HUTECH
Đại học Công nghệ Tp.HCM

HỆ ĐIỀU HÀNH

GV: Lương Trần Hy Hiến
Khoa: Công nghệ Thông tin

https://hienlth.info/os
https://hutechos.weebly.com

MÔ TẢ HỌC PHẦN

- Môn học này cung cấp cho sinh viên những khái niệm tổng quan về hệ điều hành, nhằm phục vụ cho sinh viên ngành Công Nghệ Thông tin.
- Nội dung môn học nhấn mạnh đến các nguyên tắc, các chủ đề, các phương pháp tiếp cận và giải quyết vấn đề liên quan đến các công nghệ và kiến trúc cơ bản của lĩnh vực này.

NỘI DUNG HỌC PHẦN

- Bài 1: TỔNG QUAN
- Bài 2: CẤU TRÚC HỆ ĐIỀU HÀNH
- Bài 3: TIẾN TRÌNH VÀ LUỒNG
- Bài 4: ĐIỀU PHỐI CPU
- Bài 5: ĐỒNG BỘ HÓA TIẾN TRÌNH
- Bài 6: TẮC NGHẼN
- Bài 7: QUẢN LÝ BỘ NHỚ
- Bài 8: QUẢN LÝ BỘ NHỚ ẢO

ĐÁNH GIÁ HỌC PHẦN

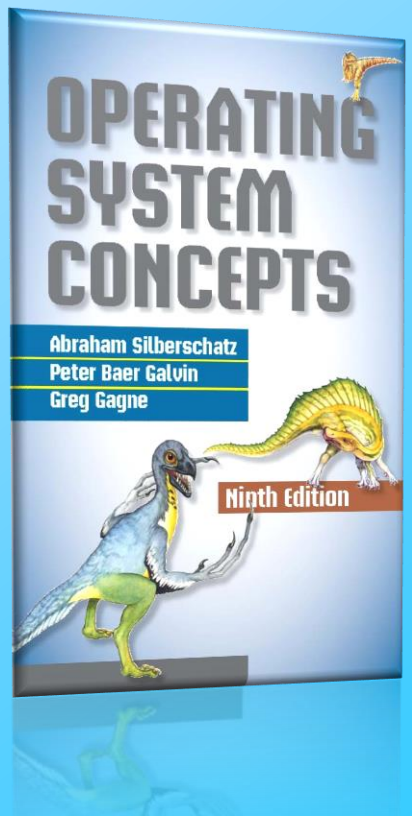
- **Điểm thi thực hành: 30%.**

  Hình thức, nội dung do giáo viên thực hành quyết định.
- **Điểm trên lớp lý thuyết 20%: Chuyên cần + Bài tập**
- **Điểm thi lý thuyết: 50%.**

  Bài thi tự luận trong 90 phút, không được mang tài liệu vào phòng thi. Nội dung gồm các câu hỏi và bài tập tương tự như các câu hỏi và bài tập về nhà.

TÀI LIỆU THAM KHẢO

- **Giáo trình Hệ điều hành HUTECH.**
- Abraham Silberschatz, Peter Baer Galvin, Greg Gagne (2013), Operating System Concepts - 9th edition, ISBN: 978-1-118-06333-0



BÀI 1 : TỔNG QUAN

1.1. Hệ điều hành là gì?

1.2. Cấu trúc phần cứng.

1.3. Đa chương và Chia sẽ thời gian.

1.4. Hoạt động của Hệ điều hành.

1.1 Hệ điều hành là gì?

Giới thiệu -1

Phần mềm

- Ứng dụng

- Hệ thống

Windows XP, 7, 8, ...

Phần cứng

- Thiết bị xử lý và lưu trữ

- Thiết bị xuất (monitor)

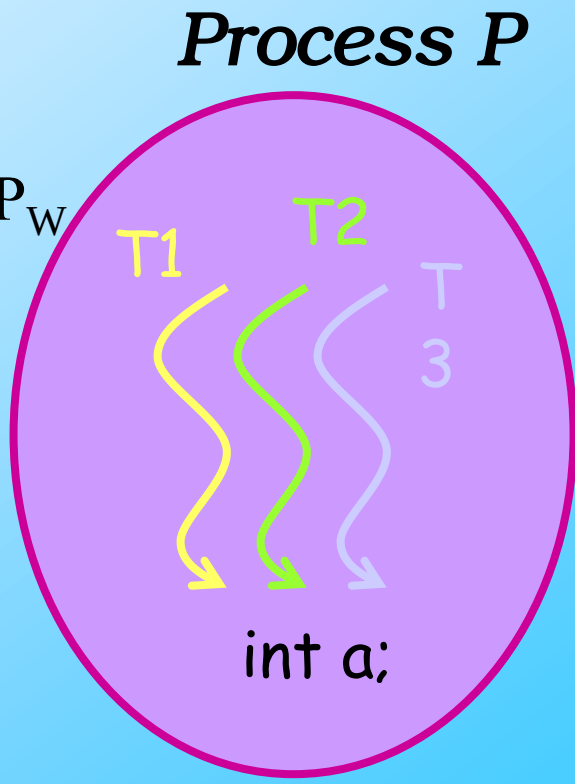
- Thiết bị nhập liệu

Giới thiệu -2

- Hệ điều hành
  - ***Là 1 chương trình*** quản lý phần cứng máy tính
  - ***Trung gian*** giữa người dùng và phần cứng máy tính
  - ***Cung cấp môi trường*** cho các ứng dụng khác thực thi
- Hệ điều hành mạng
  - Là 1 hệ điều hành
  - Cung cấp những khả năng cần thiết để kết nối mạng
  - VD: WinXP, Win 2000 server

Giới thiệu -3

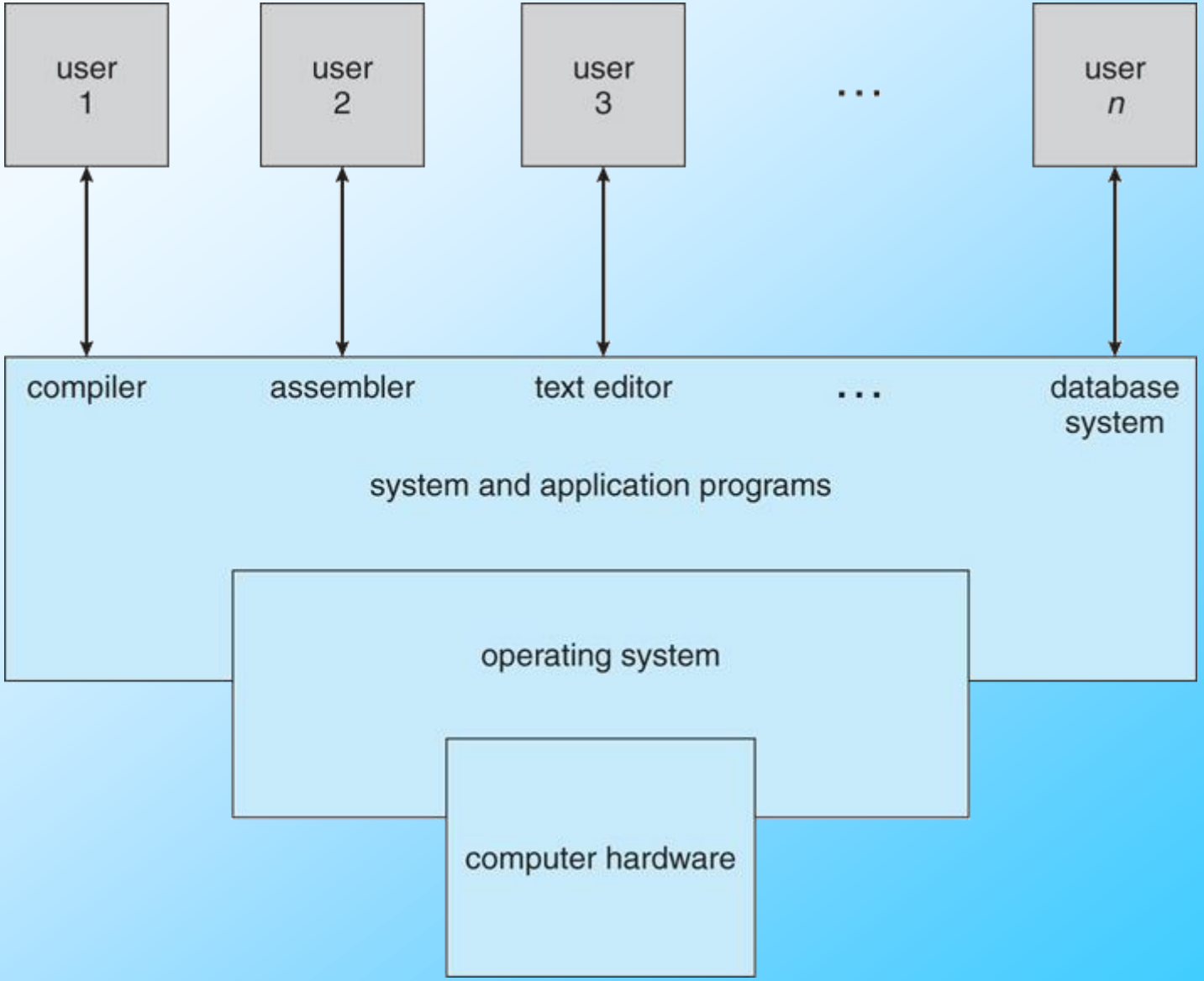
- **Tiến trình (Process)**
  - Chương trình đang thực thi trên máy
  - VD: mở 1 file word ➔ tạo ra 1 tiến trình P_W
- **Tiểu trình (thread)**
  - Một dòng xử lý trong 1 tiến trình
  - Một tiến trình có 1 hay nhiều tiểu trình
  - VD: trong tiến trình P_W
    - Luồng nhận thao tác của người dùng
    - Luồng kiểm tra lỗi
    - …

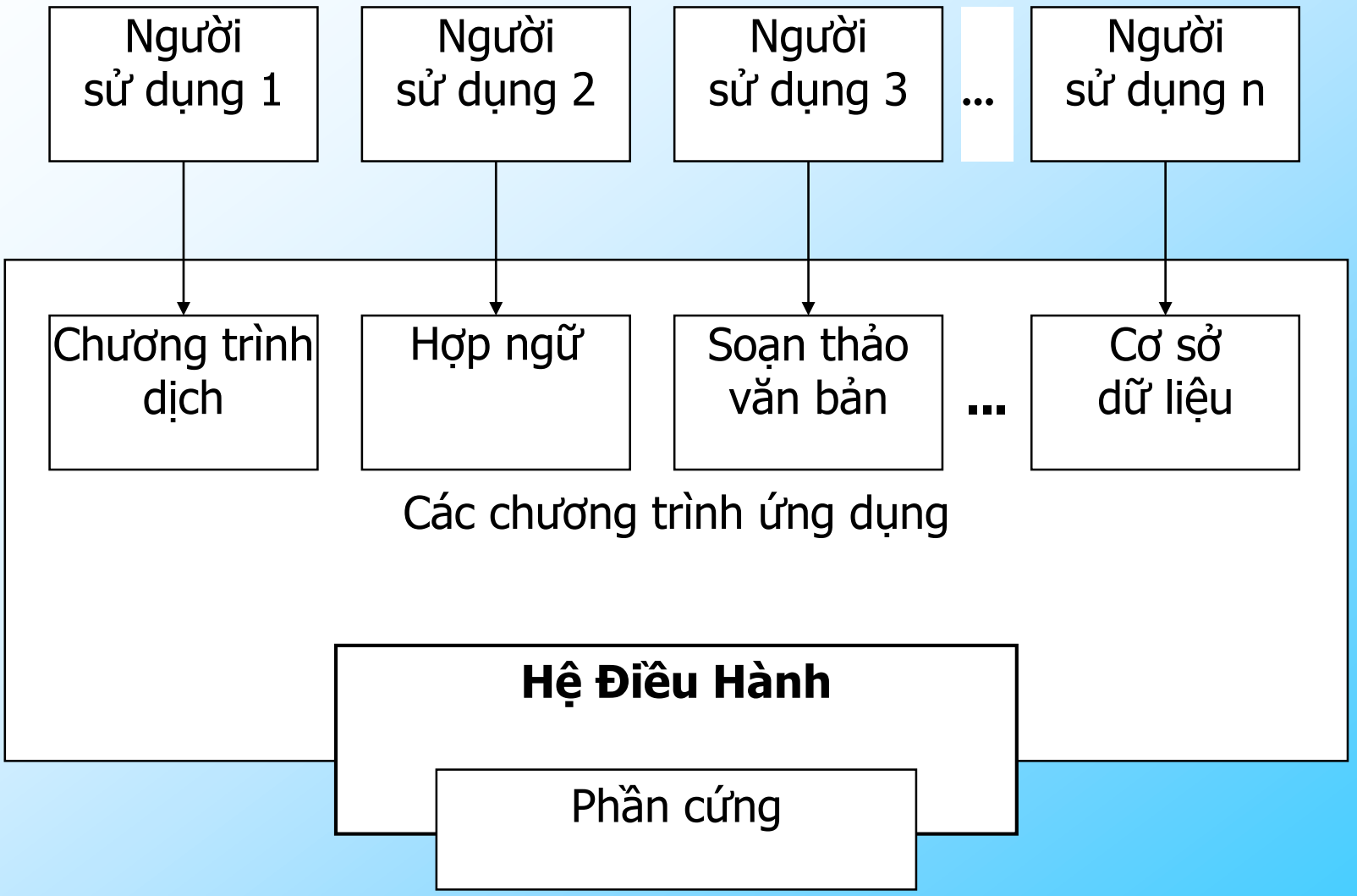


4 thành phần Hệ thống máy tính

- Hệ thống máy tính có thể được chia thành bốn thành phần:
  - phần cứng,
  - hệ điều hành,
  - các chương trình ứng dụng và chương trình hệ thống ngoài hệ điều hành,
  - người dùng.

4 thành phần Hệ thống máy tính

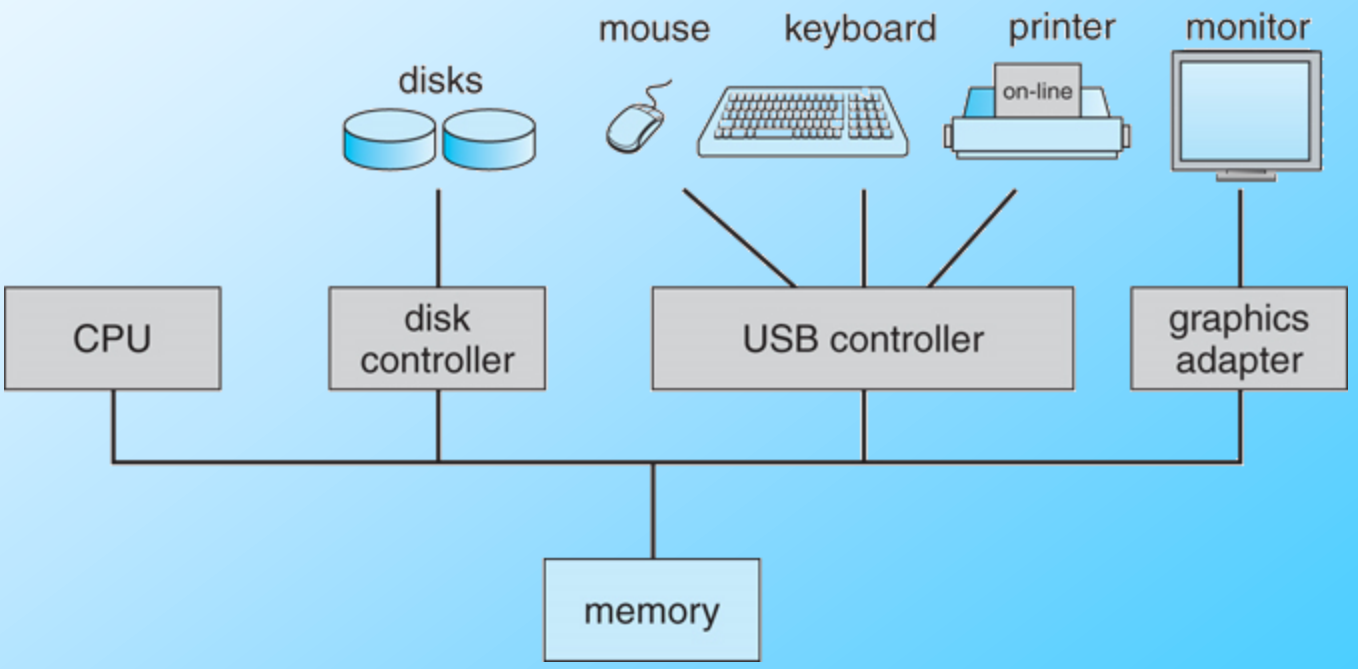



Người sử dụng 1
Người sử dụng 2
Người sử dụng 3
...
Người sử dụng n
Chương trình dịch
Hợp ngữ
Soạn thảo văn bản
...
Cơ sở dữ liệu
Các chương trình ứng dụng
Hệ Điều Hành
Phần cứng

Khái niệm Hệ điều hành

- Hệ điều hành là một bộ chương trình liên quan mật thiết đến phần cứng, có các chức năng chủ yếu sau :
  - Cung cấp môi trường làm việc cho phép người dùng thực hiện và phát triển các chương trình máy tính một cách thuận tiện, hiệu quả.
  - Phân bổ tài nguyên máy tính cho các chương trình và người dùng đang hoạt động một cách công bằng và hiệu quả nhất.
  - Điều khiển, giám sát các thiết bị I/O và các chương trình người dùng, đảm bảo an ninh hệ thống.

1.2 Cấu trúc Phần cứng



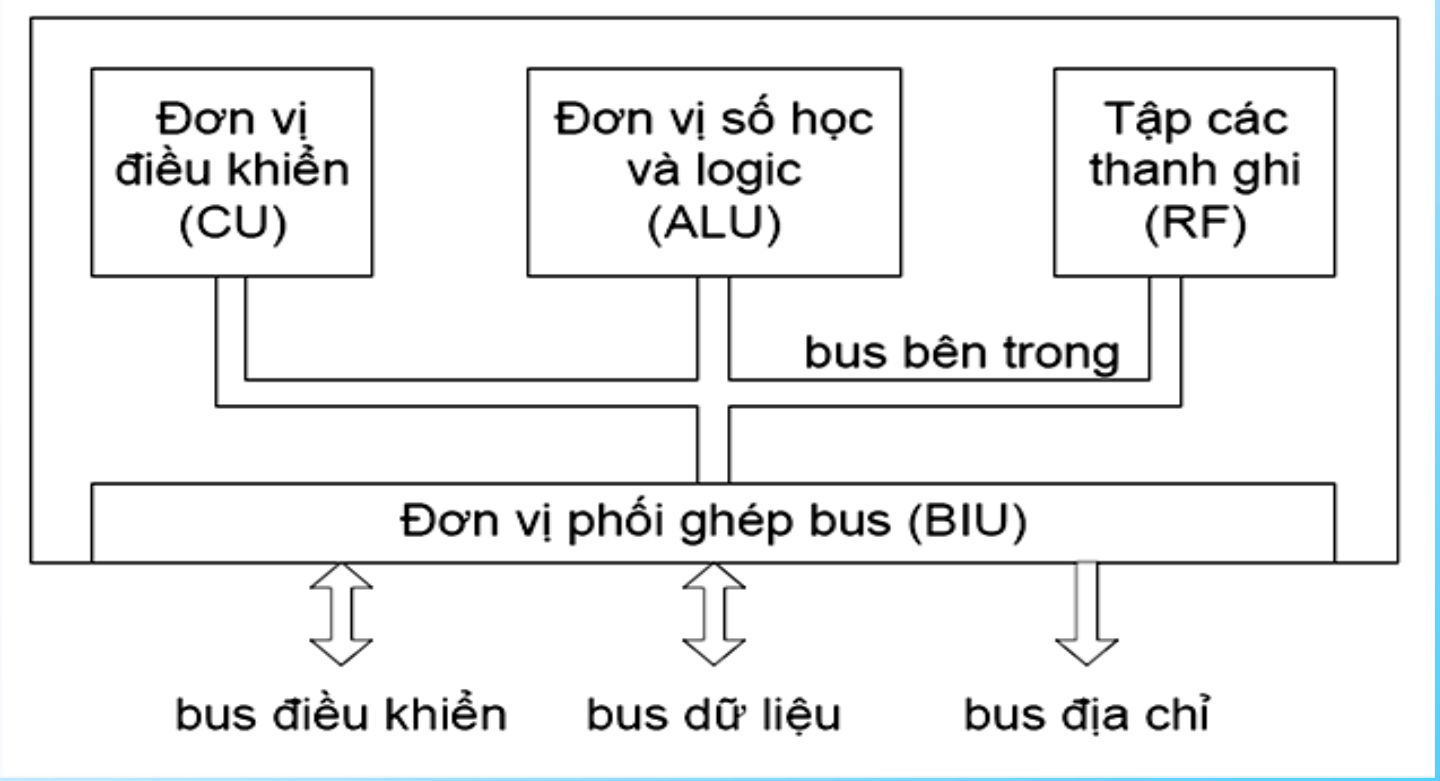
1.2.1 Bộ xử lý trung ương (CPU)

- Đa số là hệ thống một CPU.
- Ngoài ra còn có hệ thống nhiều CPU nhằm:
  - Tăng thông lượng
  - Tiết kiệm về quy mô
  - Tăng độ tin cậy

Bộ xử lý trung ương (CPU)

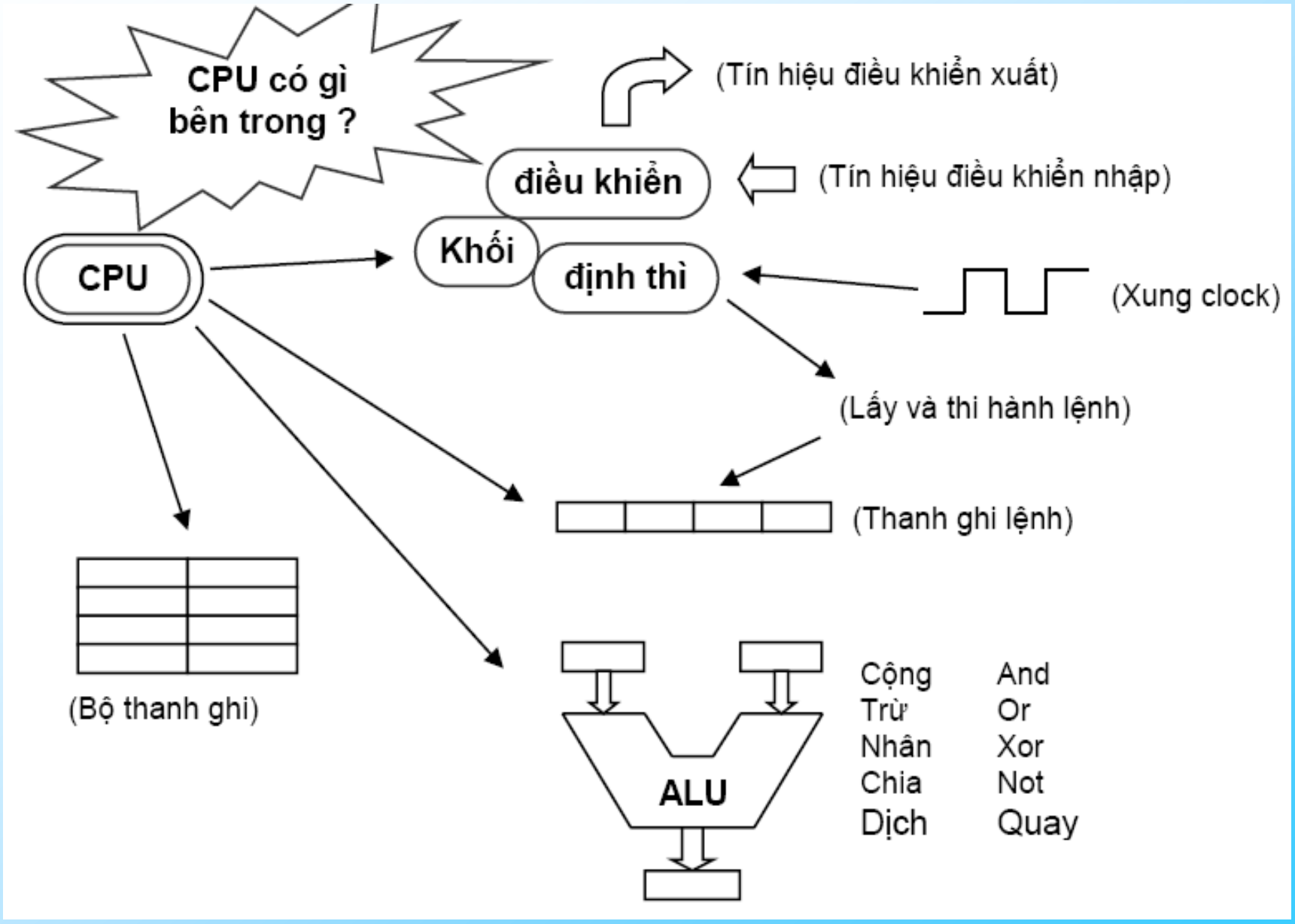
- Chức năng
  - điều khiển hoạt động của máy tính
  - xử lý dữ liệu
- Nguyên tắc hoạt động cơ bản:
  - CPU hoạt động theo chương trình nằm trong bộ nhớ chính.

Các thành phần cơ bản của CPU



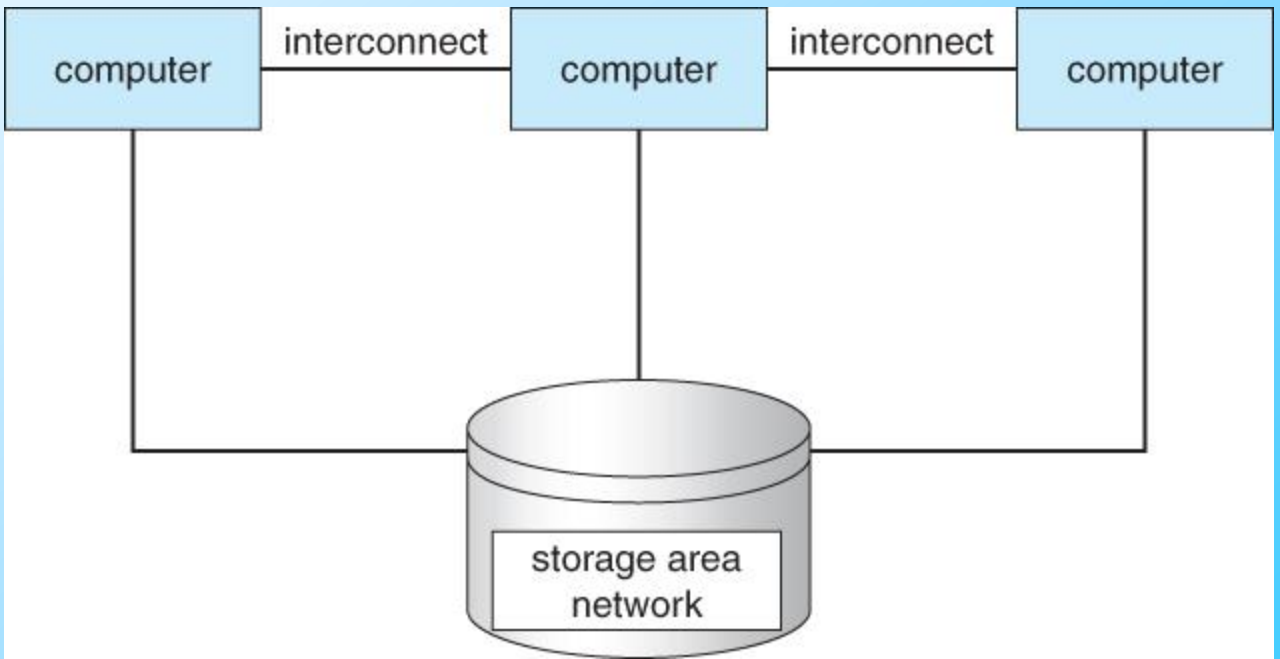
Các thành phần cơ bản của CPU

- **Đơn vị điều khiển** (Control Unit – CU):
  - điều khiển hoạt động của máy tính theo chương trình đã định sẵn.
- **Đơn vị số học và logic** (Arithmetic and Logic Unit – ALU):
  - thực hiện các phép toán số học và các phép toán logic trên các dữ liệu cụ thể.
- **Tập thanh ghi** (Register File - RF):
  - lưu giữ các thông tin tạm thời phục vụ cho hoạt động của CPU.
- **Đơn vị nối ghép bus** (Bus interface Unit - BIU):
  - kết nối và trao đổi thông tin giữa bus bên trong (internal bus) và bus bên ngoài (external bus)


CPU có gì
bên trong ?
(Tín hiệu điều khiển xuất)
điều khiển
(Tín hiệu điều khiển nhập)
Khối
định thì
CPU
(Xung clock)
(Lấy và thi hành lệnh)
(Thanh ghi lệnh)
(Bộ thanh ghi)
ALU
Cộng
Trừ
Nhân
Chia
Dịch
And
Or
Xor
Not
Quay

1.2.2 Hệ thống nhóm (Clustered System)

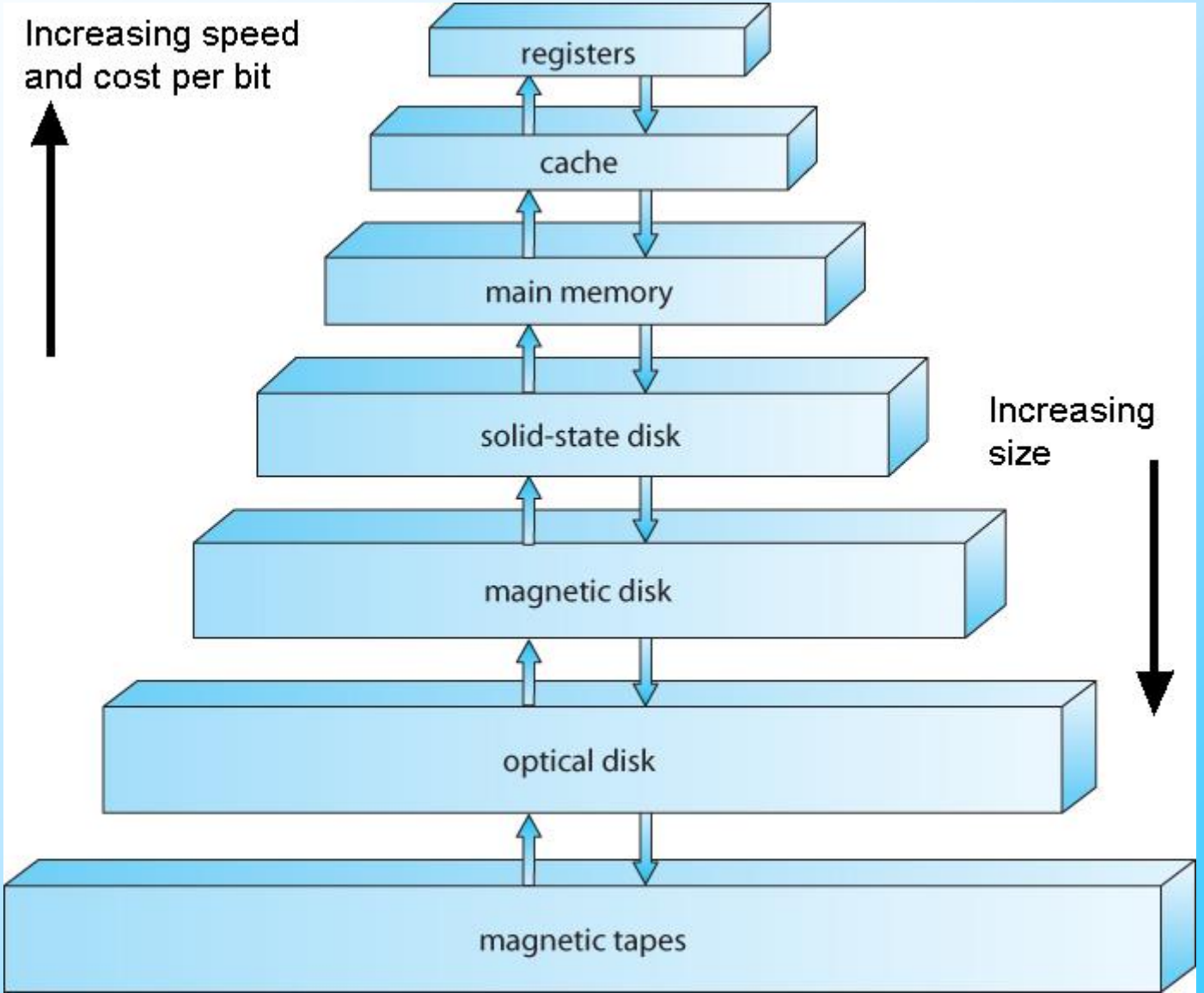
- Hệ thống nhóm tập hợp nhiều CPU để thực hiện công việc tính toán.
- Được sử dụng để cung cấp các dịch vụ có tính sẵn sàng cao.



