Chương 7. Quản lý bộ nhớ

- Khái niệm cơ sở
- Các kiểu địa chỉ nhớ (physical address , logical address)
- Chuyển đổi địa chỉ nhớ
- Overlay và swapping
- Mô hình quản lý bộ nhớ đơn giản
  - Fixed partitioning
  - Dynamic partitioning
  - Cơ chế phân trang (paging)
  - Cơ chế phân đoạn (segmentation)
  - Segmentation with paging

Khái niệm cơ sở

- Chương trình phải được mang vào trong bộ nhớ và đặt nó trong một tiến trình để được xử lý
- Input Queue – Một tập hợp của những tiến trình trên đĩa mà đang chờ để được mang vào trong bộ nhớ để thực thi.
- User programs trải qua nhiều bước trước khi được xử lý.

Khái niệm cơ sở

- Quản lý bộ nhớ là công việc của hệ điều hành với sự hỗ trợ của phần cứng nhằm phân phối, sắp xếp các process trong bộ nhớ sao cho hiệu quả.
- Mục tiêu cần đạt được là nạp càng nhiều process vào bộ nhớ càng tốt (gia tăng mức độ đa chương)
- Trong hầu hết các hệ thống, kernel sẽ chiếm một phần cố định của bộ nhớ; phần còn lại phân phối cho các process.
- Các yêu cầu đối với việc quản lý bộ nhớ
  - Cấp phát bộ nhớ cho các process
  - Tái định vị (relocation): khi swapping,…
  - Bảo vệ: phải kiểm tra truy xuất bộ nhớ có hợp lệ không
  - Chia sẻ: cho phép các process chia sẻ vùng nhớ chung
  - Kết gán địa chỉ nhớ luận lý của user vào địa chỉ thực

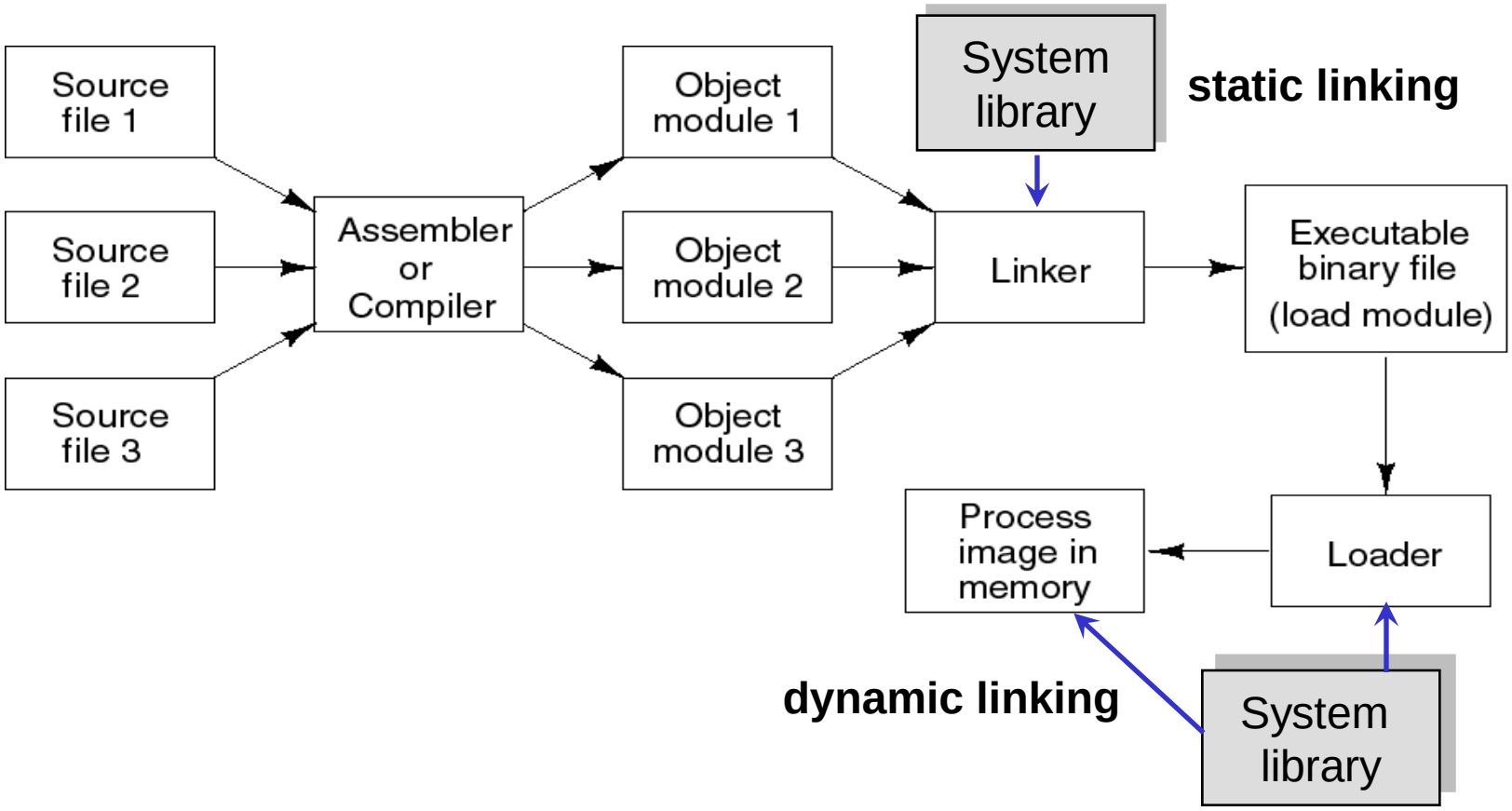
Các kiểu địa chỉ nhớ

- *Địa chỉ vật lý* (physical address) (địa chỉ *thực*) là một vị trí thực trong bộ nhớ chính.

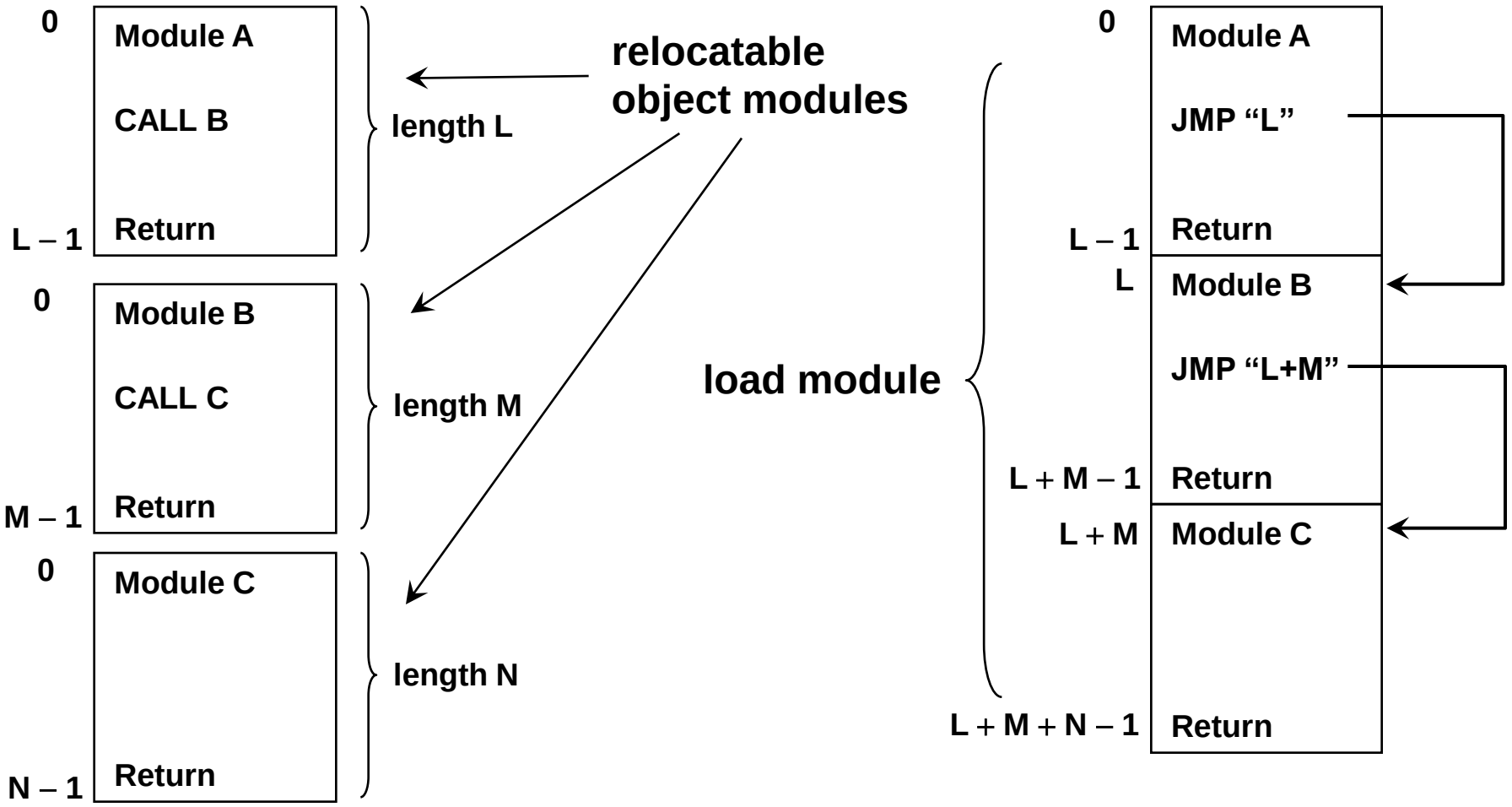
- *Địa chỉ luận lý* (logical address) là một vị trí nhớ được diễn tả trong một chương trình (còn gọi là địa chỉ ảo virtual address)
  - Các trình biên dịch (compiler) tạo ra mã lệnh chương trình mà trong đó mọi tham chiếu bộ nhớ đều là địa chỉ luận lý
  - *Địa chỉ tương đối* (relative address) (địa chỉ *khả tái định vị*, relocatable address) là một kiểu địa chỉ luận lý trong đó các địa chỉ được biểu diễn tương đối so với một vị trí xác định nào đó trong chương trình.
    - Ví dụ: 12 byte so với vị trí bắt đầu chương trình,…
  - *Địa chỉ tuyệt đối* (absolute address): địa chỉ tương đương với địa chỉ thực.

Nạp chương trình vào bộ nhớ

- Bộ linker: kết hợp các object module thành một file nhị phân khả thực thi gọi là *load module*.
- Bộ loader: nạp load module vào bộ nhớ chính

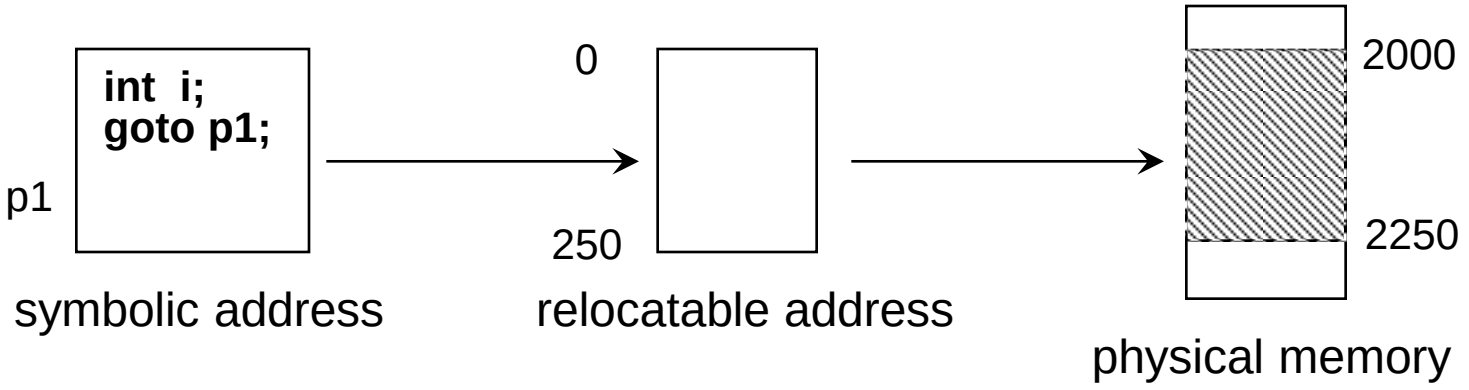


Cơ chế thực hiện linking



Chuyển đổi địa chỉ

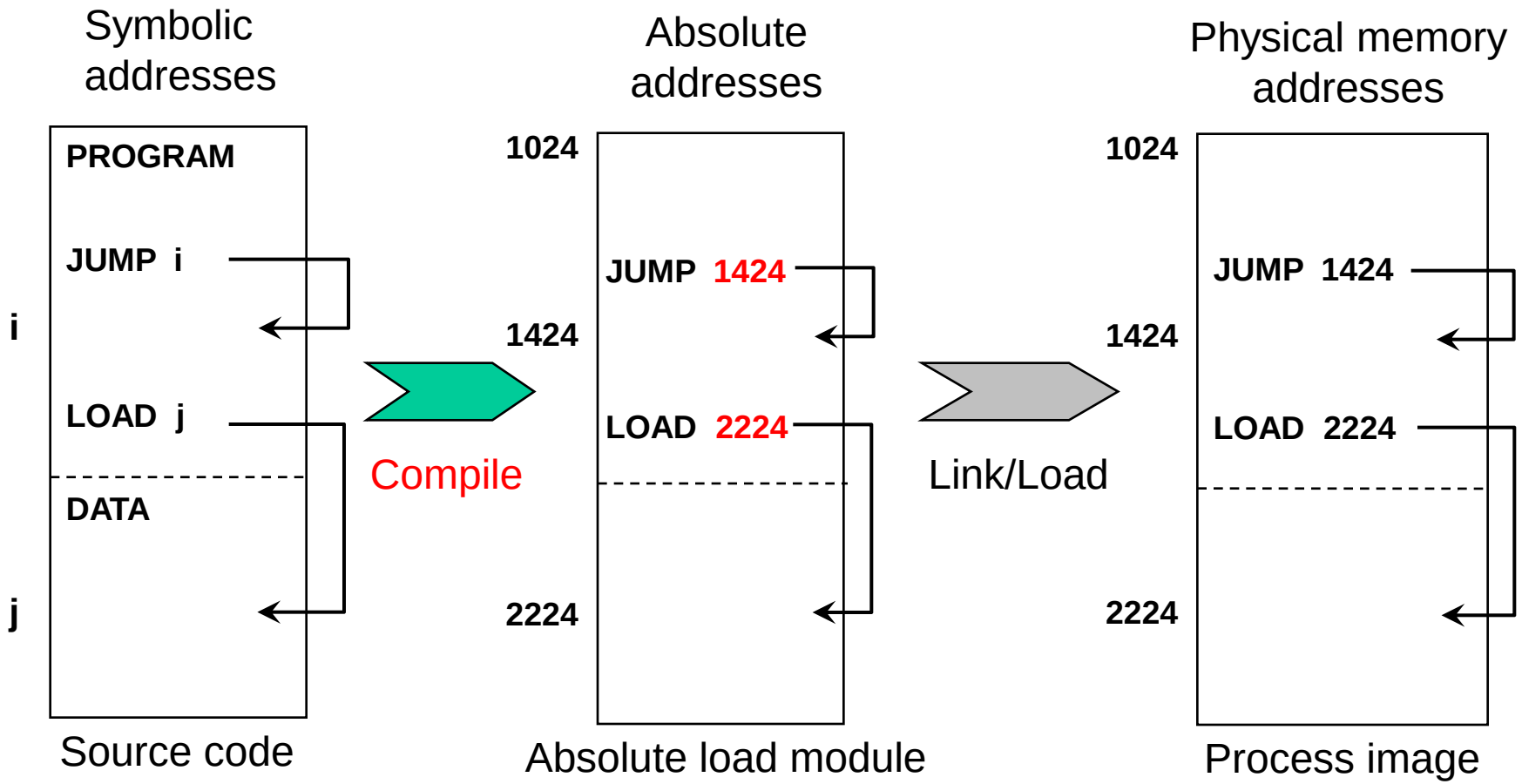
- *Chuyển đổi địa chỉ:* quá trình ánh xạ một địa chỉ từ không gian địa chỉ này sang không gian địa chỉ khác.
- Biểu diễn địa chỉ nhớ
  - Trong source code: symbolic (các biến, hằng, pointer,…)
  - Thời điểm biên dịch: thường là địa chỉ khả tái định vị
    - Ví dụ: a ở vị trí 14 bytes so với vị trí bắt đầu của module.
  - Thời điểm linking/loading: có thể là địa chỉ thực. Ví dụ: dữ liệu nằm tại địa chỉ bộ nhớ thực 2030



