

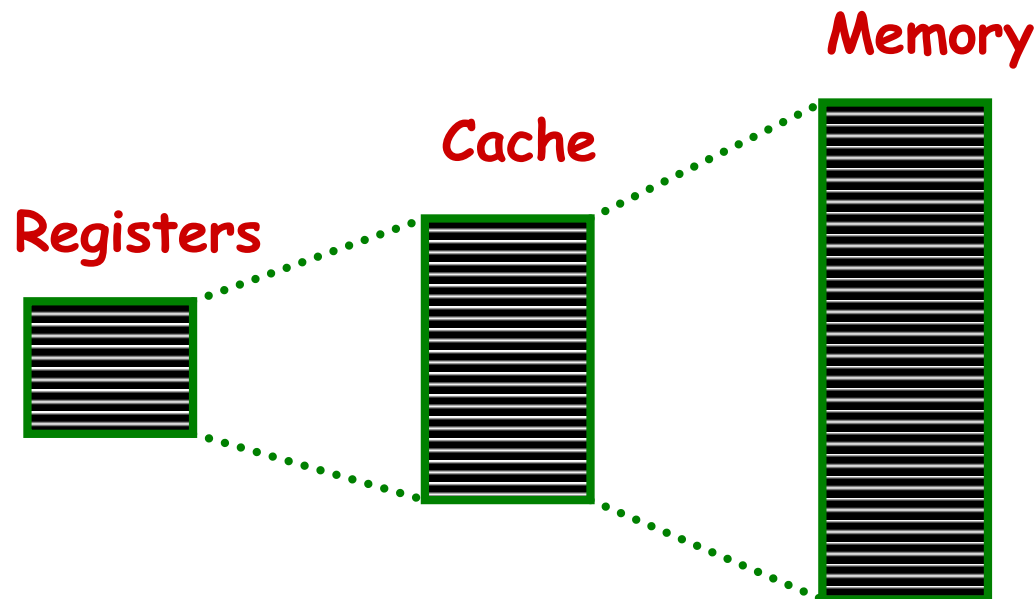


Bài giảng 7: Bộ nhớ Ảo

- Vấn đề với Real Memory
- Ý tưởng Virtual Memory
- Thực hiện Virtual Memory
- Các chiến lược của Virtual Memory
 - Chiến lược nạp
 - Chiến lược thay thế trang
 - Chiến lược cấp phát khung trang
- Hiện tượng thrashing
 - Nguyên nhân
 - Giải pháp

Các cấp bộ nhớ

- Cho đến nay : Nạp toàn bộ tiến trình vào bộ nhớ rồi thực hiện nó...
 - Nếu kích thước tiến trình lớn hơn dung lượng bộ nhớ chính ?



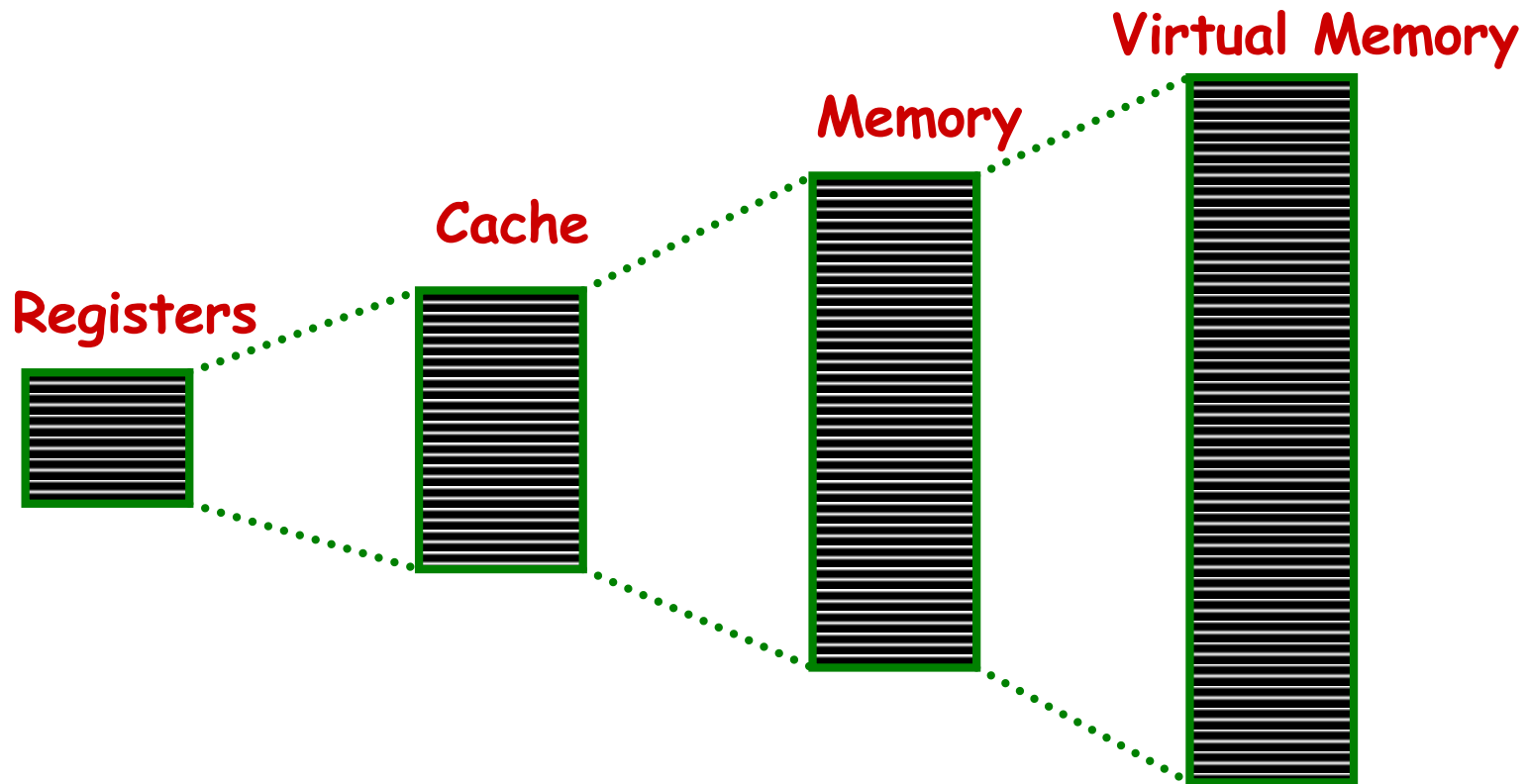


Giải pháp

- Tại một thời điểm chỉ có 1 chỉ thị được thi hành
 - Tại sao phải nạp tất cả tiến trình vào BNC cùng 1 lúc ?
- Ý tưởng
 - Cho phép **nạp và thi hành từng phần** tiến trình
 - Ai điều khiển việc thay đổi các phần được nạp và thi hành ?
 - Tại một thời điểm chỉ giữ trong BNC các chỉ thị và dữ liệu cần thiết tại thời điểm đó
 - Các phần khác của tiến trình nằm ở đâu ?
- Giải pháp → Bộ nhớ ảo (virtual memory)

Virtual Memory

Nếu có một **Virtual Memory** với dung lượng rất rất lớn cho LTV làm việc...
Hoan hô !





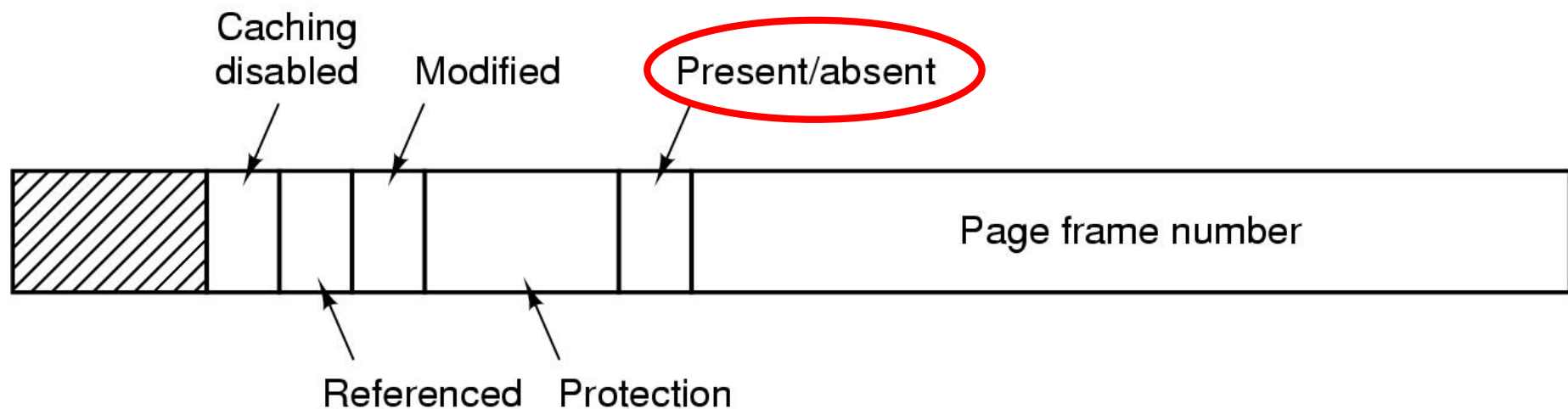
Ý tưởng

Tách biệt KGĐC và KGVL

- LTV : mỗi tiến trình làm việc với **KGĐC 2^m** của mình (địa chỉ từ 0 (2^m-1))
- HĐH : chịu trách nhiệm nạp **các KGĐC** vào **một KGVL chung**
- Giải pháp của HĐH : Nạp từng phần tiến trình
 - Phân chia KGĐC thành các phần ?
 - Paging/Segmentation
 - Mở rộng BNC để lưu trữ các phần của tiến trình chưa được nạp
 - Dùng BNP(disk) để mở rộng BNC
 - Nhận biết phần nào của KGĐC chưa được nạp ?
 - Bổ sung bit cờ hiệu để nhận dạng tình trạng của một page/segment là đã được nạp vào BNC hay chưa
 - Cơ chế chuyển đổi qua lại các phần của tiến trình giữa BNC và BNP
 - Swapping...