1.2.3 Cấu trúc lưu trữ (Storage Structure)

- Bao gồm các thanh ghi, bộ nhớ chính, các ổ đĩa từ tính và còn có thể bao gồm bộ nhớ cache, CD-ROM, flash, băng từ, v.v.
- Mỗi hệ thống lưu trữ cung cấp các chức năng cơ bản của quá trình lưu trữ một dữ kiện và giữ dữ kiện đó cho đến khi nó được lấy ra sau một thời gian.
- Sự khác biệt chính giữa các hệ thống lưu trữ khác nhau nằm ở tốc độ, chi phí, quy mô và tính bay hơi.

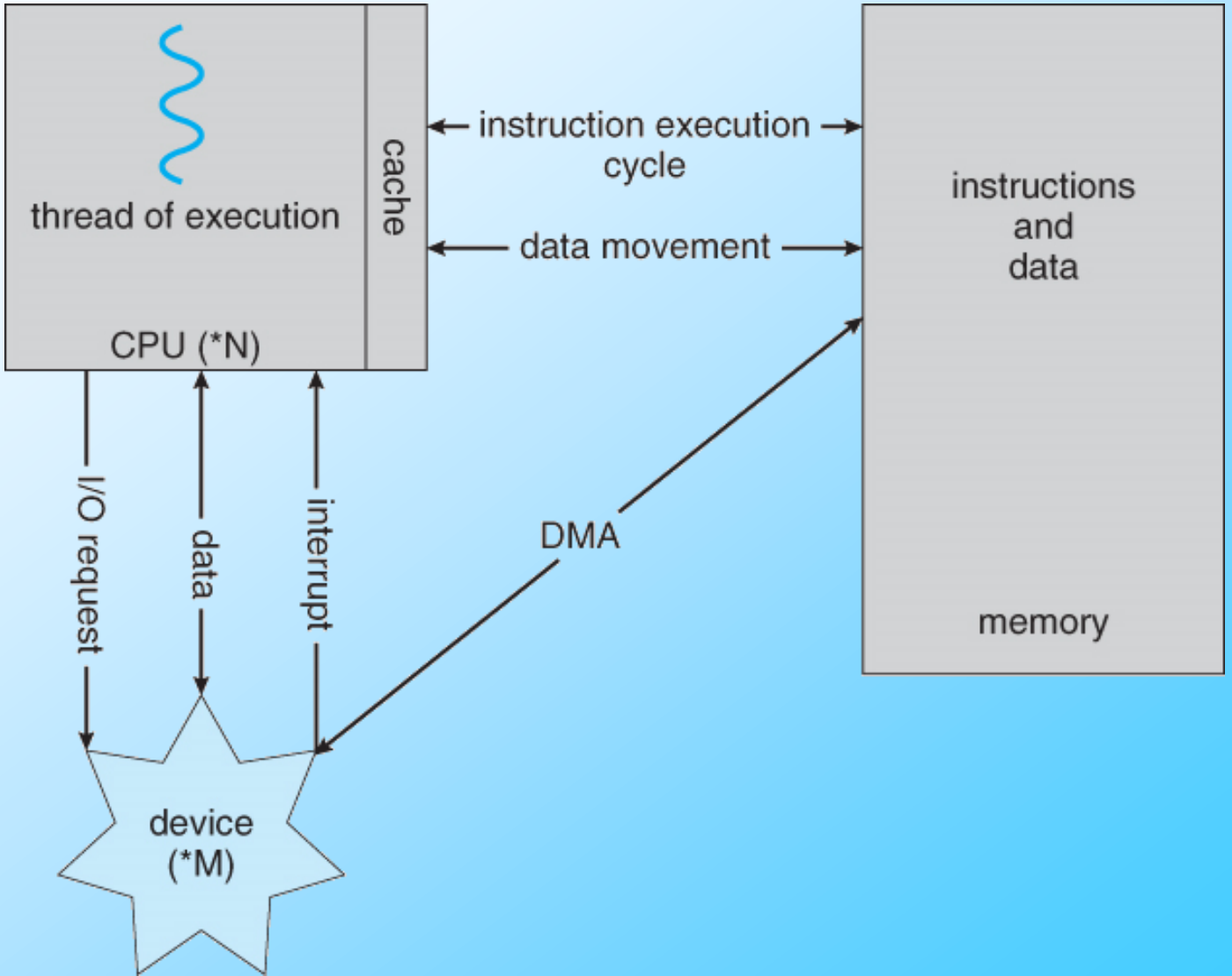
1.2.3 Cấu trúc lưu trữ (Storage Structure)



1.2.4 Cấu trúc I/O

- Thiết bị lưu trữ là một trong nhiều loại thiết bị I/O trong máy tính.
- Một máy tính thường có nhiều bộ điều khiển thiết bị - device controller (tương ứng nhiều nhiều trình điều khiển thiết bị - device driver).

1.2.4 Cấu trúc I/O



1.3
Đa chương và Chia sẽ Thời gian

Đa chương (Multiprogramming)

CPUs vẫn sẽ nhàn rỗi mỗi khi chương trình thực thi cần giao tiếp với thiết bị ngoại vi

- Đọc dữ liệu từ đĩa

Hệ thống đa chương theo lô (Multiprogrammed batch systems) ra đời

- Nạp nhiều chương trình vào đĩa cùng một thời gian (sau này là vào bộ nhớ)
- Chuyển sang công việc kế tiếp nếu công việc hiện thời đang thực hiện lệnh I/O
- Thiết bị ngoại vi thường chậm hơn trên đĩa (hay bộ nhớ)
- Đồng thời thực hiện I/O của chương trình này và tính toán cho chương trình khác
- Thiết bị ngoại vi phải là bất đồng bộ
- Phải biết khi nào công việc I/O xong: ngắt vs. polling

Tăng khả năng phục vụ của hệ thống, có thể tốn nhiều thời gian hơn để phản hồi

- Khi nào thì tốt cho thời gian phản hồi? Khi nào thì xấu cho thời gian phản hồi?

Chia sẻ thời gian (Time-Sharing)

- Vấn đề
  - Làm sao chia sẻ cùng một máy tính (lúc đó rất đắt) giữa nhiều người dùng và vẫn duy trì giao diện giao tiếp với người dùng?
- Chia sẻ thời gian
  - Nối nhiều thiết bị đầu cuối đầu cuối đến một máy tính
  - Điều phối sử dụng máy tính cho nhiều người dùng
  - Chuyển đổi phục vụ giữa các chương trình người dùng sao cho đủ nhanh để người sử dụng có thể tương tác với chương trình trong khi chúng đang chạy (máy phải đủ nhanh để tạo cảm giác mỗi người dùng đang dùng máy riêng của mình)

Chia sẻ thời gian (Time-Sharing)

- Định thời công việc (job scheduling)
- Quản lý bộ nhớ (Memory Management)
  - Các công việc được hoán chuyển giữa bộ nhớ chính và đĩa
  - Virtual memory: cho phép một công việc có thể được thực thi mà không cần phải nạp hoàn toàn vào bộ nhớ chính
- Quản lý các process (Process Management)
  - Định thời CPU (CPU scheduling)
  - Đồng bộ các công việc (synchronization)
  - Tương tác giữa các công việc (process communication)
  - Tránh Deadlock
- Quản lý hệ thống file, hệ thống lưu trữ (disk management)
- Phân bổ các thiết bị tài nguyên
- Cơ chế bảo vệ (protection)

1.4
Hoạt động của Hệ điều hành

1.4. Hoạt động Hệ điều hành

- Chế độ hoạt động kép: Để đảm bảo hệ điều hành chạy tốt, phải có khả năng phân biệt giữa việc thực thi mã lệnh của hệ điều hành và việc thực thi mã lệnh của người dùng. Do vậy cần hai chế độ riêng biệt của hoạt động: chế độ người dùng (user mode) và chế độ hạt nhân (kernel mode, còn gọi là chế độ giám sát, chế độ hệ thống, hoặc chế độ đặc quyền).
- Timer: Bộ đếm thời gian được dùng để ngăn chặn một chương trình người dùng chạy quá lâu.

CÂU HỎI ÔN TẬP BÀI 1

1. Hãy cho biết khái niệm hệ điều hành.
2. Hãy vẽ sơ đồ và mô tả 4 thành phần của hệ thống máy tính.
3. Hệ thống đa chương là gì?
4. Hệ thống chia sẻ thời gian là gì?
5. Sự khác nhau giữa hệ thống đa chương và hệ thống chia sẻ thời gian? Ưu điểm của hệ thống chia sẻ thời gian so với hệ thống đa chương là gì?
6. Hệ thống chia sẻ thời gian phải giải quyết được gì?

BÀI 2 : CẤU TRÚC HỆ ĐIỀU HÀNH

2.1. Các dịch vụ lõi

2.2. Giao diện người dùng

2.3. Lời gọi hệ thống

2.4. Các kiểu cấu trúc HĐH

2.5 Máy ảo

2.6 Khởi động Hệ thống

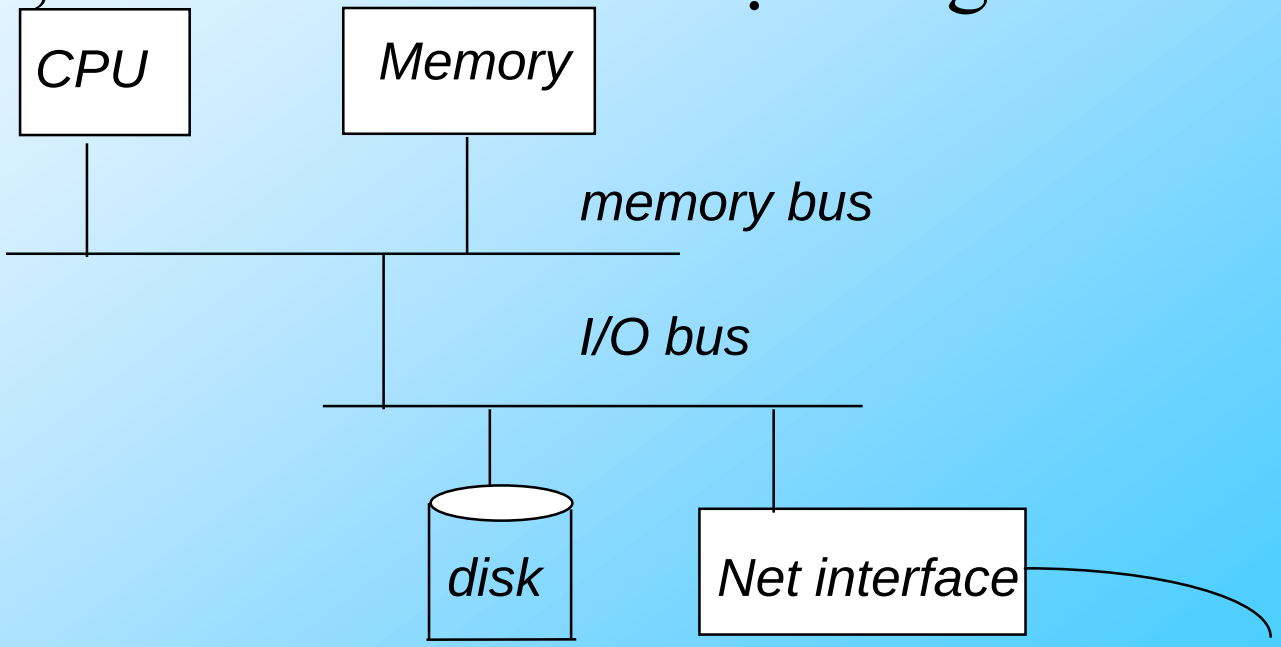
2.1. Các dịch vụ lõi

Các dịch vụ lõi

- Các dịch vụ lõi nằm ở hạt nhân của hệ điều hành. Bao gồm:
  - Thực thi chương trình
  - Hoạt động I/O.
  - Thao tác hệ thống tập tin.
  - Truyền thông
  - Phát hiện lỗi
  - Phân bổ tài nguyên
  - Kế toán
  - Bảo vệ an ninh.

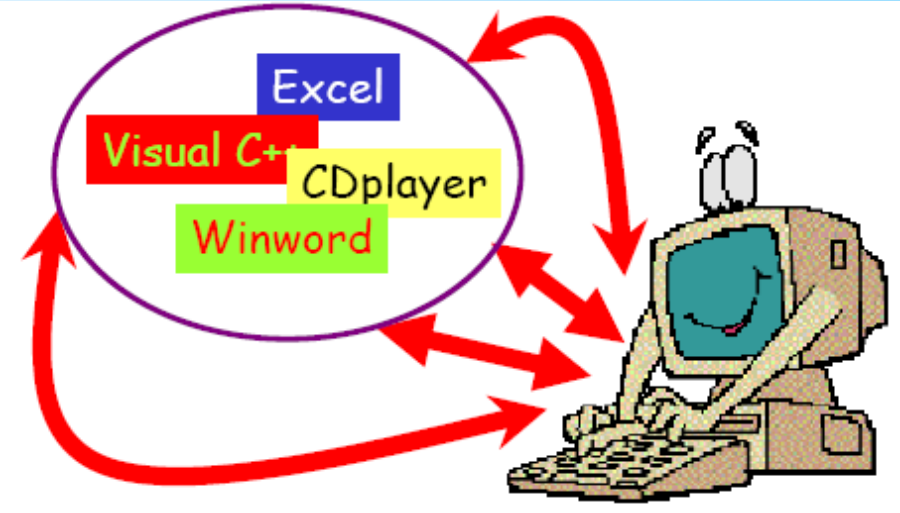
Các dịch vụ lõi

- Sáu nhóm dịch vụ đầu nhằm cung cấp môi trường làm việc thuận tiện cho người dùng.
- Ba nhóm sau nhằm đảm bảo sự hoạt động hiệu quả, an toàn của chính hệ thống.



Quản lý tiến trình

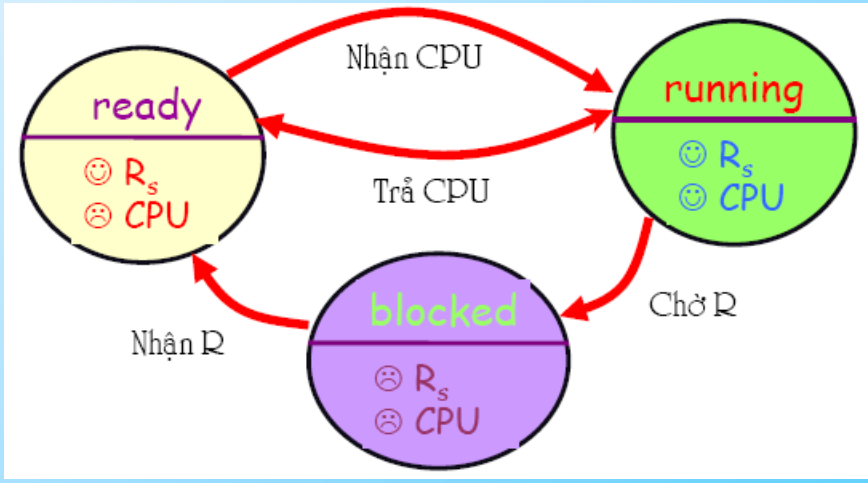
- Tiến trình là chương trình đang thực thi.
- Một tiến trình cần sử dụng các tài nguyên: CPU, bộ nhớ, tập tin, thiết bị nhập xuất để hoàn tất công việc của nó
- Hệ thống đa chương: sẽ có nhiều tiến trình chạy cùng lúc.



- Số lượng tài nguyên << số lượng tiến trình chạy cùng lúc !!!
  → Tranh chấp

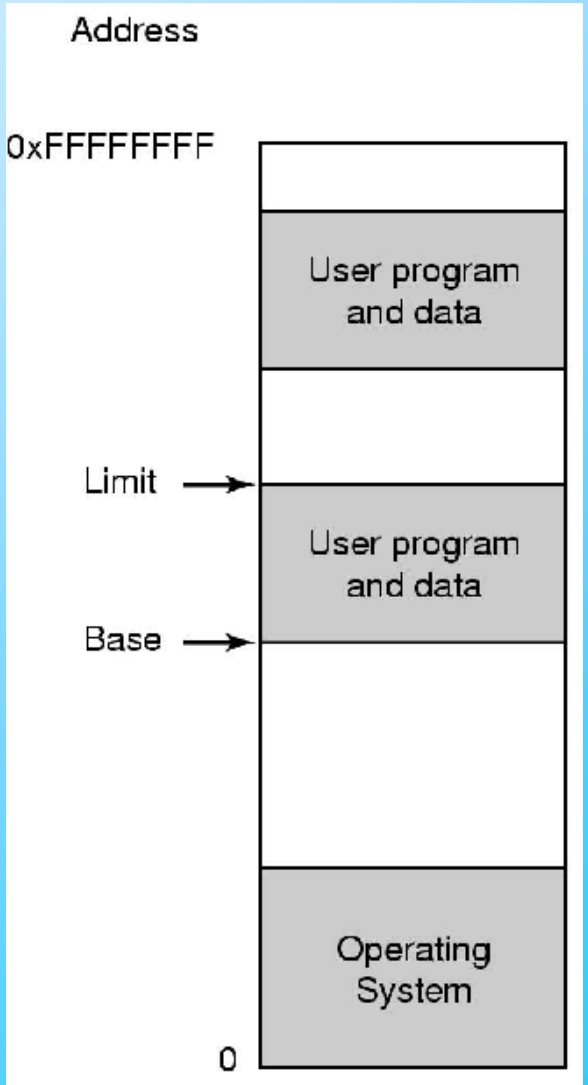
Quản lý tiến trình(2/2)

- Vai trò của HĐH trong việc quản lý tiến trình
  - Tạo, hủy, tạm dừng và thực hiện tiếp tiến trình
  - Quản lý trạng thái các tiến trình
  - Điều phối tiến trình: FIFO, Round Robin, SJF, …
  - Cung cấp cơ chế đồng bộ tiến trình
    - Độc quyền truy xuất
    - Phối hợp hoạt động
  - Cung cấp cơ chế kiểm soát deadlock
  - Cung cấp cách thức trao đổi thông tin giữa các tiến trình
    - Chia sẻ tài nguyên dùng chung
    - Trao đổi thông điệp



Quản lý bộ nhớ

- Mọi chương trình (mã nguồn + dữ liệu) cần được nạp vào bộ nhớ chính để thi hành
- Nhiều tiến trình chạy đồng thời → Quản lý bộ nhớ sao cho tối ưu việc tận dụng CPU và đáp ứng kịp thời cho người sử dụng
- Vai trò của HĐH trong việc quản lý bộ nhớ:
  - Tổ chức cấp phát, thu hồi bộ nhớ khi cần thiết
    - Mô hình cấp phát (liên tục, không liên tục)
    - Quản lý không gian địa chỉ của tiến trình
  - Quản lý bộ nhớ ảo
    - Quyết định chương trình/ một phần chương trình *nào* được nạp vào/ ra bộ nhớ *như thế nào*

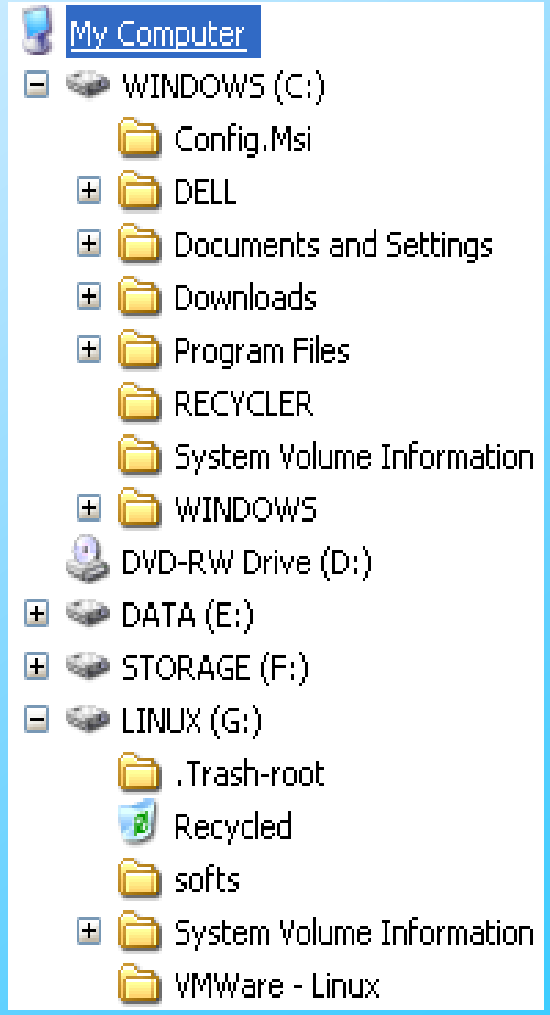


Quản lý nhập xuất

- Hệ thống quản lý nhập xuất chịu trách nhiệm:
  - Tạo môi trường giao tiếp đơn giản, đồng nhất với các thiết bị nhập xuất
    - Trình điều khiển thiết bị: che đi đặc thù phức tạp của các thiết bị nhập xuất đối với người sử dụng
  - Quản lý hiệu quả nhập xuất:
    - Điều phối yêu cầu nhập xuất
    - Tổ chức lưu trữ đệm (buffering, spooling,…)
    - Quản lý lỗi

Quản lý tập tin

- Nhiều loại thiết bị lưu trữ đa dạng về tốc độ truy xuất, đơn vị lưu trữ, phương thức truy xuất (đĩa cứng, USB, CD,…)
- HĐH cung cấp cái nhìn logic và đồng nhất về việc lưu trữ thông tin – tập tin
- Tập tin thường được tổ chức trong các thư mục
- Vai trò HĐH trong việc quản lý tập tin:
  - Tổ chức tập tin, thư mục trên đĩa
  - Hỗ trợ các thao tác trên tập tin và thư mục
  - Quản lý quyền truy cập
  - Sao lưu dự phòng tập tin trên các thiết bị lưu trữ



Bảo vệ & Bảo mật

- Hệ thống máy tính luôn đứng trước các mối nguy cơ:
  - Khách quan: thiên tai, lỗi sử dụng, lỗi phần cứng, phần mềm.
  - Chủ quan:
    - Tấn công phá hoại: virus, worm, DoS, …
    - Ăn cắp tài nguyên: trojan horses, trap doors, Man-in-the-middle, …

- Bảo vệ (protection) và Bảo mật (security):
  - Kiểm soát quá trình truy xuất tài nguyên của tiến trình/ người dùng
  - Phòng thủ, chống lại các tấn công
- Một số cơ chế:
  - Hoạt động ở 2 chế độ (kernel mode vs. user mode)
  - Sao lưu dự phòng (Backup)
  - Xác thực người dùng (User Authentication)
  - Phân quyền (Authorization), chính sách bảo mật (Policy)
  - Kiểm soát nhật ký (Audit log)

2.2. Giao diện người dung
(User Interface)

Giao diện người dùng

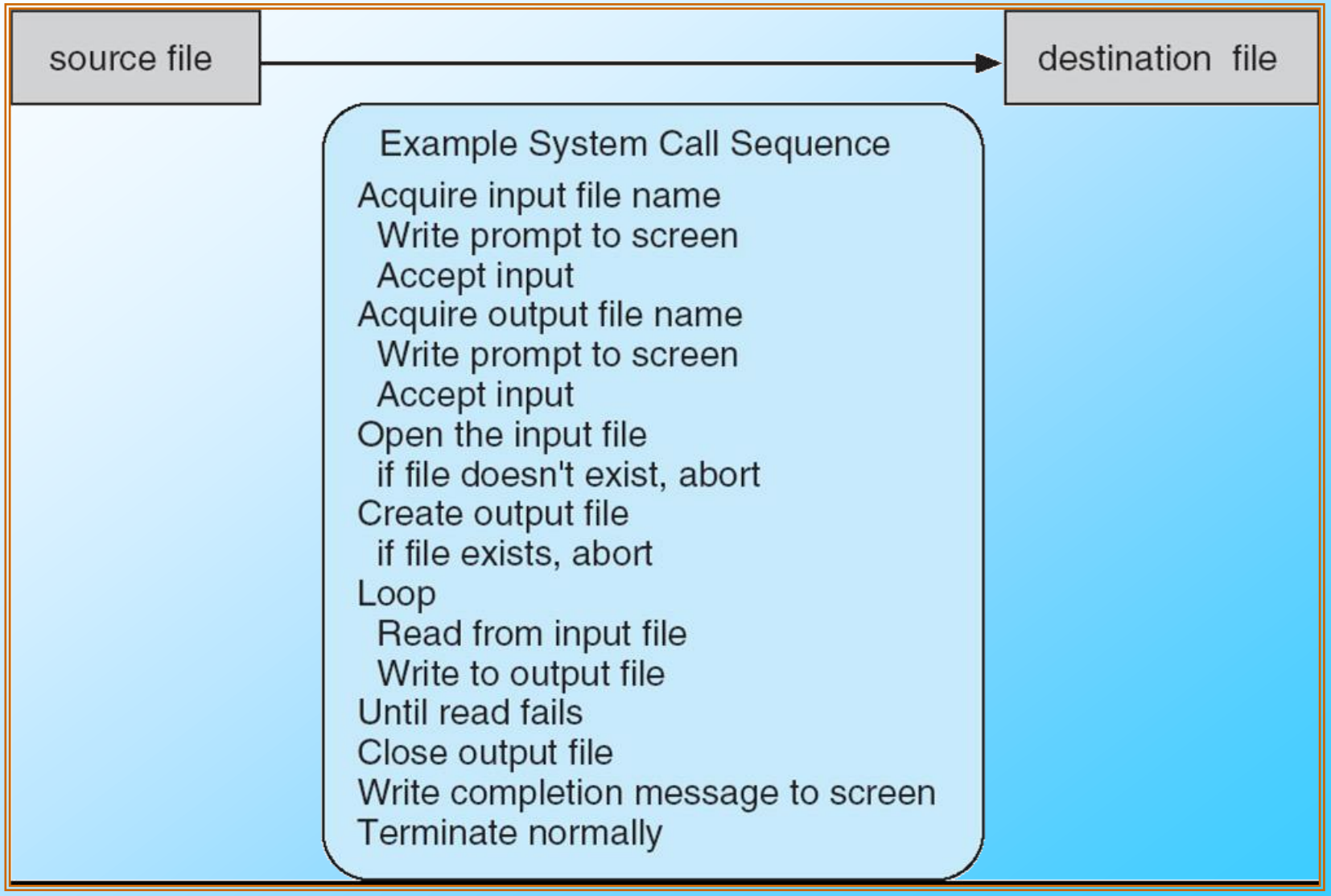
- **Giao diện dòng lệnh** (CLI – command line interface), còn gọi là bộ thông dịch lệnh (command interpreter).
- **Giao diện đồ họa** (GUI - graphical user interface) → phổ biến nhất hiện nay.
- **Giao diện nhóm** (batch interface) : Thường được thể hiện dưới dạng một tập tin văn bản mà mỗi dòng là một lệnh (và tham số) của hệ điều hành.