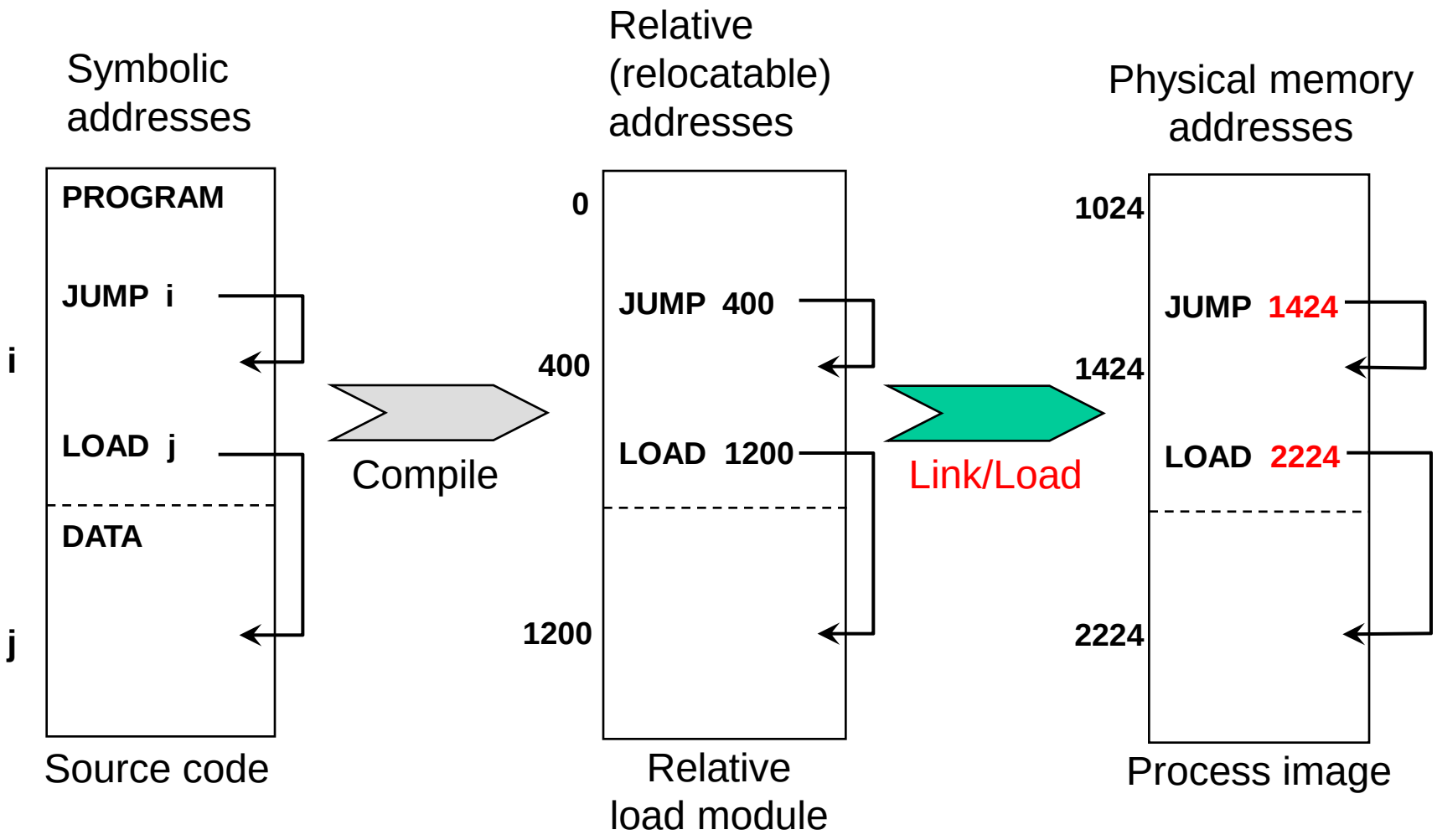
Chuyển đổi địa chỉ (tt)

- Địa chỉ lệnh (instruction) và dữ liệu (data) được chuyển đổi thành địa chỉ thực có thể xảy ra tại ba thời điểm khác nhau
  - Compile time: nếu biết trước địa chỉ bộ nhớ của chương trình thì có thể kết gán địa chỉ tuyệt đối lúc biên dịch.
    - Ví dụ: chương trình .COM của MS-DOS
    - Khuyết điểm: phải biên dịch lại nếu thay đổi địa chỉ nạp chương trình
  - Load time: Vào thời điểm loading, *loader* phải chuyển đổi địa chỉ khả tái định vị thành địa chỉ thực dựa trên một *địa chỉ nền* (base address).
    - Địa chỉ thực được tính toán vào thời điểm nạp chương trình $\Rightarrow$ phải tiến hành reload nếu địa chỉ nền thay đổi.

Sinh địa chỉ tuyệt đối vào thời điểm dịch

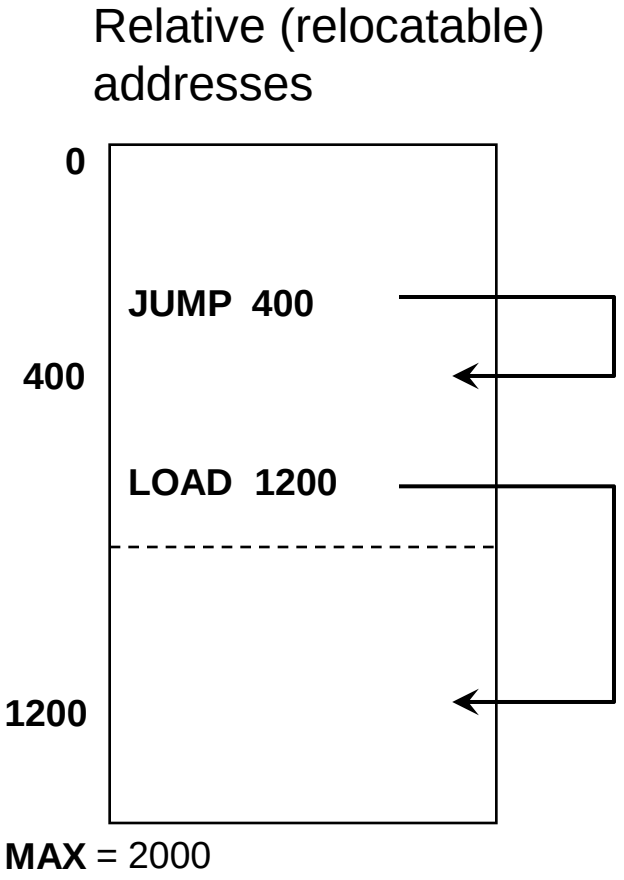


Sinh địa chỉ thực vào thời điểm nạp



Chuyển đổi địa chỉ (tt)

- Execution time: khi trong quá trình thực thi, process có thể được di chuyển từ segment này sang segment khác trong bộ nhớ thì quá trình chuyển đổi địa chỉ được trì hoãn đến thời điểm thực thi
  - Cần sự hỗ trợ của phần cứng cho việc ánh xạ địa chỉ.
    - Ví dụ: trường hợp địa chỉ luận lý là relocatable thì có thể dùng thanh ghi base và limit,…
  - Sử dụng trong đa số các OS đa dụng (general-purpose) trong đó có các cơ chế swapping, paging, segmentation

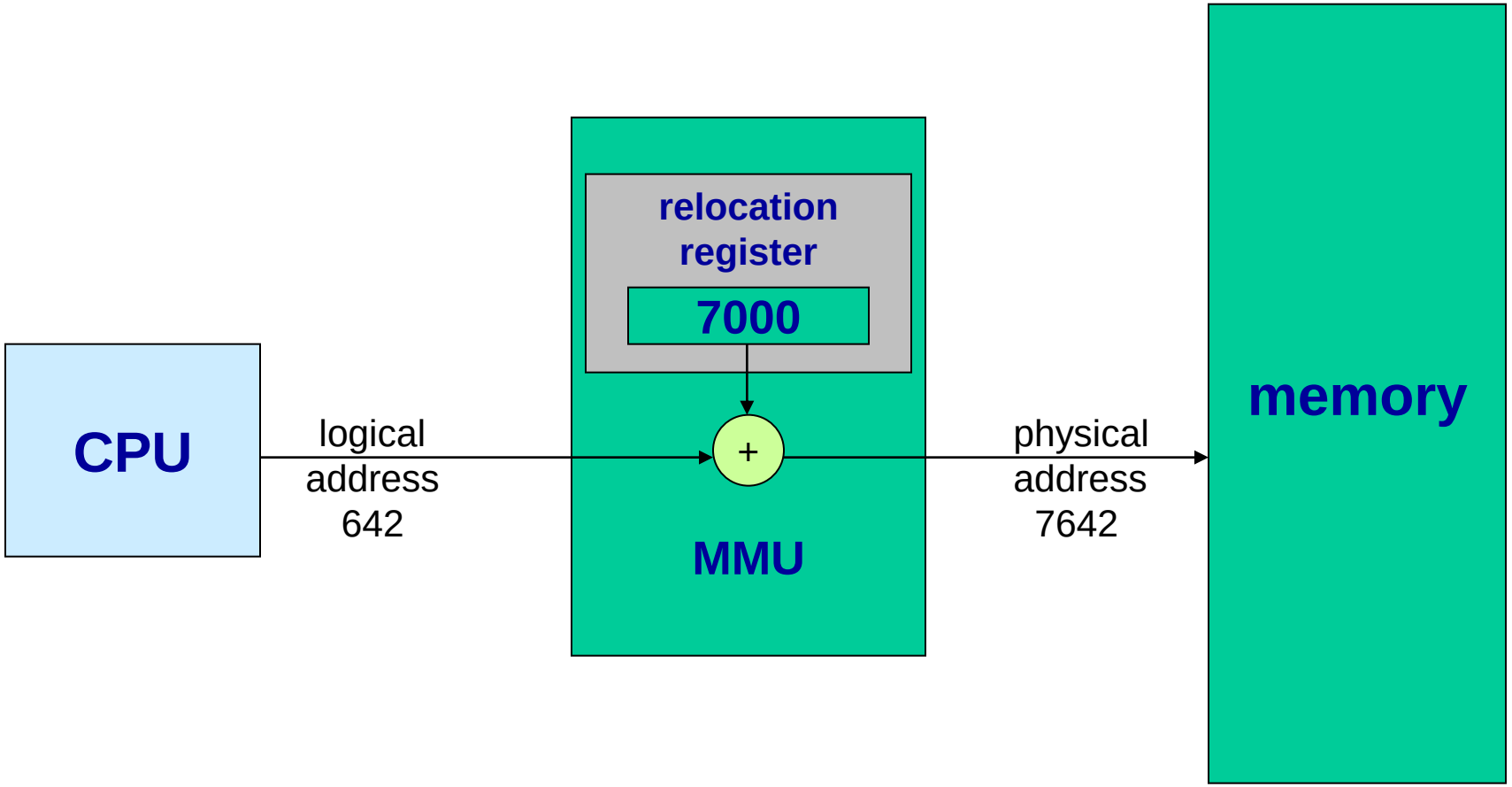


Không gian địa chỉ

- Địa chỉ được tạo bởi CPU – Địa chỉ logic (logical address). Tập hợp địa chỉ logic gọi là không gian địa chỉ logic
- Địa chỉ nạp vào MAR – địa chỉ vật lý (physical address). Tập hợp địa chỉ vật lý gọi là không gian địa chỉ vật lý
- **compile-time** and **load-time:**
  - Địa chỉ Logical và physical là xác định
- **Tại thời điểm thực thi:**

  địa chỉ logic khác vật lý, thường gọi là địa chỉ ảo
- Việc ánh xạ giữa hai địa chỉ được thực thi bởi *Memory Management Unit* (MMU)

Tái định vị sử dụng relocation register



Liên kết động(Dynamic linking)

- Quá trình link đến một *module ngoài* (external module) được thực hiện sau khi đã tạo xong load module (i.e. file có thể thực thi, executable)
  - Ví dụ trong Windows: module ngoài là các file .DLL còn trong Unix, các module ngoài là các file .so (shared library)
- Load module chứa các stub tham chiếu (refer) đến routine của external module.
  - Lúc thực thi, khi stub được thực thi lần đầu (do process gọi routine lần đầu), stub nạp routine vào bộ nhớ, tự thay thế bằng địa chỉ của routine và routine được thực thi.
  - Các lần gọi routine sau sẽ xảy ra bình thường
- Stub cần sự hỗ trợ của OS (như kiểm tra xem routine đã được nạp vào bộ nhớ chưa).

Ưu điểm của dynamic linking

- Thông thường, external module là một thư viện cung cấp các tiện ích của OS. Các chương trình thực thi có thể dùng các phiên bản khác nhau của external module mà không cần sửa đổi, biên dịch lại.
- *Chia sẻ mã* (code sharing): một external module chỉ cần nạp vào bộ nhớ một lần. Các process cần dùng external module này thì cùng chia sẻ đoạn mã của external module $\Rightarrow$ tiết kiệm không gian nhớ và đĩa.
- Phương pháp dynamic linking cần sự hỗ trợ của OS trong việc kiểm tra xem một thủ tục nào đó có thể được chia sẻ giữa các process hay là phần mã của riêng một process (bởi vì chỉ có OS mới có quyền thực hiện việc kiểm tra này).

Nạp động(Dynamic loading)

- **Cơ chế**: chỉ khi nào cần được gọi đến thì một thủ tục mới được nạp vào bộ nhớ chính $\Rightarrow$ tăng độ hiệu dụng của bộ nhớ (memory utilization) bởi vì các thủ tục không được gọi đến sẽ không chiếm chỗ trong bộ nhớ

- Rất hiệu quả trong trường hợp tồn tại khối lượng lớn mã chương trình có tần suất **sử dụng thấp**, không được sử dụng thường xuyên (ví dụ các thủ tục xử lý lỗi)

- Hỗ trợ từ hệ điều hành
  - Thông thường, user chịu trách nhiệm thiết kế và hiện thực các chương trình có dynamic loading.
  - Hệ điều hành chủ yếu cung cấp một số thủ tục thư viện hỗ trợ, tạo điều kiện dễ dàng hơn cho lập trình viên.

Cơ chế phủ lắp (overlay)

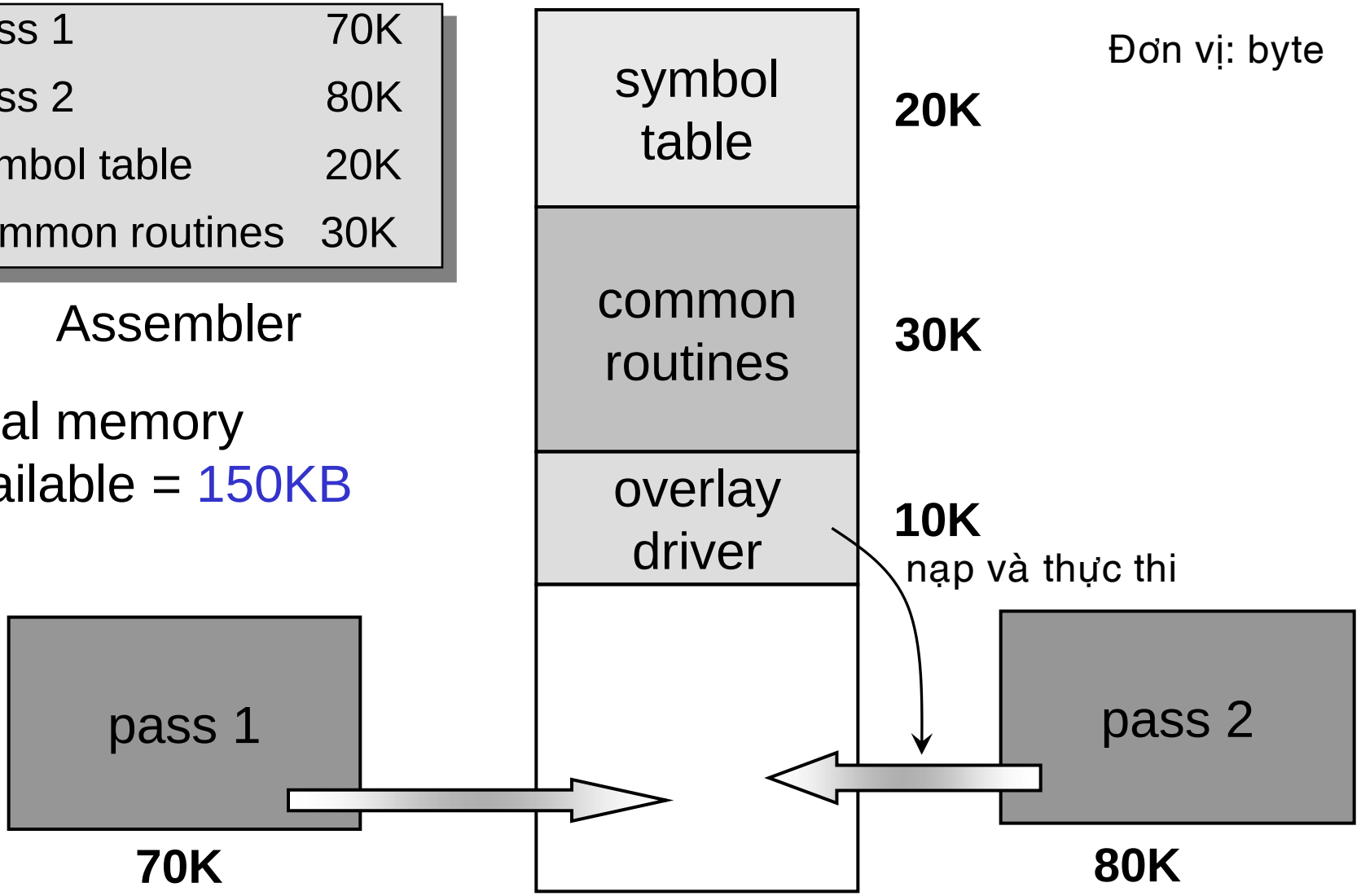
- Tại mỗi thời điểm, chỉ giữ lại trong bộ nhớ những lệnh hoặc dữ liệu cần thiết, giải phóng các lệnh/dữ liệu chưa hoặc không cần dùng đến.
- Cơ chế này rất hữu dụng khi kích thước một process lớn hơn không gian bộ nhớ cấp cho process đó.
- Cơ chế này được điều khiển bởi người sử dụng (thông qua sự hỗ trợ của các thư viện lập trình) chứ không cần sự hỗ trợ của hệ điều hành

Cơ chế overlay (tt)

Pass 1	70K
Pass 2	80K
Symbol table	20K
Common routines	30K

Assembler

Total memory available = 150KB



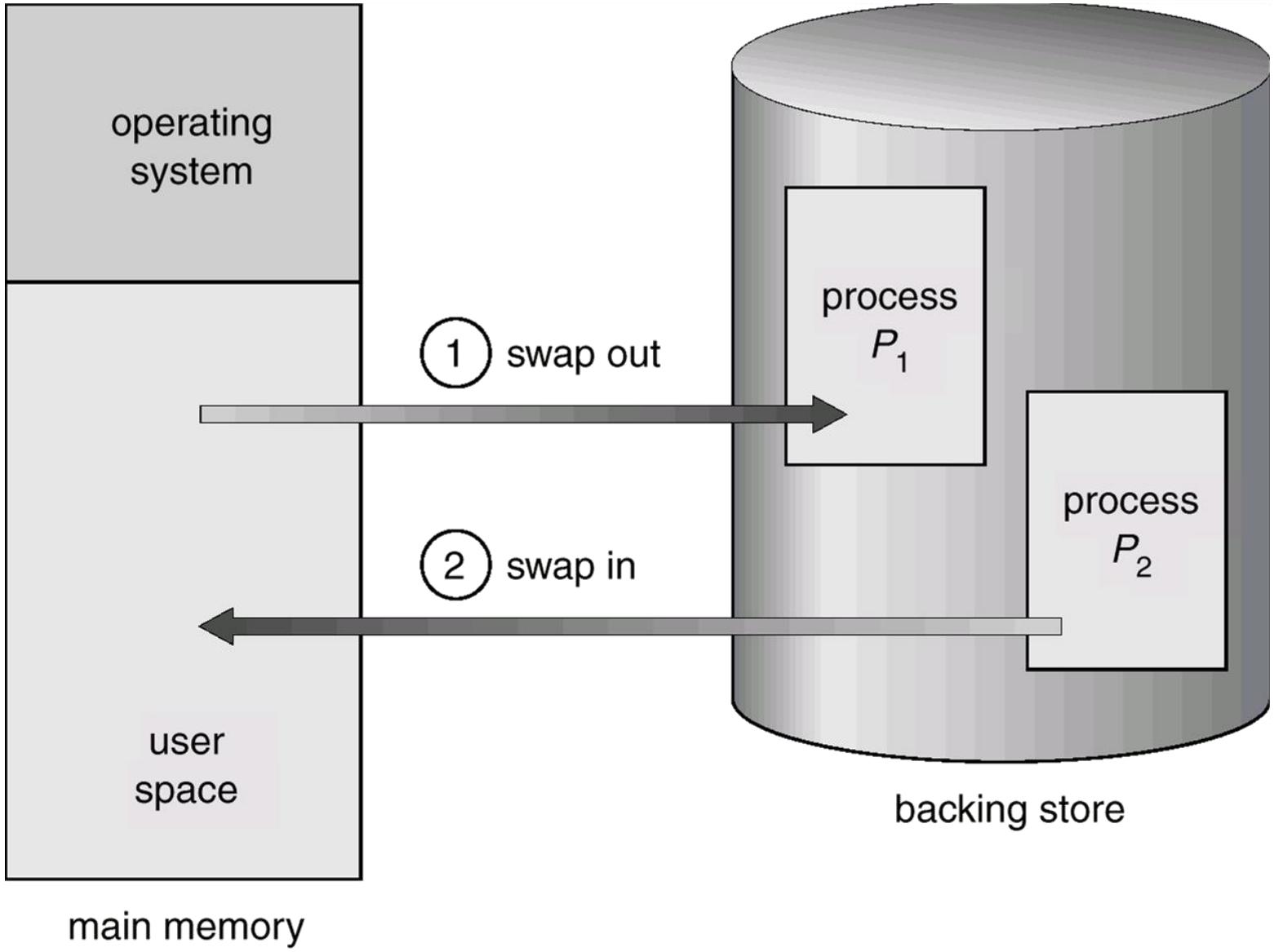
Cơ chế hoán vị (swapping)

- Một process có thể tạm thời bị swap ra khỏi bộ nhớ chính và lưu trên một hệ thống lưu trữ phụ. Sau đó, process có thể được nạp lại vào bộ nhớ để tiếp tục quá trình thực thi.

