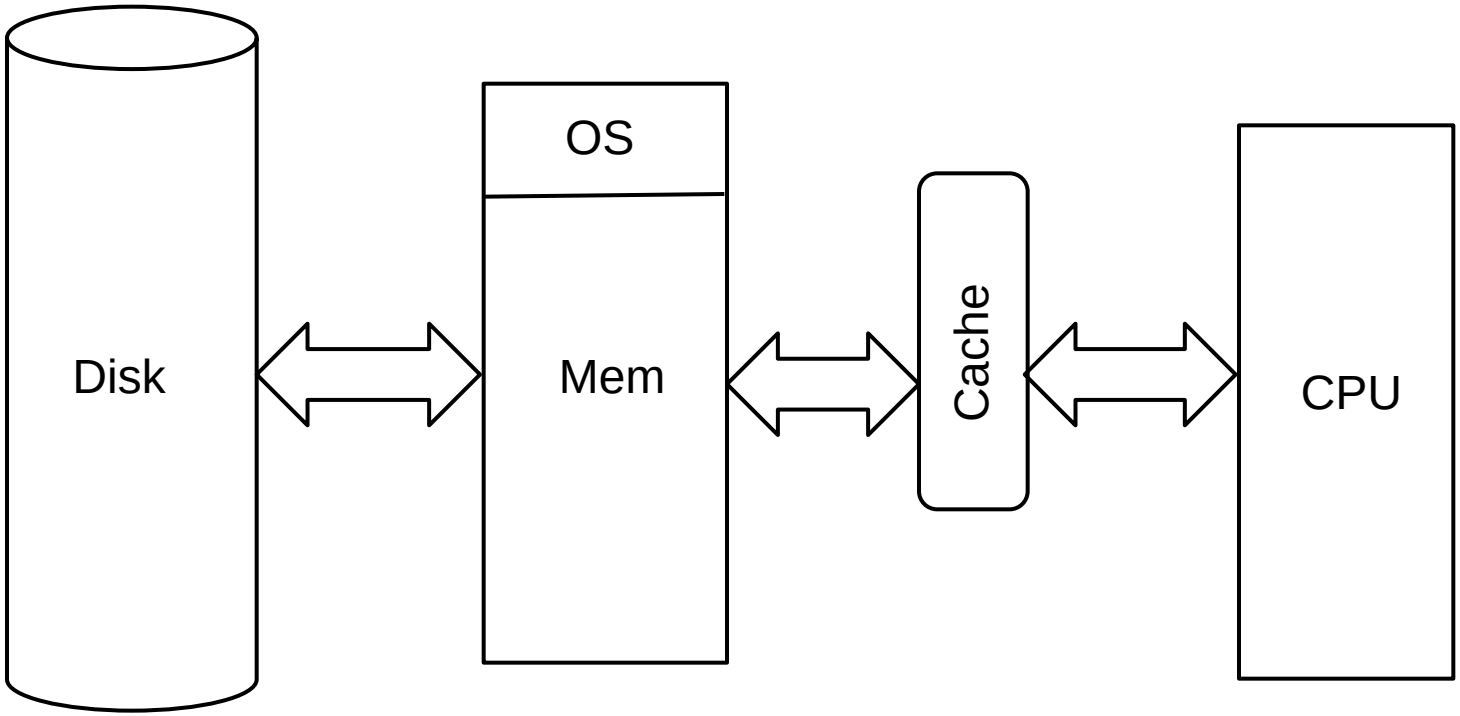
Thay trang
(Page Replacement)

Kiến trúc bộ nhớ



Tổ chức thực hiện

- Một hoặc nhiều chương trình (Process) cùng lúc.
- Dung lượng cấp phát giống nhau hay khác nhau.
- Công việc được thiết kế chạy trên vùng cố định hay có thể thay đổi.
- Công việc có thể phải nạp vào vùng liên tục hay gián đoạn.

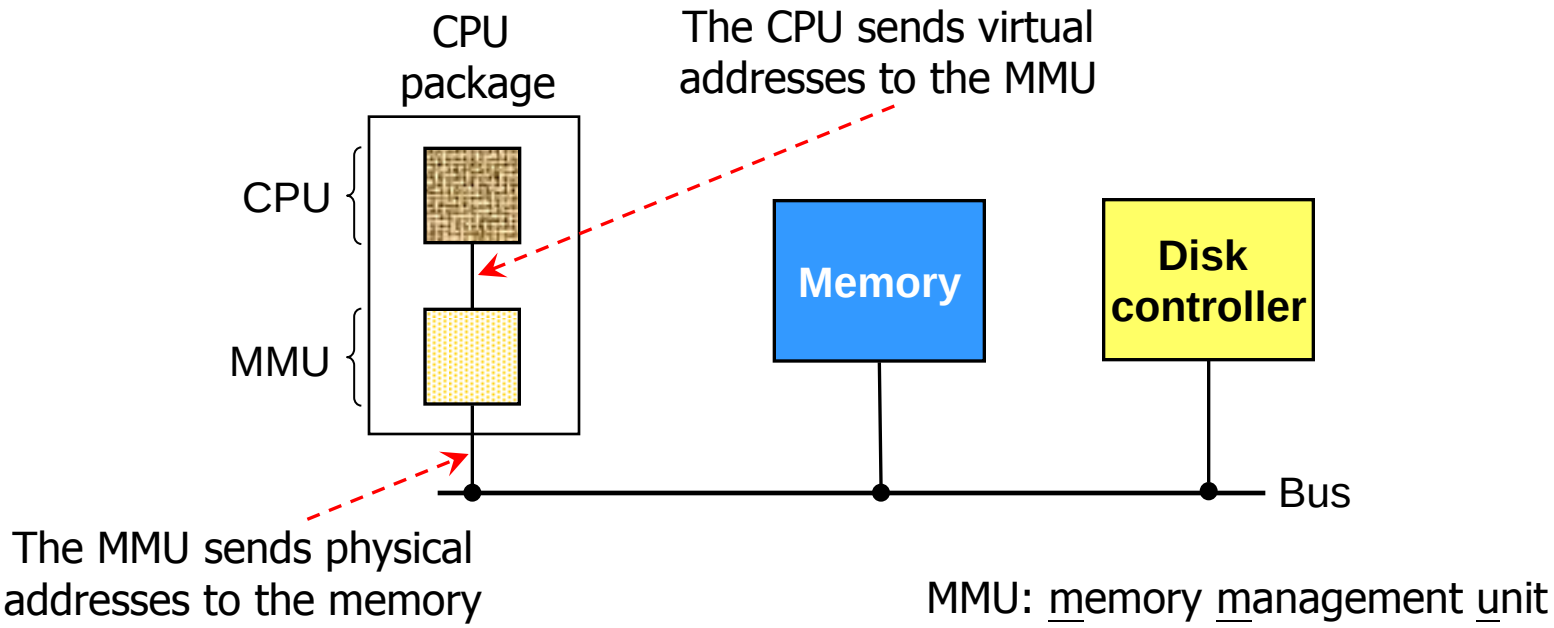
Chiến lược quản lý

- Chiến lược nạp:
  - Nạp theo yêu cầu
  - Nạp trước
- Chiến lược sắp đặt:
  - Best fit
  - First fit
  - Next fit
  - Worst fit

- Chiến lược thay thế: Chọn vùng nào đang bị chiếm để lấy bộ nhớ cấp cho một yêu cầu:
  - OPT
  - Random
  - FIFO
  - LRU
  - NRU
  - Page faults
  - ..

Nhìn lại paging và segmentation

- Các tham chiếu đến bộ nhớ được chuyển đổi động thành địa chỉ thực lúc process đang thực thi



- Một process gồm các phần nhỏ (page hay segment), các phần này được nạp vào các vùng có thể không liên tục trong bộ nhớ chính

Bộ nhớ ảo

- Nhận xét: không phải tất cả các phần của một process cần thiết phải được nạp vào bộ nhớ chính tại cùng một thời điểm
  - Ví dụ:
    - Đoạn mã điều khiển các lỗi hiếm khi xảy ra
    - Các arrays, list, tables được cấp phát bộ nhớ (cấp phát tĩnh) nhiều hơn yêu cầu thực sự
    - Một số tính năng ít khi được dùng của một chương trình
  - Ngay cả khi toàn bộ chương trình đều cần dùng thì có thể không cần dùng toàn bộ cùng một lúc.

Bộ nhớ ảo (tt.)

- Bộ nhớ ảo (virtual memory)
  - Kỹ thuật được hiện thực trong hệ điều hành để cho phép thực thi một quá trình mà chỉ cần giữ trong bộ nhớ chính một phần của không gian địa chỉ luận lý của nó, còn phần còn lại được giữ trên bộ nhớ đại trà (đĩa).
- Ưu điểm của bộ nhớ ảo
  - Số lượng process trong bộ nhớ sẽ nhiều hơn
  - Một process có thể thực thi ngay cả khi kích thước của nó lớn hơn bộ nhớ thực

Bộ nhớ ảo (tt.)

