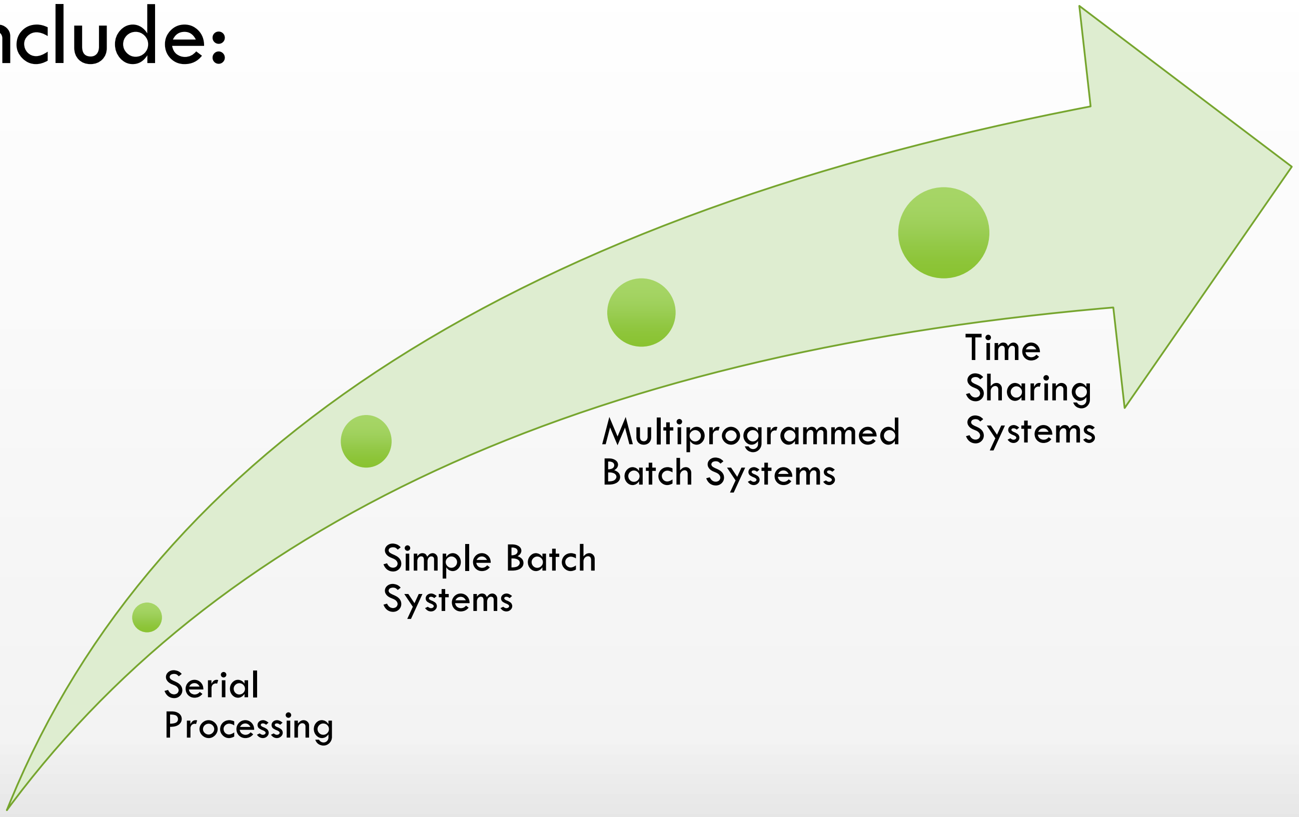
THE EVOLUTION OF OPERATING SYSTEMS

Stages include:



SERIAL PROCESSING

Earliest computers, late 1940s to the mid-1950s

- No Operating System
- The programmer interacted directly with the computer hardware
- These computers were run from a console consisting of display lights, toggle switches, some form of input device, and a printer.
- Programs in machine code were loaded via the input device (e.g., a card reader).
- If an error halted the program, the error condition was indicated by the lights.
- If the program proceeded to a normal completion, the output appeared on the printer.
- This mode of operation could be termed ***serial processing***, as users have access to the computer in series

Problems

- **Scheduling:**

Most installations used a hardcopy sign-up sheet to reserve computer time. A user might sign up for an hour and finish in 45 minutes; this would result in wasted computer processing time. On the other hand, the user might run into problems, not finish in the allotted time, and be forced to stop before resolving the problem.

