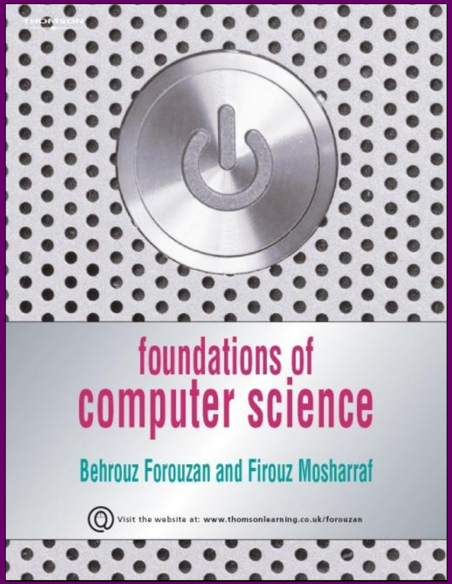
7
Operating Systems


foundations of
computer science
Behrouz Forouzan and Firouz Mosharraf
Visit the website at: www.thomsonlearning.co.uk/forouzan

Objectives

After studying this chapter, the student should be able to:

- ❑ Understand the role of the operating system.
- ❑ Understand the process of bootstrapping to load the operating system into memory.
- ❑ List the components of an operating system.
- ❑ Discuss the role of the memory manager.
- ❑ Discuss the role of the process manager.
- ❑ Discuss the role of the device manager.
- ❑ Discuss the role of the file manager in an operating system.
- ❑ Understand the main features of three common operating systems: UNIX, Linux and Windows NT.

A computer is a system composed of two major components: hardware and software. Computer hardware is the physical equipment. Software is the collection of programs that allows the hardware to do its job. Computer software is divided into two broad categories: the **operating system** and **application programs** (Figure 7.1). Application programs use the computer hardware to solve users' problems. The operating system, on the other hand, controls the access to hardware by users.

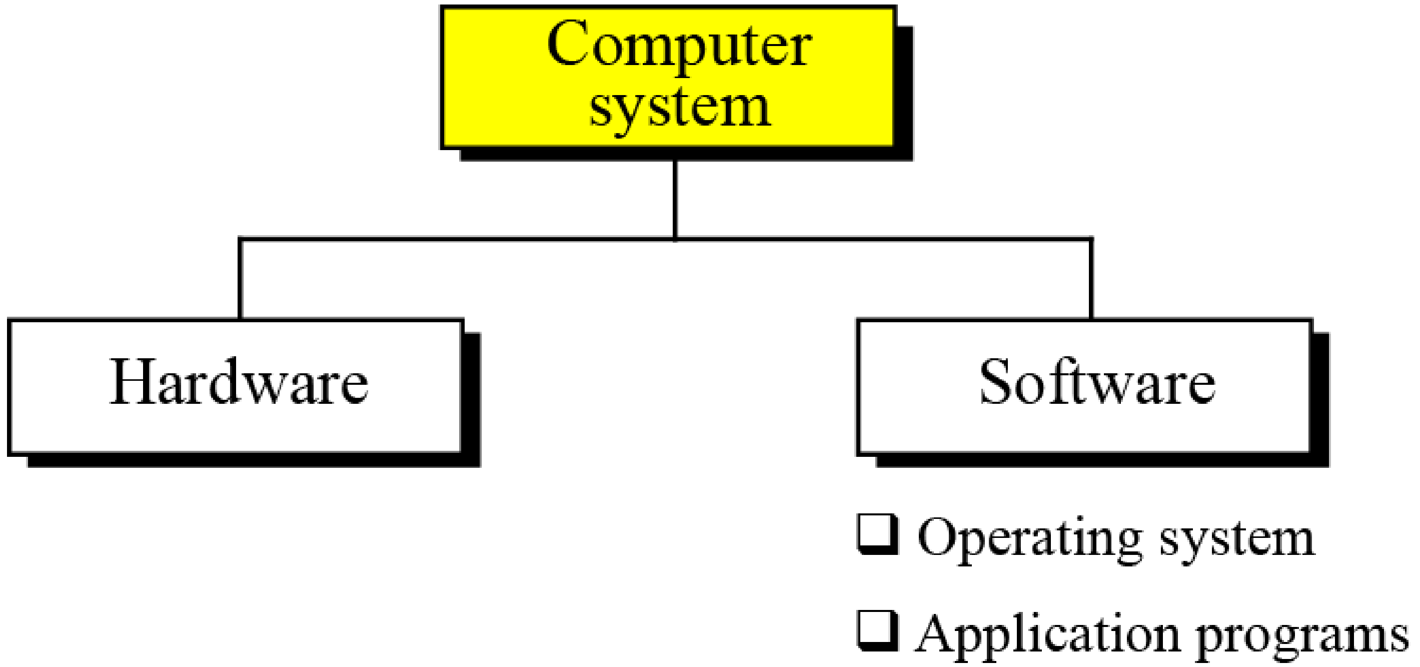


Figure 7.1 **A computer system**

7-1 INTRODUCTION

An **operating system** is complex, so it is difficult to give a simple universal definition. Instead, here are some common definitions:

- ❑ **An operating system is an interface between the hardware of a computer and the user (programs or humans).**
- ❑ **An operating system is a program (or a set of programs) that facilitates the execution of other programs.**
- ❑ **An operating system acts as a general manager supervising the activity of each component in the computer system.**

An operating system is an interface between the hardware of a computer and the user (programs or humans) that facilitates the execution of other programs and the access to hardware and software resources.

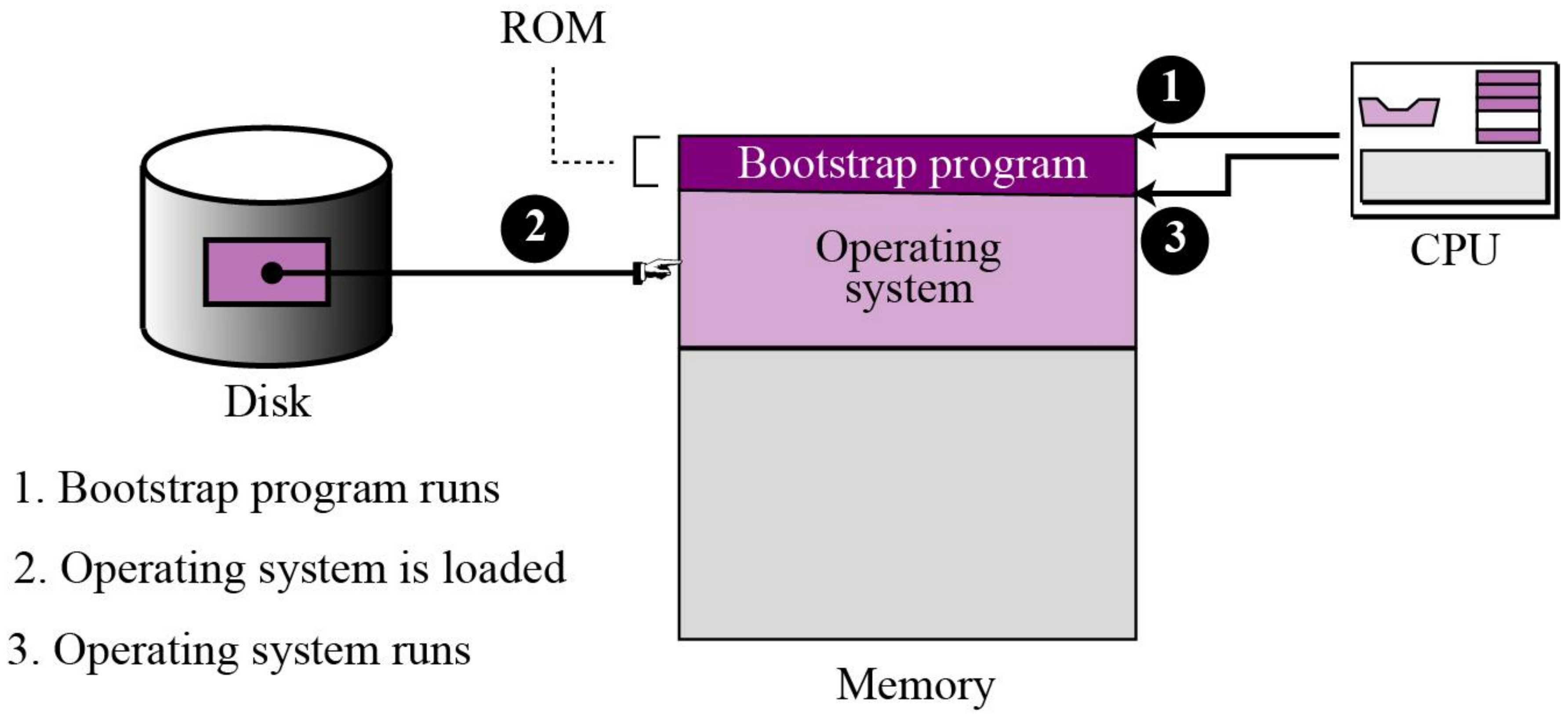
Two major design goals of an operating system are:

- ❑ **Efficient use of hardware.**
- ❑ **Ease of use of resources.**

Bootstrap process

The operating system, based on the above definitions, provides supports for other programs. For example, it is responsible for loading other programs into memory for execution. However, the operating system itself is a program that needs to be loaded into the memory and be run. How is this dilemma solved?

