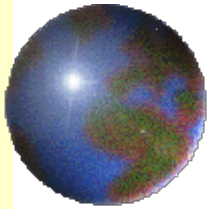
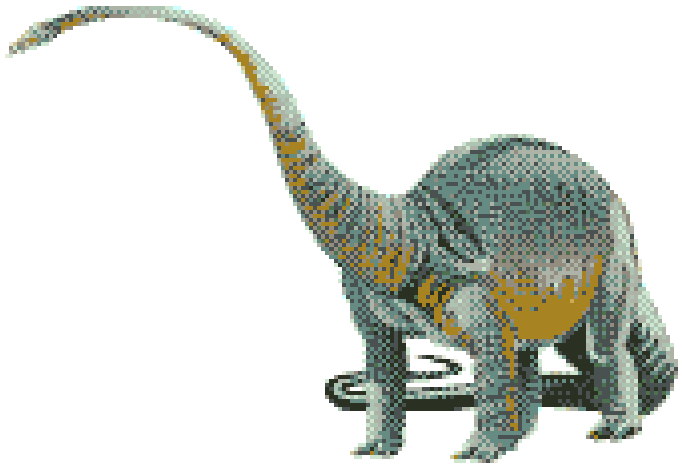


作業系統(Operating Systems)

Course 2: Computer-System Structures(電腦系統結構)

授課教師:陳士杰

國立聯合大學　資訊管理學系

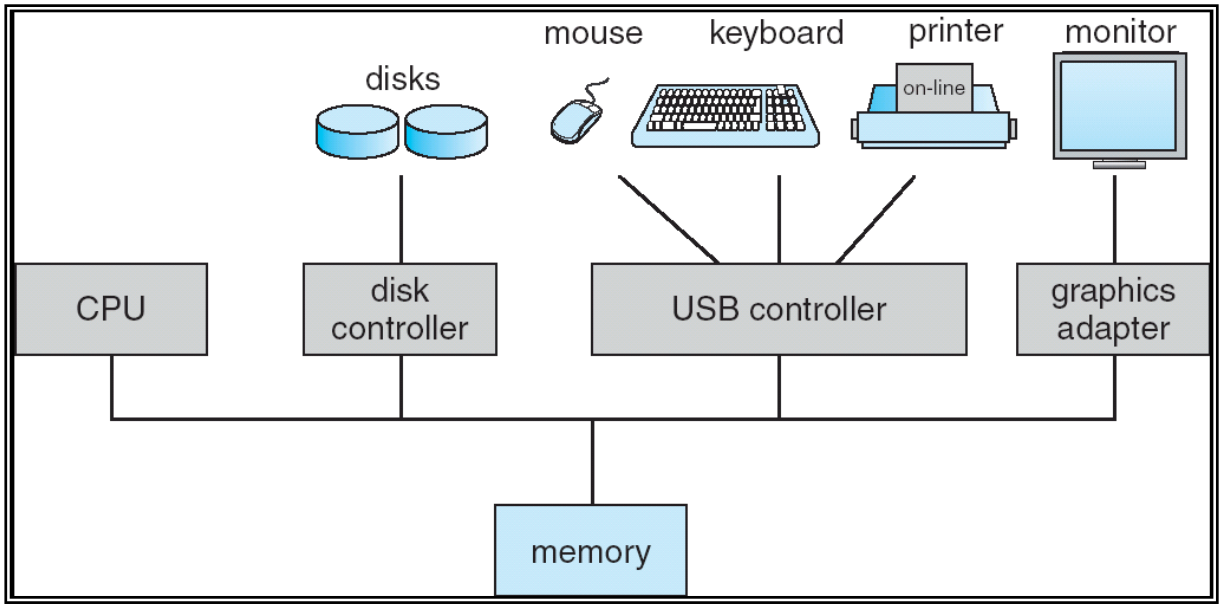


本章重點

- CPU與I/O的運作方式
- 硬體資源的保護 (Hardware Resource Protection)
  - 基礎設施 (前題): 雙模式運作 (Dual Mode)、特權指令 (Privileged Instructions)
  - I/O保護 (I/O Protection)
  - 記憶體保護 (Memory Protection)
  - CPU保護 (CPU Protection)
- Interrupt 處理程序、種類
- Interrupt vs. Trap

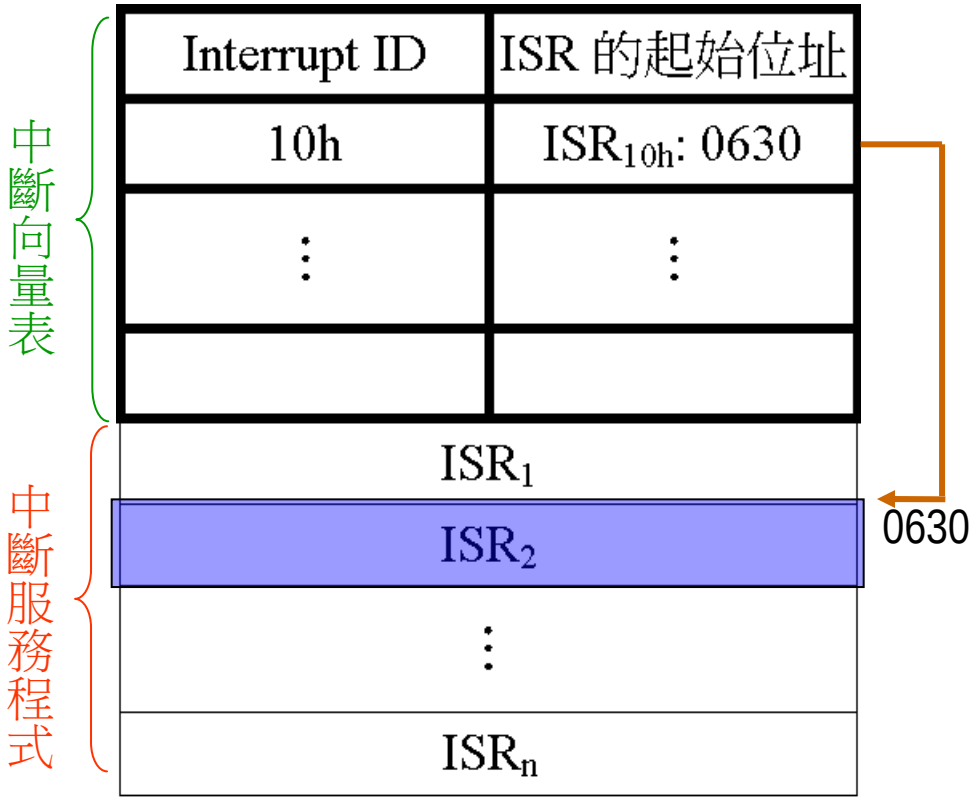
■電腦系統的操作

- 每一個**Device Controller**負責一個特定型態的裝置。
- **I/O devices 和 CPU 可以同時執行並競爭Memory。**
- 每一個**Device Controller**有自已的**local buffer (暫存器)**
- 在電腦系統中，發生一個事件時，通常是由硬體或軟體產生**中斷 (Interrupt)** 來通知。
- 近代的作業系統是**中斷驅動 (Interrupt Driven) 式**。



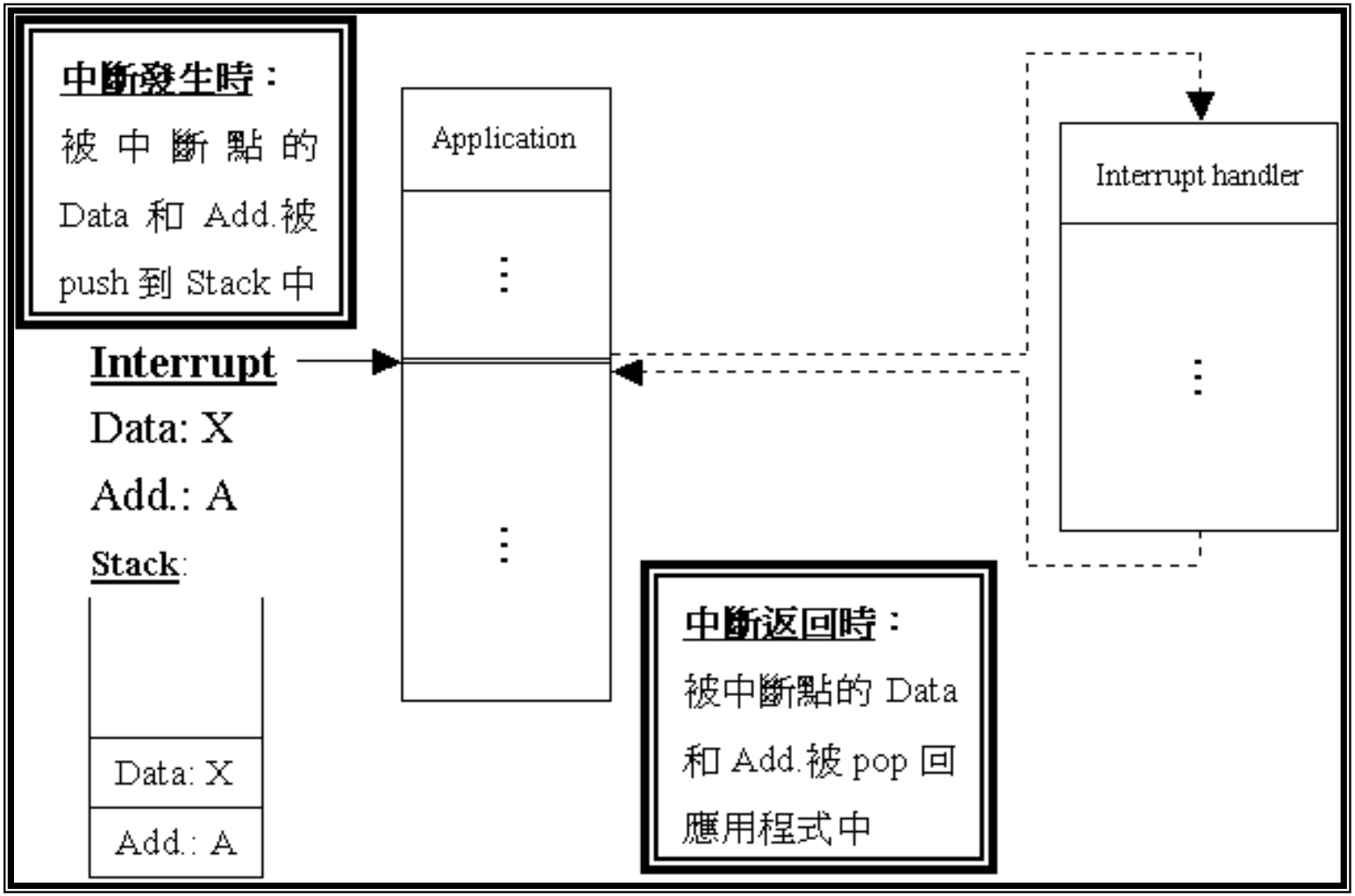
■中斷 (Interrupt)

- **中斷 (Interrupt)** 是電腦架構中一個重要的部份，每個電腦的設計均擁有自已的中斷技術。
- 在**O.S. Area**內會存在：
  - **中斷向量表 (Interrupt Vector)**
  - 一組**中斷服務程式 (Interrupt Service Routine; ISR)**



中斷發生時的處理步驟

1) O.S.要求系統 (CPU)暫停目前Process的執行，同時保存其當時執行的狀態。
2) 根據Interrupt ID去查詢Interrupt Vector，以便可以找到相對應ISR之起始位址。
3) Jump to 相對應的ISR之起始位址，系統執行ISR。
4) ISR執行完畢，將控制權交回給O.S.。
5) Resume原先中斷前的Process之執行 (原則上)。

