Chương 3: Quản lý bộ nhớ

Hệ điều hành

ThS. Đinh Xuân Trường

truongdx@ptit.edu.vn

Posts and Telecommunications Institute of Technology

Faculty of Information Technology 1

CNTT1

Học viện Công nghệ Bưu chính Viễn thông

August 15, 2022

Mục lục

Nội dung chương 3

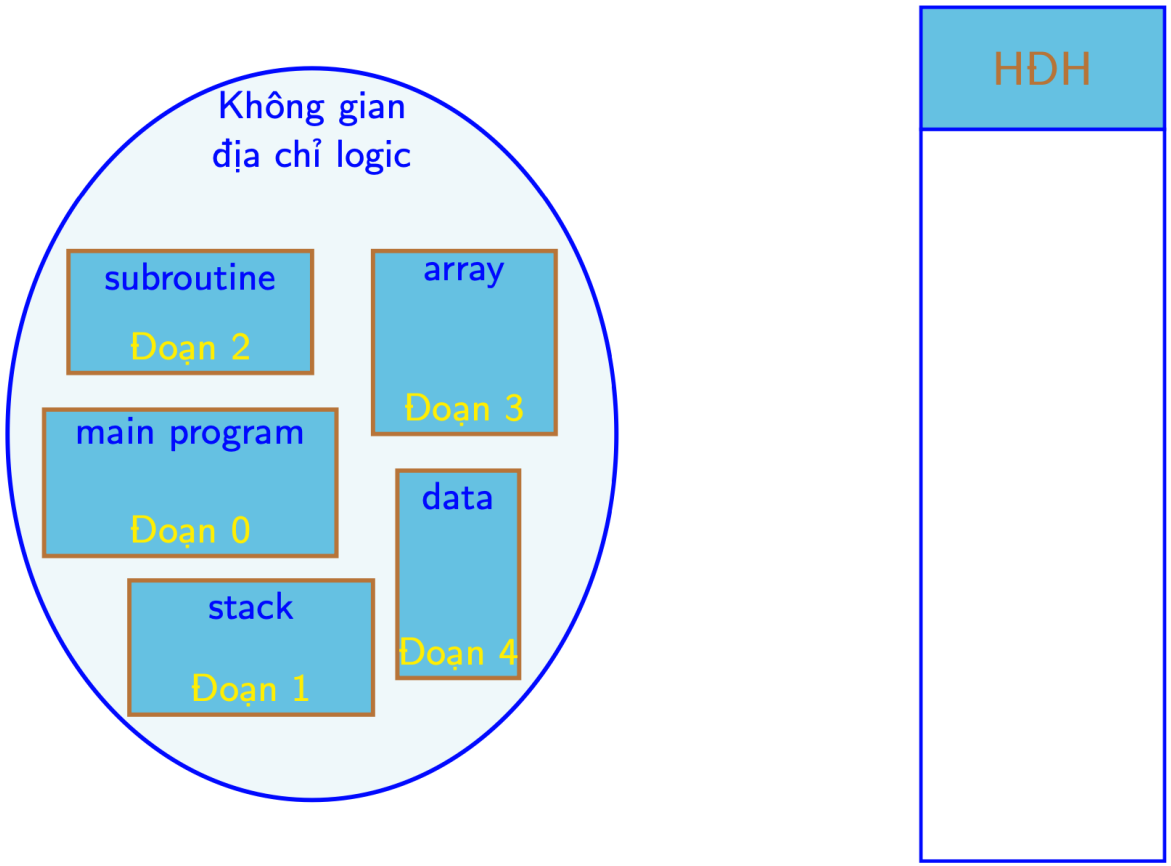
1. Địa chỉ và các vấn đề liên quan
2. Một số cách tổ chức chương trình
3. Các yêu cầu quản lý bộ nhớ
4. Phân chương bộ nhớ
5. Phân trang bộ nhớ
6. Phân đoạn bộ nhớ
7. Bộ nhớ ảo

- Chương trình thường được chia thành nhiều phần:
  - Một chương trình chính (main program)
  - Tập các chương trình con
  - Các biến, các cấu trúc dữ liệu
- Các module, đối tượng trong chương trình được xác định bằng tên:
  - Hàm sqrt(), thủ tục printf()...
  - x, y, counter, Buffer...
- Các phần tử trong module được xác định theo độ lệch với vị trí đầu:
  - Câu lệnh thứ 5 của hàm sqrt()...
  - Phần tử thứ 12 của mảng Buffer...

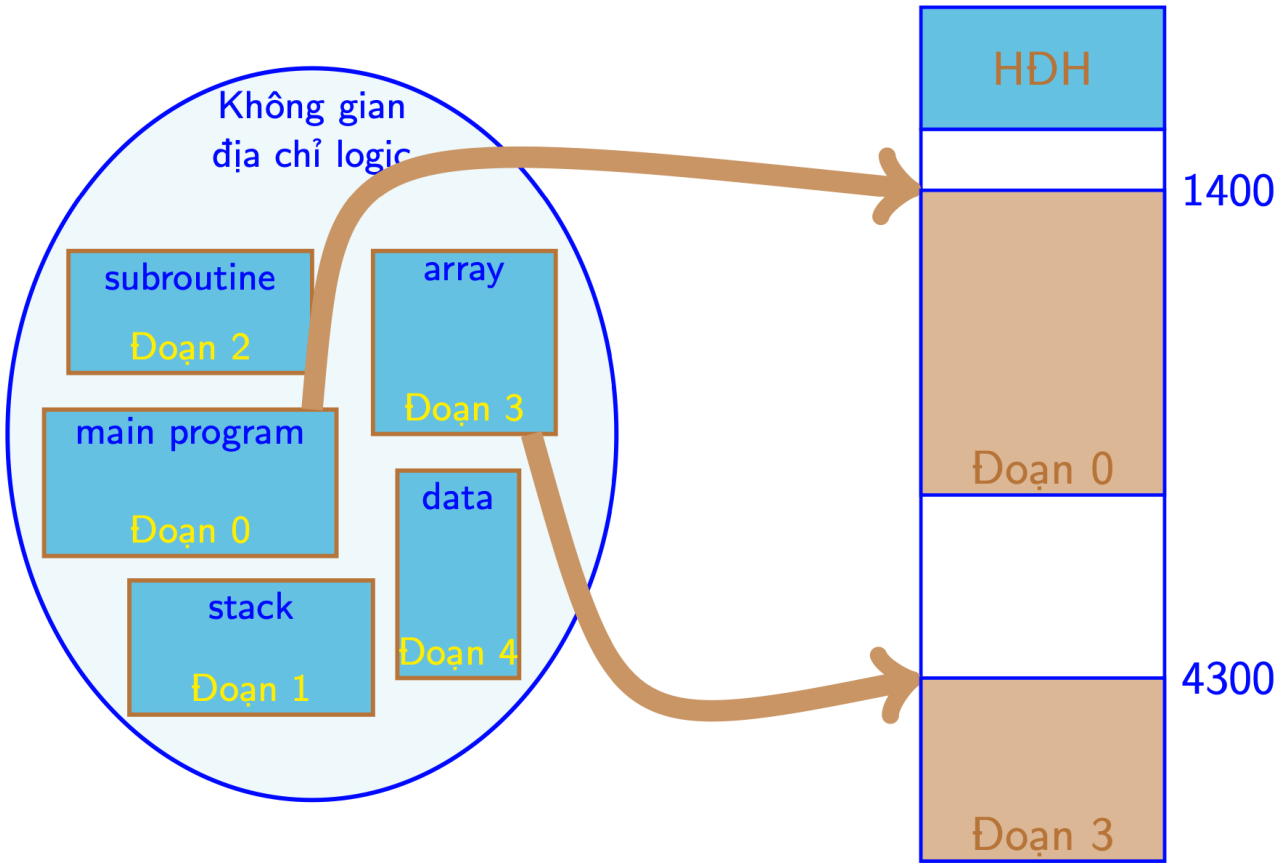
Chương trình được tổ chức như thế nào trong bộ nhớ?

- Khi phân trang, chương trình được chia thành các **trang** có kích thước đều nhau, không quan tâm tới tổ chức logic và ý nghĩa của từng thành phần.
- Một cách tổ chức khác, cho phép chia chương trình thành các **đoạn** theo cấu trúc logic
  - Đoạn chương trình - Đoạn mã: chứa toàn bộ mã chương trình, một số hàm hoặc thủ tục của chương trình.
  - Đoạn dữ liệu: chứa các biến toàn cục, các mảng.
  - Đoạn ngăn xếp: chứa ngăn xếp của tiến trình
- Mỗi đoạn chiếm một vùng liên tục
  - Có vị trí bắt đầu và kích thước
  - Có thể nằm tại bất cứ đâu trong bộ nhớ
- Đối tượng, phần tử trong từng đoạn được xác định bởi vị trí tương đối so với đầu đoạn

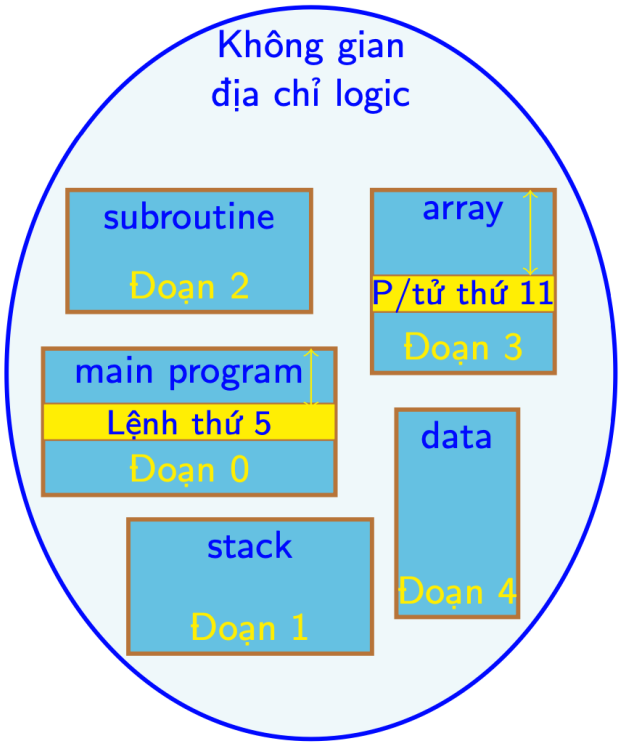
Ví dụ về phân đoạn bộ nhớ

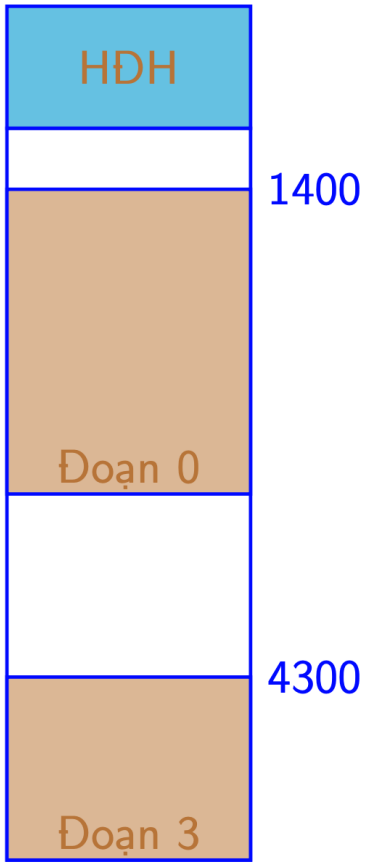


Ví dụ về phân đoạn bộ nhớ

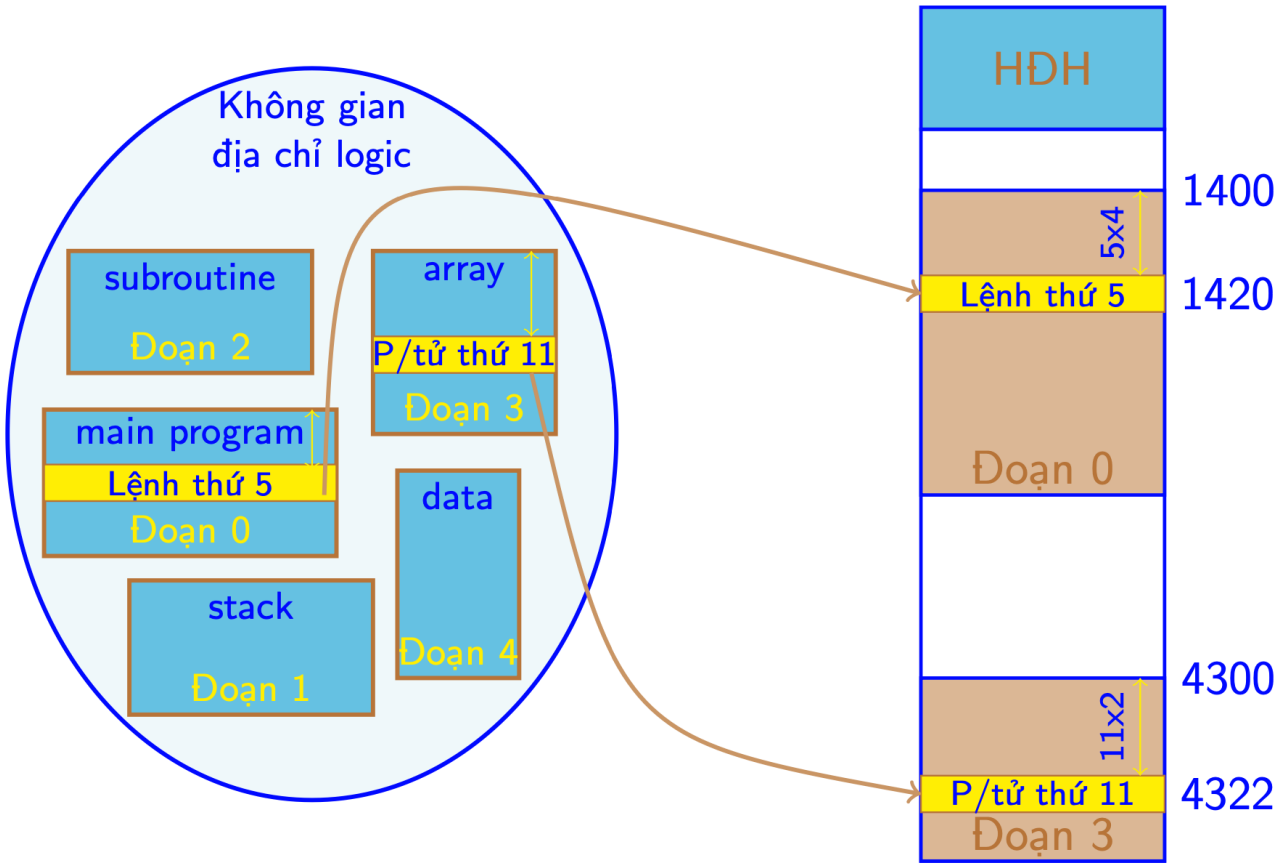


Ví dụ về phân đoạn bộ nhớ

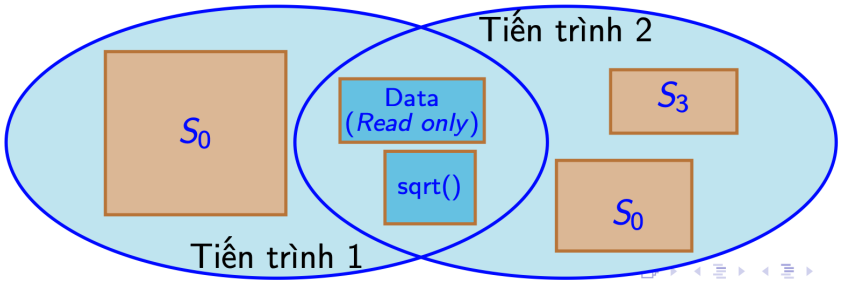




Ví dụ về phân đoạn bộ nhớ



- So sánh với phân chương động:
  - **Giống**: bộ nhớ được cấp phát theo từng vùng kích thước thay đổi
  - **Khác**: chương trình có thể chiếm nhiều hơn 1 đoạn và không cần liên tiếp nhau trong MEM
- Ưu điểm:
  - Tránh hiện tượng phân mảnh trong, dễ sắp xếp bộ nhớ
  - Dễ chia sẻ các đoạn giữa các tiến trình khác nhau