  Swapping policy: hai ví dụ

  - *Round-robin*: swap out P_1 (vừa tiêu thụ hết quantum của nó), swap in P_2 , thực thi P_3 ,…
  - *Roll out, roll in*: dùng trong cơ chế định thời theo độ ưu tiên (priority-based scheduling)
    - Process có độ ưu tiên thấp hơn sẽ bị swap out nhường chỗ cho process có độ ưu tiên cao hơn mới đến được nạp vào bộ nhớ để thực thi
- Hiện nay, ít hệ thống sử dụng cơ chế swapping trên

Minh họa cơ chế swapping



Mô hình quản lý bộ nhớ

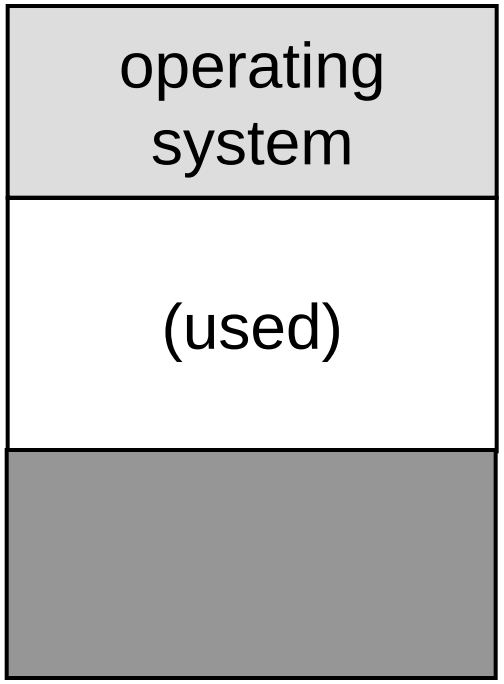
- Trong chương này, mô hình quản lý bộ nhớ là một mô hình đơn giản, không có bộ nhớ ảo.
- Một process phải được nạp hoàn toàn vào bộ nhớ thì mới được thực thi (ngoại trừ khi sử dụng cơ chế overlay).
- Các cơ chế quản lý bộ nhớ sau đây rất ít (hầu như không còn) được dùng trong các hệ thống hiện đại
  - Phân chia cố định (fixed partitioning)
  - Phân chia động (dynamic partitioning)
  - Phân trang đơn giản (simple paging)
  - Phân đoạn đơn giản (simple segmentation)

Phân mảnh (fragmentation)

- *Phân mảnh ngoại* (external fragmentation)
  - Kích thước không gian nhớ còn trống đủ để thỏa mãn một yêu cầu cấp phát, tuy nhiên không gian nhớ này không liên tục ⇒ có thể dùng cơ chế *kết khối* (compaction) để gom lại thành vùng nhớ liên tục.

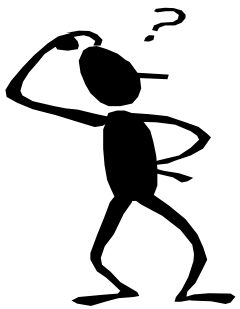
- *Phân mảnh nội* (internal fragmentation)
  - Kích thước vùng nhớ được cấp phát có thể hơi lớn hơn vùng nhớ yêu cầu.
    - Ví dụ: cấp một khoảng trống 18,464 bytes cho một process yêu cầu 18,462 bytes.
  - Hiện tượng phân mảnh nội thường xảy ra khi bộ nhớ thực được chia thành các khối kích thước cố định (fixed-sized block) và các process được cấp phát theo đơn vị khối. Ví dụ: cơ chế phân trang (paging).

Phân mảnh nội

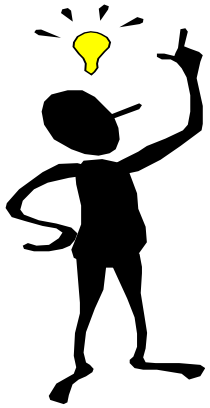


hole kích thước 18,464 bytes

yêu cầu kế tiếp là 18,462 bytes !!!



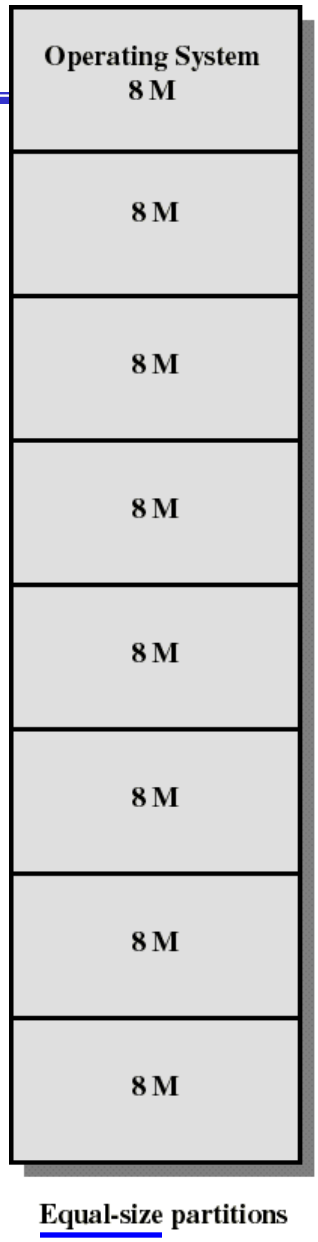
cần quản lý khoảng trống 2 bytes !?!



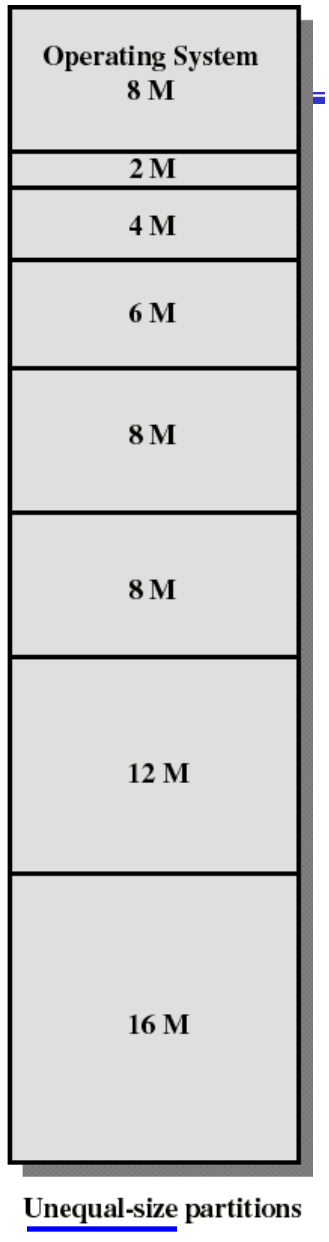
OS sẽ cấp phát hẳn khối 18,464 bytes cho process ⇒ dư ra 2 bytes không dùng!

Fixed partitioning

- Khi khởi động hệ thống, bộ nhớ chính được chia thành nhiều phần rời nhau gọi là các *partition* có kích thước bằng nhau hoặc khác nhau
- Process nào có kích thước nhỏ hơn hoặc bằng kích thước partition thì có thể được nạp vào partition đó.
- Nếu chương trình có kích thước lớn hơn partition thì phải dùng cơ chế overlay.
- Nhận xét
  - Không hiệu quả do bị phân mảnh nội: một chương trình dù lớn hay nhỏ đều được cấp phát trọn một partition.



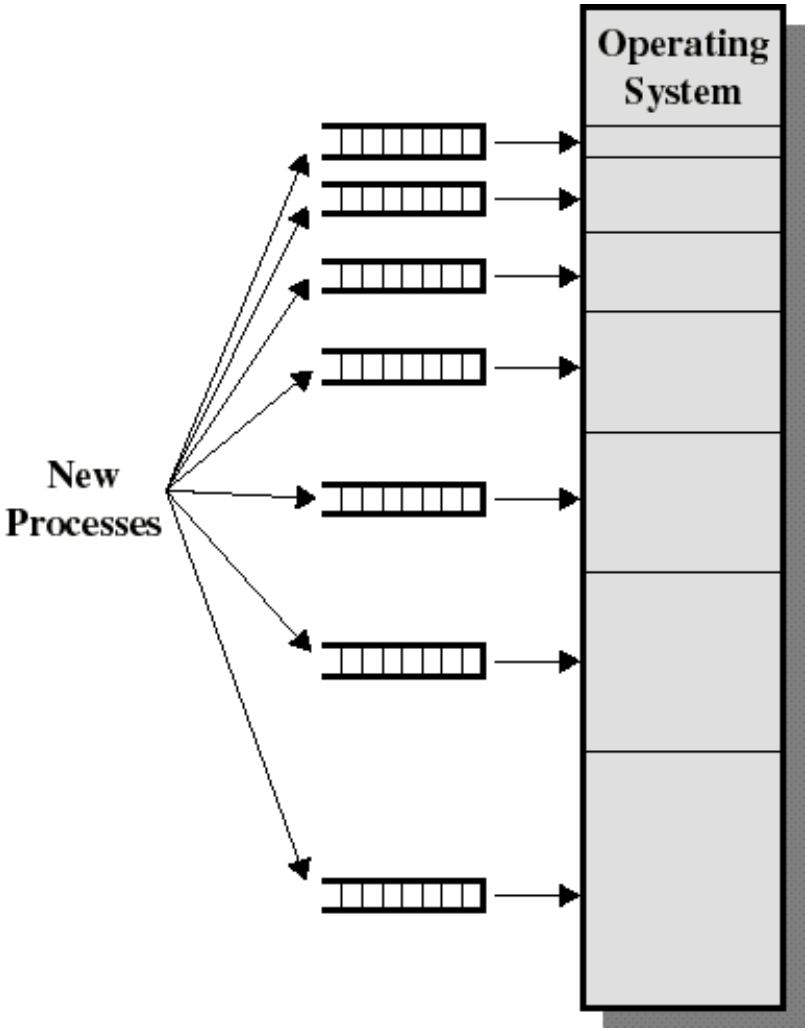
Equal-size partitions



Unequal-size partitions

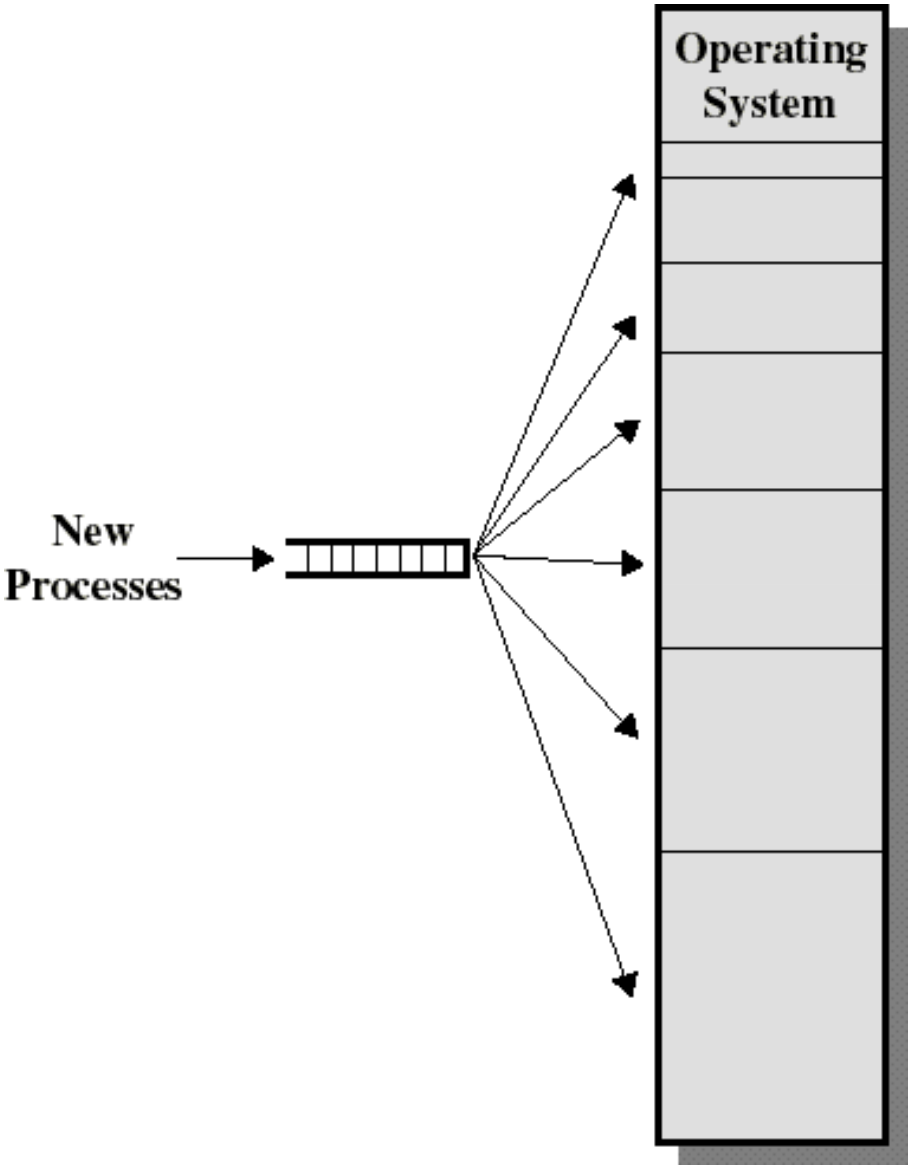
Chiến lược placement (tt)

- Partition có kích thước bằng nhau
  - Nếu còn partition trống ⇒ process mới sẽ được nạp vào partition đó
  - Nếu không còn **partition trống**, nhưng trong đó có process đang bị blocked ⇒ swap process đó ra bộ nhớ phụ nhường chỗ cho process mới.
- Partition có kích thước không bằng nhau: giải pháp 1
  - Gán mỗi process vào partition nhỏ nhất phù hợp với nó
  - Có hàng đợi cho mỗi partition
  - Giảm thiểu phân mảnh nội
  - **Vấn đề**: có thể có một số hàng đợi trống không (vì không có process với kích thước tương ứng) và hàng đợi dày đặc



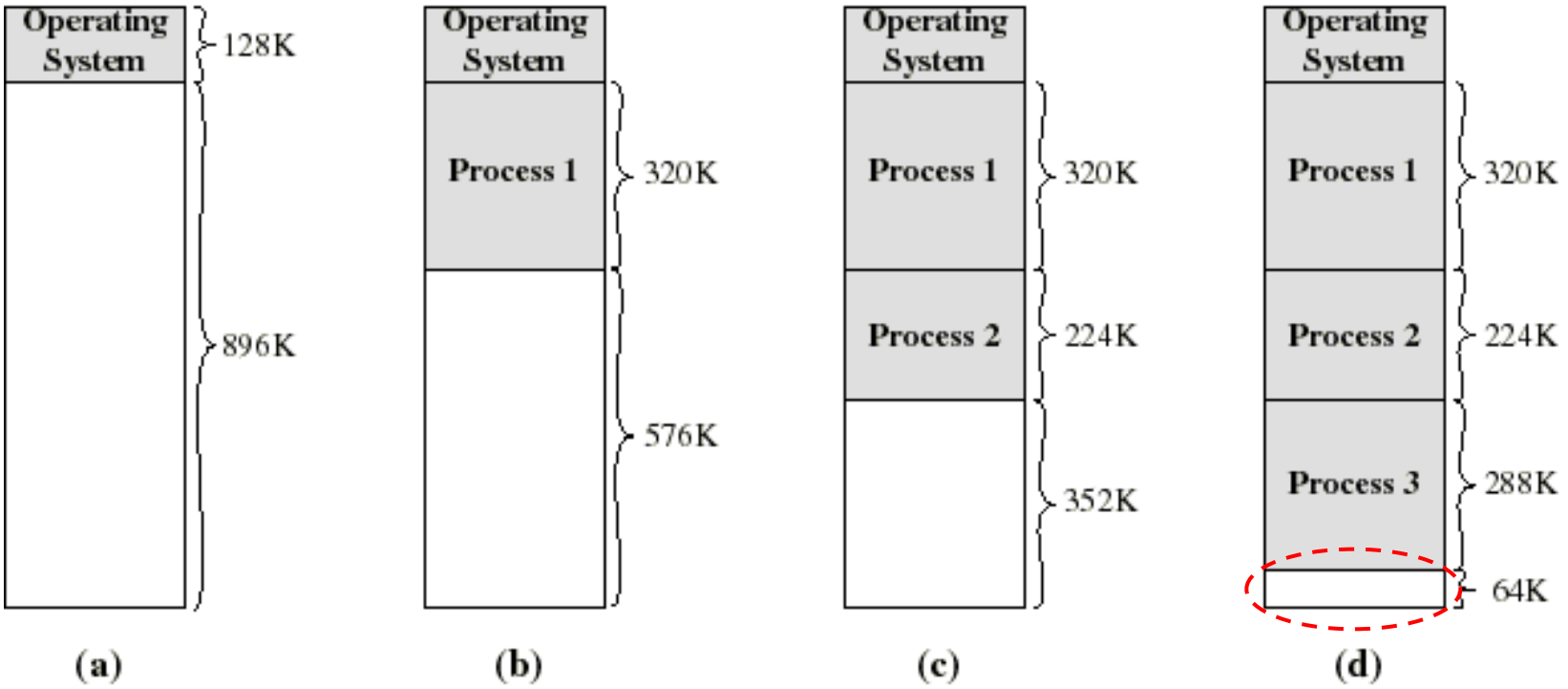
Chiến lược placement (tt)

