 - Consist of 4-kB blocks of main memory
- Sections and supersections are supported to allow mapping of a large region of memory while using only a single entry in the TLB
- The translation table held in main memory has two levels:
 - First-level table
 - Holds section and supersection translations, and pointers to second-level table
 - Second-level tables
 - Hold both large and small page translations

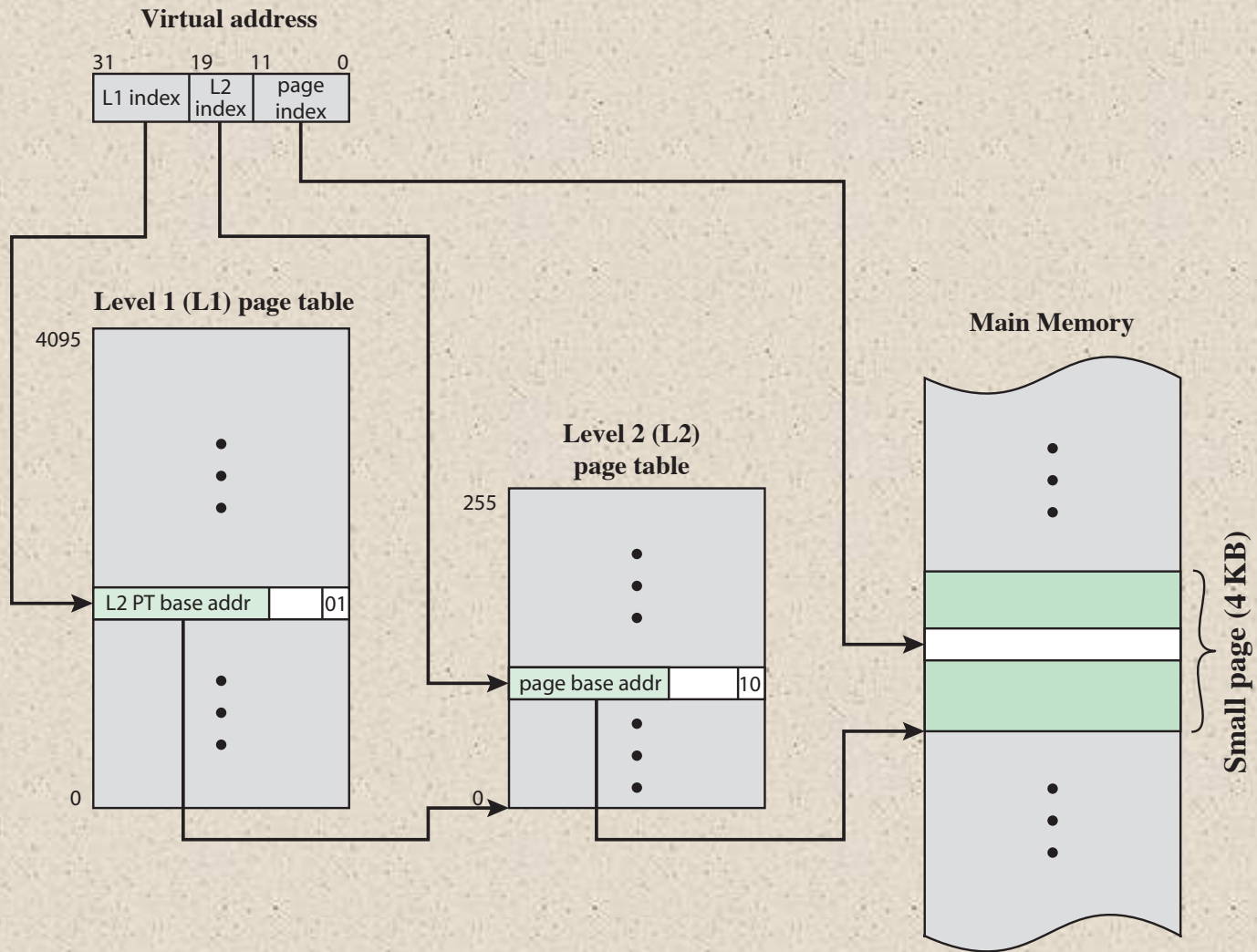
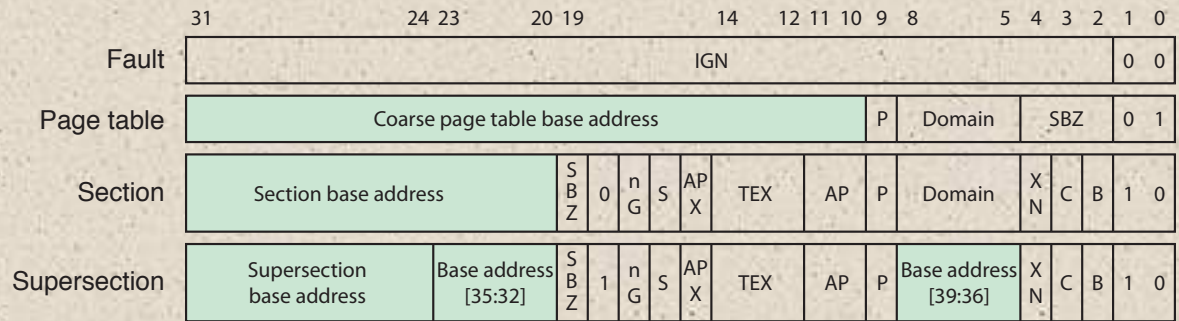
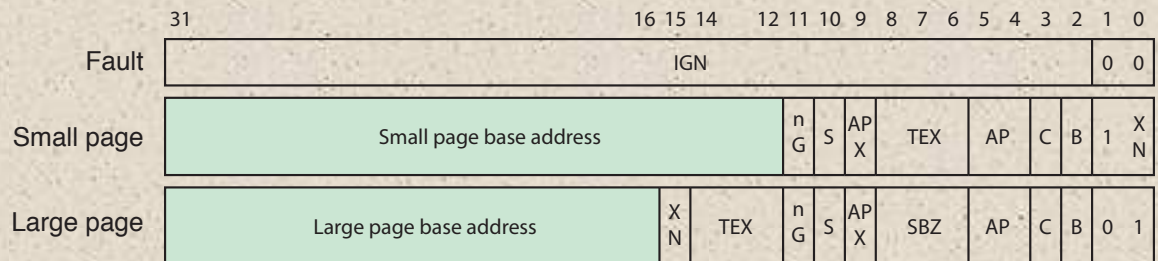