  - Kích thước mỗi đoạn có thể thay đổi không ảnh hưởng đoạn khác
- Nhược điểm:
  - Có phân mảnh ngoài

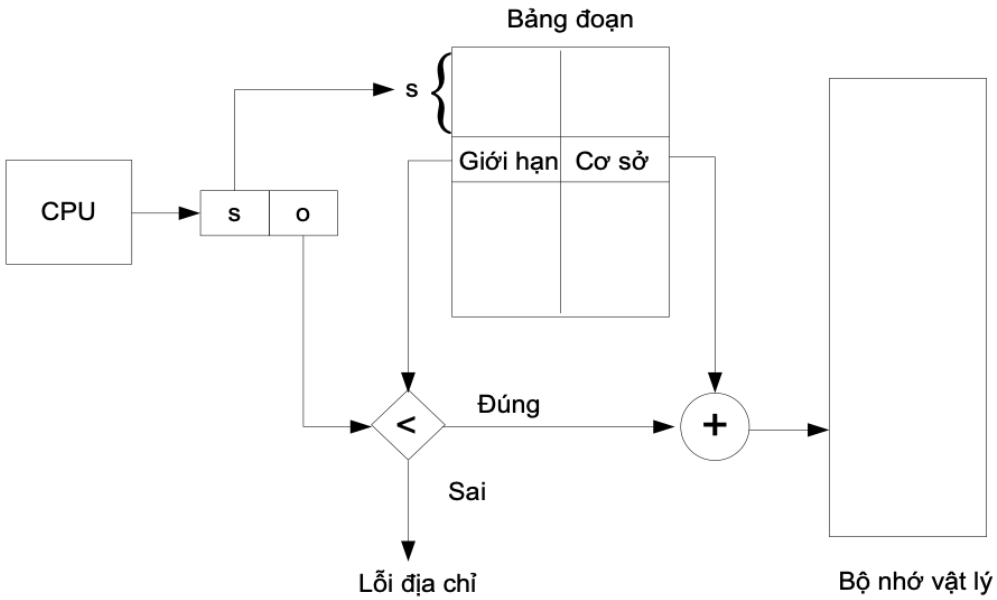
- Sử dụng **bảng đoạn** (**SCB**: *Segement Control Block*) cho mỗi tiến trình. Mỗi phần tử của bảng tương ứng với 1 đoạn, chứa:

	Mark	Address	Length
0			
⋮	...	...	...
n	...	...	...

  - Dấu hiệu (**Mark (0/1)**): Đoạn đã tồn tại trong bộ nhớ
  - Địa chỉ cơ sở (**Address**): Vị trí bắt đầu của đoạn trong bộ nhớ
  - Độ dài đoạn (**Length**): Sử dụng để chống truy cập trái phép ngoài đoạn
- Địa chỉ logic gồm 2 thành phần (s, o):
  - s: số thứ tự, tên đoạn
  - o: độ dịch trong đoạn
- **Địa chỉ truy nhập**: *<Tên (số stt) đoạn, Độ dịch trong đoạn>*

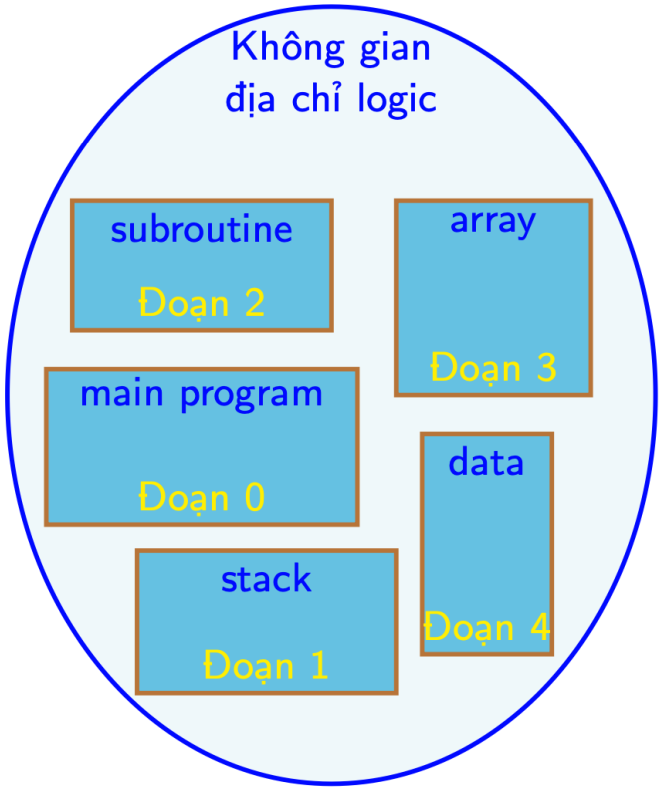
Phân đoạn bộ nhớ (cont.)

Ánh xạ địa chỉ

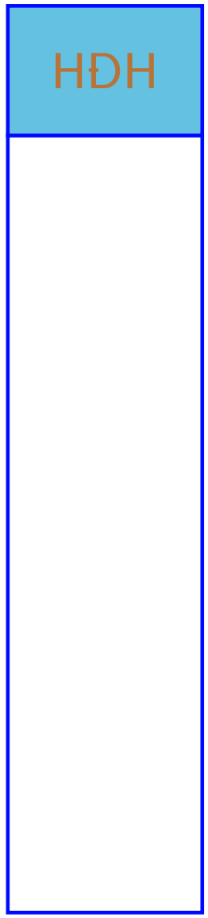


- Từ chỉ số đoạn s, vào bảng đoạn, tìm địa chỉ vật lý bắt đầu của đoạn
- So sánh độ dịch o với chiều dài đoạn, nếu lớn hơn => ịa chỉ sai - Lỗi truy cập
- Địa chỉ vật lý mong muốn là tổng của địa chỉ vật lý bắt đầu đoạn và địa chỉ lệch

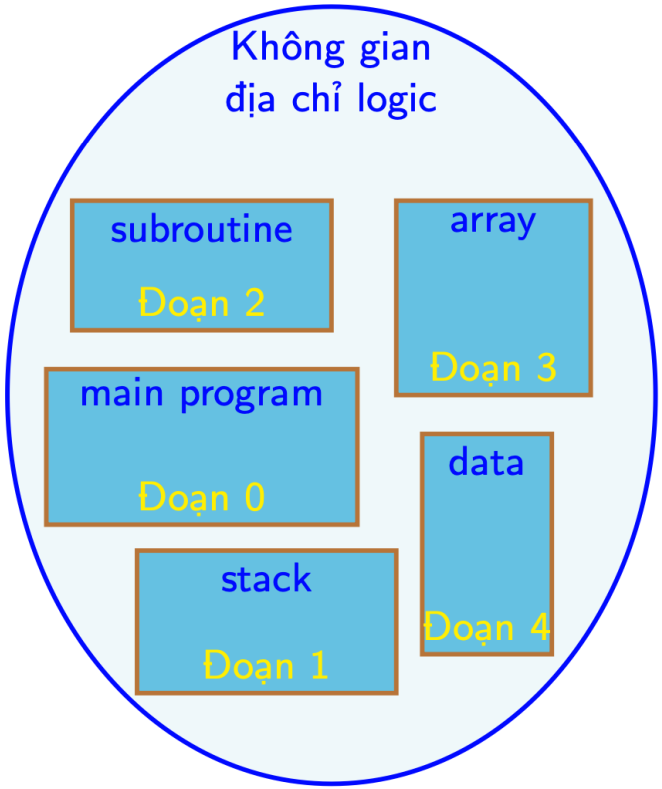
Ví dụ: Xét chiến lược phân đoạn của hệ thống sau và tính địa chỉ logic sau: $< 3, 345 >, < 1, 240 >, < 2, 120 >, < 2, 450 >$



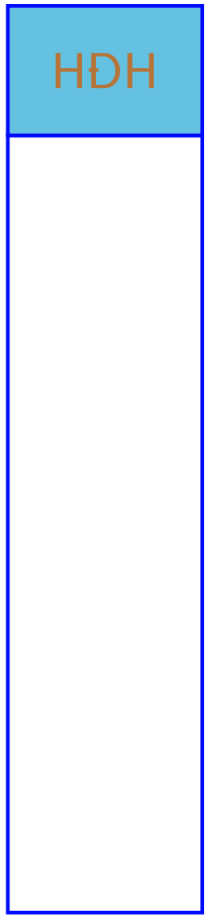
M	*A*	*L*
0	-	1000
0	-	400
0	-	400
0	-	1100
0	-	1000
SCB		



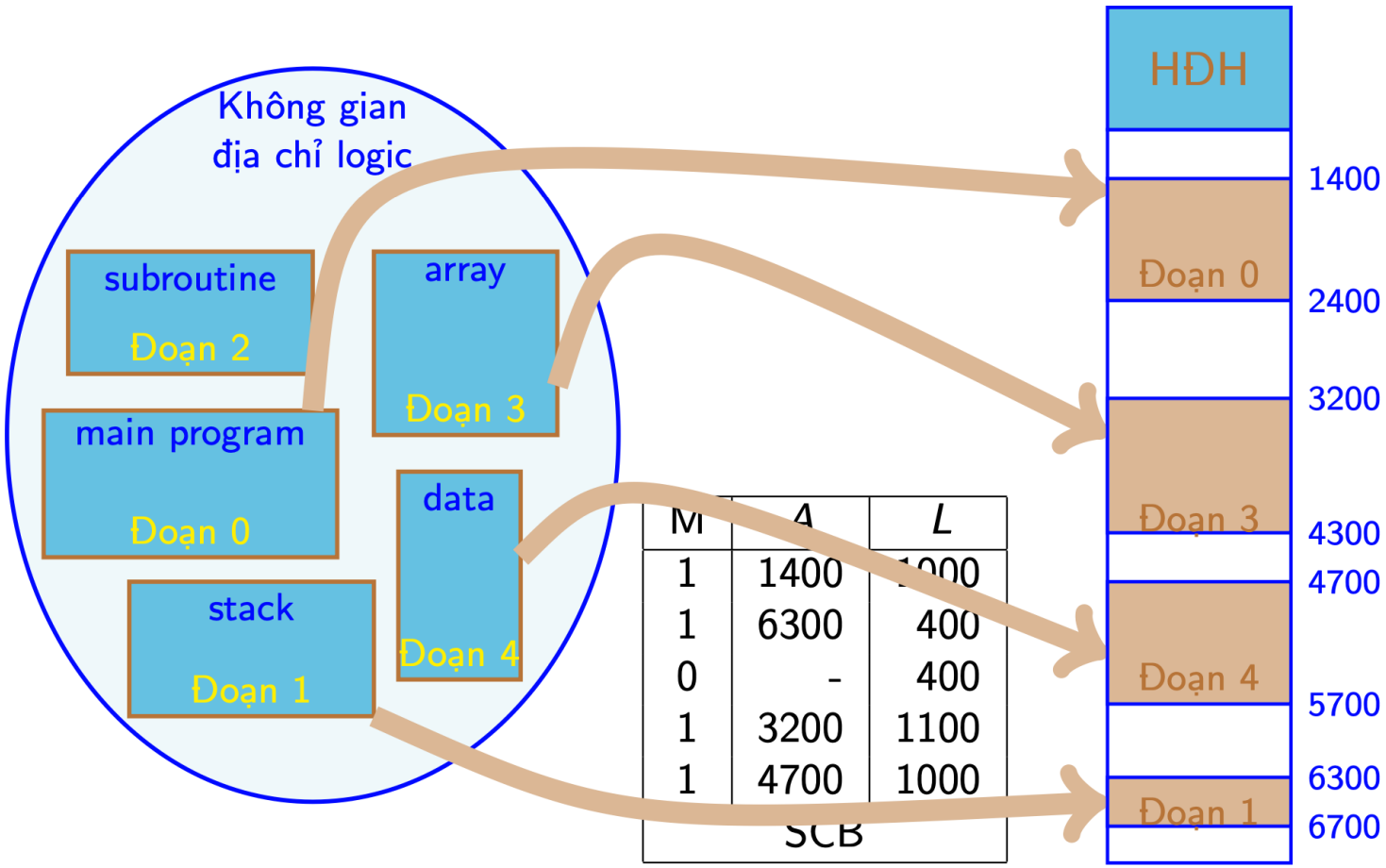
Ví dụ: Xét chiến lược phân đoạn của hệ thống sau và tính địa chỉ logic sau: $< 3, 345 >, < 1, 240 >, < 2, 120 >, < 2, 450 >$



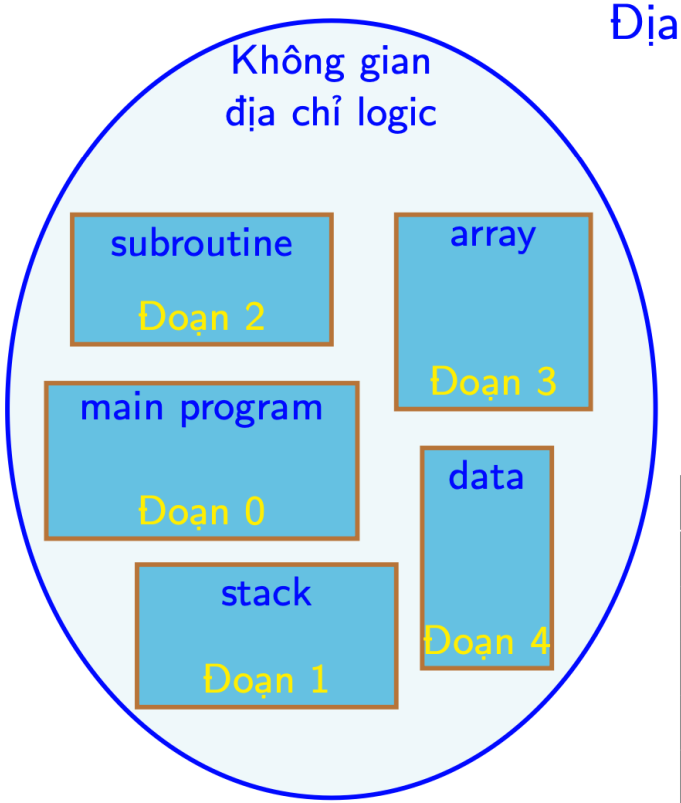
M	*A*	*L*
1	1400	1000
1	6300	400
0	-	400
1	3200	1100
1	4700	1000
SCB		



Ví dụ: Xét chiến lược phân đoạn của hệ thống sau và tính địa chỉ logic sau: $< 3, 345 >, < 1, 240 >, < 2, 120 >, < 2, 450 >$