2.3. Lời gọi hệ thống
(System Call)

Lời gọi hệ thống

- Lời gọi hệ thống là lệnh do Hệ Điều Hành cung cấp dùng để giao tiếp giữa tiến trình và Hệ Điều Hành.
- Lời gọi hệ thống cung cấp một giao diện cho các dịch vụ được cung cấp bởi hệ điều hành. Các chương trình người dùng sử dụng các dịch vụ của hệ điều hành thông qua giao diện là các lời gọi hệ thống. Các lời gọi hệ thống được viết bằng C, C++ hoặc hợp ngữ (Assembler).
- VD chuỗi các lời gọi hệ thống được thực hiện để sao chép nội dung của một tập tin sang một tập tin khác

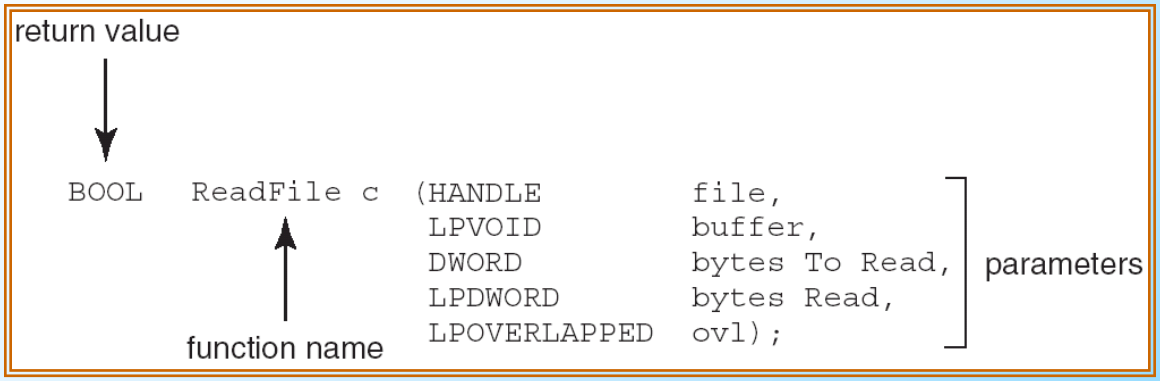
Lời gọi hệ thống

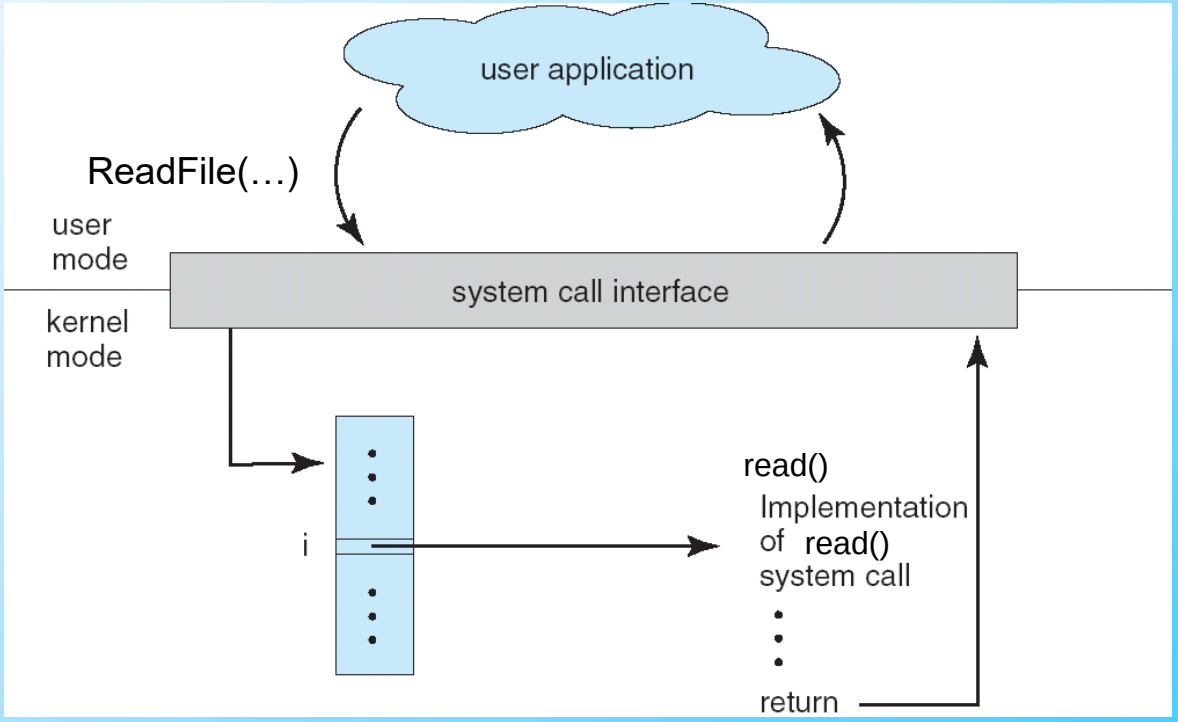


Hàm thư viện lập trình

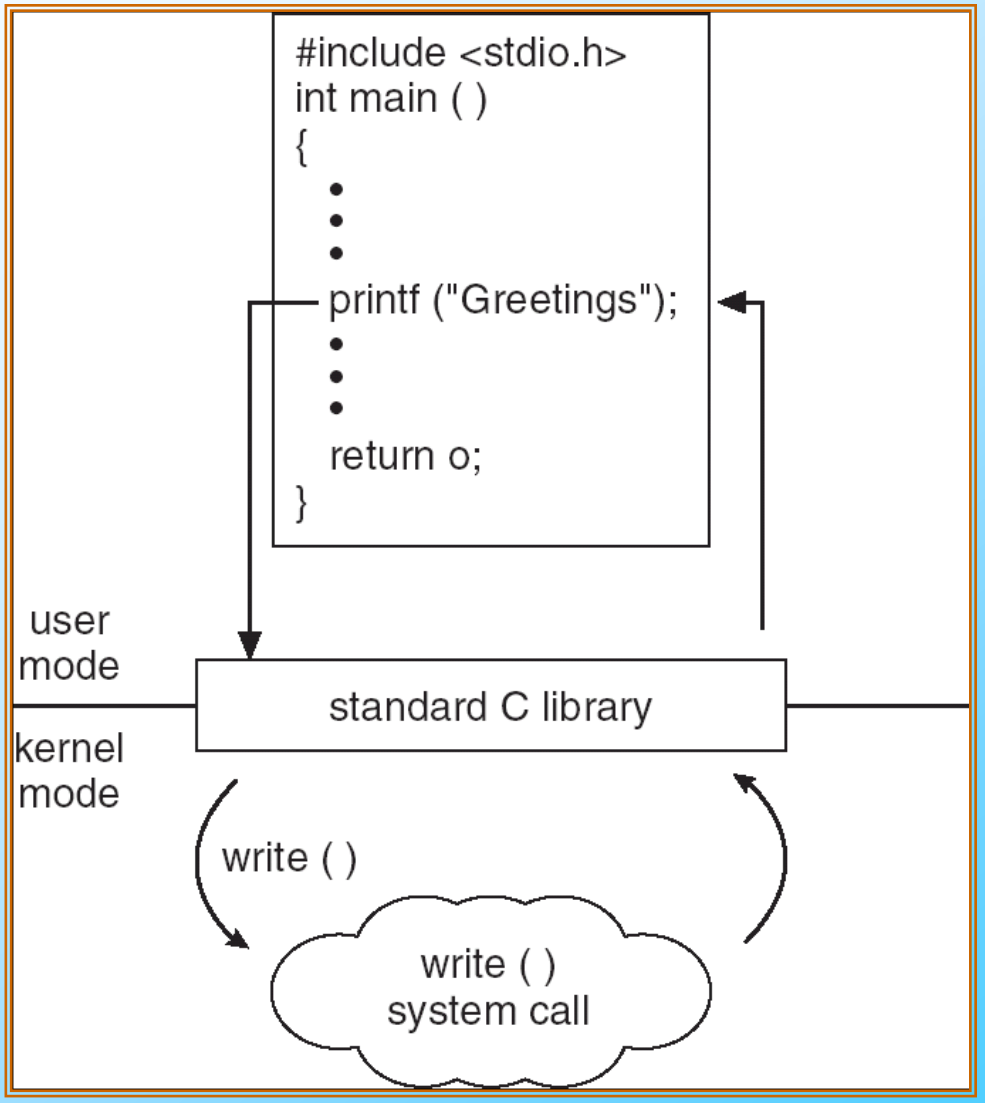
- Thông thường, chương trình người dùng không gọi trực tiếp các lời gọi hệ thống của hệ điều hành.
- Hệ điều hành thường cung cấp bộ thư viện các hàm lập trình, chẳng hạn Win32 API, POSIX API (application programming interface), giúp việc lập trình dễ dàng hơn phải dùng các lời gọi hệ thống.
- Tuy nhiên, thông thường người lập trình thường dùng các hàm thư viện của các ngôn ngữ lập trình như thư viện C, Java,… (do dễ sử dụng hơn các hàm thư viện của hệ điều hành).
- Cho dù là sử dụng hàm ngôn ngữ lập trình hay hàm thư viện hệ điều hành thì cuối cùng cũng sẽ chuyển thành các lời gọi hệ thống tương ứng.

API và Lời gọi hệ thống





Hàm thư viện C và Lời gọi hệ thống



Các loại Lời gọi hệ thống

- Điều khiển tiến trình,
- Quản lý tập tin,
- Quản lý thiết bị,
- Bảo trì thông tin,
- Truyền thông,
- Bảo vệ.

Lời gọi hệ thống (system call)

Có ba phương pháp để chuyển tham số cho HĐH:

+ Chuyển tham số vào thanh ghi.
+ Lưu trữ tham số trong khối hoặc bảng trong bộ nhớ.
+ Dùng stack.

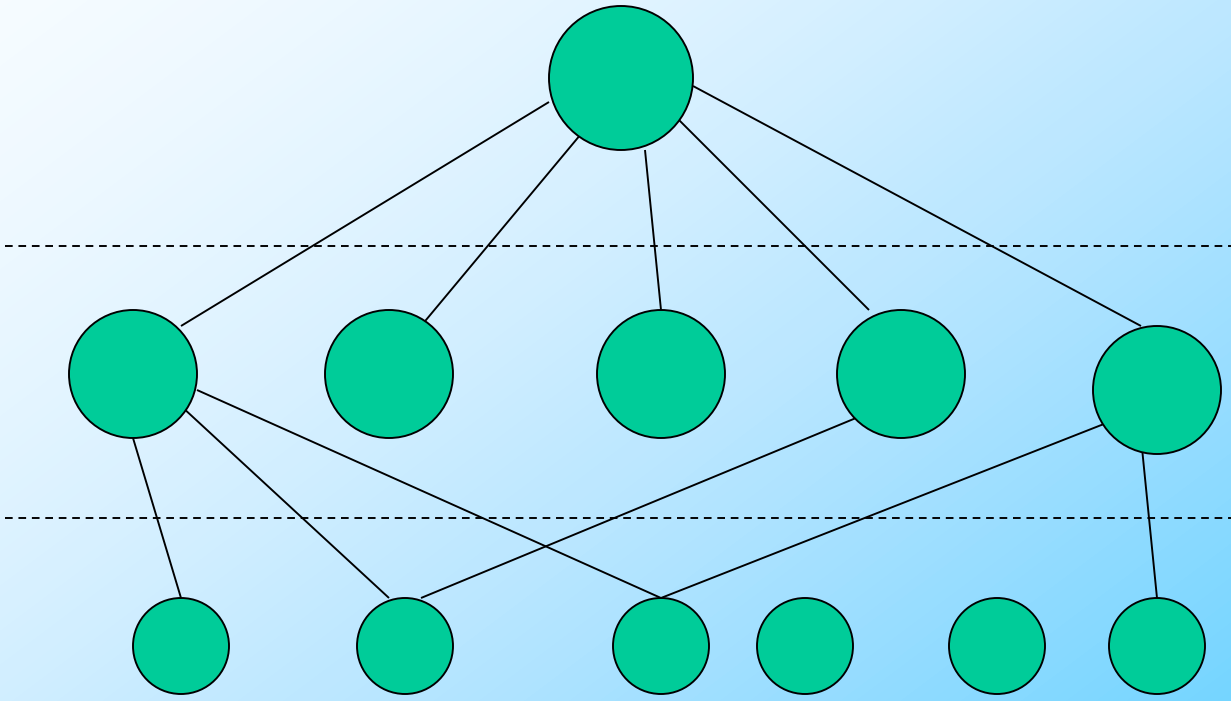
Ví dụ:

UNIX	Win32	Description
fork	CreateProcess	Create a new process
waitpid	WaitForSingleObject	Can wait for a process to exit
execve	(none)	CreateProcess = fork + execve
exit	ExitProcess	Terminate execution
open	CreateFile	Create a file or open an existing file
close	CloseHandle	Close a file
read	ReadFile	Read data from a file
write	WriteFile	Write data to a file
lseek	SetFilePointer	Move the file pointer
stat	GetFileAttributesEx	Get various file attributes
mkdir	CreateDirectory	Create a new directory
rmdir	RemoveDirectory	Remove an empty directory
link	(none)	Win32 does not support links
unlink	DeleteFile	Destroy an existing file
mount	(none)	Win32 does not support mount
umount	(none)	Win32 does not support mount
chdir	SetCurrentDirectory	Change the current working directory
chmod	(none)	Win32 does not support security (although NT does)
kill	(none)	Win32 does not support signals
time	GetLocalTime	Get the current time

UNIX/Win32 API (Application Programming Interface)

2.4. Các kiểu kiến trúc HĐH

Kiến trúc đơn giản (1/2)

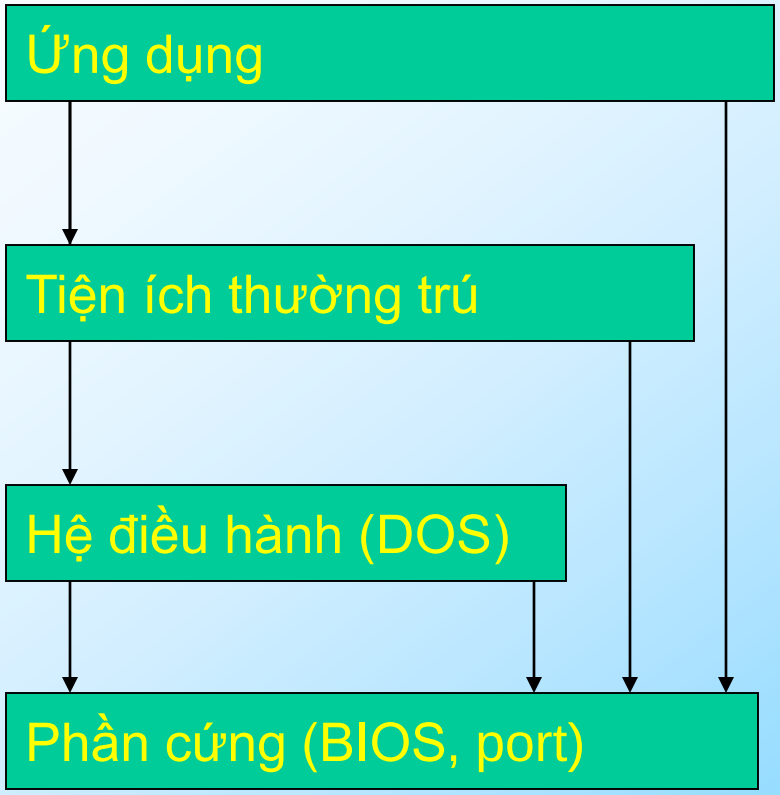


Thủ tục chính

Thủ tục dịch vụ

Thủ tục tiện ích

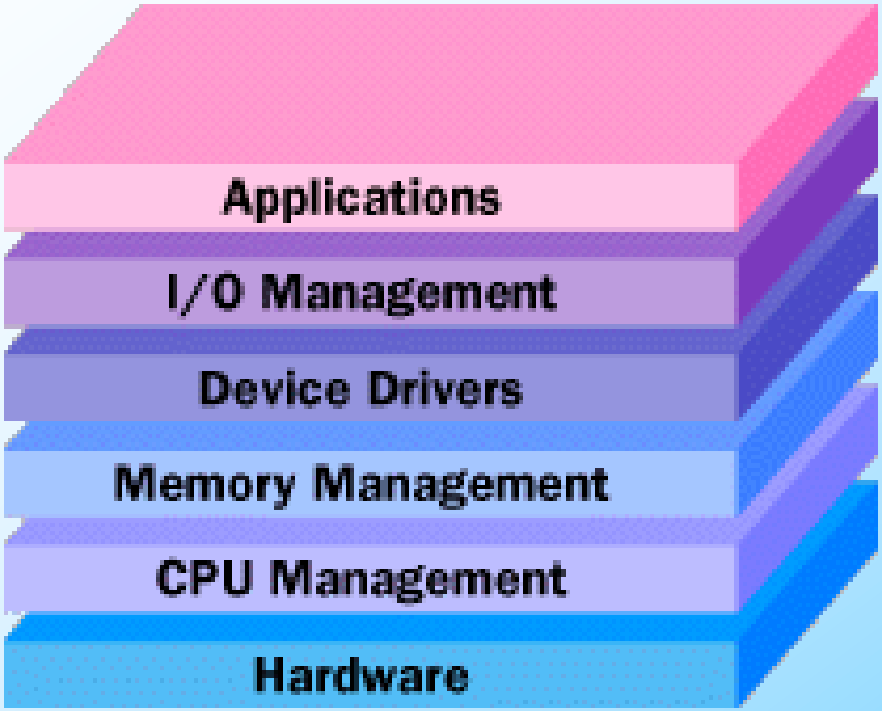
Kiến trúc đơn giản(2/2)



Ví dụ với HĐH DOS

- Ví dụ điển hình: HĐH MS-DOS
- HĐH chỉ làm một số nhiệm vụ quản lý khá đơn giản và cung cấp thêm một số dịch vụ.
- HĐH = Thư viện hàm.
- Ứng dụng của người dùng vẫn có thể truy cập trực tiếp phần cứng thông qua BIOS, cổng phần cứng
- **Không hỗ trợ đa nhiệm.**
- **Đánh giá: khi chương trình treo?**

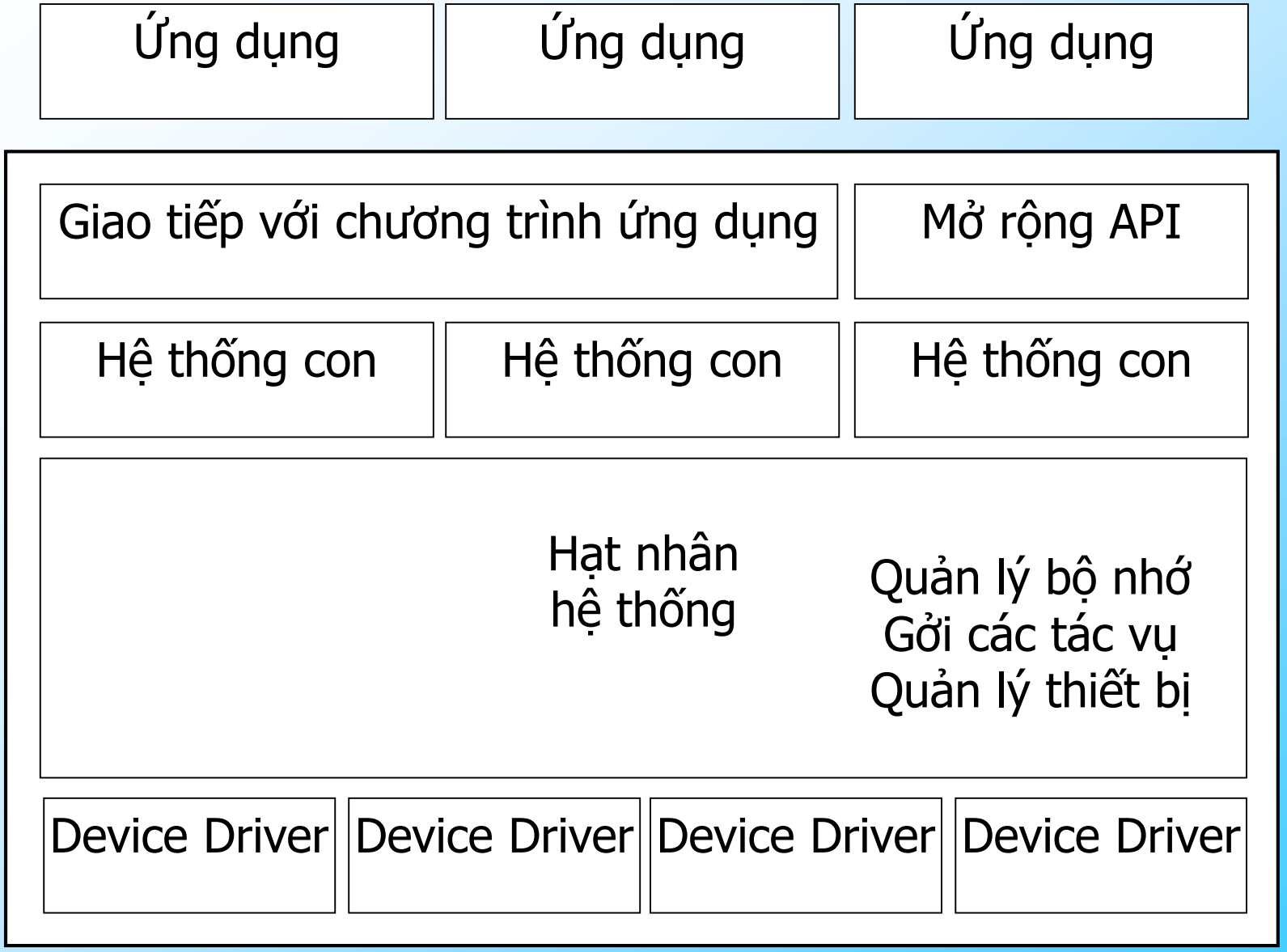
Kiến trúc phân lớp (1/3)



- HĐH phân thành nhiều lớp. Mỗi lớp phụ trách 1 chức năng đặc thù.
- Lớp bên trên sử dụng chức năng do các lớp bên dưới cung cấp.

Lớp 5: Chương trình ứng dụng
Lớp 4: Quản lý bộ đệm cho t/bị xuất nhập
Lớp 3:Trình quản lý thao tác console
Lớp 2: Quản lý bộ nhớ
Lớp 1: Điều phối CPU
Lớp 0: Phần cứng

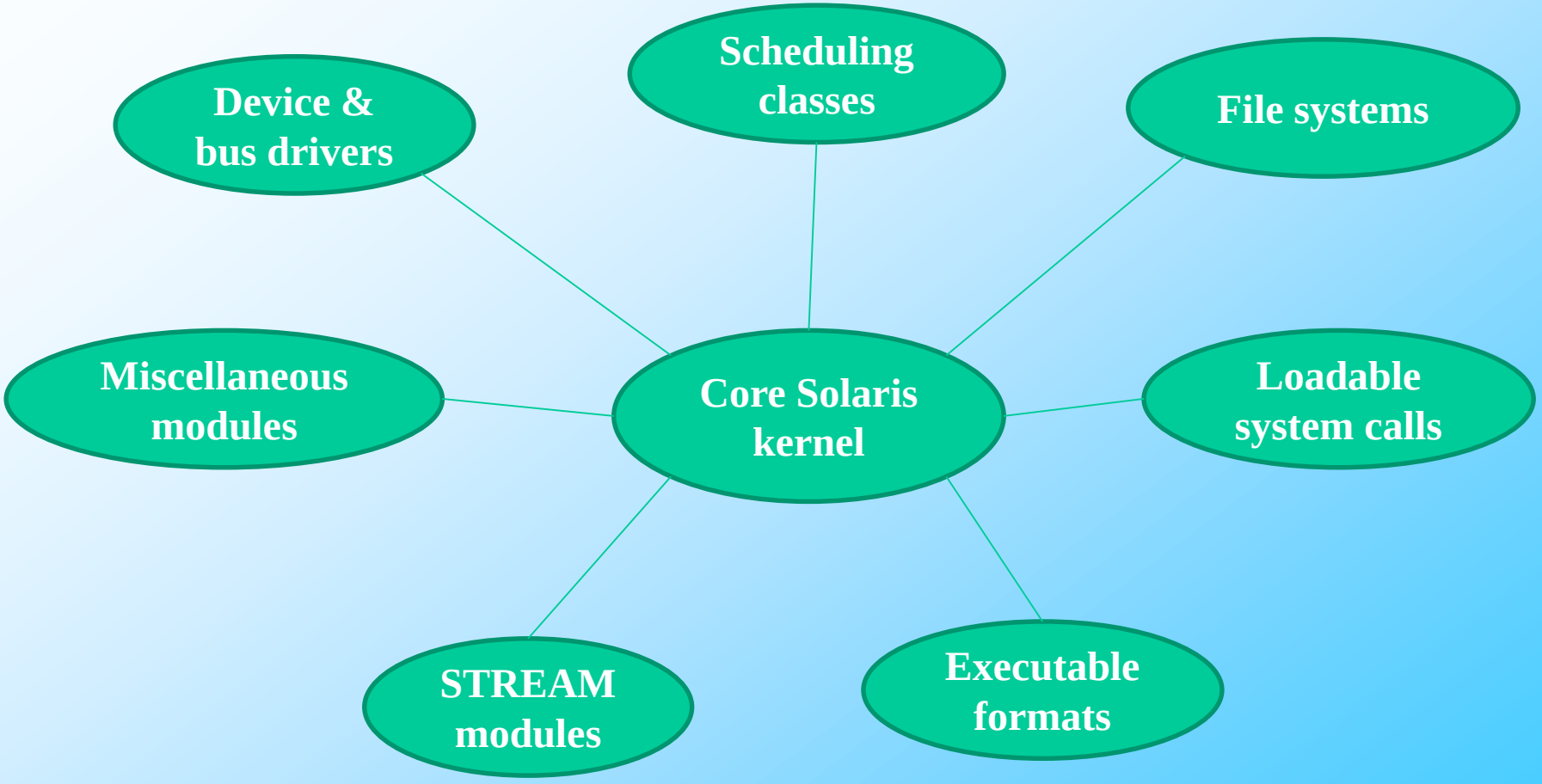
Kiến trúc phân lớp (2/3)



Kiến trúc phân lớp(3/3)

- Ưu điểm:
  - đơn giản việc tìm lỗi và kiểm chứng hệ thống
  - Đơn giản trong việc thiết kế và cài đặt
- Khuyết điểm:
  - Bao nhiêu lớp là đủ ?, thứ tự sắp xếp các lớp ?
  - Kém hiệu quả do 1 lời gọi thủ tục có thể kích hoạt lan truyền các thủ tục ở các lớp bên trong => chi phí truyền thông số, chuyển đổi ngữ cảnh tăng

Kiến trúc modules



Ví dụ kiến trúc của HĐH Solaris

Kiến trúc HĐH Solaris

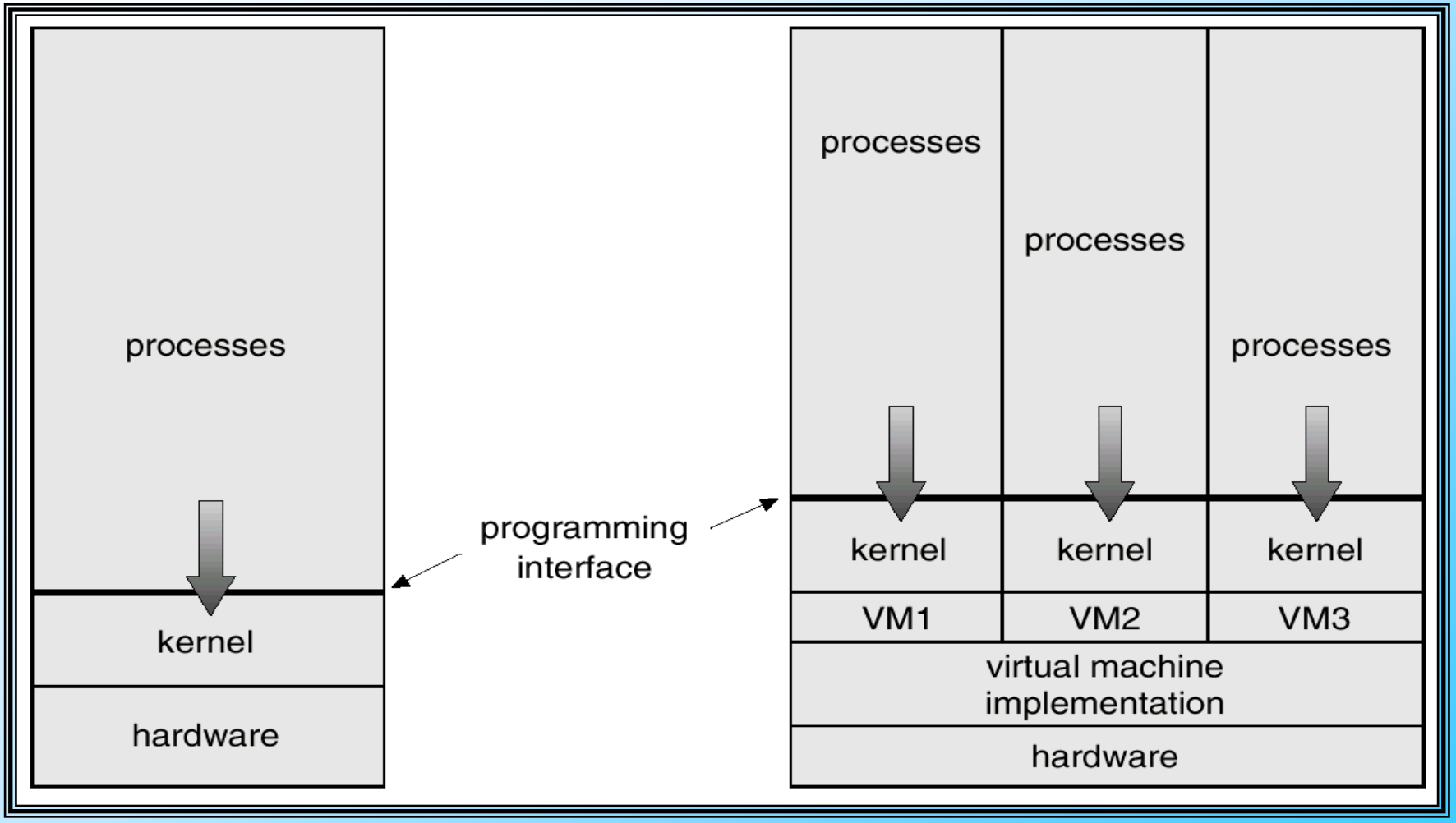
- được tổ chức xung quanh một hạt nhân nòng cốt với bảy loại của mô-đun hạt nhân khả nạp:
  1. Các lớp điều phối
  2. Các hệ thống tập tin
  3. Các lời gọi hệ thống khả nạp
  4. Các định dạng thi hành được
  5. Các mô-đun dòng
  6. Các mô-đun hỗn hợp
  7. Các trình điều khiển thiết bị và bus

