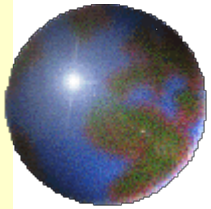
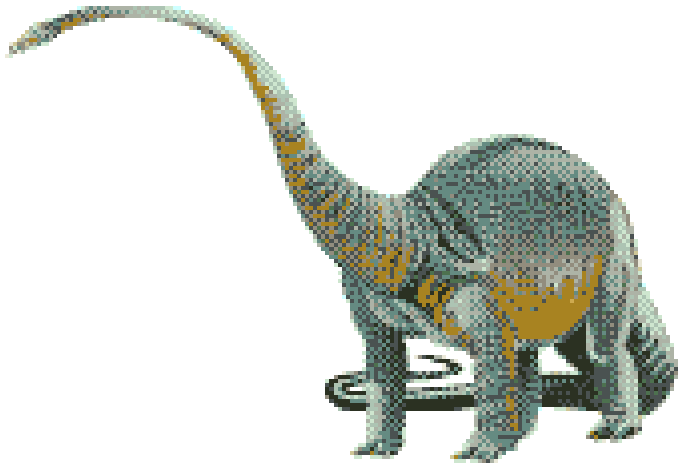


作業系統(Operating Systems)

Course 3: Operating-System Structures (作業系統結構)



授課教師:陳士杰

國立聯合大學　資訊管理學系

本章重點

- O.S.之User Interface
  - Command Interpreter
  - System Call
    - 定義
    - 種類
    - 參數傳遞方式
- Virtual Machine
  - 定義
  - 目的
- Java Virtual Machine
- Microkernel

- O.S.提供程式執行的環境，在開始設計之前，先以一些不同的方向觀察一個O.S.的架構：
  - 檢查它所提供的服務 (2.1)
  - 察看它對使用者和程式人員(即：User Program)所提供的介面 (2.2~2.4)
  - 觀察O.S.的元件(1.6~1.9)和連接部份(2.7)

■作業系統服務 (O.S. Service)

- O.S.對User Program與User提供了一定的服務:
  - 程式執行
    - O.S.必須能夠將一個程式載入記憶體中執行，並以正常的方式來終止其運行。
  - I/O運作
    - 一個執行中的程式可能會要求I/O運作的服務。因為使用者不能直接執行I/O運作，所以O.S.必須提供介面讓User Program可間接執行I/O。
  - 檔案系統處理
    - 程式可以讀、寫、建立和刪除檔案。
  - 通訊
    - 一個Process可能需要和另一個Process交換資訊。
    - 通訊方式有:共享記憶體和訊息傳遞兩個技術

- 錯誤偵測
  - **0.S.**需要經常對潛在錯誤進行偵測。
- 資源分配
  - 當系統中有多個使用者或多項工作同時運行時，必然會產生資源分配的問題。
- 記帳
  - 記錄哪一個使用者使用了多少資源及使用了何種資源。
- 保護
  - 當幾個不同的工作正在同時執行時，不應出現工作之間相互干擾，或工作干擾**0.S.**本身的情況。

一般的系統元件 (Common System Components)

- 通常我們想要建構像作業系統一般的一個大且複雜的系統時，只有先將系統**分割成較小的部份**才比較容易進行。
- 許多現行的作業系統大多共有下列的系統元件：
  - **Process Management (行程管理)**
  - **Main Memory Management (主記憶體管理)**
  - **Storage Management (儲存體管理)**
    - **File Management (檔案管理)**
    - **I/O System Management (I/O系統管理)**
    - **Secondary Management (輔助記憶體管理)**
  - **Protection System (保護系統)**

行程管理 (Process Management)

- **Process vs. Program**
  - 程式 (Program): 被動的實體
  - 行程 (Process): 主動的實體
- 行程需要某些特定的資源以完成其工作，如：
  - **CPU時間**
  - **記憶體**
  - **檔案**
  - **I/O裝置**
- 在行程管理方面，作業系統需提供下列的功能：
  - 系統與使用者**行程的建立與刪除**
  - 行程的**暫停與回復**
  - 提供行程的**同步**、**通訊**與**死結處理**機制
- **Ch. 3~ Ch. 6**

主記憶體管理 (Main-Memory Management)

- 主記憶體為**CPU唯一可以直接定址與存取**的大型儲存裝置。指令必須在主記憶體之中，CPU才能執行它們。
- 為了改善**CPU使用率**和**電腦對使用者的回應速度**，必須**保持許多程式在記憶體**之中。
- 在記憶管理方面，作業系統需提供下列的功能：
  - 記錄哪一部份的記憶體**正在被使用**，及**是誰在使用**
  - 當記憶體空間為**可用**時，決定**載入的行程**
  - 在需要時，負責**配置與回收記憶體空間**
- **Ch. 8 ~ Ch. 9**

儲存體管理 (Storage Management)

- 檔案管理
- 輔助儲存體管理
- I/O系統管理

檔案管理 (File Management)

- 檔案是由建檔者所定義之相關資訊所構成的集合。通常，檔案表示**程式 (原始碼或是目的碼) 與資料**。
- 在檔案管理方面，作業系統需提供下列的功能：
  - 檔案的建立與刪除
  - 目錄的建立與刪除
  - 作為處理檔案和目錄的原始支援
  - 檔案與輔助記憶體之間的對映
  - 檔案備份於穩定(非揮發性)的儲存媒體
- **Ch. 10 ~ Ch. 11**

I/O系統管理 (I/O System Management)

- 作業系統的目的之一，就是**對user隱藏特定硬體裝置的特性**。
- I/O子系統包括：
  - 記憶體管理的構成要素
    - **緩衝**、**快取**與**Spooling**(連線同時週邊作業)
  - 通用的**裝置驅動程式介面**
  - **特定硬體裝置驅動程式**
- **Ch. 1 and Ch. 13**

輔助記憶體管理 (Secondary-Storage Management)

