Process concept

- Process is a program in execution.
- A process is the unit of work in a modern time-sharing system
- A process is more than the program code, which is sometimes known as the text section .
- It also includes the current activity, as represented by the value of the program counter and the contents of the processor's registers.
- A process generally also includes the process stack, which contains temporary data (such as function parameters, return addresses, and local variables), and a data section, which contains global variables.
- A process may also include a heap, which is memory that is dynamically allocated during process run time.
- The structure of a process in memory is shown in Figure 3.1.

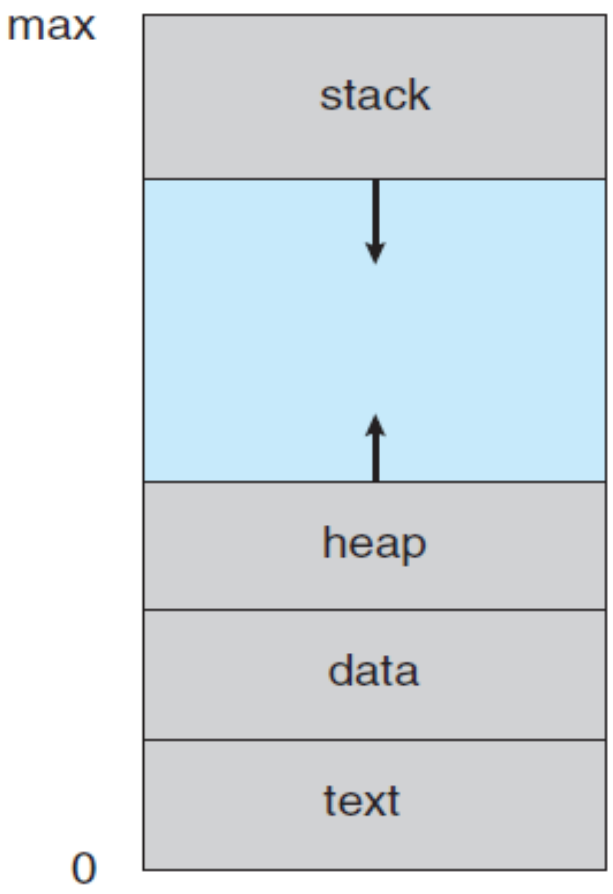

Figure 3.1 Process in memory.

- We emphasize that a program by itself is not a process. A program is a *passive entity, such as a file containing a list of instructions stored on disk* (often called an executable file).
- In contrast, a process is an *active entity,* with a program counter specifying the next instruction to execute and a set of associated resources.
- A program becomes a process when an executable file is loaded into memory.
- Two common techniques for loading executable files are double-clicking an icon representing the executable file and entering the name of the executable file on the command line (as in prog.exe or a.out).

Process State

- As a process executes, it changes **state.**
- The state of a process is defined in part by the current activity of that process. A process may be in one of the following states:
- **New.** The process is being created**.**
- **Running**. Instructions are being executed.
- **Waiting.** The process is waiting for some event to occur (such as an I/O completion or reception of a signal).
- **Ready.** The process is waiting to be assigned to a processor.
- **Terminated.** The process has finished execution**.**

