Chương 1

TỔNG QUAN

NỘI DUNG CHƯƠNG 1

- Cấu trúc hệ thống máy tính
- Khái niệm về hệ điều hành (HĐH)
- Các thành phần cơ bản của HĐH
- Lịch sử phát triển của HĐH
- Chức năng của HĐH
- Bài tập

CẤU TRÚC HỆ THỐNG MÁY TÍNH

- Phần cứng (hardware)
  - CPU
  - Bộ nhớ :RAM, ROM,đĩa từ, băng từ…
  - Thiết bị I/O : Màn hình, bàn phím, card I/O…
- Phần mềm (software)
  - Phần mềm hệ thống
  - Phần mềm ứng dụng
  - Công cụ phần mềm
- Phần dẻo (firmware)

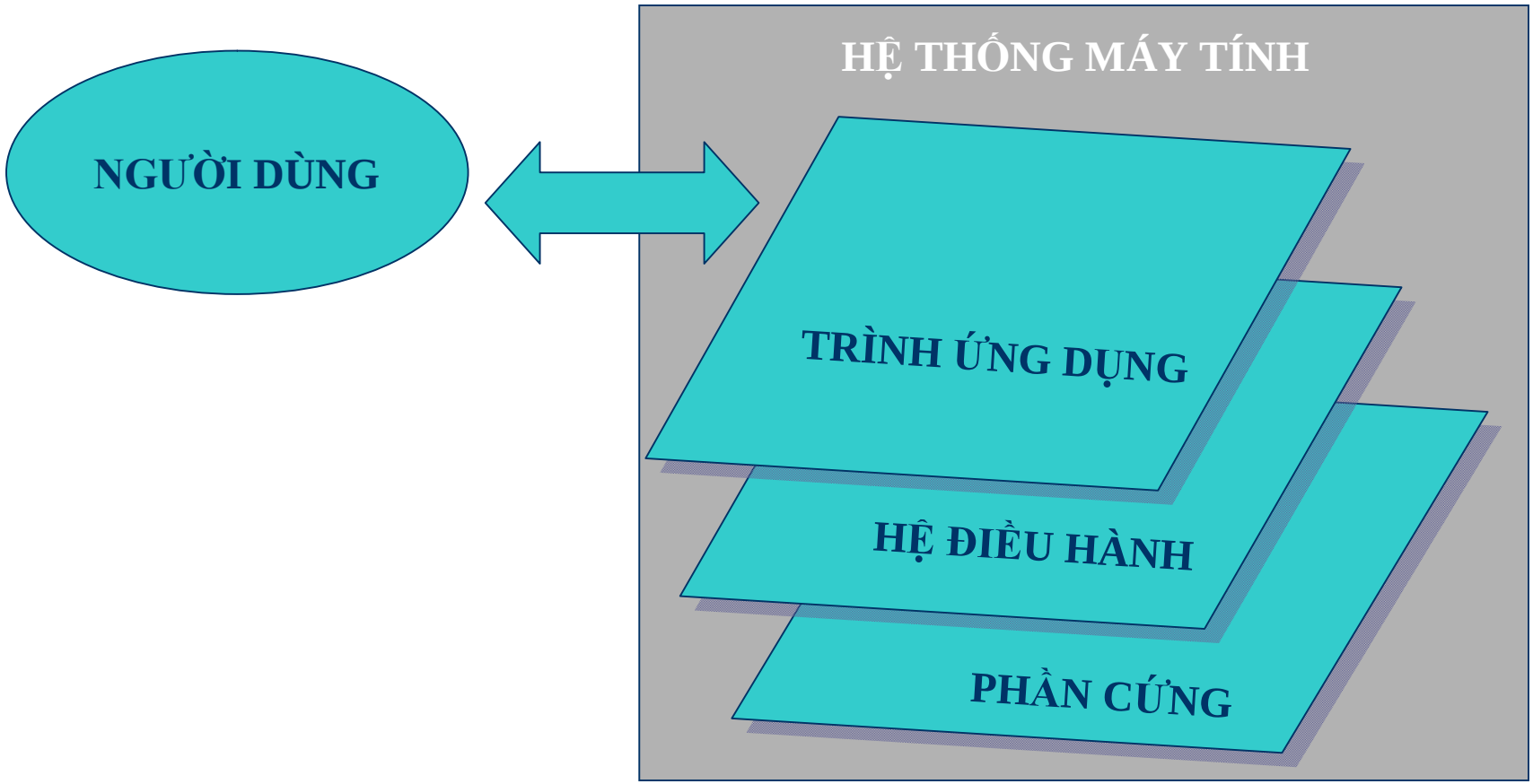
VÍ DỤ

- Phần mềm hệ thống
  - Hệ điều hành
  - Hệ quản trị CSDL: Oracle, SQL Server
  - Tiện ích: Norton Disk Doctor, SiSoft Sandra.
- Phần mềm ứng dụng:
  - MS Office, Corel Draw, Netscape Navigator
- Hệ điều hành:
  - MS-DOS, Windows 9x/ NT/ ME/ 2000/ XP…
  - Linux, Solaris, HP-UX, AIX, BSD, MacOS,…
  - Novell Netware

KHÁI NIỆM HỆ ĐIỀU HÀNH

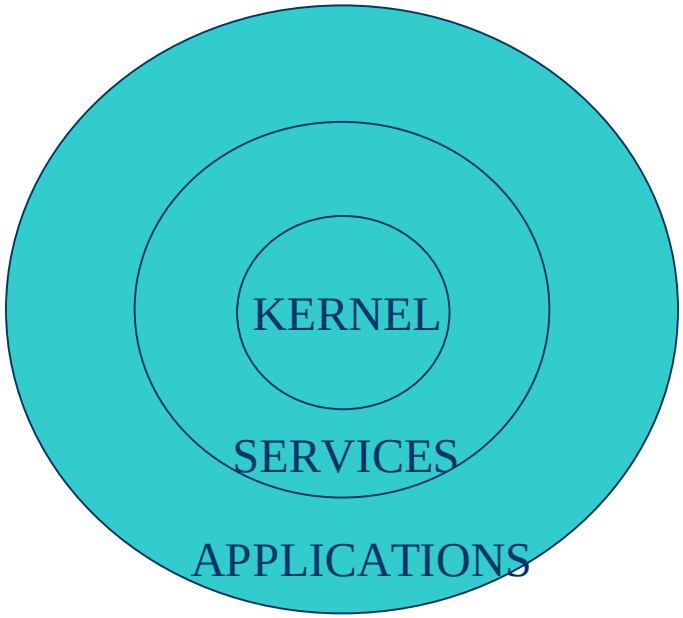
- Hệ điều hành – Operating System
- Phần mềm nằm giữa phần cứng máy tính và người dùng
  - Điều khiển phần cứng
  - Cung cấp các dịch vụ cho các chương trình ứng dụng
- Phần mềm quản lý và phân phối tài nguyên máy tính

HỆ ĐIỀU HÀNH



CÁC THÀNH PHẦN CỦA HĐH

- Phần lõi (kernel)
  - Quản lý quá trình
  - Quản lý bộ nhớ
  - Quản lý hệ thống file
  - Quản lý xuất nhập



- Phần giao diện:
  - Trình thông dịch lệnh (Command Interpreter, Shell)
  - Giao diện đồ họa (Graphic User Interface)

THIẾT KẾ CỦA HỆ ĐIỀU HÀNH

- **Monolithic OS**:
  - Cung cấp tập các chức năng cần thiết
  - Không phân biệt chức năng của hệ thống và chức năng của người dùng
  - Các ứng dụng ràng buộc với Hệ Điều Hành
  - Khó có khả năng mở rộng khi co thêm yêu cầu về dịch vụ
  - Thường dùng cho các hệ thống nhúng
- **Microkernel OS và thiết kế phân lớp:**
  - Kernel cung cấp các dịch vụ cơ bản nhất về quá trình, bộ nhớ và liên lạc giữa các quá trình
  - Các dịch vụ được đưa vào được xây dựng trên các dịch vụ cơ bản nhất.
  - Thường được dùng trong các hệ điều hành hiện đại và đa dụng

THIẾT KẾ CỦA HỆ ĐIỀU HÀNH

- **Virtual machine:**
  - Hệ thống được xem như có nhiều máy tính khác nhau.
  - Các ứng dụng có thể chạy đồng thời giống như chạy trên các cấu trúc phần cứng khác nhau
  - Cần nhiều tài nguyên hệ thống phục vụ cho các virtual machine khác nhau
  - Vi dụ: VMWare, Java Virtual Machine

LỊCH SỬ PHÁT TRIỂN HĐH

- Thao tác viên *(Operator)*
- Hệ thống xử lý bó *(batch programming system)*
- Hệ thống đa chương*(multiprogramming system)*
- Hệ thống đa nhiệm *(multitasking system)*:
- Hệ thống đa người dùng *(multiuser system)*
- Hệ thống đa xử lý *(multiprocessing system)*
- Hệ thống nhúng *(embedded systems)*
- Hệ thống thời gian thực *(real-time systems)*
- Hệ điều hành phân bố *(distributed OS)*

VÍ DỤ VỀ CÁC HĐH

- MS-DOS
- Windows 3.11/ 95/ 97/ 98/ 99/ ME
- Windows NT/ 2000/ XP…
- UNIX: Solaris, Linux, SCO, HP-UX, AIX, …
- BeOS, RTLinux, …
- Mach, Amoeba…

CHỨC NĂNG HỆ ĐIỀU HÀNH

- **Quản lý quá trình**

(process management)

- **Quản lý bộ nhớ**

(memory management)

- **Quản lý hệ thống lưu trữ**

(storage management)

- **Giao tiếp với người dùng**

(user interaction)

BÀI TẬP

1. Phân loại các chương trình sau :

Photoshop, Internet Explorer, Win 2000 Datacenter, Win2000 Ađvanced Server, Oracle, MySQL, MS Powerpoint, BeOS, MacOS, Solaris, Linux, MS-DOS, Norton Utilities.

2. Phân loại các hệ điều hành sau :

Windows NT, Win98SE, Windows ME,Windows XP, Linux, BeOS, Solaris, SCO-UNIX, MS-DOS.

3. Nhiệm vụ nào sau đây là của hệ điều hành :

- Kiểm tra quyền sử dụng hệ thống
- Kiểm tra quyền tạo, xóa một file
- Kiểm tra và tắt các chương trình virus, worm.
- Kiểm tra và tát các chương trình ảnh hưởng đến hoạt động của kernel
- Kiểm tra và qnản lý các thiết bị phần cứng

BÀI TẬP

4. Thiết lập trình tự cho các hoạt động sau

Chạy ứng dụng, nạp driver, POST, đọc MBR, nạp OS, chạy các dịch vu.

5. Chọn lựa hệ điều hành sử dụng đối với

- Người dùng cá nhân
- Công ty nhỏ
- Công ty vừa, lớn
- Ngân hàng

6. Linux có ưu điểm gì ? HĐH này có thể phổ biến với mọi người dùng như Windows hay không ?

Chương 2

QUÁ TRÌNH

CHƯƠNG 2 : QUÁ TRÌNH

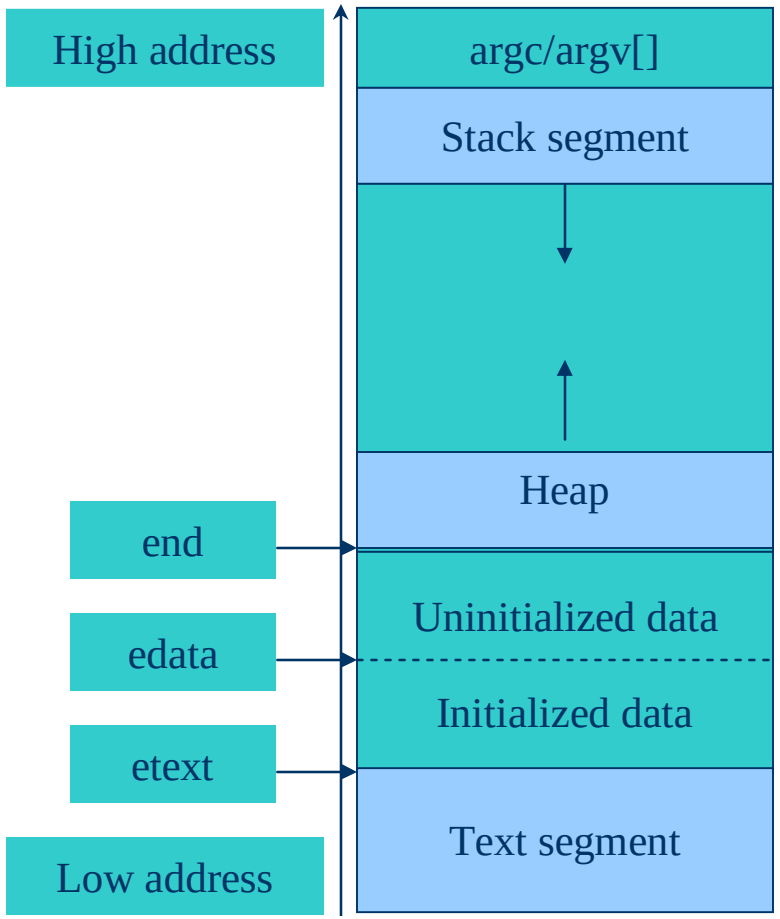
- Khái niệm về quá trình
- Trạng thái của quá trình
- Khối điều khiển quá trình
- Chuyển đổi trạng thái quá trình
- Các tác vụ thực hiện đ/v quá trình
- Ngắt quáng
- Chuyển ngữ cảnh
- Bài tập

QUÁ TRÌNH (*PROCESS*)

- Một chương trình đang thực thi
- Có thời gian sống (life cycle)
- Là một thực thể tích cực
- Có nhiều trạng thái và có thể chuyển trạng thái
- Một quá trình có các thông tin: bộ đếm chương trình, stack, vùng chứa dữ liệu và biến môi trường...
- Trong một số tài liệu, quá trình (proces) ≈ công việc (job) ≈ tác vụ (task)

CẤU TRÚC PROCESS CỦA UNIX

```
1.  int a = 0, b, *c;
2.  int main( int argc, char *argv[ ] ) {
3.      b= increase(a);
4.      c =(int*)malloc(10*sizeof(int));
5.      c[5]= b;
6.  }
7.  int  increase(int x) {
8.      return x ++;
9.  }
```



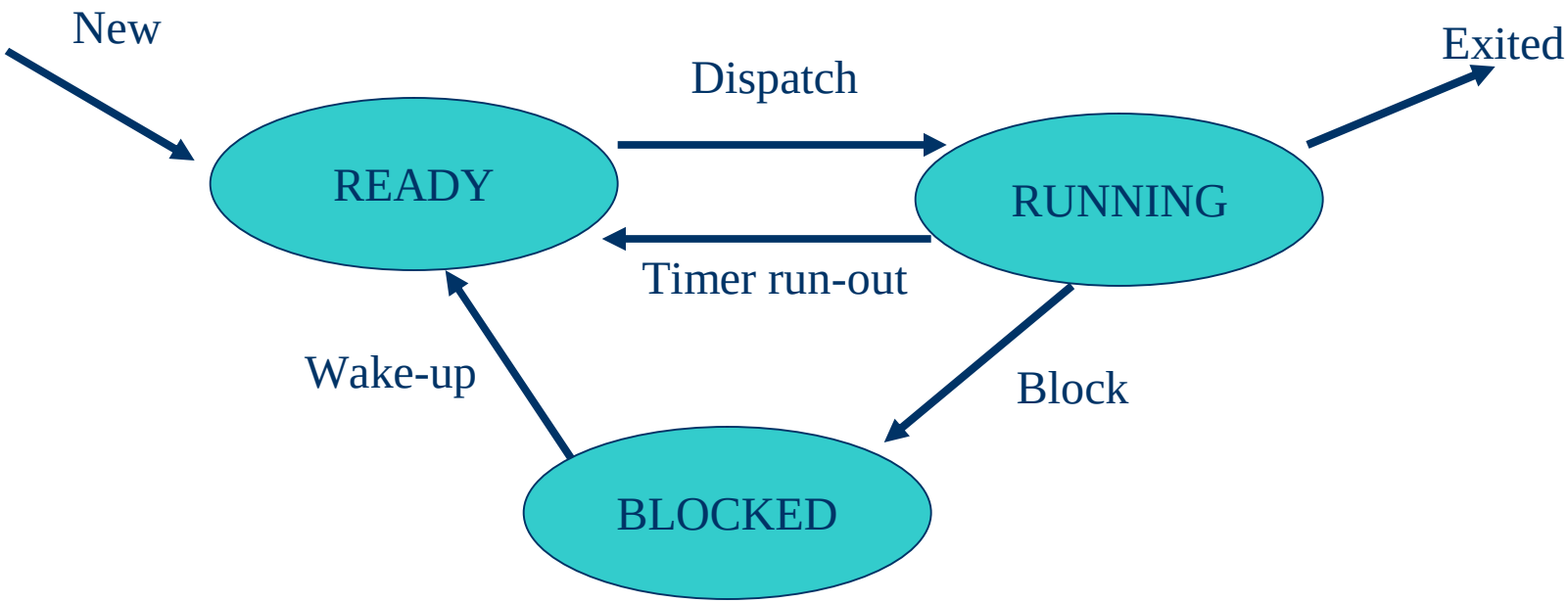
KHỐI ĐIỀU KHIỂN QUÁ TRÌNH

- **Cấu trúc dữ liệu của HĐH để quản lý quá trình**
- **Chứa thông tin nhận dạng, trạng thái, định vị tài nguyên cho quá trình**
  - Danh định cho quá trình (PID)
  - Bộ đếm chương trình
  - Vùng lưu giá trị thanh ghi CPU
  - Độ ưu tiên của quá trình
  - Thông tin định vị bộ nhớ quá trình
  - Thông tin bảo mật
  - Con trỏ đếùn các quá trình cha, con
  - …

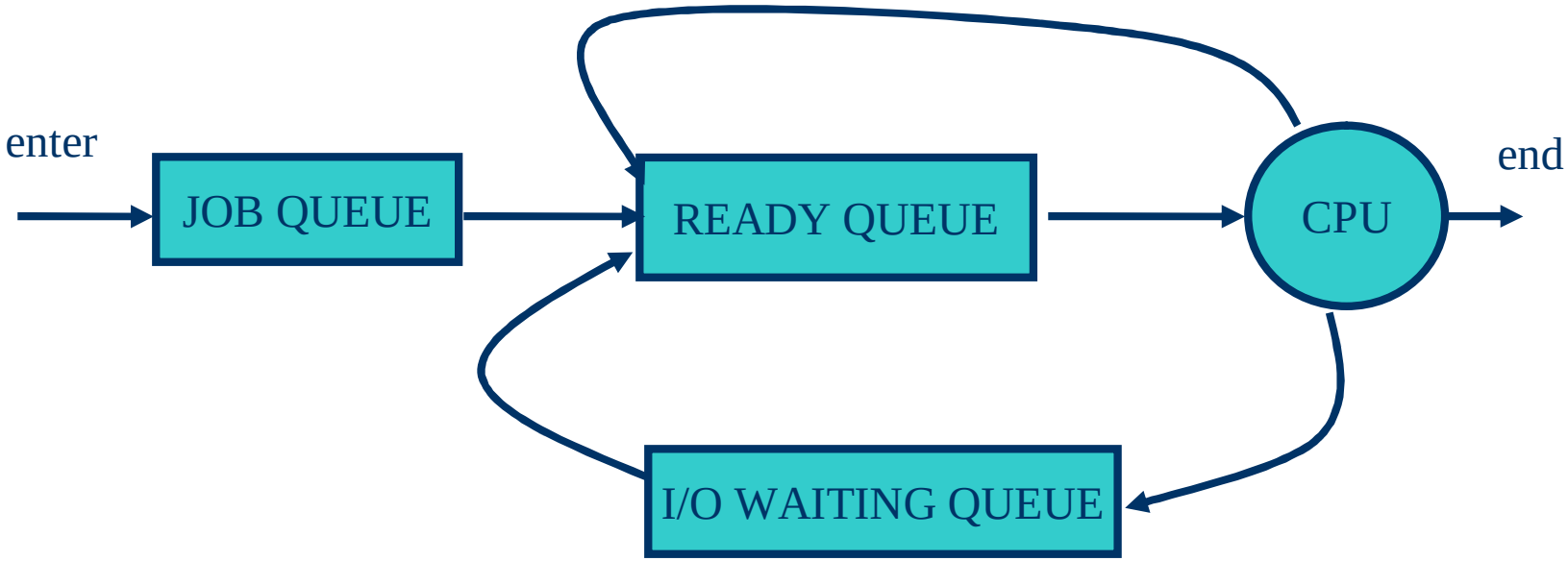
TRẠNG THÁI QUÁ TRÌNH

- **Sẵn sàng (*ready*)**
  - Quá trình đợi để gán cho CPU xử lý
- **Thực thi (*running*)**
  - Quá trình đang được CPU thực thi các lệnh
- **Bị chặn (*blocked*)**
  - Quá trình đợi một sự kiện nào đó
  - Sự kiện có thể là do việc xuất nhập dữ liệu hoặc từ một quá trình khác tạo ra

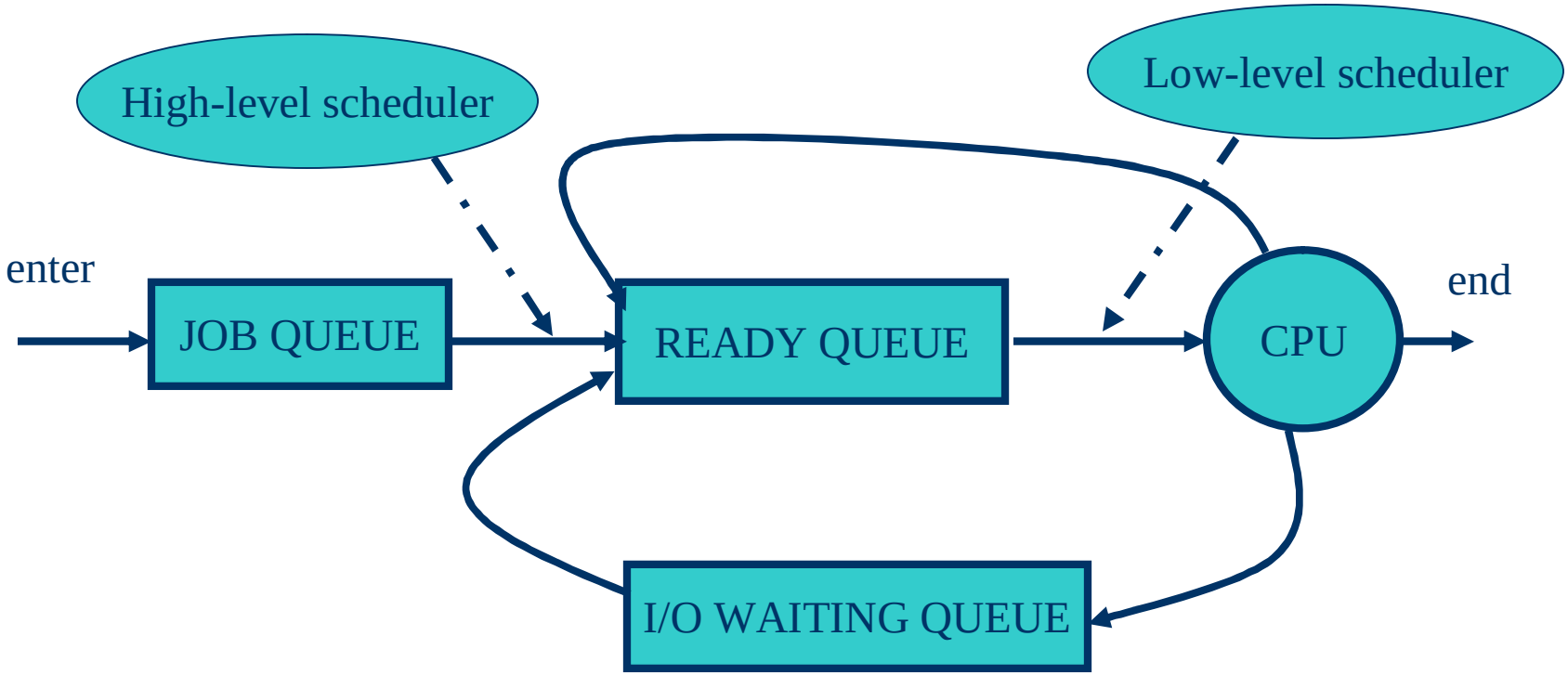
CHUYỂN TRẠNG THÁI QUÁ TRÌNH



CÁC HÀNG ĐỢI QUÁ TRÌNH



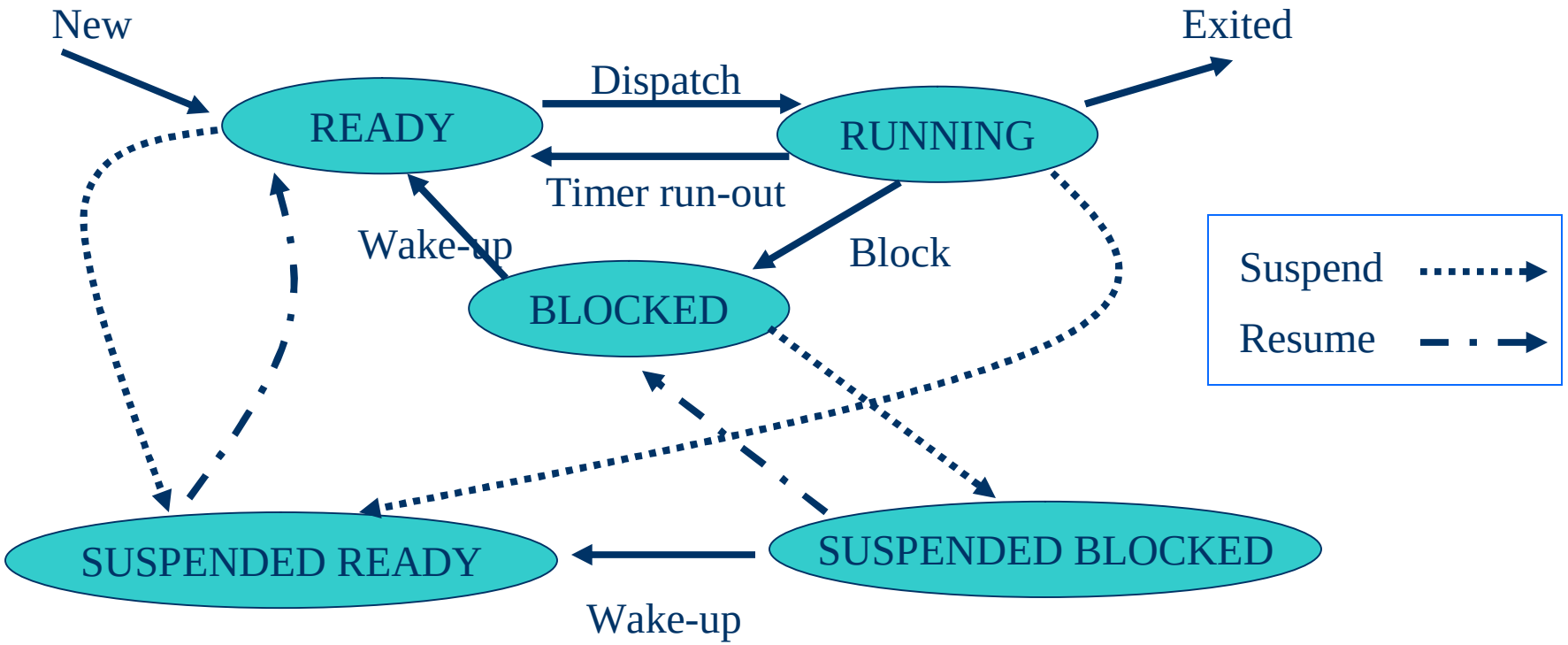
BỘ ĐỊNH THỜI (*SCHEDULER*)



TRẠNG THÁI TREO (*SUSPENDED*)

- **Quá trình bị treo trả lại mọi tài nguyên**
- **Tác nhân treo quá trình**
  - Bản thân quá trình
  - Hệ thống/quá trình khác
- **Xảy ra khi**
  - Cần giám sát quá trình
  - Hệ thống có sự cố
  - Hệ thống quá tải
- **Phục hồi (*resume*)**
  - Nhờ HĐH hoặc quá trình khác

LƯỢC ĐỒ CHUYỂN TRẠNG THÁI

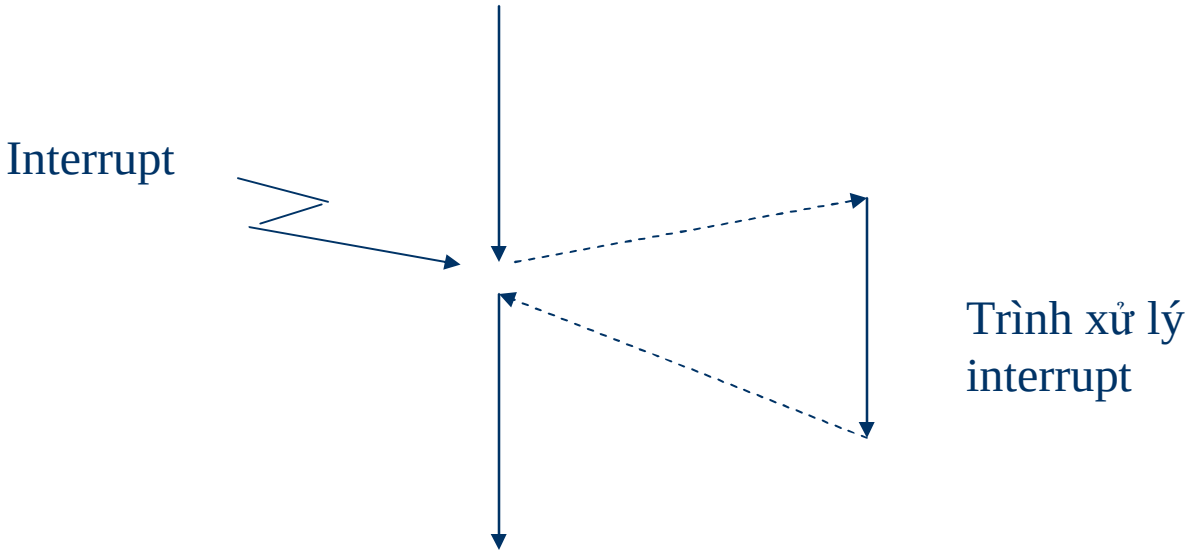


TÁC VỤ TRÊN QUÁ TRÌNH

- Tạo quá trình
  - Hàm hệ thống fork(), shell, GUI …
- Hủy quá trình
  - Hàm exit(), lệnh kill,…
- Thay đổi trạng thái quá trình
  - Suspend / Resume / Block / Wake-up/ Dispatch
  - Thực hiện bằng cách gởi tín hiệu đến quá trình
- Thay đổi độ ưu tiên của quá trình
- Cho phép các quá trình giao tiếp

NGẮT QUÃNG (*INTERRUPT*)

- Sự kiên làm thay đổi tiến trình thực thi của CPU
- Cơ chế:



CÁC BƯỚC XỬ LÝ INTERRUPT

- Hệ điều hành chiếm lại quyền thực thi
- Lưu trữ trạng thái của quá trình bị ngắt
- Phân tích ngắt quãng (dùng phần cứng)
- Thực thi chương trình xử lý ngắt quãng
- Chọn một quá trình tiếp theo để chạy
- Có thể cho phép interrupt xử lý lồng nhau

CÁC LOẠI INTERRUPT

- **Supervisor call (SVC) interrupt**: khi quá trình yêu cầu dịch vụ của hệ thống
- **I/O interrupt**: do các thiết bị I/O sinh ra khi chuyển trạng thái
- **External interrupt**: sinh ra từ đồng hồ hệ thống, bàn phím
- **Restart interrupt**: ngắt khởi động lại máy
- **Program check**: sinh ra khi chương trình thực thi lệnh bị lỗi (chia cho 0, tràn số, …)
- **Machine check:** do nhà sản xuất tạo ra dành riêng cho việc kiểm tra phần cứng

CHUYỂN NGỮ CẢNH (*CONTEXT SWITCHING*)

- Chuyển xử lý từ quá trình này sang quá trình khác.
- Xảy ra khi có interrupt
- Qui trình:
  - Lưu trạng thái của quá trình hiện hành
  - Chọn quá trình mới để xử lý
  - Đưa trạng thái của quá trình mới vào phần cứng để xử lý
- Một context bao gồm:
  - tập thanh ghi chứa thông tin trạng thái quá trình về trạng thái CPU, bộ nhớ … của quá trình.

  ➔ Program Status Word (PSW).

CHUYỂN NGỮ CẢNH (*CONTEXT SWITCHING*)

- **Các loại PSW:**
  - Old_PSW
  - Current _PSW
  - New_PSW
- **Hệ thống một bộ xử lý có 1 Current_PSW và 6 New_PSW và 6 Old_PSW (6 loại interrupt)**
- **Quá trình chuyển ngữ cảnh diễn ra qua sự thay đổi giữa các PSW trong hệ thống**
  - Current_PSW → Old_PSW
  - New_PSW → Current_PSW

BÀI TẬP

1. Những hệ điều hành nào sau đây cho phép nhiều quá trình cùng nằm trong bộ nhớ

Windows NT, Win98SE, Linux, MS-DOS.

2.Phân loại các Interrupt sau :

- Nhấn Ctrl+C giết quá trình
- Card mạng báo có dữ liệu tới
- Ổ đĩa CD báo quá trình chuyển dữ liệu đã xong
- Gọi hàm hệ thống fork()
- Nhấn nút Reset của máy tính

3.Tìm hiểu các hàm/ lệnh tạo, huỷ quá trình trên Linux.

4.Viết chương trình in ra địa chỉ các vùng nhớ của một quá trình trên Linux

Chương 3

ĐỊNH THỜI BỘ XỬ LÝ

CHƯƠNG 3 : ĐỊNH THỜI BỘ XỬ LÝ

- Bài toán định thời
- Các thuật ngữ
- Mục tiêu định thời
- Tiêu chí để định thời
- Tiêu chuẩn đánh gia
- Các giải thuật định thời
  - Định thời hạn chót
  - FIFO
  - SJF, SRT
  - RR
  - HRRN
  - Hàng đa mức hồi tiếp
- Bài tập

BÀI TOÁN ĐỊNH THỜI

- Định nghĩa :
  - Phân chia thời gian thực thi cho các quá trình đồng thời trong hệ thống sao cho các quá trình kết thúc và kết thúc nhanh nhất.
- Cấp độ định thời
  - Cấp cao (high-level)
  - Cấp trung (intermediate-level)
  - Cấp thấp (low-level)

CÁC THUẬT NGỮ

- CPU burst
- I/O burst
- Time slice / Quantum
- Interval Timer
- Các kiểu định thời
  - non-preemptive
  - preemptive

MỤC TIÊU ĐỊNH THỜI

1. Công bằng
2. Tăng hiệu suất tối đa
3. Cực đại số người dùng tương tác
4. Có thể dự đoán trước
5. Phí tổn ít
6. Cân đối việc sử dụng tài nguyên
7. Tránh trì hoãn vô hạn định (dùng độ ưu tiên)
8. Ưu tiên quá trình giữ tài nguyên quan trọng
9. Phục vụ tốt các quá trình có hướng thuận lợi
10. Điều phối tối ưu khi tải không cân đối

TIÊU CHÍ ĐỂ ĐỊNH THỜI

1. Mức độ dùng I/O (*I/O boundness*)
2. Mức độ dùng CPU (*CPU boundness*)
3. Đặc tính quá trình : batch, interactive,real-time…
4. Độ khẩn cấp của quá trình
5. Độ ưu tiên của quá trình
6. Tần suất gây lỗi tham khảo trang (*page fault*)
7. Tần suất bị giành CPU
8. Thời gian được CPU phục vụ từ khi tạo ra
9. Thời gian chạy còn lại của quá trình

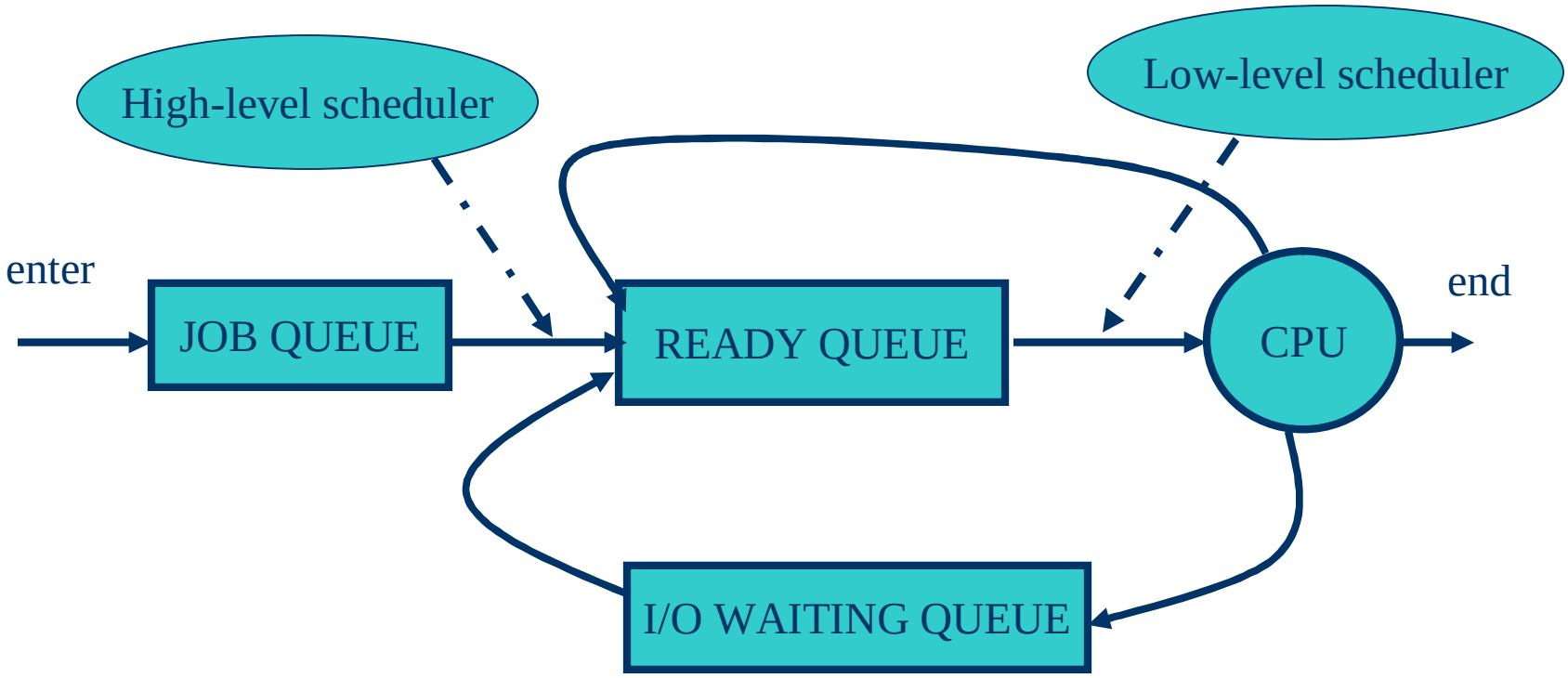
TIÊU CHUẨN ĐÁNH GIÁ GIẢI THUẬT ĐỊNH THỜI

1. Độ lợi CPU (*CPU utilization*)
2. Thông lượng (*throughput*)
3. Thời gian xử lý (*turnaround time*)
4. Thời gian đợi (*waiting time*)
5. Thời gian đáp ứng (*response time*)

BỘ ĐỊNH THỜI VÀ BỘ ĐIỀU VẬN

- **Bộ định thời quá trình** (*scheduler*)
  - Chọn lựa quá trình cho CPU phục vụ
  - Hoạt động vào những thời điểm
    1. Khi quá trình running → ready
    2. Khi quá trình từ running → blocked
    3. Khi quá trình từ blocked → ready
    4. Khi có quá trình kết thúc
- **Bộ điều vận** (*dispatcher*)
  - Chuyển điều khiển CPU sang cho quá trình.
  - Thực hiện bước chuyển ngữ cảnh:
    - Chuyển ngữ cảnh sang cấp người dùng
    - Nhảy sang vị trí thích hợp của quá trình và thực thi

BỘ ĐỊNH THỜI QUÁ TRÌNH



MỘT SỐ GIẢI THUẬT ĐỊNH THỜI

1. **Định thời hạn chót** (*Deadline Scheduling*)
2. **FIFO** (*First In First Out*)
3. **SJF** (*Shortest Job First*)
4. **SRT** (*Shortest Remaining Time*)
5. **RR** (*Round Robin*)
6. **HRRN** (*Highest Response Ratio Next*)
7. **Hàng đa mức hồi tiếp**

