# 系統程式設計
# Systems Programming

## 鄭卜王教授
## 臺灣大學資訊工程系

Tei-Wei Kuo, Chi-Sheng Shih, and Hao-Hua Chu©2008
Department of Computer Science and Information Engineering
Graduate institute of Multimedia and Networking, National Taiwan University

# Contents

1. Basic OS Concepts
2. File I/O
3. Standard I/O Library
4. Files and Directories
5. System Data Files and Information
6. Environment of a Unix Process
7. Process Control
8. Signals
9. Inter-process Communication
10. Thread Programming
11. Network Programming

# Chapter 0

- **Operating System Concepts Using UNIX as an Example**

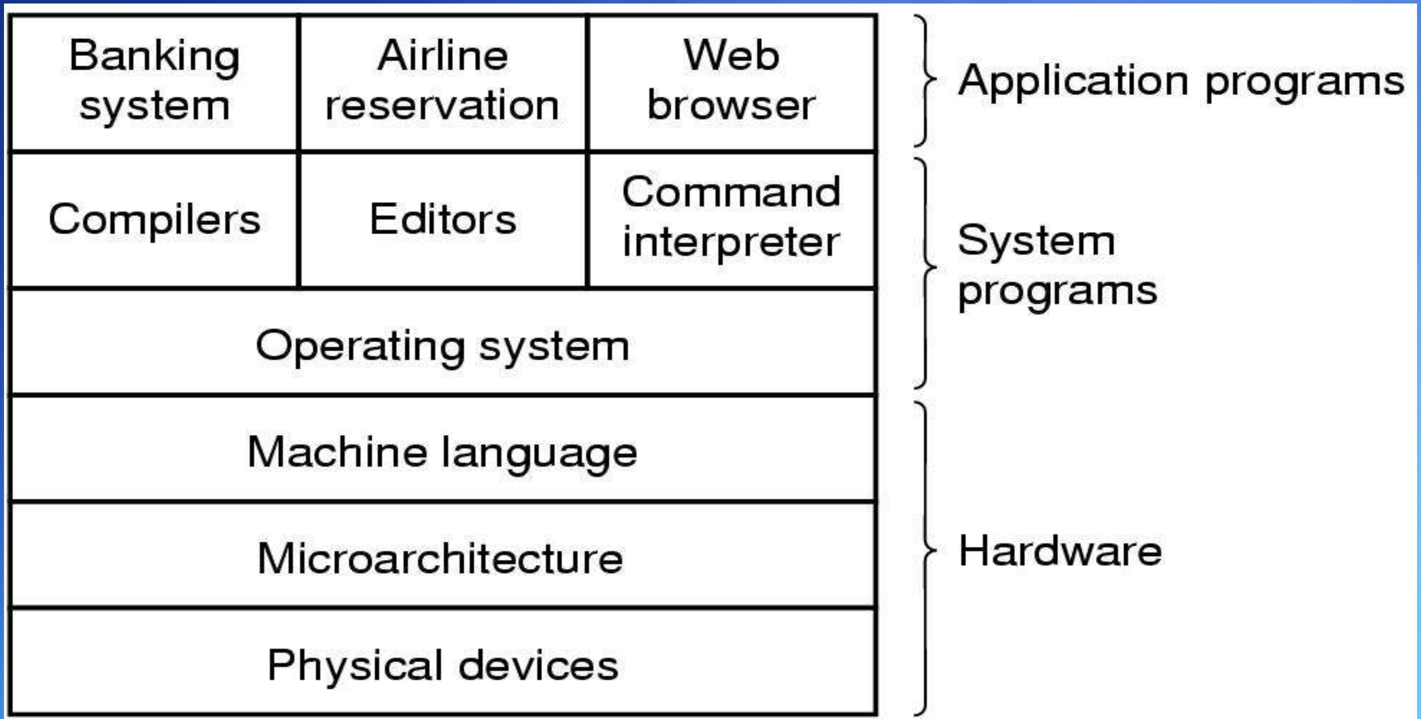
**UNIX is a kind of Operating Systems**

ISA

# Computer System

- **A computer system consists of**

  hardware, system programs, application programs



Banking system | Airline reservation | Web browser — Application programs

Compilers | Editors | Command interpreter

Operating system — System programs

Machine language

Microarchitecture

Physical devices — Hardware

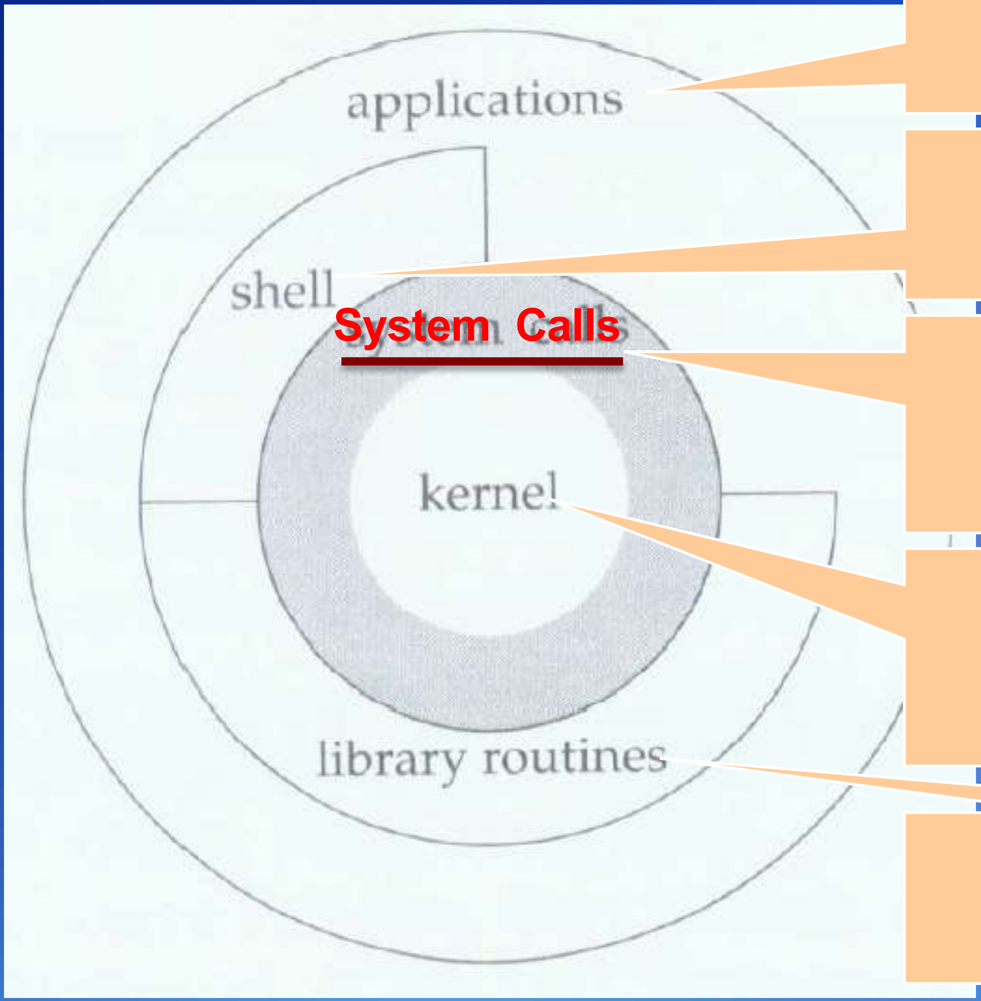
# What is Operating System

- **It is an extended machine (vertical)**
  - Presents user with a virtual machine, easier to use (establishes a user interface)
  - Hides the messy details which must be performed (executes and provides services **safely**)
- **It is a resource manager (horizontal)**
  - Resources: CPU, memory, I/O devices (disks, printers)
  - Each program gets time with the resource
  - Each program gets space on the resource
  - OS makes sure programs have a **fair** use

# What is Operating System

- **It is an extended machine (vertical)**
  - Presents user with a virtual machine, easier to use (establishes a user interface)
  - Hides the messy details which must be performed (executes and provides services safely)
- **It is a resource manager (horizontal)**
  - Resources: CPU, memory, I/O devices (disks, printers)
  - Each program gets time with the resource
  - Each program gets space on the resource
  - OS makes sure programs have a fair use

# UNIX Architecture (vertical viewpoint)



Written by programmer
Compiled by programmer

Interactive interface
Users can issue commands (e.g., ls)

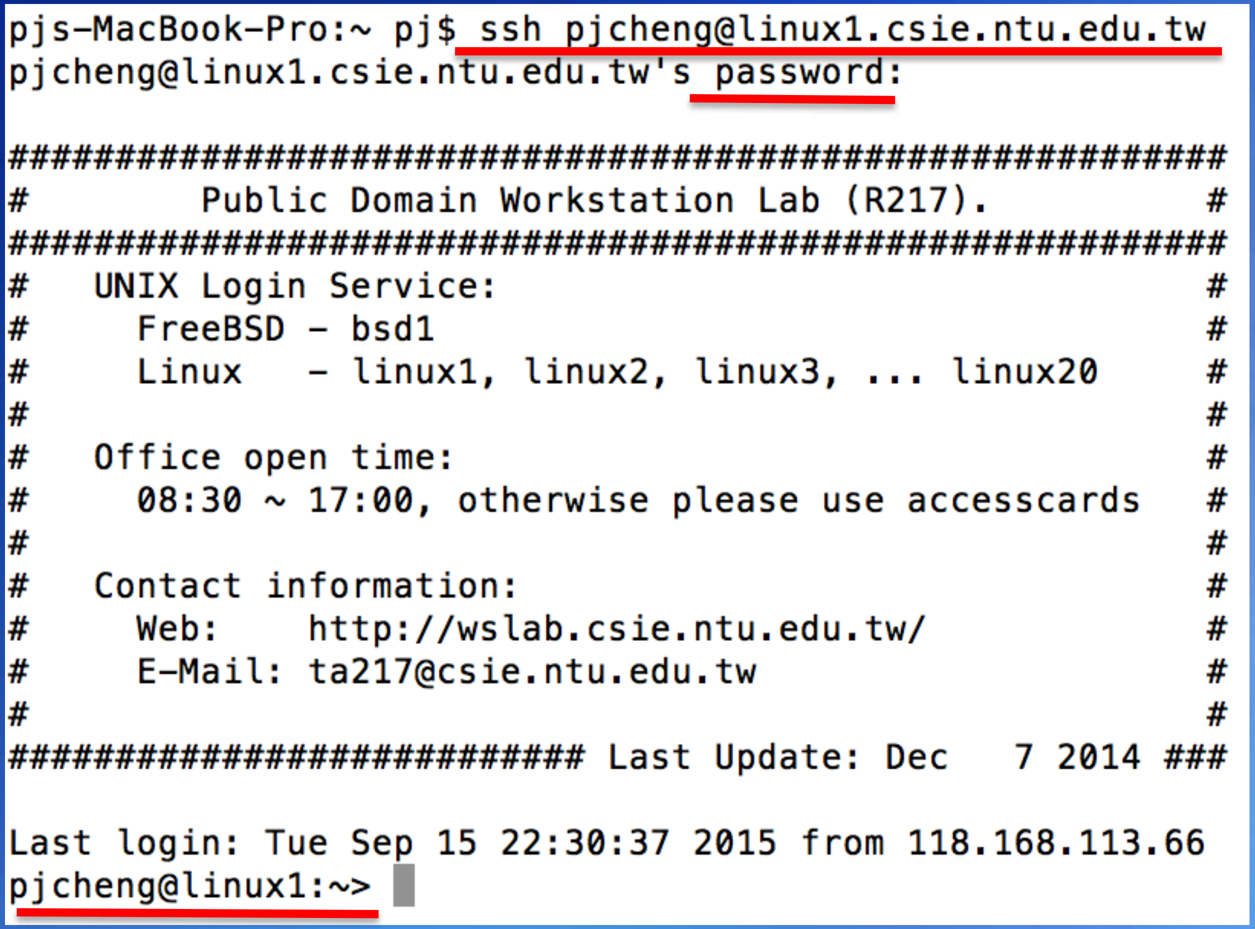
Application program can request service from kernel
Portable OS layer

Bootstrap, system initialization, interrupt and exception, process, memory &I/O management

Provided pre-compiled object codes
Defined in headers (e.g., stdio.h)

# Interactive Interface to Unix

OpenSSH SSH client (remote login program)

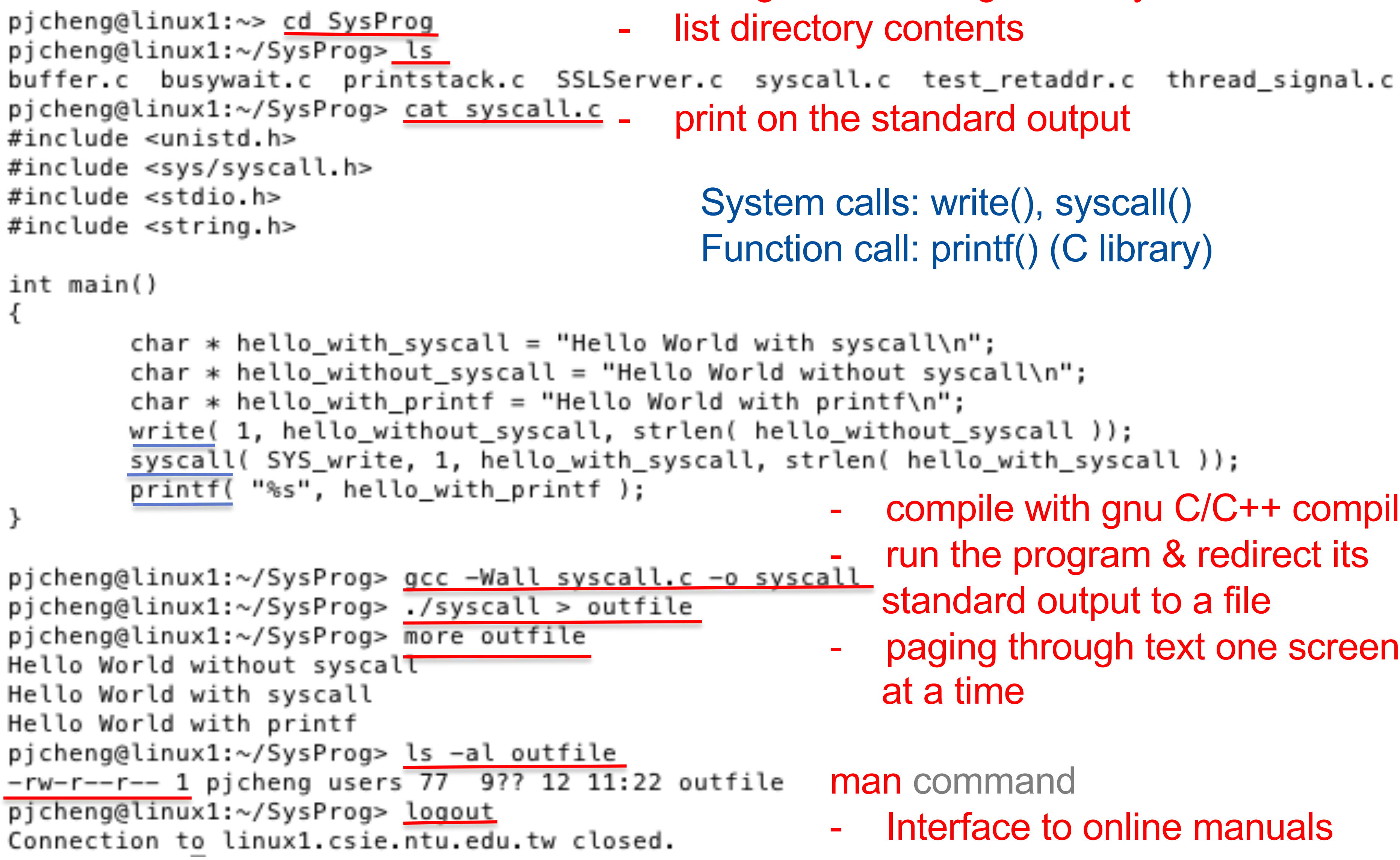


```
pjs-MacBook-Pro:~ pj$ ssh pjcheng@linux1.csie.ntu.edu.tw
pjcheng@linux1.csie.ntu.edu.tw's password:

#########################################################
#        Public Domain Workstation Lab (R217).          #
#########################################################
#   UNIX Login Service:                                 #
#     FreeBSD - bsd1                                    #
#     Linux   - linux1, linux2, linux3, ... linux20     #
#                                                       #
#   Office open time:                                   #
#     08:30 ~ 17:00, otherwise please use accesscards   #
#                                                       #
#   Contact information:                                #
#     Web:    http://wslab.csie.ntu.edu.tw/             #
#     E-Mail: ta217@csie.ntu.edu.tw                     #
#                                                       #
########################### Last Update: Dec   7 2014 ###

Last login: Tue Sep 15 22:30:37 2015 from 118.168.113.66
pjcheng@linux1:~>
```

shell prompt: where you type commands

- change the working directory
- list directory contents
- print on the standard output

System calls: write(), syscall()
Function call: printf() (C library)



```
pjcheng@linux1:~> cd SysProg
pjcheng@linux1:~/SysProg> ls
buffer.c  busywait.c  printstack.c  SSLServer.c  syscall.c  test_retaddr.c  thread_signal.c
pjcheng@linux1:~/SysProg> cat syscall.c
#include <unistd.h>
#include <sys/syscall.h>
#include <stdio.h>
#include <string.h>

int main()
{
        char * hello_with_syscall = "Hello World with syscall\n";
        char * hello_without_syscall = "Hello World without syscall\n";
        char * hello_with_printf = "Hello World with printf\n";
        write( 1, hello_without_syscall, strlen( hello_without_syscall ));
        syscall( SYS_write, 1, hello_with_syscall, strlen( hello_with_syscall ));
        printf( "%s", hello_with_printf );
}

pjcheng@linux1:~/SysProg> gcc -Wall syscall.c -o syscall
pjcheng@linux1:~/SysProg> ./syscall > outfile
pjcheng@linux1:~/SysProg> more outfile
Hello World without syscall
Hello World with syscall
Hello World with printf
pjcheng@linux1:~/SysProg> ls -al outfile
-rw-r--r-- 1 pjcheng users 77  9?? 12 11:22 outfile
pjcheng@linux1:~/SysProg> logout
Connection to linux1.csie.ntu.edu.tw closed.
```

