Bài 8
Virtual Memory (Bộ nhớ ảo)

Tại sao cần có bộ nhớ ảo?

- Chương trình muốn thực thi thì cần nạp vào bộ nhớ chính, nhưng việc nạp cả chương trình vào bộ nhớ chính không khả thi vì:
  - Giới hạn về kích thước của bộ nhớ chính so với nhu cầu của tất cả chương trình.
  - Không phải phần nào của chương trình cũng cần nạp vào bộ nhớ chính cùng thời điểm.
  - Chương trình đôi khi yêu cầu bộ nhớ lớn hơn nhu cầu thực sự của nó.
- Nếu chỉ nạp một phần chương trình vào bộ nhớ:
  - Chương trình sẽ không bị giới hạn bởi dung lượng bộ nhớ vật lý của thiết bị.
  - Có thể nạp nhiều chương trình vào bộ nhớ cùng lúc → tăng mức độ đa chương và hiệu quả sử dụng CPU.
  - Tốn ít thời gian để nạp các phần của chương trình → tăng tốc độ chương trình.

Bộ nhớ ảo là gì?

- Bộ nhớ ảo (virtual memory) là một kỹ thuật quản lý bộ nhớ, trong đ bộ nhớ phụ có thể được sử dụn như thể nó là một phần của b nhớ chính.
- Nói cách khác, bộ nhớ ảo là một k thuật cho phép một không gian đị chỉ logic lớn có thể ánh xạ vào mộ bộ nhớ vật lý nhỏ hơn.

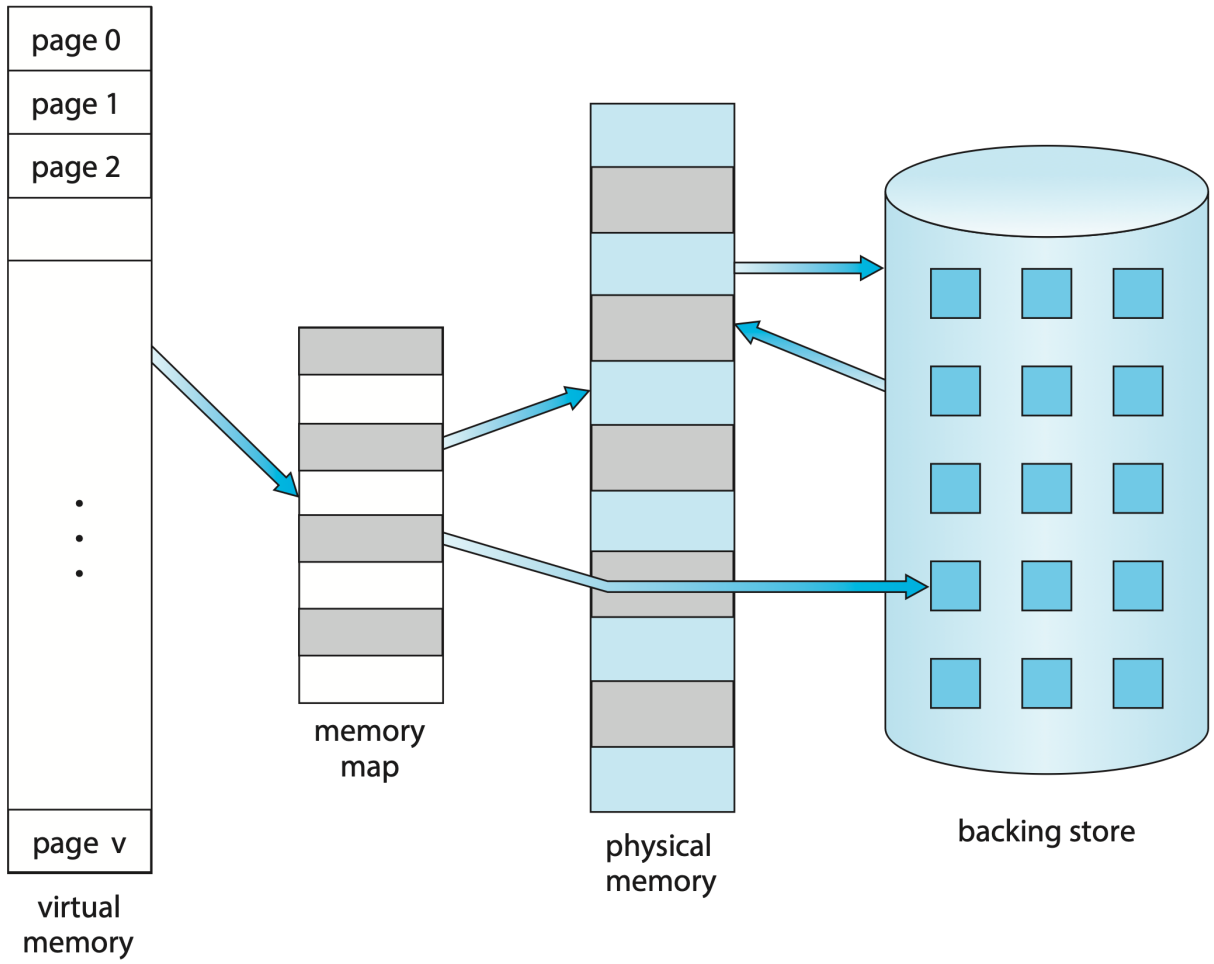


Figure 10.1 Diagram showing virtual memory that is larger than physical memory.

