



Chương 5 - Quản lý bộ nhớ

cuu duong than cong. com

- Tổng quan
 - Nhu cầu bộ nhớ của tiến trình
 - Các vấn đề về bộ nhớ
- Chuyển đổi địa chỉ
 - Các công đoạn
 - Các mô hình chuyển đổi địa chỉ
- Vai trò Quản lý bộ nhớ của HĐH
 - Các yêu cầu
- Các mô hình tổ chức bộ nhớ
 - Mô hình Liên tục
 - Mô hình Không liên tục

Tổng quan

- 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
 - **Ai cấp phát BNC cho tiến trình ?**
- Chương trình nguồn sử dụng địa chỉ symbolic
 - Tiến trình thực thi truy cập địa chỉ thực trong BNC
 - **Ai chuyển đổi địa chỉ ?**

HDH
Bộ phận Quản lý Bộ nhớ

Mô hình tổ chức ?

Cơ chế hỗ trợ

Chiến lược thực hiện



Tổng quan : Các vấn đề về Bộ nhớ

- Cấp phát Bộ nhớ :
 - Uniprogramming : Không khó
 - **Multiprogramming :**
 - 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
 - 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 ?

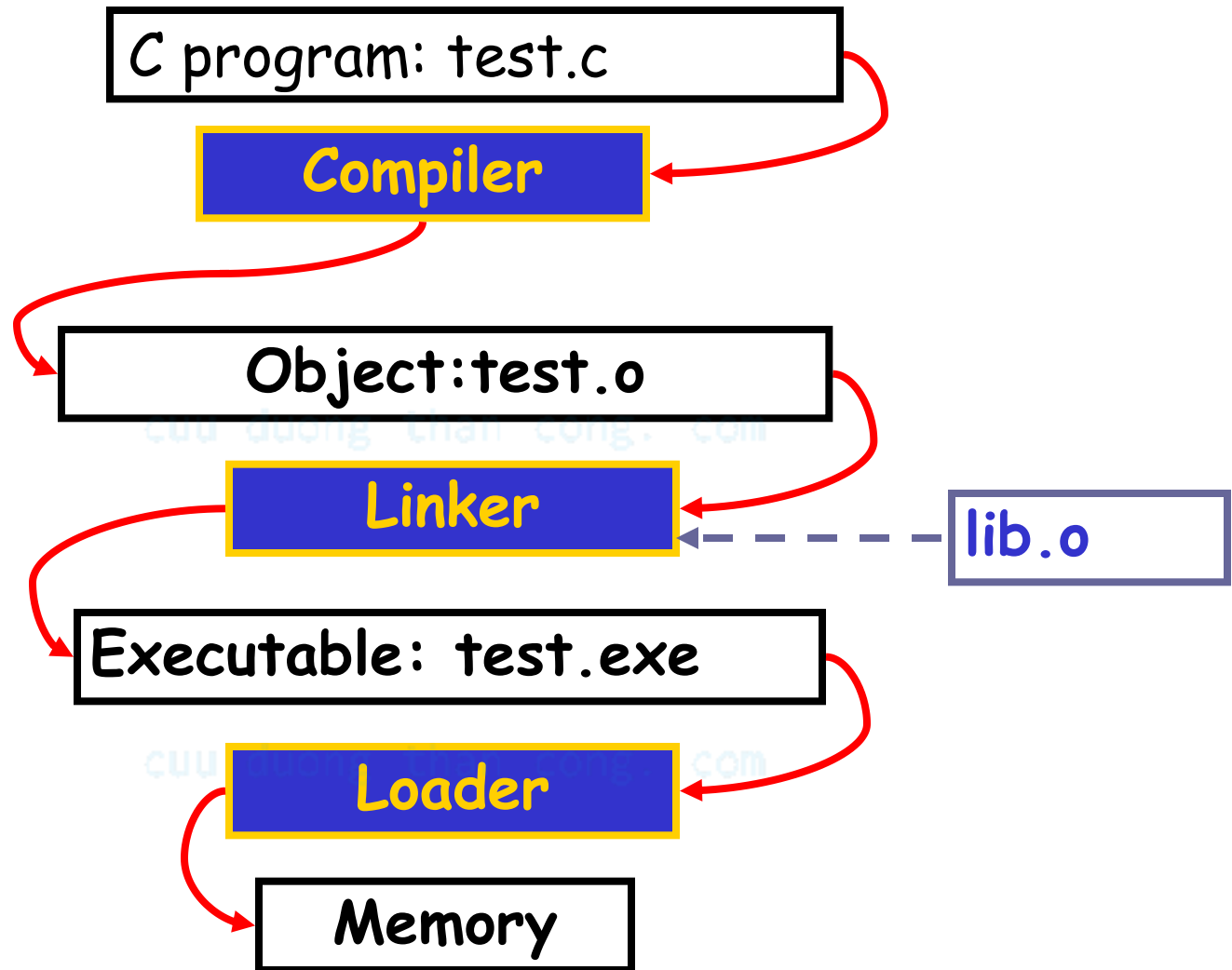
Ví dụ

Môi trường đa nhiệm

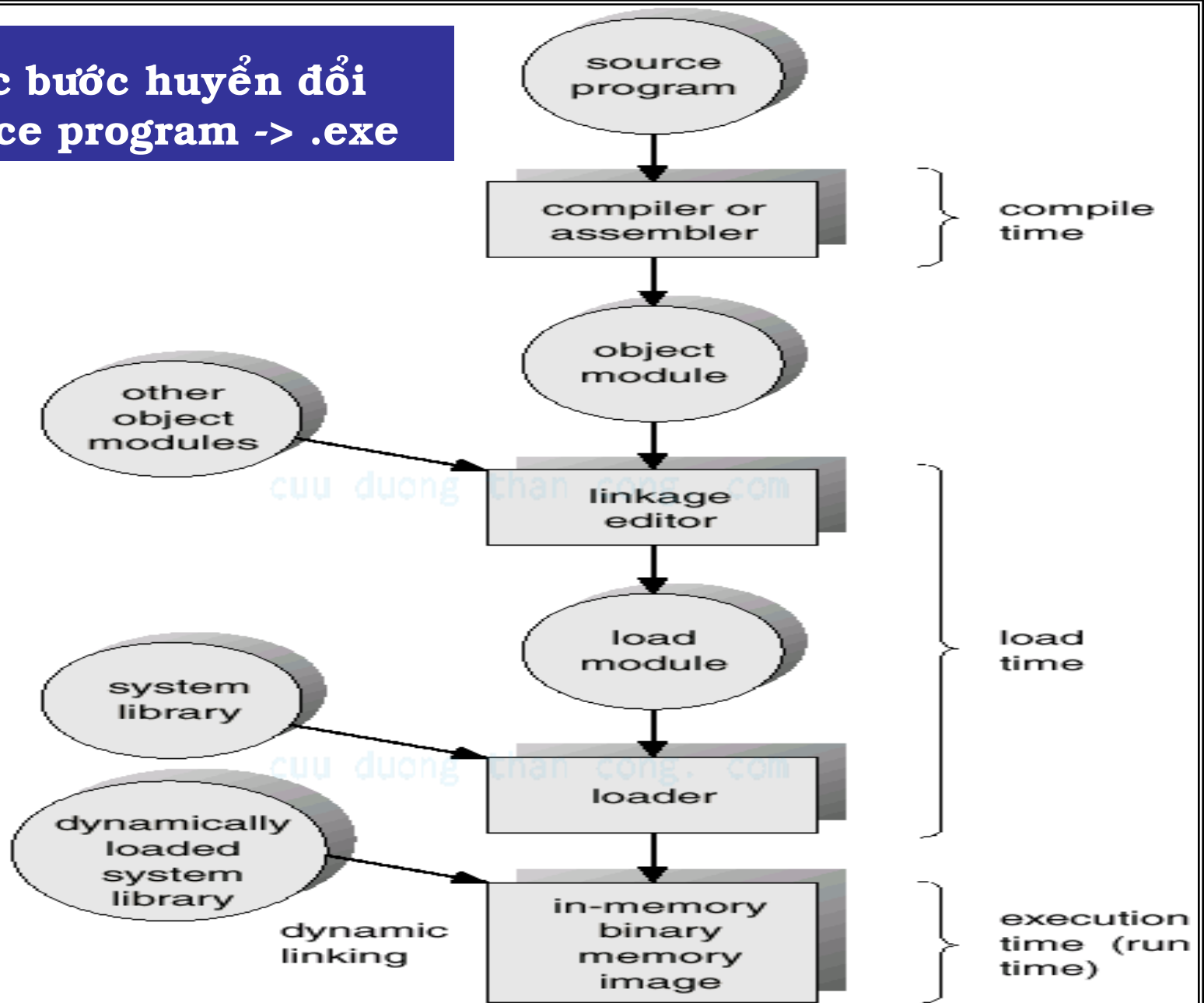
OS	0x9000
gcc	0x7000
nachos	0x4000
emacs	0x3000
	0x0000

- Nếu **nachos** cần thêm không gian ?
- Nếu **nachos** có lỗi và thực hiện thao tác ghi vào địa chỉ 0x7100?
- Khi nào **gcc** biết rằng nó thường trú tại 0x4000?
- Nếu **emacs** cần nhiều bộ nhớ hơn dung lượng vật lý hiện có?

Các bước chuyển đổi chương trình



Các bước chuyển đổi source program -> .exe



A.C

```
int x;
int y;
x = 12;
y = 5;
F();
```

B.C

```
F()
{
printf("Hi");
}
```

A.O

```
0 // x
2 // y
4 // [0] = 12;
5 // [2] = 5;
6 // jmp F
  // external
  // object
```

B.O

```
0 -2 // F() ...
```

```
0 // F()
3 // x
5 // y
7 // [3] = 12;
8 // [5] = 5;
9 // jmp 0
```

Test.exe

OS

```
? // F()
? // x
? // y
? // [?] = 12;
? // [?] = 5;
? // jmp ?
```


- **Địa chỉ logic** – còn gọi là địa chỉ ảo , là tất cả các địa chỉ do bộ xử lý tạo ra
- **Địa chỉ physic** - là địa chỉ thực tế mà trình quản lý bộ nhớ nhìn thấy và thao tác
- **Không gian địa chỉ** – là tập hợp tất cả các địa chỉ ảo phát sinh bởi một chương trình
- **Không gian vật lý** – là tập hợp tất cả các địa chỉ vật lý tương ứng với các địa chỉ ảo

Nhu cầu bộ nhớ của tiến trình

Tiến trình gồm có:

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

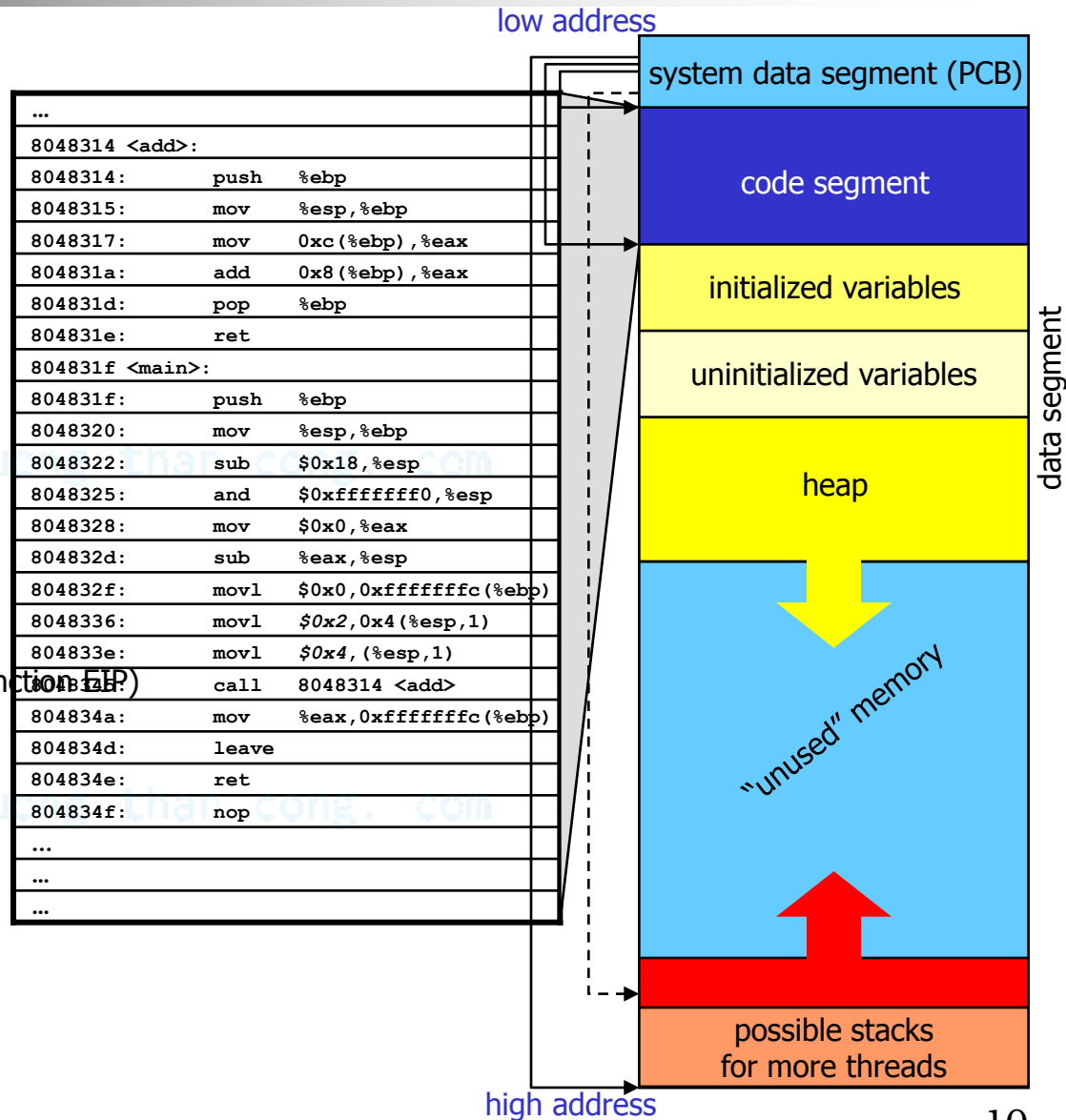
stack segment

- variables in a function
- stored register states (e.g. calling function arguments)
- grows against lower addresses

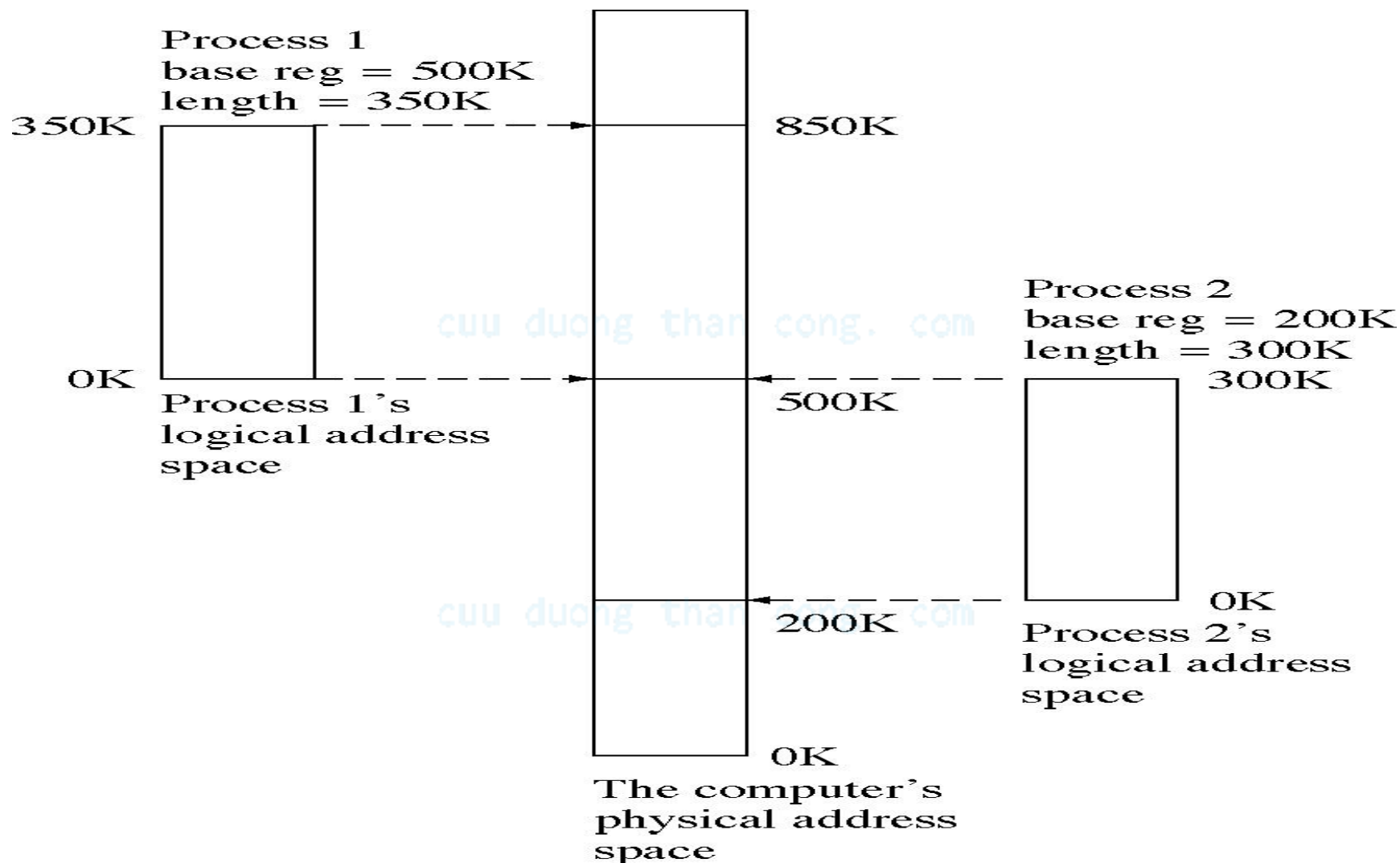
system data segment (PCB)

- segment pointers
- pid
- program and stack pointers
- ...

