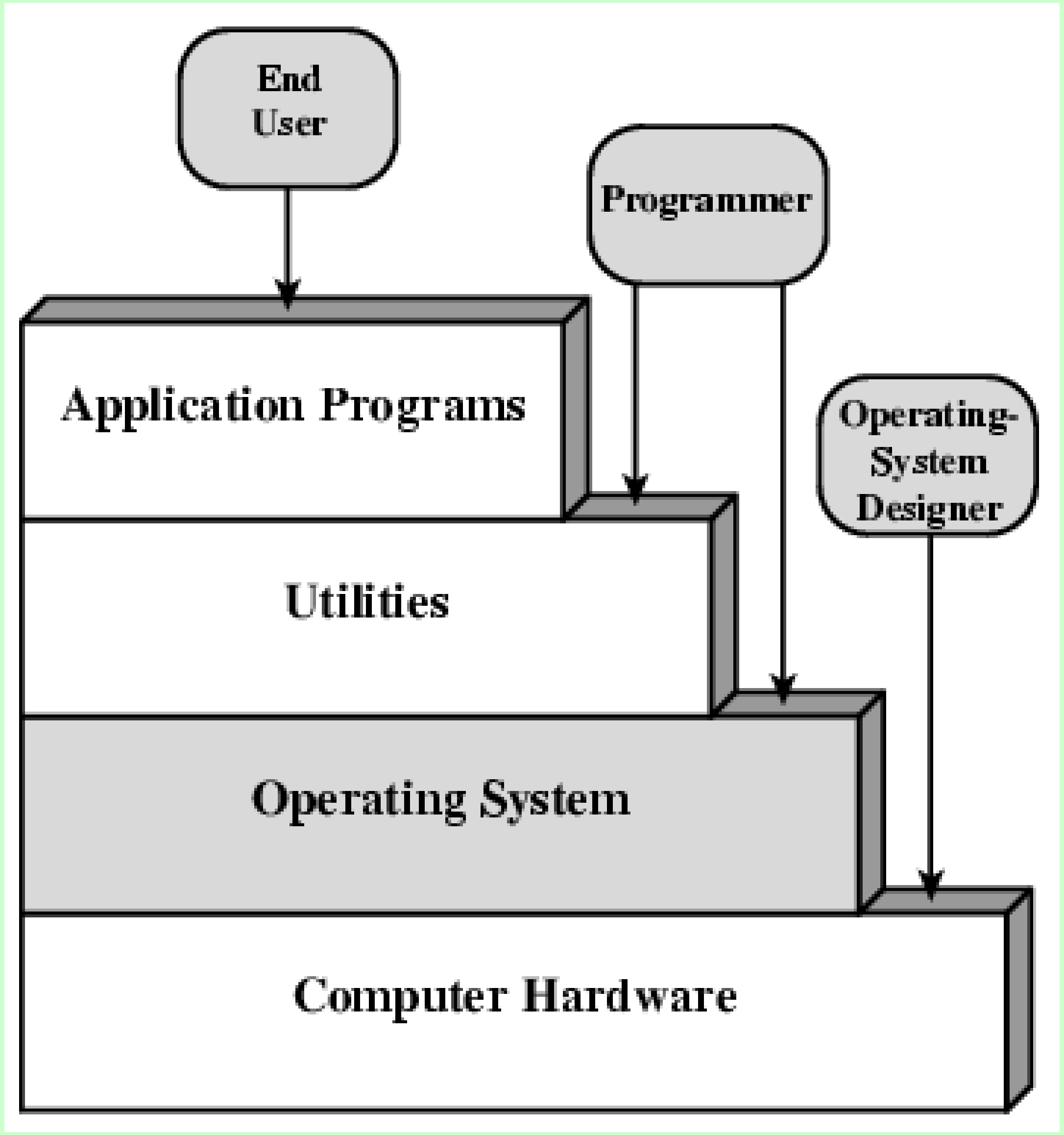
William Stallings
Computer Organization and Architecture
8th Edition

Chapter 8
Operating System Support

Objectives and Functions

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

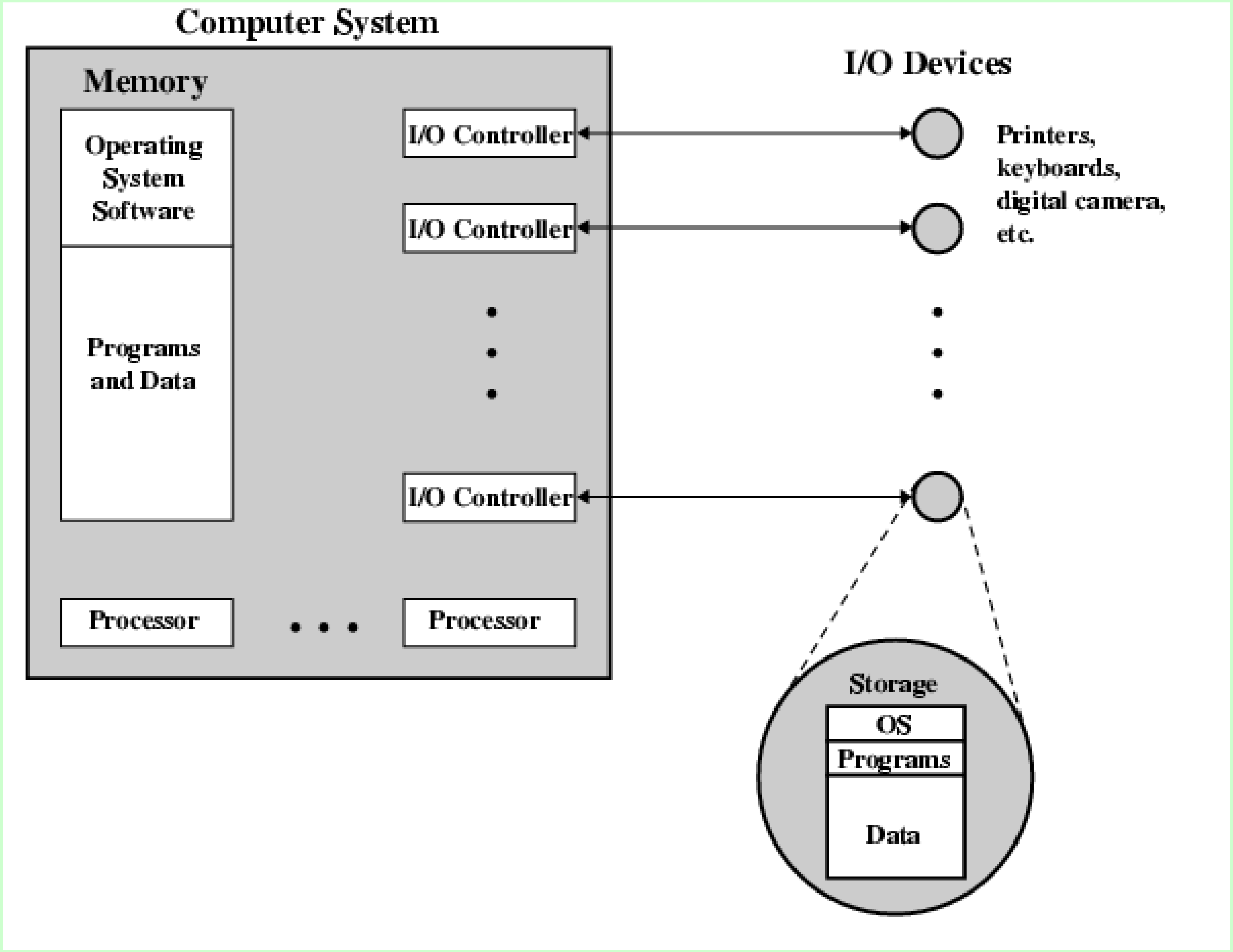
Layers and Views of a Computer System



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)

