Kỹ thuật phân mảnh bộ nhớ

Nội dung

Có 2 kỹ thuật chính:

- Kỹ thuật phân trang (paging)
- Kỹ thuật phân đoạn (segmentation)

- Kỹ thuật kết hợp

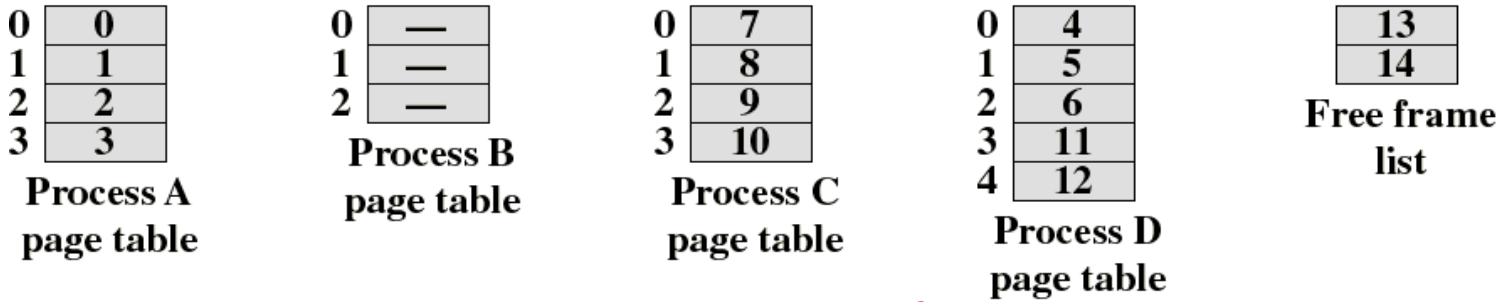
Kỹ thuật phân trang

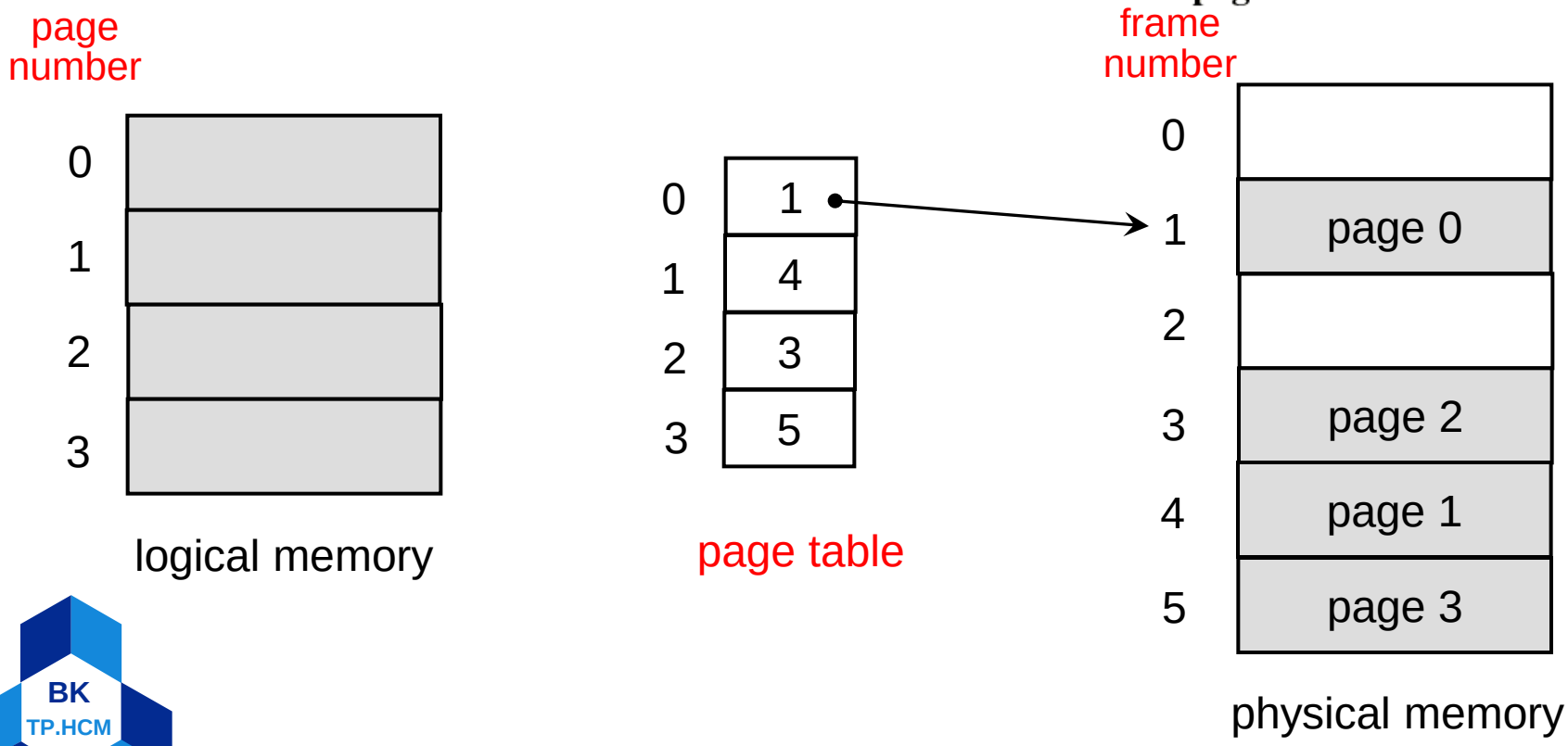
- Không gian địa chỉ của 1 quá trình được chia thành những khối có cùng kích thước gọi là trang. Kỹ thuật *phân trang* (paging) cho phép không gian địa chỉ vật lý (physical address space) của một process không cần liên tục.
- Bộ nhớ thực được chia thành các khối cố định và có kích thước bằng nhau gọi là *frame (tương ứng với trang)*.
  - Thông thường kích thước của frame là lũy thừa của 2, từ khoảng 512 byte đến 16 MB.
- *Bộ nhớ luận lý* (logical memory) hay *không gian địa chỉ luận lý* là tập mọi địa chỉ luận lý của một quá trình.
  - Địa chỉ luận lý có thể được quá trình sinh ra bằng cách dùng indexing, base register, segment register,…

Kỹ thuật phân trang (tt.)

- Frame và trang nhớ có kích thước bằng nhau.
- Hệ điều hành phải thiết lập một *bảng phân trang* (page table) để ánh xạ địa chỉ luận lý thành địa chỉ thực
  - Mỗi process có một bảng phân trang, được quản lý qua một con trỏ lưu giữ trong PCB.
  - Thiết lập bảng phân trang cho process là một phần của chuyển ngữ cảnh
- Kỹ thuật phân trang khiến bộ nhớ bị phân mảnh nội, nhưng khắc phục được phân mảnh ngoại.

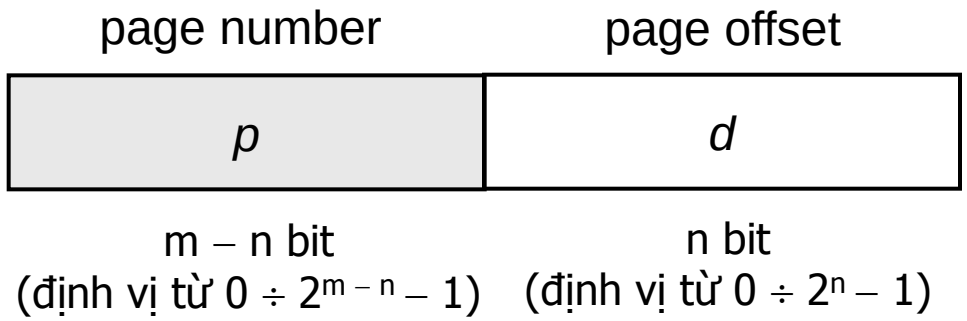
Ví dụ: Phân trang





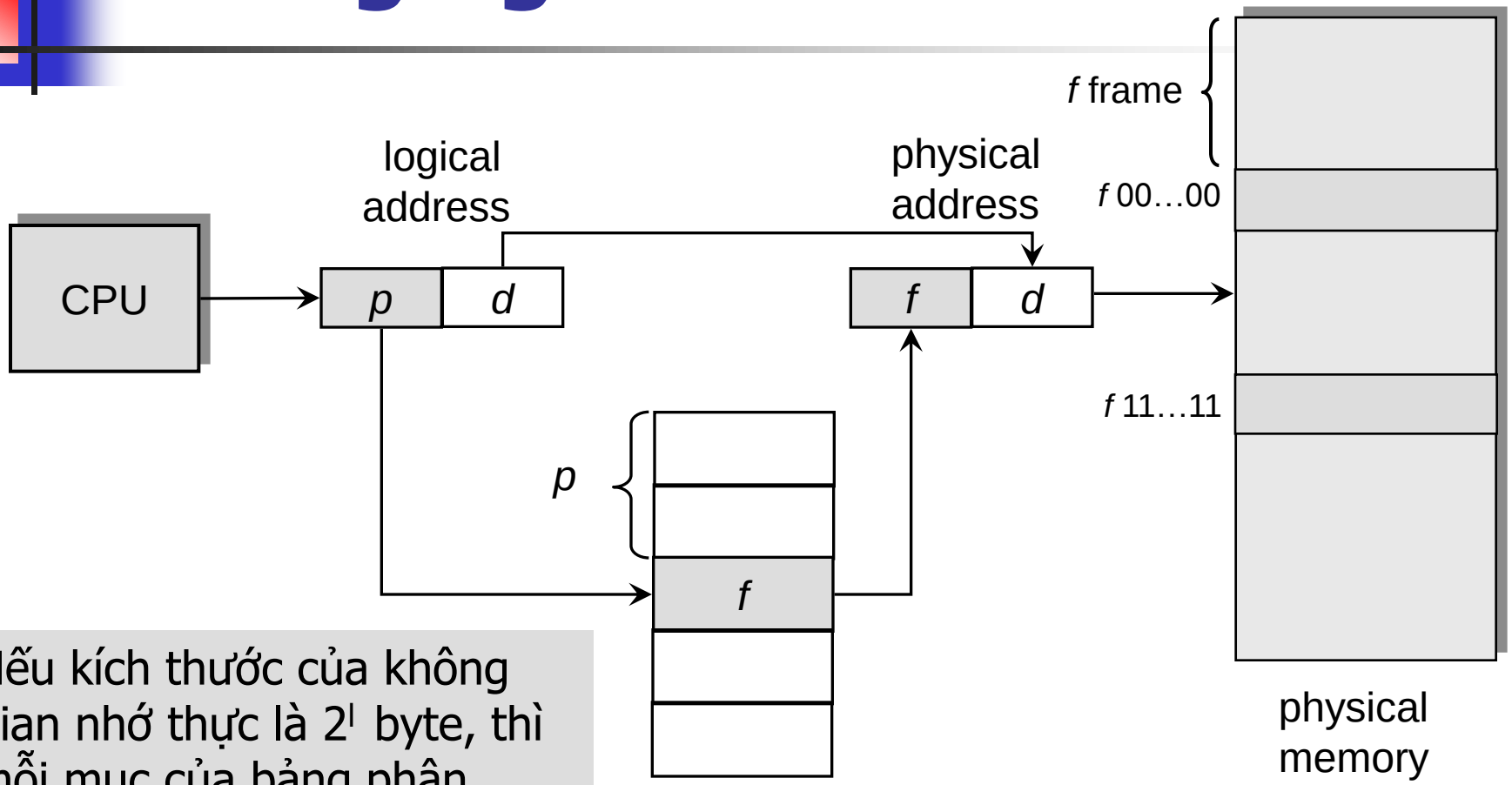
Chuyển đổi địa chỉ trong paging

- *Địa chỉ luận lý* gồm có:
  - *Page number*, *p*, được dùng làm chỉ mục vào bảng phân trang. Mỗi mục (entry) trong bảng phân trang chứa chỉ số frame (còn gọi là số frame cho gọn) của trang tương ứng trong bộ nhớ thực.
  - *Page offset*, *d*, được kết hợp với *địa chỉ nền* (base address) của frame để định vị địa chỉ thực.
- Nếu kích thước của không gian địa chỉ ảo là 2^m, và kích thước của trang là 2^n (byte hay word tùy theo kiến trúc máy)