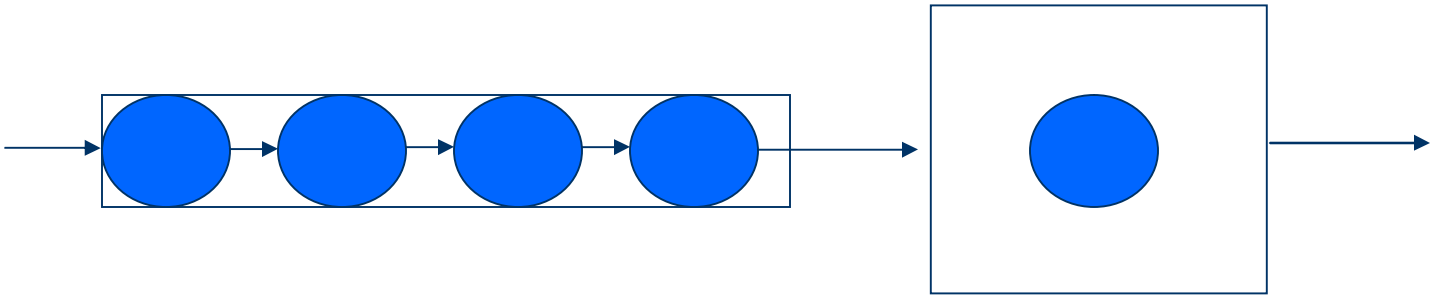
   (*Multilevel Feedback Queue*)

ĐỊNH THỜI HẠN CHÓT (Deadline Scheduling)

- Còn gọi là real-time scheduling
  - Hard real-time
  - Soft real-time
- Định thời sao cho các quá trình được thực thi theo một bảng thời gian xác định trước
- Mục đích : hoàn thành tác vụ kịp lúc
- Ứng dụng : công nghiệp, viễn thông, quân sự…
- Rất phức tạp
- Chỉ có giải thuật cho từng hệ thống cụ thể

FIFO (*First In First Out*)

- Còn gọi là **FCFS** (*First Come First Served*)
- Xét định thời quá trình theo thời gian đến hàng đợi ready của quá trình
- Quá trình vào trước sẽ được phục vụ trước
- Định thời theo kiểu non-preemptive



Processor

VÍ DỤ 1 : GIẢI THUẬT FIFO

- Thứ tự đến
  P1, P2, P3
- Thứ tự thực hiện
  P1→ P2 → P3

Quá trình	Thời gian thực thi (giây)
P1	24
P2	5
P3	2

VÍ DỤ 1 : GIẢI THUẬT FIFO

- **Thời gian xử lý (turnaround time)**

  P1: 24s P2: 29s P3: 31s

- **Thời gian xử lý trung bình**

  (24+29+31)/3 = 28s

- **Thời gian đợi (waiting time)**

  P1: 0s P2: 24s P3: 29s

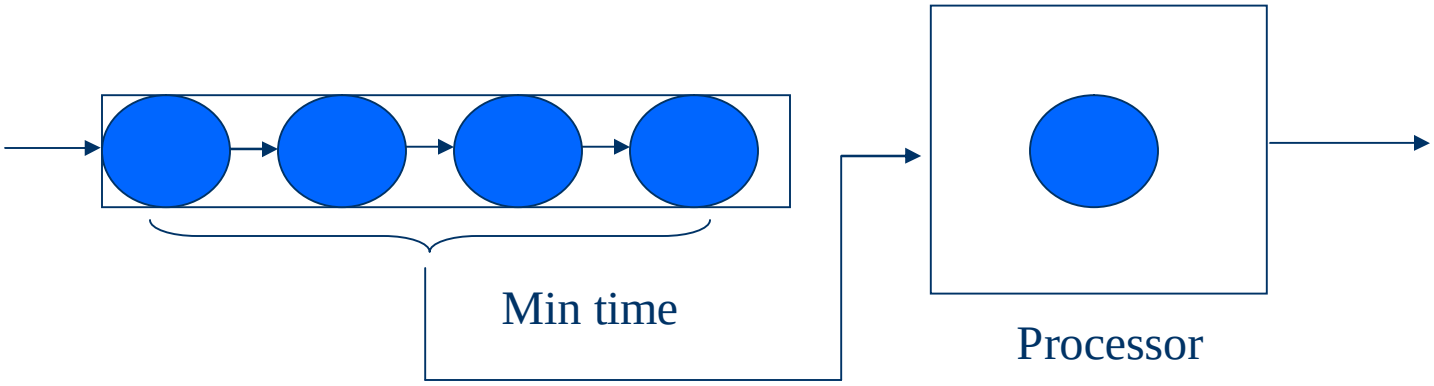
- **Thời gian đợi trung bình**

  (0+24+29)/3=17.67s

- **Nếu thứ tự đến các quá trình là P3 → P2 → P1 thì sao ?**
- **Nhận xét**

SJF (*Shortest Job First*)

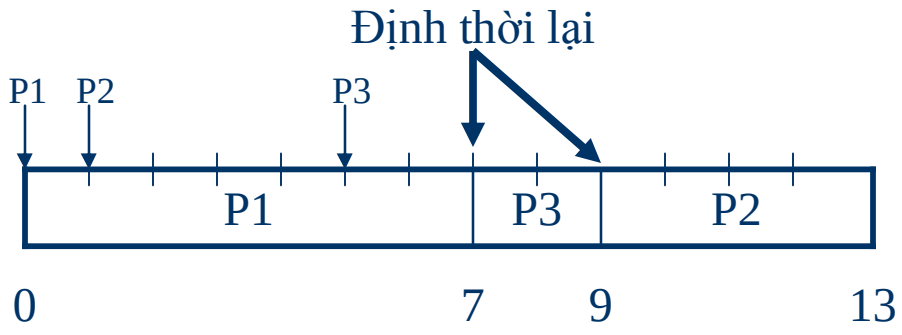
- Định thời theo kiểu **non-premptive**
- Quá trình có thời gian xử lý nhỏ nhất sẽ được xử lý trước
- Việc định thời được thực hiện sau khi có quá trình kết thúc



VÍ DỤ 2 : GIẢI THUẬT SJF

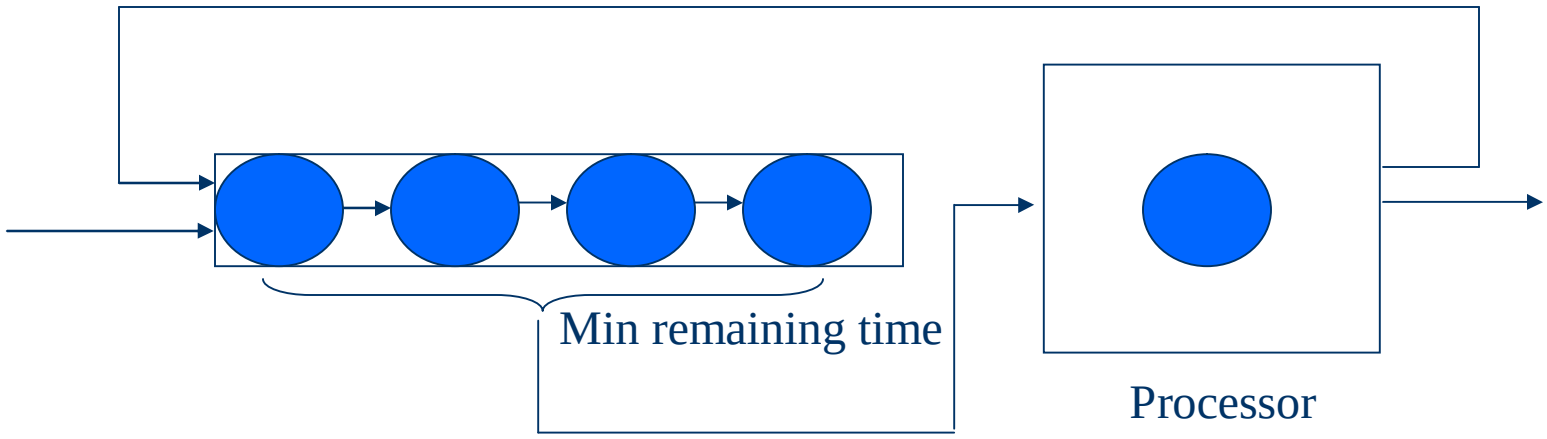
- Định thời
  P1→P3→P2
- Tính các thông số ?
- So sánh với
  định thời theo FIFO ?
- Nhược điểm ?

Quá trình	Thời gian đến	Thời gian thực thi (giây)
P1	0	7
P2	1	4
P3	5	2



SRT (*Shortest Remaining Time*)

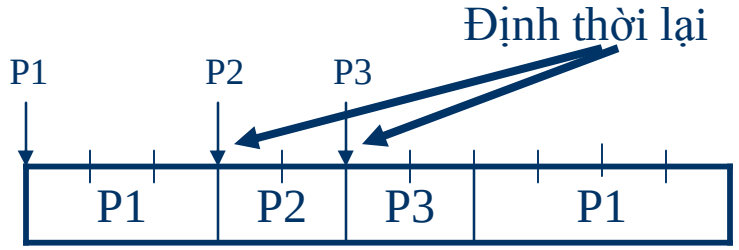
- Định thời theo kiểu **pre-emptive**
- Quá trình có thời gian xử lý còn lại nhỏ nhất sẽ được xử lý trước
- Việc định thời được thực hiện ngay cả khi có quá trình đến hệ thống



VÍ DỤ 3 : GIẢI THUẬT SRT

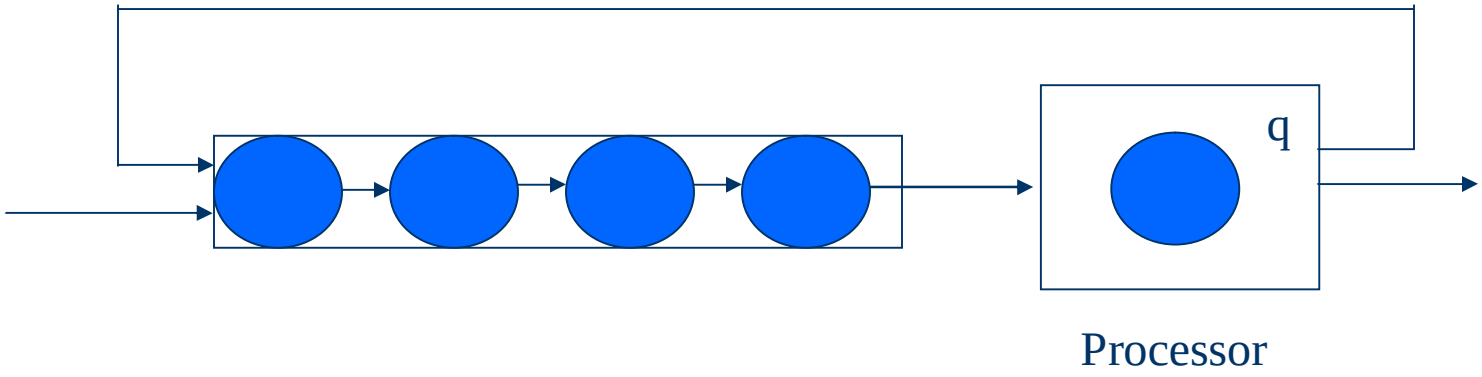
- Định thời : P1→P2→P3→P1
- Tính các thông số ?
- So sánh với SJF ?
- Nhược điểm ?

Quá trình	Thời gian đến	Thời gian thực thi (giây)
P1	0	7
P2	3	2
P3	5	2



RR(*Round Robin*)

- Định thời theo kiểu **pre-emptive**
- Quá trình chỉ được chiếm CPU trong khoảng thời gian q (quantum time). Nếu trong khoảng thời gian đó quá trình chưa kết thúc thì nó trả CPU lại cho Hệ điều hành và quay về cuối hàng đợi Ready.

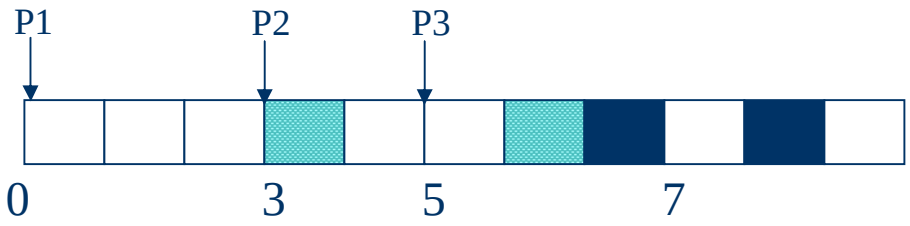


VÍ DỤ 4 : GIẢI THUẬT RR

- Tính các thông số ?
  Cho t_switch = 0

- Nhận xét

Quá trình	Thời gian đến	Thời gian thực thi (giây)
P1	0	7
P2	3	2
P3	5	2



Định thời Round robin với Quantum time là 1 giây

HRRN (*Highest Response Ration Next*)

- Cải tiến giải thuật SJF
- Định thời theo kiểu **non-preemptive**
- **Độ ưu tiên** của quá trình được tính theo công thức:

$$p = (t_w + t_s)/t_s$$

t_w ***waiting time***

t_s ***service time***

- Quá trình có độ ưu tiên lớn nhất được phục vụ
- Độ ưu tiên động, tính lại khi có quâ trình kết thúc

VÍ DỤ 5 : GIẢI THUẬT HRRN

- Khi P1 kết thúc, hệ thống định thời lại.

- Độ ưu tiên

  P2: (6+4)/4=2.5

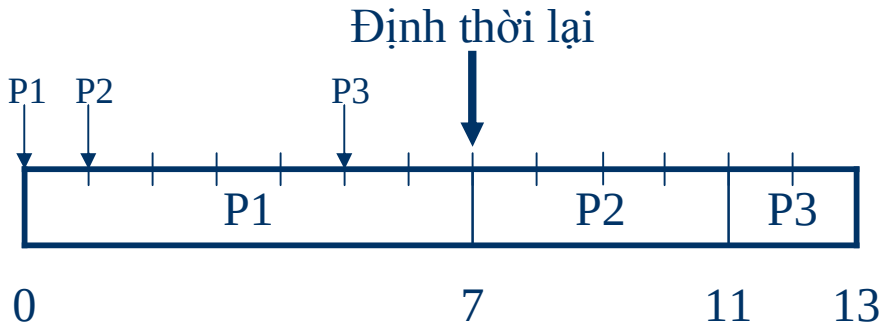
  P3: (2+2)/2=2

  P2 được ưu tiên

- Thứ tự định thời: P1→P2→P3

- Nhận xét

Quá trình	Thời gian đến	Thời gian thực thi (CPU burst time) (giây)
P1	0	7
P2	1	4
P3	5	2

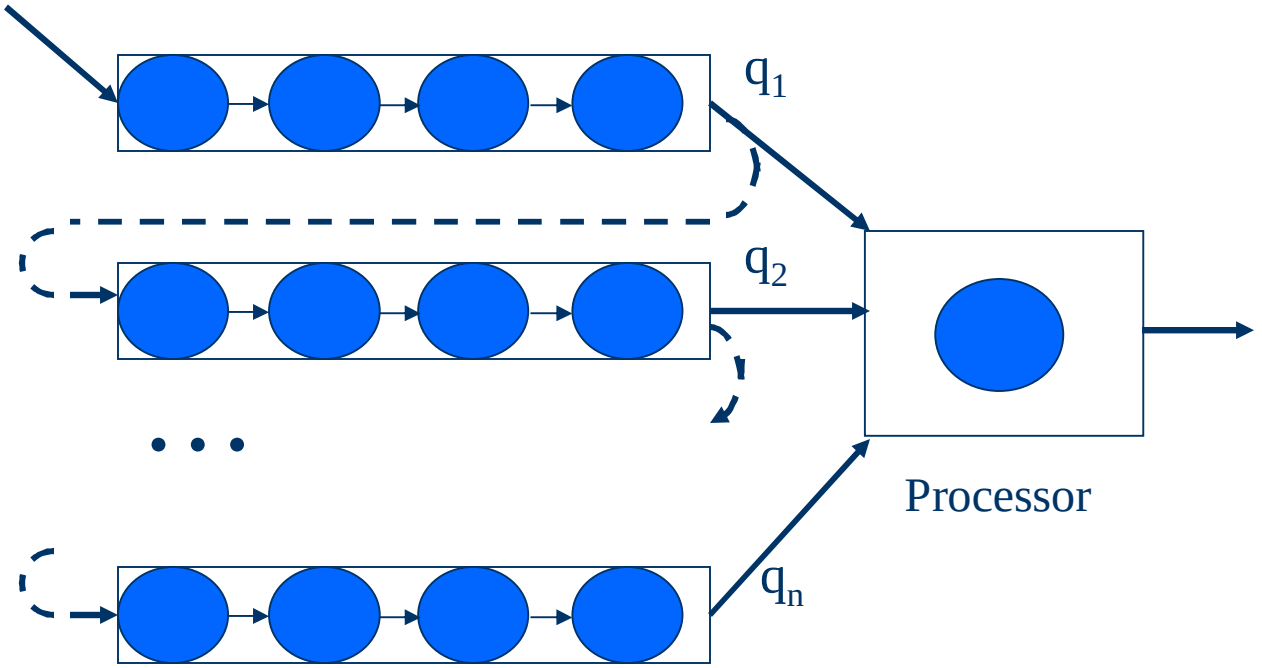


HÀNG ĐA MỨC HỒI TIẾP (*Multilevel Feedback Queue*)

- Định thời theo kiểu **preemptive**
- Hệ thống gồm **n hàng đợi**
- Các **hàng đợi từ 1 đến n-1** được định thời theo kiểu FIFO có quantum time là: $\mathbf{q_1, q_2, \ldots q_{n-1}}$ (thông thường $q_1<q_2<\ldots<q_{n-1}$) .
- Nếu quá trình ở **hàng đợi k** ($1\leq k \leq n-1$) chiếm CPU **hết thời gian q** ➔ sẽ xếp vào **cuối hàng k+1**
- Những **quá trình trong hàng k** ($2\leq k \leq n$) chỉ được phục vụ khi và chỉ khi **không có quá trình** nào trong tất cả các **hàng đợi từ 1 đến k-1**
- Các quá trình ở **hàng đợi thứ n** được định thời theo kiểu **Round Robin**

HÀNG ĐA MỨC HỒI TIẾP (*Multilevel Feedback Queue*)

- Nhận xét



Chương 4

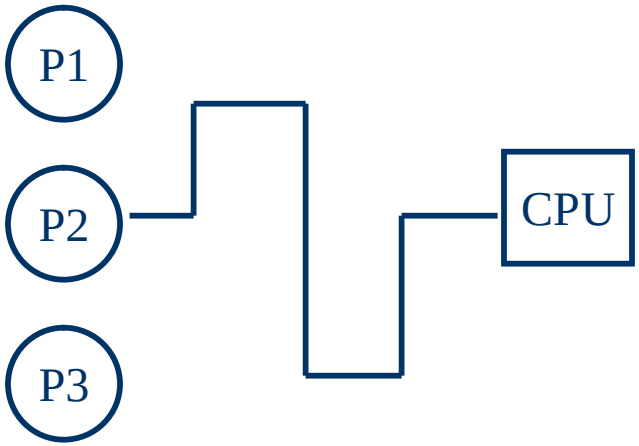
ĐỒNG BỘ GIỮA CÁC QUÁ TRÌNH ĐỒNG THỜI

CHƯƠNG 4 : ĐỒNG BỘ GIỮA CÁC QUÁ TRÌNH ĐỒNG THỜI

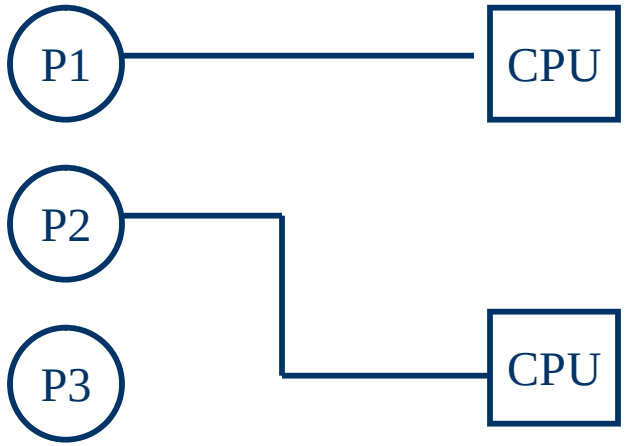
- Các quá trình đồng thời
- Vấn đề tranh chấp và tính loại trừ tương hỗ
- Giải quyết tranh chấp
  - Phương pháp phần mềm
  - Phương pháp phần cứng
  - Nhờ sự hỗ trợ của hệ điều hành
- Một số bài toán về đồng bộ

CÁC QUÁ TRÌNH ĐỒNG THỜI

- Xử lý đồng thời



- Xử lý song song



CẤU TRÚC “parbegin-parend”

- Dùng để mô tả việc xử lý đồng thời / song song

parbegin
Phát biểu 1;
Phát biểu 2;
...
Phát biểu n
parend;

- Một lệnh chỉ bắt đầu xử lý song song các công việc được nêu sau đó.
- Phát biểu sau **parbegin-parend** chỉ được thực hiện khi mọi công việc trong parbegin..parend đã kết thúc.

VÍ DỤ

- **Tính biểu thức sau:**

S := -x + 2*y+(x + 2) * (y-1)

Các công việc cần tính tuần tự :

1. –x
2. 2 * y
3. x + 2
4. y – 1
5. -x + 2*y
6. (x + 2) * (y – 1)
7. –x +2*y + (x + 2) * (y – 1)

- **Có thể xử lý song song:**

```
parbegin
    temp1 := -x;
    temp2 := 2*y;
    temp3 := x+2;
    temp4:= y-1
parend;
parbegin
    temp5 := temp1 + temp2;
    temp6 := temp3 * temp4
parend;
S:= temp5 +temp6;
```

VẤN ĐỀ TRANH CHẤP

- ***Ví dụ 1:***

```
a:=0;
parbegin
    P1;
    P2;
parend
```

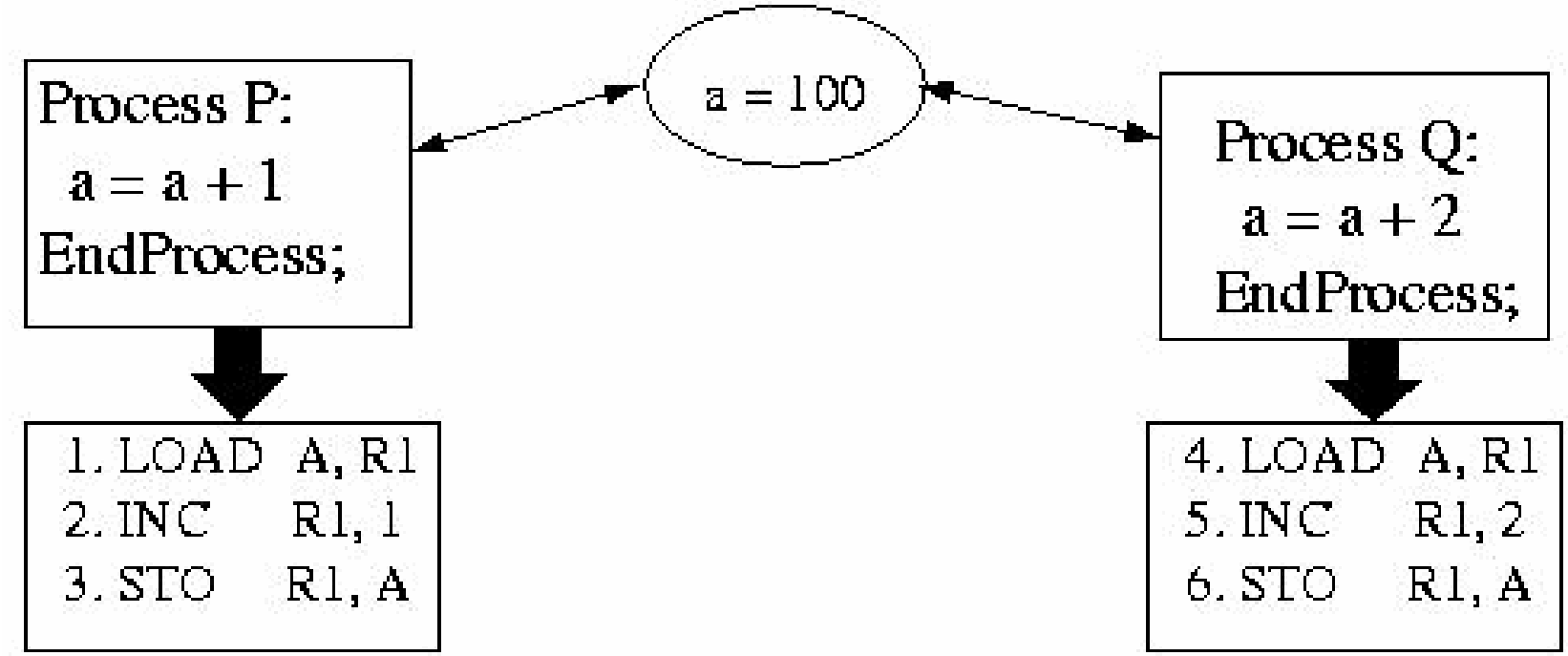
P1:	**P2:**
1. a:=10; 2. b:=20; 3. a:=a+b; 4. print(a);	1. read(a); 2. a:=a+1; 3. print(a);

- ***Ví dụ 2 :***

```
a:=100;
parbegin
    P;
    Q;
parend
print(a);
```

P:	**Q:**
1. a:=a+1;	1. a:=a+2;

VÍ DỤ 2

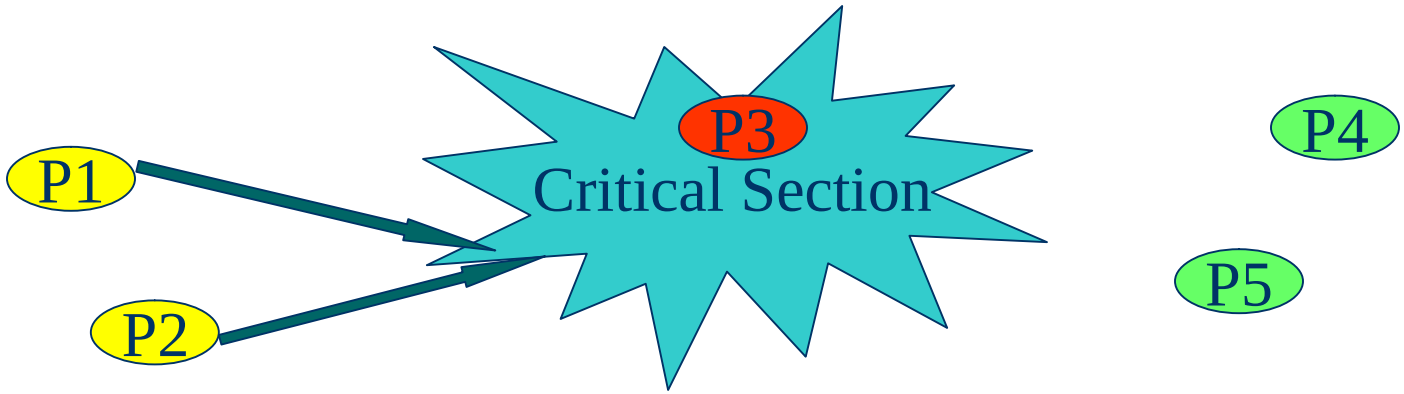


1, 4, 5, 6, 2, 3 ➡ a = 101 and not 103.

GIẢI QUYẾT TRANH CHẤP

- **Nguyên tắc** : khi quá trình nào đang thao tác trên tài nguyên thì quá trình khác phải đợi cho đến khi quá trình đang thao tác hoàn tất công việc

  → Đảm bảo tính loại trừ tương hỗ (*mutual exclusion*) trên vùng tranh chấp (*critical section*)



- **Các vấn đề quan tâm :** vùng tranh chấp, các thao tác liên quan và tính loại trừ tương hỗ

GIẢI THUẬT TỔNG QUÁT

- Mỗi quá trình tham gia vào vùng tranh chấp thực hiện

```
begin
    ...
    enter_mutual_exclusion;
        critical_section;
    exit_mutual_exclusion;
    ...
end;
```

- **enter_mutual_exclusion**

  if *có quá trình đang ở trong vùng tranh chấp* **then** *đợi*
  else *được phép vào*

- **exit_mutual_exclusion**
  cho phép một trong các quá trình đang đợi (nếu có) được vào vùng tranh cháp.

NGUYÊN TẮC GIẢI QUYẾT TRANH CHẤP

1. Các lệnh phần mềm là lệnh đơn vị
2. Các quá trình đồng thời có thể không đồng bộ nhau
3. Quá trình ngoài vùng tranh chấp không có quyền cấm quá trình khác xin vào vùng tranh chấp.
4. Quá trình không bị trì hoãn vô hạn định khi xin vào vùng tranh chấp.

CÁC PHƯƠNG PHÁP GIẢI QUYẾT TRANH CHẤP

- **Các phương pháp hiện thực :**
  - **Phần mềm : giải thuật Dekker, Peterson, Bakery**
  - **Phần cứng : lệnh testandset**
  - **Các cẩu trúc đặc biệt : semaphore, monitor…**
- **Các phương pháp giải quyết tranh chấp có thể là *busy-waiting* hay *sleep-wakeup*:**
  - **Busy waiting: quá trình vẫn chiếm CPU khi không vào vùng tranh chấp được để kiểm tra điều kiện.**
  - **Sleep and Wakeup: ngược lại**

GIẢI THUẬT DEKKER –VERSION1

```
program      versionone;
var     processnumber : integer;

procedure     processone;
 begin
   while true do
    begin
       while processnumber = 2 do ;
          criticalsectionone;
          processnumber := 2;
          otherstuffone;
     end
 end;

procedure     processtwo;
  begin
    while true do
     begin
        while processnumber = 1 do ;
           criticalsectiontwo;
           processnumber := 2;
           otherstufftwo;
    end
  end;

begin
        processnumber := 1;
        parbegin
              processone;
              processtwo;
        parend
```

GIẢI THUẬT DEKKER – VERSION2

```
                    program     versiontwo;
                    var     p1inside, p2inside : boolean;
procedure     processone;                    procedure     processtwo;
  begin                                        begin
        while true do                                while true do
          begin                                        begin
               while p2inside do ;                          while p1inside do ;
               p1inside := true;                            p2inside := true;
               criticalsectionone;                          criticalsectiotwo;
               p1inside := false;                           p2inside := false;
               otherstuffone;                               otherstufftwo;
          end                                          end
  end;                                         end;
                    begin
                          p1inside := false;
                          p2inside := false;
                          parbegin
                                processone;
                                processtwo;
                          parend
```

GIẢI THUẬT DEKKER – VERSION3

```
program     versionthree;
var     p1wantstoenter, p2wantstoenter : boolean;

procedure       processone;
  begin
        while true do
    begin
          p1wantstoenter := true;
          while p2wantstoenter do ;
          criticalsectionone;
          p1wantstoenter := false;
          otherstuffone;
    end
  end;

procedure       processtwo;
  begin
  while true do
    begin
          p2wantstoenter := true;
          while p1wantstoenter do ;
          criticalsectiotwo;
          p2wantstoenter := false;
          otherstufftwo;
    end
  end;

begin
          p1wantstoenter := false;
          p2wantstoenter := false;
          parbegin
                processone;
                processtwo;
          parend
end.
```

GIẢI THUẬT DEKKER – VERSION4

```
program     versionfour;
var     p1wantstoenter, p2wantstoenter : boolean;
procedure       processone;
  begin
    while true do
      begin
          p1wantstoenter := true;
          while p2wantstoenter do
            begin
              p1wantstoenter := false;
              delay(random, fewcycles);
              p1wantstoenter := true;
            end;
          criticalsectionone;
          p1wantstoenter := false;
          otherstuffone;
      end
  end;
```

```
procedure       processtwo;
  begin
    while true do
      begin
          p2wantstoenter := true;
          while p1wantstoenter do
            begin
              p2wantstoenter := false;
              delay(random, fewcycles);
              p2wantstoenter := true;
            end;
          criticalsectiotwo;
          p2wantstoenter := false;
          otherstufftwo;
      end
  end;
```

```
begin
        p1wantstoenter := false; p2wantstoenter := false;
        parbegin
              processone;
              processtwo;
        parend
end.
```

GIẢI THUẬT DEKKER

```
procedure   processone;
  begin
      while true do
        begin
          p1wantstoenter := true;
          while p2wantstoenter do
          if favoredprocess=second then
           begin
             p1wantstoenter := false;
             while favoredprocess = second do;
             p1wantstoenter := true;
           end;
          criticalsectionone;
          favoredprocess := second;
          p1wantstoenter := false;
          otherstuffone;
        end
 end;
```

```
procedure  processtwo;
begin
    while true do
      begin
          p2wantstoenter := true;
          while p1wantstoenter do
            if favoredprocess = first then
             begin
              p2wantstoenter := false;
              while favoredprocess = first do ;
              p2wantstoenter := true;
             end;
          criticalsectiotwo;
          favoredprocess := first;
          p2wantstoenter := false;
          otherstufftwo;
    end
end;
```

GIẢI THUẬT PETERSON

```
program petersonsalgorithm;
var     favoredprocess : (first, second);
        P1wantstoenter, p2wantstoenter : boolean;

procudure processone;
  begin
      while true do
        begin
          p1wantstoenter := true;
          favoredprocess := second;
          while p2wantstoenter and
                favoredprocess = second do ;
          criticalsectionone;
          p1wantstoenter := false;
          otherstuffone;
        end
  end;

procudure processtwo;
begin
    while true do
      begin
        p2wantstoenter := true;
        favoredprocess := first;
        while p1wantstoenter and
              favoredprocess = first do ;
        criticalsectiontwo;
        p2wantstoenter := false;
        otherstufftwo;
      end
end;

begin
        p1wantstoenter := false;
        p2wantstoenter := false;
        favoredprocess := first;
        parbegin
              processone;
              processtwo;
        parend
end.
```

GIẢI THUẬT BAKERY

- Trước khi vào vùng tranh chấp, mỗi quá trình chọn cho mình một con số.
- K quá trình nào giữ k số nhỏ nhất sẽ được vào vùng tranh chấp.
- Nếu hai quá trình $\mathbf{P_i}$ và $\mathbf{P_j}$ giữ hai số bằng nhau thì quá trình $\mathbf{P_i}$ sẽ được vào vùng tranh chấp trước nếu **i < j**.
- Các giá trị mà các quá trình có được là một dãy không giảm: 1,2,3,3,3,3,4,5...

GIẢI THUẬT BAKERY

Process i:

```
repeat
  choosing[i]:=true;
  number[i]:=max(number[0], number[1], ..., number[n - 1]) +1;
  choosing[i]:=false;
  for j:= 0  to  n - 1 do  begin
    while  choosing[ j]  do ;
    while  number[j]≠0  and  (number[j], j) < (number[ i],i) do;
  end;
  critical section;
  number[ i]:=0;
  remainder section;
until false;
```

GIẢI THUẬT BAKERY

- Ghi chú:
  - (a,b) < (c,d): if a < c
    or (a=c) and (b < d)
  - choosing: **array** [0.. n - 1] **of** boolean;

(được khởi động tất cả các giá trị là false)

  - number: **array** [0.. n- 1] **of** integer;

(được khởi động tất cả các giá trị là 0)

PHƯƠNG PHÁP PHẦN CỨNG

- Phải được phần cứng (CPU) hỗ trợ lệnh
- Lệnh **testandset(a,b)**: thực hiện ba thao tác sau liên tục, không bị ngắt quãng:
  1. **read b;**
  2. **a:=b;**
  3. **b:=true;**

  (a, b: kiểu Boolean)

PHƯƠNG PHÁP PHẦN CỨNG

```
var active : boolean;
procedure processone;
  varonecannotenter : boolean;
  begin
     while true do
      begin
        onecannotenter := true;
        while onecannotenter do
            testandset(onecannotenter,
                       active);
        criticalsectionone;
        active := false;
        otherstuffone;
      end
  end;
procedure processtwo;
  vartwocannotenter : boolean;
  begin
     while true do
      begin
        twocannotenter := true;
        while twocannotenter do
            testandset(twocannotenter,
                       active);
        criticalsectiontwo;
        active := false;
        otherstufftwo;
      end
  end;
begin
    active := false;
    parbegin
            processone;
            processtwo;
    parend
end.
```

SEMAPHORE

- Là cầu trúc dữ liệu đặc biệt chỉ cho phép truy xuất thông qua các tác vụ được thực hiện như lệnh đơn vị như sau
  - **init(semaphore S, integer num)** : *S:=num;*
  - **P(semaphore S)**

    if S > 0 then S := S - 1 else wait on S) ;
  - **V(semaphore S**

    if (có quá trình đang đợi trên S)

    then (kích khởi 1 trong các quá trình đó)

    else S := S + 1;
- Có hai loại semaphore:
  - Semaphore nhị phân: có giá trị 0 hoặc 1
  - Semaphore đến : có giá trị từ 2 trở lên
- Hiện thực semaphore : cấp hệ điều hành / cấp phần mềm

CÁC PHƯƠNG PHÁP GIẢI QUYẾT TRANH CHẤP KHÁC

- **Mutex**
- **Lock file**
- **Lock, Unlock**
- **Monitor**
- **Condition variable**

SỬ DỤNG SEMAPHORE

- var S : semaphore;
- Process P_i

  enter_critical_section = P (S)

  exit_critical_section = V (S)
- Lúc đầu phải khởi động semaphore S bằng số quá trình tối đa được phép vào vùng tranh cháp đồng thời

  init(S, 1):

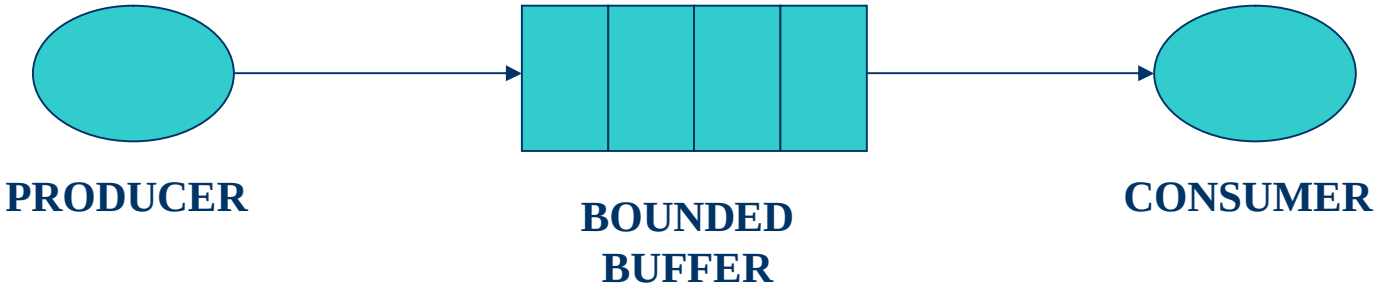
  init (S, num);

ĐỒNG BỘ NHIỀU QUÁ TRÌNH

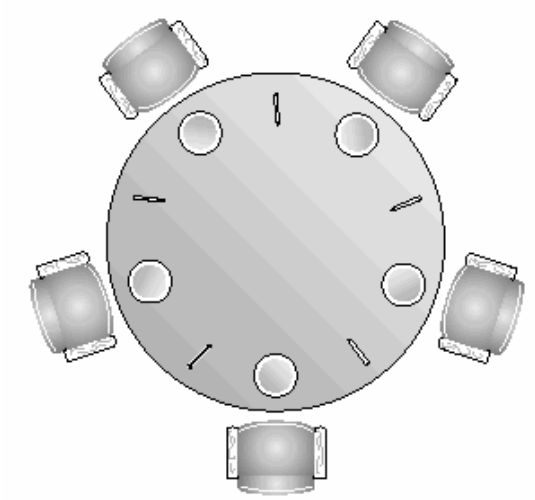
- Vấn đề
  - Vùng tranh chấp cho phép tối đa k quá trình cùng vào đồng thời
- Giải quyết
  - Dùng semamphore
  - Giải thuật Bakery

MỘT SỐ BÀI TOÁN VỀ ĐỒNG BỘ

- Bài toán Producer-Consumer



- Bài toán 5 triết gia ăn tối



- Bài toán tiệm cắt tóc

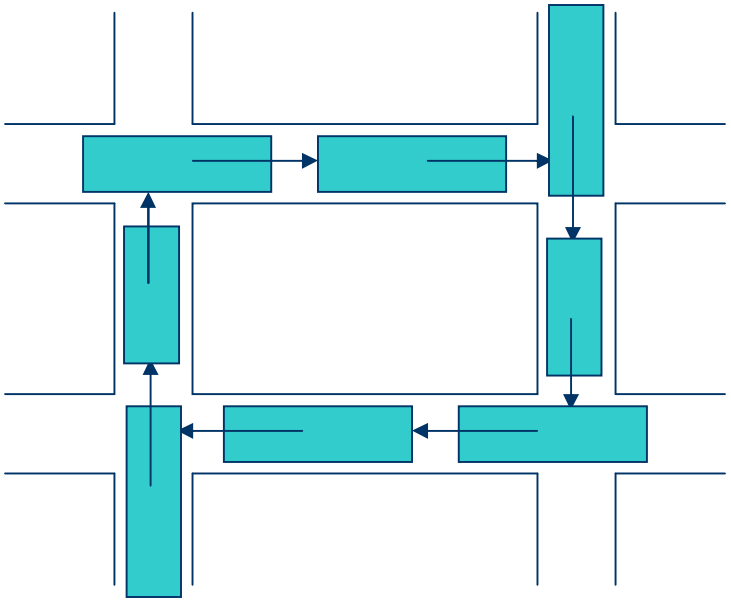
Chương 5
DEADLOCK

CHƯƠNG 5 : DEADLOCK

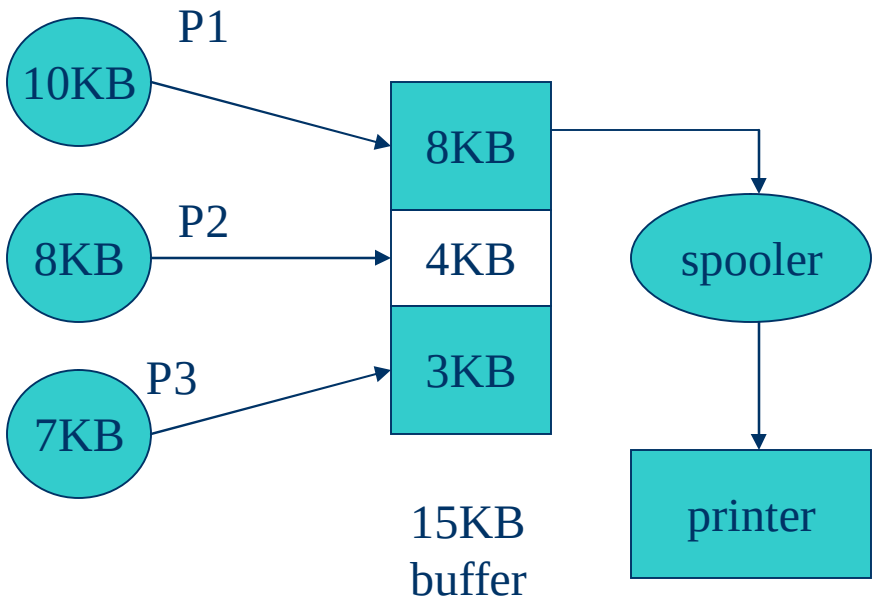
- Định nghĩa deadlock
- Điều kiện để có deadlock
- Các phương pháp giải quyết
  - Chống deadlock
  - Tránh deadlock
  - Phát hiện deadlock
  - Phục hồi deadlock
- Bài tập

ĐỊNH NGHĨA

- Quá trình deadlock : đợi một sự kiện không bao giờ xảy ra.
- Một hệ thống bị deadlock : có quá trình bị deadlock.



