Nanyang Polytechnic

IT2164/IT2561 Operating Systems and Administration

Chapter 1

Introduction to Operating Systems

Introduction to Operating Systems

At the end of this chapter, you should be able to

- Define what is an operating system
- Describe the difference between system software and application software
- Describe concepts of resource abstraction and resource sharing
- Understand technique of multiprogramming
- Understand Operating System strategies

Introduction to Operating Systems

- An operating system is a program that acts as an intermediary between the user of a computer and the computer hardware

- Provide an environment in which user can execute programs in a ***convenient*** and ***efficient*** manner

- Performs no useful function by itself

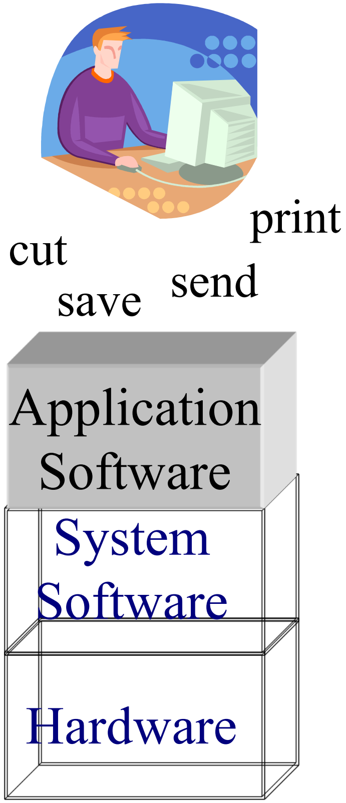
Introduction to Operating Systems

- Why study Operating System ?
  - Understand the *model of operation*
    - Easier to see how to use the system
    - Enables you to write efficient code
  - Learn to design an OS
- Even so, OS is pure overhead of real work
- Application programs have the real value to person who buys the computer

Computers and Software

- Computer systems consist of software and hardware
- Software is differentiated according to its purpose
  - **Application software** is software that allows the user to perform some intended task, function or activity and includes productivity tools.
  - **System software** provides an interface with hardware and serves as a platform for running programs and maintaining the efficiency of the system. It can be divided into operating systems and utility programs

Perspectives of the Computer



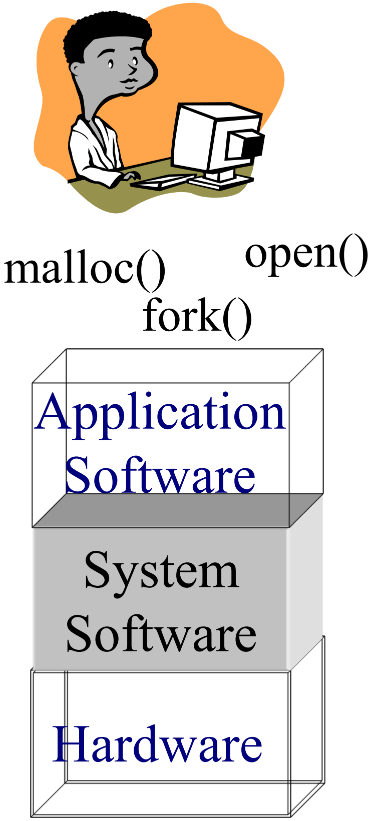
cut save send print

Application Software

System Software

Hardware

(a) End User View



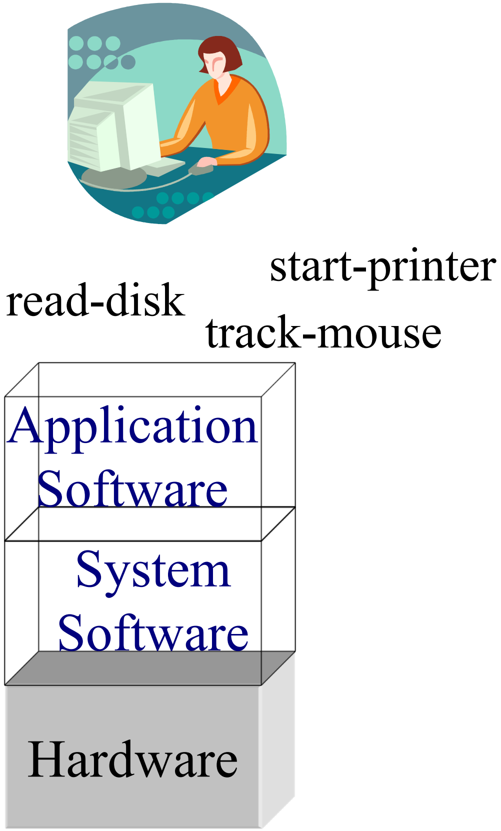
malloc() fork() open()

