

# HỆ ĐIỀU HÀNH

(Operation System)

Th.S Nguyễn Hoàng Thành

Email: [thanhnh@ptithcm.edu.vn](mailto:thanhnh@ptithcm.edu.vn)

Tel: 0909 682 711

# QUẢN LÝ BỘ NHỚ

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. Bộ nhớ ảo
6. Cấp phát khung trang
7. Tình trạng trì trệ

# 1. 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 Tiến trình trong bộ nhớ sao cho hiệu quả
- Mục tiêu cần đạt được là **nạp càng nhiều Tiến trình vào bộ nhớ càng tốt** (gia tăng mức độ đa chương trình)

# 1. Địa chỉ và các vấn đề liên quan

- Các yêu cầu đối với việc quản lý bộ nhớ
  - **Cấp phát** bộ nhớ cho các Tiến trình
  - **Tái định vị** (relocation): khi swapping,...
  - **Bảo vệ**: phải kiểm tra sự truy xuất bộ nhớ từ các Tiến trình có hợp lệ hay không?
  - **Chia sẻ**: cho phép các Tiến trình chia sẻ vùng nhớ chung. Đây là tính mềm dẻo mà các chiến lược quản lý cần có.
- Kết gắn địa chỉ nhớ luận lý của user vào địa chỉ thực

# 1. 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.

# 1. 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.

# 1. 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 này sang không gian địa chỉ khác
- Biểu diễn địa chỉ nhớ
  - Trong source code: symbolic (các biến, hằng, pointer,...)
  - Thời điểm **biên dịch**: thường là **địa chỉ 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 1212

000000000000000000 <main>:

```

  0: f3 0f 1e fa          endbr64
  4: 55                  push    %rbp
  5: 48 89 e5            mov     %rsp,%rbp
  8: 48 83 ec 10         sub     $0x10,%rsp
 c: c7 45 fc 00 00 00 00 movl    $0x0,-0x4(%rbp)
13: 8b 45 fc            mov     -0x4(%rbp),%eax
16: 89 c6              mov     %eax,%esi
18: 48 8d 05 00 00 00 00 lea     0x0(%rip),%rax      # 1f <
1f: 48 89 c7            mov     %rax,%rdi
22: b8 00 00 00 00     mov     $0x0,%eax
27: e8 00 00 00 00     call    2c <main+0x2c>
2c: b8 00 00 00 00     mov     $0x0,%eax
31: c9                leave
32: c3                ret
```

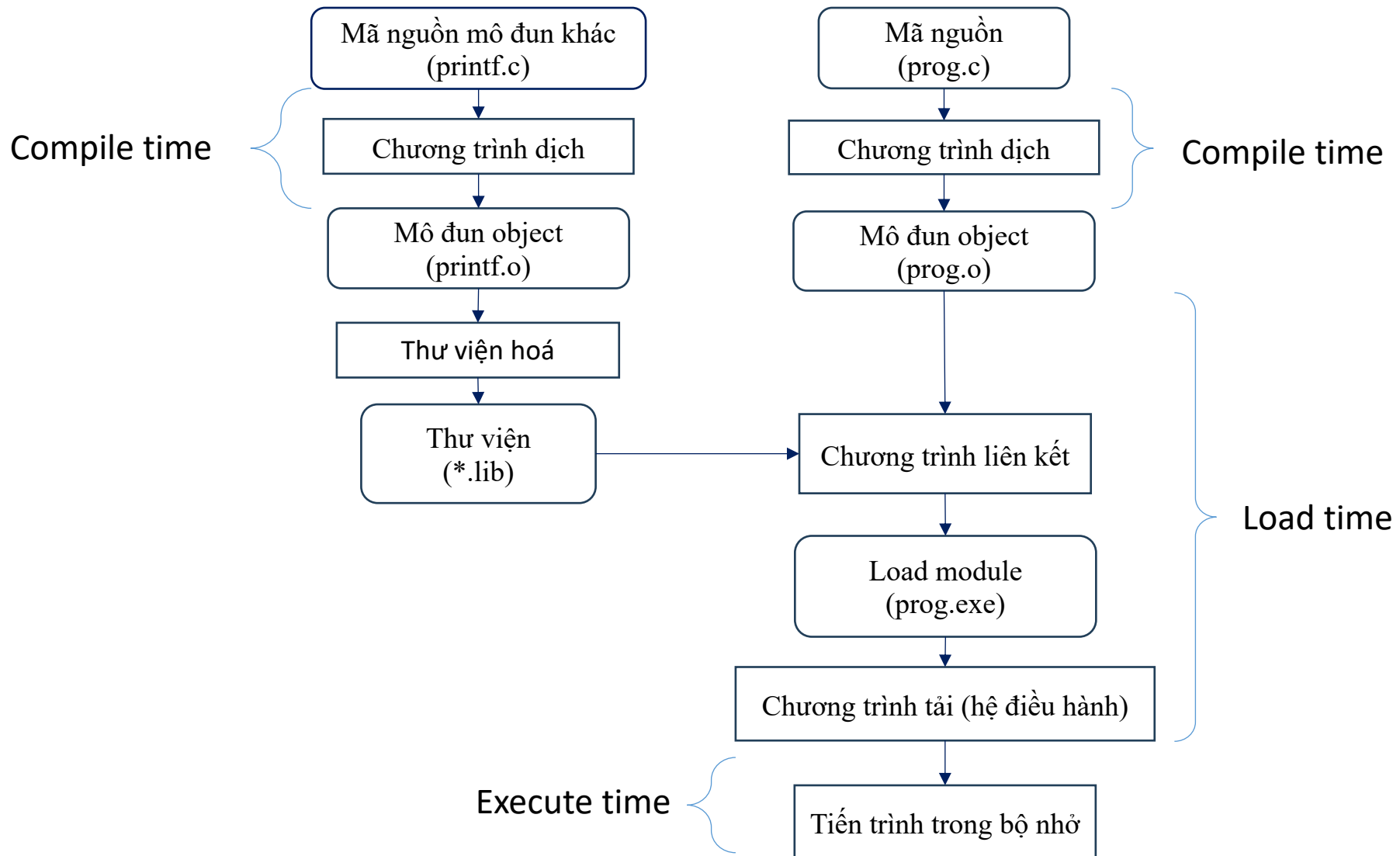


# 1. Địa chỉ và các vấn đề liên quan

## 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**
  - Load time
  - Execute time

# 1. Địa chỉ và các vấn đề liên quan

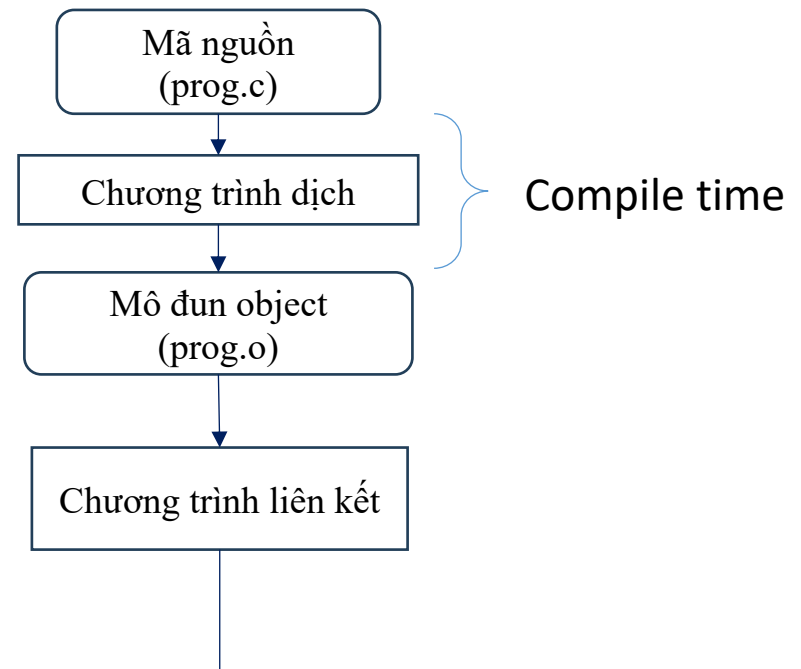


# 1. Địa chỉ và các vấn đề liên quan

## Chuyển đổi địa chỉ

- **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



# 1. 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.
  - **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  
=> phải tiến hành reload nếu địa chỉ nền thay đổi.

# 1. 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.
    - **Load time:** khi trong quá trình thực thi, process có thể được di chuyển từ segment này sang segment khác trong bộ nhớ thì quá trình chuyển đổi địa chỉ được trì hoãn đến thời điểm thực thi.
  - CPU tạo ra địa chỉ luận lý cho Tiến trình
  - 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, trong đó có các cơ chế swapping, paging, segmentation.

## 2. 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

## 2. Một số cách tổ chức chương trình

### Cấu trúc tuyến tính

- Tất cả các module, thư viện sử dụng trong một 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 lớn.

## 2. 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.



## 2. 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.
- ...
- Mức n: Chứa các module được gọi bởi mức n-1

## 2. Một số cách tổ chức chương trình

### Cấu trúc overlay:

- 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.

## 2. 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)

## 2. 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 các bộ điều khiển đoạn (**SCT** – Segment Control Table)

### 3. 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 1 Tiến trình.

- 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.

### 3. 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.

### 3. 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.

### 3. 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.



### 3. 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

[illegible]

### 3. 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

[illegible]



### 3. Phân chương bộ nhớ

- Phân vùng động:
  - Bản đồ bit

[illegible]



# 3. 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								

# 3. 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					

# 3. 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				



# 3. 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		

# 3. 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

### 3. Phân chương bộ nhớ

- Phân vùng động:

- Danh sách liên kết

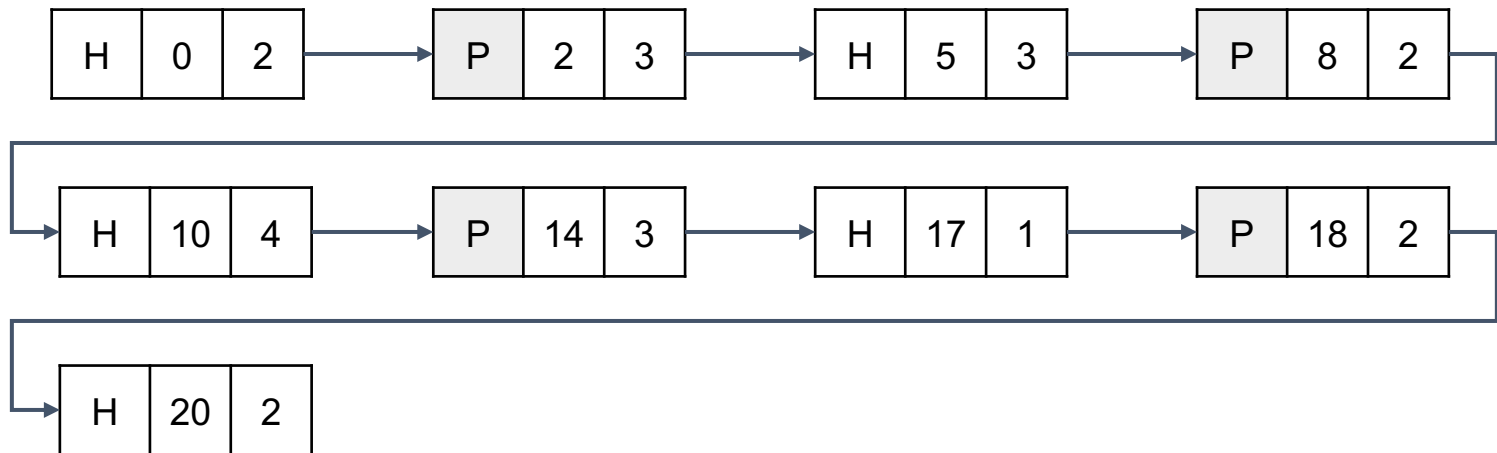
		<b>A</b>						<b>B</b>						<b>C</b>				<b>D</b>			
00	01	02	03	04	05	06	07	08	09	10	11	12	13	14	15	16	17	18	19	20	21

### 3. Phân chương bộ nhớ

- Phân vùng động:

- Danh sách liên kết

		<b>A</b>						<b>B</b>						<b>C</b>				<b>D</b>			
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.

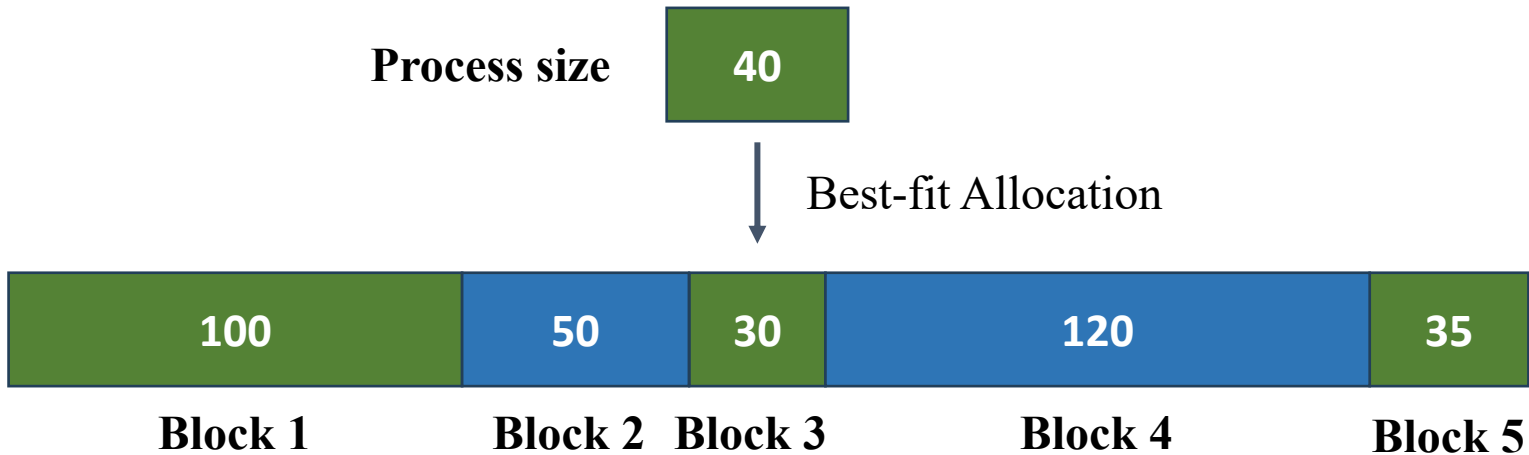
P	2	3
---	---	---

Không gian 3 khối đã cấp phát bắt đầu từ khối 2.

### 3. 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.

# 3. Phân chương bộ nhớ



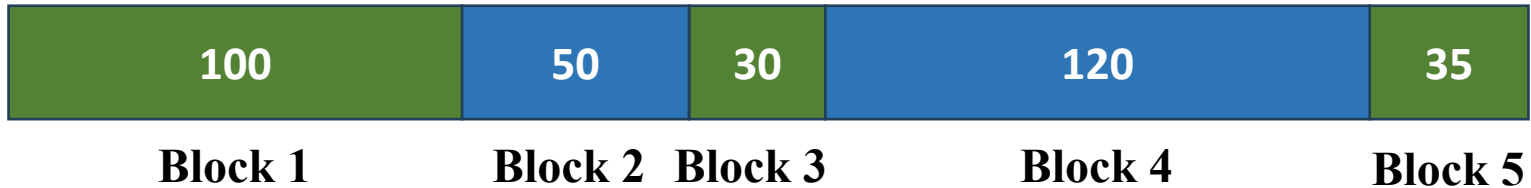
	Size	Can Occupy?	Memory Wastage After Process Occupies	
Block 1	100			
Block 2	50			
Block 3	30			
Block 4	120			
Block 5	35			

### 3. Phân chương bộ nhớ

Process size

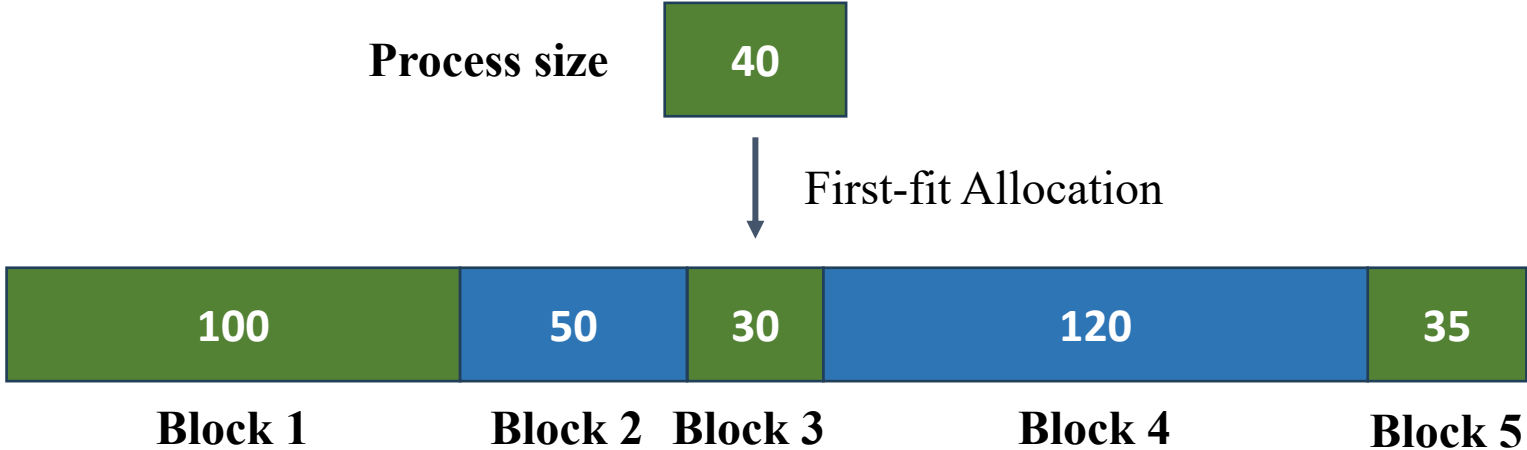
40

Best-fit Allocation



	Size	Can Occupy?	Memory Wastage After Process Occupies	
Block 1	100	Yes	$100 - 40 = 60$	
Block 2	50	Yes	$50 - 40 = 10$	Best
Block 3	30	No	-	
Block 4	120	Yes	$120 - 40 = 80$	
Block 5	35	No	-	

# 3. Phân chương bộ nhớ



	Size	Can Occupy?	Memory Wastage After Process Occupies	
Block 1	100			
Block 2	50			
Block 3	30			
Block 4	120			
Block 5	35			

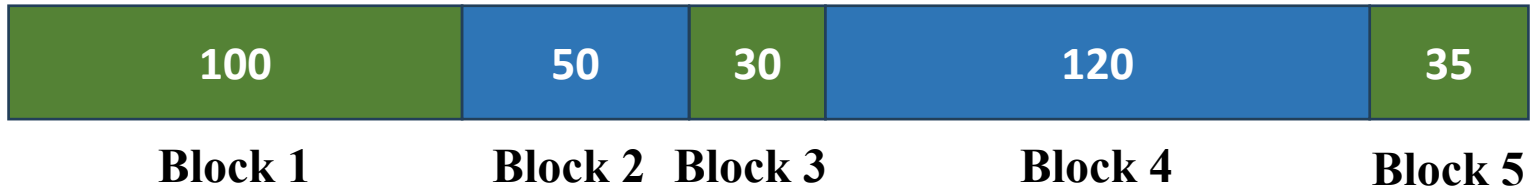


### 3. Phân chương bộ nhớ

Process size

40

First-fit Allocation



	Size	Can Occupy?	Memory Wastage After Process Occupies	
Block 1	100		$100 - 40 = 60$	First
Block 2	50		$50 - 40 = 10$	Next
Block 3	30		-	
Block 4	120		$120 - 40 = 80$	
Block 5	35		-	

# 3. Phân chương bộ nhớ

Process size



Worst-fit Allocation



Block 1

Block 2   Block 3

Block 4

Block 5

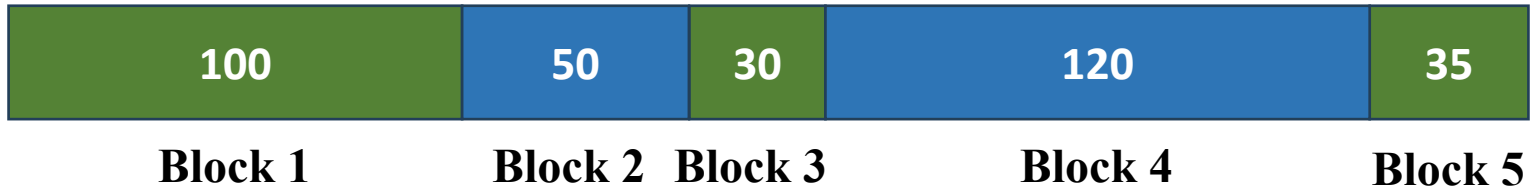
	Size	Can Occupy?	Memory Wastage After Process Occupies	
Block 1	100			
Block 2	50			
Block 3	30			
Block 4	120			
Block 5	35			