Tắc nghẽn trong giao thông



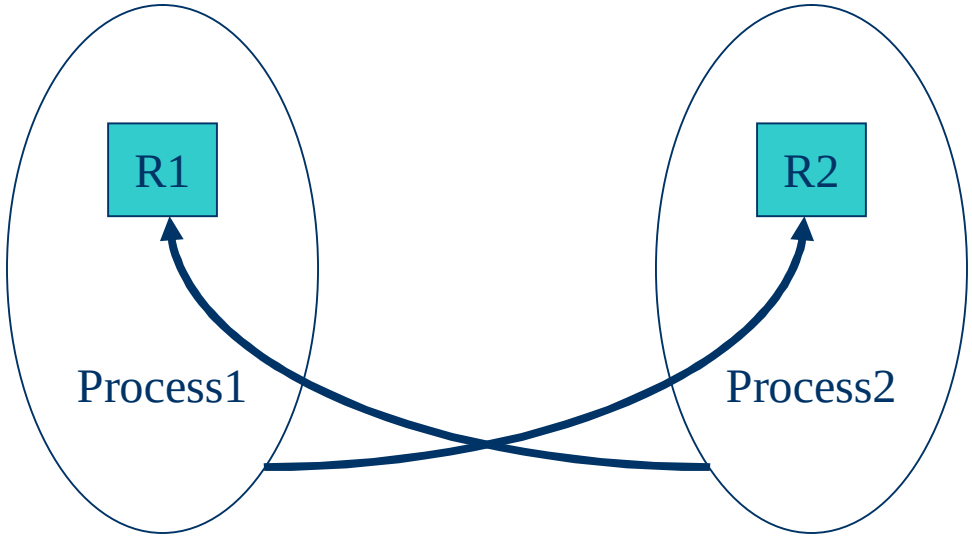
Tắc nghẽn trong quản lý in ấn

VÍ DỤ & BẢN CHẤT DEADLOCK

Hai quá trình bị deadlock:

Process1	Process2
P(S1);	P(S2);
P(S2);	P(S1);
Critical Section;	*Critical Section;*
V(S1);	V(S2);
V(S2);	V(S1);

Dạng deadlock:



DEADLOCK VÀ TRÌ HOÃN VÔ HẠN ĐỊNH

- **Deadlock**
  - Đợi sự kiện không bao giờ xảy ra
  - Nguyên nhân ?
- **Trì hoãn vô hạn định** (*Indefinite postponement*)
  - Đợi sự kiện có thể xảy ra nhưng không xác định thời điểm
  - Biểu hiện như deadlock
  - Nguyên nhân ?
- **Deadlock có khác vòng lặp vô hạn ?**

ĐIỀU KIỆN XẢY RA DEADLOCK

1. Điều kiện **mutual exclusion**: các quá trình cần thực hiện loại trừ tương hỗ trên vùng tranh chấp
2. Điều kiện **hold & wait**: quá trình đang giữ tài nguyên có thể yêu cầu thêm tài nguyên khác
3. Điều kiện **no-preemption**: tài nguyên chỉ được giải phóng khi quá trình dùng xong
4. Điều kiện **circular-wait**: các quá trình giữ và đợi tài nguyên tạo thành vòng luẩn quẩn

GIẢI QUYẾT DEADLOCK

- **Ngăn ngừa deadlock *(deadlock prevention)***
  - Qui định cấp , dùng tài nguyên nghiêm ngặt
  - Không cho các điều kiện deadlock xảy ra
- **Tránh deadlock *(deadlock avoidance)***
  - Vẫn cho các điều kiện deadlock tồn tại
  - Cấp tài nguyên hợp lý, an toàn
- **Phát hiện deadlock *(deadlock detection)***
- **Phục hồi deadlock *(deadlock recovery)***

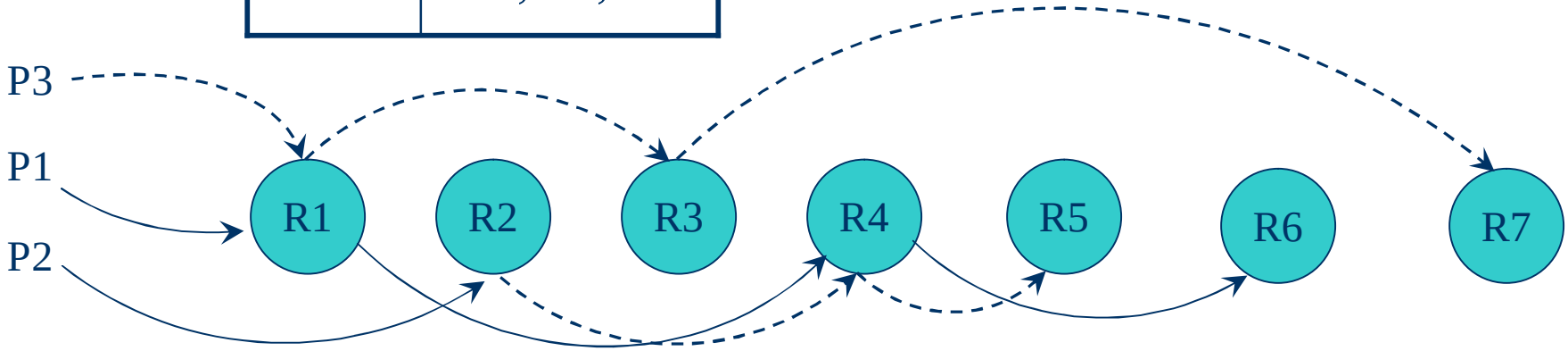
NGĂN NGỪA DEADLOCK (*Havender*)

- Cấm điều kiện **multual-exclusion** ?
- Cấm điều kiện **hold & wait**
  - Quá trình yêu cầu tất cả tài nguyên một lần
  - Chỉ được xử lý khi đã đủ tất cả tài nguyên cầøn thiết
- Cấm điều kiện **no-preemption**
  - Nếu yêu cầu tài nguyên không được, quá trình phải giải phóng tất cả tài nguyên đang giữ và yêu cầu lại. **(?)**
- Loại bỏ **circular-wait**
  - Sắp xếp tài nguyên theo trật tự và chung cấp cho quá trình theo đúng trật tự đó. **Chứng minh ?**

NGĂN NGỪA DEADLOCK (*Havender*)

- **Ví dụ**

Quá trình	Yêu cầu thực tế
P1	R6, R4, R1
P2	R2, R5, R4
P3	R3, R7, R1



- **Nhận xét về p/p ngăn ngừa deadlock**

TRÁNH DEADLOCK

- **Giải thuật nhà băng (Banker's Algorithm)**
  - Hệ điều hành ~ nhà Băng
  - Quá trình ~ khách hàng
  - Tài nguyên ~ vốn vay
- **Ràng buộc**
  - Yêu cầu vay cực đại ≤ vốn nhà băng
  - Khách không trả vốn nếu vay chưa đủ yêu cầu cực đại
  - Khi vay đủ, khách phải trả đủ vốn sau thời gian hữu hạn
- **Trạng thái nhà băng**
  - An toàn (**Safe**): thỏa yêu cầu mọi khách, ngân hàng thu vốn đủ
  - Không an toàn (**Unsafe**) : ngược lại ➔ có thể deadlock
- **Hệ thống phải cấp phát tài nguyên sao cho không rơi vào trạng thái Unsafe**

VÍ DỤ

- Trạng thái sau là an toàn. Tại sao ?

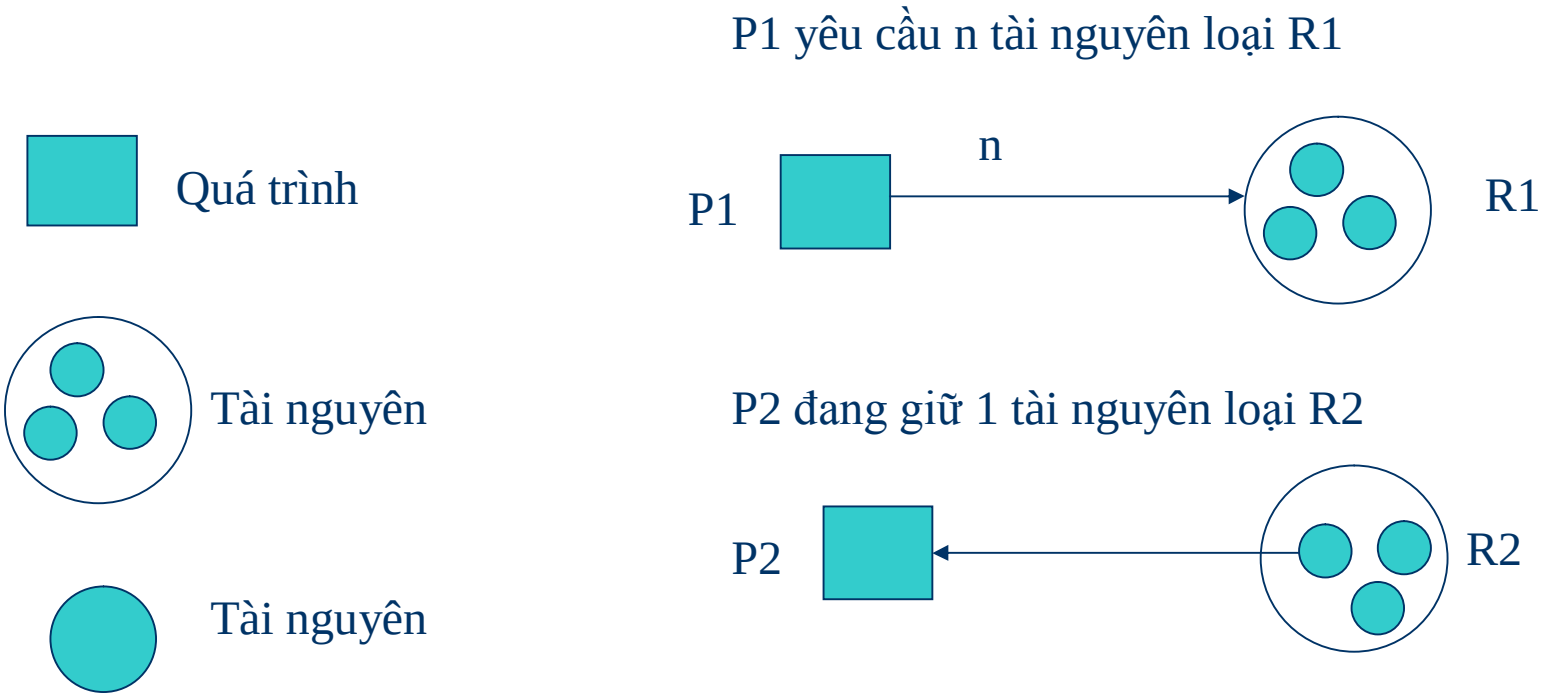
	Đang mượn	Cần tối đa	Cần thêm
P1	1	4	3
P2	4	6	2
P3	5	8	3
Vôn 12 , còn lại 2			

- Trạng thái sau là không an toàn. Tại sao ?

	Đang mượn	Cần tối đa	Cần thêm
P1	8	10	2
P2	2	5	3
P3	1	3	2
Vôn 12 , còn lại 1			

PHÁT HIỆN DEADLOCK

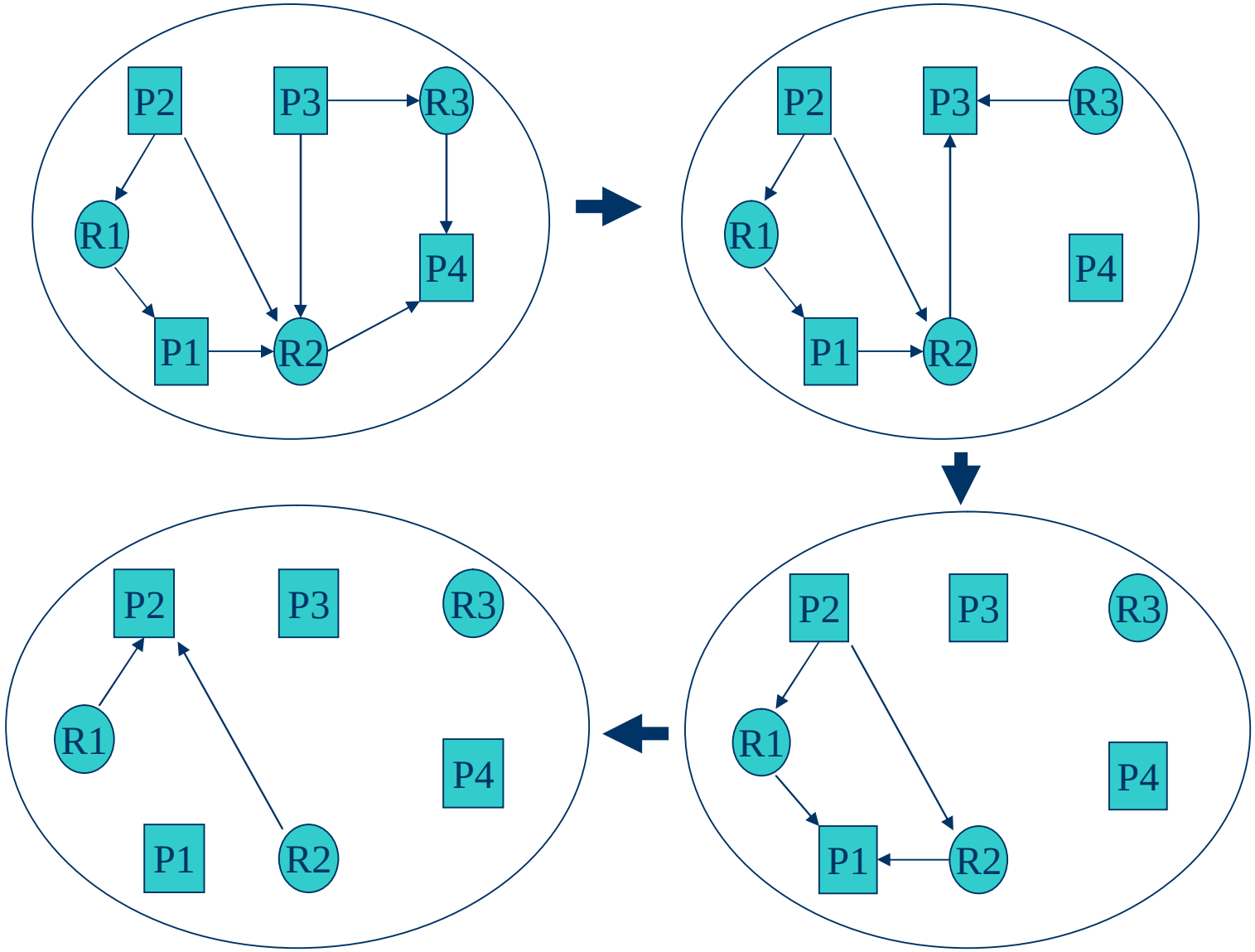
- Ghi nhận, theo dõi yêu cầu, sự cấp phát tài nguyên cho các quá trình.
- Dùng đồ thị RAG (*Resource Allocation Graph*)



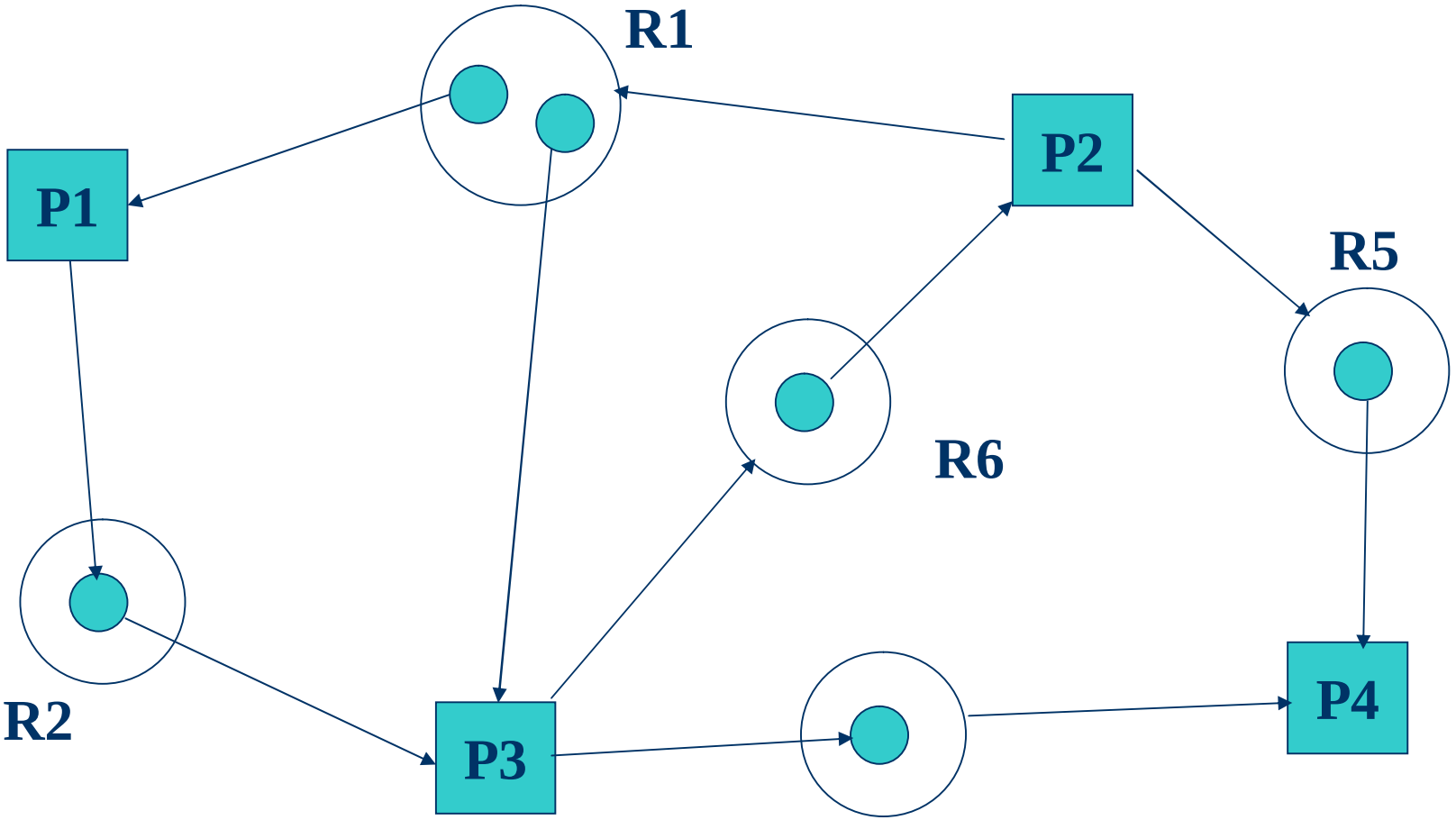
PHÁT HIỆN DEADLOCK

- Giản lược RAG
  1. Tài nguyên rảnh → cấp cho quá trình yêu cầu
  2. Quá trình đủ tài nguyên
     → xoá mọi cạnh vào, xoá quá qtrình
  3. Lặp lại 1 với các quá trình khác đến khi tối giản
- Khi giải thuật dừng
  - RAG không còn cạnh: không có deadlock
  - RAG có chu trình (cycle): deadlock
- Nhận xét
  - Phí tổn lớn

VÍ DỤ 1 : GIẢN ƯỚC RAG



BÀI TẬP : GIẢN ƯỚC RAG



PHỤC HỒÀI DEADLOCK

- Đi kèm với p.p phát hiện deadlock
- Thực hiện
  - Chọn lựa quá trình để thu hồi tài nguyên
  - Treo (*suspend*) quá trình
  - Thu hồi tài nguyên (*preemptive*)
  - Phục hồi (*resume*) các quá trình còn lại
- Khó có thể giải quyết trọn vẹn

BÀI TẬP

1. Hệ thống 1 tốn 10% thời gian cho mỗi ứng dụng để ngăn ngừa deadlock. Hệ thống 2 không ngăn ngừa nên cần 10% thời gian ứng dụng để phục hồi mỗi ứng dụng bị deadlock. So sánh phí tổn 2 hệ thống.

2. Tìm trạng thái của hệ thống sau

Khách	Đã mượn			Nhu cầu tối đa		
	A	**B**	**C**	**A**	**B**	**C**
P1	1	1	0	1	3	0
P2	0	1	1	1	1	2
P3	1	0	0	1	1	3
P4	0	1	0	2	1	0

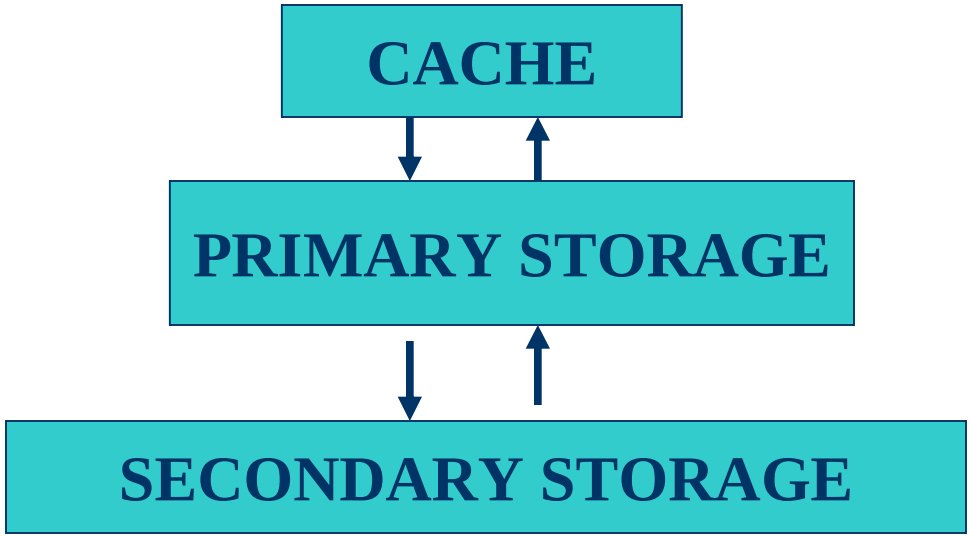
Chương 6
BỘ NHỚ THỰC

CHƯƠNG 6 : BỘ NHỚ THỰC

- Phân cấp bộ nhớ & các vấn đề quan tâm
- Các chiến lược quản lý bộ nhớ
  - Chiến lược nạp
  - Chiến lược sắp đặt
  - Chiến lược thay thế
- Tổ chức bộ nhớ thực
  - Đơn lập trình
  - Đa lập trình phân đoạn cố định / thay đổi
  - Đa lập trình có thay thế vùng nhớ

PHÂN CẤP BỘ NHỚ

- **Từ trên xuống**
  - Tốc độ giảm
  - Dung lượng tăng
  - Giá thành giảm



- **Các vấn đề quan tâm**
  - Bộ nhớ chính chứa 1 hay nhiều quá trình ?
  - Các qt dùng vùng nhớ như nhau / khác nhau ?
  - Bảo vệ vùng nhớ của OS và của từng qt ?
  - Vùng nhớ của qt là liên tục / gián đoạn ?

CHIẾN LƯỢC QUẢN LÝ BỘ NHỚ

- Chiến lược nạp (*fetch strategies*)
  - Nạp phần nào của quá trình vào bộ nhớ và khi nào nạp ?
  - Nạp theo yêu cầu & nạp tiên đoán
- Chiến lược sắp đặt (*placement strategies*)
  - Nạp quá trình mới vào đâu ?
- Chiến lược thay thế (*replacement strategies*)
  - Đưa quá trình nào ra bộ nhớ phụ ?

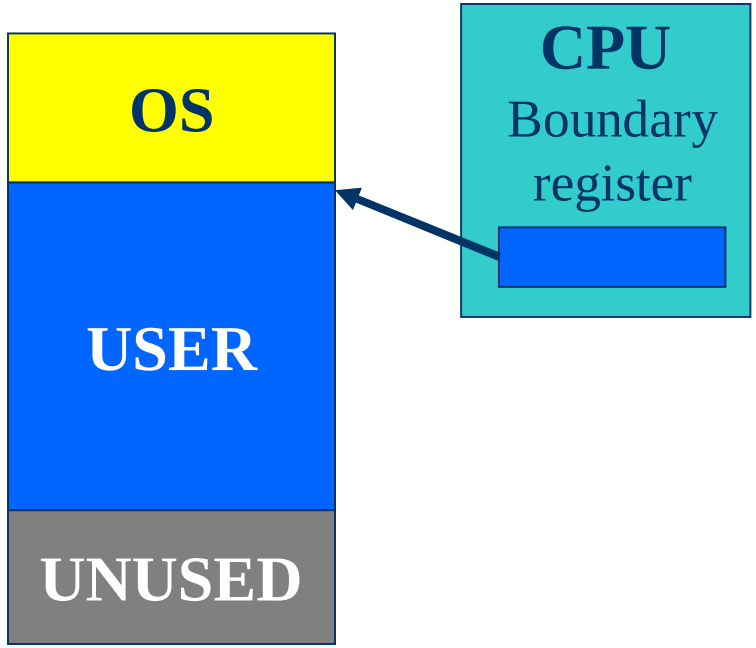
TỔ CHỨC BỘ NHỚ THỰC

- Cấp phát bộ nhớ liên tục
  - Đơn lập trình
  - Đa lập trình phân đoạn cố định
  - Đa lập trình phân đoạn thay đổi
  - Đa lập trình có thay thế vùng nhớ
- Cấp phát bộ nhớ không liên tục

HỆ THỐNG PHÂN PHỐI LIÊN TỤC MỘT NGƯỜI DÙNG

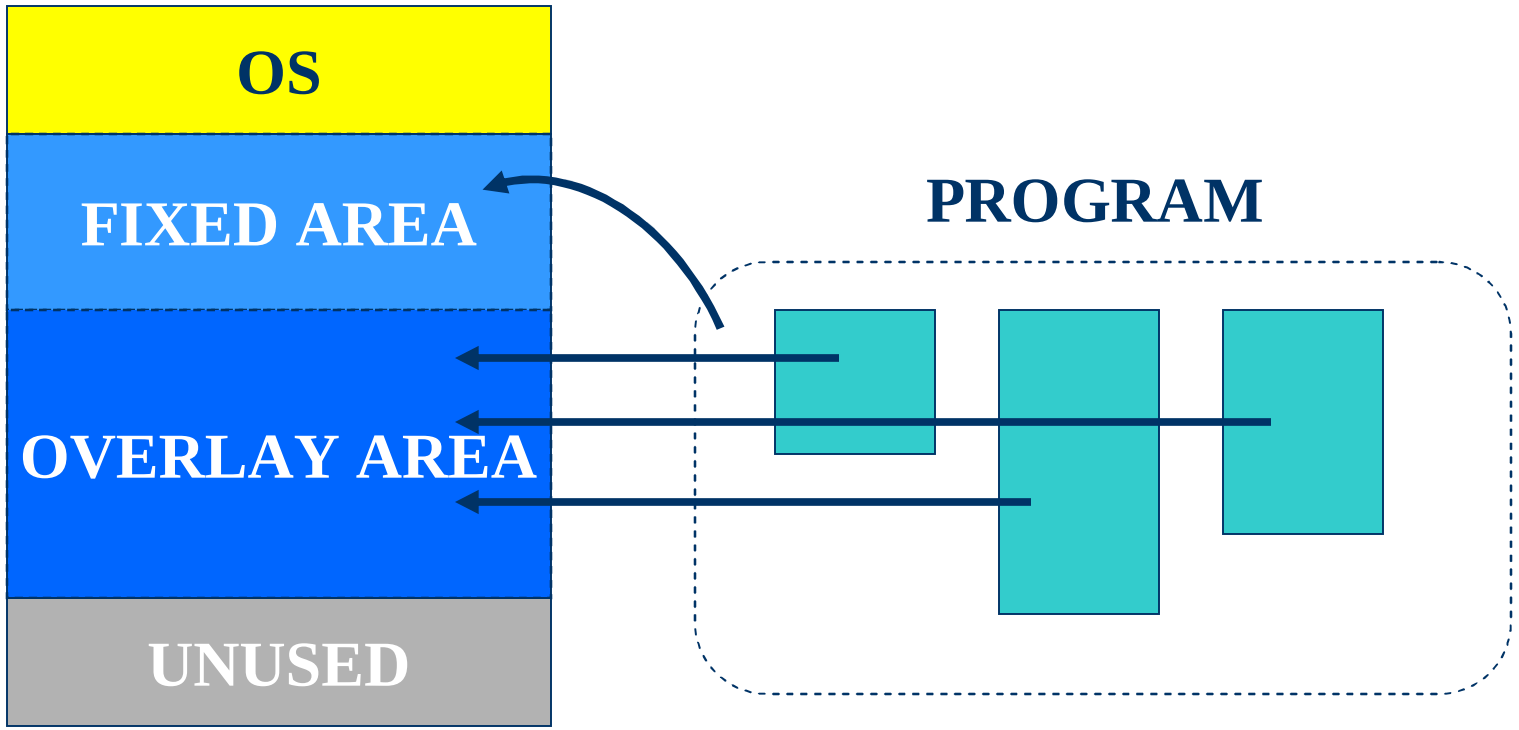
(Single User Continuous Storage Allocation)

- Phục vụ 1 qt, 1 user
- Bảo vệ vùng nhớ ?
- Không cần chiến lược sắp đặt và thay thế
- Kỹ thuật nap : overlay



KỸ THUẬT OVERLAY

- Dùng để chạy chương trình có kích thước lớn hơn kích thước bộ nhớ thực



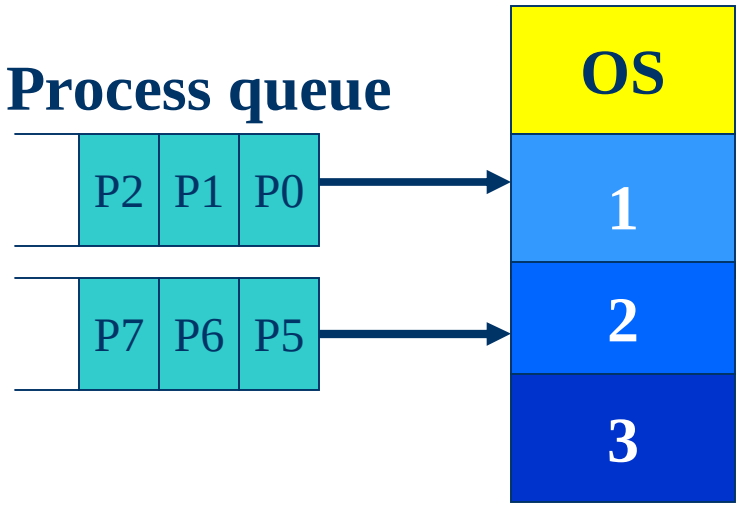
HỆ THỐNG ĐA CHƯƠNG PHÂN ĐOẠN CỐ ĐỊNH

(Fixed Partitioning Multiprogramming)

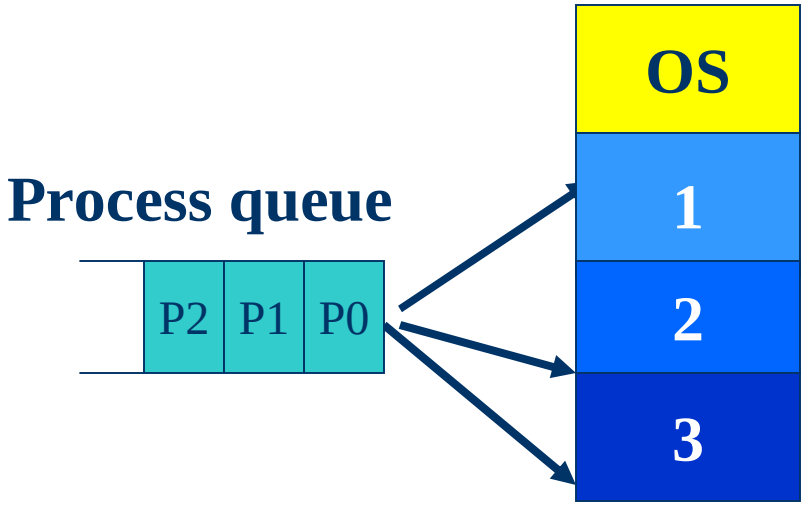
- Chia bộ nhớ n phần bằng nhau, mỗi qt 1 phần

➔ Không cần chiến lược sắp đặt

- Dịch và nạp cố định

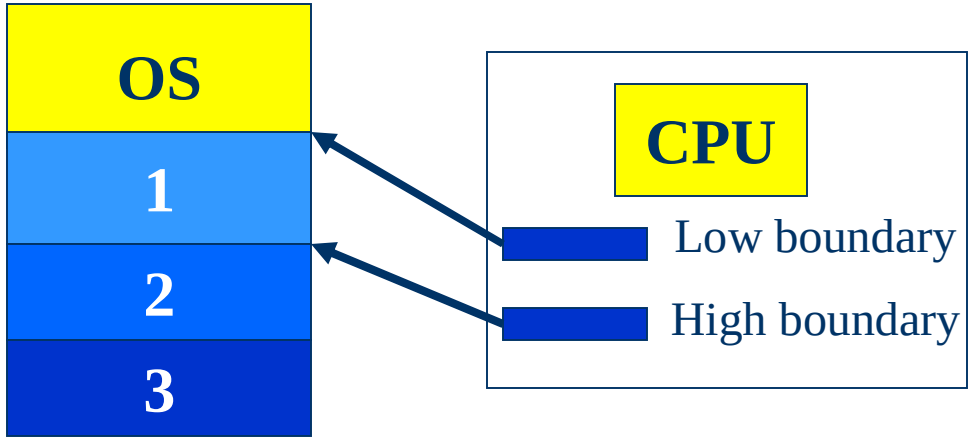


- Dịch và nạp xác định lại



HỆ THỐNG ĐA CHƯƠNG PHÂN ĐOẠN CỐ ĐỊNH

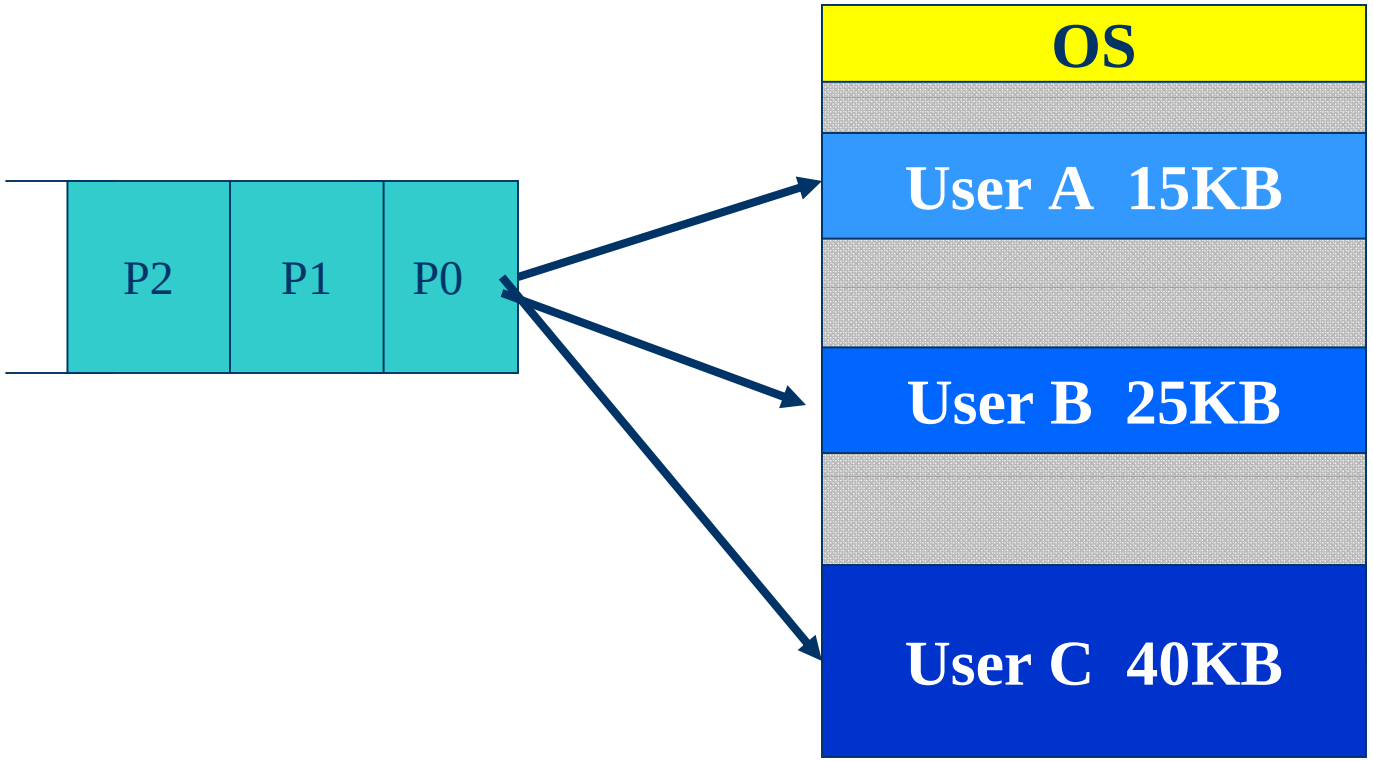
- Bảo vệ vùng nhớ



- Vấn đề phân mảnh vùng nhớ (*fragmentation*)

