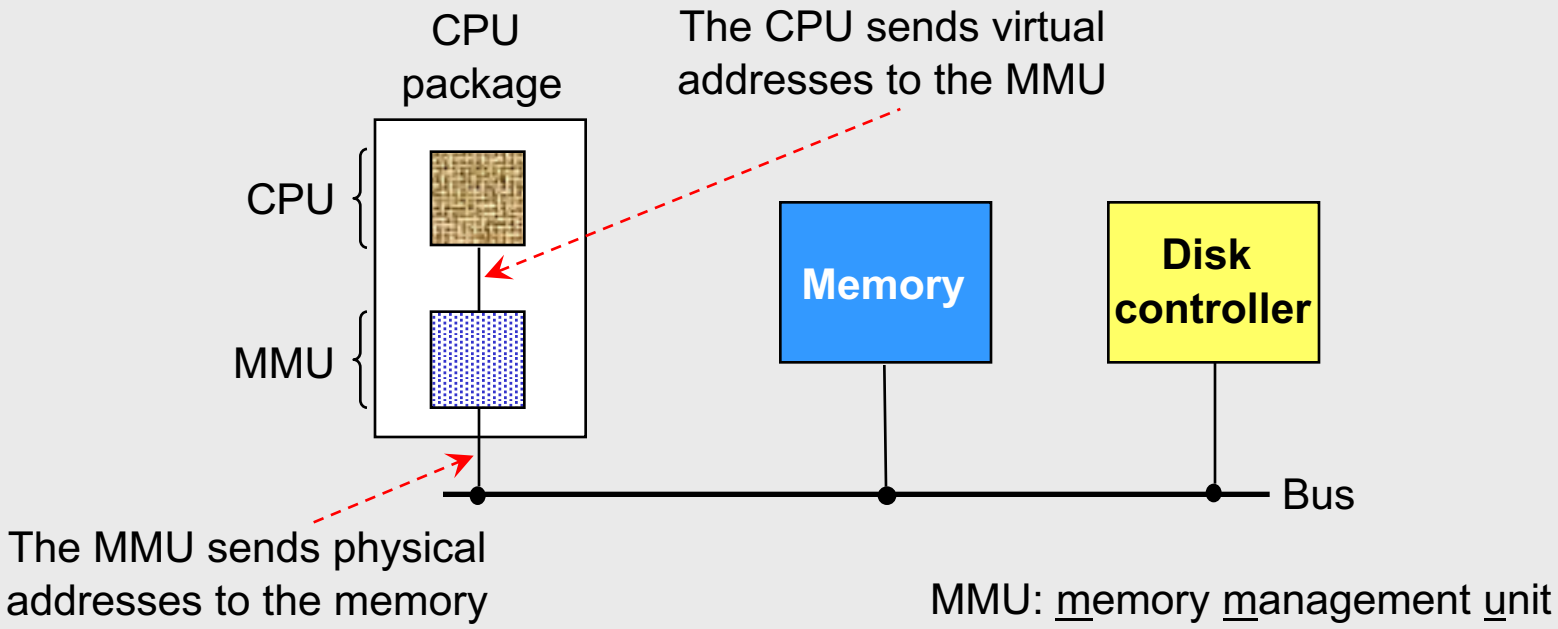
Thay Thế Trang

Nhìn lại paging và segmentation

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



- 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 (1/3)

- Nhận xét: không cần thiết phải có tất cả các page/segment của process trong bộ nhớ chính cùng lúc
  - Ví dụ
    - Đoạn mã xử lý các lỗi hiếm khi xảy ra
    - Các array, list, table 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
      - Vd 'Tools' của MS Word
  - Ngay cả khi toàn bộ chương trình đều cần dùng thì có thể không cần dùng toàn bộ cùng một lúc

Bộ nhớ ảo (2/3)

- Nhìn lại kỹ thuật overlay
- Bộ nhớ ảo (virtual memory)
  - Kỹ thuật 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
    - Phần còn lại được giữ trên bộ nhớ phụ (đĩa)

- 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 kích thước bộ nhớ thực

Bộ nhớ ảo (3/3)

- 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 về 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
- Hai cách hiện thực bộ nhớ ão cho hệ thống paging
  - Demand paging: nạp trang vào bộ nhớ khi được yêu cầu
  - Prepaging: nạp trước vào bộ nhớ cùng một lúc tất cả các trang sẽ được cần đến
- Trong chương này,
  - Quan tâm đến demand paging
  - Phần cứng hỗ trợ hiện thực bộ nhớ ảo
  - Các giải thuật liên quan

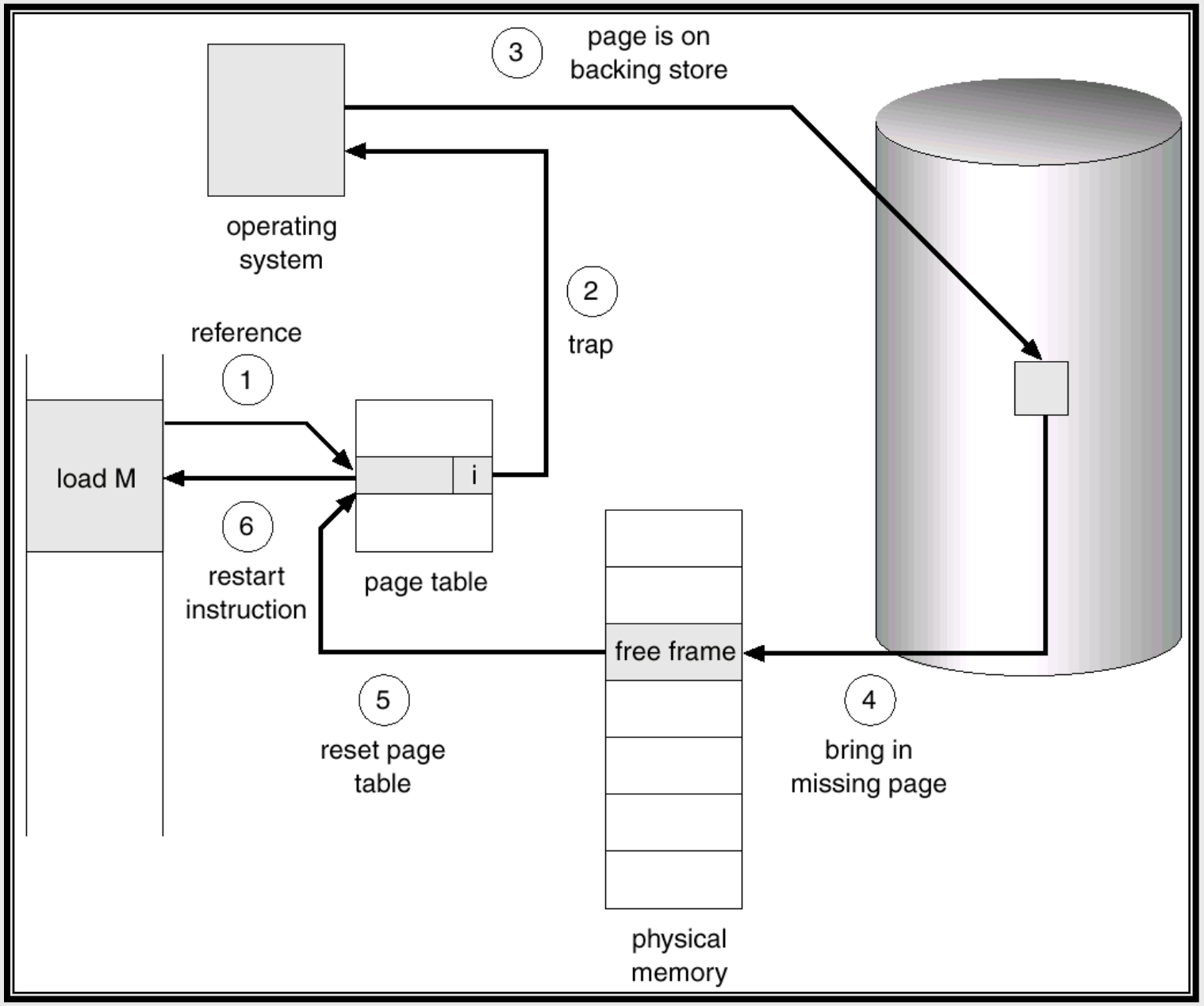
Phần cứng hỗ trợ bộ nhớ ảo

- Phần cứng hỗ trợ phân trang đã được khảo sát trong chương trước
  - Nhắc lại, mỗi mục của bảng phân trang có một valid-invalid bit
- Ở đây valid-invalid bit được sử dụng như sau. Nếu bit có trị
  - "valid" $\Rightarrow$ trang hợp lệ và hiện trong bộ nhớ chính
  - "invalid" $\Rightarrow$ trang không hợp lệ hoặc hợp lệ nhưng không trong bộ nhớ chính
    - Khi có một tham chiếu đến một trang mà không có trong bộ nhớ chính ("invalid") thì phần cứng (MMU) sẽ gây ra page-fault trap

