HỌC VIỆN CÔNG NGHỆ BƯU CHÍNH VIỄN THÔNG


BÀI GIẢNG MÔN

# HỆ ĐIỀU HÀNH

Giảng viên: ThS. Nguyễn Thị Ngọc Vinh

Bộ môn: Khoa học máy tính- Khoa CNTT1

Học kỳ/Năm biên soạn: I/ 2009 - 2010

# CHƯƠNG 3: QUẢN LÝ BỘ NHỚ

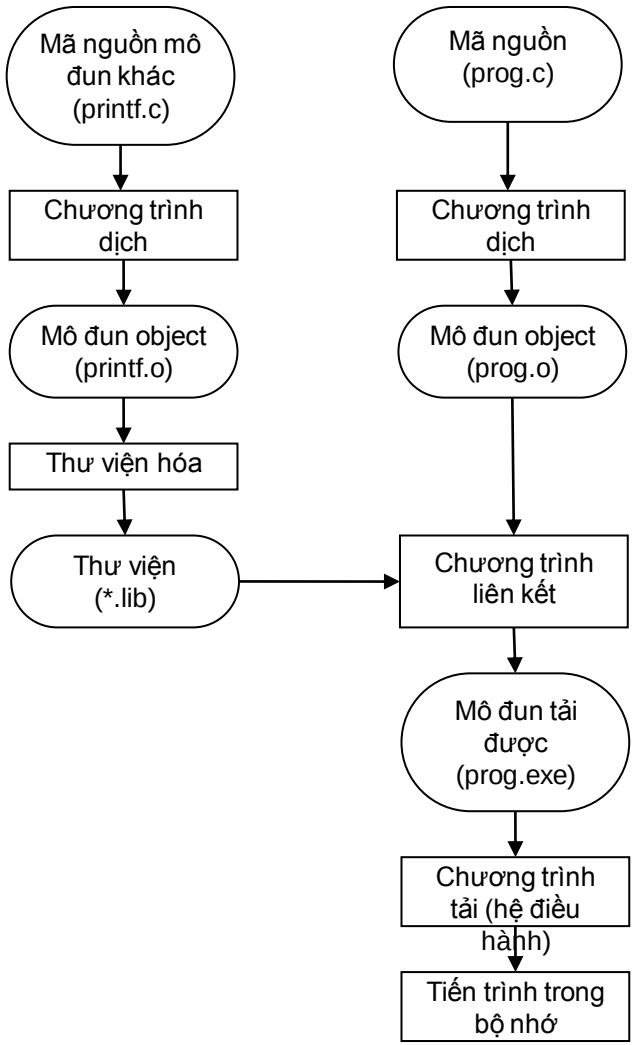
# NỘI DUNG

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

# I. ĐỊA CHỈ VÀ CÁC VẤN ĐỀ LIÊN QUAN

- Vấn đề gán địa chỉ:
  - Khi viết chương trình, sử dụng địa chỉ dưới dạng tên (biến, hàm)
  - Khi dịch, chương trình dịch ánh xạ các tên đó theo địa chỉ tương đối tính từ đầu file obj(biến, hàm)
  - Chương trình liên kết ánh xạ tiếp địa chỉ đó thành địa chỉ tương đối tính từ đầu chương trình
- HDH đọc chương trình vào bộ nhớ để thực hiện; vị trí trong bộ nhớ không biết trước

=> HDH cần có khả năng gán địa chỉ



- Địa chỉ logic:
  - Gán cho các lệnh và dữ liệu không phụ thuộc vào vị trí cụ thể tiến trình trong bộ nhớ
  - Chương trình ứng dụng chỉ nhìn thấy và làm việc với địa chỉ logic này
  - Là địa chỉ tương đối tức là mỗi phần tử của chương trình được gán một địa chỉ tương đối đối với một vị trí nào đó

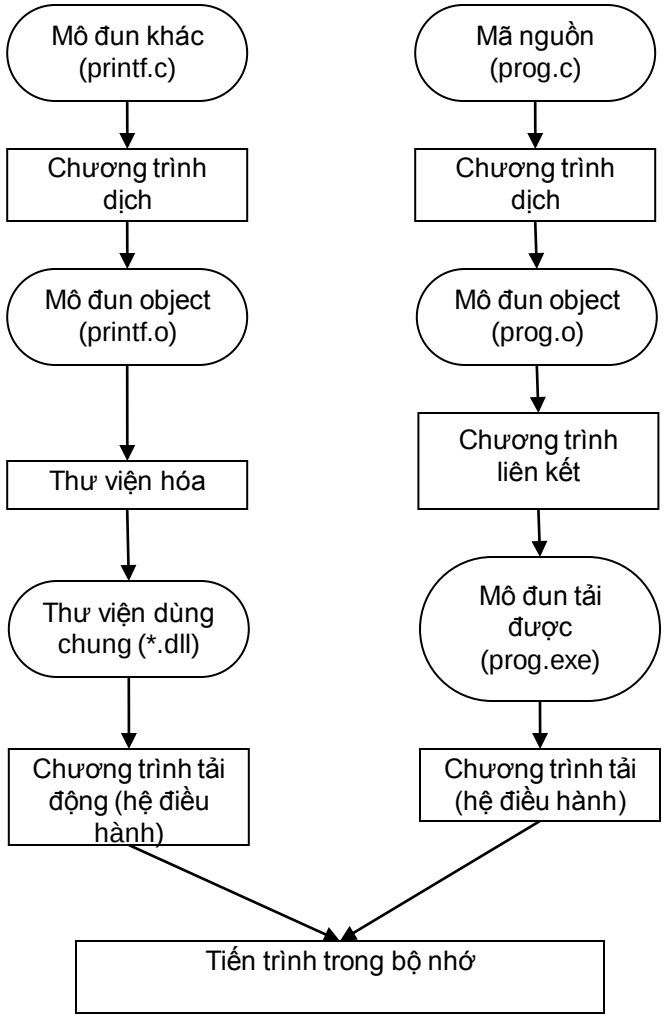
- Địa chỉ vật lý:
  - Là địa chỉ chính xác trong bộ nhớ máy tính
  - Các mạch nhớ sử dụng để truy nhập tới chương trình và dữ liệu
- Địa chỉ logic được chuyển thành địa chỉ vật lý nhờ khối ánh xạ địa chỉ.

## 1. Tải trong quá trình thực hiện

- Hàm chưa bị gọi thì chưa tải vào bộ nhớ
- Chương trình chính được load vào bộ nhớ và chạy
- Khi có lời gọi hàm:
  - Chương trình sẽ kiểm tra hàm đó được tải vào chưa.
  - Nếu chưa, chương trình sẽ tiến hành tải sau đó ánh xạ địa chỉ hàm vào không gian chung của chương trình và thay đổi bảng địa chỉ để ghi lại các ánh xạ đó
- Lập trình viên đảm nhiệm, HDH cung cấp các hàm thư viện cho tải động

## 2. Liên kết động và thư viện dùng chung

- Liên kết tĩnh: các hàm và thư viện được liên kết luôn vào chương trình
- Lãng phí không gian cả trên đĩa và MEM trong
- Trong giai đoạn liên kết, không kết nối các hàm thư viện vào chương trình mà chỉ chèn các thông tin về hàm thư viện đó (stub)



## 2. Liên kết động và thư viện dùng chung

- Các modul thư viện được liên kết trong quá trình thực hiện:
  - Không giữ bản sao các modul thư viện mà tiến trình giữ stub chứa thông tin về modul thư viện
  - Khi stub được gọi, nó kiểm tra modul tương ứng đã có trong bộ nhớ chưa. Nếu chưa, thì tải phần còn lại và dùng.
  - Lần tiếp theo cần sử dụng, modul thư viện sẽ được chạy trực tiếp
  - Mỗi modul thư viện chỉ có 1 bản sao duy nhất chứa trong MEM
- Cần hỗ trợ từ HDH

## 1. Cấp phát lại

- Cần có khả năng tráo đổi các tiến trình vào và ra ngoài MEM để tối đa sử dụng vi xử lý
- Không thể yêu cầu tiến trình được chuyển lại vào MEM thì phải vào đúng chỗ nó đã dùng trước khi bị chuyển ra

## 2. Bảo vệ

- Mỗi tiến trình phải được bảo vệ khỏi các tham chiếu không mong muốn từ các tiến trình khác vào vùng nhớ dành cho mình
- Mọi tham chiếu bộ nhớ của 1 tiến trình phải được kiểm tra lúc chạy
- HDH không đoán trước được mọi tham chiếu MEM => phần cứng VXL đảm nhiệm

## 3. Chia sẻ

- Nhiều tiến trình cần và được phép truy cập vào cùng 1 vùng nhớ
- Các tiến trình đang cộng tác cần chia sẻ truy nhập tới 1 cấu trúc dữ liệu
- => Phải cho phép truy cập tới các vùng chia sẻ

- Cấu trúc logic:
  - MEM được cấu trúc 1 cách tuyến tính gồm các byte, còn chương trình được tổ chức thành các modul
  - Phải đáp ứng để:
    - Các modul có thể được viết và thông dịch 1 cách độc lập
    - Mức độ bảo vệ có thể khác nhau
    - Modul có thể được chia sẻ giữa các tiến trình

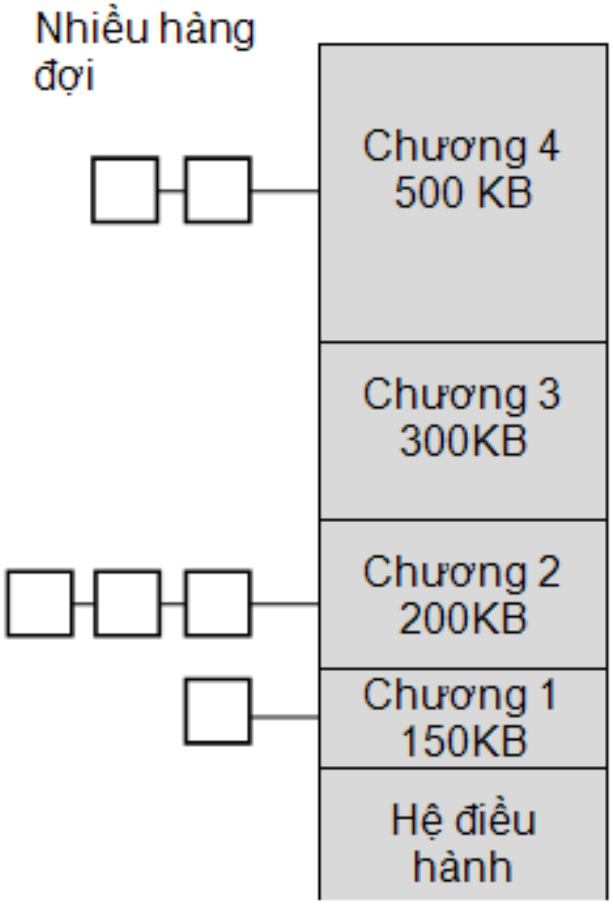
- Cấu trúc vật lý:
  - 2 mức:
    - Bộ nhớ chính: nhanh; chi phí cao, dung lượng ít
    - Bộ nhớ phụ: dung lượng lớn, cho phép lưu chương trình và dữ liệu trong thời gian dài
  - Hệ thống có trách nhiệm chuyển đổi thông tin giữa 2 mức