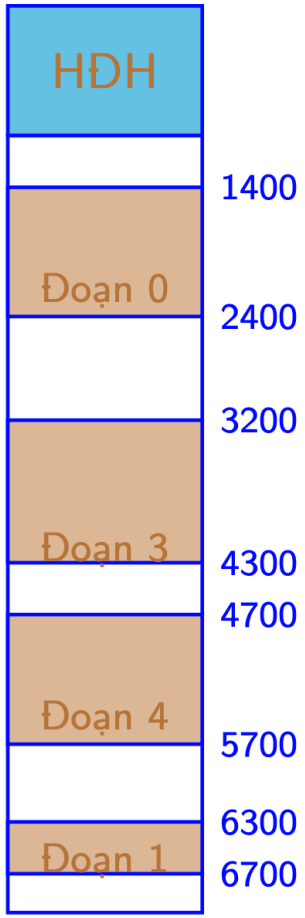


Ví dụ: Xét chiến lược phân đoạn của hệ thống sau và tính địa chỉ logic sau: $< 3, 345 >, < 1, 240 >, < 2, 120 >, < 2, 450 >$

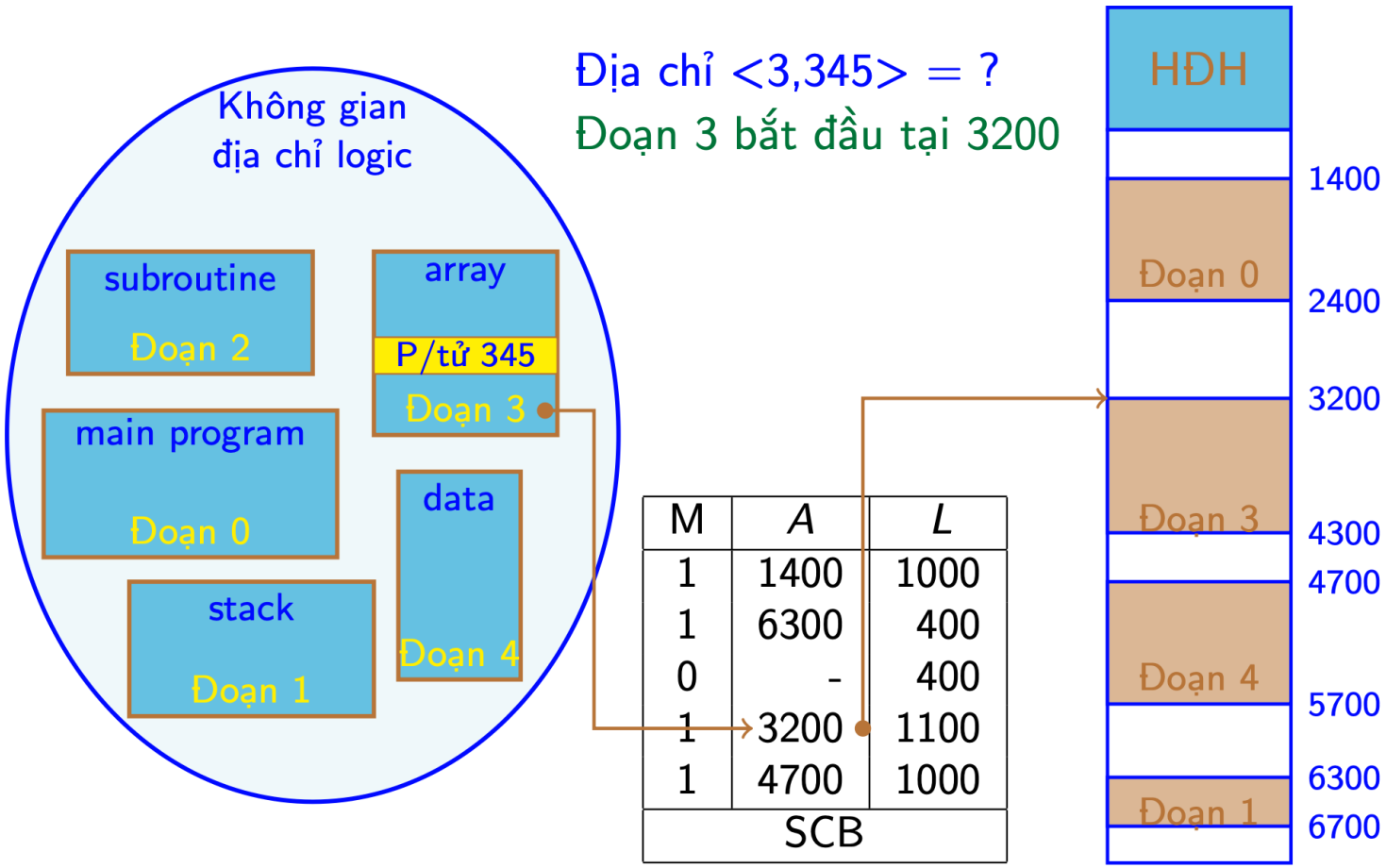


Địa chỉ <3,345> = ?

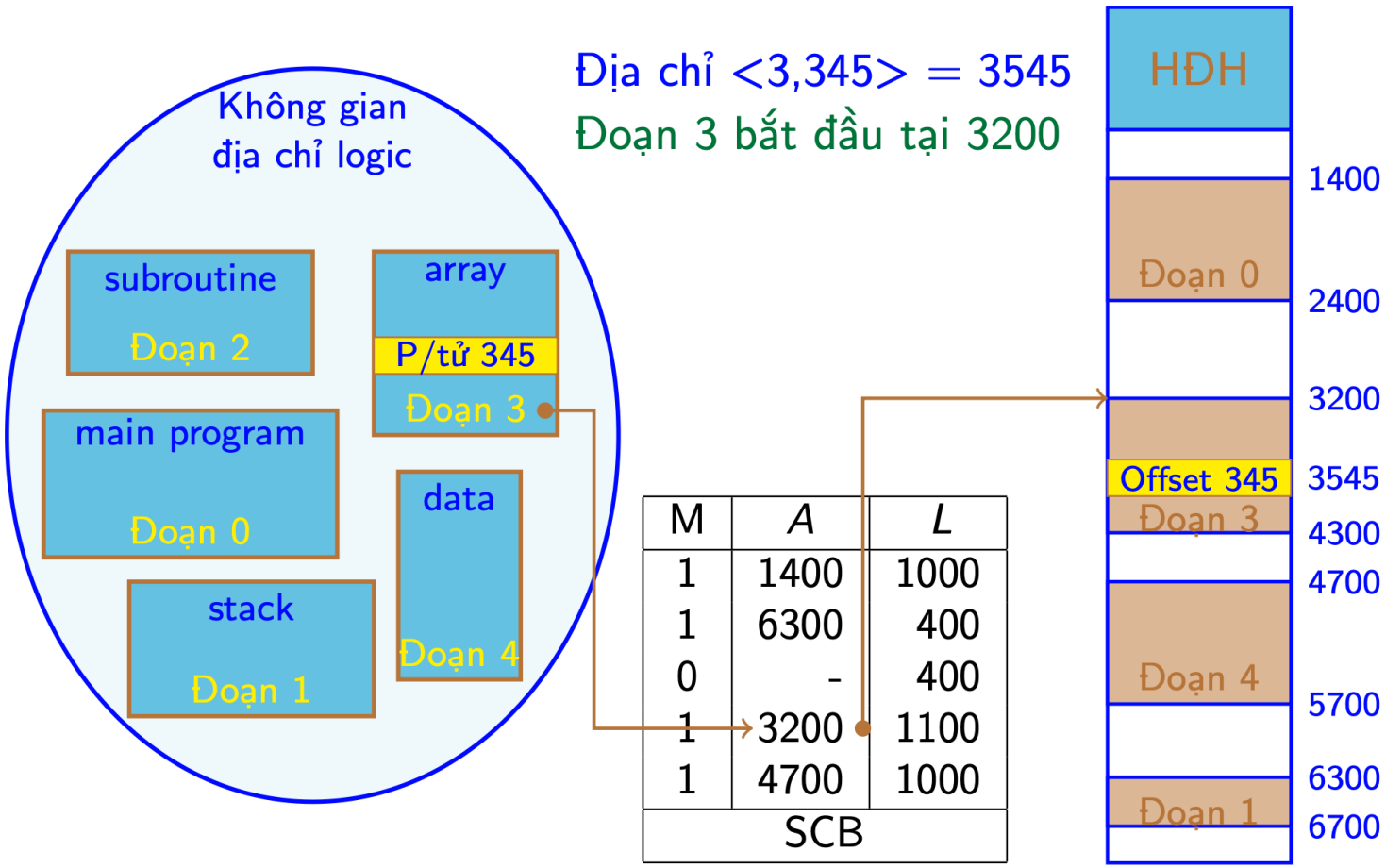
M	*A*	*L*
1	1400	1000
1	6300	400
0	-	400
1	3200	1100
1	4700	1000
SCB		



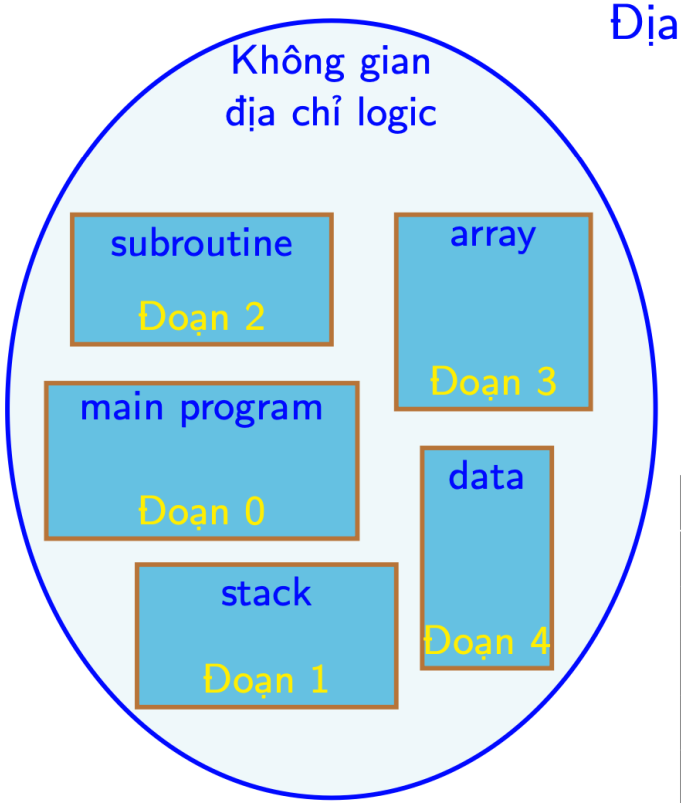
Ví dụ: Xét chiến lược phân đoạn của hệ thống sau và tính địa chỉ logic sau: $< 3, 345 >, < 1, 240 >, < 2, 120 >, < 2, 450 >$



Ví dụ: Xét chiến lược phân đoạn của hệ thống sau và tính địa chỉ logic sau: $< 3, 345 >, < 1, 240 >, < 2, 120 >, < 2, 450 >$

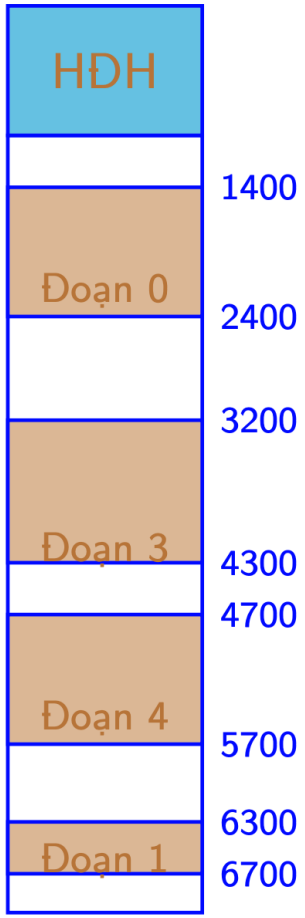


Ví dụ: Xét chiến lược phân đoạn của hệ thống sau và tính địa chỉ logic sau: $< 3, 345 >, < 1, 240 >, < 2, 120 >, < 2, 450 >$

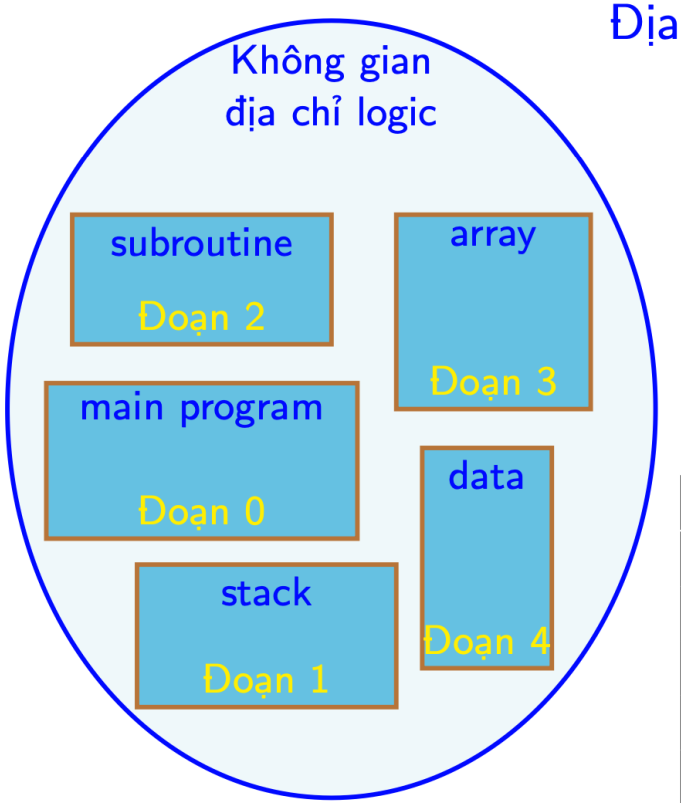


Địa chỉ <1,240> = ?

