Operating Systems: Internals and Design Principles

Chapter 2
Operating System Overview

Ninth Edition

By William Stallings

Operating System

- An OS is a program that controls the execution of application programs
- Acts as an interface between applications and hardware

Main objectives of an OS:

- Convenience
- Efficiency
- Ability to evolve

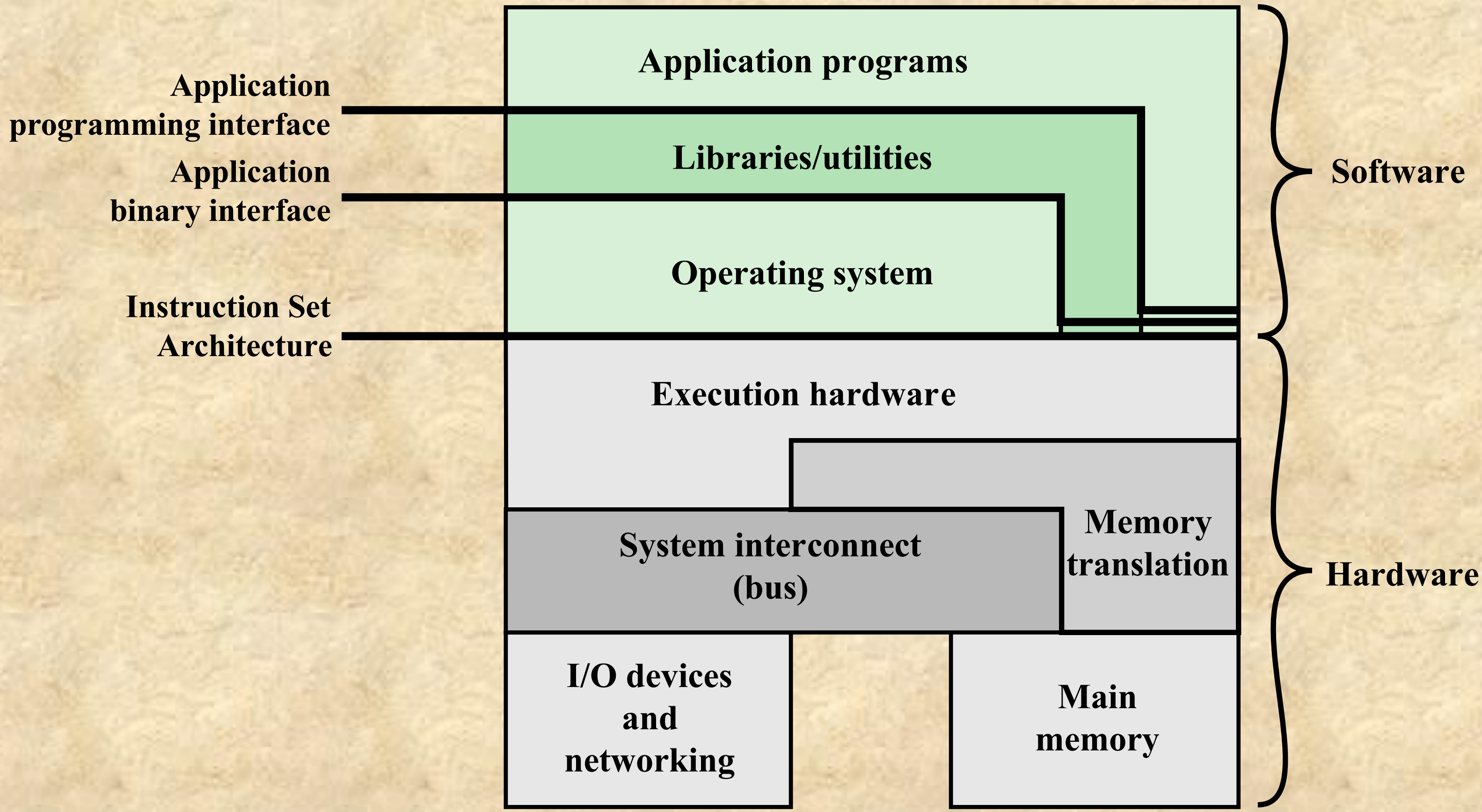


Figure 2.1 Computer Hardware and Software Structure

Operating System Services

- Program development
- Program execution
- Access I/O devices
- Controlled access to files
- System access
- Error detection and response
- Accounting

Key Interfaces

- Instruction set architecture (ISA): The ISA defines the repertoire of machine language instructions that a computer can follow. This interface is the boundary between hardware and software
- Application binary interface (ABI): The ABI defines a standard for binary portability across programs. The ABI defines the system call interface to the operating system, and the hardware resources and services available in a system through the user ISA