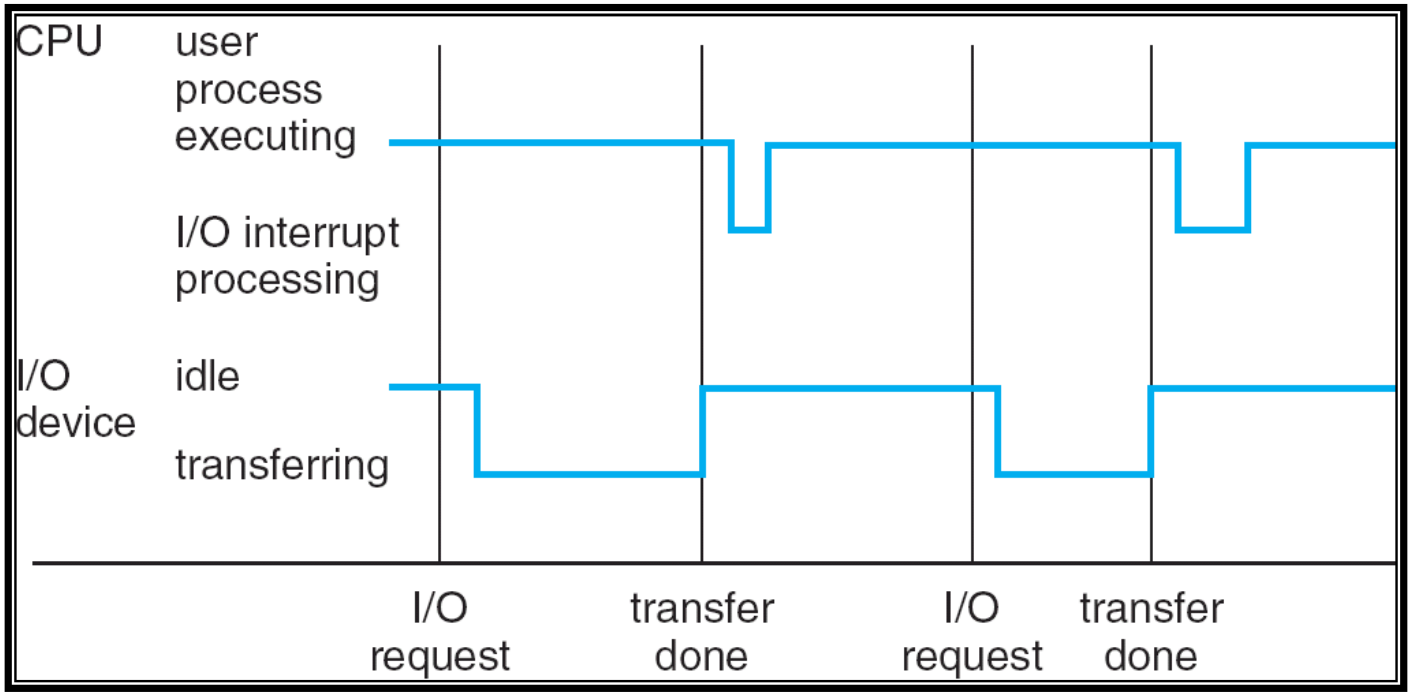

中斷發生時：
被中斷點的 Data 和 Add.被 push 到 Stack 中
Application
Interrupt handler
Interrupt
Data: X
Add.: A
Stack:
Data: X
Add.: A
中斷返回時：
被中斷點的 Data 和 Add.被 pop 回應用程式中

中斷的種類

- **External Interrupt**
  - 由CPU以外的週邊設備所引發。
    - e.g. Machine Check, I/O Complete, Device Error.
- **Internal Interrupt**
  - 由CPU本身所引發。
    - e.g. Overflow, 除以0, 執行到illegal Instruction.
- **Software Interrupt**
  - 當User Process於執行過程中，若需要O.S.提供服務時，則User Process會發出此中斷通知O.S.，提供所指定的Service，亦稱Trap (陷阱)。
    - e.g. System Call.

Interrupt v.s. Trap

- **Interrupt**
  - **Hardware-Generated Interrupt.**
    - I/O device 發出 “I/O Complete”中斷
- **Trap**
  - **Software-Generated Interrupt.**
    - User Process需要O.S.提供時會發出
    - 當有錯誤的數學運算 (Arithmetic Errors) 發生時 (e.g., 除0)

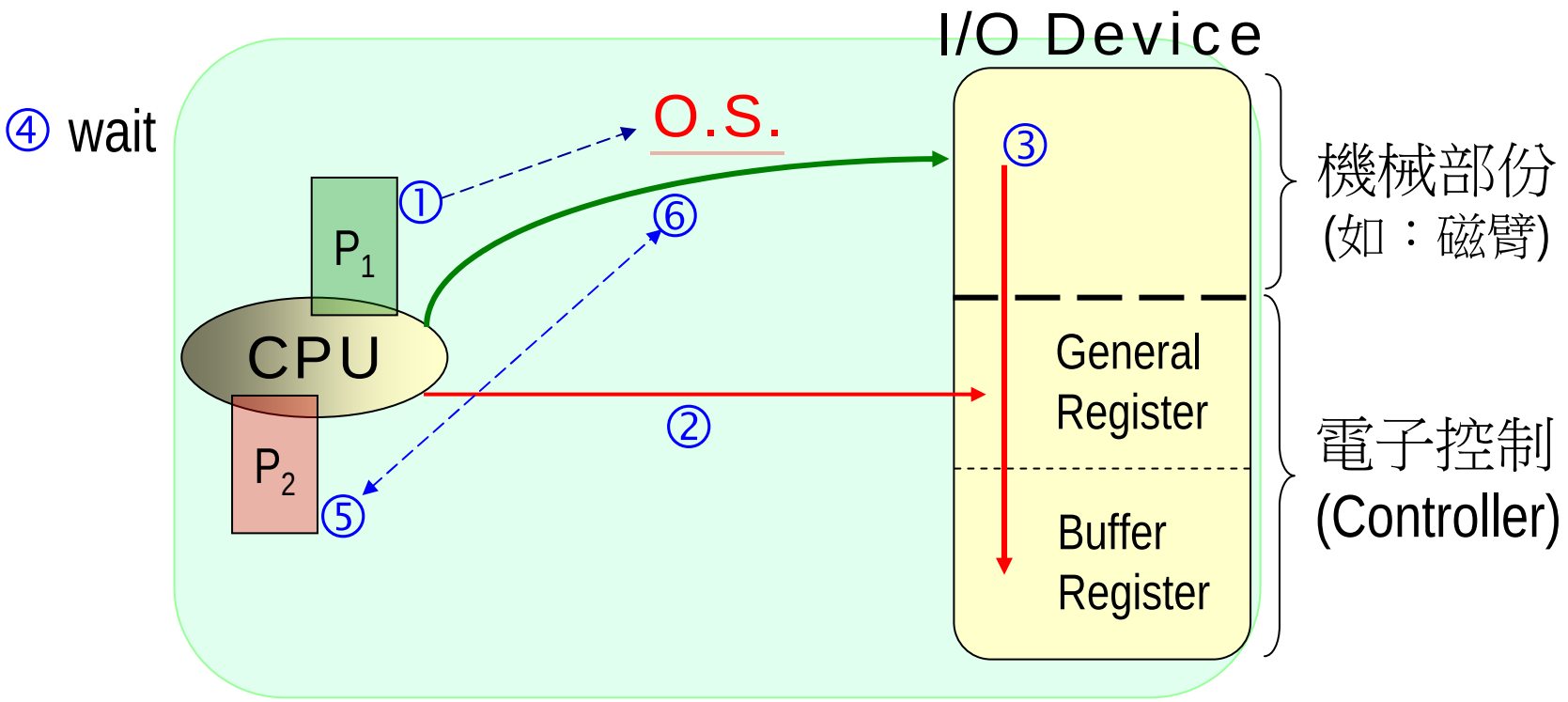

CPU
user process executing
I/O interrupt processing
I/O device
idle
transferring
I/O request
transfer done
I/O request
transfer done

■CPU與I/O的運作方式

- Polling I/O (詢問式I/O)
- Interrupt I/O (中斷式I/O)
- DMA

Polling I/O (詢問式I/O)

- 也可以稱Busy Waiting I/O、Programmed I/O。



- **General Register**: 供CPU設定I/O命令，即接收命令之用。
- **Buffer Register**: 資料在傳送時，暫存資料之用。

Polling I/O執行步驟 :

1) 工作P_1發出I/O Request (向O.S.請求)
2) 由O.S.透過CPU來設定I/O command to general register
3) I/O運作
4) P_1 wait for I/O complete
5) CPU切給P_2執行
6) CPU會不斷去詢問I/O Device其I/O運作之結果完成與否

- CPU設定完I/O Command後，CPU雖然會將其切換給其它的Process執行，然而在執行的過程中，CPU仍需**耗費大部份時間來監控I/O運作的執行**。所以CPU雖然看起來很busy，但實際對Process之執行進度助益不大。
  - ∵CPU並未全部投入在Process的執行，浪費許多Polling Time
- ∴對Process的**Throughput (產能)**無益。

Interrupt I/O (中斷式I/O)

- **Interrupt I/O執行步驟 (前①~⑤同Polling I/O):**

6) I/O完成後，Controller會發出 "I/O Complete" Interrupt通知O.S.

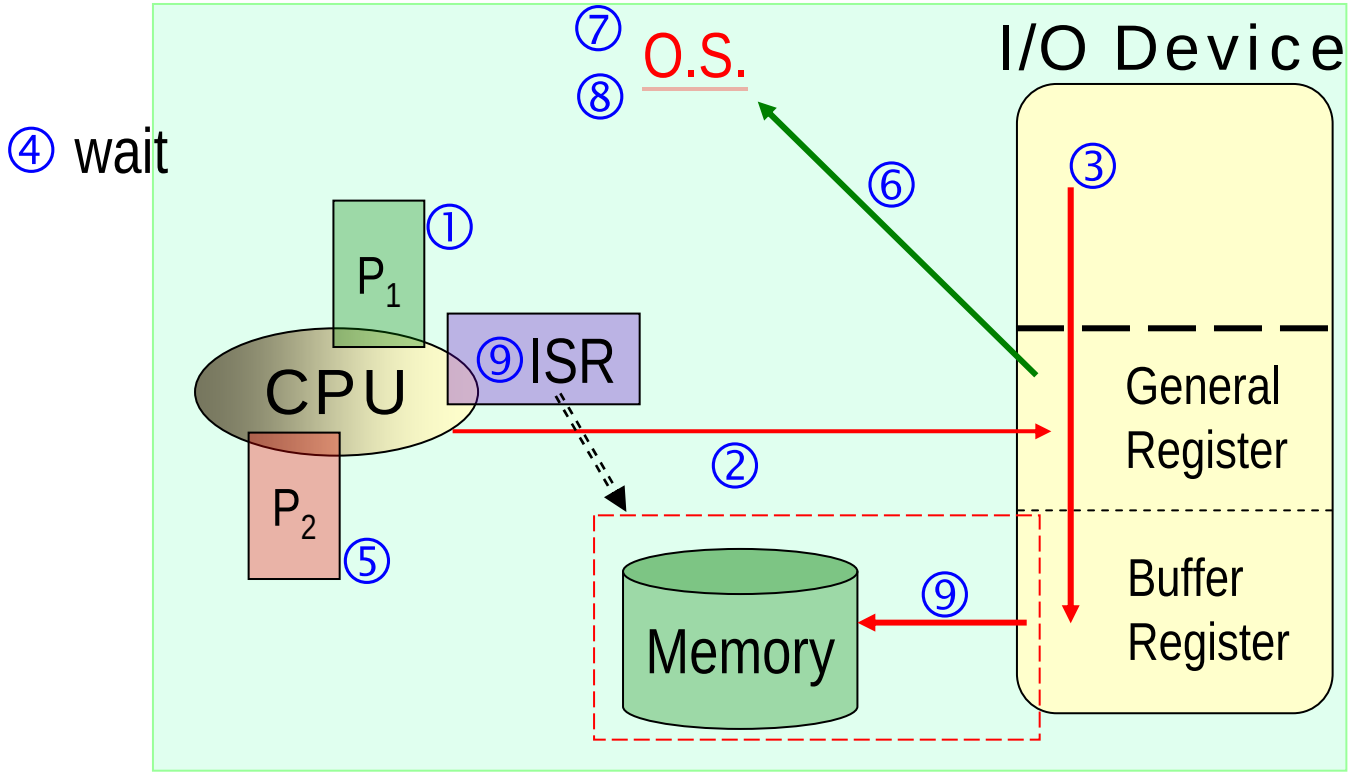
7) 暫停目前的Process (P_2)的執行 (會保留相關資料)

8) O.S.會根據Interrupt ID，查詢Interrupt Vector，取出對應的Interrupt Service Routine (ISR) 之起始位址

