Chapter 1: Introduction (Part 2)

Chapter 1: Introduction

- What Operating Systems Do
- Computer-System Organization
  - System Call
  - Interrupts
  - Interrupt handling
  - DMA
  - Storage Device Hierarchy
- Computer-System Architecture
- Operating-System Structure (Evolution of OS)
- Operating-System Operations
- Process Management
- Memory Management
- Storage Management

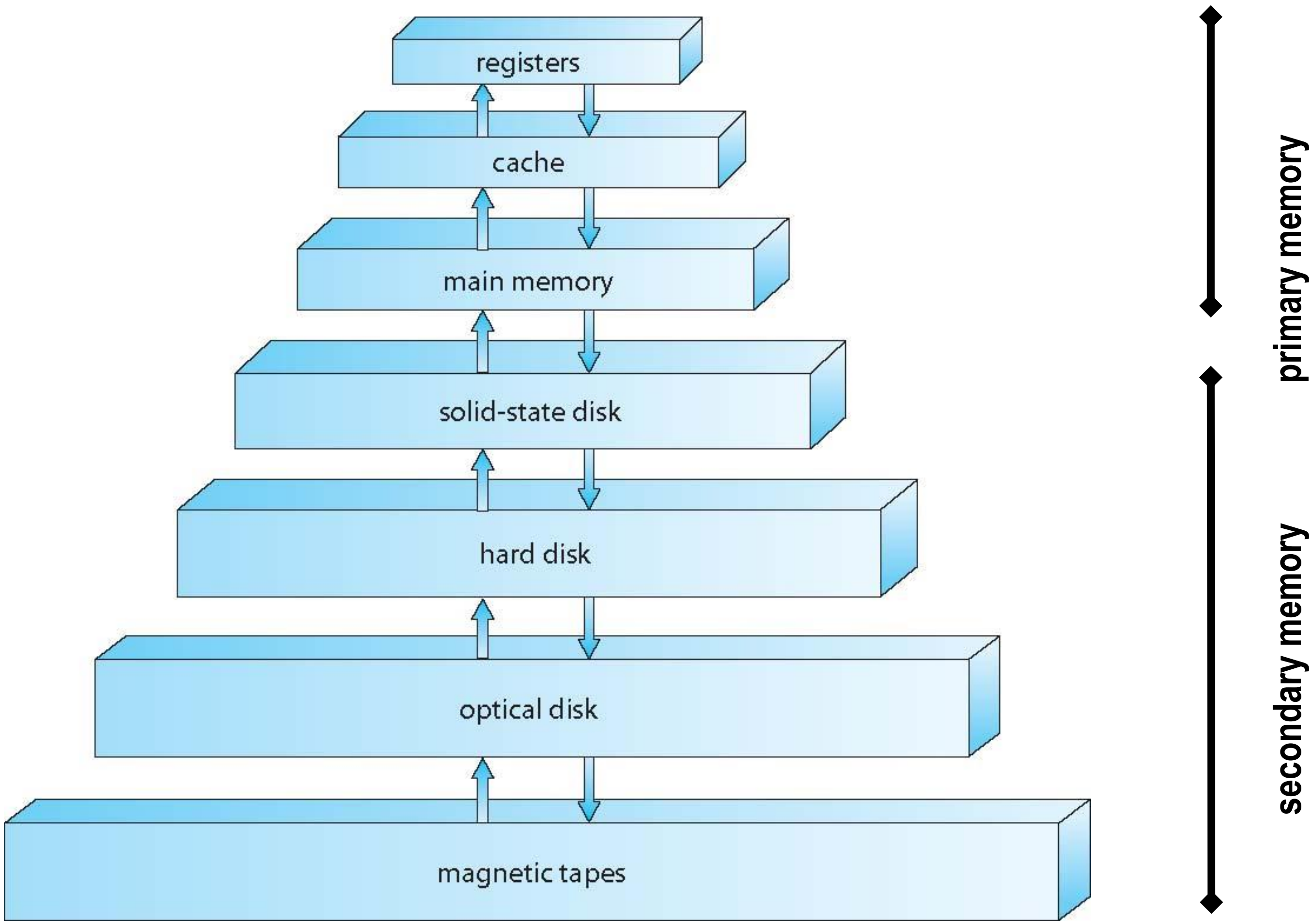
Storage Definitions and Notation Review

The basic unit of computer storage is the **bit**. A bit can contain one of two values, 0 and 1. All other storage in a computer is based on collections of bits. Given enough bits, it is amazing how many things a computer can represent: numbers, letters, images, movies, sounds, documents, and programs, to name a few. A **byte** is 8 bits, and on most computers it is the smallest convenient chunk of storage. For example, most computers don't have an instruction to move a bit but do have one to move a byte. A less common term is **word**, which is a given computer architecture's native unit of data. A word is made up of one or more bytes. For example, a computer that has 64-bit registers and 64-bit memory addressing typically has 64-bit (8-byte) words. A computer executes many operations in its native word size rather than a byte at a time.

Computer storage, along with most computer throughput, is generally measured and manipulated in bytes and collections of bytes.
A **kilobyte**, or **KB**, is 1,024 bytes
a **megabyte**, or **MB**, is $1{,}024^2$ bytes
a **gigabyte**, or **GB**, is $1{,}024^3$ bytes
a **terabyte**, or **TB**, is $1{,}024^4$ bytes
a **petabyte**, or **PB**, is $1{,}024^5$ bytes

Computer manufacturers often round off these numbers and say that a megabyte is 1 million bytes and a gigabyte is 1 billion bytes. Networking measurements are an exception to this general rule; they are given in bits (because networks move data a bit at a time).

Storage-Device Hierarchy



Storage structure

- Memory is organized in a hierarchy

tradeoffs: speed vs cost vs volatility

- **Primary memory**
  - fast, expensive, volatile
  - data is stored in electronic circuitry
  - Example: Main memory – only large storage media that the CPU can access directly
- **Secondary memory**
  - slow, cheap, permanent
  - data is stored magnetically
  - can store massive amounts of inactive data, must be copied to primary memory to be accessed
  - **Example:** Hard disks, solid state disks
  - Solid-state disks – faster than hard disks, nonvolatile
  - Various technologies
  - Becoming more popular

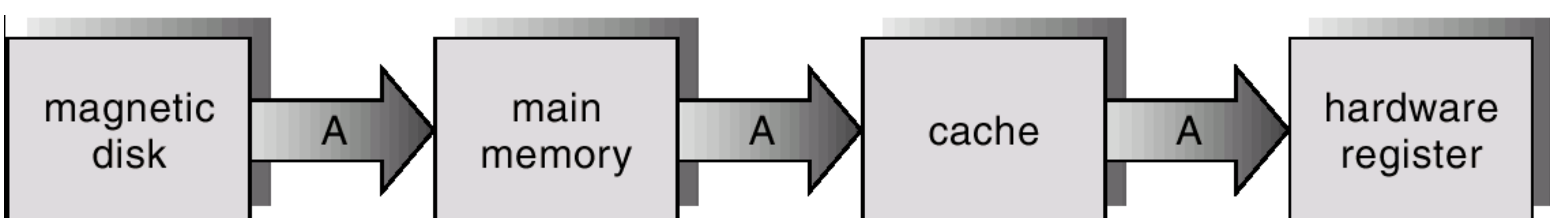
Primary memory: RAM vs. cache

- RAM and cache both store data in (volatile) electronic circuitry
  - cache uses faster, more expensive technology
  - Level-1 cache is stored directly on the CPU chip (runs at speeds comparable to processor speed, ~10x RAM access speed)
  - Level-2 cache is stored on nearby chip, ~2x RAM access speed common

common approach:

- when data from RAM is needed by CPU, first copy into cache
- CPU then accesses cache directly
- cache retains recently used (most active?) data, fast access if needed again

note: cache is to RAM as RAM is to secondary memory

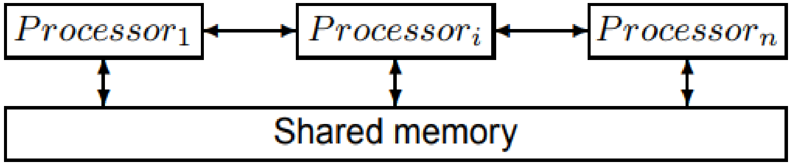


Chapter 1: Introduction

- What Operating Systems Do
- Computer-System Organization
  - System Call
  - Interrupts
  - Interrupt handling
  - DMA
  - Storage Device Hierarchy
- **Computer-System Architecture**
- Operating-System Structure (Evolution of OS)
- Operating-System Operations
- Process Management
- Memory Management
- Storage Management