Ưu điểm của bộ nhớ ảo

- Chỉ một phần chương trình cần được nạp vào bộ nhớ để thực thi.
- Không gian địa chỉ logic sẽ lớn hơn nhiều so với không gian địa chỉ vật lý → người lập trình không cần lo về dung lượng bộ nhớ mà chương trình cần.
- Cho phép không gian địa chỉ được chia sẻ giữa các tiến trình.
- Giúp quá trình tạo tiến trình hiệu quả hơn.
- Nhiều chương trình hoạt động đồng thời → tăng tính đa chương
- Giảm thời gian I/O cần để nạp hoặc chuyển giữa các tiến trình.

Ưu điểm của bộ nhớ ảo

- Chia sẻ dữ liệu giữa các tiến trình nhờ bộ nhớ ảo

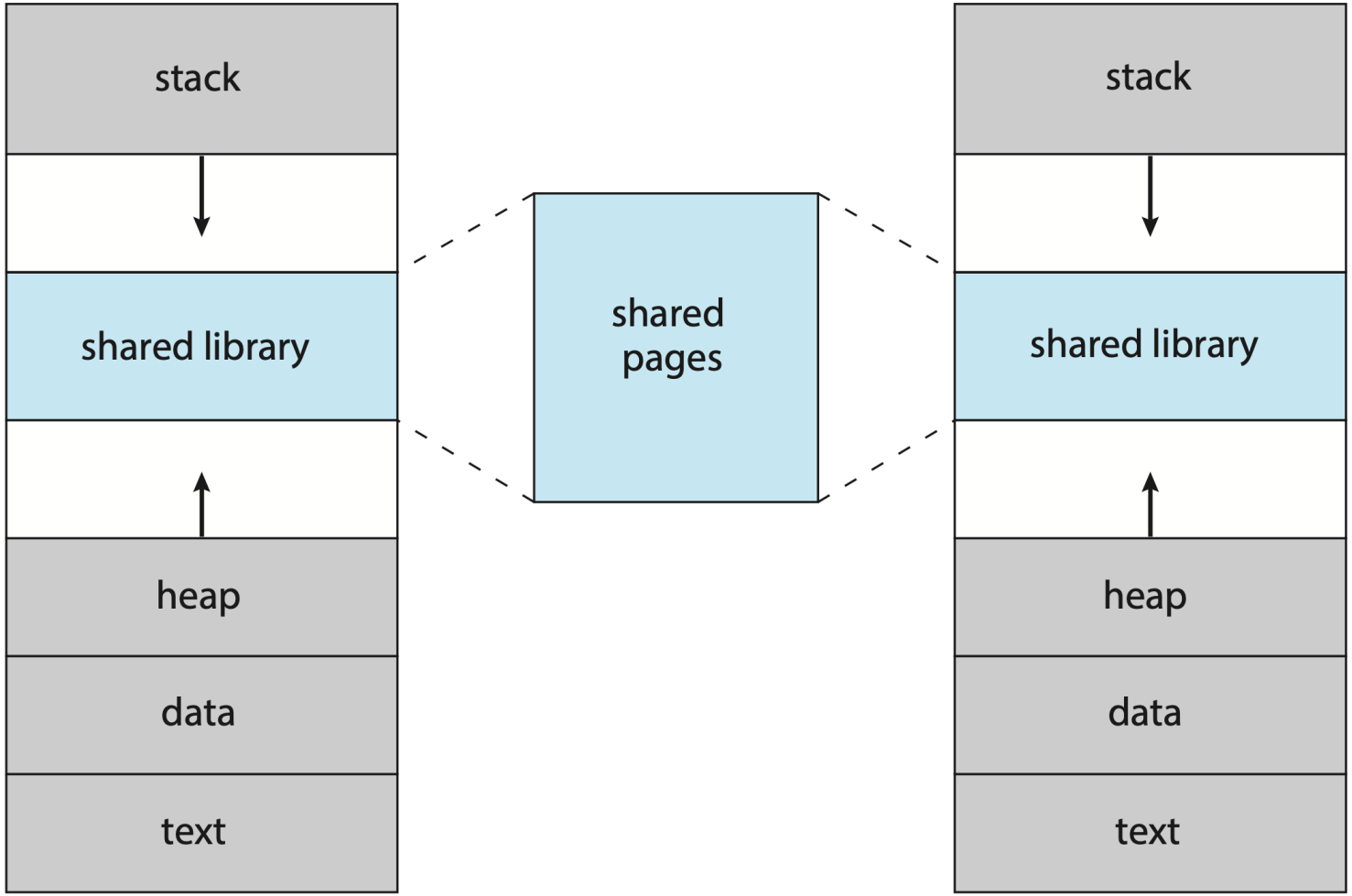


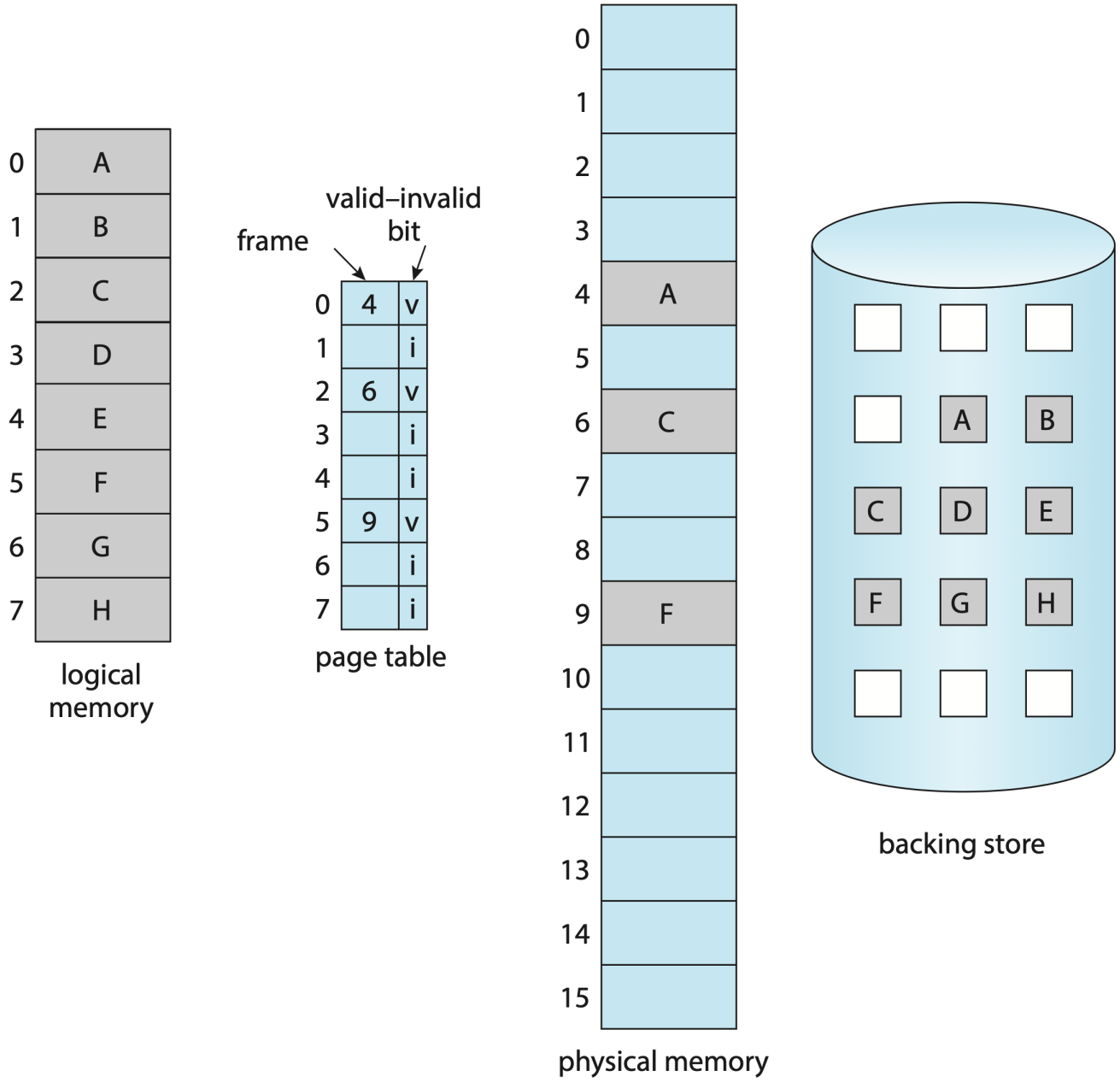
Figure 10.3 Shared library using virtual memory.