9) 執行相對應的ISR

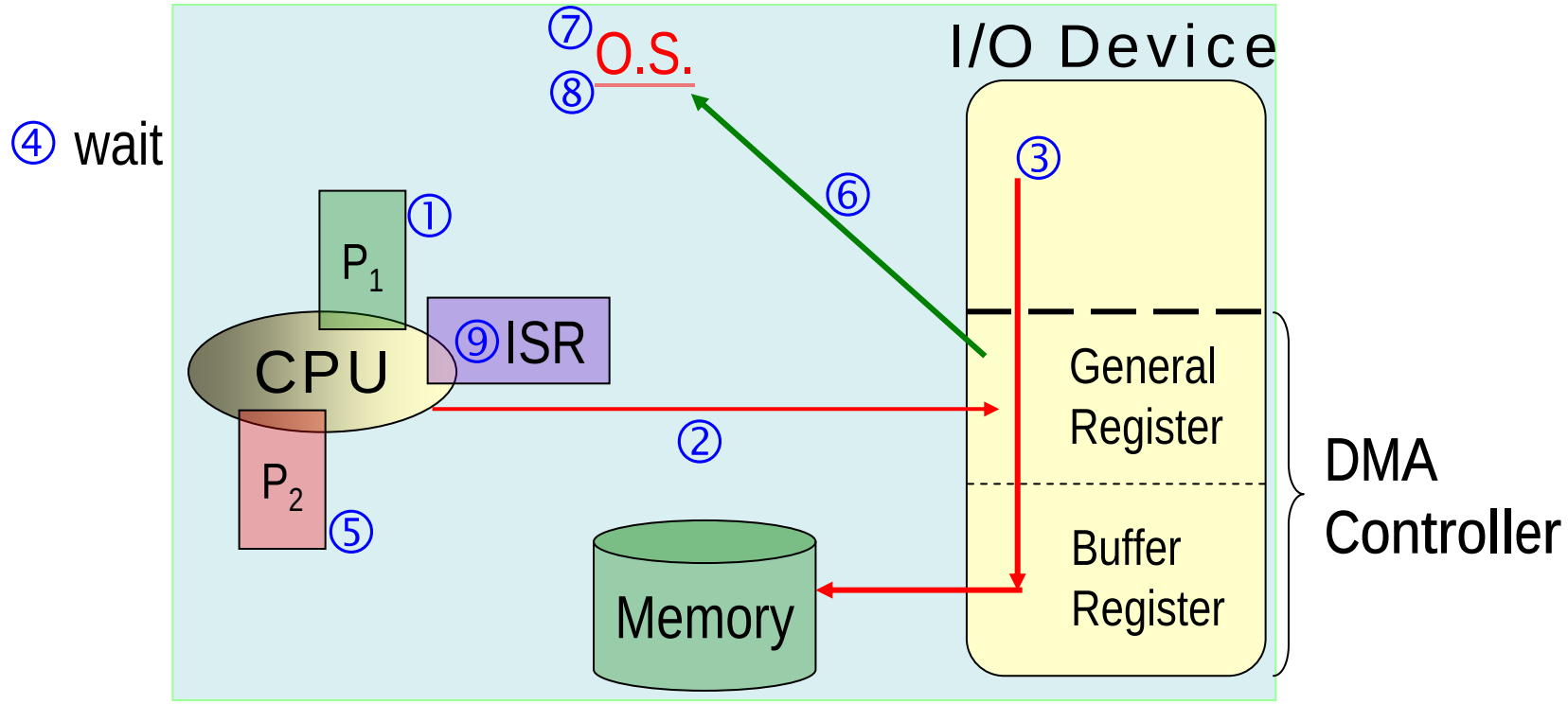
10) ISR完成後，O.S.通知P_1其I/O Request已完成 (Complete)，將P_1的Wait state改成Ready state

11) 恢復P_1的執行 (依據CPU排班法則挑選Process執行)


⑦
⑧
O.S.
I/O Device
④ wait
⑥
③
①
P1
⑨ISR
CPU
General
Register
②
P2
⑤
Buffer
Register
⑨
Memory

DMA (Direct Memory Access) 直接記憶體存取

- DMA Controller負責Memory 與I/O Device之間的資料傳輸，傳輸過程中不需CPU的參與或監督，故可再增加CPU Utilization (利用度)與Throughput (產能) (∵CPU有更多的時間投入在Process的執行上)。
- 通常用在高速的Block-Transfer I/O Device (e.g., Disk)。
- DMA也會發出中斷，只是發出的時間點與Interrupt I/O不同。

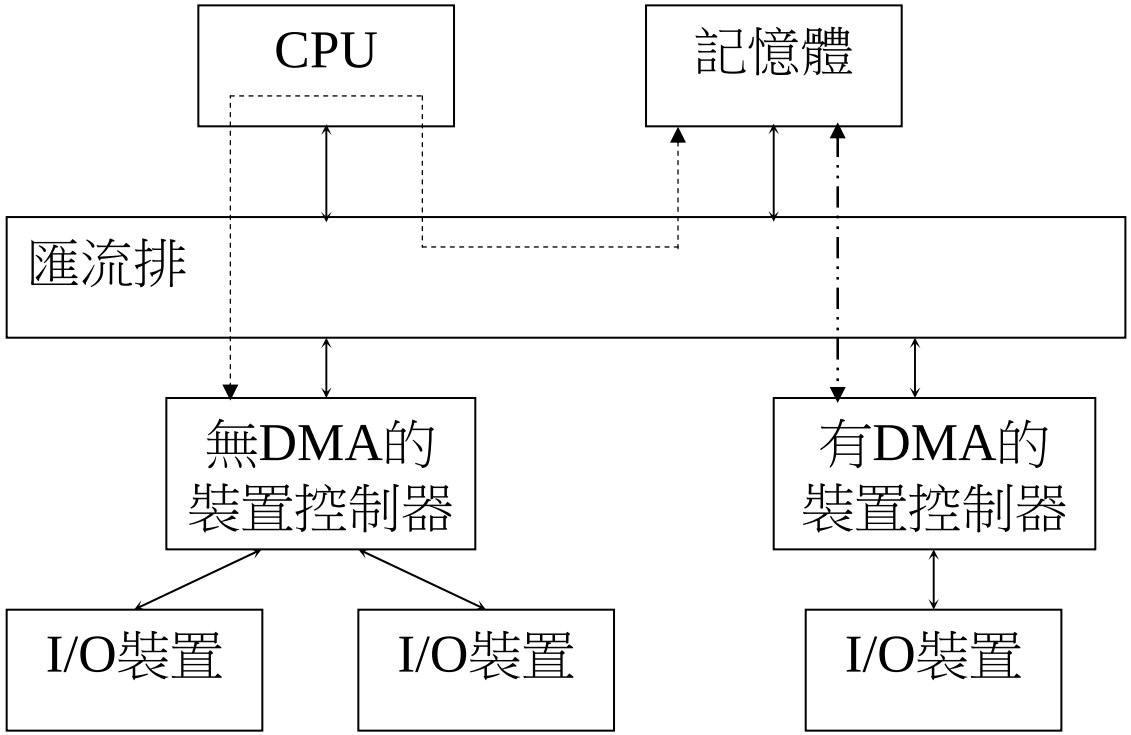


第②步只作I/O command的設定嗎?

CPU設定DMA Controller的運作有哪些

- I/O Command (e.g., r/w)
- Physical Device Location (e.g., Track, Sector)
- Memory Location
- Counter (表示 : 傳輸量的大小)
  - 傳輸量到達了之後才會發出中斷。

有無DMA Controller的區別



DMA存取路徑

無DMA存取路徑

資料可傳遞路徑

CPU與DMA之間對Memory的運作方式

- DMA與CPU之間對Memory都是以Interleaving (交替)的方式運行。此時所採用的技術稱為 "Cycle Stealing" (週期偷取)。

What is Cycle Stealing?

機器指令的執行週期

- 機器指令的執行週期有5個Stages:
  - IF: Instruction Fetch
  - DE: Decode
  - FO: Fetch Operand
  - EX: Execution
  - WM: Write Result to Memory

- 哪些Stage會做Memory Access:
  - 一定會 : IF Stage
  - 不一定會 : FO Stage, WM Stage

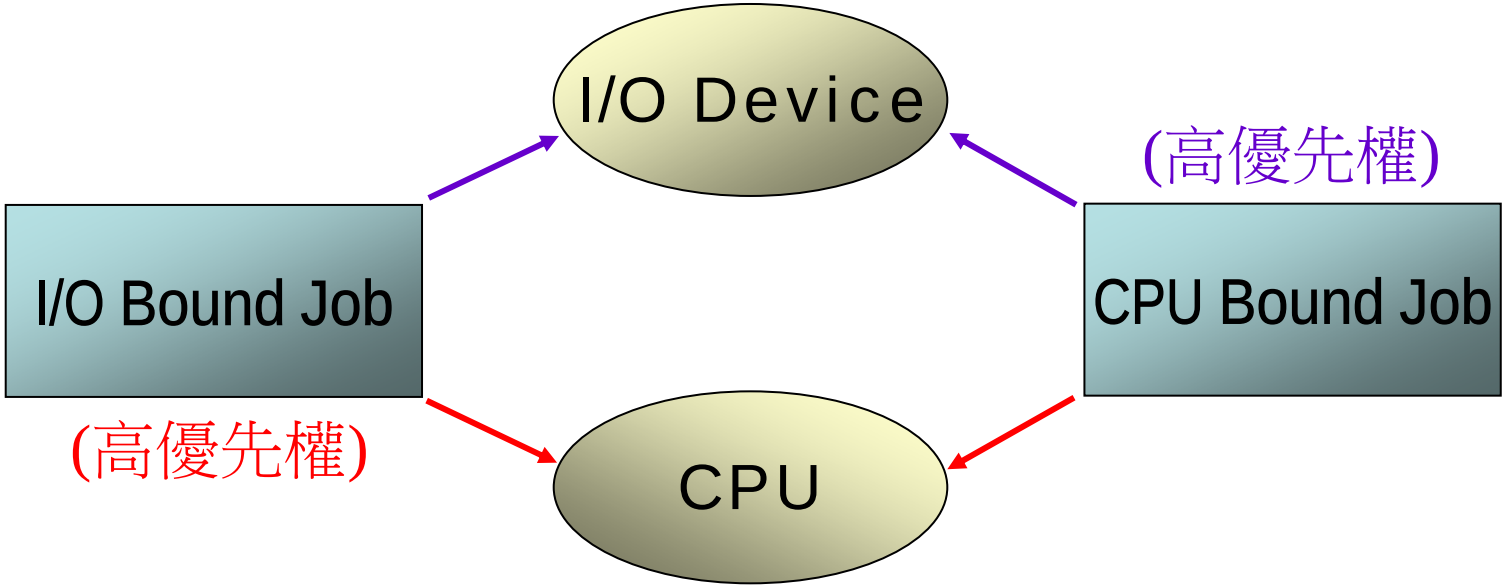
- **FO與WM這兩個Stage，CPU有可能不會去對Memory進行存取，DMA Controller可以去steal (偷) 這兩個Stage的時間進行Memory與I/O Devices 之間的資料傳輸，且不會影響到對Memory的存取與指令的執行。**

CPU若與DMA發生Memory Conflict (衝突) 時如何處置?