Computer-System Architecture

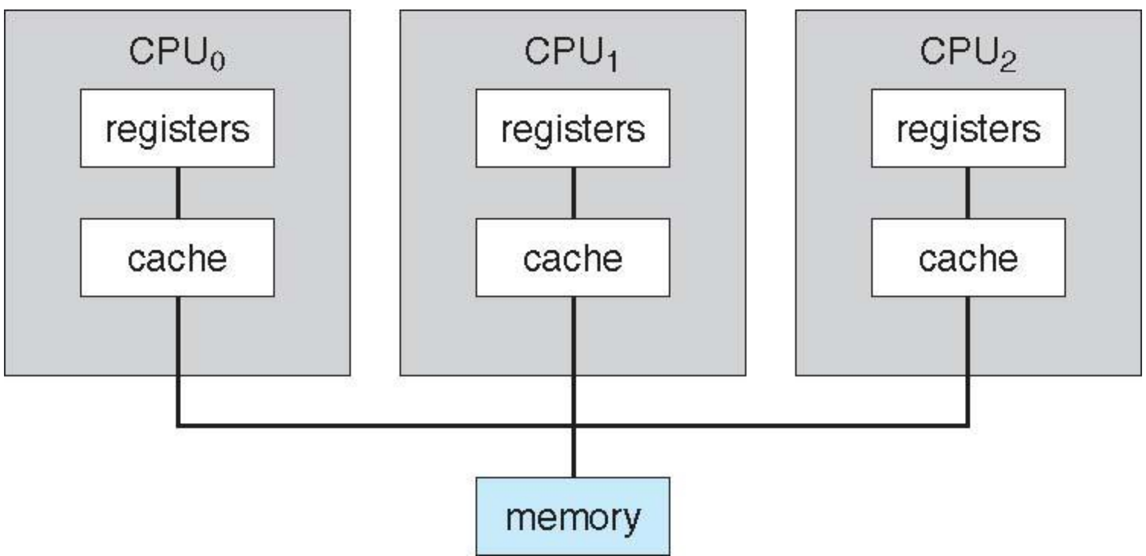
- Most systems use a single general-purpose processor
  - Most systems have special-purpose processors as well
- **Multiprocessors** systems growing in use and importance
  - Also known as **parallel systems**, **tightly-coupled systems** (i.e., All processors share the same memory)



  - Advantages include:
    1. **Increased throughput:** expect to get more work done in less time
    2. **Economy of scale:** share peripherals, mass storage, and power supplies
    3. **Increased reliability** – graceful degradation or fault tolerance
  - Two types:
    1. **Asymmetric Multiprocessing** – Boss-Worker-Each Processor is assigned a specific task
    2. **Symmetric Multiprocessing (SMP)**– - all processors are peers- each processor performs all tasks

Symmetric Multiprocessing Architecture

- A stand-alone computer system with the following characteristics:
  - two or more similar processors of comparable capability
  - processors share the same main memory and are interconnected by a bus or other internal connection scheme
  - processors share access to I/O devices
  - all processors can perform the same functions
  - the system is controlled by an integrated operating system that provides interaction between processors and their programs at the job, task, file, and data element levels



A Multicore Design

- Multiple computing cores (Processing Units) on a single chip (Single Physical Package)
- More efficient than multiple chips with single cores
  - On chip communication is faster than between chip communication

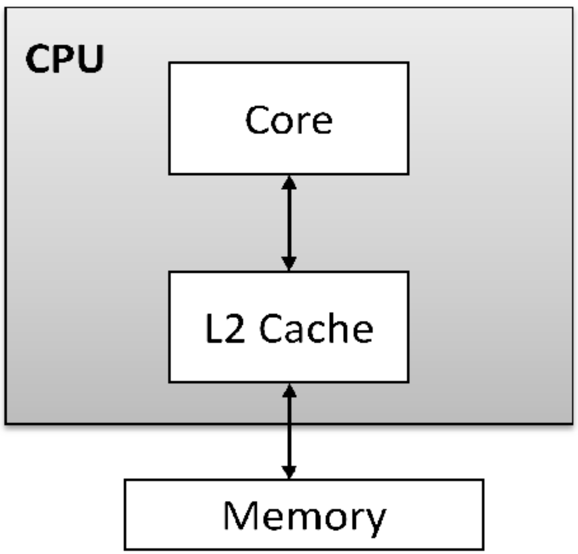


Figure 1:
Single-core Architecture

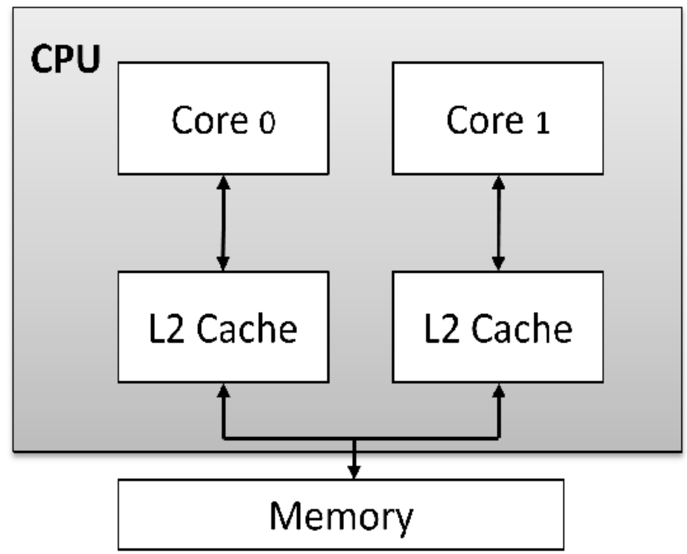
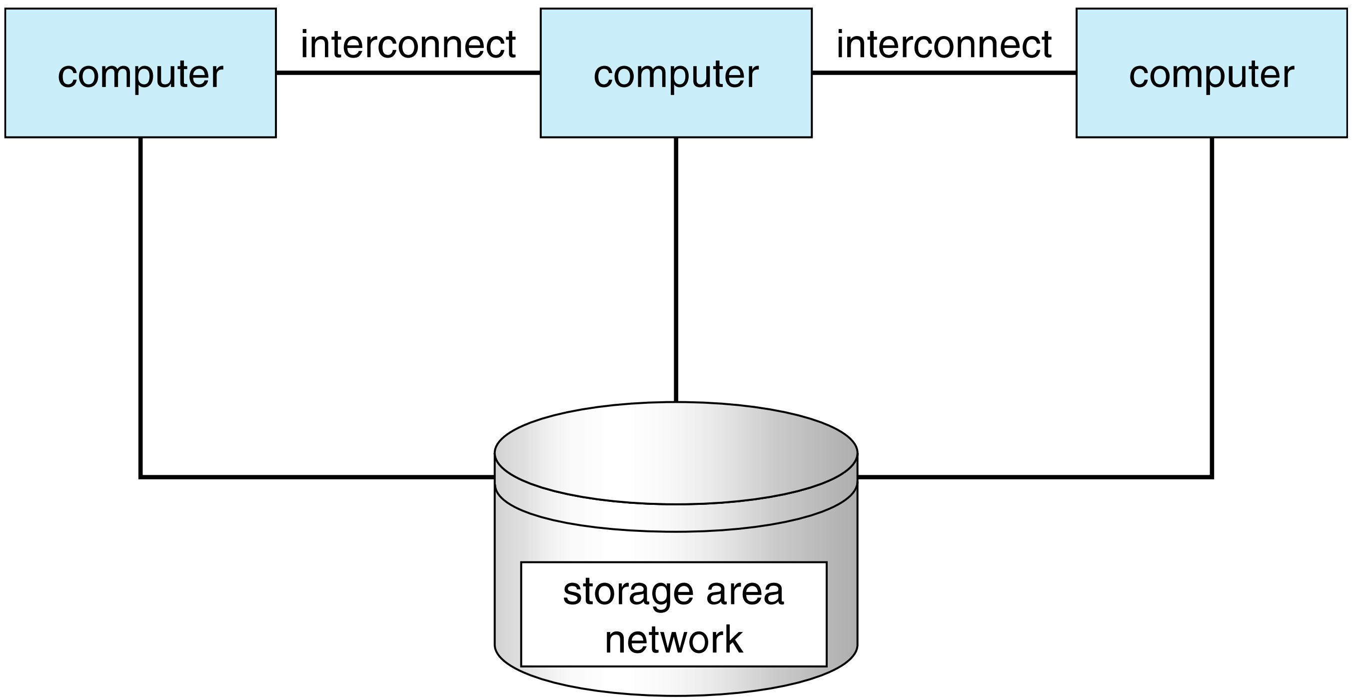


Figure 2:
Multi-core Architecture

Clustered Systems



Clustered Systems

- Like multiprocessor systems, but multiple systems working together
- Usually sharing storage via a **storage-area network (SAN)**
- Provides a **high-availability** service which survives failures
  - **Asymmetric clustering** has one machine in *hot-standby mode*
    - Hot-standby host monitor the active server and takes over if the later fails
  - **Symmetric clustering** has multiple nodes running applications, monitoring each other
- Some clusters are for **high-performance computing (HPC)**
  - Applications must be written to use **parallelization**
- Hosts can perform conflicting operations which can be prevented by locking and access control mechanisms, also called **distributed lock manager** (**DLM**).