- compile with gnu C/C++ compile
- run the program & redirect its standard output to a file
- paging through text one screenfu at a time

man command

- Interface to online manuals

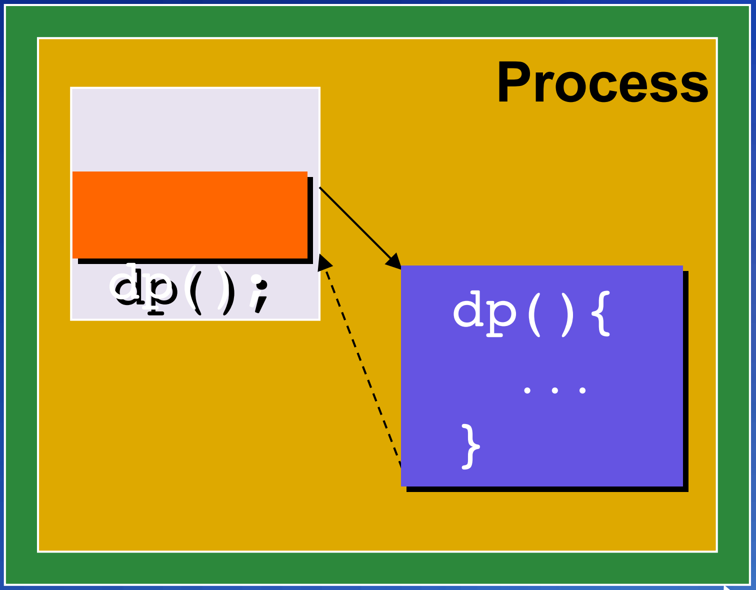
A material from Stanford (source)

Compile & link program, gcc, make, gdb, shell commands

# System Calls vs. Function Calls

- **System Call:** a request to the operating system to perform some activity

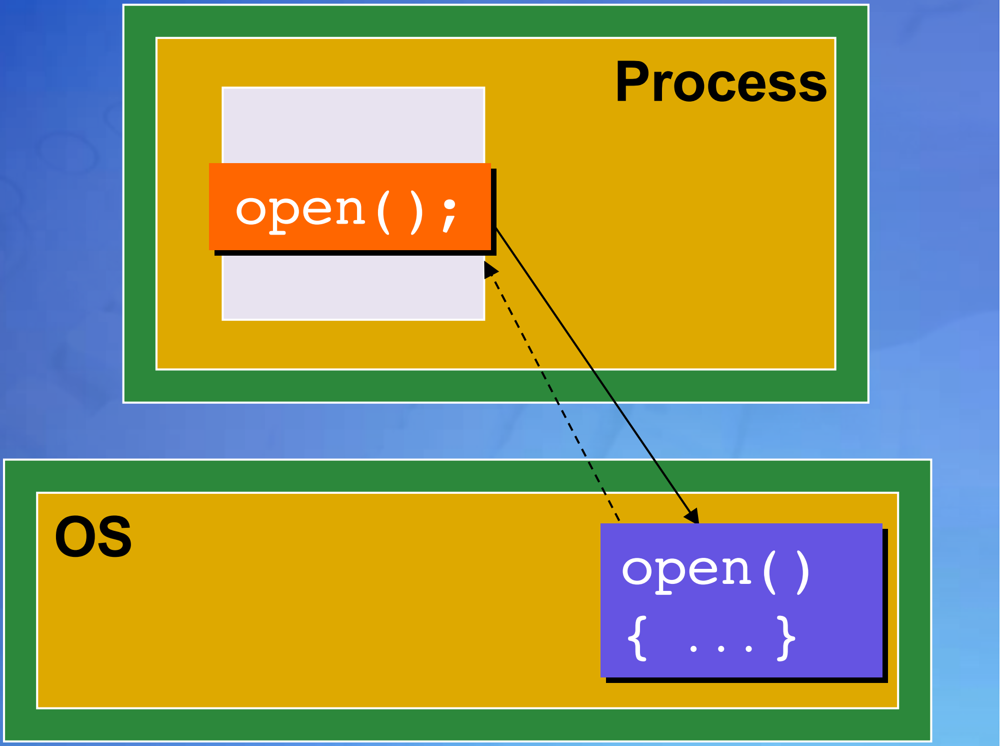
Function Call



Caller and callee are in the same process

- Same user
- Same "domain of trust"

System Call



- OS is trusted; user is not.
- OS has super-privileges; user does not
- Must take measures to prevent abuse

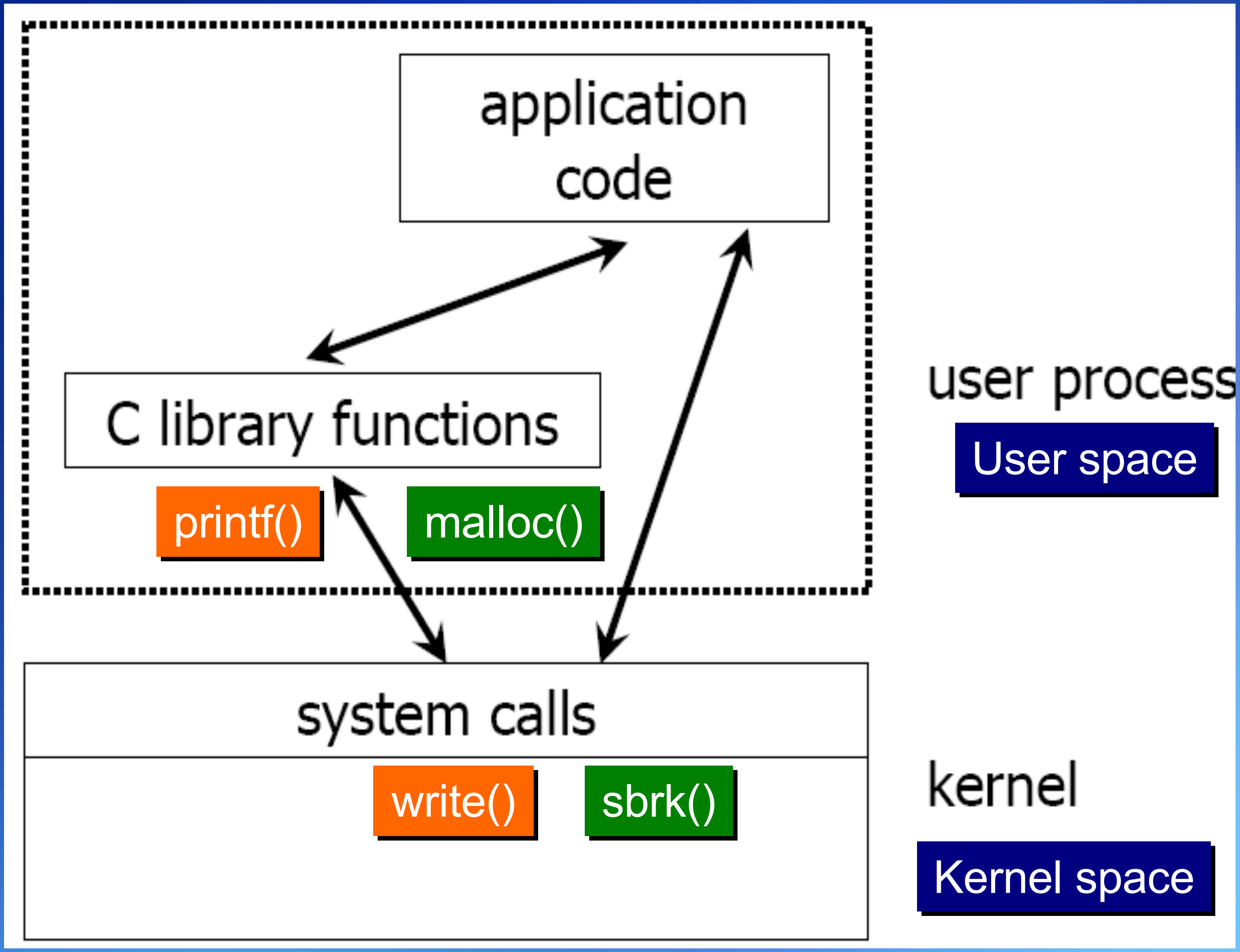
# User Mode vs. Kernel Mode

- **Modes of Execution (for protection)**
  - User mode vs. kernel mode
- **Most CPUs support at least two modes of execution: privileged (kernel-mode) and non-privileged (user-mode)**
- **User mode: a non-privileged mode in which processes are forbidden to access those portions of memory that have been allocated to the kernel or to other programs.**
- **When a user mode process wants to use a service that is provided by the kernel (e.g. a system call), the system must switch temporarily into kernel mode.**

# User Space vs. Kernel Space

- **System memory is divided into two parts**
  - User space
    - a process executing in user mode is executing in user space
    - each user process is protected (isolated) from another (except for shared memory segments and mmapings, which will be discussed later)
  - Kernel space
    - a process executing in kernel mode is executing in kernel space
- **Kernel space is the area wherein the kernel executes**

  **user space is the area where a user program normally executes, *except when it performs a system call.***



application code

C library functions

printf()

malloc()

user process

User space

system calls

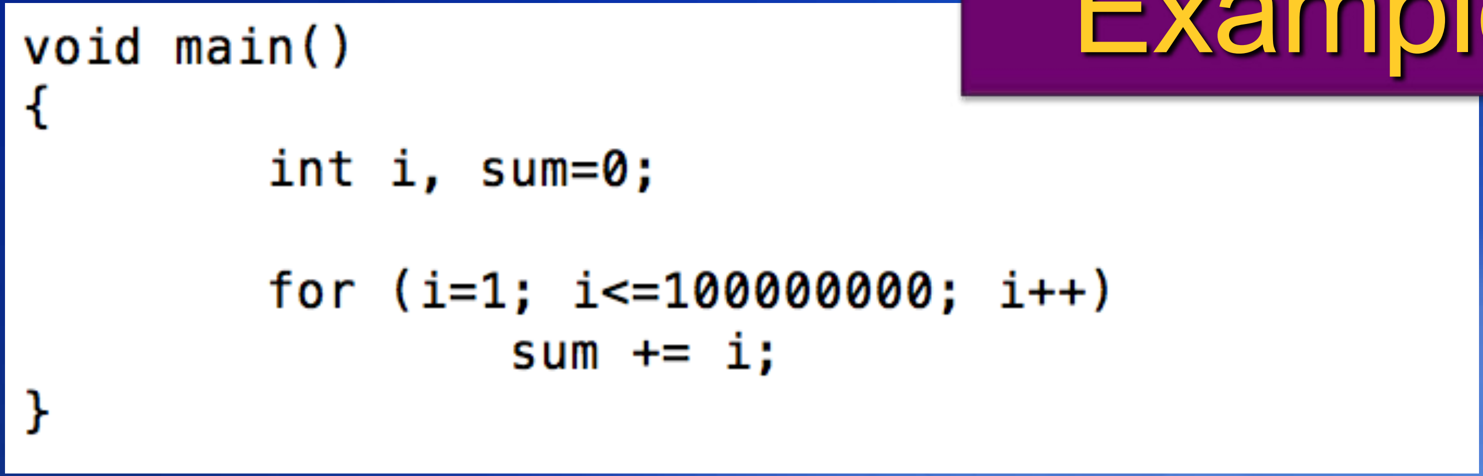
write()

sbrk()

kernel

Kernel space

# Example 1



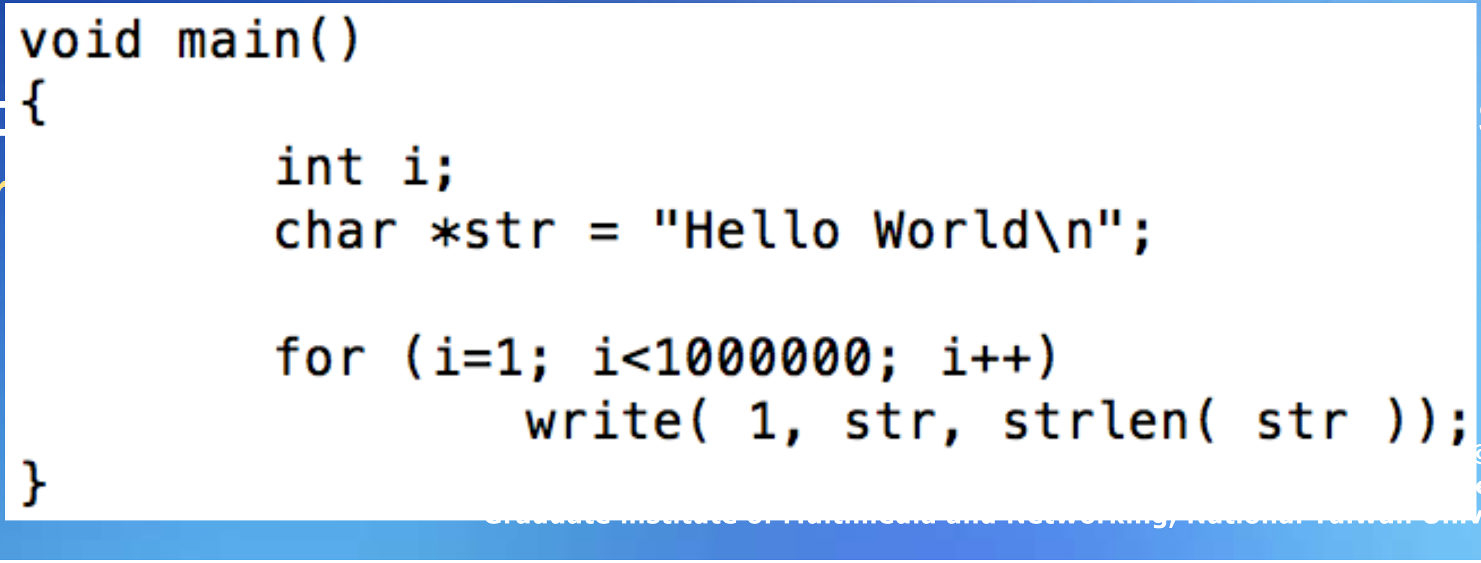
```
void main()
{
        int i, sum=0;

        for (i=1; i<=100000000; i++)
                sum += i;
}
```

Shell command: time –p execfile

(time: run execfile & summarize time usage)

A) real 0.43, user 0.42, sys 0.00 (in seconds) ? or

B) real 2.99, user 0.04, sys 1.32 (in seconds) ?



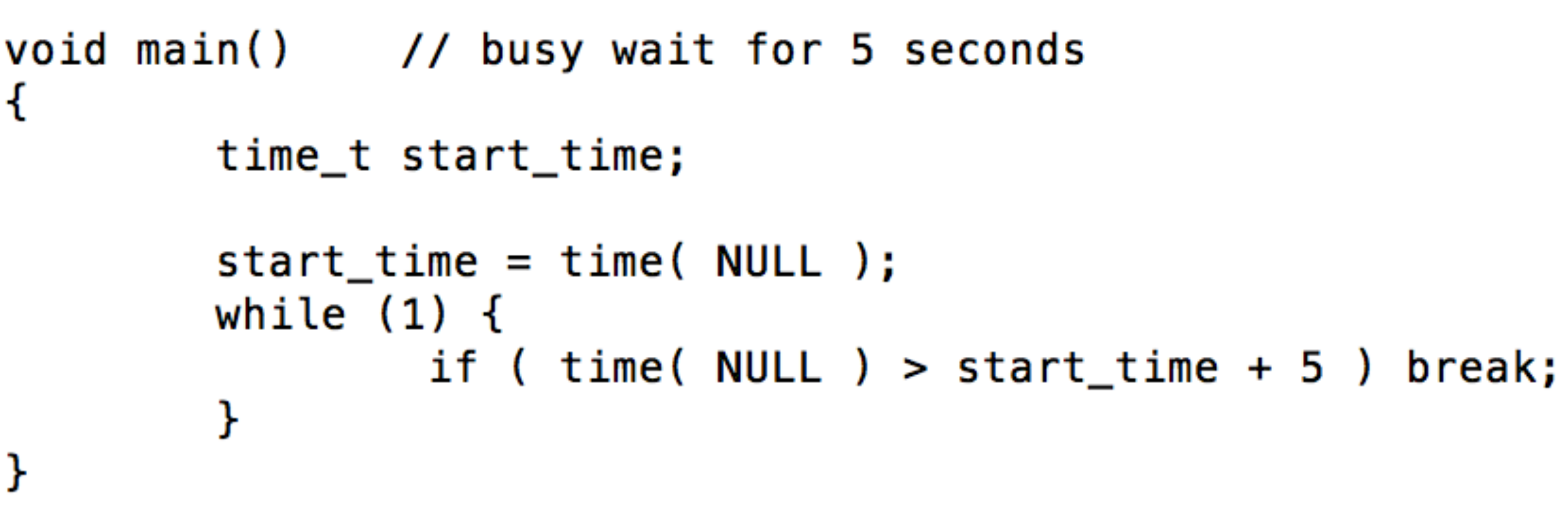
```
void main()
{
        int i;
        char *str = "Hello World\n";

        for (i=1; i<1000000; i++)
                write( 1, str, strlen( str ));
}
```

real: s

user

sys:

# Example 2



```
void main()    // busy wait for 5 seconds
{
        time_t start_time;

        start_time = time( NULL );
        while (1) {
                if ( time( NULL ) > start_time + 5 ) break;
        }
}
```

real 5.60, user 5.50, sys 0.00        (bad)