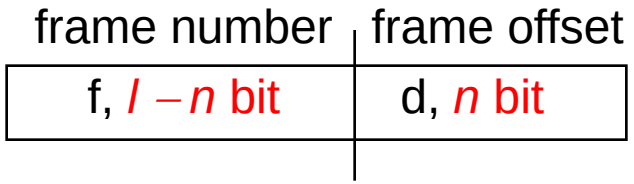


Bảng phân trang sẽ có tổng cộng $2^m/2^n = 2^{m-n}$ mục

Paging hardware

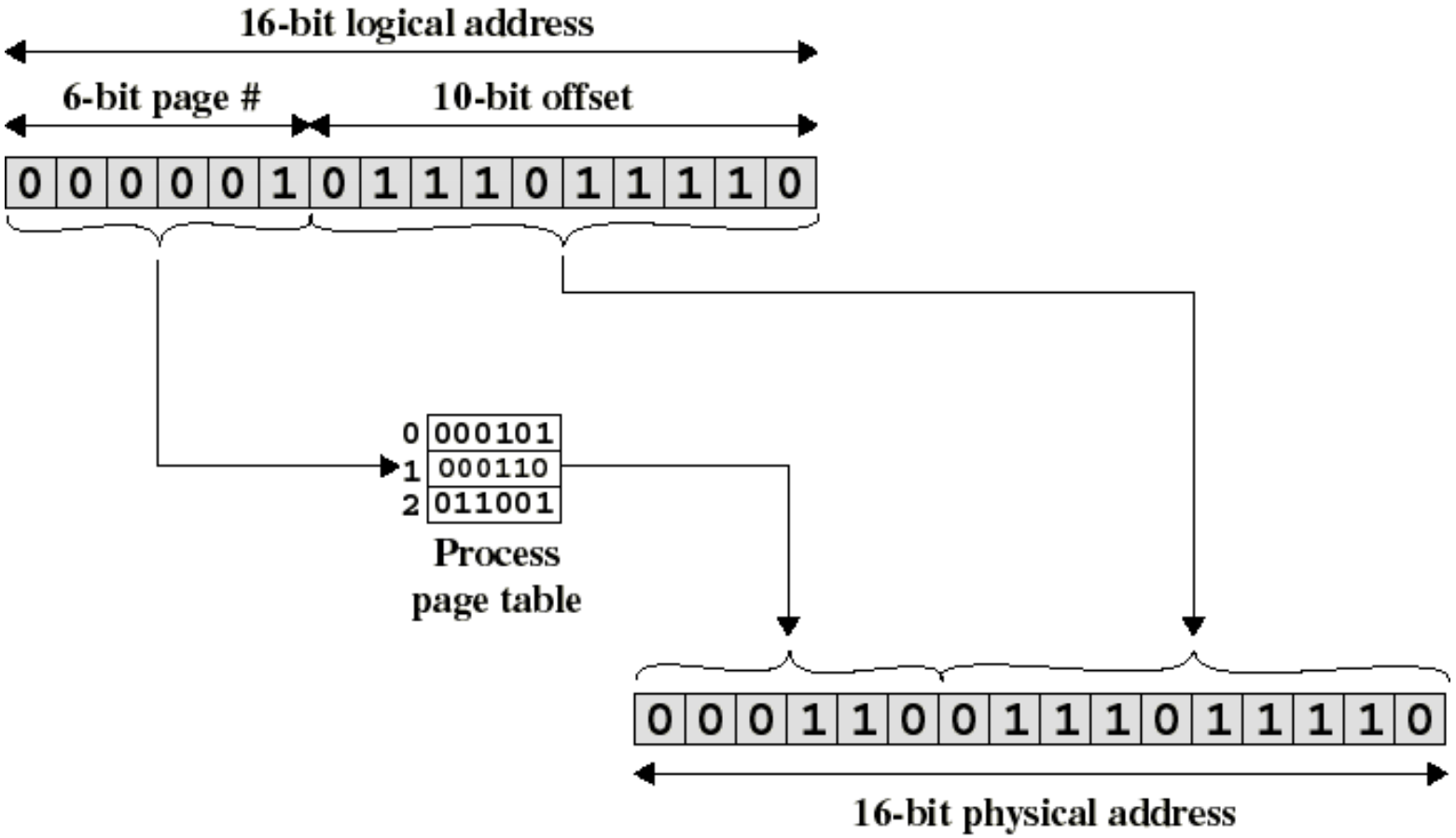


Nếu kích thước của không gian nhớ thực là 2^l byte, thì mỗi mục của bảng phân trang có l – n bit



Chuyển đổi địa chỉ nhớ trong paging

- Ví du:



Hiện thực bảng phân trang

- Bảng phân trang thường được lưu giữ trong bộ nhớ chính
  - Mỗi process được hệ điều hành cấp một bảng phân trang
  - Thanh ghi *page-table base* (PTBR) trỏ đến bảng phân trang
  - Thanh ghi *page-table length* (PTLR) biểu thị kích thước của bảng phân trang (có thể được dùng trong cơ chế bảo vệ bộ nhớ)
- Mỗi truy cập dữ liệu/lệnh cần hai thao tác truy xuất vùng nhớ
  - Dùng page number p làm index để truy cập mục trong bảng phân trang nhằm lấy số frame, và kế đến là dùng page offset d để truy xuất dữ liệu/lệnh trong frame
- Thường dùng một cache phần cứng có tốc độ truy xuất và tìm kiếm cao, gọi là *thanh ghi kết hợp* (associative register) hoặc *translation look-aside buffers* (TLBs)

Associative register (hardware)

- hay TLB, là thanh ghi hỗ trợ tìm kiếm truy xuất dữ liệu với tốc độ cực nhanh.

Page #	Frame #

Số mục của TLB khoảng 8 ÷ 2048

TLB là "cache" của bảng phân trang

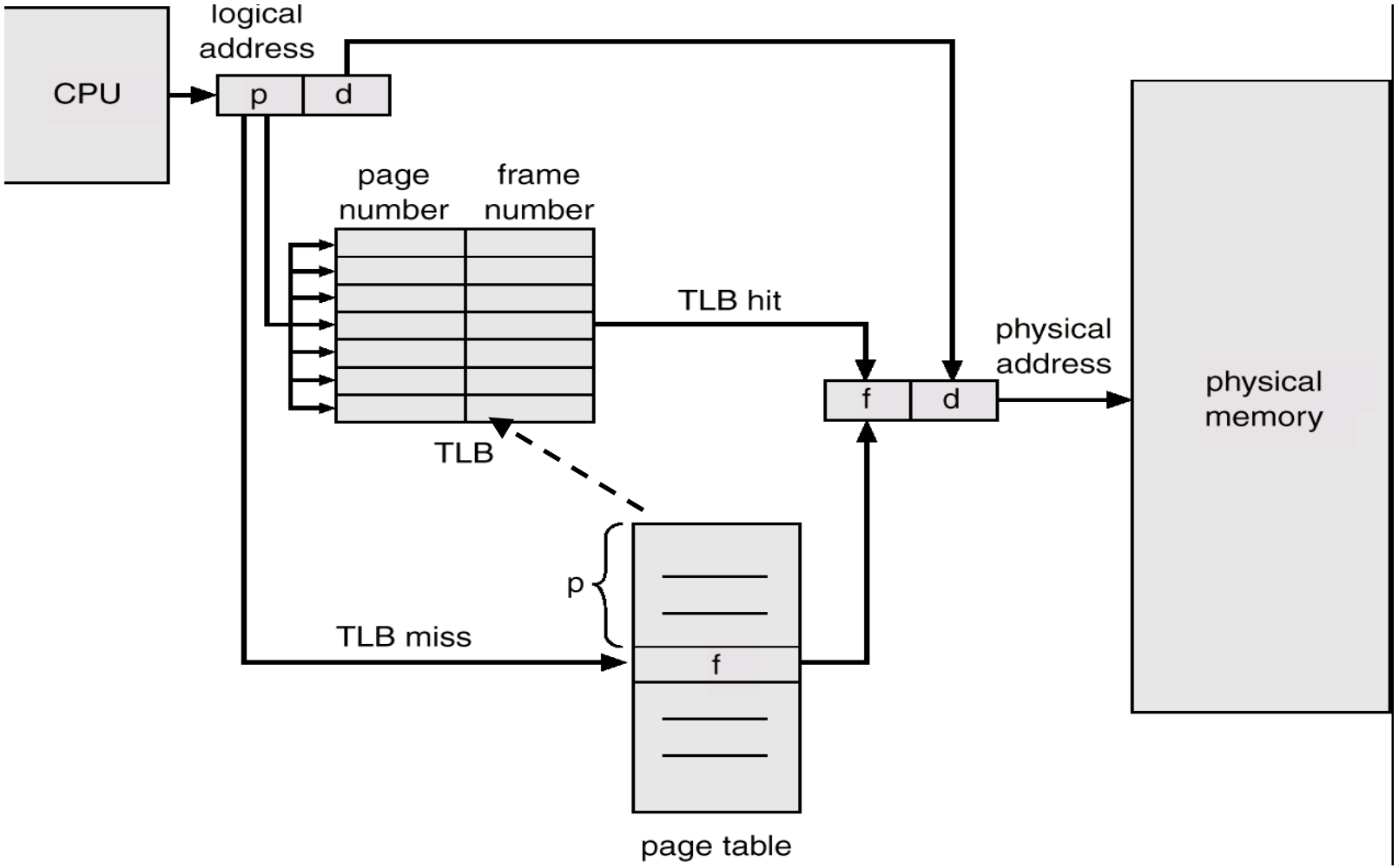
Khi có chuyển ngữ cảnh, TLB bị xóa

Khi TLB đầy, thay thế mục dùng LRU

Ánh xạ page #

- Nếu page number nằm trong TLB (: hit, trúng) ⇒ lấy ngay được số frame ⇒ tiết kiệm được thời gian truy cập bộ nhớ chính để lấy số frame trong bảng phân trang.
- Ngược lại (: miss, trật), phải lấy số frame từ bảng phân trang như bình thường.