Application Software

System Software

Hardware

(b) Application Programmer View



read-disk track-mouse start-printer

Application Software

System Software

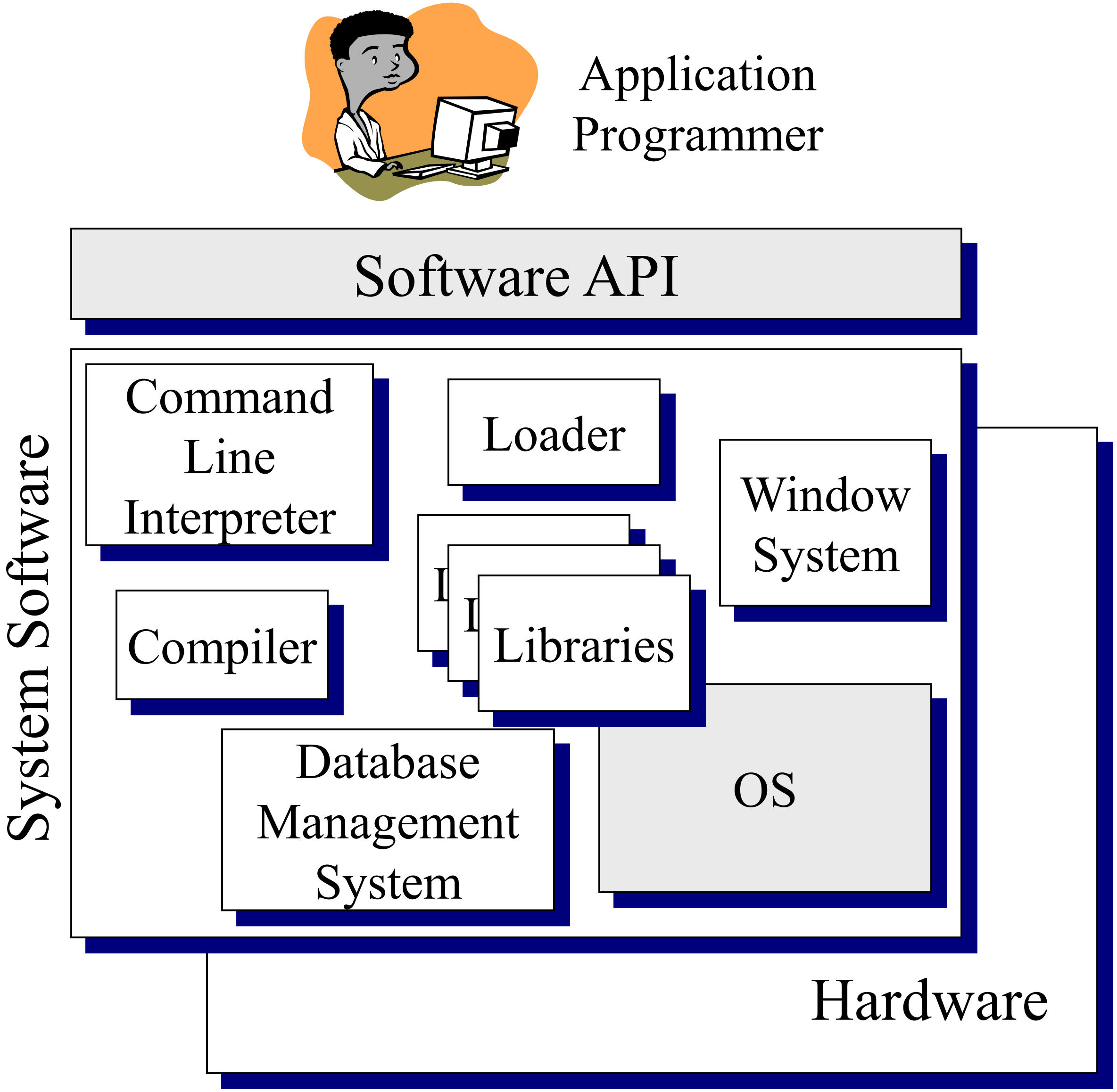
Hardware

(c) OS Programmer View

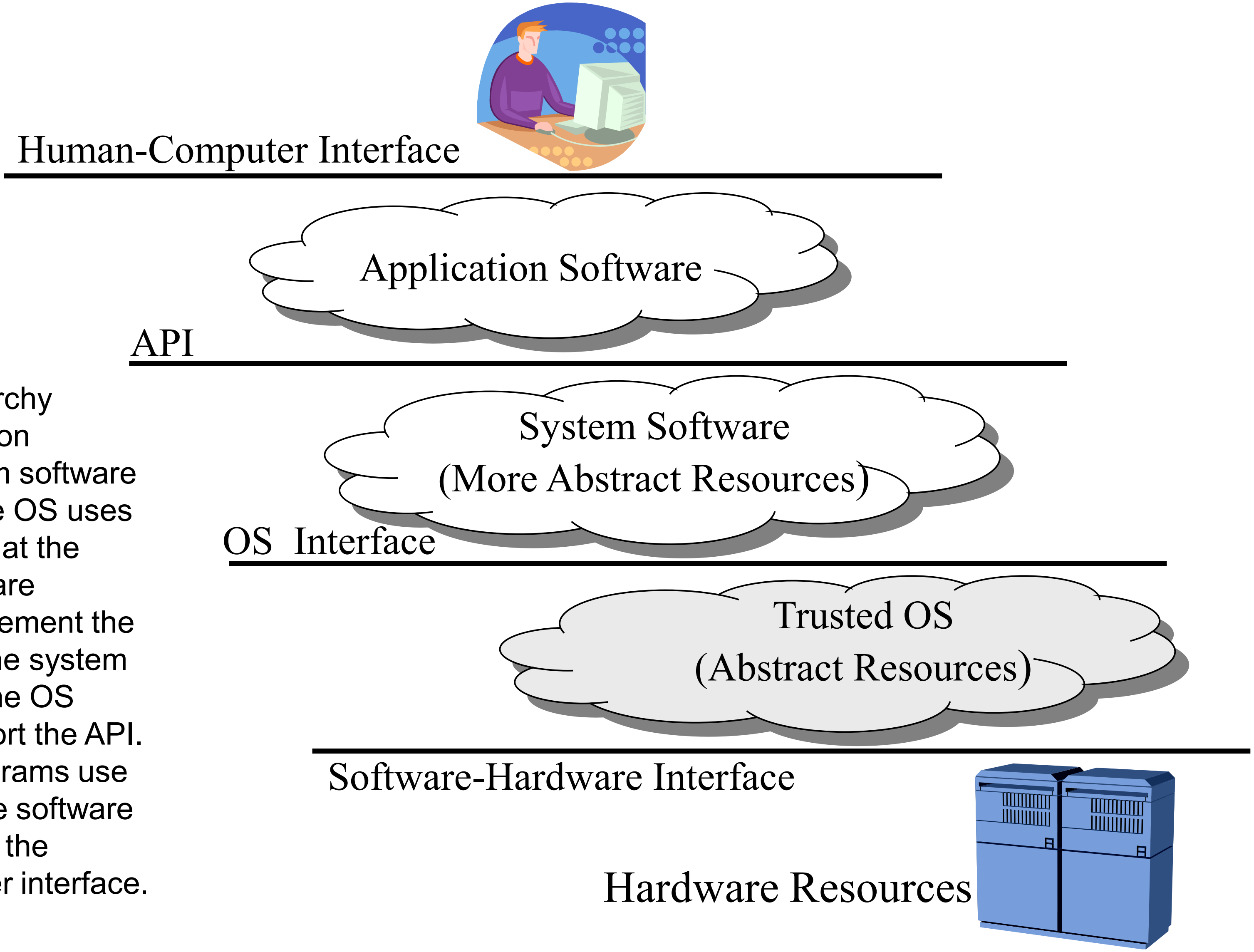
System Software

- System software provides two kinds of environment
  - Allows human users to interact with the computer
  - Provides tools and subassemblies used with application programs
- Independent of individual applications, but common to all of them
- Examples
  - C library functions
  - A window system
  - A database management system
  - Resource management functions
  - The OS

Using the System Software



Application Software, System Software, and the OS



Human-Computer Interface

Application Software

API

System Software

(More Abstract Resources)

OS Interface

Trusted OS

(Abstract Resources)

Software-Hardware Interface

Hardware Resources

There is a hierarchy among application software, system software and the OS. The OS uses the functionality at the software-hardware interface to implement the OS interface. The system software uses the OS interface to export the API. Application programs use the API to create software that implements the human-computer interface.

