System Structures — Chapter 2

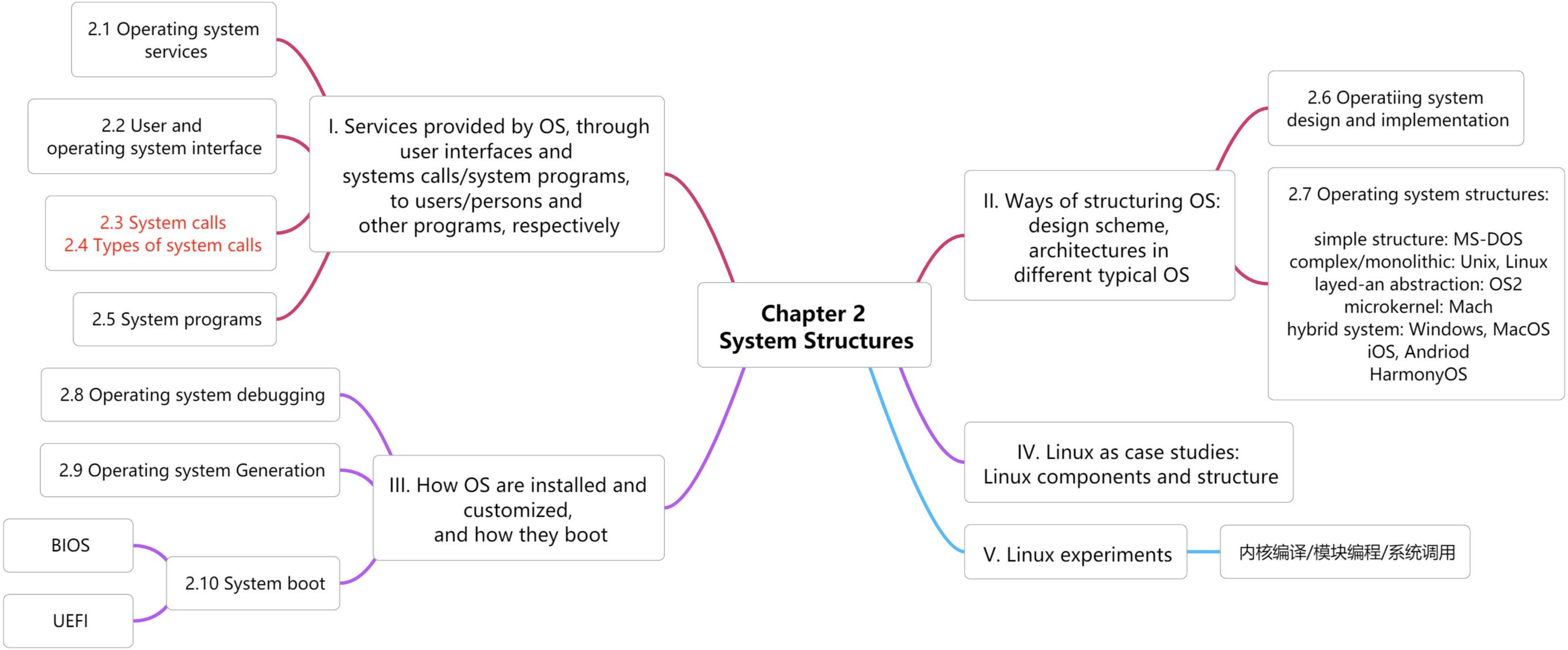
2022年9月

薛哲

School of Computer Science (National Pilot Software Engineering School)

北京郵電大學

Outline



Chapter 2
System Structures

I. Services provided by OS, through user interfaces and systems calls/system programs, to users/persons and other programs, respectively

- 2.1 Operating system services
- 2.2 User and operating system interface
- 2.3 System calls
- 2.4 Types of system calls
- 2.5 System programs

II. Ways of structuring OS: design scheme, architectures in different typical OS

- 2.6 Operatiing system design and implementation
- 2.7 Operating system structures:

  simple structure: MS-DOS
  complex/monolithic: Unix, Linux
  layed-an abstraction: OS2
  microkernel: Mach
  hybrid system: Windows, MacOS
  iOS, Andriod
  HarmonyOS

III. How OS are installed and customized, and how they boot

- 2.8 Operating system debugging
- 2.9 Operating system Generation
- 2.10 System boot
  - BIOS
  - UEFI

IV. Linux as case studies: Linux components and structure

V. Linux experiments

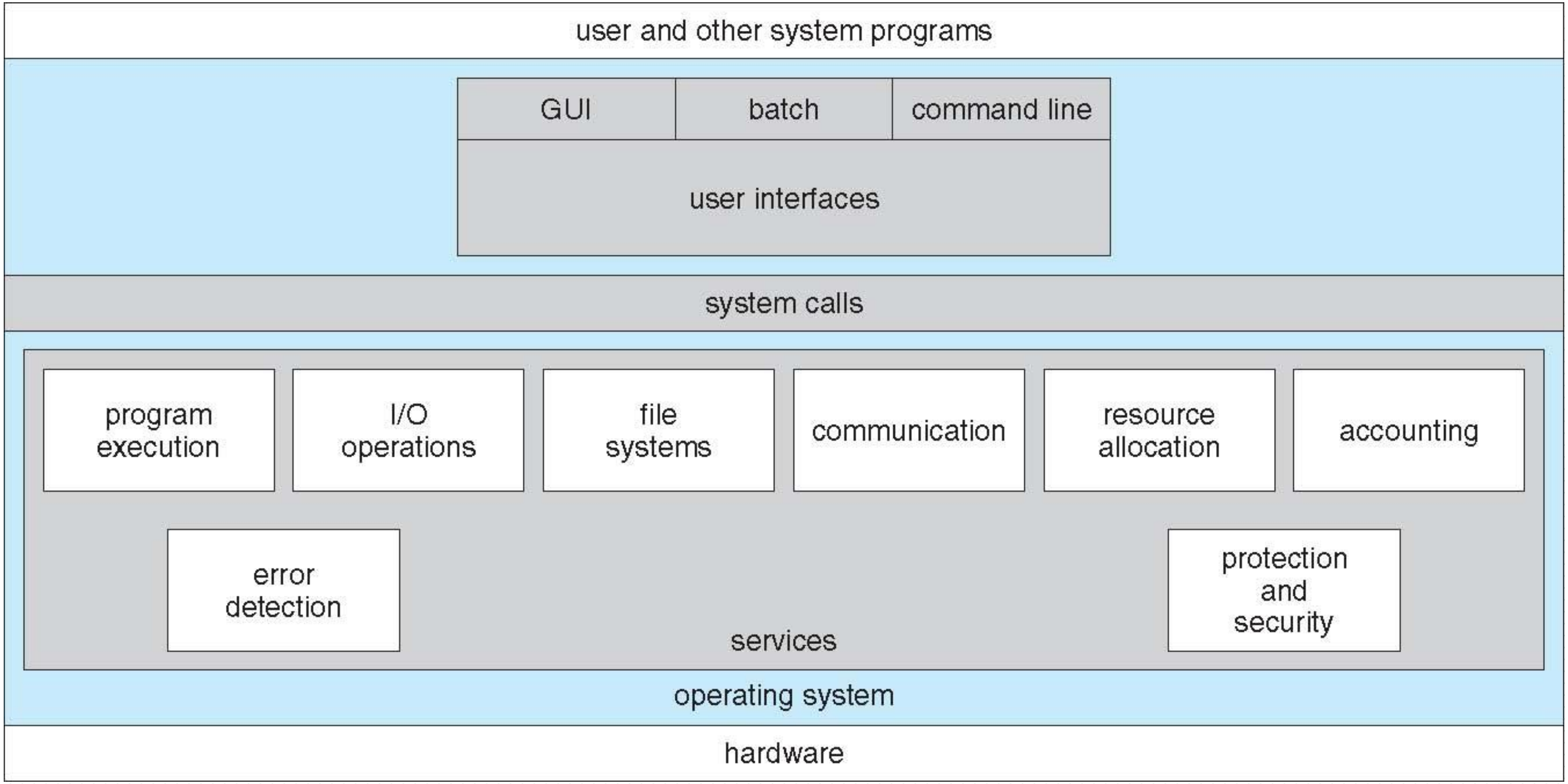
- 内核编译/模块编程/系统调用

2.1 Operating Systems Services

- Operating systems provide an environment for execution of programs and services to programs and users/persons
- One set of operating-system services provides functions that are helpful to the user:
  - **User interface** - Almost all operating systems have a user interface (**UI**).
    - Varies between **Command-Line** (**CLI**), **Graphics User Interface** (**GUI**), **Batch**
  - **Program execution** - The system must be able to load a program into memory and to run that program, and execution, either normally or abnormally (indicating error)
  - **I/O operations** - A running program may require I/O, which may involve a file or an I/O device
  - **File-system manipulation** - The file system is of particular interest. Programs need to read and write files and directories, create and delete them, search them, list file Information, permission management.

2.1 Operating Systems Services

- Operating systems provide an environment for execution of programs and services to programs and users/persons



2.1 Operating Systems Services

- One set of operating-system services provides functions that are helpful to the user:
  - **User interface** - Almost all operating systems have a user interface (**UI**).
    - Varies between **Command-Line** **(CLI)**, **Graphics User Interface** **(GUI)**, **Batch**
  - **Program execution** - The system must be able to load a program into memory and to run that program, end execution, either normally or abnormally (indicating error)
  - **I/O operations** - A running program may require I/O, which may involve a file or an I/O device
  - **File-system manipulation** - The file system is of particular interest. Programs need to read and write files and directories, create and delete them, search them, list file Information, permission management
  - **Communications** – Processes may exchange information, on the same computer or between computers over a network
    - Communications may be via shared memory or through message passing (packets moved by the OS)

Operating Systems Services (Cont.)

- **Error detection** – OS needs to be constantly aware of possible errors
  - May occur in the CPU and memory hardware, in I/O devices, in user program
  - For each type of error, OS should take the appropriate action to ensure correct and consistent computing
  - Debugging facilities can greatly enhance the user′s and programmer′s abilities to efficiently use the system

Operating Systems Services (Cont.)

- Another set of OS functions exists for ensuring the efficient operation of the system itself via resource sharing
  - **Resource allocation -** When multiple users or multiple jobs running concurrently, resources must be allocated to each of them
    - Many types of resources -   CPU cycles, main memory, file storage, I/O devices.
  - **Accounting -** To keep track of which users use how much and what kinds of computer resources
  - **Protection and security -** The owners of information stored in a multiuser or networked computer system may want to control use of that information, concurrent processes should not interfere with each other
    - **Protection** involves ensuring that all access to system resources is controlled
    - **Security** of the system from outsiders requires user authentication（身份验证）, extends to defending external I/O devices from invalid access attempts

Services: what functions does the OS provide for OS users

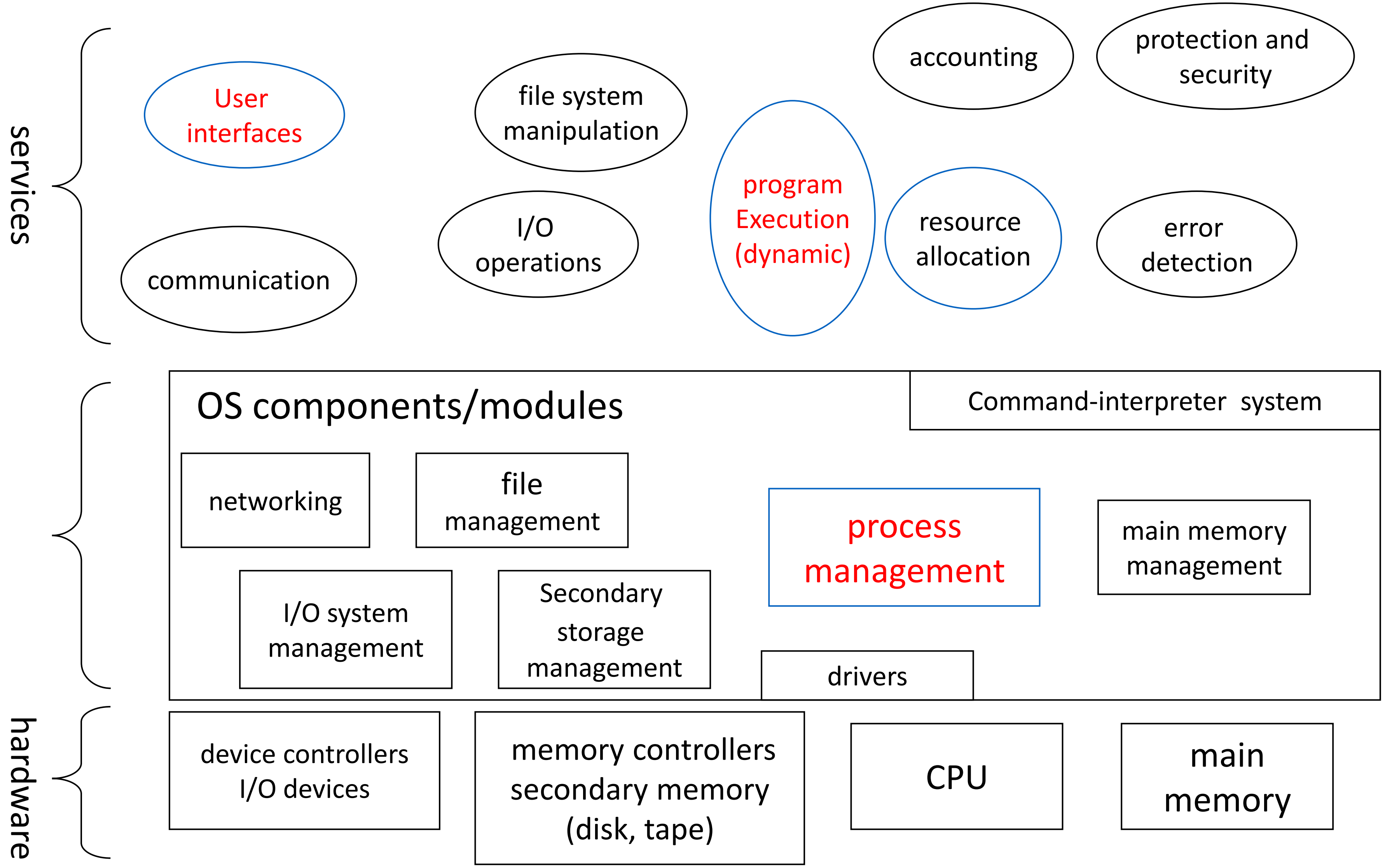
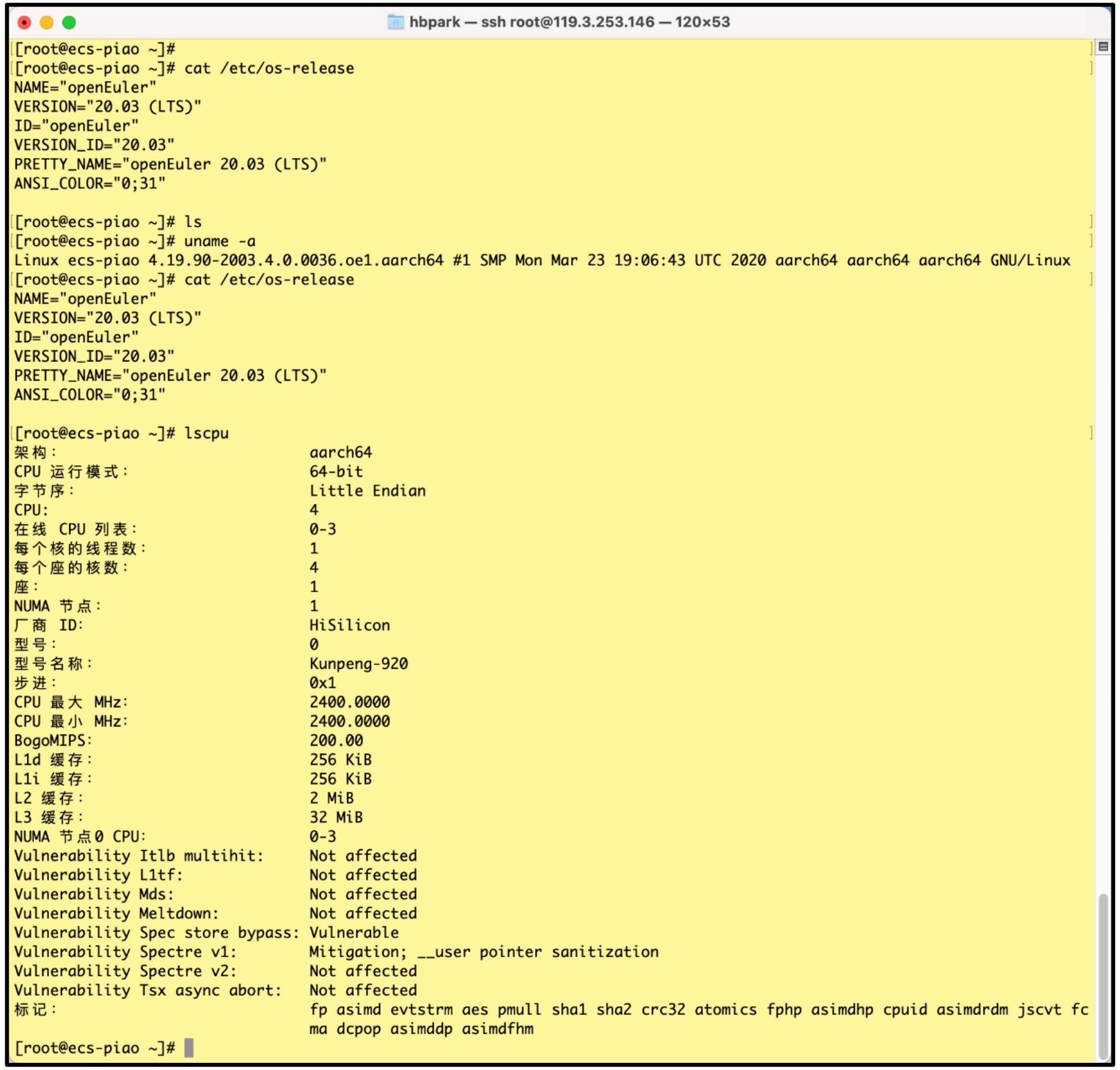


Fig. 2.0.1

2.2 User Operating System Interface

- Interaction between OS users and operating systems/computers
  - **persons as user**
    - CLI, GUI, touchscreen
    - voice/speech, gesture/body posture, brain-machine interface, VR, AR
  - system and application programs
    - system calls/API, Job Control Language JCL in batch systems
- CLI or **command interpreter** allows direct command entry
  - Sometimes implemented in kernel
    - e.g. MS DOS (Disk Operating System)
  - Sometimes by systems program, sometimes multiple flavors implemented – **shells**
    - interpreter + command programing language
    - Bourne/Bush Shell in Unix/Linux
  - Primarily fetches a command from user and executes it

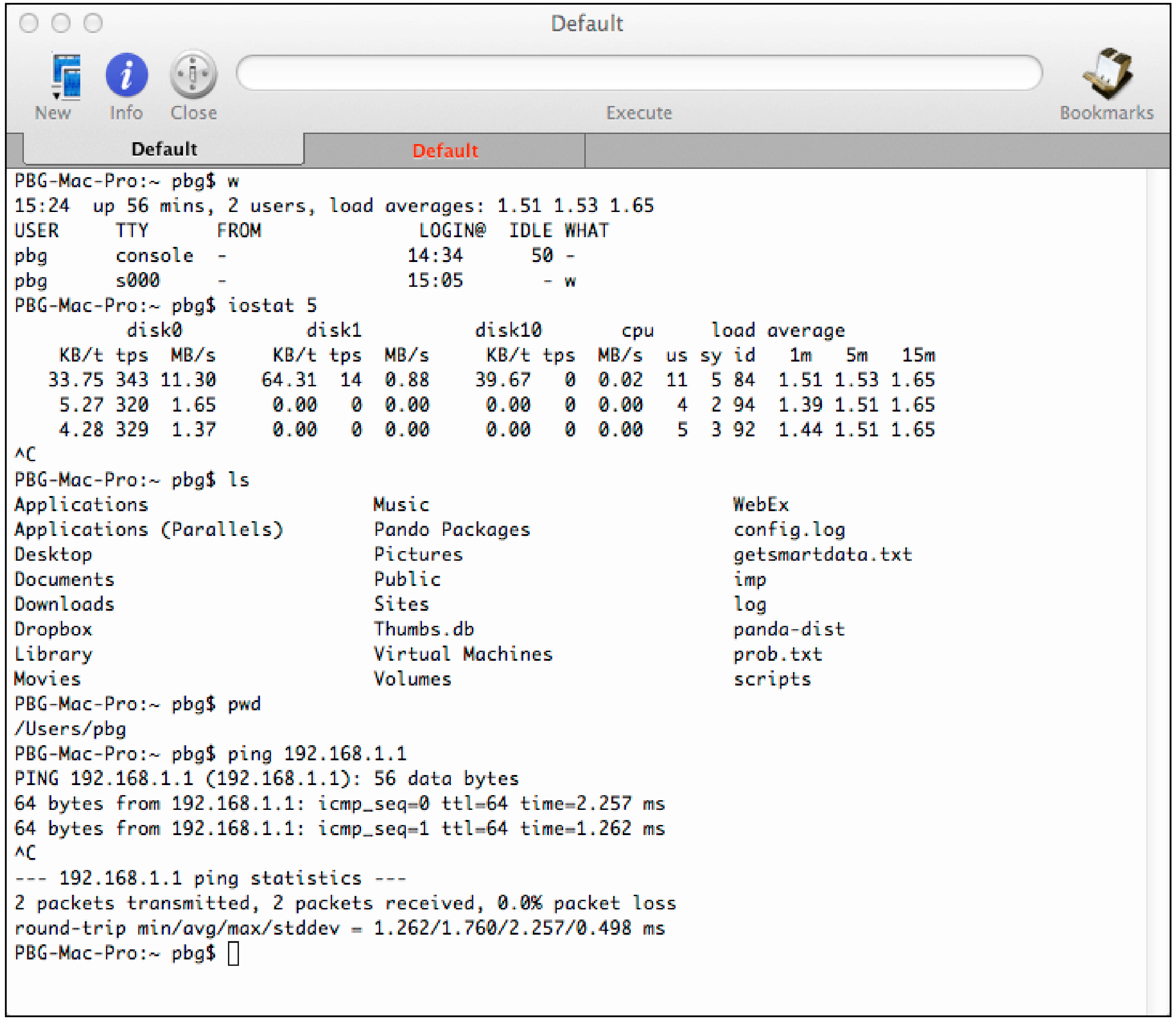


```
[root@ecs-piao ~]#
[root@ecs-piao ~]# cat /etc/os-release
NAME="openEuler"
VERSION="20.03 (LTS)"
ID="openEuler"
VERSION_ID="20.03"
PRETTY_NAME="openEuler 20.03 (LTS)"
ANSI_COLOR="0;31"

[root@ecs-piao ~]# ls
[root@ecs-piao ~]# uname -a
Linux ecs-piao 4.19.90-2003.4.0.0036.oe1.aarch64 #1 SMP Mon Mar 23 19:06:43 UTC 2020 aarch64 aarch64 aarch64 GNU/Linux
[root@ecs-piao ~]# cat /etc/os-release
NAME="openEuler"
VERSION="20.03 (LTS)"
ID="openEuler"
VERSION_ID="20.03"
PRETTY_NAME="openEuler 20.03 (LTS)"
ANSI_COLOR="0;31"

[root@ecs-piao ~]# lscpu
架构：                           aarch64
CPU 运行模式：                   64-bit
字节序：                         Little Endian
CPU:                             4
在线 CPU 列表：                  0-3
每个核的线程数：                 1
每个座的核数：                   4
座：                             1
NUMA 节点：                      1
厂商 ID:                         HiSilicon
型号：                           0
型号名称：                       Kunpeng-920
步进：                           0x1
CPU 最大 MHz:                    2400.0000
CPU 最小 MHz:                    2400.0000
BogoMIPS:                        200.00
L1d 缓存：                       256 KiB
L1i 缓存：                       256 KiB
L2 缓存：                        2 MiB
L3 缓存：                        32 MiB
NUMA 节点0 CPU:                  0-3
Vulnerability Itlb multihit:     Not affected
Vulnerability L1tf:              Not affected
Vulnerability Mds:               Not affected
Vulnerability Meltdown:          Not affected
Vulnerability Spec store bypass: Vulnerable
Vulnerability Spectre v1:        Mitigation; __user pointer sanitization
Vulnerability Spectre v2:        Not affected
Vulnerability Tsx async abort:   Not affected
标记：                           fp asimd evtstrm aes pmull sha1 sha2 crc32 atomics fphp asimdhp cpuid asimdrdm jscvt fc
                                 ma dcpop asimddp asimdfhm
[root@ecs-piao ~]# 
```

vim in openEuler



```
PBG-Mac-Pro:~ pbg$ w
15:24  up 56 mins, 2 users, load averages: 1.51 1.53 1.65
USER     TTY      FROM              LOGIN@  IDLE WHAT
pbg      console  -                14:34      50 -
pbg      s000     -                15:05       - w
PBG-Mac-Pro:~ pbg$ iostat 5
          disk0           disk1           disk10          cpu     load average
    KB/t tps  MB/s     KB/t tps  MB/s     KB/t tps  MB/s  us sy id   1m   5m   15m
   33.75 343 11.30    64.31  14  0.88    39.67   0  0.02  11  5 84  1.51 1.53 1.65
    5.27 320  1.65     0.00   0  0.00     0.00   0  0.00   4  2 94  1.39 1.51 1.65
    4.28 329  1.37     0.00   0  0.00     0.00   0  0.00   5  3 92  1.44 1.51 1.65
^C
PBG-Mac-Pro:~ pbg$ ls
Applications                Music                       WebEx
Applications (Parallels)    Pando Packages              config.log
Desktop                     Pictures                    getsmartdata.txt
Documents                   Public                      imp
Downloads                   Sites                       log
Dropbox                     Thumbs.db                   panda-dist
Library                     Virtual Machines            prob.txt
Movies                      Volumes                     scripts
PBG-Mac-Pro:~ pbg$ pwd
/Users/pbg
PBG-Mac-Pro:~ pbg$ ping 192.168.1.1
PING 192.168.1.1 (192.168.1.1): 56 data bytes
64 bytes from 192.168.1.1: icmp_seq=0 ttl=64 time=2.257 ms
64 bytes from 192.168.1.1: icmp_seq=1 ttl=64 time=1.262 ms
^C
--- 192.168.1.1 ping statistics ---
2 packets transmitted, 2 packets received, 0.0% packet loss
round-trip min/avg/max/stddev = 1.262/1.760/2.257/0.498 ms
PBG-Mac-Pro:~ pbg$ 
```