Paging hardware với TLB



Effective access time (EAT)

Tính thời gian truy xuất hiệu dụng (EAT)

- Thời gian tìm kiếm trong TLB (associative lookup): ε
- Thời gian một chu kỳ truy xuất bộ nhớ: x
- *Hit ratio:* tỉ số giữa số lần chỉ số trang được tìm thấy (hit) trong TLB và số lần truy xuất khởi nguồn từ CPU
  - Kí hiệu hit ratio: α
- Thời gian cần thiết để có được địa chỉ thực
  - Khi chỉ số trang có trong TLB (hit) $\varepsilon + x$
  - Khi chỉ số trang không có trong TLB (miss) $\varepsilon + x + x$
- *Thời gian truy xuất hiệu dụng*

$$\mathbf{EAT} = (\varepsilon + x)\alpha + (\varepsilon + 2x)(1 - \alpha)$$
$$= (2 - \alpha)x + \varepsilon$$

Effective access time (tt.)

- Giả sử (đơn vị thời gian: nano giây)
  - Associative lookup = 20
  - Memory access = 100

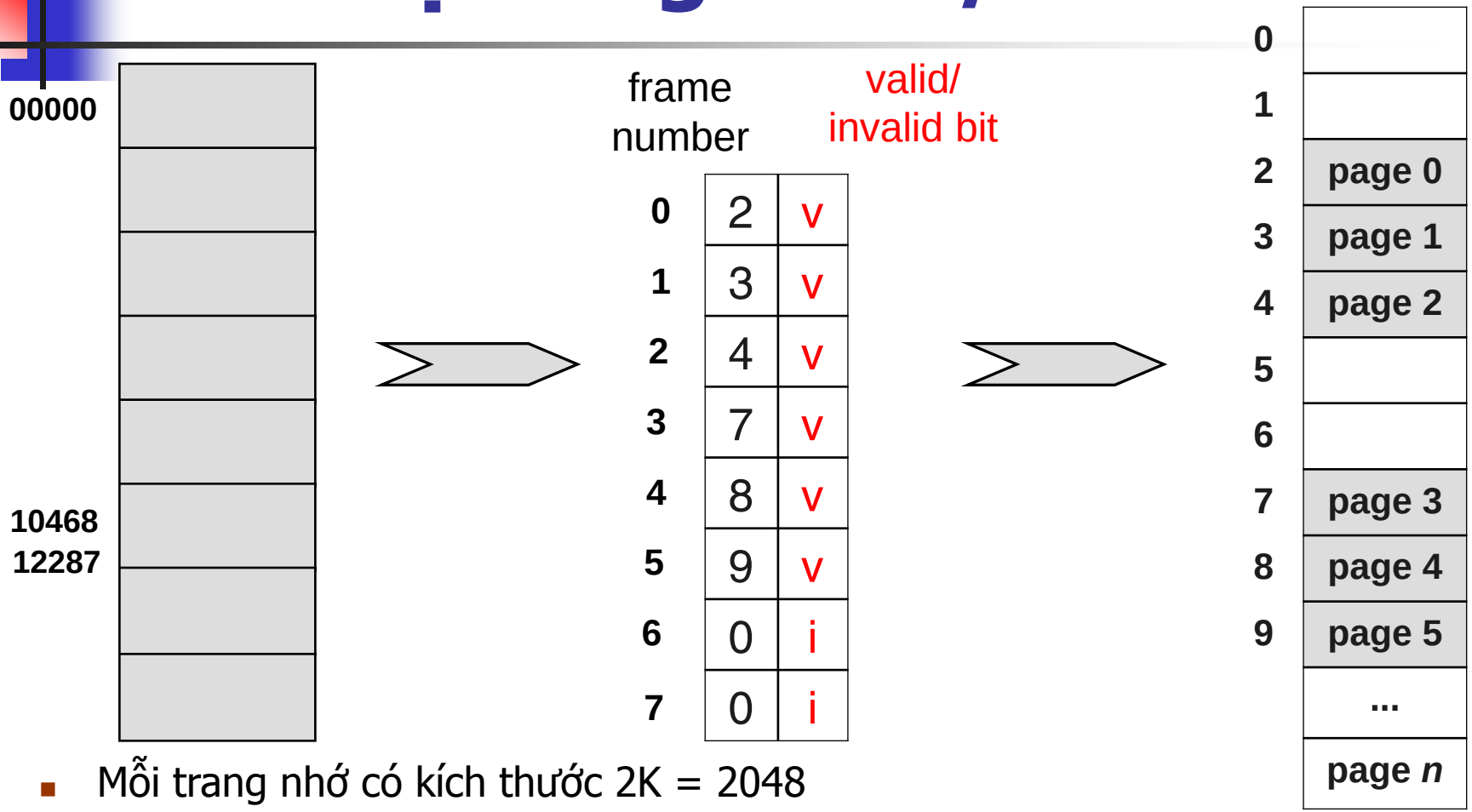
- Ví dụ 1
  - Hit ratio = 0,8
  - EAT = (100 + 20) × 0,8 + (200 + 20) × 0,2
    = 1,2 × 100 + 20
    = 140

- Ví dụ 2
  - Hit ratio = 0,98
  - EAT = (100 + 20) × 0,98 + (200 + 20) × 0,02
    = 1,02 × 100 + 20
    = 122

Bảo vệ bộ nhớ

- Việc bảo vệ bộ nhớ được hiện thực bằng cách gắn với frame các *bit bảo vệ* (protection bit) được giữ trong bảng phân trang. Các bit này biểu thị các thuộc tính sau
  - read-only, read-write, execute-only
- Ngoài ra, còn có một *valid/invalid bit* gắn với mỗi mục trong bảng phân trang
  - "valid": cho biết là trang của process, do đó là một trang hợp lệ.
  - "invalid": cho biết là trang không của process, do đó là một trang bất hợp lệ.

Bảo vệ bằng valid/invalid bit



- Mỗi trang nhớ có kích thước 2K = 2048
- Process có kích thước 10.468 ⇒ phân mảnh nội ở frame 9 (chứa page 5), các địa chỉ ảo > 12287 là các địa chỉ invalid.
- Dùng PTLR để kiểm tra truy xuất đến bảng phân trang có nằm trong bảng hay không.

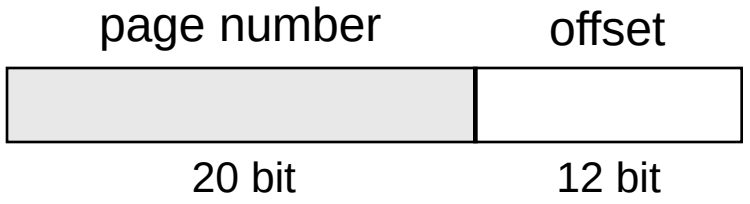
Bảng phân trang 2 mức

- Các hệ thống hiện đại đều hỗ trợ không gian địa chỉ ảo rất lớn (2^{32} đến 2^{64}), ở đây giả sử là 2^{32}
  - Giả sử kích thước trang nhớ là 4 KB (= 2^{12})
    $\Rightarrow$ bảng phân trang sẽ có $2^{32}/2^{12} = 2^{20}$ = 1 M mục.
  - Giả sử mỗi mục gồm 4 byte thì mỗi process cần 4 MB cho bảng phân trang ☹
- Một giải pháp là, thay vì dùng một bảng phân trang duy nhất cho mỗi process, "paging" bảng phân trang này, và chỉ giữ những bảng phân trang cần thiết trong bộ nhớ $\rightarrow$ *bảng phân trang 2 mức* (two-level page table).

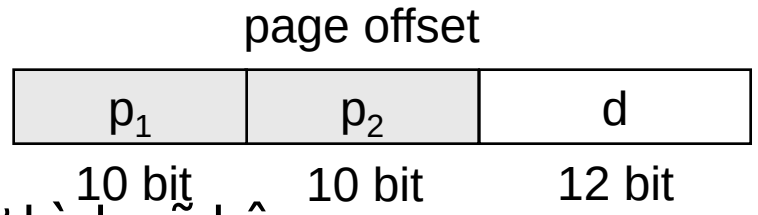
Bảng phân trang 2-mức (tt.)

Ví dụ

- Một địa chỉ luận lý trên hệ thống 32-bit với trang nhớ 4K được chia thành các phần sau:
  - Page number: 20 bit
    - Nếu mỗi mục dài 4 byte
      $\Rightarrow$ Cần 2^{20} 4 byte = 4 MB
      cho mỗi page table



- Bảng phân trang cũng được chia nhỏ nên page number cũng được chia nhỏ thành 2 phần:
  - 10-bit page number
  - 10-bit page offset



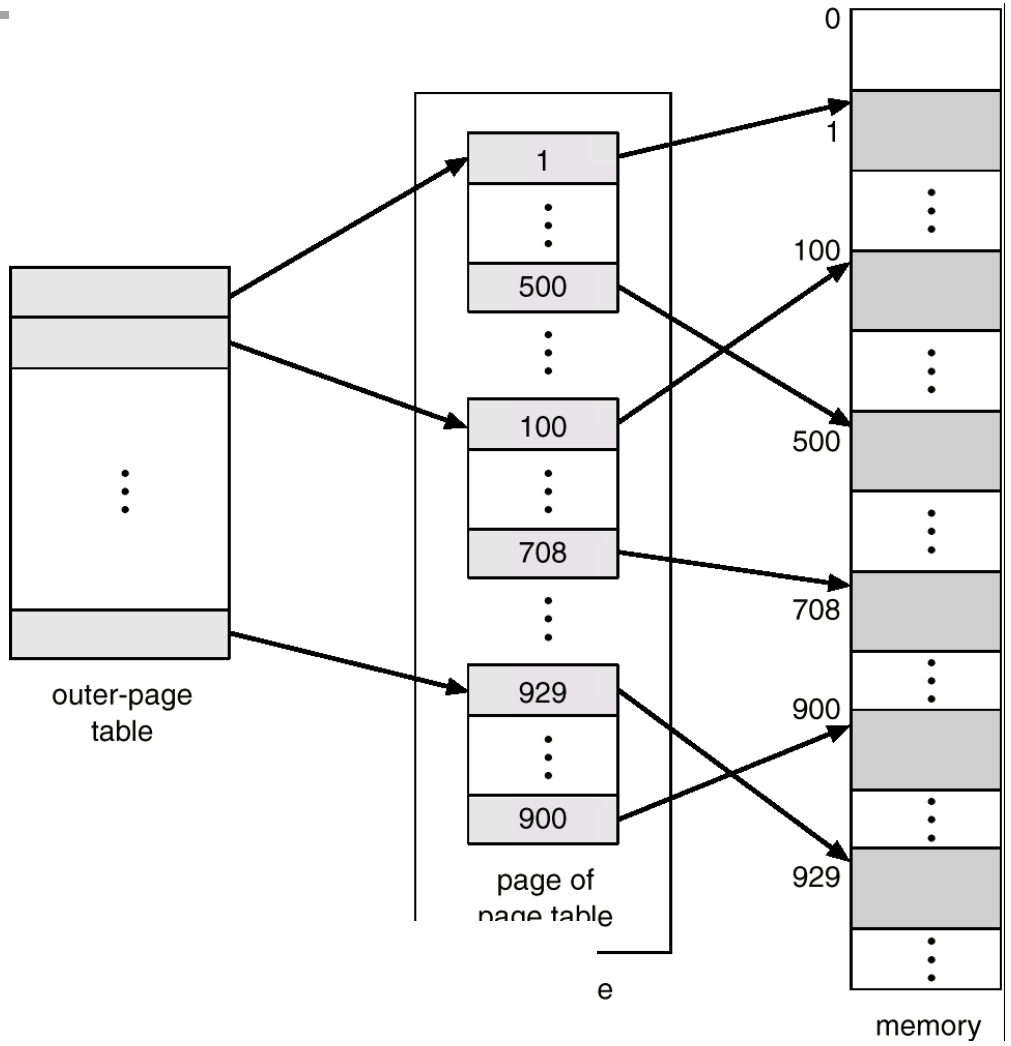
- Vì vậy, một địa chỉ luận lý sẽ như hình vẽ bên