2.5 Máy ảo
(Virtual Machine)

Cấu trúc máy ảo (1/4)

- Do mục tiêu của HĐH là chạy được nhiều chương trình đồng thời trên một máy tính nên cách tốt nhất là tạo ra nhiều máy tính ảo từ một máy tính thật để các chương trình chạy riêng trên các máy ảo.
- Về nguyên tắc các chương trình không biết mình đang chạy trên máy ảo (trong suốt với chương trình), cũng không biết mình đang phải chia sẻ tài nguyên với các chương trình khác. Ví dụ:
  - CPU ảo: mỗi chương trình sở hữu một CPU ảo
  - Bộ nhớ ảo: mỗi chương trình một không gian nhớ riêng
- Đa nhiệm và phân chia thời gian
- Phân tách 2 chức năng của hđh:
  - Cung cấp đa chương (multiprogramming)
  - Cung cấp 1 máy tính mở rộng

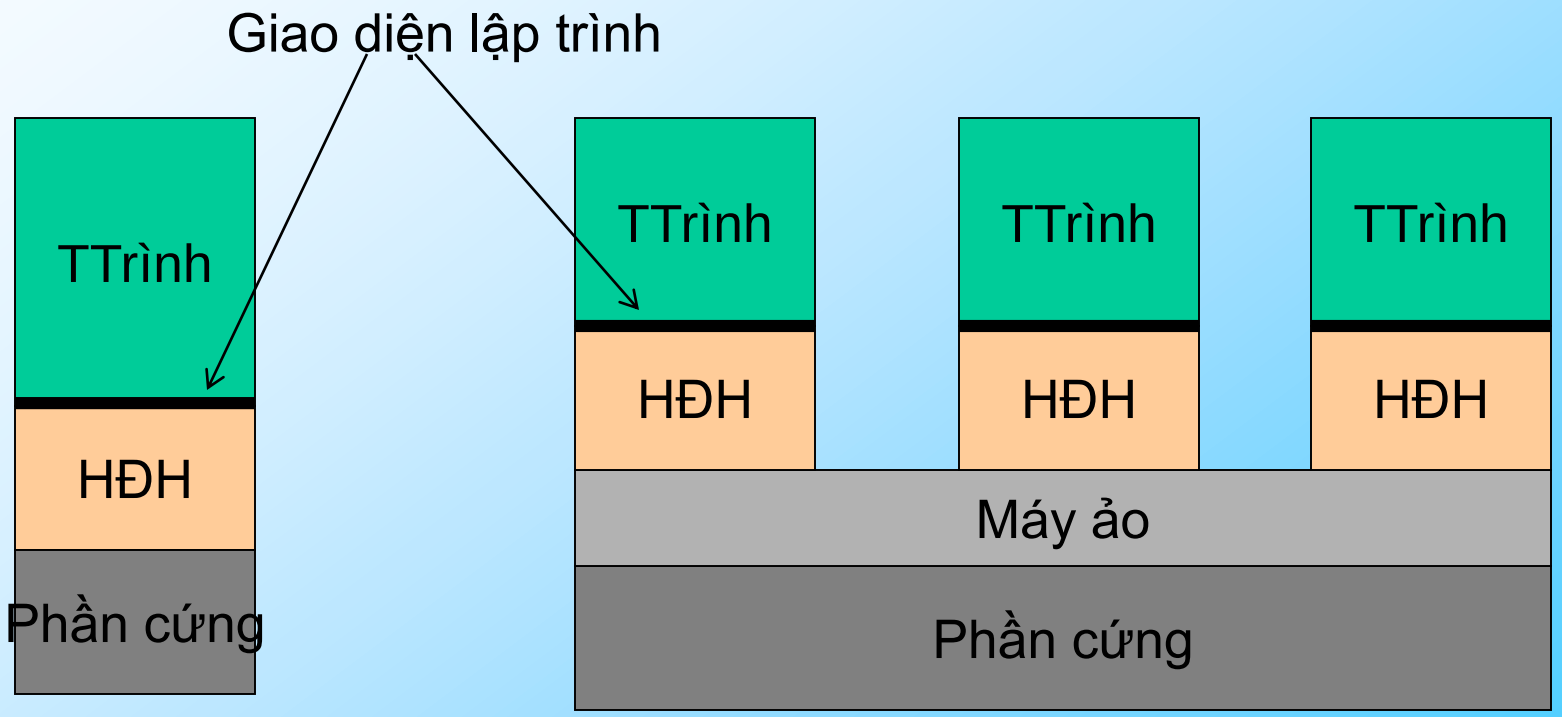
Cấu trúc máy ảo(2/4)



Non-virtual Machine

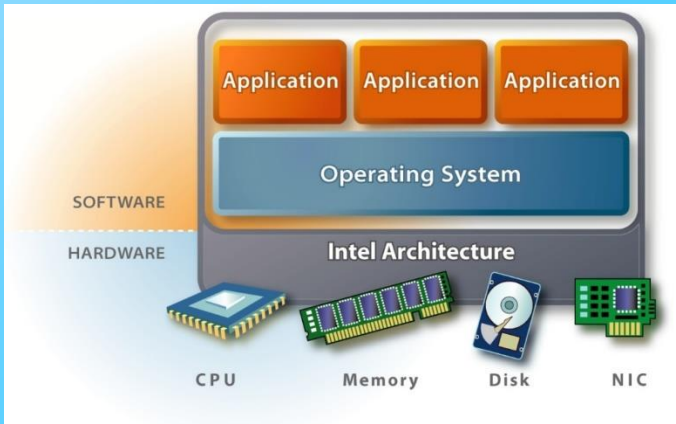
Virtual Machine

Cấu trúc máy ảo(3/4)



Cấu trúc máy ảo (4/4)

- Ưu điểm:
  - Môi trường thuận lợi cho sự tương thích
  - Tăng tính an toàn cho hệ thống do các VM độc lập
  - Dễ phát triển các HĐH đơn nhiệm cho các VM độc lập.
  - Tài nguyên hệ thống được bảo vệ hoàn toàn
  - Phân tách đa chương và máy tính mở rộng
- Khuyết điểm
  - Phức tạp trong việc giả lập phần cứng



2.6 Khởi động Hệ thống
(System Boot)

System boot


Bootstrap
OS
Power on
Reboot
Khởi tạo hệ thống
CPU, device controller, main memory, load đoạn code khởi động hđh

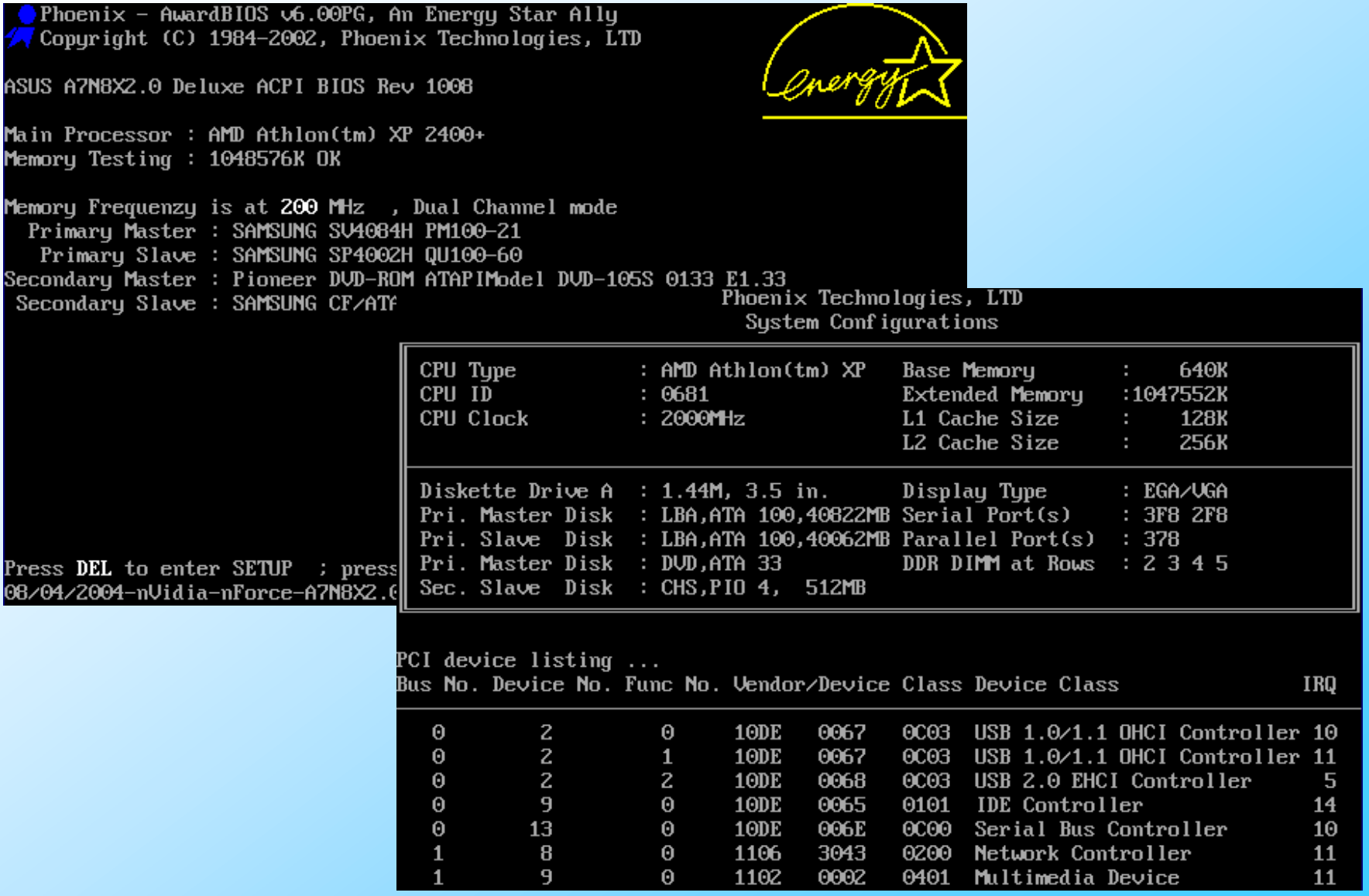
Quá trình khởi động máy tính

- Hệ điều hành nằm ở đâu ?
- Làm sao để máy tính nạp và chạy HĐH lúc khởi động?
  - Quá trình để khởi động HĐH gọi là booting
- Quá trình khởi động của các máy hiện đại gồm 3 giai đoạn
  - CPU thực thi lệnh từ địa chỉ cố định biết trước (boot ROM)
  - Firmware nạp boot loader
  - Boot loader nạp HĐH
- (1) CPU thực thi lệnh từ địa chỉ biết trước trong bộ nhớ
  - Địa chỉ vùng nhớ này thường trỏ tới vùng nhớ chỉ đọc (ROM – read-only memory)
  - Với x86, địa chỉ này là 0xFFFF0, trỏ tới địa chỉ chương trình BIOS (basic input-output system) trong ROM

Quá trình khởi động máy tính (tt)

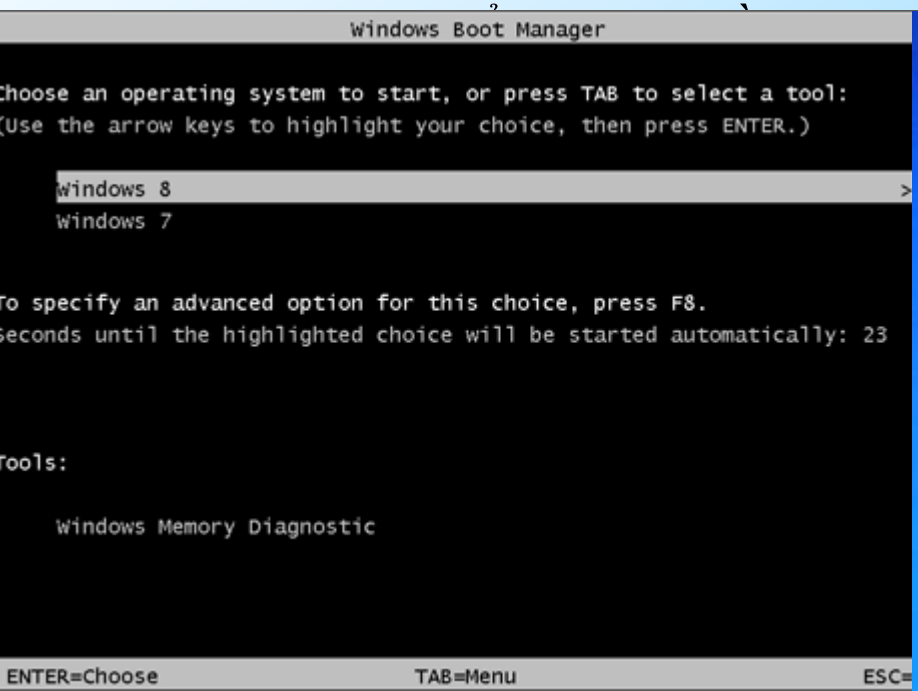
- (2) ROM chứa mã nguồn "boot"
  - Loại phần mềm chỉ đọc này gọi là firmware
  - Với x86, chương trình BIOS thực hiện lần lượt các công việc:
    - Kiểm tra cấu hình trong CMOS (complementary metal oxide semiconductor)
    - Nạp trình quản lý ngắt (interrupt handler) và các trình điều khiển thiết bị
    - Khởi tạo các thanh ghi và quản lý nguồn cung cấp (power management)
    - Thực hiện quá trình kiểm tra phần cứng (POST – power-on self-test)
    - Hiển thị các thiết lập hệ thống
    - Xác định các thiết bị có khả năng khởi động
    - Tiếp tục quá trình khởi động
  - Nạp và thực thi chương trình boot loader.

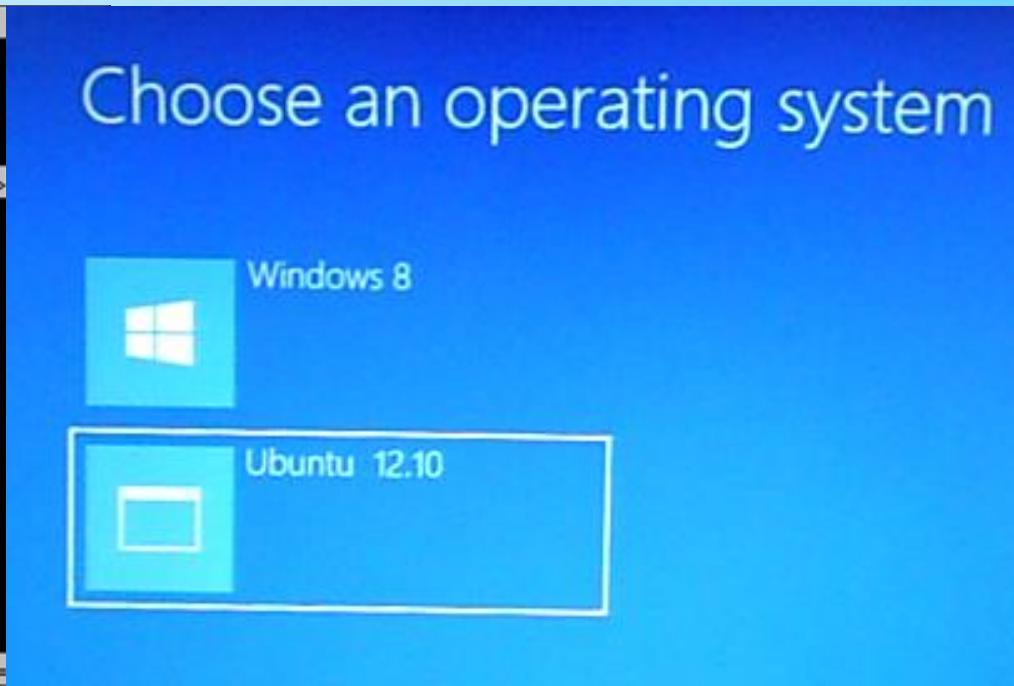
Thực thi firmware



Quá trình khởi động máy tính (tt)

- (3) Boot loader sau đó nạp phần còn lại của HĐH. Chú ý rằng tại thời điểm này HĐH vẫn chưa chạy
  - Boot loader hiểu được nhiều hệ điều hành khác nhau





Nạp Hệ Điều Hành

CÂU HỎI ÔN TẬP BÀI 2

1. Vẽ sơ đồ và mô tả 3 thành phần HĐH?
2. Liệt kê 3 loại giao diện người dùng của HĐH.
3. Liệt kê dịch vụ lõi chính của HĐH.
4. Lời gọi hệ thống là gì? Vẽ sơ đồ hoạt động?
5. Các kiểu cấu trúc HĐH?
6. Máy ảo là gì? Lợi ích của mô hình máy ảo?
7. Trình bày quá trình khởi động máy tính.

BÀI 3 : TIẾN TRÌNH VÀ LUỒNG

3.1. Khái niệm tiến trình

3.2 Giao tiếp giữa các tiến trình

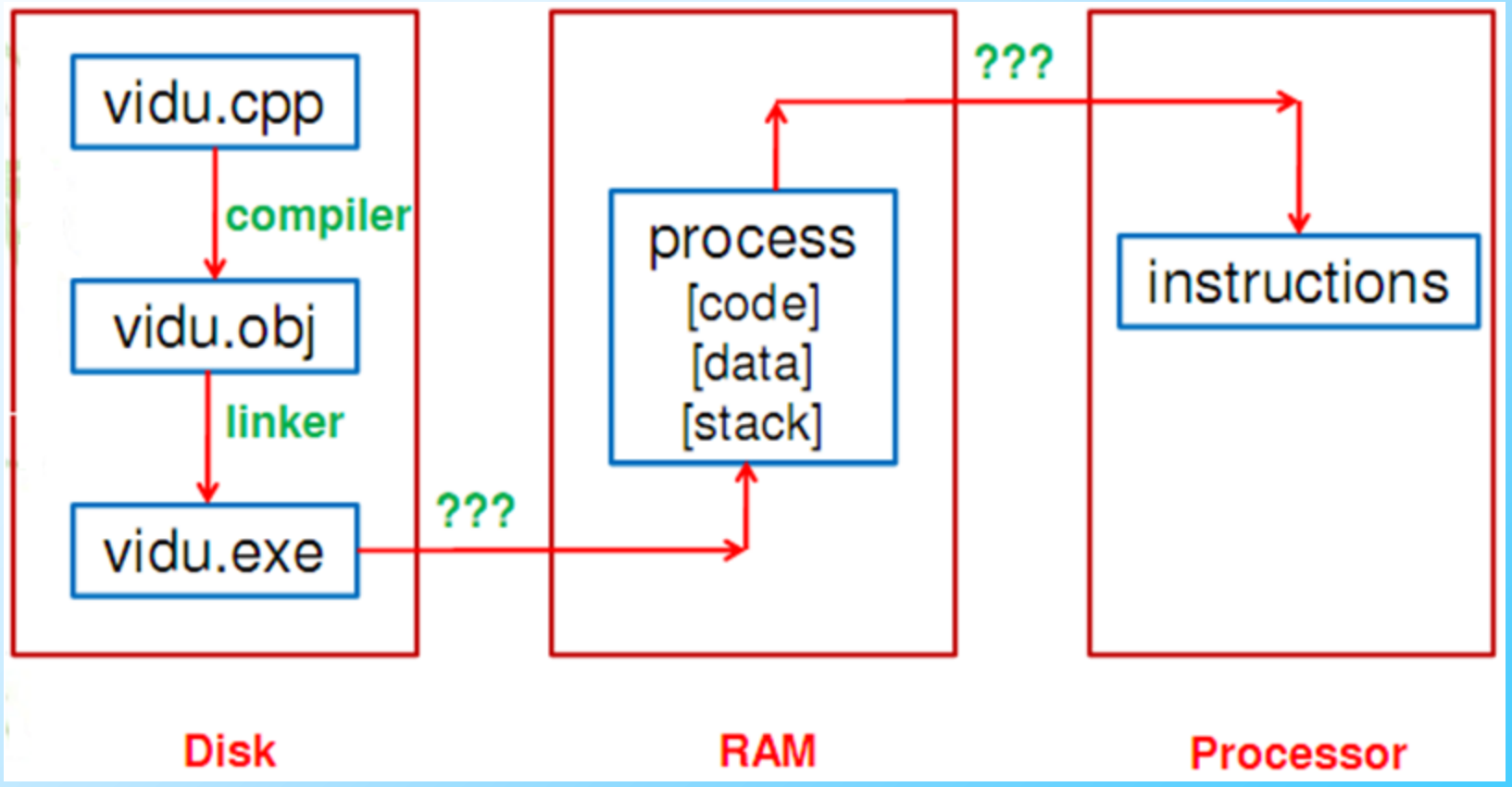
3.3 Hệ thống IPC trong Windows

3.4 Giao tiếp trong hệ thống Khách – Chủ

3.5 Luồng

3. 1 – Khái niệm tiến trình

Thực thi chương trình



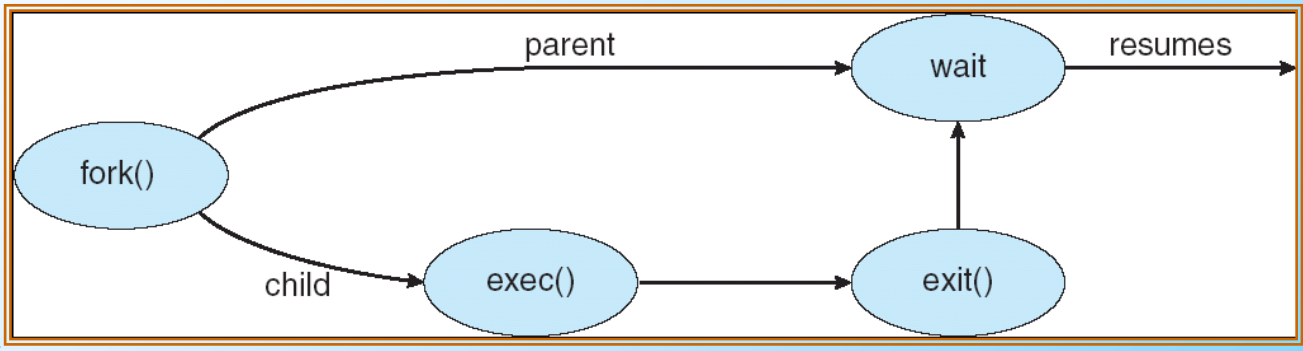
Tiến trình là gì?

- ***Tiến trình*** = một thể hiện của việc thi hành một chương trình
  - Thường gọi là "HeavyWeight Process"
- Tiến trình là một sự trừu tượng hóa cung cấp bởi HĐH, chỉ ra những gì là cần thiết để thi hành một chương trình
  - Một ngữ cảnh tính toán tách biệt cho mỗi ứng dụng
- Ngữ cảnh tính toán
  - Trạng thái CPU + không gian địa chỉ + môi trường
- **Tiến trình là một chương trình đang thực thi.**

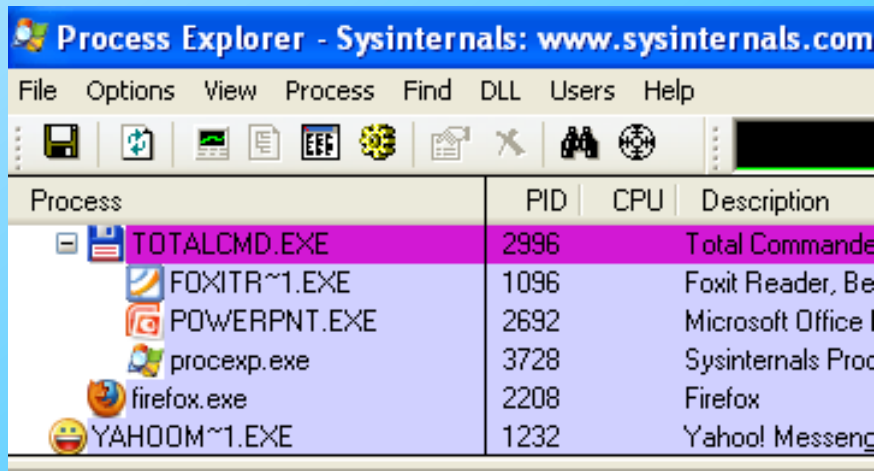
Tiến trình là gì?