Key Interfaces

- Application programming interface (API): The API gives a program access to the hardware resources and services available in a system through the user ISA supplemented with high-level language (HLL) library calls

The Operating System as Resource Manager

- The OS is responsible for controlling the use of a computer's resources, such as I/O, main and secondary memory, and processor execution time

The Operating System as Resource Manager

- Functions in the same way as ordinary computer software
- Program, or suite of programs, executed by the processor
- Frequently relinquishes control and must depend on the processor to allow it to regain control

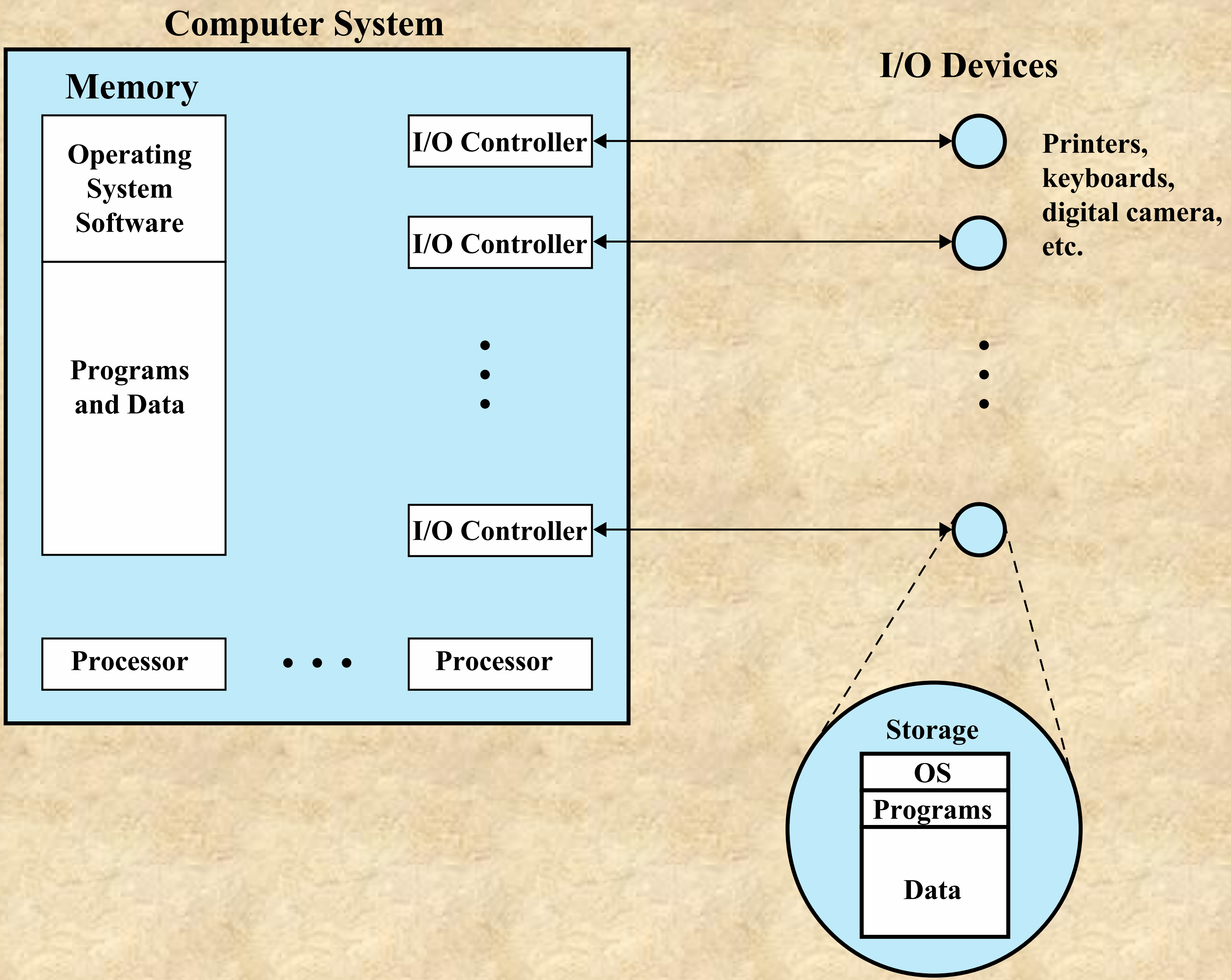


Figure 2.2 The Operating System as Resource Manager

Evolution of Operating Systems

- A major OS will evolve over time for a number of reasons:

