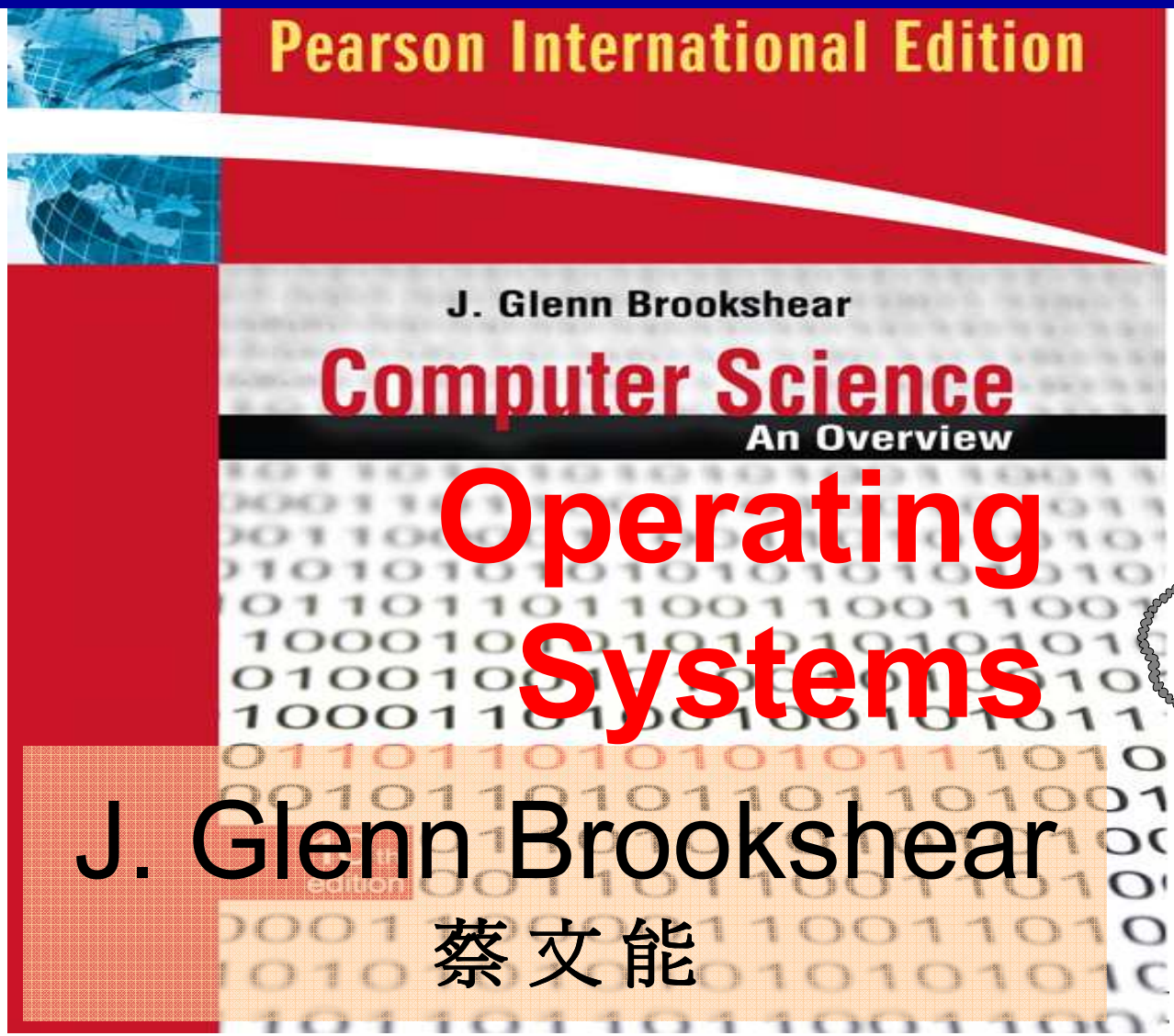
# Chapter 3




# Operating Systems

J. Glenn Brookshear
蔡文能

# Agenda


- **Review of the Computer Architecture**
  - **The central processing unit.**
  - **Instructions & The stored-program concept.**
  - **Program execution.**
- **The Computer Software**
  - **Application** Software
  - **System** Software
    - The Operating Systems : Kernel + Shell(s)
    - Utility Software
- The booting process (開機過程)
- Competition among Processes
  - **Critical** Sections and **Dead Lock**

# 傷腦筋的 頭字語 (Acronym)

- **ENIAC -- 1946/02/14**
  - Electronic Numerical Integrator And Computer (Calculator ?)

http://en.wikipedia.org/wiki/ENIAC

- **IBM**
  - I Believe Money
  - International Big Mouth (OOP 的 櫻櫻美代子)
  - International Business Machine
- **IDF**
  - I Don't Fly
  - I Don't Fight
  - I Do Fly , I Do Fight
- **CS**
  - Computer Science
  - Counter Strike ☺
- **OOP**
  - Object Oriented Programming
  - Office Of President

**http://www.oop.gov.tw**

- ICQ (I Seek You)

**TLA : Three Letter Acronym**

# Measuring Memory Capacity

Km = 千米=公里

- **Kilobyte:** $2^{10}$ bytes = 1024 bytes
  - Example: 3 KB = 3 times1024 bytes
  - Sometimes “kibi” rather than “kilo”
- **Megabyte:** $2^{20}$ bytes = 1,048,576 bytes
  - Example: 3 MB = 3 x 1,048,576 bytes
  - Sometimes “megi” rather than “mega”
- **Gigabyte:** $2^{30}$ bytes = 1,073,741,824 bytes = **$10^9$ bytes**
  - Example: 3 GB = 3 x 1,073,741,824 bytes
  - Sometimes “gigi” rather than “giga”

硬碟容量 1.5TB ?

通訊頻寬 20Gbps ?

- Tera = 1024 Giga = **$10^{12}$**
- Peta = 1024 Tera = **$10^{15}$**
- Exa = 1024 Peta = **$10^{18}$**

- Zeta = 1024 Exa = $10^{21}$
- Yotta = 1024 Zeta = $10^{24}$

http://en.wikipedia.org/wiki/Exa-

# $\mu s$ = micro second

## 760 *mm* Hg (Atmospheric pressure)

- ***d*** = **deci** = $10^{-1}$
- ***c*** = **centi** = $10^{-2}$
- ***m*** = **milli** = $10^{-3}$
- ***μ*** = **micro** = $10^{-6}$
- ***n*** = **nano** = $10^{-9}$
- ***p*** = **pico** = $10^{-12}$

- *f* = femto = $10^{-15}$
- *a* = atto = $10^{-18}$
- *z* = zepto = $10^{-21}$
- *y* = yocto = $10^{-24}$

*cm* = 公分=厘米

*nano* second 奈秒 = $10^{-9}$ 秒

*mm* = 毫米

*pico* second 披秒 = $10^{-12}$ 秒

*nm* = 奈米

http://en.wikipedia.org/wiki/Exa-

# The Frequencies of Various Wireless Media

Ultraviolet : 0.75P ~30PHz ; 10nm ~ 400nm
Infrared light : 1T~430THz; 0.7~300micrometers

*Frequency ($H_z$)*

$10^{16}$
$10^{15}$
$10^{14}$
$10^{13}$
$10^{12}$
$10^{11}$
$10^{10}$
$10^{9}$
$10^{8}$
$10^{7}$
$10^{6}$
$10^{5}$
$10^{4}$
$10^{3}$
$10^{2}$
$10^{1}$

X rays, gamma rays
Ultraviolet light
Visible light
Infrared light

紫外線

紅外線

Millimeter waves

Microwaves
UHF television
VHF television
VHF TV (high band)
FM radio
VHF TV (low band)
Short-wave radio
AM radio

毫米波

Very Low Frequency

微波

**極超高頻 SHF Super High Frequency**, 頻率範圍 3GHz~30GHz.

**超高頻 UHF Ultra High Frequency**, 頻率範圍 300MHz~3000MHz.

**極高頻 VHF Very High Frequency**, 頻率範圍 30MHz~300MHz.

**高頻HF** High Frequency, 頻率範圍 3MHz~30MHz.

# Electromagnetic Spectrum

waveLength * frequency = Light Speed = 299,792,458 m/second ($3*10^8$米/秒)

**Y= $\gamma$ =Gamma rays**
**HX = Hard X-Rays**
**SX = Soft X-Rays**
**EUV = Extreme UltraViolet**
**NUV = Near UltraViolet**
**NIR = Near Infrared**
**MIR = Mid Infrared**
**FIR = Far Infrared**
**EHF= Extremely High Freq.**
**SHF= Super High Freq.**
**UHF= Ultra High Freq.**
**VHF= Very High Freq**
**High / Medium / Low Freq.**
**VLF= Very Low Frequency**
**VF/ULF= Voice Frequency**
**SLF= Super Low Frequency**
**ELF= Extremely low freq.**

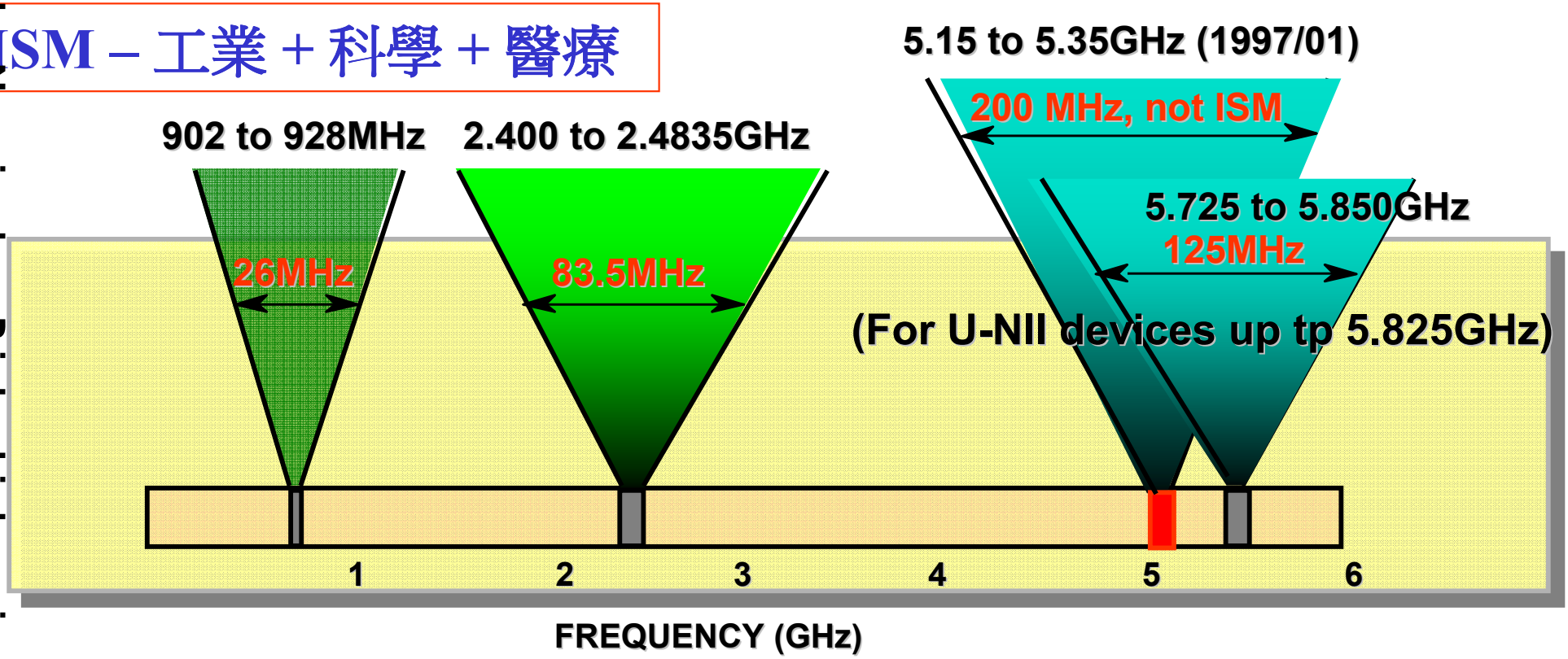

| CLASS | FREQUENCY | WAVELENGTH | ENERGY |
|---|---|---|---|
| Y | 300 EHz | 1 pm | 1.24 MeV |
| HX | 30 EHz | 10 pm | 124 keV |
| | 3 EHz | 100 pm | 12.4 keV |
| SX | 300 PHz | 1 nm | 1.24 keV |
| | 30 PHz | 10 nm | 124 eV |
| EUV | 3 PHz | 100 nm | 12.4 eV |
| NUV | 300 THz | 1 μm | 1.24 eV |
| NIR | 30 THz | 10 μm | 124 meV |
| MIR | 3 THz | 100 μm | 12.4 meV |
| FIR | 300 GHz | 1 mm | 1.24 meV |
| EHF | 30 GHz | 1 cm | 124 μeV |
| SHF | 3 GHz | 1 dm | 12.4 μeV |
| UHF | 300 MHz | 1 m | 1.24 μeV |
| VHF | 30 MHz | 10 m | 124 neV |
| HF | 3 MHz | 100 m | 12.4 neV |
| MF | 300 kHz | 1 km | 1.24 neV |
| LF | 30 kHz | 10 km | 124 peV |
| VLF | 3 kHz | 100 km | 12.4 peV |
| VF/ULF | 300 Hz | 1 Mm | 1.24 peV |
| SLF | 30 Hz | 10 Mm | 124 feV |
| ELF | 3 Hz | 100 Mm | 12.4 feV |

Source: http://en.wikipedia.org

# Industrial, Scientific and Medical (ISM) Bands

http://www.fcc.gov/Bureaus/Engineering_Technology/Orders/1997/fcc97005.pdf

ISM – 工業 + 科學 + 醫療




- UNLICENSED OPERATION GOVERNED BY FCC DOCUMENT 15.247, PART 15
- SPREAD SPECTRUM ALLOWED TO MINIMIZE INTERFERENCE
- 2.4GHz ISM BAND
  - More Bandwidth to Support Higher Data Rates and Number of Channels
  - Available Worldwide
  - Good Balance of Equipment Performance and Cost Compared with 5.725GHz Band
  - IEEE 802.11 Global WLAN Standard

**UNII : Unlicensed National Information Infrastructure**

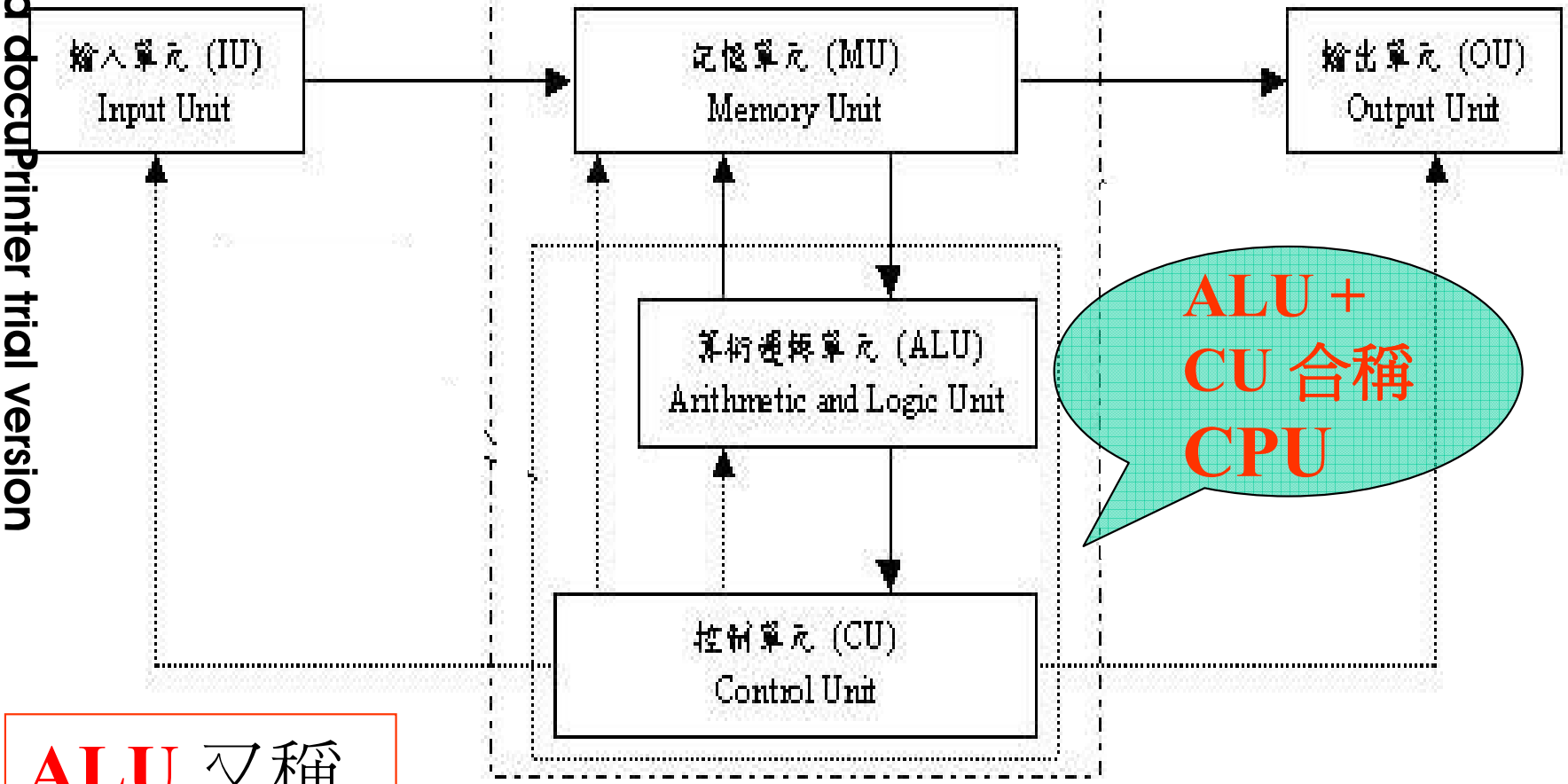
# 電腦硬體五大單元

## 計算 2+3=?

人會如何做?

電腦如何做?