M	*A*	*L*
1	1400	1000
1	6300	400
0	-	400
1	3200	1100
1	4700	1000
SCB		

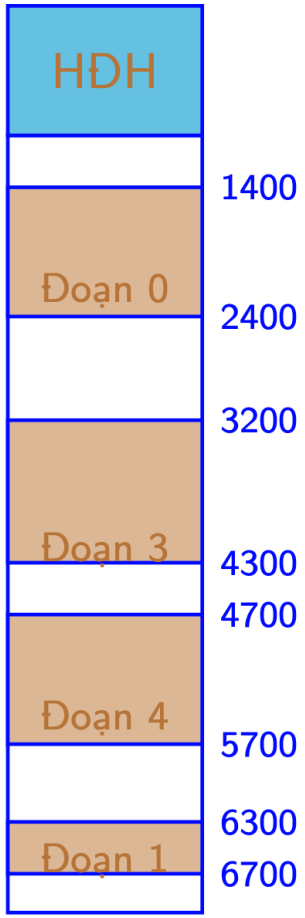


Ví dụ: Xét chiến lược phân đoạn của hệ thống sau và tính địa chỉ logic sau: $< 3, 345 >, < 1, 240 >, < 2, 120 >, < 2, 450 >$

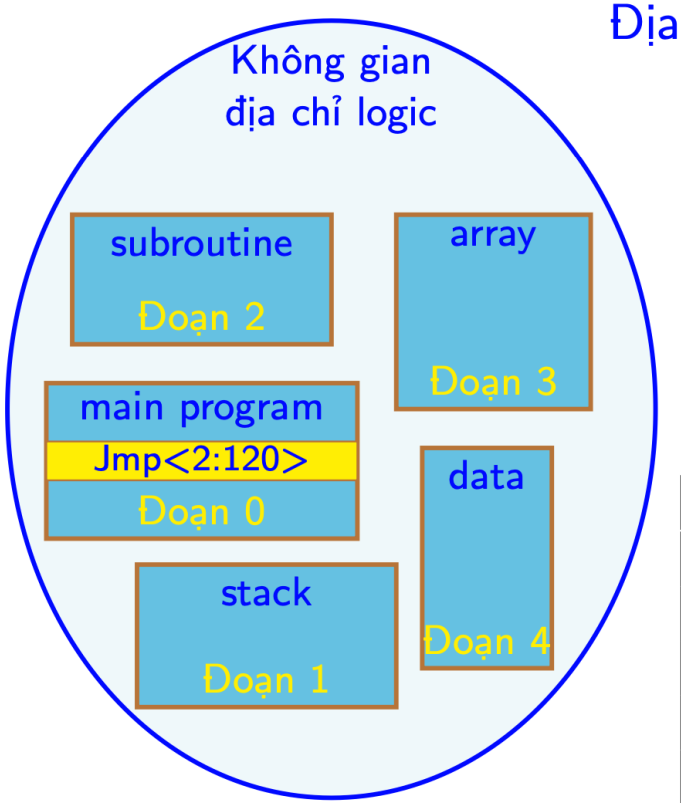


Địa chỉ <1,240> = 6540

M	*A*	*L*
1	1400	1000
1	6300	400
0	-	400
1	3200	1100
1	4700	1000
SCB		

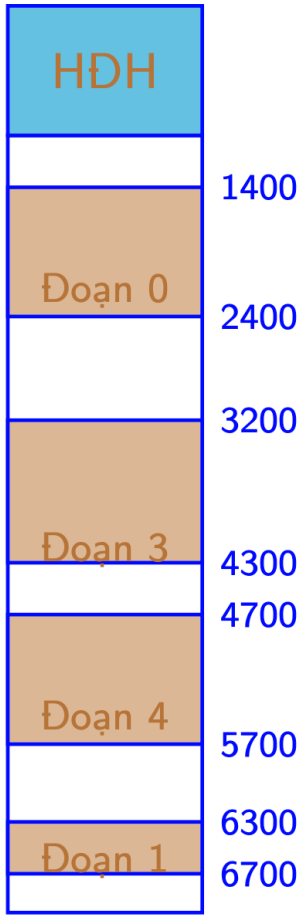


Ví dụ: Xét chiến lược phân đoạn của hệ thống sau và tính địa chỉ logic sau: $< 3, 345 >, < 1, 240 >, < 2, 120 >, < 2, 450 >$

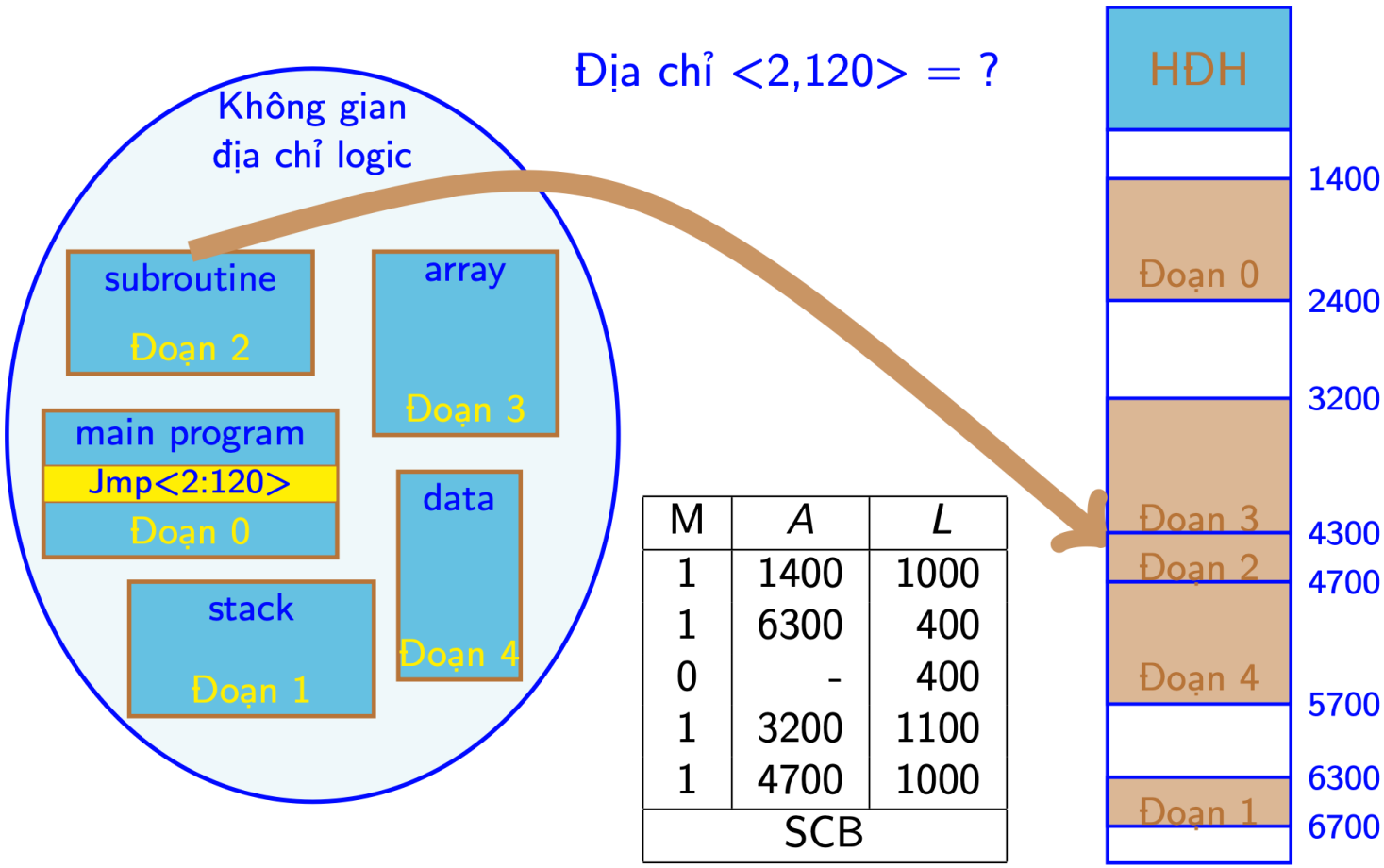


Địa chỉ <2,120> = ?

M	*A*	*L*
1	1400	1000
1	6300	400
0	-	400
1	3200	1100
1	4700	1000
SCB		

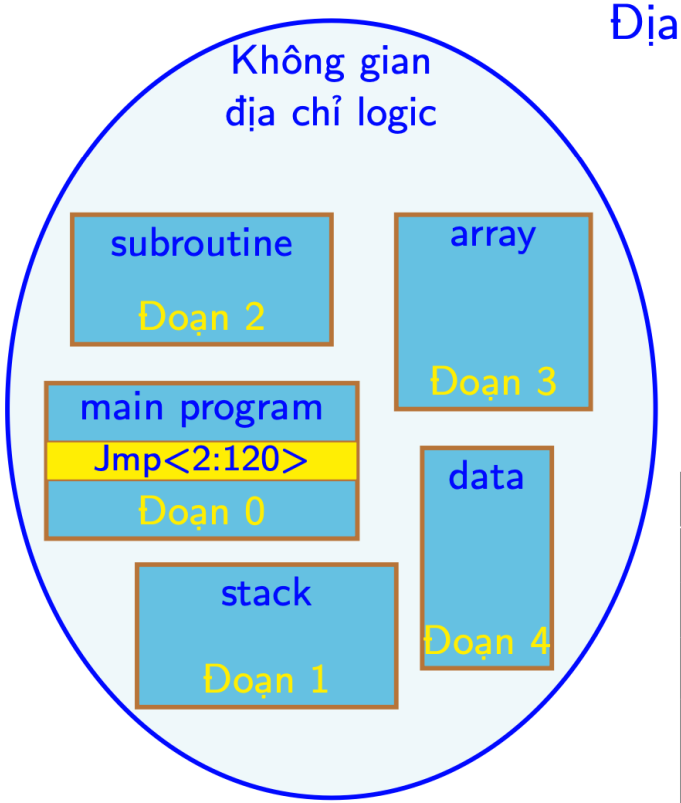


Ví dụ: Xét chiến lược phân đoạn của hệ thống sau và tính địa chỉ logic sau: $< 3, 345 >, < 1, 240 >, < 2, 120 >, < 2, 450 >$



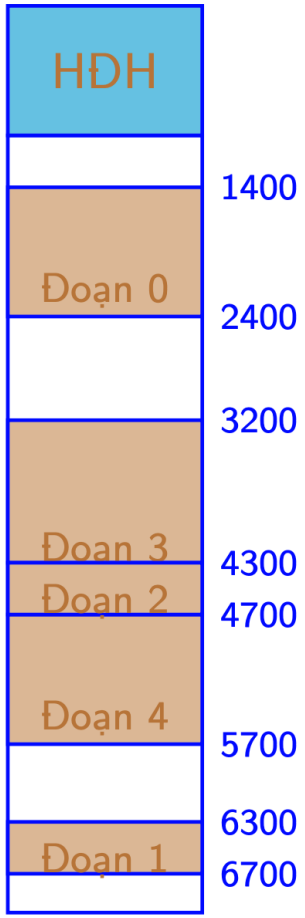
M	*A*	*L*
1	1400	1000
1	6300	400
0	-	400
1	3200	1100
1	4700	1000
SCB		

Ví dụ: Xét chiến lược phân đoạn của hệ thống sau và tính địa chỉ logic sau: $< 3, 345 >, < 1, 240 >, < 2, 120 >, < 2, 450 >$

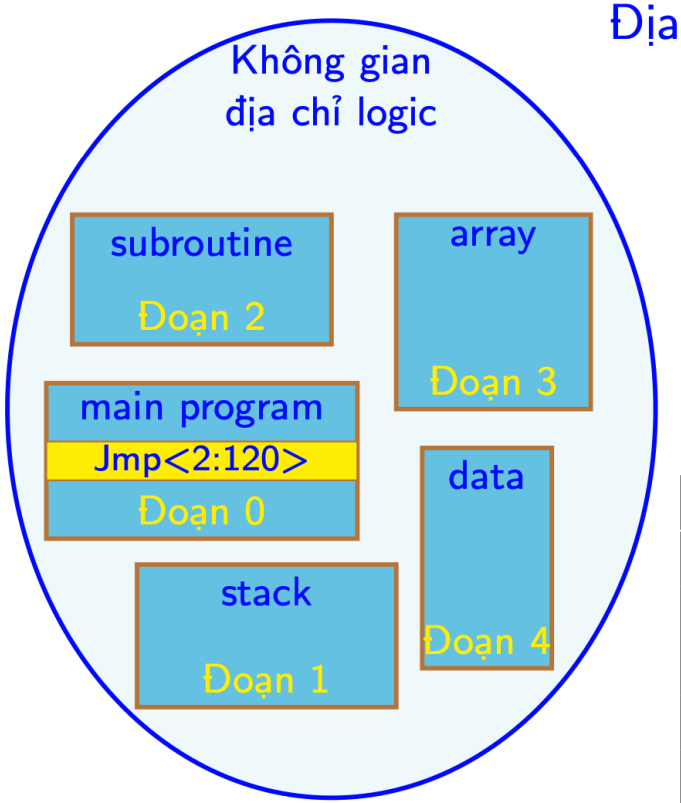


Địa chỉ <2,120> = ?

M	*A*	*L*
1	1400	1000
1	6300	400
1	4300	400
1	3200	1100
1	4700	1000
SCB		

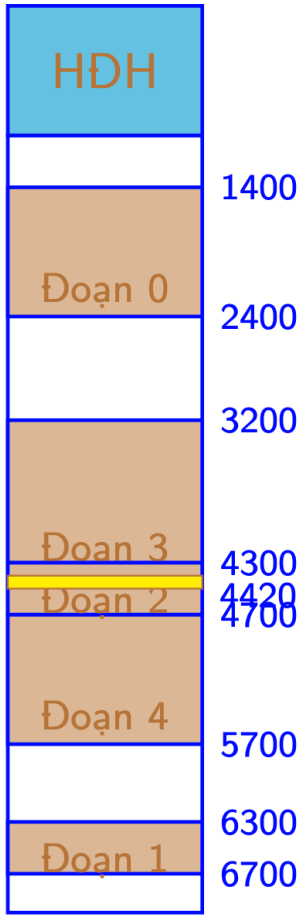


Ví dụ: Xét chiến lược phân đoạn của hệ thống sau và tính địa chỉ logic sau: $< 3, 345 >, < 1, 240 >, < 2, 120 >, < 2, 450 >$

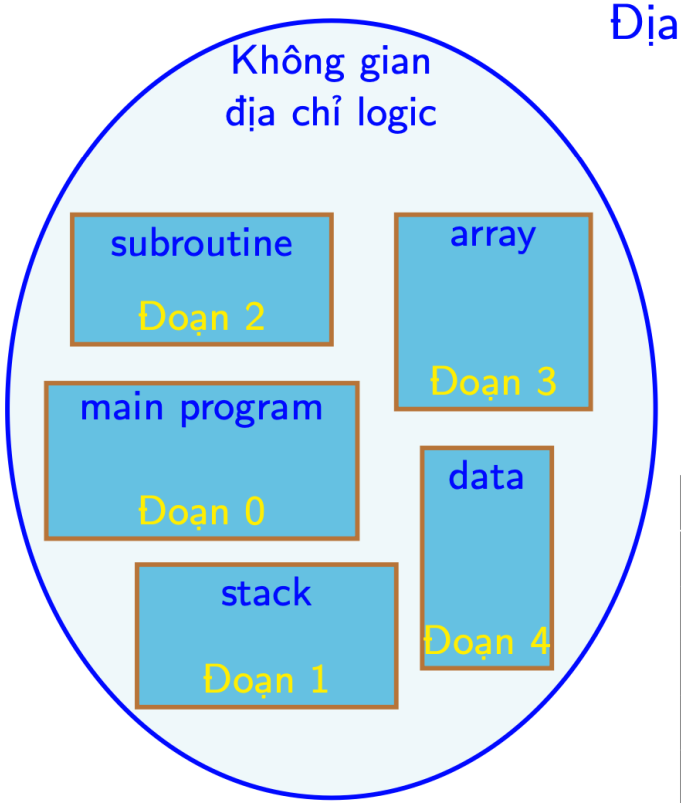


Địa chỉ <2,120> = 4420

M	*A*	*L*
1	1400	1000
1	6300	400
1	4300	400
1	3200	1100
1	4700	1000
SCB		

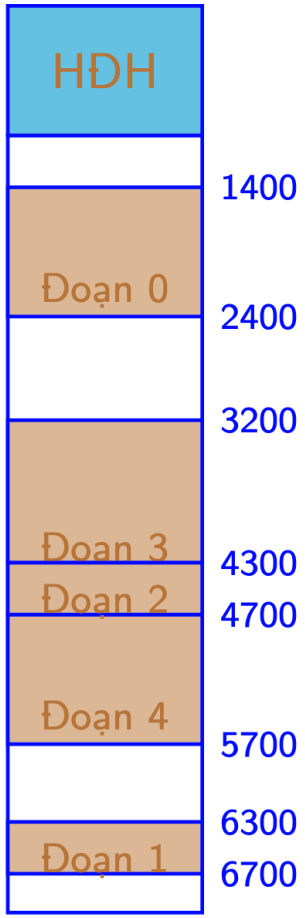


Ví dụ: Xét chiến lược phân đoạn của hệ thống sau và tính địa chỉ logic sau: $< 3, 345 >, < 1, 240 >, < 2, 120 >, < 2, 450 >$

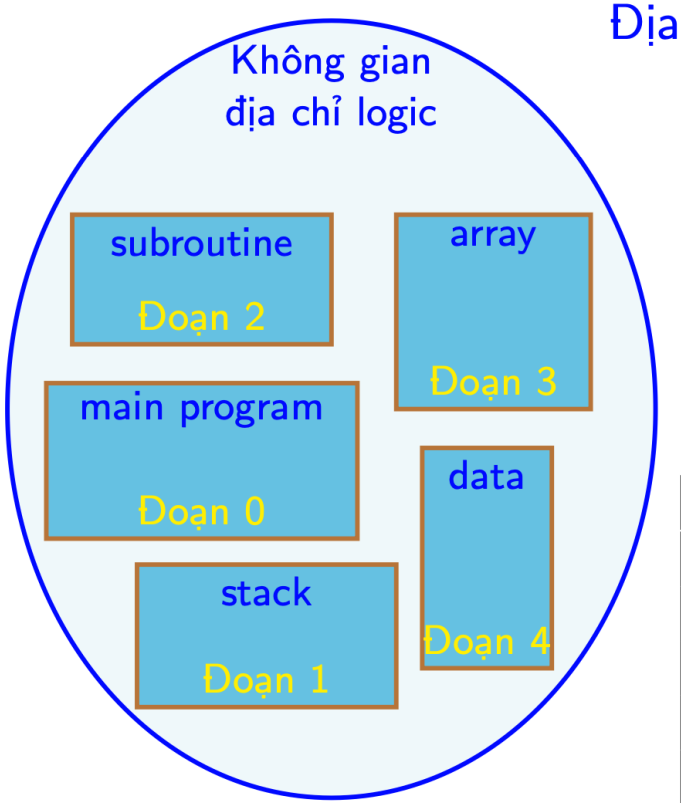


Địa chỉ <2,450> = ?

