Operating system

Part II: Problems and Ideas in OS

By **KONG** LingBo (**孔令波**)

Ser
vice
s

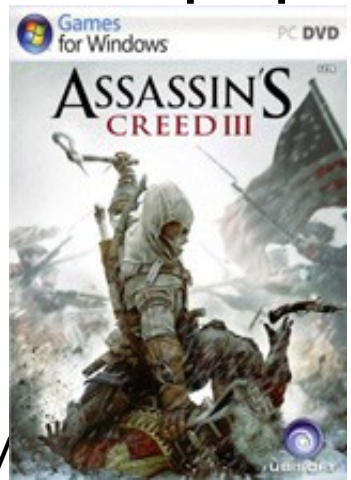
- Problems and ideas when implementing OS
  - Problems: EMM+GSD
    - Concurrency, Resource limitation, Security, GUI, …
  - Ideas: Multiplex resource + Synchronize programs
    - **Multiplexing** resources based on interrupt mechanism
    - **Synchronize** the access of the limited resource among the related programs
- Evolution of OSs' structures
  - Construct OS with so many functions
- System Call/API
  - How to use the functions defined in OS?
- How to load and run OS?

How to understand the problems in OS?

- Same as in other courses/theories, we could start from **SIMPLE** situation
  - 1:1:M is only for **one program** whose size is smaller than that of the available space in MM
    - "the available" here is because "OS also needs some MM space"
- EMM indicates the common problems: HOW
  - **EXECUTION** ☾ CPU
    - How to configure CPU (set instruction and parameter to registers) to run your program?
      - **You've understood this through CO course (I hope)**
  - **MAPPING 2** ☾ from File to Hard disk space
    - Your program is first represented as files in hard disk (usually). How to organize many files?
  - **MAPPING 1** ☾ from File to Main Memory space
    - The fact from von Neumann gives that your programs should copied into MM space

However, OS is more complicated

- **Complexity is caused by the res ource limitation**, just as in real life
  - EMM is still the hint to understand the complic ated situation
- Mapping 1 ☾ **“LARGE PROGRAM V.S SMA LL MM” problem**
  - 4GB now is popular, but

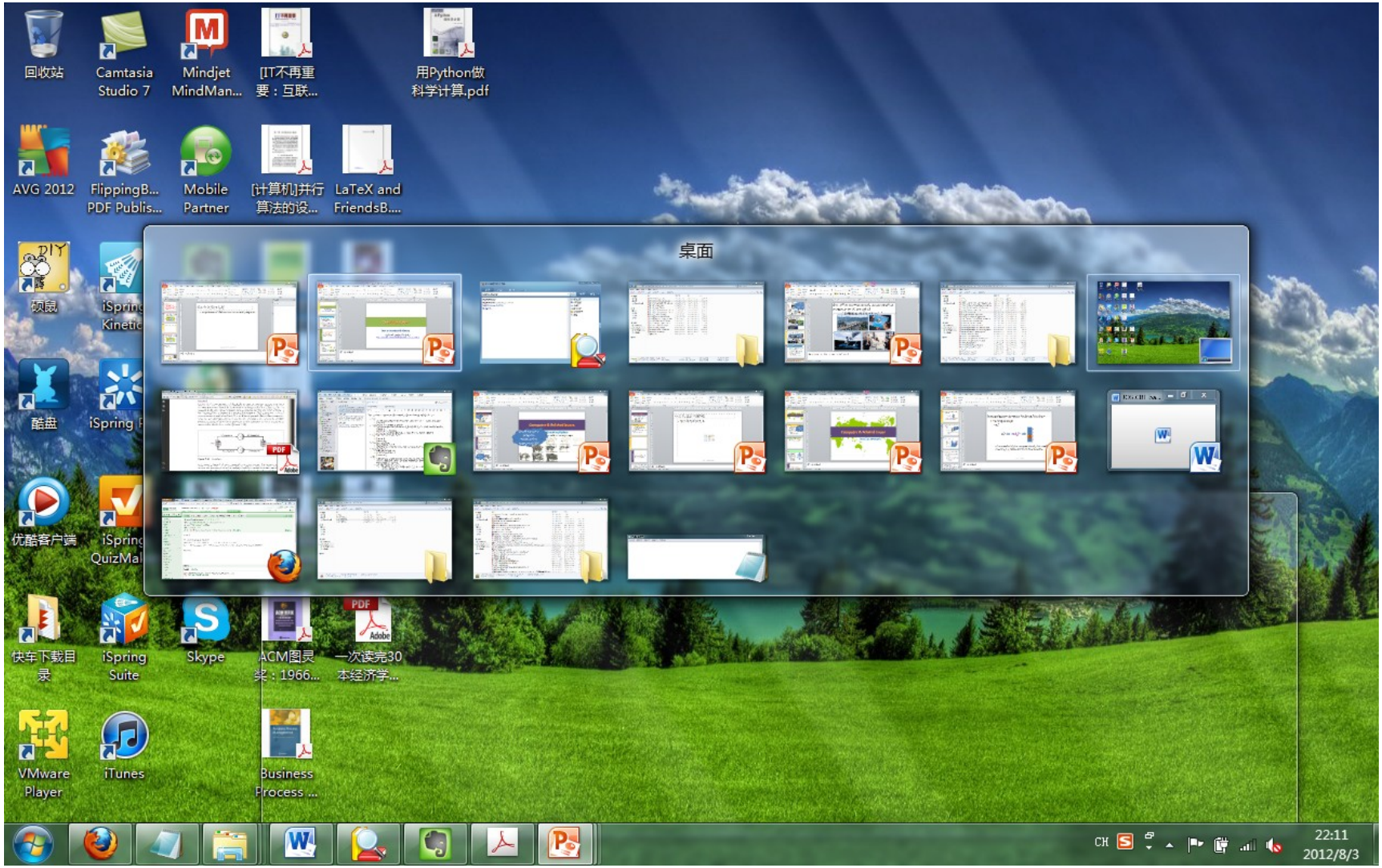


ttp://www.gamersky ng

Modern OS is more complicated

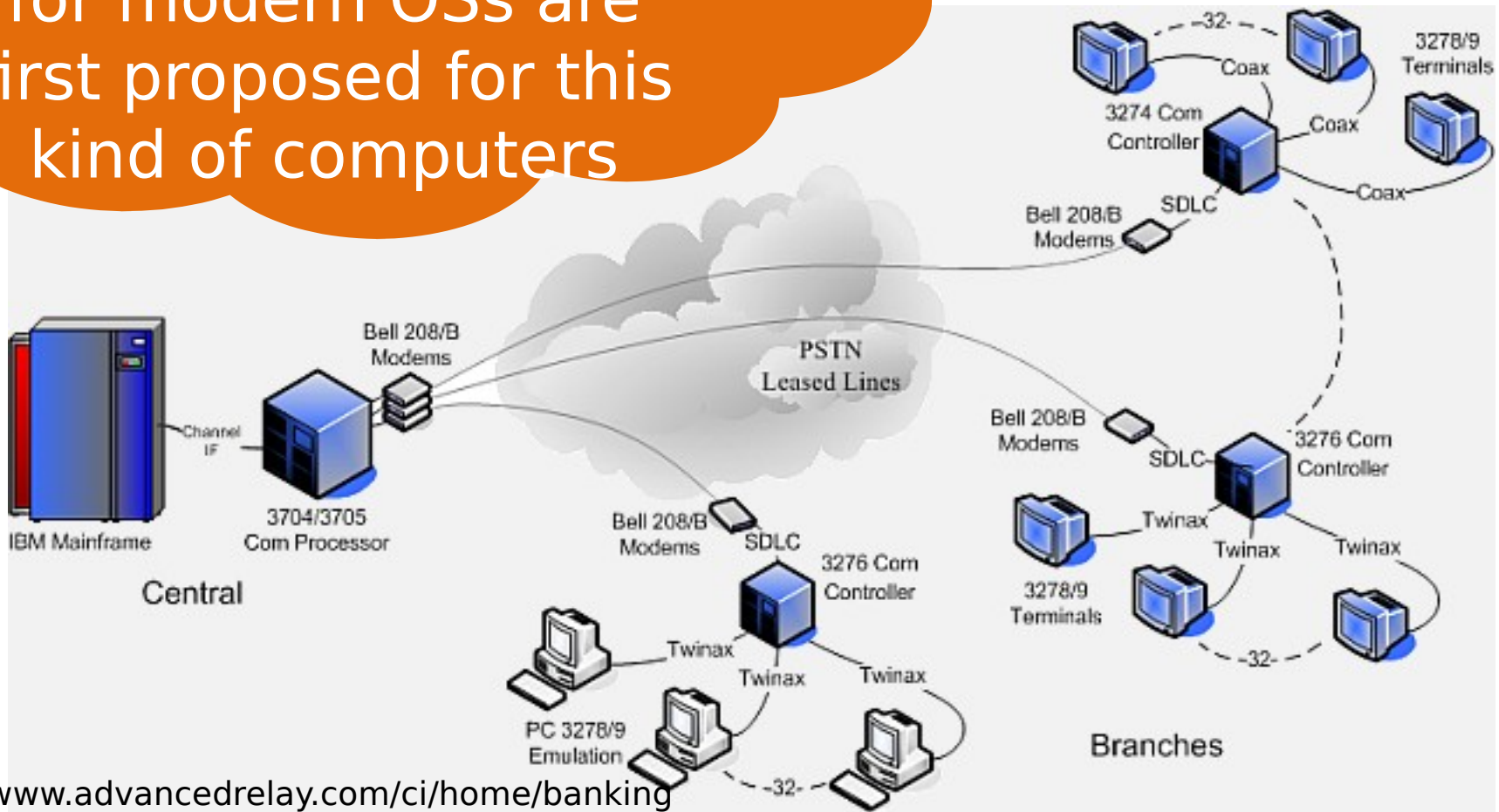
- And, the concurrency makes OS more complicated.
  - By **CONCURRENCY**, it means
    - **MANY** programs are running **IN A PERIOD**; but **ONLY ONE** is running **AT A TIME**!
    - You’ ve benefited from this by using modern OSs – Windows, Ubuntu, Mac OS  
      - You can listen to some music, “**at same time**”, you can edit some documents, maybe you can also read some materials for references
  - **<u>Concurrency is now the indispensable property of modern OSs!</u>**

Concurrency is now the indispensable property of modern OSs



- Concurrency is more critical for larger com
  s for many clients
  s mainframe

In fact, most topics (problems and ideas) for modern OSs are first proposed for this kind of computers



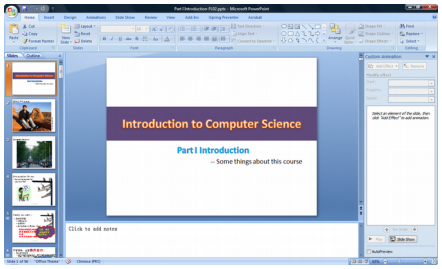
http://www.advancedrelay.com/ci/home/banking

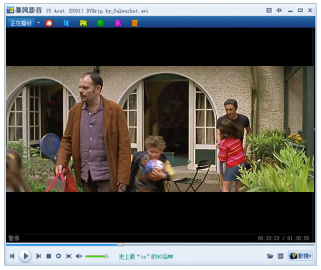
- **However, benefit always comes along with pains**! Concurrency leads modern OS to be more complicated
- Mapping 1 ☾ “**SMALL MM V.S LARGE PROGRAM**” **problem** becomes more rigid even with larger and larger MM

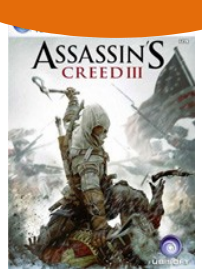
1999	A high-end Personal Computer	128,000,000	128MB
2001	High-end Computers	Over 4,000,000,000	4 GB
2002	Large-scale Mainframe Servers	64,000,000,000	64 GB
Future	Terabyte Memory Computers	Over 1,000,000,000,000	1 TB

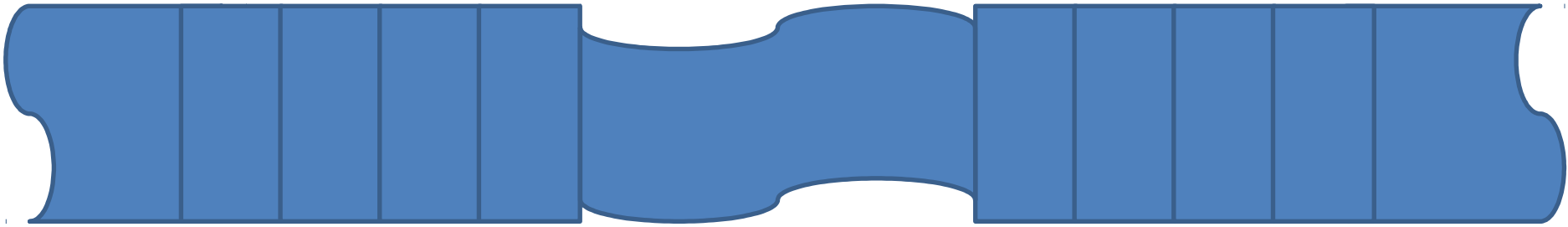
- Main memory [主存]
  - Linear addressed

User's requests could always break down the ability of the provider's









0010 0011 0012 0013 0110 0111 0112 0113

- Idea to overcome “**SMALL MM V.S LARGE PROGRAM**” **problem**?
- **MULTIPLEX**ing MM!
  - Cut programs into smaller segments, and copy the segment (containing instructions for execution) when needed
  - This is like you paint a long picture on a small desktop
    - Of course you should find other position to keep the rest.