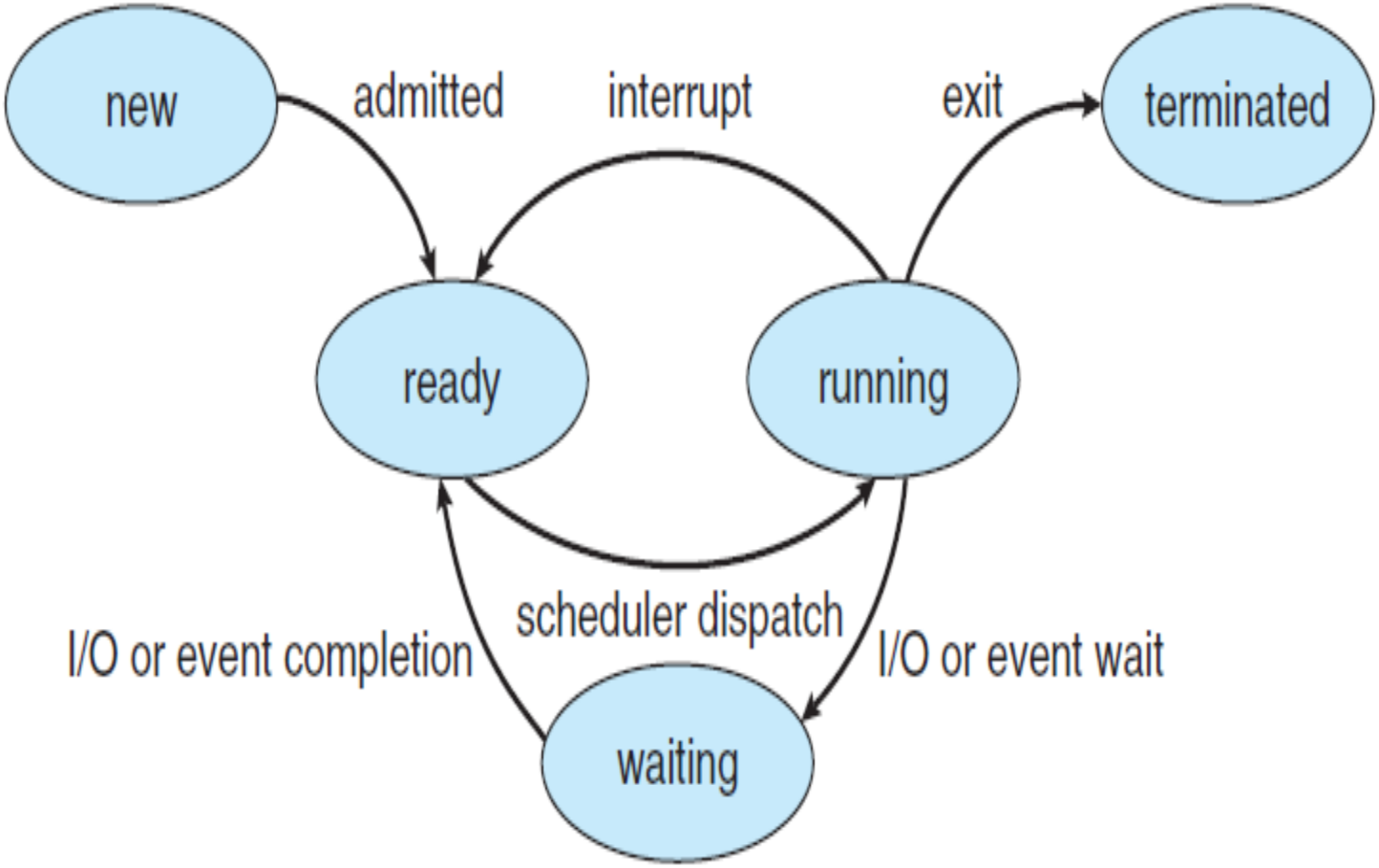

Figure 3.2 Diagram of process state.

Process Control Block

- Each process is represented in the operating system by a process control block (PCB)—also called a task control block.
- A PCB contains many pieces of information associated with a specific process, including these

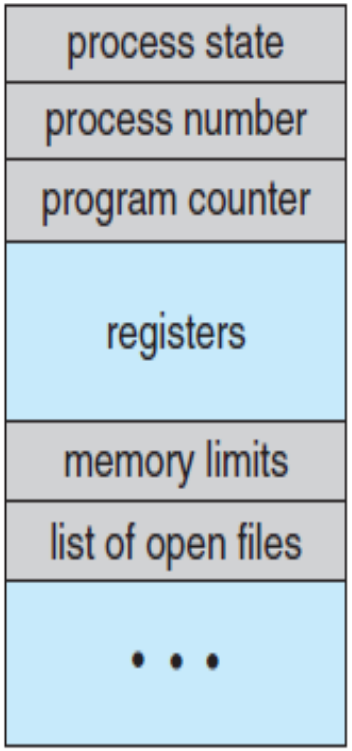

Figure 3.3 Process control block (PCB).

- **Process state.** The state may be new, ready, running, waiting, halted, and so on.
- **Program counter.** The counter indicates the address of the next instruction to be executed for this process.
- **CPU registers.** The registers vary in number and type, depending on the computer architecture. They include accumulators, index registers, stack pointers, and general-purpose registers, plus any condition-code information. Along with the program counter, this state information must be saved when an interrupt occurs, to allow the process to be continued correctly afterward (Figure 3.4).

