

# Chương 4

## QUẢN LÝ BỘ NHỚ



# Nội dung chương 4

---

1. Địa chỉ và các vấn đề liên quan.
2. Một số cách tổ chức chương trình.
3. Phân chương bộ nhớ.
4. Phân đoạn bộ nhớ.
5. Phân trang bộ nhớ.
6. Bộ nhớ ảo.
7. Cấp phát khung trang.
8. Tình trạng trì trệ.



# Địa chỉ và các vấn đề liên quan

---

- ❖ 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 trình)

# Địa chỉ và các vấn đề liên quan

- ❖ 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 sự truy xuất bộ nhớ từ các process có hợp lệ không.
  - **Chia sẻ**: cho phép các process chia sẻ vùng nhớ chung. Đây là tính mềm dẻo mà các chiến lược quản lý cần có.
  - Kết gán địa chỉ nhớ luận lý của user vào địa chỉ thực

# Địa chỉ và các vấn đề liên quan

## Địa chỉ nhớ

- ❖ **Địa chỉ vật lý** (physical address - địa chỉ *thực*): một vị trí thực trong bộ nhớ chính.
- ❖ **Địa chỉ luận lý** (logical address): một vị trí nhớ được diễn tả trong một chương trình
  - ✓ **Trình biên dịch** (compiler) tạo ra mã lệnh chương trình trong đó mỗi tham chiếu bộ nhớ đều là địa chỉ luận lý
  - ✓ **Địa chỉ tương đối** (relative address): 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



# Địa chỉ và các vấn đề liên quan

## Địa chỉ nhớ

- ❖ Khi một lệnh được thực thi, các tham chiếu đến địa chỉ luận lý phải được chuyển đổi thành địa chỉ thực.
- ❖ Thao tác chuyển đổi thường có sự hỗ trợ của phần cứng để đạt hiệu suất cao

# Địa chỉ và các vấn đề liên quan

## Chuyển đổi địa chỉ

- ❖ Là quá trình ánh xạ một địa chỉ từ không gian địa chỉ nay 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 tương đối**  
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<sup>7</sup>1212

# Địa chỉ và các vấn đề liên quan

## Chuyển đổi địa chỉ

- ❖ 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 gán địa chỉ tuyệt đối lúc biên dịch.
    - Khuyết điểm: cần biên dịch lại nếu thay đổi địa chỉ nạp chương trình
  - ✓ **Load time**: tại thời điểm biên dịch, nếu chưa biết tiến trình sẽ nằm ở đâu trong bộ nhớ thì compiler phải sinh mã địa chỉ tương đối. Vào thời điểm loading, **loader** phải chuyển đổi địa chỉ tương đối 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.



# Địa chỉ và các vấn đề liên quan

## Chuyển đổi địa chỉ

- ✓ **Execution time**: khi trong quá trình thực thi, process có thể được di chuyển từ segment nay sang segment khác trong bộ nhớ thì quá trình chuyển đổi địa chỉ được trì hoãn đến thời điểm thực thi
  - CPU tạo ra địa chỉ luận lý cho process
  - Phần cứng cần hỗ trợ cho việc ánh xạ địa chỉ.
- Ví dụ: trường hợp địa chỉ luận lý là tương đối thì có thể dùng thanh ghi base và limit, ...
  - Sử dụng trong đa số các Hệ điều hành đa dụng (general-purpose) trong đó có các cơ chế swapping, paging, segmentation.



# Một số cách tổ chức chương trình

- ❖ Các chương trình được tổ chức theo các cấu trúc sau:
  - Cấu trúc tuyến tính
  - Cấu trúc động
  - Cấu trúc Overlay
  - Cấu trúc phân trang
  - Cấu trúc phân đoạn



# Một số cách tổ chức chương trình

- Cấu trúc tuyến tính:
  - ✓ Tất cả các module, thư viện sử dụng trong chương trình khi biên dịch sẽ được biên dịch thành một module duy nhất
  - ✓ Khi thực thi, hệ điều hành phải nạp toàn bộ module này vào bộ nhớ
  - ✓ Chương trình dạng này có tính độc lập cao, tốc độ thực thi cao
  - ✓ Chi phí về bộ nhớ cao.



# Một số cách tổ chức chương trình

- Cấu trúc động:
  - ✓ Chương trình được viết dưới dạng các module riêng rẽ.
  - ✓ Các module riêng rẽ được biên dịch thành các module riêng, không tích hợp.
  - ✓ Khi thực thi, hệ điều hành chỉ nạp module chính vào bộ nhớ. Các module khác sẽ nạp khi cần thiết.
  - ✓ Tiết kiệm chi phí về bộ nhớ.
  - ✓ Tốc độ thực thi thấp hơn dạng tuyến tính.



# Một số cách tổ chức chương trình

- Cấu trúc overlay:
  - ✓ Chương trình được biên dịch thành các module riêng rẽ.
  - ✓ Các module chương trình được chia thành các mức khác nhau:
    - Mức 0: Chứa module gốc dùng để nạp chương trình.
    - Mức 1: Chứa các module được gọi bởi mức 0.
    - Mức 2: Chứa các module được gọi bởi mức 1.
    - ...

# Một số cách tổ chức chương trình

- Cấu trúc overlay (tt):
  - ✓ Các module trong cùng một mức có thể có kích thước khác nhau. Kích thước của module lớn nhất trong lớp được xem là kích thước của mức.
  - ✓ Bộ nhớ dành cho chương trình cũng được tổ chức thành các mức tương ứng với các chương trình.
  - ✓ Khi thực thi chương trình hệ điều hành nạp sơ đồ overlay của chương trình vào bộ nhớ, sau đó nạp các module cần thiết ban đầu vào bộ nhớ.
  - ✓ Hệ điều hành dựa vào sơ đồ overlay để nạp các module khác nếu cần.



# Một số cách tổ chức chương trình

- Cấu trúc phân trang:
  - ✓ Các module chương trình được biên dịch thành một module duy nhất nhưng sau đó được chia thành các phần có kích thước bằng nhau được gọi là các trang.
  - ✓ Bộ nhớ phải được phân trang, tức chia thành các không gian nhớ bằng nhau gọi là khung trang.
  - ✓ Hệ điều hành có các bộ điều khiển trang (**PCT** – Page Control Table)

# Một số cách tổ chức chương trình

- Cấu trúc phân đoạn:
  - ✓ Chương trình được biên dịch thành nhiều module độc lập, được gọi là các đoạn.
  - ✓ Bộ nhớ được phân đoạn thành các không gian có kích thước khác nhau tương ứng với kích thước của các đoạn chương trình.
  - ✓ Khi thực thi chương trình, hệ điều hành có thể nạp tất cả các đoạn, hoặc một vài đoạn cần thiết, vào các phân đoạn nhớ liên tiếp hoặc không liên tiếp.
  - ✓ Hệ điều hành có bộ điều khiển đoạn (**SCT** – Segment Control Table)



# Phân chương bộ nhớ

## ❖ Phân vùng cố định:

- Không gian địa chỉ được chia thành 2 vùng cố định:
  - ✓ Vùng địa chỉ thấp dùng để chứa hệ điều hành
  - ✓ Vùng còn lại cấp cho các tiến trình được nạp vào bộ nhớ chính (vùng dành cho người dùng).
- Với hệ điều hành đơn chương:
  - ✓ Vùng dành cho người dùng chỉ cấp cho một process
  - ✓ Hệ điều hành dùng thanh ghi giới hạn để phân biệt 2 vùng.
  - ✓ Khi chương trình người dùng đưa ra địa chỉ cần truy xuất, hệ điều hành sẽ so sánh với giá trị giới hạn trong thanh ghi giới hạn để kiểm soát việc truy xuất.

# Phân chương bộ nhớ

## ❖ Phân vùng cố định:

### ➤ Với hệ điều hành đa chương:

- ✓ Vùng nhớ người dùng được chia thành nhiều phần có kích thước khác nhau. Mỗi phần được gọi là phân vùng.
- ✓ Mỗi tiến trình có thể được nạp vào một phân vùng còn trống bất kỳ, nếu kích thước của nó là phù hợp.
- ✓ Khi có tiến trình cần được nạp vào bộ nhớ, nếu không còn vùng nhớ trống thì hệ điều hành sẽ chuyển (**swap out**) tiến trình nào độc lập và đang ở trạng thái **ready** hoặc **running** ra ngoài để nạp tiến trình này.