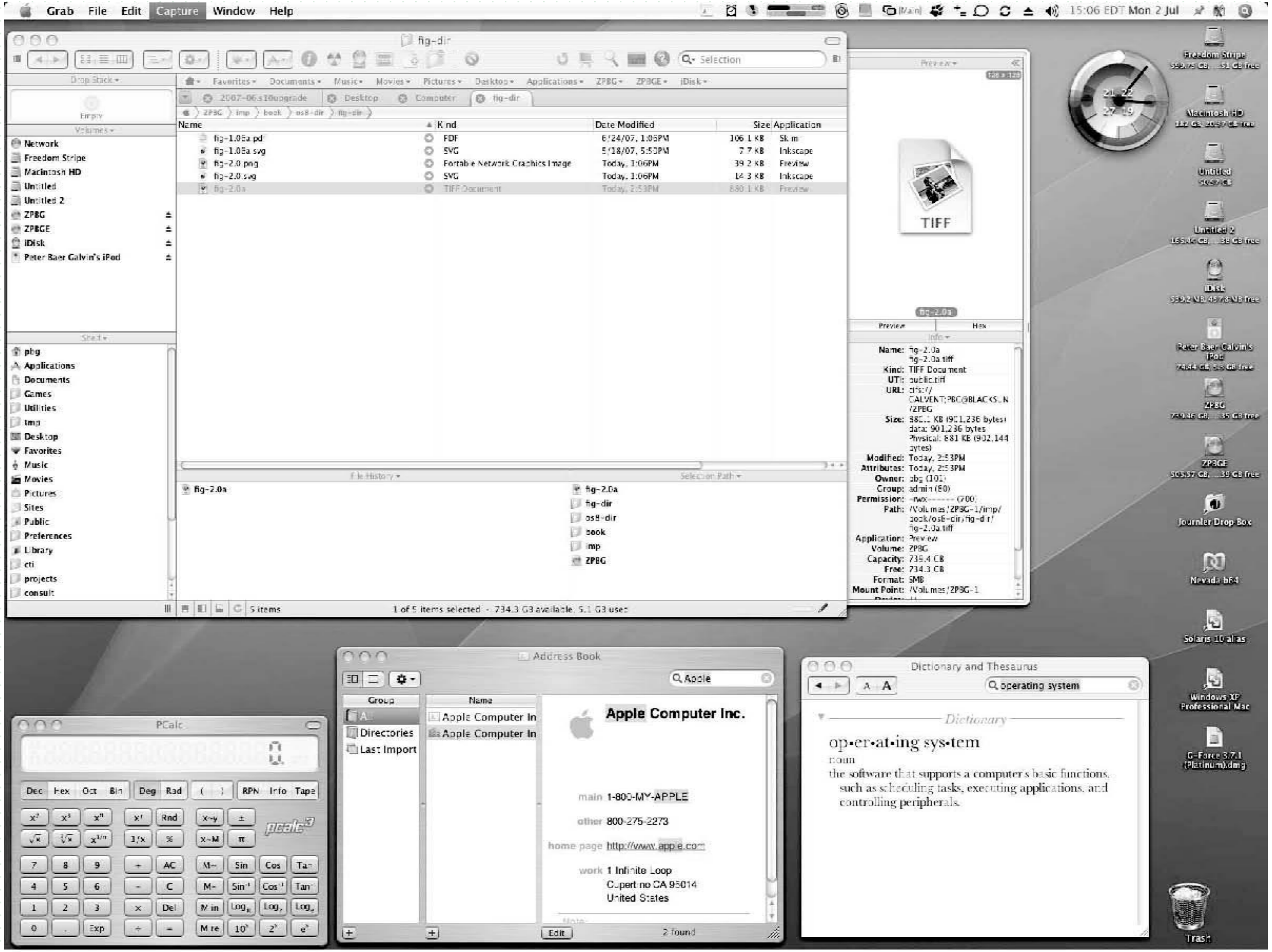
Bourne Shell Command Interpreter

User Operating System Interface - GUI

- User-friendly **desktop** metaphor interface
  - Usually mouse, keyboard, and monitor
  - **Icons** represent files, programs, actions, etc
  - Various mouse buttons over objects in the interface cause various actions (provide information, options, execute function, open directory (known as a **folder**)
  - Invented at Xerox PARC

- Many systems now include both CLI and GUI interfaces
  - Microsoft Windows is GUI with CLI ″command″ shell
  - Apple Mac OS X is ″Aqua″ GUI interface with UNIX kernel underneath and shells available
  - Unix and Linux have CLI with optional GUI interfaces (CDE, KDE, GNOME)
    - 联机控制级窗口X-Windows for Linux

Mac OS X GUI



Name	Kind	Date Modified	Size	Application
fig-1.05a.pdf	PDF	6/24/07, 1:06PM	106.1 KB	Skim
fig-1.05a.svg	SVG	5/18/07, 5:50PM	7.7 KB	Inkscape
fig-2.0.png	Portable Network Graphics Image	Today, 1:06PM	39.2 KB	Preview
fig-2.0.svg	SVG	Today, 1:06PM	14.3 KB	Inkscape
fig-2.0a	TIFF Document	Today, 2:53PM	880.1 KB	Preview

5 items

1 of 5 items selected · 734.3 GB available, 5.1 GB used

Name: fig-2.0a
fig-2.0a.tiff
Kind: TIFF Document
UTI: public.tiff
URL: cifs://CALVENT;PBC@BLACKSUN/ZPBG
Size: 880.1 KB (901,236 bytes)
data: 901,236 bytes
Physical: 881 KB (902,144 bytes)
Modified: Today, 2:53PM
Attributes: Today, 2:53PM
Owner: pbg (101)
Group: admin (80)
Permission: -rwx------ (700)
Path: /Volumes/ZPBG-1/imp/book/os8-dir/fig-dir/fig-2.0a.tiff
Application: Preview
Volume: ZPBG
Capacity: 735.4 GB
Free: 734.3 GB
Format: SMB
Mount Point: /Volumes/ZPBG-1

Apple Computer Inc.

main 1-800-MY-APPLE
other 800-275-2273
home page http://www.apple.com
work 1 Infinite Loop
Cupertino CA 95014
United States

op·er·at·ing sys·tem
noun
the software that supports a computer's basic functions, such as scheduling tasks, executing applications, and controlling peripherals.

Touchscreen Interfaces

- Touchscreen devices require new interfaces
  - Mouse not possible or not desired
  - Actions and selection based on **gestures**
  - Virtual keyboard for text entry
- Gesture/Posture-based interface

Data Glove

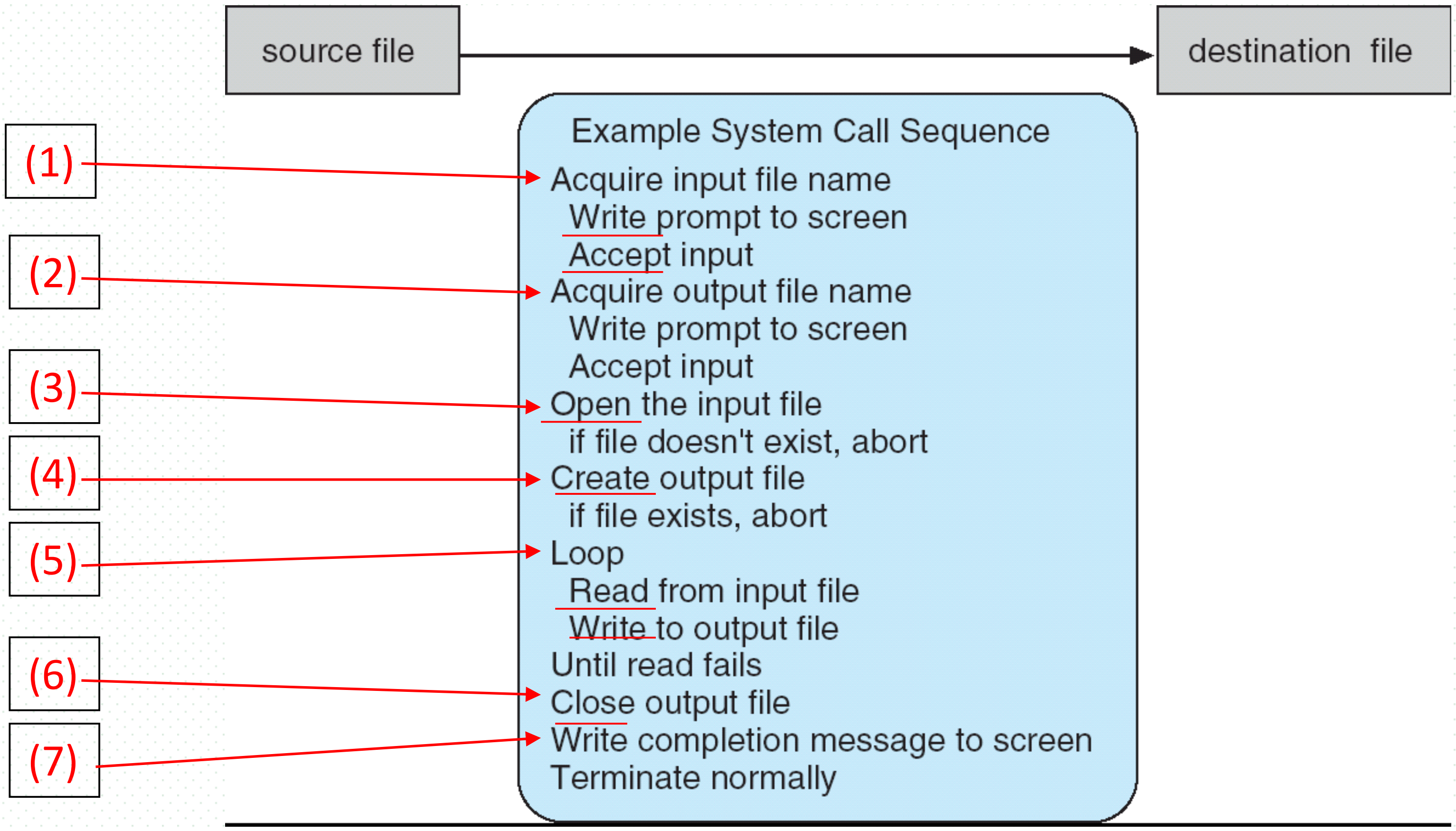
微软Kinect：
动作感应游戏控制器

- Voice commands, VUI
  - e.g. 小爱同学 in xiaomi, based on speech recognition and NLP

2.3 System Call

- ***System calls*** provide the programming interface to the services made available by the operating system, e.g. *file access*
- Typically written in a high-level language (C or C++)
- Mostly accessed by programs via a high-level Application Program Interface (API) rather than direct system call use
- System call *vs* API
- Three most common APIs
  - Win32 API for Windows
  - POSIX API for POSIX-based systems (including virtually all versions of UNIX, Linux, and Mac OS X), Pthread API
  - Java API for the Java virtual machine (JVM)

Example

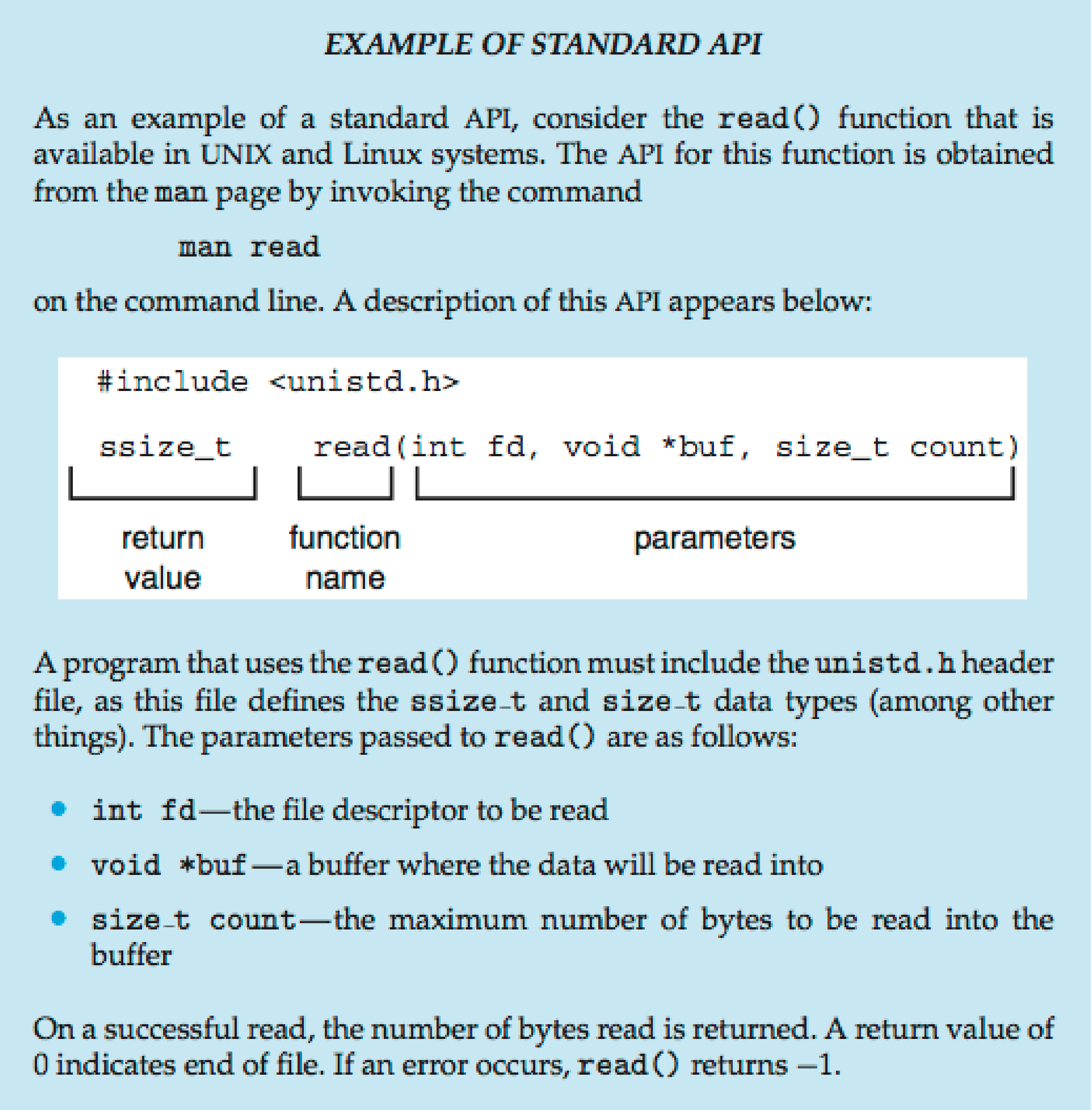


Example System Call Sequence

```
Acquire input file name
  Write prompt to screen
  Accept input
Acquire output file name
  Write prompt to screen
  Accept input
Open the input file
  if file doesn't exist, abort
Create output file
  if file exists, abort
Loop
  Read from input file
  Write to output file
Until read fails
Close output file
Write completion message to screen
Terminate normally
```

Fig. 2.1 System call to copy the contents of one file to another file

Example of Standard API



EXAMPLE OF STANDARD API

As an example of a standard API, consider the `read()` function that is available in UNIX and Linux systems. The API for this function is obtained from the `man` page by invoking the command

```
man read
```

on the command line. A description of this API appears below:

```
#include <unistd.h>

ssize_t     read(int fd, void *buf, size_t count)
```

return value

function name

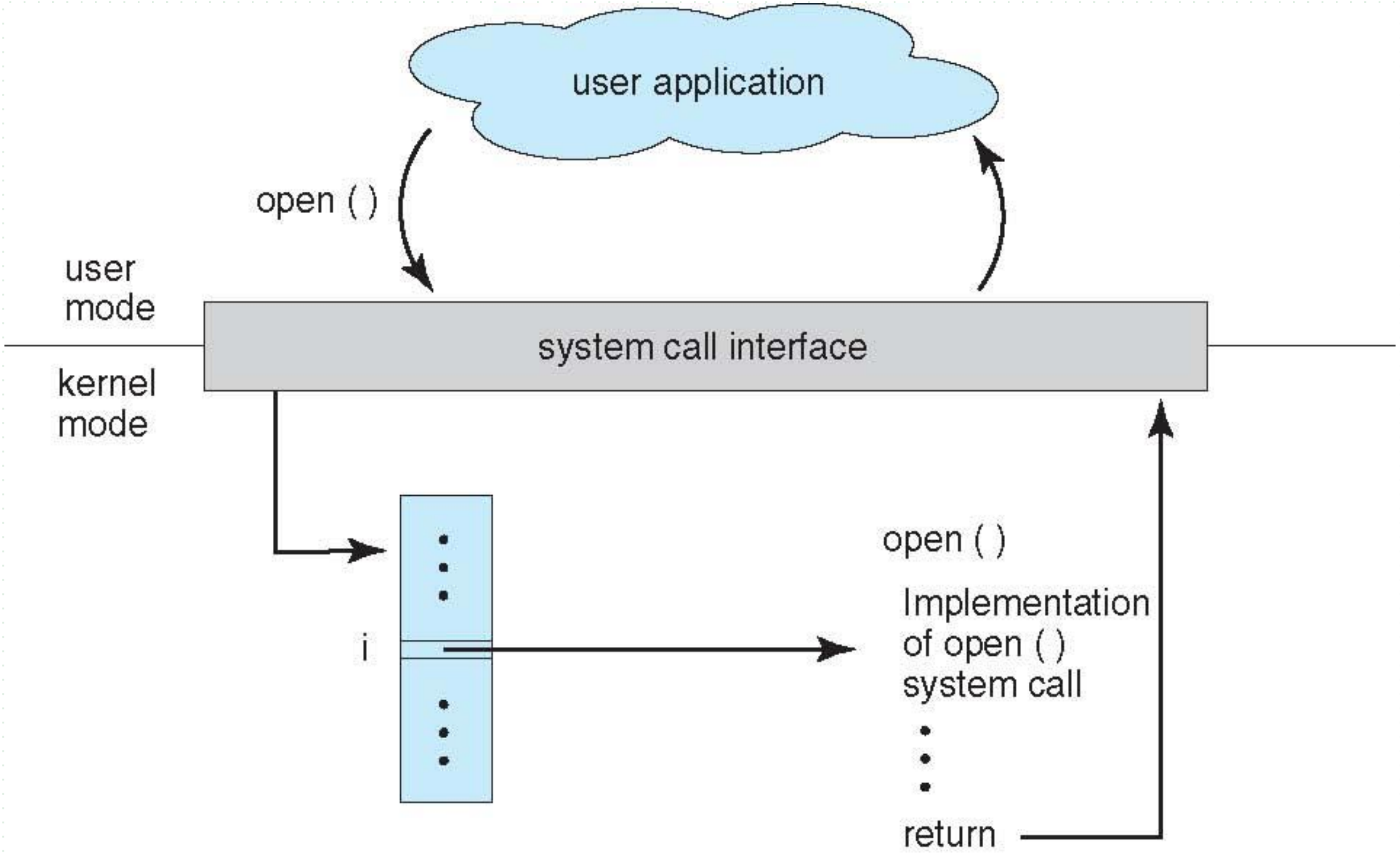
parameters

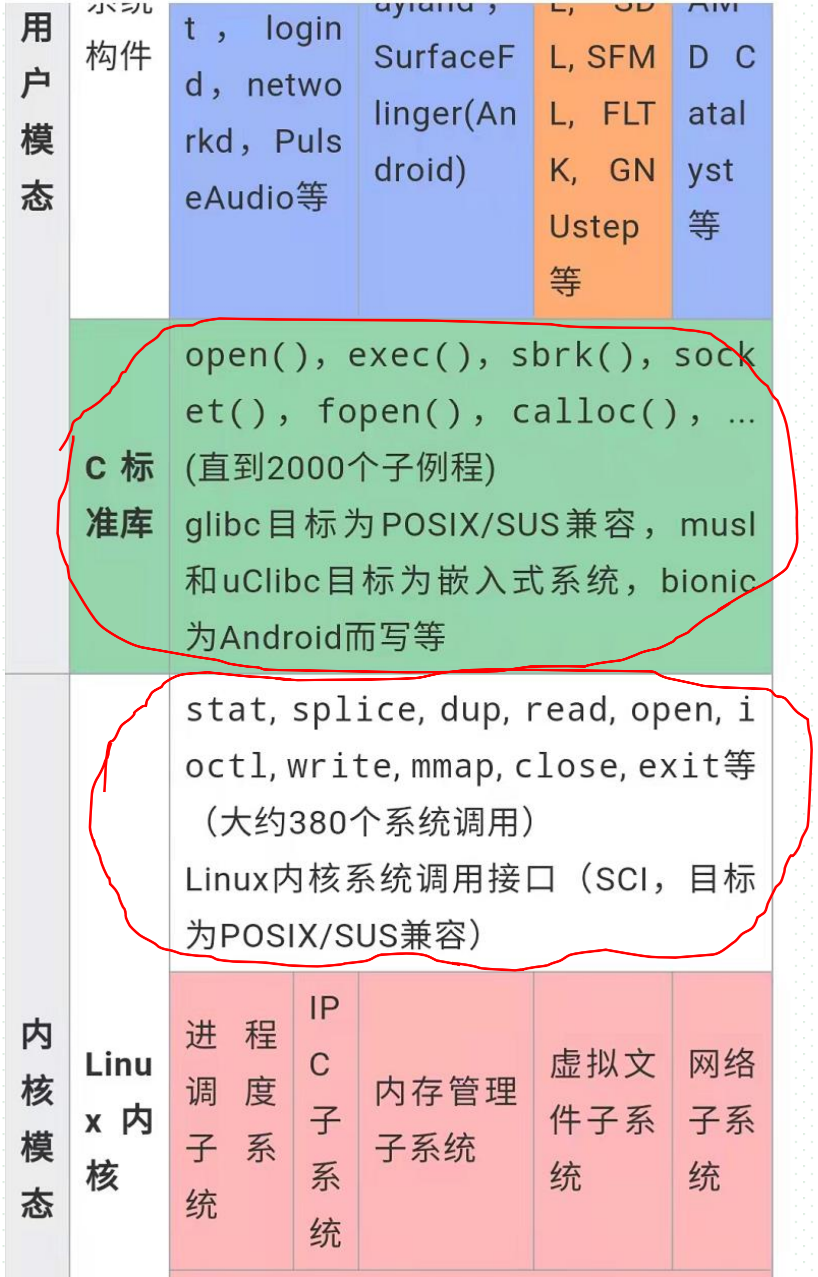
A program that uses the `read()` function must include the `unistd.h` header file, as this file defines the `ssize_t` and `size_t` data types (among other things). The parameters passed to `read()` are as follows:

- `int fd`—the file descriptor to be read
- `void *buf`—a buffer where the data will be read into
- `size_t count`—the maximum number of bytes to be read into the buffer

On a successful read, the number of bytes read is returned. A return value of 0 indicates end of file. If an error occurs, `read()` returns −1.