Comparison:  time –p sleep 5

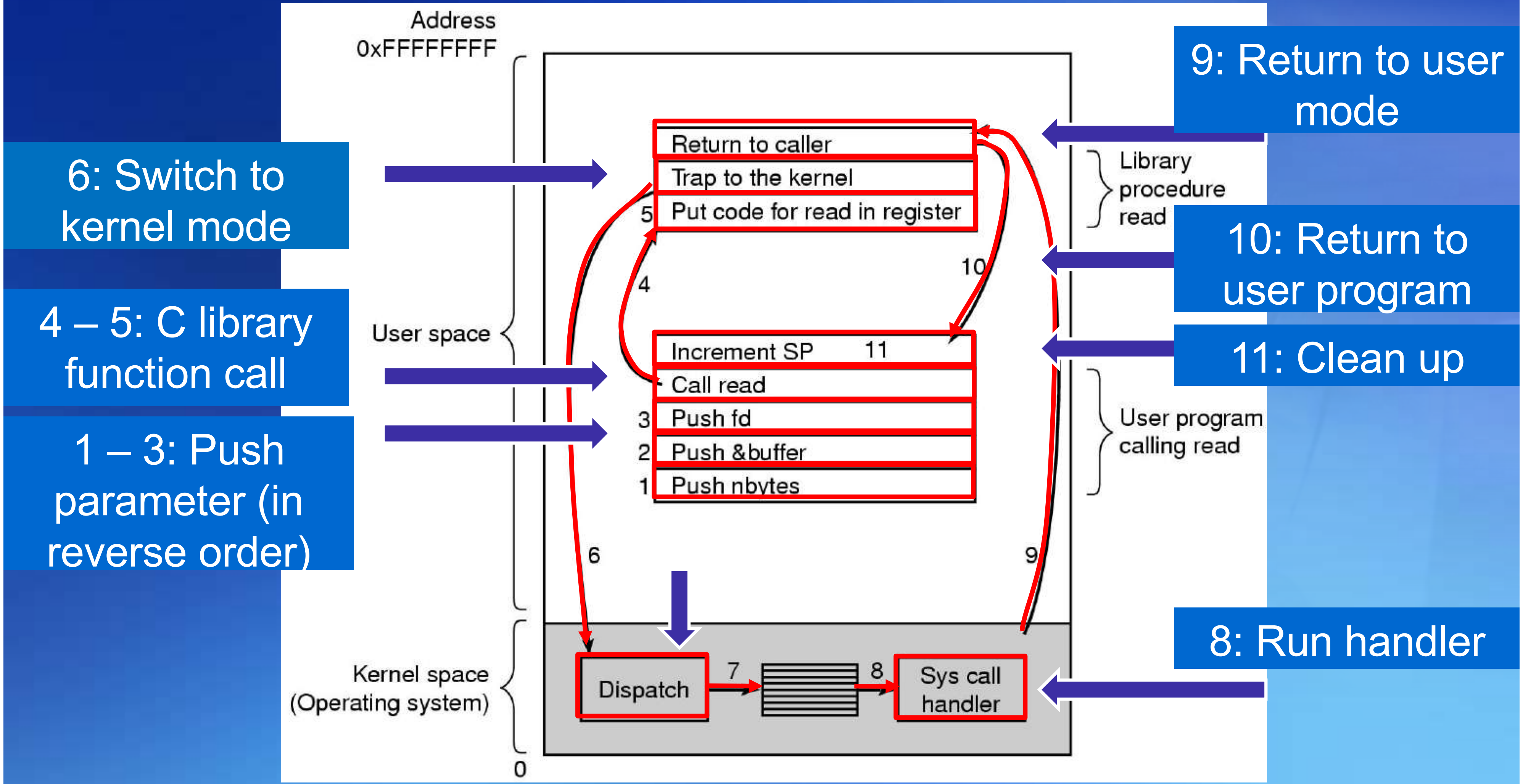
real 5.00, user 0.00, sys 0.00        (good)

(sleep: delay for a specified amount of time)

# Steps for Making a System Call

Making a system call is expensive
Example: **read (fd, buffer, nbytes)**

(Reading: concept of system call)



6: Switch to kernel mode

4 – 5: C library function call

1 – 3: Push parameter (in reverse order)

9: Return to user mode

10: Return to user program

11: Clean up

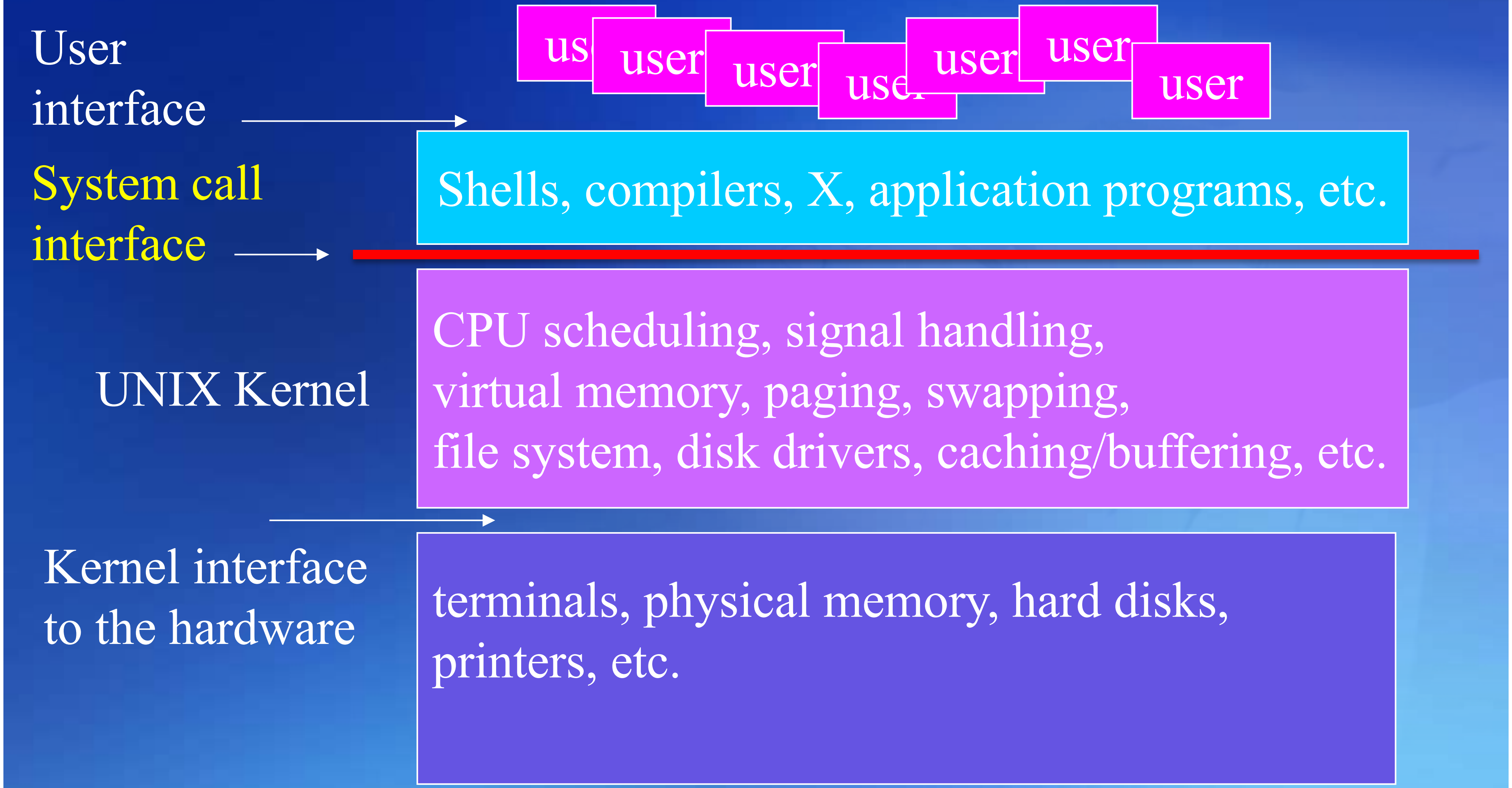
8: Run handler

See also: long syscall( long number, … )

# What is an Operating System

- It is an extended machine (vertical)
  - Presents user with a virtual machine, easier to use (establishes a user interface)
  - Hides the messy details which must be performed (executes and provides services safely)
- It is a resource manager (horizontal)
  - Resources: CPU, memory, I/O devices (disks, printers)
  - Each program gets time with the resource
  - Each program gets space on the resource
  - OS makes sure programs have a fair use

# UNIX Architecture



us user user use user user user

User interface

Shells, compilers, X, application programs, etc.

System call interface

UNIX Kernel

CPU scheduling, signal handling, virtual memory, paging, swapping, file system, disk drivers, caching/buffering, etc.

Kernel interface to the hardware

terminals, physical memory, hard disks, printers, etc.

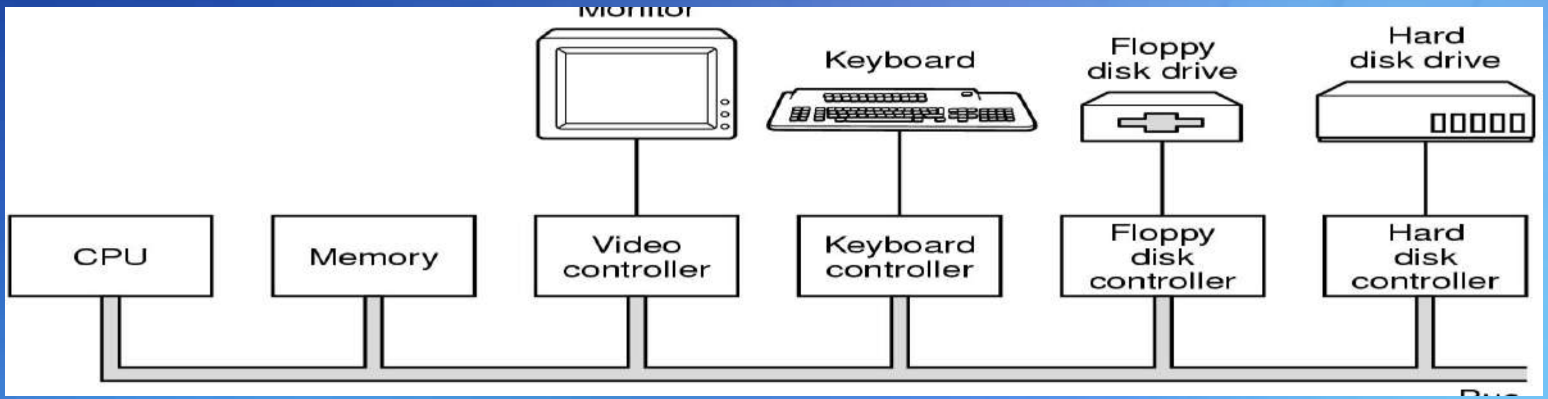
# Resource Manager (horizontal viewpoint)

Services for Application Programs:

User Identification, Process Management, Memory Management, File/Directory Management, Inter-process Communication, Signal, I/O Management (e.g., Terminal, Network, etc), …

OS Kernel:

CPU Scheduling, Virtual Memory, File System, Protection, Security, Synchronization, I/O Control, …



Example of hardware components for a PC

# Services for Application Programs

Chapter 1

- User Identification
- Process Management
- Memory Management
- File/Directory Management

# Services for Application Programs

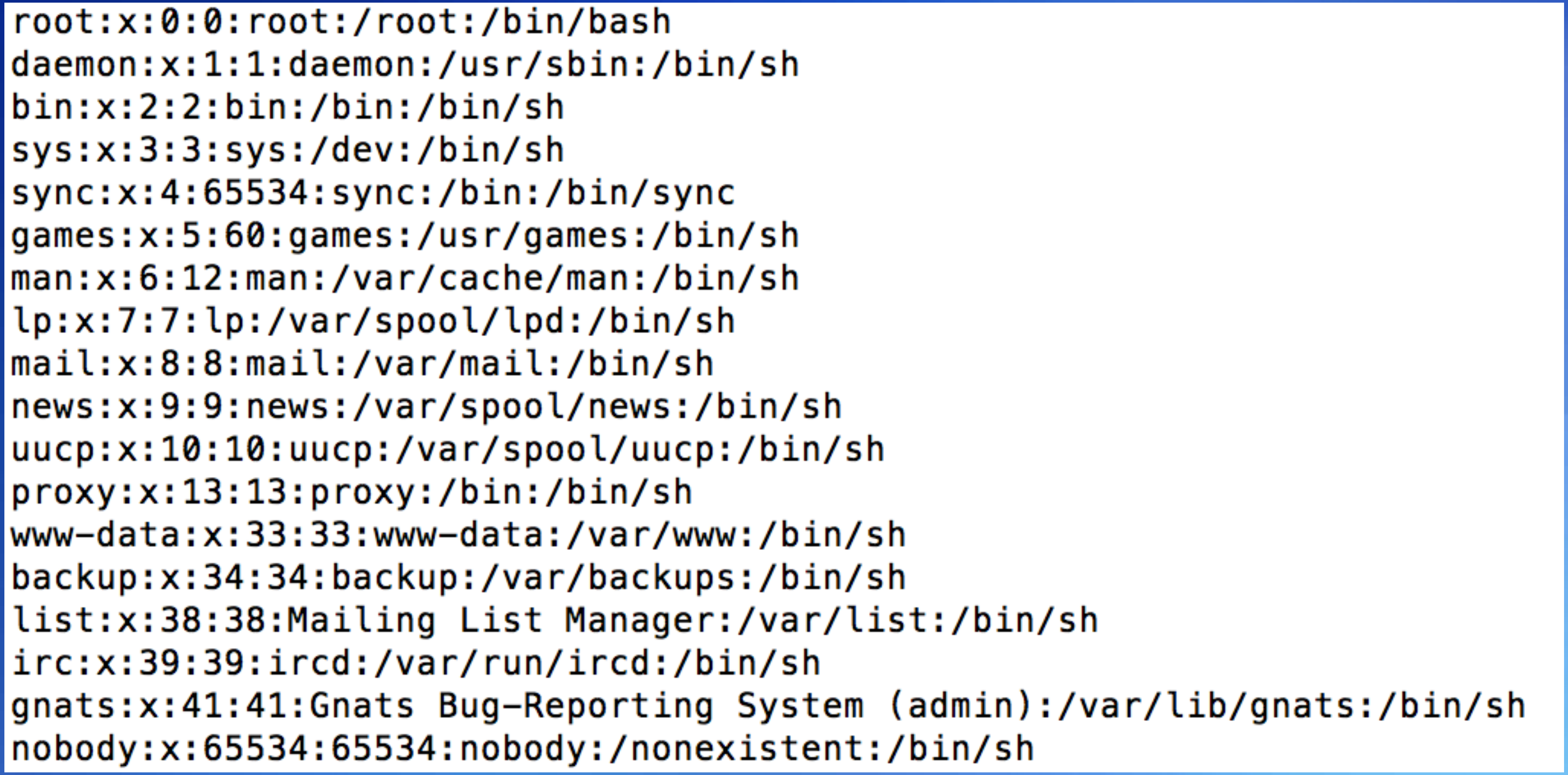
- User Identification
- Process Management
- Memory Management
- File/Directory Management

# User Identification in UNIX

- **Logging In (See Ch1.3, 6.2, 6.3)**
  - /etc/passwd – local machine or NIS DB
    - root:x:0:1:Super-User:/root:/bin/tcsh
    - Login-name, encrypted passwd, numeric user-ID, numeric group ID, comment, home dir, shell program
  - /etc/shadow – with "x" indicated for passwd
- **Related shell command:**

  passwd (i.e., change user password)

## Shell command: cat /etc/passwd



```
root:x:0:0:root:/root:/bin/bash
daemon:x:1:1:daemon:/usr/sbin:/bin/sh
bin:x:2:2:bin:/bin:/bin/sh
sys:x:3:3:sys:/dev:/bin/sh
sync:x:4:65534:sync:/bin:/bin/sync
games:x:5:60:games:/usr/games:/bin/sh
man:x:6:12:man:/var/cache/man:/bin/sh
lp:x:7:7:lp:/var/spool/lpd:/bin/sh
mail:x:8:8:mail:/var/mail:/bin/sh
news:x:9:9:news:/var/spool/news:/bin/sh
uucp:x:10:10:uucp:/var/spool/uucp:/bin/sh
proxy:x:13:13:proxy:/bin:/bin/sh
www-data:x:33:33:www-data:/var/www:/bin/sh
backup:x:34:34:backup:/var/backups:/bin/sh
list:x:38:38:Mailing List Manager:/var/list:/bin/sh
irc:x:39:39:ircd:/var/run/ircd:/bin/sh
gnats:x:41:41:Gnats Bug-Reporting System (admin):/var/lib/gnats:/bin/sh
nobody:x:65534:65534:nobody:/nonexistent:/bin/sh
```