p_1 : chỉ số của mục trong *bảng phân trang mức 1* (outer-page table)

p_2 : chỉ số của mục trong *bảng phân trang mức 2*

Bảng phân trang 2-mức (tt.)

Minh họa

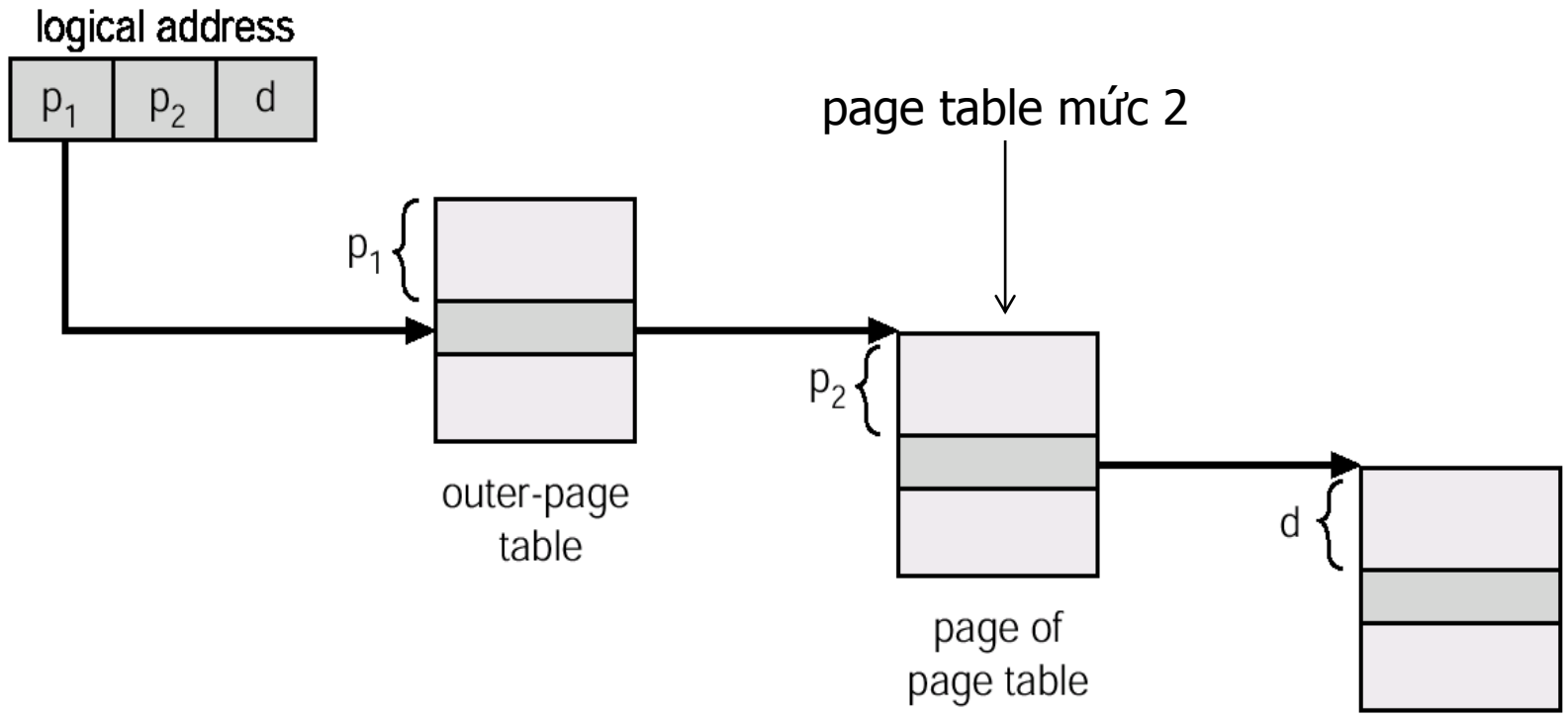


- Có 2^{p1} mục trong bảng phân trang mức 1
- Mỗi bảng phân trang mức 2 chứa 2^{p2} mục

các bảng phân trang mức 2

Bảng phân trang 2 mức (tt.)

- Sơ đồ chuyển đổi địa chỉ (address-translation scheme) cho kỹ thuật phân trang 2 mức, với 32-bit địa chỉ



Bảng phân trang 2 mức (tt.)

- Bảng phân trang 2 mức giúp tiết kiệm bộ nhớ:
  - Vùng màu đỏ tương ứng với phần chưa được sử dụng của không gian địa chỉ ảo. Các mục màu đỏ được đánh dấu là không có frame nên sẽ gây ra page fault nếu được tham chiếu đến; khi đó OS sẽ tạo thêm bảng phân trang mức 2 mới.

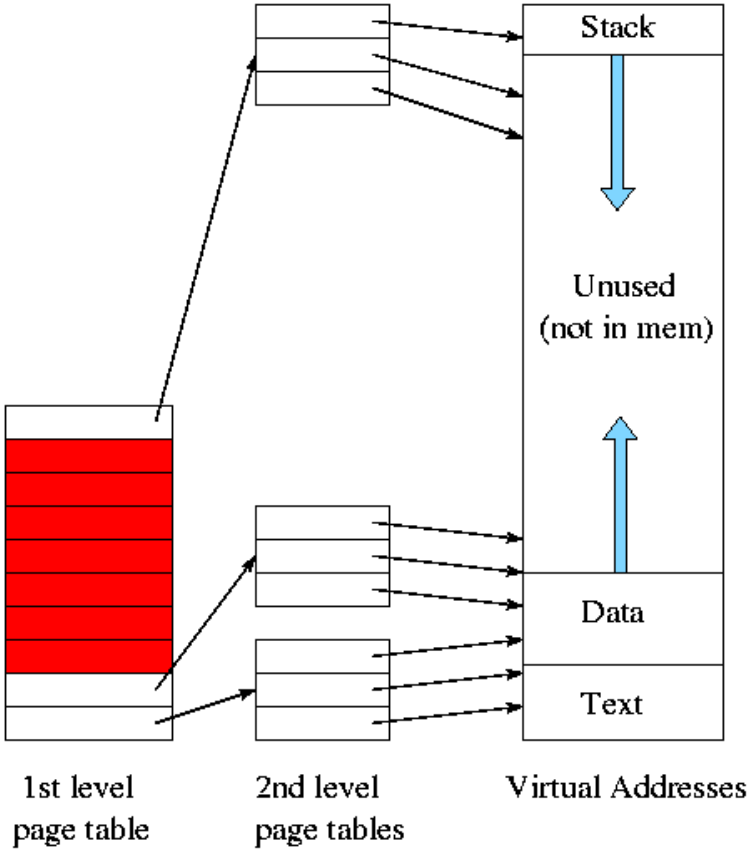
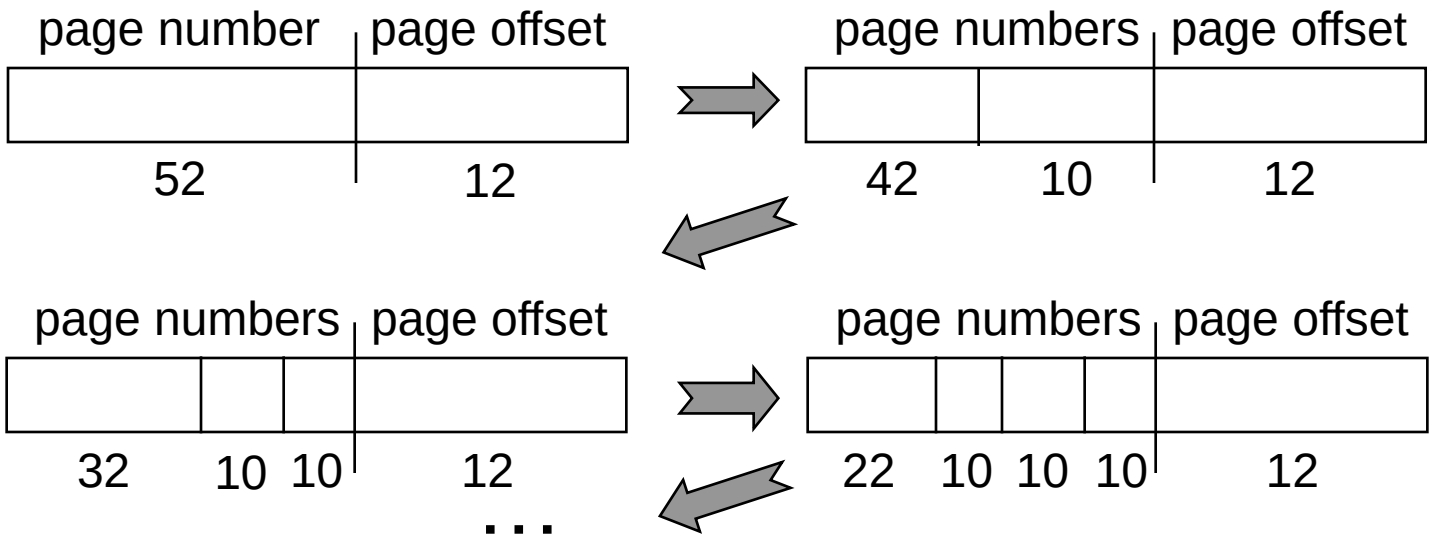


