Security of embedded Systems

Peter Langendörfer

telefon: 0335 5625 350
fax: 0335 5625 671

e-mail: langendoerfer [at] ihp-microelectronics.com
web: http://www.tu-cottbus.de/fakultaet1/de/sicherheit-in-pervasiven-systemen/

Organizational stuff

- Exam: most probably oral
- Lecture schedule subject of change due to „Bahn issues“
- No lectures at:
  - October 22
  - November 5

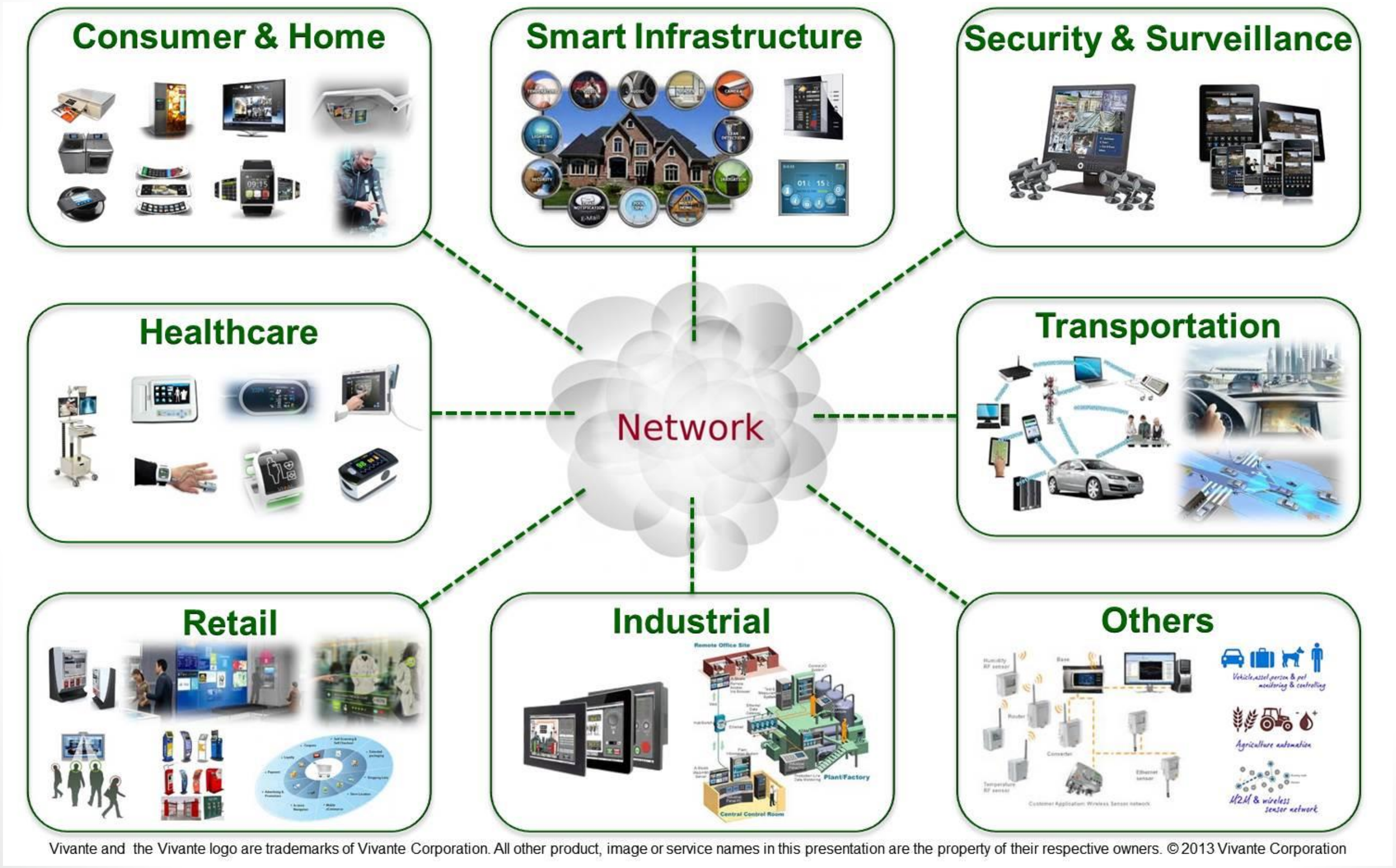
- Exercises on demand: Dr. Z. Dyka
  - dyka@ihp-microelectronics.com

Outline

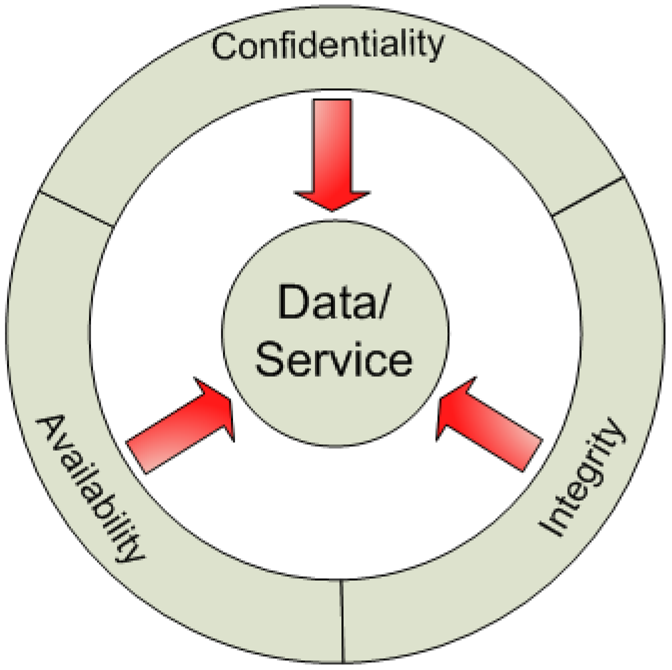
- Introduction
- Design Principles
- Memory Management recap
- Attacks
- Protection means
  - Hypervisor/Micro kernels
  - Canaries
  - Isolation
  - Access control/rights management
- Code Attestation
- Secure Code Update

Introduction

Introduction: Application Areas


Consumer & Home
Smart Infrastructure
Security & Surveillance
Healthcare
Network
Transportation
Retail
Industrial
Others
Vehicle, asset, person & pet monitoring & controlling
Agriculture automation
M2M & wireless sensor network
Vivante and the Vivante logo are trademarks of Vivante Corporation. All other product, image or service names in this presentation are the property of their respective owners. © 2013 Vivante Corporation

Introduction: Security Goals



- ***Confidentiality:*** keeping information secret from unauthorized access
- ***Integrity:*** only authorized users can change information
- ***Availability:*** information needs to be available to authorized users

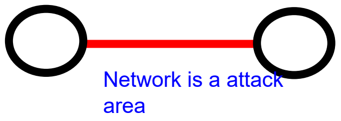
Introduction: Threat Model

Threat model

Counter measures

Dolev-Yao:

Application area: Industrial automation

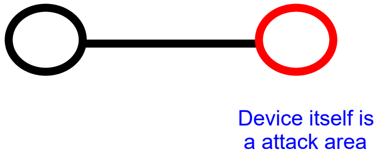


Cryptographic functions:

low energy, low cost accelerators, integration into sensor nodes

Extended Dolev-Yao:

Application area: homeland security



Cryptographic functions, tamper resistance

disabling scan chains; MPUs for MCU

Introduction: Attack Types

Types of Potential Attacks

- ***Local attacks*** assume a physical or remote access to the target that has weaknesses in hardware or runs vulnerable software.
- ***Remote attacks*** include all types of attacks on the network layer to confuse or cut communication connections.
- ***Tamper attacks*** include physical attacks on the system's hardware to gather stored data or engineering information.