Chapter 1: Introduction

- What Operating Systems Do
- Computer-System Organization
- Computer-System Architecture
- **Operating-System Structure (Evolution of OS)**
- Operating-System Operations
- Process Management
- Memory Management
- Storage Management

Evolution of an Operating Systems

- Must adapt to hardware upgrades and new types of hardware.

  Examples:

  - For example, Introduction of paging hardware

- Must offer new services, e.g., internet support.

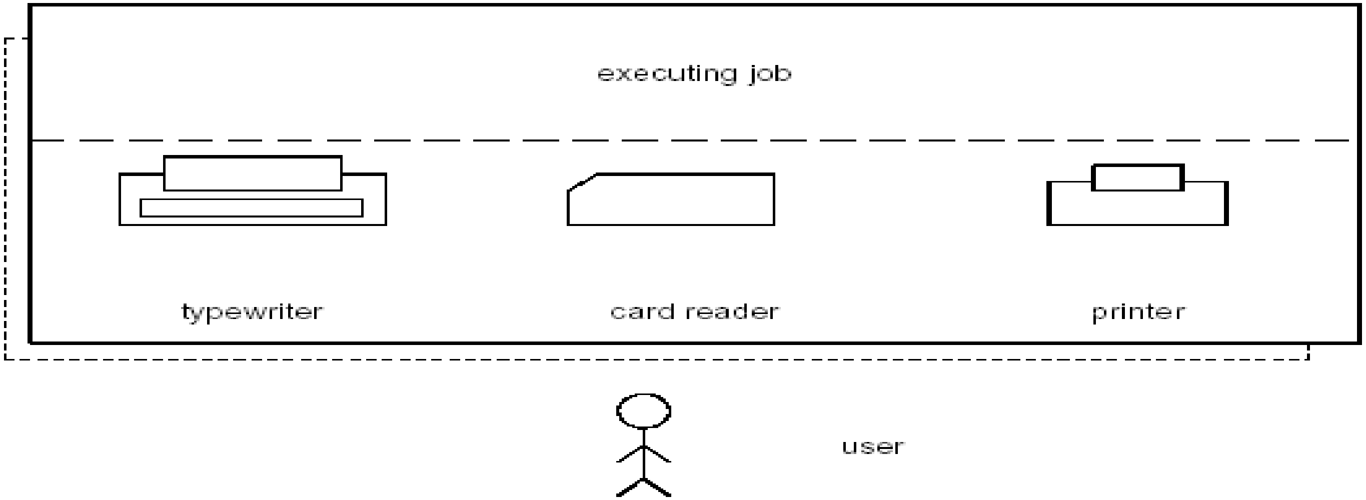
- The need to change the OS on regular basis place requirements on it’s design:

  - modular construction with clean interfaces.

  - object oriented methodology.

Early Systems: Serial Processing (1940-1955)

- Structure
  - Single user system.
  - Programmer/User as operator
  - Large machines run from console (display lights, toggle switches)
  - Input received from Paper Tape or Punched cards.
  - No OS



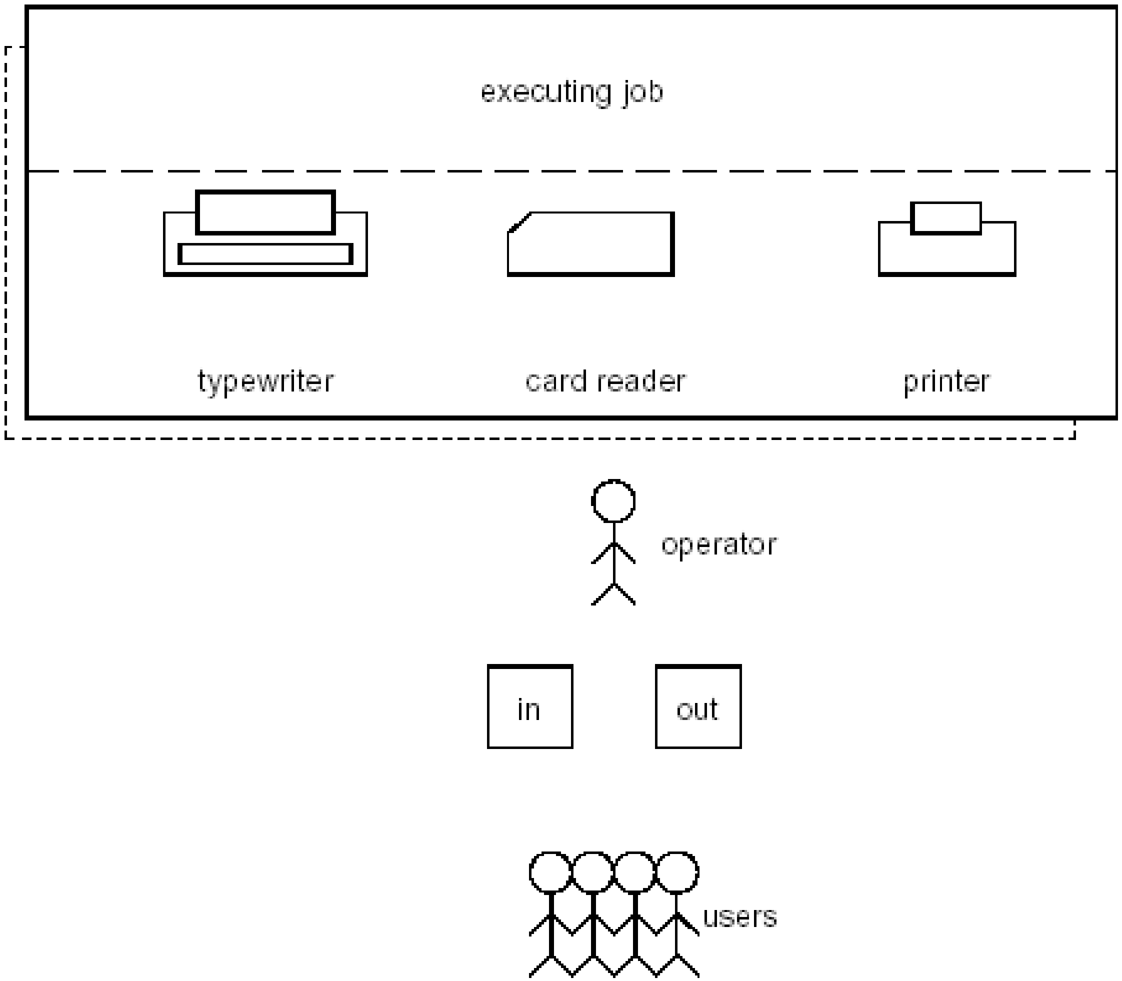
Characteristics of Early Systems

- Early software: Assemblers, Libraries of common subroutines (I/O), Device Drivers, Compilers, Linkers.
- Need significant amount of setup time.
- Extremely slow I/O devices.
- Very low CPU utilization.

Simple Batch Systems

- Use of high-level languages, magnetic tapes (instead of paper tapes).
- Jobs are batched together by type of languages.
- An operator was hired to perform the repetitive tasks of loading jobs, starting the computer, and collecting the output (Operator-driven Shop).
  - Overall process became a little fast
- It was not feasible for users to inspect memory or patch programs directly.

Operator-driven Shop



Operation of Simple Batch Systems

- The user submits a job (written on cards or tape) to a computer operator.
- The computer operator place a batch of several jobs on an input device.
- A special program, the monitor, manages the execution of each program in the batch.
- Monitor utilities are loaded when needed.
- “Resident monitor” is always in main memory and available for execution.
    - first OS
    - initial control is in monitor.
    - loads next program and transfers control to it.
    - when job completes, the control transfers back to monitor.
    - Automatically transfers control from one job to another, no idle time between programs.

Idea of Simple Batch Systems

- Reduce setup time by batching similar jobs.
- Alternate execution between user program and the monitor program.
- Rely on available hardware to effectively alternate execution from various parts of memory.
- Use Automatic Job Sequencing – automatically transfer control from one job when it finishes to another one.

Simple Batch Systems: Job Control Language (JCL)

- JCL is the language that provides instructions to the monitor:
  - what compiler to use
  - what data to use

- Example of job format: ------->>
- **'$'** at the beginning denotes the job control instruction
  - `$FTN` loads the compiler and transfers control to it.
  - `$LOAD` loads the object code (in place of compiler).
  - `$RUN` transfers control to user program.

