Quản lý Bộ nhớ
Bộ nhớ ảo

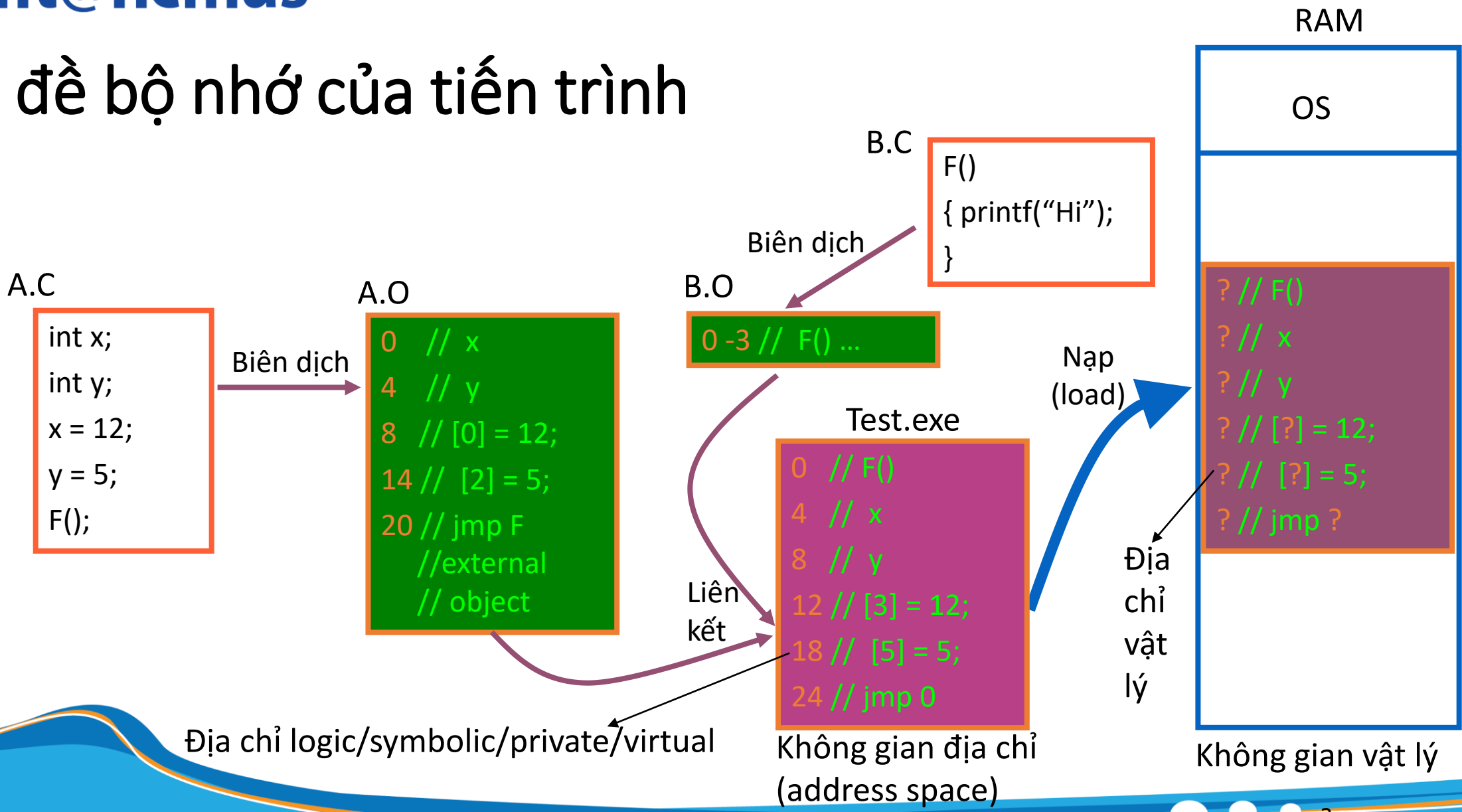
Môn học: Hệ điều hành

fit@hcmus
KHOA CÔNG NGHỆ THÔNG TIN
TRƯỜNG ĐẠI HỌC KHOA HỌC TỰ NHIÊN

Nội dung

- Tổng quan về quản lý bộ nhớ
- Vấn đề chuyển đổi địa chỉ
- Mô hình tổ chức bộ nhớ
  - Mô hình liên tục
  - Mô hình không liên tục
- Bộ nhớ ảo
- Tổ chức bộ nhớ Intel x86

Vấn đề bộ nhớ của tiến trình

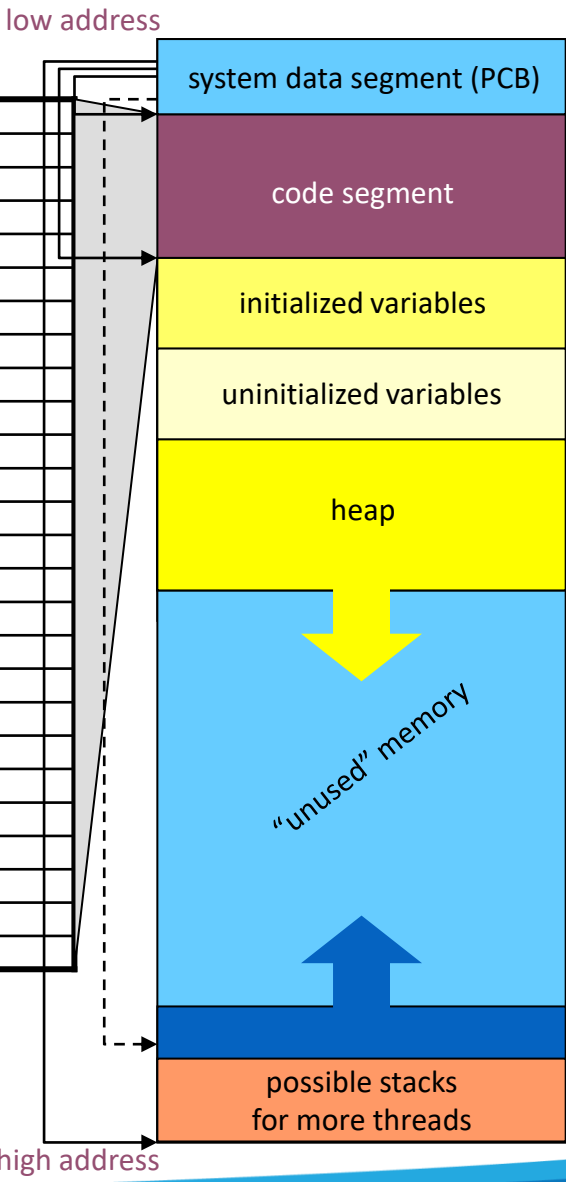


fit@hcmus

Nhu cầu bộ nhớ của tiến trình

- Code segment:
  - Read from program file by exec.
  - Usually read-only.
  - Can be shared.
- Data segment:
  - Initialized global variables (0 / NULL).
  - Uninitialized global variables.
  - Heap:
    - Dynamic memory.
    - E.g., allocated using malloc.
    - Grows against higher addresses.
- Stack segment:
  - Variables in a function.
  - Stored register states (e.g, calling function EIP).
  - Grows against lower addresses.
- System data segment (PCB).
  - Segment pointers.
  - Pid.
  - Program and stack pointers.
  - …
- Stack cho các thread.

```
…
8048314 <add>:
8048314:        push   %ebp
8048315:        mov    %esp,%ebp
8048317:        mov    0xc(%ebp),%eax
804831a:        add    0x8(%ebp),%eax
804831d:        pop    %ebp
804831e:        ret
804831f <main>:
804831f:        push   %ebp
8048320:        mov    %esp,%ebp
8048322:        sub    $0x18,%esp
8048325:        and    $0xfffffff0,%esp
8048328:        mov    $0x0,%eax
804832d:        sub    %eax,%esp
804832f:        movl   $0x0,0xfffffffc(%ebp)
8048336:        movl   $0x2,0x4(%esp,1)
804833e:        movl   $0x4,(%esp,1)
8048345:        call   8048314 <add>
804834a:        mov    %eax,0xfffffffc(%ebp)
804834d:        leave
804834e:        ret
804834f:        nop
…
…
…
```



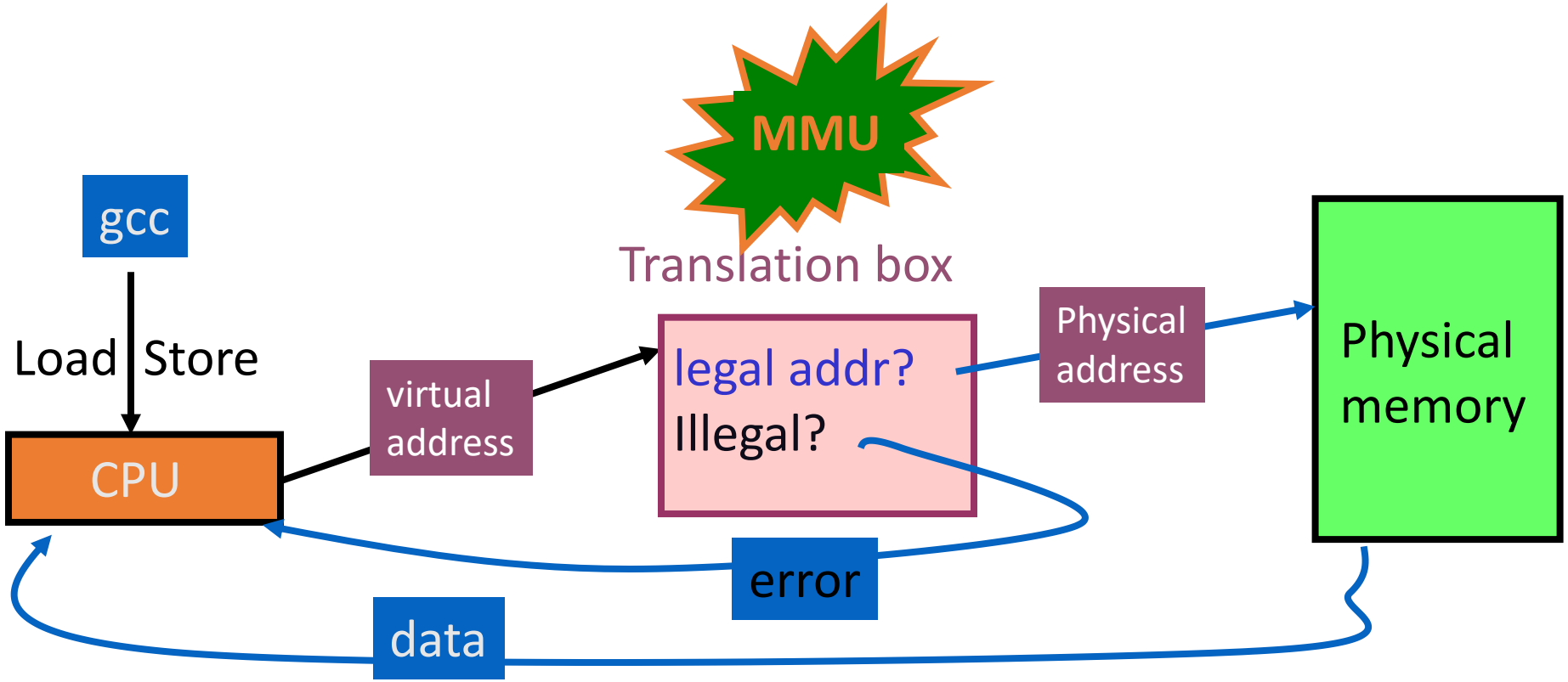
Vấn đề quản lý bộ nhớ

- Chương trình cần được nạp vào Bộ nhớ chính để thi hành.
  - CPU chỉ có thể truy xuất trực tiếp Main Memory.
  - Chương trình khi được nạp vào BNC sẽ được tổ chức theo cấu trúc của tiến trình tương ứng.
- Cấp phát Bộ nhớ:
  - BNC giới hạn, N tiến trình ?
  - Bảo vệ ? Chia sẻ ?
  - Tiến trình thay đổi kích thước ?
  - Tiến trình lớn hơn BNC ?
- Chuyển đổi địa chỉ tiến trình (address binding):
  - Thời điểm chuyển đổi địa chỉ ?
  - Công thức chuyển đổi ?
    - Phụ thuộc vào Mô hình tổ chức BNC ?
    - Cần sự hỗ trợ của phần cứng ?
  - Tiến trình thay đổi vị trí trong BNC ?

Chuyển đổi địa chỉ (address binding)

- Compile&Link-time:
  - Phát sinh địa chỉ vật lý.
  - Phải biết trước vị trí nạp chương trình.
  - Phải biên dịch lại chương trình khi vị trí nạp thay đổi.
- Load-time:
  - Khi biên dịch chỉ phát sinh địa chỉ cục bộ của chương trình.
  - Khi nạp, biết vị trí bắt đầu sẽ tính lại địa chỉ vật lý.
  - Phải tái nạp khi vị trí bắt đầu thay đổi.
- Execution-time:
  - Khi biên dịch, nạp chỉ phát sinh địa chỉ cục bộ của chương trình.
  - Trì hoãn thời điểm kết buộc địa chỉ tuyệt đối đến khi thi hành.
  - Khi đó, ai tính toán địa chỉ vật lý ?
    - Phần cứng : MMU (trong CPU chip).

Mô hình chuyển đổi địa chỉ



Mô hình tổ chức bộ nhớ

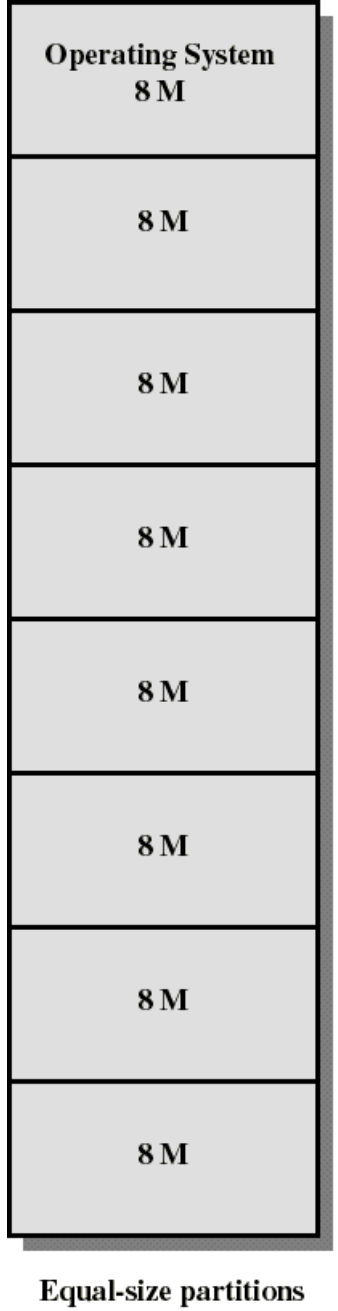
- Cấp phát liên tục (Contigous Allocation).
  - Fixed Partitioning.
  - Dynamic Partitioning.
- Cấp phát không liên tục (Non Contigous Allocation).
  - Segmentation.
  - Paging.

Cấp phát liên tục

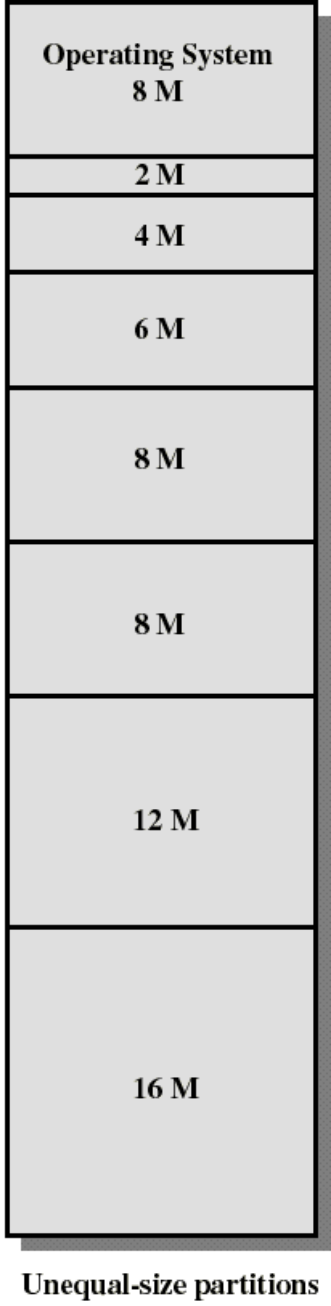
- Nguyên tắc :
  - Chương trình được nạp toàn thể vào BNC để thi hành.
  - Cần một vùng nhớ liên tục, đủ lớn để chứa Chương trình.
- Không gian địa chỉ: liên tục.
- Không gian vật lý: có thể tổ chức
  - Fixed partition.
  - Variable partition.
- Mô hình chuyển đổi địa chỉ:
  - Linker – Loader.
  - Base & Bound.

Cấp phát liên tục – Fixed Partitioning

- Phân chia KGVL thành các partitions.
- Có 2 cách phân chia partitions:
  - Kích thước bằng nhau.
  - Kích thước khác nhau.
- Mỗi tiến trình sẽ được nạp vào một partition để thi hành.
- Chiến lược cấp phát partition ?



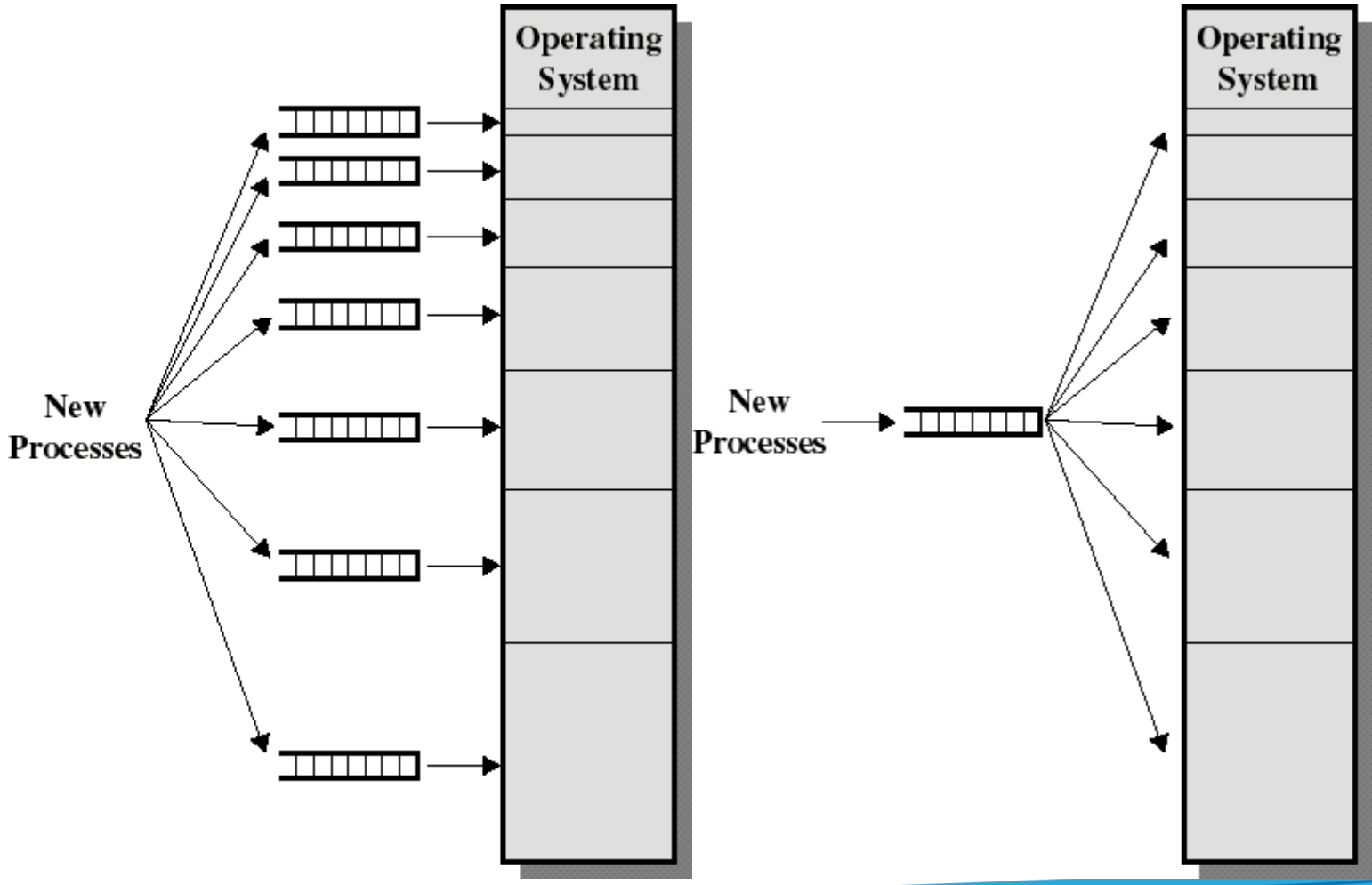
Equal-size partitions



Unequal-size partitions

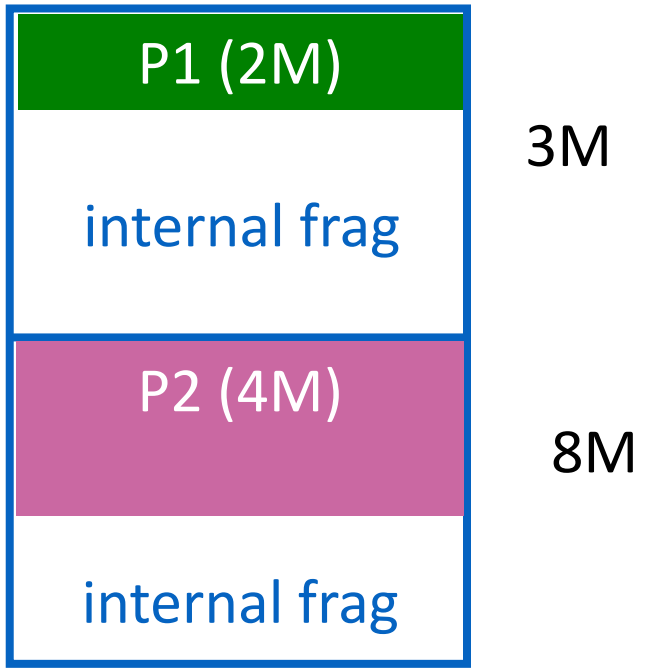
Chiến lược cấp phát partitions không bằng nhau

- Sử dụng nhiều hàng đợi:
  - Cấp cho tiến trình partition với kích thước bé nhất (đủ lớn để chứa tiên trình).
  - Khuyết điểm : phân bố các tiến trình vào các partition không đều, một số tiến trình phải đợi trong khi có partition khác trống.
- Sử dụng 1 hàng đợi:
  - Cấp cho tiến trình partition tự do với kích thước bé nhất (đủ lớn để chứa tiên trình).
  - Cần dùng một CTDL để theo dõi các partition tự do.



