Operating Systems

Tutorial

Fabian Klopfer

November 2, 2020

TOC

Intro

- Fabian, 2. Semester M.Sc. Computer & Information Science, fabian.klopfer@uni.kn
- Reminder: English Poll
- Fork the repository & do sheet 0
- More organisation stuff in the lecture
- The following slides are preliminary as the lecture is WIP
- sorry for the sloppy image references!

Processor & Compiler Reference

Not a must read for this course but a good reference for low level details:
AMD64 Manual
Intel x86 & x64 Manual
SystemV x86 Calling Convention
Comprehensive Low Level Information Collection

C Programming & Operating System Reference

Your best friend when programming:
Official C Language Reference

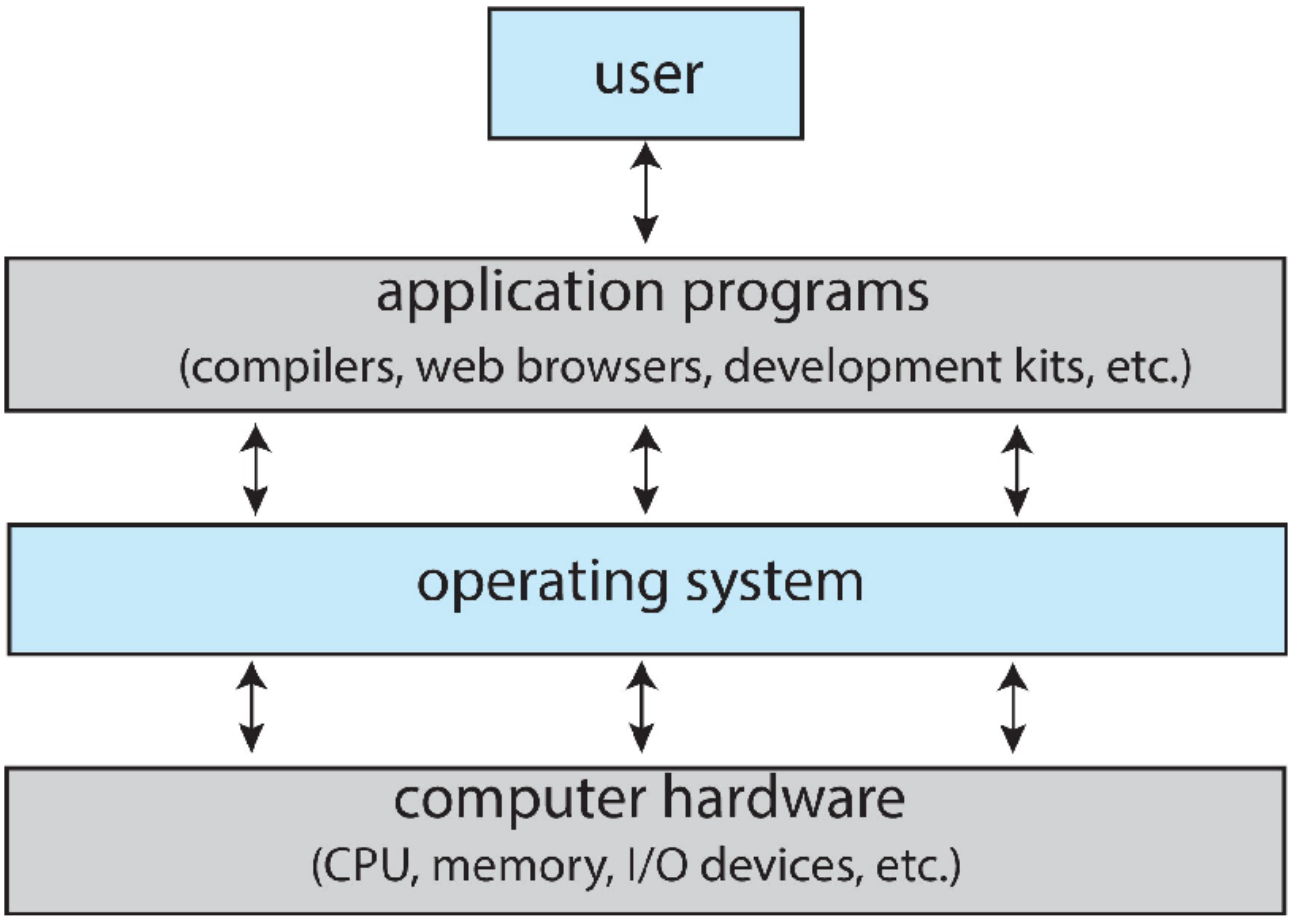
Unix & Linux references:
Linux Manuals (searchable!)
Linux Syscalls Man Page
POSIX Specification
The Linux Kernel Documentation Project

Literature hint

Get (digital) copies of all books!

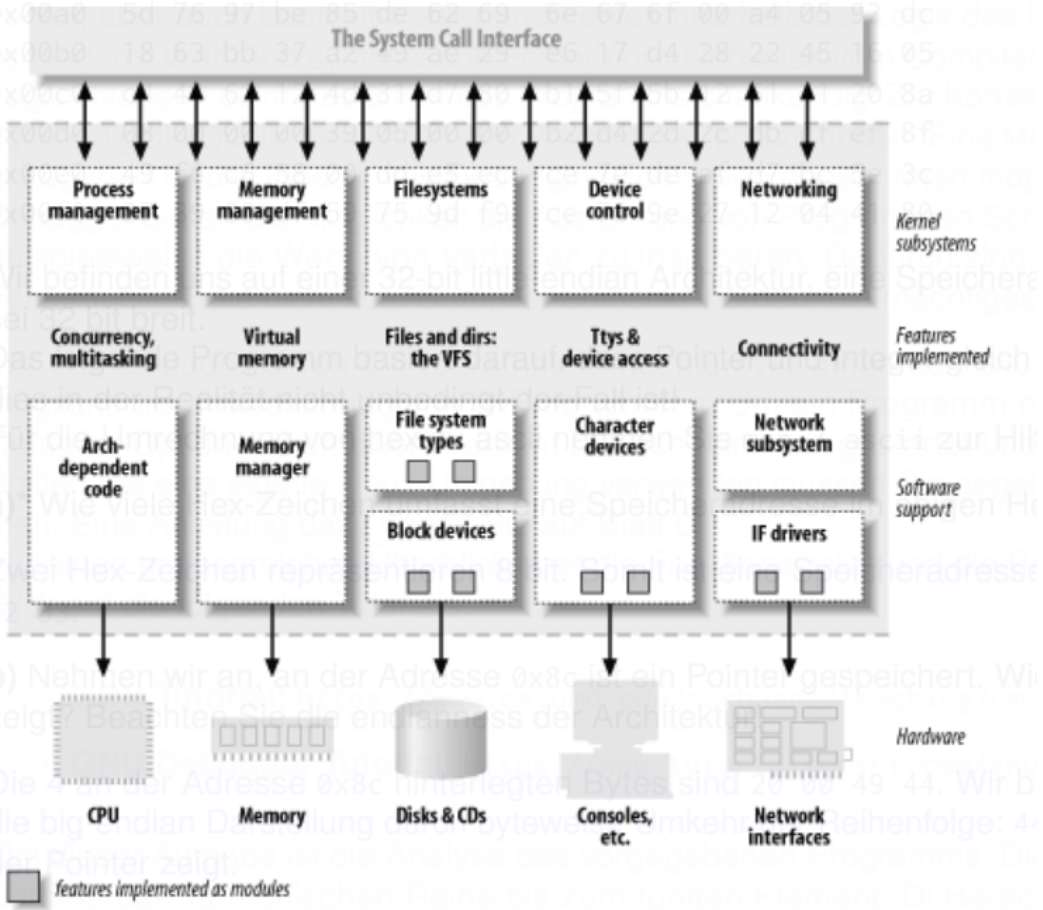
Many nice graphs:
https://dinus.ac.id/repository/docs/ajar/Operating_System.pdf

Overview I


user
application programs
(compilers, web browsers, development kits, etc.)
operating system
computer hardware
(CPU, memory, I/O devices, etc.)

Overview II

Figure 1-1. A split view of the kernel



Overview III

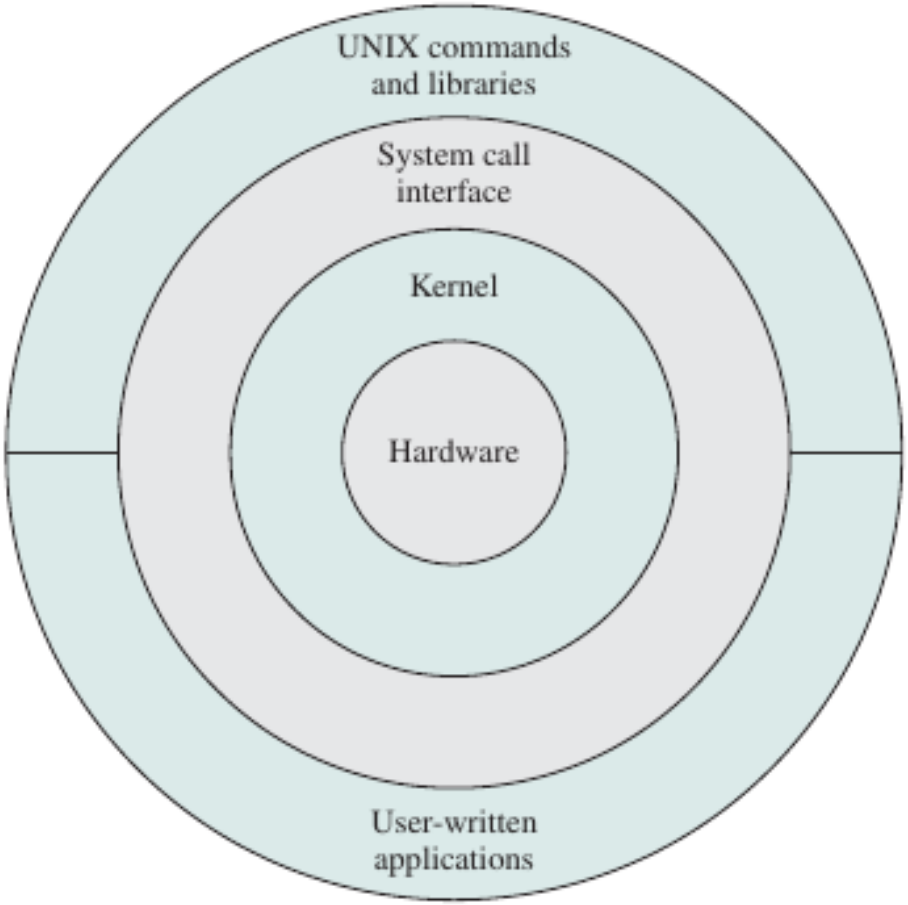
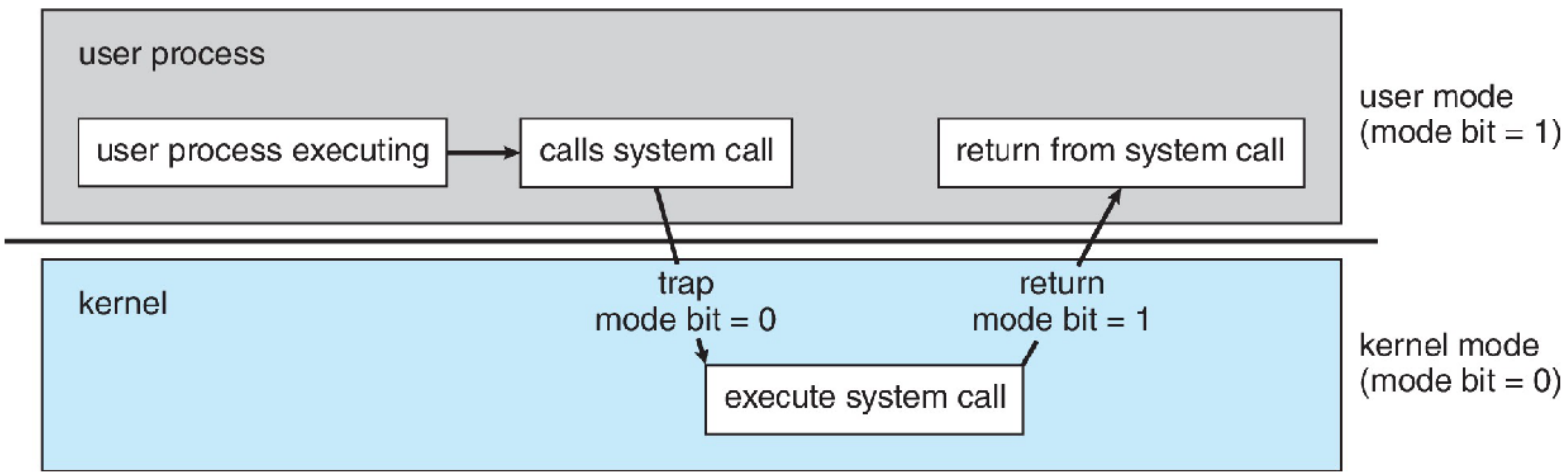


Figure 2.16 General UNIX Architecture

Overview IV


user process
user process executing
calls system call
return from system call
user mode
(mode bit = 1)
kernel
trap
mode bit = 0
return
mode bit = 1
execute system call
kernel mode
(mode bit = 0)

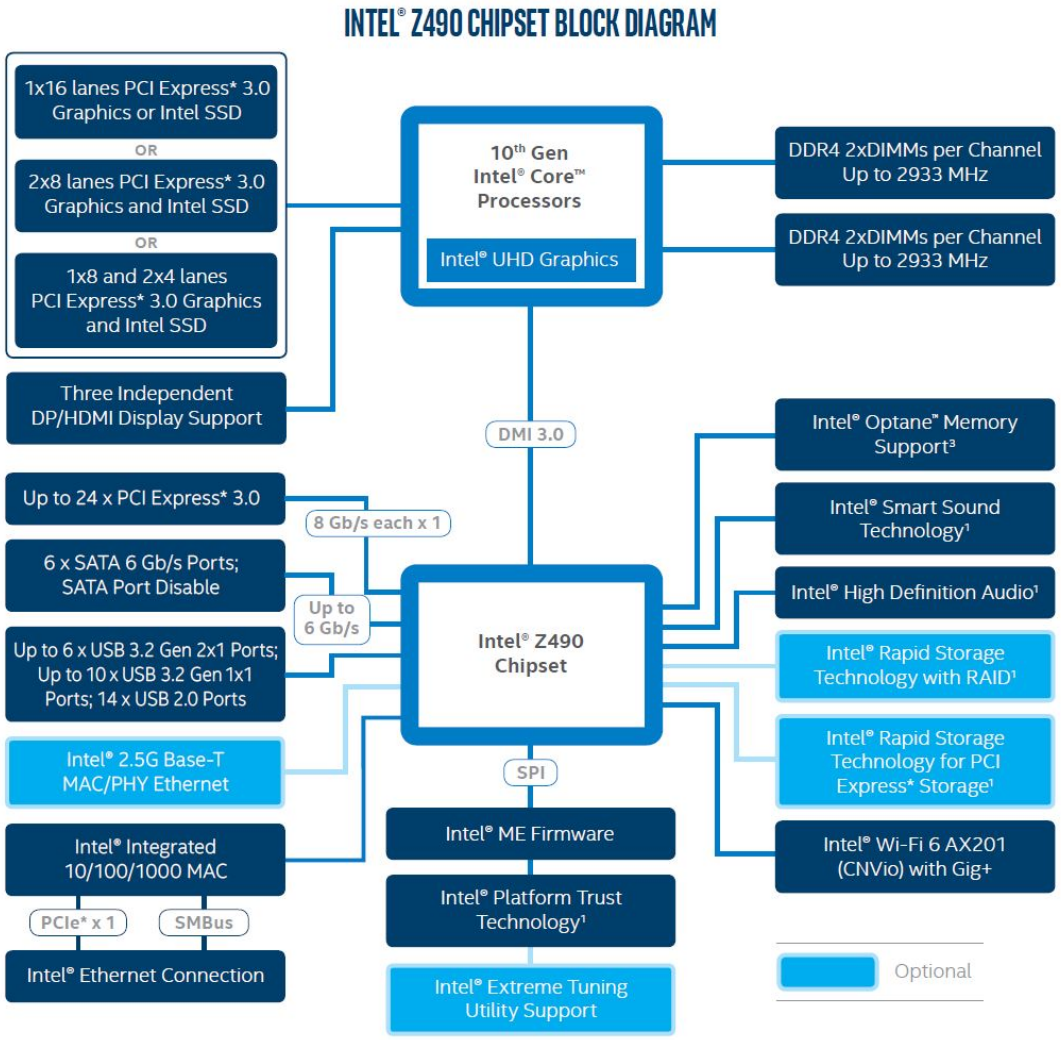
Overview V

EXAMPLES OF WINDOWS AND UNIX SYSTEM CALLS

The following illustrates various equivalent system calls for Windows and UNIX operating systems.