- **CPU-scheduling information**. This information includes a process priority, pointers to scheduling queues, and any other scheduling parameters. (Chapter 6 describes process scheduling.)
- **Memory-management information**. This information may include such items as the value of the base and limit registers and the page tables, or the segment tables, depending on the memory system used by the operating system (Chapter 8).
- **Accounting information**. This information includes the amount of CPU and real time used, time limits, account numbers, job or process numbers, and so on.
- **I/O status information**. This information includes the list of I/O devices allocated to the process, a list of open files, and so on.

In brief, the PCB simply serves as the repository for any information that may vary from process to process.

Process Scheduling

- The objective of multiprogramming is to have some process running at all times, to maximize CPU utilization.
- The objective of time sharing is to switch the CPU among processes so frequently that users can interact with each program while it is running.
- To meet these objectives, the **process scheduler selects** an available process (possibly from a set of several available processes) for program execution on the CPU.
- For a single-processor system, there will never be more than one running process. If there are more processes, the rest will have to wait until the CPU is free and can be rescheduled.

Scheduling Queues

- As processes enter the system, they are put into a job queue, which consists of all processes in the system.
- The processes that are residing in main memory and are ready and waiting to execute are kept on a list called the ready queue.
- This queue is generally stored as a linked list.
- A ready-queue header contains pointers to the first and final PCBs in the list.
- Each PCB includes a pointer field that points to the next PCB in the ready queue.
- The list of processes waiting for a particular I/O device is called a device queue.
- Each device has its own device queue

