



Chương 7:
QUẢN LÝ BỘ NHỚ



CÁC KHÁI NIỆM

- Quản lý bộ nhớ là công việc của hệ điều hành với sự hỗ trợ của phần cứng, sắp xếp các process trong bộ nhớ sao cho hiệu quả.

Mục tiêu: Càng nhiều process vào bộ nhớ càng tốt.

- Trong hầu hết các hệ thống thì kernel sẽ chiếm một phần cố định trong bộ nhớ.

Ví dụ: RAM 8GB chỉ sử dụng được 7,88GB còn lại là của kernel.

Các yêu cầu đối với việc quản lý bộ nhớ:

- Cấp phát tài nguyên cho process
- Tái định vị
- Bảo vệ
- Chia sẻ
- Kết gán địa chỉ nhớ luận lý



CÁC KIỂU ĐỊA CHỈ NHỚ

1. Địa chỉ luận lý (logical address/ virtual address – địa chỉ ảo)

- Là vị trí nhớ được diễn tả trong một chương trình.
- $ĐC_{LL} = ĐC_{page} + ĐC_{offset}$.

❖ Ngoài ra:

- **Địa chỉ tuyệt đối (absolute address):** Địa chỉ tương đương với địa chỉ thực.
- **Địa chỉ tương đối (relative address):** địa chỉ được biến đổi tương đối so với một vị trí xác định nào đó và không phụ thuộc vào vị trí thực của tiến trình trong bộ nhớ.

2. Địa chỉ vật lý (physical address – địa chỉ thực):

- Là một vị trí thực trong bộ nhớ chính.
 - $ĐC_{VL} = ĐC_{frame} + ĐC_{offset}$.
- ❖ Để truy cập bộ nhớ, địa chỉ luận lý cần được biến đổi thành địa chỉ vật lý.



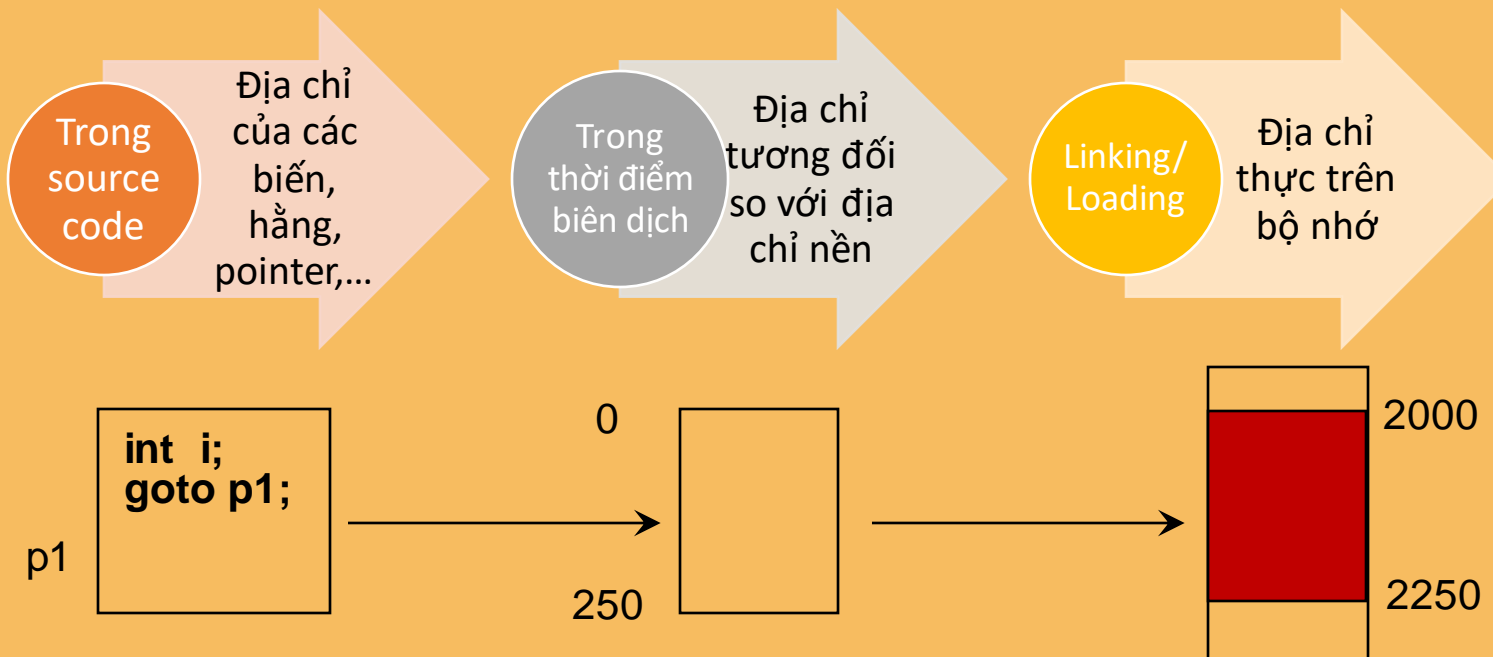
Địa chỉ dạng “12 byte so với vị trí bắt đầu của chương trình” là địa chỉ gì?

- A. Địa chỉ tuyệt đối
- B. Địa chỉ vật lý
- C. Địa chỉ thực
- D. Địa chỉ tương đối**

CHUYỂN ĐỔI ĐỊA CHỈ NHỚ

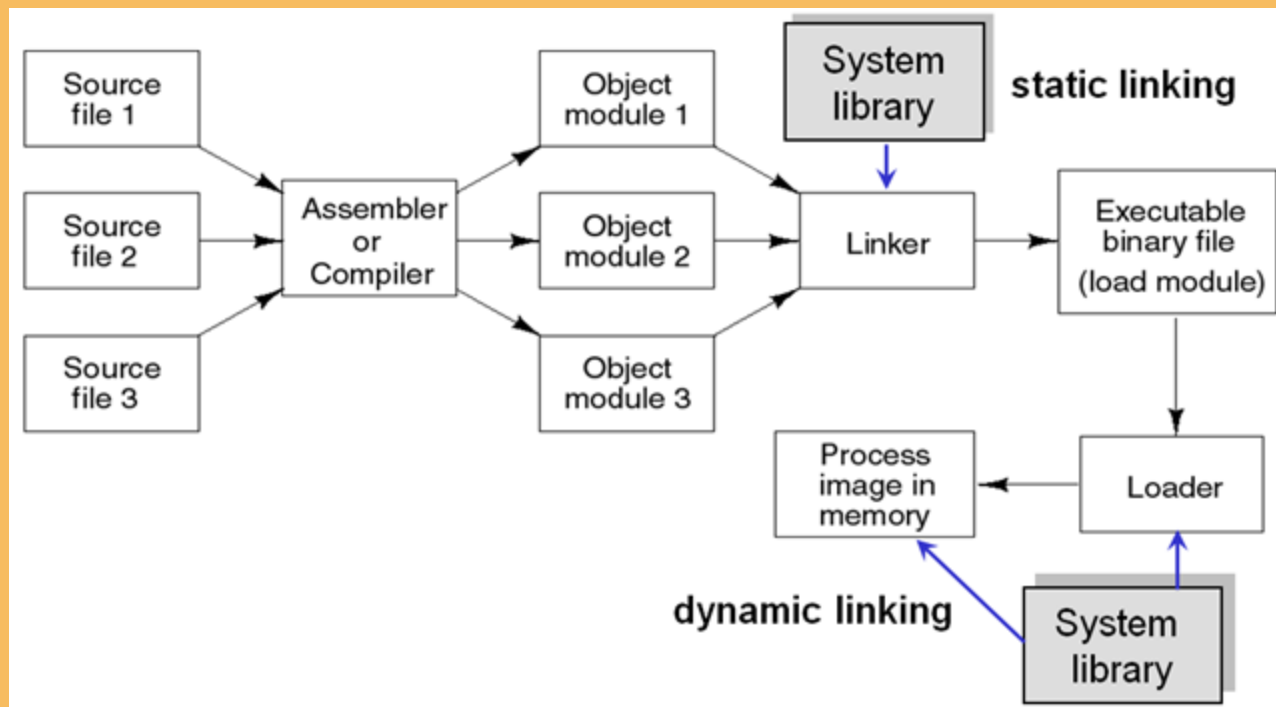
Là quá trình ánh xạ một địa chỉ từ không gian địa chỉ này sang không gian địa chỉ khác.

❖ Biểu diễn địa chỉ nhớ:



NẠP CHƯƠNG TRÌNH VÀO BỘ NHỚ

- **Linker:** Kết hợp nhiều object module thành 1 file nhị phân (file tạo thành gọi là load module).
- **Loader:** Nạp load module vào bộ nhớ chính.





Địa chỉ lệnh và dữ liệu 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.
- **Load time:** chuyển đổi địa chỉ luận lý thành địa chỉ thực dựa trên địa chỉ nền.
- **Excution time:** khi process di chuyển qua lại giữa các segment trong bộ nhớ.

MÔ HÌNH QUẢN LÝ BỘ NHỚ

- Một process phải được nạp hoàn toàn vào bộ nhớ thì mới được thực thi.
- Một số cơ chế quản lý bộ nhớ:
 - Phân chia cố định (fixed partitioning)
 - Phân chia động (dynamic partitioning)
 - Phân trang đơn giản (simple paging)



HIỆN TƯỢNG PHÂN MẢNH BỘ NHỚ

1. Hiện tượng phân mảnh nội:

Kích thước vùng nhớ được cấp phát lớn hơn vùng nhớ yêu cầu.

=> Có thể sử dụng các chiến lược placement.

2. Hiện tượng phân mảnh ngoại:

Kích thước không gian bộ nhớ trống đủ thỏa mãn yêu cầu cấp phát, tuy nhiên không liên tục.