Virtual Memory với cơ chế phân trang (Paging)

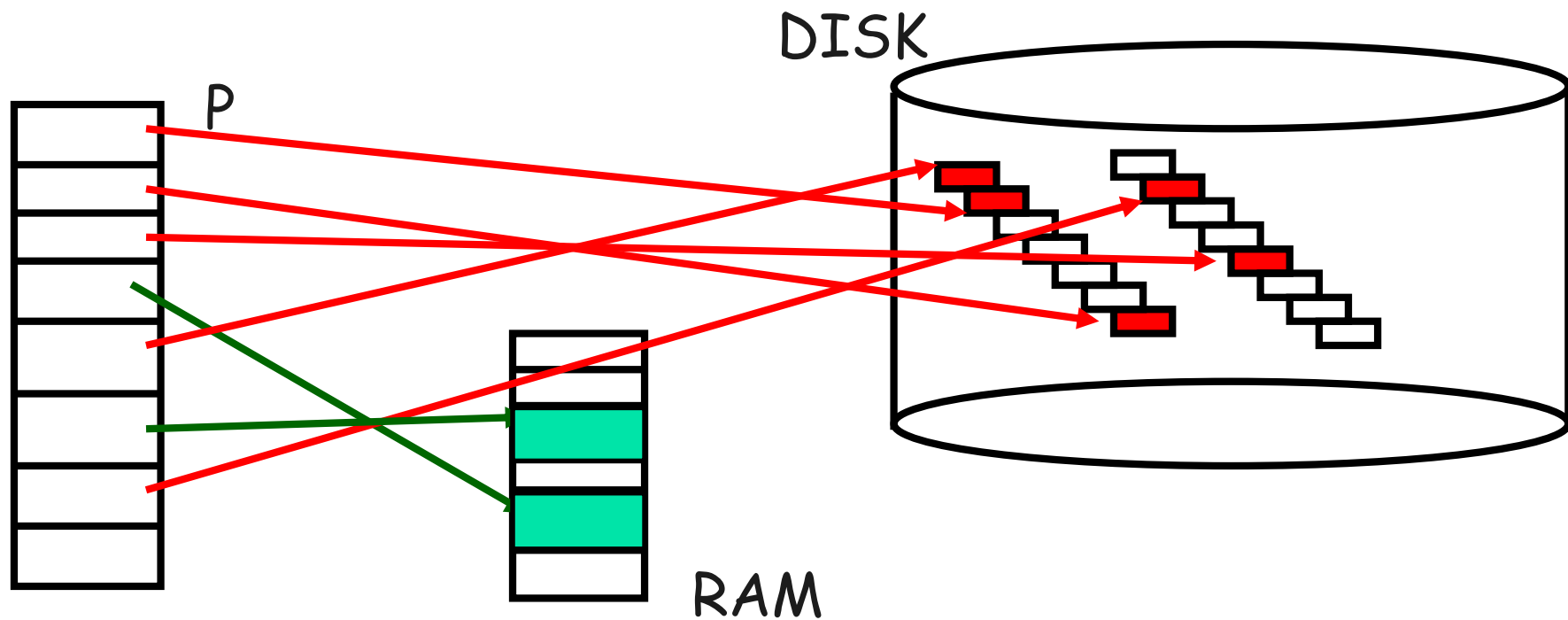
- Phân chia KGĐC thành các **page**
- Dùng BNP(disk) để mở rộng BNC, lưu trữ các phần của tiến trình chưa được nạp
- Bổ sung bit cờ hiệu trong Page Table để nhận dạng tình trạng một page đã được nạp vào BNC hay chưa .



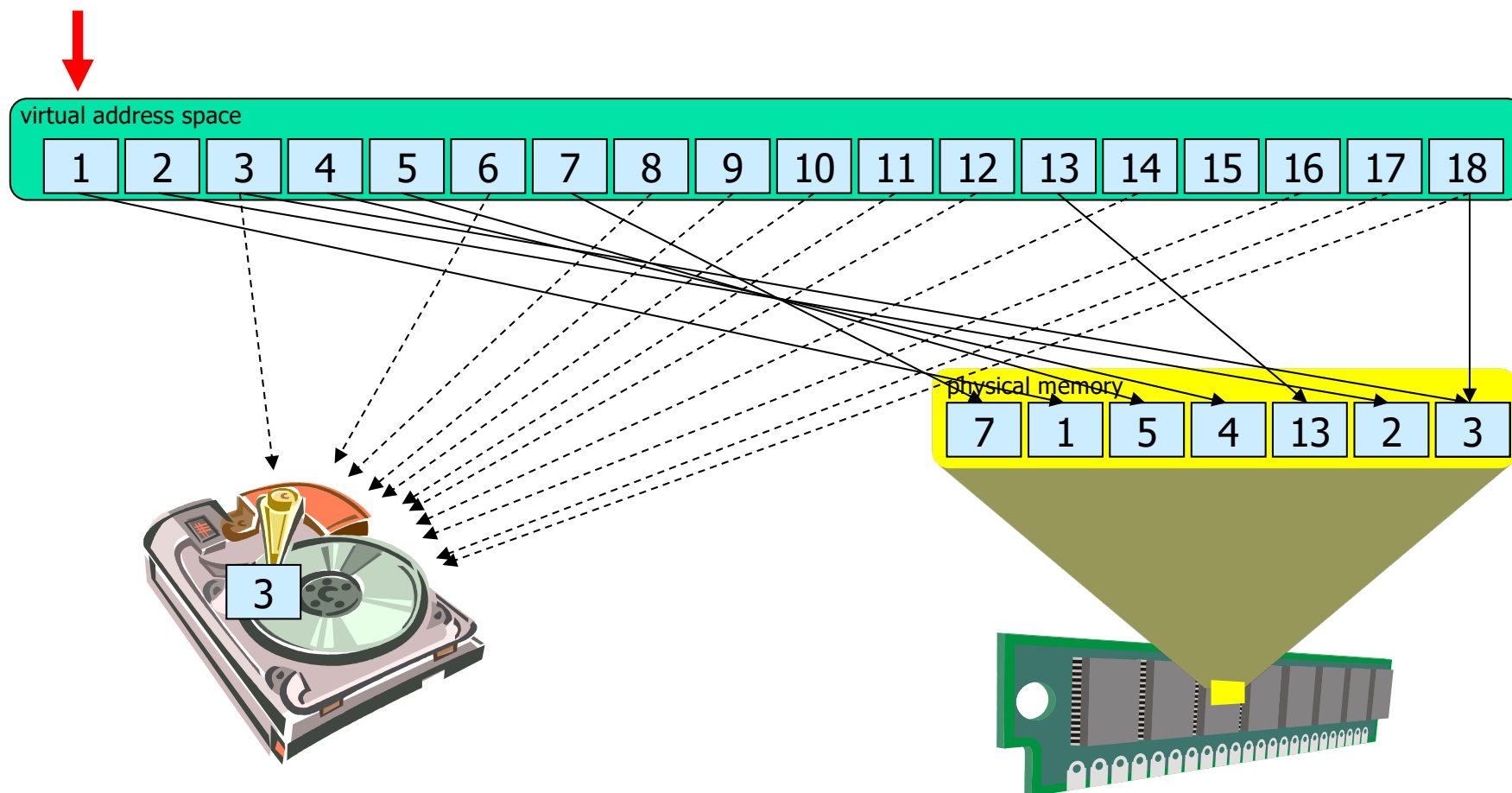
Cấu trúc một phần tử trong Page Tables

Lưu trữ KGĐC ở đâu ?