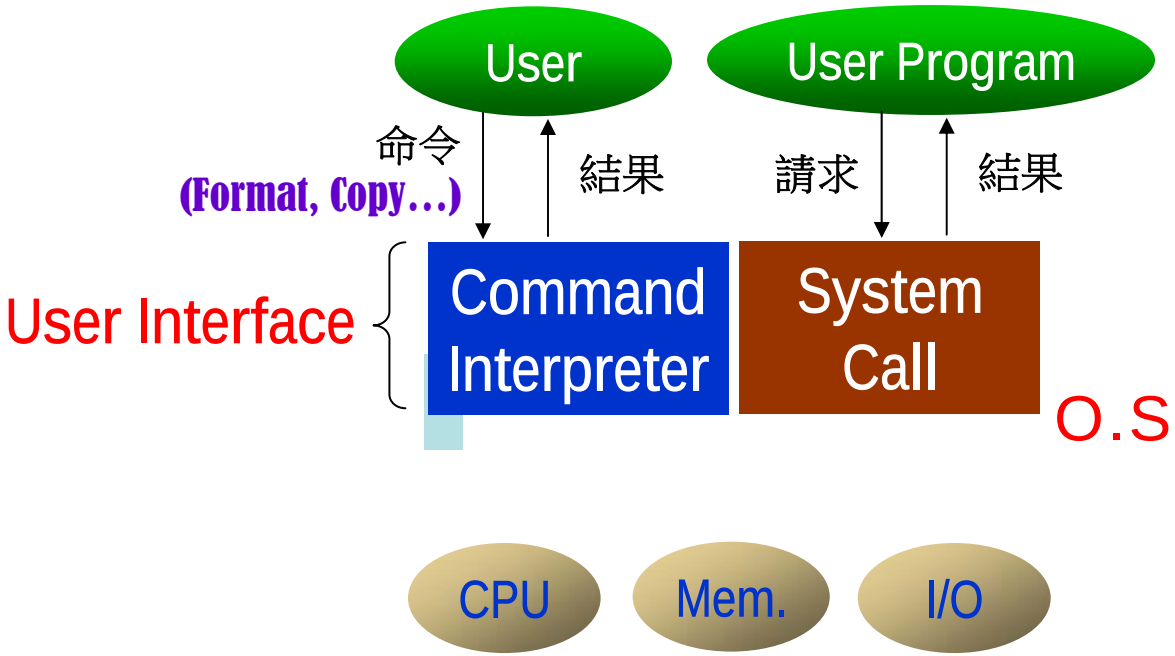
- 現今大部份的電腦系統是使用**磁碟**作為主要的線上儲存媒體，以儲存所有的程式與資料。
- 當CPU需要執行某程式片斷或是某部份資料時，才將該程式片斷或是部份資料由磁碟載入到主記憶體中。
- 因為輔助記憶體經常使用，所以它必須有效率地使用。在輔助記憶體管理方面，作業系統需提供下列的功能：
  - **可用空間的管理**
  - **儲存體的配置**
  - **磁碟排程**
- **Ch. 14**

保護系統 (Protection System)

- 如果電腦系統有**許多使用者**並且**允許同時執行多個行程**時，就必須保護每一個行程不受到其他無關的程式或活動給破壞掉。
- **保護(Protection)**是一種經由電腦系統定義的控制功能，它控制著**各程式、行程和使用者**對系統和資源進行的**存取動作**。
- 因此，一個保護機制必須：
  - 能夠分辨**經過授權**和**未經授權**的使用
  - 指定**欲施行的控制方式**
  - 提供**強制性的方法**
- **Ch. 18**

■O.S.的User Interface

- 所有的程式 (包括O.S.) 都會有User Interface。
- O.S.是服務User和User Program。
  - Command Interpreter
  - System Call



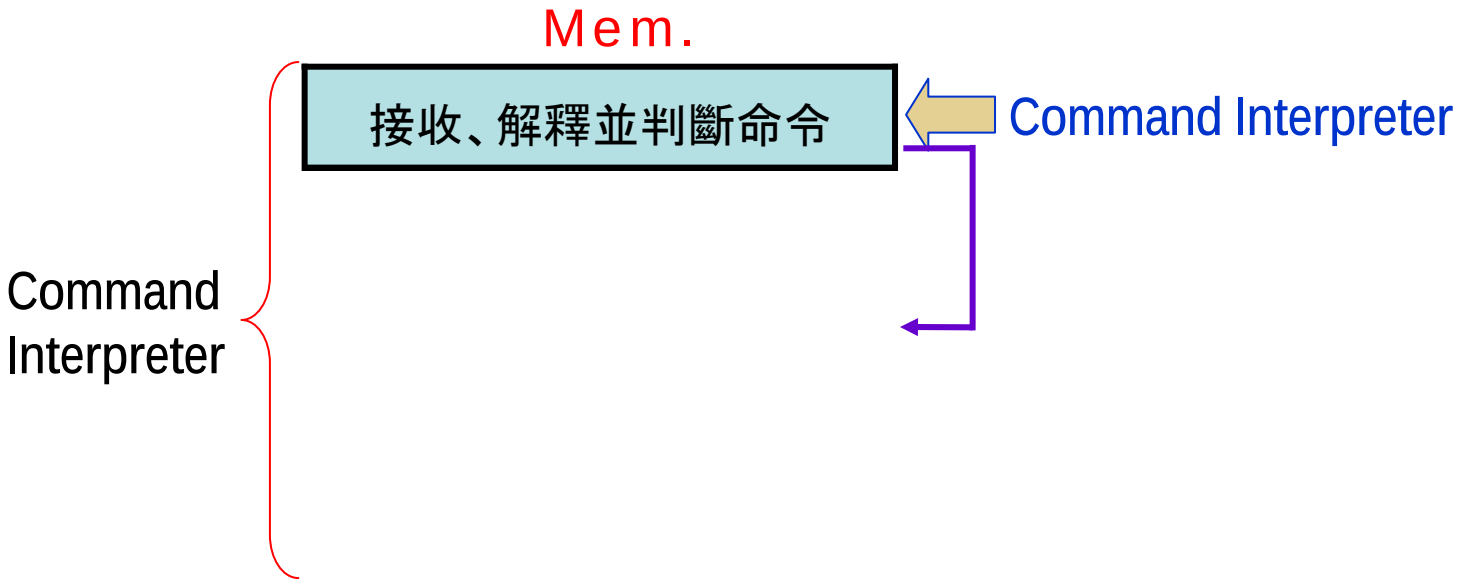
■Command Interpreter (命令解譯器)

- 目的：作為User與O.S.之間溝通的界面。
- 負責：
  - **接收**使用者所input的命令 (User Command)
  - **解釋並判斷**命令正確與否
    - 不正確的話，會有 **"Bad Command..."**等錯誤訊息出現!!
    - 若正確，則可啟動**對應的Service Routine (Command Routine)** 執行
  - 將結果呈現給User
- **Command Interpreter設計或製作時的考量議題**
  - **Command Interpreter是否要包含Service Routine?**
  - **Command Interpreter與O.S. Kernel Module的關連程度?**

Command Interpreter是否要包含Service Routine?