=> Có thể dùng cơ chế kết khối (compaction).



Nếu hệ thống cấp phát vùng nhớ có kích thước 20480 byte cho tiến trình yêu cầu 20324 byte thì sẽ dẫn đến tình trạng gì?

- A. Deadlock
- B. Phân mảnh ngoại
- ☒ C. Phân mảnh nội
- D. Số lỗi trang tăng lên



PHÂN CHIA CỔ ĐỊNH

- Bộ nhớ chính được chia thành nhiều phần (partition), có kích thước bằng nhau hoặc khác nhau.
- Process nào nhỏ hơn hoặc bằng kích thước partition thì có thể nạp vào.
- Nếu process có kích thước lớn hơn => Overlay.

=> Kém hiệu quả do bị phân mảnh nội.



PHÂN CHIA CỔ ĐỊNH

1. Chiến lược placement với partition cùng kích thước:

- Còn partition trống
=> nạp vào.
- Không còn partition trống
=> swap process đang bị blocked ra bộ nhớ phụ, nhường chỗ cho process mới.

2. Chiến lược placement với partition khác kích thước:

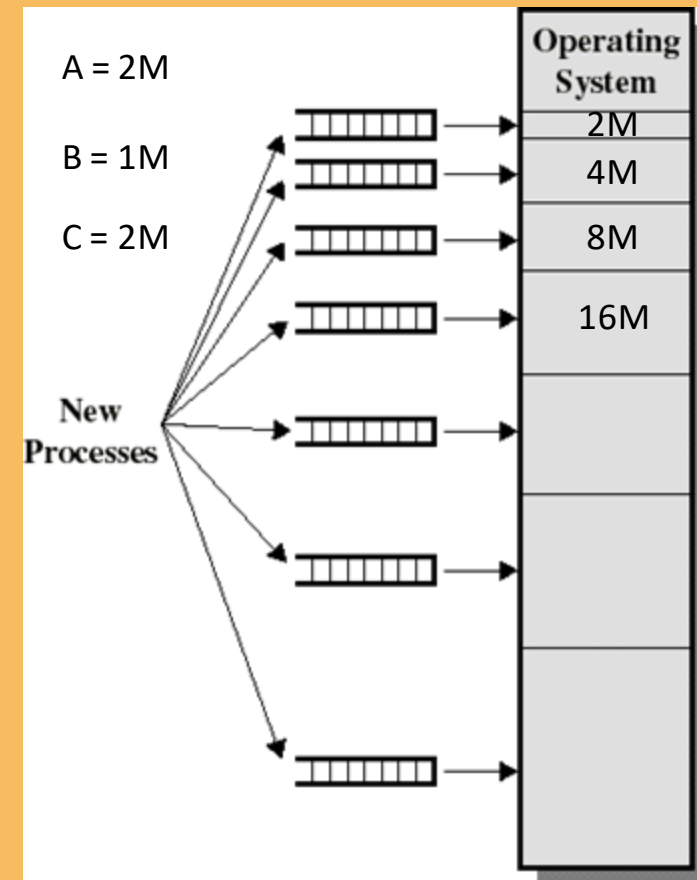
- Giải pháp 1: Sử dụng nhiều hàng đợi
 - Gán mỗi process vào partition nhỏ nhất phù hợp.
 - **Ưu điểm**: Giảm thiểu phân mảnh nội
 - **Nhược điểm**: Có thể có hàng đợi trống và hàng đợi đầy đặc.
- Giải pháp 2: Sử dụng 1 hàng đợi
 - Chỉ có một hàng đợi chung cho mọi partition.*=> Chọn partition nhỏ nhất còn trống.*

PHÂN CHIA CỐ ĐỊNH

2. Chiến lược placement với partition khác kích thước:

- Giải pháp 1: Sử dụng nhiều hàng đợi
 - Gán mỗi process vào partition nhỏ nhất phù hợp.
 - **Ưu điểm**: Giảm thiểu phân mảnh nội
 - **Nhược điểm**: Có thể có hàng đợi trống và hàng đợi đầy đặc.
- Giải pháp 2: Sử dụng 1 hàng đợi
 - Chỉ có một hàng đợi chung cho mọi partition.

=> Chọn partition nhỏ nhất còn trống.

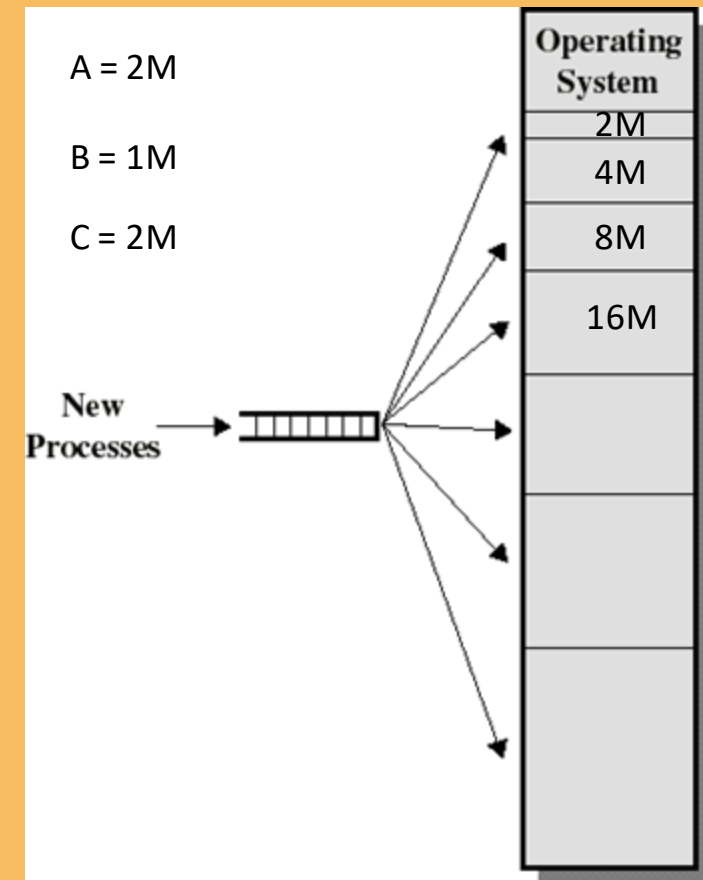


PHÂN CHIA CỐ ĐỊNH

2. Chiến lược placement với partition khác kích thước:

- Giải pháp 1: Sử dụng nhiều hàng đợi
 - Gán mỗi process vào partition nhỏ nhất phù hợp.
 - **Ưu điểm**: Giảm thiểu phân mảnh nội
 - **Nhược điểm**: Có thể có hàng đợi trống và hàng đợi đầy đặc.
- Giải pháp 2: Sử dụng 1 hàng đợi
 - Chỉ có một hàng đợi chung cho mọi partition.

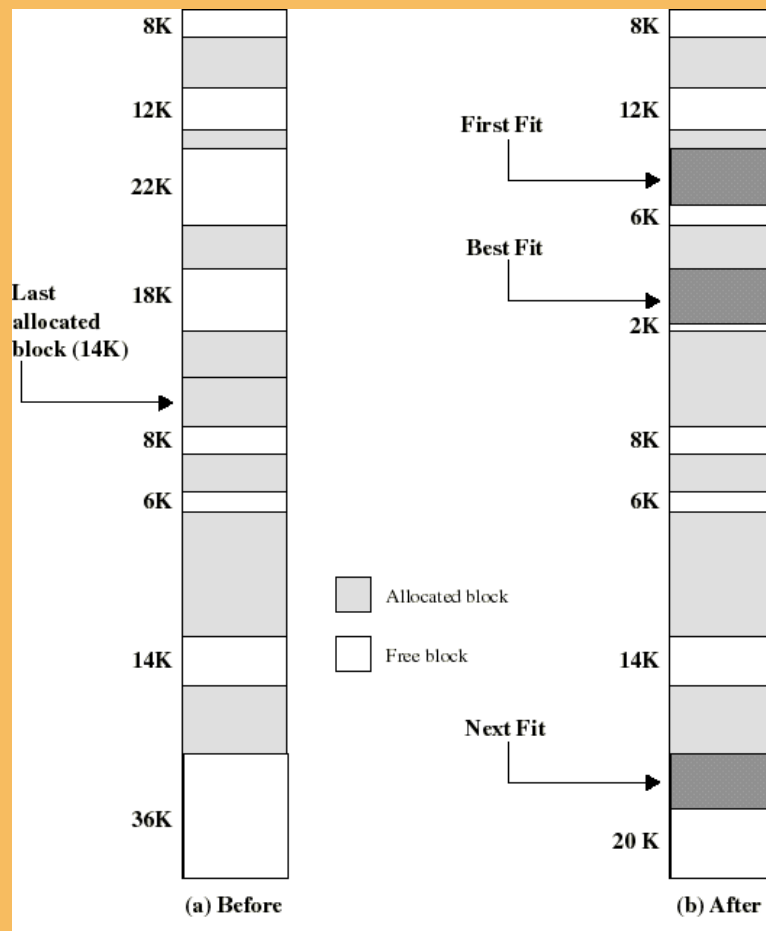
=> Chọn partition nhỏ nhất còn trống.



PHÂN CHIA ĐỘNG

- Số lượng partition và kích thước không cố định, có thể khác nhau.
- Mỗi process được cấp phát chính xác dung lượng bộ nhớ cần thiết.



=> Gây hiện tượng phân mảnh ngoại.



PHÂN CHIA ĐỘNG

Chiến lược placement (giảm chi phí compaction):

- *Best-fit*: chọn khối nhớ trống phù hợp nhỏ nhất.
- *First-fit*: chọn khối nhớ trống phù hợp đầu tiên kể từ đầu bộ nhớ.
- *Next-fit*: Chọn khối nhớ trống phù hợp đầu tiên kể từ vị trí cấp phát cuối cùng.
- *Worst-fit*: chọn khối nhớ trống lớn nhất.