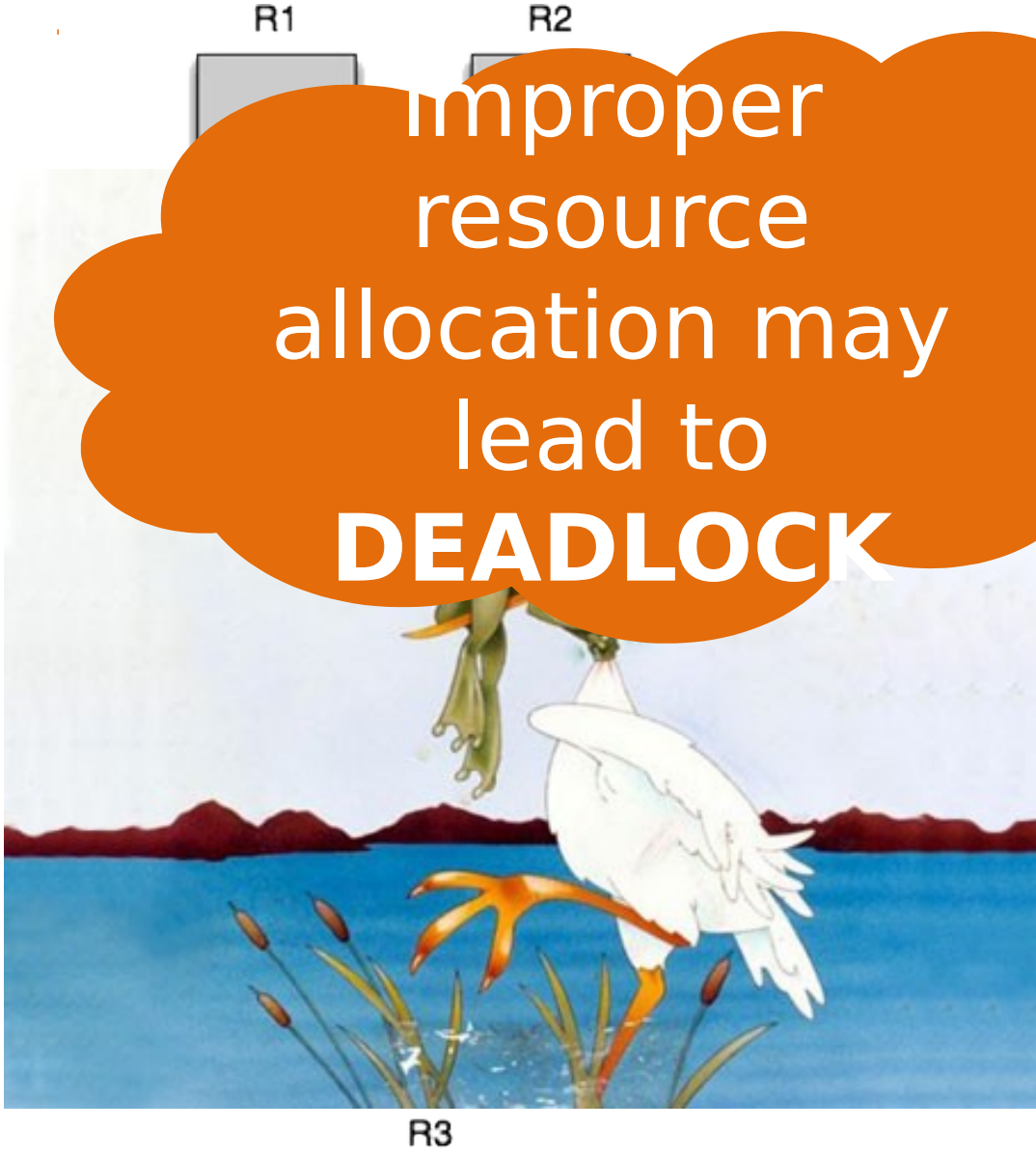
- Pay attention to another keyword – **PARALLEL**
  - Don’t be confused with **CONCURRENCY**.
- Parallel means to do many things at same time!
  - In computers, this always means a computer has many CPUs. This is the property of modern server cor
  - This also draw the long picture

- Execution 《 “1 CPU V.S MANY PROGRAM S” problem

- Idea to overcome “**1 CPU V.S MANY PROGRAMS**” **problem**?
- **MULTIPLEX**ing CPU!
  - This means to switch CPU among those concurrent programs quickly
  - This is like you paint several pictures on a small desktop, not one picture after another, but “one part of one picture after another part of other pictures”
    - Of course you should find other position to keep those pictures.

Concurrency also leads to other considerations

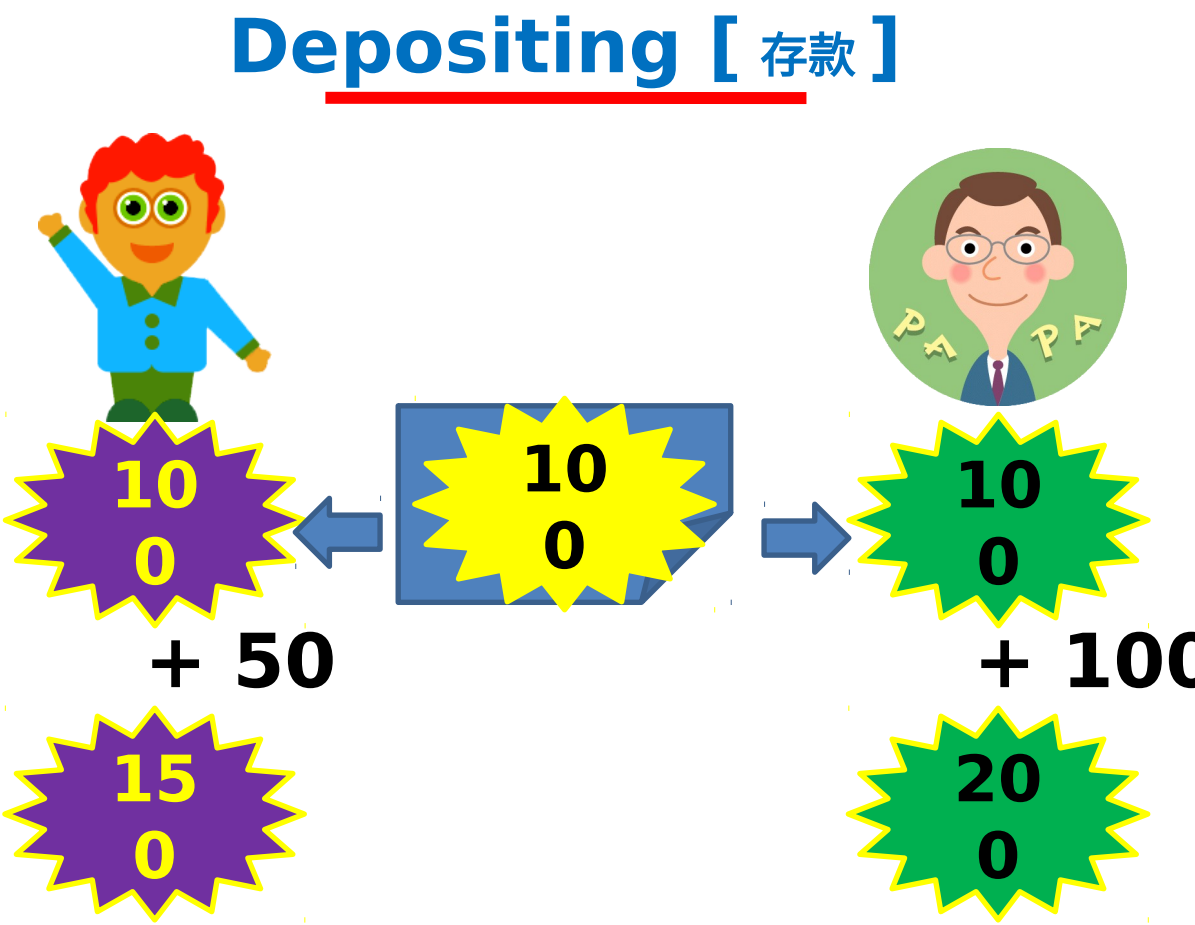
- **PROBLEM** of **RESOURCE COMPETITION**
  - Each program needs some resources – memory, HDD, CPU, etc.
  - There must be competition among programs when running them concurrently




http://go.funpic.hu

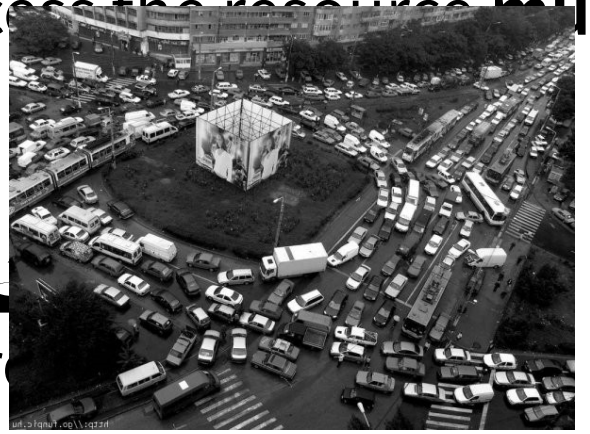
Other considerations

- **PROBLEM** of Resource co mpetition
  - **Data incons istency** is an other issue we should c onsider care fully when c ompeting.



- Idea to overcome "**RESOURCE COMPETITI**
  ON (DATA [illegible] that you should [illegible] LOC
  K)"
- **S**
  d
  - One re
    - The related programs s[illegible] acc[illegible] **tually exclusively**.
  - More than one resources:
    - ?? Complicated, and touch later
    - Could you infer some methods fr
      your own traffic experience?

conclude the conditions causing this kind of situation, as you cope with the real life problems , such as the light could not be turned on, etc



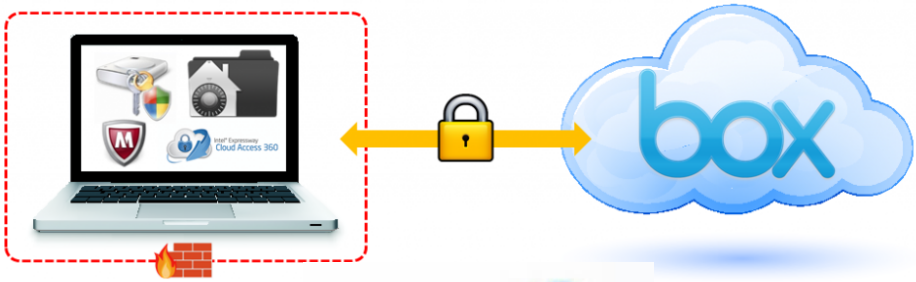
- By “**mutually exclusively**”, it’s like you and other persons share one washing machine, WC, coffer, etc.

coffer ['kɔfə] n. 保险柜，保险箱

  - No others could use this resource when you are using it.
- Could you infer the key for this?
  - The **lock** (password)!
  - and the rules to use it
    - **Apply for** the lock **first**. You can use the resource only if you succeed.
    - You should **release** the lock **after** you finish to use the resource.

SECURITY?

- **PROBLEM**: The essence is the data security!
  - **INSIDE** – in a computer system
    - Everyone can only access his/her own data! – **AUTHORIZATION**.
  - **OUTSIDE** – during the data transmission
    - No one could hack the data flowing in the network – **ENCRYPTION**.



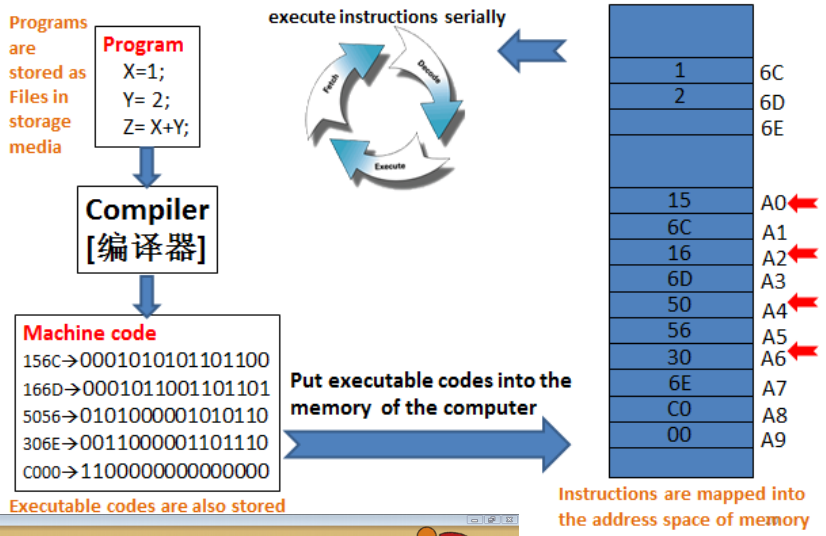
Other considerations

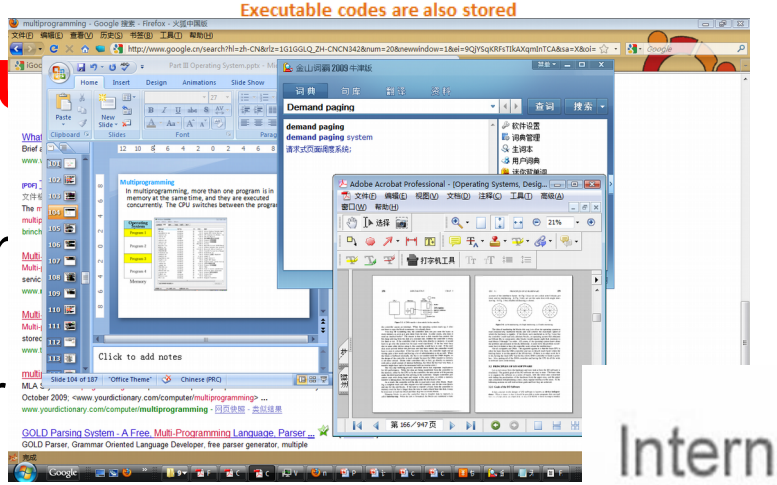
- Advanced topic – multi-processor system
  - All those issues listed before (**multiplexing**, **Synchronization**, **deadlock**, **protection** etc.) should be reconsidered

🛍 this part will be covered in this course, which depends ….