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 (valid-invalid bit = "invalid") 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. Gửi một yêu cầu đọc đĩa để nạp trang được tham chiếu vào một frame trống;
         Trong khi I/O tiến hành, một process khác được cấp CPU để thực thi
      3. Khi I/O hoàn tất, disk controller gây ra một ngắt đến hệ điều hành;
         Cập nhật page table và chuyển process về trạng thái ready

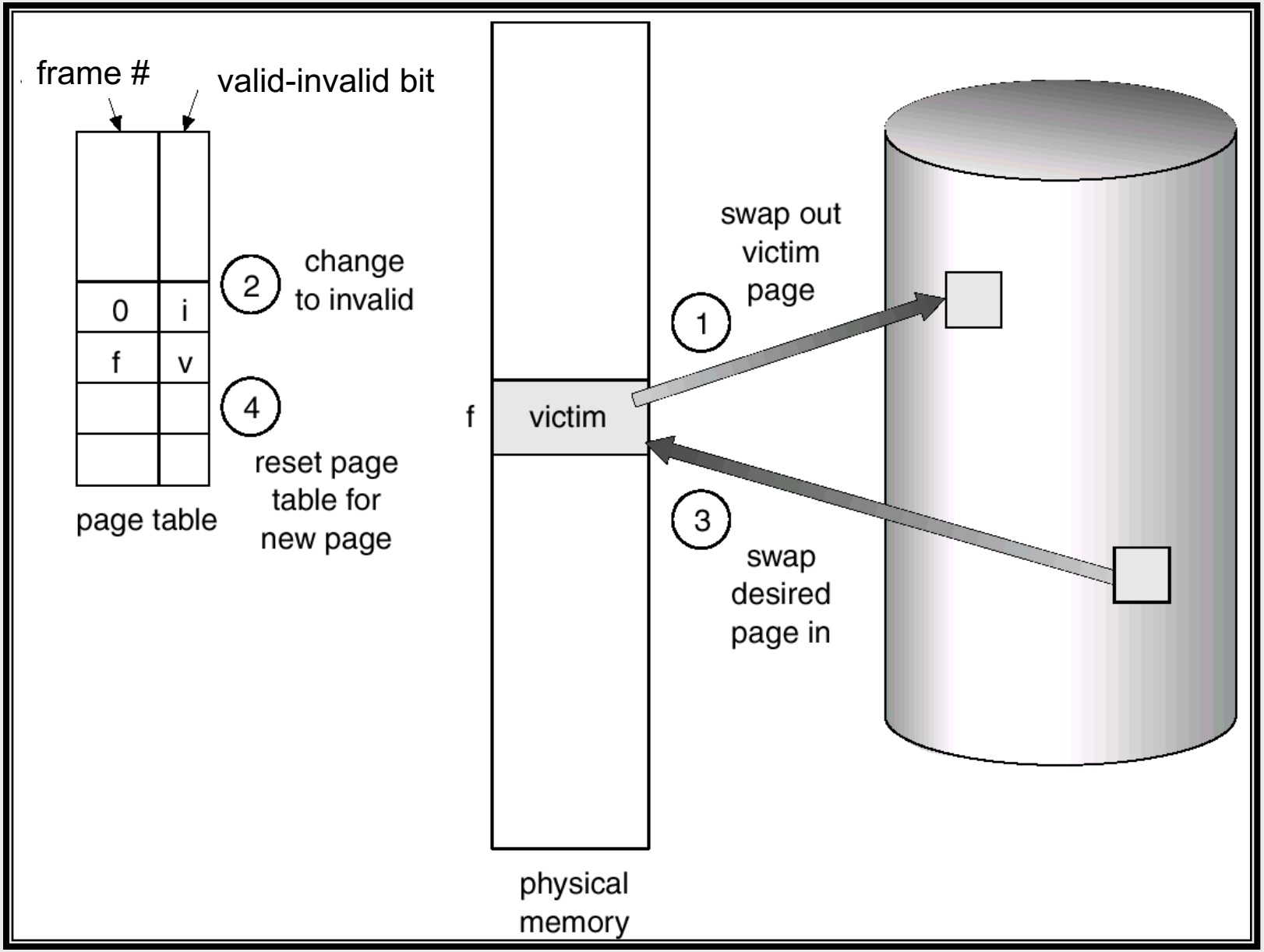
Page fault và các bước xử lý



Thay thế trang nhớ (1/2)

- Bước 2 của PFSR giả sử tìm được frame trống. Nếu không tìm được frame trống, PFSR được bổ sung để thay trang 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 bị thay thế (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)
  4. Cập nhật page table và frame table tương ứng

Thay thế trang nhớ (2/2)



Hiện thực demand paging

Hai vấn đề:

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

- Ví dụ

  Chuỗi tham chiếu các địa chỉ nhớ (hệ thống thập phân), với page size = 100:

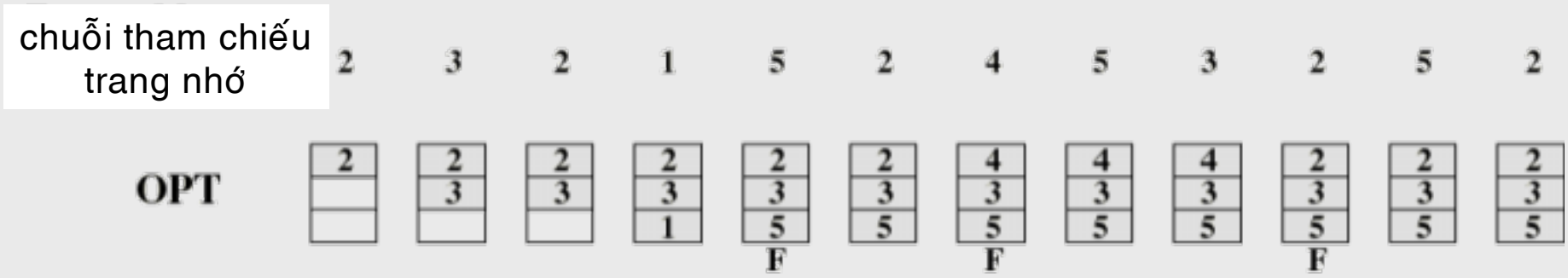
  0100, **04**32, **01**01, **06**12, **01**02, **01**03, **01**04, **01**01, **06**11, **01**02, **01**03, **01**04, **01**01, **06**10, **01**02, **01**03, **01**04, **01**01, **06**09, **01**02, **01**05