Kỹ thuật cài đặt bộ nhớ ảo

- Bộ nhớ ảo có thể hiện thực bằng các phương pháp:
  - Demand paging (phân trang theo yêu cầu): phổ biến vì tính đơn giản, hiệu suất tốt và tính linh hoạt khi quản lý bộ nhớ.
  - Demand segmentation (phân đoạn theo yêu cầu): không phổ biến bởi
    - ✓ Phức tạp vì chia chương trình thành các đoạn (segment) có kích thước khác nhau.
    - ✓ Hiệu suất bị ảnh hưởng vì phải nạp cả đoạn đôi khi chứa dữ liệu không cần thiết.
    - ✓ Ít linh hoạt do gặp khó khăn khi thay đổi thích thước các đoạn.

Kỹ thuật phân trang theo yêu cầu

- Chỉ nạp các trang (page) vào bộ nhớ khi cần → một tiến trình sẽ có một số trang nằm trên bộ nhớ chính và một số trang nằm trên bộ nhớ phụ.
- Sử dụng bit valid-invalid để phân biệt trang đang nằm ở vị trí nào.
  - valid: trang hợp lệ (đã nằm trên bộ nhớ chính và có thể sử dụng).
  - invalid: trang chưa tải vào bộ nhớ chính và chưa thể sử dụng
- Việc truy cập page invalid sẽ dẫn đến lỗi trang (page fault)



Các bước xử lý khi gặp lỗi trang (page fault)

- 1. Nếu có tham chiếu đến trang (qua page table trong PCB) là invalid → page fault
- 2. OS xác định nếu invalid chưa mang page vào trong bộ nhớ thì sang bước 3, ngược lại kết thúc tiến trình.
- 3. Xác định vị trí của page trên bộ nhớ phụ.
- 4. Tìm frame trống và mang page từ bộ nhớ phụ → bộ nhớ chính.
- 5. Chuyển bit invalid → valid vì page đã được nạp vào bộ nhớ.
- 6. Tiếp tục thực thi các instruction đã bị ngắt trước đó.

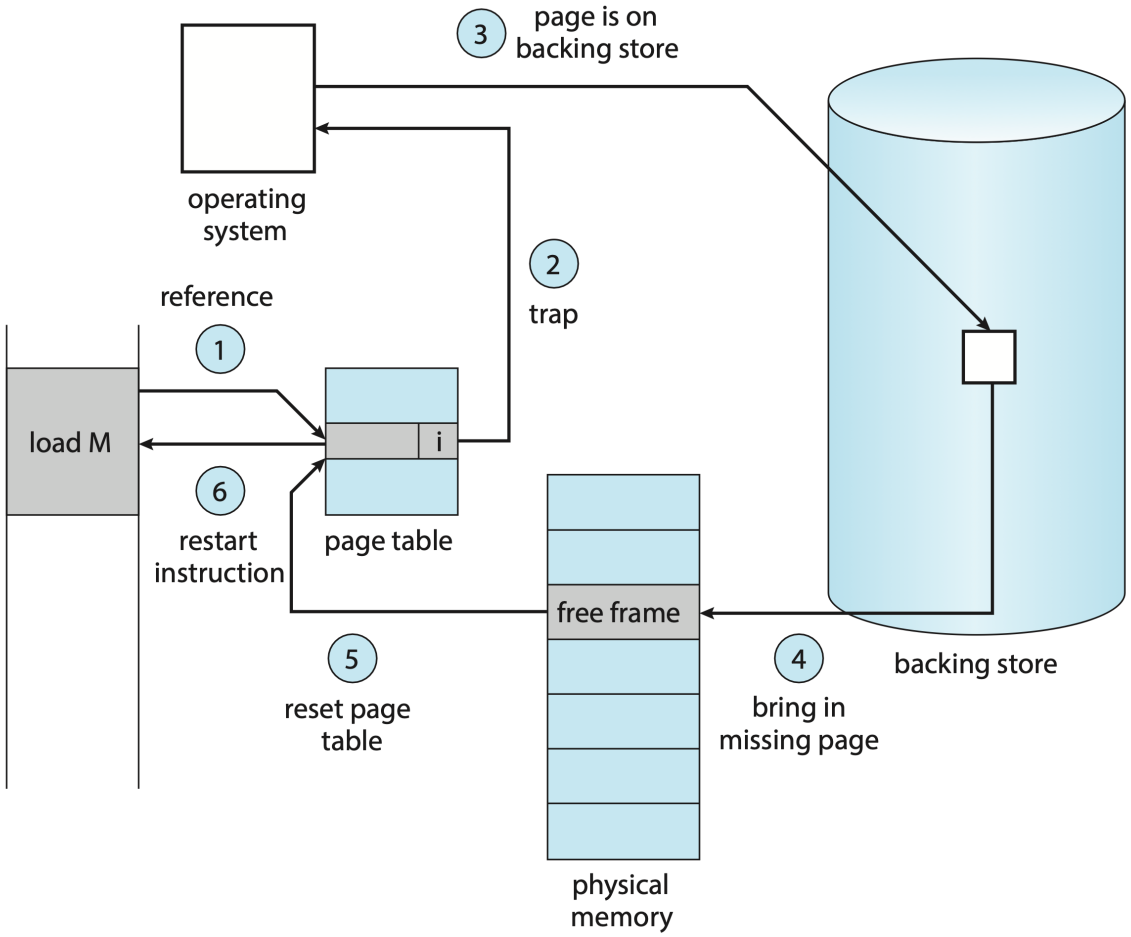


Figure 10.5 Steps in handling a page fault.