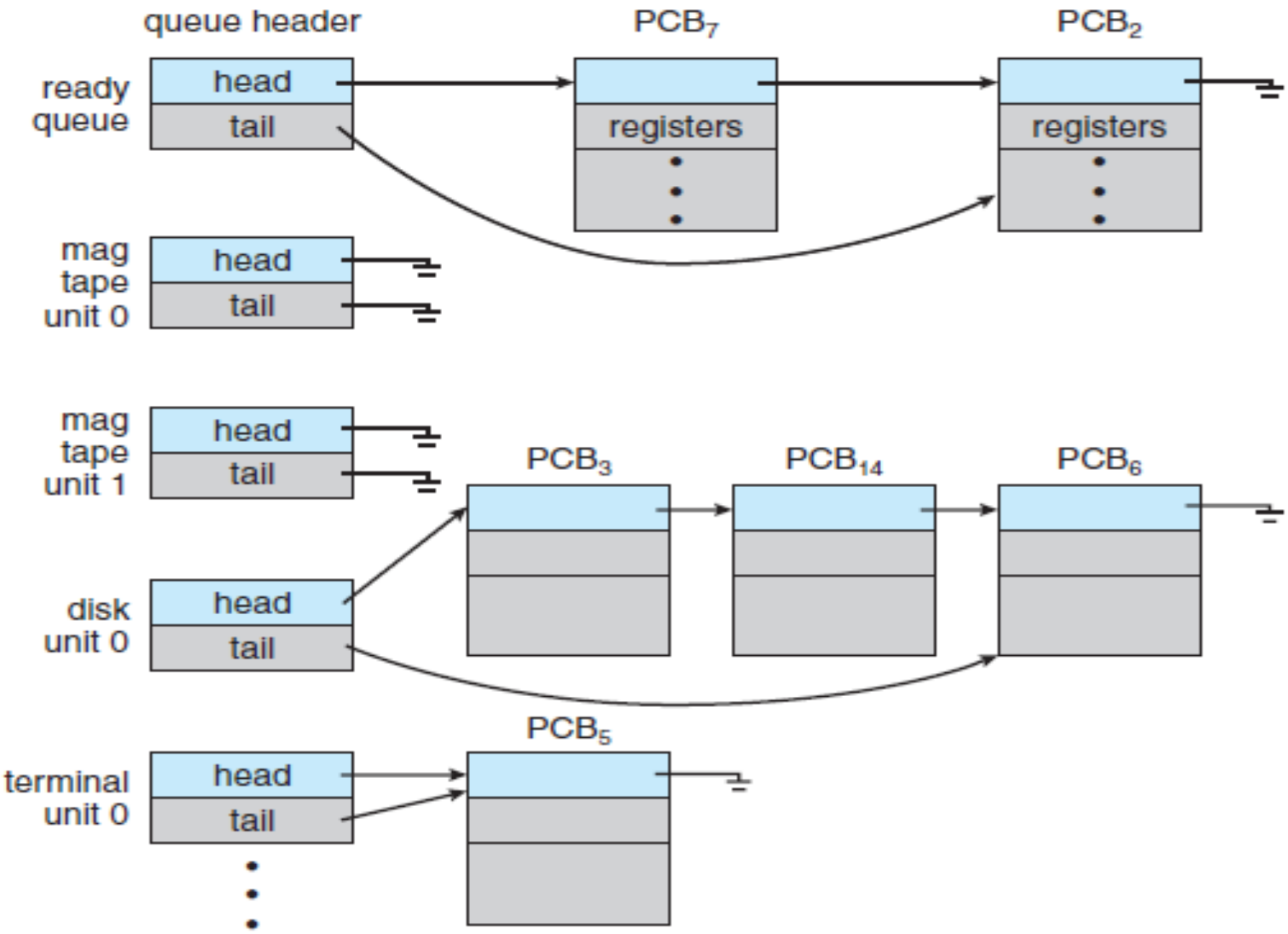

Figure 3.5 The ready queue and various I/O device queues.

- A common representation of process scheduling is a queueing diagram(in Figure 3.6)
- Each rectangular box represents a queue.
- Two types of queues are present: the ready queue and a set of device queues.
- The circles represent the resources that serve the queues, and the arrows indicate the flow of processes in the system.
- A new process is initially put in the ready queue. It waits there until it is selected for execution, or dispatched.
- Once the process is allocated the CPU and is executing, one of several events could occur:

→ The process could issue an I/O request and then be placed in an I/O queue.

→ The process could create a new child process and wait for the child's termination.

→ The process could be removed forcibly from the CPU, as a result of an interrupt, and be put back in the ready queue.