- Sử dụng bộ nhớ phụ để lưu trữ tạm thời các trang chưa sử dụng



Virtual Memory



Page table

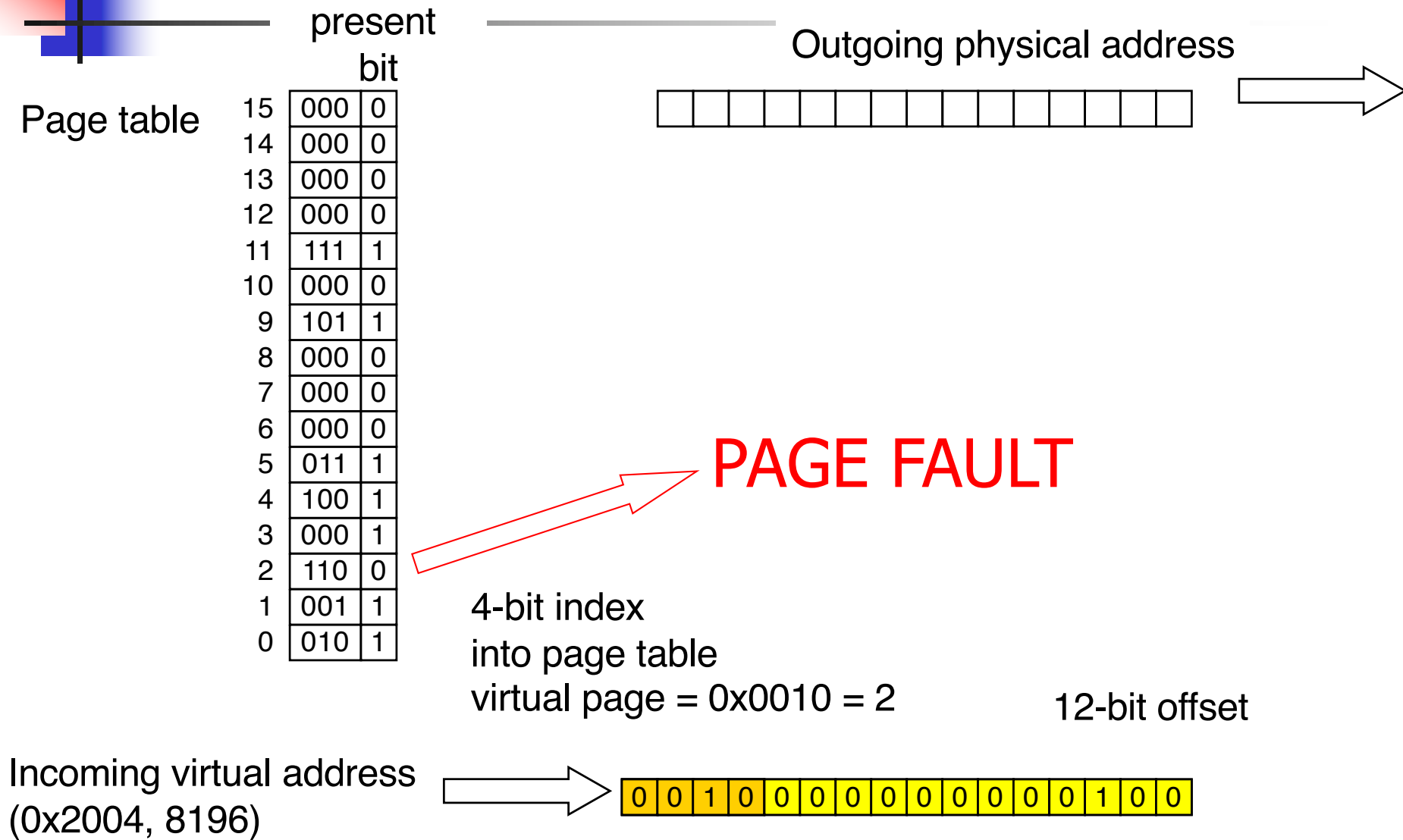
15	000	0
14	000	0
13	000	0
12	000	0
11	111	1
10	000	0
9	101	1
8	000	0
7	000	0
6	000	0
5	011	1
4	100	1
3	000	1
2	110	1
1	001	1
0	010	1

[illegible]

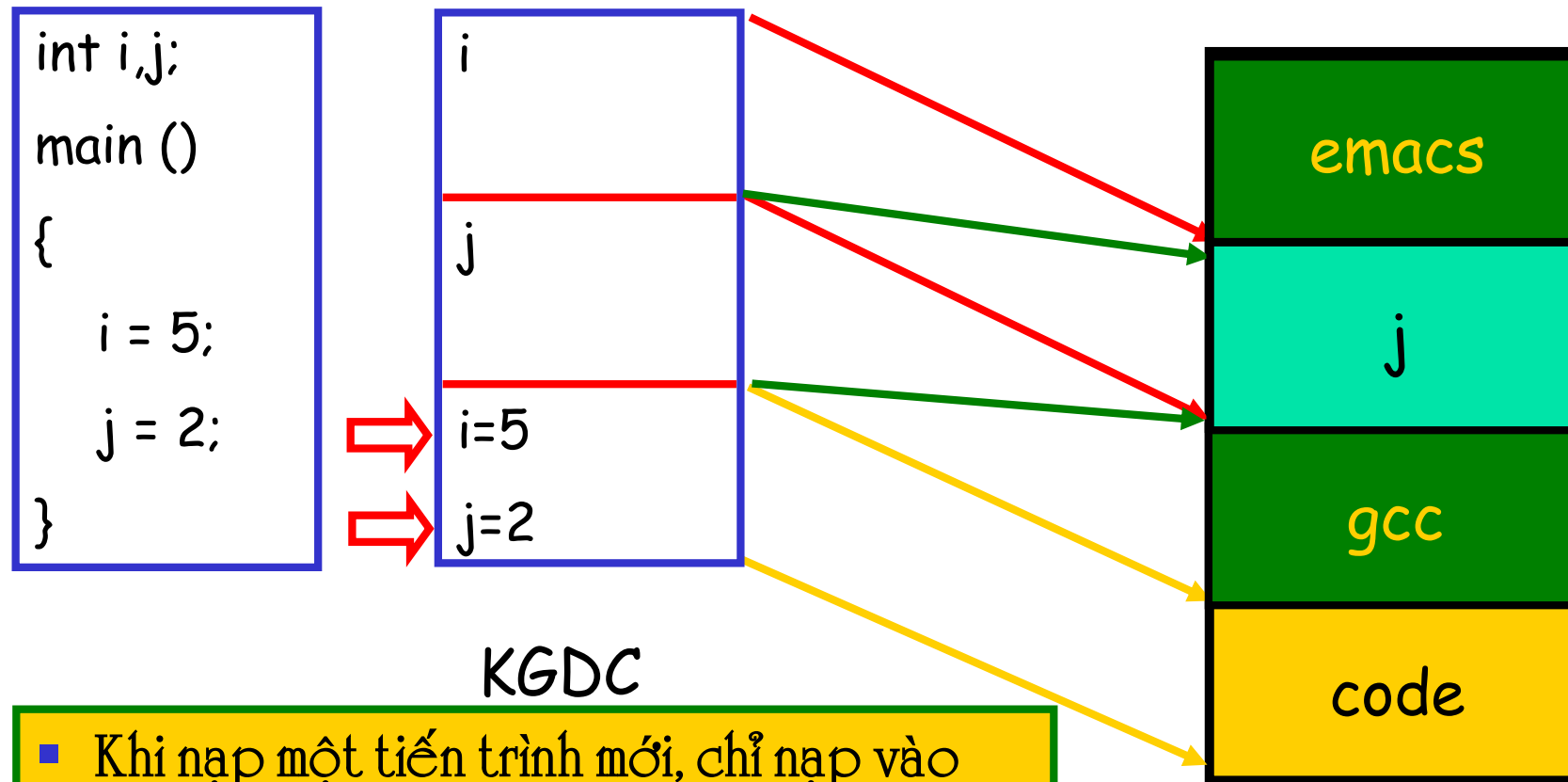
12-bit offset

[illegible]