- Thông thường phần của không gian địa chỉ luận lý của quá trình, nếu chưa cần nạp vào bộ nhớ chính, được giữ ở một vùng đặc biệt trên đĩa gọi là không gian tráo đổi (swap space).
  - Ví dụ:
    - swap partition trong Linux
    - file pagefile.sys trong Windows 2K

Tổng quan hiện thực bộ nhớ ảo

- Phần cứng memory management phải hỗ trợ paging và/hoặc segmentation
- OS phải quản lý sự di chuyển của trang/đoạn giữa bộ nhớ chính và bộ nhớ thứ cấp
  - Trong chương này:
    - Chỉ quan tâm đến paging
    - Phần cứng hỗ trợ hiện thực bộ nhớ ảo
    - Các giải thuật của hệ điều hành

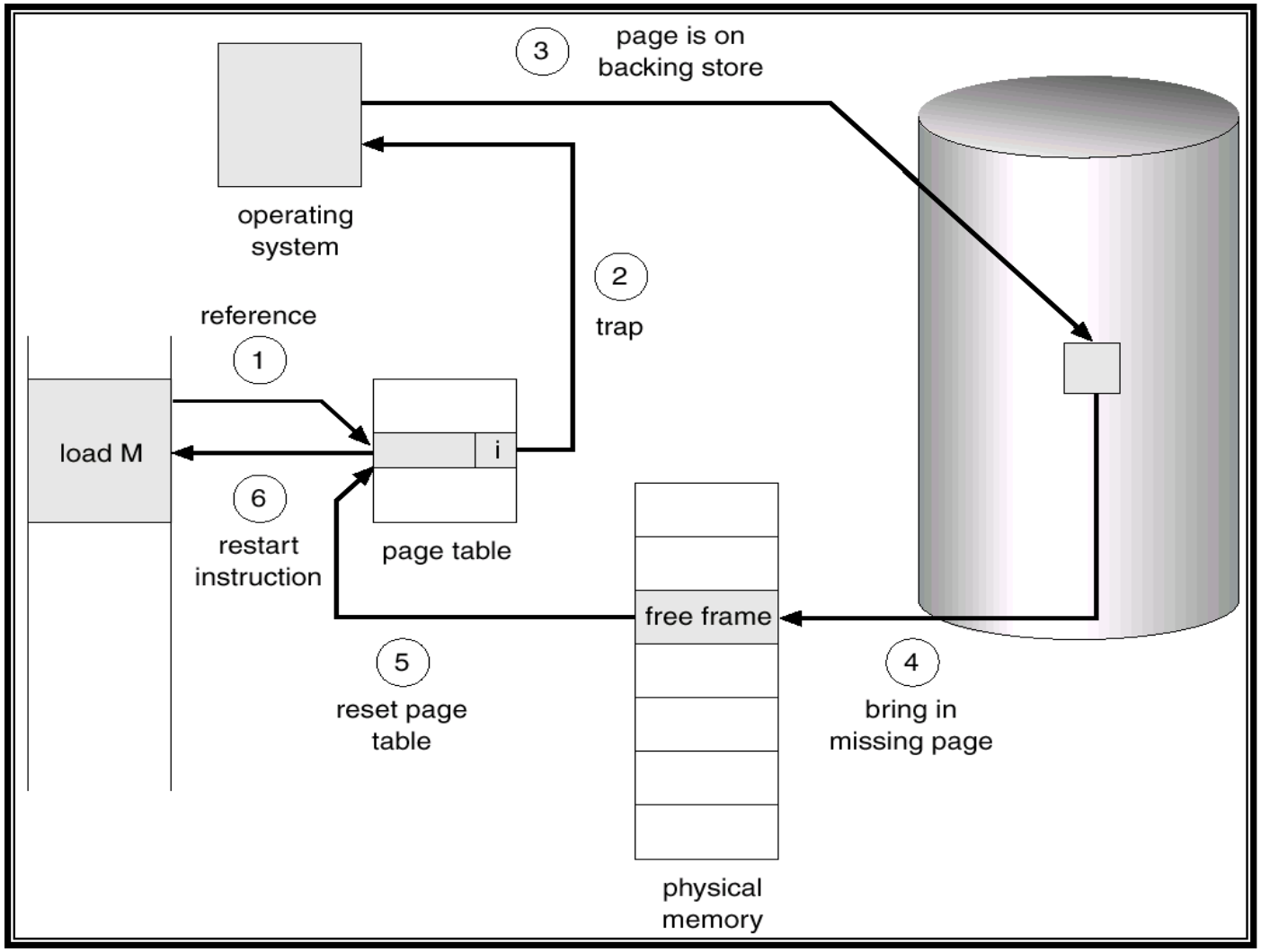
Phần cứng hỗ trợ bộ nhớ ảo

- Sự hỗ trợ của phần cứng đối với phân trang đã được khảo sát trong chương trước. Chỉ có một điểm khác biệt là mỗi mục của bảng phân trang có thêm các bit trạng thái đặc biệt
  - Present bit = 1 $\Rightarrow$ trang hợp lệ và hiện trong memory
    = 0 $\Rightarrow$ trang không hợp lệ hoặc không trong memory
    - Khi có một tham chiếu đến một trang mà không có trong bộ nhớ chính (present bit = 0) thì phần cứng sẽ gây ra một ngắt gọi là page-fault trap
  - Modified bit: cho biết trang có thay đổi kể từ khi được nạp vào memory hay không

Hiện thực bộ nhớ ảo: demand paging

- Demand paging: các trang của quá trình chỉ được nạp vào bộ nhớ chính khi được yêu cầu.
  - Khi quá trình tham chiếu đến một trang mà không có trong bộ nhớ chính (present bit = 0) thì sẽ gây ra page-fault trap kích khởi page-fault service routine (PFSR) của hệ điều hành.
  - PFSR:
    1. Chuyển process về trạng thái blocked
    2. 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
    3. 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.