- **Setup time:**

A considerable amount of time was spent just on setting up the program to run.

A single program, called a **job**, could involve loading the compiler plus the high-level language program (source program) into memory, saving the compiled program (object program), and then loading and linking together the object program and common functions. Each of these steps could involve mounting or dismounting tapes or setting up card decks. If an error occurred, the hapless user typically had to go back to the beginning of the setup sequence. Thus, a considerable amount of time was spent just in setting up the program to run

SIMPLE BATCH SYSTEMS

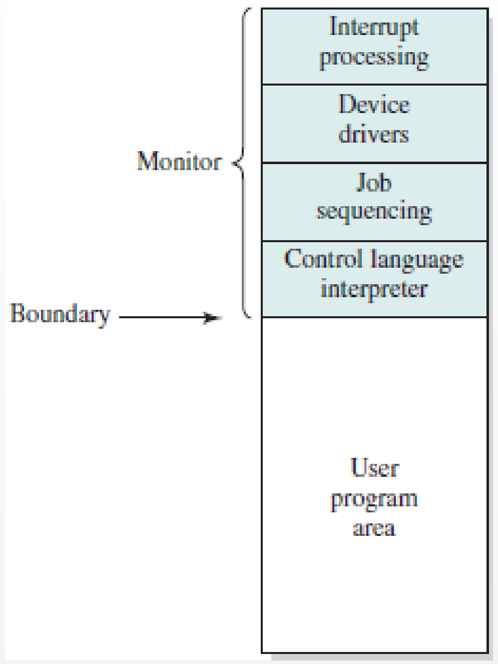
- Early computers were very expensive
  - important to maximize processor utilization
  - the wasted time due to scheduling and setup time was unacceptable.

- The idea behind the simple batch-processing is the use of a piece of software known as the **monitor**.
  - user no longer has direct access to the processor.
  - the user submits the job on cards to a computer operator, who batches the jobs together sequentially and places the entire batch on an input device, for use by the monitor.
  - Each program is constructed to branch back to the monitor when it completes processing
  - monitor automatically begins loading the next program.

SIMPLE BATCH PROCESSING SYSTEMS

MONITOR POINT OF VIEW

- *Resident Monitor* is software always in memory
- Monitor controls the sequence of events
- Monitor reads in job and gives control
- Job returns control to monitor



PROCESSOR POINT OF VIEW

- Processor executes instruction from the memory containing the monitor
- 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

SIMPLE BATCH PROCESSING
ADVANTAGES

- **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.** With each job, instructions are included in a primitive form of **job control language (JCL)**. This is a special type of programming language used to provide instructions to the monitor.
  - A simple example is that of a user submitting a program written in the programming language FORTRAN plus some data to be used by the program.

SIMPLE BATCH PROCESSING

DESIRABLE HARDWARE FEATURES

- The monitor, or batch OS, is simply a computer program. It relies on the ability of the processor to fetch instructions from various portions of main memory to alternately seize and relinquish control. Certain other hardware features are also desirable:

Memory protection for monitor

- while the user program is executing, it must not alter the memory area containing the monitor

Timer

- prevents a job from monopolizing the system

Privileged instructions

- can only be executed by the monitor

Interrupts

- gives OS more flexibility in controlling user programs

SIMPLE BATCH SYSTEMS
MODES OF OPERATION

- Considerations of memory protection and privileged instructions lead to the concept of 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