### 3. Phân chương bộ nhớ

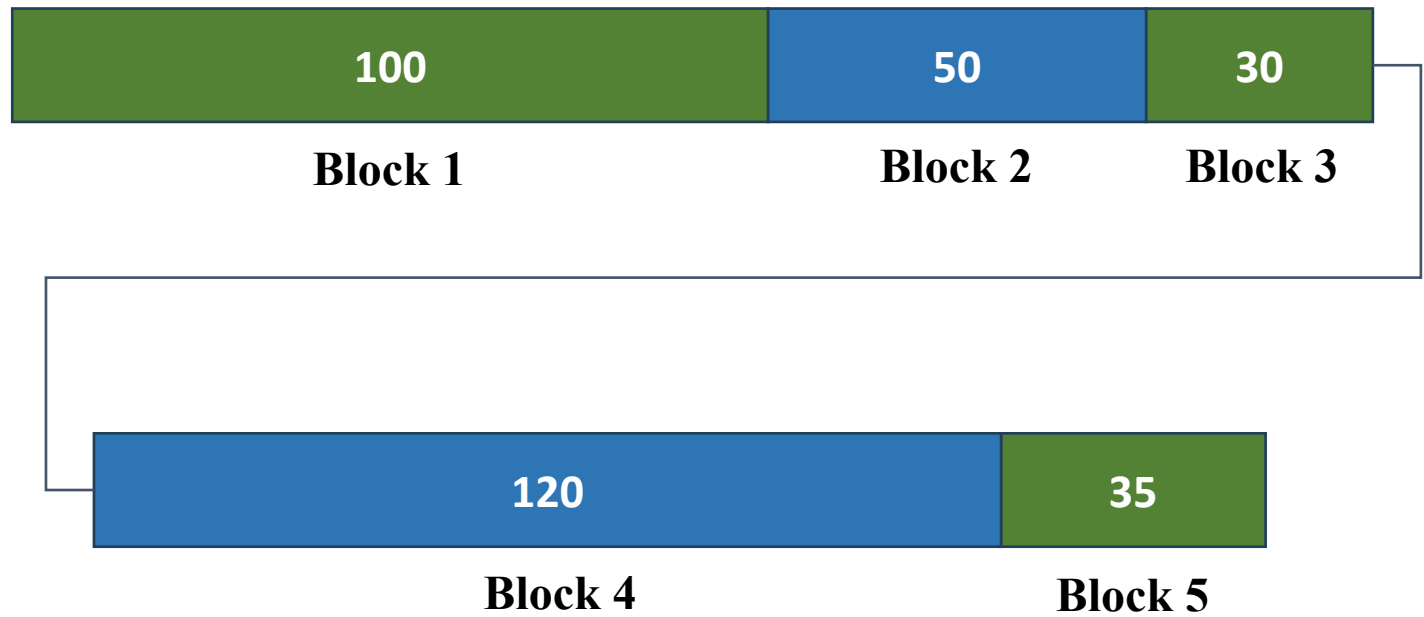
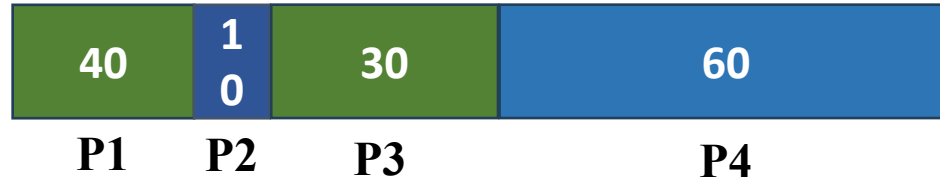
Process size

40

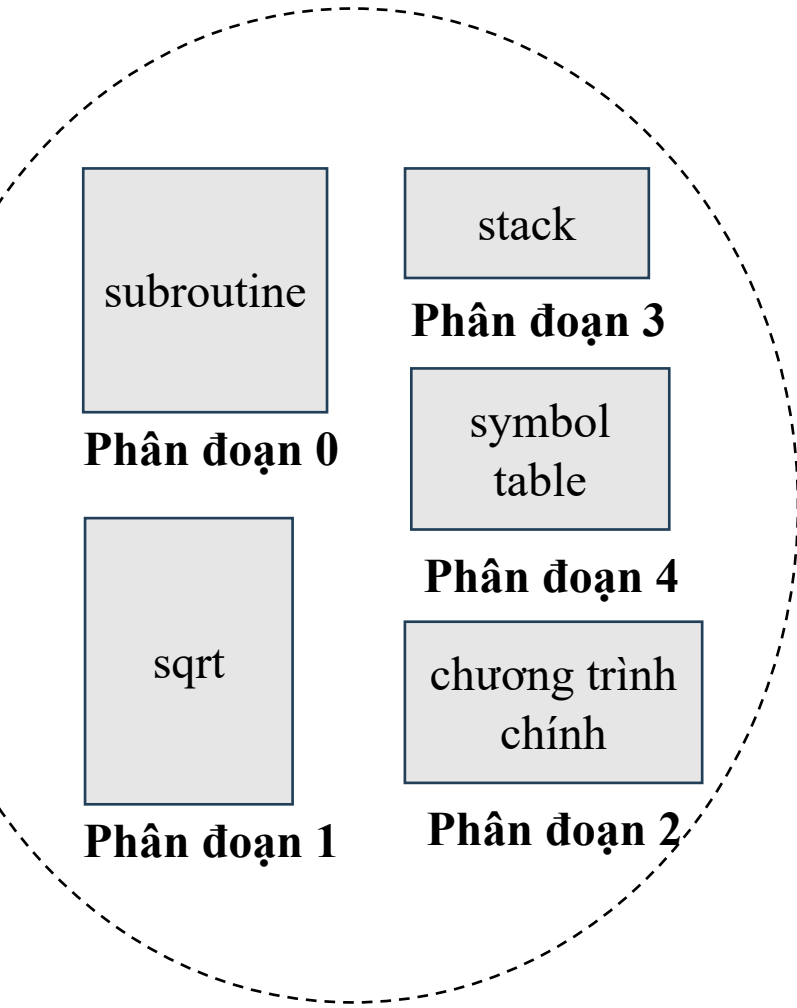
Worst-fit Allocation



	Size	Can Occupy?	Memory Wastage After Process Occupies	
Block 1	100	Yes	$100 - 40 = 60$	
Block 2	50	Yes	$50 - 40 = 10$	
Block 3	30	No	-	
Block 4	120	Yes	$120 - 40 = 80$	Worst
Block 5	35	No	-	

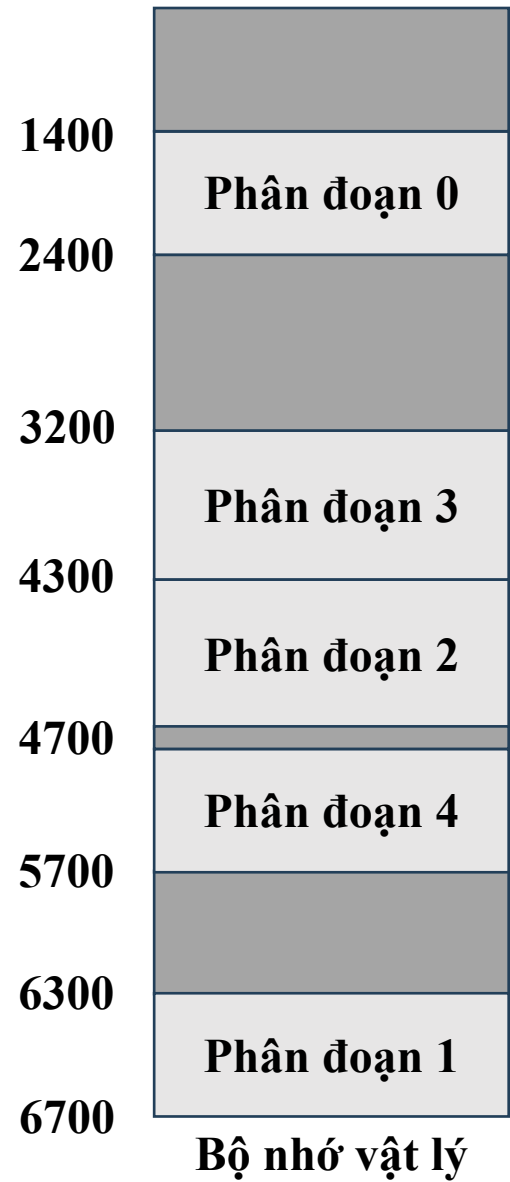


# 4. Phân đoạn bộ nhớ



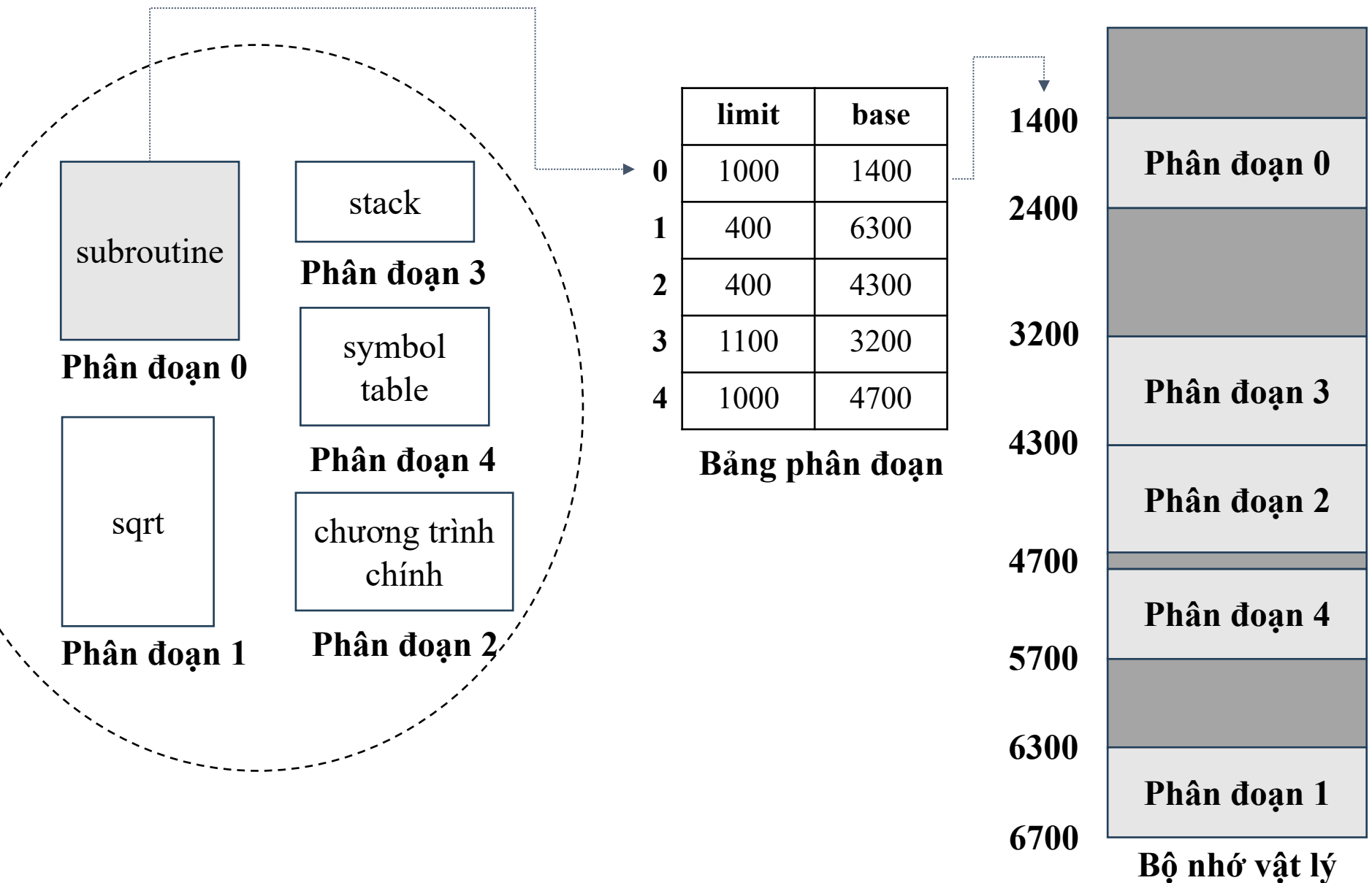
	limit	base
0	1000	1400
1	400	6300
2	400	4300
3	1100	3200
4	1000	4700

Bảng phân đoạn

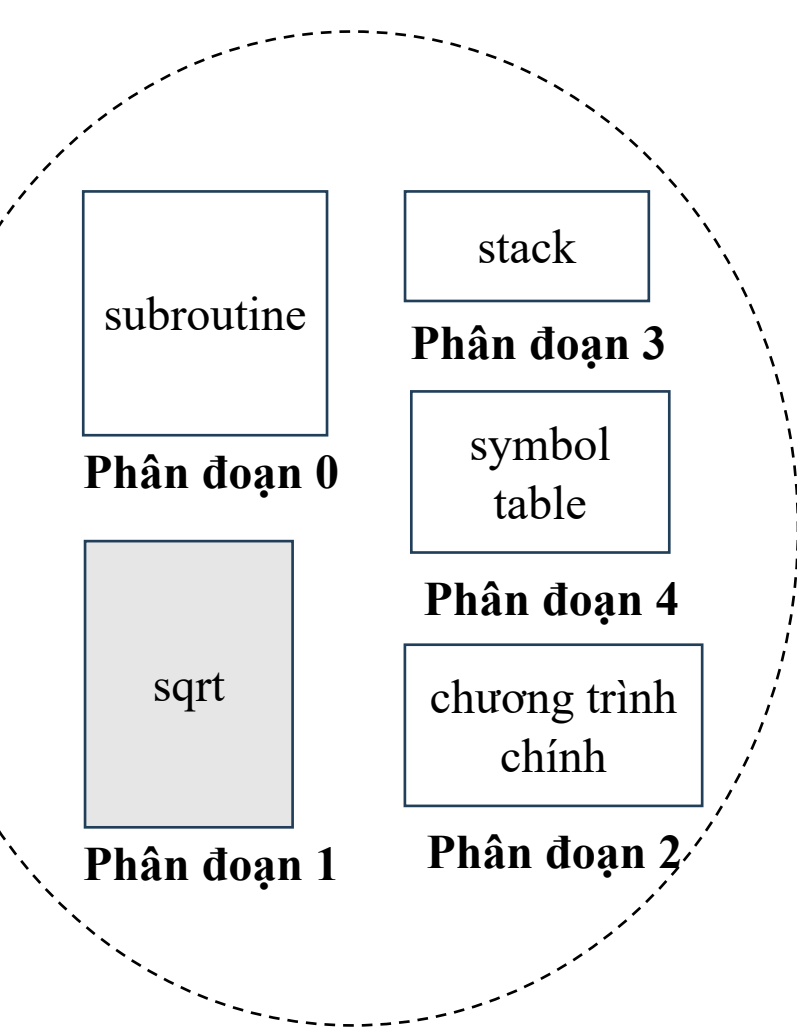


Bộ nhớ vật lý

## 4. Phân đoạn bộ nhớ

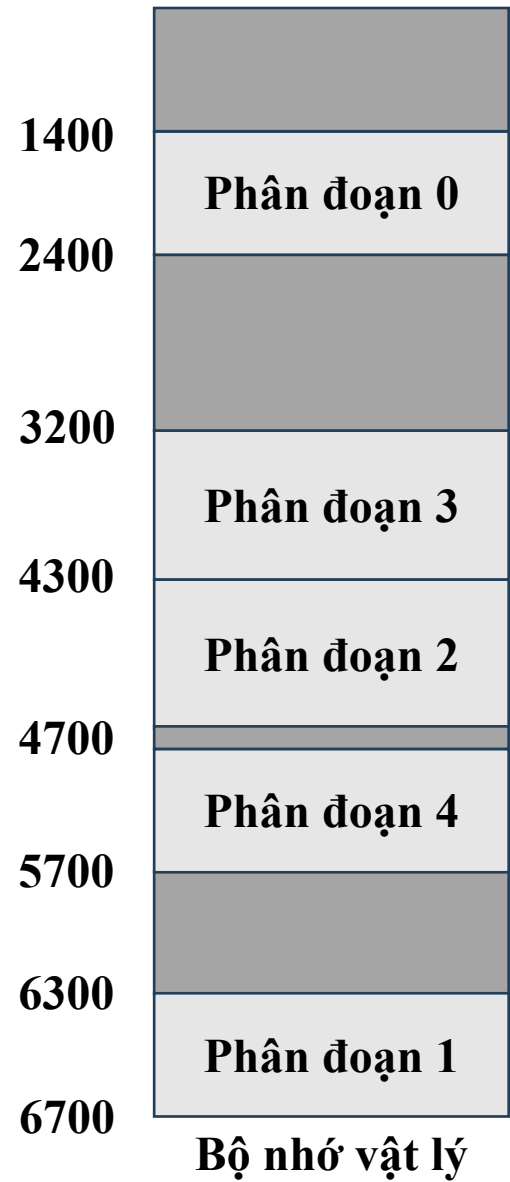


# 4. Phân đoạn bộ nhớ

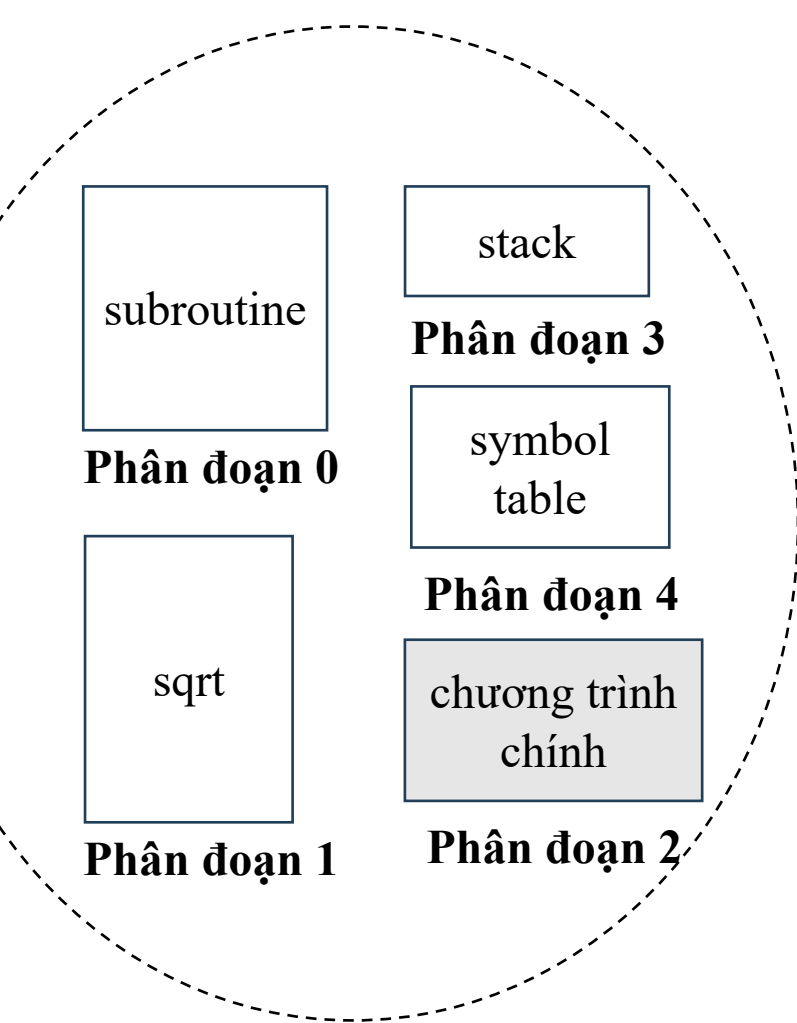


	limit	base
0	1000	1400
1	400	6300
2	400	4300
3	1100	3200
4	1000	4700

Bảng phân đoạn

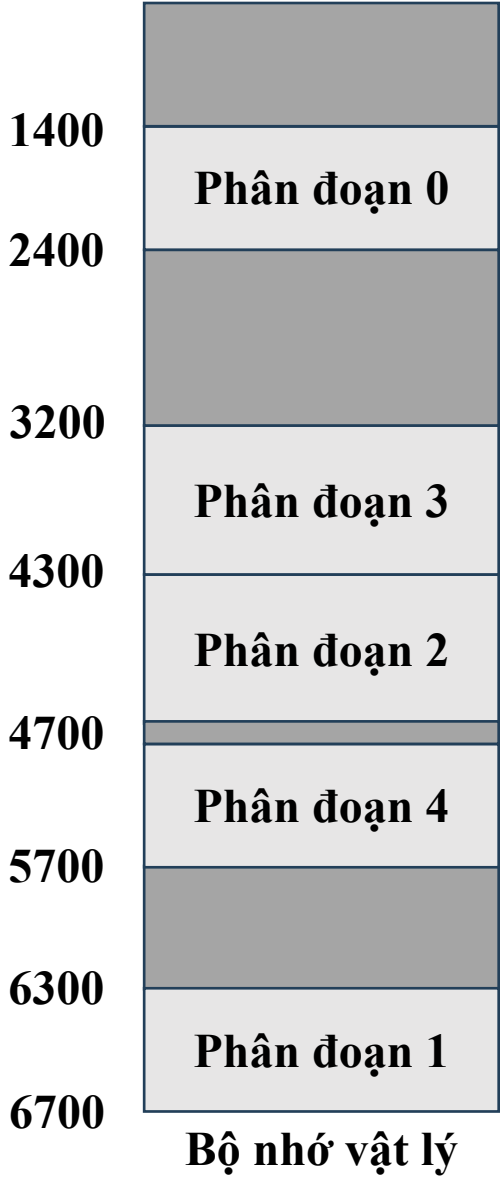


# 4. Phân đoạn bộ nhớ



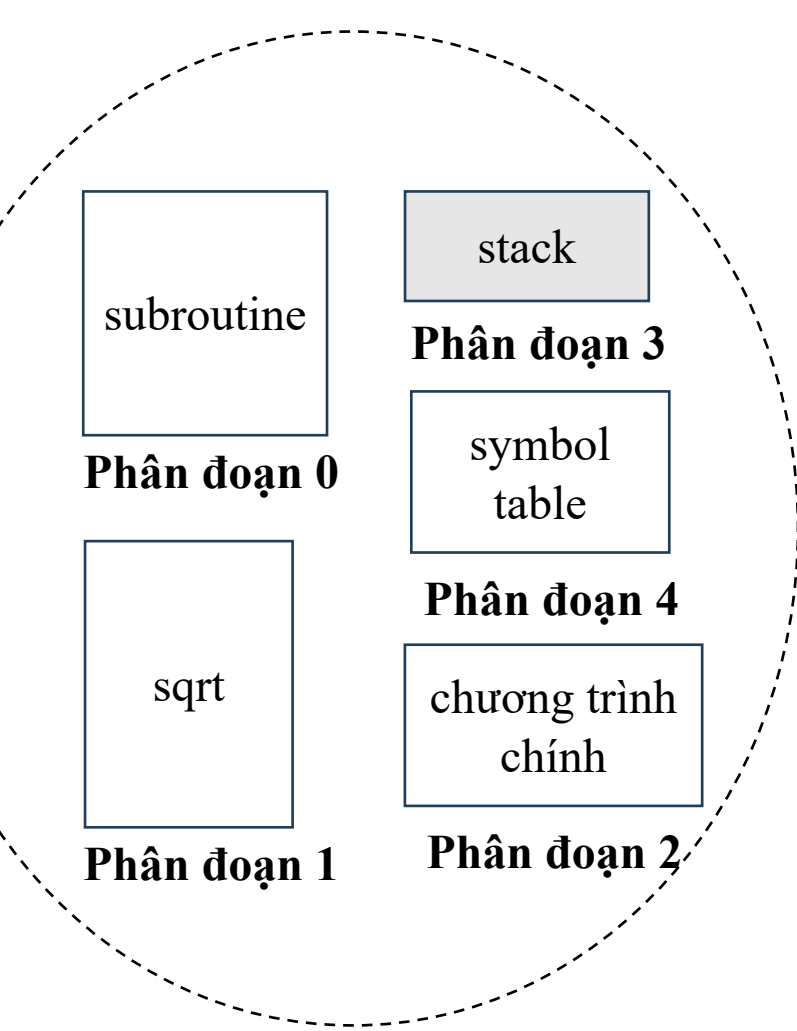
	limit	base
0	1000	1400
1	400	6300
2	400	4300
3	1100	3200
4	1000	4700