## 1. Phân chương cố định

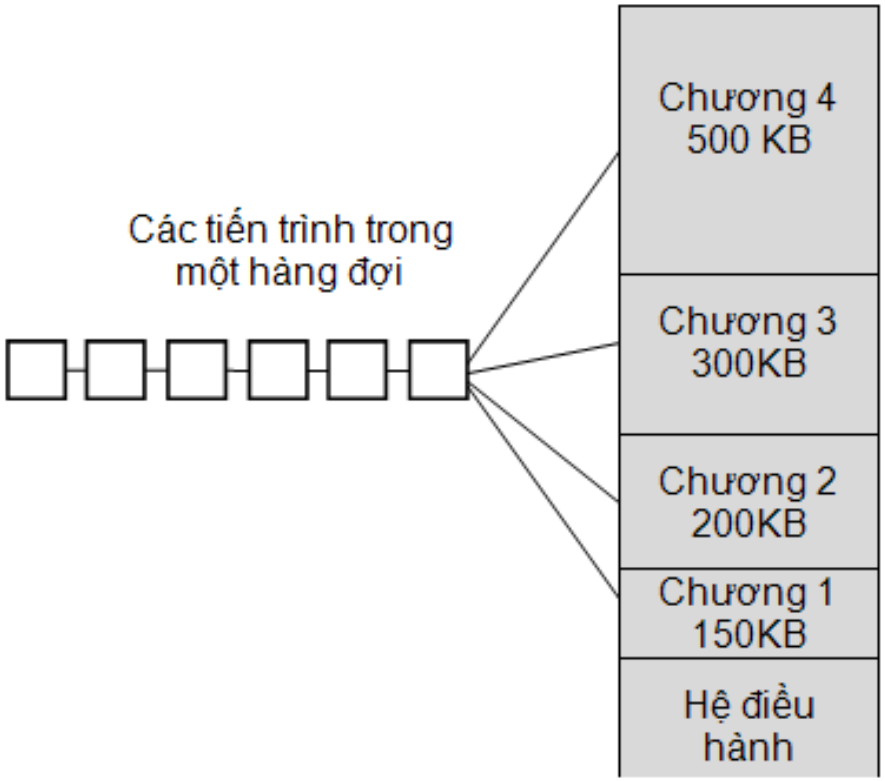
- Chia MEM thành các chương với số lượng nhất định, không thay đổi, gán cho tiến trình 1 chương chứa data, lệnh
- Kích thước các chương bằng nhau:
  - Đơn giản
  - Kích thước chương trình > kích thước chương => không thể cấp phát
  - Gây phân mảnh trong

## 1. Phân chương cố định

- Kích thước các chương khác nhau:
  - Chọn chương có kích thước nhỏ nhất: cần có hàng đợi lệnh cho mỗi chương:
    - Giảm phân mảnh trong, tối ưu cho từng chương
    - Hệ thống không tối ưu



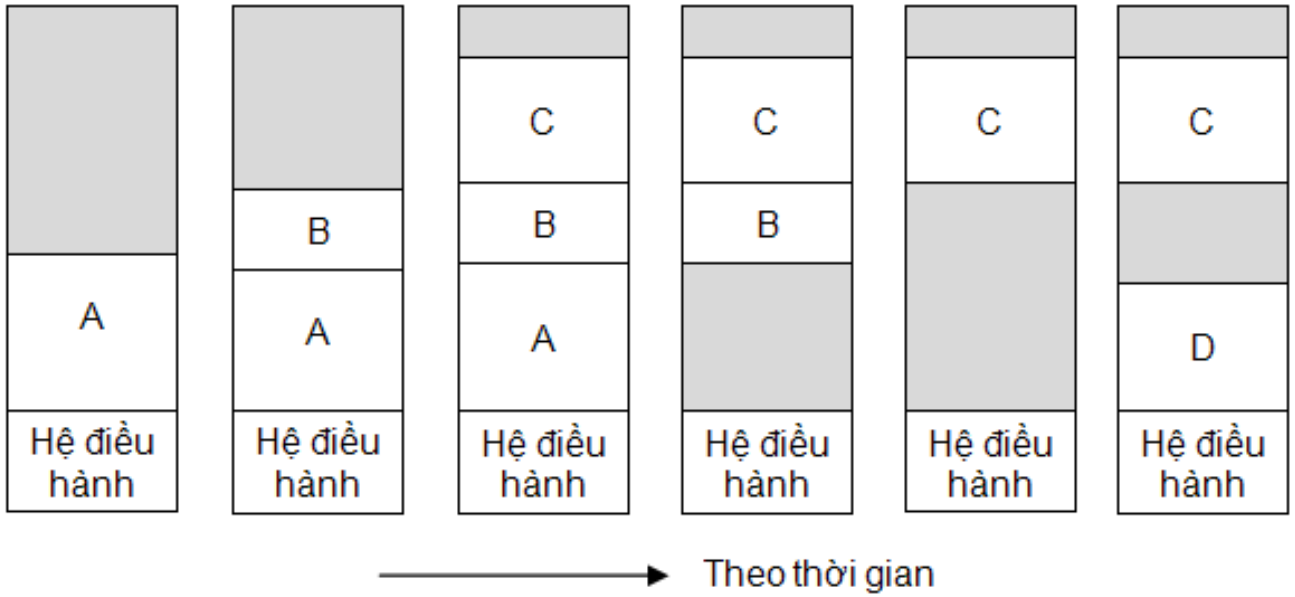
- Kích thước các chương khác nhau:
  - Dùng hàng đợi chung cho mọi chương:
    - Chương sẵn có nhỏ nhất sẽ được cấp phát
    - Khi 1 chương được giải phóng: chọn tiến trình gần đầu hàng độ nhất và có kích thước phù hợp nhất



- Ưu điểm: đơn giản, ít xử lý
- Nhược điểm:
  - Số lượng chương xác định tại thời điểm tạo hệ thống hạn chế số lượng tiến trình hoạt động
  - Kích thước chương thiết lập trước: không hiệu quả

## 2. phân chương động

- Kích thước, số lượng và vị trí chương đều có thể thay đổi
- Khi có yêu cầu, HDH cấp cho tiến trình 1 chương có kích thước đúng bằng tiến trình đó
- Khi tiến trình kết thúc sẽ tạo vùng trống trong MEM
- Các vùng trống nằm cạnh nhau được nhập lại thành vùng lớn hơn



## 2. Phân chương động

- Tránh phân mảnh trong
- Gây phân mảnh ngoài: dồn những vùng trống nhỏ thành lớn (nén)
- Sử dụng các chiến lược cấp chương
  - Chọn vùng thích hợp đầu tiên
  - Vùng thích hợp nhất
  - Vùng không thích hợp nhất

## 3. Phương pháp kề cận (buddy system)

- Các chương và khối trống có kích thước là lũy thừa của $2^k$ ($L \le k \le H$): $2^L$: kích thước nhỏ nhất của chương; $2^H$ : kích thước MEM
- Đầu tiên, toàn bộ không gian nhớ là $2^H$ , yêu cầu cấp vùng nhớ S
  - $2^{H-1} < S \le 2^H$ : cấp cả $2^H$
  - Chia đôi thành 2 vùng $2^{H-1}$ :
    - Nếu $2^{H-2} < S \le 2^{H-1}$ : cấp $2^{H-1}$
    - Tiếp tục chia đôi tới khi tìm được vùng thỏa mãn $2^{k-1} < S \le 2^k$

## 3. Phương pháp kề cận

- Sau một thời gian xuất hiện các khối trống có kích thước $2^k$
- Tạo danh sách móc nối các vùng có cùng kích thước
- Nếu có 2 khối trống cùng kích thước và kề nhau thì ghép lại thành 1
- Khi cần cấp, sẽ tìm trong danh sách khối phù hợp nhất; nếu không tìm khối lớn hơn và cắt đôi

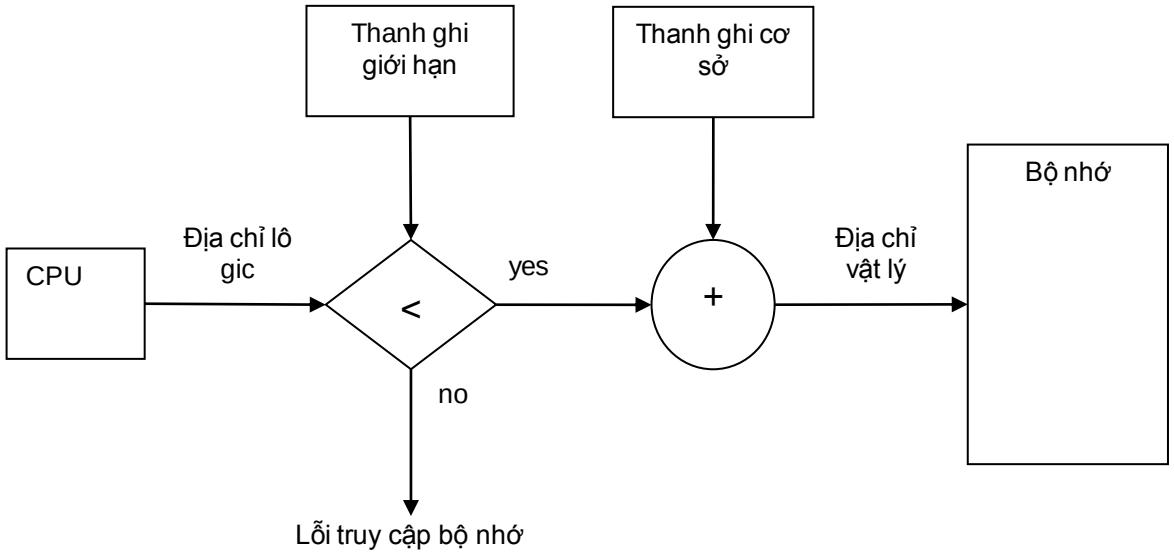
## 3. Phương pháp kề cận

Memory

| | 0 | 128 K | 256 K | 384 K | 512K | 640 K | 768 K | 896 K | 1M |
|---|---|---|---|---|---|---|---|---|---|
| **Initially** | | | | | | | | | **1 Hole** |
| request 70 | **A** | **128** | | | **256** | | | | **3** |
| request 35 | **A** | **B** | **64** | | **256** | | | | **3** |
| request 80 | **A** | **B** | **64** | **C** | **128** | | | | **3** |
| return A | **128** | **B** | **64** | **C** | **128** | | | | **4** |
| request 60 | **128** | **B** | **D** | **C** | **128** | | | | **4** |
| return B | **128** | **64** | **D** | **C** | **128** | | | | **4** |
| return D | **256** | | | **C** | **128** | | | | **3** |
| return C | **1024** | | | | | | | | **1** |