The solution is a two-stage process. A very small section of memory is made of ROM and holds a small program called the **bootstrap program**. When the computer is turned on, the CPU counter is set to the first instruction of this bootstrap program and executes the instructions in this program. When loading is done, the program counter is set to the first instruction of the operating system in RAM.



1. Bootstrap program runs
2. Operating system is loaded
3. Operating system runs

Figure 7.2 **The bootstrap process**

7-2 EVOLUTION

Operating systems have gone through a long history of evolution, which we summarize here.

Batch systems

Batch operating systems were designed in the 1950s to control mainframe computers. At that time, computers were large machines that used punched cards for input, line printers for output and tape drives for secondary storage media. Each program to be executed was called a job. A programmer who wished to execute a job sends a request to the operating system.

Time-sharing systems

To use computer system resources efficiently, *multiprogramming* was introduced. The idea is to hold several jobs in memory at a time, and only assign a resource to a job that needs it on the condition that the resource is available.

Multiprogramming brought the idea of **time sharing**: resources could be shared between different jobs, with each job being allocated a portion of time to use a resource. Because a computer is much faster than a human, time sharing is hidden from the user—each user has the impression that the whole system is serving them exclusively.

Personal systems

When personal computers were introduced, there was a need for an operating system for this new type of computer. During this era, single-user operating systems such as **DOS (Disk Operating System)** were introduced.

Parallel systems

The need for more speed and efficiency led to the design of **parallel systems**: multiple CPUs on the same machine. Each CPU can be used to serve one program or a part of a program, which means that many tasks can be accomplished in parallel instead of serially. The operating systems required for this are more complex than those that support single CPUs.

Distributed systems

Networking and internetworking, as we saw in Chapter 6, have created a new dimension in operating systems. A job that was previously done on one computer can now be shared between computers that may be thousands of miles apart. Distributed systems combine features of the previous generation with new duties such as controlling security.

Real-time systems

A real-time system is expected to do a task within a specific time constraint. They are used with real-time applications, which monitor, respond to or control external processes or environments.

7-3 COMPONENTS

Today's operating systems are very complex. An operating system needs to manage different resources in a computer system. It resembles an organization with several managers at the top level. Each manager is responsible for managing their department, but also needs to cooperate with others and coordinate activities. A modern operating system has at least four duties: **memory manager**, **process manager**, **device manager** and **file manager**.

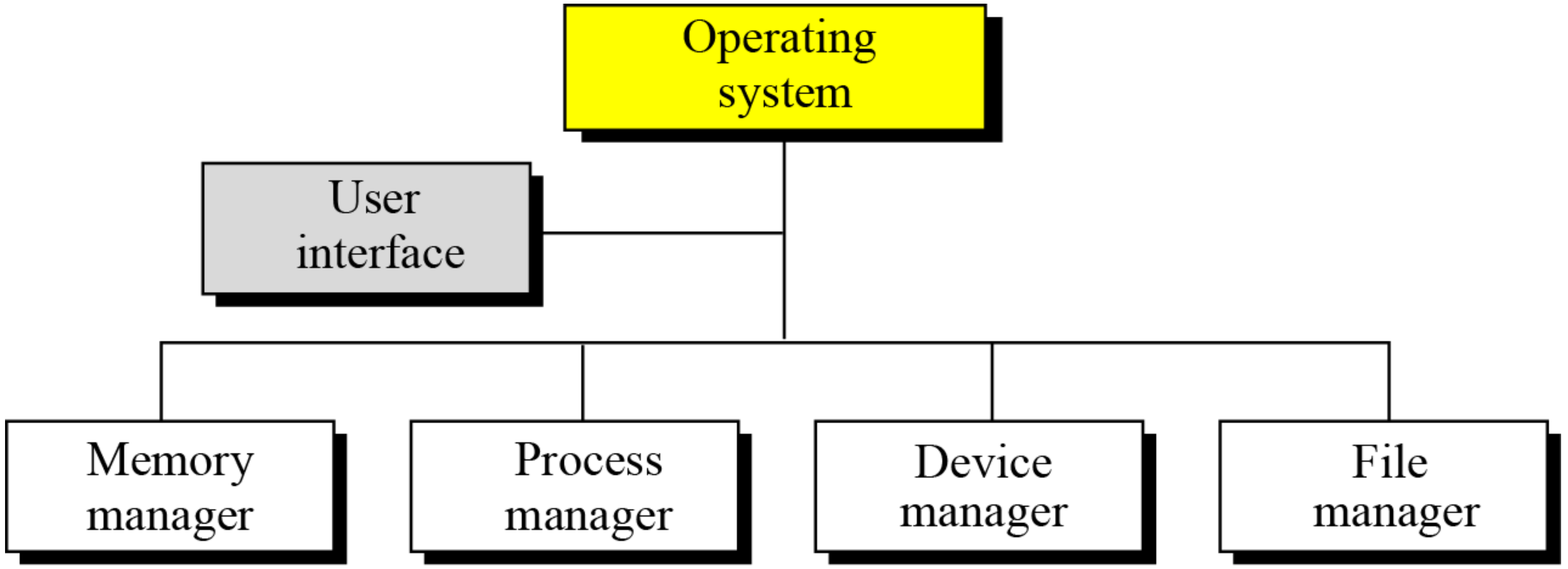


Figure 7.3 **Components of an operating system**

User interface

Each operating system has a **user interface**, a program that accepts requests from users (processes) and interprets them for the rest of the operating system. A user interface in some operating systems, such as UNIX, is called a **shell**. In others, it is called a **window** to denote that it is menu driven and has a GUI (graphical user interface) component.

Memory manager

One of the responsibilities of a modern computer system is **memory management**. Although the memory size of computers has increased tremendously in recent years, so has the size of the programs and data to be processed. Memory allocation must be managed to prevent applications from running out of memory. Operating systems can be divided into two broad categories of memory management: **monoprogramming** and ***multiprogramming***.

Monoprogramming

In **monoprogramming**, most of the memory capacity is dedicated to a single program; only a small part is needed to hold the operating system. In this configuration, the whole program is in memory for execution. When the program finishes running, the program area is occupied by another program.

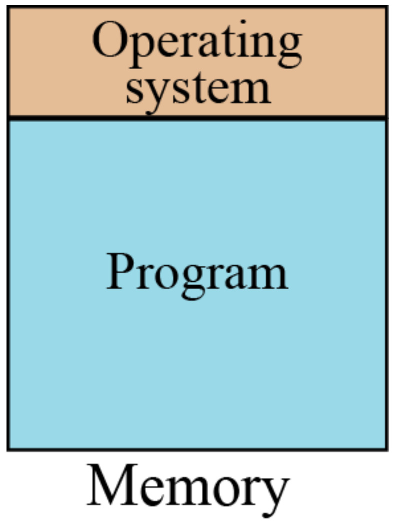


Figure 7.4 Monoprogramming

Multiprogramming

In multiprogramming, more than one program is in memory at the same time, and they are executed concurrently, with the CPU switching rapidly between the programs.