# Password Encryption and Salt

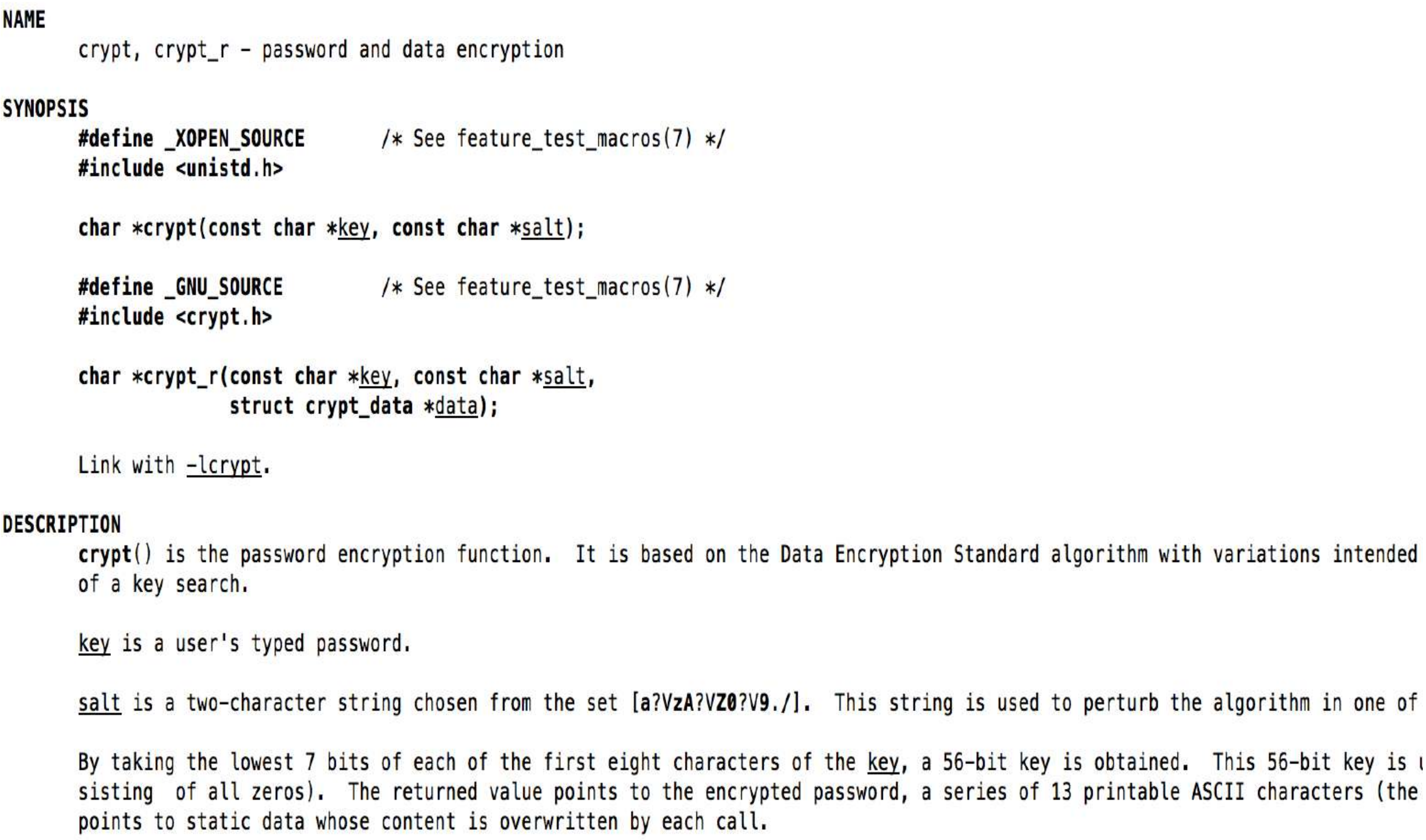
- *crypt(3)* is designed to make a key search computationally expensive
- /bin/passwd selects a salt based on the time of day
- Salt is converted into a two character string and stored in the encrypted password file

```
crypt( "apple", "am" ) = "amADBMNoVAZpc"
crypt( "water", "pm" ) = "pmRarHxhhU34U"
```

- Salt allows users to use the same password on different computers

# Shell command: man 3 crypt

man: an interface to the
on-line reference manuals



```
NAME
       crypt, crypt_r - password and data encryption

SYNOPSIS
       #define _XOPEN_SOURCE       /* See feature_test_macros(7) */
       #include <unistd.h>

       char *crypt(const char *key, const char *salt);

       #define _GNU_SOURCE         /* See feature_test_macros(7) */
       #include <crypt.h>

       char *crypt_r(const char *key, const char *salt,
                     struct crypt_data *data);

       Link with -lcrypt.

DESCRIPTION
       crypt() is the password encryption function.  It is based on the Data Encryption Standard algorithm with variations intended
       of a key search.

       key is a user's typed password.

       salt is a two-character string chosen from the set [a?VzA?VZ0?V9./].  This string is used to perturb the algorithm in one of

       By taking the lowest 7 bits of each of the first eight characters of the key, a 56-bit key is obtained.  This 56-bit key is
       sisting  of all zeros).  The returned value points to the encrypted password, a series of 13 printable ASCII characters (the
       points to static data whose content is overwritten by each call.
```

# Services for Application Programs

- User Identification
- **Process Management**
- Memory Management
- File/Directory Management

# Programs and Processes

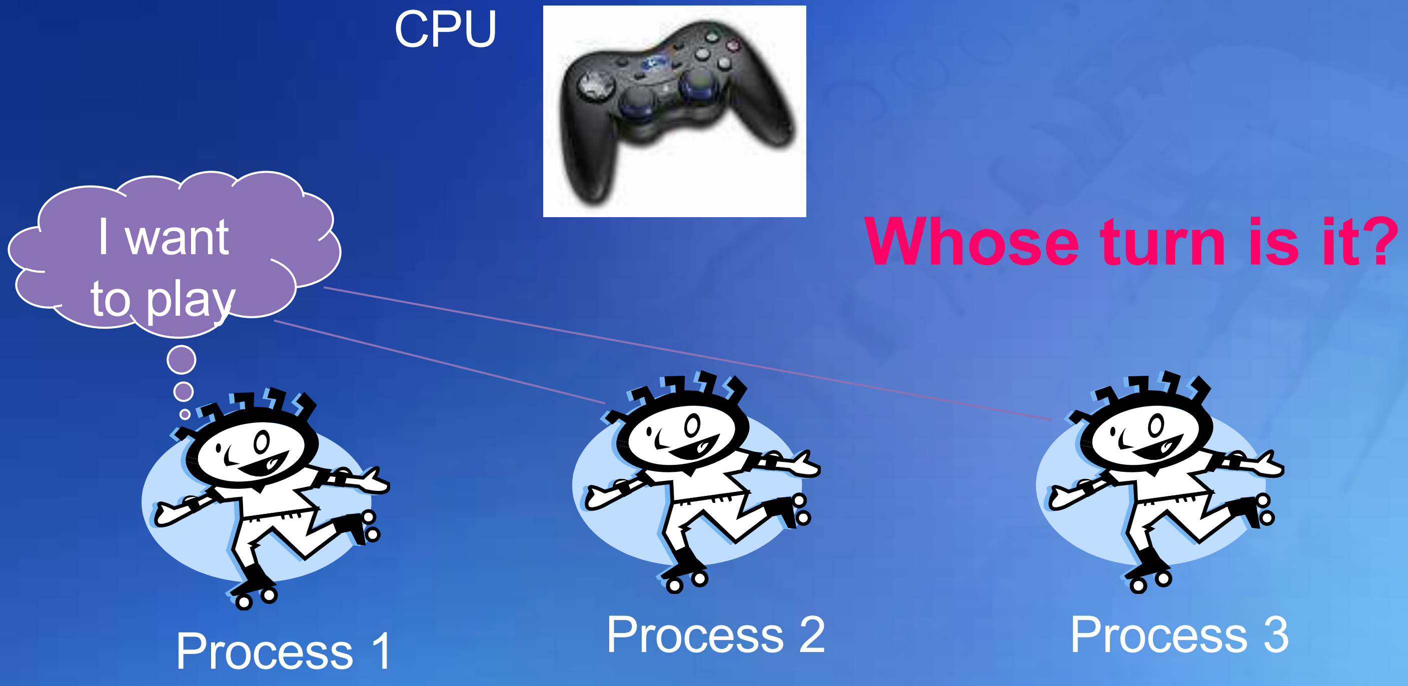
- Program
  - An executable file residing in a disk file
- Process
  - An executing instance of a program
  - Unique process ID
  - Related shell commands: ps, top

ps - report a snapshot of the current processes

top – display processes

# CPU Scheduling

- **Deciding which process should occupy the resource (CPU, disk, etc)**



CPU

I want to play

Whose turn is it?

Process 1

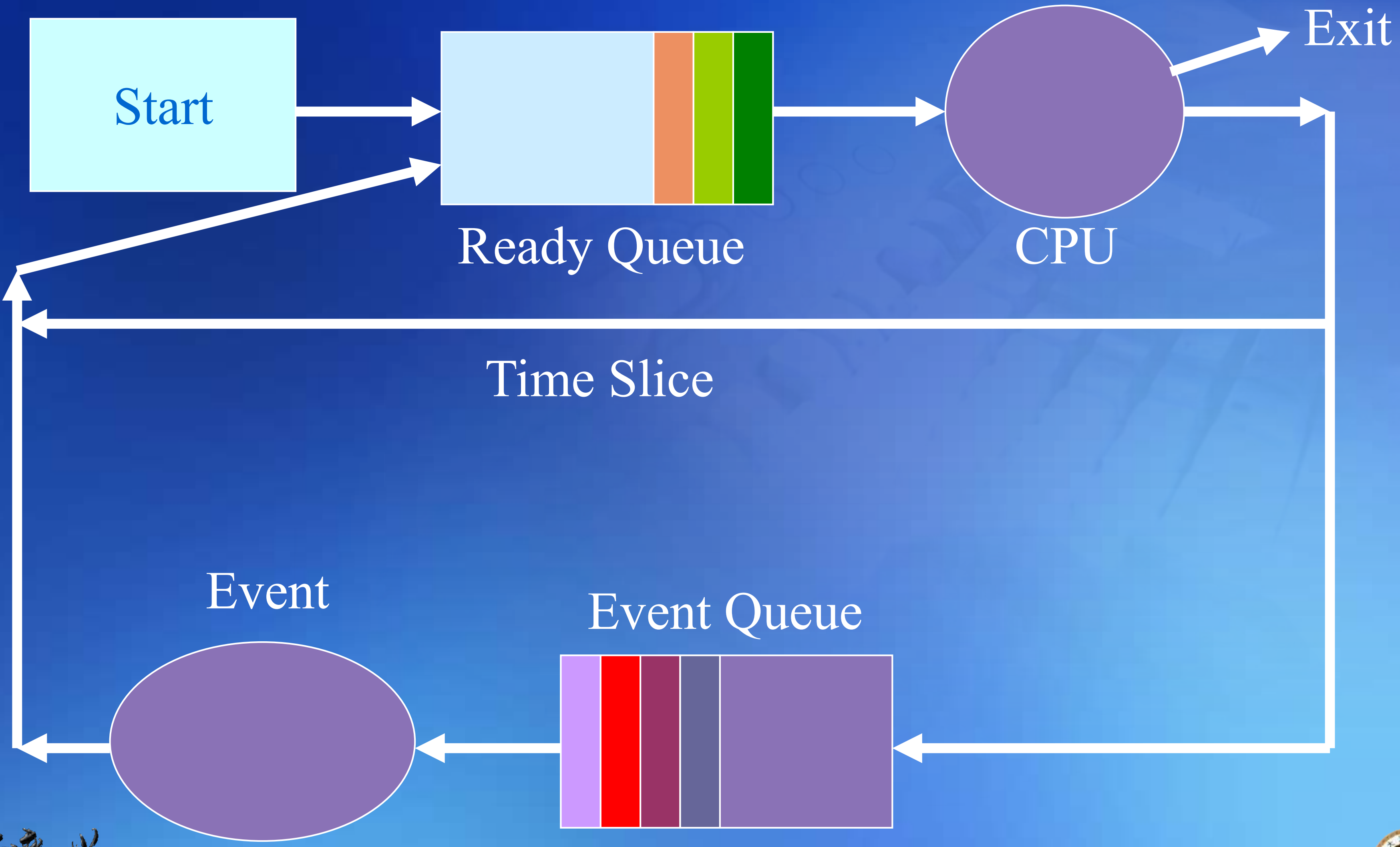
Process 2

Process 3

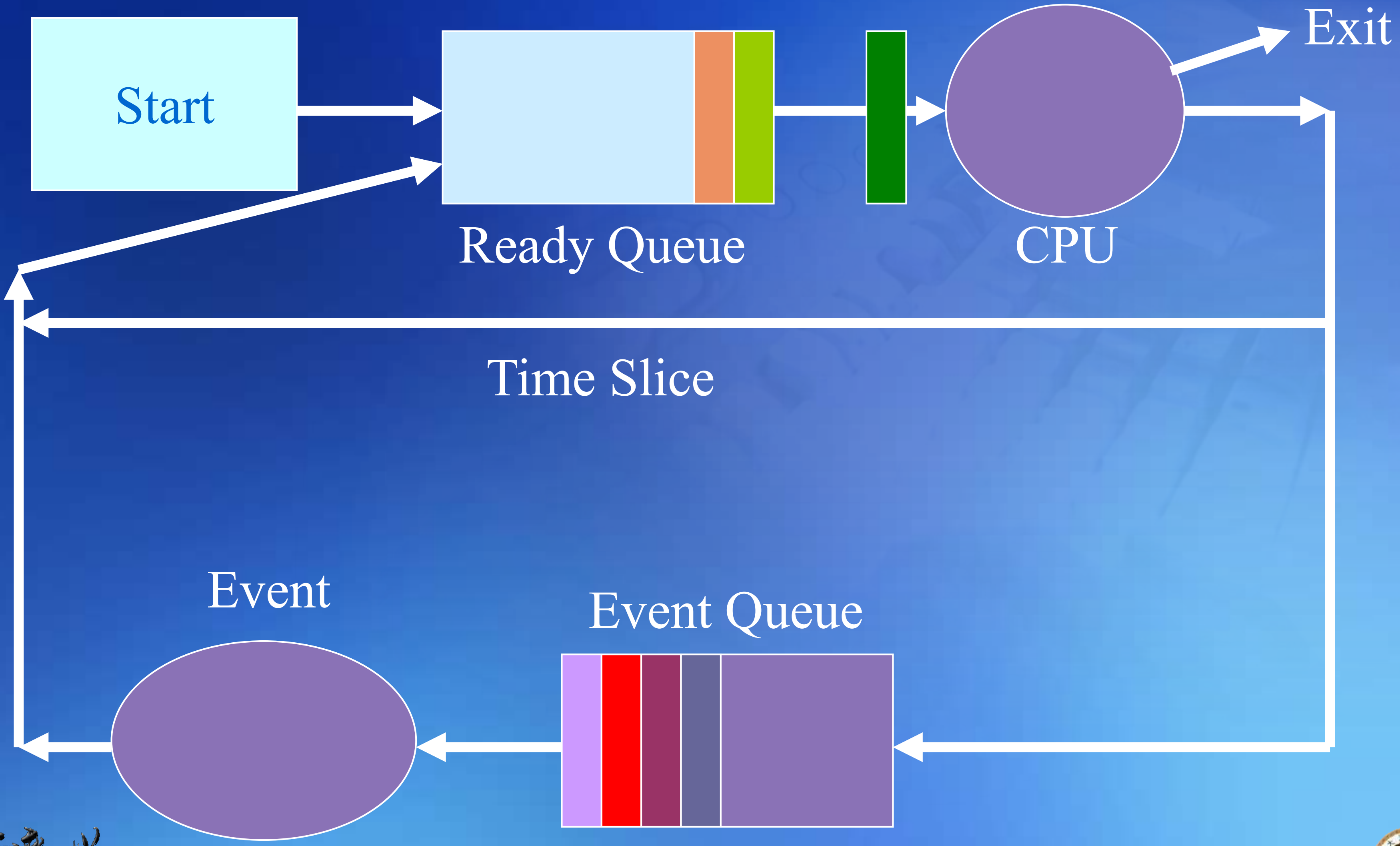
- **User can change a process priority**
  - Related shell command: nice