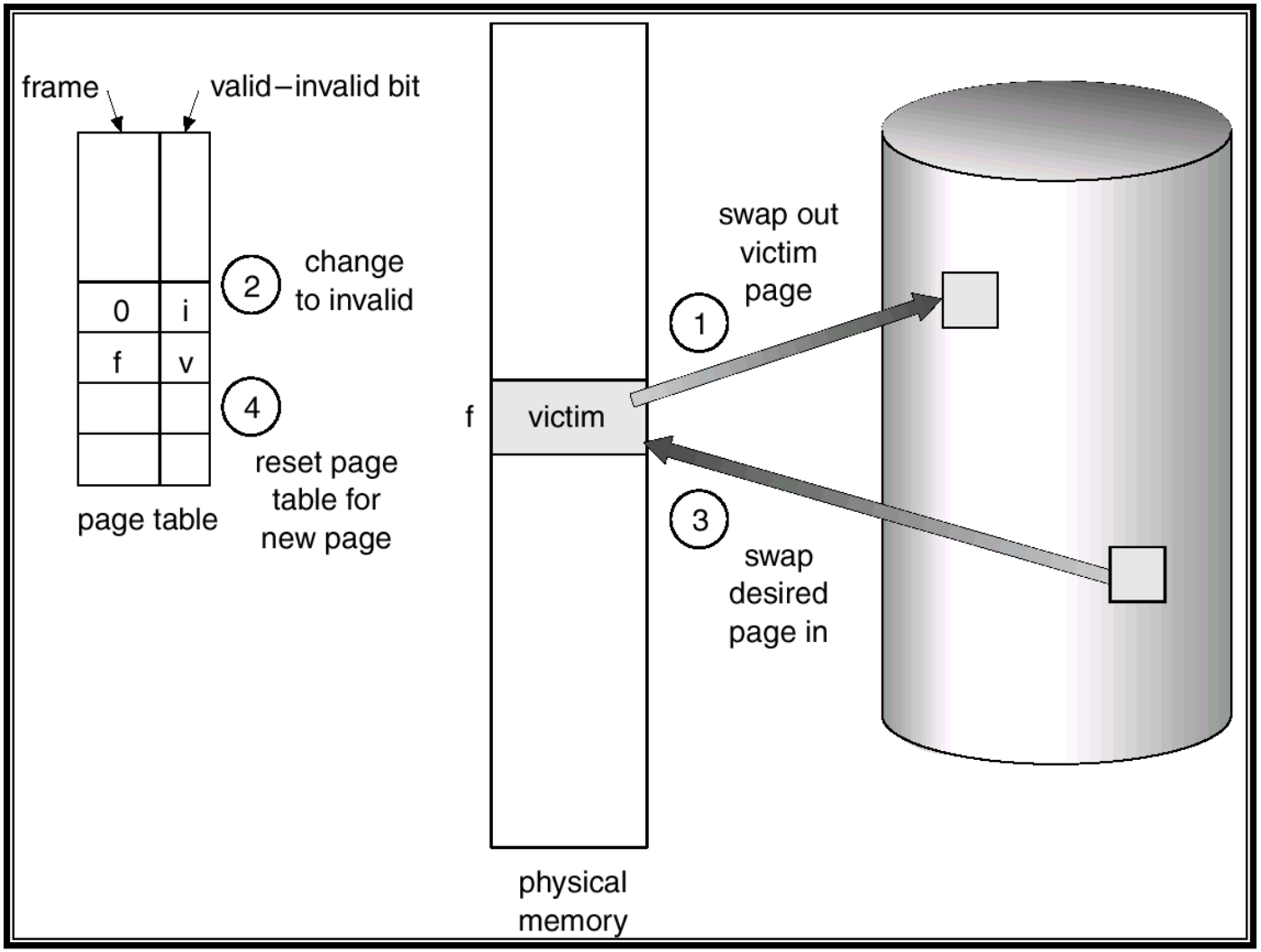
Page fault và các bước xử lý



Thay thế trang nhớ

- Bước 2 của PFSR giả sử tìm được frame trống. Để xử lý được cả trường hợp phải thay trang vì không tìm được frame trống, PFSR được bổ sung như sau:
  1. Xác định vị trí trên đĩa của trang đang cần
  2. Tìm một frame trống:
     a. Nếu có frame trống thì dùng nó
     b. Nếu không có frame trống thì dùng một giải thuật thay trang để chọn một trang hy sinh (victim page)
     c. Ghi victim page lên đĩa; cập nhật page table và frame table tương ứng
  3. Đọc trang đang cần vào frame trống (đã có được từ bước 2); cập nhật page table và frame table tương ứng.

Thay thế trang nhớ (tt.)



Hiện thực demand paging

Hai vấn đề chủ yếu:

- Frame-allocation algorithm
  - Cấp phát cho process bao nhiêu frame?
- Page-replacement algorithm
  - Chọn frame của process sẽ được thay thế trang nhớ
  - Mục tiêu: *số lượng page fault nhỏ nhất*
  - Được đánh giá bằng cách thực thi giải thuật đối với một chuỗi tham chiếu bộ nhớ (memory reference string) và xác định số lần xảy ra page fault

- Ví dụ

Thứ tự tham chiếu các địa chỉ nhớ, với page size = 100:

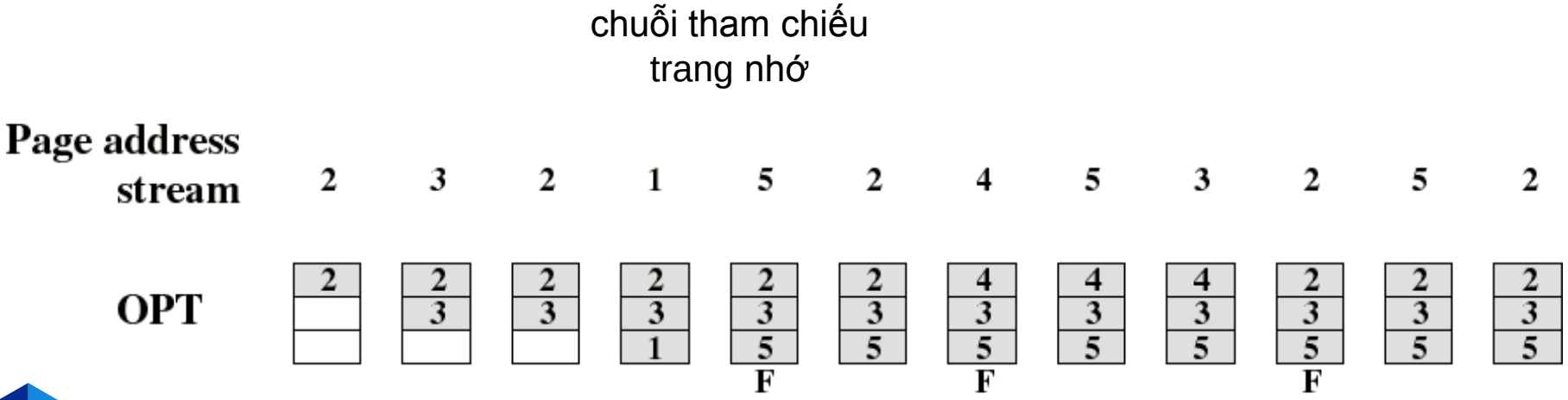
0100, **04**32, **01**01, **06**12, **01**02, **01**03, **01**04, **01**01, **06**11, **01**02, **01**03, **01**04, **01**01, **06**10, **01**02, **01**03, **01**04, **01**01, **06**09, **01**02, **01**05

⇒ chuỗi tham chiếu trang nhớ/ bộ nhớ

1, **4**, **1**, **6**, **1**, **1**, **1**, **1**, **6**, **1**, **1**, **1**, **1**, **6**, **1**, **1**, **1**, **1**, **6**, **1**, **1**

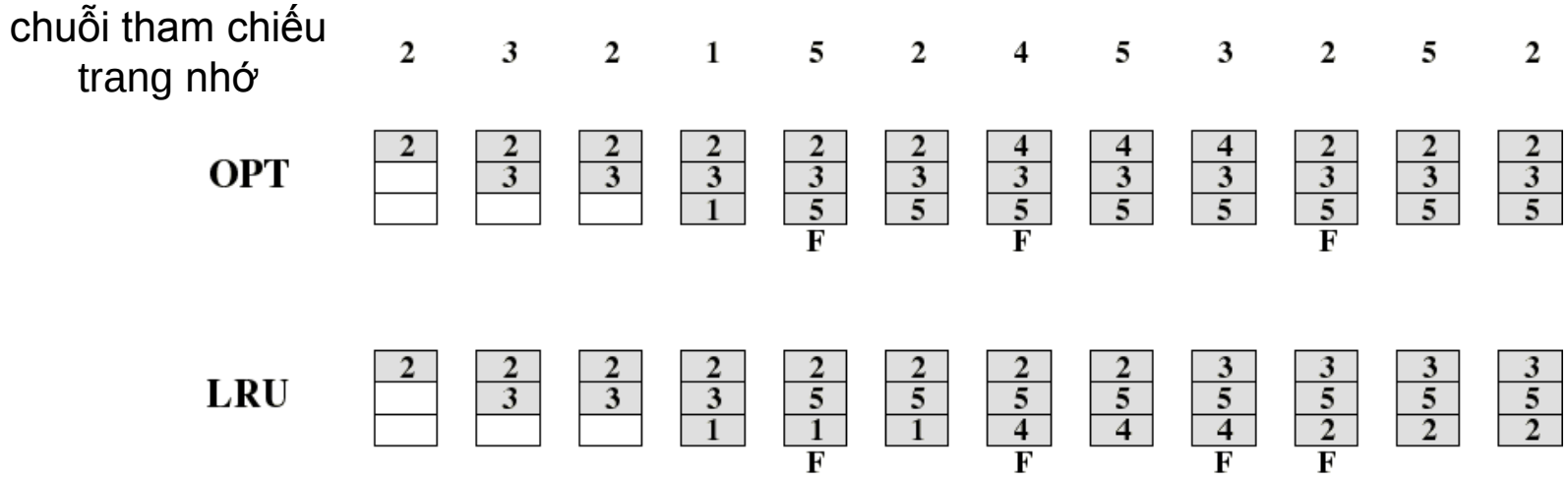
Giải thuật thay trang OPT (OPTimal)