Memory Lookup



Demand Paging

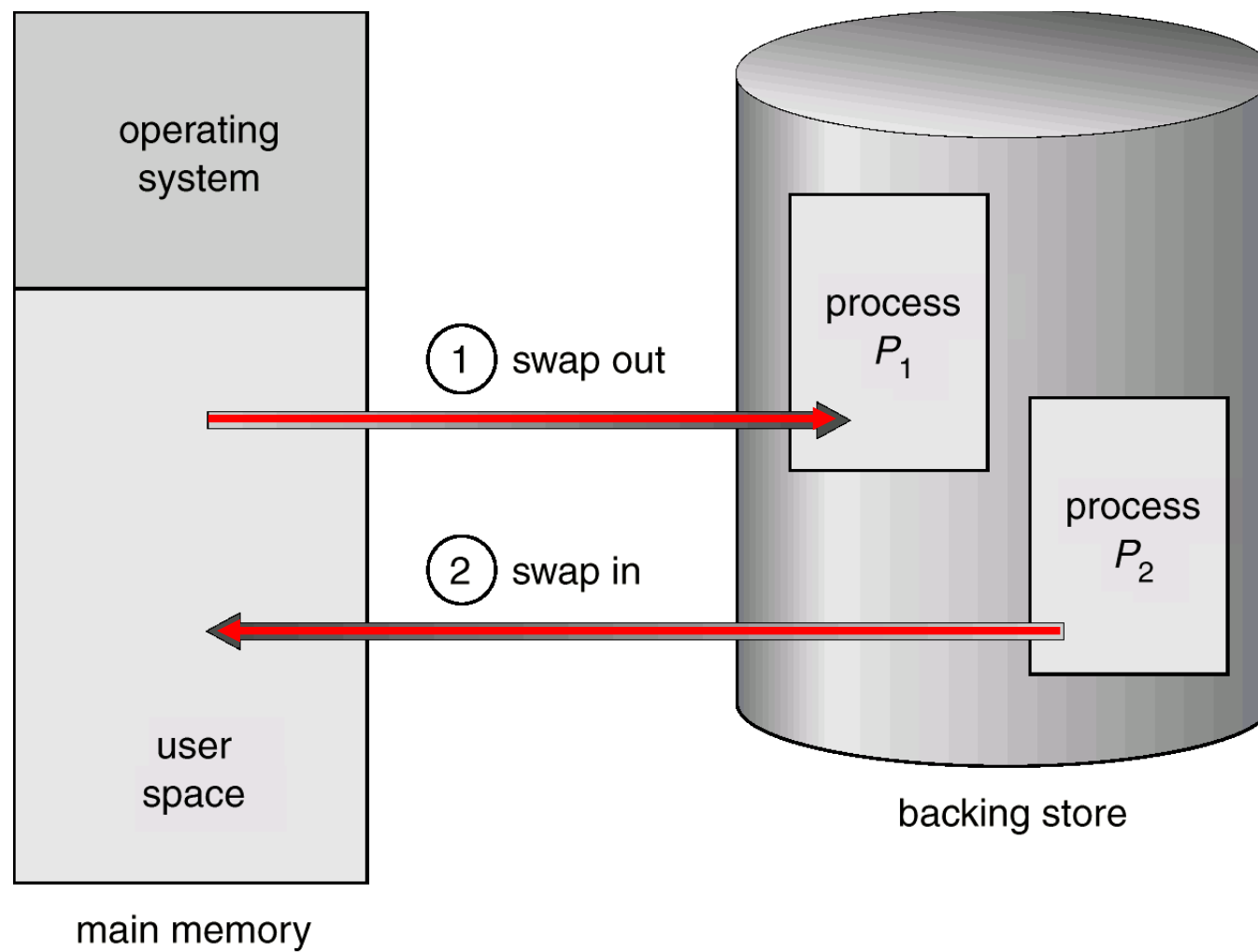


KGDC

KGVl

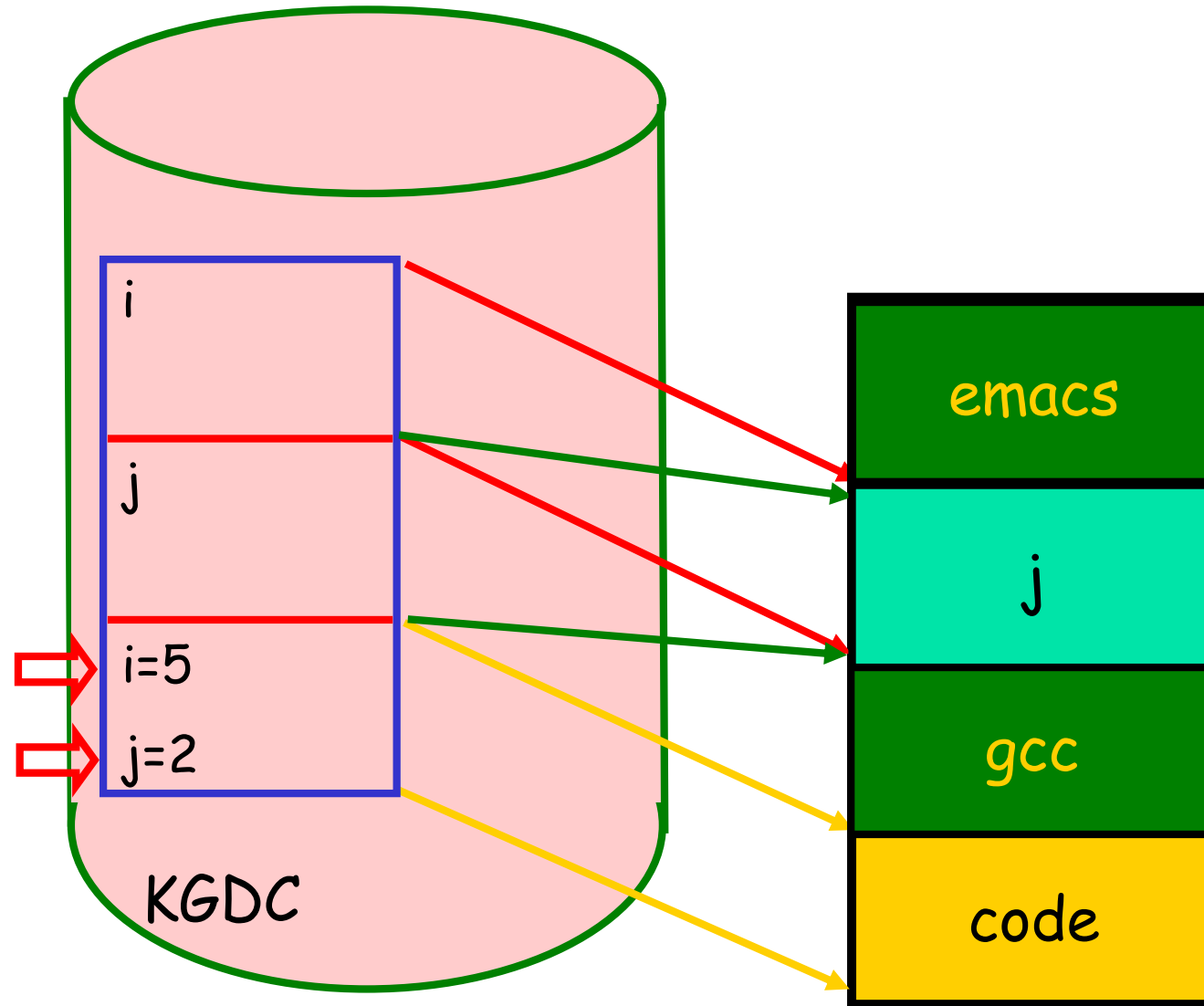
- Khi nạp một tiến trình mới, chỉ nạp vào BNC page chứa entry code
- Khi truy xuất đến một chỉ thị hay dữ liệu, page tương ứng mới được nạp vào BNC

Swapping



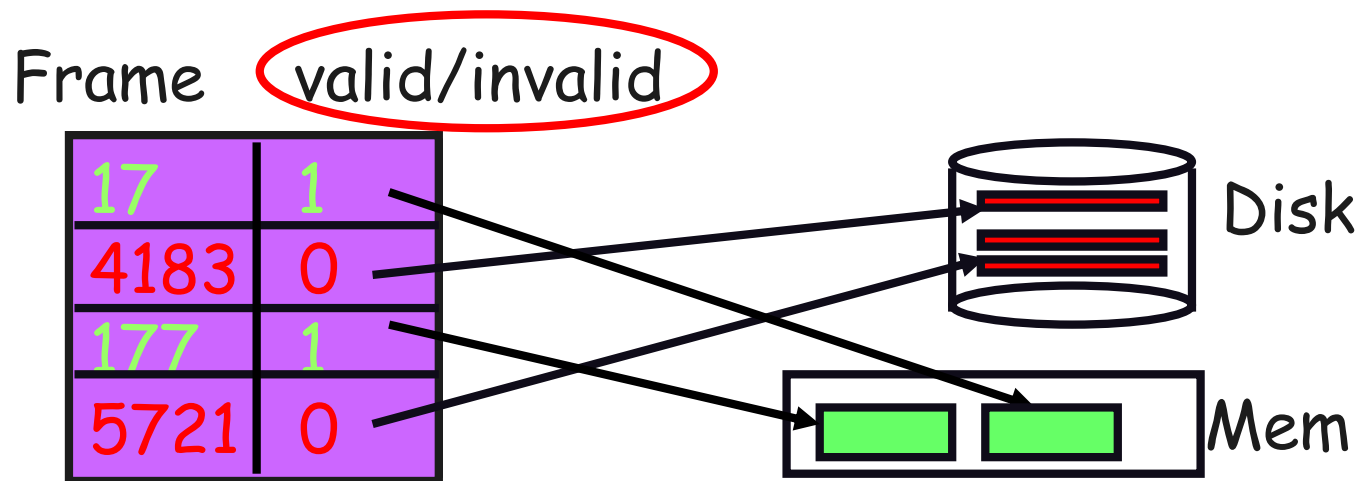
Demand Paging + Swapping

```
int i,j;  
main ()  
{  
    i = 5;  
    j = 2;  
}
```



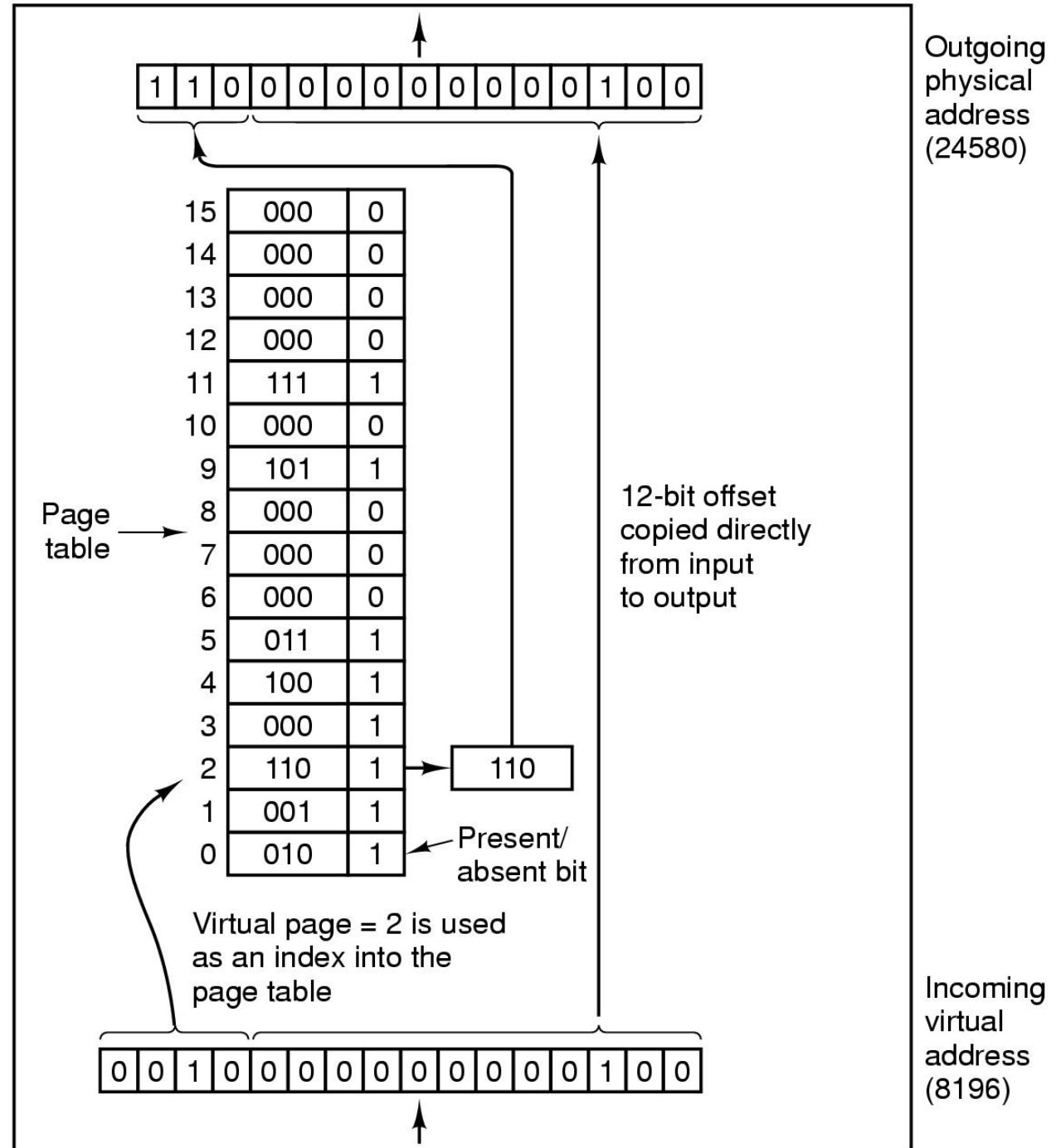
Thực hiện Bộ nhớ ảo

- Bảng trang : thêm 1 **bit valid/invalid** để nhận diện trang đã hay chưa được nạp vào RAM