Free-frame list

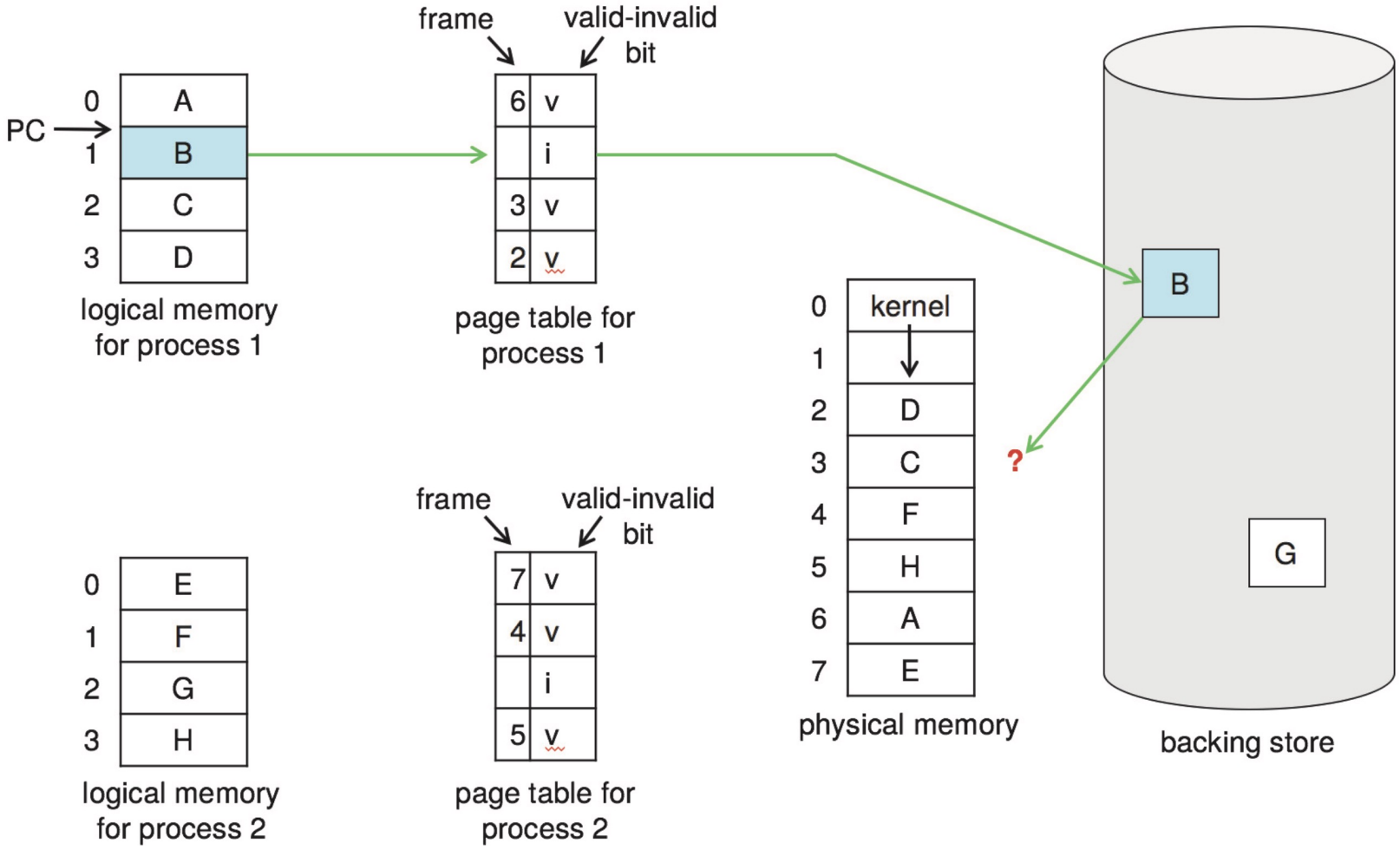
- Khi gặp lỗi trang, OS cần mang page từ bộ nhớ phụ → bộ nhớ chính.
- Page được nạp vào các frame trống trên bộ nhớ chính. Do đó OS cần duy trì một danh sách các frame còn trống, gọi là free-frame list.
- Khi hệ thống khởi động, tất cả bộ nhớ còn trống được đặt vào free-frame list. Trong quá trình hoạt động, kích thước của free-frame list sẽ dần giảm xuống.

