Chương 7
BỘ NHỚ ẢO
(Virtual Memory)



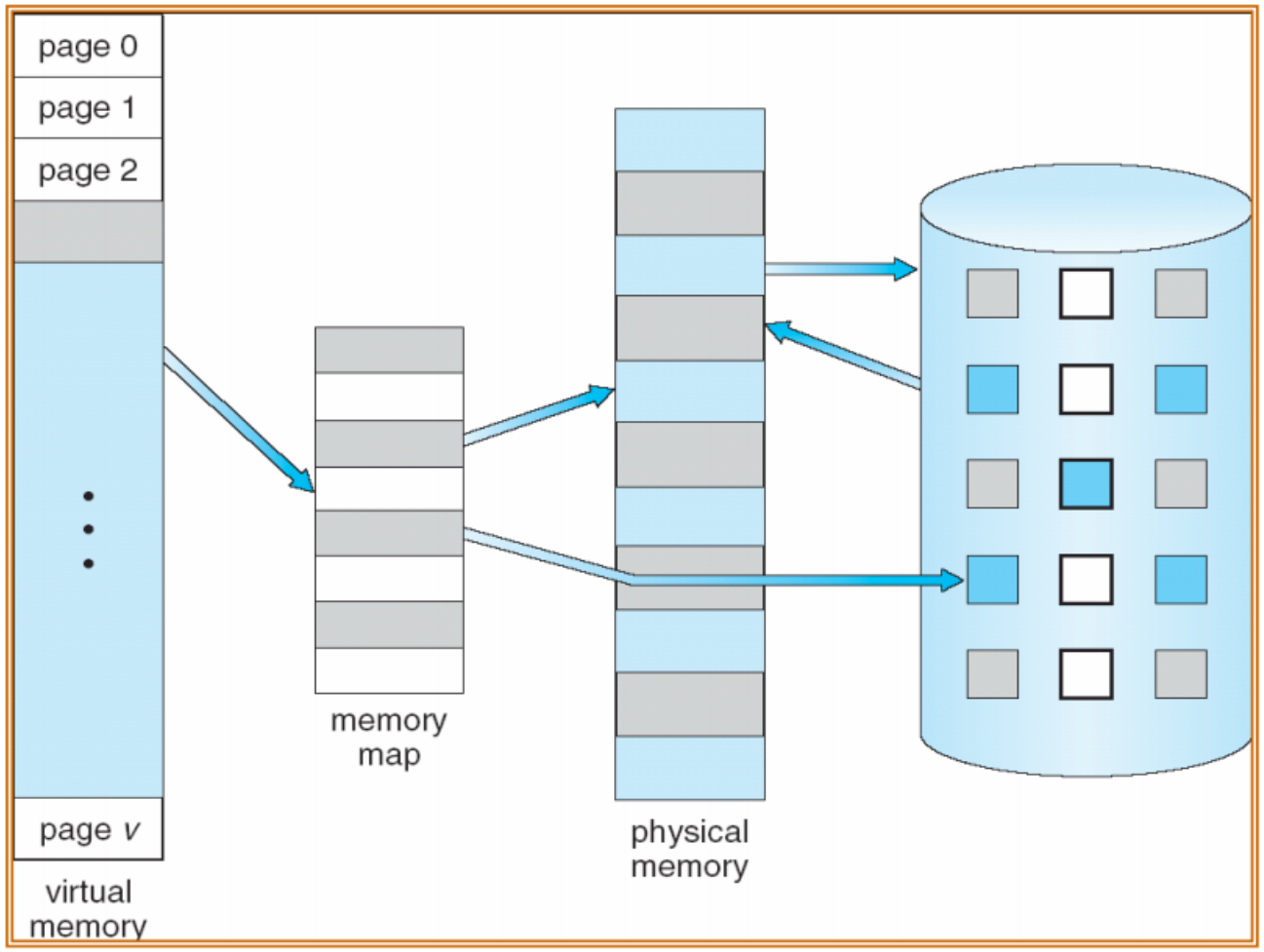
Nội dung

- Cơ sở
- Phân trang theo yêu cầu
- Hiệu năng của phân trang theo yêu cầu
- Thay thế trang
- Các thuật toán thay thế trang
- Cấp phát khung
- Thrashing
- Các vấn đề khác

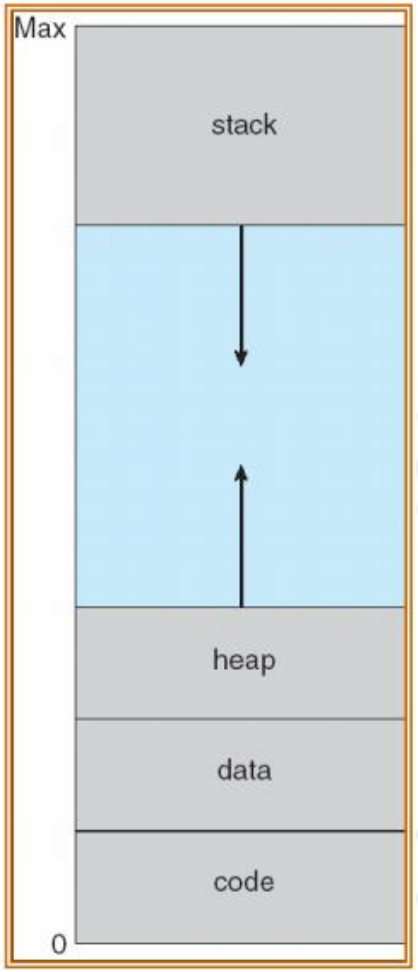
Cơ sở

- Câu lệnh/ dữ liệu cần ở trong bộ nhớ trong để thực hiện
- Không cần thường xuyên lưu toàn bộ chương trình người dùng vào trong bộ nhớ trong
- Bộ nhớ ảo – tách biệt bộ nhớ luận lý mức người dùng và bộ nhớ vật lý
  - Chỉ một phần chương trình cần trong bộ nhớ để thực thi
  - Không gian địa chỉ luận lý có thể lớn hơn nhiều không gian địa chỉ vật lý
  - Cho phép chia sẻ các không gian địa chỉ bởi một số tiến trình.
  - Cho phép tạo nhiều tiến trình một cách hiệu quả.
- Bộ nhớ ảo có thể được thực thi thông qua:
  - Phân trang theo yêu cầu
  - Phân đoạn theo yêu cầu