Bảng phân đoạn



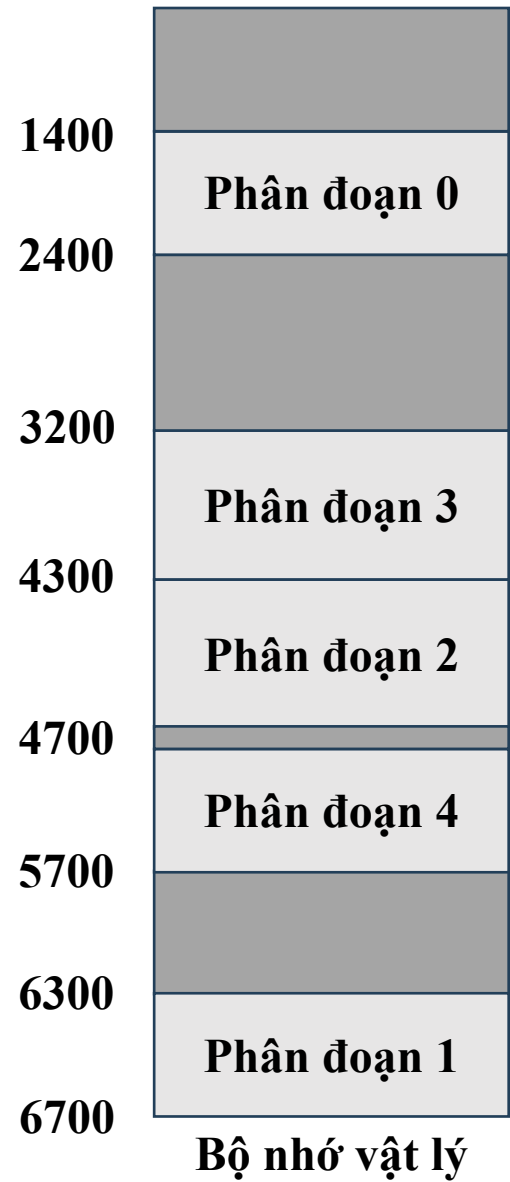


# 4. Phân đoạn bộ nhớ

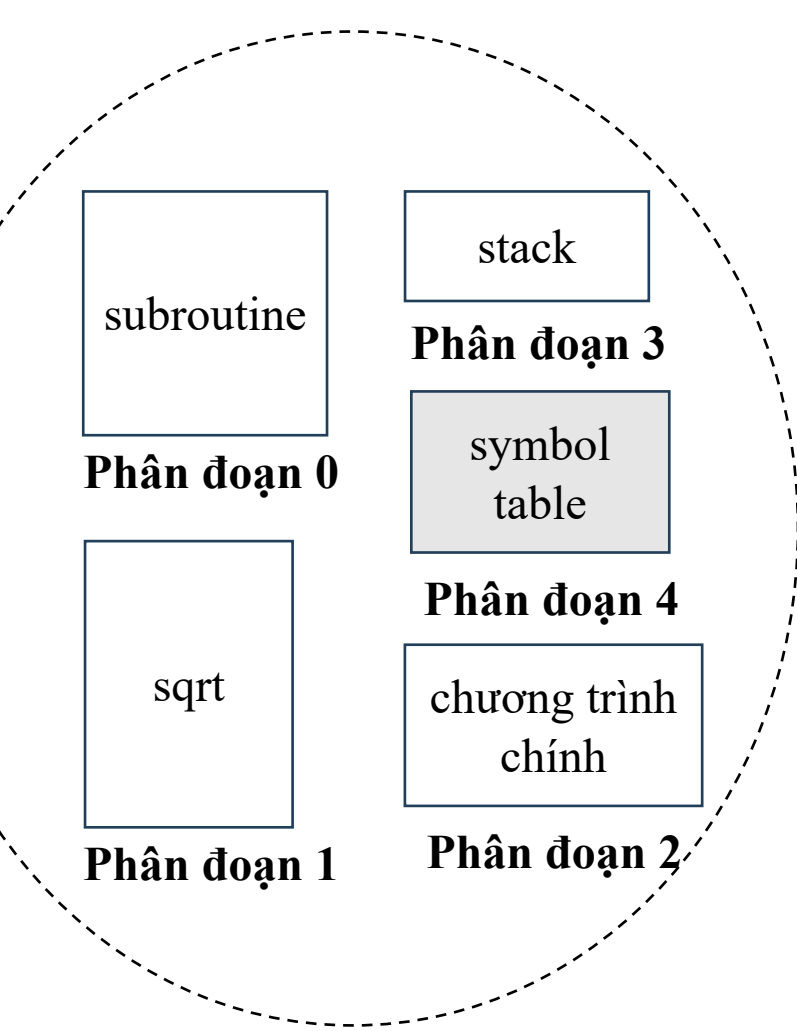


	limit	base
0	1000	1400
1	400	6300
2	400	4300
3	1100	3200
4	1000	4700

Bảng phân đoạn

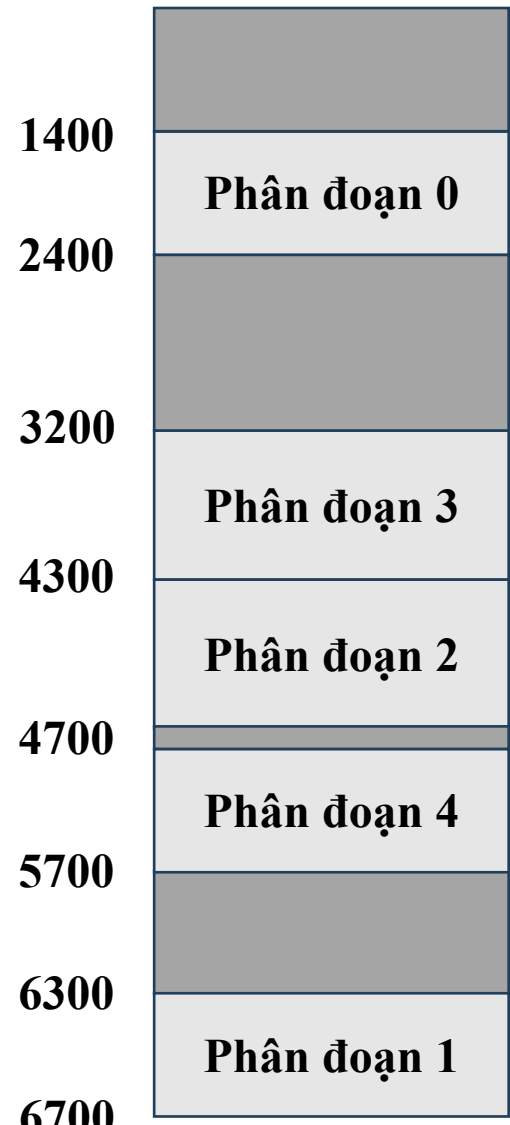


# 4. Phân đoạn bộ nhớ



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

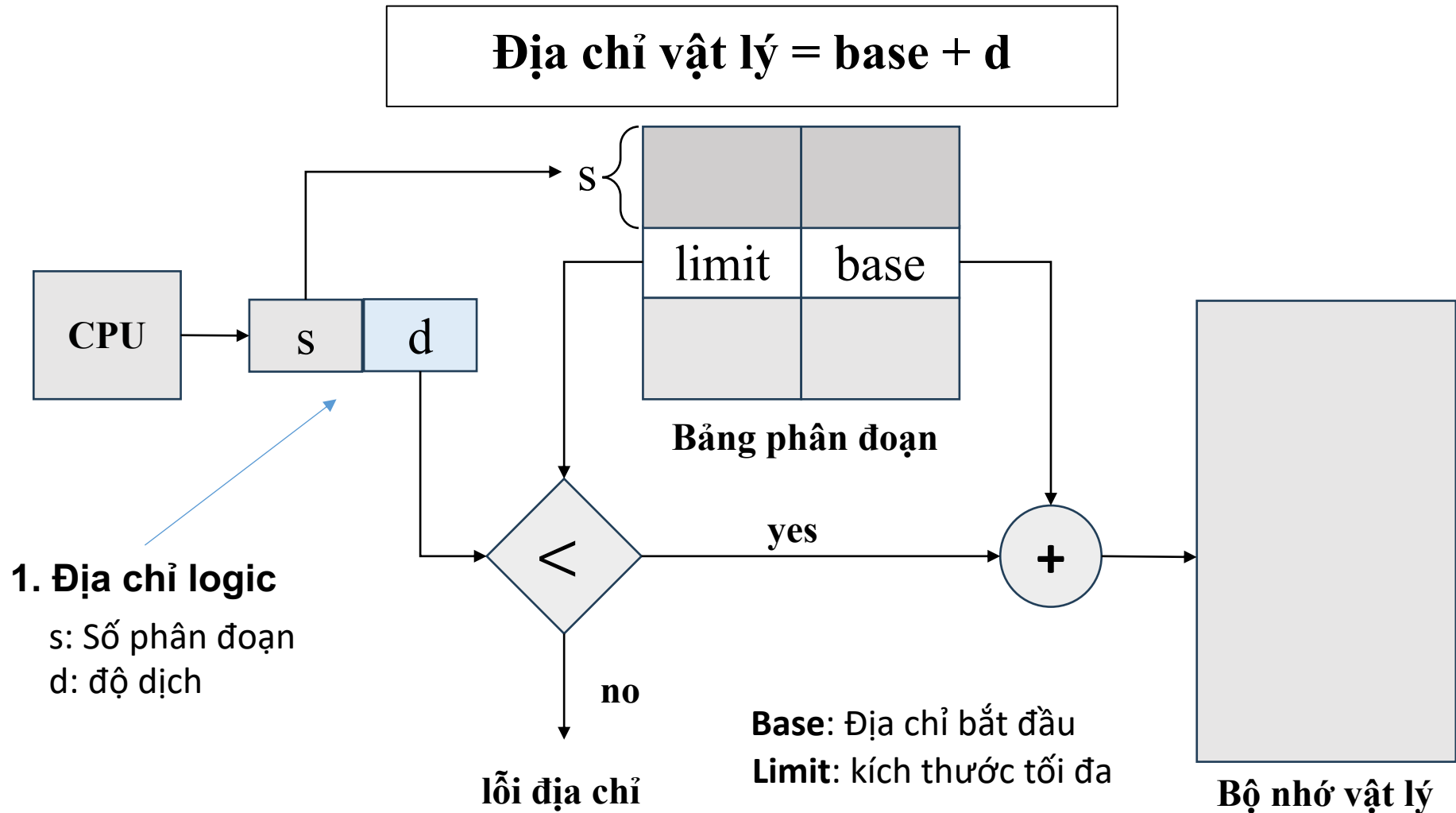
Bảng phân đoạn



Bộ nhớ vật lý

## 4. Phân đoạn bộ nhớ

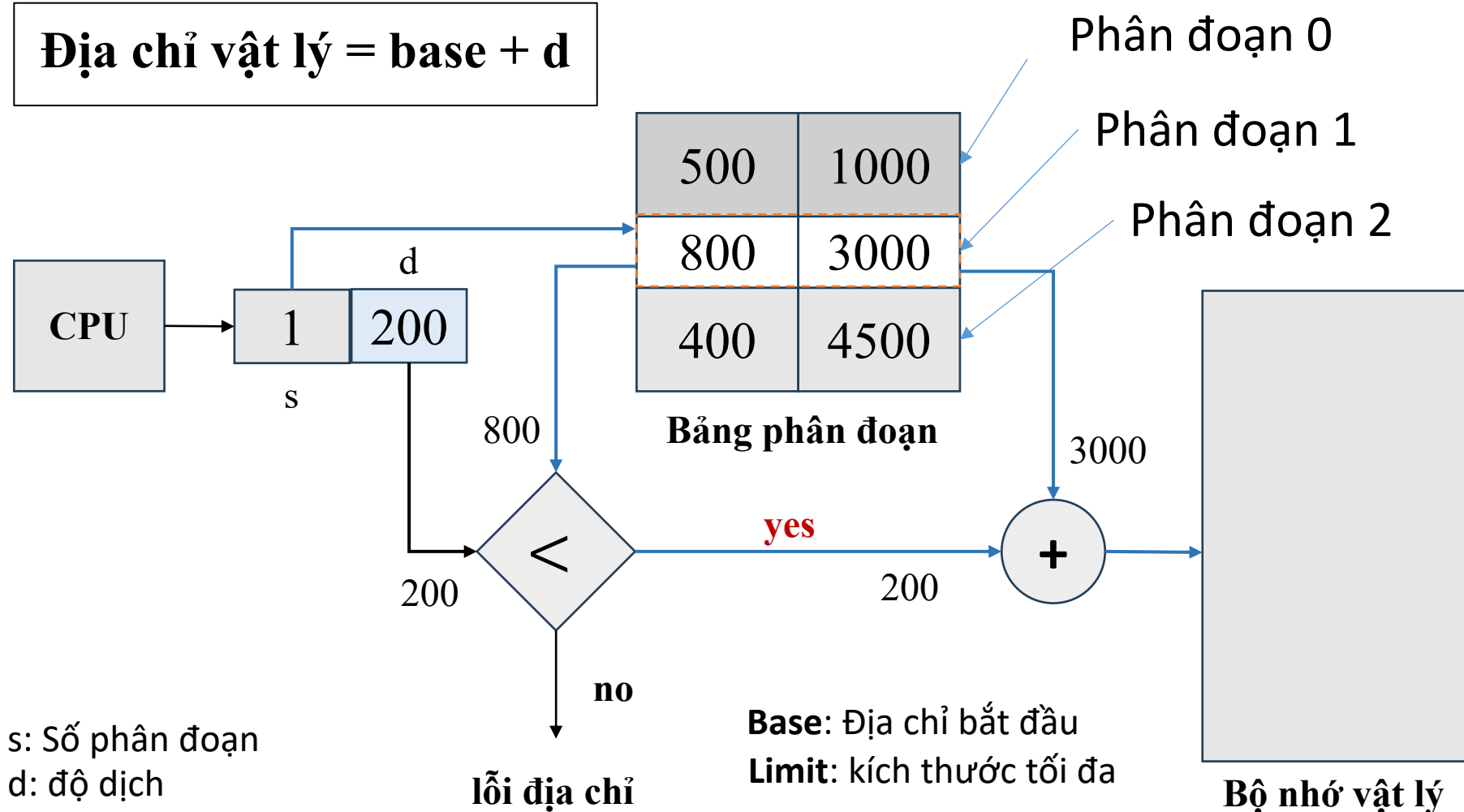
### Cơ chế MMU (Memory Management Unit) trong kỹ thuật phân đoạn



## 4. Phân đoạn bộ nhớ

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

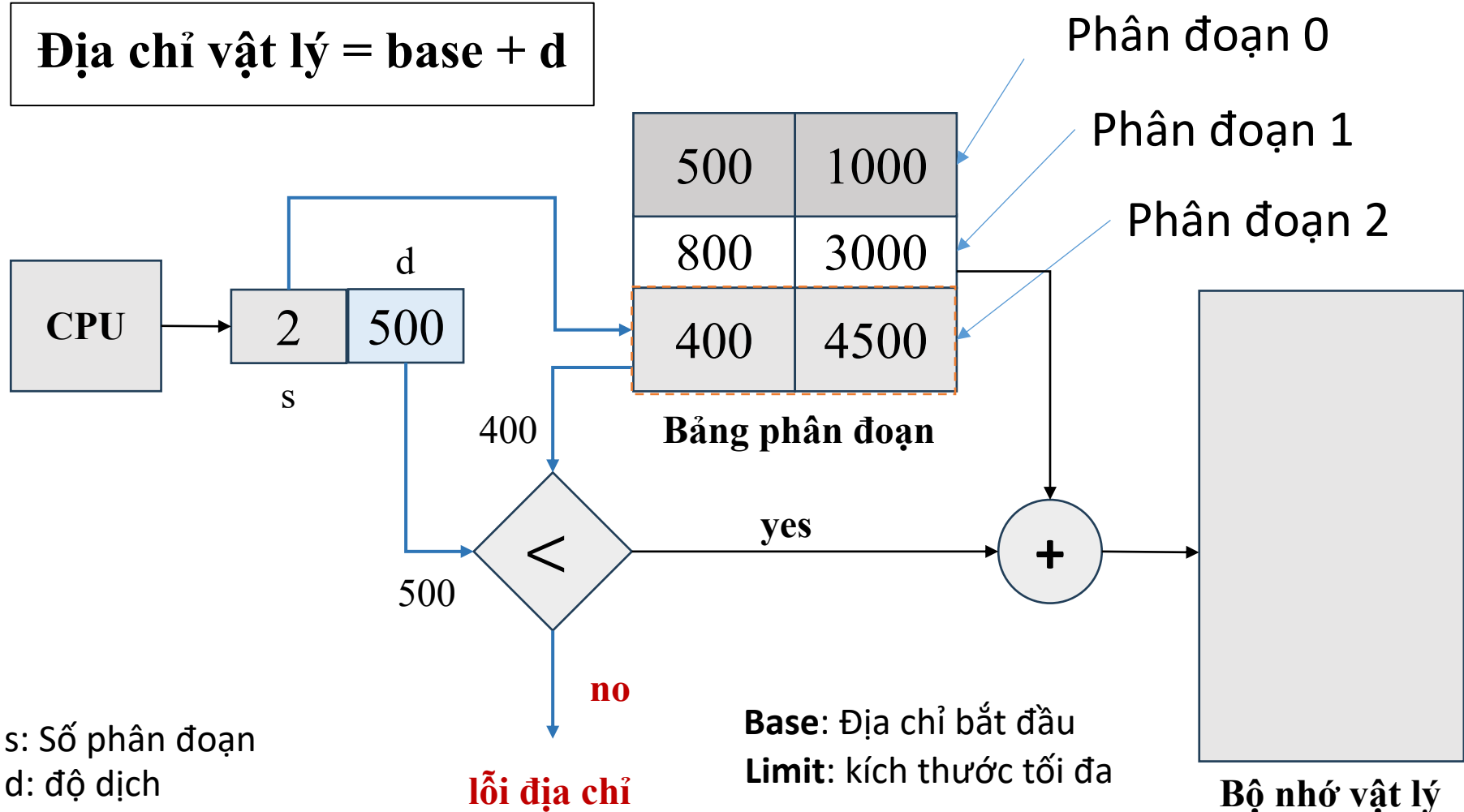
$$\text{Địa chỉ vật lý} = \text{base} + d$$