⇒ chuỗi tham chiếu trang nhớ

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

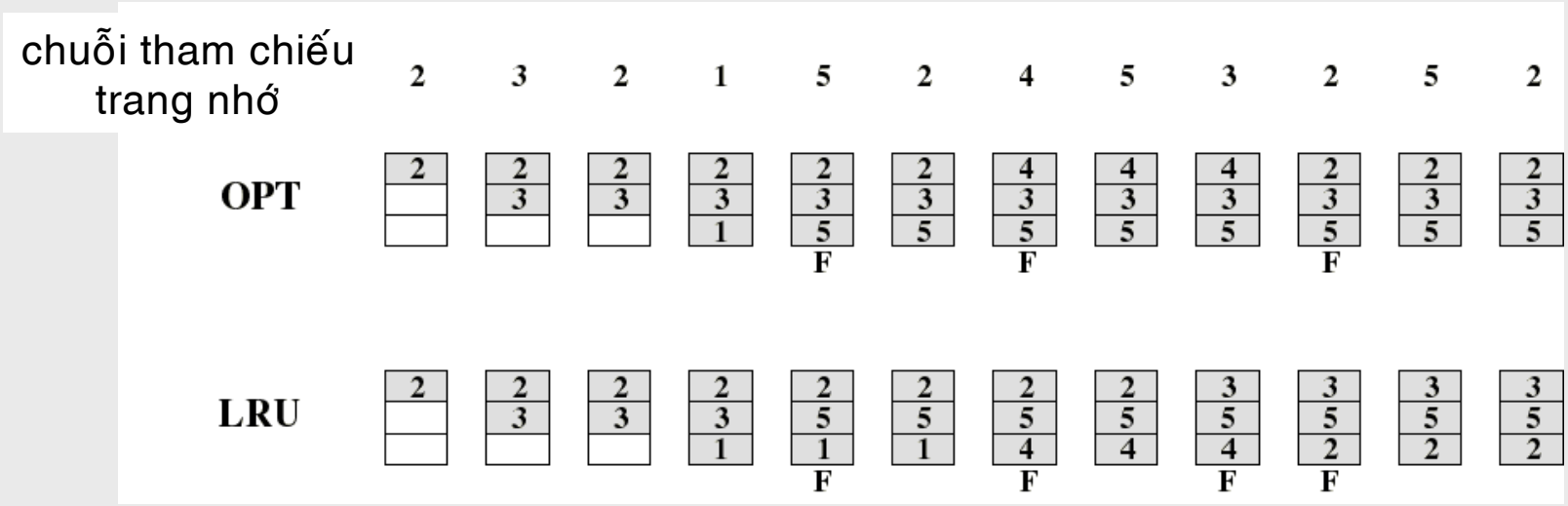
Giải thuật thay trang OPT (OPTimal)

- Trong các trang nhớ, thay thế trang nhớ mà có thời điểm lần tới nó được tham chiếu là xa nhất (hoặc sẽ không được tham chiếu nữa)
  - Ví dụ: một process có 5 trang, và được cấp 3 frame



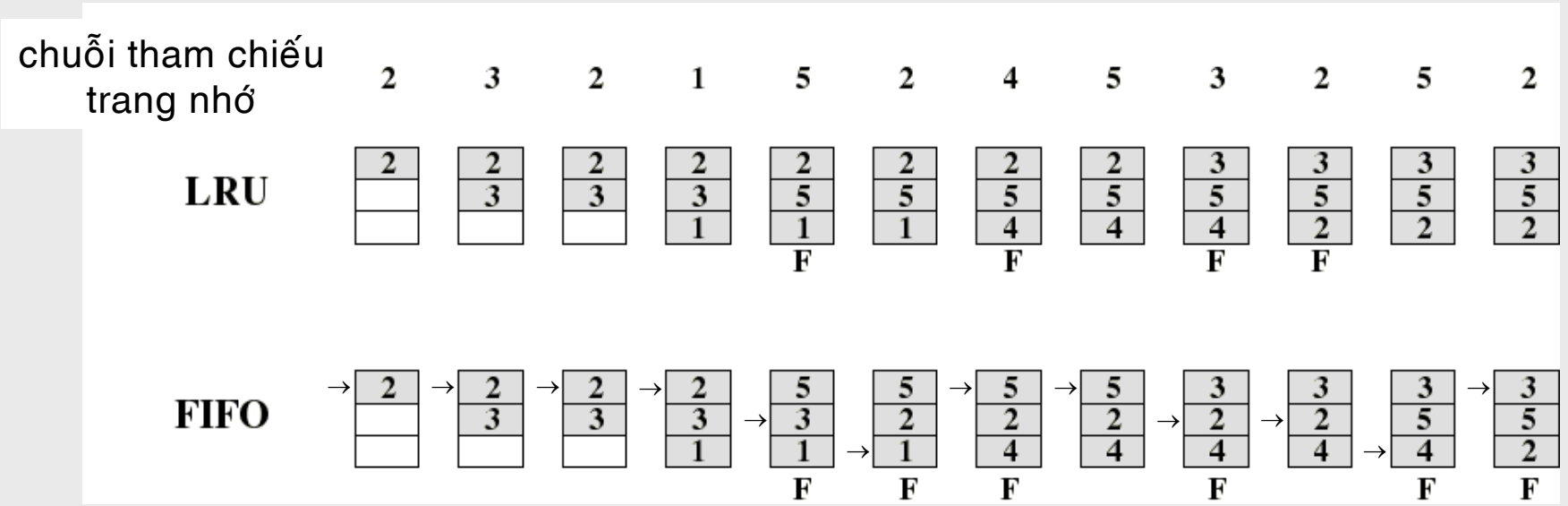
Giải thuật thay trang LRU (Least Recently Used)

- Trong các trang nhớ, thay thế trang nhớ mà có thời điểm lần cuối trong quá khứ nó được tham chiếu là xa nhất
  - Ví dụ: một process có 5 trang, và được cấp 3 frame