# Phân chương bộ nhớ

## ❖ Phân vùng cố định:

- Khi chương trình có kích thước quá lớn so với phân vùng?
  - ✓ Có thể thiết kế theo cấu trúc overlay.
  - ✓ Chỉ nạp phần cần thiết vào bộ nhớ. Khi cần nạp thêm module mới thì ghi đè lên nội dung đang nhớ.
- Khi chương trình có kích thước quá nhỏ so với phân vùng?
  - ✓ Xảy ra hiện tượng phân mảnh trong.
- Để khắc phục các tình huống trên có thể dùng hàng đợi cho từng phân vùng hoặc một hàng đợi cho tất cả các phân vùng.



# Phân chương bộ nhớ

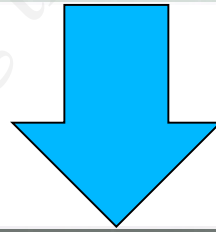
## ❖ Phân vùng động:

- Vùng nhớ người dùng được phân chia trước.
- Khi một tiến trình được nạp vào bộ nhớ thì hệ điều hành sẽ cấp phát không gian nhớ vừa đủ cho nó.
- Khi tiến trình kết thúc, hệ điều hành sẽ thu hồi để cấp cho tiến trình khác.
- Hệ điều hành phải có cơ chế thích hợp để quản lý các khối nhớ đã cấp phát hay còn trống.
- Hai cơ chế thường được sử dụng: Bản đồ bit và Danh sách liên kết.
- Cả hai cơ chế trên đều dựa vào nguyên tắc chia không gian nhớ thành các vùng cùng kích thước.

# Phân chương bộ nhớ

- ❖ Phân vùng động:
  - Bản đồ bit

		A						B						C				D			
00	01	02	03	04	05	06	07	08	09	10	11	12	13	14	15	16	17	18	19	20	21

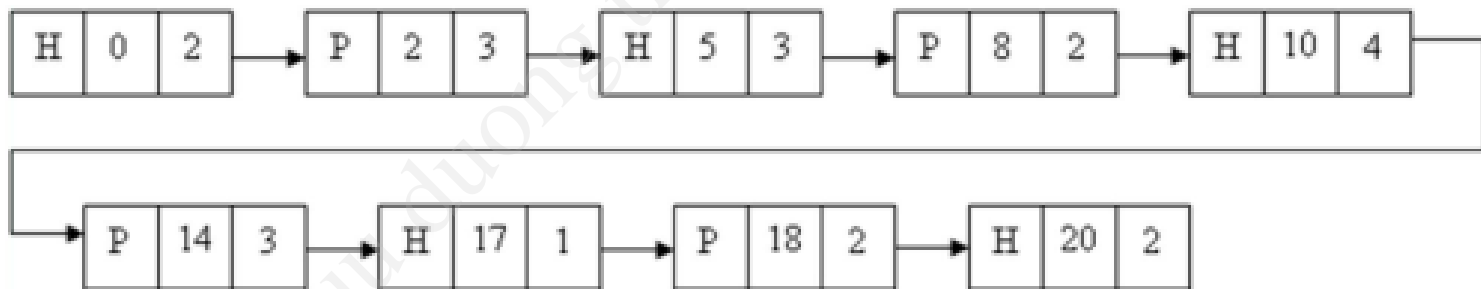


0	0	1	1	1	0	0	0	1	1	0	0	0	0	1	1	1	0	1	1	0	0
---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---

# Phân chương bộ nhớ

- ❖ Phân vùng động:
  - Danh sách liên kết

		A						B						C				D			
00	01	02	03	04	05	06	07	08	09	10	11	12	13	14	15	16	17	18	19	20	21



- ✓ H|0|2: Không gian trống gồm 2 khối, bắt đầu từ khối 0.
- ✓ P|2|3: Không gian 3 khối đã cấp phát bắt đầu từ khối 2.

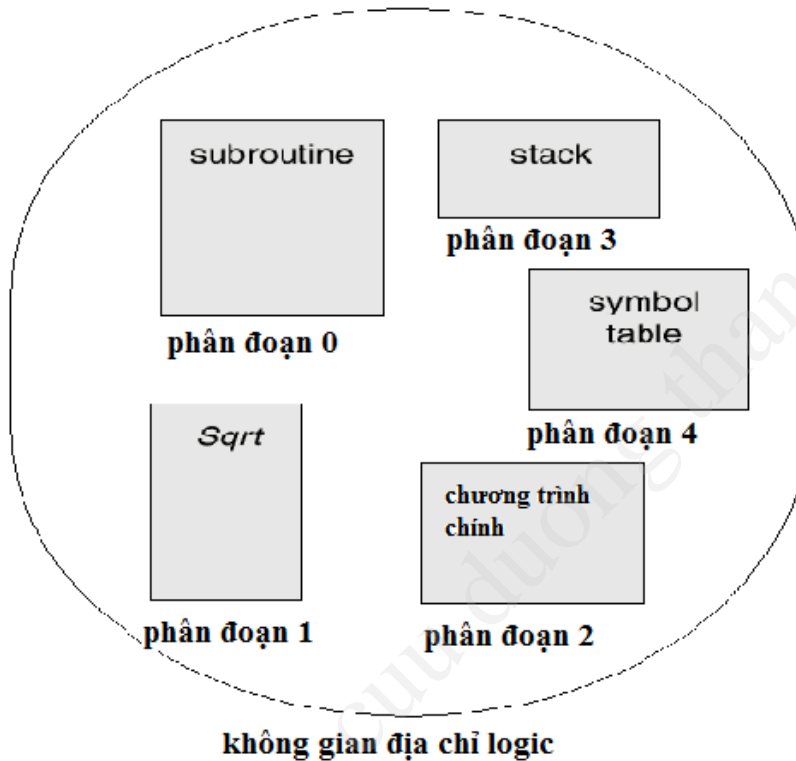
# Phân chương bộ nhớ

## ❖ Phân vùng động:

### ➤ Các chiến lược cấp phát động:

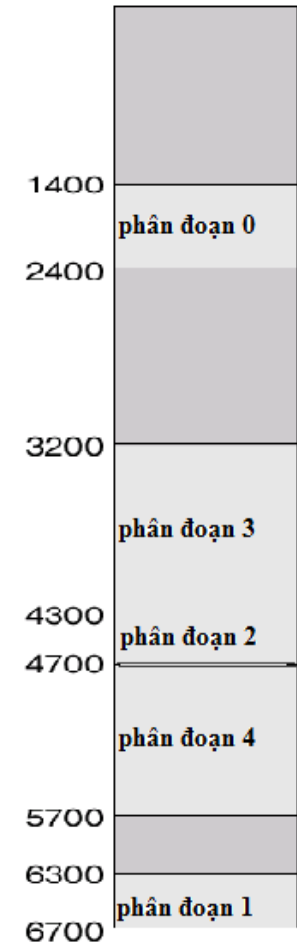
- ✓ **Best-fit**: Chọn khối nhớ có kích thước vừa đúng bằng kích thước của tiến trình cần nạp vào bộ nhớ.
- ✓ **First-fit**: Hệ điều hành sẽ tìm kiếm khối nhớ phù hợp với yêu cầu. Khối nhớ đầu tiên phù hợp tìm được sẽ được chọn để cấp phát.
- ✓ **Next-fit**: Chọn khối nhớ phù hợp ngay sau khối nhớ vừa được cấp phát để nạp tiến trình.
- ✓ **Worst-fit**: Chọn khối nhớ trống lớn nhất.