$JOB
$FTN
. . .
FORTRAN program
. . .
$LOAD
$RUN
. . .
Data
. . .
$END

Simple Batch Systems (1)

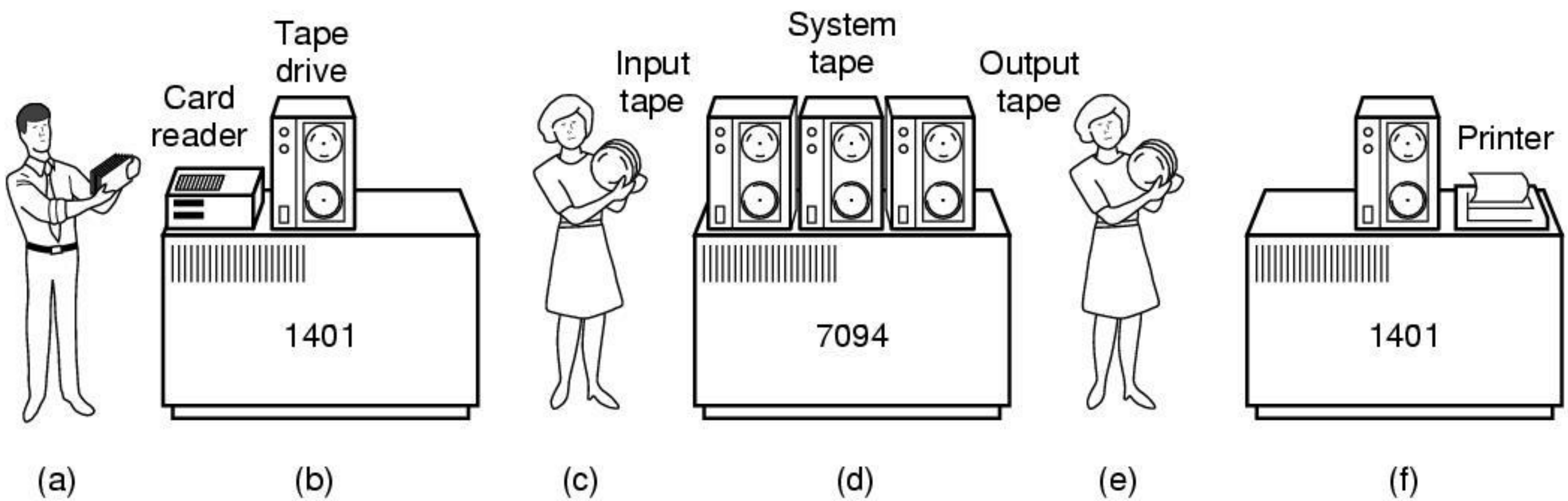


Figure 1-3. An early batch system.
(a) Programmers bring cards to 1401.
(b)1401 reads batch of jobs onto tape.

Simple Batch Systems (2)

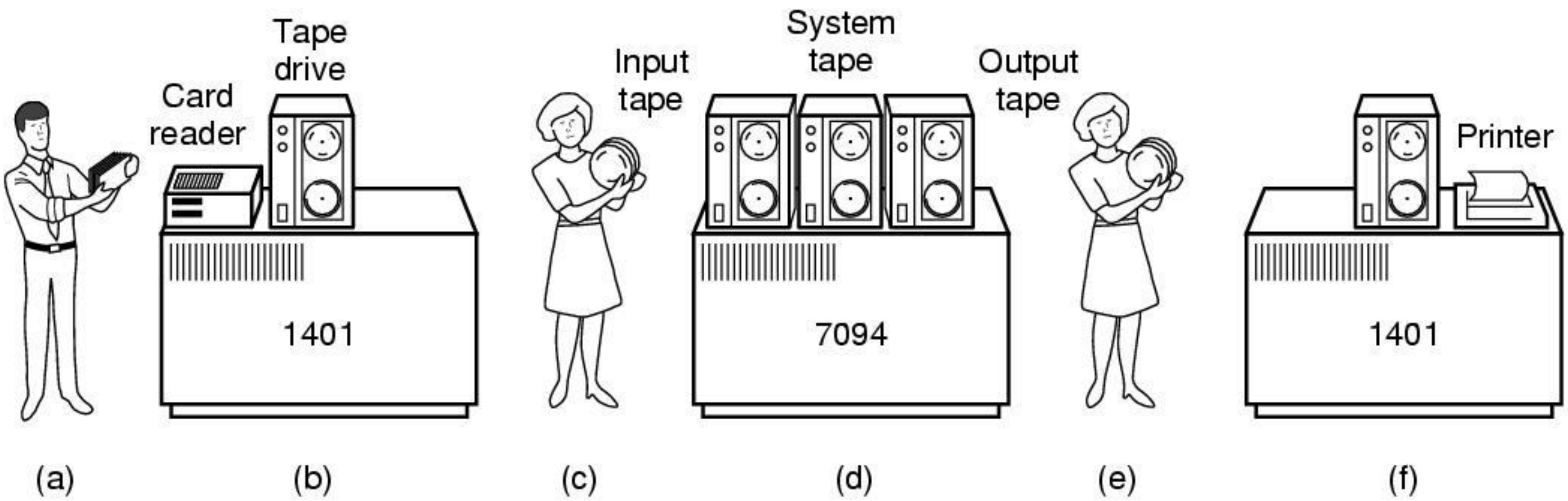
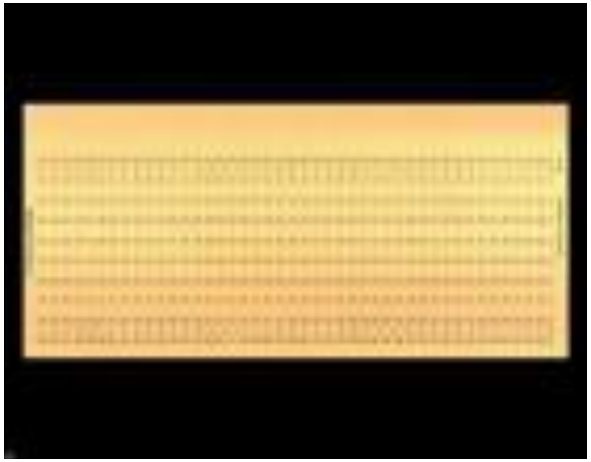
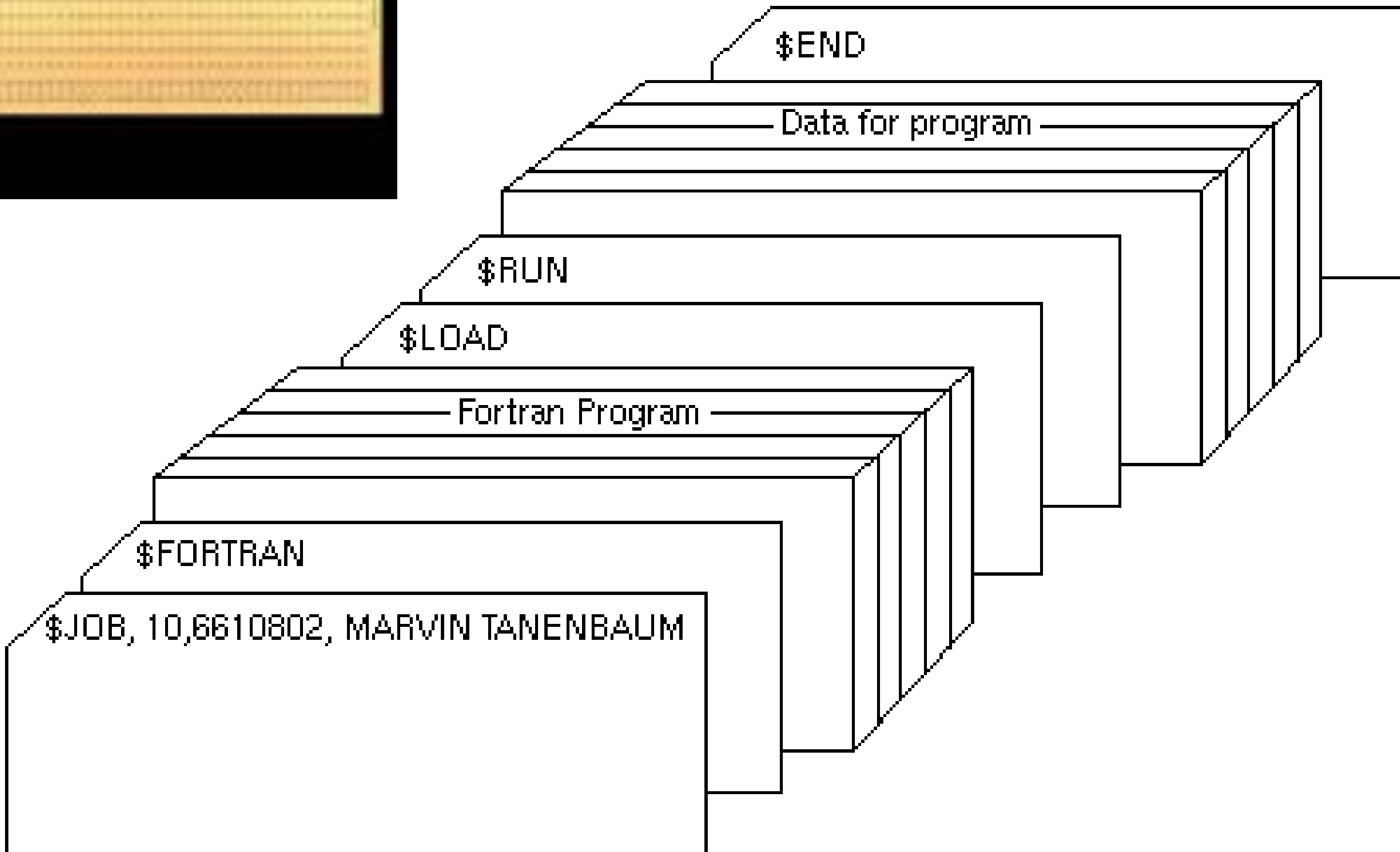


Figure 1-3. (c) Operator carries input tape to 7094. (d) 7094 does computing. (e) Operator carries output tape to 1401. (f) 1401 prints output.