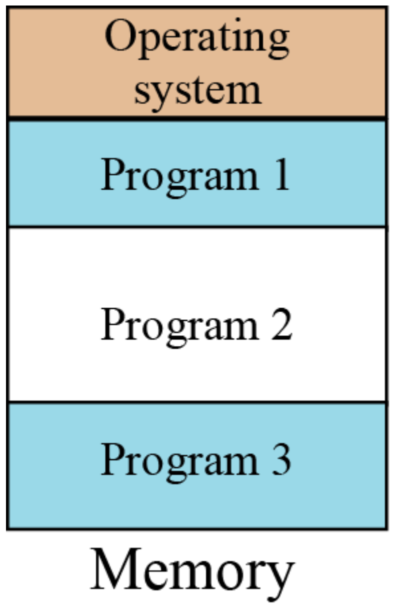


Figure 7.5 **Multiprogramming**

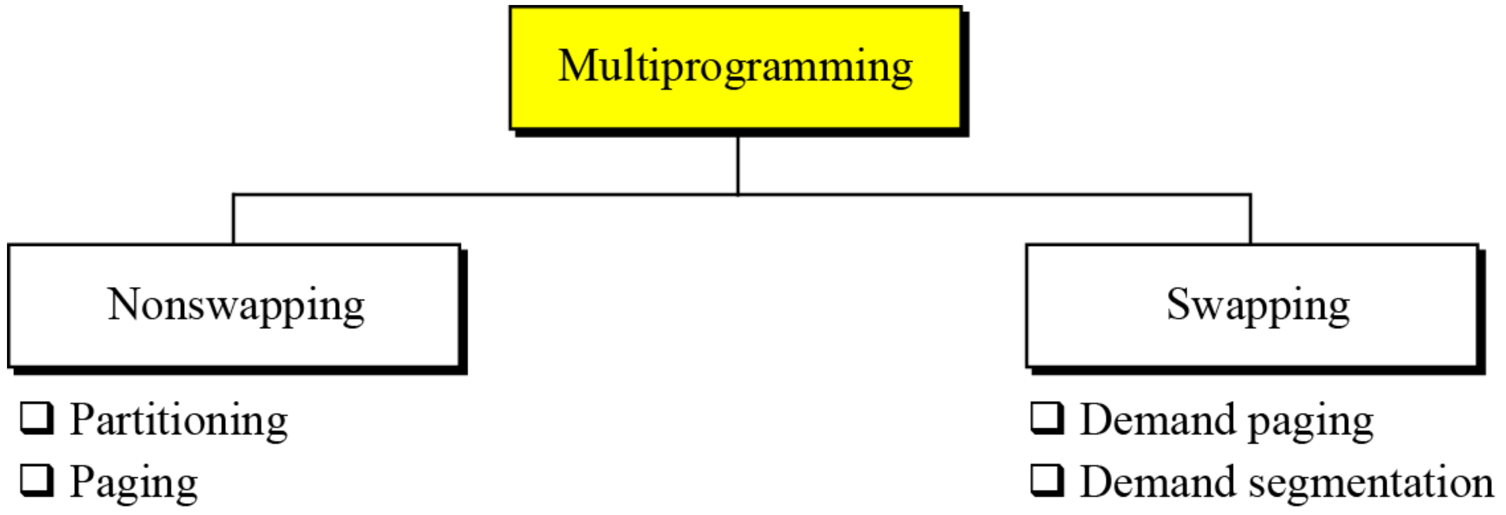


Figure 7.6 **Categories of multiprogramming**

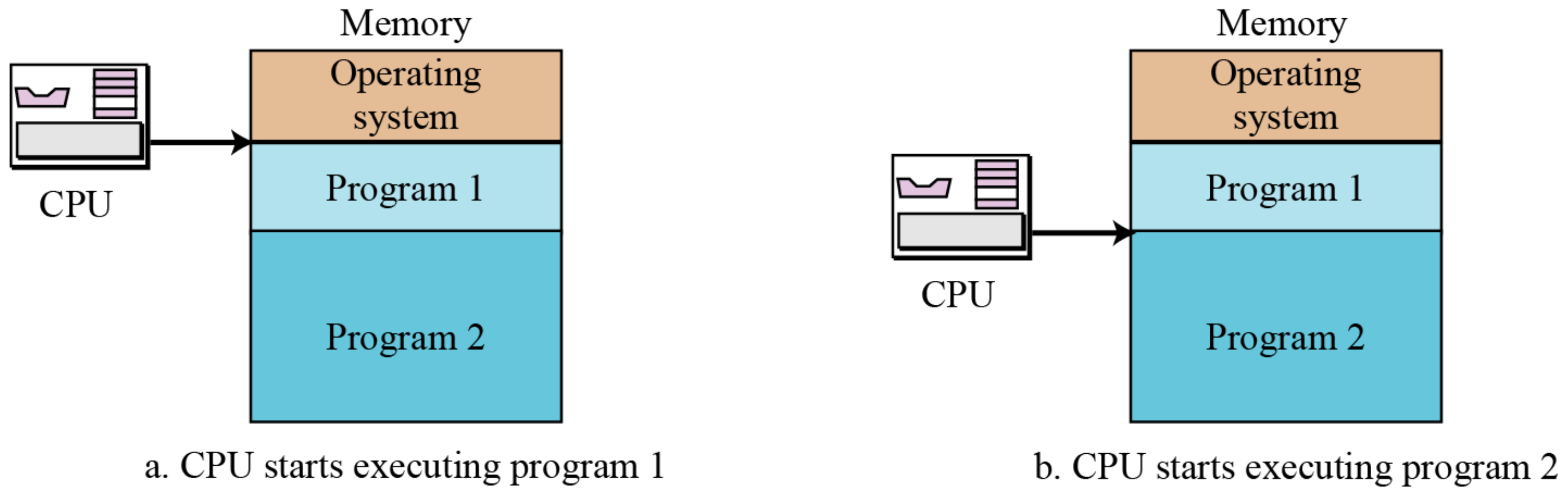


Figure 7.7 Partitioning

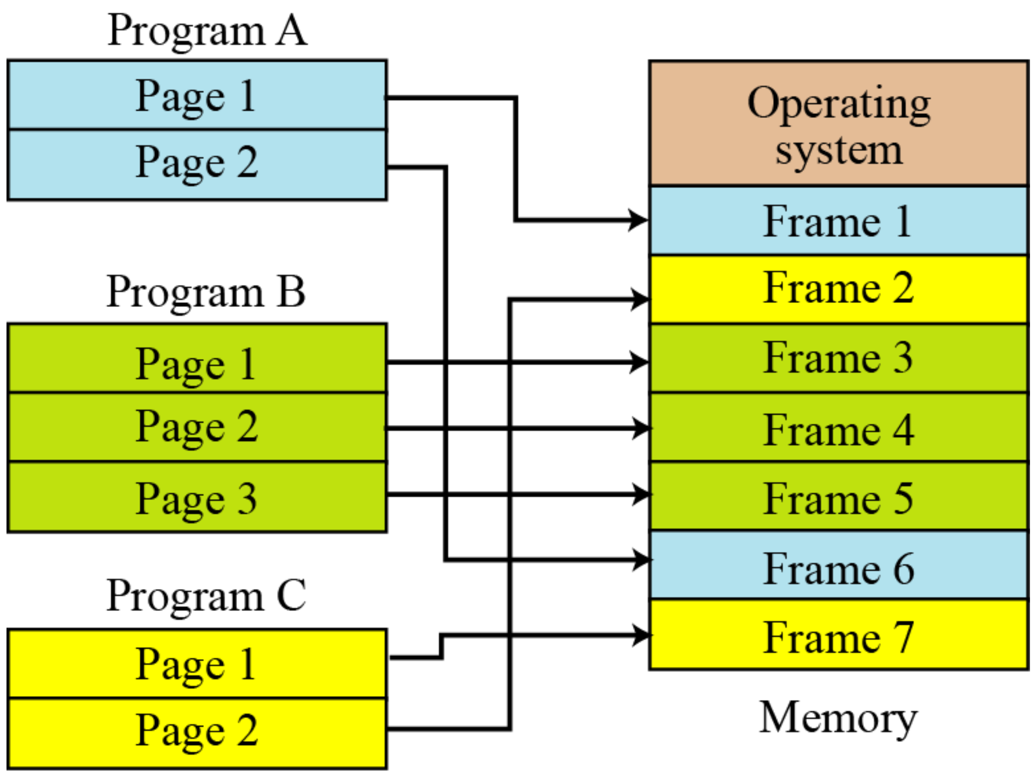


Figure 7.8 **Paging**

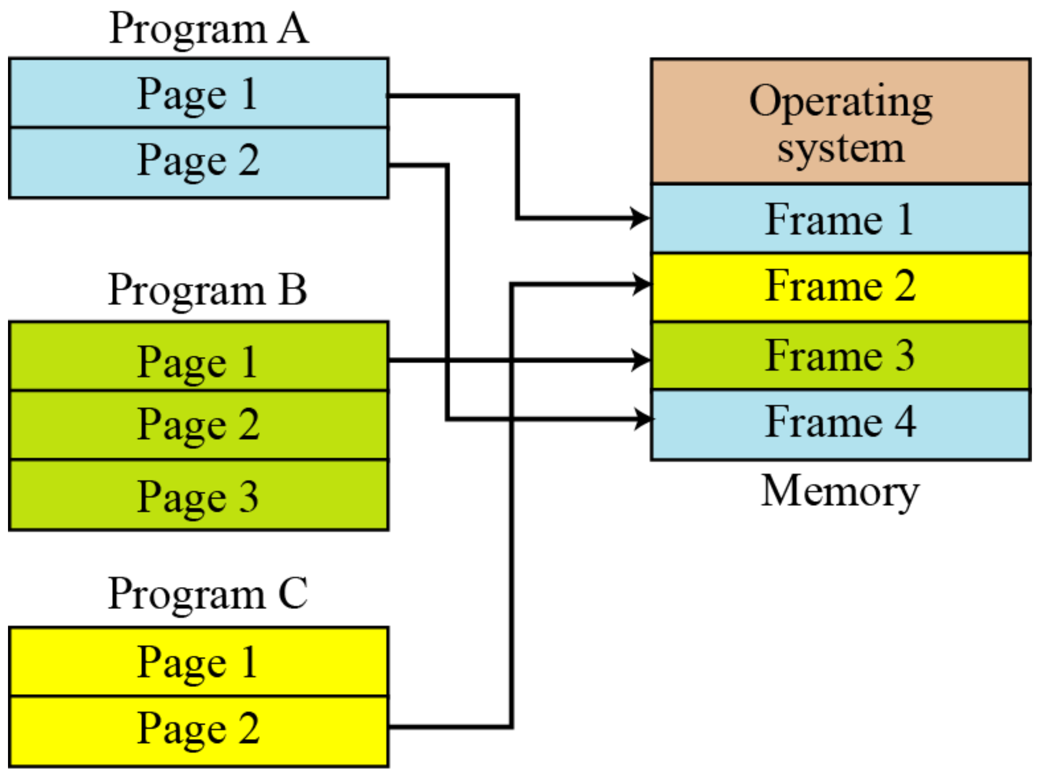


Figure 7.9 Demand paging

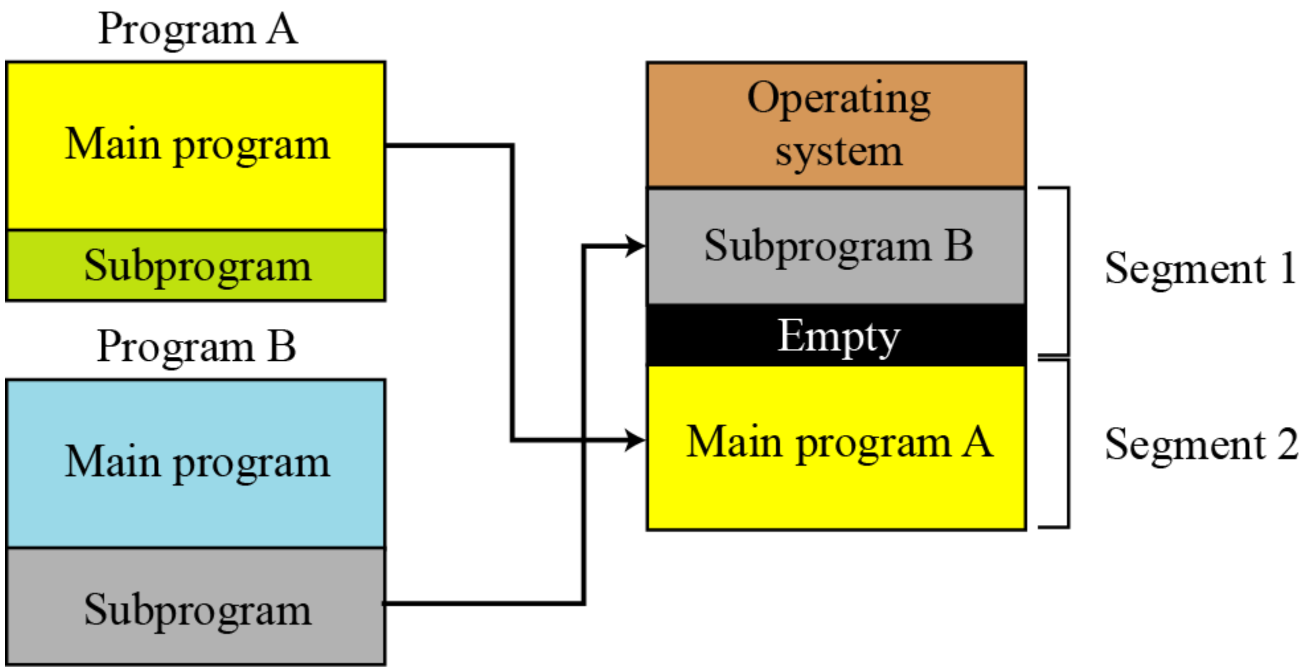


Figure 7.10 **Demand segmentation**

Virtual memory

Demand paging and demand segmentation mean that, when a program is being executed, part of the program is in memory and part is on disk. This means that, for example, a