The OS as a Resource Manager

- Resource: Anything that is needed for a executing program to run
  - Memory
  - Space on a disk
  - The CPU

- Operating system can be viewed as a resource manager
  - "An OS creates resource abstractions"
  - "An OS manages resource sharing"

Resource Abstraction

- Abstraction is when an OS hides the actual tasks needed to manage and use resources
- Allows user programs to use these resources by using simpler commands to access these resources.
- Makes it easy for user programs to use resources in a computer system.

Resource Abstraction

- Examples:
  - Writing a file to disk
  - Displaying text/graphics on screen
  - Running an application
- Simplifies usage but limits flexibility
  - Certain operations become easy to perform while other operations may be impossible to achieve

Resource Abstraction

Example- Copying information from memory to disk

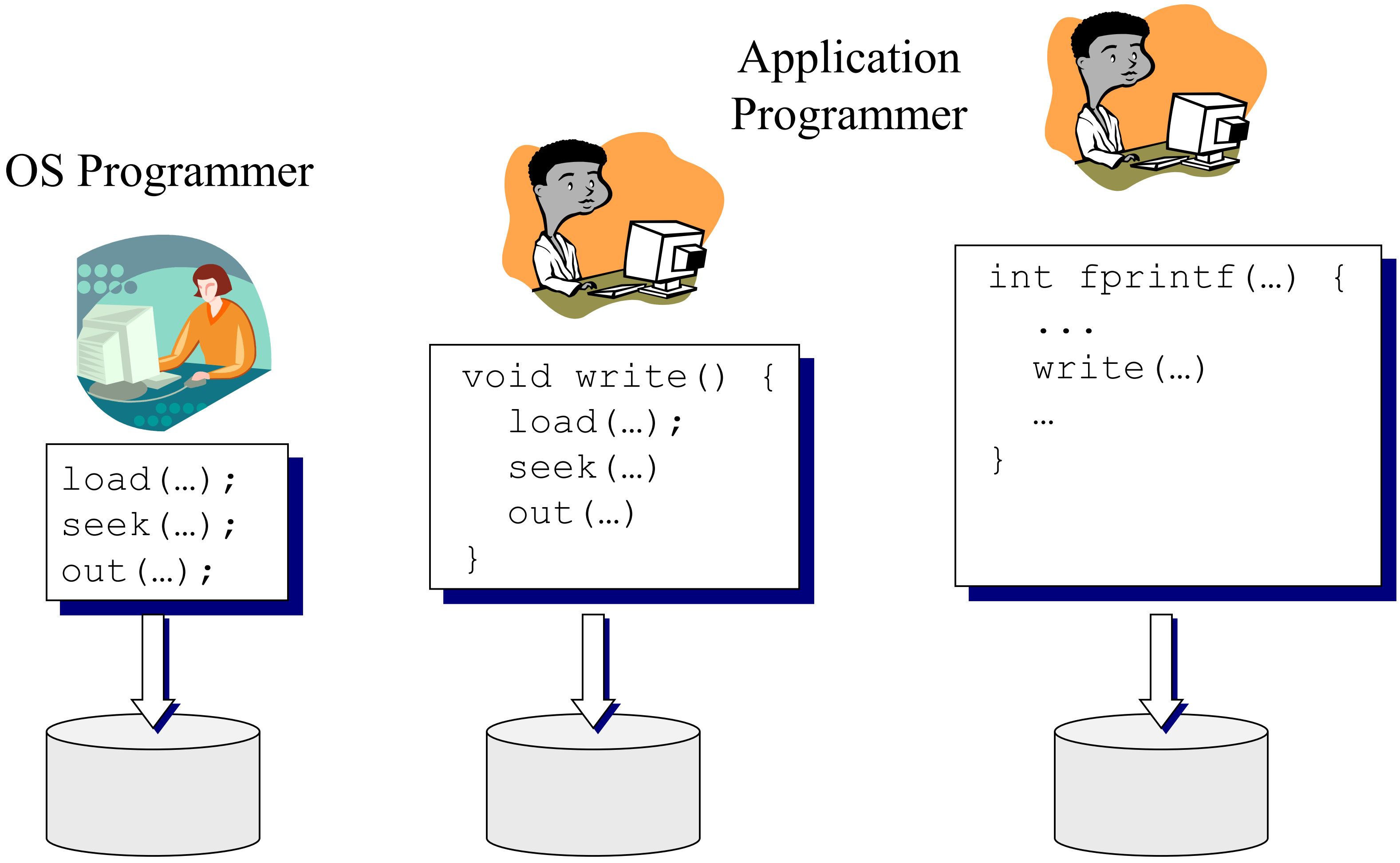
```
load(block, length, device);
seek(device, 236);
out(device, 9)
```

```
write(char *block, int len, int device,
                  int track, int sector) {
   ...
  load(block, length, device);
  seek(device, 236);
  out(device, 9);
   ...
}
```

```
write(char *block, int len, int device,int addr);
```

```
fprintf(fileID, “%d”, datum);
```

Disk Abstractions



OS Programmer

```
load(…);
seek(…);
out(…);
```

Application Programmer

```
void write() {
   load(…);
   seek(…)
   out(…)
}
```