API – System Call – OS Relationship

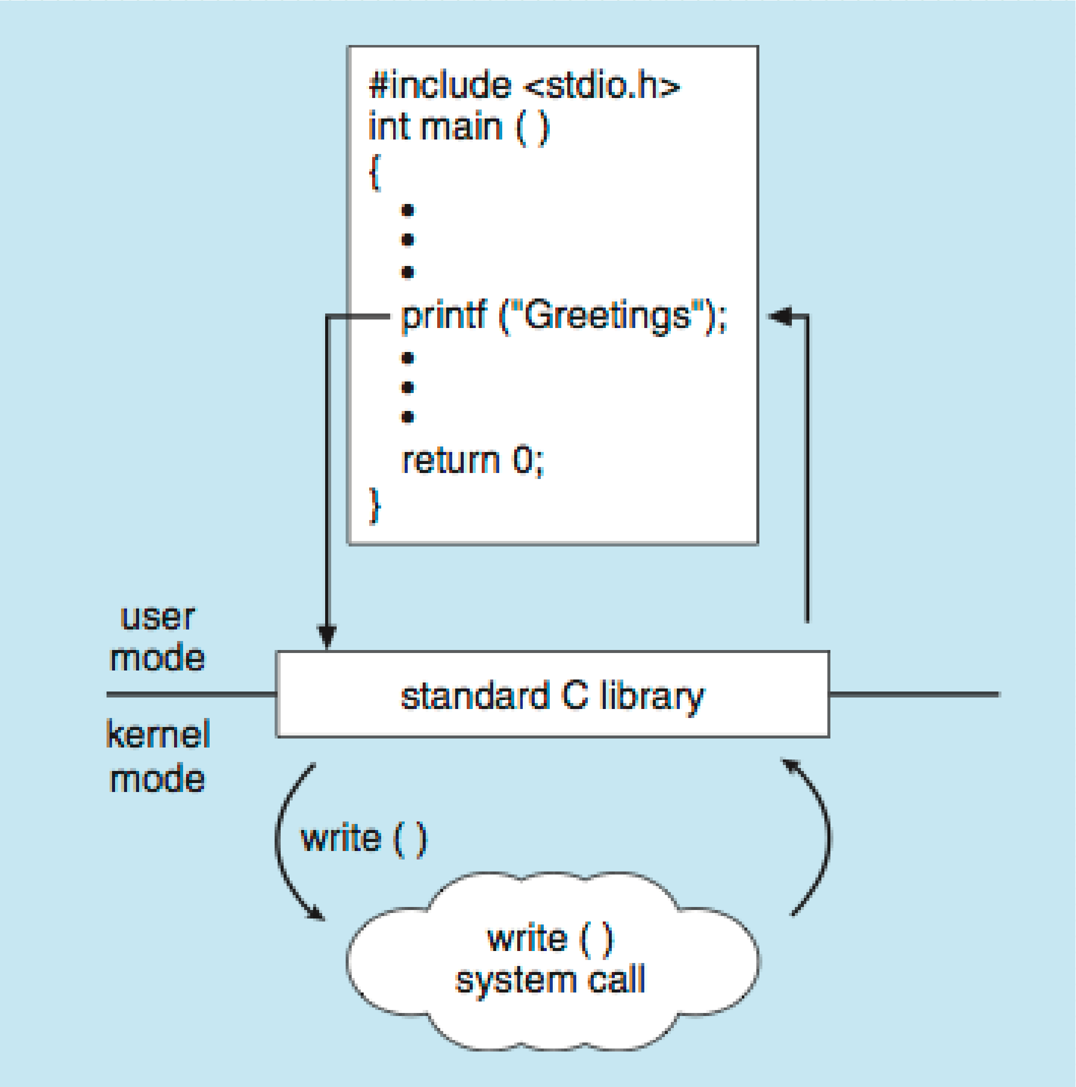




用户模态	系统构件	t，login d，networkd，PulseAudio等	ayland，SurfaceFlinger(Android)	L，SDL, SFML, FLTK, GNUstep等	AMD Catalyst等
	C标准库	open()，exec()，sbrk()，socket()，fopen()，calloc()，...(直到2000个子例程) glibc目标为POSIX/SUS兼容，musl和uClibc目标为嵌入式系统，bionic为Android而写等				
内核模态		stat, splice, dup, read, open, ioctl, write, mmap, close, exit等（大约380个系统调用） Linux内核系统调用接口（SCI，目标为POSIX/SUS兼容）				
	Linux内核	进程调度子系统	IPC子系统	内存管理子系统	虚拟文件子系统	网络子系统

Standard C Library Example

- C program invoking printf() library call, which calls write() system call

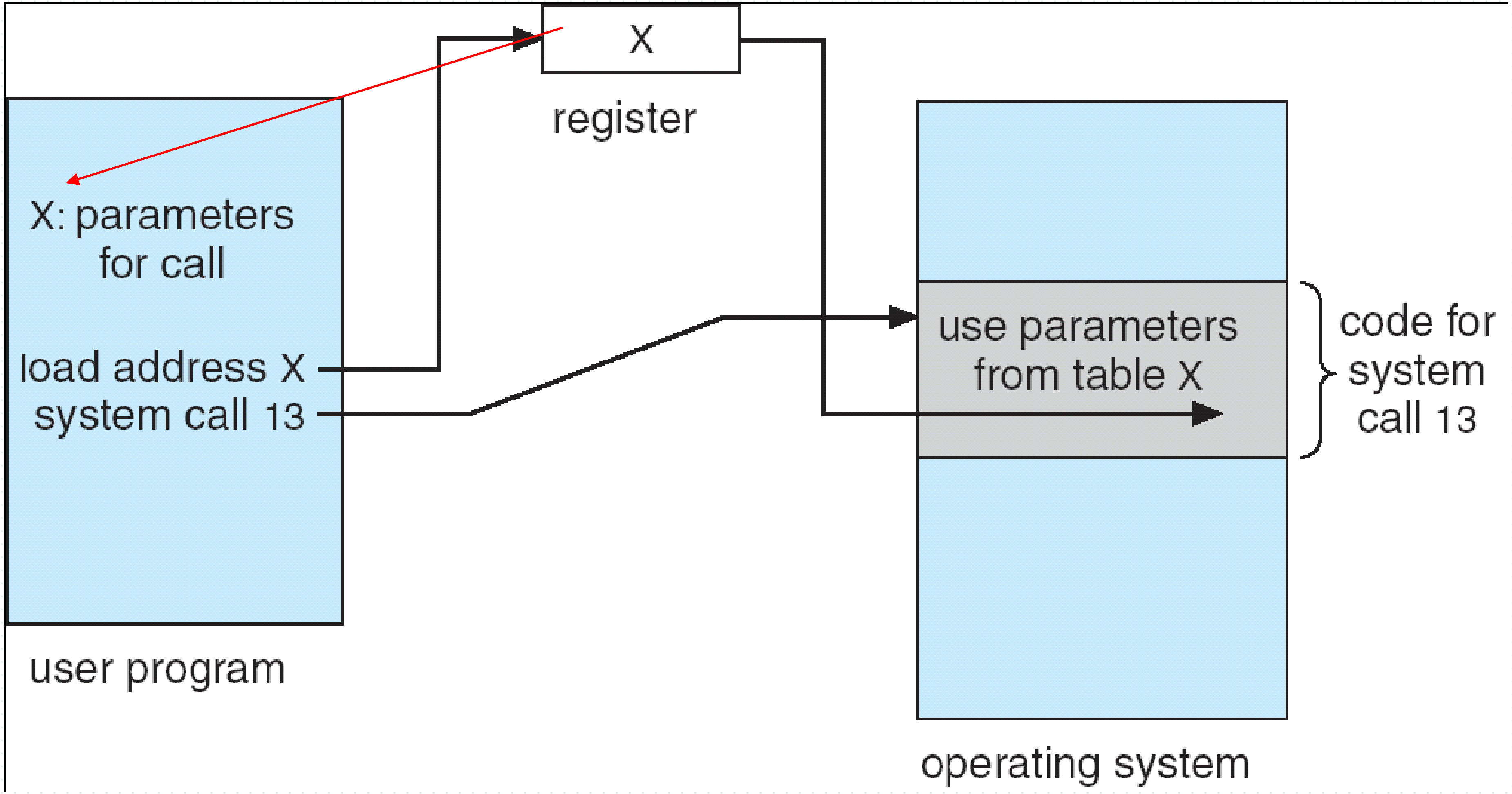


```
#include <stdio.h>
int main ( )
{
   •
   •
   •
   printf ("Greetings");
   •
   •
   •
   return 0;
}
```

System Call Parameter Passing

- Often, more information is required than simply identity of desired system call
  - Exact type and amount of information vary according to OS and call
- Three general methods used to pass parameters to the OS
  - Simplest: pass the parameters in **registers**
    - In some cases, may be more parameters than registers
  - Parameters stored in a **block**, or **table**, in memory, and address of block passed as a parameter in a register
    - This approach taken by Linux and Solaris
  - Parameters placed, or **pushed**, onto the **stack** by the program and **popped** off the stack by the operating system
  - Block and stack methods do not limit the number or length of parameters being passed

Parameter Passing via Table



2.4 Types of System Calls

- Categories
  - Process control
  - File management
  - Device management
  - Information maintenance
  - Communications
  - Protection

Types of System Calls

- Process control
  - create process, terminate process
  - end, abort
  - load, execute
  - get process attributes, set process attributes
  - wait for time
  - wait event, signal event
  - allocate and free memory
  - Dump memory (内存转储文件或者内存快照文件) if error
  - **Debugger** for determining **bugs, single step** execution
  - **Locks** for managing access to shared data between processes

- File management
  - create file, delete file
  - open, close file
  - read, write, reposition
  - get and set file attributes

- Device management
  - request device, release device
  - read, write, reposition
  - get device attributes, set device attributes
  - logically attach or detach devices

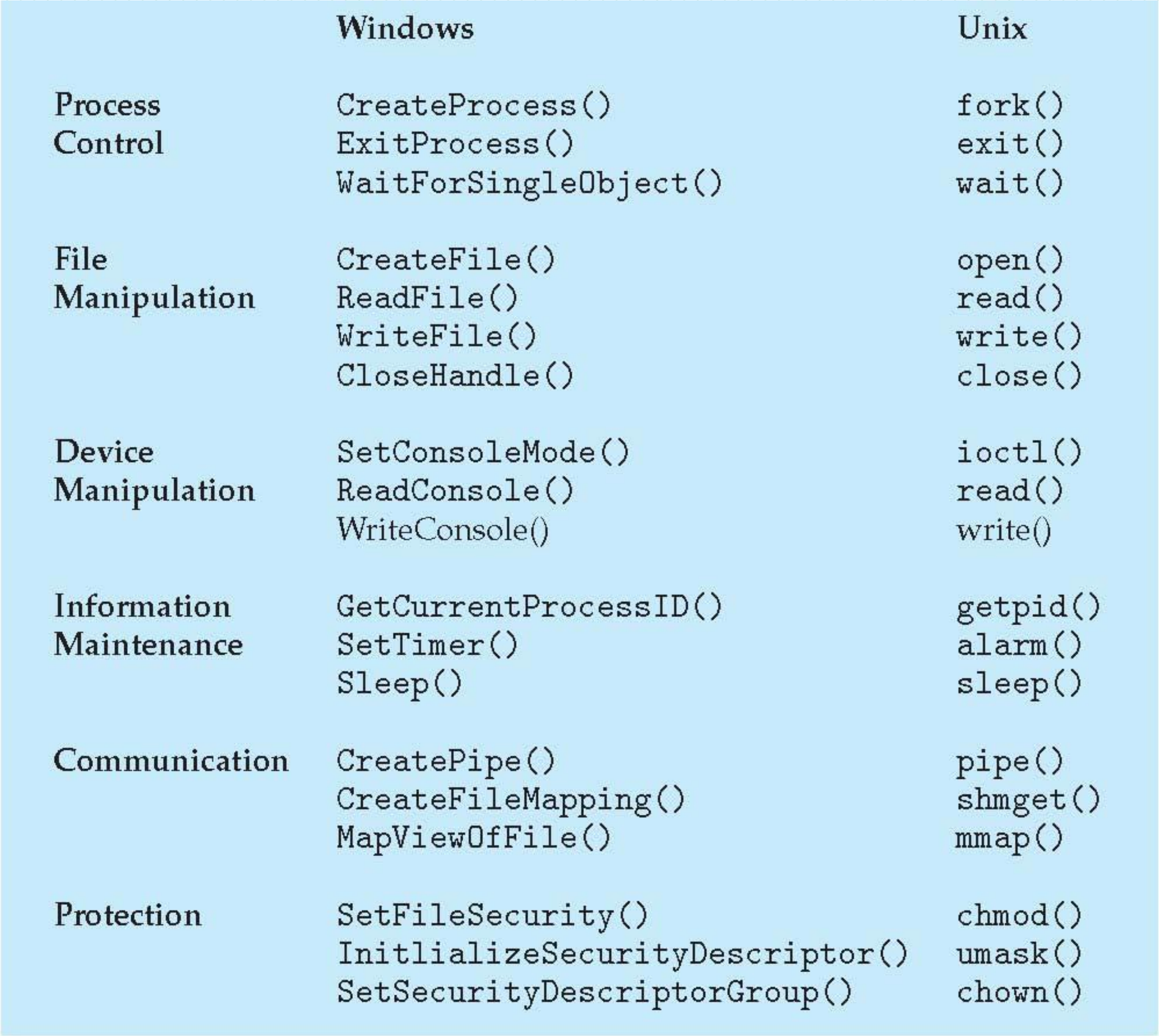
Types of System Calls

- Information maintenance
  - get time or date, set time or date
  - get system data, set system data
  - get and set process, file, or device attributes

- Communications
  - create, delete communication connection
  - send, receive messages if **message passing model** to **host name** or **process name**
    - From **client** to **server**
  - **Shared-memory model** create and gain access to memory regions
  - transfer status information
  - attach and detach remote devices

- Protection
  - Control access to resources
  - Get and set permissions
  - Allow and deny user access

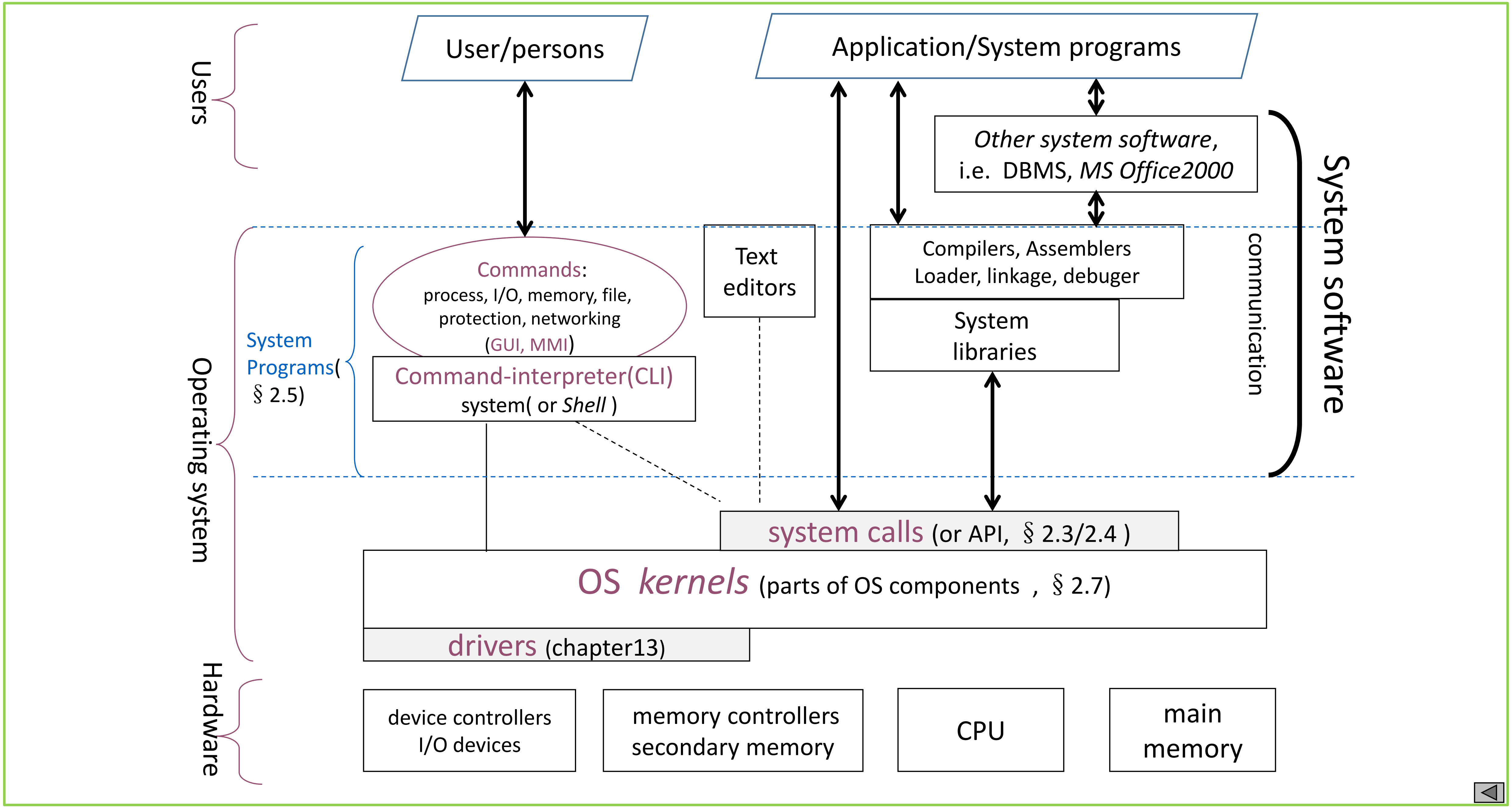
Examples of Windows and Unix System Calls



	Windows	Unix
Process Control	CreateProcess()	fork()
	ExitProcess()	exit()
	WaitForSingleObject()	wait()
File Manipulation	CreateFile()	open()
	ReadFile()	read()
	WriteFile()	write()
	CloseHandle()	close()
Device Manipulation	SetConsoleMode()	ioctl()
	ReadConsole()	read()
	WriteConsole()	write()
Information Maintenance	GetCurrentProcessID()	getpid()
	SetTimer()	alarm()
	Sleep()	sleep()
Communication	CreatePipe()	pipe()
	CreateFileMapping()	shmget()
	MapViewOfFile()	mmap()
Protection	SetFileSecurity()	chmod()
	InitlializeSecurityDescriptor()	umask()
	SetSecurityDescriptorGroup()	chown()

2.5 System Programs

- OS distribution version （发行版本)
  - kernel + system programs
  - e.g. Linux distribution versions openEuler, Kylin, AliOS, Ubuntu, Redhat
- System programs provide a convenient environment for program development and execution. They can be divided into:
  - File manipulation
  - Status information sometimes stored in a File modification
  - Programming language support
  - Program loading and execution
  - Communications
  - Background services
  - Application programs
- Most users′ view of the operation system is defined by system programs, not the actual system calls


Users
User/persons
Application/System programs
Other system software,
i.e. DBMS, MS Office2000
System software
communication
Operating system
System Programs(§2.5)
Commands:
process, I/O, memory, file, protection, networking
(GUI, MMI)
Command-interpreter(CLI)
system(or Shell)
Text editors
Compilers, Assemblers
Loader, linkage, debuger
System libraries
system calls (or API, §2.3/2.4)
OS kernels (parts of OS components , §2.7)
drivers (chapter13)
Hardware
device controllers
I/O devices
memory controllers
secondary memory
CPU
main memory

System Programs (Cont.)

- **File management** - Create, delete, copy, rename, print, dump, list, and generally manipulate files and directories

- **Status information**
  - Some ask the system for info - date, time, amount of available memory, disk space, number of users
  - Others provide detailed performance, logging, and debugging information
  - Typically, these programs format and print the output to the terminal or other output devices
  - Some systems implement a **registry (注册表)** - used to store and retrieve configuration information

System Programs (Cont.)

- **File modification**
  - Text editors to create and modify files
  - Special commands to search contents of files or perform transformations of the text
- **Programming-language support** - Compilers, assemblers, debuggers and interpreters sometimes provided
- **Program loading and execution**- Absolute loaders, relocatable loaders, linkage editors, and overlay-loaders, debugging systems for higher-level and machine language
- **Communications** - Provide the mechanism for creating virtual connections among processes, users, and computer systems
  - Allow users to send messages to one another′s screens, browse web pages, send electronic-mail messages, log in remotely, transfer files from one machine to another

System Programs (Cont.)

- **Background Services** （背景服务）
  - Launch at boot time
    - Some for system startup, then terminate
    - Some from system boot to shutdown
  - Provide facilities like disk checking, process scheduling, error logging, printing
  - Run in user context not kernel context
  - Known as **services**, **subsystems**, **daemons(守护程序)**
- **Application programs**
  - Don’t pertain to system (不属于系统)
  - Run by users
  - Not typically considered part of OS
  - Launched by command line, mouse click, finger poke

2.6 System Design and Implementation

- Design and Implementation of OS not "solvable", but some approaches have proven successful

- Internal structure of different Operating Systems can vary widely

- Start the design by defining goals and specifications

- Affected by choice of hardware, type of system

- **User** goals and **System** goals
  - User goals – operating system should be convenient to use, easy to learn, reliable, safe, and fast
  - System goals – operating system should be easy to design, implement, and maintain, as well as flexible, reliable, error-free, and efficient

Operating System Design and Implementation (Cont.)

- Important principle to separate
  - **Policy**: ***What*** will be done? (要做什么？)
  - **Mechanism**: ***How*** to do it? （如何做到？）
- Mechanisms determine how to do something, policies decide what will be done
- The separation of policy from mechanism is a very important principle, it allows maximum flexibility if policy decisions are to be changed later
- Specifying and designing an OS is highly creative task of **software engineering**

Implementation

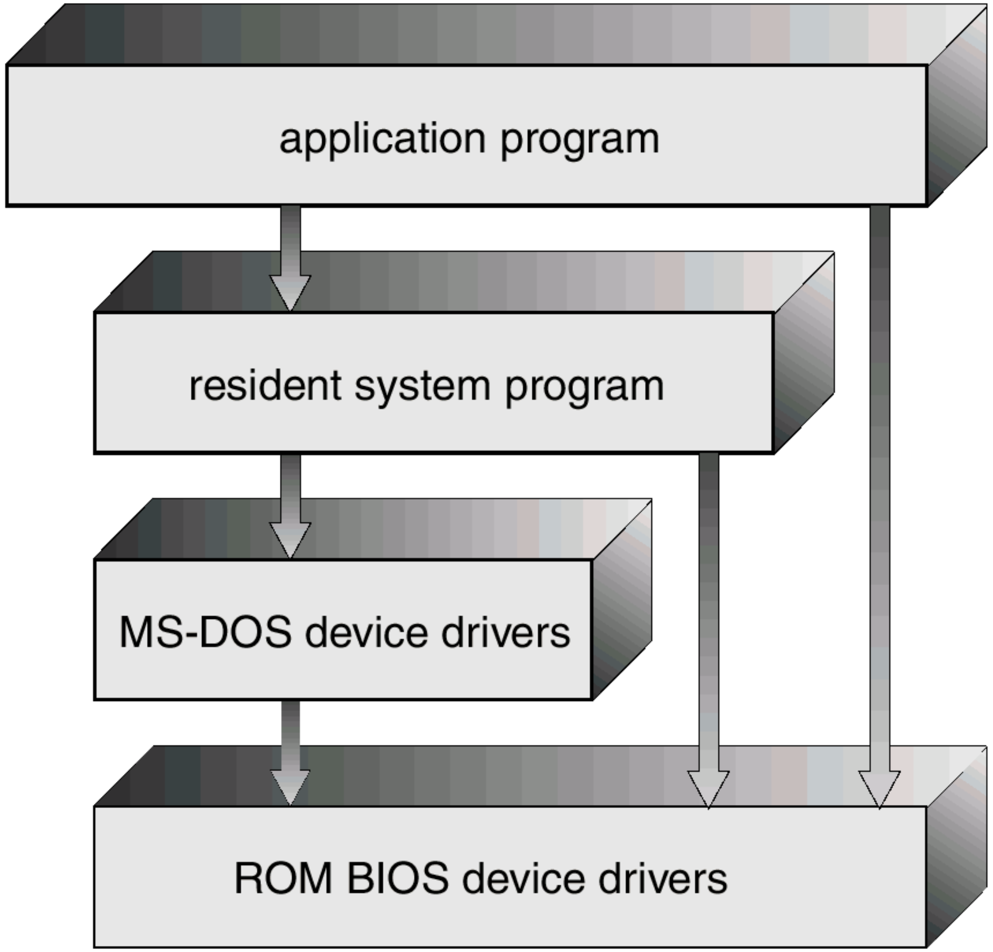
- Much variation
  - Early OSes in assembly language
  - Then system programming languages like Algol, PL/1
  - Now C, **C++**
    - C++ is first invented for coding Unix kernel in Bell Lab during late 1960s
- Actually usually a mix of languages
  - Lowest levels in assembly
  - Main body in C
  - Systems programs in C, C++, scripting languages like PERL, Python, shell scripts
  - **Rust** in Linux kernel programming
- More high-level language easier to **port (移植)** to other hardware
  - But slower

Operating System Structure

- General-purpose OS is very large program
- Various ways to structure ones
  - simple structure – MS-DOS
  - more complex – UNIX
    - monolithic/macro kernel
  - layered – an abstraction-OS2
  - microkernel –Mach
  - hybrid system-iOS, Android

Simple Structure: MS-DOS

- For the introduction to DOS, refer to Appendix 2.B ▷
- In view of architectures, MS-DOS aims to provide the most functionality in the least space
- coded in *assembly language*, not divided into modules
- although MS-DOS has some structure, its interfaces and levels of functionality are not well separated， i.e. ill-structured

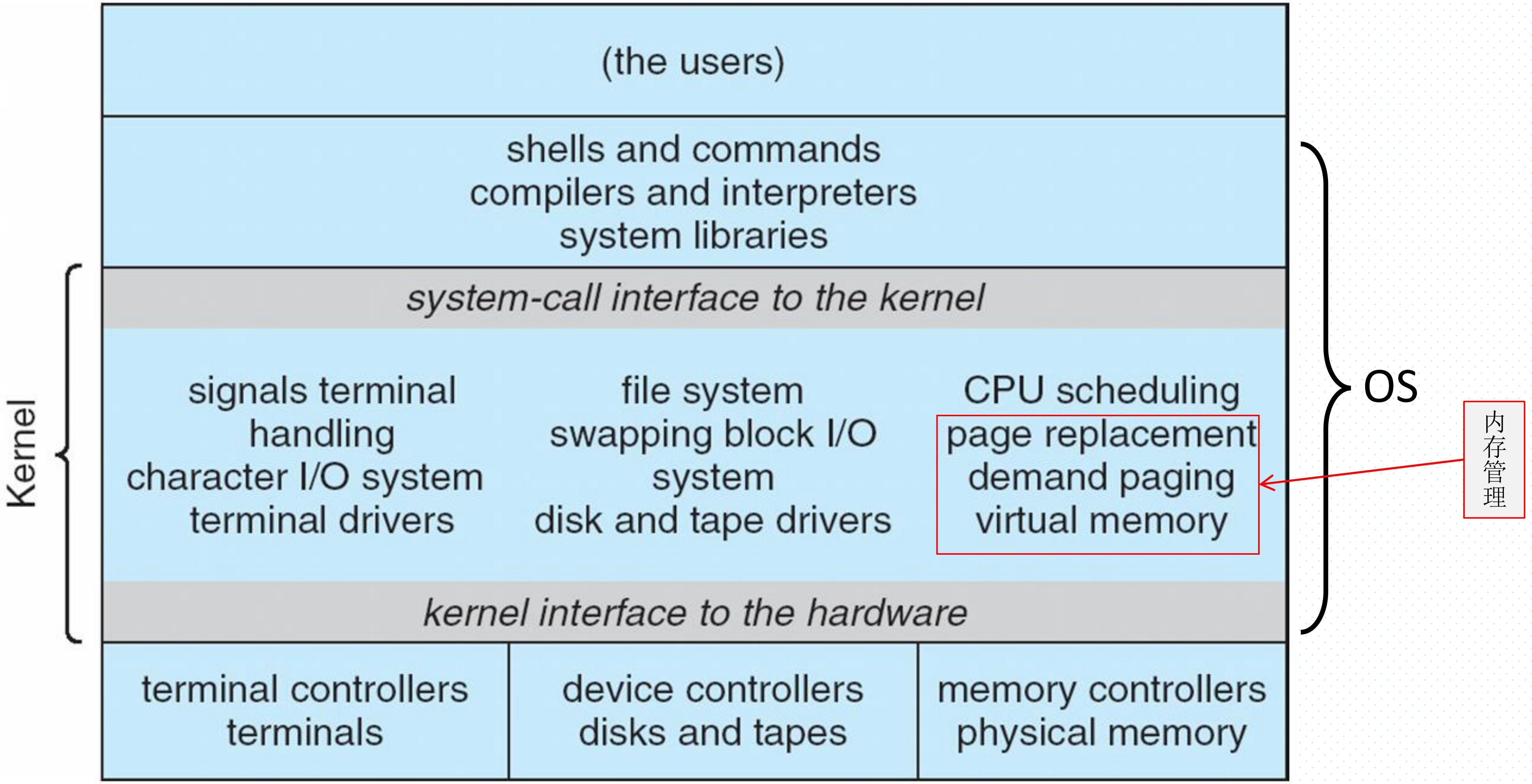


Simple Structure: Unix

- For the introduction to Unix, refer to Appendix 2.C ▷
- UNIX – limited by hardware functionality, the original UNIX operating system had limited structuring. The UNIX OS consists of two separable parts
  - Systems programs
  - The kernel
    - Consists of everything below the system-call interface and above the physical hardware
    - Provides the file system, CPU scheduling, memory management, and other operating-system functions;
    - a large number of functions for one level, ill-structured

Traditional UNIX System Structure: kernel + system programs

- Beyond simple but not fully layered



(the users)

shells and commands
compilers and interpreters
system libraries

system-call interface to the kernel

signals terminal handling
character I/O system
terminal drivers

file system
swapping block I/O system
disk and tape drivers

CPU scheduling
page replacement
demand paging
virtual memory

kernel interface to the hardware

terminal controllers
terminals

device controllers
disks and tapes

memory controllers
physical memory

Kernel

OS

内存管理

Unix System Structure

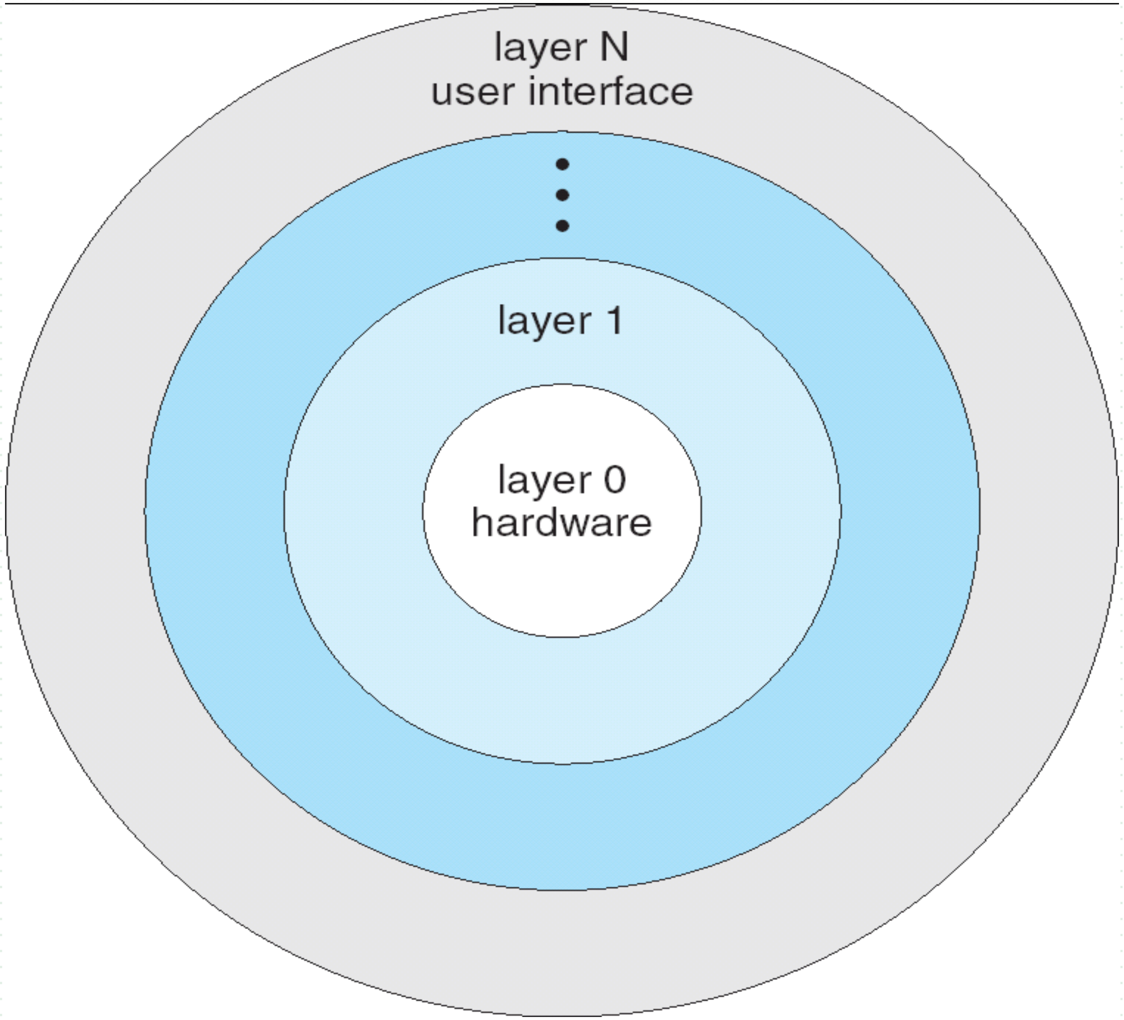
Simple Structure: Unix (Cont.)