# Phân đoạn bộ nhớ (Segmentation)



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

Bảng phân đoạn

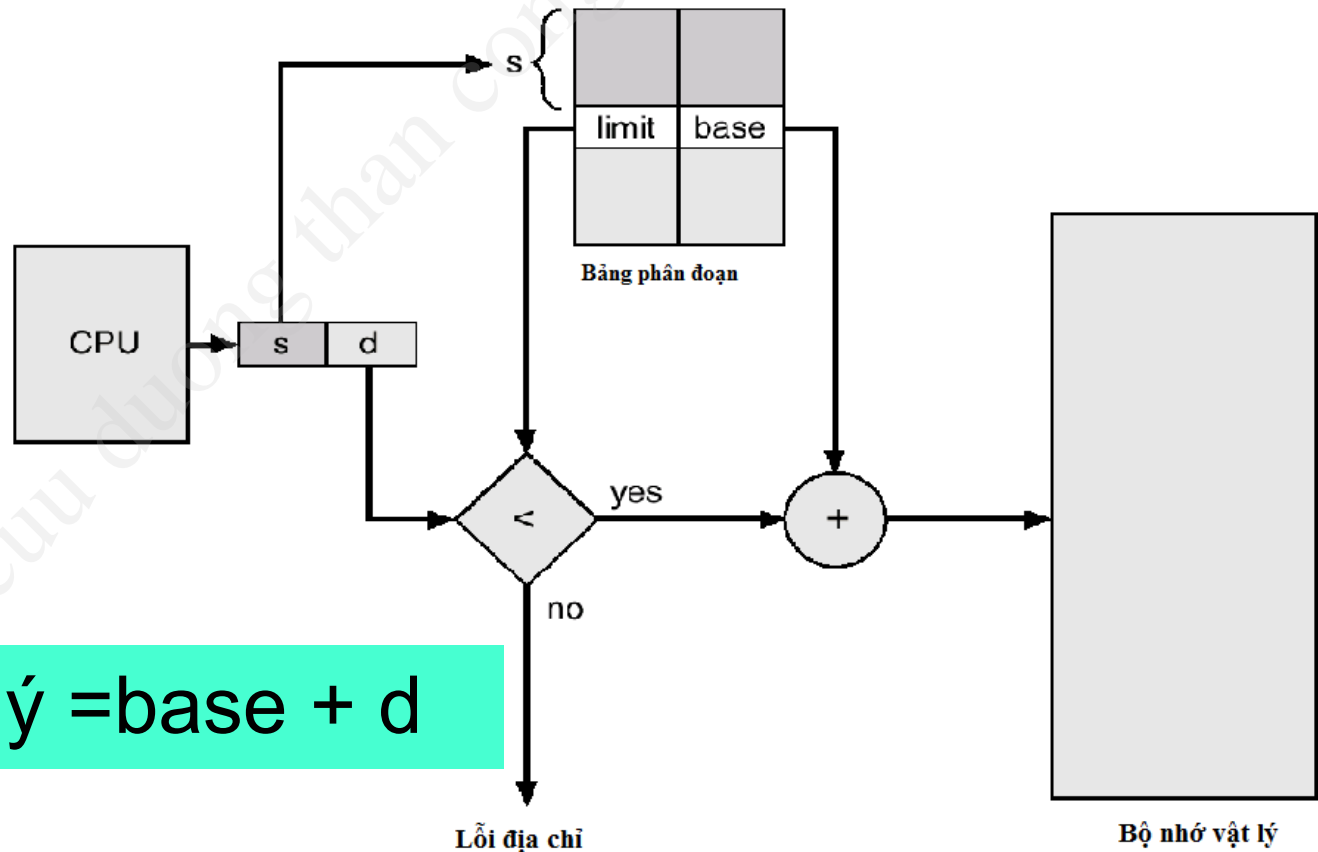


Bộ nhớ vật lý



# Phân đoạn bộ nhớ (Segmentation)

- ❖ Cơ chế MMU (Memory Management Unit) trong kỹ thuật phân đoạn

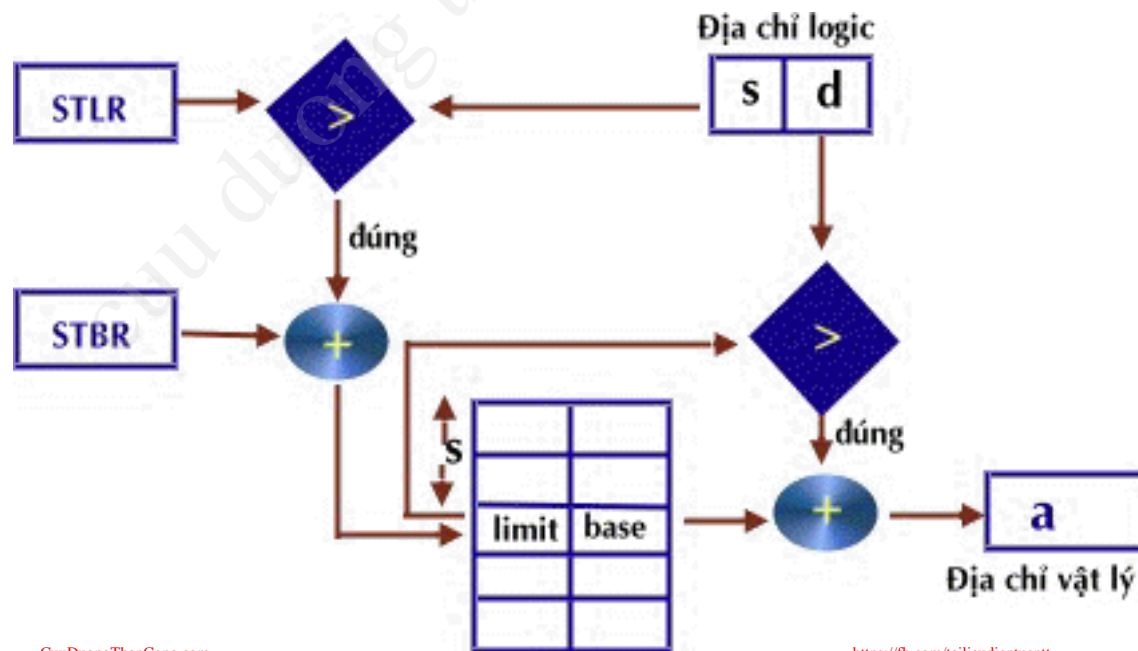


Địa chỉ vật lý = base + d

# Phân đoạn bộ nhớ (Segmentation)

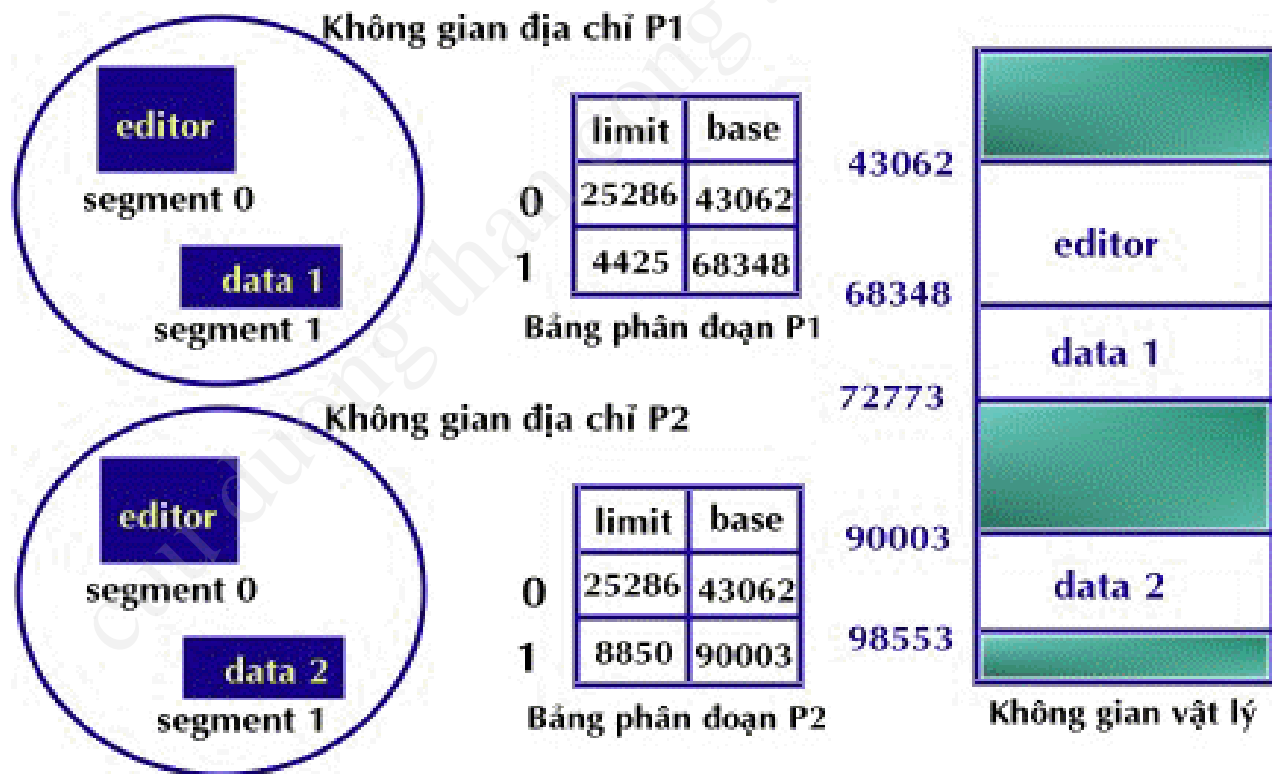
## ❖ Cài đặt bảng phân đoạn

- STBR (**Segment Table Base Register**) để lưu địa chỉ bắt đầu của bảng phân đoạn.
- STLR (**Segment Table Limit Register**) lưu số phân đoạn.