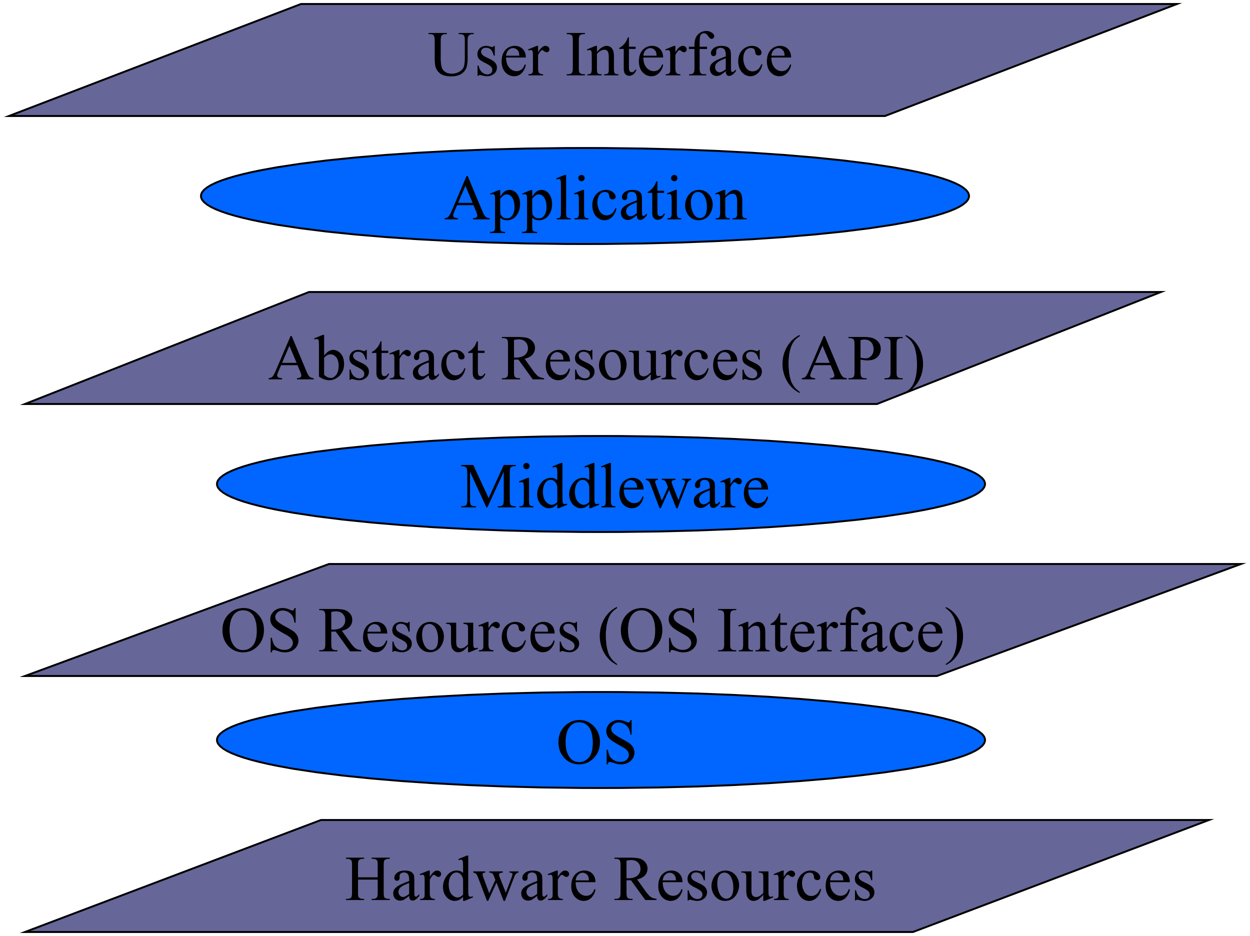
```
int fprintf(…) {
   ...
   write(…)
   …
}
```

(a) Direct Control

(b) `write()` abstraction

(c) `fprintf()` abstraction

Abstract Resources


User Interface

Application

Abstract Resources (API)

Middleware

OS Resources (OS Interface)

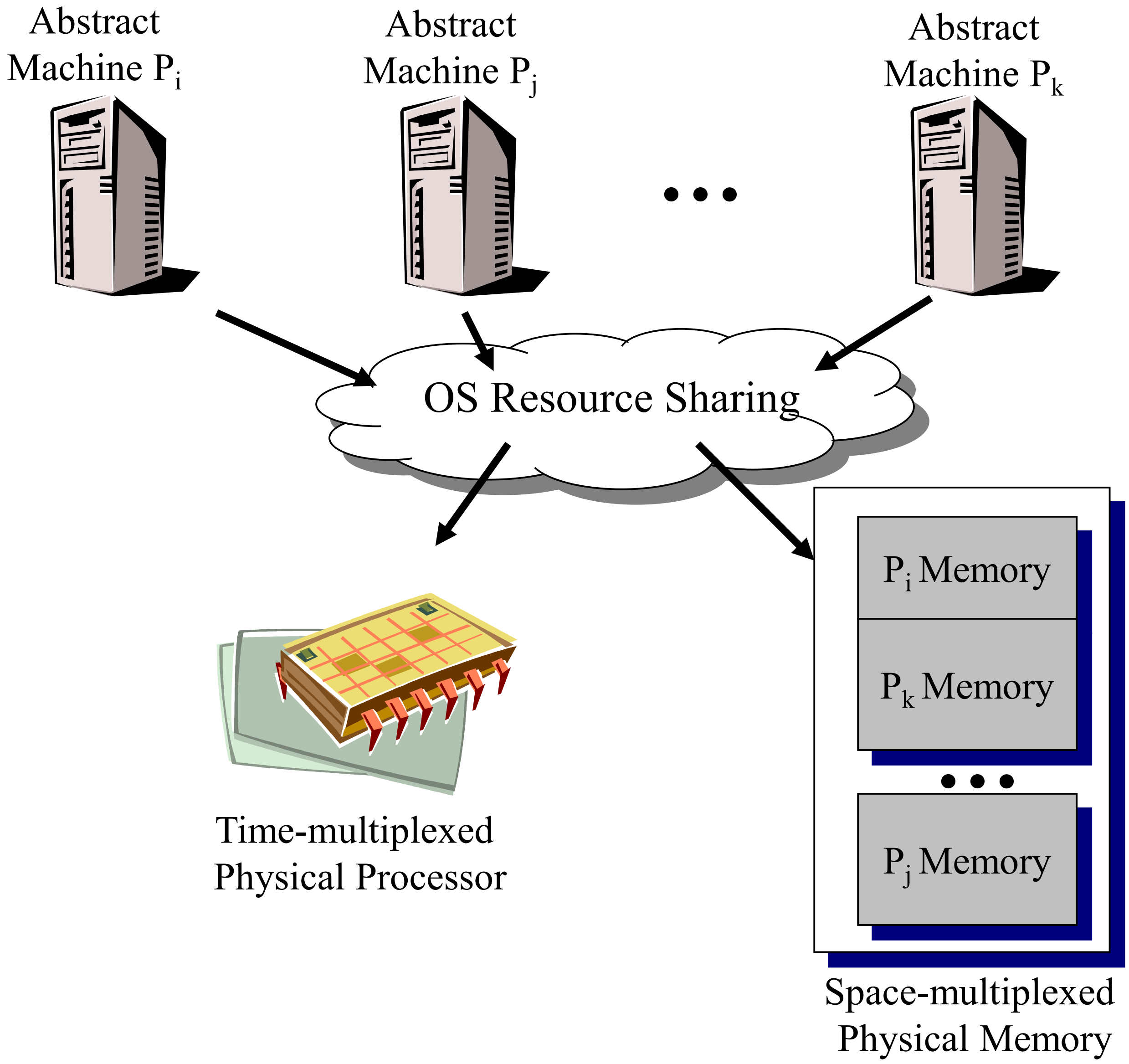
OS

Hardware Resources

Resource Sharing

- Two kinds of sharing
  - Space-multiplexed sharing
    - The resource is divided into two or more distinct units and each unit is allocated to different processes
  - Time-multiplexed sharing
    - The entire resource is allocated to a process for a period of time, after which it is then allocated to another process and so on.