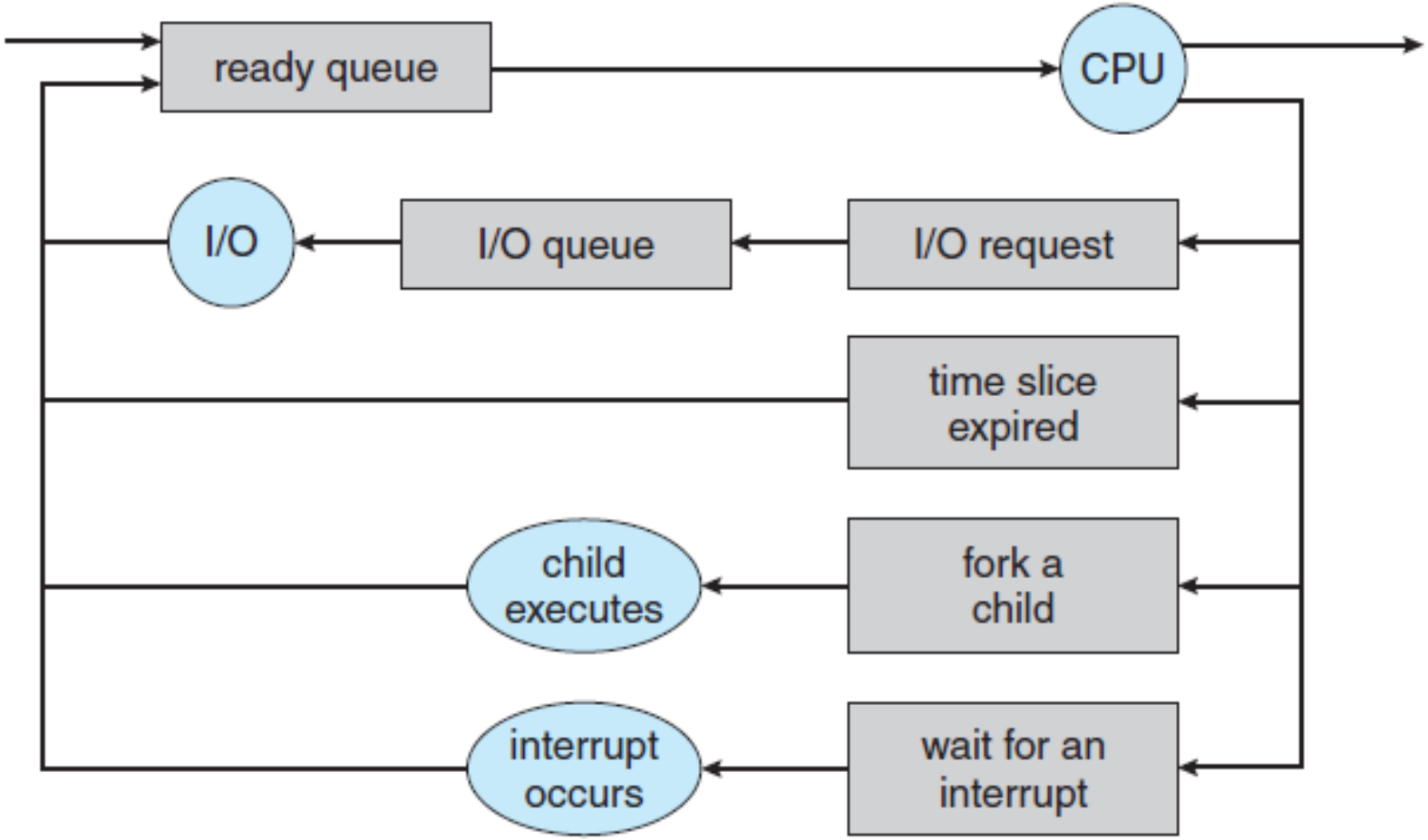

Figure 3.6 Queueing-diagram representation of process scheduling.

Schedulers

- A process migrates among the various scheduling queues throughout its lifetime.
- The operating system must select, for scheduling purposes, processes from these queues in some fashion.
- The selection process is carried out by the appropriate **scheduler.**
- Often, in a batch system, more processes are submitted than can be executed immediately. These processes are spooled to a mass-storage device (typically a disk), where they are kept for later execution.
- The **long-term scheduler, or job scheduler,** selects processes from this pool and loads them into memory for execution. The **short-term scheduler, or CPU scheduler,** selects from among the processes that are ready to execute and allocates the CPU to one of them.

- The primary distinction between these two schedulers lies in frequency of execution.
- The short-term scheduler must select a new process for the CPU frequently.
- A process may execute for only a few milliseconds before waiting for an I/O request.
- Often, the short-term scheduler executes at least once every 100 milliseconds.
- Because of the short time between executions, the short-term scheduler must be fast.
- If it takes 10 milliseconds to decide to execute a process for 100 milliseconds, then 10/(100 + 10) = 9 percent of the CPU is being used (wasted) simply for scheduling the work.

- The long-term scheduler executes much less frequently; minutes may separate the creation of one new process and the next.
- The long-term scheduler controls the degree of multiprogramming (the number of processes in memory).
- If the degree of multiprogramming is stable, then the average rate of process creation must be equal to the average departure rate of processes leaving the system.
- Thus, the long-term scheduler may need to be invoked only when a process leaves the system.
- Because of the longer interval between executions, the long-term scheduler can afford to take more time to decide which process should be selected for execution

- It is important that the long-term scheduler make a careful selection. In
- general, most processes can be described as either I/O bound or CPU bound.
- An I/O-bound process is one that spends more of its time doing I/O than
- it spends doing computations.
- A CPU-bound process, in contrast, generates I/O requests infrequently, using more of its time doing computations.
- It is important that the long-term scheduler select a good *process mix of I/O-bound* and CPU-bound processes.
- If all processes are I/O bound, the ready queue will almost always be empty, and the short-term scheduler will have little to do.
- If all processes are CPU bound, the I/O waiting queue will almost always be empty, devices will go unused, and again the system will be unbalanced.
- The system with the best performance will thus have a combination of CPU-bound and I/O-bound processes.

- Some operating systems, such as time-sharing systems, may introduce an
- additional, intermediate level of scheduling.
- This **medium-term scheduler** -- The key idea behind a medium-term scheduler is that sometimes it can be advantageous to remove a process from memory (and from active contention for the CPU) and thus reduce the degree of multiprogramming.
- Later, the process can be reintroduced into memory, and its execution can be continued where it left off. This scheme is called **swapping.**
- The process is swapped out, and is later swapped in, by the medium-term scheduler.
- Swapping may be necessary to improve the process mix or because
- a change in memory requirements has overcommitted available memory, requiring memory to be freed up.