- System calls define ***API*** as interfaces for programmers
- The set of system programs commonly available defines ***user interfaces***

- Demerits of Unix
  - too many functions in kernel, i.e. too many OS components in one layer, and these functions or components may interference with each others
    - too big a kernel, couples among components
  - big kernel makes UNIX difficult to enhance, as changes in one section could adversely affect other areas

Layered Approach: IBM OS2

- The operating system is divided into a number of layers (levels), each built on top of lower layers. The bottom layer (layer 0), is the hardware; the highest (layer N) is the user interface.
- With modularity, layers are selected such that each uses functions (operations) and services of only lower-level layers
- E.g. IBM OS/2
- Demerits
    - difficulties in layers definition, less efficient

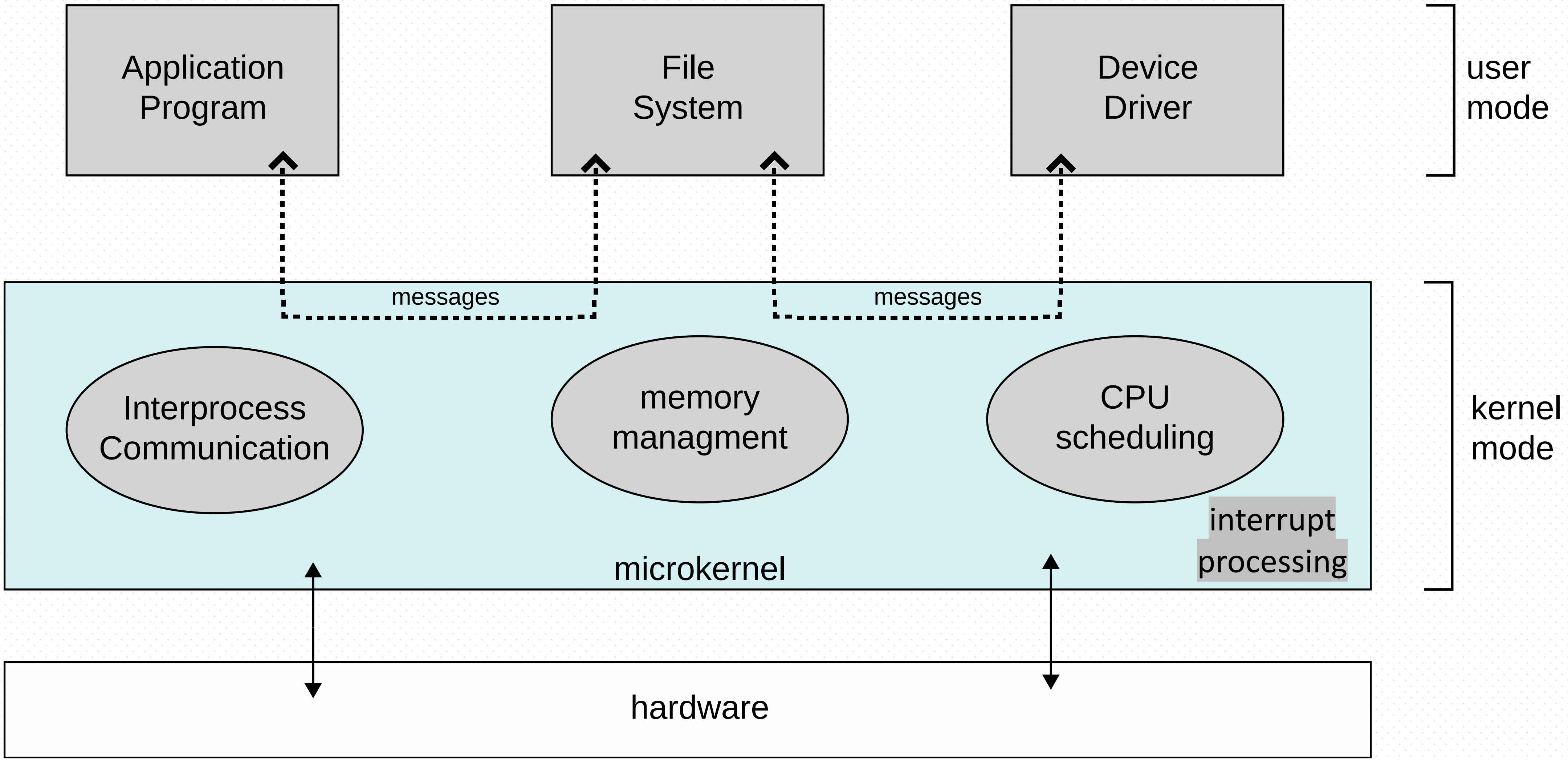


Microkernel System Structure: Mach

- OS is structured by removing all nonessential components from the kernel in the kernel mode, and implement them as system-level and user-level programs
  - moves as much from the kernel into "*user*" space/mode, thus result in a smaller kernel *called microkernel* running *in kernel mode*
  - the removed OS functional components run as server processes in *user* mode, and provide services such as file management services and driver services, etc
  - Communication takes place between user modules using **message passing**

- Benefits:
  - Easier to extend a microkernel
  - Easier to port the operating system to new architectures
  - More reliable (less code is running in kernel mode)
  - More secure
- Detriments:
  - Performance overhead of user space to kernel space communication
  - when a program in user space wishes to access a OS service, it indirectly interacts with the OS server process that provides this service in the same user space via exchanging message with the microkernel

Microkernel System Structure

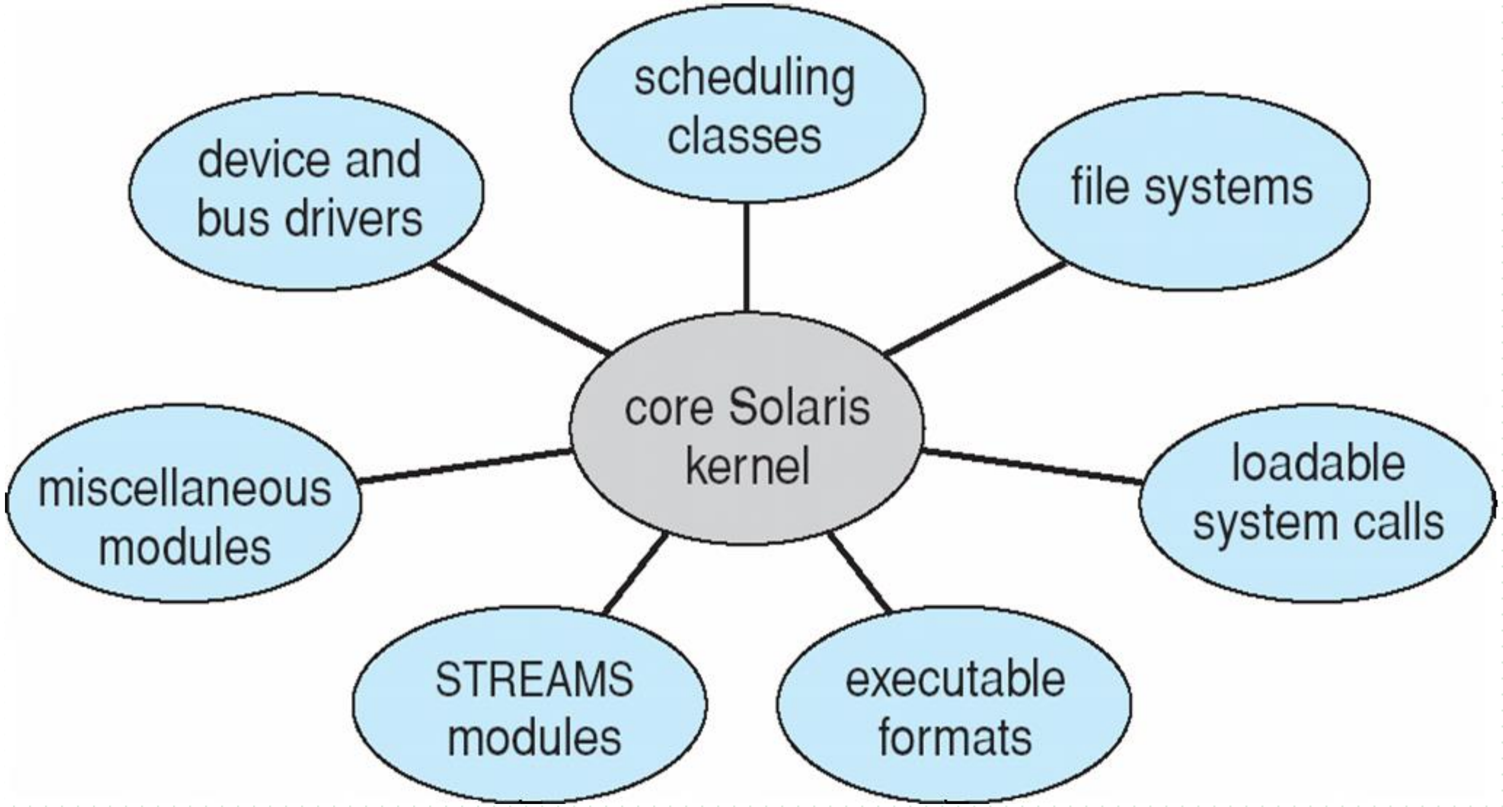


Microkernel System Structure

- OS = kernel + system programs
  - kernel = microkernel + OS server processes in the user mode
- **Mach** example of **microkernel**
  - first microkernel-based OS, developed at CMU, in mid-1980s
  - Mac OS X kernel (**Darwin**) partly based on Mach
- Other microkernel-based OS
  - Tru64 Unix, QNX
  - Huawei Harmony

Modules

- Many modern operating systems implement **loadable kernel modules**
  - Uses object-oriented approach
  - Each core component is separate
  - Each talks to the others over known interfaces
  - Each is loadable as needed within the kernel
- Overall, similar to layers but with more flexible
  - Linux, Solaris, etc



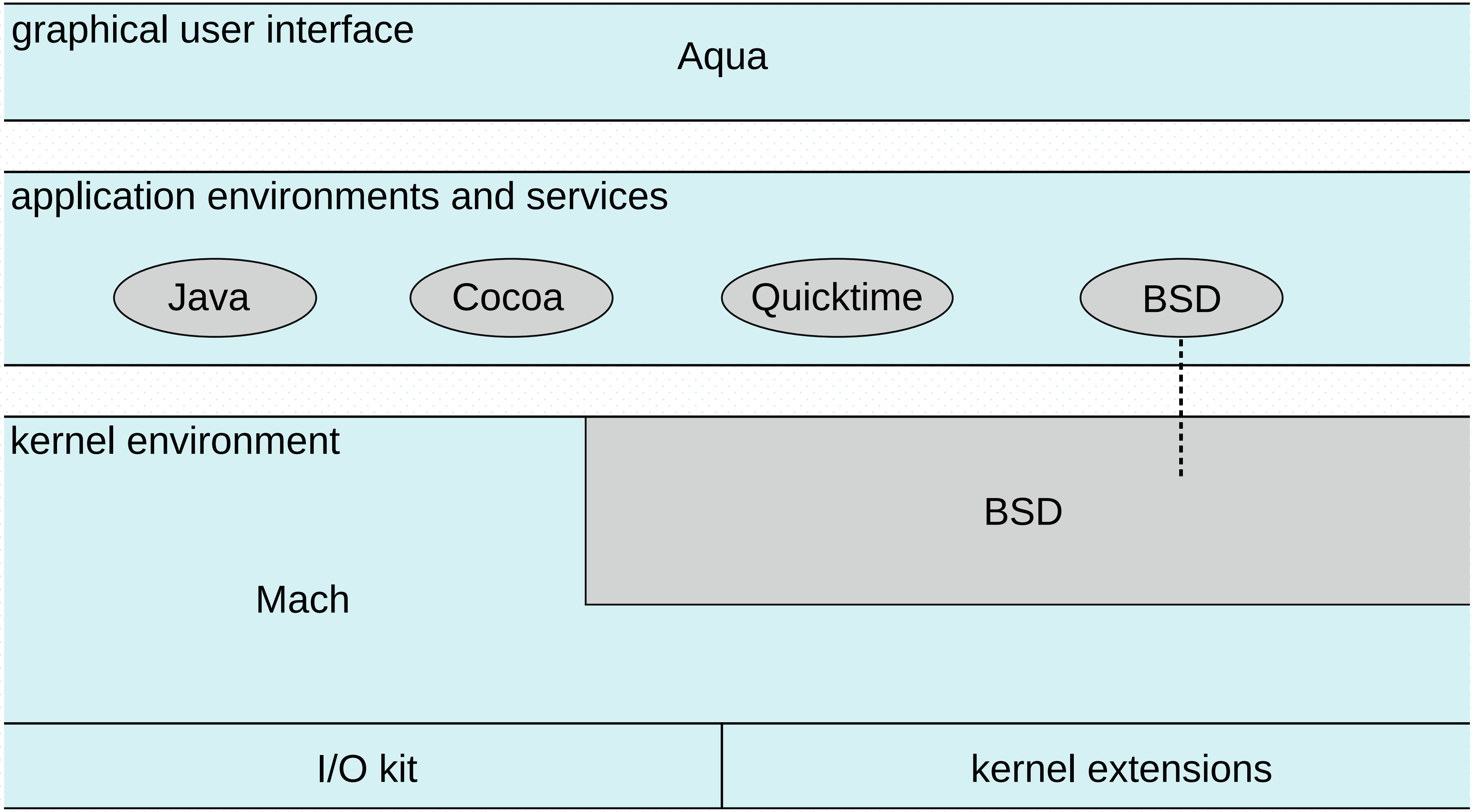
Killed by Oracle in 2017

Hybrid Systems

- Most modern operating systems are actually not one pure model
  - Hybrid combines multiple approaches to address performance, security, usability needs
  - Linux and Solaris kernels in kernel address space, so monolithic (宏内核), plus modular for dynamic loading of functionality
  - Windows mostly monolithic, plus microkernel for different subsystem ***personalities***
- Windows NT:
  - Kernel + system
  - Kernel = layered + microkernel
- Apple Mac OS X hybrid, layered, **Aqua** UI plus **Cocoa** programming environment
  - Below is kernel consisting of Mach microkernel and BSD Unix parts, plus I/O kit and dynamically loadable modules (called **kernel extensions**)

Mac OS X Structure

Apple Mac OS / iOS	Mach OS + Free BSD



graphical user interface

Aqua

application environments and services

Java

Cocoa

Quicktime

BSD

kernel environment

BSD

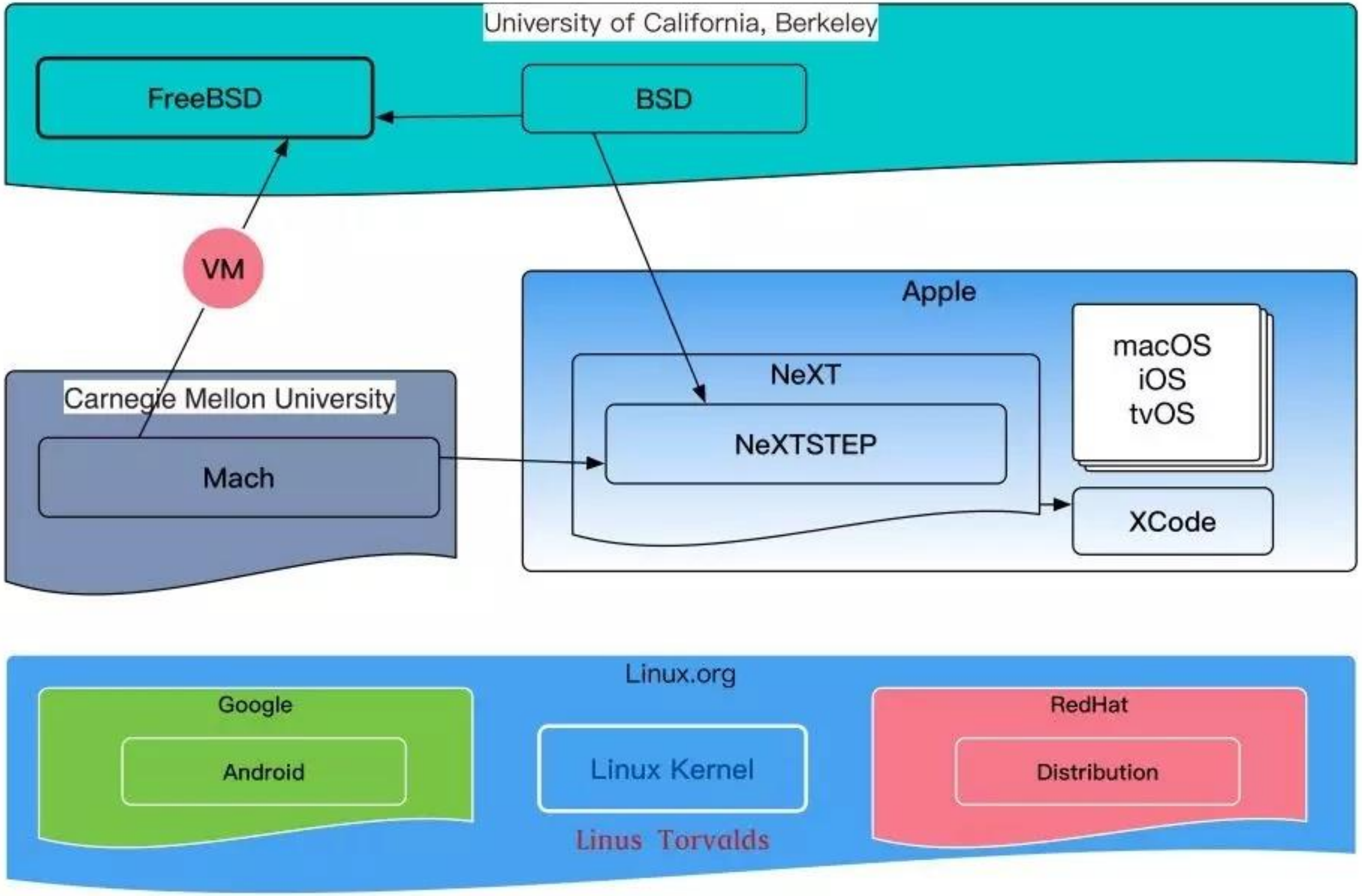
Mach

I/O kit

kernel extensions

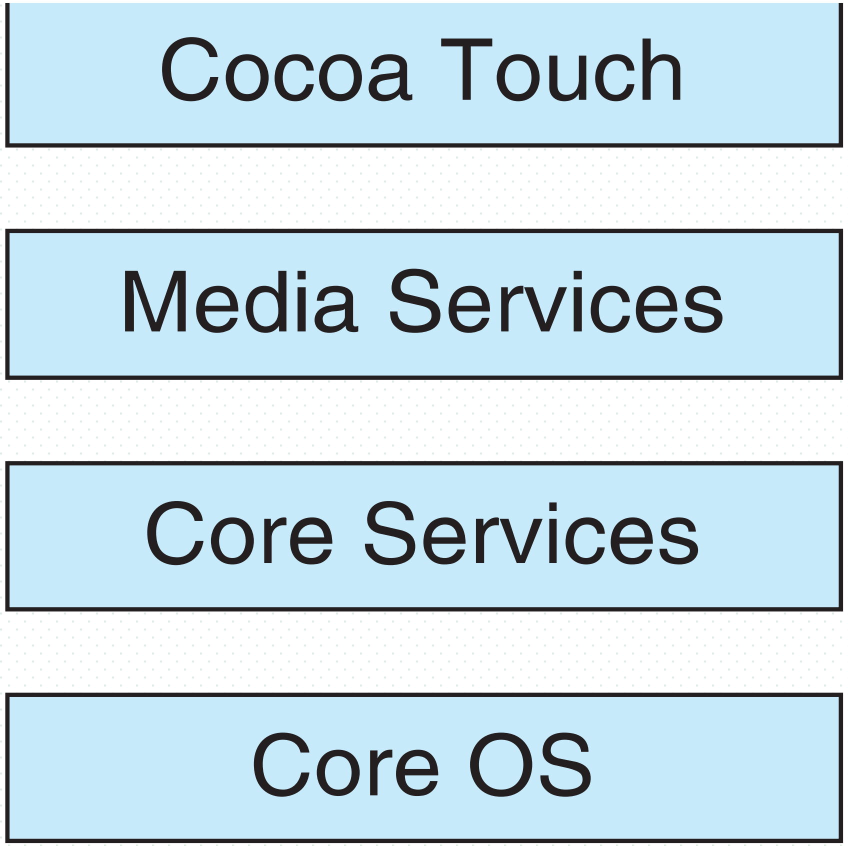
History: macOS vs Linux

@RiboseYim



iOS

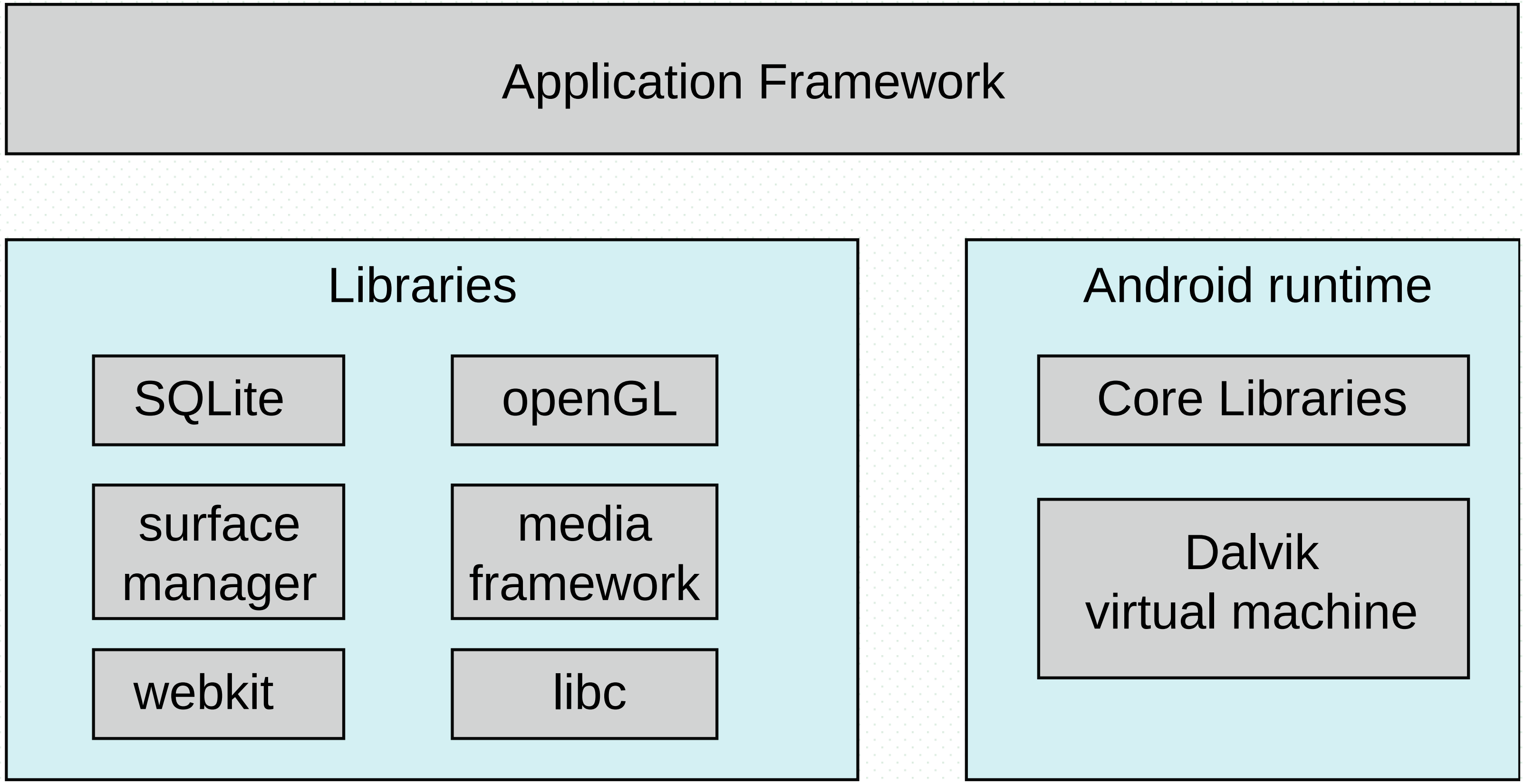
- Apple mobile OS for ***iPhone***, ***iPad***
  - Structured on Mac OS X, added functionality
  - Does not run OS X applications natively
    - Also runs on different CPU architecture (ARM vs. Intel)
  - **Cocoa Touch** Objective-C API for developing apps
  - **Media services** layer for graphics, audio, video
  - **Core services** provides cloud computing, databases
  - Core operating system, based on Mac OS X kernel