## 4. Ánh xạ địa chỉ và chống truy cập trái phép

- Vị trí các chương thường không biết trước và có thể thay đổi => cần có cơ chế biến đổi địa chỉ logic thành vật lý
- Cấm truy cập trái phép: tiến trình này truy cập tới phần MEM của tiến trình khác
- Ánh xạ địa chỉ do phần cứng đảm nhiệm



- Khi tiến trình được tải vào MEM, CPU dành 2 thanh ghi:
  - Thanh ghi cơ sở: chứa địa chỉ bắt đầu của tiến trình
  - Thanh ghi giới hạn: chứa độ dài chương
- Địa chỉ logic được so sánh với nội dung của thanh ghi giới hạn
  - Nếu lớn hơn: lỗi truy cập
  - Nhỏ hơn: được đưa tới bộ cộng với thanh ghi cơ sở để thành địa chỉ vật lý
- Nếu chương bị di chuyển thì nội dung của thanh ghi cơ sở bị thay đổi chứa địa chỉ vị trí mới

## 5. Trao đổi giữa bộ nhớ và đĩa (swapping)

- Các tiến trình đang thực hiện có thể bị tạm thời tải ra đĩa nhường chỗ để tải các tiến trình khác vào
- Sau đó lại được tải vào (nếu chưa kết thúc) để thực hiện tiếp
- Xảy ra khi:
  - Một tiến trình đã hết khoảng thời gian sử dụng CPU của mình
  - Nhường chỗ cho một tiến trình khác có thứ tự ưu tiên cao hơn

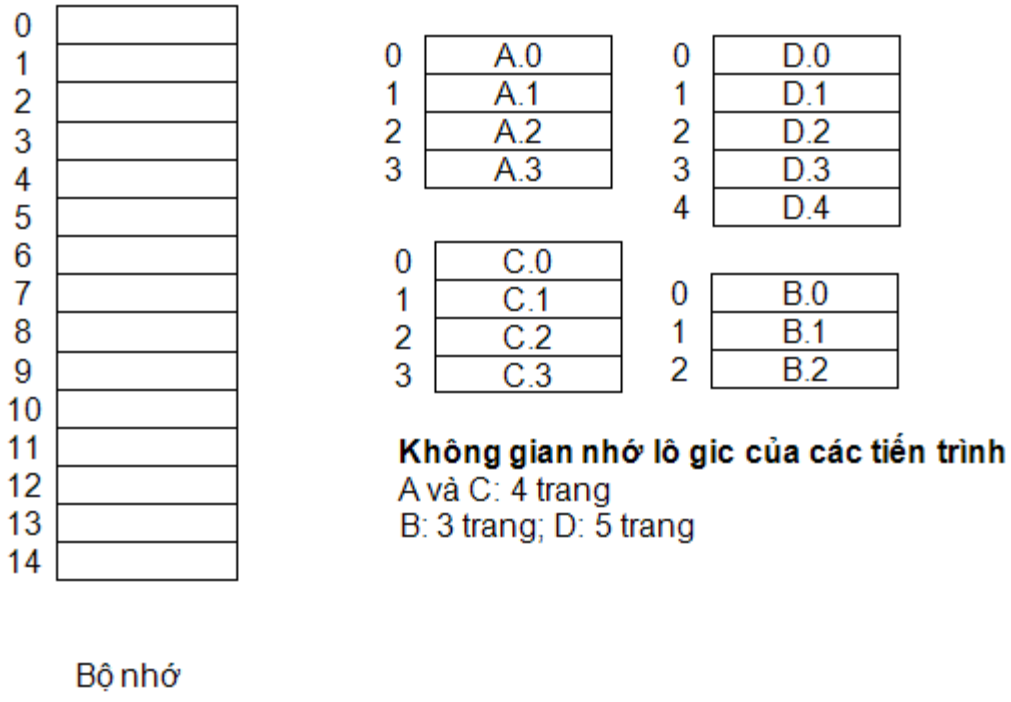
## 5. Trao đổi giữa bộ nhớ và đĩa (swapping)

- Thời gian tải phụ thuộc vào tốc độ truy cập đĩa, tốc độ truy cập bộ nhớ và kích thước tiến trình
- Khi được tải vào lại, tiến trình có thể được chứa vào chương ở vị trí cũ hoặc được cấp cho một chương địa chỉ hoàn toàn mới
- Các tiến trình bị trao đổi phải ở trạng thái nghỉ, đặc biệt không thực hiện các thao tác vào ra

# V. PHÂN TRANG BỘ NHỚ

## 1. Khái niệm phân trang

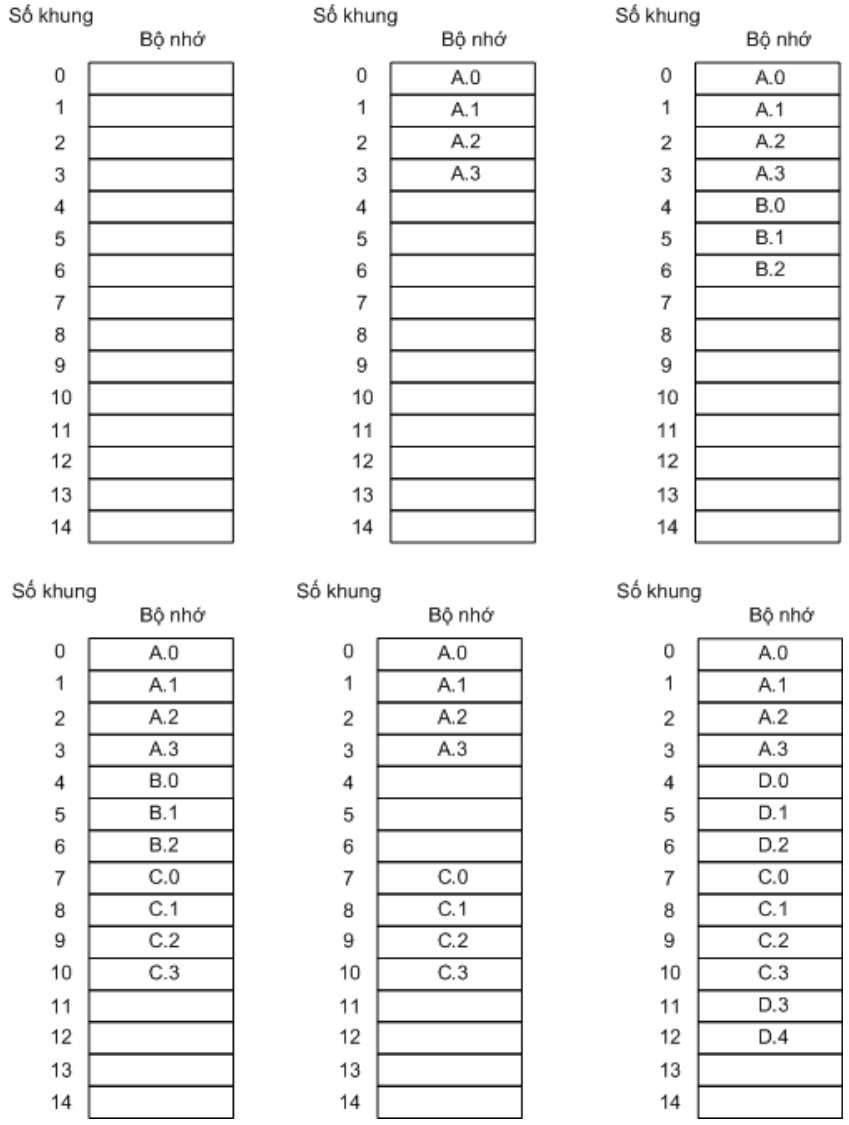
- Bộ nhớ vật lý được chia thành các khối nhỏ, kích thước cố định và bằng nhau gọi là khung trang (page frame)
- Không gian địa chỉ logic của tiến trình được chia thành những khối gọi là trang (page), có kích thước bằng khung



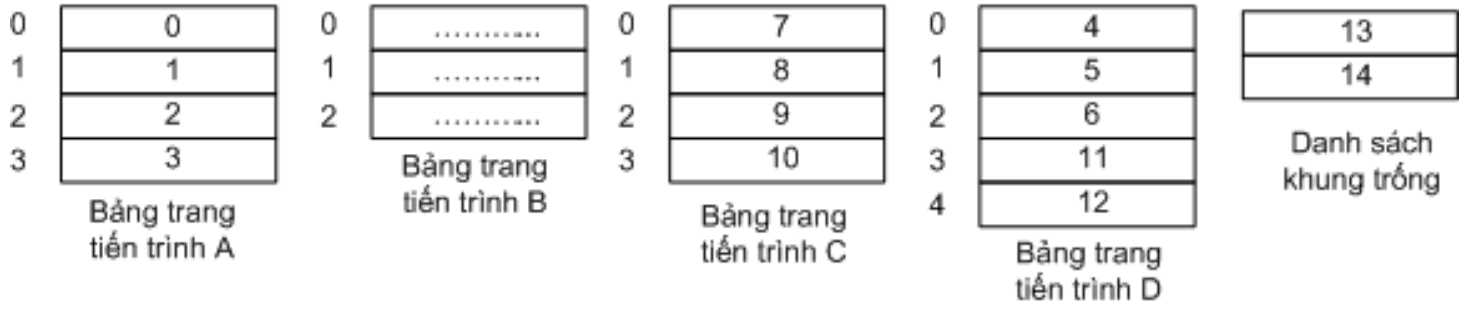
# V. PHÂN TRANG BỘ NHỚ

## 1. Khái niệm phân trang

- Tiến trình được cấp các khung để chứa các trang của mình.
- Các trang có thể chứa trong các khung nằm rải rác trong bộ nhớ