Hardware upgrades

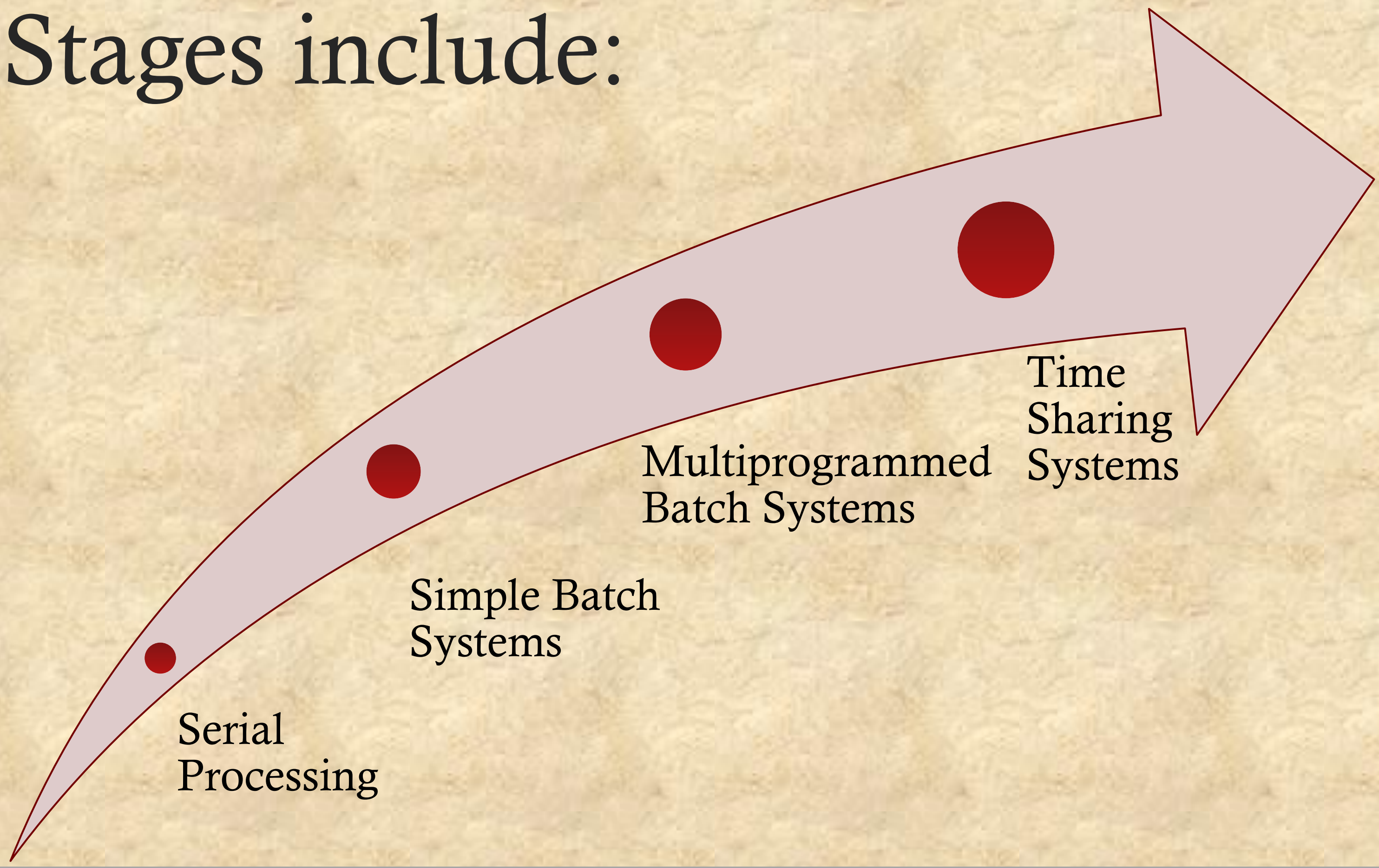
New types of hardware

New services

Fixes

Evolution of Operating Systems

- Stages include:



Serial Processing

Earliest Computers:

- No operating system
  - Programmers interacted directly with the computer hardware
- Computers ran from a console with display lights, toggle switches, some form of input device, and a printer
- Users have access to the computer in "series"

Problems:

- Scheduling:
  - Most installations used a hardcopy sign-up sheet to reserve computer time
    - Time allocations could run short or long, resulting in wasted computer time
- Setup time
  - A considerable amount of time was spent on setting up the program to run

https://www.youtube.com/watch?v=uFQ3sajIdaM

Simple Batch Systems

- Early computers were very expensive
  - Important to maximize processor utilization
- Monitor
  - A piece of software and a type of OS
  - User no longer has direct access to processor
  - Job is submitted to computer operator who batches them together and places them on an input device
  - Program branches back to the monitor when finished