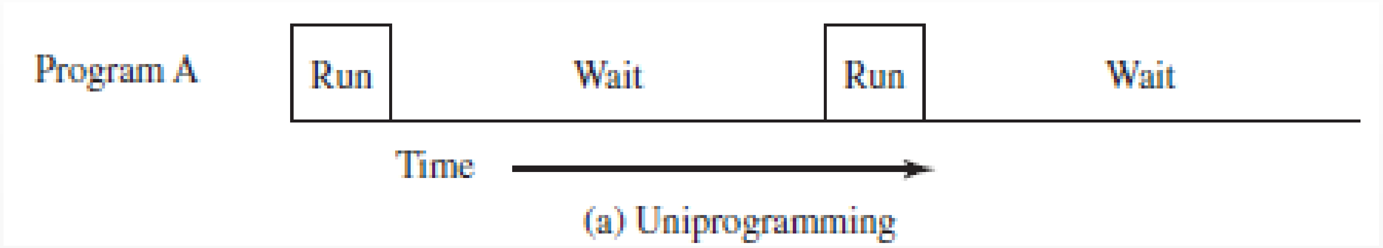
- monitor executes in kernel mode
- privileged instructions may be executed
- protected areas of memory may be accessed

SIMPLE BATCH PROCESSING
OVERHEAD

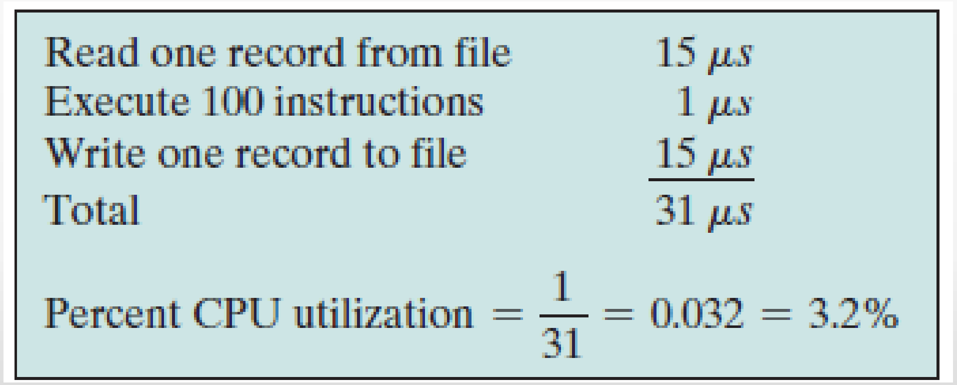
- Processor time alternates between execution of user programs and execution of the monitor
- Sacrifices:
  - some main memory is now given over to the monitor
  - some processor time is consumed by the monitor
- Despite overhead, the simple batch system improves utilization of the computer.

INEFFICIENCY OF SIMPLE BATCH PROCESSING

- Processor is often idle even with automatic job sequencing provided by a simple batch OS. Its because I/O devices are slow compared to processor
- In 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



