Chapter 2 – Hardware and Software Concepts

Outline

Chapter 2 – Hardware and Software Concepts

Outline (continued)

Objectives

- After reading this chapter, you should understand:
  - hardware components that must be managed by an operating system.
  - how hardware has evolved to support operating system functions.
  - how to optimize performance of various hardware devices.
  - the notion of an application programming interface (API).
  - the process of compilation, linking and loading.

2.1 Introduction

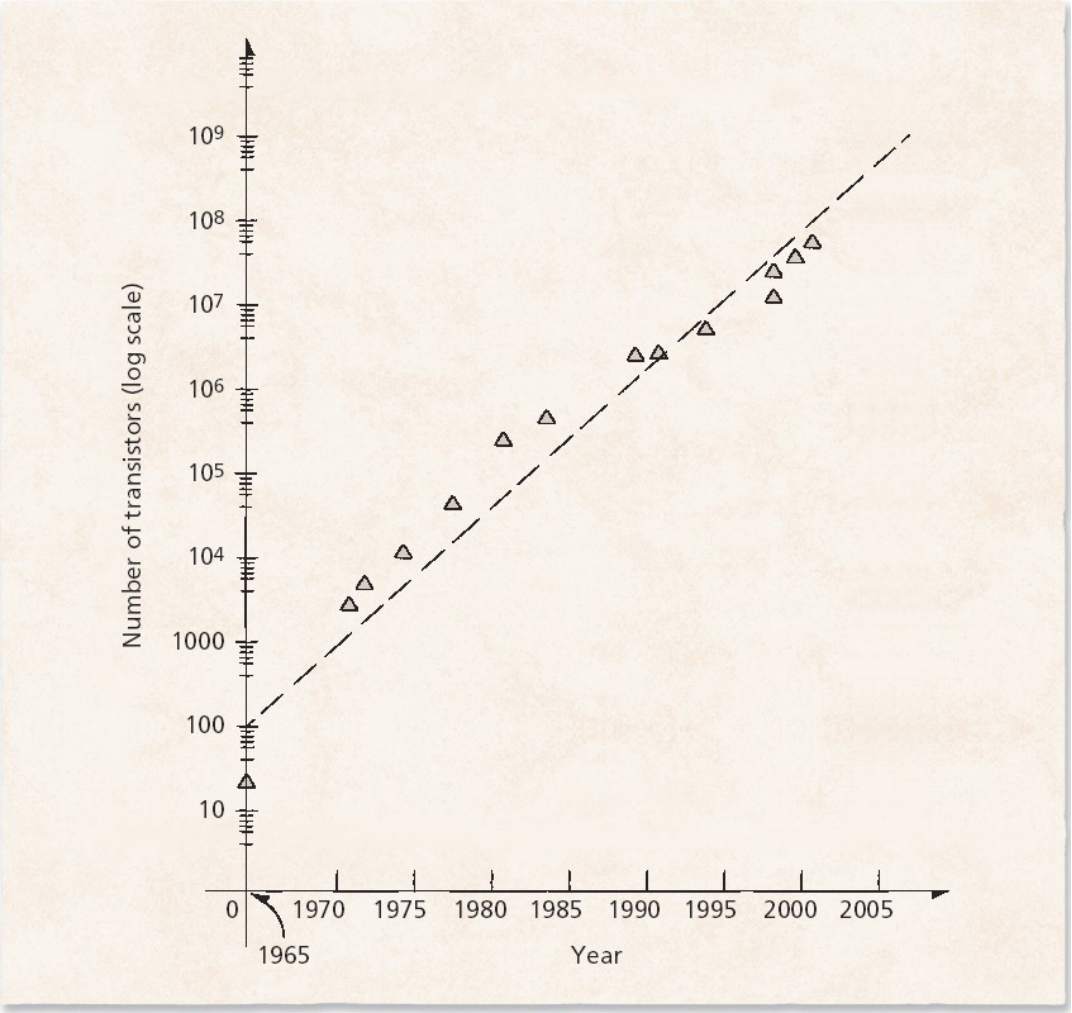
- An operating system is primarily a resource manager
  - Design is tied to the hardware and software resources the operating system must manage
    - processors
    - memory
    - secondary storage (such as hard disks)
    - other I/O devices
    - processes
    - threads
    - files
    - databases

2.2 Evolution of Hardware Devices

- Most operating systems are independent of hardware configurations
  - Operating systems use device drivers to perform device-specific I/O operations
    - For example, plug-and-play devices when connected instruct the operating system on which driver to use without user interaction

2.2 Evolution of Hardware Devices

Figure 2.1 Transistor count plotted against time for Intel processors.



2.3 Hardware Components

- A computer's hardware consists of:
  - processor(s)
  - main memory
  - input/output devices

2.3.1 Mainboards

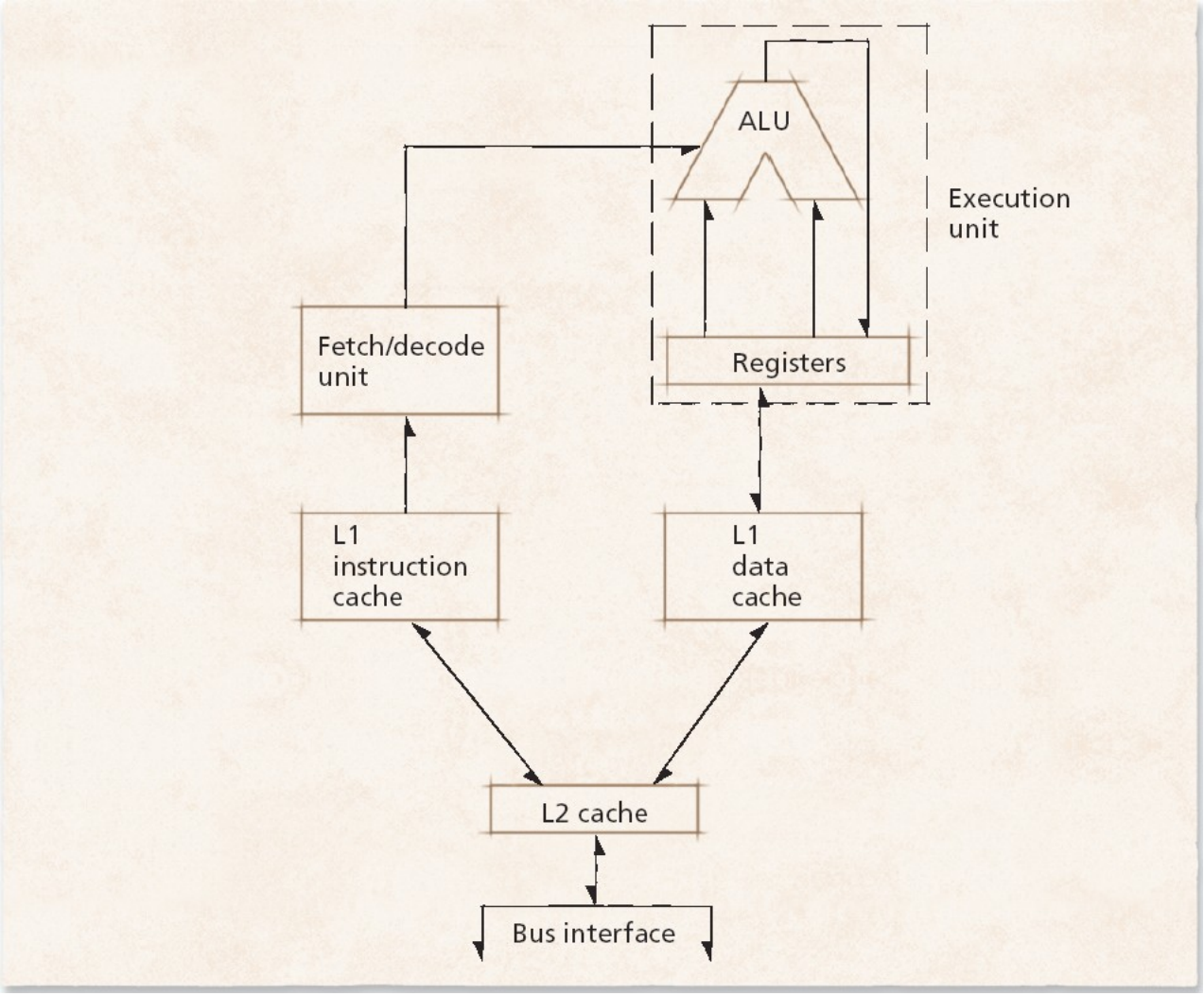
- Printed Circuit Board
  - Hardware component that provides electrical connections between devices
  - The mainboard is the central PCB in a system
    - Devices such as processors and main memory are attached
    - Include chips to perform low-level operations (e.g., BIOS)