# Example of Multitasking

Time sharing: CPU's time is shared among multiple tasks simultaneously



# Example of Multitasking



Start

Ready Queue

CPU

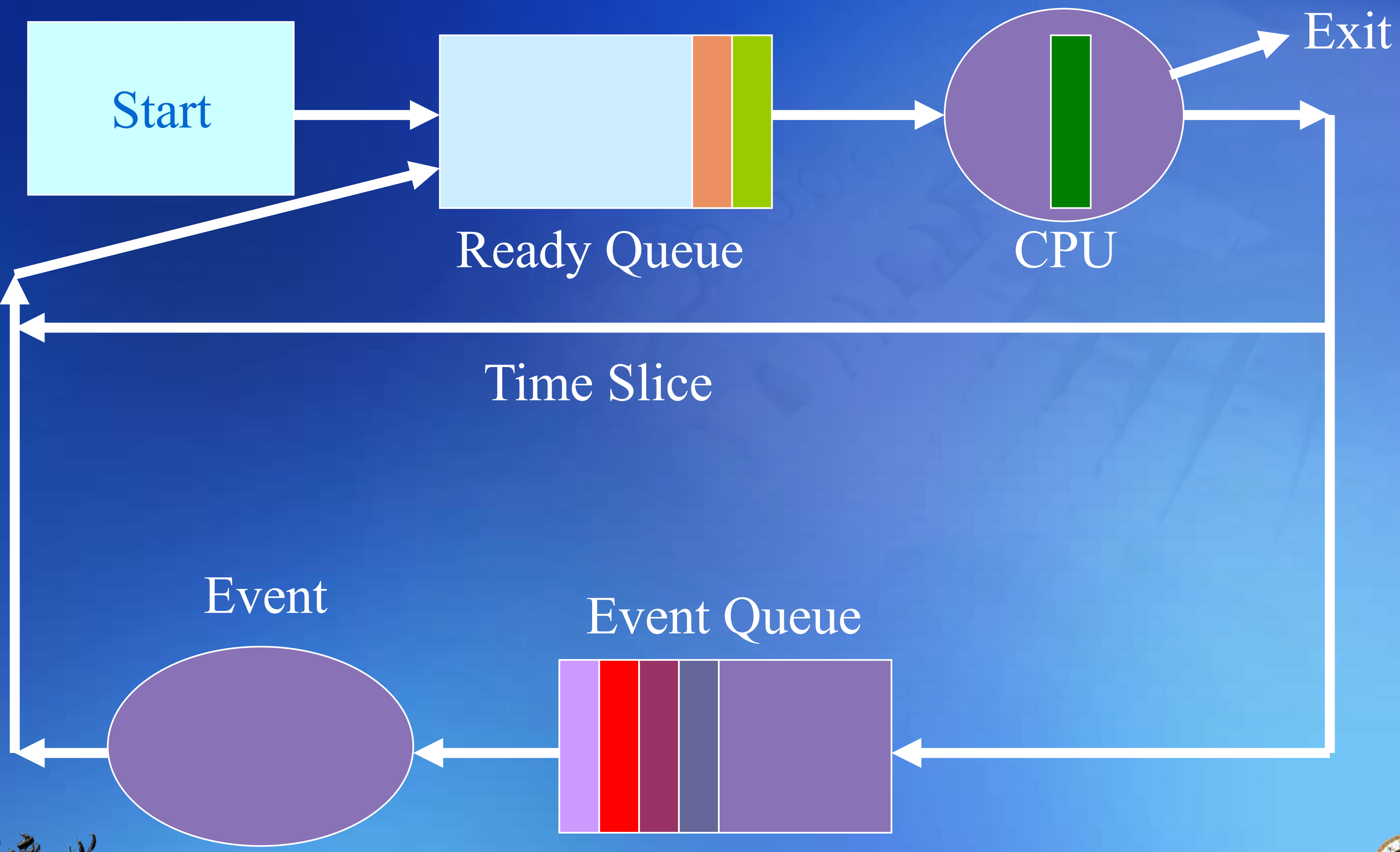
Exit

Time Slice

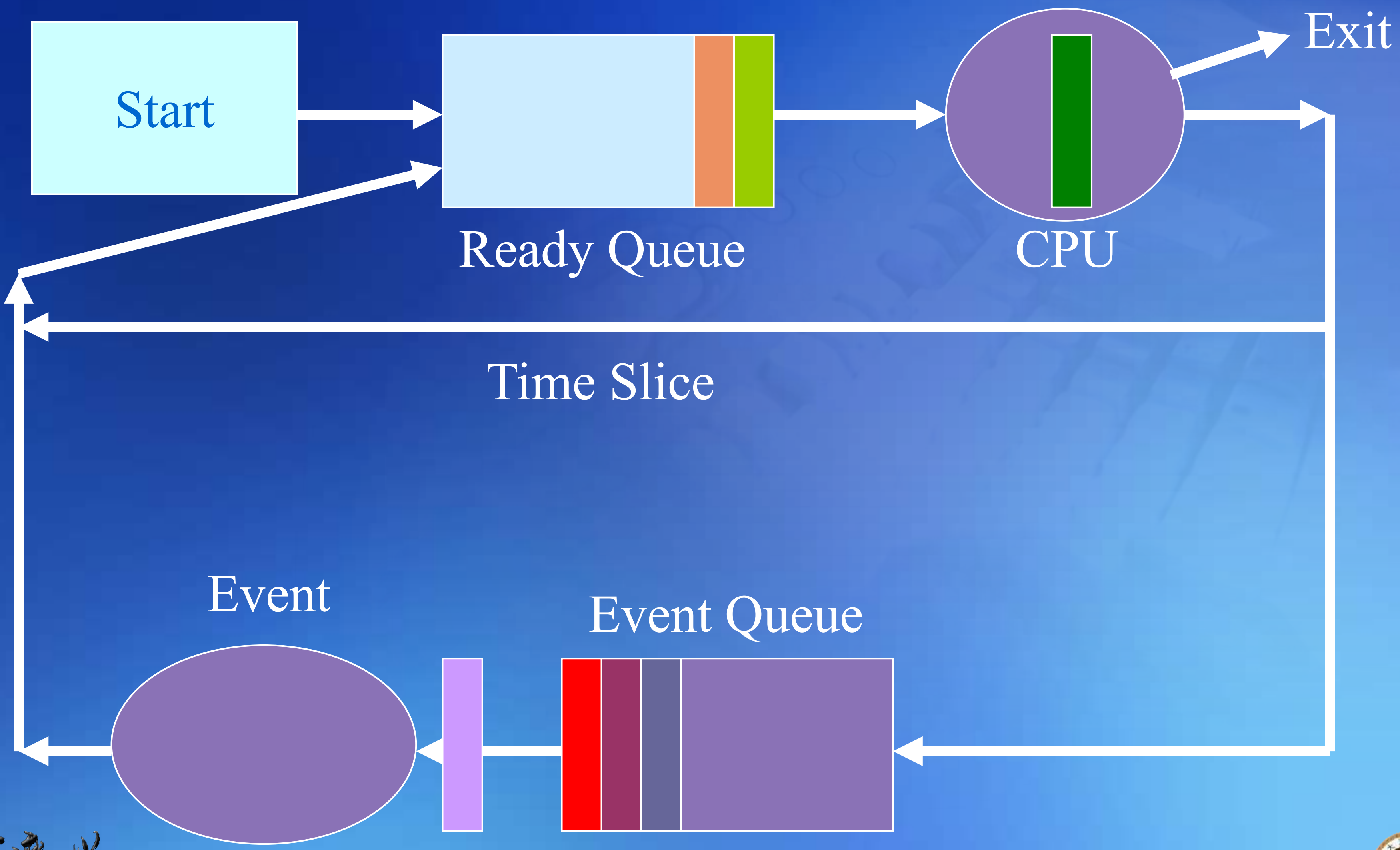
Event

Event Queue

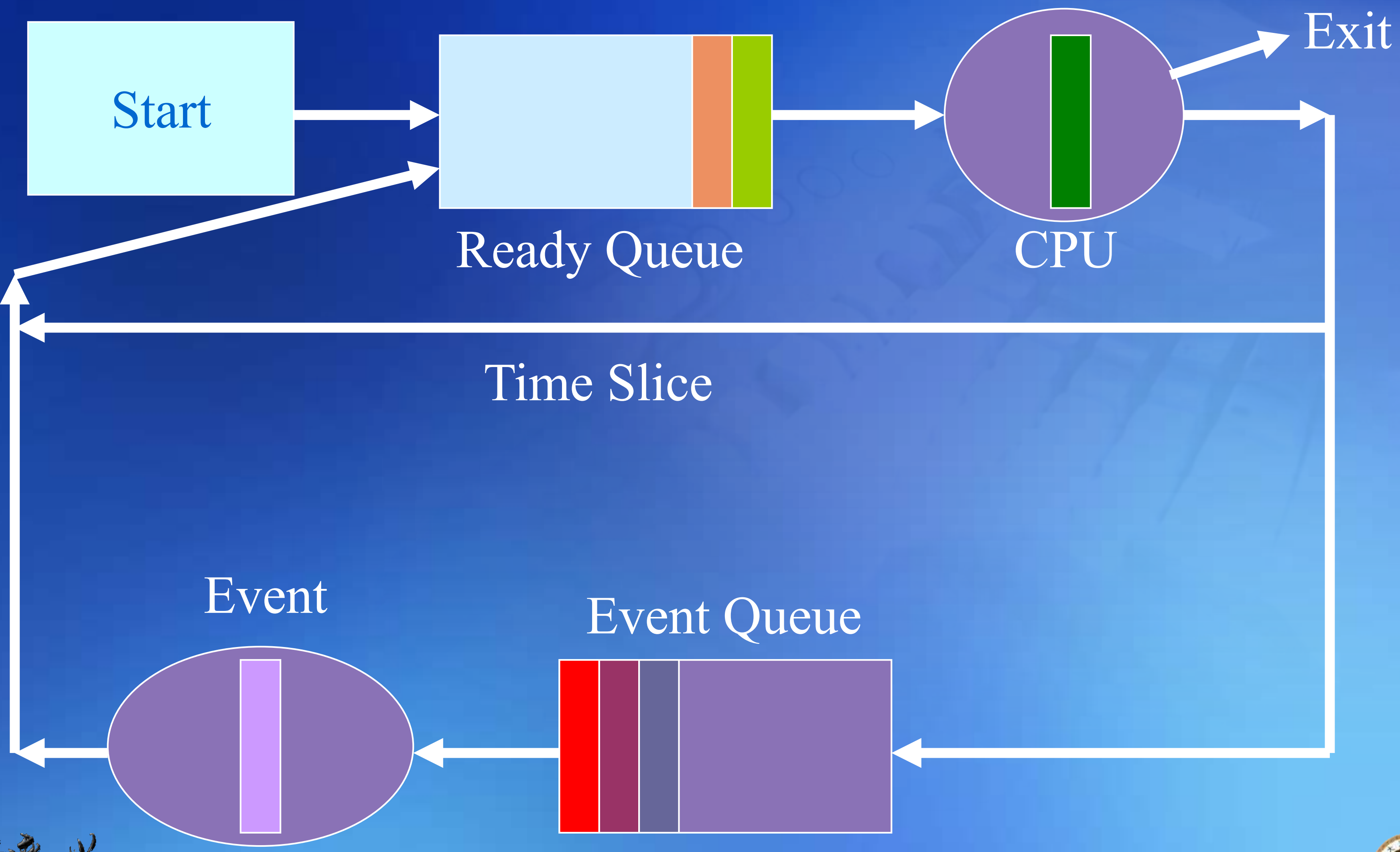
# Example of Multitasking



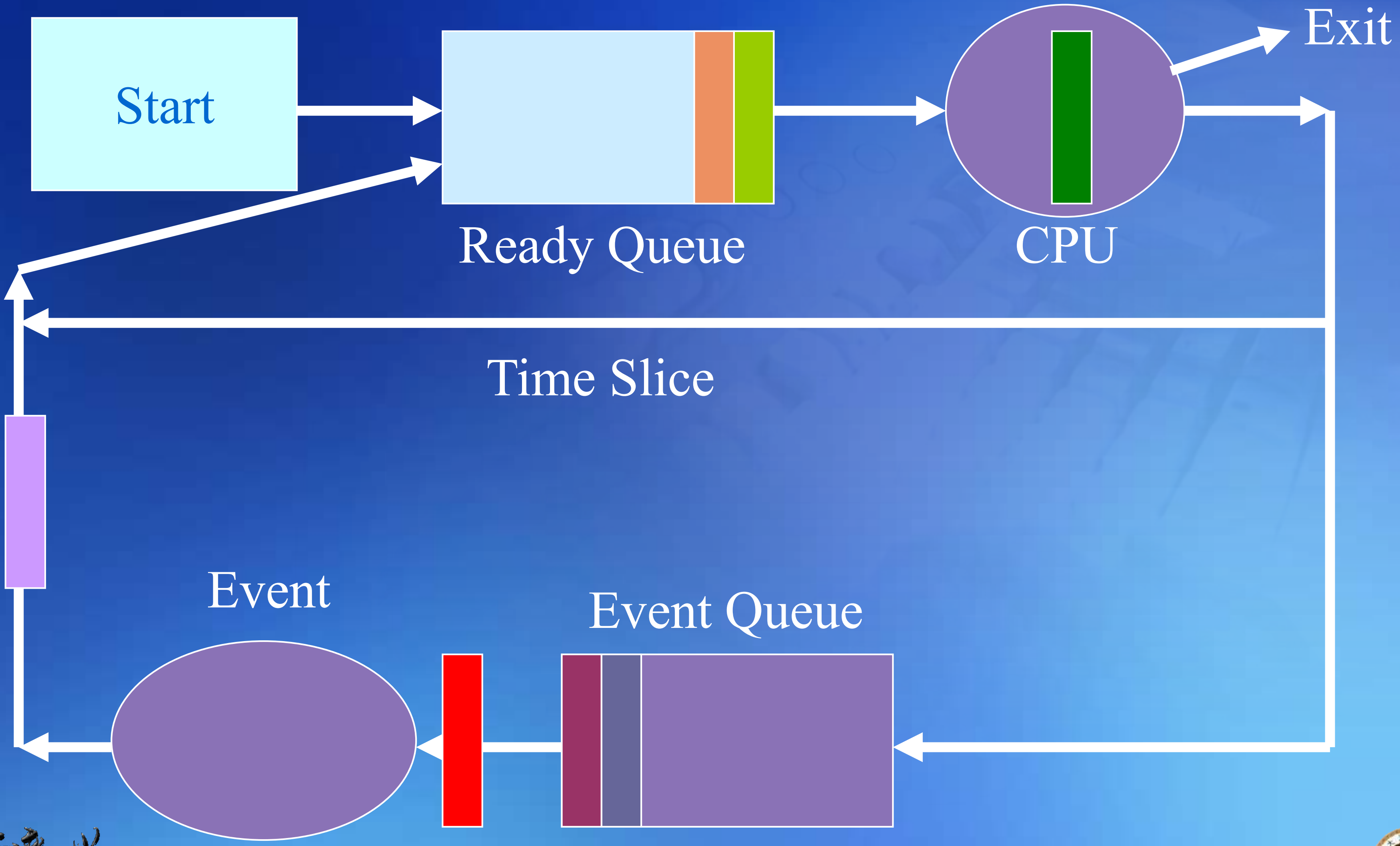
# Example of Multitasking



# Example of Multitasking

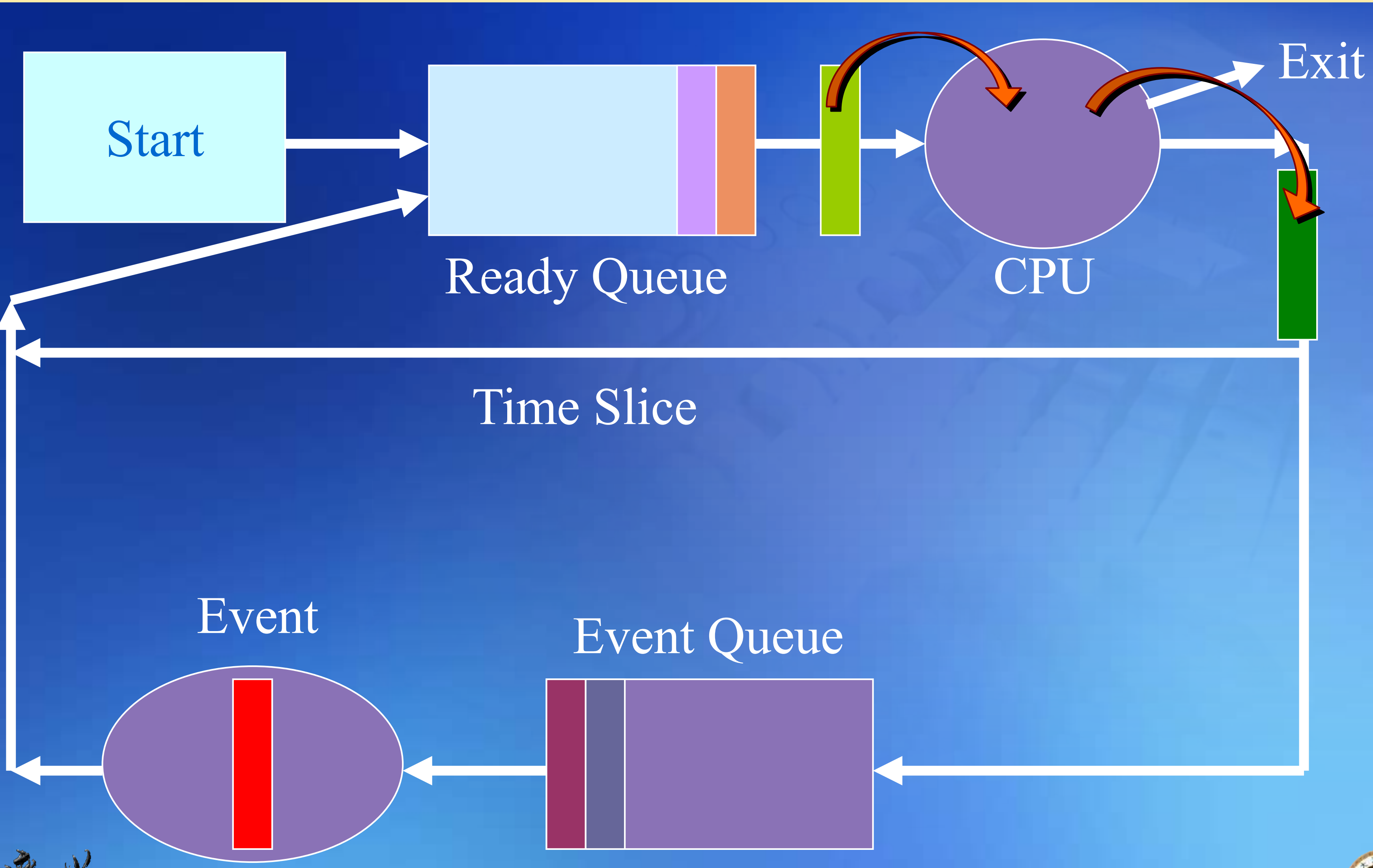


# Example of Multitasking

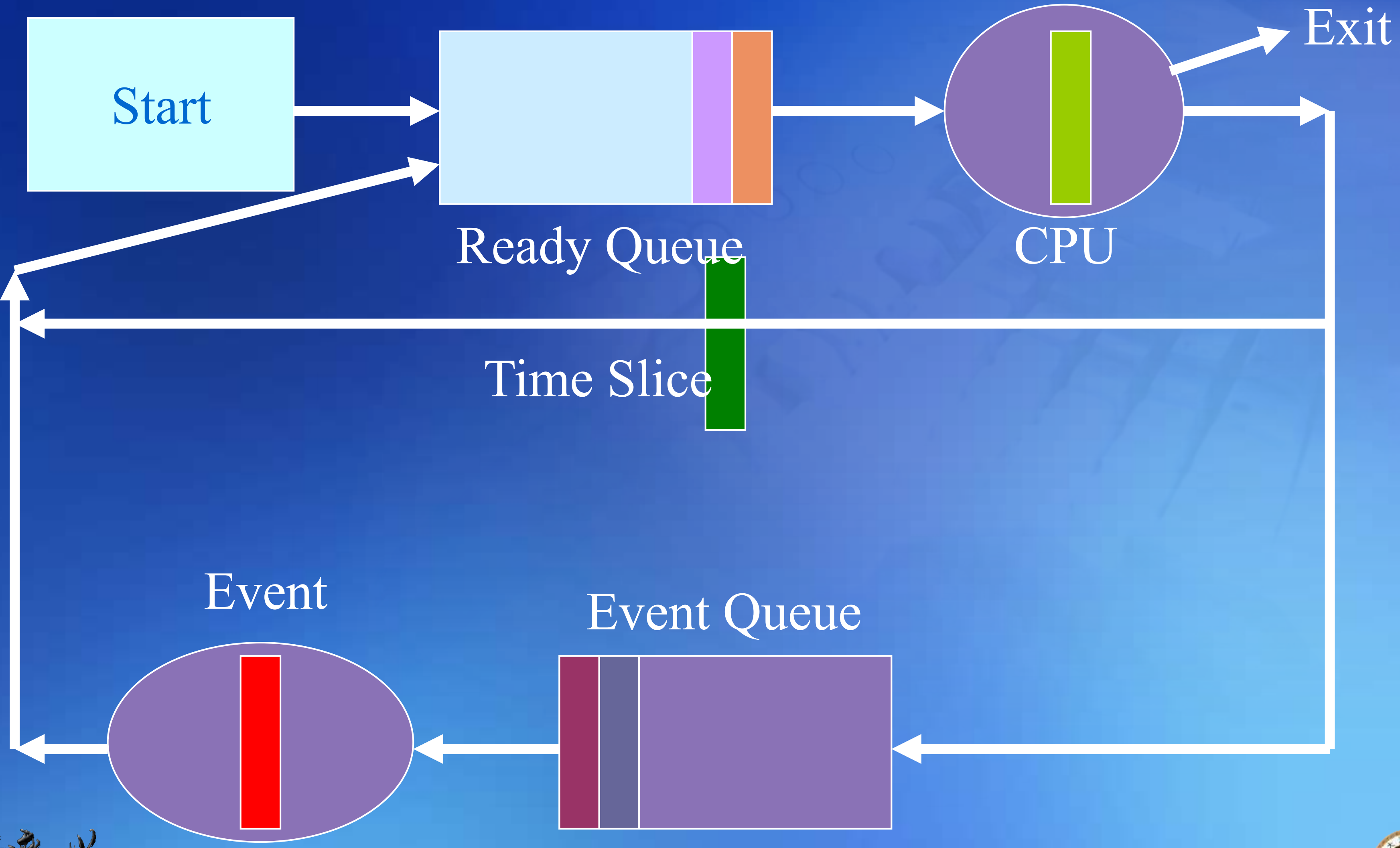


# Example of Multitasking

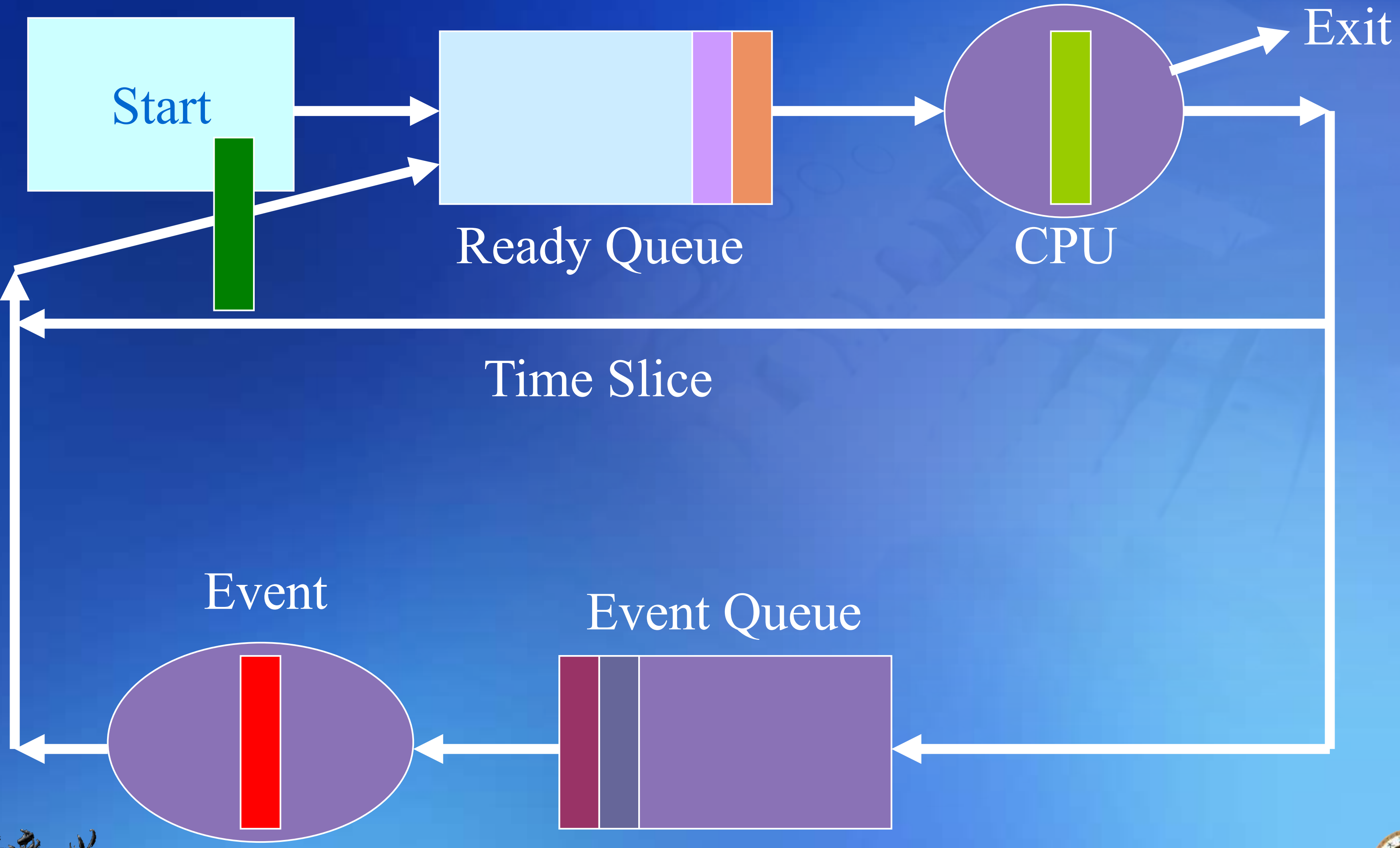
Context switch: process of storing and restoring execution context of a process



