OS_110_CH8

Background

- 唯一 CPU 可以直接 access
- 把很多 porcess 放在 memory
- moved between disk and memory

User Program in Multi step

- Compiler > linkage > loader

Address Binding (How to refer memory in a program ?)

彈性：3 > 2 > 1
設計複雜、速度：1 > 2 > 3

Compile Time

- 決定變數和程式碼的位址在哪裡
- 編譯成組合語言 absolute code
- 搬運程式就必須 recompiler

Load Time

- 可以 relocatable code，可以將程式搬運到空的空間，非固定的空間
- 但要搬運記憶體，還是得 recompiler

Execution Time (Run Time)

- 是邏輯位置，不是實體位置
- 會再傳到 CPU 前，會經過 MMU 轉錄程式碼

MMU (Memory-Management Unit)

- 硬體實作的
- 映射虛擬位置到實體位置

- 重新定位暫存器的位址

Logical vs. Physical Address

- Logical address : 從 CPU 送出來的
  - a.k.a. virtual address
- Physical address : Memory module
- compile time & load time address binding
  - L = P
- Execution-time address binding
  - L != P
- 使用者處理的是 L ，從不會看到真實的 P

Static/dynamic loading and linking (How to load a program into memory ?)

Dynamic Loading

- 動態分配記憶體，當被呼叫的時候
- 更好的記憶體運用
  - 從不載入未使用的例程
  - 當大量代碼不經常使用特定用處，也不載入
- 沒有特別 OS 輔助
- 簡單來說，呼叫的時候出現就好，結束了就會被踢走

Static Linking

- 將整包的 lib 載入
- 每隻程式都有載入
- 會有多餘的 lib 在沒有用到 lib 的程式上，但會比較快一點

Dynamic Linking

- 一樣引用，但只會載入一次，之後要用到可以看之前程式是否有引用
- 可以分享的意思
- 編譯有兩種 lib
  - Dynamic (Dynamic Link Library)

    - Static
- 可以節省程式的記憶體，但會比較慢一點

Swap (How to move a program between memory & disk?)

- 在 memory 跟 disk 之間的互動

- Backing store
    - 在 disk 上的一塊空間
    - 為 MMU 之間管理
- Swap back memory location
    - compile / load time
        - 必須 swap 回原來的位置
    - execution time
        - 回來的地方可以不一樣
- when a process to swapped ⇒ **must be idle**
    - 簡單來說，沒有做 CPU 也沒有做 I/O
    - 所以在 I/O 操作時，被 swap 會有記憶體衝突
    - 解決辦法
        - 掛載 I/O 時，不做 swap
        - 透過將 swap 到 OS buffer ，在 I/O 結束後再複製回使用者

Contiguous Memory Allocation 連續的

Fixed-partition allocation

- 將連續的空間分割，切成固定大小的空間
- 切多少格就是，Degree of multi-programming
- 會有多餘的空間

Variable-size partition

- 要多少空間，就給多少
- 會有洞，甚至可能會有很小的洞，沒人塞得進去
- fit
  - First-fit
  - Best-fit
  - Worst-fit

Fragmentation

- Memory 空間浪費的專有名詞

External fragmentation 程式外

- 有空的位置，卻不能讓程式塞進去
- Occur in variable-size allocation
- 解決辦法：compaction 壓縮記憶體，將空的記憶體整理