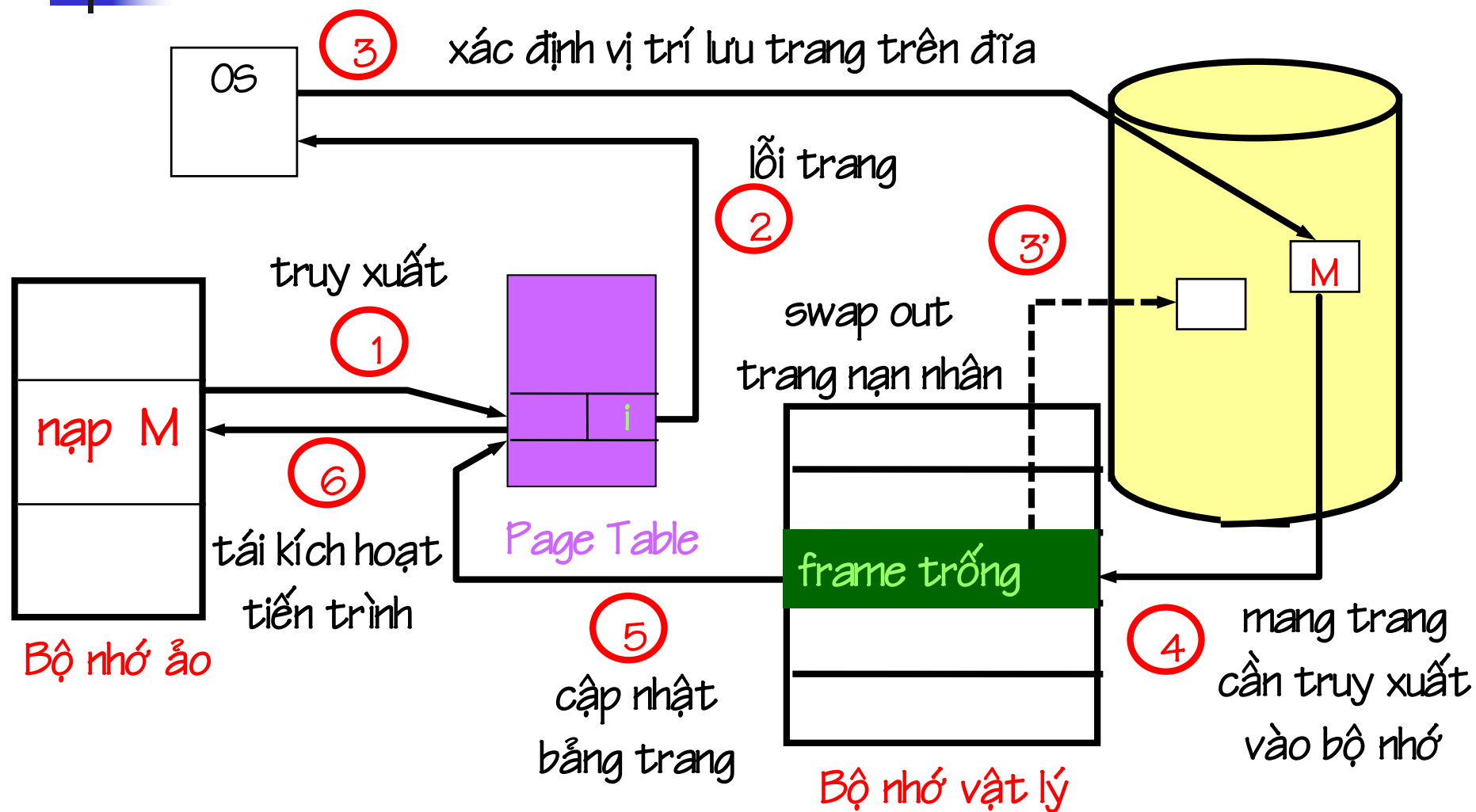


- Truy xuất đến một trang chưa được nạp vào bộ nhớ :
 - **lỗi trang (page fault)**

Page Tables



Xử lý lỗi trang





Các bước xử lý lỗi trang

1. Kiểm tra truy xuất đến bộ nhớ là hợp lệ hay bất hợp lệ
2. Nếu truy xuất bất hợp lệ : kết thúc tiến trình
Ngược lại : đến bước 3
3. Tìm vị trí chứa trang muốn truy xuất trên đĩa.
4. Tìm một khung trang trống trong bộ nhớ chính :
 - a. Nếu tìm thấy : đến bước 5
 - b. Nếu không còn khung trang trống, chọn một khung trang nạn nhân để swap out, cập nhật bảng trang tương ứng rồi đến bước 5
5. Chuyển trang muốn truy xuất từ bộ nhớ phụ vào bộ nhớ chính : nạp trang cần truy xuất vào khung trang trống đã chọn (hay vừa mới làm trống) ; cập nhật nội dung bảng trang, bảng khung trang tương ứng.
6. Tái kích hoạt tiến trình người sử dụng.

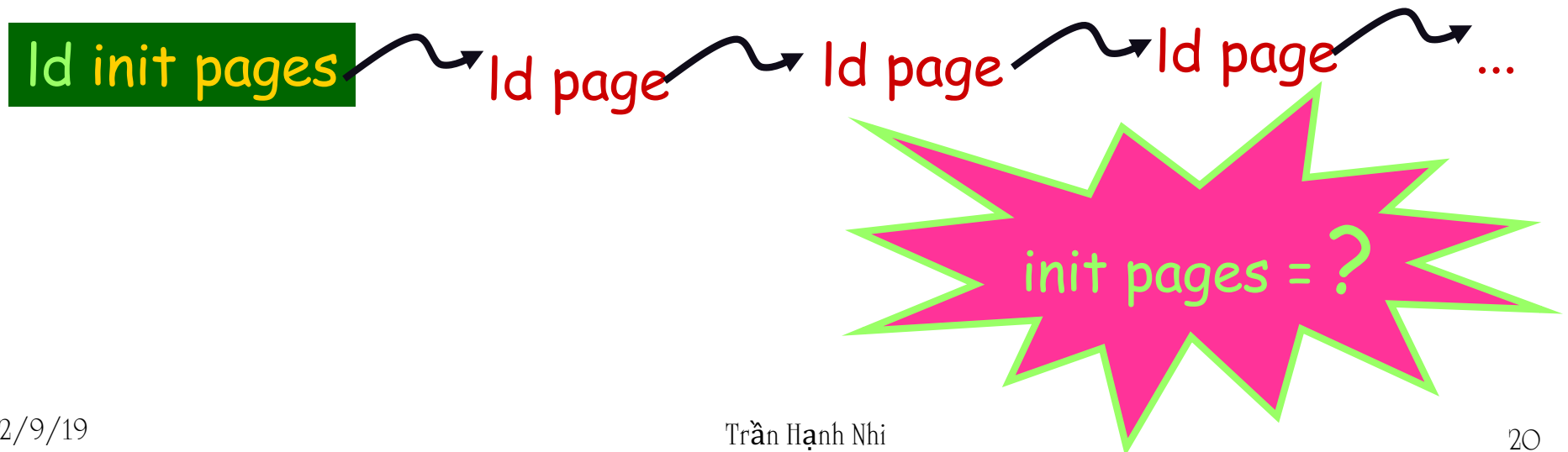


Các câu hỏi

1. Chọn trang nào để nạp ? => Chiến lược nạp
 - Demand Paging / Prepageing
2. Chọn trang nạn nhân ? => Chiến lược thay thế trang
 - FIFO / OPTIMAL/LRU
3. Cấp phát khung trang => Chiến lược cấp phát khung trang
 - Công bằng/ Tỷ lệ...

Chiến lược nạp

- Quyết định thời điểm nạp một/nhiều page vào BNC
 - Nạp trước : làm sao biết ? => **prepaging**
 - Nạp sau : tần suất lỗi trang cao ? => **pure demand paging**
- **Prepaging** :
 - Nạp sẵn một số trang cần thiết vào BNC trước khi truy xuất chúng
- **Demand paging** :
 - Chỉ nạp trang khi được yêu cầu truy xuất đến trang đó





Chiến lược thay thế trang (Page Replacement)

Mục tiêu :

- thay thế trang sao cho tần suất xảy ra lỗi trang thấp nhất
- Đánh giá
 - Sử dụng số frame cụ thể
 - Giả sử có một chuỗi truy xuất cụ thể
 - adresse : 0100, 0432, 0101, 0612, 0102, 0103, 0104, 0611
 - # page : 1, 4, 1, 6, 1, 1, 1, 6,
 - Thực hiện một thuật toán thay thế trang trên chuỗi truy xuất này
 - Đếm số lỗi trang phát sinh
- Chuỗi truy xuất
 - 7, 0, 1, 2, 0, 3, 0, 4, 2, 3, 0, 3, 2, 1, 2, 0, 1, 7, 0, 1
 - 3 frames

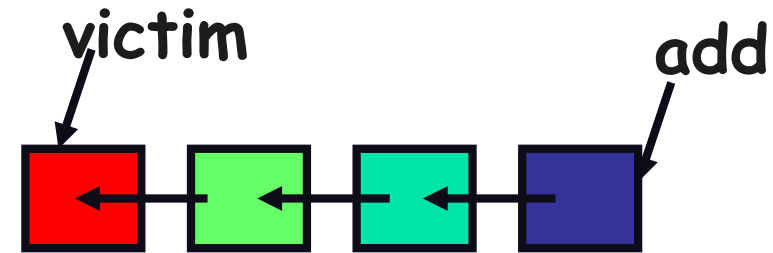


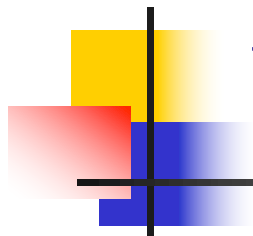
Chiến lược thay thế trang

- FIFO
- Optimal
- LRU (Least Recently Used)