	Windows	Unix
Process control	`CreateProcess()`	`fork()`
	`ExitProcess()`	`exit()`
	`WaitForSingleObject()`	`wait()`
File management	`CreateFile()`	`open()`
	`ReadFile()`	`read()`
	`WriteFile()`	`write()`
	`CloseHandle()`	`close()`
Device management	`SetConsoleMode()`	`ioctl()`
	`ReadConsole()`	`read()`
	`WriteConsole()`	`write()`
Information maintenance	`GetCurrentProcessID()`	`getpid()`
	`SetTimer()`	`alarm()`
	`Sleep()`	`sleep()`
Communications	`CreatePipe()`	`pipe()`
	`CreateFileMapping()`	`shm_open()`
	`MapViewOfFile()`	`mmap()`
Protection	`SetFileSecurity()`	`chmod()`
	`InitlializeSecurityDescriptor()`	`umask()`
	`SetSecurityDescriptorGroup()`	`chown()`

Hardware I


INTEL® Z490 CHIPSET BLOCK DIAGRAM
1x16 lanes PCI Express* 3.0 Graphics or Intel SSD
OR
2x8 lanes PCI Express* 3.0 Graphics and Intel SSD
OR
1x8 and 2x4 lanes PCI Express* 3.0 Graphics and Intel SSD
Three Independent DP/HDMI Display Support
10th Gen Intel® Core™ Processors
Intel® UHD Graphics
DDR4 2xDIMMs per Channel Up to 2933 MHz
DDR4 2xDIMMs per Channel Up to 2933 MHz
DMI 3.0
Up to 24 x PCI Express* 3.0
8 Gb/s each x 1
6 x SATA 6 Gb/s Ports; SATA Port Disable
Up to 6 Gb/s
Up to 6 x USB 3.2 Gen 2x1 Ports; Up to 10 x USB 3.2 Gen 1x1 Ports; 14 x USB 2.0 Ports
Intel® 2.5G Base-T MAC/PHY Ethernet
Intel® Integrated 10/100/1000 MAC
PCIe* x 1
SMBus
Intel® Ethernet Connection
Intel® Z490 Chipset
SPI
Intel® ME Firmware
Intel® Platform Trust Technology¹
Intel® Extreme Tuning Utility Support
Intel® Optane™ Memory Support³
Intel® Smart Sound Technology¹
Intel® High Definition Audio¹
Intel® Rapid Storage Technology with RAID¹
Intel® Rapid Storage Technology for PCI Express* Storage¹
Intel® Wi-Fi 6 AX201 (CNVio) with Gig+
Optional

Hardware II

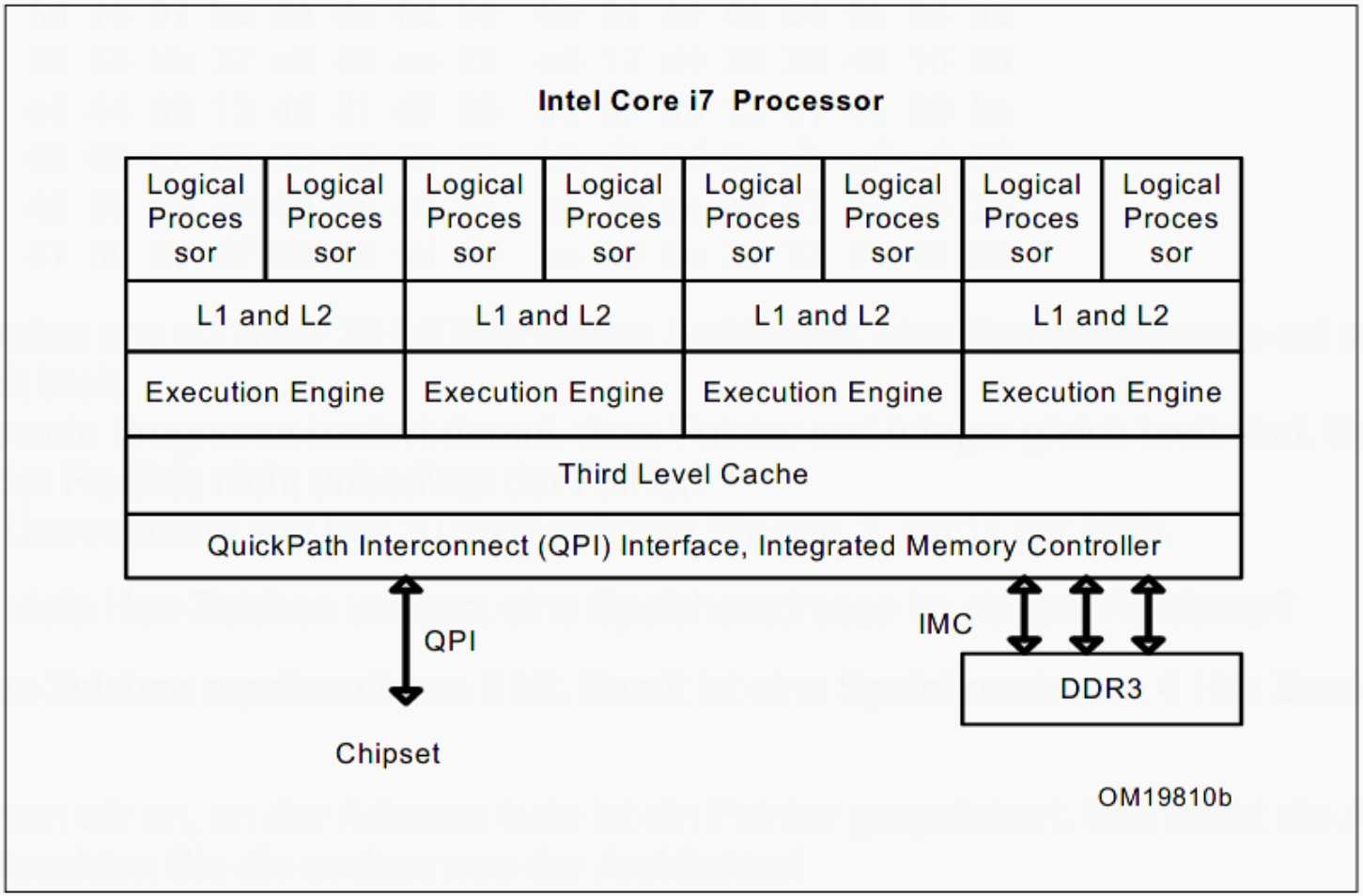


Figure 2-8. Intel Core i7 Processor

Hardware III

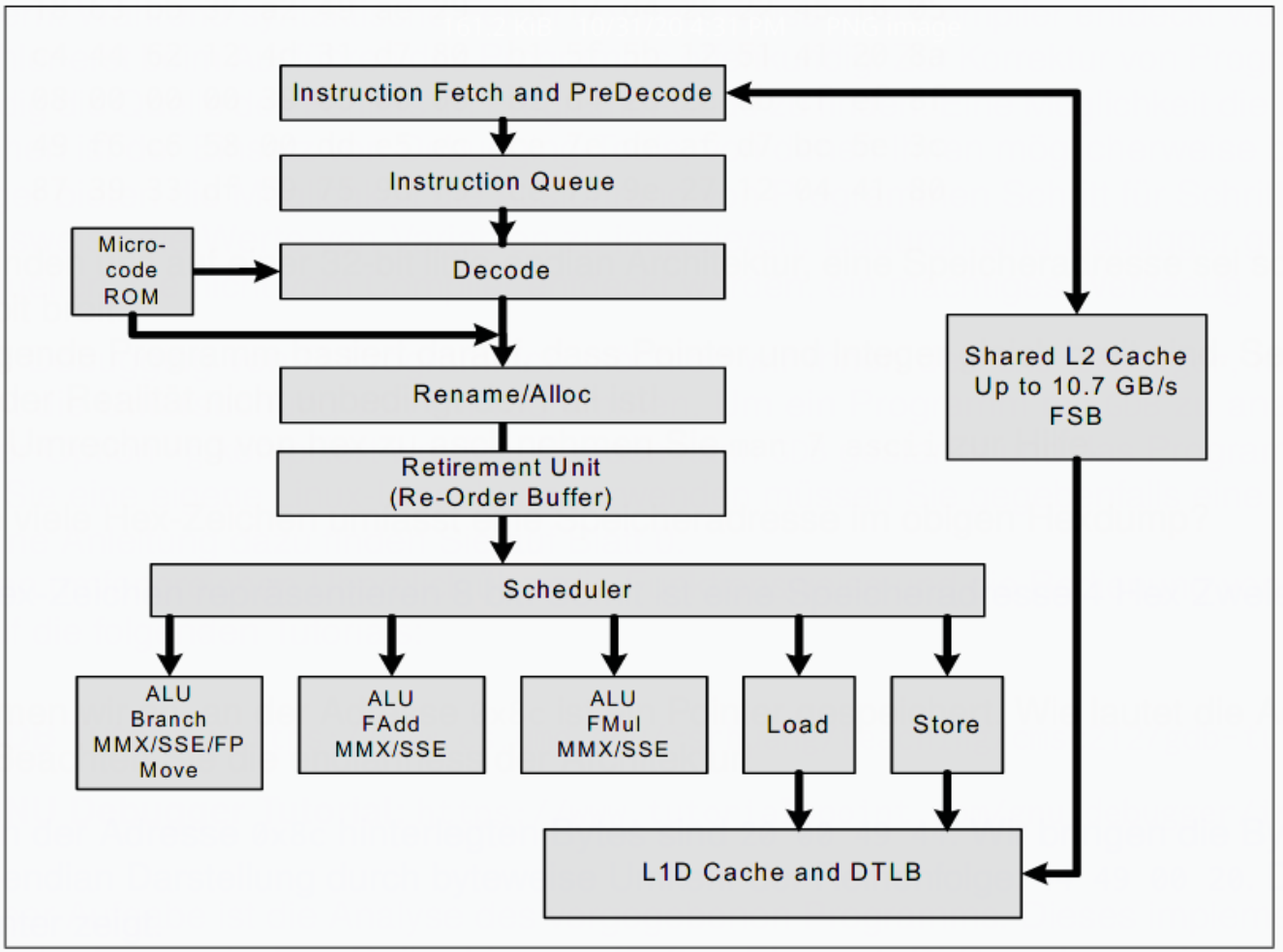


Figure 2-3. The Intel Core Microarchitecture Pipeline Functionality

Hardware IV

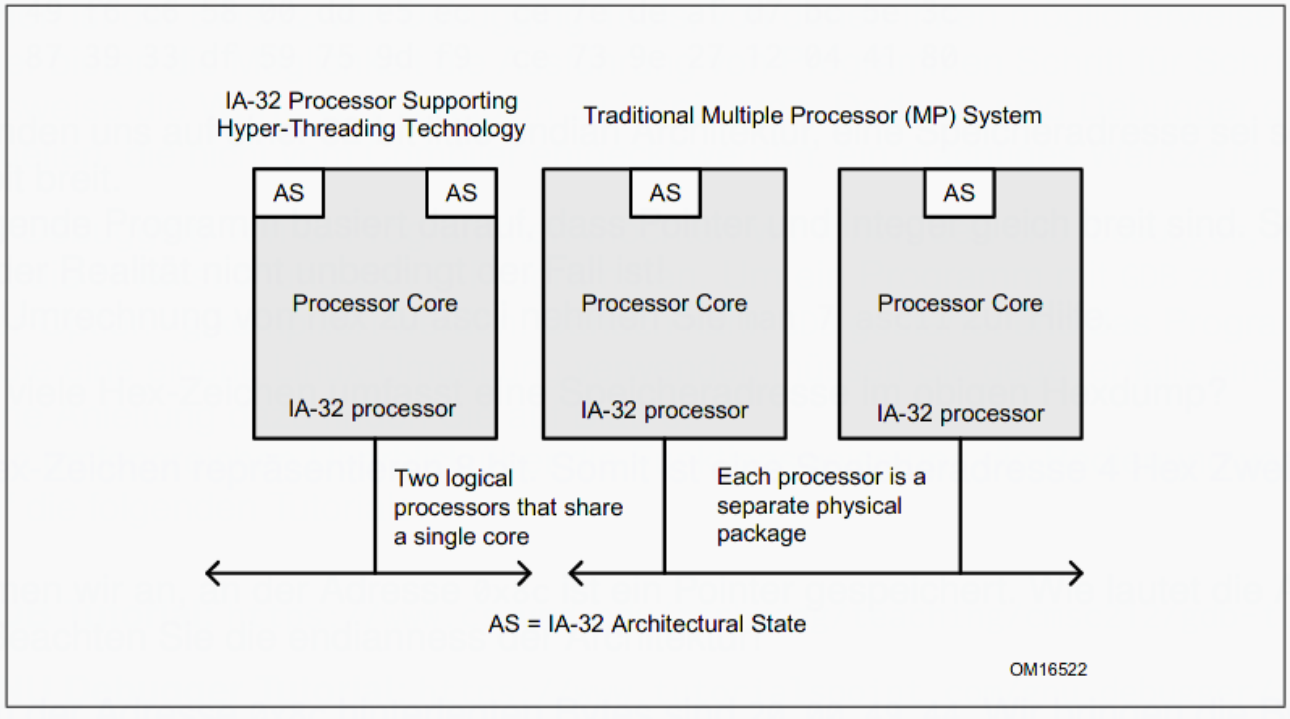


Figure 2-5. Comparison of an IA-32 Processor Supporting Hyper-Threading Technology and a Traditional Dual Processor System

Hardware V

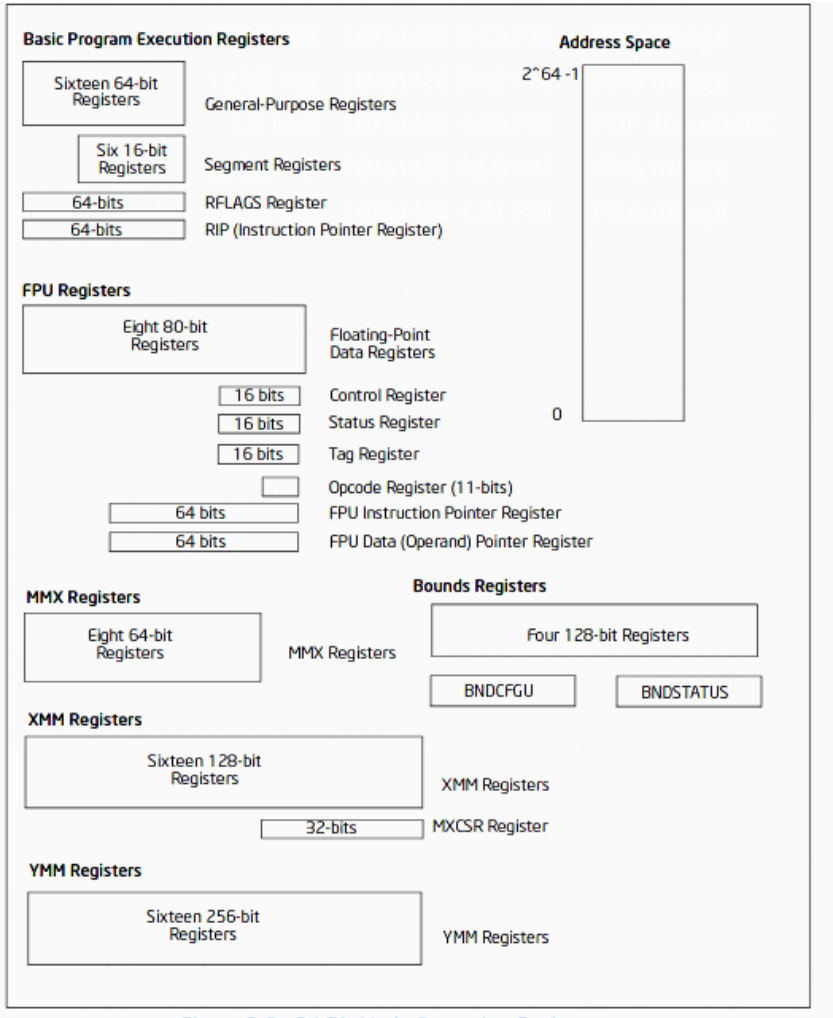


Figure 3-2. 64-Bit Mode Execution Environment

Hardware VI

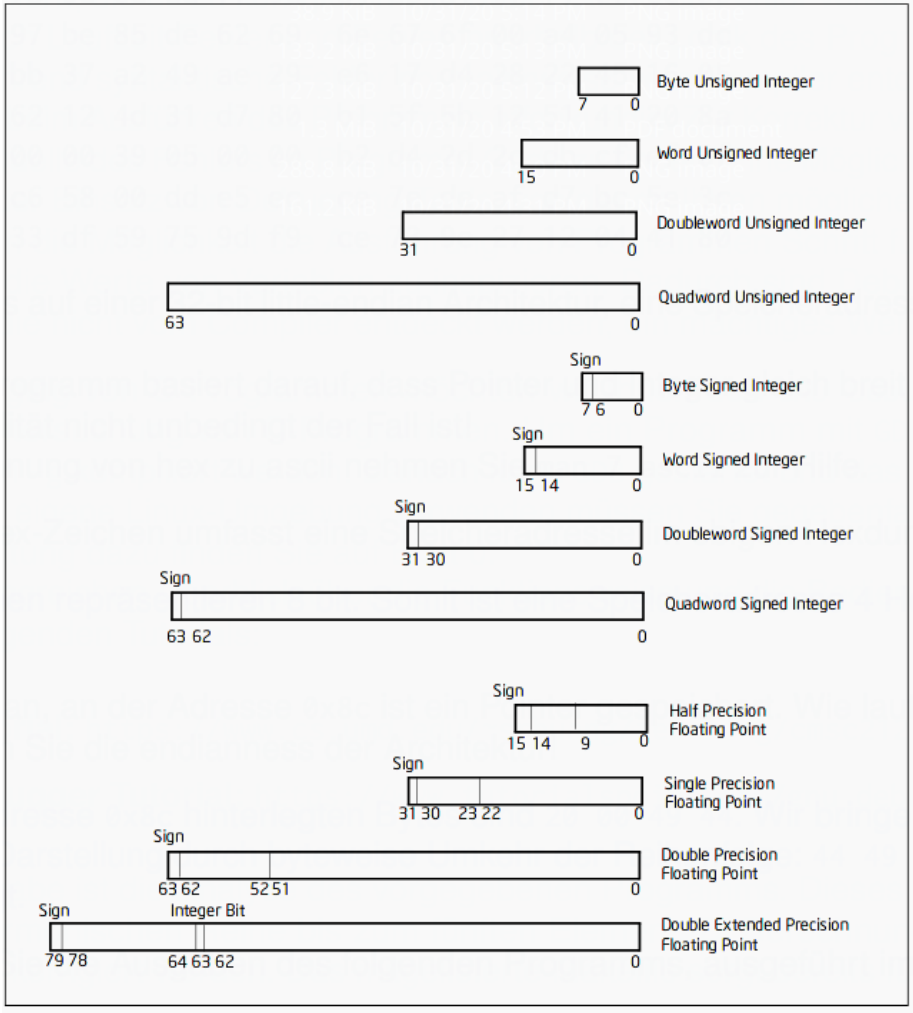
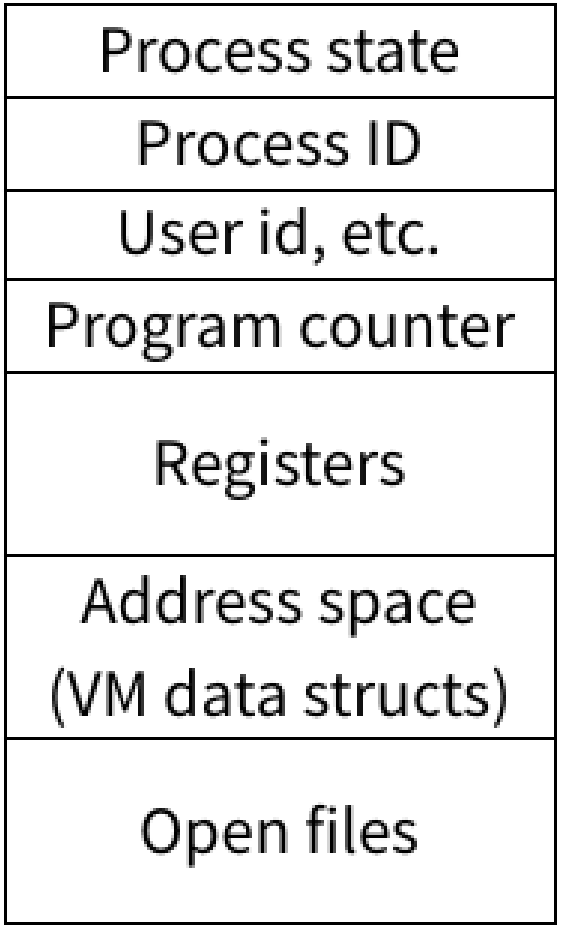