## 4. Phân đoạn bộ nhớ

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

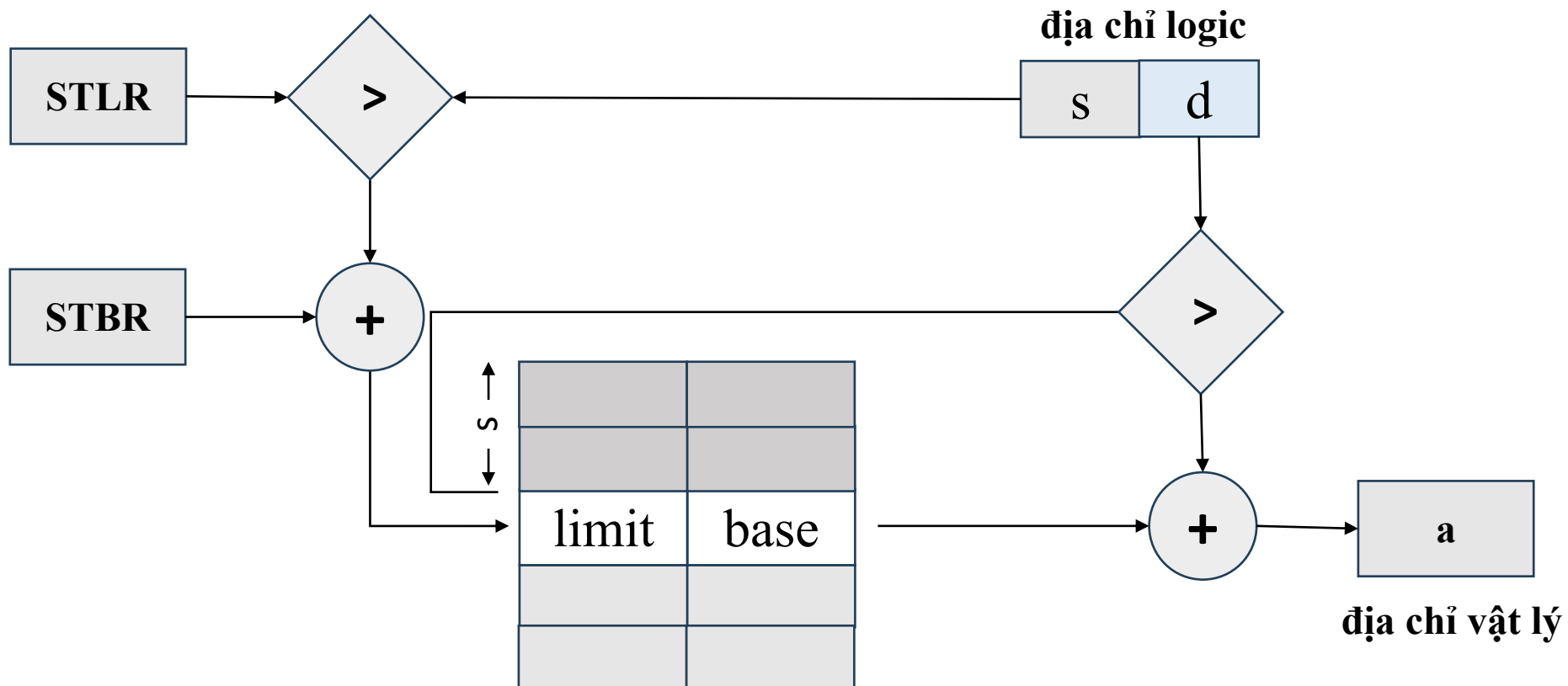
$$\text{Địa chỉ vật lý} = \text{base} + d$$



## 4. Phân đoạn bộ nhớ

Cài đặt bảng phân đoạn

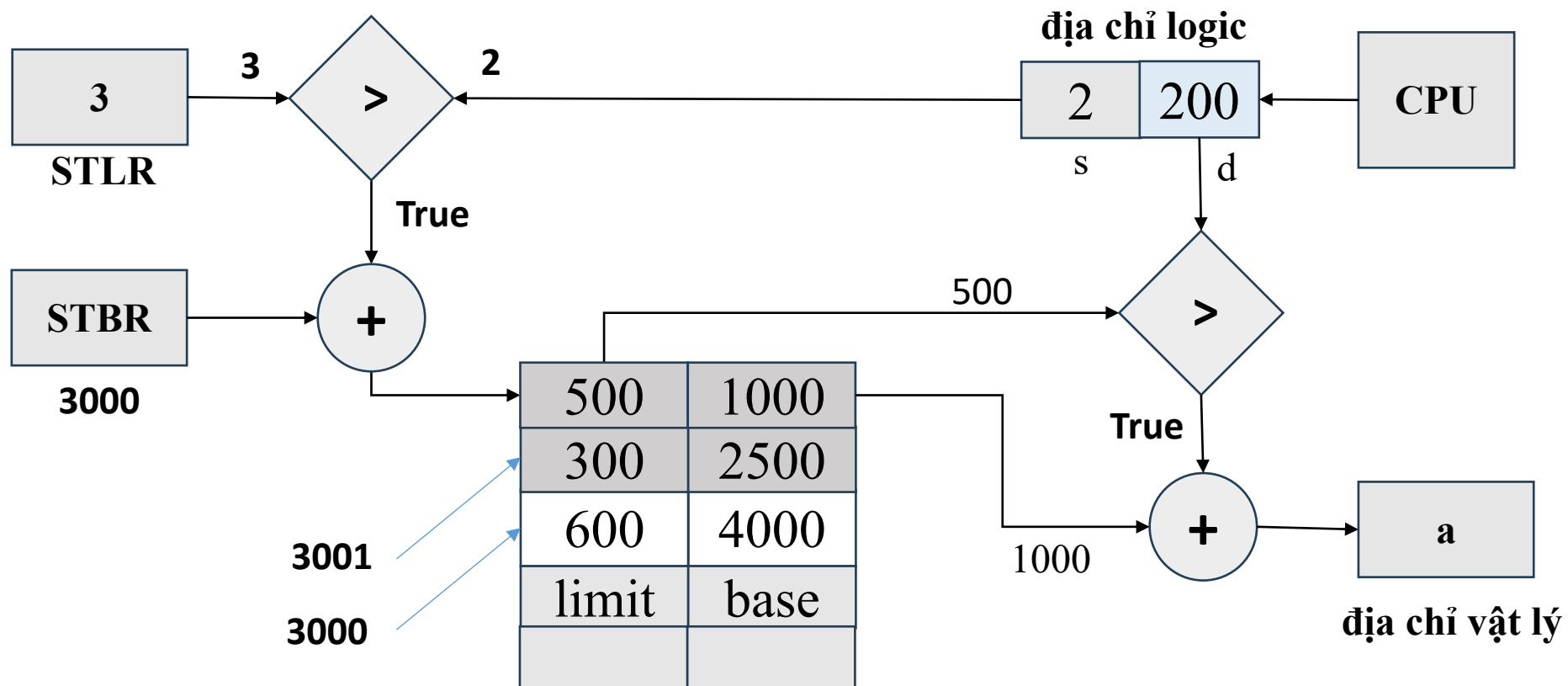
- **STBR (Segment Table Base Register)** để lưu địa chỉ bắt đầu của bảng phân đoạn.
- **STLR (Segment Table Limit Register)** lưu số phân đoạn.



## 4. Phân đoạn bộ nhớ

- Cài đặt bảng phân đoạn

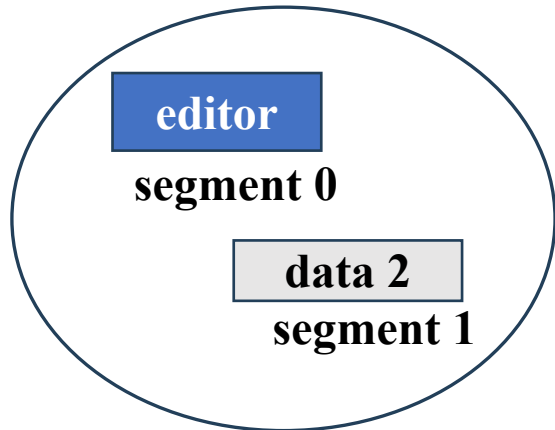
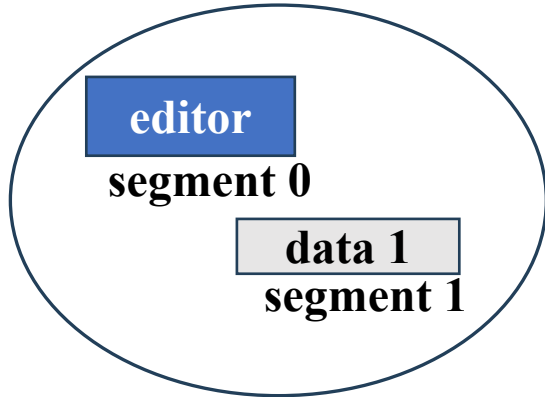
- **STBR (Segment Table Base Register)** để lưu địa chỉ **bắt đầu** của bảng phân đoạn.
- **STLR (Segment Table Limit Register)** lưu **số phân đoạn**.



# 4. Phân đoạn bộ nhớ

## Chia sẻ phân đoạn

Logical Memory Process  $P_1$



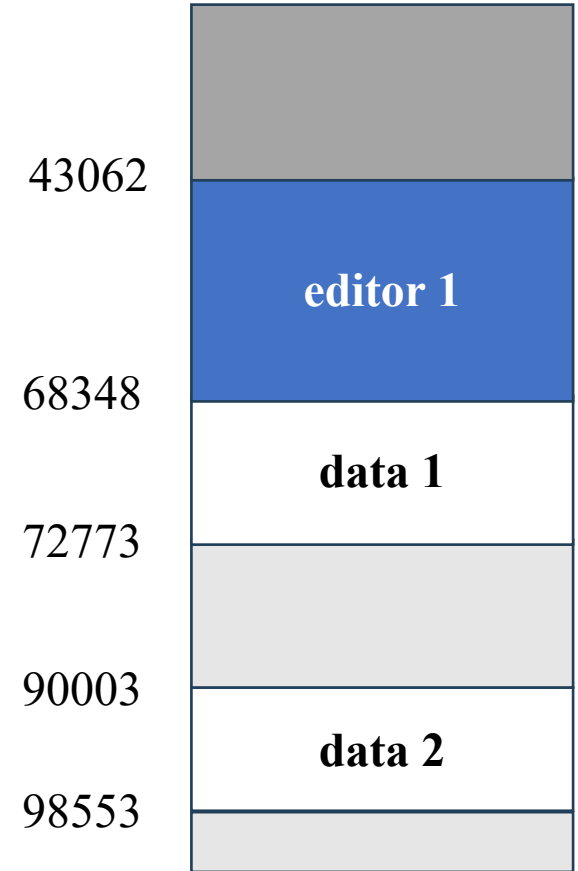
Logical Memory Process  $P_2$

	limit	base
0	25286	43062
1	4425	68348

Bảng phân đoạn P1

	limit	base
0	25286	43062
1	8850	90003

Bảng phân đoạn P2



Không gian vật lý

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



## 4. Phân đoạn bộ nhớ

- **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

## 5. Phân trang bộ nhớ (paging)

### ▪ Phân trang đơn:

- Bộ nhớ chính được chia thành các **phần bằng nhau** và cố định, 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 của 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.

# 5. Phân trang bộ nhớ (paging)

## Không gian địa chỉ ảo

60K - 64K	X	} TRANG
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	

## Không gian địa chỉ vật lý

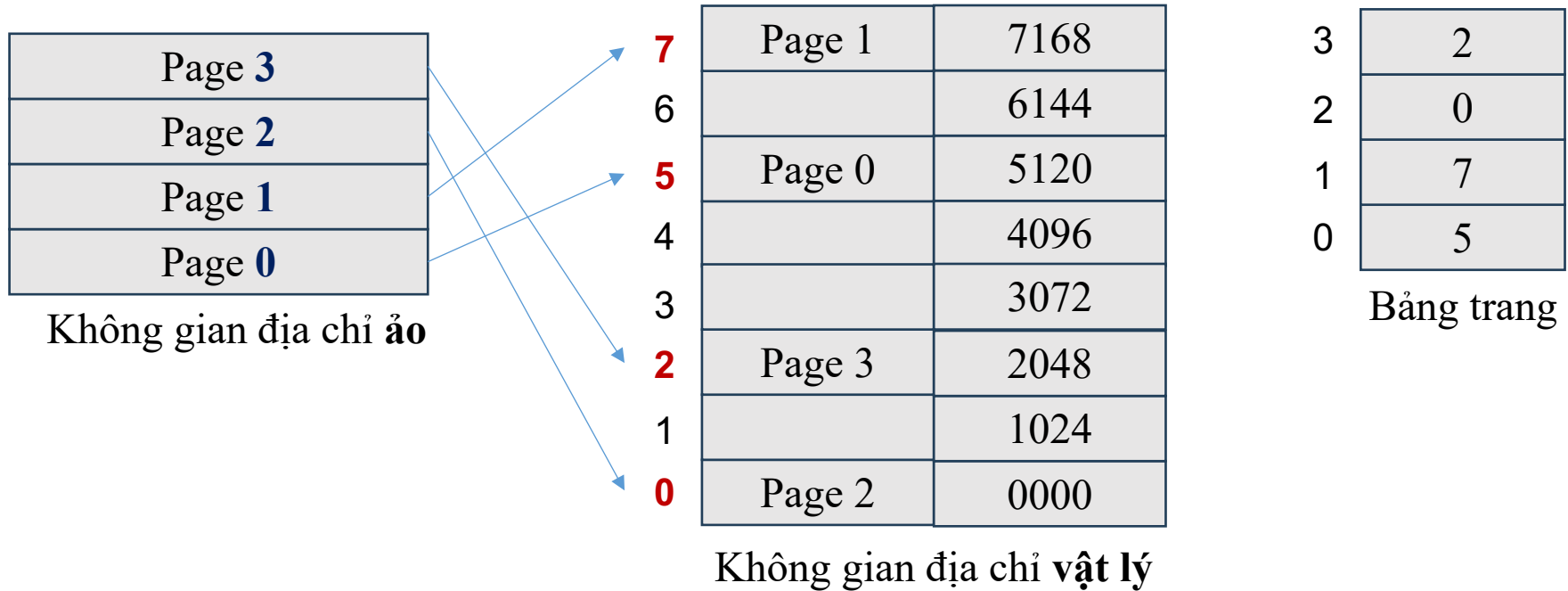
28K - 32K
24K - 28K
20K - 24K
16K - 20K
12K - 16K
8K - 12K
4K - 8K
0K - 4K

Khung trang

**Không gian địa chỉ ảo** là không gian bộ nhớ mà chương trình "nhìn thấy" và có thể sử dụng.

**Không gian địa chỉ vật lý** là bộ nhớ thật sự có trên hệ thống, thường giới hạn hơn so với không gian địa chỉ ảo.

## 5. Phân trang bộ nhớ (paging)



**Bảng trang** là cấu trúc dữ liệu lưu thông tin ánh xạ giữa **trang ảo** (page) và **khung trang vật lý** (frame).

# Mô hình phân trang (Paging)

## Không gian địa chỉ ảo

60K - 64K	X	} TRANG
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	

Page 3
Page 2
Page 1
Page 0

Không gian  
địa chỉ  
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

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

## 5. Phân trang bộ nhớ (paging)

### ▪ Cấu trúc địa chỉ ảo

- Kích thước của trang là lũy thừa của  $2^n$  ( $9 \leq n \leq 13$ )
  - 512 – 8192 bytes
- Kích thước của không gian địa chỉ ảo là  $2^m$  (CPU dùng địa chỉ ảo m bit)
  - Ví dụ: CPU dùng 23-bit biểu diễn địa chỉ ảo thì kích thước của địa chỉ ảo là  $2^{23}$  bytes = 8 Gigabytes
  - 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 <b>m-n</b> bit cao	d là địa chỉ tương đối trong trang và chiếm <b>n</b> bit thấp

## 5. Phân trang bộ nhớ (paging)

### ▪ Cấu trúc địa chỉ ảo

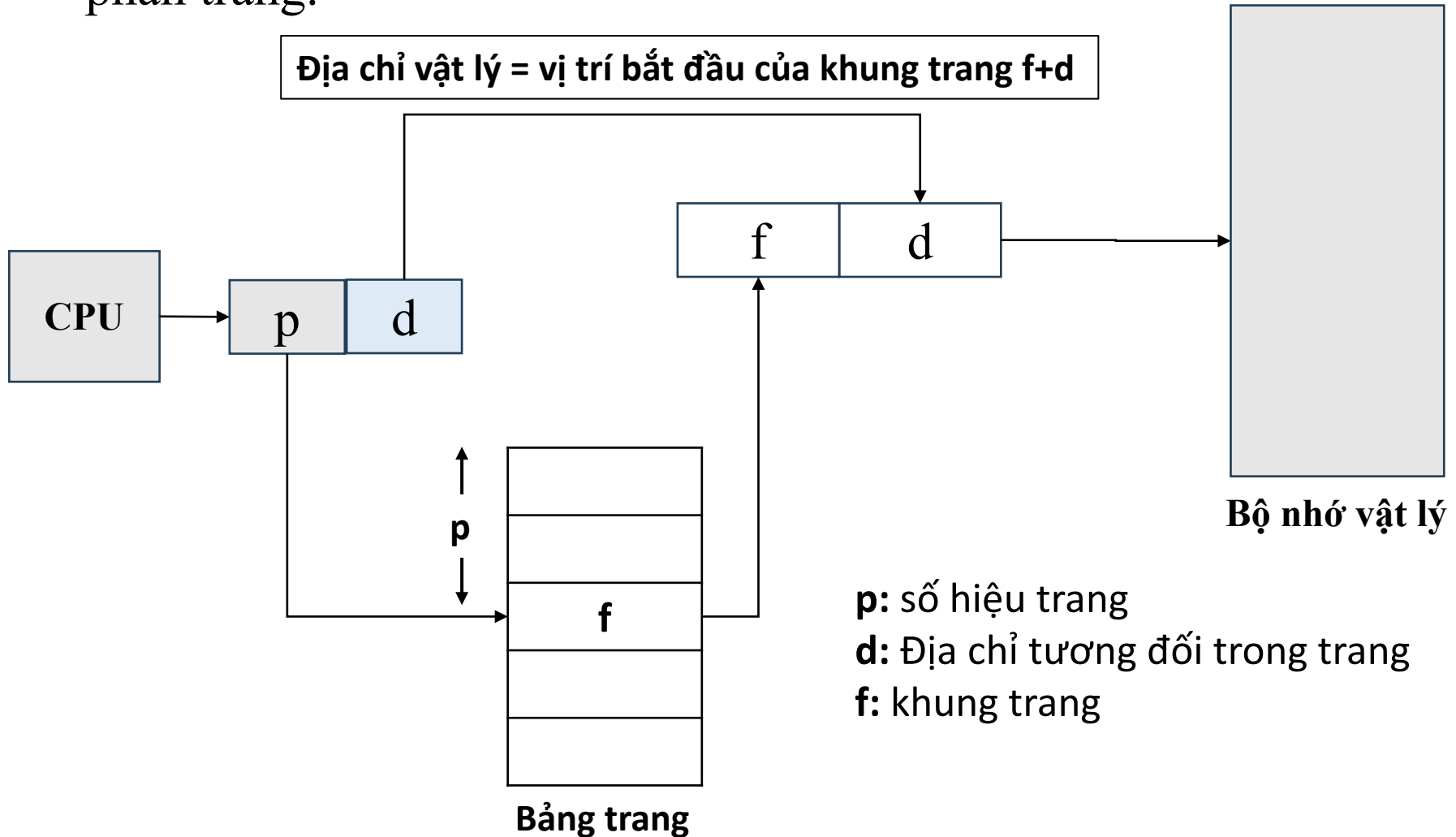
Địa chỉ ảo dạng (p, d) có m bit	
p là số hiệu trang và chiếm <b>m-n</b> bit cao	d là địa chỉ tương đối trong trang và chiếm <b>n</b> bit thấp

### ▪ Ví dụ:

- Kích thước của trang là 4 KB (tức là  $2^{12}$  byte)  $\rightarrow n = 12$  (vì  $2^{12} = 4096$  byte).
- Không gian địa chỉ ảo là 32-bit (tức là  $m = 32$ , vì CPU sử dụng địa chỉ ảo 32-bit).
- Số hiệu trang p:  $m - n = 32 - 12 = 20$  bit cao
- Địa chỉ tương đối trong trang d:  $n = 12$  bit thấp

## 5. Phân trang bộ nhớ (paging)

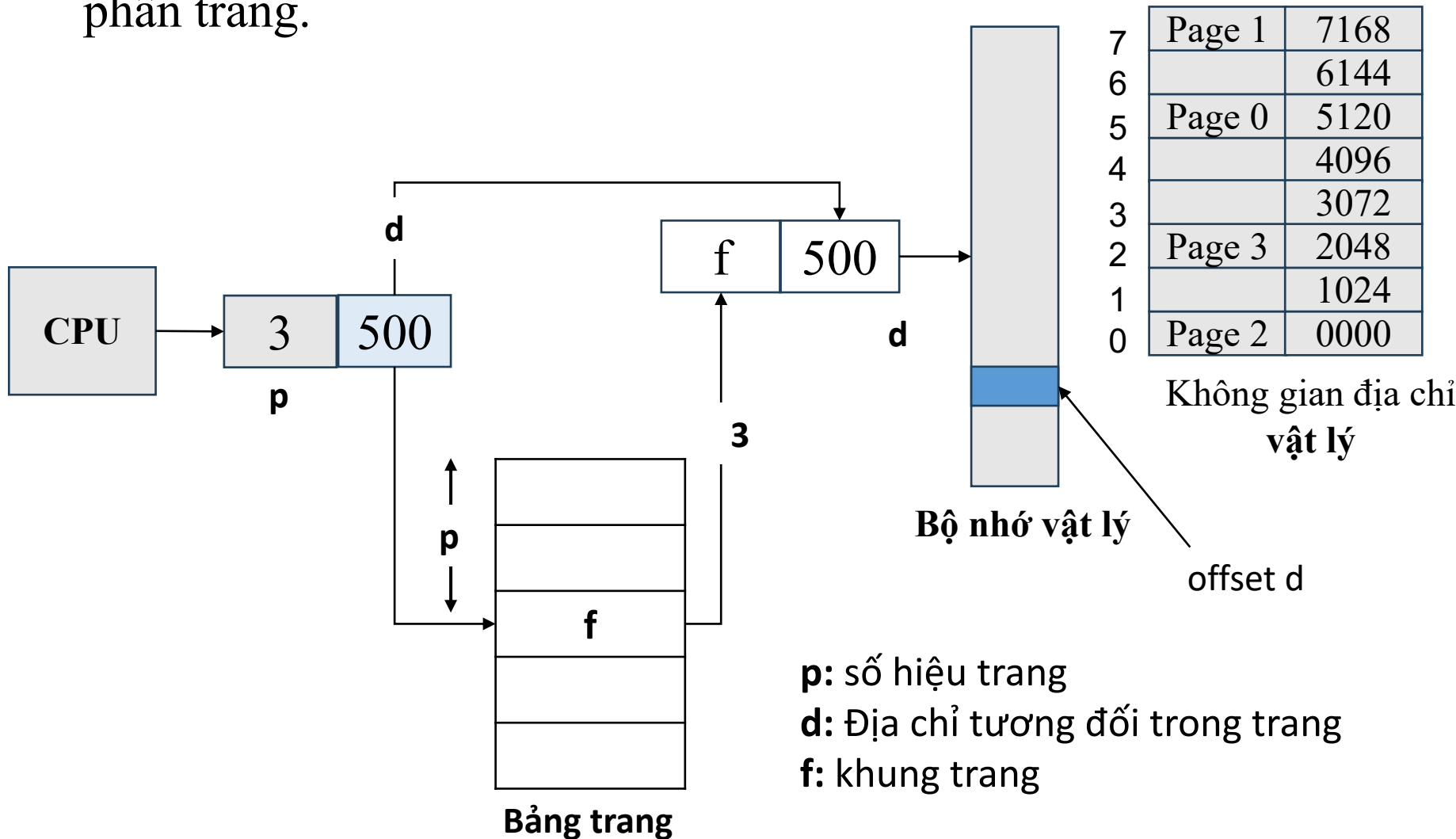
- Cơ chế MMU (Memory Management Unit) trong mô hình phân trang.