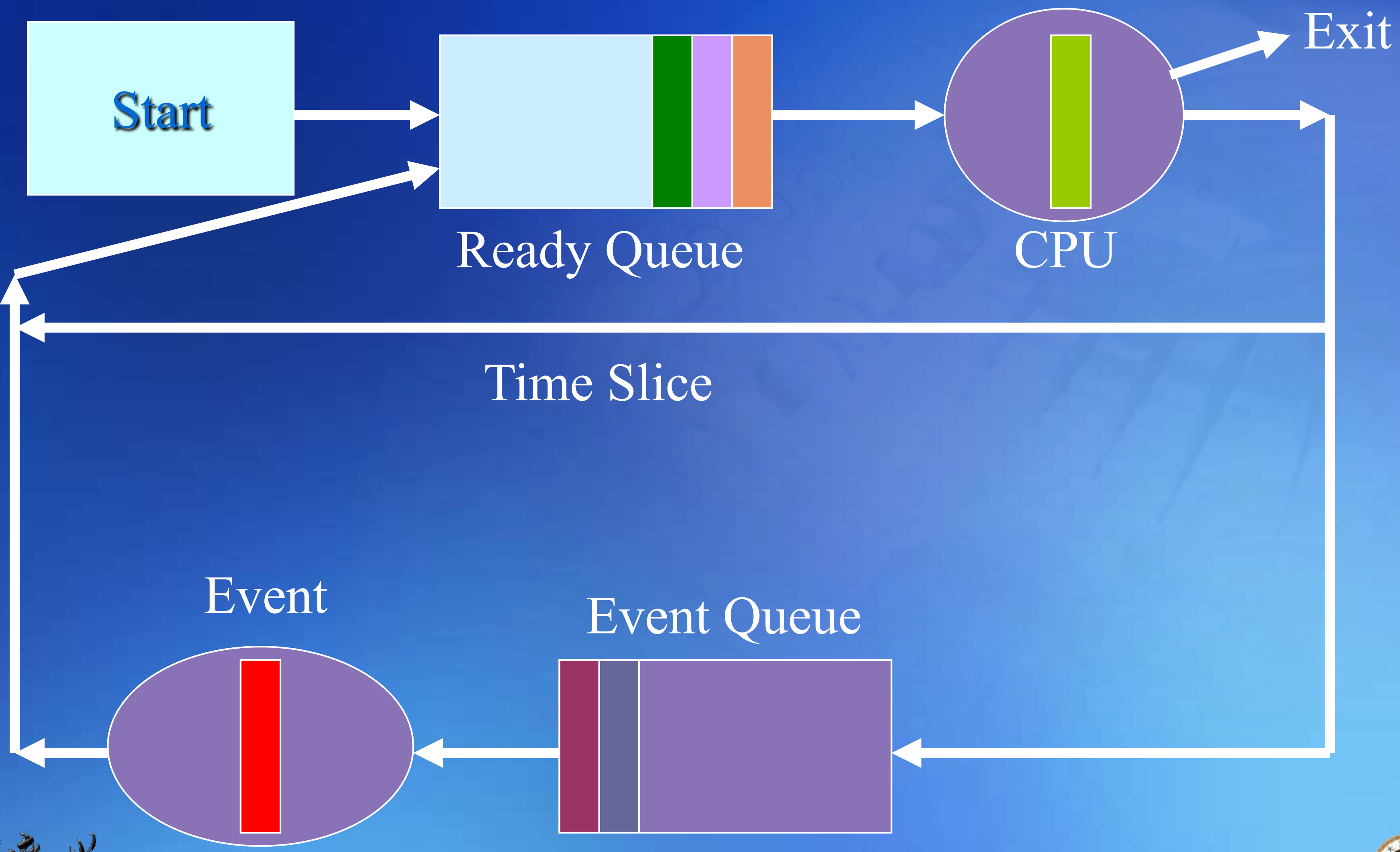
# Example of Multitasking



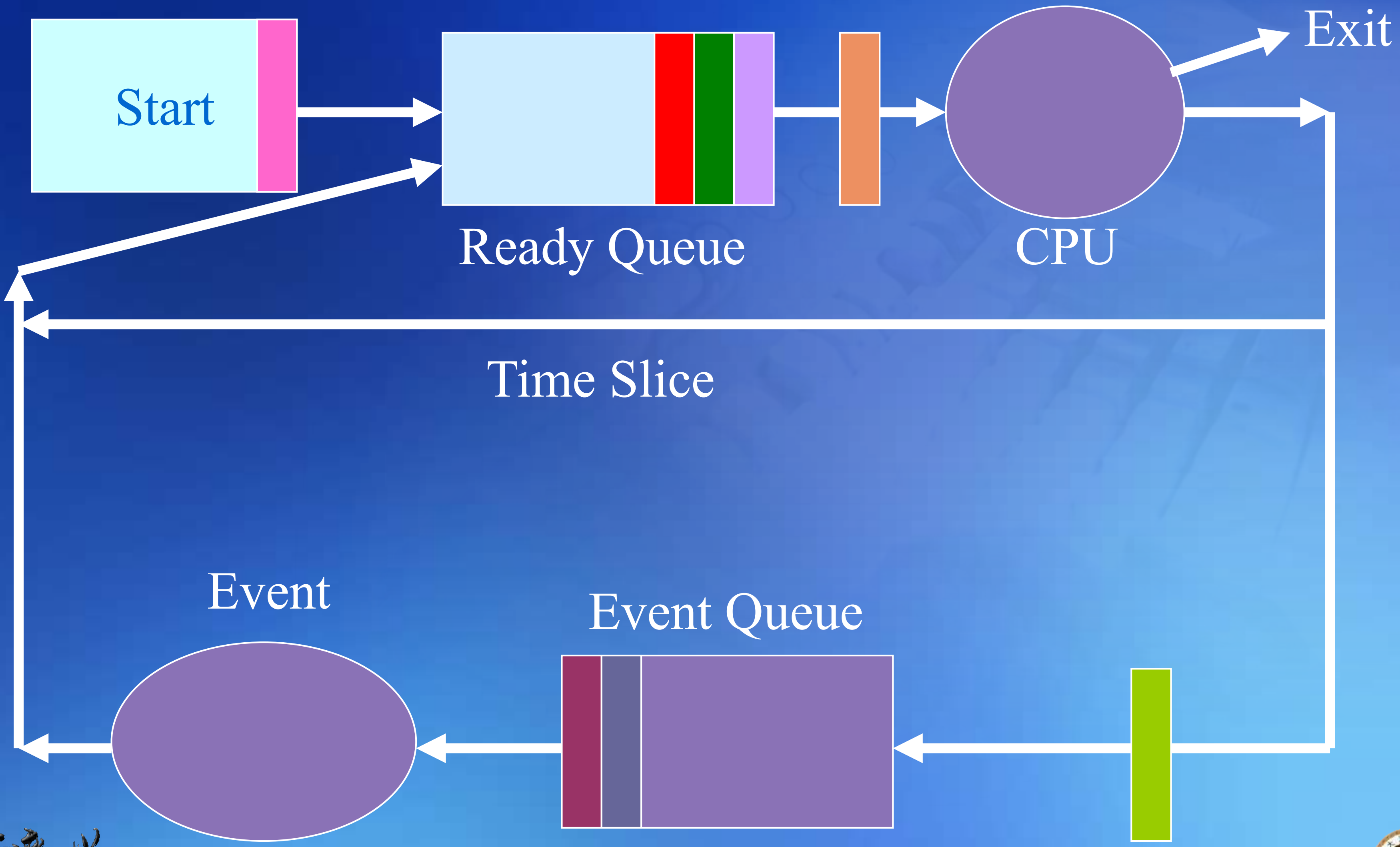
# Example of Multitasking



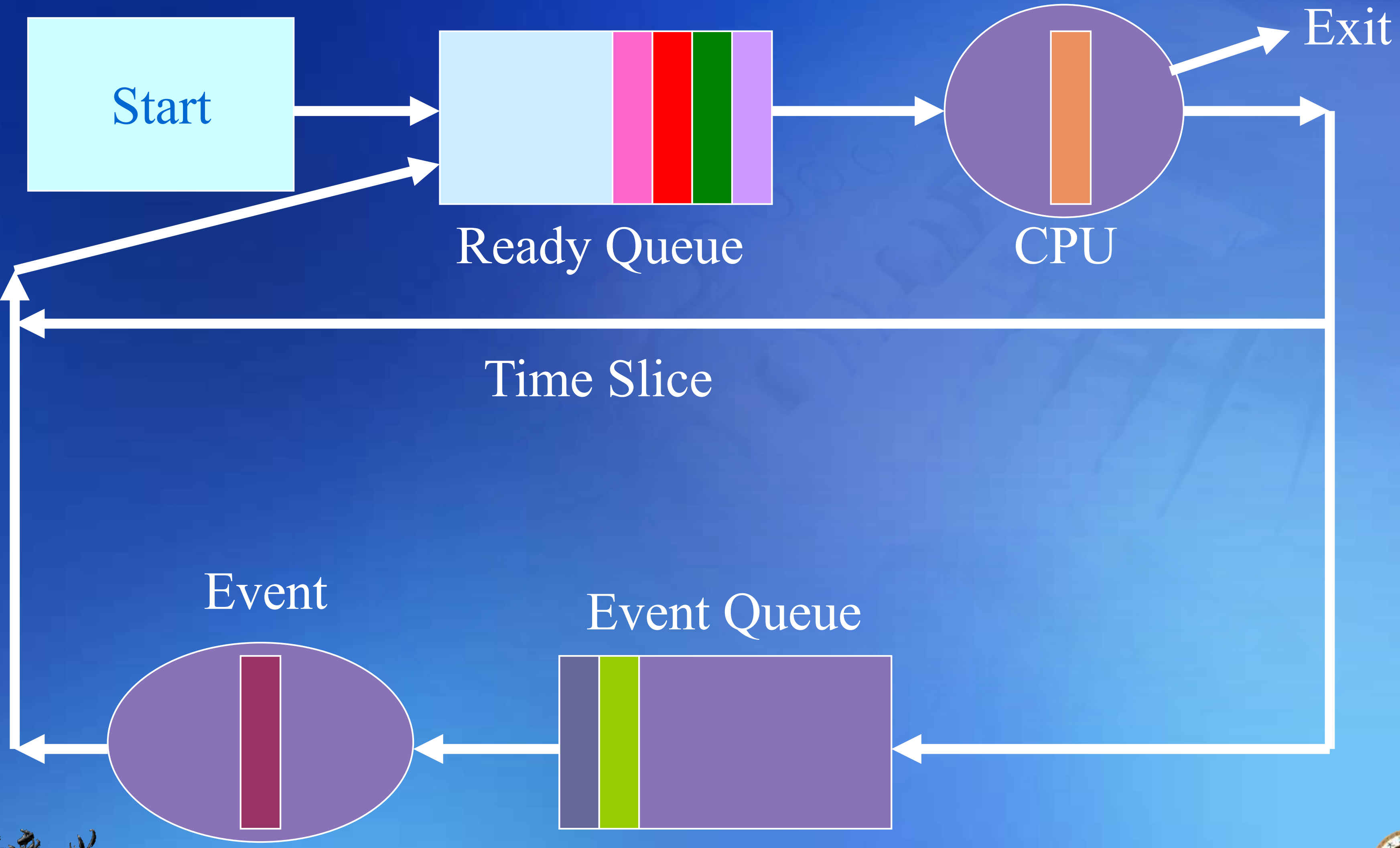
# Example of Multitasking



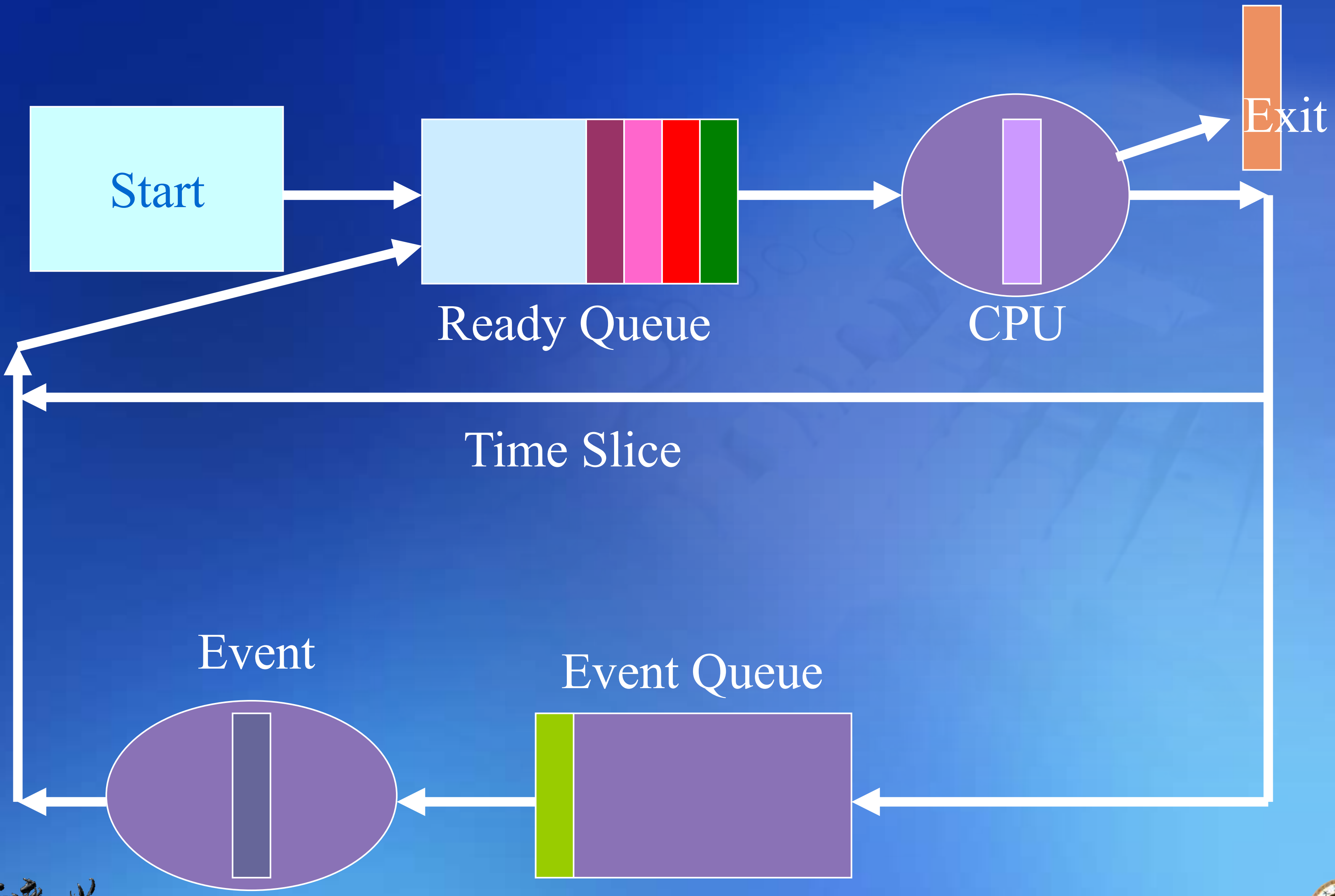
# Example of Multitasking



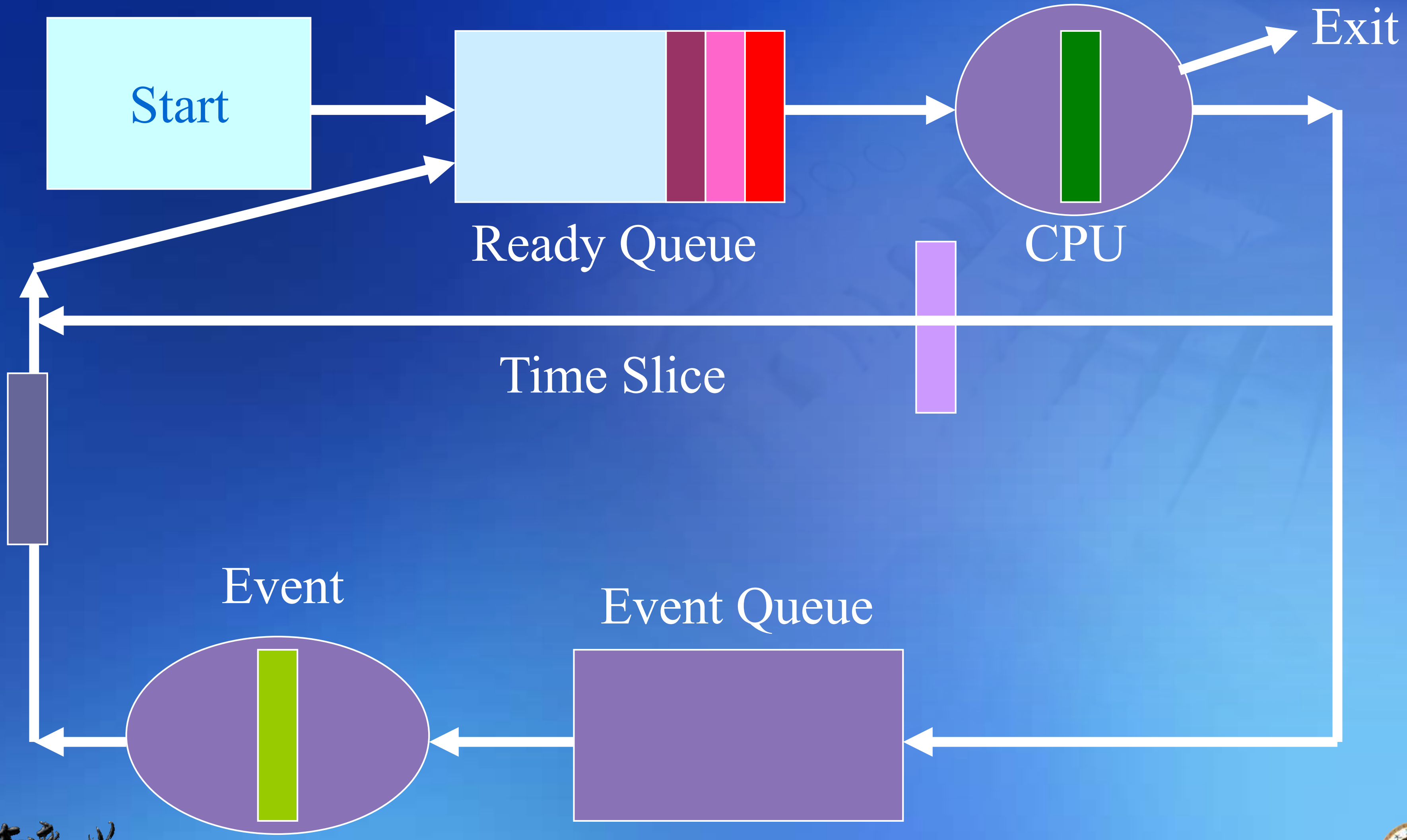
# Example of Multitasking

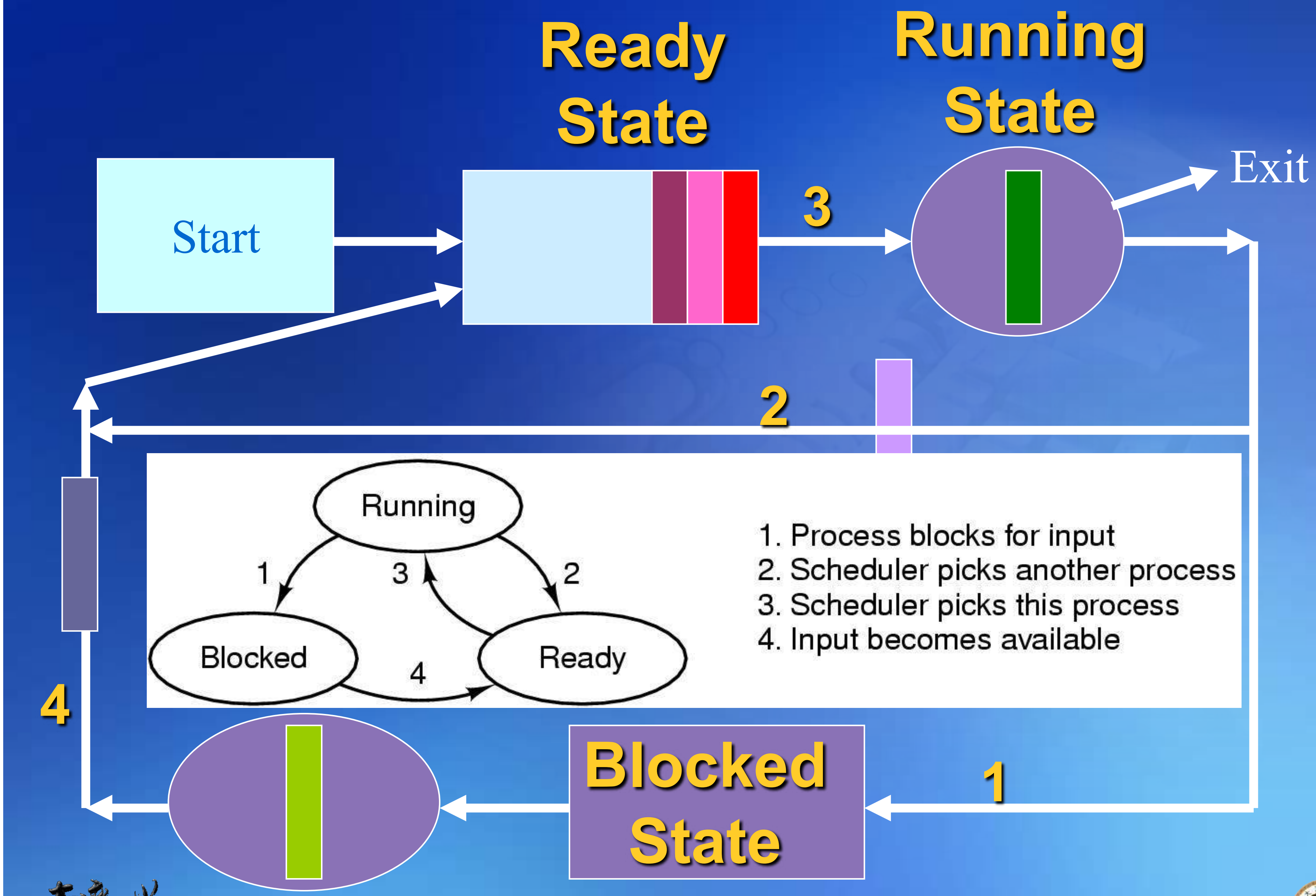


# Example of Multitasking



# Example of Multitasking



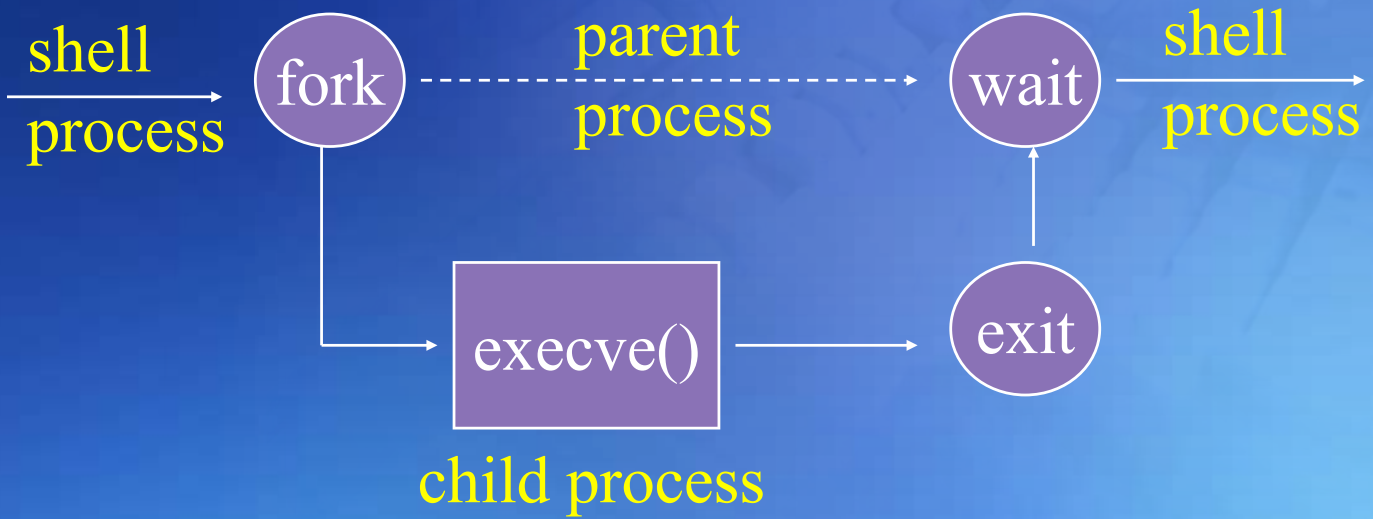


1. Process blocks for input
2. Scheduler picks another process
3. Scheduler picks this process
4. Input becomes available

# UNIX Process

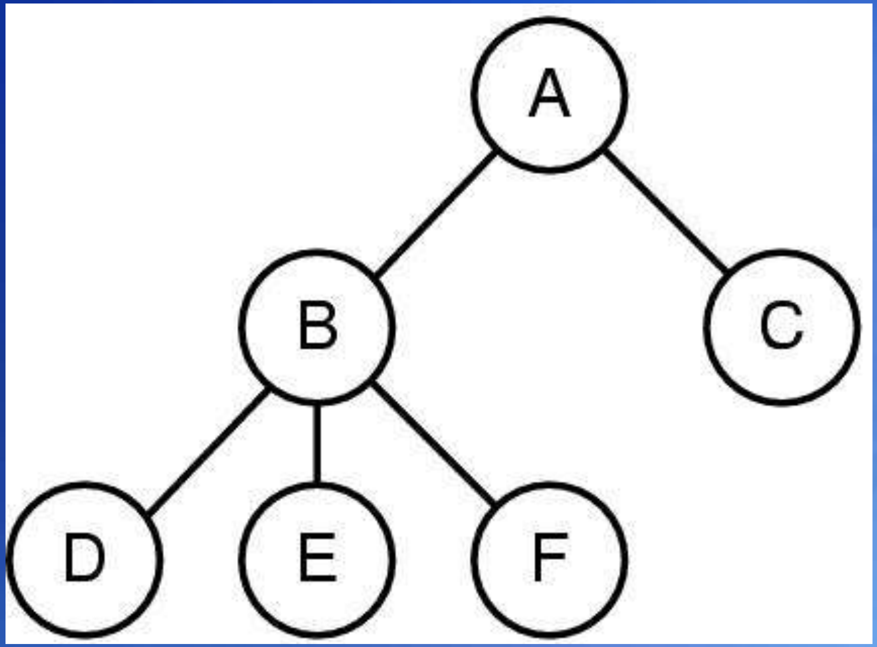
- **Shell**
  - Command interpreters

    e.g., ls, pwd



**(Will be explained in details later; Ch8)**

# A Process Tree



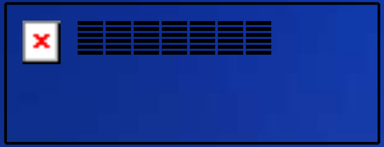
- A created two child processes, B and C
- B created three child processes, D, E, and F