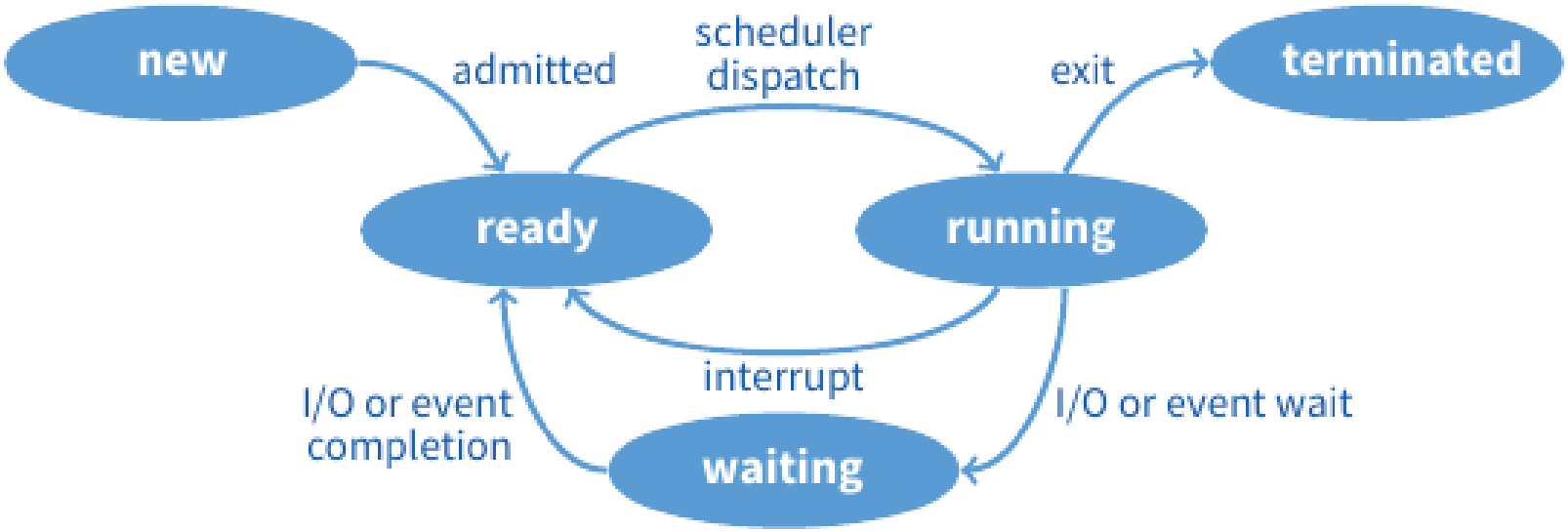
Figure 4-3. Numeric Data Types

Processes & Thread I

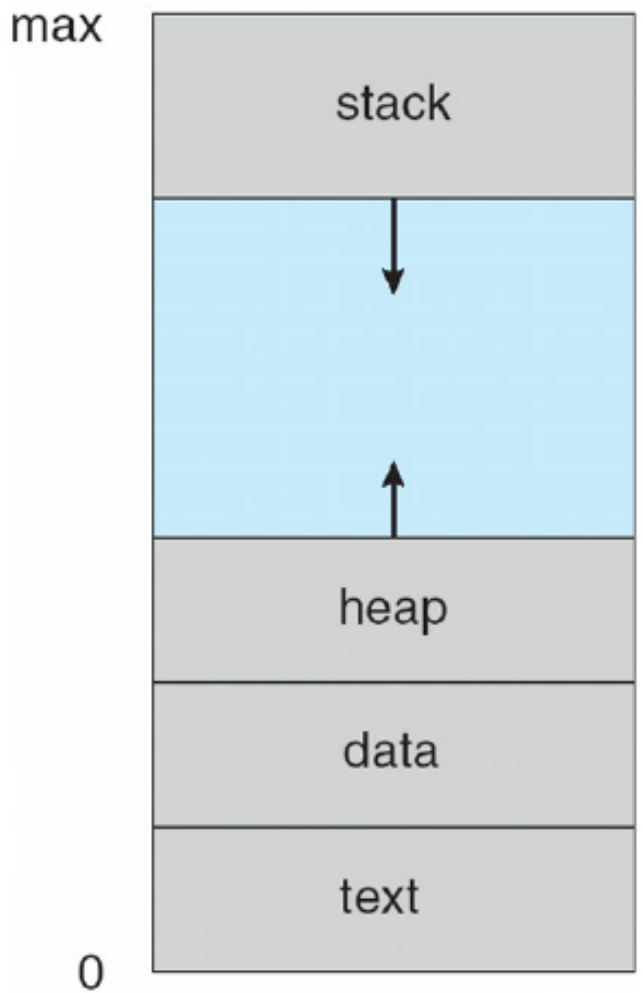


PCB

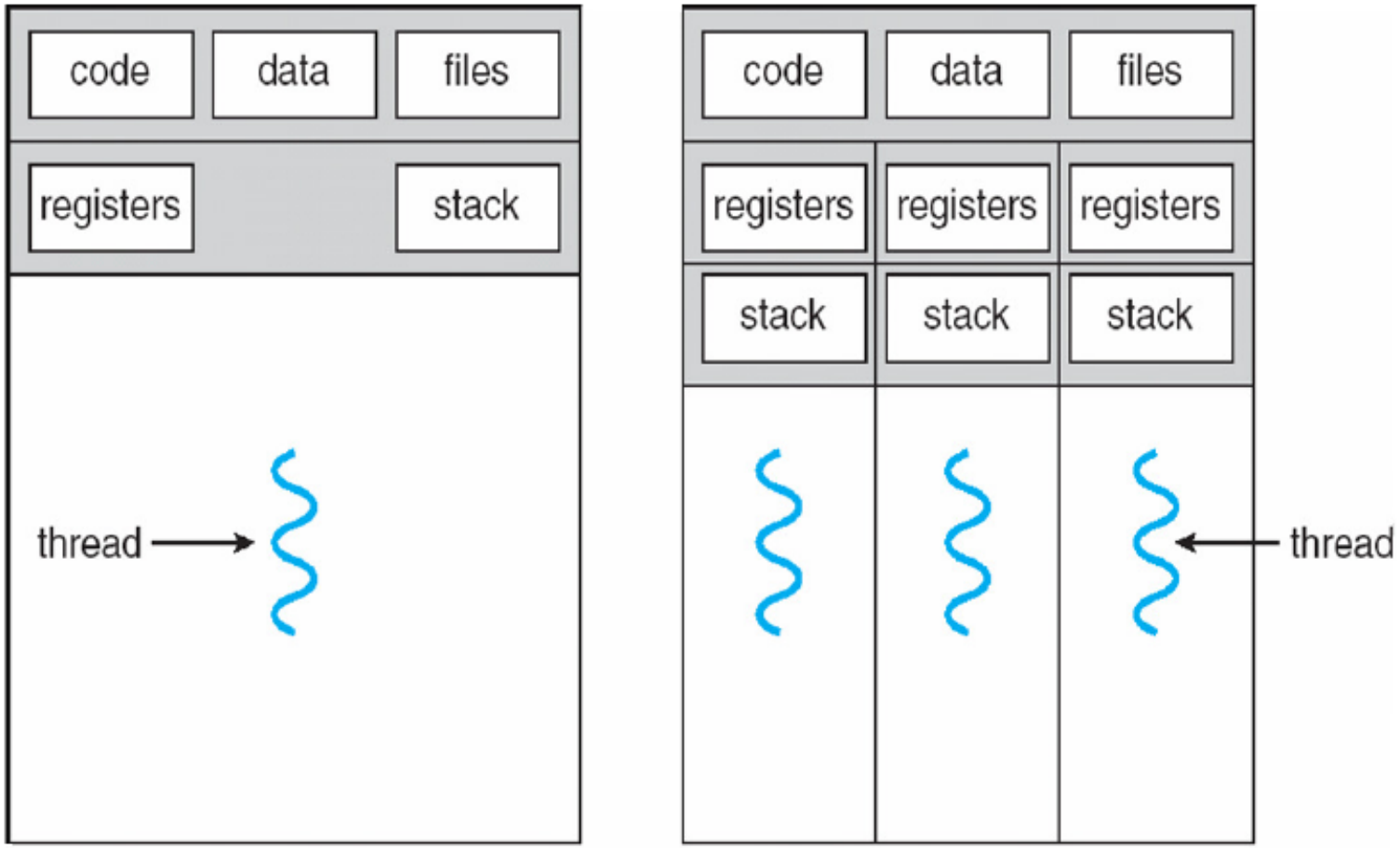
Processes & Thread II



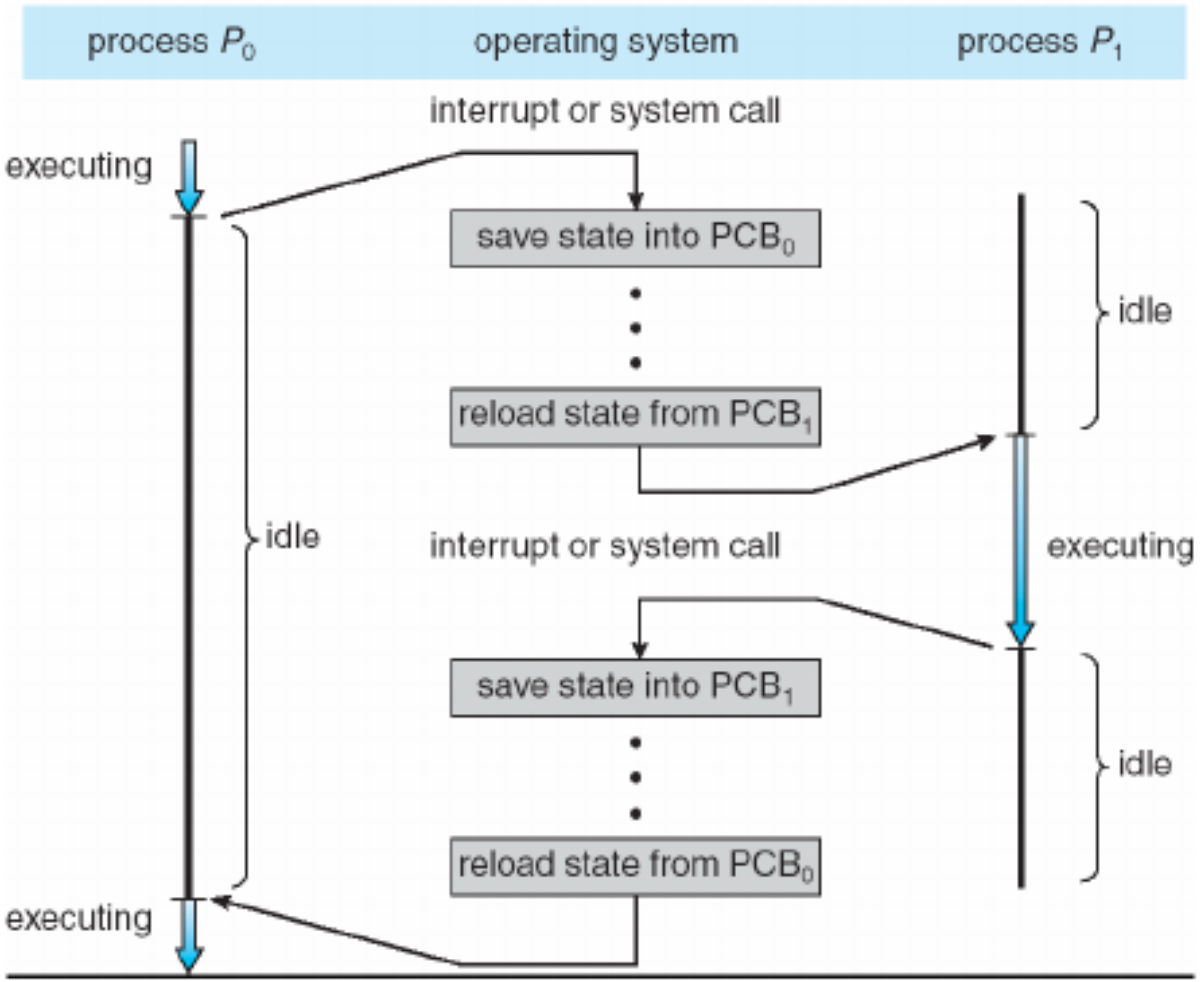
Processes & Thread III



Processes & Thread IV



Processes & Thread V



Scheduling I

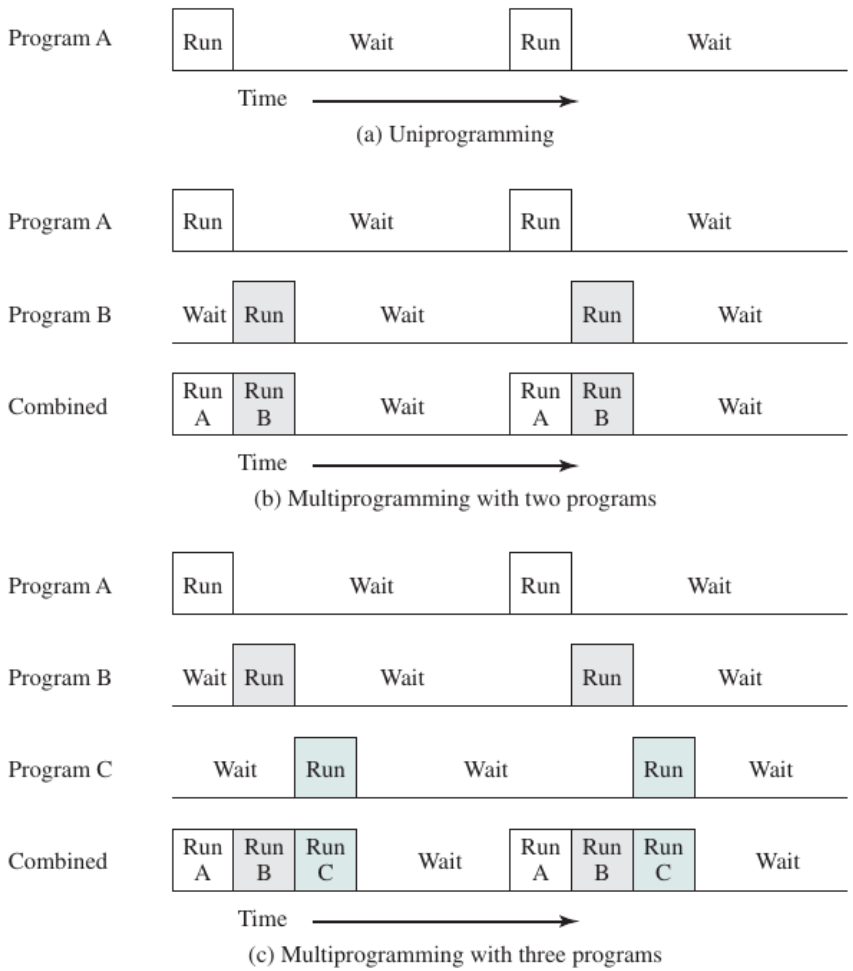


Figure 2.5 **Multiprogramming Example**

Scheduling II

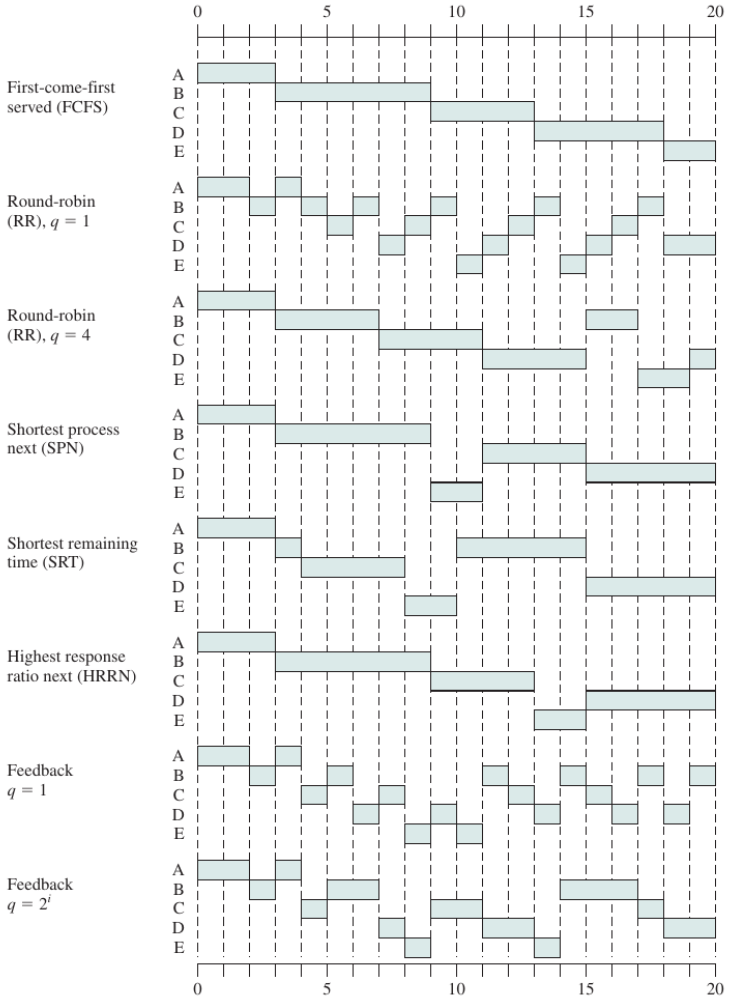
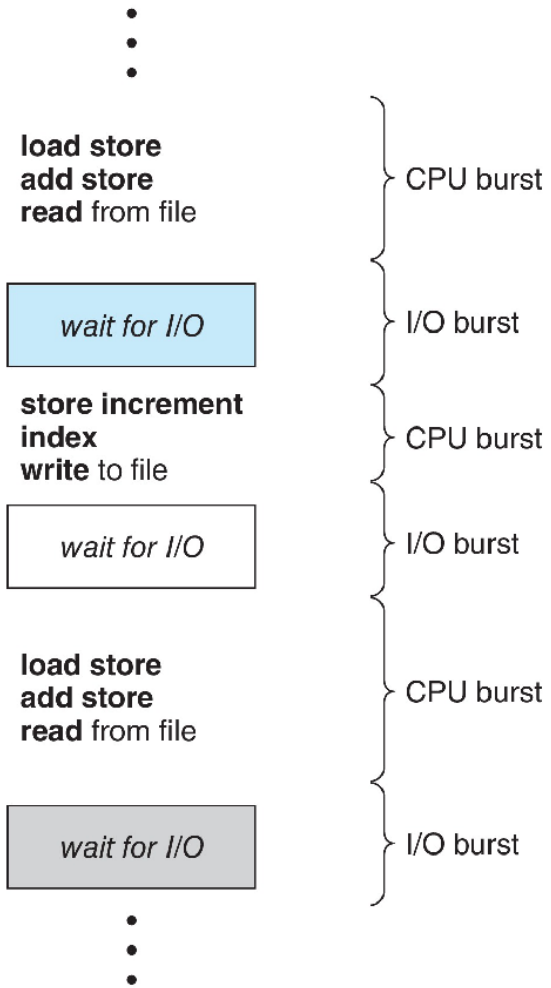
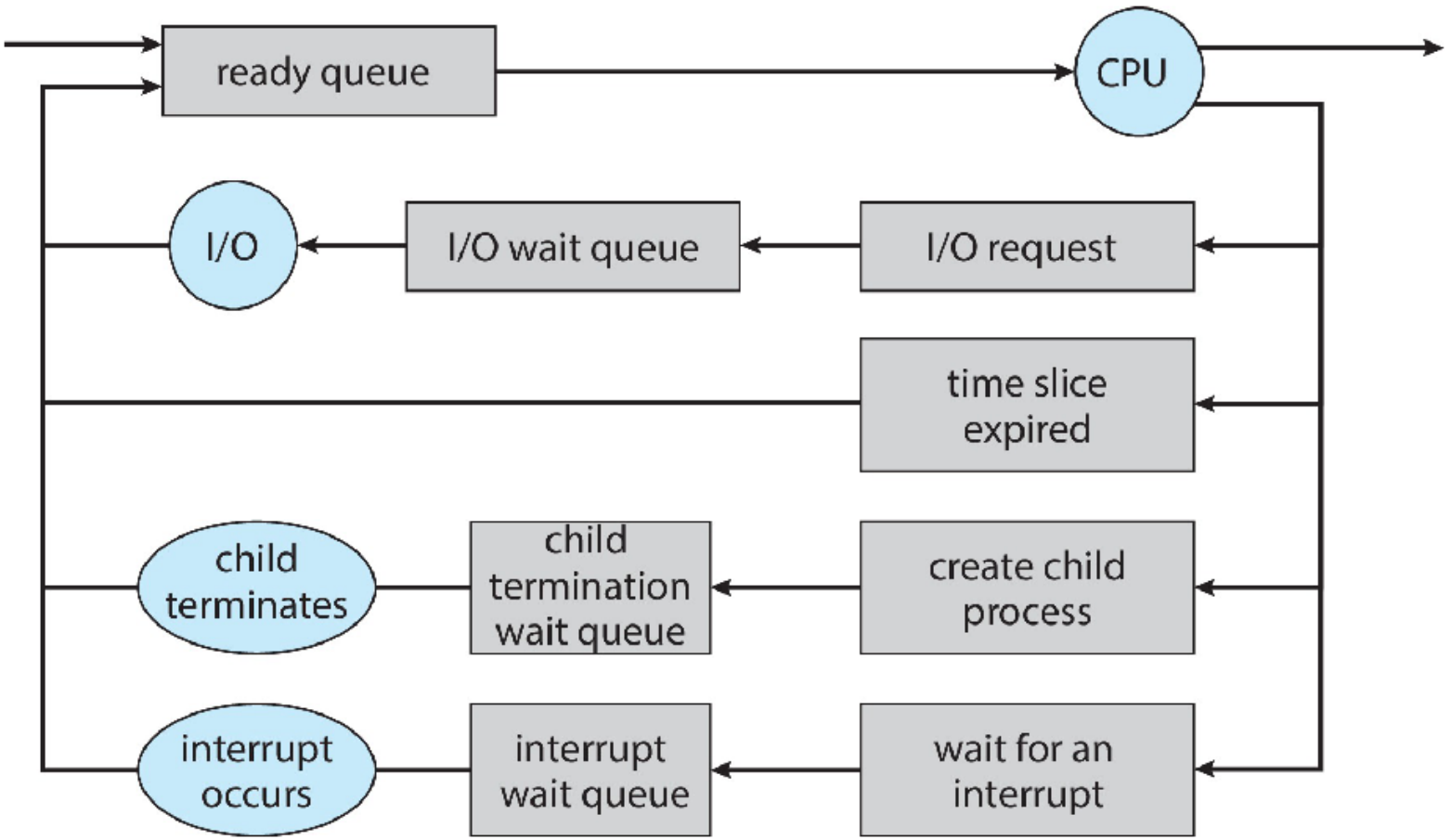


Figure 9.5 **A Comparison of Scheduling Policies**