Stack cho các thread



Logical and Physical Address Spaces



Truy xuất bộ nhớ

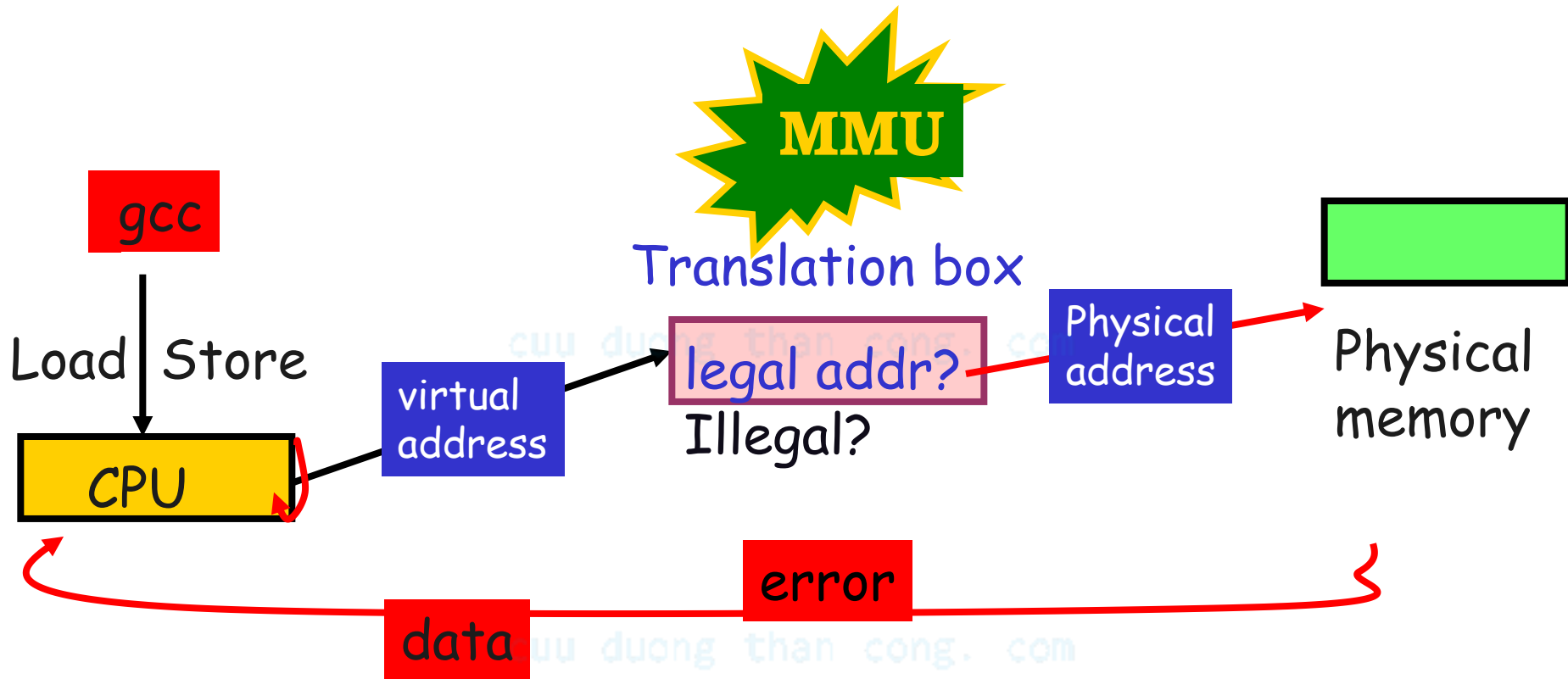
- Địa chỉ của Instruction và data trong program source code là symbolic:
 - `goto errjmp;`
 - `X = A + B;`
- Những địa chỉ symbolic này cần được liên kết (bound) với các địa chỉ thực trong bộ nhớ vật lý trước khi thi hành code
- **Address binding**: ánh xạ địa chỉ từ không gian địa chỉ (KGĐC) này vào KGĐC khác
- Thời điểm thực hiện address binding ?
 - compile time
 - load time
 - execution time.



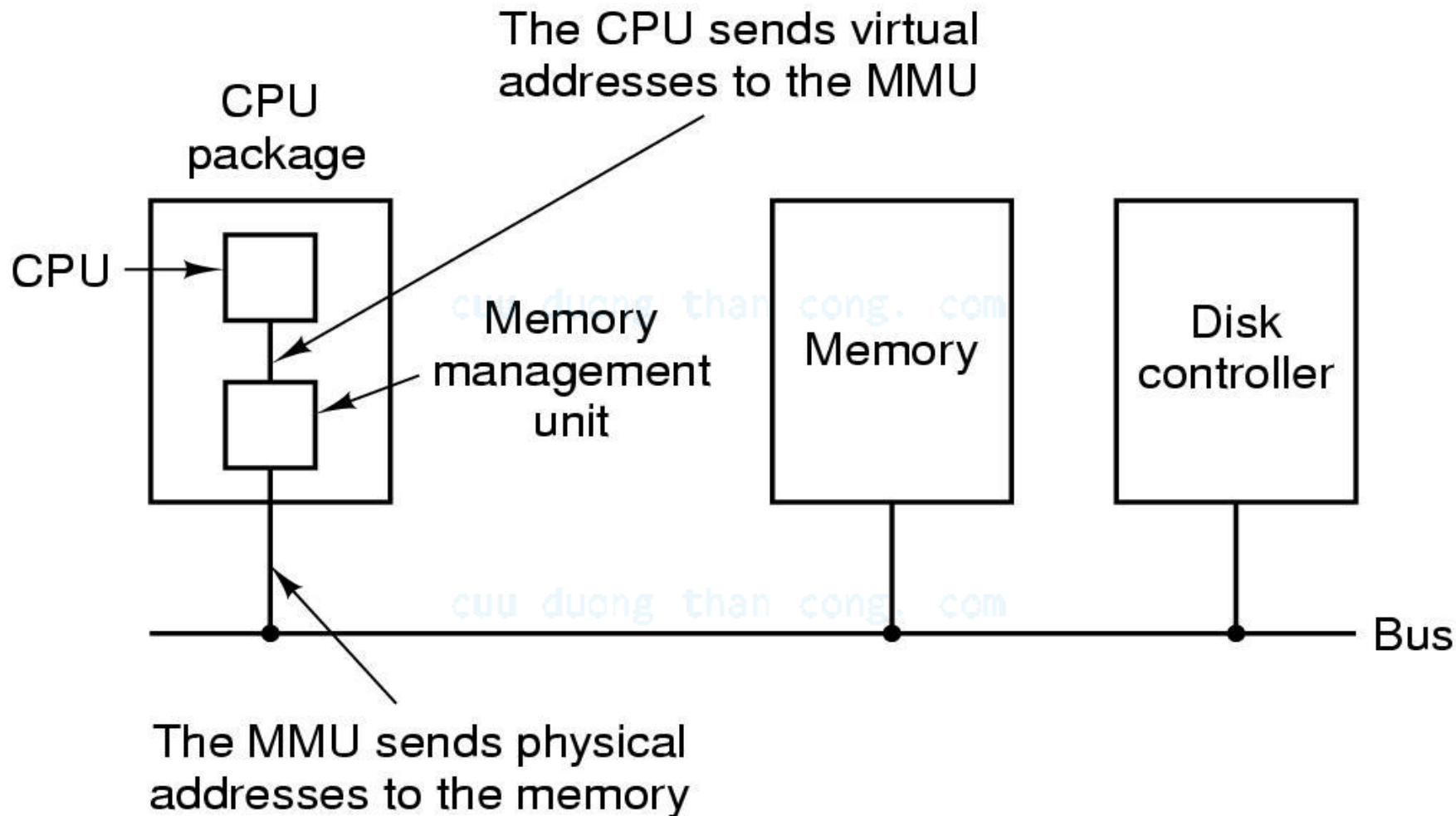
Thời điểm kết buộc địa chỉ ?

- Có thể thực hiện việc kết buộc địa chỉ tại 1 trong 3 thời điểm :
 - **Compile-time:**
 - Phát sinh địa chỉ tuyệt đối
 - 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 tương đối
 - Khi nạp, biết vị trí bắt đầu sẽ tính lại địa chỉ tuyệt đối
 - Phải tái nạp khi vị trí bắt đầu thay đổi
 - **Execution-time:**
 - Khi biên dịch, nạp chỉ phát sinh địa chỉ tương đối
 - Trì hoãn thời điểm kết buộc địa chỉ tuyệt đối đến khi thi hành
 - Khi đó ai tính toán địa chỉ tuyệt đối ?
 - Phần cứng : **MMU**

Chuyển đổi địa chỉ



CPU, MMU and Memory





Yêu cầu quản lý bộ nhớ

- Tăng hiệu suất sử dụng CPU
 - Cần hỗ trợ Multiprogramming
 - Lưu trữ cùng lúc nhiều tiến trình trong BNC ?
- Các yêu cầu khi tổ chức lưu trữ tiến trình:
 1. Relocation
 2. Protection
 3. Sharing
 4. Logical Organization
 5. Physical Organization



Tái định vị (Relocation)

- Không biết trước chương trình sẽ được nạp vào BN ở vị trí nào để xử lý.
- Một tiến trình có thể được di dời trong bộ nhớ sau khi đã nạp C
 - Tiến trình tăng trưởng ?
 - HĐH sắp xếp lại các tiến trình để có thể sử dụng BNC hiệu quả hơn.

cuu duong than cong. com



Bảo vệ (Protection)

- Không cho phép tiến trình truy cập đến các vị trí nhớ đã cấp cho tiến trình khác (khi chưa có phép).
- Không thể thực hiện việc kiểm tra hợp lệ tại thời điểm biên dịch hay nạp, vì chương trình có thể được tái định vị.
- Thực hiện kiểm tra tại thời điểm thi hành
 - Cần sự hỗ trợ của phần cứng.



Chia sẻ (Sharing)

- Cần cho phép nhiều tiến trình tham chiếu đến cùng một vùng nhớ mà không tổn hại đến tính an toàn hệ thống :
 - Tiết kiệm chỗ lưu trữ cho các module dùng chung.
 - Cho phép các tiến trình cộng tác có khả năng chia sẻ dữ liệu.

[cuu duong than cong. com](http://cuuduongthancong.com)



Tổ chức logic (Logical Organization)

- Người dùng viết chương trình gồm nhiều module, với các yêu cầu bảo vệ cho từng module có thể khác nhau:
 - instruction modules : execute-only.
 - data modules : read-only hay read/write.
 - một số module là private, số khác có thể là public.
- OS cần hỗ trợ các cơ chế có thể phản ánh mô hình logic của chương trình



Tổ chức vật lý (Physical Organization)

- Cấp phát vùng nhớ vật lý sao cho hiệu quả
- Và dễ dàng chuyển đổi chương trình qua lại giữa BNChính và BNPhụ

cuu duong than cong. com

cuu duong than cong. com



Các mô hình tổ chức bộ nhớ

- **Cấp phát Liên tục (Contiguous Allocation)**
 - Linker – Loader
 - Base & Bound
- **Cấp phát Không liên tục (Non Contiguous Allocation)**
 - Segmentation
 - Paging

cuu duong than cong. com

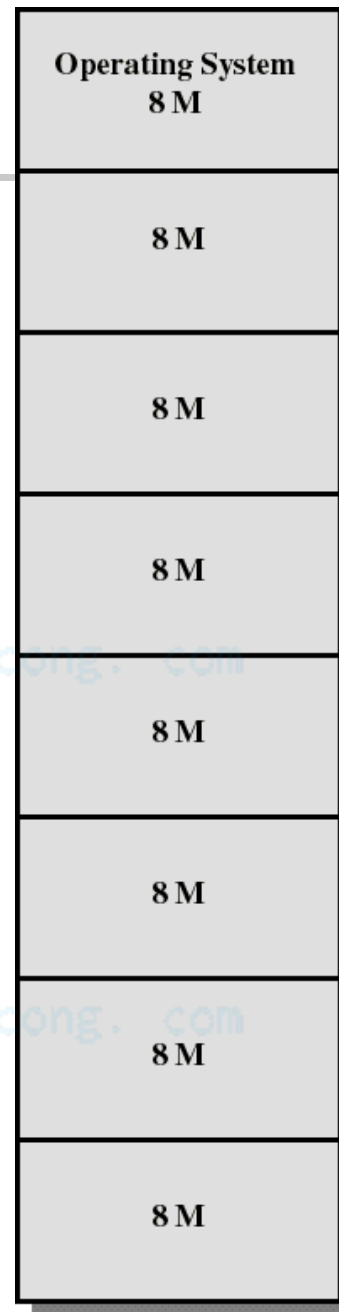


Cấp phát Liên tục (Contiguous Allocation)

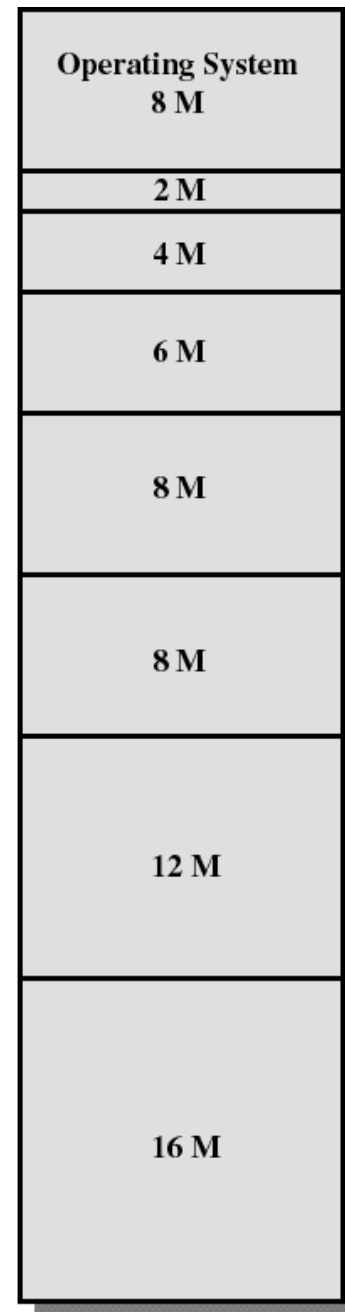
- Nguyên tắc :
 - Chương trình được nạp toàn thể vào BNC để thi hành
 - Cần một vùng nhớ liên tục, đủ lớn để chứa Chương trình
- Không gian địa chỉ : liên tục
- Không gian vật lý : có thể tổ chức
 - Fixed partition
 - Variable partition
- 2 mô hình đơn giản
 - Linker – Loader
 - Base & Bound

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 ?