2.3.2 Processors

- A processor is hardware that executes machine-language
  - CPU executes the instructions of a program
  - Coprocessor executes special-purpose instructions
    - Ex., graphics or audio coprocessors
  - Registers are high-speed memory located on processors
    - Data must be in registers before a processor can operate on it
  - Instruction length is the size of a machine-language instruction
    - Some processors support multiple instruction lengths

2.3.2 Processors

Figure 2.2 Processor components.



2.3.3 Clocks

- Computer time is measured in cycles
  - One complete oscillation of an electrical signal
  - Provided by system clock generator
  - Processor speeds are measured in GHz (billions of cycles per second)
    - Modern desktops execute at hundreds of megahertz or several GHz

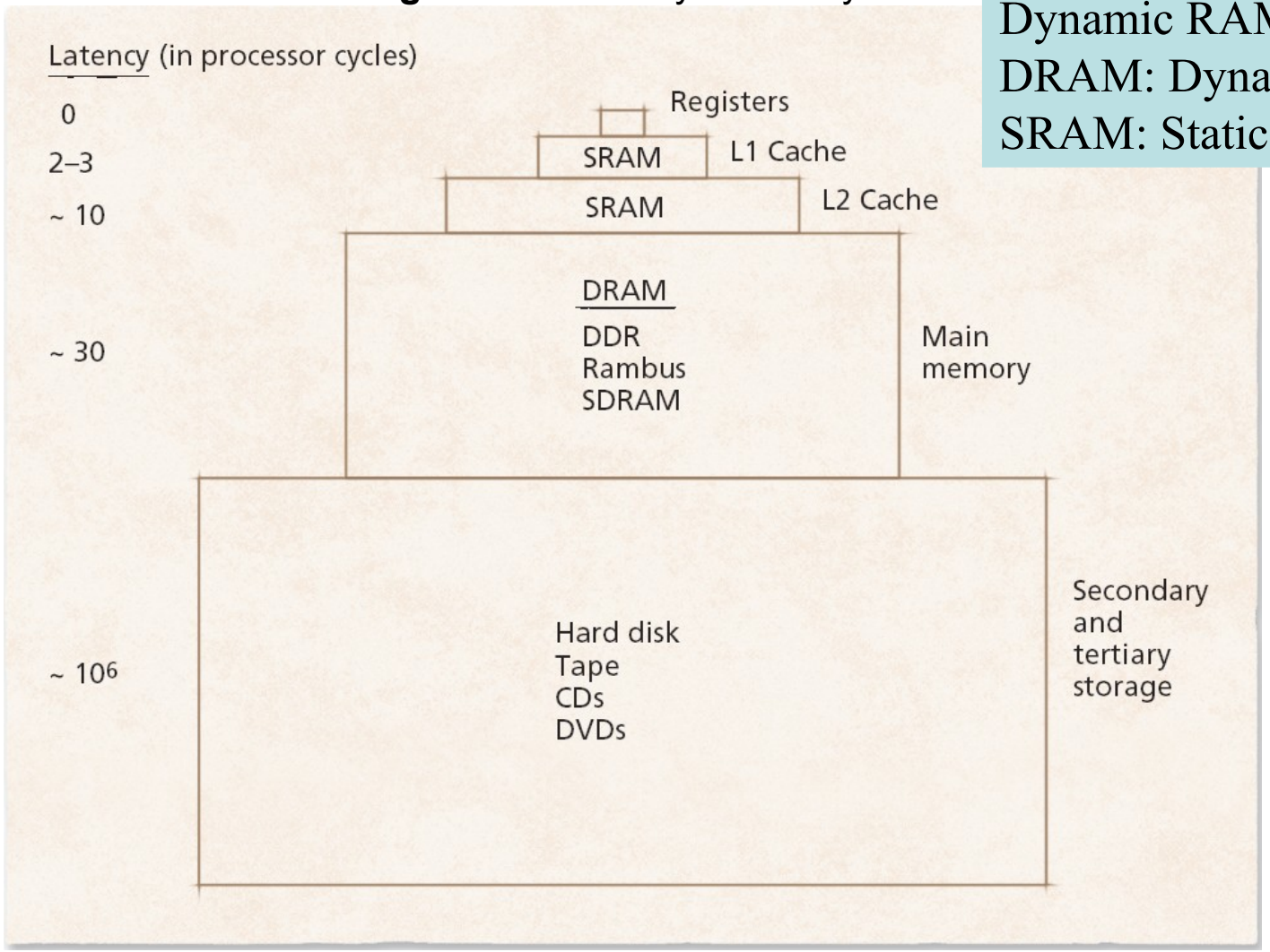
2.3.4 Memory Hierarchy

- The memory hierarchy is a scheme for categorizing memory
  - Fastest and most expensive at the top, slowest and least expensive at the bottom
    - Registers
    - L1 Cache
    - L2 Cache
    - Main Memory
    - Secondary and tertiary storage (CDs, DVDs and floppy disks)
  - Main memory is the lowest data referenced directly by processor
    - Volatile – loses its contents when the system loses power

2.3.4 Memory Hierarchy

SDRAM: Synchronous Dynamic RAM
DRAM: Dynamic RAM
SRAM: Static RAM

Figure 2.3 Memory hierarchy.