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



Giải thuật thay trang LRU *(Least Recently Used)*

- Thay thế trang nhớ không được tham chiếu lâu nhất



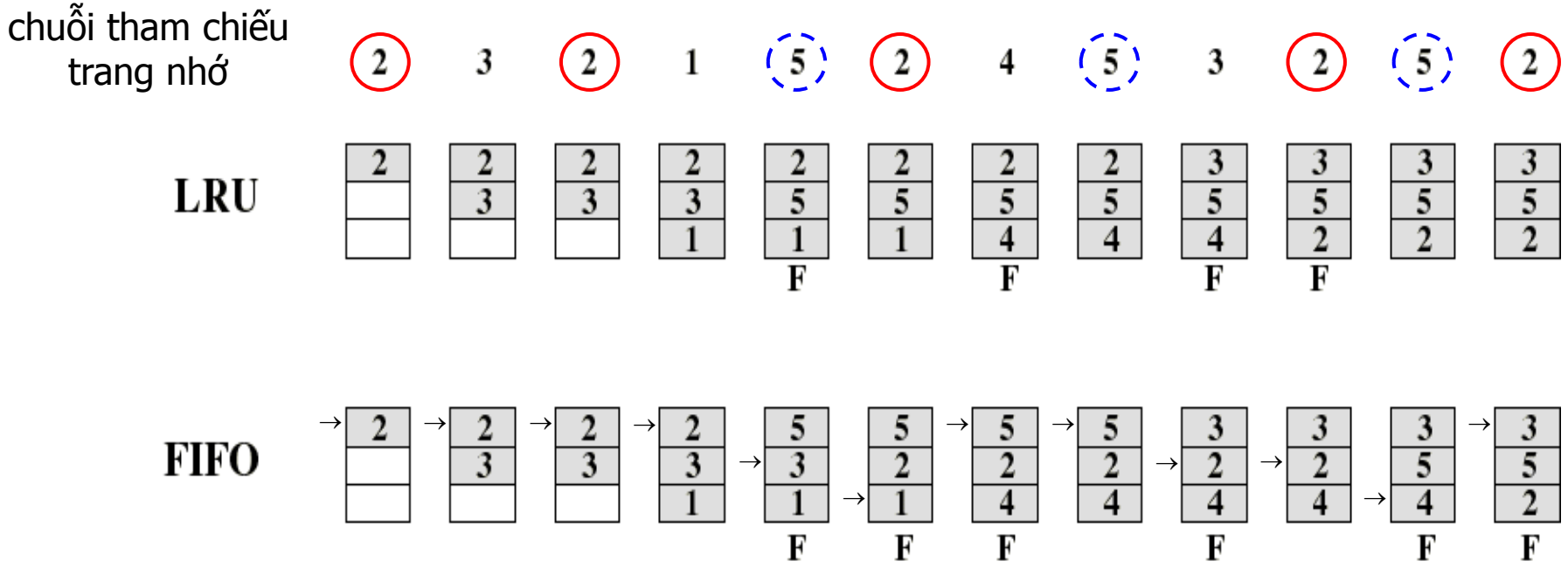
- Mỗi trang được ghi nhận (trong bảng phân trang) thời điểm được tham chiếu. Trang LRU là trang nhớ có thời điểm được tham chiếu nhỏ nhất
  - OS tốn chi phí tìm kiếm trang nhớ LRU này mỗi khi có thay trang. Do vậy, LRU cần sự hỗ trợ của phần cứng. Ít CPU cung cấp đủ sự hỗ trợ phần cứng cho giải thuật LRU.

Giải thuật thay trang FIFO

- Xem các frame được cấp phát cho process như là circular buffer
  - Khi bộ đệm đầy, trang nhớ cũ nhất sẽ được thay thế: first-in first-out
  - Một trang nhớ hay được dùng sẽ thường là trang cũ nhất nên hay bị thay thế bởi giải thuật FIFO
  - Hiện thực đơn giản: chỉ cần một con trỏ xoay vòng các frame của process

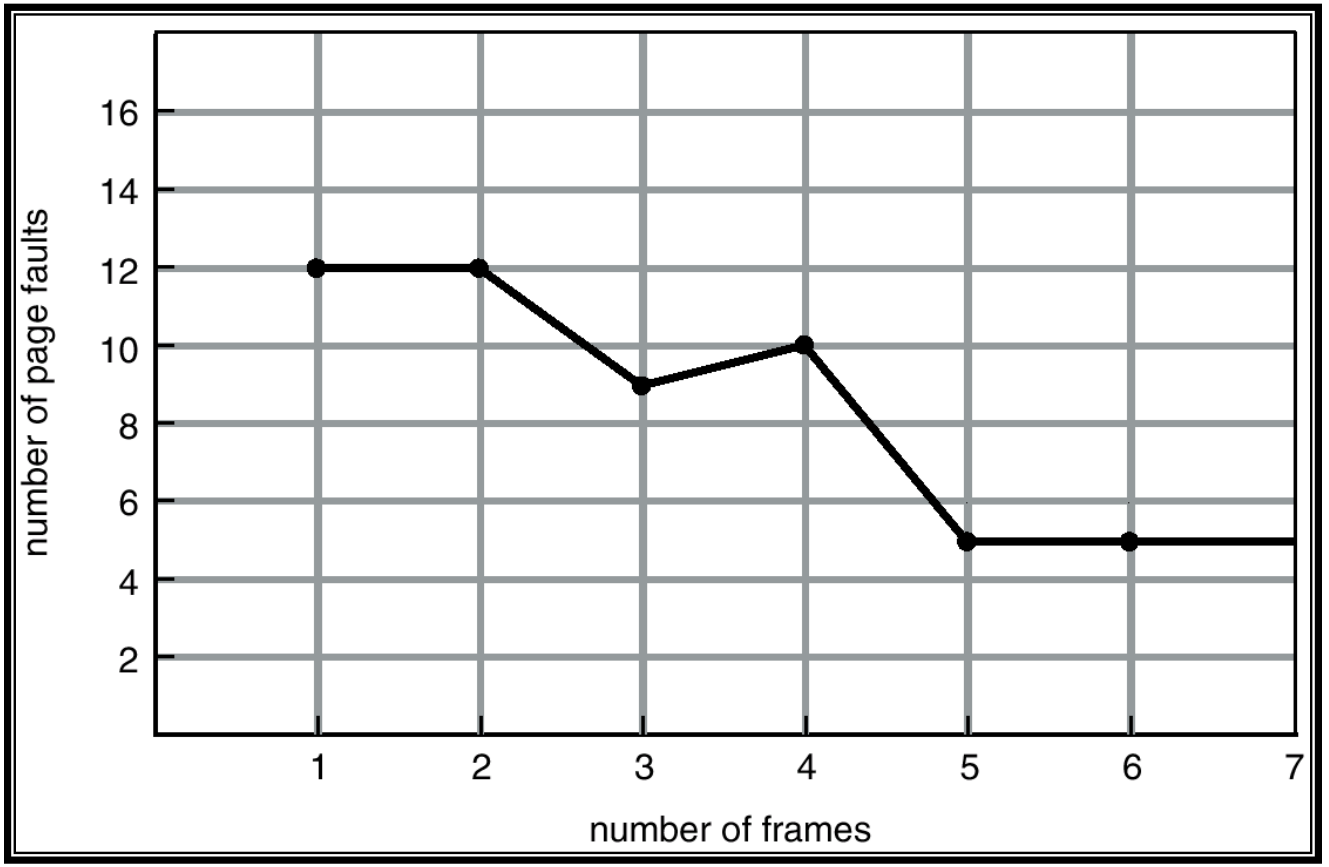
LRU vs. FIFO

- So sánh các giải thuật thay trang LRU và FIFO



Giải thuật FIFO: Belady's anomaly

- Số page fault tăng mặc dầu quá trình đã được cấp nhiều frame hơn.