# Phân đoạn bộ nhớ (Segmentation)

## ❖ Chia sẻ phân đoạn



- ✓ MMU gán hai phần tử trong hai bảng phân đoạn của hai tiến trình cùng giá trị

# Phân đoạn bộ nhớ (Segmentation)

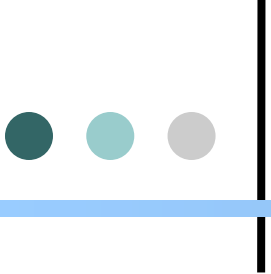
## ❖ Bảo vệ phân đoạn

- Attribute: R (chỉ đọc), X (thực thi), W (ghi),...

Limit	Base	Attribute
-------	------	-----------

## ➤ Nhận xét về kỹ thuật phân đoạn:

- ✓ Hiện tượng phân mảnh ngoại vi vẫn xảy ra.
- ✓ Ưu điểm: mã chương trình và dữ liệu được tách riêng -> dễ dàng bảo vệ mã chương trình và dễ dàng dùng chung dữ liệu hoặc hàm



# Phân trang bộ nhớ (paging)

## ❖ Phân trang đơn:

- Bộ nhớ chính được chia thành các phần bằng nhau và cố định. Đánh số thứ tự cho các phần này, bắt đầu từ số 0. Các phần này được gọi là các khung trang.
- Không gian địa chỉ của các tiến trình cũng được chia thành các phần có kích thước bằng kích thước khung trang và được gọi là các trang.
- Khi tiến trình nạp vào bộ nhớ thì các trang được nạp vào các khung trang bất kỳ còn trống. Các khung trang này có thể không liên tiếp nhau.

# Phân trang bộ nhớ (paging)

Không  
gian địa  
chỉ ảo

60K-64K	X
56K-60K	X
52K-56K	X
48K-52K	X
44K-48K	7
40K-44K	X
36K-40K	5
32K-36K	X
28K-32K	X
24K-28K	X
20K-24K	3
16K-20K	4
12K-16K	0
8K-12K	6
4K-8K	1
0K-4K	2

} Trang ảo

Không  
gian địa  
chỉ vật lý

28K-32K
24K-28K
20K-24K
16K-20K
12K-16K
8K-12K
4K-8K
0K-4K

Khung trang

Page 3
Page 2
Page 1
Page 0

Không  
gian địa  
chỉ ảo

7	Page 1	7168
6		6144
5	Page 0	5120
4		4096
3		3072
2	Page 3	2048
1		1024
0	Page 2	0000

Không gian địa chỉ  
vật lý

3	2
2	0
1	7
0	5

Bảng  
trang

Bộ nhớ vật lý: khung trang (page frame). Không gian địa chỉ ảo: trang (page).

# Phân trang bộ nhớ (paging)

## ❖ Cấu trúc địa chỉ ảo:

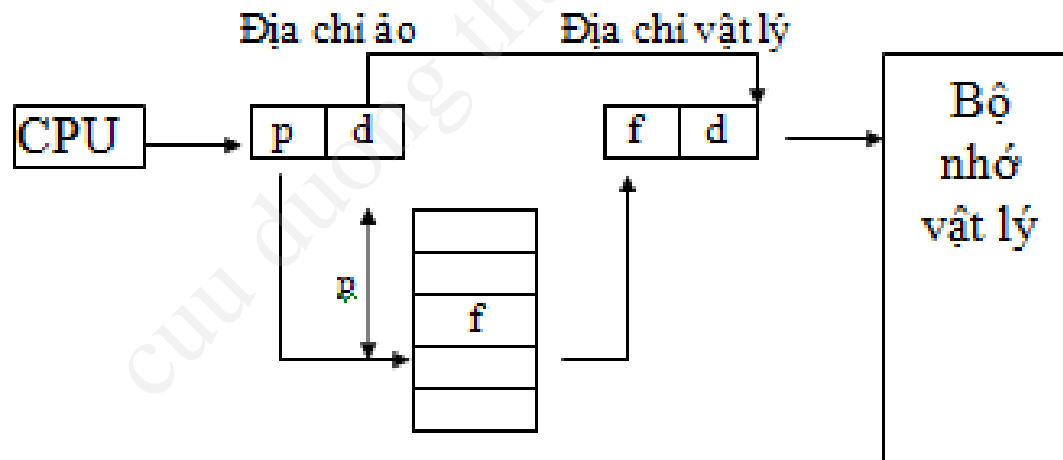
- Kích thước của trang là lũy thừa của  $2^n$  ( $9 \leq n \leq 13$ )
- Kích thước của không gian địa chỉ ảo là  $2^m$  (CPU dùng địa chỉ ảo  $m$  bit)
  - ✓  $m-n$  bit cao của địa chỉ ảo sẽ biểu diễn số hiệu trang, và  $n$  bit thấp biểu diễn địa chỉ tương đối trong trang

địa chỉ ảo dạng (p,d) có $m$ bit	
p là số hiệu trang và chiếm $m-n$ bit cao	d là địa chỉ tương đối trong trang và chiếm $n$ bit thấp

# Phân trang bộ nhớ (paging)

- ❖ Cơ chế MMU (Memory Management Unit) trong mô hình phân trang

Địa chỉ vật lý = vị trí bắt đầu của khung trang  $f$  +  $d$



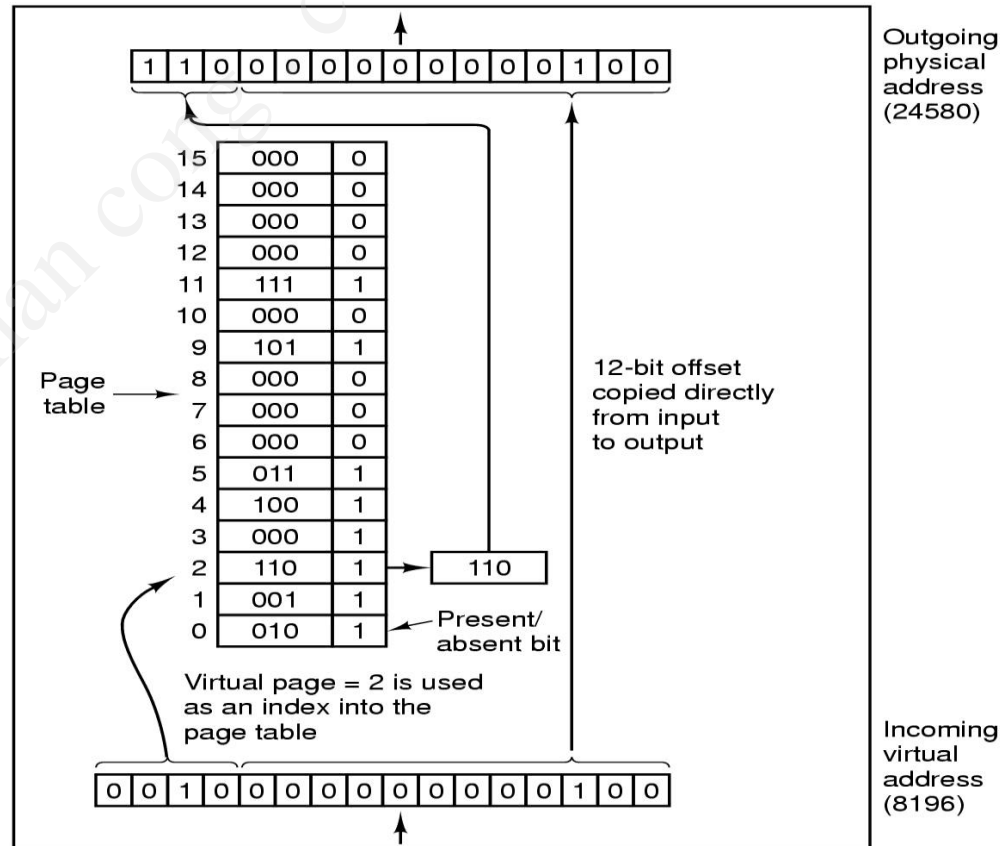
Giả sử tiến trình truy xuất địa chỉ ảo  $(p, d) = (3, 500)$   
→ địa chỉ vật lý là  $2048 + 500 = 2548$ .