(a) Uniprogramming



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

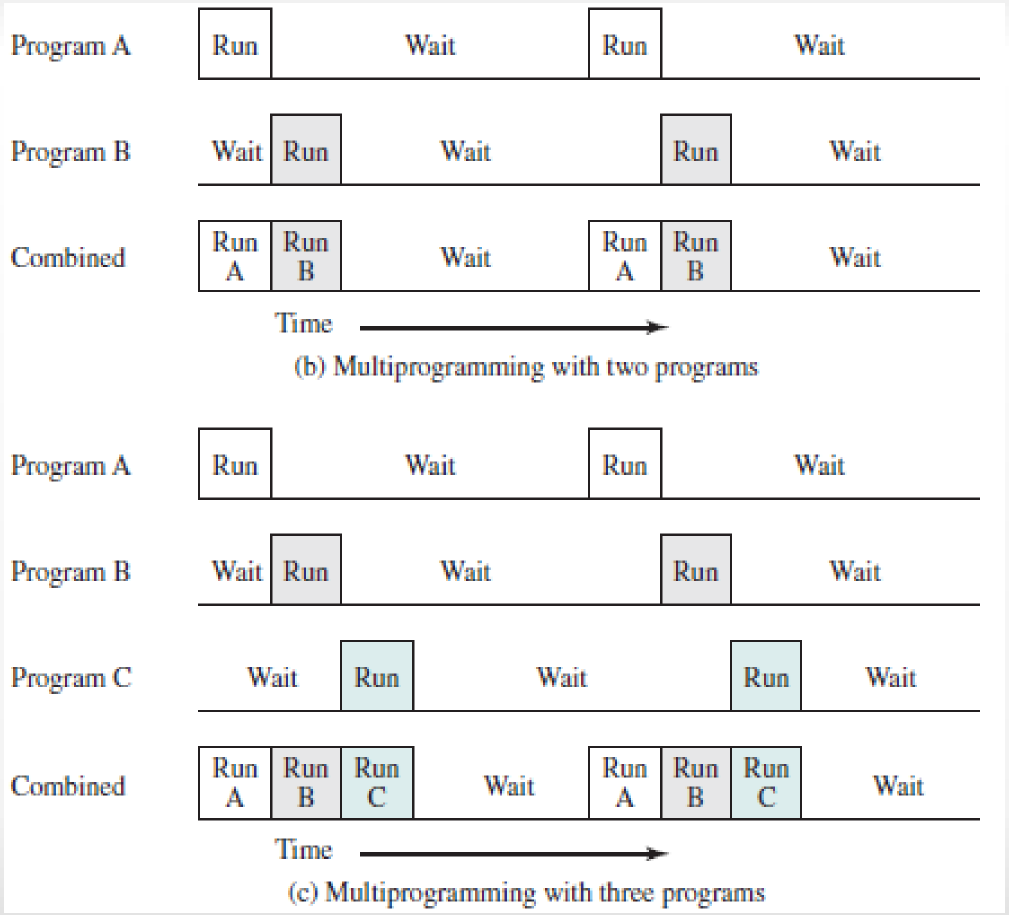
$$\text{Percent CPU utilization} = \frac{1}{31} = 0.032 = 3.2\%$$

MULTIPROGRAMMING BATCH SYSTEMS

- **Multiprogramming** also known as **multitasking**

- Multiprogramming was designed to keep the processor and I/O devices, including storage devices, simultaneously busy to achieve maximum efficiency.

- The key mechanism is this: Memory is expanded to hold the OS (resident monitor) and multiple user programs. When one job needs to wait for I/O, the processor can switch to the other job residing in main memory, which is likely not waiting for I/O.