- Tiến trình thường gồm có hai phần:
  - Một dãy các lệnh mà nó cần phải thi hành
    - Code
    - Trạng thái CPU
  - Các tài nguyên của riêng nó
    - Trạng thái bộ nhớ chính (không gian địa chỉ)
    - Trạng thái nhập xuất (file đang thao tác)

Thao tác trên tiến trình (1/2)



- Tạo tiến trình
  - Khởi động hệ thống
  - Người dùng kích hoạt một chương trình
  - Một tiến trình tạo một tiến trình khác
    - Unix/ Linux: **exec(), fork()**
    - Windows: **CreateProcess()**
  - Cây tiến trình
    - Unix/ Linux: các tiến trình cha, con có mối quan hệ chặt chẽ
    - Windows: các tiến trình cha, con độc lập với nhau



Thao tác trên tiến trình (2/2)

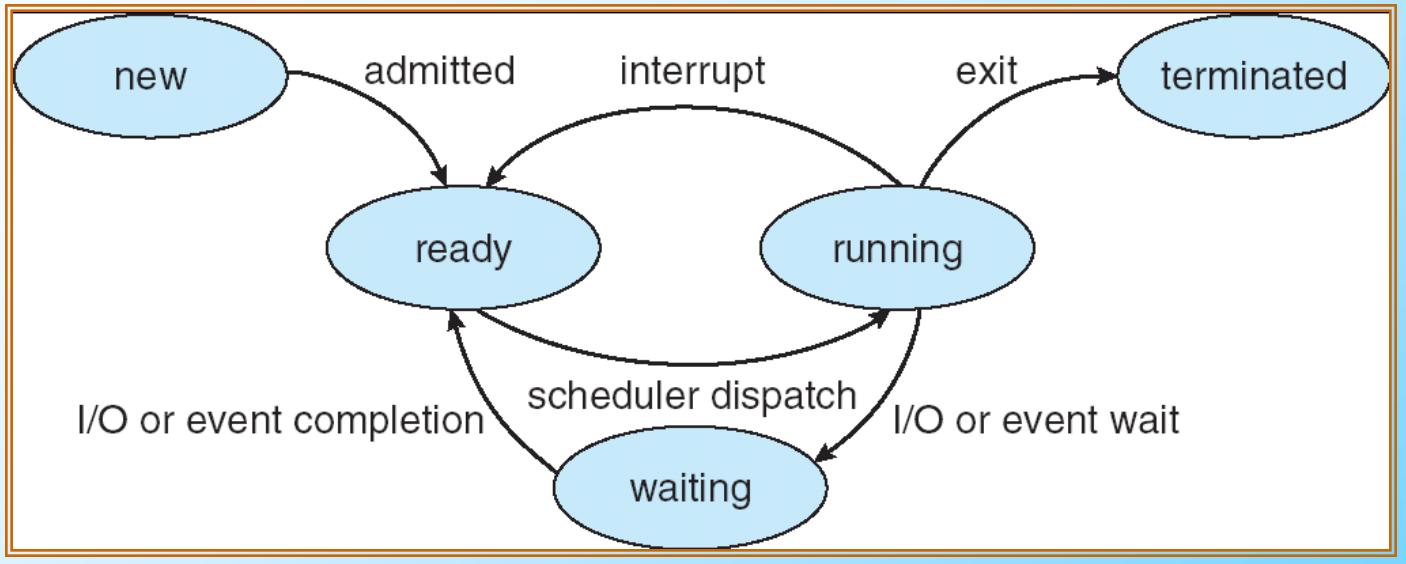
- Dừng tiến trình
  - Xử lý xong lệnh cuối cùng hay gọi lệnh kết thúc
    - Unix/ Linux: **exit()**
    - Windows: **ExitProcess()**
  - Một tiến trình yêu cầu dừng một tiến trình khác
    - Unix/ Linux: **kill()**
    - Windows: **TerminateProcess()**

    Điều gì xảy ra nếu tiến trình "nạn nhân" vẫn chưa muốn "chết"?
  - Do lỗi chương trình

Trạng thái tiến trình

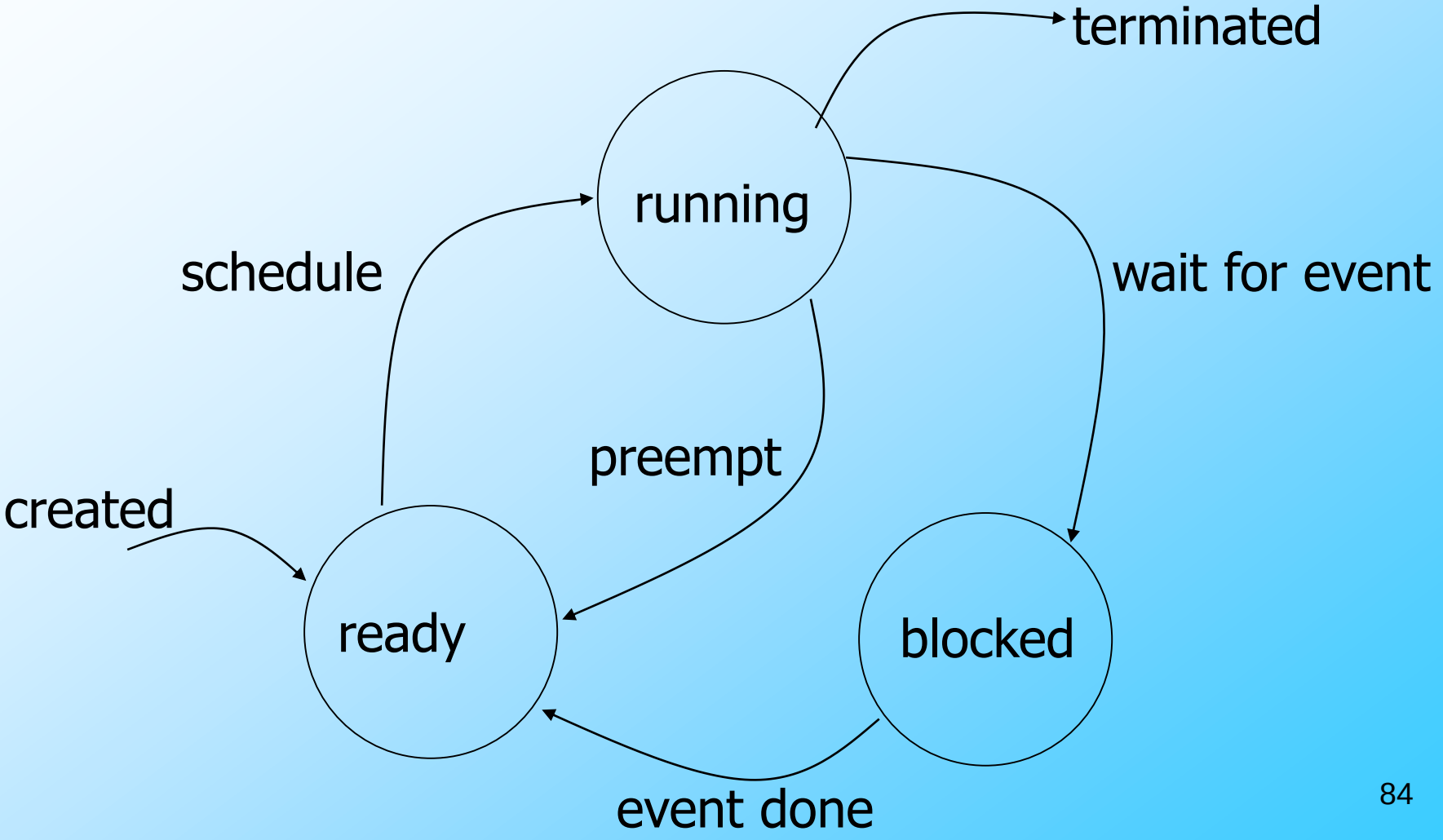
- Khi một tiến trình thực thi, nó thay đổi trạng thái.
- Trạng thái của một tiến trình được xác định bởi hoạt động hiện tại của nó.
- Mỗi tiến trình có thể ở một trong những trạng thái sau: mới (new), sẵn sàng (ready), đang chạy (running), chờ (waiting), hay kết thúc (terminated).

Lưu đồ trạng thái của tiến trình

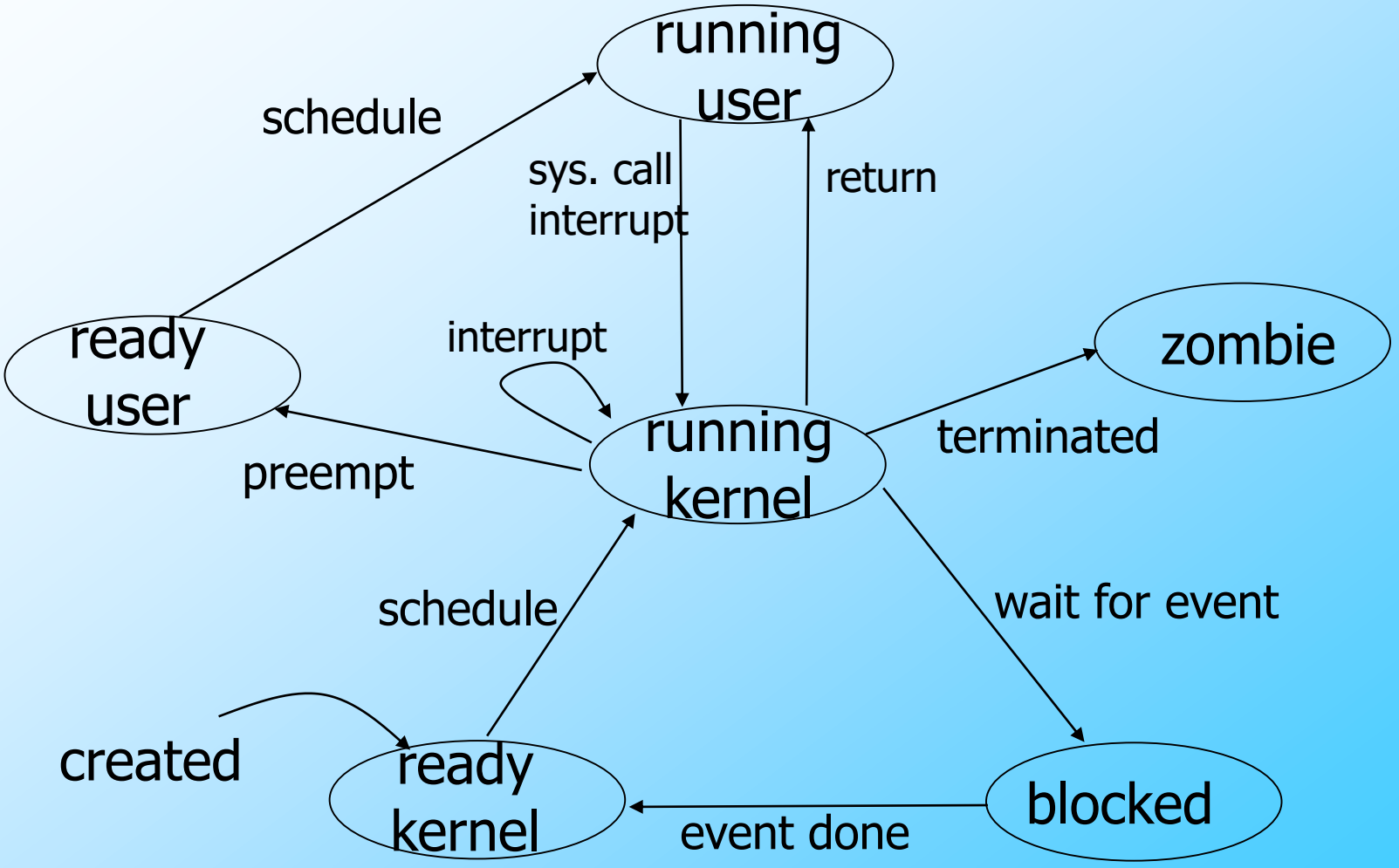


- ***Trạng thái của tiến trình***
  - **new**: Tiến trình vừa được tạo (chạy chương trình)
  - **ready**: Tiến trình sẵn sàng để chạy (đang chờ cấp CPU)
  - **running**: Tiến trình đang chạy (thi hành lệnh)
  - **waiting**: Tiến trình chờ đợi một sự kiện
  - **terminated**: Tiến trình kết thúc thi hành lệnh

Trạng thái của tiến trình (khác)

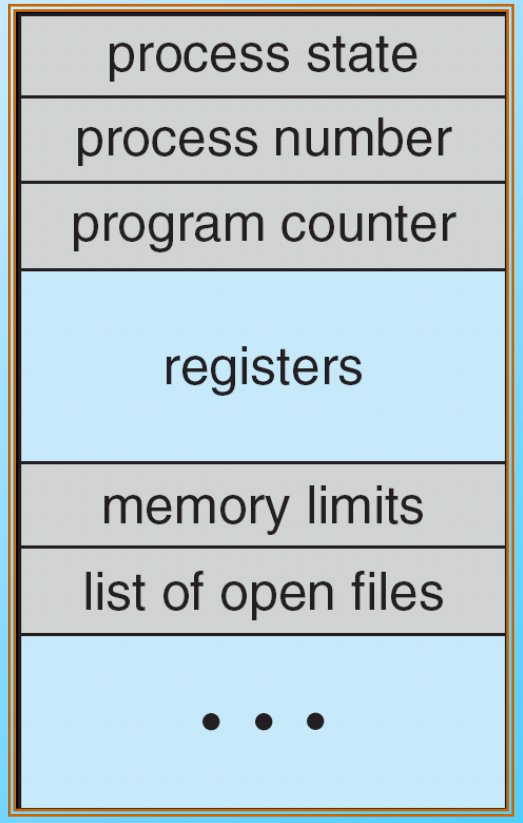


Môi trường UNIX



Khối điều khiển tiến trình (PCB)

- Ngữ cảnh tính toán của mỗi tiến trình được lưu trong một ***khối điều khiển tiến trình (Process Control Block: PCB)***

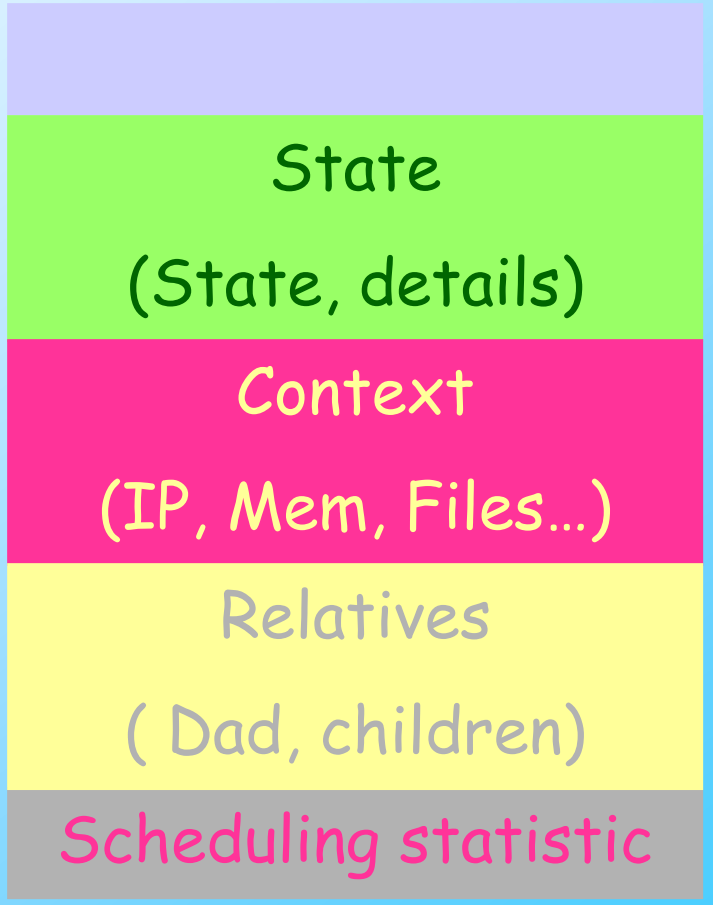


Process Control Block

- Thông tin gắn với mỗi tiến trình:
  - Trạng thái tiến trình
  - Con trỏ chương trình
  - CPU register
  - Thông tin lập lịch CPU
  - Thông tin quản lý bộ nhớ
  - Thông tin kế toán (ai đang sử dụng bao nhiêu resource)
  - Thông tin trạng thái I/O

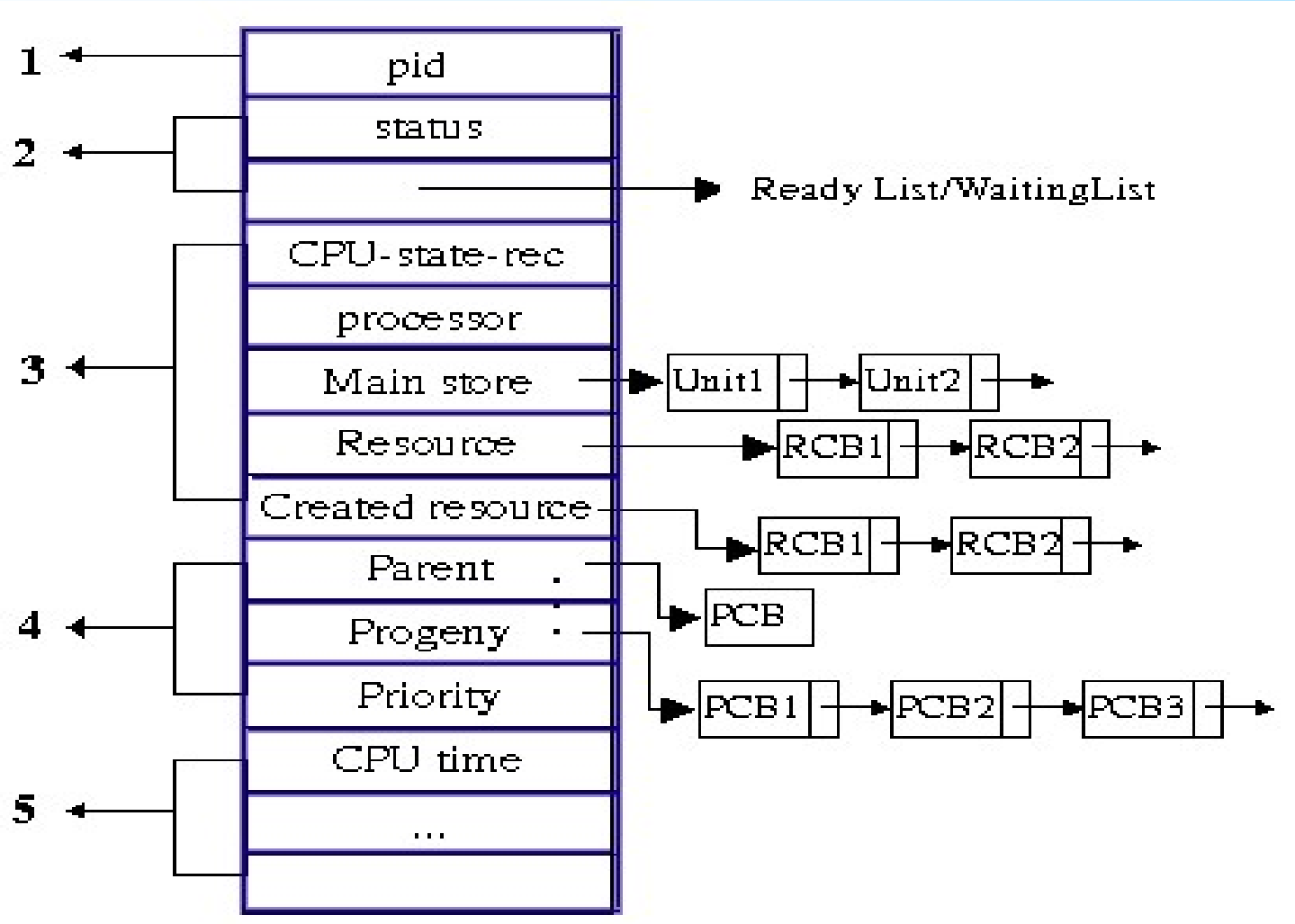
Khối điều khiển tiến trình (PCB) (2/4)

- Định danh (Process ID)
- Trạng thái tiến trình
- Ngữ cảnh tiến trình
  - Trạng thái CPU
  - Bộ xử lý (cho máy nhiều CPU)
  - Bộ nhớ chính
  - Tài nguyên sử dụng /tạo lập
- Thông tin giao tiếp
  - Tiến trình cha, tiến trình con
  - Độ ưu tiên
- Thông tin thống kê

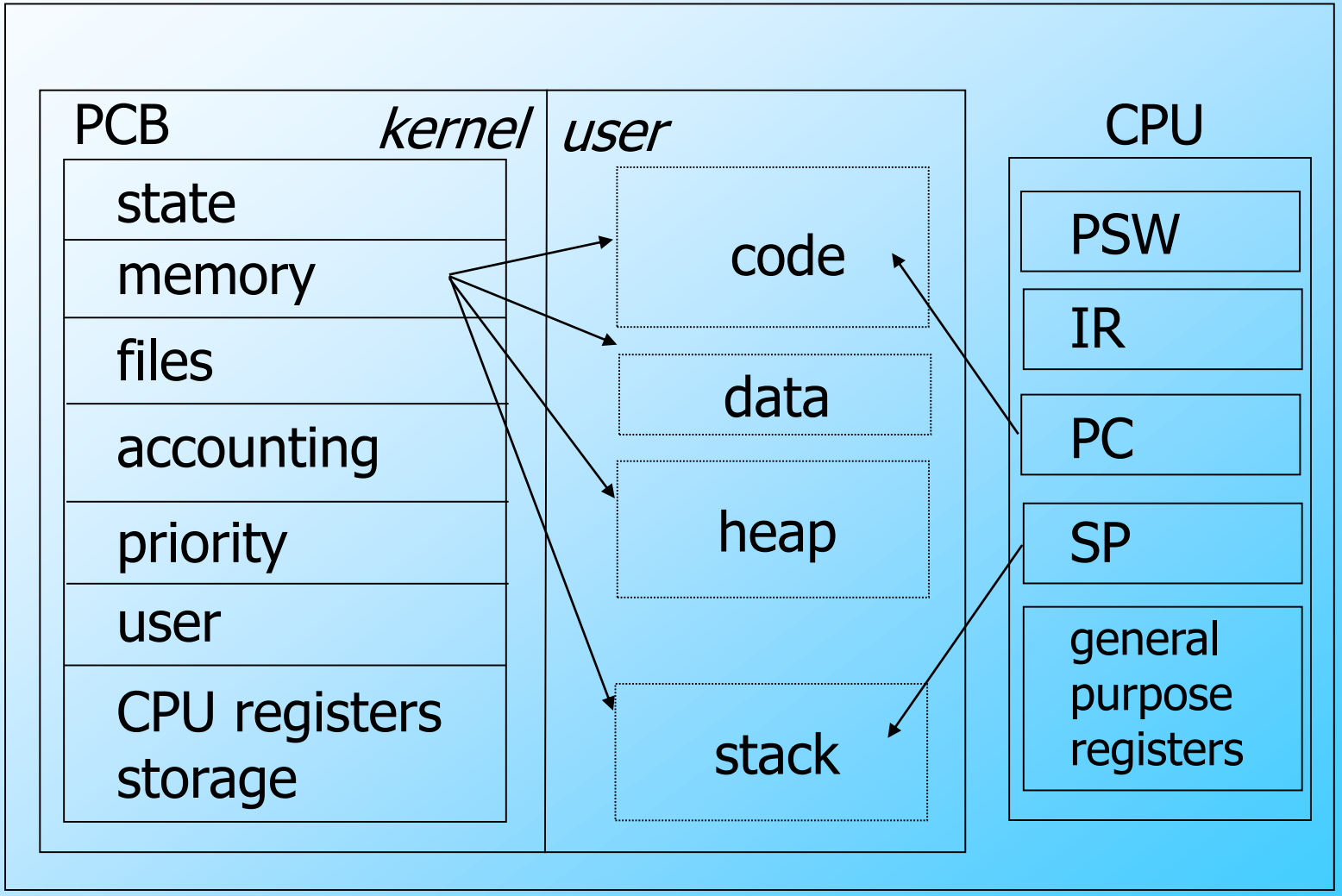


Process control Block – PCB

Khối điều khiển tiến trình (PCB) (3/4)

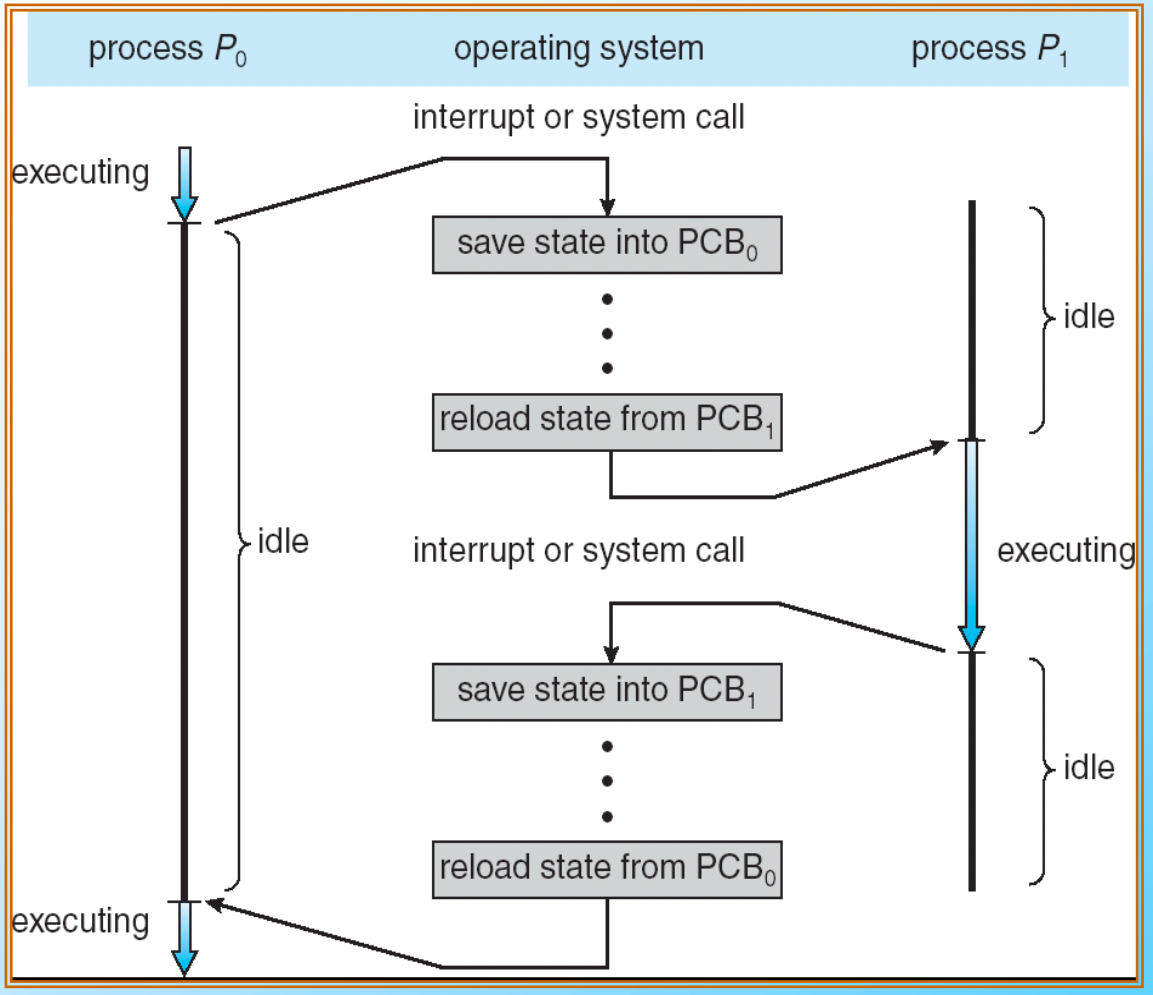


Khối điều khiển tiến trình (PCB) (4/4)



Chuyển đổi ngữ cảnh tiến trình

Context switching – Nhiệm vụ của Dispatcher



- CPU chuyển đổi tiến trình này sang tiến trình khác
- ***Chuyển đổi ngữ cảnh (context switching) → overhead***
- Trạng thái của tiến trình luôn thay đổi

Chuyển đổi ngữ cảnh tiến trình (2/4)

- Chuyển đổi ngữ cảnh xảy ra khi chuyển CPU qua lại giữa các quá trình. Quá trình diễn ra như sau:
  - Lưu trạng thái của tiến trình cũ vào PCB của nó bao gồm giá trị các thanh ghi, trạng thái tiến tình, thông tin quản lý bộ nhớ,…
  - Nạp ngữ cảnh được lưu của quá trình mới được bộ định thời CPU chọn để thực thi.

Chuyển đổi ngữ cảnh tiến trình (3/4)

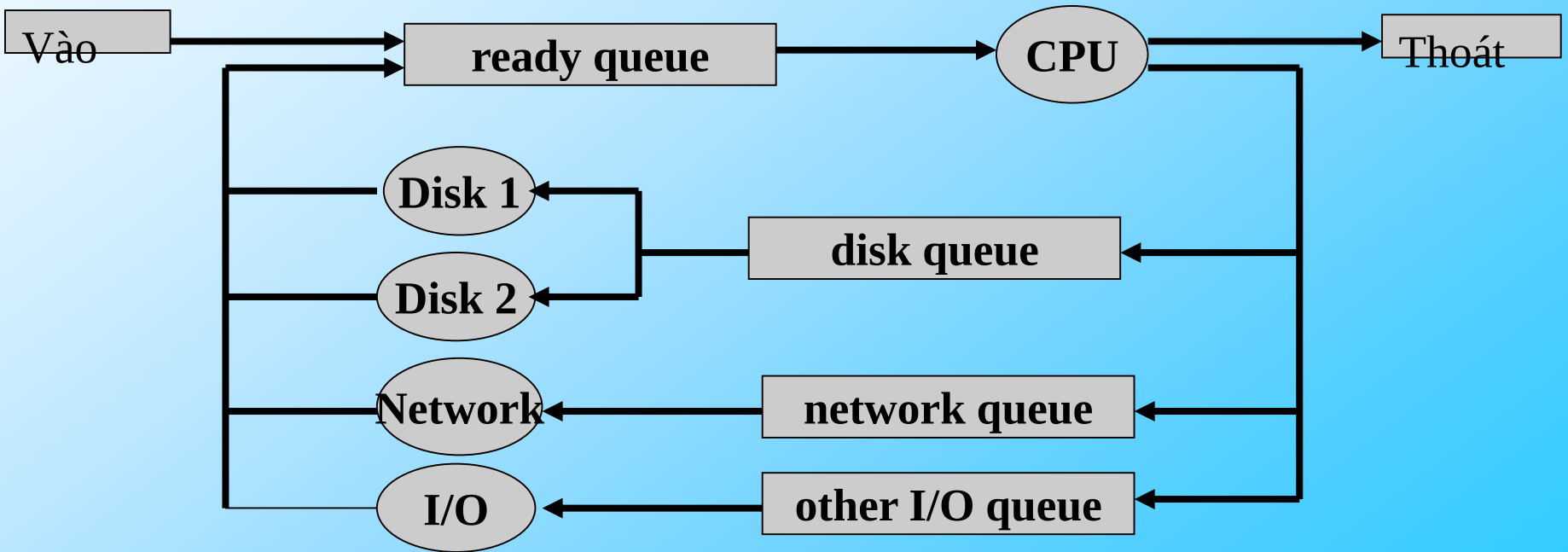
- Bản thân HĐH cũng là 1 phần mềm, nghĩa là cũng sử dụng CPU để có thể chạy được.
- Câu hỏi: Khi tiến trình A đang chiếm CPU, làm thế nào HĐH có thể thu hồi CPU lại được ? (vì lúc này HĐH không giữ CPU)
  - Ép buộc tiến trình thỉnh thoảng trả CPU lại cho HĐH ? Có khả thi ?
  - Máy tính phải có 2 CPU, 1 dành riêng cho HĐH ?
  - HĐH sử dụng ngắt đồng hồ (ngắt điều phối) để kiểm soát hệ thống
    - Mỗi khi có ngắt đồng hồ, HĐH kiểm tra xem có cần thu hồi CPU từ 1 tiến trình nào đó lại hay không ?
    - HĐH chỉ thu hồi CPU khi có ngắt đồng hồ phát sinh.
    - Khoảng thời gian giữa 2 lần ngắt điều phối gọi là chu kỳ đồng hồ (tối thiểu là 18.2 lần / giây).

Chuyển đổi ngữ cảnh tiến trình

- Thời gian chuyển ngữ cảnh là chi phí thuần, lãng phí. Tốc độ chuyển ngữ cảnh tùy thuộc vào tốc độ bộ nhớ, số lượng thanh ghi, (thường từ 1 -> 1000 mili giây).
- Chuyển ngữ cảnh phụ thuộc nhiều vào hỗ trợ phần cứng. Nếu một chuyển ngữ cảnh vượt quá giới hạn thanh ghi thì phải sắp xếp lại và phải làm nhiều công việc dễ dẫn tới tìn trạng thắt cổ chai năng lực thực hiện.

