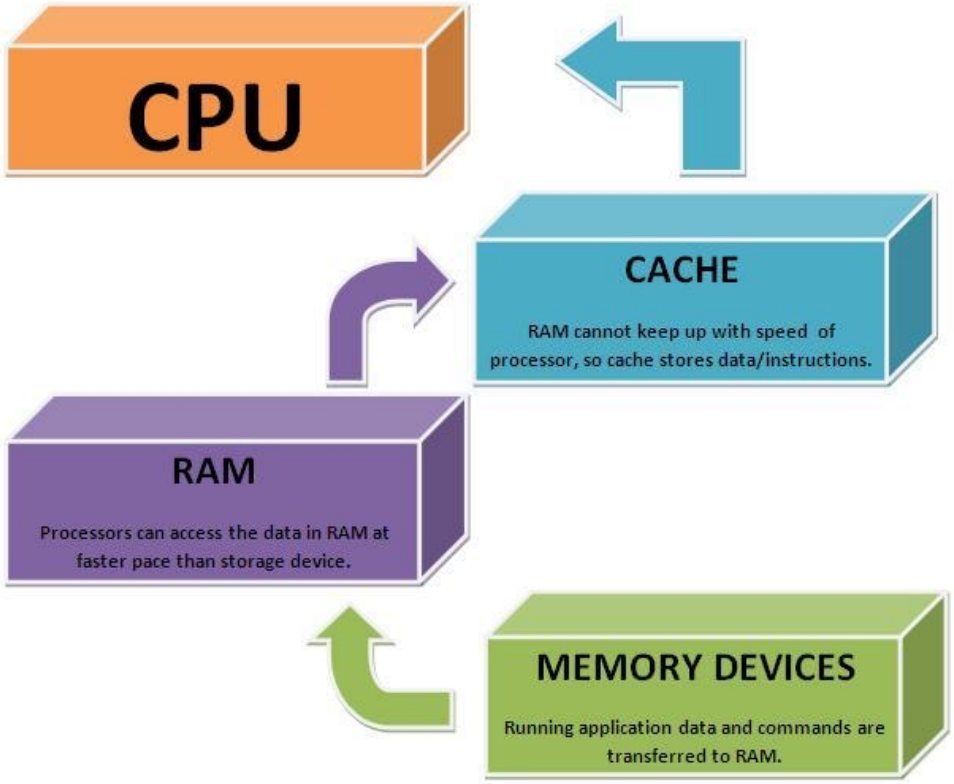
Bài 7
Memory Management

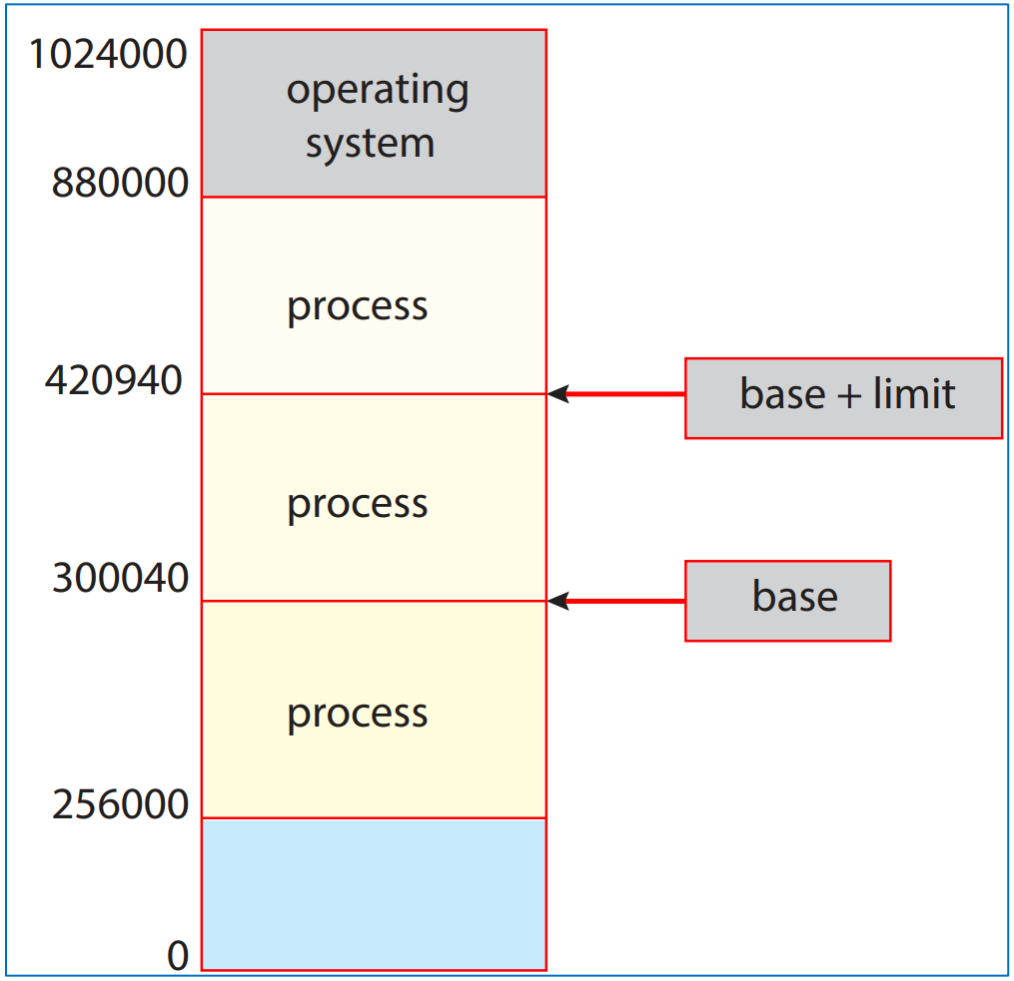
Một số kiến thức cơ bản

- Để program → process, OS cần nạp chương trình từ đĩa vào bộ nhớ chính, bao gồm các chỉ thị (instructions) và dữ liệu (data).
- CPU chỉ có thể truy cập trực tiếp vào các bộ nhớ: bộ nhớ chính (main memory), các thanh ghi (registers) và bộ nhớ đệm (cache).
  - Truy cập thanh ghi hoàn tất trong 1 xung nhịp CPU (hoặc ít hơn).
  - Truy cập bộ nhớ chính tốn nhiều xung nhịp CPU.
  - Bộ nhớ đệm: là bộ nhớ trung gian giữa register và main memory nhằm tăng tốc độ
- Bộ nhớ cần được bảo vệ để đảm bảo hệ thống hoạt động chính xác.



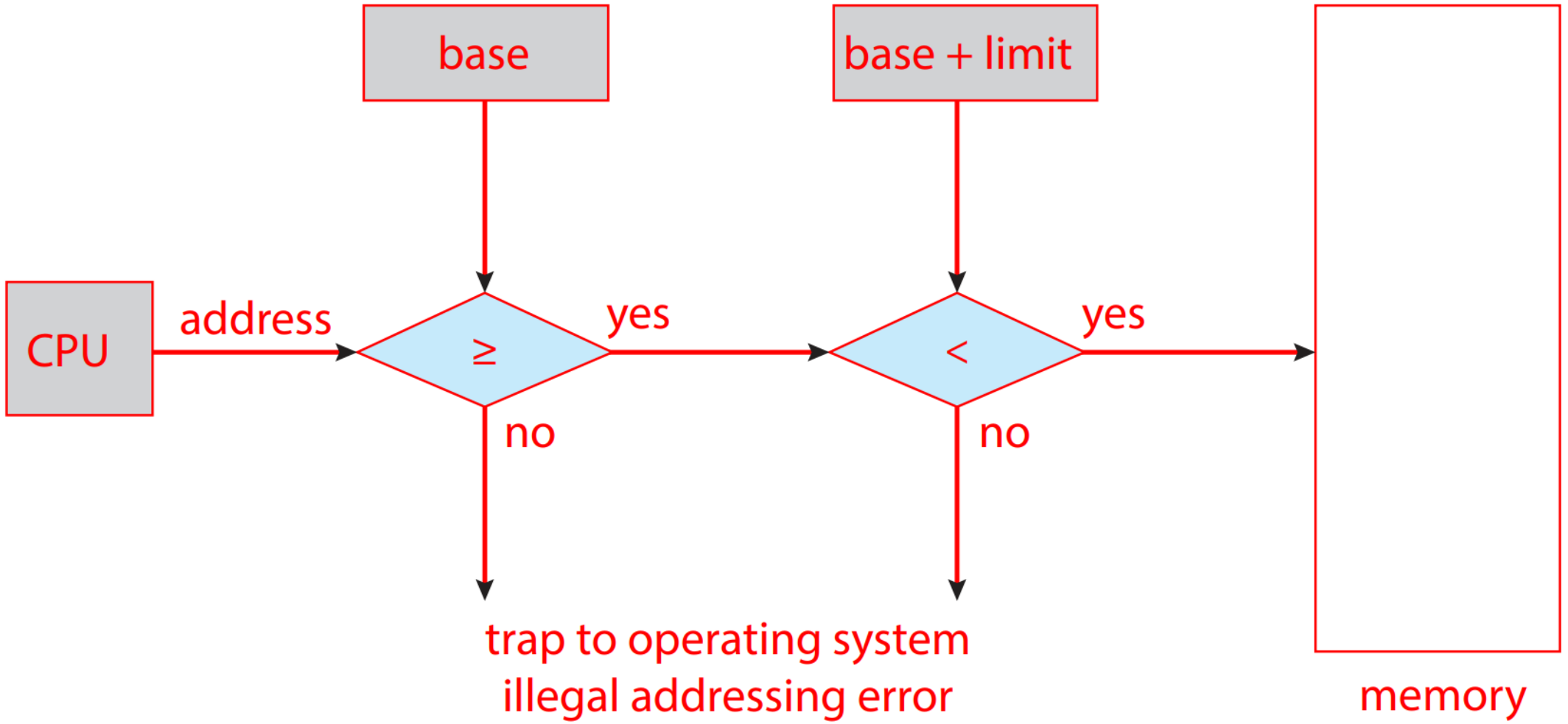
Bảo vệ vùng nhớ

- Bộ nhớ cần có cơ chế bảo vệ nhằm đảm bảo một tiến trình chỉ được truy cập vào cùng nhớ mà nó được OS cấp phát, cần sự hỗ trợ của phần cứng, sử dụng một cặp thanh ghi base register & limit register.
- Ví dụ base register = 300040 và limit register = 120900 thì tiến trình chỉ được truy cập vùng nhớ từ 300040 đến 420940.
- OS (thực thi ở chế độ kernel mode), được phép truy cập toàn bộ bộ nhớ và cặp thanh ghi trên. Các tiến trình thực thi ở user mode), chỉ được phép truy cập vùng nhớ được cho phép.



Bảo vệ vùng nhớ

- CPU kiểm tra mỗi yêu cầu truy cập bộ nhớ nhằm đảm bảo mỗi tiến trình chỉ truy cập vùng nhớ mà nó được hệ điều hành cấp phát.



Quản lý bộ nhớ (Memory management)

- Do chương trình chỉ được thực thi trong bộ nhớ chính → bộ nhớ càng lớn thì OS càng nạp được nhiều chương trình vào bộ nhớ chính cùng lúc → tăng mức độ đa chương của hệ thống & tối ưu hóa việc sử dụng CPU.
- Kernel của OS sẽ chiếm một phần bộ nhớ chính, phần còn lại được phân phối cho các tiến trình.
- Vậy:
  - Quản lý bộ nhớ trong hệ điều hành là gì?
  - Tại sao phải quản lý bộ nhớ?

Quản lý bộ nhớ (Memory management)

- Quản lý bộ nhớ: là một chức năng quan trọng của OS
  - Xác định thời điểm cần cấp phát bộ nhớ cho tiến trình. Cấp phát bao nhiêu ô nhớ, tại những vị trí nào trên bộ nhớ chính.
  - Quản lý các vùng nhớ đã cấp phát cho các tiến trình, cấp phát thêm hoặc thu hồi vùng nhớ khi cần thiết.
  - Quản lý các vùng nhớ trống, bảo vệ vùng nhớ đã được cấp cho các tiến trình.
  - Theo dõi mức độ sử dụng bộ nhớ của các tiến trình, sắp xếp các tiến trình trong bộ nhớ sao cho hệ thống hoạt động tốt nhất.
- Mục tiêu của quản lý bộ nhớ là nạp được càng nhiều tiến trình vào bộ nhớ càng tốt, nhằm tăng tính đa chương trình/đa nhiệm.

Bộ nhớ là hữu hạn, yêu cầu bộ nhớ là vô hạn.

Quản lý bộ nhớ (Memory management)

- Nhiệm vụ cụ thể khi quản lý bộ nhớ:
  - Tổ chức và quản lý bộ nhớ vật lý.
  - Tổ chức và quản lý bộ nhớ logic.
  - Định vị và tái định vị các tiến trình.
  - Chia sẻ bộ nhớ cho các tiến trình.
  - Bảo vệ vùng nhớ của các tiến trình.
- Các khía cạnh mà OS cần xem xét khi cấp phát vùng nhớ cho tiến trình:
  - Sự tương ứng giữa địa chỉ logic và địa chỉ vật lý.
  - Quản lý bộ nhớ vật lý.
  - Chia sẻ thông tin (giữa các tiến trình trong bộ nhớ)
  - Bảo vệ vùng nhớ.