## 5. Phân trang bộ nhớ (paging)

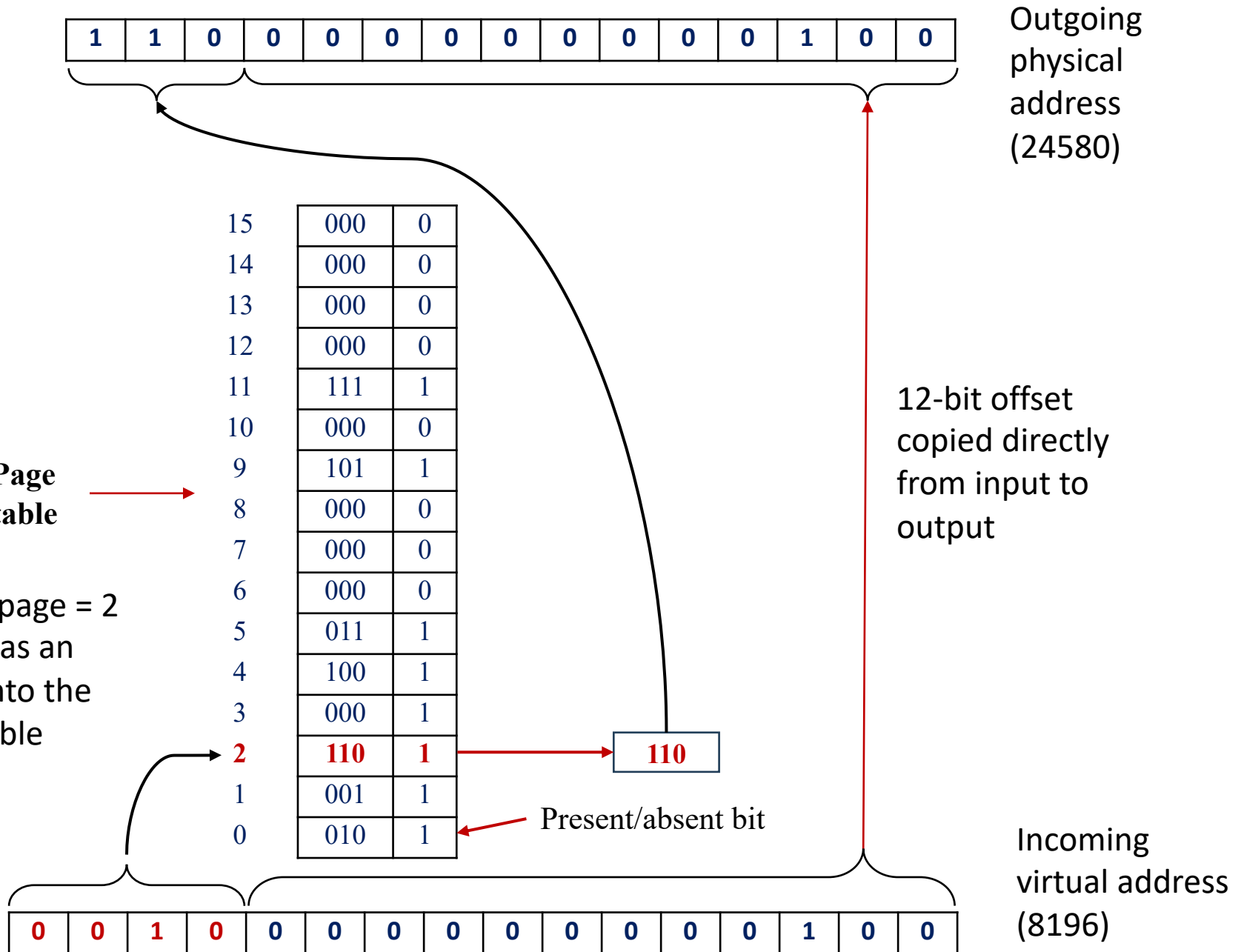
- Cơ chế MMU (Memory Management Unit) trong mô hình phân trang.



## 5. 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 4KB).
  - Bit Present/Absent: 0 / 1
    - 1 là trang hiện ở trong bộ nhớ
    - 0 là ở bộ nhớ phụ

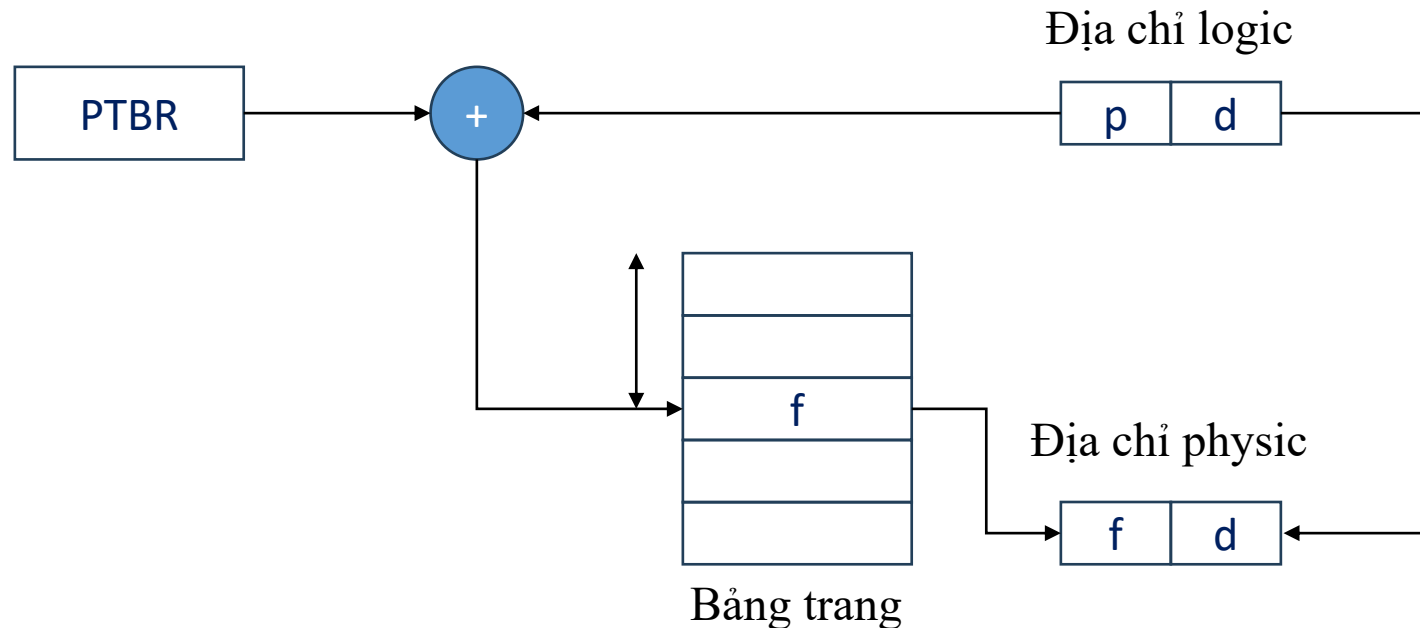
## 5. Phân trang bộ nhớ (paging)



## 5. Phân trang bộ nhớ (paging)

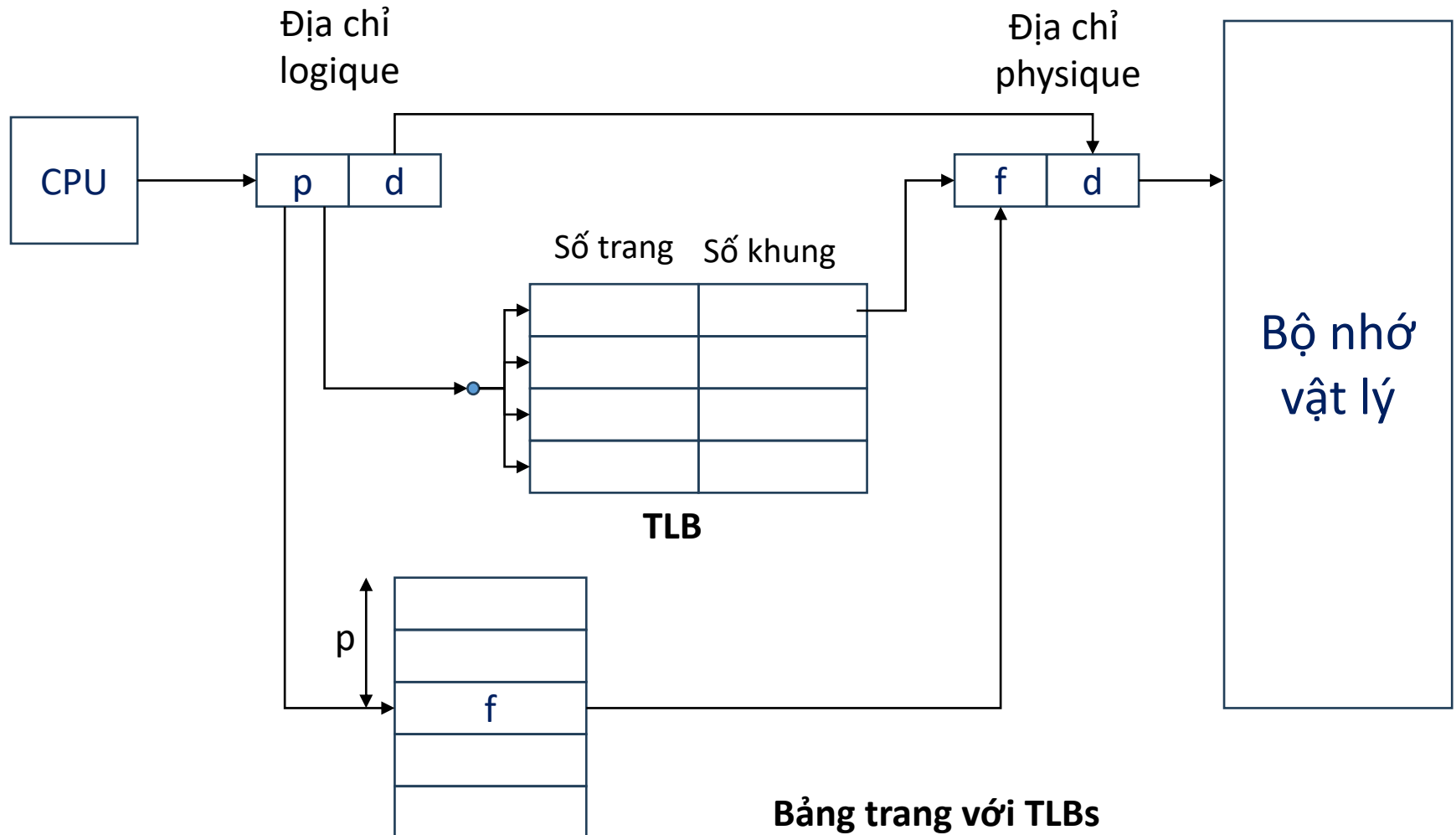
### ❖ Cài đặt bảng trang

- 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.



## 5. Phân trang bộ nhớ (paging)

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



## 5. 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 + với số bit của d là 32, mà 1 trang là  $4K = 2^{12}$  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 = 4GB.
- Nhận xét: **Máy tính n bit quản lý được tiến trình kích thước lớn nhất là  $2^n$  byte.**

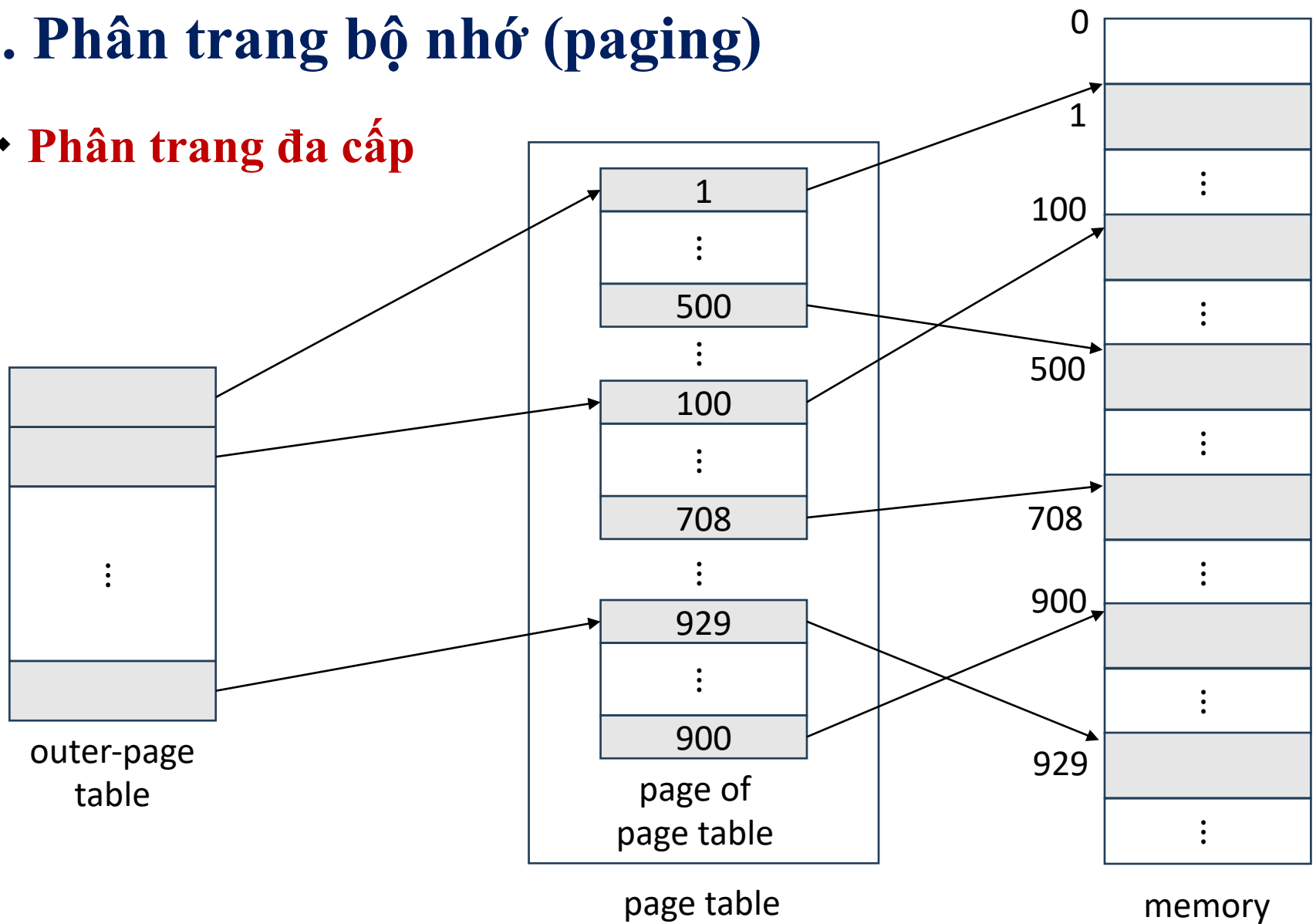
## 5. Phân trang bộ nhớ (paging)

### ❖ Tổ chức bảng phân trang

- Phân trang đa cấp
- Bảng trang băm
- Bảng trang nghịch đảo

## 5. Phân trang bộ nhớ (paging)

### ❖ Phân trang đa cấp



Bảng trang cấp 1

Bảng trang cấp 2

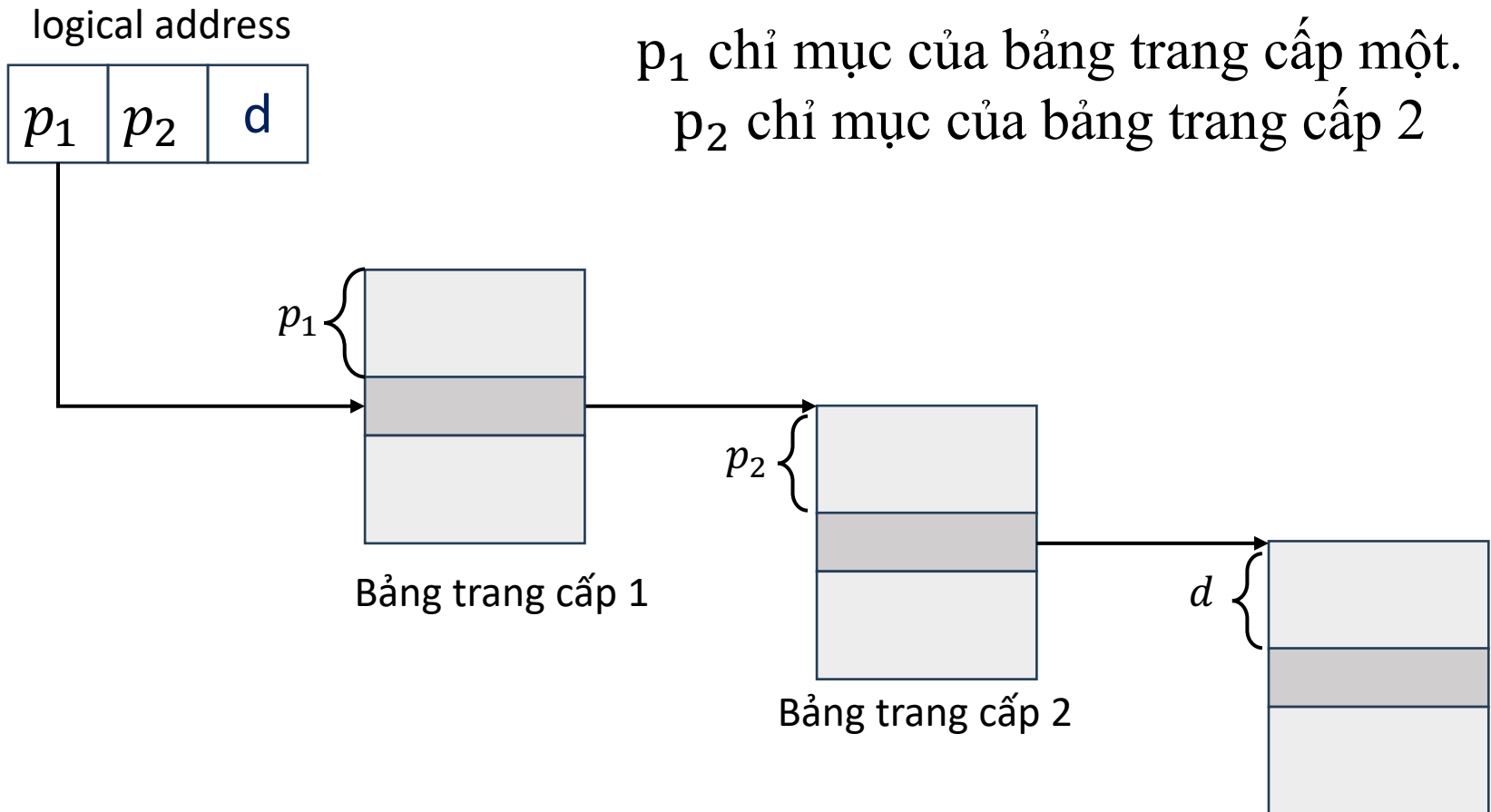


## 5. Phân trang bộ nhớ (paging)

### ❖ 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



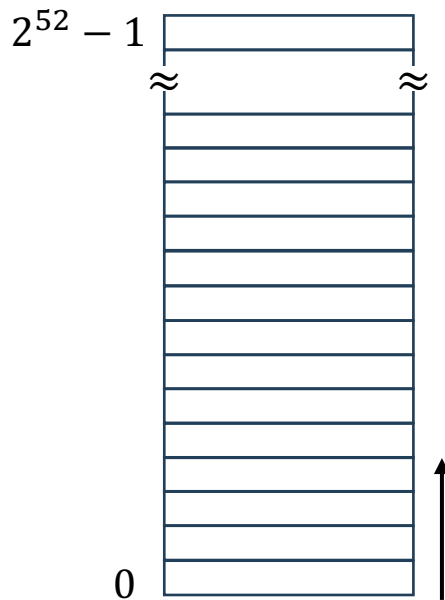
# 5. Phân trang bộ nhớ (paging)