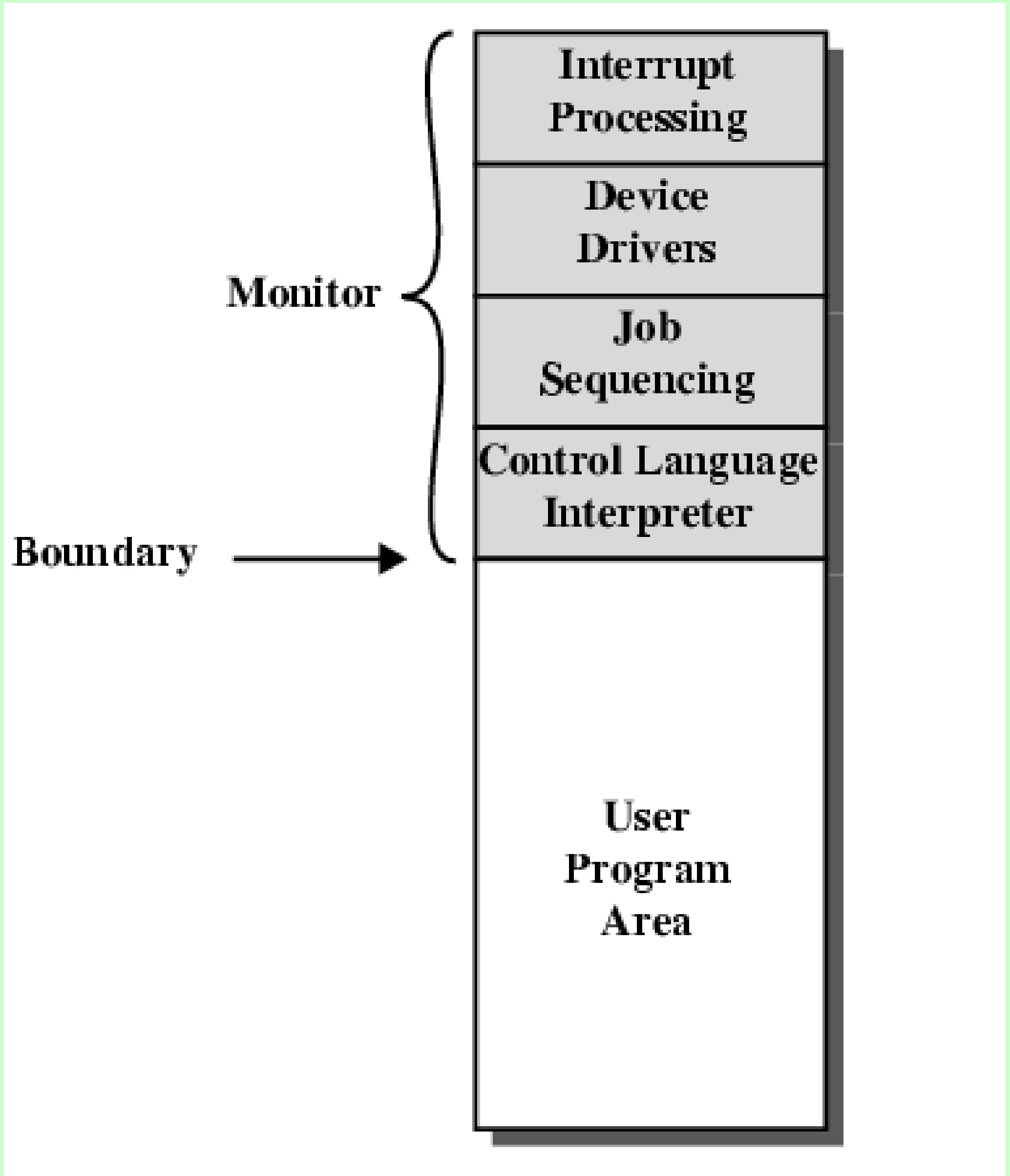
Early Systems

- 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

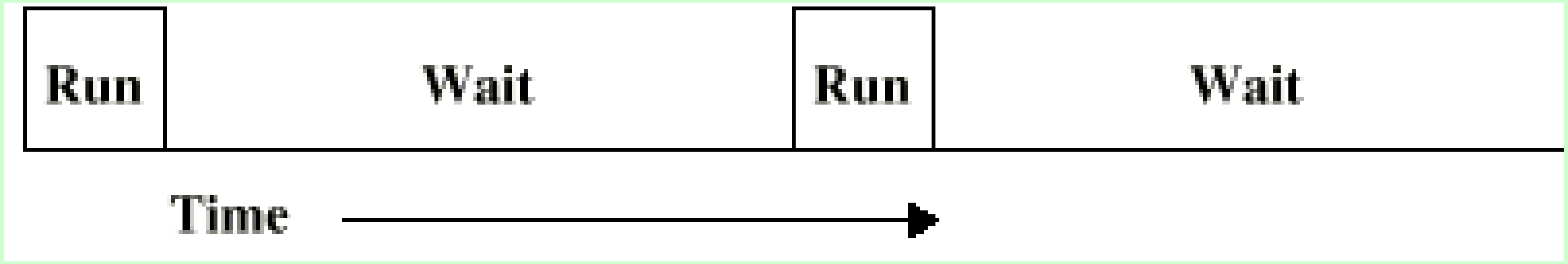
Desirable Hardware Features

- 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

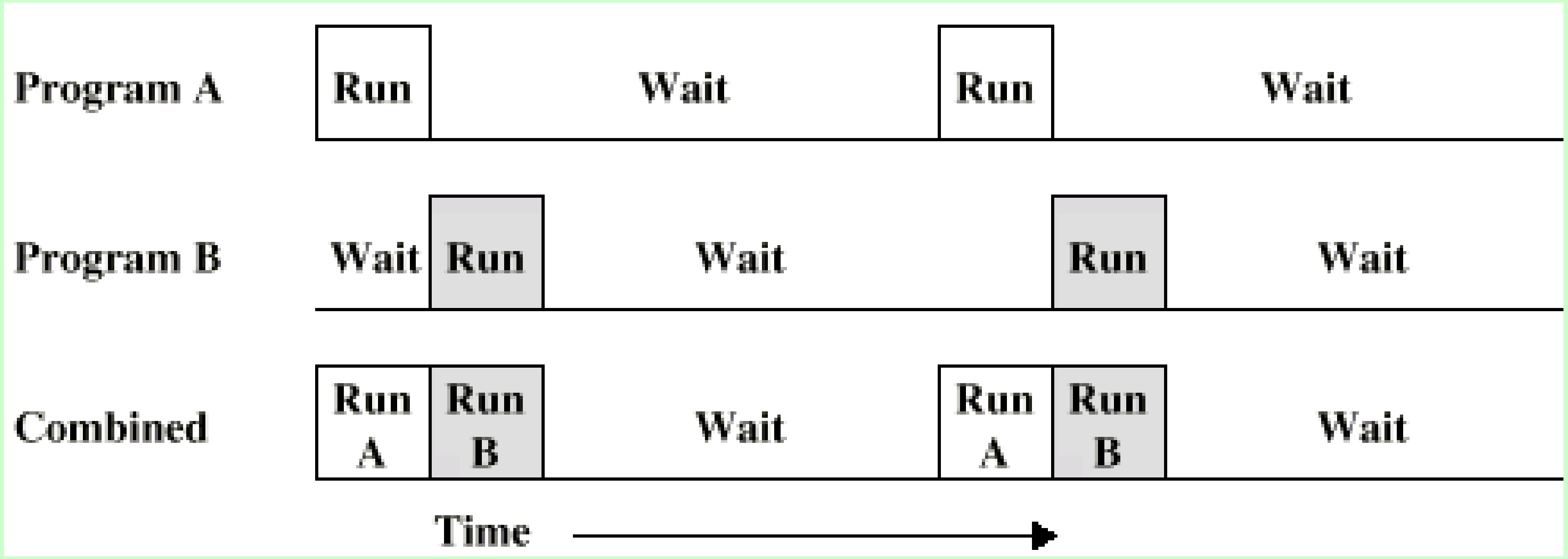
Multi-programmed Batch Systems

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

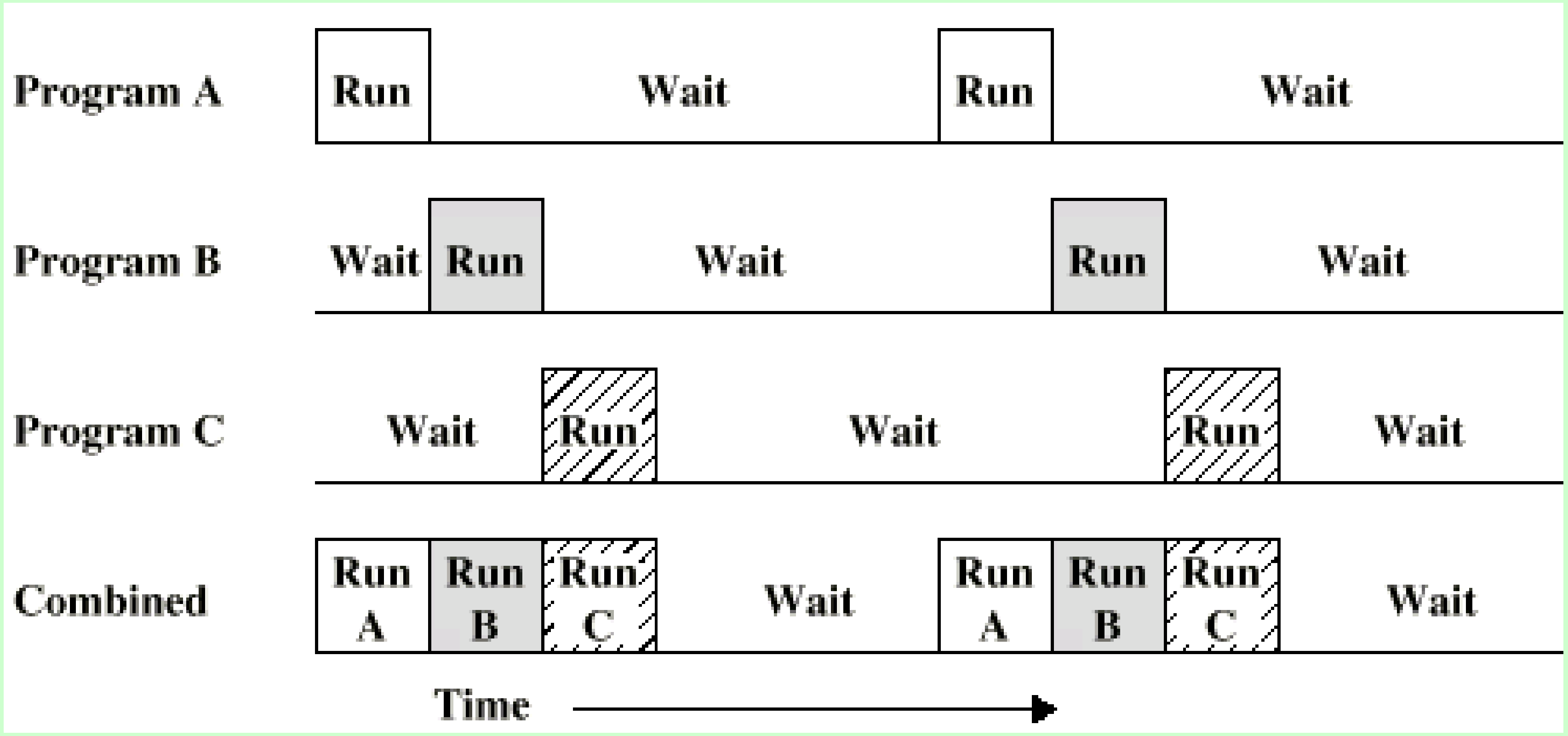
Single Program



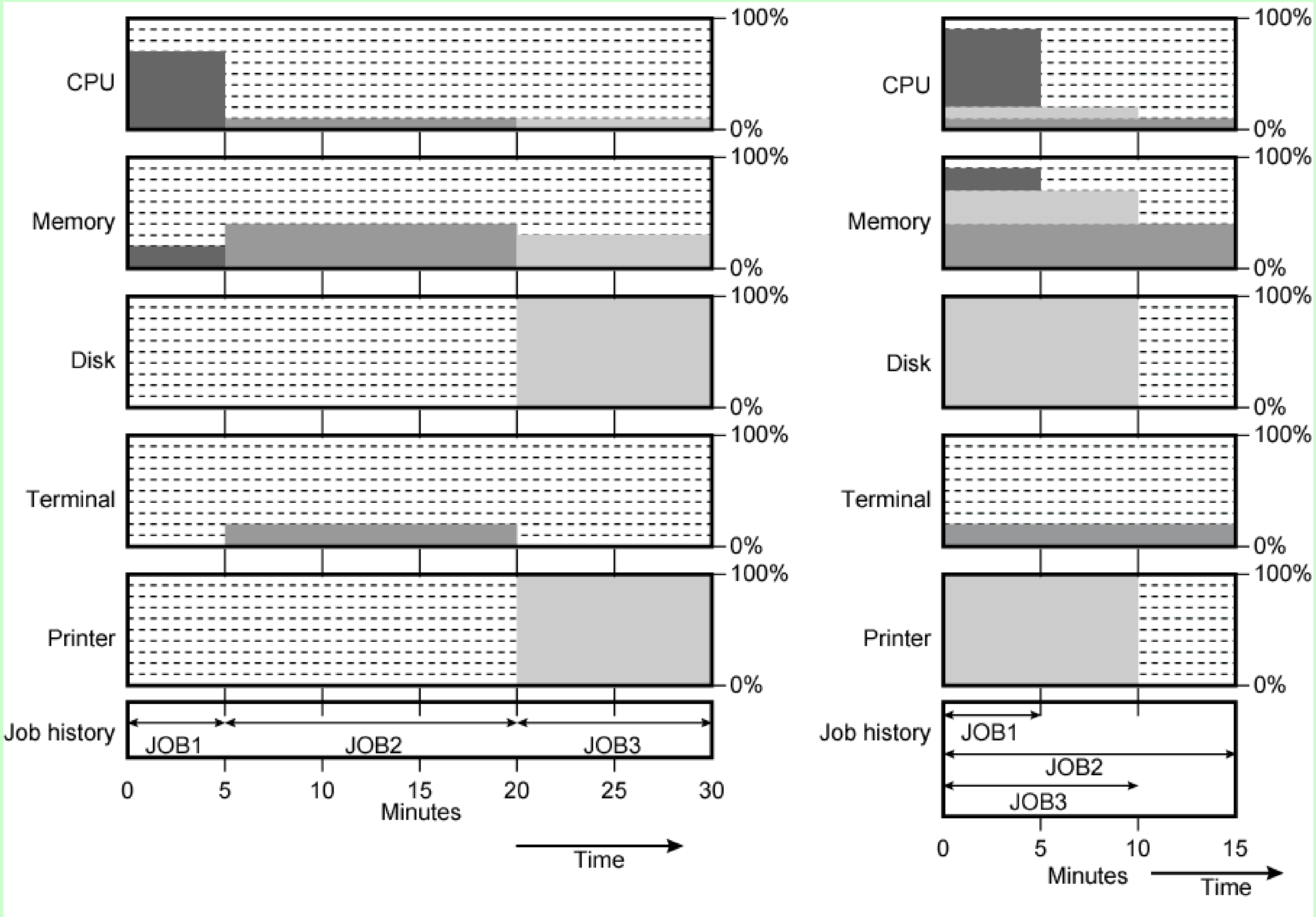
Multi-Programming with Two Programs



Multi-Programming with Three Programs



Utilization



(a) Uniprogramming

(b) Multiprogramming

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)