Giả sử bộ nhớ chính được cấp phát các phân vùng có kích thước là 600K, 500K, 200K, 300K (theo thứ tự), sau khi thực thi xong, các tiến trình có kích thước 212K, 417K, 112K, 426K (theo thứ tự) sẽ được cấp phát bộ nhớ như thế nào, nếu sử dụng: Thuật toán First fit, Best fit, Next fit, Worst fit?

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



Thứ tự vùng nhớ: 600K, 500K, 200K, 300K

Thứ tự tiến trình: 212K, 417K, 112K, 426K



Best-fit: chọn khối nhớ trống phù hợp nhỏ nhất.

212K

300K

417K

500K

112K

200K

426K

600K

Thứ tự vùng nhớ: 600K, 500K, 200K, 300K

Thứ tự tiến trình: 212K, 417K, 112K, 426K



First-fit: chọn khối nhớ trống phù hợp đầu tiên kể từ đầu bộ nhớ.

212K

600K

417K

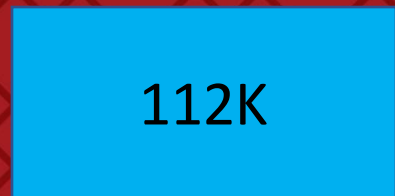
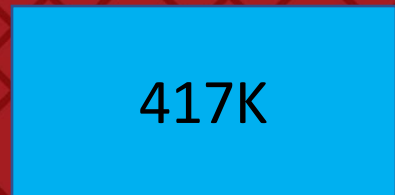
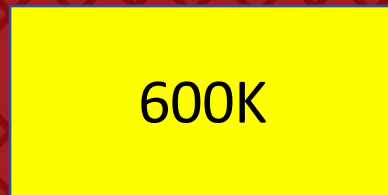
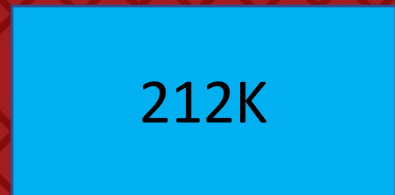
500K

112K

200K

426K

300K



Thứ tự vùng nhớ: 600K, 500K, 200K, 300K

Thứ tự tiến trình: 212K, 417K, 112K, 426K



Next-fit: Chọn khối nhớ trống phù hợp đầu tiên kể từ vị trí cấp phát cuối cùng.

212K

600K

417K

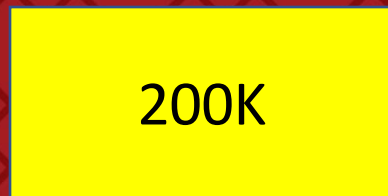
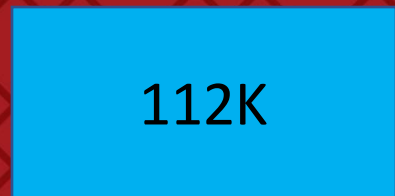
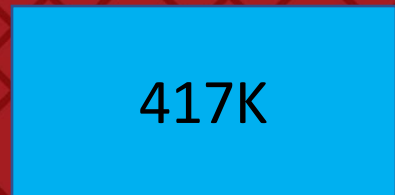
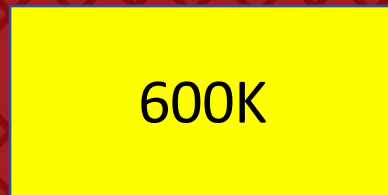
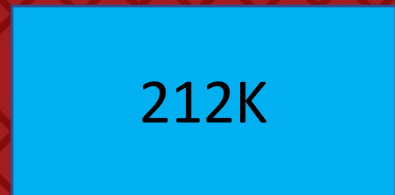
500K

112K

200K

426K

300K



Thứ tự vùng nhớ: 600K, 500K, 200K, 300K

Thứ tự tiến trình: 212K, 417K, 112K, 426K



Worst-fit: chọn khối nhớ trống lớn nhất.

212K

600K

417K

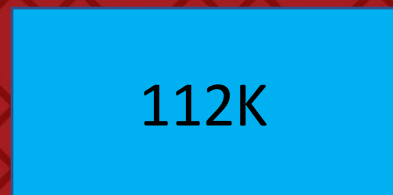
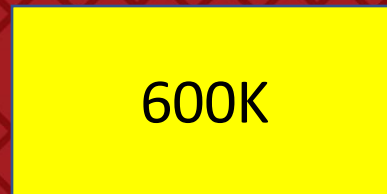
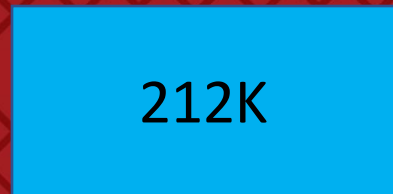
500K


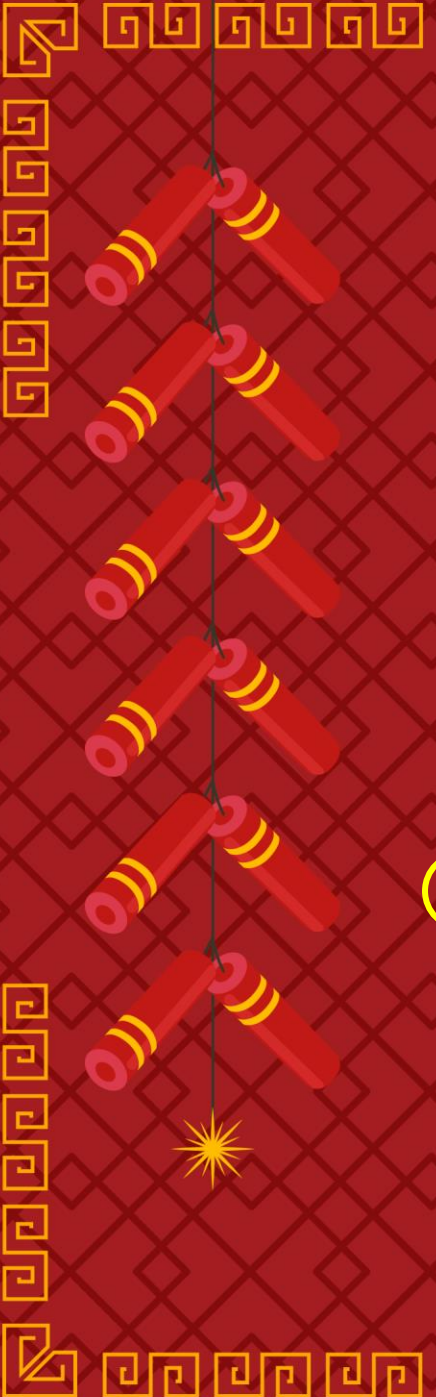
112K

300K


426K

200K





Giả sử bộ nhớ chính được phân chia thành các vùng nhớ cố định thứ tự như sau: 1 (200KB), 2 (180KB), 3 (400KB), 4 (220KB), 5 (360KB). Biết con trỏ đang ở vùng nhớ thứ 2, vùng nhớ thứ 2 đã được cấp phát, các vùng nhớ khác vẫn còn trống. Hỏi tiến trình P có kích thước 210KB sẽ được cấp phát cho vùng nhớ nào, nếu dùng giải thuật first-fit?

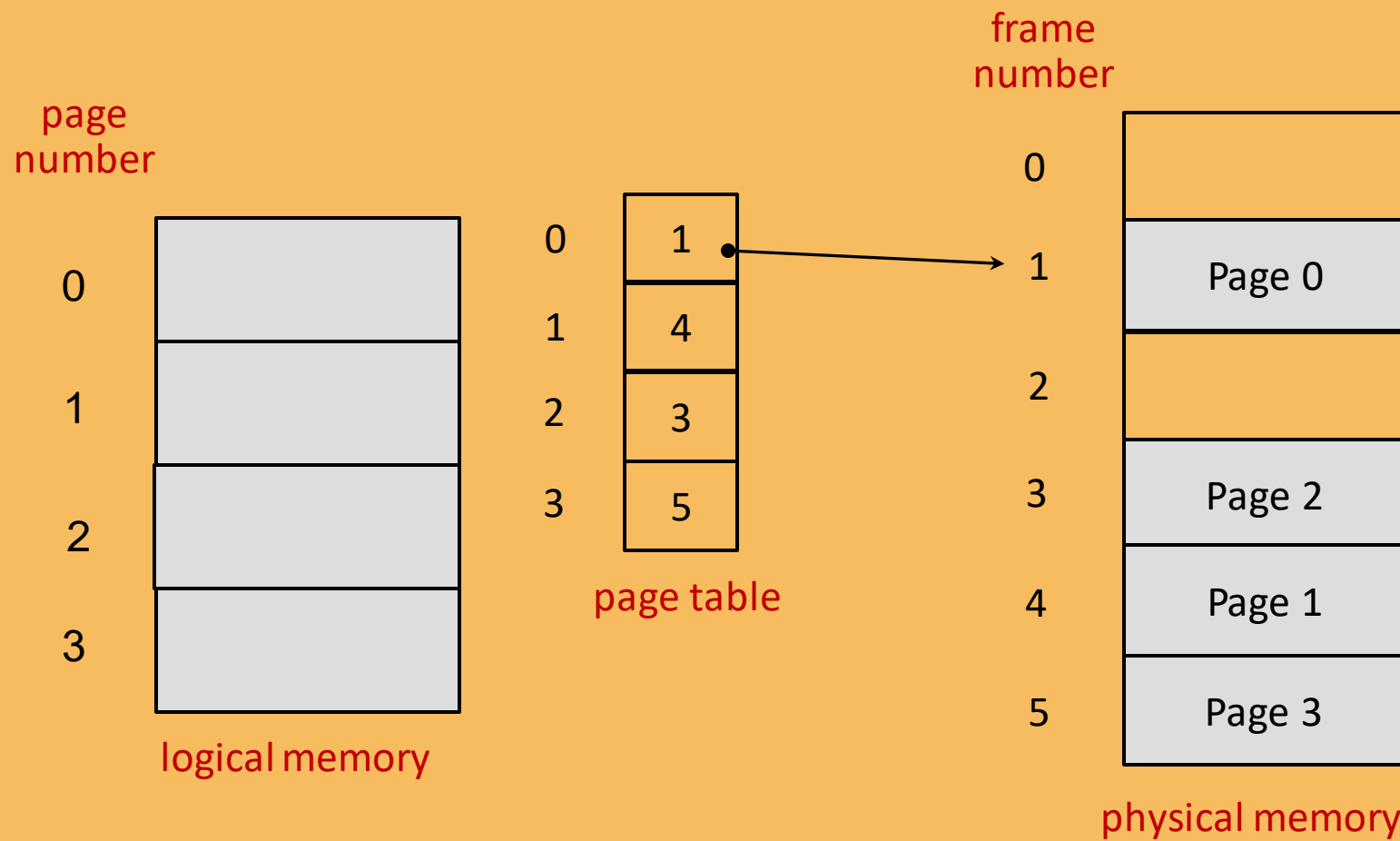
- ☒ A. 3
 - ☐ B. 1
 - ☐ C. 4
 - ☐ D. 2
- 