- 此議題是考量Command Interpreter 與Service Routine是否合併在一起?
- 作法有二:
  - Service Routine為Command Interpreter之一部份
  - Service Routine與Command Interpreter分開

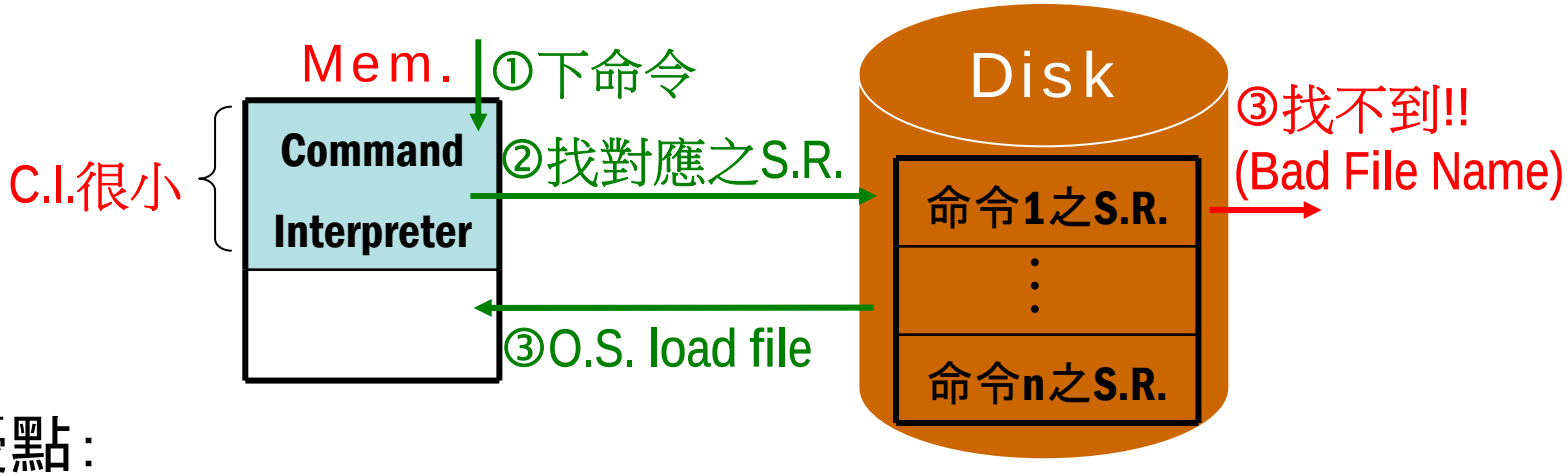
Service Routine (S.R.) 為Command Interpreter之一部份



- 優點:
  - 命令執行速度快 (∵O.S.載入Mem.時，將Command Interpreter和所有的S.R.也載入Mem.中)
    - 如:DOS中的Command.com
- 缺點:
  - 命令之數量受限於Command Interpreter大小
  - 無法增刪命令，彈性度低 (∵增刪命令要將O.S.重新Compile)

Service Routine (S.R.) 與Command Interpreter分開

- 所有的Service Routine皆以**files型式**存在於**Storage System** (即：Disk)中。
- Command Interpreter只扮演**Loader**之角色。



- 優點：
  - Service Routine不會佔用大量的Mem. Space。
  - 命令增/刪容易，且不影響O.S.的Limit。
  - 不同的User亦可以創造不同的User Environment (使用者操作環境)
- 缺點：
  - 命令執行速度較慢 (∵需增加Load File的Disk I/O Time)

- 在DOS中是採用上述**兩個作法的混合**，
  - 將經常使用的命令副程式讓它們成為C.I.的一部份。
- 所以當下錯命令時，會出現：

"Bad Command or File Name"

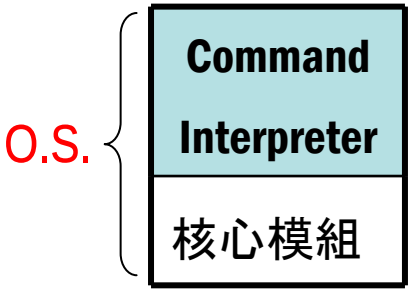
(在Mem.中內建的命令)
如：dir, copy...等較常用的命令。

(在Disk中的外部命令)
是較不常用的命令。

Command Interpreter與O.S. Kernel Module的關連程度?

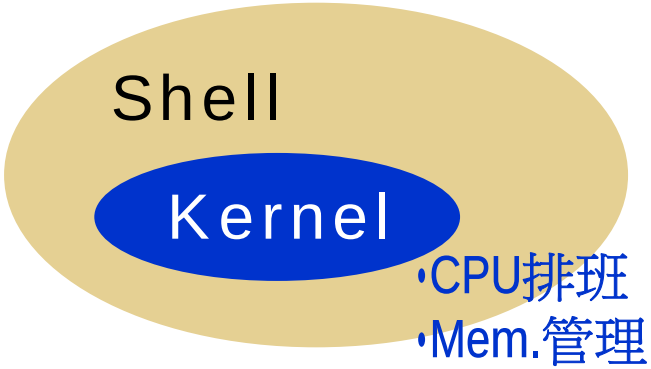
- 此議題是考量Command Interpreter 與O.S.之Kernel Program是否合併在一起?
- 作法有二:
  - 與Kernel合併
  - 與Kernel分開

- **與Kernel合併 (緊密結合)**
  - **Ex: Windows系列**



- **缺點:**
  - **Command Interpreter變更，則O.S.亦隨之變更。**

- **與Kernel分開 (鬆散結合)**
  - 如 : **UNIX**的**Shell**與**Kernel**是分開的。
    - **Shell**可任意更改，而不會影響**Kernel**，即**User Interface**之變更不會影響到**O.S.**的**Kernel**。
    - 每個**User**可建立自已的**Shell**。