2.3.5 Main Memory

- Main memory consists of volatile random access memory (RAM)
  - Processes can access data locations in any order
  - Common forms of RAM include:
    - dynamic RAM (DRAM) – requires refresh circuit
    - static RAM (SRAM) – does not require refresh circuit
  - Bandwidth is the amount of data that can be transferred per unit of time

2.3.6 Secondary Storage

- Secondary storage stores large quantities of persistent data at low cost
  - Accessing data on a hard disk is slower than main memory
    - Mechanical movement of read/write head
    - Rotational latency
    - Transfer time
  - Removable secondary storage facilitates data backup and transfer
    - CDs (CD-R, CD-RW)
    - DVDs (DVD-R, DVD+R)
    - Zip disks
    - Floppy disks
    - Flash memory cards
    - Tapes

2.3.7 Buses

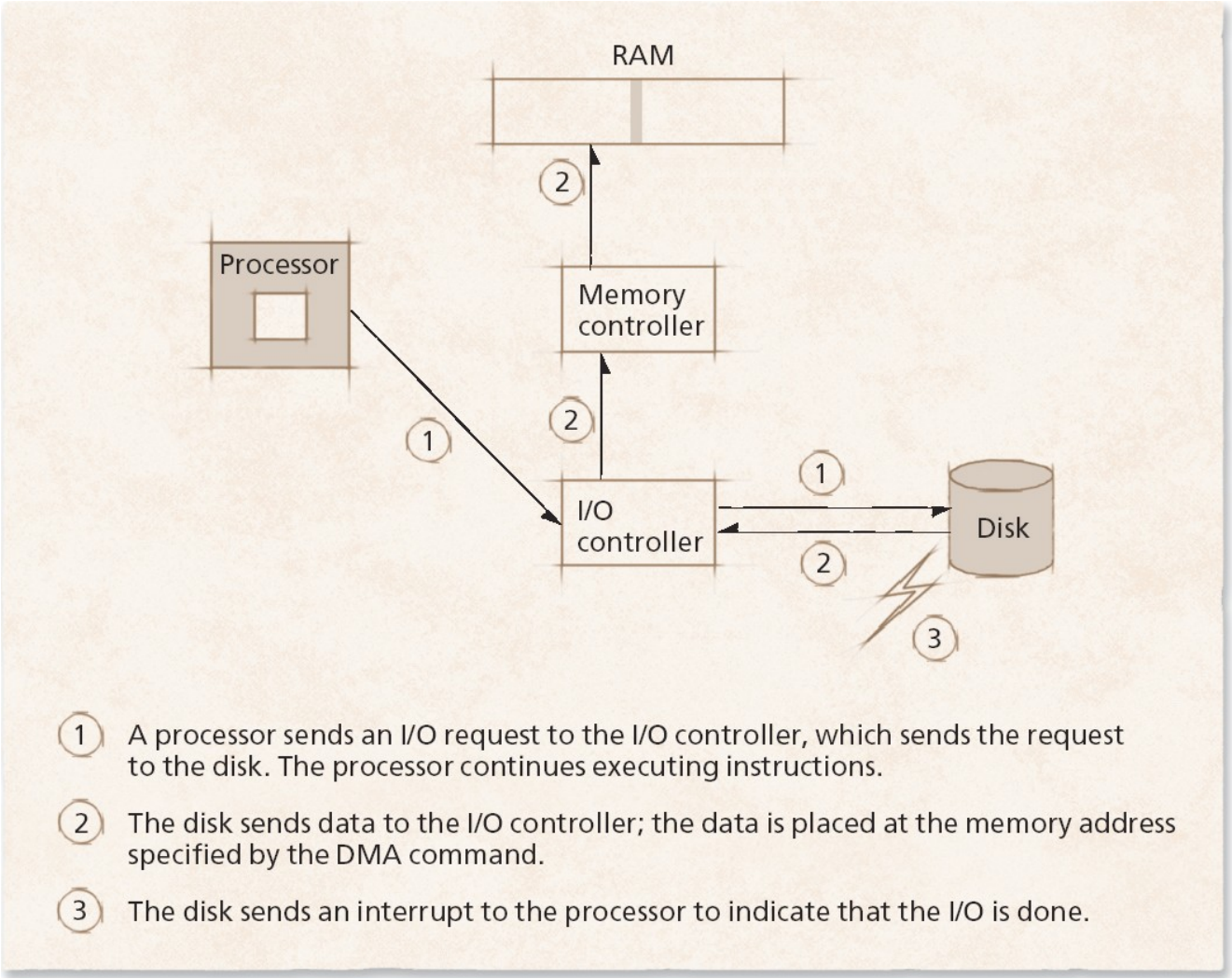
- A bus is a collection of traces
  - Traces are thin electrical connections that transport information between hardware devices
  - A port is a bus that connects exactly two devices
  - An I/O channel is a bus shared by several devices to perform I/O operations
    - Handle I/O independently of the system's main processors
  - Example, the frontside bus (FSB) connects a processor to main memory

2.3.8 Direct Memory Access (DMA)

- DMA improves data transfer between memory and I/O devices
  - Devices and controllers transfer data to and from main memory directly
  - Processor is free to execute software instructions
  - DMA channel uses an I/O controller to manage data transfer
    - Notifies processor when I/O operation is complete
  - Improves performance in systems that perform large numbers of I/O operations (e.g., mainframes and servers)

2.3.8 Direct Memory Access (DMA)

Figure 2.4 Direct memory access (DMA).



2.3.9 Peripheral Devices

- Peripheral devices
  - Any device not required for a computer to execute software instructions
  - Internal devices are referred to as integrated peripheral devices
    - Network interface cards, modems, sound cards
    - Hard disk, CD and DVD drives
  - Character devices transfer data one bit at a time
    - Keyboards and mice
  - Can be attached to a computer via ports and other buses
    - Serial ports, parallel ports, USB, IEEE 1394 ports and SCSI

2.3.9 Peripheral Devices

Figure 2.5 Peripheral devices (1 of 2).

Device	Description
CD-RW drive	Reads data from, and writes data to, optical disks.
Zip drive	Transfers data to and from a removable, durable magnetic disk.
Floppy drive	Reads data from, and writes data to, removable magnetic disks.
Mouse	Transmits the change in location of a pointer or cursor in a graphical user interface (GUI).
Keyboard	Transmits characters or commands that a user types.
Multifunction printer	Can print, copy, fax and scan documents.
Sound card	Converts digital signals to audio signals for speakers. Also can receive audio signals via a microphone and produce a digital signal.

2.3.9 Peripheral Devices

Figure 2.5 Peripheral devices (2 of 2).

Video accelerator	Displays graphics on the screen; accelerates two- and three-dimensional graphics.
Network card	Sends data to and receives data from other computers.
Digital camera	Records, and often displays, digital images.
Biometric device	Scans human characteristics, such as fingerprints and retinas, typically for identification and authentication purposes.
Infrared device	Communicates data between devices via a line-of-sight wireless connection.
Wireless device	Communicates data between devices via an omnidirectional wireless connection.