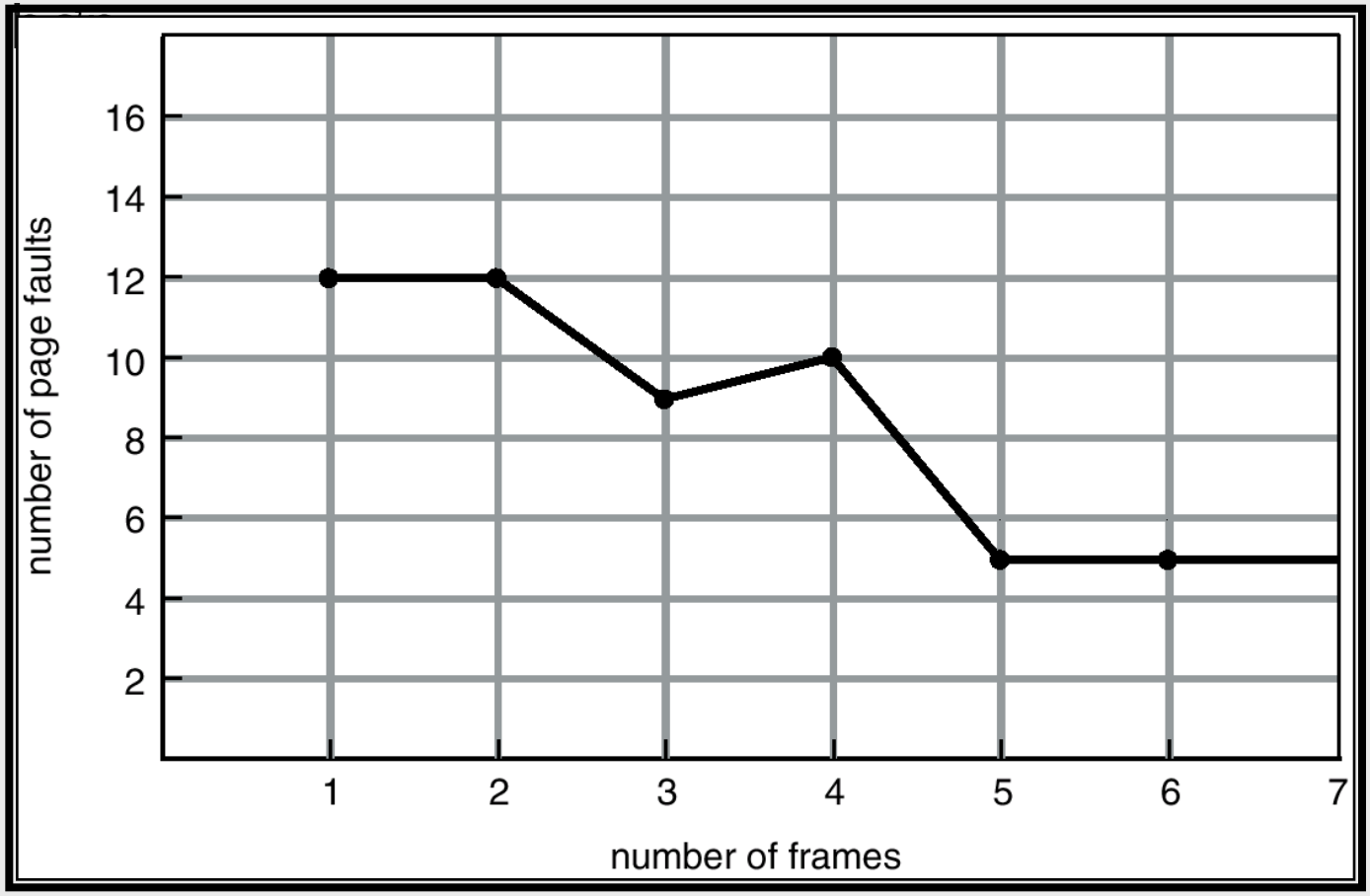
Giải thuật thay trang FIFO

- Thay thế trang nhớ theo thứ tự mà chúng đã được nạp vào (= thay trang đã ở lâu nhất trong bộ nhớ)
  - Hiện thực: Xem các frame được cấp phát cho process như là một bộ đệm xoay vòng (circular buffer)
- Sử dụng trong Windows 2000
- 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



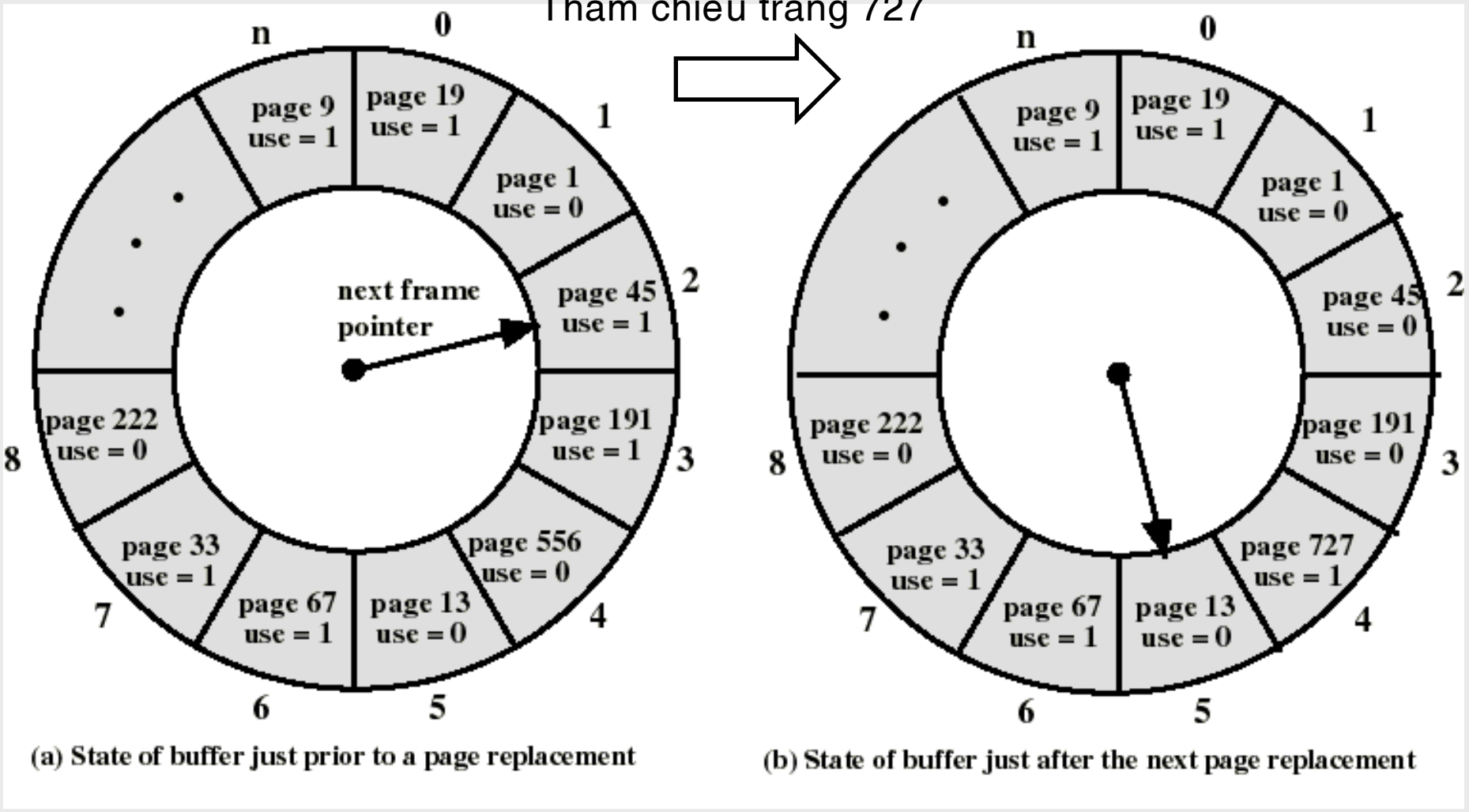
1,2,3,4,1,2,5,1,2,3,4,5

Giải thuật thay trang Clock (1/2)

- Các frame cấp cho process được xem như một bộ đệm xoay vòng
  - Việc tìm trang nhớ thay thế bắt đầu ở frame chỉ bởi con trỏ
- 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 (2/2)