- Partition có kích thước không bằng nhau: giải pháp 2
  - Chỉ có một hàng đợi chung cho mọi partition
  - Khi cần nạp một process vào bộ nhớ chính ⇒ chọn partition nhỏ nhất còn trống



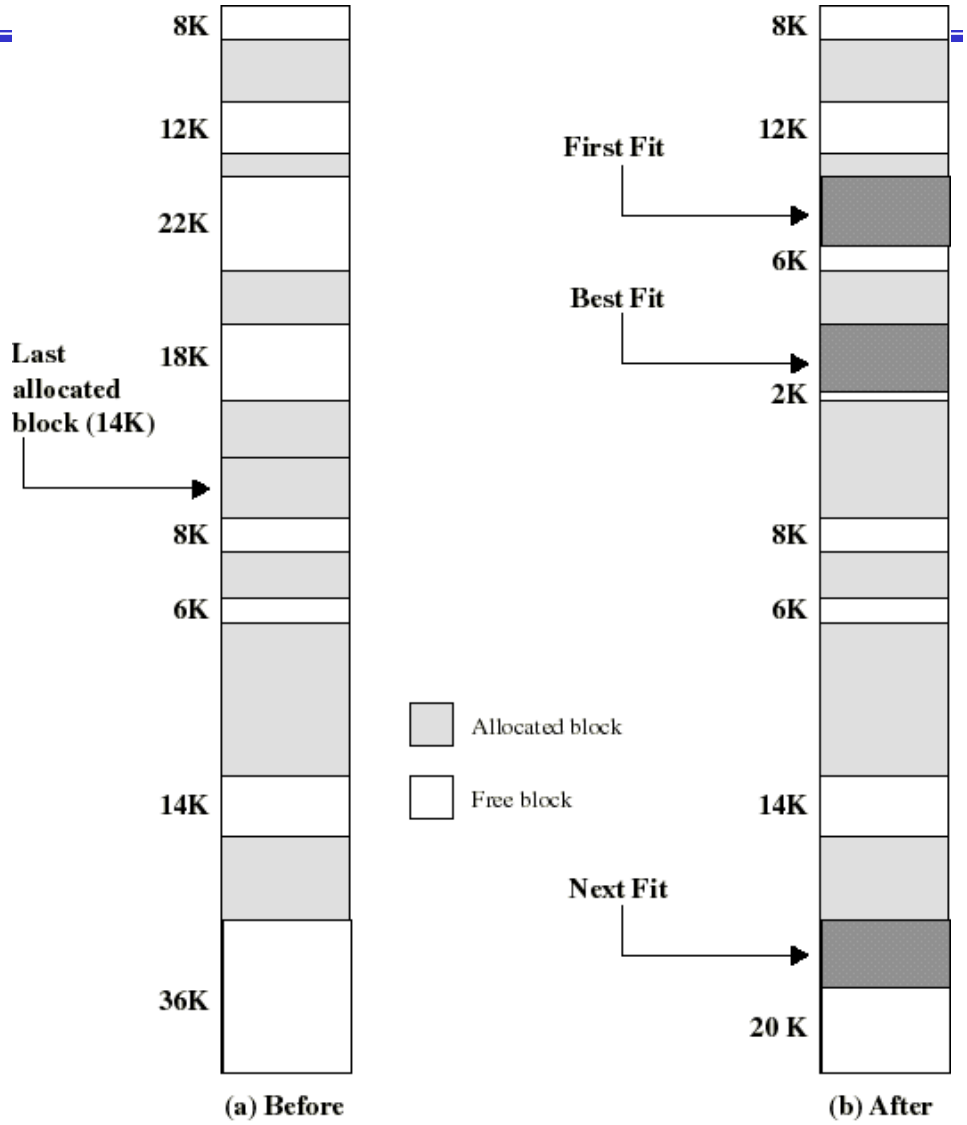
Dynamic partitioning

- Số lượng partition không cố định và partition có thể có kích thước khác nhau
- Mỗi process được cấp phát chính xác dung lượng bộ nhớ cần thiết
- Gây ra hiện tượng phân mảnh ngoại



Chiến lược placement

- Dùng để quyết định cấp phát khối bộ nhớ trống nào cho một process
- Mục tiêu: giảm chi phí compaction
- Các chiến lược placement
  - *Best-fit:* chọn khối nhớ trống nhỏ nhất
  - *First-fit:* chọn khối nhớ trống phù hợp đầu tiên kể từ đầu bộ nhớ
  - *Next-fit:* chọn khối nhớ trống phù hợp đầu tiên kể từ vị trí cấp phát cuối cùng
  - *Worst-fit:* chọn khối nhớ trống lớn nhất



Example Memory Configuration Before and After Allocation of 16 Kbyte Block

Cấp phát không liên tục

1.Cơ chế *phân trang* (paging)

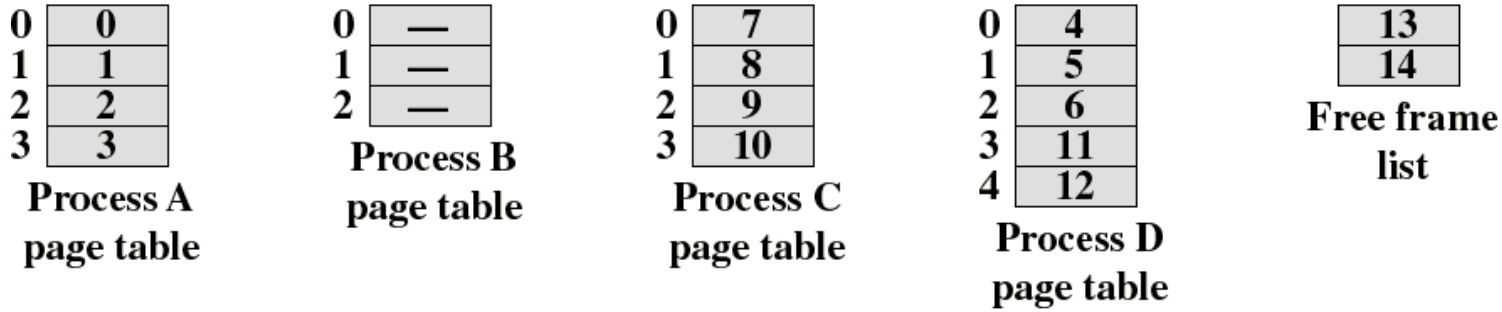
Bộ nhớ vật lý →khung trang (*frame)*.

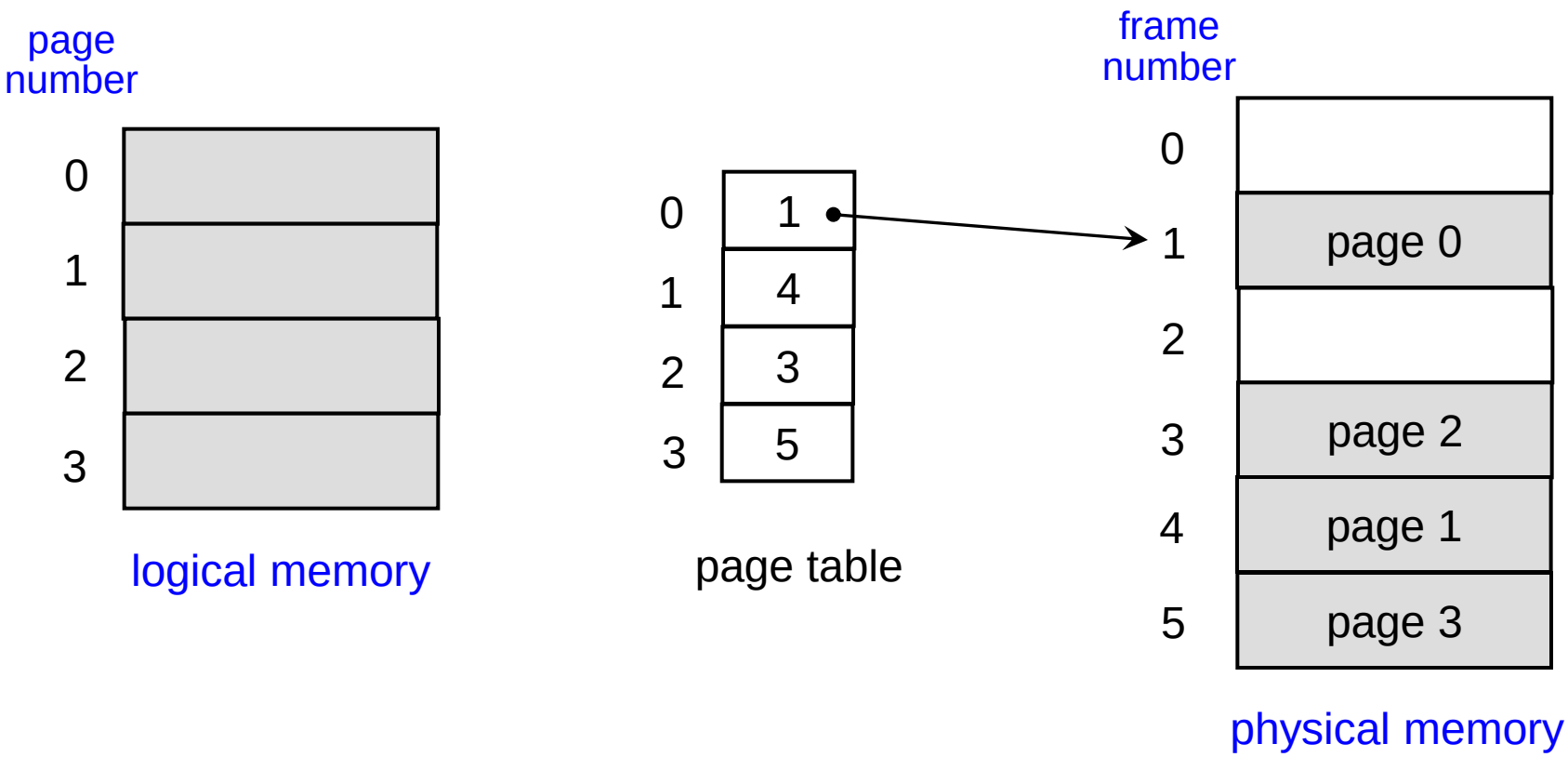
- Kích thước của frame là lũy thừa của 2, từ khoảng 512 byte đến 16MB.

- *Bộ nhớ luận lý* (logical memory) hay *không gian địa chỉ luận lý* là tập mọi địa chỉ luận lý mà một chương trình bất kỳ có thể sinh ra → page.
  - Ví dụ

    MOV REG,1000 //1000 là một địa chỉ luận lý
- *Bảng phân trang* (page table) để ánh xạ địa chỉ luận lý thành địa chỉ thực

1.Cơ chế phân trang (tt)

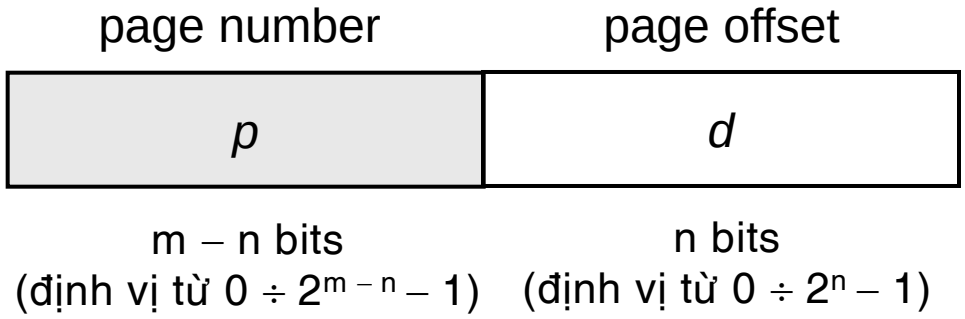




1.Cơ chế phân trang (tt)

A) Chuyển đổi địa chỉ trong paging

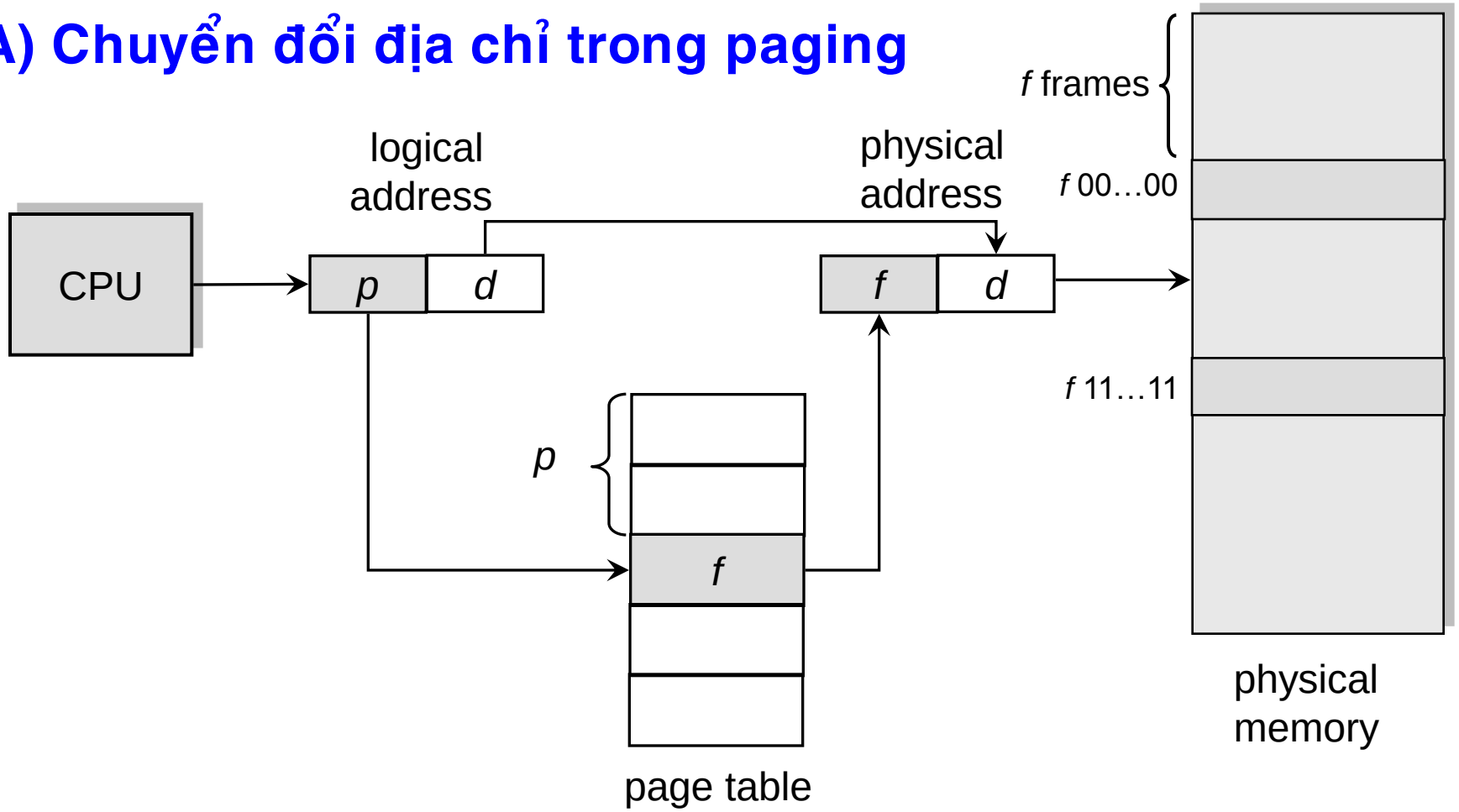
- Địa chỉ luận lý gồm có:
  - *Số hiệu trang (Page number)* *p*
  - *Địa chỉ tương đối trong trang (Page offset)* *d*
- Nếu kích thước của không gian địa chỉ luận lý là 2^m, và kích thước của trang là 2^n (đơn vị là byte hay word tùy theo kiến trúc máy) thì



Bảng phân trang sẽ có tổng cộng $2^m/2^n = 2^{m-n}$ *mục* (entry)