Fig from Gottlieb

Bảng phân trang đa mức

- Ví dụ: Không gian địa chỉ luận lý 64-bit với trang nhớ 4K
  - Bảng phân trang 2-mức vẫn còn quá lớn! Tương tự bảng phân trang 2 mức, ta có bảng phân trang 3, 4,..., n mức

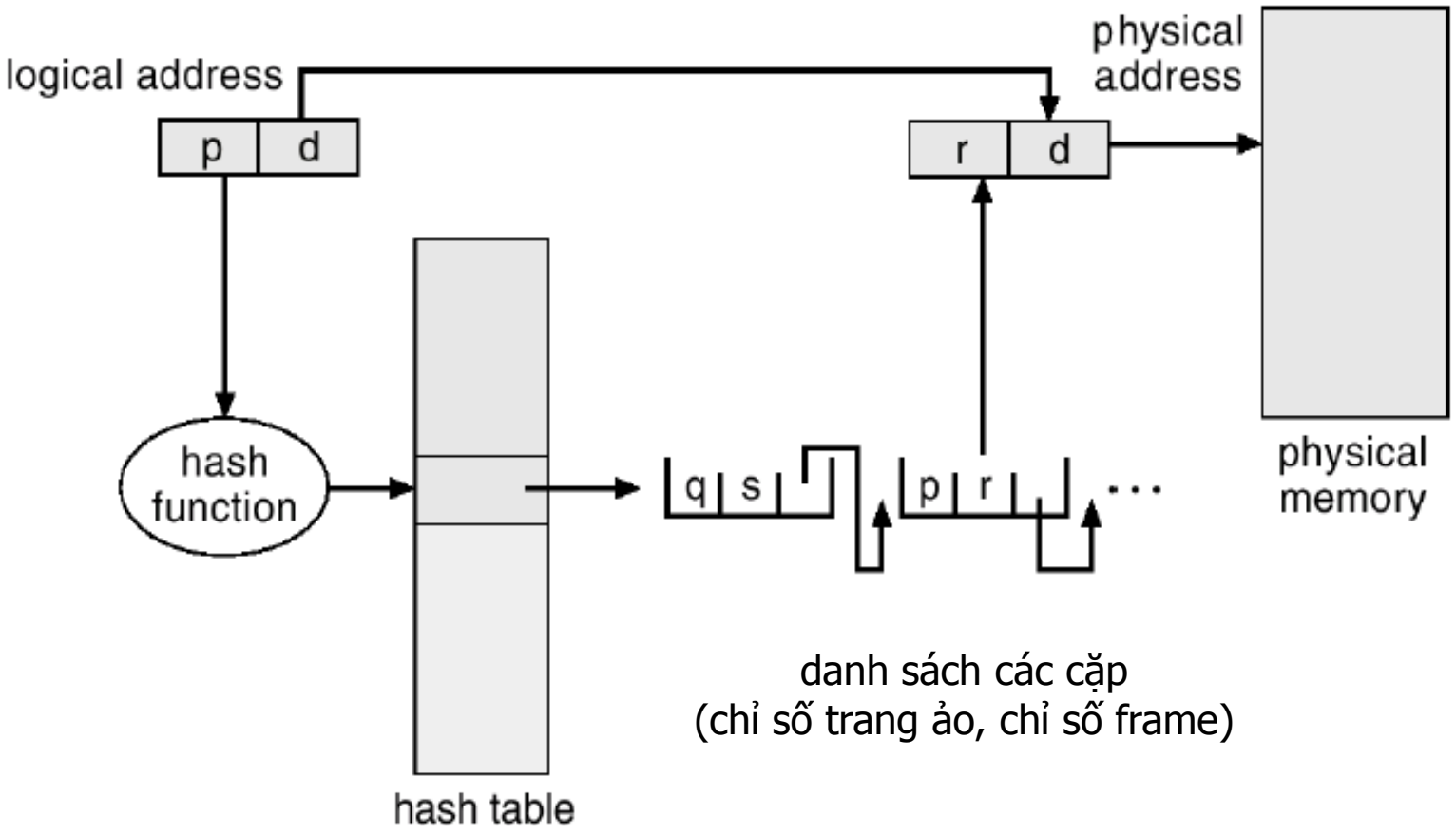


- Tiết kiệm chổ trong bộ nhớ chính bằng cách chỉ giữ trong bộ nhớ chính các bảng phân trang mà process hiện đang cần.

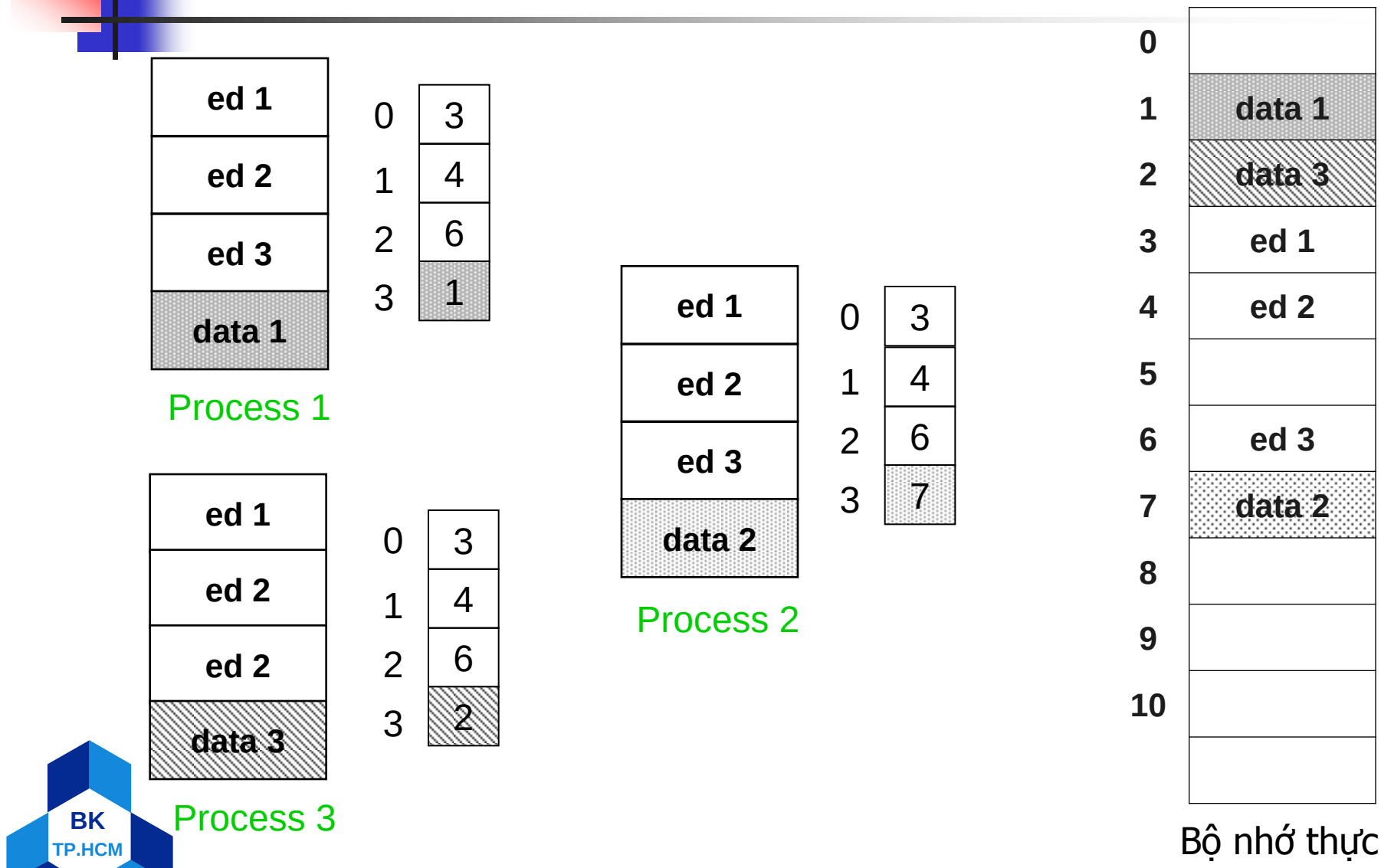
Bảng phân trang băm

- Dùng kỹ thuật băm để giảm không gian bảng phân trang
  - Phổ biến trong các hệ thống có địa chỉ lớn hơn 32 bit.
- Để giải quyết đụng độ, mỗi mục của bảng băm được gắn một danh sách liên kết. Mỗi phần tử của danh sách là một cặp (chỉ số trang ảo, chỉ số frame).
  - Chỉ số trang ảo (virtual page number) được biến đổi qua hàm băm thành một *hashed value*. Cặp (chỉ số trang ảo, chỉ số frame) sẽ được lưu vào danh sách liên kết tại mục có chỉ số là hashed value.
- Giải thuật tìm trang:
  - Chỉ số trang ảo được biến đổi thành hashed value. Hashed value là chỉ số của mục cần truy cập trong bảng băm. Sau đó, tìm trong danh sách liên kết phần tử chứa chỉ số trang ảo để trích ra được chỉ số frame tương ứng.

Hashed page table (tt.)



Chia sẻ các trang nhớ



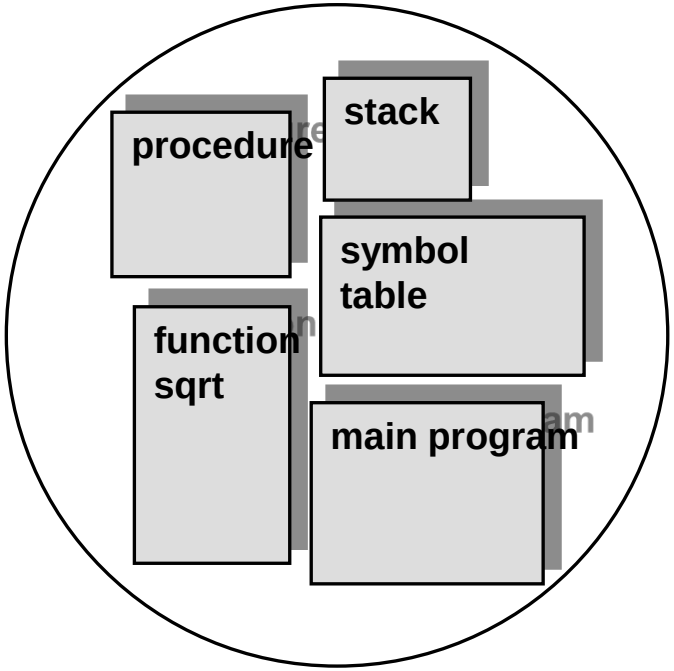
Phân đoạn

- Nhìn lại kỹ thuật phân trang
  - user view (không gian địa chỉ ảo) tách biệt với không gian bộ nhớ thực. Kỹ thuật phân trang thực hiện phép ánh xạ user-view vào bộ nhớ thực.