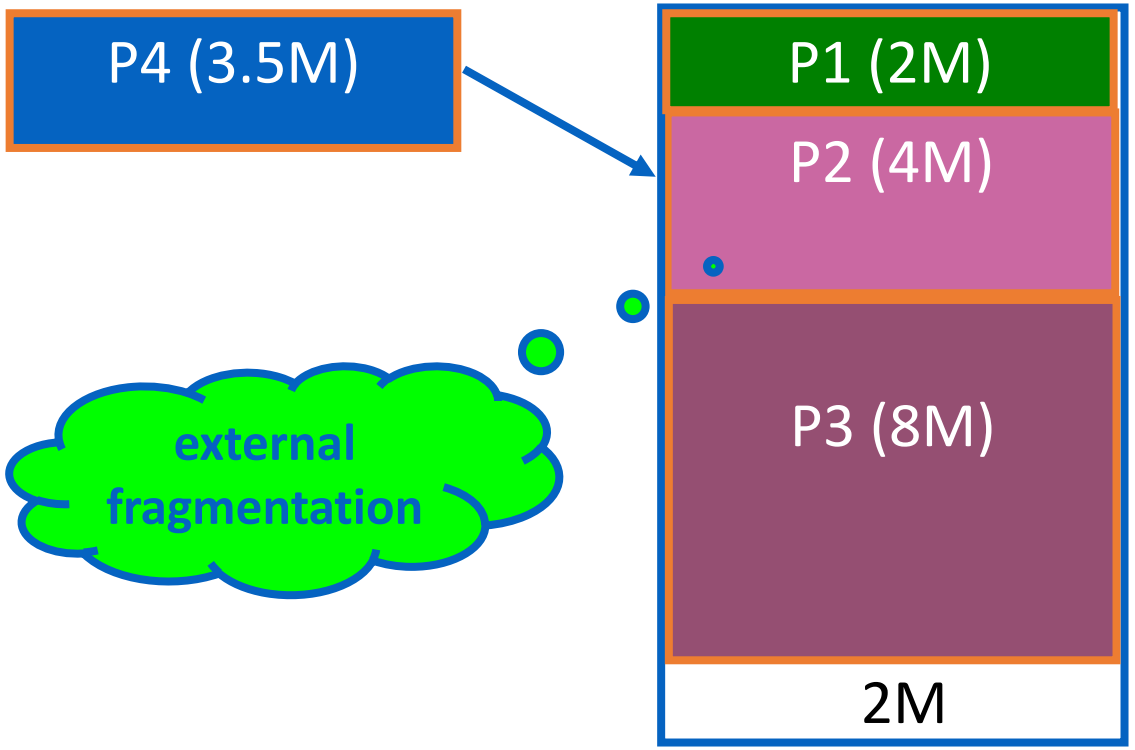
Fixed Partitioning – Nhận xét

- Sử dụng BN không hiệu quả.
  - Internal fragmentation : kích thước chương trình không đúng bằng kích thước partition.
- Mức độ đa chương của hệ thống (Số tiến trình được nạp) bị giới hạn bởi số partitions.



Cấp phát liên tục – Dynamic Partitioning

- BNC không được phân chia trước.
  - Các partition có kích thước tùy ý, sẽ hình thành trong quá trình nạp các tiến trình vào hệ thống.
- Mỗi tiến trình sẽ được cấp phát đúng theo kích thước yêu cầu.
  - Không còn internal fragmentation.
- Tiến trình vào sau không lấp đầy chỗ trống tiến trình trước để lại
  - External fragmentation.



Dynamic Partitioning – Quản lý vùng nhớ

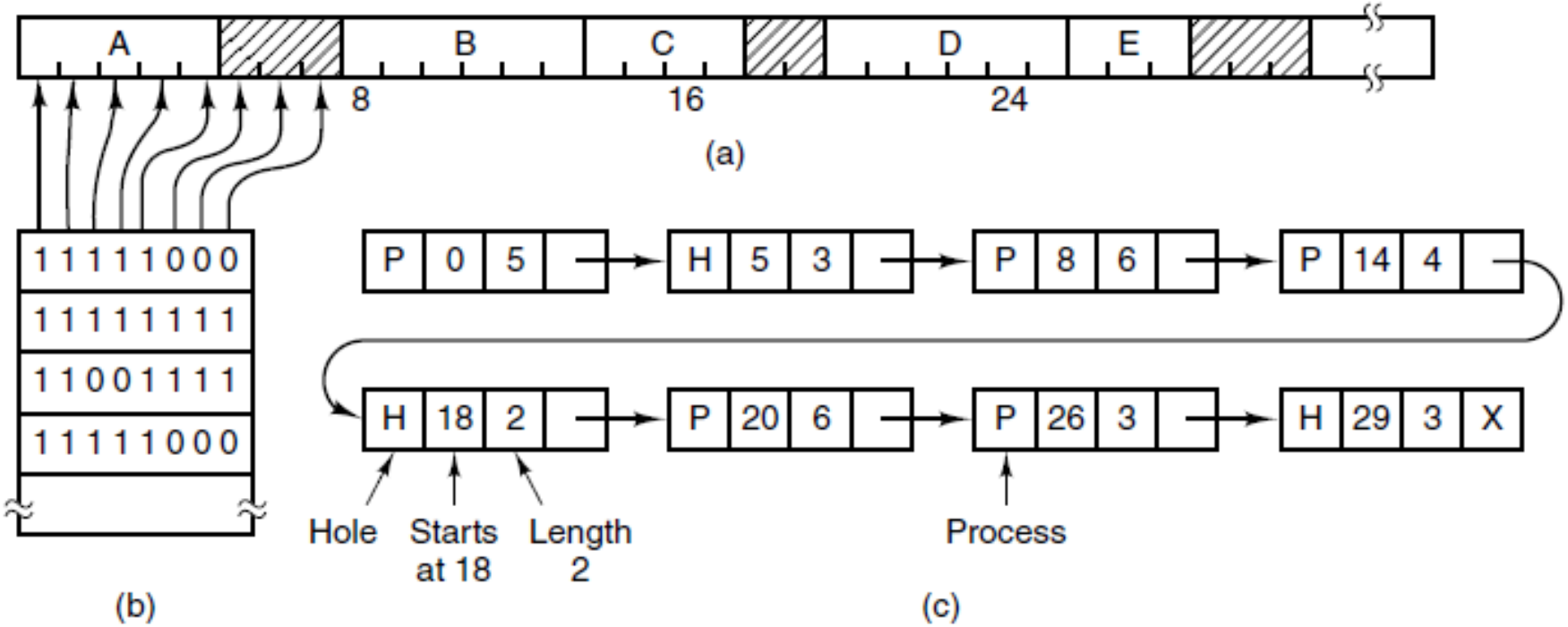
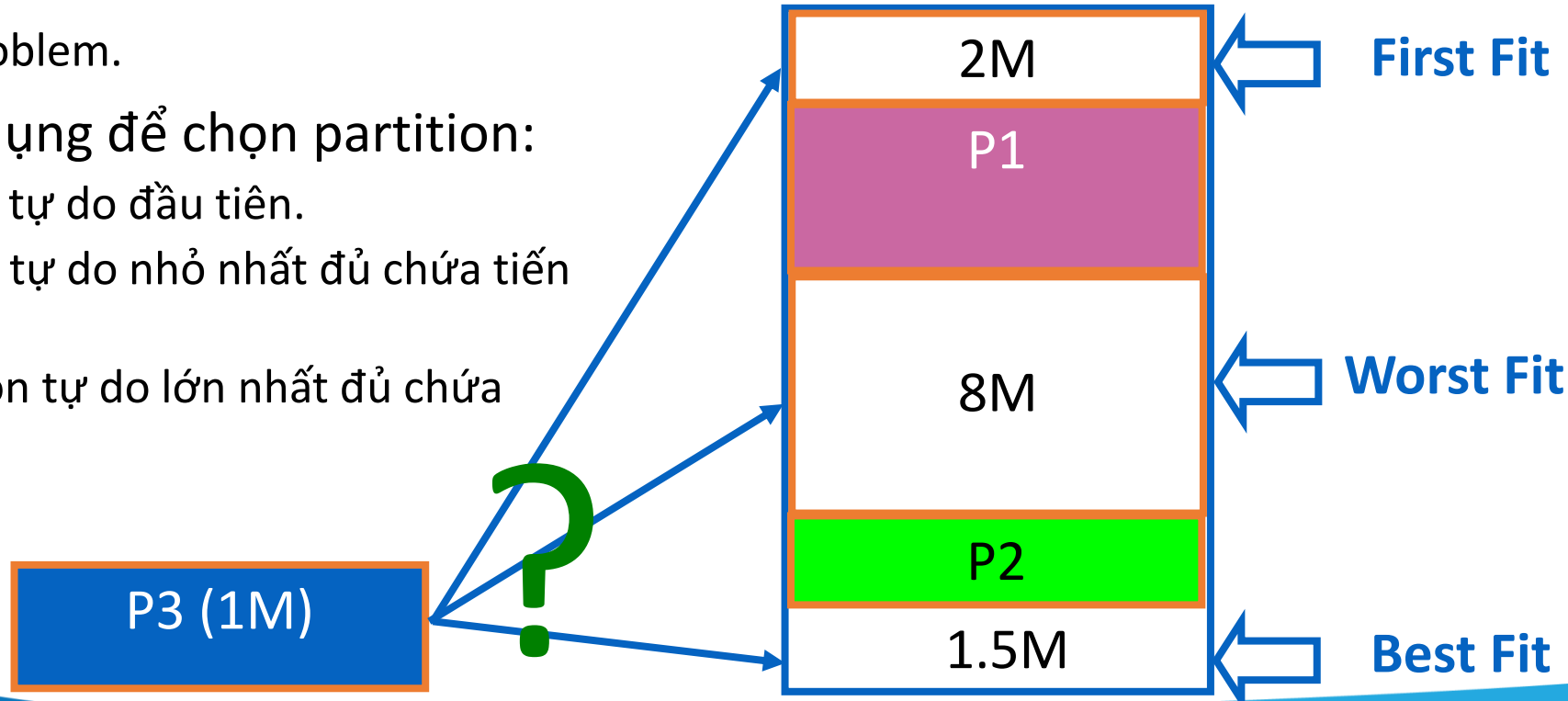


Figure 3-6. (a) A part of memory with five processes and three holes. The tickmarks show the memory allocation units. The shaded regions (0 in the bitmap) are free. (b) The corresponding bitmap. (c) The same information as a list.

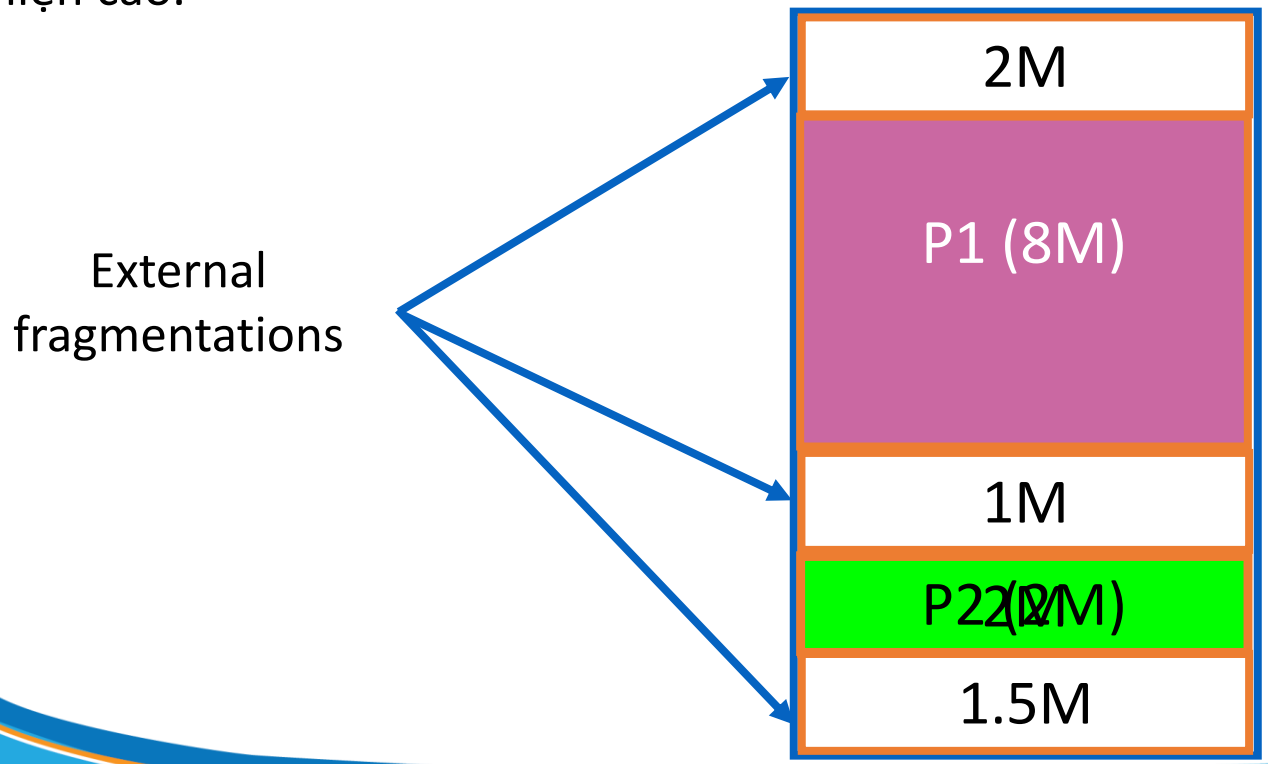
Dynamic Partitioning – Thuật toán lựa chọn vùng trống

- Chọn lựa partition để cấp phát cho tiến trình ?
  - Đồng thời có nhiều partition tự do đủ lớn để chứa tiến trình.
  - Dynamic Allocation problem.
- Các chiến lược thông dụng để chọn partition:
  - First-fit: chọn partition tự do đầu tiên.
  - Best-fit: chọn partition tự do nhỏ nhất đủ chứa tiến trình.
  - Worst-fit: chọn partition tự do lớn nhất đủ chứa tiến trình.



Dynamic Partitioning – Memory Compaction (Garbage Collection)

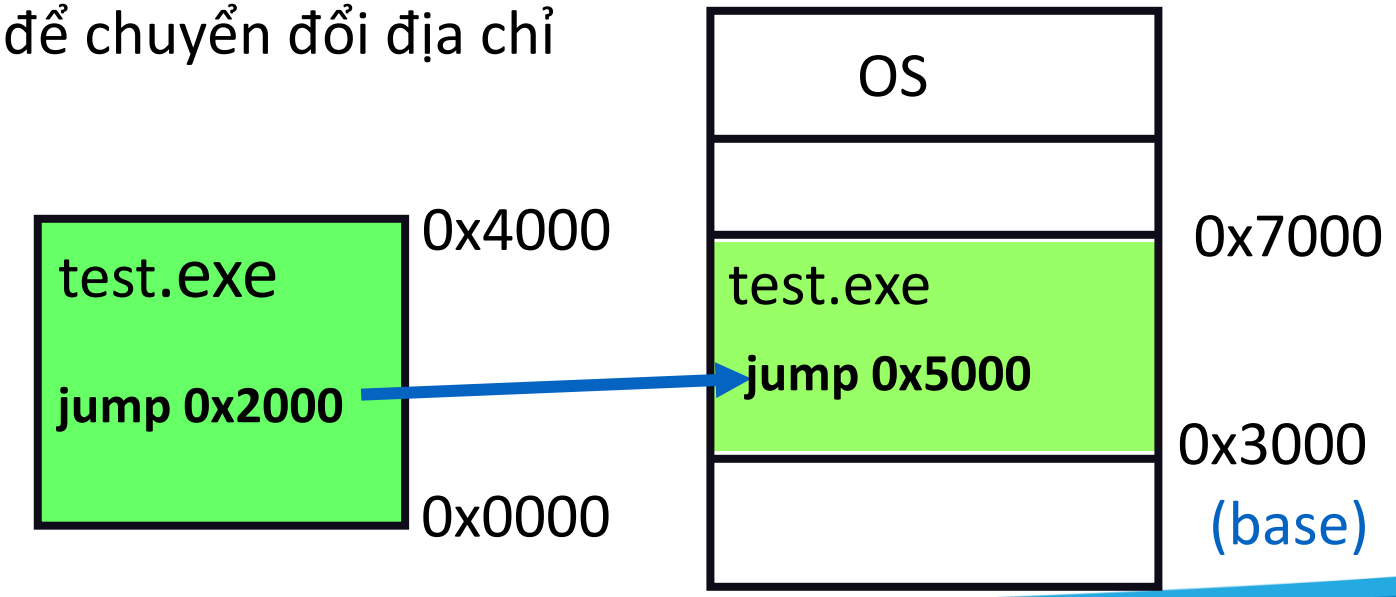
- Giải quyết vấn đề External Fragmentation:
  - Dồn các vùng bị phân mảnh lại với nhau để tạo thành partition liên tục đủ lớn để sử dụng.
  - Chi phí thực hiện cao.



Chuyển đổi địa chỉ – Link-Loader

- Tại thời điểm Link, giữ lại các địa chỉ logic.
- Vị trí base của tiến trình trong bộ nhớ xác định được vào thời điểm nạp:

  địa chỉ physic = địa chỉ logic + base
- Nhận xét:
  - Không cần sự hỗ trợ phần cứng để chuyển đổi địa chỉ
    - Loader thực hiện
  - Bảo vệ ?
    - Không hỗ trợ
  - Dời chuyển sau khi nạp ?
    - Không hỗ trợ tái định vị
    - Phải nạp lại !



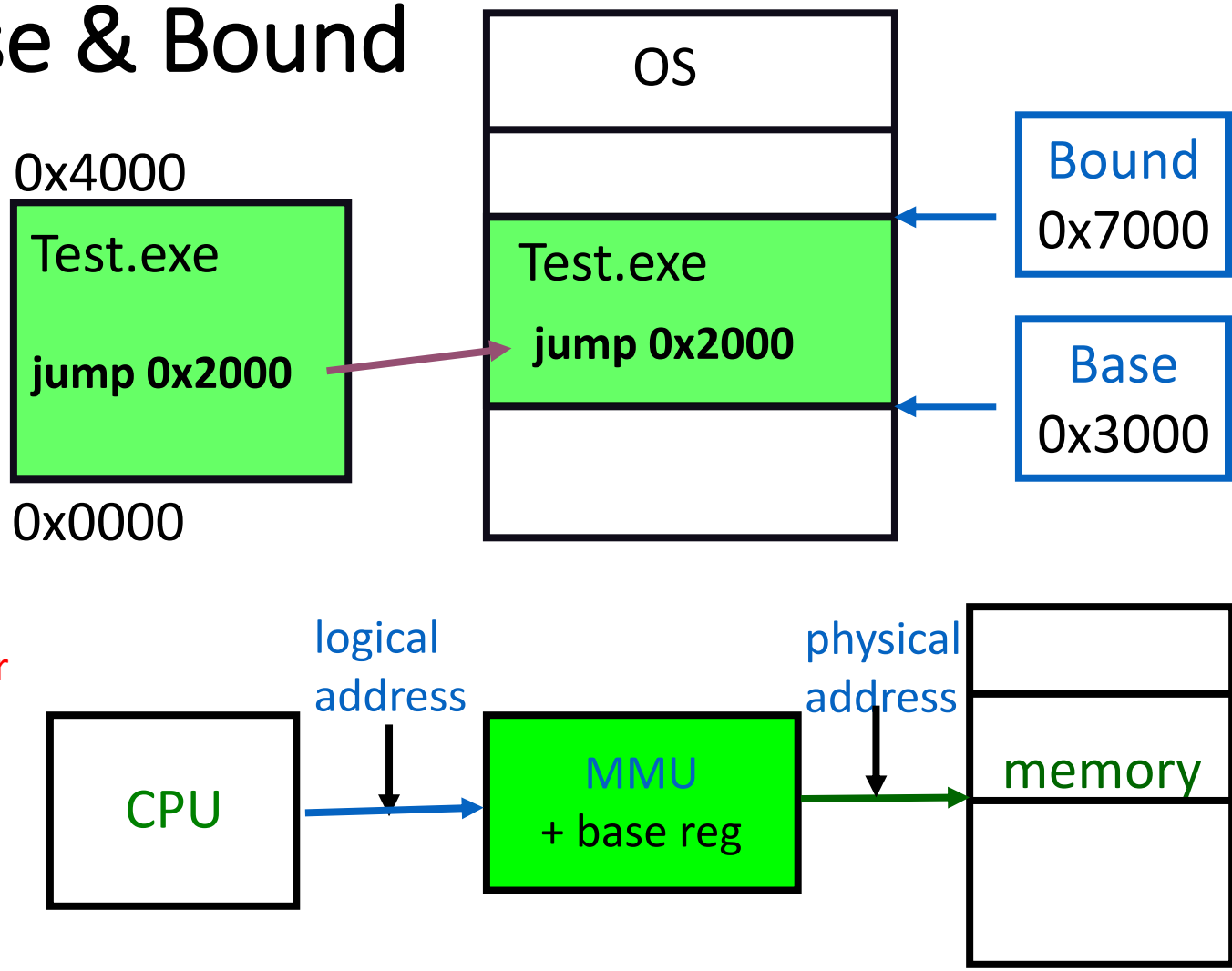
Chuyển đổi địa chỉ – Base & Bound

- Tại thời điểm Link, giữ lại các địa chỉ logic.
- Vị trí base, bound được ghi nhận vào 2 thanh ghi.
- Kết buộc địa chỉ vào thời điểm thi hành.