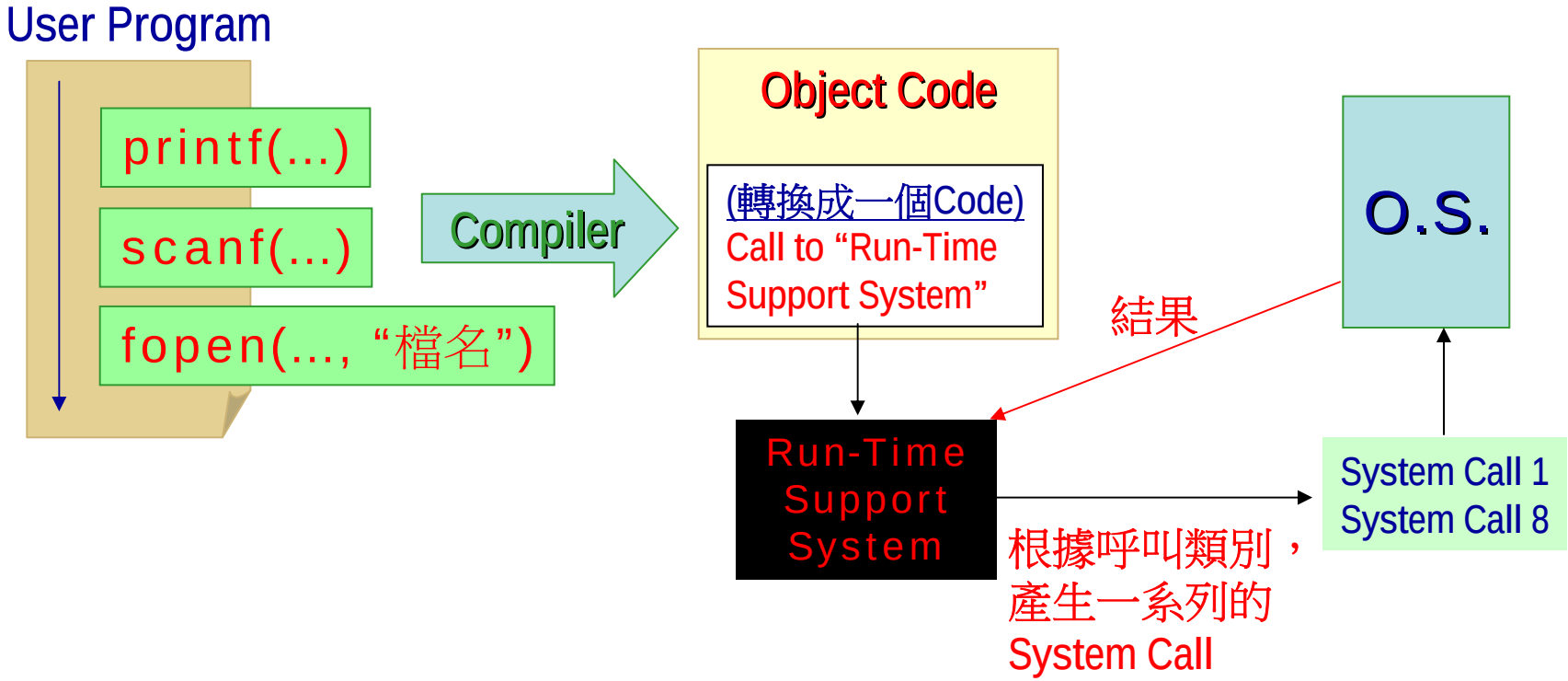


- **缺點:**
  - 相同的命令名稱格式，在不同的**User Environment**會有不同的解釋，可能造成誤用。

System Call (系統呼叫)

- **Def: 作為User Program和O.S.之間的溝通界面**
  - 當User Program執行時，若需要O.S.提供Service，則透過發出System Call通知O.S.，由O.S.執行相對應的Service Routine，以完成其要求的服務。
    - e.g., I/O Service。
  - System Call會伴隨一個Trap，以從User Mode轉換成Monitor Mode。
- System Call的類型可概分如下：
  - 行程控制 (Process Control)
  - 檔案管理 (File Management)
  - 裝置管理 (Device Management)
  - 資訊維護 (Information Maintenance)
  - 通信 (Communications)

- 通常User都看不到這些System Call的細節，因為透過In-Line 或Run-Time Support System的方式為大多數的程式語言提供了一個簡單的界面。



使用系統呼叫的實例

- 寫一個小程式，從一個檔案中讀取資料，並將之拷貝到另一個檔案中：
  - 程式要知道輸入和輸出檔案的名稱
    - 在Screen上顯示訊息
    - 從鍵盤上讀入字元作為檔案的名稱
  - 打開輸入檔案，並建立輸出檔案
    - 有一系列反應正常或錯誤狀況的系統呼叫
  - 讀取輸入檔案的資料，並將資料寫到輸出檔案
    - 每次讀和寫都必須傳回狀態消息以便注意各種可能的錯誤狀況
  - 關閉兩個檔案 (System Call)
  - 正常終止程式 (System Call)

System Call的參數如何傳遞給O.S.?

- [方法1] 利用Registers (暫存器)儲存這些參數
  - 優:速度快 (∵不用作記憶體的存取)
  - 缺:不適用於參數個數多的情況
- [方法2] 將參數利用Table或Block的方式儲存在Memory中，同時利用一個Register記錄此Table或Block的起始位址，並傳給O.S.
  - 優:適用於參數個數較多的情況
  - 缺:速度慢
- [方法3] 利用System Stack (堆疊)。要存放參數時，將之Push到Stack，再由O.S.從Stack中Pop取出參數。
  - Stack在系統中比Register多一些 (可用Mem. 或其它硬體 (如:Register)來實作此Stack)
- 方法2和方法3較不會限制欲傳遞的參數之數量或長度。

Microkernel (微核心)

- **Def**: 自核心移走一些非必要性的模組，這些模組交由**System Program**或**User Program**來製作，如此可得到**具有較少模組數的Kernel**稱之。