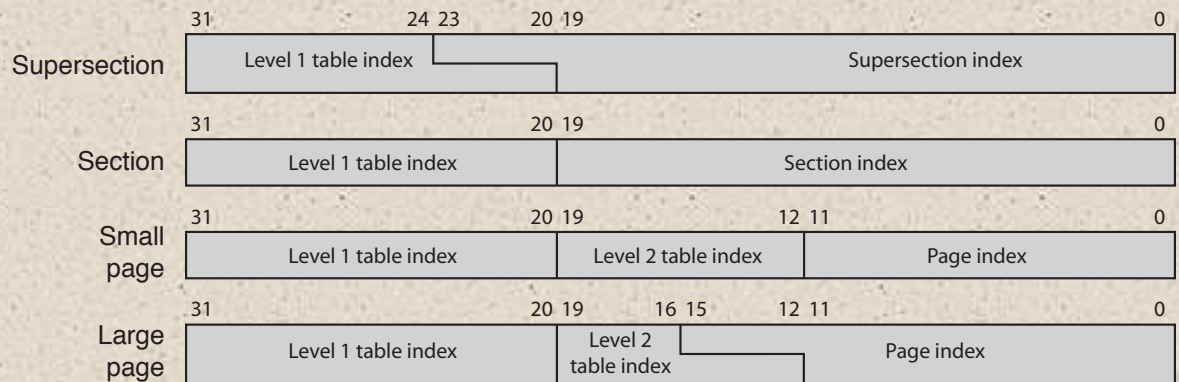
Figure 8.23 ARM Virtual Memory Address Translation for Small Pages



(a) Alternative first-level descriptor formats



(b) Alternative second-level descriptor formats



(c) Virtual memory address formats

Figure 8.24 ARM Memory Management Formats

Table 8.6 ARM Memory-Management Parameters

Access Permission (AP), Access Permission Extension (APX)

These bits control access to the corresponding memory region. If an access is made to an area of memory without the required permissions, a Permission Fault is raised.

Bufferable (B) bit

Determines, with the TEX bits, how the write buffer is used for cacheable memory.

Cacheable (C) bit

Determines whether this memory region can be mapped through the cache.

Domain

Collection of memory regions. Access control can be applied on the basis of domain.

not Global (nG)

Determines whether the translation should be marked as global (0), or process specific (1).

Shared (S)

Determines whether the translation is for not-shared (0), or shared (1) memory.

SBZ

Should be zero.

Type Extension (TEX)

These bits, together with the B and C bits, control accesses to the caches, how the write buffer is used, and if the memory region is shareable and therefore must be kept coherent.

Execute Never (XN)

Determines whether the region is executable (0) or not executable (1).

+ Access Control

- The AP access control bits in each table entry control access to a region of memory by a given process
- A region of memory can be designated as:
 - No access
 - Read only
 - Read-write
- The region can be privileged access only, reserved for use by the OS and not by applications
- ARM employs the concept of a domain:
 - Collection of sections and/or pages that have particular access permissions
 - The ARM architecture supports 16 domains
 - Allows multiple processes to use the same translation tables while maintaining some protection from each other
- Two kinds of domain access are supported:
 - Clients
 - Users of domains that must observe the access permissions of the individual sections and/or pages that make up that domain
 - Managers
 - Control the behavior of the domain and bypass the access permissions for table entries in that domain

+ Summary

Chapter 8

- Operating system overview
 - Operating system objectives and functions
 - Types of operating systems
- Scheduling
 - Long-term scheduling
 - Medium-term scheduling
 - Short-term scheduling
- Intel x86 memory management
 - Address space
 - Segmentation
 - paging

Operating System Support

- Memory management
 - Swapping
 - Partitioning
 - Paging
 - Virtual memory
 - Translation lookaside buffer
 - Segmentation
- ARM memory management
 - Memory system organization
 - Virtual memory address translation
 - Memory-management formats
 - Access control