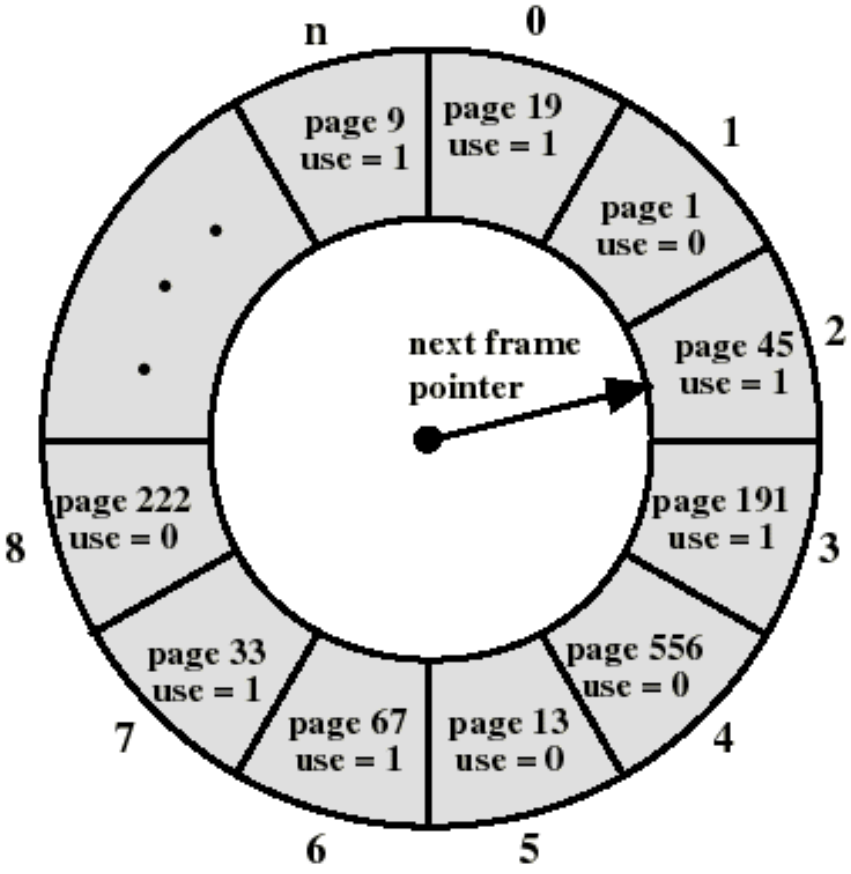
Hiện tượng bất bình thường trong FIFO

Hành vi CT	Trang (9 faults)	Thay trang theo FIFO (3 trang)			Trang (10 faults)	Thay trang theo FIFO (4 trang)			
A	Fault	A	-	-	Fault	A	-	-	-
B	Fault	B	A	-	Fault	B	A	-	-
C	Fault	C	B	A	Fault	C	B	A	-
D	Fault	D	C	B	Fault	D	C	B	A
A	Fault	A	D	C	N-F	D	C	B	A
B	Fault	B	A	D	N-F	D	C	B	A
E	Fault	E	B	A	Fault	E	D	C	B
A	N-F	E	B	A	Fault	A	E	D	C
B	N-F	E	B	A	Fault	B	A	E	D
C	Fault	C	E	B	Fault	C	B	A	E
D	Fauly	D	C	E	Fault	D	C	B	A
E	N-F	D	C	E	Fault	E	D	C	B

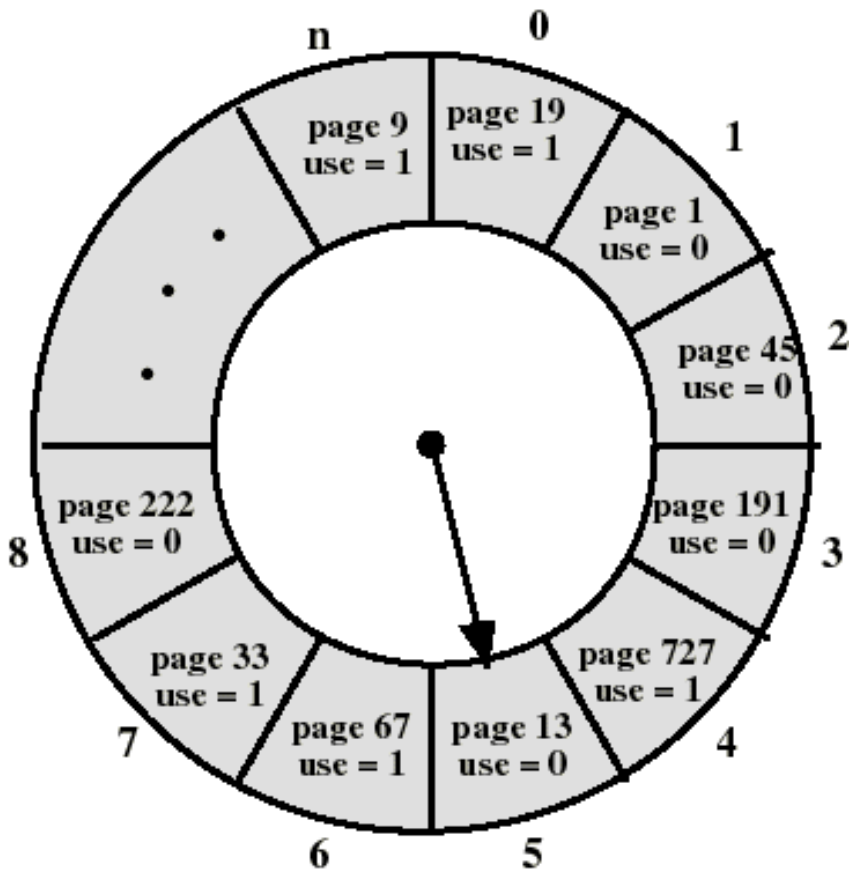
Giải thuật thay trang clock

- Các frame cấp cho process được xem như một bộ đệm xoay vòng (circular buffer)
- Khi một trang được thay, con trỏ sẽ chỉ đến frame kế tiếp trong buffer
- Mỗi frame có một use bit. Bit này được thiết lập trị 1 khi
  - Một trang được nạp vào frame
  - Trang chứa trong frame được tham chiếu
- Khi cần thay thế một trang nhớ, trang nhớ nằm trong frame đầu tiên có use bit bằng 0 sẽ được thay thế.
  - Trên đường đi tìm trang nhớ thay thế, tất cả use bit được reset về 0

Giải thuật thay trang clock (tt.)

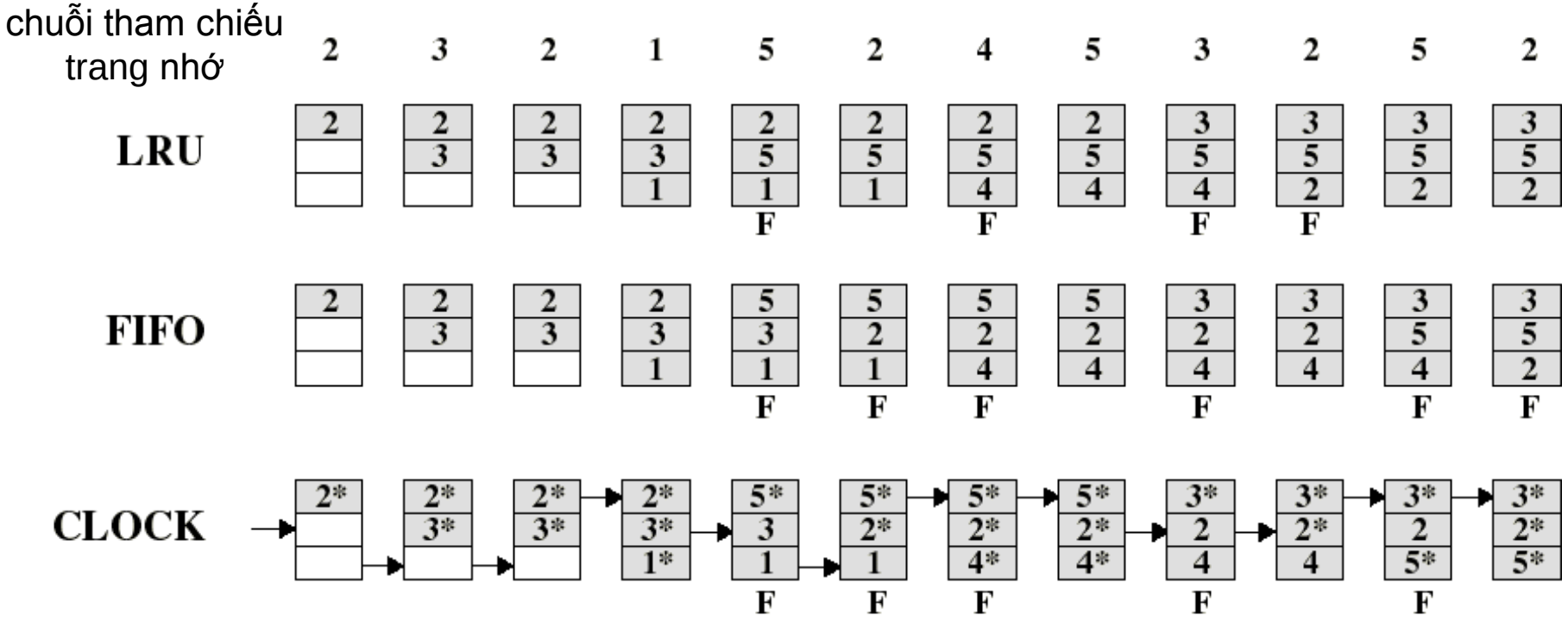


(a) State of buffer just prior to a page replacement



(b) State of buffer just after the next page replacement

So sánh LRU, FIFO, và clock



- Dấu *: use bit tương ứng được thiết lập trị 1
- Giải thuật clock bảo vệ các trang thường được tham chiếu bằng cách thiết lập use bit bằng 1 với mỗi lần tham chiếu
- Một số kết quả thực nghiệm cho thấy clock có hiệu suất gần với LRU

Not-Used-Recently

- Reference bit = 0 : trang chưa được tham chiếu
  1 : trang đ được tham chiếu
  Modified bit = 0 : trang chưa bị thay đổi
  1 : trang đ bị thay đổi

Group 1	Chưa tham chiếu (0)	Chưa thay đổi (0)
Group 2	Chưa tham chiếu (0)	Thay đổi (1)
Group 3	Tham chiếu (1)	Chưa thay đổi (0)
Group 4	Tham chiếu (1)	Thay đổi (1)