Android

- Developed by Open Handset Alliance (mostly Google)
  - Open Source + Closed Source
- Similar stack to IOS
- **Based on Linux kernel but modified**
  - Provides process, memory, device-driver management
  - Adds power management
  - AOSP: Android Open-Source Project
- Runtime environment includes core set of libraries and Dalvik virtual machine
  - Apps developed in Java plus Android API
    - Java class files compiled to Java bytecode then translated to executable than runs in Dalvik VM
- Libraries include frameworks for web browser (webkit), database (SQLite), multimedia, smaller libc

Android Architecture



Application Framework

Libraries

SQLite

openGL

surface manager

media framework

webkit

libc

Android runtime

Core Libraries

Dalvik virtual machine

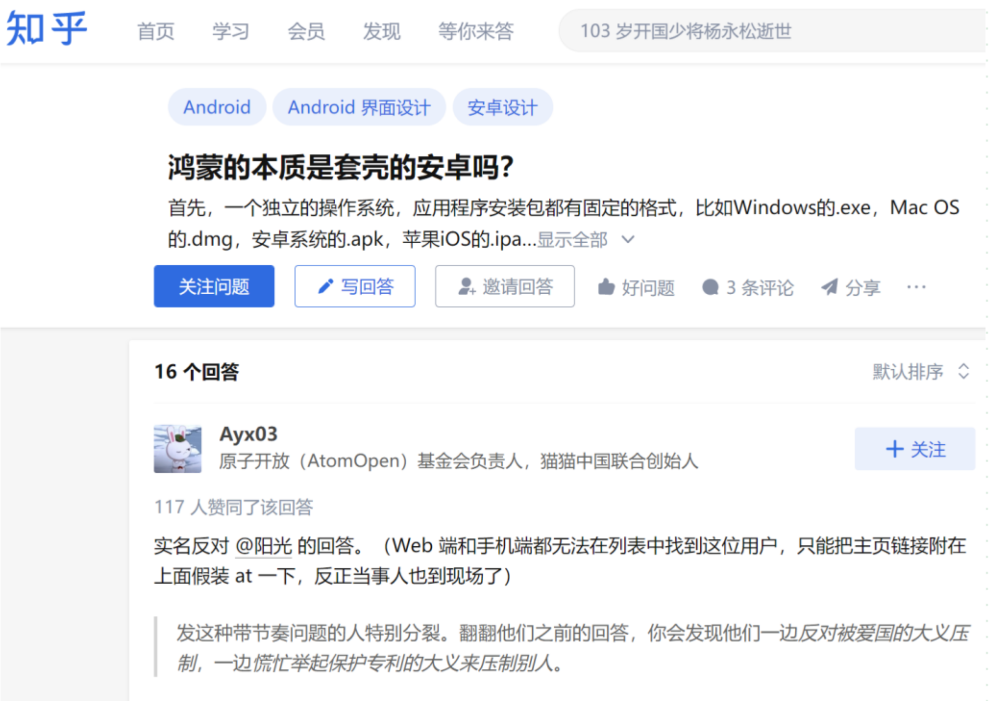
- Android = AOSP + GMS(Google Mobile Services) + … + Application
  - GMS, Google Mobile Services, controlled by Google, including Map services, E-mail services

Android vs HarmonyOS

- Android OS =
  - AOSP， Android Open Source Project, https://source.android.com
    - Android开源项目的开放部分， open source, linux-based
    - 谷歌、华为、三星等众多厂商共同开发，华为位列代码贡献全球前三名
    - free of charge
  - GMS， Google Mobile Services，
    - GooglePlay， Search， Search by Voice， Gmail， Contact Sync， Calendar Sync， Talk， Maps， Street View， YouTube， Android Market
    - controlled by Google， 闭源(closed source)， 使用许可， 付费
  - Device Drivers
  - Application Framework + APP

HarmonyOS vs Android

华为鸿蒙到底是不是安卓系统套了个壳?



鸿蒙的本质是套壳的安卓吗?

首先，一个独立的操作系统，应用程序安装包都有固定的格式，比如Windows的.exe，Mac OS的.dmg，安卓系统的.apk，苹果iOS的.ipa...显示全部

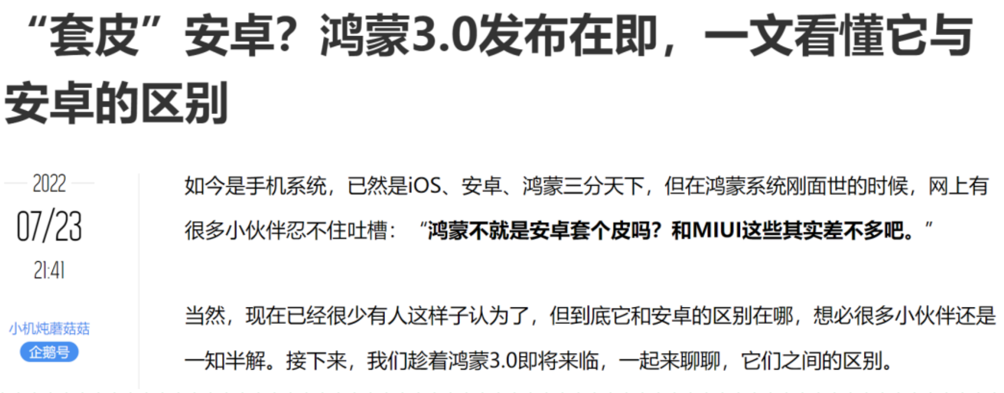
16 个回答 默认排序

Ayx03
原子开放（AtomOpen）基金会负责人，猫猫中国联合创始人

117 人赞同了该回答

实名反对 @阳光 的回答。（Web 端和手机端都无法在列表中找到这位用户，只能把主页链接附在上面假装 at 一下，反正当事人也到现场了）

> 发这种带节奏问题的人特别分裂。翻翻他们之前的回答，你会发现他们一边*反对被爱国的大义压制*，一边*慌忙举起保护专利的大义来压制别人*。



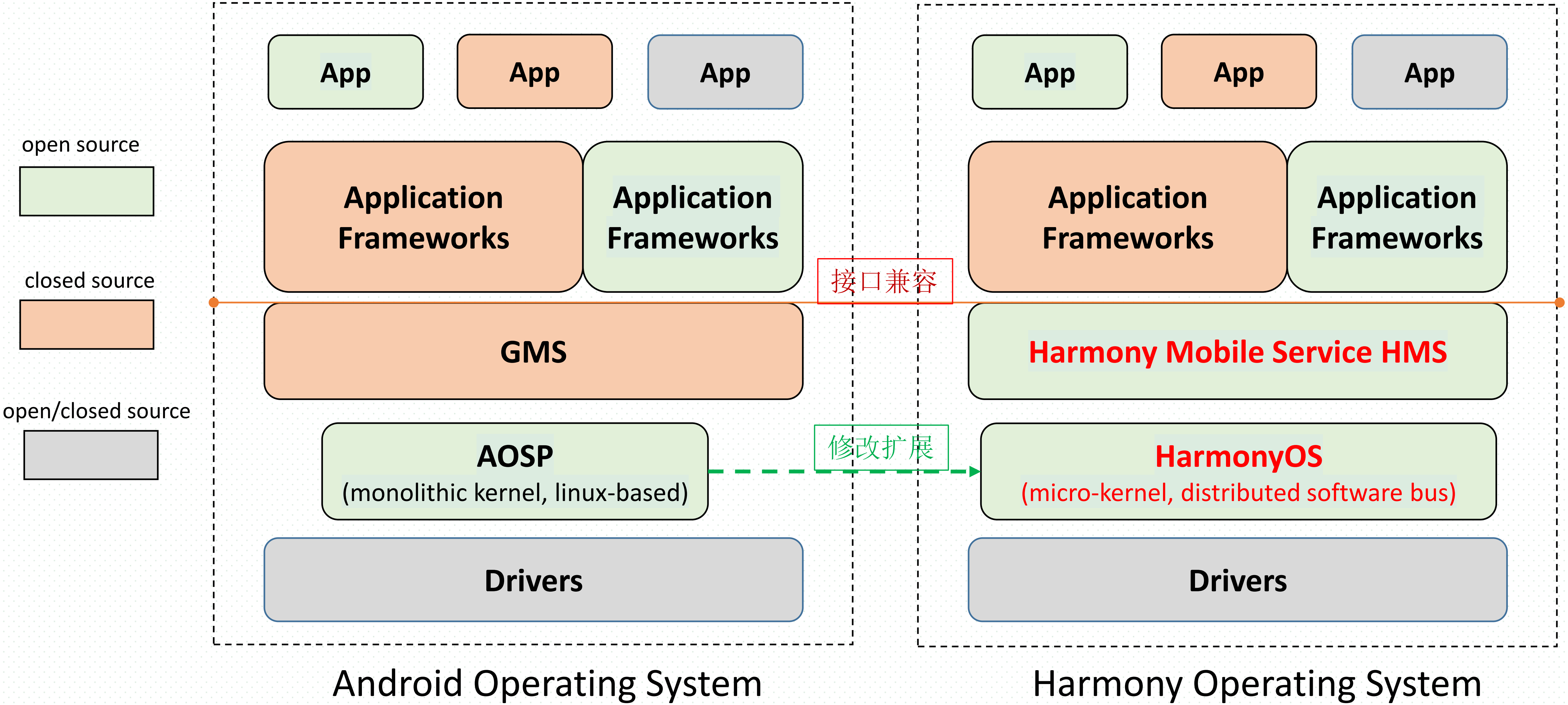
“套皮”安卓？鸿蒙3.0发布在即，一文看懂它与安卓的区别

2022
07/23
21:41
小机炖蘑菇菇
企鹅号

如今是手机系统，已然是iOS、安卓、鸿蒙三分天下，但在鸿蒙系统刚面世的时候，网上有很多小伙伴忍不住吐槽：“**鸿蒙不就是安卓套个皮吗？和MIUI这些其实差不多吧。**”

当然，现在已经很少有人这样子认为了，但到底它和安卓的区别在哪，想必很多小伙伴还是一知半解。接下来，我们趁着鸿蒙3.0即将来临，一起来聊聊，它们之间的区别。

Android vs HarmonyOS



open source 🟩

closed source 🟥

open/closed source ⬜

App

App

App

Application Frameworks

Application Frameworks

GMS

AOSP
(monolithic kernel, linux-based)

Drivers

接口兼容

修改扩展

App

App

App

Application Frameworks

Application Frameworks

Harmony Mobile Service HMS

HarmonyOS
(micro-kernel, distributed software bus)

Drivers

Android Operating System

宏内核架构

Harmony Operating System

微内核架构

Summary

OS	Features
DOS	simple/ill-structured
Unix	simple、macro/monolithic-kernel
Mach, HarmonyOS	**micro-kernel**
~~OS2~~	layered
Linux, ~~Solaris~~	module
Windows XP/NT/Server	(micro-) kernel + layered
Apple Mac OS / iOS	Mach OS + Free BSD
Android	Linux-based

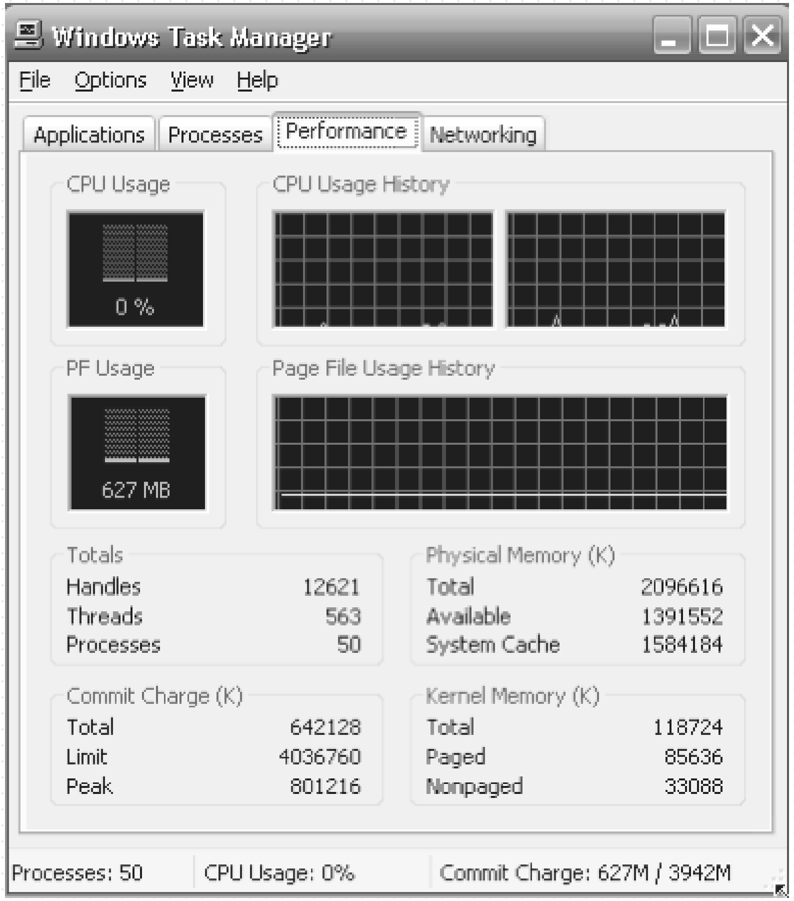
2.8 Operating-System Debugging

- **Debugging** is finding and fixing errors, or **bugs**
- OS generate **log files** containing error information
- Failure of an application can generate **core dump** file capturing memory of the process
- Operating system failure can generate **crash dump** file containing kernel memory
- Beyond crashes, performance tuning can optimize system performance
  - Sometimes using ***trace listings*** of activities, recorded for analysis
  - **Profiling** is periodic sampling of instruction pointer to look for statistical trends

Kernighan’s Law:
“Debugging is twice as hard as writing the code in the first place. Therefore, if you write the code as cleverly as possible, you are, by definition, not smart enough to debug it.”

Performance Tuning

- Improve performance by removing bottlenecks
- OS must provide means of computing and displaying measures of system behavior
- For example, “top” program or Windows Task Manager



CPU Usage: 0 %

PF Usage: 627 MB

Totals	
Handles	12621
Threads	563
Processes	50

Physical Memory (K)	
Total	2096616
Available	1391552
System Cache	1584184

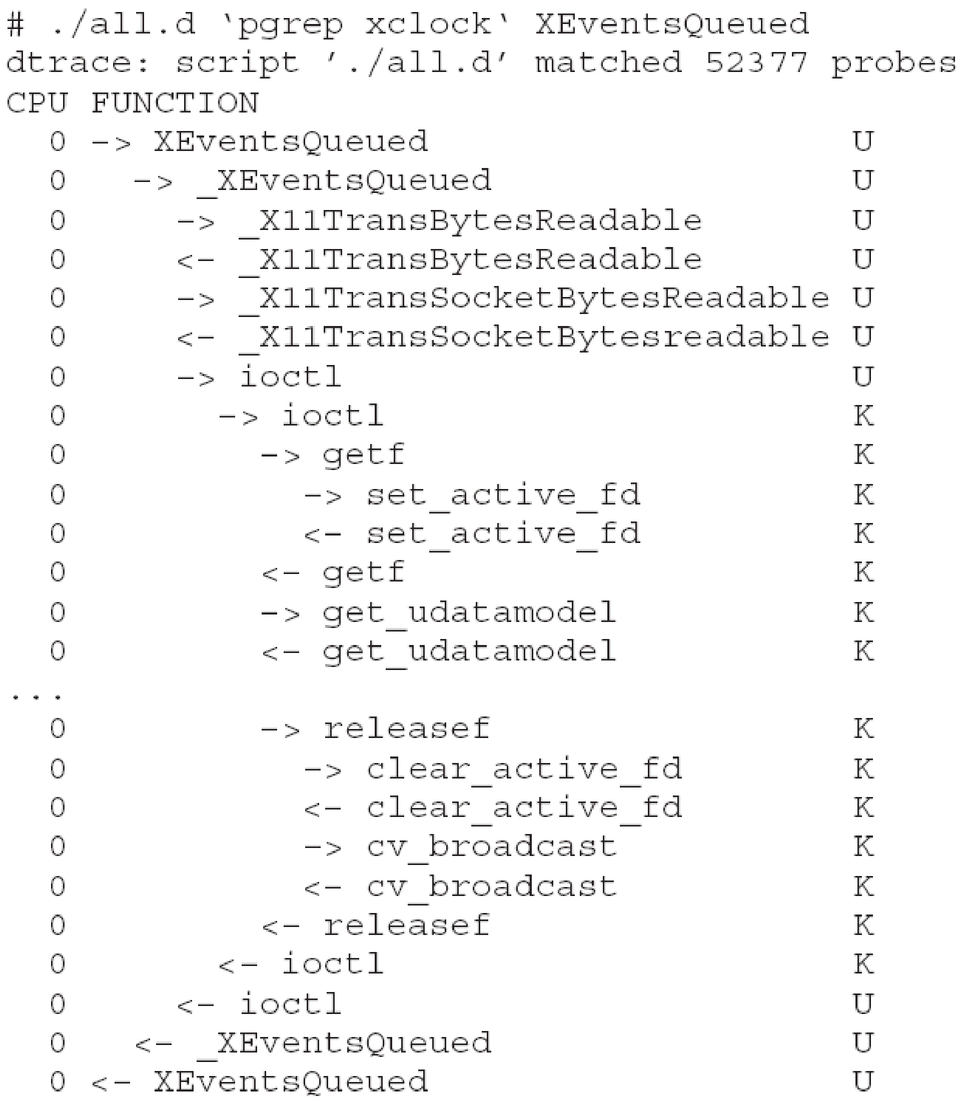
Commit Charge (K)	
Total	642128
Limit	4036760
Peak	801216

Kernel Memory (K)	
Total	118724
Paged	85636
Nonpaged	33088

Processes: 50 CPU Usage: 0% Commit Charge: 627M / 3942M

DTrace

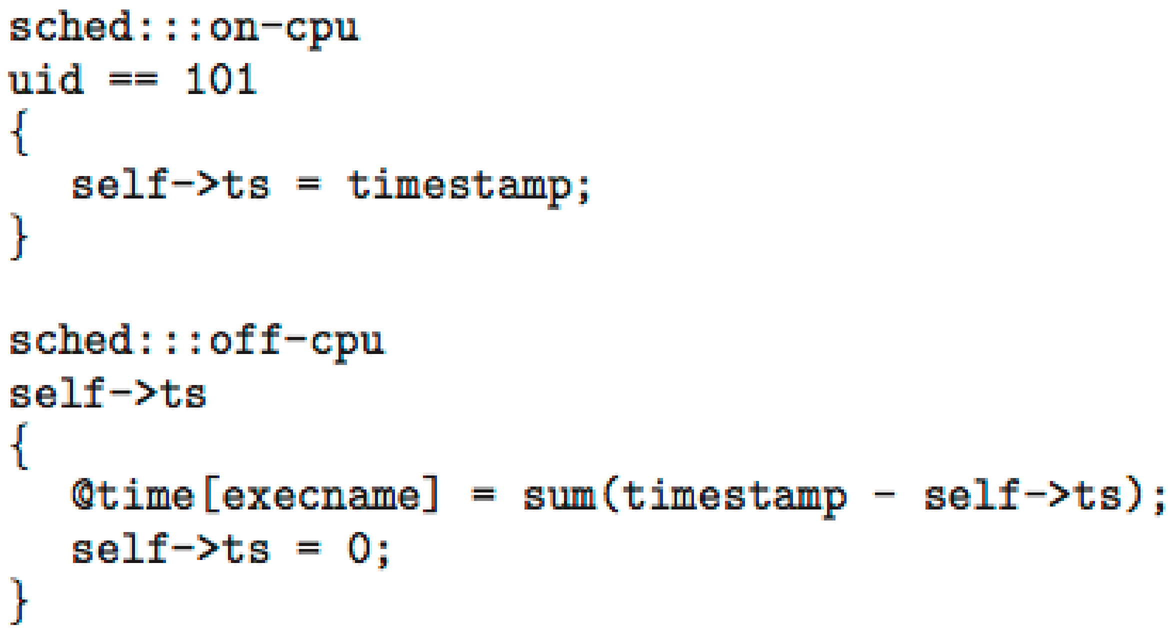
- DTrace tool in Solaris, FreeBSD, Mac OS X allows live instrumentation on production systems
- **Probes** fire when code is executed within a **provider**, capturing state data and sending it to **consumers** of those probes
- Example of following XEventsQueued system call move from libc library to kernel and back



```
# ./all.d ‘pgrep xclock‘ XEventsQueued
dtrace: script ’./all.d’ matched 52377 probes
CPU FUNCTION
  0 -> XEventsQueued                    U
  0   -> _XEventsQueued                 U
  0     -> _X11TransBytesReadable       U
  0     <- _X11TransBytesReadable       U
  0     -> _X11TransSocketBytesReadable U
  0     <- _X11TransSocketBytesreadable U
  0     -> ioctl                        U
  0       -> ioctl                      K
  0         -> getf                     K
  0           -> set_active_fd          K
  0           <- set_active_fd          K
  0         <- getf                     K
  0         -> get_udatamodel           K
  0         <- get_udatamodel           K
...
  0         -> releasef                 K
  0           -> clear_active_fd        K
  0           <- clear_active_fd        K
  0           -> cv_broadcast           K
  0           <- cv_broadcast           K
  0         <- releasef                 K
  0       <- ioctl                      K
  0     <- ioctl                        U
  0   <- _XEventsQueued                 U
  0 <- XEventsQueued                    U
```

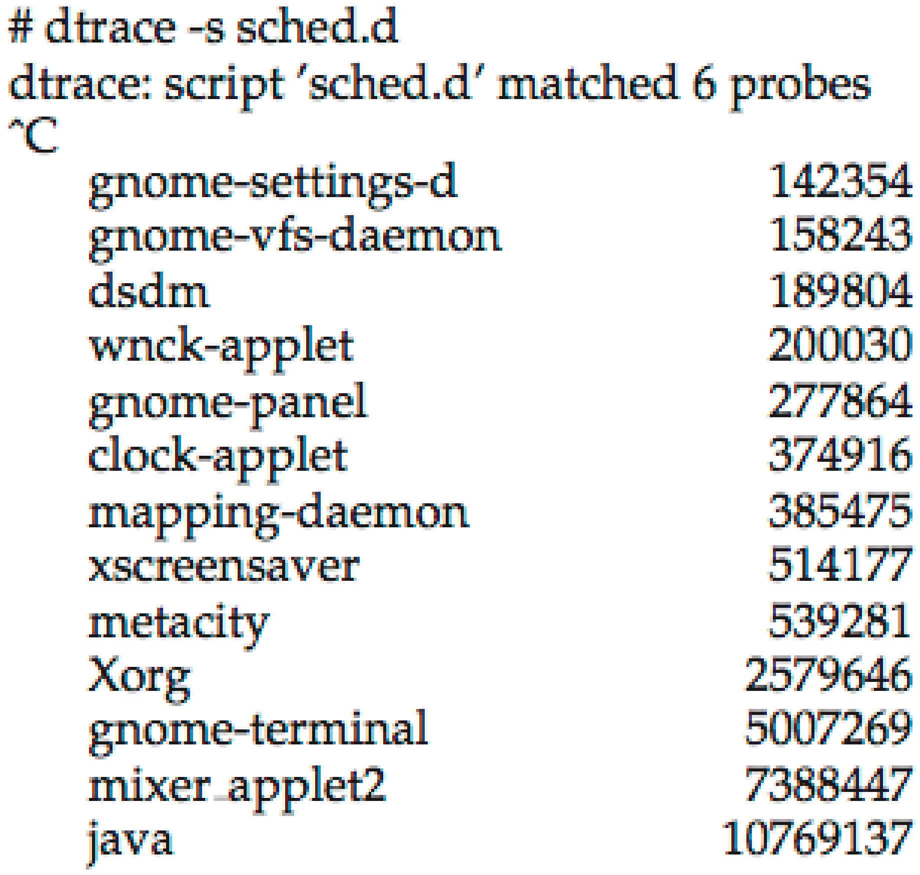
Dtrace (Cont.)

- DTrace code to record amount of time each process with UserID 101 is in running mode (on CPU) in nanoseconds



```
sched:::on-cpu
uid == 101
{
   self->ts = timestamp;
}

sched:::off-cpu
self->ts
{
   @time[execname] = sum(timestamp - self->ts);
   self->ts = 0;
}
```



```
# dtrace -s sched.d
dtrace: script 'sched.d' matched 6 probes
^C
    gnome-settings-d           142354
    gnome-vfs-daemon           158243
    dsdm                       189804
    wnck-applet                200030
    gnome-panel                277864
    clock-applet               374916
    mapping-daemon             385475
    xscreensaver               514177
    metacity                   539281
    Xorg                      2579646
    gnome-terminal            5007269
    mixer_applet2             7388447
    java                     10769137
```

Figure 2.21 Output of the D code.