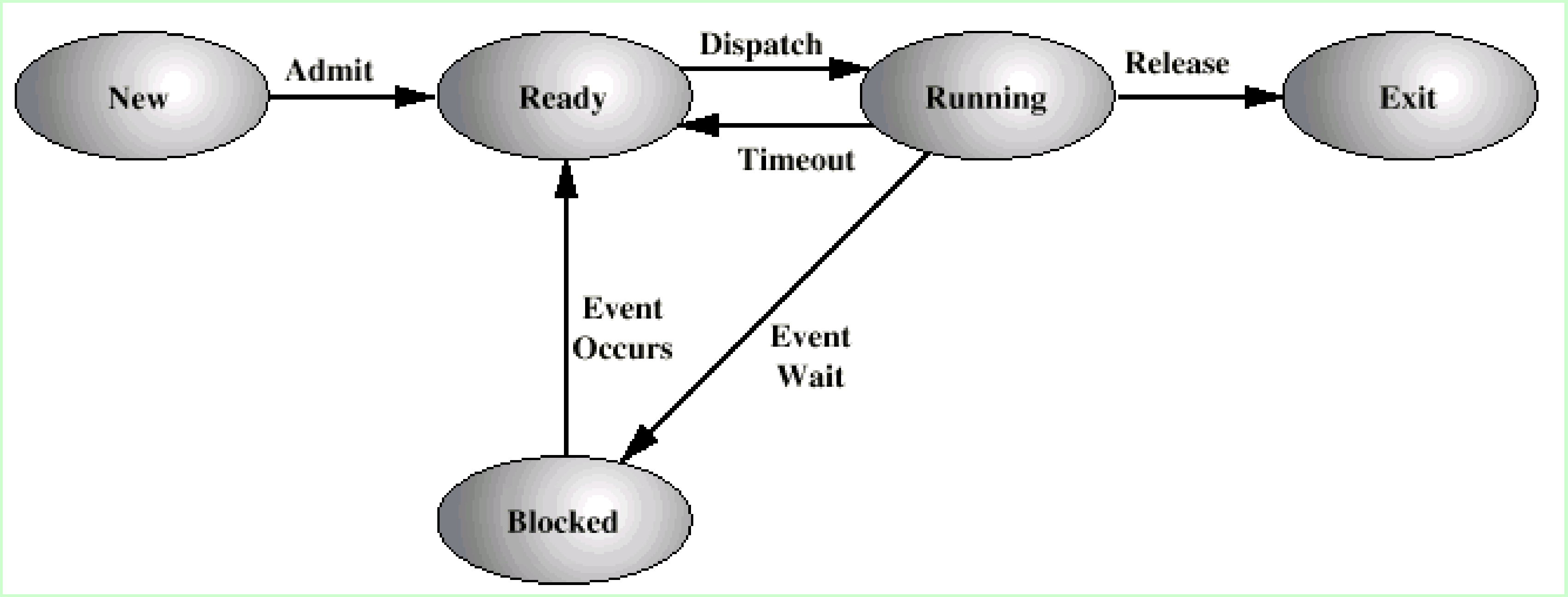
Medium Term Scheduling

- Part of the swapping function (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