Hàng đợi (queue) tiến trình

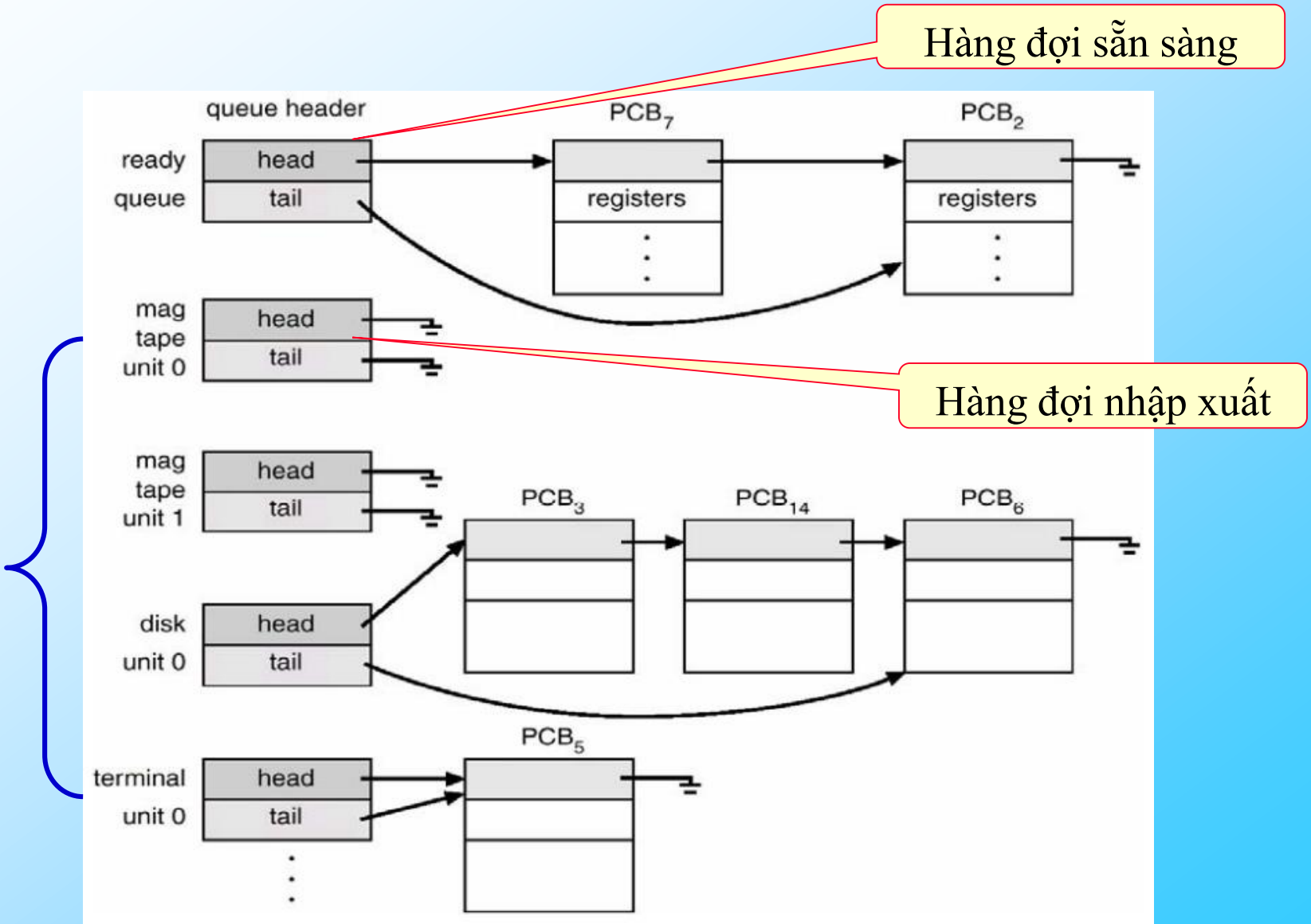
- Tiến trình khi không thực thi, được đặt vào hàng đợi.
- Các loại:
  - **Job queue** – tất cả các tiến trình trong hệ thống
  - **Ready queue** – các tiến trình đang ở trong bộ nhớ và sẵn sàng thực thi
  - **Device queues** – các tiến trình đang chờ thiết bị I/O
- Các tiến trình di chuyển giữa các queue, không cố định
- Sơ đồ hàng đợi:



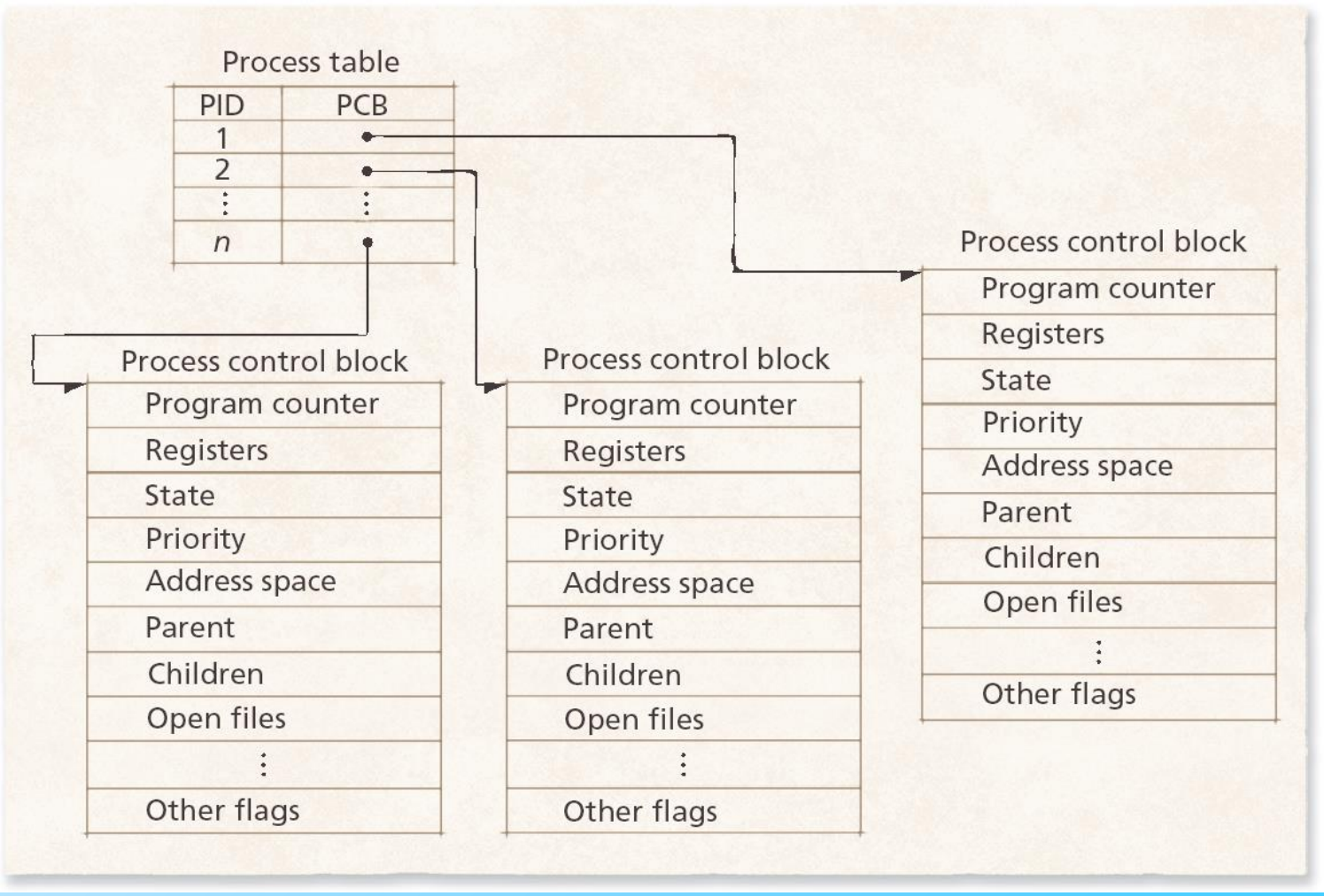
Hàng đợi (queue) tiến trình (2/3)

- Hàng đợi của một sự kiện chứa tất cả tiến trình đang ở trạng thái chờ đợi và đang chờ sự kiện đó xảy ra.
- Hàng đợi của một tài nguyên chứa tất cả tiến trình đang ở trạng thái chờ đợi và đang chờ được cấp tài nguyên đó.
- Hàng đợi lập thời biểu:
  - Hàng đợi công việc (Job queue): khi các tiến trình đưa vào hệ thống chúng sẽ nằm trong hàng đợi công việc. HDCV chứa tất cả các tiến trình trong hệ thống.
  - Hàng đợi sẵn sàng (Ready queue): tập các tiến trình nằm trên bộ nhớ chính sẵn sàng chờ được thực thi. Được lưu như 1 danh sách liên kết, đầu của HDSS chứa 2 con trỏ: 1 -> PCB đầu tiên và 1 -> PCB cuối cùng. Chúng ta bổ sung thêm trong mỗi PCB một trường con trỏ chỉ tới PCB kế tiếp.
  - Hàng đợi nhập xuất (I/O queue): tập danh sách các tiến trình chờ một thiết bị nhập xuất cụ thể. Mỗi thiết bị sẽ có hàng đợi của chính nó.
  - Một tiến trình di dời giữa 2 hàng đợi định thời khác nhau suốt thời gian sống của nó.

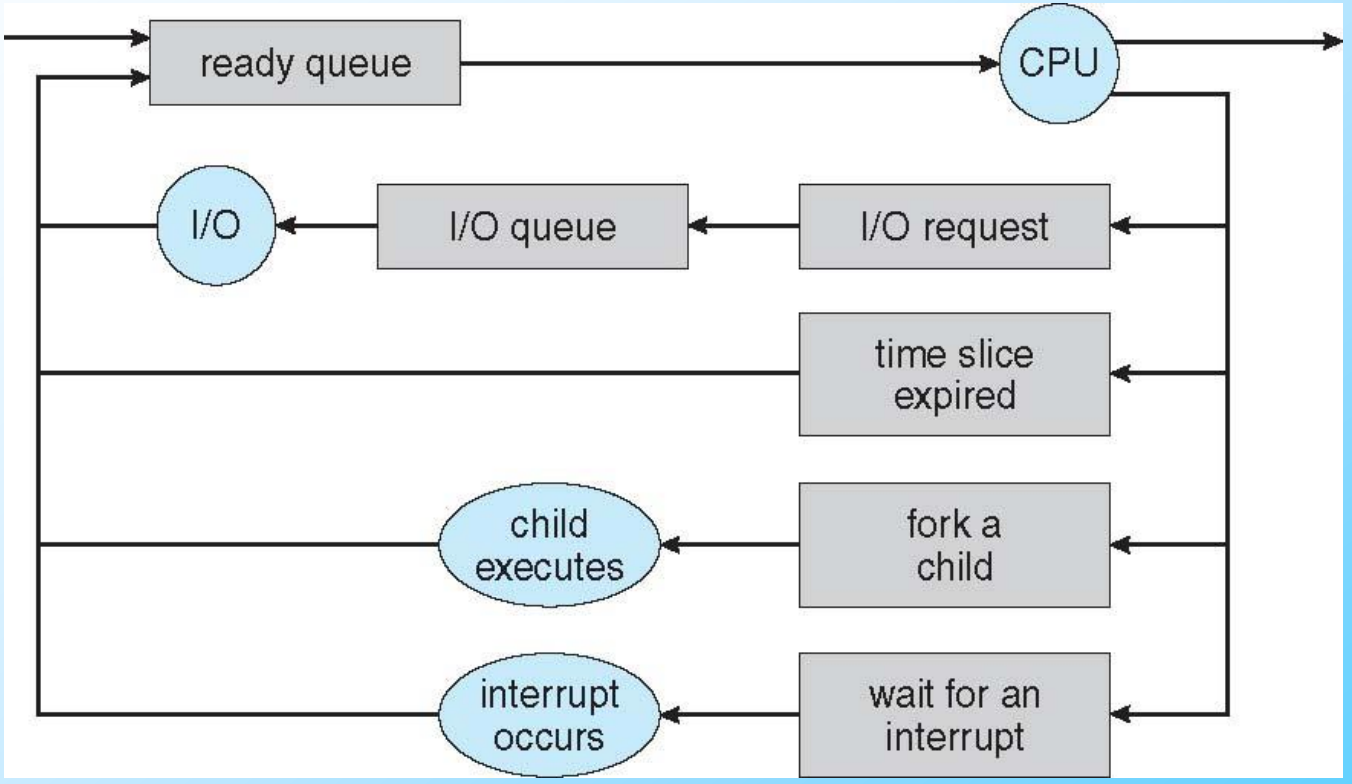
Hàng đợi (queue) tiến trình (3/3)



Biểu diễn của lập lịch tiến trình

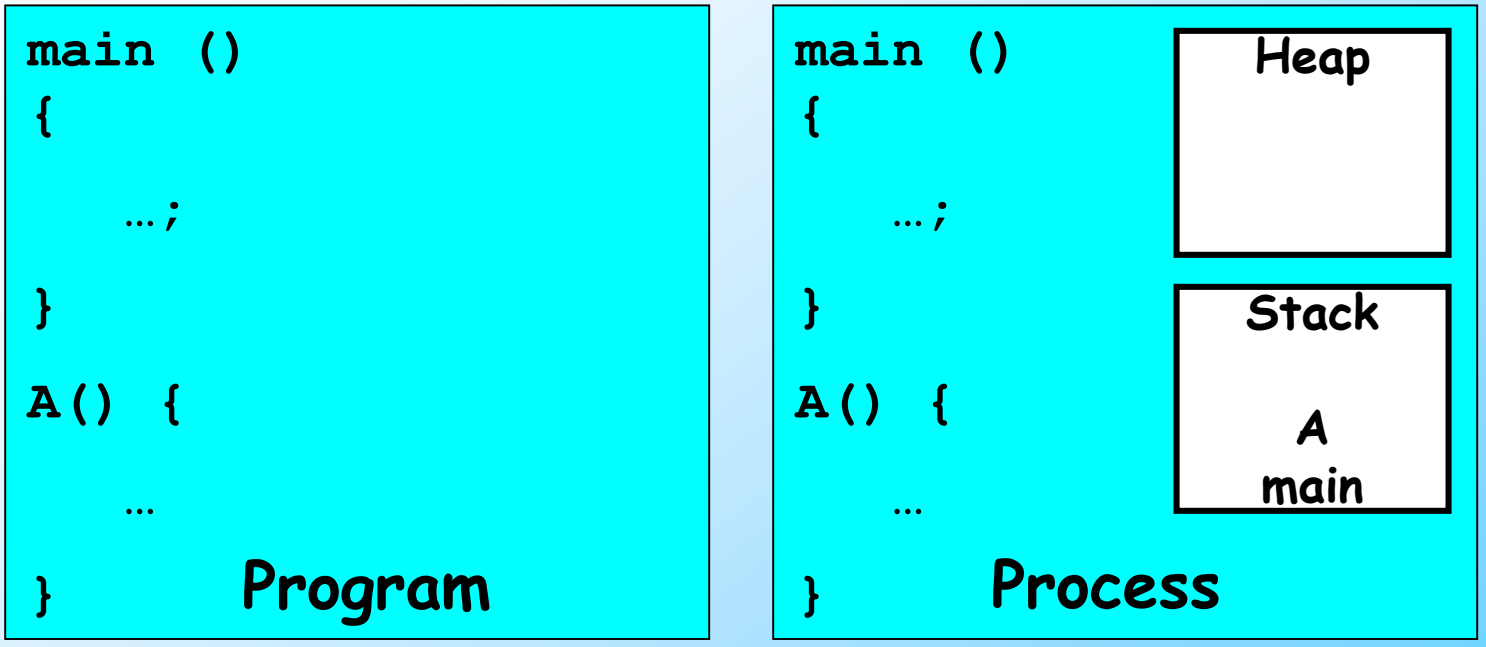


Biểu diễn của lập lịch tiến trình(2/2)



- Có nhiều hàng đợi:
  - ready queue: hàng đợi chứa các tiến trình sẵn sàng chạy
  - I/O queue: hàng đợi chứa các tiến trình sẵn sàng thi hành I/O
- Lựa chọn tiến trình nào → ***điều phối tiến trình***

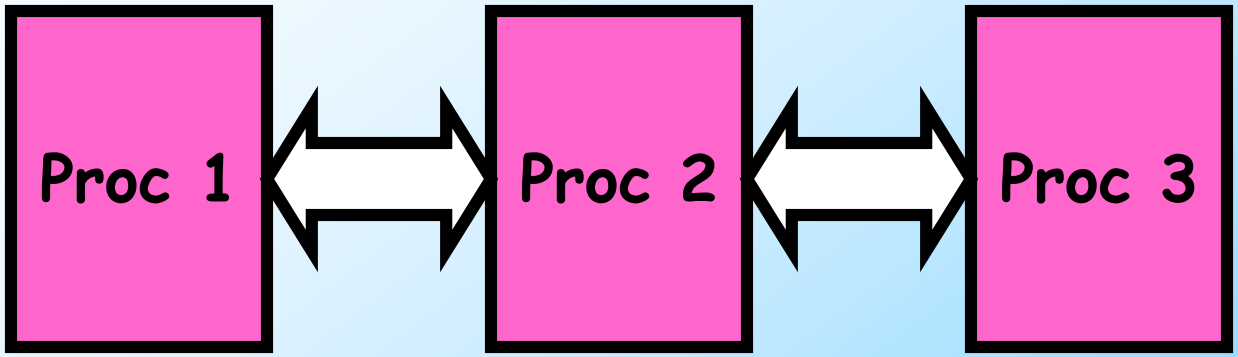
Tiến trình = Chương trình?



- Một chương trình có thể có nhiều tiến trình
  - Mở *Notepad.exe* xem file a.txt → 1 tiến trình.
  - Mở *Notepad.exe* xem file b.txt → 1 tiến trình.
- Chương trình nhìn từ góc độ mã lệnh chỉ là một phần của tiến trình.

3.2 - Giao tiếp giữa các tiến trình

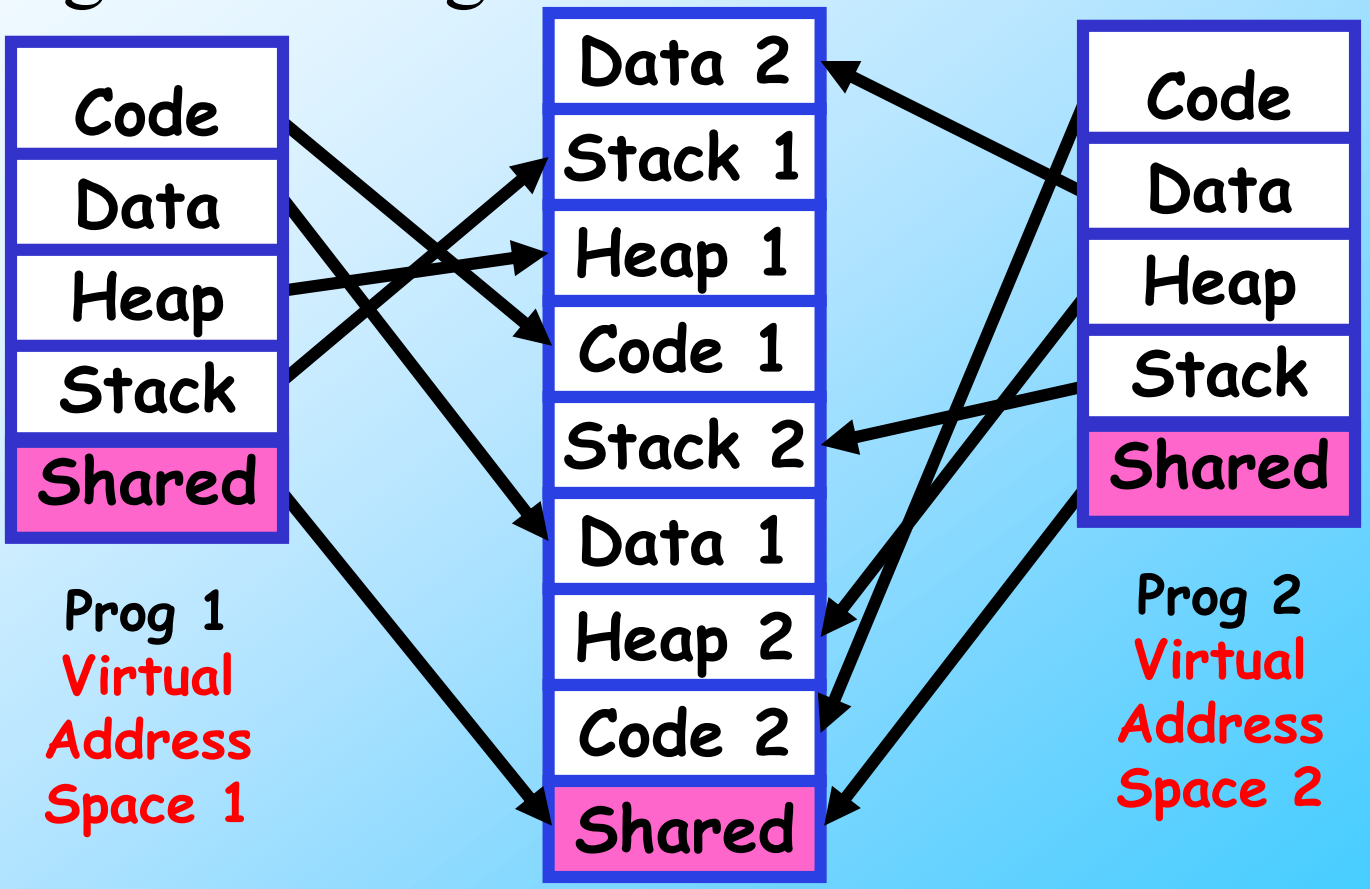
Nhiều tiến trình hợp tác



- Có những công việc cần có nhiều tiến trình hợp tác với nhau để hoàn thành.
- Thời gian để tạo tiến trình
  - Tạo khối PCB
  - Tạo không gian địa chỉ
- Thời gian chuyển đổi các tiến trình
- Cần có một cơ chế giao tiếp:
  - Tách biệt không gian địa chỉ của các tiến trình với nhau
  - Ánh xạ vùng nhớ chia sẻ
  - Truyền thông điệp
    - `send()` và `receive()`

Giao tiếp bằng Shared-Mem

- Giao tiếp thông qua thao tác đọc/ghi trên vùng nhớ chung



Giao tiếp giữa các tiến trình (IPC)

- (Inter-Process Communication) Cơ chế cho phép các tiến trình giao tiếp với nhau và đồng bộ hóa hành động của chúng
- Hệ thống thông điệp – các tiến trình giao tiếp với nhau không cần phải qua các biến dùng chung
- IPC cung cấp hai thao tác cơ bản:
  - **send**(*message*)
  - **receive**(*message*)
- Nếu tiến trình *P* và *Q* muốn giao tiếp với nhau, chúng phải:
  - tạo một đường giao tiếp giữa chúng
  - trao đổi các thông điệp thông qua send/receive

Giao tiếp giữa các tiến trình (IPC) (2/2)

- Các tiến trình đang thực thi có thể là độc lập hay hợp tác.
- Các tiến trình hợp tác phải có phương tiện giao tiếp với nhau : chia sẻ bộ nhớ, truyền thông điệp.
- Phương pháp chia sẻ bộ nhớ yêu cầu các tiến trình giao tiếp chia sẻ một số biến. Các tiến trình trao đổi thông tin thông qua việc sử dụng các biến dùng chung này. Chỉ hệ điều hành được cung cấp hệ thống bộ nhớ chia sẻ.
- Phương pháp truyền thông điệp cho phép các tiến trình trao đổi thông điệp.
- Nhiệm vụ cung cấp cơ chế giao tiếp có thể tách rời với hệ điều hành.
- Hai cơ chế này có thể được dùng cùng một lúc trong một hệ điều hành.

3.3 – Hệ thống IPC trong Windows

LPC – Local Procedute Call

- LPC trong Windows dùng để giao tiếp giữa hai tiến trình trên cùng một máy.
- Windows sử dụng đối tượng cổng (port) để thiết lập và duy trì một kết nối giữa 2 tiến trình
- Windows sử dụng hai loại cổng: cổng kết nối và các cổng giao tiếp.

Giao tiếp hoạt động của cổng kết nối

- Tiến trình chủ mở một đối tượng cổng kết nối có tên và đợi các tiến trình khách kết nối.
- Tiến trình khách sẽ gửi một yêu cầu kết nối tới cổng có tên của tiến trình chủ bằng một thông điệp kết nối.
- Tiến trình chủ tạo ra hai cổng giao tiếp riêng tư và trả quyền sở hữu của một cổng cho tiến trình khách.
- Tiến trình khách và tiến trình chủ sử dụng cổng tương ứng để gửi thông điệp cho nhau và lắng nghe trả lời.

3.4 – Giao Tiếp trong hệ thống Khách – Chủ

Giao tiếp trong hệ thống Client - Server

- 2 chiến lược để giao tiếp trong hệ thống client-server: ổ cắm và các cuộc gọi thủ tục từ xa.
- Một ổ cắm (socket) được định nghĩa là một thiết bị đầu cuối để liên lạc.
- Mô hình cuộc gọi từ xa (RPC - Remote Procedure Calls). RPC được thiết kế như là một cách trừu tượng hóa cơ chế gọi thủ tục để sử dụng giữa các hệ thống với kết nối mạng.

3.5 - Luồng

Khái niệm Luồng

- Luồng (thread) là một dòng điều khiển trong phạm vi một tiến trình.
- Tiến trình đa luồng gồm nhiều dòng điều khiển khác nhau trong cùng không gian địa chỉ.
- Những lợi điểm của đa luồng gồm đáp ứng nhanh đối với người dùng, chia sẻ tài nguyên trong tiến trình, tính kinh tế và khả năng thuận lợi trong kiến trúc đa xử lý.
- Tách biệt:
  - Trạng thái CPU, ngăn xếp
- Chia sẻ:
  - Mọi thứ khác: Data, Code, Heap, môi trường
  - Đặc biệt: Không gian địa chỉ (Tại sao?)

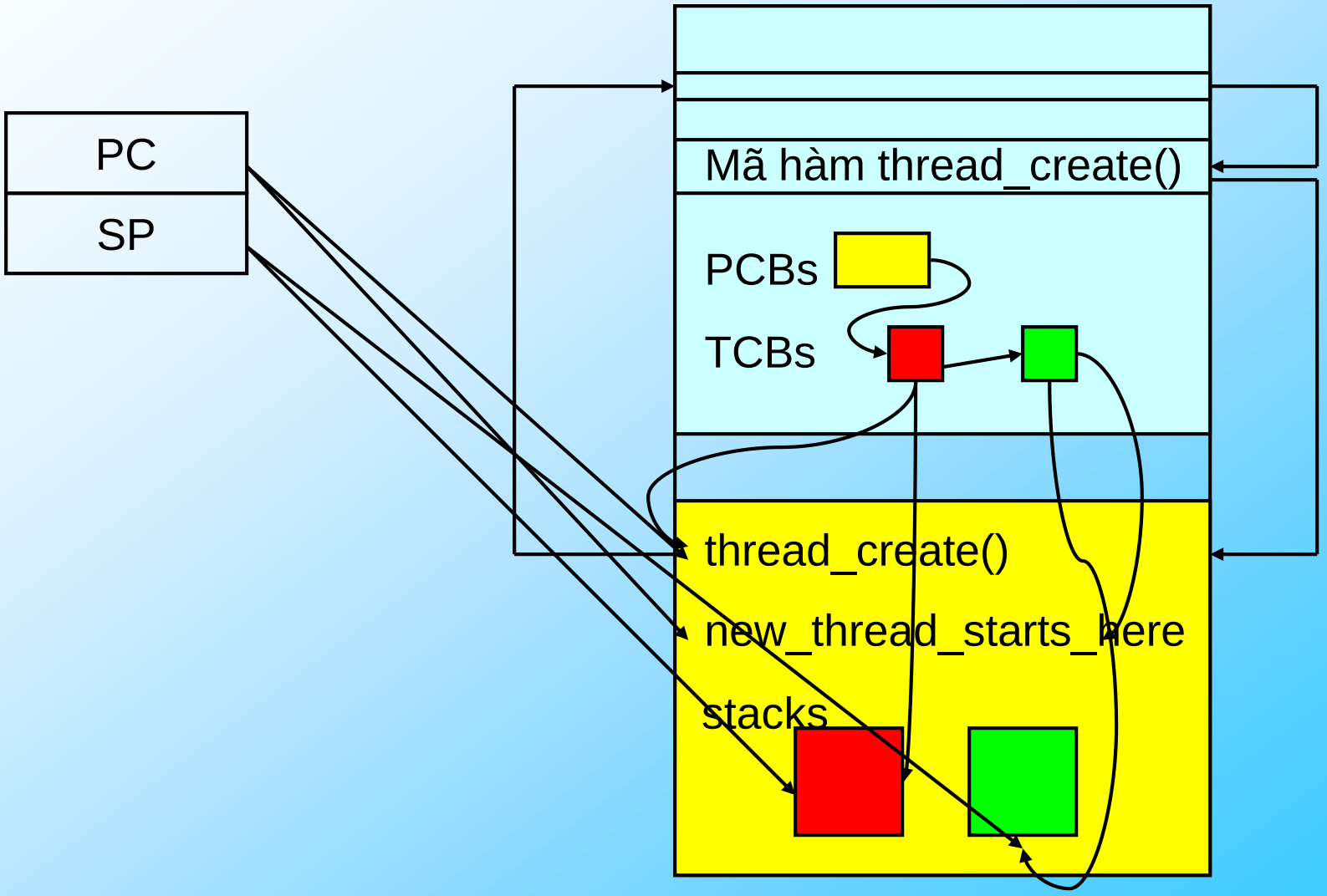
Khái niệm Luồng(2/2)

- Mỗi tiến trình luôn có một luồng chính (dòng xử lý cho hàm main())
- Ngoài luồng chính, tiến trình còn có thể có nhiều luồng con khác
- Các luồng của một tiến trình
  - Chia sẻ không gian vùng code và data
  - Có vùng stack riêng
- MultiThreading = một chương trình được tạo ra bằng một số các hoạt động đồng thời.
- HeaveWeight Process = Tiến trình với duy nhất một luồng.