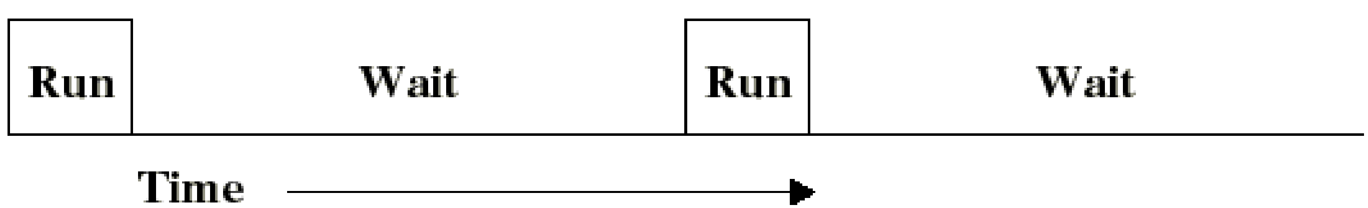
Example card deck of a Job





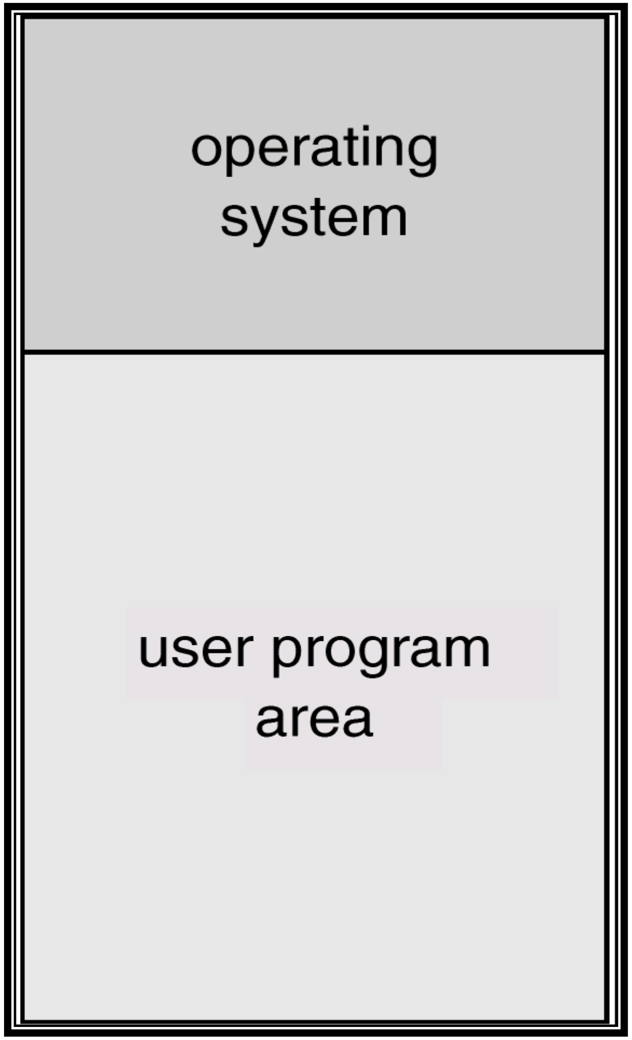
We assumed Uniprogramming until now

- I/O operations are exceedingly slow (compared to instruction execution).
- A program containing even a very small number of I/O operations, will spend most of its time waiting for them.
- Hence: poor CPU usage when only one program is present in memory.



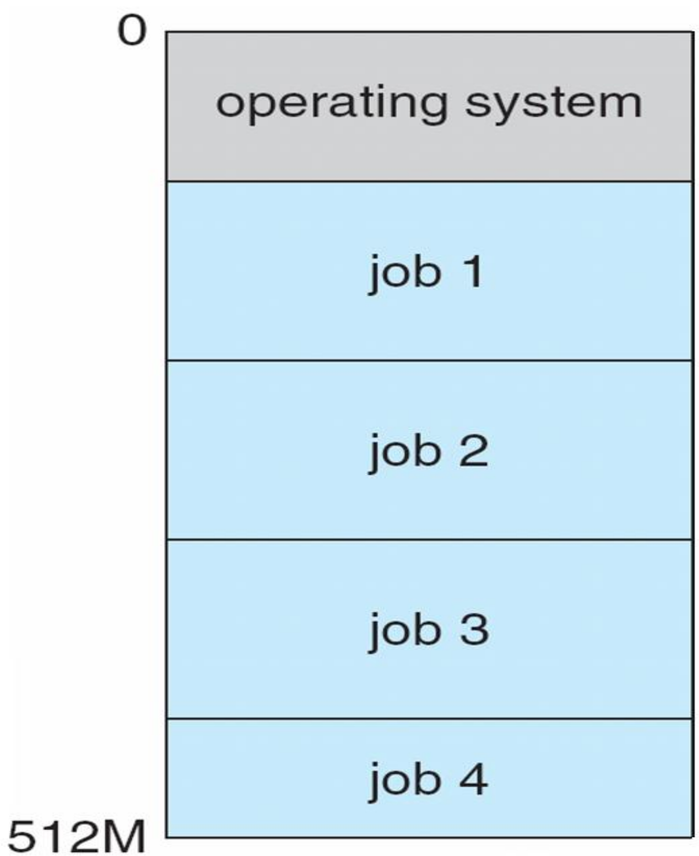
Uniprogramming

Memory Layout for Uniprogramming



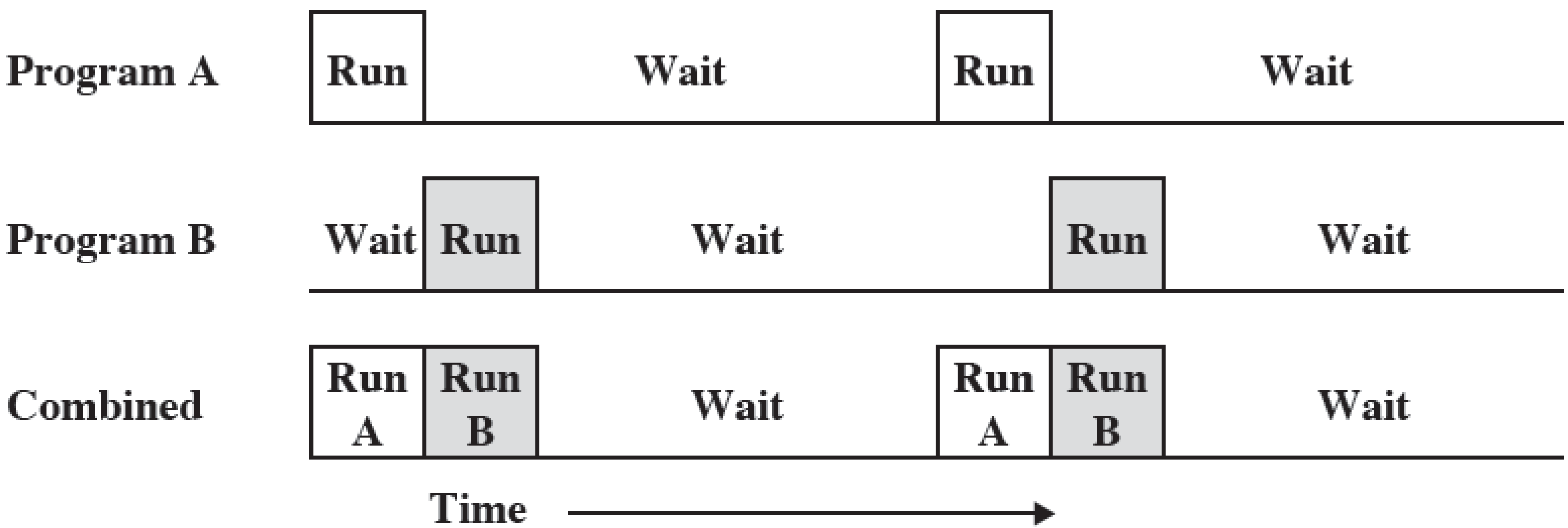
Memory Layout for Batch Multiprogramming

Several jobs are kept in main memory at the same time, and the CPU is multiplexed among them.