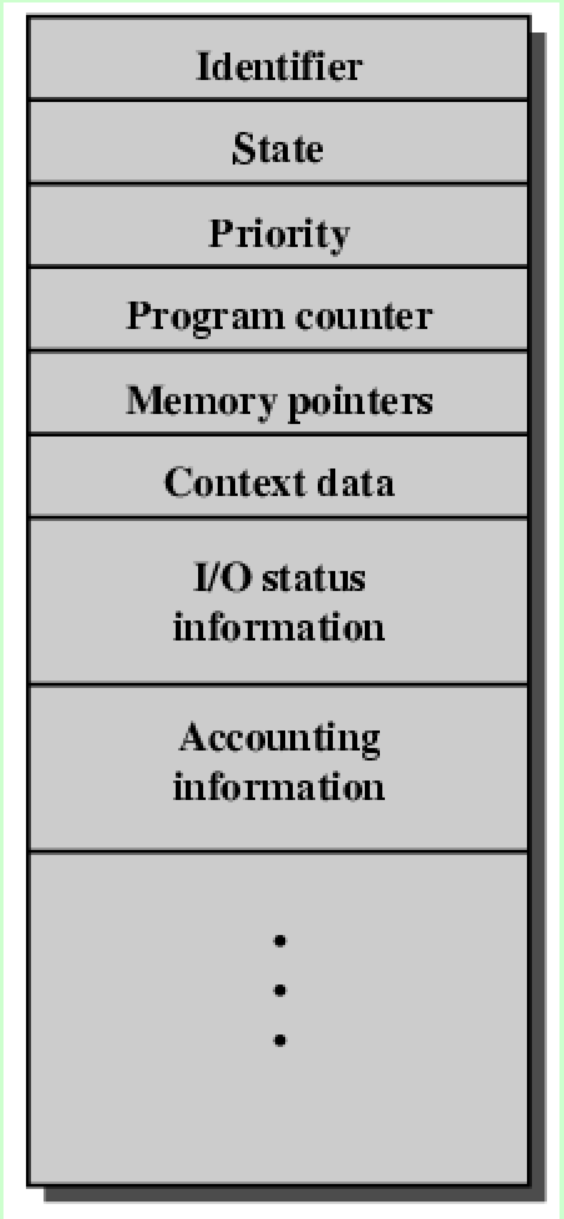
Five State Process Model



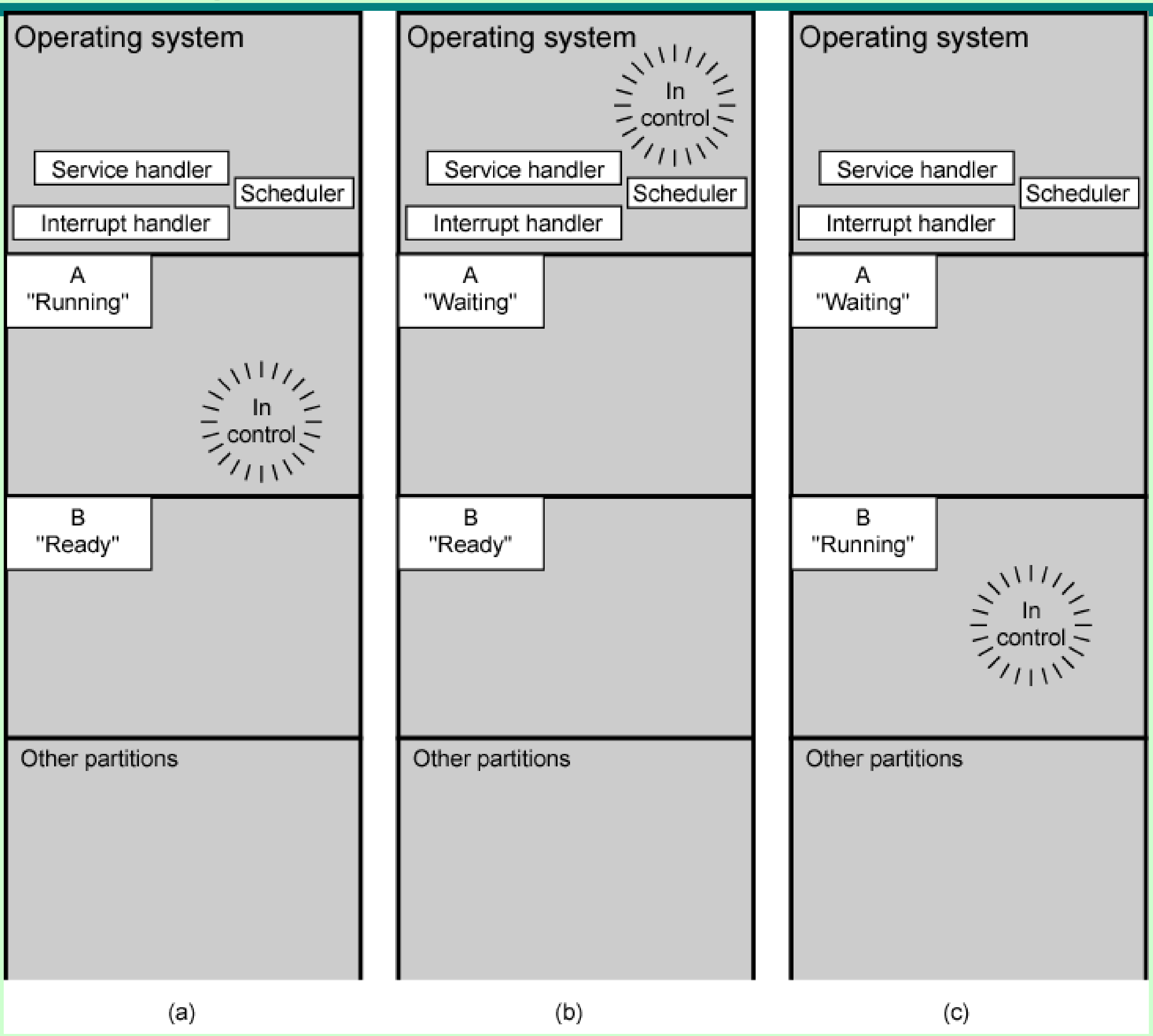
Process Control Block

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

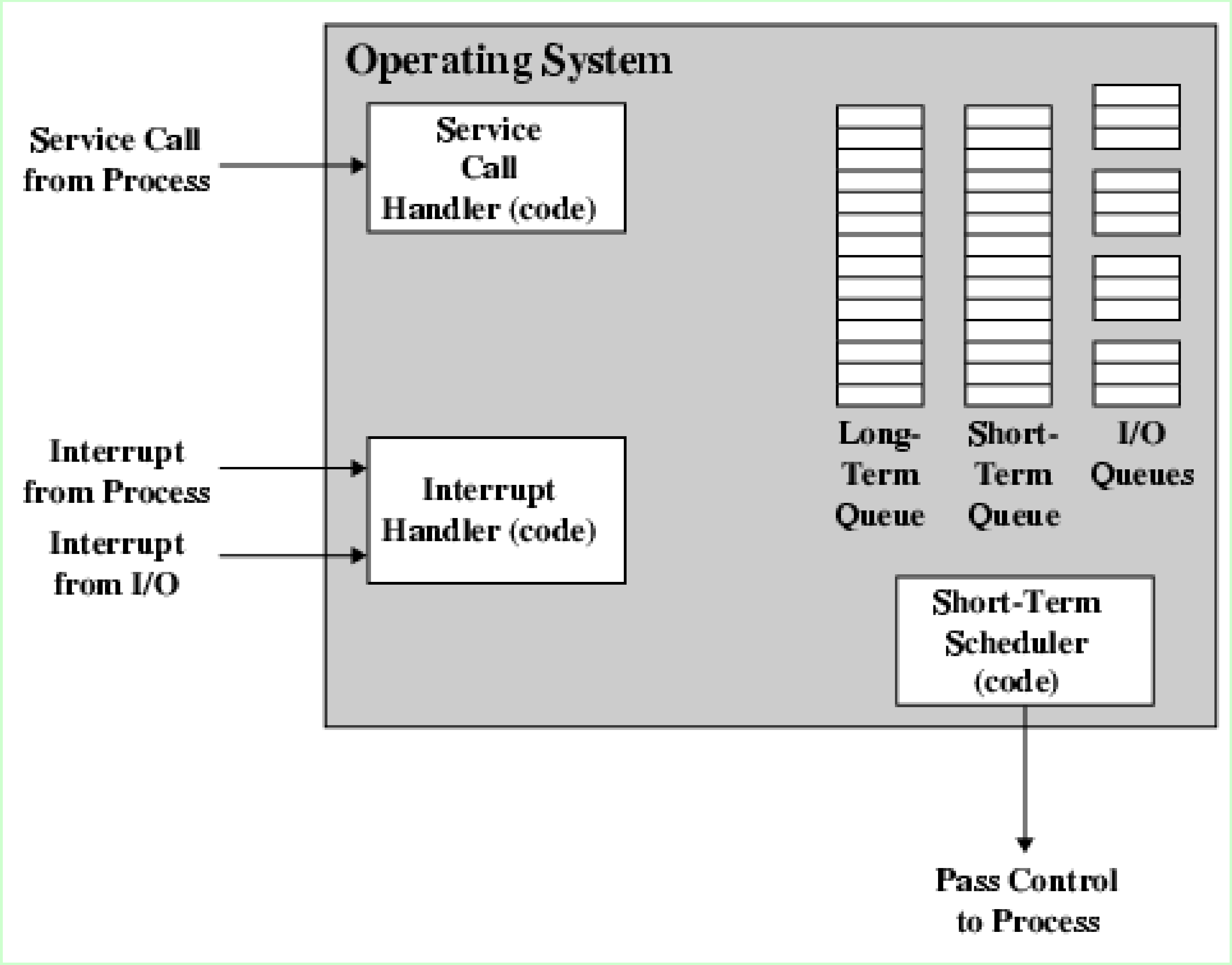
PCB Diagram


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

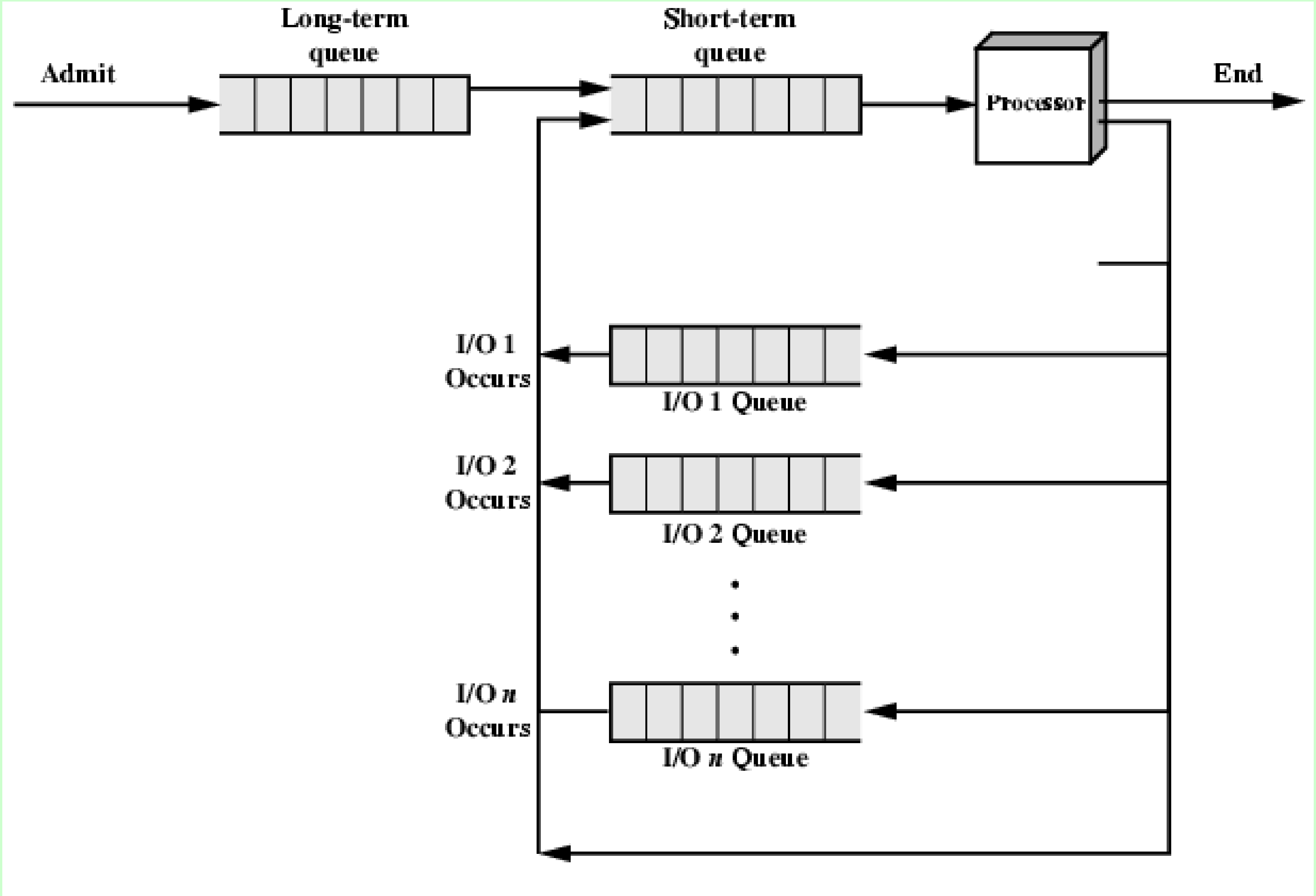
Scheduling Example



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

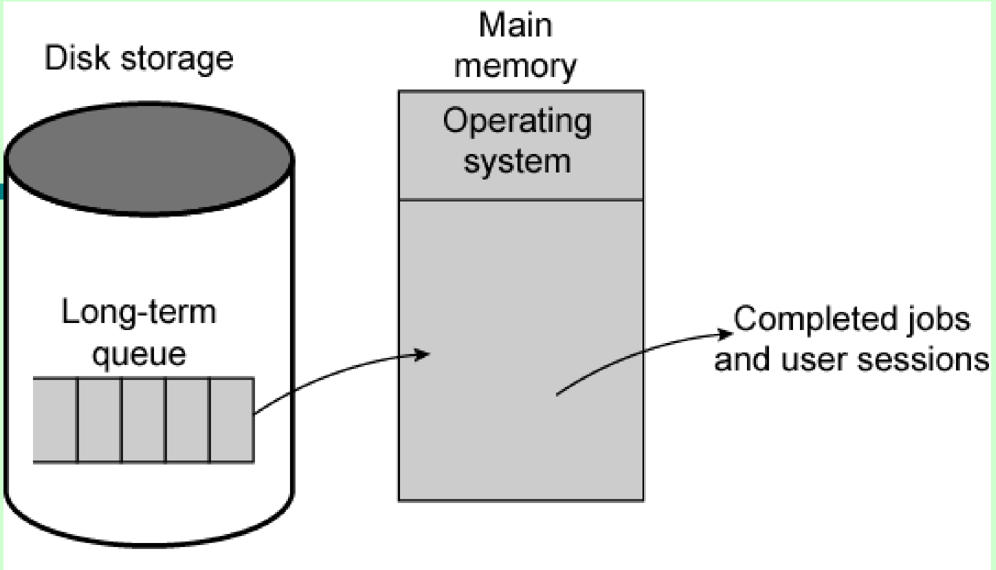
Swapping

- 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

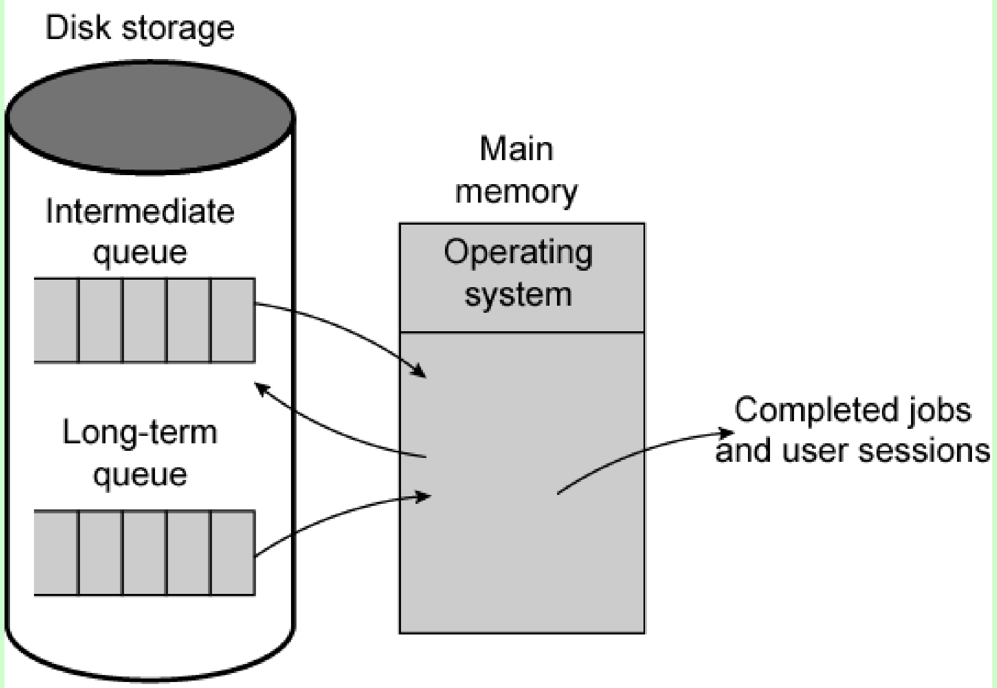
What is Swapping?