CPU與DMA發生Memory Conflict時之處理

- 萬一CPU與DMA Controller同時欲爭取Memory的使用而發生Memory Conflict時，此時O.S.會給予**DMA Controller**較高的優先權。
  - 由資源服務要求較少的行程優先服務，可以得到較少的平均等待時間 (由O.S.來判斷)。
    - 因為CPU一直在RUN，所以**CPU對Memory的需求量很高**。
    - 而**DMA對Memory的需求量較少**，因為它只在有需要時才執行。

分配資源通則



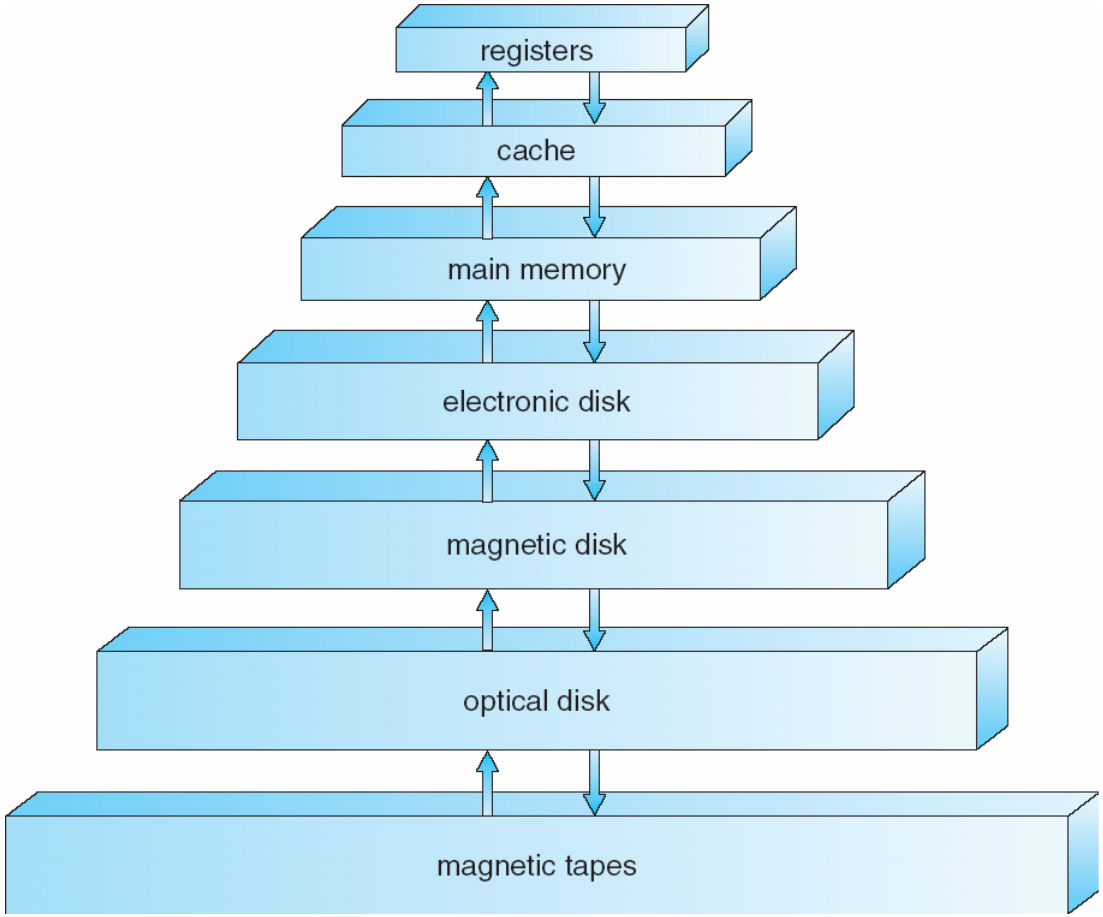
- **例外**:
  - 以**Real Time System** 來說，**CPU要給CPU Bound Job較高的優先權。**

■儲存體結構 (Storage Structure)

- 主記憶體 (Main Memory)
  - CPU唯一能夠直接存取的外接儲存區域。
  - 容量小，具揮發性 (Volatile)。
- 輔助儲存體 (Secondary Storage)
  - 為主記憶體的擴展，具非揮發性 (Nonvolatile) 的大容量記憶體。

儲存體裝置的階層 (Storage Hierarchy)

- 速度
- 成本
- 容量
- 揮發性



- 為何要談Storage Hierarchy?
  - 兼顧Access Speed與Cost之平衡。

暫存器 (Storage Hierarchy)

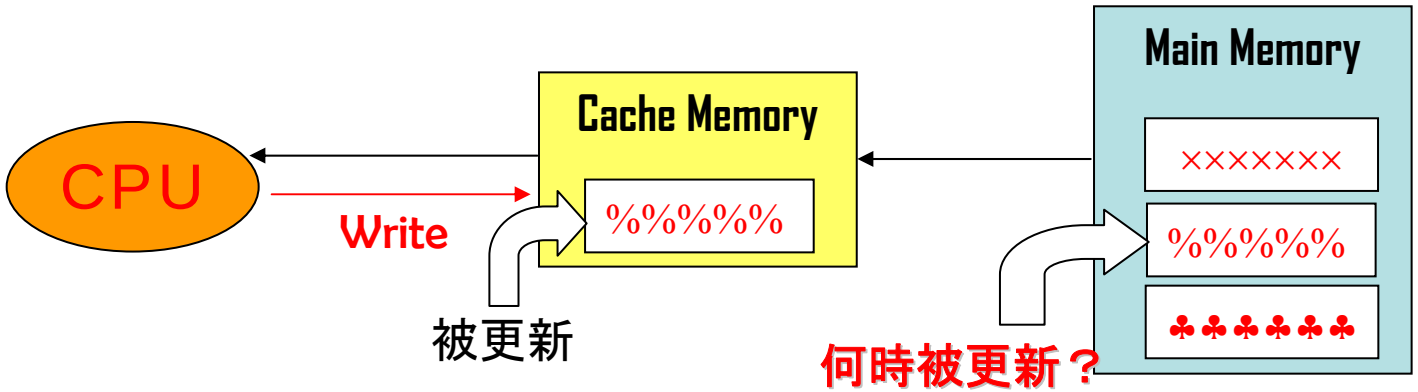
- Program Counter (PC; 程式計數器)
  - 暫存下一條指令所在之Memory Address
- Instruction Register (IR; 指令暫存器)
  - 暫存由Memory中取出的指令，以便待會供Control Unit 解碼
- Memory Data Register (MDR; 記憶體資料暫存器)
  - 暫存欲取出或存入Memory的data或intermediate results (中間結果)之用
- Memory Address Register (MAR; 記憶體位址暫存器)
  - 暫存欲取出或存入Memory的data或intermediate results (中間結果)所在之Memory Address
- Process Status Word (PSW; 行程狀態字元)
  - 記錄ALU執行指令後的狀態 (ex: 是正或負, 是否為0, 有無overflow, ...)
- Base Register/Limit Register (基底暫存器/限制暫存器)
  - Base Register: 記錄Program執行的起始位置
  - Limit Register: 記錄Program所需記憶體大小

快取記憶體 (Cache Memory)

- 目的：改善CPU對主記憶體之存取速度。
- 作法：
  - 將記憶體中經常被存取的區域之內容置於Cache Memory中。CPU未來存取指令或資料時，會先到Cache Memory中尋找。若Hit(命中)，則無須到Memory中存取；否則才會到Memory中存取。
- 若Cache Memory Hit Ratio高，則Performance佳。
  - 快取記憶體的設計是用來節省資料收尋的時間。然而，快取記憶體的容量加倍，並不代表速度也會加倍，是跟資料命中率有很大的關係，也就是有多少次數能準確找出資料。
- 在硬體系統中，快取記憶體分為兩種：
  - 內建在CPU中的L1快取；在CPU之外 (在主機板)，稱為L2快取
  - L1快取比L2快取記憶體快，所以CPU找尋資料時，會先從L1快取尋找，找不到再從L2記憶體尋找，最後才到主記憶體裡 (RAM) 尋找。

Cache Memory之更新策略

- **Write Through**
  - **Def:** 一旦**Cache**內容更新，立刻寫回**Memory**
  - 優點：**Cache**與主記憶體內容一致
  - 缺點：耗時，喪失使用**Cache memory**之好處 (尤其是**Write**指令很頻繁時)
- **Write Back**
  - **Def:** 當**Cache** 內容要被置換出去時，才將更改後的內容寫回主記憶體中
  - 優點：節省時間
  - 缺點：**Cache**與主記憶體內容可能不一致



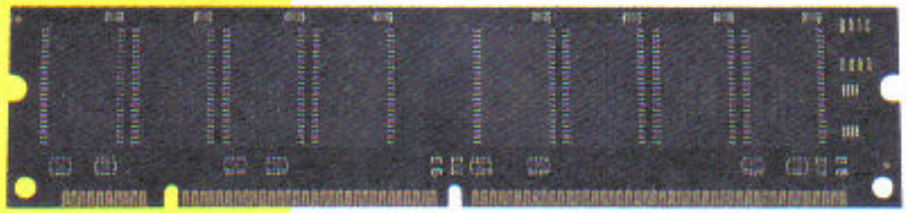
- 在多重處理器 (Multiprocessor) 的環境下，由於不同的行程與處理器都能夠並行地執行同一個工作，如果有一個正在處理的資料被更新，則更新後的資料一致性問題則是一個重大的考驗。而這個問題在分散式 (Distributed) 作業環境下更是一個常見的問題。
- 由於快取記憶體的容量有限制，所以快取記憶體管理的策略是必要 (第9章)

主記憶體 (Main Memory)

- 主要可區分成下列兩種:

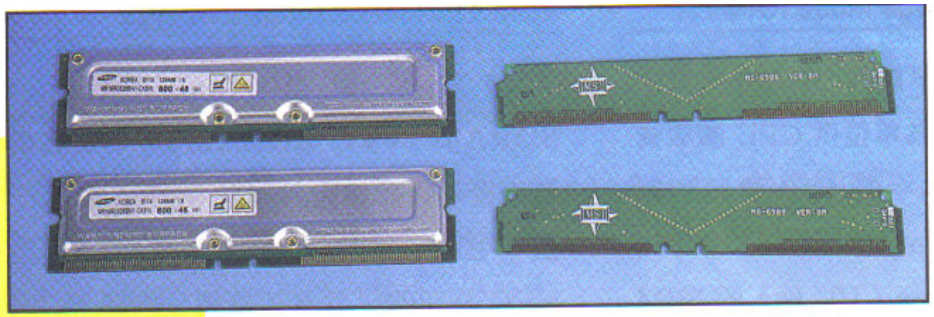
RAM	ROM
Random Access Memory	**Read Only Memory**
資料可隨意存取	資料只能唯讀，不能重寫入
Power off，資料隨即消失 (揮發性)	**Power off**，資料仍存在 (非揮發性)
容量大	容量小

- 市面上的**RAM**有：



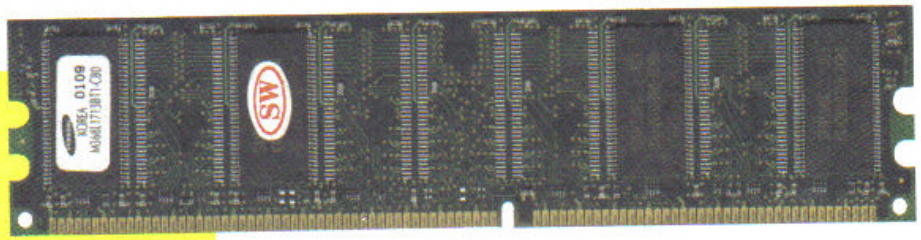
3-9 168pin DIMM RAM Model

Synchronize DRAM (SDRAM)



3-11 RAMBUS DRAM

Direct Rambus DRAM



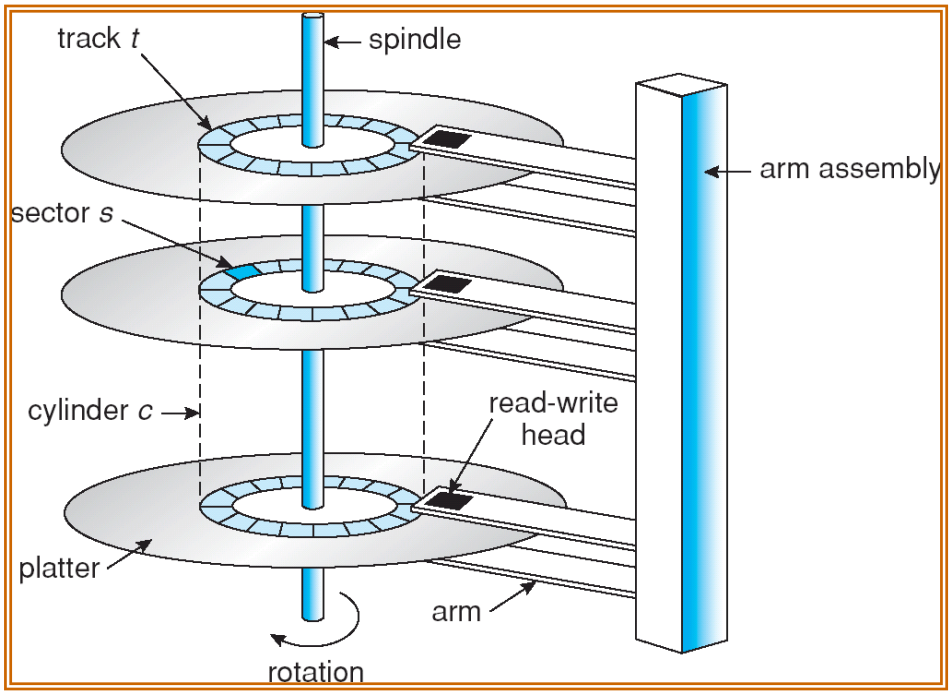
3-12 DDR SDRAM

Double Data Rate SDRAM (DDR SDRAM)