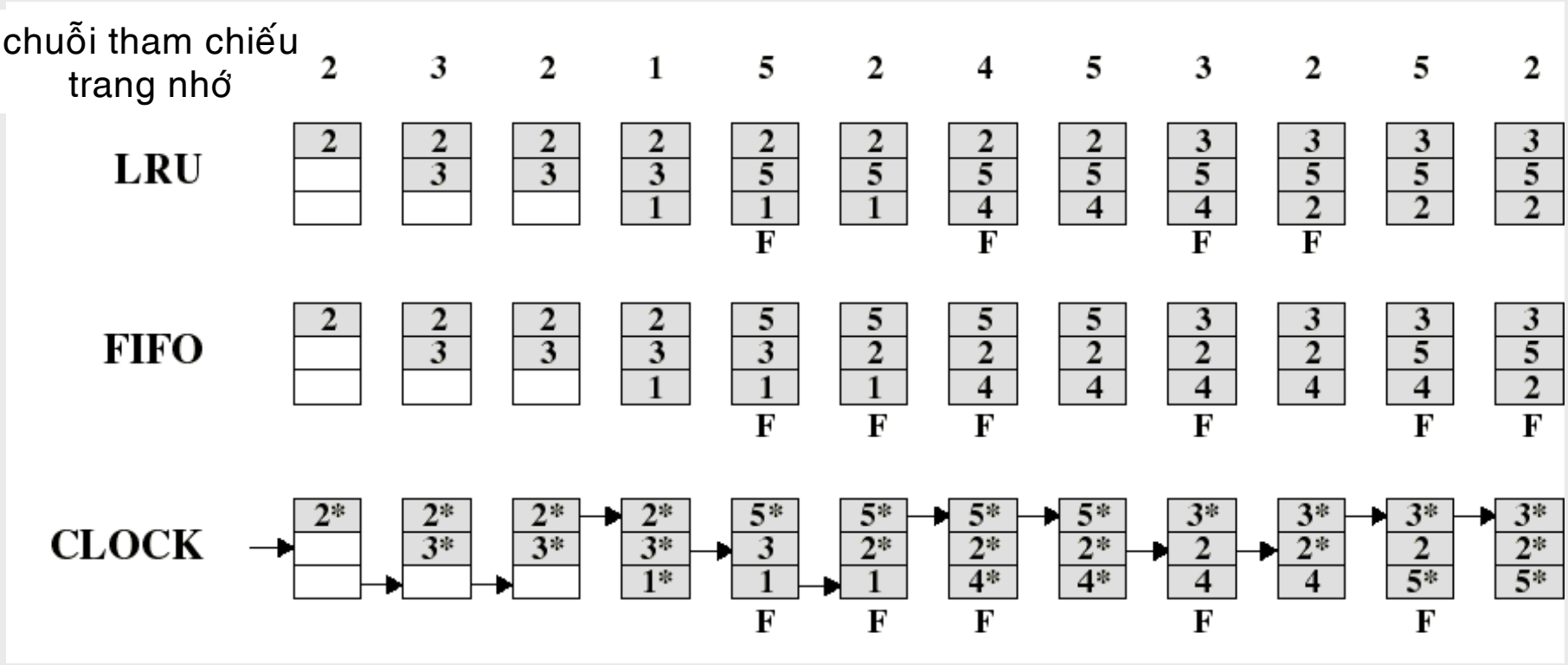
Tham chiếu trang 727



(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
- Một số kết quả thực nghiệm cho thấy Clock có hiệu suất gần với LRU

Nhìn lại

- Tách biệt chính sách và cơ chế trong kỹ thuật bộ nhớ ảo
  - Các chính sách thay trang: LRU, FIFO, Clock,…
  - Cơ chế paging: được xây dựng với frame, page, page table, page fault,…

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, độ ưu tiên…)
- Chiến lược cấp phát động (variable allocation)
  - Số frame cấp cho mỗi process có thể thay đổi trong khi nó chạy
  - OS phải tốn chi phí (overhead) để theo dõi hành vi của các process (vd số lượng page fault của process trong mỗi đơn vị thời gian)

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

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

$S = \quad s_i$

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

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

Ví dụ

$m = 64$

$s_1 = 10$

$s_2 = 127$

$$a_1 = \frac{10}{137} \cdot 64 \quad 5$$

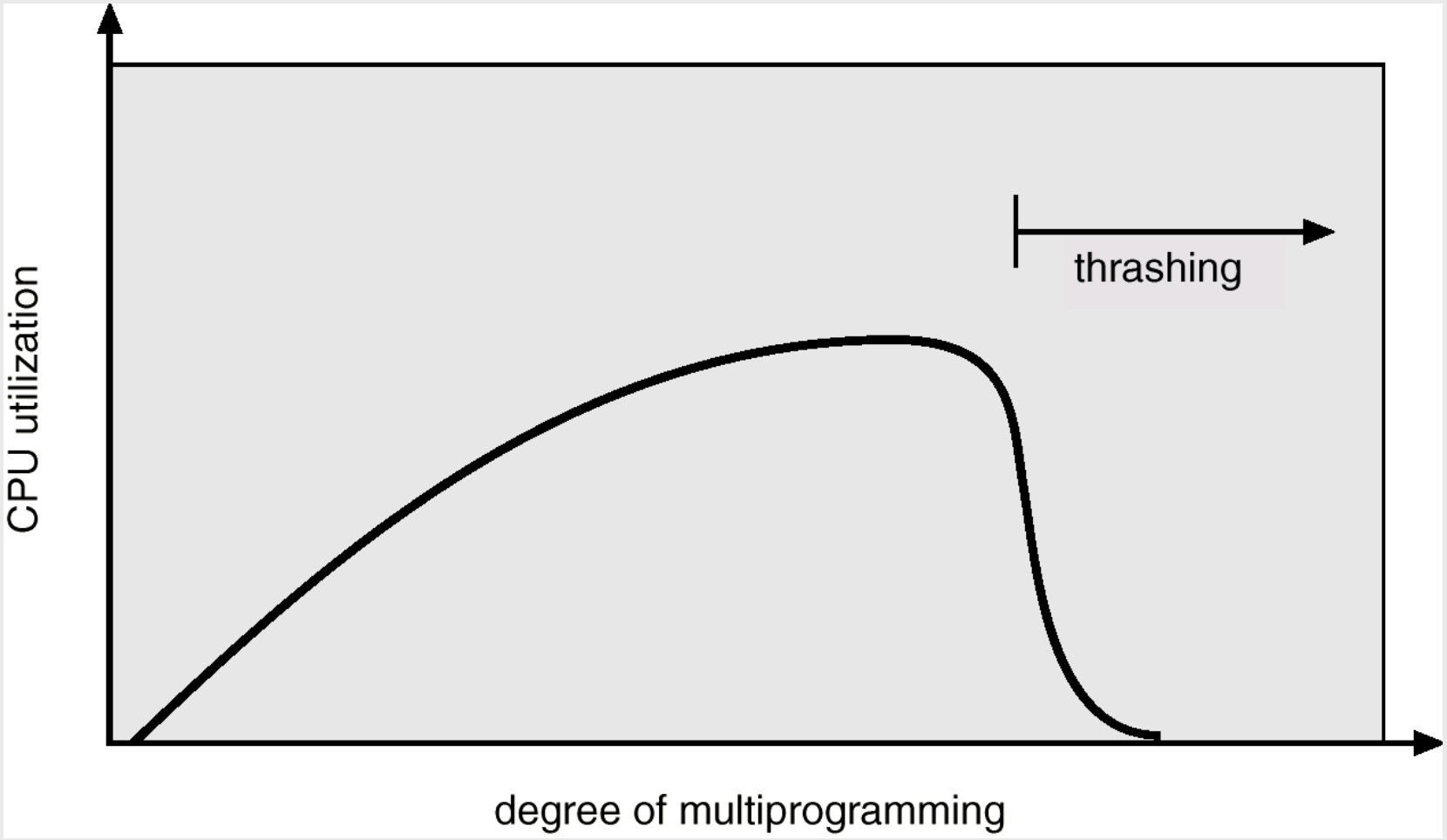
$$a_2 = \frac{127}{137} \cdot 64 \quad 59$$

Thrashing (1/3)

- Khi một process không được cấp đủ số frame cần thiết thì số page faults/sec cao $\Rightarrow$ hiệu suất CPU thấp
  - 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: 0 1 2 3 0 1 2 3 0 1 2 3… gây ra ít nhất N page fault

- Thrashing: hiện tượng các trang nhớ của một process bị hoán chuyển vào/ra liên tục

Thrashing (2/3)



Thrashing (3/3)

- Để hạn chế thrashing, hệ điều hành phải cung cấp cho process “đủ” frame. Bao nhiêu frame thì đủ cho một process thực thi hiệu quả?