# Phân trang bộ nhớ (paging)

- ❖ Cách tính địa chỉ vật lý của MMU:
  - chép d vào n bit thấp của địa chỉ vật lý.
  - chép f vào (m-n) bit cao của địa chỉ vật lý

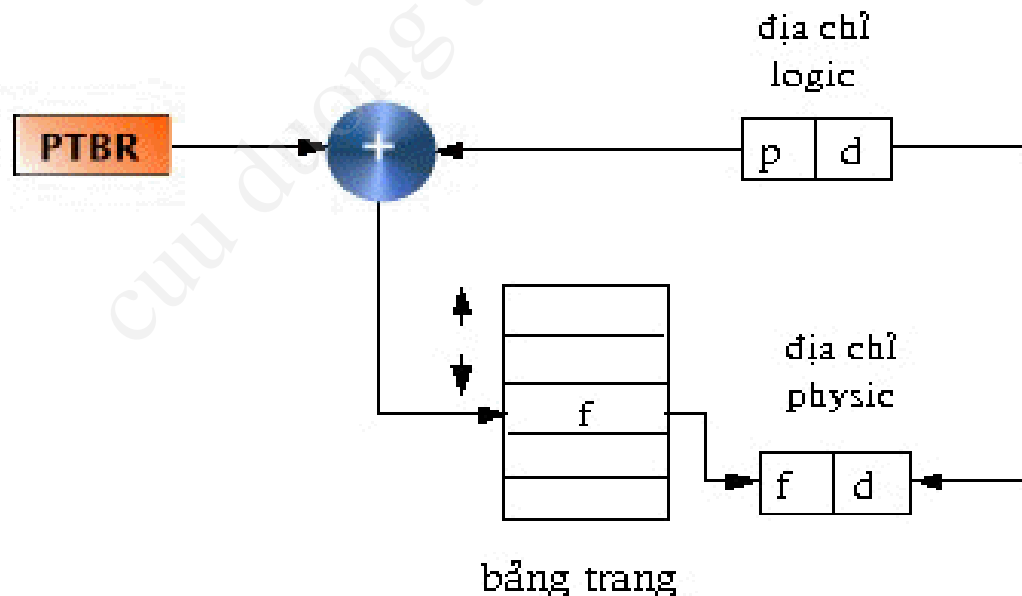
Ví dụ: Một hệ thống có địa chỉ ảo 16 bit dạng (p,d) với p có 4 bit, d có 12 bit (hệ thống có 16 trang, mỗi trang 4 KB) . Bit Present/absent =1 nghĩa là trang hiện ở trong bộ nhớ và =0 là ở bộ nhớ phụ.



# Phân trang bộ nhớ (paging)

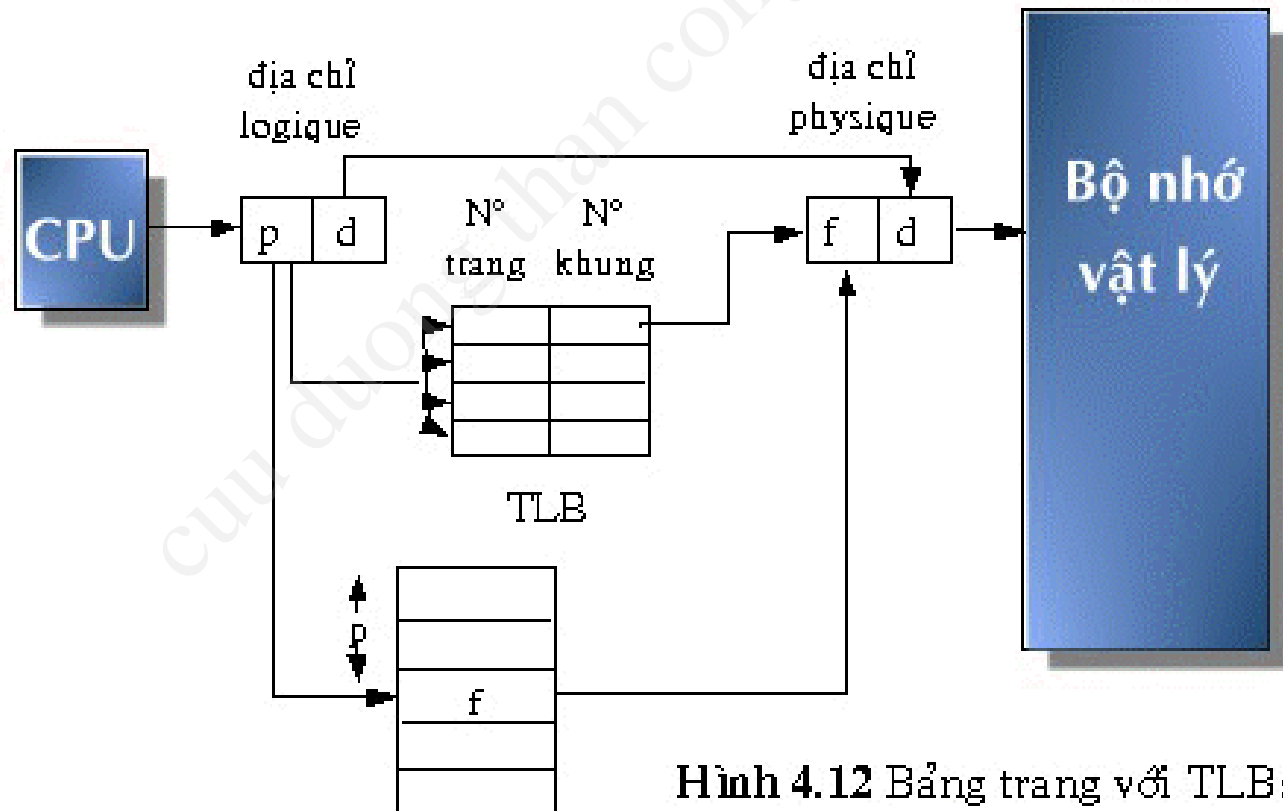
## ❖ Cài đặt bảng trang

- Thanh ghi PTBR (**Page Table Base Register**): lưu địa chỉ bắt đầu của bảng trang.
- Thanh ghi PTLR (**Page Table Limit Register**): lưu số phần tử trong bảng trang.

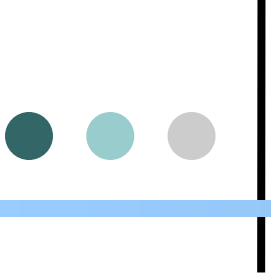


# Phân trang bộ nhớ (paging)

## ❖ Bộ nhớ kết hợp (Translation Lookaside Buffers - TLBs)



Hình 4.12 Bảng trang với TLBs



# Phân trang bộ nhớ (paging)

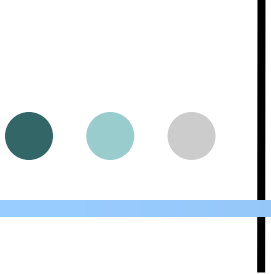
Ví dụ:

➤ Một hệ thống máy tính 32 bit: kích thước 1 khung trang là 4K. Hỏi hệ thống quản lý được tiến trình kích thước tối đa là bao nhiêu?