Monitor Point of View

- Monitor controls the sequence of events
- *Resident Monitor* is software always in memory
  - The rest of the monitor consists of utilities and common functions that are loaded as subroutines to the user program at the beginning of any job that requires them
- Monitor reads in a job and passes control to it
- Job returns control to monitor

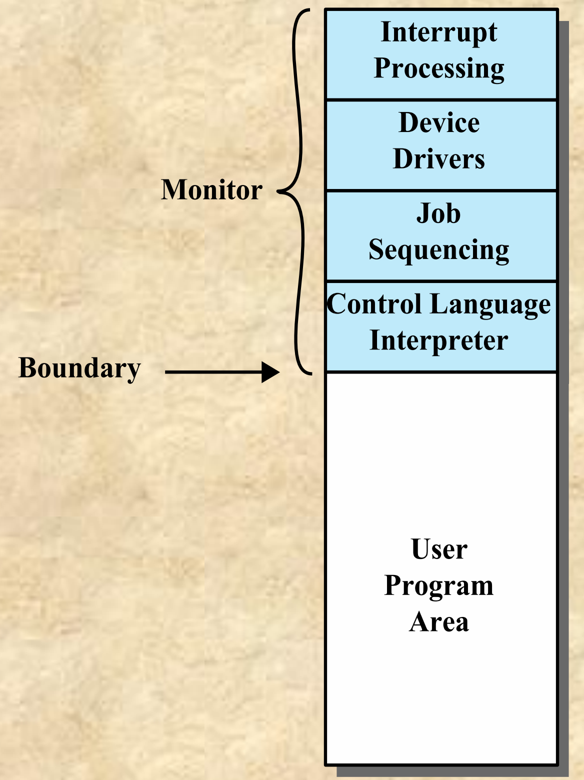


Figure 2.3 Memory Layout for a Resident Monitor

Processor Point of View

- At a certain point, the processor is executing instructions from the portion of memory containing the monitor. These instructions cause the next job to be read into another portion of main memory
- The processor then executes the instructions in the user program until it encounters an ending or error condition
- "*Control is passed to a job*" means processor is fetching and executing instructions in a user program
- "*Control is returned to the monitor*" means that the processor is fetching and executing instructions from the monitor program

Job Control Language (JCL)

- The monitor performs a scheduling function: A batch of jobs is queued up, and jobs are executed as rapidly as possible, with no intervening idle time
- The monitor improves job setup time as well. With each job, instructions are included in a primitive form of **job control language (JCL)**
  - a special type of programming language used to provide instructions to the monitor
    - What complier to use, which files/devices to use, etc.

https://en.wikipedia.org/wiki/Job_Control_Language

Desirable Hardware Features

- **Memory protection**: while the user program is executing, it must not alter the memory area containing the monitor
  - If such an attempt is made, the processor hardware should detect an error and transfer control to the monitor. The monitor would then abort the job, print out an error message, and load in the next job
- **Timer** prevents a single job from monopolizing the system
  - The timer is set at the beginning of each job. If the timer expires, the user program is stopped, and control returns to the monitor

Desirable Hardware Features

- **Privileged instructions**: certain machine level instructions are designated privileged and can be executed only by the monitor. If the processor encounters such an instruction while executing a user program, an error occurs causing control to be transferred to the monitor
- **Interrupts** gives the OS more flexibility in relinquishing control to and regaining control from user programs

Modes of Operation

User Mode

- User program executes in user mode
- Certain areas of memory are protected from user access
- Certain instructions may not be executed