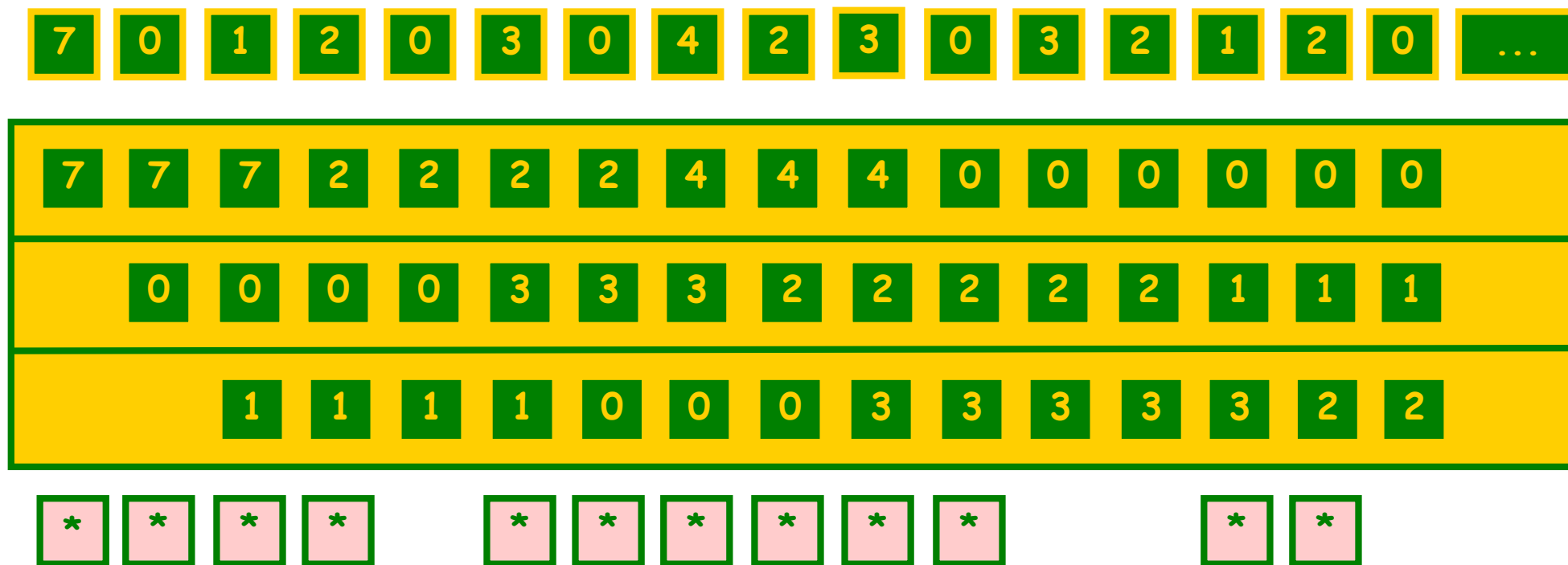
Chiến lược thay thế trang FIFO

- **Nguyên tắc**: Nạn nhân là trang “già” nhất
 - Được nạp vào lâu nhất trong hệ thống
- **Thực hiện**
 - Lưu thời điểm nạp, so sánh để tìm min
 - Chi phí cao
 - Tổ chức FIFO các trang theo thứ tự nạp
 - Trang đầu danh sách là nạn nhân
- **Nhận xét**
 - Đơn giản
 - Công bằng ?
 - Không xét đến tính sử dụng!
 - Trang được nạp vào lâu nhất có thể là trang cần sử dụng thường xuyên !





Ví dụ : FIFO



FIFO và hiệu ứng Belady

- Sử dụng càng nhiều frame...càng có nhiều lỗi trang !

All pages frames initially empty

	0	1	2	3	0	1	4	0	1	2	3	4	
Youngest page		0	1	2	3	0	1	4	4	4	2	3	3
			0	1	2	3	0	1	1	1	4	2	2
Oldest page				0	1	2	3	0	0	0	1	4	4
		P	P	P	P	P	P				P	P	

9 Page faults

(a)

		0	1	2	3	0	1	4	0	1	2	3	4
Youngest page		0	1	2	3	3	3	4	0	1	2	3	4
			0	1	2	2	2	3	4	0	1	2	3
Oldest page				0	1	1	1	2	3	4	0	1	2
					0	0	0	1	2	3	4	0	1
		P	P	P	P			P	P	P	P	P	P

10 Page faults

(b)

Chiến lược thay thế trang : Optimal

- **Nguyên tắc** : Nạn nhân là trang lâu sử dụng đến nhất trong tương lai
 - Làm sao biết ?
- **Nhận xét**
 - Bảo đảm tần suất lỗi trang thấp nhất
 - Không khả thi !

AGBDCAB**C**ABC**G**ABC

victim

Cur page

Ví dụ : Optimal

7	0	1	2	0	3	0	4	2	3	0	3	2	1	2	0	...
7	7	7	2	2	2	2	2	2	2	2	2	2	2	2	0	
	0	0	0	0	0	0	4	4	4	0	0	0	0	0	1	
		1	1	1	3	3	3	3	3	3	3	3	1	1	2	
*	*	*	*		*		*			*			*			

Chiến lược thay thế trang : LRU

- **Nguyên tắc** : Nạn nhân là trang lâu nhất chưa sử dụng đến trong quá khứ
 - Nhìn lui : đủ thông tin
- **Nhận xét**
 - Xấp xỉ Optimal
 - Thực hiện ?





Ví dụ : LRU

7	0	1	2	0	3	0	4	2	3	0	3	2	1	2	0	...
7	7	7	2	2	2	2	4	4	4	0	0	0	1	1	1	
	0	0	0	0	0	0	0	0	3	3	3	3	3	3	0	
		1	1	1	3	3	3	2	2	2	2	2	2	2	2	
*	*	*	*		*		*	*	*	*		*		*		



Thực hiện LRU

■ Sử dụng bộ đếm:

- Thêm trường **reference time** cho mỗi phần tử trong bảng trang
- Thêm vào cấu trúc của CPU một bộ đếm **counter**.
- mỗi lần có sự truy xuất đến một trang trong bộ nhớ
 - giá trị của counter tăng lên 1.
 - giá trị của counter được ghi nhận vào reference time của trang tương ứng.
- thay thế trang có reference time là min .

■ Sử dụng stack:

- tổ chức một stack lưu trữ các số hiệu trang
- mỗi khi thực hiện một truy xuất đến một trang, số hiệu của trang sẽ được xóa khỏi vị trí hiện hành trong stack và đưa lên đầu stack.
- trang ở đỉnh stack là trang được truy xuất gần nhất, và trang ở đáy stack là trang lâu nhất chưa được sử dụng..

Thực hiện LRU với stack

reference string

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

2
1
0
7
4

stack before a

7
2
1
0
4

stack after b

a

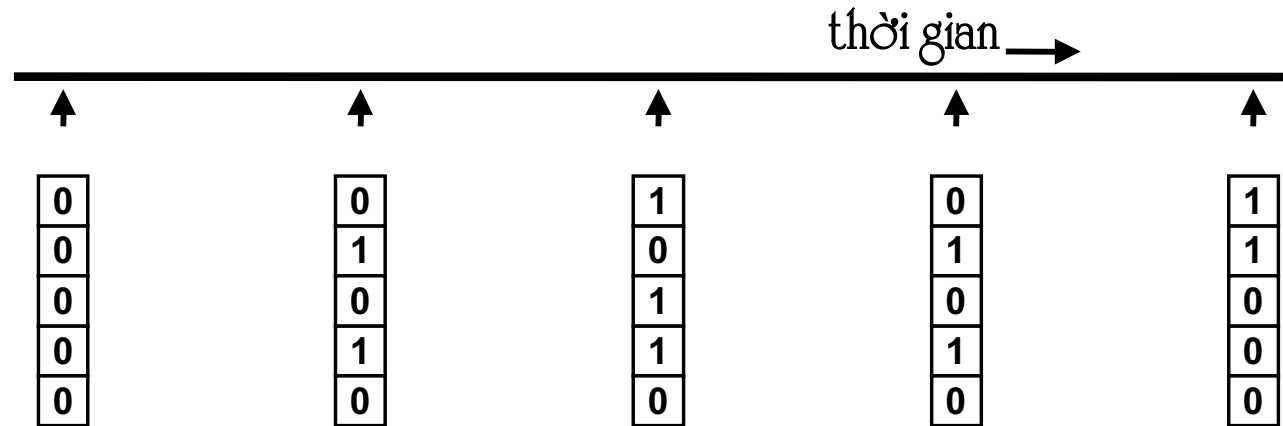
b

Thực hiện LRU : thực tế

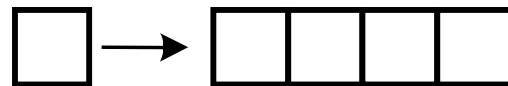
- Hệ thống được hỗ trợ phần cứng hoàn chỉnh để cài đặt LRU ?
 - Đừng có mơ!
- Hệ thống chỉ được trang bị thêm một bit **reference** :
 - gắn với một phần tử trong bảng trang.
 - được khởi gán là 0
 - được phần cứng đặt giá trị 1 mỗi lần trang tương ứng được truy cập
 - được phần cứng gán trở về 0 sau từng chu kỳ qui định trước.
- Bit reference chỉ giúp xác định những trang có truy cập, không xác định **thứ tự truy cập**
 - Không cài đặt được LRU
 - Xấp xỉ LRU...