➤ Máy tính 32 bit  $\Rightarrow$  địa chỉ ảo (p,d) có 32 bit  $\Rightarrow$  số bit của p + số bit của d = 32, mà 1 trang  $4K = 2^{12}$  bytes  $\Rightarrow$  d có 12 bit  $\Rightarrow$  p có 20 bit  $\Rightarrow$  1 bảng trang có  $2^{20}$  phần tử.

$\Rightarrow$  Hệ thống quản lý được tiến trình có tối đa  $2^{20}$  trang  $\Rightarrow$  kích thước tiến trình lớn nhất là  $2^{20} \times 2^{12}$  byte =  $2^{32}$  byte = 4 GB.

Nhận xét: Máy tính n bit quản lý được tiến trình kích thước lớn nhất là  $2^n$  byte.



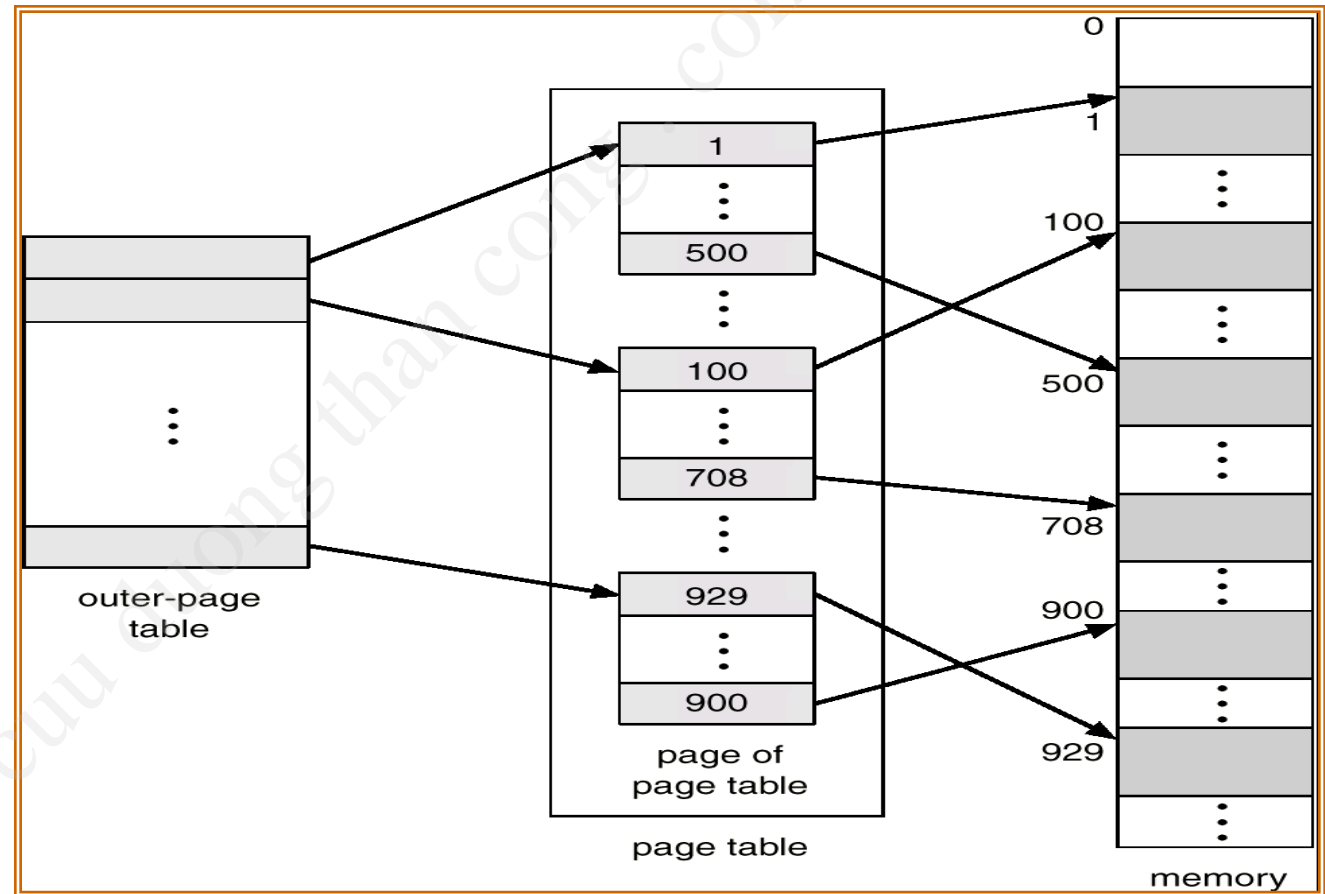
# Phân trang bộ nhớ (paging)

---

- ❖ Tổ chức bảng phân trang
  - Phân trang đa cấp.
  - Bảng trang băm.
  - Bảng trang nghịch đảo

# Phân trang bộ nhớ (paging)

## ❖ Phân trang đa cấp



Bảng trang cấp 1

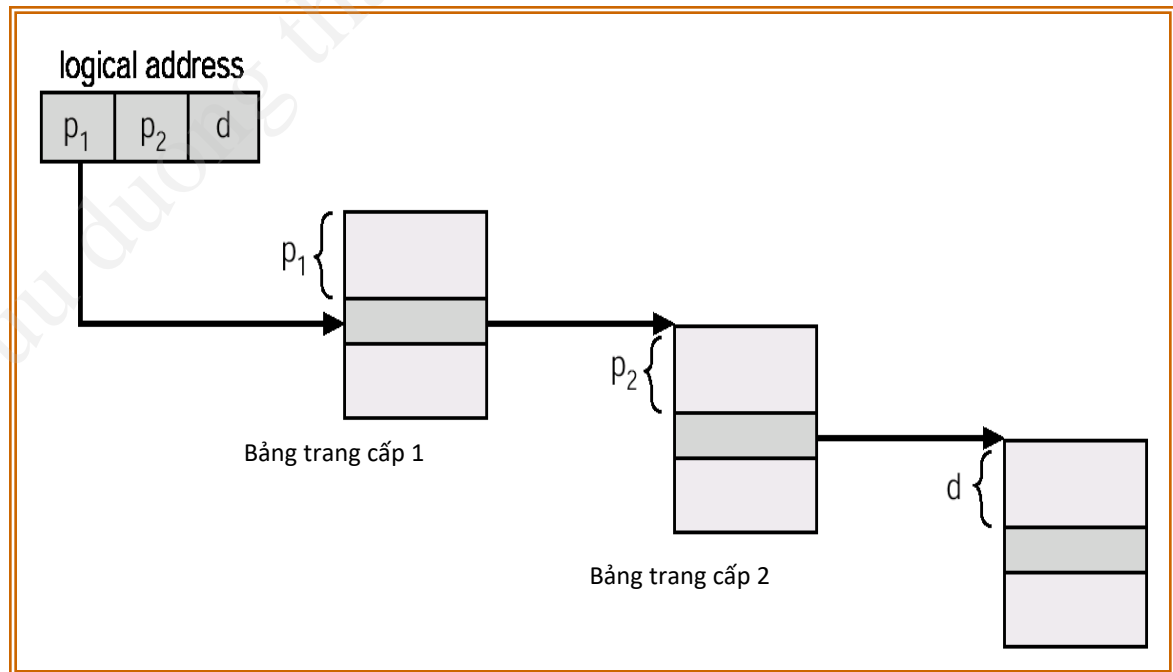
Các bảng trang cấp 2

# Phân trang bộ nhớ (paging)

## ❖ Phân trang đa cấp

page number		page offset
$p_1$	$p_2$	$d$
10	10	12

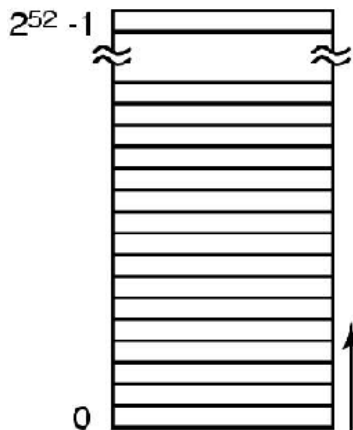
$p_1$  chỉ mục của bảng trang cấp một.  $p_2$  chỉ mục của bảng trang cấp 2