  => Tái định vị được.

  địa chỉ physic = địa chỉ logic + base register

  => Cần sự hỗ trợ của phần cứng.
- Bảo vệ:
  - Địa chỉ hợp lệ [base, bound]



Cấp phát liên tục – Nhận xét

- Không có vùng nhớ liên tục đủ lớn để nạp tiến trình ?
  - Bó tay ...
  - Sử dụng BNC không hiệu quả !

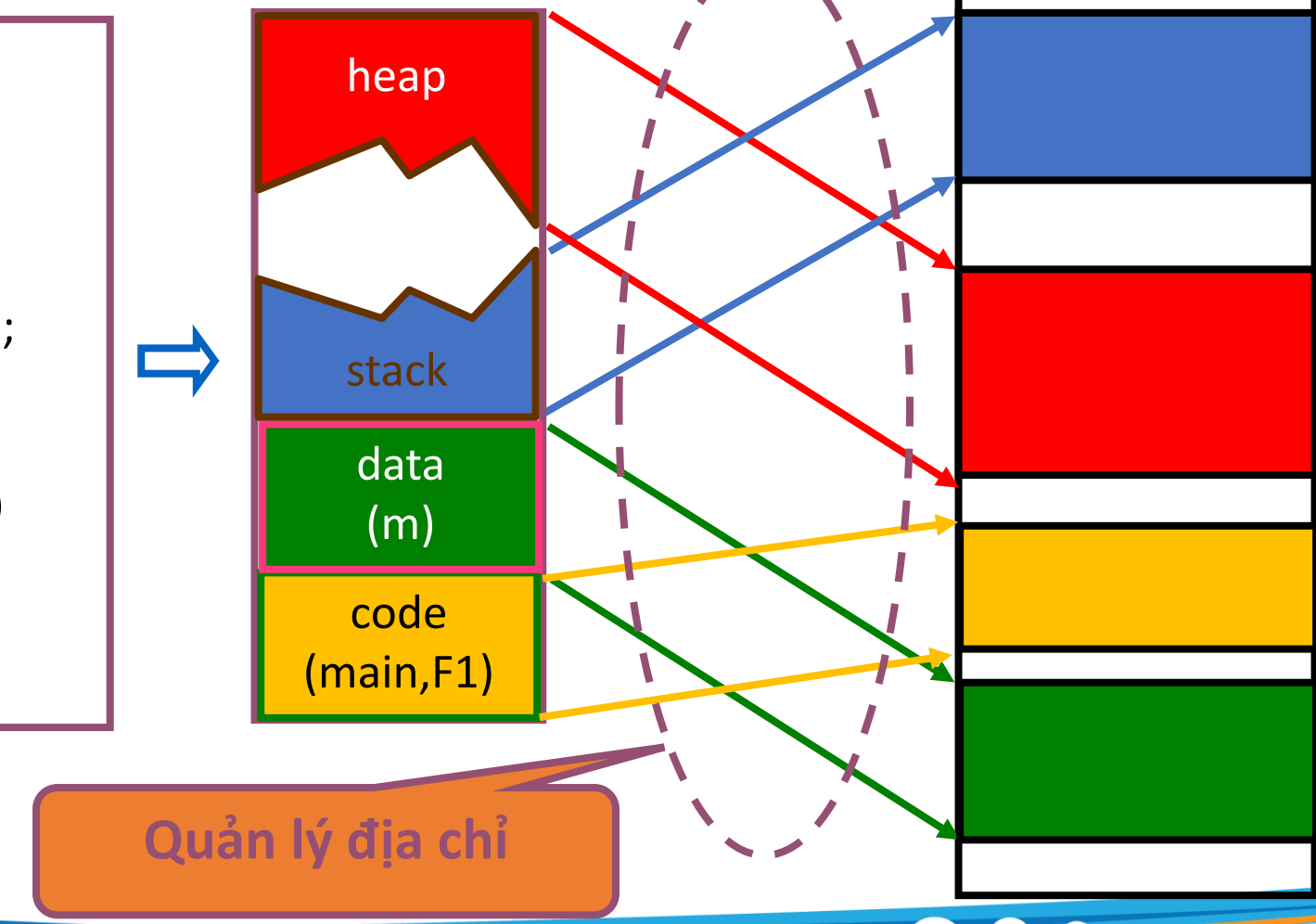
Cấp phát không liên tục

- Cho phép nạp tiến trình vào BNC ở nhiều vùng nhớ không liên tục.
- Không gian địa chỉ: phân chia thành nhiều partition.
  - Phân đoạn (Segmentation).
  - Phân trang (Paging).
- Không gian vật lý: có thể tổ chức
  - Fixed partition : Paging
  - Variable partition : Segmentation

Cấp phát không liên tục – Segmentation

- KGĐC: phân chia thành các segment.
- KGVL: tổ chức thành dynamic partitions.
- Nạp tiến trình:
  - Mỗi segment cần được nạp vào một partition liên tục, tự do, đủ lớn cho segment.
    - partition nào ? ...Dynamic Allocation !
  - Các segment của cùng 1 chương trình có thể được nạp vào những partition không liên tục.

```
int m;

main ()

{

    F1(m);

}

F1(int x)

{   x = 9;

}
```

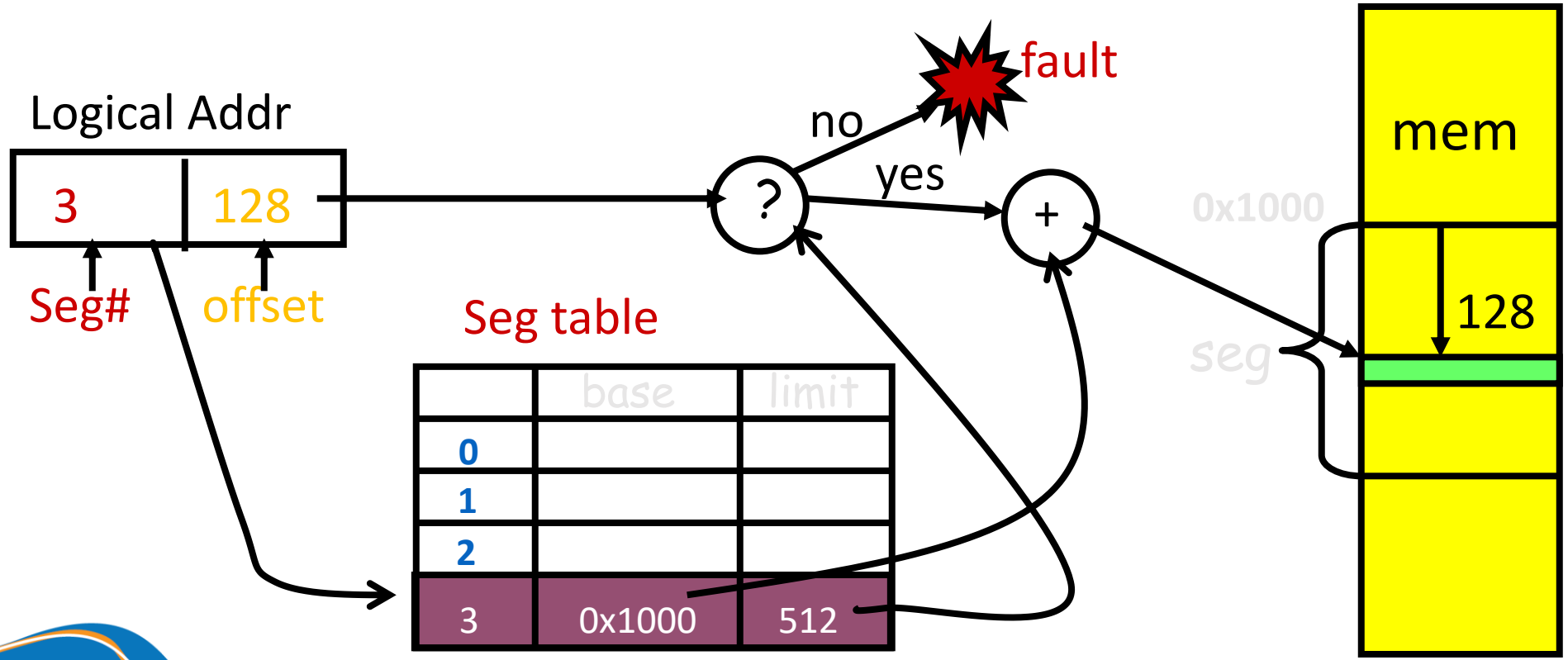


Quản lý địa chỉ

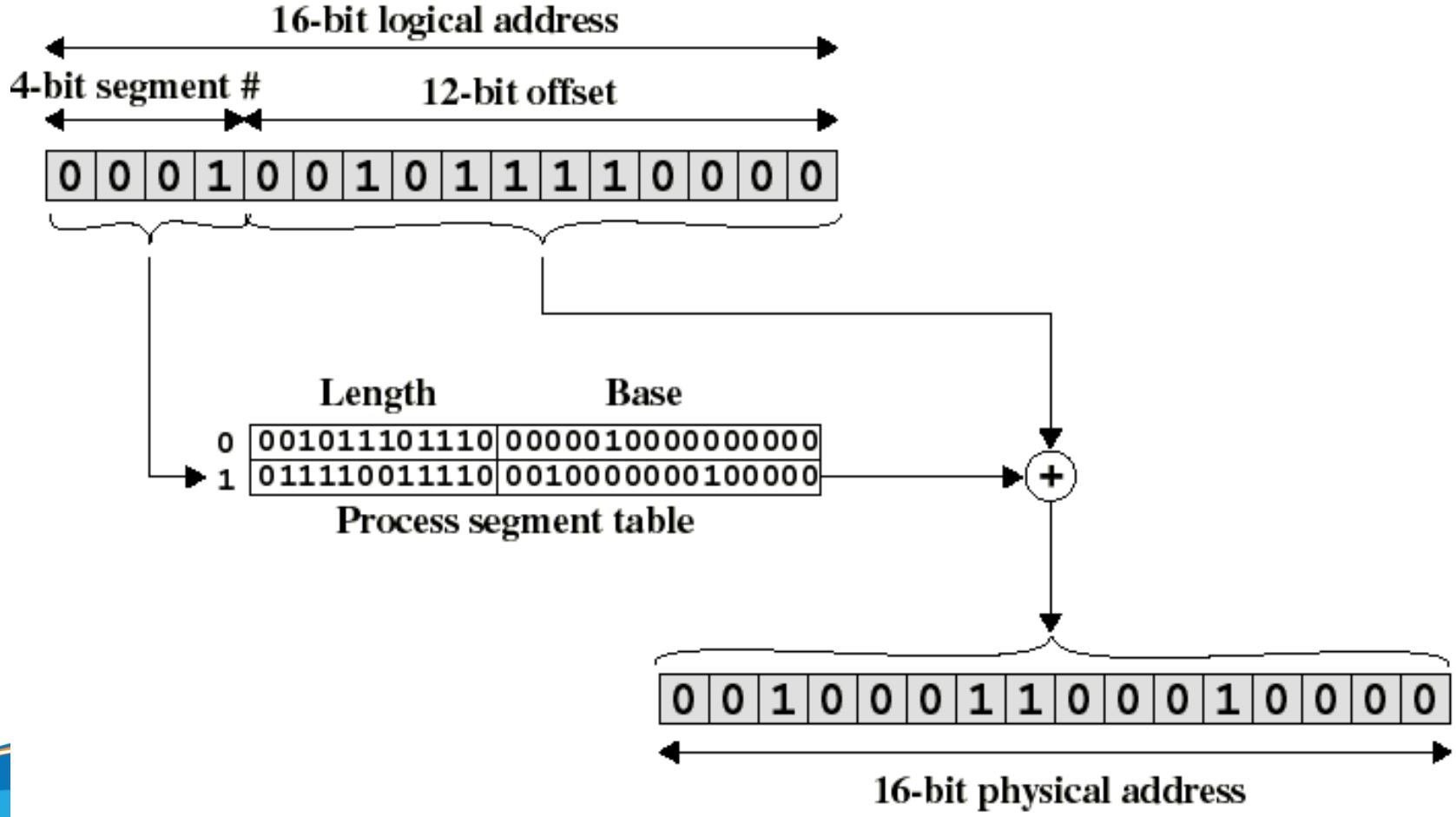
Segmentation – Tổ chức

- Địa chỉ logic : <segment-number, offset>
- Địa chỉ physic : <real address>
- Chuyển đổi địa chỉ : <s,d> <r>
- Chuyển đổi địa chỉ vào thời điểm thi hành.
  - MMU thi hành.
  - Sử dụng Segment Table (bảng phân đoạn) để lưu thông tin cấp phát BNC, làm cơ sở thực hiện ánh xạ địa chỉ.
  - Mỗi tiến trình có một Segment Table.
- Segment Table:
  - Số phần tử của Segment Table = Số Segment của chương trình
  - Mỗi phần tử của Segment Table mô tả cho 1 segment, và có cấu trúc :
    - base: địa chỉ vật lý trong BNC của partition chứa segment.
    - limit : kích thước segment.
  - Lưu trữ Segment Table ?
    - Cache : nếu đủ nhỏ.
    - BNC : Segment-table base register (STBR), Segment-table length register (STLR).

Segmentation – Mô hình chuyển đổi địa chỉ



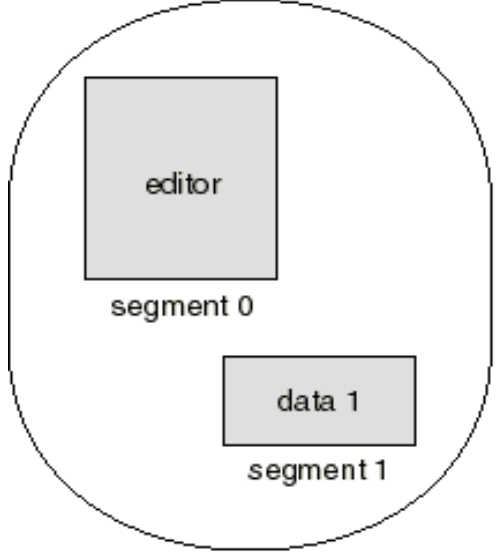
Segmentation – Chuyển đổi địa chỉ



Segmentation – Nhận xét

- Cấp phát không liên tục => tận dụng bộ nhớ hiệu quả.
- Hỗ trợ tái định vị.
  - Từng Segment.
- Hỗ trợ Bảo vệ và Chia sẻ được ở mức module.
  - Ý nghĩa của “mức module” ?
- Chuyển đổi địa chỉ phức tạp.
  - Đã có MMU ...
- Sử dụng dynamic partition: chịu đựng.
  - Dynamic Allocation : chọn vùng nhớ để cấp cho một segment.
    - First fit, Best fit, Worst fit.
  - External Fragmentation:
    - Memory Compaction: chi phí cao.