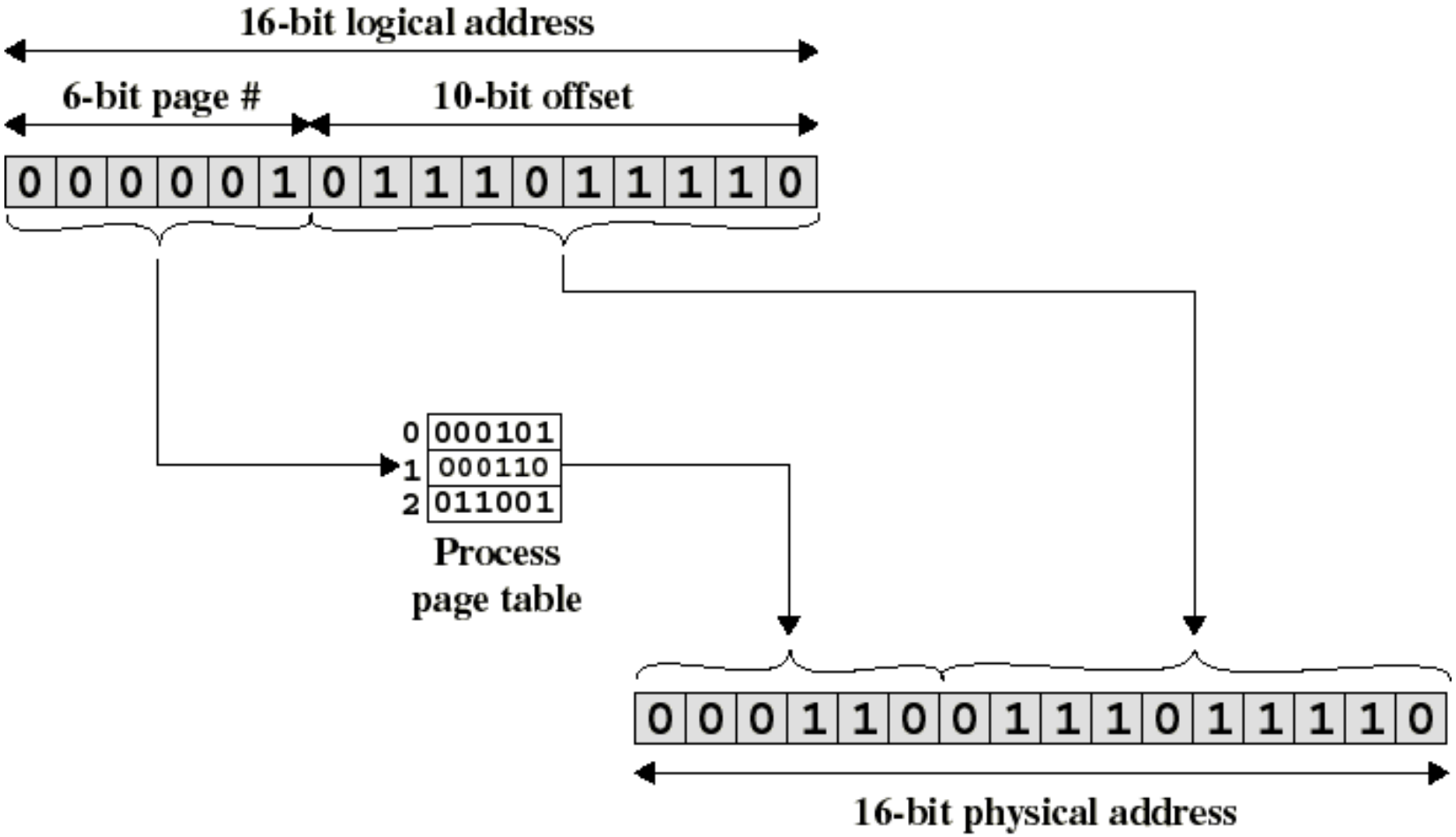
1.Cơ chế phân trang (tt)

A) Chuyển đổi địa chỉ trong paging



1.Cơ chế phân trang (tt)

Ví dụ: Chuyển đổi địa chỉ nhớ trong paging



Ví dụ

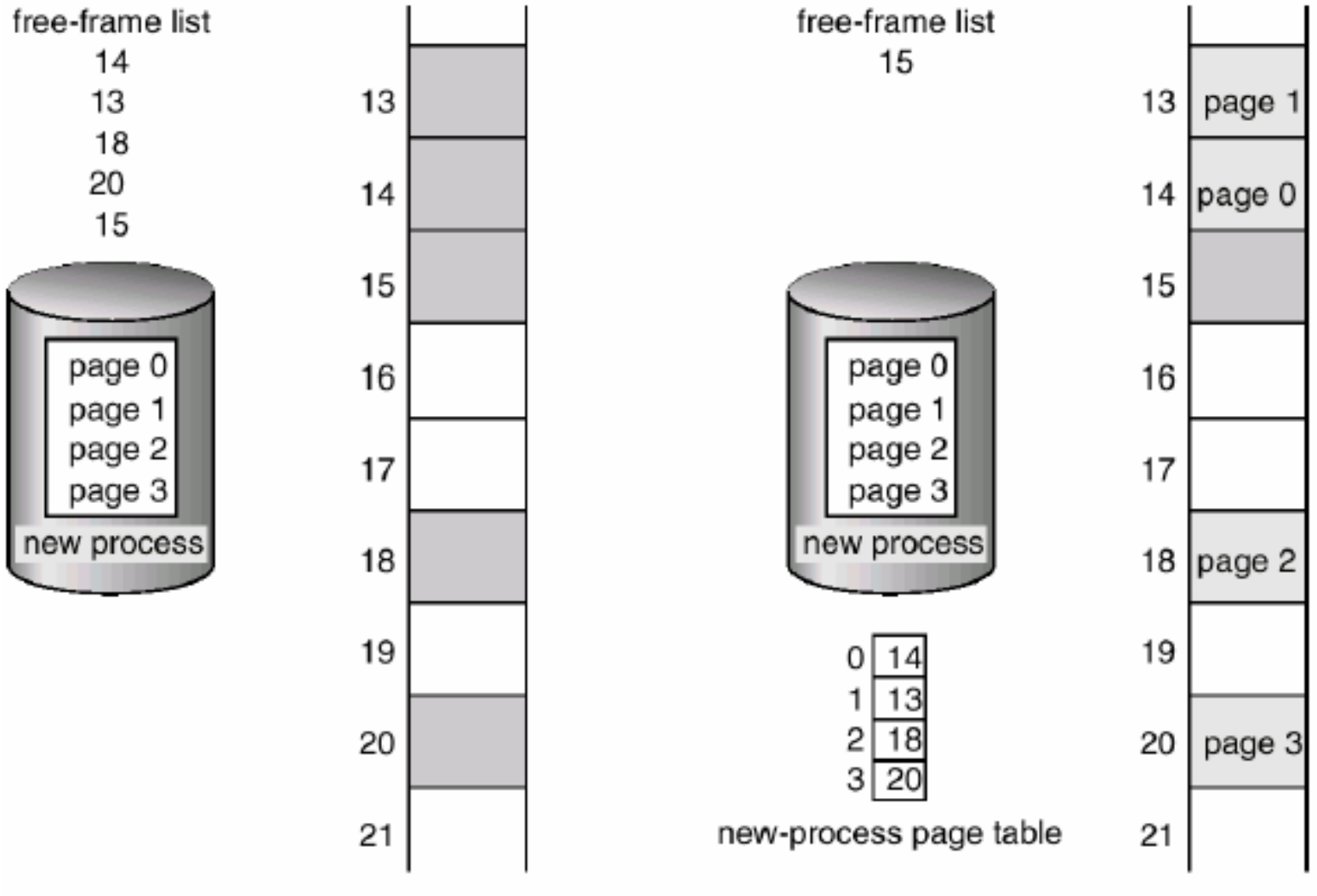
Xét một không gian địa chỉ có 8 trang, mỗi trang có kích thước 1K. ánh xạ vào bộ nhớ vật lý có 32 khung trang

a) Địa chỉ logic gồm bao nhiêu bit ?

b) Địa chỉ physic gồm bao nhiêu bit ?

c) ng trang bao nhiêu c? i c trong ng trang n bao nhiêu bit?

1.Cơ chế phân trang (tt)



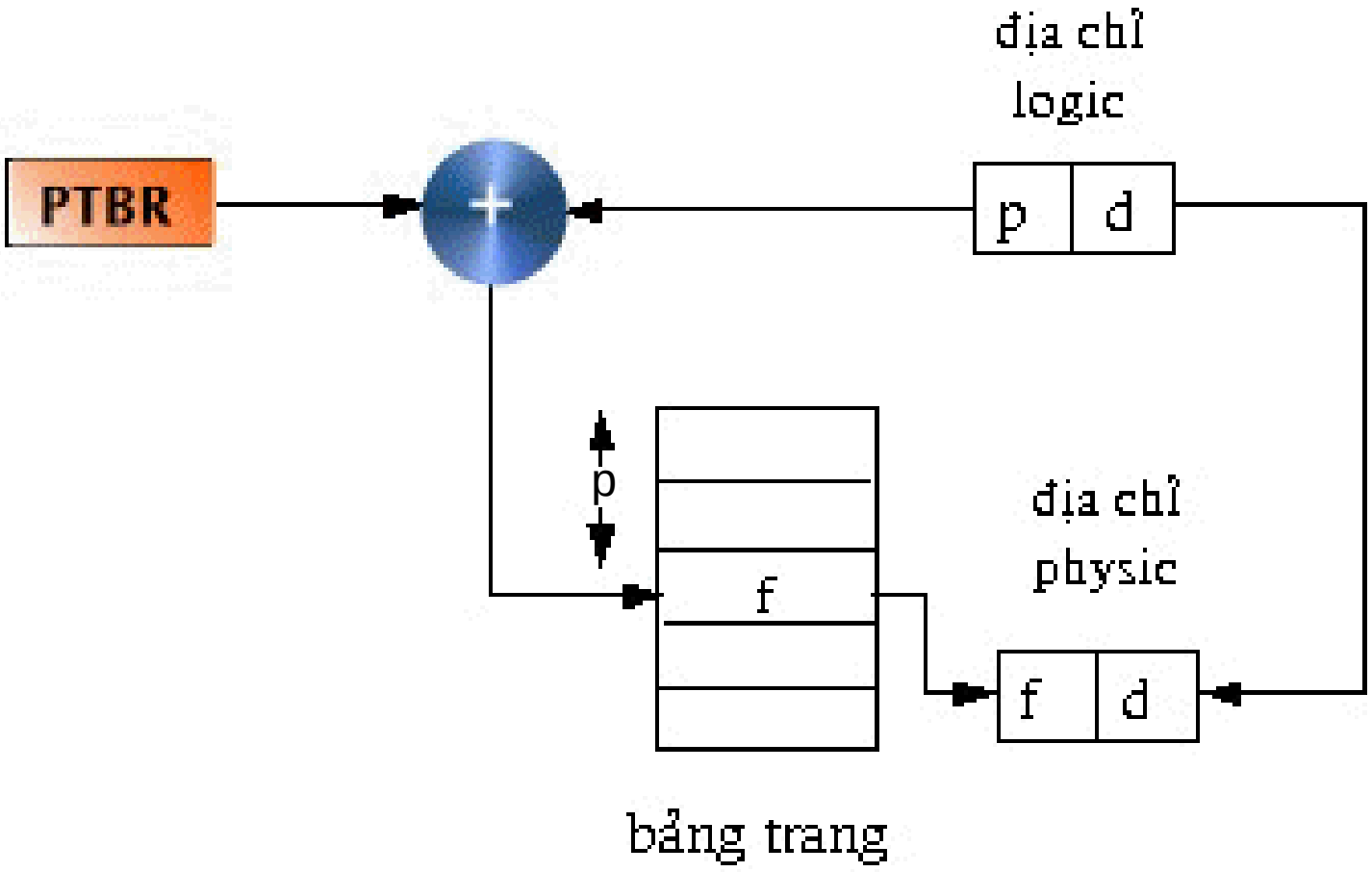
Trước khi và sau khi cấp phát cho Process mới

B) Cài đặt bảng trang (Paging hardware)

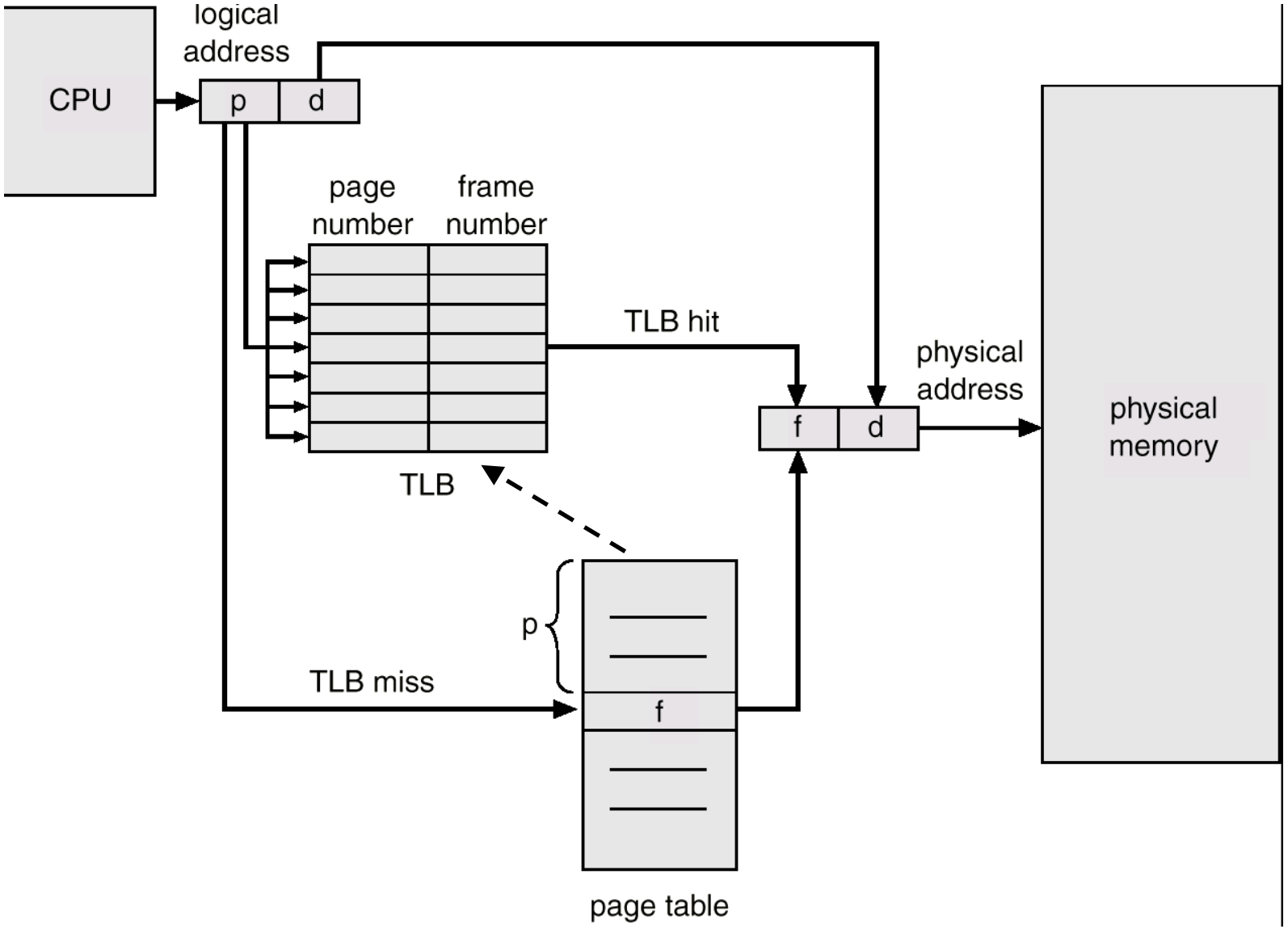
- Bảng phân trang thường được lưu giữ trong bộ nhớ chính
  - Mỗi process được hệ điều hành cấp một bảng phân trang
  - Thanh ghi *page-table base* (PTBR) trỏ đến bảng phân trang
  - Thanh ghi *page-table length* (PTLR) biểu thị kích thước của bảng phân trang (có thể được dùng trong cơ chế bảo vệ bộ nhớ)
- Thường dùng một bộ phận cache phần cứng có tốc độ truy xuất và tìm kiếm cao, gọi là *thanh ghi kết hợp* (associative register) hoặc *translation look-aside buffers* (TLBs)

B) Cài đặt bảng trang (Paging hardware)

- Dùng thanh ghi Page-Table Base Register (PTBR)



Paging hardware với TLB



C) Effective access time (EAT)

Tính thời gian truy xuất hiệu dụng (effective access time, EAT)

- Thời gian tìm kiếm trong TLB (associative lookup): ε
- Thời gian một chu kỳ truy xuất bộ nhớ: x
- *Hit ratio:* tỉ số giữa số lần chỉ số trang được tìm thấy (hit) trong TLB và số lần truy xuất khởi nguồn từ CPU
  - Kí hiệu hit ratio: α
- Thời gian cần thiết để có được chỉ số frame
  - Khi chỉ số trang có trong TLB (hit) $\varepsilon + x$
  - Khi chỉ số trang không có trong TLB (miss) $\varepsilon + x + x$
- *Thời gian truy xuất hiệu dụng*

$$\begin{aligned} EAT &= (\varepsilon + x)\alpha + (\varepsilon + 2x)(1 - \alpha) \\ &= (2 - \alpha)x + \varepsilon \end{aligned}$$

C) Effective access time (EAT)

- Ví dụ 1: đơn vị thời gian nano giây
  - Associative lookup = 20
  - Memory access = 100
  - Hit ratio = 0.8
  - EAT = $(100 + 20) \times 0.8 + (200 + 20) \times 0.2$
    $= 1.2 \times 100 + 20$
    $= 140$

- Ví dụ 2
  - Associative lookup = 20
  - Memory access = 100
  - Hit ratio = 0.98
  - EAT = $(100 + 20) \times 0.98 + (200 + 20) \times 0.02$
    $= 1.02 \times 100 + 20$
    $= 122$

Ví dụ

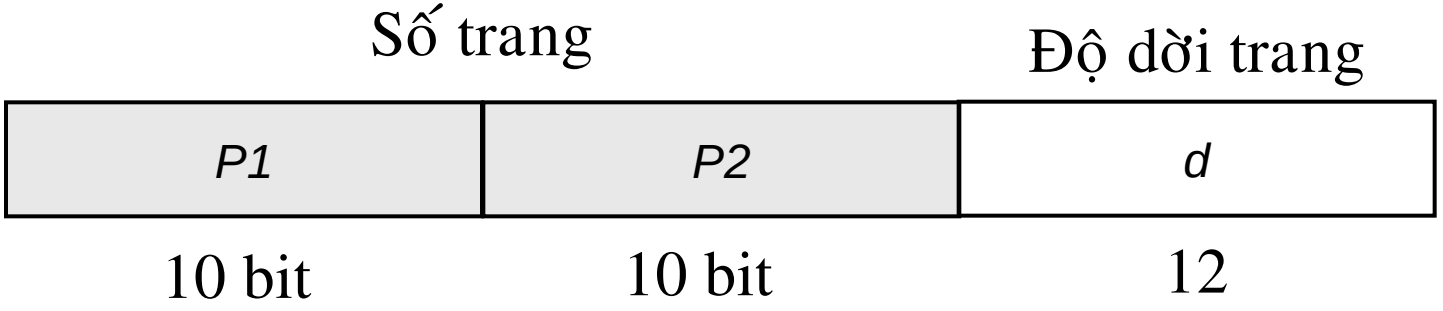
Xét một hệ thống sử dụng kỹ thuật phân trang, với bảng trang được lưu trữ trong bộ nhớ chính.