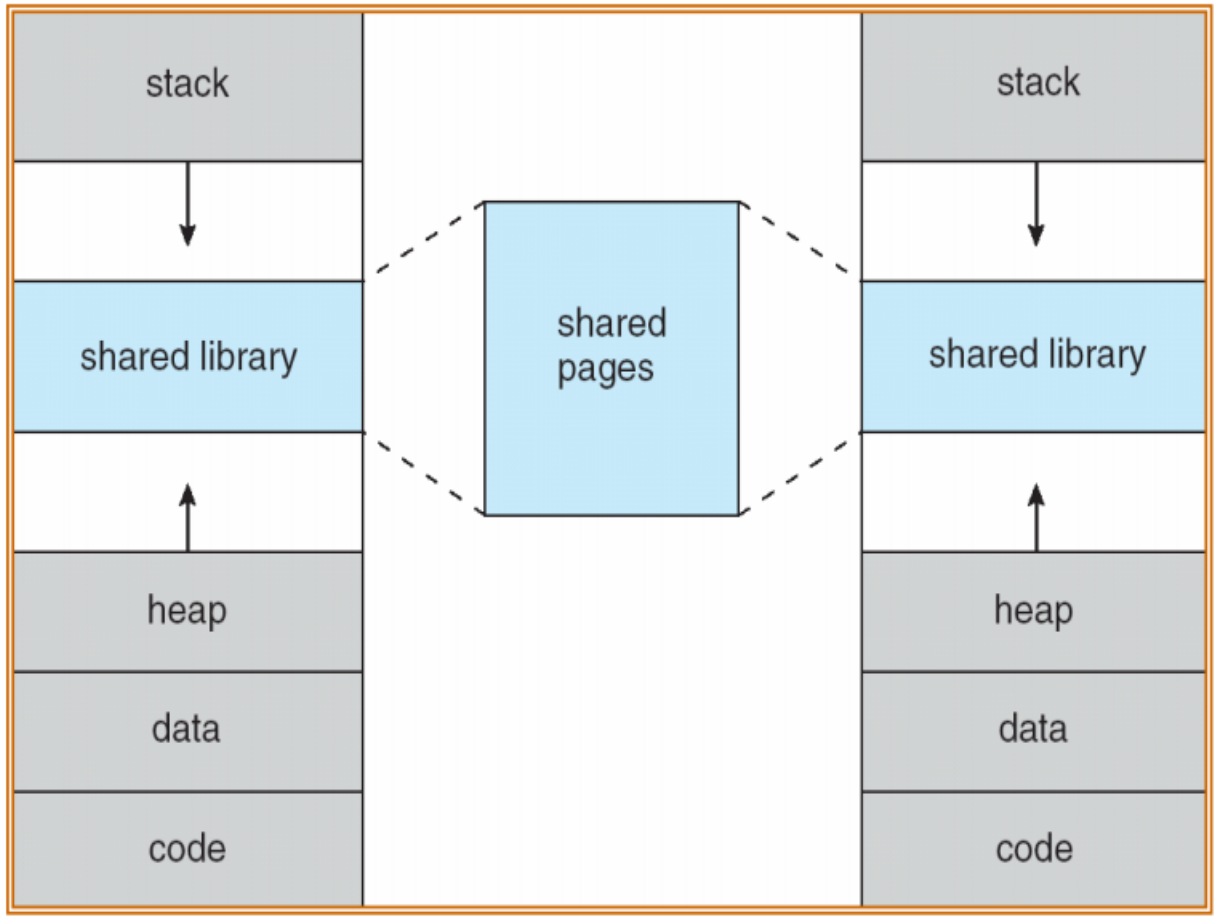
Bộ nhớ ảo lớn hơn bộ nhớ vật lý



Không gian địa chỉ ảo



Thư viện chia sẻ dùng bộ nhớ ảo



Phân trang theo yêu cầu

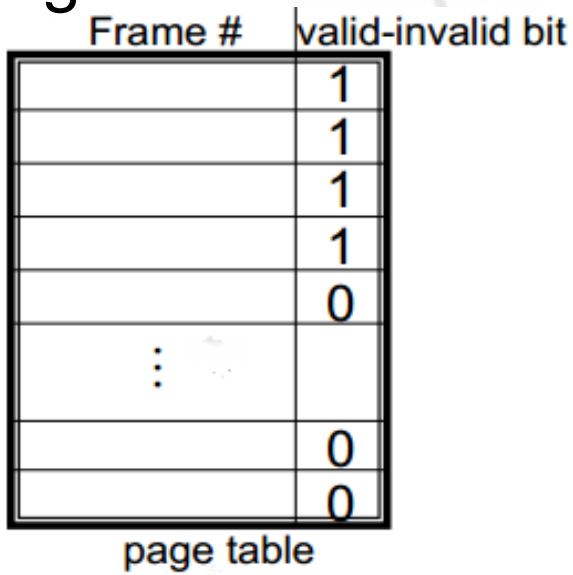
- Chỉ tải một trang vào bộ nhớ khi cần thiết
  - Cần vào/ ra ít
  - Cần bộ nhớ ít
  - Phản ứng nhanh hơn
  - Cho phép nhiều người dùng hơn
- Cần một trang ⇒ tham chiếu đến nó
  - Tham chiếu không hợp lệ ⇒ bỏ qua
  - Tham chiếu hợp lệ nhưng trang không trong bộ nhớ ⇒ tải vào bộ nhớ

Bit valid/ invalid

- Liên kết mỗi phần tử của bảng trang với một bit valid/ invalid

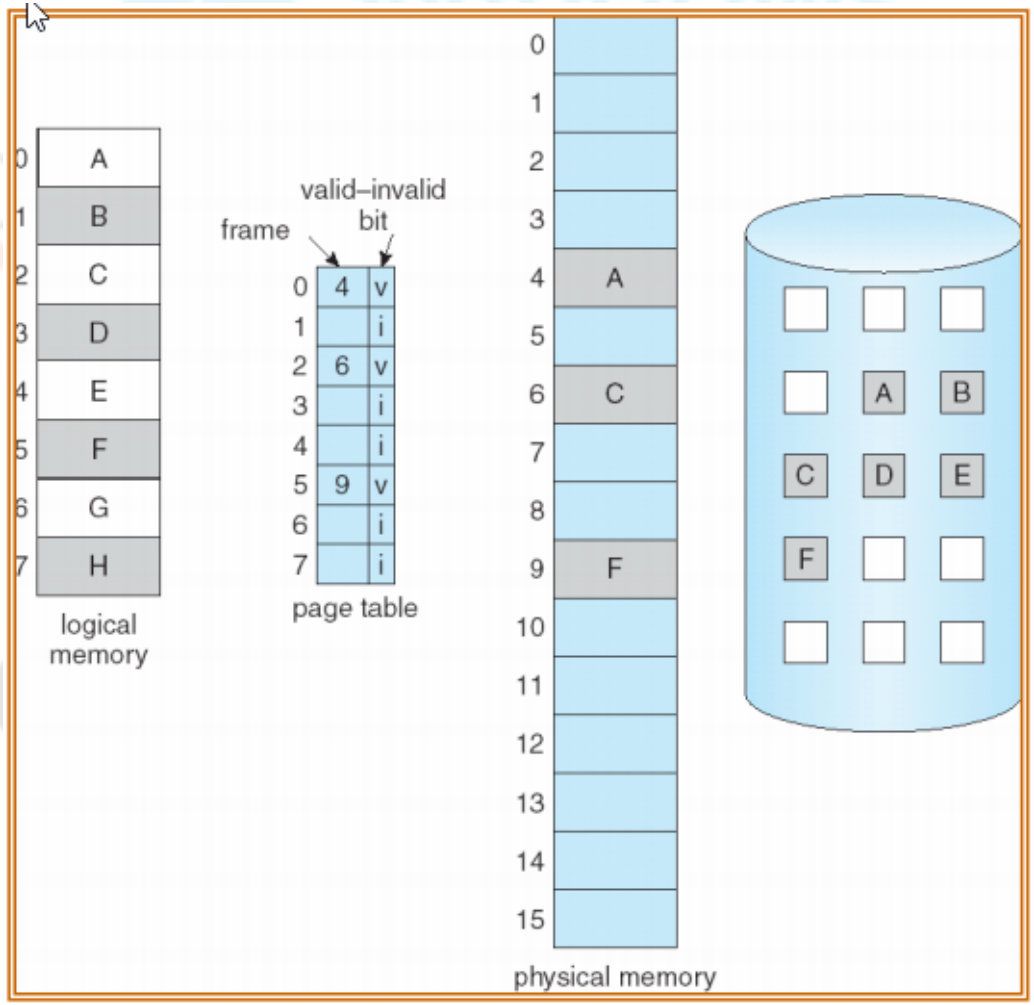
(1 ⇒ in-memory, 0 ⇒ not-in-memory)

- Bit valid - invalid được khởi tạo bằng 0 với mọi phần tử của bảng trang.
- Ví dụ về một bảng trang.