Resource Sharing



Abstract Machine P_i

Abstract Machine P_j

• • •

Abstract Machine P_k

OS Resource Sharing

Time-multiplexed Physical Processor

P_i Memory

P_k Memory

• • •

P_j Memory

Space-multiplexed Physical Memory

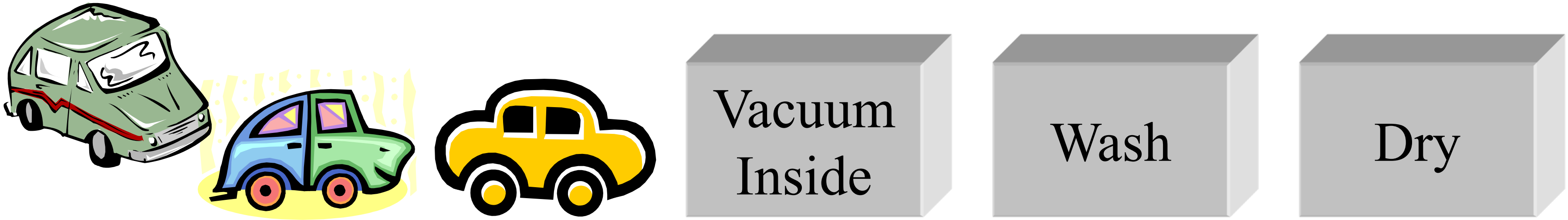
Multiprogramming

- Refers to the technique for **sharing** the CPU among **runnable** processes

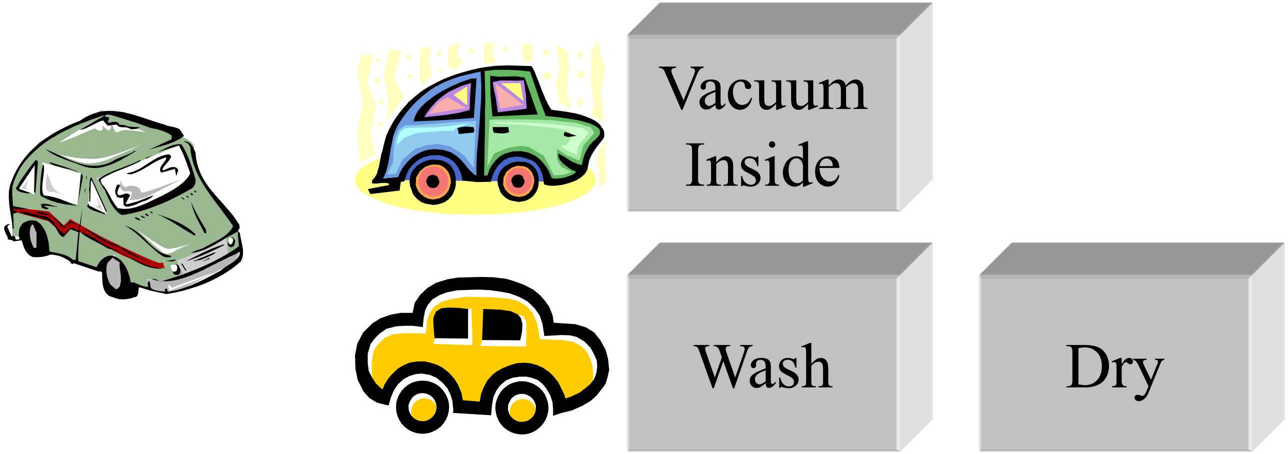
- How does it work ?
  - Process may be *blocked* on I/O
  - Process may be *blocked* waiting for other resource, including the CPU
  - While one process is blocked, another might be able to run
  - Increases CPU utilization