µContoller and crypto hardware

- Microcontrollers are an essential building block in the considered application area
- Public key cryptography normally considered to be to costly:
  Elliptic Curve Cryptography (ECC): TelosB (16bit µC, 8MHz: 12s)
- MSP430 compliant µC + Plus crypto suite

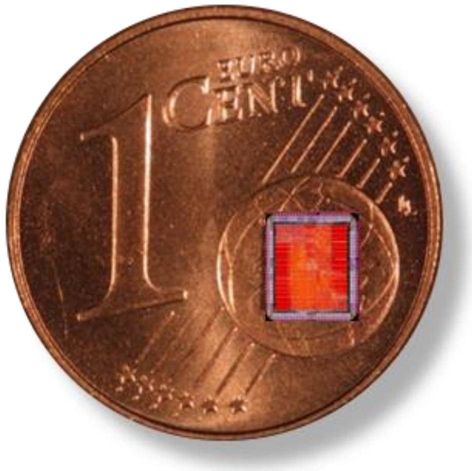
IHP 0.25µm technology

ECC hardware

- Time for one ECPM: 0.42 ms
- Energy for one ECPM: 17.4 µJ (41.4mW·0.42ms)

AES hardware

- Time to encrypt 128bit: 0.002 ms
- Energy to encrypt 128bit: 0.025 µJ (13.7mW·1.8µs)



Isn’t Crypto All We Need?

Limitations of Crypto

- Can’t prevent traffic analysis
- Can’t prevent re-transmitted packets
- Can’t prevent replayed packets
- Can’t prevent delayed packets
- Can’t prevent packets from being jammed
- Can’t prevent malicious insiders

- Crypto is not magic fairy dust
  It won’t magically make insecure services secure

- Can’t prevent captured nodes but they can render crypto useless

Introduction: Design Principles

Core Principle that guarantee design/building of a secure system

- ***Small interface***
- ***Access-control contracts***
- ***Tunneling***
- ***Secure booting***
- ***Effective resource control***

Introduction: Design Principles

Core Principle that guarantee design/building of a secure system

- ***Small interface*** approach can be built by using two alternative approaches:
  - The μ-kernel approach separates the system in small pieces,
  - extensible systems use safe languages or transaction-like mechanisms

- ***Access-control contracts*** an object or a group of objects declare their needs and the specific functions that they provide
  - role-based access control (RBAC).

Introduction: Design Principles

Core Principle that guarantee design/building of a secure system

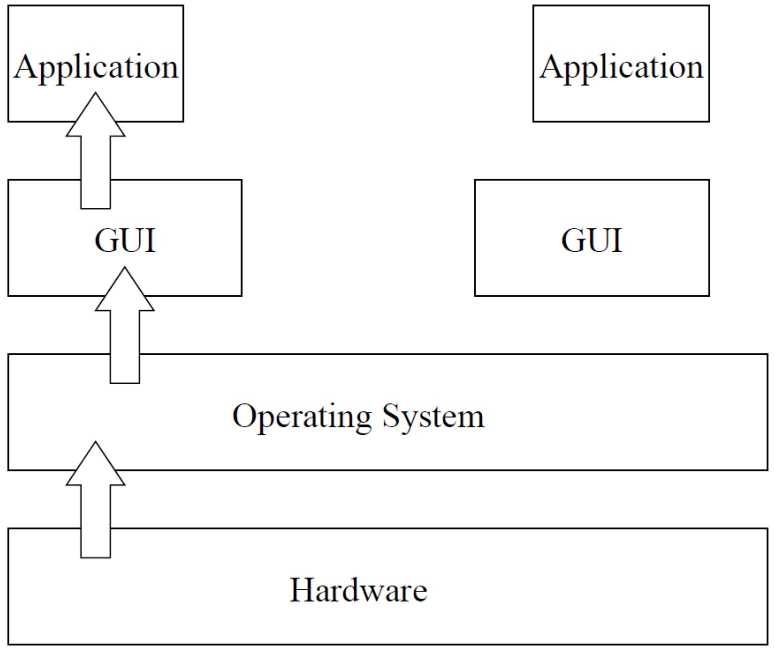
- ***Tunneling*** adding a required property to a software component by using an additional layer.
  - This may include an insecure communication channel that is used to transfer data. Hence, the provided security level of the software component that implements the additional layer can be ignored.

- ***Effective resource control*** providing an effective defense against denial-of-service attacks
  - Becoming key in the field of embedded system

Introduction: Design Principles

Core Principle that guarantee design/building of a secure system

- ***Secure booting*** ensures that a specific OS and a specific application is indeed running on a specific device by establishing a secure boot chain with a (hardware-based) trust anchor.



Introduction: Design Principles

Core Principle that guarantee design/building of a secure system

- ***Virtual machines*** provide a high level separation of software components by an emulation of a hardware architecture.
  - the costs of emulating the hardware architecture are an issue (reproduction of the function or action of a different computer, software system, etc.)
  - (reproduce the function or action of (a different computer, software system, etc.)
  - Currently considered to be acceptable for powerful devices only
- ***Separation of mechanisms and policies*** is important
  - Mechanisms are a collection of functions and facilities that are necessary to enforce policies.
  - system designer is in control of complex decisions and operations
- Here we consider ***protection*** as a mechanism to ensure the integrity of an operation implemented in a device
- The know how ***how to build a secure system***
  - Is well-established in traditional OSs.
  - these technologies are more or less applicable for embedded systems as well, or need to be properly adapted.

Memory Management

Slides taken from: Operating Systems Internals and Design Principles, 6/E William Stallings

Roadmap

- Basic requirements of Memory Management
- Memory Partitioning
- Basic blocks of memory management
  - Paging
  - Segmentation

The need for memory management

- Memory is cheap today, and getting cheaper
  - But applications are demanding more and more memory, there is never enough!
- Memory Management, involves swapping blocks of data from secondary storage.
- Memory I/O is slow compared to a CPU
  - The OS must cleverly time the swapping to maximise the CPU’s efficiency

Memory Management

Memory needs to be allocated to ensure a reasonable supply of ready processes to consume available processor time

Memory Management Requirements

- Relocation
- Protection
- Sharing
- Logical organisation
- Physical organisation

Requirements: Relocation

- The programmer does not know where the program will be placed in memory when it is executed,
  - it may be swapped to disk and return to main memory at a different location (relocated)
- Memory references must be translated to the actual physical memory address

Memory Management Terms

Table 7.1 Memory Management Terms

Term	Description
Frame	***Fixed***-length block of main memory.
Page	***Fixed***-length block of data in secondary memory (e.g. on disk).
Segment	***Variable-length*** block of data that resides in secondary memory.