3 hints to follow!

- How to manage res ources for
  - Running **A PROGRA M**?
  - Supporting **CONCU RENCY**?
    - 1 CPU, many progr ms
    - Small MM, large pr am
    - Synchronization prob lem
  - Keeping **SECURITY**?
    - Control illegal access





and,

Not “**E**lectro-**M**agnetic **M**easurement 电磁测量”

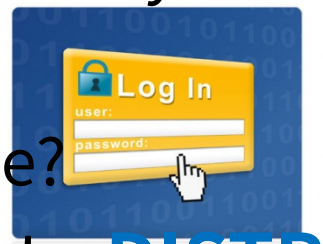
- 3 core tasks – **EMM**
  - **EXECUTION** ☾ CPU
    - Configurations for executing one program
    - Switching execution of concurrent programs
  - **MAPPING 2** ☾ from File to Hard disk space
    - How to represent files in hard disks?
      - This is the fundaments to understand storing data in other media
  - **MAPPING 1** ☾ from File to Main Memory space
    - Problem of Small MM v.s Large size of programs
      - How to allocate space
        - » for a program?
        - » for many programs?

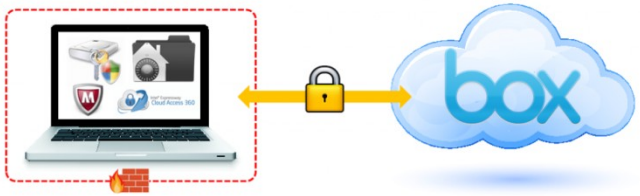
and,

Not "**G**reenwich **S**idereal **D**ate: 格林尼治恒星日期

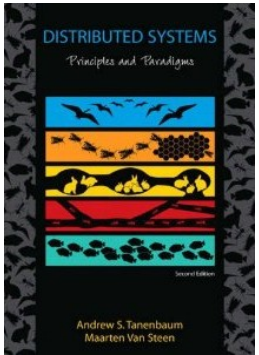
- 3 value-added tasks – **GSD**
  - **GRAPHIC USER INTERFACE**
    - How to provide friendly interface for use

  - **SECURITY**
    - Inside and Outside?





  - Extension of EMM to **DISTRIBUTED** systems
    - Multiprocessor system ☾ Parallel
      - Many programs could run at the same time now!
    - Data consistency problem and Deadlock are more u



Support to execute many programs concurrently [with limited resources] [Local **Friendly** Remote **Distributed**

OS's responsibility [programming]

Interface:
GUI system (based on OO)

Computing:
Providing services for remote computers

Carry out the security strategy:
Control the resource access safely

Carefully sharing resources among programs:
Multiplexing + Synchronization

Manage the resources:
Define the necessary information (Data Structure) for those resources

Limited resources: 1:1:M – von Neumann

- Other fundamental problems
  - How to identify the program you want to run?
  - How to represent and organize files in HDD?
    - With the help of CO to understand how those bits are recor ded
  - How to allocate limited MM to programs?
  - How to execute the ready program?
    - You have learned in **C**ompu
  - How to support GUI?
  - How to carry out inner
  - How to carry out CPU switch… nism, …
- All we need is to design **data structure and relat ed operations**

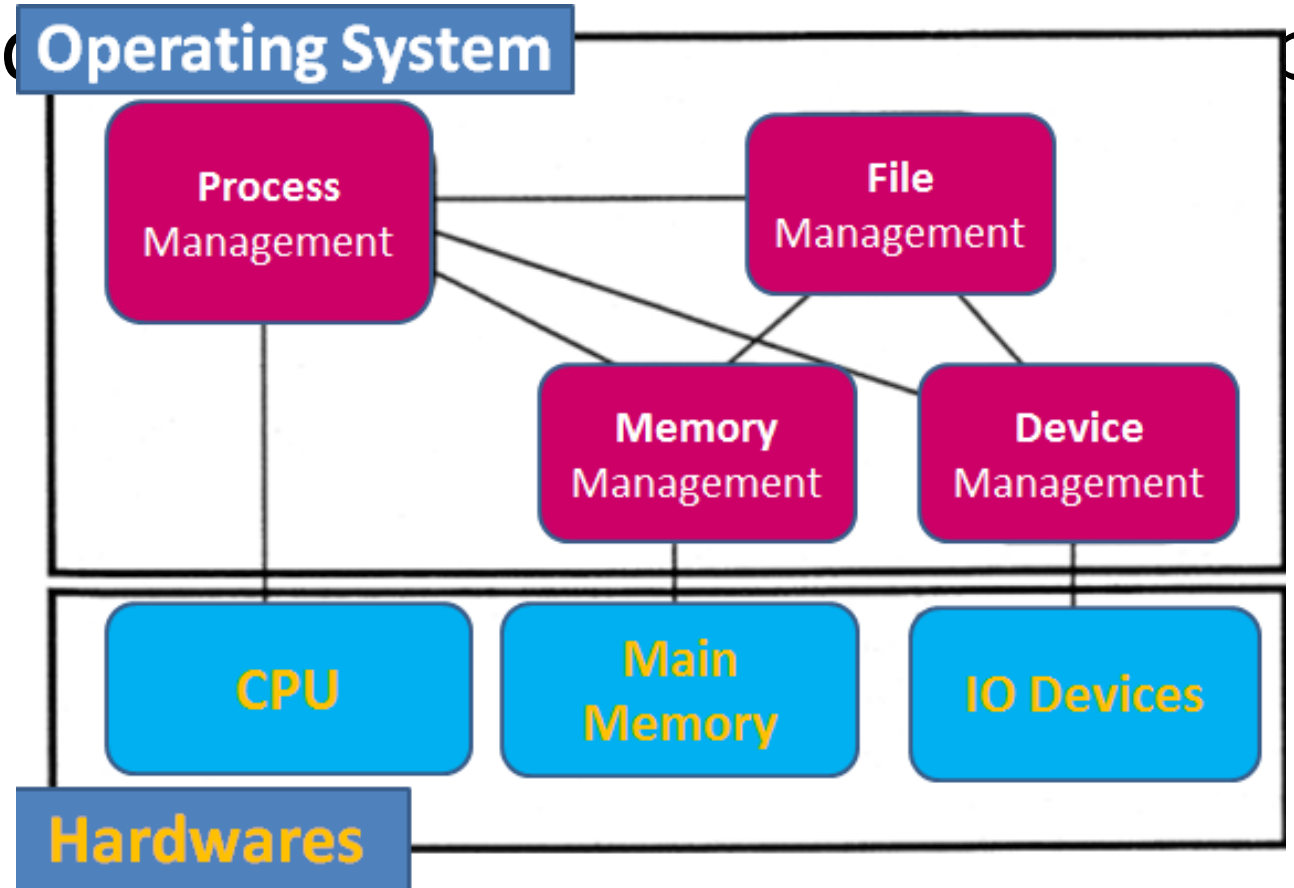
Yes, DSA is important!

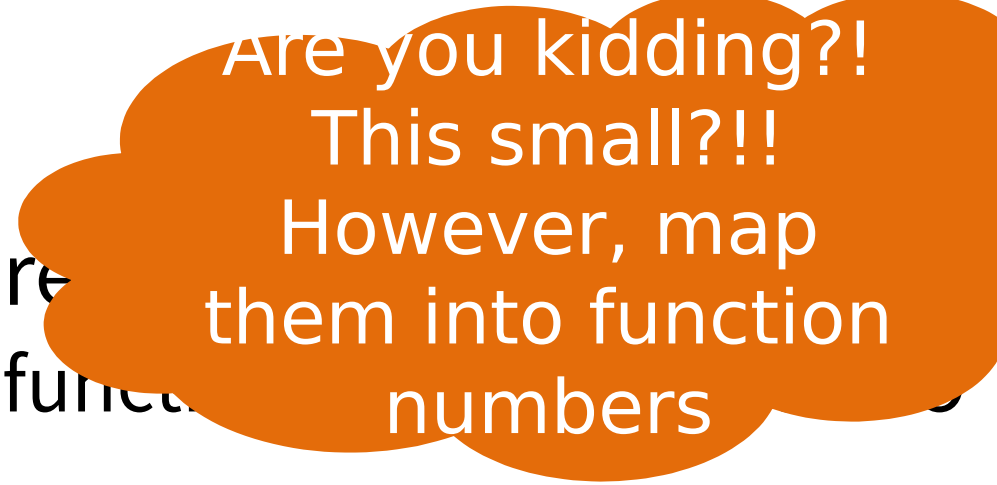
Ser
vice
s

- Problems and ideas when implementing OS
  - Problems: EMM+GSD
  - Ideas: Multiplex resource + Synchronize programs
- Construct OS
  - Evolution of OSs’ s structures
    - Simple structure/monolithic structure
    - Modular, Layered, Microkernel
    - Trend – Virtual machine
- How to use the functions defined in OS?
  - System Call/API
- How to load and run OS?

We’ ve learned – 4 components

- We can distill 4 kinds of resources / concepts
  - **CPU**, **Main Memory**, **Hard Disk**, **File**
- Mo ponent s





- However, OS is more
  - There are so many funct… S!
- Kernel size (**EMM+S**, without those drivers, etc.) of many OSs
  - Unix 32/V (VAX) 180k
  - BSD 4.3 (VAX) 640k
  - Mach 2.0 (VAX) 1000k
  - SunOS 4.03 (Sun 3, Sun 4) 2400k
  - Windows NT (x86) 4000k

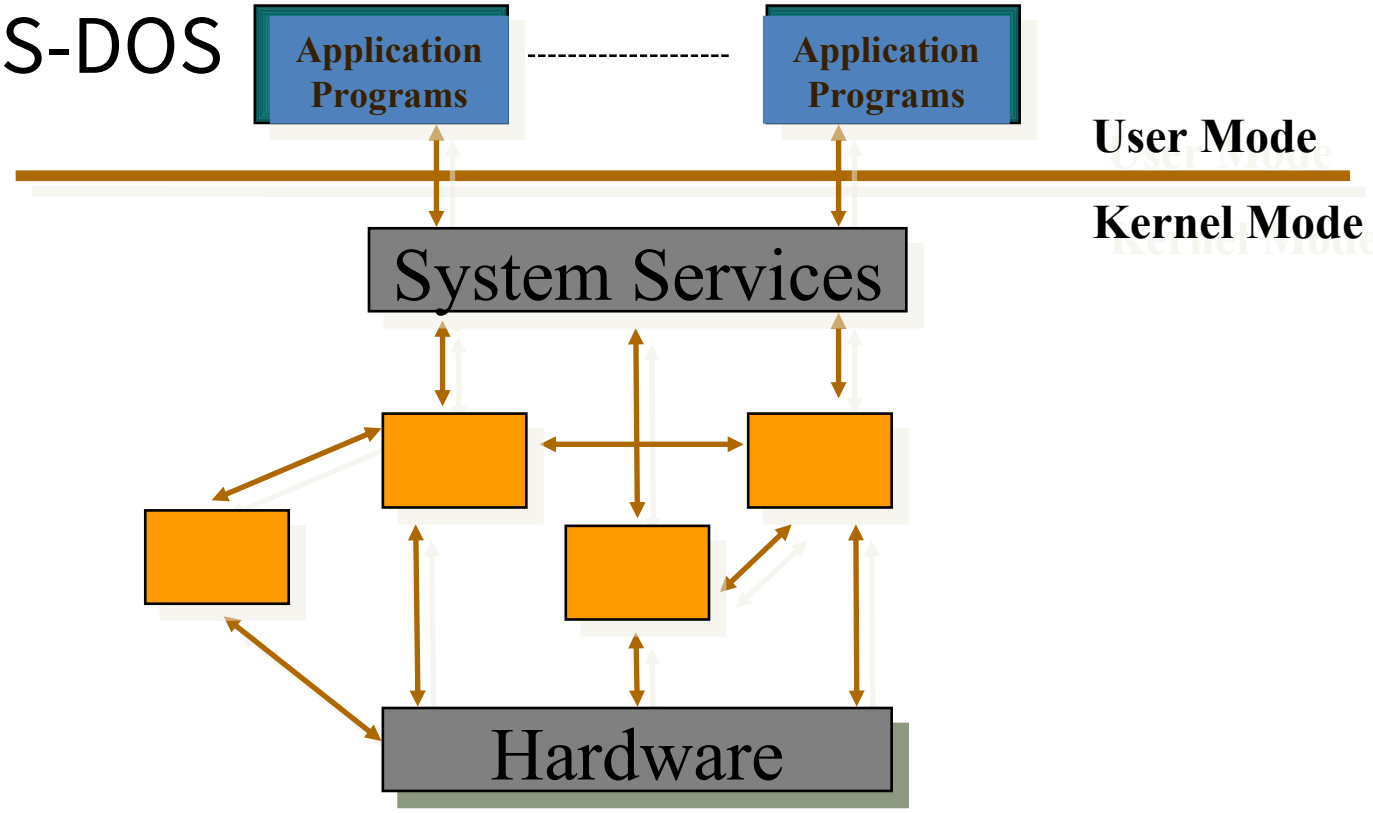
http://www.gordoni.com/os-sizes.htm

- Based on my experience in Code::Blocks, the C code of whole classic linked list is 3KB. Therefore,
  - Unix 32/V (VAX) ~60
  - BSD 4.3 (VAX) ~214
  - Mach 2.0 (VAX) ~334
  - SunOS 4.03 (Sun 3, Sun 4)
  - Windows NT (x86) ~1334
- Have you ever
es?
  - Like 2
ve som