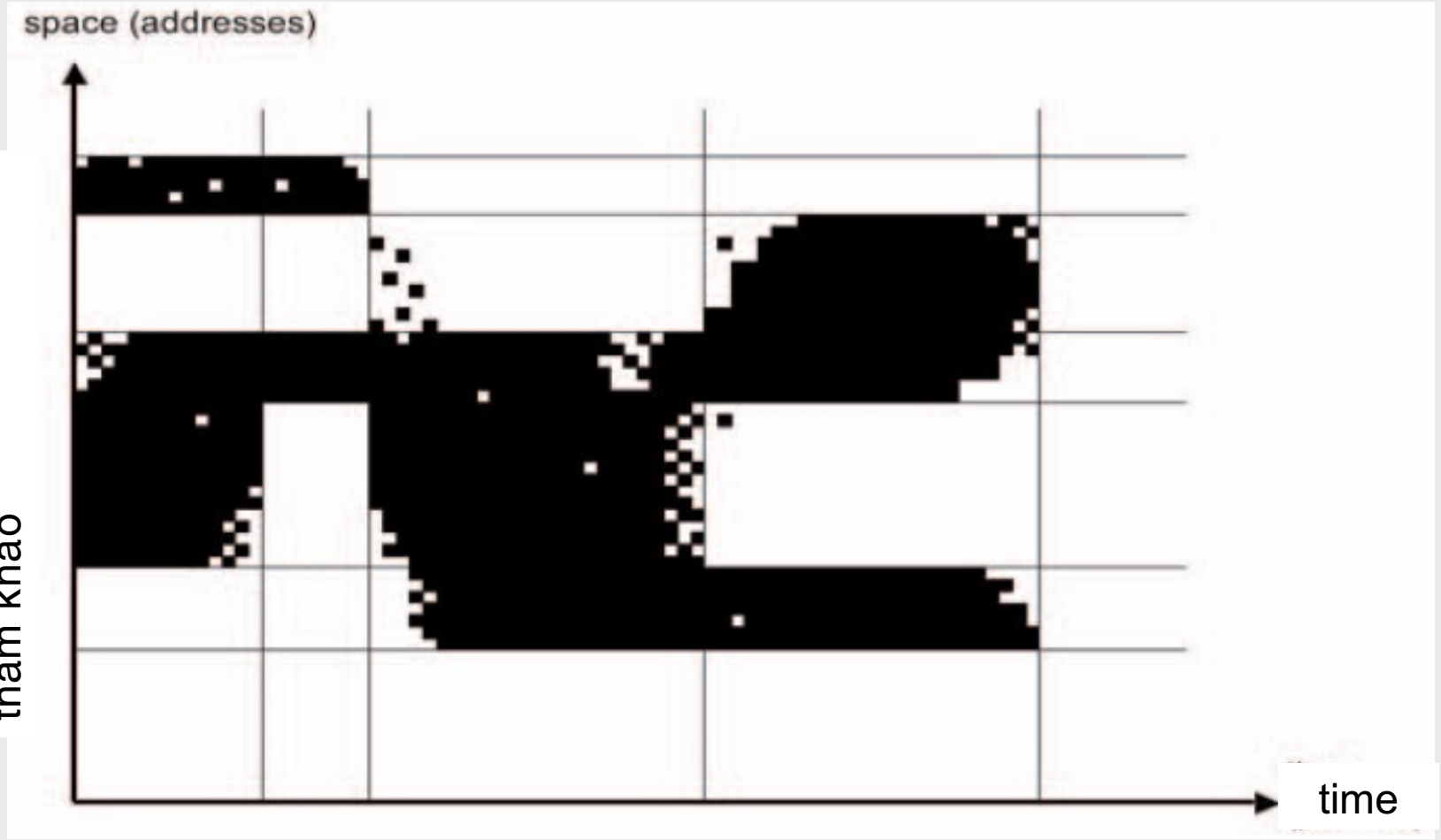
Nguyên lý locality (1/3)

- Nguyên lý locality (locality principle)
  - *Spatial* locality: if a given address is accessed, *nearby* addresses will also be accessed
  - *Temporal* locality: if a given address was referenced, it will be referenced again *soon*

Nguyên lý locality (2/3)

- Process gồm nhiều locality, và trong khi 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)

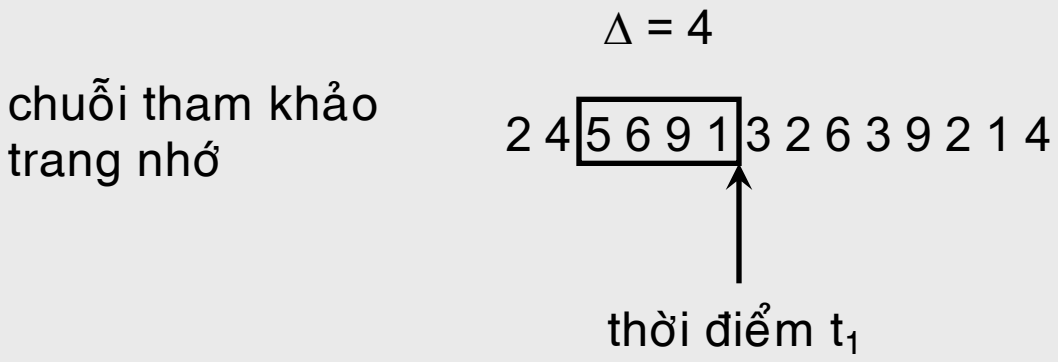
Nguyên lý locality (3/3)

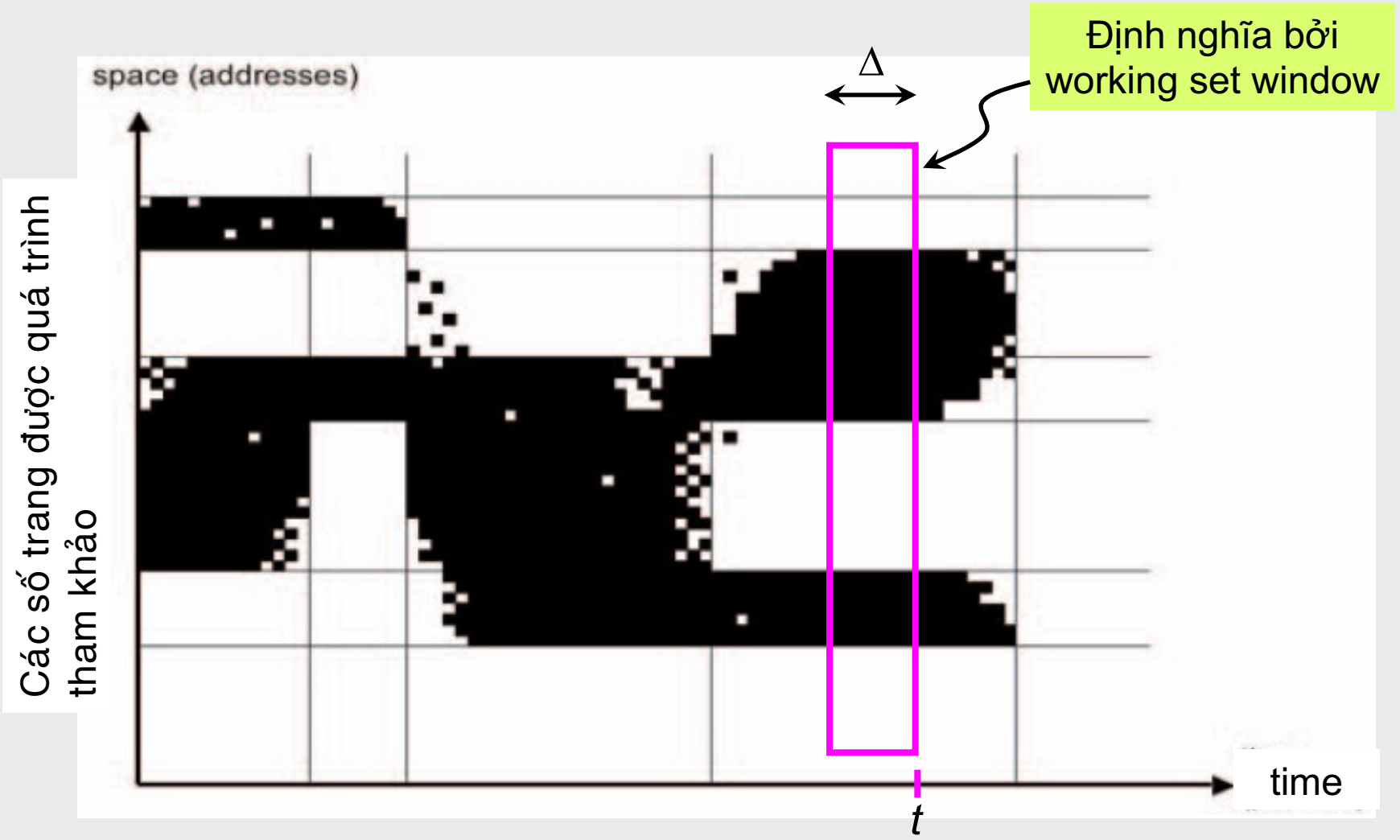


Hình từ “The locality principle”, P.J.Denning

Hạn chế thrashing: Giải pháp working set (1/4)