Số lượng frame cấp cho process

- OS phải quyết định cấp cho mỗi process bao nhiêu frame.
  - Cấp ít frame $\Rightarrow$ nhiều page fault
  - Cấp nhiều frame $\Rightarrow$ giảm mức độ multiprogramming
- 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

Chiến lược cấp phát tĩnh

- Cấp phát bằng nhau
  - Ví dụ, có 100 frame và 5 process thì mỗi process được 20 frame
- Cấp phát theo tỉ lệ: dựa vào kích thước process

$s_i = \text{size of process } p_i$

$S = \sum s_i$

$m = \text{total number of frames}$

$$a_i = \text{allocation for } p_i = \frac{s_i}{S} \times m$$

Ví dụ

$m = 64$

$s_1 = 10$

$s_2 = 127$

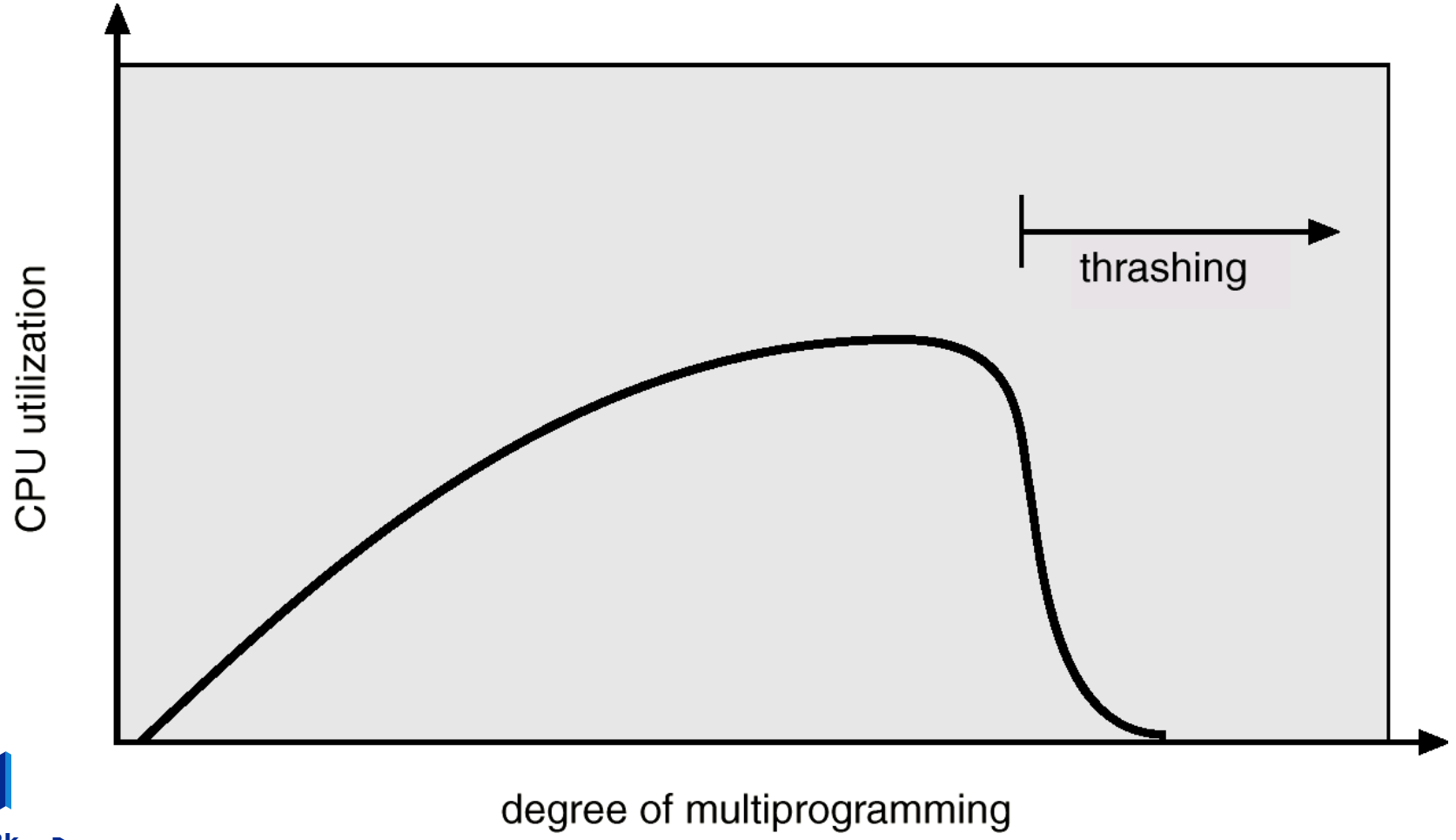
$$a_1 = \frac{10}{137} \times 64 \approx 5$$

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

Thrashing

- Nếu một process không có đủ số frame cần thiết thì tỉ số page faults/sec rất cao. Điều này khiến giảm hiệu suất CPU rất nhiều.
  - Ví dụ: một vòng lặp N lần, mỗi lần tham chiếu đến địa chỉ nằm trong 4 trang nhớ trong khi đó process chỉ được cấp 3 frame.
    - Chuỗi tham chiếu trang: 012301230123
- Thrashing: hiện tượng các trang nhớ của một process bị hoán chuyển vào/ra liên tục.

Thrashing diagram

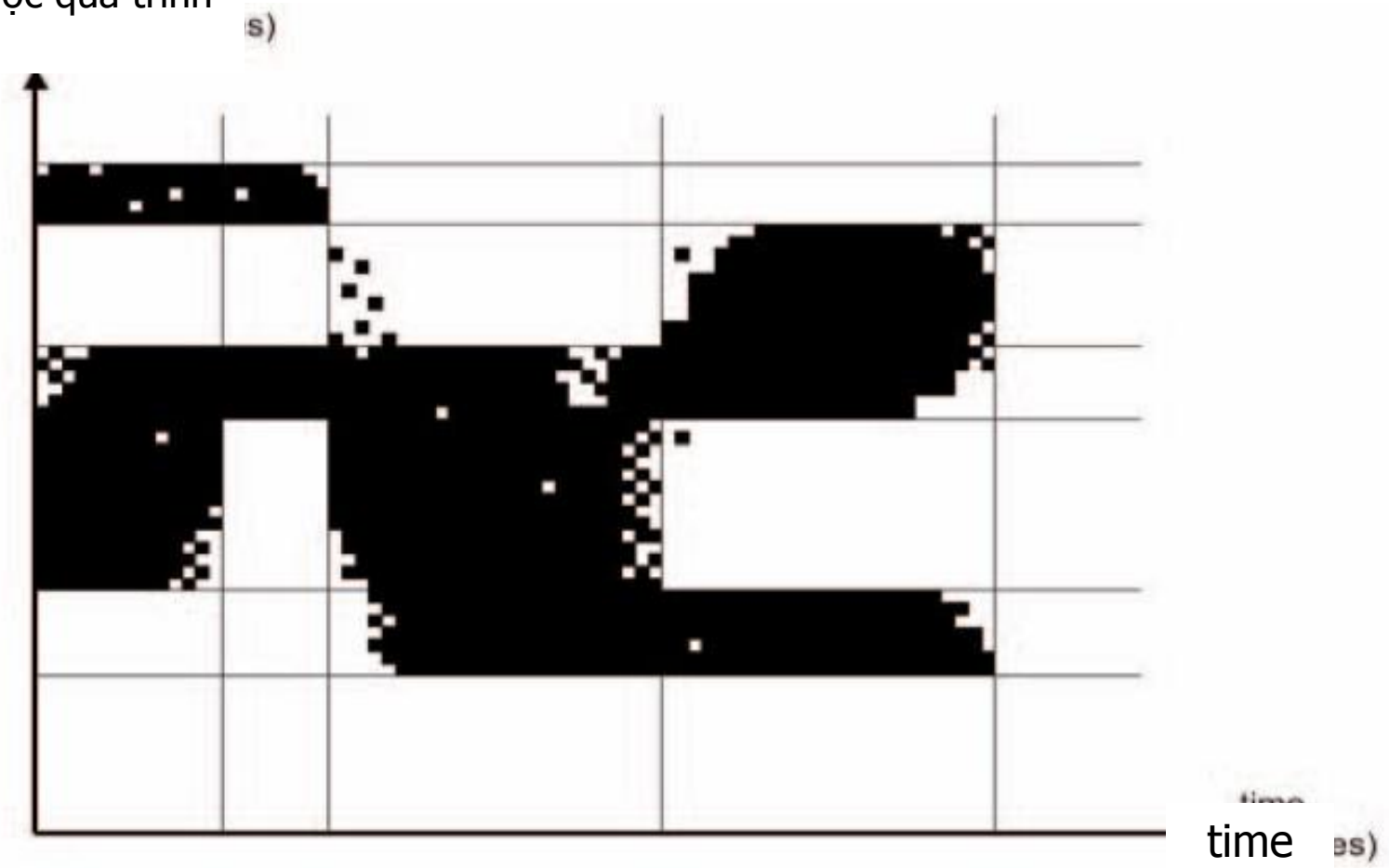


Nguyên lý locality (tt.)