## ❖ Bảng trang băm

- Khi không gian

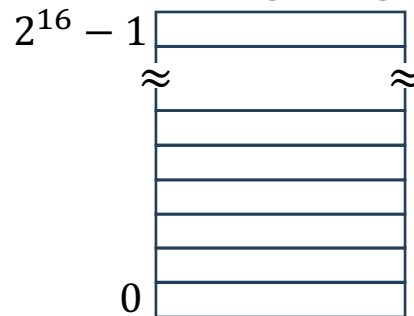
địa chỉ ảo > 32 bit

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

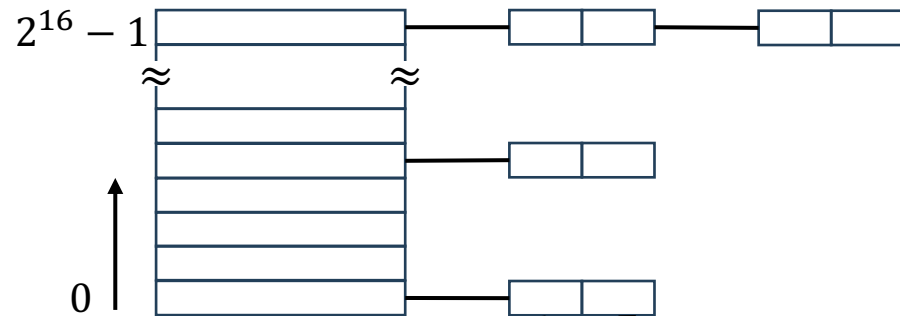


Được chỉ mục bằng trang ảo

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



Bảng băm



Được chỉ mục bằng phương pháp băm trên trang ảo

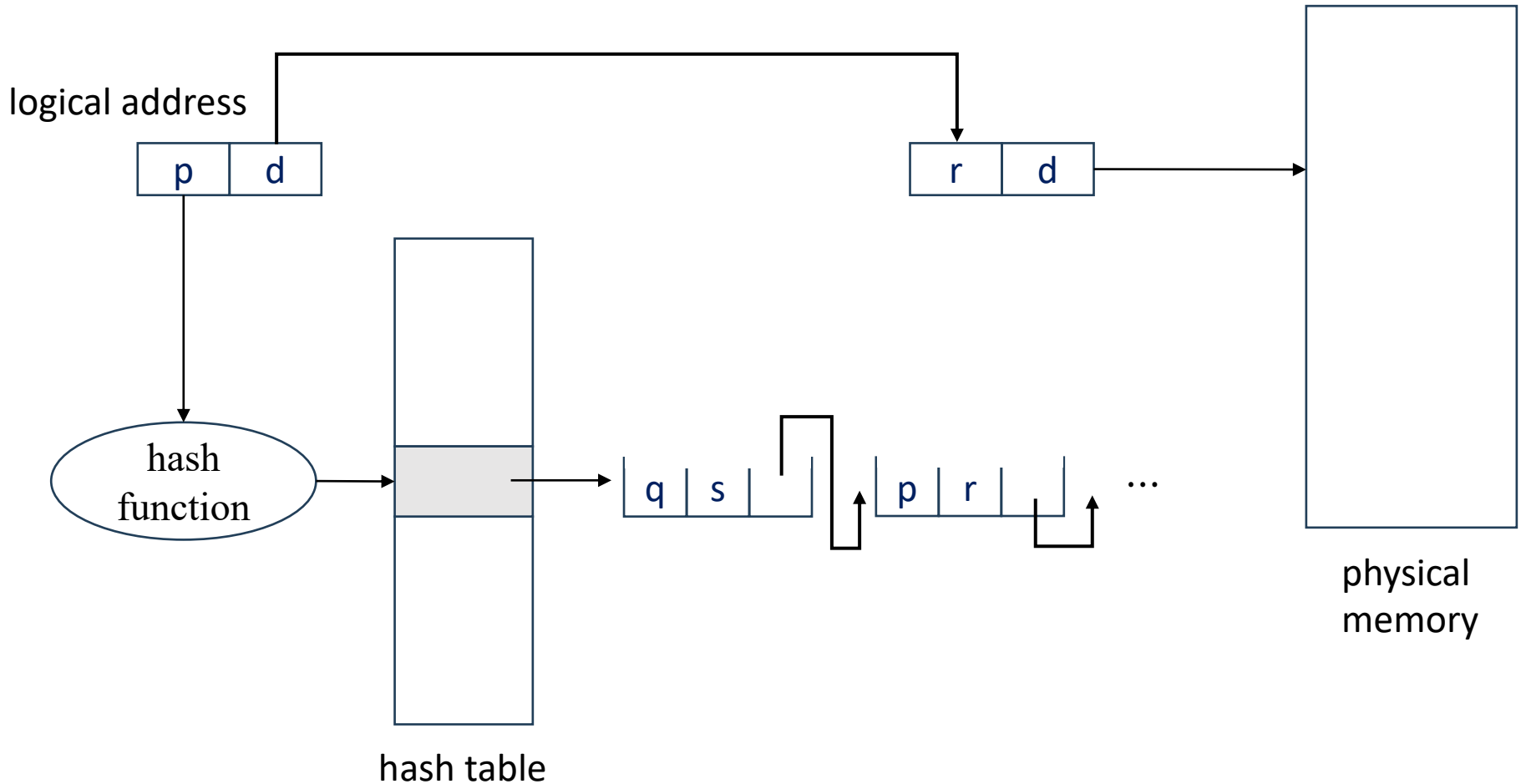
trang ảo

khung trang

- Một máy tính 64 bit. Có RAM 256 MB, kích thước 1 khung trang là 4KB.
- Bảng trang thông thường 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  $\text{hasfunc}(p) = p \bmod 2^{16}$

## 5. Phân trang bộ nhớ (paging)

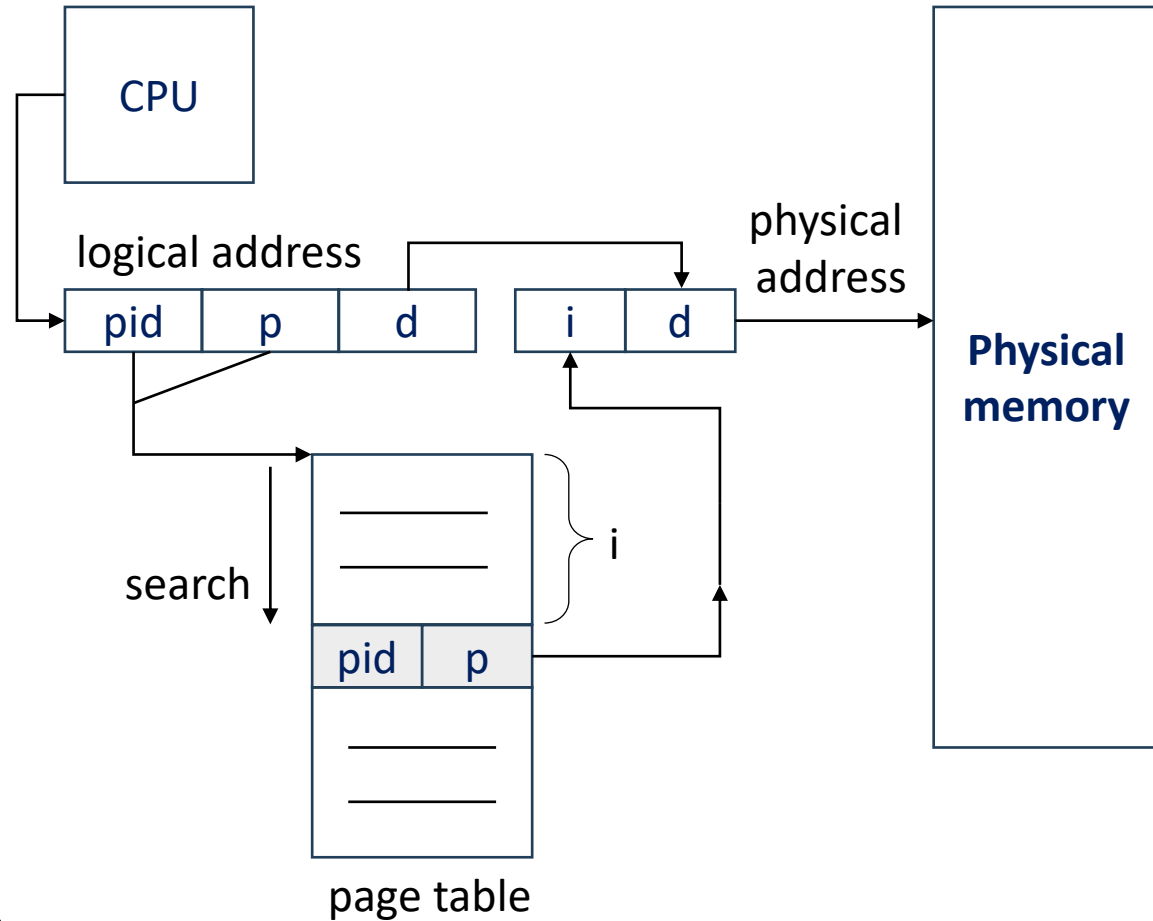
❖ **Bảng trang băm:** Cơ chế chuyển đổi địa chỉ khi sử dụng bảng trang băm.



## 5. Phân trang bộ nhớ (paging)

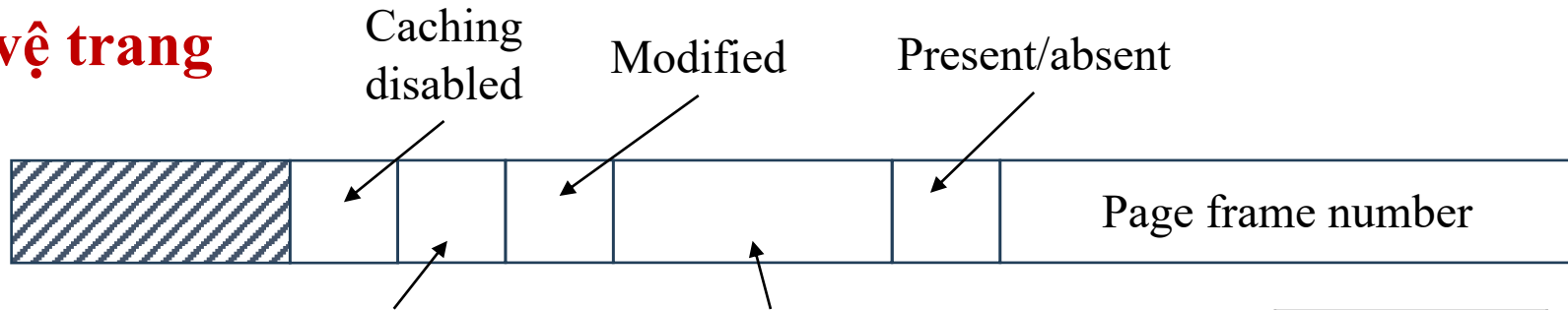
### ❖ Bảng trang nghịch đảo

- 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)



# 5. Phân trang bộ nhớ (paging)

## ❖ Bảo vệ trang



10,468  
12,287

Page 0
Page 1
Page 2
Page 3
Page 4
Page 5

frame number

valid – invalid bit

0	2	v
1	3	v
2	4	v
3	7	v
4	8	v
5	9	v
6	0	i
7	0	i

page table

0	
1	
2	page 0
3	page 1
4	page 2
5	
6	
7	page 3
8	page 4
9	page 5
	⋮
	page n

## 5. 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	data 1
2	code 3
1	code 2
0	code 1

P1

3	0
2	6
1	3
0	2

Bảng trang P1

3	data 2
2	code 3
1	code 2
0	code 1

P2

3	4
2	6
1	3
0	2

Bảng trang P2

7	
6	code 3
5	
4	data 2
3	code 2
2	code 1
1	
0	data 1

Bộ nhớ vật lý

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

## 5. Phân trang bộ nhớ (paging)

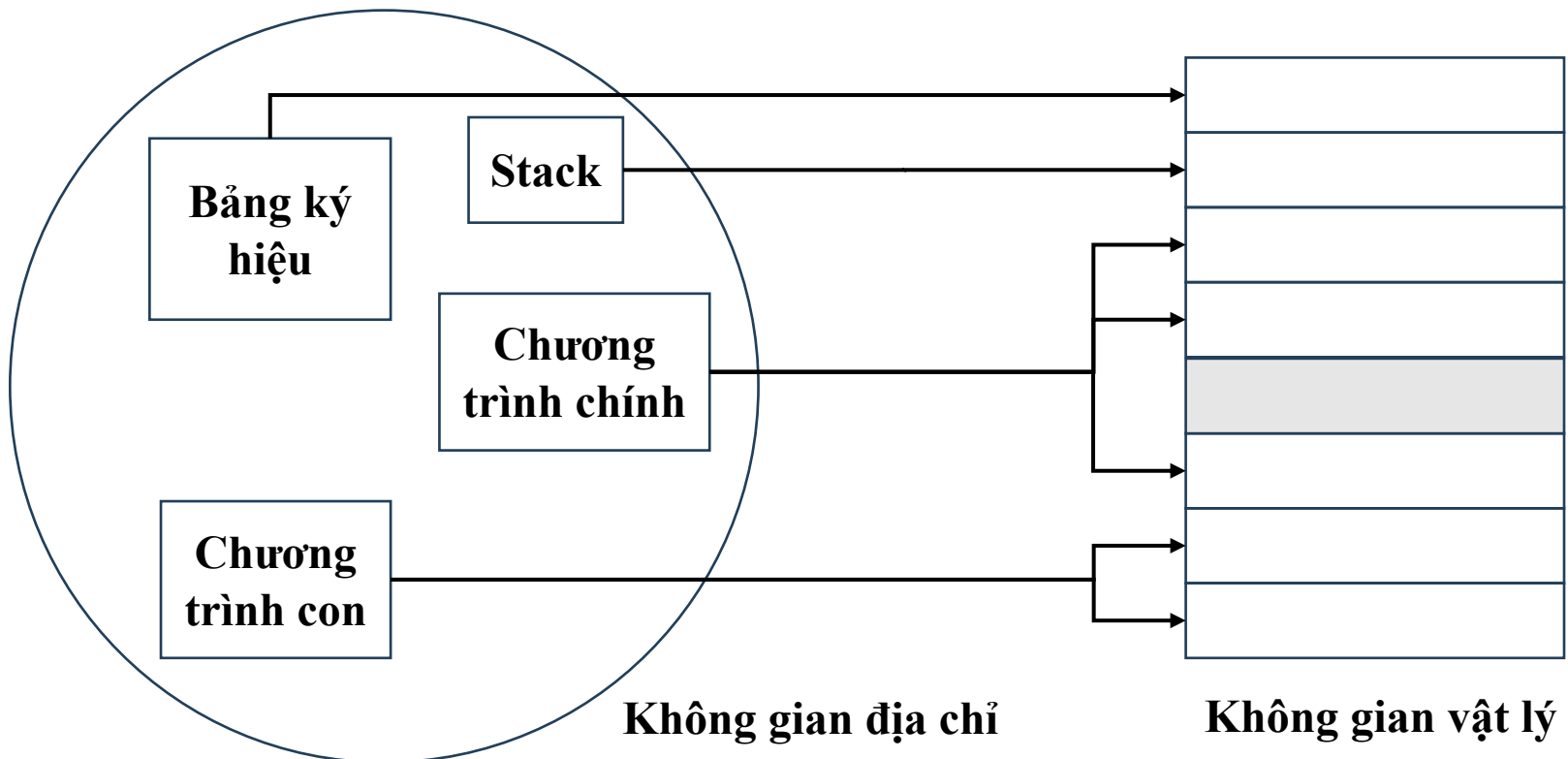
### ❖ Nhận xét

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

## 5. 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.

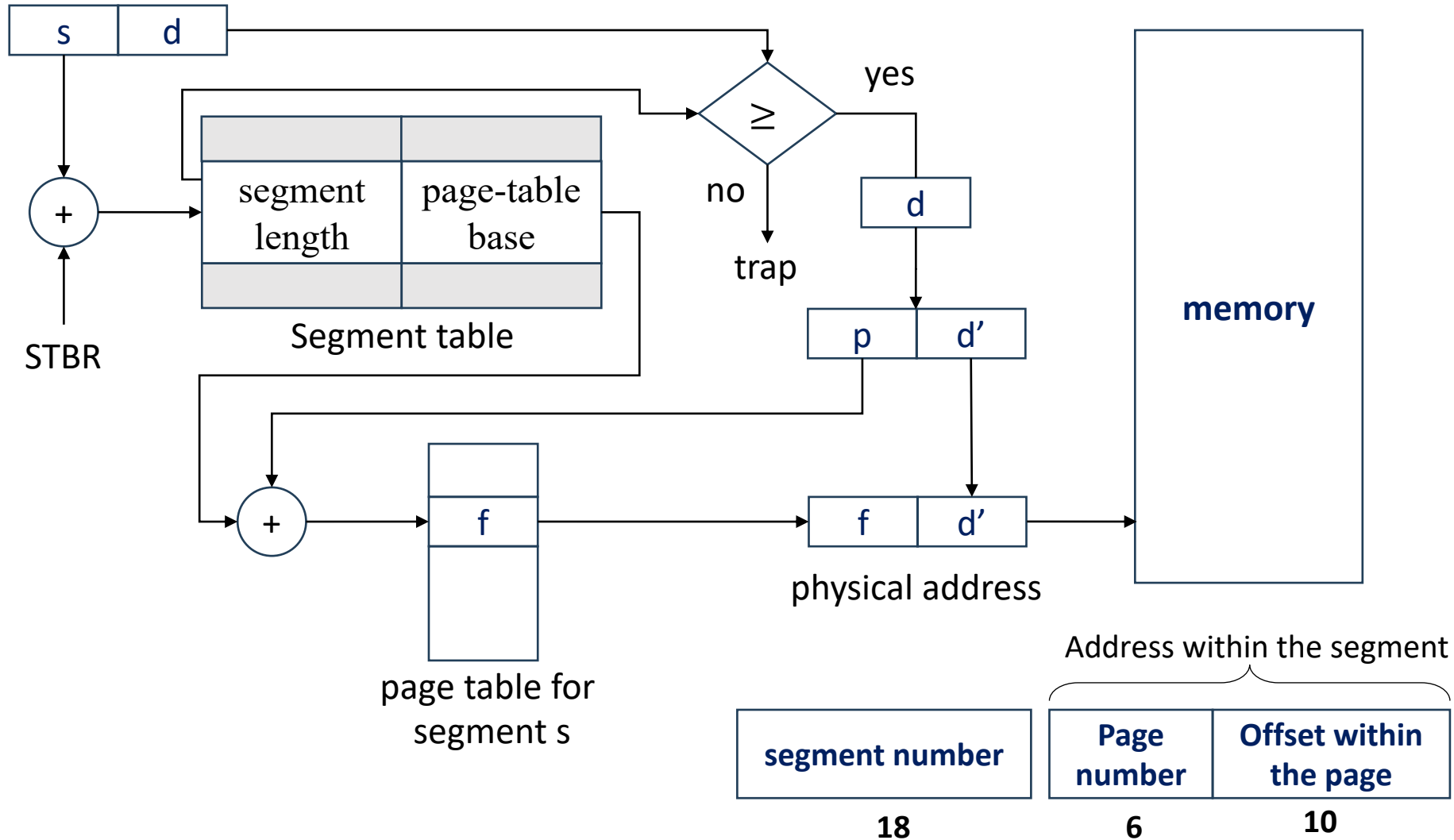




## 5. Phân trang bộ nhớ (paging)

### ▪ Cơ chế MMU trong mô hình phân đoạn kết hợp phân trang

logical address

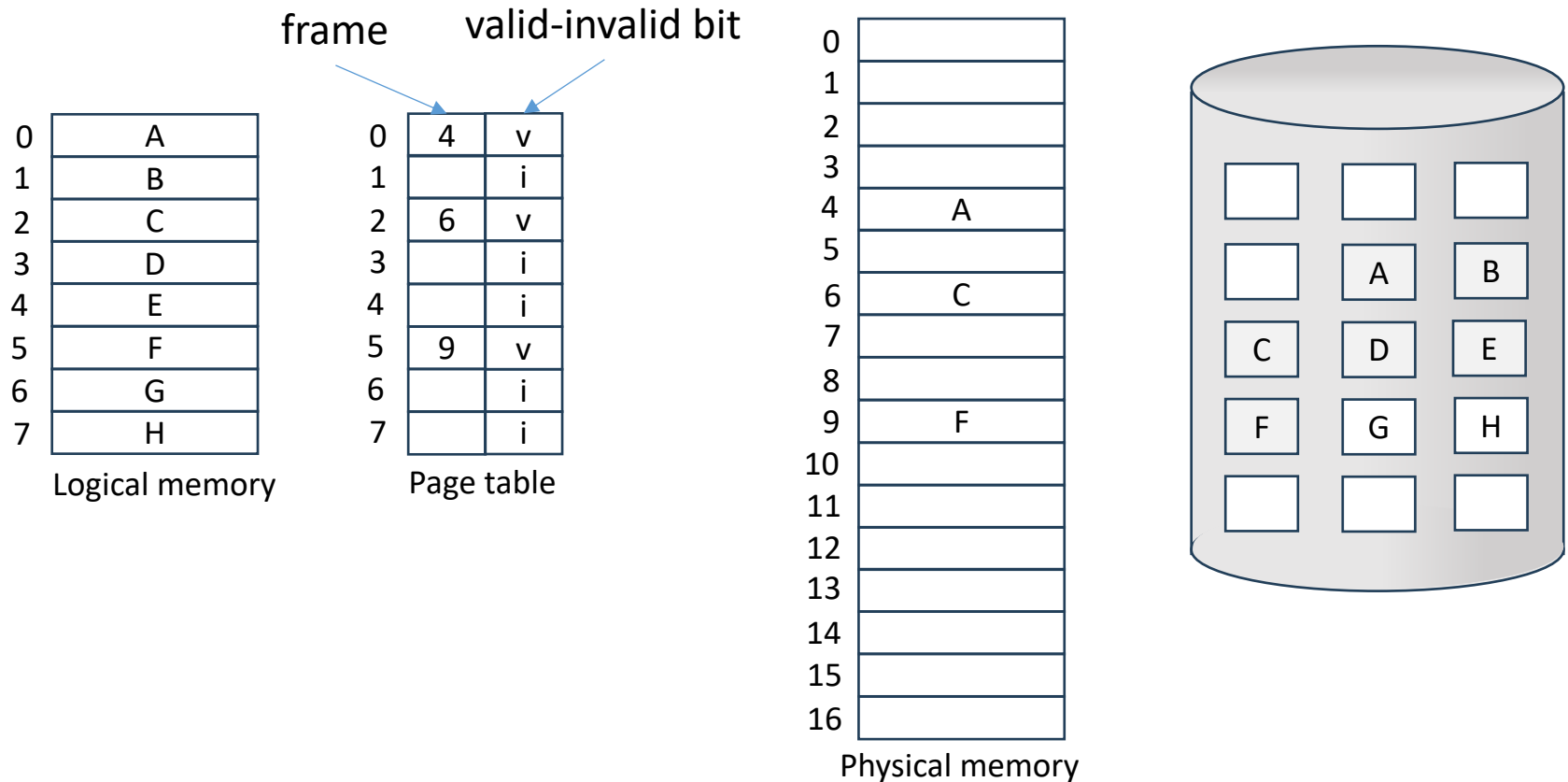


## 6. Bộ nhớ ảo

- Dùng bộ nhớ phụ lưu trữ tiến trình, các phần của tiến trình được chuyển vào – ra giữa bộ nhớ chính và bộ nhớ phụ.
- Phân trang theo yêu cầu (Demand paging)
- Phân đoạn theo yêu cầu (Demand segmentation)