Điều gì xảy ra nếu bộ nhớ chính không còn frame trống?

- Giả sử hệ thống có 40 frames. Nếu một process cần 10 pages (nhưng thực sự sử dụng chỉ có 5 pages) thì việc nạp 8 process thay vì 4 process sẽ tăng mức độ đa chương của hệ thống → over-allocating.
- Tuy nhiên hệ thống còn cần bộ nhớ chính cho kernel, I/O buffers… → khó xác định được cụ thể chúng cần bao nhiêu bộ nhớ chính.
- Over-allocating dẫn đến bộ nhớ chính có khả năng không đủ cho các tiến trình (hết frame) → dẫn đến lỗi trang → kỹ thuật thay thế trang (page replacement):
  - Nếu không có frame trống, tìm một frame đang không được dùng và giải phóng nó (chuyển page tương ứng → bộ nhớ phụ).
  - Nạp page cần thiết vào frame trống vừa giải phóng (chuyển page từ bộ nhớ phụ → bộ nhớ chính).

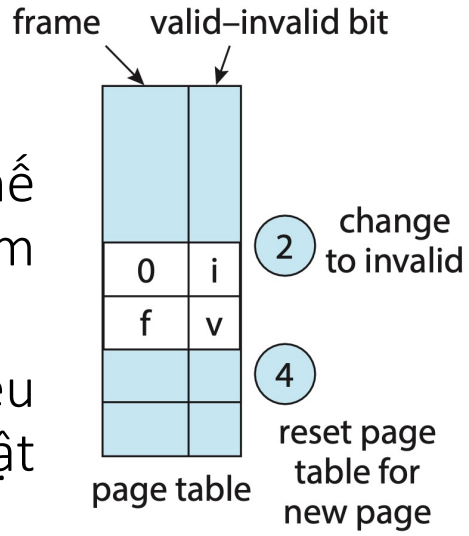
Điều gì xảy ra nếu bộ nhớ chính không còn frame trống?

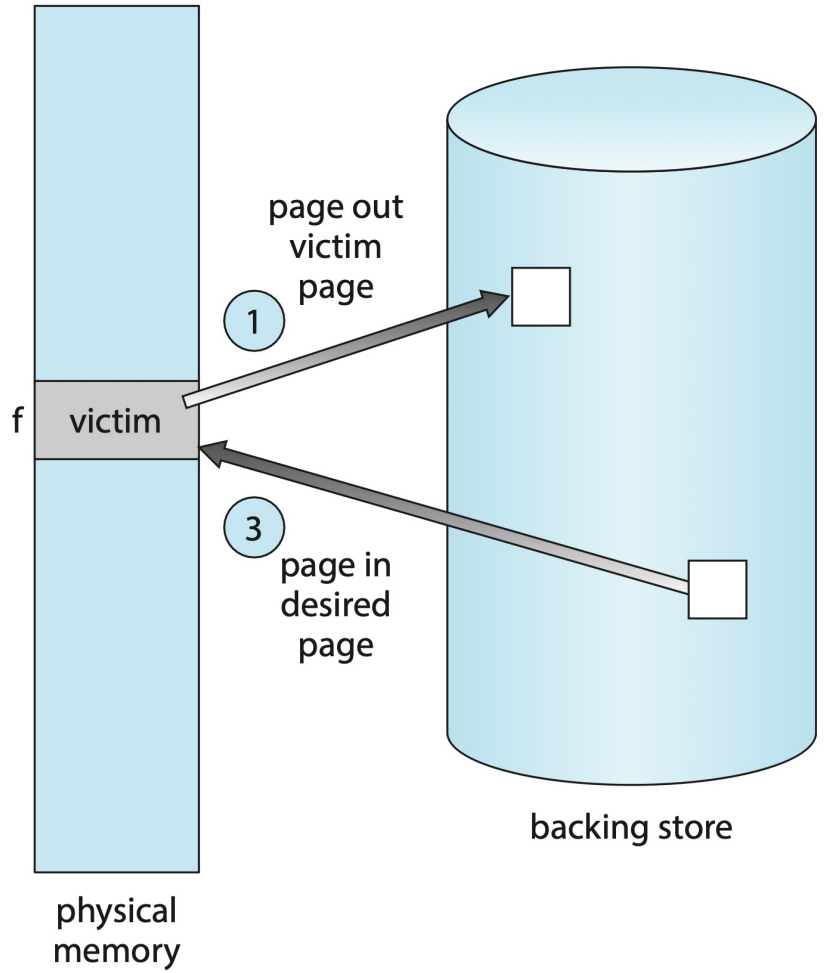
- Minh họa nhu cầu thay thế trang:
  - Tiến trình cần truy cập page chứa dữ liệu B.
  - Page này đang nằm trên bộ nhớ phụ, bộ nhớ chính không còn frame trống.
  - Cần chọn 1 frame trên bộ nhớ chính và giải phóng nó để nạp page chứa dữ liệu B.