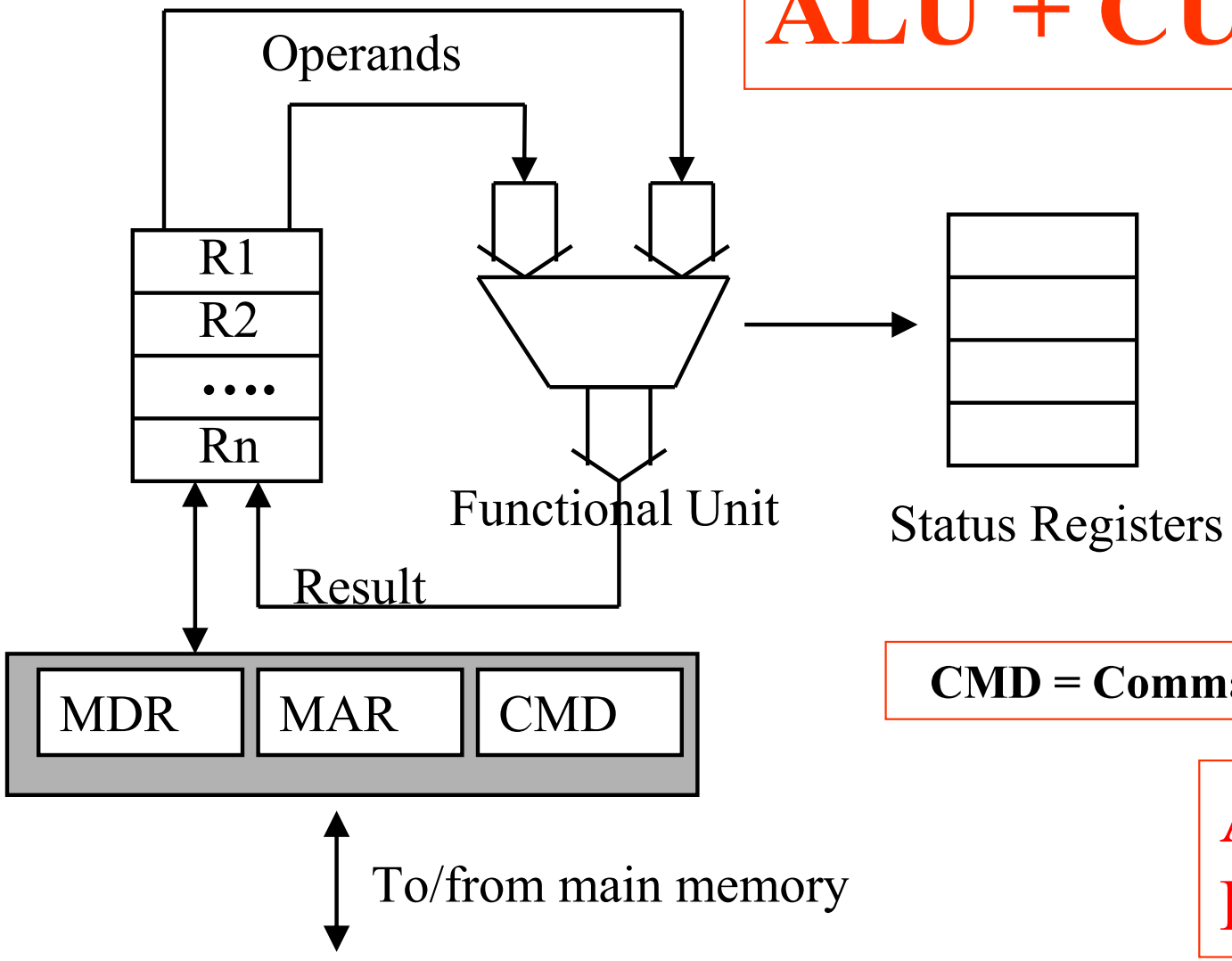
單晶片微電腦 = CPU + Memory




**ALU** 又稱 Data Path

# The ALU = Arithmetic & Logic Unit

**ALU + CU = CPU**



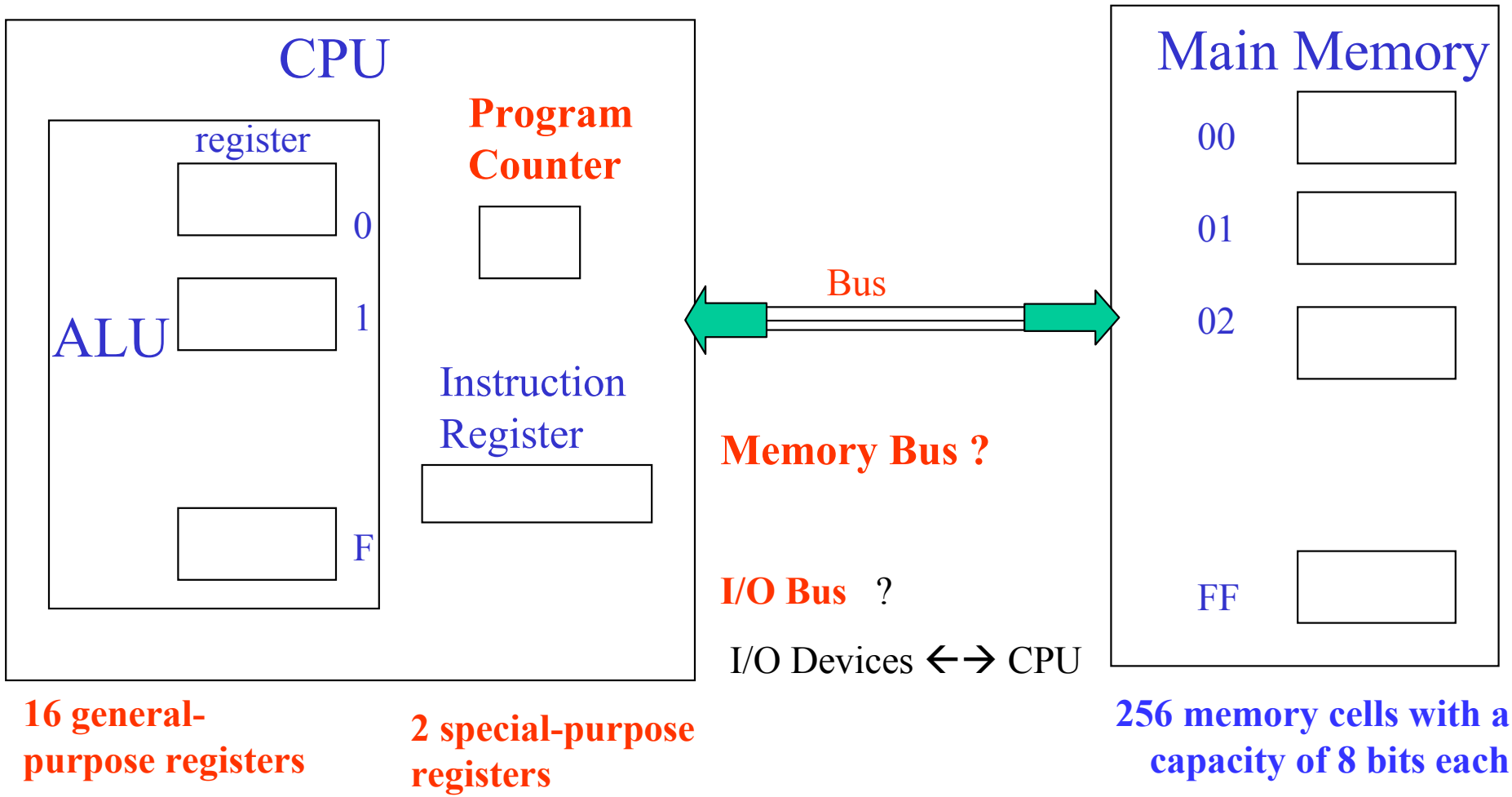

**CMD = Command OP code**

ALU 又稱
Data Path

**MDR = Memory Data Register**

**MAR = Memory Address Register**

# The architecture of a sample machine




**Program Counter** (**Instruction Pointer**) address of next instruction to be executed

**Instruction Register** hold instruction being executed

**Stack Pointer** (這假想電腦沒有; 現代CPU都有) **point to top of the STACK in memory**

# Machine Instrugctions 1/3

**Data transfer** Movement of data from one location to another

| | |
|---|---|
| LOAD | fill a register with contents of a memory cell |
| LOADI | fill a register with constant in the instruction Immediately |
| STORE | transfer contents of a register to a memory cell |
| Move | transfer cpntents of a register to another register |

**Arithmetic/Logic**

Arithmetic operations ADD, FADD

Logic operations OR, AND, XOR

ROTATE

**FLAGs: …condition code..**
Negative Zero V Carry
Sign Zero Overflow C

**Control** direct execution of program

JUMP direct control unit to execute an instruction other than the next one

| | |
|---|---|
| Unconditional | Skip to step 5 |
| Conditional | If resulting value is 0, then skip to step 5 |

電腦有左邊這些指令就夠用了

# Machine Instructions 2/3

Example for a conditional JUMP  Division

1- LOAD a register R1 with a value from memory
2- LOAD register R2 with another value from memory
3- If contents of R2 is zero, JUMP to step 6

4- Divide contents of R1 by contents of R2, result stored in R3
5- STORE the content of R3 into memory
6- STOP

Example: Avoid from dividing by zero using conditional JUMP

# Machine Instructions 3/3

## The Instructions of a sample machine

| Op-Code | Operand | Assembly Language |
|---|---|---|
| 1 | RXY | LOAD R, XY ; Load the Register R with data in memory XY |
| 2 | RXY | LOADI R, XY ; Load the constant XY into Register R |
| 3 | RXY | STORE R, XY ; Store the data in Register R into memory XY |
| 4 | 0RS | MOVE R, S ; copy R to S |
| 5 | RST | ADD R, S, T ; R = S + T |
| 6 | RST | FADD R, S, T ; floating Add |
| 7 | RST | OR R, S, T ; R = S or T |
| 8 | RST | AND R, S, T ; R = S and T |
| 9 | RST | XOR R, S, T ; R = S xor T |
| A | R0X | ROTR R, X ; Rotate the Register R to the Right X times |
| B | RXY | JUMP R, XY ; goto XY if [Register R] == [R0] |
| C | 000 | HALT |

How to do a **Subtraction**?

# Computer-Peripheral Communication protocols

電腦週邊設備

Parallel communications.
- Centronics
- Internal bus: rates measured in Mbps

Bit per second

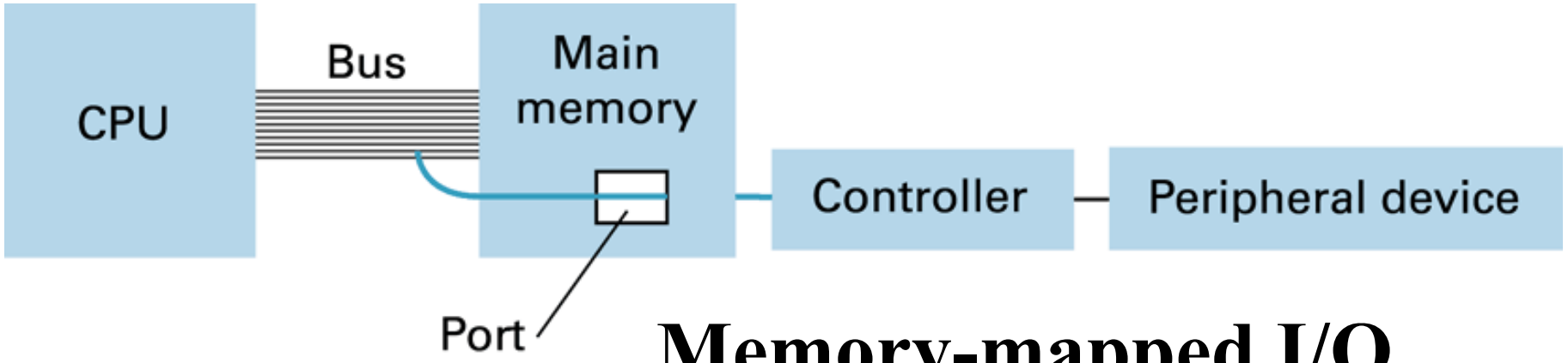
Serial communication
- RS232 - 300 bps ~ 115 Kbps
- RS422/485 , IEEE488
- USB ( Universal Serial Bus)
  - USB1.1 - 2Mbsp ~ 12Mbps
  - USB2.0 - 480Mbps
- IEEE 1394 (FireWire™) (400mbps)
- Telephone line: Kbps; Mbps (xDSL)
- Coaxial cable (第四台用的即是其一) – 10Base2, 10Base5
- Twisted Pair (use RJ45接頭Plugs) – 100BaseT, 1000BaseT, 10000BaseT
  - Cat 3 (10Mbps), Cat 5 (100Mpbs), Cat 6 (1Gbps), Cat 7 (10Gbps)
- Optic fiber: near Gbps

2S 1P ?

2S 1P 4USB ?

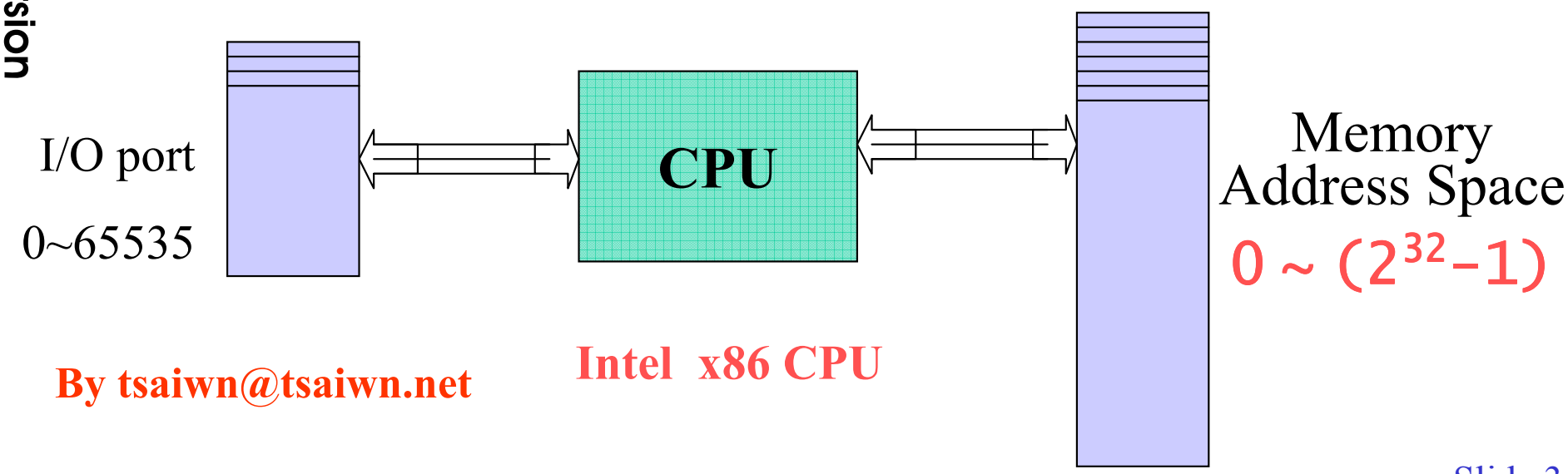
tsaiwn@tsaiwn.net

# Figure 2.14: A conceptual representation of memory-mapped I/O




## Memory-mapped I/O

Some CPU has no I/O Ports




By tsaiwn@tsaiwn.net

Intel x86 CPU

# 計算機架構 (Architecture)

- CISC Complex Instruction Set Computer
  - 例如： Intel x86, DEC VAX
- RISC Reduced Instruction Set Computer
  - 例如： IBM RISC6000, SUN SPARK, SGI MIPS
- Parallel Processing
  - Pipeline -- 提高 throughput
  - Multiprocessor machine多處理單元
    - SISD, MIMD, SIMD (page 107, text book)

Computer **Architecture** : **What** ?

Computer Organization : How ?

# Stored-Program Concept

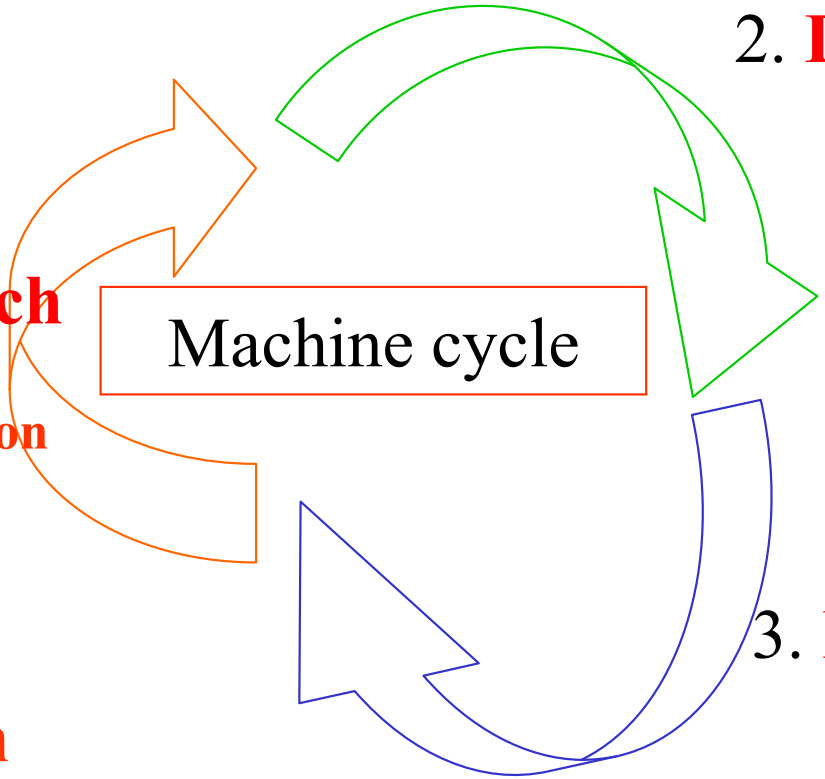
- In early computing, the program is built into the control unit as a part of the machine. The user rewires the control unit to adapt different programs.
- Program (instructions) stored in memory instead of being built into the control unit as part of the machine
- A computer's program can be changed merely by changing the contents of the computer's memory instead of rewiring the control unit
- A machine recognizes a bit pattern as representing a specific instruction
- Instruction consists of two parts

| Op-code (operation code) | operand(s) field(s) |
|---|---|

- STORE operands would be
  - ✓Register containing data to be stored
  - ✓Address of memory cell to receive data

# Program Execution(The machine cycle )

JUMP Instruction **B258**$_{16}$




1.**Fetch**

**Retrieve next instruction from memory (as per program counter) and then**

**increment program counter**

2. **Decode**

**Decode bit pattern in instruction register**

3. **Execute**

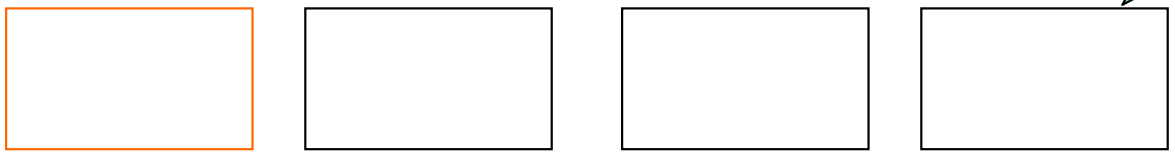
**Perform action requested by instruction in the instruction register**

# The composition of an instruction

## Instruction Format

半個 Byte
稱爲
1個 Nibble

Instruction consists of 4 hex digits (2 bytes)



**Op-code** **Operand Fields**

$1347_{16}$ LOAD register 3 with contents of the memory cell at address $47_{16}$