- ROM可分成：
  - **PROM (Programmable ROM)**：只允許使用者寫入一次資料，即無法更改。
  - **EPROM (Erasable PROM)**：允許使用者多次寫入，可用紫外線照射使其內部資料消失。
  - **EEPROM (Electrically EPROM)**：利用電流的方式消除資料。
  - **Flash Memory or Flash ROM**：具低耗電性，可透過**BIOS**廠商提供的更新程式來更改資料，凡支援隨插即用的電腦都必須使用**Flash Memory**來儲存**BIOS**。

磁碟 (Magnetic disks)

- 磁碟的每個**Disk**，都被邏輯性地劃分成數個**磁軌(Tracks)**，而每個磁軌又被劃分出數個**磁區(sectors)**。
- 磁碟裝置和電腦間資料傳送是由**磁碟控制器**來決定。要完成一個磁碟的**IO**動作，電腦利用訊息的方式將命令傳送到磁碟控制器中，由磁碟控制器操作磁碟機的硬體，以完成這個命令。
- 在磁碟控制器中，通常會有一個內附的暫存器，磁碟機的資料傳送是將資料從磁碟表面移到暫存器中，再由暫存器將資料傳送到電腦。



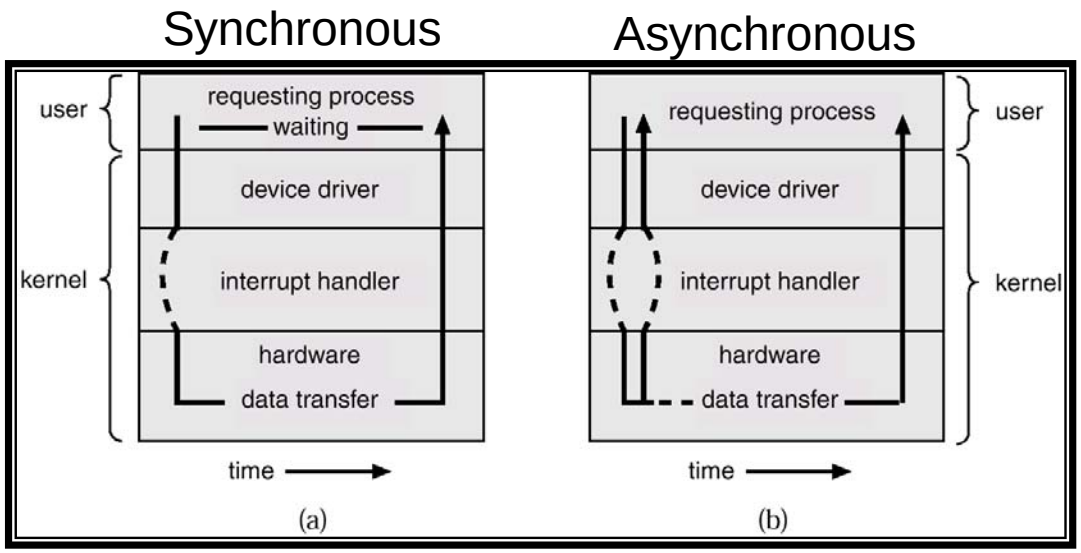
磁碟存取時間 (Disk Access Time)

- 磁碟存取時間 (Disk Access Time)是由下列三個時間的加總：
  - **尋找時間 (Seek Time)**: 將Read/Write Head移到指定Track上方所花的時間。
  - **旋轉潛伏時間 (Rotational Latency Time)**: 將欲Access的Sector轉到Read/Write Head下方所花的時間。
  - **傳輸時間 (Transfer Time)**: 資料在Disk及Memory之間的傳輸時間。
- 上述三者以**Seek Time** 耗時最多(∵是機器動作)。

Level	1	2	3	4
Name	registers	cache	main memory	disk storage
Typical size	< 1 KB	> 16 MB	> 16 GB	> 100 GB
Implementation technology	custom memory with multiple ports, CMOS	on-chip or off-chip CMOS SRAM	CMOS DRAM	magnetic disk
Access time (ns)	0.25 – 0.5	0.5 – 25	80 – 250	5,000.000
Bandwidth (MB/sec)	20,000 – 100,000	5000 – 10,000	1000 – 5000	20 – 150
Managed by	compiler	hardware	operating system	operating system
Backed by	cache	main memory	disk	CD or tape

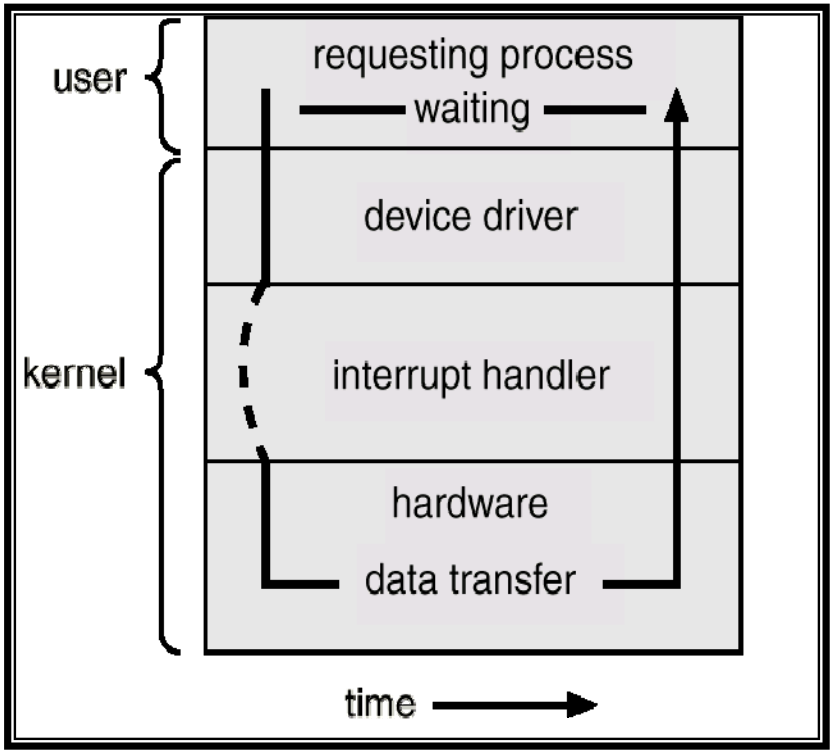
■I/O結構 (I/O Structure)

- 當User Process發出I/O Request 後，系統的控制權多久會交還給User Process?
  - 即：**User Process 可不可以繼續往下執行?**
- 有兩種架構:
  - **Synchronous I/O structure (同步I/O架構)**
  - **Asynchronous I/O structure (非同步I/O架構)**



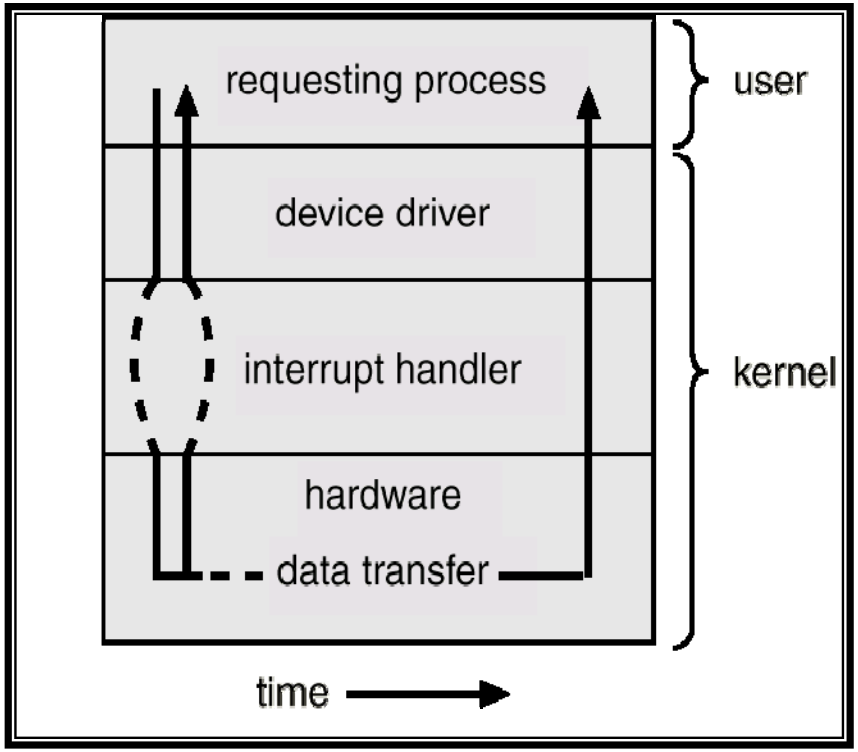
同步I/O結構 (Synchronous I/O Structure)

- 是當I/O完成運作後才會將控制權交還給User Process。
- Process可藉由Busy Waiting Loop或特殊的Wait指令來達到等待的目的。
- 討論:
  - 優點:在一段時間內只有一個I/O請求產生。∴當I/O Complete 中斷產生時, O.S.即知是由何種Device發出。
  - 缺點:不支援並行的I/O處理。

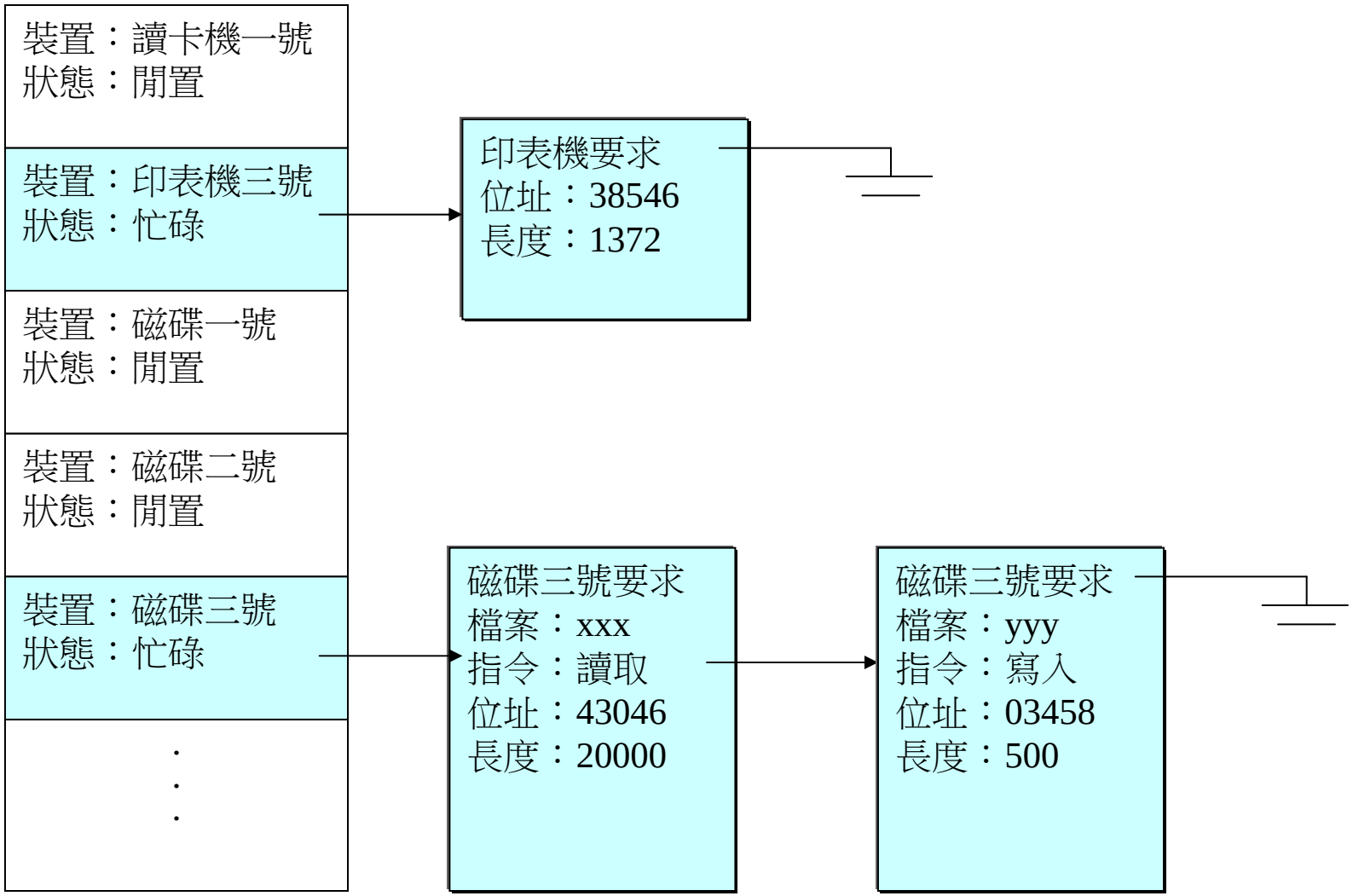


非同步I/O結構 (Asynchronous I/O structure)