Một số kiến thức về bộ nhớ

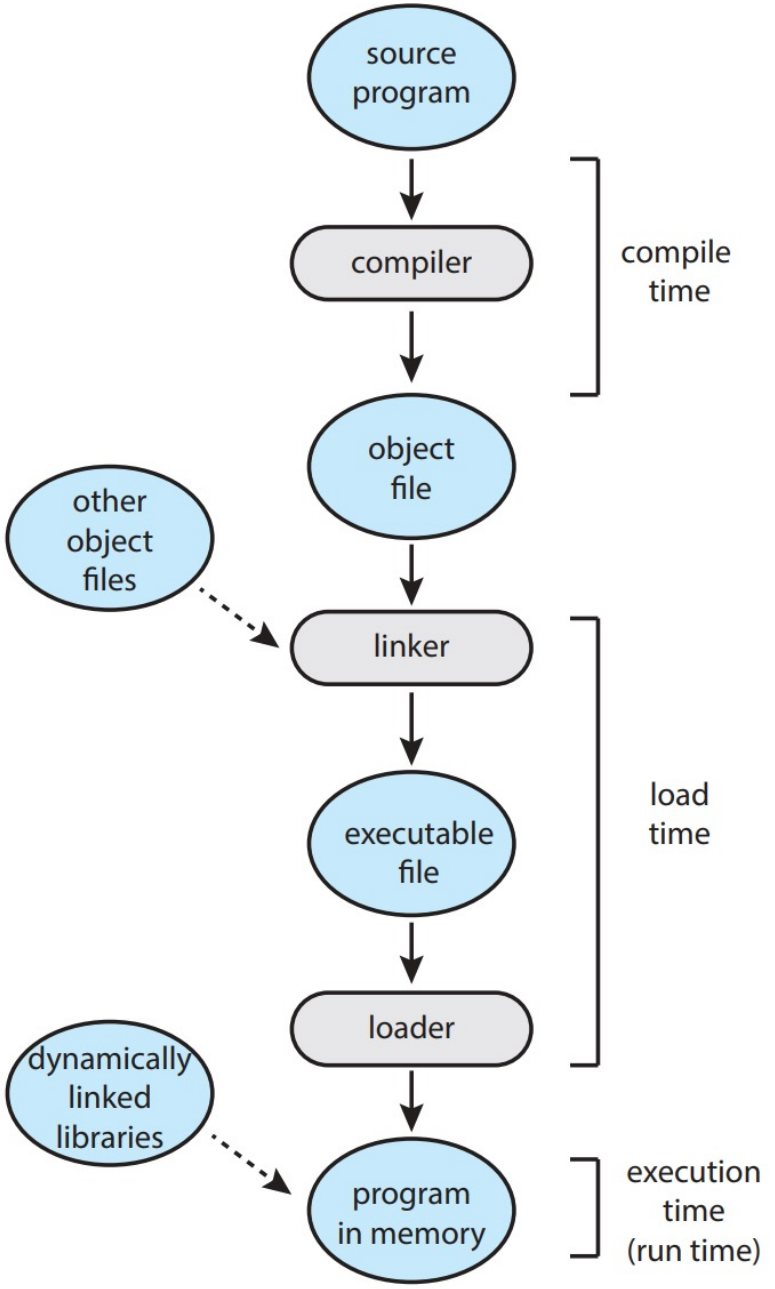
- Bộ nhớ chính (RAM) là một mảng lớn (từ hàng trăm ngàn đến hàng tỉ) các từ nhớ (words) hoặc bytes.
- Việc trao đổi thông tin trên bộ nhớ chính được thực hiện thông qua thao tác đọc/ghi vào một địa chỉ cụ thể trong bộ nhớ.
- Chương trình là tập hợp các chỉ thị (instructions) và dữ liệu (data).
- Nạp chương trình vào bộ nhớ là đặt các chỉ thị lệnh và dữ liệu vào địa chỉ các ô nhớ của tiến trình trong vùng nhớ được OS cấp.
- Các địa chỉ trong chương trình là địa chỉ tượng trưng, để thực thi được thì cần chuyển các địa chỉ này thành địa chỉ thực trong bộ nhớ chính (địa chỉ vật lý) → gọi là quá trình chuyển đổi địa chỉ (address binding)

Address binding (chuyển đổi địa chỉ)

- Chuyển đổi (kết buộc/gắn kết) địa chỉ: là quá trình ánh xạ địa chỉ từ không gian địa chỉ này sang không gian địa chỉ khác (thường là logic → vật lý).
  - Địa chỉ sử dụng trong mã nguồn (biến, tham số) thường sử dụng địa chỉ tượng trưng.
  - Địa chỉ sử dụng trong mã nguồn đã biên dịch sẽ được chuyển thành các địa chỉ có thể tái định vị (relocatable address). Ví dụ: 14 bytes kể từ phần bắt đầu của module này.
  - Khi chương trình thực thi, bộ linker hoặc loader sẽ chuyển đổi các địa chỉ có thể tái định vị thành các địa chỉ tuyệt đối (absolute address).
- Quá trình chuyển đổi địa chỉ có thể xảy ra ở 3 giai đoạn: biên dịch (compile time), nạp (load time), thực thi (execution time).

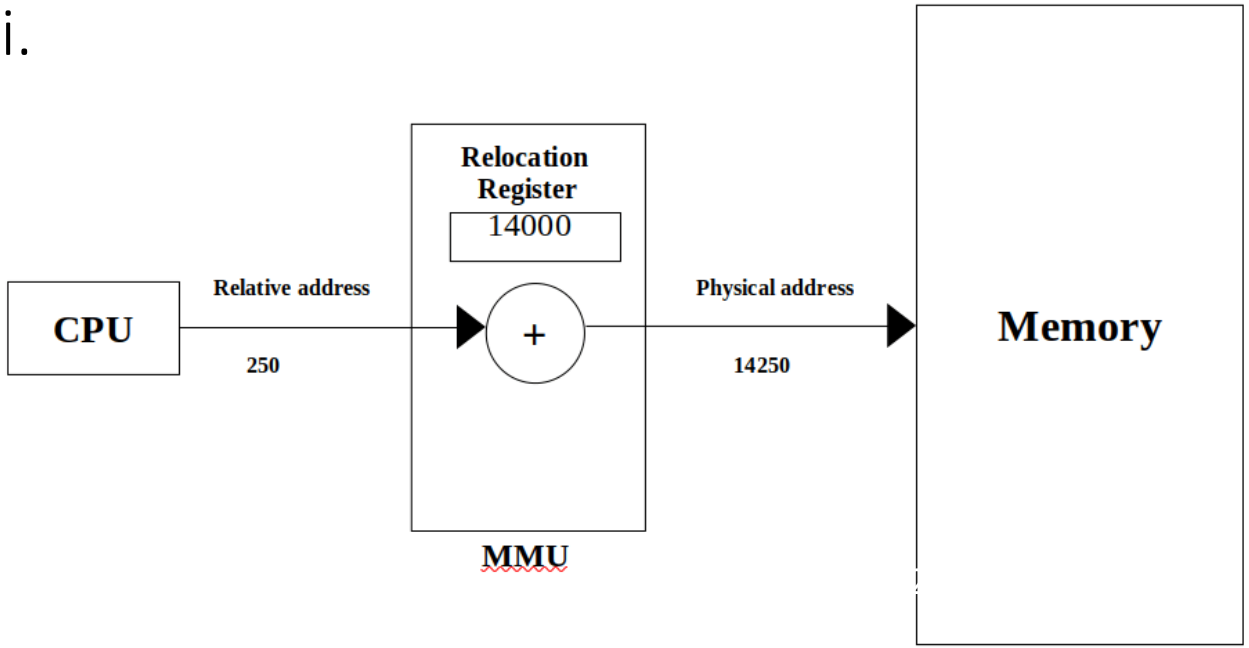
Thời điểm chuyển đổi địa chỉ

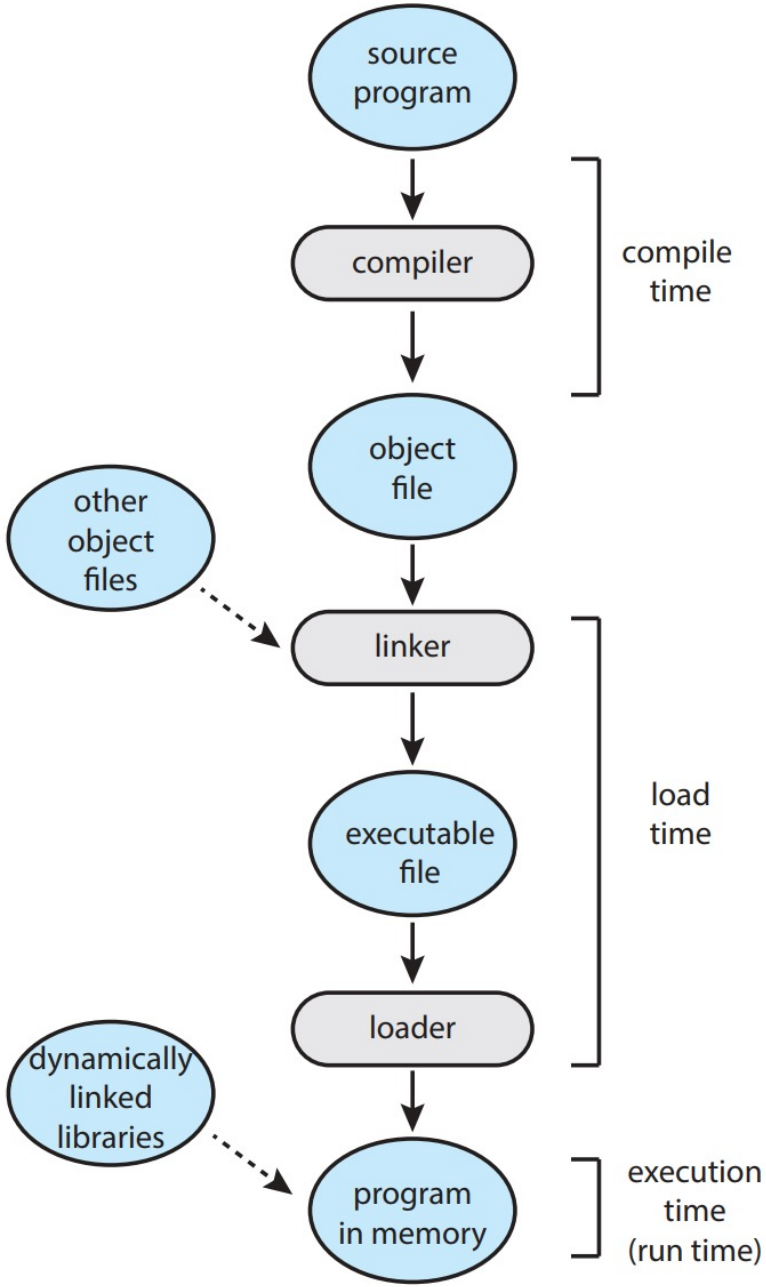
- **Thời điểm biên dịch (compile time):** tại thời điểm biên dịch, nếu biết trước chương trình sẽ nằm tại vị trí nào trong bộ nhớ thì có thể tạo địa chỉ tuyệt đối (absolute address).
  - Chỉ có thể áp dụng nếu biết trước tiến trình cần sử dụng bao nhiêu bộ nhớ, biết trước vị trí của tiến trình trên bộ nhớ và vị trí đó phải cố định.
  - Nếu thay đổi vị trí tiến trình trên bộ nhớ -> cần biên dịch lại chương trình.
  - Một hệ thống có sử dụng chuyển đổi địa chỉ tại thời điểm biên dịch là MS-DOS.



Thời điểm chuyển đổi địa chỉ

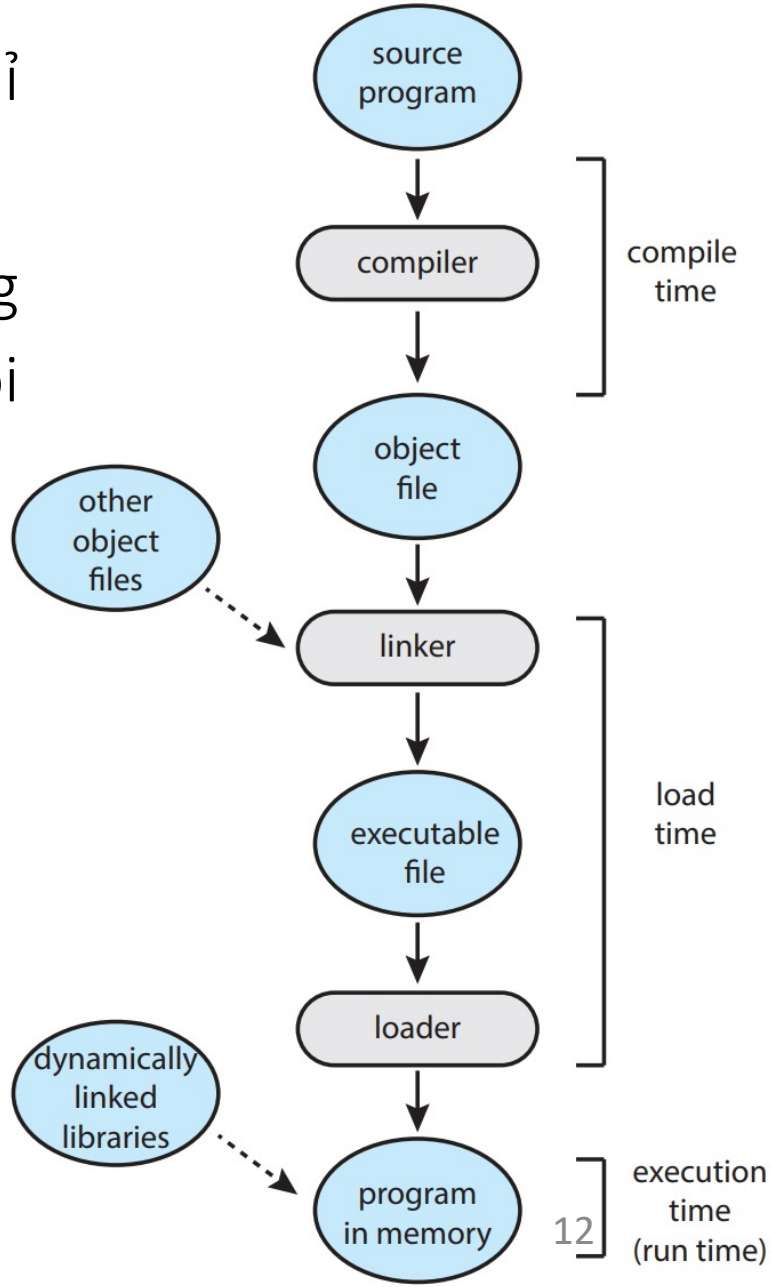
- **Thời điểm nạp (load time):** chuyển đổi địa chỉ xảy ra trong quá trình nạp chương trình vào bộ nhớ.
- Trình biên dịch chuyển các địa chỉ tượng trưng trong mã nguồn → địa chỉ tương đối, địa chỉ tương đối → địa chỉ tuyệt đối nhờ trình quản lý bộ nhớ (loader).
- Có thể thay đổi vị trí chương trình mà không cần biên dịch lại.





Thời điểm chuyển đổi địa chỉ

- **Thời điểm thực thi (execution time):** chuyển đổi địa chỉ thực hiện linh hoạt trong lúc chương trình đang thực thi.
- Phương pháp này cực kỳ linh hoạt, nhưng gây ảnh hưởng hiệu suất do phải thực hiện nhiều phép tính chuyển đổi địa chỉ.
- Phần lớn hệ điều hành sử dụng cơ chế này.