2.4 Hardware Support for Operating Systems

- Computer architectures contain:
  - Features that perform operating system functions quickly in hardware to improve performance
  - Features that enable the operating system to rigidly enforce protection

2.4.1 Processor

- A processor implements operating system protection mechanisms
  - Prevents processes from accessing privileged instructions or memory
  - Computer systems generally have several different execution modes:
    - User mode (user state or problem state)
      - User may execute only a subset of instructions
    - Kernel mode (supervisor state)
      - Processor may access privileged instructions and resources on behalf of processes

2.4.1 Processor

- Memory protection and management
  - Prevents processes from accessing memory that has not been assigned to them
  - Implemented using processor registers modified only by privileged instructions
- Interrupts and Exceptions
  - Most devices send a signal called an interrupt to the processor when an event occurs
  - Exceptions are interrupts generated in response to errors
  - The OS can respond to an interrupt by notifying processes that are waiting on such events

2.4.2 Timers and Clocks

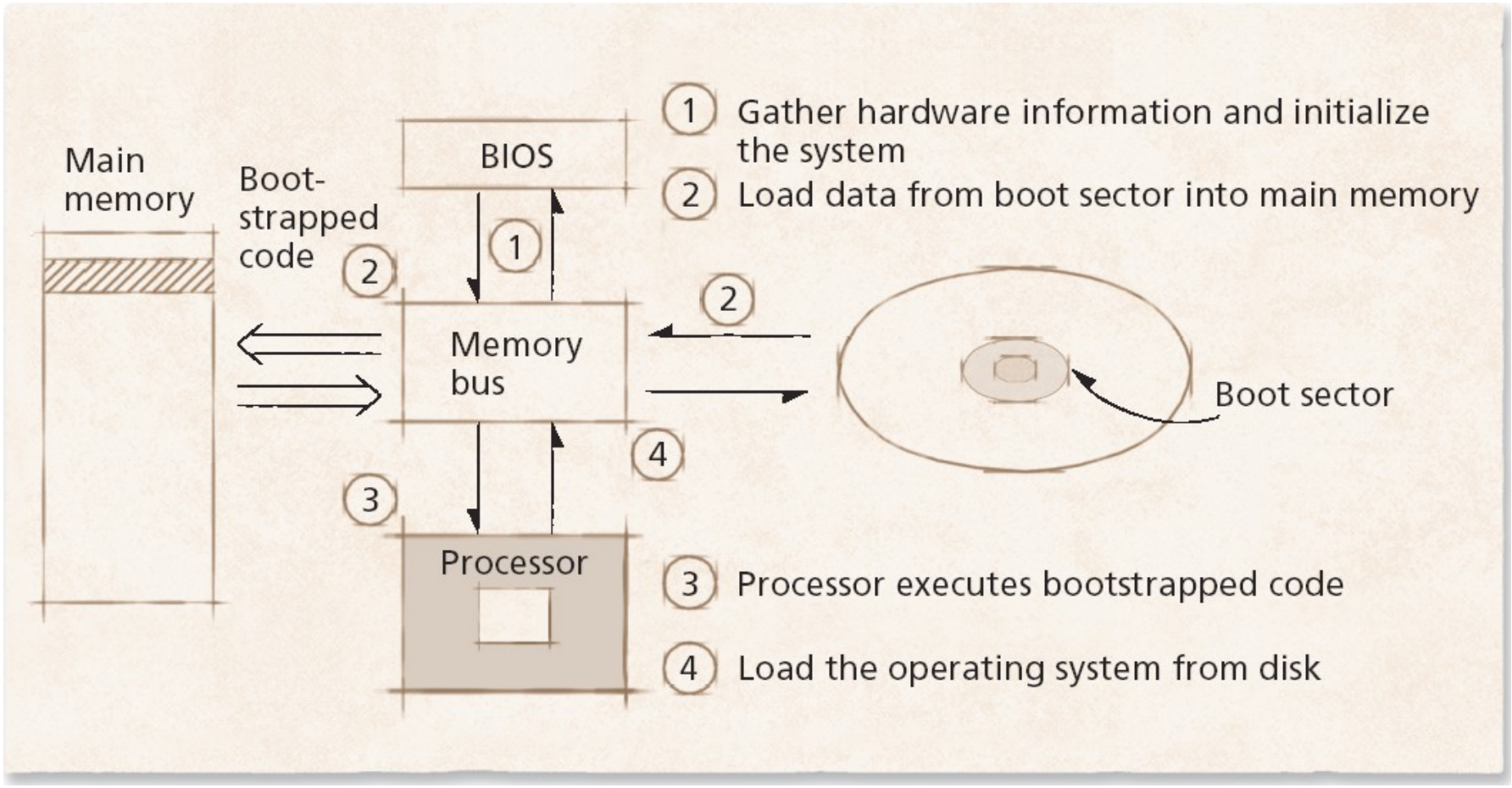
- Timers
  - An interval timer periodically generates an interrupt
  - Operating systems use interval timers to prevent processes from monopolizing the processor
- Clocks
  - Provide a measure of continuity
  - A time-of-day clock enables an OS to determine the current time and date

HERE 2.4.3 Bootstrapping

- Bootstrapping: loading initial OS components into memory
  - Performed by a computer's Basic Input/Output System (BIOS)
    - Initializes system hardware
    - Loads instructions into main memory from a region of secondary storage called the boot sector
  - If the system is not loaded, the user will be unable to access any of the computer's hardware

2.4.3 Bootstrapping

Figure 2.6 Bootstrapping.



2.4.4 Plug and Play

- Plug and Play technology
  - Allows operating systems to configure newly installed hardware without user interaction
  - To support plug and play, a hardware device must:
    - Uniquely identify itself to the operating system
    - Communicate with the OS to indicate the resources and services the device requires to function properly
    - Identify the driver that supports the device and allows software to configure the device (e.g., assign the device to a DMA channel)

2.5 Caching and Buffering

- Caches
  - Relatively fast memory
  - Maintain copies of data that will be accessed soon
  - Increase program execution speed
  - Examples include:
    - L1 and L2 processor caches
    - Main memory can be viewed as a cache for hard disks and other secondary storage devices

2.5 Caching and Buffering

- Buffers
  - Temporary storage area that holds data during I/O transfers
  - Primarily used to:
    - Coordinate communications between devices operating at different speeds
    - Store data for asynchronous processing
    - Allow signals to be delivered asynchronously
- Spooling
  - Buffering technique in which an intermediate device, such as a disk, is interposed between a process and a low-speed I/O device
  - Allows processes to request operations from a peripheral device without requiring that the device be ready to service the request

2.6 Software Overview

- Programming languages
  - Some are directly understandable by computers, others require translation
  - Classified generally as either:
    - Machine language
    - Assembly language
    - High-level language

2.6.1 Machine Language and Assembly Language

- Machine language
  - Defined by the computer's hardware design
  - Consists of streams of numbers (1s and 0s) that instruct computers how to perform elementary operations
  - A computer can understand only its own machine language
- Assembly language
  - Represents machine-language instructions using English-like abbreviations
  - Assemblers convert assembly language to machine language
  - Speeds programming, reduces potential for bugs

2.6.2 Interpreters and Compilers

- High-level languages
  - Instructions look similar to everyday English
  - Accomplish more substantial tasks with fewer statements
  - Require compilers and interpreters
- Compiler
  - Translator program that converts high-level language programs into machine language
- Interpreter
  - Program that directly executes source code or code that has been reduced to a low-level language that is not machine code