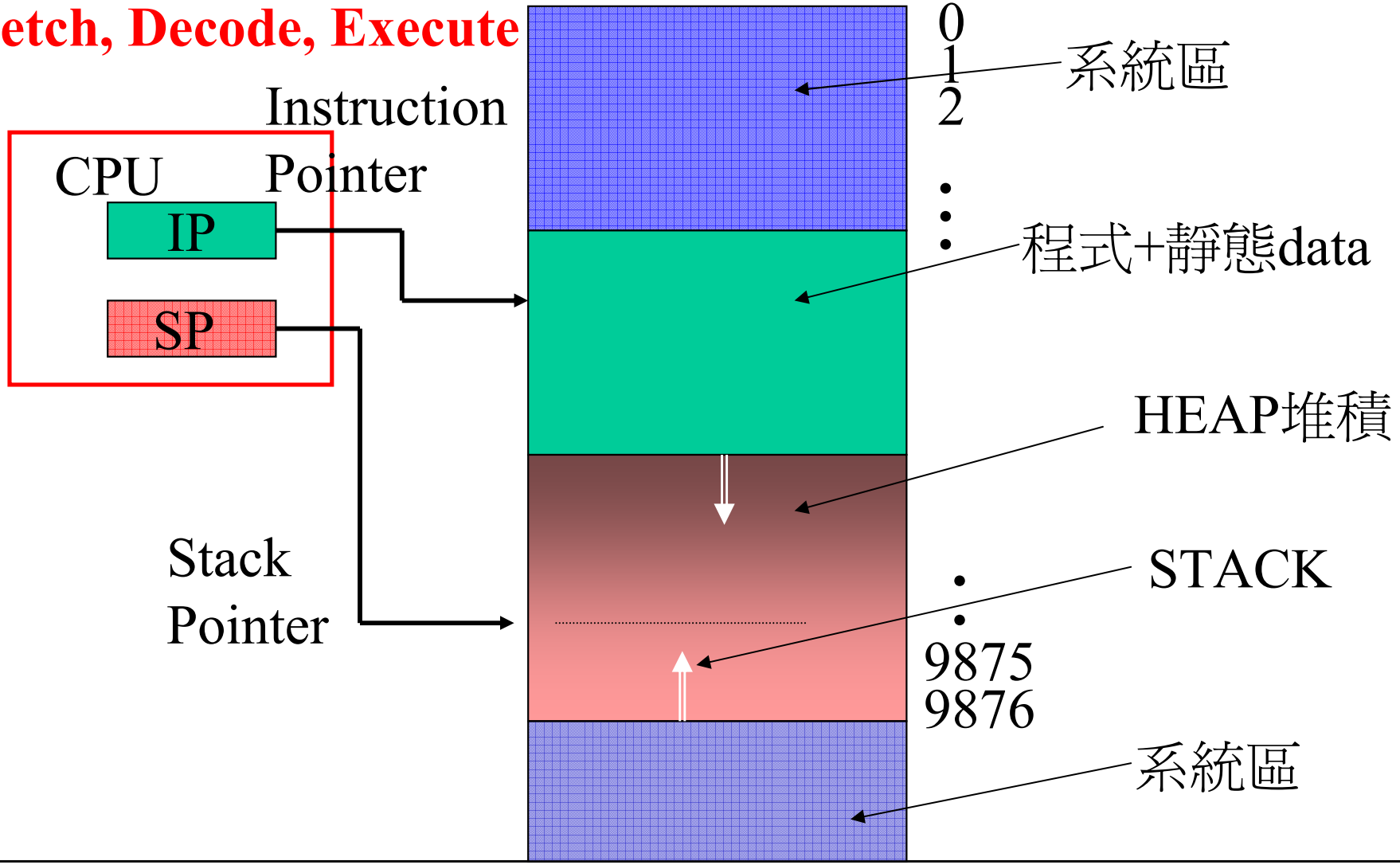
Textual representation might be “LOAD R3,47”

$B258_{16}$

JUMP to instruction at address $58_{16}$ if **contents of register 2 is the same as register 0**

# 電腦如何運作? How it works?

- **Auto 變數就是沒寫 static 的 Local 變數**



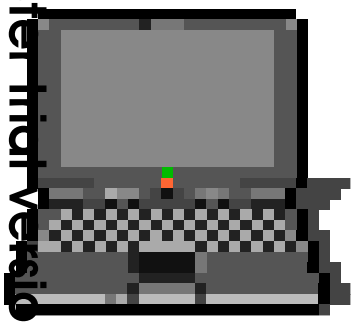

**Auto 變數佔用STACK區memory**

# Software(軟體) --電腦的靈魂

- **Operating Systems (OS)**
  - Kernel (核心, 直接控制硬體的一些程式)
    - OS components
  - Shell (又稱 Command Interpreter 命令解譯器)
- **Programming languages**
  - **Algorithms** + Data Structures = **Programs**
  - **Compiler vs. Interpreter**
- Software engineering Tools (例如 Rational **Rose**)
- **D**ata **B**ase **M**anagement **S**ystem (**DBMS**)

# Web Applications

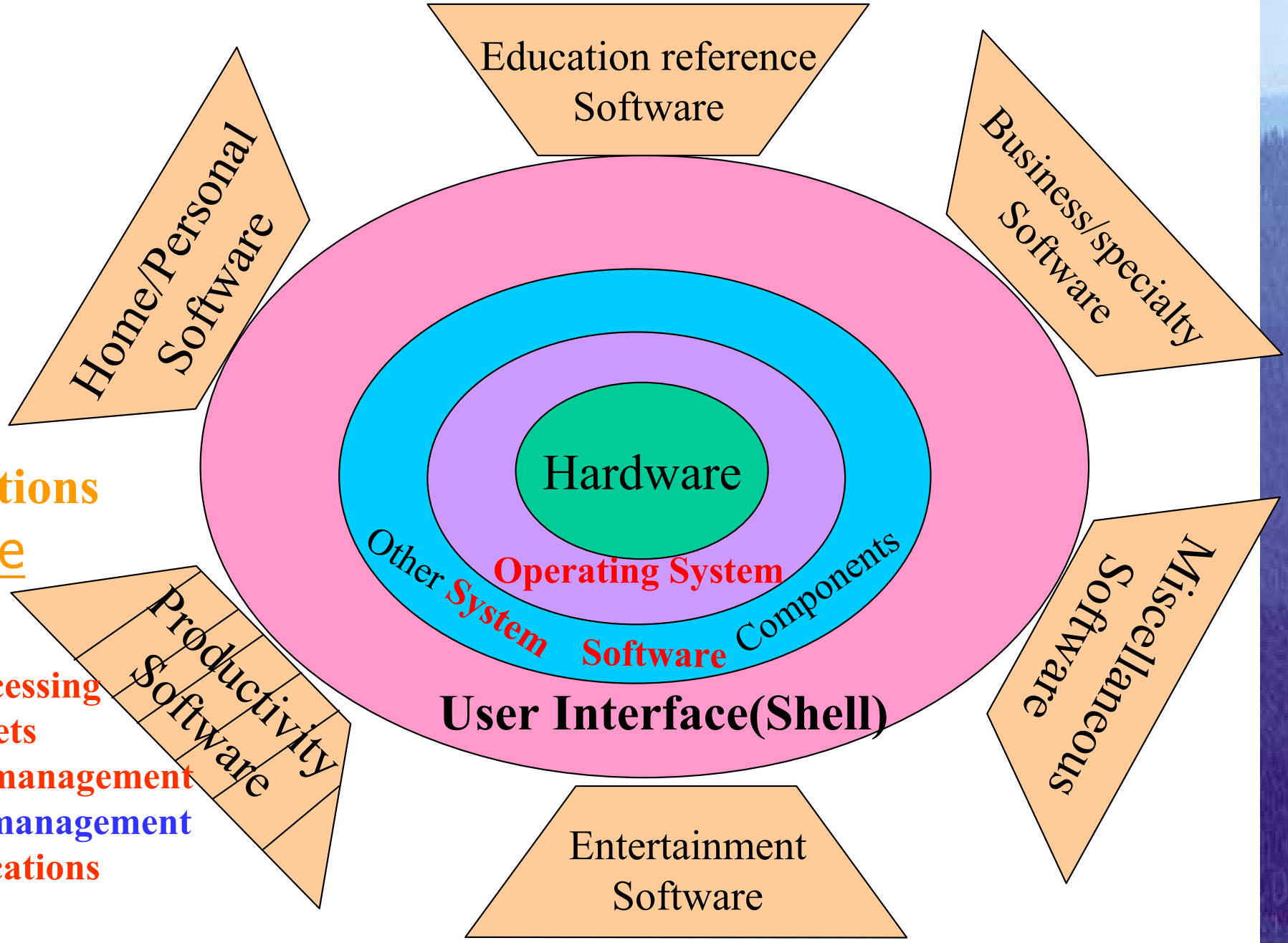
# Computer Applications





# 硬體軟體蔥皮圖






**Applications Software**

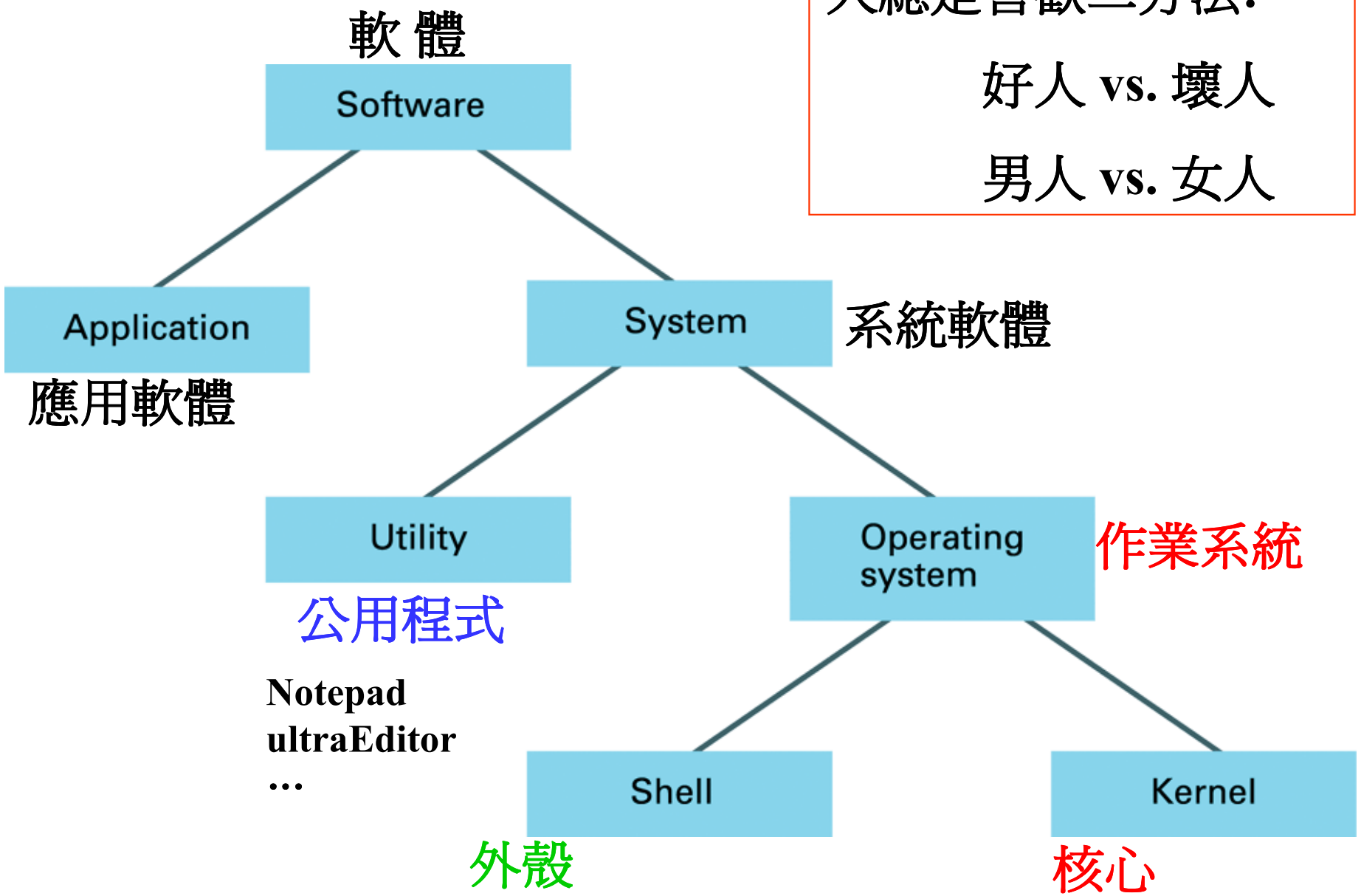
**Word Processing**
**Spreadsheets**
**Financial management**
**Database management**
**Communications**
**Suites**
**Browsers**

# Figure 3.3: Software classification

人總是喜歡二分法:

好人 vs. 壞人

男人 vs. 女人



# Types of software

- Applications software (應用軟體)
  - Performs tasks specific to the machine's utilization.
  - Generally transportable (即容易換到別的系統)
- System Software (系統軟體)
  - Performs tasks common to computer systems in general
  - Startup Software (Bootstrap Loader)
    - ✓**POST**- Power On Self Test
    - ✓**BIOS**- Basic Input/Output System (Subroutines)
  - Operating Systems vary based on the hardware they're used on
  - Utility Software – 公用程式 / 工具軟體

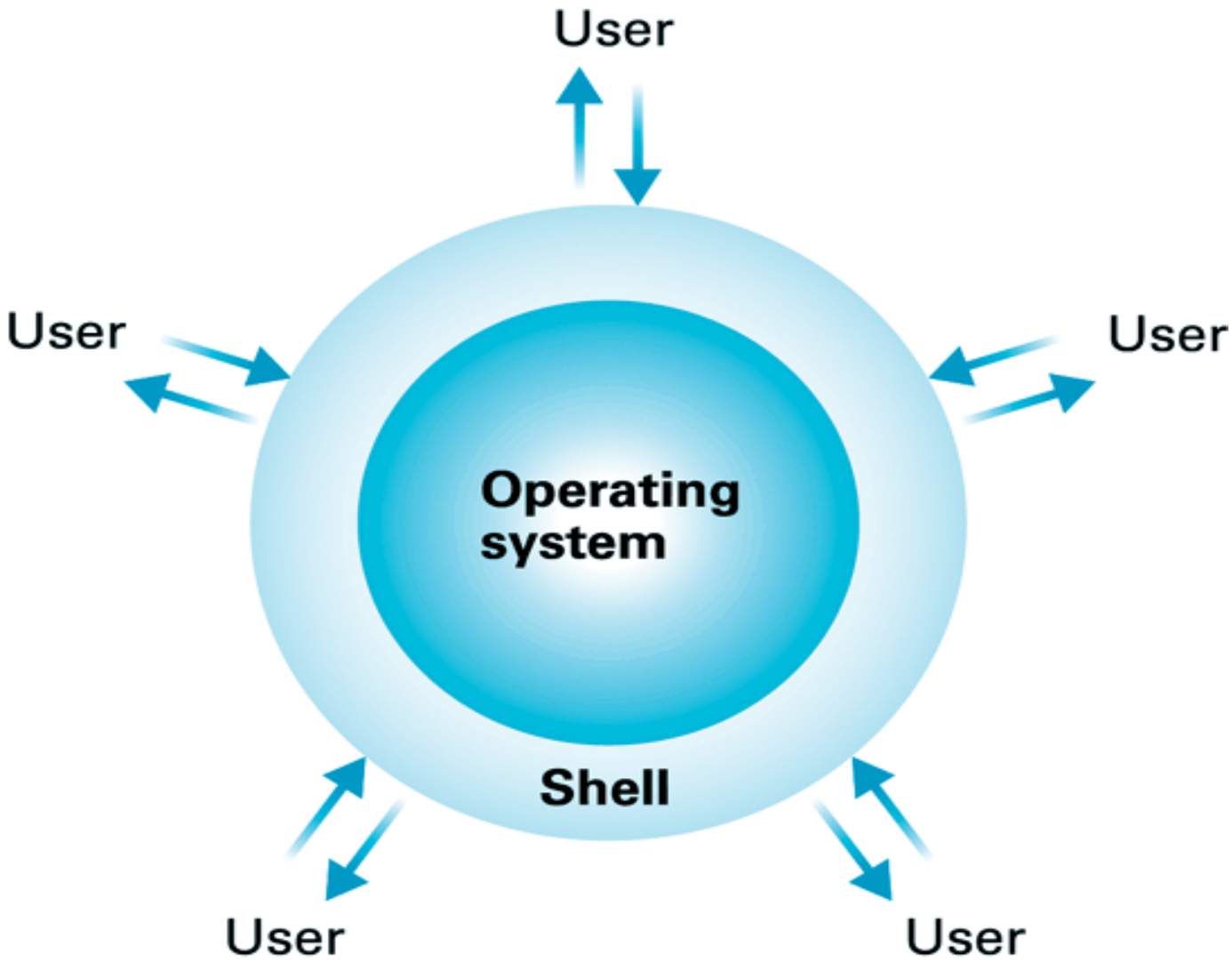
# Types of System Software

- ✓ Operating System (**OS**)
  - ➢ **Shell ( also known as Command Interpreter )**
  - ➢ **Kernel**
- ✓ **Utility** software
  - – Kind of System Software
  - – providing fundamental activities, yet not included with OS
  - – "extend" the **OS**

❖ **What is the difference between them?**

- – Distinction between applications and **utilities** is often vague (不明確的, 不清楚的)
- – Distinction between **OS** and **utilities** is also vague

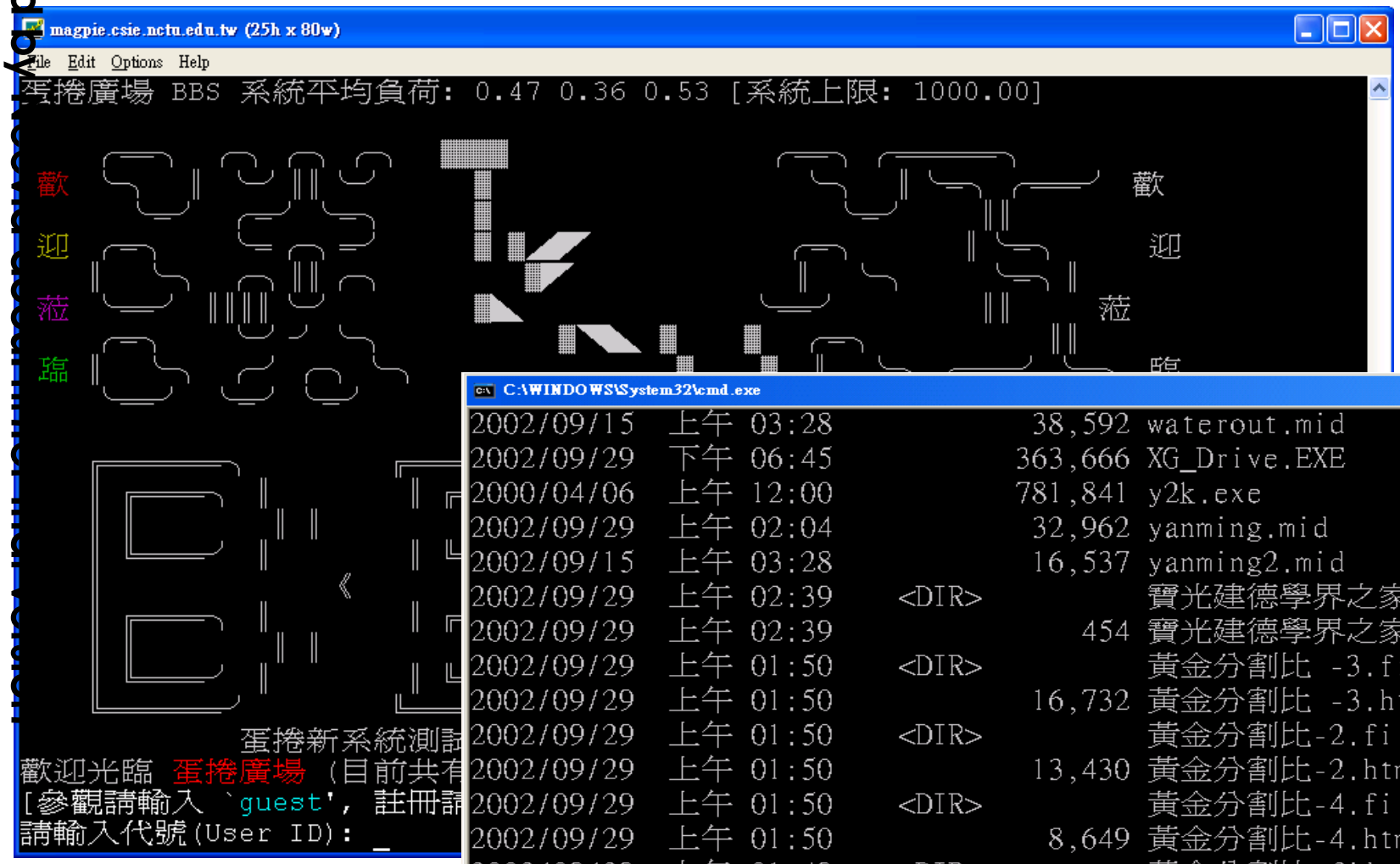
# Figure 3.4: The **shell** as an interface between users and the operating system



# More About Shell

- Also known as **Command Interpreter**
- Types of shell
  - Command driven
  - Menu driven (restricted shell)
  - **GUI (Graphical User Interface)**

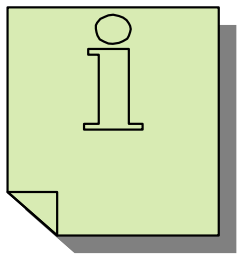
# Command Shell examples



# Graphical Shell

## GUI – (pronounced "goo-ee")

- The Graphical User Interface – features of the GUI…
- Users work with on-screen pictures called icons and with menus rather than keyed-in.