Addressing

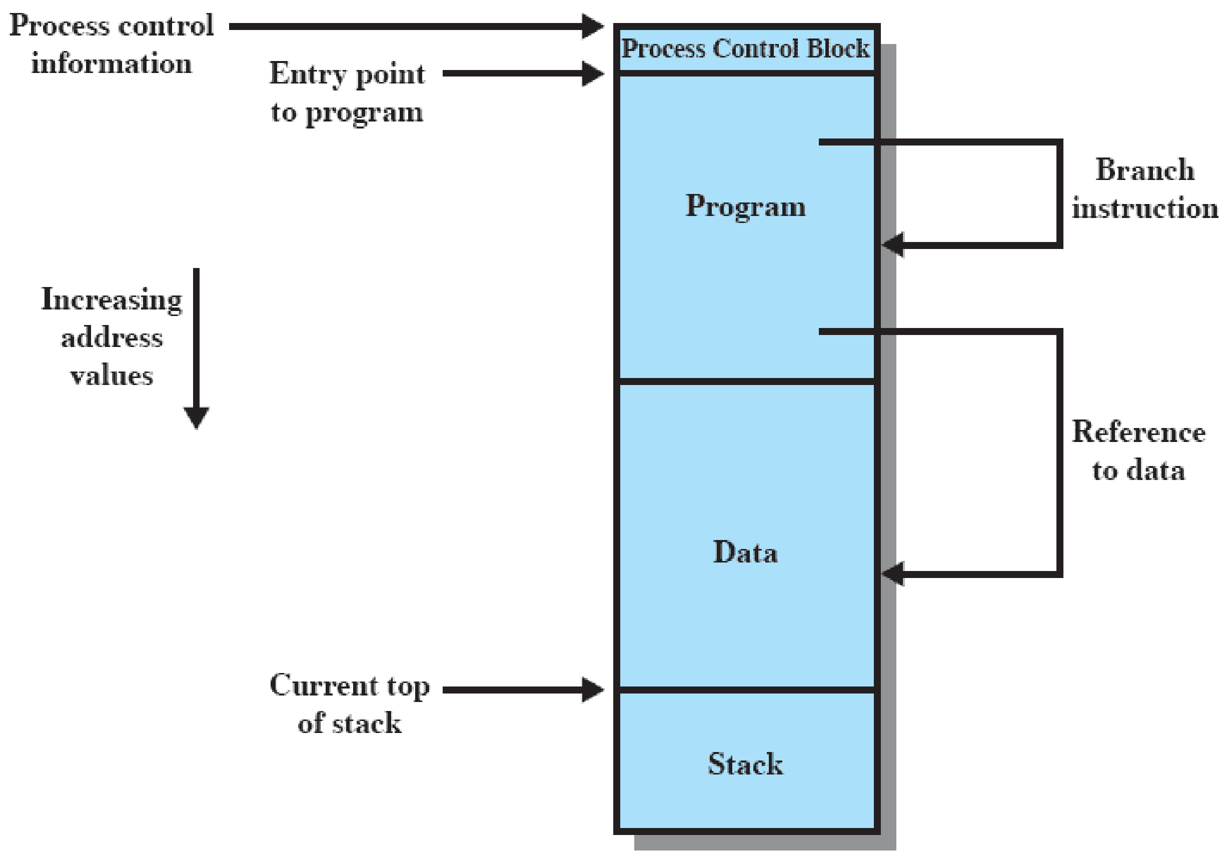


Figure 7.1 Addressing Requirements for a Process

Requirements: Protection

- Processes should not be able to reference memory locations in another process without permission
- Impossible to check absolute addresses at compile time
- Must be checked at run time

Requirements: Sharing

- Allow several processes to access the same portion of memory
- Better to allow each process access to the same copy of the program rather than have their own separate copy

Requirements: Logical Organization

- Memory is organized linearly (usually)
- Programs are written in modules
  - Modules can be written and compiled independently
- Different degrees of protection given to modules (read-only, execute-only)
- Share modules among processes
- Segmentation helps here

Requirements: Physical Organization

- Cannot leave the programmer with the responsibility to manage memory
- Memory available for a program plus its data may be insufficient
    - Overlaying allows various modules to be assigned the same region of memory but is time consuming to program
- Programmer does not know how much space will be available

Partitioning

- An early method of managing memory
  - Pre-virtual memory
  - Not used much now
- But, it will clarify the later discussion of virtual memory if we look first at partitioning
  - Virtual Memory has evolved from the partitioning methods

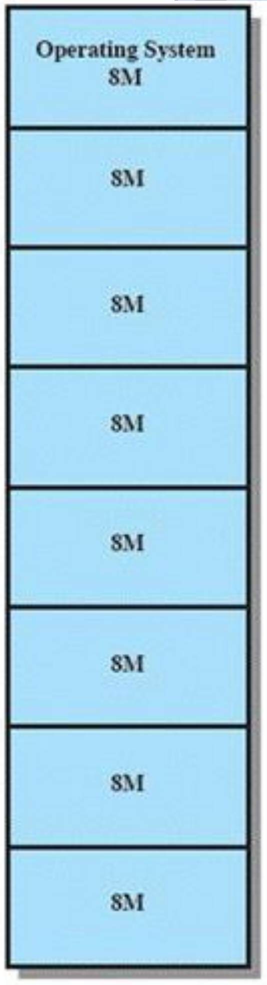
(it changes or develops over time)

Types of Partitioning

- Fixed Partitioning
- Dynamic Partitioning
- Simple Paging
- Simple Segmentation
- Virtual Memory Paging
- Virtual Memory Segmentation

Fixed Partitioning

- Equal-size partitions (see fig 7.3a)
  - Any process whose size is less than or equal to the partition size can be loaded into an available partition
- The operating system can swap a process out of a partition
  - If none are in a ready or running state



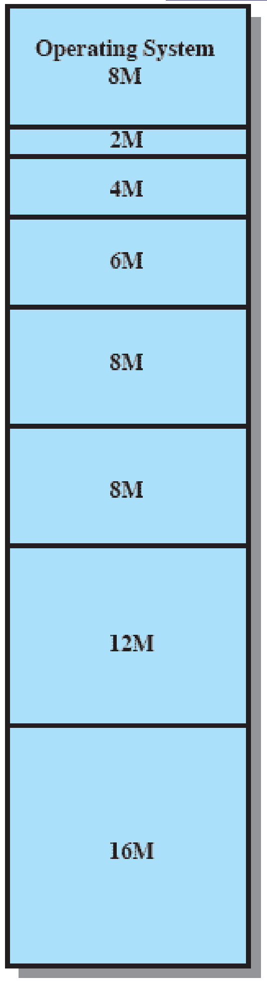
(a) Equal-size partitions

Fixed Partitioning Problems

- A program may not fit in a partition.
  - The programmer must design the program with overlays
- Main memory use is inefficient.
  - Any program, no matter how small, occupies an entire partition.
  - This is results in ***internal fragmentation.***

Solution – Unequal Size Partitions

- Lessens both problems
  - but doesn't solve completely
- In Fig 7.3b,
  - Programs up to 16M can be accommodated without overlay
  - Smaller programs can be placed in smaller partitions, reducing internal fragmentation



(b) Unequal-size partitions

Placement Algorithm

- Equal-size
  - Placement is trivial (no options)

(A trivial problem is easy to solve)

- Unequal-size
  - Can assign each process to the smallest partition within which it will fit
  - Queue for each partition
  - Processes are assigned in such a way as to minimize wasted memory within a partition