- Multiprogramming OS accomplishes CPU sharing “automatically” – *scheduling*

Speeding Up the Car Wash

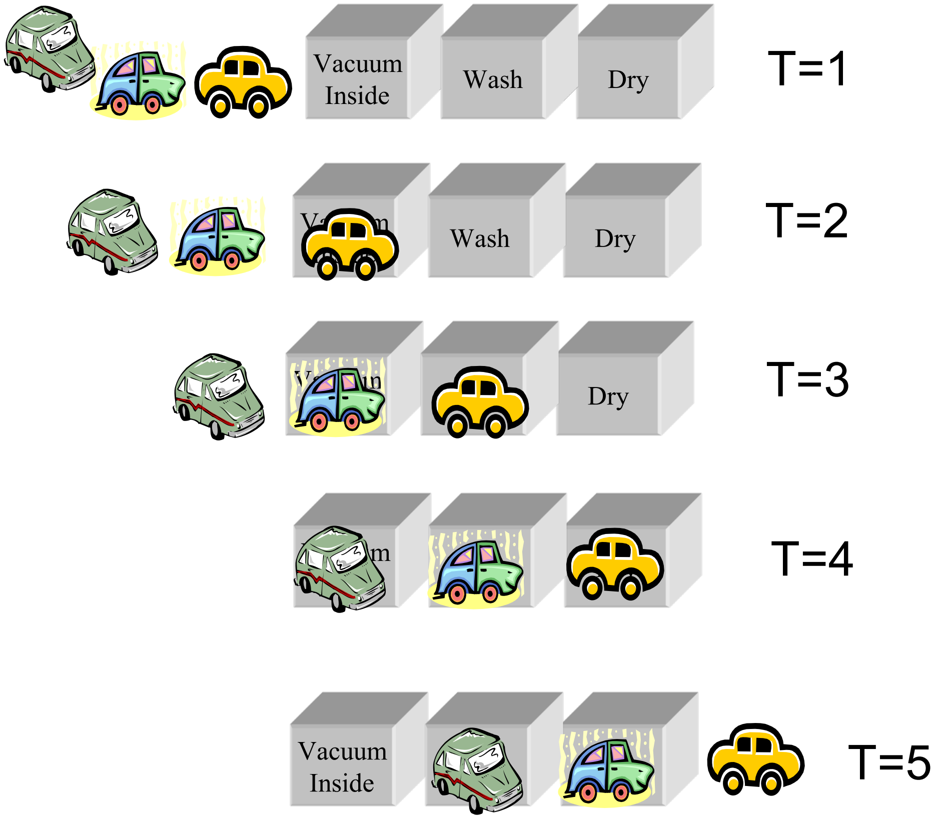


(a) The Sequential Car Wash

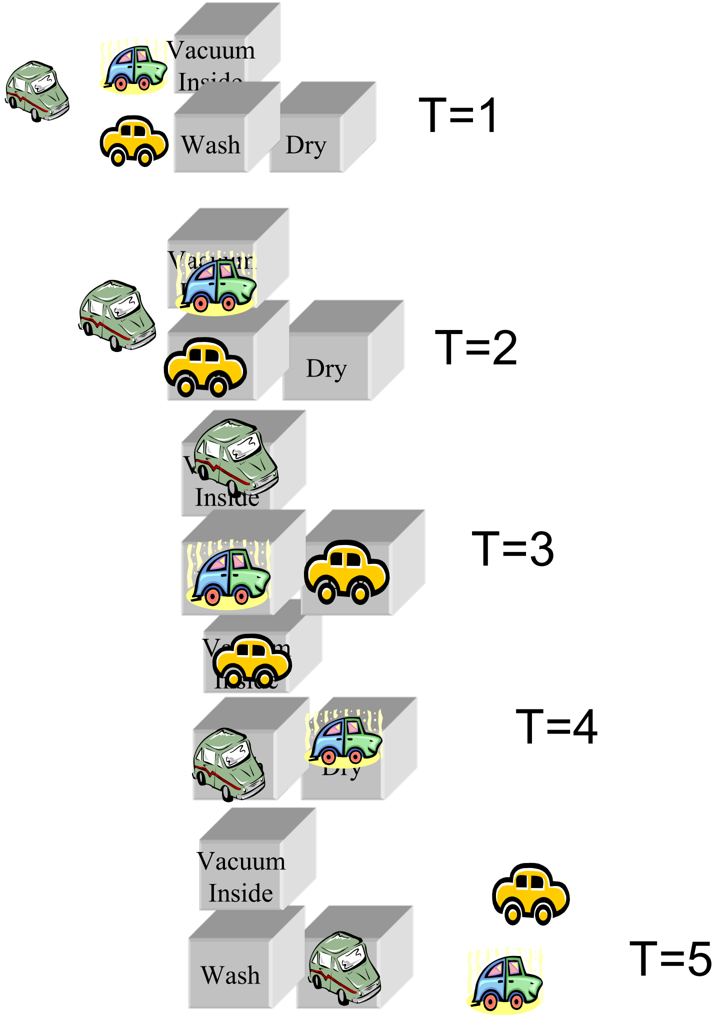


(b) The Parallel Car Wash

Speeding up the Car Wash

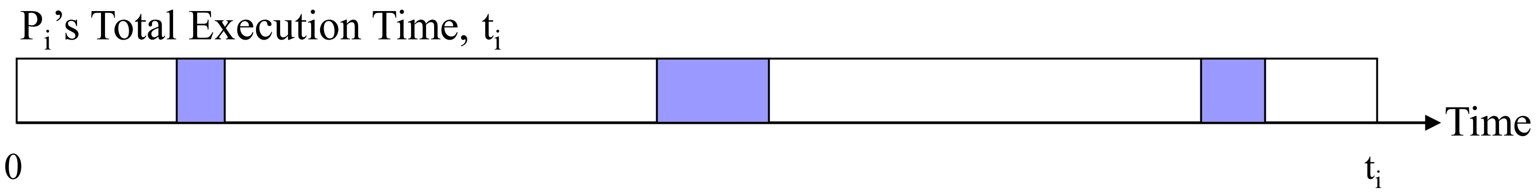


(a) The Sequential Car Wash

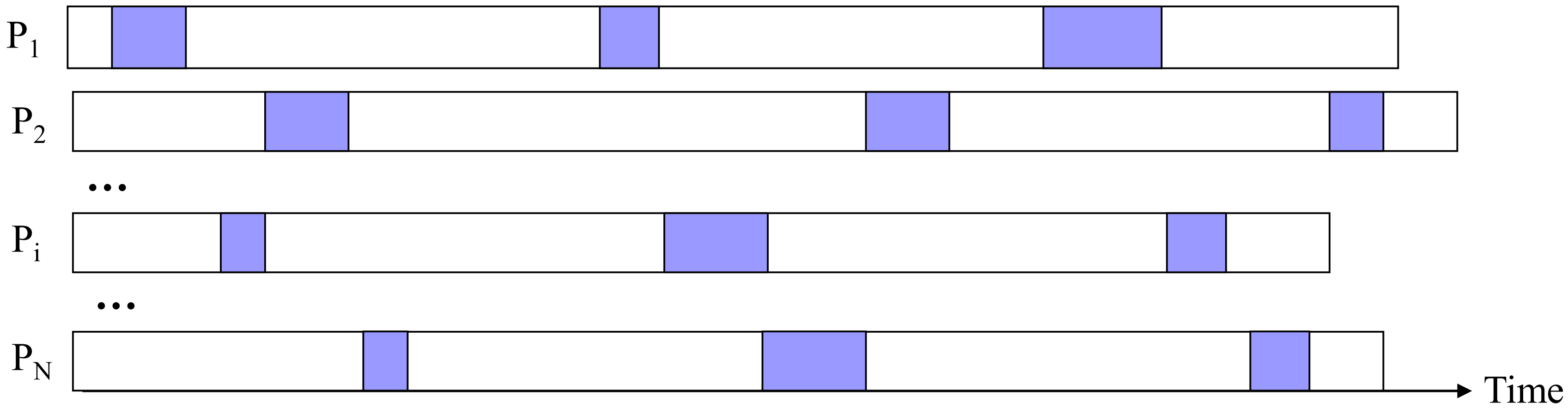


(b) The Parallel Car Wash

Multiprogramming Performance



(a) P_i's Use of Machine Resources



(a) All Processes' Use of Machine Resources

■ Using the processor

□ I/O operation

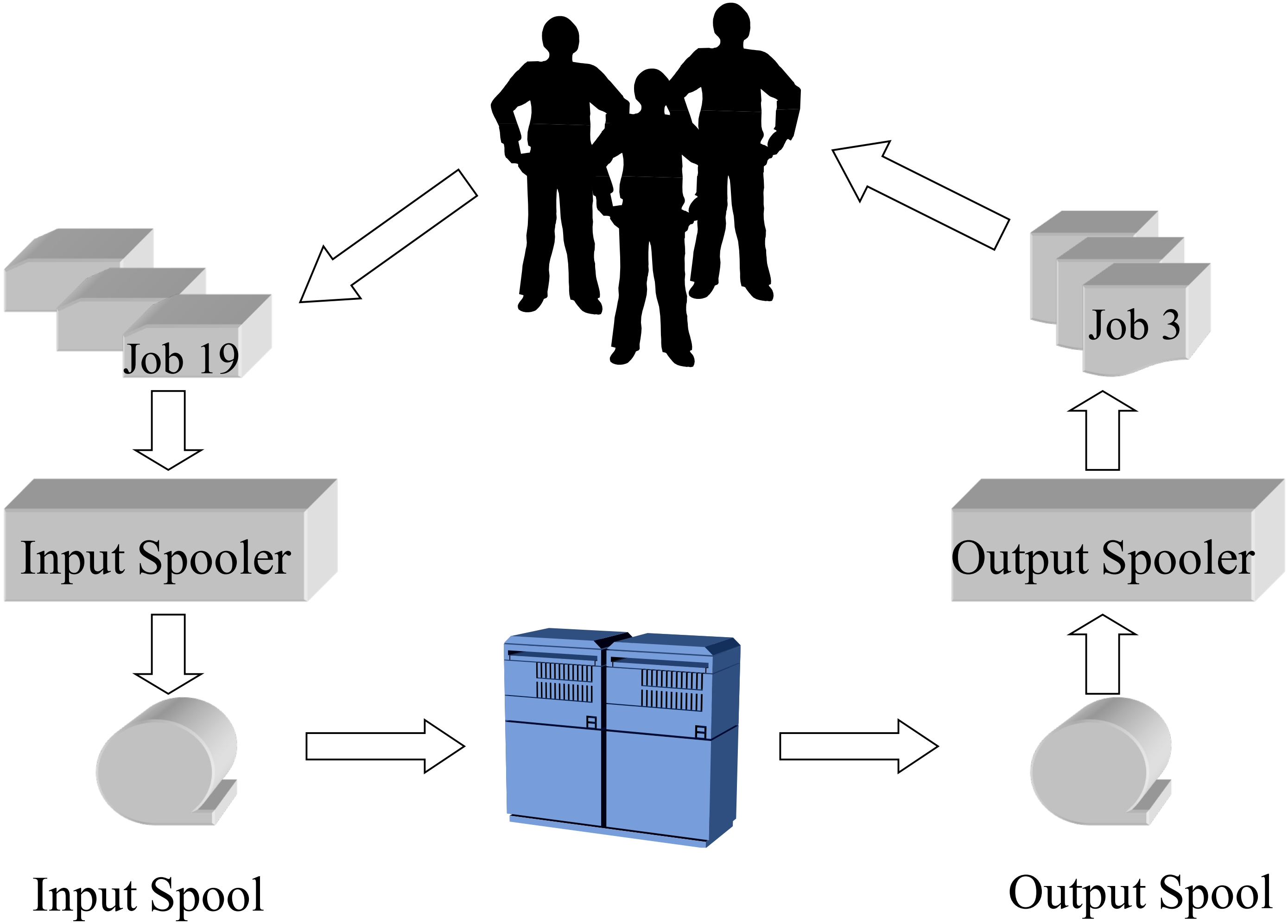
OS Strategies

- Different strategies have been used to provide OS services
- Refers to the general characteristics of the programmer’s abstract machine.