# What does Shell can do?

- Read Command from the user and take some action(s)
  - ➢Internal commands (and the Aliases)
  - ➢External commands
    - **Current directory ? ( . )**
      - **Unix vs. DOS/Windows ?**
    - **Path**
      - **Command path**
      - **Data path**

# OS can have many different Shells

- Defines interface between OS and users
  - Windows GUI
  - UNIX command line (Command driven)
  - UNIX users can choose among a variety of shells
    - sh is the "Borne shell"
    - csh is the "C shell" (因語法像 C; by Berkeley Univ.)
    - tcsh is an enhanced "C shell"
    - **ksh is the "Korn shell"**
    - bash is "Borne Again Shell"
  - **Shell programming (Batch file/Script file)**

# The Operating System Kernel

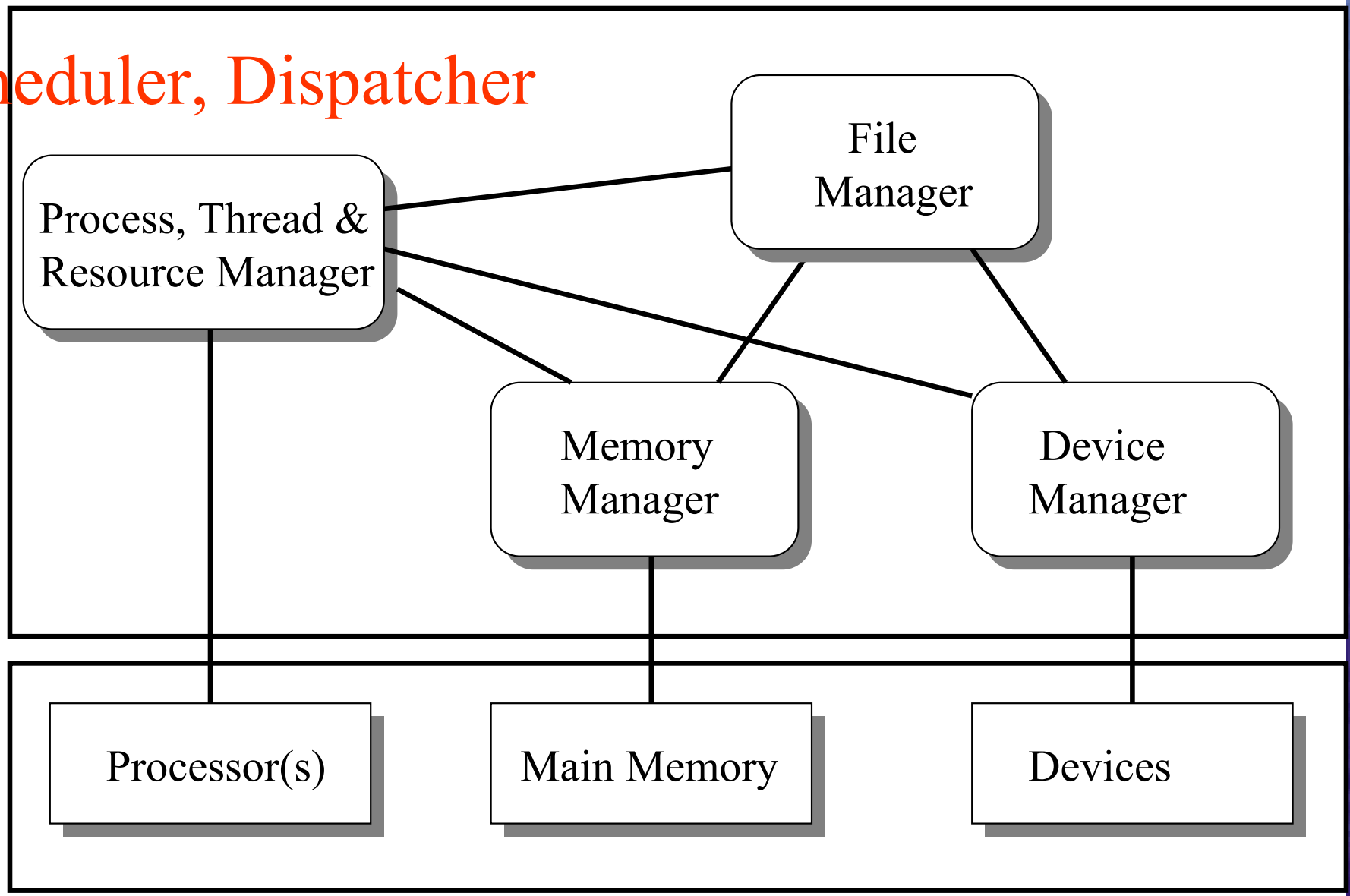
The internal part of the OS is often called the Kernel .

- Resident in memory, running in privileged mode
- System calls offer general purpose **services**
- Controls and mediates access to **hardware**
- Schedules / allocates system **resources**:
  - CPU, memory, disk, devices, etc.
- **Event driven**:
  - Responds to user requests for service (system calls)
  - Attends interrupts and exceptions
  - **Context switch** at quantum time expiration

# OS Kernel Components (1/2)

- Kernel Components
  1) **File Manager** -- manages mass storage
  2) **Memory Manager** -- manages main memory
  3) **Device Drivers** -- communicate with peripherals
  4) **Scheduler**-- manage processes 排班
  5) **Dispatcher**-- manage processes 指揮
- The **`trap`** instruction is used to switch from **user** to **supervisor** mode, entering the OS

# OS Kernel Components (2/2)



# File Manager : OS Component 1/5

- Maintains information about the **files** that are available on the system
- Where files are located in mass storage, their size and type and their protections, what part of mass storage is available
- Files usually allowed to be grouped in ***directories*** or ***folders***. Allows hierarchical organization.

# Memory Manager : OS Component 2/5

- This unit is responsible for coordinating the use of the machine's **main memory**.
- It decides what area of memory is to be allocated for a program and it's data
- It allocates and deallocates memory **for different programs** and always knows what areas are free.

# Device Drivers : OS Component 3/5

- **Software to communicate with peripheral devices or controllers**
- Each driver is unique
- Translates general requests into specific steps for that device

- Drive != Driver (車子 != 司機)

不等於

# Scheduler : OS Component 4/5

- Maintains a record of processes that are present, adds new processes, removes completed processes
  - memory area(s) assigned
  - priority
  - state of readiness to execute (ready/wait)

Scheduler == 排班者

# Dispatcher : OS Component 5/5

- Ensures that processes that are ready to run are actually executed
- Time is divided into small segments (e.g., 50 ms) called a ***time slice***. (時間片斷)
- When the **time slice** is over, the dispatcher allows scheduler to update process state for each process, then selects the next process (from ready queue) to run.

**Dispatcher** == 指揮官; 調度官

# Process Management

- **A *process* is a program in execution.** A process needs certain resources, including CPU time, memory, files, and I/O devices, to accomplish its task. (正在跑的程式就叫 ***process 行程***)
- The operating system is responsible, through Scheduler and Dispatcher, for the following activities in connection with process management.
  - Process creation and deletion.
  - process suspension and resumption.
  - Provision of mechanisms for:
    - process **synchronization**
    - process **communication**

# More about the OS

- OS History
- OS kernel
- Types of OS
  - Batch vs. Interactive
  - Multi-Programming
  - Time Sharing
  - Real Time Operating System (RTOS)
- Other Topics regarding OS
- OS Loading
  - The Booting process (Bootstrapping)

# Brief History of Operating Systems

- 1940's -- First Computers (NO OS in it)
- 1950's -- Batch Processing
- 1960's -- Multiprogramming / Timesharing
- 1970's -- Minicomputers & Microprocessors
- Late 1970's, 1980's -- Networking, Distributed Systems, Parallel Systems
- 1990's and Beyond -- PC's, WWW, Mobile Systems, Real-time System

http://en.wikipedia.org/wiki/History_of_operating_systems

# Early Computing History

- In the 1940s and 1950s, all computers were **personal computers** in the sense that a user would sign up to use the machine and then take over the whole machine for that period.
  - **ENIAC 1946/02/14 於賓州大學**
- The early 1960s were dominated by **batch systems** in which a user would **submit** a job on **punched cards** and wait, usually hours, before any printed **output** appeared on a **printer**.

http://en.wikipedia.org/wiki/ENIAC

# MULTICS project in MIT

- To get around this unproductive environment, the **concept** of **timesharing** was **invented** by Dartmouth College and M.I.T. (1969)
- The M.I.T system CTSS (Compatible Time Sharing System) was an enormous success.
- M.I.T., Bell Labs, and General Electric created a **second** generation **timesharing** system named **MULTICS** (**MULT**iplexed **I**nformation and **C**omputing **S**ervice). (1964-1969)多工式資訊與計算服務

**MULT**i-user **I**nteractive **C**omputing **S**ystem ?

http://en.wikipedia.org/wiki/MULTICS

http://en.wikipedia.org/wiki/Unix

# Early UNIX History (1/4)

- At Bell Labs, **Ken Thompson** decided to write a stripped down version of MULTICS for the very small **PDP-7** minicomputer which he called **UNICS**.
- **Dennis Ritchie**, also at Bell Labs, joined **Thompson** in further developments of what was now called **UNIX**.
- Together they ported the system to the larger and very popular **PDP-11**/20 and PDP-11/45 minicomputers.

**PDP-7 → PDP-11　　7-11** ?

http://en.wikipedia.org/wiki/Unix

# Early UNIX History (2/4)

- **Thompson** also tried to rewrite the operating system in high level language of his own design which he called **B**., which is a modified version from **BCPL**.
- The **B** language lacked many features and **Ritchie** decided to design a successor to B which he called **C**.
- **They then rewrote UNIX in the C programming language to aid in portability. (C + Assembly)**

BCPL: **B**asic **C**omputer **P**rogramming **L**anguage

BCPL → B → C → C++ → Java → C#

# Early UNIX History (3/4)

- In 1974, **Ritchie** and **Thompson** published a paper about UNIX and received the prestigious ACM Turing Award.
- This publication stimulated many universities to request a copies of UNIX.
- Since Bell Labs, part of AT&T, was not allowed to be in the computer business, it licensed UNIX to universities.
- Also, at that time, the **PDP-11** series was the workhorse of most computer science departments.
- Result: UNIX was a hit on campus.

http://en.wikipedia.org/wiki/Unix

# Early UNIX History (4/4)

- In Version 6, the source code of UNIX was **8200 lines of C** and **900 lines of assembler**.
- The first **portable** version arrived with Version 7 which had **18,800 lines of C** and **2100 lines of assembly instruction**.
- By the 1980s the use of UNIX was widespread with many vendors selling their own versions based on Version 7.

The C Programming Language,
by **Brian Kernighan** and **Dennis Ritchie**
(the first edition of which is sometimes referred to as K&R)

http://en.wikipedia.org/wiki/The_C_Programming_Language_(book)

# The BSD UNIX

One of the many universities that had received license for UNIX was the **University of California at Berkeley**.

Aided by many government grants, Berkeley released an improved version named **1BSD** (First **Berkeley Software Distribution**)

**In subsequent, versions Berkley added many new features including a new visual editor (vi) and a new shell (csh). (原來的是 Borne shell)**

**csh**：通稱 C-Shell, 因其語法很像 **C** 語言

**Shell script**：就是 BATCH file (批次檔)

# System-V vs. BSD-4

- Because of these and other enhancements, many companies based their **UNIX** on **Berkeley's** version (BSD) as opposed to AT&T's so-called **System V**.
- By the late 1980s, two different and somewhat incompatible versions of UNIX were in widespread use:
  - BSD 4.4
  - System V release 4. (SVR4)

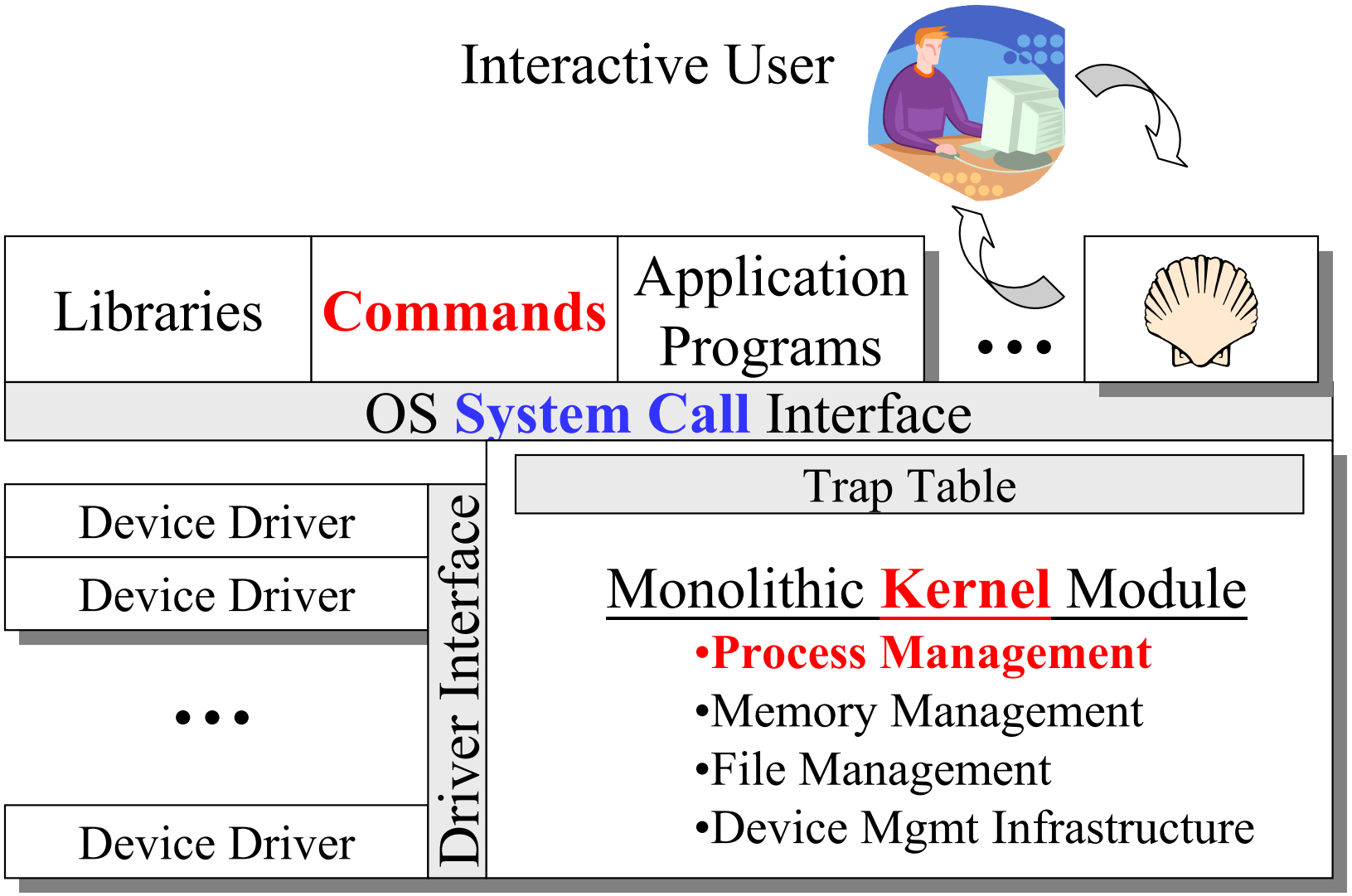
http://en.wikipedia.org/wiki/Berkeley_Software_Distribution

http://en.wikipedia.org/wiki/UNIX_System_V

# Services provided by OS

- Program creation
- Program execution
- Process management
- Provide System calls
- Access to I/O devices
- Controlled access to files
- Error detection and response
  - hardware errors (e.g., memory error, device failures)
  - software errors (e.g., overflow, out of memory boundary)
- Accounting
  - collect usage statistics, monitor performance

# The OS Architecture



# System Calls (1/2)

- System calls provide the interface between a running program and the operating system.
  - Generally available as assembly-language instructions.
  - Languages defined to replace assembly language for systems programming allow system calls to be made directly by high level languages(e.g., C, C++)
- Three general methods are used to pass parameters between a running program and the operating system.
  - Pass parameters in *registers*.
  - Store the parameters in a table in memory, and the table address is passed as a parameter in a register.
  - *Push* (store) the parameters onto the *stack* by the program, and *pop* off the stack by operating system.
- Types of System Calls
  - File management
  - Device management
  - Information maintenance
  - Process control
  - Communications

http://en.wikipedia.org/wiki/System_call

# System Calls (2/2)

- On Unix, Unix-like and other POSIX-compatible Operating Systems, popular system calls are open, read, write, close, exit, wait, signal, fork, exec, and kill.
- How many system calls in OS ?
  - The **first Unix** has **47** system calls
  - Current **Linux kernel** has **319** system calls
  - Current **FreeBSD kernel** has almost **500** system calls

http://en.wikipedia.org/wiki/System_call

# UNIX Standards : IEEE1003 (1/3)

- In addition to the AT&T Unix and Berkeley BSD, every vendor added its own nonstandard enhancements.
- In an attempt to unify the troops, the IEEE Standards Board undertook the **POSIX** Project (POS stands for Portable Operating System) and IX to make it UNIX like. And the result is the IEEE 1003.
- IEEE 1003.1 emerged as a common ground standard.
- IEEE 1003.1 **is the intersection of System V and BSD. (A feature had to be on both to be included in the standard**)
- IEEE 1003.2 is about the Shell and the Utility programs

# UNIX Standards : IEEE 1003(2/3)

- The **POSIX** standard defined a set of library procedures and **systems calls** that all compliant UNIX systems.
- It appeared that the split between System V and BSD had been somewhat dealt with.
- As of 2009 POSIX:2008 or IEEE Std 1003.1-2008 represents the current version. A free online copy is available here:

  **http://www.opengroup.org/onlinepubs/9699919799/**

# UNIX Standards : IEEE1003 (3/3)

Unfortunately, a funny thing happened on the way back form the standards meeting.