2.9 Operating System Generation

- Operating systems are designed to run on any of a class of machines; the system must be configured for each specific computer site
- **SYSGEN** program obtains information concerning the specific configuration of the hardware system
    - Used to build system-specific compiled kernel or system-tuned
    - Can general more efficient code than one general kernel

System Boot

- When power initialized on system, execution starts at a fixed memory location
  - Firmware ROM used to hold initial boot code
- Operating system must be made available to hardware so hardware can start it
  - Small piece of code – **bootstrap loader**, stored in **ROM** or **EEPROM** locates the kernel, loads it into memory, and starts it
  - Sometimes two-step process where **boot block** at fixed location loaded by ROM code, which loads bootstrap loader from disk
- Common bootstrap loader, **GRUB**, allows selection of kernel from multiple disks, versions, kernel options
- Kernel loads and system is then **running**
- Two booting modes
  - BIOS
  - UEFI

System Boot: BIOS

- Booting (系统引导/启动/加载）
  - staring a computer by loading OS kernel
- BIOS
  - acronym for Basic Input Output System on PC-compatible computers, the set of routines that test hardware at startup, start OS, and support the transfer of data among hardware devices
  - BIOS is stored in ROM (firmware) , so that it can be executed first of all when the computer is turned on.
  - also known as **bootstrap** or a part of bootstrap
- BIOS主要功能
  - 初始化硬件：检测，初始化
  - 提供硬件的软件抽象
    - BIOS提供了主板、主板上外插设备的软件抽象。通过探测、Training和枚举，BIOS掌握系统硬件的信息
    - 通过几组详细定义好的接口，将硬件信息抽象后传递给OS，这些信息包括SMBIOS、ACPI表（ACPI与UEFI）、内存映射表（E820或者UEFI运行时）等等
    - 通过这层映射，操作系统可以适配到各种机型和硬件
- BIOS now evolves into **UEFI**

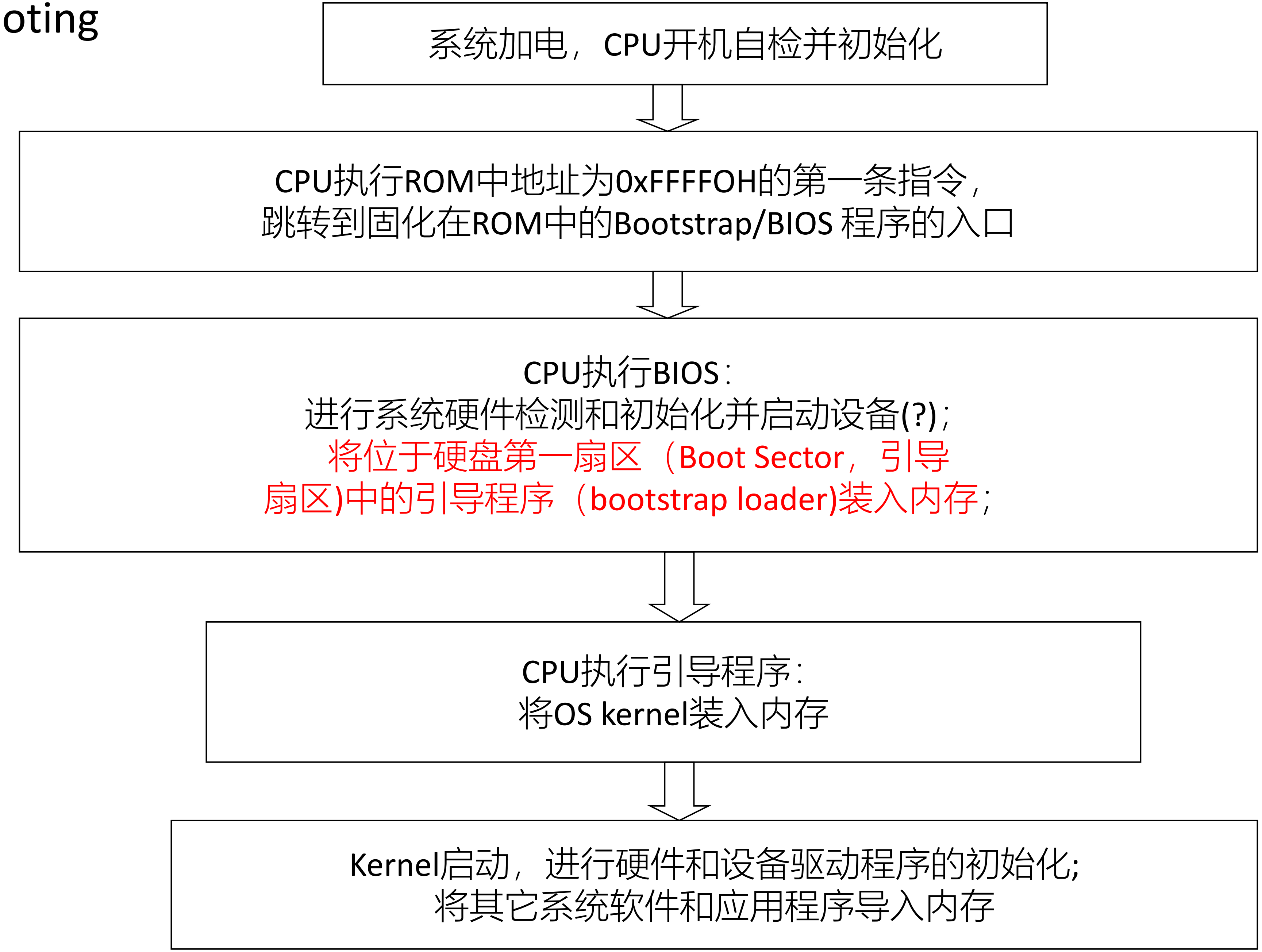
System Boot: BIOS

- Step1. 系统加电， CPU复位， CPU开机自检 (Power On Self Test, POST)并初始化自身，如设置CPU寄存器的初值， 包括程序计数器指针PC；进入内核模式
- Step2. CPU根据初始化后的PC= e.g. 0xFFFF0H（或者 ??），执行第一条指令
  - 该指令的地址为0xFFFF0H，固化在ROM中，为跳转指令
  - 通过执行此指令， CPU进入到固化在ROM中的Bootstrap/BIOS程序的入口地址
- Setp3. 采用*one-step或two-step*法， CPU执行ROM中的Bootstrap/BIOS程序，进行系统硬件检测(RAM、键盘、显示器、软硬磁盘等）和初始化，并将OS kernel装入内存
- Step4. Kernel启动，进一步进行（部分）硬件和设备驱动程序的初始化; 然后将其它系统软件和应用程序导入内存

- 系统启动时CPU模式变化
  - 内核模式： *ROM*: 0xFFFF0H(?): *Jump* →*ROM*: *bootstrap* → *RAM*: *kernel*

    →用户模式： *RAM*:*系统软件*→ *RAM*:*应用软件*

- PC two-step booting



系统加电，CPU开机自检并初始化

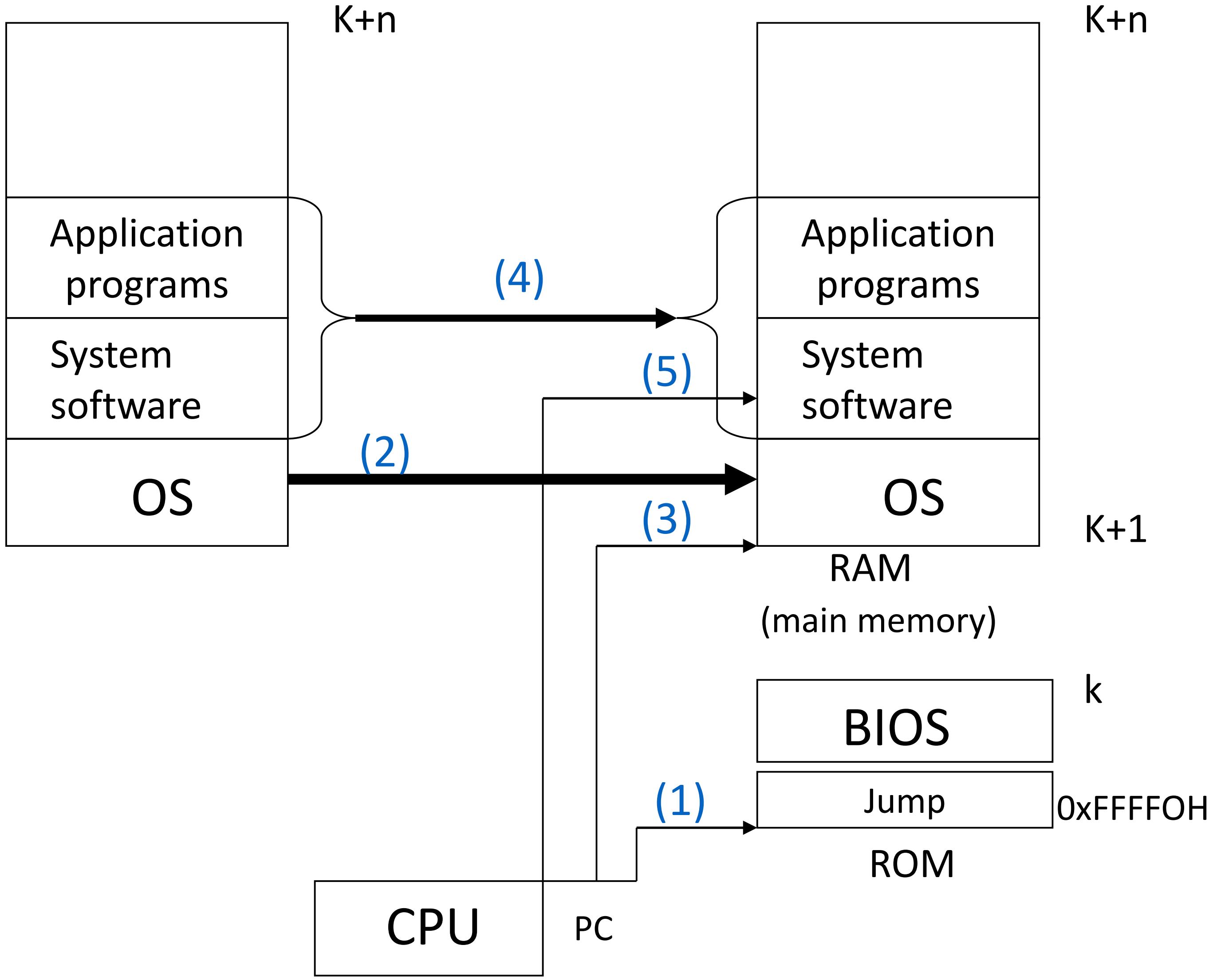
CPU执行ROM中地址为0xFFFFOH的第一条指令，
跳转到固化在ROM中的Bootstrap/BIOS 程序的入口

CPU执行BIOS：
进行系统硬件检测和初始化并启动设备(?)；
将位于硬盘第一扇区（Boot Sector，引导
扇区)中的引导程序（bootstrap loader)装入内存；

CPU执行引导程序：
将OS kernel装入内存

Kernel启动，进行硬件和设备驱动程序的初始化;
将其它系统软件和应用程序导入内存

One-step booting



System Boot: BIOS

计算机是如何启动的？
https://zhuanlan.zhihu.com/p/43802526?utm_source=com.android.email&utm_medium=social&utm_oi=679111968938397696

计算机是如何启动的?

System Boot: UEFI

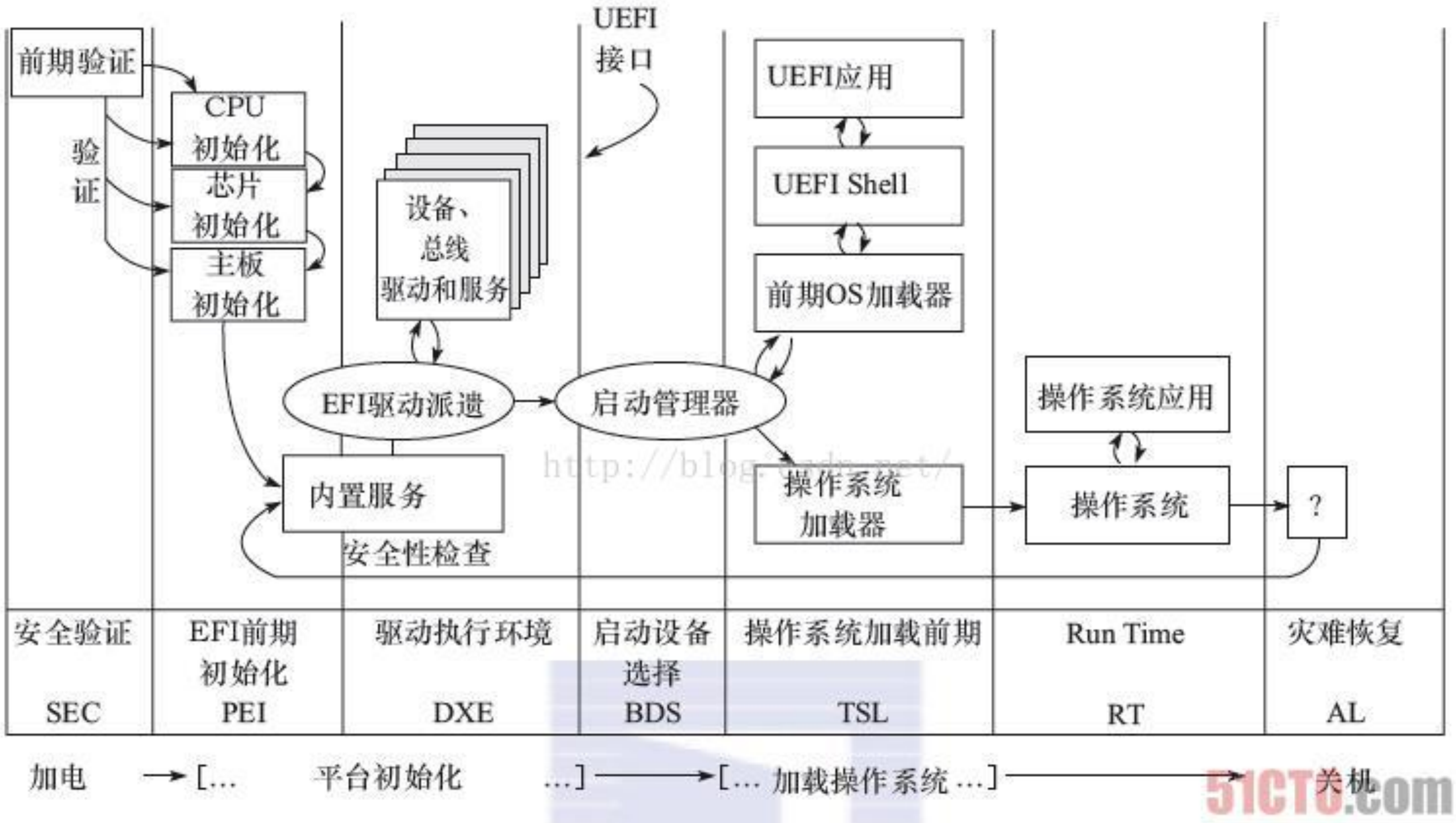
- UEFI
  - Unified Extensible Firmware Interface，统一可扩展固件接口
  - PC系统规范(specification)，定义操作系统与系统固件之间的软件界面
  - 负责加电自检（POST）、联系OS，提供连接OS与硬件的接口
- UEFI组成
  - 1. Pre-EFI初始化模块

    系统开机时，最先执行，负责CPU、芯片组及存储器的初始化，然后载入EFI驱动程序执行环境（DXE）
  - 2. EFI驱动程序执行环境DXE

    枚举并加载、初始化各个EFI驱动程序，e.g. PCI适配器的EFI驱动程序
  - 3. EFI驱动程序
    - 负责初始化硬件和设备
    - EFI驱动程序可以放置于系统的任何位置，如某个磁盘的EFI系统分区（ESP）中
    - EFI系统分区可以被UEFI固件访问，可用于存放操作系统的引导程序、EFI应用程序和EFI驱动程序，UEFI固件通过运行EFI系统分区中的启动程序启动操作系统
  - 4. 兼容性支持模块（CSM）
    - x86平台UEFI系统中，为不具备UEFI引导能力的操作系统（如Windows XP）提供类似于传统BIOS的系统服务
  - 5. EFI高层应用
  - 6. GUID磁盘分区表

UEFI启动过程

- https://blog.csdn.net/gjq_1988/article/details/50593564
- 遵循UEFI平台初始化（Platform Initialization）标准，系统从加电到关机分为7个阶段：
  - SEC（安全验证）
  - →PEI（EFI前期初始化）
  - →DXE（驱动执行环境）
  - →BDS（启动设备选择）
  - →TSL（操作系统加载前期）
  - →RT（Run Time）
  - →AL（系统灾难恢复期）



安全验证	EFI前期初始化	驱动执行环境	启动设备选择	操作系统加载前期	Run Time	灾难恢复
SEC	PEI	DXE	BDS	TSL	RT	AL

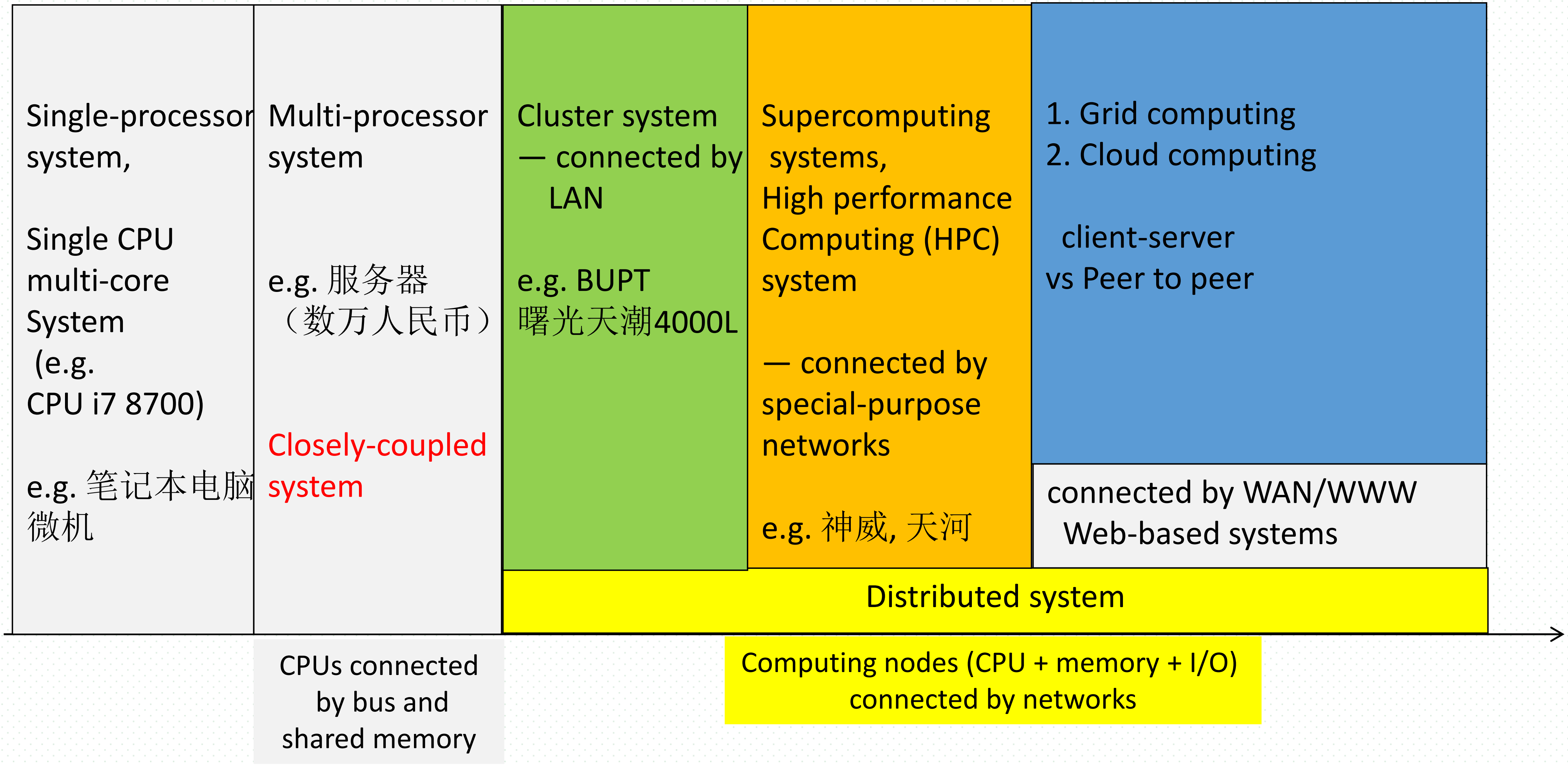
图 1-2　UEFI 系统的 7 个阶段

System Boot: UEFI

UEFI系统的启动过程
https://blog.csdn.net/gjq_1988/article/details/50593564

UEFI系统的启动?

Evolution of computer system structures



Single-processor system,

Single CPU multi-core System (e.g. CPU i7 8700)

e.g. 笔记本电脑 微机

Multi-processor system

e.g. 服务器（数万人民币）

Closely-coupled system

CPUs connected by bus and shared memory

Cluster system — connected by LAN

e.g. BUPT 曙光天潮4000L

Supercomputing systems,
High performance Computing (HPC) system

— connected by special-purpose networks

e.g. 神威, 天河

1. Grid computing
2. Cloud computing

client-server vs Peer to peer

connected by WAN/WWW
Web-based systems

Distributed system

Computing nodes (CPU + memory + I/O) connected by networks

Appendix 2.A 操作系统的兴衰

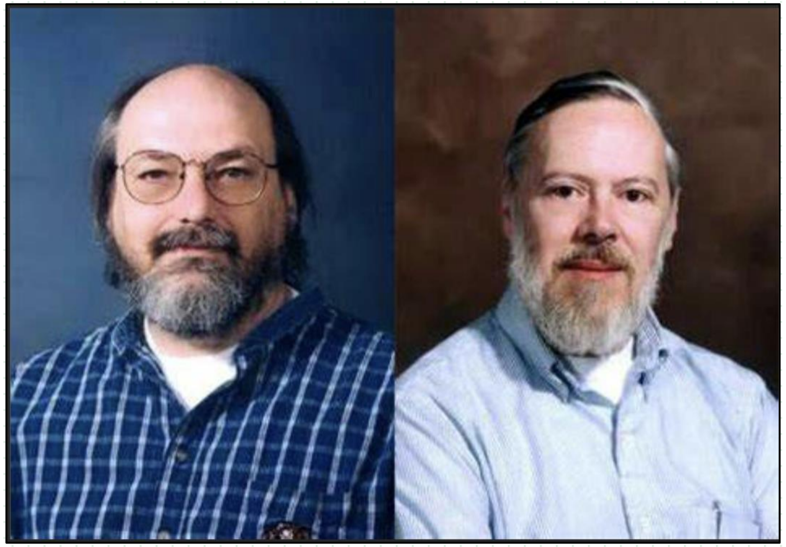
- 万赟，中国计算机学会通讯-2019年第8期
- 内容
  - IBM 的OS/360，第一个真正意义上的大型通用操作系统
  - UNIX 与开源运动
  - 微软视窗Windows
  - 开源运动与Linux
  - 苹果iOS与安卓
  - 鸿蒙的未来

Appendix 2.B DOS

- DOS, Acronym for Disk Operating System
- Command Line (CLI) user-machine interaction
- Single-user OS for personal computers
- Developed by IBM and Microsoft coordinately in 1980s, first deployed in 1981
  - PC-DOS by IBM, MS-DOS by Microsoft
- Functionalities
  - initially, command processing, file management, device management, very simple CPU scheduling and memory management
  - in the versions following up DOS 4.0, improving CPU and memory management

Appendix 2.C Introduction to Unix

- Unix is the *general-purpose, multi-user, interactive and time-sharing* OS
  - first distributed in 1969, by *AT&T Bell Lab*, implemented in PDP-11 assembly language
  - Unix 3.* was re-programmed in C
- Running on PC, servers, workstations and **mainframes**
- C language and its compiler was designed for re-implementing Unix efficiently during 1969-1973
  - Ken Thompson, Dennis M. Ritchie
    - Unix设计开发者
    - Turing Award Winners
  - Ken发明B语言， Dennis在此基础上发明C语言（1973年）
  - 二人使用C语言，重新设计改写用汇编语言编写的UNIX，得到Unix VersionV



Introduction to Unix (Cont.)

- In late 1970s, source codes of Unix implemented in C were distributed over industry and academic areas, *free of charge*, helping to improvement and popularization of Unix in practice
  - evil consequences: kernel forking(内核分叉)
    - 魔改Unix内核，造成各个Unix发行版本不兼容
- As free and open-source OS, Unix has many versions developed by computer companies, universities and research institutes
  - AT&T Unix: System III, System V, SVR 3.2
  - University of California State at Berkeley: Unix BSD (Berkeley Software Distribution)
  - IBM: AIX
  - HP: HP-UX;
  - Sun: Solaris, which is based on BSD
  - SGI: Irix;
  - Microsoft: XENIX
- **Gradually replaced by Linux** due to kernel forking

Introduction to Unix (Cont.)