- 立刻將控制權交還給User Process，不需空等I/O運作完成。
- 在一段時間內可有多個I/O Request同時發生。
- O.S.必須有一個 "Device Status Table"以記錄各種Device的位址、使用狀況和位於某Device的I/O請求之執行狀況。



裝置狀態表 (Device-Status Table)



■硬體保護 (Hardware Protection)

- 基礎設施 (前題):
  - 雙模式運作 (Dual-Mode Operation)
  - 特權指令 (Privileged Instruction)
- 三種硬體保護
  - I/O 保護 (I/O Protection)
    - 防止User Program直接使用I/O設備
  - 記憶體保護 (Memory Protection)
    - 防止User Program誤用記憶體空間 (如:系統區域、其它Program所在區域)
  - CPU保護 (CPU Protection)
    - 防止User Program無限期佔用CPU

實施Protection的前題

- 系統必須提供**Dual-Mode運作**。
- 必須將會引起系統危害的指令，設定為**特權指令 (Privileged Instruction)**。

雙模式運作 (Dual-Mode Operation)

- 系統運作的狀態可分為兩種：
  - **Monitor Mode (監督模式)**
    - O.S.的System Process可以執行的狀態。在此模式下，O.S.掌控對系統的控制權，只有O.S.的工作才可運行，User Program不允許在此Mode下運作。
    - 又稱 **Supervisor Mode**、**System Mode** 或是 **Kernel Mode**
    - 在此Mode下，才有權利執行特權指令 (Privileged Instruction)
  - **User Mode (使用者模式)**
    - User Program 在此模式下允許被執行，即User Program可執行時的系統狀態
    - 在此模式下，不能執行特權指令，否則會引起 "**illegal instruction error**"，產生錯誤中斷 (Trap)，O.S.會強迫Process中止

硬體對Dual-Mode製作的支援

- Dual-Mode需要硬體的額外支援，即提供一個**模式位元 (Mode Bit)** 用以區分此兩種模式，通常
  - 0: 表示Monitor Mode
  - 1: 表示User Mode

Dual-Mode的目的

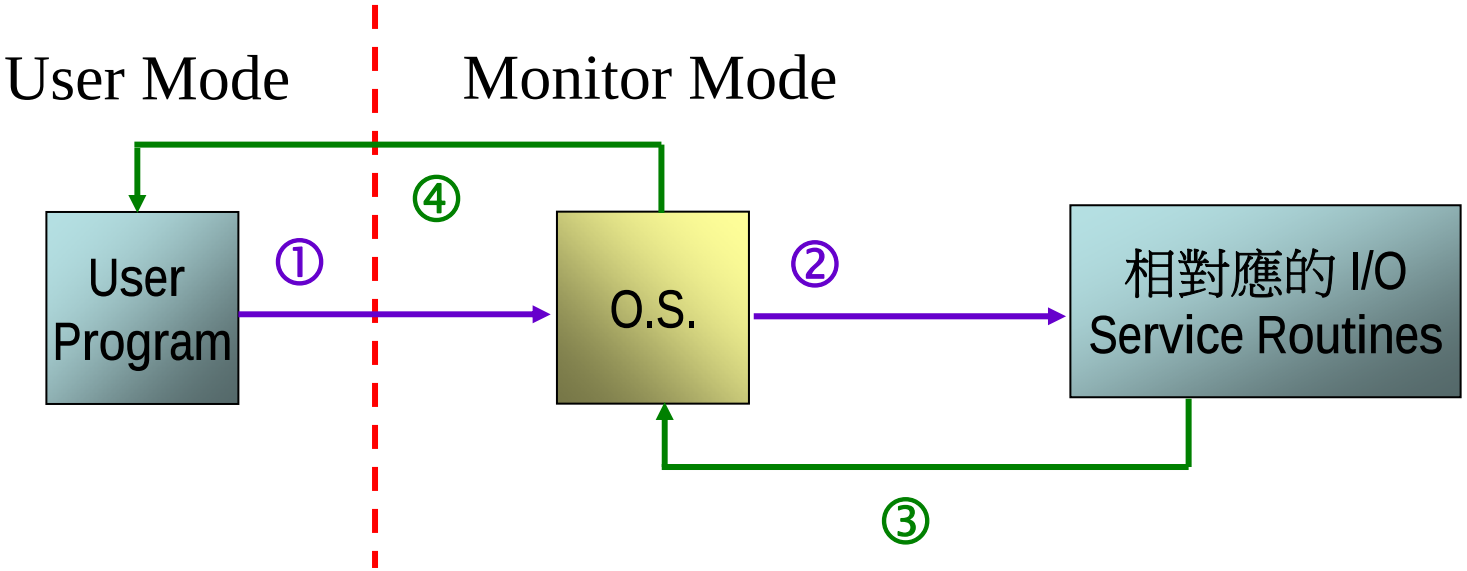
- 對重要的 H.W. Resource 實施 Protection，將可能會引起系統危害的指令設定成特權指令。
- 由於User在User Mode下無法執行特權指令，故可防止User Program執行這些特權指令，對系統或是其它User Program造成危害。

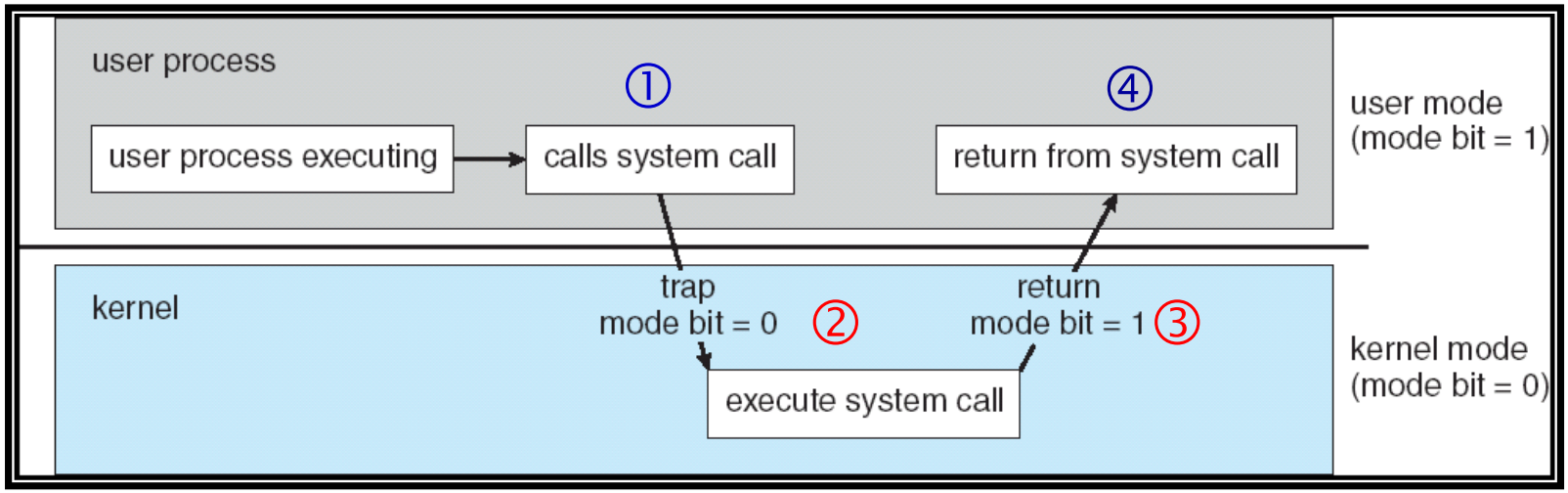
特權指令的種類

- I/O指令
- 與記憶體管理有關的暫存器之修改指令
- 與Timer設定有關的指令
- Enable/Disable Interrupt 指令
- 系統停止 (Halt) 指令
- 從使用者模式改變到監督模式的指令

■I/O保護 (I/O Protection)

- 防止User Program在執行時，直接使用I/O Device。User Program必須透過O.S.提出I/O Request，再由O.S.控制I/O運作，並將I/O Result告知User Program。
- 執行流程：
  - 發出 I/O Request，即System Call，會伴隨一些Trap，以轉換Modes
  - 執行相對應的I/O服務
  - I/O回傳結果給O.S.
  - O.S.再將結果回傳給User Program

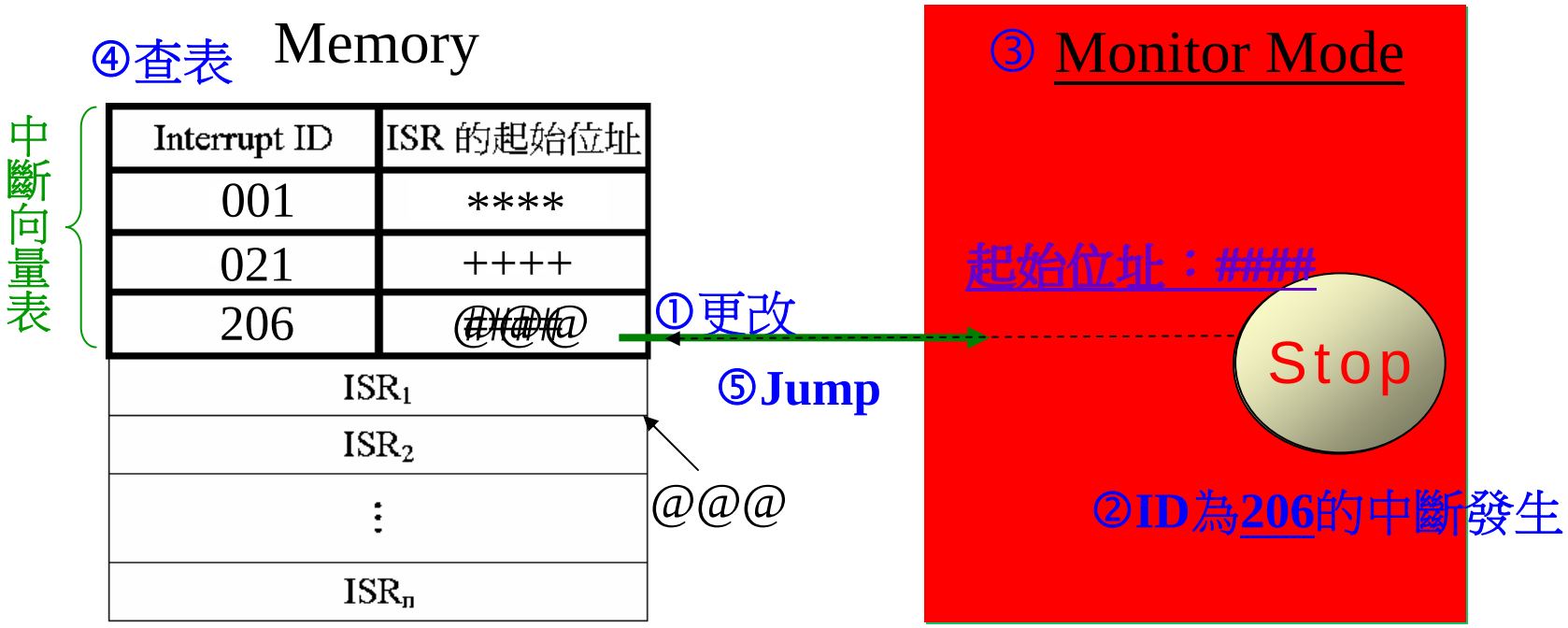

User Mode
Monitor Mode
User Program
O.S.
相對應的 I/O Service Routines
①
②
③
④


user process
user process executing
calls system call
return from system call
user mode
(mode bit = 1)
kernel
trap
mode bit = 0
return
mode bit = 1
execute system call
kernel mode
(mode bit = 0)