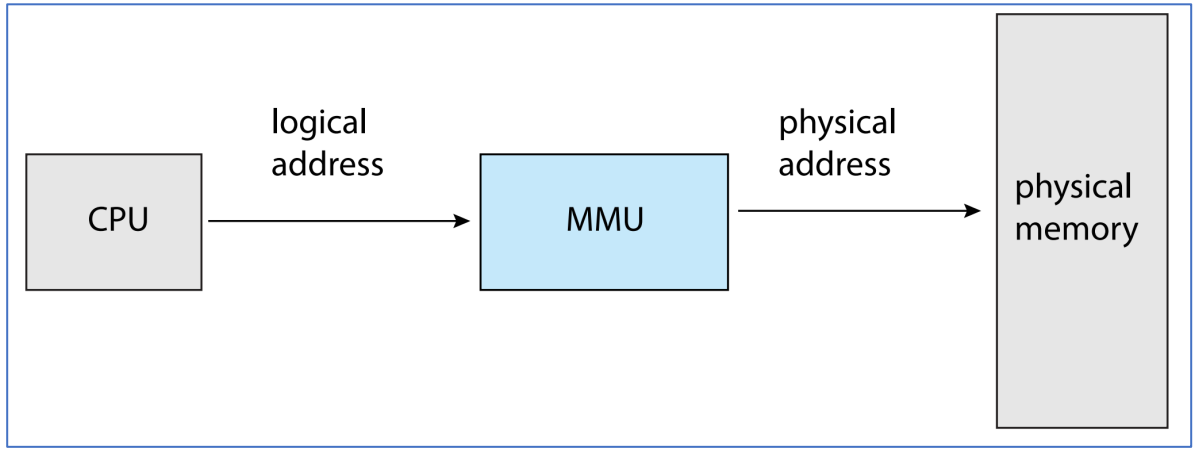


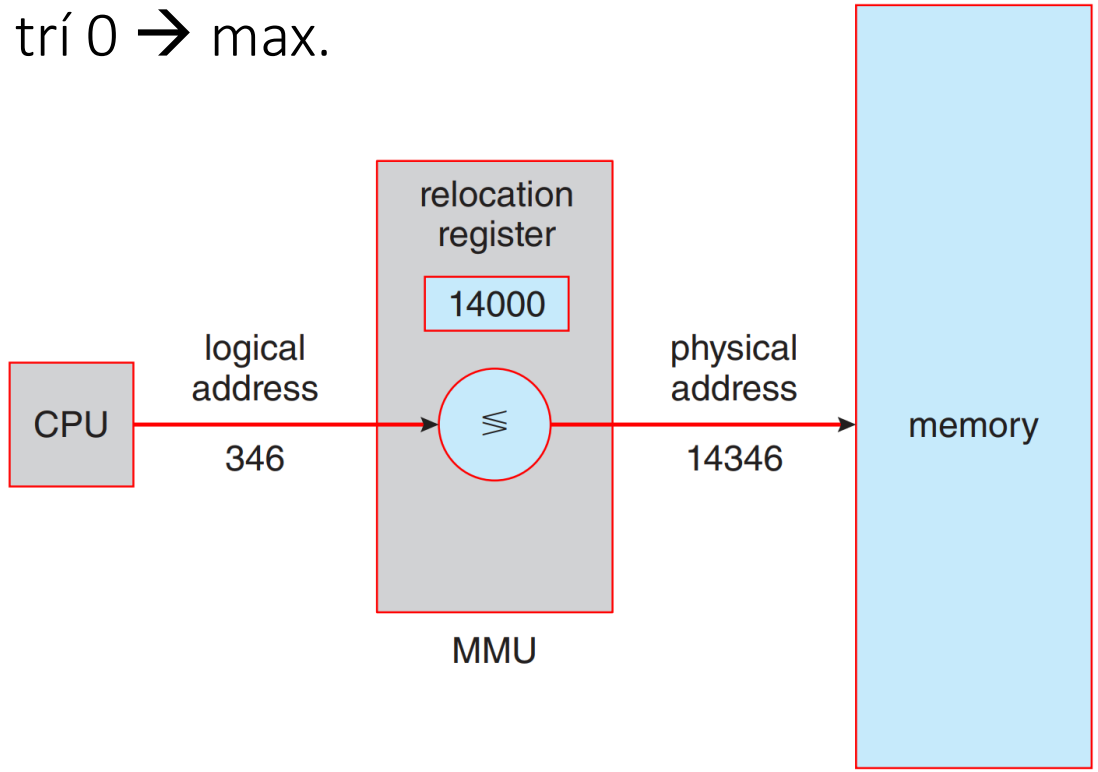
Địa chỉ logic (logical address) và địa chỉ vật lý (physical address)

- Địa chỉ logic (logical address): được tạo ra bởi CPU, còn được gọi là địa chỉ ảo.
- Không gian địa chỉ logic (logical address space): tập hợp tất cả địa chỉ logic phát sinh bởi một chương trình
- Địa chỉ vật lý (physical address): là địa chỉ thực sự trỏ đến vị trí vật lý của dữ liệu trong bộ nhớ, mỗi địa chỉ vật lý tương ứng với 1 vị trí cụ thể trên bộ nhớ chính.
- Không gian địa chỉ vật lý (physical address space): tập hợp tất cả các địa chỉ vật lý tương ứng với địa chỉ logic.

Memory Management Unit (MMU)

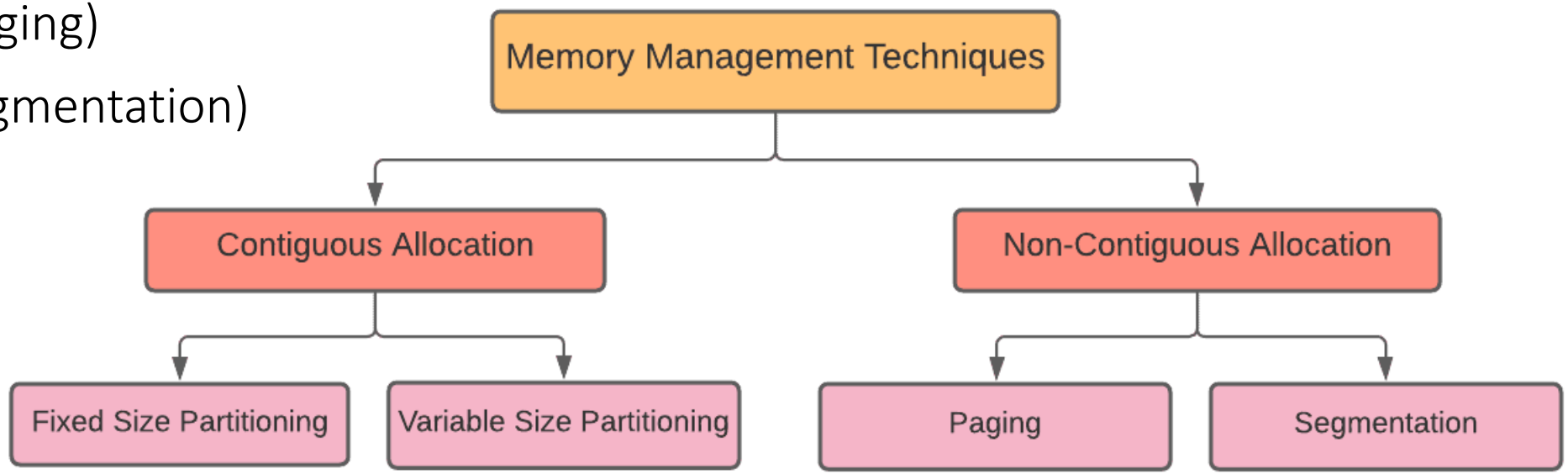
- Là một cơ chế phần cứng chịu trách nhiệm ánh xạ địa chỉ logic thành địa chỉ vật lý vào thời điểm thực thi.
- Chương trình của người dùng chỉ quan tâm địa chỉ logic, không nhìn thấy địa chỉ vật lý.
- Như vậy, địa chỉ logic từ [0,max] và địa chỉ vật lý [R+0;R+max] (R là giá trị của thanh ghi tái định vị - relocation register). Chương trình của người dùng chỉ tạo ra địa chỉ logic và nghĩ rằng tiến trình được nạp trong bộ nhớ từ vị trí 0 → max.





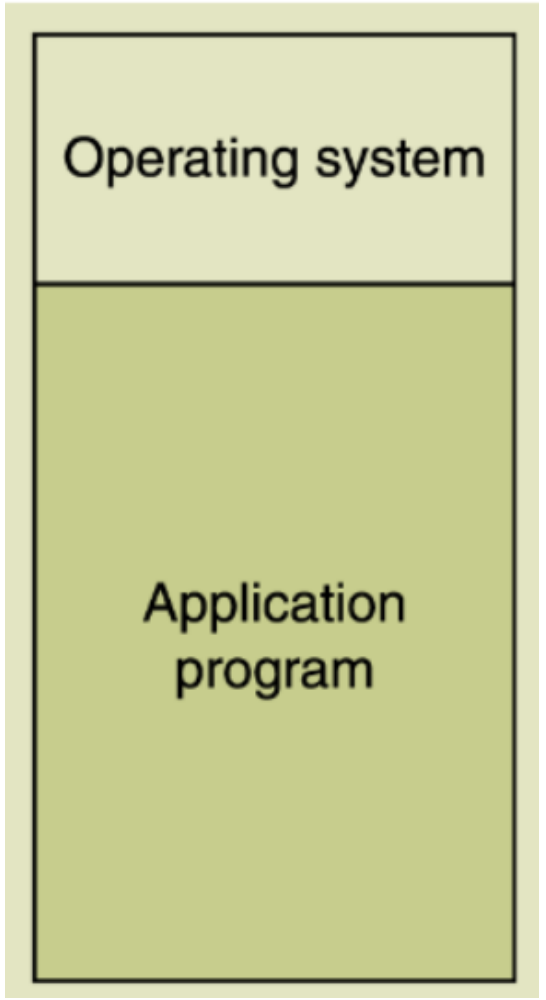
Cấp phát bộ nhớ trong OS

- Cấp phát bộ nhớ liên tục (Contiguous memory)
  - Single contiguous allocation
  - Partitioned allocation (Multiple partitioning):
    - ✓ Fixed partitioning
    - ✓ Dynamic partitioning (variable)
- Cấp phát bộ nhớ không liên tục (Non-Contiguous memory)
  - Phân trang (paging)
  - Phân đoạn (segmentation)



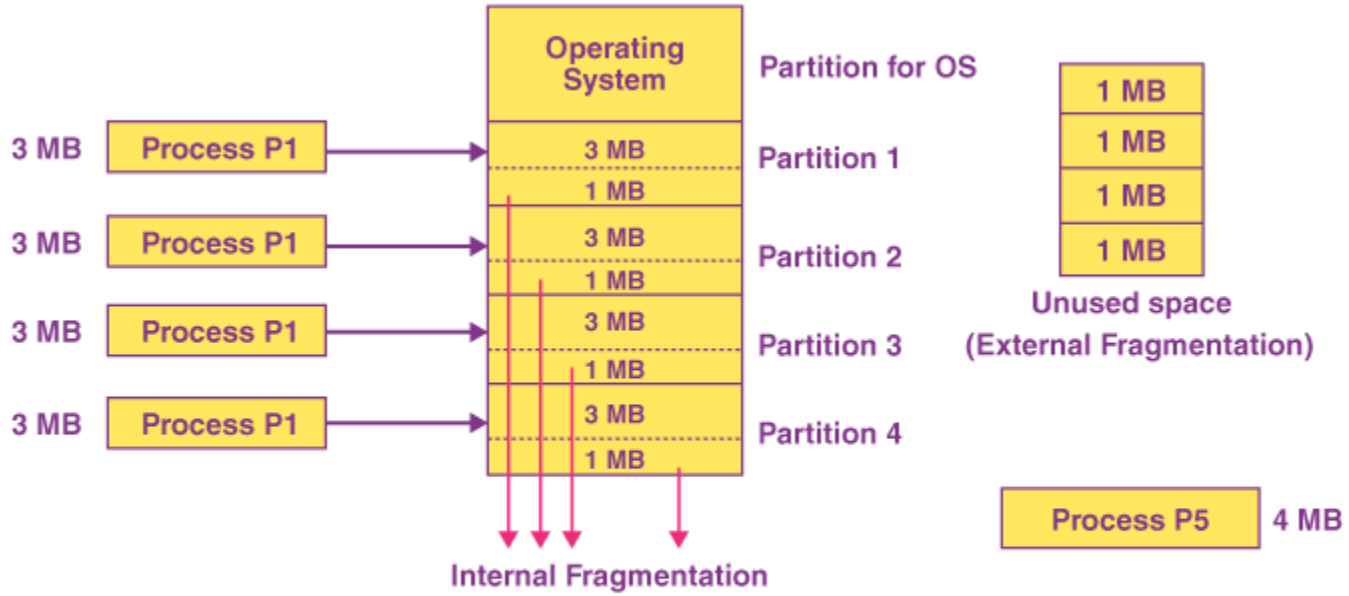
Cấp phát bộ nhớ liên tục (Contiguous memory)

- Cấp phát bộ nhớ liên tục: cung cấp một vùng liên tục của bộ nhớ cho một chương trình.
- Đối với Single contiguous allocation: ít phổ biến
  - Bộ nhớ chính được chia thành 2 phân vùng: vùng bộ nhớ thấp để chứa OS, vùng bộ nhớ cao chứa các tiến trình.
  - Như vậy chỉ có 2 đối tượng trên bộ nhớ chính: OS và 1 tiến trình người dùng → lãng phí bộ nhớ chính và thời gian CPU.