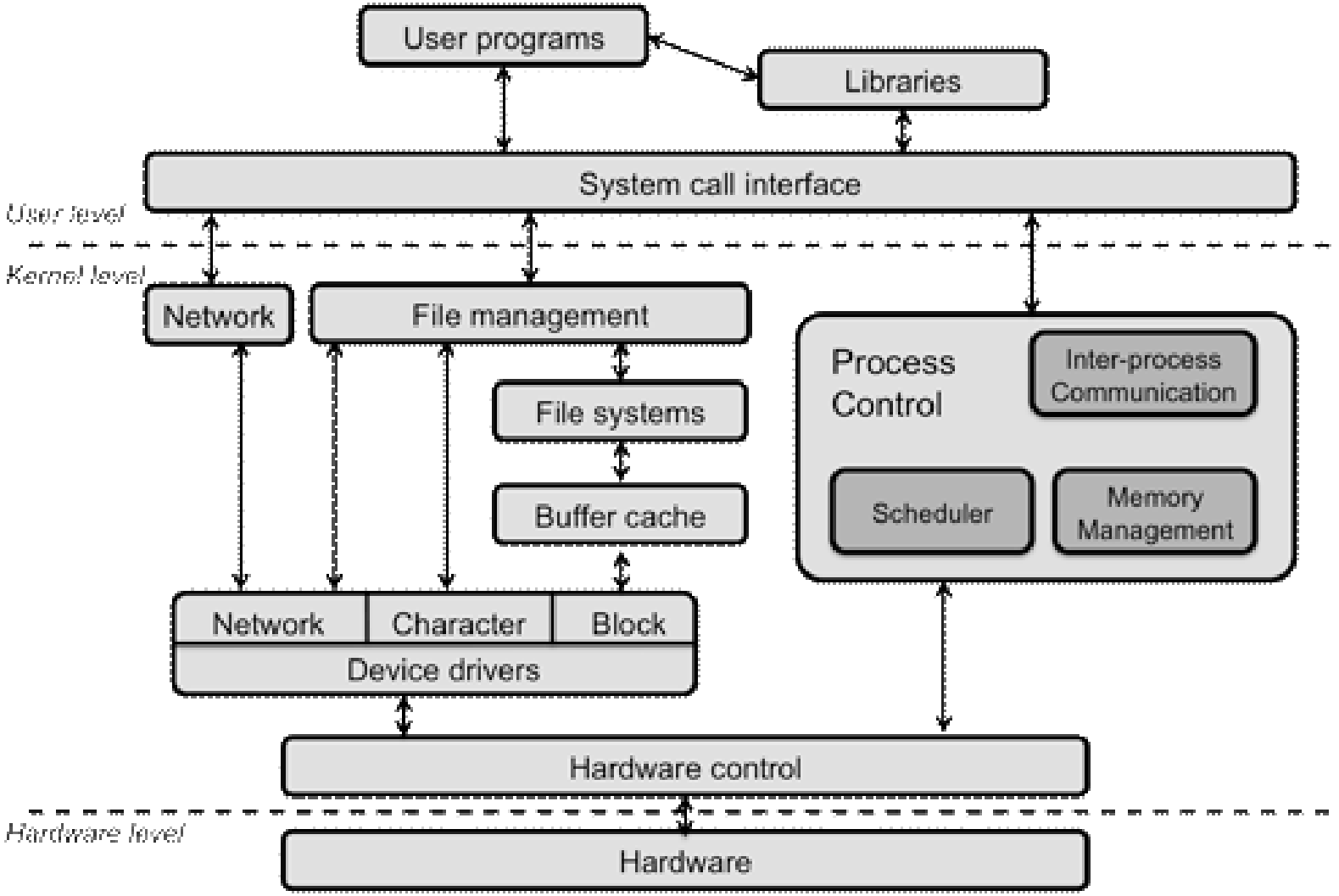
Organized structure! OS uses this too!

BTW, do you remember bonus? You could do me a favor to find more precise information about the implementation of current OSs?

- **Modular structure (Monolithic)**
  - Better application Performance
  - Difficult to extend
  - Ex: MS-DOS

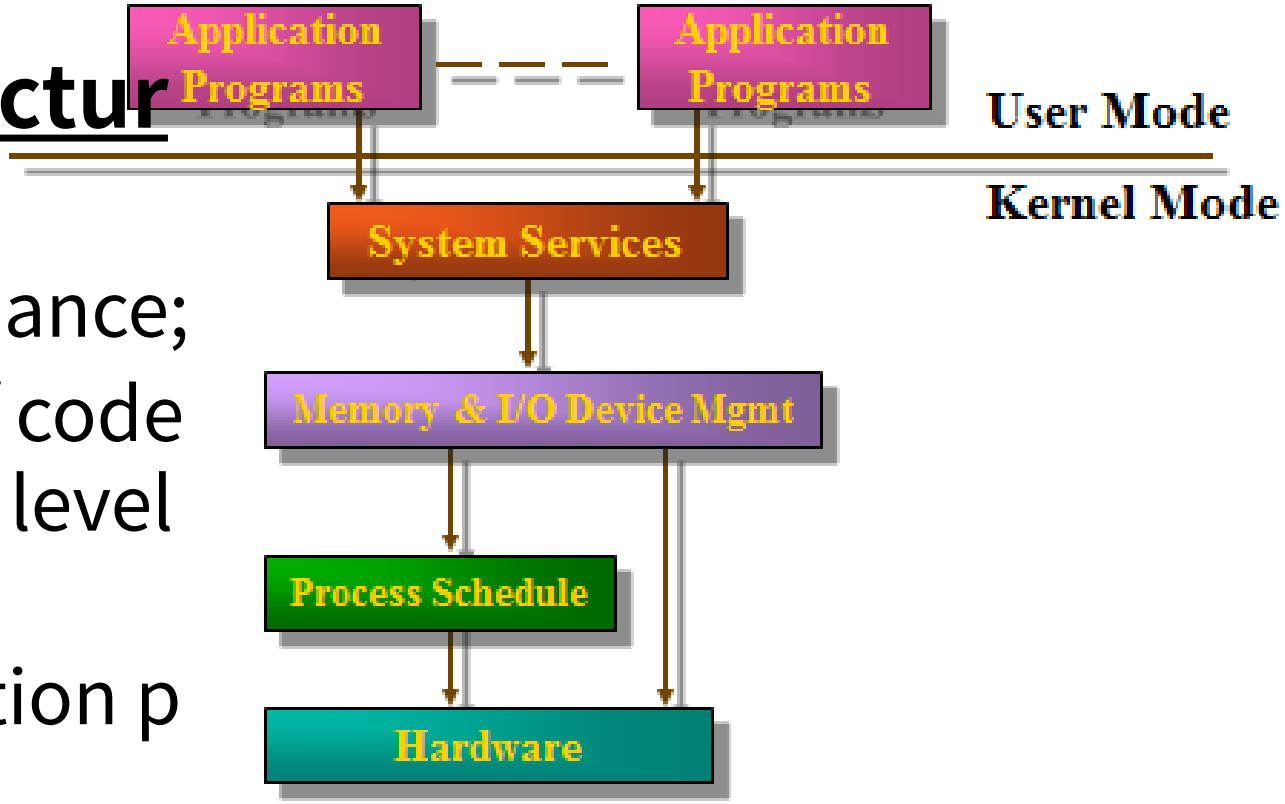


Structure of a typical operating system



http://www.cs.rutgers.edu/~pxk/416/notes/03-concepts.html

- **Layered structure**
  - Easier to enhance;
  - Each layer of code access lower level interface
  - Low-application performance
  - Ex: Old Unix