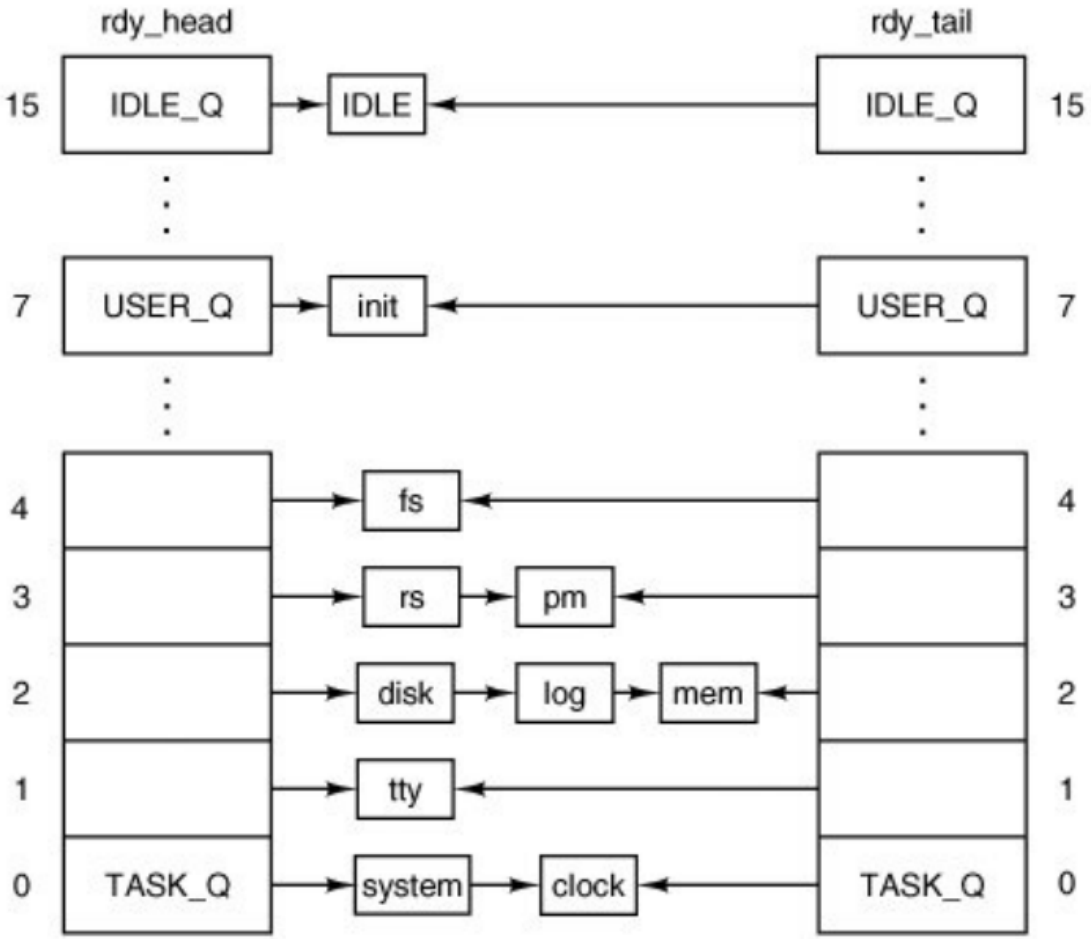
Scheduling III



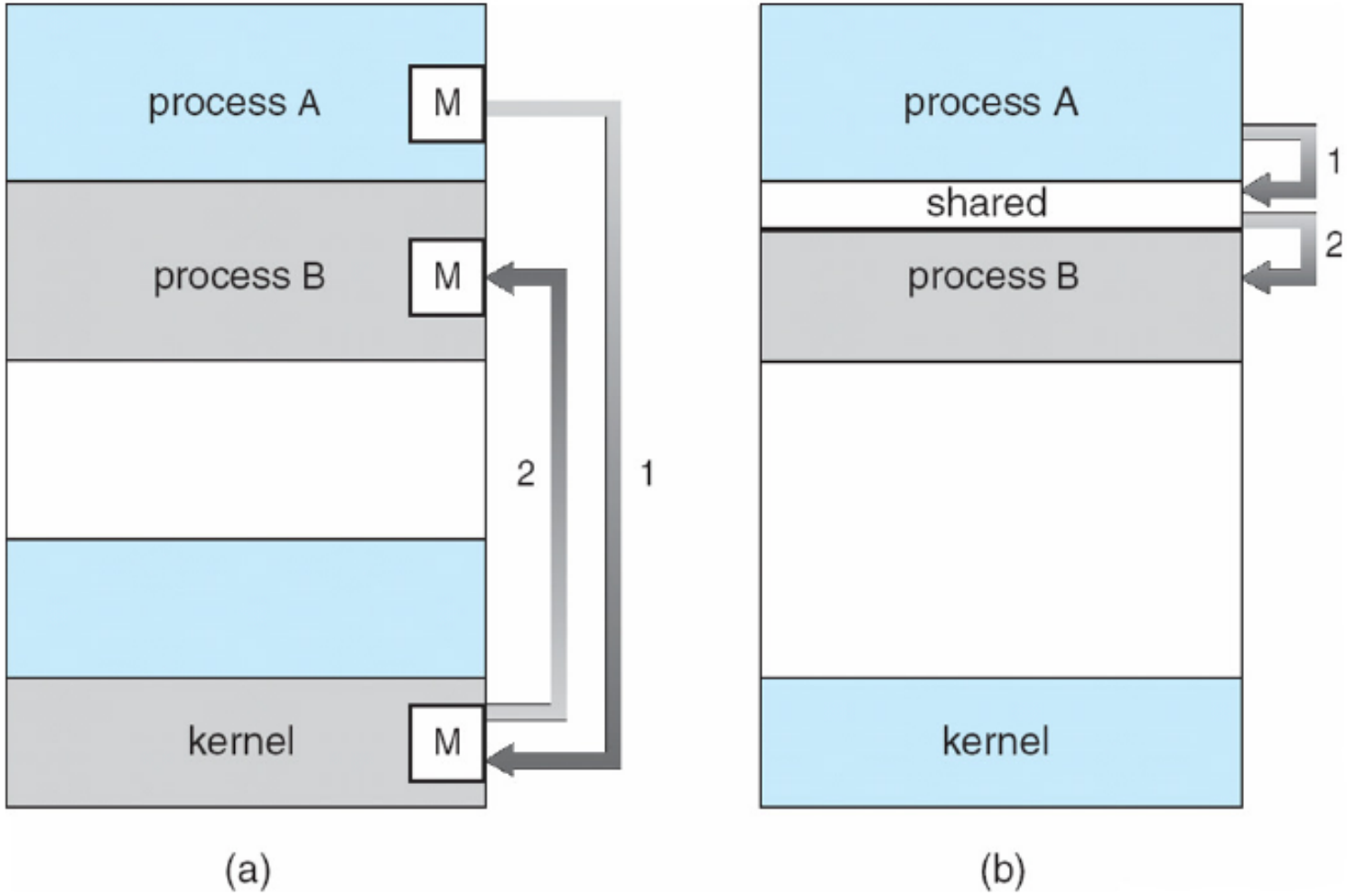
Scheduling IV



Scheduling V



Memory Management I



Memory Management II

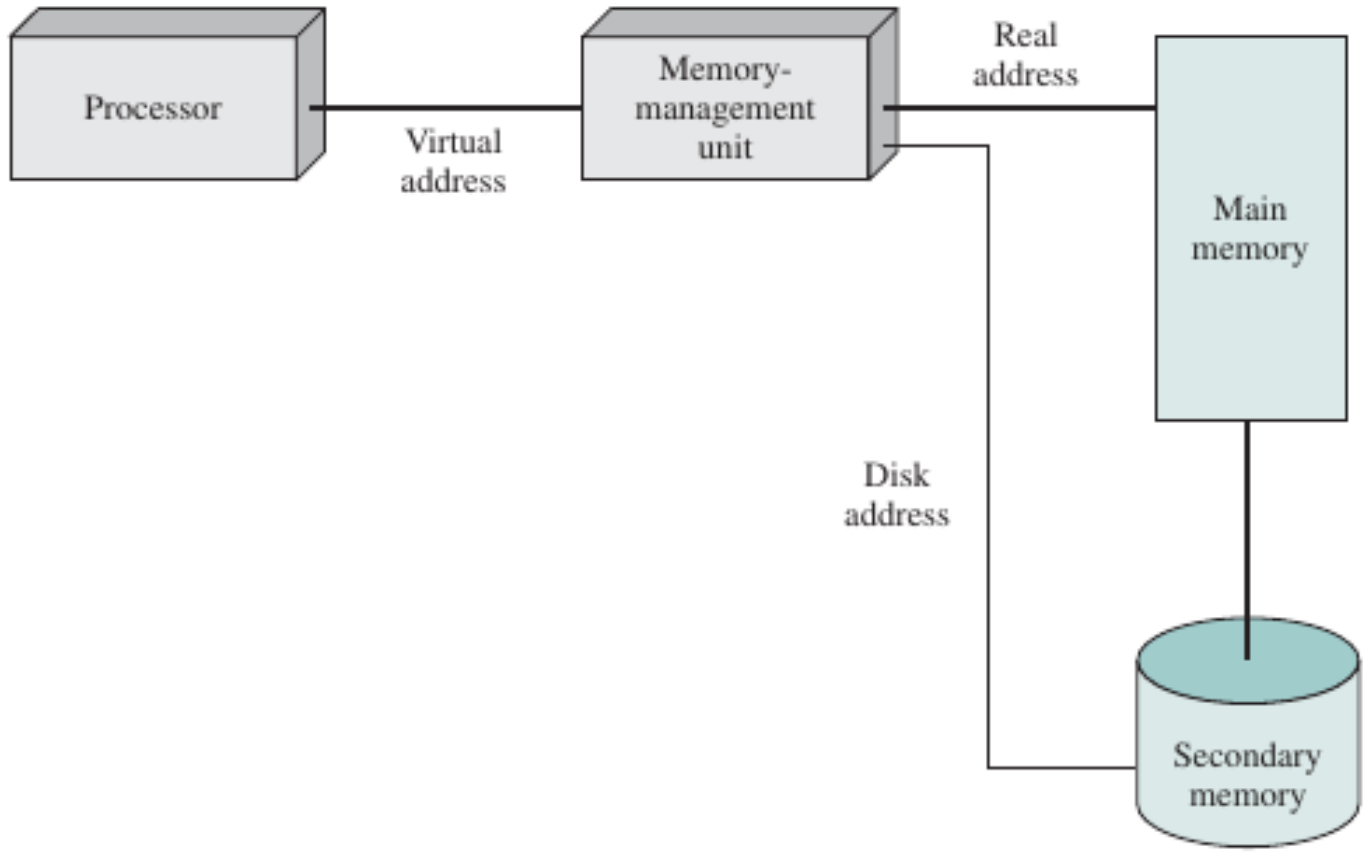


Figure 2.10 **Virtual Memory Addressing**

Memory Management III

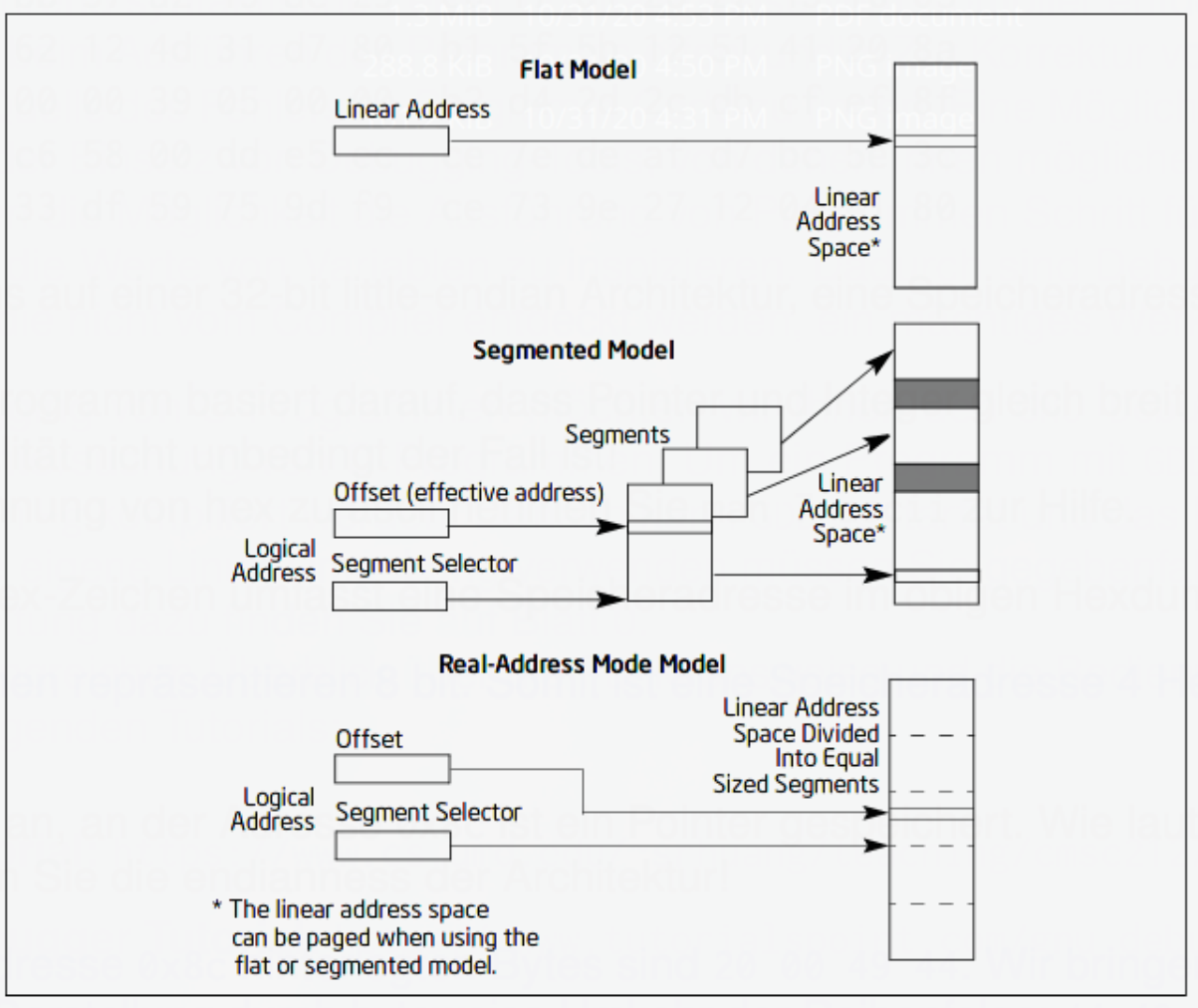
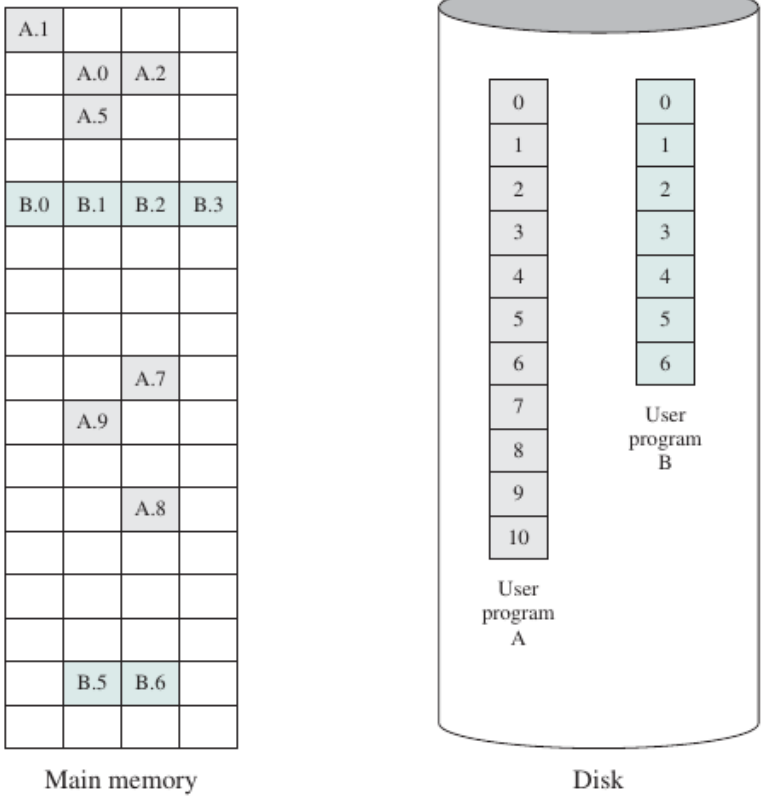


Figure 3-3. Three Memory Management Models

Memory Management IV



Main memory consists of a number of fixed-length frames, each equal to the size of a page. For a program to execute, some or all of its pages must be in main memory.