Cấp phát bộ nhớ liên tục (Contiguous memory)

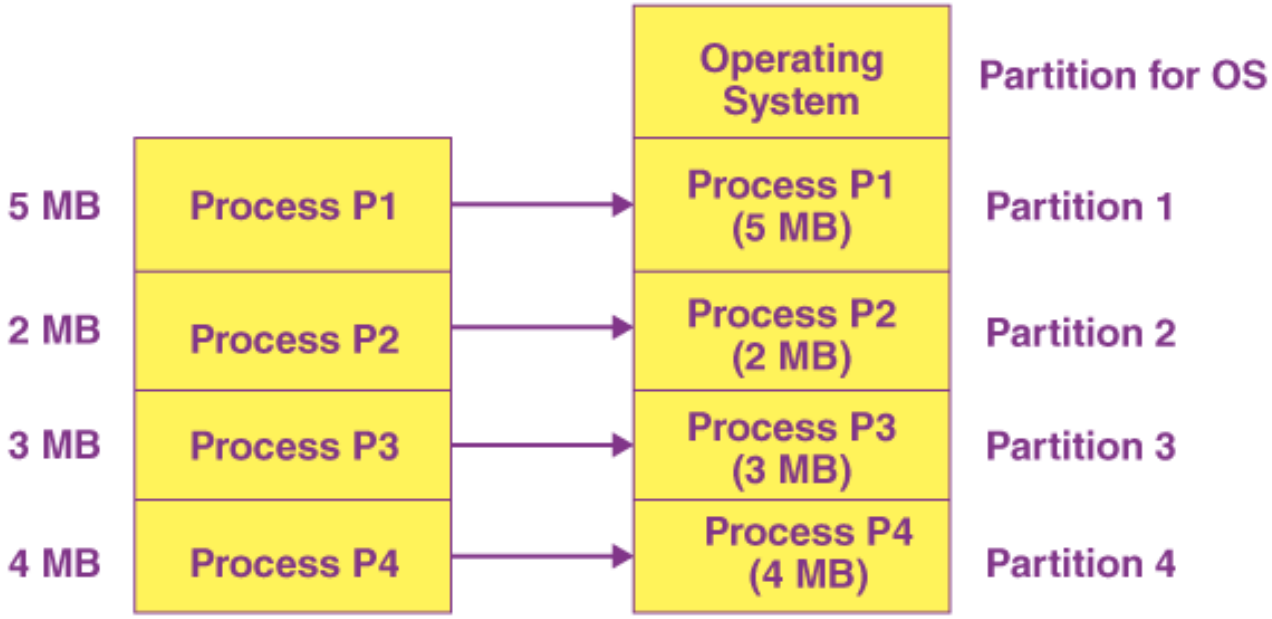
- Partitioned allocation (Multiple partitioning): chia bộ nhớ chính thành nhiều phần và nạp nhiều tiến trình vào bộ nhớ chính cùng lúc.
- Gồm 2 loại:
  - Fixed partitioning: chia bộ nhớ chính thành nhiều phân vùng có kích thước bằng nhau hoặc khác nhau. Mỗi tiến trình được nạp vào 1 phân vùng.



Fixed Partitioning

(Contiguous memory allocation)

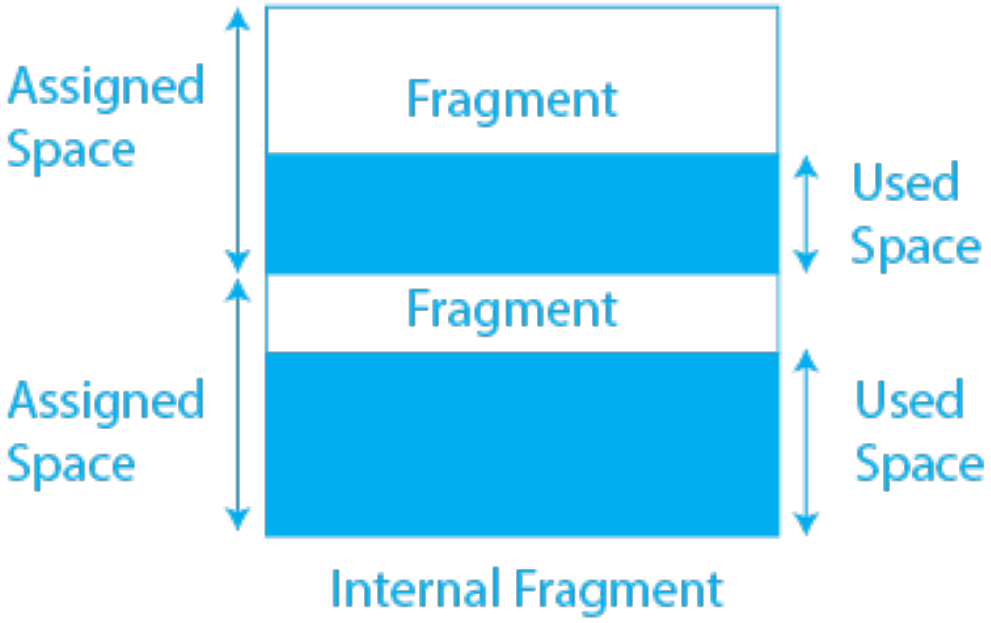
- Partitioned allocation (Multiple partitioning): chia bộ nhớ chính thành nhiều phần và nạp nhiều tiến trình vào bộ nhớ chính cùng lúc.
- Gồm 2 loại:
  - Dynamic partitioning: chia bộ nhớ thành nhiều phân vùng, kích thước mỗi phân vùng phụ thuộc vào tiến trình được nạp vào.

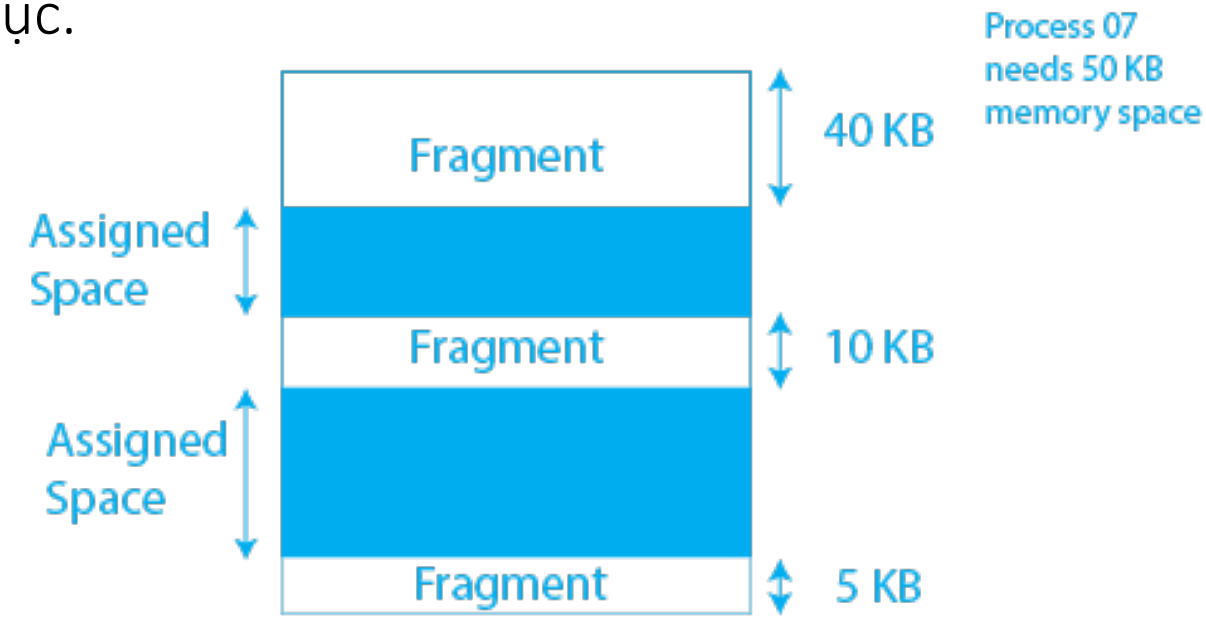


Dynamic Partitioning
(Process Size = Patition Size)

Cấp phát bộ nhớ liên tục (Contiguous memory)

- Vấn đề phân mảnh bộ nhớ (fragmentation): không gian bộ nhớ bị phân thành nhiều "lỗ trống" (gọi là hole) không liên tục, bao gồm:
  - Phân mảnh nội (**internal fragmentation**): vùng nhớ được cấp phát lớn hơn vùng nhớ yêu cầu.
  - Phân mảnh ngoại (**external fragmentation**): vùng nhớ còn trống đủ lớn để cấp phát cho tiến trình, nhưng lại không liên tục.



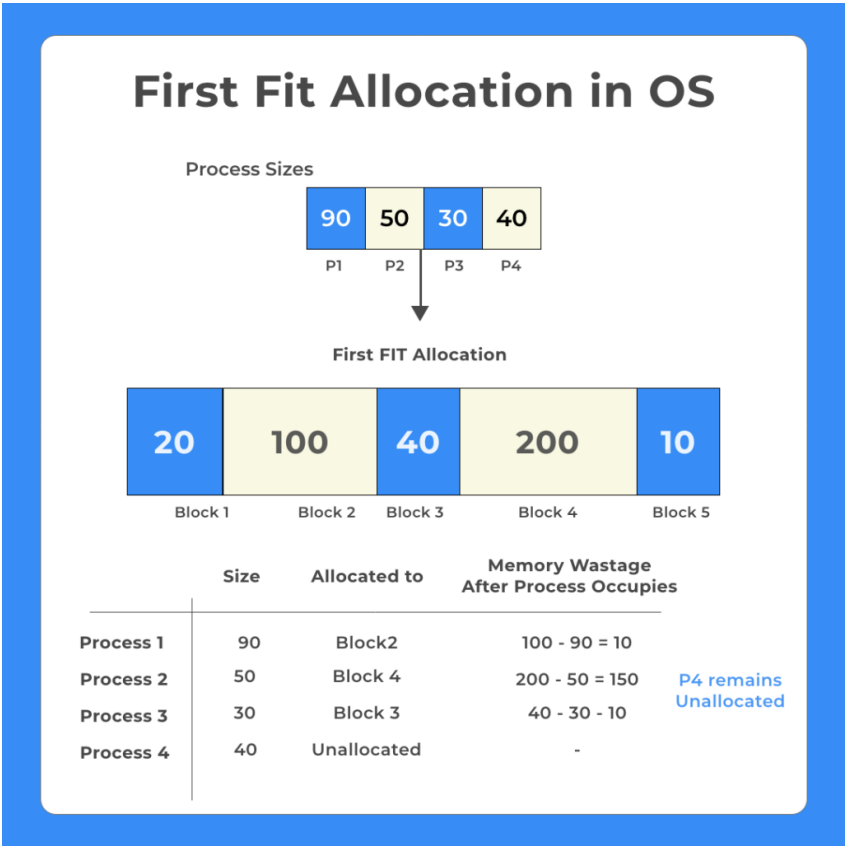


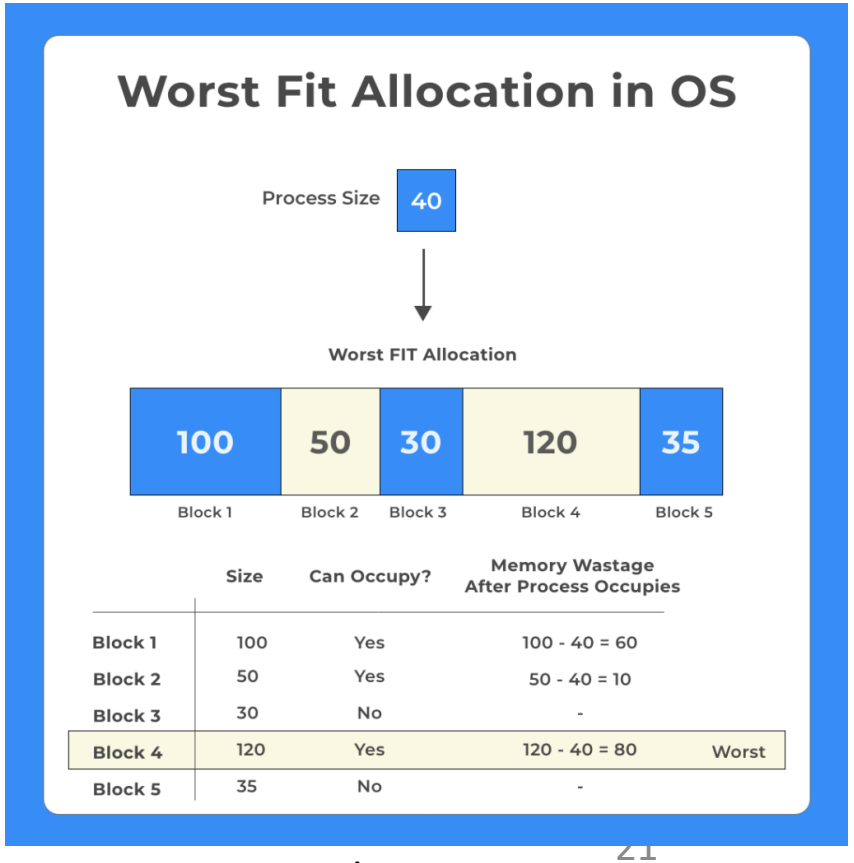
Cấp phát bộ nhớ liên tục (Contiguous memory)

- Vấn đề phân mảnh bộ nhớ đối với cấp phát bộ nhớ liên tục:
  - Fixed partitioning: gặp cả phân mảnh nội và phân mảnh ngoại.
  - Dynamic partitioning: không phân mảnh nội nhưng có phân mảnh ngoại.
- Hướng giải quyết vấn đề phân mảnh bộ nhớ:
  - Đề ra chiến lược cấp phát hợp lý: first-fit, best-fit, worst-fit.
  - Nén bộ nhớ (compaction)
  - Sử dụng kỹ thuật hoán đổi (swapping)
  - Sử dụng kỹ thuật phủ lấp (overlay)

Cấp phát bộ nhớ liên tục (Contiguous memory)

- Hướng giải quyết vấn đề phân mảnh bộ nhớ → chiến lược cấp phát hợp lý:
  - First-fit: tìm "lỗ trống" đầu tiên đủ lớn trên bộ nhớ để nạp tiến trình.
  - Best-fit: tìm "lỗ trống" nhỏ nhất, vừa đủ trên bộ nhớ để nạp tiến trình (tìm trên toàn bộ "lỗ trống")
  - Worst-fit: tìm "lỗ trống" lớn nhất trên bộ nhớ để nạp tiến trình (tìm trên toàn bộ "lỗ trống").