Example Assembly Code

Addr	Label	Instruction	Object code[6]
		.begin	
		.org 2048	
	a_start	.equ 3000	
2048		ld length,%	
2064		be done	00000010 10000000 00000000 00000110
2068		addcc %r1,-4,%r1	10000010 10000000 01111111 11111100
2072		addcc %r1,%r2,%r4	10001000 10000000 01000000 00000010

Assembly Code (cont)

2076		ld %r4,%r5	11001010 00000001 00000000 00000000
2080		ba loop	00010000 10111111 11111111 11111011
2084		addcc %r3,%r5,%r3	10000110 10000000 11000000 00000101
2088	done:	jmpl %r15+4,%r0	10000001 11000011 11100000 00000100
2092	length:	20	00000000 00000000 00000000 00010100
2096	address:	a_start	00000000 00000000 00001011 10111000
		.org a_start	
3000	a:		

2.6.3 High-level languages

- Popular high-level languages
  - Typically are procedural or object-oriented
  - Fortran
    - Used for scientific and engineering applications
  - COBOL
    - For business applications that manipulate large volumes of data
  - C
    - Development language of the UNIX OS
  - C++/Java
    - Popular object-oriented languages
  - C#
    - Object-oriented development language for the .NET platform

2.6.4 Structured programming

- Disciplined approach to creating programs
  - Programs are clear, provably correct and easy to modify
  - Structured programming languages include:
    - Pascal
      - Designed for teaching structured programming
    - Ada
      - Developed by the US Department of Defense
    - Fortran

2.6.5 Object-Oriented Programming

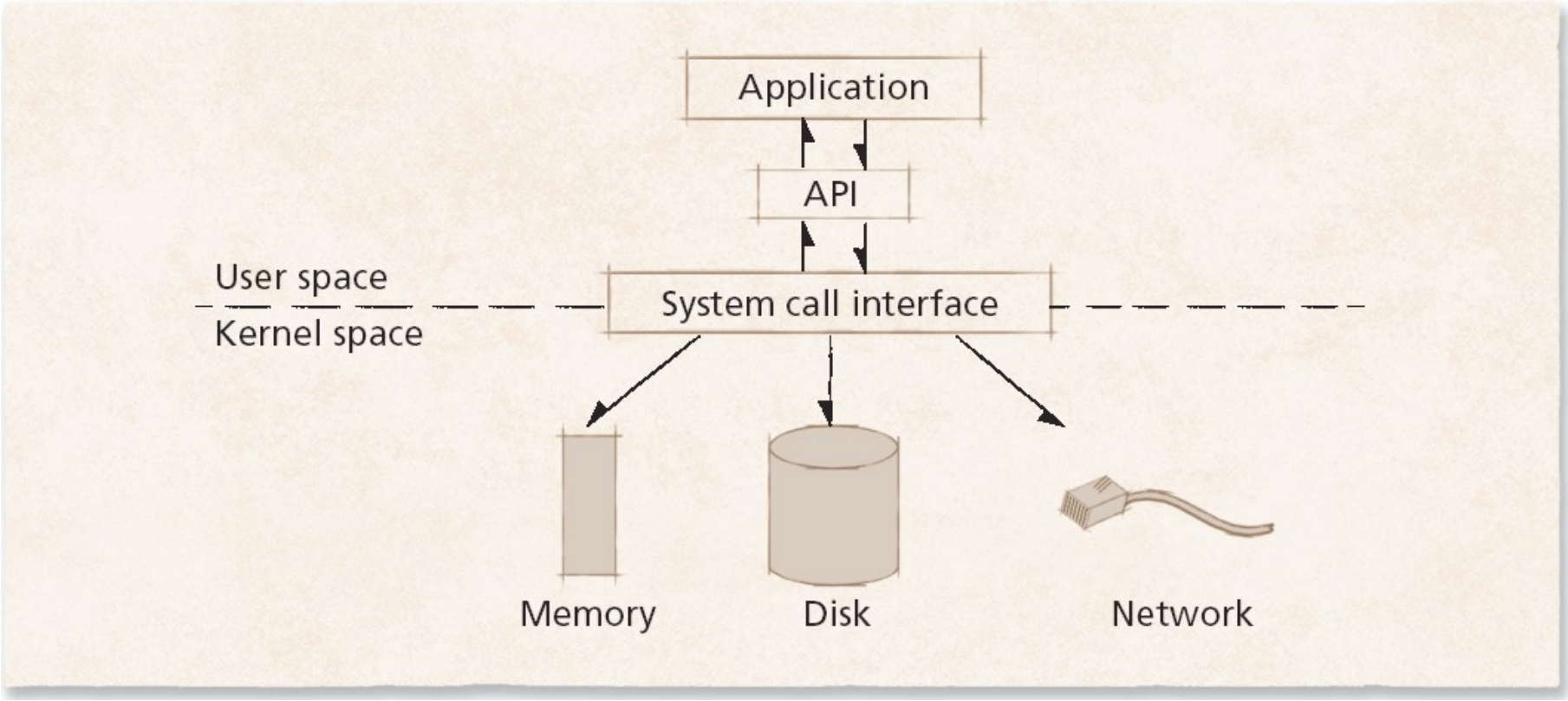
- Objects
  - Reusable software unit (any noun can be represented)
  - Easy to modify and understand
  - Have properties (e.g., color) and perform actions (e.g., moving)
- Classes
  - Types of related objects
  - Specify the general format of an object and the properties and actions available to it
- Object-oriented programming
  - Focuses on behaviors and interactions, not implementation
  - C++, Java and C# are popular object-oriented languages

2.7 Application Programming Interfaces (APIs)

- A set of routines
  - Programmers use routines to request services from the operating system
  - Programs call API functions, which may access the OS by making system calls
  - Examples of APIs include:
    - Portable Operating System Interface (POSIX) standard
    - Windows API

2.7 Application Programming Interfaces (APIs)

Figure 2.7 Application programming interface (API).