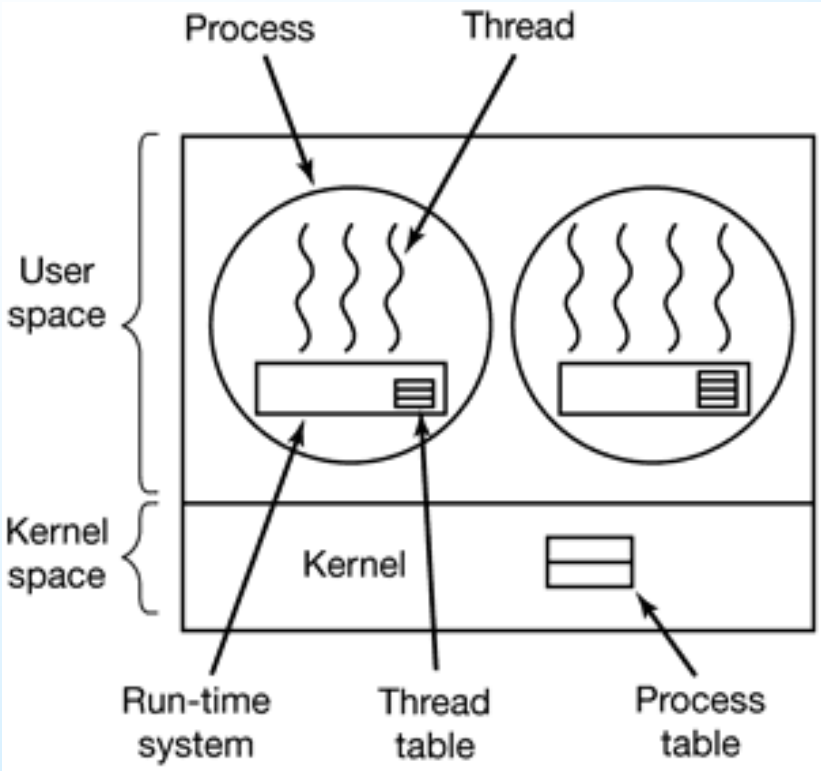
Khái niệm Khối quản lý luồng (Thread Control Block – TCB)

- TCB thường chứa các thông tin riêng của mỗi luồng
  - ID của luồng
  - Không gian lưu các thanh ghi
  - Con trỏ tới vị trí xác định trong ngăn xếp
  - Trạng thái của luồng
- Thông tin chia sẻ giữa các luồng trong một tiến trình
  - Các biến toàn cục
  - Các tài nguyên sử dụng như tập tin,…
  - Các tiến trình con
  - Thông tin thống kê
  - …

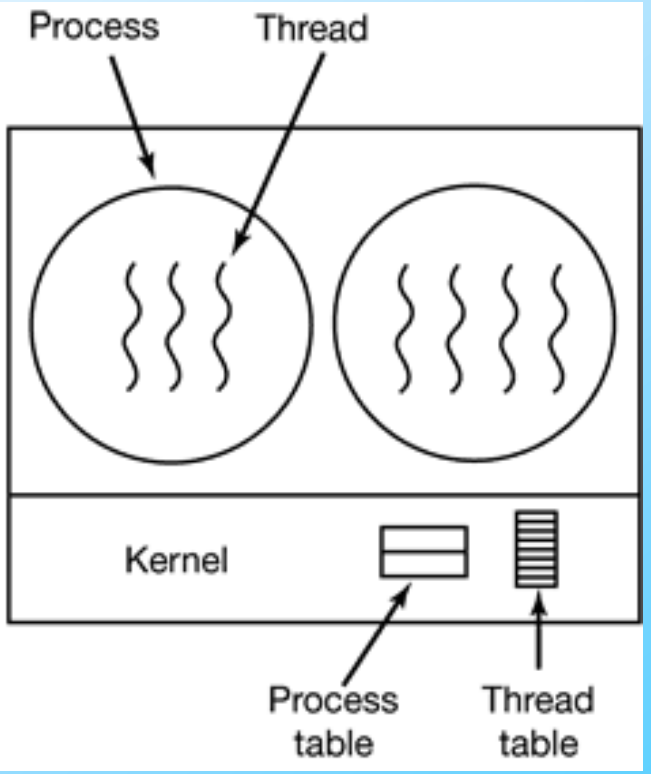
Tạo Luồng


PC
SP
Mã hàm thread_create()
PCBs
TCBs
thread_create()
new_thread_starts_here
stacks

Tổ chức Luồng

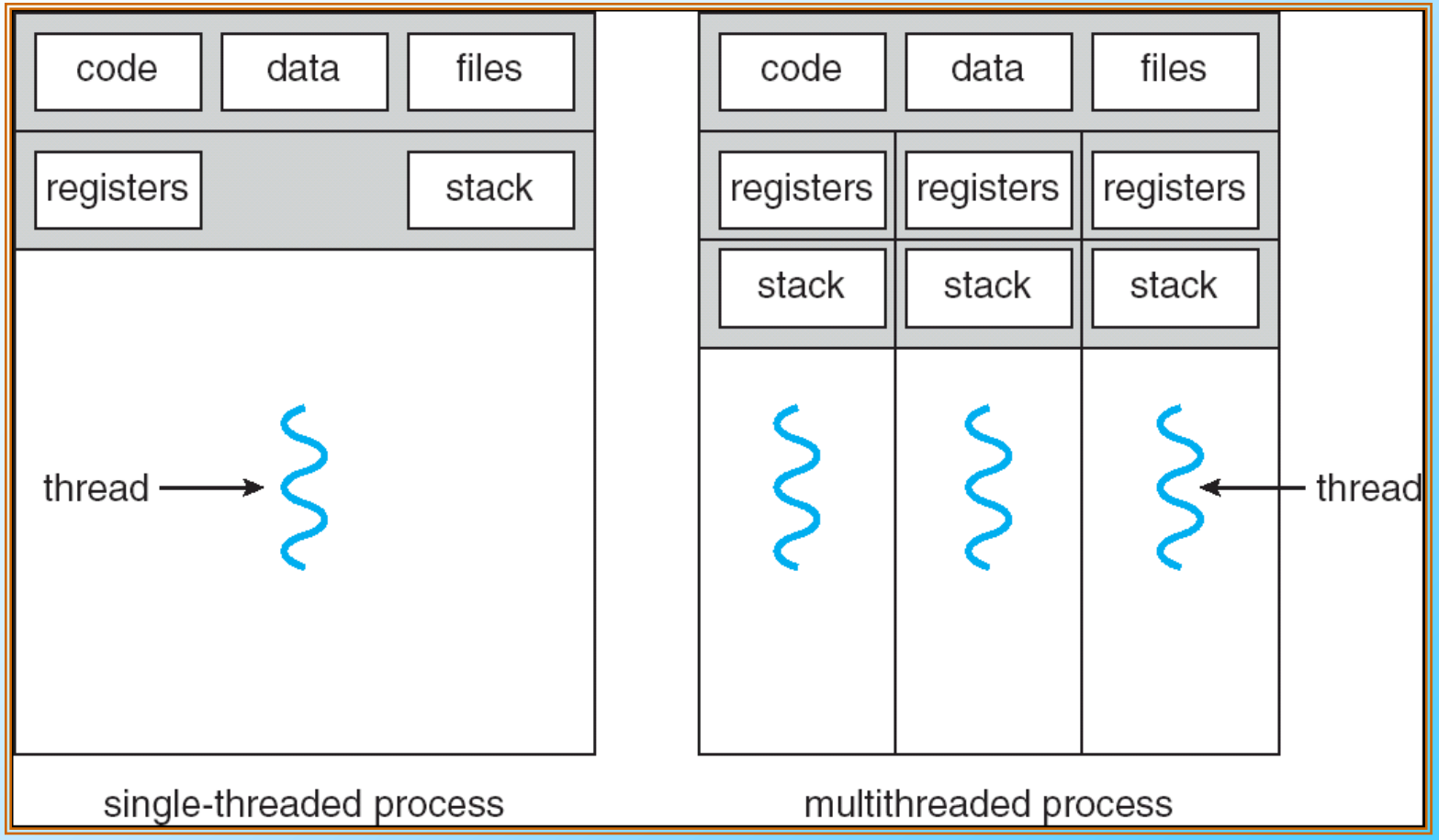


- Quản lý tiểu trình mức người dùng
- 3 thư viện chính hỗ trợ:
  - POSIX Pthreads
  - Win32 threads
  - Java threads



- Quản lý tiểu trình mức hệ thống
- Hệ điều hành hỗ trợ:
  - Windows XP/2000
  - Solaris
  - Linux
  - Mac OS X

Đơn Luồng – Đa Luồng



VD về chương trình Đa Luồng

- Database server:
  - Nhiều kết nối và cơ sở dữ liệu cùng một lúc
- Network Server:
  - Truy cập đồng thời từ môi trường mạng
  - Một tiến trình – nhiều thao tác đồng thời
  - File Server, Web server, ...
- Paralell Programming (có nhiều CPU)
  - Chia chương trình thành nhiều thread để tận dụng nhiều CPU
  - Còn gọi là Multi - Processing

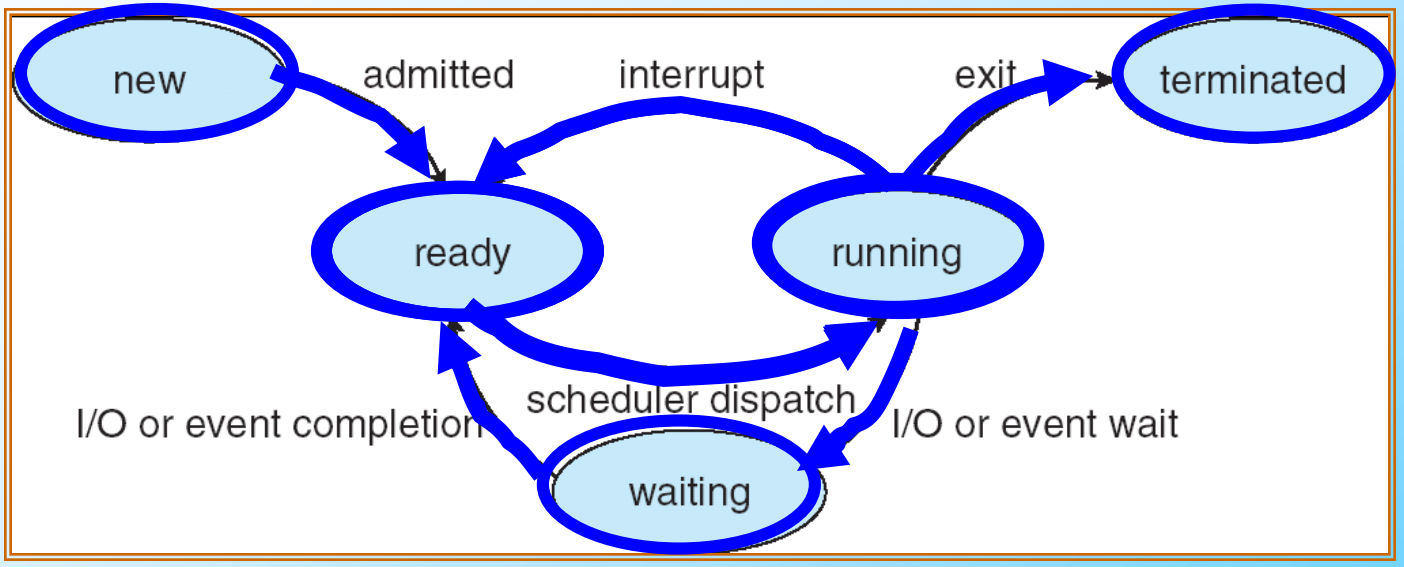
Hỗ trợ Luồng

- Hệ điều hành
  - Ưu điểm: lập lịch luồng được thực hiện bởi OS: Tối ưu hóa CPU
  - Khuyết điểm: nhiều luồng → overhead
- Mức người dùng
  - Ưu điểm: overhead thấp
  - Khuyết điểm: OS không nhận ra cụ thể
    - VD: một luồng bị block do I/O sẽ block tất cả các luồng khác cùng một tiến trình
- Các chương trình đa luồng đưa ra nhiều thử thách cho việc lập trình.
- Pthread API cung cấp tập hợp các hàm để tạo và quản lý luồng tại cấp người dùng.
- Java cung cấp một API tương tự cho việc hỗ trợ luồng. Tuy nhiên, vì các luồng Java được quản lý bởi JVM và không phải thư viện luồng cấp người dùng hay nhân, chúng không rơi vào loại luồng người dùng hay nhân.

So sánh Luồng và Tiến trình

- Tại sao không dùng nhiều tiến trình để thay thế cho việc dùng nhiều luồng ?
  - Các tác vụ điều hành luồng (tạo, kết thúc, điều phối, chuyển đổi,…) ít tốn chi phí thực hiện hơn so với tiến trình
  - Liên lạc giữa các luồng thông qua chia sẻ bộ nhớ, không cần sự can thiệp của kernel

Chuyển đổi trạng thái của Thread



- Tương tự như tiến trình:
  - new: Luồng được tạo mới
  - ready: Luồng đang chờ để chạy
  - running: Luồng đang được thi hành
  - waiting: Luồng đang chờ sự kiện
  - terminated: Luồng kết thúc thi hành
- Thông tin luồng lưu trong TCB

CÂU HỎI ÔN TẬP BÀI 3

1. Hãy nêu khái niệm tiến trình và luồng.
2. Xử lý đồng hành và ích lợi của nó?
3. Hãy vẽ sơ đồ thể hiện góc nhìn vật lý và góc nhìn logic (góc nhìn của người sử dụng) về xử lý đồng hành.
4. Làm thế nào để xem được danh sách các tiến trình đang hoạt động trong một máy PC chạy hệ điều hành Windows XP.
5. Hãy nêu các trạng thái của tiến trình.
6. Hãy nêu các thao tác quản lý tiến trình
7. Hãy vẽ sơ đồ trạng thái và các thao tác chuyển trạng thái tiến trình.

Câu hỏi

- Tiến trình là

  A. Chương trình viết bằng C#

  B. Một chương trình đang được CPU thực hiện

  C. Công việc in một văn bản ra máy in

  D. Không có câu nào đúng

- Tiến trình có bao nhiêu trạng thái
  A. 2
  B. 3
  C. 4
  D. 5

Hàng đợi điều phối tiến trình

A. 2

B. 3

C. 4

D. 5

BÀI 4 : Điều phối CPU

4.1 Các khái niệm cơ bản

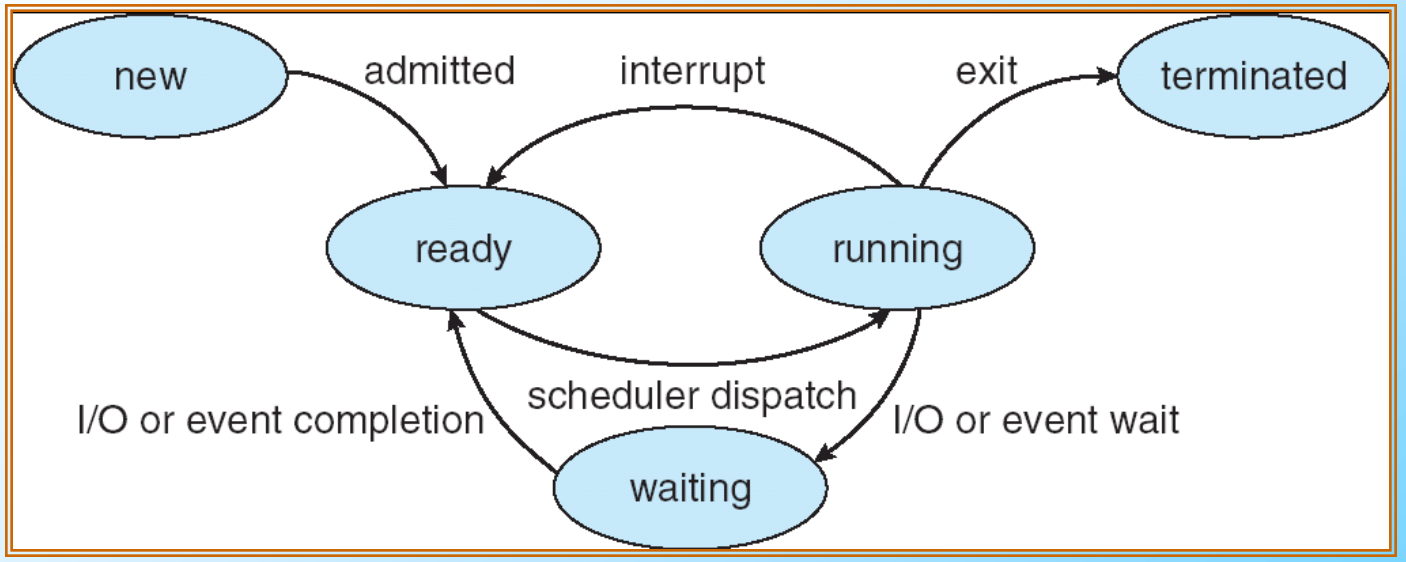
4.2 Các tiêu chuẩn điều phối

4.3 Các giải thuật điều phối

4.4 Điều phối đa bộ xử lý

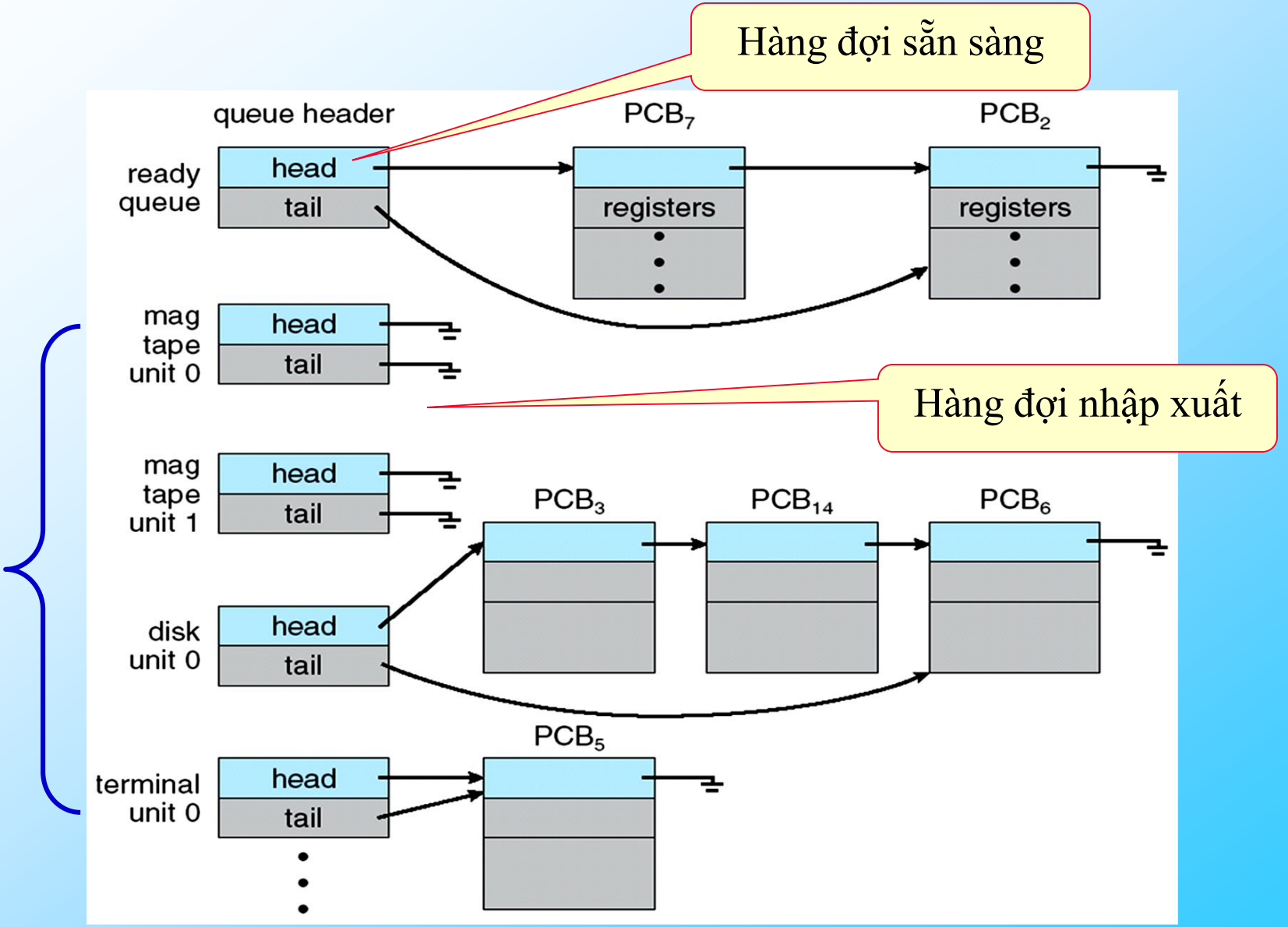
4.1 – Các khái niệm cơ bản

Lưu đồ trạng thái của tiến trình

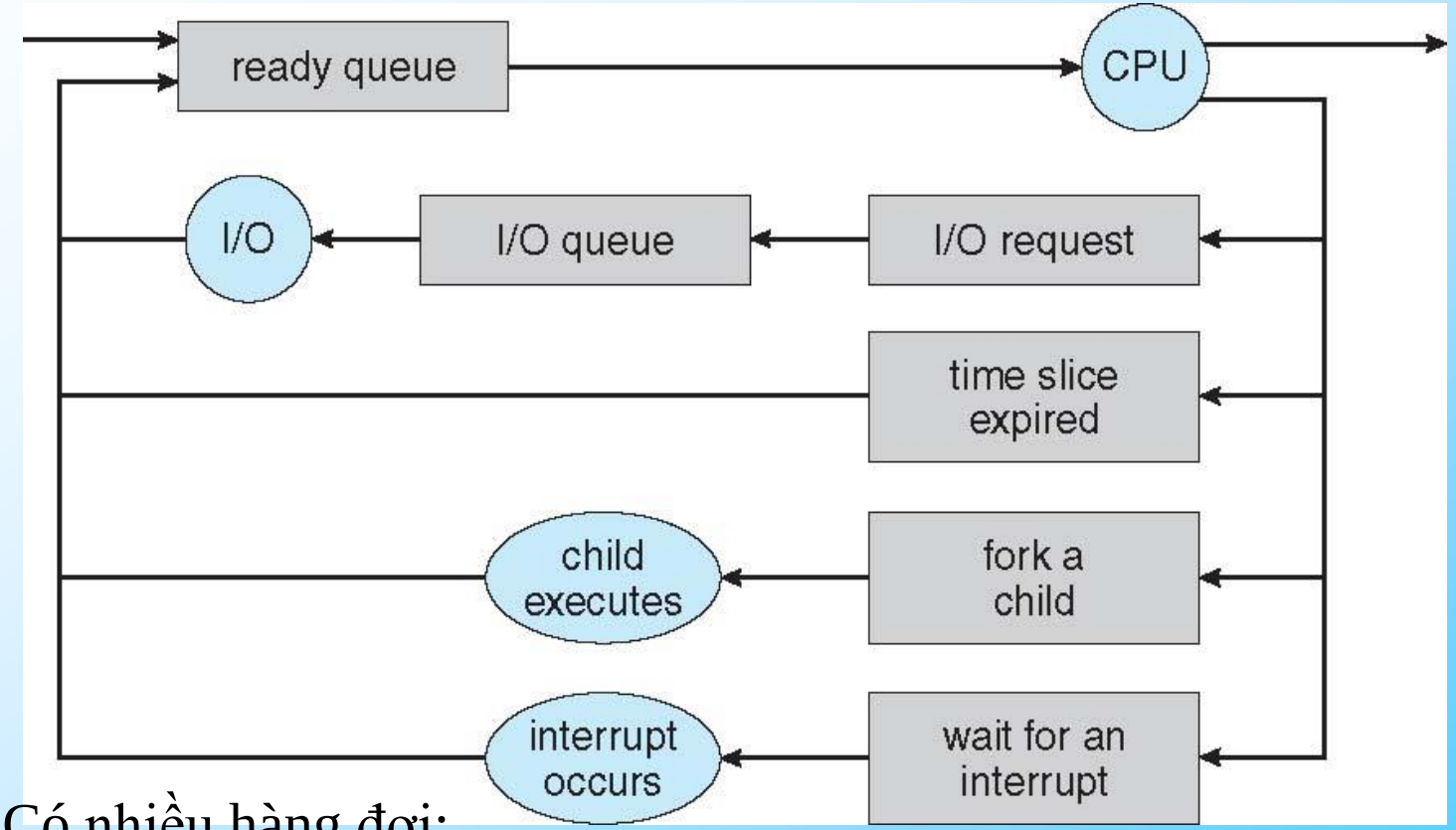


- ***Trạng thái của tiến trình***
  - **new**: Tiến trình vừa được tạo (chạy chương trình)
  - **ready**: Tiến trình sẵn sàng để chạy (đang chờ cấp CPU)
  - **running**: Tiến trình đang chạy (thi hành lệnh)
  - **waiting**: Tiến trình chờ đợi một sự kiện
  - **terminated**: Tiến trình kết thúc thi hành lệnh

Hàng đợi tiến trình



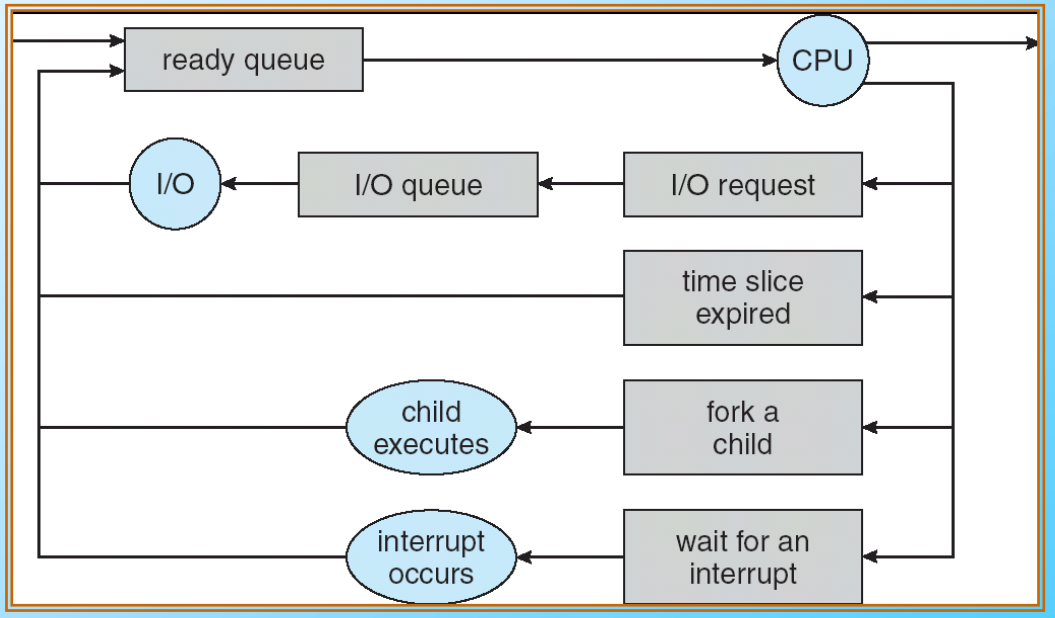
Biểu diễn của lập lịch tiến trình



- Có nhiều hàng đợi:
  - ready queue: hàng đợi chứa các tiến trình sẵn sàng chạy
  - I/O queue: hàng đợi chứa các tiến trình sẵn sàng thi hành I/O
- Lựa chọn tiến trình nào → ***điều phối tiến trình***

Lập lịch các tiến trình

- Tình huống:
  - Có nhiều tiến trình nhưng tại một thời điểm chỉ có một tiến trình có thể được thực thi (trạng thái là running)
  - Vấn đề: chọn tiến trình nào để thực thi ở bước kế tiếp (từ trạng thái **ready** chuyển sang trạng thái **running**)
- ***Lập lịch*** là thao tác quyết định tiến trình nào được quyền thực thi.