memory size of 10 MB can execute 10 programs, each of size 3 MB, for a total of 30 MB. At any moment, 10 MB of the 10 programs are in memory and 20 MB are on disk. There is therefore an actual memory size of 10 MB, but a virtual memory size of 30 MB. Figure 7.11 shows the concept. **Virtual memory**, which implies demand paging, demand segmentation or both, is used in almost all operating systems today.

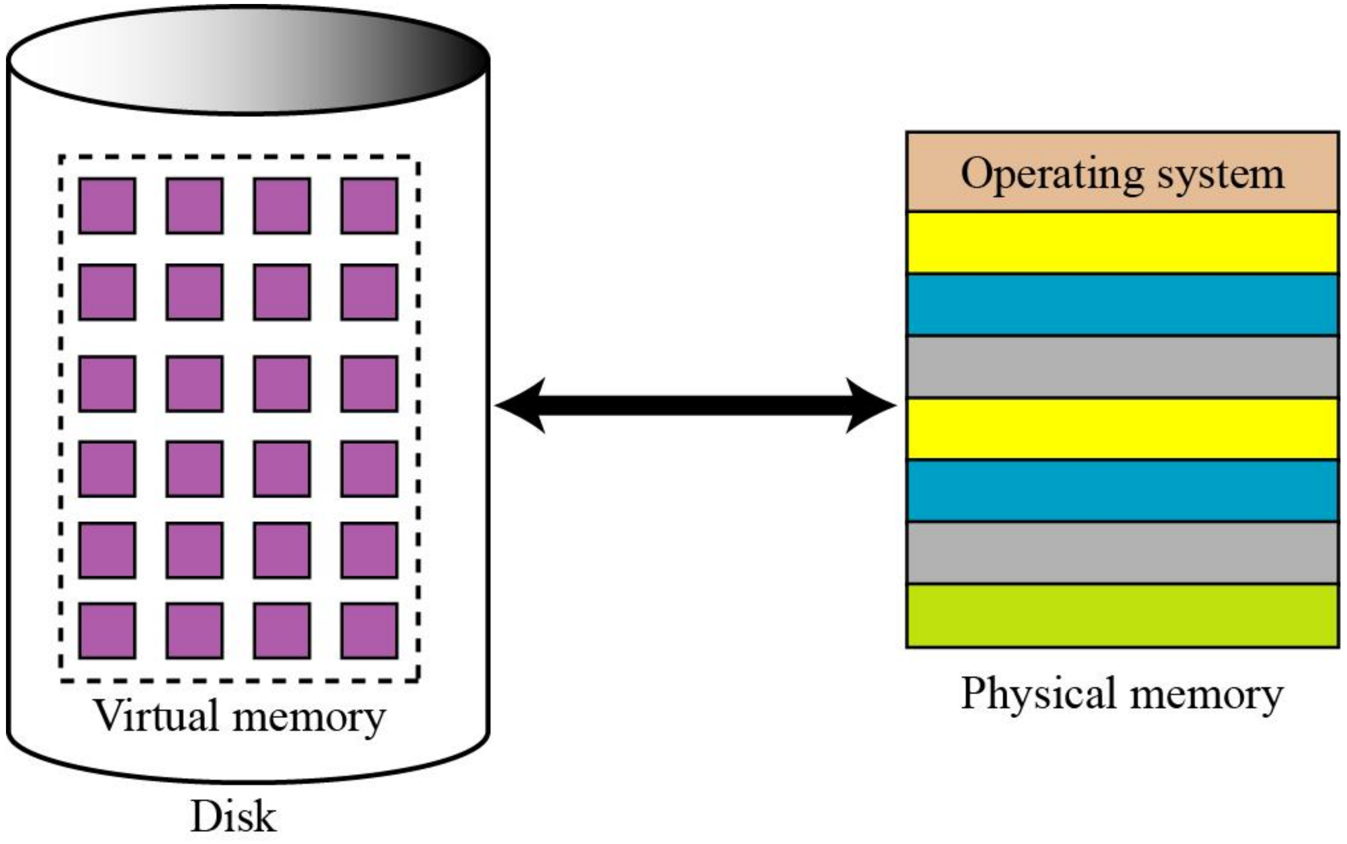


Figure 7.11 **Virtual memory**

Process manager

A second function of an operating system is process management, but before discussing this concept, we need to define some terms.