- 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…

Use of Swapping



(a) Simple job scheduling

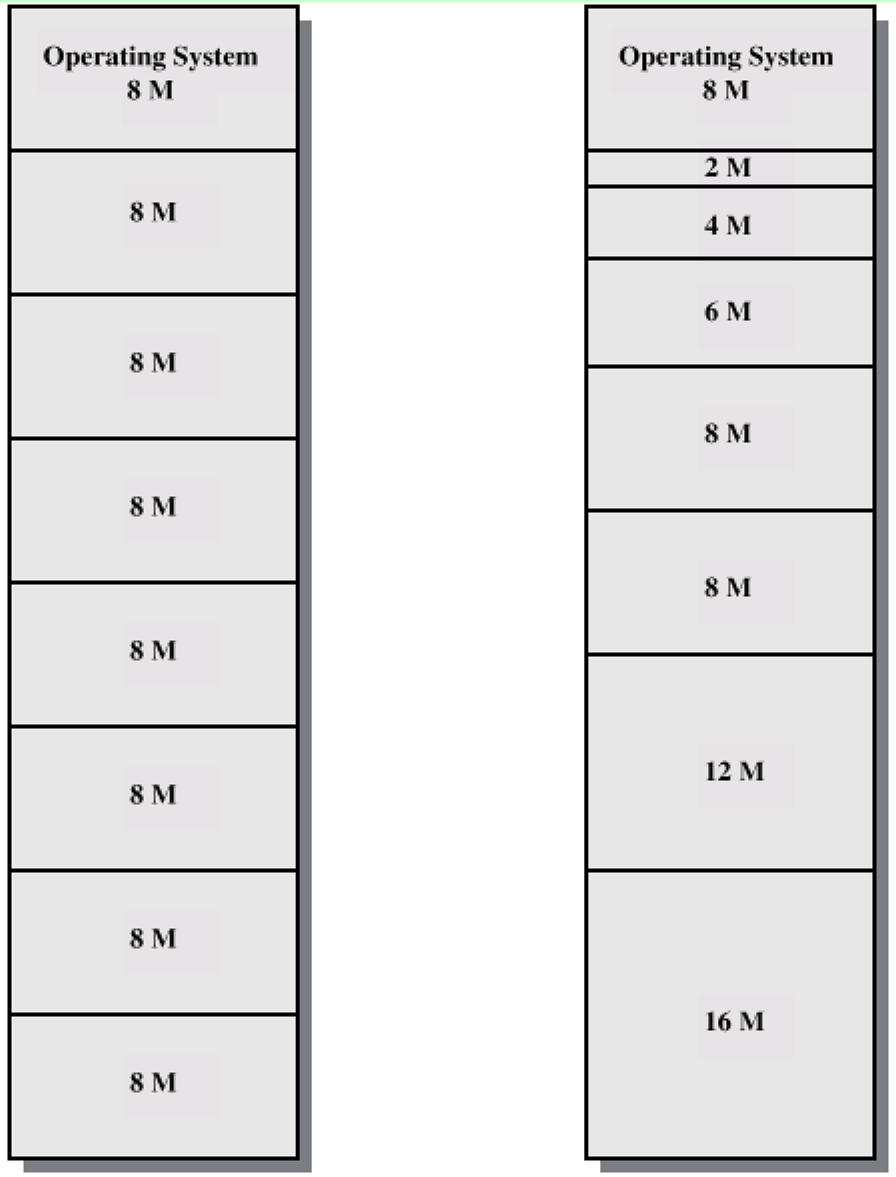


(b) Swapping

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

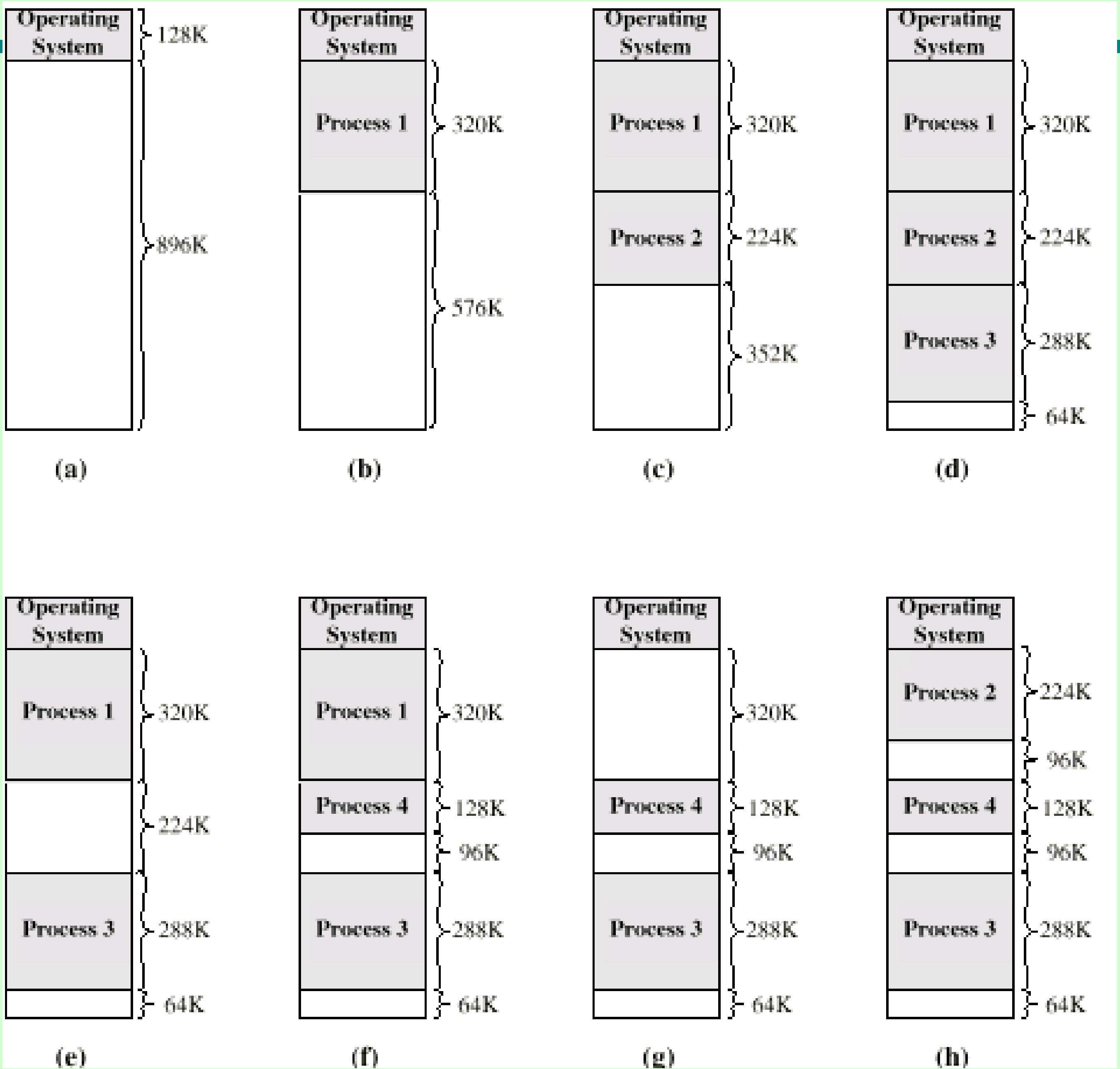
Variable Sized Partitions (1)