- Depends on business and engineering criteria
  - How will the computer be used?
  - Is human interaction important?
  - Will there be more than one person using?
  - Is response time critical?

OS Strategies

- Batch processing
- Timesharing
- Personal computer & workstations
- Others
  - Process control & real-time
  - Network
  - Distributed
  - Small computers

Batch Processing



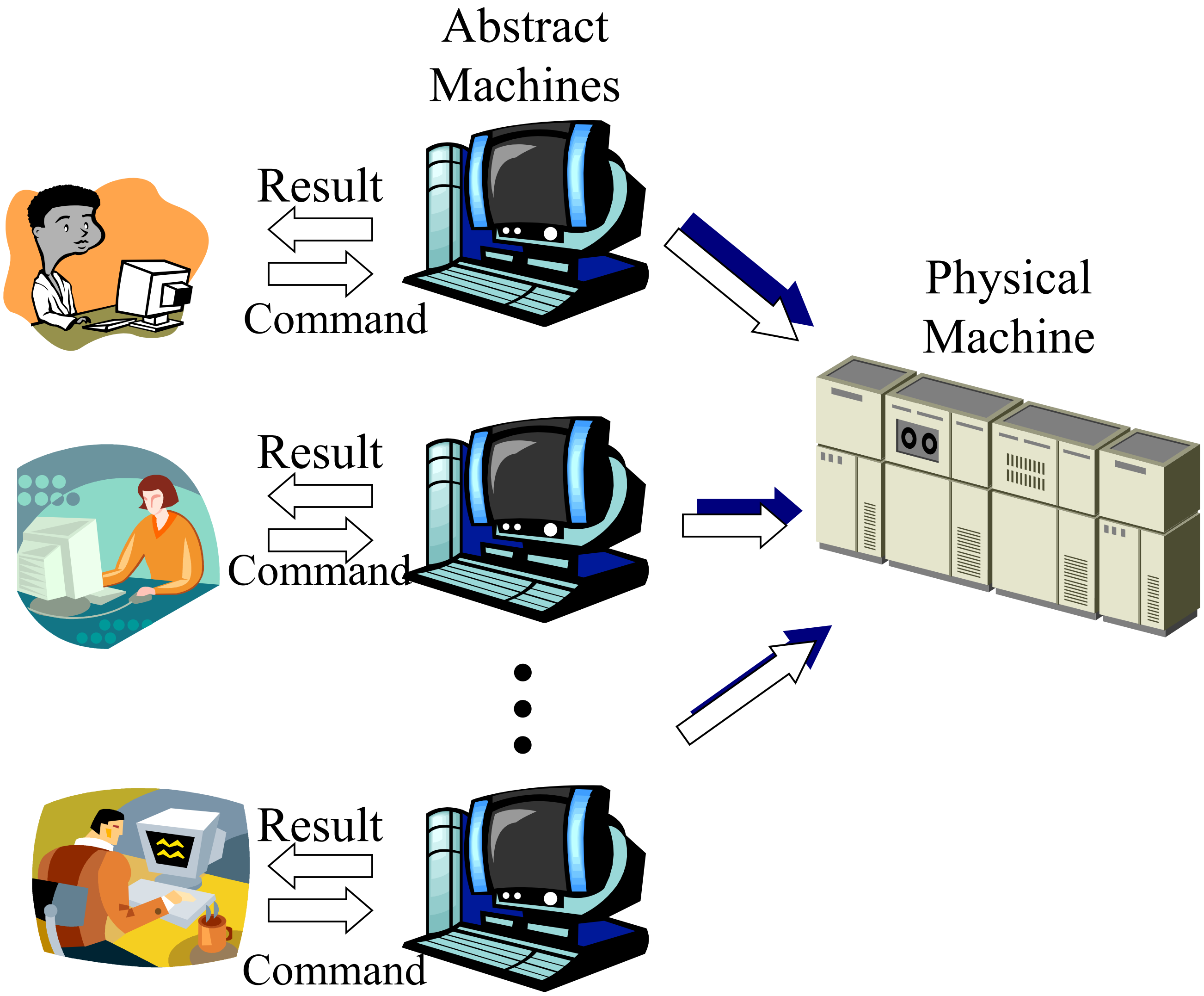
Batch Processing

- Uses multiprogramming
- Job (file of OS commands) prepared offline
- Batch of jobs given to OS at one time
- OS processes jobs one-after-the-other
- No human-computer interaction
- OS optimizes resource utilization
- Batch processing (as an option) still used today