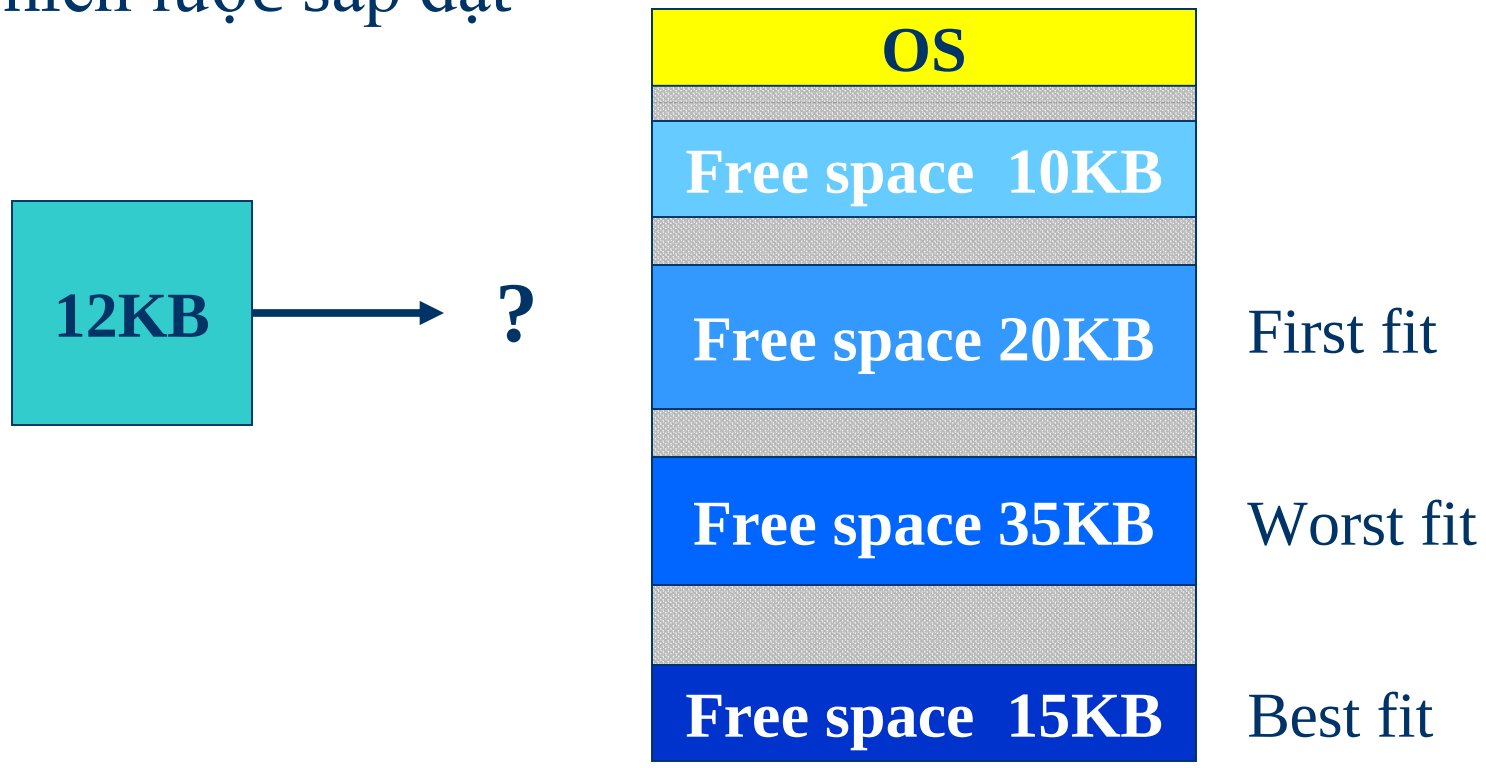
HỆ THỐNG ĐA CHƯƠNG PHÂN ĐOẠN THAY ĐỔI

(Variable Partitioning Multiprogramming)



HỆ THỐNG ĐA CHƯƠNG PHÂN ĐOẠN THAY ĐỔI

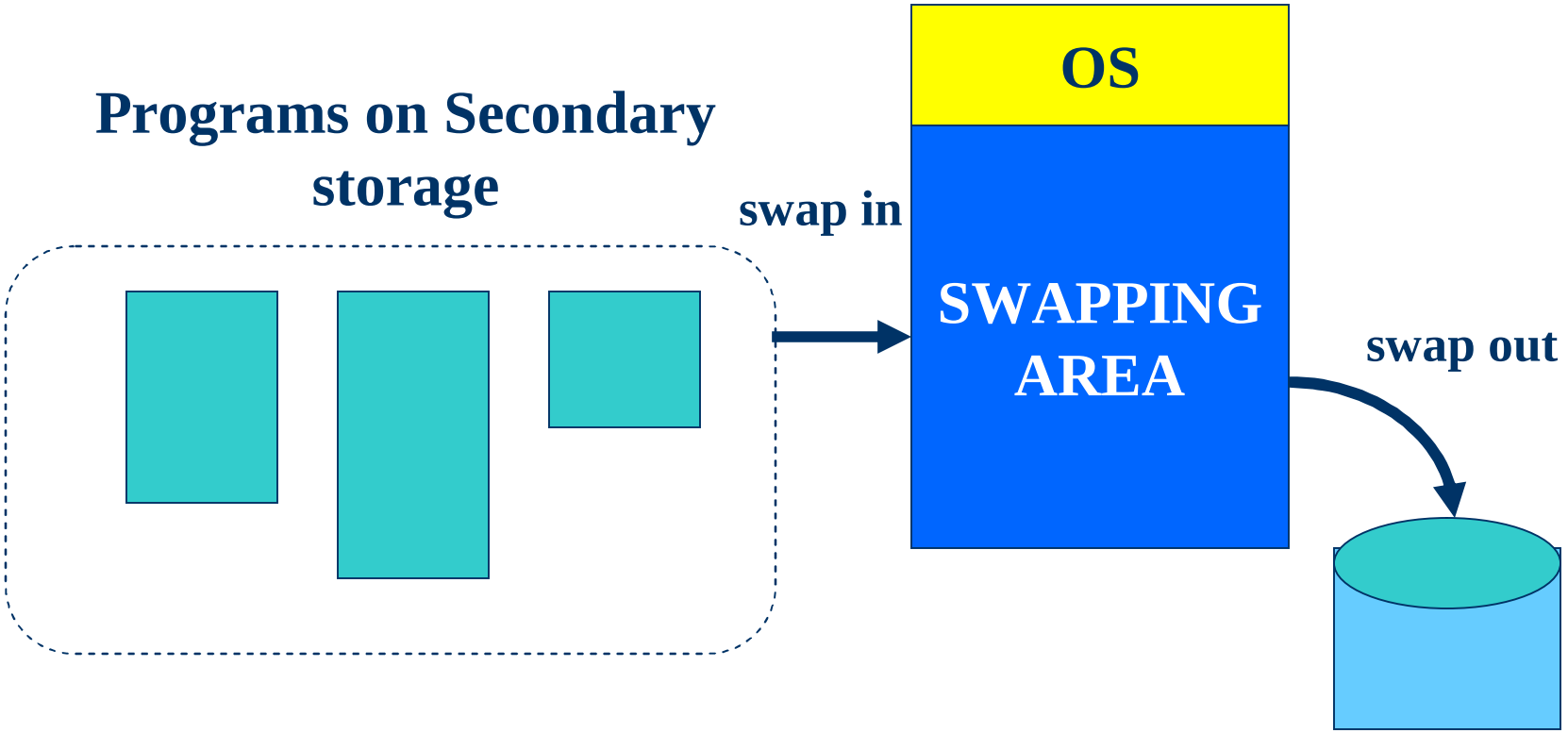
- Chiến lược sắp đặt



- Vấn đề phân mảnh vùng nhớ (*fragmentation*)

HỆ THỐNG ĐA CHƯƠNG CÓ THAY THẾ VÙNG NHỚ

(Multiprogramming With Storage Swapping)



CHƯƠNG 7 : TỔ CHỨC BỘ NHỚ ẢO

- Khái niệm bộ nhớ ảo
- Ánh xạ địa chỉ`
- Kỹ thuật phân trang
- Vấn đề xác định kích cỡ trang
- Kỹ thuật phân đoạn
- Phối hợp phân trang và phân đoạn
- Bài tập

VÍ DỤ MINH HỌA VỀ BỘ NHỚ

```
1.  int a = 0, *p ;
2.  void main( int argc, char *argv[ ] ) {
3.        p=&a;
4.        printf("Address of a =%u", p);
5.       ….thực hiện các công việc không làm thay đổi giá trị p…
6.       printf("Address of a =%u", p);
7.  }
```

Câu hỏi :

1. Các địa chỉ của a ở trên là địa chỉ trên bộ nhớ vật lý (RAM) ?

2. Địa chỉ của a in ra ở các dòng 4 & 6 có khi nào khác nhau không ?

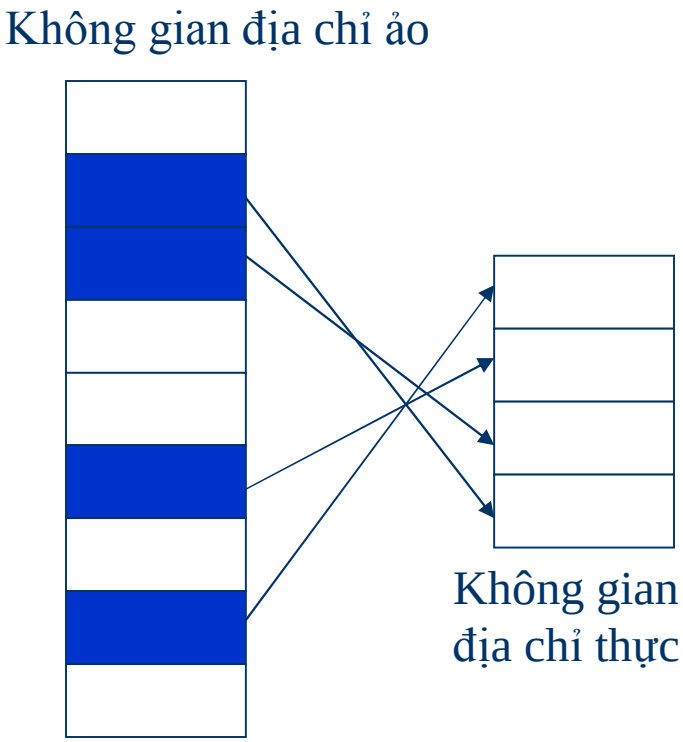
KHÁI NIỆM BỘ NHỚ ẢO

- **Là hình ảnh của bộ nhớ thực**
- **Tách rời địa chỉ quá trình truy cập và địa chỉ trên bộ nhớ thực**
  - Địa chỉ ảo V: tham khảo bởi process
  - Địa chỉ thực R : có trong bộ nhớ thực

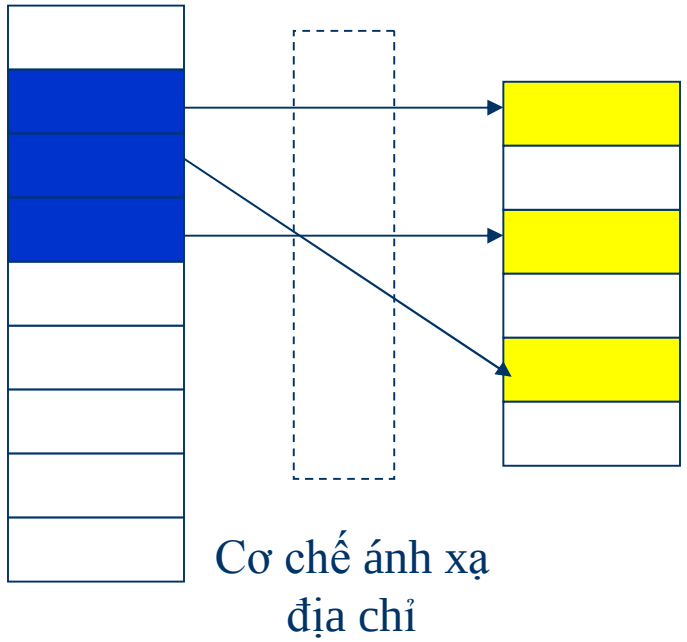
    |V| >> |R|
- **Địa chỉ ảo được ánh xạ thành địa chỉ thực mỗi khi quá trình thực thi → dynamic address translation**
- **Sự cần thiết của bộ nhớ ảo**
  - Dễ phát triển ứng dụng
  - Lưu trữ được nhiều quá trình trong bộ nhớ
  - Tái định vị (relocation) các quá trình
  - Cho các quá trình chia xẻ vùng nhớ dễ dàng

ÁNH XẠ ĐỊA CHỈ

- **Cách thực hiện : ánh xạ khối (hình 1)**
- **Dùng giả lập sự liên tục của bộ nhớù (hình 2)**

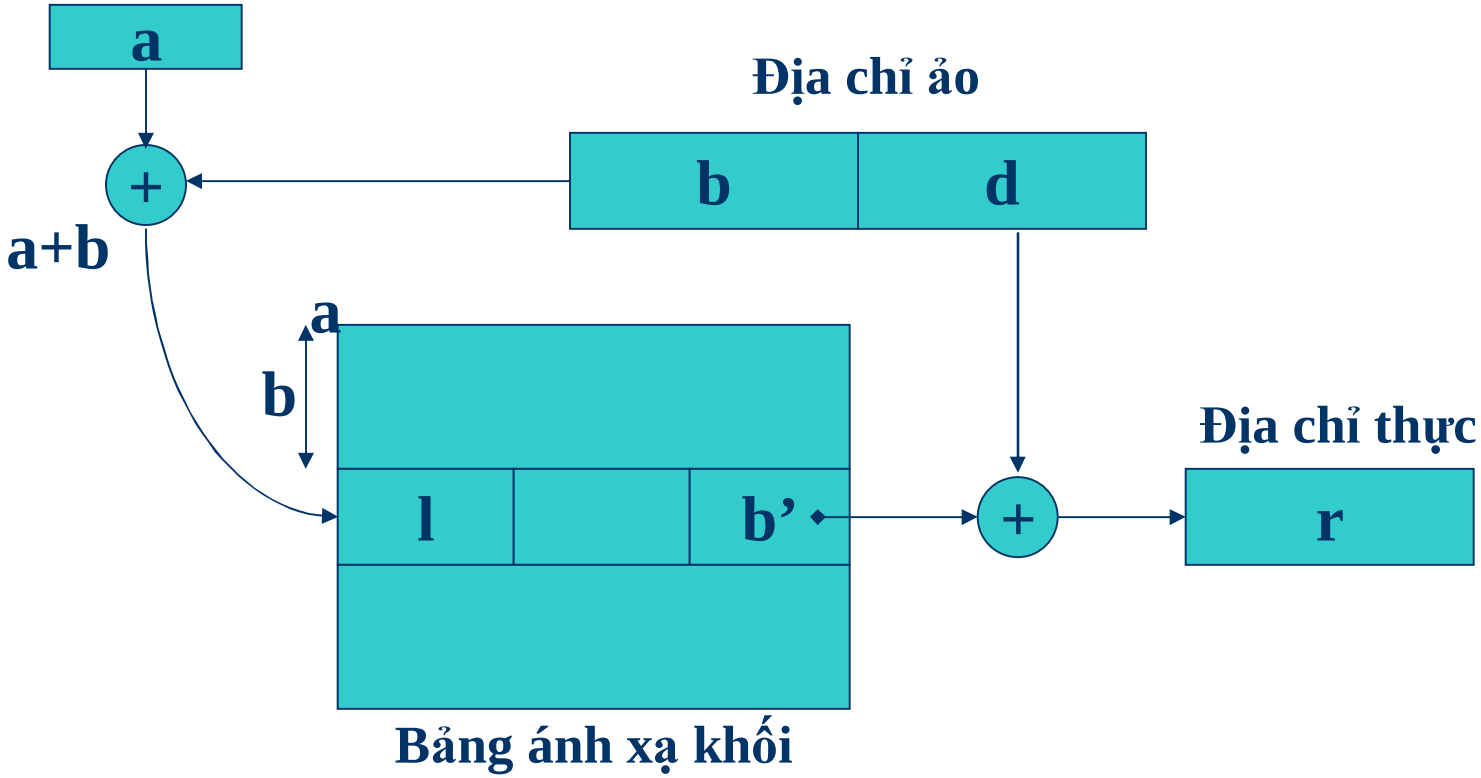


Hình 1



Hình 2

CÁCH THỰC HIỆN ÁNH XẠ KHỐI



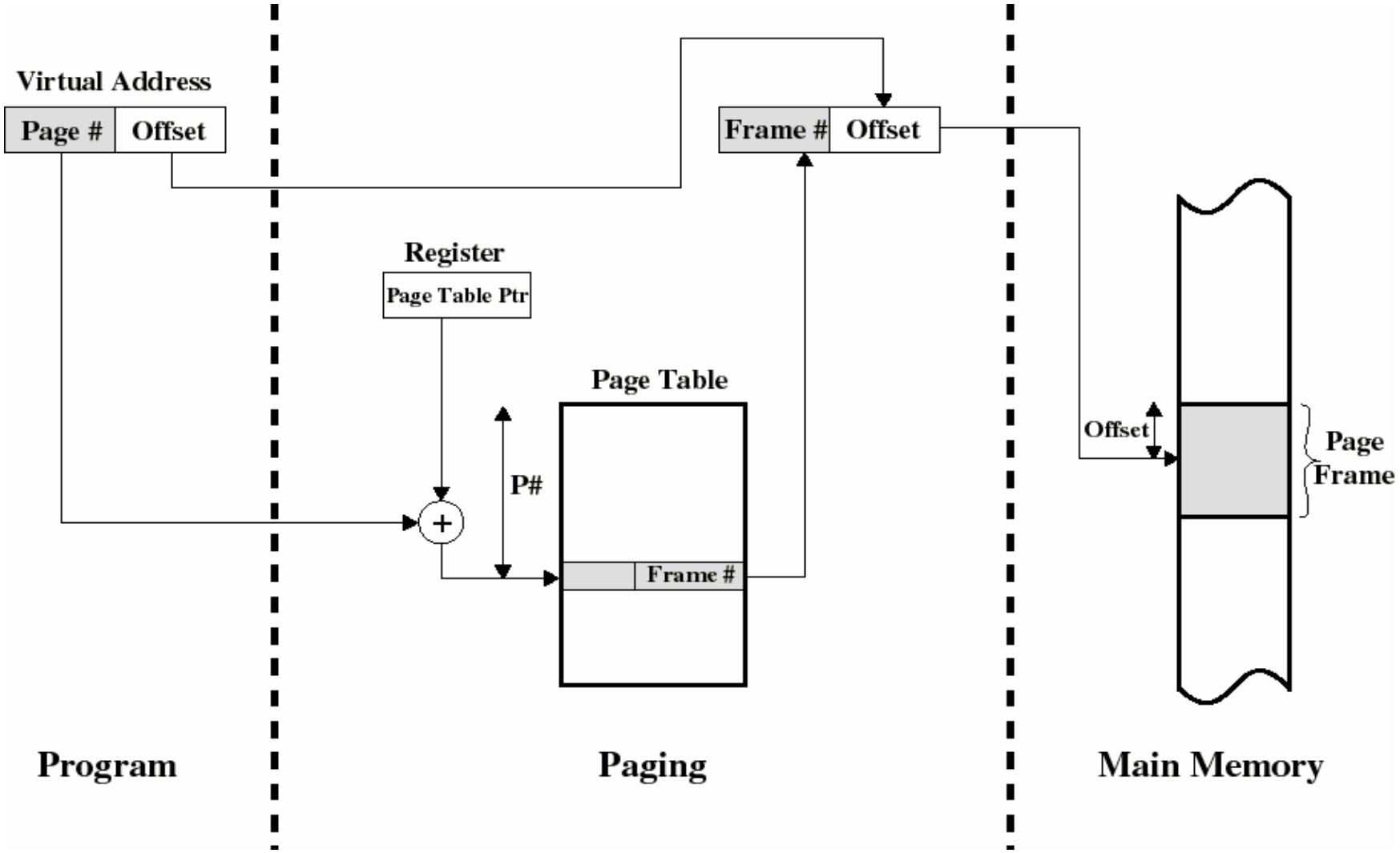
b = chỉ số khối **a= địa chỉ bảng ánh xạ khối** **b'= chỉ số khối thực**

d= độ dời trong khối **l = bít hiện diện**

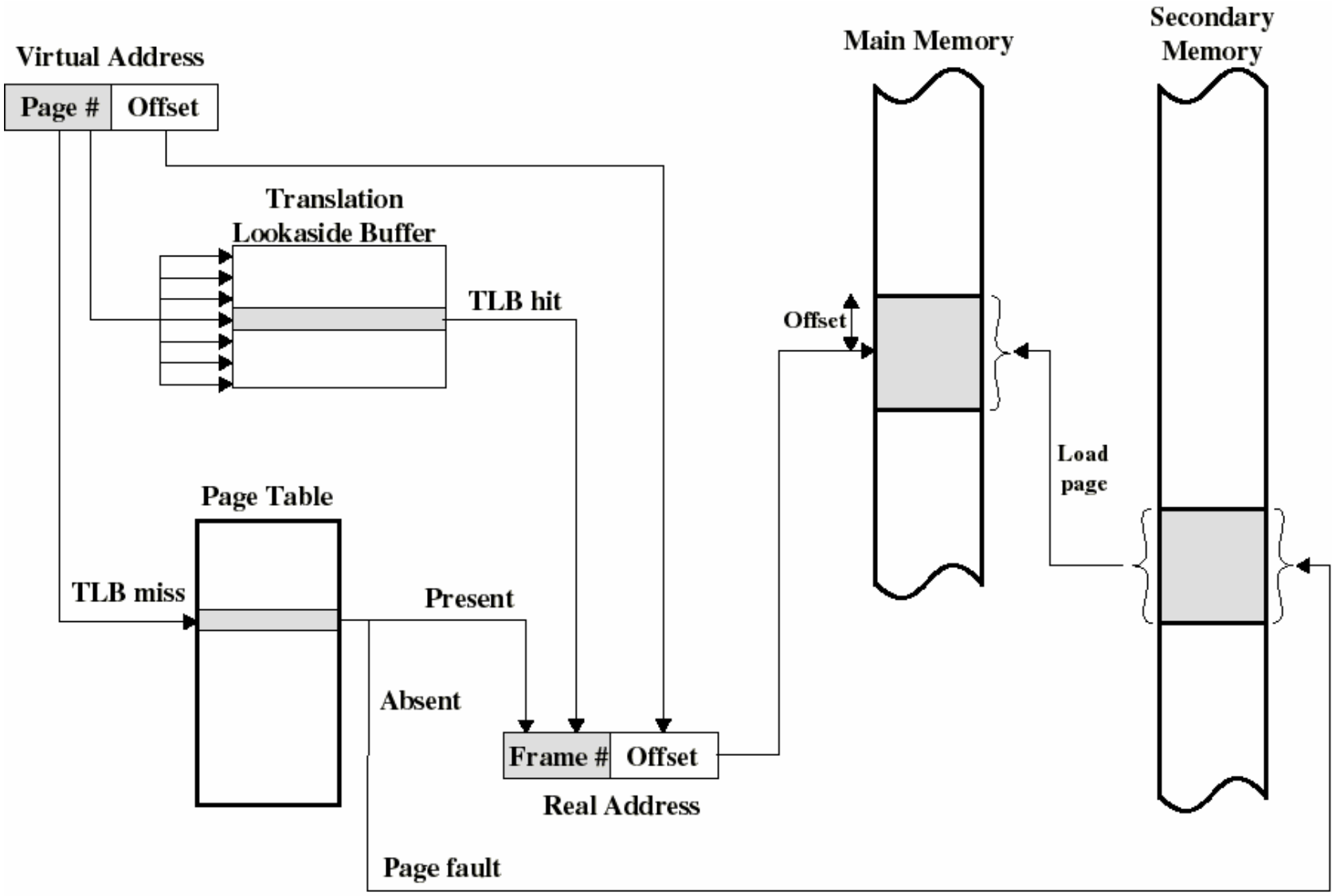
KỸ THUẬT PHÂN TRANG (*PAGING*)

- **Các khối bộ nhớ có kích thước bằng nhau**
  - Khối trên bộ nhớ ảo : trang (***page***)
  - Khối trên bộ nhớ thựïc : ***page frame***
- **Mỗi địa chỉ ảo có hai thành phần:**
  - Chỉ số trang (***page number***)
  - Độ dời của ô nhớ trong trang đó (***offset***)
- **Mỗi quá trình có một bảng ánh xạ trang (*page table*)**
- **Mỗi mục (*entry*) của bảng ánh xạ trang chứa**
  - Present bit
  - Secondary storage address
  - Page frame number
  - Modified bit
  - Các bít điều khiển khác
- **Dùng 1 register chứa địa chỉ thực của bảng ánh xạ trang của quá trình đang chạy**

ÁNH XẠ ĐỊA CHỈ TRỰC TIẾPTRONG HỆ THỐNG PHÂN TRANG

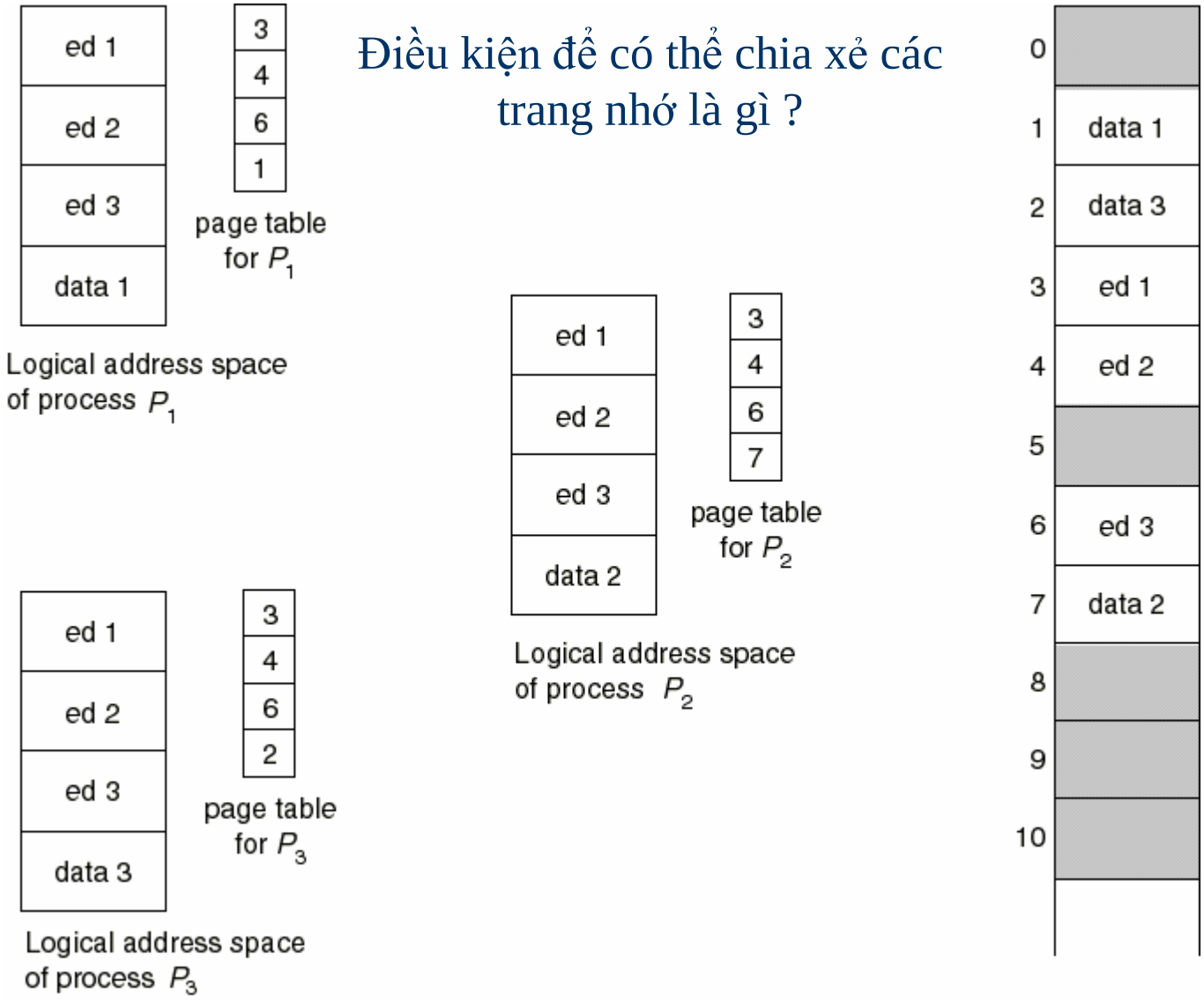


ÁNH XẠ TRANG DÙNG BỘ NHỚ KẾT HỢP



DÙNG CHUNG BỘ NHỚ

Điều kiện để có thể chia xẻ các trang nhớ là gì ?



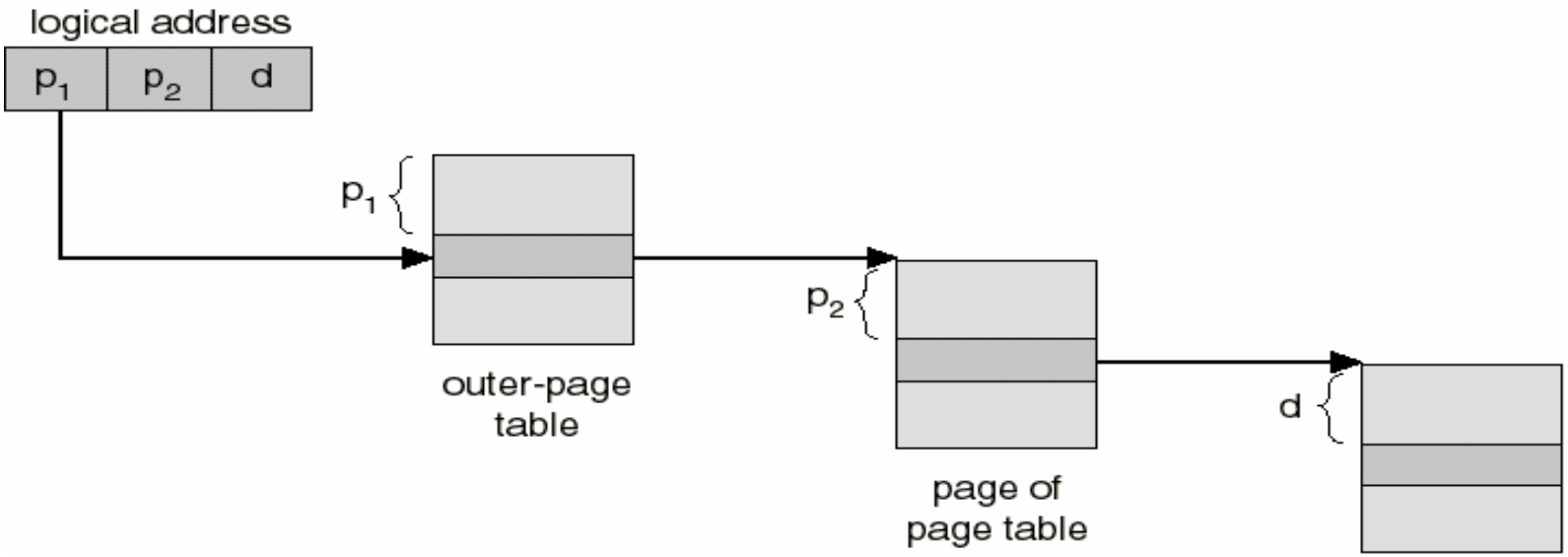
LƯU TRỮ BẢNG ÁNH XẠ TRANG

- **Không gian địa chỉ ảo rất lớn**
  - Dùng 32 → 64 bit địa chỉ
  - Với 32 bít địa chỉ, trang có size 4KB, bảng ánh xạ trang sẽ có 2^20 mục

  → **Làm sao lưu trữ bảng ánh xạ trang của mọi qt ?**
- **Giải pháp**
  - Lưu trữ page table trong bộ nhớ ảo và phân trang nó
  - Một số hiện thực
    - Bảng ánh xạ trang đa cấp
    - Bảng ánh xạ trang ngoợc

BẢNG ÁNH XẠ TRANG ĐA CẤP

- Chỉ số trang được chia ra thành n chí số nhỏ
- Ví dụ : 386, Pentium dùng n = 2
  - Chỉ số trang được chia làm 2 chỉ số p1 và p2

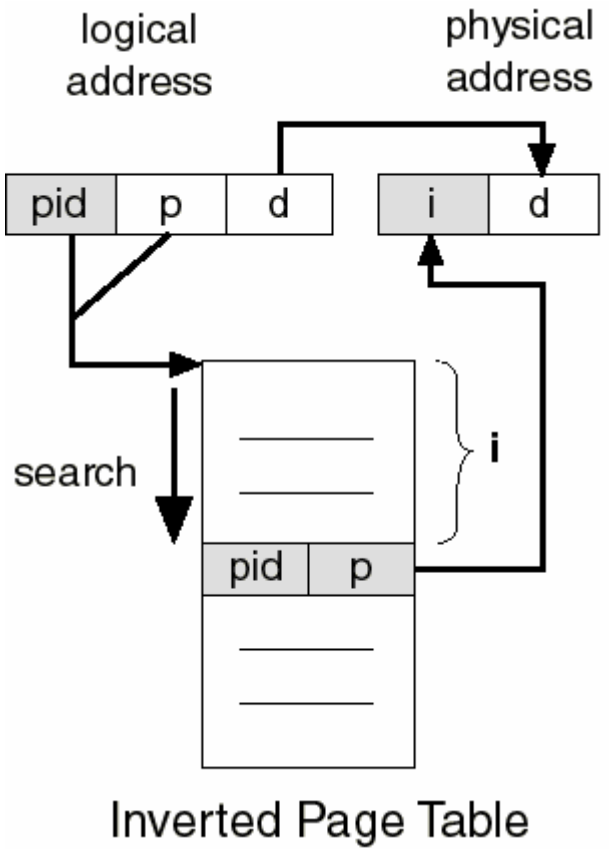


BỘ NHỚ ẢO WINDOWS NT TRÊN Intel x86

- **Phân trang 4KB, địa chỉ ảo 32 bit**
- **Mỗi quá trình có bảng ánh xạ trang 2 cấp**
  - Page directory chứa 1024 mục (PDE), mỗi mục 4B
  - 1 PDE chỉ tới một bảng ánh xạ trang cấp 2. Bảng này có 1024 mục (PTE), mỗi mục 4B
  - Page directory luôn ở trong bộ nhớ thực
  - Các bảng ánh xạ cấp 2 có thể
- **Quá trình khi mới tạo chỉ được cấp một số trang**
- **1 PTE chứa các thông tin**
  - 1 Present bit, 1 dirty bit, các protection bit
  - Các bit chỉ file lưu trữ trang tương ứng
  - Các bít chỉ trạng thái trang : committed, reserved, not used.