- HDH quản lý việc cấp phát khung cho mỗi tiến trình bằng bảng trang (bảng phân trang): mỗi ô tương ứng với 1 trang và chứa số khung cấp cho trang đó
- Mỗi tiến trình có bảng trang riêng
- Duy trì danh sách các khung trống trong MEM



## 1. Khái niệm phân trang

- Tương tự như phân chương cố định: khung tương tự chương, kích thước và vị trí không thay đổi
- Tuy nhiên kích thước các phần tương đối nhỏ và các phần cho 1 tiến trình không cần liên tục nhau
- Không có phân mảnh ngoài
- Có phân mảnh trong

## 2. Ánh xạ địa chỉ

- Để tính toán địa chỉ hiệu quả, kích thước khung được chọn là lũy thừa của 2
- Địa chỉ logic gồm 2 phần:
  - Số thứ tự trang (p)
  - Độ dịch (địa chỉ lệch) của địa chỉ so với đầu trang (o)
- Nếu kích thước trang là $2^n$. Biểu diễn địa chỉ logic dưới dạng địa chỉ có độ dài (m + n) bit
  - m bit cao: biểu diễn số thứ tự trang
  - n bit thấp: biểu diễn độ dịch trong trang nhớ

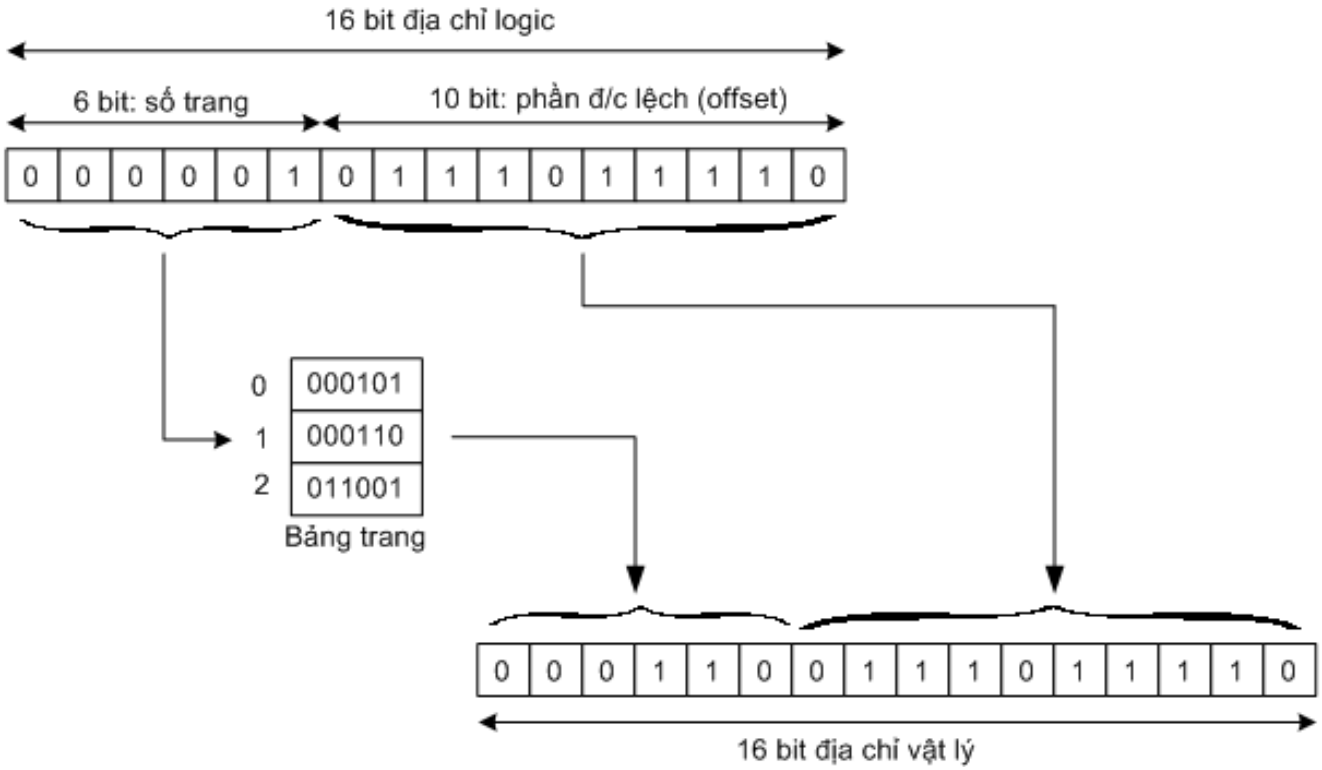
| Địa chỉ lô gic | số thứ tự trang (*p*) | độ dịch trong trang (*0*) |
|---|---|---|
| Độ dài | *m* | *n* |

- Quá trình chuyển địa chỉ logic sang địa chỉ vật lý:
  - Lấy m bit cao của địa chỉ => được số thứ tự trang
  - Dựa vào bảng trang, tìm được số thứ tự khung vật lý (k)
  - Địa chỉ vật lý bắt đầu của khung là $k*2^n$
  - Địa chỉ vật lý của byte được tham chiếu là địa chỉ bắt đầu của khung cộng với địa chỉ lệch (độ dịch)

  => Chỉ cần thêm số khung vào trước dãy bit biểu diễn độ lệch

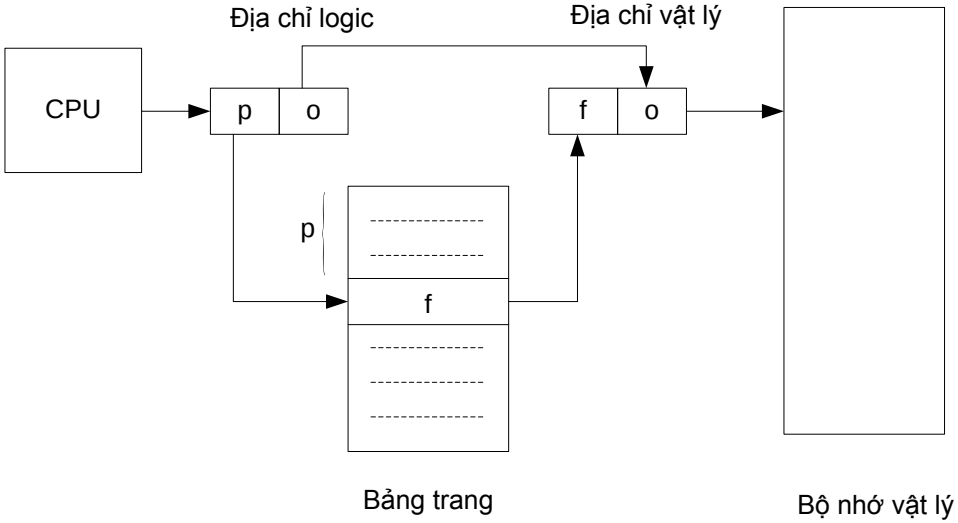
## 2. Ánh xạ địa chỉ

- Kích thước khung là 1KB
- Địa chỉ logic được biểu diễn bằng 16 bit
- => Sử dụng 10 bit để biểu diễn địa chỉ lệch (n=10)
- 6 bit biểu diễn STT trang/ khung
- Địa chỉ logic 1502
- ↔ byte 478 trong trang 1



## 2. Ánh xạ địa chỉ

- Quá trình biến đổi từ địa chỉ logic sang địa chỉ vật lý được thực hiện bằng phần cứng
- Kích thước trang  là lũy thừa của 2, nằm trong khoảng từ 512B đến 16MB
- Việc tách phần p và o trong địa chỉ logic được thực hiện dễ dàng bằng phép dịch bit



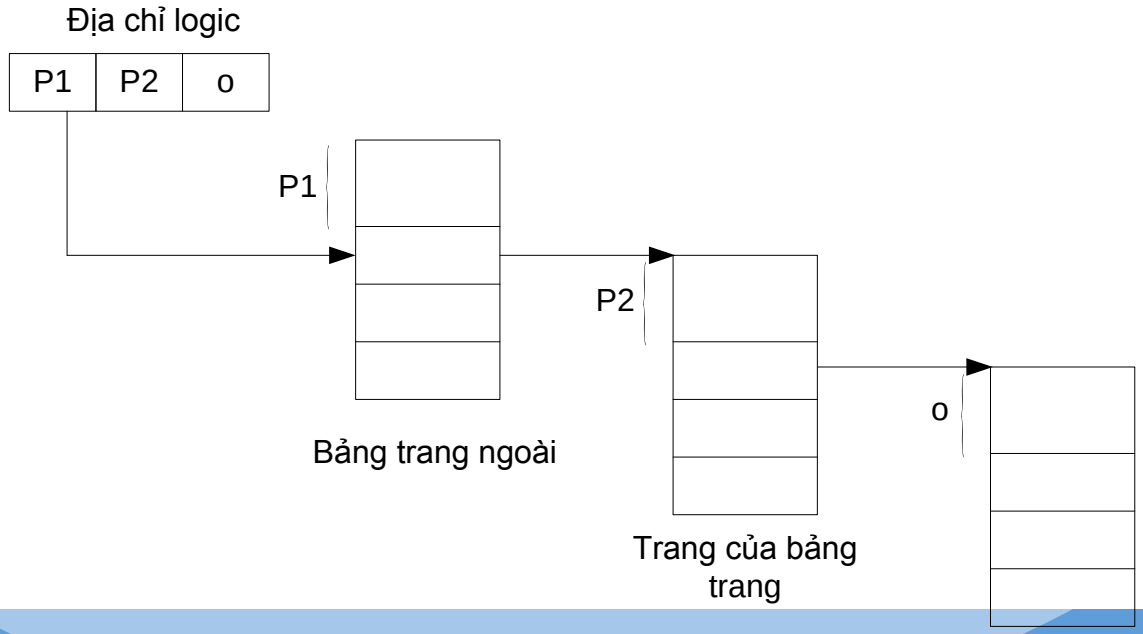
- Phân mảnh trong khi phân trang có giá trị trung bình bằng nửa trang
- => giảm kích thước trang cho phép tiết kiệm MEM
- Kích thước trang nhỏ => số lượng trang tăng => bảng trang to, khó quản lý
- Kích thước trang nhỏ: không tiện cho việc trao đổi với đĩa
- Windows 32bit: kích thước trang 4KB
- Cơ chế ánh xạ giữa hai loại địa chỉ hoàn toàn trong suốt đối với chương trình