- 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



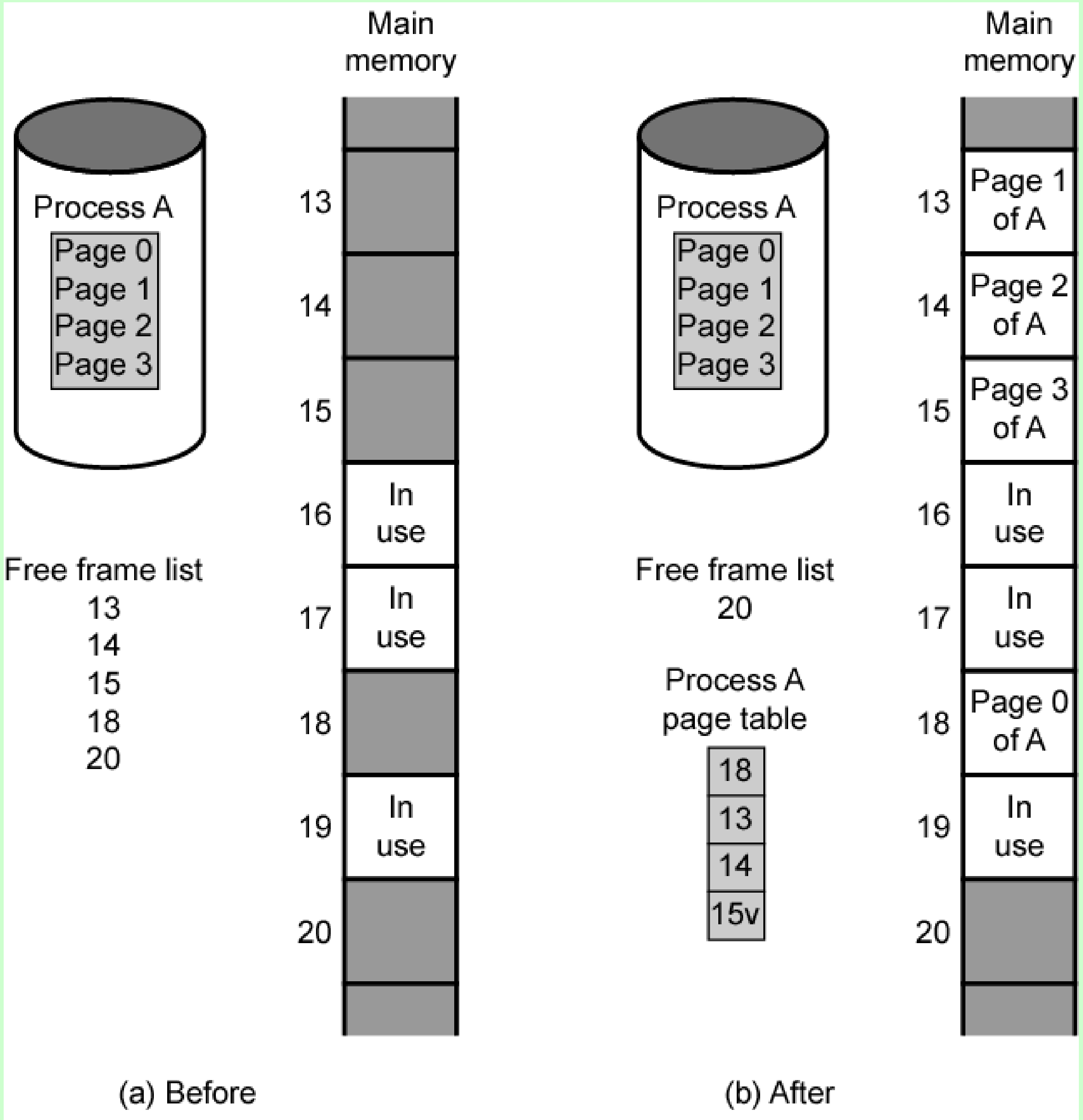
Relocation

- 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

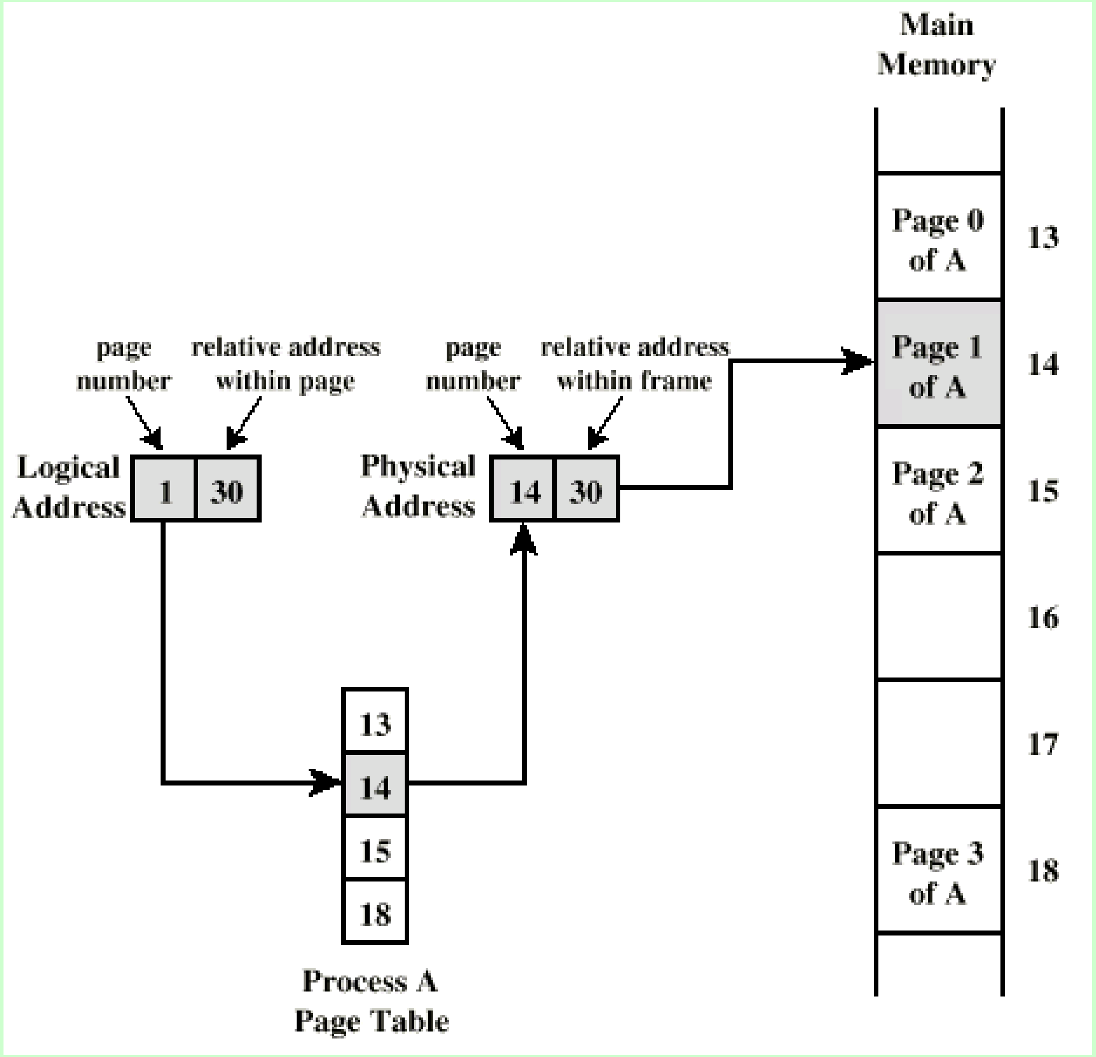
Paging

- 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

Allocation of Free Frames



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

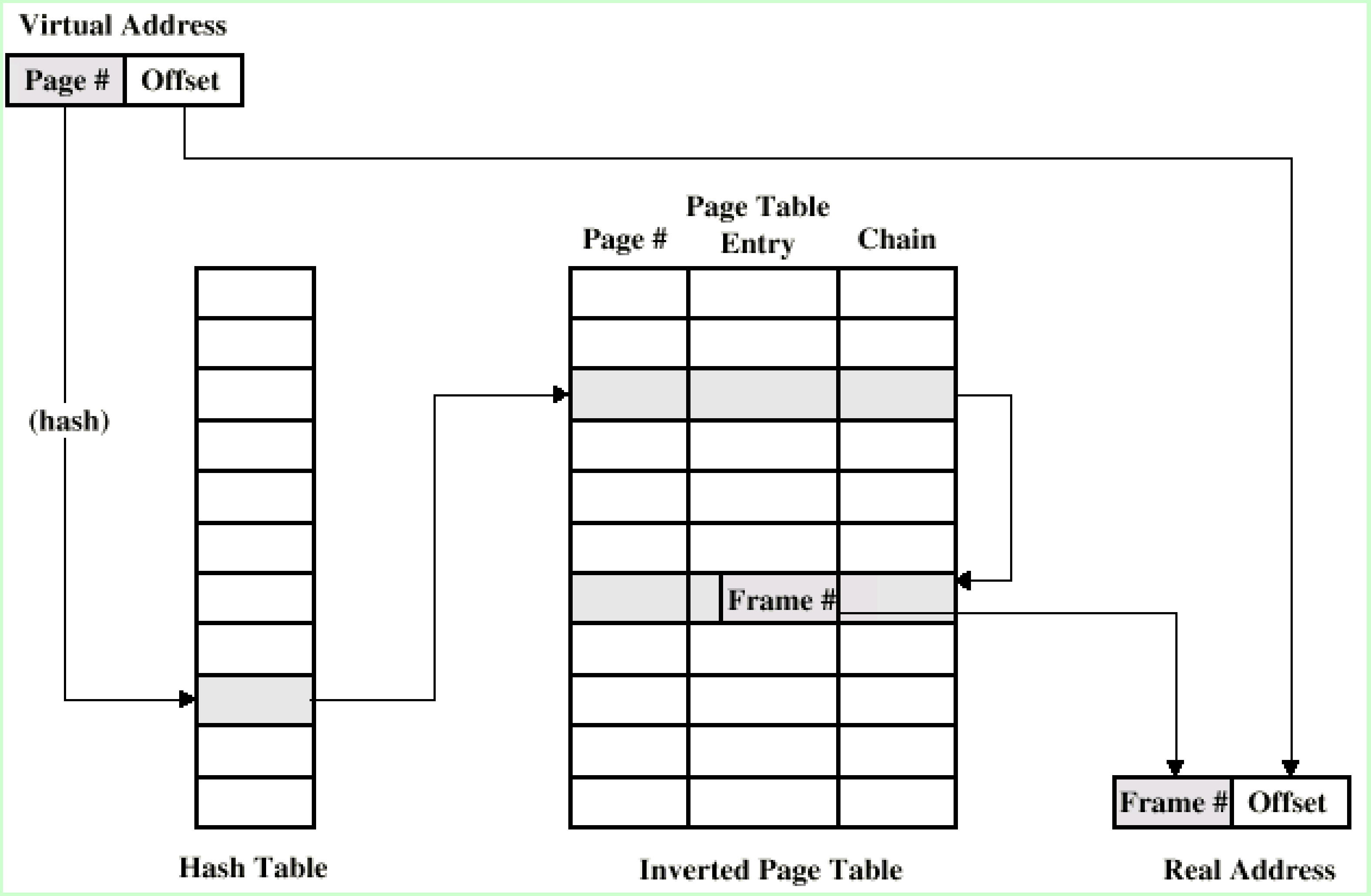
- Solutions
  - Good page replacement algorithms
  - Reduce number of processes running
  - Fit more memory