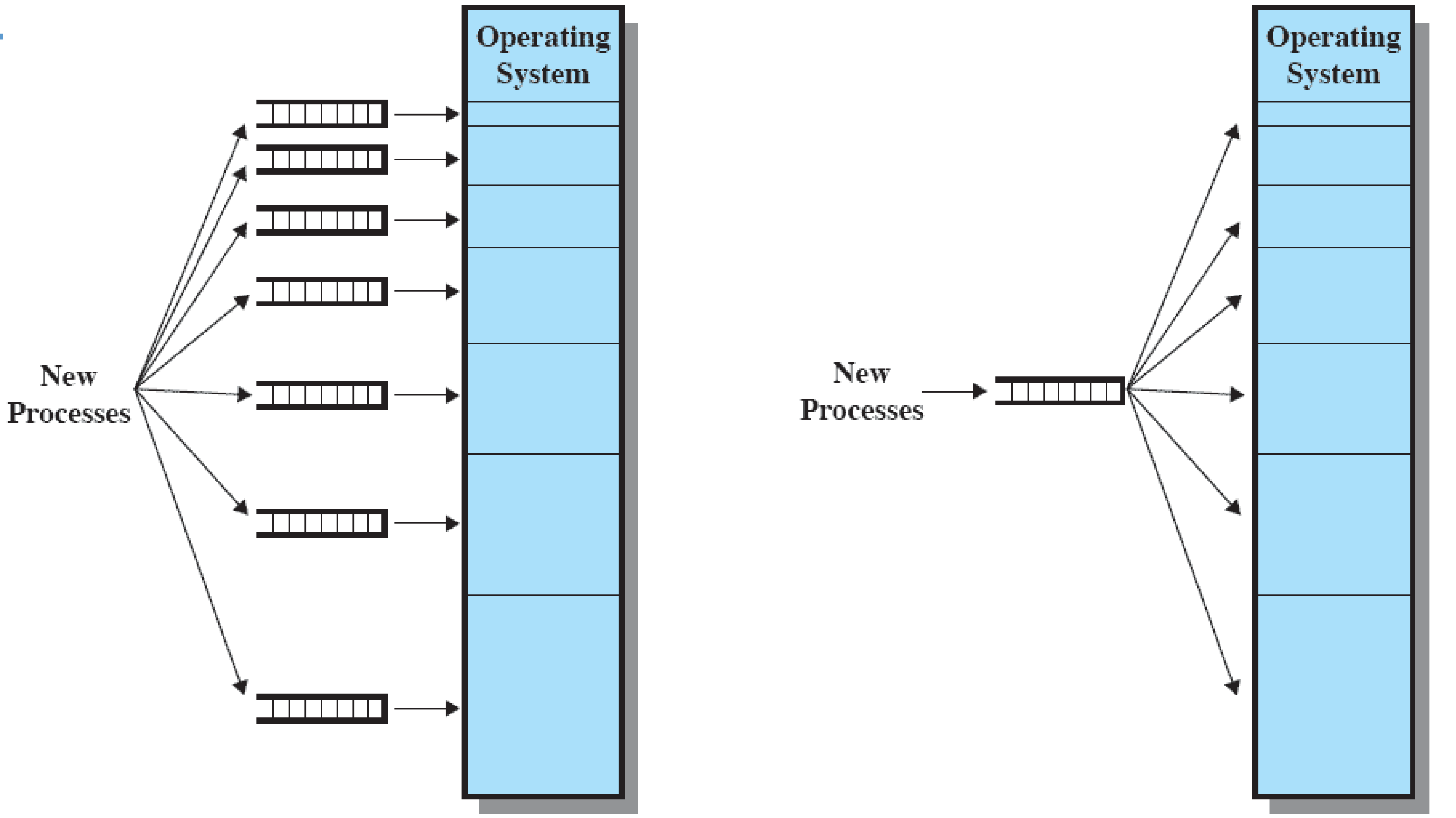
Placement Algorithm with Partitions

Equal-size partitions
If there is an available partition, a process can be loaded into that partition
because all partitions are of equal size, it does not matter which partition is used
If all partitions are occupied by blocked processes, choose one process to swap out to make room for the new process

Unequal-size partitions: use of multiple queues
assign each process to the smallest partition within which it will fit
A queue for each partition size
tries to minimize internal fragmentation
Problem: some queues will be empty if no processes within a size range is present

Unequal-size partitions: use of a single queue
When its time to load a process into main memory the smallest available partition that will hold the process is selected
increases the level of multiprogramming at the expense of internal fragmentation

Fixed Partitioning



(a) One process queue per partition

(b) Single queue

Figure 7.3 Memory Assignment for Fixed Partitioning

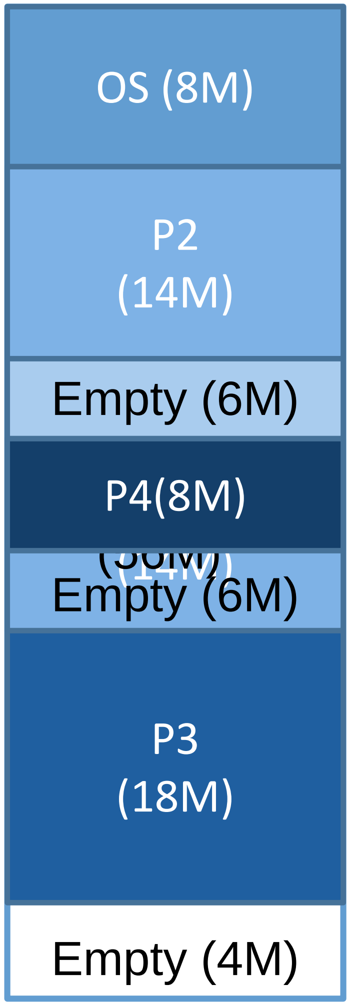
Remaining Problems with Fixed Partitions

- The number of active processes is limited by the system
  - I.E limited by the pre-determined number of partitions
- A large number of very small process will not use the space efficiently
  - In either fixed or variable length partition methods

Dynamic Partitioning

- Partitions are of variable length and number
- Process is allocated exactly as much memory as required

Dynamic Partitioning Example



Refer to Figure 7.4

- ***External Fragmentation***
- Memory external to all processes is fragmented
- Can resolve using ***compaction***
  - OS moves processes so that they are contiguous (sharing a common border; touching.)
  - Time consuming and wastes CPU time

Dynamic Partitioning

- Operating system must decide which free block to allocate to a process
- Best-fit algorithm
  - Chooses the block that is closest in size to the request
  - Worst performer overall
  - Since smallest block is found for process, the smallest amount of fragmentation is left
  - Memory compaction must be done more often

Dynamic Partitioning

- First-fit algorithm
  - Scans memory form the beginning and chooses the first available block that is large enough
  - Fastest
  - May have many process loaded in the front end of memory that must be searched over when trying to find a free block

Dynamic Partitioning

- Next-fit
  - Scans memory from the location of the last placement
  - More often allocate a block of memory at the end of memory where the largest block is found
  - The largest block of memory is broken up into smaller blocks
  - Compaction is required to obtain a large block at the end of memory

Allocation

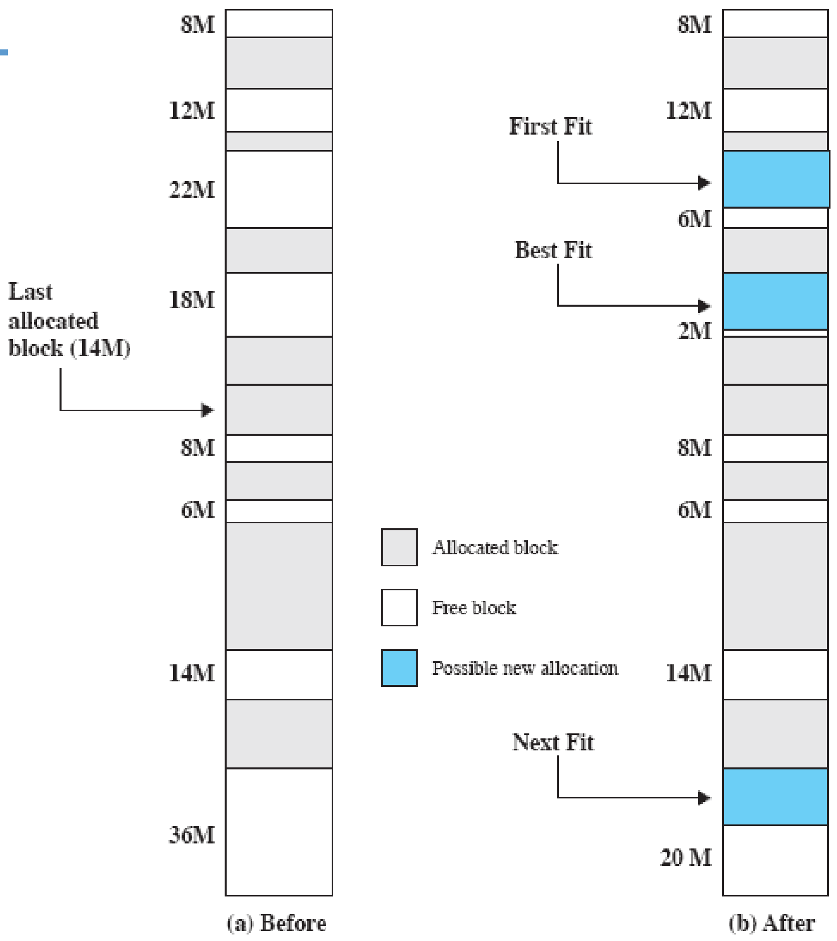


Figure 7.5 Example Memory Configuration before and after Allocation of 16-Mbyte Block