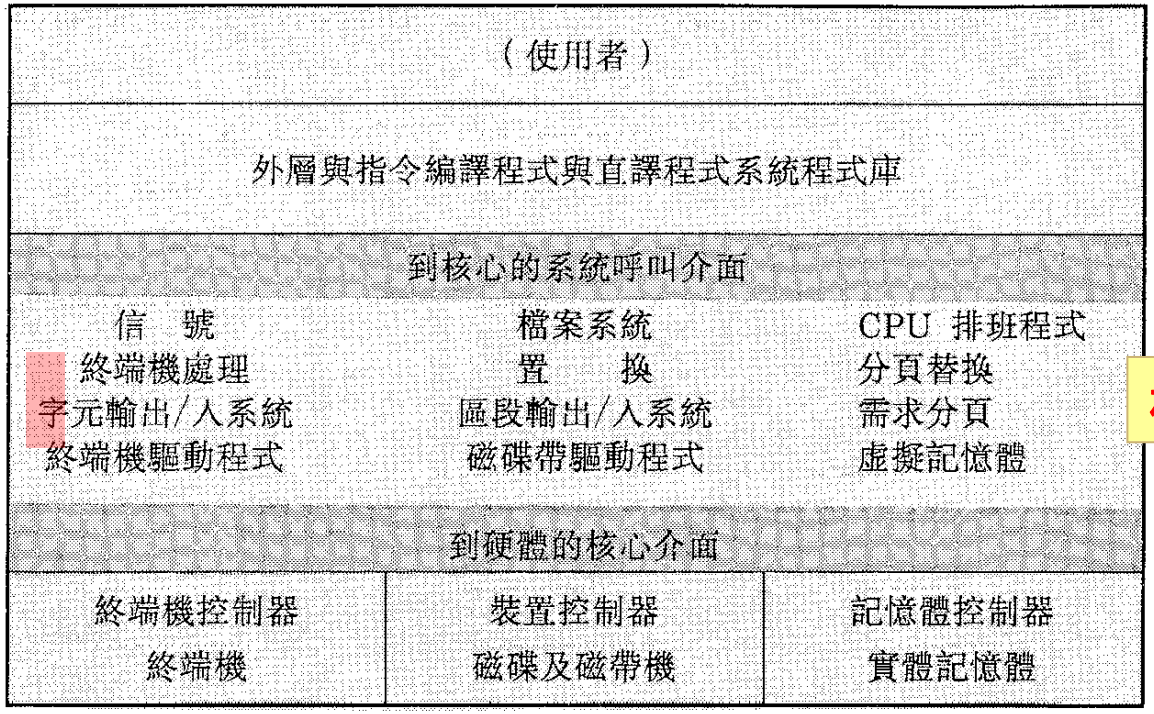
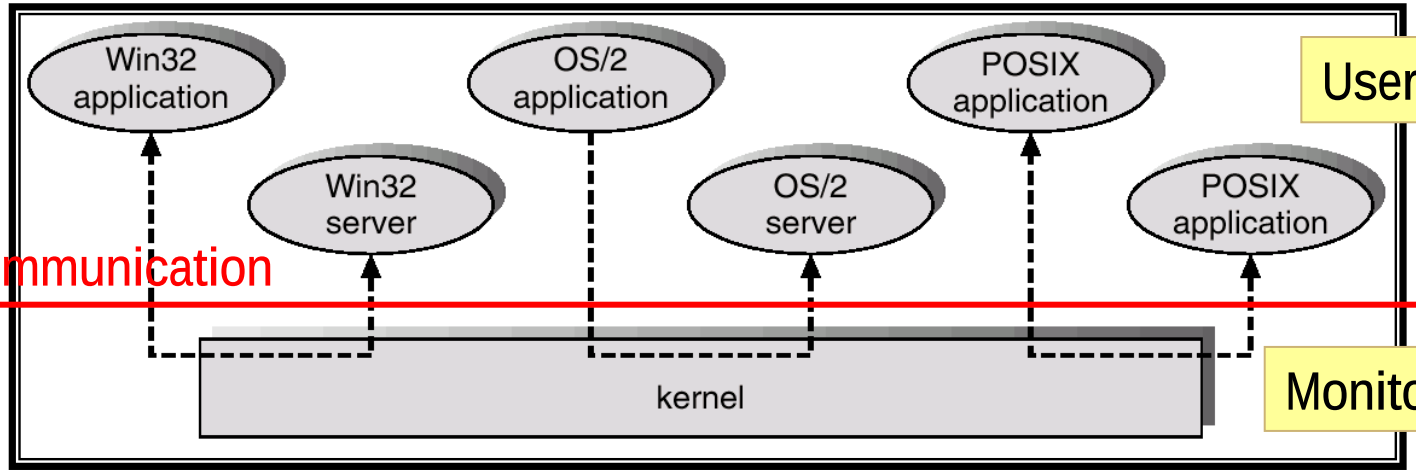