A group of vendors led by IBM, DEC, Hewlett-Packard, and others formed the **OSF** (Open Software Foundation) to standardize an enhanced version of UNIX in an attempt to derail AT&T's efforts to regain control of UNIX. (註: DEC 已經倒掉)

AT&T, Sun, UNISYS, Data General, and other companies countered and formed UI (UNIX International) based on System V.

SUN OS (史丹弗大學; SUN: Stanford University Network)

- BSD based for version 4.x and before
- SVR4 based for version 5.x (**Solaris** system)

**2010/01/27 SUN 已正式被 Oracle 併購@ US$7.4billion**

**Oracle – 美商甲古文公司 – DBMS leader**

# The Open Group (was OSF)

- **The Open Group** is an industry consortium to set vendor- and technology-neutral open standards for computing infrastructure.
- **The Open Group** was formed when X/Open merged with the Open Software Foundation (OSF) in 1996.
- **The Open Group** is most famous as the certifying body for the UNIX trademark, in the past the group was best known for its publication of the Single UNIX Specification paper, which **extends** the **POSIX** standards and is the official definition of UNIX.

http://www.opensoftwarefoundry.org/

http://en.wikipedia.org/wiki/The_Open_Group

# The Unix International (UI)

- Unix International or UI was an association created in 1988 to promote open standards.
- UI was formed by AT&T, Sun, UNISYS, Data General, and some other companies. The Unix standard promoted by UI is based on AT&T System V.
- In May 1993, the major members of both UI and OSF announced the Common Open Software Environment (COSE) initiative. This was followed by the merging of **UI** and **OSF** into a "**new OSF**" in March 1994, which in turn merged with X/Open in 1996, forming **The Open Group**.

http://en.wikipedia.org/wiki/Unix_International

# Major Unix Flavors

- **First** Edition: Bell Labs, 1969
- BSD1.0: UC, Berkeley, 1977 -- BSD4.6
- System V: 1983 -- SVR1, SVR2, SVR3, SVR4
- **POSIX** standard
- Solaris (Sun OS 5.x) (有 PC 版)
- AIX (IBM)
- Linux (open source); GNU/Linux
- FreeBSD (open source)

# UNIX Like Systems

- In a new trend, UNIX like operating systems began to appear.
- MINIX, by **Andrew Tanenbaum**, used a microkernel design with only **1600 lines of C** and **800 lines of assembler** in its first version.
- **GNU** project started by Richard M. Stallman(RMS) at **1984/01/05**. (announced at **1983/09/27**)
- In 1991, a Finnish student named **Linus Torvalds** released another **UNIX** clone named **Linux** version **0.01**. (芬蘭大二學生)

http://en.wikipedia.org/wiki/Richard_Stallman

# GNU is Not Unix

- **GNU** is a project started by **Richard M. Stallman** (RMS) to write a completely free implementation of Unix available. (coding begun at 1984/01/05)
- **GNU** stands for "**G**NU is **N**ot **U**nix", which is **recursively** defining itself.
- Most of Unix has been rewritten by him and his friends.
- Many other software packages have been released for free.
- My favorite linux **distribution** has over 13,000 packages.

**http://www.gnu.org/gnu/thegnuproject.html**

# **Richard** M. **Stallman** (MIT Professor)

Father
of
GNU

# Free software according to RMS

- Free software comes with four freedoms
  - The freedom to **run** the software, for any purpose
  - The freedom to **study** how the program works, and adapt it to your needs
  - The freedom to **redistribute** copies so you can help your neighbor
  - The freedom to **improve** the program, and release your improvements to the public, so that the whole community benefits

四大自由

RMS -- Richard M.Stallman.

http://en.wikipedia.org/wiki/Richard_Stallman

# Linux operating system

- Linux is a **monolithic** design. (**9,300** lines of C and **950 instructions of assembly code in ver. 0.01**)
- Linux quickly grew in size and functionality.
- Linux Version 1.0, shipped in 1994, contained about **165,000 lines of code**.
- Version 2 in 1996 contained about **470,000 lines of C** and **8000 lines of assembly code**.
- Linux is released under the GNU Public License (**GPL**), which, very basically means that anyone can copy and change it.

**Linux 2.7**

**http://kerneltrap.org/forum/linux/kernel/**

# Linux Distributions

- Linux itself is free. It is aggregated with installation and management tools, and many other software packages, and made available for a small fee by various vendors on CD.
- These aggregates are known as distributions.
- Some common distributions are
  - Red Hat -- SuSE -- Mandrake
  - **Debian** -- **Slackware** -- **fedora (by RedHat)**
  - **Ubuntu** -- **XUbuntu**
  - **Knoppix (first Linux Live CD)**
- Differences
  - Locations of files (configuration, binaries, etc.)
  - GUI
  - Security, efficiency, etc.
- DRBL (Diskless Remote Boot in Linux)

**http://en.wikipedia.org/wiki/Linux_distribution**

# Other Unix-like systems

# Types of OS

- **Batch** vs. **Interactive** System
  - In early days (beyond PC era), computers were extremely expensive. To speed up processing, operators batched together jobs with similar needs and ran them through the computer as a group. The OS is simple that needs to only automatically transfer control from one job to the next.
- Multi-Programming System (next slide)
- **Time Sharing System**
- Real Time Operating System (RTOS)
- Multiprocessor System
- Distributed System, **Clustered** System

# Multiprogramming

記憶體內同時有多個程式

- Goal: keep CPU busy
- **Fact: I/O times are large**
- When one program is waiting for I/O to complete, run another program
- **=> Multiple programs resident in memory**
- Scheduling:
  - non-preemptive
  - Preemptive ➔ **TimeSharing**

http://en.wikipedia.org/wiki/THE_multiprogramming_system

# Time sharing

- Goal: allow access to multiple users at the same time
- **Fact: People's response time is large**
  - **in most cases users entered bursts of information followed by long pause** (thinking?)
- Schedule the programs fast + process switch
  - the "state" of each user and their programs should be kept in the machine, and then switched between quickly.
- Scheduling: preemptive

Time sharing = Multiprogramming + preemptive

http://en.wikipedia.org/wiki/Time-sharing

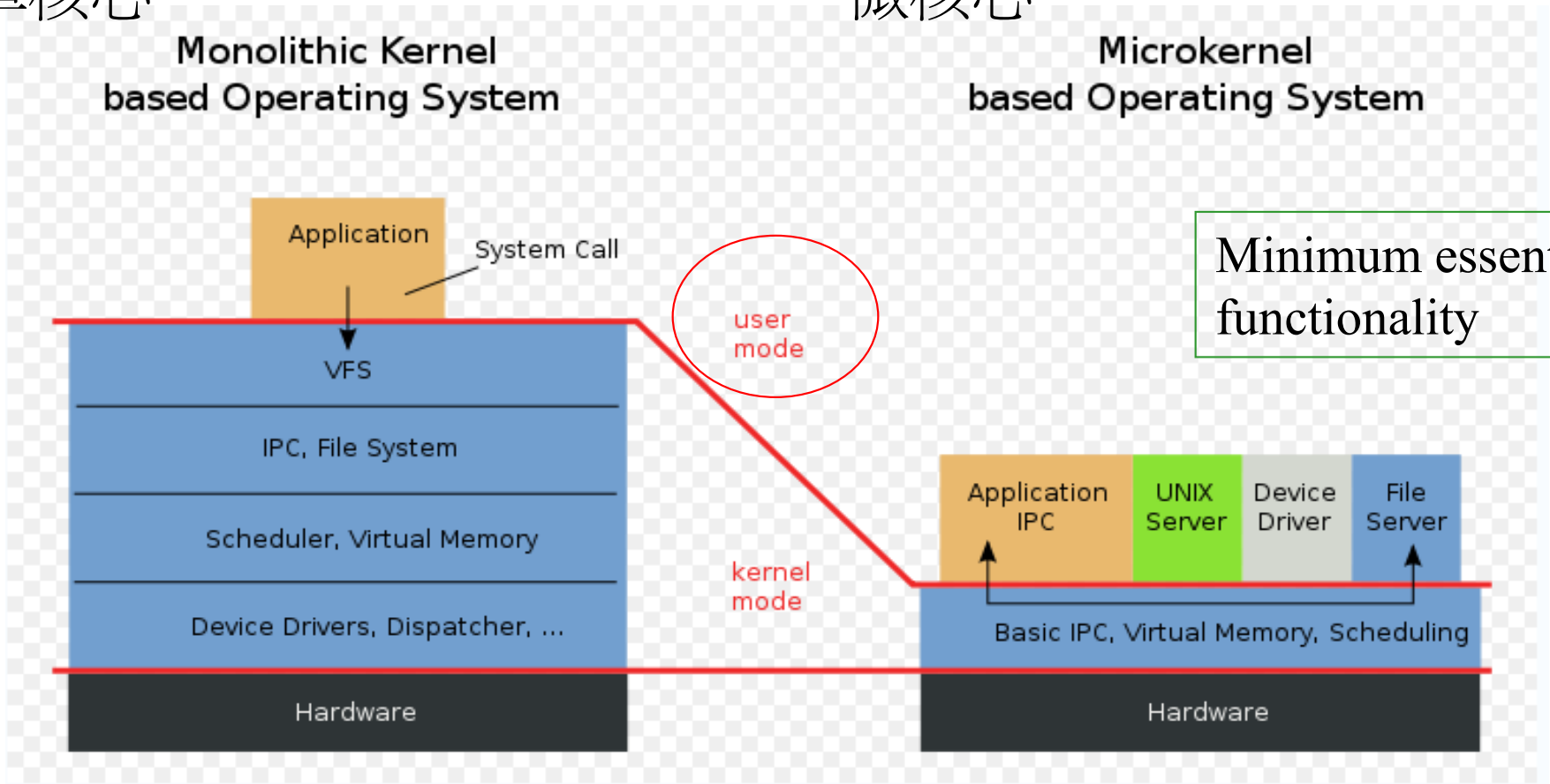
# Other Topics regarding OS

- Microkernel architecture
- Process state transition
- Processor **modes** give OS privilege
  - **User** mode vs. ***Supervisor*** mode
- Trap instruction makes true OS possible
- Context switch, process switch
- OS Loading
  - How does the System start ? (**Bootstrapping**)
  - **Cold** start vs. **Warm** start

http://en.wikipedia.org/wiki/Context_switch

# Monolithic Kernel vs. Microkernel

單核心 全核心 微核心




Minimum essential functionality

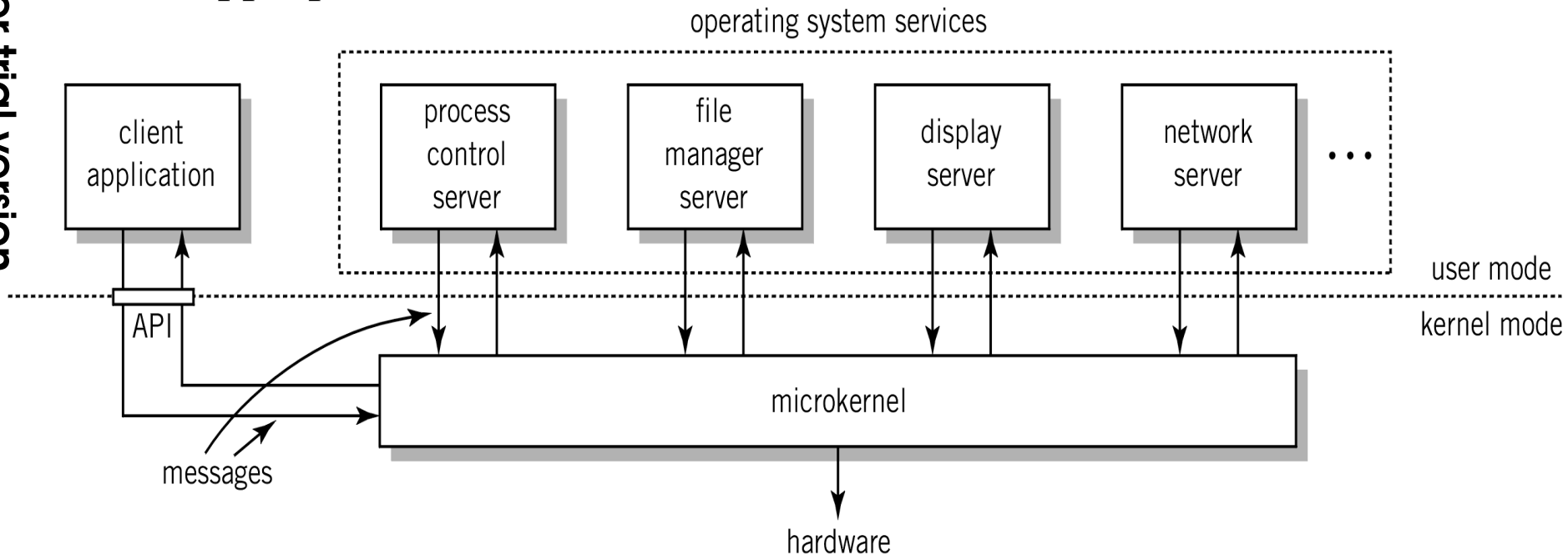
http://en.wikipedia.org/wiki/Microkernel

Windows XP, Linux are **Monolithic Kernel**

# Microkernel

CMU Mach, Apple **OS X**, GNU Hurd

- Minimum essential functionality
- Client-Server system on same system
  - Microkernel servers are essentially **daemon** programs like any others, except that the kernel grants some of them privileges to interact with parts of physical memory
  - **Clients request services from microkernel which passes message onto appropriate server**




Nano kernel, Pico kernel

# Processor Modes

- Mode bit: ***Supervisor*** or ***User*** mode
- **Supervisor** mode
  - Can execute all machine instructions
  - Can reference all memory locations
- **User** mode
  - Can only execute a subset of instructions
  - Can only reference a subset of memory locations

**Supervisor** Mode is designed for OS

# Trap instruction

Intel x86:
int 0 ~ int 255

- **Trap** is like a function call
  - aka hardware **interrupt**
  - mode is set to *supervisor*
  - address of function is looked up from a table **(interrupt** vector)
  - the function body is executed
  - The function body is aka Interrupt Service Routine (ISR) or Interrupt Handler
- Direct invocation of the function is not permitted.

若 CPU 沒提供 Trap instruction
則真正的 OS 寫不出來!
**(especially hardware trap)**

# OS Loading

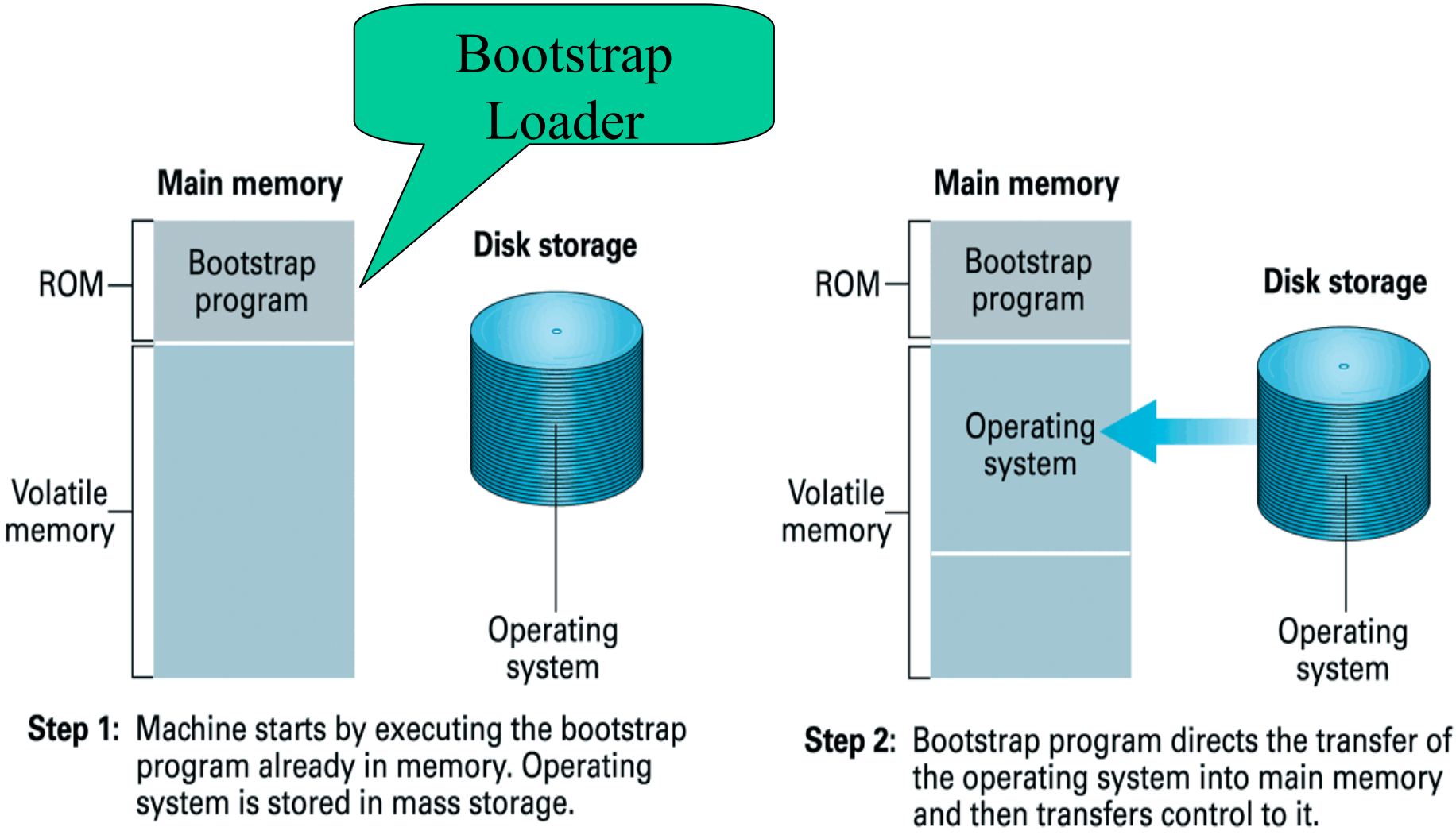
## How does OS loads into the first place?

- There's something called a *bootstrap **loader***, which is stored in ROM and brings the necessary pieces of OS (***bootstrap***) from the disks' **boot sector**, which know how to load the OS further, and transfers control to it.
- Then all the other parts of OS and device drivers are loaded.
- This self-sustaining process that will finally load the full OS into the computer and transfer control to the OS is called "**bootstrapping**". (to pull oneself up by one's bootstraps)

Where is the *bootstrap **loader*** before the ROM invented?

http://en.wikipedia.org/wiki/Bootstrapping_(computing)

# Figure 3.5: The booting process




http://en.wikipedia.org/wiki/Bootstrapping

# Start up the computer (1/2)

- When the computer is started, the control unit branches to a fixed memory location; e.g. initial PC value hardwired. (e.g., CS:IP = 0xffff:0000 in Intel_based computer.) (依照 Intel CPU 規定 **ffff:0** 等於 **0xffff0** 也就是 1M 倒數16byte處)
- The fixed location is a ROM address that contains a **POST** (Power On Self Test) program + a small ***bootstrap loader*** + **some *Input/Output* Routines . (in BIOS ROM)**
- The **POST** process is a series of tests conducted on the computer's main memory, input/output devices, disk drives, and so on, ... (測試電腦硬體有沒問題).
- **Bootstrapping** process starts after successful **POST** process.