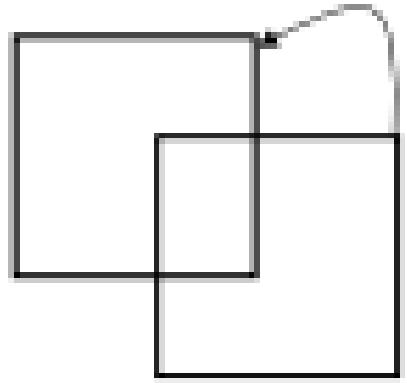
Trong quá trình dịch địa chỉ, nếu bit valid-invalid trong phần tử bảng trang là 0 ⇒ lỗi trang

Bảng trang khi một vài trang không trong bộ nhớ chính

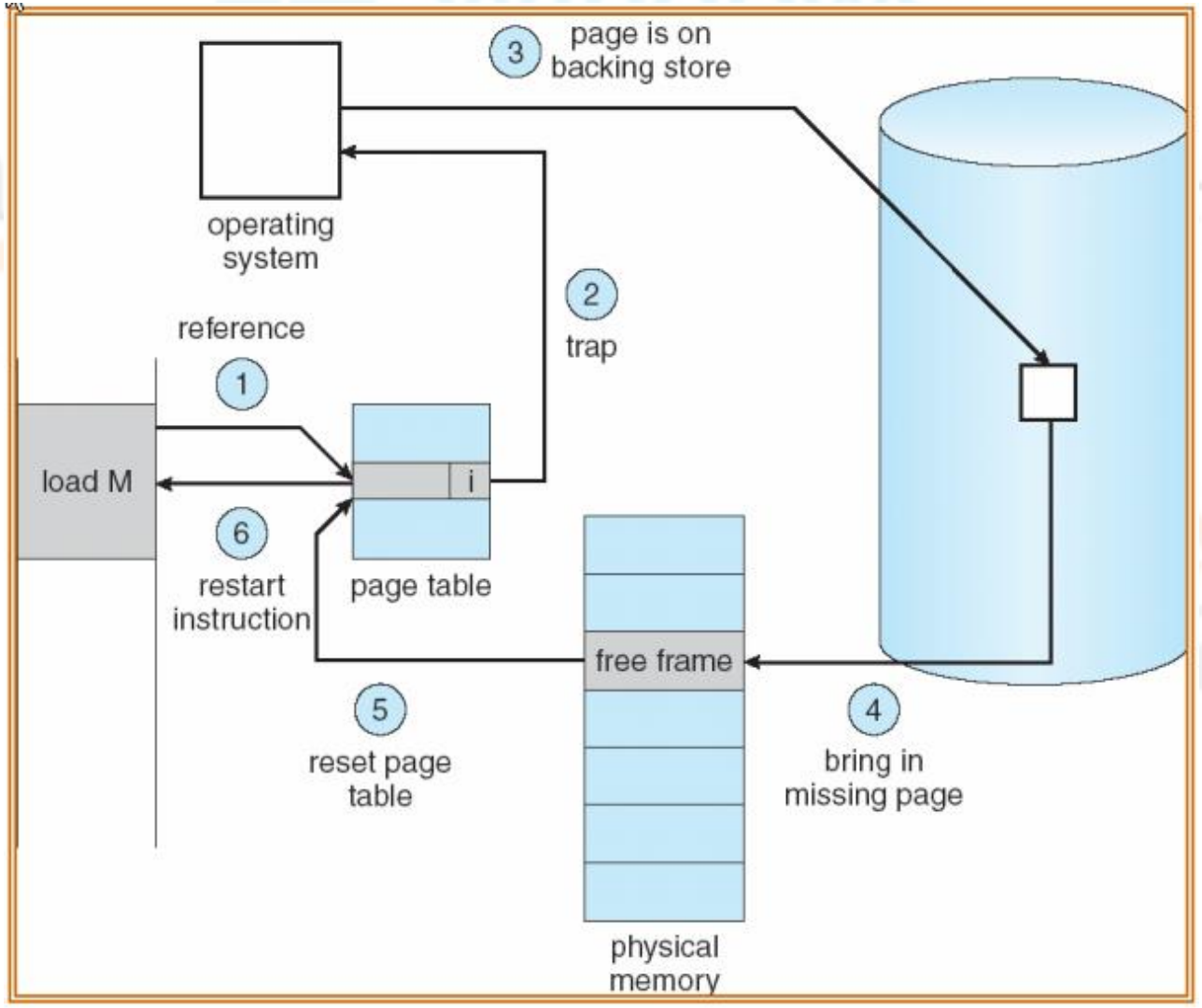


Lỗi trang

- Tham chiếu đến một trang không có trong bộ nhớ, trước tiên tham chiếu làm sập một bẫy của hệ điều hành ⇒ lỗi trang
- Hệ điều hành kiểm tra bảng khác để xác định:
  - Tham chiếu không hợp lệ ⇒ bỏ qua.
  - Tham chiếu hợp lệ, nhưng trang không có trong bộ nhớ:
    - Lấy ra một frame rỗng
    - Tráo đổi trang vào frame
    - Thiết lập lại các bảng, bit valid = 1
    - Khởi tạo lại lệnh:
  - Chuyển khối



Các bước xử lý lỗi trang



Trường hợp không còn frame rỗi

- Thay thế trang
  - Tìm một trang nào đó hiện trong bộ nhớ, nhưng đang không được sử dụng, swap nó ra.
  - Thuật toán thay trang
  - Hiệu năng – cần một thuật toán trả lại với ít lỗi trang nhất có thể.
- Trang có thể được tải vào bộ nhớ một vài lần.