Program, job, and process

- ❑ A ***program*** is a non-active set of instructions stored on disk.
- ❑ A program becomes a ***job*** from the moment it is selected for execution until it has finished running and becomes a program again.
- ❑ A ***process*** is a program in execution. It is a program that has started but has not finished.

State diagrams

The relationship between a program, a job and a process becomes clearer if we consider how a program becomes a job and how a job becomes a process. This can be illustrated with a state diagram that shows the different states of each of these entities.

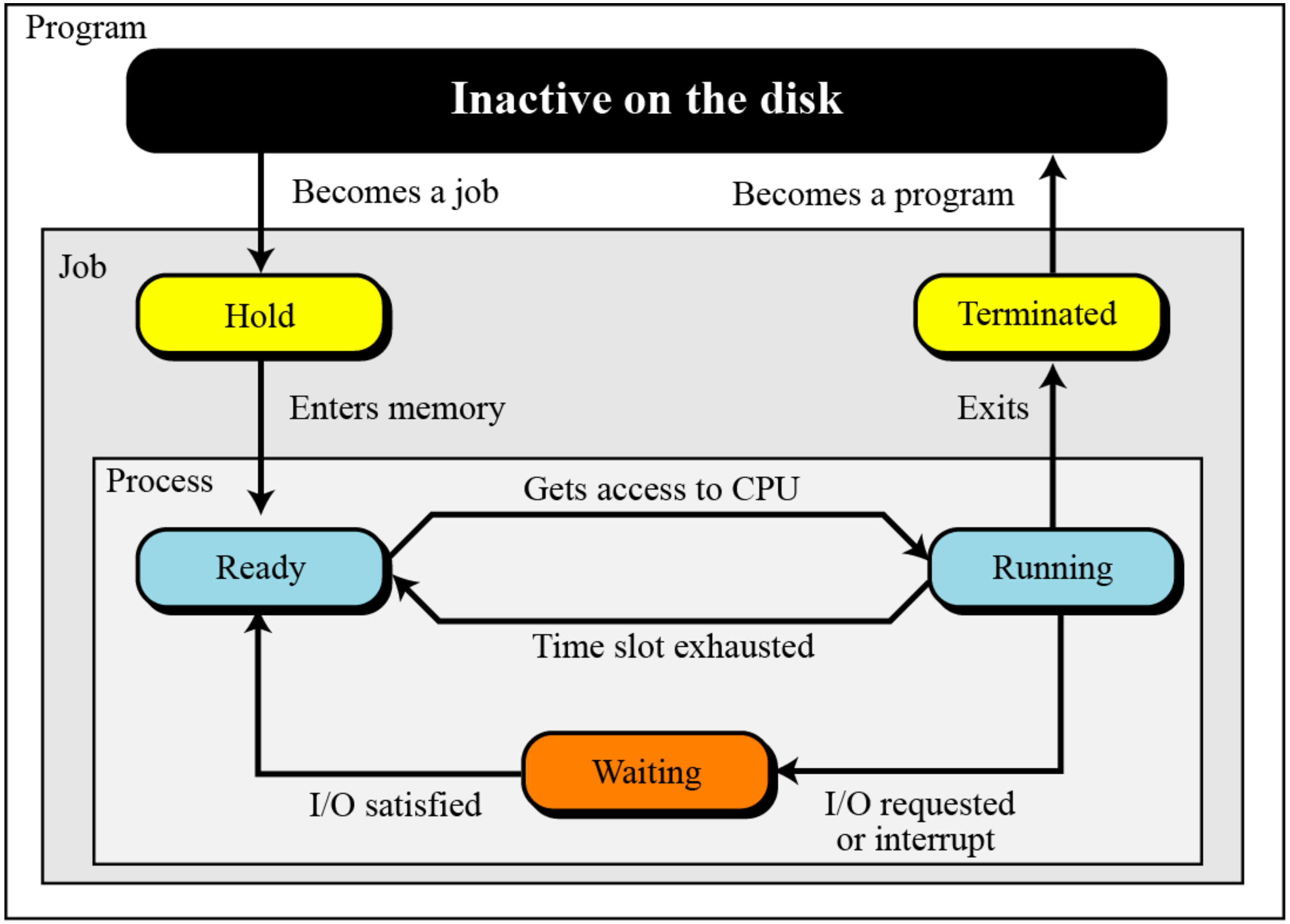


Figure 7.12 **State diagram with boundaries between program, job and process**

Schedulers

To move a job or process from one state to another, the process manager uses two schedulers: the ***job scheduler*** and the ***process scheduler***.