## 3. Tổ chức bảng trang

- Mỗi thao tác truy cập bộ nhớ đều đòi hỏi truy cập bảng phân trang
- => tổ chức bảng phân trang sao cho tốc độ truy cập là cao nhất
- Sử dụng tập hợp các thanh ghi làm bảng phân trang:
  - Tốc độ truy cập rất cao
  - Số lượng thanh ghi hạn chế => không áp dụng được
- Giữ các bảng trang trong MEM:
  - Vị trí mỗi bảng được trỏ bởi thanh ghi cơ sở bảng trang PTBR (Page Table Base Register)
  - Nhiều thời gian để truy cập bảng
  - => sử dụng bộ nhớ cache tốc độ cao

- Không gian địa chỉ logic lớn ($2^{32}$ -> $2^{64}$) => kích thước bảng trang tăng
- Giả sử không gian địa chỉ logic là $2^{32}$, kích thước trang là 4KB = $2^{12}$
- => số lượng khoản mục cần có trong bảng trang là $2^{20}$
- Mỗi khoản mục có kích thước 4B
- => kích thước bảng trang là 4MB
- => cần chia bảng trang thành những phần nhỏ hơn
- Tổ chức bảng trang nhiều mức: Khoản mục của bảng mức trên chỉ tới bảng trang khác

- Ví dụ bảng 2 mức: địa chỉ 32 bit chia thành 3 phần
  - P1: 10 bit cho phép định vị khoản mục trong bảng mức trên => tìm được bảng mức dưới tương ứng
  - P2: định vị khoản mục trong bảng mức dưới (chứa địa chỉ khung tương ứng)
  - O: 12 bit, chứa độ dịch trong trang



# VI. PHÂN ĐOẠN BỘ NHỚ

## 1. Khái niệm

- Chương trình thường được chia thành nhiều phần: dữ liệu, lệnh, ngăn xếp
- Chia chương trình thành các đoạn theo cấu trúc logic
- Mỗi đoạn được phân vào 1 vùng nhớ, có kích thước không bằng nhau
- Mỗi đoạn tương ứng với không gian địa chỉ riêng, được phân biệt bởi tên (STT) và độ dài của mình
- Các vùng nhớ thuộc các đoạn khác nhau có thể nằm ở vị trí khác nhau

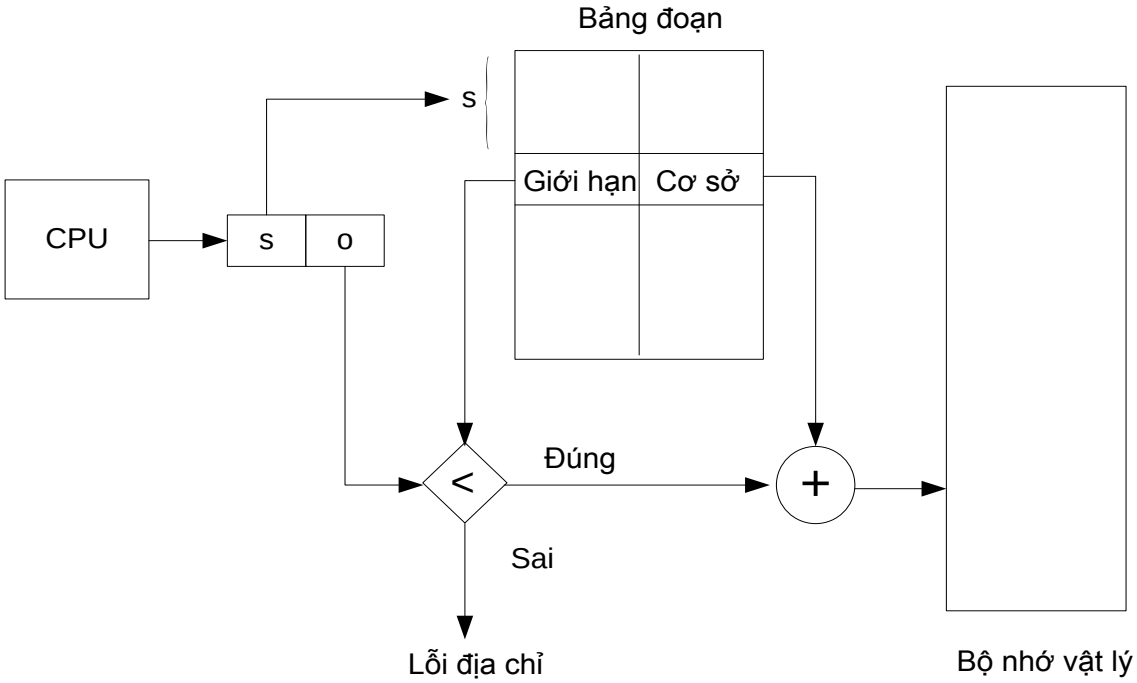
# VI. PHÂN ĐOẠN BỘ NHỚ

## 1. Khái niệm

- Giống phân chương động: bộ nhớ được cấp phát theo từng vùng kích thước thay đổi
- Khác phân chương động: chương trình có thể chiếm nhiều hơn 1 đoạn và không cần liên tiếp nhau trong MEM
- Tránh hiện tượng phân mảnh trong
- Có phân mảnh ngoài
- Dễ sắp xếp bộ nhớ
- Dễ chia sẻ các đoạn giữa các tiến trình khác nhau
- Kích thước mỗi đoạn có thể thay đổi mà không ảnh hưởng tới các đoạn khác

- Sử dụng bảng đoạn cho mỗi tiến trình. Mỗi ô tương ứng với 1 đoạn, chứa:
  - Địa chỉ cơ sở: vị trí bắt đầu của đoạn trong bộ nhớ
  - Địa chỉ giới hạn: độ dài đoạn, sử dụng để chống truy cập trái phép ra ngoài đoạn
- Địa chỉ logic gồm 2 thành phần, (s, o):
  - S: số thứ tự/ tên đoạn
  - O: độ dịch trong đoạn

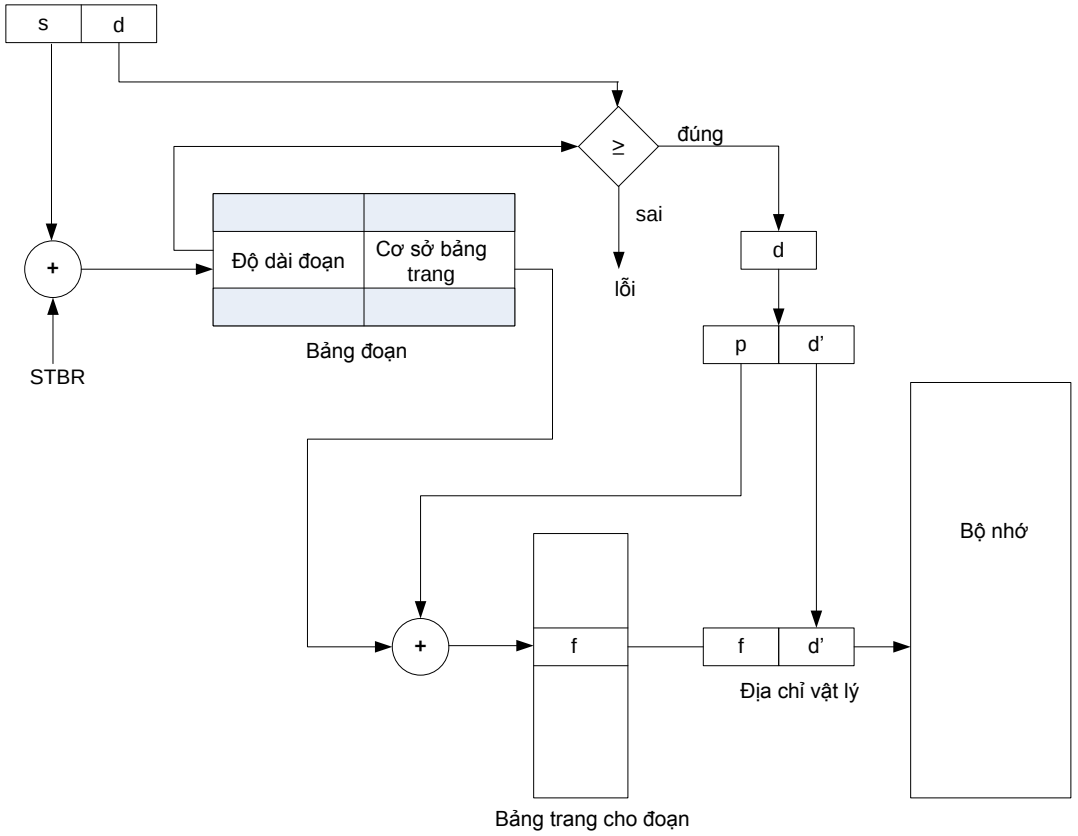
## 2. Ánh xạ địa chỉ




- 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 => địa chỉ sai
- Địa chỉ vật lý mong muốn là tổng của địa chỉ vật lý bắt đầu đoạn và địa chỉ lệch

## 3. Kết hợp phân trang và Phân đoạn

- 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



# V. BỘ NHỚ ẢO
## 1. Khái niệm

- Tiến trình có thể chia thành các phần nhỏ nằm rải rác trong bộ nhớ
- Tất cả các phép biến đổi là trong suốt với người dùng và người lập trình chỉ làm việc với không gian nhớ logic
- Không phải tiến trình nào khi chạy cũng sử dụng tất cả các lệnh và dữ liệu của mình 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