# Phân trang bộ nhớ (paging)

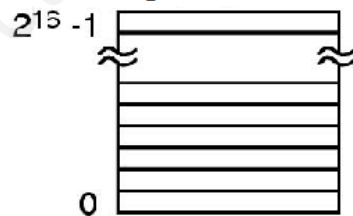
- ❖ Bảng trang băm
  - Khi không gian địa chỉ ảo > 32 bit)

Bảng trang thông thường với một entry có  $2^{52}$  trang



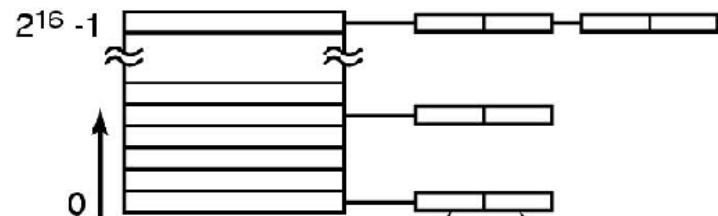
Được chỉ mục bằng trang ảo

Bộ nhớ vật lý 256MB có  $2^{16}$  4KB khung trang



- ✓ Một máy tính 64 bit, có RAM 256MB, kích thước 1 khung trang là 4KB.
- ✓ Bảng trang thông thường phải có  $2^{52}$  mục, nếu dùng bảng trang băm có thể sử dụng bảng có số mục bằng số khung trang vật lý là  $2^{16}$  ( $\ll 2^{52}$ ) với hàm băm là  $\text{hasfunc}(p) = p \bmod 2^{16}$

Bảng băm



Được chỉ mục bằng phương pháp băm trên trang ảo

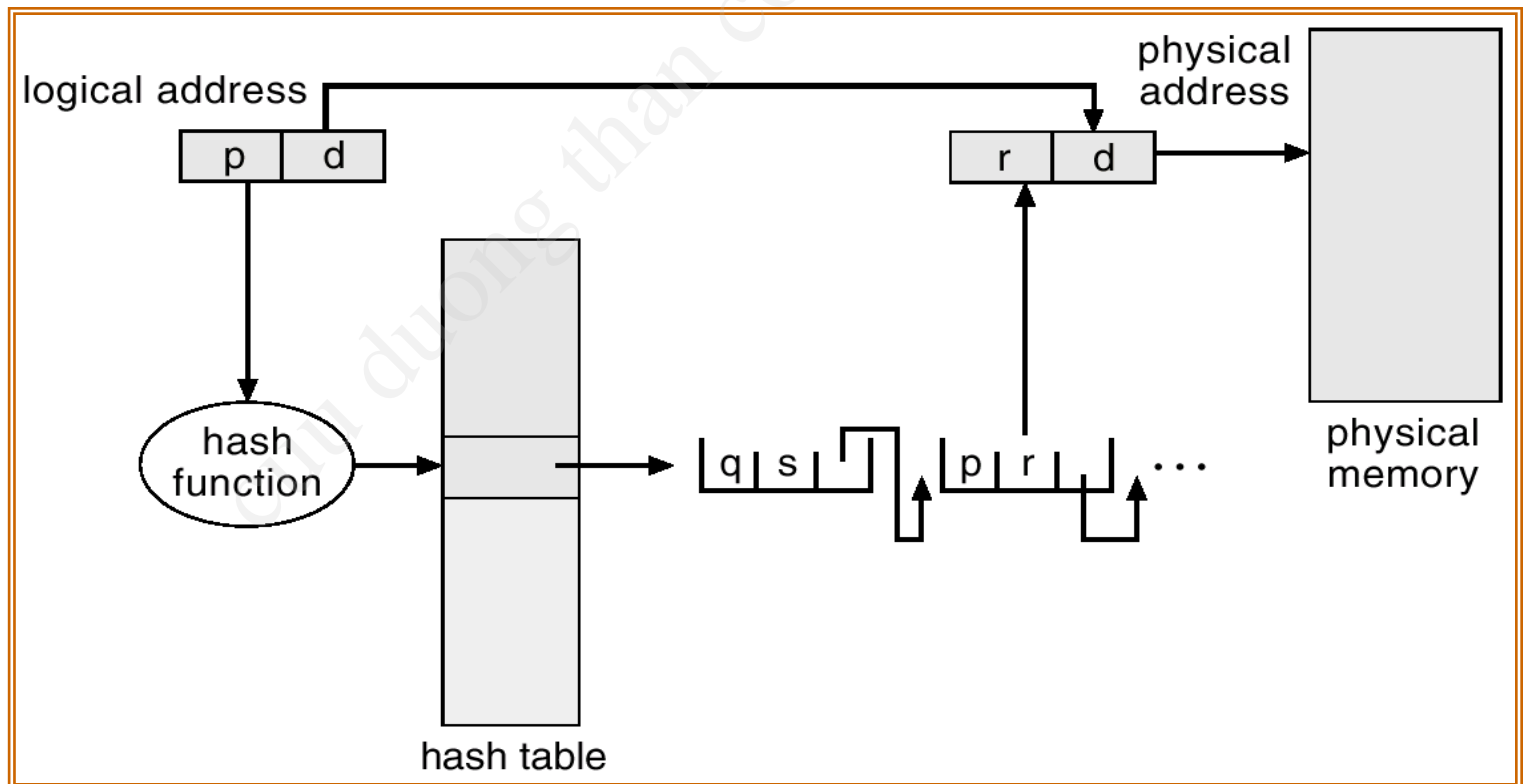
trang ảo

khung trang



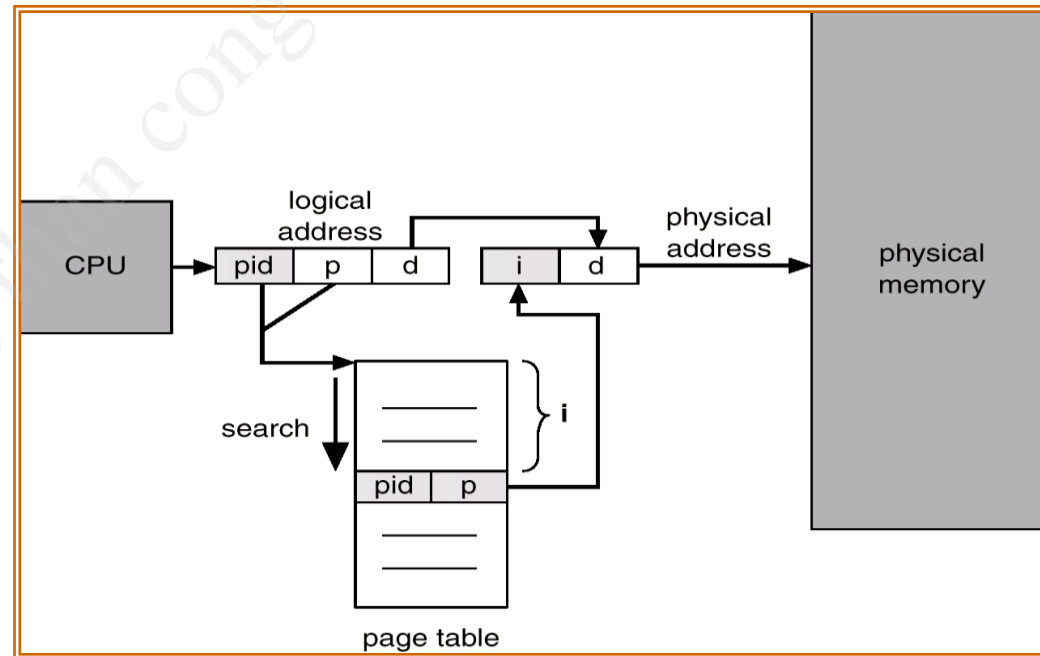
# Phân trang bộ nhớ (paging)