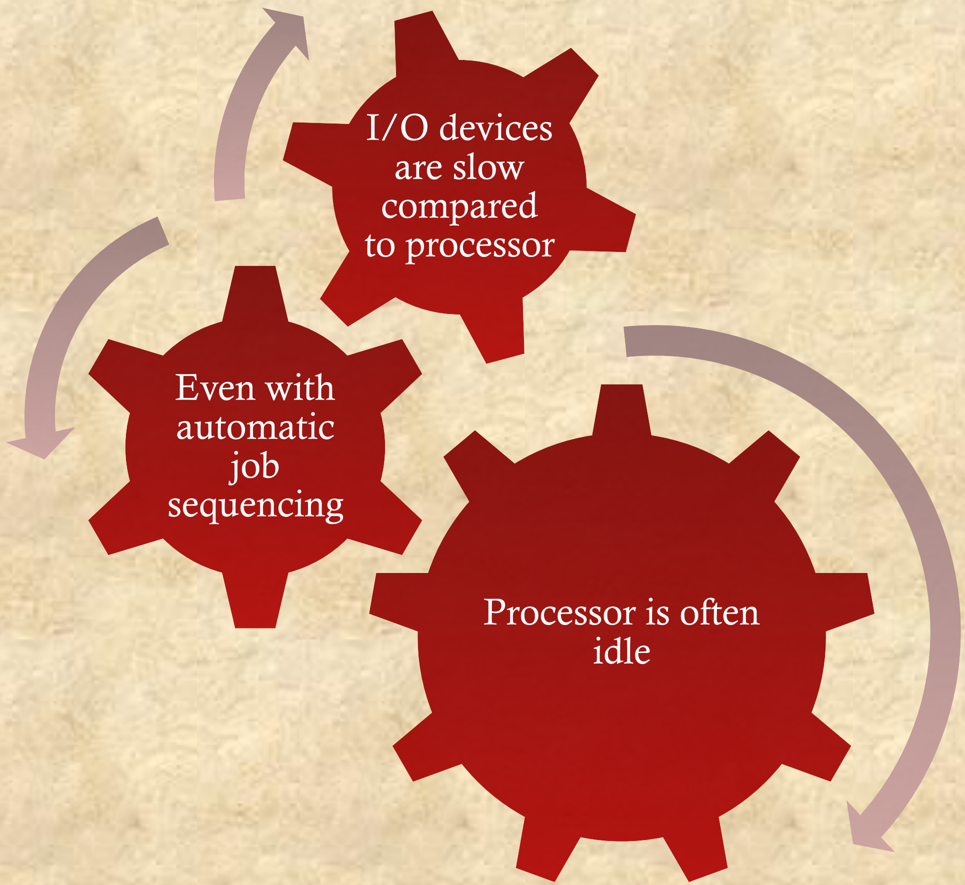
Kernel Mode

- Monitor executes in kernel mode
- Privileged instructions may be executed
- Protected areas of memory may be accessed

Simple Batch System Overhead

- Processor time alternates between execution of user programs and execution of the monitor
- Sacrifices:
  - Some main memory is now given over to the monitor
  - Some processor time is consumed by the monitor
- Despite overhead, the simple batch system improves utilization of the computer

Multiprogrammed Batch Systems

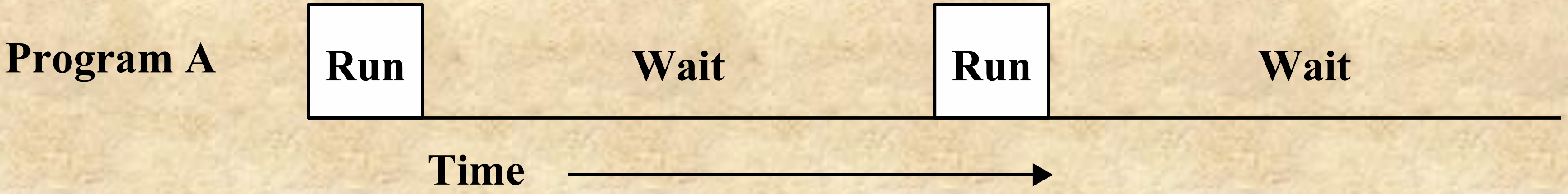


Read one record from file	15 µs
Execute 100 instructions	1 µs
Write one record to file	15 µs
TOTAL	31 µs