- Trong thực tế, dưới góc nhìn của user, một chương trình cấu thành từ nhiều *đoạn* (segment). Mỗi đoạn là một đơn vị luận lý của chương trình, như
  - main program, procedure, function
  - local variables, global variables, common block, stack, symbol table, arrays,…

Chương trình: user view

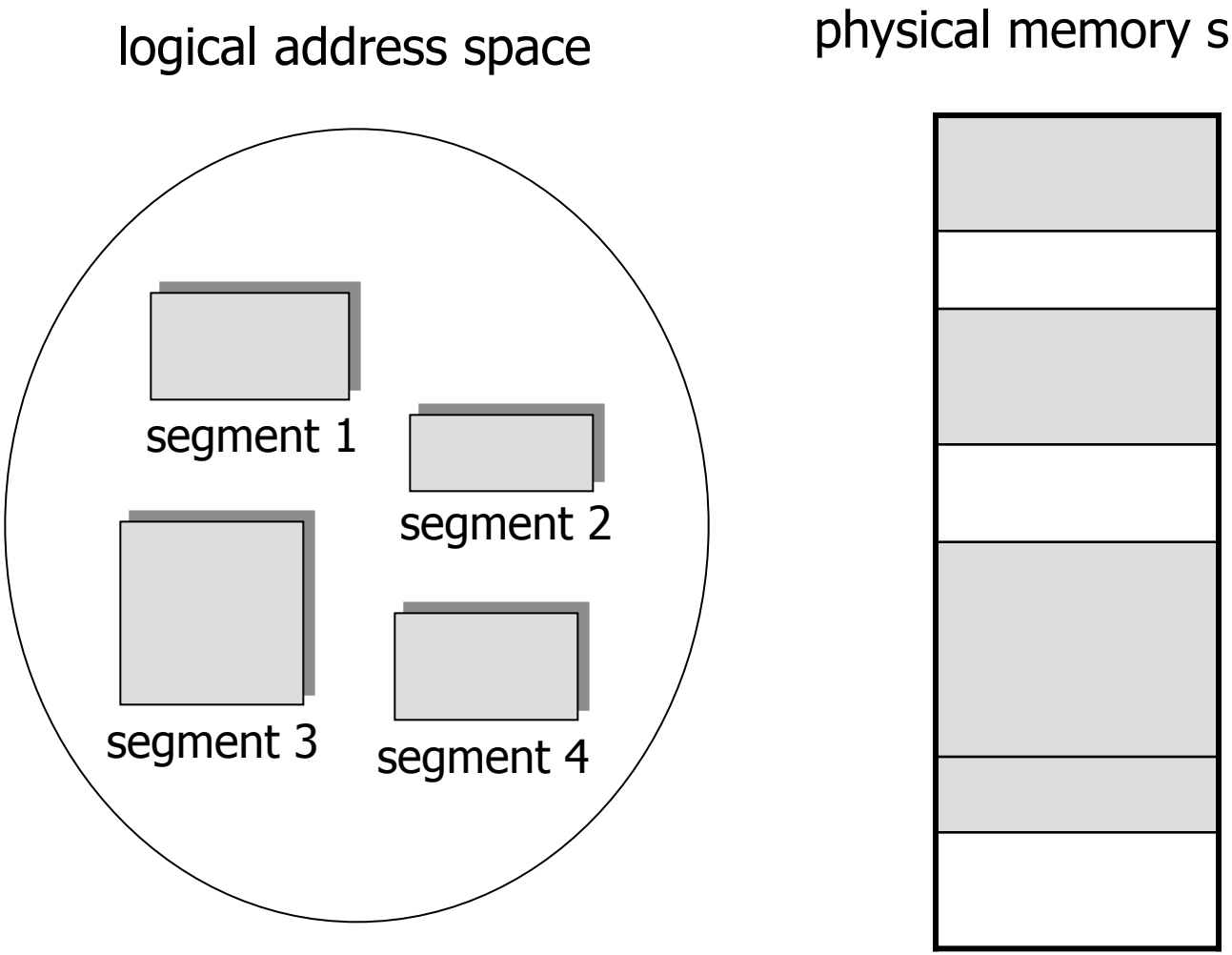
- Thông thường, một chương trình được biên dịch. Trình biên dịch sẽ tự động xây dựng các segment.
- Ví dụ, trình biên dịch Pascal sẽ tạo ra các segment sau:
  - Global variables
  - Procedure call stack
  - Procedure/function code
  - Local variable
- Trình loader sẽ gán mỗi segment một số định danh riêng.



Phân đoạn (tt.)

- Dùng kỹ thuật *phân đoạn* (segmentation) để quản lý bộ nhớ có hỗ trợ user view
  - *Không gian địa chỉ ảo* là một tập các đoạn, mỗi đoạn có tên và kích thước riêng.
  - Một địa chỉ luận lý được định vị bằng tên đoạn và độ dời (offset) bên trong đoạn đó (so sánh với phân trang!)

Phân đoạn (tt.)


logical address space
physical memory space
segment 1
segment 2
segment 3
segment 4

Hiện thực phân đoạn

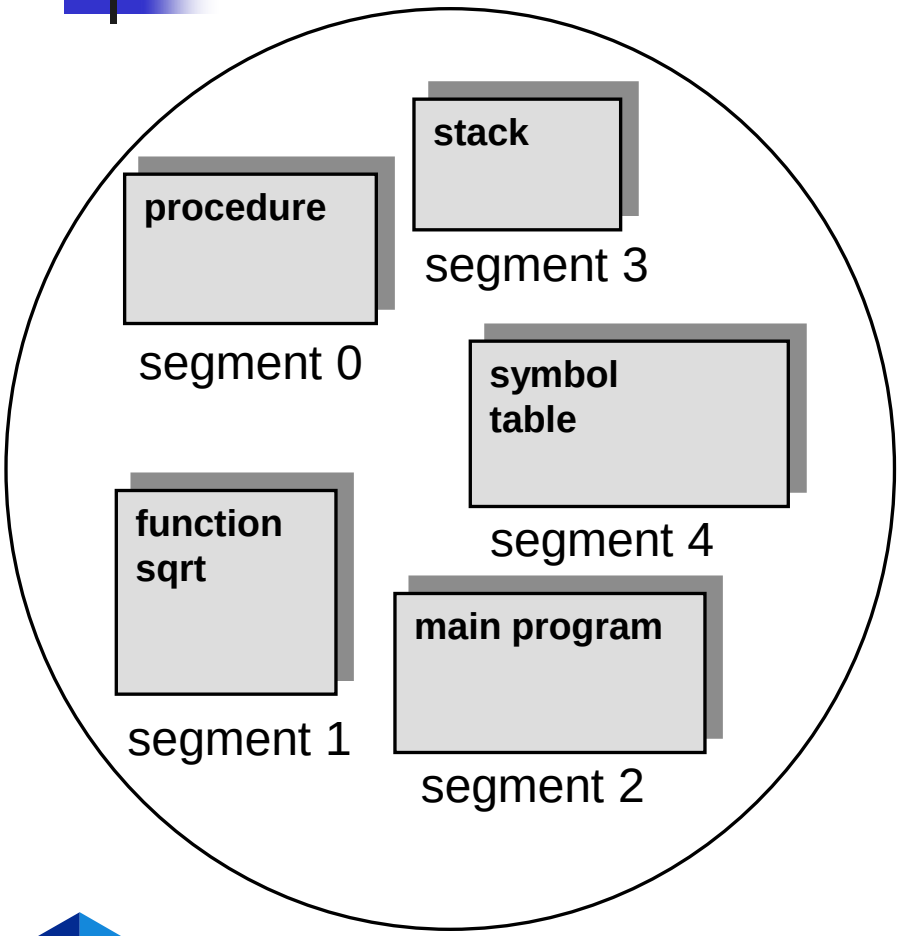
- *Địa chỉ luận lý* là một cặp giá trị

  (segment number, offset)

- *Bảng phân đoạn* (segment table): gồm nhiều mục, mỗi mục chứa
  - base, chứa địa chỉ khởi đầu của segment trong bộ nhớ
  - limit, xác định kích thước của segment
- *Segment-table base register* (STBR): trỏ đến vị trí bảng phân đoạn trong bộ nhớ
- *Segment-table length register* (STLR): số lượng segment của chương trình