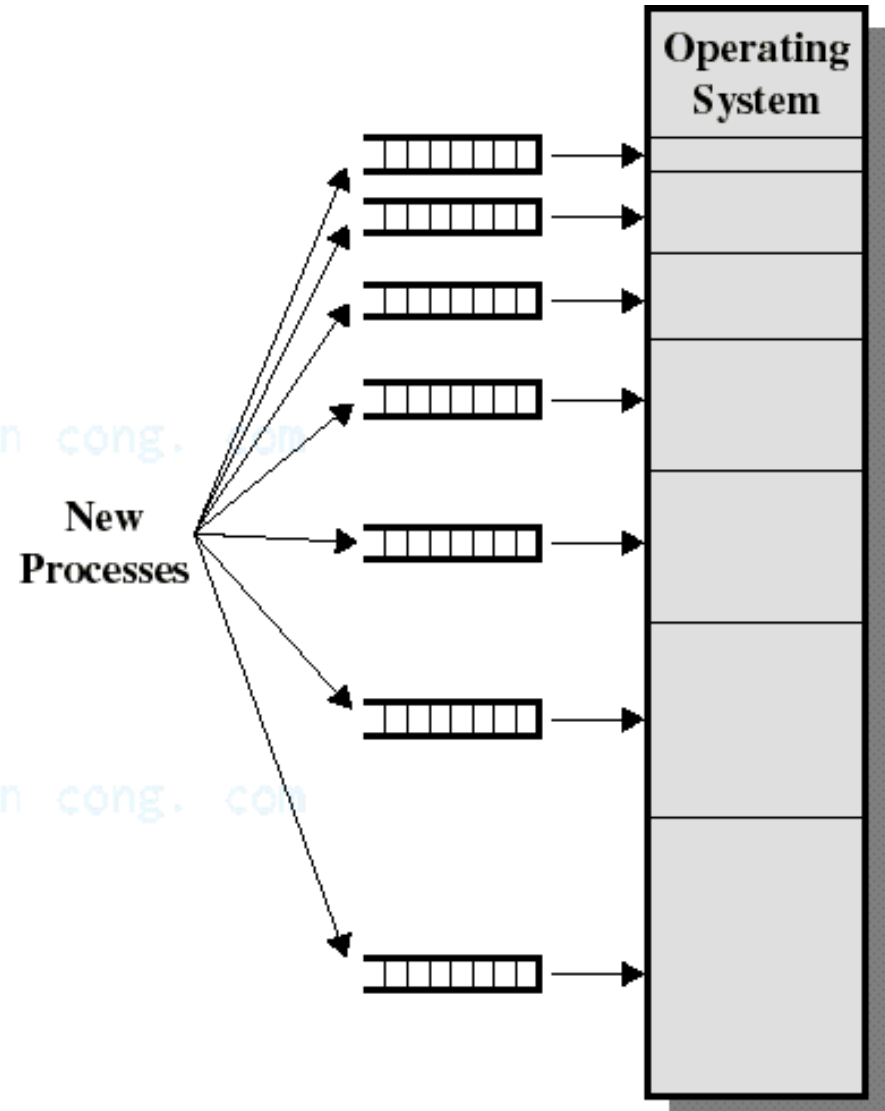
Equal-size partitions



Unequal-size partitions

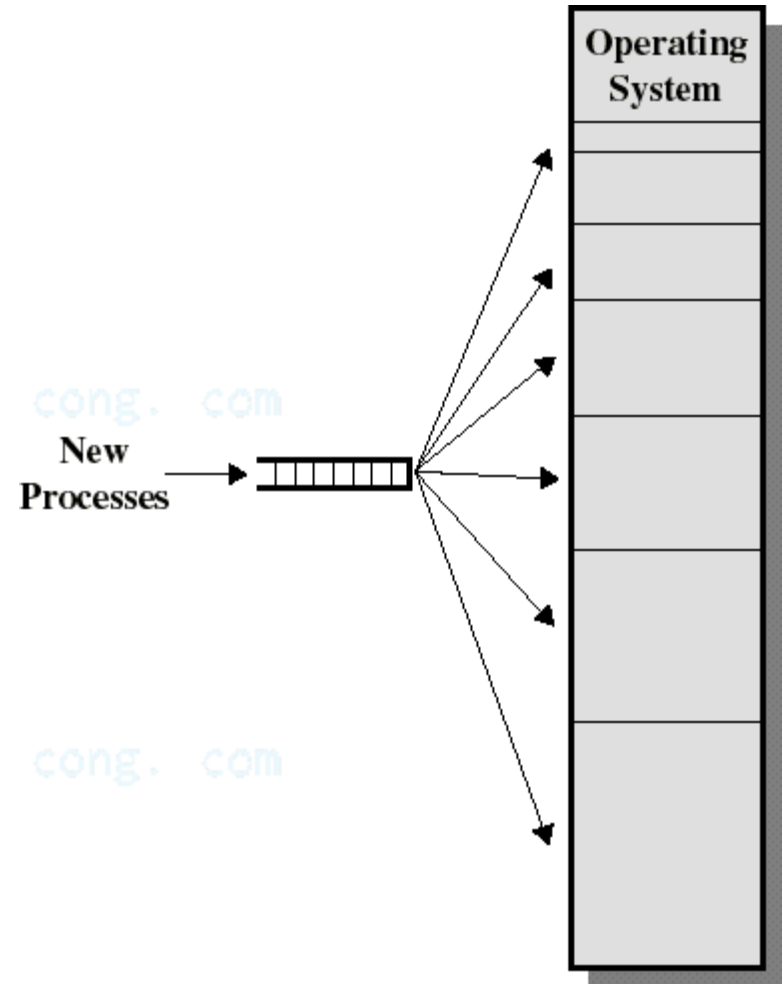
Chiến lược cấp phát partitions cho tiến trình

- Kích thước partition bằng nhau
 - không có gì phải suy nghĩ !
- Kích thước partition không bằng nhau :
 - Sử dụng nhiều hàng đợi
 - Cấp cho tiến trình partition với kích thước bé nhất (đủ lớn để chứa tiến trình)
 - Khuyết điểm : phân bố các tiến trình vào các partition không đều, một số tiến trình phải đợi trong khi có



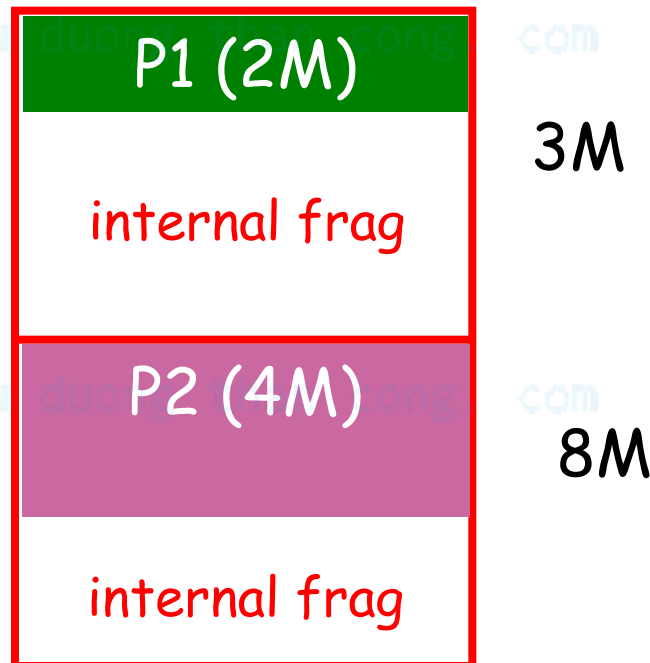
Chiến lược cấp phát partitions cho tiến trình

- Kích thước partition không bằng nhau :
 - Sử dụng 1 hàng đợi
 - Cấp cho tiến trình partition tự do với kích thước bé nhất (đủ lớn để chứa tiến trình)
 - Cần dùng một CTDL để theo dõi các partition tự do



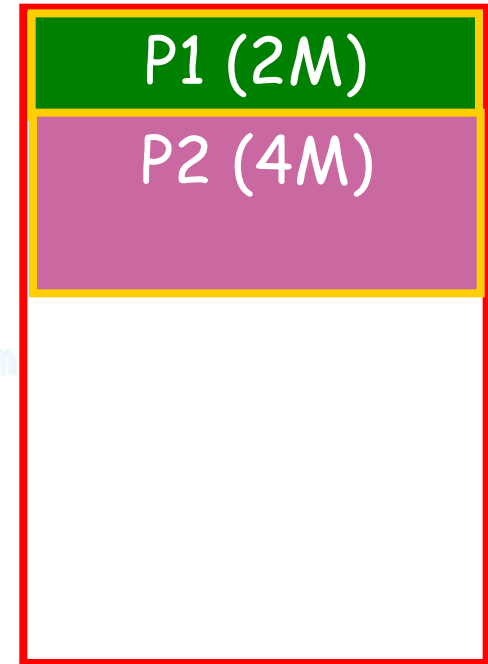
Nhận xét Fixed Partitioning

- 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

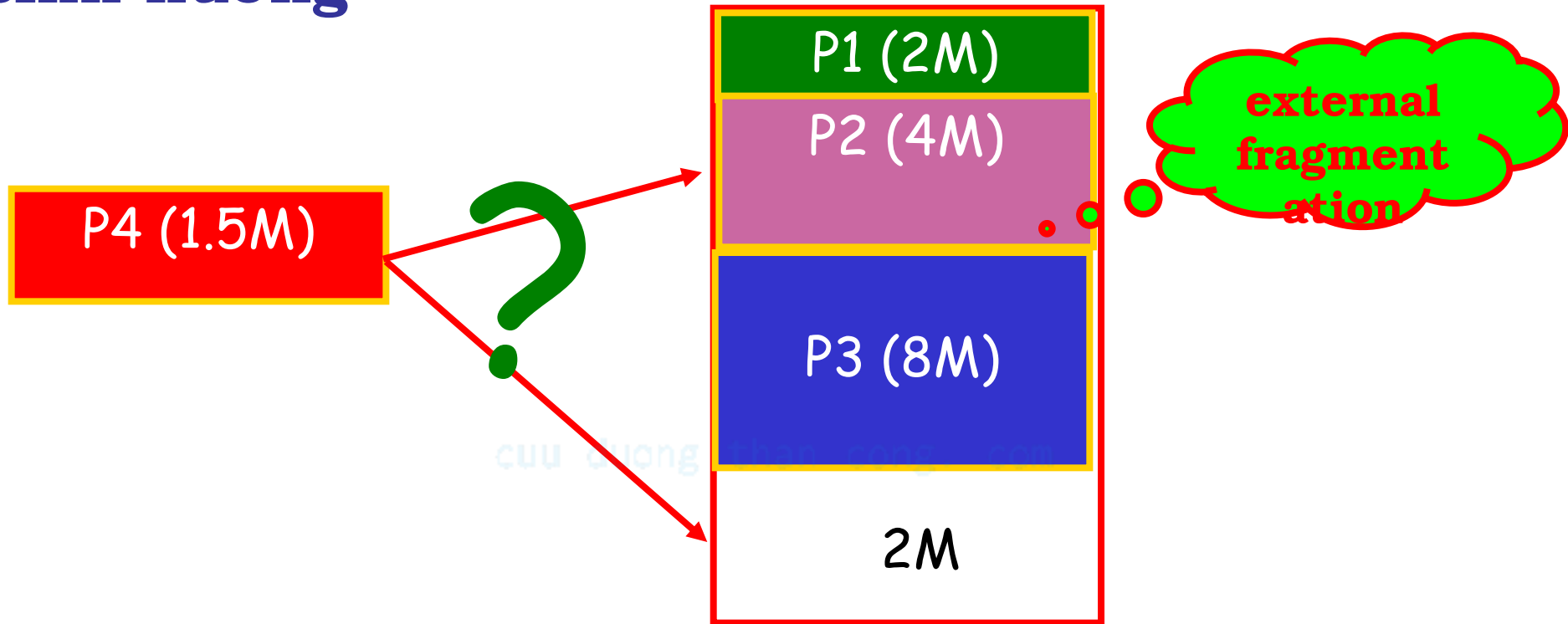


Dynamic Partitioning

- 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

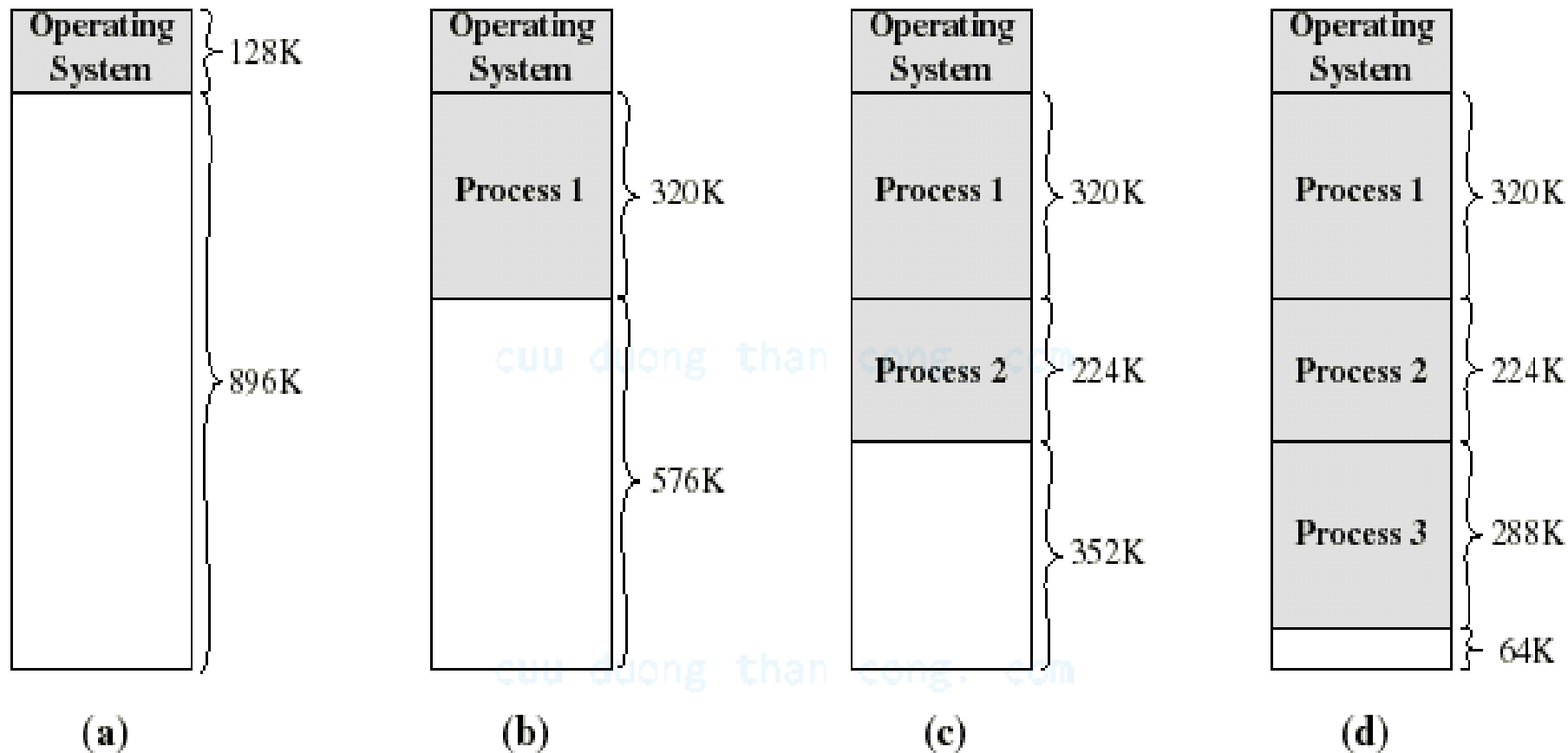


Dynamic Partitioning: tình huống

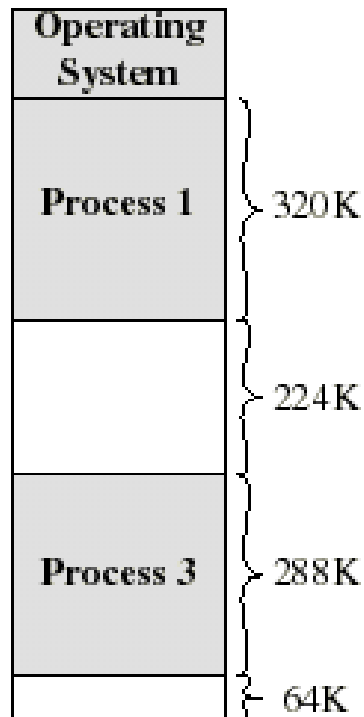


- Chọn lựa partition để cấp phát cho tiến trình ?
 - Đồng thời có nhiều partition tự do đủ lớn để chứa tiến trình
 - Dynamic Allocation problem
- Tiến trình vào sau không lấp đầy chỗ trống tiến trình trước để lại
 - external fragmentation