Hiệu năng của phân trang theo yêu cầu

- Tỉ lệ lỗi trang 0 ≤p≤1.0
  - Nếu p= 0, không có lỗi trang
  - Nếu p= 1, mọi tham chiếu đều lỗi
- Thời gian truy cập hiệu quả(EAT)

EAT = (1 –p) x thời gian truy cập bộ nhớ

+ p (phụ trội do lỗi trang

+ [swap trang ra]

+ swap trang vào

+ phụ trội do khởi động lại)

Ví dụ về phân trang theo yêu cầu

- Thời gian truy cập bộ nhớ = 1 micro second
- 50% trang bị thay thế cần phải cập nhật lại (do đã có sửa đổi)
  $\rightarrow$ 50% trang cần phải được swap ra
- Thời gian swap trang = 10 msec = 10,000 microsec

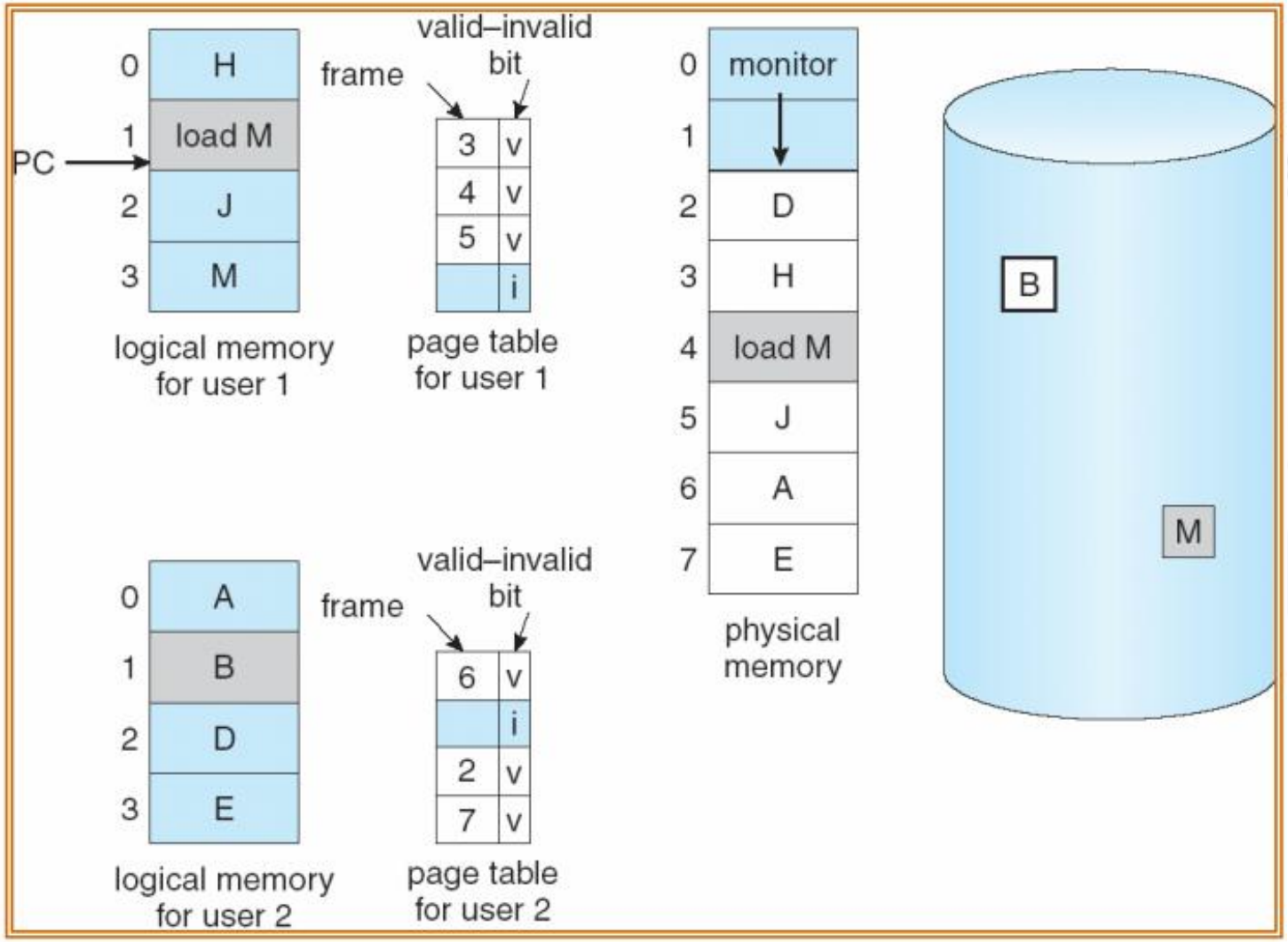
$$EAT = (1 - p) \times 1 + p\ (15000)$$

$$= 1 + 15000p \text{ (in microsec)}$$

Thay thế trang

- Tránh tình trạng phân phối bộ nhớ quá tải
  - Dịch vụ lỗi trang bao gồm việc thay trang.
- Sử dụng bit sửa đổi (dirty)
  - Giảm thời gian phụ trội của việc chuyển trang – chỉ có các trang bị sửa đổi mới phải ghi lại lên đĩa.
- Thay trang làm tăng sự tách biệt giữa bộ nhớ luận lý và bộ nhớ vật lý