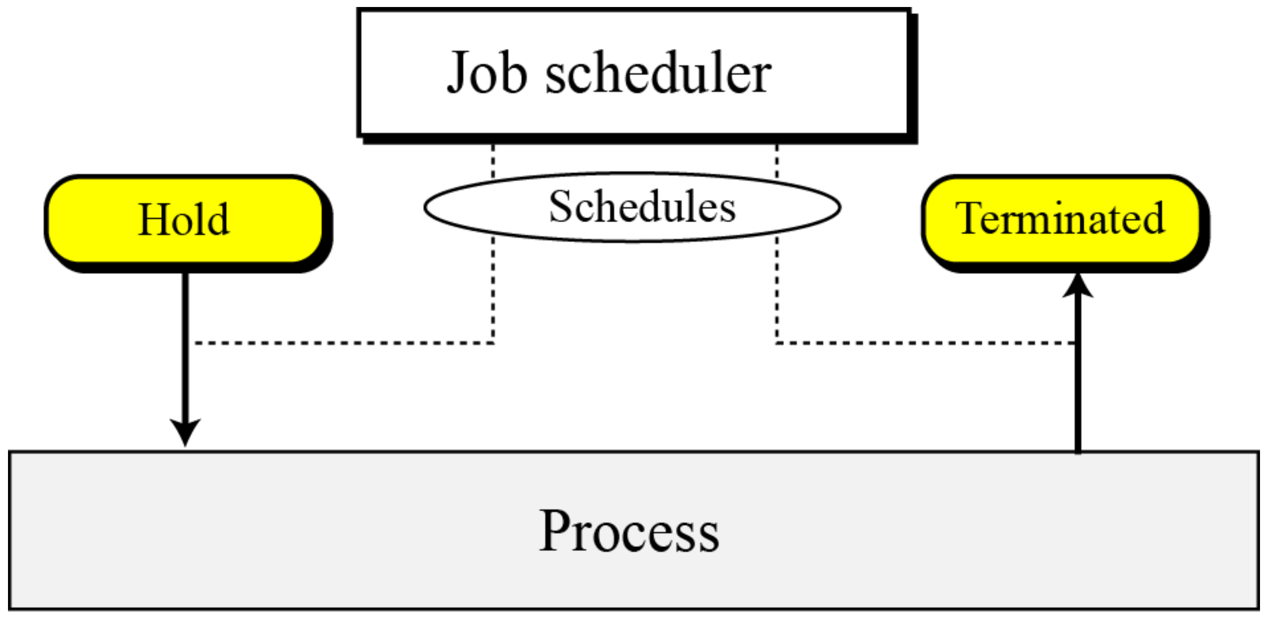


Figure 7.13 **Job scheduler**

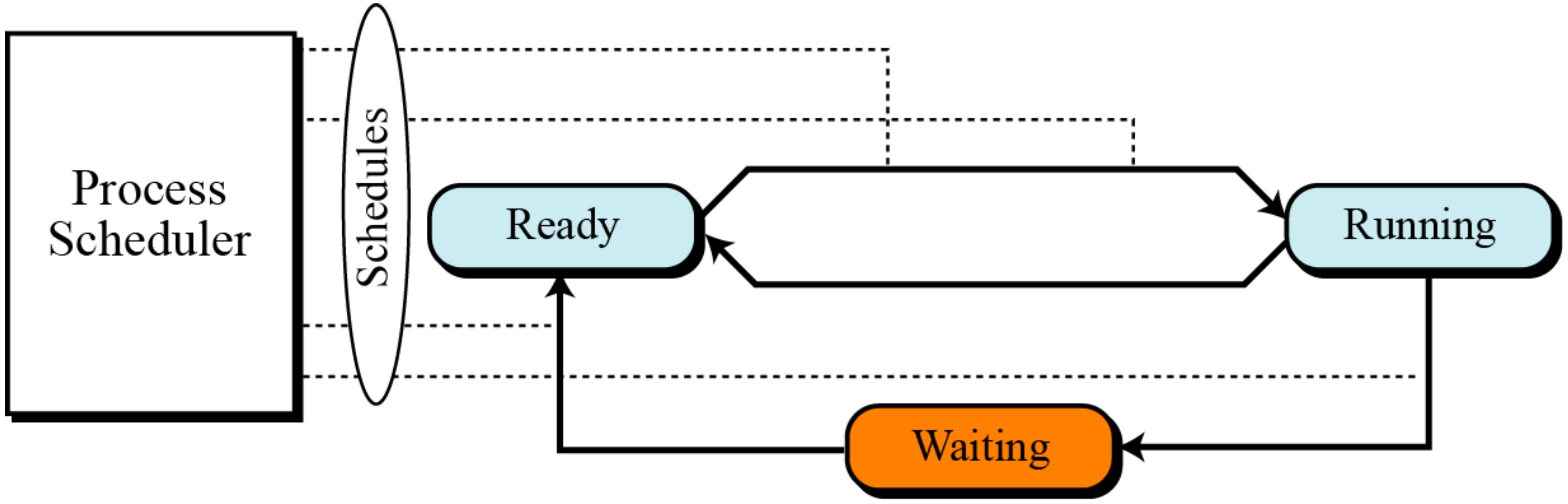


Figure 7.14 **Process scheduler**

Queuing

Our state diagram shows one job or process moving from one state to another. In reality, there are many jobs and many processes competing with each other for computer resources. To handle multiple processes and jobs, the process manager uses queues (waiting lists). A job control block or process control block is associated with each job or process. This is a block of memory that stores information about that job or process. The process manager stores the job or process control block in the queues instead of the job or process itself.

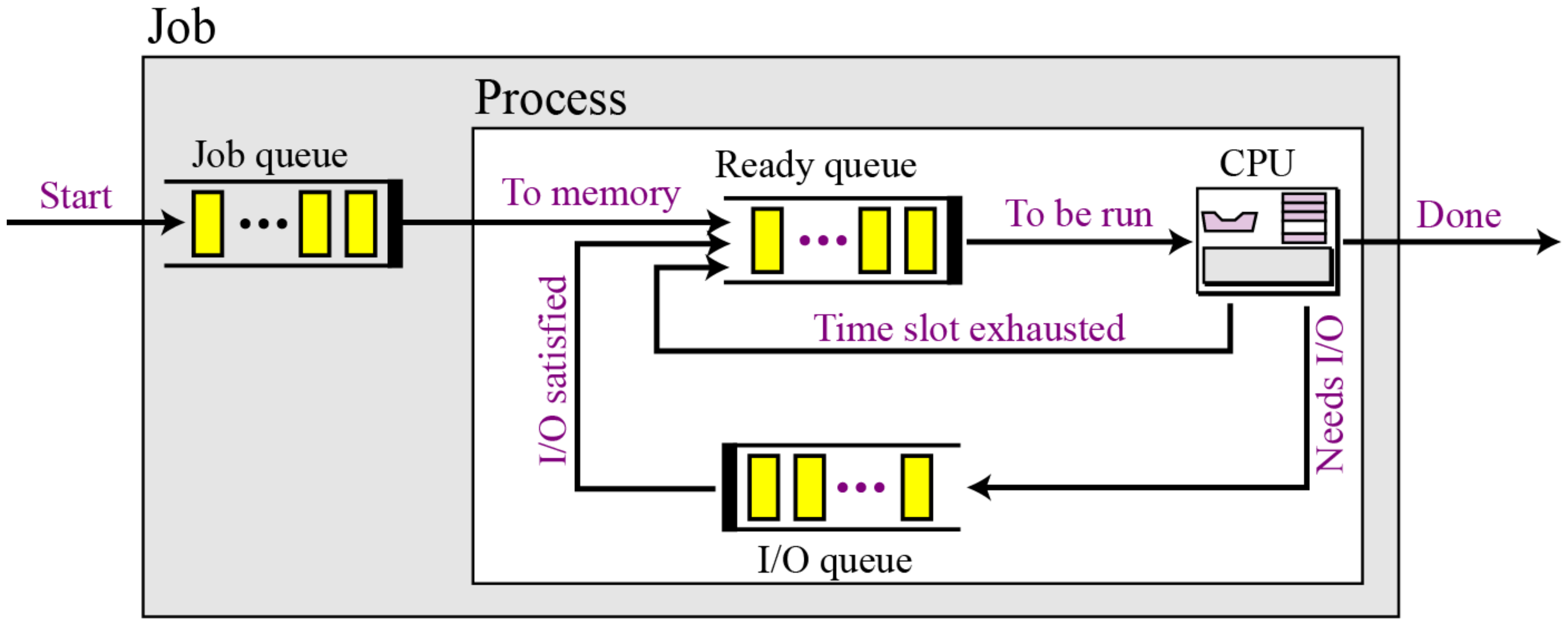


Figure 7.15 **Queues for process management**

Process synchronization

The whole idea behind process management is to synchronize different processes with different resources. Whenever resources can be used by more than one user (or process, in this case), we can have two problematic situations: ***deadlock*** and ***starvation***.

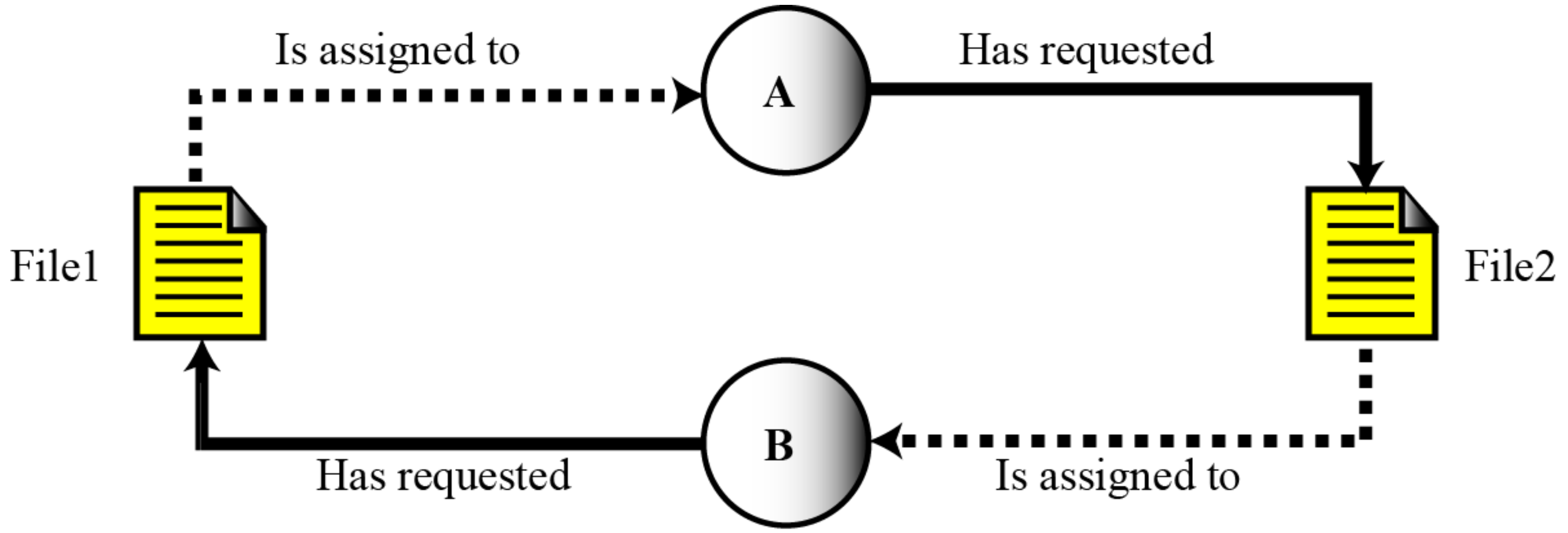


Figure 7.16 **Deadlock**

Deadlock occurs when the operating system does not put resource restrictions on processes.