BIOS = Basic Input Output System

Intel CPU: CS:IP 0xffff:0x0000 ➔ 0xffff0

http://en.wikipedia.org/wiki/Bootstrapping_(computing)

# Start up the computer (2/2)

- The **bootstrap loader** may be comprehensive enough to load the nucleus of the OS (**kernel**); **Otherwise**, it loads a **loader program** that does so. The **loader** program is usually called ***bootstrap***. (注意***bootstrap***必須是與 OS kernel 在同一儲存體才叫***bootstrap*** ;類似大馬靴的 ***bootstrap*** )
- Once bootstrap phase is done, any program can be run by loading it in memory and loading its initial address in the PC **(fetch-decode-exec** algorithm)

BIOS = Basic Input Output System

Intel CPU: CS:IP 0xffff:0x0000 ➔ 0xffff0

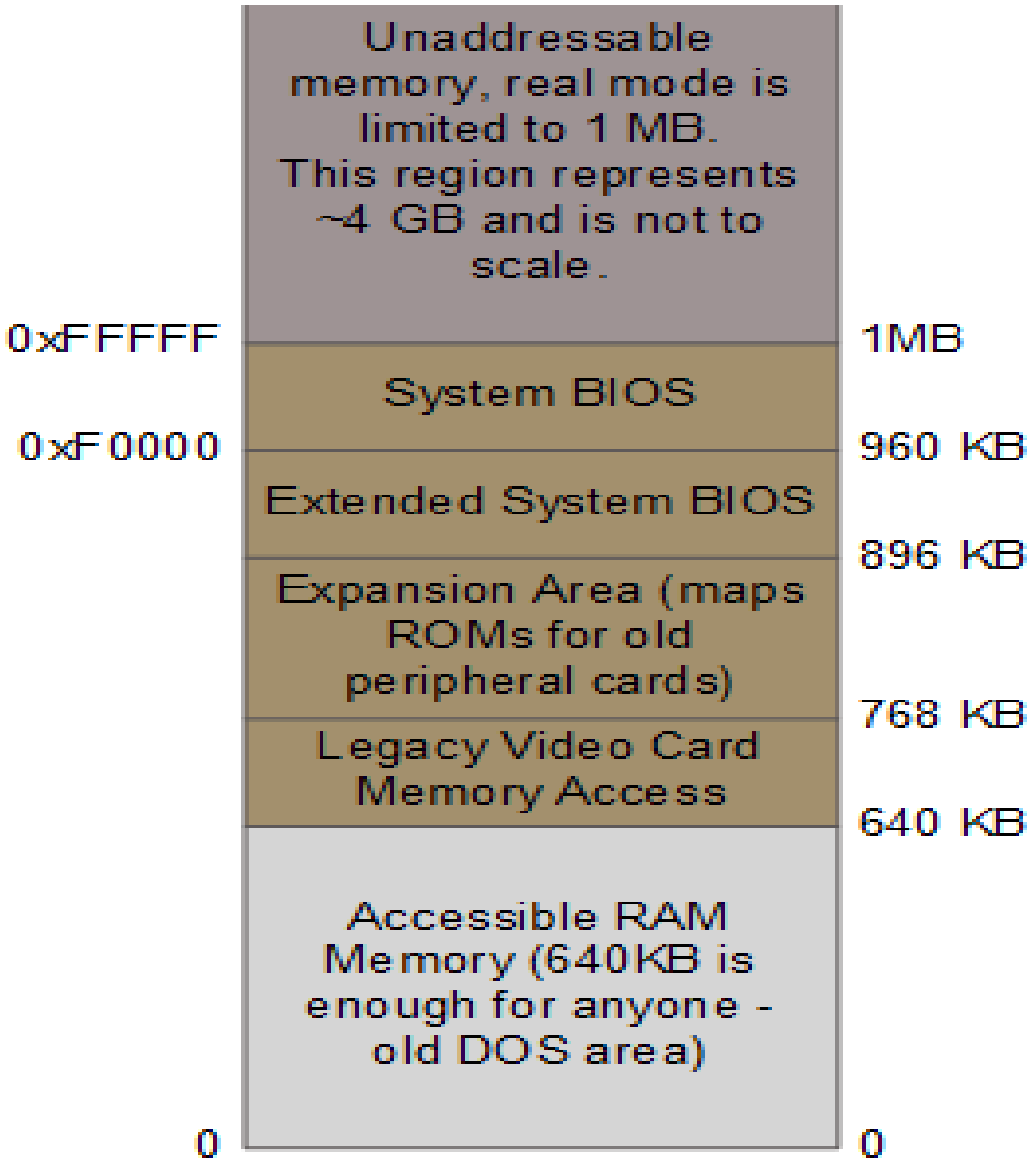
http://en.wikipedia.org/wiki/Bootstrapping_(computing)

# POST – Power On Self Test

- Power-On Self-Test (**POST**) is the common term for a computer, router or printer‘s pre-boot sequence. Including a series of tests conducted on the computer’s main memory, input/output devices, disk drives, and so on, ... (測試電腦硬體有沒問題)
- **Bootstrapping** process starts after successful **POST** process.
- Original IBM PC POST beep code
  - 1 short beep - Normal POST -- system is OK
  - 2 short beeps - POST error -- error code shown on screen
  - 1 long, 2 short beeps -- Display adapter problem (MDA, CGA)
  - 1 long, 3 short beeps -- Enhanced Graphics Adapter (EGA)
  - …
- POST beep codes on CompTIA A+ Hardware Core exam
  - One long, two short beeps -- Video card failure
  - Long continuous beep tone -- Memory failure
  - …

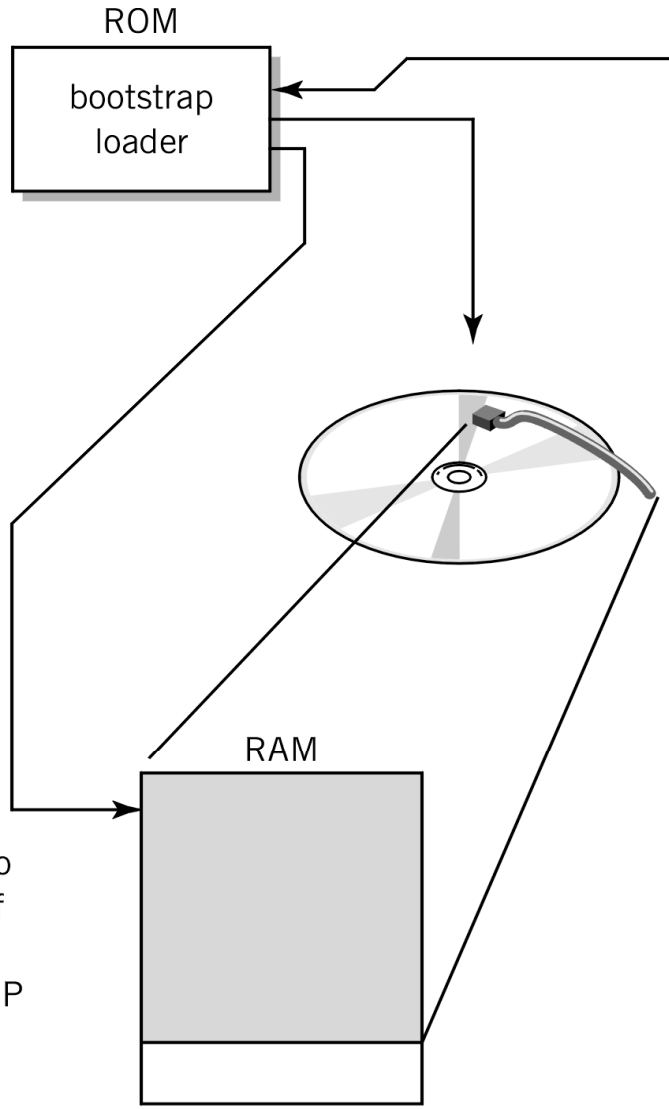
http://en.wikipedia.org/wiki/Power-on_self-test

# Memory layout on Intel-based PC



# Bootstrapping

- **Cold boot** vs. **warm boot** (does not **re**test the system)



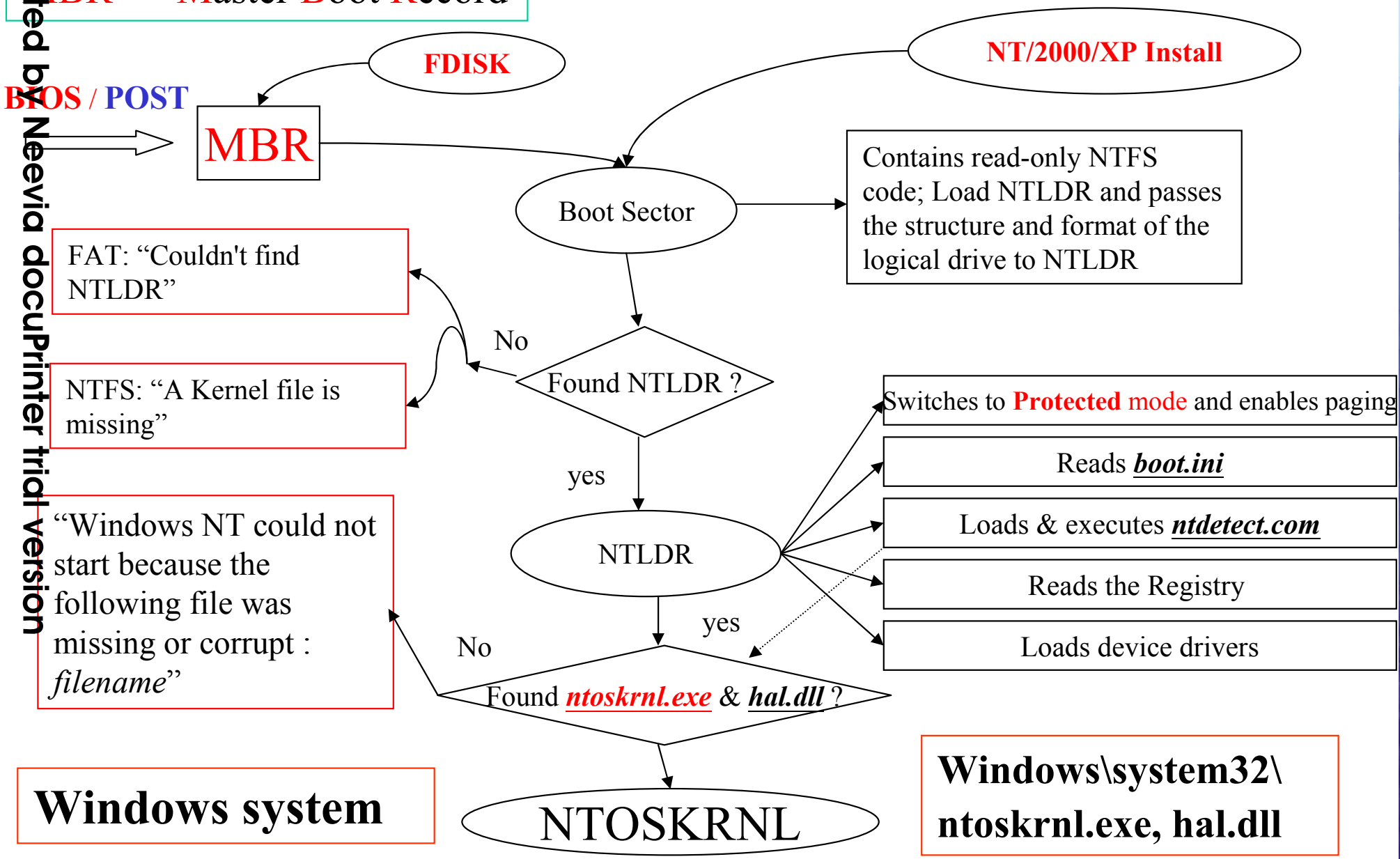

e.g., JUMP 0:7c00

# Bootstrapping Microsoft OS (DOS/Windows)

- **Two** step process when booting from Floppy:
  - a simple **bootstrap loader in ROM** fetches a more complex **boot program** from the **boot sector** in diskette, which in turn loads the **kernel**
- **Three** step process when booting from Hard Disk:
  - a simple **bootstrap loader in ROM (BIOS ROM)** fetches a pre-boot program in the MBR (Master Boot Record), and the Pre-boot program then find the bootable partition and load a the **boot program** from the **boot sector** in that partition (分割區), which in turn loads the **kernel**.