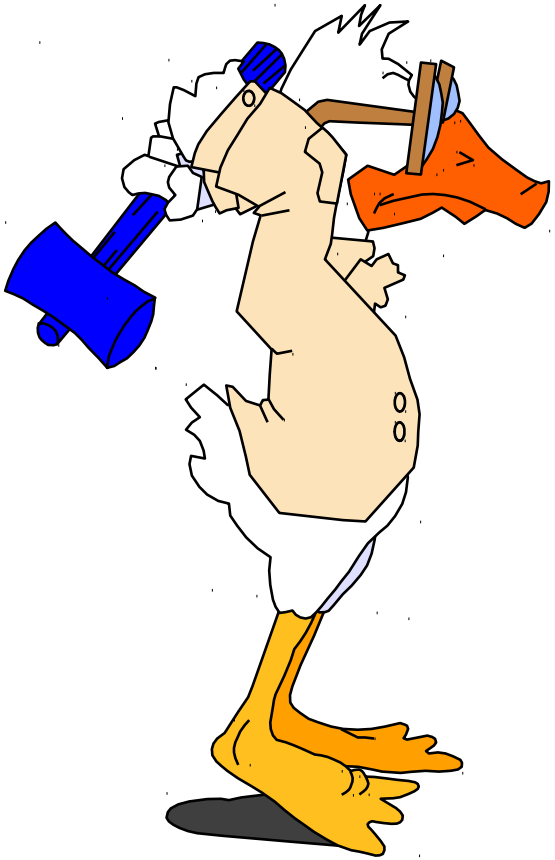
Traditional OS

Application Programs

Application Programs

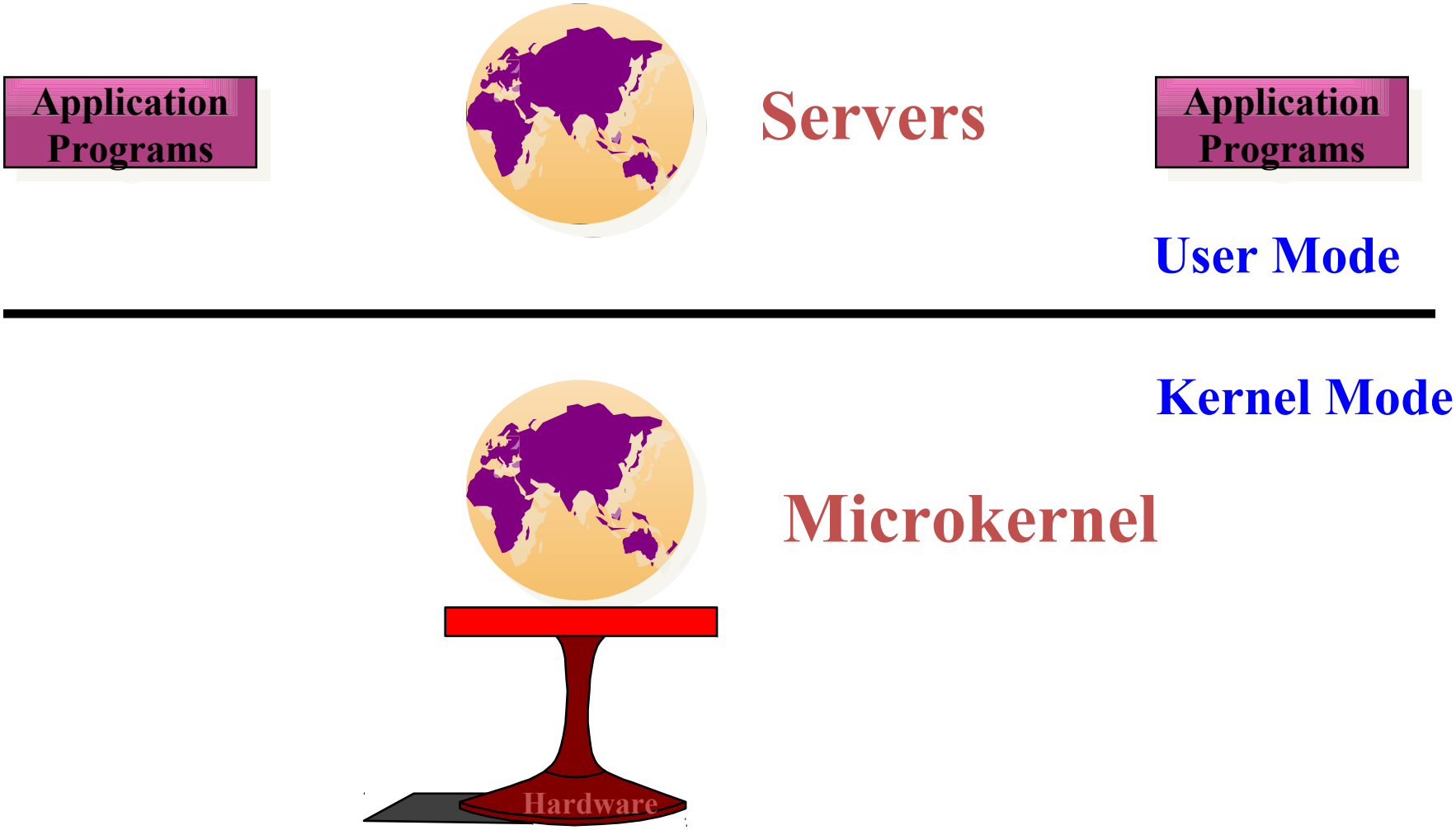
User Mode

Kernel Mode

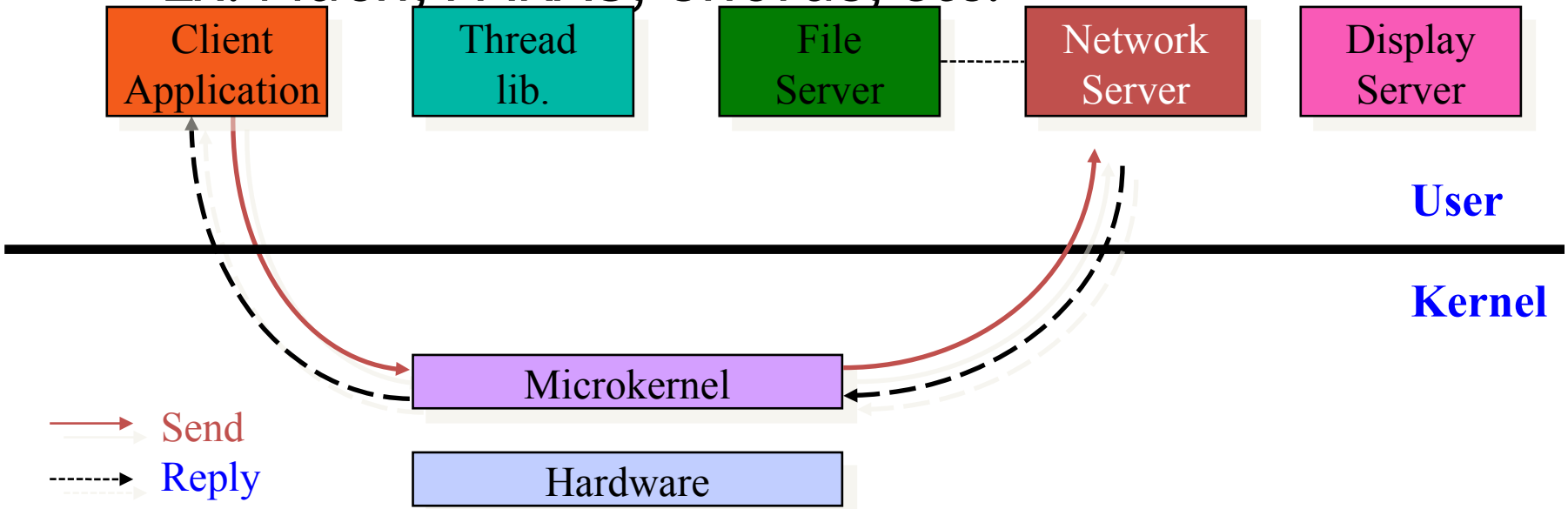


OS Designer

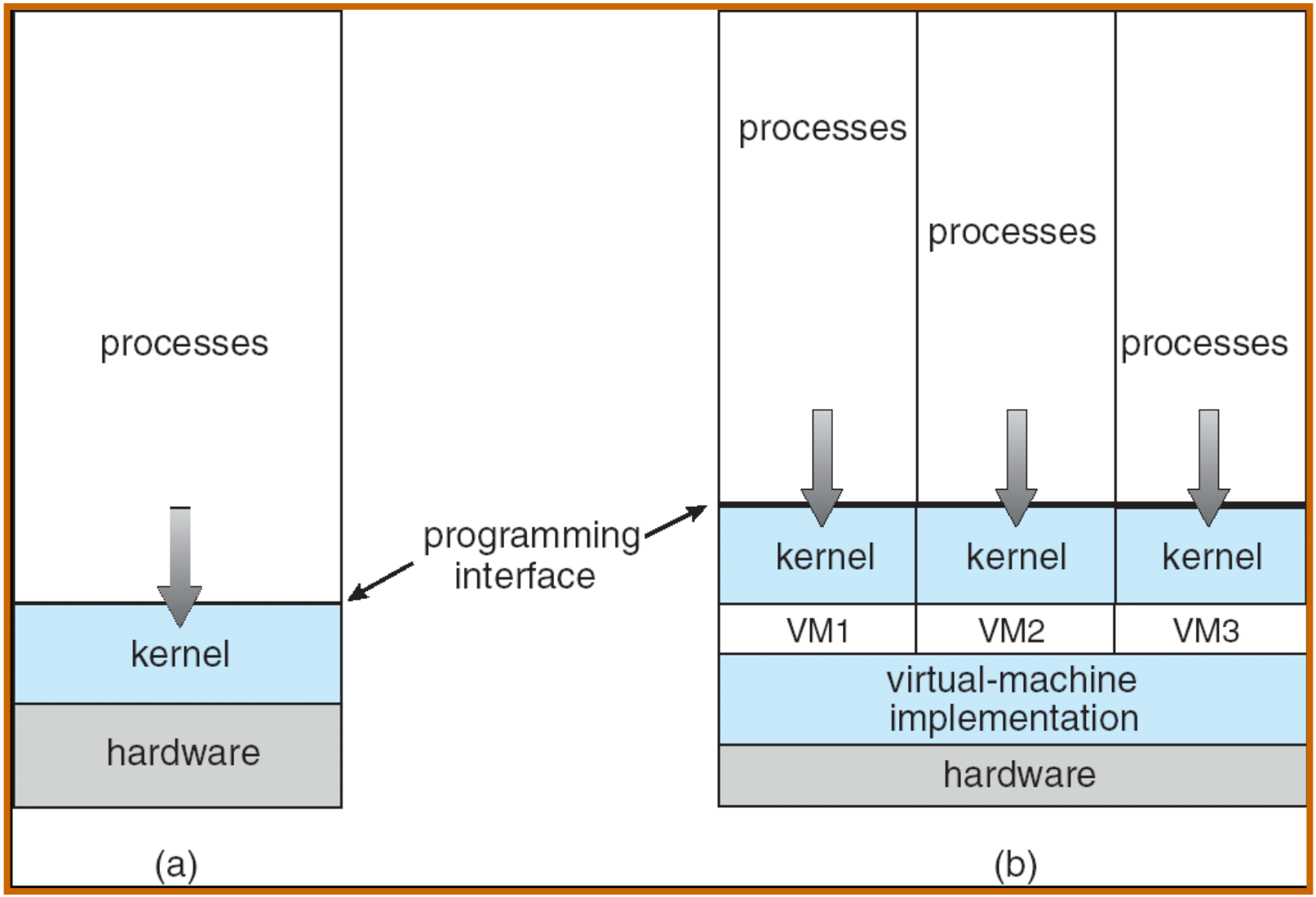
New trend in OS design



- **Microkernel/Client Server OS 1990s**
  - Tiny OS kernel providing basic primitive (process, memory, IPC)
  - Traditional services becomes subsystems
  - Monolithic Application Perf. Competence
  - OS = Microkernel + User Subsystems
  - Ex: Mach, PARAS, Chorus, etc.



- **Virtual Machines**: simulate OSs on one co mputer hardware
  - Hardware is abstracted into several different ex ecution environments
    - Virtual machines
  - Each virtual machine provides an interface tha t is identical to the bare hardware
  - A *guest* process/kernel can run on top of a virtu al machine.
    - We can run several operating systems on the same *host*.
    - Each virtual machine will run another operating sys tem



Non-virtual Machine

Virtual Machine

- However, VM idea was first proposed in 1970s
  - The **C**onversational **M**onitor **S**ystem(CMS) is a single-user operating system, while the OS/370 and DOS/370 are multiprogramming operating systems.
  - A user process is unaware of the presence of the VM/370 —it sees only the guest OS that it uses.

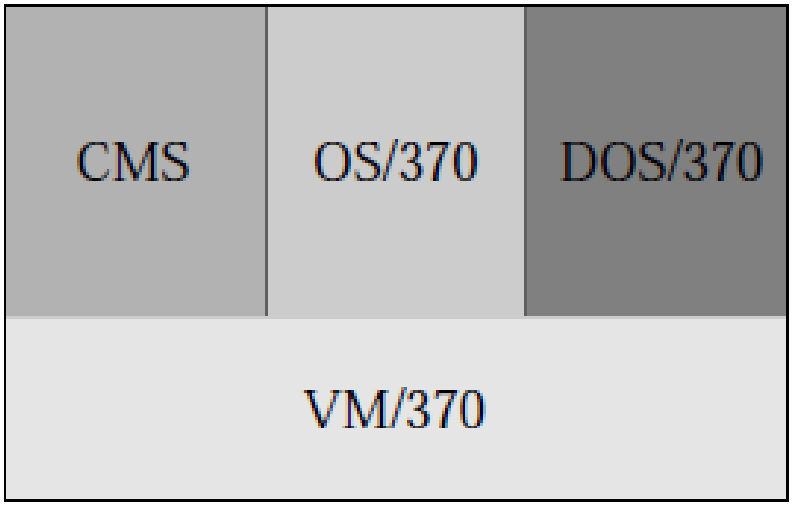
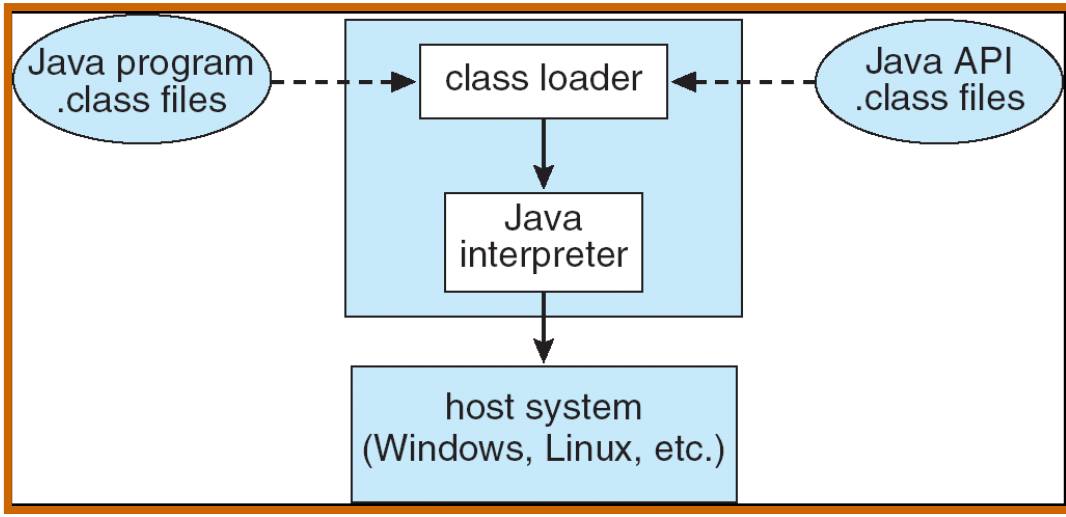


Figure 4.5 Virtual machine operating system VM/370.

VM: Examples

- VMware
  - Abstracts Intel X86 hardware
- Java virtual machine
  - Specification of an abstract computer
- .NET Framework

The Java Virtual Machine

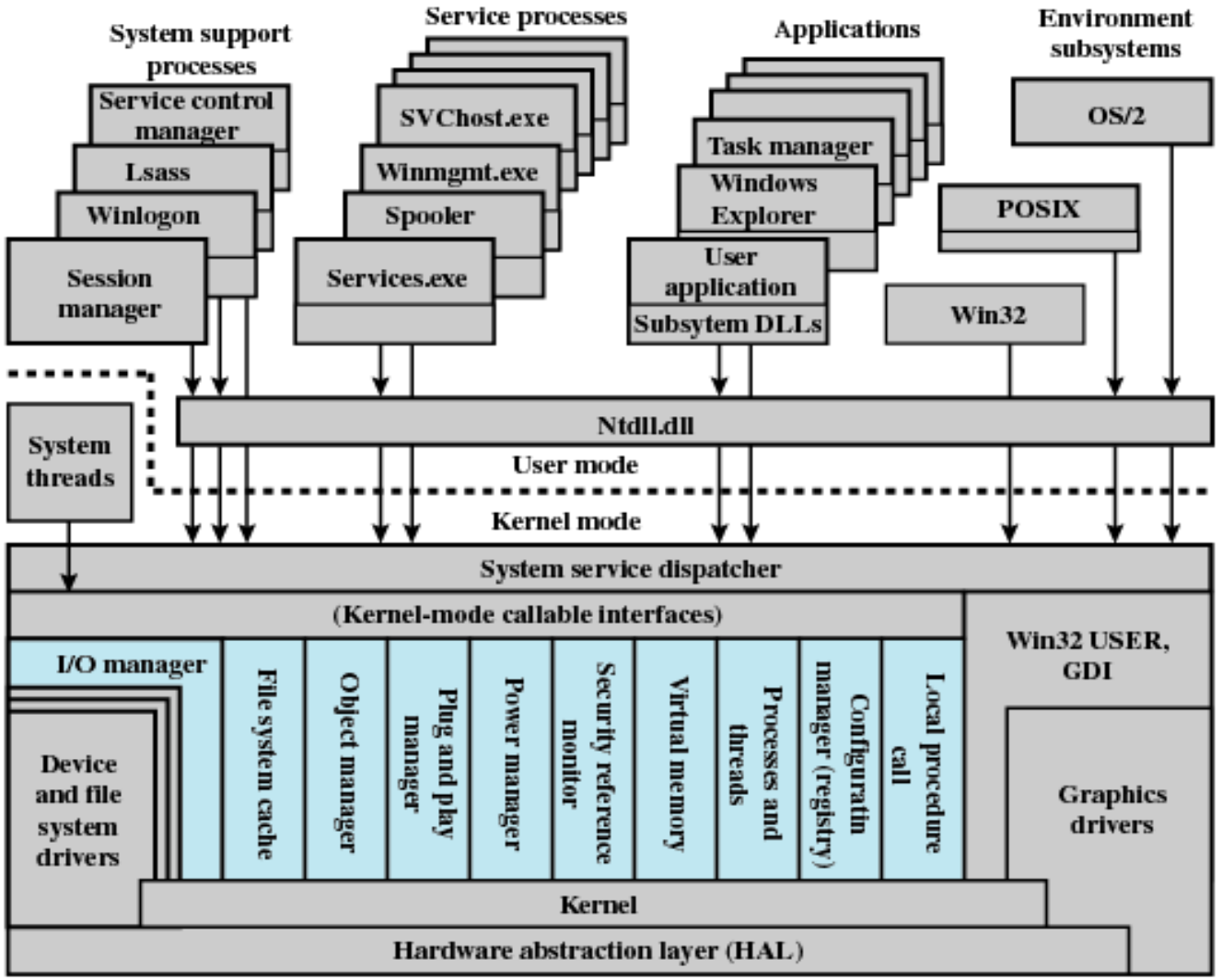


Java consists of

1. Programming language specification
2. Application programming interface (API)
3. Virtual machine specification

The Java Virtual Machine allows Java code to be portable between various hardware and OS