I/O保護的作法

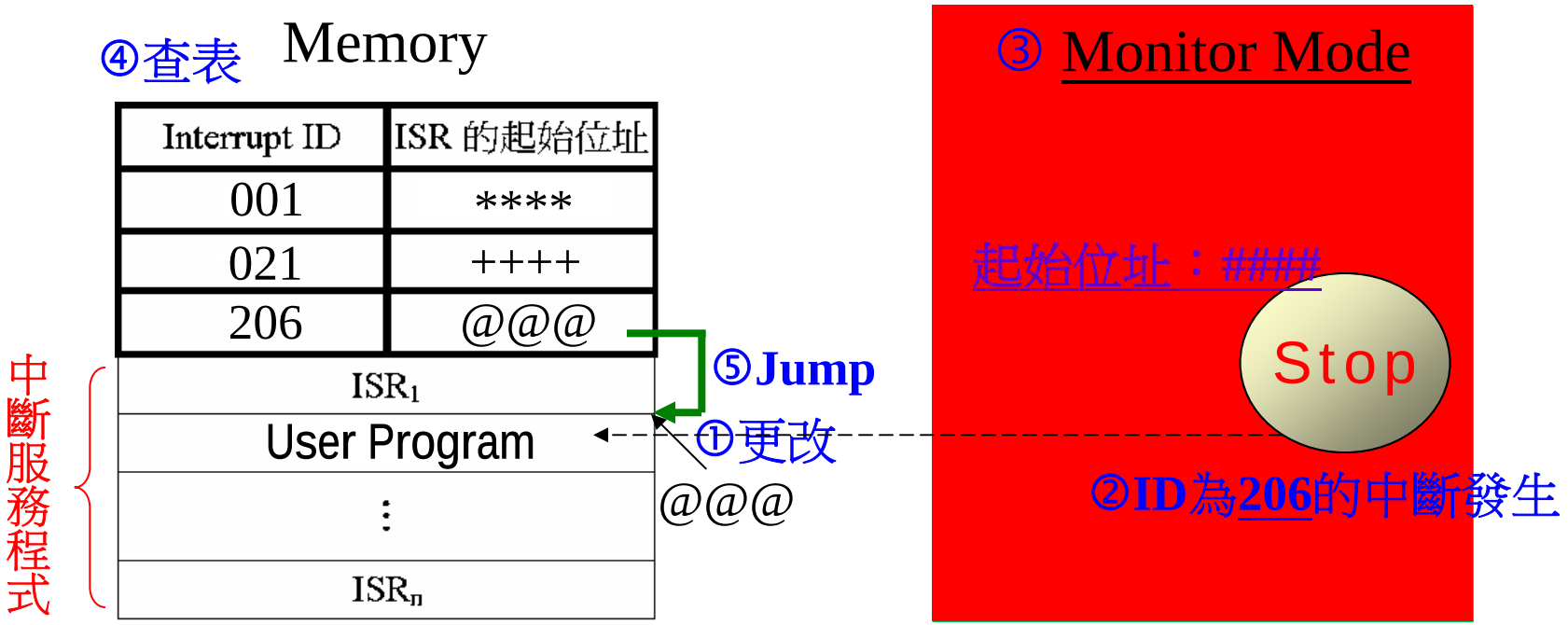
- 必須靠下列機制配合實現:
  - 系統必須**提供Dual Mode運作**。
  - 所有**I/O指令**皆設為**特權指令**。
  - 必須保護**Interrupt Vector**及**ISR所在的Memory Area**不被User Program更改，以防止User Program在Monitor Mode下，取得對系統的控制權。

若User Program可更改Interrupt Vector



(此時User Program在Monitor Mode 下執行)

若User Program可更改ISR



(此時User Program亦可在Monitor Mode 下執行)

■記憶體保護 (Memory Protection)

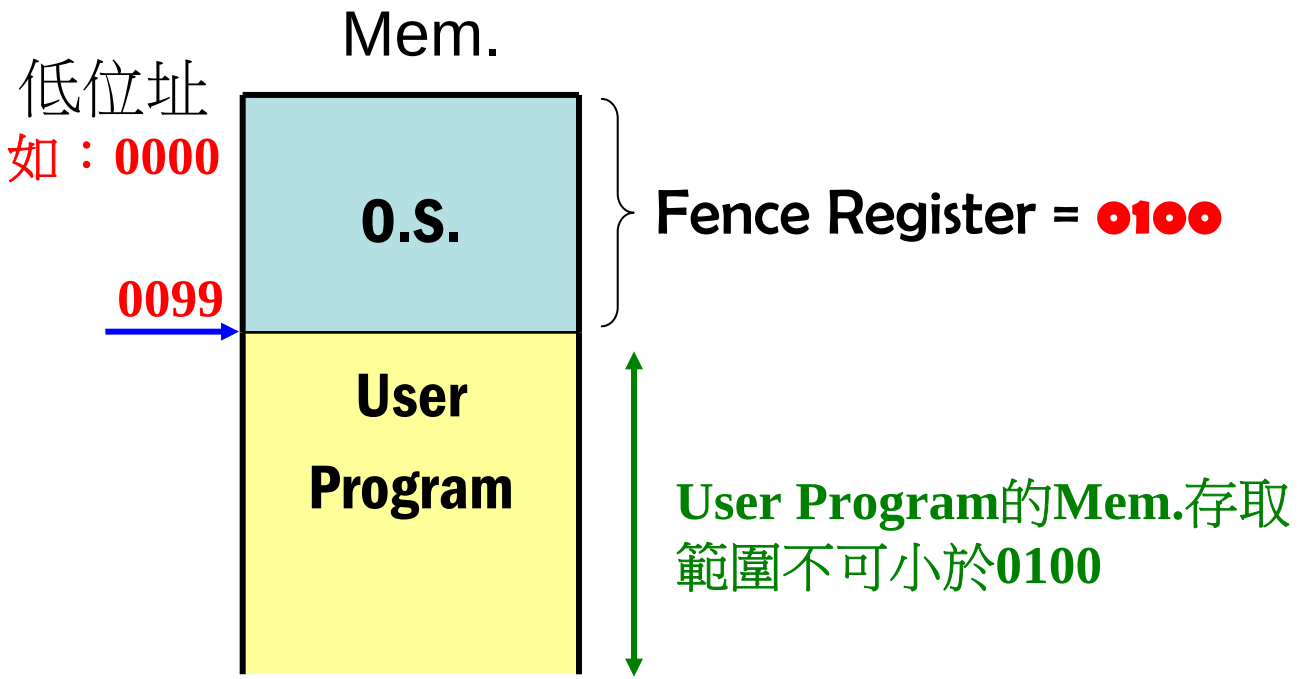
- 保護執行中的程式(或O.S.)不會被其它程式所干擾。
  - **Monitor Area的保護**:
    - 保護Monitor (O.S.) 所在的區域不被User Program修改。
  - **各個User Area的保護**:
    - 對各個User Program所在的Memory Area進行保護，以防止User Program之間企圖更改其它User Program所在的Memory Area。

Monitor Area保護的作法

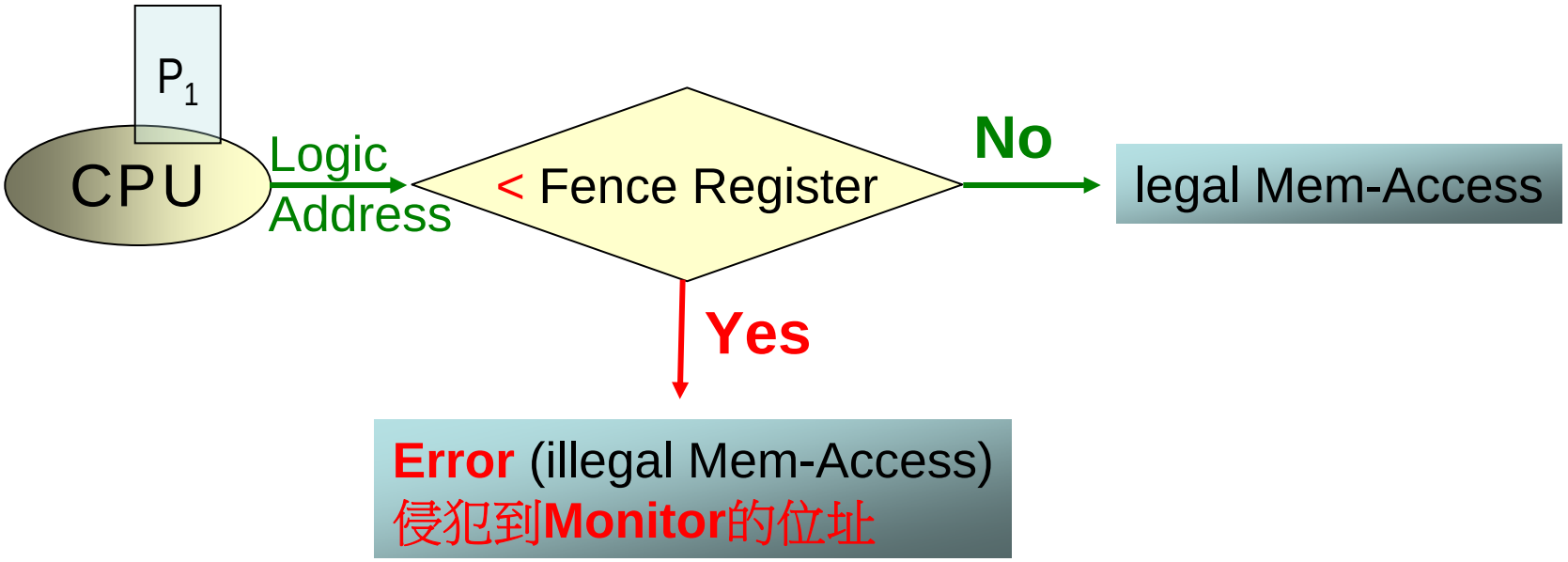
- 有兩種作法:
  - 使用**Fence Register**
  - 讓Monitor Area與User Area往**相反方向增長**

方法一:使用Fence Register

- **Fence Register (界限暫存器)**是用來記錄**O.S.的大小 (Limit)**。



- **User Program執行時，對於欲存取的Memory Address做下列的檢查流程：**



- 需將**會修改或設定Fence Register的指令**設為**特權指令**。

缺點

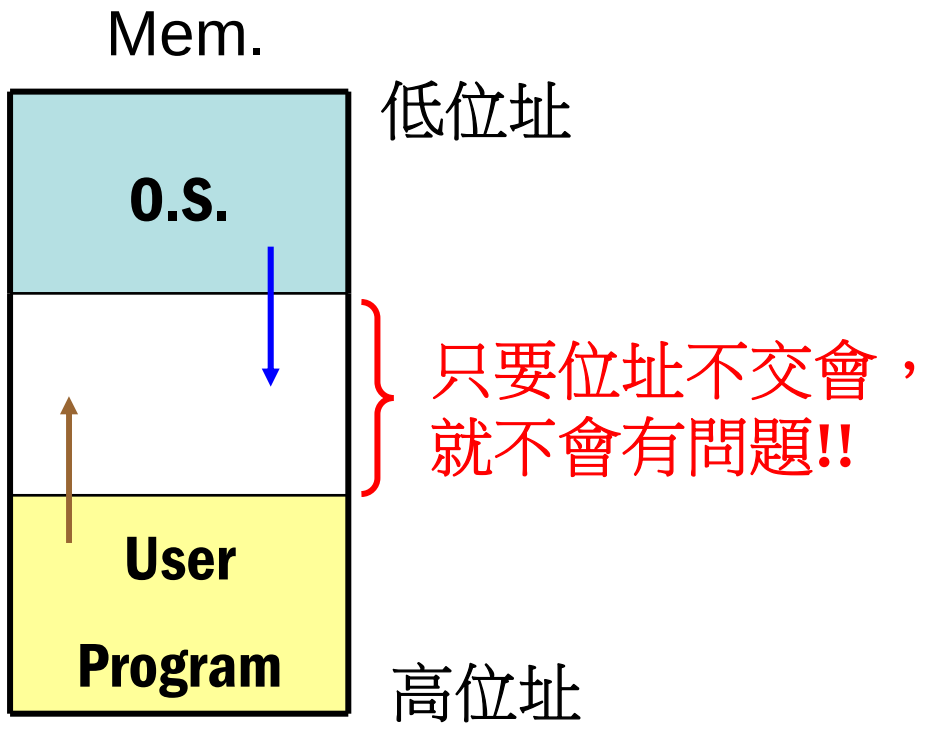
- 若O.S.所在的區域大小動態改變，會造成User Process 的執行起始位置有所改變，此時所有的User Program皆須重新定址。
- 需靠Dynamic Binding支援，且Execution Performance變差。(Ch. 8)

為什麼O.S.所在的區域大小會改變?