- Để hạn chế thrashing, hệ điều hành phải cung cấp cho process càng "đủ" frame càng tốt. Bao nhiêu frame thì đủ cho một process thực thi hiệu quả?
- Nguyên lý locality (locality principle)
  - Locality là tập các trang được tham chiếu gần nhau
    - Trong ví dụ trước, locality sẽ bao gồm 4 trang
  - Process gồm nhiều locality, và trong quá trình thực thi, process sẽ "chuyển từ locality này sang locality khác"
    - Ví dụ khi một thủ tục được gọi thì sẽ có một locality mới. Trong locality này, tham chiếu bộ nhớ bao gồm lệnh của thủ tục, biến cục bộ và một phần biến toàn cục. Khi thủ tục kết thúc, process sẽ thoát khỏi locality này (và có thể quay lại sau này).

Nguyên lý locality (tt.)

Các trang được quá trình tham khảo



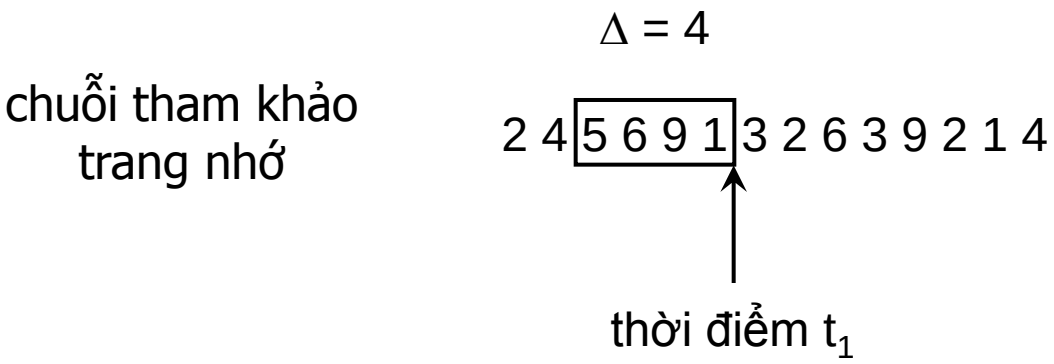
Hình từ "The locality principle", P.J.Denning

Nguyên lý locality (tt.)

- "*The working set idea was based on an implicit assumption that the pages seen in the backward window were highly likely to be used again in the immediate future.*" (P. Denning)

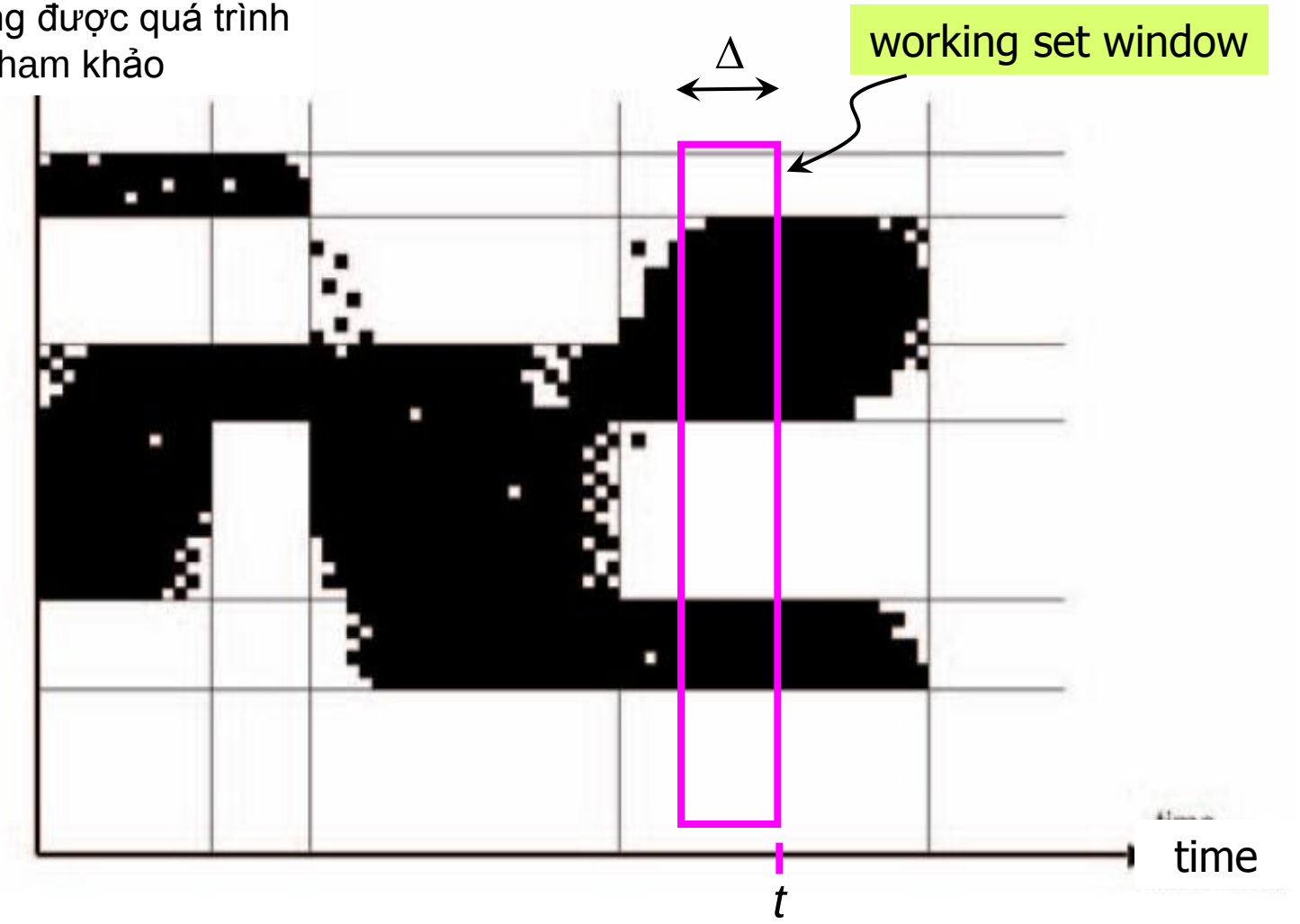
Hạn chế thrashing: Giải pháp working set

- Giải pháp working set còn được gọi là working set model; được thiết kế dựa trên nguyên lý locality.
- Cần xác định process "thực sự" sử dụng bao nhiêu frame.
  - Tham số Δ của working-set window xác định số lượng các tham chiếu trang nhớ của process gần đây nhất cần được quan sát.
    - Ví dụ



Hạn chế thrashing: Giải pháp working set (tt.)

Các trang được quá trình tham khảo

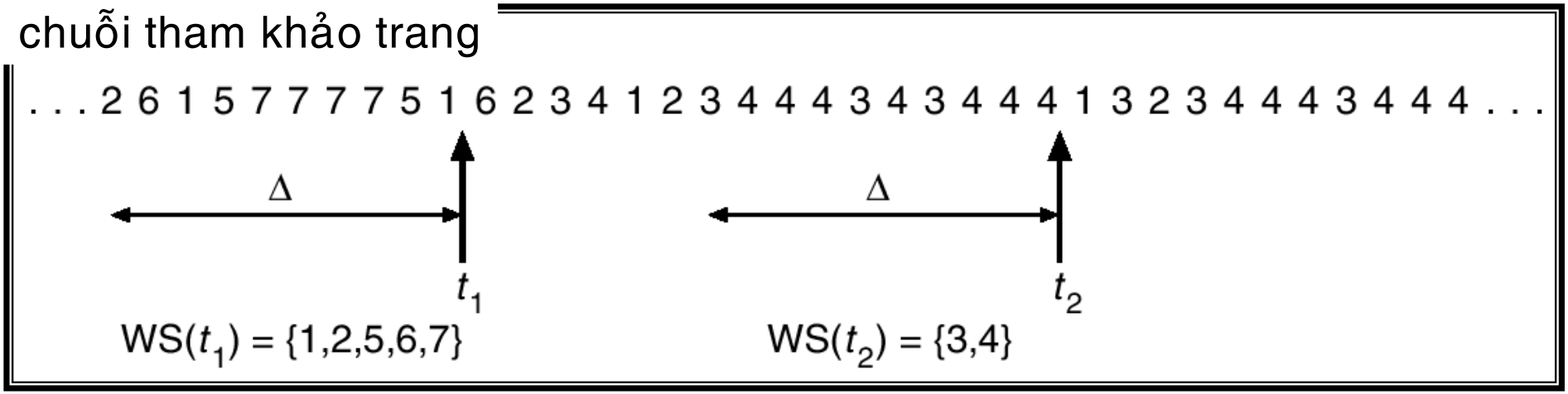


Hình từ "The locality principle", P.J.Denning

Hạn chế thrashing: Giải pháp working set (tt.)