- ❖ Bảng trang băm: Cơ chế chuyển đổi địa chỉ khi sử dụng bảng trang băm



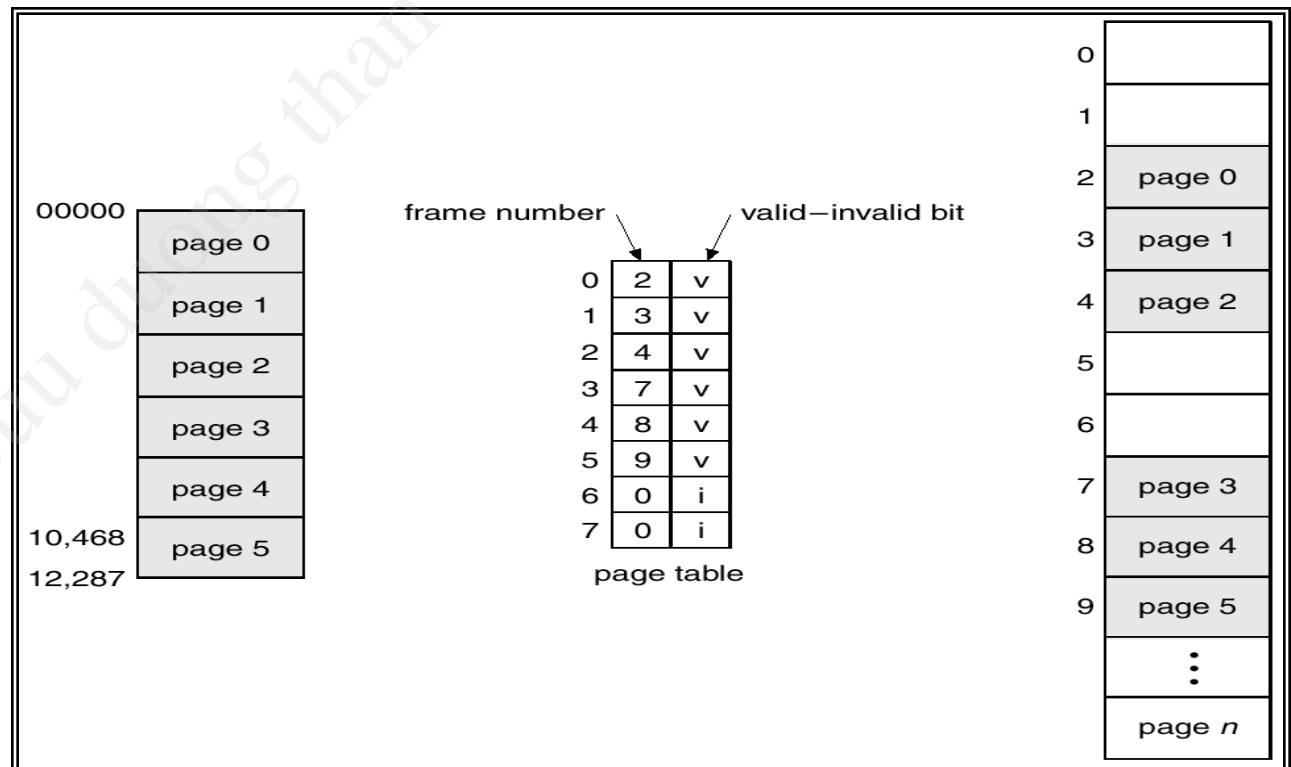
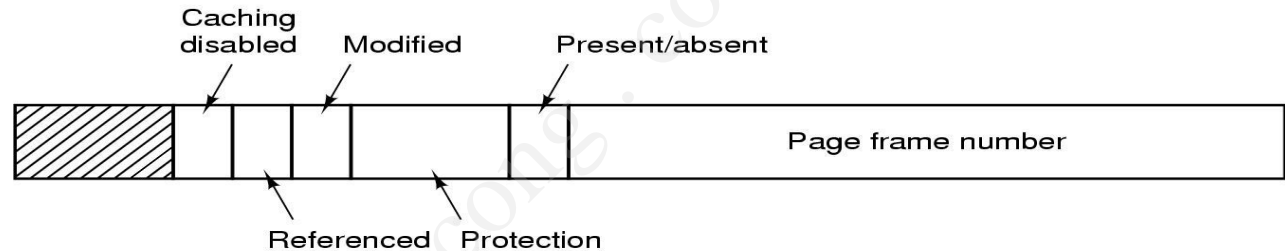
# Phân trang bộ nhớ (paging)

- ❖ Bảng trang nghịch đảo
  - Bảng trang duy nhất để quản lý bộ nhớ **của tất cả các tiến trình**.
  - Mỗi phần tử của bảng trang nghịch đảo là cặp (pid, p)
    - ✓ pid là mã số của tiến trình
    - ✓ p là số hiệu trang.
  - Mỗi địa chỉ ảo là một bộ ba (pid, p, d).



# Phân trang bộ nhớ (paging)

## ❖ Bảo vệ trang



# Phân trang bộ nhớ (paging)

## ❖ Chia sẻ bộ nhớ

- Các tiến trình dùng chung một số khung trang
  - ✓ Ghi cùng số hiệu khung trang vào bảng trang của mỗi tiến trình

3	data1
2	code3
1	code2
0	code1

P1

3	0
2	6
1	3
0	2

Bảng trang  
P1

3	data2
2	code3
1	code2
0	code1

P2

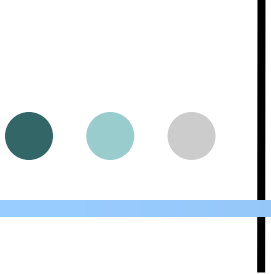
3	4
2	6
1	3
0	2

Bảng trang  
P2

7	
6	code3
5	
4	data2
3	code2
2	code1
1	
0	data1

Bộ nhớ  
Vật lý

hai tiến trình P1, P2 dùng chung ba trang 0, 1, 2



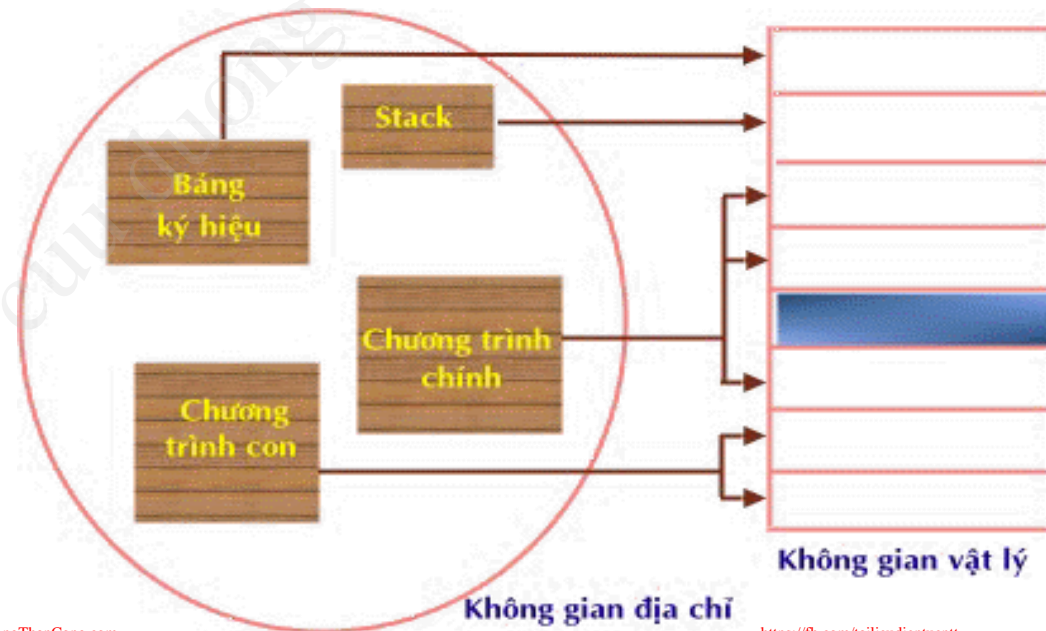
# Phân trang bộ nhớ (paging)

## ❖ Nhận xét

- Loại bỏ được hiện tượng phân mảnh ngoài.
- Vẫn có hiện tượng phân mảnh nội.
- Kết hợp cả hai kỹ thuật phân trang và phân đoạn:
  - ✓ Phân trang các phân đoạn.

# Phân trang bộ nhớ (paging)

- ❖ Mô hình phân trang kết hợp phân đoạn (Paged Segmentation)
  - Một tiến trình gồm nhiều phân đoạn.
  - Mỗi phân đoạn được chia thành nhiều trang, lưu trữ vào các khung trang có thể không liên tục.



# Phân trang bộ nhớ (paging)

- ❖ Cơ chế MMU trong mô hình phân đoạn kết hợp phân trang