BẢNG ÁNH XẠ TRANG NGƯỢC (*Inverted Page Table – IPT*)

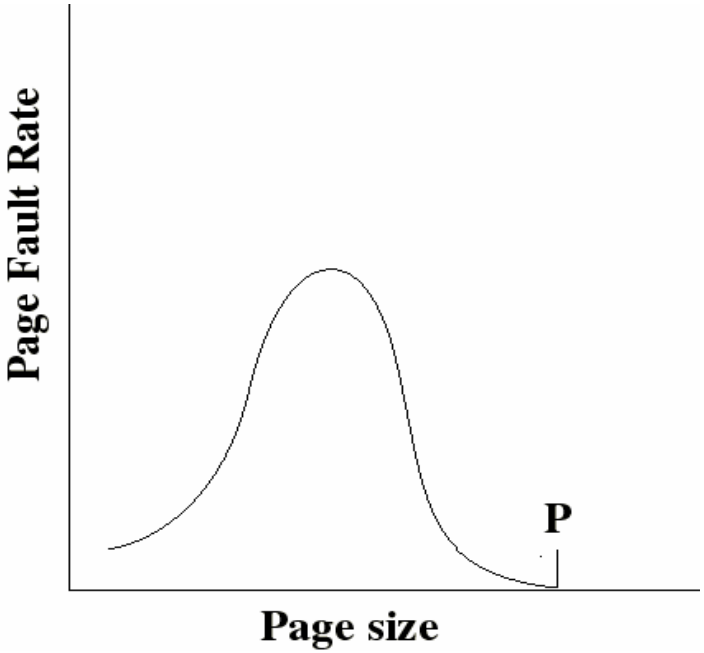
- Dùng trong PowerPC
- Toàn hệ thống có một IPT
- 1 mục của IPT
  - Tương ứng với 1 frame bộ nhớ thực
  - Chứa chỉ số trang ảo được ánh xạ vào frame đó và PID của quá trình tương ứng dùng trang ảo này
- Dùng PID + page# để tìm trong bảng IPT, từ đó suy ra frame#

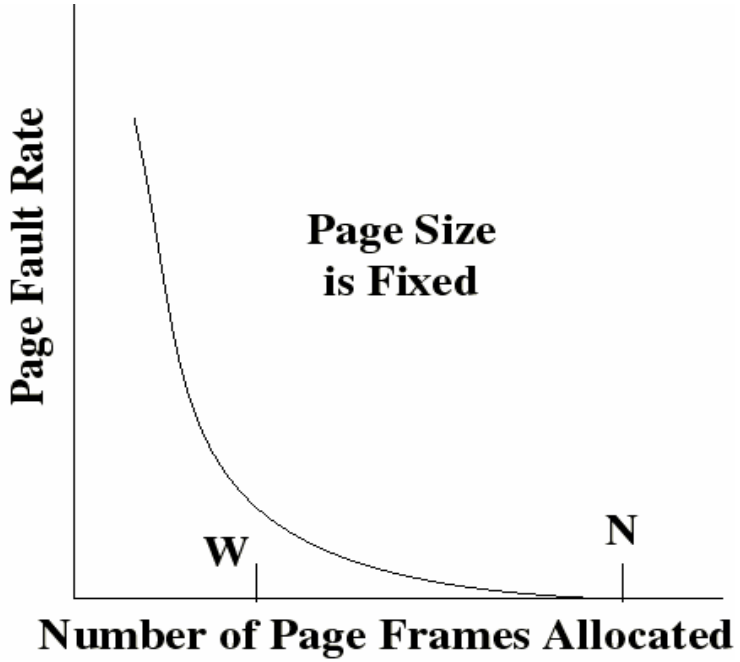


Inverted Page Table

VẤN ĐỀ KÍCH THƯỚC TRANG

- Phụ thuộc phần cứng (size của frame)
- Kích thước trang nên lớn hay nhỏ
- Tỉ lệ ***page fault*** phụ thuộc vào page size và số frame cấp cho quá trình



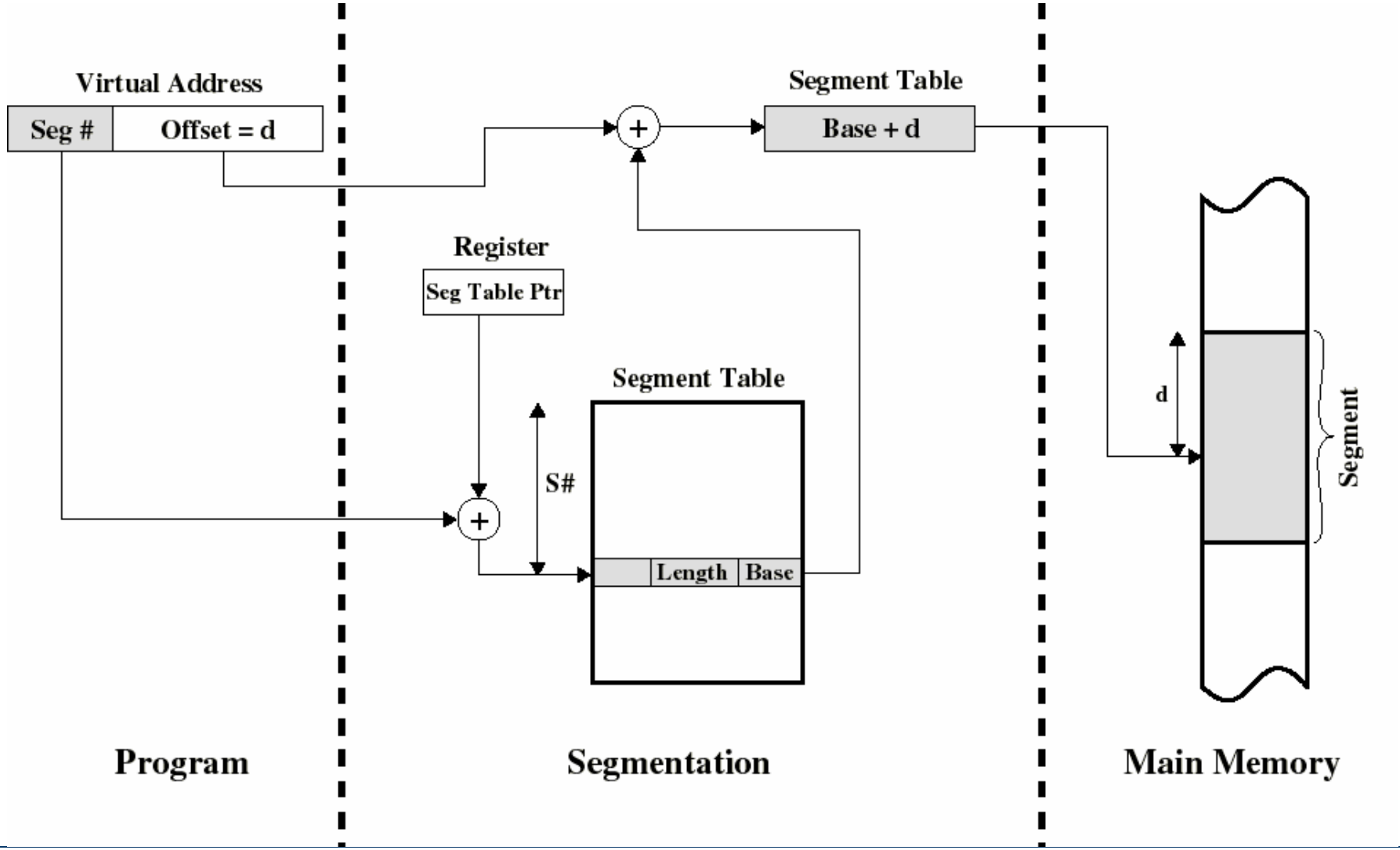


- Kích thước trang thông thường từ 1KB – 4KB

KỸ THUẬT PHÂN ĐOẠN (*SEGMENTATION*

- Các khối bộ nhớ có kích thước khác nhau tùy thuộc yêu cầu của quá trình.
- Địa chỉ ảo:
  - Chỉ số đoạn (*Segment number*)
  - Độ dời của ô nhớ trong đoạn (*Displacement*)
- Ưu điểm:
  - Dễ dàng mở rộng segment, thay đổi và tái biên dịch chương trình mà không cần link hay load lại
  - Cho phép chia sẻ, bảo vệ giữa các process.
- Mỗi quá trình có một bảng ánh xạ đoạn (*segment table*)
- Mỗi mục (*entry*) của bảng ánh xạ đoạn chứa
  - Present bit
  - Secondary storage address
  - Chỉ số segment, chiều dài segment
  - Modified bit
  - Các bít điều khiển khác

ÁNH XẠ ĐỊA CHỈ TRONG HỆ THỐNG PHÂN ĐOẠN



CƠ CHẾ BẢO VỆ & CHIA XẺ BỘ NHỚ TRONG HỆ THỐNG PHÂN TRANG

- Quyền: Read, Write, Execute
- Mức độ bảo vệ:

Mode	Read	Write	Exec
Security	N	N	N
Copy prevention	N	N	Y
Data protection	Y	N	N
Data protection	Y	N	Y
Run prevention	Y	Y	N
Full right	Y	Y	Y

- Có thể chia xẻ đoạn như chia xẻ trang

PHỐI HỢP PHÂN TRANG & PHÂN ĐOẠN

- **Địa chỉ ảo V=(s, p, d)**
  - s: chỉ số đoạn (segment #)
  - p : chỉ số trang trong đoạn (page #)
  - d : độ dời của ô nhớ trong trang (displacement)
- **Địa chỉ thực R=(p', d')**
  - p' : chỉ số trang thực (frame #)
  - d' : độ dời của ô nhớ trong trang thực
- **Ánh xạ địa chỉ**

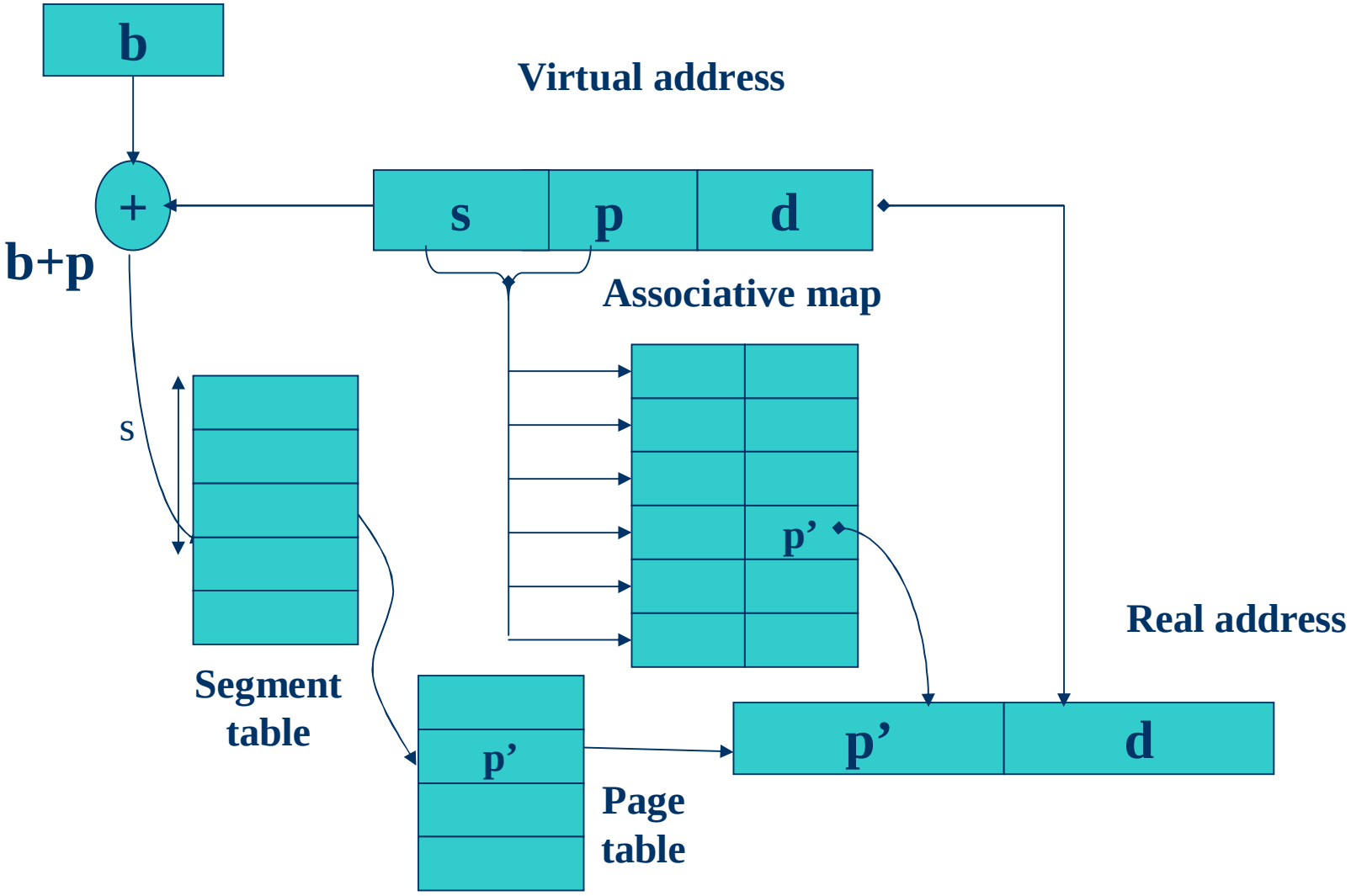
  (s, p) → Associate memory → p'

  Hoặc

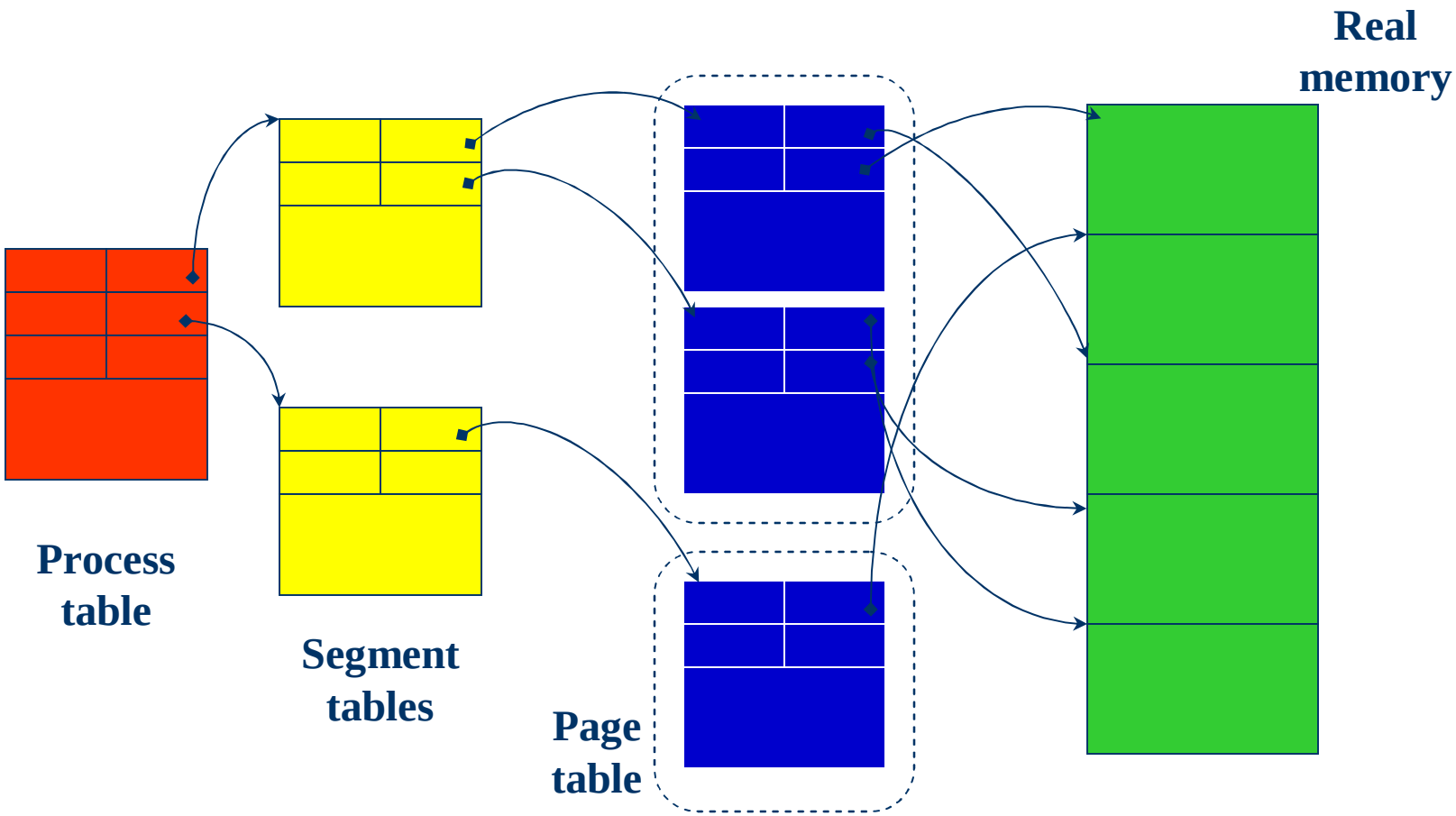
  s→s' (s', p) → p'

  (s' : địa chỉ đầu bảng ánh xạ trang với mỗi đoạn)

ÁNH XẠ ĐỊA CHỈ TRONG HỆ THỐNG PHÂN ĐOẠN KẾT HỢP PHÂN TRANG



CẤU TRÚC ÁNH XẠ BỘ NHỚ (trong hệ thống phân đoạn kết hợp phân trang)



BÀI TẬP

1. Hệthống VM phân trang có **page size 1KB**. Chỉ số **trang từ 0..49**. **Bộ nhớ thực 5KB**. Cho bảng ánh xạ trang sau, tính địa chỉ vật lý của các ô nhớ có địa chỉ ảo là **19371, 22230, 22955**

Virtual page #	…	17	18	19	20	21	22	23	..
Present bit	…	1	1	0	1	1	0	1	..
Page frame #	…	1	3	#	4	2	#	0	…

2. Cho biết tầm địa chỉ một quá trình có thể truy cập.
3. Địa chỉ ảo **32784** của quá trình có bảng ánh xạ trang trên có tương ứng với địa chỉ ô nhớ thực nào không ?

CHƯƠNG 8 : QUẢN LÝ BỘ NHỚ ẢO

- Các chiến lược quản lý bộ nhớ ảo
- Các giải thuật thay thế trang
  - Nguyên tắc tối ưu
  - Các giải thuật: OPT, FIFO, LRU, LFU, NUR, dịp may thứ hai
- Tính cục bộ (*locality*)
- Lý thuyết về tập làm việc (*working set*)
- Bài tập

CÁC CHIẾN LƯỢC QUẢN LÝ BỘ NHỚ ẢO

- Các chiến lược quản lý
  - Chiến lược nạp (*Fetch strategies*)
  - Chiến lược sắp đặt (*Placement strategies*)
  - Chiến lược thay thế(*Replacement strategies*)
- Chiến lược nạp
  - Nạp trang theo yêu cầu (*Demand paging*)
  - Nạp trang tiên đoán (*Anticipatory paging*)
  - Page fault và các bước xử lý page fault
- Chiến lược sắp đặt
- Chiến lược thay thế

CÁC GIẢI THUẬT THAY THẾ TRANG

- **Yêu cầu : Tối thiểu số page fault**
- **Nguyên tắc tối ưu : Chọn trang thay thế là**
  1. Trang không còn dùng nữa
  2. Trang sẽ không dùng lại trong thời gian xa nhất
- **Các tiêu chuẩn (thực tế) để chọn trang thay thế**
  - **Các trang không bị thay đổi**
  - **Các trang không bị khóa**
  - **Các trang không thuộc quá trình nhiều page fault**
  - **Các trang không thuộc tập làm việc của quá trình**
- **Một số giải thuật thay thế trang**
  - **Thay thế trang ngẫu nhiên**
  - **FIFO, LRU, giải thuật xấp xỉ LRU, LFU, NUR**

GIẢI THUẬT TỐI ƯU (OPT)

- **Chọn trang thay thế là trang sẽ không được tham khảo trong thời gian lâu nhất**

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

Thời điểm t												
	0	1	2	3	4	5	6	7	8	9	10	11

Bộ nhớ thực có 3 frame												
	1	**1**	**1**	**1**	**1**	**1**	**1**	**1**	**1**	**1**	**1**	**1**
		2	**2**	**2**	**2**	**2**	**2**	**2**	**2**	**3**	**4**	**4**
			3	**4**	**4**	**4**	**5**	**5**	**5**	**5**	**5**	**5**

7 page fault

- **Nhận xét?**

GIẢI THUẬT FIFO

- **Chọn trang thay thế là trang ở trong bộ nhớ thực trong khoảng thời gian lâu nhất**
- **Nghịch lý Belady**

	1	2	3	4	1	2	5	1	2	3	4	5	
Bộ nhớ thực có 3 frame	1	1	1	4	4	4	5	5	5	5	5	5	**9 page fault**
		2	2	2	1	1	1	1	1	3	3	3	
			3	3	3	2	2	2	2	2	4	4	
Bộ nhớ thực có 4 frame	1	1	1	1	1	1	5	5	5	5	4	4	**10á page fault**
		2	2	2	2	2	2	1	1	1	1	5	
			3	3	3	3	3	3	2	2	2	2	
				4	4	4	4	4	4	3	3	3	

GIẢI THUẬT LRU (*Least Recently Used*)

- **Chọn trang thay thế là trang đã không được tham khảo trong thời gian lâu nhất**

Thời điểm t	0	1	2	3	4	5	6	7	8	9	10	11
Chuỗi tham khảo	1	2	3	4	1	2	5	1	2	3	4	5
Bộ nhớ thực có 3 frame	**1**	1	1	**4**	4	4	**5**	5	5	**3**	3	**5**
		2	2	2	**1**	1	1	1	1	1	**4**	4
			3	3	3	**2**	2	2	2	2	2	2

- **Nhận xét?**
- **So sánh với FIFO**

GIẢI THUẬT NUR (Not Used Recently)

- **Là giải thuật xấp xỉ LRU**
- **Dùng thêm 2 bit cho mỗi trang**
  - **Referenced bit R**
  - **Modified bit M (còn gọi là dirty bit)**
- **Trang sẽ thuộc 1 trong 4 nhóm, thay thế trang sẽ theo độ ưu tiên của nhóm trang**

R	M	Ý nghĩa đối với trang nhớ
0	0	Chưa tham chiếu, chưa sửa đổi
0	1	Chưa tham chiếu, đã sửa đổi ?
1	0	Đã tham chiếu, chưa sửa đổi
1	1	Đã tham chiếu, đã sửa đổi

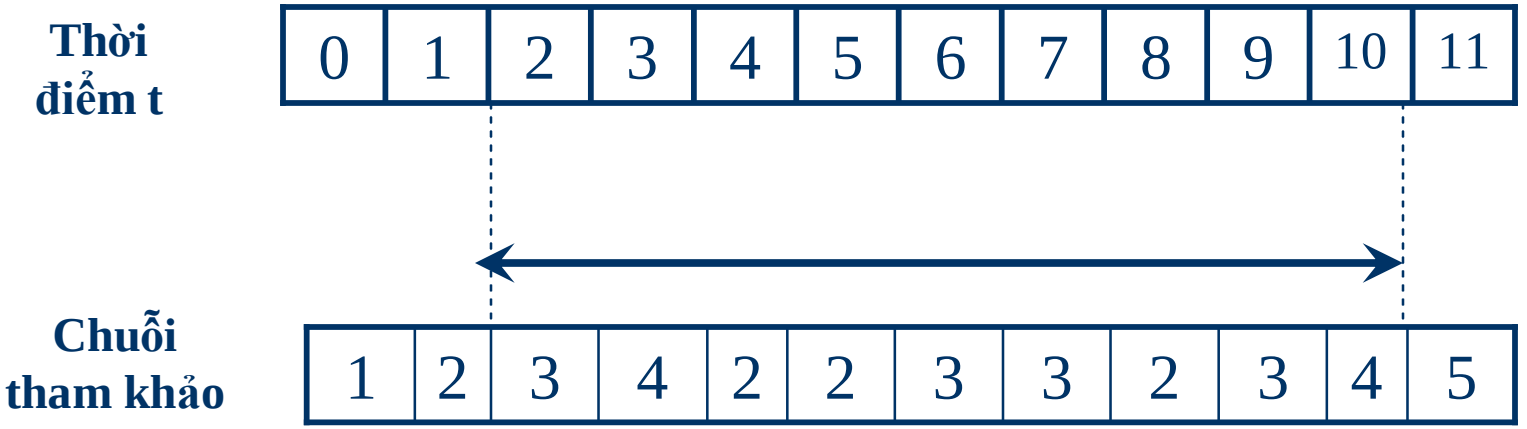
Thứ tự ưu tiên thay thế trang giảm dần

DỊP MAY THỨ HAI (Second Chance)

- Là giải thuật xấp xỉ LRU
- Còn gọi là giải thuật FIFO cải tiến
- Mỗi trang có 1 bit tham chiếu R, lúc đầu là 0
- Trang được chọn xét thay thế theo kiểu FIFO.
  - Trang có R=0 sẽ được thay thế ngay
  - Trang có R=1 được đưa vào cuối hàng và đặt lại R=0. Hệ thống chọn lựa các trang còn lại trong hàng đợi.
- Nhận xét?

GIẢI THUẬT LFU (*Least Frequently Used*)

- **Là giải thuật xấp xỉ LRU**
- **Chọn trang thay thế là trang có tần số được tham khảo là nhỏ nhất trong 1 khoảng thời gian nhất định**



- **Tại t=11, nếu trong bộ nhớ còn 3 trang 2, 3, 4 ta sẽ chọn trang 4 để thay thế**
- **Nhận xét?**

LÝ THUYẾT VỀ TÍNH CỤC BỘ (*Locality*)

- **Tính cục bộ về thời gian (*temporal locality*)**
  - **Các sự việc xảy ra ở thời điểm t rất có thể đang xảy ra ở các thời điểm lân cận (t + dt, t – dt)**
  - **Ví dụ :** *một vùng nhớ đang được tham khảo có thể sẽ được tham khảo đến trong tương lai gần*
- **Tính cục bộ về không gian(*spatial locality*)**
  - **Biến cố xảy ra ở một vùng rất có thể đang xảy ra ở các vùng lân cận**
  - **Ví dụ :** *những vùng nhớ đang được tham khảo gần đây thường kề nhau*
- **Ý nghĩa**
  - **Trong lập trình**
  - **Trong OS : giải thuật thay thế trang`**

KỸ THUẬT ĐỆM TRANG (*Page Buffering*)

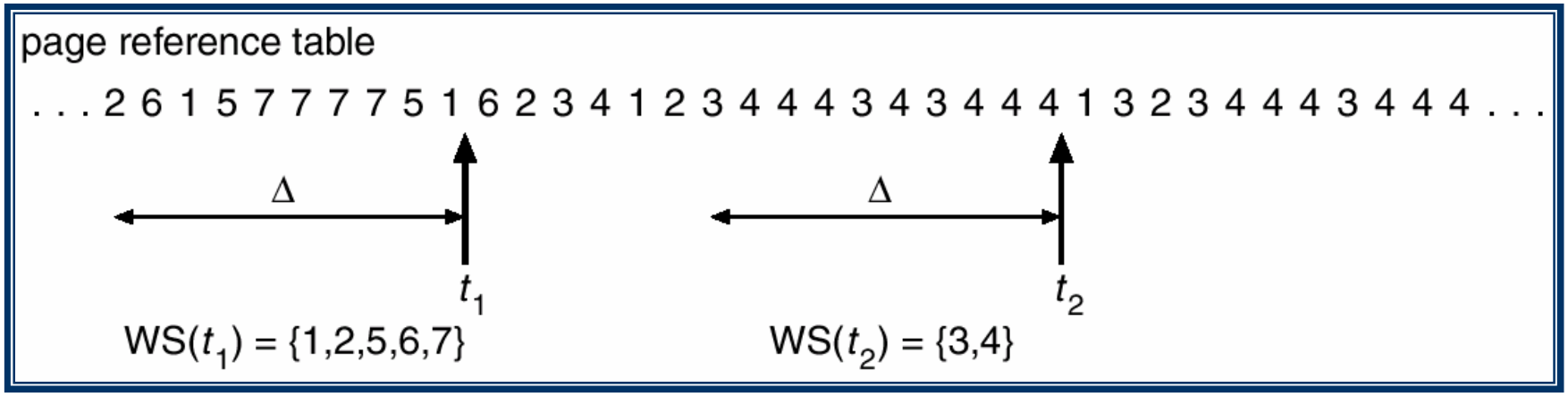
- Tạm thời giữ lại các trang được chọn để thay thể ➔ tránh tác động của giải thuậtt thay thế trang kém hiệu quả. Sử dụng 2 danh sách
  - Free page list
  - Modified page list
- Khi có page fault, hệ thống tìm xem trang cần nạp có còn trong bộ nhớ không trước khi nạp trang.
  - Trang nạp sẽ nạp vào đầu free page list
  - Modified page list dùng để ghi các trang ra theo từng cụm nhiều trang ➔ giảm chi phí I/O

CÁC VẤN ĐỀ KHÁC

- **Tầm vực thay thế trang (*resident scope*)**
  - Tầm vực cục bộ : chỉ chọn trang thay thế trong nhứng trang của quá trình liên quan
  - Tầm vực toàn cục: chọn bất kỳ trang nào không bị lock để thay thế
- **Số frame cấp cho quá trình(*resident set size*)**
  - Không đổi (*fixed allocation*): chia đều/ theo tỉ lệ kích thược quá trình
  - Thay đổi trong quá trình chạy (*variable allocation*)
- **Điều khiển tải (*Load control*)**
  - Số quá trình cần nạp vào bộ nhớ ?

LÝ THUYẾT VỀ TẬP LÀM VIỆC

- **Tập làm việc (working set-WS) = tập những trang quá trình cần sử dụng để làm việc trong thời gian Δ (hình vẽ)**



- **Lý tưởng: WS của quá trình nằm hoàn toàn trong bộ nhớ chính**
- **Theo dõi working set của các quá trình ntn?**

BÀI TẬP

1. Tìm số page fault tương ứng khi sử dụng **OPT**, **FIFO**, **LRU** để thay thế trang với chuỗi tham khảo **2, 3 ,2, 1, 5, 2, 4, 5, 3, 2, 5, 2 & sồ frame=3**.
2. Tìm thời gian truy cập trung bình trong hệ VM có các thông số về thời gian phục vụ như sau:

Thư tự truy cập bộ nhớ ↓

Bộ nhớ	Hit rate	Thời gian phục vụ
CPU cache	90%	1ns
Main memory	75%	1us
Page fault	100%	10ms

CHƯƠNG 9: GIAO TIẾP VỚI HỆ THỐNG FILE

- Tổng quan về dữ liệu và file
- Các thuộc tính & thao tác trên file
- Các phương pháp truy cập file
- Tổ chức thư mục
- Mount hệ thống file
- Bảo vệ hệ thống file
- Sao lưu và phục hồi dữ liệu

TỔNG QUAN VỀ DỮ LIỆU & FILE

- **Yêu cầu lưu trữ của user**
  - Lưu trữ lâu dài
  - Truy cập nhanh
  - Lưu đực nhiêu dữ liệu
  - Chia xẻ và bảo vệ tốt
  - Dễ sử dụng

➔ **cần sự hỗ trợ của phần cứng và OS**

- **Khái niệm file (tập tin, tệp)**
  - Đơn vị lưu trữ luận lý của OS
  - Phân loại: chương trình hoặc dữ liệu
  - Có thể có/ không có cấu trúc:

CÁC THUỘC TÍNH & THAO TÁC TRÊN FILE

- **Thuộc tính file (*file attribute*)**
  - Tên, kiểu, vị trí lưu trũ , kích cỡ, thông tin bảo vệ…
- **Thao tác về dữ liệu trên file (*data operation*)**
  - create, write, read, seek, delete, truncate
  - open(F_i)
  - close (F_i)
- **Thao tác về đặt tên file (*naming operation*)**
  - Tạo hard link, soft link, rename,
  - Thiết lập thuộc tính, lấy thuộc tính

CẤU TRÚC DỮ LIỆU QUẢN LÝ FILE

- **Bảng thông tin về các file đang mở (*Open File Table*).**
  - Dành cho n quá trình dùng chung một file
  - Chứa: biến điếm sử dụng, thuộc tính file, vị trí file trên đĩa, con trỏ đến vị trí của file trong bộ nhớ.
- **Bảng thông tin về các file của từng quá trình**

(*Per-process File Table*): Với mỗi file, bảng này chứa:

- Con trỏ đến mục tương ứng trong Open File Table
- Vị trí hiện tại trong file
- Chế độ truy cập của quá trình với file (r, w, rw)
- Con trỏ tới file buffer

TÁC VỤ FILE (1)

- **Tạo file:** Create(name)
  - Cấp không gian lưu trữ
  - Tạo file descriptor chứa thông tin quản lý file
  - Thêm file descriptor vào thư mục chứa file
- **Xoá file:** Delete(name)
  - Tìm thư mục chứa file
  - Giải phóng các khối đĩa dành cho file
  - Xoá file descriptor khỏi thư mục chứa file
- **Mở file:** file_id = Open(name, mode)
  - Kiểm tra file có mở hay chưa → chia xẻ file.
  - Kiểm tra quyền sử dụng file.
  - Tăng open count của file.
  - Tạo và thêm thông tin quản lý file đang mở vào bảng file của hệ thống và của quá trình.
- **Đóng file:** Close(file_id) ?

TÁC VỤ FILE (2)

- **Đọc file:**
  - Read(file_id, from, size, buf_addr) : đọc ngẫu nhiên
  - Read(file_id, size, buf_addr) : đọc tuần tự
- **Ghi file:**
  - Tương tự đọc file
  - Thực hiện copy dữ liệu từ buffer vào file
- **Seek:**
  - Cập nhật vị trí con trỏ file
- **Ánh xạ file vào bộ nhớ (*memory mapping a file*):**
  - Aùnh xạ 1 vùng địa chỉ ảo vào nội dung file
  - Tác vụ đọc/ ghi lên vùng nhớ ⇔ đọc/ ghi file

CÁC PHƯƠNG PHÁP TRUY CẬP FILE

- **Theo quan điểm người lập trình**
  - **Tuần tự**: xử lý dữ liệu (byte, record…) theo trật tự
  - **Theo khoá**: tìm khối dữ liệu theo giá trị khóa
- **Theo quan điểm hệ điều hành**
  - **Truy cập tuần tự** (*sequential access*): giữ và cập nhật con trỏ đến vị trí truy cập kế tiếp trong file
  - **Truy cập trực tiếp** (*random access):* truy cập dữ liệu theo offset của khôi dữ liệu trong file.

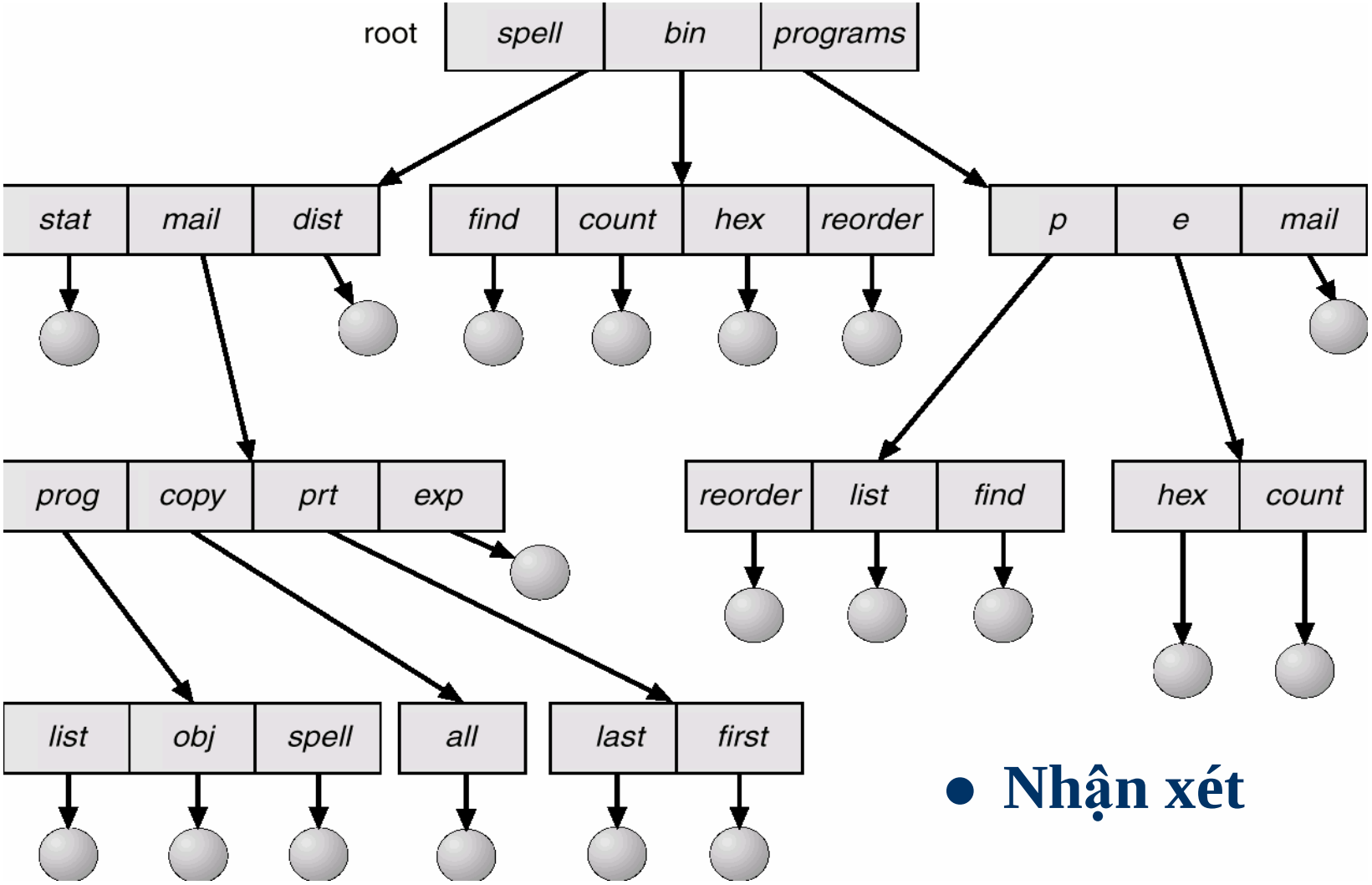
TỔ CHỨC THƯ MỤC

- **Thư mục (*directory*)**
  - Cấu trúc dữ liệu của HĐH để ánh xạ tên sang số nhận dạng file của HĐH
- **Tác vụ thực hiện trên thư mục**
  - Tìm file, tạo file, xoá file, liệt kê nội dung thư mục, đổi tên file, duyệt hệ thống file
- **Yêu cầu khi tổ chức hệ thống thư mục**
  - Hiệu quả
  - Tiện lợi cho người sử dụng
  - Có khả năng nhóm các file theo thuộc tính

CÁCH TỔ CHỨC THƯ MỤC

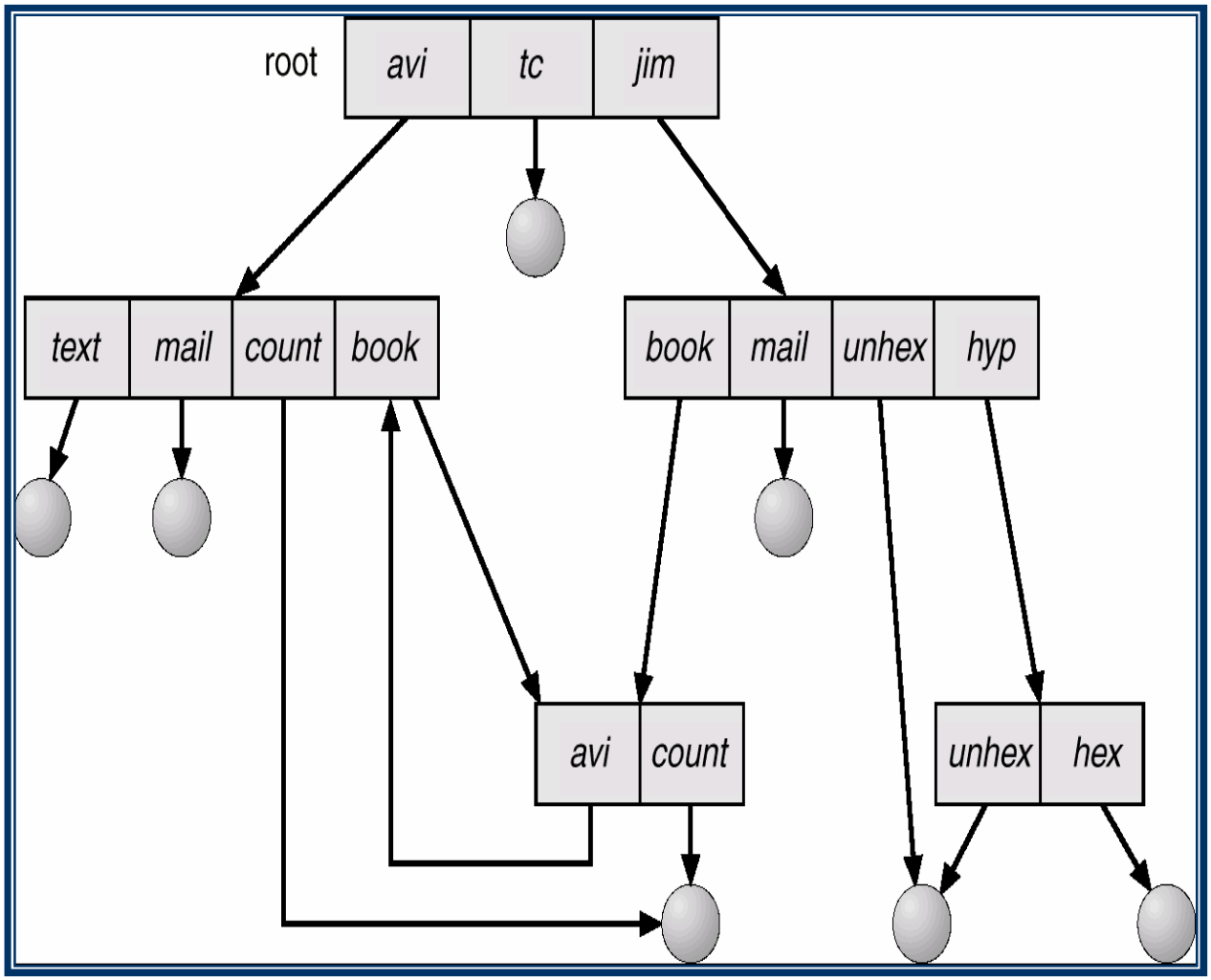
- **Tổ chức 1 cấp** *(Single-Level Directory)*
  - Sử dụng 1 không gian tên (thư mục) duy nhất cho mọi user
  - Việc đặt tên dễ đụng độ
  - Không có khả năng nhóm các file
- **Tổ chức 2 cấp** *(Two-Level Directory)*
  - 1 user có một thư mục riêng
  - Sử dụng đường dẫn để xác định nơi lưu file
  - Tìm kiếm nhanh
  - Vẫn có khả năng đụng độ khi đặt tên
  - Không có khả năng nhóm các file

TỔ CHỨC THƯ MỤC ĐA CẤP
(*Multilevel Directory*)



- **Nhận xét**

TỔ CHỨC THƯ MỤC DẠNG ĐỒ THỊ TỔNG QUÁT (*General Graph*)



- **K/niệm link**
  - **Hard link**
  - **Soft link**
- **Vấn đề?**
- **Giải quyết?**

MOUNT HỆ THỐNG FILE

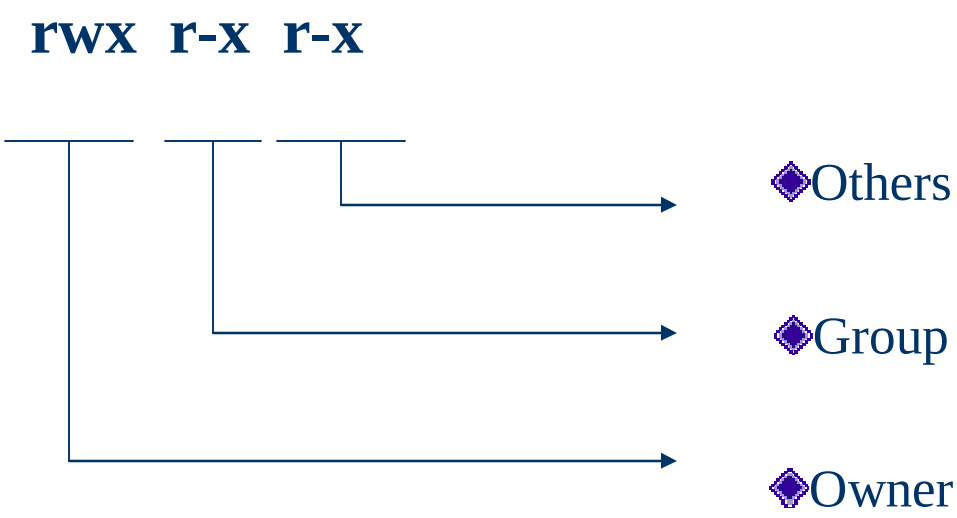
- **Mount**
  - Gắn hệ thống file trên 1 thiết bị lưu trữ vào hệthống thư mục chính để truy cập
- **Mount point**
  - Thư mục nơi gắn hệ thống file ở ngoài vào
- **Unmount**
  - Tách hệ thống file của thiết bị lưu trữ ra khỏi mount point
- **Loại hệ thống file được mount:**
  - tùy thuộc sự hỗ trợ của hệ điều hành