a) Nếu thời gian cho một lần truy xuất bộ nhớ bình thường là 200nanoseconds, thì mất bao nhiêu thời gian cho một thao tác truy xuất bộ nhớ trong hệ thống này ?

b) Nếu sử dụng TLBs với hit-ratio (tỉ lệ tìm thấy) là 75%, thời gian để tìm trong TLBs xem như bằng 0, tính thời gian truy xuất bộ nhớ trong hệ thống (effective memory reference time)

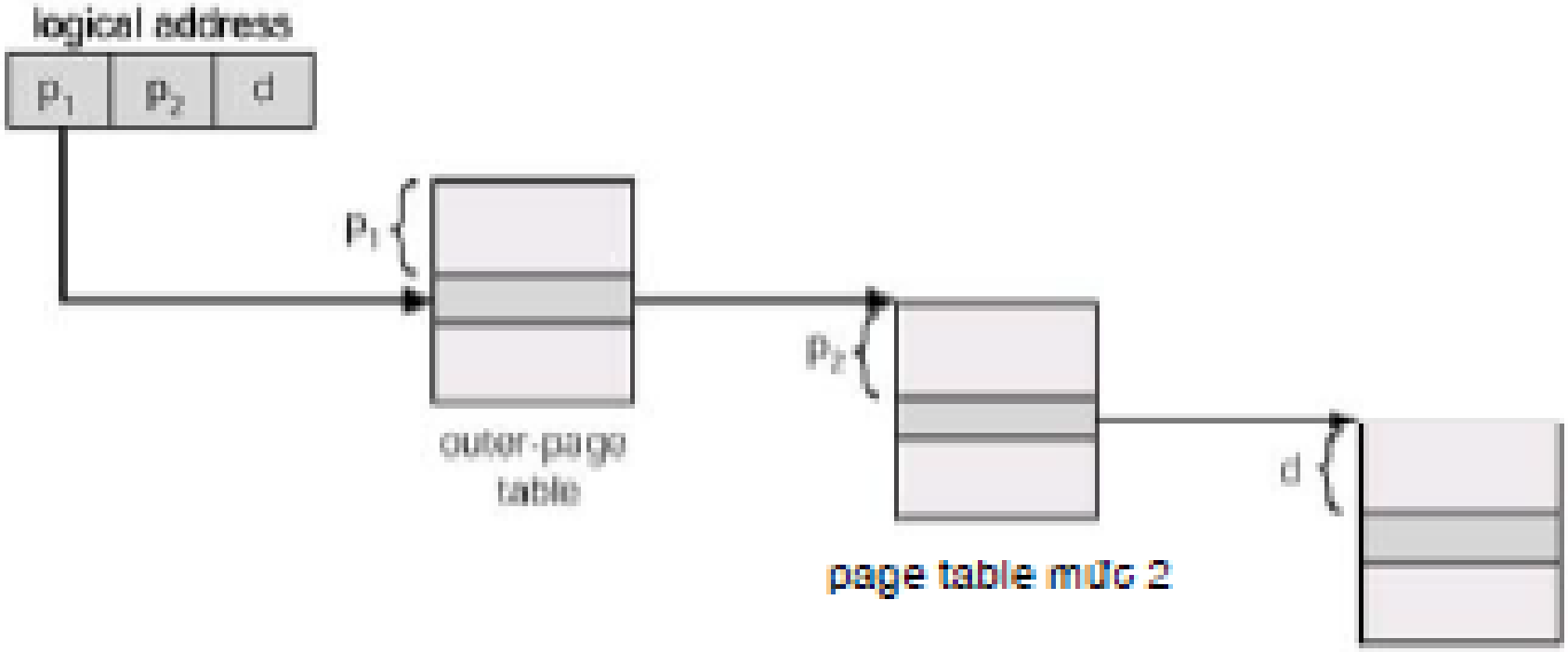
D) Tổ chức bảng trang - *Phân trang đa cấp*

- Các hệ thống hiện đại đều hỗ trợ không gian địa chỉ ảo rất lớn (2^{32} đến 2^{64}), ở đây giả sử là 2^{32}
  - Giả sử kích thước trang nhớ là 4KB (= 2^{12})
    $\Rightarrow$ bảng phân trang sẽ có $2^{32}/2^{12} = 2^{20}$ = 1M mục.
  - Giả sử mỗi mục gồm 4 byte thì mỗi process cần 4MB cho bảng phân trang
    - VD: Phân trang hai cấp



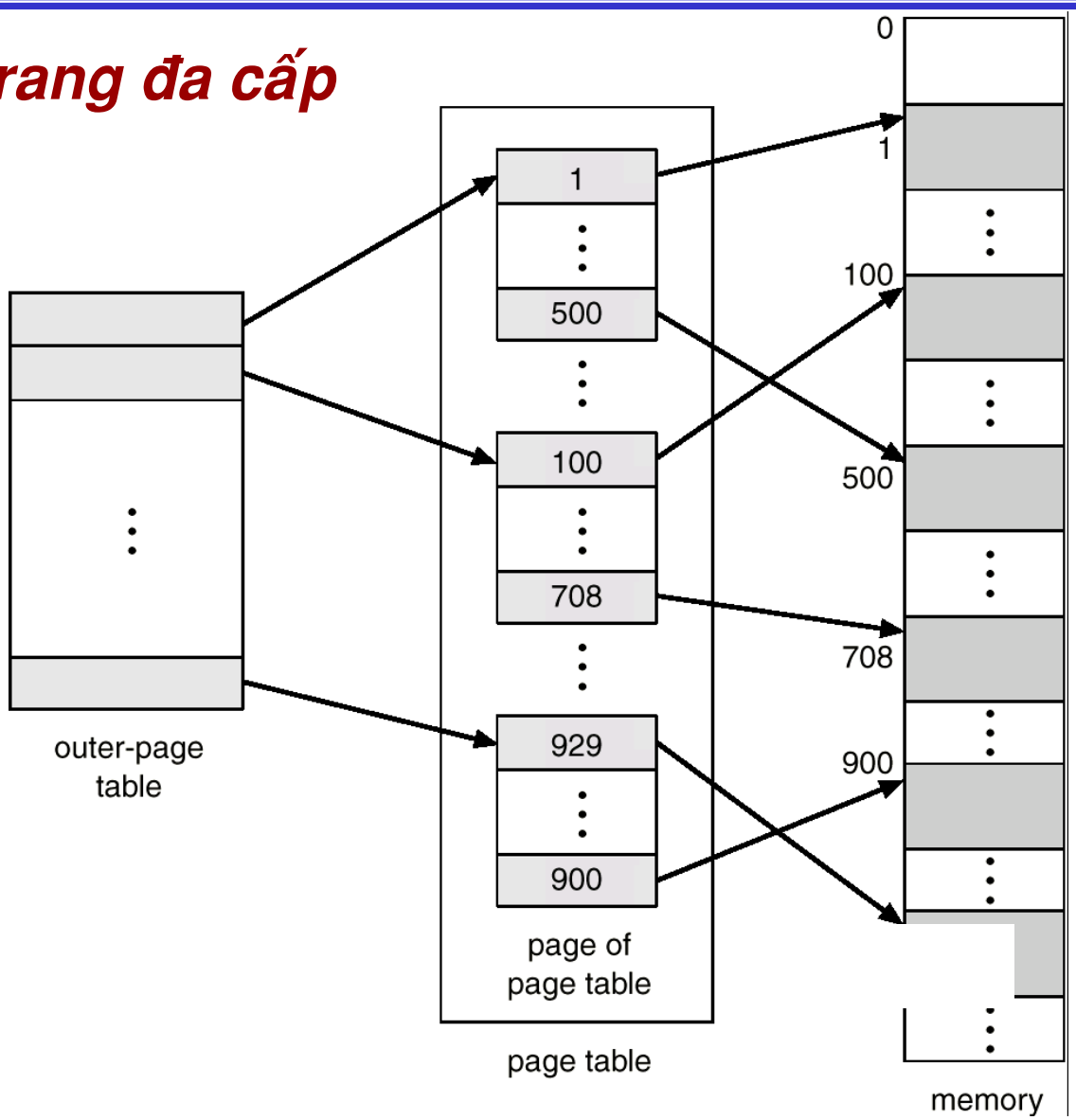
Phân trang đa cấp (tt)

- Sơ đồ chuyển đổi địa chỉ (address-translation scheme) cho cơ chế bảng phân trang 2 mức, 32-bit địa chỉ



D) Tổ chức bảng trang

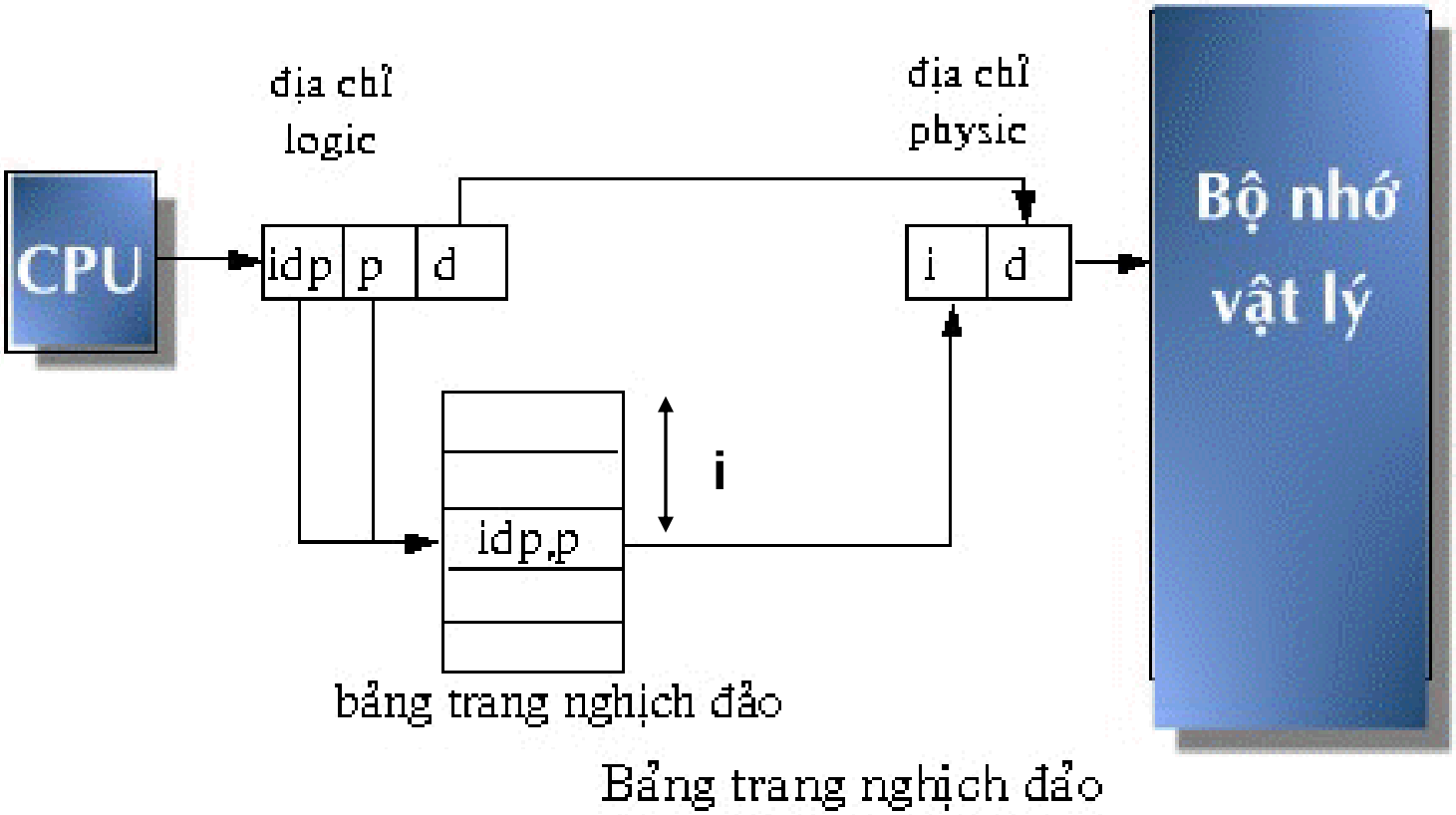
Phân trang đa cấp



D) Tổ chức bảng trang

- Bảng trang nghịch đảo: sử dụng cho tất cả các Process

<IDP,p,d>



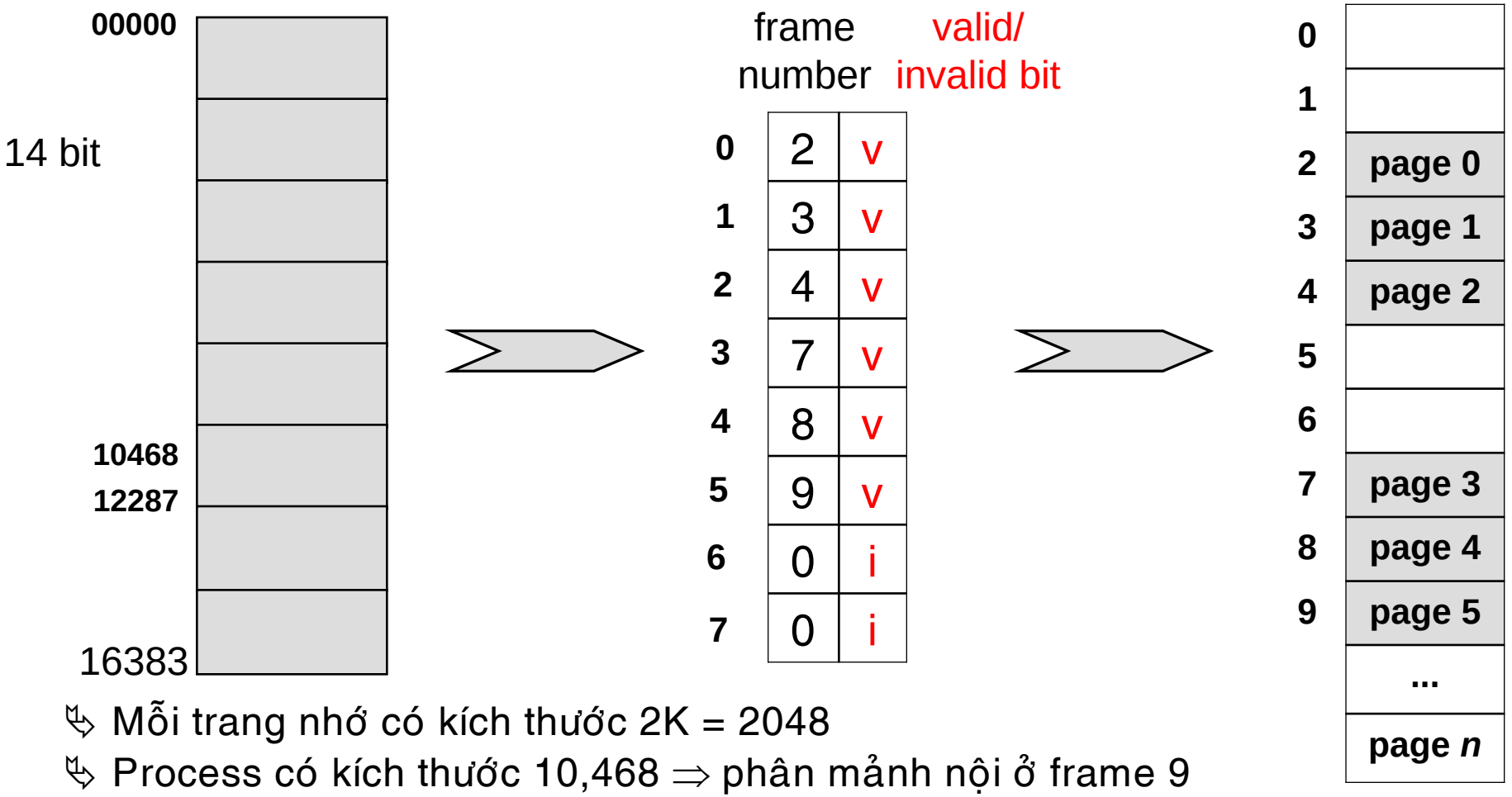
Bảng trang nghịch đảo

E) Bảo vệ bộ nhớ

- Việc bảo vệ bộ nhớ được hiện thực bằng cách gắn với frame các *bit bảo vệ* (protection bits) được giữ trong bảng phân trang. Các bit này biểu thị các thuộc tính sau
  - read-only, read-write, execute-only

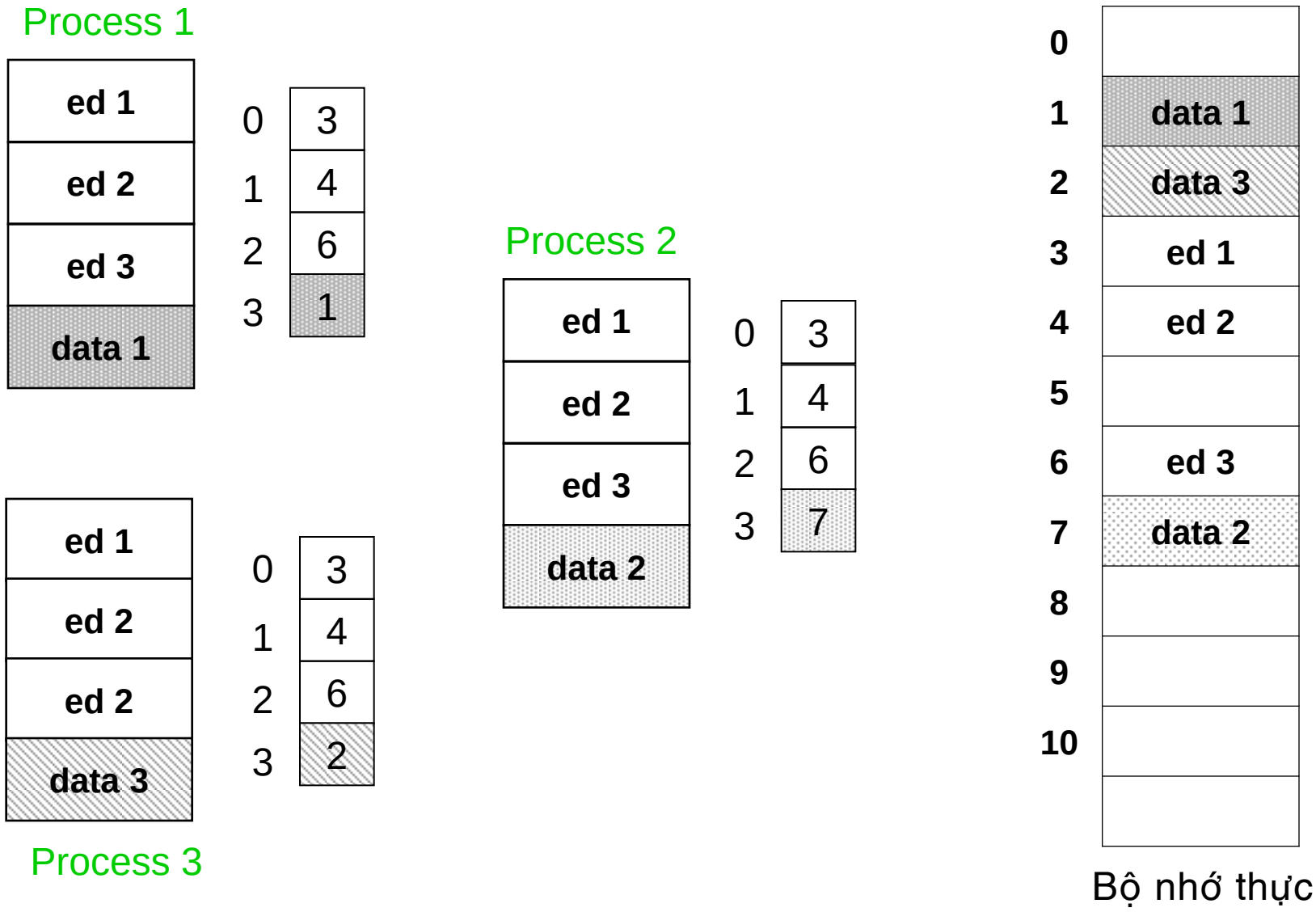
- Ngoài ra, còn có một *valid/invalid bit* gắn với mỗi mục trong bảng phân trang
  - "valid": cho biết là trang của process, do đó là một trang hợp lệ.
  - "invalid": cho biết là trang không của process, do đó là một trang bất hợp lệ.