Xấp xỉ LRU : Sử dụng các bits History



bit reference



các bits history

- sử dụng thêm N bit history phụ trợ
- Sau từng chu kỳ, bit reference sẽ được chép lại vào một bit history trước khi bị reset
 - N bit history sẽ lưu trữ tình hình truy xuất đến trang trong N chu kỳ cuối cùng.

Thời gian →



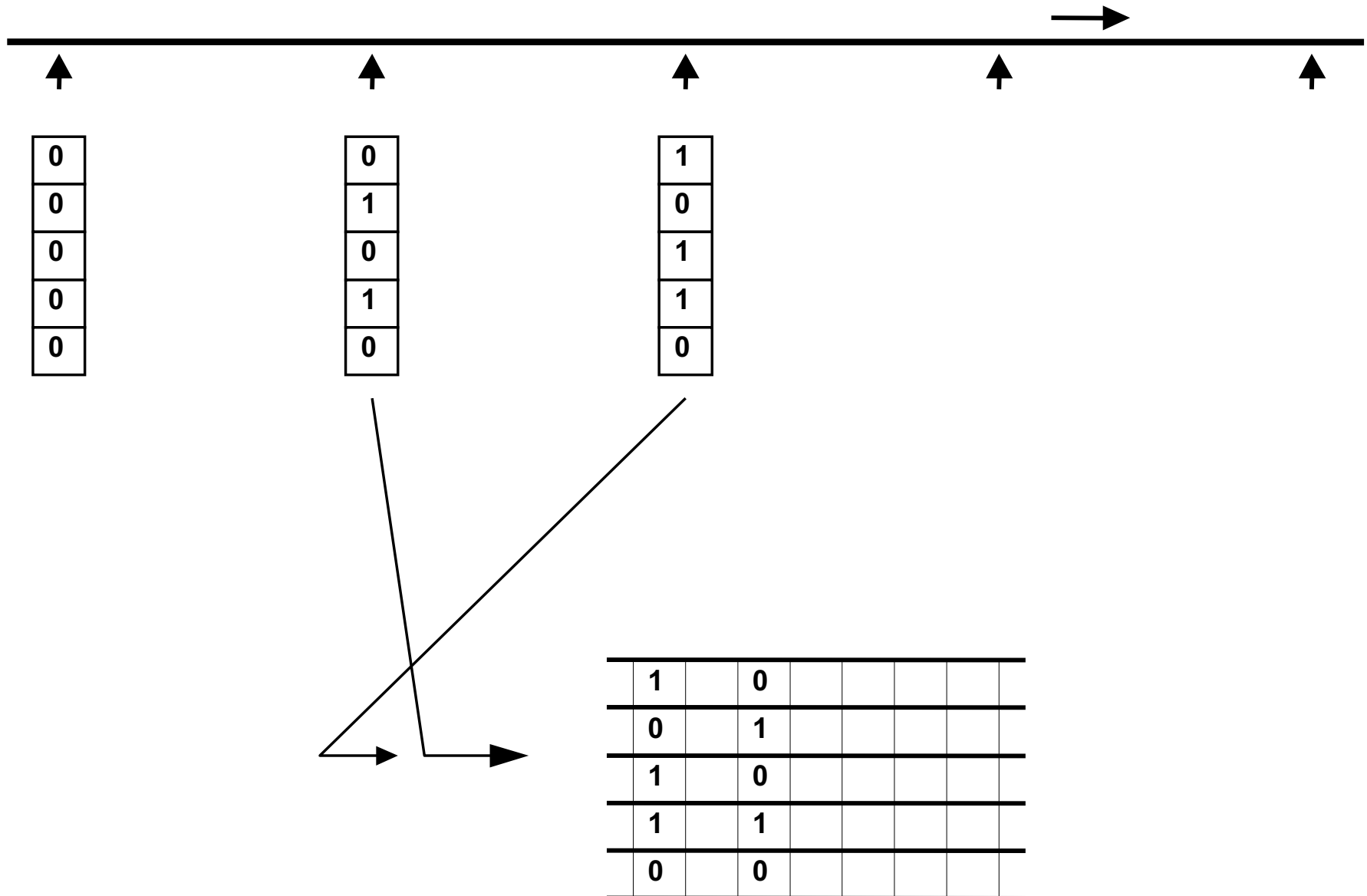
0
0
0
0
0

0
1
0
1
0

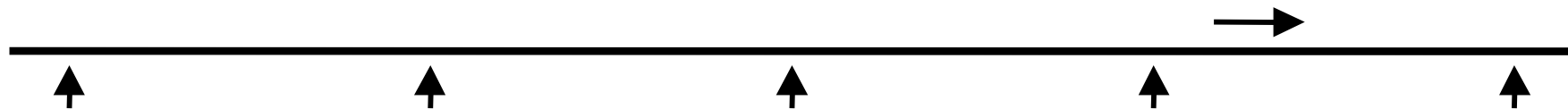


0							
1							
0							
1							
0							

Thời gian



Thời gian

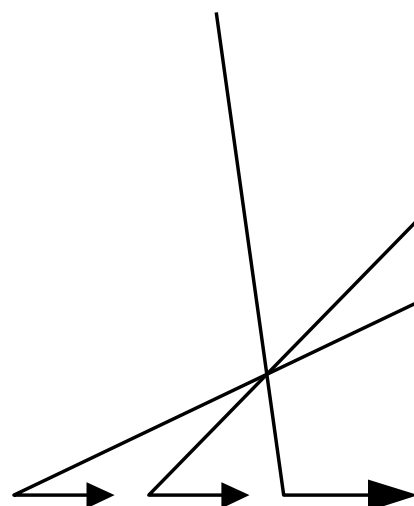


0
0
0
0
0

0
1
0
1
0

1
0
1
1
0

0
1
0
1
0



0		1		0			
1		0		1			
0		1		0			
1		1		1			
0		0		0			

Thời gian



0
0
0
0
0



0
1
0
1
0



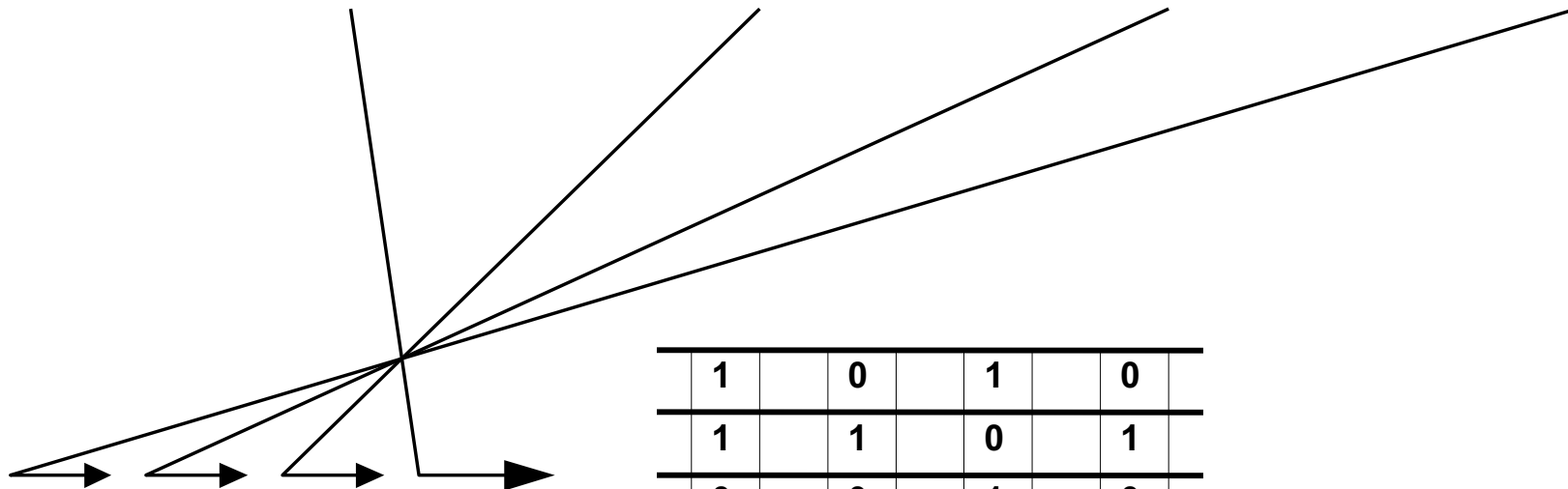
1
0
1
1
0



0
1
0
1
0



1
1
0
0
0



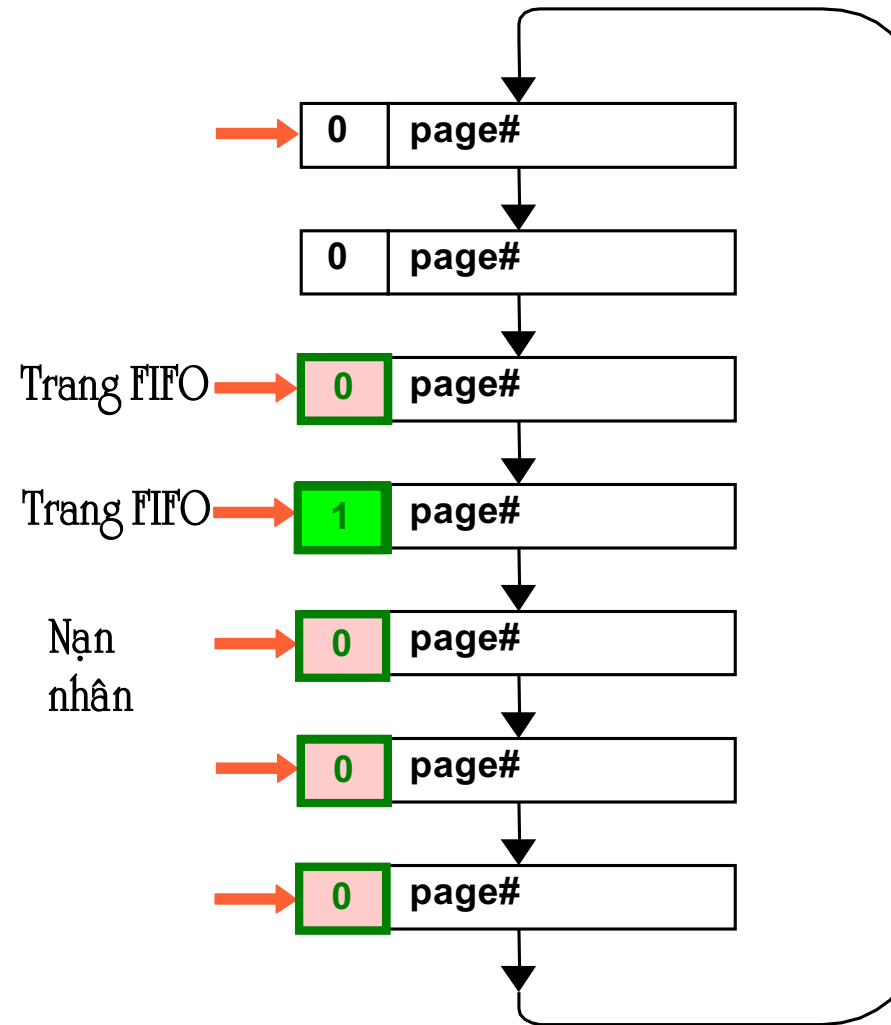
1		0		1		0
1		1		0		1
0		0		1		0
0		1		1		1
0		0		0		0



Xấp xỉ LRU : Cơ hội thứ 2 (Clock algorithm)

- Sử dụng một bit reference duy nhất.
- Chọn được trang nạn nhân theo FIFO
- Kiểm tra bit reference của trang đó :
 - Nếu reference = 0, đúng là nạn nhân rồi ☺
 - Nếu reference = 1, cho trang này một cơ hội thứ hai
 - reference = 0
 - thời điểm vào Ready List được cập nhật lại là thời điểm hiện tại.
- Chọn trang FIFO tiếp theo...
- Nhận xét :
 - Một trang đã được cho cơ hội thứ hai sẽ không bị thay thế trước khi hệ thống đã thay thế hết những trang khác.
 - Nếu trang thường xuyên được sử dụng, bit reference của nó sẽ duy trì được giá trị 1, và trang hầu như không bao giờ bị thay thế.

Xấp xỉ LRU : Cơ hội thứ 2 (Clock algorithm)



Xấp xỉ LRU : NRU

- Sử dụng 2 bit **Reference** và **Modify**
- Với hai bit này, có thể có 4 tổ hợp tạo thành 4 lớp sau :
 - **(0,0)** không truy xuất, không sửa đổi
 - **(0,1)** không truy xuất gần đây, nhưng đã bị sửa đổi
 - **(1,0)** được truy xuất gần đây, nhưng không bị sửa đổi
 - **(1,1)** được truy xuất gần đây, và bị sửa đổi
- Chọn trang nạn nhân là trang có độ ưu tiên cao nhất khi kết hợp bit **R** và bit **M**

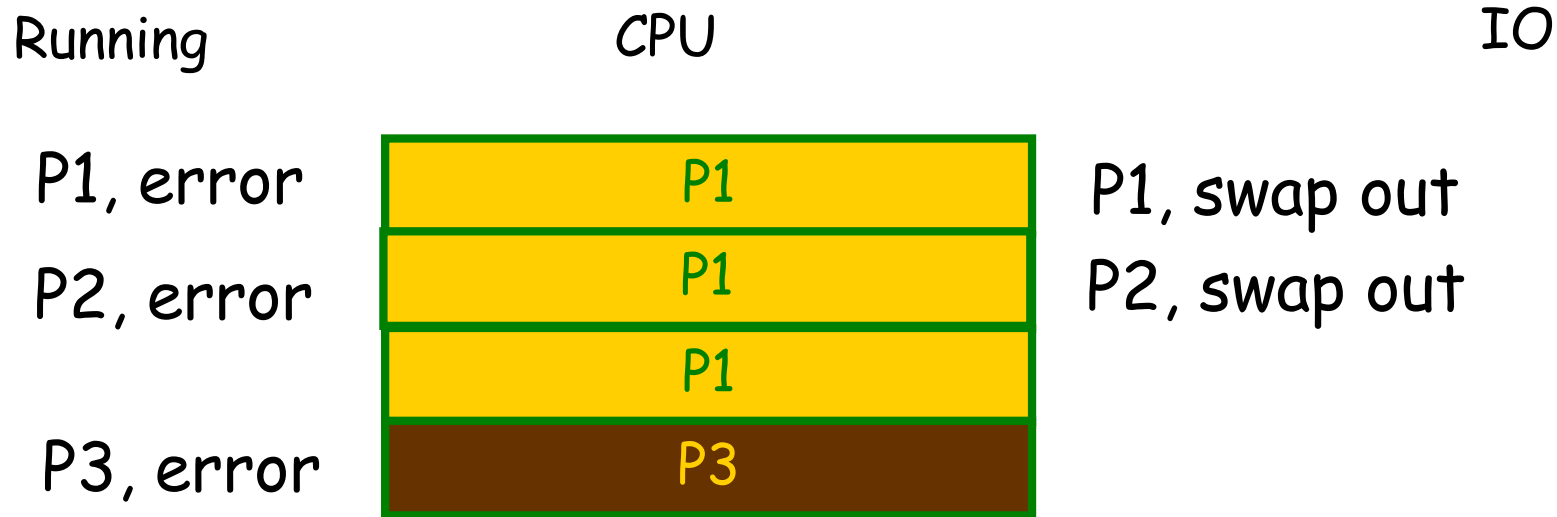
Priority	R	M
4	0	0
3	0	1
2	1	0
1	1	1



Chiến lược cấp phát frame

- Số frame cần cấp phát cho mỗi tiến trình ?