PHÂN TRANG ĐƠN GIẢN

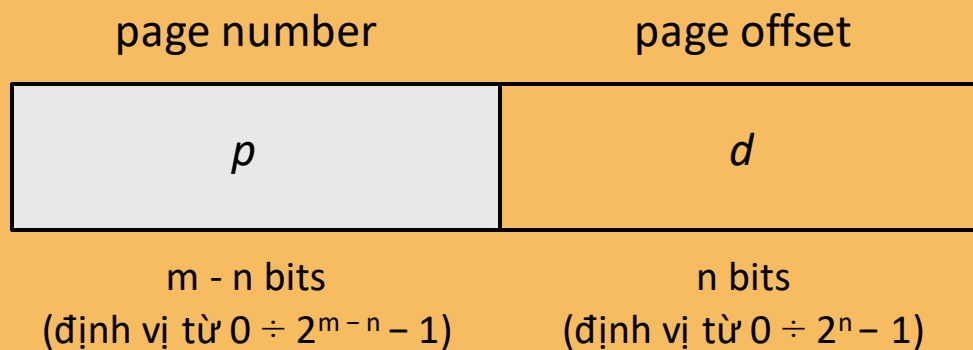
- Chia bộ nhớ vật lý thành các khối nhỏ có kích thước bằng nhau → khung trang (frame). Kích thước của frame là lũy thừa của 2 (từ khoảng 512 byte đến 16MB).
- Chia bộ nhớ luận lý của tiến trình thành các khối nhỏ có kích thước bằng nhau → trang (page).
- Bảng phân trang (page table) để ánh xạ địa chỉ luận lý thành địa chỉ thực

CHUYỂN ĐỔI ĐỊA CHỈ TRONG PAGING

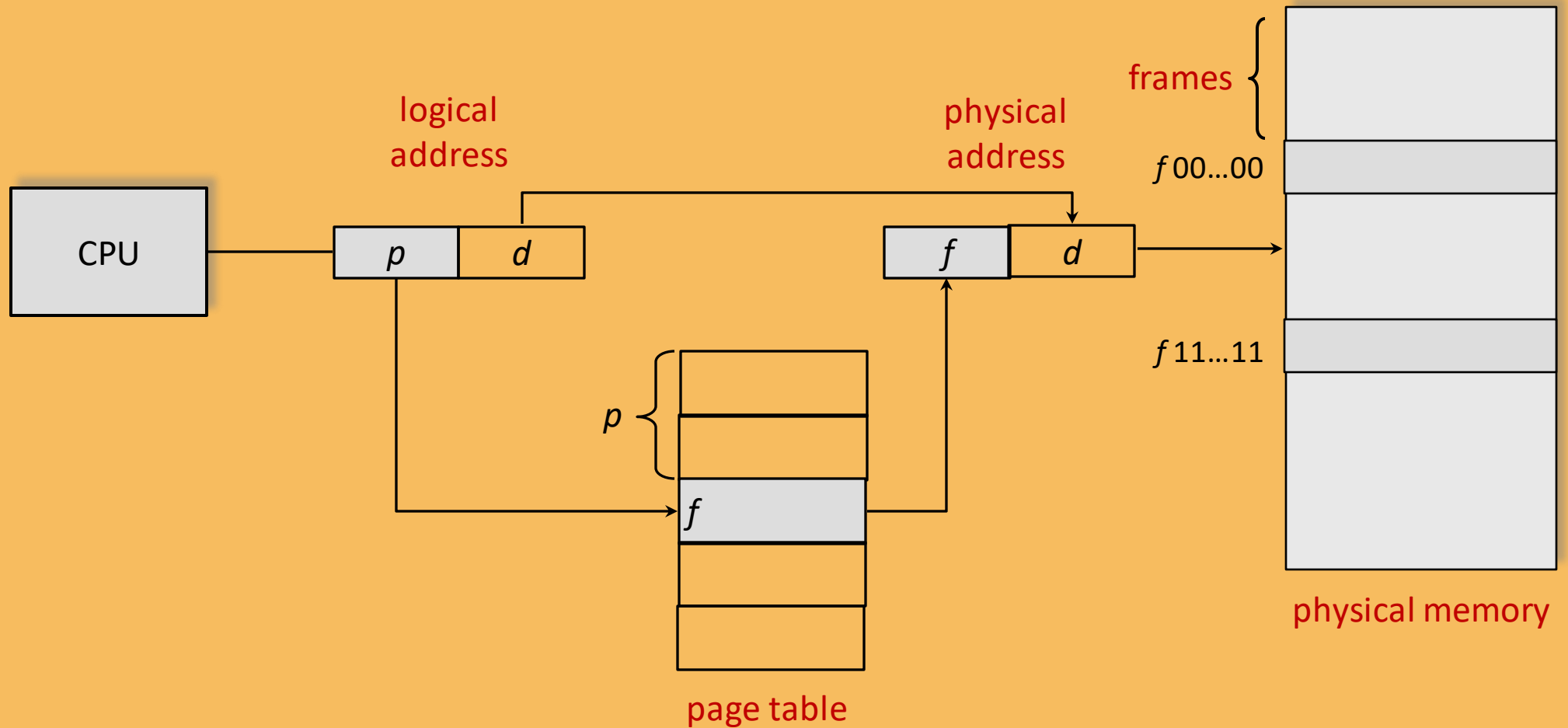


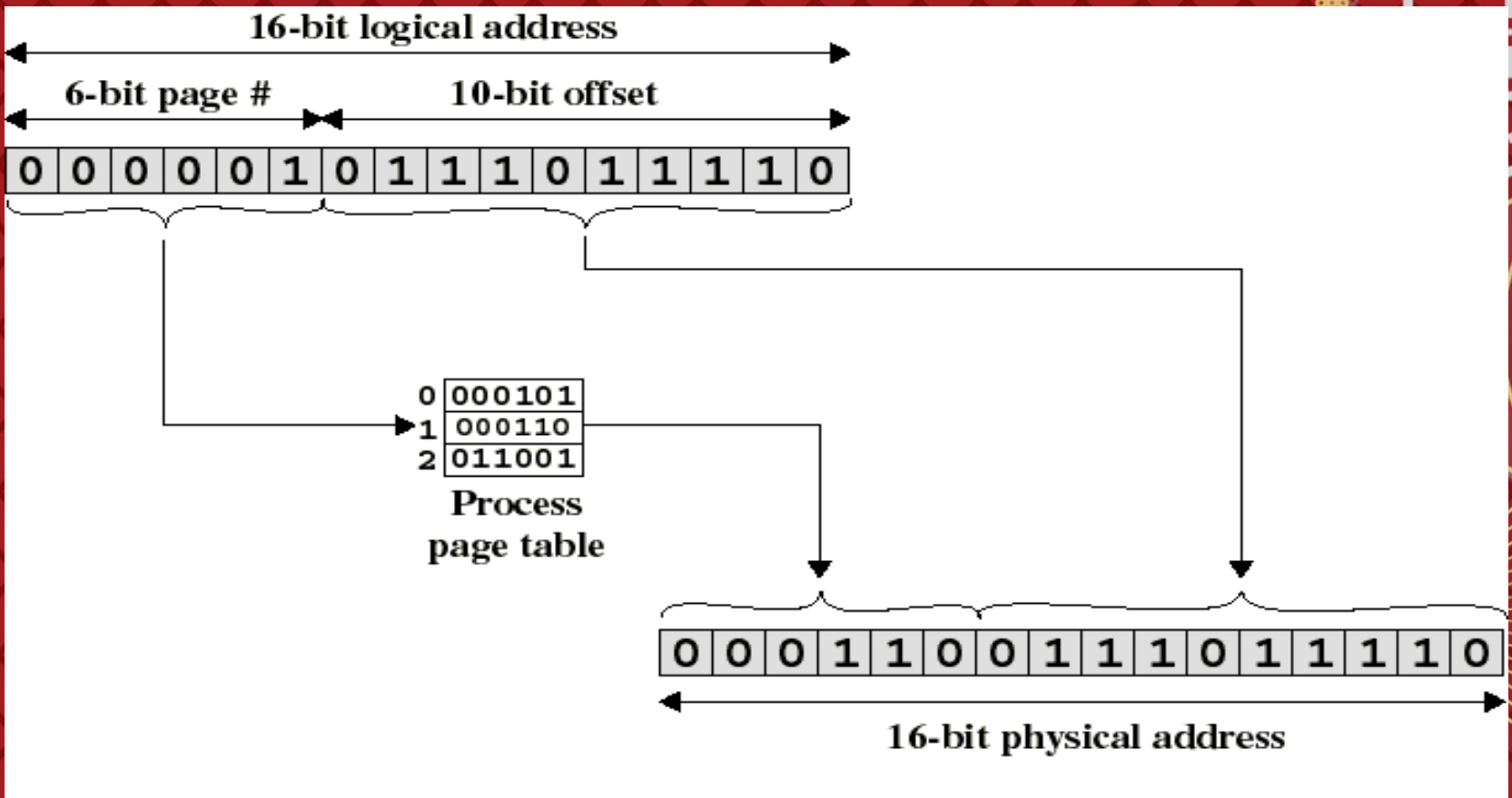
PHÂN TRANG ĐƠN GIẢN

- Địa chỉ luận lý gồm có:
 - Số hiệu trang (page number)
 - Độ dời của địa chỉ tính từ đầu trang (page offset)
- Nếu kích thước của không gian địa chỉ ảo là 2^m , và kích thước của trang là 2^n (đơn vị là byte hay word tùy theo kiến trúc máy) thì bảng trang sẽ có tổng cộng $2^m/2^n = 2^{m-n}$ mục (entry)