Bảo vệ bằng valid/invalid bit



- Mỗi trang nhớ có kích thước 2K = 2048
- Process có kích thước 10,468 ⇒ phân mảnh nội ở frame 9 (chứa page 5), các địa chỉ ảo > 12287 là các địa chỉ invalid.
- Dùng PTLR để kiểm tra truy xuất đến bảng phân trang có nằm trong bảng hay không.

F) Chia sẻ các trang nhớ



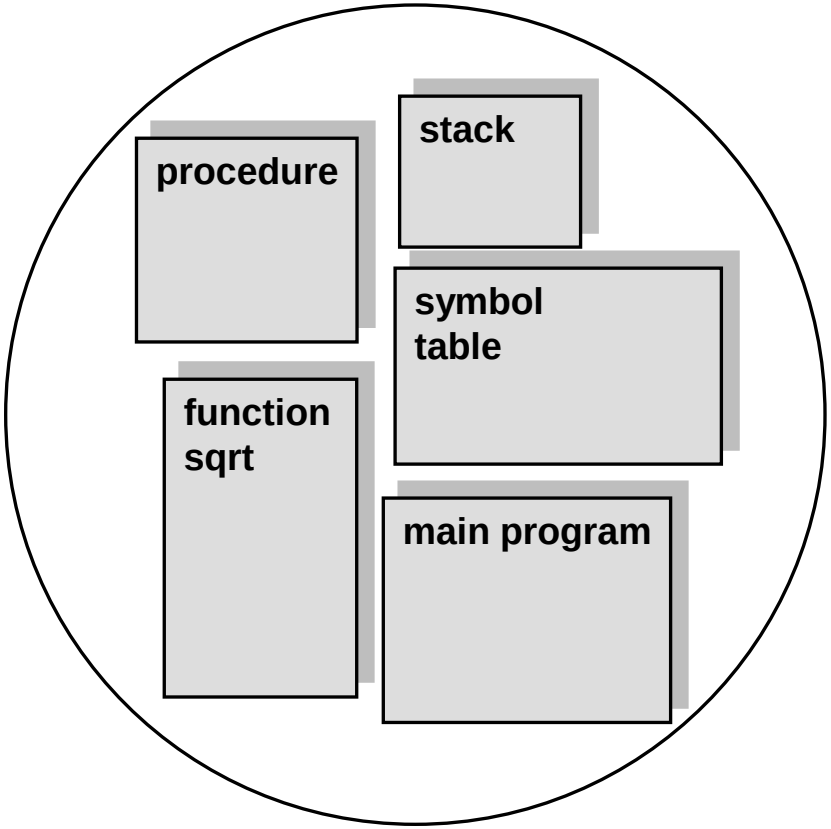
2.Phân đoạn (segmentation)

- Nhìn lại cơ chế phân trang
  - user view (không gian địa chỉ ảo) tách biệt với không gian bộ nhớ thực. Cơ chế phân trang thực hiện phép ánh xạ user-view vào bộ nhớ thực.

- Trong thực tế, dưới góc nhìn của user, một chương trình cấu thành từ nhiều *đoạn* (segment). Mỗi đoạn là một đơn vị luận lý của chương trình, như
  - main program, procedure, function
  - local variables, global variables, common block, stack, symbol table, arrays,…

User view của một chương trình

- Thông thường, một chương trình được biên dịch. Trình biên dịch sẽ tự động xây dựng các segment.
- Ví dụ, trình biên dịch Pascal sẽ tạo ra các segment sau:
  - Global variables
  - Procedure call stack
  - Procedure/function code
  - Local variable
- Trình loader sẽ gán mỗi segment một số định danh riêng.

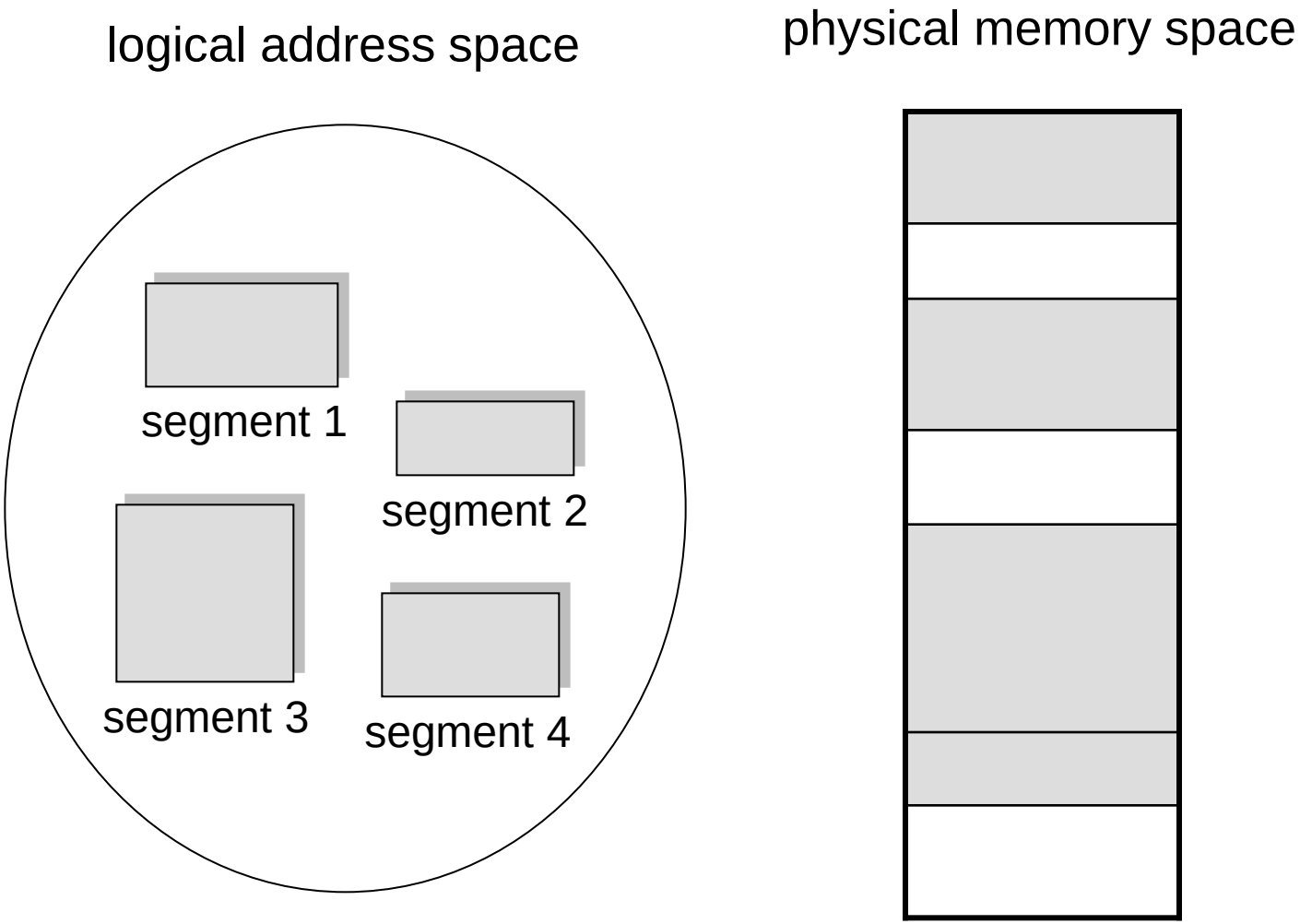


Logical address space

Phân đoạn

- Dùng cơ chế *phân đoạn* để quản lý bộ nhớ có hỗ trợ user view
  - *Không gian địa chỉ ảo* là một tập các đoạn, mỗi đoạn có tên và kích thước riêng.
  - Một địa chỉ luận lý được định vị bằng tên đoạn và độ dời (offset) bên trong đoạn đó (so sánh với phân trang!)

Phân đoạn (tt)


logical address space
segment 1
segment 2
segment 3
segment 4
physical memory space

Cài đặt phân đoạn

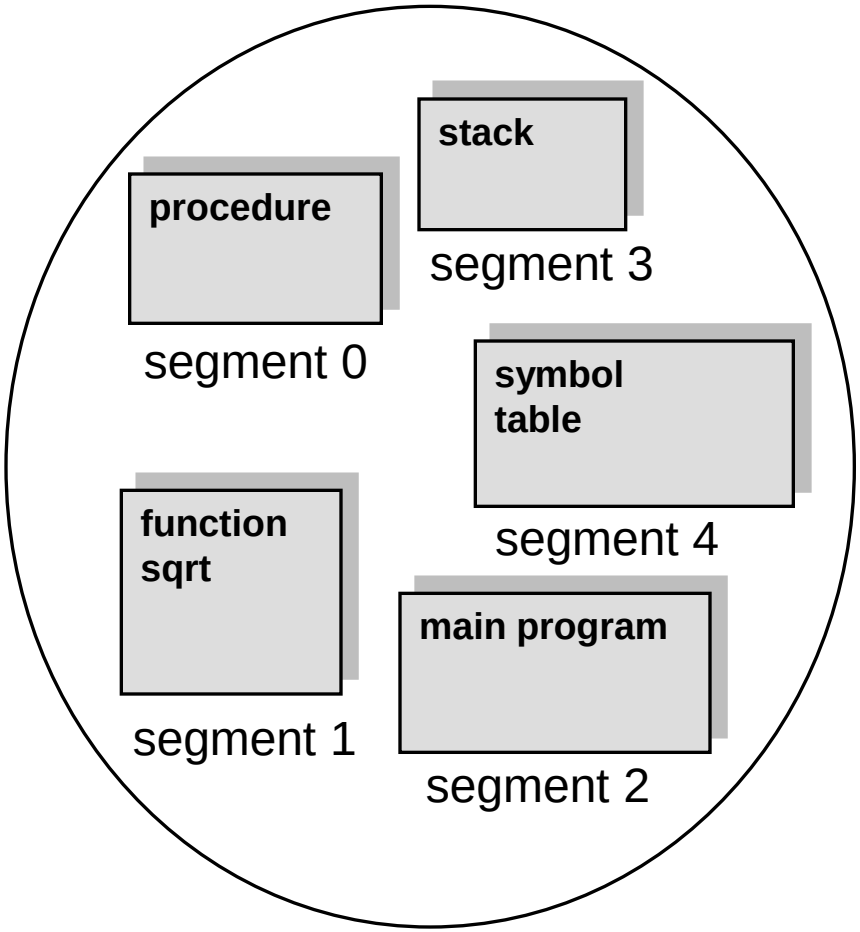
- *Địa chỉ luận lý* là một cặp giá trị

  (segment number, offset)

- *Bảng phân đoạn* (segment table): gồm nhiều mục, mỗi mục chứa
  - base, chứa địa chỉ khởi đầu của segment trong bộ nhớ
  - limit, xác định kích thước của segment
- *Segment-table base register* (STBR): trỏ đến vị trí bảng phân đoạn trong bộ nhớ
- *Segment-table length register* (STLR): số lượng segment của chương trình

  $\Rightarrow$ Một chỉ số segment s là hợp lệ nếu $s < STLR$

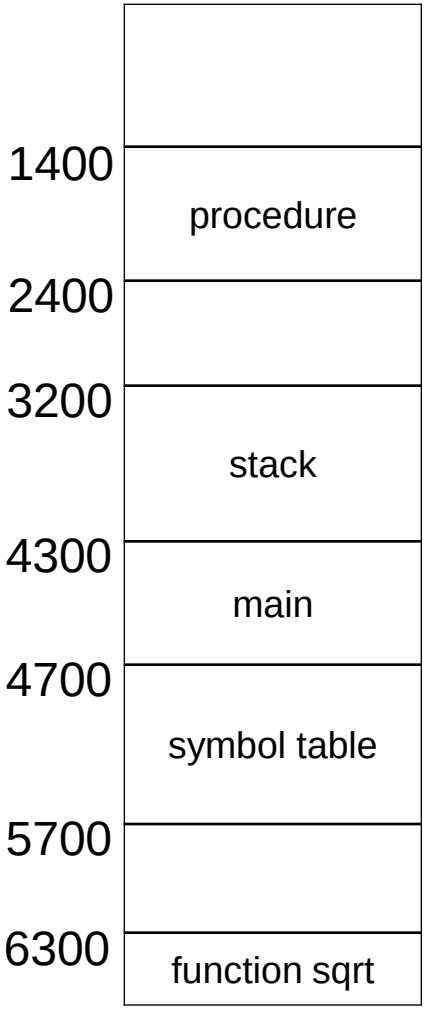
Một ví dụ về phân đoạn



logical address space

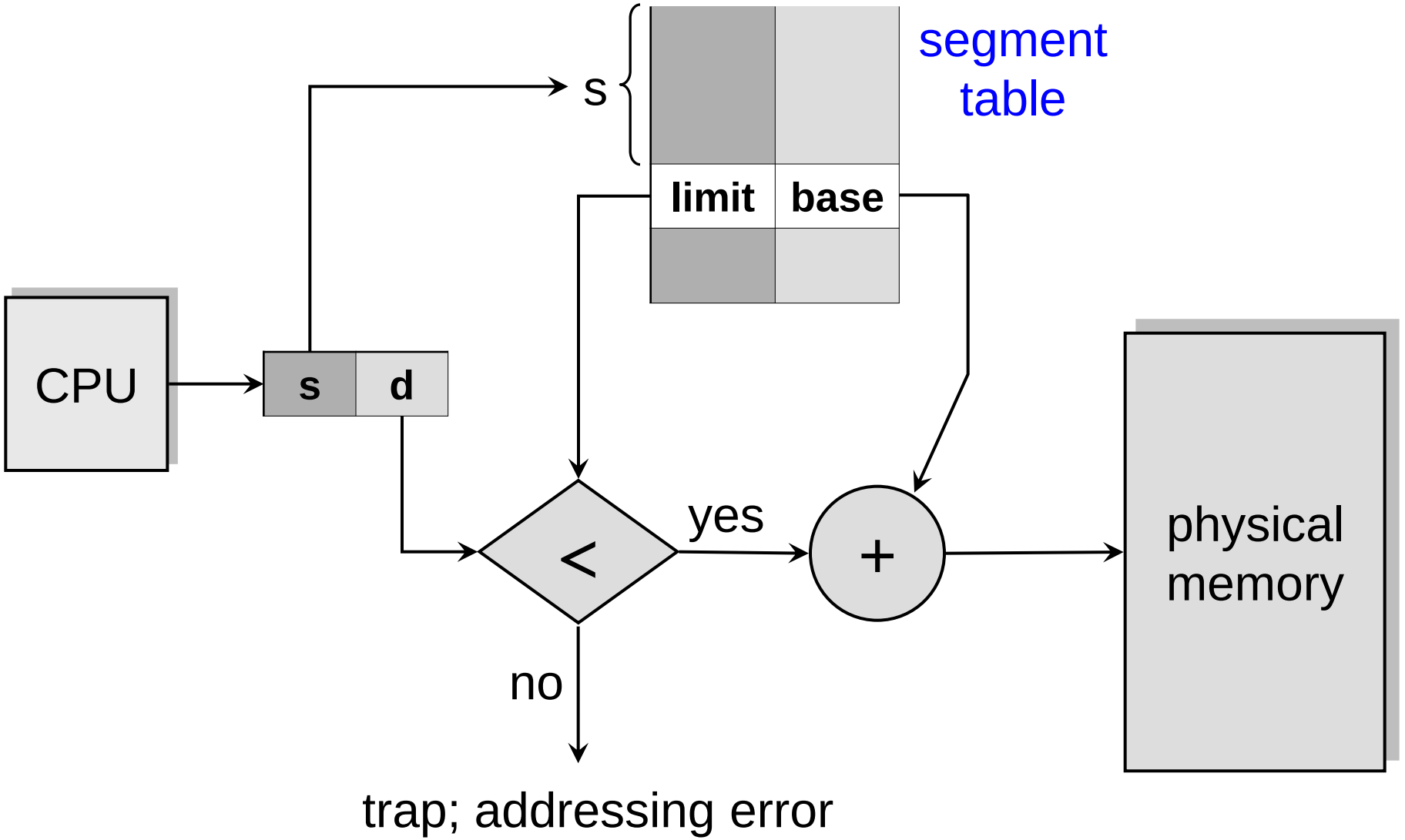
	limit	base
0	1000	1400
1	400	6300
2	400	4300
3	1100	3200
4	1000	4700