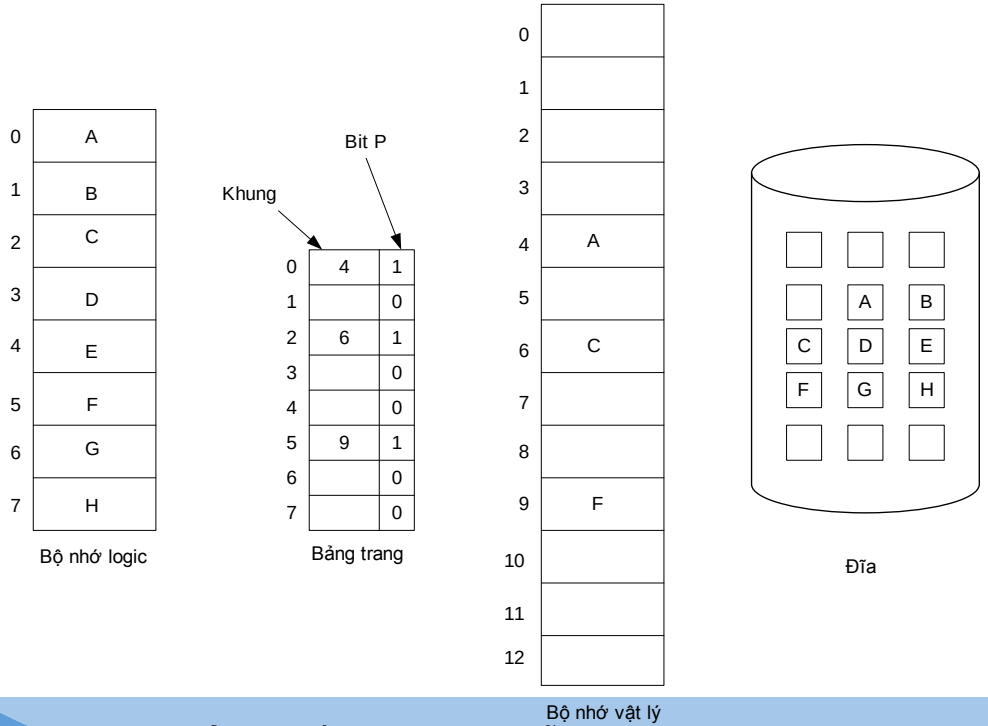
# V. BỘ NHỚ ẢO
## 1. Khái niệm

- Việc thực hiện các tiến trình chỉ nằm một phần trong bộ nhớ có một số ưu điểm:
  - Có thể viết chương trình có kích thước lớn hơn kích thước thực của MEM
  - Cùng 1 lúc nhiều tiến trình cùng được tải vào MEM hơn
- => Bộ nhớ ảo là bộ nhớ lôgic theo cách nhìn của người lập trình và tiến trình và không bị hạn chế bởi bộ nhớ thực.
  - **Bộ nhớ ảo có thể lớn hơn bộ nhớ thực rất nhiều và bao gồm cả không gian trên đĩa**
  - Bộ nhớ ảo thường được xây dựng dựa trên phương pháp phân trang trong đó các trang là đơn vị để nạp từ đĩa vào khi cần

# V. BỘ NHỚ ẢO

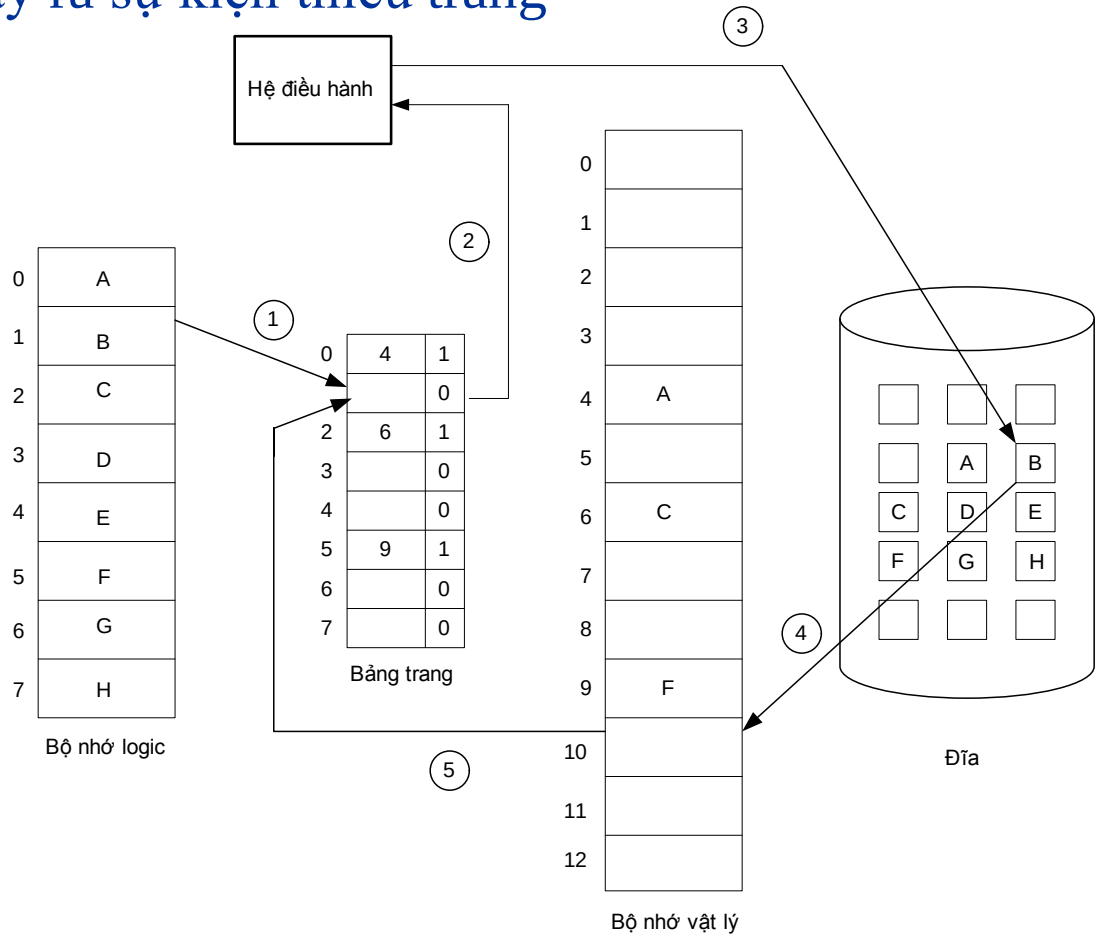
## 2. Nạp trang theo nhu cầu

- Tiến trình được phân trang và chứa trên đĩa
- Khi cần thực hiện, nạp tiến trình vào MEM: chỉ nạp những trang cần dùng
- 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)



## 2. Nạp trang theo nhu cầu

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



# V. BỘ NHỚ ẢO

## 2. Nạp trang theo nhu cầu

- 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 HDH 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

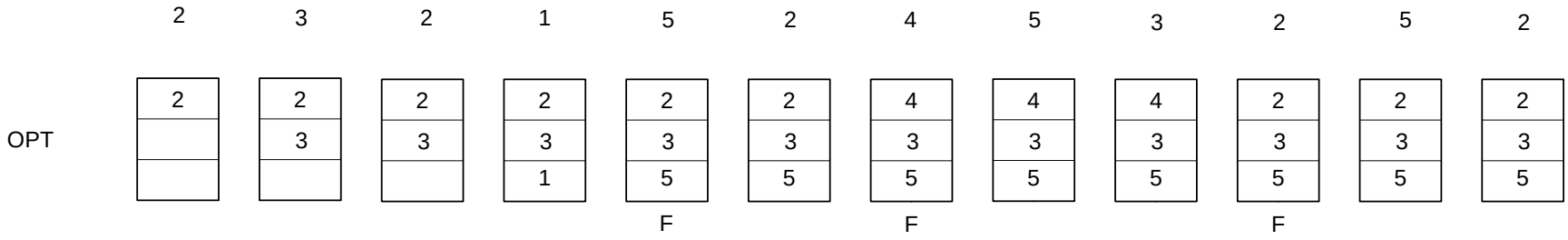
- Bộ nhớ ảo > bộ nhớ thực và chế độ đa chương trình -> có lúc không còn khung nào trống để nạp trang mới

- Giải pháp:
  - Kết thúc tiến trình
  - Trao đổi tiến trình ra đĩa và chờ thời điểm thuận lợi hơn
  - Đổi trang

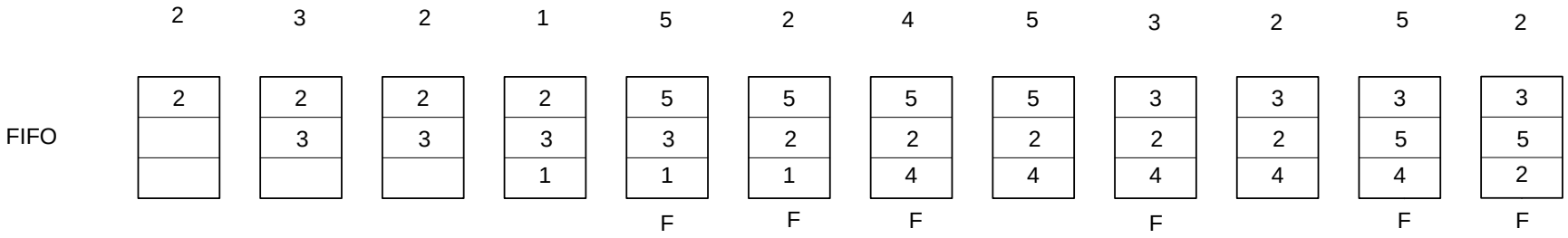
- Nếu không còn khung nào trống, HDH chọn 1 khung đã cấp phát nhưng hiện không dùng tới và giải phóng nó
- Quá trình đổi trang:
  - B1: Xác định trang cần nạp vào trên đĩa
  - B2: Nếu có khung trống thì chuyển sang B4
  - B3:
    - Lựa chọn 1 khung để giải phóng, theo 1 thuật toán nào đó
    - Ghi nội dung khung bị đổi ra đĩa (nếu cần), cập nhật bảng trang và bảng khung
  - B4: Đọc trang cần nạp vào khung vừa giải phóng; cập nhật bảng trang và bảng khung
  - B5: Thực hiện tiếp tiến trình từ điểm bị dừng trước khi đổi trang

- Đổi trang có ghi và đổi trang không ghi:
  - Việc ghi trang bị đổi ra đĩa làm tăng đáng kể thời gian nạp trang
  - => 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
  - Một số khung sẽ không bị giải phóng trong quá trình tìm kiếm khung để đổi trang => các khung bị khóa
  - VD: Khung chứa tiến trình nhân của HDH, chứa các cấu trúc thông tin điều khiển quan trọng
  - Nhận biết bởi 1 bit riêng chứa trong khoản mục

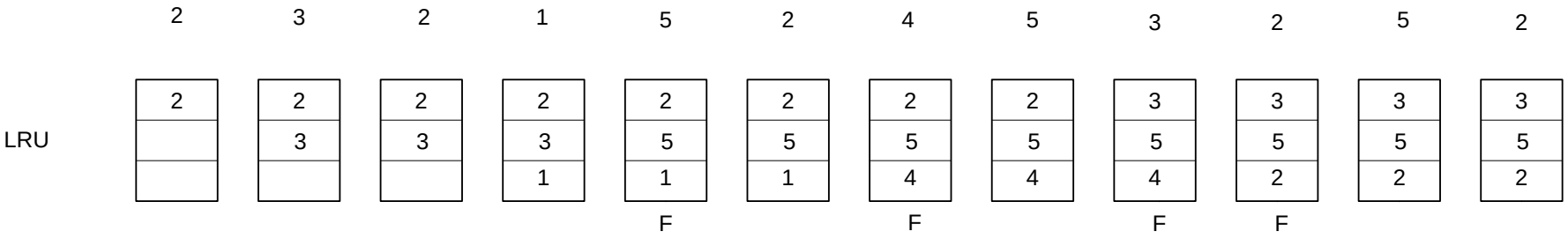
- Đổi trang tối ưu (OPT):
  - Chọn trang sẽ không được dùng tới trong khoảng thời gian lâu nhất để đổi
  - Cho phép giảm tối thiểu sự kiện thiếu trang và do đó là tối ưu theo tiêu chuẩn này
  - HDH không đoán trước được nhu cầu sử dụng các trang trong tương lai
  - => không áp dụng trong thực tế mà chỉ để so sánh với các chiến lược khác