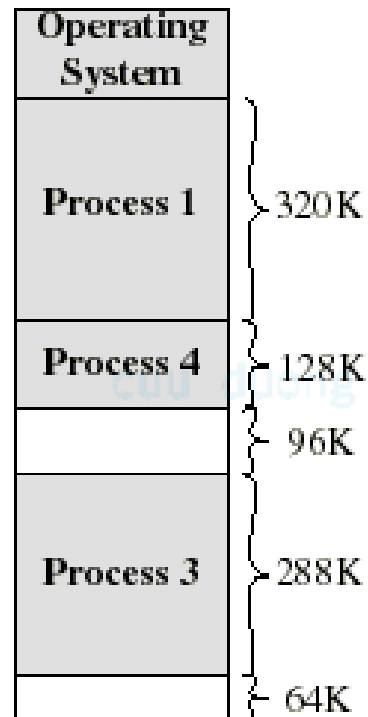
Ví dụ Dynamic Partitioning



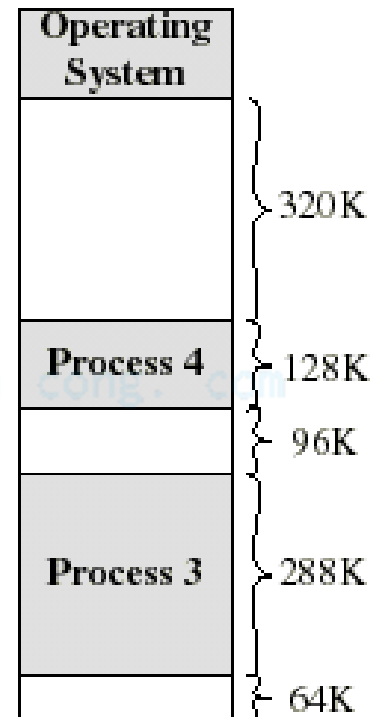
Ví dụ Dynamic Partitioning



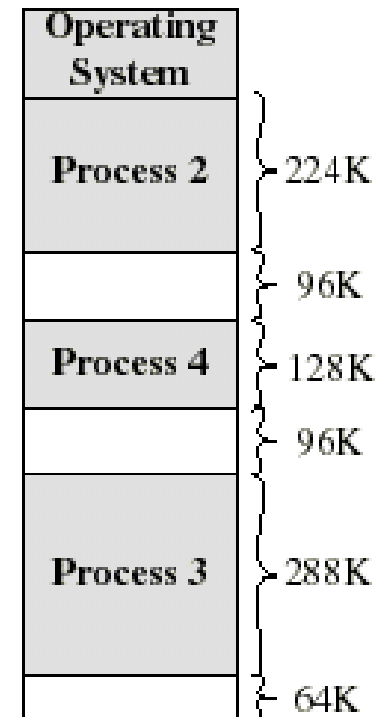
(e)



(f)



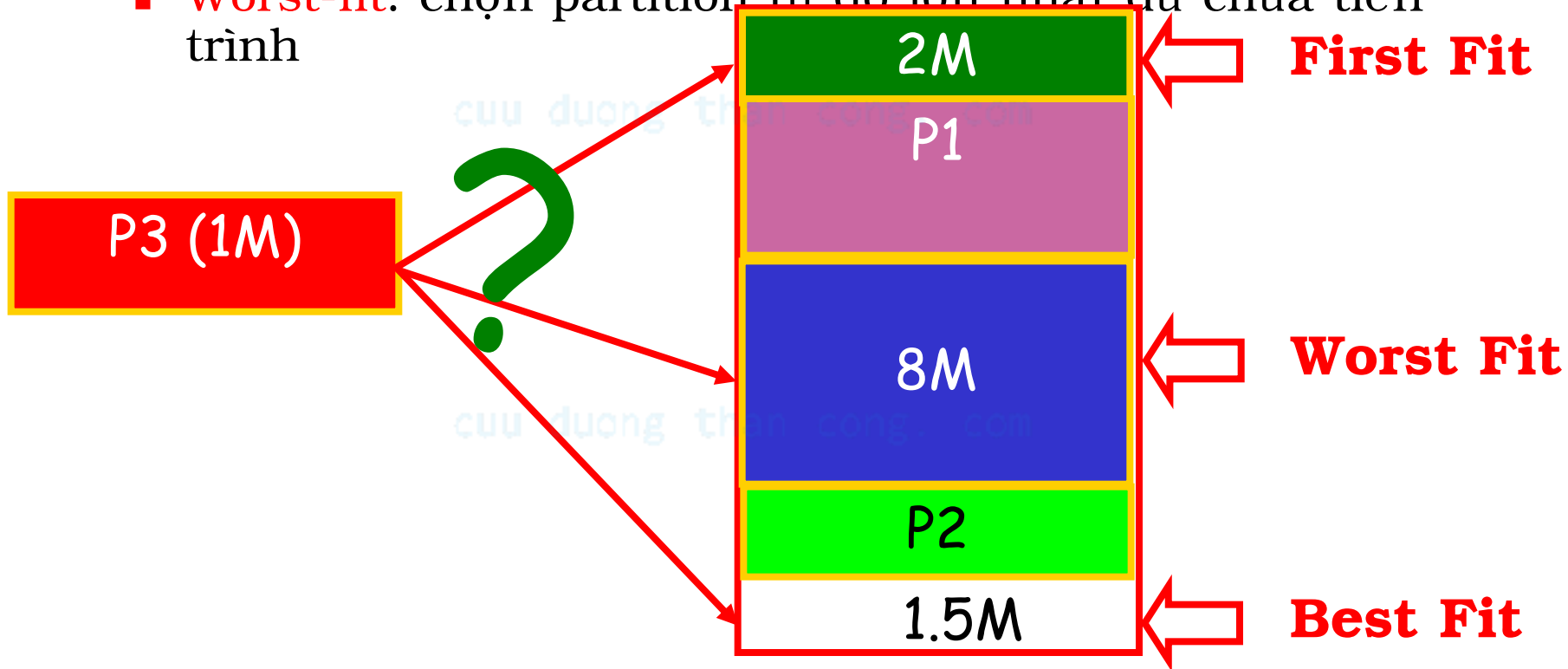
(g)



(h)

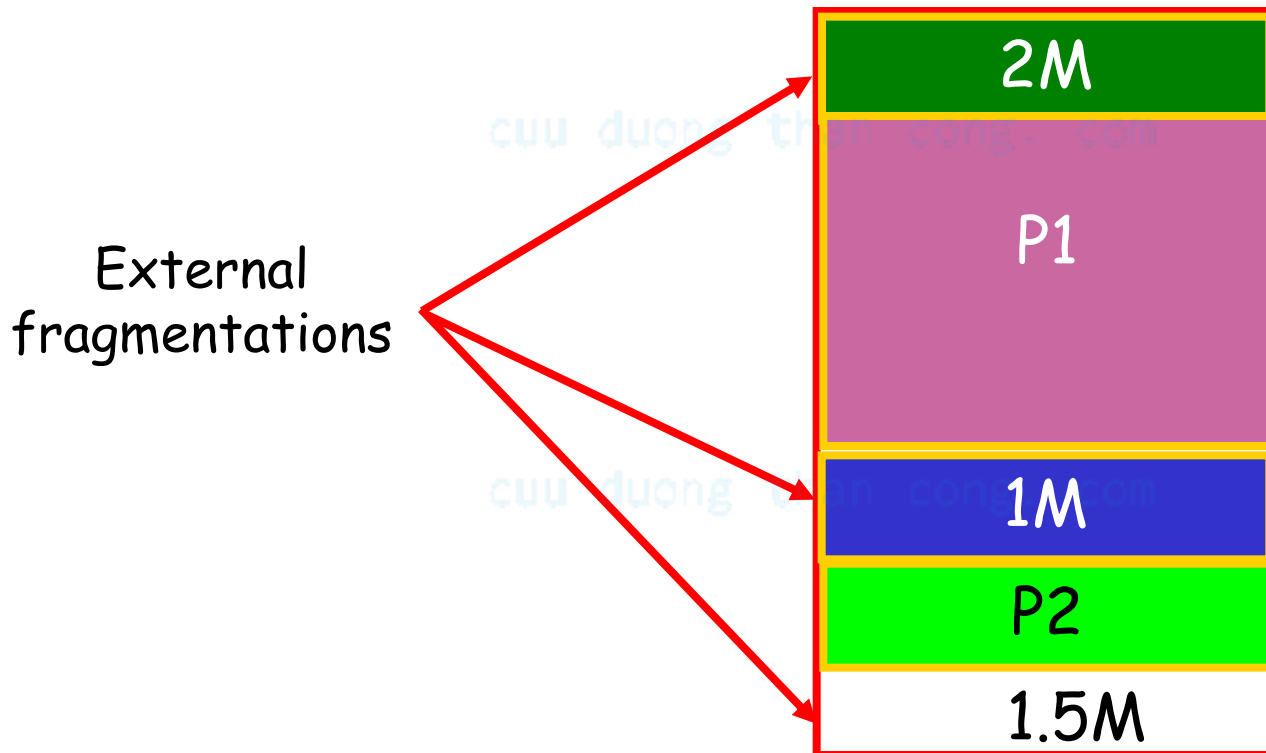
Giải quyết vấn đề Dynamic Allocation

- 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



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



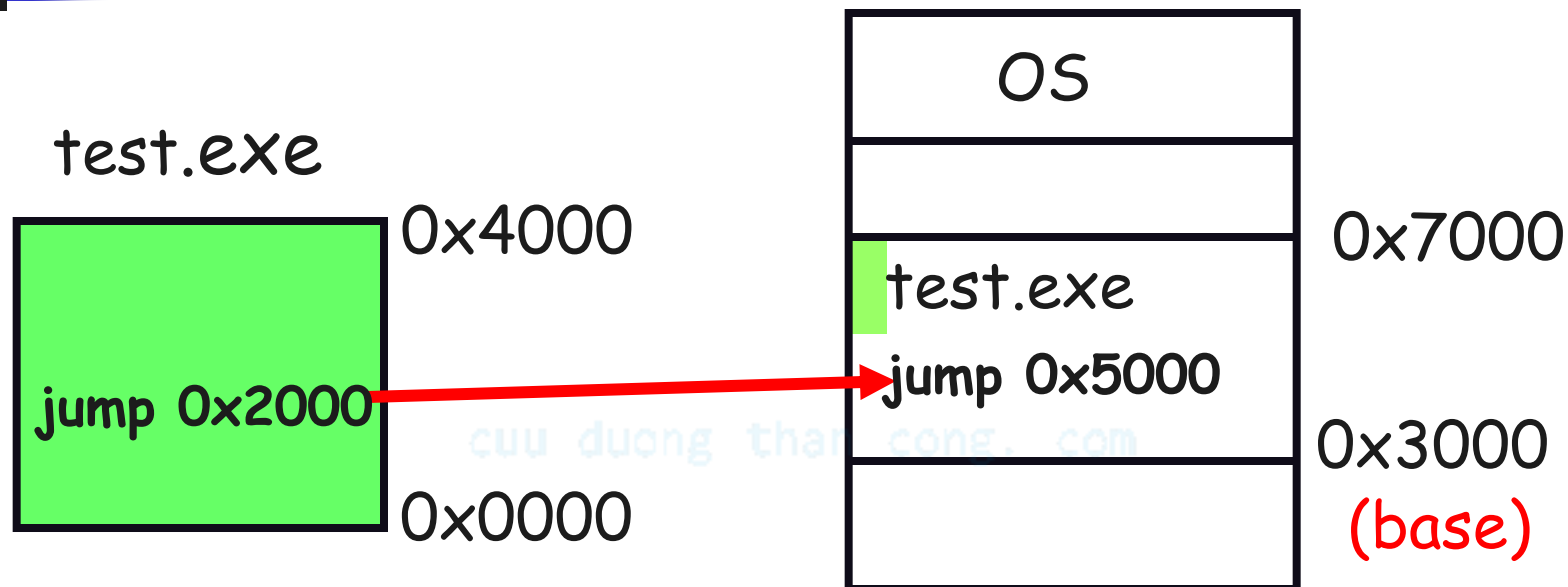


Các mô hình chuyển đổi địa chỉ

- Fixed/Dynamic partition là mô hình tổ chức nạp tiến trình vào KGVL
- Cần có mô hình để chuyển đổi địa chỉ từ KGĐC vào KGVL
 - Linker Loader
 - Base & Bound

cuu duong than cong. com

Mô hình Linker-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 :

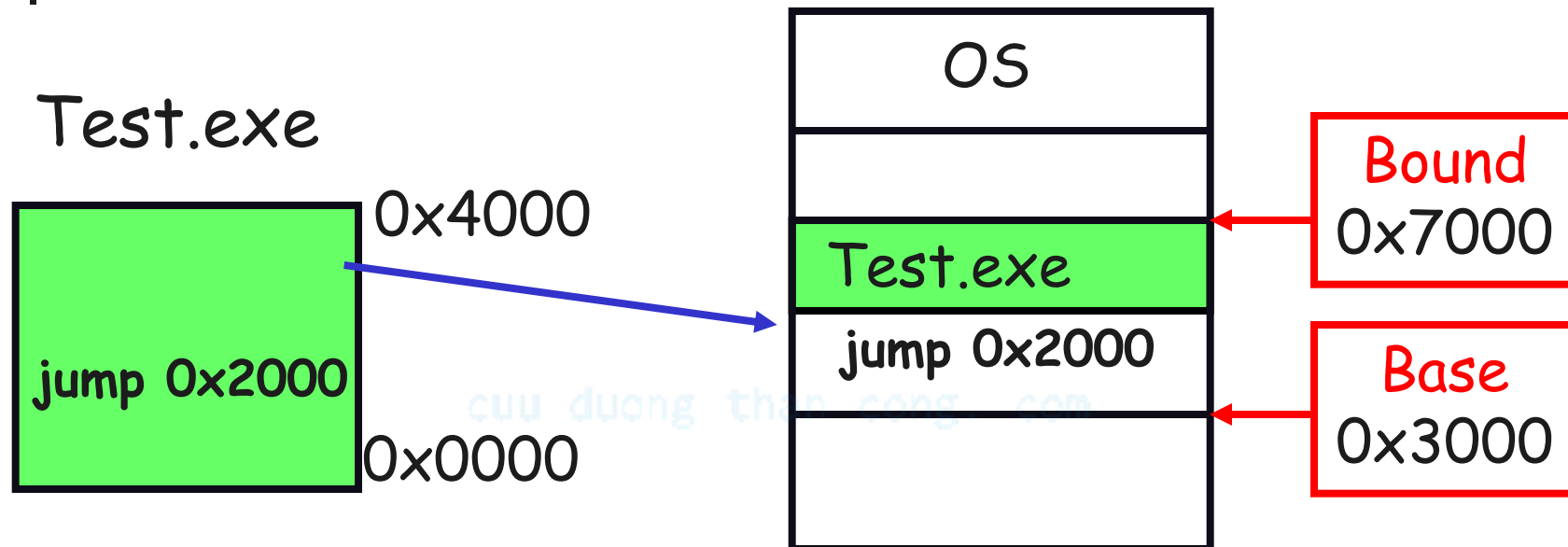
địa chỉ physic = địa chỉ logic + base



Nhận xét mô hình Linker-Loader

- **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 !