- Giải pháp working set, còn 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



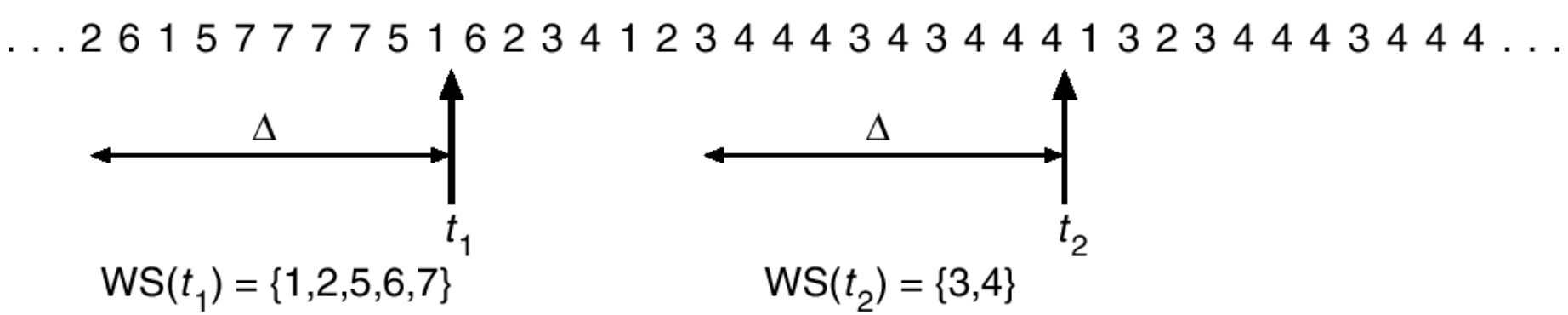


Hình từ “The locality principle”, P.J.Denning

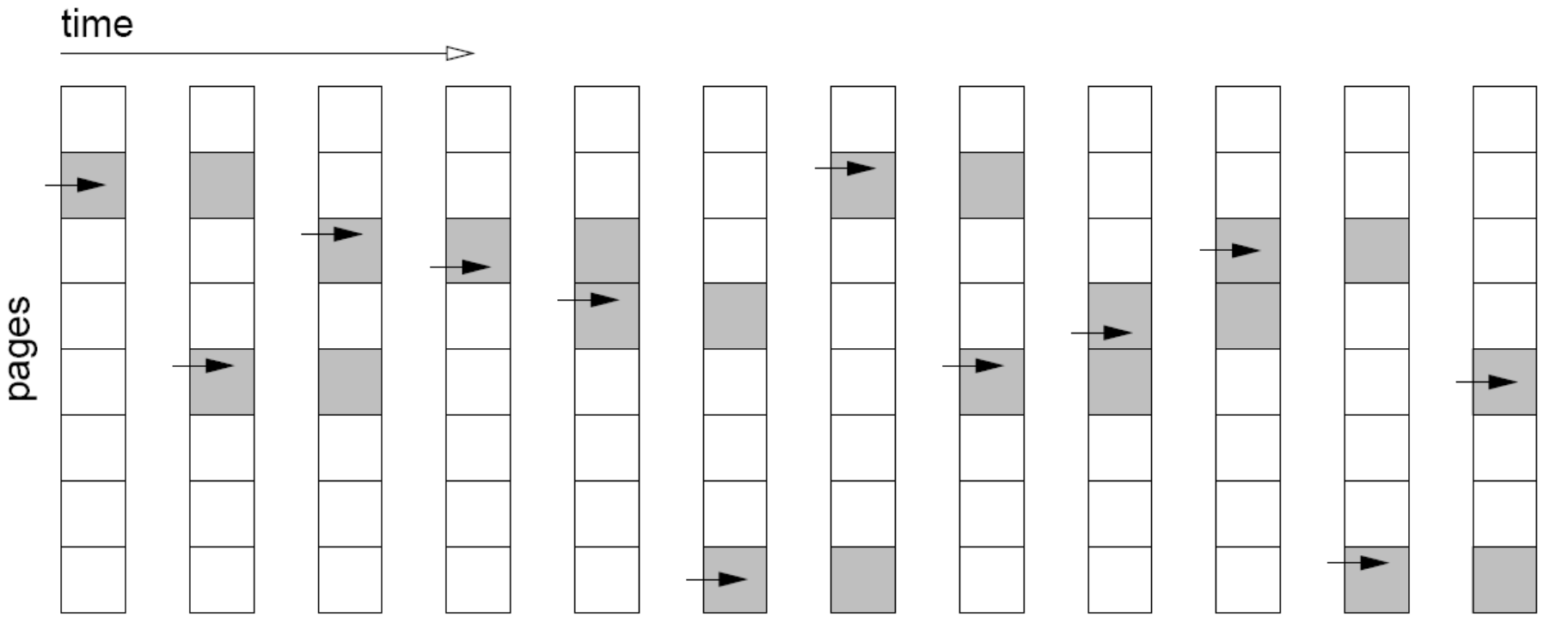
Hạn chế thrashing: Giải pháp working set (2/4)

- Định nghĩa: working set của process P_i , ký hiệu WS_i , là tập các số trang trong working set window. (Thời điểm t là một tham số của định nghĩa.)

Ví dụ: Δ = 10 và chuỗi tham khảo trang



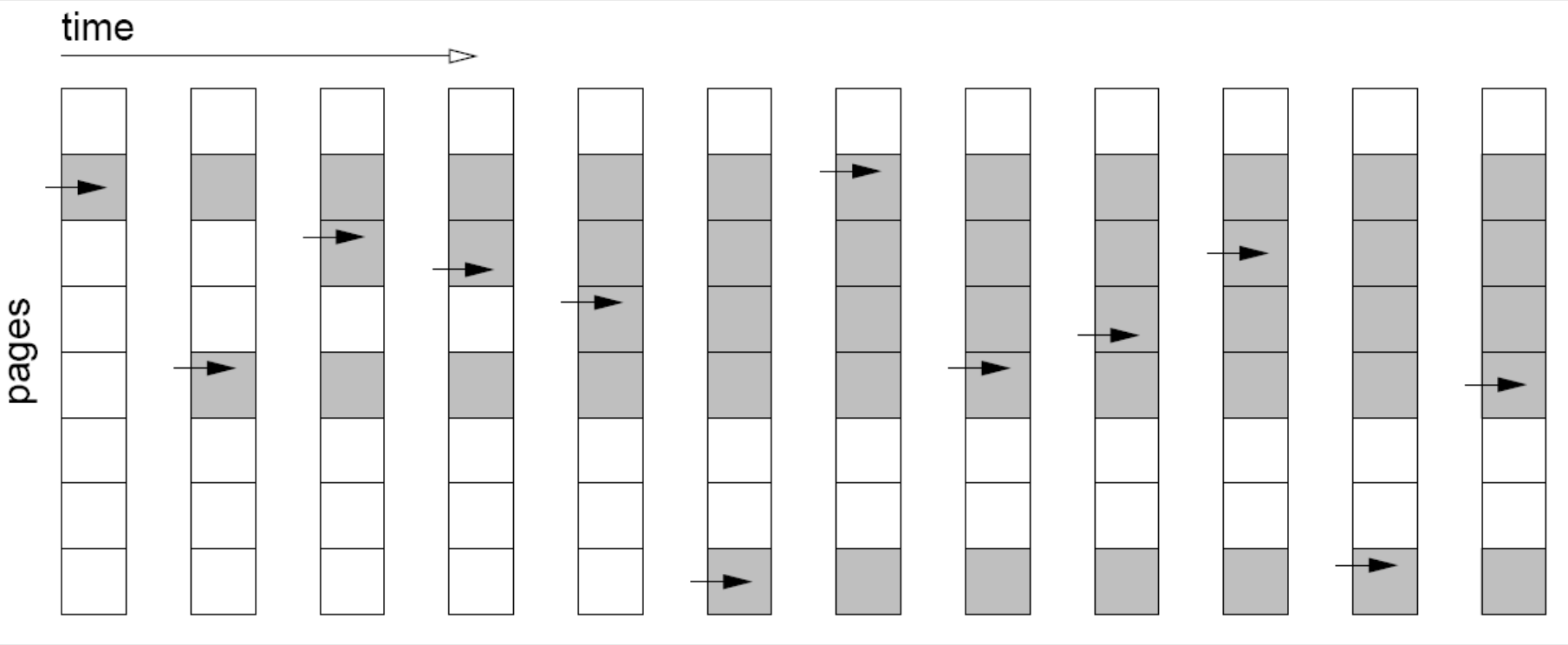
- Nhận xét:
  - Δ quá nhỏ $\Rightarrow$ không đủ bao phủ toàn bộ locality
  - Δ quá lớn $\Rightarrow$ bao phủ nhiều locality khác nhau
  - $\Delta = \infty$ $\Rightarrow$ bao gồm tất cả các trang được sử dụng