# Bootstrapping flow Diagram

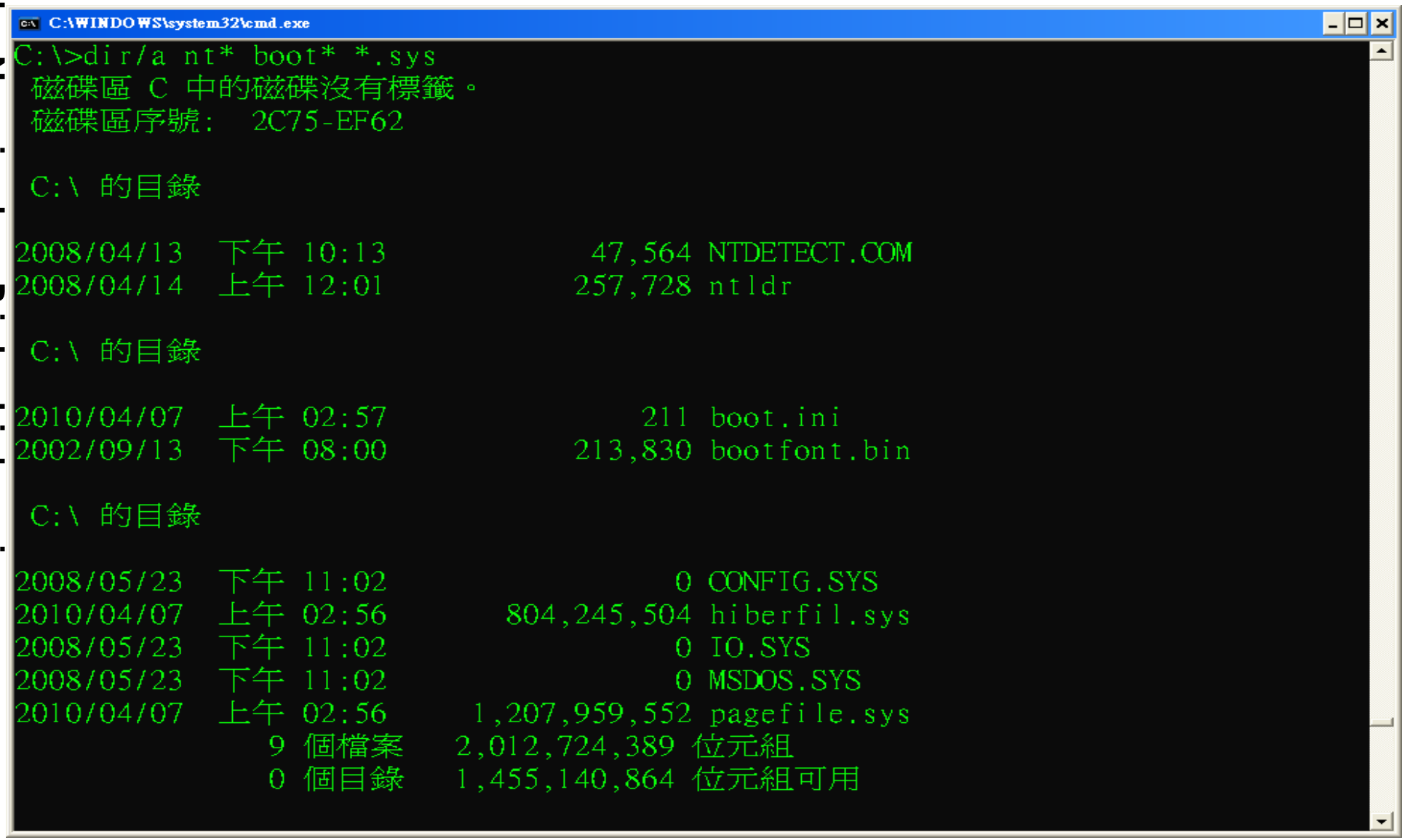
MBR = Master Boot Record




BIOS = Basic Input Output System

POST = Power On Self Test

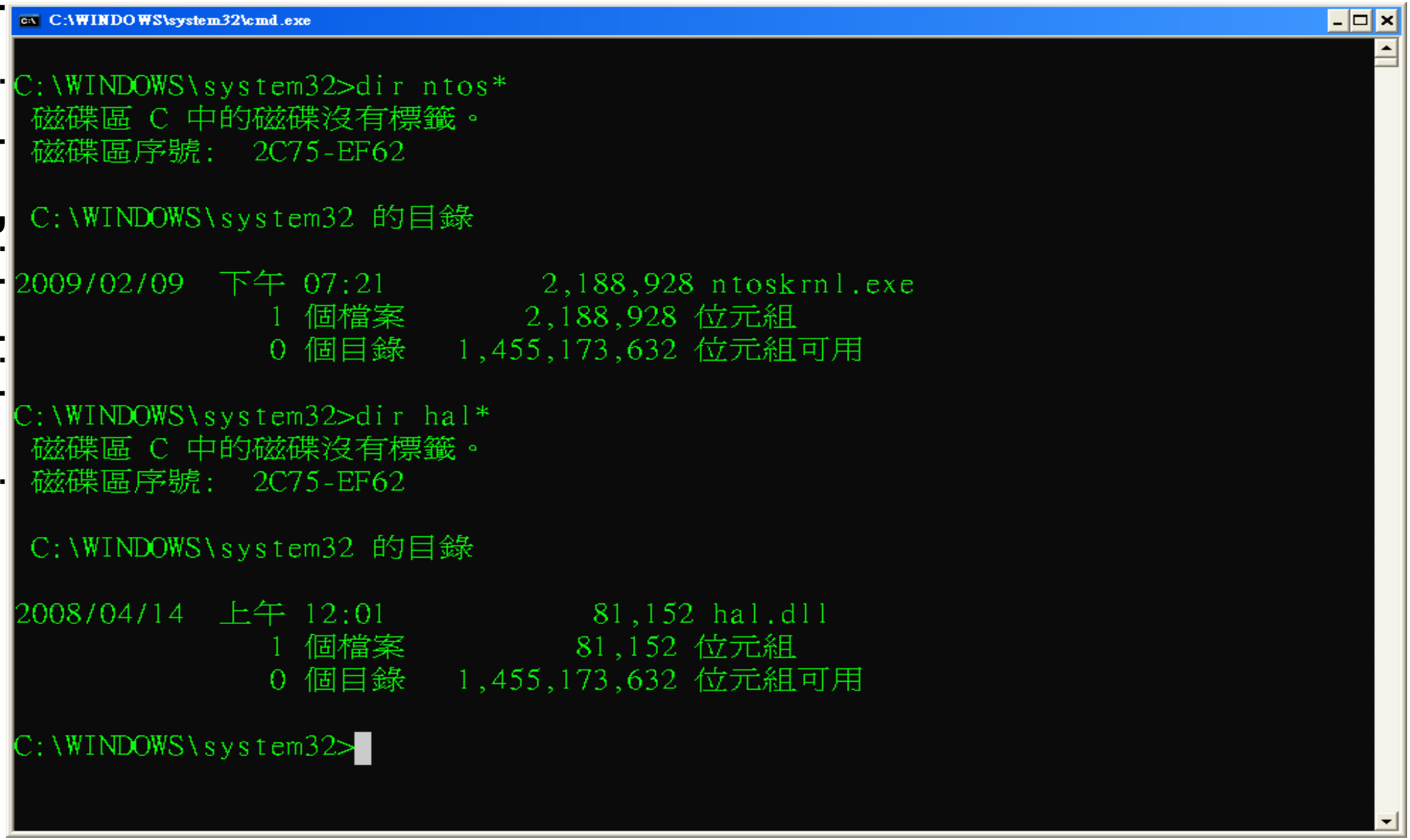
# NTLDR, boot.ini, NTDETECT.COM, …



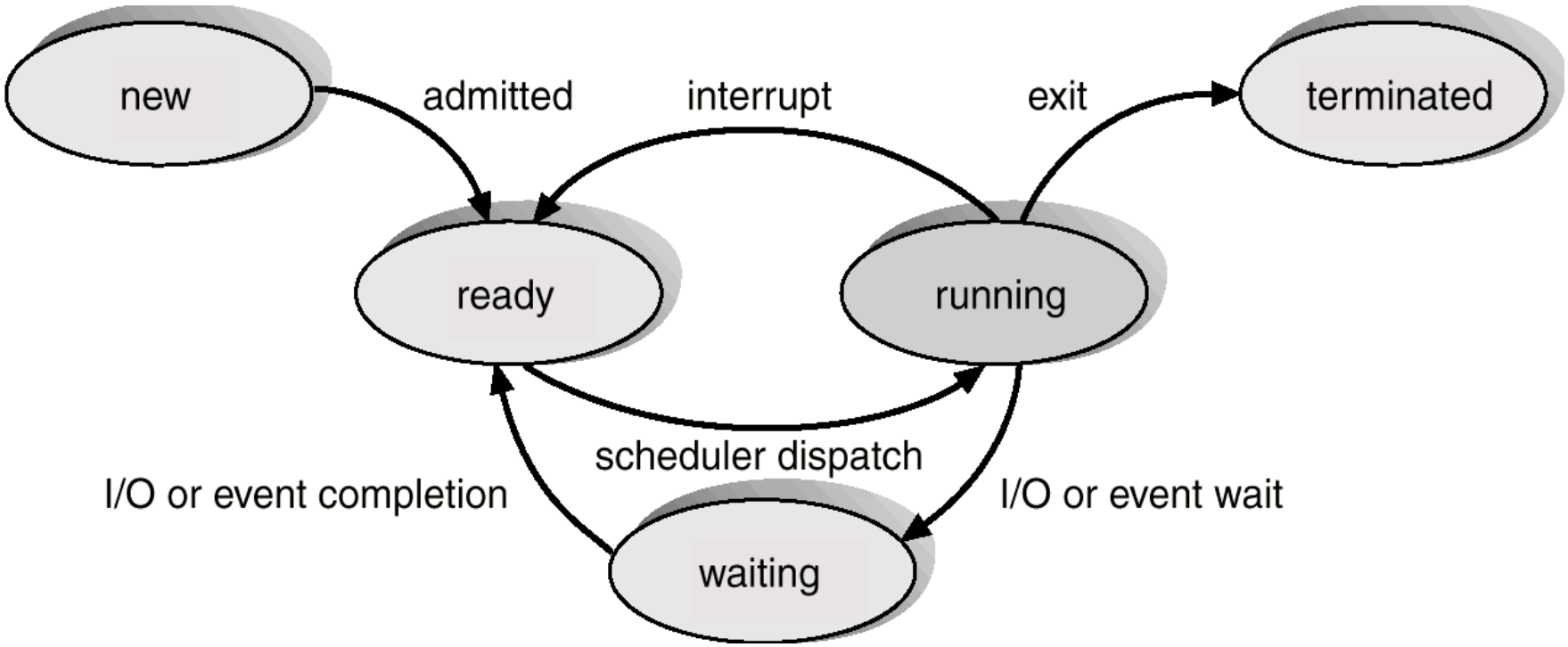

http://en.wikipedia.org/wiki/NTLDR

# NTOSKRNL.EXE, hal.dll

http://en.wikipedia.org/wiki/NTLDR



# Diagram of process/thread State Transition




thread vs. process

thread == Light weight process

Kernel thread vs. User thread

# Terminologies regarding OS

- Critical section: a section of code which reads or writes shared data
- Race condition: potential for interleaved execution of a critical section by multiple threads/processes
  - Results are non-deterministic
- Mutual exclusion: synchronization mechanism to avoid race conditions by ensuring exclusive execution of critical sections (Java 用 synchronized 達成)
- Deadlock: permanent blocking of threads/processes
- Starvation: execution but no progress (Livelock)
  (Dining Philosopher problem)

http://en.wikipedia.org/wiki/Critical_section

# Competition among Processes

- **Shared data (shared resource) problem**
  - **Concurrency with shared data problem**
  - **Deadlock, Livelock, …**
    - 兩隻狗從獨木橋兩端同時走到橋中間互不相讓?
- **Mutual exclusion**
  - **Semaphores (next slide)**
    - Test-and-set instruction (a primitive)
      - **if ( mem[k] == 0 ) mem[k] = 1; //?**
    - Critical region (**Critical section**)
    - Race condition
    - Mutual exclusion
  - **Monitor (next slide)**

http://en.wikipedia.org/wiki/Mutual_exclusion

# The **Critical Section** Problem

- n processes all **competing** to use some **shared** data
- Each process has a code segment, called critical section (CS), in which the shared data is accessed.
- Problem – ensure that when one process is executing in its critical section, no other process is allowed to execute in its critical section
  - Even with multiple CPUs
- Each process must request the permission to enter its CS
- The critical section problem is to design a protocol that the processes can use so that their action will not depend on the order in which their execution is interleaved (possibly on many processors)
  - **Semaphore, Monitor : provided by OS**
  - **Test-and-Set instruction : Hardware**

http://en.wikipedia.org/wiki/Critical_section

# Semaphore

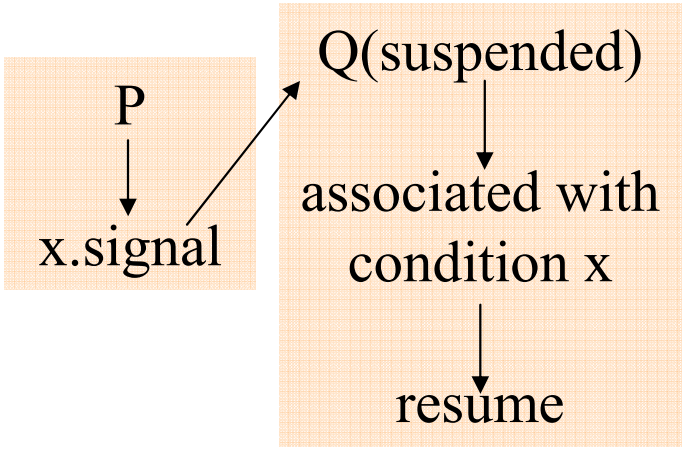
- Synchronization tool (provided by the OS) that does not require busy waiting
- Formally, a semaphore is comprised of:
  - An integer variable: value
  - Two **atomic** operations: P( ) and V( )
- P( ) --- also known as wait( )
  - While value = 0, sleep
  - Decrement value and return
- V( ) --- also known as signal( )
  - Increments value
  - If there are any threads sleeping waiting for value to become non-zero, wakeup at least 1 thread

Java:
notify( )
notifyAll( )

http://en.wikipedia.org/wiki/Semaphore_(programming)

http://en.wikipedia.org/wiki/Producers-consumers_problem

# Monitors




- A Monitor is essentially a class with private methods, plus a queue.

- A monitor is a set of programmer-defined operations that are provided mutual exclusion within the monitor (the monitor construct prohibits concurrent access to all procedures defined within the monitor)
- Type of *condition*: *x.wait* and *x.signal*
- *Signal-and-Wait:* P=>wait Q to leave the monitor or another condition
- *Signal-and-Continue:* Q=>wait P to leave the monitor or other condition

**en.wikibooks.org/wiki/Operating_System_Design/Critical_Section_Problem/Monitor**

Java uses Monitors for Mutual Exclusion:
wait( ), notify( ), notifyAll( )

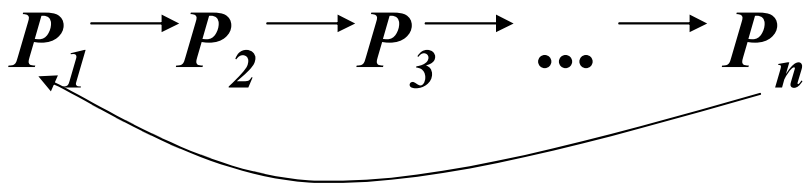
# Deadlock

- Permanent **block**ing of a set of processes
- Normally due to the fact that they
  - wait for limited system resources for which they compete **or**
  - wait for messages
  - since messages can be seen as resources, in general it can be said that it is due to contention on resources.
- There is **no satisfactory solution** in the general case
  - to determine whether a program contains a potential deadlock is a computationally unsolvable problem

http://en.wikipedia.org/wiki/Deadlock

# 4 Necessary Conditions for Deadlock

- 1) Mutual exclusion:
  - One process hold a resource in a non-sharable mode. Other processes requesting resource must wait for resource to be released.
- 2) Hold-and-wait:
  - A process must hold at least one allocated resource while awaiting other resources (one or more) held by other processes.
- 3) No preemption:
  - No resource can be forcibly removed from a process holding it. That is, resources are voluntarily released by the process holding it.
- 4) Circular wait (循環等待)
  - A closed chain of processes exists, such that each process holds at least one resource needed by the next process in the chain.



# Dining Philosopher Problem

In 1965, ***Edsger Dijkstra*** set an examination question on a synchronization problem where five computers competed for access to five shared **tape drive** peripherals.

Soon afterwards the problem was retold by ***Tony Hoare*** as the dining philosophers problem.

- Five philosophers sitting at a round table doing one of two things: **eating** or **thinking**.
- A fork is placed in between each pair of adjacent philosophers, and as such, each philosopher has one fork to his left and one fork to his right.
- It is assumed that a philosopher must eat with two forks. Assuming each philosopher takes a different fork as a first priority and then looks for another fork whenever he/she wants to eat.

http://en.wikipedia.org/wiki/Dining_philosophers_problem

Starvation : LiveLock

# Processes vs. threads

- Different meanings when operating system terminology
- Regular processes
  - Heavyweight process
  - Own virtual address space (stack, data, code)
  - System resources (e.g., open files)
- Threads
  - Lightweight process
  - Subprocess within process
  - Only program counter, stack, and registers
  - **Shares address space, system resources with other threads**
    - Allows quicker communication between threads
  - Small compared to heavyweight processes
    - Can be created quickly
    - Low cost switching between threads

The system call **fork( )** is used to fork a process

# Critical Section in Java program

```
public String aLock = "just a Lock ha ha ha";
///… critical section 可能只有一個片斷程式
synchronized( a Lock ) {
    /// critical code accessing shared data
}
/// critical section 也可能由好幾個片斷程式合起來組成
synchronized( aLock ) {
    /// Other critical code accessing shared data
}
```

考慮有很多份(thread)上述程式“同時”在執行

考慮黑板上寫著教室內現有人數, 由兩人負責更新?

# Programming Model

- Client / Server

  http://en.wikipedia.org/wiki/Client-server

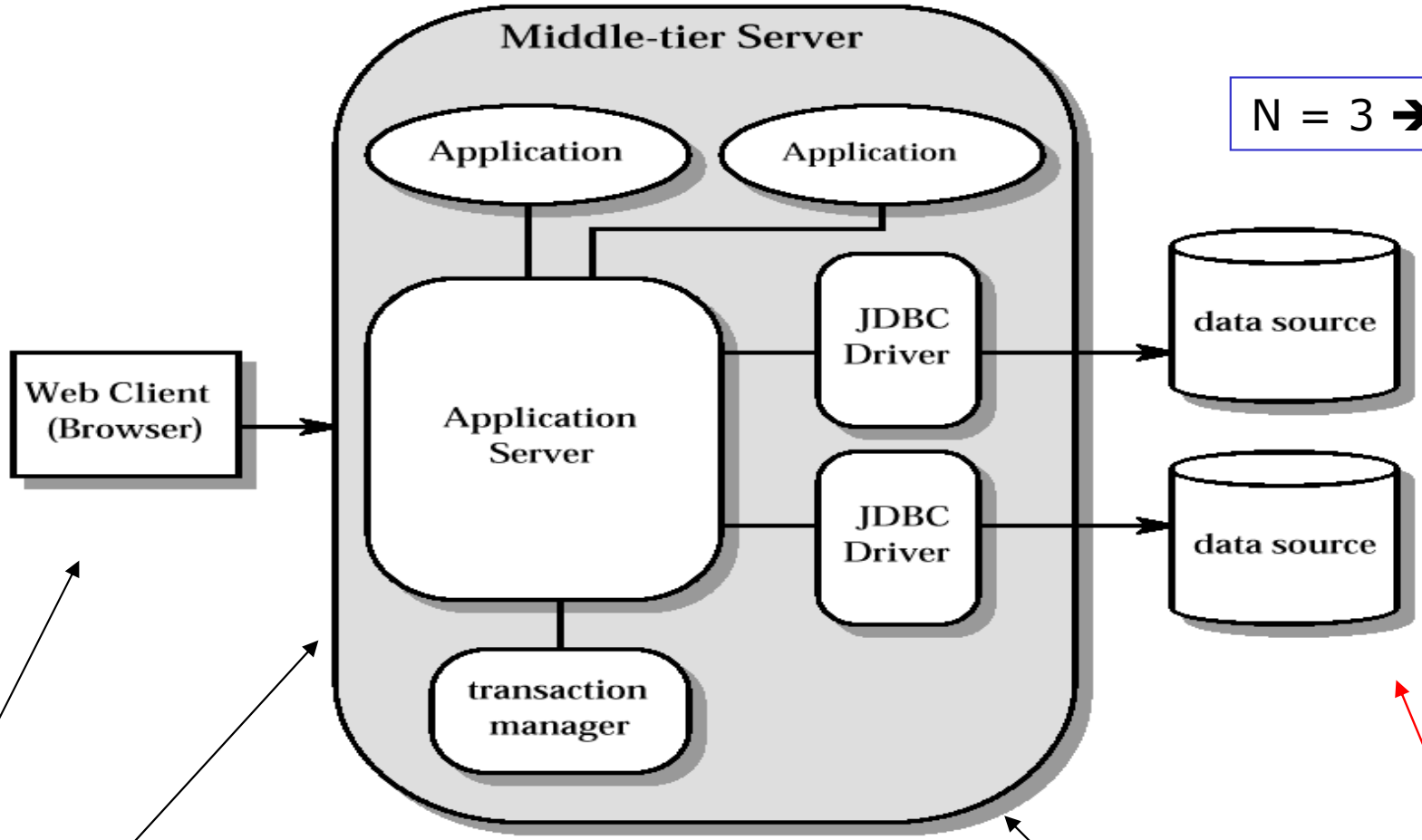
  – BBS
  – News
  – FTP
- Traditional (non-component) N-Tier Systems
  – 3-Tier Programming (N == 3)
    - Browser + HTTP Server + DBMS

http://en.wikipedia.org/wiki/Multitier_architecture

- N-Tier Programming, JavaEE

http://en.wikipedia.org/wiki/Java_Platform,_Enterprise_Edition

# Traditional (**non-component**) N-Tier Systems

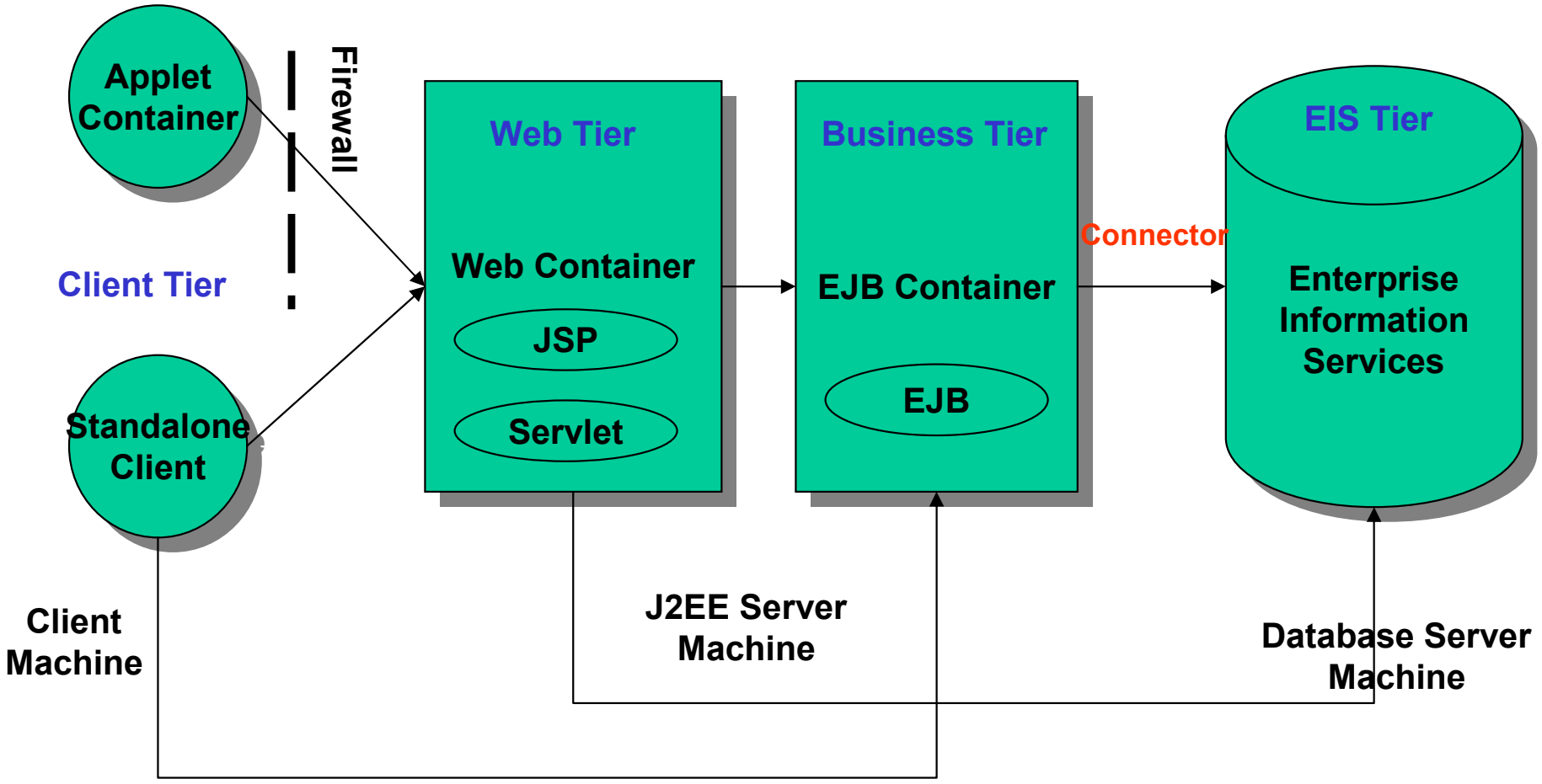



- Client Tier – Presentation Logic
- Application Tier – Business Logic
- Data Tier – Database Management Services