$$\text{Percent CPU Utilization} = \frac{1}{31} = 0.032 = 3.2\%$$

Figure 2.4 System Utilization Example

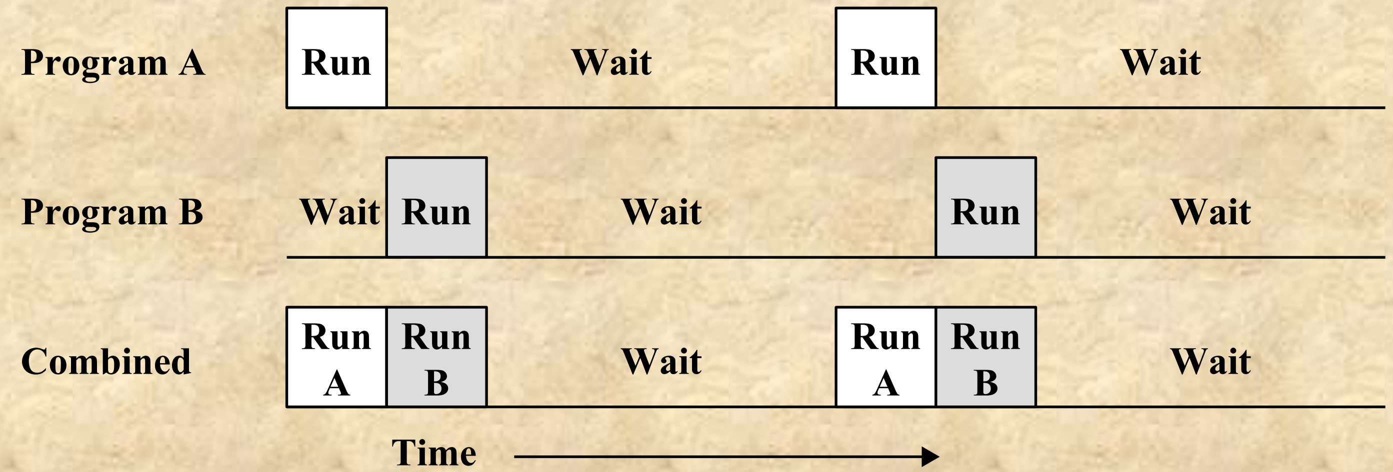
Uniprogramming



(a) Uniprogramming

The processor spends a certain amount of time executing, until it reaches an I/O instruction; it must then wait until that I/O instruction concludes before proceeding

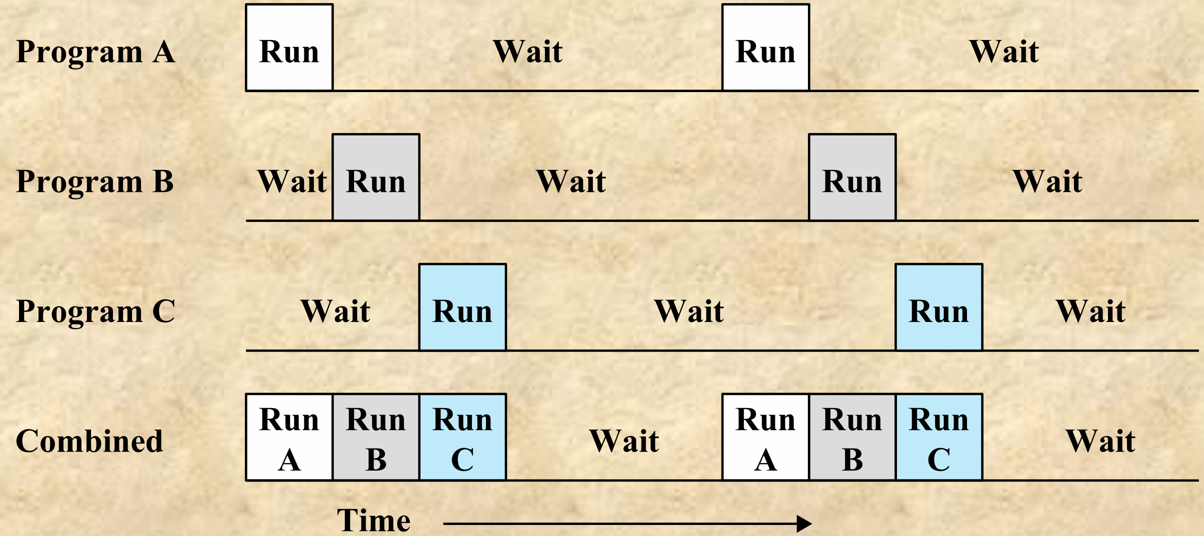
Multiprogramming



(b) Multiprogramming with two programs

- Suppose that there is enough memory to hold the OS (resident monitor) and two user programs
- When one job needs to wait for I/O, the processor can switch to the other job, which is likely not waiting for I/O

Multiprogramming



(c) Multiprogramming with three programs

- Also known as ***multitasking***
- Memory is expanded to hold three, four, or more programs and switch among all of them

Multiprogramming Example

Consider a computer with 250 Mbytes of available memory (not used by the OS), a disk, a terminal, and a printer

	JOB1	JOB2	JOB3
Type of job	Heavy compute	Heavy I/O	Heavy I/O
Duration	5 min	15 min	10 min
Memory required	50 M	100 M	75 M
Need disk?	No	No	Yes
Need terminal?	No	Yes	No
Need printer?	No	No	Yes

Table 2.1 Sample Program Execution Attributes

	Uniprogramming	Multiprogramming
Processor use	20%	40%
Memory use	33%	67%
Disk use	33%	67%
Printer use	33%	67%
Elapsed time	30 min	15 min
Throughput	6 jobs/hr	12 jobs/hr
Mean response time	18 min	10 min

Table 2.2 Effects of Multiprogramming on Resource Utilization

What is finish time of each job with multiprogramming?