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

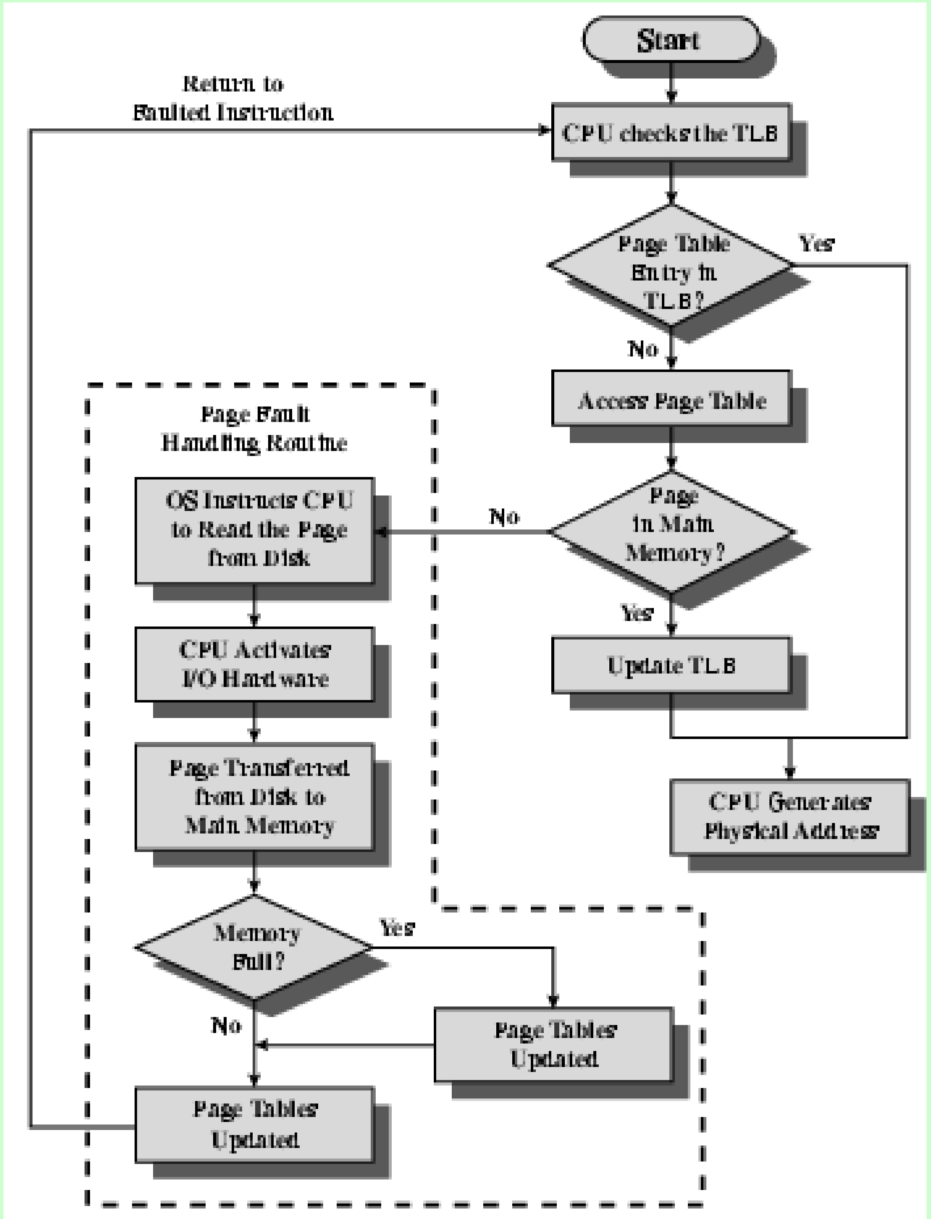
Inverted Page Table Structure



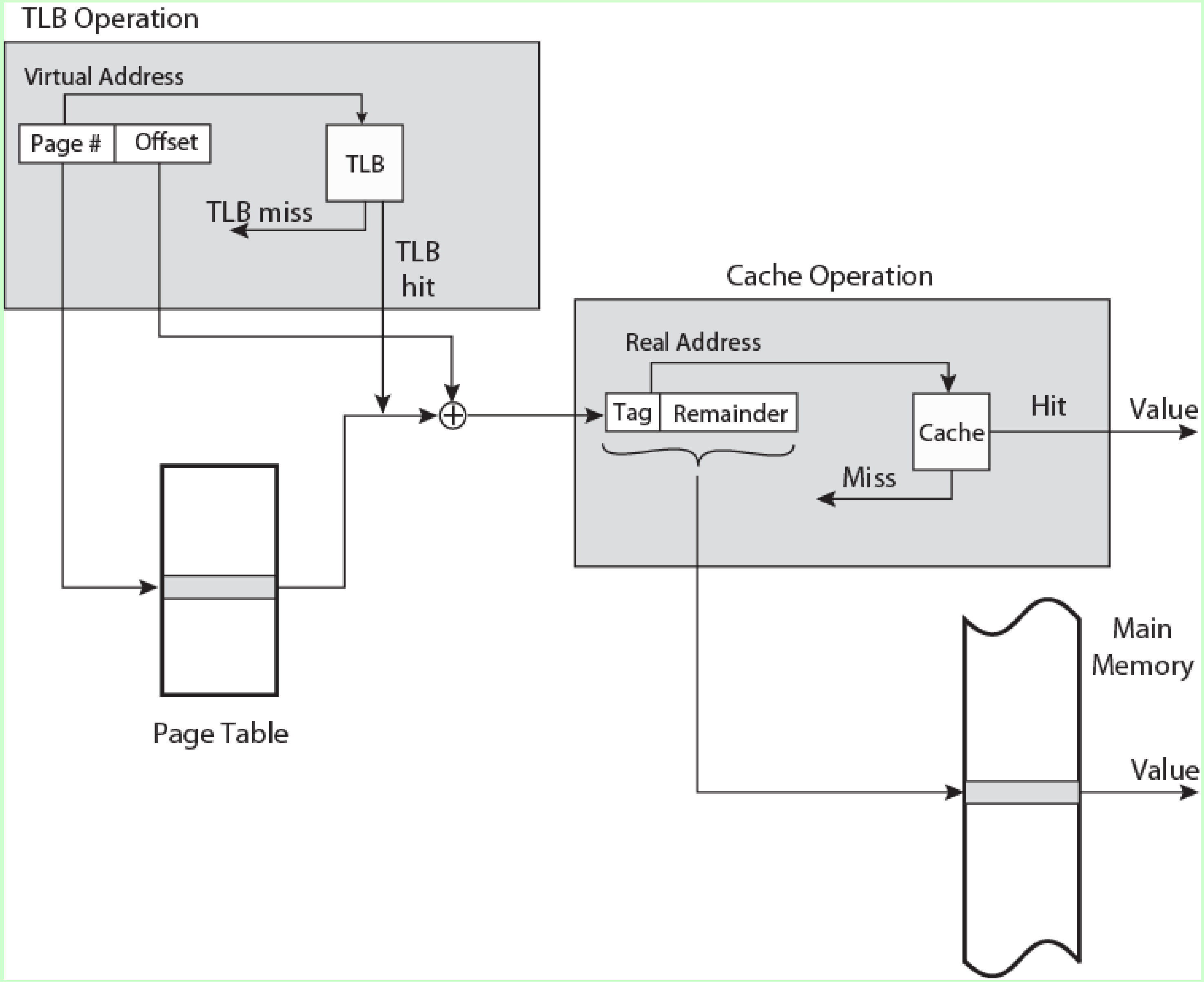
Translation Lookaside Buffer

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

TLB Operation


Start
Return to Faulted Instruction
CPU checks the TLB
Page Table Entry in TLB?
Yes
No
Access Page Table
Page Fault Handling Routine
OS Instructs CPU to Read the Page from Disk
No
Page in Main Memory?
Yes
Update TLB
CPU Activates I/O Hardware
Page Transferred from Disk to Main Memory
CPU Generates Physical Address
Memory Full?
Yes
No
Page Tables Updated
Page Tables Updated

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

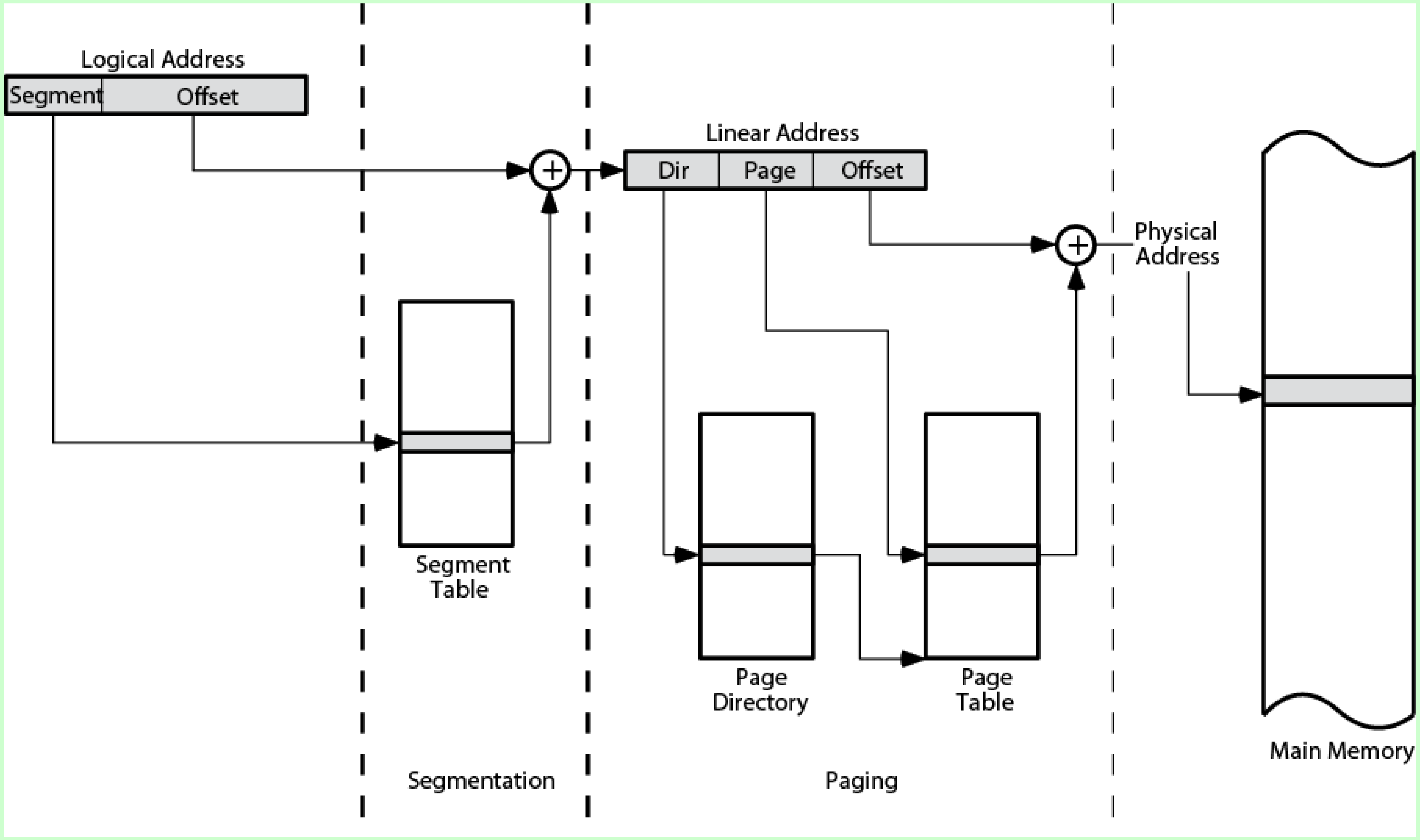
Advantages of Segmentation

- 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} = 4Gbytes
- Segmented 2^{46}=64 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