Mô hình 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

- Bảo vệ : **địa chỉ hợp lệ \subseteq [base, bound]**

Nhận xét mô hình Base & Bound

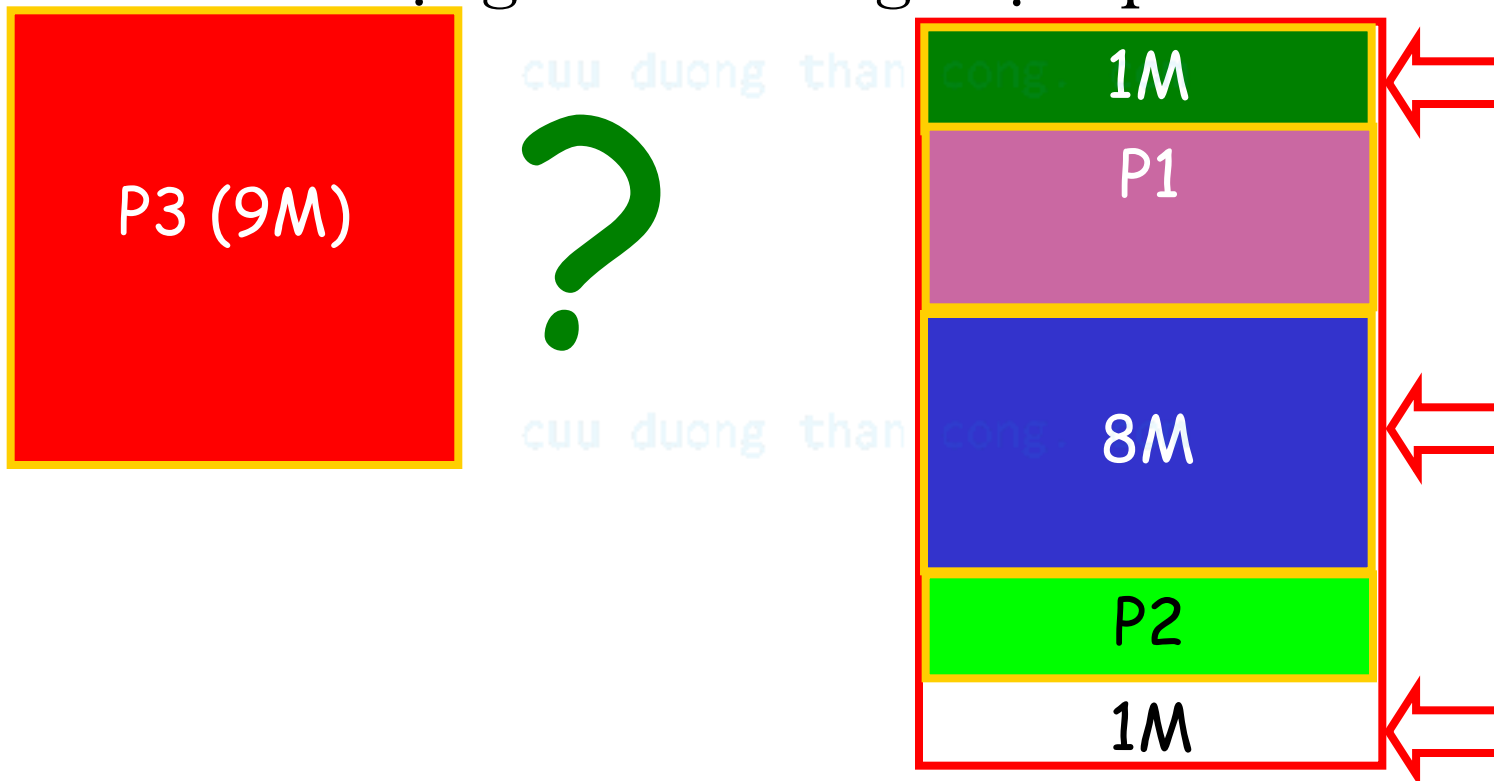
- Kết buộc địa chỉ tại thời điểm thi hành => cần hỗ trợ của phần cứng



- Hỗ trợ
 - Bảo vệ
 - Tái định vị

Khuyết điểm của cấp phát liên tục

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





Các mô hình tổ chức bộ nhớ

- Cấp phát Liên tục (Contiguous Allocation)
 - Linker – Loader
 - Base & Bound
- Cấp phát Không liên tục (Non Contiguous Allocation)
 - Segmentation
 - Paging

cuu duong than cong. com



Các mô hình cấp phát không liên tục

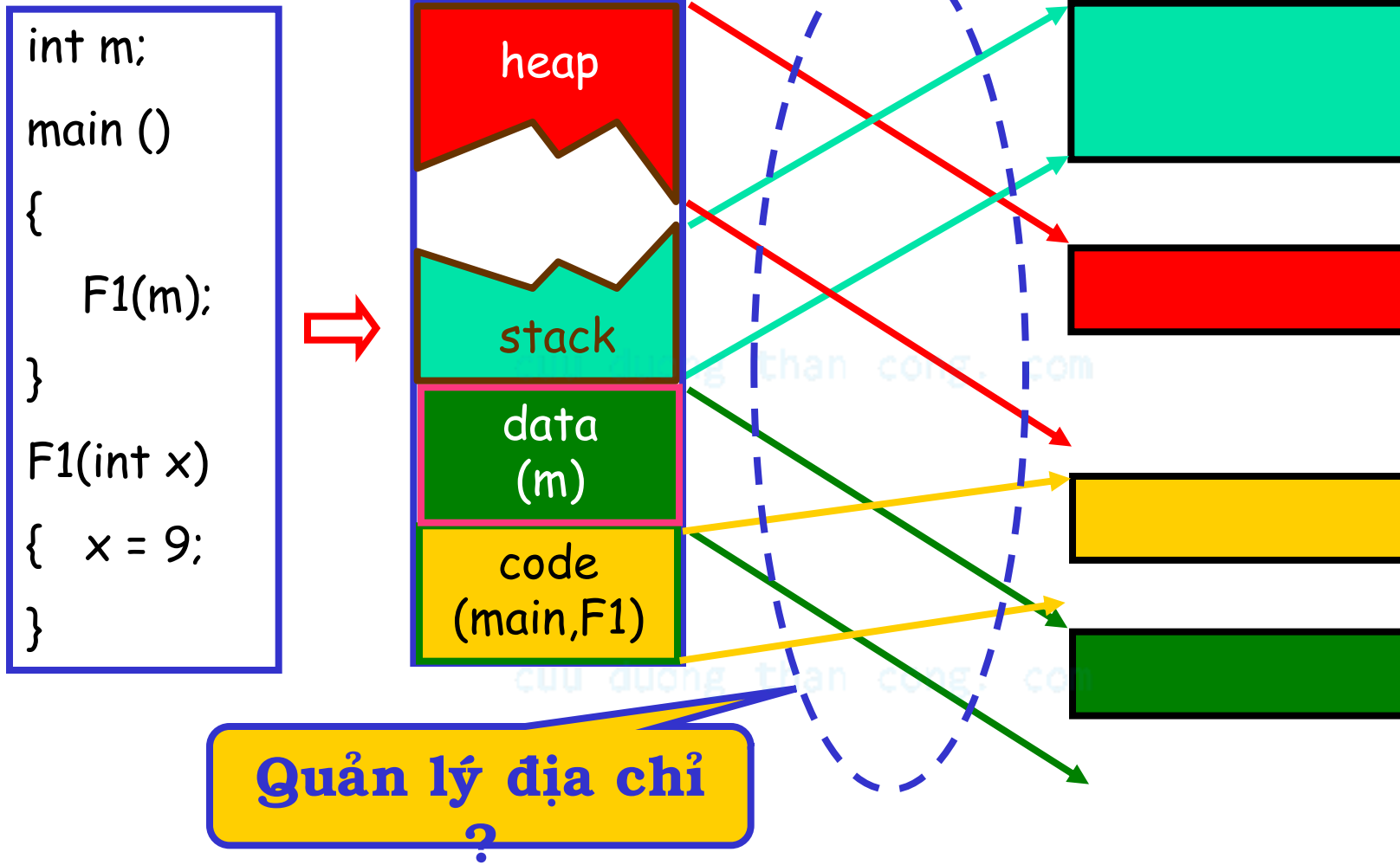
- Cho phép nạp tiến trình vào BNC ở nhiều vùng nhớ không liên tục
- Không gian địa chỉ : phân chia thành nhiều partition
 - Segmentation
 - Paging
- Không gian vật lý : có thể tổ chức
 - Fixed partition : Paging
 - Variable partition : Segmentation



Segmentation

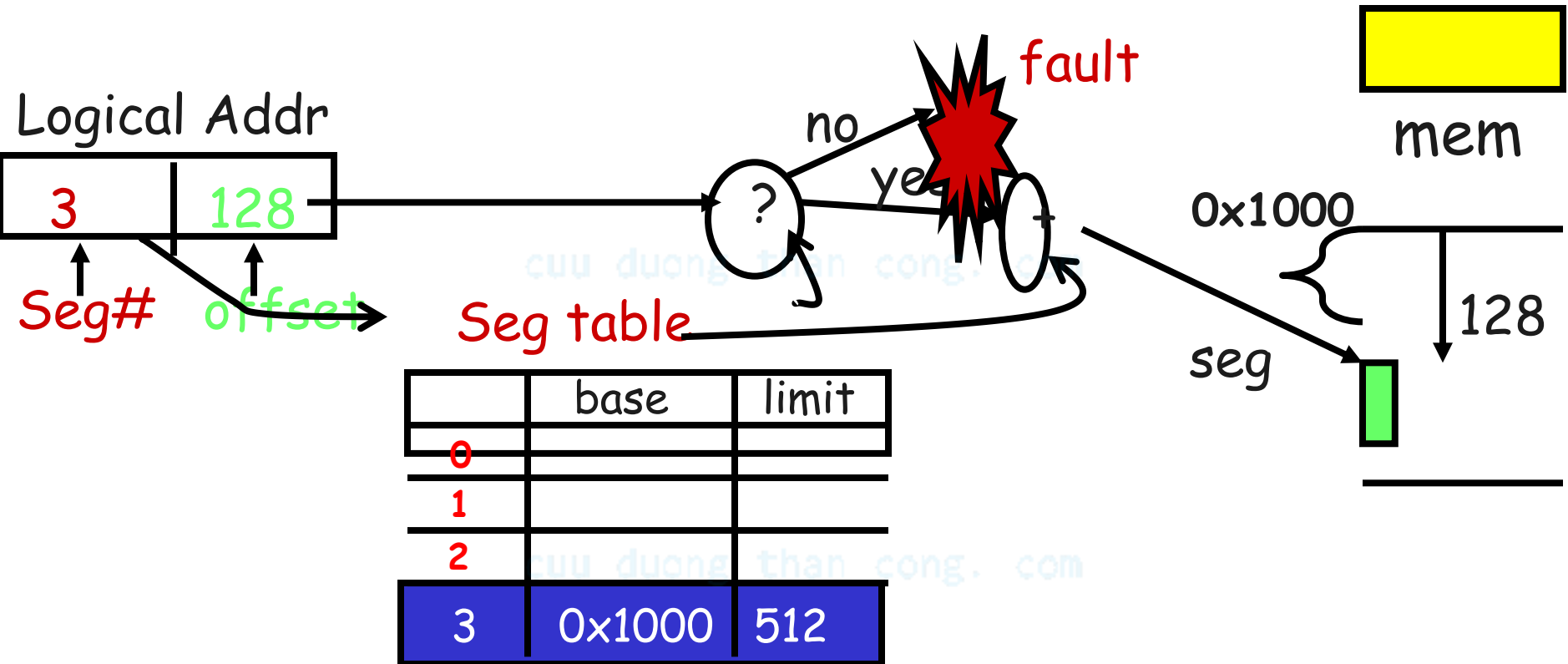
- Lập trình viên : chương trình là một tập các segments
 - Một segment là một đơn vị chương trình gồm các đối tượng có cùng nhóm ngữ nghĩa
 - Ví dụ : main program, procedure, function, local variables, global variables, common block, stack, symbol table, arrays...
 - Các segment có thể có kích thước khác nhau
- Mô hình 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

Mô hình Segmentation

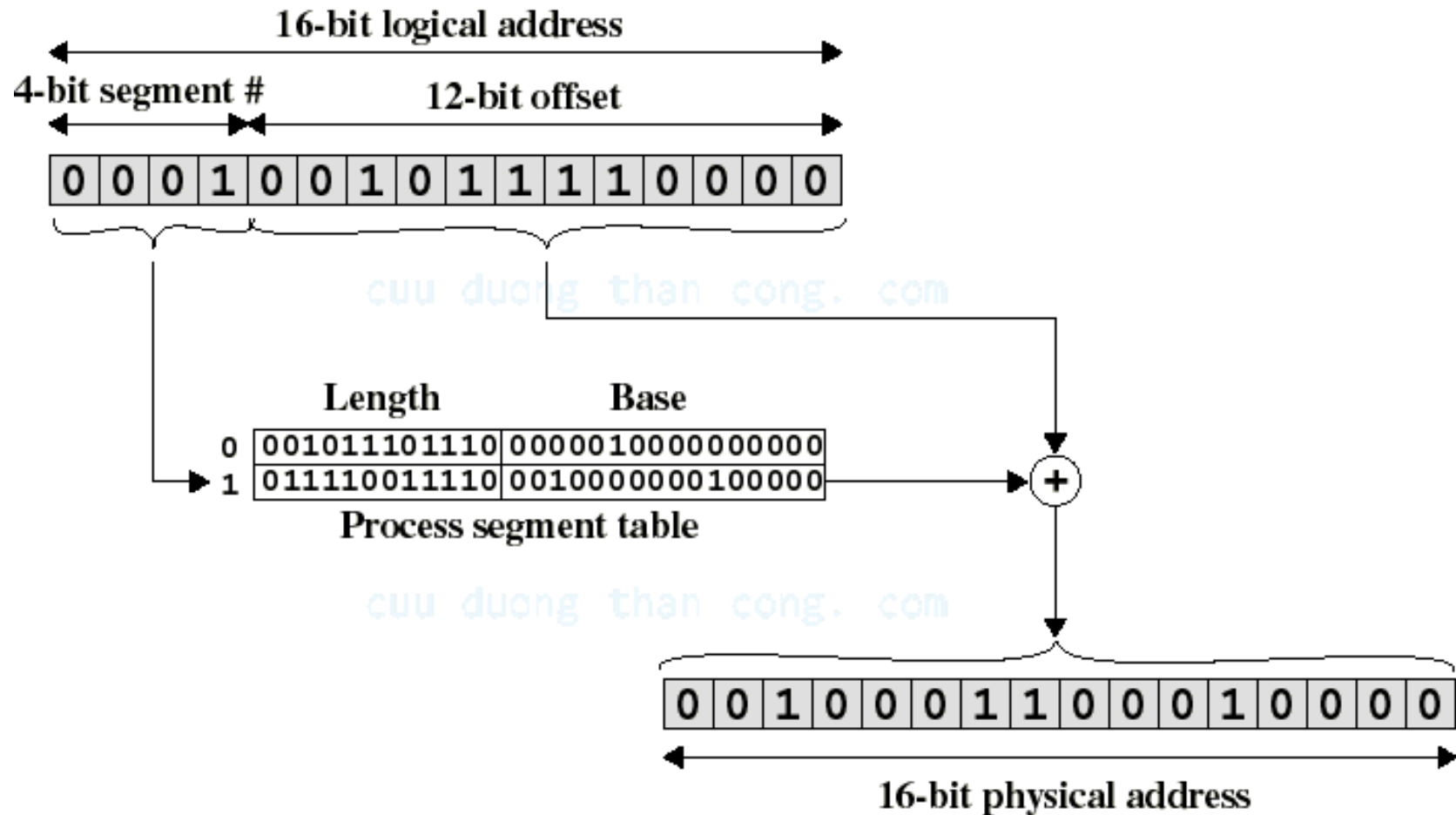


KGVL

Chuyển đổi địa chỉ trong mô hình Segmentation



Logical-to-Physical Address Translation in segmentation

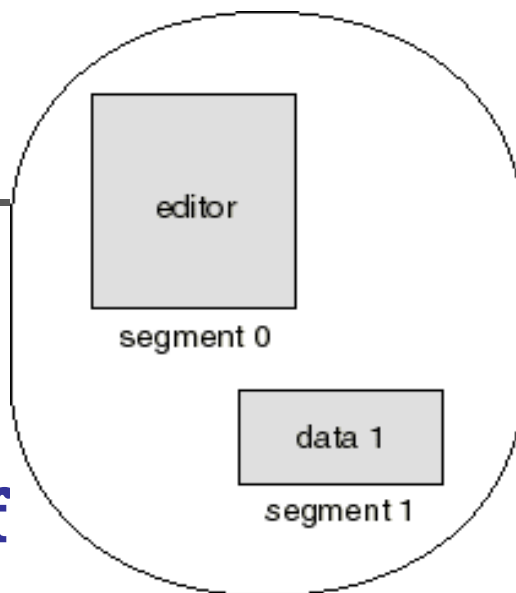


Nhận xét Mô hình Segmentation

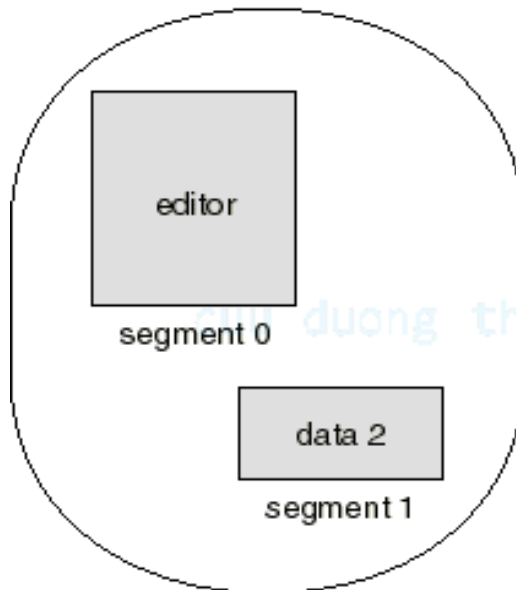
- Cấp phát không liên tục => tận dụng bộ nhớ hiệu quả
- Hỗ trợ tái định vị
 - Từng Segment
- Hỗ trợ Bảo vệ và Chia sẻ được ở mức module
 - Ý nghĩa của “mức module” ?
- ☹ Chuyển đổi địa chỉ phức tạp
 - ☺ Đã có MMU...
- ☹ Sử dụng dynamic partition : chịu đựng
 - ☹ **Dynamic Allocation** : chọn vùng nhớ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