## 6. Bộ nhớ ảo

### ❖ Phân trang theo yêu cầu (Demand paging)

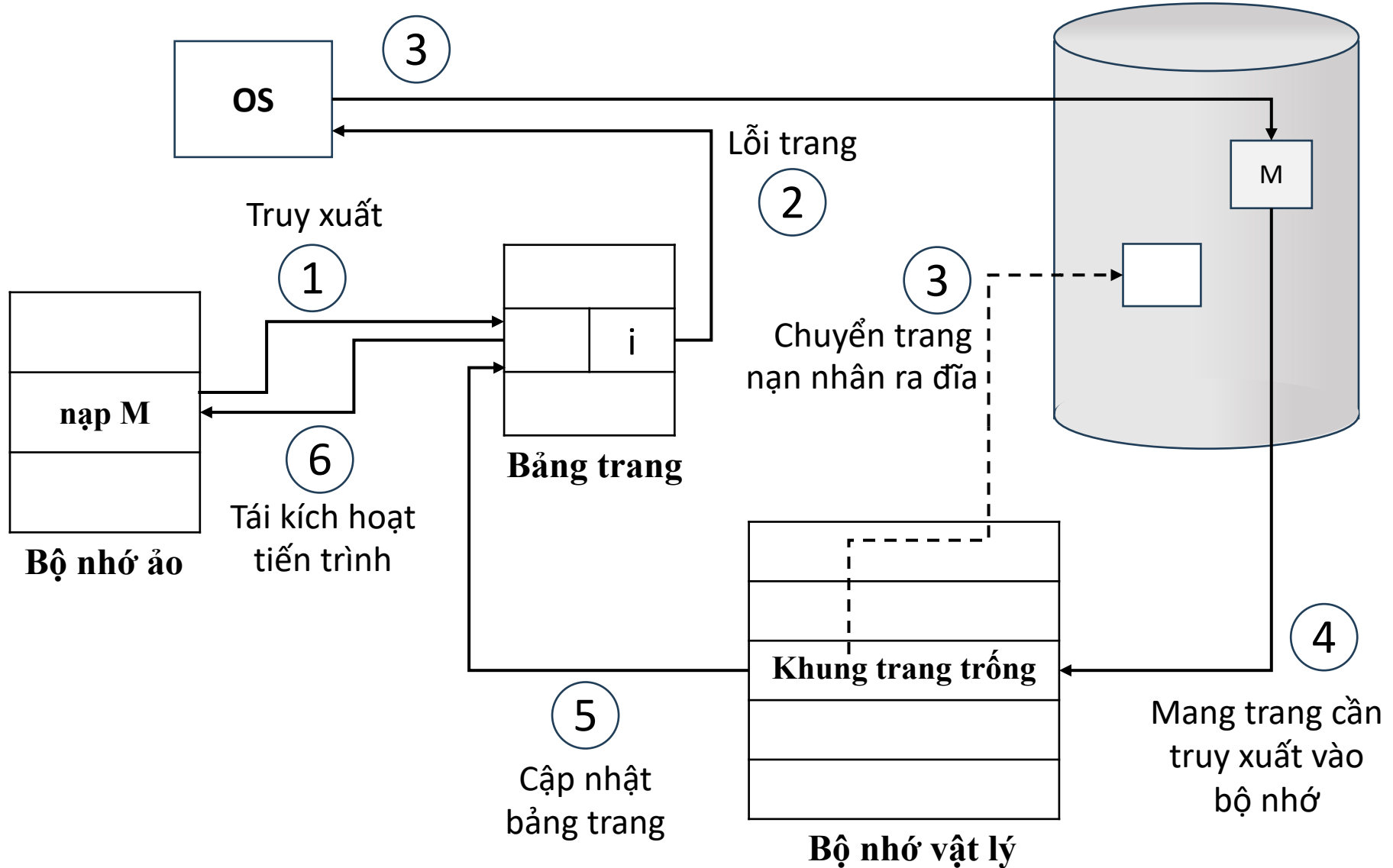


**Trường chứa bit “kiểm tra”:**

- 1 (valid) là trang đang ở trong bộ nhớ chính
- 0 (invalid) là trang đang được lưu trên bộ nhớ phụ hoặc trang không thuộc tiến trình

## 6. Bộ nhớ ảo

### ❖ Chuyển địa chỉ ảo (p, d) thành địa chỉ vật lý



## 6. Bộ nhớ ảo

### ❖ Thay thế trang

- Bit “cập nhật” (dirty bit):
  - 1: nội dung trang có bị sửa đổi.
  - 0: nội dung trang không bị thay đổi.

Số hiệu khung trang chứa trang hoặc địa chỉ trên đĩa của trang	Bit nhận diện trang có trong bộ nhớ (bit valid-invalid)	Bit nhận diện trang có thay đổi (bit dirty)
--	---	---

## 6. Bộ nhớ ảo

### ❖ Thời gian thực hiện một yêu cầu truy xuất bộ nhớ

- **P**: xác suất xảy ra lỗi trang ( $0 \leq p \leq 1$ )
- **Memory access** (ma): thời gian một lần truy xuất bộ nhớ.
- **Effective Access Time** (EAT): thời gian thực hiện một yêu cầu truy xuất bộ nhớ.
- **Page fault overhead** (pfo): thời gian xử lý một lỗi trang.
- **Swap page in** (spi): thời gian chuyển trang từ đĩa vào bộ nhớ.
- **Swap page out** (spo): thời gian chuyển trang ra đĩa (swap page out có thể bằng 0).
- **Restart overhead** (ro): thời gian tái khởi động lại việc truy xuất bộ nhớ.

$$EAT = (1 - p) \times ma + p(pfo + [spo] + spi + ro)$$

## 6. Bộ nhớ ảo

- Ví dụ: Thời gian một lần truy xuất bộ nhớ là 1 micro second và giả sử 40% trang được chọn đã thay đổi nội dung và thời gian hoán chuyển trang ra/vào là 10 mili second. Tính ETA.

$$\begin{aligned} EAT &= (1 - p) \times ma + p(pfo + [spo] + spi + ro) \\ &= (1 - p) \times 1 \\ &\quad + p(pfo + 10000 * 0.4 + 10000 + ro) \text{ micro second} \end{aligned}$$

## 6. Bộ nhớ ảo

### ❖ Các thuật toán chọn trang nạn nhân

- Trang nạn nhân: trang mà sau khi thay thế sẽ gây ra ít lỗi trang nhất.
- **Thuật toán FIFO (First In First Out)**
- **Thuật toán tối ưu (Optical Replacement Algorithm)**
- **Thuật toán LRU (Least Recently used)**
- Các thuật toán xấp xỉ LRU
  - Thuật toán với các bit history
  - Thuật toán cơ hội thứ hai
  - Thuật toán cơ hội thứ hai nâng cao (Not Recently Used Page Replacement Algorithm: NRU)
- Các thuật toán thống kê
  - Thuật toán LFU (Least Frequently Used)
  - Thuật toán MFU (Most Frequently Used)



## 6. Bộ nhớ ảo

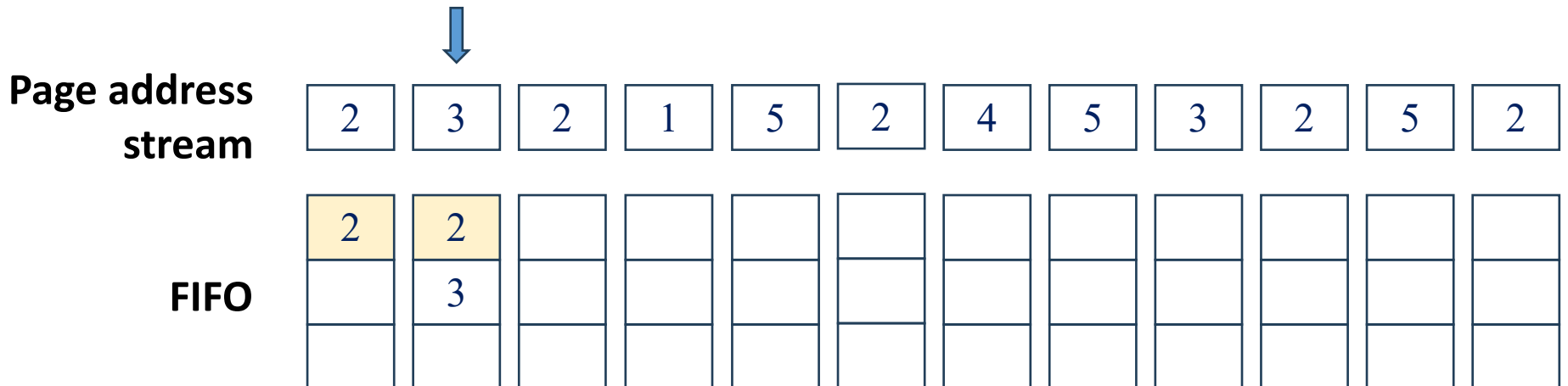
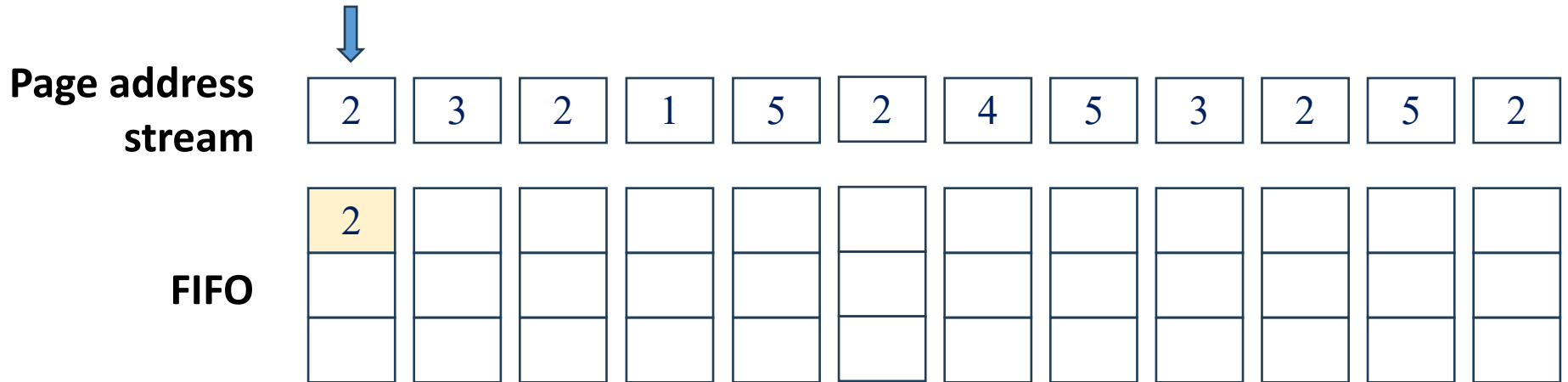
## ❖ Giải thuật thay trang FIFO

- Xem các Frame được cấp phát cho process như circular buffer
- Khi bộ đệm đầy, trang nhớ cũ nhất sẽ được thay thế: FIFO
- Một trang nhớ hay được dùng sẽ là trang cũ nhất  
⇒ hay bị thay thế bởi giải thuật FIFO
- Đơn giản: cần một con trỏ xoay vòng các frame của process

[illegible]

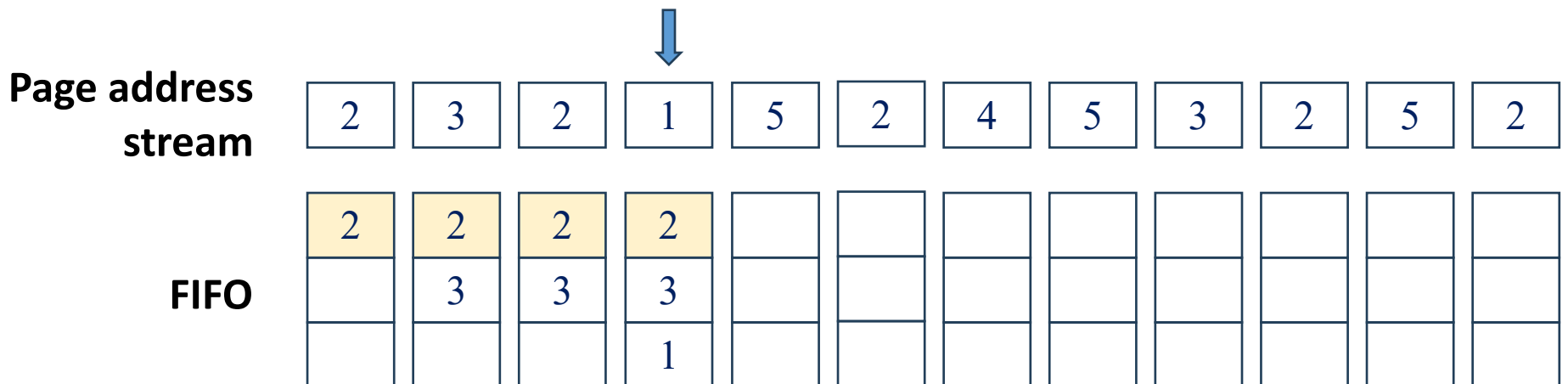
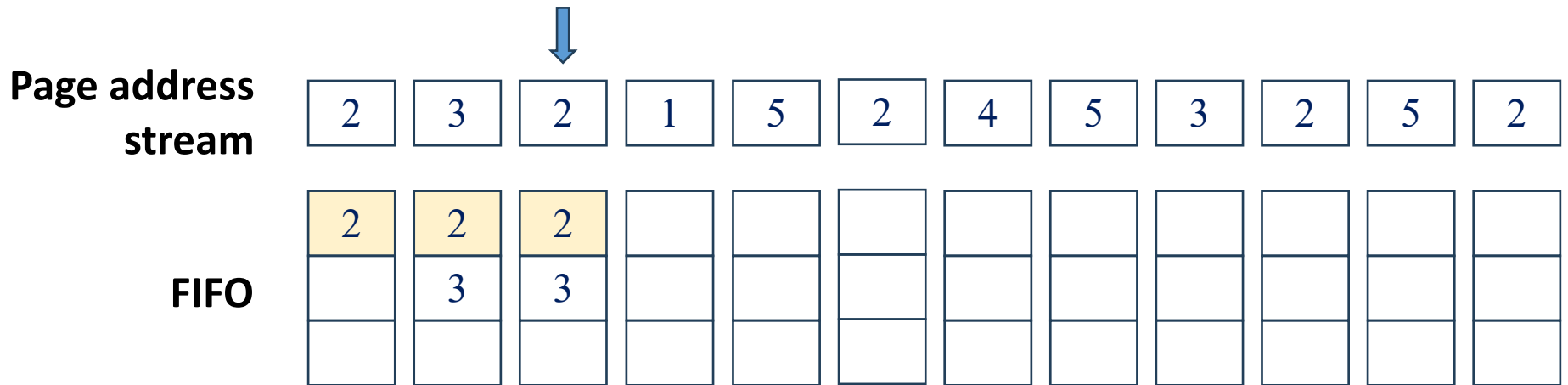
## 6. Bộ nhớ ảo

## ❖ Giải thuật thay trang FIFO



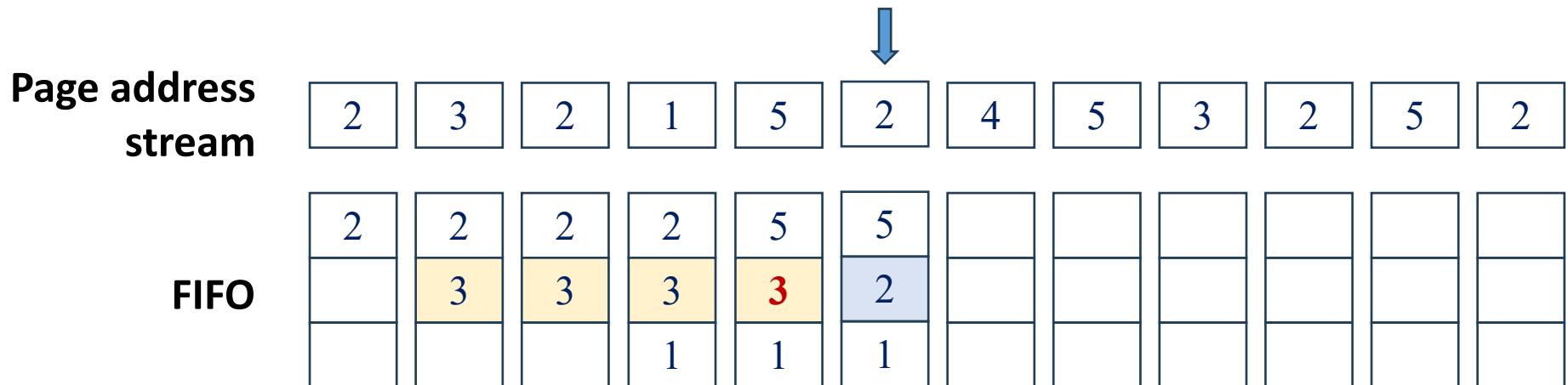
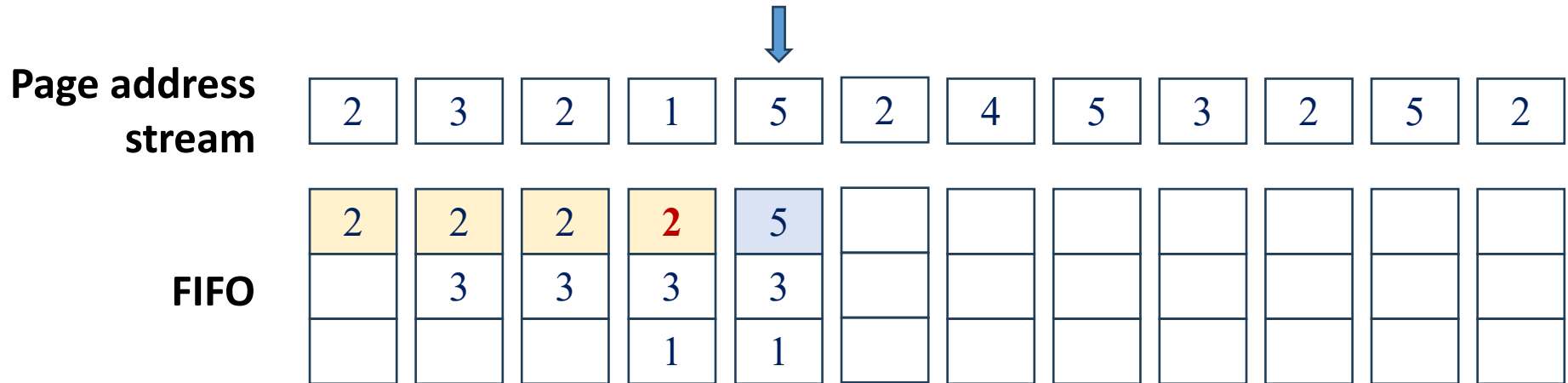
## 6. Bộ nhớ ảo