# Common UNIX Shells

- **Shells**
  - Bourne shell, /bin/sh
    - Steve Bourne at Bell Labs
  - C shell, /bin/csh
    - Bill Jay at Berkeley
      - Command-line editing, history, job-control, etc
  - KornShell, /bin/ksh
    - David Korn (successor of Bourne shell)
    - Command-line editing, job-control, etc
  - Related shell command: chsh (i.e.,change shell)

# Inter-process Communication

- **Pipe**



  - A way to send the output of one command to the input of another
- **Filter**
  - A program that takes input and transforms it in some way
  - `wc` - gives a count of words/lines/chars
  - `grep` - searches for lines with a given string
  - `more`
  - `sort` - sorts lines alphabetically or numerically

# Examples of Filter & Pipe

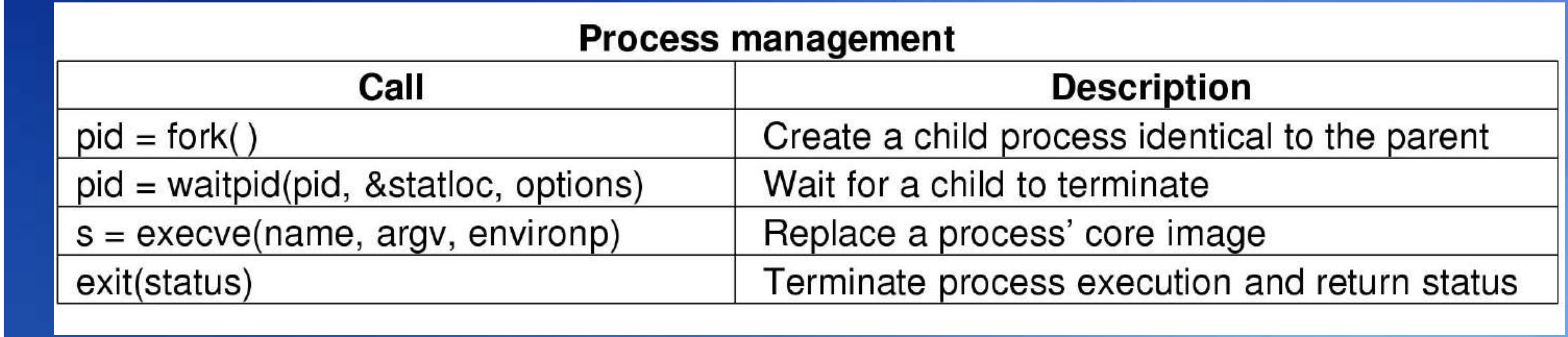
- `ls -la | more`
- `cat file | wc`
- `man ksh | grep “history”`
- `ls -l | grep “bowman” | wc`
- `who | sort > current_users`

```
(Try these commands yourself)
```