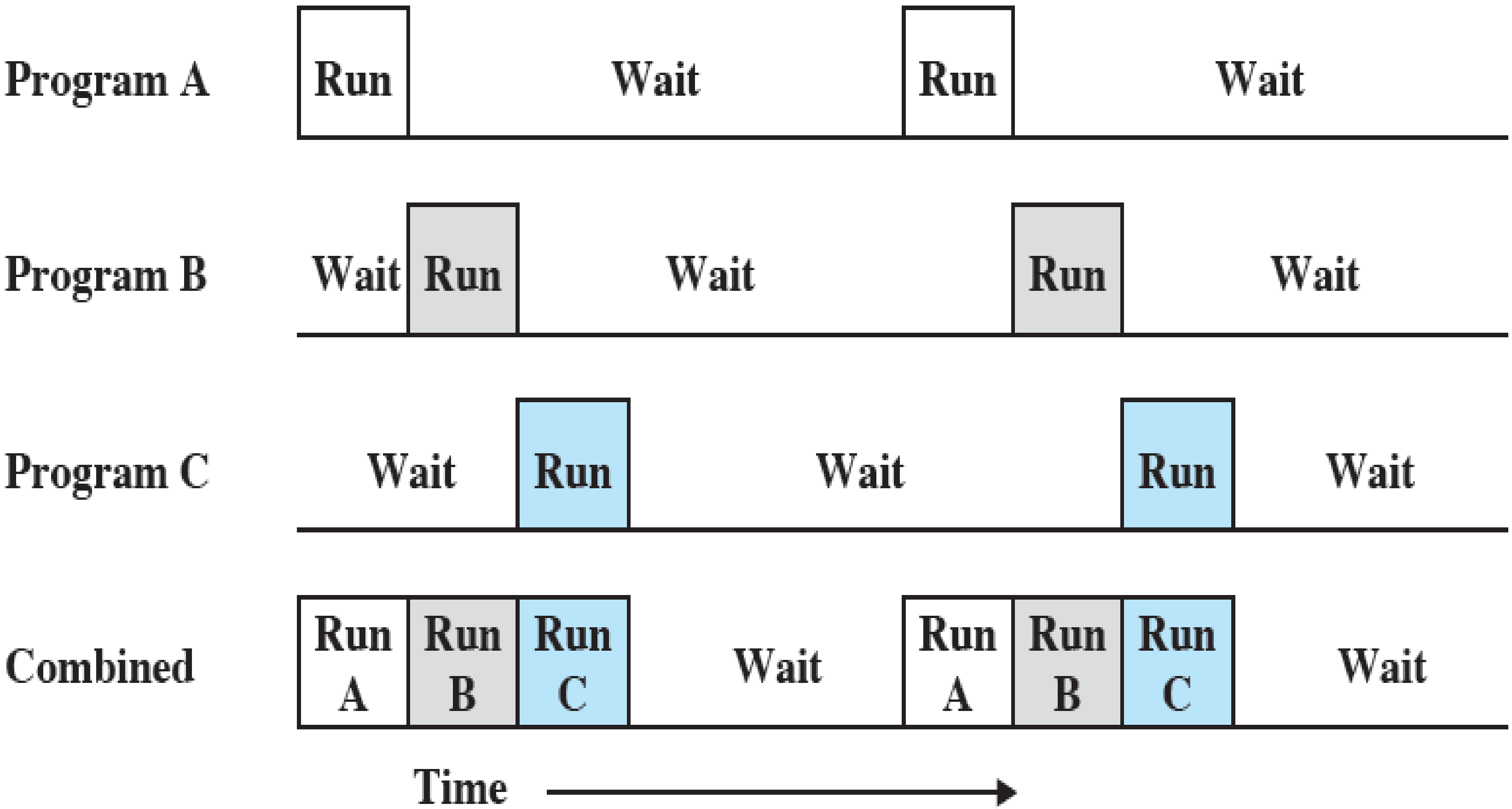
Multiprogramming (1)

(a) Uniprogramming



(b) Multiprogramming with two programs

Multiprogramming (2)



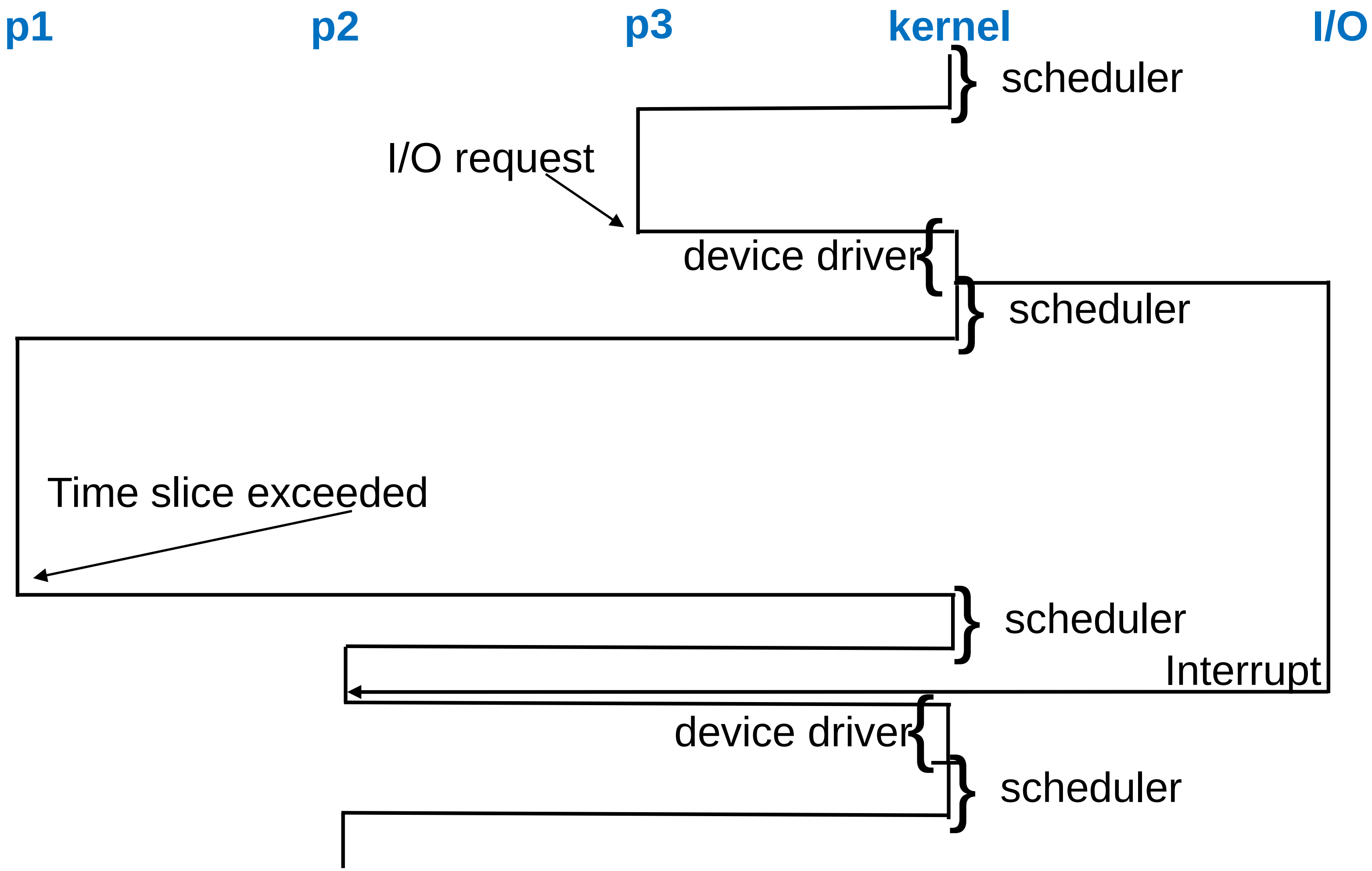
(c) Multiprogramming with three programs

Why Multiprogramming?