MULTIPROGRAMMING EXAMPLE

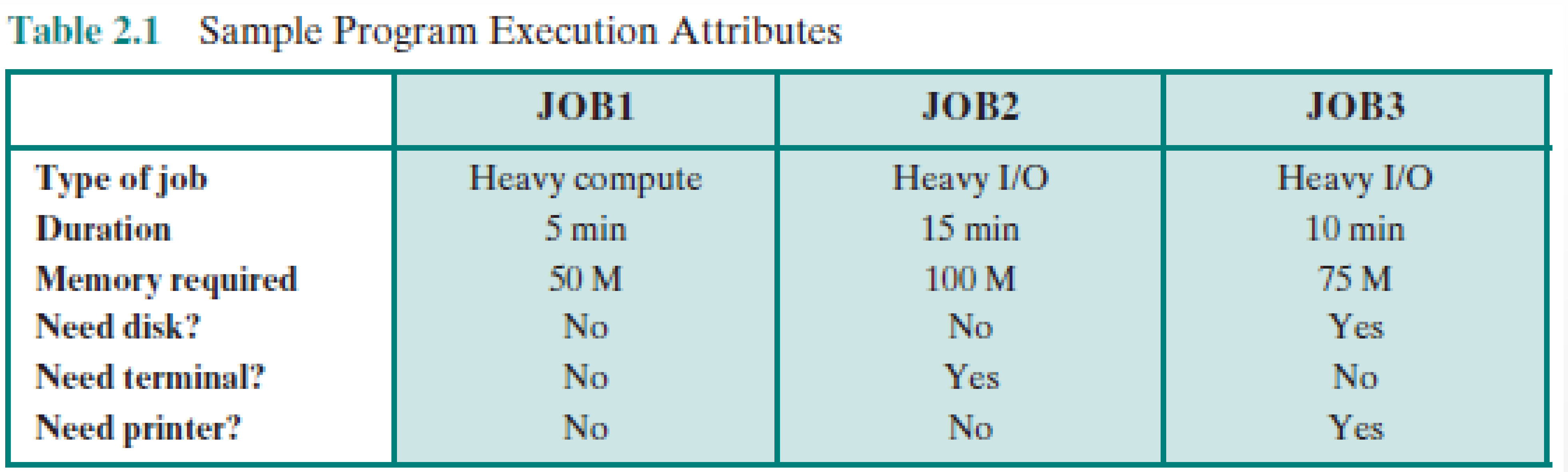


Table 2.1 Sample Program Execution Attributes

	