CHUYỂN ĐỔI ĐỊA CHỈ TRONG PAGING







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

- a) Địa chỉ luận lý gồm bao nhiêu bit?
- b) Địa chỉ vật lý gồm bao nhiêu bit?
- c) Bảng trang có bao nhiêu mục? Mỗi mục trong bảng trang cần bao nhiêu bit?

- a) $\text{ĐC}_{LL} = \text{ĐC}_{\text{page}} + \text{ĐC}_{\text{offset}} = 8 * 1024 = 2^3 * 2^{10} = 2^{13}$
 $\Rightarrow 13 \text{ bit}$
- b) $\text{ĐC}_{VL} = \text{ĐC}_{\text{frame}} + \text{ĐC}_{\text{offset}} = 32 * 1024 = 2^5 * 2^{10} = 2^{15}$
 $\Rightarrow 15 \text{ bit}$
- c) Số mục trong bảng phân trang = Số trang = 8
Số bit của mỗi mục trong bảng trang = Số bit của 1 khung trang = 5 bit



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

- a) Địa chỉ bộ nhớ vật lý có bao nhiêu bit?
- b) Địa chỉ bộ nhớ luận lý có bao nhiêu bit?
- c) Có bao nhiêu mục trong bảng phân trang? Mỗi mục chứa bao nhiêu bit?

- a) $\text{ĐCVL} = \text{ĐC frame} + \text{ĐC offset} = 32 * 2048 = 2^5 * 2^{11} = 2^{16}$
 $\Rightarrow 16 \text{ bit}$
- b) $\text{ĐCLL} = \text{ĐC page} + \text{ĐC offset} = 64 * 2048 = 2^6 * 2^{11} = 2^{17}$
 $\Rightarrow 17 \text{ bit}$
- c) Số mục trong bảng phân trang = Số trang = 64
Số bit của mỗi mục trong bảng trang = Số bit của 1 khung trang = 5 bit

a) Địa chỉ vật lý 6568 sẽ được chuyển thành địa chỉ ảo bao nhiêu? Biết rằng kích thước mỗi frame là 1K bytes
b) Địa chỉ ảo 3254 sẽ được chuyển thành địa chỉ vật lý bao nhiêu? Biết rằng kích thước mỗi frame là 2K bytes

a) Kích thước frame = 1024 bytes
 $f = 6568 / 1024 = 6$ (Lấy phần nguyên)
Từ bảng trang, ta có: $p = 0$
 $d = 6568 \% 1024 = 424$ (lấy phần dư)
ĐC ảo = $0 * 1024 + 424 = 424$

b) Kích thước frame = 2048 bytes
 $p = 3254 / 2048 = 1$ (Lấy phần nguyên)
Từ bảng trang, ta có: $f = 4$
 $d = 3254 \% 2048 = 1206$ (lấy phần dư)
ĐC thực = $4 * 2048 + 1206 = 9398$

0	6
1	4
2	5
3	7
4	1
5	9

Bảng trang của P1

1
1024
1025
2048

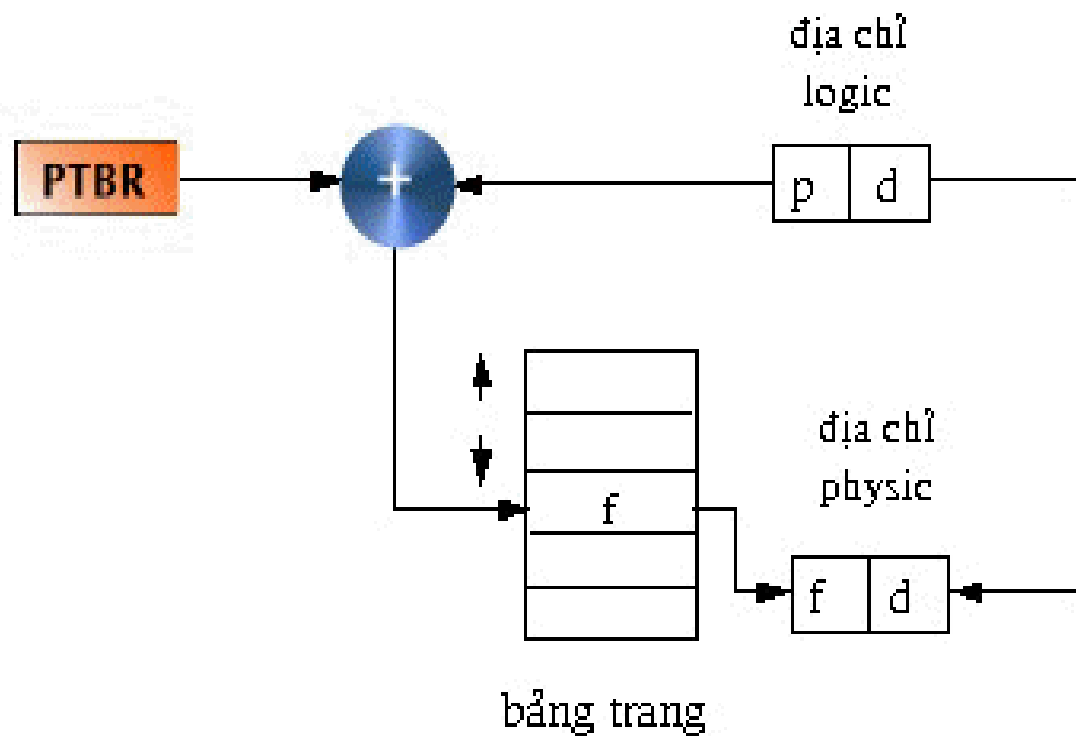
6145
8192



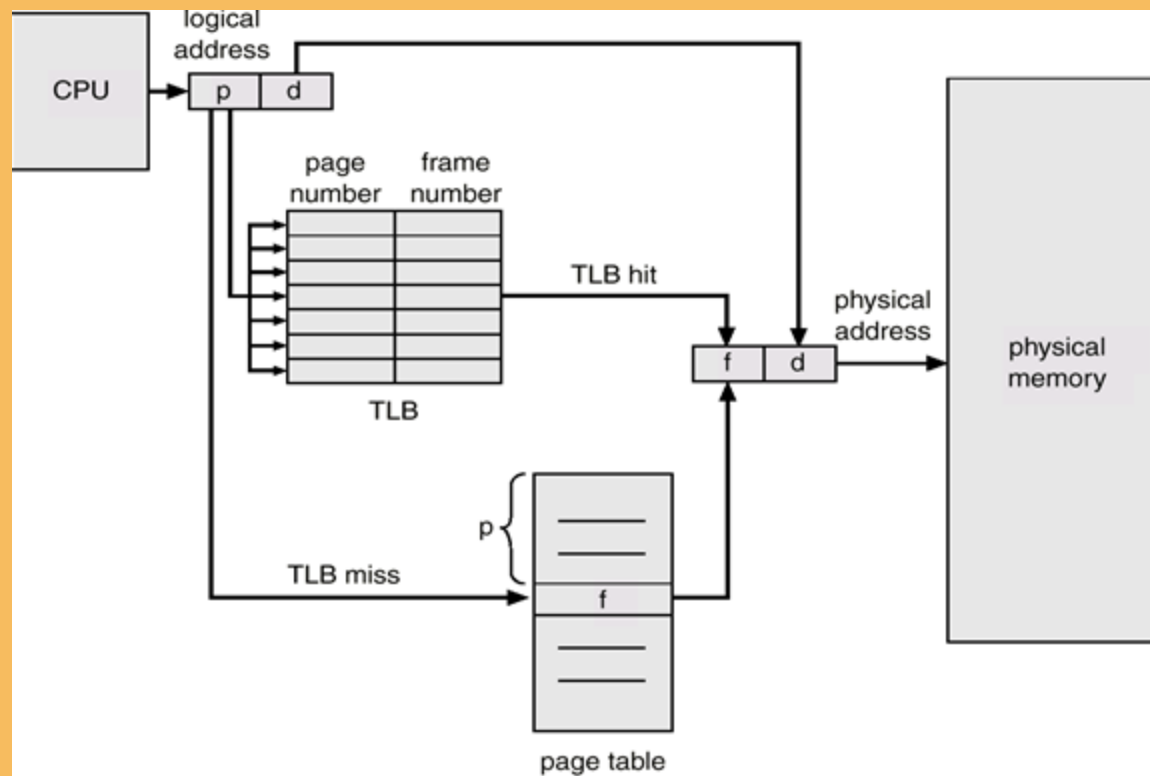
CÀI ĐẶT BẢNG TRANG

- Bảng phân trang được lưu trữ trong bộ nhớ chính
 - Mỗi process được hệ điều hành cấp một bảng phân trang.
 - Thanh ghi page-table base (PTBR) trỏ đến bảng phân trang.
 - Thanh ghi page-table length (PTLR) biểu thị kích thước của bảng phân trang (có thể dùng trong cơ chế bảo vệ bộ nhớ).
- Có hai cách cài đặt là PTBR và TLB.