M	*A*	*L*
1	1400	1000
1	6300	400
1	4300	400
1	3200	1100
1	4700	1000
SCB		



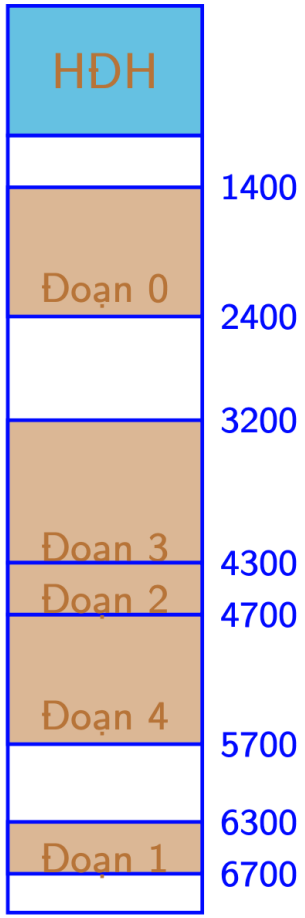
Ví dụ: Xét chiến lược phân đoạn của hệ thống sau và tính địa chỉ logic sau: $< 3, 345 >, < 1, 240 >, < 2, 120 >, < 2, 450 >$



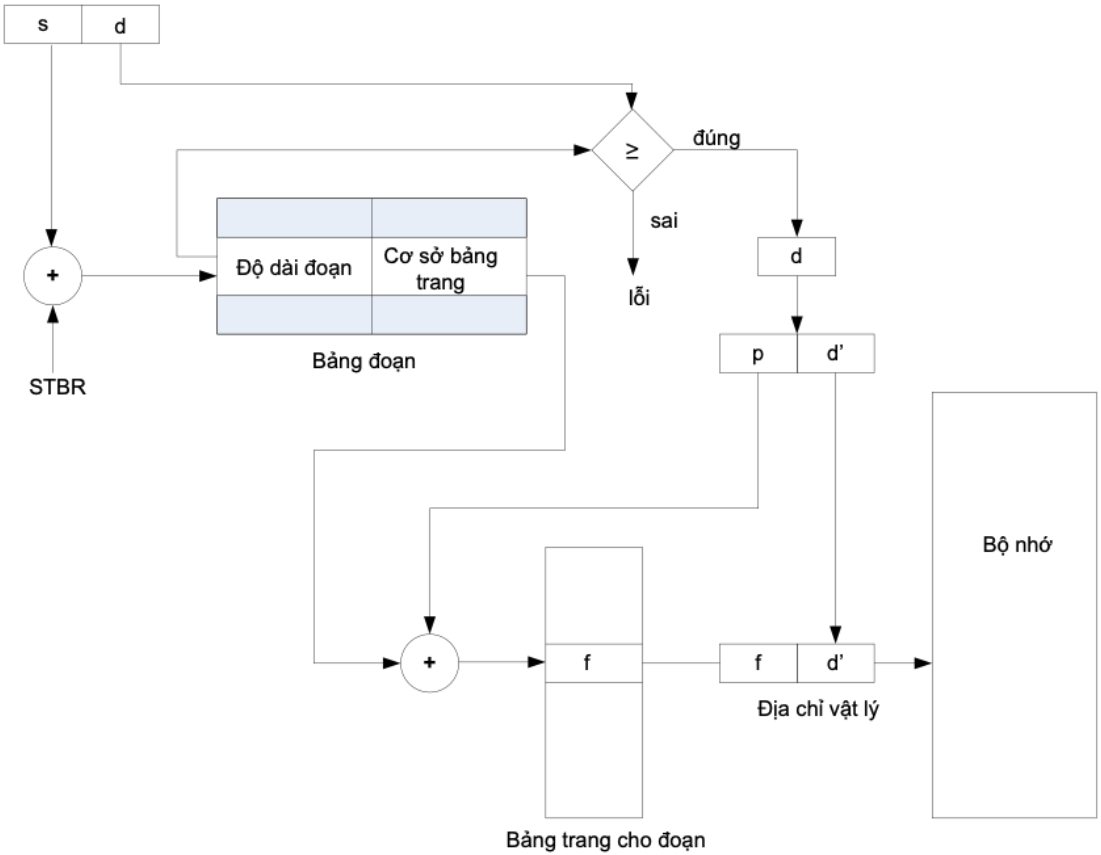
Địa chỉ <2,450> = ?

Lỗi truy nhập!

M	*A*	*L*
1	1400	1000
1	6300	400
1	4300	400
1	3200	1100
1	4700	1000
SCB		



- Phân đoạn chương trình, mỗi đoạn sẽ tiến hành phân trang
- Địa chỉ gồm: số thứ tự đoạn, số thự tự trang, độ dịch trong trang
- Tiến trình có 1 bảng phân đoạn, mỗi đoạn có 1 bảng phân trang

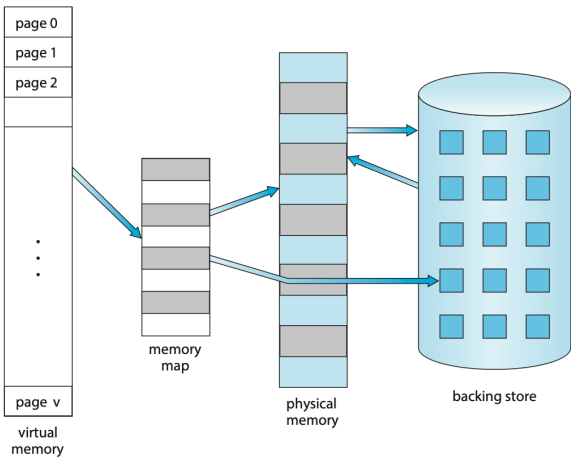


Bộ nhớ ảo

- Mục đích của hệ thống máy tính: thực hiện chương trình
- Chương trình và dữ liệu (*toàn bộ hoặc một phần*) phải nằm trong bộ nhớ chính, có thể chia thành các phần nhỏ nằm rải rác trong bộ nhớ trong khi thực hiện
- Không phải tiến trình nào khi chạy cũng sử dụng tất cả các lệnh và dữ liệu với tần số như nhau
  - Không nhất thiết toàn bộ các trang/ đoạn của một tiến trình phải có mặt đồng thời trong bộ nhớ khi tiến trình chạy
  - Các trang hoặc đoạn có thể được trao đổi từ đĩa vào bộ nhớ khi có nhu cầu truy cập tới.
- Phần chương trình chưa đưa vào bộ nhớ chính được lưu trên bộ nhớ thứ cấp (VD: *đĩa cứng*) => **Bộ nhớ ảo**
  - Cho phép lập trình viên không lo lắng về giới hạn bộ nhớ vật lý

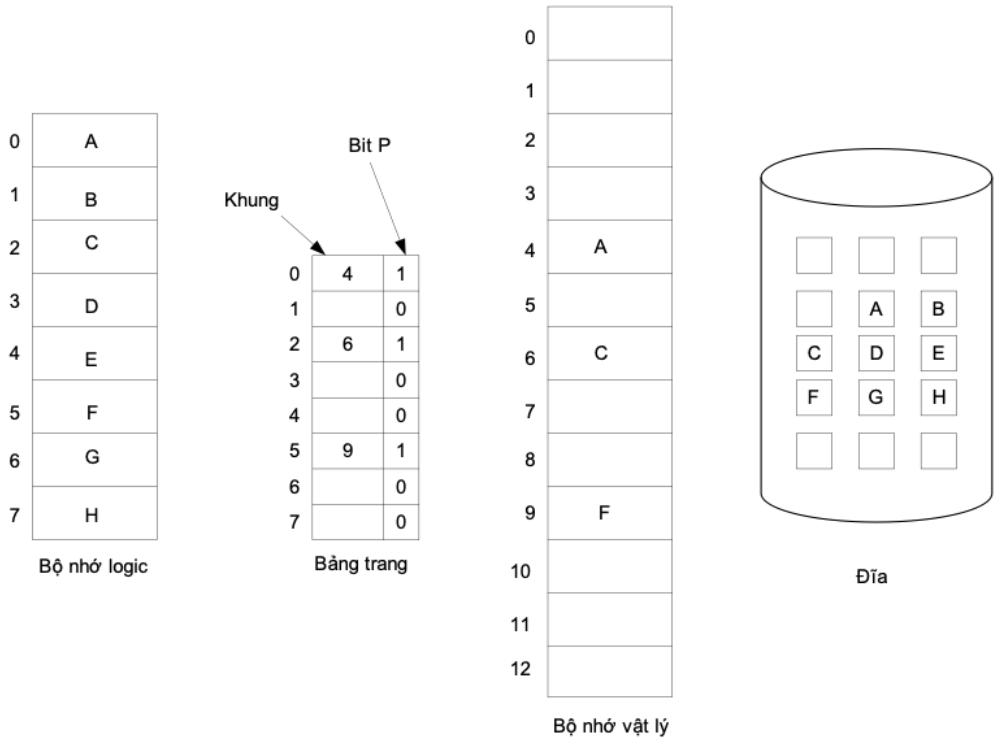
Bộ nhớ ảo:

- Kỹ thuật dùng bộ nhớ phụ lưu trữ tiến trình
- Các phần tiến trình chuyển vào-ra giữa bộ nhớ chính và bộ nhớ phụ
- Thực thi tiến trình không cần nạp toàn bộ vào bộ nhớ chính

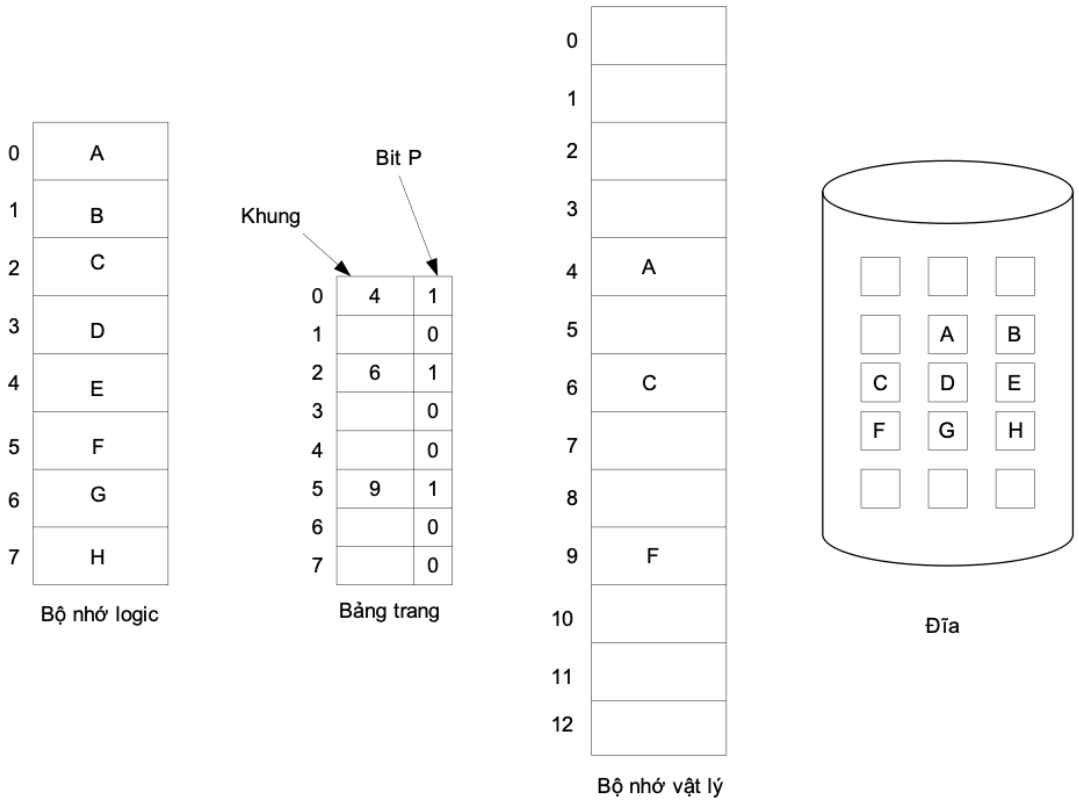


- Có hai phương pháp cài đặt kỹ thuật bộ nhớ ảo:
  - Nạp trang theo nhu cầu (*Demand paging*)
  - Nạp đoạn theo nhu cầu (*Demand segmentation*)