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

MULTIPROGRAMMING EXAMPLE

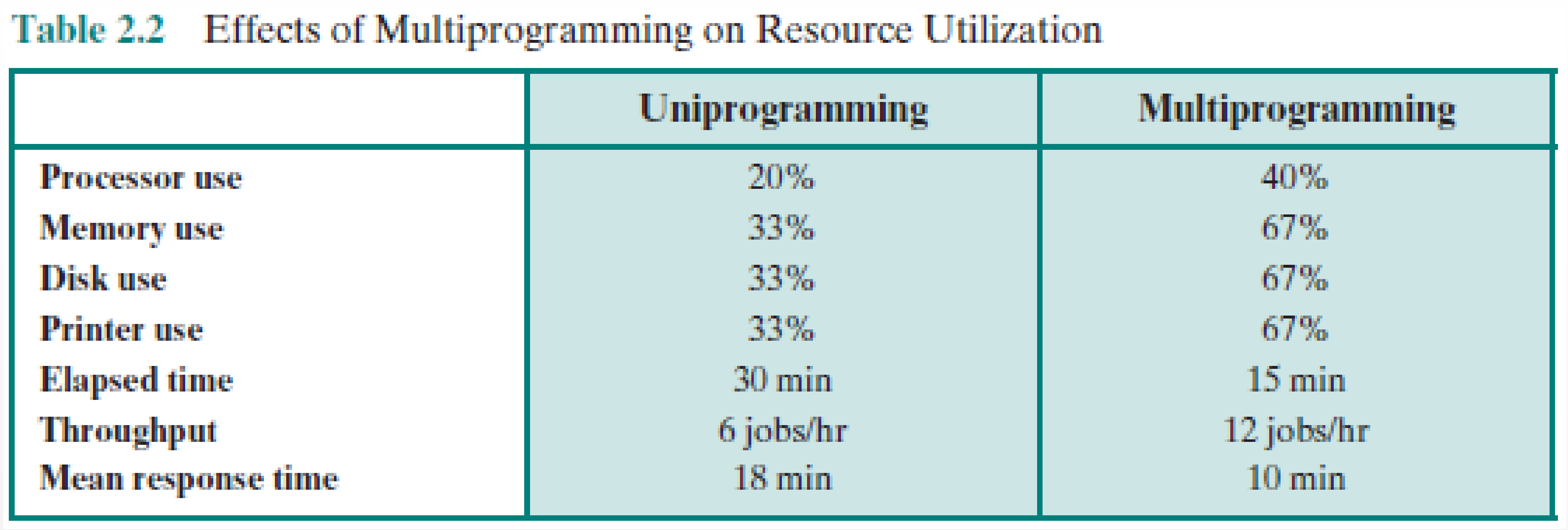
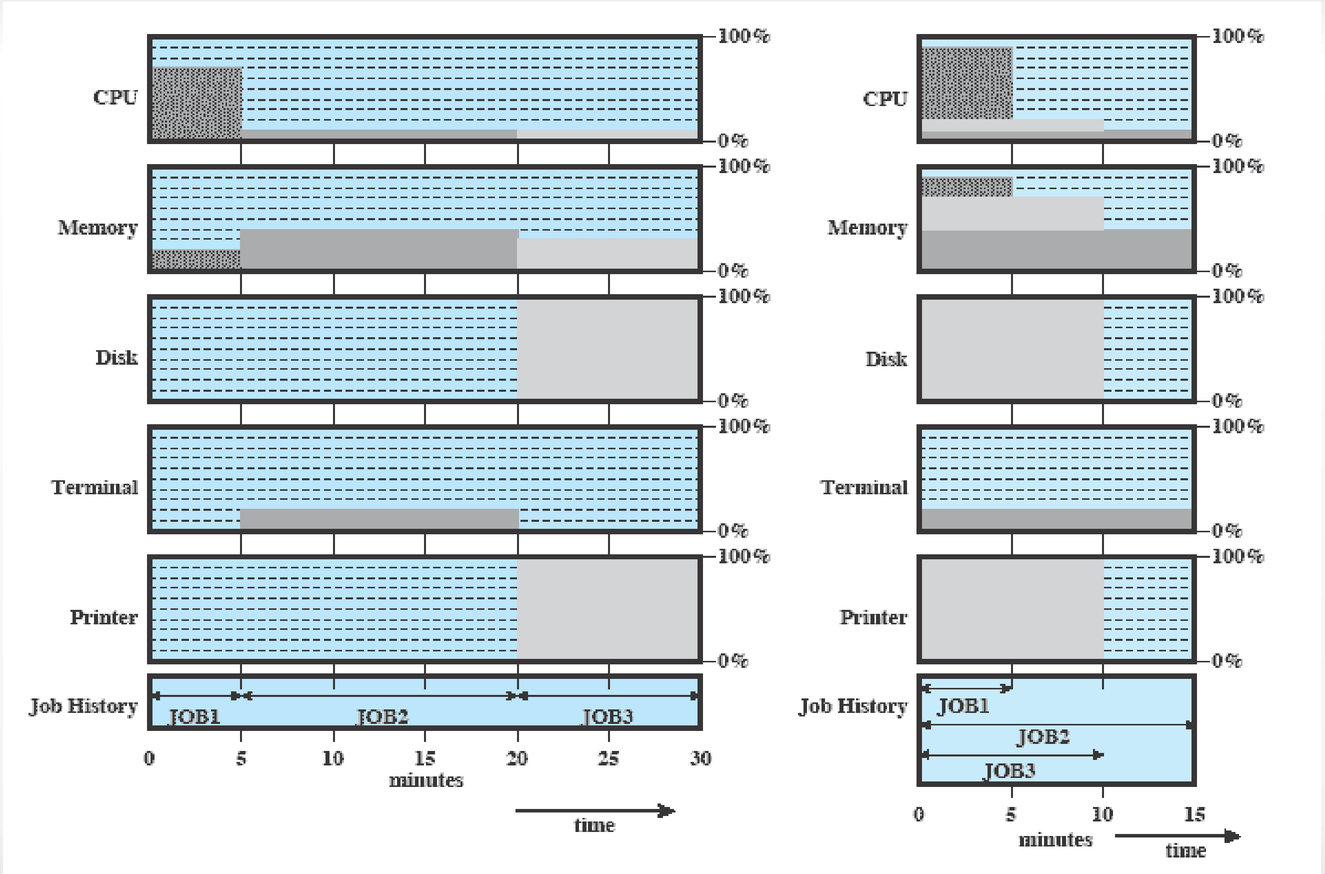