Buddy System

- Entire space available is treated as a single block of 2^U
- If a request of size *s* where $2^{U-1} < s <= 2^U$
  - entire block is allocated
- Otherwise block is split into two equal buddies
  - Process continues until smallest block greater than or equal to *s* is generated

Example of Buddy System

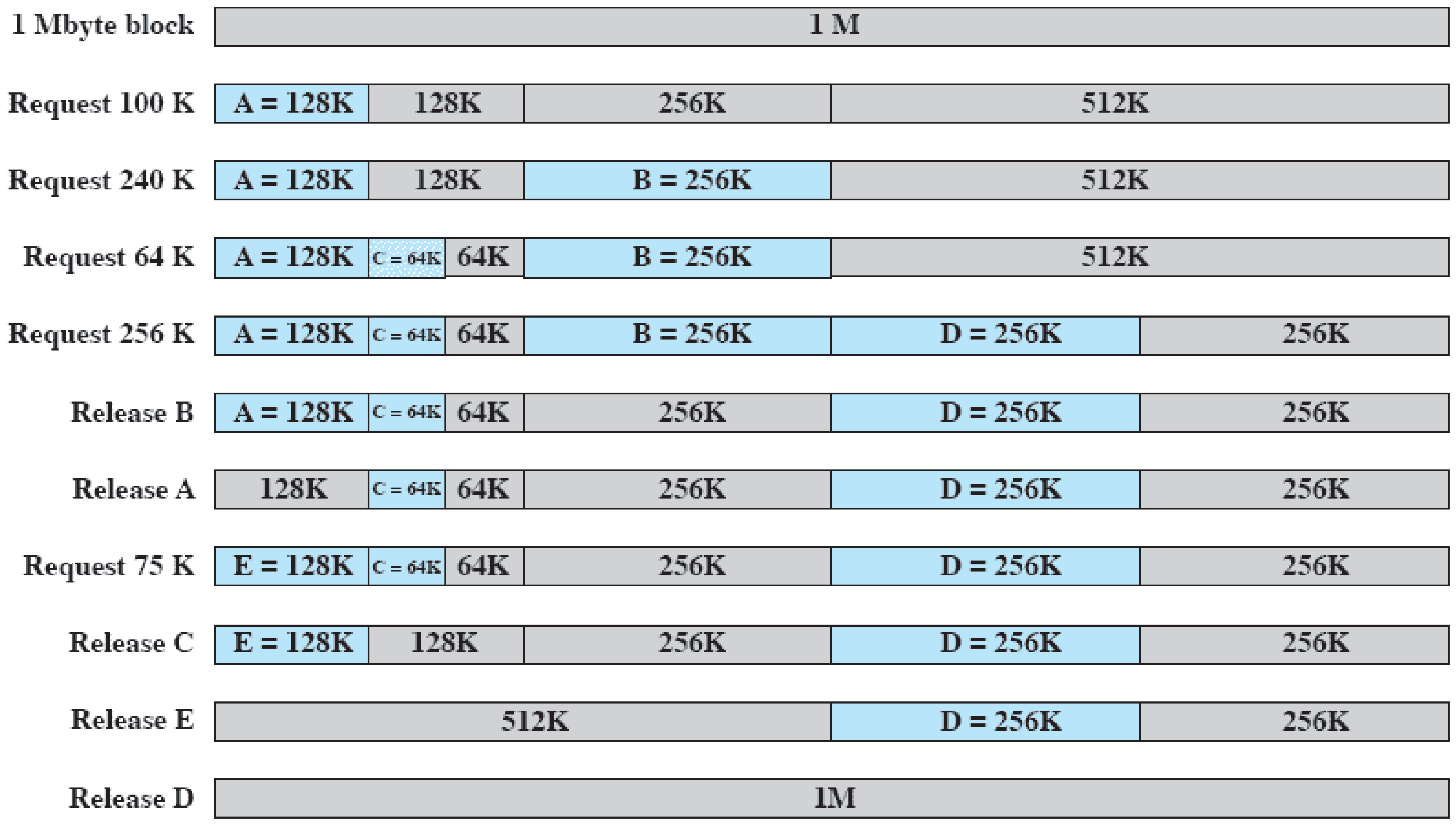

1 Mbyte block | 1 M
Request 100 K | A = 128K | 128K | 256K | 512K
Request 240 K | A = 128K | 128K | B = 256K | 512K
Request 64 K | A = 128K | C = 64K | 64K | B = 256K | 512K
Request 256 K | A = 128K | C = 64K | 64K | B = 256K | D = 256K | 256K
Release B | A = 128K | C = 64K | 64K | 256K | D = 256K | 256K
Release A | 128K | C = 64K | 64K | 256K | D = 256K | 256K
Request 75 K | E = 128K | C = 64K | 64K | 256K | D = 256K | 256K
Release C | E = 128K | 128K | 256K | D = 256K | 256K
Release E | 512K | D = 256K | 256K
Release D | 1M

Figure 7.6 Example of Buddy System

Tree Representation of Buddy System

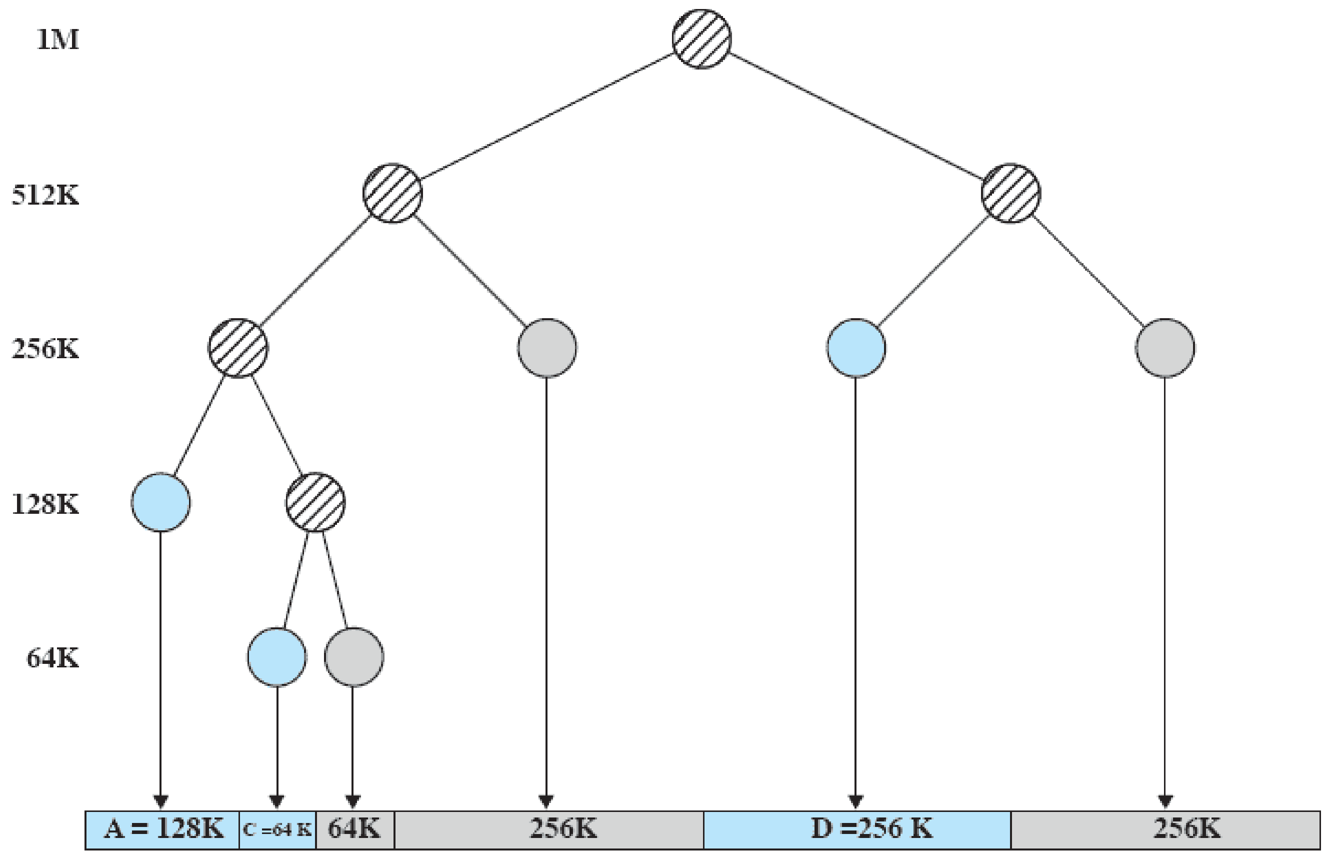


Figure 7.7 Tree Representation of Buddy System

Relocation

- When program loaded into memory the actual (absolute) memory locations are determined
- A process may occupy different partitions which means different absolute memory locations during execution
  - Swapping
  - Compaction

Addresses

- Logical
  - Reference to a memory location independent of the current assignment of data to memory.
- Relative
  - Address expressed as a location relative to some known point.
- Physical or Absolute
  - The absolute address or actual location in main memory.