**UNIX philosophy**:

- Write programs that do one thing and do it well
- Write programs that work together
- Write programs that handle text streams, because that is the universal interface

# Some System Calls For Process Management



**Process management**

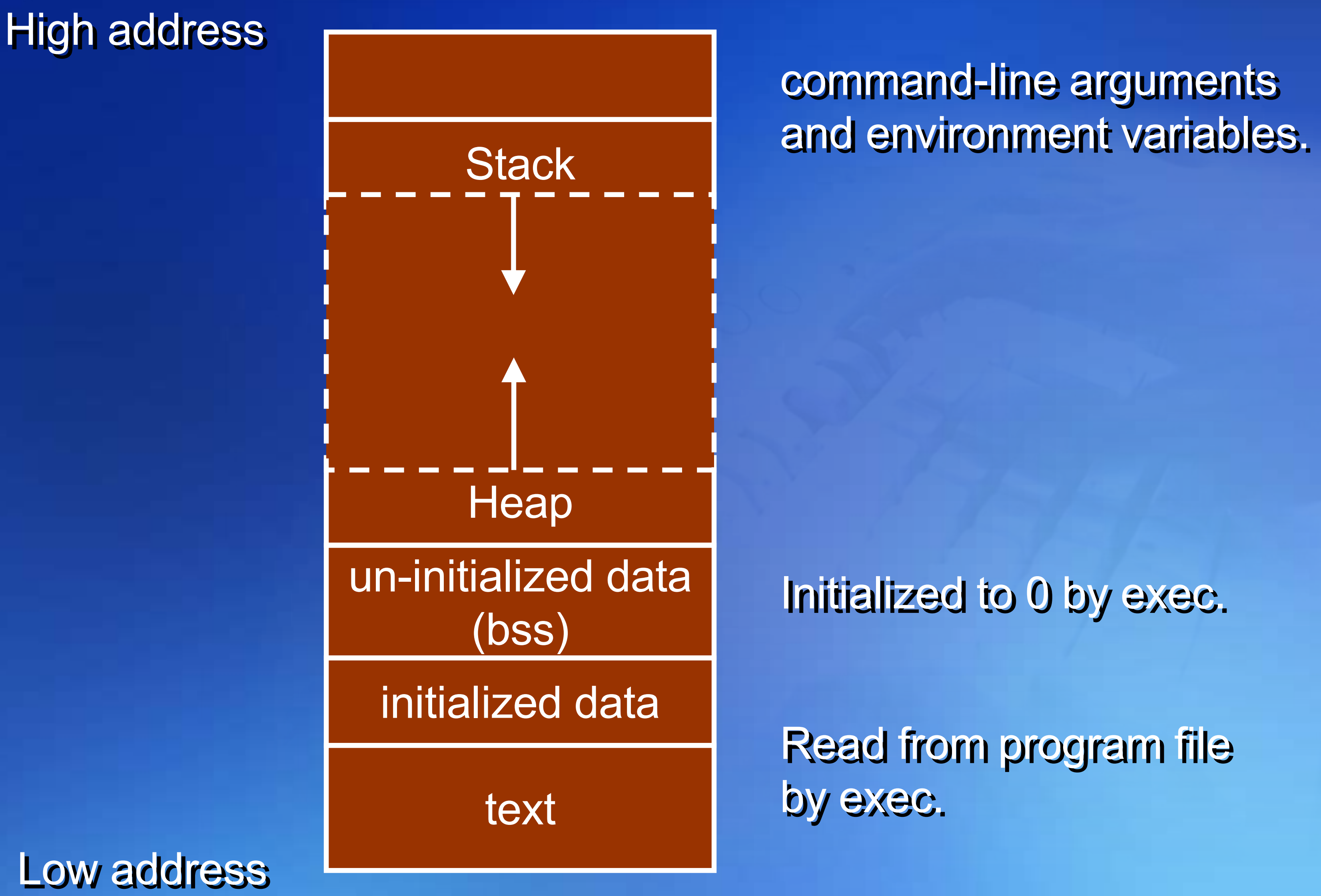
| Call | Description |
|---|---|
| pid = fork( ) | Create a child process identical to the parent |
| pid = waitpid(pid, &statloc, options) | Wait for a child to terminate |
| s = execve(name, argv, environp) | Replace a process' core image |
| exit(status) | Terminate process execution and return status |

**(Will be explained in details later; Ch8)**

# Services for Application Programs

- User Identification
- Process Management
- Memory Management
- File/Directory Management

# Typical Memory Arrangement


High address

command-line arguments and environment variables.

Stack

Heap

un-initialized data (bss)

Initialized to 0 by exec.

initialized data

Read from program file by exec.

text

Low address

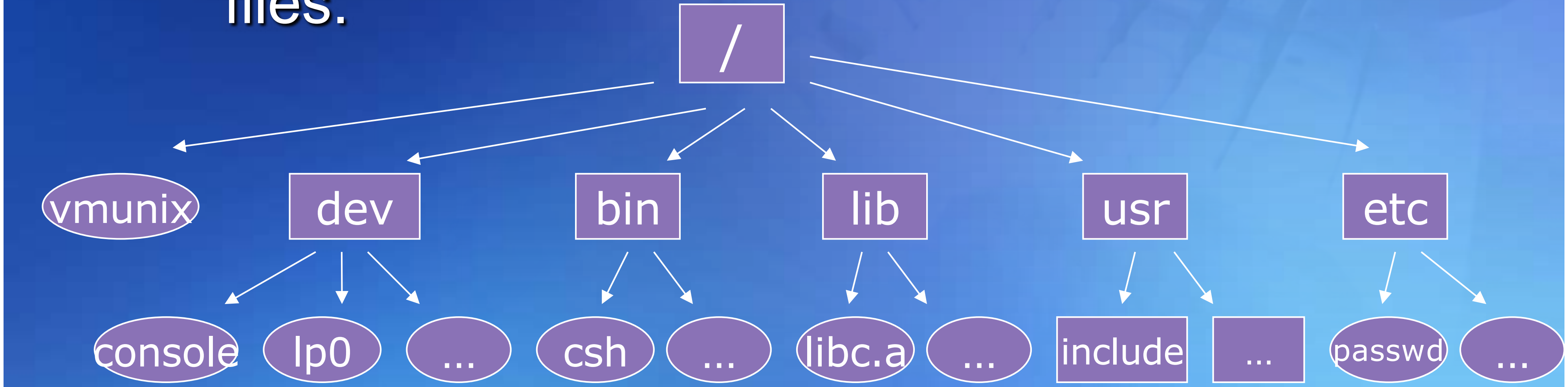
**(Will be explained in details later; Ch7)**

# Services for Application Programs

- User Identification
- Process Management
- Memory Management
- File/Directory Management

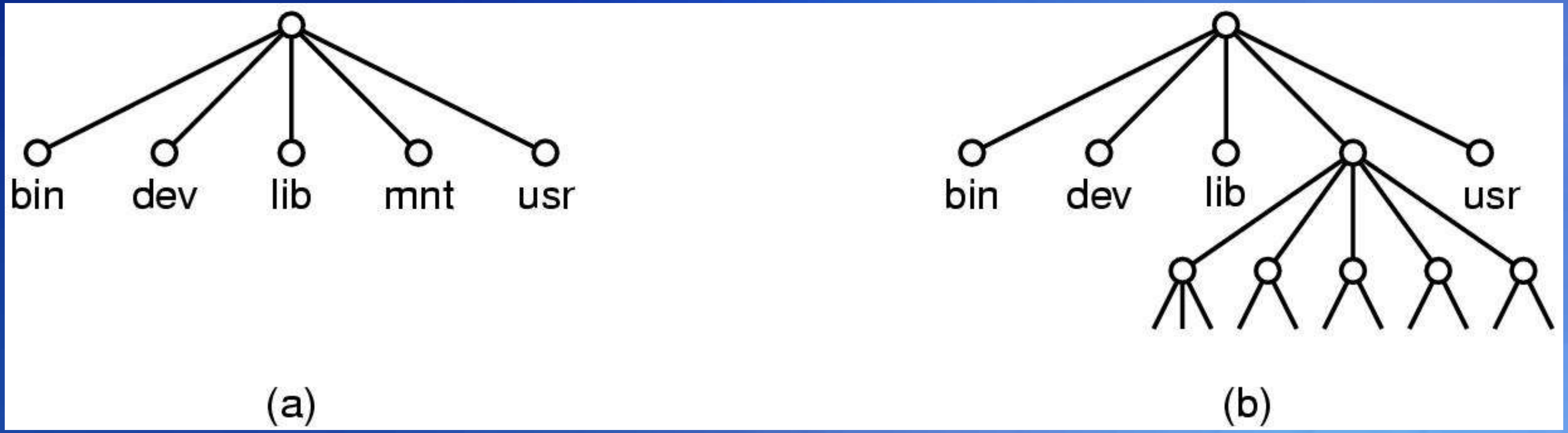
# UNIX File and Directory

- **File**
  - A sequence of bytes
- **Directory**
  - A file that includes info on how to find other files.



* Use command "mount" to show all mounted file systems!

# Mount File Systems



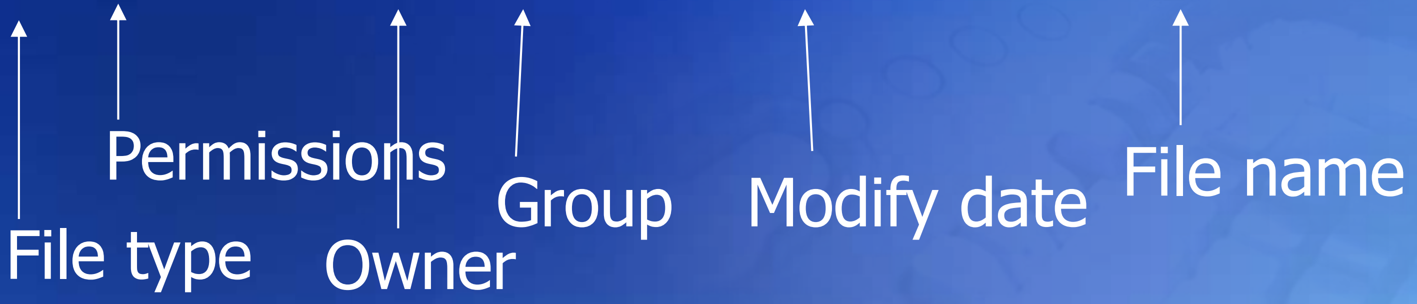
(a) File system before the mount

(b) File system after the mount

# File Permission

## Output of ls -l

```
total 4
lrwxr-xr-x 1 test user 18 Aug 28 13:41 home -> /usr/people/maria/
-rw-r--r-- 1 test user 94 Aug 28 13:42 nothing.txt
drwxr-xr-x 2 test user  9 Aug 28 13:40 test_dir/
```



File type

Permissions

Owner

Group

Modify date

File name

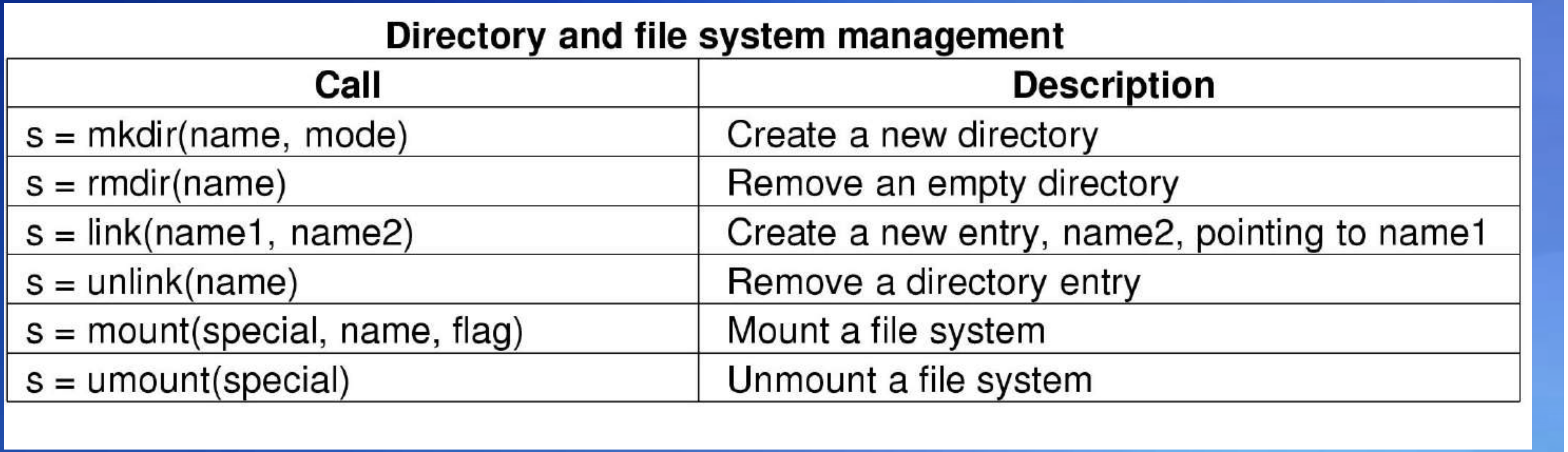
Read (r) 4, write (w) 2, and execute (x) 1

For owner, group, and world (everyone)

```
chmod <mode> <file(s)>
chmod 700 file.txt
chmod g+rw file.txt
```

Related shell commands: chmod,chown,touch

# Some System Calls For Directory Management

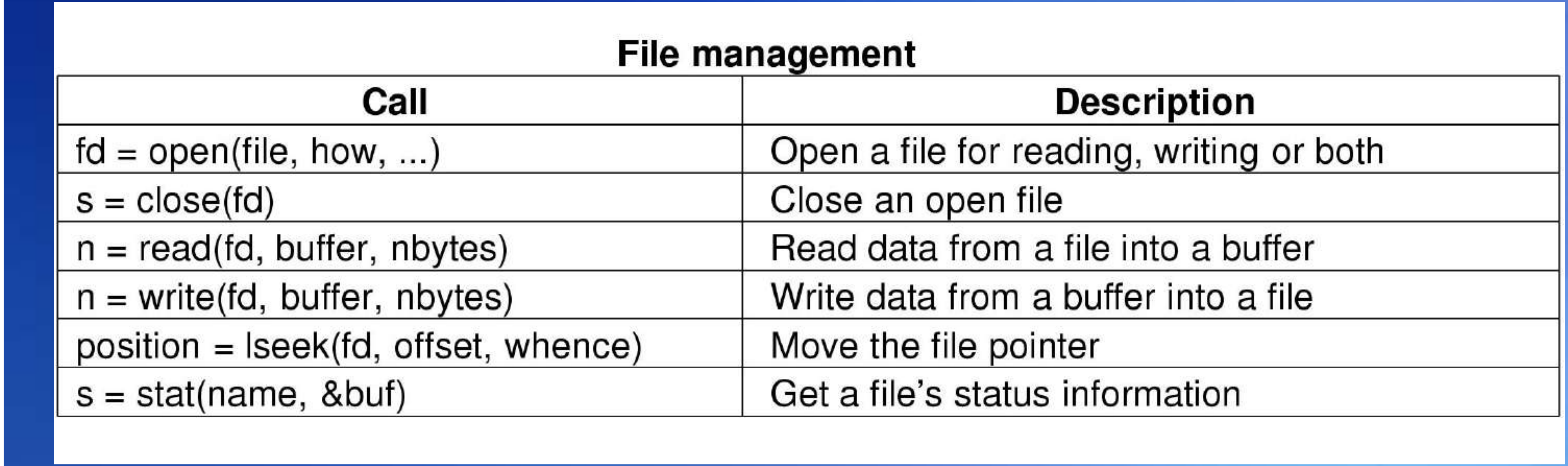


Directory and file system management

| Call | Description |
|---|---|
| s = mkdir(name, mode) | Create a new directory |
| s = rmdir(name) | Remove an empty directory |
| s = link(name1, name2) | Create a new entry, name2, pointing to name1 |
| s = unlink(name) | Remove a directory entry |
| s = mount(special, name, flag) | Mount a file system |
| s = umount(special) | Unmount a file system |

**(Will be explained in details later; Ch4)**

# Some System Calls For File Management



**File management**

| Call | Description |
|---|---|
| fd = open(file, how, ...) | Open a file for reading, writing or both |
| s = close(fd) | Close an open file |
| n = read(fd, buffer, nbytes) | Read data from a file into a buffer |
| n = write(fd, buffer, nbytes) | Write data from a buffer into a file |
| position = lseek(fd, offset, whence) | Move the file pointer |
| s = stat(name, &buf) | Get a file's status information |

**(Will be explained in details later; Ch3)**

# What You Should Know

- What is Operating System
- UNIX architecture
- Basic concepts of
  - System call vs. function call
  - Kernel/user mode/space
  - Multitasking, time sharing
  - User identification
  - Process
  - Memory arrangement
  - Directory and file