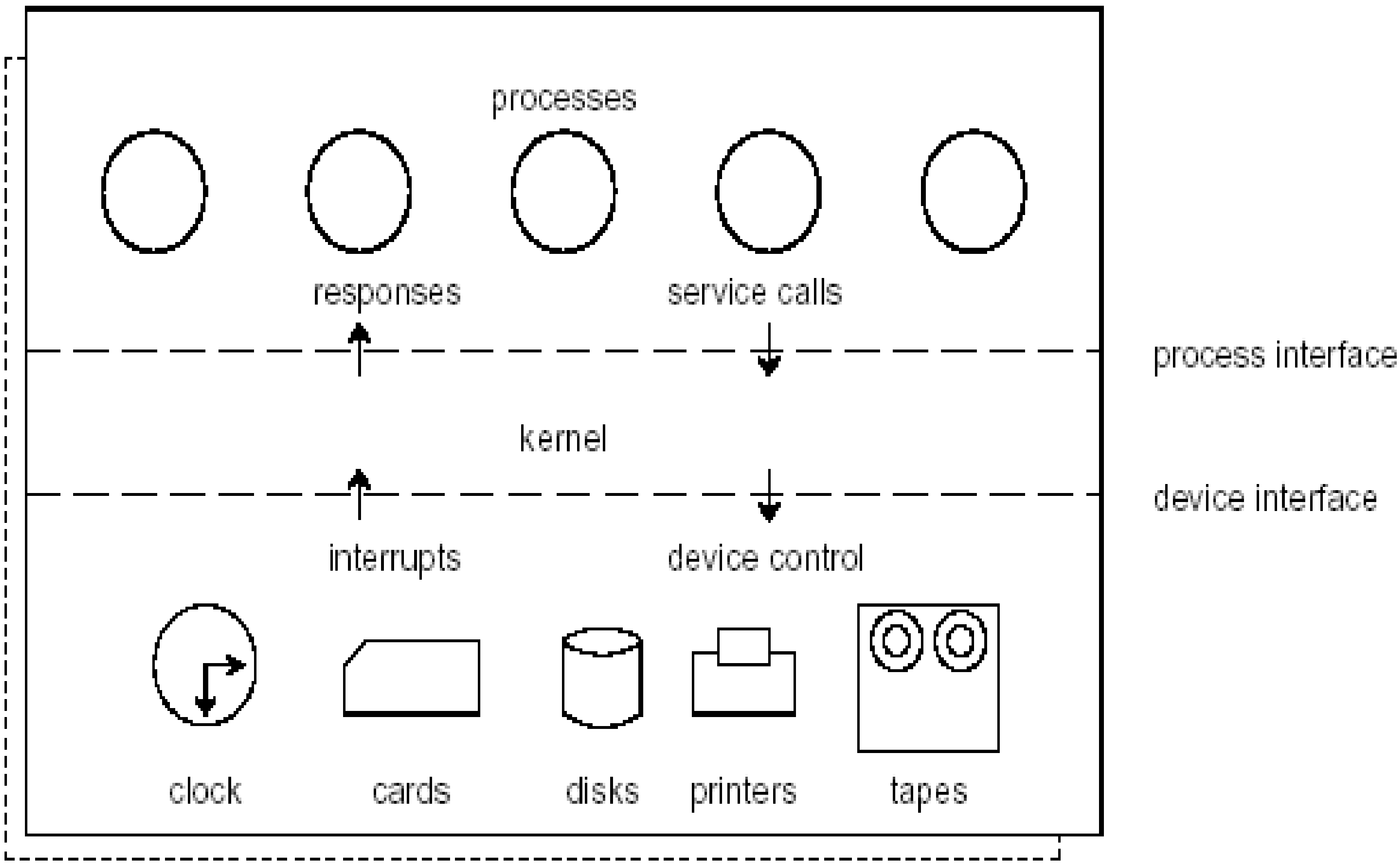
- Multiprogramming needed for efficiency:
  - Single user cannot keep CPU and I/O devices busy at all times.
  - Multiprogramming organizes jobs (code and data) so CPU always has one to execute.
  - A subset of total jobs in system is kept in memory.
  - One job selected and run via job scheduling.
  - When it has to wait (for I/O for example), OS switches to another job.

Example of Multiprogramming

- Three processes (jobs) and OS are in memory.



Components of Multiprogramming



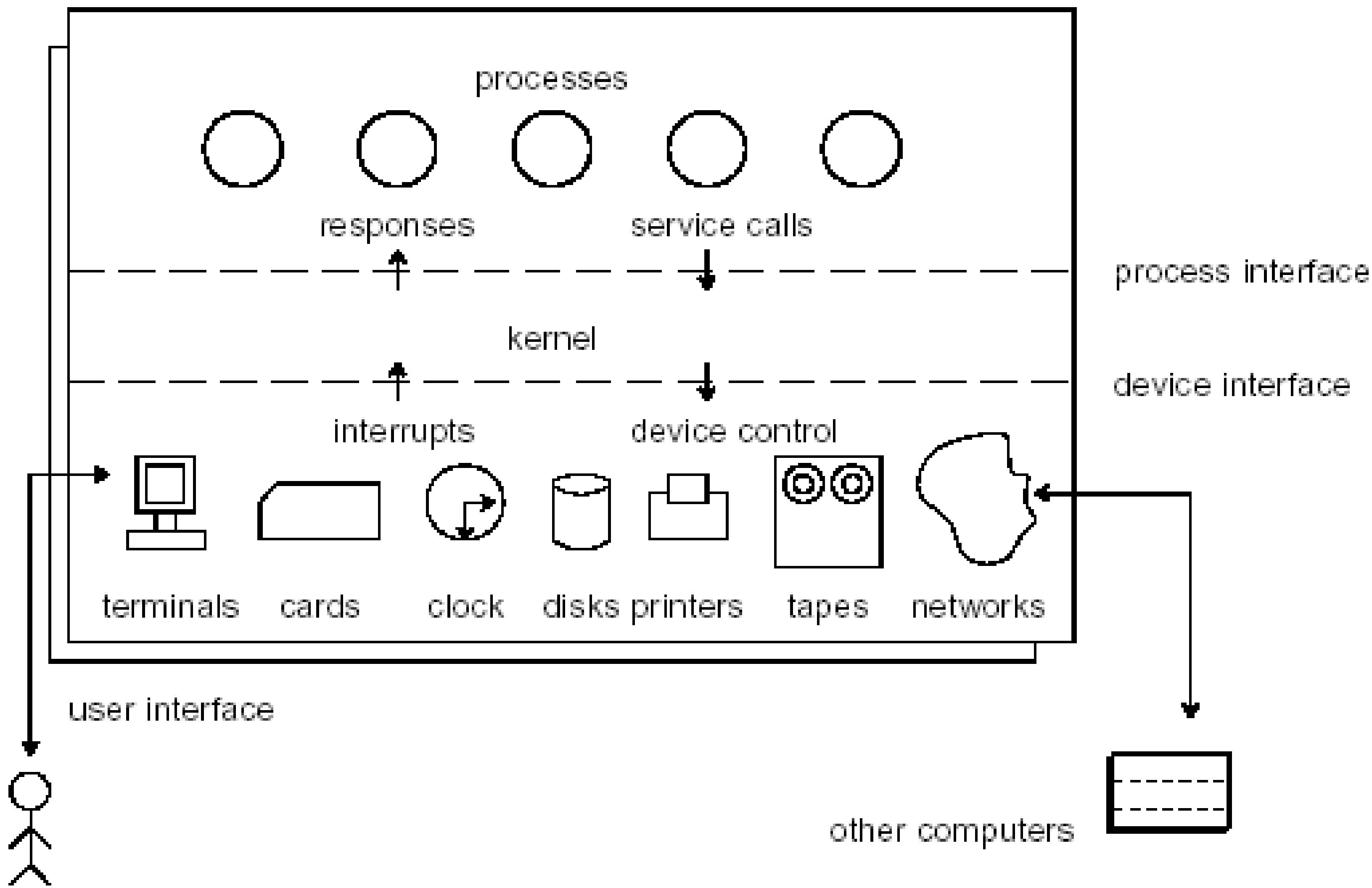
Requirements for Multiprogramming

- Hardware support:
  - I/O interrupts and DMA controllers
    - in order to execute instructions while I/O device is busy.
  - Timer interrupts for CPU to gain control.
  - Memory management
    - several ready-to-run jobs must be kept in memory.
  - Memory protection (data and programs).
- Software support from the OS:
  - For scheduling (which program is to be run next).
  - To manage resource contention.

Time-Sharing Systems (1970s)

- Batch multiprogramming does not support interaction with users.
- **Time-sharing** extends Batch Multiprogramming to handle multiple interactive jobs –
  it's Interactive Multiprogramming.
- Multiple users simultaneously access the system through commands entered at terminals.
- Processor's time is shared among multiple users.

Time-sharing Architecture



Why Time-sharing?

- In **Time-sharing** the CPU switches jobs so frequently that users can interact with each job while it is running, creating interactive computing:
  - System clock generate interrupt at the rate of 0.2 seconds and OS regains control and assigns processor to another user.
  - Each user has at least one program (process) executing in memory.
  - CPU scheduling supports several jobs ready to run at the same time.
  - If processes don’t fit in memory, swapping moves them in and out to run.
  - Virtual memory allows execution of processes not completely in memory.
  - Both batch processing and time-sharing uses multi-programming.