Thay thế trang (page replacement)

- 1. Tìm vị trí của trang mong muốn trên bộ nhớ phụ.
- 2. Tìm trang trống:
  - Nếu có thì sử dụng trang trống đó.
  - Không có thì sử dụng giải thuật thay thế trang để tìm 1 frame nạn nhân (“victim frame”).
  - Ghi victim frame vào bộ nhớ phụ (nếu modify/dirty bit được gán), cập nhật bảng trang & bảng khung.
- Đọc trang mong muốn từ bộ nhớ phụ vào frame mới được giải phóng, cập nhật bảng trang & bảng khung.
- Tiếp tục thực thi tiến trình từ vị trí page fault xảy ra.





Page replacement & Frame allocation algorithm

- Hai vấn đề chính của phân trang theo yêu cầu:
  - Frame-allocation algorithm: xác định cần cấp bao nhiêu frames cho mỗi tiến trình; những frame nào cần phải thay thế.
  - Page-replacement algorithm: xác định page cần thay thế sao cho tỉ lệ lỗi trang là thấp nhất.
- Đánh giá hiệu quả của các thuật toán thay thế trang bằng cách thực hiện thuật toán trên một chuỗi truy xuất trang (reference string) và tính số lỗi trang trên chuỗi đó.
  - Chuỗi truy xuất trang chỉ chứa chỉ số trang, không phải là địa chỉ trang.
  - Truy cập lại cùng một trang sẽ không gây lỗi trang.
  - Kết quả thuật toán phụ thuộc vào số frame trống.

Page Replacement Algorithms

- Vấn đề thứ 1 của phân trang theo yêu cầu là các thuật toán thay thế trang:
  - FIFO Algorithm.
  - Optimal Algorithm.
  - Least Recently Used Algorithm.
  - LRU Approximation Algorithm.

Thuật toán thay thế trang FIFO

- Ý tưởng: page nào nạp vào bộ nhớ trước thì sẽ bị loại khỏi bộ nhớ sớm nhất khi có lỗi trang → first in, first out.
- Ưu điểm:
  - Dễ hiểu, dễ cài đặt.
- Vấn đề:
  - Không mang lại hiệu quả nếu các page đầu tiên quan trọng, cần truy xuất thường xuyên.
  - Nghịch lý Belady (Belady's anomaly): tăng số lượng frame lại làm tăng số lượng lỗi trang.

Thuật toán thay thế trang FIFO

- Reference string: 7, 0, 1, 2, 0, 3, 0, 4, 2, 3, 0, 3, 2, 1, 2, 0, 1, 7, 0, 1
  - Số lỗi trang (page fault): 15 → 75%.
  - Số lần truy cập trang thành công: 5/20 → 25%.
  - Lỗi trang càng nhiều → càng phải thay thế frame → giảm hiệu năng.
  - Truy cập trang thành công: hit. Truy cập trang không thành công: miss.

No	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
Ref string	7	0	1	2	0	3	0	4	2	3	0	3	2	1	2	0	1	7	0	1
Frame 1	7	7	7	**2**		2	2	**4**	4	4	**0**			0	0			**7**	7	7
Frame 2		0	0	0		**3**	3	3	**2**	2	2			**1**	1			1	**0**	0
Frame 3			1	1		1	**0**	0	0	**3**	3			3	**2**			2	2	**1**

Thuật toán thay thế trang Optimal

- Ý tưởng: còn gọi là thuật toán tối ưu, chọn page lâu được sử dụng nhất trong tương lai để thay thế.
- Ưu điểm:
  - Thuật toán có tỉ lệ lỗi trang thấp nhất.
  - Không gặp phải nghịch lý Belady
- Vấn đề:
  - Khó cài đặt vì không thể dự đoán các chuỗi truy xuất trang trong tương lai (vấn đề tương tự với SJF).