圖 3.7　UNIX 系統結構。

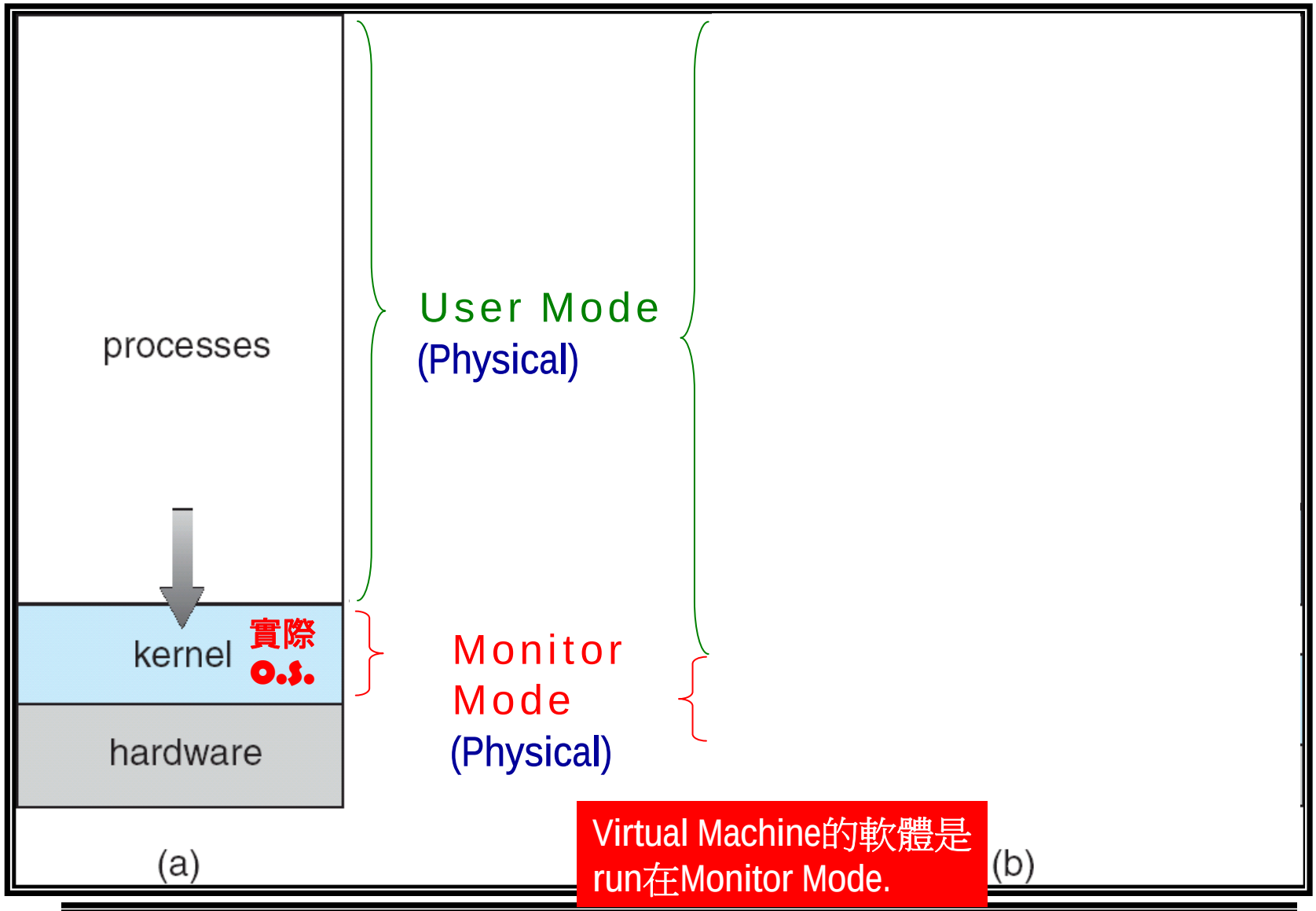
- 一般而言，**Microkernel**所保留的功能有：
  - **Basic Process Management**
  - **Basic Memory Management**
  - **Process Communication (e.g., Message Passing)**
- 微核心的主要功能是提供**Client User-Program**和同樣在使用者空間執行的**其它Server**彼此之間**通訊的便利**(經由**訊息傳遞(Message Passing)**提供通訊)。



- 優點:
  - **O.S.容易擴充**
    - 所有新加入的System Service都是加入到User Mode上執行，所以不會影響Kernel，就算是有，亦是小幅修改。
    - 所得到的O.S.易於從一個硬體平台轉移到另一個硬體平台上。
  - 確保安全性及實用性
    - 大部份的System Service是在User Mode執行，一旦Fail，也不會影響系統其它部份。

■虛擬機器 (Virtual Machine)

- **Def:** 透過**軟體模擬**的機制，創造一份與**底層Hardware一模一樣的功能介面**，稱為**Virtual Machine (VM)**。即：
  - 利用**CPU Scheduling**技術創造出有**多顆CPU**的效果。
  - 透過**Virtual Memory**技術，**擴大Memory Space**，創造出更多的空間。
  - 透過**Spooling**技巧，提供多套的**I/O Device**，達到**I/O Device共用**的目的。
- 虛擬機器最主要的概念是希望每個在電腦系統中的使用者都認為自己擁有一整部電腦，而且這部電腦上的資源都由一個人獨享，甚至隨時可以重新開機。
  - 對真正實際的系統而言，**V.M.**上的**O.S.**只相當於一個執行中的**User Program.** (開發**O.S.**時較為安全且方便)
  - 效能沒有真正的系統好(∵模擬的嘛!!)



Virtual Machine的軟體是run在Monitor Mode.

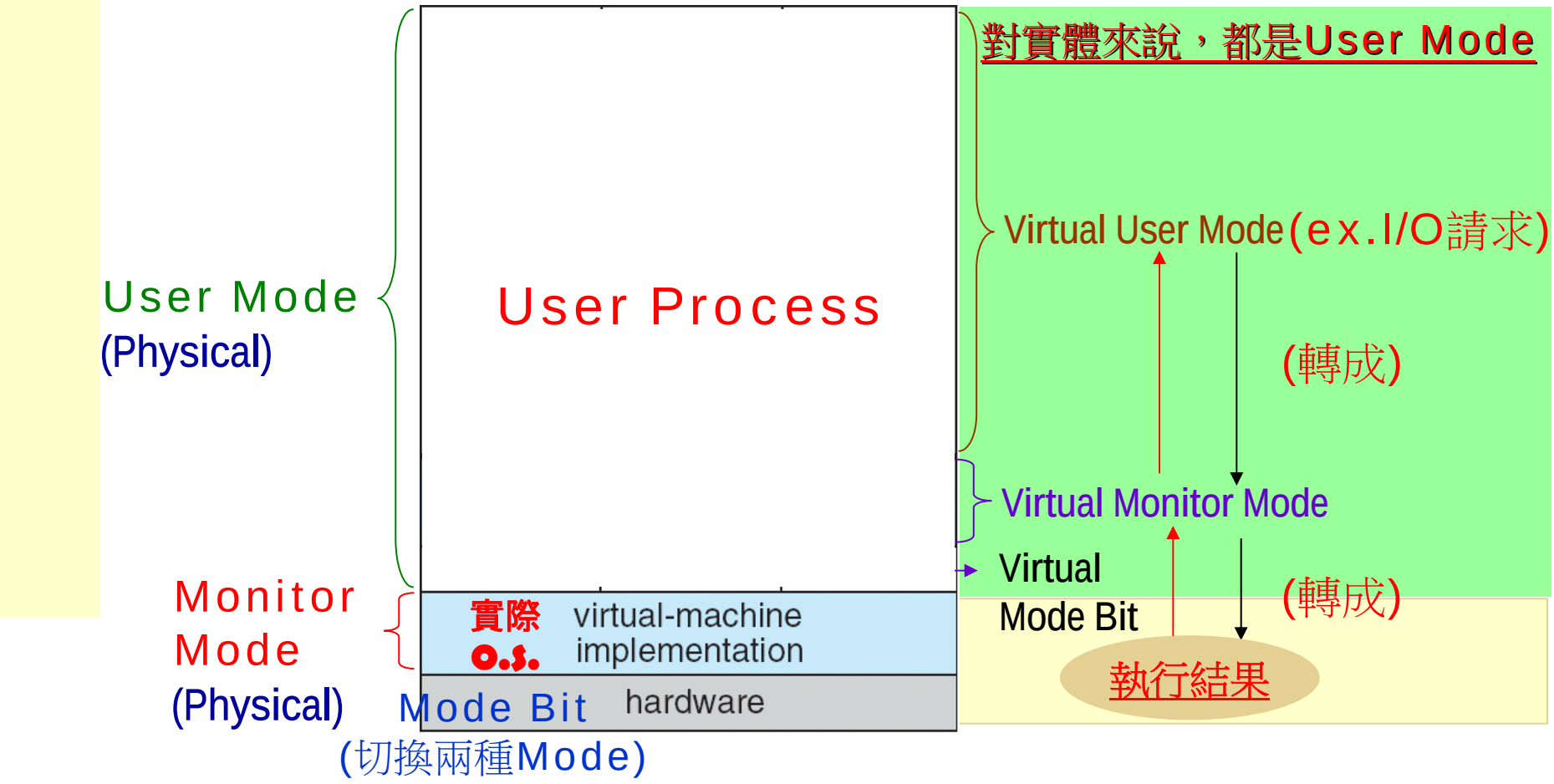
- VM的軟體是在**Monitor Mode**下執行。
- 優點:
  - 對於**O.S.的測試與開發**，是一個很好的平台工具。
    - 測試O.S.可在Virtual Machine上執行，而其它User及User Program仍可在其它的Virtual Machine上或實體機器上正常運行，不會中止。
    - 因為Virtual Machine是isolated，所以在O.S.的開發或測試過程中所引起的不當錯誤，也不會對實體系統造成危害。(O.S.的開發非常複雜)
  - Virtual Machine具有**安全性**。
  - 同一部(實體)機器上可執行**多套不同的O.S.**。
- 缺點:
  - 製作極為困難 (∵要將底層硬體的運作精確地複製是非常困難的)
  - Virtual Machine之間非常的isolated，所以不易Communicate及Resource Sharing。(需借助Virtual Message Passing的技術)