2.8 Compiling, Linking and Loading

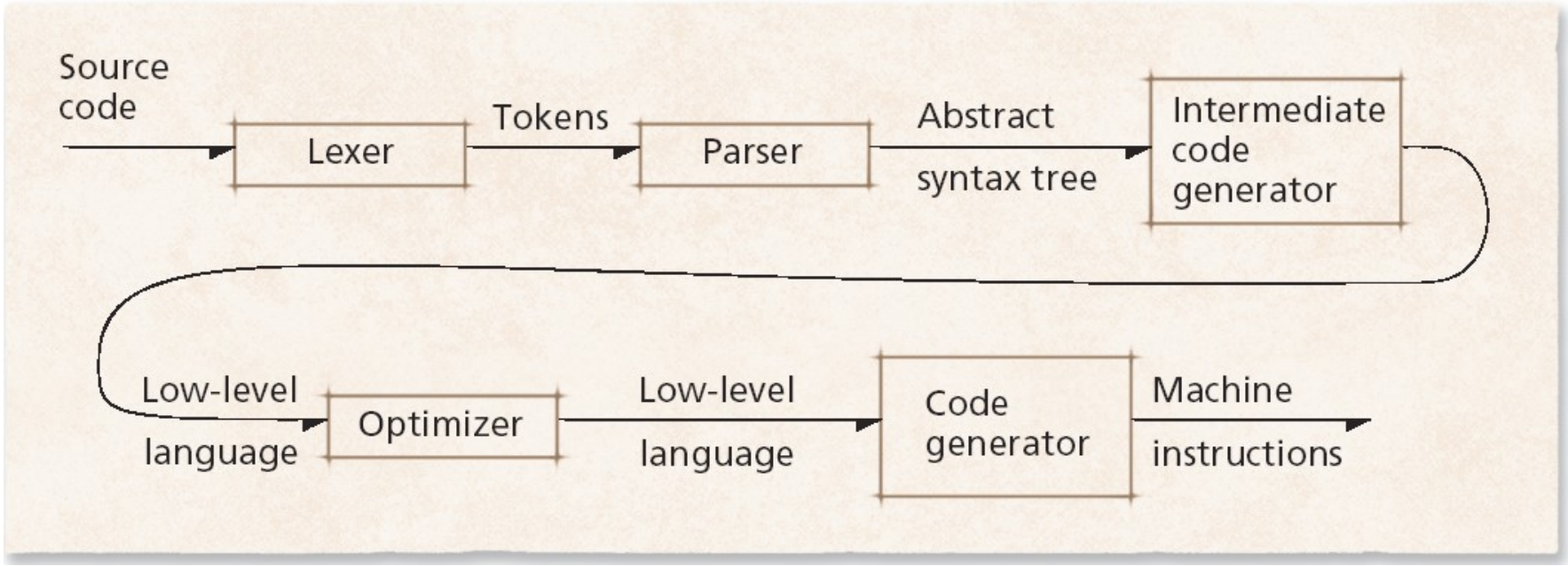
- Before a high-level-language program can execute, it must be:
  - Translated into machine language
  - Linked with various other machine-language programs on which it depends
  - Loaded into memory

2.8.1 Compiling

- Translating high-level code to machine code
  - Accepts source code as input and returns object code
  - Compilation phases include:
    - Lexer
      - Separates the characters of a program's source into tokens
    - Parser
      - Groups tokens into syntactically correct statements
    - Intermediate code generator
      - Converts statements into a stream of simple instructions
    - Optimizer
      - Improves code execution efficiency and memory requirements
    - Code generator
      - Produces the object file containing the machine-language

2.8.1 Compiling

Figure 2.8 Compiler phases.

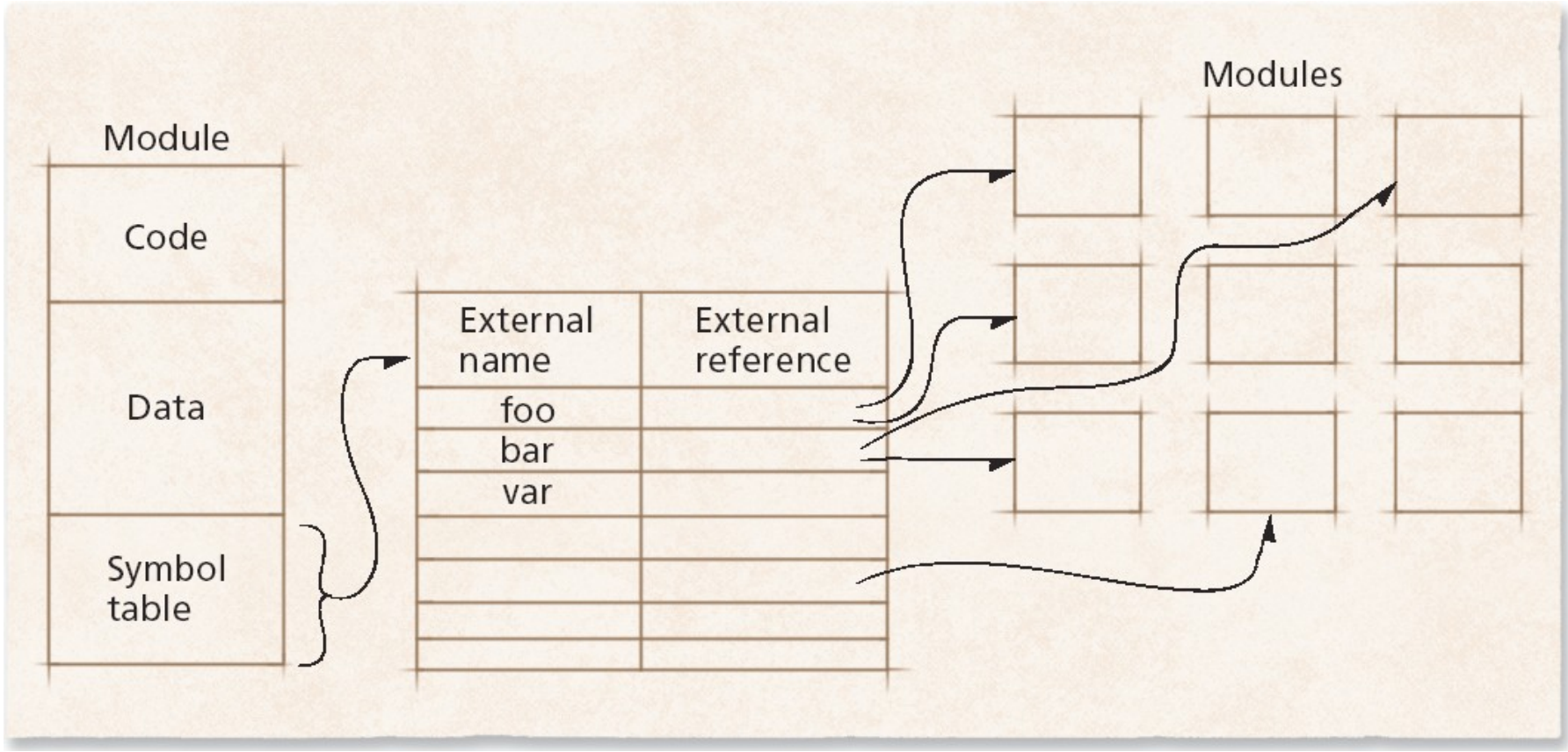


2.8.2 Linking

- Linkers
  - Create a single executable unit
  - Integrate precompiled modules called libraries referenced by a program
  - Assign relative addresses to different program or data units
  - Resolve all external references between subprograms
  - Produce an integrated module called a load module
  - Linking can be performed at compile time, before loading, at load time or at runtime