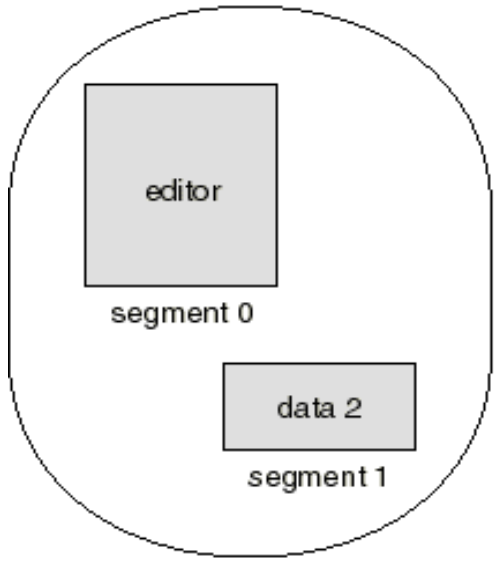
Segmentation – Chia sẻ



logical address space
process P_1

	limit	base
0	25286	43062
1	4425	68348

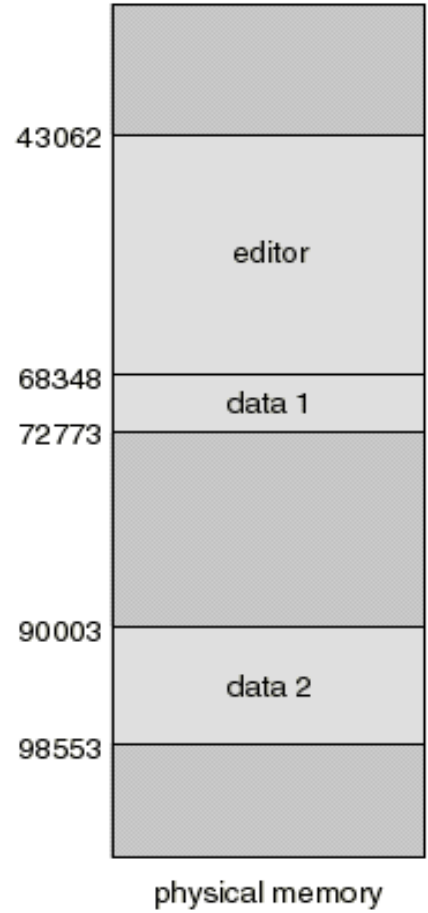
segment table
process P_1



logical address space
process P_2

	limit	base
0	25286	43062
1	8850	90003

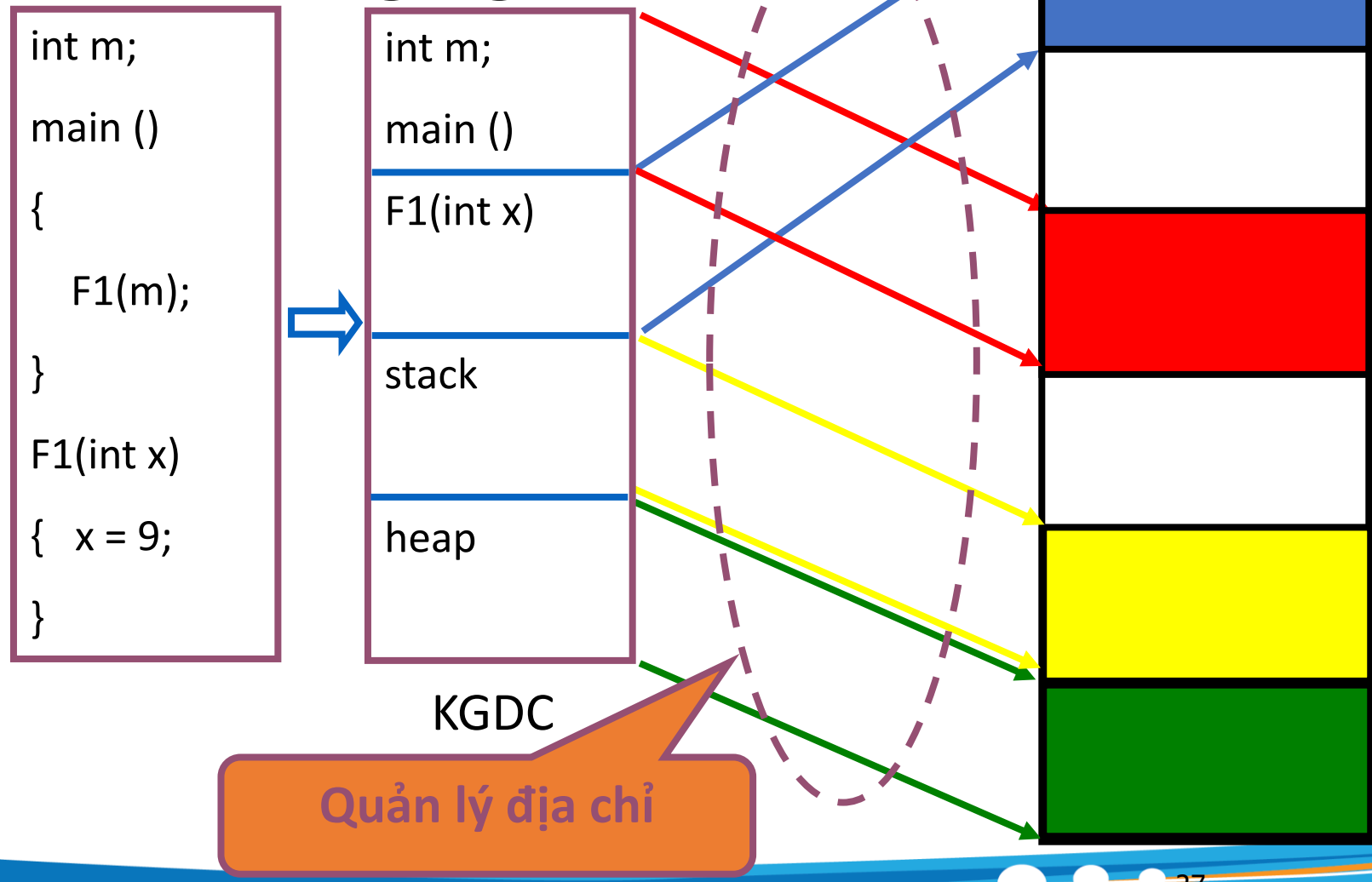
segment table
process P_2



physical memory

Cấp phát không liên tục – Paging

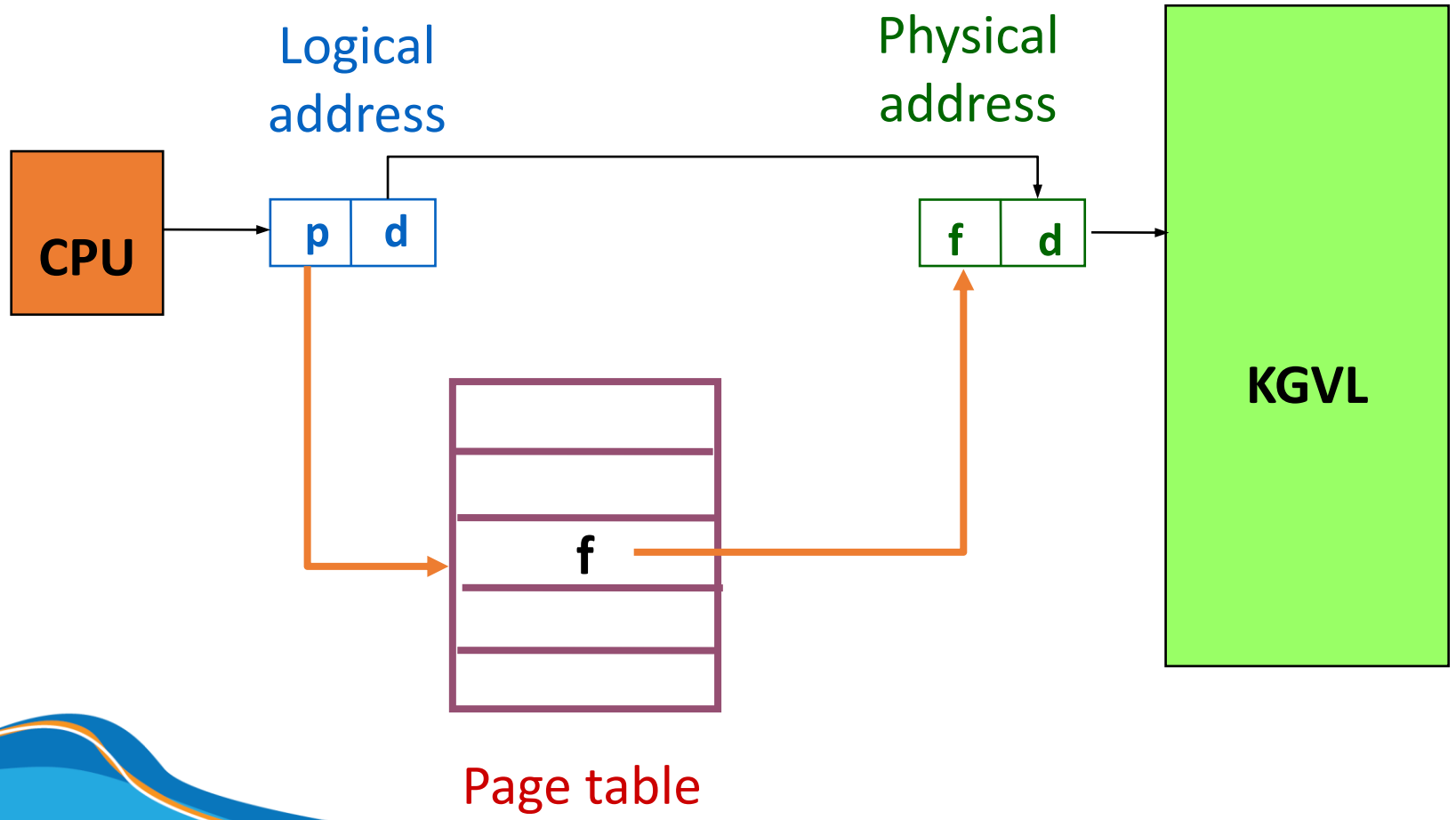
- Hỗ trợ HĐH khắc phục bài toán cấp phát bộ nhớ động, và loại bỏ external fragmentation.
- KGĐC : phân chia chương trình thành các page có kích thước bằng nhau.
  - Không quan tâm đến ngữ nghĩa của các đối tượng nằm trong page.
- KGVL : tổ chức thành các fixed partitions có kích thước bằng nhau gọi là frame.
- page size = frame size
- Nạp tiến trình :
  - Mỗi page cần được nạp vào một frame tự do.
  - Các pages của cùng 1 chương trình có thể được nạp vào những frames không kế cận nhau.
  - Tiến trình kích thước N pages -> cần N frame tự do để nạp.



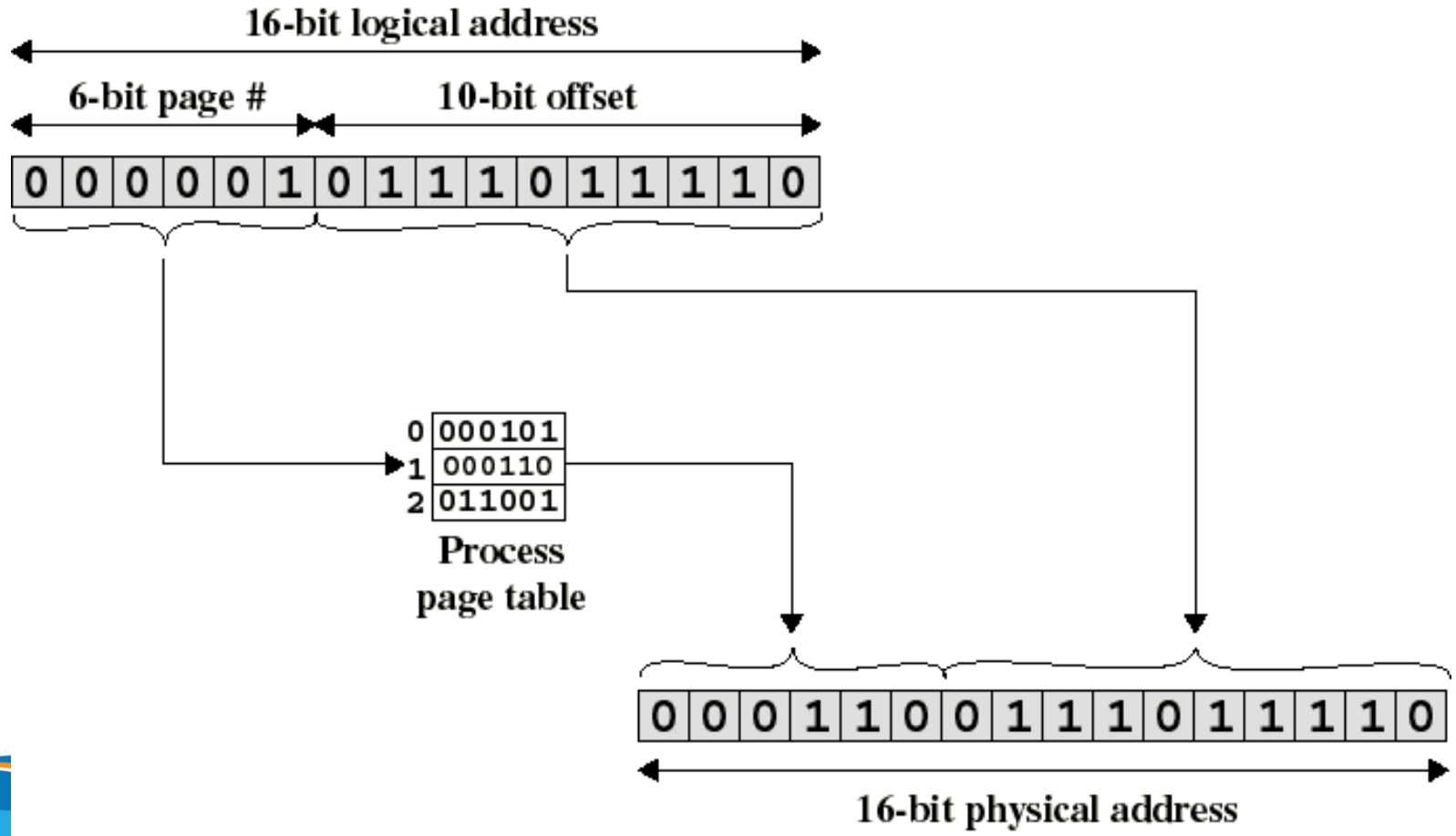
Paging – Tổ chức

- Địa chỉ logic : <page-number, offset>
- Địa chỉ physic : <frame-number, offset>
- Chuyển đổi địa chỉ : <p,d> <f,d>
- Chuyển đổi địa chỉ vào thời điểm thi hành.
  - MMU thi hành.
  - Sử dụng Page Table để lưu thông tin cấp phát BNC, làm cơ sở thực hiện ánh xạ địa chỉ.
  - Mỗi tiến trình có một Page Table.
- Page Table.
  - Số phần tử của Page Table = Số Page trong KGĐC của chương trình.
  - Mỗi phần tử của bảng Page Table mô tả cho 1 page, và có cấu trúc:
    - frame: số hiệu frame trong BNC chứa page.
  - Lưu trữ Page Table ?
    - Cache : không đủ.
    - BNC : Page-table base register (PTBR), Page-table length register (PTLR).

Paging – Mô hình chuyển đổi địa chỉ



Paging – Chuyển đổi địa chỉ



fit@hcmus

Paging – Nhận xét

- Loại bỏ:
  - Dynamic Allocation.
  - External Fragmentation.
- "Trong suốt" với LTV.
- Hỗ trợ Bảo vệ và Chia sẻ ở mức page.
- Internal Fragmentation.
- Lưu trữ Page Table trong bộ nhớ:
  - Tốn chỗ.
  - Những page không sử dụng: lãng phí.
  - Tăng thời gian chuyển đổi địa chỉ.

- Giả sử hệ thống sử dụng m bit địa chỉ.
  - Size of KGĐC = 2^m
- Kích thước page:
  - Trên nguyên tắc tùy ý, thực tế chọn pagesize = 2^n
- Số trang trong KGĐC: #pages = 2^m / 2^n = 2^{m-n}
  - Ví dụ : 32-bits địa chỉ, pagesize = 4K
  - KGĐC = 232 -> #pages= 232-212 = 220 = 1.000.000 pages !
  - #pages = #entry trong PT.
- Địa chỉ logic :

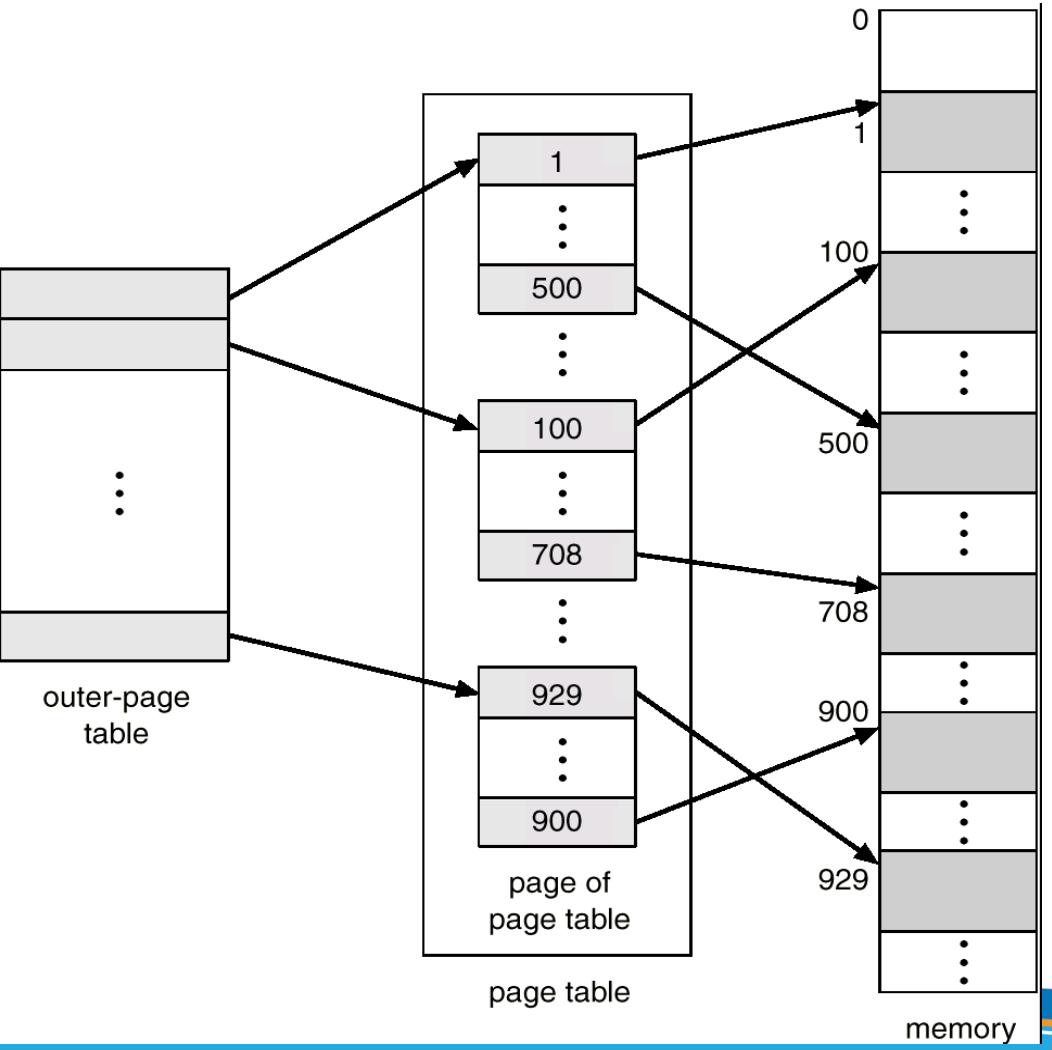
- Page Table:
  - Mỗi tiến trình lưu 1 Page Table.
  - Số lượng phần tử quá lớn -> Lưu BNC.
  - Mỗi truy xuất địa chỉ sẽ tốn 2 lần truy xuất BNC.

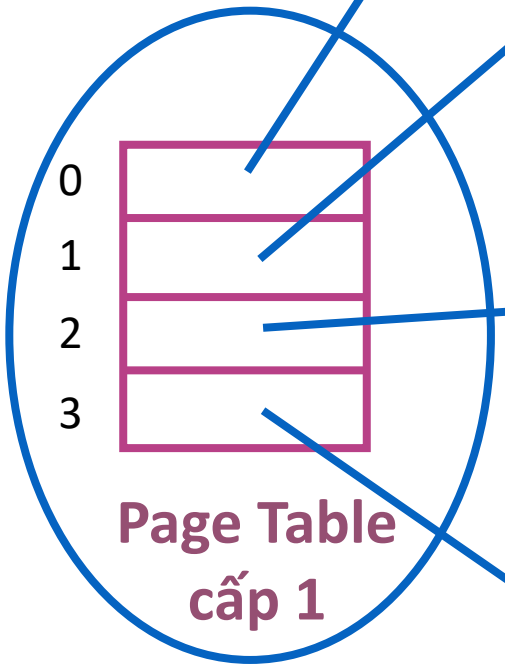
Paging – Tổ chức page table tiết kiệm không gian

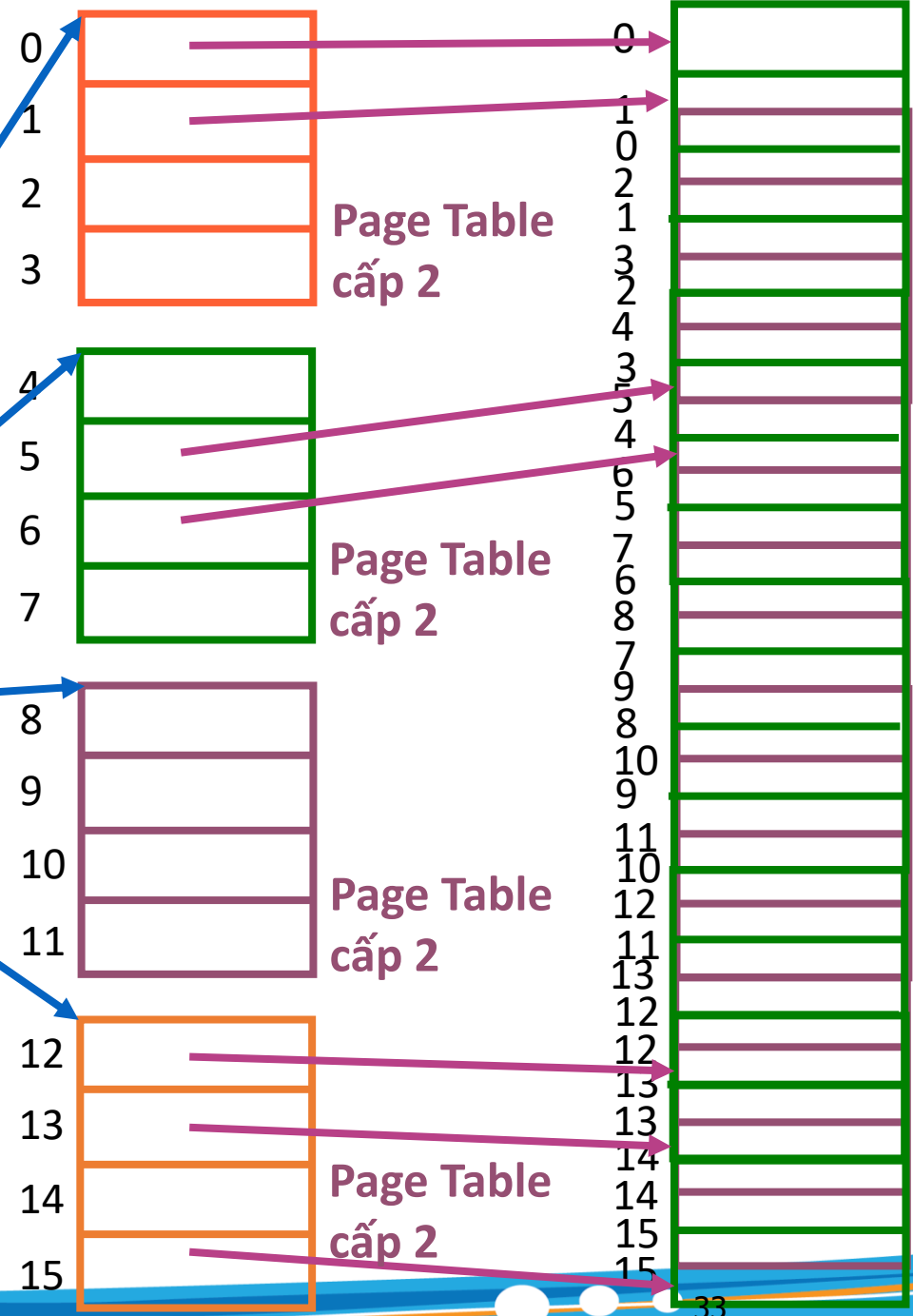
- Sử dụng bảng trang đa cấp
  - Chia bảng trang thành các phần nhỏ, bản thân bảng trang cũng sẽ được phân trang
  - Chỉ lưu thường trực bảng trang cấp 1, sau đó khi cần sẽ nạp bảng trang cấp nhỏ hơn thích hợp...
  - Có thể loại bỏ những bảng trang chứa thông tin về miền địa chỉ không sử dụng
- Sử dụng Bảng trang nghịch đảo
  - Mô tả KGVL thay vì mô tả KGĐC -> 1 IPT cho toàn bộ hệ thống