VM實例 : Virtual Box

- 目前市面上比較知名的虛擬機器, 分別是
  - VMware
  - Virtual Box
  - Virtual PC

Linux 中執行 Windows 系列的OS

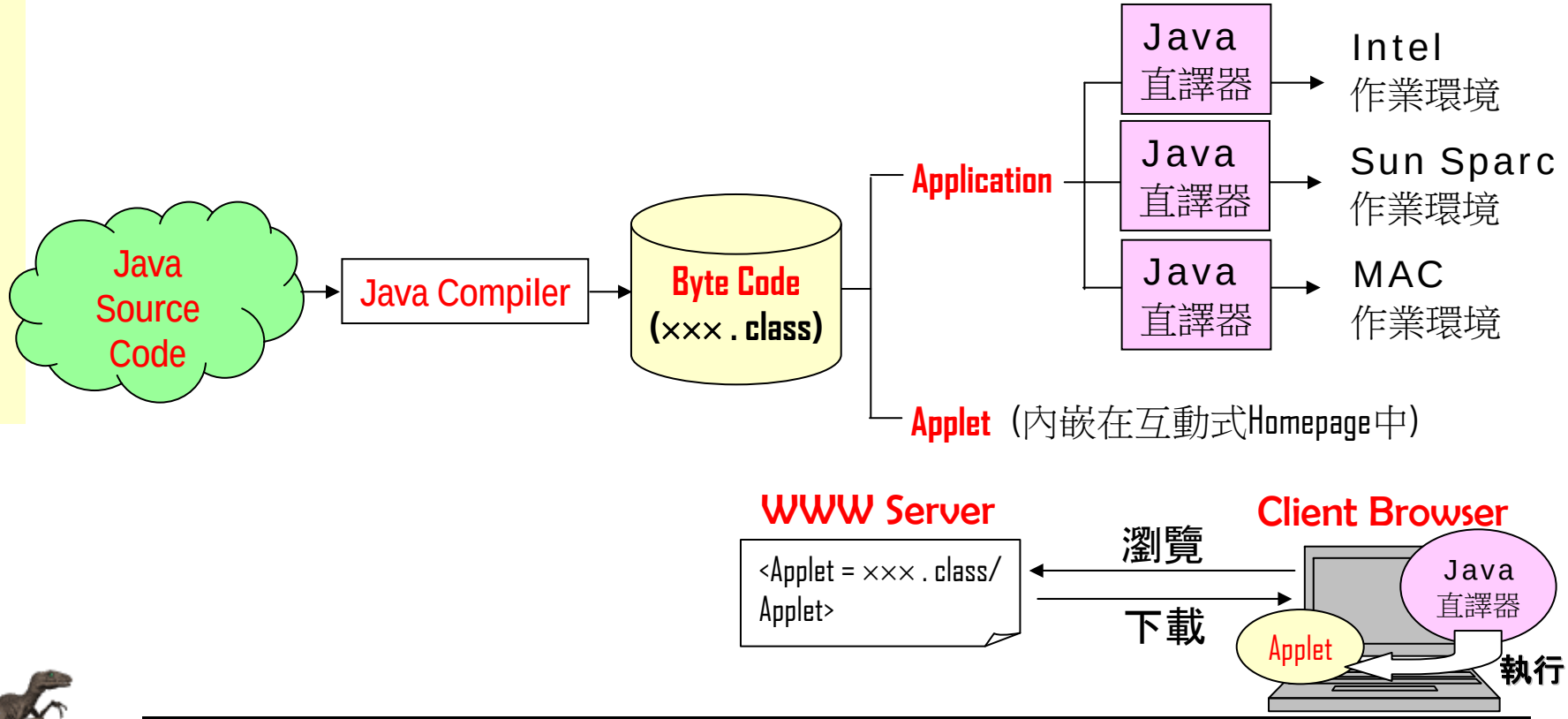
為何製作Virtual Machine極為困難?