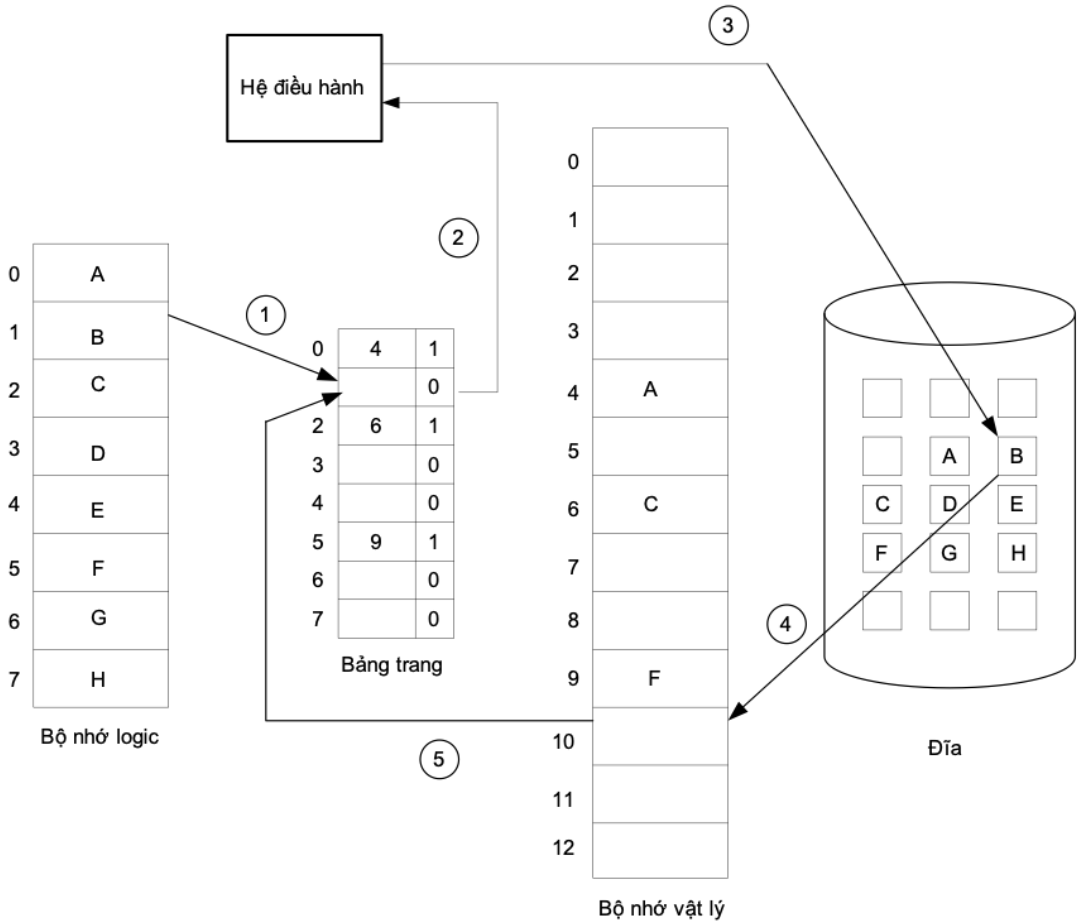
- Nạp trang theo nhu cầu dựa trên phân trang kết hợp trao đổi bộ nhớ-đĩa => Tiến trình được phân trang và các trang chứa trên đĩa
- Khi thực thi, nạp tiến trình vào MEM: chỉ nạp trang cần dùng => Các trang không nhất thiết nạp cùng một lúc



- Tiến trình gồm các trang trên đĩa và trong MEM: thêm bit P trong khoản mục bảng trang để phân biệt (P=1: đã nạp vào MEM)



- Quá trình kiểm tra và nạp trang:
  - Tiến trình truy cập tới 1 trang, kiểm tra bit P. Nếu P=1, truy cập diễn ra bình thường. Nếu P=0, xảy ra sự kiện thiếu trang



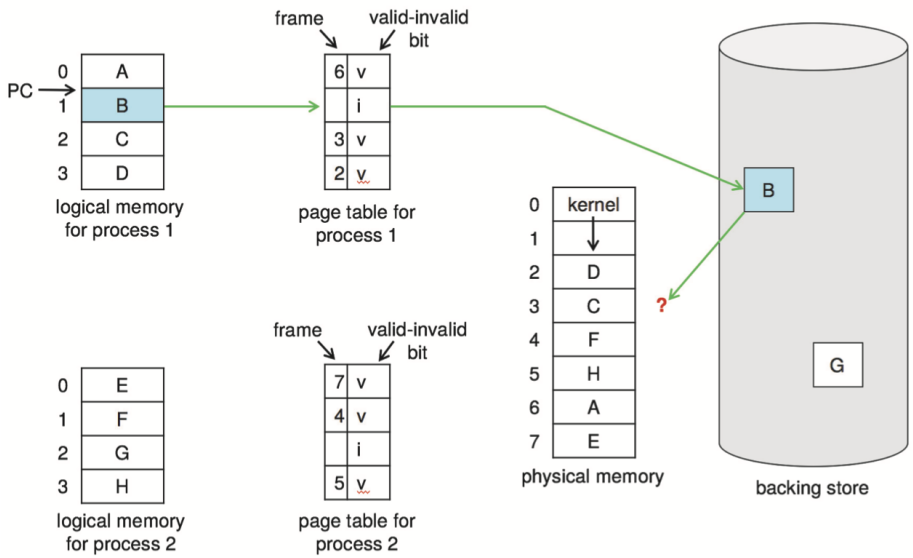
- Ngắt xử lý thiếu trang:
  - Bước 1: HĐH tìm 1 khung trống trong MEM
  - Bước 2: Đọc trang bị thiếu vào khung trang vừa tìm được
  - Bước 3: Sửa lại khoản mục tương ứng trong bảng trang: đổi bit P=1 và số khung đã cấp cho trang
  - Bước 4: Khôi phục lại trạng thái của tiến trình và thực hiện tiếp lệnh bị ngắt

- *Nạp trang hoàn toàn theo nhu cầu*:
  - Bắt đầu một tiến trình mà không nạp bất kỳ trang nào vào bộ nhớ
  - Khi con trỏ lệnh được HĐH chuyển tới lệnh đầu tiên của tiến trình để thực hiện, sự kiện thiếu trang sẽ sinh ra và trang tương ứng được nạp vào
  - Tiến trình sau đó thực hiện bình thường cho tới lần thiếu trang tiếp theo
- *Nạp trang trước: khác với nạp trang theo nhu cầu*
  - Các trang chưa cần đến cũng được nạp vào bộ nhớ
  - Không hiệu quả

- Khi xảy ra tình trạng thiếu trang: HĐH phải tìm ra một **khung** trống trong bộ nhớ đọc trang thiếu vào khung và tiến trình sau đó hoạt động bình thường.
- Bộ nhớ ảo > bộ nhớ thực và chế độ đa chương trình có lúc không còn **khung** nào trống để nạp trang mới
- Giải pháp đưa ra:
  - Kết thúc tiến trình do không thỏa mãn nhu cầu bộ nhớ
  - Trao đổi tiến trình ra đĩa và chờ thời điểm thuận lợi hơn mới nạp lại tiến trình vào bộ nhớ để thực hiện tiếp
  - Sử dụng kỹ thuật **đổi trang**

Kỹ thuật đổi trang

- Nếu không còn khung nào trống, HĐH chọn 1 khung đã cấp phát nhưng hiện không dùng tới và giải phóng khung này.
- Nội dung của khung sẽ được trao đổi ra đĩa, trang nhớ chứa trong khung sẽ được đánh dấu không được nằm trong bộ nhớ (bằng cách thay đổi bit P ở bảng trang).
- Khung được giải phóng cấp phát cho trang mới cần nạp vào



Quá trình đổi trang:

- *Bước 1:* Xác định trang trên đĩa cần nạp vào MEM
- *Bước 2*: Nếu có khung trống trên MEM thì chuyển sang B4
- *Bước 3*:
  - Lựa chọn 1 khung trên MEM để giải phóng, theo 1 thuật toán nào đó
  - Ghi nội dung khung bị đổi ra đĩa (nếu cần), cập nhật bảng trang và bảng khung
- *Bước 4*: Đọc trang cần nạp vào khung vừa giải phóng; cập nhật bảng trang và bảng khung
- *Bước 5*: Thực hiện tiếp tiến trình từ điểm bị dừng trước khi đổi trang

Đổi trang có ghi và đổi trang không ghi:

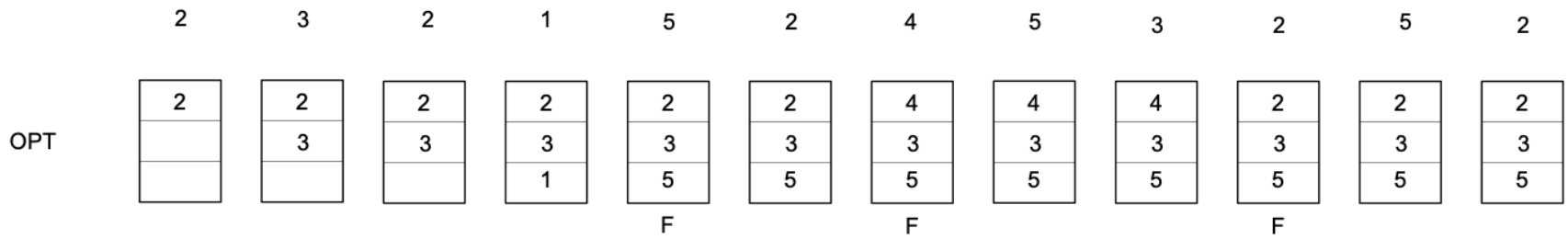
- Nếu nhu cầu đổi trang xuất hiện khi nạp trang mới, thời gian nạp trang sẽ tăng đáng kể do thêm nhu cầu ghi trạng bị đổi ra đĩa.
- Giúp nhận biết các trang không thay đổi từ lúc nạp và không ghi ngược ra đĩa
- Sử dụng thêm bit sửa đổi M trong khoản mục trang để đánh dấu trang đã bị sửa đổi (1) hay chưa (0)
- Các khung bị khóa:
  - Khi tìm các trang để giải phóng và đổi trang, HĐH sẽ trừ ra một số khung
  - Một số khung sẽ không bị giải phóng trong quá trình tìm kiếm khung để đổi trang là *các khung bị khóa*
  - VD: *Khung chứa tiến trình nhân của HĐH, chứa các cấu trúc thông tin điều khiển quan trọng*
  - Nhận biết bởi 1 bit riêng chứa trong khoản mục

Các chiến lược đổi trang:

- OPT/MIN: Thuật toán đổi trang tối ưu
- FIFO (First In First Out): Vào trước ra trước
- LRU (Least Recently Used): Trang có lần sử dụng cuối cách lâu nhất
- CLOCK: Thuật toán đồng hồ
- LFU (Least Frequently used):Tần xuất sử dụng thấp nhất
- MFU (Most Frequently used): Tần xuất sử dụng cao nhất
- ...

Đổi trang tối ưu (OPT):

- *Chọn trang sẽ không được dùng tới trong khoảng thời gian lâu nhất trong tương lai để đổi hay trang có lần sử dụng tiếp theo xa nhất*
- Cho phép giảm tối thiểu sự thiếu trang và tối ưu theo tiêu chuẩn này
- HĐH đoán trước được nhu cầu sử dụng các trang trong tương lai
- Không áp dụng trong thực tế chỉ so sánh với các chiến lược khác



Bài tập: *Đổi trang tối ưu (OPT):*
Giả sử tiến trình được cấp 3 khung, không gian nhớ của tiến trình có 5 trang và các trang của tiến trình được truy cập theo thứ tự như sau:

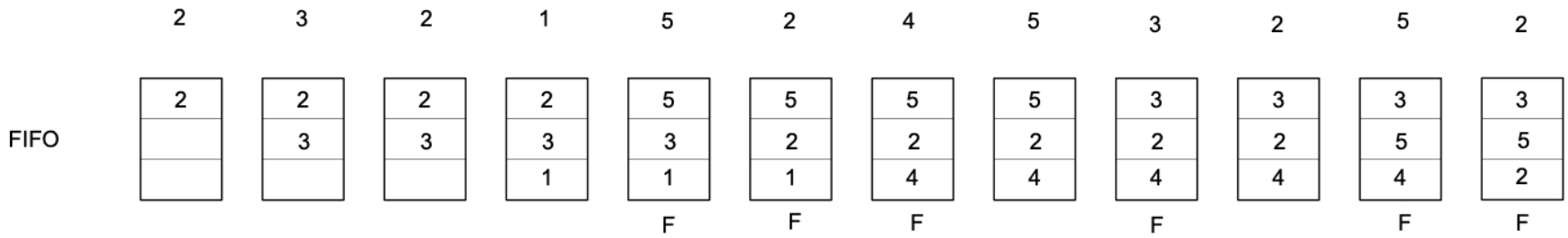
2, 3, 2, 1, 5, 2, 4, 5, 3, 2, 5, 2.

Xác định thứ tự nạp đổi trang sử dụng thuật toán tối ưu OPT.

Vào trước ra trước (FIFO):

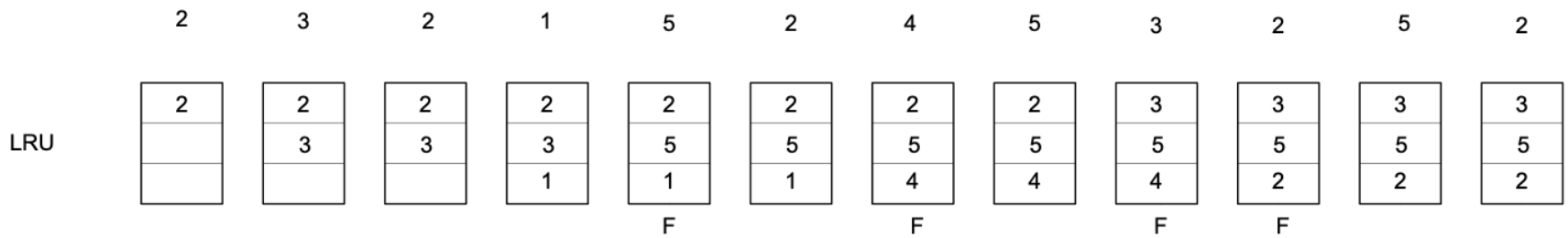
- Trang nào được nạp vào trước thì bị đổi ra trước
- Phương pháp đơn giản nhất
- Trang bị trao đổi là trang nằm lâu nhất trong bộ nhớ

Ví dụ: Giả sử tiến trình được cấp 3 khung, không gian nhớ của tiến trình có 5 trang và các trang của tiến trình được truy cập theo thứ tự như sau: 2, 3, 2, 1, 5, 2, 4, 5, 3, 2, 5, 2



Đổi trang có lần sử dụng cuối cách lâu nhất (LRU):

- Trang bị đổi là trang mà thời gian từ lần truy cập cuối cùng đến thời điểm hiện tại là lâu nhất (lâu rồi không truy cập thì thay thế)
- Theo nguyên tắc cục bộ về thời gian, đó chính là trang ít có khả năng sử dụng tới nhất trong tương lai
- Thực tế LRU cho kết quả tốt gần như phương pháp đổi trang tối ưu



Đổi trang có lần sử dụng cuối cách lâu nhất (LRU):

- Xác định được trang có lần truy cập cuối diễn ra cách thời điểm hiện tại lâu nhất?
- Sử dụng biến đếm:
  - Mỗi khoản mục của bảng phân trang sẽ có thêm một trường chứa thời gian truy cập trang lần cuối - Là thời gian logic
  - CPU cũng được thêm một bộ đếm thời gian lôgic này
  - Chỉ số của bộ đếm tăng mỗi khi xảy ra truy cập bộ nhớ
  - Mỗi khi một trang nhớ được truy cập, chỉ số của bộ đếm sẽ được ghi vào trường thời gian truy cập trong khoản mục của trang đó
  - Trường thời gian luôn chứa thời gian truy cập trang lần cuối
  - Trang bị đổi là trang có giá trị thời gian nhỏ nhất

Đổi trang có lần sử dụng cuối cách lâu nhất (LRU):