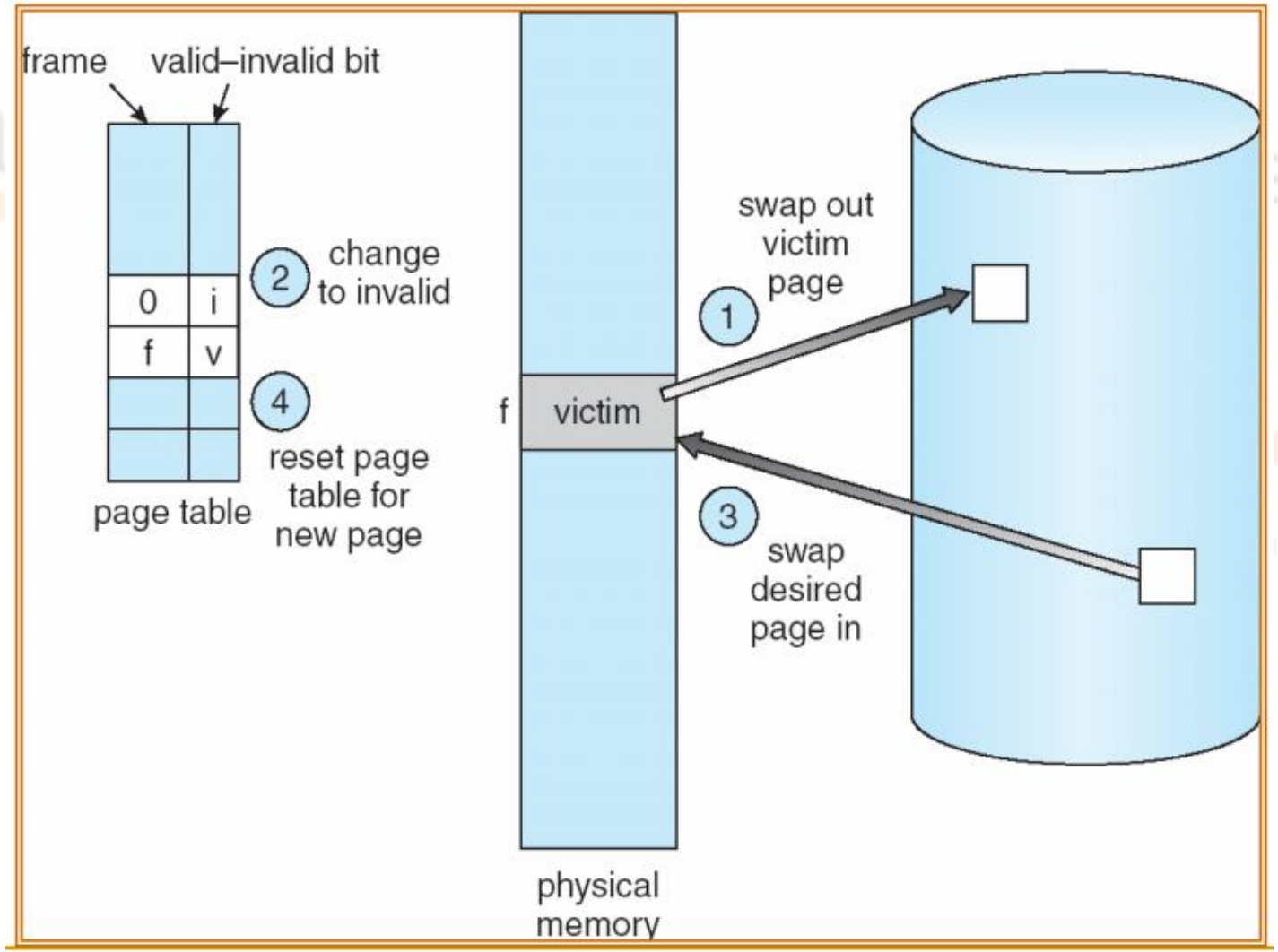
Yêu cầu thay trang



Kĩ thuật thay trang cơ bản

- Tìm vị trí của trang yêu cầu trên đĩa.
- Tìm một frame rỗi:
  - Nếu có một frame rỗi, tận dụng frame đó.
  - Nếu không có frame rỗi, áp dụng thuật toán thay trang để lựa chọn một frame nạn nhân.
- Đọc trang yêu cầu vào frame mới rỗi. Cập nhật trang và bảng frame.
- Khởi động lại tiến trình.

Thay trang

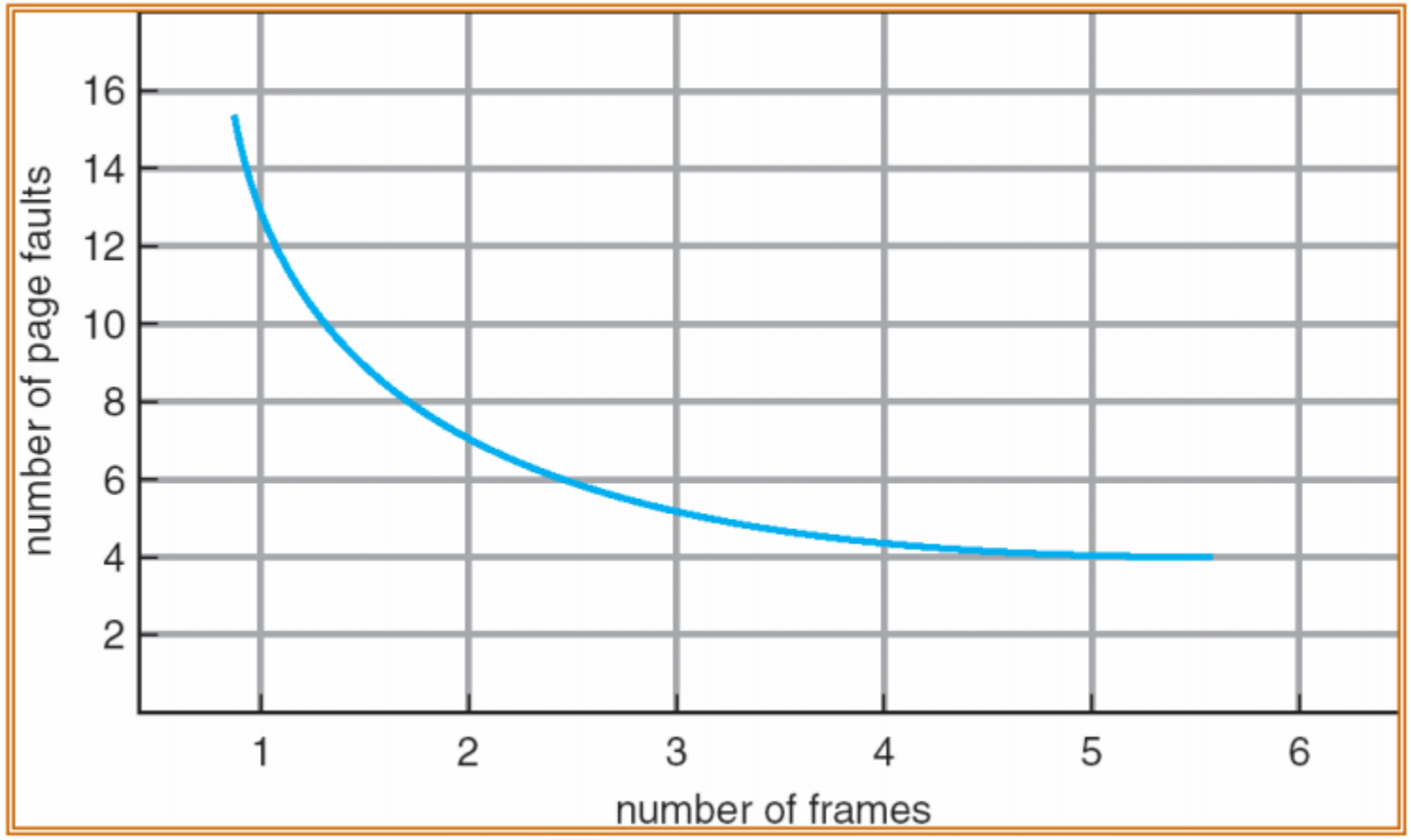

frame
valid–invalid bit
0 i
f v
page table
2
change to invalid
4
reset page table for new page
f
victim
physical memory
1
swap out victim page
3
swap desired page in

Các thuật toán thay thế trang

- Mục đích: tỉ lệ lỗi trang thấp nhất.
- Đánh giá thuật toán:
  - Áp dụng thuật toán trên một chuỗi các tham chiếu bộ nhớ
  - Tính toán số lỗi trang trên chuỗi đó.
  - Trong tất cả các ví dụ sau, ta sử dụng chuỗi tham chiếu sau

    1, 2, 3, 4, 1, 2, 5, 1, 2, 3, 4, 5.