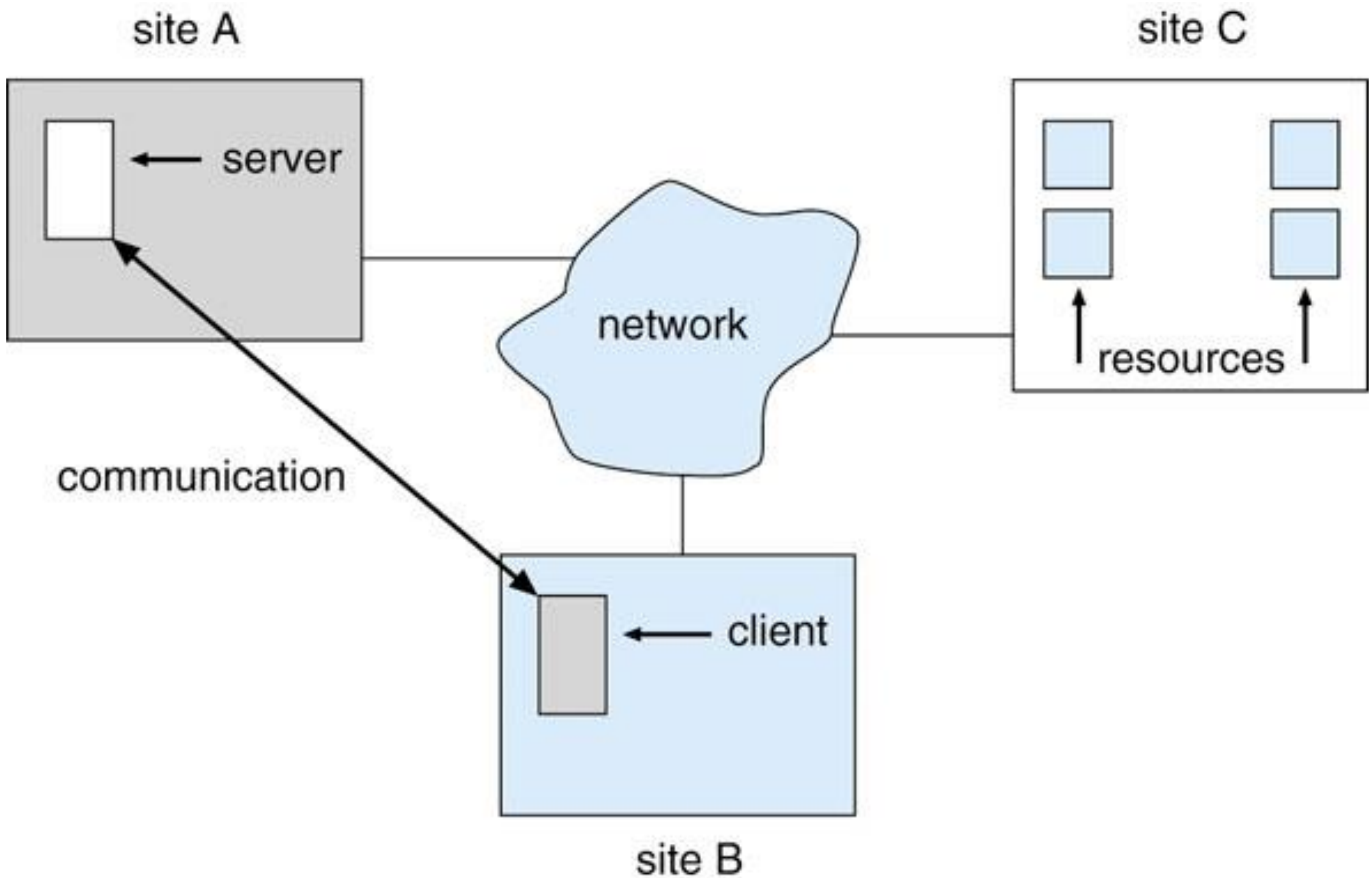
Why did Time-Sharing work?

- Because of **slow human reaction** time, a typical user needs 2 seconds of processing time per minute.
- Then many users should be able to share the same system without noticeable delay in the computer reaction time.
- The user should get a good response time.

Personal/Desktop Computers (1980s)

- *Personal computers* – computer system dedicated to a single user.
- I/O devices: keyboards, mice, display screens, small printers.
- User convenience and responsiveness.
- Can adopt technology developed for larger operating system; often individuals have sole use of computer.
- May run several different types of operating systems (Windows, MacOS, UNIX, Linux)

Distributed Systems



Distributed Systems

- Distributed system is collection of loosely coupled processors interconnected by a communications network.
- Processors variously called *nodes, computers, machines, hosts.*
- Reasons for distributed systems:
  - Resource sharing:
    - sharing files at remote sites.
    - processing information in a distributed database.
    - using remote specialized hardware devices.
  - Computation speedup – load sharing.
  - Reliability – detect and recover from site failure, function transfer, reintegrate failed site.
  - Communication – message passing.

Chapter 1: Introduction

- What Operating Systems Do
- Computer-System Organization
- Computer-System Architecture
- Operating-System Structure
- **Common System Components**
  - **Process Management**
  - Memory Management
  - Storage Management

1. Process Management

- A process is a program in execution. It is a unit of work within the system. Program is a ***passive entity***, process is an ***active entity***.
- Process needs resources to accomplish its task
  - CPU, memory, I/O, files
  - Initialization data
- Process termination requires reclaim of any reusable resources

1. Process Management Activities

- The operating system is responsible for the following activities in connection with process management:

  - Creating and deleting both processes
  - Suspending and resuming processes
  - Providing mechanisms for process synchronization
  - Providing mechanisms for process communication
  - Providing mechanisms for deadlock handling

- We will study more about Process Management in Chapter#3

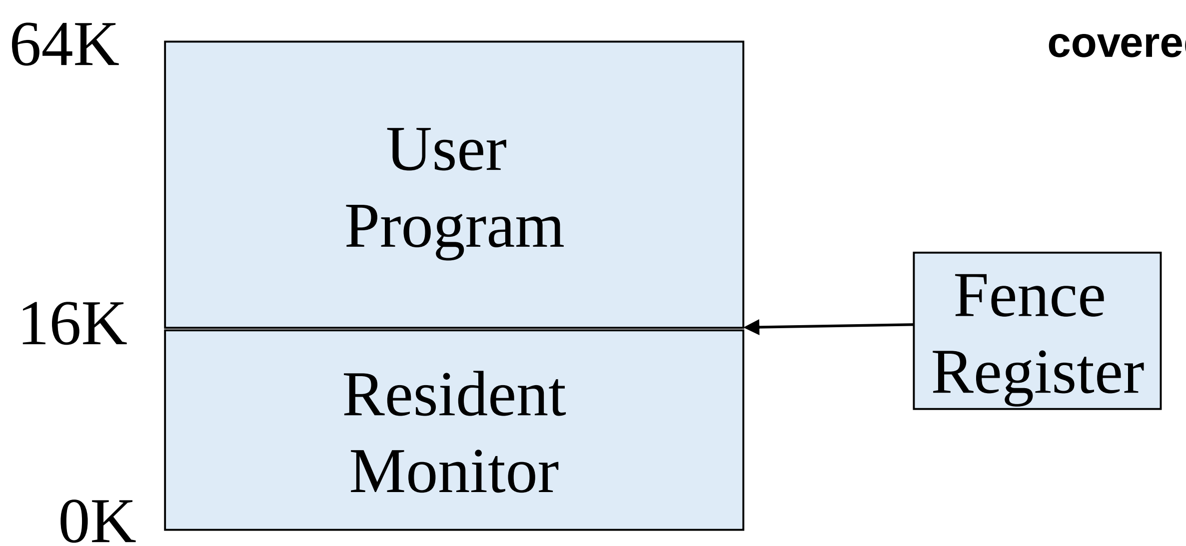
2. Main Memory Management

- Memory management determines what is in memory and when
  - ***Optimizing*** CPU utilization and computer response to users
- Memory management activities
  - Keeping track of which parts of memory are currently being used and by whom
  - Deciding which processes (or parts thereof) and data to move into and out of memory
  - Allocating and deallocating memory space as needed

Memory Management Dynamics

- Sharing system resources requires the operating system to ensure that an incorrect program cannot cause other programs to execute incorrectly.
- **Resident Monitor** is a “Trusted Program” but how to protect it from damage by the user program?
- **Solution:** Fence Register (a dedicated register) and addressing access logic.

- **Memory Management will be covered in Chapter#8 and #9**



3. Storage Management

- OS provides uniform, logical view of information storage
  - ***Abstracts*** physical properties to logical storage unit - **file**
  - Each medium is controlled by device (i.e., disk drive, tape drive)
    - Varying properties include *access speed*, *capacity*, *data-transfer rate*, *access method*
- File-System management
  - Files usually organized into directories
  - Access control on most systems to determine who can access what
  - OS activities include
    - Creating and deleting files and directories
    - ***Primitives*** (e.g., copy, move, delete) to manipulate files and directories
    - Backup files onto stable (non-volatile) storage media
- We will study more about Storage Management in Chapter#10,11

End of Chapter 1