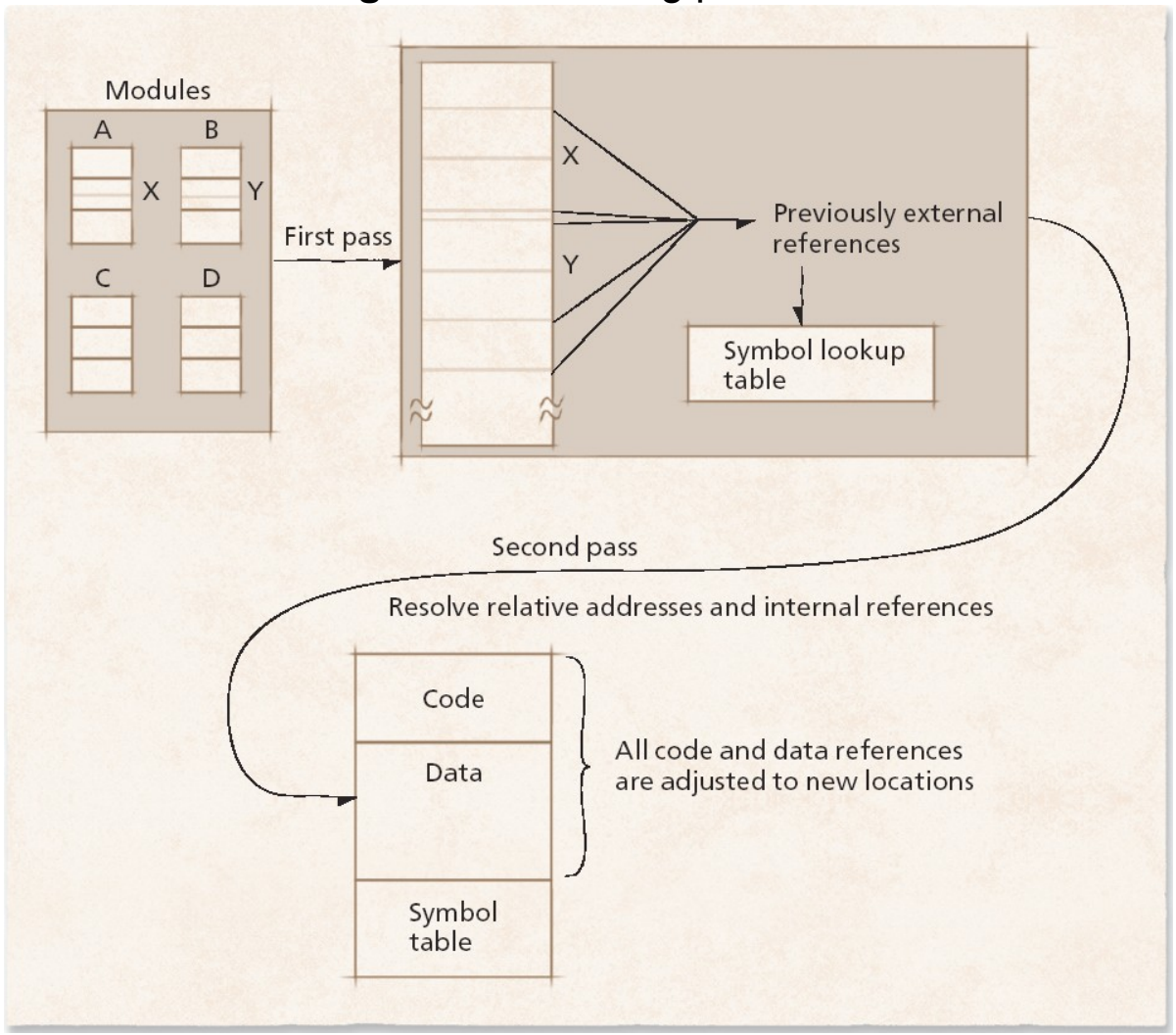
2.8.2 Linking

Figure 2.9 Object module.



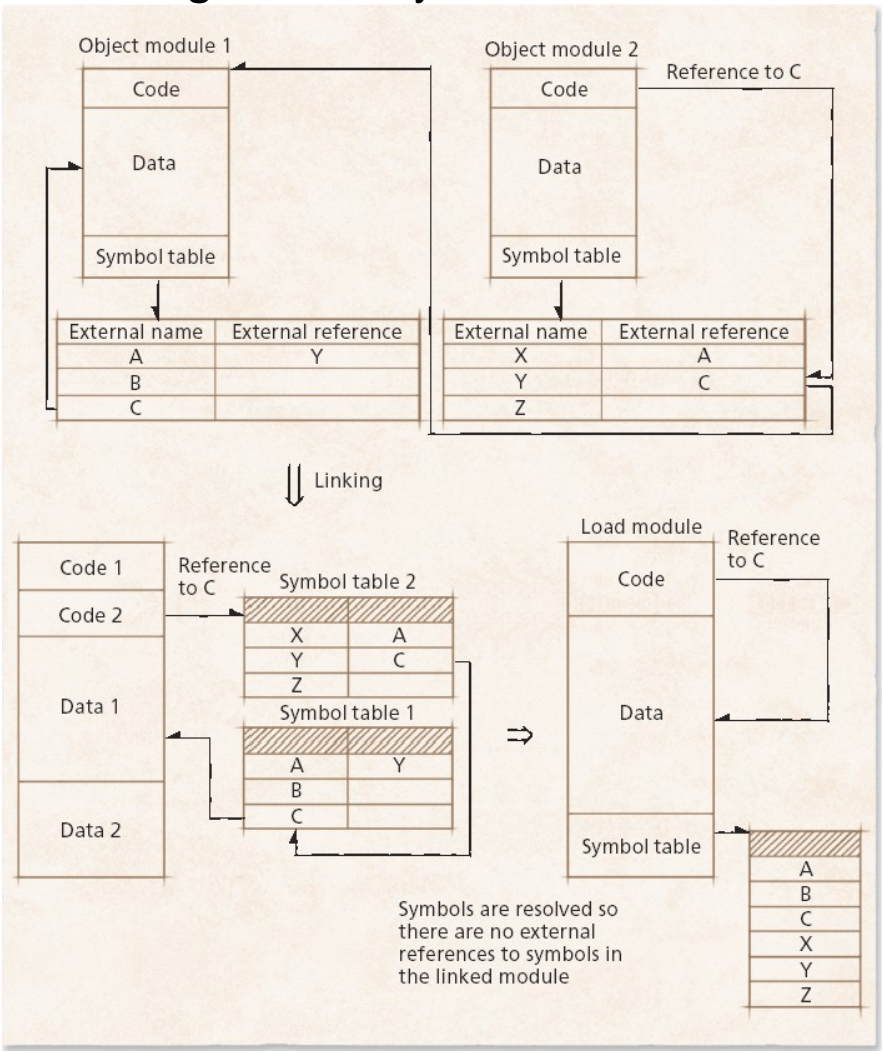
2.8.2 Linking

Figure 2.10 Linking process.



2.8.2 Linking

Figure 2.11 Symbol resolution.