Đồ thị mô tả số lỗi trang theo số Frames



Thuật toán vào trước ra trước (FIFO)

- Chuỗi tham chiếu: 1, 2, 3, 4, 1, 2, 5, 1, 2, 3, 4, 5
- 3 frame (mỗi tiến trình chỉ có 3 trang cùng ở trong bộ nhớ tại cùng một thời điểm)
-

	1	2	3	4	1	2	5	1	2	3	4	5
Frame1	1	1	1	4	4	4	5	5	5	5	5	5
Frame2		2	2	2	1	1	1	1	1	3	3	3
Frame3			3	3	3	2	2	2	2	2	4	4
Lỗi	x	x	x	x	x	x	x			x	x	

9 lỗi

	1	2	3	4	1	2	5	1	2	3	4	5
Frame1	1	1	1	1	1	1	5	5	5	5	4	4
Frame2		2	2	2	2	2	2	1	1	1	1	5
Frame3			3	3	3	3	3	3	2	2	2	2
Frame4				4	4	4	4	4	4	3	3	3
Lỗi	x	x	x	x			x	x	x	x	x	x

10 lỗi

- Thay thế FIFO – Belady's Anomaly:
  - Nhiều frame ⇒ nhiều lỗi trang

Thuật toán tối ưu (Optimal algorithm)

- Thay trang sẽ không được sử dụng trong thời gian dài
- Ví dụ 4 frame:

	1	2	3	4	1	2	5	1	2	3	4	5
Frame1	1	1	1	1	1	1	1	1	1	1	4	4
Frame2		2	2	2	2	2	2	2	2	2	2	2
Frame3			3	3	3	3	3	3	3	3	3	3
Frame4				4	4	4	5	5	5	5	5	5
Lỗi	x	x	x	x			x				X	

5 lỗi

- Làm thế nào biết được điều này? Không thực tế!
- Được sử dụng để đánh giá hiệu suất thuật toán sử dụng

Thuật toán LRU (Least Recently Used)

- Chuỗi tham chiếu: 1, 2, 3, 4, 1, 2, 5, 1, 2, 3, 4, 5

	1	2	3	4	1	2	5	1	2	3	4	5
Frame1	1	1	1	1	1	1	1	1	1	1	1	5
Frame2		2	2	2	2	2	2	2	2	2	2	2
Frame3			3	3	3	3	5	5	5	5	4	4
Frame4				4	4	4	4	4	4	3	3	3
Lỗi	x	x	x	x			x			x	x	x

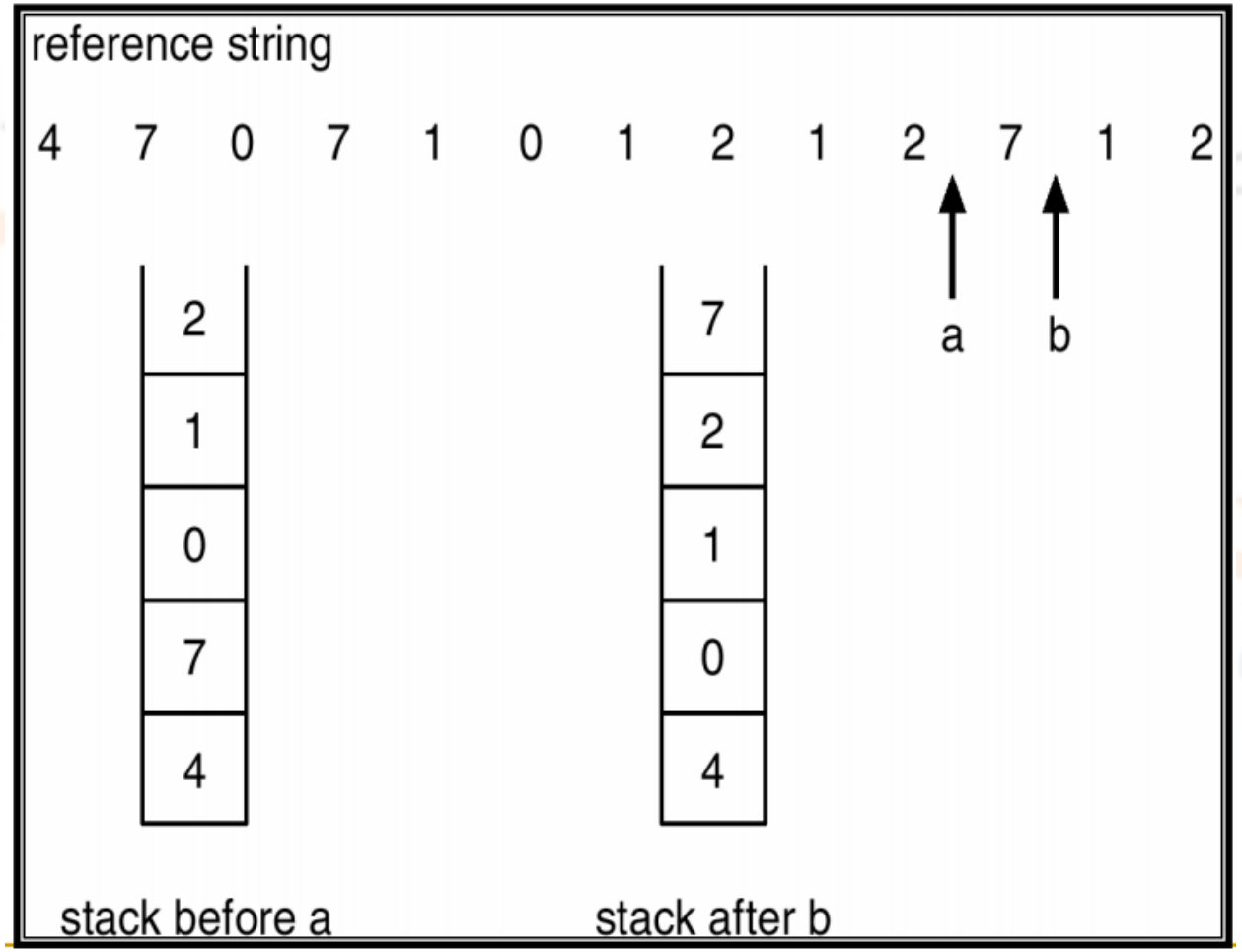
8 lỗi

- Thực thi counter
  - Mọi phần tử trang có một counter; mỗi lần trang được tham chiếu đến → cập nhật counter bằng thời điểm tham chiếu mới.
  - Khi một trang cần thay đổi, xem xét các counter để xác định trang nạn nhân

Thuật toán LRU

- Cài đặt bằng ngăn xếp
  - Lưu giữ số hiệu trang trong một ngăn xếp
    - Cài đặt một danh sách liên kết kép
  - Tham chiếu trang:
    - Chuyển lên đầu ngăn xếp
    - Cần thay đổi tổng cộng 6 con trỏ
  - Không đòi hỏi tìm kiếm khi thay trang