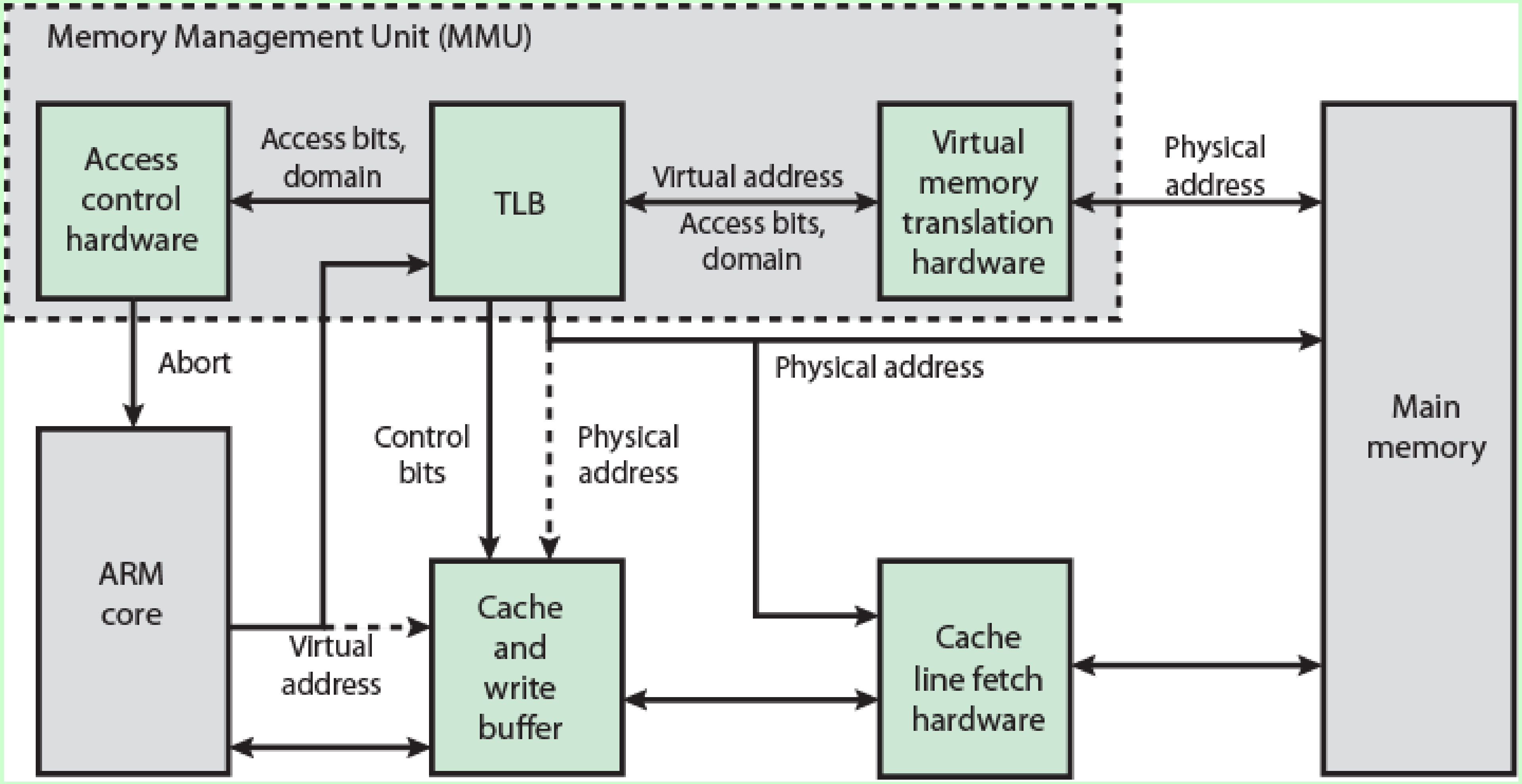
Pentium II Protection

- 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

ARM Memory System Overview



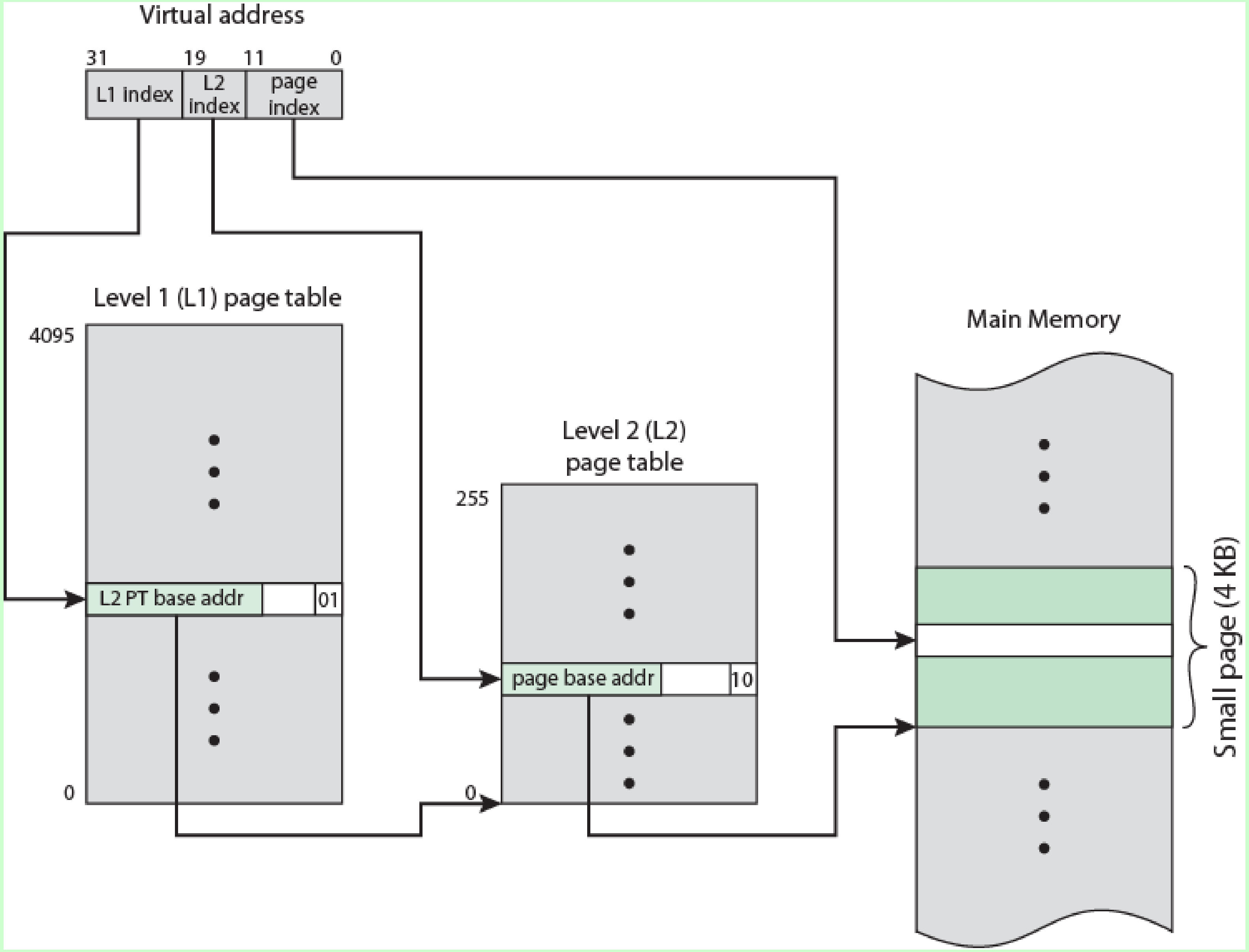
ARM Memory Management

- Virtual memory translation
  - One or two levels of tables
- Translation lookaside buffer (TLB)
  - Cache of recent entries
  - If available, TLB directly sends physical address to main memory
- Data exchanged between processor and main memory via cache
- Logical cache organization
  - On cache miss, ARM supplies address directly to cache as well as TLB
- Physical cache organization
  - TLB must supply physical address to cache
- Access control bits also in translation tables

Virtual Memory Address Translation

- Memory access based on either sections or pages
- Supersections (optional)
  - 16-MB blocks of main memory
- Sections
  - 1-MB blocks of main memory
- Large pages
  - 64-KB blocks of main memory
- Small pages
  - 4-KB blocks of main memory
- Sections and supersections allow mapping of large region of memory with single TLB entry
- Additional access control mechanisms
  - Small pages use 1KB subpages
  - Large pages use 16KB subpages
- Two level translation table held in main memory
  - First-level table holds section and supersection translations, and pointers to second-level tables
  - Second-level tables: Hold both large and small page translations
- MMU
  - Translates virtual to physical addresses
  - Derives and checks access permission
  - After TLB miss
- Start with first-level fetch
  - Section-mapped access only requires first-level fetch
  - Page-mapped access requires second-level fetch

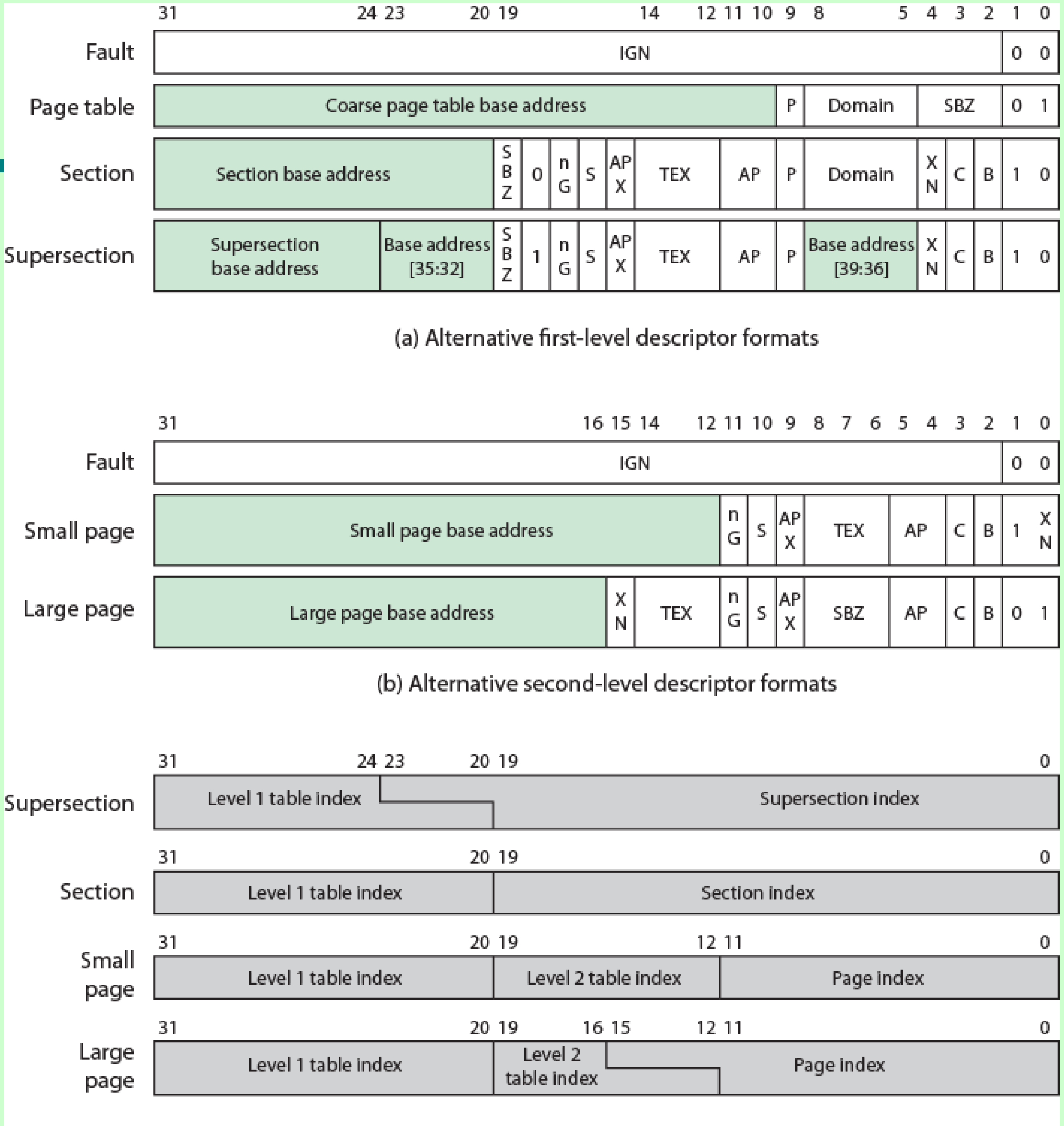
ARM Virtual Memory Address Translation for Small Pages - Diagram



ARM Virtual Memory Address Translation for Small Pages

- Single L1 page table
  - 4K 32-bit entries
  - Each L1 entry points to L2 page table
- Each L2 page table
  - 255 32-bit entries
  - Each L2 entry points to 4-KB page in main memory
- 32-bit virtual address
  - Most significant 12 bits index L1 page table
  - Next 8 bits index relevant L2 page table
  - Least significant 12 bits index a byte in relevant main memory page
- Similar procedure for large pages
- Sections and supersection only use L1 page table

ARMv6 Memory Management Formats



(a) Alternative first-level descriptor formats

(b) Alternative second-level descriptor formats

(c) Virtual memory address formats

ARM Memory-Management Parameters

- **Access Permission (AP), Access Permission Extension (APX)**
  - Control access to corresponding memory region
  - Access without required permissions raises Permission Fault
- **Bufferable (B) bit**
  - With TEX bits, how write buffer is used
- **Cacheable (C) bit**
- Can memory region be mapped through cache?
- **Domain**
  - Collection of memory regions
  - Access control can be applied on the basis of domain
- **not Global (nG)**
  - Translation marked as global (0), or process specific (1)?
- **Shared (S)**
  - Translation is for not-shared (0), or shared (1) memory?
- **SBZ**
  - Should be zero
- **Type Extension (TEX)**
  - Together with B and C bits, control accesses to caches
  - How write buffer is used
  - If memory region is shareable
    - Must be kept coherent
- **Execute Never (XN)**
  - Region is executable (0) or not executable (1)?

Memory Management Formats – L1 table

- L1 table
  - Entry describes how associated 1-MB virtual address range is mapped
- Bits [1:0] = 00
  - Virtual addresses unmapped
  - Attempts to access generate translation fault
- Bits [1:0] = 01
  - Physical address of L2 page table which specifies how associated virtual address range is mapped
- Bits [1:0] = 01 and bit 19 = 0
  - Section descriptorBits [1:0] = 01 and bit 19 = 1
  - Supersection descriptor
- Entries with bits [1:0] = 11
  - Reserved

L2 Table
Small and Large Pages

- For memory structured into pages
- L1 page entry bits [31:10] point to a L2 page table
- Small pages
  - L2 entry holds 20-bit pointer to base address of 4-KB page in main memory
- Large pages
  - Virtual address includes 12-bit index to L1 table and an 8-bit index to L2 table
  - 64-KB large pages have 16 bit page index portion
  - Four-bit overlap between page index field and L2 table index field
  - Page table entries in L2 page table replicated 16 times
  - L2 page table reduced from 256 entries to 16 if all refer to large pages
  - L2 page can service mixed large and small pages

L2 Table
Sections and Supersections

- Sections or supersections
  - One-level page table access
- Sections
  - L1 entry Bits [31:20] hold 12-bit pointer to 1-MB section
- For supersections
  - L1 bits [31:24] hold 8-bit pointer to base of the 16-MB section
- Page table entry replication is required
  - Supersections L1 table index portion of virtual address overlaps 4 bits with supersection index portion of virtual address
  - 16 identical L1 page table entries
- Physical address space can be expanded by up to eight additional address bits
  - Bits [23:20] and [8:5]
  - Implementation dependent
  - Interpreted as extending physical memory by up to $2^8 = 256$
  - Physical memory may be up to 256 times memory space available to each individual process

Access Control

- Region of memory can be designated as no access, read only, or read/write
- Region can be designated privileged access (operating Systems) only
- Domain
  - Collection of sections and/or pages with particular access permissions
  - 16
  - Multiple processes can use same translation tables while maintaining some protection from each other
  - Page table entry and TLB entry contain domain field
  - Two-bit field in the Domain Access Control Register controls access to each domain
  - Whole memory areas can be swapped very efficiently
- Clients
  - Must observe permissions of individual sections and/or pages in domain
- Managers
  - Control domain behavior
    - Sections and pages in domain access
    - Bypass access permissions for table entries in domain
- Programs can be
  - Client of some domains
  - Manager of other domains
  - No access to remaining domains

Required Reading

- Stallings chapter 8
- Stallings, W. [2004] *Operating Systems*, Pearson
- Loads of Web sites on Operating Systems