- Characteristics of Unix
  - 多用户多任务，C语言编写，易读、可移植性好
  - 体系结构上分内核部分和应用子系统，可扩展性好，易于做成开放系统
  - 分层可装卸卷的文件系统，提供文件保护功能
  - 提供I/O缓冲技术，系统效率高
  - 抢占式动态优先级CPU调度，支持分时功能
  - 请求分页式虚拟内存管理
  - 以shell作为用户界面，命令语言丰富
  - 网络与通信功能强大

- Demerits
  - more than one hundred versions in 1990s, many of which are not compatible to each other

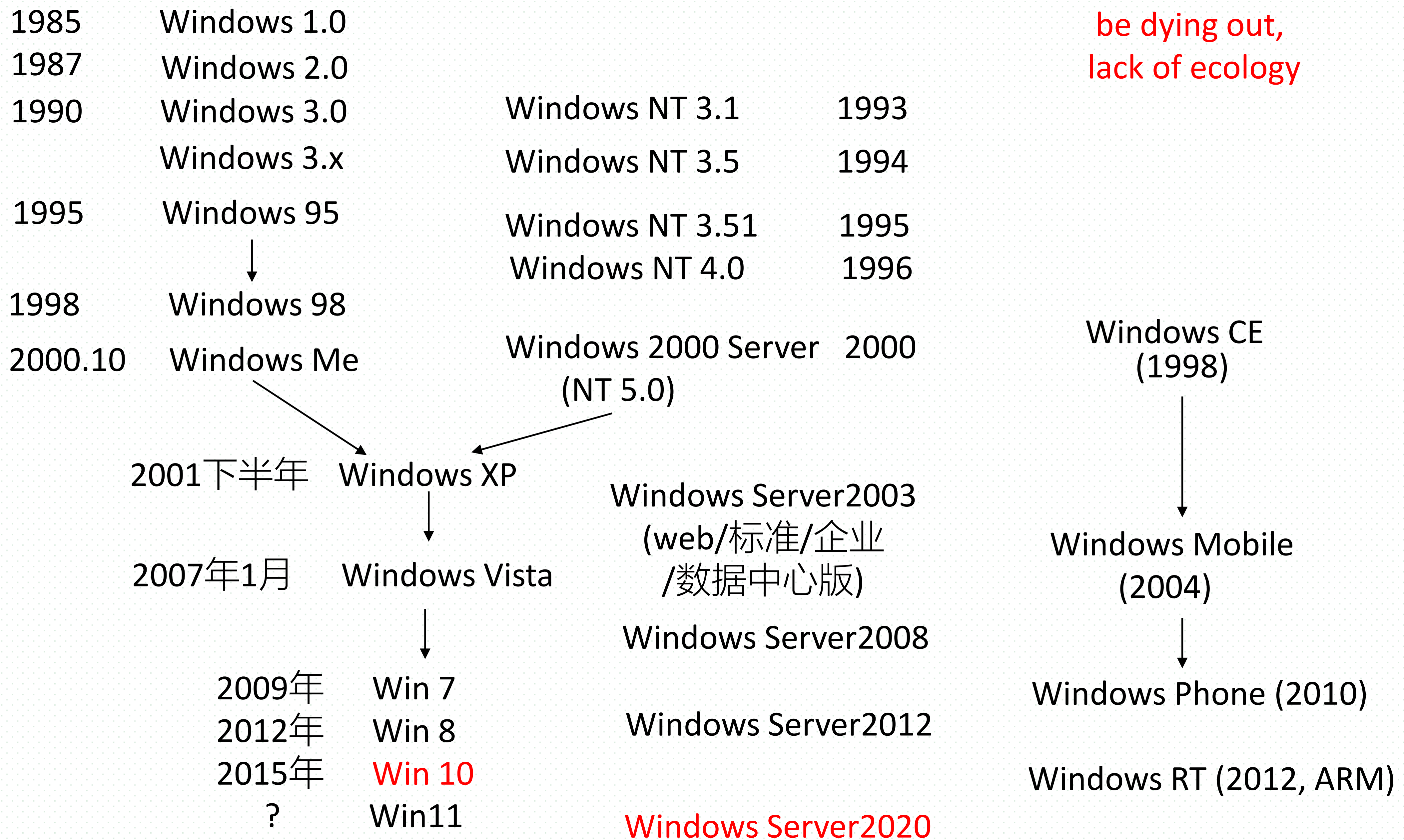
- Unix has been the standards for operating systems
  - ***POSIX***, Unix standards defined by IEEE, which defines the minimal set of system call interfaces and tools that all the compatible(兼容) *Unix-like OS* must support

Appendix 2.D Introduction to MS-Windows

- Three categories of Windows OS products
  - 个人OS，服务器OS，嵌入式OS
    - 没有面向大型主机的产品
  - 运行平台

    个人计算机，工作站/服务器, 嵌入式/手持设备
  - 参见下图

单用户OS 工作站/服务器OS 嵌入式OS



单用户OS

1985	Windows 1.0
1987	Windows 2.0
1990	Windows 3.0
	Windows 3.x
1995	Windows 95
1998	Windows 98
2000.10	Windows Me
2001下半年	Windows XP
2007年1月	Windows Vista
2009年	Win 7
2012年	Win 8
2015年	Win 10
?	Win11

工作站/服务器OS

Windows NT 3.1	1993
Windows NT 3.5	1994
Windows NT 3.51	1995
Windows NT 4.0	1996
Windows 2000 Server (NT 5.0)	2000

Windows Server2003 (web/标准/企业/数据中心版)

Windows Server2008

Windows Server2012

Windows Server2020

嵌入式OS

be dying out, lack of ecology

Windows CE (1998)

Windows Mobile (2004)

Windows Phone (2010)

Windows RT (2012, ARM)

Windows NT/Server

- NT 1993年推出，多用户OS
- Layered + microkernel，object-oriented (面向对象技术设计)
  - 内核态部分(NT 执行体) + 用户态部分
  - 参见下图
- NT 执行体：
  - 实现OS主要功能
  - 包含内核部分
  - OS其他功能则与应用程序一样作为服务器进程，以C/S模式在用户态进行

- （微）内核
  - 内核线程调度、进程转换、异常和中断管理、多处理器同步，实现内核对象
  - 利用线程提高处理并行性
- 内核与执行体内其他部分的区别
  - 内核提供最基本功能，执行体其他部分调用内核所提供的功能完成进一步的工作
  - 内核常驻内存，不会被页面调度程序调出内存
  - 内核不可再被拆分
- 其它特点:
  - 支持SMPP (双CPU 或4CPU) 、 GUI 界面、网络与多媒体功能

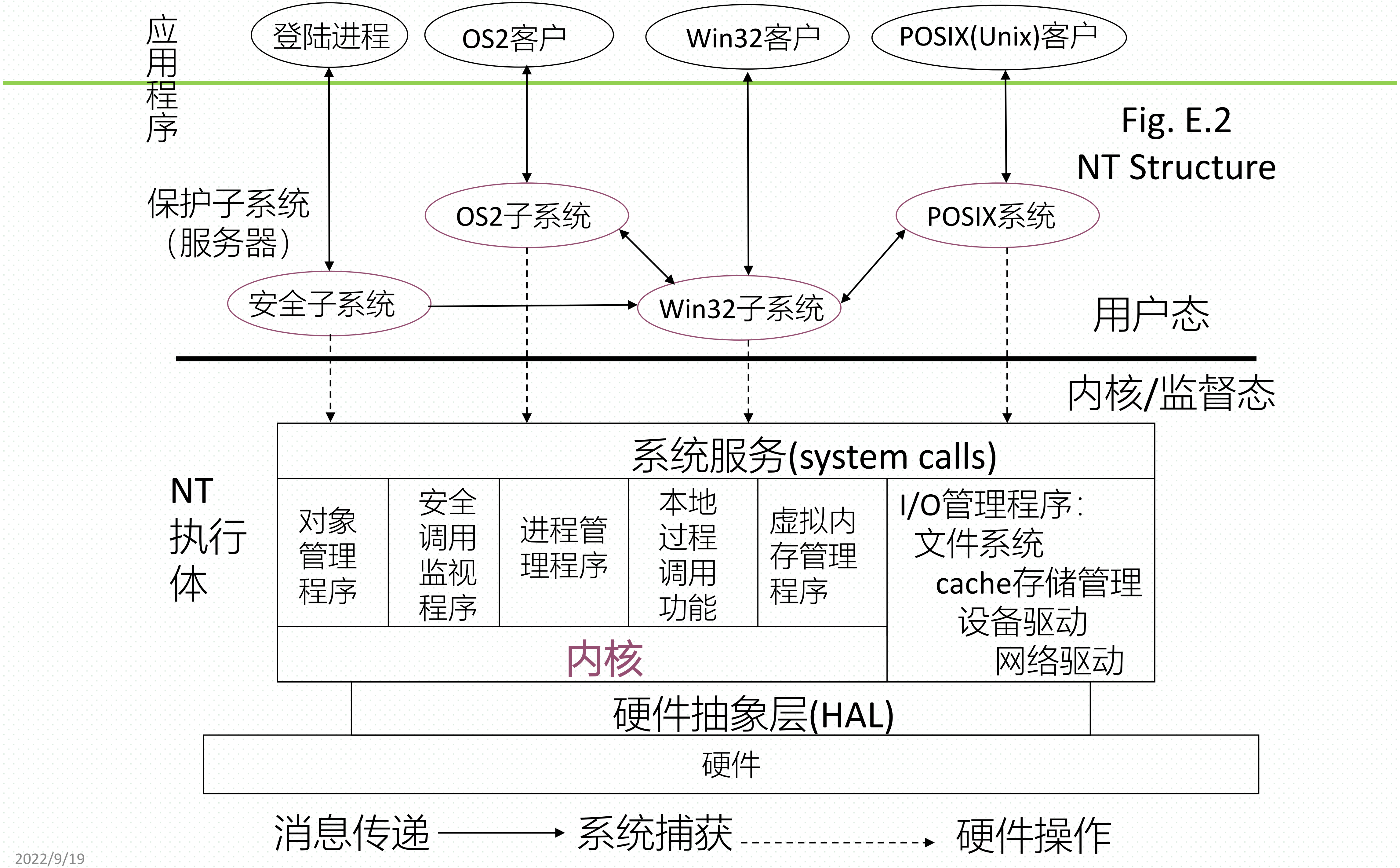


Fig. E.2

NT Structure

应用程序

登陆进程

OS2客户

Win32客户

POSIX(Unix)客户

保护子系统（服务器）

OS2子系统

POSIX系统

安全子系统

Win32子系统

用户态

内核/监督态

系统服务(system calls)

NT 执行体

对象管理程序

安全调用监视程序

进程管理程序

本地过程调用功能

虚拟内存管理程序

I/O管理程序：

文件系统

cache存储管理

设备驱动

网络驱动

内核

硬件抽象层(HAL)

硬件

消息传递 → 系统捕获 ---> 硬件操作

Appendix Virtual Machines

- 第7版2.8节，第9版1.11节
- Concepts and Principles（*掌握*）
- 逻辑虚拟模式
- 硬件虚拟模式
- 软件虚拟模式
- 应用虚拟模式

Concepts and Principles

- Virtual machine **(VM)** /* 资源独占性，简化编程
  - software that mimics the performance of a hardware devices, to allow each application process to run on its own running space (its own hardware resources), and each to be provided with a virtual copy of the underlying computer
  - Fig. 1.20 ▶
- Another definition /*程序可移植性
  - software that mimics the performance of hardware devices, such as a program that allows application programs written for an Intel processor to be run on a Motorola chip
  - e.g. SkyEye
- VM实质
  - 将用户编程可使用的指令集映射到计算机的实际指令集

How to create VM

- The resources of the physical computer are shared to create the virtual machines——多重化技术
  - *CPU scheduling* can create the appearance that users have their own processor.
  - *spooling* and *a file system* can provide virtual card readers and virtual line printers.
  - a normal user time-sharing terminal serves as the virtual machine operator’s console

Features of Virtual Machines

- Advantages of Virtual Machines
  - VM provides complete protection of system resources, since each VM is isolated from all otherVMs. This isolation, however, permits no direct sharing of resources.

    — 用户程序资源独占性，简化编程
  - a VM is a perfect vehicle for operating-systems research and development. System development is done on VM, instead of on a physical machine and so does not disrupt normal system operation, for example **Java VM**

    — 程序与硬件间的物理独立性，可移植性，兼容性

- Disadvantage of virtual machine
  - VM concept is difficult to implement (?)
    - due to the effort required to provide an *exact* duplicate to the underlying machine

逻辑虚拟模式

- 最早源自于IBM大型主机的逻辑分区技术
  - Fig. 2.0.7 ▶
- Principles
  - Fig. 1.14
  - 利用虚拟机软件，将计算机物理资源通过多重化(即虚拟出多个CPU、memory、I/O设备等)、共享技术等软件手段改造成多个硬件副本—虚拟机，提高硬件资源利用率
  - 可实现资源共享和系统保护

    每个虚拟机的用户可以独立、不受其它用户影响地（e.g. 越界访问）在虚拟机上运行程序、存储数据，当虚拟机崩溃时也不会影响到整个系统和其它虚拟机及其用户

逻辑虚拟模式

越界访问，资源竞争

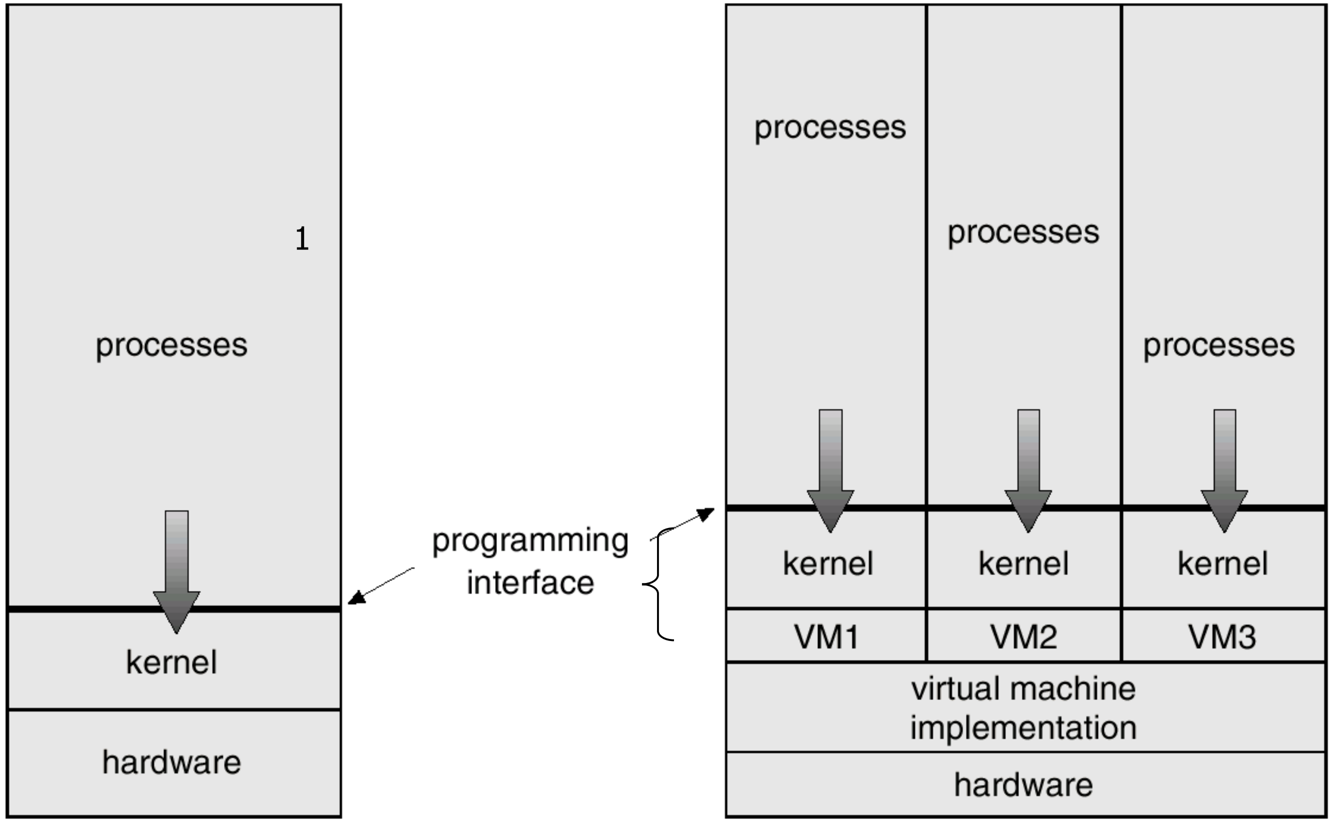


Fig. 1.20 System Models

逻辑虚拟模式

- VM was first introduced in IBM CP/CMS (later renamed VM/370) on mainframe systems in 1979, and widely used in IBM S/390 etc. today

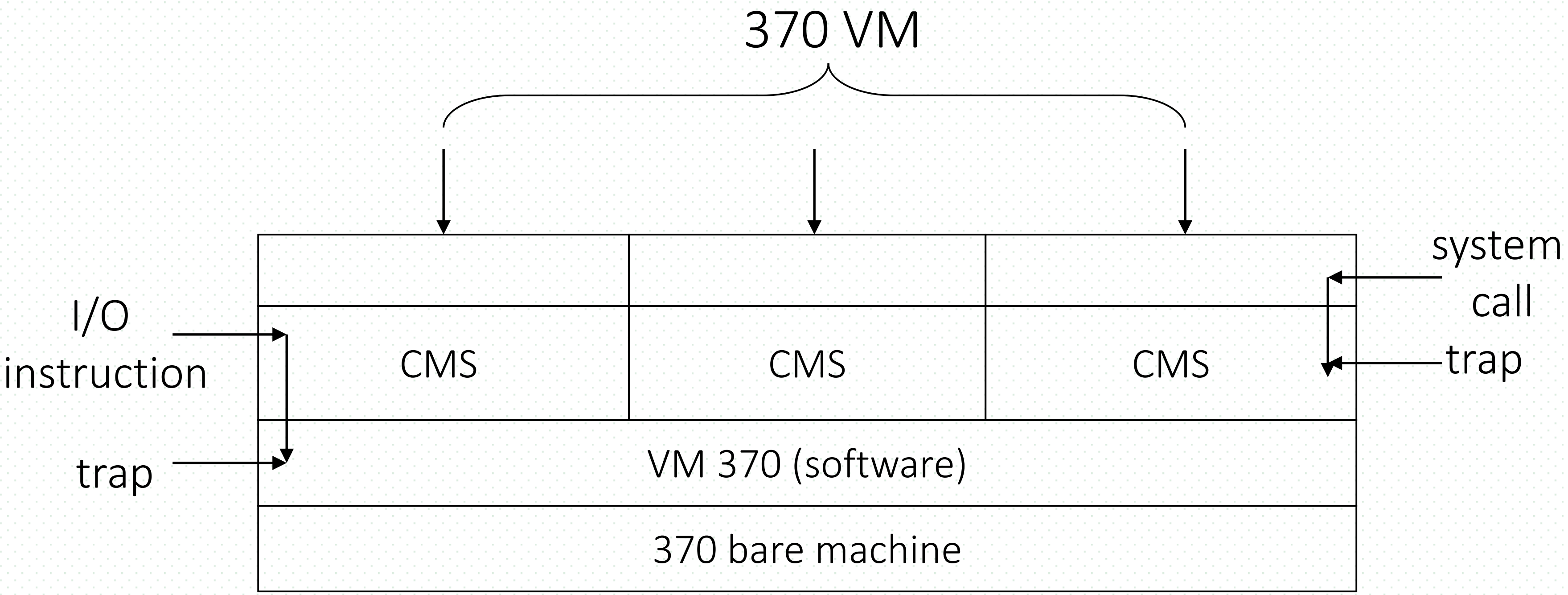


Fig. 2.07 IBM 370 virtual machine

逻辑虚拟模式

- 方法：

  在硬件和操作系统之间引入**虚拟控制层软件**

  - 控制层软件管理实际的物理硬件，并为用户模拟出不同的虚拟机（硬件）
- 代表产品
  - IBM VM370
  - HP vPAR、
  - Vmware ESX Server
  - Xen
- E.g. Fig. 2.0.9

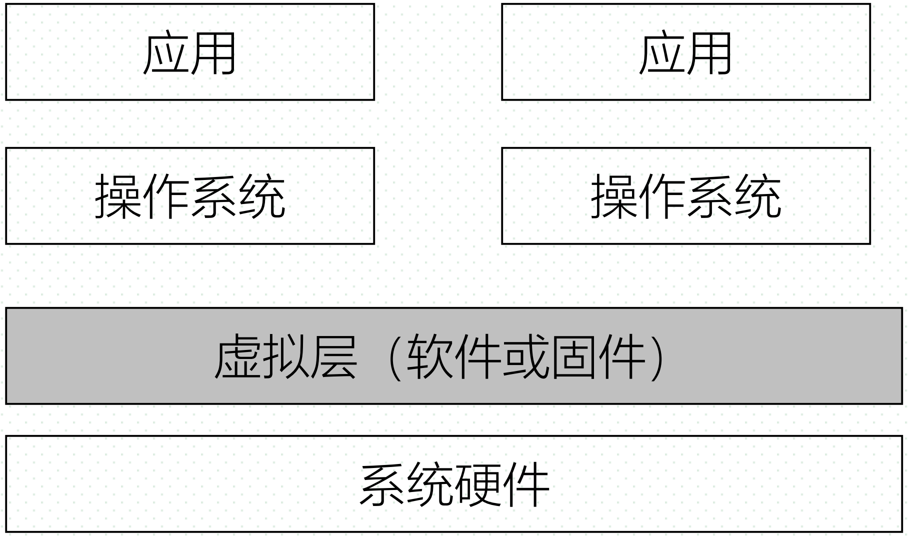
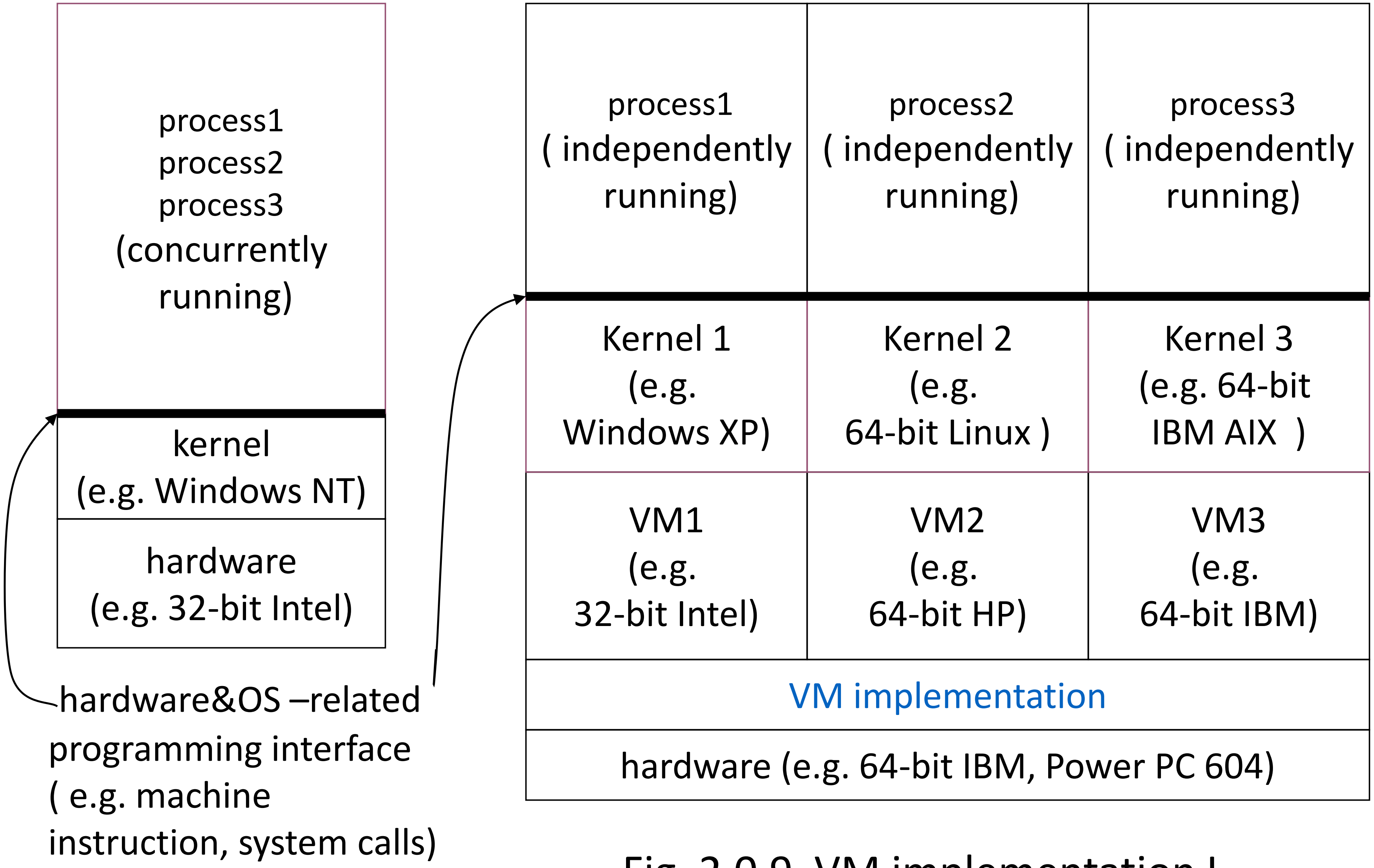


Fig. 2.0.8 逻辑虚拟实现技术

Non-virtual Machine

Virtual Machine: 逻辑虚拟模式



process1
process2
process3
(concurrently running)

kernel
(e.g. Windows NT)

hardware
(e.g. 32-bit Intel)

hardware&OS –related programming interface (e.g. machine instruction, system calls)

process1 (independently running)	process2 (independently running)	process3 (independently running)
Kernel 1 (e.g. Windows XP)	Kernel 2 (e.g. 64-bit Linux)	Kernel 3 (e.g. 64-bit IBM AIX)
VM1 (e.g. 32-bit Intel)	VM2 (e.g. 64-bit HP)	VM3 (e.g. 64-bit IBM)

VM implementation

hardware (e.g. 64-bit IBM, Power PC 604)

Fig. 2.0.9 VM implementation I

逻辑虚拟模式：Intel, AMD CPU：HT + multi-core

- HT (超线程技术) in Intel/AMD CPU
  - in desktop systems, CPU芯片内部以硬件方式实现逻辑虚拟 (implemented by CPU)
  - refer to Chapter 4

硬件虚拟模式

- HP和SUN通过在Unix服务器上采用硬分区（或称物理分区），将系统硬件分为多个单元，每个单元可独立运行不同的操作系统
  - Fig. 2.0.10
  - 系统采用MBB(Modular Building Block)架构，由多个BB构成(Building Block)
  - 每个BB可包含4路CPU、若干内存和I/O卡；各BB间通过Crossbar Switch交换机制连接在一起
  - 每个BB可拥有独立的CPU频率，可运行各自不同的操作系统，构成一个独自为用户服务的虚拟机服务器
  - 任一分区内的操作系统和硬件故障不影响其他分区