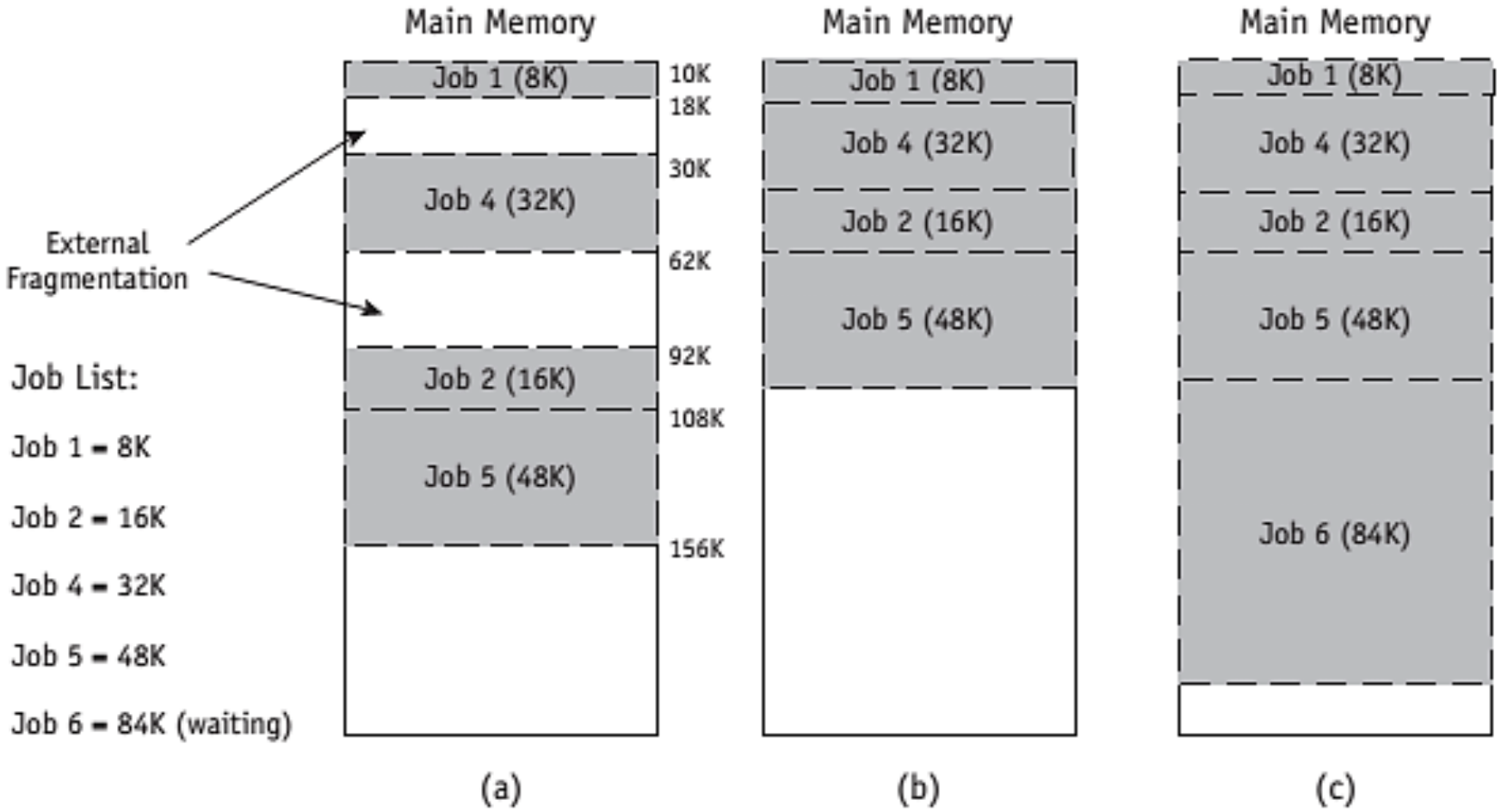




Đọc thêm: https://prepinsta.com/operating-systems/first-fit-best-fit-worst-fit-in-os-example/

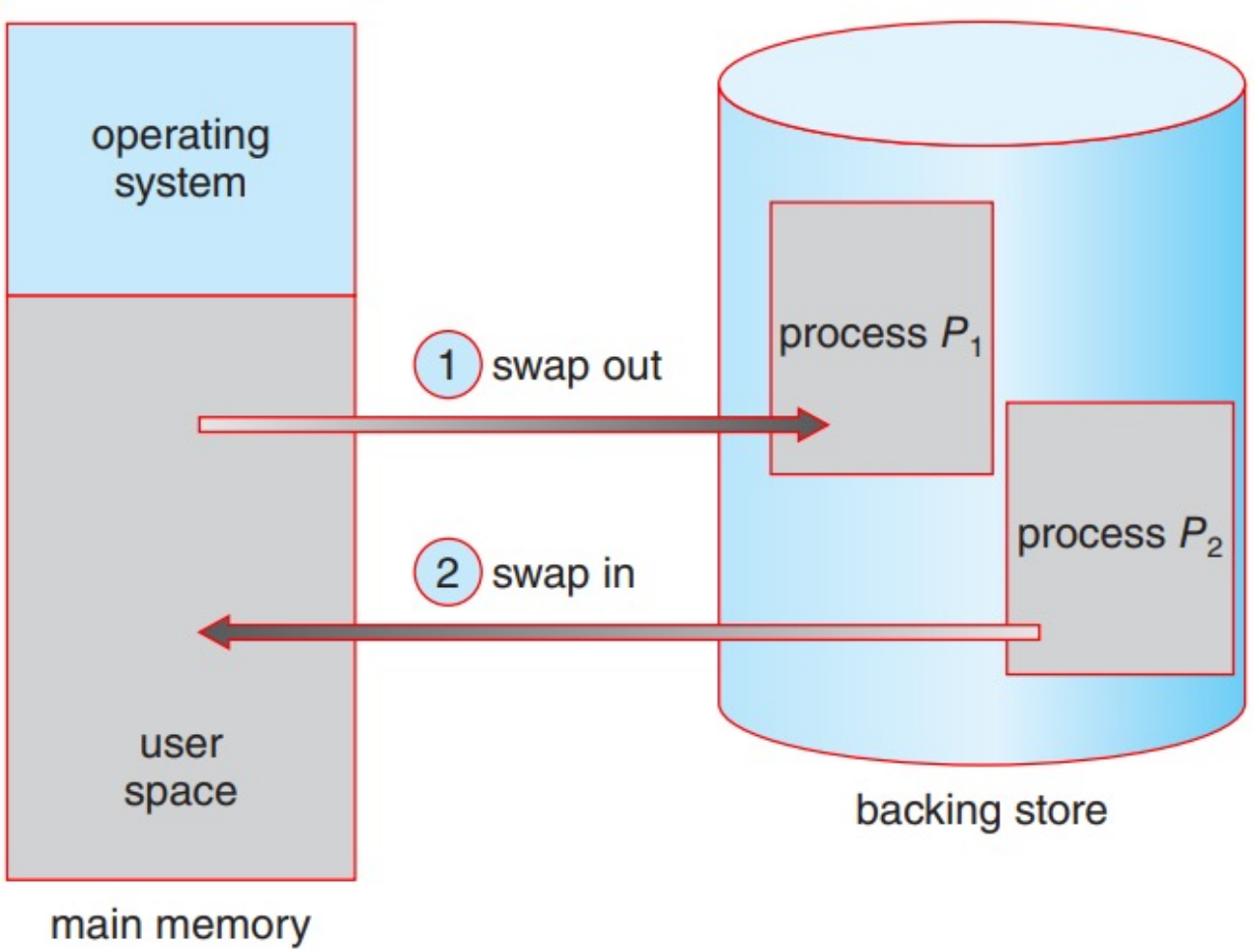
Cấp phát bộ nhớ liên tục (Contiguous memory)

- Hướng giải quyết vấn đề phân mảnh bộ nhớ → nén bộ nhớ (compaction)
  - Kết hợp các vùng nhớ nhỏ, rời rạc thành một vùng nhớ trống đủ lớn.
  - Đòi hỏi thời gian xử lý, chuyển địa chỉ phải xảy ra lúc thực thi.



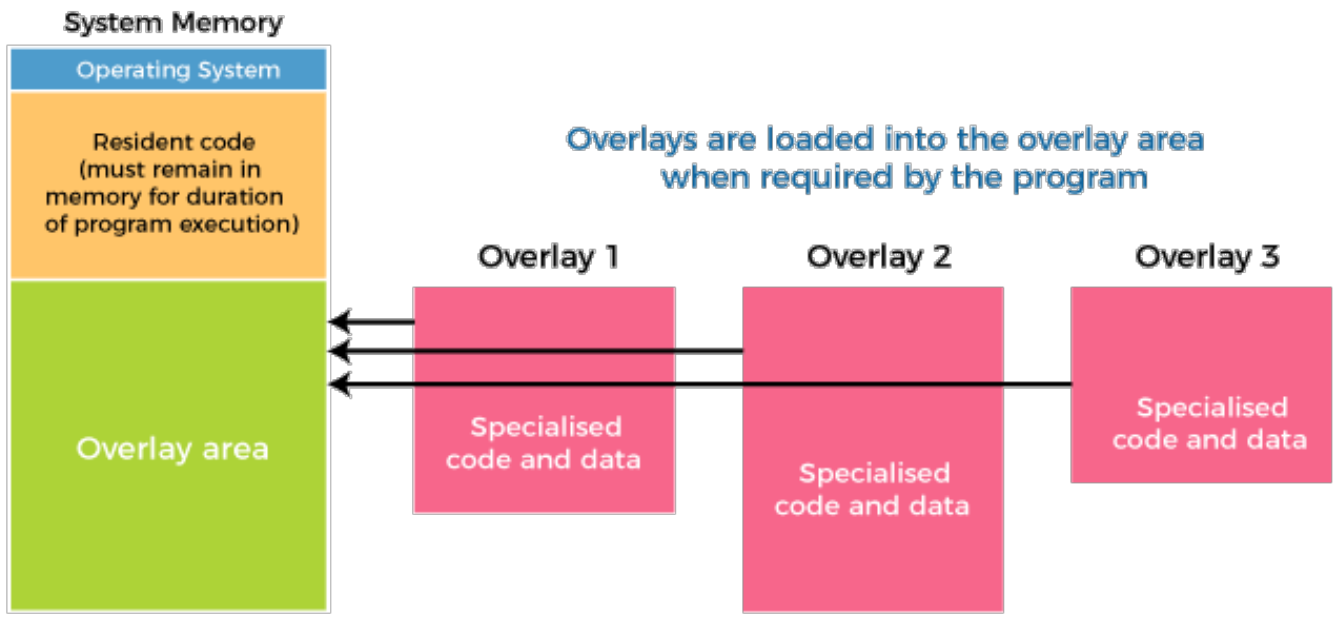
Cấp phát bộ nhớ liên tục (Contiguous memory)

- Hướng giải quyết vấn đề phân mảnh bộ nhớ → hoán đổi (swapping)
  - Di chuyển process khỏi bộ nhớ chính và lưu trên bộ nhớ phụ (quá trình swap out). Khi thích hợp, nạp lại process vào bộ nhớ chính để thực thi (quá trình swap in).
  - Có thể áp dụng các chiến lược:
    - ✓ Round robin
    - ✓ Roll out, roll in: theo độ ưu tiên.



Cấp phát bộ nhớ liên tục (Contiguous memory)

- Hướng giải quyết vấn đề phân mảnh bộ nhớ → phủ lấp (overlay)
  - Chia nhỏ một chương trình thành các phần nhỏ hơn (gọi là overlay).
  - Chỉ giữa lại overlay cần thiết, các overlay còn lại được lưu trên thiết bị khác.
  - Chuyển đổi giữa các overlay.



- Ví dụ chương trình assembler được tổ chức như hình dưới đây, mỗi pass tương ứng với 1 công việc. Giả sử rằng bộ nhớ chính có kích thước 150KB và kích thước chương trình cần là 200KB
  - Giả sử kích thước của overlays driver là 10KB.
  - Mỗi pass sẽ được nạp lần lượt vào bộ nhớ chính. Cả hai pass đều cần Symbol table và Common routine.
  - Kích thước bộ nhớ pass 1 cần: 70KB + 30KB + 20KB + 10KB = 130KB
  - Kích thước bộ nhớ pass 2 cần: 80KB + 30KB + 20KB + 10KB = 140KB
  - Như vậy, nhờ kỹ thuật overlay, chương trình 200KB có thể thực thi với bộ nhớ tối thiểu 140KB

Pass 1.....70KB

Pass 2.....80KB

Symbol table.....30KB

Common routine.....20KB

Cấp phát bộ nhớ không liên tục (Non-Contiguous Memory)

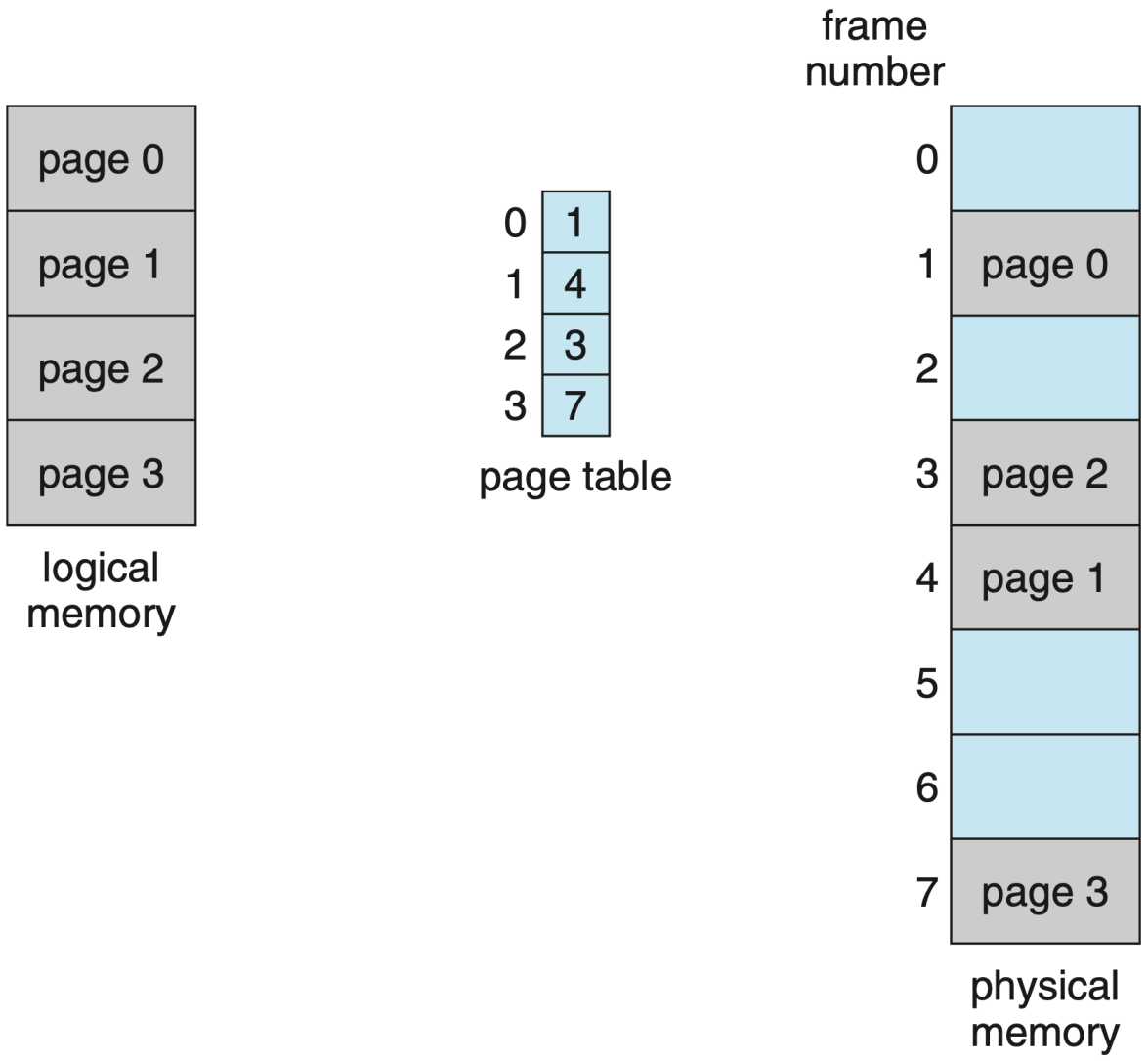
- Ý tưởng: chương trình được chia thành các khối khác nhau (gọi là pages/fragment) và nạp vào những phần khác nhau của bộ nhớ chính (gọi là frame/segment) , không nhất thiết phải liền kề nhau.
- Gồm một số loại:
  - Phân trang (paging).
  - Phân đoạn (segmentation).
  - Phân trang kết hợp phân đoạn (segmentation with paging)
- Cấp phát bộ nhớ không liên tục giảm sự lãng phí bộ nhớ do phân mảnh, nhưng tăng chi phí chuyển đổi địa chỉ bộ nhớ. Ngoài ra, các phần của tiến trình được lưu ở nhiều phần khác nhau của bộ nhớ nên ảnh hưởng đến tốc độ thực thi.