BẢO VỆ HỆ THỐNG FILE

- **Người tạo/ sở hữu file phải điều khiển được**
  - Các thao tác có thể thực hiện trên file
  - Ai có quyền thực hiện các thao tác trên
- **Các quyền thao tác trên file**
  - Read, Write, Execute, Append, Delete, List
- **Phương pháp bảo vệ**
  - Access list & group (Windows NT)
  - Access control bits (UNIX)
- **Điều khiển truy cập đồng thời**
  - Khóa toàn bộ file
  - Khóa từng phần file

BẢO VỆ FILE TRÊN UNIX

- Chế độ truy cập : read, write, execute
- 3 loại người dùng: owner, group, others
- Biểu diến quyền truy cập file bằng tổ hợp bit

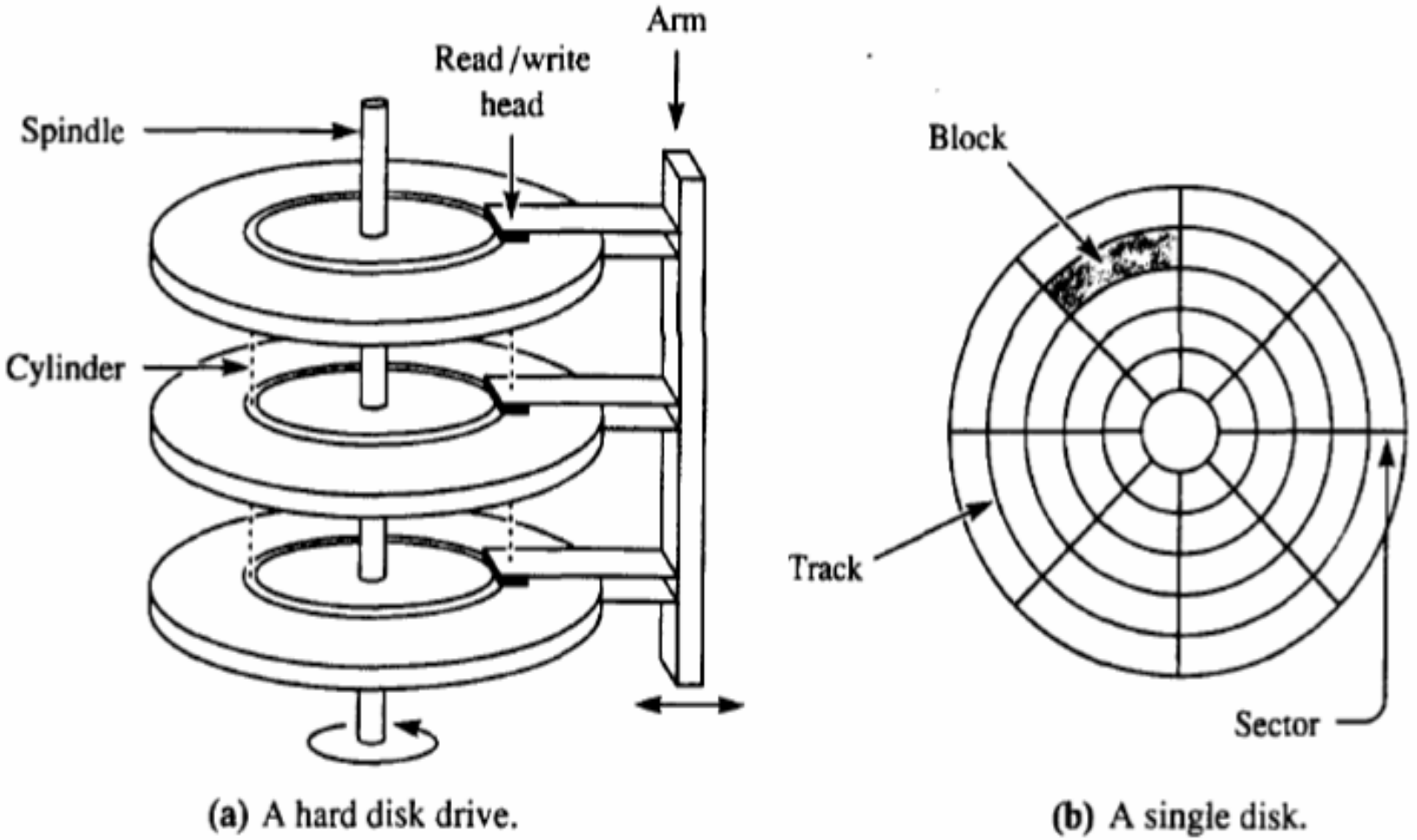


- Kiểm tra quyền sử dụng lần lượt theo owner, group rồ user

CHƯƠNG 10: HIỆN THỰC HỆ THỐNG FILE

- **Cấu trúc đĩa cứng**
- **Cấu trúc hệ thống file**
- **Hiện thực cấu trúc thư mục**
- **Cơ chế cấp phát vùng lưu trữ**
  - Cấp liên tục, theo liên kết, theo chỉ số
  - Hệ thống file của UNIX
- **Quản lý vùng trống**
- **Độ hiệu quả/ hiệu suất hệ thống file**
- **Sao lưu và phục hồi dữ liệu**
- **Bài tập**

CẤU TRÚC ĐĨA CỨNG

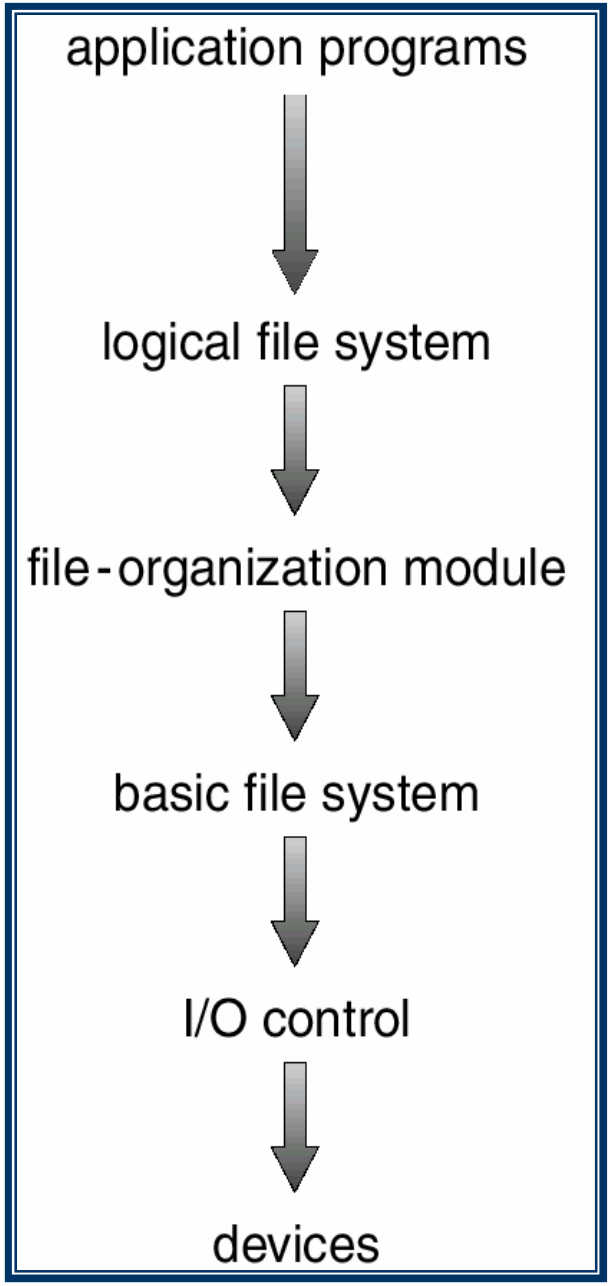


(a) A hard disk drive. (b) A single disk.

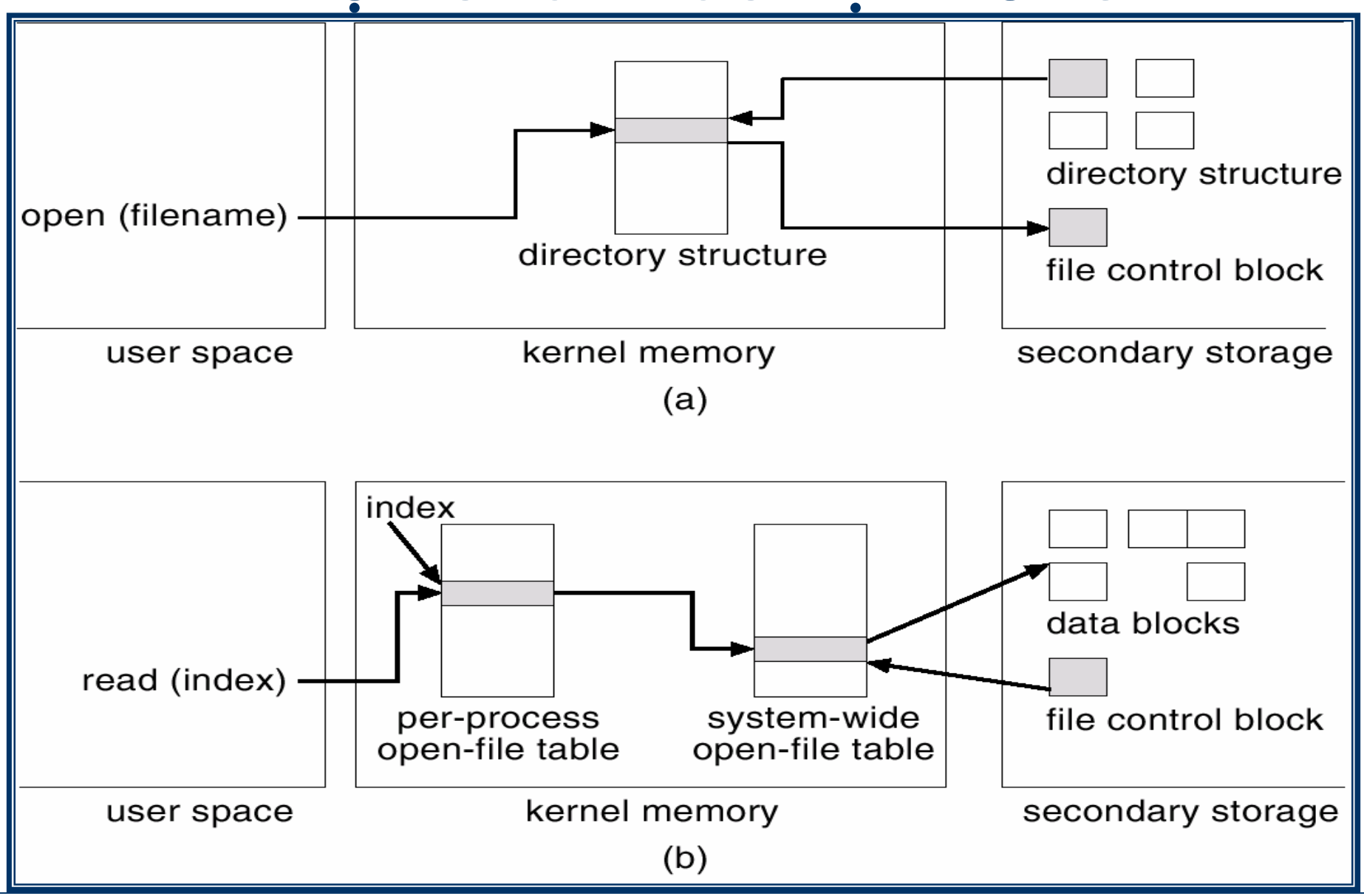
- **Hệ điều hành xem đĩa cứng như một chuỗi các block liên tiếp với kích thước cố định.**

CẤU TRÚC HỆ THỐNG FILE

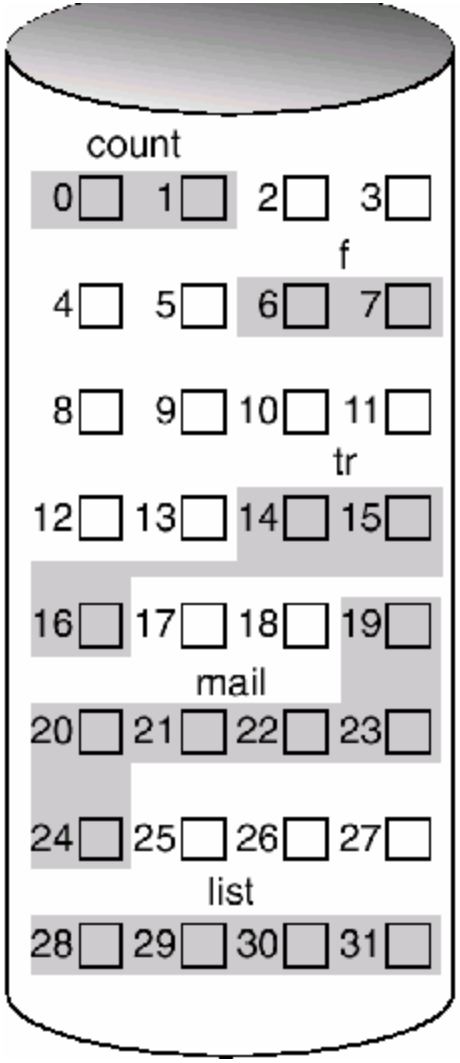
- **Tổ chức theo phân lớp**
- **File Control Block (FCB)**
  - Nằm trên đĩa cứng, chứa
    - Thông tin bảo mật file
    - Thông tin nơi lưu trữ file
- **Virual File System (VFS)**
  - Cung cấp API chung đẻ truy xuất nhiều loại hệ thống file khác nhau
- **Cấu trúc thư mục**
  - Dùng danh sách liên kết
  - Dùng bảng băm



MINH HỌA CẤU TRÚC HỆ THỐNG FILE



CẤP PHÁT VÙNG LƯU TRỮ LIÊN TỤC (*Contiguous Allocation*)`

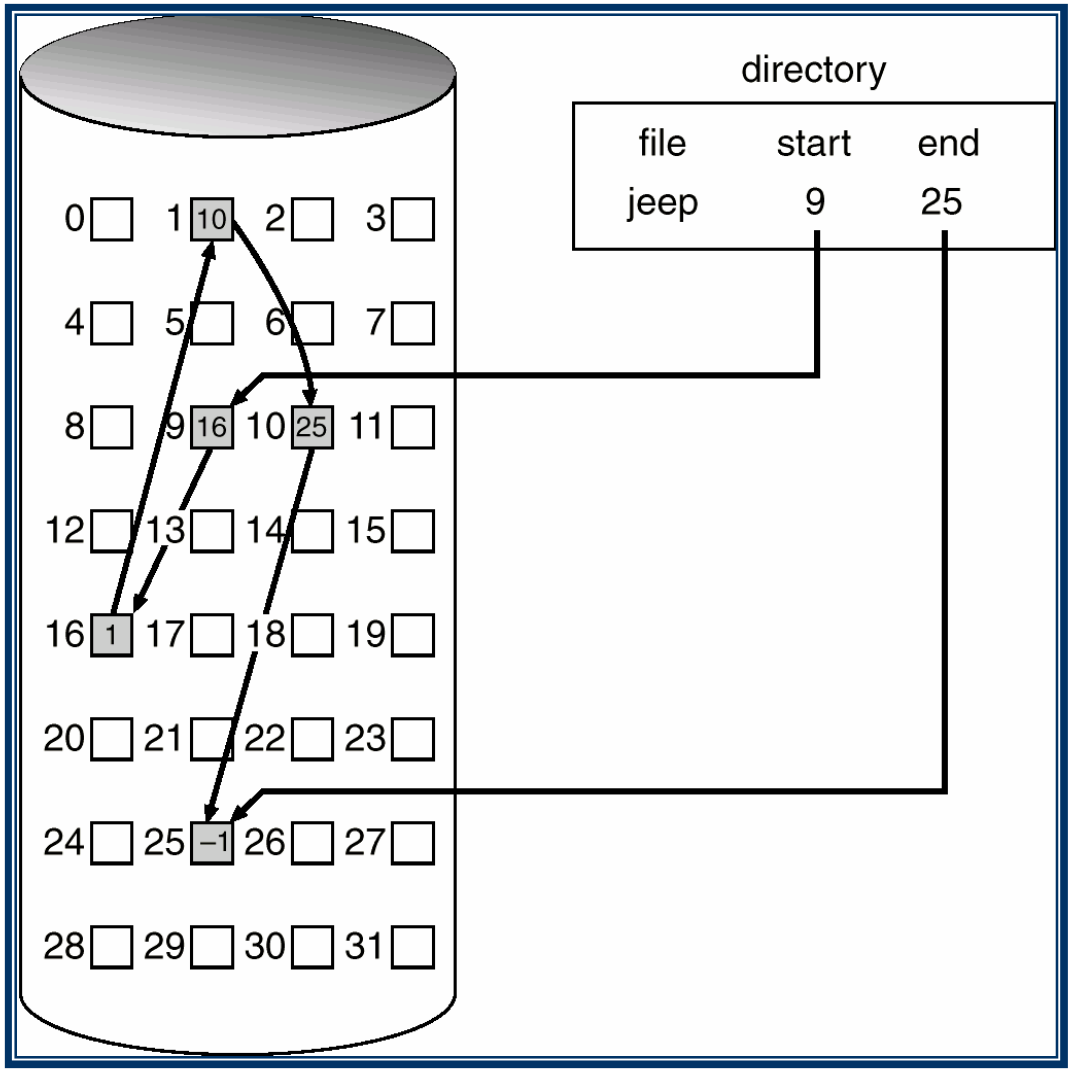


- File gồm n block liên tục
- Thông tin cấp phát:
  - Chỉ số block đầu, số block cấp
- Nhận xét ưu, nhược điểm.
- Khắc phục?

directory

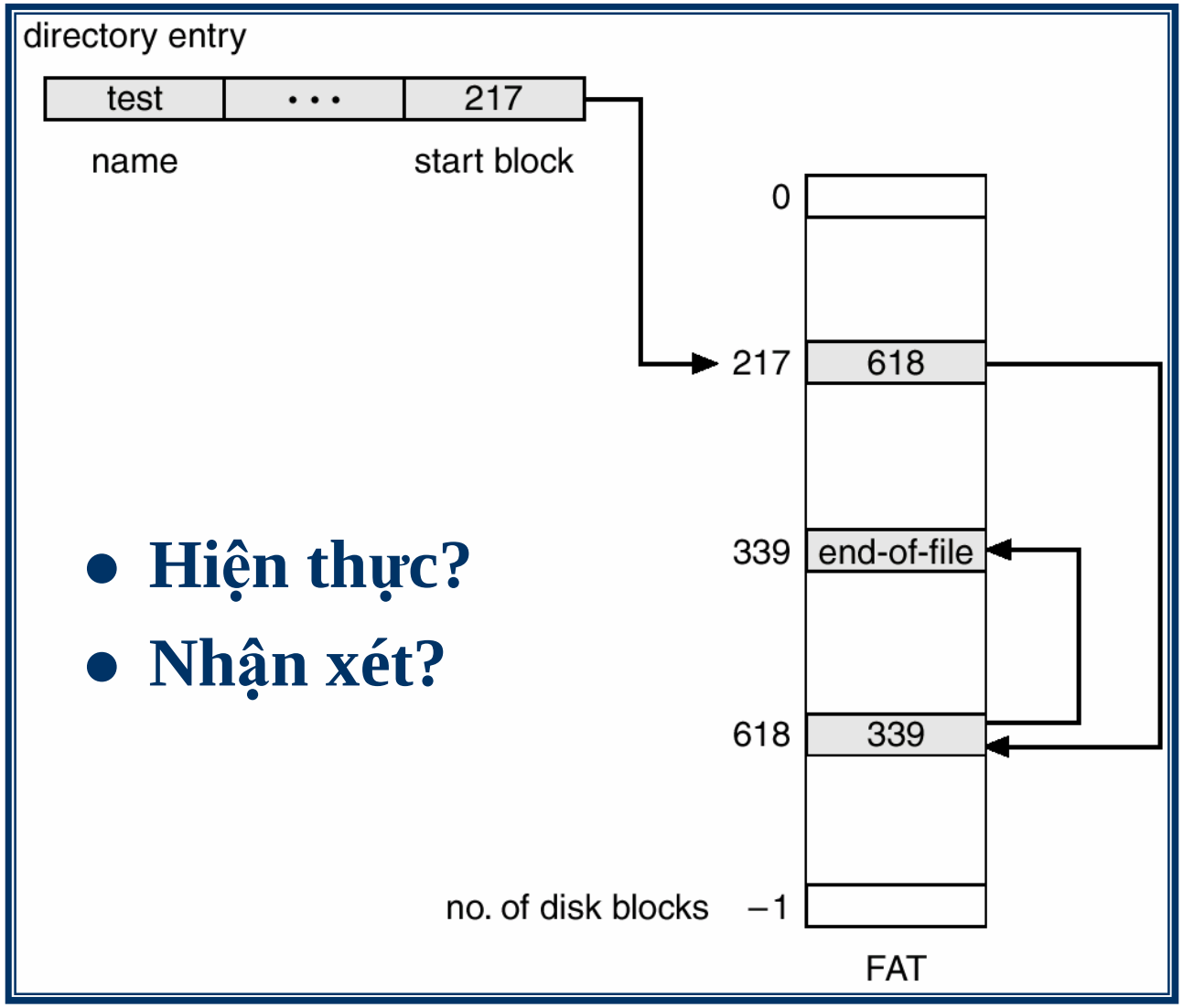
file	start	length
count	0	2
tr	14	3
mail	19	6
list	28	4
f	6	2

CẤP PHÁT VÙNG LƯU TRỮ THEO LIÊN KẾT (*Linked Allocation*)



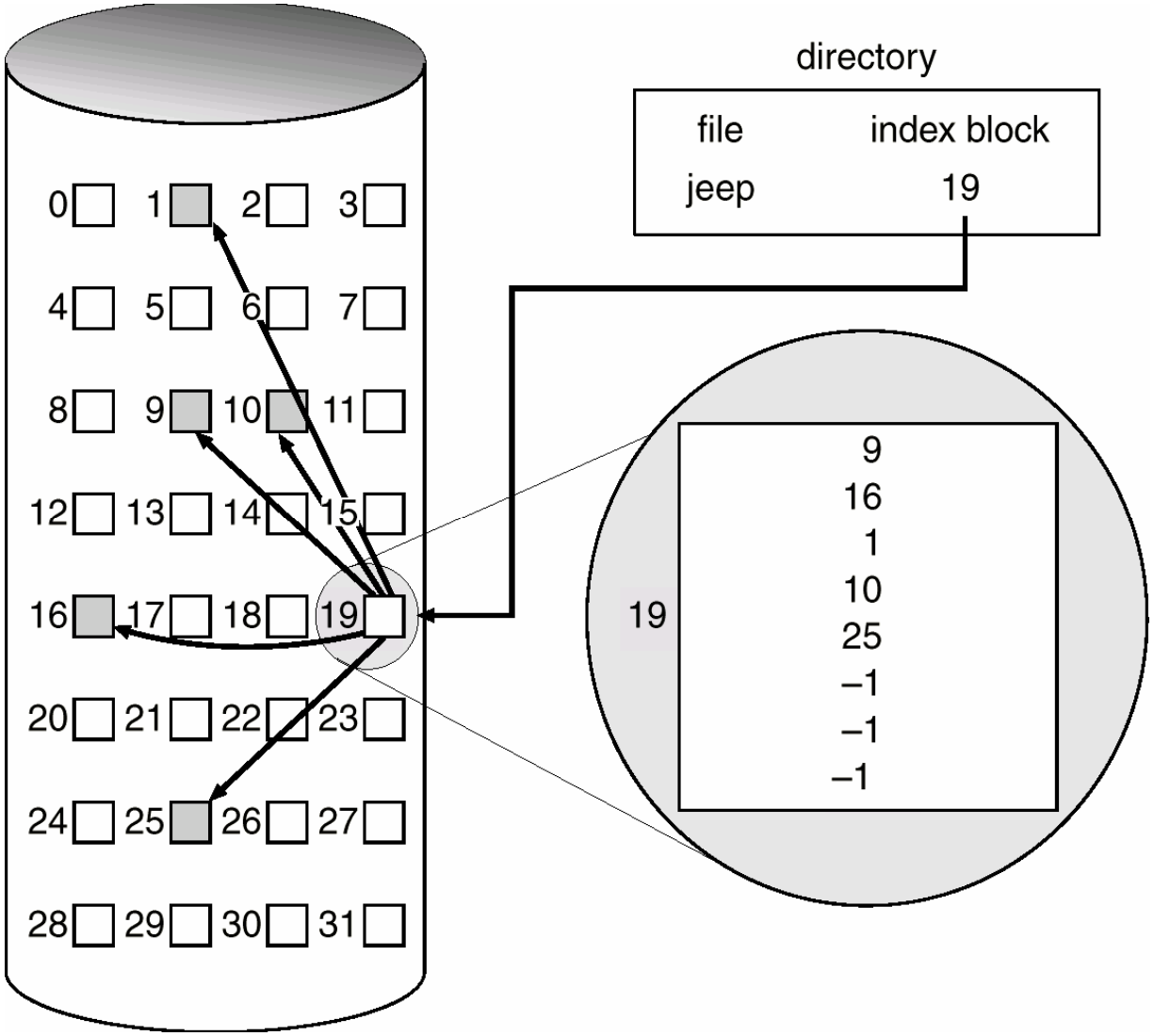
- **File là danh sách liên kết của các block rải rác**
- **Hiện thực?**
- **Nhận xét?**

FILE ALLOCATION TABLE (FAT)



- **Hiện thực?**
- **Nhận xét?**

CẤP PHÁT VÙNG LƯU TRỮ THEO CHỈ SỐ (*Indexed Allocation*)



- Dùng bảng các chỉ số để lưu các con trỏ đến các block dữ liệu của file
- Nhận xét?

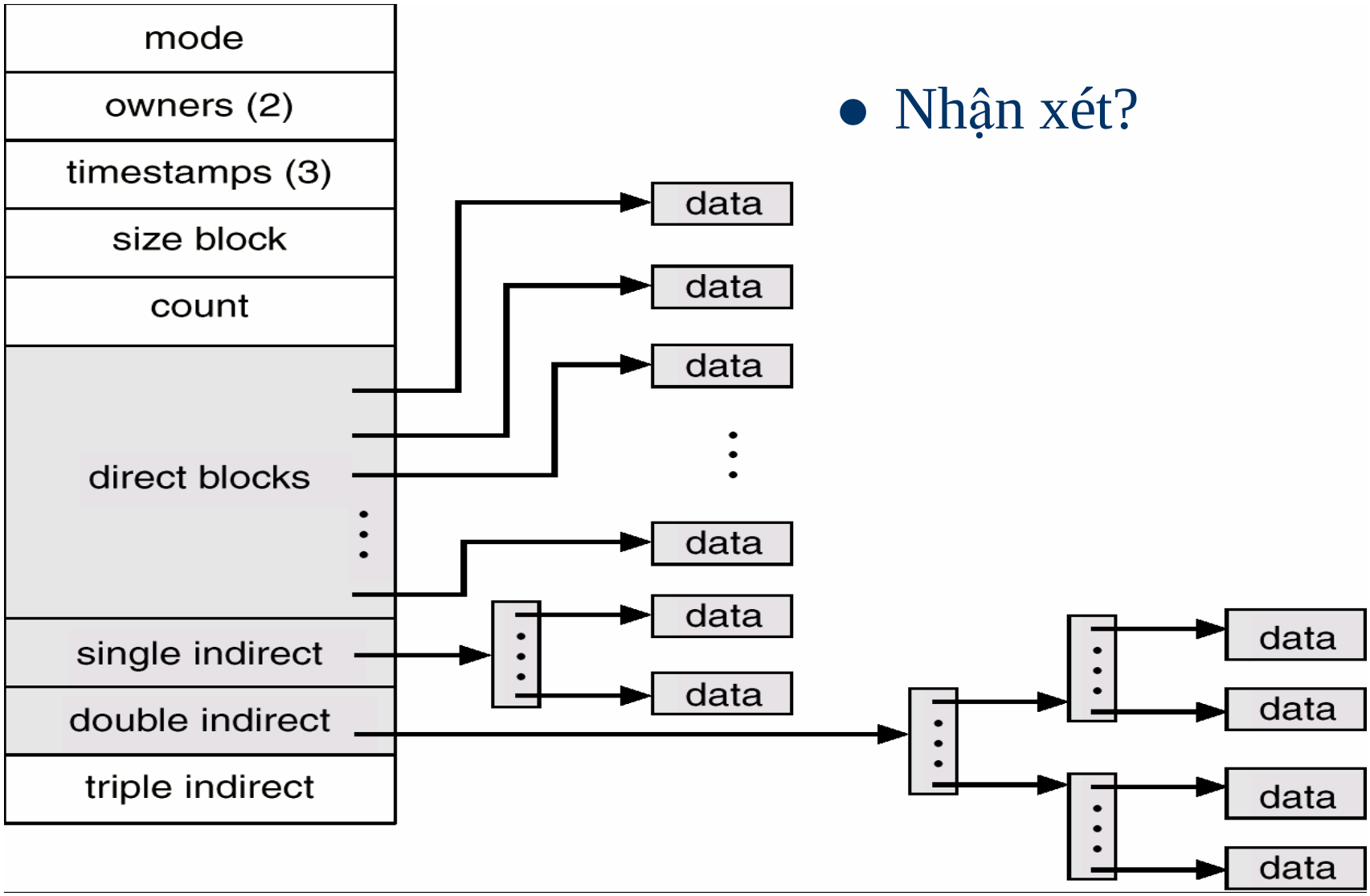
LƯU TRỮ BẢNG CHỈ SỐ CỦA FILE

- Lưu liên tục
  - Bảng chỉ số lưu trong 1 block của đĩa
- Lưu theo kiểu liên kết
  - Bảng chỉ số lưu trong n block của đĩa nối với nhau bằng danh sách liên kết
- Lưu bằng bảng chỉ số đa cấp
  - Dùng bảng chỉ số khác để lưu các con trỏ đến các index block của file
- Sử dụng cơ chế kết hợp

HỆ THỐNG FILE CỦA UNIX

- Đĩa cứng chia thành nhiều block
  - Boot block
  - Super block
  - Các block chứa danh sách các i-node
  - Các block dữ liệu
- Thông tin lưu trong 1 i-node
  - Mode truy cập
  - Owner UID
  - Số link trỏ tới file
  - Thông tin về thời điểm truy cập , tạo file…
  - Kích thước file
  - Dãy các địa chỉ khối chứa dữ liệu
  - …

CẤU TRÚC I-NODE CỦA BSD UNIX



- Nhận xét?

I-NODE CỦA THƯ MỤC

- Thư mục /

Chỉ số i-node	Tên file / thư mục con
2	.
2	..
5	etc
10	home
12	usr

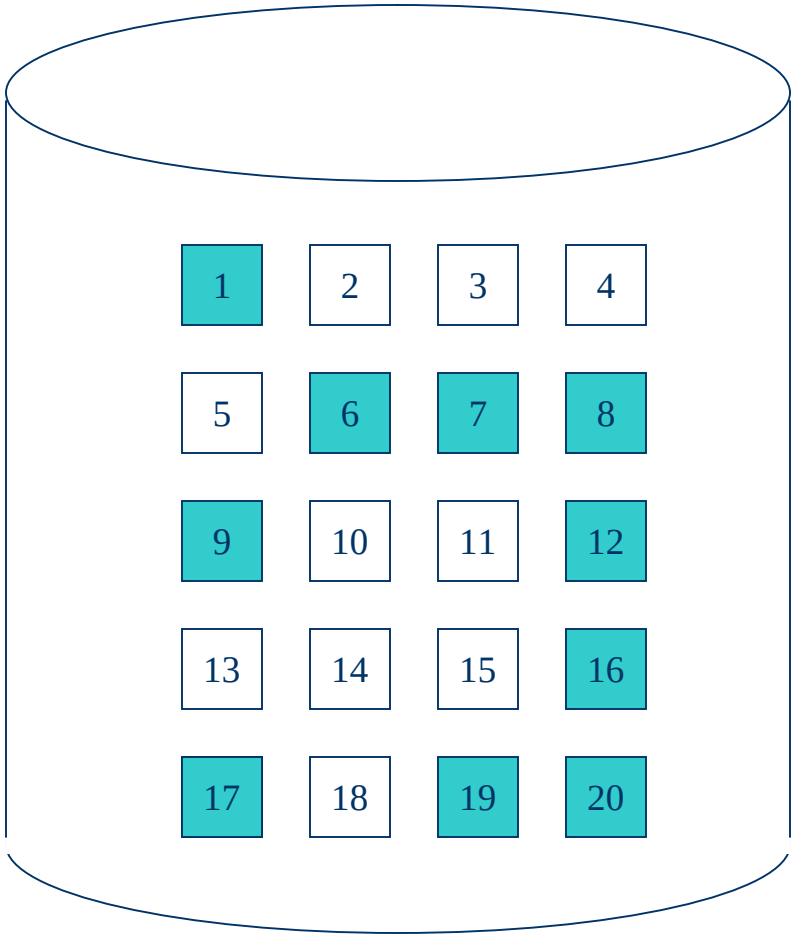
- Thư mục /home

Chỉ số i-node	Tên file / thư mục con
10	.
2	..
15	hung
20	Os01
21	Os02

QUẢN LÝ VÙNG TRỐNG (1/2)

- **Dùng bit vector: N bit quản lý N block data**
  - Bít =0 : block đã cấp
  - Bit=1: block còn trống
- **Dùng danh sách liên kết các block trống**
- **Nhóm các block trống (Grouping)**
  - Chứa địa chỉ N block trong 1 block trống đầu tiên
  - N-1 địa chỉ đầu trỏ đến các block trống thực sự
  - Địa chỉ cuối trỏ đến block chứa N địa chỉ block trống khác
- **Đếm khoảng trống (Counting)**
  - Mỗi block trống lưu trữ số khoảng trống liên tục tiếp theo nó & địa chỉ block trống không kê tiếp.

QUẢN LÝ VÙNG TRỐNG (2/2)



18 Vùng trống

17 Vùng đã cấp phát

- **P/p grouping**
  - Block 2: 3,4, 5,10
  - Block 10: 11, 13,14,15
  - Block 15: 18,-,-,-
- **P/p counting**
  - Block 2: 3, 5
  - Block 5: 0,10
  - Block 10: 1, 13
  - Block 13: 2, 18`

ĐỘ HIỆU QUẢ/ HIỆU SUẤTÛ CỦA HỆ THỐNG FILE

- **Độ hiệu quả hệ thống file phụ thuộc**
  - Cách cấp phát đĩa, các giải thuật trên thư mục
  - Loại dữ liệu trong mục của bảng thư mục
- **Tăng hiệu suất hệ thống file**
  - Disk cache
  - Page cache
  - Free-behind & read-ahead
  - Virtual Disk/ RAM disk
  - Parallel I/O

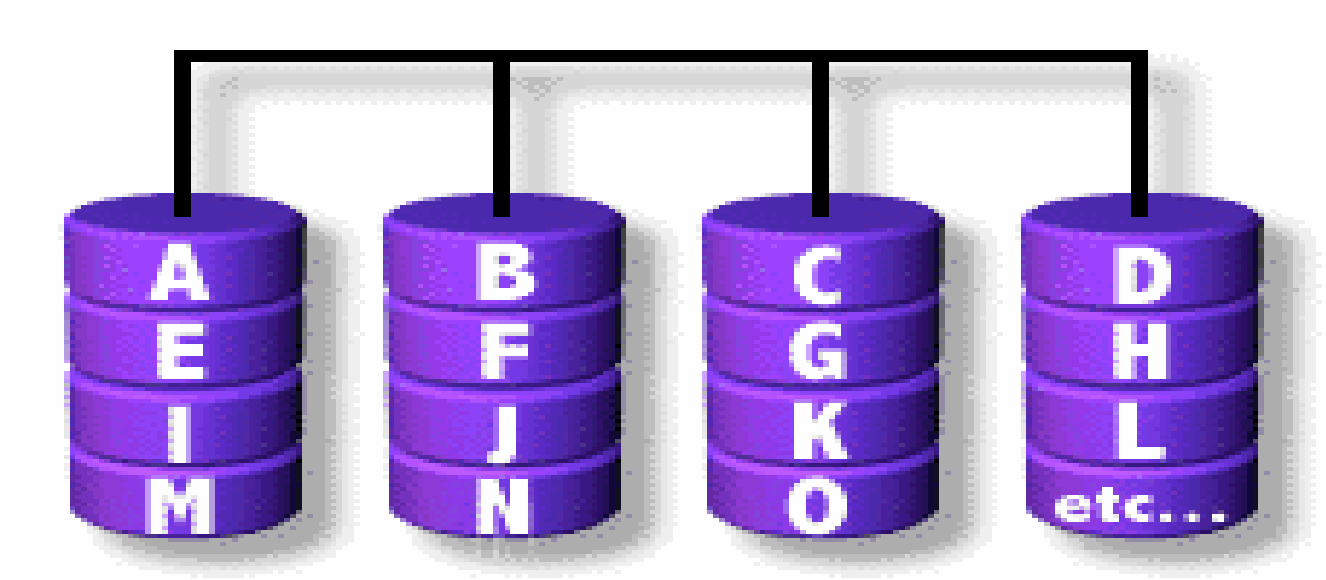
SAO LƯU VÀ PHỤC HỒI DỮ LIỆU

- **Kiểm tra sự nhất quán của dữ liệu**
  - So sánh thông tin trên block đĩa và trong thư mục
  - Sử dụng các tiện ích: ndd, fsck, scandisk,…
- **Sao lưu** (*backup*) **dữ liệu sang thiết bị lưu trữ khác**
  - Sao lưu toàn phần *(normal backup)*
  - Sao lưu tăng dần *(incremental backup)*
- **Phục hồi** (*restore*) **dữ liệu từ thiết bị sao lưu**
  - Khi có hỏng hóc hệ thống
  - Khi cần phục hồi hệ thống về trạng thái cũ
- **Hệ thống file có ghi log** *(Log Structured File System)*

RAID (*Redundant Array of Inexpensive Disks*)

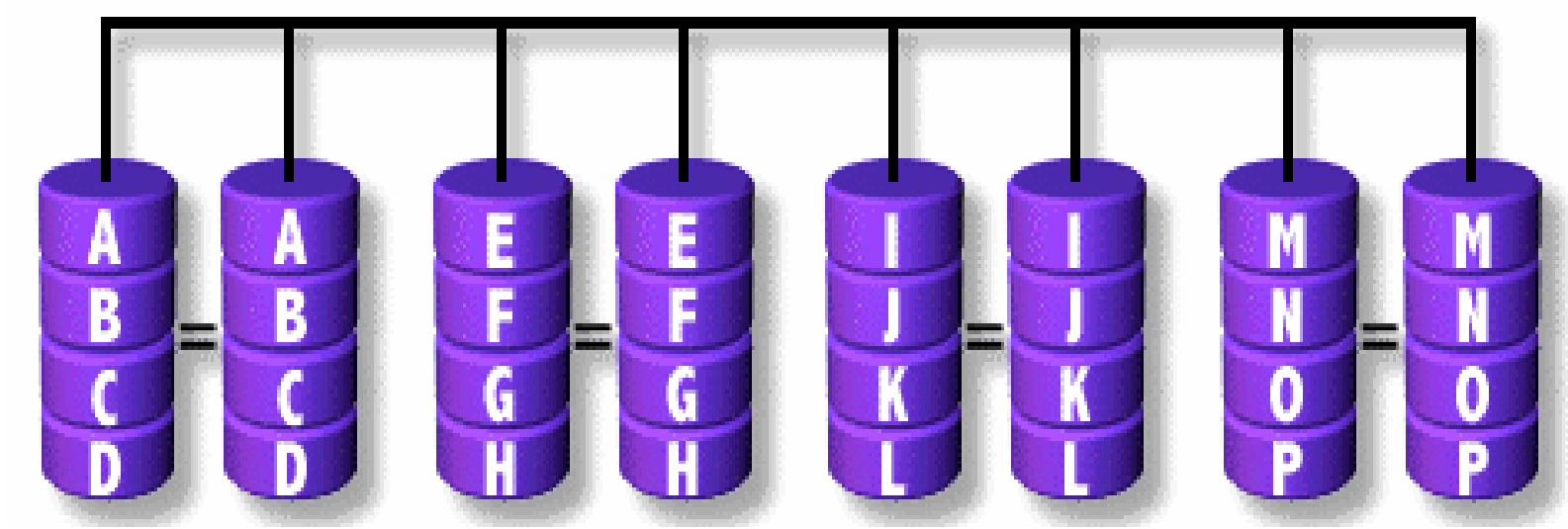
- Tập hợp các đĩa cứng được hệ điều hành xem như một thiết bị lưu trũ luận
- Dữ liệu được phân bố trên tất cả các đĩa
- Các mục tiêu chính
  - Tăng dung lượng lưu trữ
  - Tăng hiệu suất I/O
  - Tăng tính sẵn sàng cao
  - Tăng khả năng phục hồi hệ thống
- Các loại RAID
  - RAID 0 → RAID 10 (phổ biến RAID 0, 1, 3, 5)
  - Software RAID/ Hardware RAID