Chuỗi tham khảo trang; "→" : trang được tham khảo

- Nếu Δ quá nhỏ thì working set thay đổi liên tục
  - Ví dụ: Δ = 2, working set gồm các trang màu xám

Fig from Feitelson



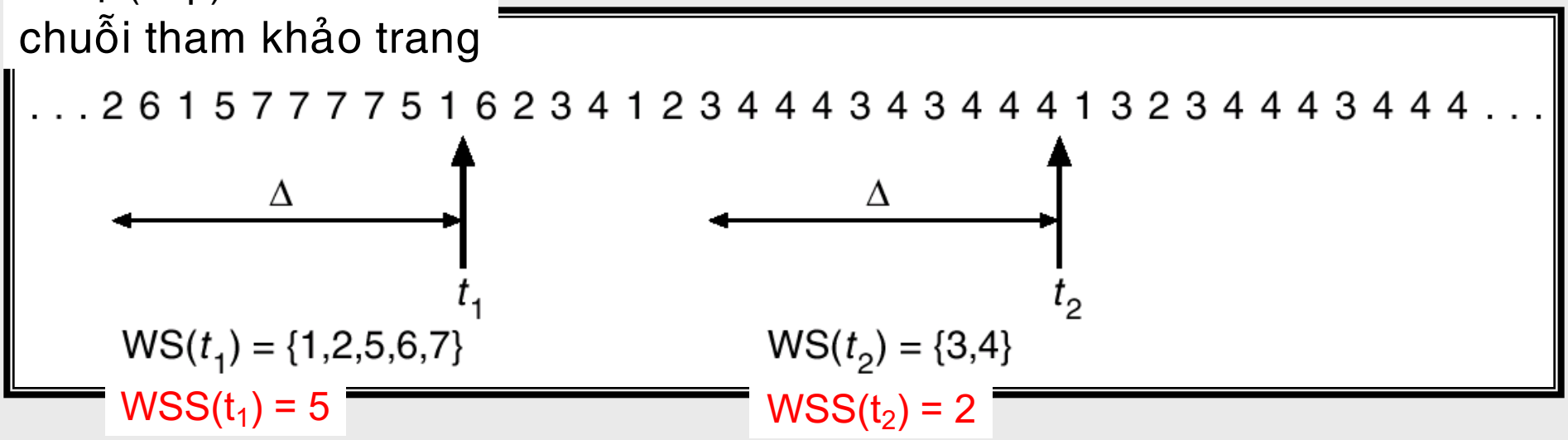
- Nếu Δ thích hợp thì working set không (hay ít) thay đổi
  - Ví dụ: cùng chuỗi tham khảo trang, nhưng $\Delta = 6$

Fig from Feitelson

Hạn chế thrashing: Giải pháp working set (3/4)

- 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): $\Delta = 10$ và chuỗi tham khảo trang



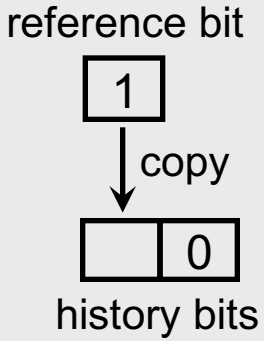
Hạn chế thrashing: Giải pháp working set (4/4)

- Đặt D = $\sum$ WSS_i = tổng các working-set size của mọi process trong hệ thống
  - Nhận xét: Nếu D > m, m là số frame của hệ thống, thì sẽ xảy ra thrashing

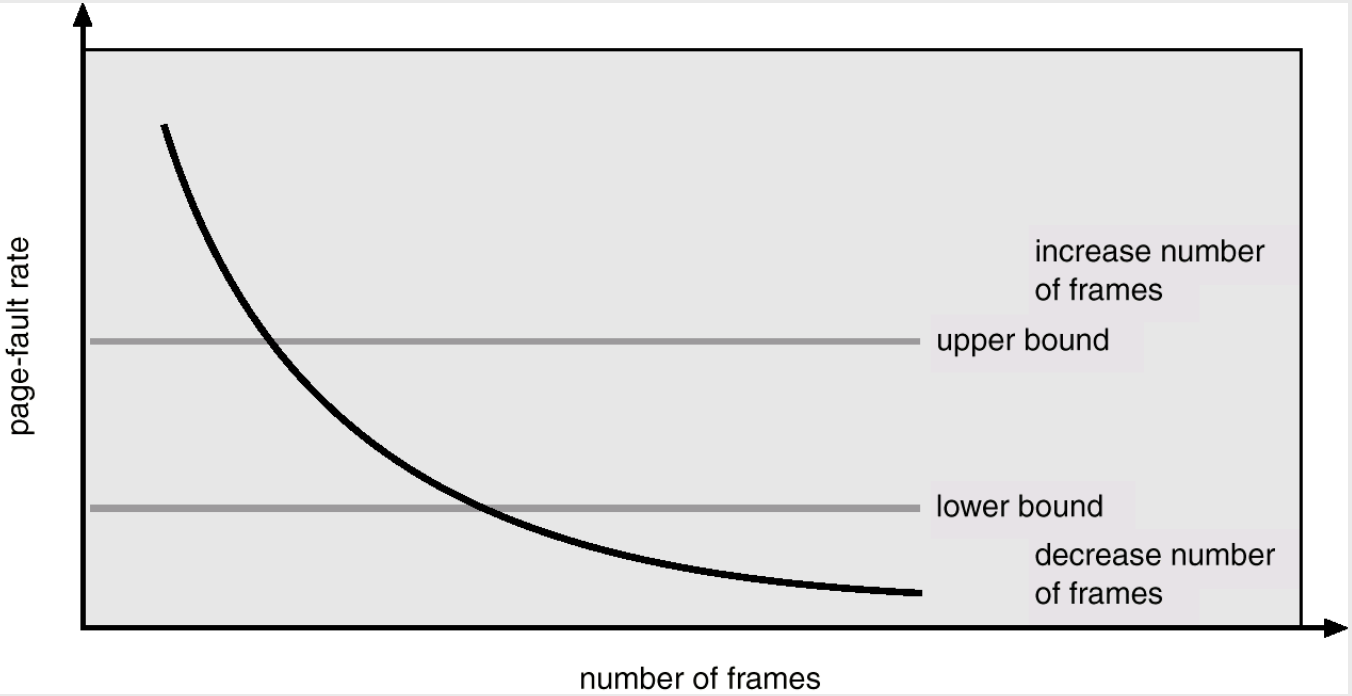
- Giải pháp working set
  - Khi khởi tạo một quá trình: cung cấp số lượng frame thỏa mản working-set size của nó
  - Nếu D > m thì suspend một trong các process
    - Các trang của process được chuyển ra đĩa cứng và các frame của nó được thu hồi

Xấp xỉ working set

- 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 và 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 10.000/5.000 = 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 các history bit 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