- E.g. Fig. 2.0.11
- 代表产品
  - HP nPAR

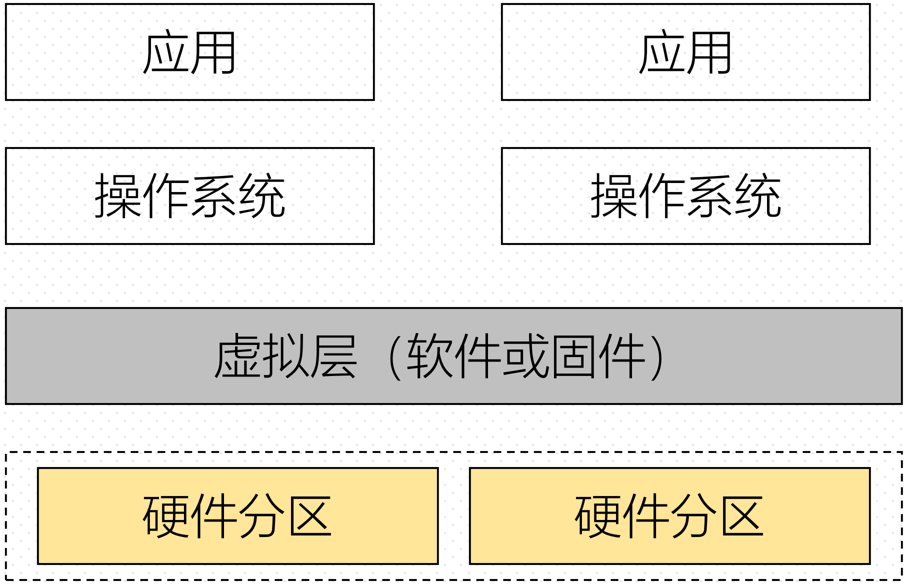


Fig. 2.0.10 硬件虚拟实现技术

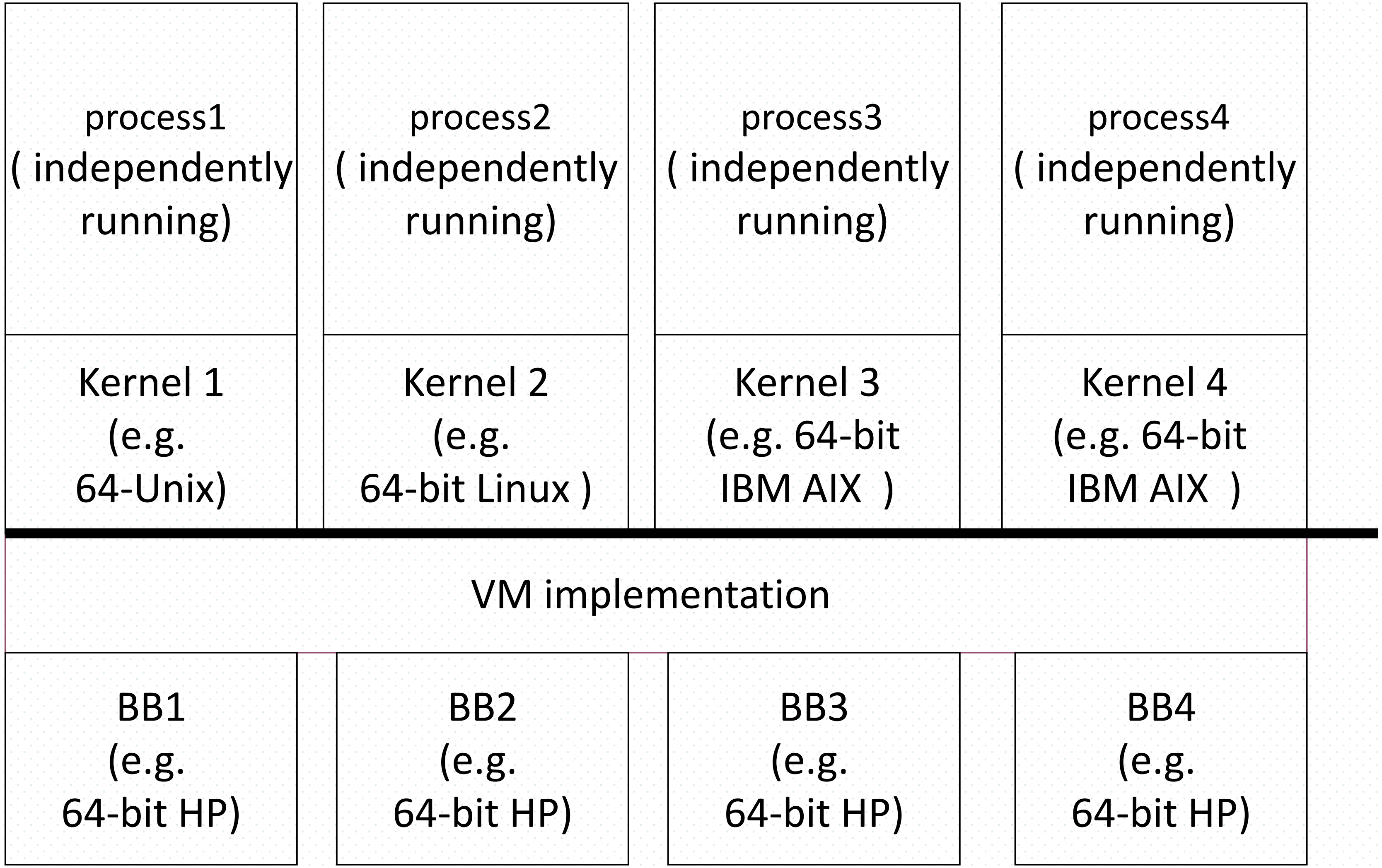

process1 (independently running)
process2 (independently running)
process3 (independently running)
process4 (independently running)
Kernel 1 (e.g. 64-Unix)
Kernel 2 (e.g. 64-bit Linux)
Kernel 3 (e.g. 64-bit IBM AIX)
Kernel 4 (e.g. 64-bit IBM AIX)
VM implementation
BB1 (e.g. 64-bit HP)
BB2 (e.g. 64-bit HP)
BB3 (e.g. 64-bit HP)
BB4 (e.g. 64-bit HP)

Fig. 2.0.11 VM implementation II

硬件虚拟模式 (cont.)

- 基于**多核微处理器**技术，Intel公司和AMD公司将硬件虚拟模式引入到desktop 系统
  - Intel公司2005年11月推出的运行于Xeron MP处理器系统的**硬件辅助虚拟化技术VT**（Vanderpool Technology）
  - Intel公司2006年推出采用Intel Virtual Machine Monitor虚拟技术的处理器
  - AMD公司2006年推出采用AMD Pacific虚拟技术的处理器

Intel, AMD CPU： HT + multi-core

硬件虚拟模式 (cont.)

- 上海超级计算中心主机系统（2015年），数万(6万?)个计算节点，
- 每个节点
  - 双路12核 Intel Xeon E5-2680 v3处理器
  - 主频2.5GHz
  - 单节点128GB内存
  - 节点间通信使用EDR InfiniBand网络 (100Gb/s)

- 可以为用户独立分配一定数量的计算节点

软件虚拟模式

- 原理
  - Fig. 2.0.12
  - 虚拟层软件运行于宿主（host）操作系统之上，与硬件层无关
  - 虚拟层软件支持多个客户（guest）操作系统，每个客户操作系统为不同用户提供了不同的运行环境，称之为虚拟机
  - 一台物理服务器上可运行多个虚拟机
  - 各虚拟机为用户应用提供独立运行环境

软件虚拟模式 (cont.)

- 软件虚拟模式 (cont.)代表产品
  - Virtual Box
  - KVM
  - Vmware GSX Server, VMware For Windows
  - Virtual PC for Windows 5_2 汉化版

- E.g. Fig. 2.0.13
- VMware architecture ,

  Fig.2.16

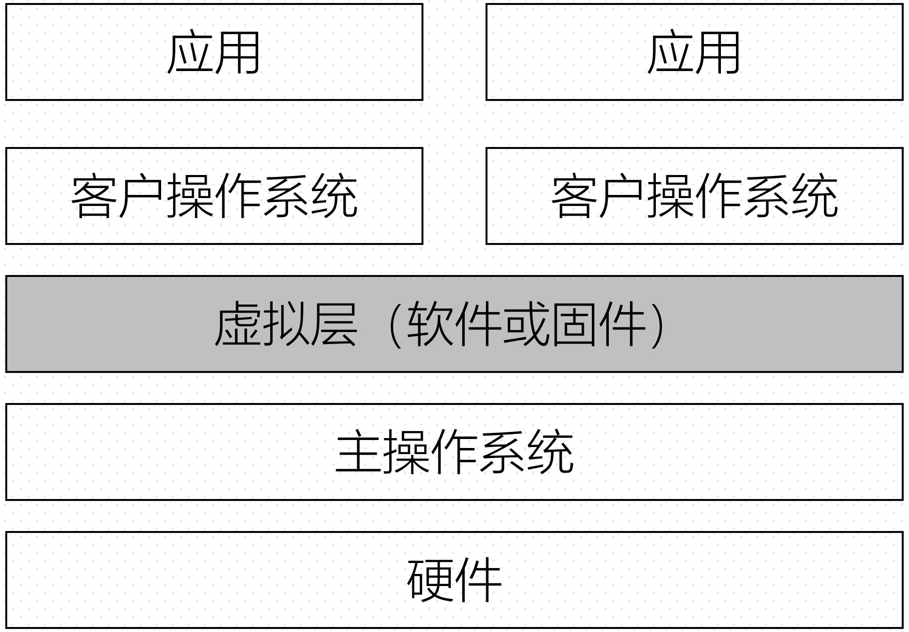


Fig. 2.0.12 软件虚拟实现技术

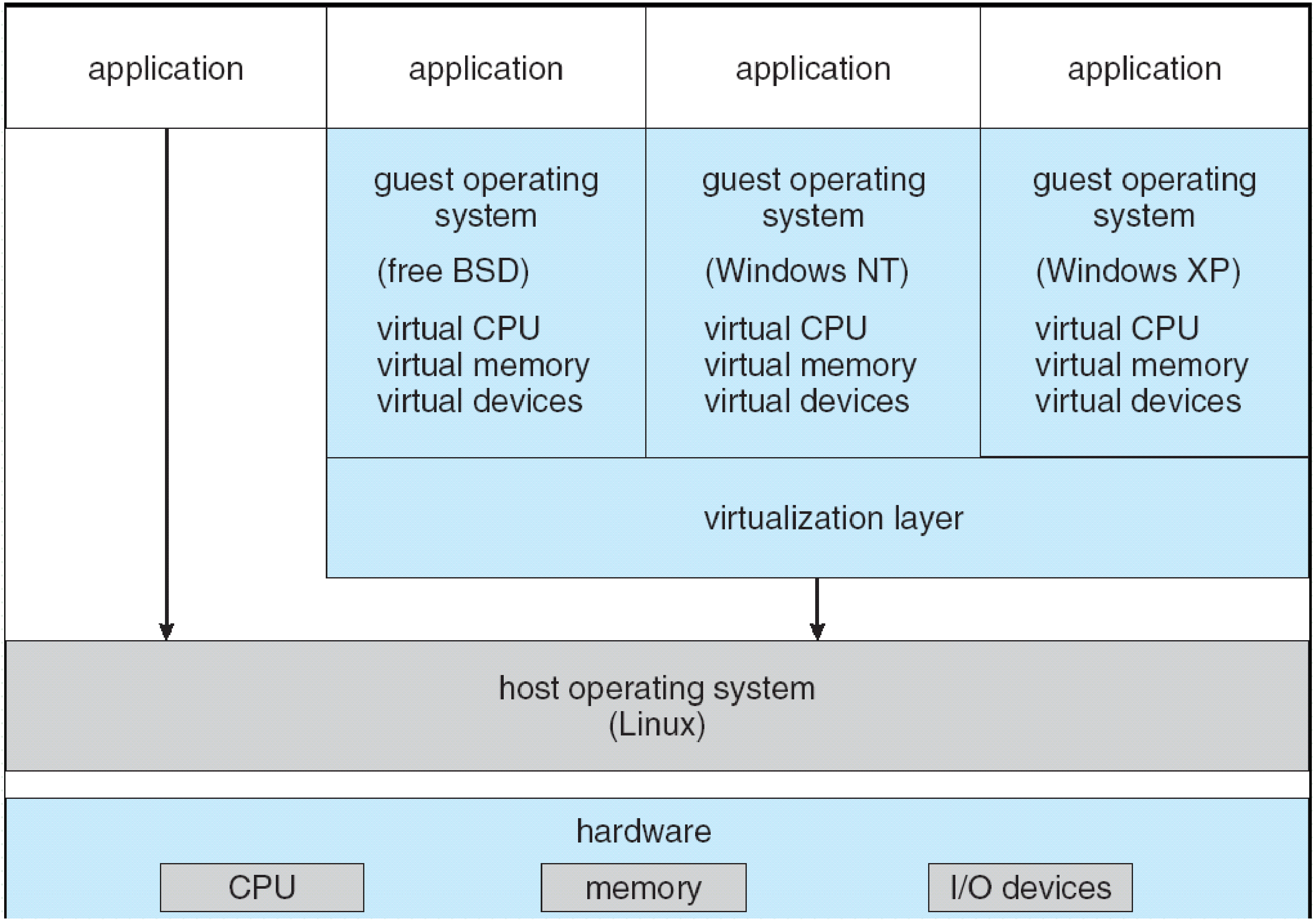
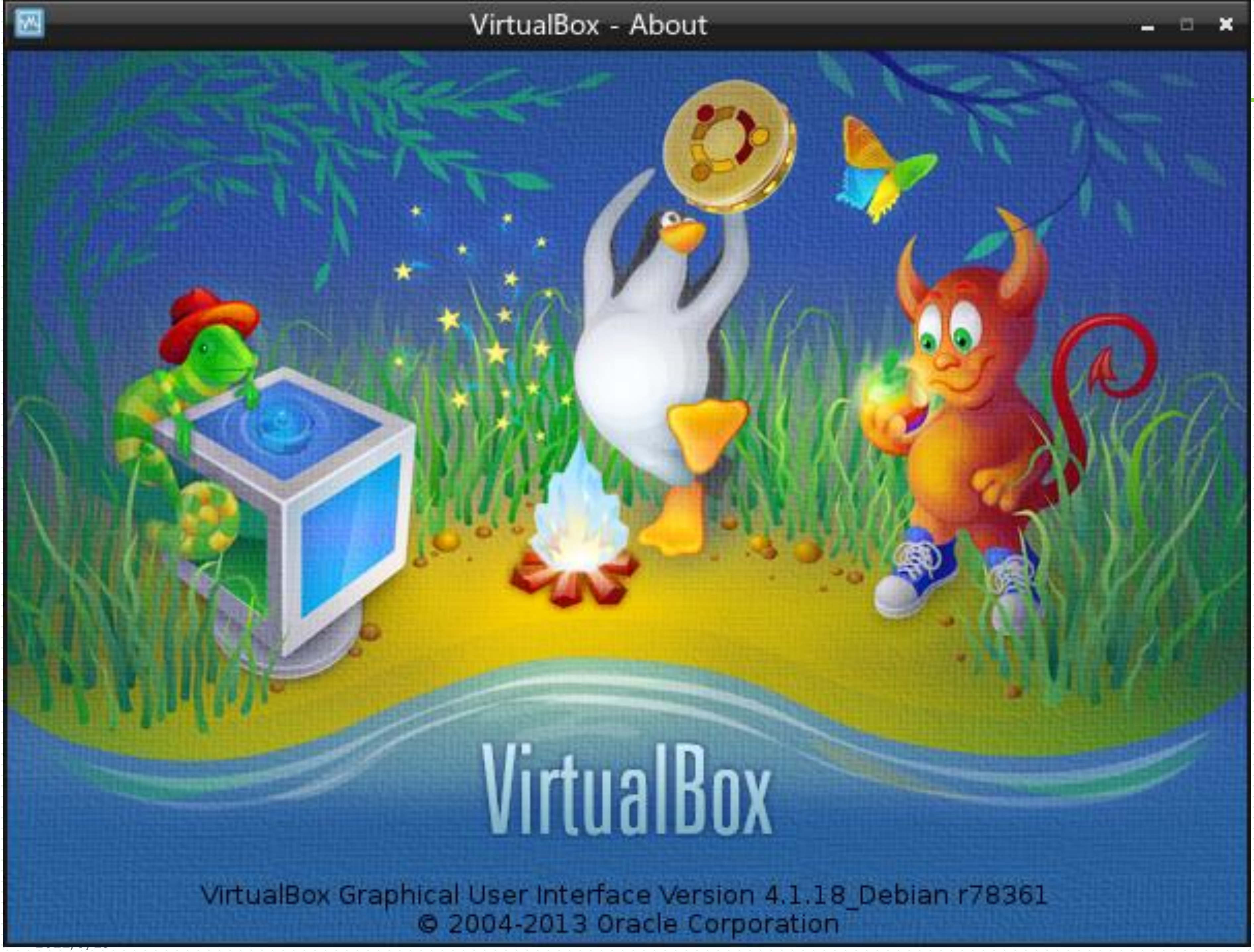


Fig. 2.16 VMware Architecture



VirtualBox Graphical User Interface Version 4.1.18_Debian r78361
© 2004-2013 Oracle Corporation

- 蜥蜴趴着的盒子: Virtualbox的logo；

Linux类：

- 企鹅: Linux；
- 绿色蜥蜴: SUSE Linux的吉祥物；
- 蜥蜴带着的红色帽子: RedHat Linux；
- 企鹅手里的鼓纹: Ubuntu Linux；
- 小红魔的尾巴: Debian Linux的标志；
- 小星星: 代表Mandriva Linux；

- 鼓旁边的四色蝴蝶: Microsoft Windows；

Unix类：

- 最右的小红魔: FreeBSD的吉祥物；
- 小红魔手里的苹果: MacOS，源于FreeBSD
- 中间的一团火: Solaris的阳光火焰；

应用虚拟模式

- 原理
  - Fig. 2.0.14
  - 虚拟层软件运行于主操作系统之上，与硬件层无关
  - 虚拟层软件之上直接运行多个应用域软件/中间件，分别对应了多个独立的用户运行环境
- 代表产品
  - Sun公司的Solaris Container
  - JVM
  - 容器，Docker——分布式计算环境）

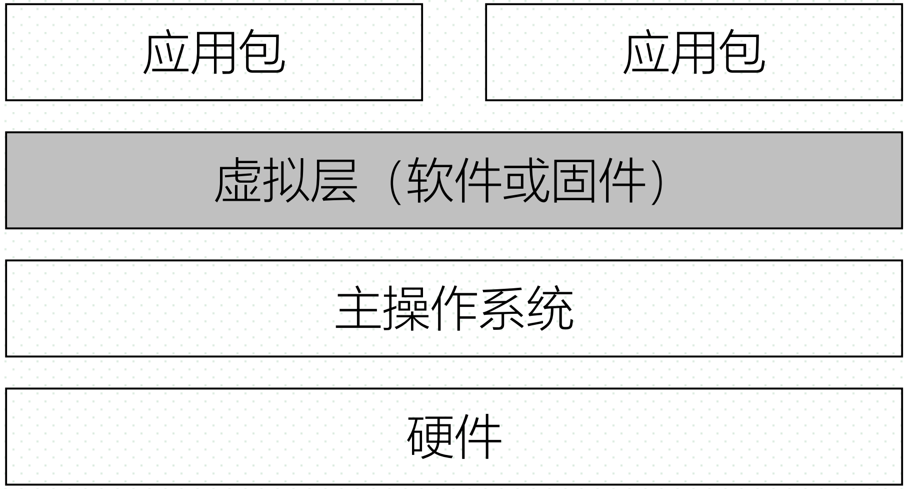


Fig. 2.0.14 应用虚拟实现技术

(与软件虚拟技术的区别)

Java Virtual Machine

- /*属于应用虚拟技术, 实现了应用程序跨平台的可移植性
  - Fig. 3.0.16

Java虚拟机简介

- JVM, a specification for an abstract computer, consists of

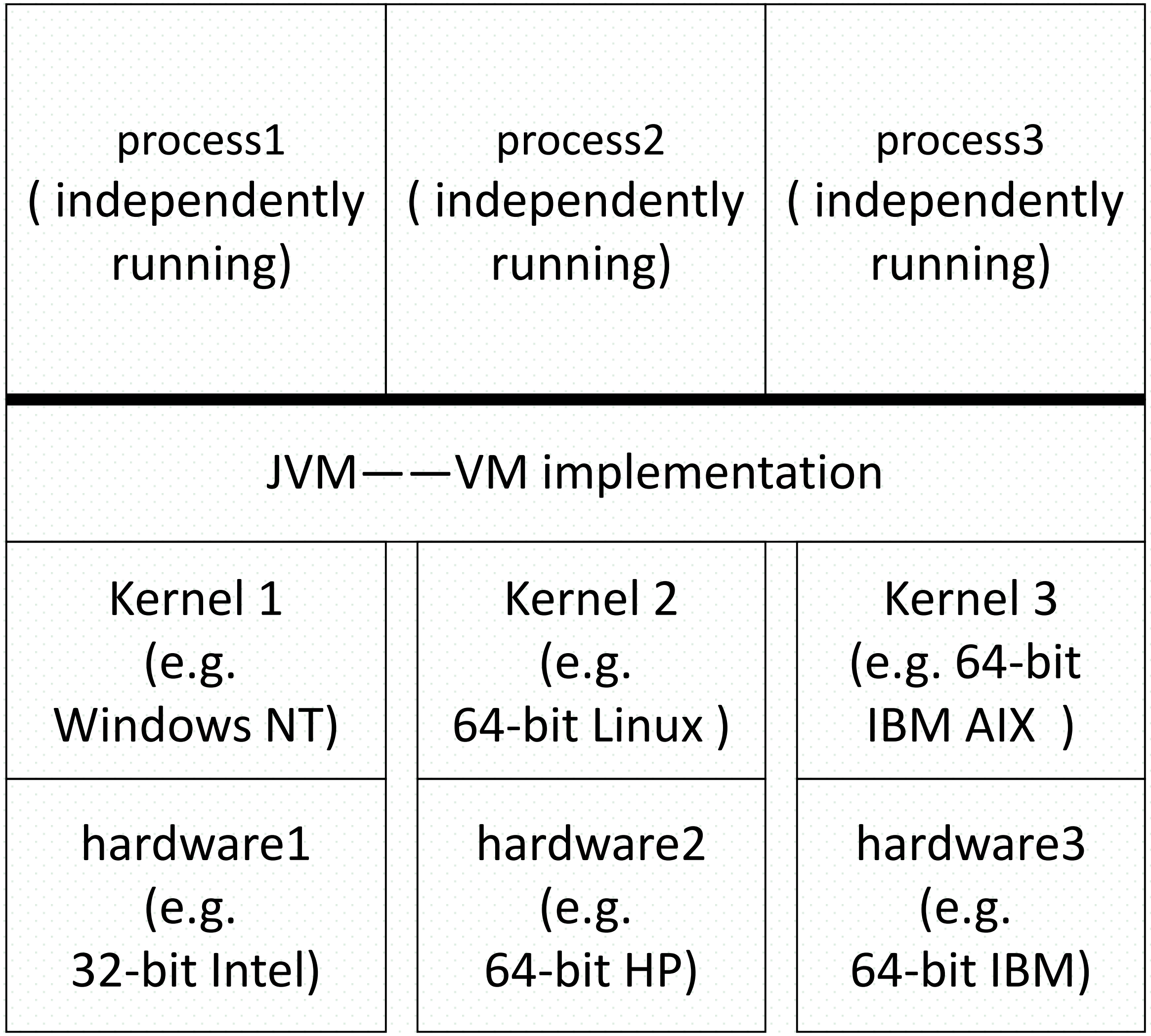
  - class loader

  - class verifier

  - runtime interpreter

- Compiled Java programs are platform-neutral bytecodes (字节码) executed by a Java Virtual Machine (JVM)
  - Java程序 (编译) →字节码 → 通过Java虚拟机JVM，解释执行在CPU上；
  - JVM：运行时 runtime，虚拟CPU，提供了字节码执行环境

Java Virtual Machine (cont.)



process1 (independently running)

process2 (independently running)

process3 (independently running)

JVM——VM implementation

Kernel 1 (e.g. Windows NT)

Kernel 2 (e.g. 64-bit Linux)

Kernel 3 (e.g. 64-bit IBM AIX)

hardware1 (e.g. 32-bit Intel)

hardware2 (e.g. 64-bit HP)

hardware3 (e.g. 64-bit IBM)

Fig. 2.0.15 JVM与应用虚拟实现技术

Java Virtual Machine (cont.)

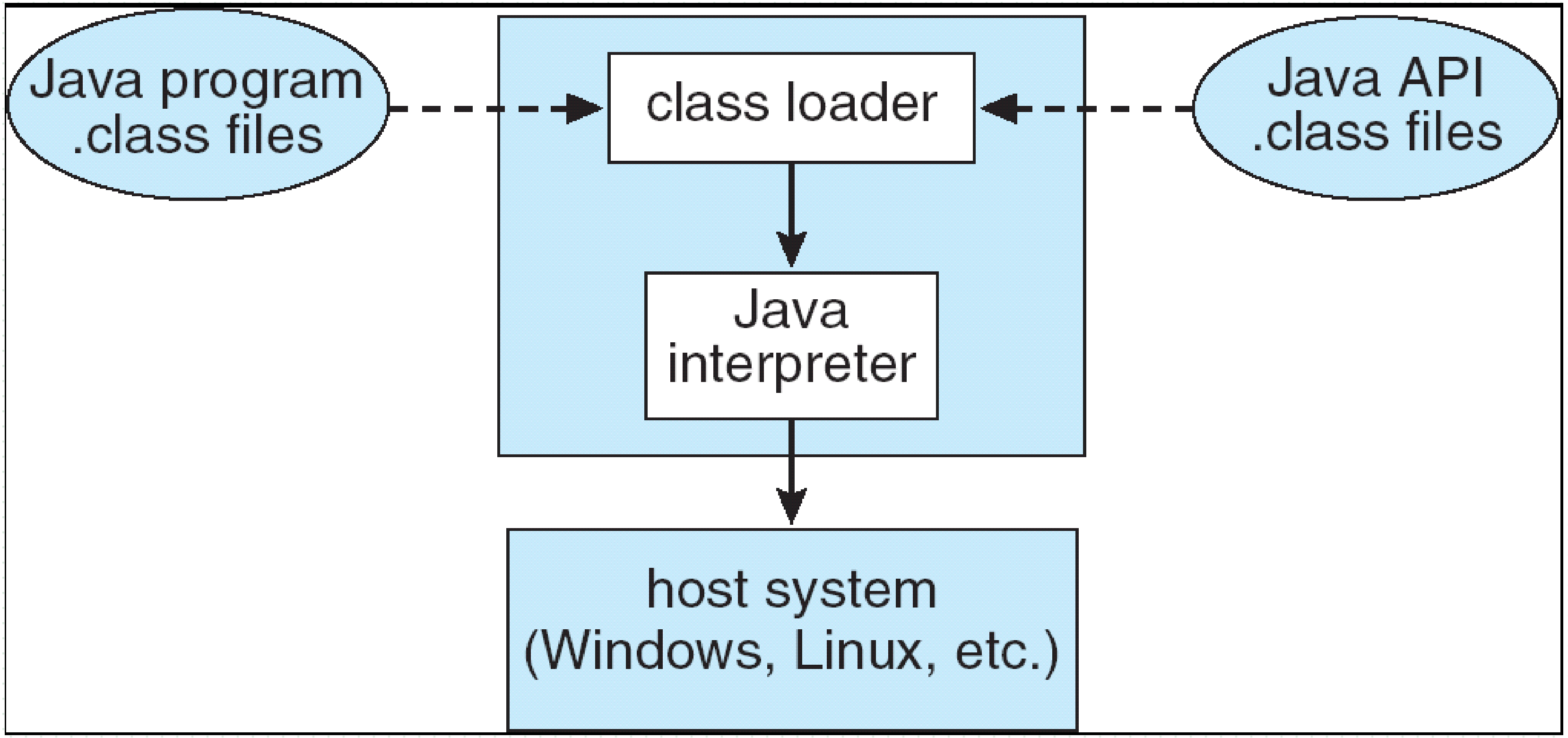


Fig.3.12 Java Virtual Machine

Java Virtual Machine (cont.)

- Java interpreter

  runs the verified java.class files by means of

  - interpreting the bytecode one at a time
  - just-in-time (JIT) compiler that turns the architecture-neutral bytecodes into native machine language for the host machine hardware that executes Java bytecodes natively.
- Just-In-Time (JIT) compilers increase performance
- JVM make it possible to develop architecture-neutral and portable programs

- For more details, refer to *Java虚拟机简介*

Java虚拟机简介

Thanks for your attention