Secondary memory (disk) can hold many fixed-length pages. A user program consists of some number of pages. Pages for all programs plus the operating system are on disk, as are files.

Figure 2.9 Virtual Memory Concepts

Memory Management V

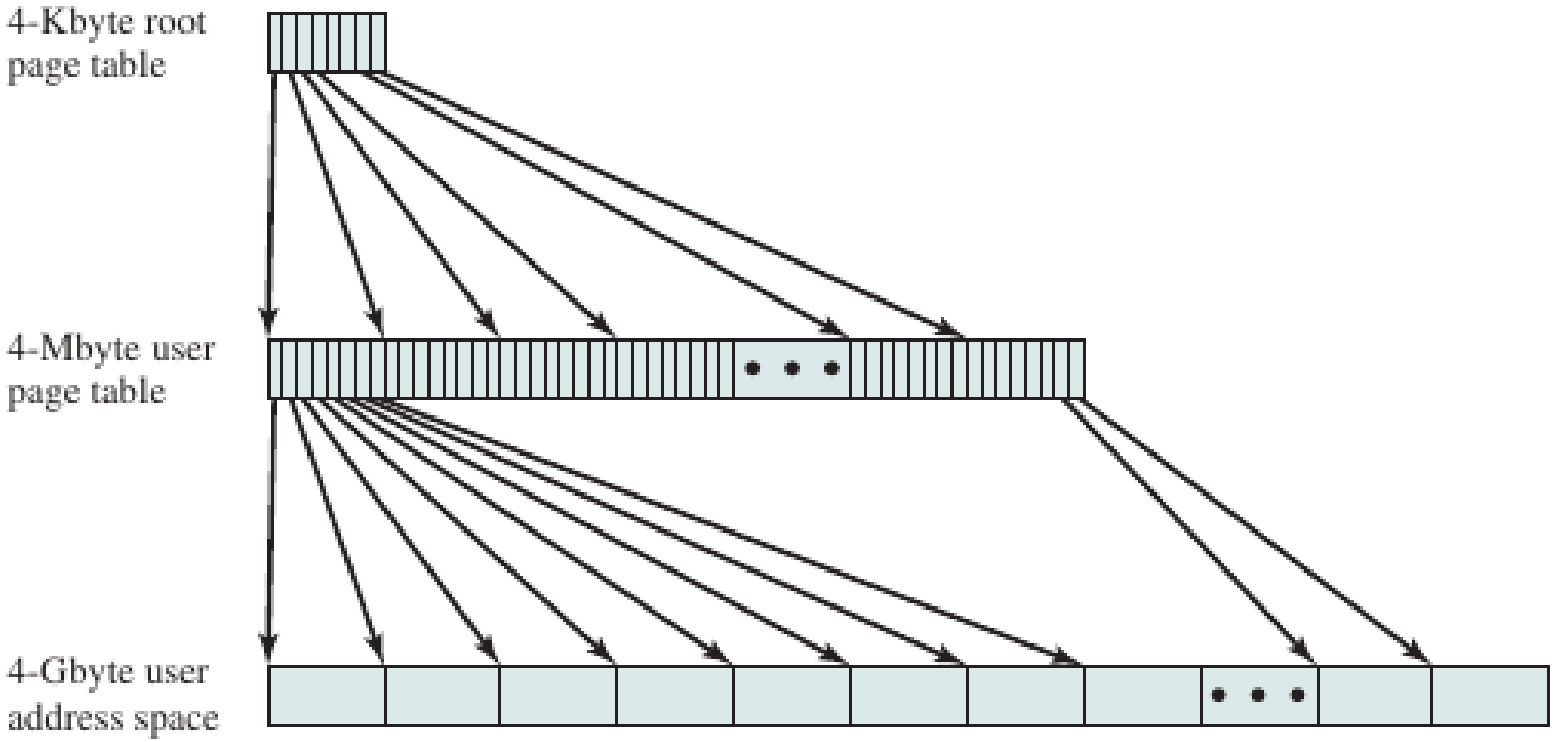


Figure 8.4 A Two-Level Hierarchical Page Table

Memory Management VI

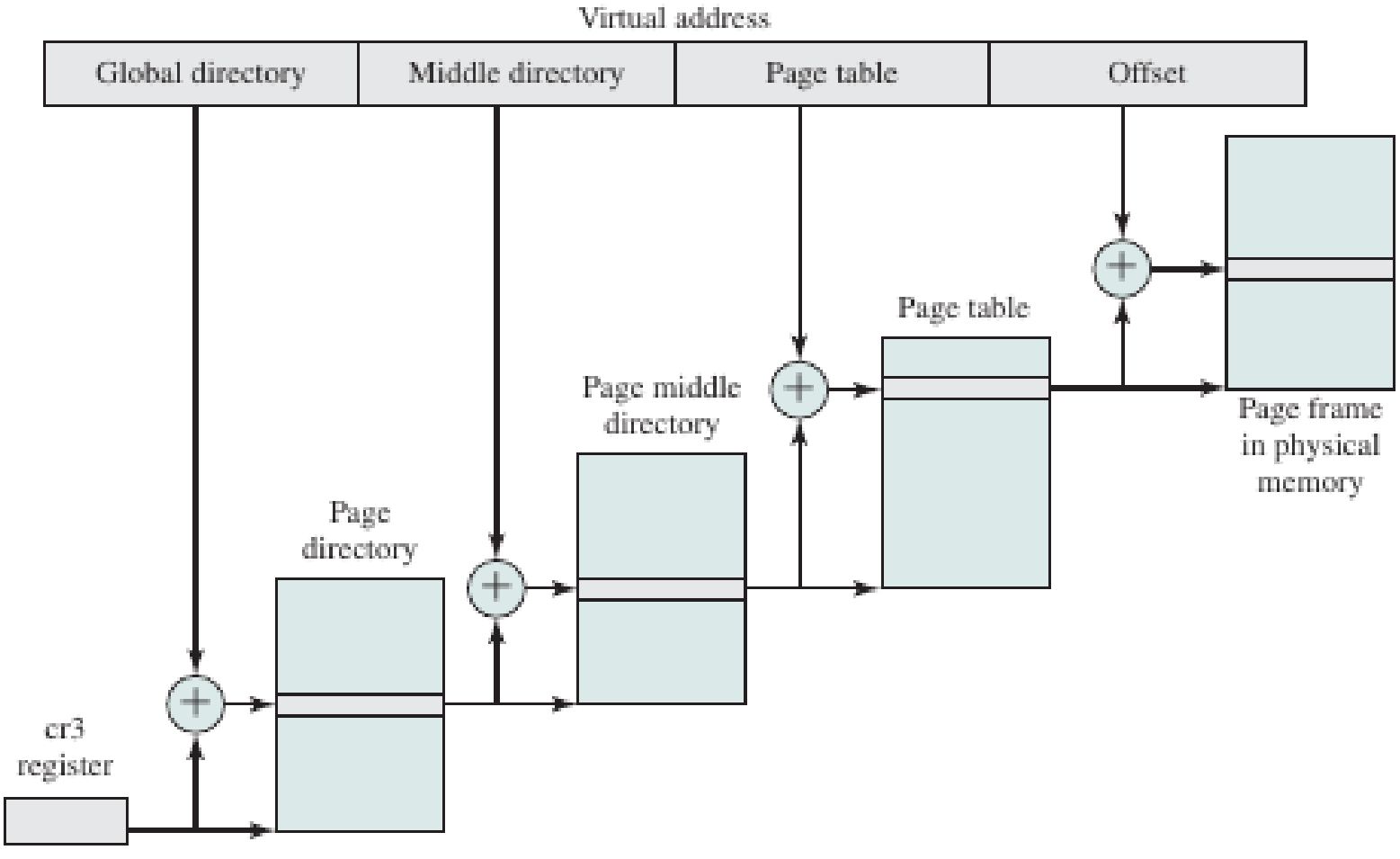


Figure 8.25 Address Translation in Linux Virtual Memory Scheme

Memory Management VII

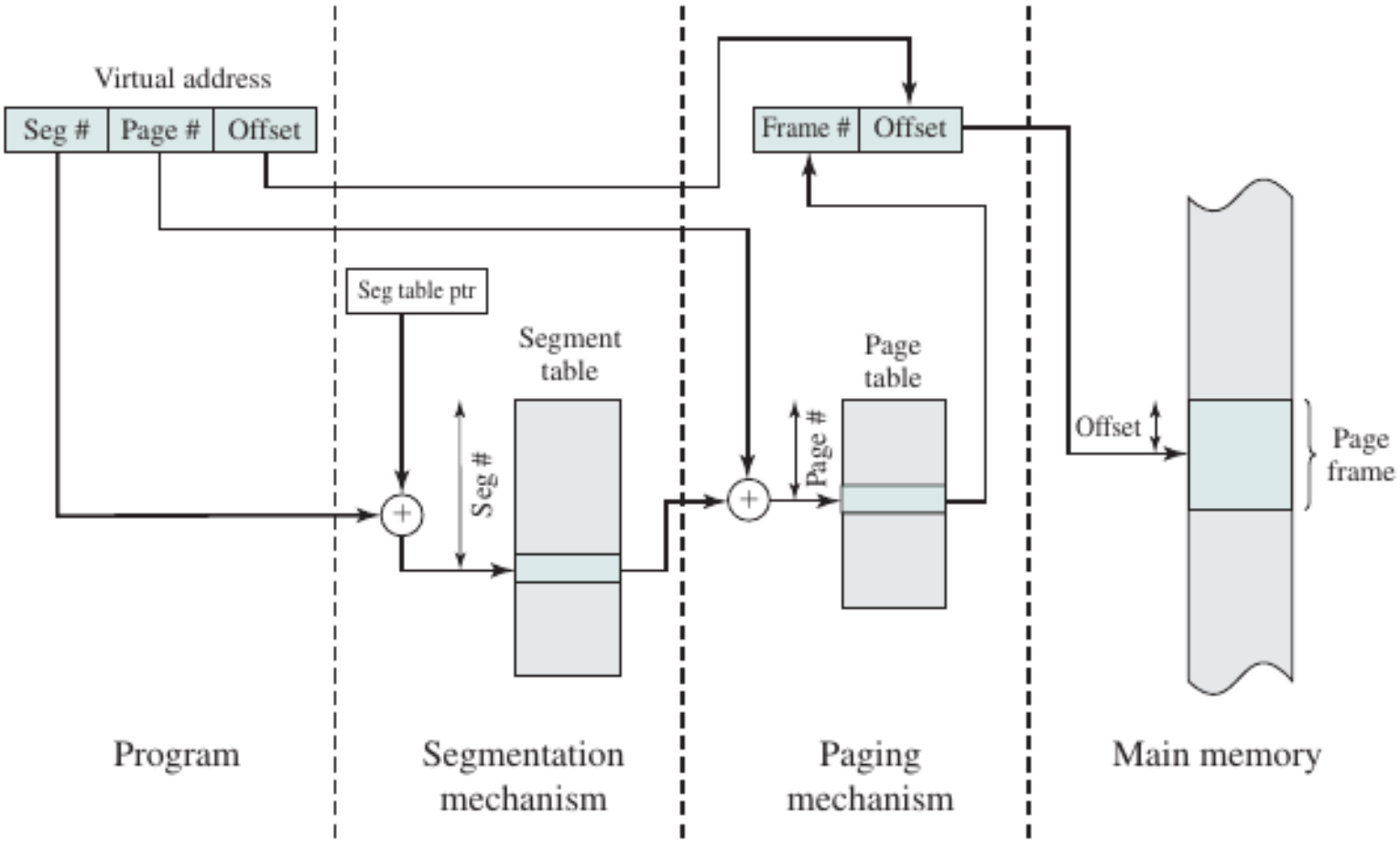
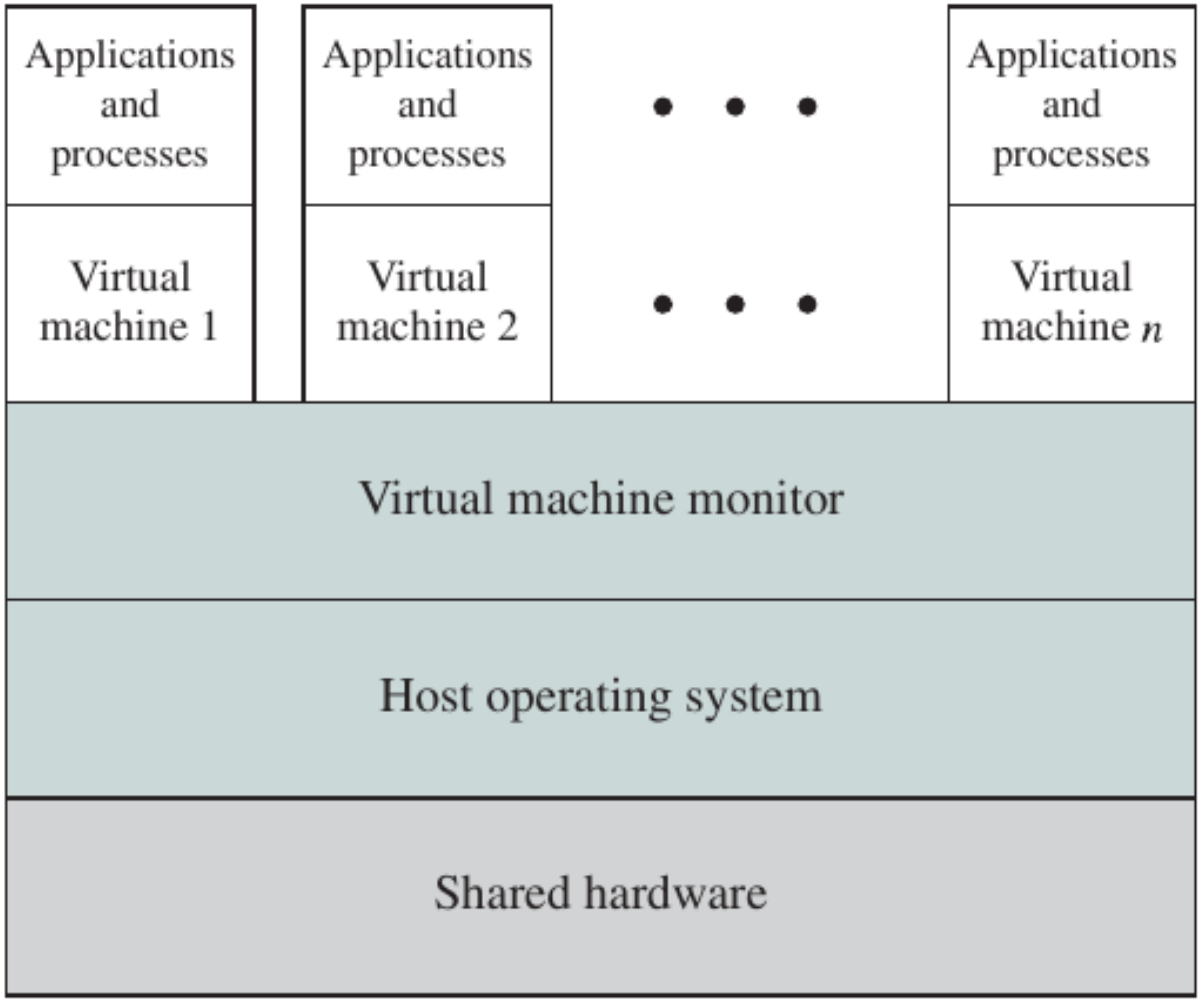


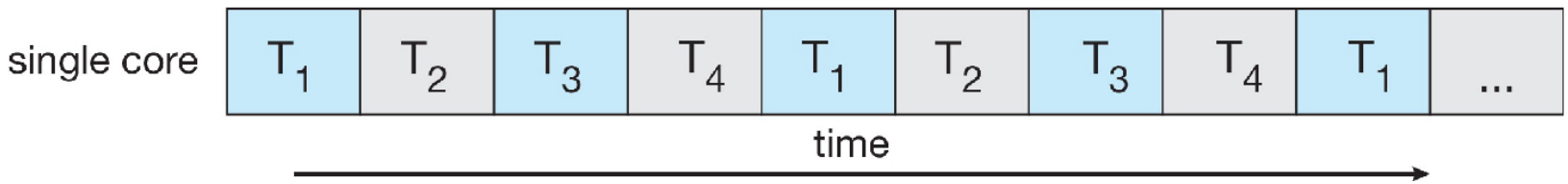
Figure 8.13 **Address Translation in a Segmentation/Paging System**