fit@hcmus

Page table – Bảng trang đa cấp



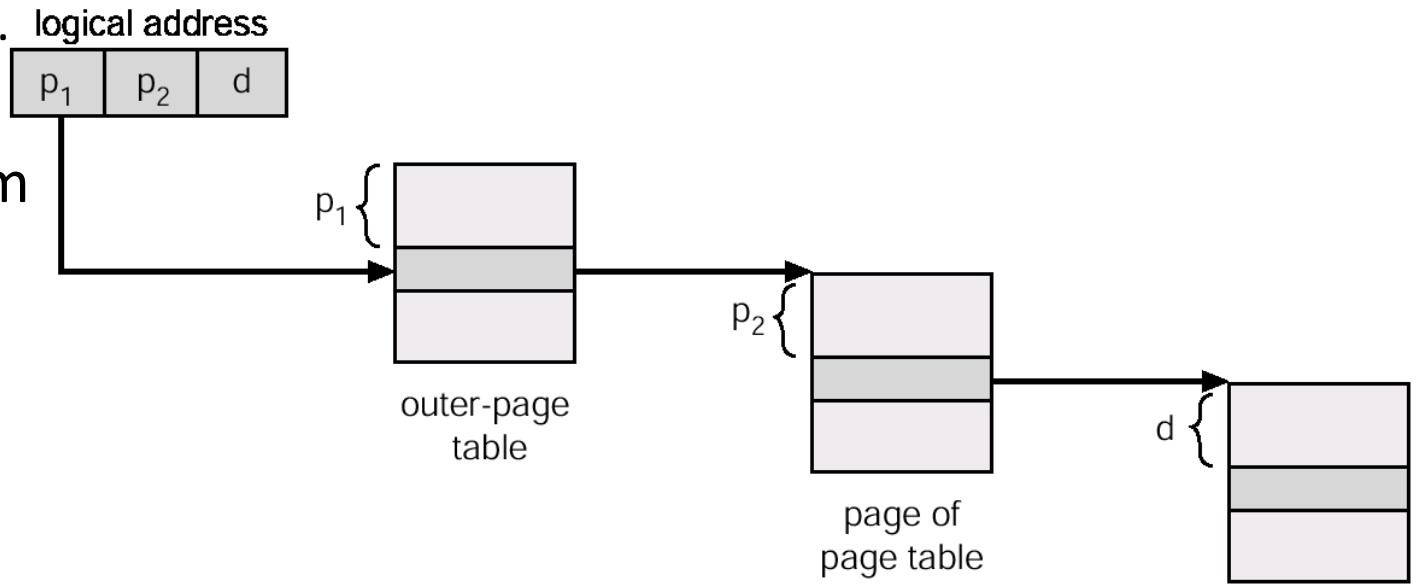




Ví dụ mô hình bảng trang 2 cấp

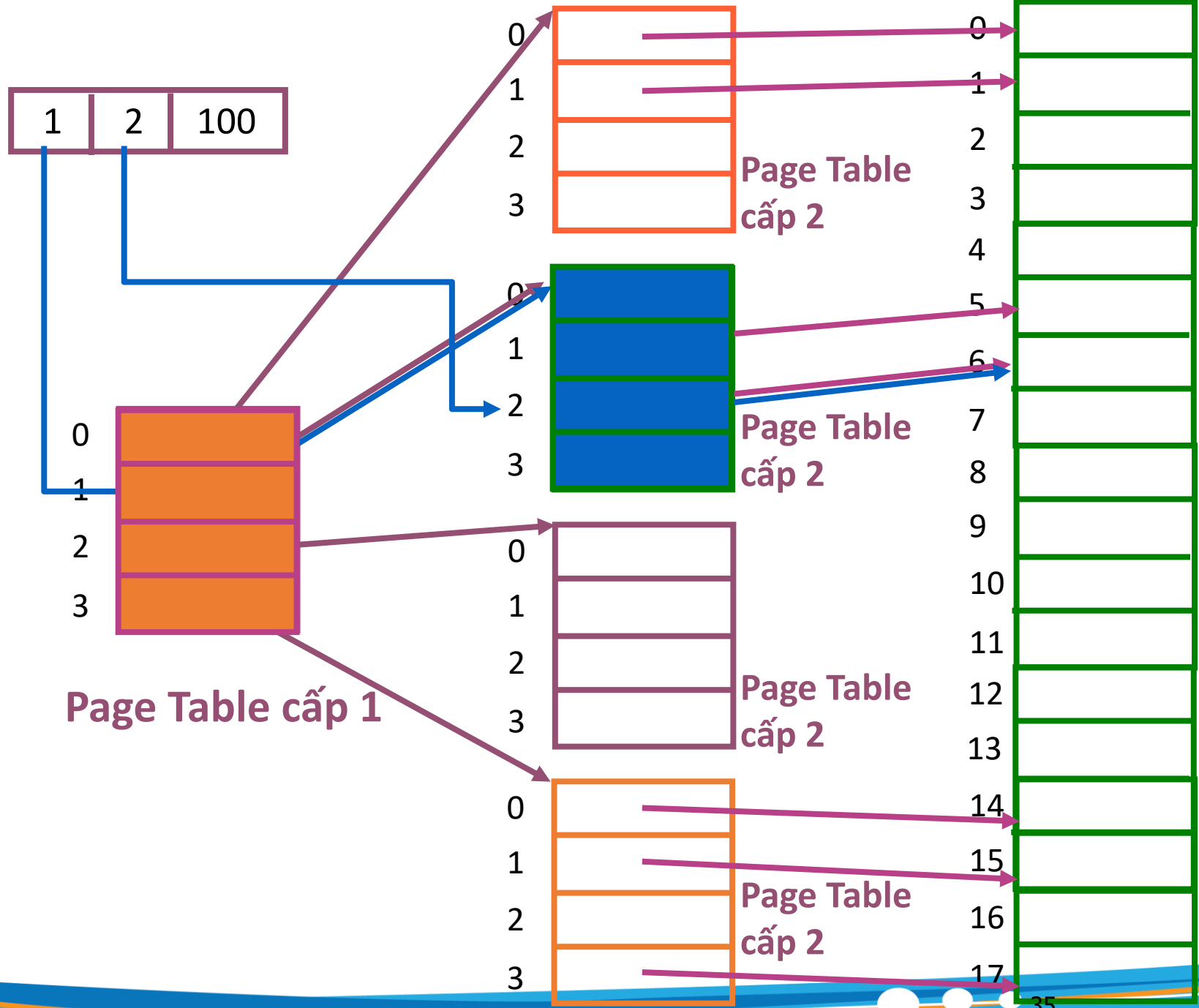
- Một máy tính sử dụng địa chỉ 32bít với kích thước trang 4Kb.
- Địa chỉ logic được chia thành 2 phần:
  - Số hiệu trang : 20 bits.
  - Offset tính từ đầu mỗi trang :12 bits.
- Vì bảng trang lại được phân trang nên số hiệu trang lại được chia làm hai phần:
  - Số hiệu trang cấp 1.
  - Số hiệu trang cấp 2.

page number		page offset
p_i	p_2	d
10	10	12



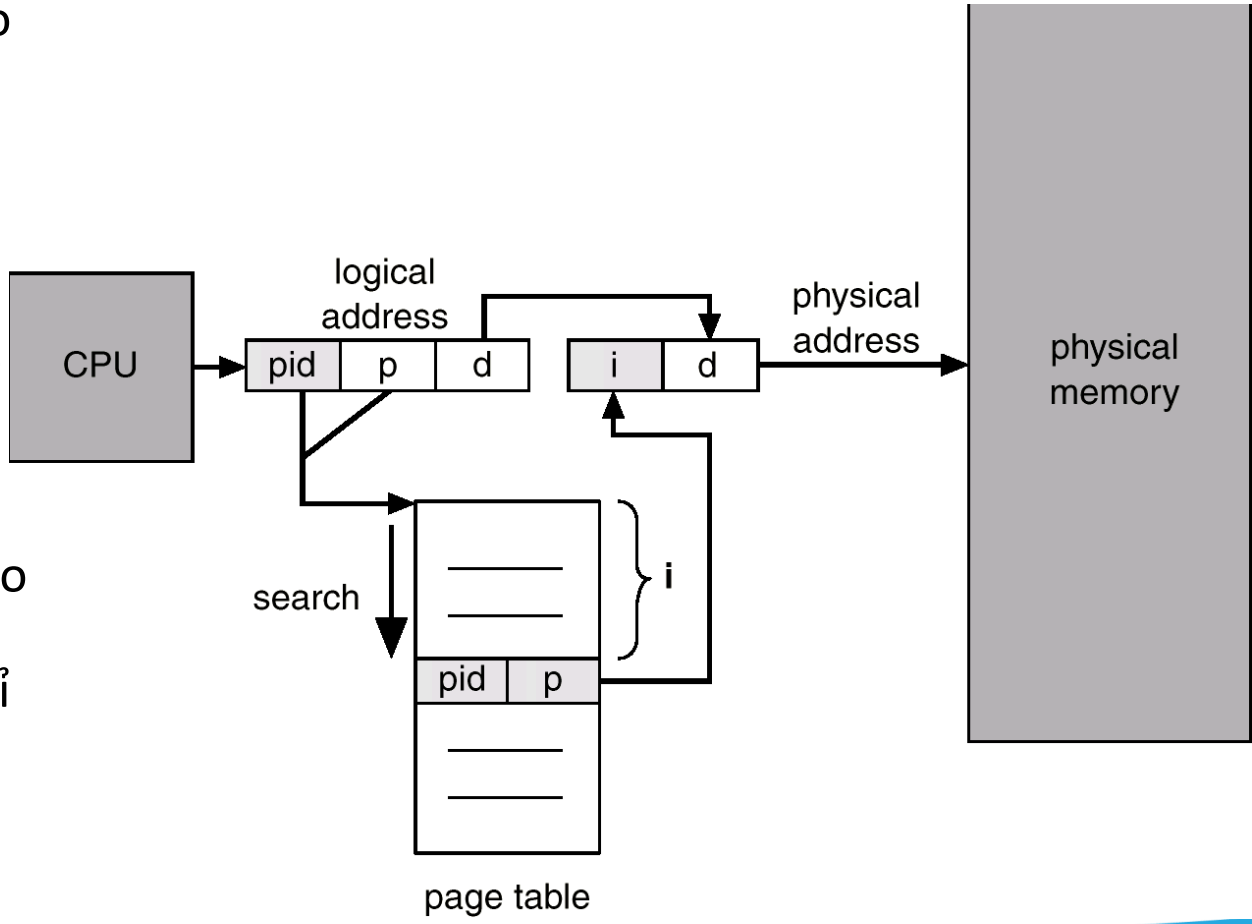
fit@hcmus

Ví dụ (tt)



Bảng trang nghịch đảo (Inverted Page Table)

- Sử dụng duy nhất một *bảng trang nghịch đảo* cho tất cả các tiến trình.
- Mỗi phần tử trong bảng trang nghịch đảo mô tả một frame, có cấu trúc:
  - <page> : số hiệu page mà frame đang chứa đựng.
  - <idp> : id của tiến trình đang được sỡ hữu trang.
- Mỗi địa chỉ ảo khi đó là một bộ ba <idp, p, d >
- Khi một tham khảo đến bộ nhớ được phát sinh, một phần địa chỉ ảo là <idp, p > được đưa đến cho trình quản lý bộ nhớ để tìm phần tử tương ứng trong bảng trang nghịch đảo, nếu tìm thấy, địa chỉ vật lý <i,d> sẽ được phát sinh.

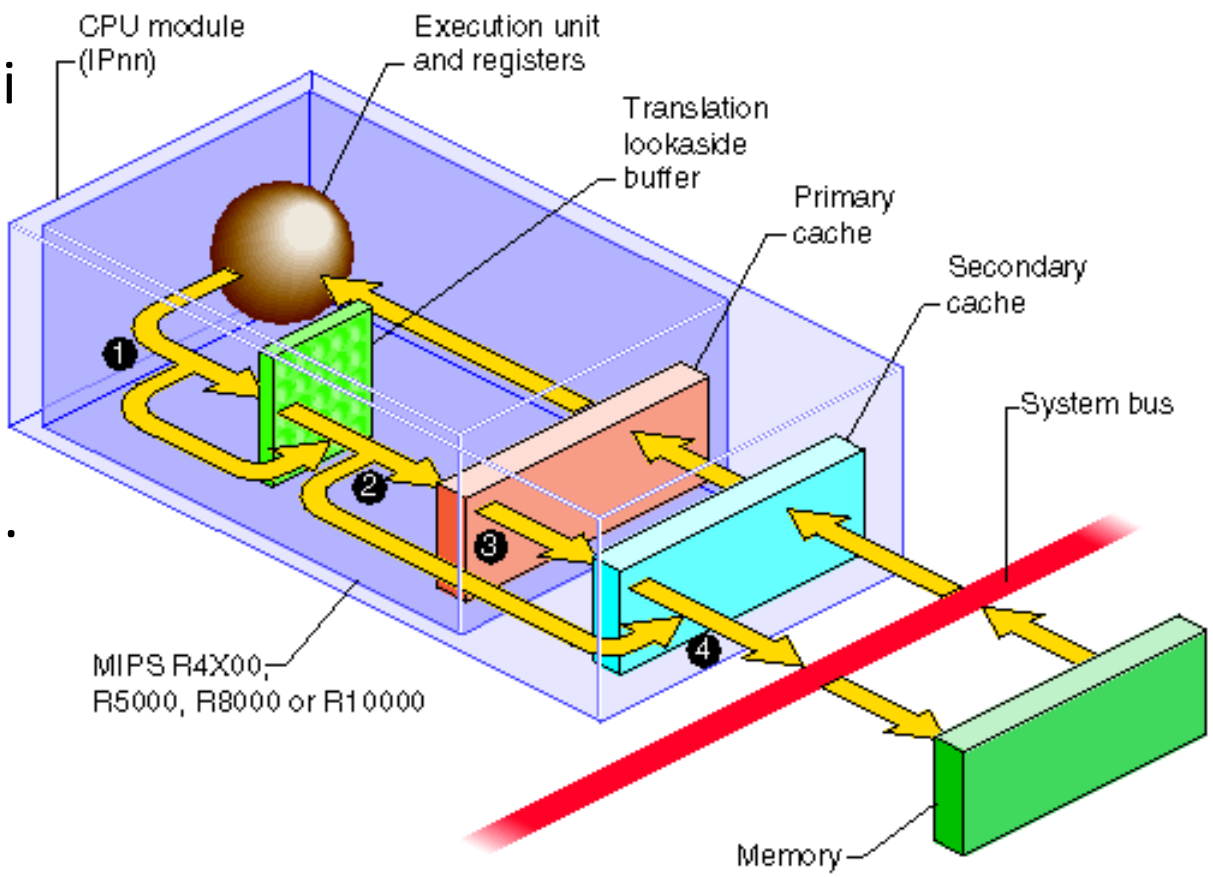


Paging – Vấn đề truy xuất bộ nhớ

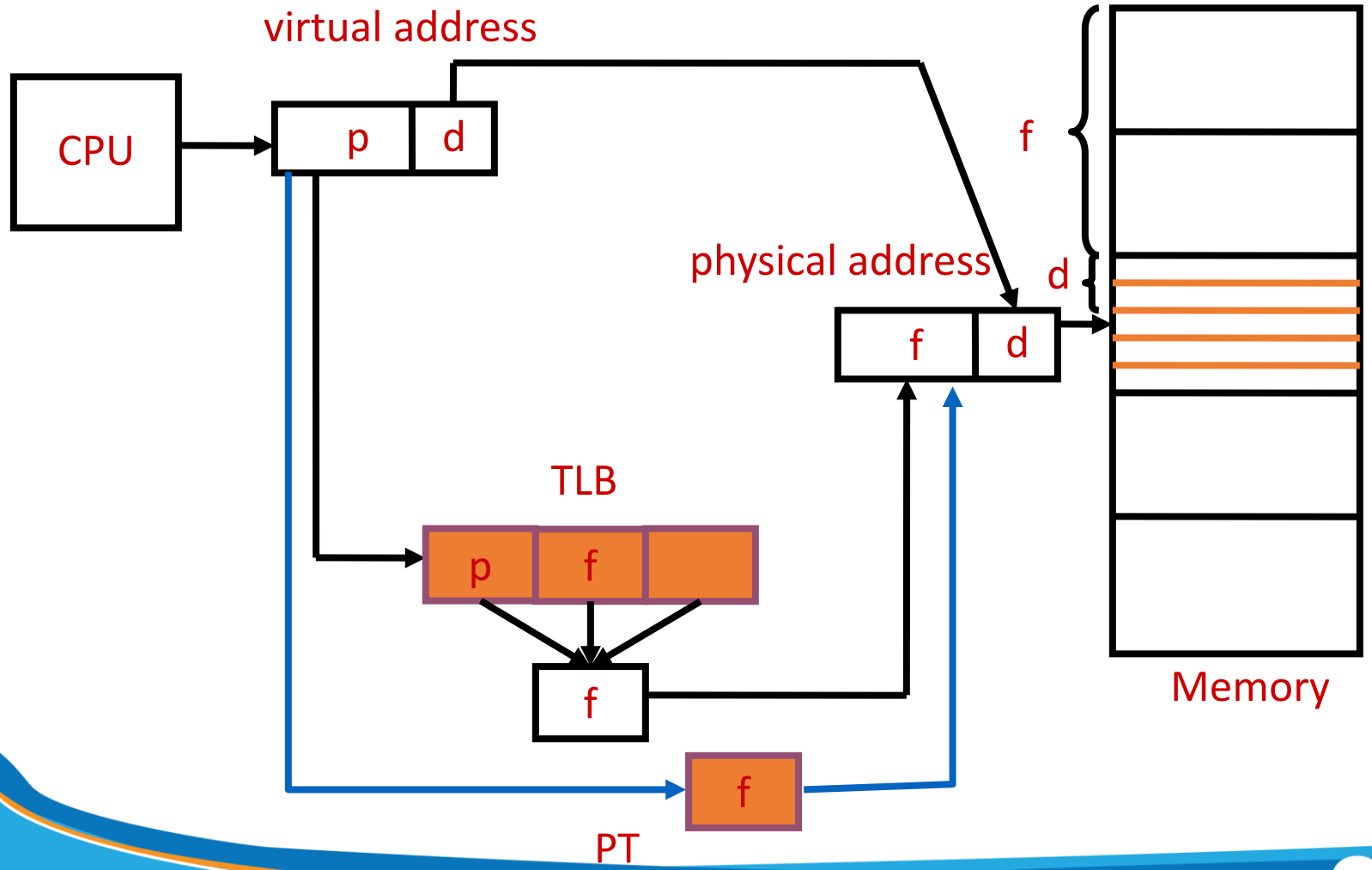
- Mỗi truy cập BNC cần truy xuất BNC 2 lần:
  - Tra cứu Page Table để chuyển đổi địa chỉ,
  - Tra cưu bản thân data,
- Làm gì để cải thiện:
  - Tìm cách lưu PT trong cache,
    - Cho phép tìm kiếm nhanh,
  - PT lớn, cache nhỏ : làm sao lưu đủ ?
    - Lưu 1 phần PT ...
    - Phần nào ?
      - Các số hiệu trang mới truy cập gần đây nhất ...

Giải pháp Translation Lookaside Buffer (TLB)

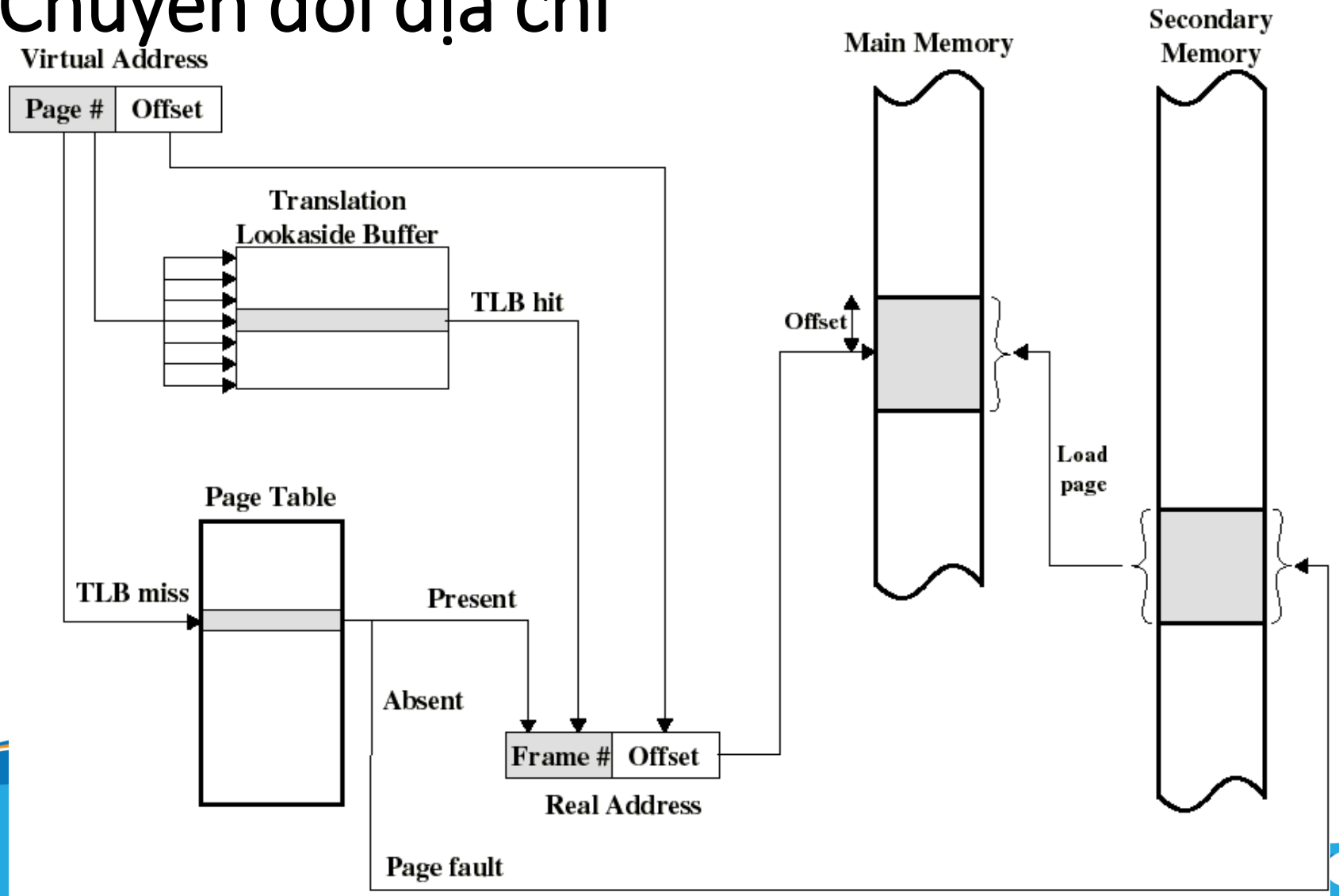
- Vùng nhớ Cache trong CPU được sử dụng để lưu tạm thời một phần của PT được gọi là Translation Lookaside Buffer (TLB).
  - Cho phép tìm kiếm tốc độ cao.
  - Kích thước giới hạn (thường không quá 64 phần tử).
- Mỗi entry trong TLB chứa một số hiệu page và frame tương ứng đang chứa page.
- Khi chuyển đổi địa chỉ, truy xuất TLB trước, nếu không tìm thấy số hiệu page cần thiết, mới truy xuất vào PT để lấy thông tin frame.