CÀI ĐẶT BẢNG TRANG



Dùng thanh ghi PTBR



Dùng TLB


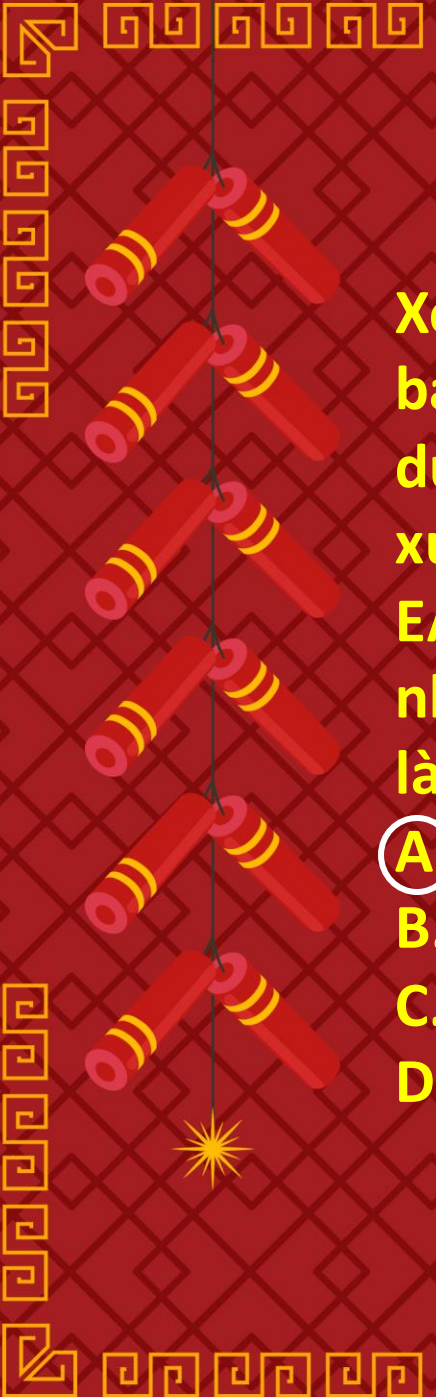


TRANSLATION LOOK-ASIDE BUFFERS (TLBS)

- Thời gian truy xuất hiệu dụng (Effective Access Time - EAT)
 - Thời gian tìm kiếm trong TLB: ϵ
 - Thời gian truy xuất bộ nhớ: x
 - Hit ratio: α
- Thời gian cần thiết để có được chỉ số frame:
 - Khi chỉ số trang có trong TLB (hit): $\epsilon + x$
 - Khi chỉ số trang không có trong TLB (miss): $\epsilon + x + x$

$$EAT = (\epsilon + x) \alpha + (\epsilon + 2x)(1 - \alpha) = (2 - \alpha) x + \epsilon$$





Xét một hệ thống dùng kỹ thuật phân trang với bảng trang được lưu vào bộ nhớ chính. Nếu sử dụng TLBs với hit ratio $\alpha = 0.85$ thì thời gian truy xuất bộ nhớ trong hệ thống (effective access time) $EAT = 230\text{ns}$. Biết thời gian một chu kỳ truy xuất bộ nhớ là $x = 180\text{ns}$. Hỏi thời gian tìm kiếm trong TLBs là bao nhiêu?

- ☒ A. 23ns
- ☐ B. 27ns
- ☐ C. 153ns
- ☐ D. 207ns

$$\begin{aligned} EAT &= (2 - \alpha) x + \epsilon \\ \Leftrightarrow 230 &= (2 - 0.85) 180 + \epsilon \\ \Leftrightarrow \epsilon &= 23\text{ns} \end{aligned}$$





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

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

b. Nếu sử dụng TLBs với hit-ratio là 75%, thời gian để tìm trong TLBs xem như bằng 0, tính thời gian truy xuất bộ nhớ trong hệ thống.

a) Một lần truy xuất bộ nhớ bình thường là 200ns.
Một thao tác thực hiện 2 lần truy xuất.
Vậy thời gian là $2 \times 200 = 400\text{ns}$.

b)
$$\begin{aligned} \text{EAT} &= (2 - \alpha) x + \varepsilon \\ &= (2 - 0.75) 200 + 0 \\ &= 250\text{ns} \end{aligned}$$

BẢO VỆ BỘ NHỚ

Việc bảo vệ bộ nhớ được hiện thực bằng cách gắn với frame các bit bảo vệ (proctection bits) được giữ trong bảng phân trang. Các biến này được biểu diễn: **read -only, read-write, execute-only.**

CƠ CHẾ HOÁN VỊ (SWAPPING)

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





Chương 8:
BỘ NHỚ ẢO



TỔNG QUAN BỘ NHỚ ẢO

Bộ nhớ ảo (virtual memory): Bộ nhớ ảo là một kỹ thuật cho phép xử lý một tiến trình không được nạp toàn bộ vào bộ nhớ vật lý.

Ưu điểm của bộ nhớ ảo:

- Số lượng process trong bộ nhớ nhiều hơn.
- Một process có thể thực thi ngay cả khi kích thước của nó lớn hơn bộ nhớ thực.
- Giảm nhẹ công việc của lập trình viên.

Có hai kỹ thuật cài đặt bộ nhớ ảo:

- Phân trang theo yêu cầu (Demand Paging)
- Phân đoạn theo yêu cầu (Demand Segmentation)



Lựa chọn nào dưới đây **KHÔNG** phải là ưu điểm của bộ nhớ ảo?

- A. Số lượng tiến trình trong bộ nhớ nhiều hơn.
- B. Một tiến trình có thể thực thi ngay cả khi kích thước của nó lớn hơn bộ nhớ thực.
- ☒ C. Giảm thời gian truy xuất bộ nhớ.
- D. Giảm nhẹ công việc của lập trình viên



PHÂN TRANG THEO YÊU CẦU

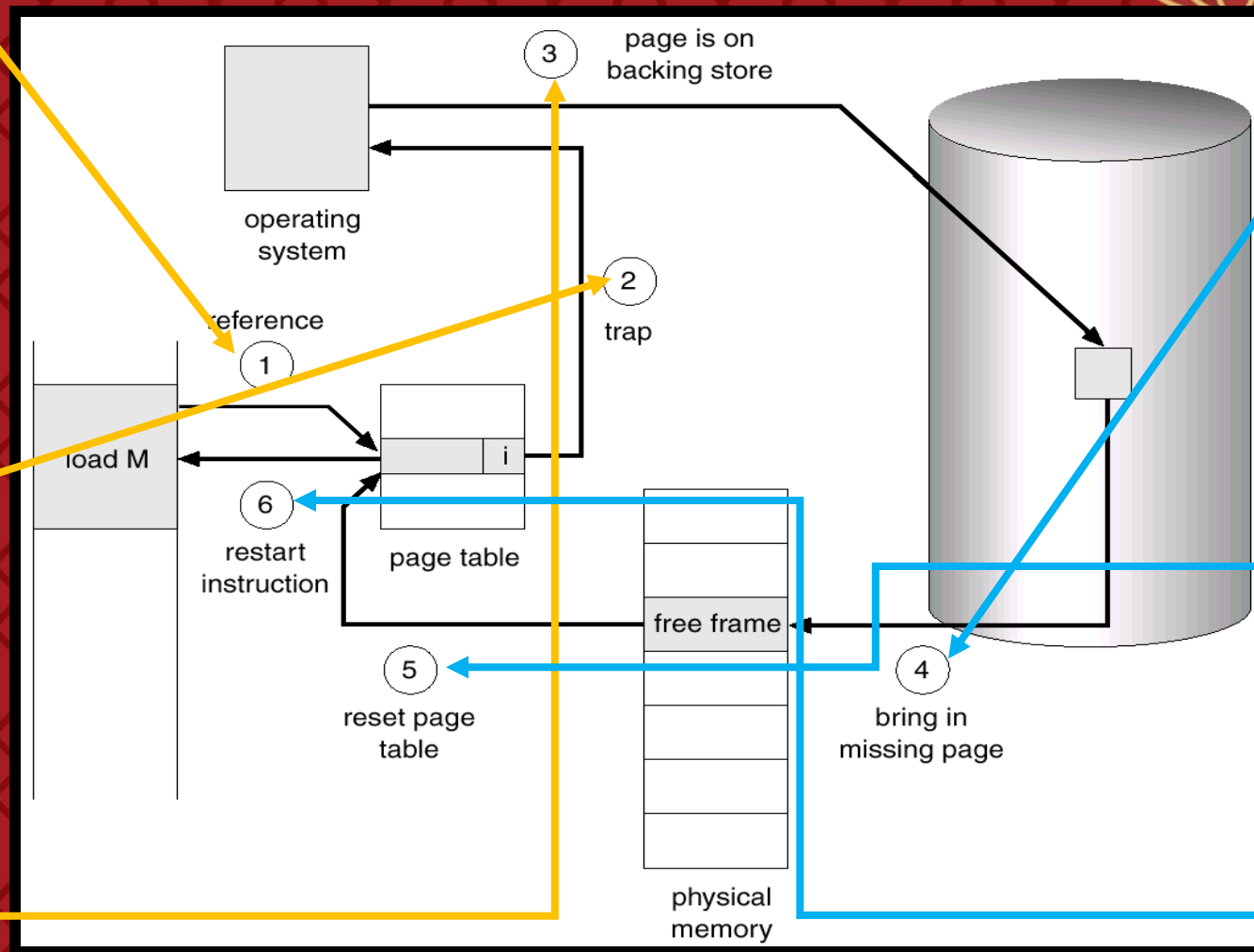
- Các trang của tiến trình chỉ được nạp vào bộ nhớ chính khi được yêu cầu.
- **Page-fault:** khi tiến trình tham chiếu đến một trang không có trong bộ nhớ chính (valid bit) thì phần cứng sẽ gây ra một lỗi trang (page-fault).