Hybrid

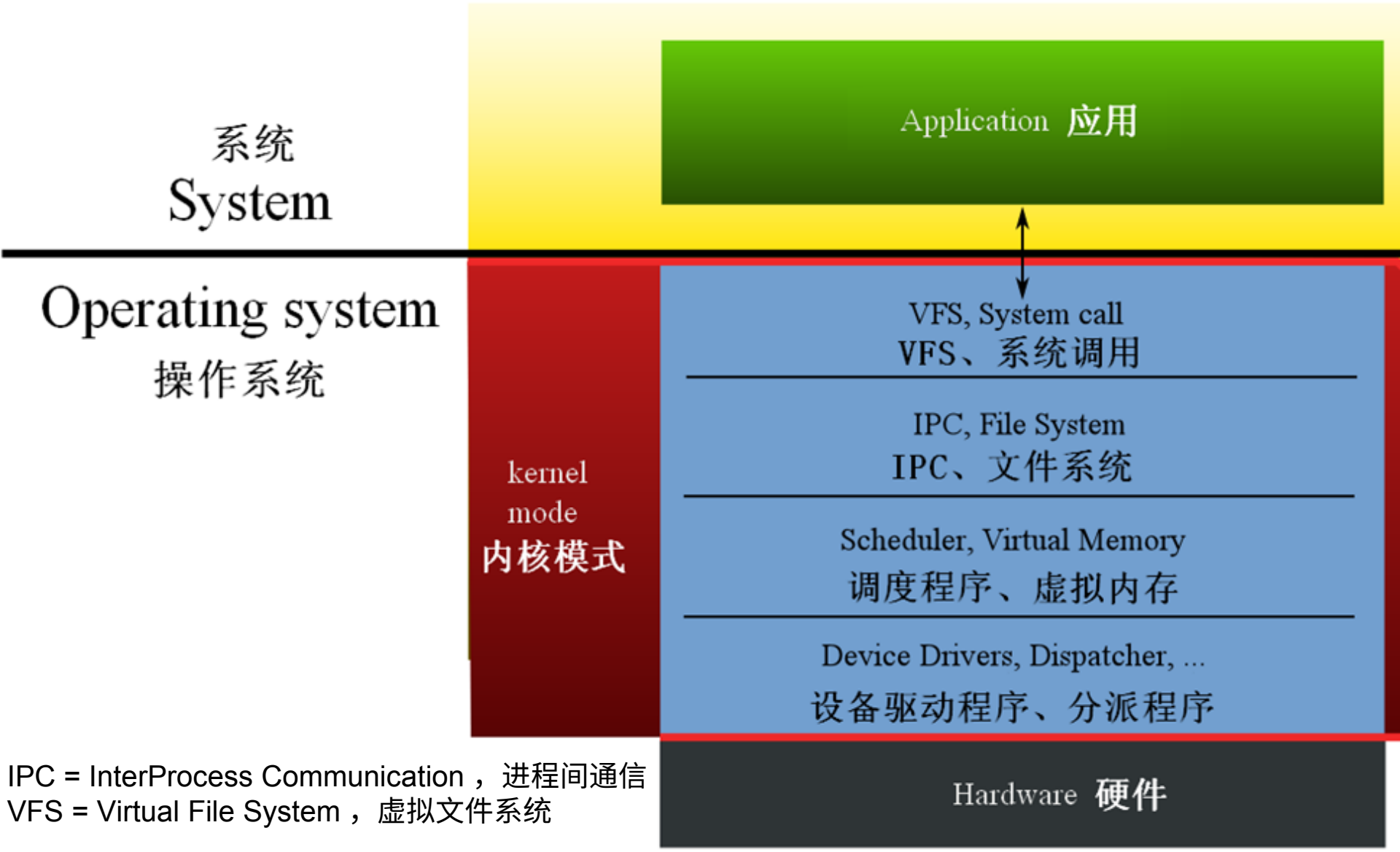


Lsass = local security authentication server
POSIX = portable operating system interface
GDI = graphics device interface
DLL = dynamic link libraries

Colored area indicates Executive

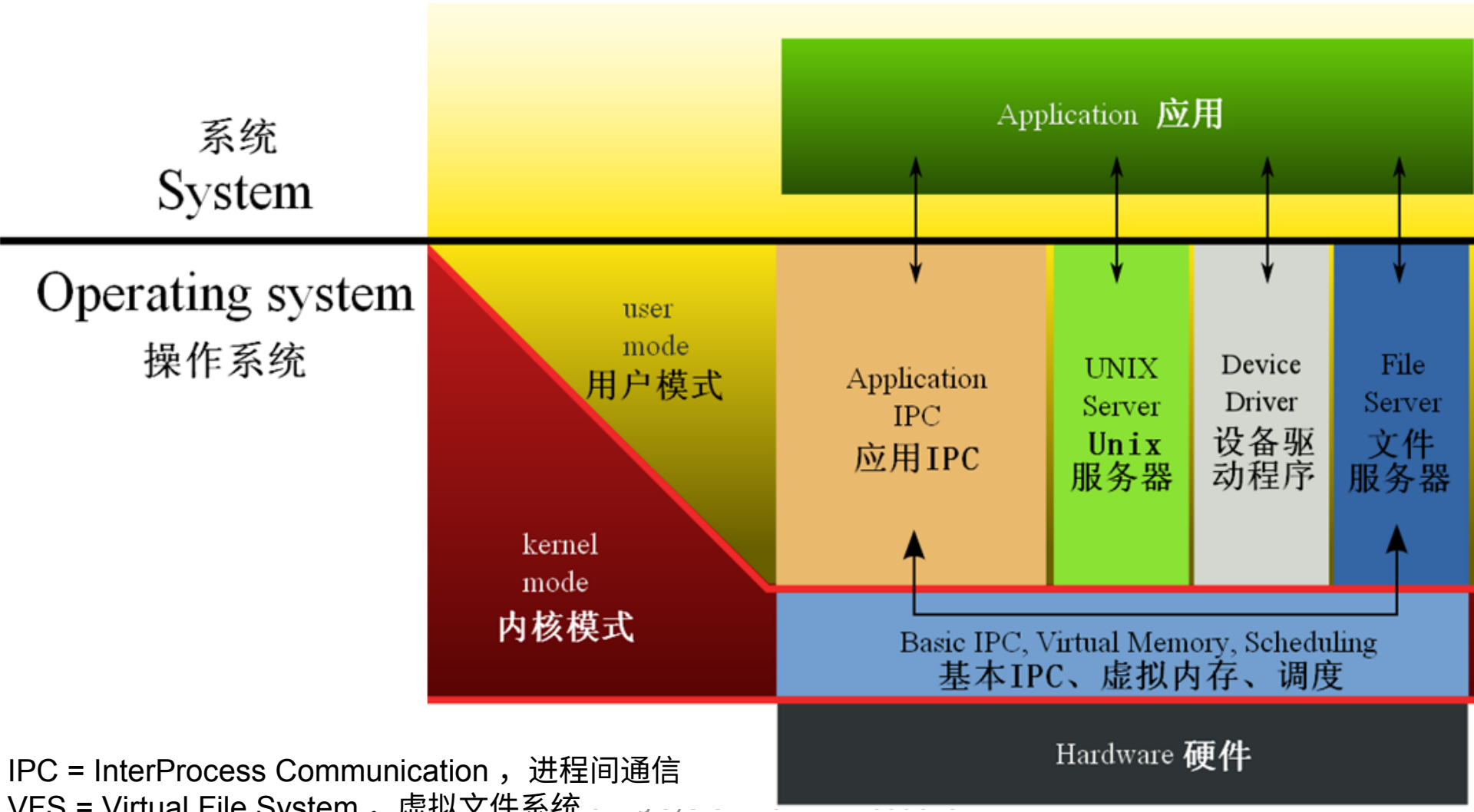
Figure 2.13 Windows 2000 Architecture [SOLO00]

基于单体内核的操作系统
Monolithic Kernel based Operating System



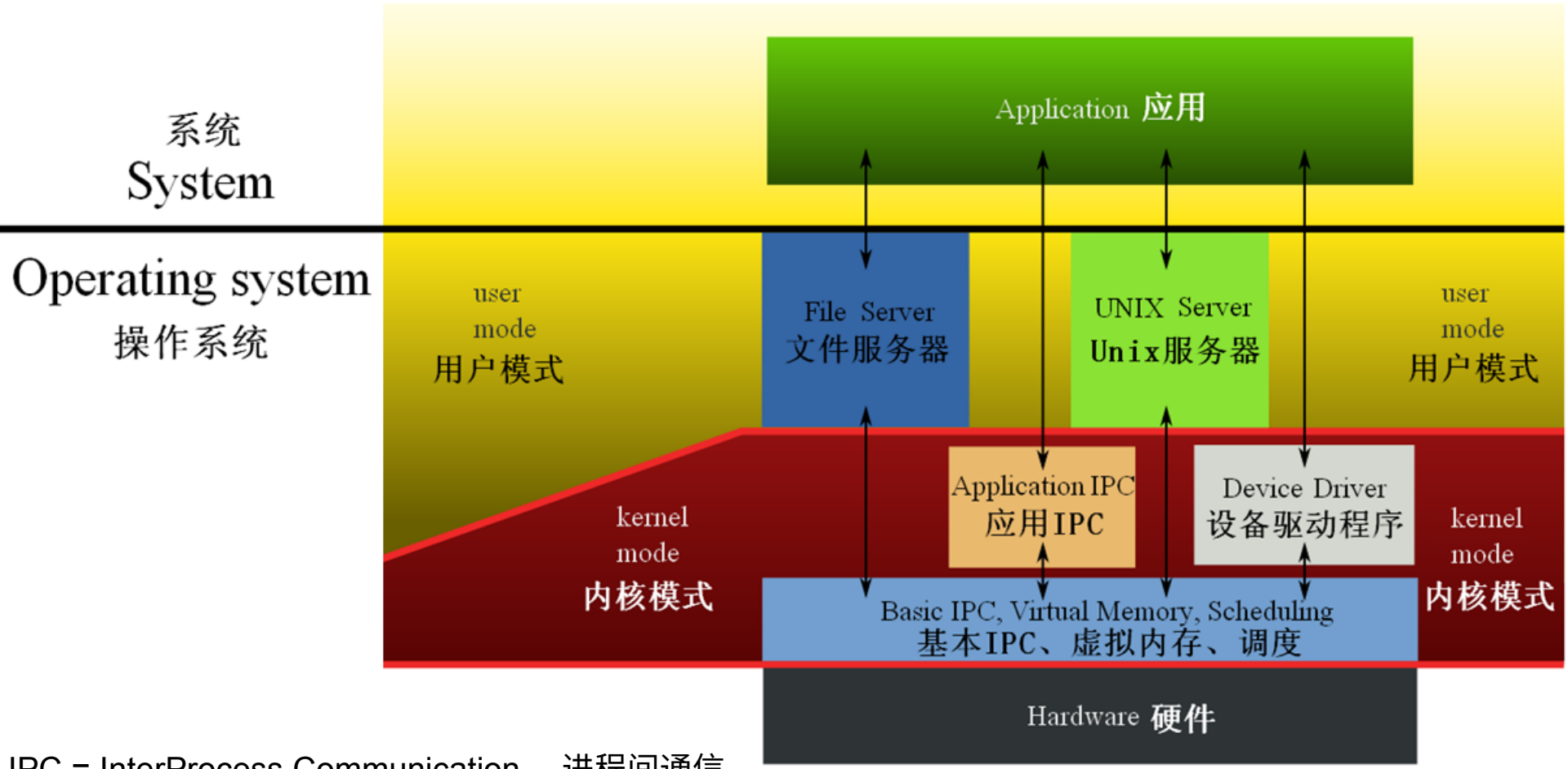
IPC = InterProcess Communication，进程间通信
VFS = Virtual File System，虚拟文件系统

基于微内核的操作系统
Microkernel based Operating System



IPC = InterProcess Communication，进程间通信
VFS = Virtual File System，虚拟文件系统

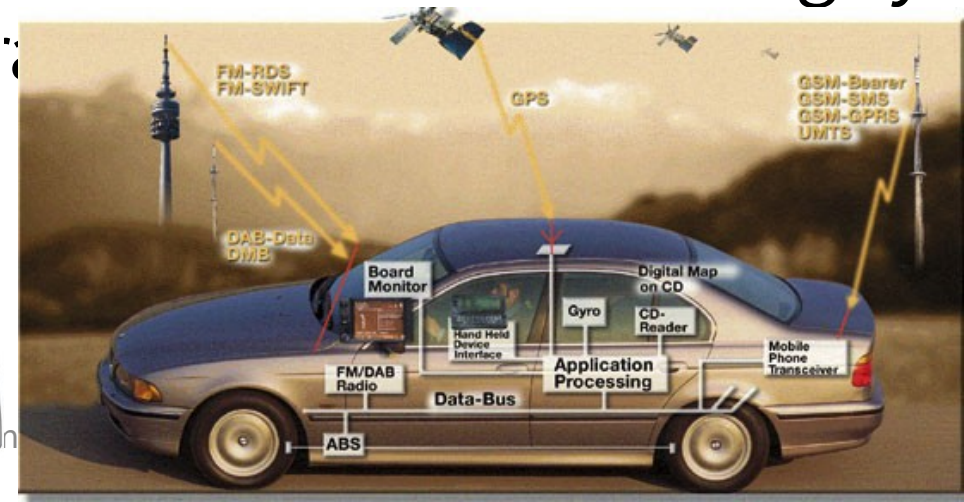
基于“混合内核”的操作系统
"Hybrid kernel"
based Operating System