A Shell Script Batch File

```
cc -g -c menu.c
cc -g -o driver driver.c menu.o
driver < test_data > test_out
lpr -PthePrinter test_out
tar cvf driver_test.tar menu.c driver.c test_data test_out
uuencode driver_test.tar driver_test.tar >driver_test.encode
```

Timesharing Systems



Abstract Machines

Result

Command

Result

Command

Result

Command

Physical Machine

Timesharing Systems

- Uses multiprogramming
- Support interactive computing model (Illusion of multiple consoles)
- Different scheduling & memory allocation strategies than batch
- Tends to propagate processes
- Considerable attention to resource isolation (security & protection)
- Tend to optimize response time

Examples of Modern OS

- UNIX
  - Developed by AT&T Bell labs researchers in 1970
  - Due to need for a simple, small OS
  - Primarily a command line oriented operating system
  - Open operating system – easier to extend
  - Variants: System V Unix, BSD Unix, HP-UX, Sun Solaris, Mach OS
  - Development of standardised UNIX system call interface – POSIX.1
  - Time sharing OS

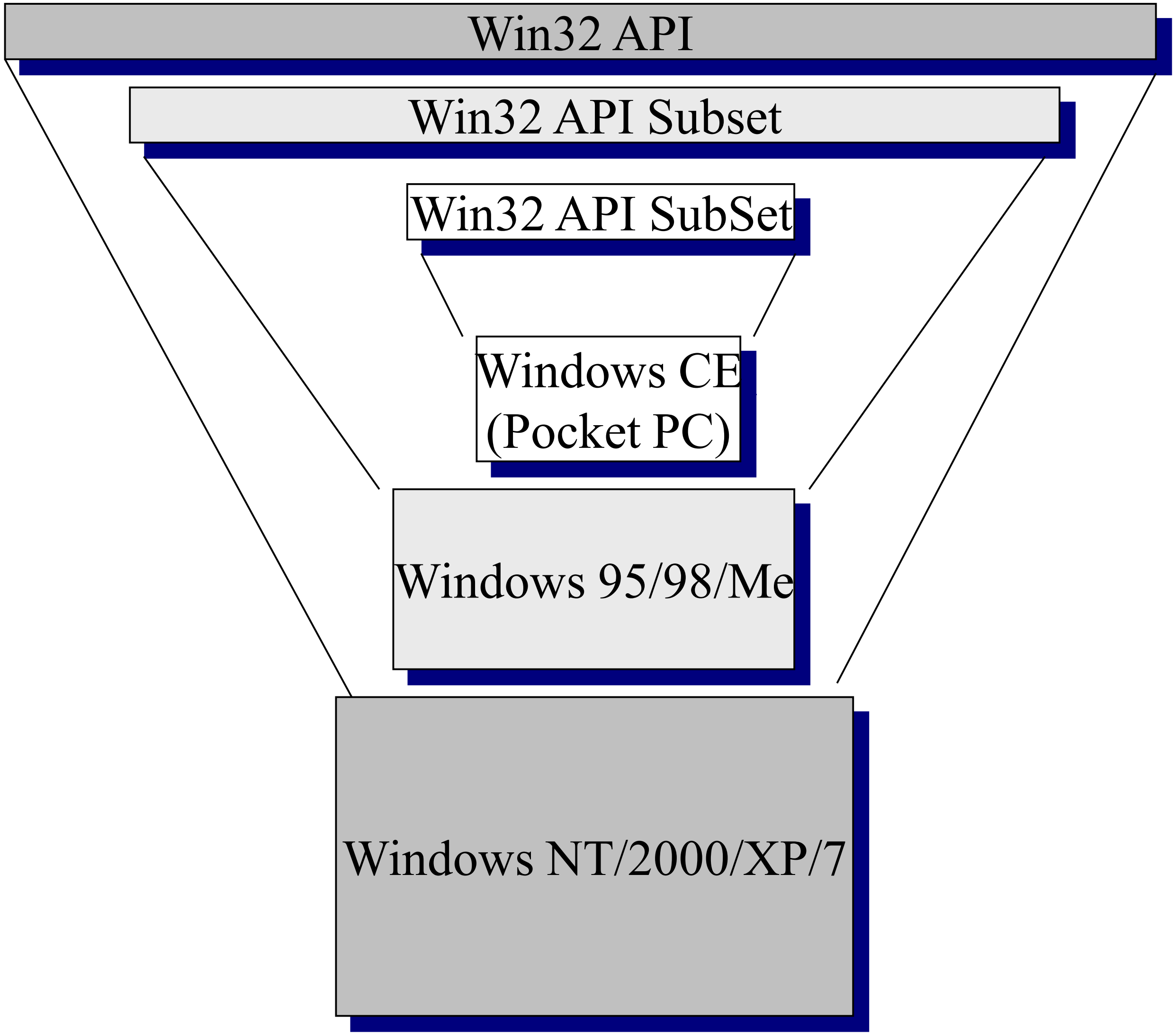
Examples of Modern OS

- Linux
  - □ "Open source UNIX"
  - □ Developed by Linus Torvalds for 80386 processor in 1991
  - □ Evolved as collaboration by many users corresponding over internet
  - □ Variants :Redhat Linux, SuSE Linux
  - □ Multiuser, multitasking OS with full set of UNIX-compatible tools
  - □ Recent work concentrates on standardization: POSIX

Examples of Modern OS

- Windows family of OS
  - Evolves through Windows 3.x, Windows 95/98, Windows NT/ 2000/XP, Windows Vista, Windows CE, Windows Mobile
  - Heavily window-oriented
  - Command line available – `cmd.exe`
  - Object oriented design
  - Heavy use of threads
  - File system: FAT, FAT32, NTFS, DFS
  - Active Directory – authentication, rights, policies
  - System call interface – Win32 API

The Microsoft OS Family



Conclusion

- Operating systems perform overhead, but critical tasks in the functioning of a computer system.
- Many different OSs are developed for different purposes.
- The modern OS uses resource abstractions and sharing to allow programs to use resources.
- Operating systems aims to maximize use of computer resources without the user’s or program’s involvement.