- O.S.通常有許多的**錯誤處理程序**，但不會一次就全部載入到記憶體中。
- 當有一個錯誤發生，而O.S.在記憶體中查不到對應的處理程序時，就會利用**Dynamic Loading**的方式(即：Load on Call) 從Disk中載入相對應的處理程序，所以O.S.的大小會改變。

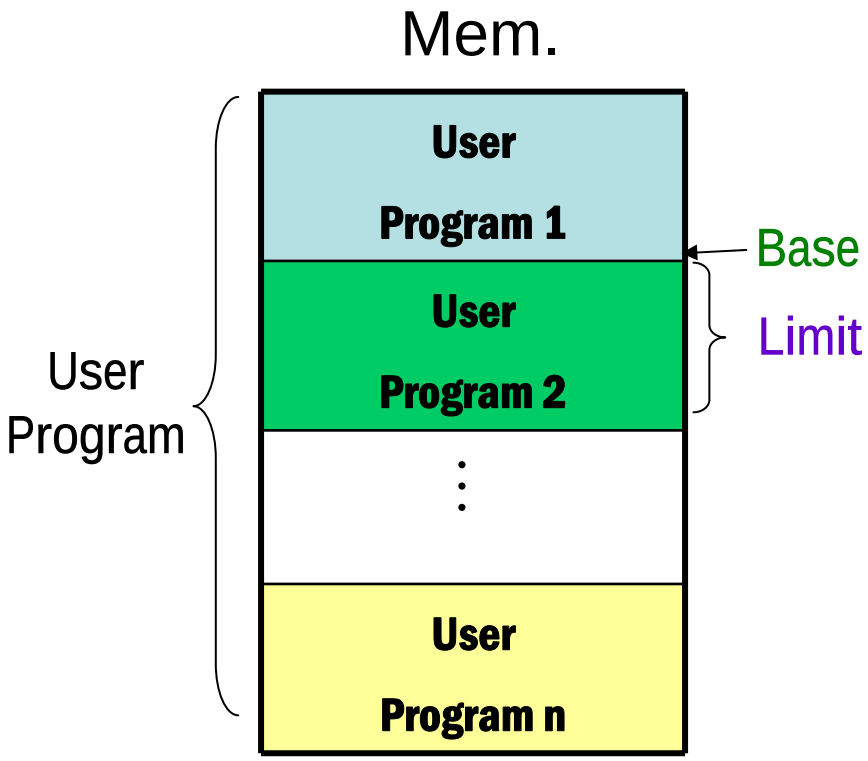
方法二:Monitor Area與User Area往反向增長

- **Monitor Area:**由低往高擺。
- **User Area:**由高往低擺。

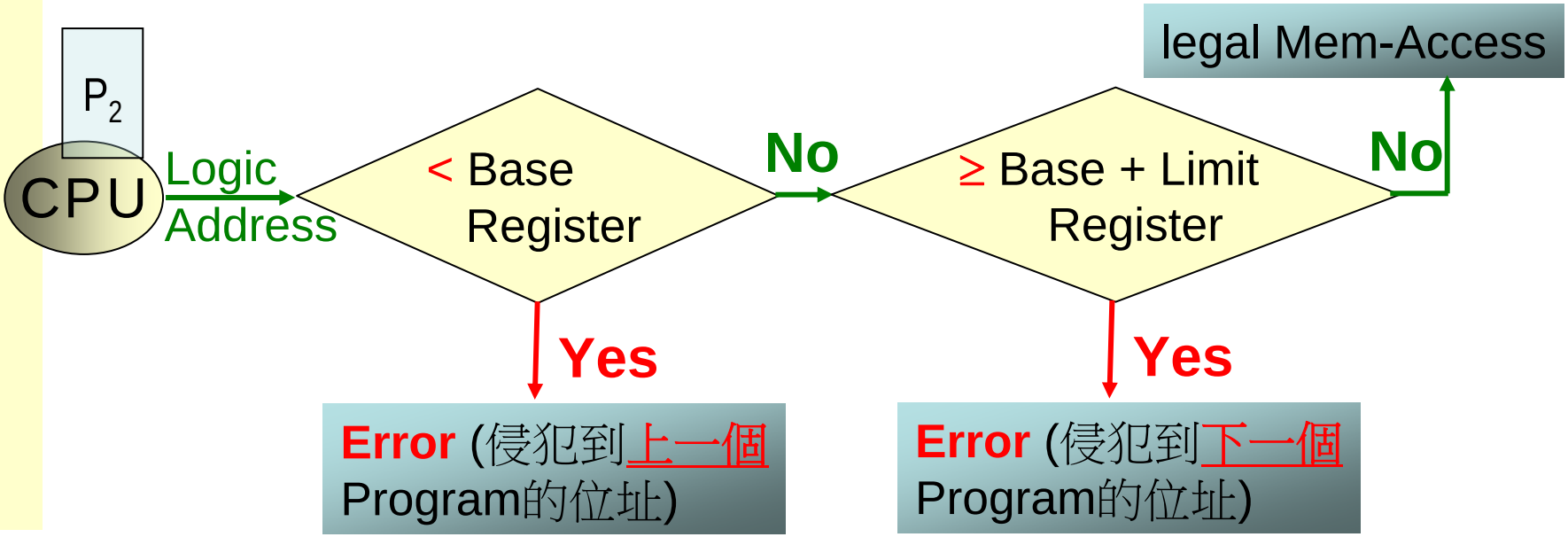


各個User Area的保護

- 使用一套Register來實作:
  - Base Register (基底暫存器)
  - Limit Register (限制暫存器)
- Base Register用來記錄User Program的起始位址。
- Limit Register用來記錄User Program的大小。



- **User Program執行時，對於欲存取的Memory Address做下列的檢查流程：**



- 需將**會修改或設定Base/Limit Register的指令**設為**特權指令**。

■CPU保護 (CPU Protection)

- 防止User Program長期 (甚至無窮) 霸佔CPU而不釋放之狀況。
- O.S.會規定出一個User Program使用CPU time的合理最大值。一旦Process獲取CPU控制權時，會將此值設定給Timer (計時器)，隨著Process的執行，Timer值會逐漸遞減直到Timer值為0。此時，Timer會發出一個 "Time Out" 的Interrupt通知O.S.，O.S.此時可以強迫Process放棄CPU。
- Timer值之設定更改指令須設為特權指令。

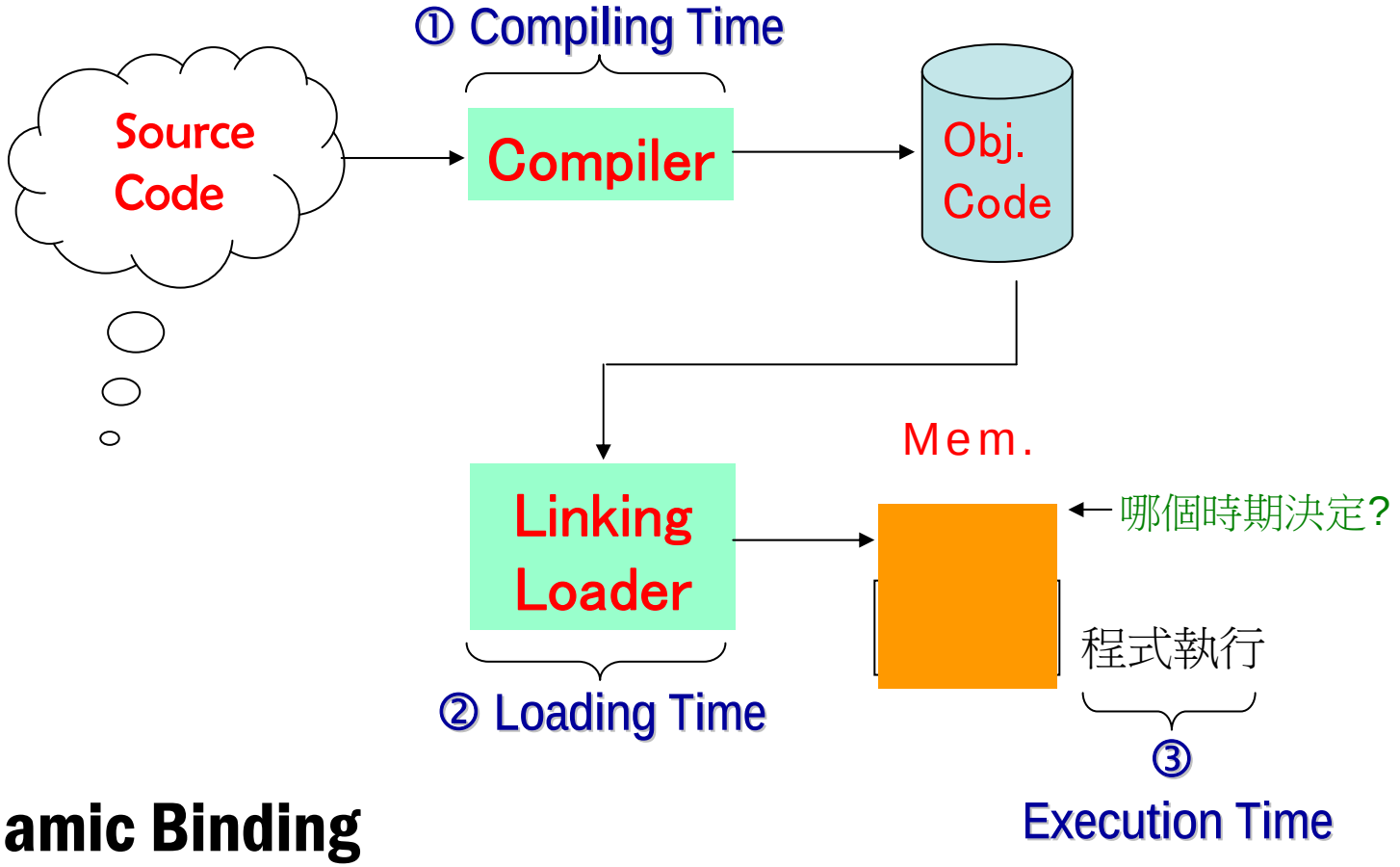
Timer的用途

- CPU Protection
- Time Sharing System
  - 即 : 製作R-R Scheduling
- DMA Controller設定中，傳輸量 (Counter) 大小的設定
- 時鐘 (Clock; 即系統時間)

補 充

補 1. Dynamic Binding 簡介

- **Binding (繫結)**
  - **Def:**
    - 決定程式執行的起始位置。即:程式要在記憶體的哪個地方開始執行。
    - 當程式執行的起始位置決定了，則連帶地程式內的資料或變數之所處記憶體位置也就決定了。
- **Binding的時期可能有三個:**
  - **Compiling Time (編輯時期)**
  - **Loading Time (載入時期)**
  - **Execution Time (執行時期)**



- **Dynamic Binding**
  - **Def:** 在**Execution Time**才決定程式執行的起始位址，表示在程式執行期間，可任意變更其起始位址