  ⇒ Một chỉ số segment s là hợp lệ nếu s < STLR

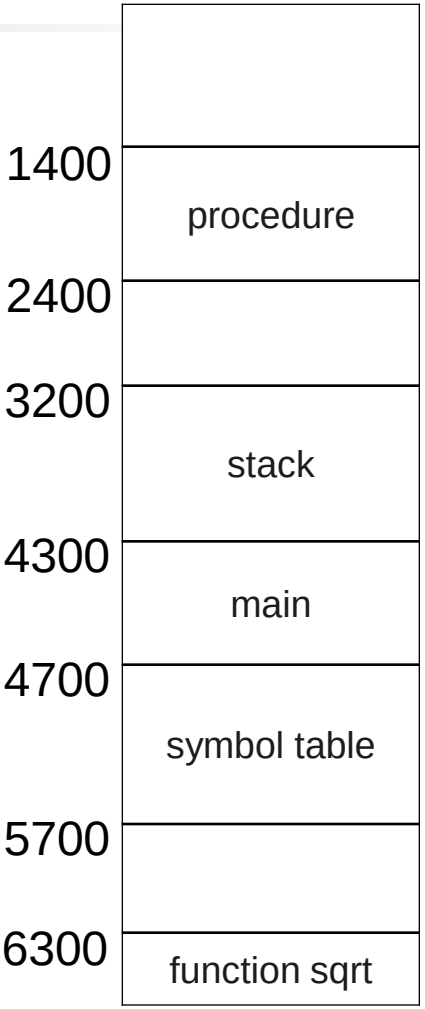
Một ví dụ về phân đoạn



logical address space

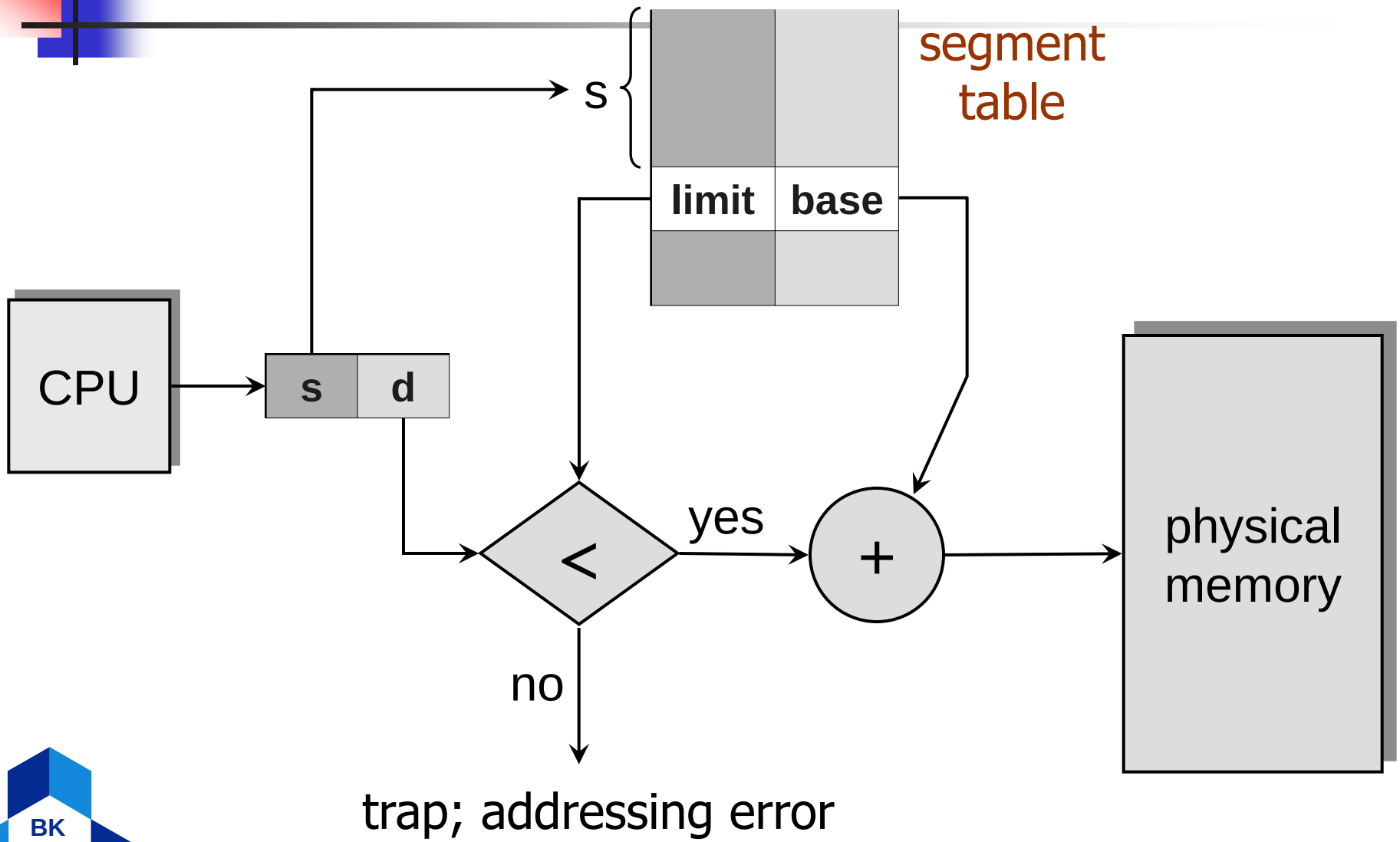
	limit	base
0	1000	1400
1	400	6300
2	400	4300
3	1100	3200
4	1000	4700

segment table



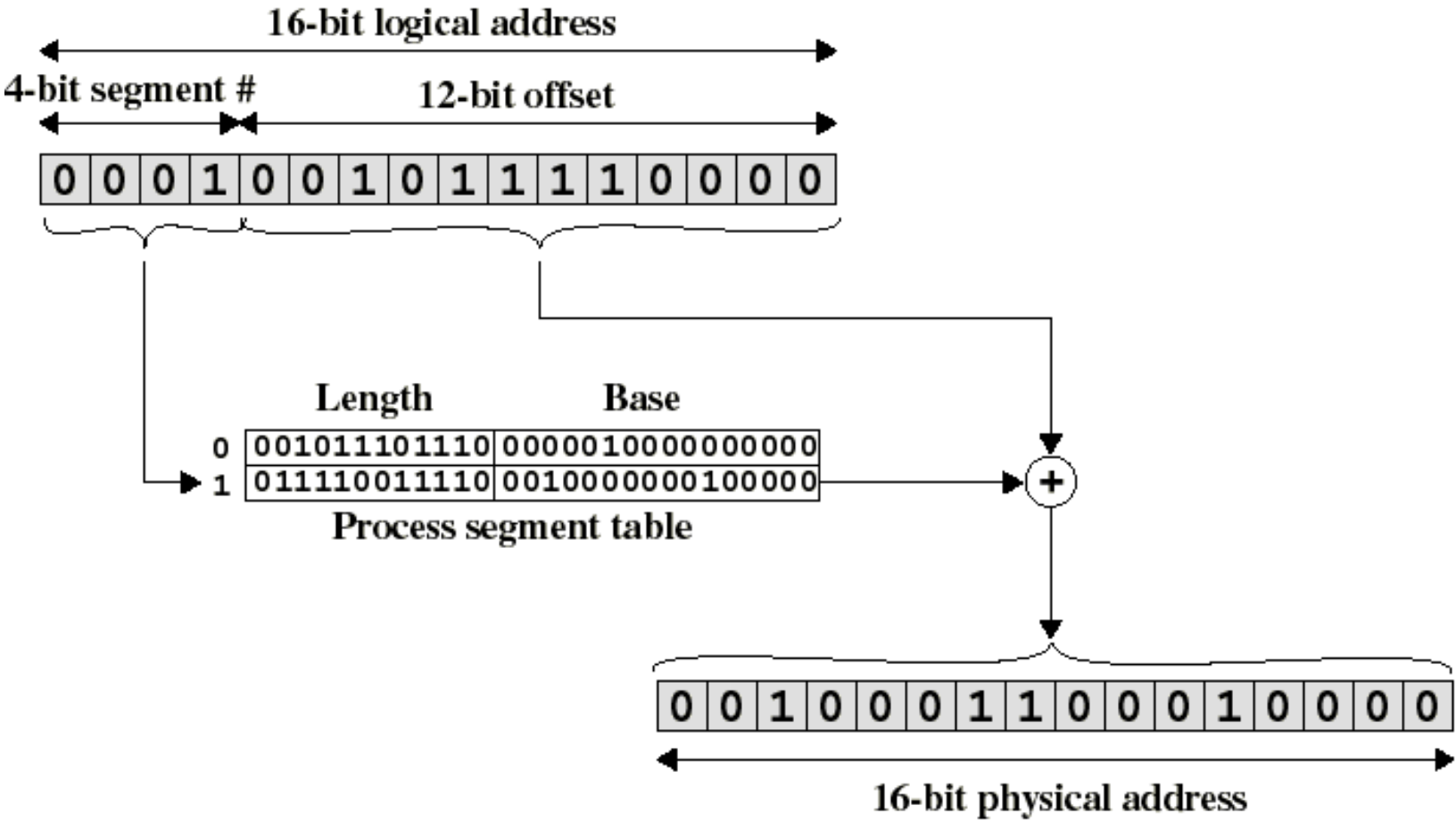
physical memory space

Phần cứng hỗ trợ phân đoạn

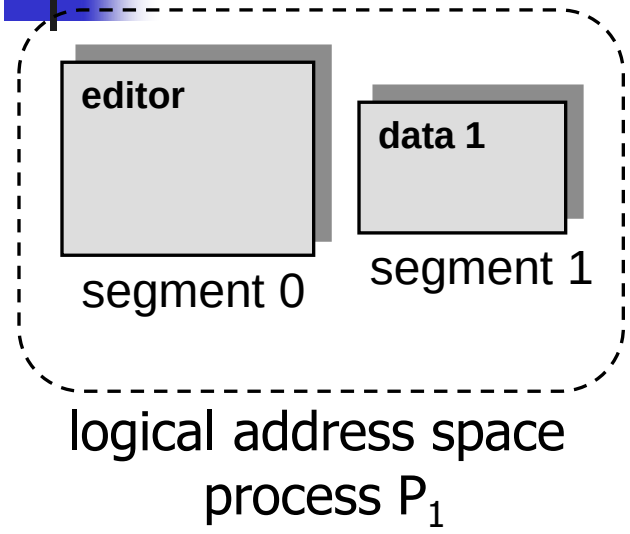


Chuyển đổi địa chỉ trong kỹ thuật phân đoạn

- Ví dụ



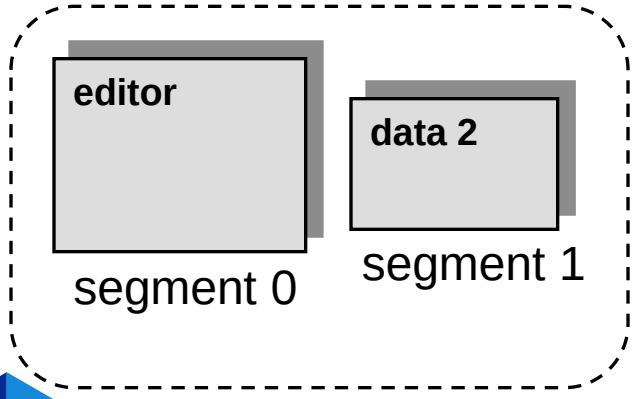
Chia sẻ các đoạn



logical address space
process P_1

	limit	base
0	25286	43062
1	4425	68348

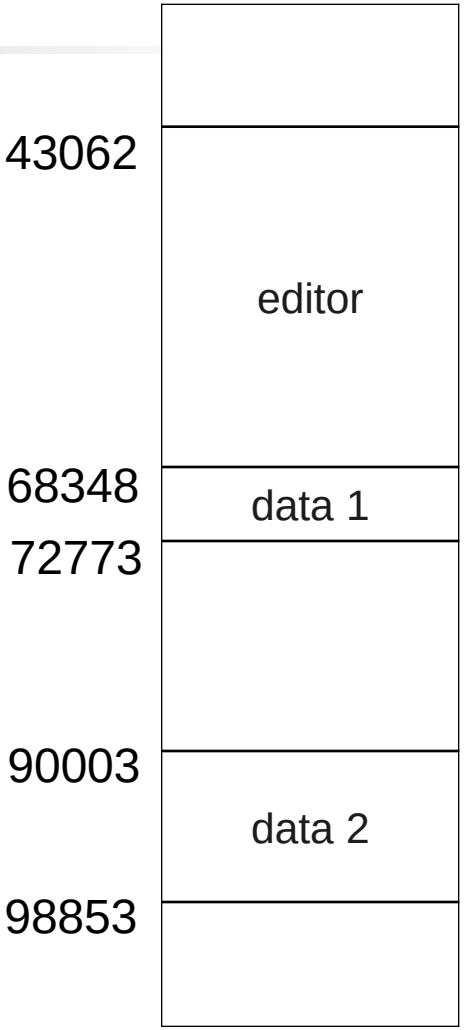
segment table
process P_1



logical address space
process P_2

	limit	base
0	25286	43062
1	8850	90003

segment table
process P_2



physical memory

Kết hợp phân trang và phân đoạn

- Kết hợp phân trang và phân đoạn nhằm tận dụng các ưu điểm và hạn chế các khuyết điểm của chúng:
  - Vấn đề của phân đoạn: một đoạn có thể không nạp được vào bộ nhớ do phân mảnh ngoại, mặc dù đủ không gian trống.
  - Ý tưởng giải quyết: paging đoạn, cho phép các page của đoạn được nạp vào các frame không cần nằm liên tục nhau.