- Vào trước ra trước (FIFO):
  - Trang nào được nạp vào trước thì bị đổi ra trước
  - Đơn giản nhất
  - Trang bị trao đổi là trang nằm lâu nhất trong bộ nhớ



- Đổi trang ít sử dụng nhất trong thời gian cuối (LRU):
  - Trang bị đổi là trang mà thời gian từ lần truy cập cuối cùng đến thời điểm hiện tại là lâu nhất
  - 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



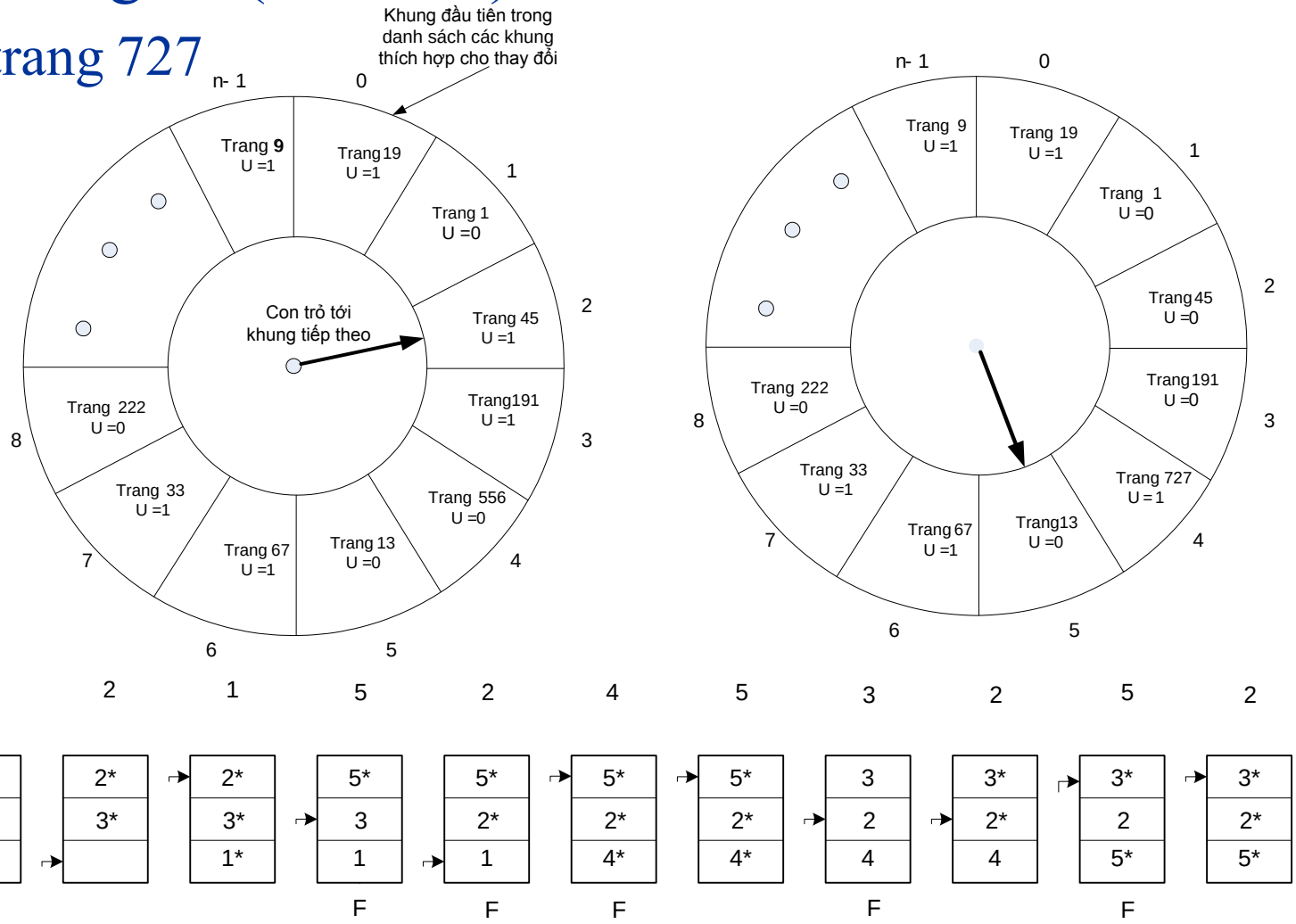
- Đổi trang ít sử dụng nhất trong thời gian cuối (LRU):
  - Xác định được trang có lần truy cập cuối diễn ra cách thời điểm hiện tại lâu nhất?
    - Sử dụng biến đếm:
      - Mỗi khoản mục của bảng phân trang sẽ có thêm một trường chứa thời gian truy cập trang lần cuối - Là thời gian logic
      - CPU 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 ít sử dụng nhất trong thời gian cuối (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

- Thuật toán đồng hồ (CLOCK):
  - Cải tiến FIFO nhằm tránh thay những trang mặc dù đã được nạp vào lâu nhưng 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 có thể bị đổi (các trang tương ứng) 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

- Thuật toán đồng hồ (CLOCK):
  - 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
  - Nếu U của tất cả các trang trong danh sách =1 thì con trỏ sẽ quay đúng 1 vòng, đặt U của tất cả các trang =0 và trang hiện thời đang bị trỏ sẽ bị đổi

- Thuật toán đồng hồ (CLOCK):
  - Cần nạp trang 727



- Thuật toán đồng hồ (CLOCK):
  - 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 cơ hội thứ 2

- 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

- HDH 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 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

# VI. CẤP PHÁT KHUNG TRANG

## 1. Giới hạn khung

- HDH 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 giảm 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

# VI. CẤP PHÁT KHUNG TRANG

## 1.1. 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

# VI. CẤP PHÁT KHUNG TRANG

## 1.2. 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

## 2. Phạm vi cấp phát khung

- 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

# VII. TÌNH TRẠNG TRÌ TRỆ (thrashing)

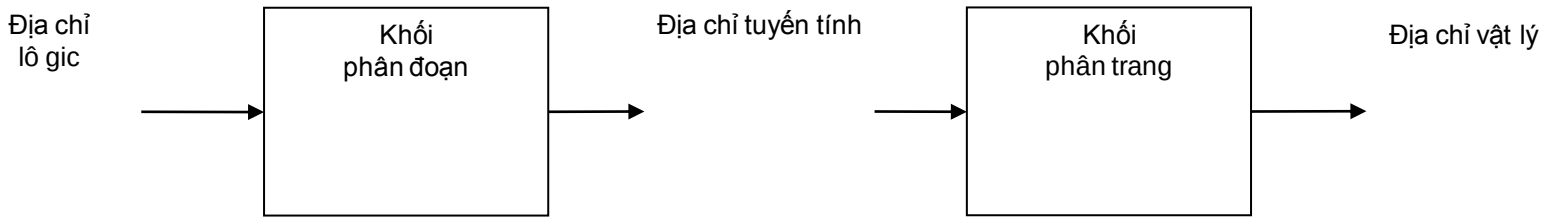
- Là tình trạng đổi trang liên tục do không đủ bộ nhớ
- Thời gian đổi trang của tiến trình lớn hơn thời gian thực hiện
- Xảy ra khi:
  - Kích thước bộ nhớ hạn chế
  - Tiến trình đòi hỏi truy cập đồng thời nhiều trang nhớ
  - Hệ thống có mức độ đa chương trình cao

# VII. TÌNH TRẠNG TRÌ TRỆ (thrashing)
## Kiểm soát tần suất thiếu trang

- 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
- 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

# VIII. QUẢN LÝ BỘ NHỚ TRONG INTEL PENTIUM

- 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

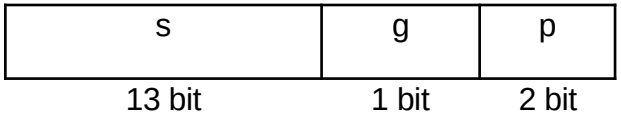


## Phân đoạn

- 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 HDH, và cũng gồm tối đa 8KB đoạn
- LDT(Local Descriptor Table) & 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