 - Giải sử có m frame và n process
 - Cấp phát công bằng: $\#frame(P_i) = m/n$
 - Công bằng ???
 - Cấp phát theo tỷ lệ: $\#frame(p_i) = (s_i / (\sum s_i)) * m$
 - s_i = kích thước của bộ nhớ ảo cho tiến trình p_i
 - Lỗi trang xảy ra tiếp theo, cấp phát thêm frame cho tiến trình như thế nào ?
 - → Tùy thuộc chiến lược thay thế trang
 - Cục bộ: chỉ chọn trang nạn nhân trong tập các trang của tiến trình phát sinh lỗi trang -> số frame không tăng
 - Toàn cục: được chọn bất kỳ trang nạn nhân nào (dù của tiến trình khác) -> số frame có thể tăng, lỗi trang lan truyền

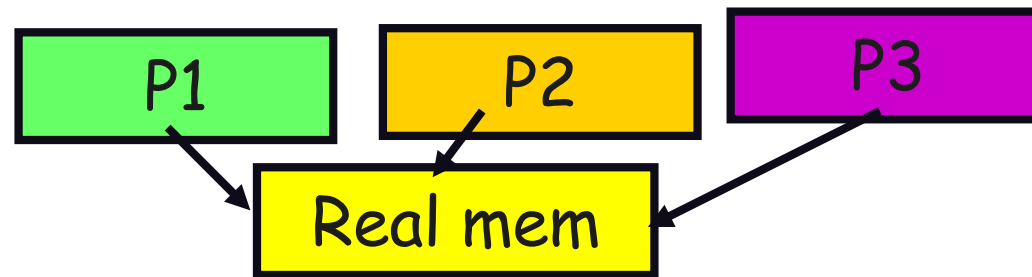
Thay thế trang toàn cục và...kết cục bi thảm !



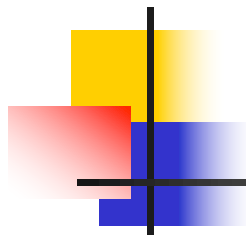
- Tất cả các tiến trình bận rộn thay thế trang !

Thrashing

- Tất cả tiến trình đều bận rộn xử lý lỗi trang !
- IO hoạt động 100 %, CPU rảnh !
- Hệ thống ngừng trệ



- Virtual Memory = Thà hồ xài bộ nhớ
Thrashing = ảo tưởng sụp đổ !
- Các tiến trình trong hệ thống yêu cầu bộ nhớ nhiều hơn khả năng cung cấp của hệ thống !



Working set (1968, Denning)

- Working set:

- **Working set** = tập hợp các trang tiến trình đang truy xuất tại 1 thời điểm
 - Các pages được truy xuất trong Δ lần cuối cùng sẽ nằm trong working set của tiến trình
 - Δ : working set parameter
 - Kích thước của WS thay đổi theo thời gian tùy vào locality của tiến trình

Working-Set Model

- $\Delta \equiv \text{working-set window} \equiv \text{số lần truy cập}$
VD: 10,000 instruction
- 2 6 1 5 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
 - $\Delta=10$
 - $WS(t1) = \{1,2,5,6,7\}$, $WS(t2) = \{3,4\}$
- WS_i (working set of Process P_i) =
tổng số trang được truy cập trong Δ lần gần đây nhất
- $D = \sum WS_i \equiv$ Tổng các frame cần cho N tiến trình trong hệ thống
- if $D > m \Rightarrow \text{Thrashing}$
 - if $D > m$, chọn mộ/một số tiến trình để đình chỉ tạm thời.



Giải quyết thrashing với mô hình Working set

- Sử dụng Working set
 - Cache partitioning: Cấp cho mỗi tiến trình số frame đủ chứa WS của nó
 - Page replacement: ưu tiên swap out các non-WS pages.
 - Scheduling: chỉ thi hành tiến trình khi đủ chỗ để nạp WS của nó