- Định nghĩa: working set của process P_i , ký hiệu WS_i , là tập các số trang trong working set window.

Ví dụ: □ = 10 và chuỗi tham khảo trang

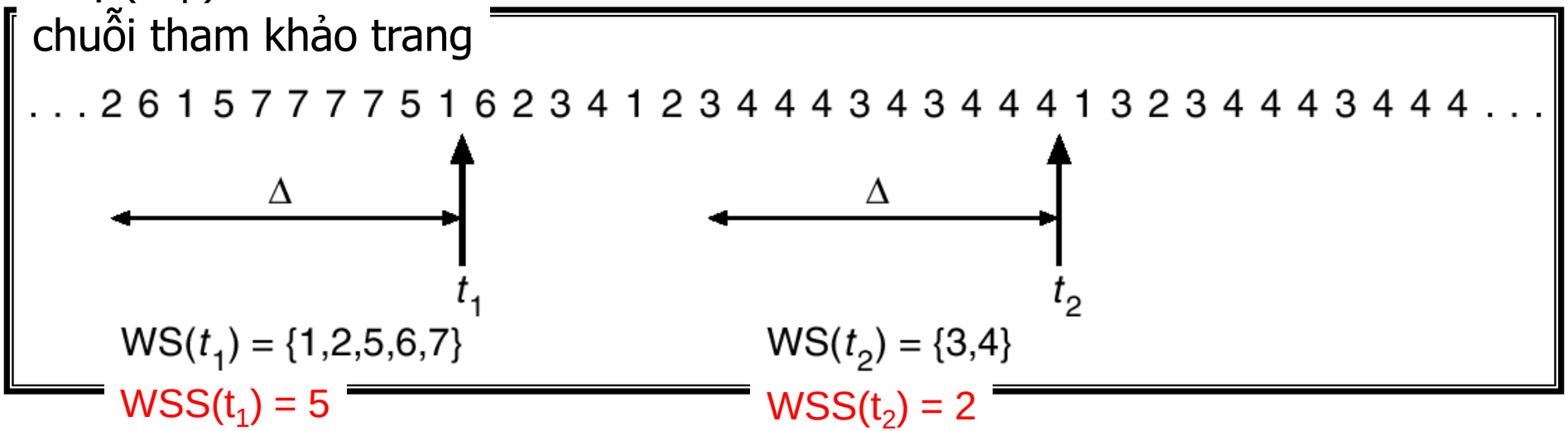


- Nhận xét:
  - Δ quá nhỏ ⇒ không đủ bao phủ toàn bộ locality.
  - Δ quá lớn ⇒ bao phủ nhiều locality khác nhau.
  - Δ = ∞ ⇒ bao gồm tất cả các trang được sử dụng.
- Dùng working set của một process để xấp xỉ locality của nó.

Hạn chế thrashing: Giải pháp working set (tt.)

- Định nghĩa WSS_i là kích thước của working set của P_i :
  - WSS_i = số lượng các trang trong WS_i

Ví dụ (tiếp): ∆ = 10 và chuỗi tham khảo trang



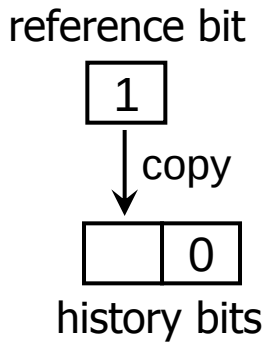
Hạn chế thrashing: Giải pháp working set (tt.)

- Đặt D = Σ WSS_i = tổng các working-set size của mọi process trong hệ thống.
  - Nhận xét: Nếu D > m (số frame của hệ thống) thì sẽ xảy ra thrashing.

- Giải pháp working set:
  - Khi khởi tạo một quá trình: cung cấp cho quá trình số lượng frame thỏa mản working-set size của nó.
  - Nếu D > m thì suspend một trong các process.
    - Các trang của quá trình được chuyển ra đĩa cứng và các frame của nó được thu hồi.

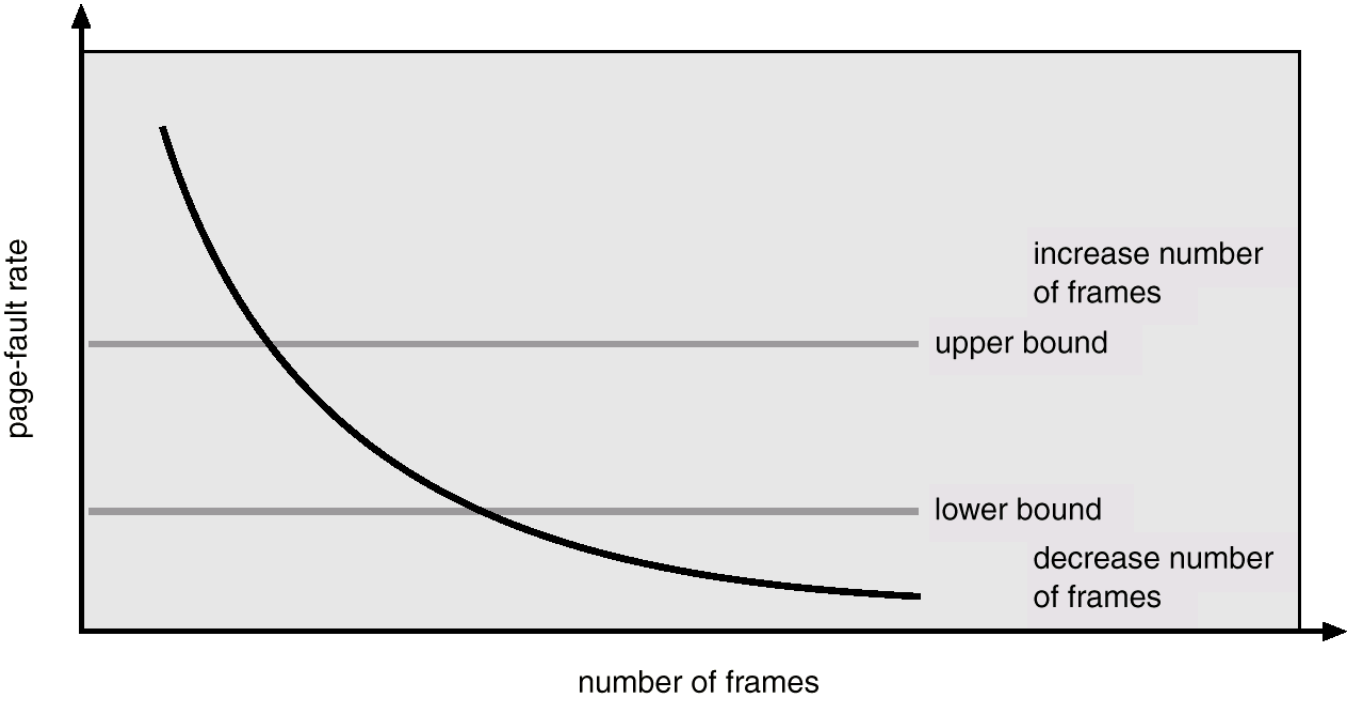
Xấp xỉ working set

- Theo định nghĩa, một trang là ở trong working set nếu nó được tham chiếu trong working-set window
- Giả sử hardware hỗ trợ một reference bit cho mỗi page: khi page được tham chiếu, reference bit được set thành 1.
  - Dùng interval timer kết hợp với reference bit để xấp xỉ working set



  - Ví dụ: Δ = 10.000
    - Timer interrupt định kỳ, sau mỗi 5000 đơn vị thời gian.
    - Giữ trong bộ nhớ 2 bit (history bits) cho mỗi trang nhớ.
    - Khi timer interrupt xảy ra, shift history bits một vị trí sang phải, copy reference bit vào history bit trái, và reset reference bit = 0.
    - Trang nào có history bits chứa 1 thì thuộc về working set.

Hạn chế thrashing: Điều khiển page-fault rate



- Dùng giải thuật PFF (Page-Fault Frequency) để điều khiển page-fault rate (số page-faults/sec) của process:
  - Page-fault rate quá thấp: process có quá nhiều frame ⇒ giảm số frame.
  - Page-fault rate quá cao: process cần thêm frame ⇒ cấp thêm frame.