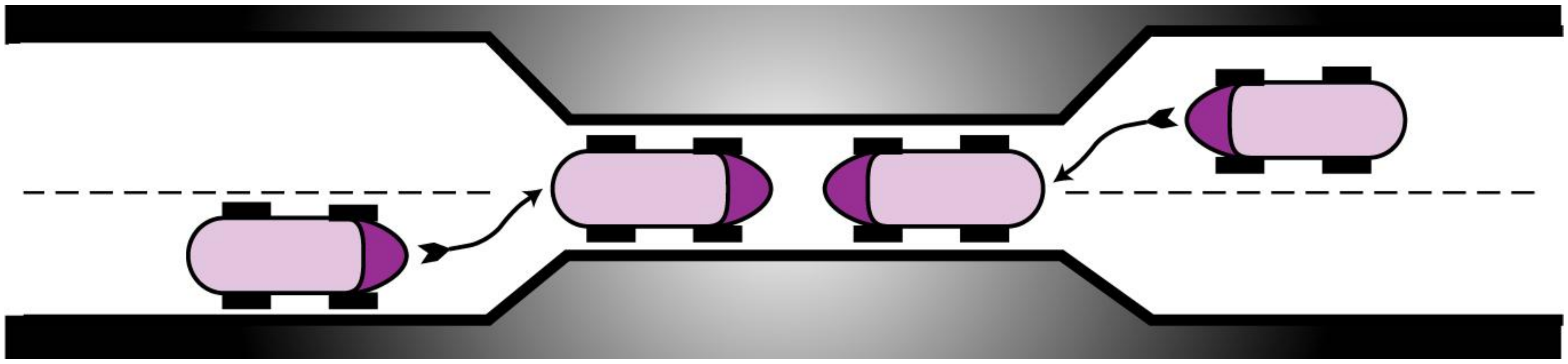
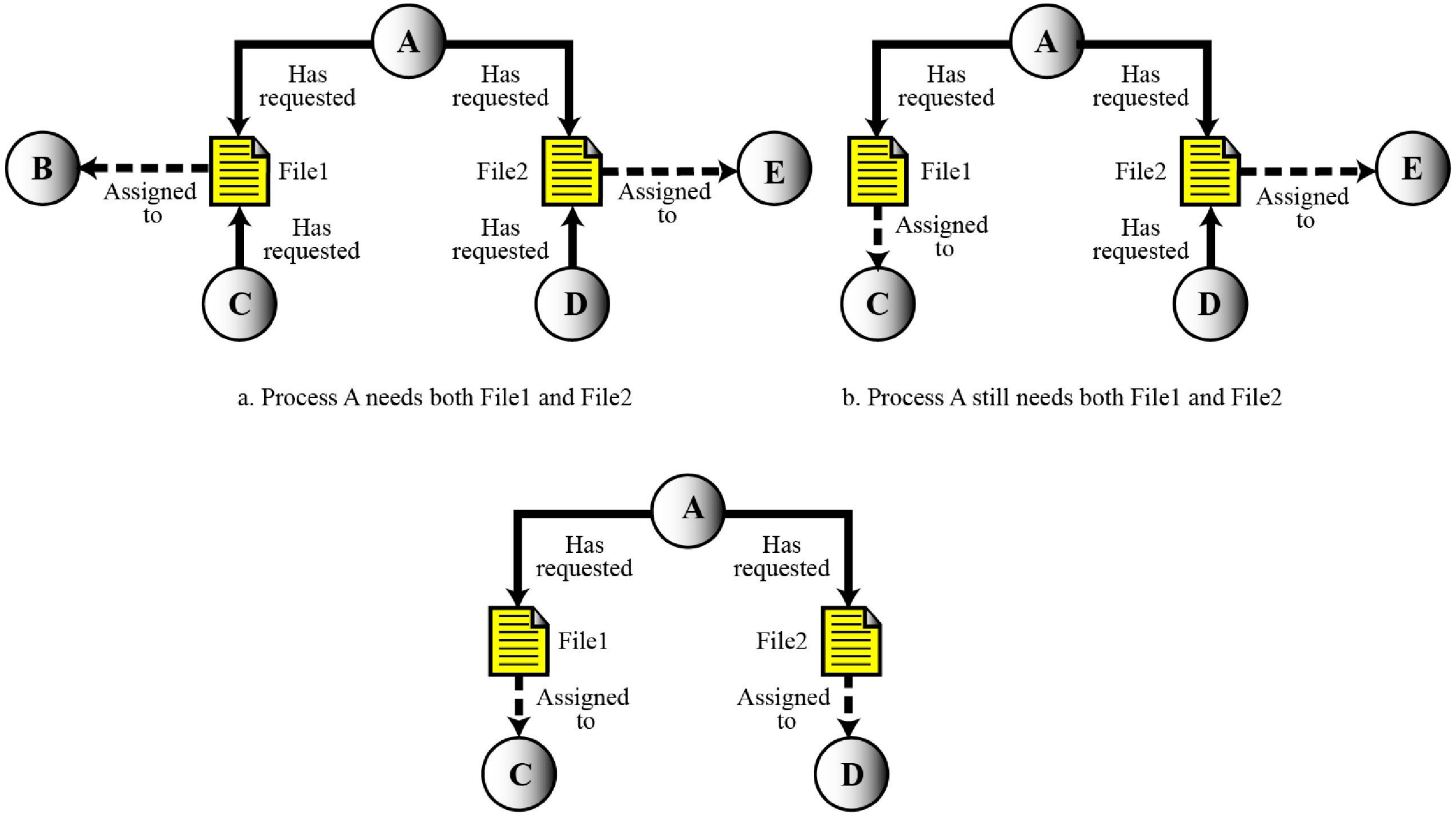


Figure 7.17 **Deadlock on a bridge**



a. Process A needs both File1 and File2

b. Process A still needs both File1 and File2

c. Process A still needs both File1 and File2 (starving)

Figure 7.18 **Starvation**

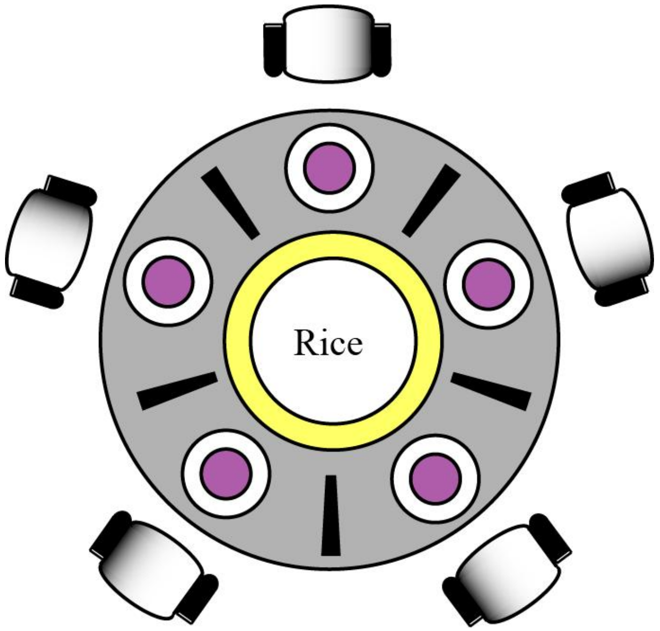


Figure 7.19 **The dining philosophers problem**

i

Starvation is the opposite of deadlock. It can happen when the operating system puts too many resource restrictions on a process.

Device manager

The device manager, or input/output manager, is responsible for access to input/ output devices. There are limitations on the number and speed of input/output devices in a computer system.

- ❑ **The device manager monitors every input/output device constantly to ensure that the device is functioning properly.**
- ❑ **The device manager maintains a queue for each input/output device or one or more queues for similar input/output devices.**
- ❑ **The device manager controls the different policies for accessing input/output devices.**

File manager

Operating systems today use a **file manager** to control access to files. A detailed discussion of the file manager also requires advanced knowledge of operating system principles and file access concepts that are beyond the scope of this book. The file manager:

- ❑ **controls access to files.**
- ❑ **supervises the creation, deletion, and modification of files.**
- ❑ **controls the naming of files.**
- ❑ **supervises the storage of files.**
- ❑ **is responsible for archiving and backups.**

7-4 A SURVEY OF OPERATING SYSTEMS

In this section we introduce some popular operating systems and encourage you to study them further. We have chosen three operating systems that are familiar to most computer users: UNIX, Linux and Windows.