Relocation

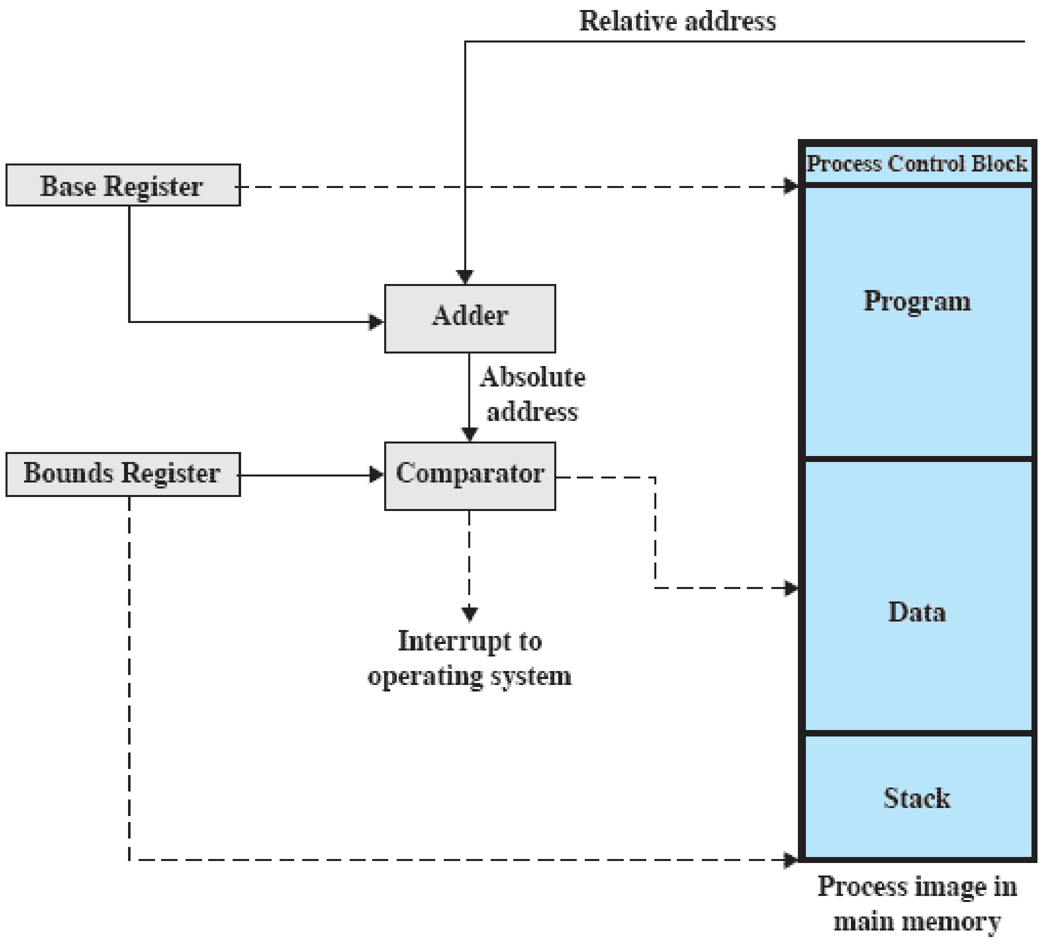


Figure 7.8 Hardware Support for Relocation

Registers Used during Execution

- Base register
  - Starting address for the process
- Bounds register
  - Ending location of the process
- These values are set when the process is loaded or when the process is swapped in

Registers Used during Execution

- The value of the base register is added to a relative address to produce an absolute address
- The resulting address is compared with the value in the bounds register
- If the address is not within bounds, an interrupt is generated to the operating system

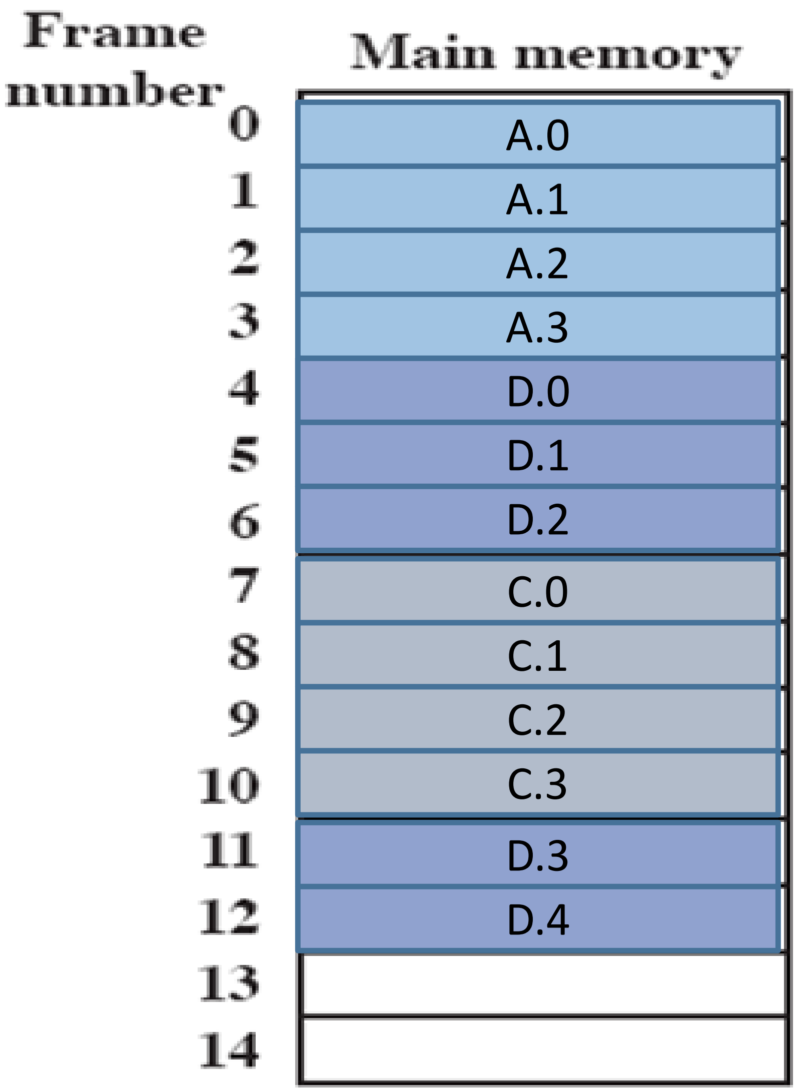
Paging

- Partition memory into small equal fixed-size chunks and divide each process into the same size chunks
- The chunks of a process are called ***pages***
- The chunks of memory are called ***frames***

Paging

- Operating system maintains a page table for each process
  - Contains the frame location for each page in the process
  - Memory address consist of a page number and offset within the page