### ❖ Giải thuật thay trang FIFO



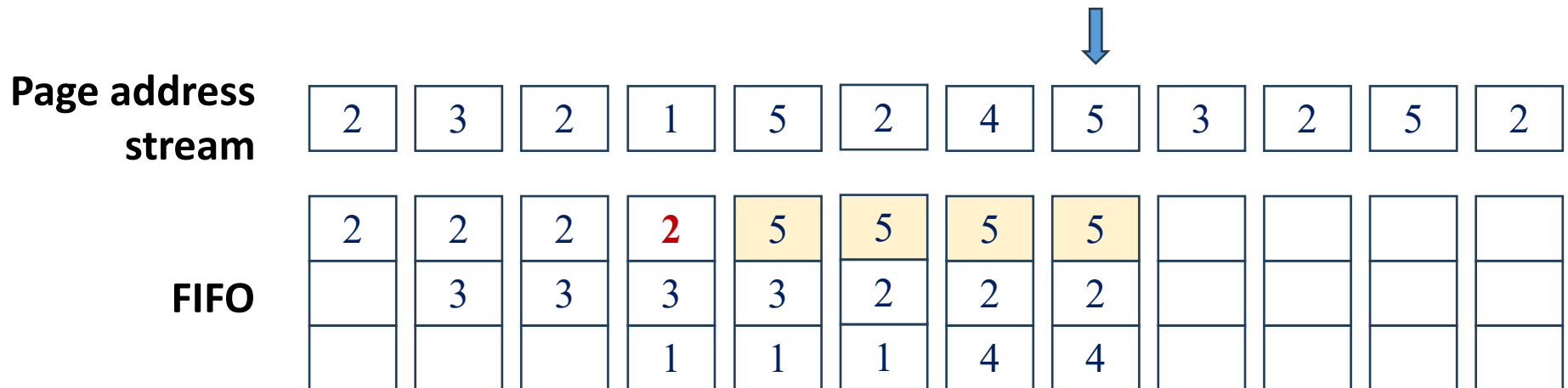
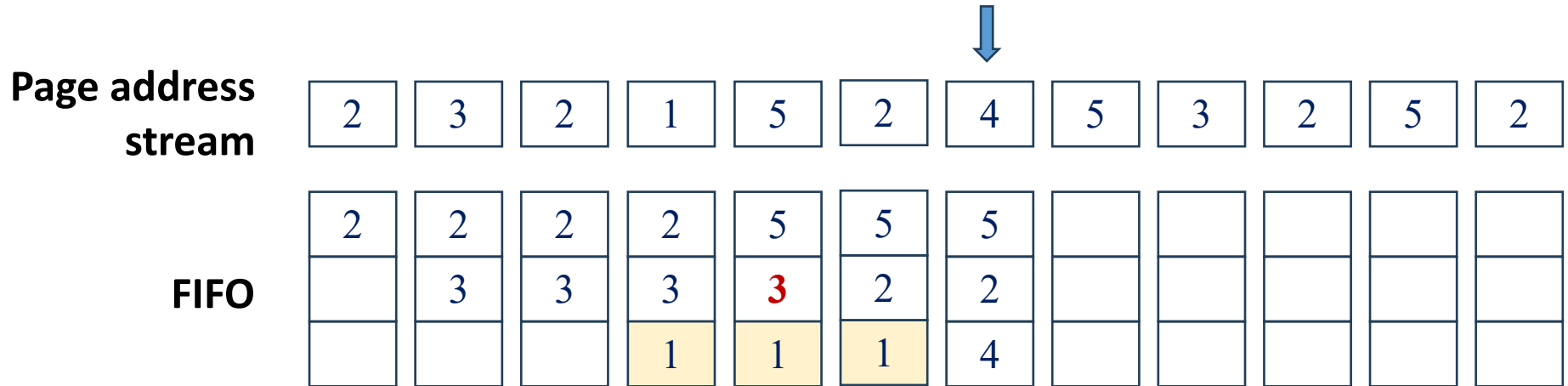
## 6. Bộ nhớ ảo

### ❖ Giải thuật thay trang FIFO



## 6. Bộ nhớ ảo

### ❖ Giải thuật thay trang FIFO



## 6. Bộ nhớ ảo

### ❖ Giải thuật thay trang FIFO

Page address stream												
	2	3	2	1	5	2	4	5	3	2	5	2
FIFO	2	2	2	2	5	5	5	5	3			
		3	3	3	3	2	2	2	2			
				1	1	1	4	4	4			

Page address stream												
	2	3	2	1	5	2	4	5	3	2	5	2
FIFO	2	2	2	2	5	5	5	5	3	3		
		3	3	3	3	2	2	2	2	2		
				1	1	1	4	4	4	4		

## 6. Bộ nhớ ảo

### ❖ Giải thuật thay trang FIFO

Page address stream	<div>↓</div>											
	2	3	2	1	5	2	4	5	3	2	5	2
FIFO	2	2	2	2	5	5	5	5	3	3	3	
		3	3	3	3	2	2	2	2	2	5	
				1	1	1	4	4	4	4	4	

Page address stream	<div>↓</div>											
	2	3	2	1	5	2	4	5	3	2	5	2
FIFO	2	2	2	2	5	5	5	5	3	3	3	3
		3	3	3	3	2	2	2	2	2	5	5
				1	1	1	4	4	4	4	4	2

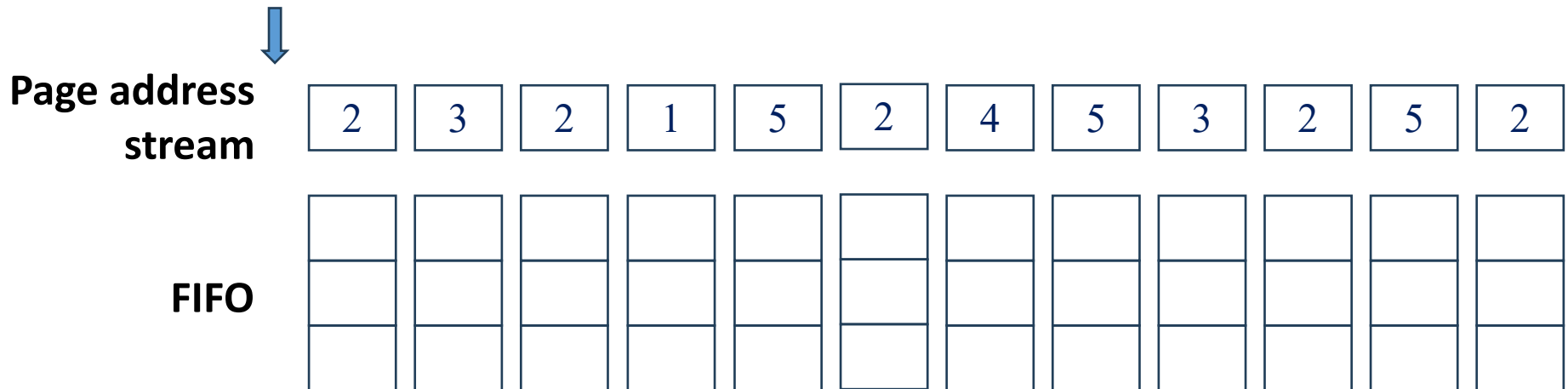
Page address stream	1	2	3	4	1	2	5	1	2	3	4	5
FIFO	1	1	1	1	1	1	5	5	5	5	4	4
		2	2	2	2	2	2	1	1	1	1	5
			3	3	3	3	3	3	2	2	2	2
				4	4	4	4	4	4	3	3	3



## 6. Bộ nhớ ảo

## ❖ Giải thuật thay trang OPT – Optimal

- Thay thế trang nhớ sẽ được tham chiếu trễ nhất trong tương lai
- Ví dụ: Một process có 5 trang, và được cấp 3 frame



## 6. Bộ nhớ ảo

## ❖ Giải thuật thay trang OPT – Optimal

[illegible][illegible]

## 6. Bộ nhớ ảo

### ❖ Giải thuật thay trang OPT – Optimal



Page address  
stream

2	3	2	1	5	2	4	5	3	2	5	2
---	---	---	---	---	---	---	---	---	---	---	---

FIFO

2	2	2									
	3	3									



Page address  
stream

2	3	2	1	5	2	4	5	3	2	5	2
---	---	---	---	---	---	---	---	---	---	---	---

FIFO

2	2	2	2								
	3	3	3								
			1								

## 6. Bộ nhớ ảo

### ❖ Giải thuật thay trang OPT – Optimal



Page address  
stream

2	3	2	1	5	2	4	5	3	2	5	2
---	---	---	---	---	---	---	---	---	---	---	---

FIFO

2	2	2	2	2							
	3	3	3	3							
			1	5							



Page address  
stream


2	3	2	1	5	2	4	5	3	2	5	2
---	---	---	---	---	---	---	---	---	---	---	---

FIFO


2	2	2	2	2	2						
	3	3	3	3	3						
			1	5	5						

## 6. Bộ nhớ ảo

### ❖ Giải thuật thay trang OPT – Optimal



Page address stream	2	3	2	1	5	2	4	5	3	2	5	2
FIFO	2	2	2	2	2	2	4					
		3	3	3	3	3	3					
				1	5	5	5					



Page address stream	2	3	2	1	5	2	4	5	3	2	5	2
FIFO	2	2	2	2	2	2	4	4				
		3	3	3	3	3	3	3				
				1	5	5	5	5				

## 6. Bộ nhớ ảo

### ❖ Giải thuật thay trang OPT – Optimal



Page address  
stream

2	3	2	1	5	2	4	5	3	2	5	2
---	---	---	---	---	---	---	---	---	---	---	---

FIFO

2	2	2	2	2	2	4	4	4			
	3	3	3	3	3	3	3	3			
			1	5	5	5	5	5			



Page address  
stream

2	3	2	1	5	2	4	5	3	2	5	2
---	---	---	---	---	---	---	---	---	---	---	---

FIFO

2	2	2	2	2	2	4	4	4	2		
	3	3	3	3	3	3	3	3	3		
			1	5	5	5	5	5	5		

## 6. Bộ nhớ ảo

### ❖ Giải thuật thay trang OPT – Optimal

												↓
Page address stream	2	3	2	1	5	2	4	5	3	2	5	2
FIFO	2	2	2	2	2	2	4	4	4	2	2	
		3	3	3	3	3	3	3	3	3	3	
				1	5	5	5	5	5	5	5	

												↓
Page address stream	2	3	2	1	5	2	4	5	3	2	5	2
FIFO	2	2	2	2	2	2	4	4	4	2	2	2
		3	3	3	3	3	3	3	3	3	3	3
				1	5	5	5	5	5	5	5	5

## 6. Bộ nhớ ảo

### ❖ Giải thuật thay trang OPT – Optimal

- Số lượng lỗi trang phát sinh là thấp nhất
- Không bị nghịch lý Belady
- Khó cài đặt
- Phù hợp với hệ điều hành cho thiết bị gia dụng



## 6. Bộ nhớ ảo

### ❖ Giải thuật thay trang Least Recently Used (LRU)

- Thay thế trang nhớ không được tham chiếu lâu nhất
- Ví dụ một process có 5 trang, và được cấp 3 frame

Page address  
stream

LRU

2	3	2	1	5	2	4	5	3	2	5	2
2	2	2	2	2	2	2	2	3	3	3	3
	3	3	3	5	5	5	5	5	5	5	5
			1	1	1	4	4	4	2	2	2

## 6. Bộ nhớ ảo

### ❖ Cài đặt thuật toán LRU

#### ▪ Sử dụng bộ đếm

- Cấu trúc phần tử trong bảng trang: thêm trường ghi nhận “thời điểm truy xuất gần nhất”
- Cấu trúc của CPU: thêm một thanh ghi đếm (counter)

Số hiệu khung trang chứa trang hoặc địa chỉ trang trên đĩa	Bit valid - invalid	Bit nhận diện trang có thay đổi (bit dirty)	Thời điểm truy xuat gần nhất
--	------------------------	---	---------------------------------

#### ▪ Sử dụng danh sách liên kết

- Trang ở cuối danh sách là trang được truy xuất gần nhất
- Trang ở đầu danh sách là trang lâu nhất chưa được sử dụng

## 6. Bộ nhớ ảo

### ❖ Các thuật toán xấp xỉ LRU

- Mỗi phần tử trong bảng trang có thêm bit reference:
  - Được khởi gán là 0 bởi hệ điều hành
  - Được phần cứng gán là 1 mỗi lần trang tương ứng được truy cập

Số hiệu khung trang chứa trang hoặc địa chỉ trang trên đĩa	Bit valid - invalid	Bit nhận diện trang có thay đổi (bit dirty)	Bit reference
--	------------------------	---	---------------

- Các thuật toán xấp xỉ LRU
  - Thuật toán với các bit history
  - Thuật toán cơ hội thứ hai
  - Thuật toán cơ hội thứ hai nâng cao (Not Recently Used Page Replacement Algorithm: NRU)

## 6. Bộ nhớ ảo

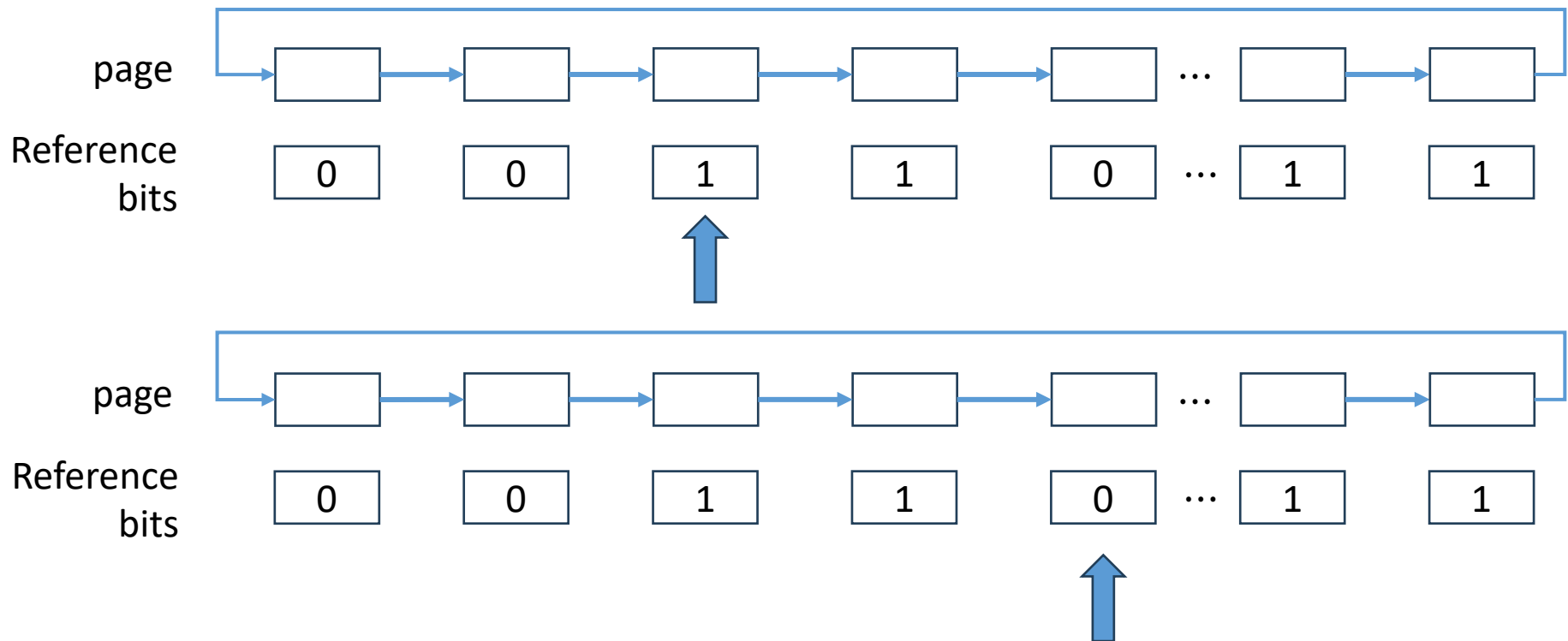
### ❖ Thuật toán với các bit history

- Mỗi trang sử dụng thêm 8 bit lịch sử (history).
- Cập nhật các bit history:
  - Dịch các bit history sang phải 1. i trí để loại bỏ bit thấp nhất.
  - Đặt bit reference của mỗi trang vào bit cao nhất trong 8 bit history của trang đó.
- 8 bit history sẽ lưu trữ tình hình truy xuất đến trang trong 8 chu kỳ cuối cùng.
- Trang nạn nhân là trang có giá trị history nhỏ nhất.

## 6. Bộ nhớ ảo

### ❖ Thuật toán cơ hội thứ hai:

- Tìm một trang theo nguyên tắc FIFO
- Kiểm tra bit reference của trang đó.
  - Nếu bit reference là 0, chọn trang này.
  - Nếu bit reference là 1 thì gán lại là 0 rồi tìm trang FIFO tiếp theo.



## 6. Bộ nhớ ảo

### ❖ Thuật toán cơ hội thứ hai:

7 0 1 2 0 3 0 4 2 3 0 3 2 1 2 0 1 7 0 1

7	7	7	2	2	2	2	4	4	4	0	0	0	1	1	1	1	1	1	1
	0	0	0	0	0	0	0	0	0	3	3	3	3	3	3	0	0	0	0
		1	1	1	3	3	3	2	2	2	2	2	2	2	2	2	7	7	7

\* \* \* \* \* \* \* \* \* \* \* \*

7 0 1 2 0 3 0 4 2 3 0 3 2 1 2 0 1 7 0 1

7(1)	7(1)	1(1)	0(0)	1(0)	2(1)	2(1)	3(0)	0(0)	4(1)	2(0)	2(0)	0(1)	3(0)	3(0)	1(0)	2(1)	0(0)	1(0)	7(1)
	0(1)	0(1)	1(0)	2(1)	0(1)	3(1)	0(0)	4(1)	2(1)	3(0)	0(1)	3(1)	2(0)	1(0)	2(1)	0(1)	1(0)	7(1)	0(1)
		1(1)	2(1)	0(1)	3(1)	0(1)	4(1)	2(1)	3(1)	0(1)	3(1)	2(1)	1(0)	2(1)	0(1)	1(1)	7(1)	0(1)	1(1)

## 6. Bộ nhớ ảo

### ❖ Thuật toán cơ hội thứ hai nâng cao (Not Recently Used Page Replacement Algorithm – NRU):

- Lớp 1(0,0): gồm những trang có  $(\text{ref}, \text{dirty}) = (0, 0)$ . (độ ưu tiên thấp)
  - Không được truy xuất gần đây và không bị sửa đổi.
  - Tốt nhất để thay thế.
- Lớp 2 (0,1):
  - Không truy xuất gần đây nhưng đã bị sửa đổi.
  - Trường hợp này không thật tốt, vì trang cần được lưu trữ lại trước khi thay thế.

## 6. Bộ nhớ ảo

### ❖ Thuật toán cơ hội thứ hai nâng cao (Not Recently Used Page Replacement Algorithm – NRU):

- Lớp 3(1,0):
  - Được truy xuất gần đây, nhưng không bị sửa đổi.
  - Trang có thể nhanh chóng được tiếp tục được sử dụng.
- Lớp 4 (1,1):
  - trang được truy xuất gần đây, và bị sửa đổi.
  - Trang có thể nhanh chóng được tiếp tục được sử dụng và trước khi thay thế cần phải được lưu trữ lại.
- Trang nạn nhân: trang đầu tiên tìm thấy trong lớp có độ ưu tiên thấp nhất.



## 6. Bộ nhớ ảo

### ❖ Các thuật toán thống kê

- Biến đếm: lưu số lần truy xuất đến một trang.
- Thuật toán LFU (Least Frequently Used):
  - Thay thế trang có giá trị biến đếm nhỏ nhất, nghĩa là trang ít được sử dụng nhất.
- Thuật toán MFU (Most Frequently Used):
  - Thay thế trang có giá trị biến đếm lớn nhất, nghĩa là trang được sử dụng nhiều nhất.

## 6. Bộ nhớ ảo

### ❖ Chiến lược cấp phát khung trang

#### ▪ Cấp phát ngang bằng:

- $m$  khung trang và  $n$  tiến trình
- Mỗi tiến trình được cấp  $m/n$  khung trang.

#### ▪ Cấp phát theo tỷ lệ kích thước

- $s_i$ : kích thước của tiến trình  $p_i$
- $S = \sum s_i$  là tổng kích thước của tất cả tiến trình
- $m$ : số lượng khung trang có thể sử dụng
- $a_i$ : số khung trang được cấp phát cho tiến trình  $p_i$
- $a_i = \frac{s_i}{S} \times m$

## 6. Bộ nhớ ảo

### ❖ Chiến lược cấp phát khung trang

- Cấp phát ngang bằng
- Cấp phát theo tỷ lệ kích thước
- Ví dụ: Tiến trình 1 = 10K, tiến trình 2 = 127K, có 62 khung trang trống. Khi đó có thể cấp cho

$$a_i = \frac{s_i}{S} \times m$$

○ tiến trình 1:

$$a_i = \frac{10}{10 + 127} \times 62 \approx 4$$

○ tiến trình 2:

$$a_i = \frac{127}{10 + 127} \times 62 \approx 57$$

- Cấp phát theo tỷ lệ độ ưu tiên

## 6. Bộ nhớ ảo

### ❖ Thay thế trang

#### ■ Thay thế toàn cục

- Chọn trang nạn nhân từ tập tất cả các khung trang trong hệ thống
- Có nhiều khả năng lựa chọn hơn
- Số khung trang cấp cho một tiến trình có thể thay đổi.
- Các tiến trình không thể kiểm soát được tỷ lệ phát sinh lỗi trang của mình.

#### ■ Thay thế cục bộ

- Chỉ chọn trang thay thế trong tập các khung trang được cấp cho tiến trình phát sinh lỗi trang.
- Số khung trang cấp cho một tiến trình sẽ không thay đổi.

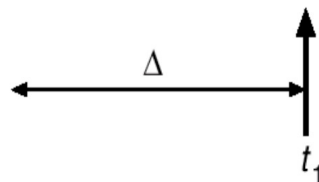
## 7. Tình trạng trì trệ

### ❖ Mô hình tập làm việc (working set)

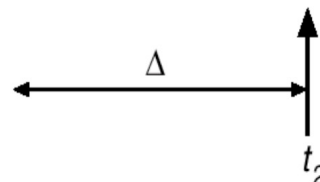
- **WSSI**( $\Delta, t$ ): số phần tử của tập working set của tiến trình  $P_i$  tại thời điểm  $t$ .
  - Tập các trang được tiến trình truy xuất đến trong  $\Delta$  lần truy cập cuối cùng tình tại thời điểm  $t$ .
- $m$ : số khung trang trống.
- $D = \sum \mathbf{WSSI}$ : tổng số khung trang yêu cầu cho toàn hệ thống
- Tại thời điểm  $t$ : cấp cho  $P_i$  số khung trang bằng **WSSI**( $\Delta, t - 1$ )
- $D > m$ : Trì trệ hệ thống

page reference table

... 2 6 1 5 7 7 7 7 5 1 6 2 3 4 1 2 3 4 4 4 3 4 3 4 4 4 1 3 2 3 4 4 4 3 4 4 4 ...



$$WS(t_1) = \{1, 2, 5, 6, 7\}$$



$$WS(t_2) = \{3, 4\}$$