TLB – Mô hình chuyển đổi địa chỉ



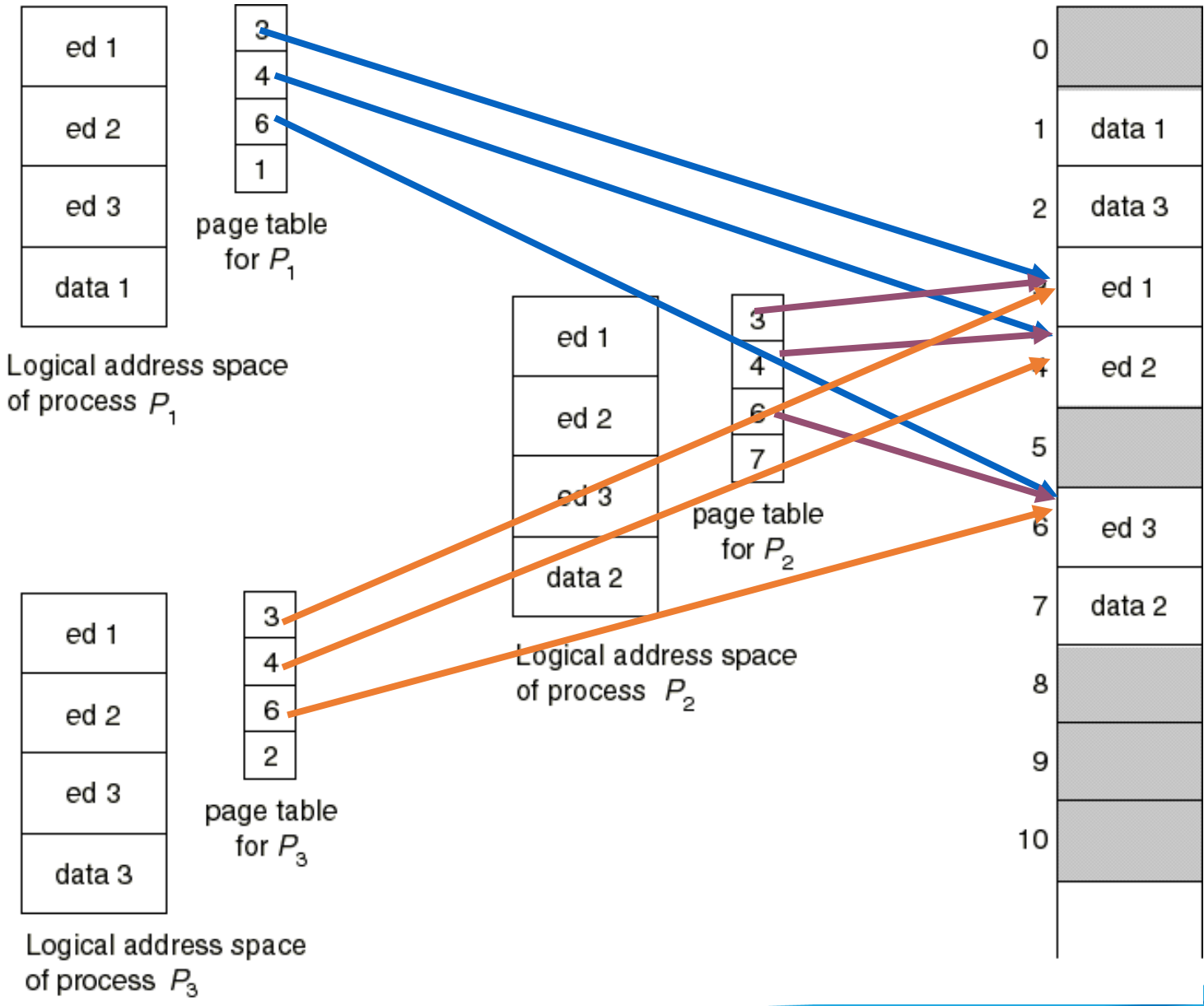
TLB – Chuyển đổi địa chỉ



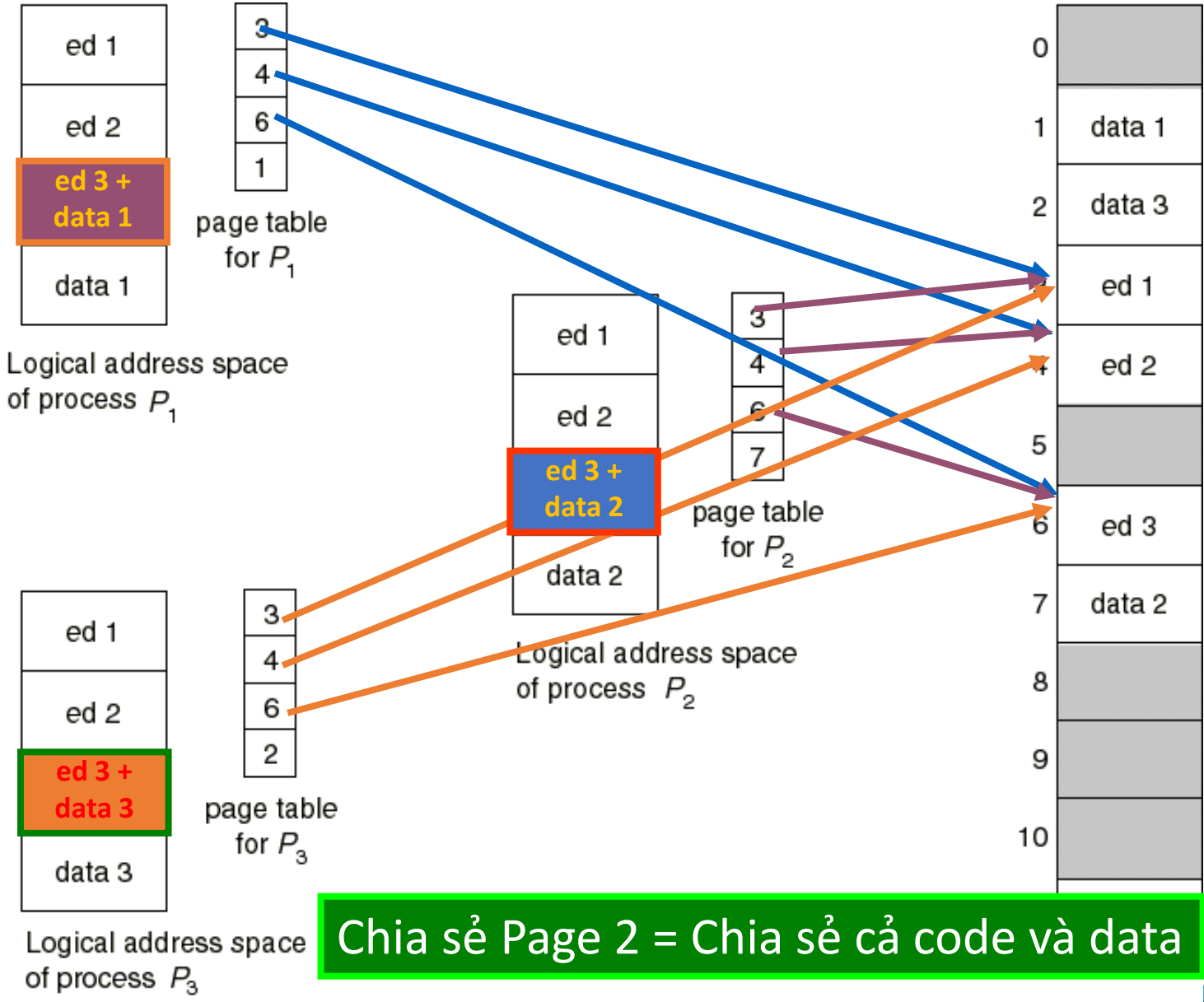
Bảo vệ và chia sẻ trong Segmentation và Paging

- Bảo vệ:
  - Segmentation: mỗi phần tử trong ST được gắn thêm các bit bảo vệ.
    - Mỗi segment có thể được bảo vệ tùy theo ngữ nghĩa của các đối tượng bên trong segment.
  - Paging: mỗi phần tử trong PT được gắn thêm các bit bảo vệ.
    - Mỗi page không nhận thức được ngữ nghĩa của các đối tượng bên trong page, nên bảo vệ chỉ áp dụng cho toàn bộ trang, không phân biệt.
- Chia sẻ: Cho nhiều phần tử trong KGĐC cùng trỏ đến 1 vị trí trong KGVL.
  - Segmentation : chia sẻ mức module chương trình.
  - Paging : chia sẻ các trang.

Paging – Chia sẻ (1/2)



Paging – Chia sẻ (1/2)



Chia sẻ Page 2 = Chia sẻ cả code và data

Đánh giá các mô hình chuyển đổi địa chỉ

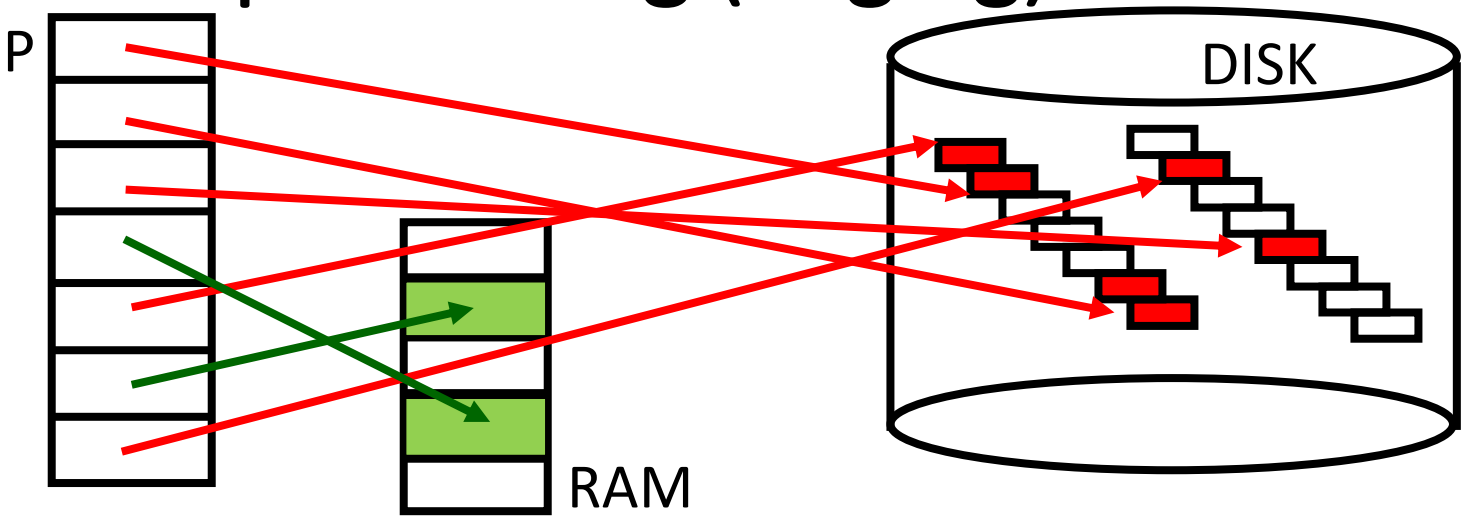
- Giả sử có:
  - t_m : thời gian truy xuất BNC.
  - t_c : thời gian truy xuất cache.
  - hit-ration : tỉ lệ tìm thấy một số hiệu trang p trong TLB.
- Công thức tính thời gian truy cập thực tế (Time Effective Access) đến một đối tượng trong BNC:
  - Bao gồm thời gian chuyển đổi địa chỉ và thời gian truy xuất dữ liệu.
  - TEA = (time biding add + time acces memory).
- Linker-Loader:
  - TEA = t_m (data)
- Base & Bound:
  - TEA = (t_c + t_c) (Base & Bound) + t_m (data)
- Segmentation:
  - TEA = t_c (ST trong cache) + t_m (data)
- Paging:
  - Không sử dụng TLB:
    - TEA = t_m (PT trong mem) + t_m (data)
  - Có sử dụng TLB :
    - TEA = hit-ratio (t_c (TLB) + t_m (data)) + (1- hit-ratio)(tc (TLB) + tm (PT) + tm (data))

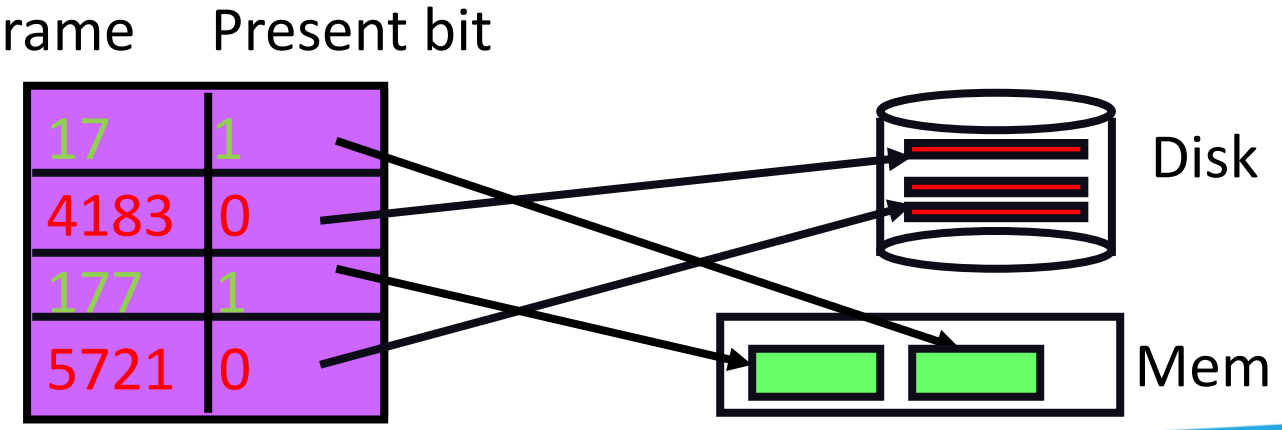
Bộ nhớ ảo (Virtual Memory)

- Vấn đề:
  - Nếu tiến trình cần thực thi có kích thước lớn hơn dung lượng bộ nhớ chính còn trống ?
  - Tại một thời điểm chỉ có 1 chỉ thị được thi hành, tại sao phải nạp tất cả tiến trình vào BNC cùng 1 lúc ?
- Ý tưởng:
  - Cho phép nạp và thi hành từng phần tiến trình.
  - Tại một thời điểm chỉ giữ trong BNC các chỉ thị và dữ liệu cần thiết tại thời điểm đó.
  - Các phần khác của tiến trình giữ trên đĩa, được hoán đổi vào/ra bộ nhớ khi cần.
- Giải pháp Bộ nhớ ảo (virtual memory).
  - Được bắt đầu áp dụng vào những năm 1960.
  - Tách biệt KGĐC và KGVL.
    - LTV: mỗi tiến trình làm việc với KGĐC 2^m của mình (địa chỉ từ 0 (2^m -1)).
    - HĐH: chịu trách nhiệm nạp các KGĐC vào một KGVL chung.
  - Phân chia KGĐC thành các phần ?
    - Paging/Segmentation.
  - Mở rộng BNC để lưu trữ các phần của tiến trình chưa được nạp.
    - Dùng BNP(disk) để mở rộng BNC.
  - Cơ chế chuyển đổi qua lại các phần của tiến trình giữa BNC và BNP.
    - Swapping ...

Virtual Memory với cơ chế phân trang (Paging)

- Phân chia KGĐC thành các page.
- Phân chia KGVL thành các frame.
- BNC chứa các trang của tiến trình đang được nạp.
- BNP(disk) để mở rộng BNC, lưu trữ các trang của tiến trình chưa được nạp.
- Bảng trang : thêm 1 bit “present” có giá trị 0/1 vào mỗi phần tử để nhận diện trang đã hay chưa được nạp vào RAM.



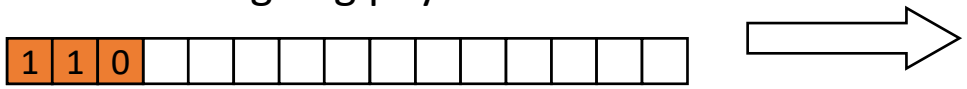


Chuyển đổi địa chỉ (1/2)

Page table

		present bit
15	000	0
14	000	0
13	000	0
12	000	0
11	111	1
10	000	0
9	101	1
8	000	0
7	000	0
6	000	0
5	011	1
4	100	1
3	000	1
2	110	1
1	001	1
0	010	1

Outgoing physical address



(0x6004, 24580)

4-bit index
into page table
virtual page = 0x0010 = 2

12-bit offset

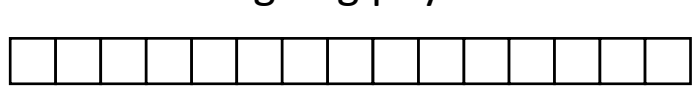
Incoming virtual address
(0x2004, 8196)

Chuyển đổi địa chỉ (2/2)

Page table

		present bit
15	000	0
14	000	0
13	000	0
12	000	0
11	111	1
10	000	0
9	101	1
8	000	0
7	000	0
6	000	0
5	011	1
4	100	1
3	000	1
2	110	0
1	001	1
0	010	1

Outgoing physical address

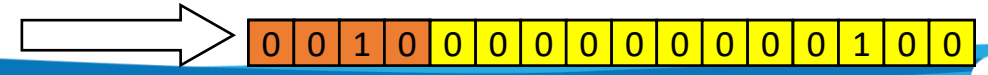


PAGE FAULT

4-bit index
into page table
virtual page = 0x0010 = 2

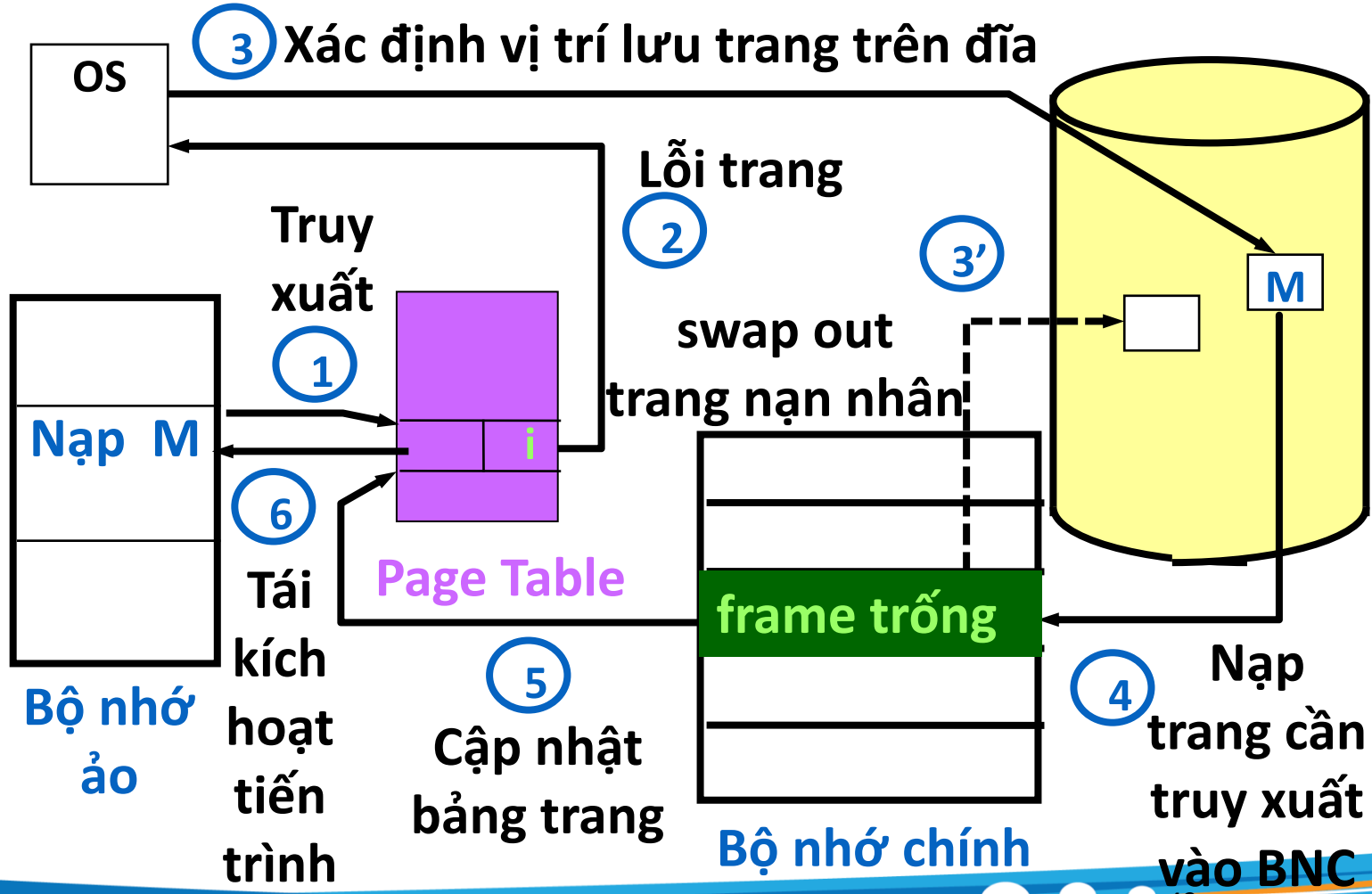
12-bit offset

Incoming virtual address
(0x2004, 8196)



Xử lý lỗi trang (Page Fault)

1. Kiểm tra truy xuất đến bộ nhớ là hợp lệ hay bất hợp lệ.
2. Nếu truy xuất bất hợp lệ: kết thúc tiến trình Ngược lại: đến bước 3.
3. Tìm vị trí chứa trang muốn truy xuất trên đĩa.
4. Tìm một khung trang trống trong bộ nhớ chính:
   a. Nếu tìm thấy: đến bước 5.
   b. Nếu không còn khung trang trống, chọn một khung trang nạn nhân để swap out, cập nhật bảng trang tương ứng rồi đến bước 5.
5. Chuyển trang muốn truy xuất từ bộ nhớ phụ vào bộ nhớ chính : nạp trang cần truy xuất vào khung trang trống đã chọn (hay vừa mới làm trống) ; cập nhật nội dung bảng trang, bảng khung trang tương ứng.
6. Tái kích hoạt tiến trình người sử dụng.