Sharing of Segments

Text Editor



logical address space
process P_1



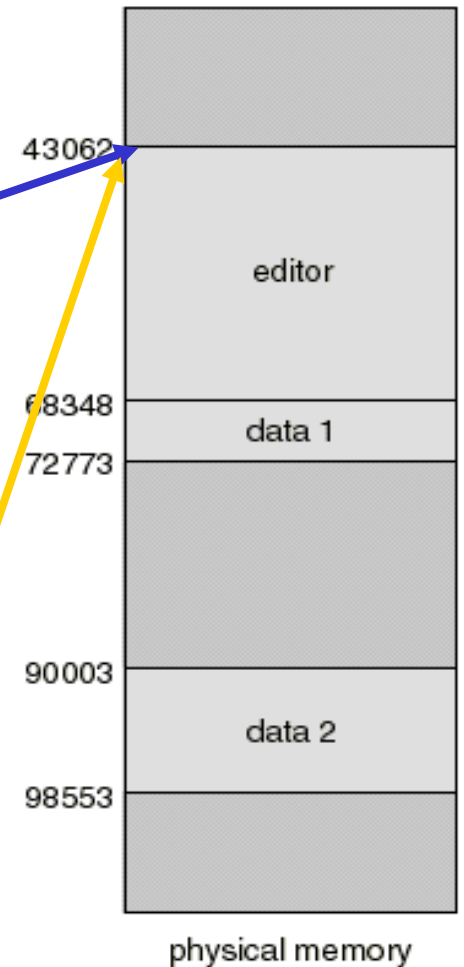
logical address space
process P_2

	limit	base
0	25286	43062
1	4425	68348

segment table
process P_1

	limit	base
0	25286	43062
1	8850	90003

segment table
process P_2

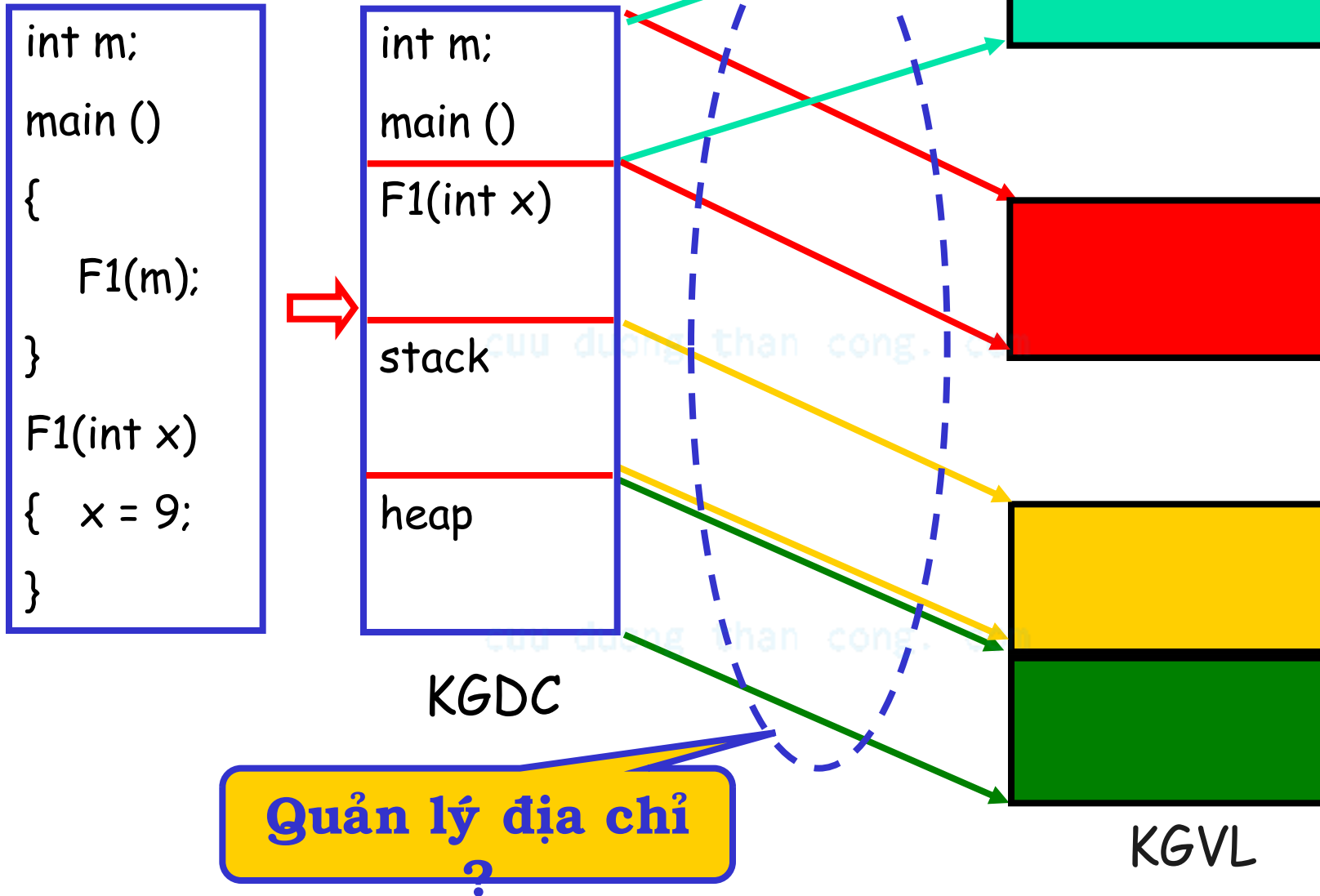




Paging

- Hỗ trợ HĐH khắc phục bài toán cấp phát bộ nhớ động, và loại bỏ external fragmentation
- Mô hình Paging :
 - **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

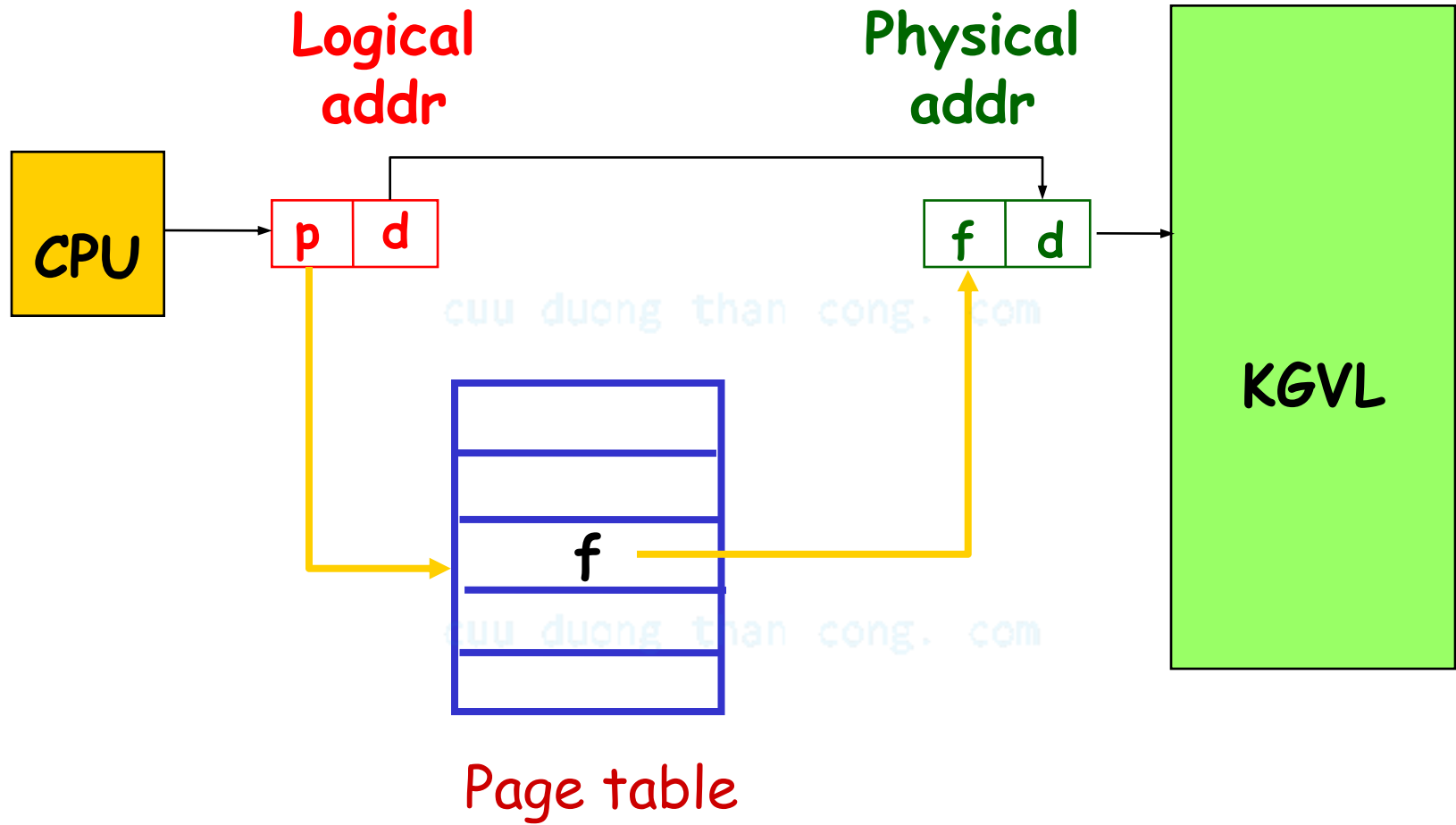
Mô hình Paging



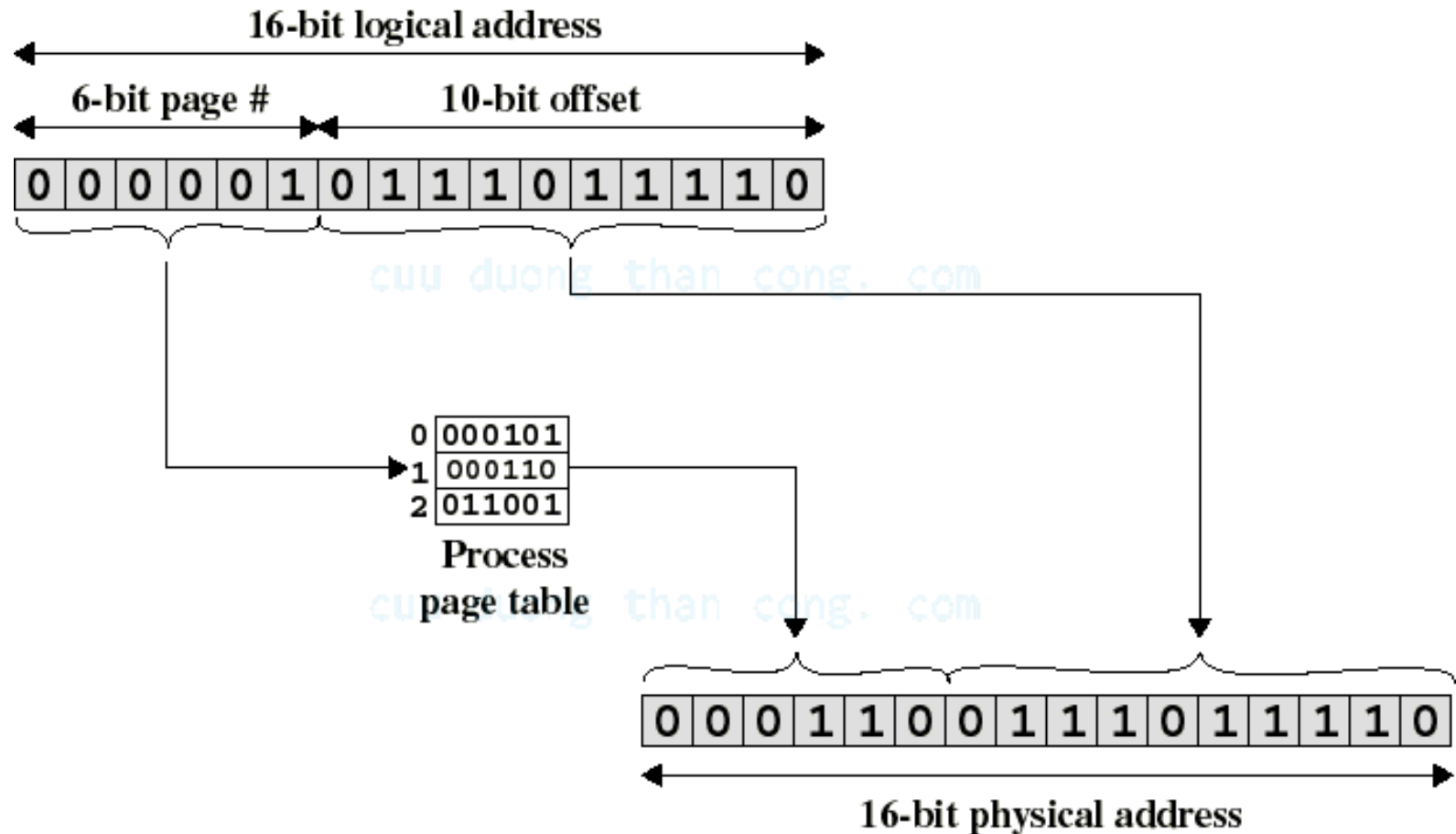
Tổ chức Paging

- Địa chỉ logic : $\langle \text{page-number}, \text{offset} \rangle$
- Địa chỉ physic : $\langle \text{frame-number}, \text{offset} \rangle$
- Chuyển đổi địa chỉ : $\langle p, d \rangle \Rightarrow \langle f, d \rangle$
- 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

Chuyển đổi địa chỉ trong mô hình Paging



Logical-to-Physical Address Translation in Paging



Nhận xét Mô hình Paging

- 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
 - ☹ Tăng thời gian chuyển đổi địa chỉ

Lưu trữ Page Table

- 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 $\text{pagesize} = 2^n$
 - Tại sao ?
- Số trang trong KGĐC: $\# \text{pages} = 2^m / 2^n = 2^{m-n}$
 - Ví dụ : 32-bits địa chỉ, $\text{pagesize} = 4K$
 - KGĐC = $2^{32} \rightarrow \# \text{pages} = 2^{32} / 2^{12} = 2^{20} = 1.000.000$ pages !
 - $\# \text{pages} = \# \text{entry trong PT}$
- Địa chỉ logic :

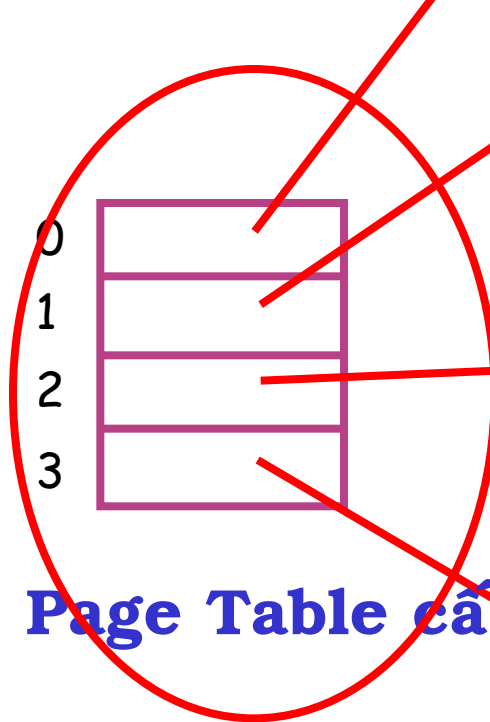
The diagram shows a blue rectangular box. Inside this box, there are two main sections labeled 'p' and 'd'. The 'p' section is on the left and is further divided into two smaller, unlabeled sections by a vertical line. The 'd' section is on the right.