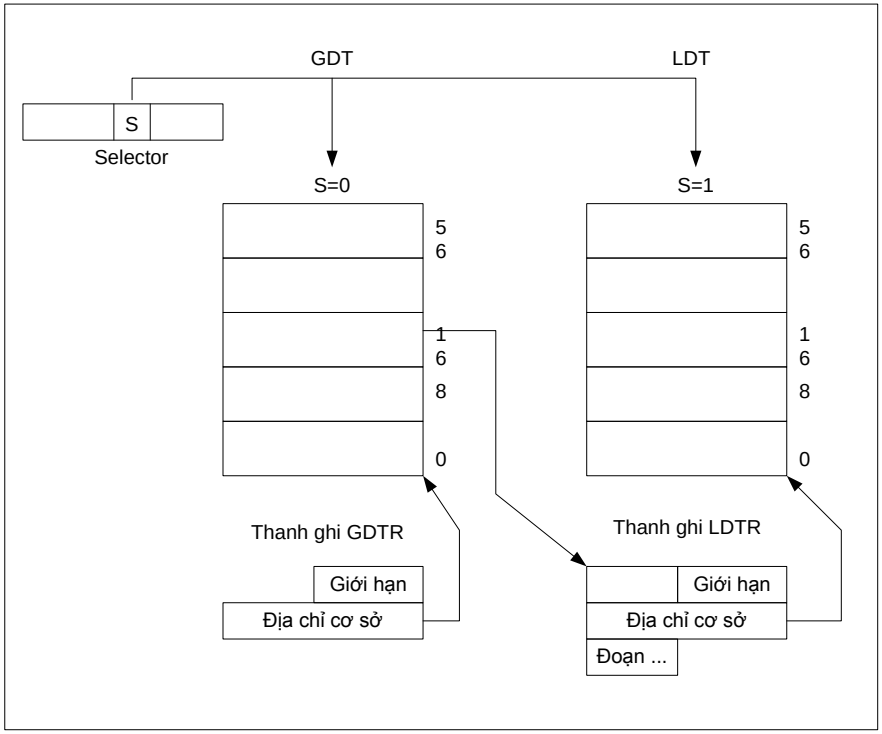
## Phân đoạn

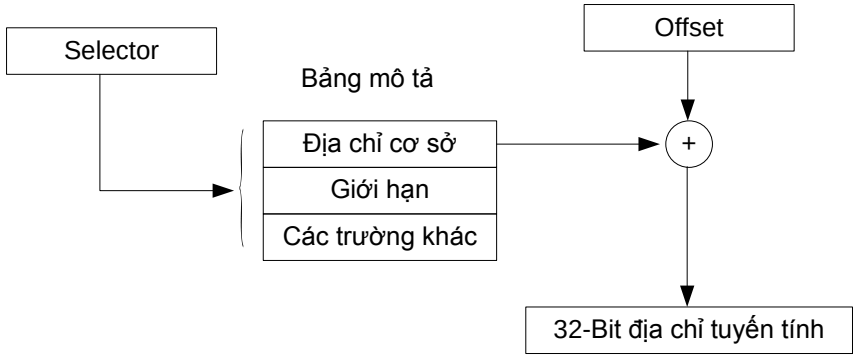
- Có 6 thanh ghi đoạn: cho phép tiến trình truy cập đồng thời 6 đoạn
- Thông tin về đoạn được chứa trong 6 thanh ghi 8 byte
- Địa chỉ logic gồm (selector, offset):
  - Selector: chọn ô tương ứng từ hai bảng mô tả LDT, GDT




    - 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

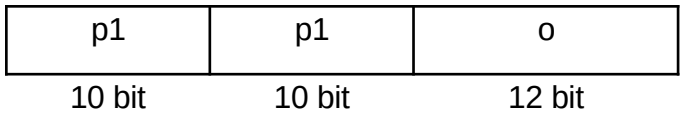
- Biến đổi địa chỉ logic thành địa chỉ tuyến tính:



## Phân trang

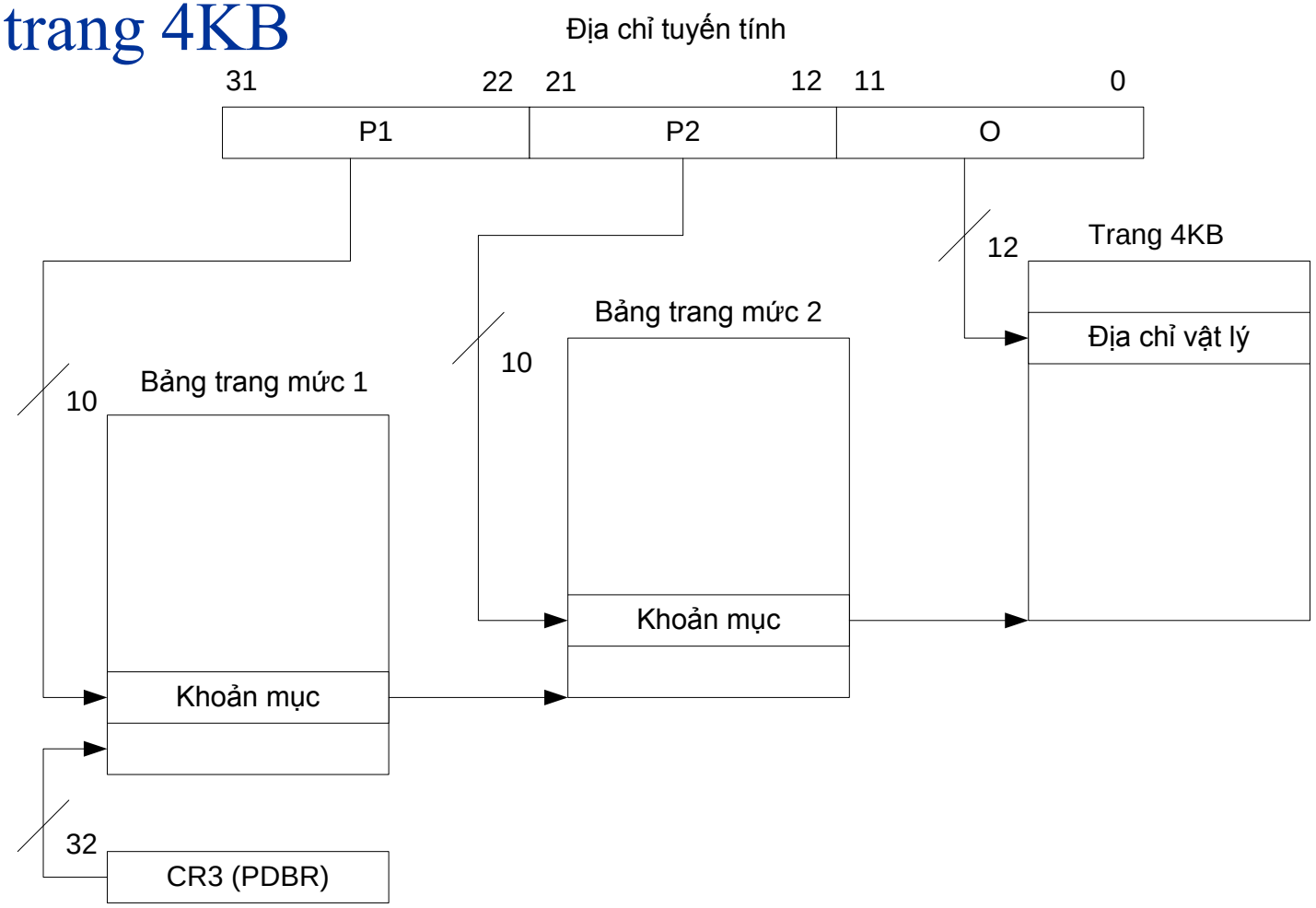
- Hỗ trợ kích thước trang 4KB hoặc 4MB, tùy thuộc vào giá trị cờ kích thước trang
- Trang kích thước 4KB: tổ chức bảng trang thành 2 mức
  - Địa chỉ tuyến tính có kích thước 32 bit



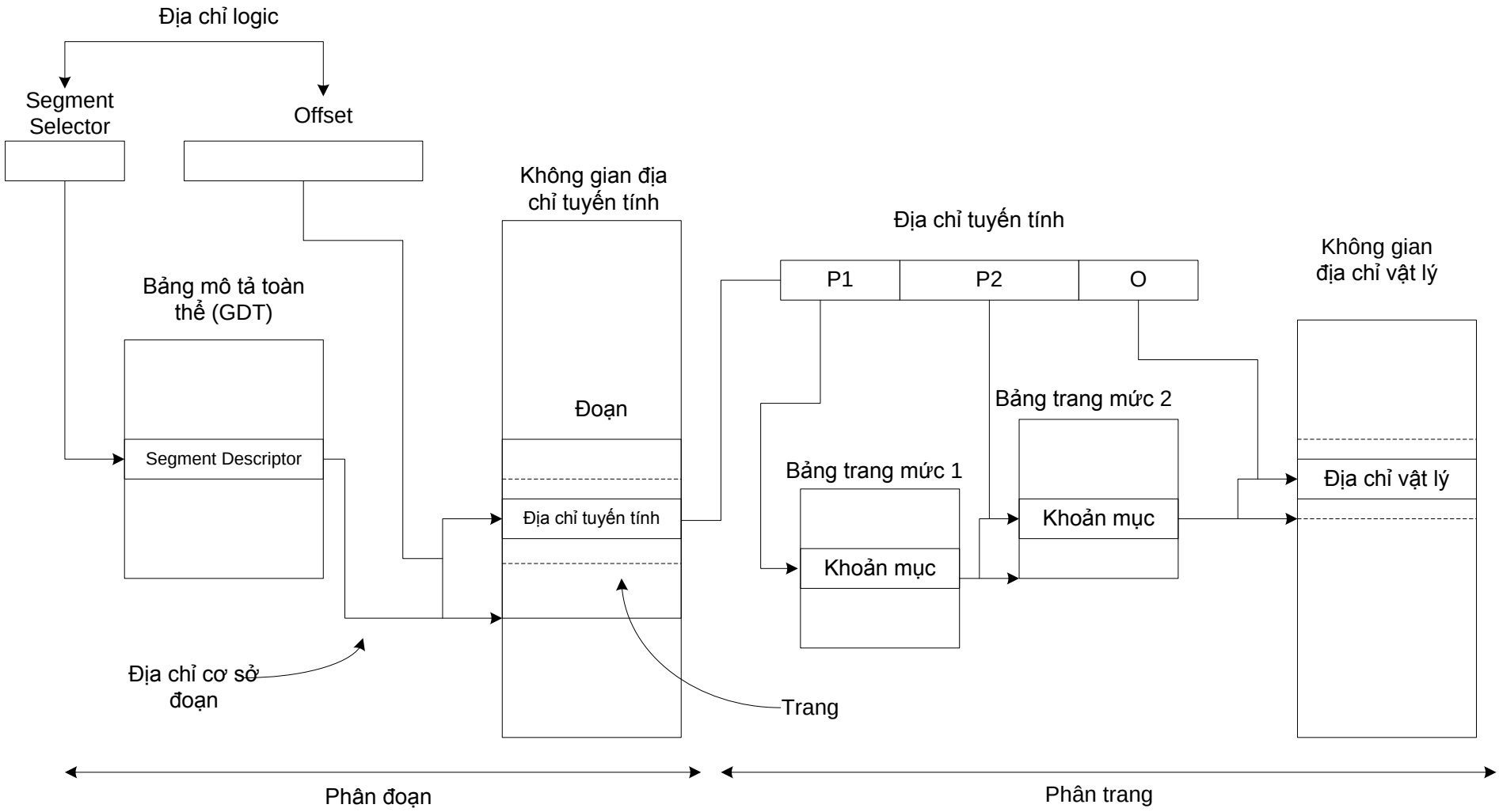

  - P1: cho phép tìm bảng trang mức hai
  - P2: tìm ô tương ứng trong bảng trang mức 2 kết hợp với độ dịch o tạo ra địa chỉ vật lý
- Trang kích thước 4MB: Bảng trang chỉ có một mức
    - P :10bit
    - O: độ dịch, kích thước 22bit cho phép trỏ tới vị trí cụ thể trong trang nhớ 4MB

## Phân trang

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



## Ánh xạ địa chỉ



# IX. 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

# IX. QUẢN LÝ BỘ NHỚ TRONG WINDOWS XP

- 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
- HDH 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