segment table



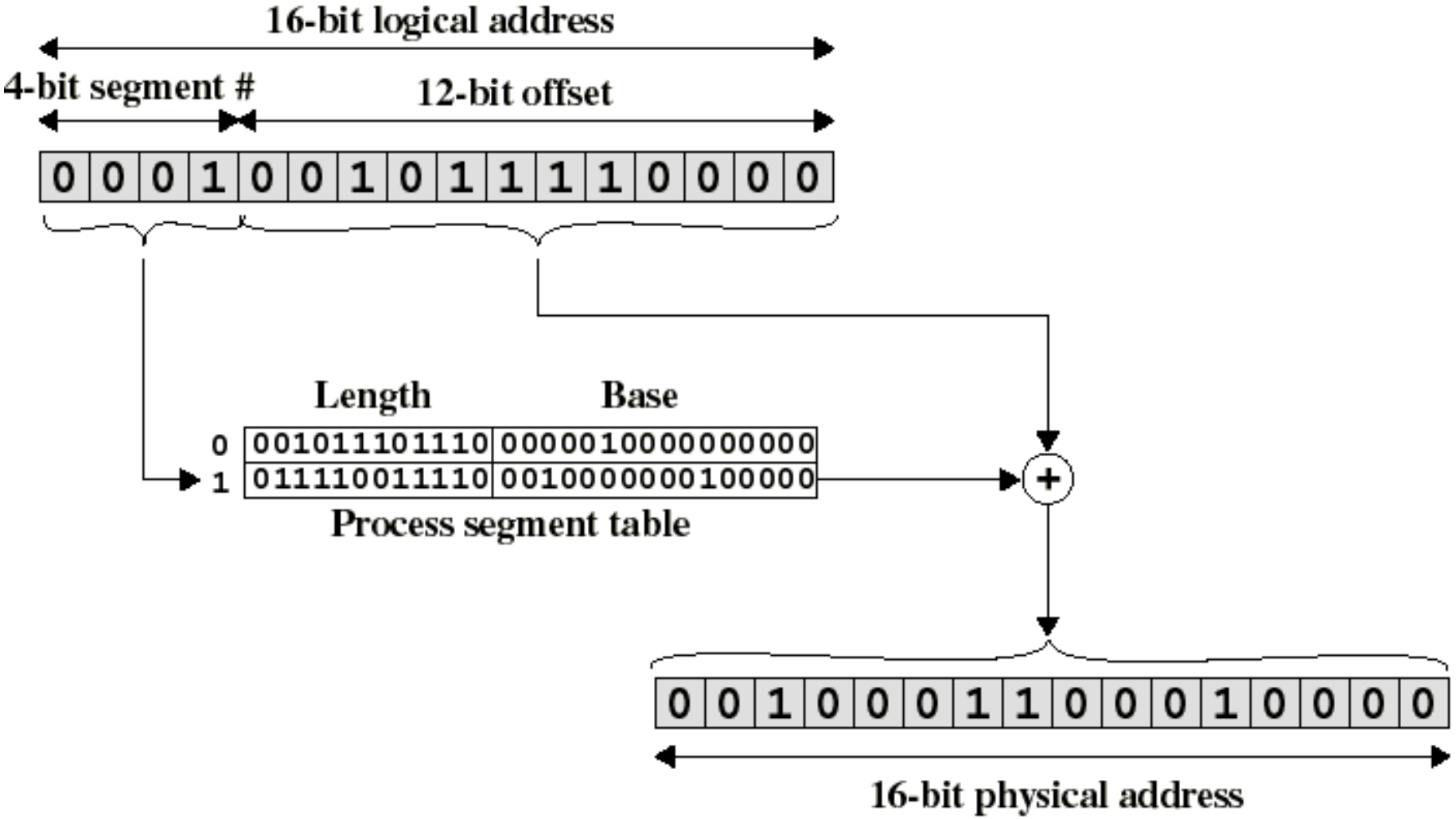
physical memory space

Phần cứng hỗ trợ phân đoạn

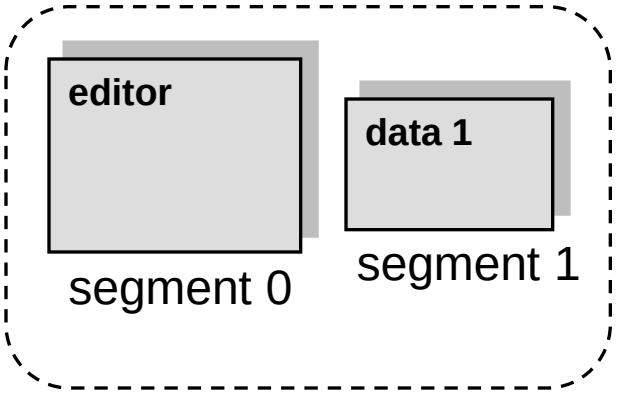


Chuyển đổi địa chỉ trong cơ chế phân đoạn

Ví dụ



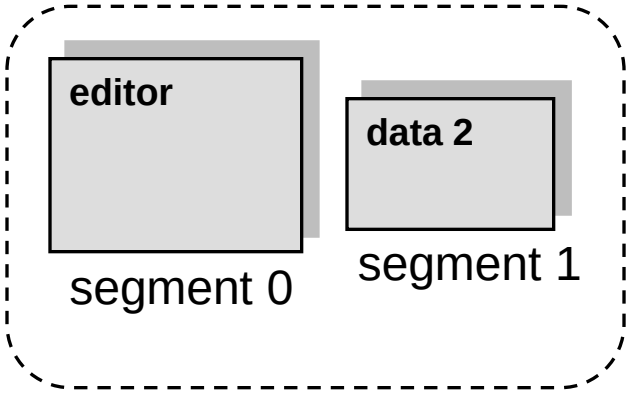
Chia sẻ các đoạn



logical address space
process P_1

	limit	base
0	25286	43062
1	4425	68348

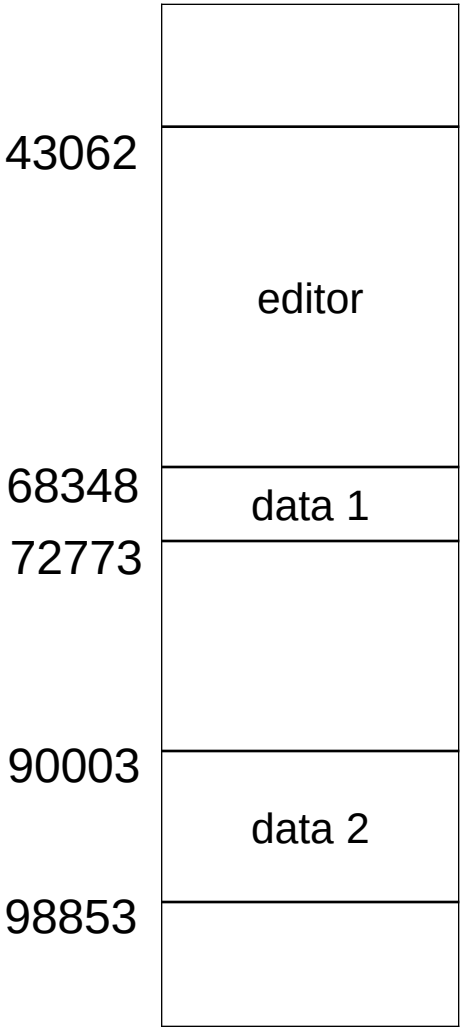
segment table
process P_1



logical address space
process P_2

	limit	base
0	25286	43062
1	8850	90003

segment table
process P_2



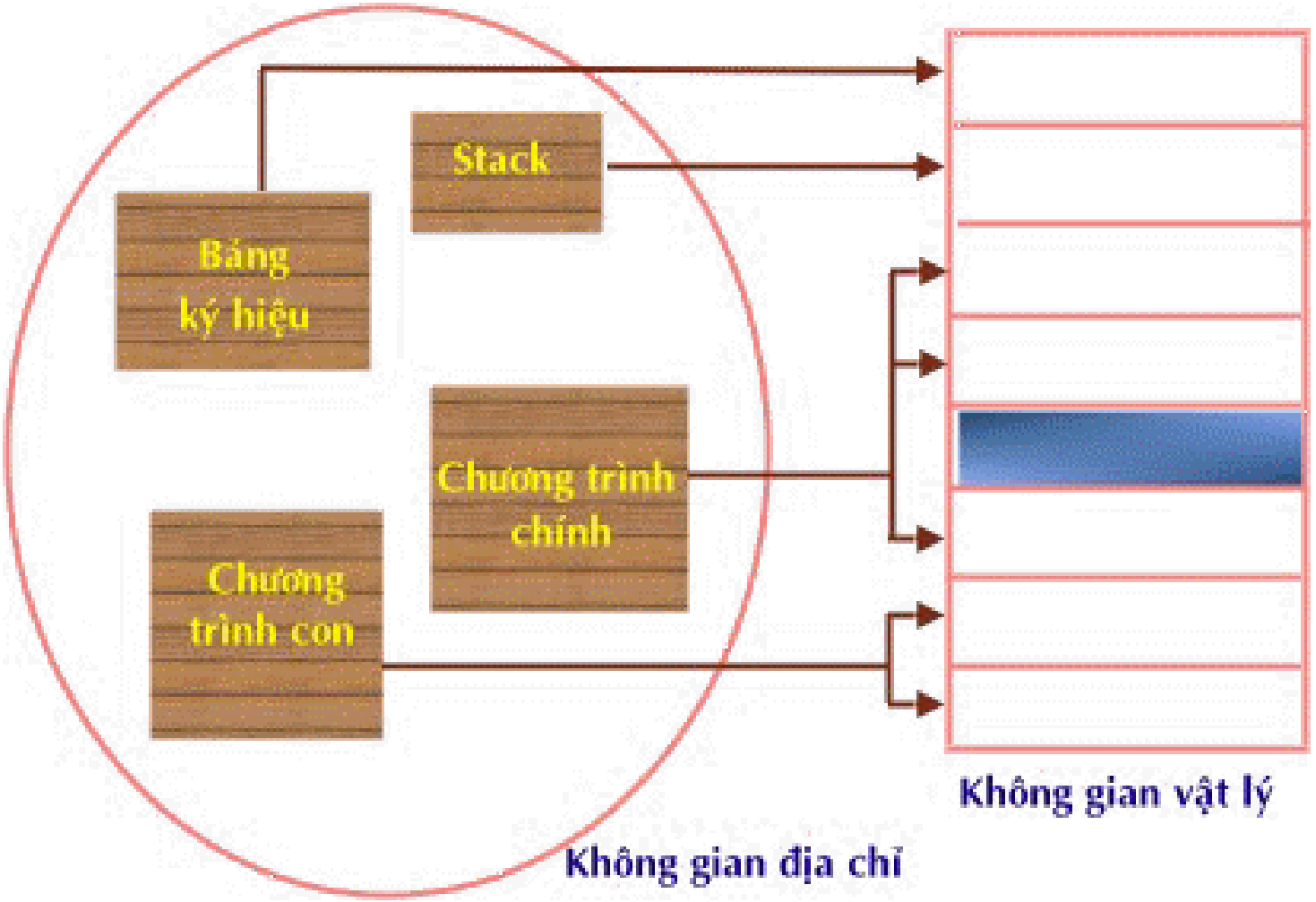
physical memory

3.Kết hợp phân trang và phân đoạn

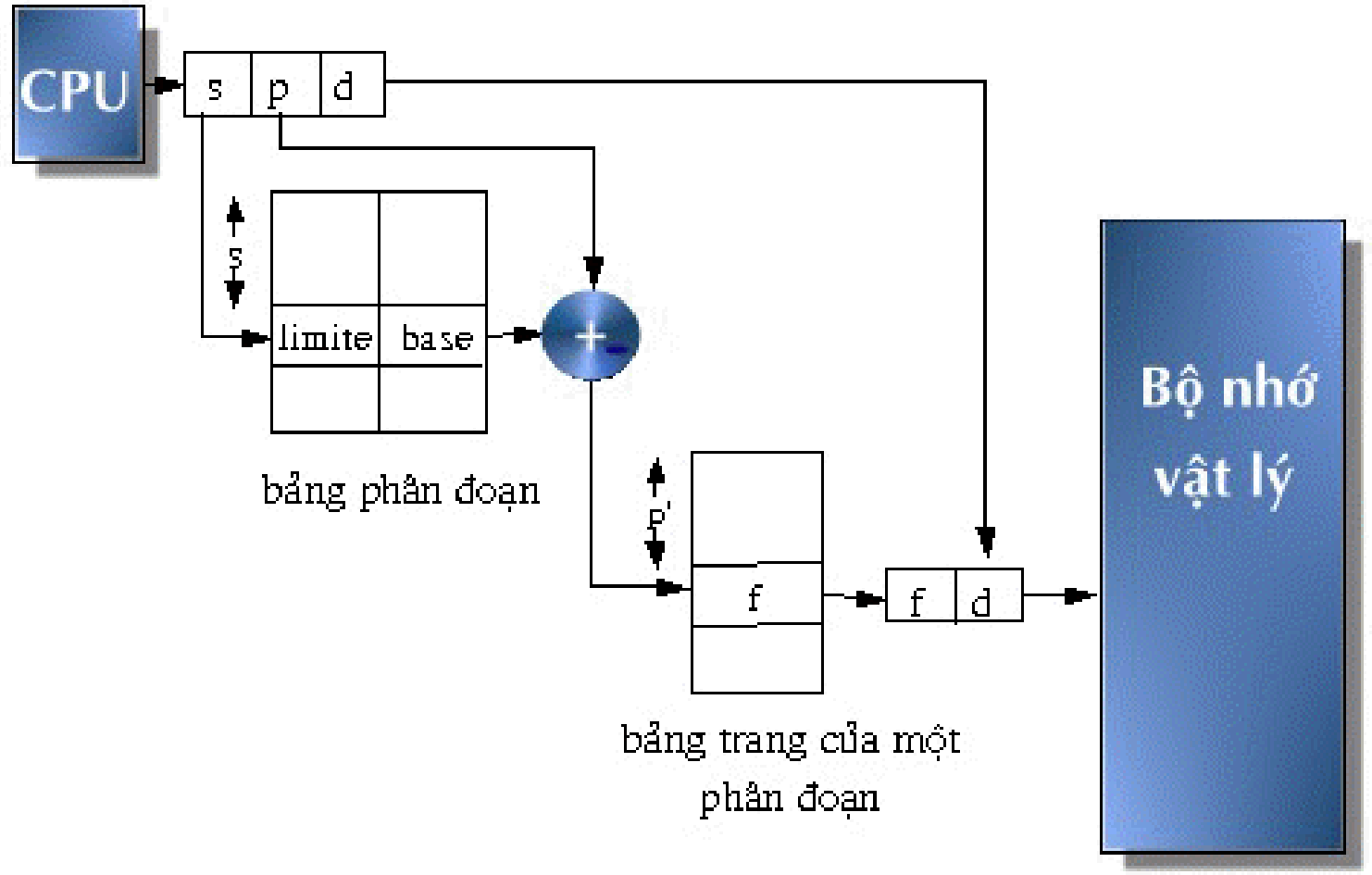
- Kết hợp phân trang và phân đoạn nhằm kết hợp các ưu điểm đồng thời hạn chế các khuyết điểm của phân trang và phân đoạn:
  - Vấn đề của phân đoạn: Nếu một đoạn quá lớn thì có thể không nạp nó được vào bộ nhớ.
  - Ý tưởng giải quyết: paging đoạn, khi đó chỉ cần giữ trong bộ nhớ các page của đoạn hiện đang cần.

Logic Addr = <s,p,d>

3.Kết hợp phân trang và phân đoạn



3.Kết hợp phân trang và phân đoạn



Cơ chế phần cứng của sự phân đoạn kết hợp phân trang

Bài Tập

- Giả sử bộ nhớ chính được phân thành các phân vùng có kích thước là 600K, 500K, 200K, 300K (theo thứ tự), cho biết các tiến trình có kích thước 212K, 417K, 112K và 426K (theo thứ tự) sẽ được cấp phát bộ nhớ như thế nào, nếu sử dụng :

a) Thuật toán First fit

b) Thuật toán Best fit

c) Thuật toán Worst fit

Thuật toán nào cho phép sử dụng bộ nhớ hiệu qủa nhất trong trường hợp trên ?

Bài Tập

- Xét một không gian có bộ nhớ luận lý kích thước 1 trang là 1kb. Tính số trang và độ dời (offset) của từng địa chỉ sau:

a) 2.375

b) 19.366

c) 30.000

d) 256

e) 16.385

Bài Tập

- Xét một không gian có bộ nhớ luận lý có 64 trang, mỗi trang có 1024 từ, mỗi từ là 2 byte được ánh xạ vào bộ nhớ vật lý có 32 trang:

a) Địa chỉ bộ nhớ vật lý có bao nhiêu bit?

b) Địa chỉ bộ nhớ luận lý có bao nhiêu bit?

c) Có bao nhiêu mục trong bảng phân trang? Mỗi mục chứa bao nhiêu bit?

Bài tập

Xét một hệ thống sử dụng kỹ thuật phân trang, với bảng trang được lưu trữ trong bộ nhớ chính.

a) Nếu thời gian cho một lần truy xuất bộ nhớ bình thường là 100 nanoseconds, thì mất bao nhiêu thời gian cho một thao tác truy xuất bộ nhớ trong hệ thống này ?

b) Nếu sử dụng TLBs với hit-ratio (tỉ lệ tìm thấy) là 85%, thời gian để tìm trong TLBs là 20 nanosecond, tính thời gian truy xuất bộ nhớ trong hệ thống (effective memory reference time)

Bài tập

Xét bảng phân đoạn sau đây :

Segment	Base	Length
0	219	600
1	2300	14
2	90	100
3	1327	580
4	1952	96

Cho biết địa chỉ vật lý tương ứng với các địa chỉ logic sau đây :

a. 0,430 b. 1,100 c. 2,500 d. 3,400 e. 4,112