Cấp phát bộ nhớ không liên tục (Non-Contiguous Memory)

- Phân trang (Paging):
  - Không gian địa chỉ vật lý của một tiến trình có thể không liên tục.
  - Chia bộ nhớ chính thành các khối có kích thước cố định, gọi là frames (khung trang). Kích thước là là hàm mũ của 2, từ 512 bytes → 16 MB
  - Chia bộ nhớ ảo thành các khối có kích thước tương đương, gọi là pages (trang).
  - Lưu trạng thái của tất cả các frames.
  - Để thực thi một chương trình có kích thước N pages, cần tìm N frames trống để nạp chương trình.
  - Cần tạo bảng trang (page table) để chuyển đổi địa chỉ logic → vật lý.
  - Vẫn có hiện tượng phân mảnh nội.

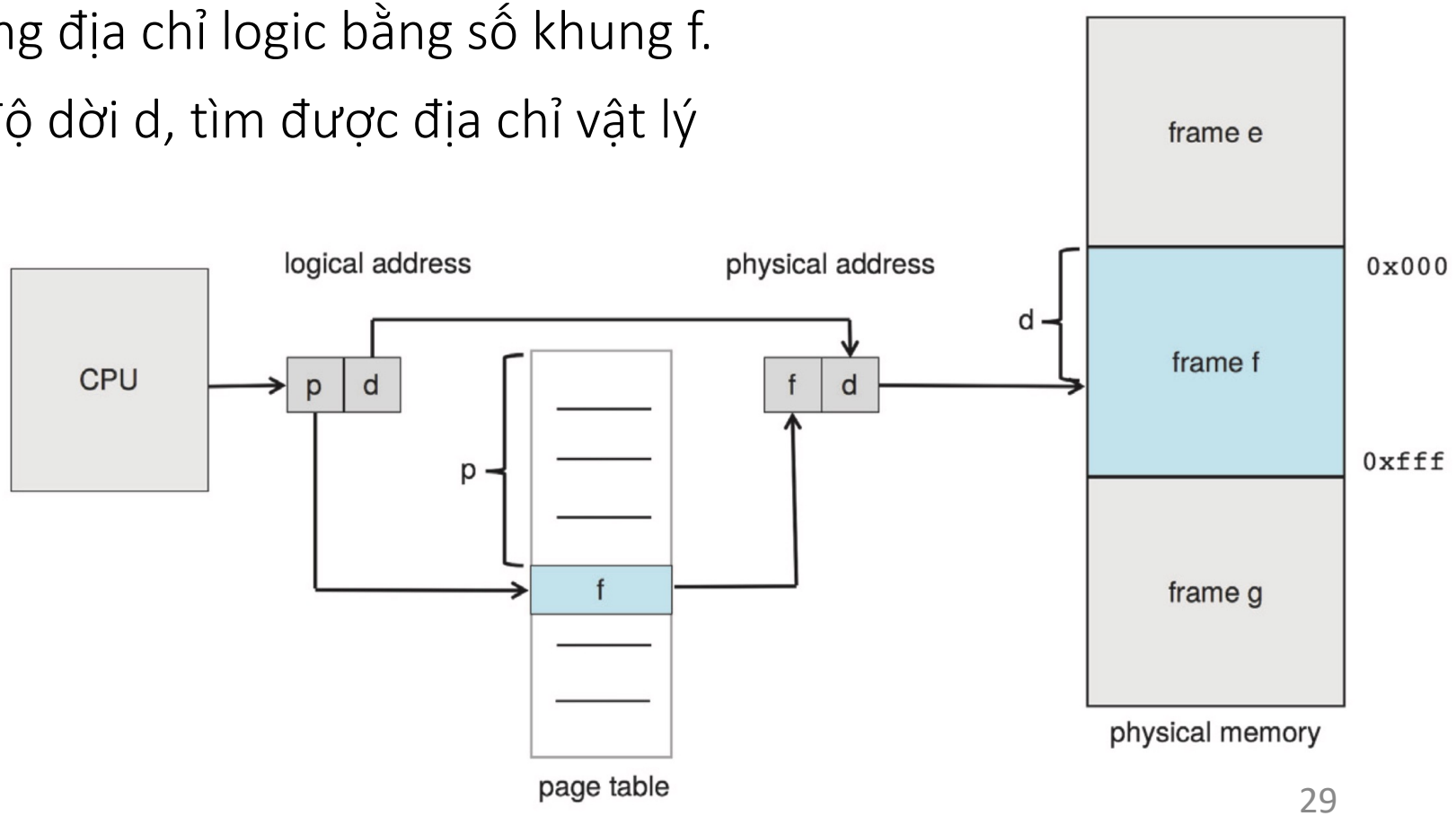
Cấp phát bộ nhớ không liên tục (Non-Contiguous Memory)

- Phân trang (Paging):



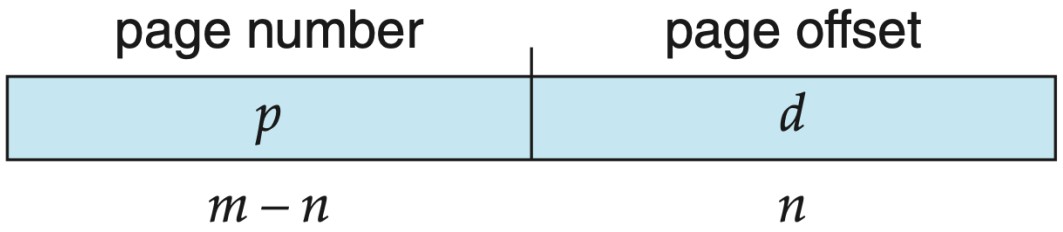
- Phân trang (Paging): địa chỉ được tạo ra bởi CPU gồm có 2 phần
  - Chỉ số trang (page number): p – được dùng như chỉ số trong bảng trang (page table). Bảng trang chứa địa chỉ cơ sở (bắt đầu) của mỗi frame trong bộ nhớ vật lý.
  - Page offset: d - kết hợp với địa chỉ cơ sở, cho biết độ dời trong frame được chọn, để định ra không gian địa chỉ vật lý.

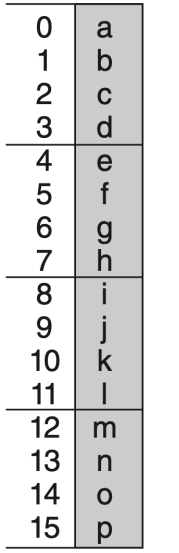
- Phân trang (Paging): MMU thực hiện các bước để chuyển địa chỉ logic → vật lý:
  - Lấy số trang p và dùng nó như chỉ số để dò trong bảng trang.
  - Dựa vào đó, tìm thấy frame f trong bảng trang.
  - Thay thế số trang p trong địa chỉ logic bằng số khung f.
  - Kết hợp số khung f và độ dời d, tìm được địa chỉ vật lý



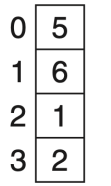
Cấp phát bộ nhớ không liên tục (Non-Contiguous Memory)

- Phân trang (Paging): chuyển đổi địa chỉ trong phân trang
  - Nếu không gian logic có kích thước là 2^m bytes
  - Kích thước của mỗi trang là 2^n bytes
  - Thì số lượng trang là: 2^{m-n}

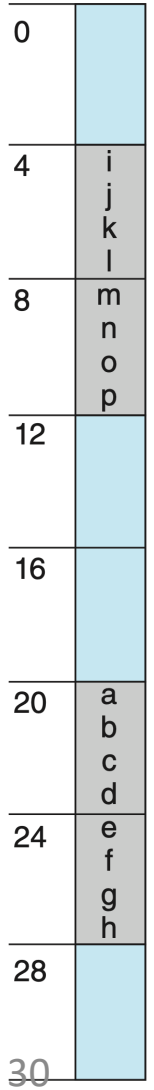




logical memory



page table



physical memory

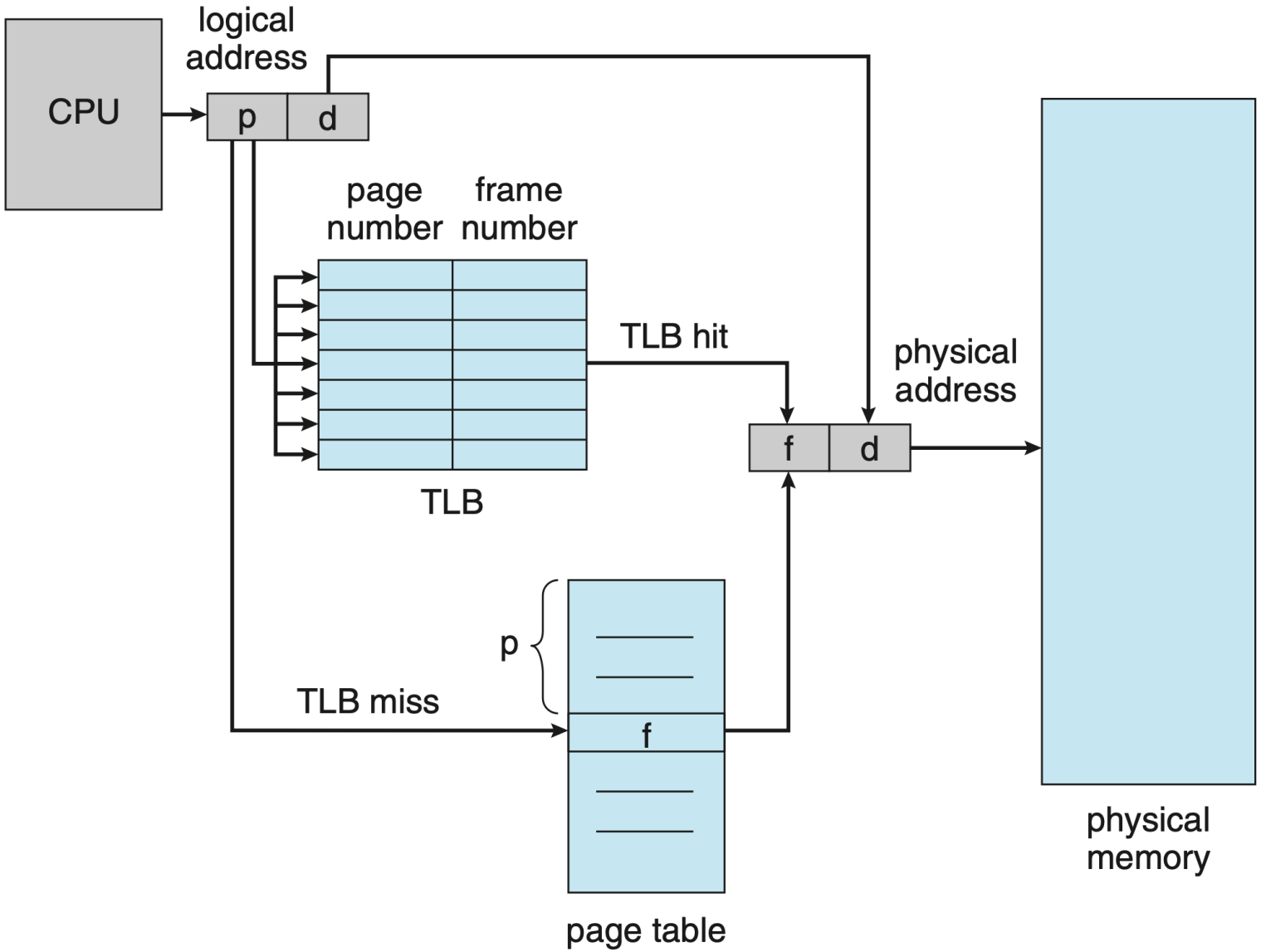
- Ví dụ hình bên:
  - Không gian logic m=4, n=2 → Có 4 trang, kích thước mỗi trang là 4 bytes.
  - Cho bộ nhớ vật lý có kích thước 32 bytes (8 pages).
  - Chuyển đổi địa chỉ:
    - ✓ Địa chỉ logic 5 (tương ứng page 1, offset 1) → địa chỉ vật lý 25 = 6*4 + 1.
    - ✓ Địa chỉ logic 14 (tương ứng page 3, offset 2) → địa chỉ vật lý 10 = 2*4 + 2.