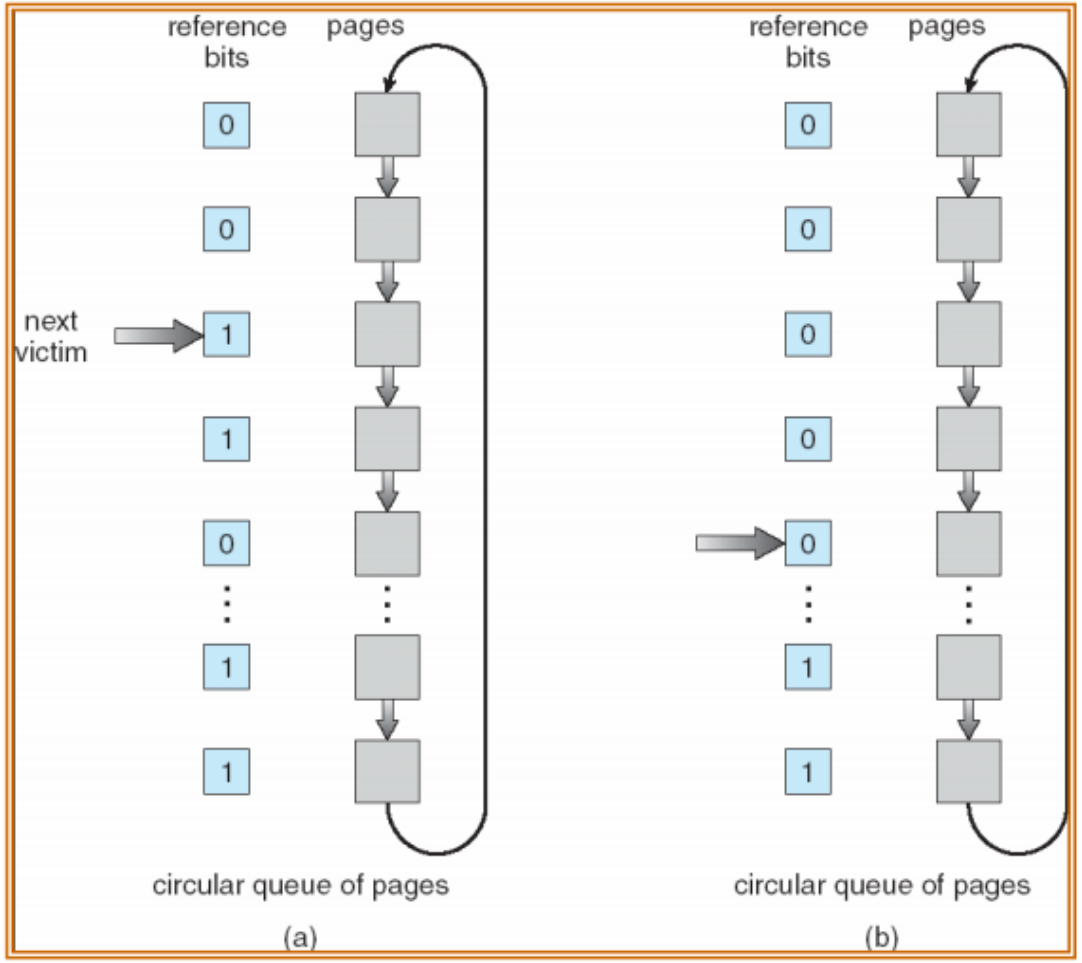
Sử dụng một ngăn xếp để lưu trữ hầu hết các tham chiếu mới



Các thuật toán xấp xỉ LRU

- Bit tham chiếu
  - Mỗi trang liên kết với một bit, bit này được khởi tạo bằng 0
  - Khi một trang được tham chiếu đến, bit được thiết lập bằng 1
  - Thay thế bit 0 (nếu có). Tuy nhiên ta không biết thứ tự thay thế.
- Cơ hội thứ hai
  - Cần bit tham chiếu.
  - Thay thế đồng hồ.
  - Nếu trang chuẩn bị được thay thế (theo thứ tự đồng hồ) có bit tham chiếu = 1.
    - Thiết lập bit tham chiếu bằng 0.
    - Để lại trang đó trong bộ nhớ.
    - Thay thế trang kế tiếp (theo thứ tự đồng hồ), theo cùng một số luật.

Thuật toán Cơ hội thứ hai



Cấp phát frame

- Mỗi tiến trình cần một số lượng ít nhất các trang cần dùng
- Ví dụ: IBM 370 – cần 6 trang để thực hiện lệnh SS MOVE:
  - Lệnh 6 bytes, lưu trong 2 trang
  - 2 trang để xử lý from
  - 2 trang để xử lý to
- Hai lược đồ cấp phát cơ bản
  - Cấp phát cố định
  - Cấp phát ưu tiên

Cấp phát cố định

- Cấp phát đều – Ví dụ: Nếu có 100 frame và 5 tiến trình, cấp cho mỗi tiến trình 20 frame.
- Cấp phát tỉ lệ – Cấp phát theo kích cỡ của tiến trình
  - s_i = size of process p_i
  - $S = \sum s_i$
  - m = total number of frames
  - a_i = allocation for $p_i = \frac{s_i}{S} \times m$

$m = 64$

$s_i = 10$

$s_2 = 127$

$a_1 = \frac{10}{137} \times 64 \approx 5$

$a_2 = \frac{127}{137} \times 64 \approx 59$

Cấp phát ưu tiên

- Lược đồ cấp phát tỉ lệ theo độ ưu tiên (thay vì theo kích cỡ)
- Nếu tiến trình P_i phát sinh một lỗi trang
  - Chọn để thay thế một trong các frame của nó
  - Chọn để thay thế một frame từ một tiến trình với độ ưu tiên thấp hơn