RAID-0



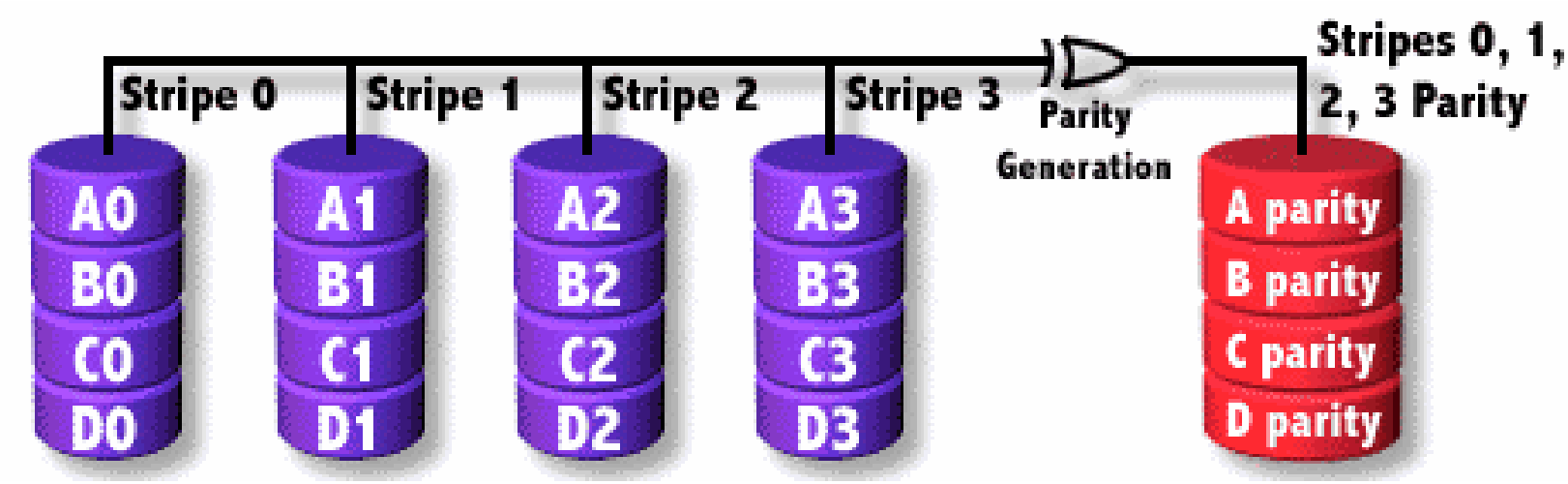
- Dữ liệu lưu trữ trải đều trên các đĩa
- Tăng không gian lưu trữ
- Tăng hiệu suất hệ thống
- Tính sẵn sàng của dữ liệu thấp

RAID-1



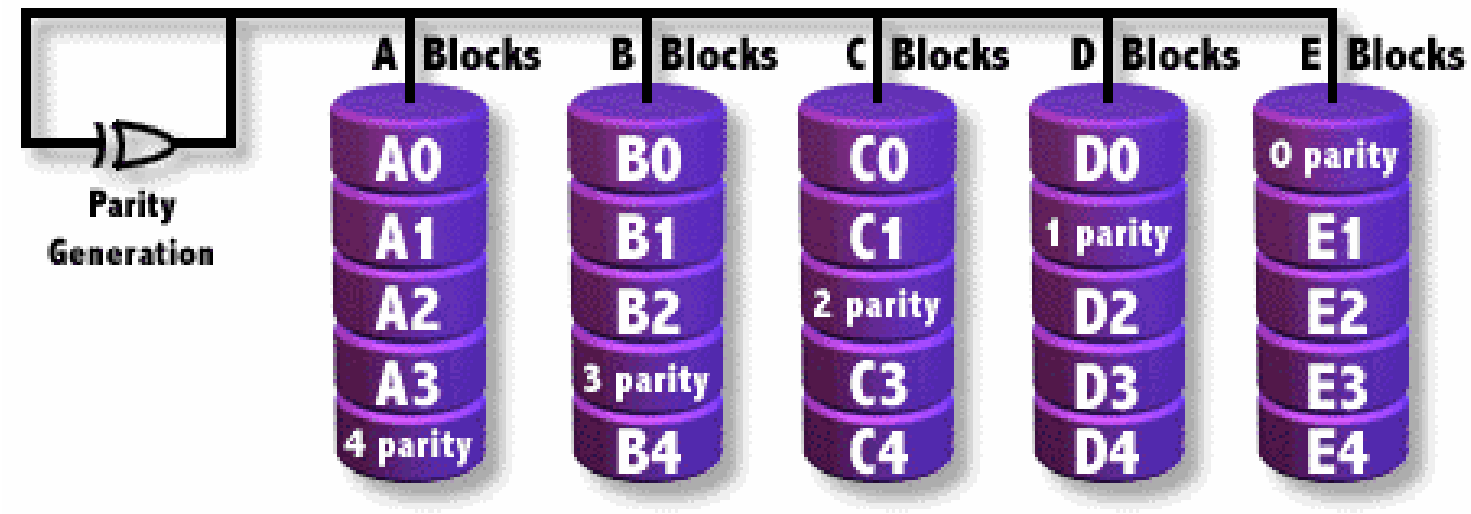
- Nhân bản dữ liệu trên các đĩa tách biệt
- Tính sẵn sàng & tốc độ đọc dữ liệu rất cao
- Yêu cầu dung lượng đĩa gấp đôi
- Tốc độ ghi chậm hơn

RAID-3



- Lưu dữ liệu trải đều trên các đĩa
- Sử dụng một đĩa lưu thông tin kiểm tra dữ liệu
- Tính sẵn sàng cao, chi phí hợp lý
- Hiệu suất I/O thấp

RAID-5



- Dữ liệu, thông tin kiểm tra được lưu trải đều trên các đĩa
- Tính sẵn sàng dữ liệu trung bình, chi phí hợp lý
- Tốc độ ghi thấp
- Yêu cầu phần cứng đặc biệt

BÀI TẬP

1. So sánh thời gian các lệnh copy, move, delete trong tất cả các trường hợp có thể có.
2. Tại sao trong UNIX không có system call detete(…) để xoá file mà chỉ có system call unlink(…) để xoá một link đến file?
3. Đĩa có N block, dùng p/p grouping (4 block) để quản lý vùng trống. Tính thời gian trung bình để tìm được n khối trống và cấp phát cho file.

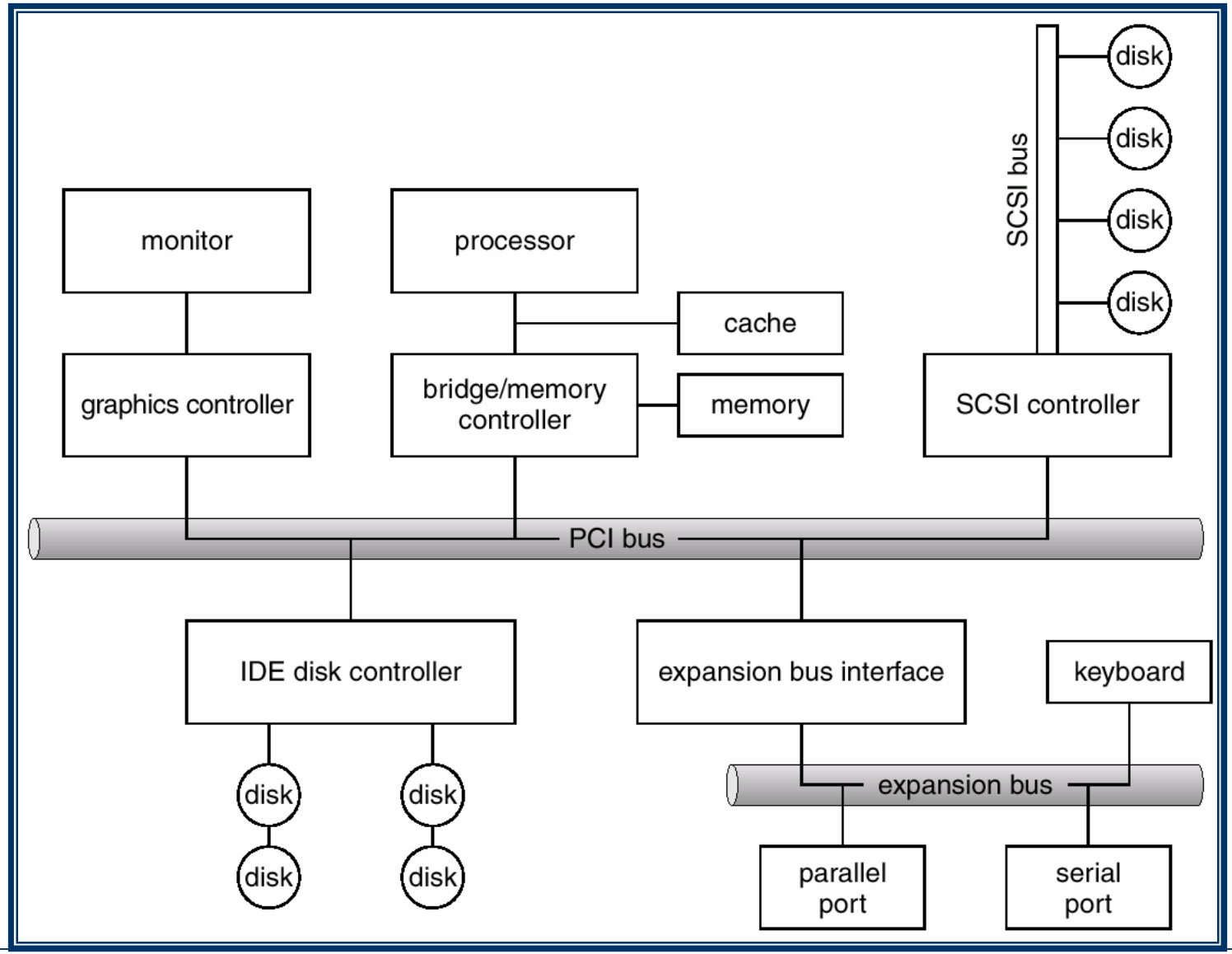
CHƯƠNG 11: HỆ THỐNG NHẬP/ XUẤT (I/O SYSTEMS)

- Giới thiệu hệ thống nhập/xuất
- Phân loại thiết bị I/O
- Các phương pháp truy cập I/O
  - Polling
  - Dùng interrupt
  - Dùng DMA
- Các dịch vụ I/O của hệ điều hành
- Các chức năng quản lý I/O
- Nâng cao hiệu suất của hệ thống I/O

GIỚI THIỆU HỆ THỐNG NHẬP XUẤT

- Hỗ trợ giao tiếp giữa người dùng –hệ thống và giữa các hệ thống với nhau
- Thiết bị phần cứng phục vụ I/O (*I/O Hardware*)
  - Device
  - Bus
  - Controller
  - Port: status, control, data in, data out
- Các phần mềm phục vụ I/O (*I/O Software*)

KẾT NỐI CÁC THIẾT BỊ I/O TRÊN PC



TIÊU CHUẨN PHÂN LOẠI CÁC THIẾT BỊ I/O

- Thiết bị khối (block device)
  - Có khả năng định địa chỉ trực tiếp
  - Không định địa chỉ trực tiếp
- Thiết bị theo ký tự (character-stream device)
- Thiết bị truy xuất tuần tự / ngẫu nhiên
- Thiết bị truy xuất đồng bộ/ bất đồng bộ
- Thiết bị chia xẻ được hay thiết bị dành riêng
- Tốc độ của thiết bị : nhanh, chậm
- Chế độ truy cập thiết bị:
  - Thiết bị chỉ đọc, chỉ ghi, đọc / ghi

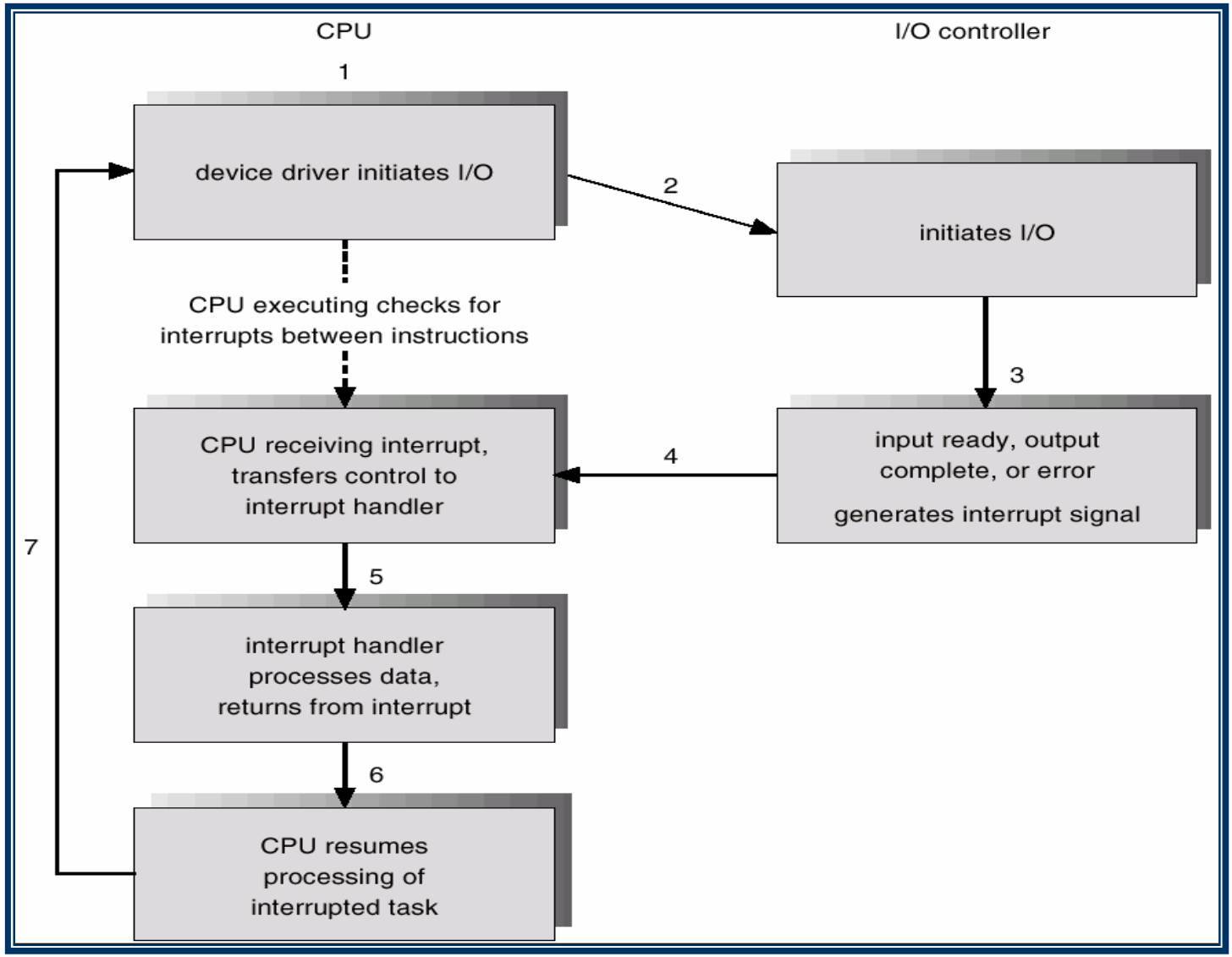
TRUY CẬP I/O BẰÈNG PHƯƠNG PHÁP POLLING

- Hiện thực:
  - Trạng thái thiết bị: busy, error hay command-ready
  - Khi thiết bị rảnh, CPU ra lệnh truy cập I/O
  - Controller đọc lệnh và thực thi tác vụ.
  - Khi thực thi xong, controller đặt lại trạng thái của thiết bị: idle hoặc error.
  - CPU liên tục kiểm tra trạng thái thiết bị để đọc, ghi dữ liệu nếu cần thiết
- Nhận xét

TRUY CẬP I/O BẰÈNG CÁCH SỬ DỤNG NGẮT QUÃNG

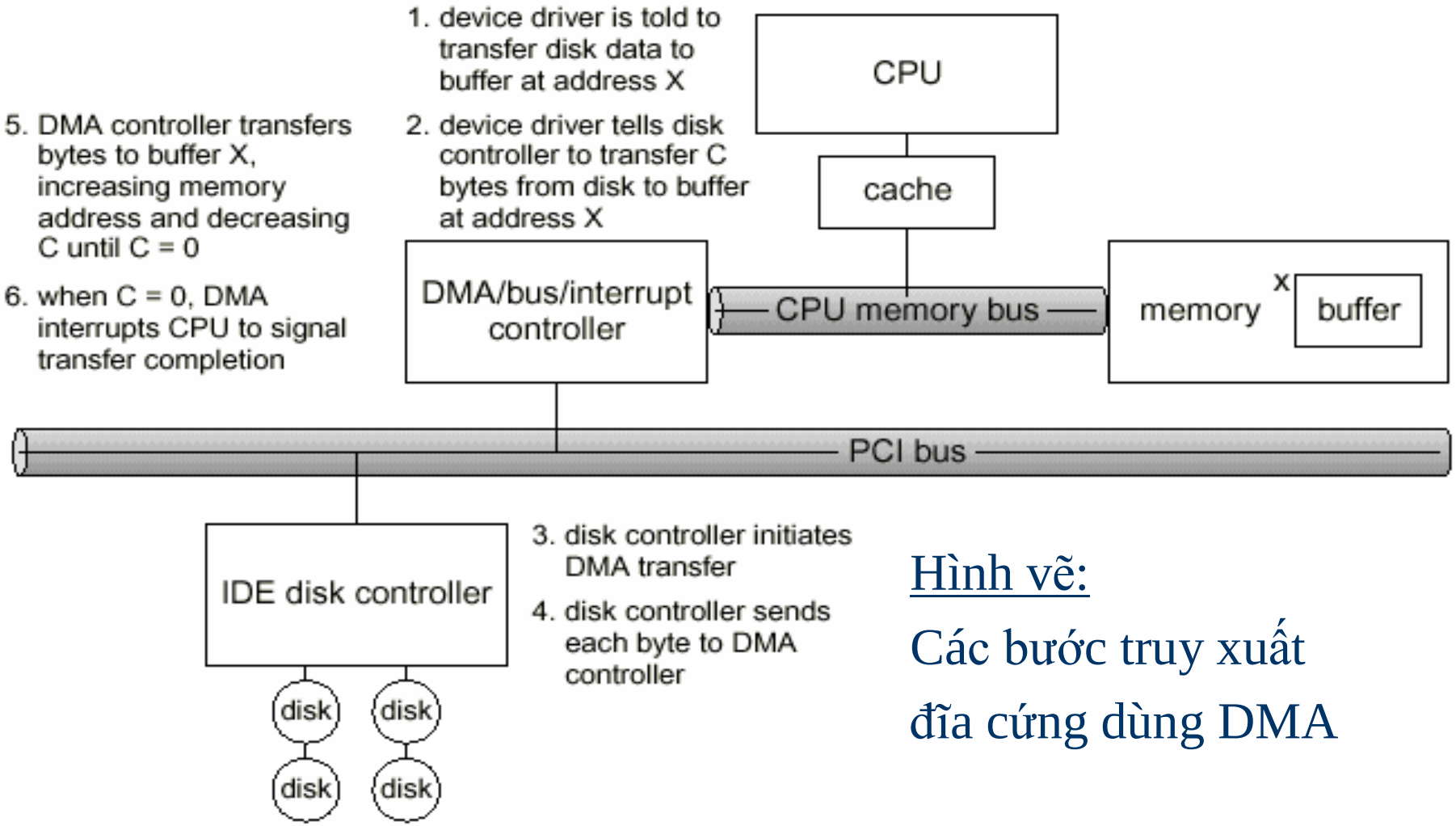
- Thiết bị I/O tạo ngắt quãng khi hoàn tất tác vụ
- Khi có ngắt quãng
  - Trình xử lý ngắt quãng nhận interrupt
  - Xác định thiết bị gây ngắt quãng
  - Lấy dữ liệu từ device register (lệnh trước là lệnh đọc)
  - Khởi động lệnh tiếp theo cho thiết bị đó
- Xử lý ngắt quãng theo độ ưu tiên, có thể hoãn hoặc hủy một số ngắt quãng
- Ngắt quãng cũng được dùng khi xảy ra biến cố
- Nhận xét?

CHU KỲ NGẮT QUÃNG CỦA I/O



DIRECT MEMORY ACCESS - DMA

- Cho phép thiết bị I/O trao đổi dữ liệu trực tiếp với bộ nhớ không cần thông qua CPU



Hình vẽ:

Các bước truy xuất đĩa cứng dùng DMA

CÁC PHẦN MỀM PHỤC VỤ I/O (*I/O SOFTWARE*)

- **Các trình xử lý ngắt quãng**

  (*Interrupt Service Routines*)
- **Các trình điều khiển thiết bị**

  (*Device Drivers*)
- **Các dịch vụ của hệ điều hành**
- **Thư viện lập trình và các chương trình khác**

CÁC DỊCH VỤ I/O DO HỆ ĐIỀU HÀNH CUNG CẤP

- Đặt tên thiết bị
- Các tác vụ xử lý
- Điều khiển truy cập
- Cấp phát thiết bị
- Định thời cho các thiết bị I/O
- Các kỹ thuạt bufffer, cache, spool
- Xử lý và phục hồi lỗi

GIAO TIẾP I/O BLOCKING & NONBLOCKING

- Blocking – Quá trình gọi sẽ treo đến khi giao tiếp I/O xong
  - Dễ hiểu, dễ sử dụng
  - Không hiệu quả trong một số trường hợp
- Nonblocking – Hàm I/O return ngay khi có thể
  - Hiện thực bằng kỹ thuật multi-threading
  - Trả về ngay số byte được đọc hoặc ghi
- Asynchronous – Quá trình tiếp tục chạy khi đang giao tiếp với I/O
  - Khó dùng
  - I/O subsystem báo hiệu cho quá trình khi hoàn thành tác vụ I/O.

CÁC CHỨC NĂNG QUẢN LÝ I/O

- Do module quản lý I/O của hệ điều hành (Kernel I/O Subsystem) đảm nhận
- Các chức năng chính
  - Định thời I/O (I/O scheduling)
  - Dành riêng thiết bị (device reservation)
  - Xử lý lỗi (error handling)
  - Buffering
  - Caching
  - Spooling

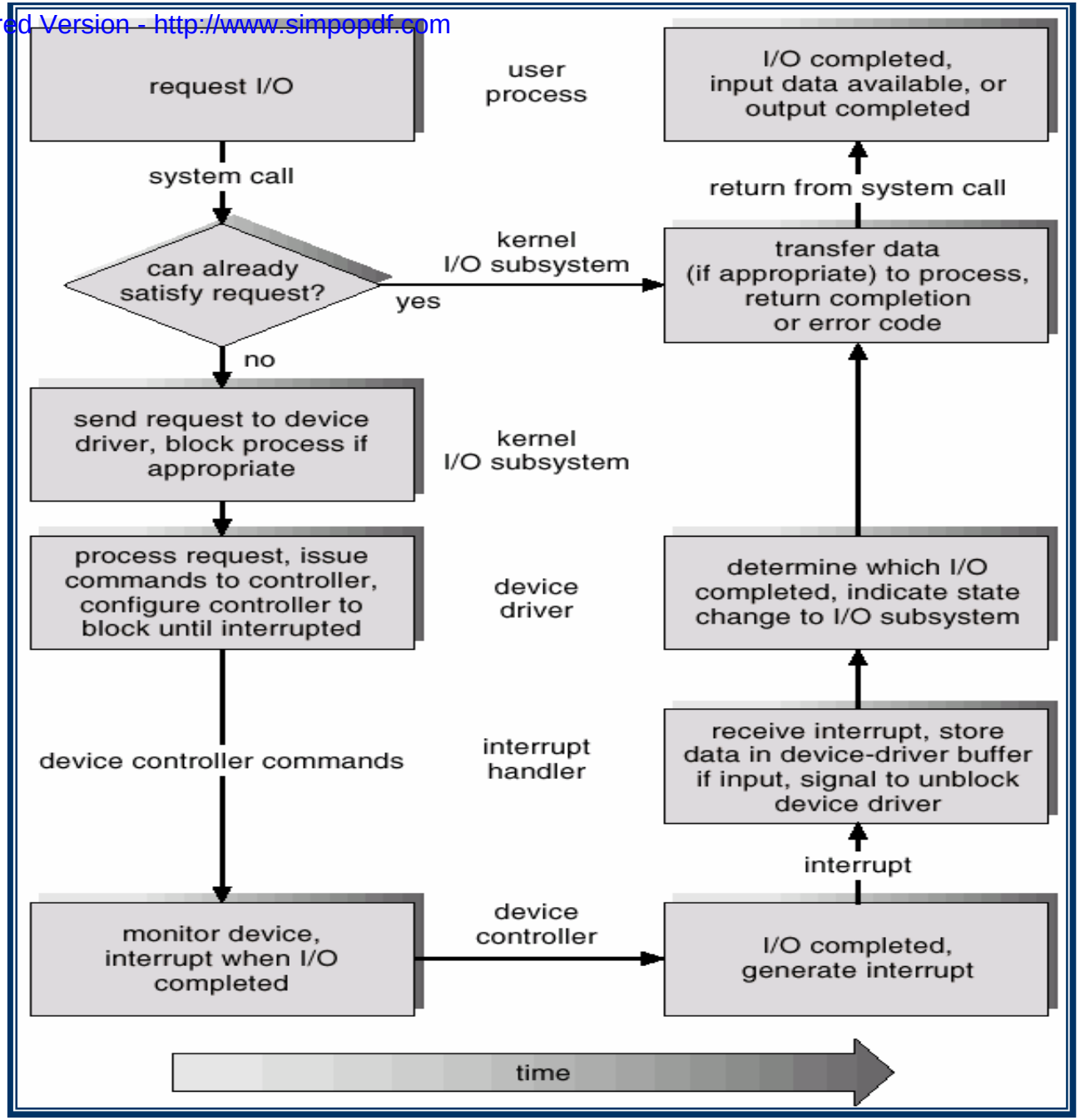
BUFFERING

- Lưu dữ liệu trong bộ nhớ thay vì chuyển trực tiếp dữ liệu giữa các thiết bị
- Có thể được hiện thực ở
  - Cấp phần cứng
  - Cấp hệ điều hành
- Mục đích
  - Xử lý vấn đề tốc độ các thiết bị khác nhau
  - Xử lý vấn đề kích thước khối dữ liệu trao đổi giữa các thiết bị khác nhau
  - Giảm thiểu thời gian quá trình bị chặn khi ghi dữ liệu

CACHING & SPOOLING

- **Caching**
  - Dùng vùng nhớ tốc độ cao để lưu bản sao của dữ liệu thường xuyên truy xuất
  - Đảm bảo tính nhất quán của cache:
    - Kỹ thuật write through
    - Kỹ thuật write back
- **Spooling** (*Simultaneous Peripheral Operation On-line*)
  - Dùng thiết bị lưu trữ tốc độ trung bình làm trung gian giao tiếp giữa 2 thiết bị có tốc độ chênh lệch nhau
  - Ví dụ : dịch vụ in ấn

CHU KỲ I/O



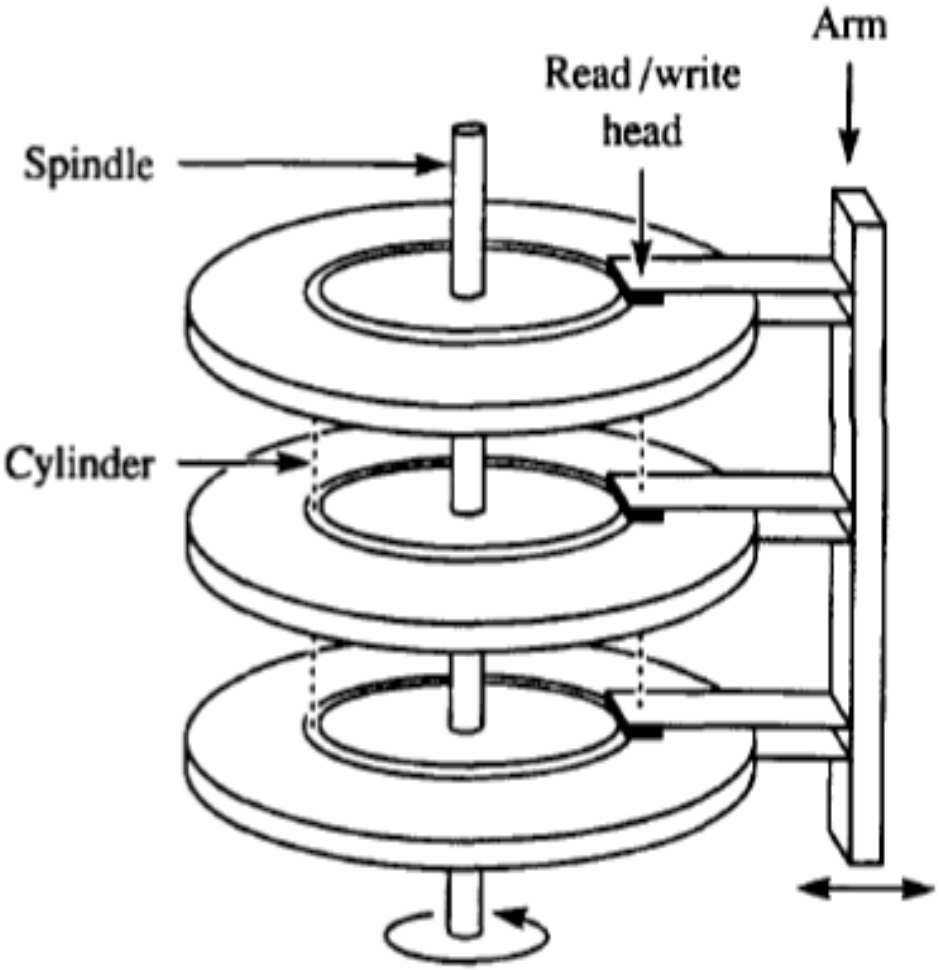
TĂNG HIỆU SUẤT HỆ THỐNG I/O

- Giảm thiểu copy dữ liệu (caching)
- Giảm tần số interrupt (dùng kích thước khối dữ liệu truyền nhận lớn, smart controller…)
- Giảm tải cho CPU bằng DMA
- Tăng số lượng thiết bị để tránh tranh chấp
- Tăng dung lượng bộ nhớ thực
- Cân bằêng hiệu suất CPU, bộ nhớ, bus và thiết bị I/O để đạt throughput cao nhất
- …

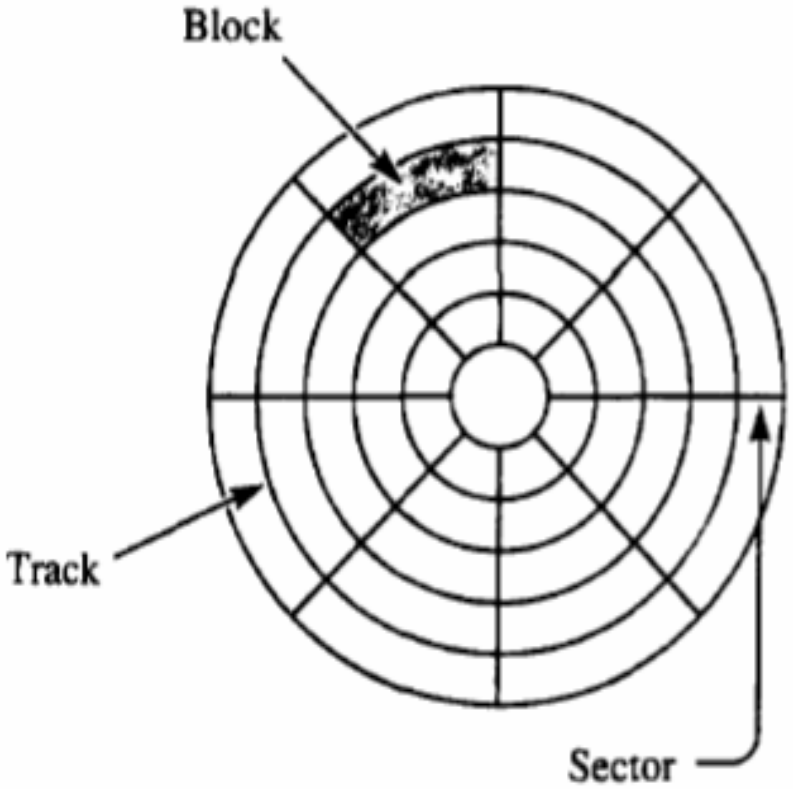
CHƯƠNG 12: QUẢN LÝ ĐĨA CỨNG

- Cấu trúc đĩa cứng
- Nội dung đĩa cứng
- Truy xuất đĩa & định thời truy xuất đĩa
- Quản lý đĩa
- Hiện thực hệ thống lưu trữ ổn định
- Các kỹ thuật tăng hiệu suất đĩa cứng

CẤU TRÚC ĐĨA CỨNG

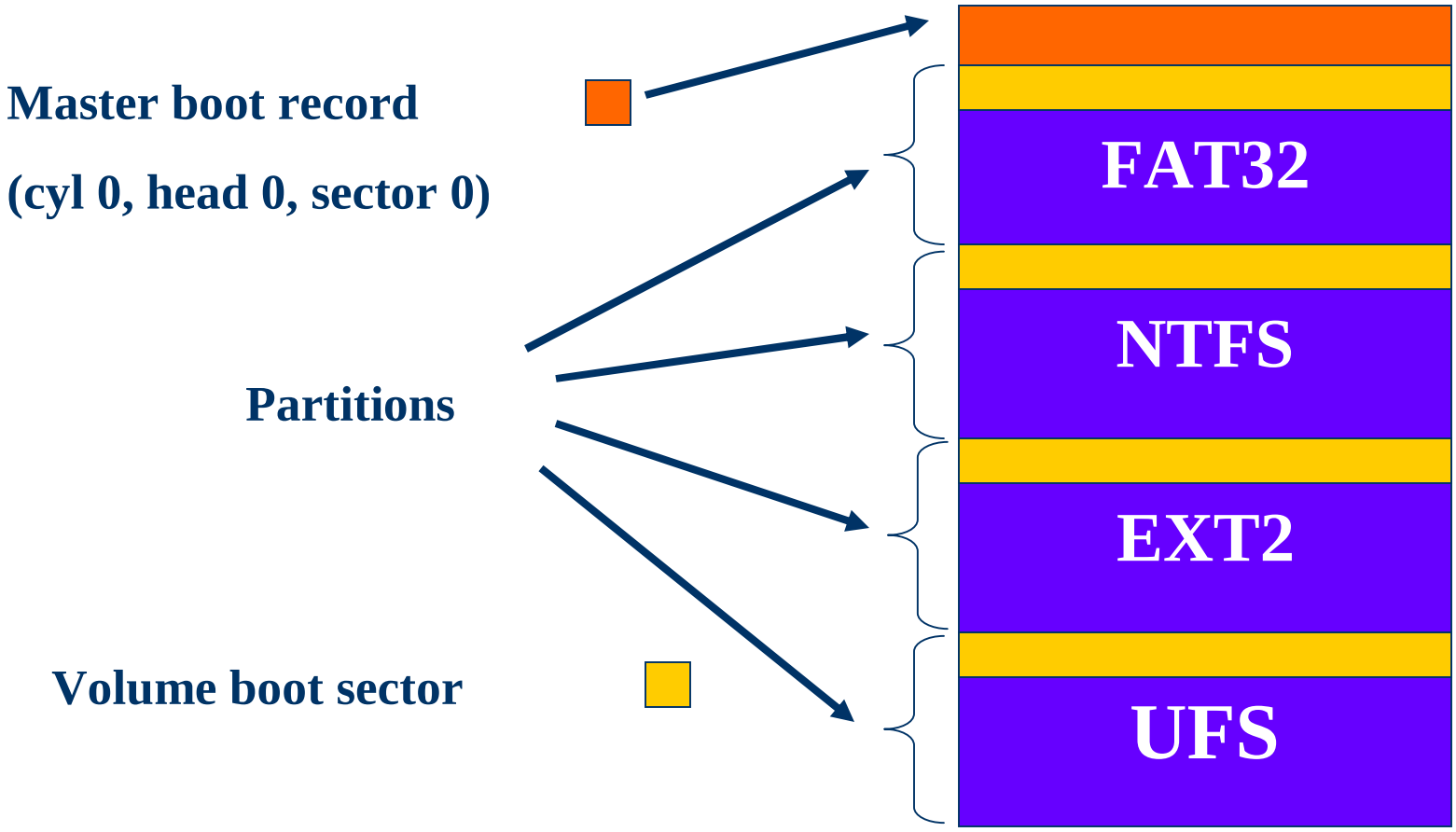


(a) A hard disk drive.



(b) A single disk.