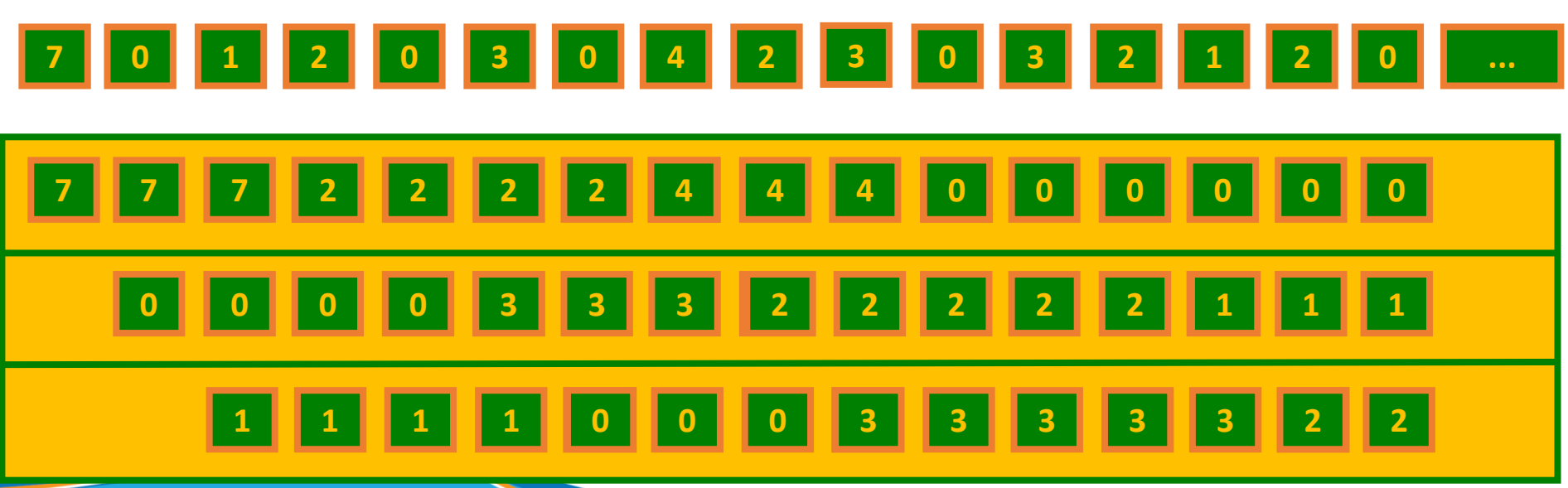


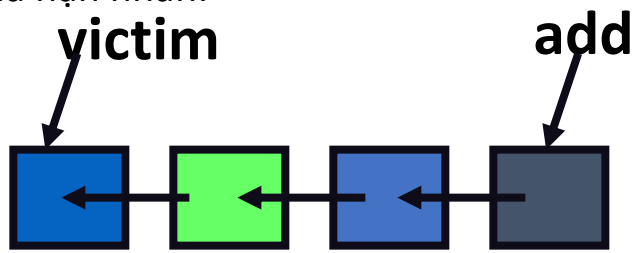
Thay thế trang (Page Replacement)

- Mục tiêu:
  - Thay thế trang sao cho tần suất xảy ra lỗi trang thấp nhất.
- Các thuật toán thay thế trang:
  - First-in, first-out (FIFO) algorithm.
  - Optimal algorithm.
  - Least recently used (LRU) algorithm.
  - Second-chance algorithm (clock model).
  - Not recently used algorithm.
  - Working set algorithm.
  - WSClock algorithm.
- Đánh giá:
  - Sử dụng số frame cụ thể. Ví dụ: 3 frames.
  - Giả sử có một chuỗi truy xuất cụ thể:
    - 7, 0, 1, 2, 0, 3, 0, 4, 2, 3, 0, 3, 2, 1, 2, 0, 1, 7, 0, 1
  - Thực hiện thuật toán thay thế trang trên chuỗi truy xuất này.
  - Đếm số lỗi trang phát sinh.

First-in, first-out (FIFO) algorithm

- Nguyên tắc:
  - Nạn nhân là trang "già" nhất.
    - Được nạp vào lâu nhất trong hệ thống.
- Thực hiện:
  - Lưu thời điểm nạp, so sánh để tìm min.
    - Chi phí cao.
  - Tổ chức FIFO các trang theo thứ tự nạp.
    - Trang đầu danh sác là nạn nhân.





FIFO – Nhận xét

- Nhận xét:
  - Đơn giản.
  - Công bằng ?
  - Không xét đến tính sử dụng !
    - Trang được nạp vào lâu nhất có thể là trang cần sử dụng thường xuyên !
- Sử dụng càng nhiều frame ... càng có nhiều lỗi.

All pages frames initially empty

	0	1	2	3	0	1	4	0	1	2	3	4
Youngest page	0	1	2	3	0	1	4	4	4	2	3	3
		0	1	2	3	0	1	1	1	4	2	2
Oldest page			0	1	2	3	0	0	0	1	4	4
	P	P	P	P	P	P	P			P	P	

9 Page faults

(a)

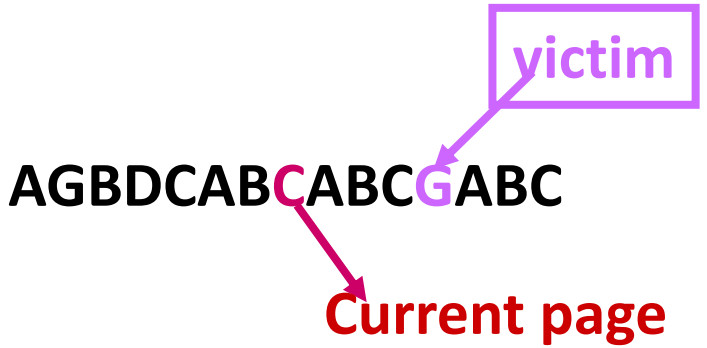
	0	1	2	3	0	1	4	0	1	2	3	4
Youngest page	0	1	2	3	3	3	4	0	1	2	3	4
		0	1	2	2	2	3	4	0	1	2	3
Oldest page			0	1	1	1	2	3	4	0	1	2
				0	0	0	1	2	3	4	0	1
	P	P	P	P			P	P	P	P	P	P

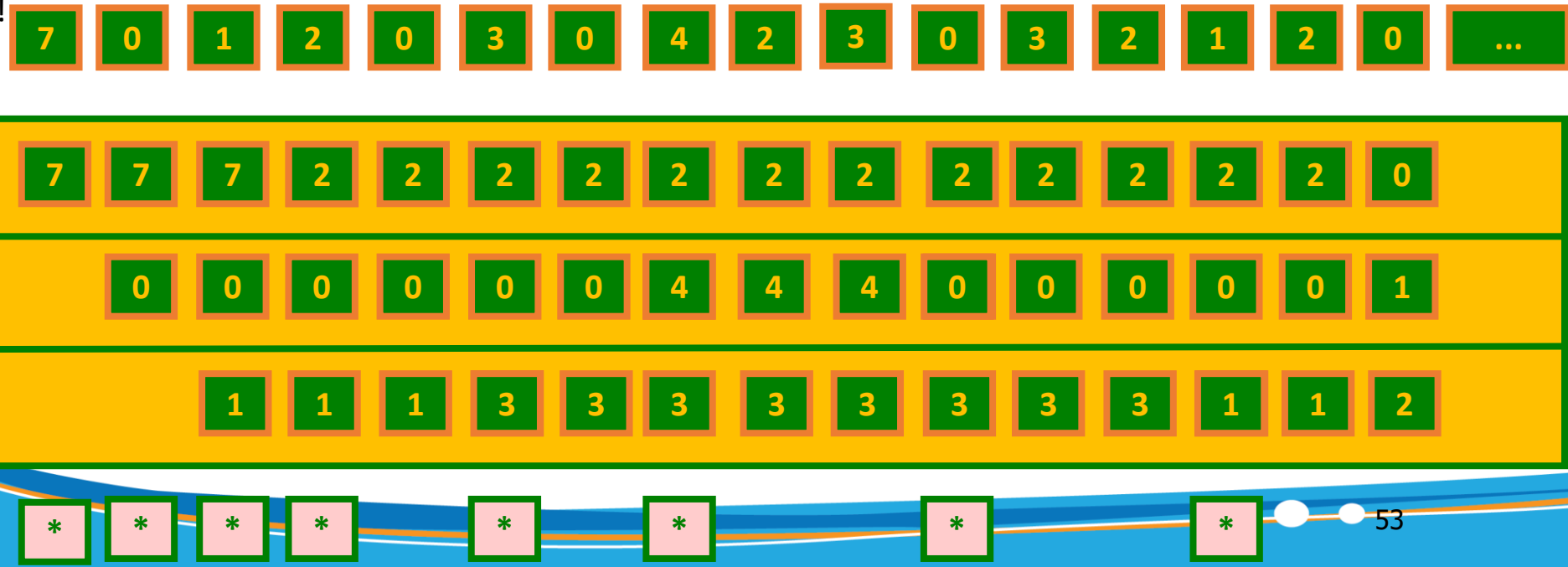
10 Page faults

(b)

Optimal algorithm

- Nguyên tắc:
  - Nạn nhân là trang lâu sử dụng đến nhất trong tương lai.
- Nhận xét:
  - Bảo đảm tần suất lỗi trang thấp nhất.
  - Làm sao biết "tương lai" ?
    - Không khả thi !





Least recently used (LRU) algorithm

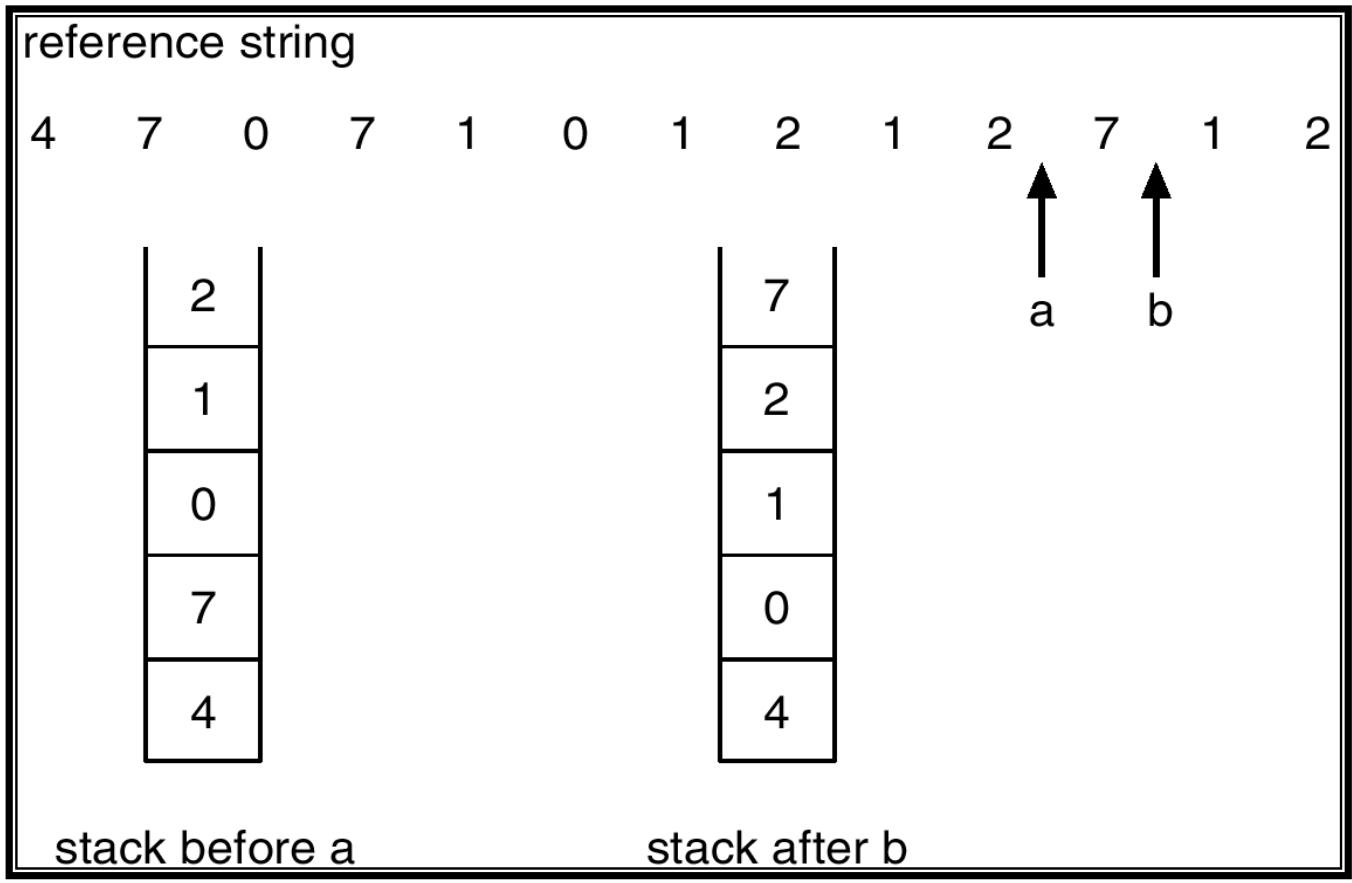
- Nguyên tắc:
  - Nạn nhân là trang lâu nhất chưa sử dụng đến trong quá khứ.
  - Nhìn lui: đủ thông tin.
- Nhận xét:
  - Xấp xỉ Optimal.

LRU – Cài đặt sử dụng hỗ trợ phần cứng

- Thêm trường reference time cho mỗi phần tử trong bảng trang.
- Thêm vào cấu trúc của CPU một bộ đếm counter.
- Mỗi lần có sự truy xuất đến một trang trong bộ nhớ:
  - Giá trị của counter tăng lên 1.
  - Giá trị của counter được ghi nhận vào reference time của trang tương ứng.
- Thay thế trang có reference time là min.
- Tuy nhiên, hệ thống có hỗ trợ phần cứng hoàn chỉnh để cài đặt LRU ?

LRU – Cài đặt sử dụng ngăn xếp

- Tổ chức một stack lưu trữ các số hiệu trang.
- Mỗi khi thực hiện một truy xuất đến một trang, số hiệu của trang sẽ được xóa khỏi vị trí hiện hành trong stack và đưa lên đầu stack.
- Trang ở đỉnh stack là trang được truy xuất gần nhất, và trang ở đáy stack là trang lâu nhất chưa được sử dụng.



LRU – Cài đặt sử dụng các bits history

- Thêm 1 bit "reference" có giá trị 0/1 vào mỗi phần tử của bảng trang:
  - Được khởi gán là 0.
  - Được phần cứng đặt giá trị 1 mỗi lần trang tương ứng được truy cập.
  - Được phần cứng gán trở về 0 sau từng chu kỳ qui định trước.
  - Bit "reference" chỉ giúp xác định những trang có truy cập, không xác định thứ tự truy cập.
- Thêm N bit history phụ trợ vào mỗi phần tử của bảng trang:
  - Sau từng chu kỳ, bit reference sẽ được chép lại vào một bit history trước khi bị reset
  - N bit history sẽ lưu trữ tình hình truy xuất đến trang trong N chu kỳ cuối cùng.

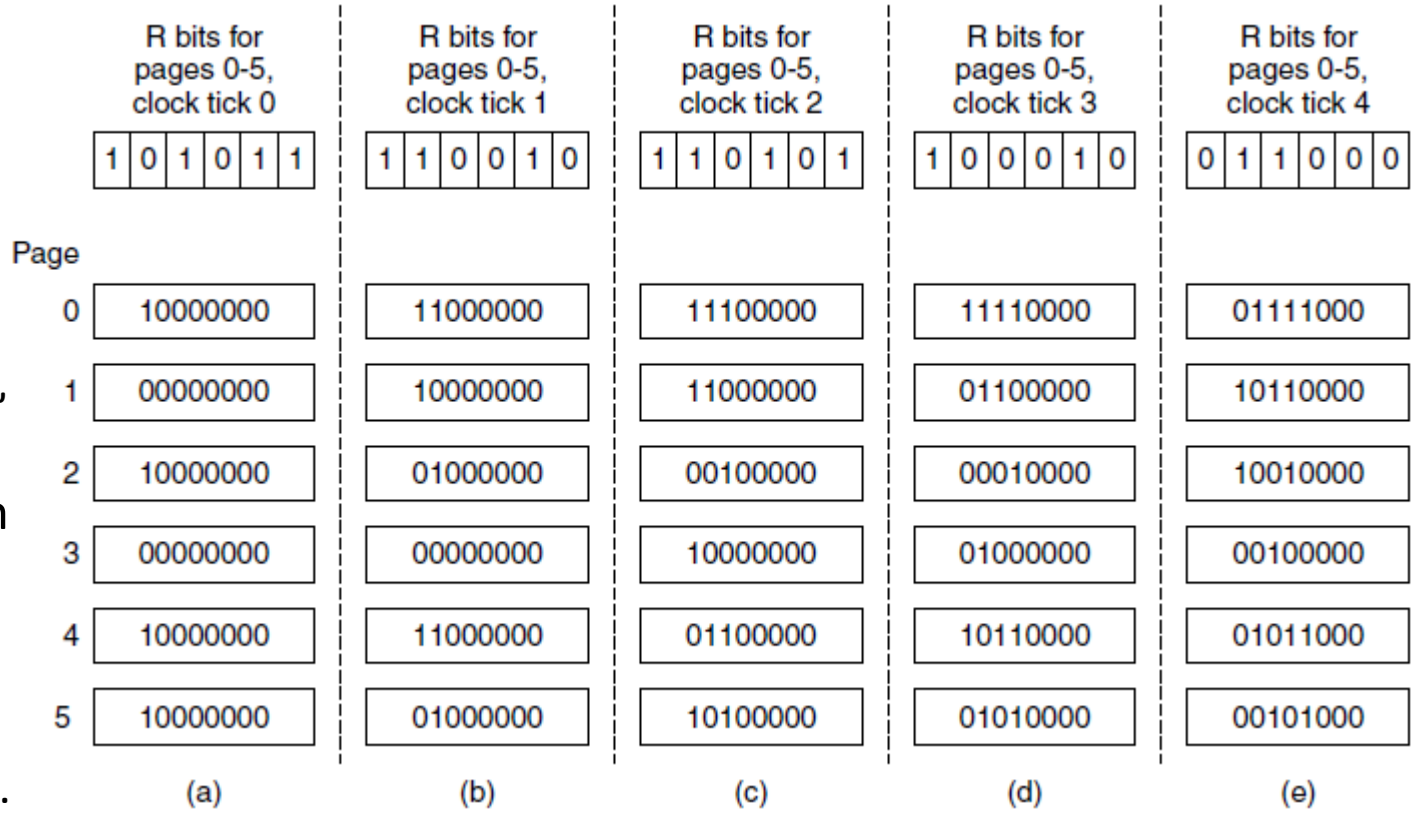
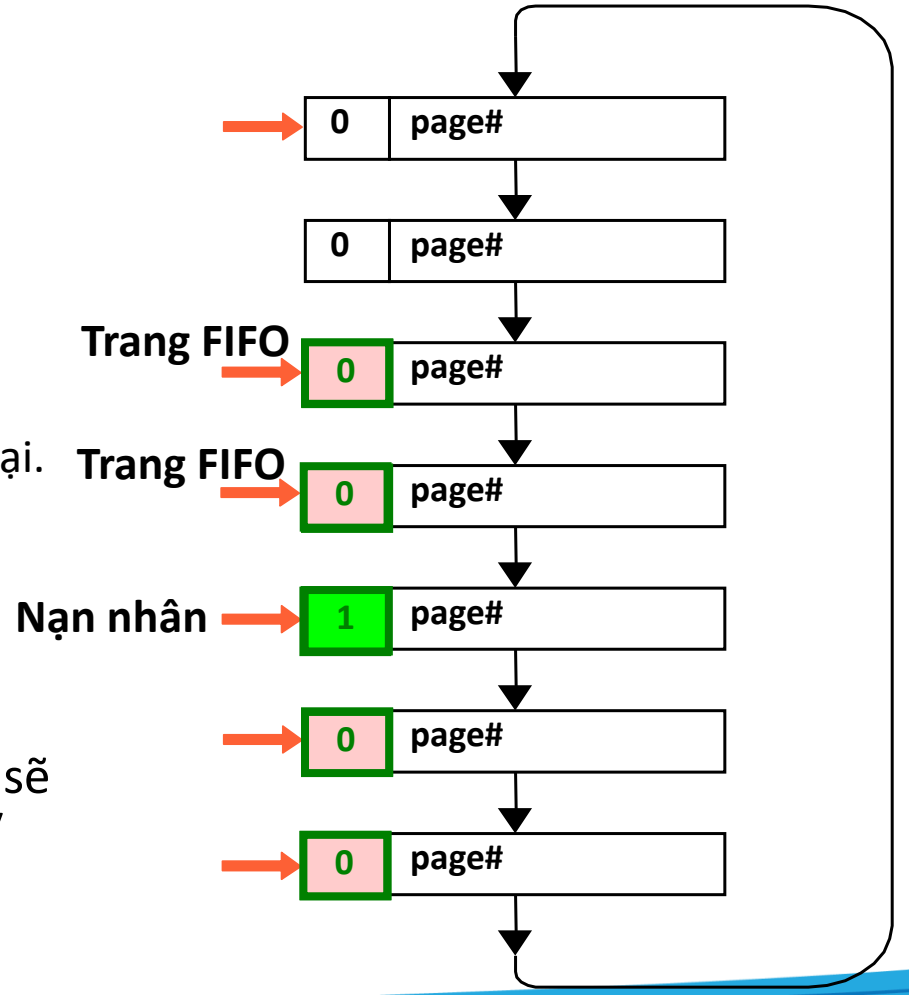


Figure 3-17. The aging algorithm simulates LRU in software. Shown are six pages for five clock ticks. The five clock ticks are represented by (a) to (e).

Second-chance algorithm (clock model)

- Sử dụng một bit reference duy nhất.
- Chọn được trang nạn nhân theo FIFO.
- Kiểm tra bit reference của trang đó:
  - Nếu reference = 0, đúng là nạn nhân rồi.
  - Nếu reference = 1, cho trang này một cơ hội thứ 2.
    - reference = 0.
    - Thời điểm vào Ready List được cập nhật lại là thời điểm hiện tại.
- Chọn trang FIFO tiếp theo ...
- Nhận xét :
  - Một trang đã được cho cơ hội thứ hai sẽ không bị thay thế trước khi hệ thống đã thay thế hết những trang khác.
  - Nếu trang thường xuyên được sử dụng, bit reference của nó sẽ duy trì được giá trị 1, và trang hầu như không bao giờ bị thay thế.