Cấp phát bộ nhớ không liên tục (Non-Contiguous Memory)

- Phân trang (Paging): cài đặt bảng phân trang
  - Bảng trang được đặt trong bộ nhớ.
  - Page-table base register (PTBR): chỉ đến bảng trang.
  - Page-table length register (PTLT): cho biết kích thước của bảng trang.
  - Với mô hình này, mọi truy cập dữ liệu hoặc chỉ thị lệnh đều đòi hỏi 2 lần truy cập bộ nhớ: 1 cho bảng trang và 1 cho chỉ thị/dữ liệu → chậm.
  - Giải quyết 2 lần truy cập bộ nhớ bằng cách sử dụng một bộ đệm phần cứng tra cứu nhanh đặc biệt, được gọi là Translation look-aside buffers (TLBs)

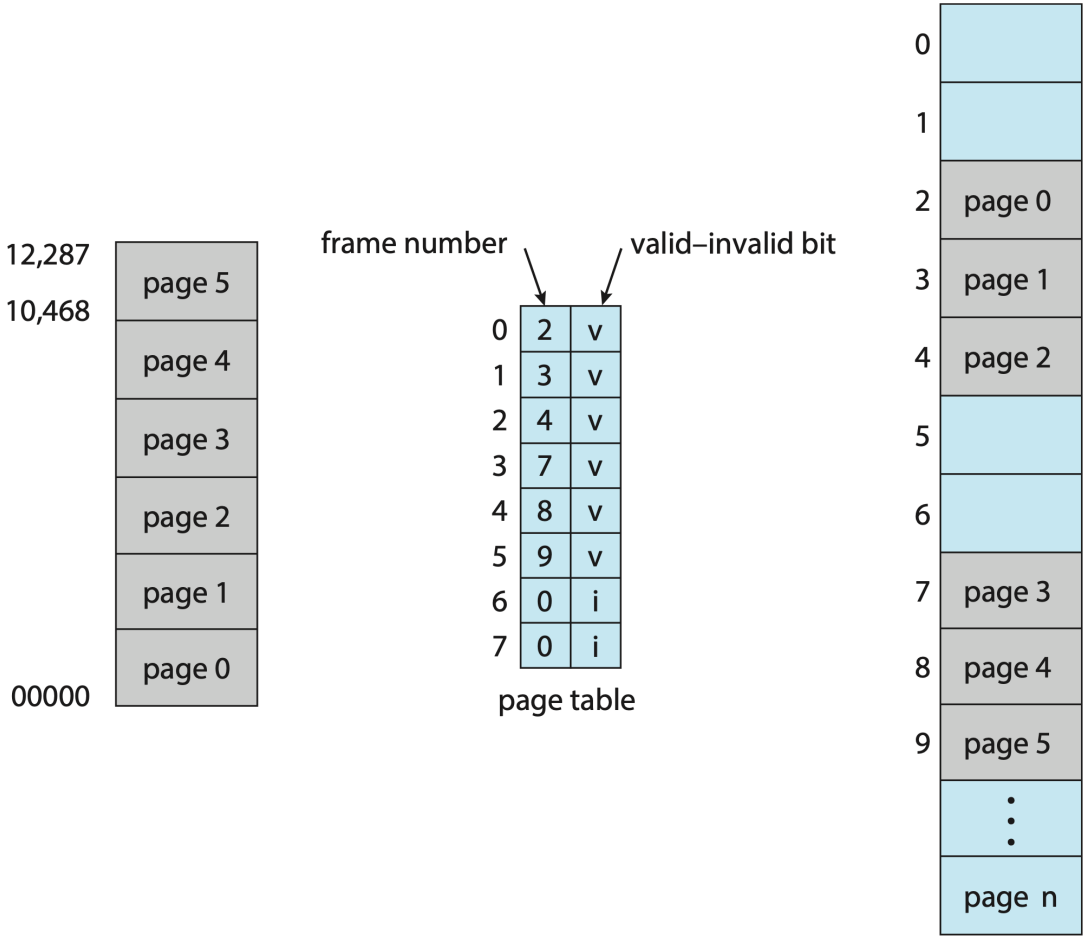
Cấp phát bộ nhớ không liên tục (Non-Contiguous Memory)

- Phân trang (Paging): cài đặt bảng phân trang
  - TLB tìm kiếm dữ liệu với tốc độ cực nhanh.
  - Nếu tìm thấy page number trong TLB → trả về địa chỉ frame mà không cần truy cập bảng phân trang.
  - Nếu không tìm thấy (miss) → truy cập bảng phân trang như thông thường, đồng thời page number sẽ được thêm vào TLB cho lần sau.



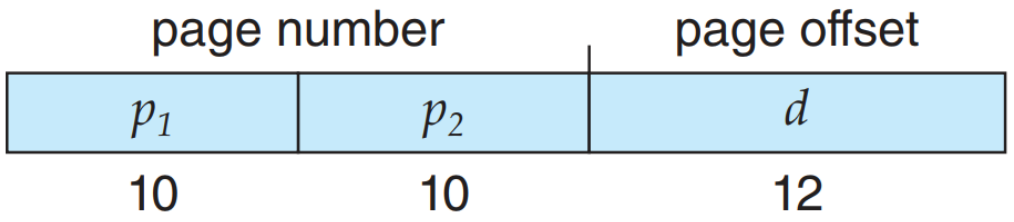
Cấp phát bộ nhớ không liên tục (Non-Contiguous Memory)

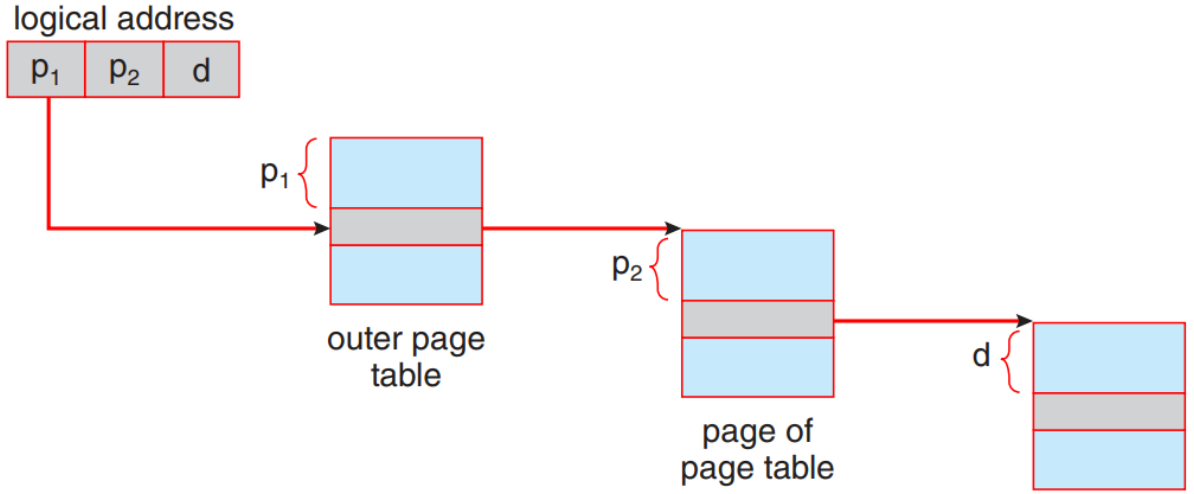
- Phân trang (Paging): bảo vệ vùng nhớ → làm cách nào biết trang nào của tiến trình nào? Làm cách nào cấm tiến trình truy xuất vào trang không phải của nó?
  - Cài đặt bằng cách liên kết một khung với một bit kiểm tra valid/invalid.
  - Bit valid/invalid được đính kèm vào mỗi ô trong bảng trang
    - ✓ Valid: trang đi kèm nằm trong không gian địa chỉ logic của tiến trình, có thể truy xuất
    - ✓ Invalid: trang đi kèm không nằm trong không gian địa chỉ logic.



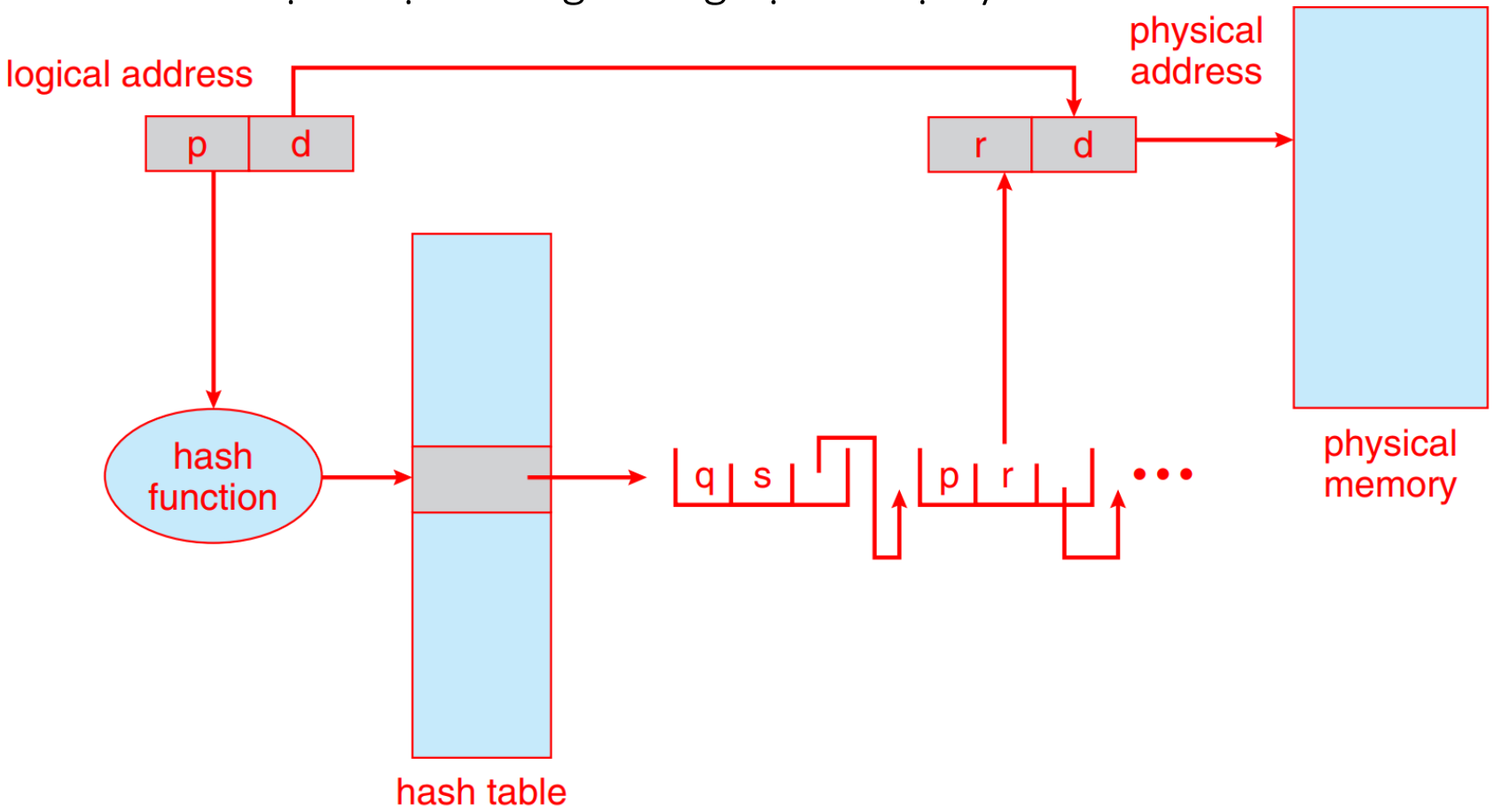
- Phân trang (Paging): cấu trúc của bảng phân trang (page table)
  - Hiện nay có một số kỹ thuật để tổ chức page table trong kỹ thuật phân trang bộ nhớ, bao gồm:
    - ✓ Phân trang phân cấp (hierarchical paging)
    - ✓ Phân trang bằng bảng băm (hashed page table)
    - ✓ Bảng trang nghịch đảo (inverted page table)

- Phân trang (Paging): cấu trúc của bảng phân trang (page table)
  - Phân trang phân cấp:
    - ✓ OS hiện đại hỗ trợ không gian địa chỉ logic lớn (từ 2^{32} đến 2^{64}), nếu page size có kích thước là 4KB (2^{12}) thì page table sẽ có hơn 1 triệu mục ($2^{20}=2^{32}/2^{12}$). Nếu kích thước mỗi mục là 4 bytes → kích thước page table mà mỗi process cần sử dụng là hơn 4MB, tương ứng hơn 4MB bộ nhớ chính bị chiếm dành riêng page table.
    - ✓ Do đó cần chia nhỏ page table, bản thân page table cũng được phân trang, ví dụ:
      - Địa chỉ logic 32 bit (tương tự mô tả trên) sẽ có số bit cho phần trang là 20, phần offset là 12. Vì page table cũng được phân trang nên số bit cho phần trang sẽ được chia ra thành 2 khối, mỗi khối 10 bit như hình dưới đây.

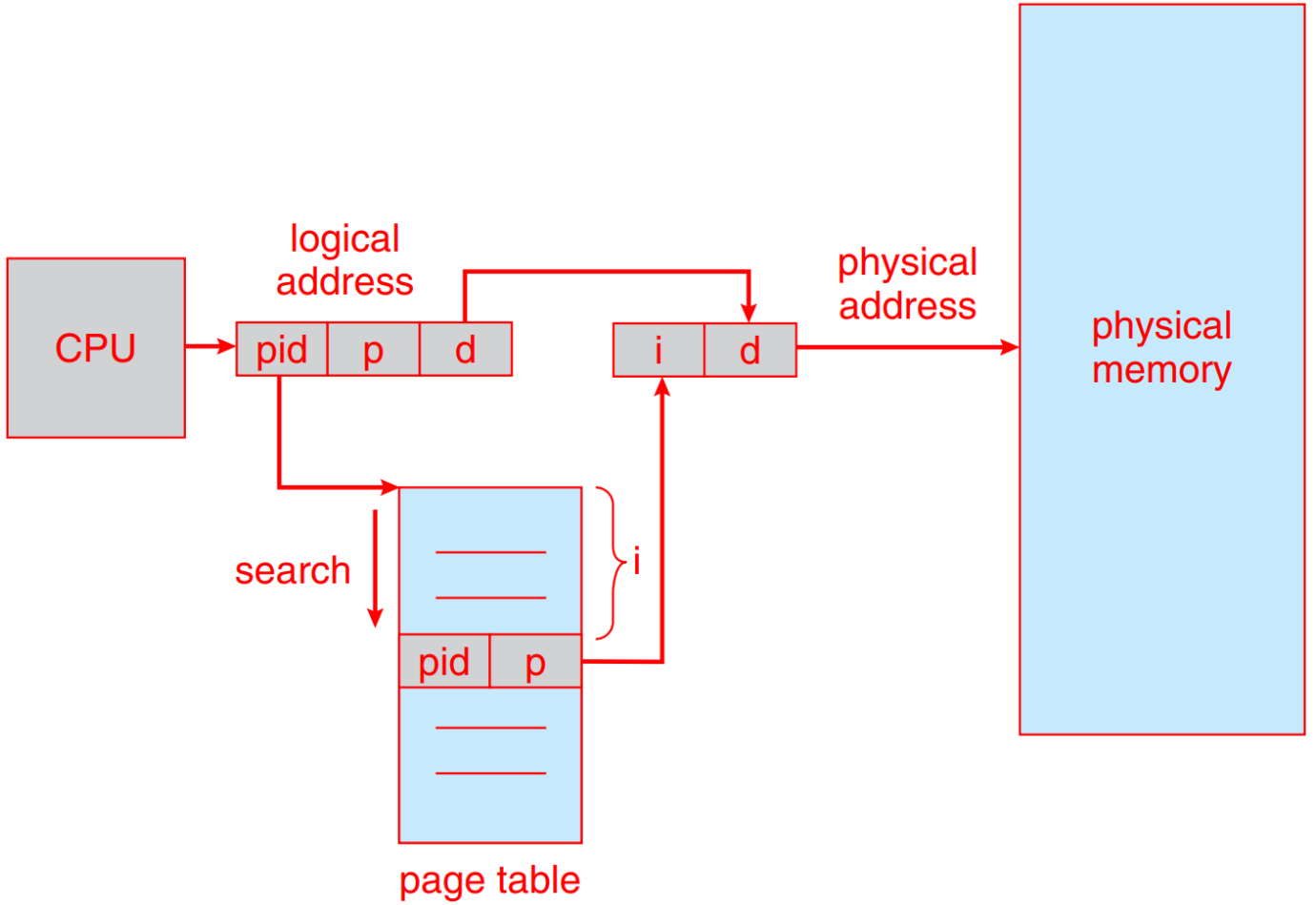




- Phân trang (Paging): cấu trúc của bảng phân trang (page table)
  - Phân trang bằng bảng băm:
    - ✓ Sử dụng hàm băm, bảng băm, danh sách liên kết để tối ưu kích thước, tăng hiệu suất tìm kiếm và ánh xạ từ địa chỉ logic sang địa chỉ vật lý.