NỘI DUNG LUẬN LÝ ĐĨA CỨNG



NỘI DUNG ĐĨA CỨNG

- Master Boot Record
  - Master Partition Table:
    - Chứa thông tin về từng partition: partition ID, Activity flags, start CHS, end CHS…
    - Link tới Extended Partition Table (chứa thông tin về ổ đĩa luận lý thứ 1 trên đĩa)
  - Master Boot Code:
    - Chứa mã nạp OS ở các partition active
- Partition
  - Vùng không gian liên tục trên đĩa
  - Chứa 1 hệ thống file hoặc n ổ đĩa luận lý (logical volume)
  - Mỗi ổ đĩa luận lý có 1 Volume Boot Sector (VBS)
    - Disk Parameter Block: thông tin về đĩa luận lý
    - Volume Boot Code: mã để khởi động OS trên ổ luận lý này

TRÌNH TỰ KHỞI ĐỘNG HỆ THỐNG

- Power-On Self Test (POST)
  - Kiểm tra phân cứng
  - Chạy các hàm BIOS mở rộng trong các ROM ở các mạch ngoại vi
- BIOS gọi interrupt 13h, nạp MBR và khởi động Master Boot Code (MBC)
- MBC nạp VBS của partition chính tích cực đầu tiên trên đĩa khởi động
- Volume Boot Code khởi động OS
- Các BIOS & OS mới có thể cho boot từ CDROM, đĩa mềm, đĩa ZIP hoặc qua mạng (Remote Boot)

TRUY XUẤT ĐĨA CỨNG

- 3 yếu tố ảnh hưởng thời gian truy xuất đĩa
  - **Seek time:** thời gian di chuyển đầu đọc tới track
  - **Latency:** thời gian để quay đĩa sao cho sector cần đọc nằm dưới đầu đọc
  - **Transfer time**: thời gian đọc/ ghi dữ liệu lên sector
- Thực tế:
  - Seek time >> latency time > transfer time
- Tối ưu seek time ➔ định thời truy xuất đĩa
- Tối ưu latency time:
  - Làm đĩa nhỏ, quay nhanh hơn, lưu trữ dữ liệu liên quan gần nhau
  - Chọn kích thước sector, nơi lưu trữ các file thường dùng hợp lý

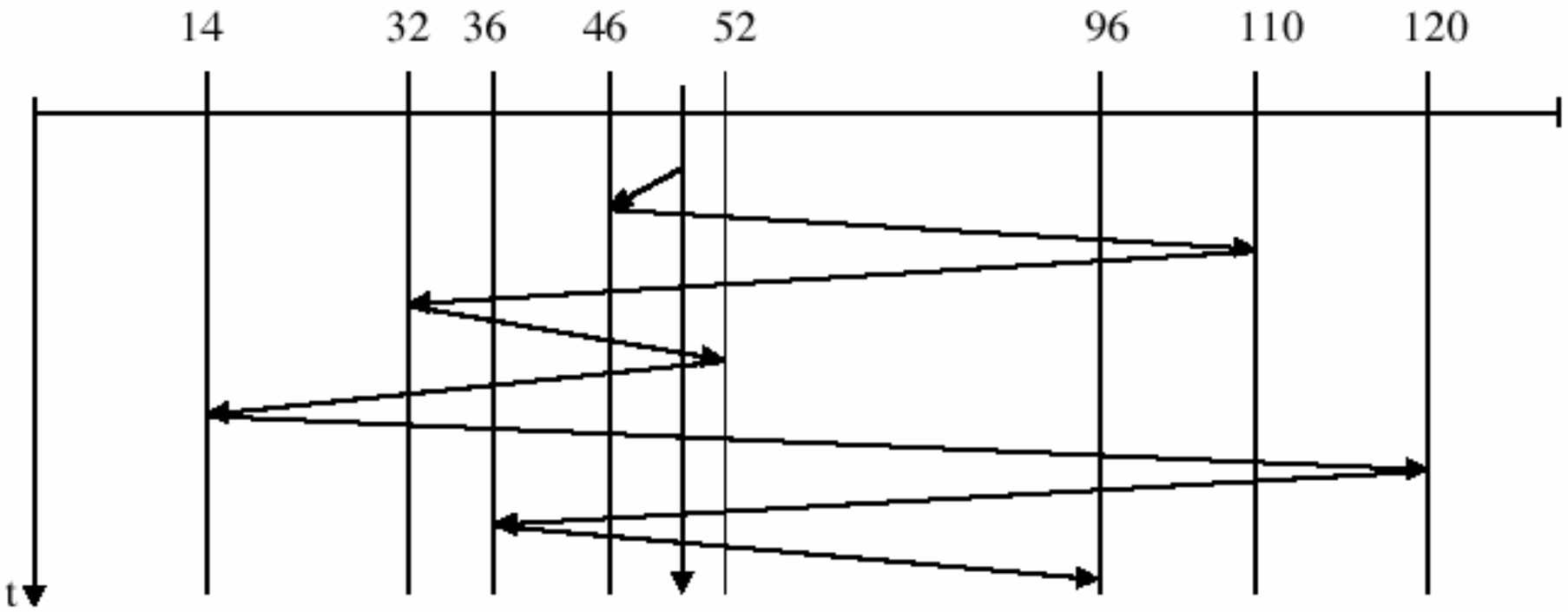
CÁC GIẢI THUẬT ĐỊNH THỜI ĐĨA

- Bài toán: Cóù n yêu cầu đọc đĩa ở các track khác nhau x1, x2, … , xN vào các thời điểm tương ứng t1, t2, …, tN

  ➔ phục vụ các yêu cầu đó vào thời điểm nào?
- Tiêu chuẩn đánh giá
  - Công bằng
  - Hiệu suất cao
  - Thời gian đáp ứng trung bình thấp
  - Dự đoán được thời gian phục vụ
- Một số giải thuật tiêu biểu:
  - FCFS
  - SSTF
  - SCAN, N-step-SCAN, C-SCAN
  - CLOOK

ĐỊNH THỜI TRUY XUẤT ĐĨA –FCFS

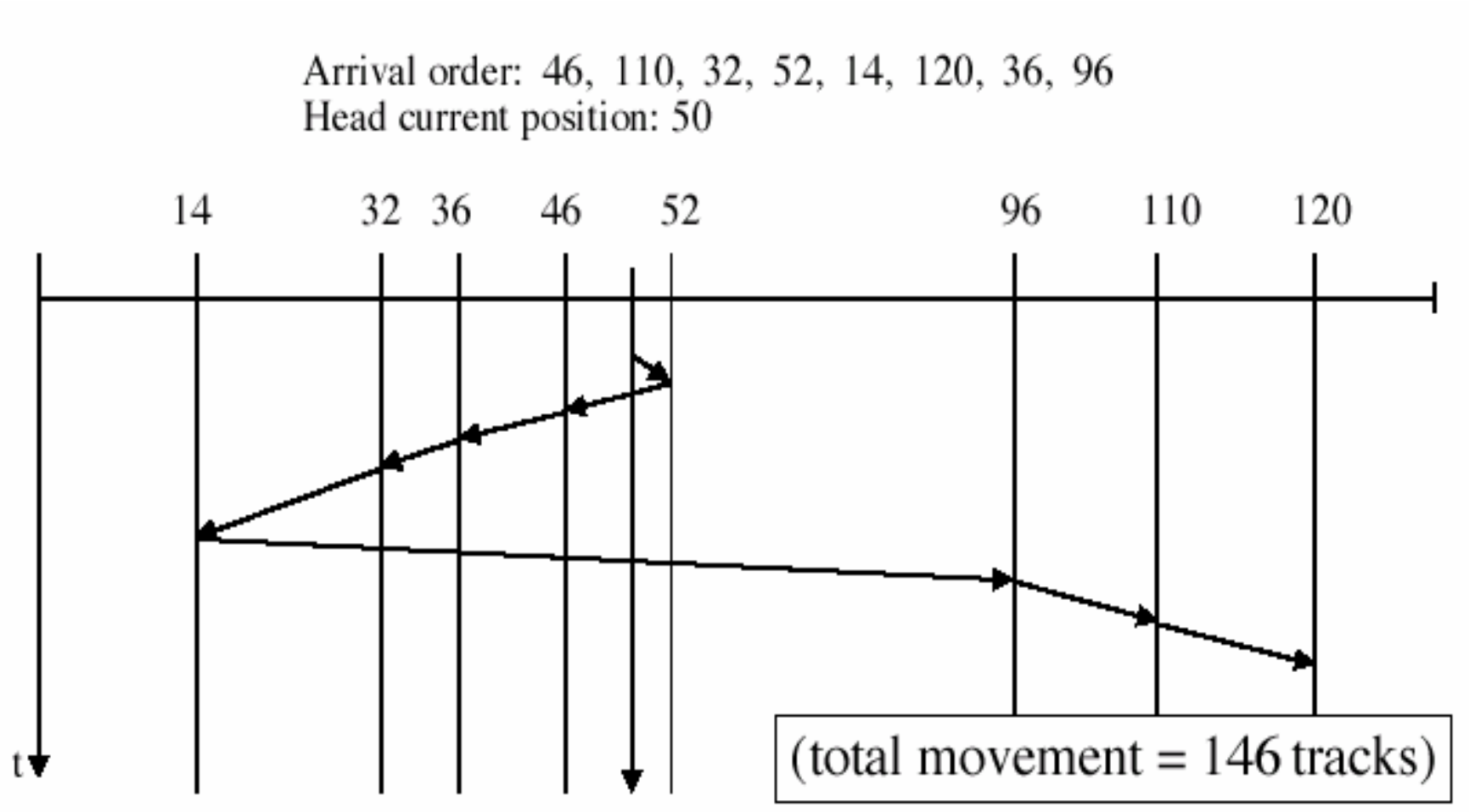
Arrival order: 46, 110, 32, 52, 14, 120, 36, 96 (track addresses)
Head current position: 50



Total head movement = 454 tracks ➔ **Nhận xét ?**

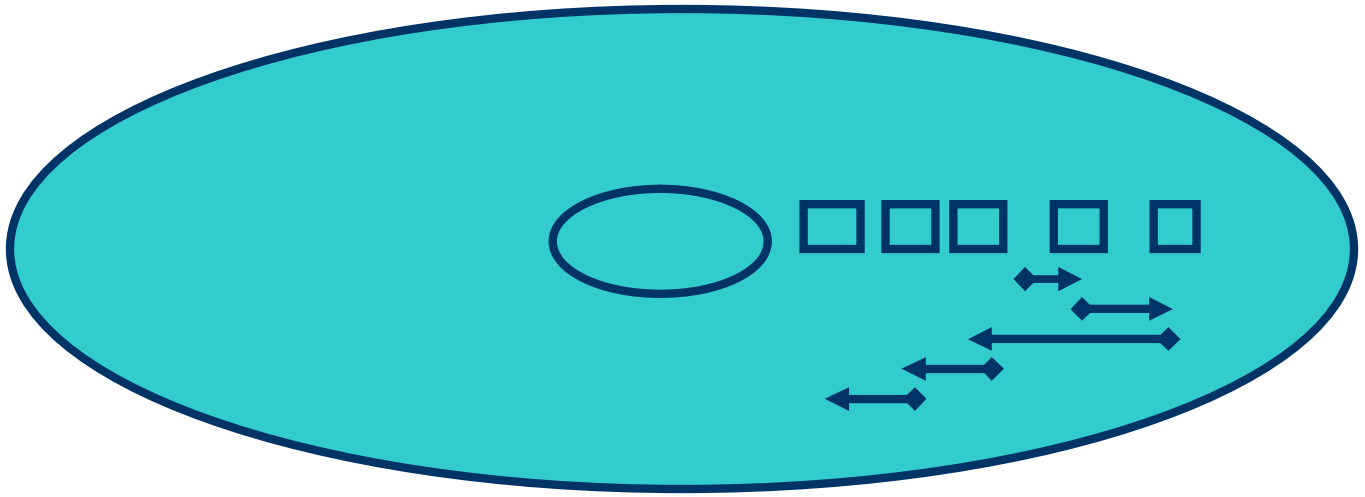
GIẢI THUẬT SSTF (Shortes Seek Time First)

- **Phục vụ yêu cầu đọc gần vị trí đầu đọc hiện tại nhất.**



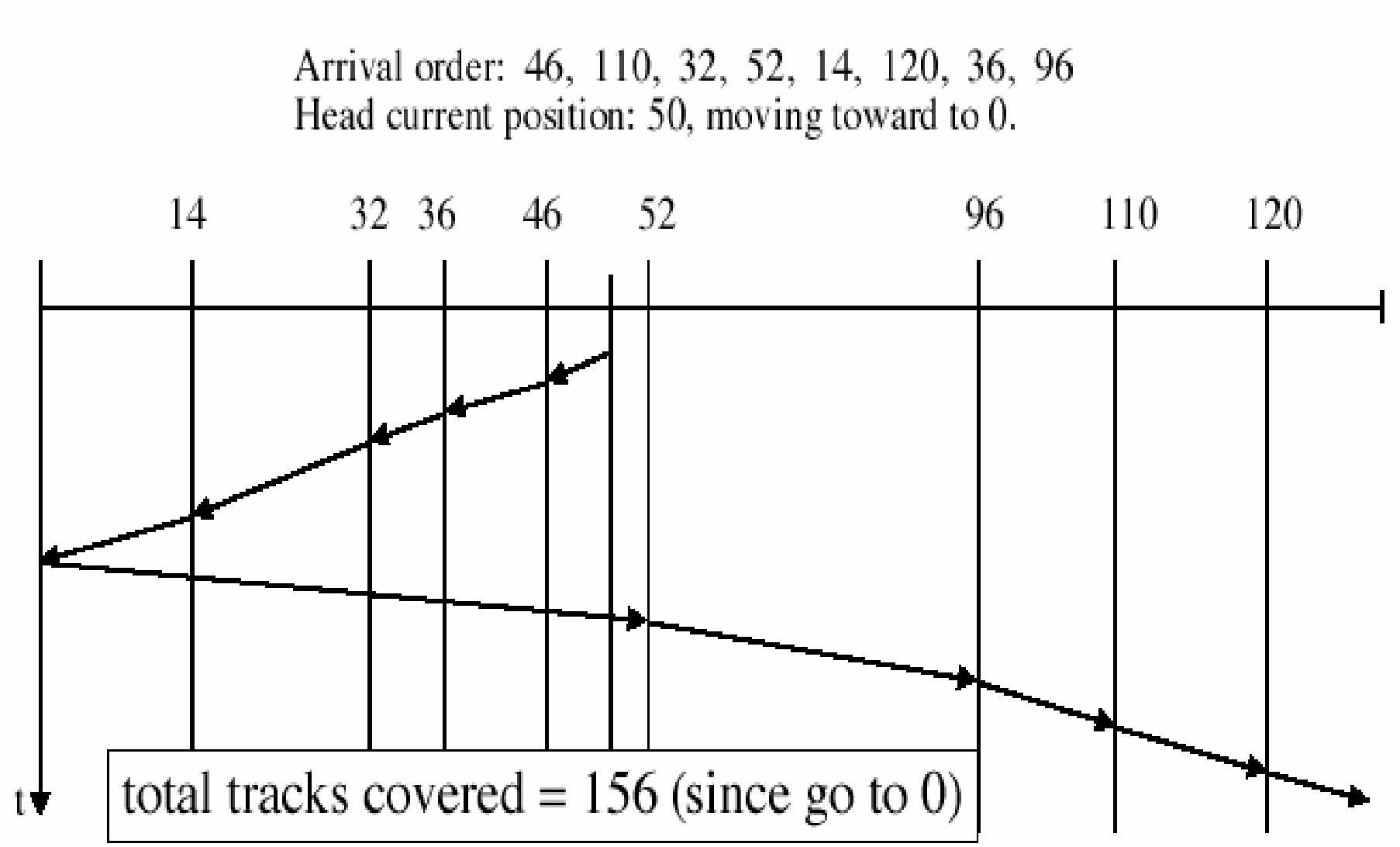
GIẢI THUẬT SCAN

- Phục vụ theo hướng phục vụ từ trong ra ngoài
- Khi đầu đọc ra tới track ngoài cùng, phục vụ theo hướng ngược lại từ ngoài vào trong



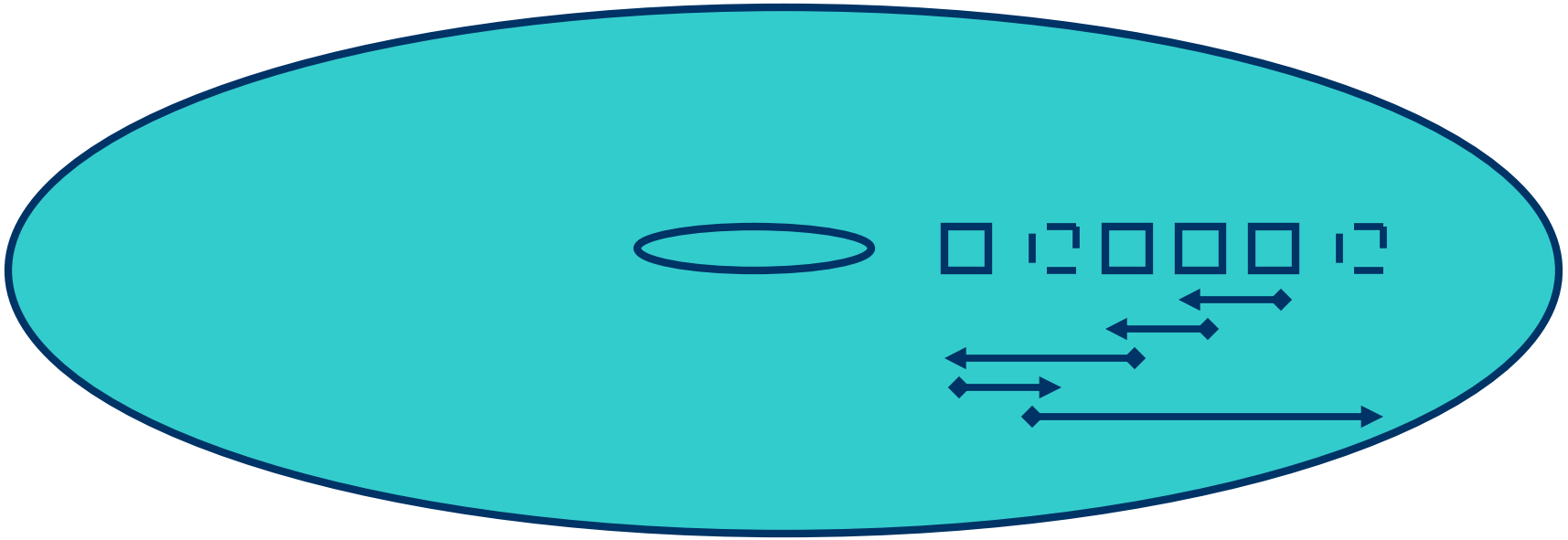
- Nhận xét?

VÍ DỤ VỀ GIẢI THUẬT SCAN



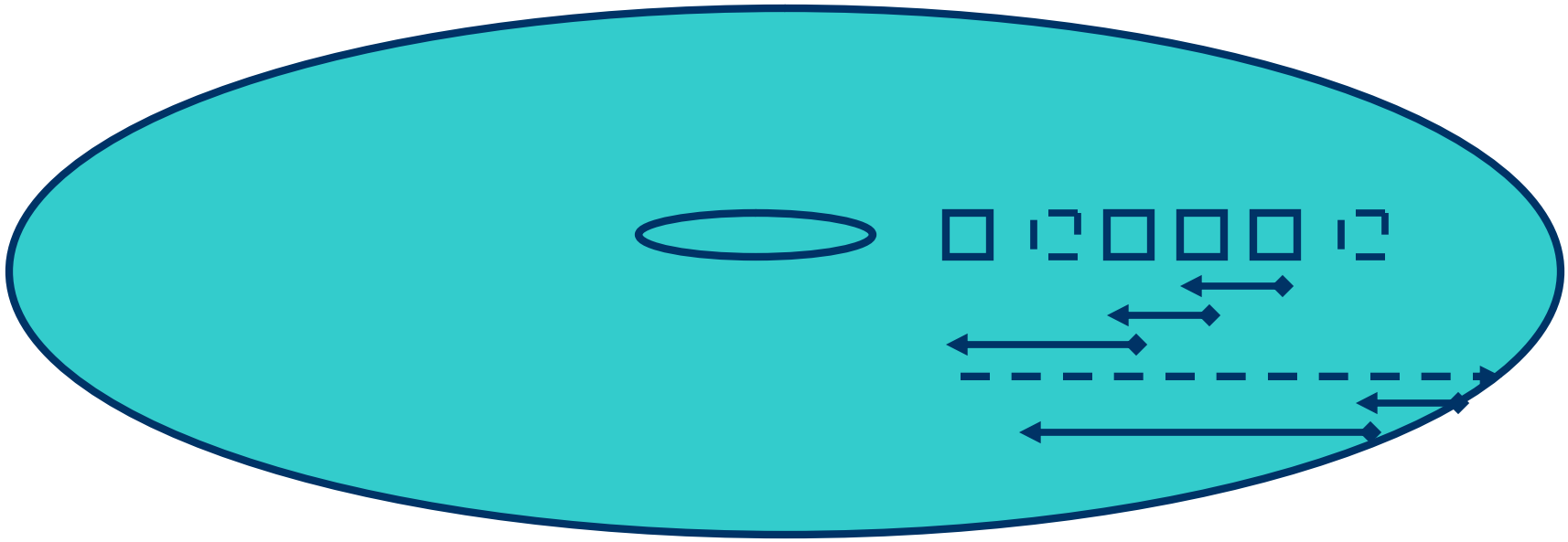
GIẢI THUẬT N-step-SCAN

- Nhóm các yêu cầu truy xuất lại
- Phục vụ nguyên 1 nhóm yêu cầu theo 1 chiều di chuyển của đầu đọc



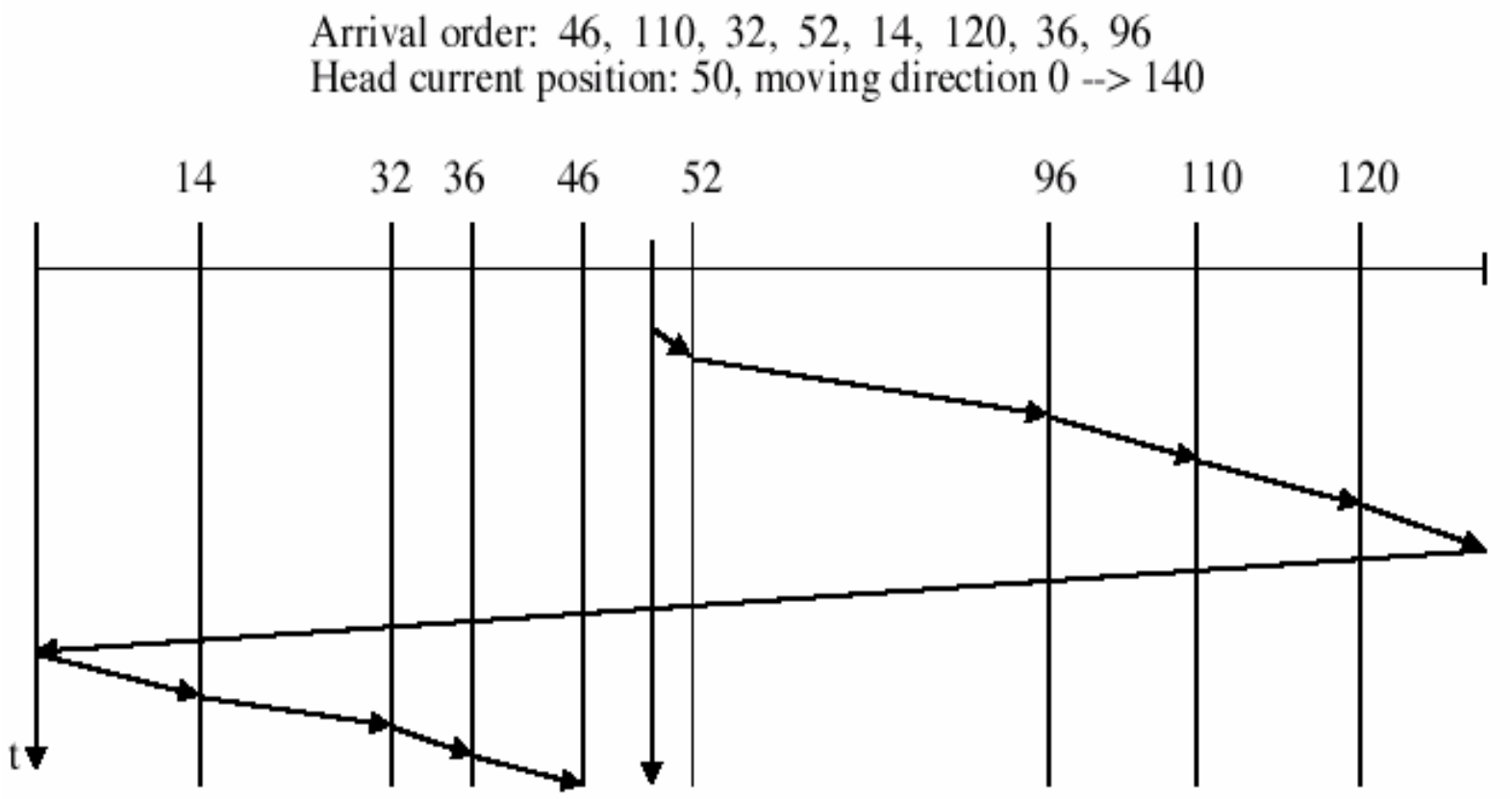
GIẢI THUẬT C-SCAN

- Như giải thuật N-step-SCAN nhưng theo chỉ phục vụ theo 1 hướng duy nhất
- Nhận xét?



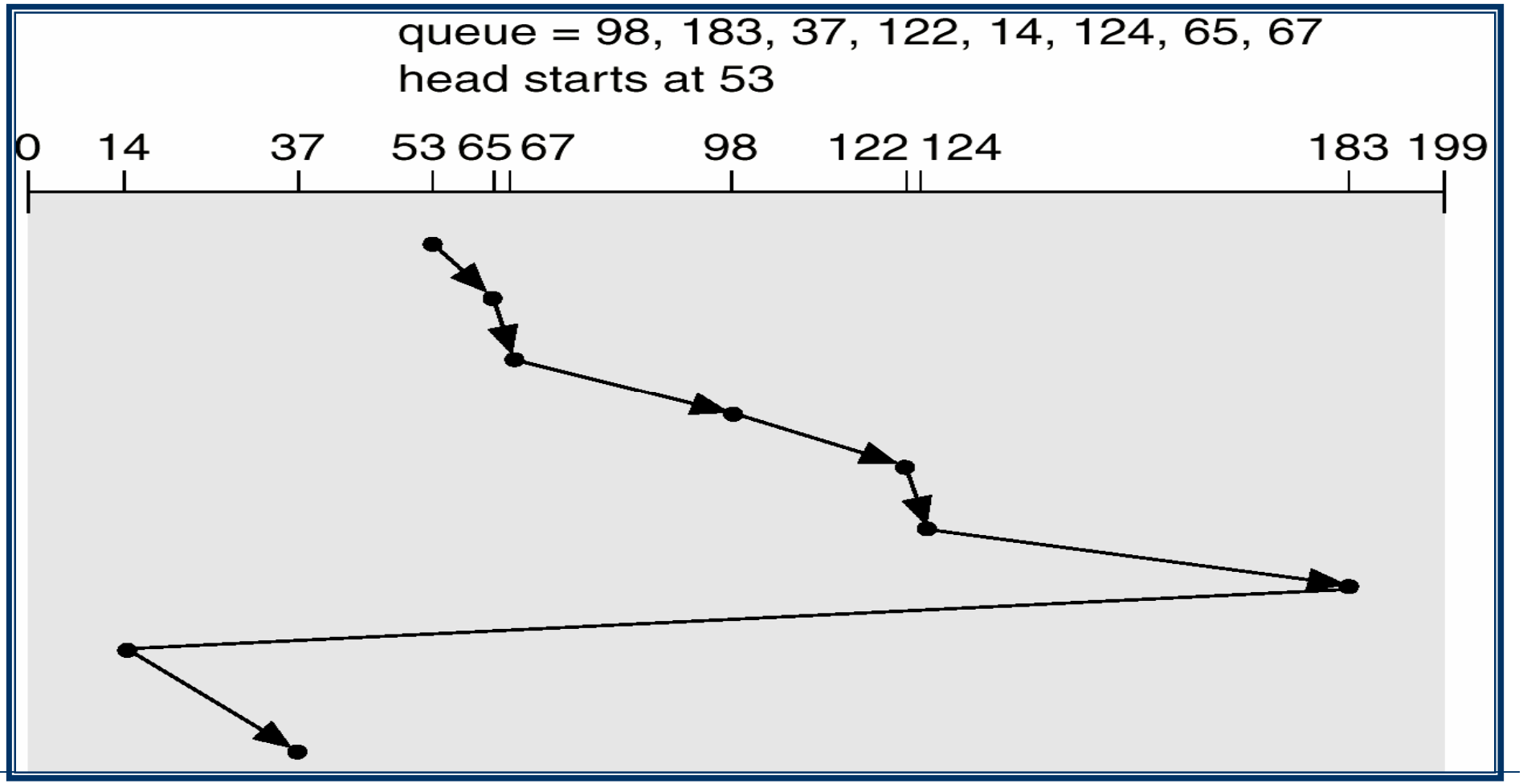
VÍ DỤ VỀ GIẢI THUẬT C-SCAN

- Như giải thuật N-step-SCAN nhưng chỉ phục vụ theo 1 hướng duy nhất
- Nhận xét?



GIẢI THUẬT C-LOOK

- Như C-SCAN, nhưng chỉ di chuyển đầu đọc tới track ngoài cùng được phục vụ rồi quay lại track trong cùng cần phục vụ



QUẢN LÝ ĐĨA

- Low-level formatting: chia đĩa ra các sector để disk controller có thể đọc, ghi được
- Lưu cấu trúc dữ liệu của OS lên đĩa
  - Partitioning: phân vùng đĩa
  - High-level formatting: tạo hệ thống file trên partition
- Tạo boot block
- Xử lý lỗi: kỹ thuật sector sparing
- Quản lý vùng swap
  - Tạo vùng swap khi nào?
  - Sử dụng dùng swap-map
- Lắp đặt đĩa
  - qua cổng I/O
  - qua mạng (Network Attached Storage)

HỆ THỐNG LƯU TRỮ ỔN ĐỊNH (Stable Storage System)

- Đảm bảo thông tin lưu trữ luôn tồn tại dù bất kỳ lỗi nào xảy ra trong quá trình đọc/ghi.
- Các vấn đề xảy khi đọc/ghi đĩa thường:
  - Ghi thành công: block đích chứa thông tin mới
  - Thất bại một phần: block đích chứa thông tin sai
  - Thất bại hoàn toàn: block đích chứa thông tin như cũ
- Hiện thực: dùng 2 block vật lý cho 1logical block
  - Ghi thông tin vào block (vật lý) thứ 1 rồi thứ 2.
  - Việc ghi thành công ⇔ block thứ 2 ghi xong
  - Kiểm tra sự giống nhau của 2 block → phát hiện lỗi và xử lý để đảm bảo tính nhất quán thông tin

CÁC KỸ THUẬT TĂNG HIỆU SUẤT ĐĨA CỨNG

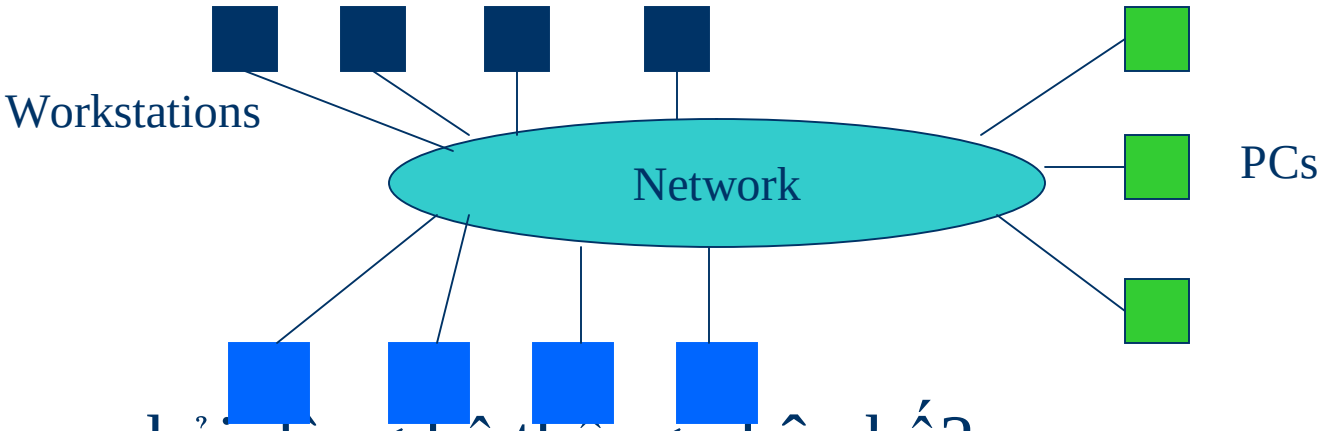
- Lưu dữ liệu truy xuất thường xuyên trong bộ nhớ
  - virtual disk, disk caching
- Kỹ thuật bufferring
  - Read – ahead, write-behind
- Defragment đĩa ➔ giảm seek time
- Phân vùng đĩa ➔ phân mảnh bị giới hạn
- Interleaving ➔ giảm latency time
- Nén dữ liệu
- Đặt các ứng dụng/ file/ directory structure ở giữa đĩa
- Dùng hệ nhiều đĩa cứng (RAID system)
- Hiện thực giải thuật định thời đĩa băng phần cứng

CHƯƠNG 13: HỆ THỐNG PHÂN BỐ (Distributed Systems)

- **Định nghĩa hệ phân bố**
- **Đặc điểm của hệ thống phân bố**
  - Tính chia xẻ tài nguyên
  - Tính mở
  - Tính đồng thời
  - Tính khả mở qui mô
  - Tính kháng lỗi
  - Tính trong suốt
- **Đặc điểm hệ điều hành phân bố**
- **Kiến trúc hệ điều hành phân bố & microkernel**

ĐỊNH NGHĨA HỆ PHÂN BỐ

- Tập các máy tính tự trị được nối mạng với nhau kết hợp lại để tính toán, được trang bị một lớp **phần mềm phân bố**, giúp việc sử dụng hệ thống như 1 máy tính duy nhất



- Tại sao phải dùng hệ thống phân bố?

ĐỊNH NGHĨA HỆ PHÂN BỐ (tt)

- **Phần mềm phân bố:**
  - Cho phép các máy tính chia xẻ tài nguyên, cung cấp dịch vụ truy cập tài nguyên như ở một máy đơn.
- **Ví dụ về hệ thống phân bố**
  - Distributed UNIX system
  - Hệ thống các máy ATM & các máy tính ngân hàng
  - Hệ thống đặt vé, kiểm tra vé máy bay, tàu hỏa…
- **Ví dụ về các dịch vụ phân bố**
  - Hệ thống file phân bố (Network File System)
  - NIS (Network Information System), NIS+, Active Directory, Lightweight Direcoty Access Protocol (LDAP)

ĐẶC ĐIỂM CỦA HỆ PHÂN BỐ

- **Chia xẻ tài nguyên (Resources sharing)**
  - → Cần quản lý tài nguyên hiệu quả
  - → Mô hình client/server hoặc object/ object manager
- **Tính mở (Openness):**
  - Hệ thống phải có khả năng mở rộng theo nhiêu hướng, không làm ảnh hưởng dịch vụ cũ
- **Tính đồng thời (Concurrency)**
  - Nhiều ngươi dùng, chương trình chạy đồng thời, sử dụng tai nguyên trên nhiều máy khác nhau.
- **Tính khả mở qui mô (Scalability)**
  - Tăng kích thước hệ thống không làm ảnh hưởng đến các phần mềm, dịch vụ đang chạy

ĐẶC ĐIỂM CỦA HỆ PHÂN BỐ (tt)

- **Tính kháng lỗi (Fault tolerance)**
  - Chương trình vẫn chạy đúng khi có máy và phần mêm, dịch vụ bị lỗi, hỏng hóc
  - Hiện thực: nguyên tắc dư thừa và phục hồi lỗi ở cấp phần mềm ➔ tính sẵn sàng cao (High Availability)
- **Tính trong suốt (Transparency)**
  - Giúp người dùng cảm nhận hệ thống là một máy tính đơn duy nhất
  - Tiêu chuẩn ISO đưa ra 8 dạng trong suốt về:

    1. Truy cập
    2. Vị trí
    3. Tính đồng thời
    4. Việc nhân bản
    5. Xử lý lỗi
    6. Việc di dời
    7. Tăng hiệu suất hệ thống
    8. Việc mở rộng qui mô

TÍNH TRONG SUỐT (TRANSPARENCY)

- **Trong suốt về truy cập (Access Transparency)**
  - Cho phép truy cập các tài nguyên cục bộ hoặc ở máy ở xa bằng các tác vụ như nhau
  - Ví dụ: NFS (Network File System)
- **Trong suốt về vị trí (Location Trans.)**
  - Người sử dụng có thể dùng tài nguyên mà không cần biết vị trí của tài nguyên trong hệ thống
  - Ví dụ: Dịch vụ tên NIS, Active Directory
- **Trong suốt về sự đồng thời (Concurrency Trans.)**
  - Các user dùng cùng tài nguyên không cần biết sự hiện diện của các user khác và không cản trở lẫn nhau.

TÍNH TRONG SUỐT (tt)

- **Trong suốt về việc nhân bản (Replication Trans.)**
  - Cho phép tạo nhiều bản sao(replica) của tài nguyên
  - User không cần biết sự tồn tại của các bản sao
  - Ví dụ: Có chế tạo bản sao của Oracle DBMS
- **Trong suốt về mặt xử lý lỗi (Failure Trans.)**
  - Che giấu lỗi nếu có xảy ra và cho phép ứng dụng của người dùng có thể hoàn thành
  - Ví dụ: xử lý giao tiếp trong Java CORBA, RMI, Jini.
- **Trong suốt về sự di dời (Migration Trans.)**
  - Sự di chuyển của các tài nguyên trong hệ thống không ảnh hưởng đến các hoạt động của người dùng và trình ứng dụng.

TÍNH TRONG SUỐT (tt)

- **Trong suốt về hiệu suất (Performance Trans.)**
  - Cho phép hệ thống có thể được tái cấu hình để cải thiện hiệu suất xử lý và thay đổi tải
  - Quá trình tái cấu hình không ảnh hưởng đến hoạt động của người dùng và ứng dụng
- **Trong suốt về mở rộng qui mô(Scaling Trans.)**
  - Cho phép hệ thống và ứng dụng mở rộng mà không thay đổi cấu trúc hệ thống và giải thuật của ứng dụng
- **Một số ví dụ và nhận xét:**
  - rlogin: không có tính trong suốt về vị trí, về truy cập
  - E-mal : có tính trong suốt về ví trí, truy cập → có tính trong suốt về mạng
  - Tính trong suốt có thể cần thiết hoặc không cần thiết

HỆ ĐIỀU HÀNH PHÂN BỐ (Distributed Operating System – DOS)

- **Đặc điểm**
  - Cho phép user lập trình dế dàng trên hệ phân bố
  - Cung cấp các k/niệm trừu tượng về tài nguyên để người dùng sử dụng mà không cần quan tâm đến vị trí của chúng
  - Gồm các kernel và các quá trình cung cấp dịch vụ
  - Ít có sự phân biệt giữa hệ điều hành, dịch vụ và ứng dụng trên hệ thốngø
- **Ví dụ**
  - Mach, Chorus: hệ thống thương mại, kỹ thuật
  - Amoeba, Clouds, V System: trong kỹ thuật

KIẾN TRÚC HỆ THỐNG PHÂN BỐ

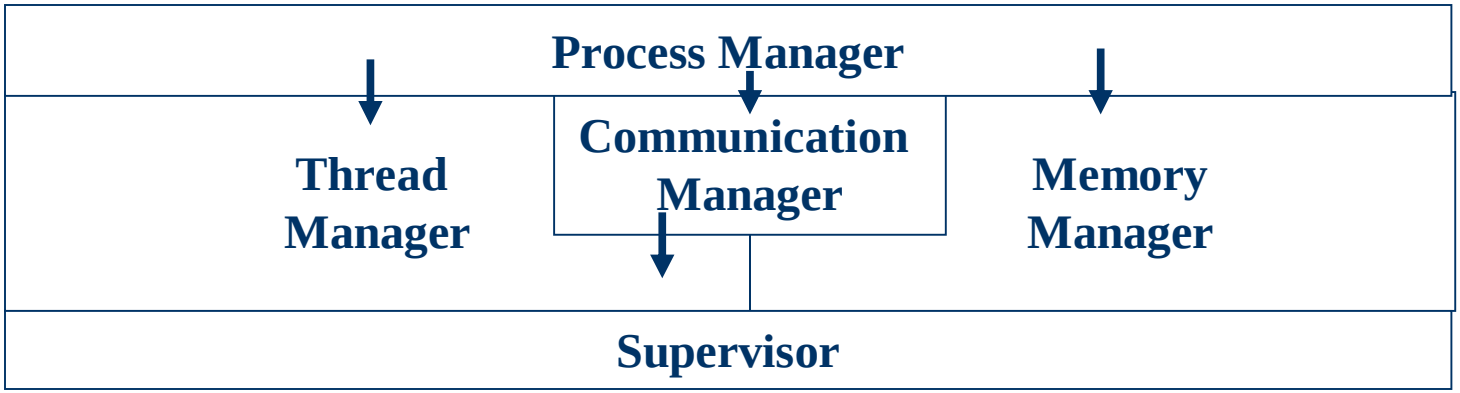
- **Các thành phầân**
  - Hạ tầng cung cấp phương thức quản lý tài nguyên
  - Kernel và các quá trình server: các chương trình quản lý tài nguyên
- **Yêu cầu**
  - Mọi tài nguyên có interface để sử dụng
  - Chi tiết quản lý tài nguyên được đóng gói
  - Xử lý song song việc truy cập các tài nguyên
  - Bảo vệ tài nguyên
- **Các vấn đề liên quan khi thiết kế hệ thống**
  - Đặt tên tài nguyên (Resource naming)
  - Giao tiếp giữa các quá trình (Communication)
  - ĐInh thời dùng tài nguyên (Scheduling)

MÔ HÌNH HỆ ĐIỀU HÀNH PHÂN BỐ

<table>
<tr><td colspan="4">Open Services & Application Processes/ Objects</td></tr>
<tr><td>Language Support Sytem</td><td>Language Support System</td><td>OS Emulation System</td><td>OS Emulation System</td></tr>
<tr><td colspan="4">Microkernel</td></tr>
<tr><td colspan="4">Computer & network hardware</td></tr>
</table>

- Microkernel: cung cấp những dịch vụ cơ bản nhất của hệ điều hành
- Các dịch vụ còn lại do các quá trình server thực hiện
- Hệ thống giả lập được nhiều OS và hỗ trợ nhiều thư viện lập trình của các ngôn ngữ khác nhau.
- Các ứng dụng không dùng dịch vụ của ukernel mà dùng các hệ thống hỗ trợ cho 1 ngôn ngữ hoặc dùng các dịch vụ do OS emulation system cung cấp

KIẾN TRÚC MICROKERNEL



- **Process manager**: quản lý &xử lý các tác vụ cấp thấp cho quá trình. Bao gồm cảhệ thống hỗ trợ các ngôn ngữ và hệ thống giả lập các OS
- **Thread manager**: tạo, đồng bộ, định thời các thread
- **Communication manager**: giao tiếp giữa các thread các quá trình ở các máy khác nhau
- **Memory manager**: quản lý bộ nhớ, cache
- **Supervisor**: gởi interrupt, system call trap, exception đến các trình xử lý tương ứng