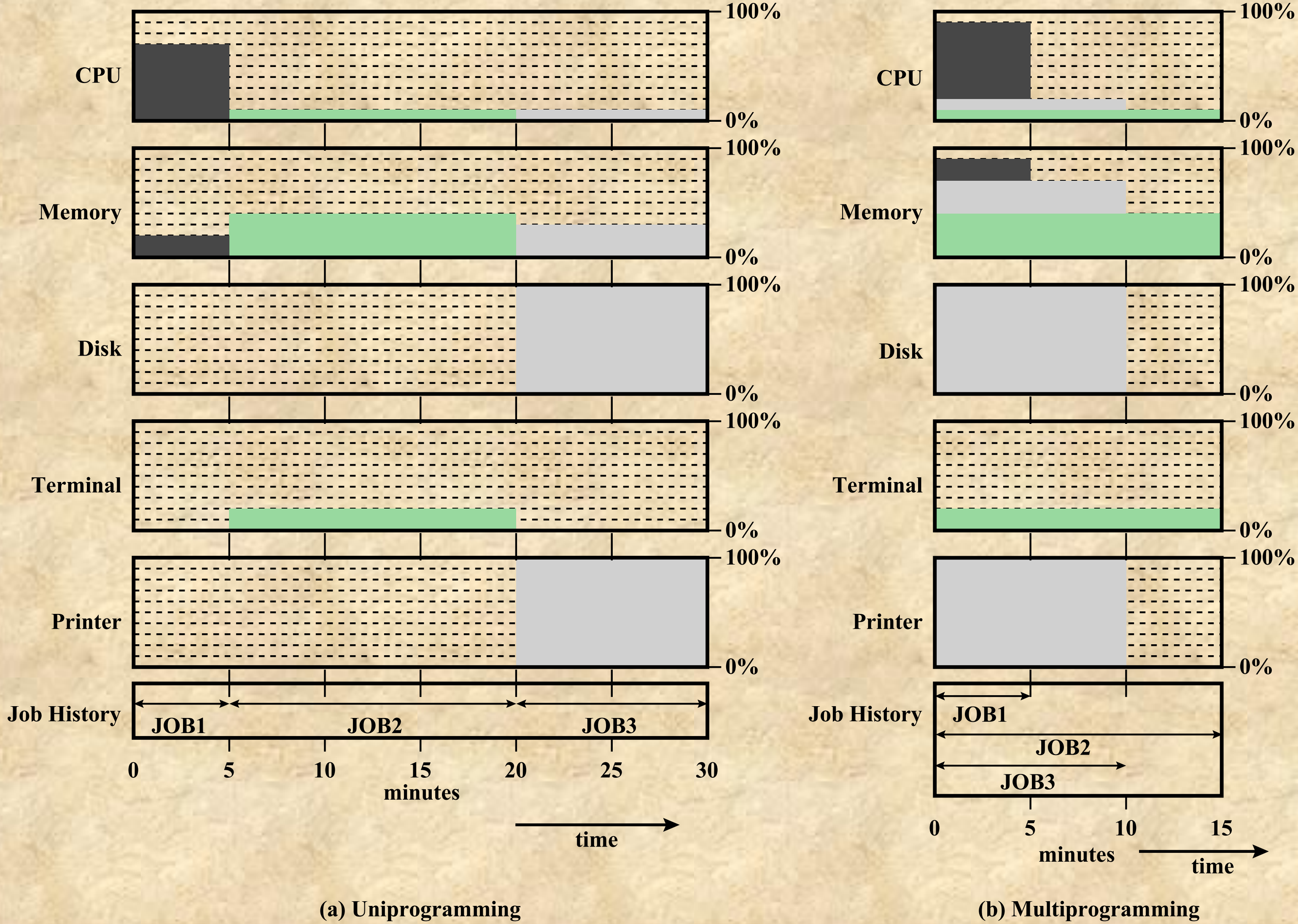


Figure 2.6 Utilization Histograms

Time-Sharing Systems

- Can be used to handle multiple interactive jobs
- Processor time is shared among multiple users
- Multiple users simultaneously access the system through terminals, with the OS interleaving the execution of each user program in a short burst or quantum of computation

	Batch Multiprogramming	Time Sharing
Principal objective	Maximize processor use	Minimize response time
Source of directives to operating system	Job control language commands provided with the job	Commands entered at the terminal