Table 2.2 Effects of Multiprogramming on Resource Utilization

	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

MULTIPROGRAMMING

UTILIZATION HISTOGRAMS



(a) Uniprogramming

(b) Multiprogramming

TIME-SHARING SYSTEMS

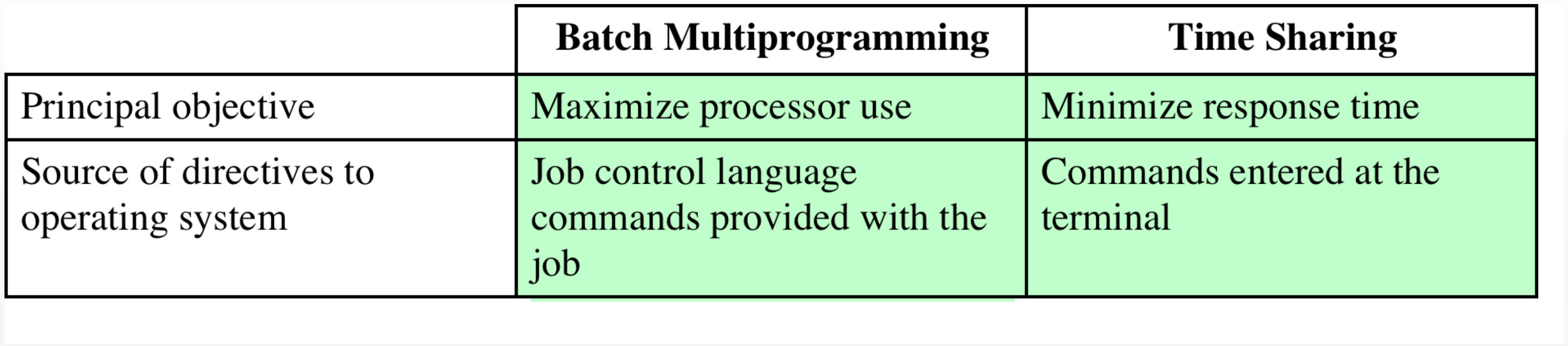
- With the use of multiprogramming, **batch processing** can be quite efficient.
- However, for many jobs, it is desirable to provide a mode in which the user interacts directly with the computer.
- So, another line of development was **general-purpose time sharing**.
- The key design objective is **responsiveness** to the individual user and yet, for cost reasons, be able to support many users simultaneously. These goals are compatible because of the relatively slow reaction time of the user. For example, if a typical user needs an average of 2 seconds of processing time per minute, then close to 30 such users should be able to share the same system without noticeable interference.

- A time-sharing system 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

REAL-TIME TRANSACTION PROCESSING SYSTEMS

- An other important line of development has been real-time transaction processing systems.
- In transaction processing, an interactive mode is essential, a number of users are entering queries or updates against a database. An example is an airline reservation system.
- The key difference between the transaction processing system and the time-sharing system is that the former is limited to one or a few applications, whereas users of a time-sharing system can engage in program development, job execution, and the use of various applications. In both cases, system response time is paramount.