Virtual Systems

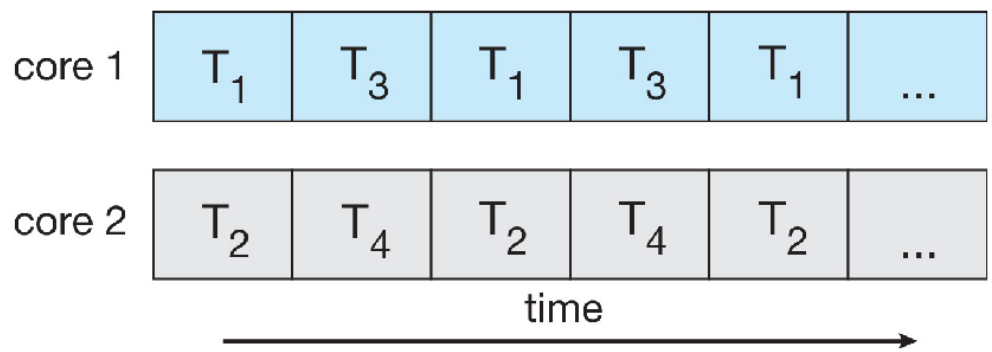


Concurrency & Parallelism I

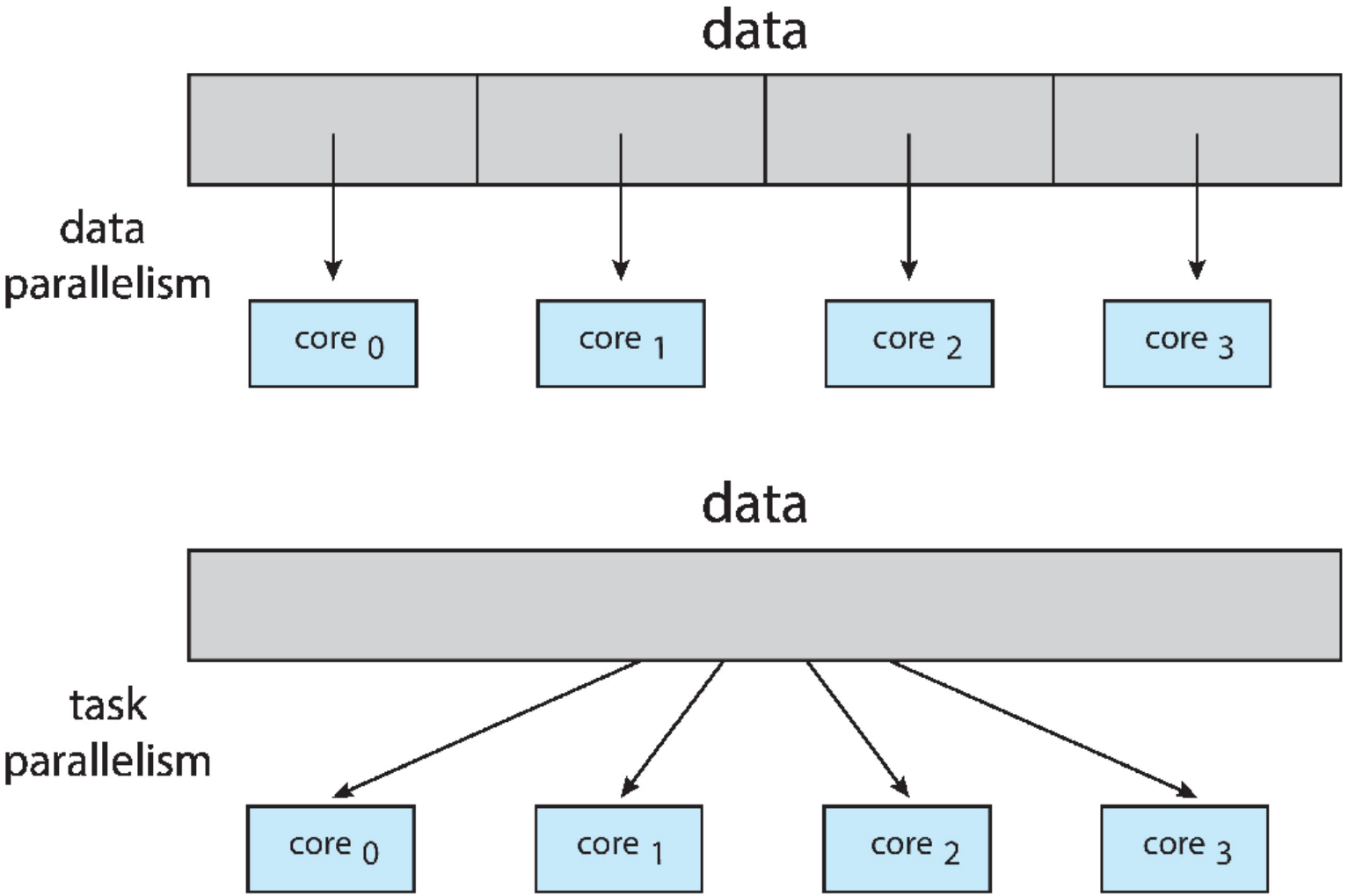
- **Concurrent execution on single-core system:**



- **Parallelism on a multi-core system:**



Concurrency & Parallelism II



Race Condition

Process P1

```
•
chin = getchar();
•
chout = chin;
putchar(chout);
•
•
```

Process P2

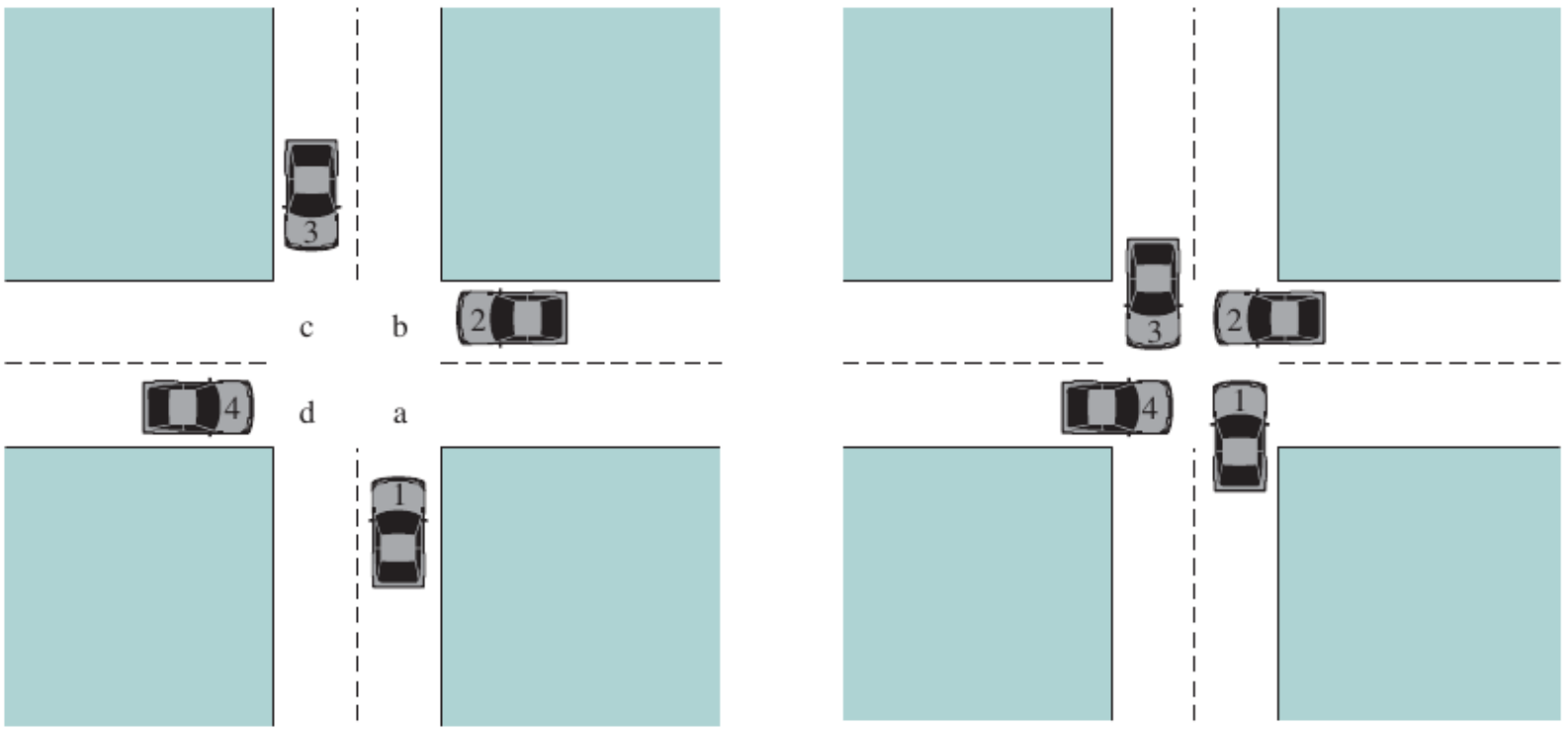
```
•
•
chin = getchar();
chout = chin;
•
putchar(chout);
•
```

Synchronization

Table 5.3 Common Concurrency Mechanisms

Semaphore	An integer value used for signaling among processes. Only three operations may be performed on a semaphore, all of which are atomic: initialize, decrement, and increment. The decrement operation may result in the blocking of a process, and the increment operation may result in the unblocking of a process. Also known as a **counting semaphore** or a **general semaphore.**
Binary Semaphore	A semaphore that takes on only the values 0 and 1.
Mutex	Similar to a binary semaphore. A key difference between the two is that the process that locks the mutex (sets the value to zero) must be the one to unlock it (sets the value to 1).
Condition Variable	A data type that is used to block a process or thread until a particular condition is true.
Monitor	A programming language construct that encapsulates variables, access procedures, and initialization code within an abstract data type. The monitor's variable may only be accessed via its access procedures and only one process may be actively accessing the monitor at any one time. The access procedures are *critical sections*. A monitor may have a queue of processes that are waiting to access it.
Event Flags	A memory word used as a synchronization mechanism. Application code may associate a different event with each bit in a flag. A thread can wait for either a single event or a combination of events by checking one or multiple bits in the corresponding flag. The thread is blocked until all of the required bits are set (AND) or until at least one of the bits is set (OR).
Mailboxes/Messages	A means for two processes to exchange information and that may be used for synchronization.
Spinlocks	Mutual exclusion mechanism in which a process executes in an infinite loop waiting for the value of a lock variable to indicate availability.

Deadlocks I



(a) Deadlock possible (b) Deadlock

Figure 6.1 Illustration of Deadlock

Deadlocks II

Approach	Resource Allocation Policy	Different Schemes	Major Advantages	Major Disadvantages
Prevention	Conservative; undercommits resources	Requesting all resources at once	• Works well for processes that perform a single burst of activity • No preemption necessary	• Inefficient • Delays process initiation • Future resource requirements must be known by processes
		Preemption	• Convenient when applied to resources whose state can be saved and restored easily	• Preempts more often than necessary
		Resource ordering	• Feasible to enforce via compile-time checks • Needs no run-time computation since problem is solved in system design	• Disallows incremental resource requests
Avoidance	Midway between that of detection and prevention	Manipulate to find at least one safe path	• No preemption necessary	• Future resource requirements must be known by OS • Processes can be blocked for long periods
Detection	Very liberal; requested resources are granted where possible	Invoke periodically to test for deadlock	• Never delays process initiation • Facilitates online handling	• Inherent preemption losses

Devices & Drivers I

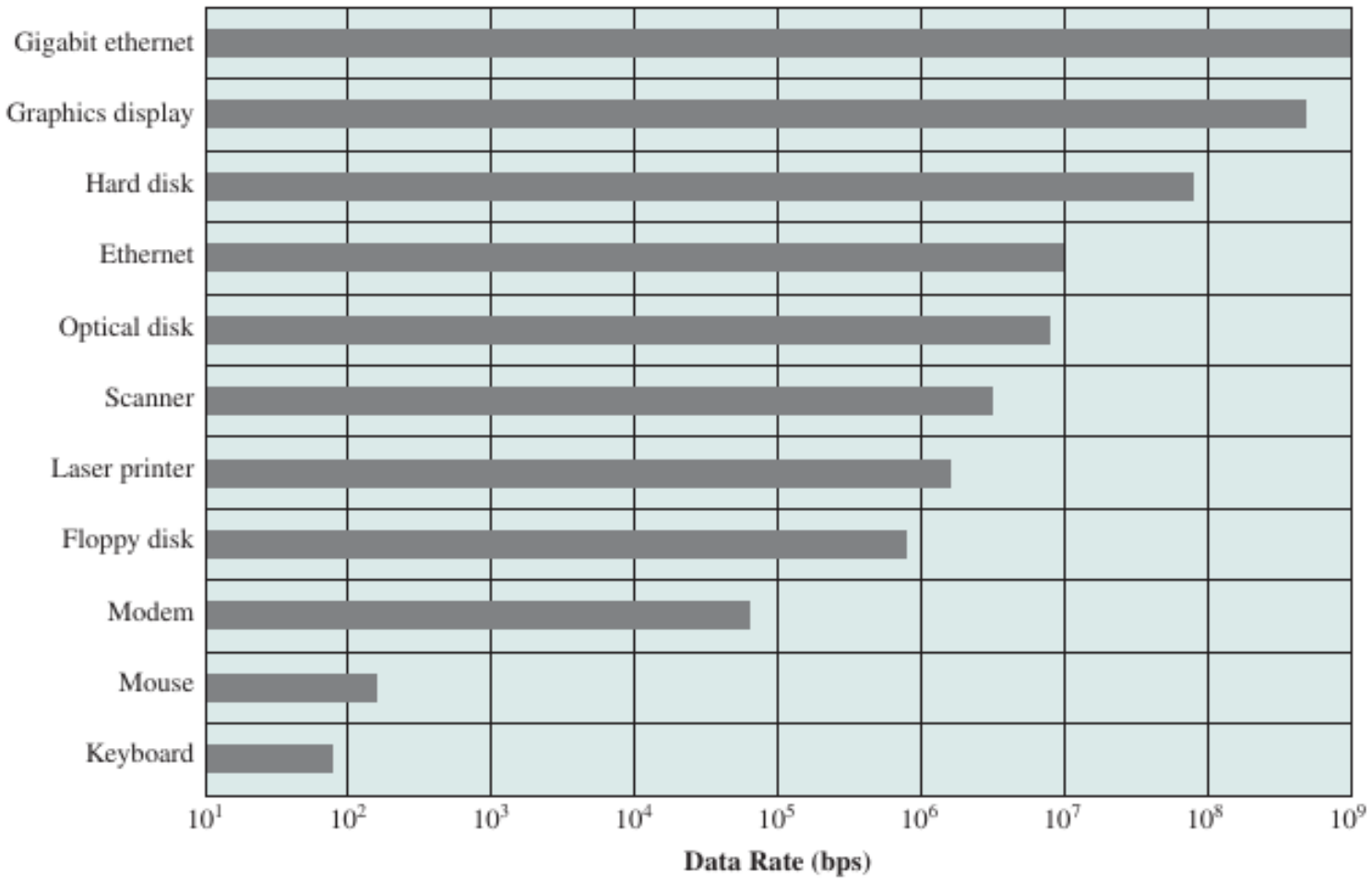


Figure 11.1 **Typical I/O Device Data Rates**

Devices & Drivers II

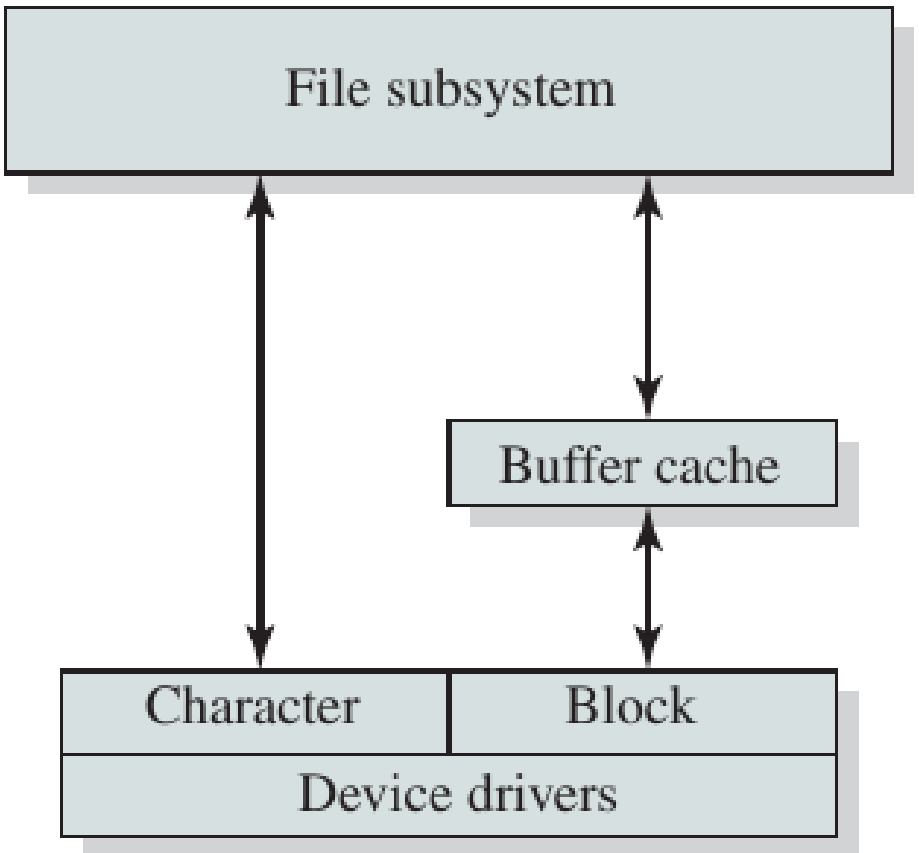
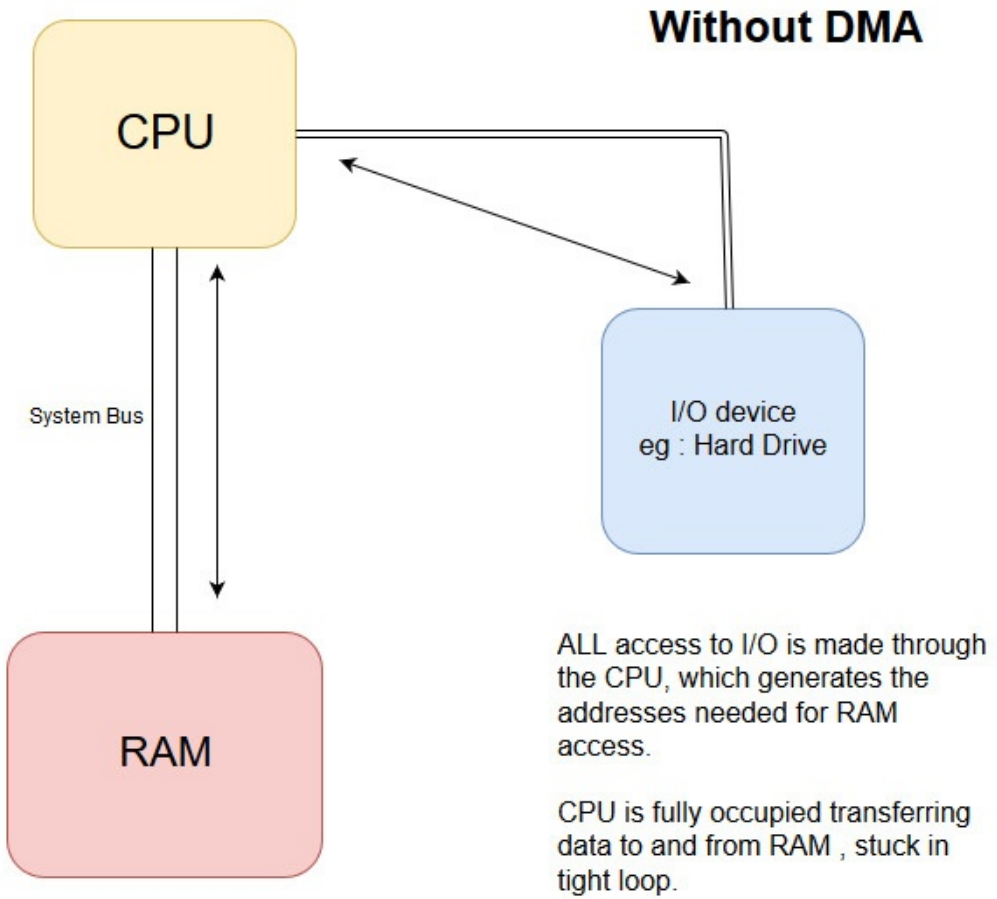