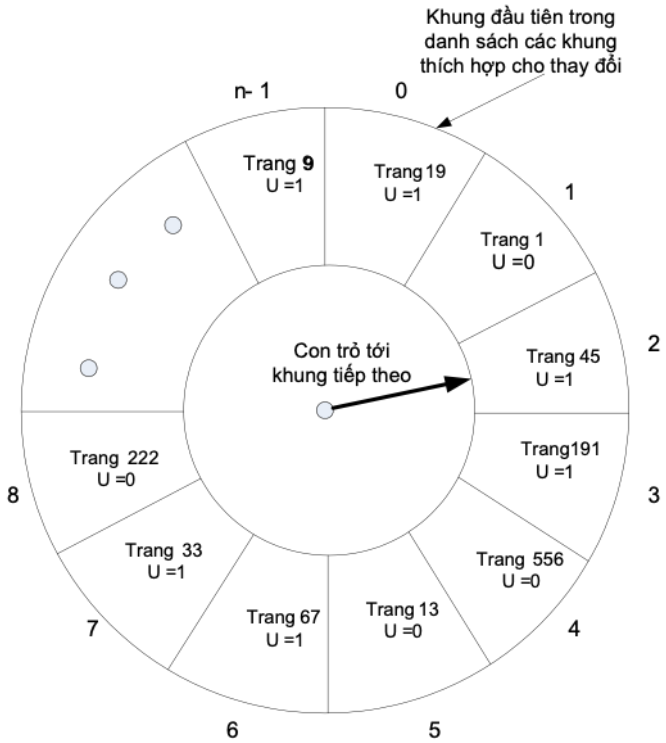
- Sử dụng ngăn xếp
  - Ngăn xếp đặc biệt được sử dụng để chứa các số trang
  - Mỗi khi một trang nhớ được truy cập, số trang sẽ được chuyển lên đỉnh ngăn xếp
  - Đỉnh ngăn xếp sẽ chứa trang được truy cập gần đây nhất
  - Đáy ngăn xếp chính là trang LRU, tức là trang cần trao đổi
  - Tránh tìm kiếm trong bảng phân trang
  - Thích hợp thực hiện bằng phần mềm

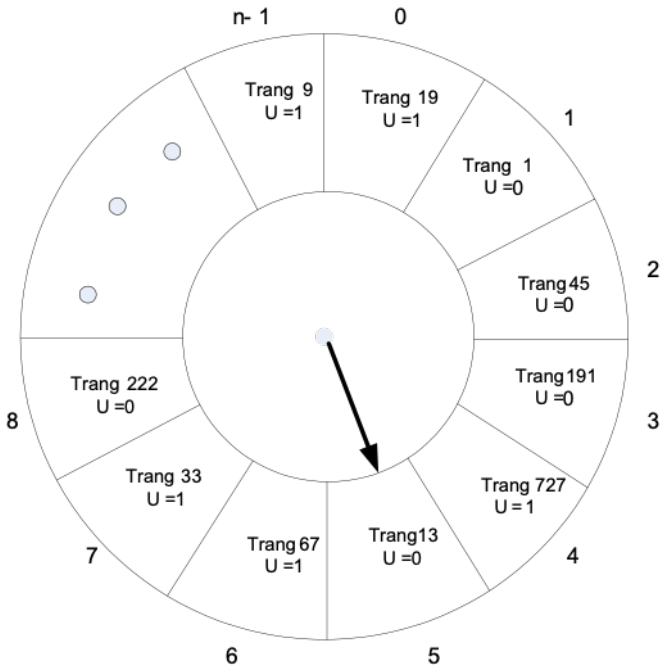
CLOCK: Thuật toán đồng hồ Cải tiến FIFO nhằm tránh thay những trang được nạp vào lâu vẫn có khả năng sử dụng

- Mỗi trang được gắn thêm 1 bit sử dụng U
  - Khi trang được truy cập đọc/ ghi: U = 1
  - Ngay khi trang được đọc vào bộ nhớ: U =1
- Các khung/ trang có thể bị đổi được liên kết vào 1 danh sách vòng
- Khi một trang nào đó bị đổi, con trỏ được dịch chuyển để trỏ vào trang tiếp theo trong danh sách
- Khi có nhu cầu đổi trang, HDH kiểm tra trang đang bị trỏ tới
  - Nếu U=0: trang sẽ bị đổi ngay
  - Nếu U=1: đặt U=0 và trỏ sang trang tiếp theo, lặp lại quá trình
- U của tất cả các trang trong danh sách =1 con trỏ sẽ quay đúng 1 vòng, đặt U của các trang =0 và trang hiện thời bị trỏ sẽ bị đổi

CLOCK: Thuật toán đồng hồ

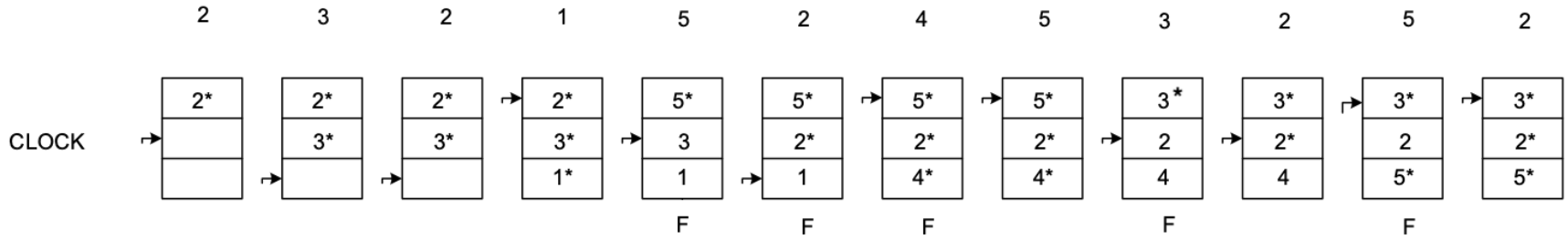
Cần nạp trang 727





CLOCK: Thuật toán đồng hồ

Ví dụ: tiến trình được cấp 3 khung, các trang nhớ của tiến trình theo thứ tự truy cập: 2, 3, 2, 1, 5, 2, 4, 5, 3, 2, 5, 2



- Căn cứ vào 2 thông tin để đưa ra quyết định đổi trang:
  - Thời gian trang được tải vào, thể hiện qua vị trí trang trong danh sách giống như FIFO
  - Gần đây trang có được sử dụng hay không, thể hiện qua bit U
- Việc kiểm tra thêm bit U tương tự việc cho trang thêm khả năng được giữ trong bộ nhớ

Thuật toán đồng hồ cải tiến:

- Sử dụng thêm thông tin về việc nội dung trang có bị thay đổi hay không bằng bit M
- Kết hợp bit U và M, có 4 khả năng:
  - U=0, M=0: trang gần đây không được truy cập và nội dung cũng không bị thay đổi, rất thích hợp để bị đổi ra ngoài
  - U=0, M=1: trang có nội dung thay đổi nhưng gần đây không được truy cập, cũng là ứng viên để đổi ra ngoài
  - U=1, M=0: trang mới được truy cập gần đây và do vậy theo nguyên lý cục bộ về thời gian có thể sắp được truy cập tiếp
  - U=1, M=1: trang có nội dung bị thay đổi và mới được truy cập gần đây, chưa thật thích hợp để đổi

Thuật toán đồng hồ cải tiến:

- Các bước thực hiện đổi trang:
- Bước 1:
  - Bắt đầu từ vị trí hiện tại của con trỏ, kiểm tra các trang
  - Trang đầu tiên có U=0 và M=0 sẽ bị đổi
  - Chỉ kiểm tra mà không thay đổi nội dung bit U, bit M
- Bước 2:
  - Nếu quay hết 1 vòng mà không tìm được trang có U và M bằng 0 thì quét lại danh sách lần 2
  - Trang đầu tiên có U=0, M=1 sẽ bị đổi
  - Đặt bit U của những trang đã quét đến nhưng được bỏ qua là 0
- Nếu chưa tìm được thì lặp lại bước 1 và cả bước 2 nếu cần

- HĐH dành ra một số khung trống được kết nối thành danh sách liên kết gọi là các trang đệm
- Trang bị đổi như bình thường nhưng nội dung trang này không bị xóa ngay khỏi bộ nhớ
- Khung chứa trang được đánh dấu là khung trống và thêm vào cuối danh sách trang đệm
- Trang mới sẽ được nạp vào khung đứng đầu trong danh sách trang đệm
- Tới thời điểm thích hợp, hệ thống sẽ ghi nội dung các trang trong danh sách đệm ra đĩa

- Kỹ thuật đệm trang cho phép cải tiến tốc độ:
- Nếu trang bị đổi có nội dung cần ghi ra đĩa, HDH vẫn có thể nạp trang mới vào ngay
  - Việc ghi ra đĩa sẽ được lùi lại tới một thời điểm sau
  - Thao tác ghi ra đĩa có thể thực hiện đồng thời với nhiều trang nằm trong danh sách được đánh dấu trống.
- Trang bị đổi vẫn được giữ trong bộ nhớ một thời gian:
  - Nếu có yêu cầu truy cập trong thời gian này, trang sẽ được lấy ra từ danh sách đệm và sử dụng ngay mà không cần nạp lại từ đĩa
  - Vùng đệm đóng vai trò giống như bộ nhớ cache

Cấp phát khung trang

Giới hạn khung

- Với bộ nhớ ảo, tiến trình không nhất thiết phải nằm hoàn toàn trong bộ nhớ máy tính. Một số trang của tiến trình được cấp phát khung nhớ trong khi một số trang khác nằm tạm trên đĩa.
- HĐH cấp phát bao nhiêu khung cho mỗi tiến trình? Khi số lượng khung tối đa cấp cho tiến trình giảm tới mức nào đó, lỗi thiếu trang diễn ra thường xuyên
- Đặt giới hạn tối thiểu các khung cấp phát cho tiến trình
- Khi số lượng khung cấp cho tiến trình tăng tới mức nào đó thì việc tăng thêm khung cho tiến trình không làm giảm đáng kể tần suất thiếu trang nữa
- Cấp phát số lượng khung cố định và số lượng khung thay đổi.

Cấp phát khung trang (cont.)

Cấp phát số lượng khung cố định

- Cấp cho tiến trình một số lượng cố định khung để chứa các trang nhớ
- Số lượng được xác định vào thời điểm tạo mới tiến trình và không thay đổi trong quá trình tiến trình tồn tại
- Cấp phát bằng nhau:
  - Các tiến trình được cấp số khung tối đa bằng nhau
  - Số lượng được xác định dựa vào kích thước MEM và mức độ đa chương trình mong muốn
- Cấp phát không bằng nhau:
  - Các tiến trình được cấp số khung tối đa khác nhau
    - Cấp số khung tỉ lệ thuận với kích thước tiến trình
    - Có mức ưu tiên

Cấp phát khung trang (cont.)

Cấp phát số lượng khung thay đổi

- Số lượng khung tối đa cấp cho mỗi tiến trình có thể thay đổi trong quá trình thực hiện
- Việc thay đổi phụ thuộc vào tình hình thực hiện của tiến trình
- Cho phép sử dụng bộ nhớ hiệu quả hơn phương pháp cố định
- Cần theo dõi và xử lý thông tin về tình hình sử dụng bộ nhớ của tiến trình

- Cấp phát toàn thể:
  - Cho phép tiến trình đổi trang mới vào bất cứ khung nào (không bị khóa), kể cả khung đã được cấp phát cho tiến trình khác
- Cấp phát cục bộ:
  - Trang chỉ được đổi vào khung đang được cấp cho chính tiến trình đó
- Phạm vi cấp phát có quan hệ mật thiết với số lượng khung tối đa:
  - Số lượng khung cố định tương ứng với phạm vi cấp phát cục bộ
  - Số lượng khung thay đổi tương ứng với phạm vi cấp phát toàn thể

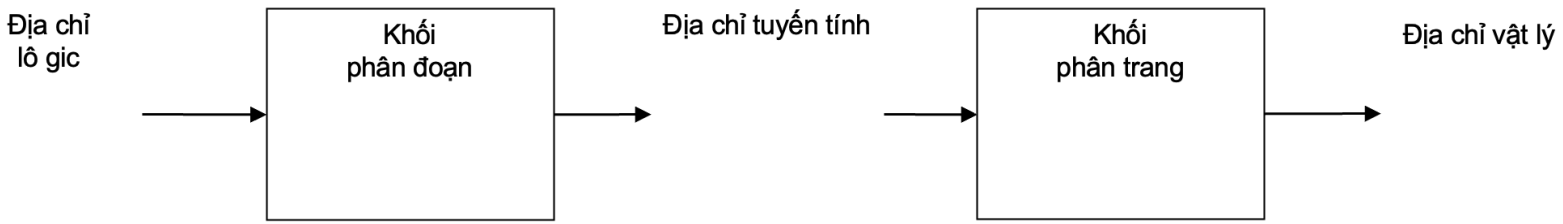
- Trì trệ là tình trạng đổi trang liên tục do không đủ bộ nhớ
- Thời gian đổi trang của tiến trình lớn hơn thời gian thực hiện
- Xảy ra khi:
  - Kích thước bộ nhớ hạn chế
  - Tiến trình đòi hỏi truy cập đồng thời nhiều trang nhớ
  - Hệ thống có mức độ đa chương trình cao
- Khi tiến trình rơi vào tình trạng trì trệ, tần suất thiếu trang tăng đáng kể
- Sử dụng để phát hiện và giải quyết vấn đề trì trệ

Kiểm soát tần suất thiếu trang

- Theo dõi và ghi lại tần suất thiếu trang
- Có thể đặt ra giới hạn trên và giới hạn dưới cho tần suất thiếu trang của tiến trình
  - Tần suất vượt giới hạn trên:
    - Cấp thêm cho tiến trình khung mới
    - Nếu không thể tìm khung để cấp thêm, tiến trình sẽ bị treo hoặc bị kết thúc
  - Tần suất thiếu trang thấp hơn giới hạn dưới: thu hồi một số khung của tiến trình

Quản lý bộ nhớ trong Intel Pentium

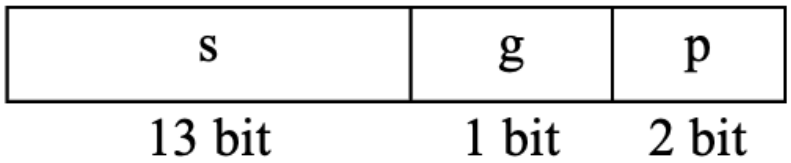
- Vi xử lý Pentium của Intel hỗ trợ cơ chế quản lý bộ nhớ: phân đoạn được kết hợp với phân trang
- Không gian nhớ của tiến trình bao gồm nhiều đoạn, mỗi đoạn có thể có kích thước khác nhau và được phân trang trước khi đặt vào bộ nhớ.
- Ánh xạ địa chỉ: 2 giai đoạn



Quản lý bộ nhớ trong Intel Pentium (cont.)