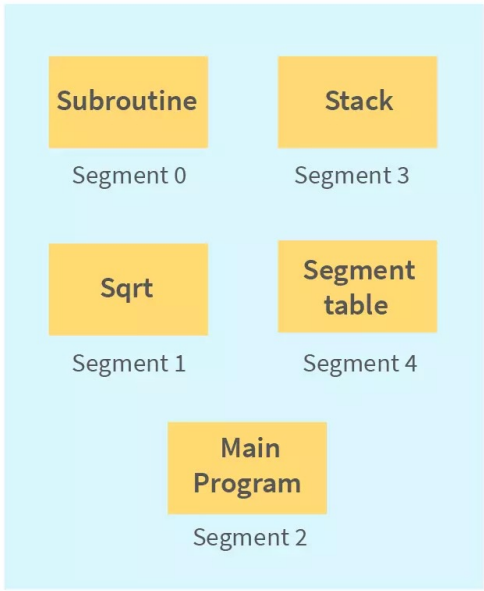


- Phân trang (Paging): cấu trúc của bảng phân trang (page table)
  - Bảng trang nghịch đảo:
    - ✓ Toàn hệ thống chỉ có 1 page table. Mỗi mục trong page table tương ứng với 1 frame trên bộ nhớ vật lý.
    - ✓ Giải pháp giảm lượng bộ nhớ dành cho page table nhưng tăng thời gian tìm kiếm và chuyển đổi địa chỉ.
    - ✓ Có thể kết hợp với bảng băm để tăng tốc độ tìm kiếm



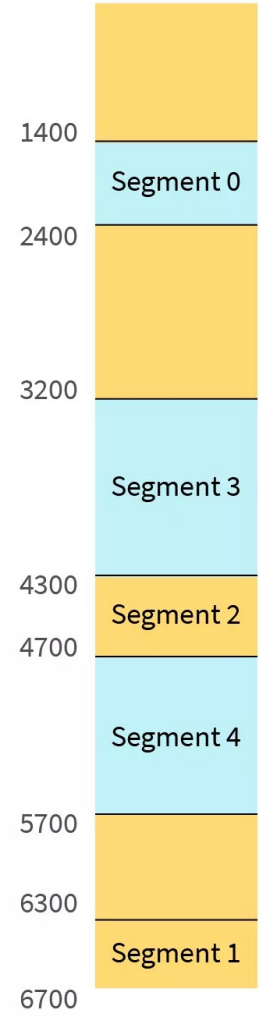
Cấp phát bộ nhớ không liên tục (Non-Contiguous Memory)

- Phân đoạn (Segmentation):
  - Ý tưởng: không gian địa chỉ logic là một tập hợp các phân đoạn (gọi là segment), segment là những phần bộ nhớ có kích thước khác nhau và có liên hệ với nhau. Mỗi segment có một địa chỉ logic gồm 2 phần: chỉ số và độ dài (<segment number; offset>)
  - Bảng phân đoạn (segment table) được sử dụng để lưu trữ thông tin của tất cả các segment thuộc process và giúp chuyển đổi địa chỉ logic → vật lý.



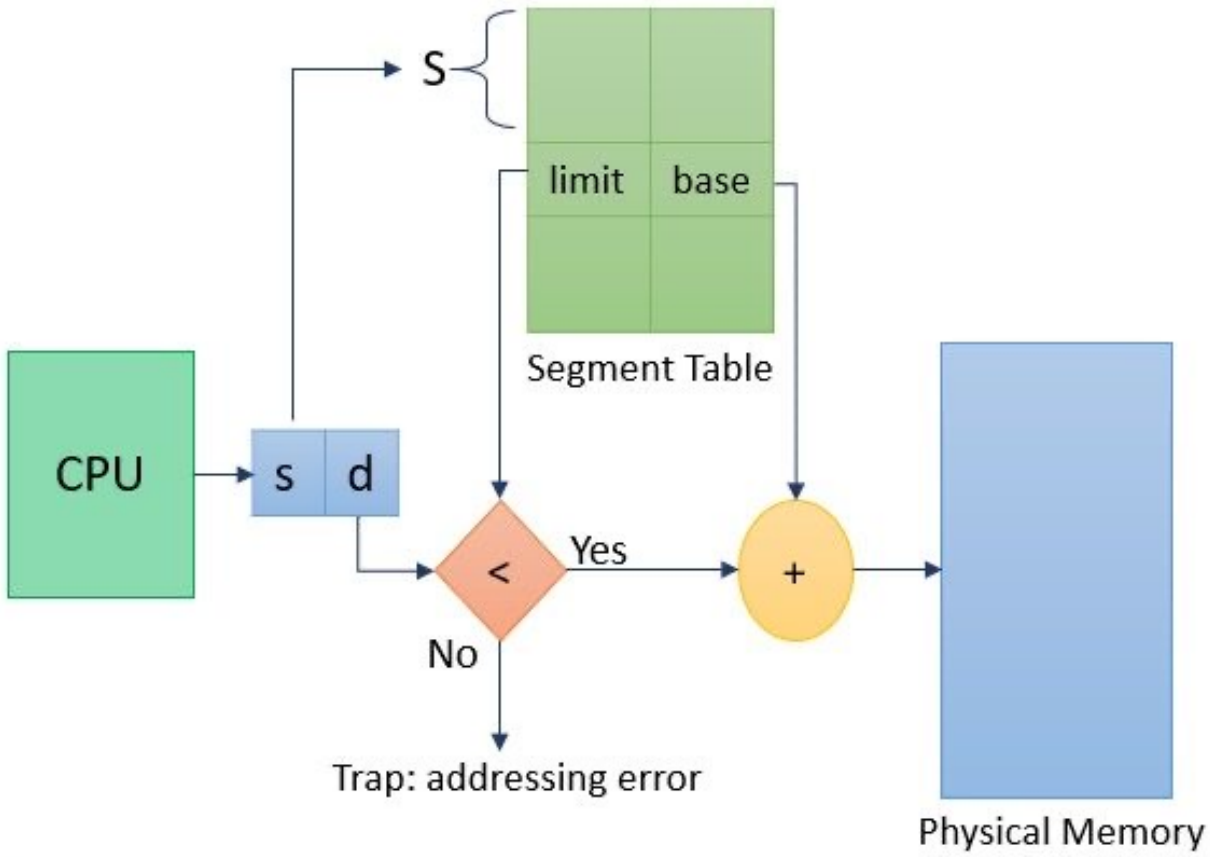
Logical Address Space

	Segment Table	
	Limit	Base
0	1000	1400
1	400	6300
2	400	4300
3	1100	3200
4	1000	4700



Physical Memory

- Phân đoạn (Segmentation):
  - Bảng phân đoạn gồm 2 trường:
    - ✓ Segment base: địa chỉ vật lý bắt đầu của phân đoạn trên bộ nhớ chính.
    - ✓ Segment limit: độ dài của segment



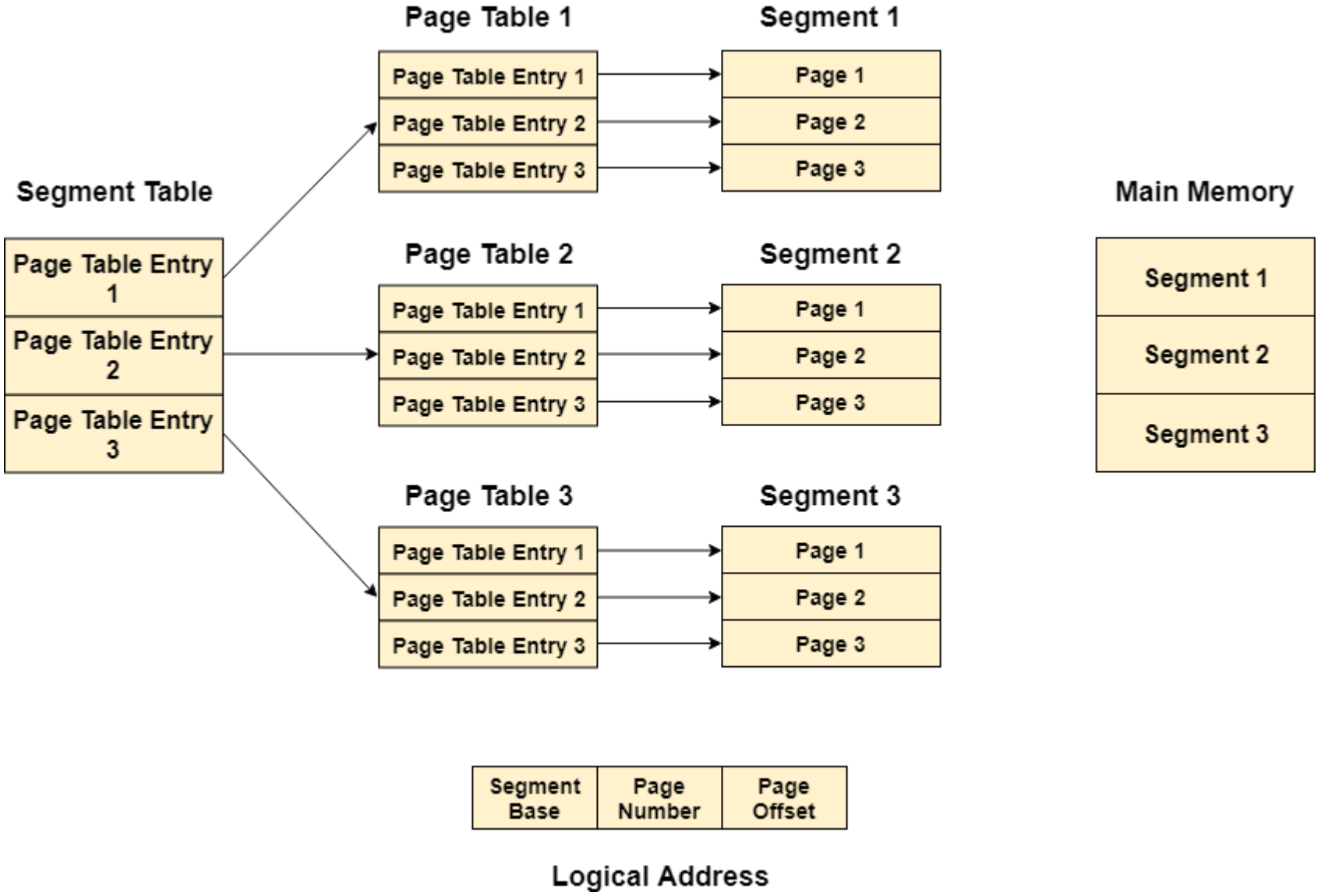
So sánh phân đoạn và phân trang

Phân trang	Phân đoạn
Cấp phát bộ nhớ không liên tục	Cấp phát bộ nhớ không liên tục
Chia chương trình thành các page có kích thước cố định	Chia chương trình thành các segment có kích thước khác nhau.
Hệ điều hành chịu trách nhiệm chủ yếu	Có sự tham gia của compiler
Phân trang nhanh hơn phân đoạn	Phân đoạn chậm hơn phân trang
Phân trang gần với góc nhìn của OS	Phân đoạn gần với góc nhìn của người dùng
Bị phân mảnh nội, không phân mảnh ngoại	Bị phân mảnh ngoại, không bị phân mảnh nội
Địa chỉ logic: <page number;page offset>	Địa chỉ logic: <segment number; segment offset>
Sử dụng bảng phân trang	Sử dụng bảng phân đoạn

- Kết hợp phân trang và phân đoạn (Segmentation with paging)
  - Vấn đề của phân đoạn: một segment có thể không nạp được vào bộ nhớ do phân mảnh ngoại.
  - Giải pháp: phân trang các segment, cho phép các page được nạp vào các frame không liên tục.
  - Cách xây dựng: mỗi process có một segment table và nhiều segment page (mỗi segment có một segment page).
  - Mỗi địa chỉ logic có dạng: <segment number; page number; page offset>

Cấp phát bộ nhớ không liên tục (Non-Contiguous Memory)

- Kết hợp phân trang và phân đoạn (Segmentation with paging)



Tham khảo

- Operating System Concepts
- Tài liệu Hệ điều hành – ThS. Lương Minh Huấn.
- https://www.javatpoint.com/os-memory-management-introduction
- https://www.studytonight.com/operating-system/segmentation-in-operating-systems
- https://www.prepbytes.com/blog/operating-system/difference-between-internal-and-external-fragmentation/
- https://www.geeksforgeeks.org/memory-management-in-operating-system/
- https://www.geeksforgeeks.org/segmentation-in-operating-system/