IPC = InterProcess Communication，进程间通信
VFS = Virtual File System，虚拟文件系统

Implementing OS

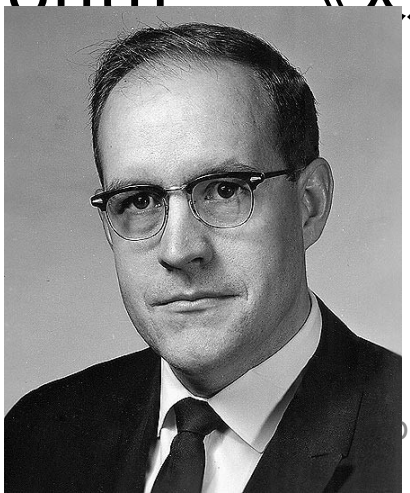
- Traditionally, operating systems have been written in assembly language.
- Now, however, they are most commonly wr itten in higher-level languages such as **C** or C++
  - This is why C is so important, and now is popul ar again because of the boom of embedding sy

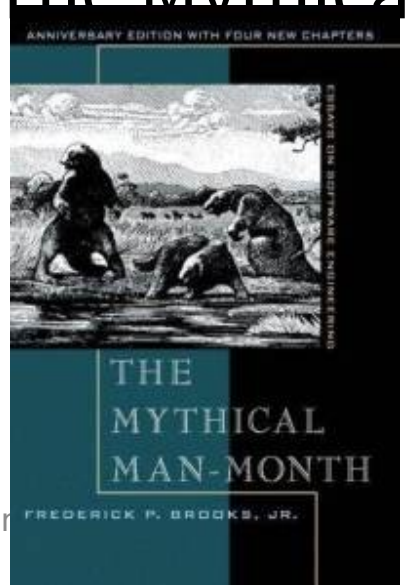


Anecdote: The complexity of implementing OS

- Even triggers the consideration of **Software Engineering**
  - IBM 360

  - Frederick Phillips Brooks, “The Mythical Man-Month” 《人月

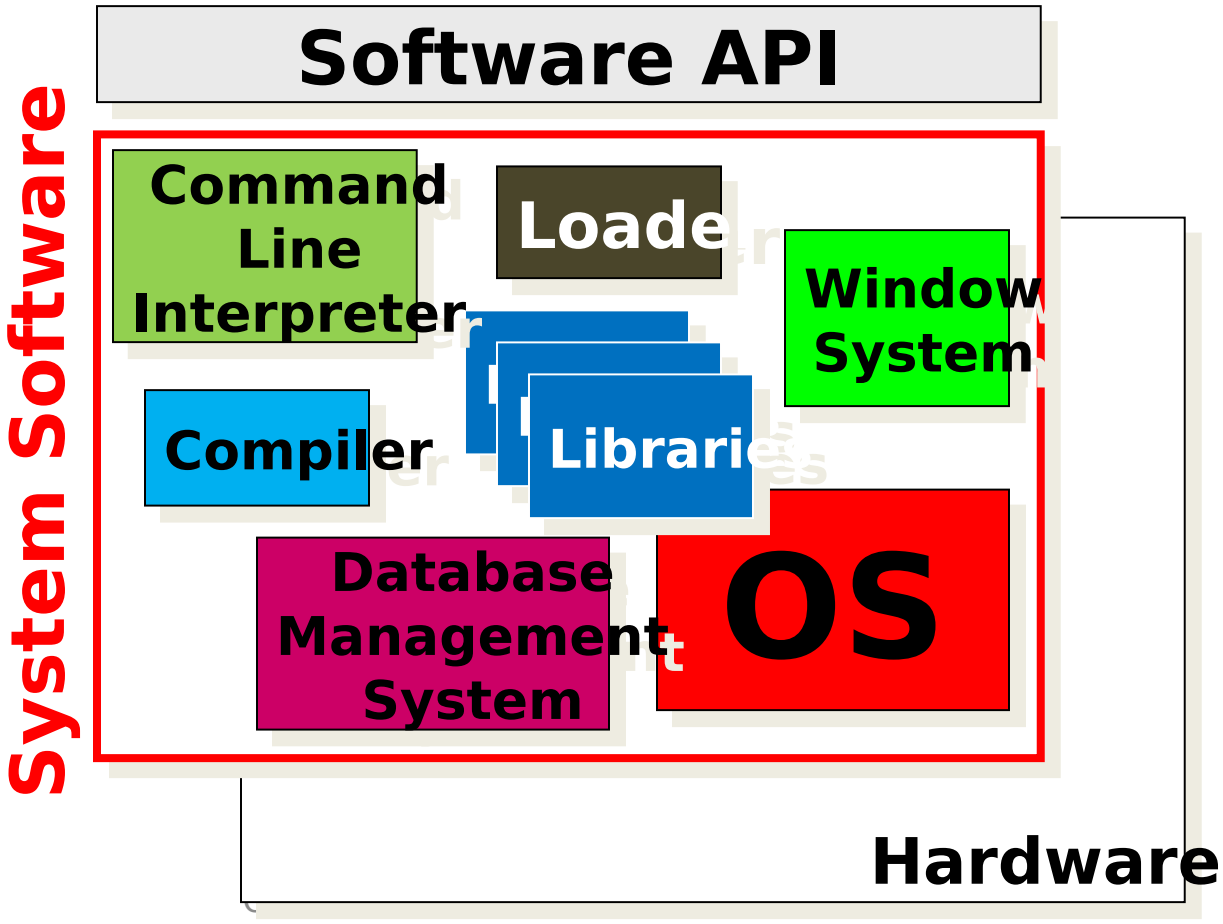


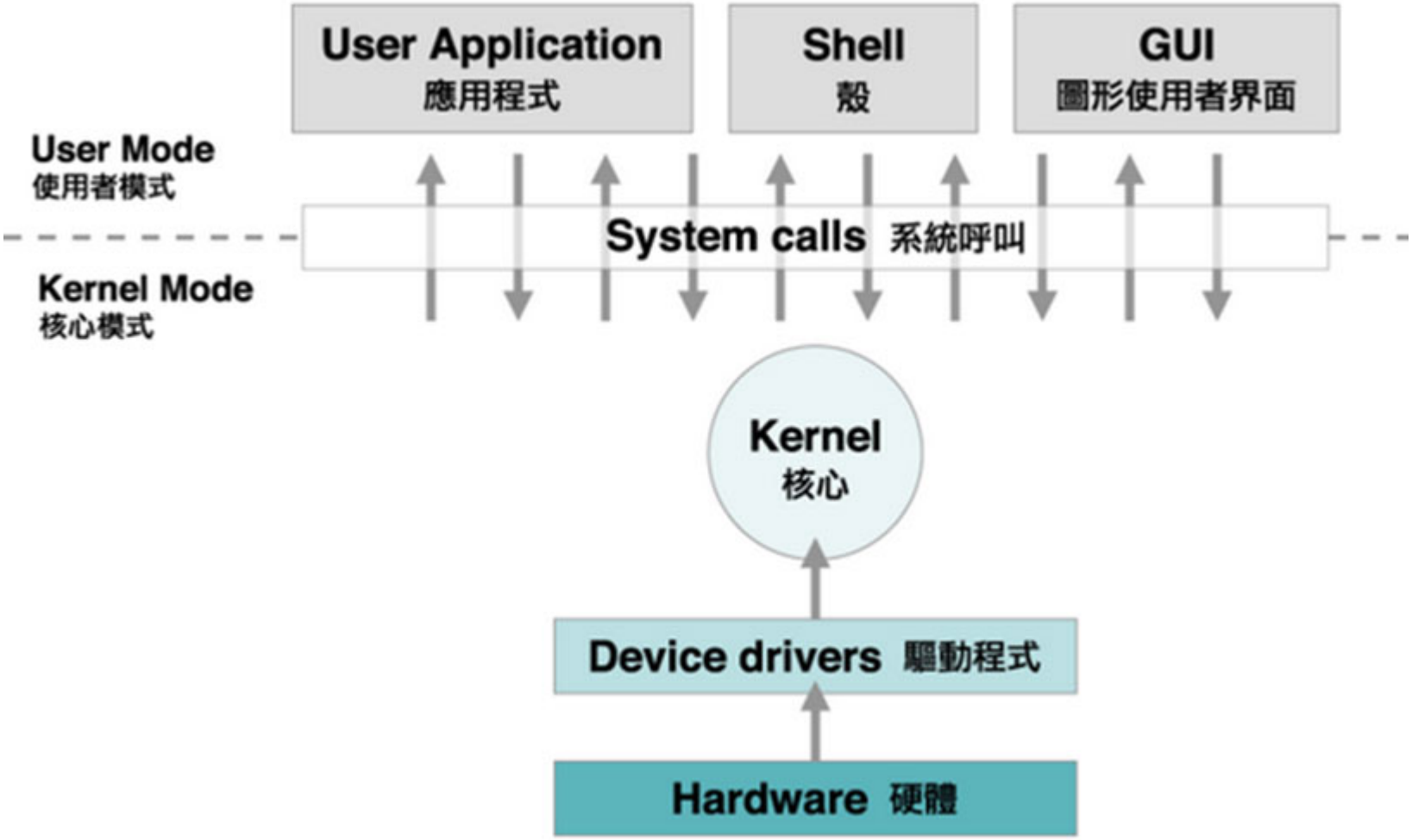


Services

- Problems and ideas when implementing OS
  - Problems: EMM+GSD
  - Ideas: Multiplex resource + Synchronize programs
- Construct OS
  - Evolution of OSs’ s structures
- How do users use the functions defined in OS?
  - System Call/API ≈ Just as Function call you’ ve learned
- How to load and run OS?
  - POST ☾ BIOS/CMOS ☾ (Boot) loader
  - Firmware (ROM)

Application Programmer

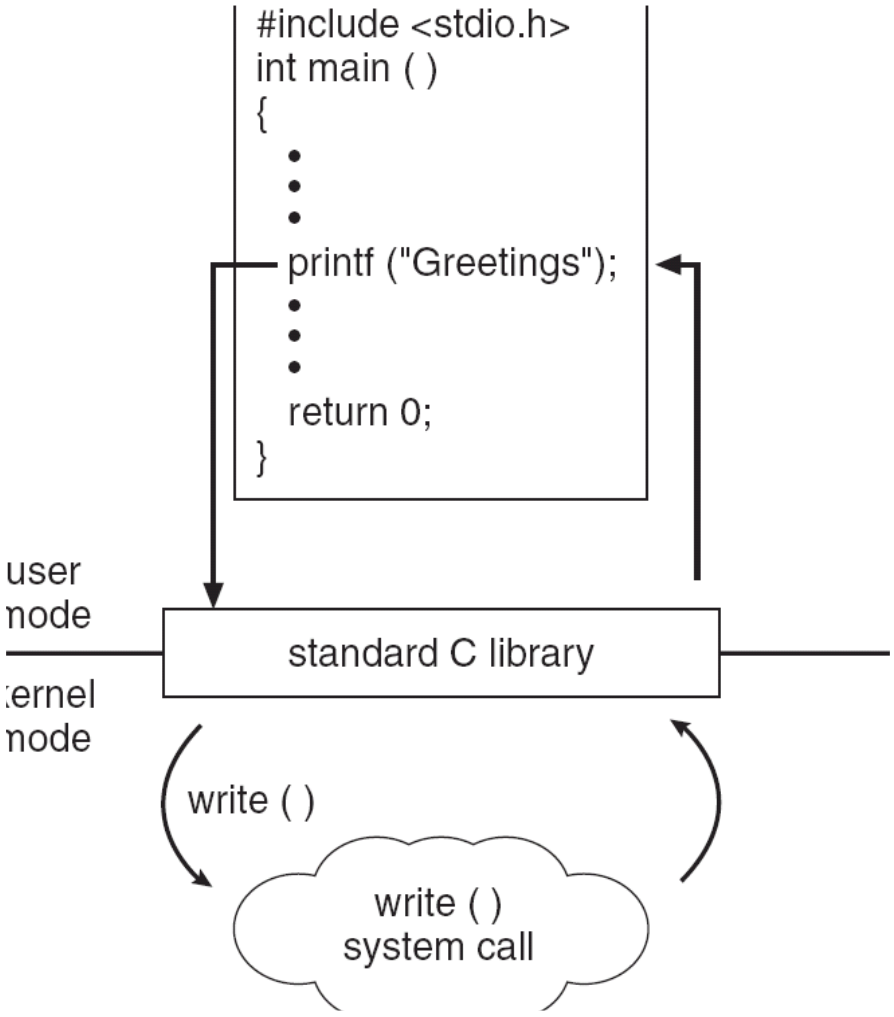




http://mlinking.blog.163.com/blog/static/185801922201311013149 48/

System Call/API:
Application Program Interface 应用程序接
interface between user programs and OS se
rvices