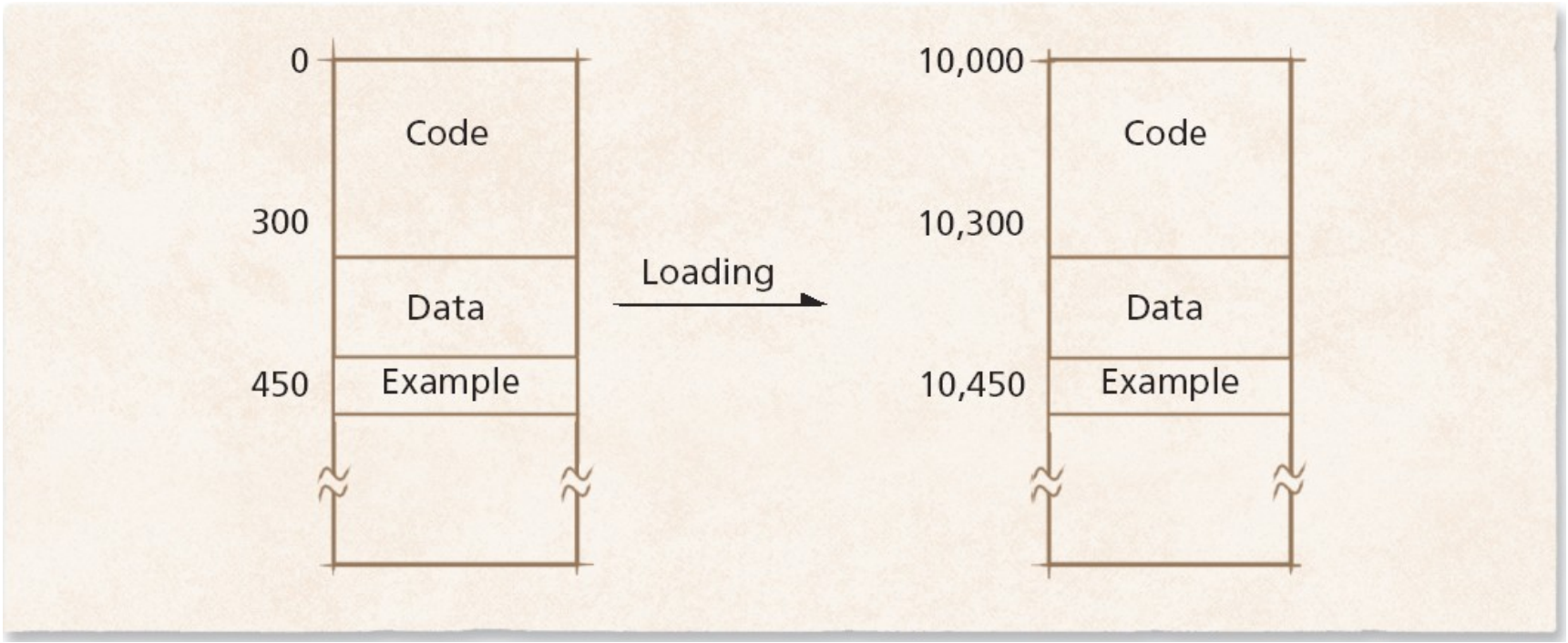


2.8.3 Loading

- Loaders
  - Convert relative addresses to physical addresses
  - Place each instruction and data unit in main memory
- Techniques for loading a program into memory
  - Absolute loading
    - Place program at the addresses specified by programmer or compiler (assuming addresses are available)
  - Relocatable loading
    - Relocate the program's addresses to correspond to its actual location in memory
  - Dynamic loading
    - Load program modules upon first use

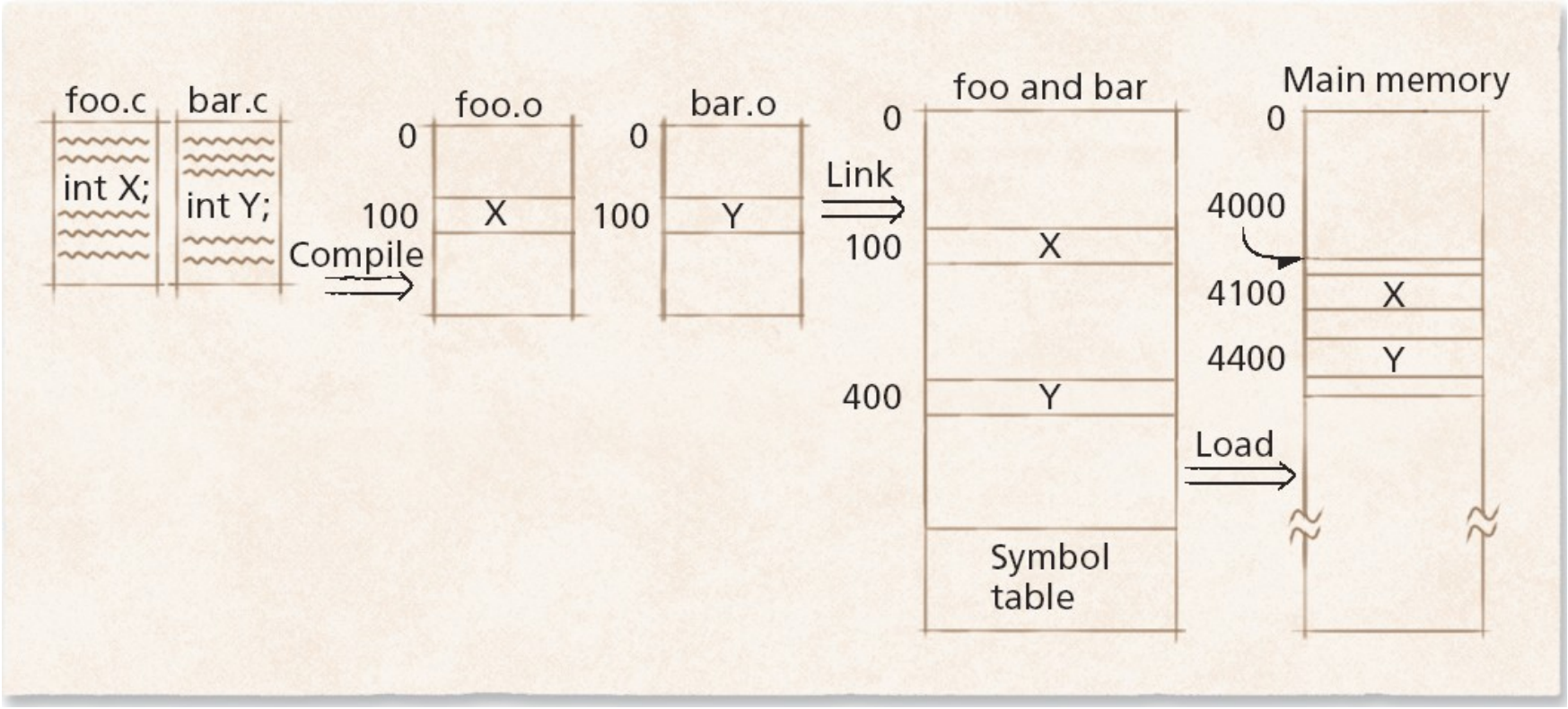
2.8.3 Loading

Figure 2.12 Loading.



2.8.3 Loading

Figure 2.13 Compiling, linking and loading.



2.9 Firmware

- Firmware contains executable instructions stored in persistent memory attached to a device
  - Programmed with microprogramming
    - Layer of programming below a computer's machine-language
    - Microcode
      - Simple, fundamental instruction necessary to implement all machine-language operations

2.10 Middleware

- Middleware is software for distributed systems
  - Enables interactions among multiple processes running on one or more computers across a network
  - Facilitates heterogeneous distributed systems
  - Simplifies application programming
  - Example, Open DataBase Connectivity (ODBC)
    - Permits applications to access databases through middleware called an ODBC driver