Source: **Sun Microsystems**, Inc., *JDBC 3.0 Specification*

# Component N-Tier Systems

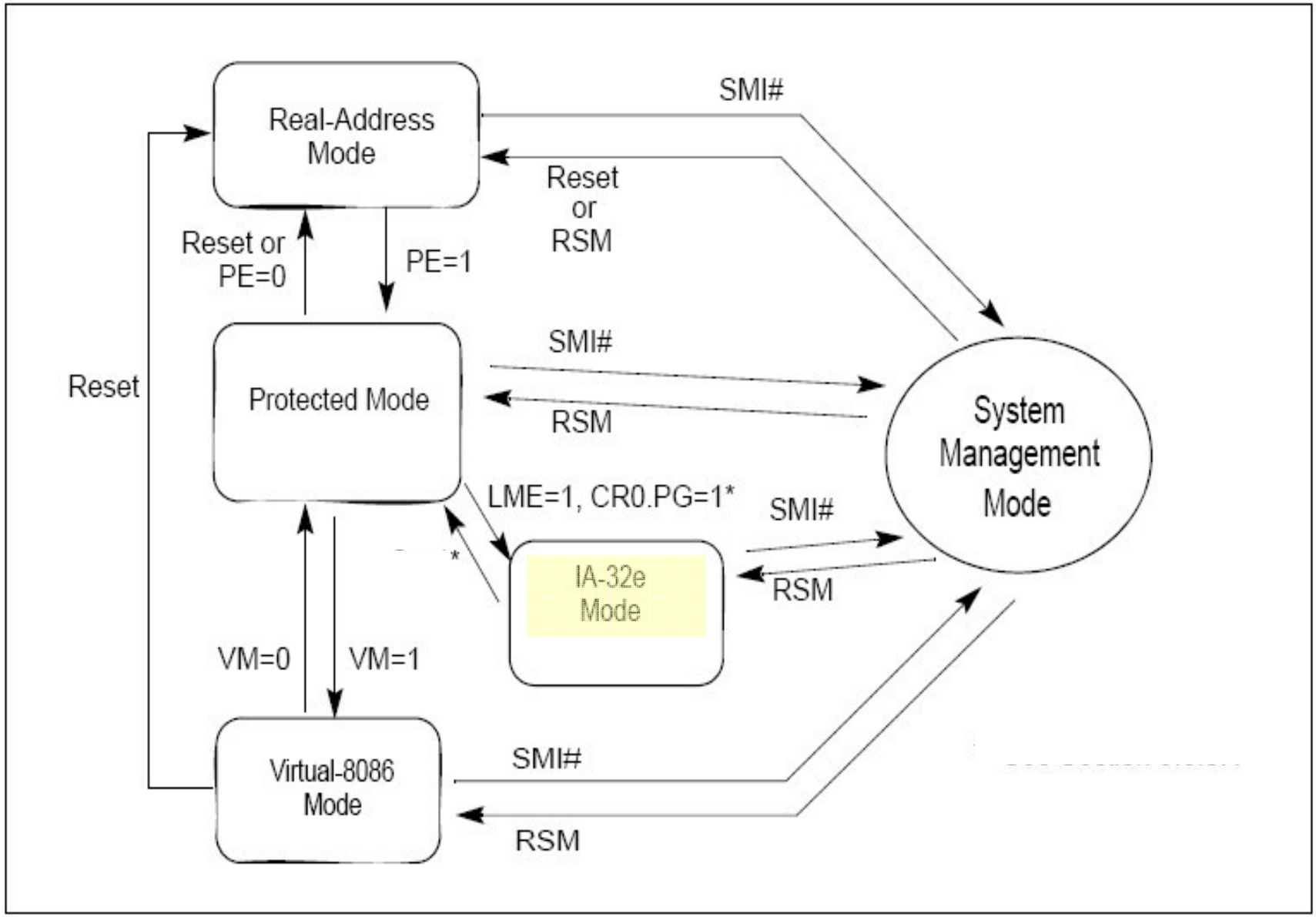
## J2EE Architecture (Java EE)




http://en.wikipedia.org/wiki/Java_Platform,_Enterprise_Edition

http://en.wikipedia.org/wiki/.NET_Framework

# X86 processor's modes Transitions



# Segmentation and paging in x86 IA-32

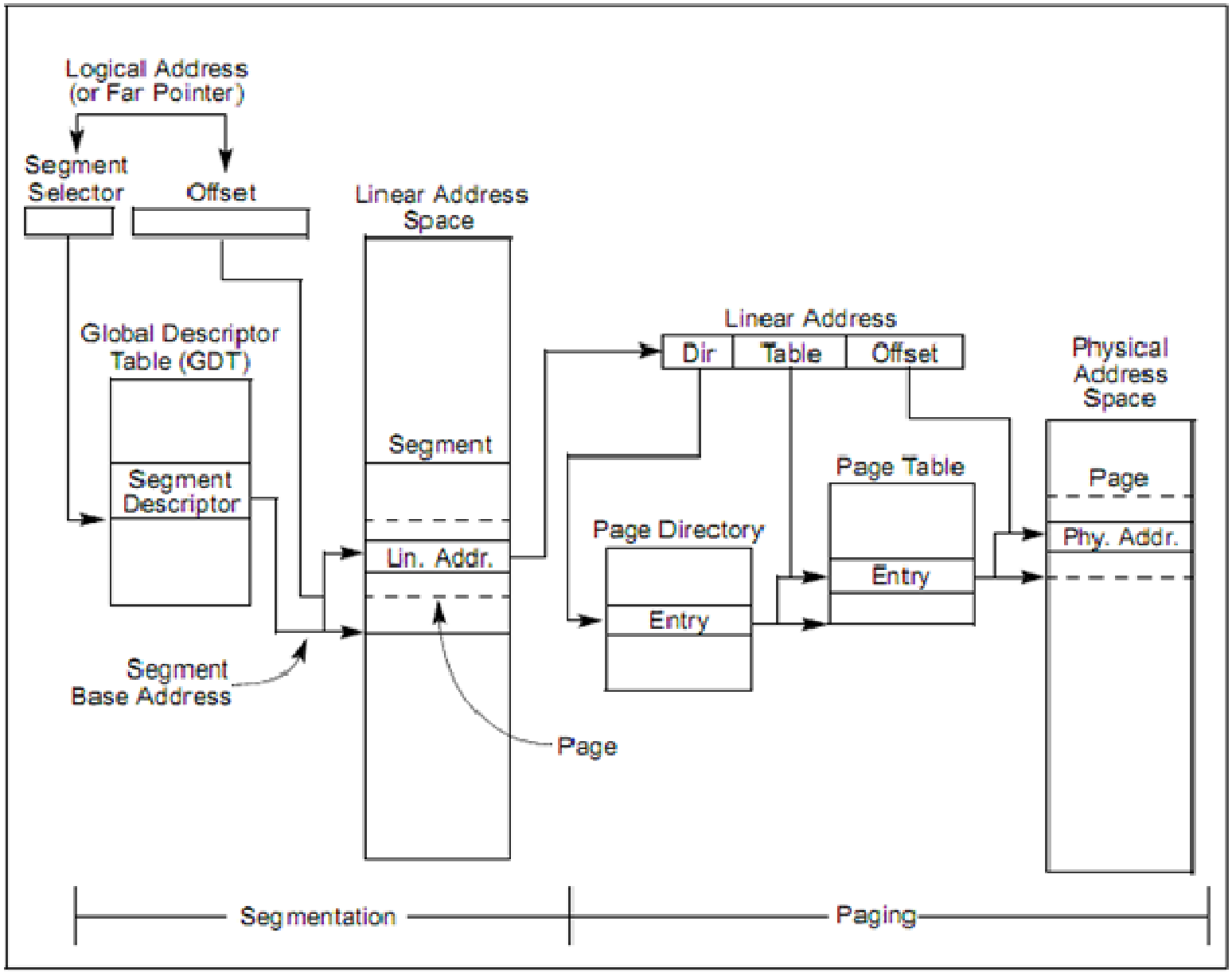



Figure 3-1. Segmentation and Paging

# Interrupt handling in x86 protected mode

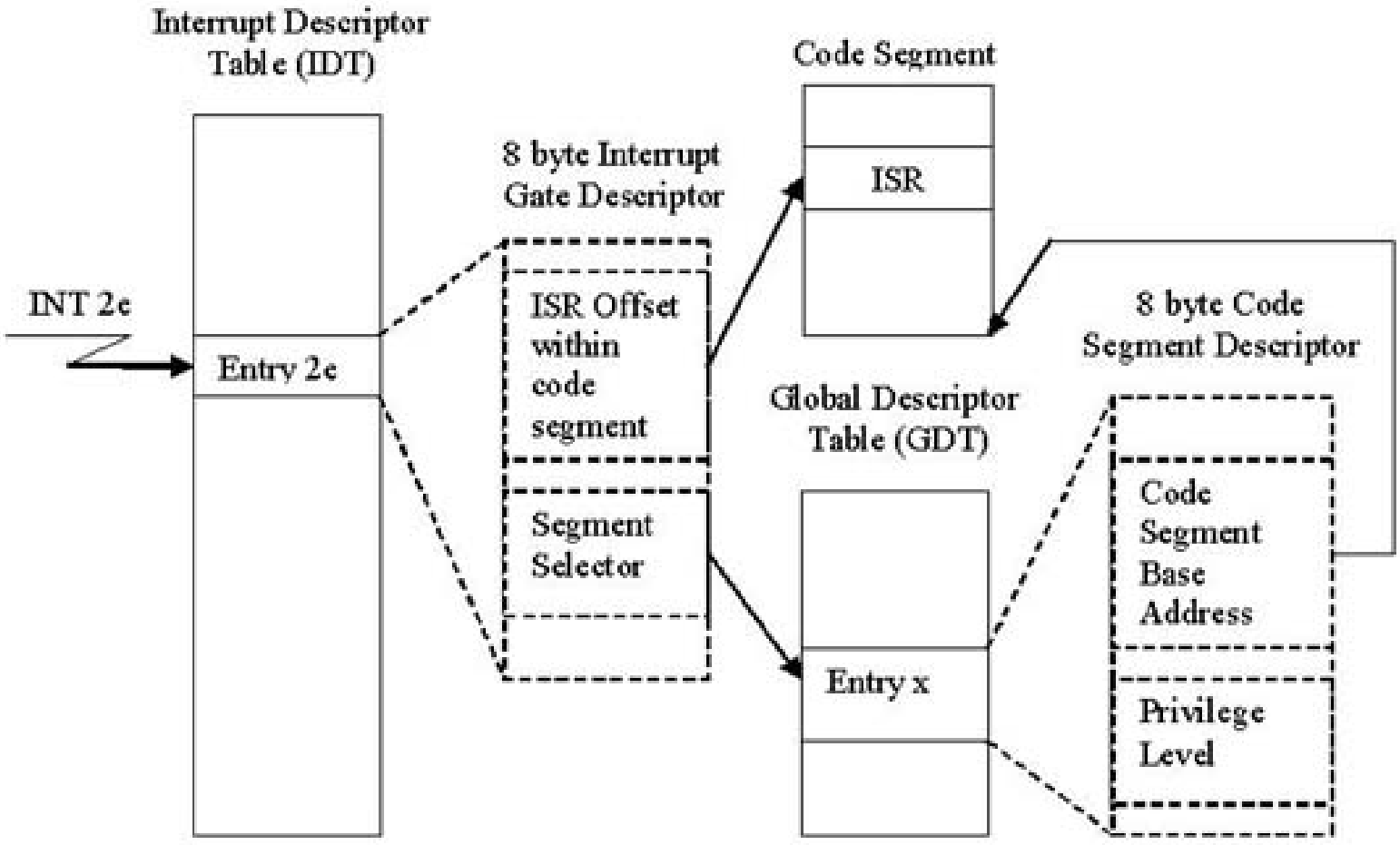
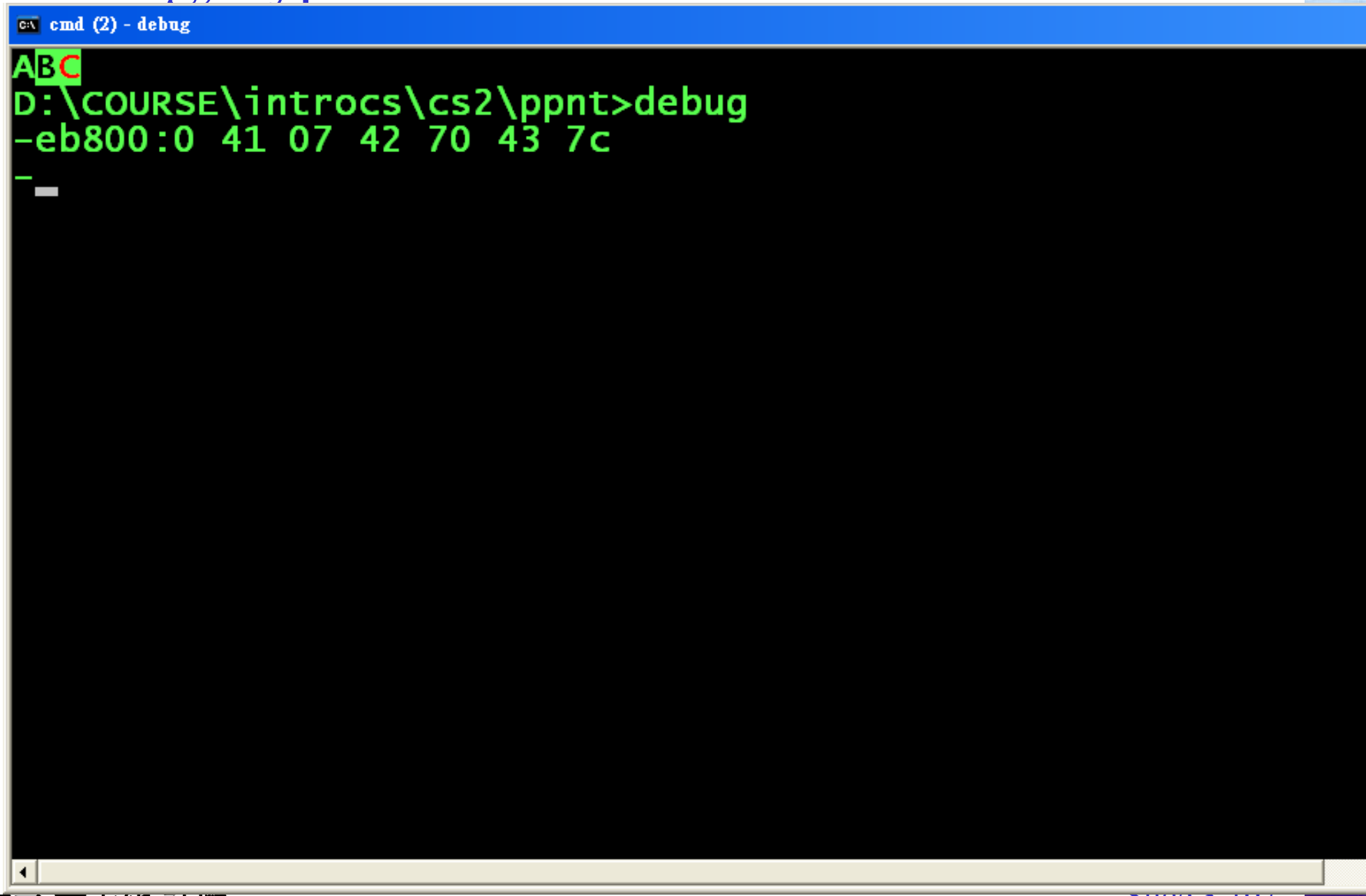



Figure 3. How the CPU finds the Interrupt Service Routine for software interrupt 2e.

# Memory mapping example -- in Debug, type eb800:0 41 07 42 70 43 7c



# Thank You!

CHAPTER 3

## Operating Systems

謝謝捧場

*tsaiwn@csie.nctu.edu.tw*

蔡文能

# 我猜 我猜 我猜 猜 猜
# 計科概期中考: 2010/04/27 Tuesday

- Briefly compare **Monolithic Kernel** with **Microkernel**. And also name some examples for each of them.

**Ans:** 主要差別在Microkernel **(微核心)**把很多原本在 kernel mode 的服務踢到 user mode, 然後採用 client-server 的架構把 OS services提供給在 user mode 的Application. 然而 **Monolithic Kernel (全核心)**則是整個OS kernel 全都在 kernel mode(或說supervisor mode), Application 直接叫用需要的 System calls 要求 OS 幫忙做事; Unix, Linux, FreeBSD 以及大多數的 OS 是**Monolithic Kernel**; Apple 的 OS X 與 iPhone 的 OS 則是採用 **Monolithic Kernel** 架構.

註: 如再畫個圖輔助說明更好 ☺

# 我猜 我猜 我猜 猜 猜

- Regarding Timesharing OS
  - Briefly describe the timesharing concept.
  - Which two schools invented the “timesharing” concept?
    Ans: Dartmouth college + MIT
    (hint: one of them invented BASIC language)
- Regarding GNU
  - What does GNU stands for? ➔ GNU is Not Unix
  - Who started the GNU project? (hint: a professor of MIT)
  - GPL (GNU Public License ) from FSF (Free Software Foundation) states that Free software comes with **four freedoms**. Briefly describe these **four freedoms**.
- Regarding Acronym (頭字語) …

# 我猜 我猜 我猜 猜 猜

- BIOS, POST, Bootstrapping
- 4 **Necessary** Conditions for Deadlock
  - **1) Mutual exclusion:**
  - **2) Hold-and-wait:**
  - **3) No preemption:**
  - **4) Circular wait (循環等待)**
- Example: **有三個 process 都要**用 讀卡機+印表機+磁帶機 ;
  這三樣都是一次只能有有一人用 (不能共同使用!) ➔ Mutual exclusion
  process A 先要了讀卡機, process B 先要了印表機, process C 先要了磁帶機

**Hold-and-wait:** 過了一會兒, A 又要印表機, B又要磁帶機, C又要讀卡機
➔ A, B, C 各自握著一個資源不放, 且又要求另外目前在別人手上的資源

*** **OS 被設計成**不會主動奪走各 process 手中資源 ➔ **No preemption**

**** A 要的在 B 手上, B 要的在 C 手上, C 要的在A 手上➔ **Circular wait**

# 我猜 我猜 我猜 猜 猜

- Hamming distance ?
  - of two bit-strings
  - of a code set
- Logical Gates
- 0111 + 0110 ➔ NZVC Cin
- ECC – Error Correcting Code ?
- 1101 ?

| | |
|---|---|
| A | 1100 |
| B | 1011 |

- SISC program

# Memory layout on Intel-based PC