Internal fragmentation 程式內

- 在固定的空間，程式並沒有將記憶體用完
- Occue in fixed-partition allocation
- 解決方式：paging

Non-Contiguous Memory Allocation - Paging

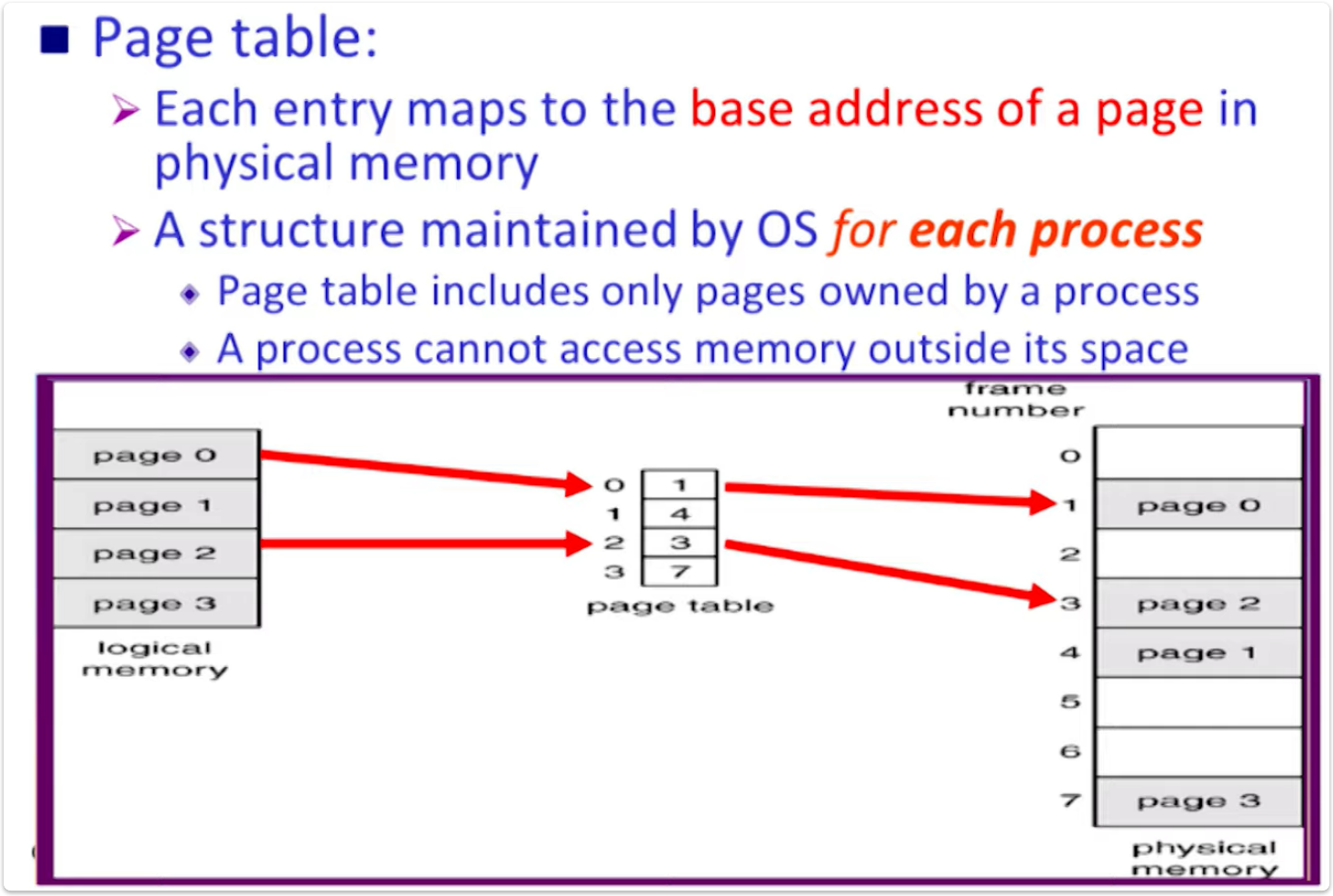
Paging Concept

- 切割
  - physical memory → frames
  - logical address space to same size → pages
- page table to translate logical to physical address
- 益處
  - 允許不連續的記憶體位址的 process
  - 避免 external fragmentation