Processes and Frames



Page Table

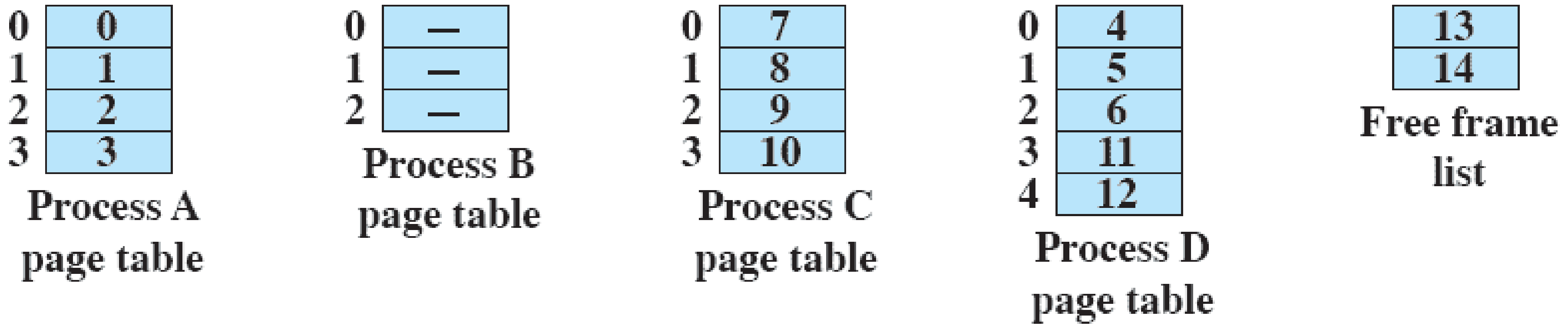


Figure 7.10 Data Structures for the Example of Figure 7.9 at Time Epoch (f)

Segmentation

- A program can be subdivided into segments
  - Segments may vary in length
  - There is a maximum segment length
- Addressing consist of two parts
  - a segment number and
  - an offset
- Segmentation is similar to dynamic partitioning

Logical Addresses

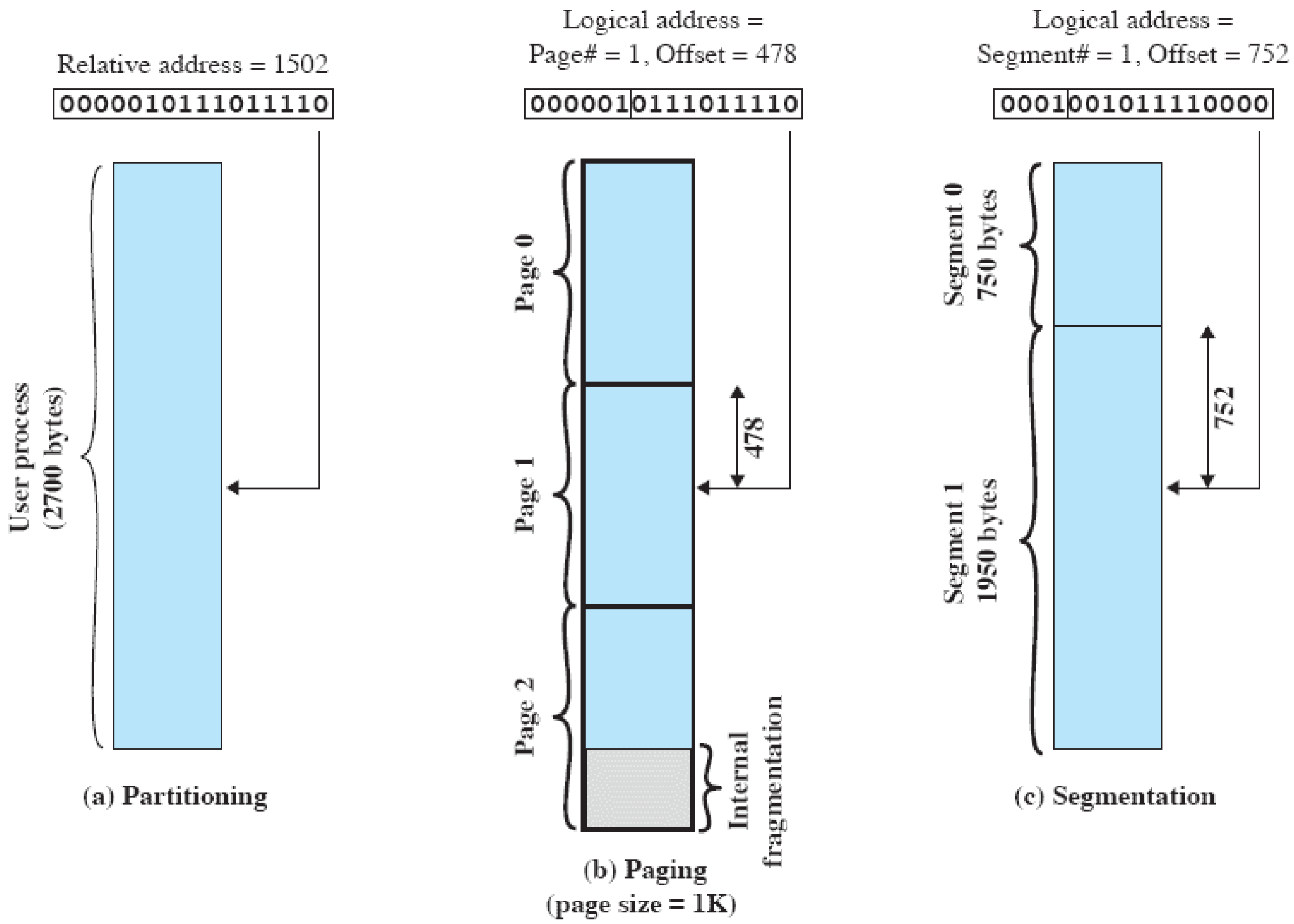
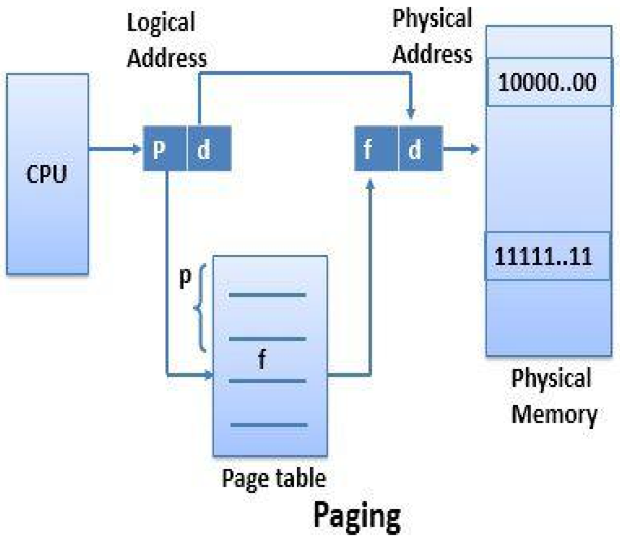
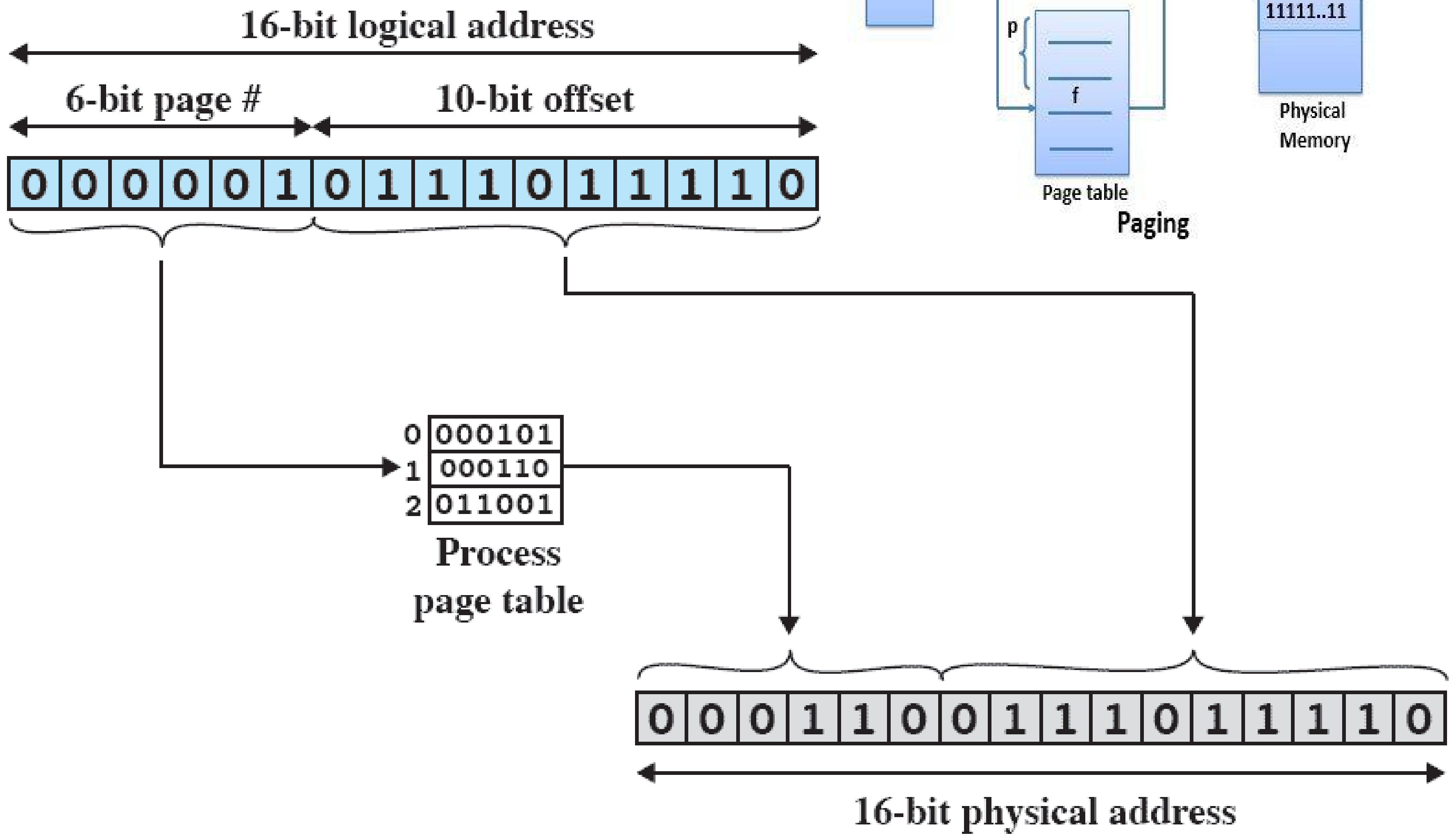