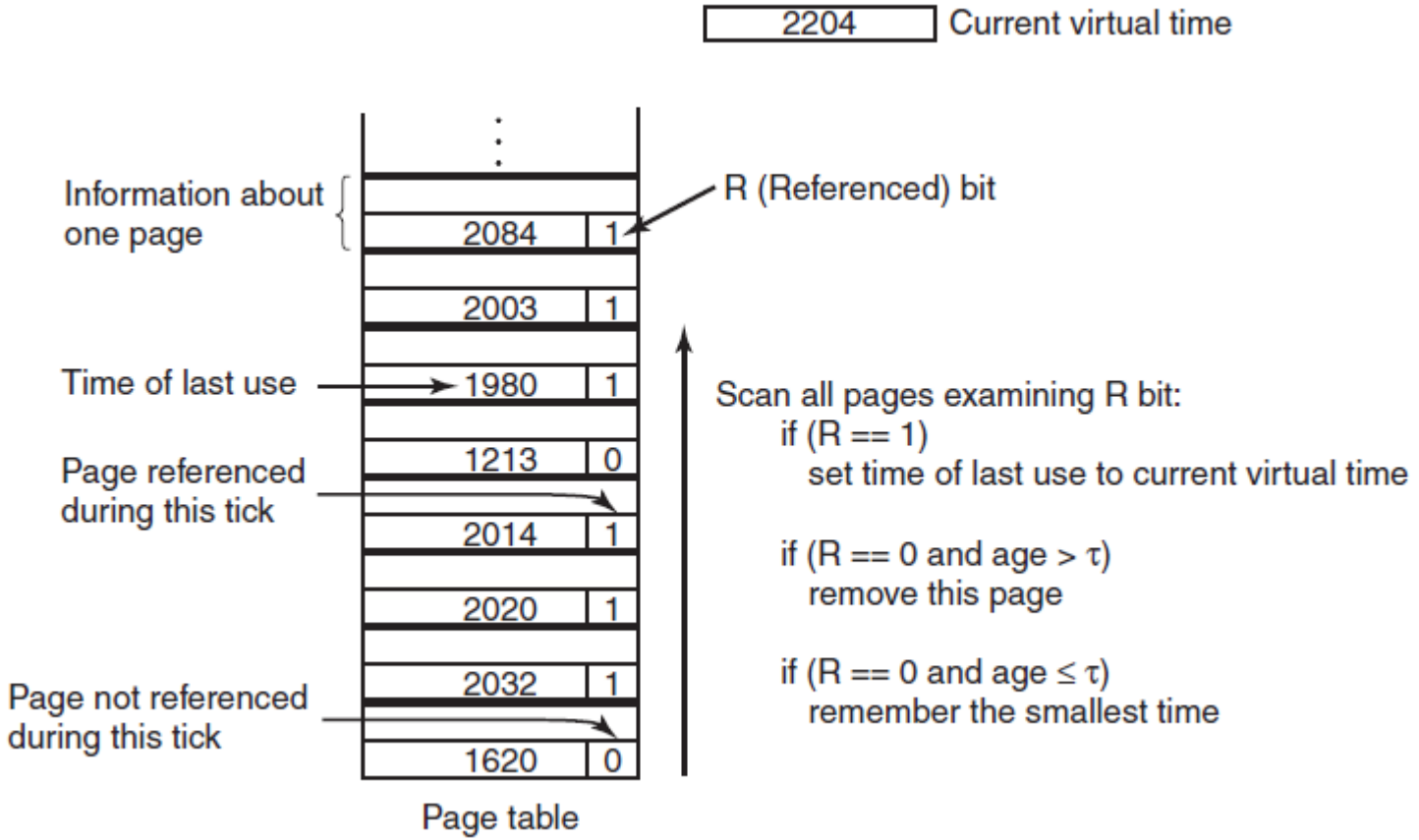
Not recently used algorithm

- Sử dụng 2 bit Reference và Modify.
- Với hai bit này, có thể có 4 tổ hợp tạo thành 4 lớp sau:
  - (0,0) không truy xuất, không sửa đổi.
  - (0,1) không truy xuất gần đây, nhưng đã bị sửa đổi.
  - (1,0) được truy xuất gần đây, nhưng không bị sửa đổi.
  - (1,1) được truy xuất gần đây, và bị sửa đổi.
- Chọn trang nạn nhân là trang có độ ưu tiên cao nhất khi kết hợp bit R và bit M.

Priority	R	M
4	0	0
3	0	1
2	1	0
1	1	1

Working Set Algorithm (1968, Denning)

- Working set = tập hợp các trang tiến trình đang truy xuất tại 1 thời điểm.
  - Các pages được truy xuất *w* trong lần cuối cùng sẽ nằm trong working set của tiến trình.
  - Kích thước của WS thay đổi theo thời gian tùy vào locality của tiến trình.
- Sử dụng Working set:
  - Cache partitioning: Cấp cho mỗi tiến trình số frame đủ chứa WS của nó.
  - Page replacement: ưu tiên swap out các non-WS pages.
  - Scheduling: chỉ thi hành tiến trình khi đủ chỗ để nạp WS của nó.



Working Set Algorithm (Clock Implementation)

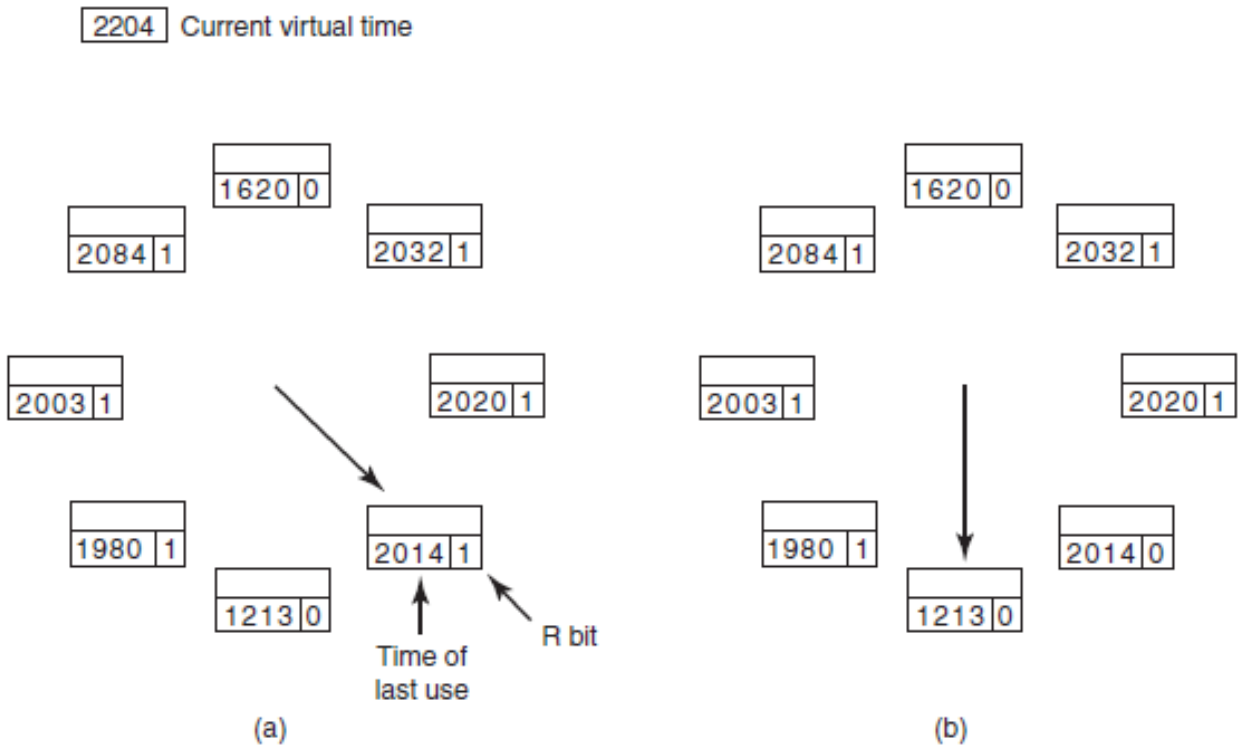


Figure 3-20. Operation of the WSClock algorithm.
(a) and (b) give an example of what happens when R = 1.

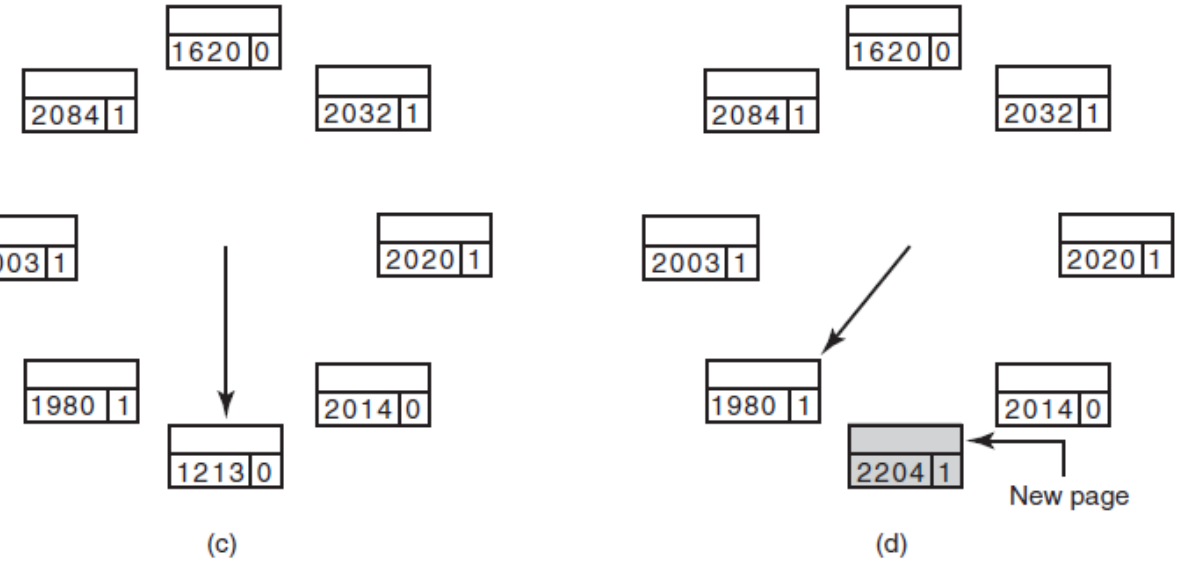
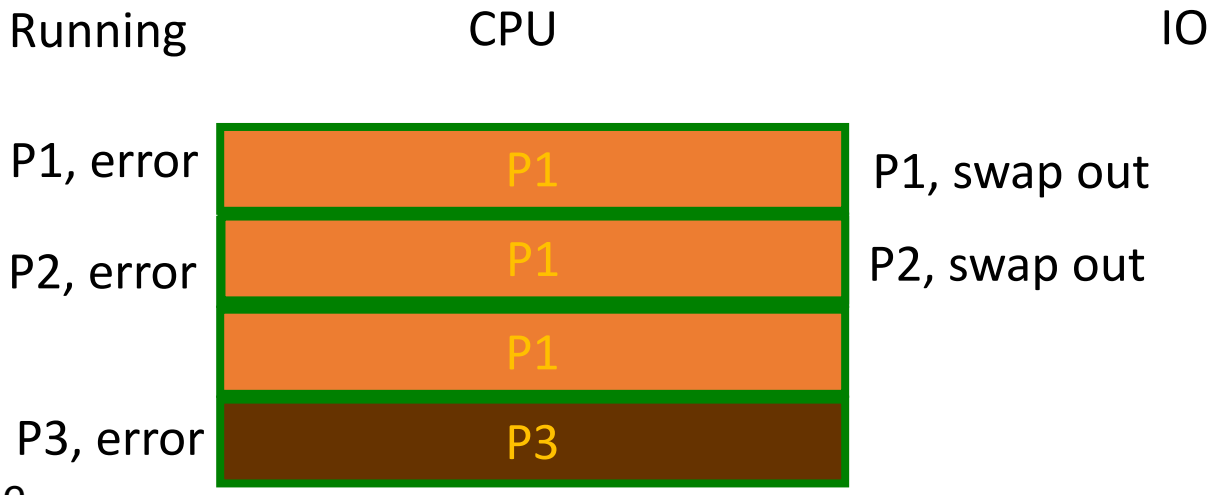


Figure 3-20. Operation of the WSClock algorithm.
(c) and (d) give an example of R = 0.

Chiến lược cấp phát frame

- Số frame cần cấp phát cho mỗi tiến trình ?
  - Giải sử có m frame và n process.
  - Cấp phát công bằng: #frame(Pi) = m/n
    - Công bằng ???
  - Cấp phát theo tỷ lệ: #frame(pi) = (si / (si))* m
    - si = kích thước của bộ nhớ ảo cho tiến trình pi.
- Lỗi trang xảy ra tiếp theo, cấp phát thêm frame cho tiến trình như thế nào ?
  - Cục bộ: chỉ chọn trang nạn nhân trong tập các trang của tiến trình phát sinh lỗi trang -> số frame không tăng.
  - Toàn cục: được chọn bất kỳ trang nạn nhân nào (dù của tiến trình khác) -> số frame có thể tăng, lỗi trang lan truyền.

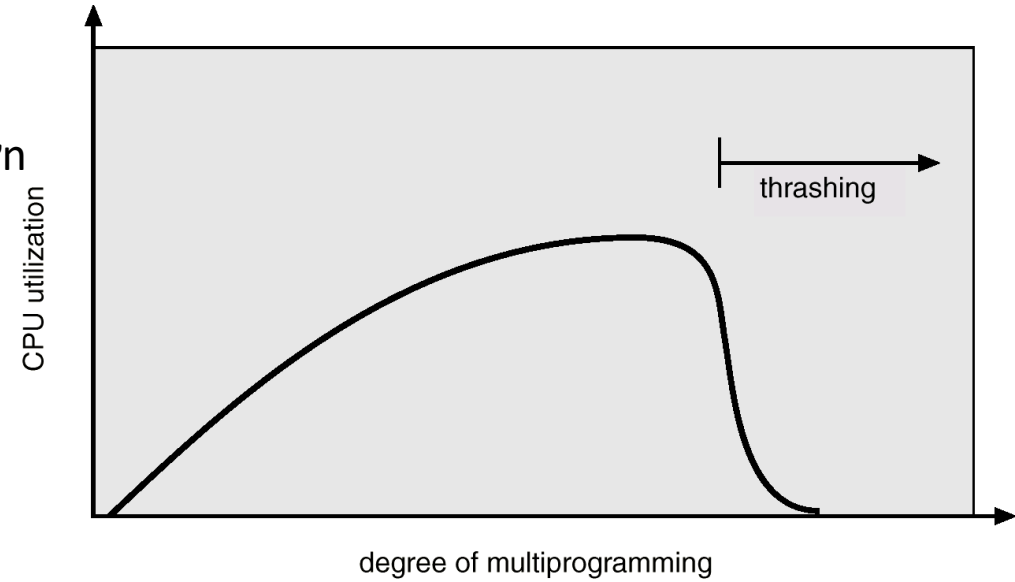


Tất cả các tiến trình bận rộn thay thế trang !

Thrashing (Denning, 1968)

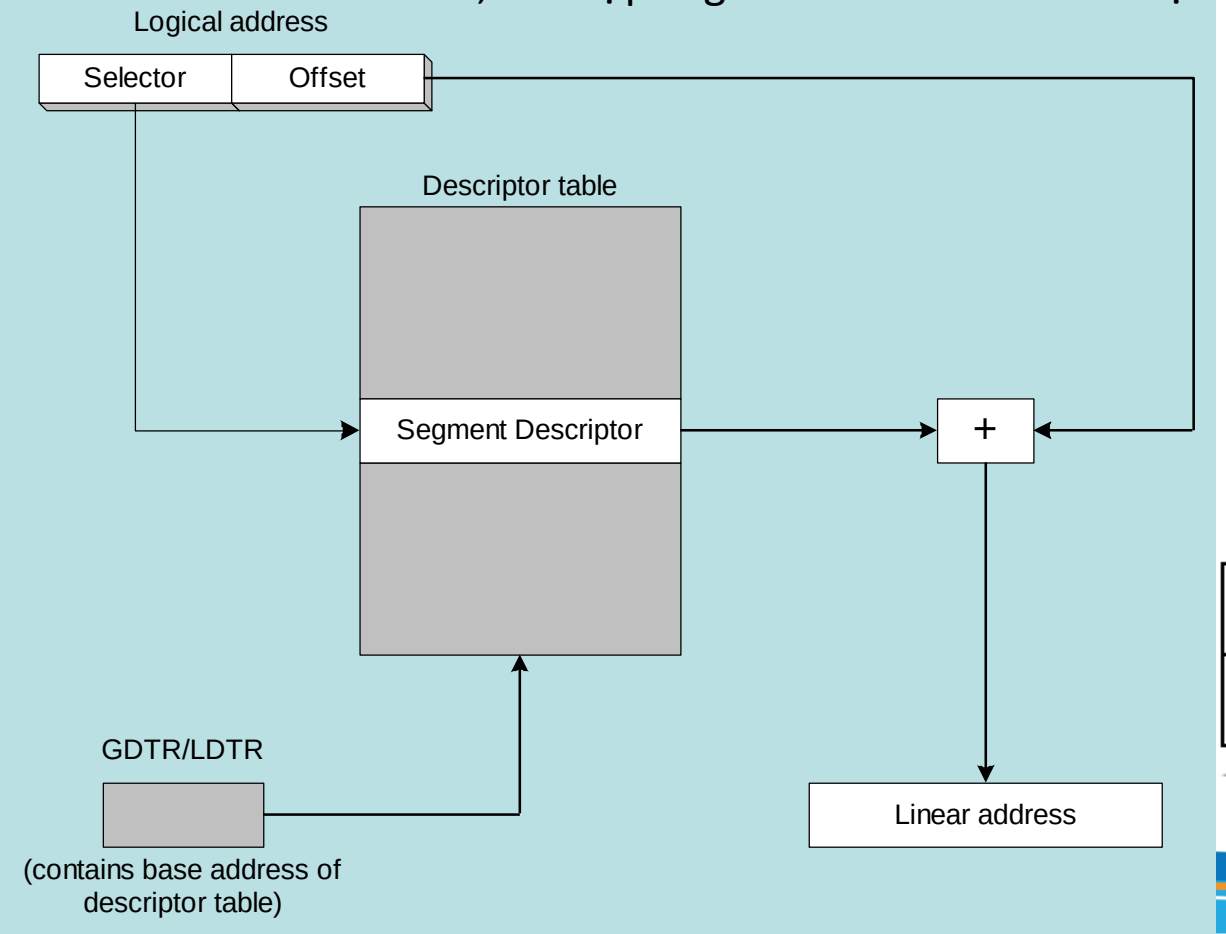
- Là vấn đề quan trọng của cơ chế bộ nhớ ảo.
- Nguyên nhân:
  - Tiến trình không tái sử dụng bộ nhớ (quá khứ != tương lai).
  - Tiến trình tái sử dụng bộ nhớ, nhưng với kích thước lớn hơn.
  - Quá nhiều tiến trình trong hệ thống, yêu cầu bộ nhớ nhiều hơn khả năng cung cấp của hệ thống !
    - Σ size of working sets > total memory size

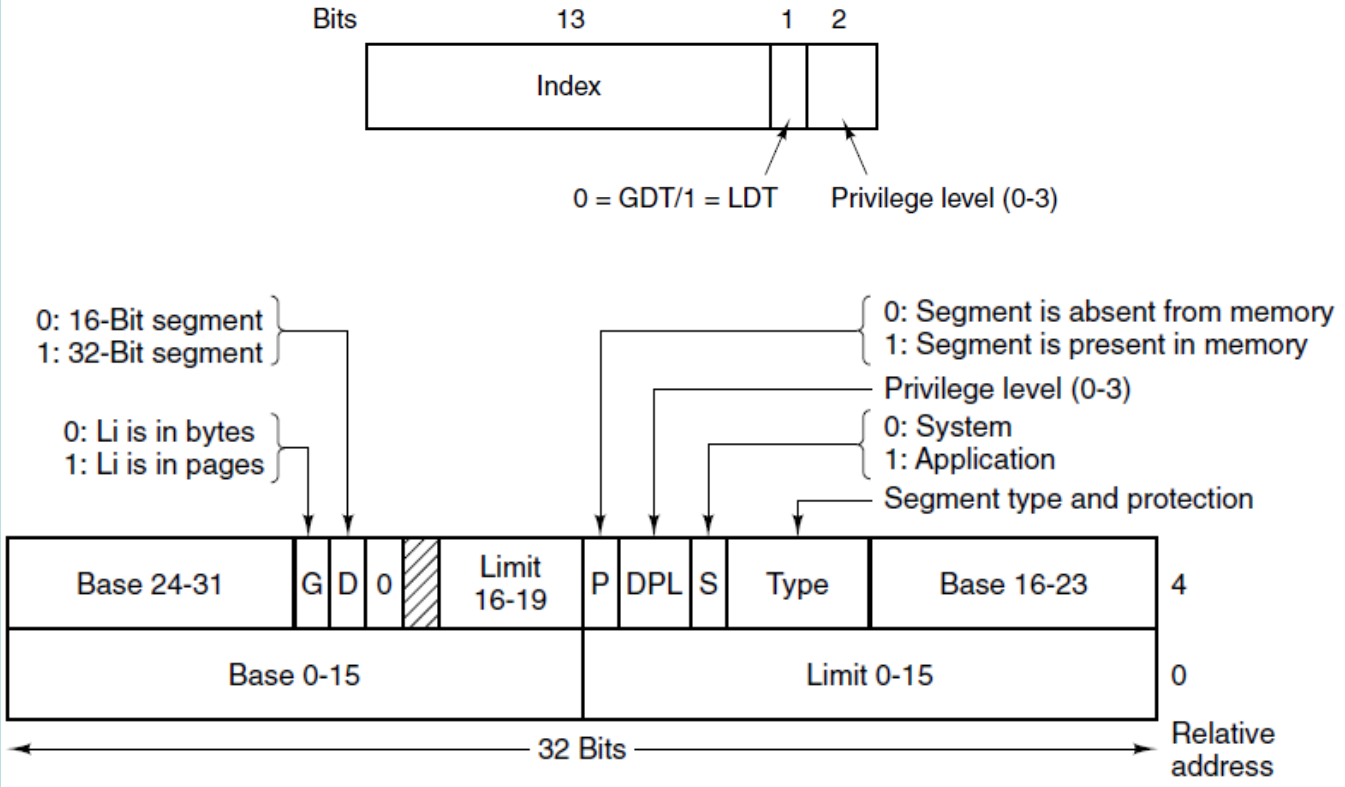
  → Chỉ có thể kiểm soát thrashing do nguyên nhân thứ 3.
- Hậu quả:
  - Tiến trình xảy ra lỗi trang sau một vài lệnh được thực thi.
  - Tất cả tiến trình đều bận rộn xử lý lỗi trang !
  - IO hoạt động 100 %, CPU rảnh !
  - Hệ thống ngừng trệ.



Tổ chức bộ nhớ Intel x86 (1/2)

- Kết hợp giữa segmentation and paging.
  - Bước 1, kết hợp segment và offset thành địa chỉ tuyến tính (linear address).





Tổ chức bộ nhớ Intel x86 (2/2)

- Bước 2, chuyển địa chỉ tuyến tính thành địa chỉ vật lý (physical address).