Figure 11.12 UNIX I/O Structure

Devices & Drivers III

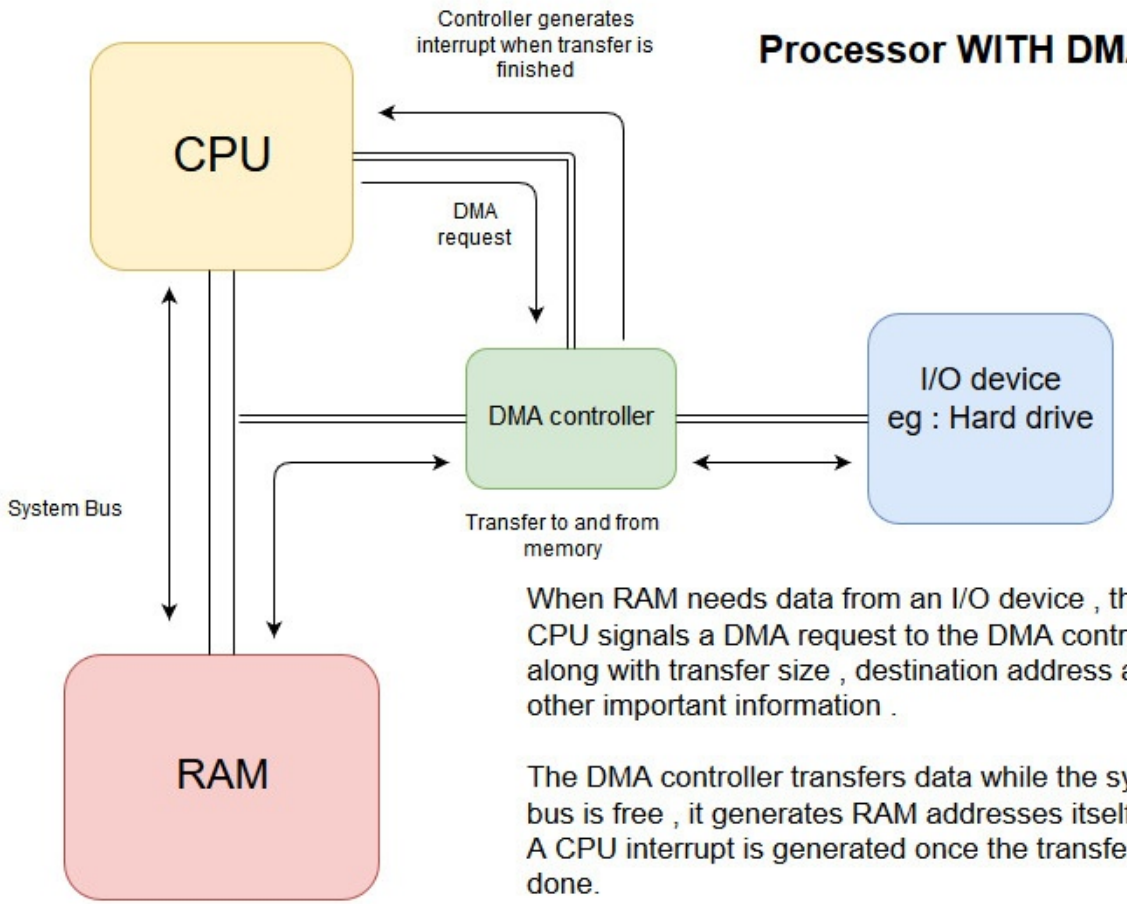


ALL access to I/O is made through the CPU, which generates the addresses needed for RAM access.

CPU is fully occupied transferring data to and from RAM , stuck in tight loop.

Problem : no useful work is being done here !

Devices & Drivers IV

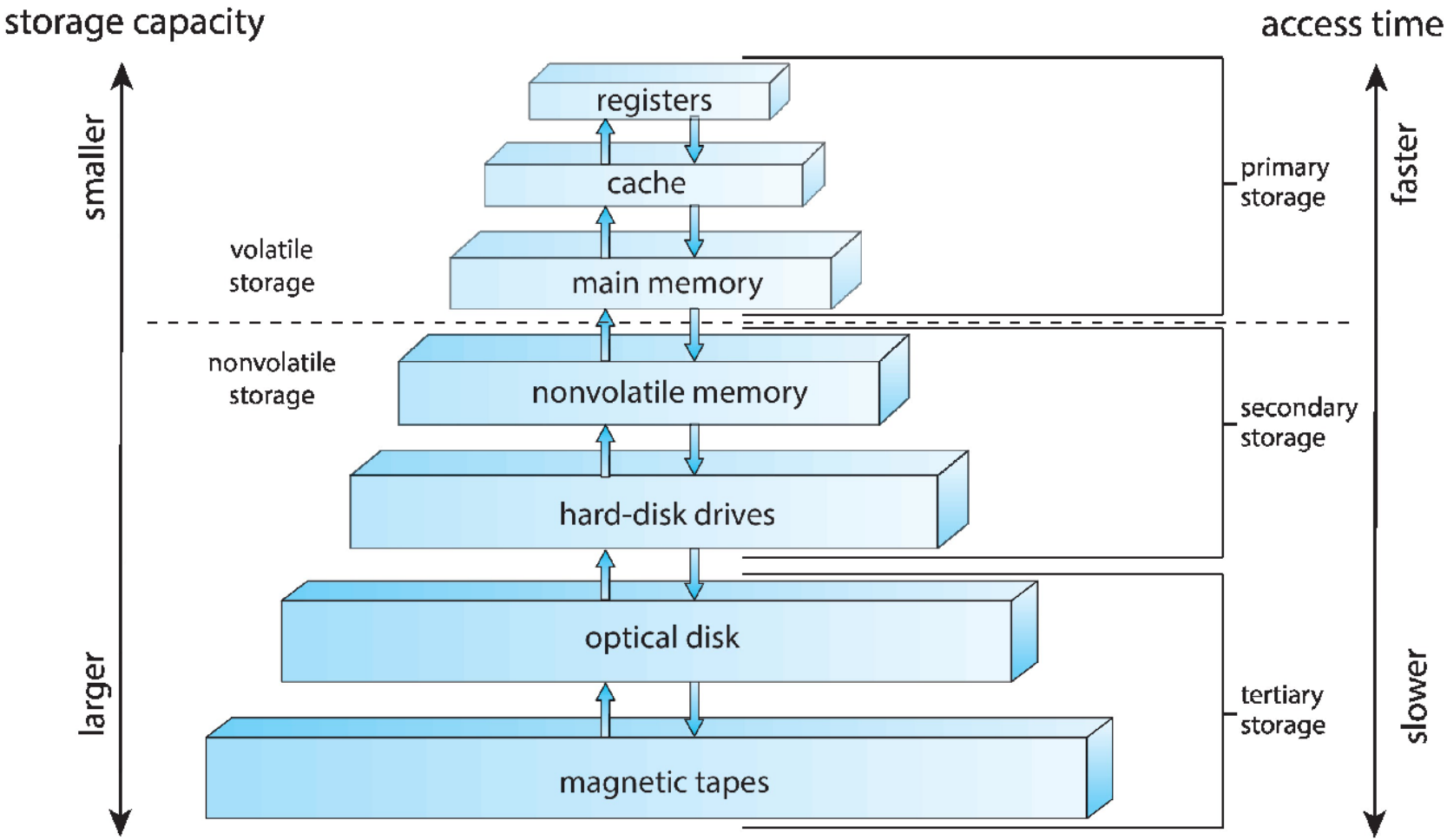


When RAM needs data from an I/O device , the CPU signals a DMA request to the DMA controller , along with transfer size , destination address and other important information .

The DMA controller transfers data while the system bus is free , it generates RAM addresses itself.
A CPU interrupt is generated once the transfer is done.

Result : CPU is free to do other things !

Devices & Drivers V

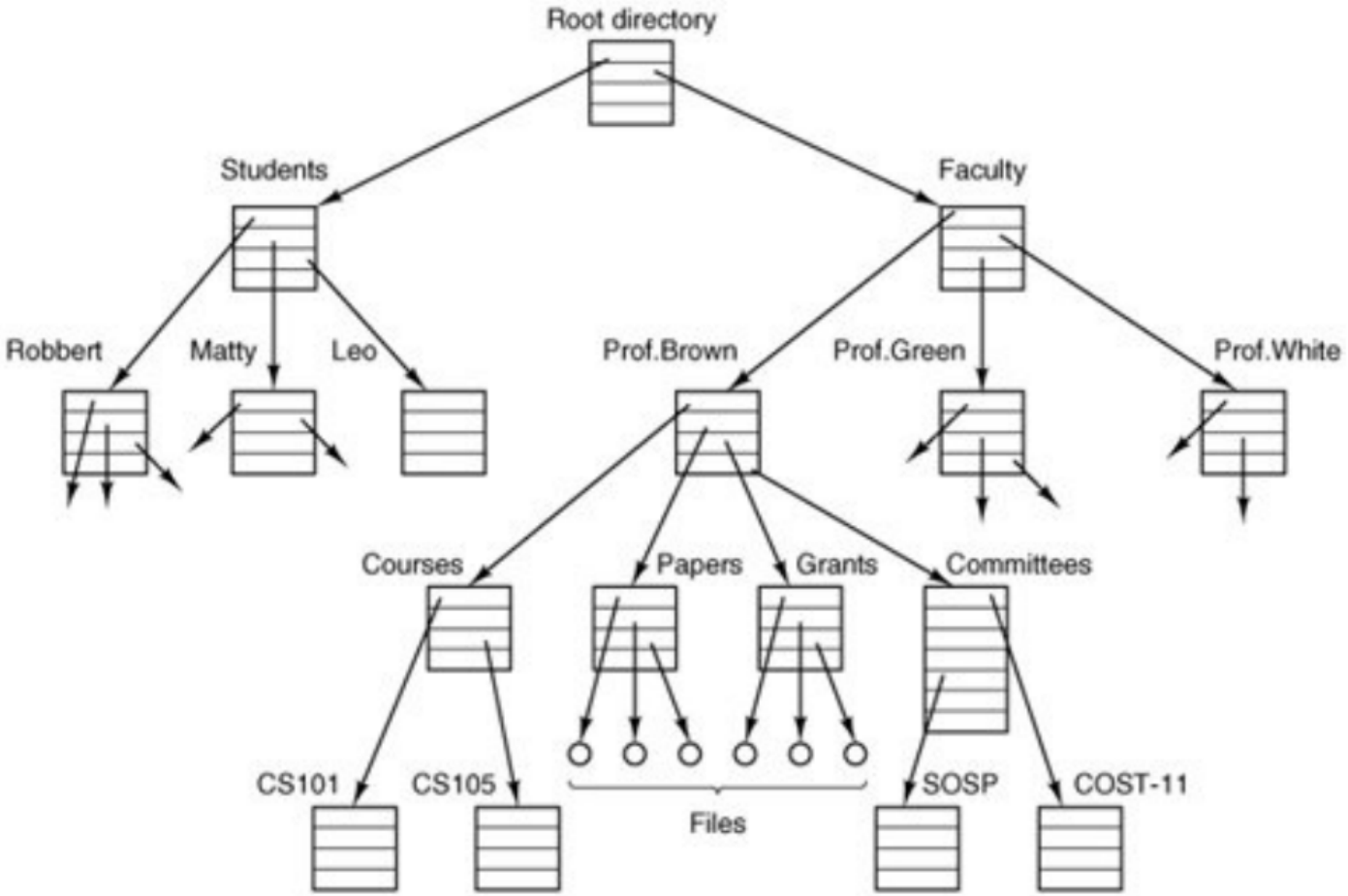


Devices & Drivers VI

Level	1	2	3	4	5
Name	registers	cache	main memory	solid-state disk	magnetic disk
Typical size	< 1 KB	< 16MB	< 64GB	< 1 TB	< 10 TB
Implementation technology	custom memory with multiple ports CMOS	on-chip or off-chip CMOS SRAM	CMOS SRAM	flash memory	magnetic disk
Access time (ns)	0.25-0.5	0.5-25	80-250	25,000-50,000	5,000,000
Bandwidth (MB/sec)	20,000-100,000	5,000-10,000	1,000-5,000	500	20-150
Managed by	compiler	hardware	operating system	operating system	operating system
Backed by	cache	main memory	disk	disk	disk or tape

Movement between levels of storage hierarchy can be explicit or implicit

File Systems I



File Systems II

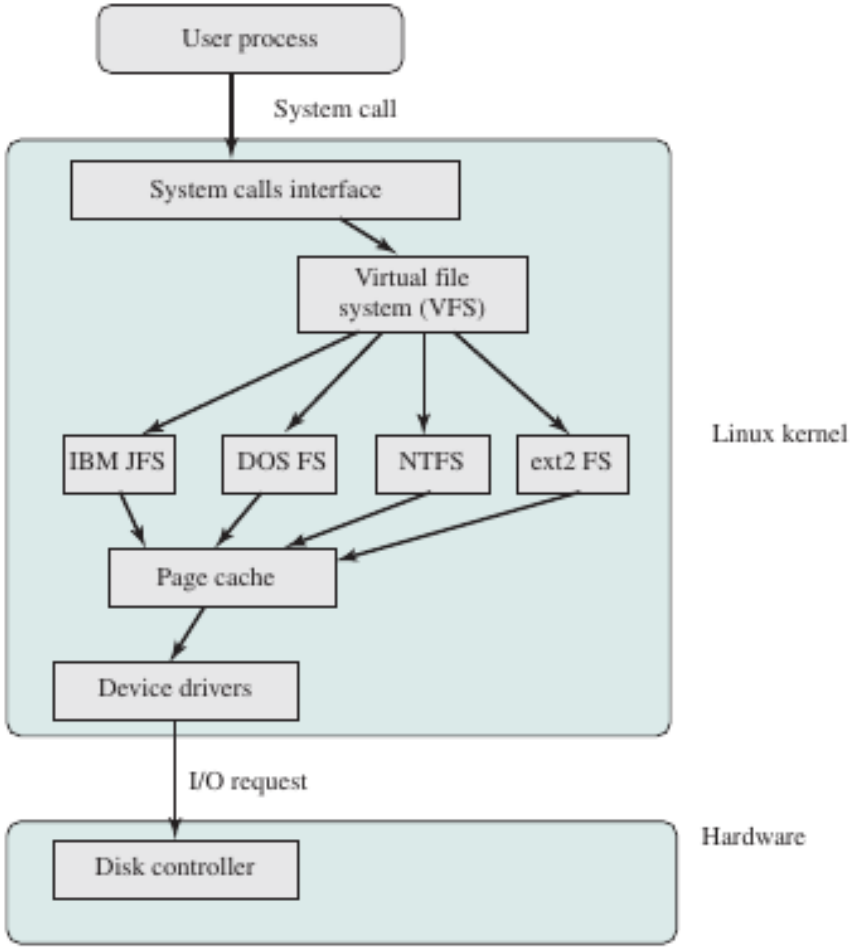
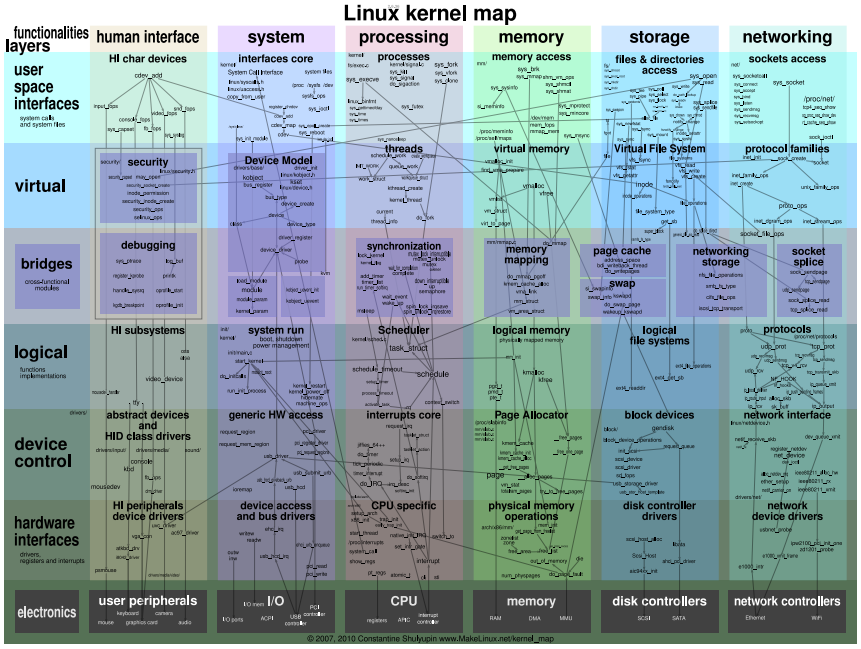


Figure 12.19 **Linux Virtual File System Context**

All in One Map

Take 5 min to explore the map!



Take another 5 min to explore the source tree visually on this site.