- 限制 internal fragmentation
- 提供 shared memory/pages

Paging Example

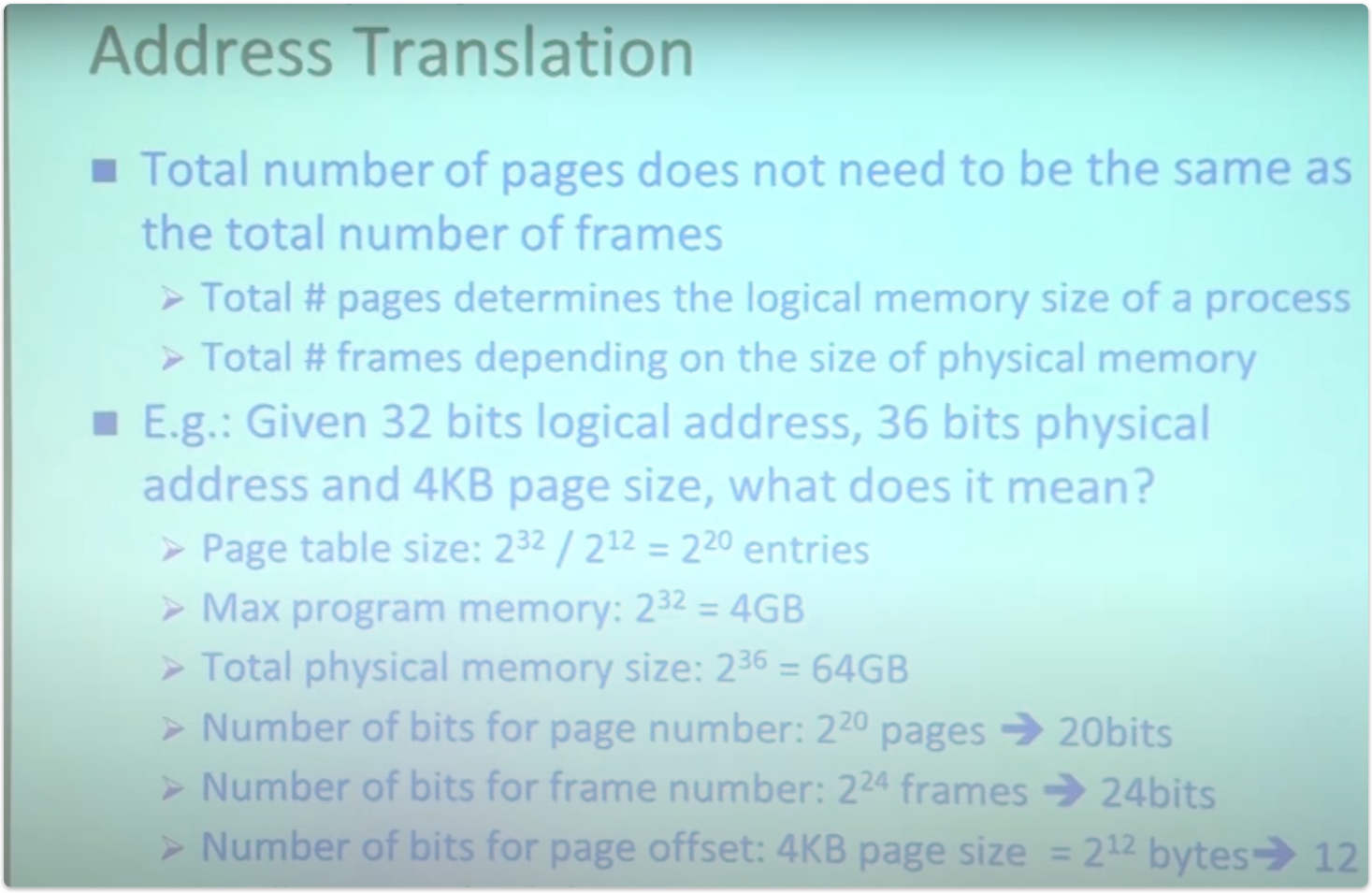
Page table 是 OS 做保持的



Address Translation Scheme

- Logical address
  - Page number (p)
    - 表示為第幾個 page
    - 在哪一個 frame
    - 會透過 Page table 進行 translate
  - Page offset (d)
    - 該 page 的位置
- Physical addr = page base addr + page offset = p + d

Ans. 數字不一定要一樣



Free Frames

- 將空的 Frame 放到一個 link list 裡面
- 用完的也放進去

Page/Frame Size

- define
  - Typically a power of 2
  - 512 bytes ~ 16MB by 1 page
  - 4KB / 8KB page is commonly used
- Internal fragmentation
  - 大尺寸的 page → 會有更多的空間浪費

Swapping

Contiguous Allocation

Paging

Segmentation

Segmentation with Paging