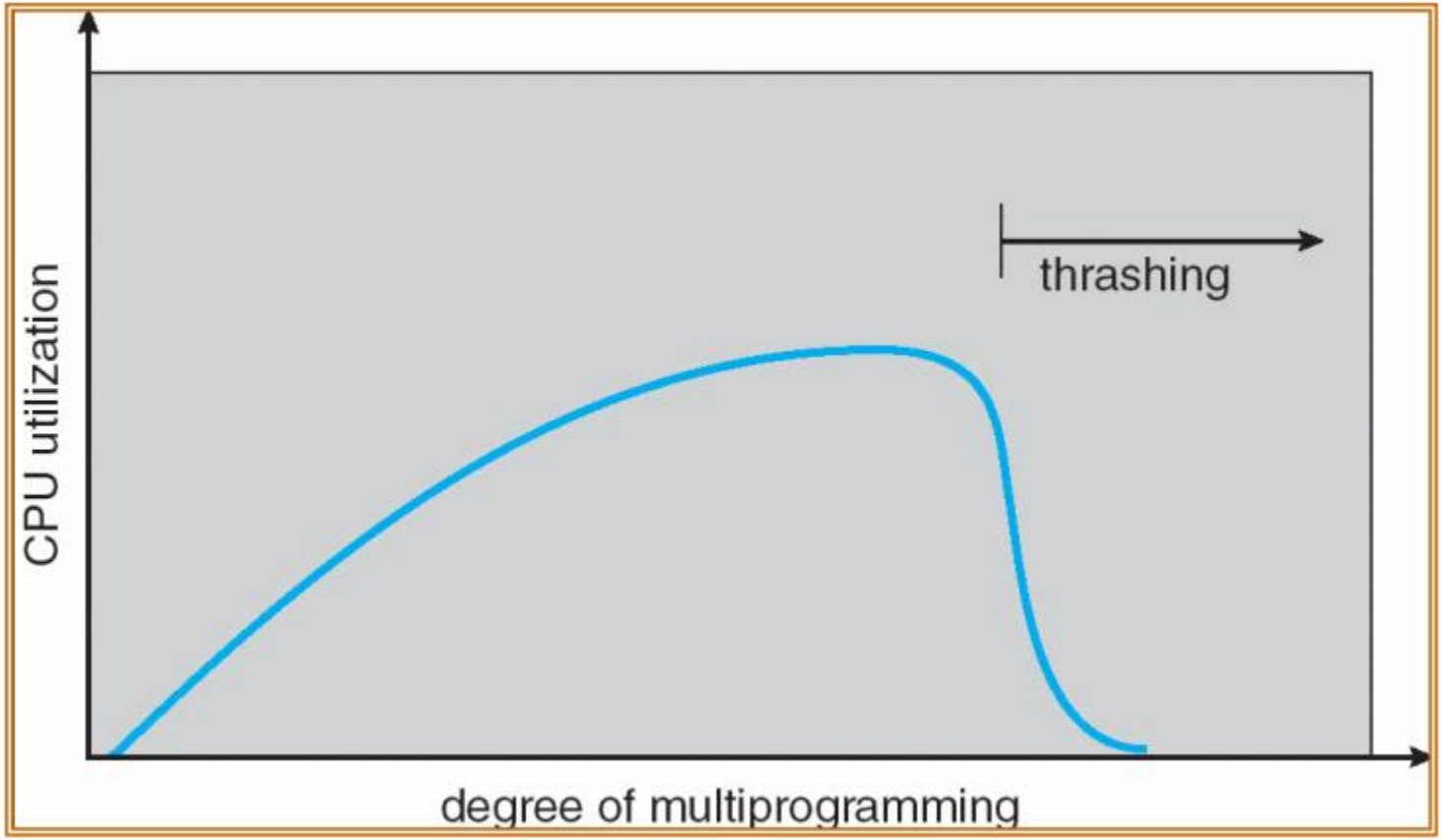
Cấp phát cục bộ và cấp phát toàn cục

- Cấp phát toàn cục - tiến trình lựa chọn một frame thay thế từ tập tất cả các frame; một tiến trình có thể lấy một frame của tiến trình khác.
- Cấp phát cục bộ - tiến trình chỉ lựa chọn frame thay thế từ tập các frame của nó

Thrashing...

- Nếu một tiến trình không có “đủ” trang, tỉ lệ lỗi trang có thể rất cao.
- Tính tận dụng CPU thấp
- Hệ điều hành muốn tăng độ đa chương trình
- Tiến trình mới được thêm vào hệ thống
- Thrashing ≡ một tiến trình dùng nhiều thời gian cho việc thay trang

...Thrashing

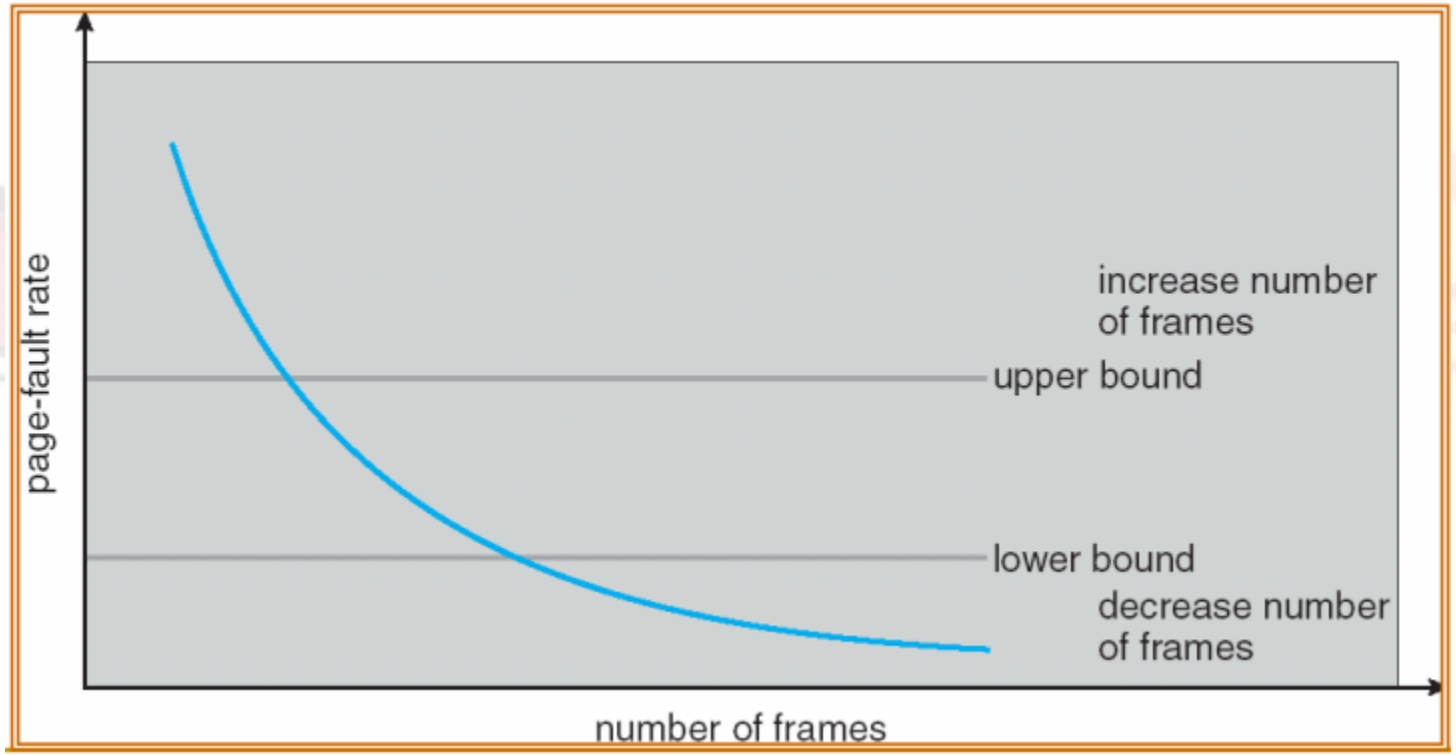


Phân trang theo yêu cầu và thrashing

- Mô hình cục bộ
  - Các tiến trình di trú từ miền cục bộ này sang miền cục bộ khác
  - Các miền cục bộ có thể bị chồng chéo.
- Vì sao lại xuất hiện thrashing?
  - $\sum$ kích cỡ các miền cục bộ > dung lượng bộ nhớ

Lược đồ tần suất lỗi trang

- Thiết lập một tỉ lệ lỗi trang chấp nhận được
  - Nếu tỉ lệ lỗi thực tế thấp, tiến trình giải phóng frame
  - Nếu tỉ lệ lỗi thực tế cao, tiến trình lấy thêm frame



Các vấn đề khác

- Thực thi “tiền phân trang”
  - Giảm một số lượng lớn lỗi trang xuất hiện vào thời điểm bắt đầu tiến trình
  - Phân trước một số trang mà tiến trình có thể cần tới
  - Thực hiện “tiền phân trang” có thể làm lãng phí thiết bị vào ra hoặc bộ nhớ
    - Nếu thời gian tiết kiệm được do lỗi trang lớn hơn thời gian lãng phí → nên dùng tiền phân trang

Kích cỡ trang

- Xem xét các yếu tố để quyết định kích cỡ trang:
  - Phân mảnh
  - Kích cỡ bảng
  - Phụ trội do vào ra
  - Tham chiếu cục bộ

Cấu trúc chương trình

- Cấu trúc chương trình
  - int[128,128] data;
  - Mỗi hàng được lưu trong 1 trang
  - Chương trình 1

```
for (j = 0; j <128; j++)
    for (i = 0; i < 128; i++)
        data[i,j] = 0;
```

128 x 128 = 16,384 lỗi trang

  - Chương trình 2

```
for (i = 0; i < 128; i++)
    for (j = 0; j < 128; j++)
        data[i,j] = 0;
```

128 lỗi trang


Question

Câu hỏi ôn tập...


Q
&
A
dreamstime.com