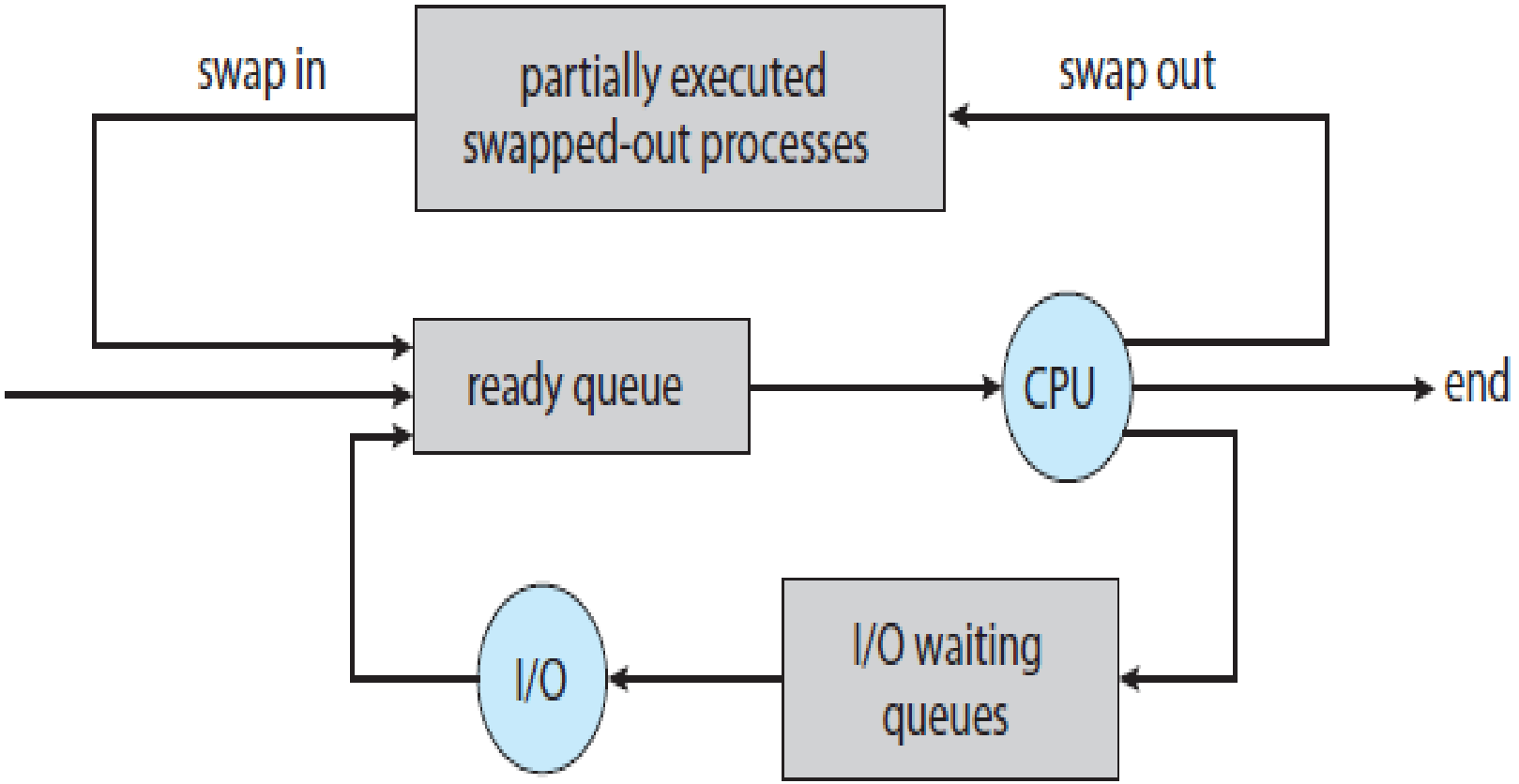

Figure 3.7 Addition of medium-term scheduling to the queueing diagram.

Context Switch

- When an interrupt occurs, the system needs to save the current context of the process running on the CPU so that it can res tore that context when its processing is done, essentially suspending the process and then resuming it.
- The context is represented in the PCB of the process. It includes the value of the CPU registers, the process state and memory-management information.
- Generically, we perform a state save of the current state of the CPU, be it in kernel or user mode, and then a state restore to resume operations.
- Switching the CPU to another process requires performing a state save of the current process and a state restore of a different process. This task is known as a context switch.

- When a context switch occurs, the kernel saves the context of the old process in its PCB and loads the saved context of the new process scheduled to run.
- Context-switch time is pure overhead, because the system does no useful work while switching.
- Switching speed varies from machine to machine, depending on the memory speed, the number of registers that must be copied, and the existence of special instructions (such as a single instruction to load or store all registers).