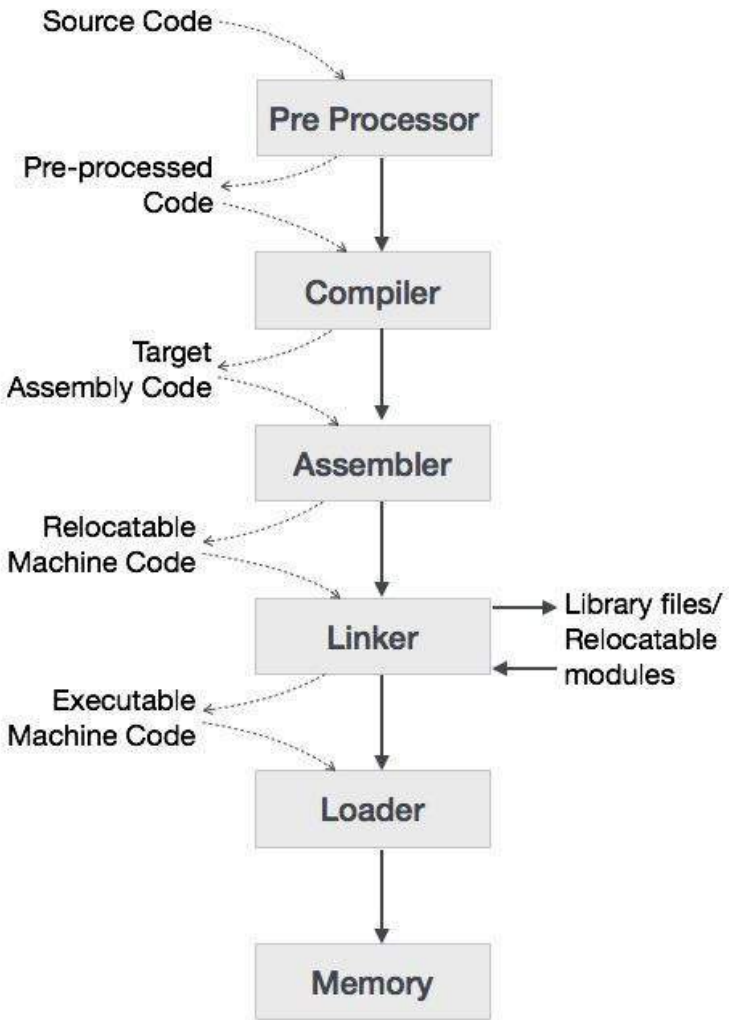
C vs. Java

```java
import java.util.Scanner;

public class Fakultaet
{
  public static void main (String[] args)
  {
    Scanner scan = new Scanner(System.in);

    int fakultaet = fak(scan.nextInt());
    System.out.println("Fakultaet: " + fakultaet);
  }
  private static int fak(int x) {
      return x <= 1 ? 1 : (x*fak(x-1));
  }
}
```

```c
#include <stdlib.h>
#include <stdio.h>

int fak(int);

int main(int argc, char *argv[]) {
  char* buf = malloc(100 * sizeof(char));
  fgets(buf, 100, stdin);

  int fakultaet = fak(atoi(buf));
  printf("Fakultaet: %d\n", fakultaet);
  free(buf);
}

int fak(int x) {
  return x <= 1 ? 1 : (x*fak(x-1));
}
```

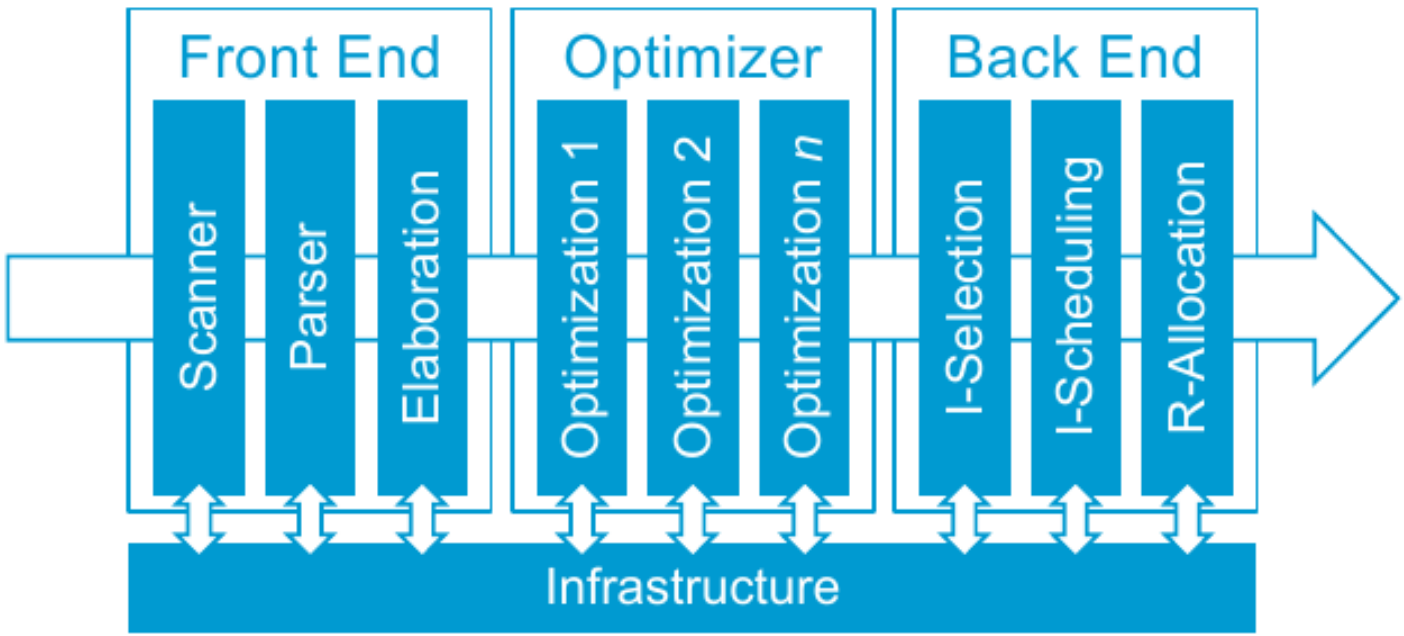
Contents of the Programming Part I

Compilers & Linkers I

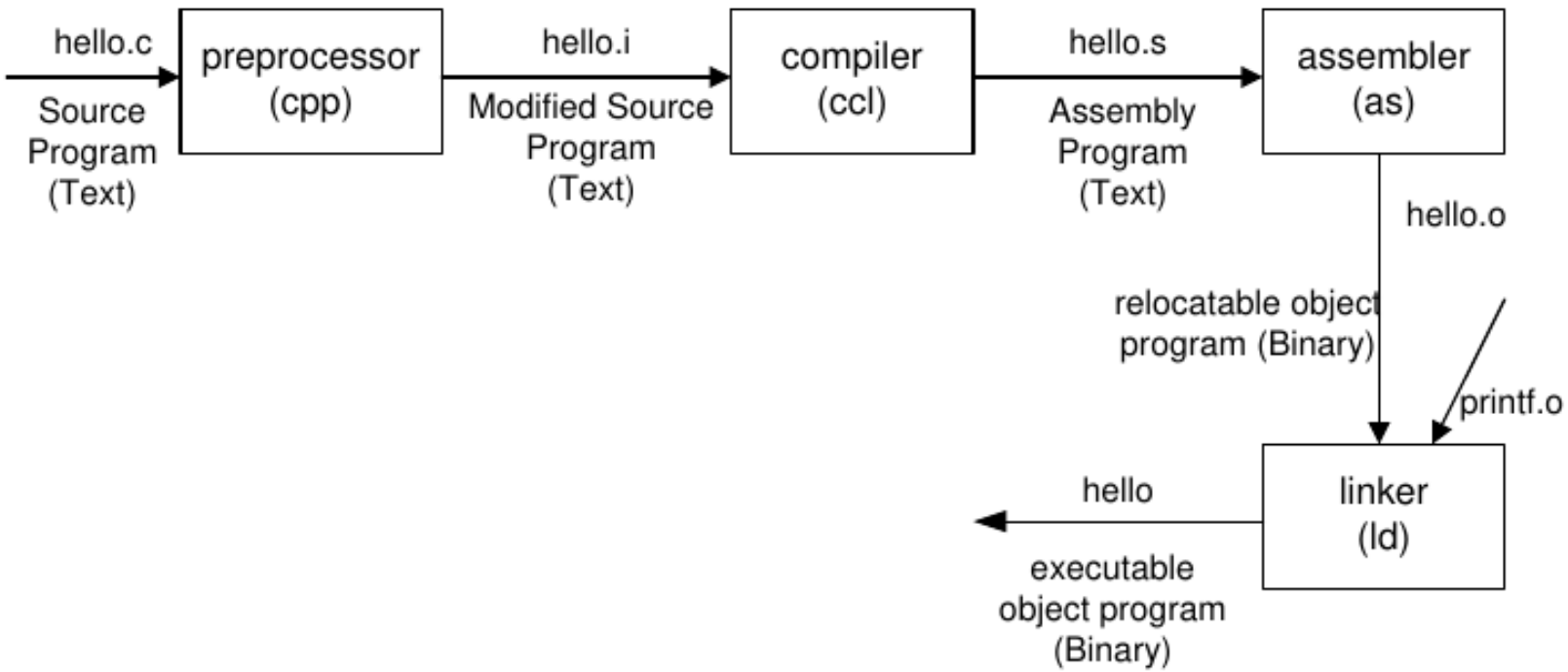


Compilers & Linkers II

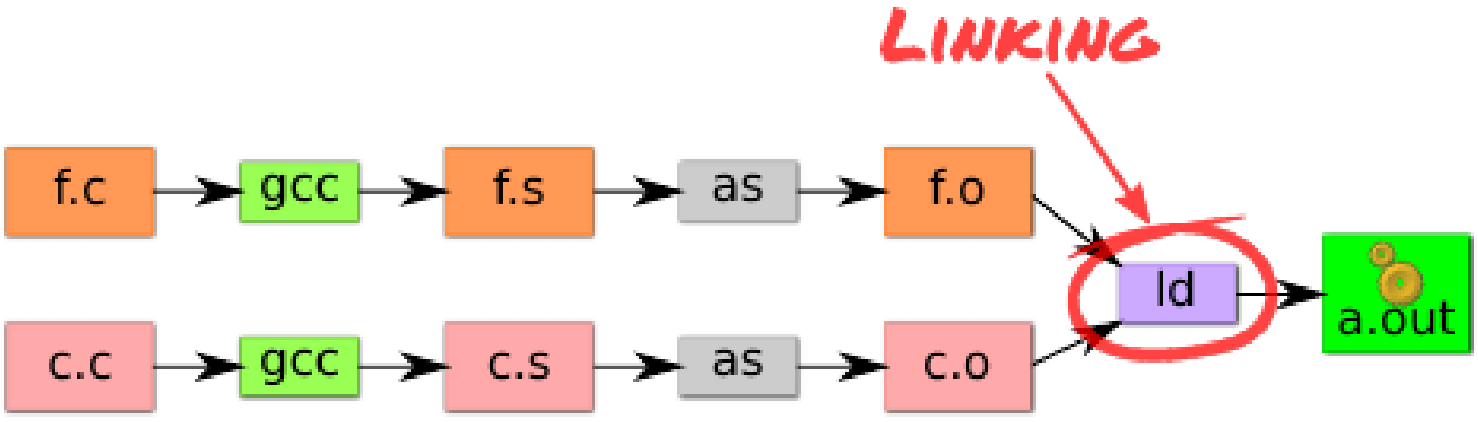
Typical Structure of a Compiler



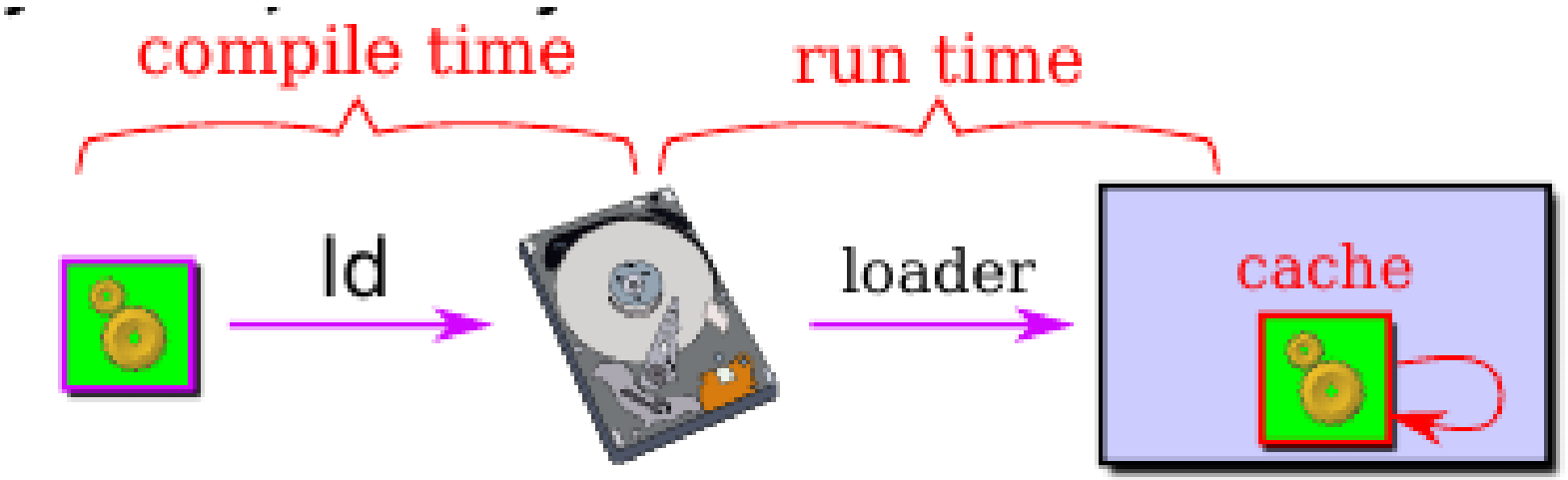
Compilers & Linkers III



Compilers & Linkers IV



Compilers & Linkers V



Compilers & Linkers VI

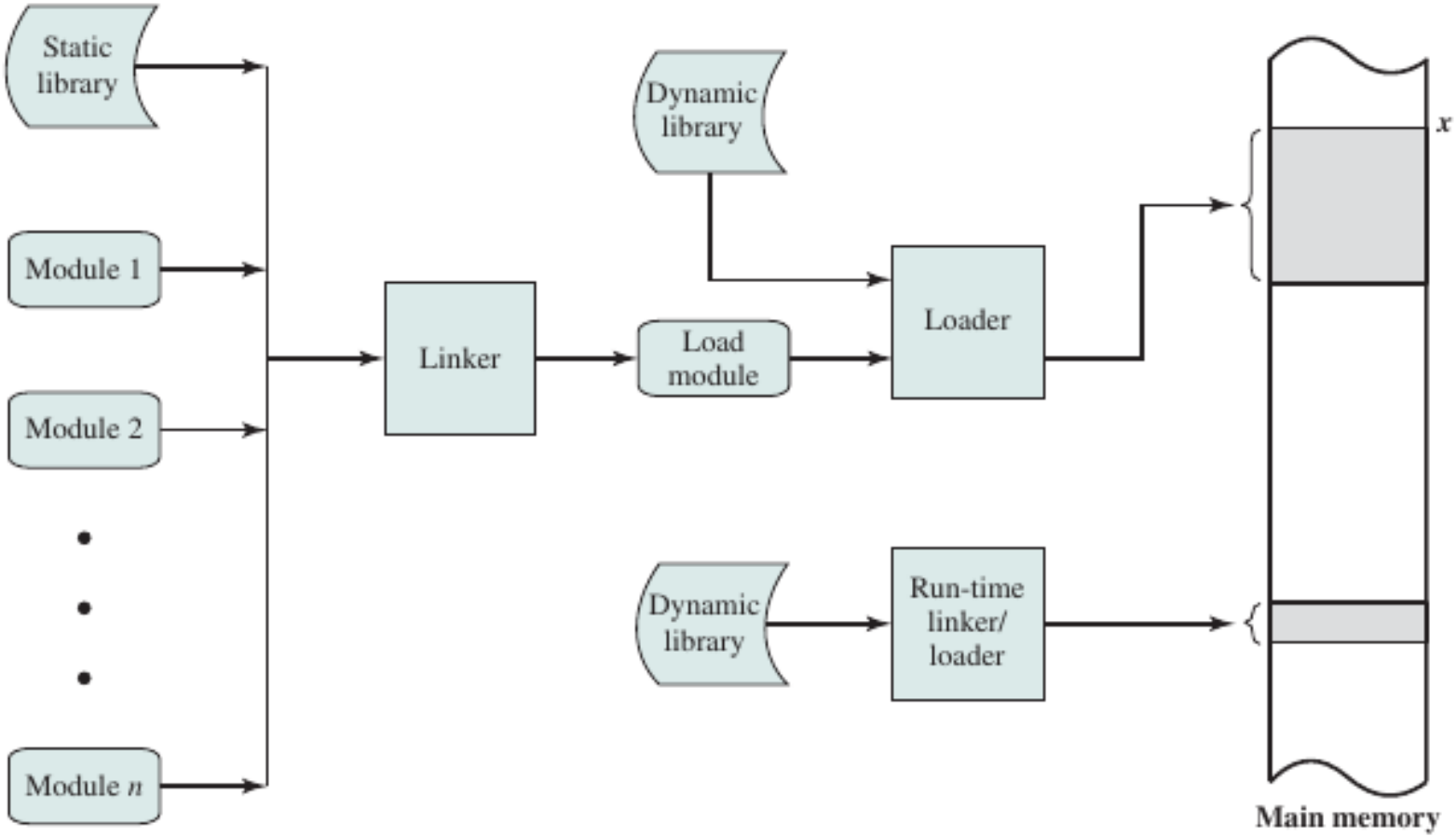


Figure 7.16 **A Linking and Loading Scenario**

References I


W. Stallings, *Operating systems: internals and design principles*. Upper Saddle River, NJ: Pearson/Prentice Hall, 2009.


A. Silberschatz, P. B. Galvin, and G. Gagne, *Operating system principles*. John Wiley & Sons, 2006.


A. Rubini and J. Corbet, *Linux device drivers*. " O'Reilly Media, Inc.", 2001.


D. Mazieres. (Mar. 16, 2010), "Uni stanford cs 140: Operating systems," [Online]. Available: https://www.scs.stanford.edu/10wi-cs140/ (visited on 11/02/2020).


A. S. Tanenbaum and H. Bos, *Modern operating systems*. Pearson, 2015.


Intel, "Intel 64 and ia-32 architectures software developers manual,", vol. 1,2,3,4, 2020.


L. Boulesteix. (Nov. 2, 2020), "What is the function of dma in a computer? - quora," [Online]. Available: https://www.quora.com/What-is-the-function-of-DMA-in-a-computer (visited on 11/02/2020).


C. Shulyupin. (Jul. 15, 2020), "Interactive map of linux kernel," [Online]. Available: https://makelinux.github.io/kernel/map/ (visited on 11/02/2020).