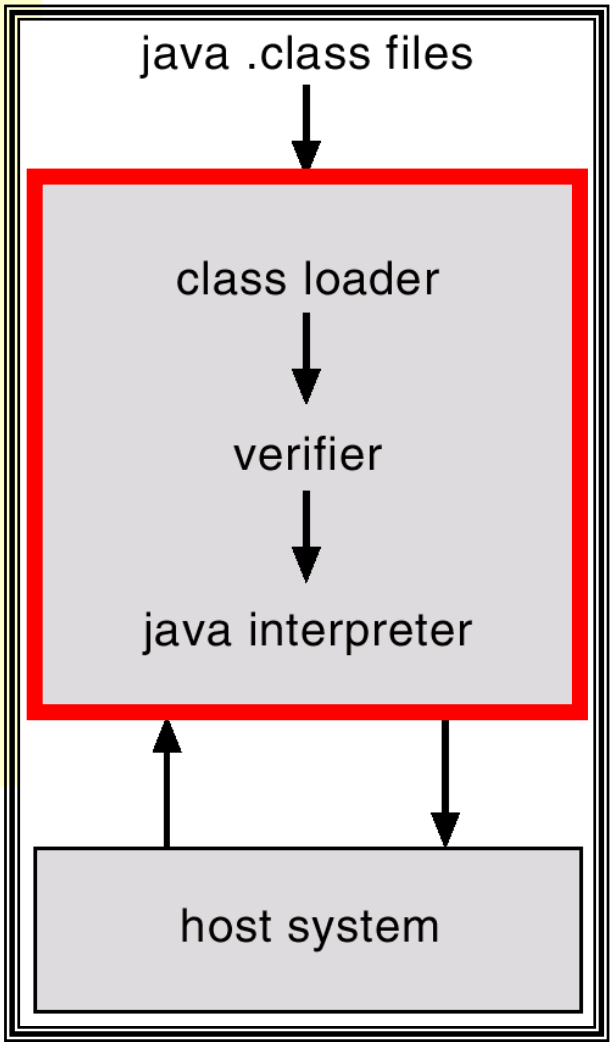
■ Java Virtual Machine (Java虛擬機器)

- Java於1995年誕生，它是由美國加州的昇陽電腦公司(Sun Microsystems, Inc.)所推出，是一種能跨平台(Cross Platform)使用、非常受歡迎的物件導向程式語言。
- 大部份的程式語言都需經過**編譯**(Compile，如：C/C++...等)或**直譯**(Interpret，如：Basic...等)才能在電腦上執行。
- 然而，Java需先經過**編譯**程序，再利用**直譯**的方式才能執行。

- 透過編譯器(Compiler)，Java程式會先被轉成**與平台無關(platform-independent)**的機器碼，稱為**位元碼(byte-codes)**。
- 透過**直譯器(Interpreter)**便可解譯為能夠在系統上執行的位元碼。
  - 翻譯一行指令，立刻Call一行硬體的指令來執行



- Byte-codes最大的好處就是**可跨平台**來執行 (Machine Independent)。
  - 前題：平台上須有Java Interpreter或支援JVM 機制
  - Java程式是先將原始碼編譯成Byte-code，再將Byte-code交由**直譯器**或是**瀏覽程式**等**各式Java虛擬機器（Java Virtual Machine；JVM）**上執行。
- Byte-code：
  - 可視為JVM的**Object Code**。
  - 具**機器獨立性(Machine Independent)**，可跨平台執行。
  - 具**高度可攜性**。



- 所謂的 Java 虛擬機器，指的是在作業系統或瀏覽器上執行的一種程式，此程式可以**解讀 Java 程式的byte-code**，並在作業系統的幫助下**執行 byte-code**。
- **JVM consists of**
  - **class loader (類別載入器)**
  - **class verifier (類別驗証器)**
    - 是否符合有效的Byte Code型式
    - Over flow
    - Under flow
    - 確保沒有指標 (point) 運算
  - **java interpreter (Java直譯器)**
- JVM藉由**Garbage Collection**自動執行記憶體回收。**(Java, Smalltalk, LISP, Ada)**

- 不論在語言本身、Byte-code的確認程序、類別載入(class loading)及記憶體等資源的管理上，Java虛擬機器都做了**相當多的防護措施與限制動作**，以確保系統安全與穩定。
- 正因為 Java虛擬機器多重的安全防護措施，也使得每個Java程式執行時需經過**重重的檢查與認證**，所需的執行時間也因而被拉長，造成**程式執行效能低落**。
- 再加上是採**直譯**的做法，翻一行做一行，沒有做最佳化的動作，故執行效率差。
- 為了提高程式執行的速度，於是發展了**Just in time compiler** (JIT compiler)取代直譯器。
  - 透過JIT compiler將Byte Code即時翻譯成Target Machine 上的Object Code，可改變Java Interpreter的效率。
  - JIT compiler的技術重點在：**減少翻譯的時間**及**產生有效率的可執行碼**。

補 充

系統呼叫的類型

行程的控制(process control)

- 正常結束, 中止執行(end, abort)
- 載入, 執行(load, execute)
- 建立行程, 終止行程(create process, terminate process)
- 獲取行程屬性, 設定行程屬性(get process attributes, set process attributes)
- 等待時間(wait for time)
- 等待事件, 顯示事件(wait event, signal event)
- 配置及釋放記憶體空間(allocate and free memory)

檔案的管理(File management)

- 建立檔案, 刪除檔案(create file, delete file)
- 開啟, 關閉(open,Close)
- 讀出, 寫入, 重定位置(read, write, reposition)
- 獲取檔案屬性, 設定檔案屬性(get file attributes, set file attributes)

裝置的管理(Device management)

- 要求裝置, 釋回裝置(request device, release device)
- 讀出, 寫入, 重定位置(read, write, reposition)
- 獲取裝置屬性, 設定裝置屬性(get device attributes, set device attributes)
- 遇輯上地加入或移去裝置(logically attach or detach devices)

資訊維護(Information maintenance)

- 取得時間或日期, 設定時間或日期(get time or date, set time or date)
- 取得系統資料, 設定系統資料(get system data, set system data)
- 取得行程、檔案或裝置的屬性(get process, file, or device attributes)
- 設定行程、檔案或裝置的屬性(set process, file, or device attributes)

通信(communication)

- 建立, 刪除通信連接(create, delete communication connection)
- 傳送, 接收訊息(send, receive messages)
- 傳輸狀況訊息(transfer status information)
- 連接或分離遠程裝置(attach or detach remote devices)