(m-n)
- Page Table
n
 - ☹ Mỗi tiến trình lưu 1 Page Table
 - ☹ Số lượng phần tử quá lớn \rightarrow Lưu BNC
 - ☹ Mỗi truy xuất địa chỉ sẽ tốn 2 lần truy xuất BNC

Lưu trữ 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

Bảng trang đa cấp



Page Table cấp 1

0	
1	
2	
3	

Page Table cấp 2

4	
5	
6	
7	

Page Table cấp 2

8	
9	
10	
11	

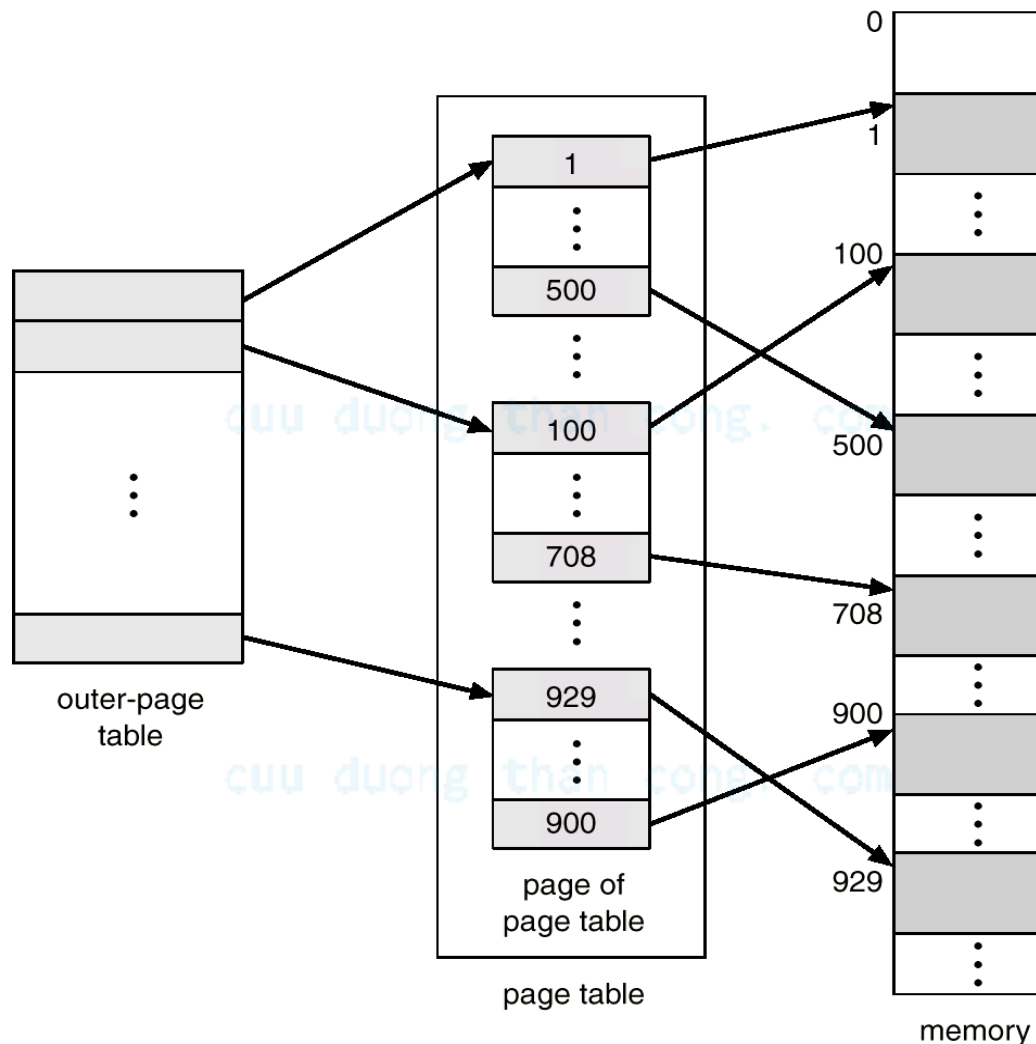
Page Table cấp 2

12	
13	
14	
15	

Page Table cấp 2

0	
1	
2	
3	
4	
5	
6	
7	
8	
9	
10	
11	
12	
13	
14	
15	

Mô hình bảng trang 2 cấp





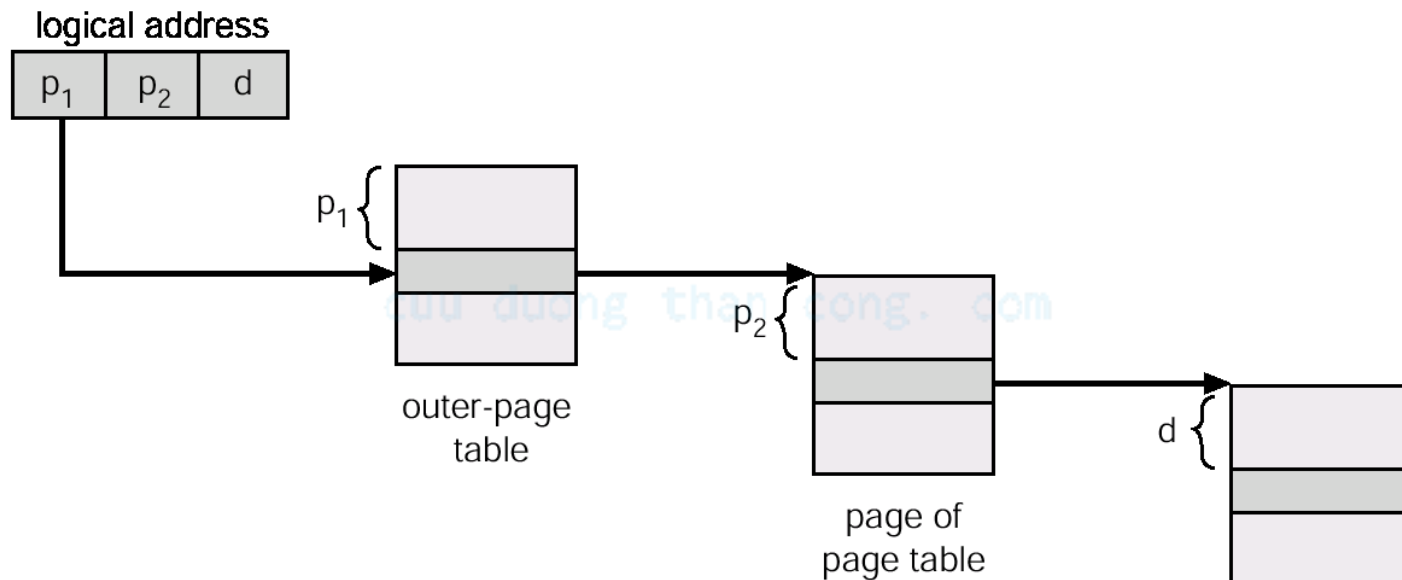
Ví dụ mô hình bảng trang 2 cấp

- 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 2 phần:
 - Số hiệu trang cấp 1.
 - Số hiệu trang cấp 2.

Ví dụ mô hình bảng trang 2 cấp

- Vì thế, địa chỉ logic sẽ có dạng như sau:

page number		page offset
p_i	p_2	d
10	10	12



Bảng trang đa cấp

1	2	100
---	---	-----

0	
1	
2	
3	

Page Table cấp 1

0	
1	
2	
3	

Page Table cấp 2

0	
1	
2	
3	

Page Table cấp 2

0	
1	
2	
3	

Page Table cấp 2

0	
1	
2	
3	

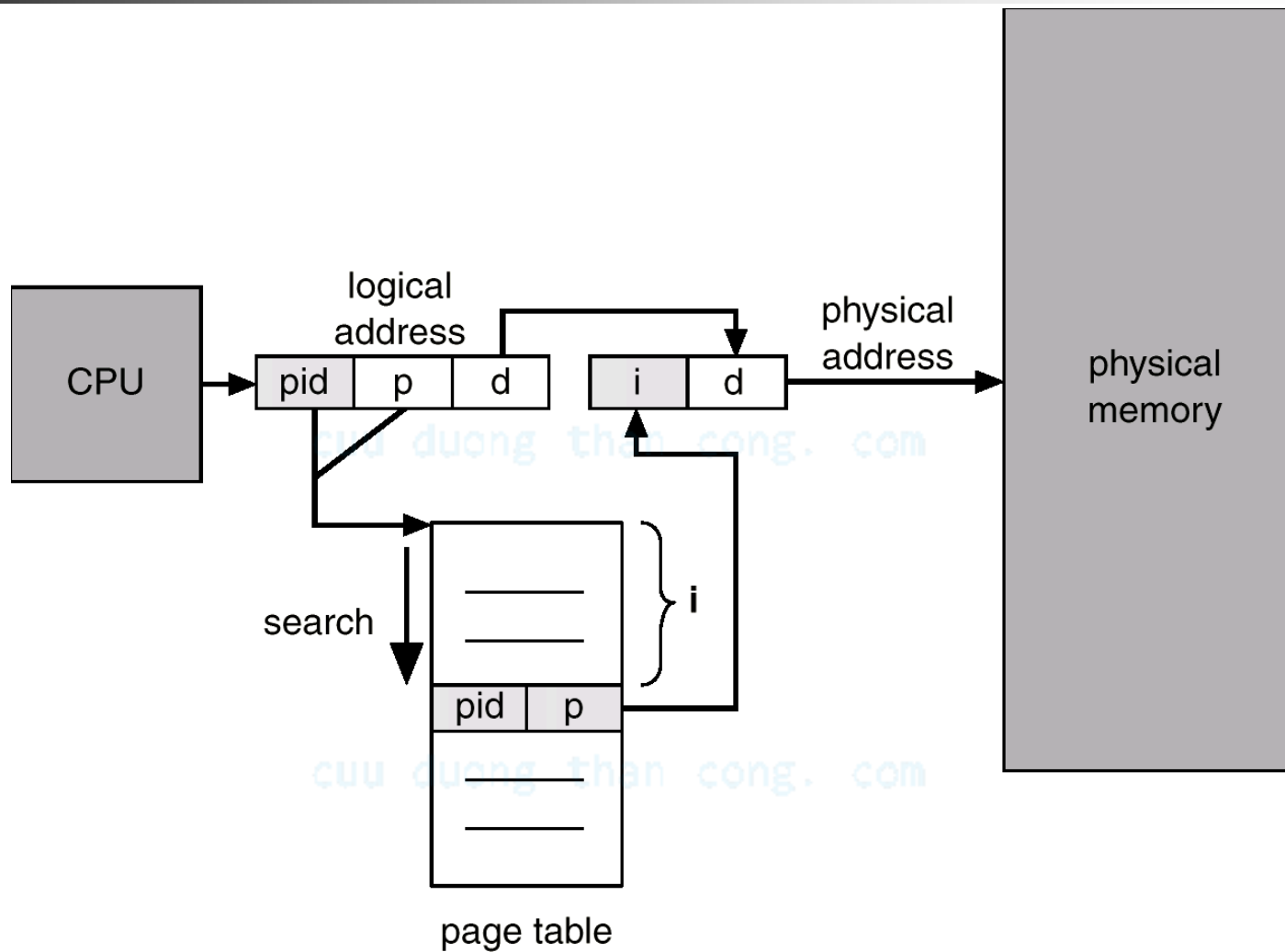
Page Table cấp 2

0	
1	
2	
3	
4	
5	
6	
7	
8	
9	
10	
11	
12	
13	
14	
15	
16	
17	

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 sở 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

Kiến trúc bảng trang nghịch đảo





Lưu trữ Page table : Tiết kiệm thời gian

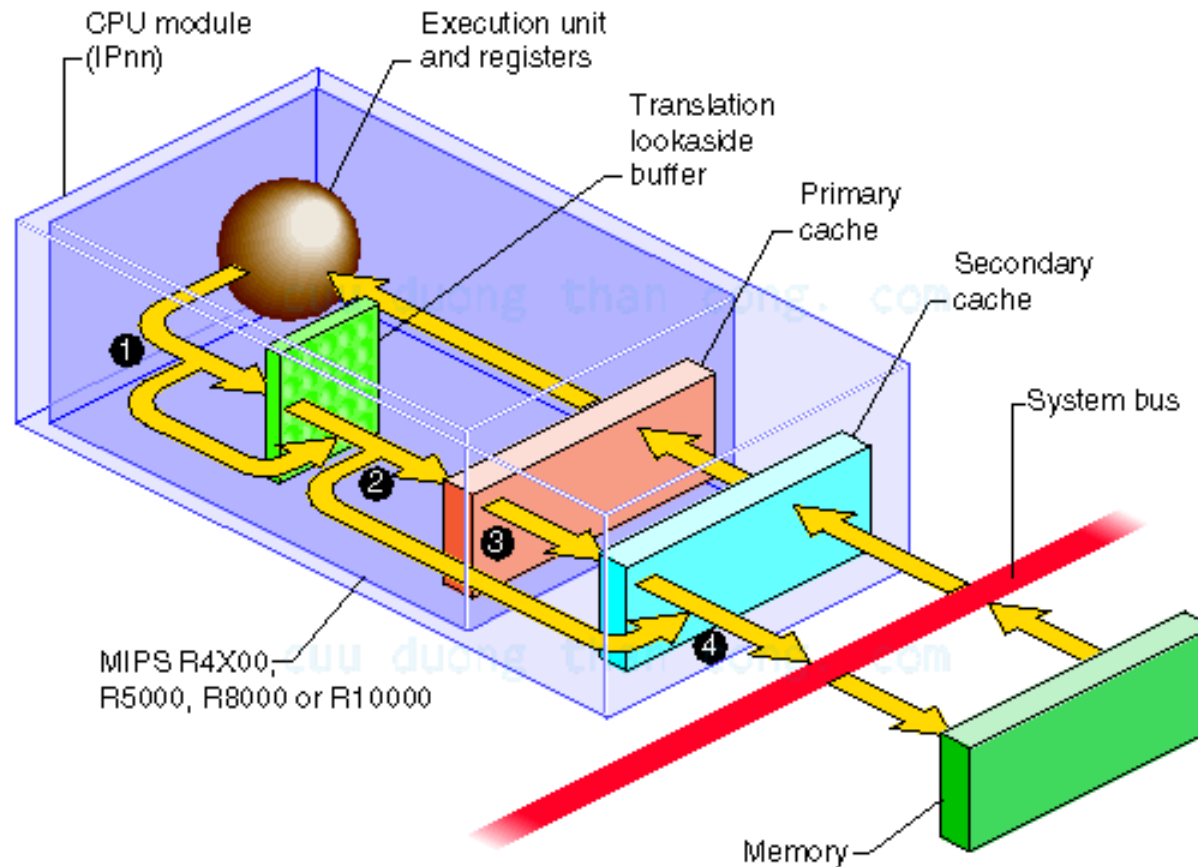
- Mỗi truy cập BNC cần truy xuất BNC 2 lần :
 - Tra cứu Page Table để chuyển đổi địa chỉ
 - Tra cứu bản thân data
- Làm gì để cải thiện :
 - Tìm cách lưu PT trong cache
 - Cho phép tìm kiếm nhanh
 - PT lớn, cache nhỏ : làm sao lưu đủ ?
 - Lưu 1 phần PT...
 - Phần nào ?
 - Các số hiệu trang mới truy cập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