1. Chương trình tham chiếu đến một trang không có trong bộ nhớ chính.

2. Phần cứng sẽ gây ra một ngắt (page-fault trap) khởi động dịch vụ page-fault service routine (PFSR) của Hệ điều hành

3. Trang mà chương trình cần đang nằm trên bộ nhớ thứ cấp (bộ nhớ tạm, đĩa,...)



4. Tìm một frame trống và load trang vào frame trống ấy.
- Hoặc sử dụng các giải thuật thay thế trang (FIFO, LRU, OPT).

5. Cập nhật lại page table và frame table tương ứng.

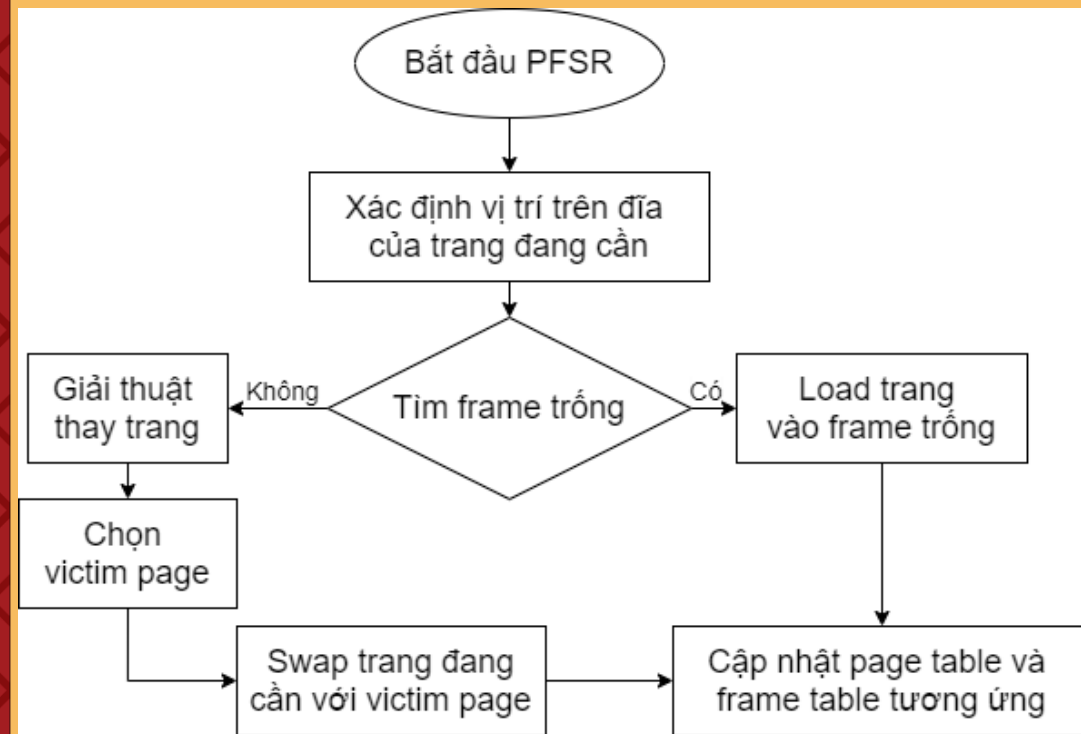
6. Khởi động lại process khi đã có nội dung page mà nó yêu cầu.

Các bước xử lý lỗi trang



PAGE-FAULT SERVICE ROUTINE (PFSR)

- Chuyển process về trạng thái blocked
- Phát ra một yêu cầu đọc đĩa để nạp trang được tham chiếu vào một frame trống; trong khi đợi I/O, một process khác được cấp CPU để thực thi.
- Sau khi I/O hoàn tất, đĩa gây ra một ngắt đến hệ điều hành; PFSR cập nhật page table và chuyển process về trạng thái ready.





GIẢI THUẬT THAY TRANG

Dữ liệu cần biết:

- Số khung trang
- Tình trạng ban đầu
- Chuỗi tham chiếu

Nghịch lý Belady: số page fault tăng mặc dù tiến trình đã được cấp nhiều frame hơn.

	LRU	FIFO	OPT
Chọn trang	Trang ít được tham chiếu nhất	Trang ở trong bộ nhớ lâu nhất	Trang sẽ được tham chiếu trễ nhất trong tương lai
Nhược điểm	Đòi hỏi sự trợ giúp từ phần cứng để sắp xếp thứ tự các trang theo thời điểm tham chiếu	Cài đặt đơn giản nhưng dễ mắc phải nghịch lý Belady	Khó hiện thực vì không thể dự đoán chính xác thời điểm một trang được tham chiếu



Giả sử có 3 khung trang và các khung trang ban đầu là rỗng. Xác định số Page Fault sử dụng FIFO, LRU, OPT?

0, 2, 1, 6, 4, 0, 1, 0, 3, 1, 2, 1

LRU: Trang ít được tham chiếu nhất

0	2	1	6	4	0	1	0	3	1	2	1
0	0	0	6	6	6	1	1	1	1	1	1
	2	2	2	4	4	4	4	3	3	3	3
		1	1	1	0	0	0	0	0	2	2
x	x	x	x	x	x	x		x		x	



FIFO: Trang ở trong bộ nhớ lâu nhất

0	2	1	6	4	0	1	0	3	1	2	1
0	0	0	6	6	6	1	1	1	1	1	1
	2	2	2	4	4	4	4	3	3	3	3
		1	1	1	0	0	0	0	0	2	2
x	x	x	x	x	x	x		x		x	

OPT: Trang sẽ được tham chiếu trễ nhất trong tương lai

0	2	1	6	4	0	1	0	3	1	2	1
0	0	0	0	0	0	0	0	3	3	3	3
	2	2	6	4	4	4	4	4	4	2	2
		1	1	1	1	1	1	1	1	1	1
x	x	x	x	x				x		x	



Xét chuỗi truy xuất bộ nhớ sau:

1, 2, 3, 4, 2, 1, 5, 6, 2, 1, 2, 3, 7, 6, 3, 2, 1, 2, 3, 6

Có bao nhiêu lỗi trang xảy ra khi sử dụng các thuật toán thay thế sau đây, giả sử có lần lượt là 2, 3, 4, 5 khung trang.

- a. LRU
- b. FIFO
- c. Chiến lược tối ưu (OPT)



LRU với 2 khung trang, 18 lỗi trang

1	2	3	4	2	1	5	6	2	1	2	3	7	6	3	2	1	2	3	6
1	1	3	3	2	2	5	5	2	2	2	2	7	7	3	3	1	1	3	3
	2	2	4	4	1	1	6	6	1	1	3	3	6	6	2	2	2	2	6
x	x	x	x	x	x	x	x	x	x		x	x	x	x	x	x		x	x

LRU với 3 khung trang, 15 lỗi trang

1	2	3	4	2	1	5	6	2	1	2	3	7	6	3	2	1	2	3	6
1	1	1	4	4	4	5	5	5	1	1	1	7	7	7	2	2	2	2	2
	2	2	2	2	2	2	6	6	6	6	3	3	3	3	3	3	3	3	3
		3	3	3	1	1	1	2	2	2	2	2	6	6	6	1	1	1	6
x	x	x	x		x	x	x	x	x		x	x	x		x	x			x



LRU với 4 khung trang, 10 lỗi trang

1	2	3	4	2	1	5	6	2	1	2	3	7	6	3	2	1	2	3	6
1	1	1	1	1	1	1	1	1	1	1	1	1	6	6	6	6	6	6	6
	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2
		3	3	3	3	5	5	5	5	5	3	3	3	3	3	3	3	3	3
			4	4	4	4	6	6	6	6	6	7	7	7	7	1	1	1	1
x	x	x	x			x	x				x	x	x			x			

LRU với 5 khung trang, 8 lỗi trang

1	2	3	4	2	1	5	6	2	1	2	3	7	6	3	2	1	2	3	6
1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2
		3	3	3	3	3	6	6	6	6	6	6	6	6	6	6	6	6	6
			4	4	4	4	4	4	4	4	3	3	3	3	3	3	3	3	3
						5	5	5	5	5	5	7	7	7	7	7	7	7	7
x	x	x	x			x	x				x	x							



FIFO với 2 khung trang, 18 lỗi trang

1	2	3	4	2	1	5	6	2	1	2	3	7	6	3	2	1	2	3	6
1	1	3	3	2	2	5	5	2	2	2	3	3	6	6	2	2	2	3	3
	2	2	4	4	1	1	6	6	1	1	1	7	7	3	3	1	1	1	6
x	x	x	x	x	x	x	x	x	x		x	x	x	x	x	x		x	x

FIFO với 3 khung trang, 16 lỗi trang

1	2	3	4	2	1	5	6	2	1	2	3	7	6	3	2	1	2	3	6
1	1	1	4	4	4	4	6	6	6	6	3	3	3	3	2	2	2	2	6
	2	2	2	2	1	1	1	2	2	2	2	7	7	7	7	1	1	1	1
		3	3	3	3	5	5	5	1	1	1	1	6	6	6	6	6	3	3
x	x	x	x		x	x	x	x	x		x	x	x		x	x		x	x