UNIX

UNIX was originally developed in 1969 by Thomson and Ritchie of the Computer Science Research Group at Bell Laboratories. UNIX has gone through many versions since then. It has been a popular operating system among computer programmers and computer scientists.

UNIX is a multiuser, multiprocessing, portable operating system.
It is designed to facilitate programming, text processing and communication.

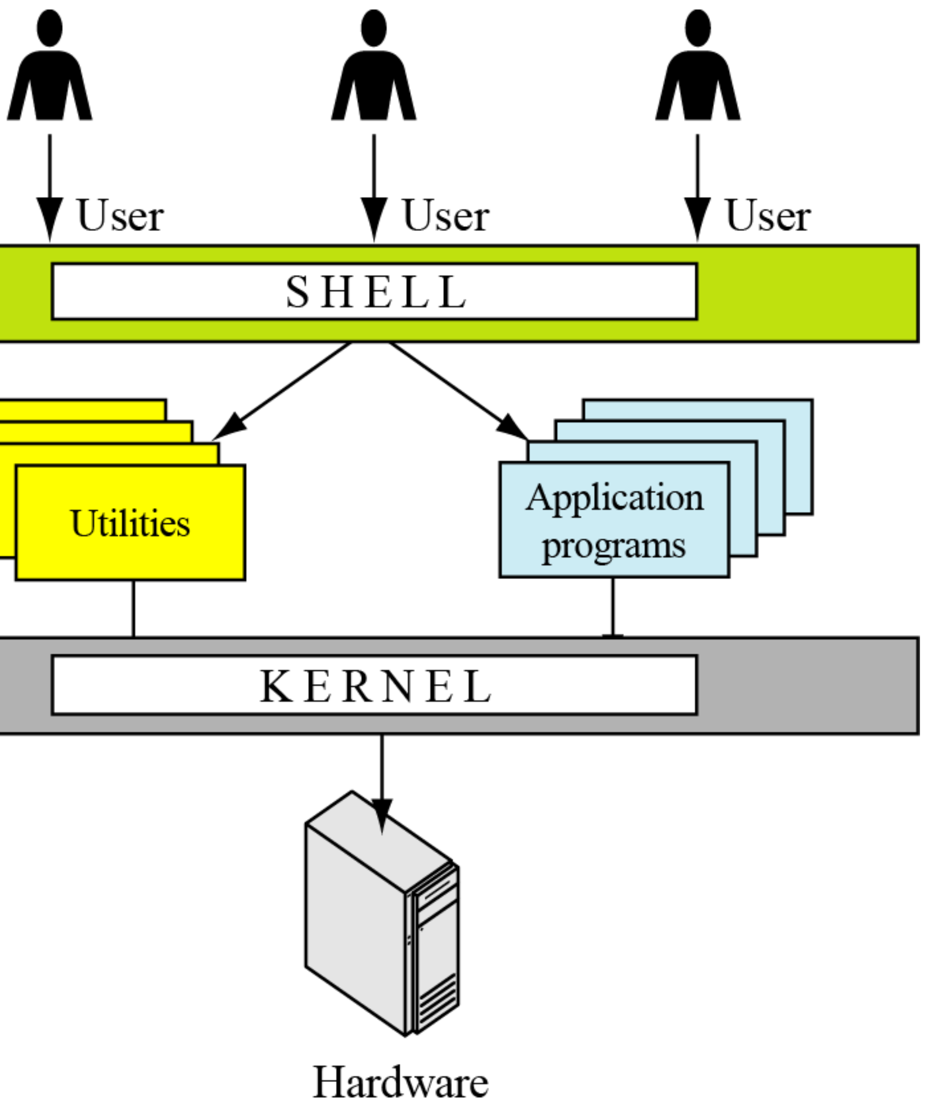


Figure 7.20 **Components of the UNIX operating system**

Linux

In 1991, Linus Torvalds, a Finish student at the University of Helsinki at the time, developed a new operating system that is known today as Linux. The initial kernel, which was similar to a small subset of UNIX, has grown into a full-scale operating system today. The Linux 2.0 kernel, released in 1997, was accepted as a commercial operating system: it has all features traditionally attributed to UNIX.

Windows NT/2000/XP

In the late 1980s Microsoft, under the leadership of Dave Cutler, started development of a new single-user operating system to replace MS-DOS (Microsoft Disk Operating System). Windows NT (NT standing for New Technology) was the result. Several versions of Windows NT followed and the name was changed to Windows 2000. Windows XP (XP stands for eXPerience) was released in 2001. We refer to all of these versions as Windows NT or just NT.

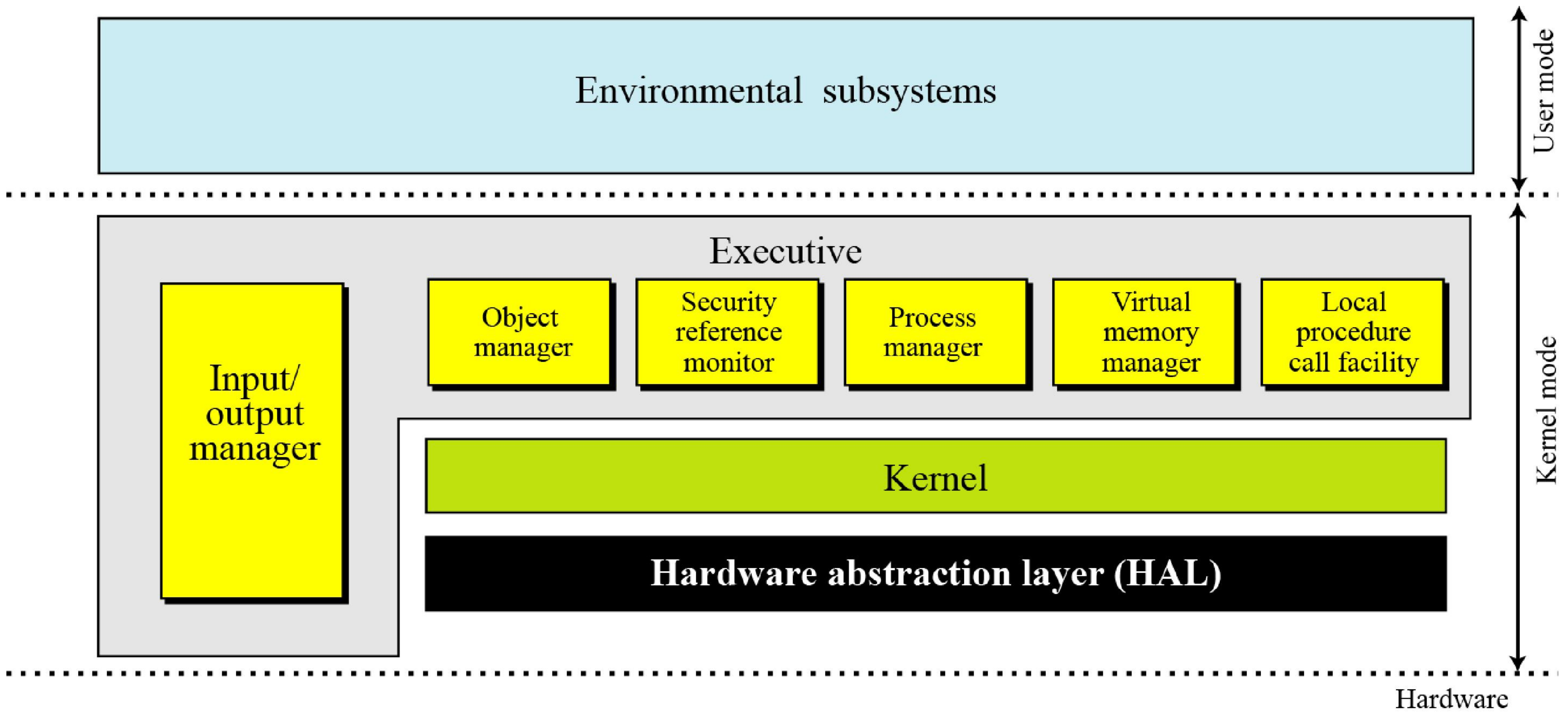


Figure 7.21 **The architecture of Windows NT**