Kết hợp phân trang và phân đoạn (tt.)

- Có nhiều cách kết hợp. Một cách đơn giản là *segmentation with paging*

Mỗi process sẽ có:

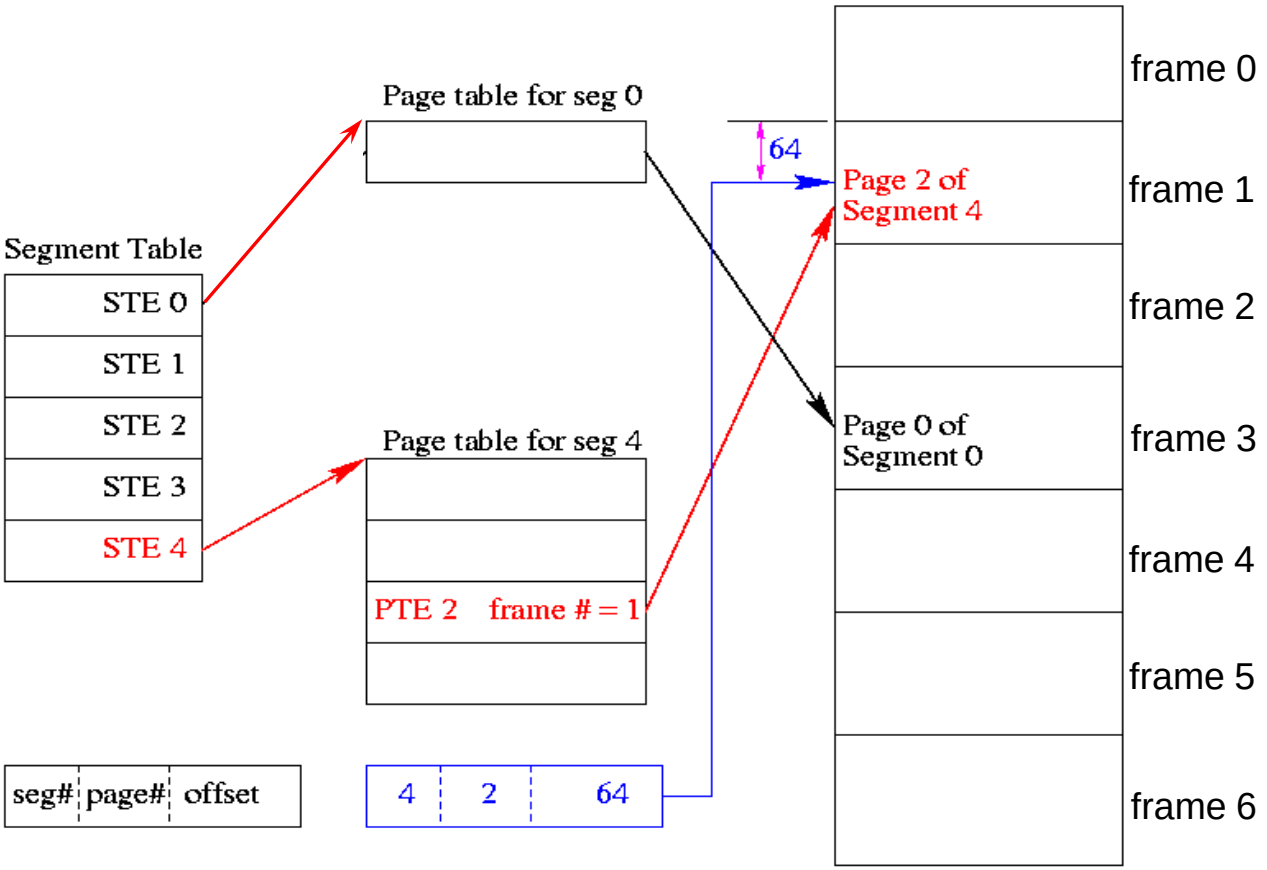
- Một bảng phân đoạn
- Nhiều bảng phân trang: mỗi đoạn có một bảng phân trang

Một *địa chỉ luận lý* (địa chỉ ảo) bao gồm:

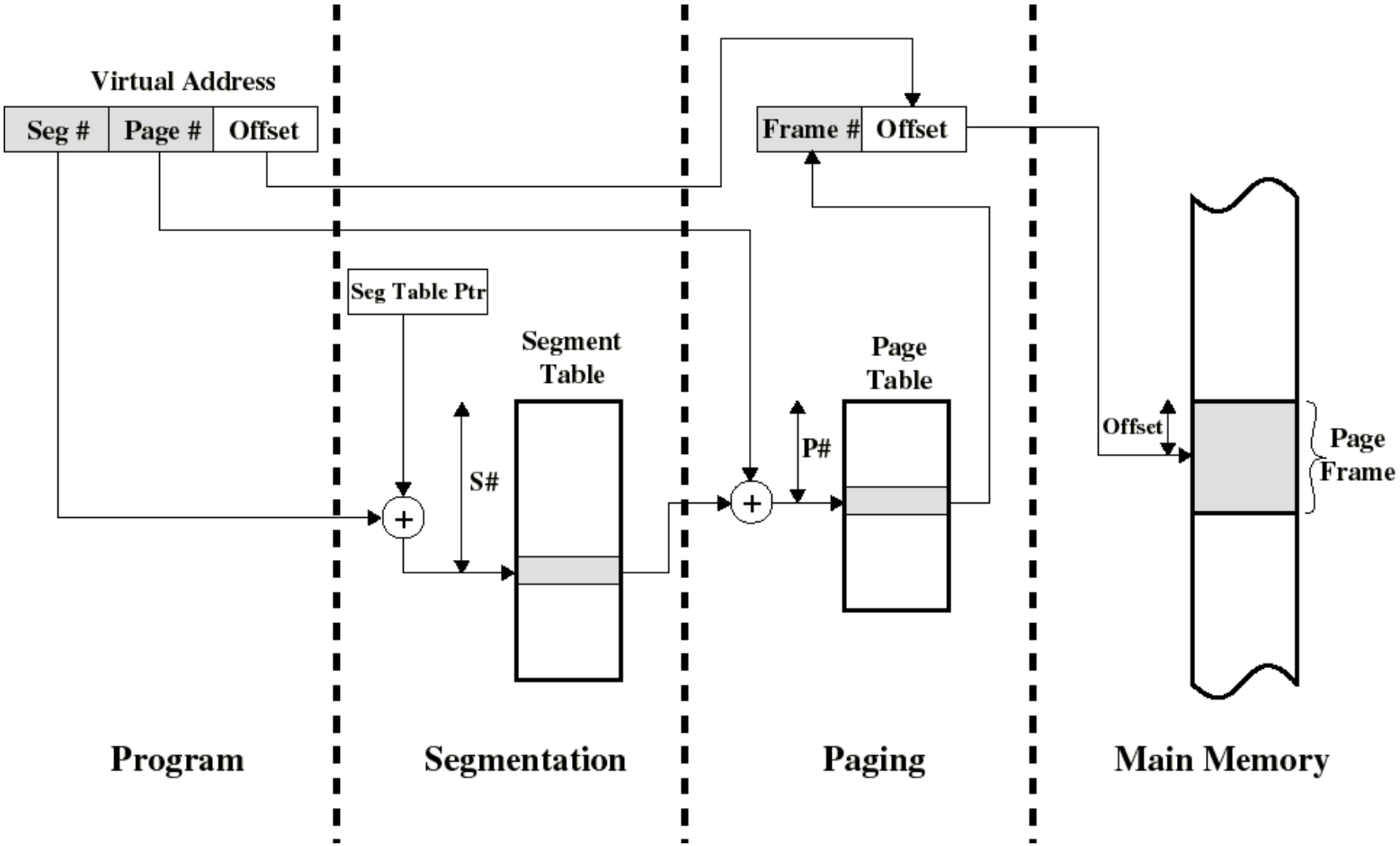
- *segment number:* là chỉ số của một mục trong bảng phân đoạn, mục này chứa địa chỉ nền (base address) của bảng phân trang cho đoạn đó
- *page number:* là chỉ số của một mục trong bảng phân trang, mục này chứa chỉ số frame trong bộ nhớ thực
- *offset:* độ dời của vị trí nhớ trong frame nói trên.

Segmentation with paging

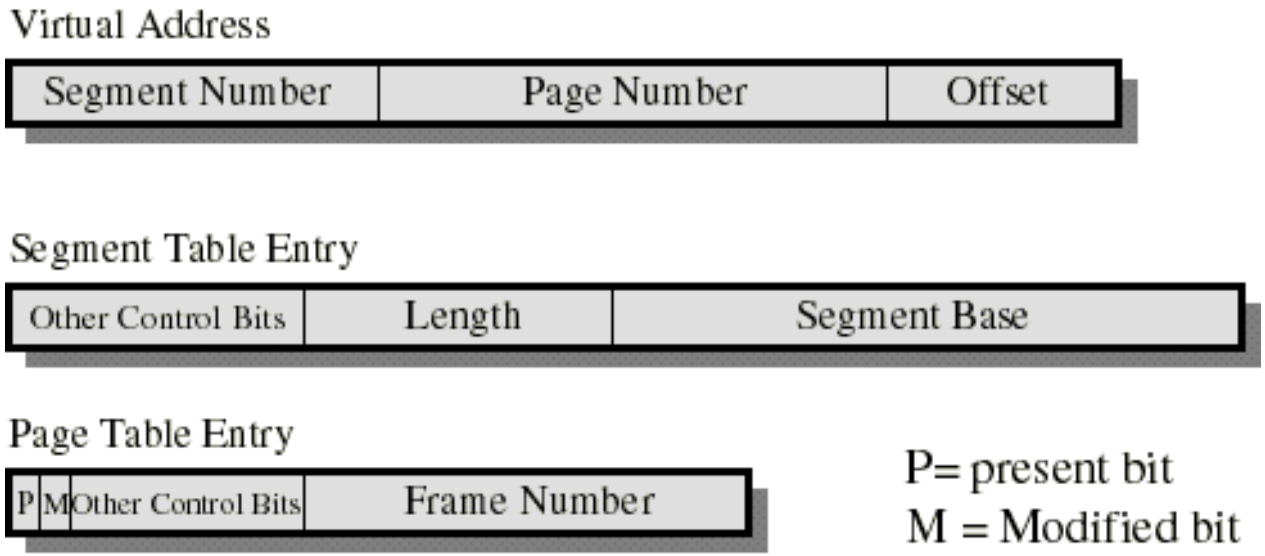
- Minh họa



Segmentation with paging (tt.)



Segmentation with paging (tt.)



- Segment base: địa chỉ thực của bảng phân trang
- Present bit và modified bit chỉ tồn tại trong bảng phân trang
- Các thông tin bảo vệ và chia sẻ vùng nhớ thường nằm trong bảng phân đoạn
  - Ví dụ: read-only/read-write bit,…