FIFO với 4 khung trang, 14 lỗi trang

1	2	3	4	2	1	5	6	2	1	2	3	7	6	3	2	1	2	3	6
1	1	1	1	1	1	5	5	5	5	5	3	3	3	3	3	1	1	1	1
	2	2	2	2	2	2	6	6	6	6	6	7	7	7	7	7	7	3	3
		3	3	3	3	3	3	2	2	2	2	2	6	6	6	6	6	6	6
			4	4	4	4	4	4	1	1	1	1	1	1	2	2	2	2	2
x	x	x	x			x	x	x	x		x	x	x		x	x		x	

FIFO với 5 khung trang, 10 lỗi trang

1	2	3	4	2	1	5	6	2	1	2	3	7	6	3	2	1	2	3	6
1	1	1	1	1	1	1	6	6	6	6	6	6	6	6	6	6	6	6	6
	2	2	2	2	2	2	2	2	1	1	1	1	1	1	1	1	1	1	1
		3	3	3	3	3	3	3	3	2	2	2	2	2	2	2	2	2	2
			4	4	4	4	4	4	4	4	3	3	3	3	3	3	3	3	3
						5	5	5	5	5	5	7	7	7	7	7	7	7	7
x	x	x	x			x	x		x	x	x	x							

OPT với 2 khung trang, 15 lỗi trang

1	2	3	4	2	1	5	6	2	1	2	3	7	6	3	2	1	2	3	6
1	1	3	4	4	1	5	6	6	1	1	3	3	3	3	3	1	1	3	6
	2	2	2	2	2	2	2	2	2	2	2	7	6	6	2	2	2	2	2
x	x	x	x		x	x	x		x		x	x	x		x	x		x	x

OPT với 3 khung trang, 11 lỗi trang

1	2	3	4	2	1	5	6	2	1	2	3	7	6	3	2	1	2	3	6
1	1	1	1	1	1	1	1	1	1	1	3	3	3	3	3	3	3	3	6
	2	2	2	2	2	2	2	2	2	2	2	7	7	7	2	2	2	2	2
		3	4	4	4	5	6	6	6	6	6	6	6	6	6	1	1	1	1
x	x	x	x			x	x				x	x			x	x			x



OPT với 4 khung trang, 8 lỗi trang

1	2	3	4	2	1	5	6	2	1	2	3	7	6	3	2	1	2	3	6
1	1	1	1	1	1	1	1	1	1	1	1	7	7	7	7	1	1	1	1
	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2
		3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3
			4	4	4	5	6	6	6	6	6	6	6	6	6	6	6	6	6
x	x	x	x			x	x					x				x			

OPT với 5 khung trang, 7 lỗi trang

1	2	3	4	2	1	5	6	2	1	2	3	7	6	3	2	1	2	3	6
1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2
		3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3
			4	4	4	4	4	4	4	4	4	7	7	7	7	7	7	7	7
						5	6	6	6	6	6	6	6	6	6	6	6	6	6
x	x	x	x			x	x					x							




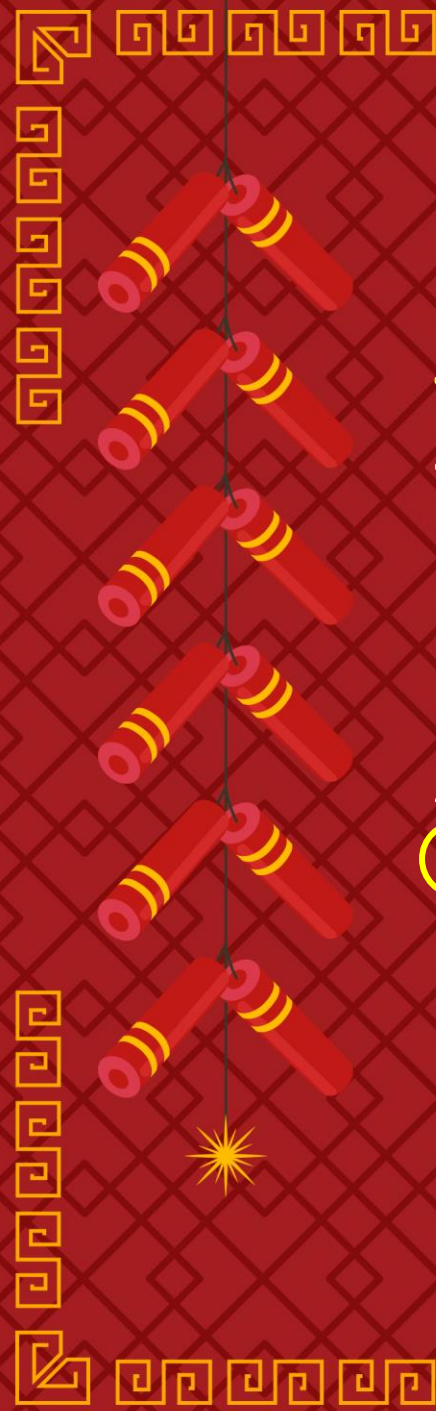
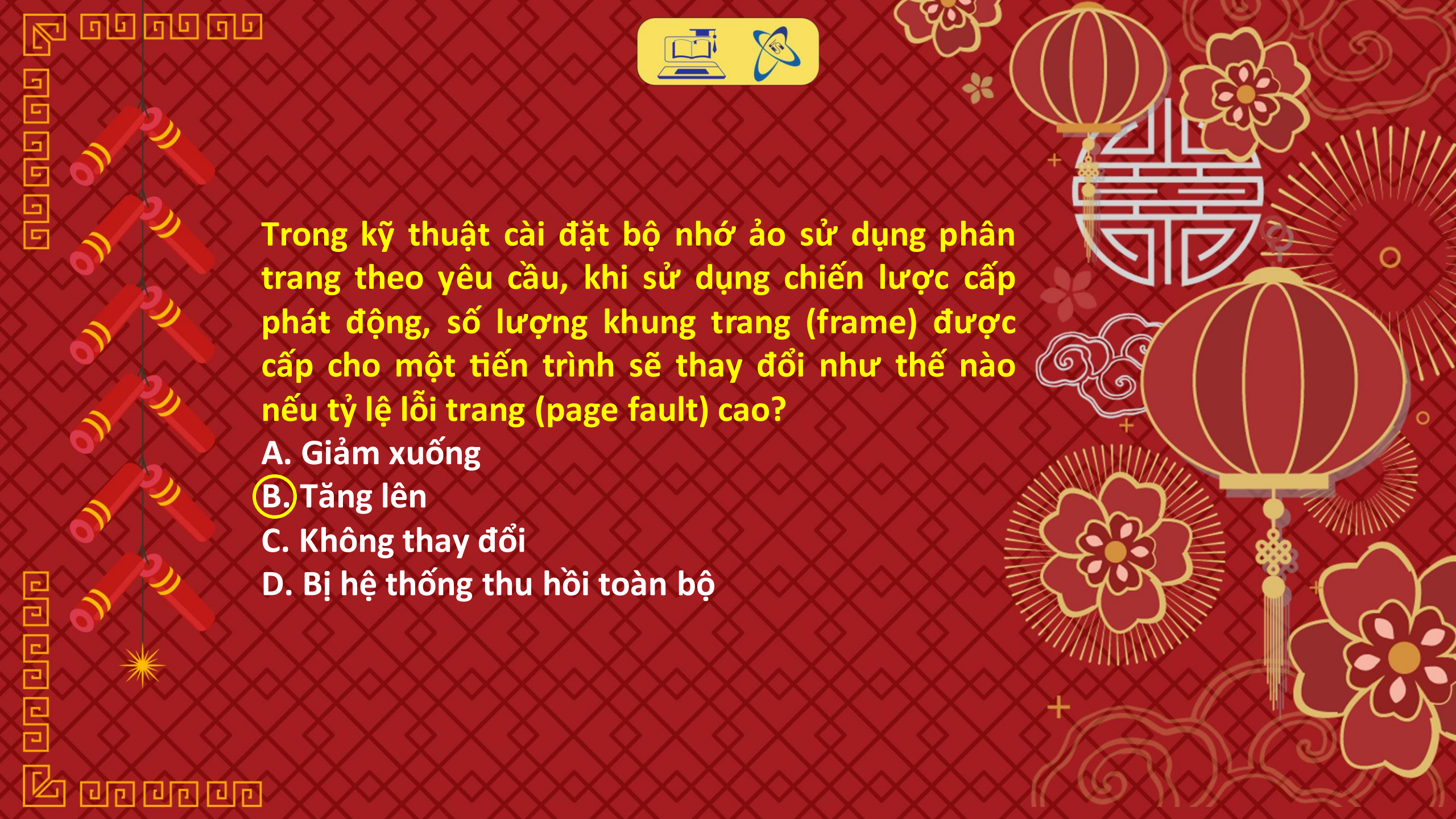
VẤN ĐỀ CẤP PHÁT FRAMES

Chiến lược cấp phát tĩnh (fixed-allocation)

Số frame cấp cho mỗi process không đổi, được xác định vào thời điểm loading và có thể tùy thuộc vào từng ứng dụng (kích thước của nó,...)

Chiến lược cấp phát động (variable-allocation)

- Số frame cấp cho mỗi process có thể thay đổi trong khi nó chạy
 - Nếu tỷ lệ page-fault cao \Rightarrow cấp thêm frame
 - Nếu tỷ lệ page-fault thấp \Rightarrow giảm bớt frame
- OS phải mất chi phí để ước định các process



Trong kỹ thuật cài đặt bộ nhớ ảo sử dụng phân trang theo yêu cầu, khi sử dụng chiến lược cấp phát động, số lượng khung trang (frame) được cấp cho một tiến trình sẽ thay đổi như thế nào nếu tỷ lệ lỗi trang (page fault) cao?

- A. Giảm xuống
- ☒ B. Tăng lên
- C. Không thay đổi
- D. Bị hệ thống thu hồi toàn bộ