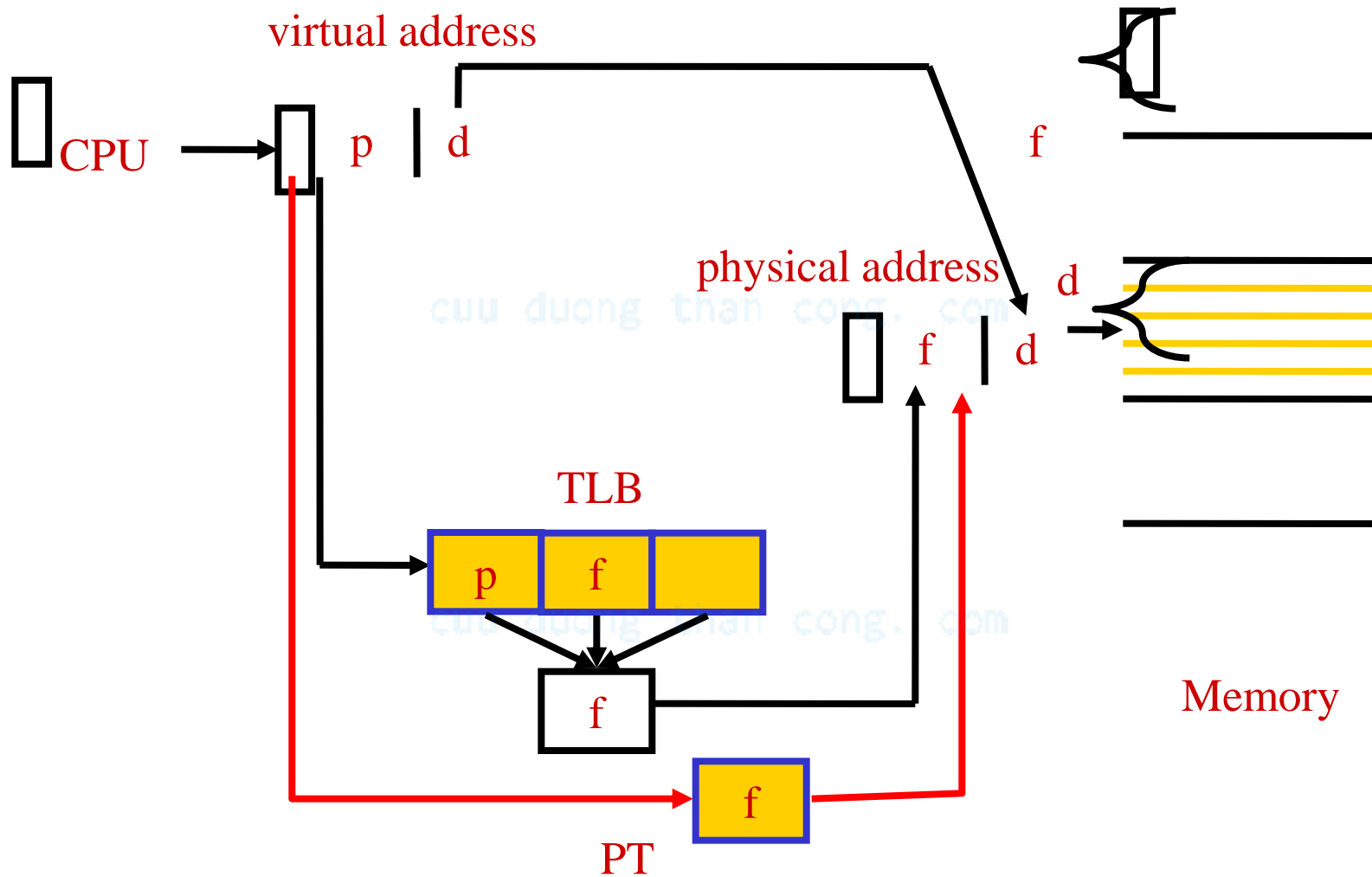
Translation Lookaside Buffer (TLB)

- 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.

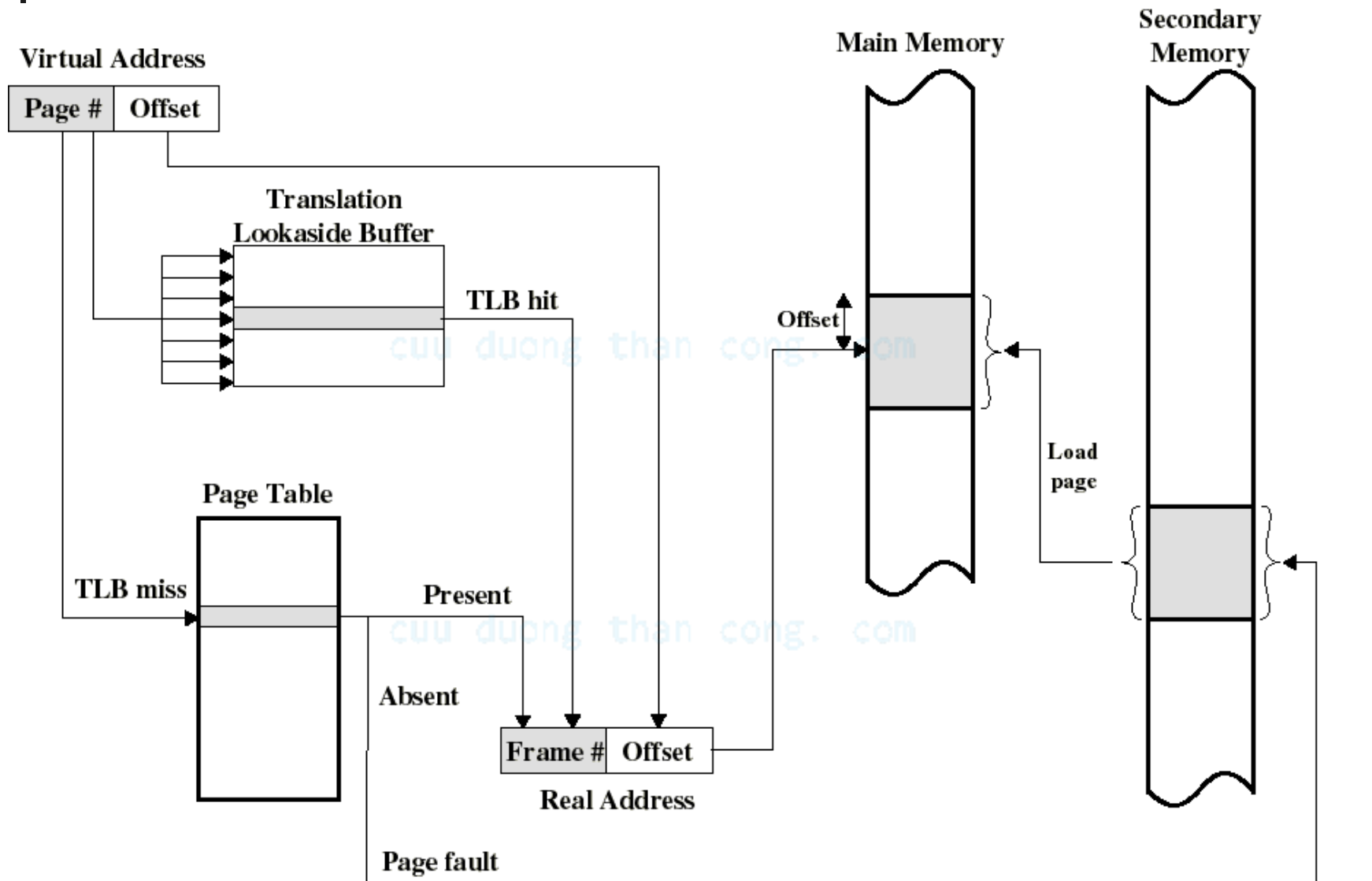
Translation Lookaside Buffer



Chuyển đổi địa chỉ với Paging



Sử dụng TBL



Bảo vệ và chia sẻ trong Segmentation và Paging

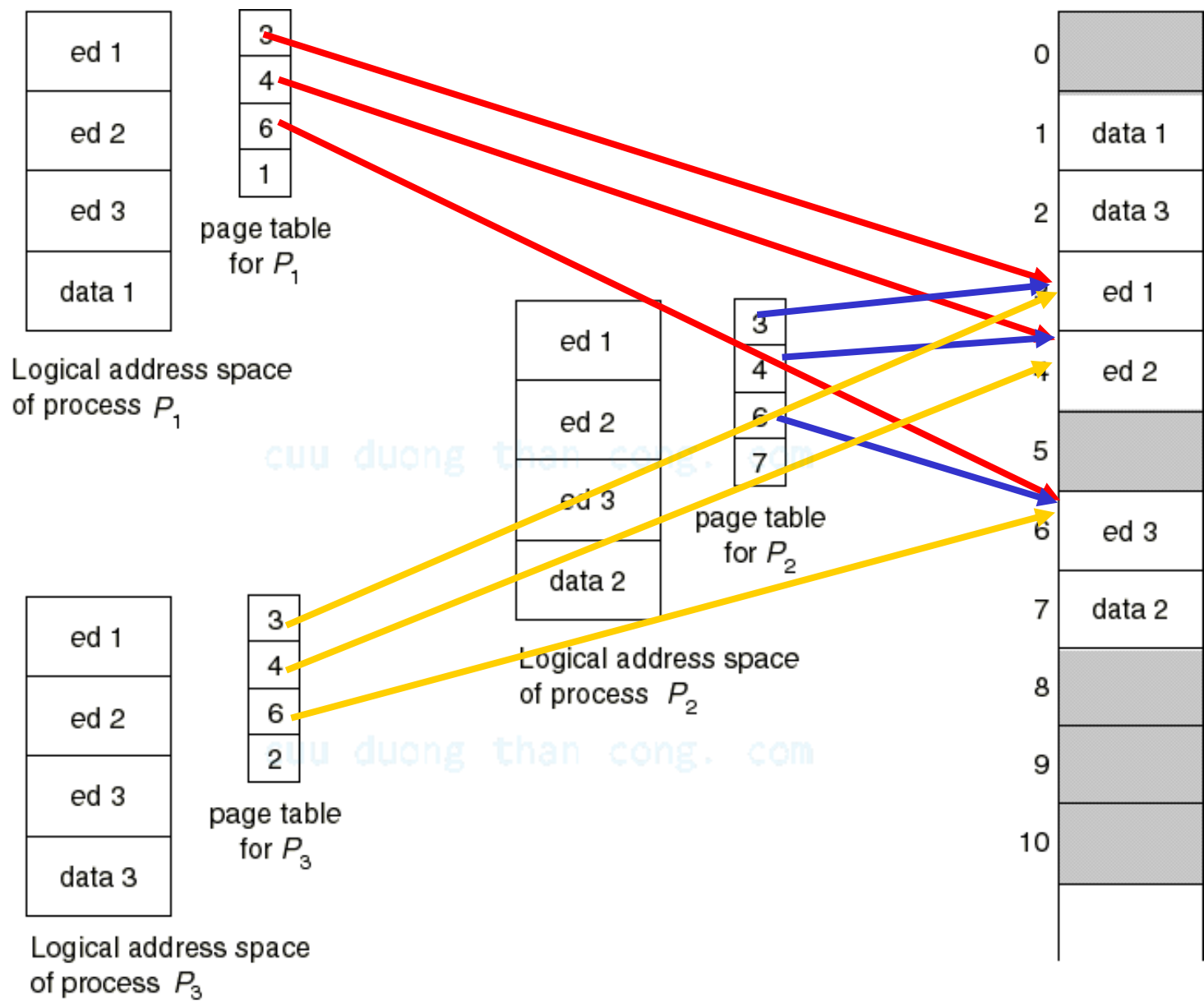
■ Bảo vệ

- 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.

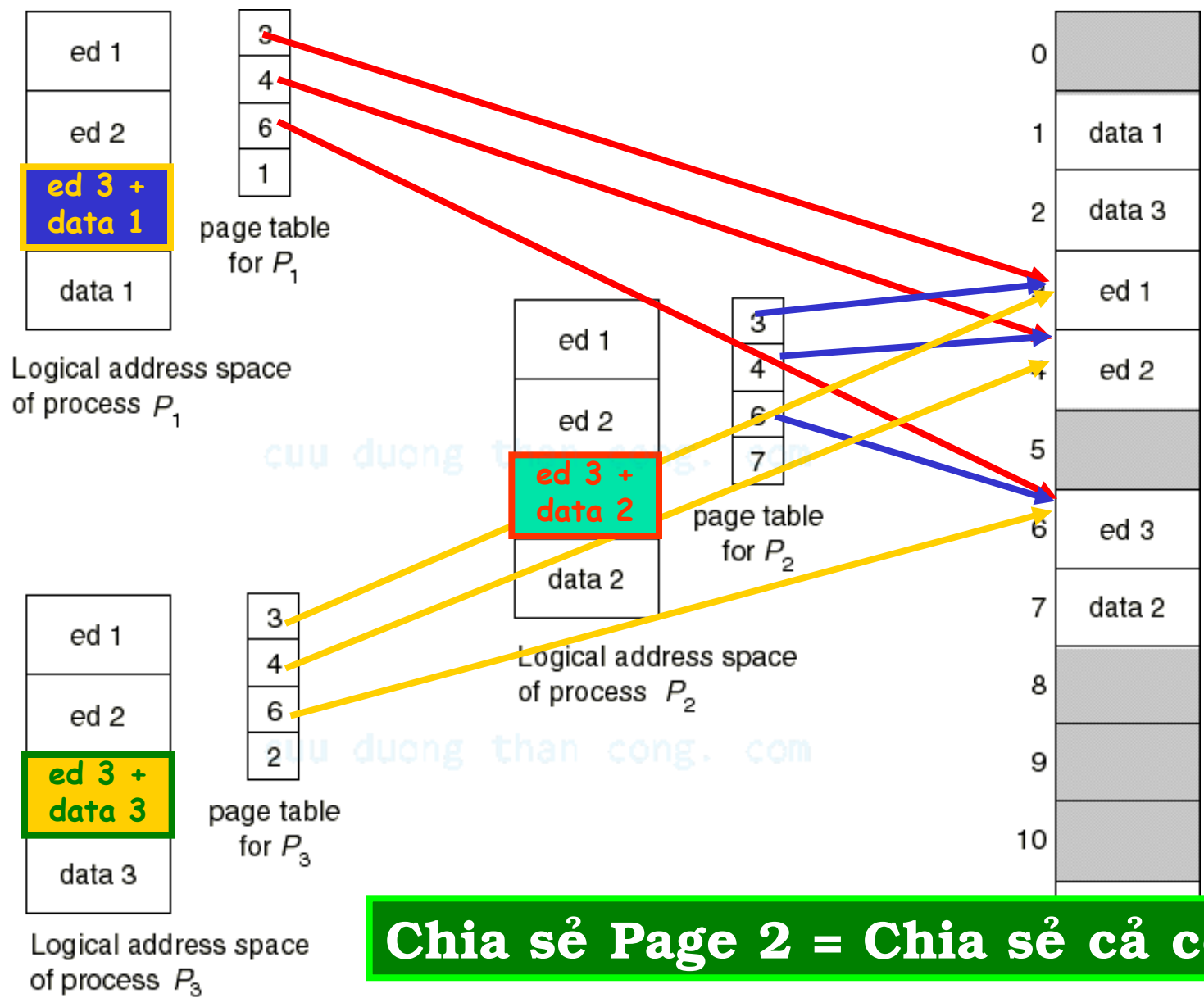
■ Chia sẻ: Cho nhiều phần tử trong KGĐC cùng trỏ đến 1 vị trí trong KGVL

- Segmentation : chia sẻ mức module chương trình
- Paging : chia sẻ các trang

Sharing Pages: A Text Editor



Sharing Pages: A Text Editor



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-rat**ion : 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 binding add + time acces memory)**

■ Linker-Loader

$$\text{TEA} = t_m \text{ (data)}$$

■ Base + Bound

$$\text{TEA} = (t_c + t_c) + t_m \text{ (Base \& Bound) (data)}$$

■ Segmentation

$$\text{TEA} = t_c + t_m \text{ (ST trong cache) (data)}$$

■ Paging

■ Không sử dụng TLB :

$$\text{TEA} = t_m + t_m \text{ (PT trong mem) (data)}$$

■ Có sử dụng TLB :

$$\text{TEA} = \text{hit-ratio} (t_c + t_m) + (1 - \text{hit-ratio})(t_c + t_m + t_m) \text{ (TLB) (data)}$$