BATCH MULTIPROGRAMMING VS. TIME SHARING



	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

COMPATIBLE TIME-SHARING SYSTEMS

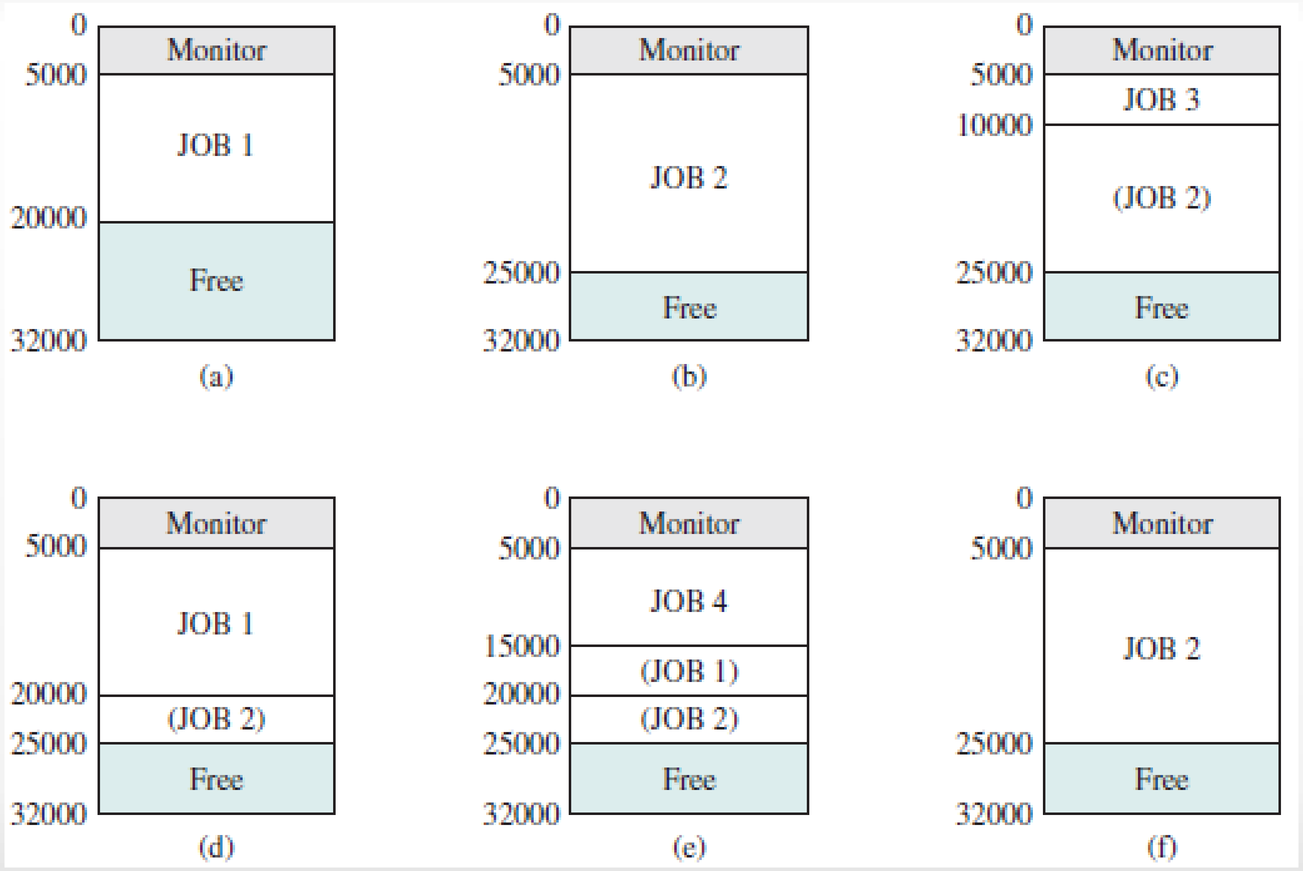
CTTS

- Compatible Time-Sharing System (CTTS)
- One of the first time-sharing operating systems
- Developed at MIT by a group known as Project MAC
- Ran on a computer with 32,000 36-bit words of main memory, with the resident monitor consuming 5000 of that
- To simplify both the monitor and memory management a program was always loaded to start at the location of the 5000th word

TIME SLICING

- System clock generates interrupts at a rate of approximately one every 0.2 seconds
- At each interrupt OS regained control and could assign processor to another user
- At regular time intervals the current user would be preempted and another user loaded in
- Old user programs and data were written out to disk
- Old user program code and data were restored in main memory when that program was next given a turn

CTTS EXAMPLE



MAJOR ACHIEVEMENTS

There have been four major theoretical advances in the development of operating systems:

- **Processes**
- **Memory management**
- **Information protection and security**
- **Scheduling and resource management**

Taken together, these four areas span many of the key design and implementation issues of modern operating systems.

THE PROCESS

- Central to the design of operating systems is the concept of ***process***.
- It is more general term than *job*. Some of the definitions have been given for the term ***process*** *are:*
    - ***A program in execution***
    - ***An instance of a program running on a computer***
    - ***The entity that can be assigned to and executed on a processor***
    - ***A unit of activity characterized by a single sequential thread of execution, a current state, and an associated set of system resources***

- The concept of process should become clearer as we proceed.