Figure 7.11 Logical Addresses

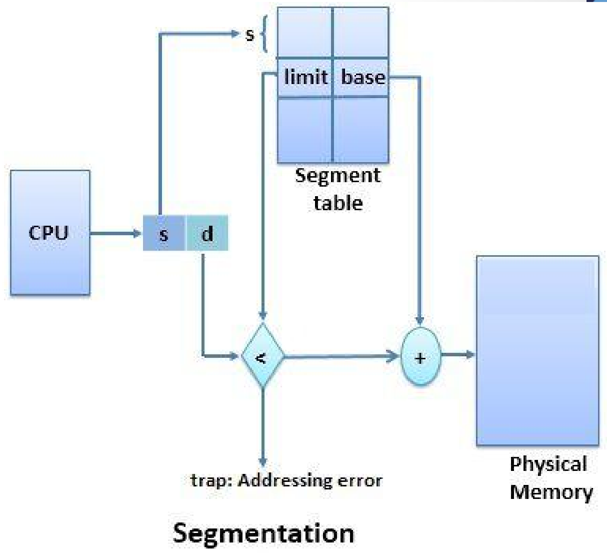
Paging

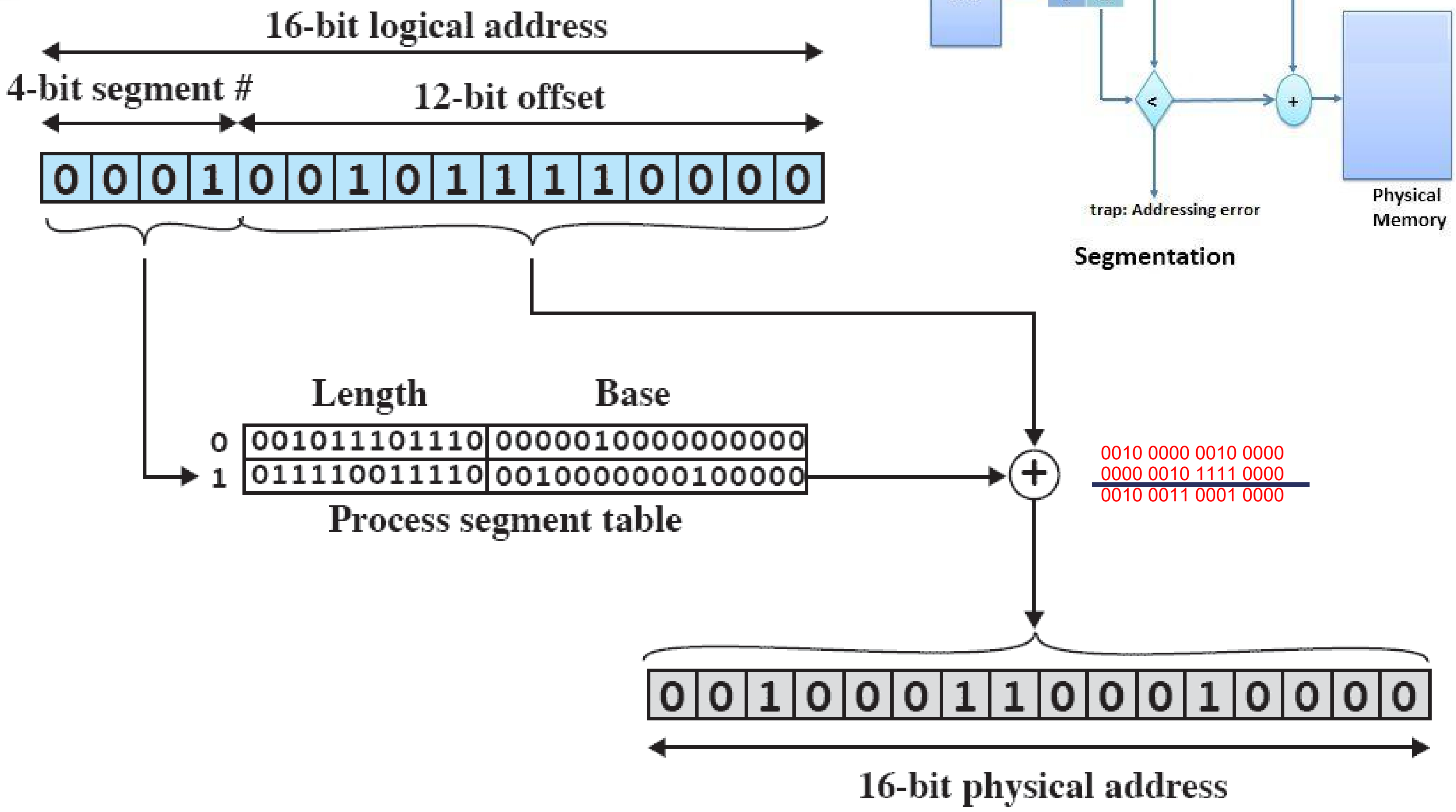




(a) Paging

Segmentation





0010 0000 0010 0000
0000 0010 1111 0000
0010 0011 0001 0000

(b) Segmentation

Figure 7.12 Examples of Logical-to-Physical Address Translation