Thuật toán thay thế trang Optimal

- Reference string: 7, 0, 1, 2, 0, 3, 0, 4, 2, 3, 0, 3, 2, 1, 2, 0, 1, 7, 0, 1
  - Số lỗi trang: 9 → 45%
  - Số lần truy cập trang thành công: 11/20 → 55%

No	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
Ref string	7	0	1	2	0	3	0	4	2	3	0	3	2	1	2	0	1	7	0	1
Frame 1	7	7	7	**2**		2		2			2			2				**7**		
Frame 2		0	0	0		0		**4**			**0**			0				0		
Frame 3			1	1		**3**		3			3			**1**				1		

Thuật toán thay thế trang Least Recently Used (LRU)

- Ý tưởng: chọn page ít được truy cập gần đây nhất.
- Ưu điểm:
  - Là thuật toán cân bằng giữa FIFO và OPT, thường được sử dụng.
- Vấn đề:
  - Tốn tài nguyên để theo dõi thứ tự các trang được truy cập.
  - Cài đặt bằng bộ đếm (counter): mỗi mục trong bảng trang có 1 biến counter ghi nhận thời điểm truy xuất mới nhất. Trang có thời điểm truy xuất nhỏ nhất sẽ bị chọn để thay thế.
  - Cài đặt bằng stack: mỗi khi trang được truy cập, nó được chuyển lên đầu stack. Trang được chọn để thay thế là trang nằm ở đáy stack.

Thuật toán thay thế trang Least Recently Used (LRU)

- Reference string: 7, 0, 1, 2, 0, 3, 0, 4, 2, 3, 0, 3, 2, 1, 2, 0, 1, 7, 0, 1
  - Số lỗi trang: 12/20 → 60%. Tốt hơn FIFO nhưng tệ hơn OPT
  - Truy cập trang thành công: 8/20 → 40%

No	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
Ref string	7	0	1	2	0	3	0	4	2	3	0	3	2	1	2	0	1	7	0	1
Frame 1	7	7	7	**2**		2		**4**	4	4	**0**			**1**		1		1		
Frame 2		0	0	0		0		0	0	**3**	3			3		**0**		0		
Frame 3			1	1		**3**		3	**2**	2	2			2		2		7		

Thuật toán thay thế trang xấp xỉ LRU

- LRU tốn kém tài nguyên, đòi hỏi sự hỗ trợ của phần cứng → Xấp xỉ LRU được phát triển để giảm độ phức tạp của LRU nhưng vẫn giữ lại hiệu quả cao như LRU
- Một số thuật toán (SV cần đọc thêm trong tài liệu tham khảo):
  - Additional-Reference-Bits Algorithm.
  - Second-Chance Algorithm.
  - Enhance Second-Change Algorithm.
  - Counting-Based Page Replacement.
  - Page-Buffering Algorithm.

Frame Allocation Algorithms

- Vấn đề thứ 2 của phân trang theo yêu cầu là các thuật toán cấp phát frame.
- Cấp phát công bằng (equal allocation):
  - Nếu có n frames và m tiến trình → mỗi tiến trình có n/m frames.
  - VD có 93 frames và 5 tiến trình → mỗi tiến trình có 18 frames. 3 frames còn dư được sử dụng để cấp phát cho các tiến trình khác nếu cần.
  - Có thật sự công bằng?
    - ✓ Hệ thống có 62 frames trống, kích thước mỗi frame là 1KB. Có 2 tiến trình.
    - ✓ Tiến trình 1 cần 10KB → được cấp 31 frames.
    - ✓ Tiến trình 2 cần 127KB → được cấp 31 frames.

Frame Allocation Algorithms

- Vấn đề thứ 2 của phân trang theo yêu cầu là các thuật toán cấp phát frame.
- Cấp phát theo tỉ lệ (proportional allocation):
  - Số frame cấp cho mỗi tiến trình tỉ lệ với kích thước của nó.
  - Cụ thể: số frame cho tiến trình i = kích thước bộ nhớ ảo của tiến trình i / tổng kích thước bộ nhớ ảo * số frame trống.
  - Đối chiếu với ví dụ trước:
    - ✓ Tiến trình 1: 10/(10+127)*62 ~ 4 frames.
    - ✓ Tiến trình 2: 127/(10+127)*62 ~ 57 frames.

Sự trì trệ (Thrashing)

- Sự trì trệ (thrashing) là hiện tượng tiến trình thường xuyên phát sinh lỗi trang và phải dùng rất nhiều thời gian sử dụng CPU để thực hiện việc thay thế trang.
- Tốc độ phát sinh lỗi trang tăng rất cao, không công việc nào có thể kết thúc vì tất cả các tiến trình đều bận rộn với việc thay thế trang → trì trệ toàn bộ hệ thống.
- Nguyên nhân do hệ thống không đủ các frame cần thiết để chứa toàn bộ hệ thống.
- Giải pháp: xác định và điều chỉnh mức độ cấp phát khung trang hợp lý cho các tiến trình.

Tham khảo

- Operating System Concepts
- https://www.javatpoint.com/os-virtual-memory
- https://www.geeksforgeeks.org/virtual-memory-in-operating-system/
- https://www.tutorialspoint.com/virtual-memory-in-the-operating-system