- Cho phép tiến trình có tối đa 16KB (hơn 16000) đoạn, mỗi đoạn có kích thước tối đa 4GB
- Không gian nhớ lô gic được chia thành hai phần:
  - Phần 1: dành riêng cho tiến trình, bao gồm tối đa 8KB đoạn
  - Phần 2: dùng chung cho tất cả tiến trình, bao gồm cả HĐH, và cũng gồm tối đa 8KB đoạn
- Thông tin quản lý tiến trình thuộc phần thứ nhất và phần thứ hai năm trong bảng LDT(Local Descriptor Table) và GDT (Global Descriptor Table): chứa thông tin quản lý :
  - Mỗi ô có kích thước 8 byte: chứa địa chỉ cơ sở và giới hạn của đoạn tương ứng
- Để tăng tốc ánh xạ địa chỉ, Petinum có 6 thanh ghi đoạn: cho phép tiến trình truy cập đồng thời 6 đoạn.
- Thông tin về đoạn được chứa trong 6 thanh ghi 8 byte
- Địa chỉ logic gồm (selector, offset):

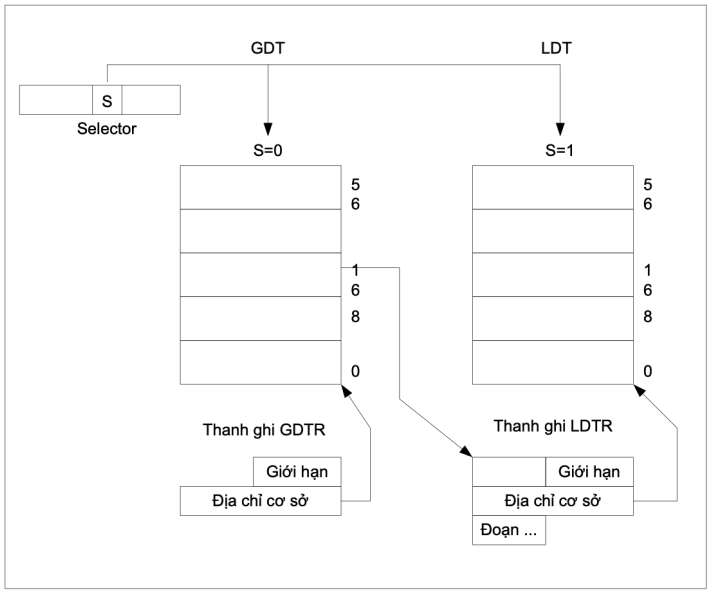
Quản lý bộ nhớ trong Intel Pentium (cont.)

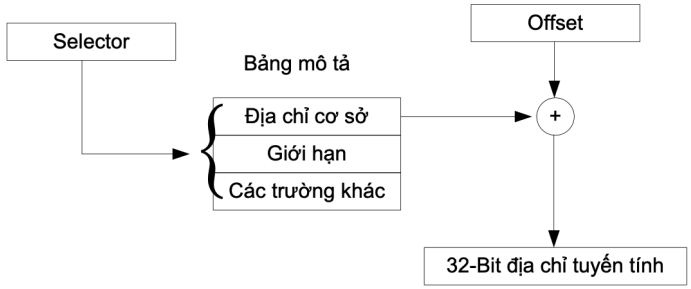


- Selector: chọn ô tương ứng từ hai bảng mô tả LDT, GDT
- S: là số thứ tự đoạn
- G: cho biết đoạn thuộc GDT (g=0) hay LDT(g=1)
- P: cho biết chế độ bảo vệ (p=0 là chế độ nhân, p=3 là chế độ người dùng)
- Offset: độ dịch trong đoạn, kích thước 32bit

Quản lý bộ nhớ trong Intel Pentium (cont.)

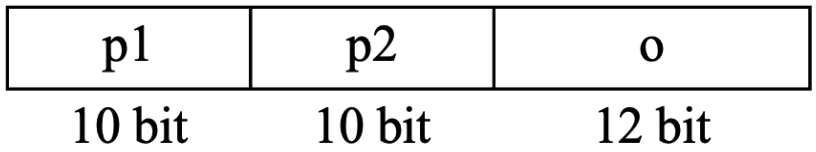
- Biến đổi địa chỉ logic thành địa chỉ tuyến tính:
  - Trước hết phần selector được sử dụng để tìm ô tương ứng trong GDT, LDT chứa mô tả đoạn (a);
  - Phần mô tả đoạn sau đó được kết hợp với độ dịch để tạo ra địa chỉ tuyến tính (b)





Quản lý bộ nhớ trong Intel Pentium (cont.)

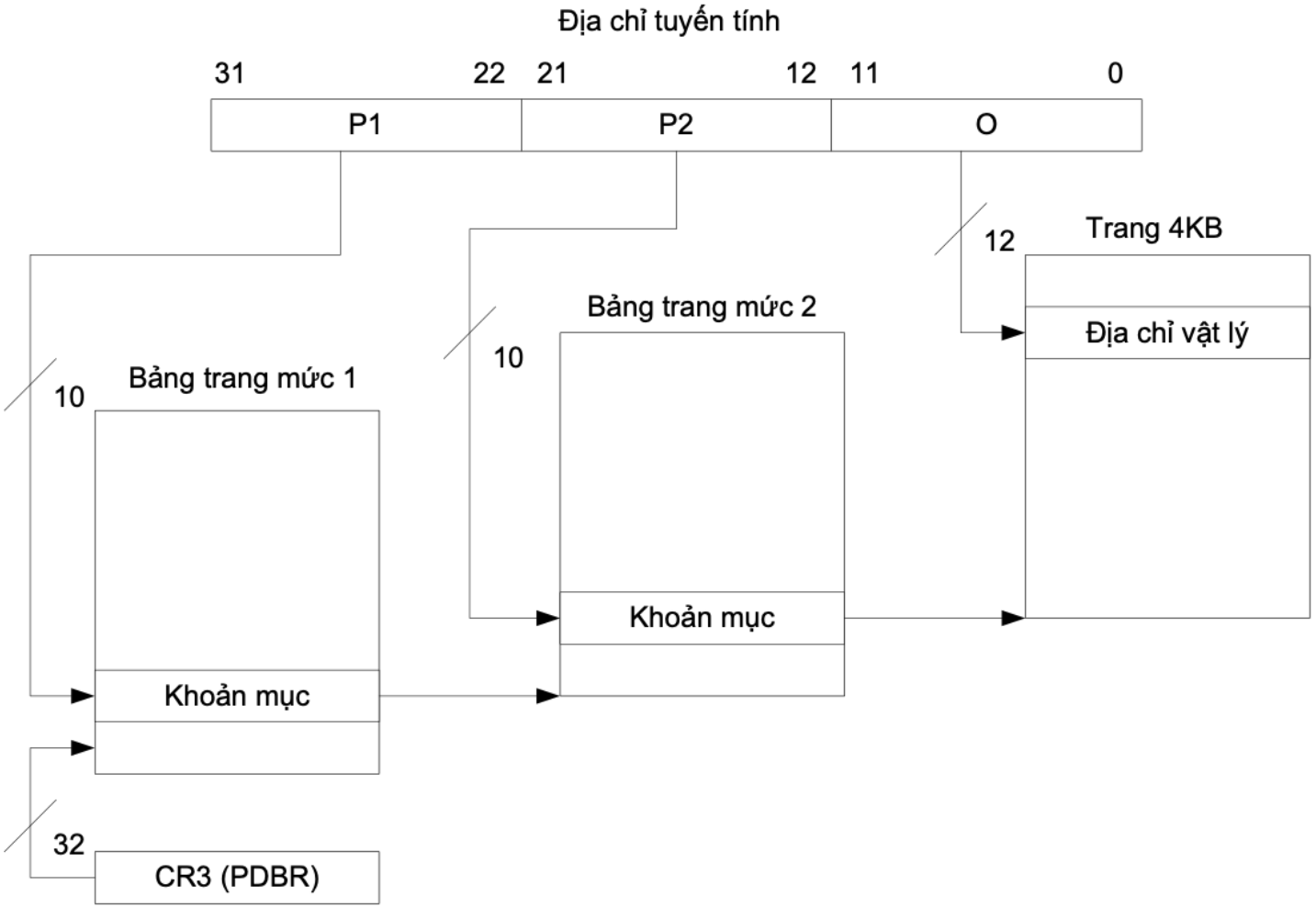
- Hỗ trợ kích thước trang 4KB hoặc 4MB, tùy thuộc vào giá trị cờ kích thước trang
- Trang kích thước 4KB: tổ chức bảng trang thành 2 mức
  - Địa chỉ tuyến tính có kích thước 32 bit



  - P1: cho phép tìm bảng trang mức 1
  - P2: tìm ô tương ứng trong bảng trang mức 2 kết hợp với độ dịch o tạo ra địa chỉ vật lý
- Trang kích thước 4MB: Bảng trang chỉ có một mức
  - P :10bit
  - O: độ dịch, kích thước 22bit cho phép trỏ tới vị trí cụ thể trong trang nhớ 4MB

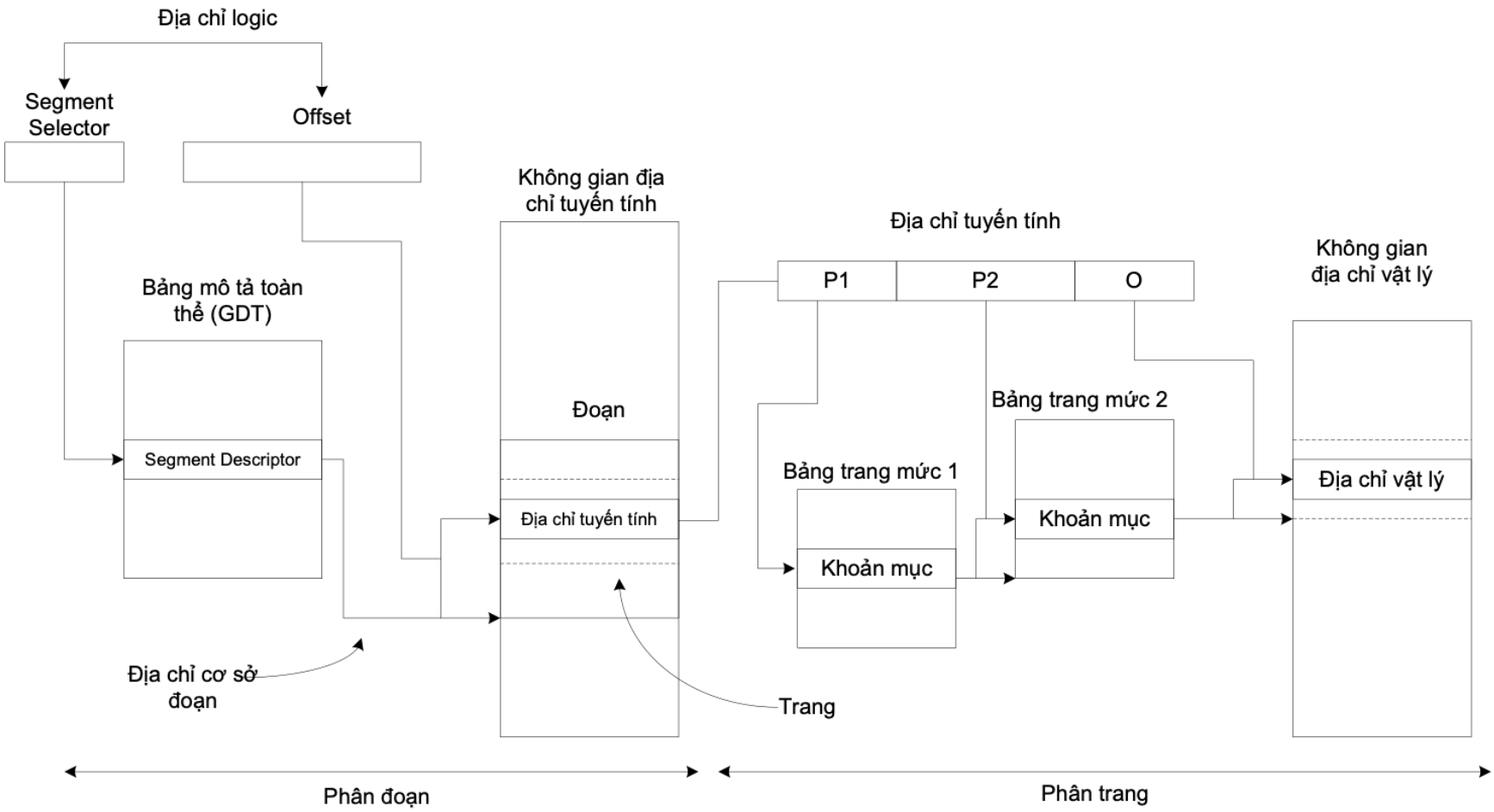
Quản lý bộ nhớ trong Intel Pentium (cont.)

- Biến đổi địa chỉ tuyến tính thành địa chỉ vật lý kích thước trang 4KB



Quản lý bộ nhớ trong Intel Pentium (cont.)

- Biến đổi địa chỉ tuyến tính thành địa chỉ vật lý kích thước trang 4KB



Quản lý bộ nhớ trong Windows XP

- Cho phép tiến trình sử dụng bộ nhớ ảo tới 4GB
  - 2GB được dùng riêng cho tiến trình
  - 2GB sau được dùng chung cho hệ thống
- Bộ nhớ ảo thực hiện bằng kỹ thuật nạp trang theo nhu cầu và đổi trang
  - Kích thước trang nhớ 4KB
  - Tổ chức bảng trang 2 mức
- Nạp trang theo cụm: khi xảy ra thiếu trang, nạp cả cụm gồm 1 số trang nằm sau trang bị thiếu

Quản lý bộ nhớ trong Windows XP (cont.)

- Kiểm soát số lượng trang: gán cho mỗi tiến trình số lượng trang tối đa và tối thiểu
- Số lượng trang tối đa và tối thiểu cấp cho tiến trình được thay đổi tùy vào tình trạng bộ nhớ trống
- HĐH lưu danh sách khung trống, và sử dụng một ngưỡng an toàn
  - Số khung trống ít hơn ngưỡng: HDH xem xét các tiến trình đang thực hiện.
  - Tiến trình có số trang lớn hơn số lượng tối thiểu sẽ bị giảm số trang cho tới khi đạt tới số lượng tối thiểu của mình.
- Tùy vào vi xử lý, Windows XP sử dụng thuật toán đổi trang khác nhau

Tổng kết

Chương 3 Quản lý bộ nhớ

- Phân đoạn bộ nhớ
- Bộ nhớ ảo

Chương 4 Quản lý tiến trình

- Các khái niệm liên quan đến tiến trình
- Luồng (thread)
- Điều độ tiến trình
- Đồng bộ hóa các tiến trình đồng thời
- Tình trạng bế tắc và đói

Câu hỏi và bài tập

1. **Câu 1**: Bộ nhớ có kích thước 1MB. Sử dụng phương pháp kề cận (buddy system) để cấp phát cho các tiến trình lần lượt với kích thước như sau: A: 112KB, B: 200KB, C: 150KB, D: 50KB
2. **Câu 2**: Kích thước khung bộ nhớ là 4096 bytes. Hãy chuyển địa chỉ logic 8207, 4300 sang địa chỉ vật lý biết rằng bảng trang như sau:

Page	Frame
0	3
1	5
2	4
3	30
4	22
5	14

Câu hỏi và bài tập (cont.)

3. **Câu 3**: Không gian địa chỉ logic của tiến trình gồm 11 trang, mỗi trang có kích thước 2048B được ánh xạ vào bộ nhớ vật lý có 20 khung
   - 3.1 Để biểu diễn địa chỉ logic cần tối thiểu bao nhiêu bit
   - 3.2 Để biểu diễn địa chỉ vật lý cần bao nhiêu bit
4. **Câu 4**: Bộ nhớ vật lý có 3 khung. Thứ tự truy cập các trang là 1, 2, 3, 4, 2, 1, 5, 6, 2, 1, 2, 3, 7, 6, 3, 2, 1, 2, 3, 6. Vẽ sơ đồ cấp phát bộ nhớ và Có bao nhiêu sự kiện thiếu trang xảy ra nếu sử dụng:
   - Thuật toán tối ưu
   - FIFO
   - LRU
   - Đồng hồ

Câu hỏi và bài tập (cont.)

5. **Câu 5**: Giả sử tiến trình được cấp 4 khung nhớ vật lý, các trang của tiến trình được truy cập theo thứ tự sau : 1,2,3,4,5,3,4,1,6,7,8,7,8,9,7,8,9,5. Hãy xác định thứ tự nạp và đổi trang nếu sử dụng:
   - Thuật toán tối ưu
   - FIFO
   - LRU
   - Đồng hồ