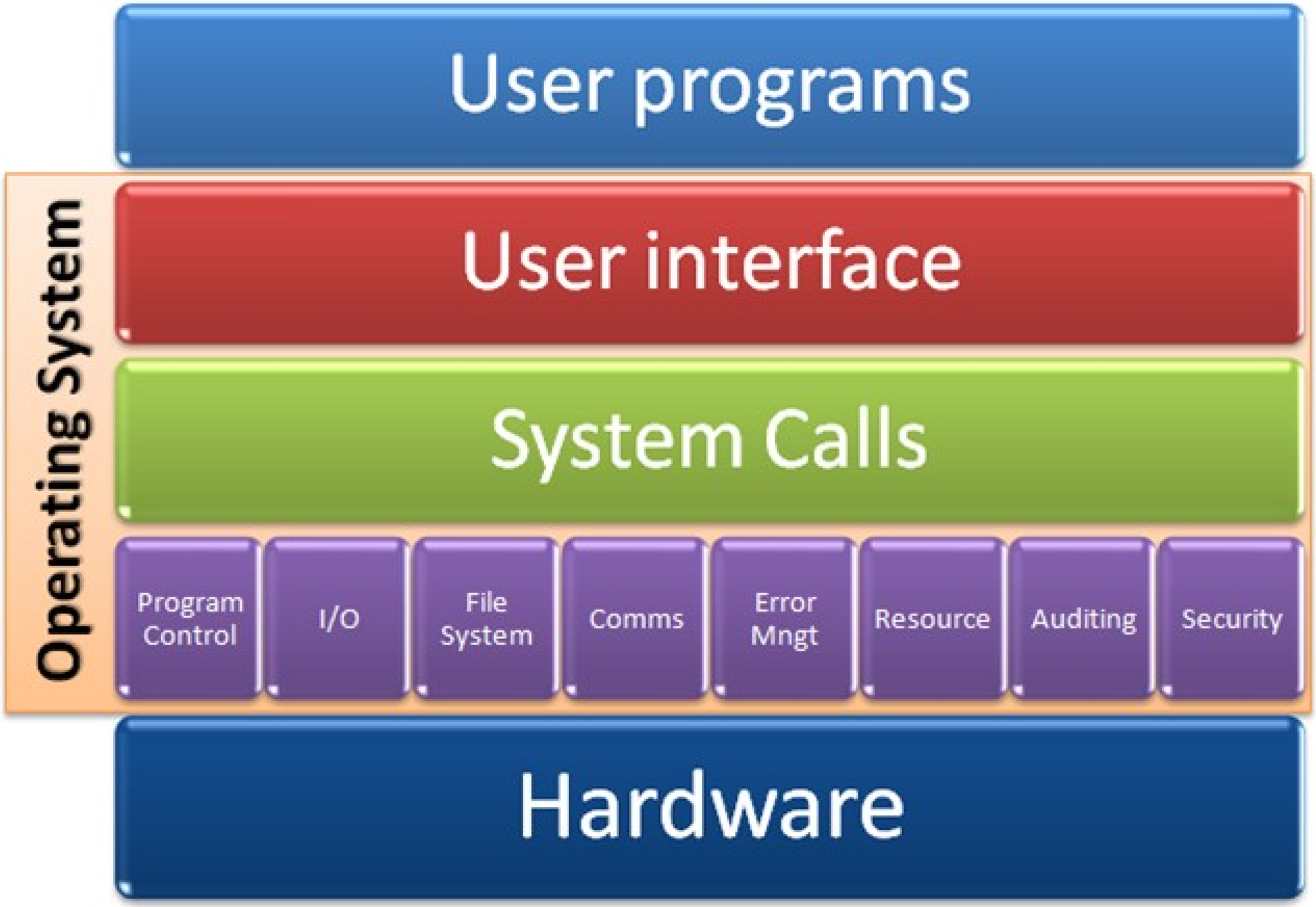
- C program invoking **printf()** library call, wh ich calls **write()** system call



Advice to learn a HPL:

- Besides the learning of the basic programming skills (syntax, function declaration, supported data types & operations, etc.), you should learn how to use the ready-to-use functions provided in predefined libraries.

duction


User programs
Operating System
User interface
System Calls
Program Control
I/O
File System
Comms
Error Mngt
Resource
Auditing
Security
Hardware

Services

- Problems and ideas when implementing OS
  - Problems: EMM+GSD
  - Ideas: Multiplex resource + Synchronize programs
- Construct OS
  - Evolution of OSs’ s structures
- How to use the functions defined in OS?
  - System Call/API
- How to load and run OS?
  - POST ☾ BIOS/CMOS ☾ (Boot) loader
  - Firmware (ROM)

Before you use your OS

- When you turn on the power of your computer, the ***first program*** that runs is usually a set of in structions kept in the computer's read-only me mory (**ROM**).
  - This code examines the system hardware to make s ure everything is functioning properly.
    - 3 subprograms: POST, BIOS, Boot loader
      - Sometimes, loader is believed as part of BIOS in some textbooks
    - This **power-on self test** (**POST**) checks the CPU, memory , and **basic input-output systems** (**BIOS**) for errors and s tores the result in a special memory location (**CMOS**).
    - Once the **POST** has successfully completed, the software loaded in **ROM** (sometimes called the **BIOS** or **firmware** [固件]) will begin to activate the computer's disk drives.


ROM PCI/ISA BIOS (2A69KGOD)
CMOS SETUP UTILITY
AWARD SOFTWARE, INC.
STANDARD CMOS SETUP
BIOS FEATURES SETUP
CHIPSET FEATURES SETUP
POWER MANAGEMENT SETUP
PNP/PCI CONFIGURATION
LOAD BIOS DEFAULTS
LOAD PERFORMANCE DEFAULTS
INTEGRATED PERIPHERALS
SUPERVISOR PASSWORD
USER PASSWORD
IDE HDD AUTO DETECTION
SAVE & EXIT SETUP
EXIT WITHOUT SAVING
Esc : Quit
↑ ↓ → ← : Select Item
F10 : Save & Exit Setup
(Shift) F2 : Change Color
Time, Date, Hard Disk Type...


CMOS Setup Utility - Copyright (C) 1984-2003 Award Software
Advanced BIOS Features
First Boot Device [Floppy]
Second Boot Device [CDROM]
Third Boot Device [HDD-0]
Password Check [Setup]
Full Screen LOGO Show [Enabled]
Item Help
Menu Level ▸
Select Boot Device Priority
[Floppy]
Boot from floppy
[LS120]
Boot from LS120
[HDD-0]
Boot from First HDD
[HDD-1]
Boot from second HDD
↑↓→←:Move Enter:Select +/-/PU/PD:Value F10:Save ESC:Exit F1:General Help
F5: Previous Values F6: Fail-Safe Defaults F7: Optimized Defaults

Before you use your OS

– The **bootstrap loader** is a small program that has a single function： **It loads the operating syste m into memory and allows it to begin operatio**

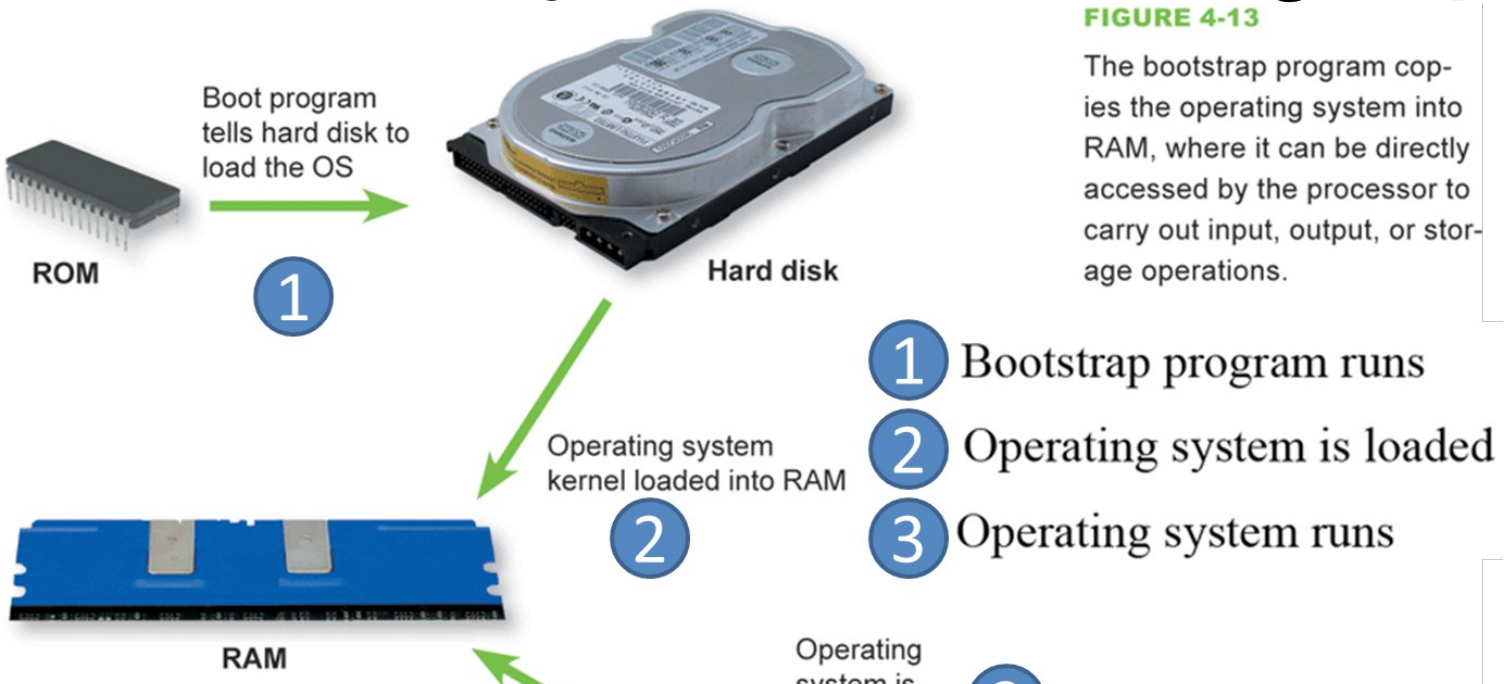
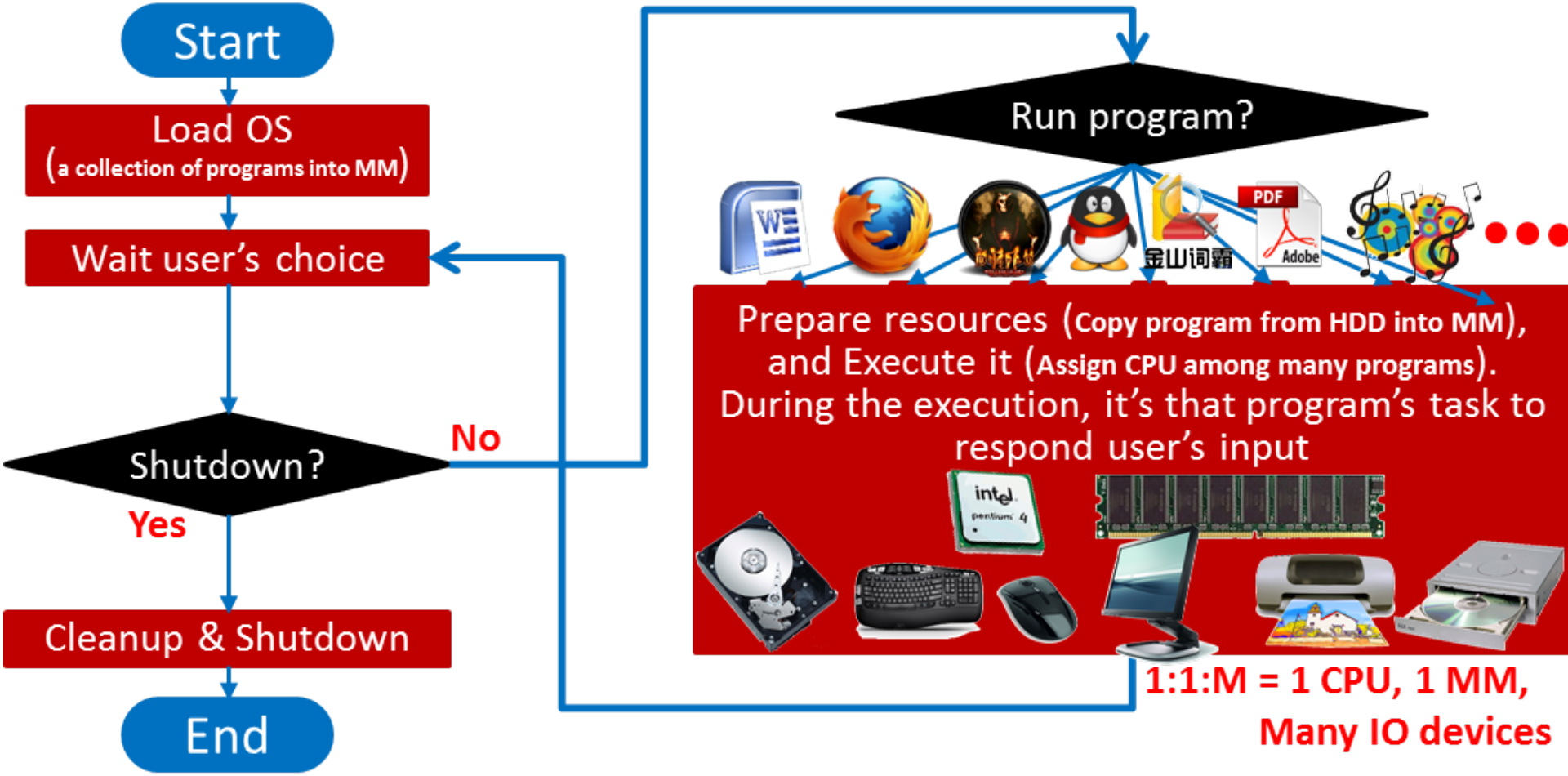


FIGURE 4-13

The bootstrap program copies the operating system into RAM, where it can be directly accessed by the processor to carry out input, output, or storage operations.

Generally its implementation consists of 2 parts stored separately in CMOS and

- Then it’ s OS!


Start
Load OS
(a collection of programs into MM)
Wait user’s choice
Shutdown?
No
Yes
Cleanup & Shutdown
End
Run program?
Prepare resources (Copy program from HDD into MM),
and Execute it (Assign CPU among many programs).
During the execution, it’s that program’s task to
respond user’s input
1:1:M = 1 CPU, 1 MM,
Many IO devices

Security risk is everywhere

- The BIOS starts any OS loader, even malware

- UEFI will only launch a verified OS loader – such as in Windows 8
- Malware cannot switch the boot loader

http://answers.microsoft.com/en-us/windows/forum/windows_7-security/uefi-secure-boot-in-windows-81/65d74e19-9572-4a91-85aa-57fa783f0759

Your try

- The UEFI Boot Manager controls the program flow across the 6 phases