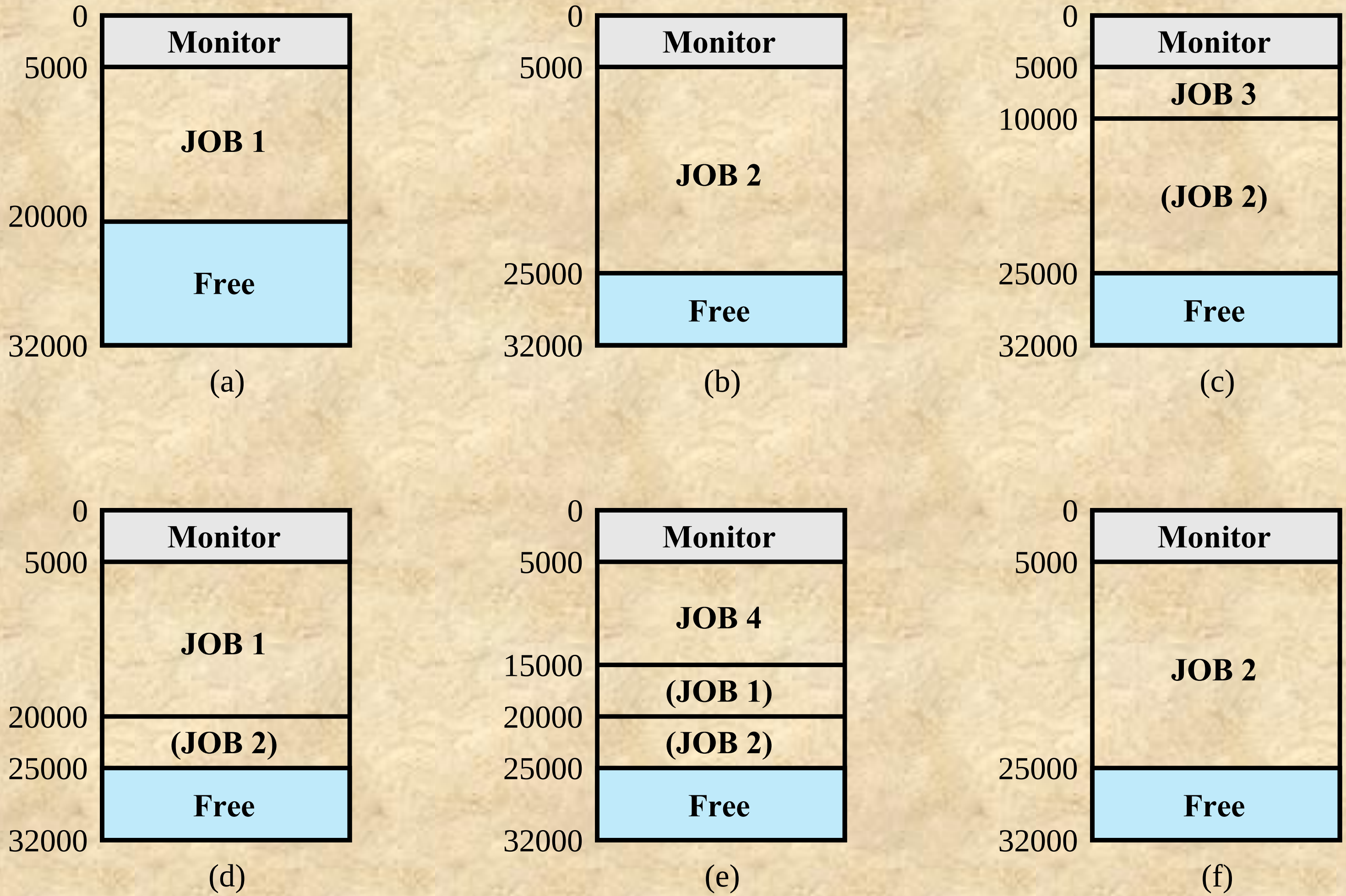
Table 2.3 Batch Multiprogramming versus Time Sharing

Compatible Time-Sharing System (CTSS)

- One of the first time-sharing operating systems
- Developed at MIT by a group known as Project MAC
- The system was first developed for the IBM 709 in 1961
- Ran on a computer with 32,000 36-bit words of main memory, with the resident monitor consuming 5000 of that
- A program was always loaded to start at the location of the 5000th word; this simplified both the monitor and memory management

Compatible Time-Sharing System (CTSS)

- Utilized a technique known as *time slicing*
  - System clock generated interrupts at a rate of approximately one every 0.2 seconds
  - At each clock interrupt the OS regained control and could assign the processor to another user
  - Thus, at regular time intervals the current user would be preempted and another user loaded in
  - To preserve the old user program status for later resumption, the old user programs and data were written out to disk before the new user programs and data were read in
  - Old user program code and data were restored in main memory when that program was next given a turn



JOB1: 15,000
JOB2: 20,000
JOB3: 5000
JOB4: 10,000

Figure 2.7 CTSS Operation