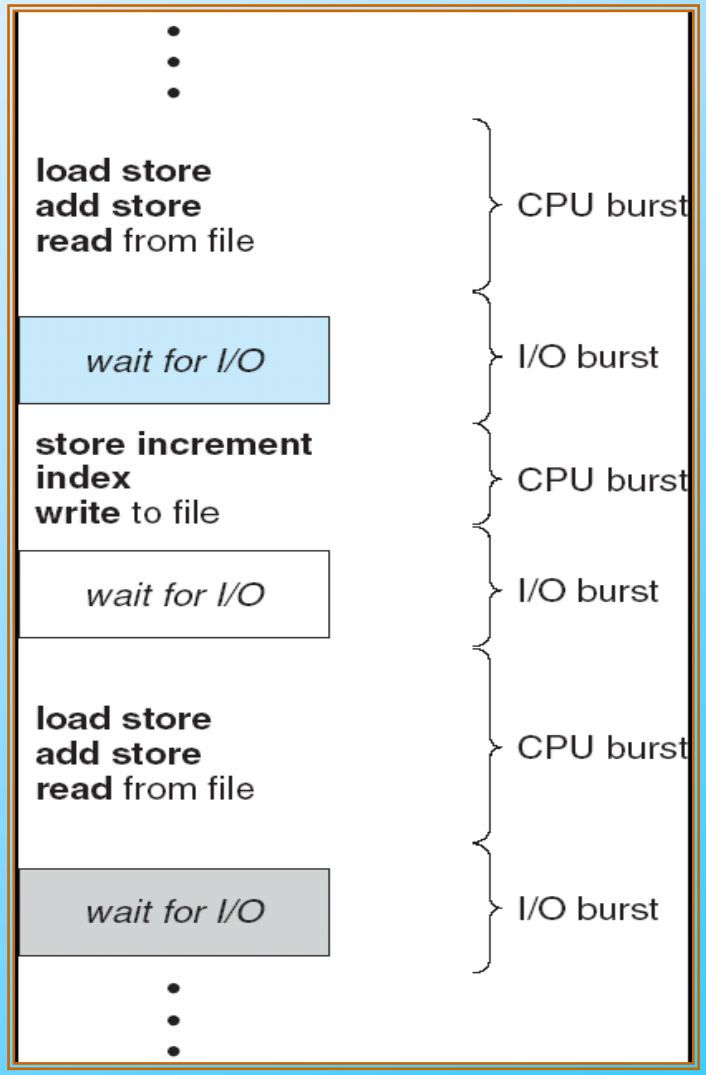


Lập lịch các tiến trình (2/3)

- Một tiến trình chỉ có một luồng/tiểu trình (HeaveWeight Process)
  - *Lưu ý*: Hệ điều hành lập lịch ở mức tiểu trình
- Các tiến trình là độc lập với nhau
  - Không có hợp tác, chia sẻ tài nguyên với nhau
  - Các tiến trình hợp tác → đồng bộ hóa tiến trình (bài kế)
- Mô hình thực thi của các tiến trình là một chuỗi thời gian sử dụng CPU và I/O xen kẽ nhau
  - Chỉ tập trung vào lập lịch cho thời gian CPU

Lập lịch các tiến trình
Chu trình CPU – I/O (3/3)

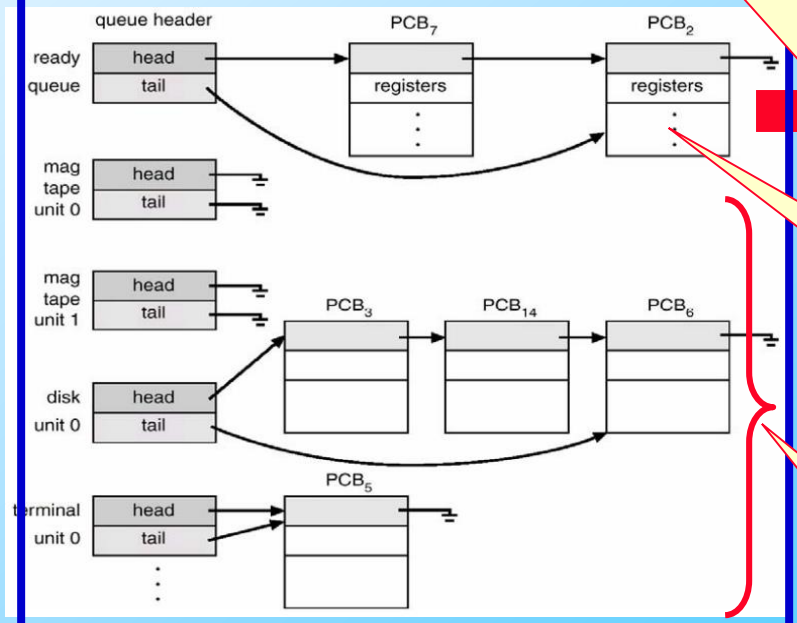
- Điều phối CPU thành công phụ thuộc vào việc thực thi tiến trình theo chu kỳ (CPU → chờ nhập/xuất).
- Chương trình sẽ sử dụng CPU trong một khoảng thời gian.
- Sau đó thi hành thao tác I/O
- Tiếp tục sử dụng CPU, ...



Các bộ điều phối

Bộ định thời CPU chọn tiến trình để CPU thực thi

RAM



CPU

DISK

1 2 6

3 4 5

Hàng đợi sẵn sàng

Hàng đợi nhập/xuất

Bộ định thời công việc sẽ chọn tiến trình đưa vào hệ thống

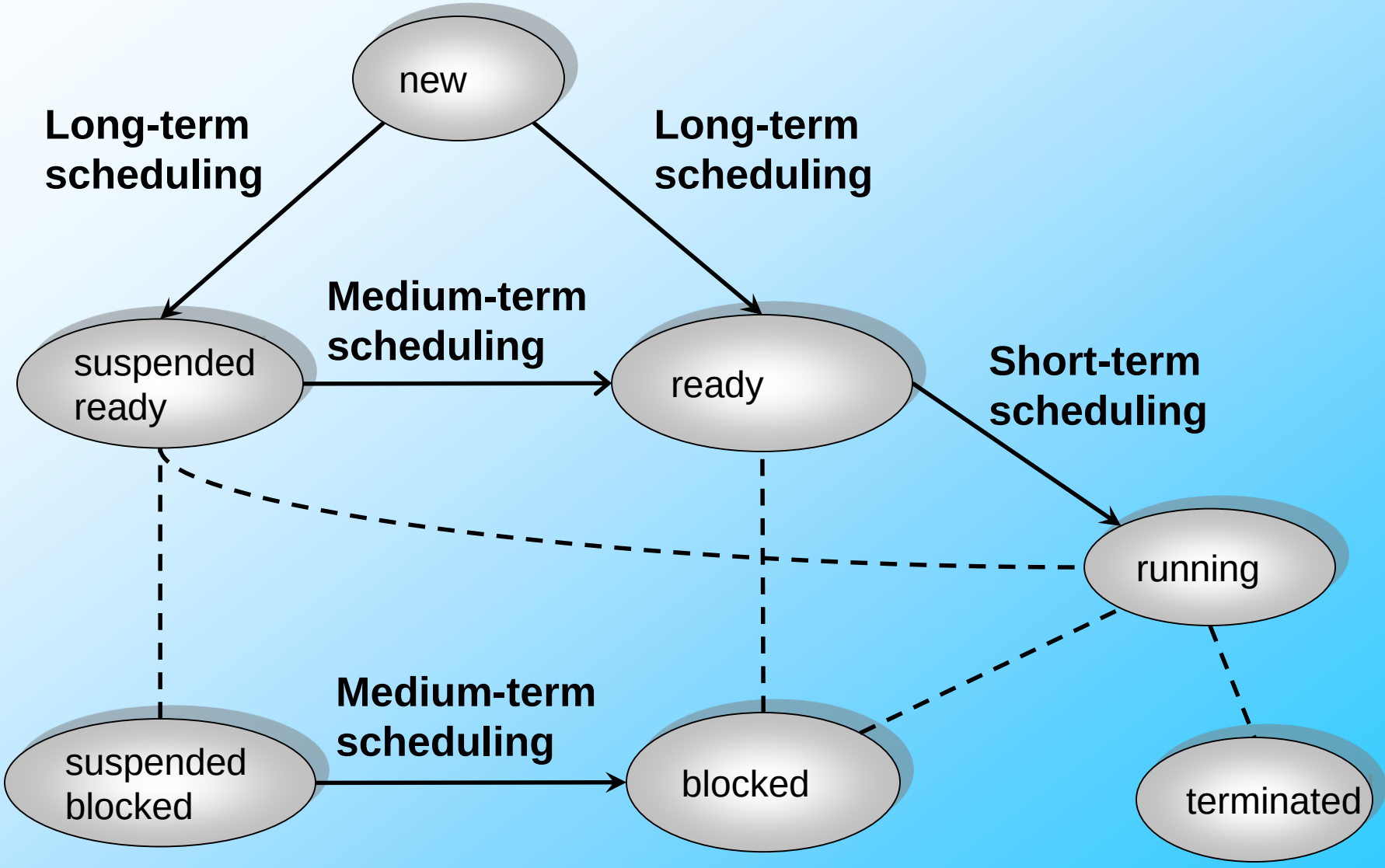
Các bộ điều phối(2/5)

- **Bộ định thời dài** (long-term scheduler) hay **bộ định thời công việc** (job scheduler): chọn các tiến trình từ vùng đệm (đĩa) và nạp chúng vào bộ nhớ để thực thi.
- **Bộ định thời ngắn** (short-term scheduler) hay **bộ định thời CPU:** chọn một tiến trình từ các tiến trình sẵn sàng thực thi và cấp phát CPU cho tiến trình đó.

Sự khác nhau:

- Bộ định thời CPU chọn tiến trình mới cho CPU thường xuyên. Thường thực thi ít nhất 1 lần trong mỗi 100 mili giây, mất 10 mili giây để quyết định việc thực thi.
- Bộ định thời công việc thực thi ít thường xuyên hơn. Có vài phút giữa việc tạo các tiến trình mới trong hệ thống. Nó điều khiển mức độ đa chương – (tốc độ tạo tiến trình bằng với tốc độ tiến trình rời hệ thống).

Các bộ điều phối(3/5)



Các bộ điều phối(4/5)

- *Long-term scheduling*
  - Xác định chương trình nào được chấp nhận nạp vào hệ thống để thực thi
  - Điều khiển mức độ multiprogramming của hệ thống
  - Long term scheduler thường cố gắng duy trì xen lẫn CPU-bound và I/O-bound process
- *Medium-term scheduling*
  - Process nào được đưa vào (swap in), đưa ra khỏi (swap out) bộ nhớ chính
  - Được thực hiện bởi phần quản lý bộ nhớ và được thảo luận ở phần quản lý bộ nhớ.

Các bộ điều phối(5/5)

Short term scheduling

- Xác định process nào trong ready queue sẽ được chiếm CPU để thực thi kế tiếp (còn được gọi là định thời CPU, CPU scheduling)
- Short term scheduler còn được gọi với tên khác là *dispatcher*
- Bộ định thời short-term được gọi mỗi khi có một trong các sự kiện/interrupt sau xảy ra:
  - Ngắt thời gian (clock interrupt)
  - Ngắt ngoại vi (I/O interrupt)
  - Lời gọi hệ thống (operating system call)
  - Signal

***Chương này tập trung* bộ lập lịch ngắn hạn** (Short-Time Scheduling)

Cơ chế điều phối

Độc quyền

```
while (true) {
   save state      Pcur
   Scheduler.NextP() → Pnext
   load state      Pnext
   resume          Pnext
   wait for        Pnext
}
```

Không độc quyền

```
while (true) {
   interrupt       Pcur
   save state      Pcur
   Scheduler.NextP() → Pnext
   load state      Pnext
   resume          Pnext
}
```

Cơ chế điều phối(2/2)

- **Preemptive (không độc quyền)**: Công việc đang thực thi có thể bị ngắt và chuyển vào trạng thái Ready.
- **Non-preemptive (độc quyền)**: một khi tiến trình ở trong trạng thái Running, nó sẽ tiếp tục thực thi cho đến khi kết thúc hoặc bị block vì I/O hay các dịch vụ của hệ thống.

4.2 – Các tiêu chuẩn điều phối

Mục tiêu điều phối

- Hiệu quả (Efficiency)
  - ⬇ Thời gian
    - ⬇ Đáp ứng (Response time)
    - ⬇ Hoàn tất (Turnaround Time = Tquit -Tarrive):
    - ⬇ Chờ (Waiting Time = T in Ready) :
  - ⬆ Thông lượng (Throughput = # jobs/s)
    - ⬆ Hiệu suất Tài nguyên
    - ⬇ Chi phí chuyển đổi
- Công bằng (Fairness): Tất cả các tiến trình đều có cơ hội nhận CPU

Các tiêu chuẩn điều phối

- Các giải thuật điều phối khác nhau có các thuộc tính khác nhau và có xu hướng thiên vị cho một loại tiến trình. Nhiều tiêu chuẩn được đề nghị để so sánh các giải thuật điều phối.
- Các tiêu chuẩn gồm:
  - Việc sử dụng CPU
  - Thông lượng
  - Thời gian hoàn thành
  - Thời gian chờ
  - Thời gian đáp ứng

Việc sử dụng CPU

- Chúng ta muốn giữ CPU bận nhiều nhất có thể.
- Việc sử dụng CPU có thể từ 0 đến 100%.
- Trong hệ thống thực, nó nên nằm trong khoảng từ 40% (cho hệ thống được nạp tải nhẹ) tới 90% (cho hệ thống được nạp tải nặng).

➔ Tối ưu hóa CPU (Efficient)

Thông lượng

- Nếu CPU bận thực thi các tiến trình thì công việc đang được thực hiện.
- Thước đo của công việc là số lượng tiến trình được hoàn thành trên một đơn vị thời gian gọi là *thông lượng* (throughput) → **tối đa hóa**.
- Đối với các tiến trình dài, tỉ lệ này có thể là 1 tiến trình trên 1 giờ; đối với các giao dịch ngắn, thông lượng có thể là 10 tiến trình trên giây.

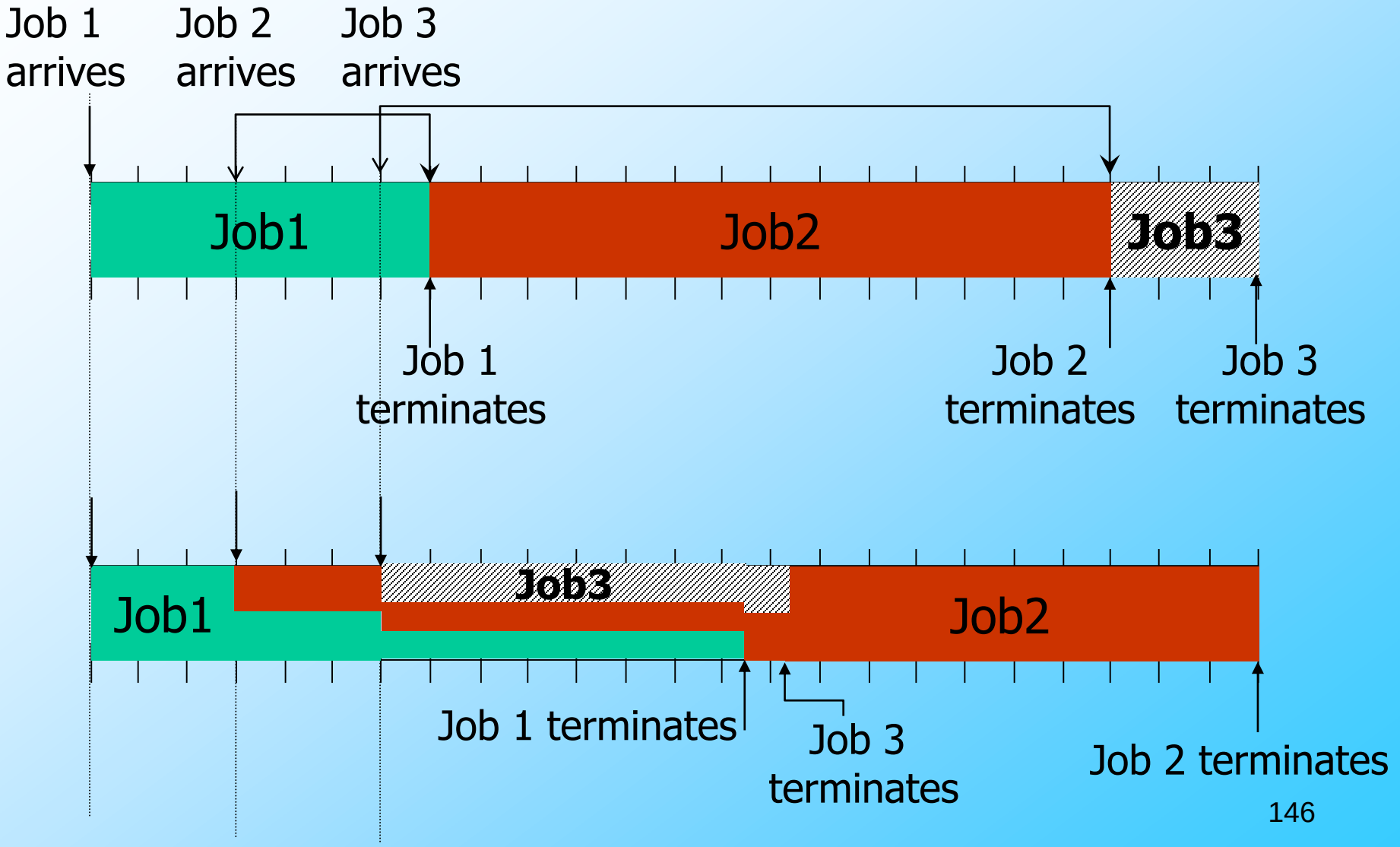
Thời gian hoàn thành

- Một chuẩn quan trọng là mất bao lâu để thực thi một tiến trình.
- Khoảng thời gian từ thời điểm khởi tạo tiến trình tới khi tiến trình hoàn thành được gọi là *thời gian hoàn thành* (turnaround time).
- Thời gian hoàn thành là tổng các thời gian chờ đưa tiến trình vào bộ nhớ, chờ ở hàng đợi sẵn sàng, thực thi CPU và thực hiện nhập/xuất.

→Tối thiểu hóa ***thời gian lưu lại trong hệ thống*** (turn around time)

- $T_{turn\ around} = T_{kết\ thúc} - T_{bắt\ đầu}$

Turn around time



Thời gian chờ

- Giải thuật điều phối CPU không ảnh hưởng lượng thời gian tiến trình thực thi hay thực hiện nhập/xuất; nó chỉ ảnh hưởng lượng thời gian một tiến trình phải chờ trong hàng đợi sẵn sàng.
- *Thời gian chờ* (waiting time) là tổng thời gian chờ trong hàng đợi sẵn sàng.

Thời gian đáp ứng

- Một thước đo khác là thời gian từ lúc gởi yêu cầu cho tới khi đáp ứng đầu tiên được tạo ra.
- Thước đo này được gọi là *thời gian đáp ứng* (response time), là lượng thời gian mất đi từ lúc bắt đầu đáp ứng nhưng không là thời gian mất đi để xuất ra đáp ứng đó.

→ Tối thiểu hóa ***thời gian phản hồi*** (response time)

4.3 – Các giải thuật điều phối

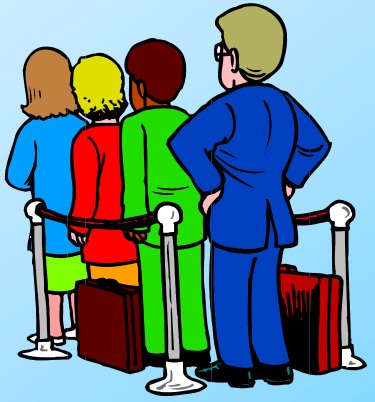
Các giải thuật

- Đến trước phục vụ trước (first come first served - FCFS)
- Ưu tiên công việc ngắn nhất (shortest job first - SJF)
- Điều phối theo độ ưu tiên (priority-scheduling)
- Điều phối luân phiên (round robin - RR)
- Hàng đợi nhiều cấp (multilevel queue)
- Hàng đợi phản hồi đa cấp (multilevel feedback queue)

First-Come, First-Served (FCFS)

- Lập lịch các công việc theo thứ tự xuất hiện của chúng.
  - Off-line FCFS lập lịch theo thứ tự xuất hiện trong dữ liệu đầu vào của nó
- Thi hành lần lượt mỗi công việc cho đến khi hoàn thành
  - Nguyên thủy: hoàn thành kể cả tính I/O
  - Hiện đại: dừng lại khi bị block (gặp I/O)
- Có cả on-line lẫn off-line
- Đơn giản, dùng làm cơ sở để phân tích các phương pháp khác
- Thời gian phản hồi kém

FCFS (2/3)



- Ví dụ:

Process	Burst Time
P_1	24
P_2	3
P_3	3

  - VD: 3 tiến trình vào hàng đợi theo thứ tự: P_1 , P_2 , P_3
    Sơ đồ Gantt:

    Thời gian chờ P_1 = 0; P_2 = 24; P_3 = 27
  - Thời gian chờ trung bình: (0 + 24 + 27)/3 = 17
  - Thời gian hoàn thành trung bình: (24 + 27 + 30)/3 = 27
- *Điểm yếu:* Các tiến trình có thời gian CPU ngắn vào sau tiến trình có thời gian CPU dài.

FCFS (3/3)

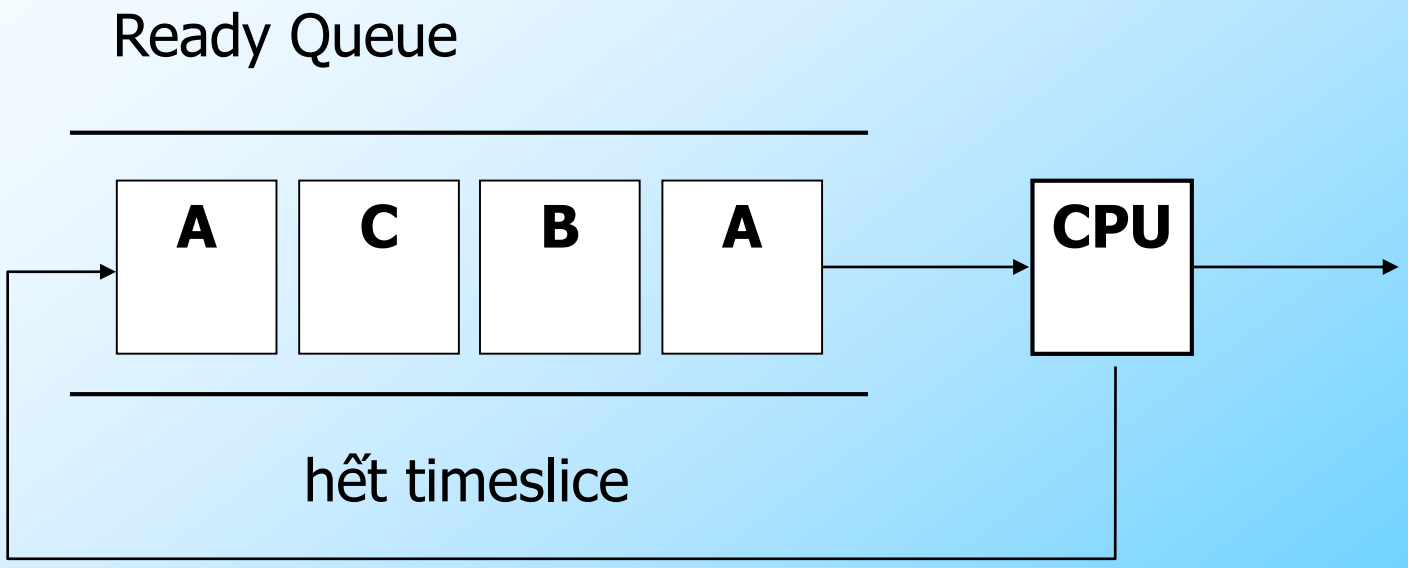
- Điểm yếu:
  - Giả sử vào hàng đợi theo thứ tự: P_2 , P_3 , P_1
    Sơ đồ Gantt:

  - Thời gian chờ $P_1 = 6; P_2 = 0, P_3 = 3$
  - Thời gian chờ trung bình: (6 + 0 + 3)/3 = 3
  - Thời gian hoàn thành trung bình: (3 + 6 + 30)/3 = 13
- Trường hợp 2:
  - Thời gian chờ trung bình tốt hơn (3 < 17)
  - Thời gian hoàn thành trung bình tốt hơn (13 < 27)

Round Robin (RR)

- Mô hình FCFS: Không tốt cho những tiến trình thời gian ngắn!
  - Phụ thuộc hoàn toàn vào thứ tự
- ***Mô hình Round Robin***
  - Mỗi tiến trình sẽ nhận được một khoảng thời gian sử dụng CPU khá nhỏ (*time quantum*), thường là 10-100 milli giây
  - Sau khi khoảng thời gian này kết thúc, tiến trình sẽ bị cưỡng chế chuyển vào hàng đợi sẵn sàng (không cho dùng CPU nữa).
  - Giả sử có *n* tiến trình trong hàng đợi và time quantum là $q \Rightarrow$
    - Mỗi lần chạy tiến trình sẽ có tối đa *q* đơn vị thời gian
    - Không có tiến trình nào phải đợi quá $(n-1)q$ đơn vị thời gian
- Đánh giá hiệu năng
  - *q* lớn $\Rightarrow$ FCFS
  - *q* nhỏ $\Rightarrow$ thời gian overhead lớn → không hiệu quả

Round Robin

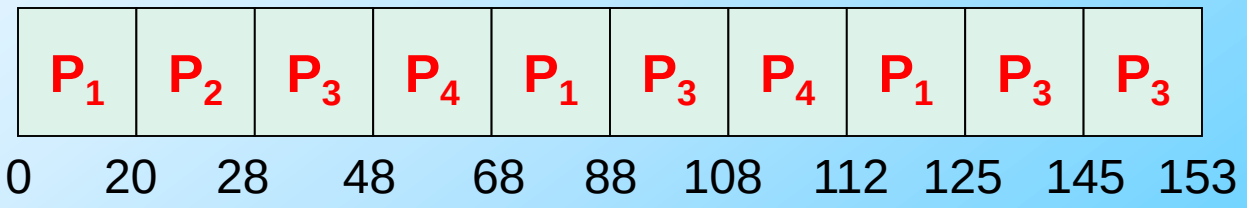


Round Robin (q=20)

- **Ví dụ:**

Process	Burst Time
P_1	53
P_2	8
P_3	68
P_4	24

  - Sơ đồ Gantt:



  - Thời gian chờ

P_1=(68-20)+(112-88)=72

P_2=(20-0)=20

P_3=(28-0)+(88-48)+(125-108)=85

P_4=(48-0)+(108-68)=88

  - Thời gian chờ trung bình = (72+20+85+88)/4=66¼
  - Thời gian hoàn thành trung bình = (125+28+153+112)/4 = 104½

- **Đánh giá:**
  - Tốt cho các tiến trình có thời gian CPU ngắn
  - Thêm thời gian chuyển đổi ngữ cảnh cho các tiến trình có thời gian CPU dài

CÂU HỎI

- Ví dụ:

Process	Burst Time
P_1	53
P_2	8
P_3	68
P_4	24

  - Sơ đồ Gant?
  - Thời gian chờ P_1?
    P_2 ?
    P_3?
    P_4?
  - Thời gian chờ trung bình ?
  - Thời gian hoàn thành trung bình ?
- Khi dùng Round Robin với q =40, First Come- First Service (FCFS) trong trường hợp xấu nhất với 4 process trên.

So sánh FCFS và Round Robin

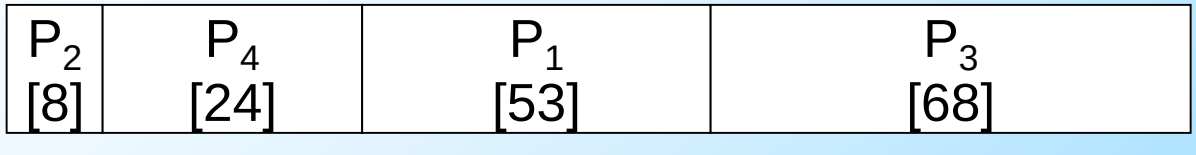
- Giả sử thời gian chuyển đổi ngữ cảnh không đáng kể, RR hay FCFS tốt hơn?
- Xét ví dụ: 10 tiến trình, mỗi tiến trình sử dụng 100s CPU
  q = 1s
  Tất cả tiến trình vào hàng đợi cùng 1 thời điểm
- Thời gian hoàn thành:

P #	FCFS	RR
1	100	991
2	200	992
…	…	…
9	900	999
10	1000	1000

  - Cả RR và FCFS đều hoàn thành 10 tiến trình tại cùng 1 thời điểm
  - Thời gian phản hồi của RR rất tệ!
  - Không nên dùng trong trường hợp các tiến trình có thời gian sử dụng CPU gần nhau

Round Robin với q khác nhau

Best FCFS:



	Quantum	P_1	P_2	P_3	P_4	Average
Wait Time	**Best FCFS**	**32**	**0**	**85**	**8**	**31¼**
	Q = 1	**84**	**22**	**85**	**57**	**62**
	Q = 5	**82**	**20**	**85**	**58**	**61¼**
	Q = 8	**80**	**8**	**85**	**56**	**57¼**
	Q = 10	**82**	**10**	**85**	**68**	**61¼**
	Q = 20	**72**	**20**	**85**	**88**	**66¼**
	Worst FCFS	**68**	**145**	**0**	**121**	**83½**
Completion Time	**Best FCFS**	**85**	**8**	**153**	**32**	**69½**
	Q = 1	**137**	**30**	**153**	**81**	**100½**
	Q = 5	**135**	**28**	**153**	**82**	**99½**
	Q = 8	**133**	**16**	**153**	**80**	**95½**
	Q = 10	**135**	**18**	**153**	**92**	**99½**
	Q = 20	**125**	**28**	**153**	**112**	**104½**
	Worst FCFS	**121**	**153**	**68**	**145**	**121¾**

Sự cưỡng chế (Pre-emptive)

- ***Sự cưỡng chế*** là hành động dừng một công việc đang chạy để lập lịch cho công việc khác

→ chuyển đổi ngữ cảnh

- Ví dụ: Tiến trình P1 đang chạy (sử dụng CPU)→ dừng tiến trình P1 lại (chuyển ra hàng đợi ready) và giao CPU cho tiến trình P2 nào đó.
- Lưu ý: Tiến trình P1 không bị dừng bởi thao tác I/O hoặc các sự kiện khác

Sự cưỡng chế (2/2)

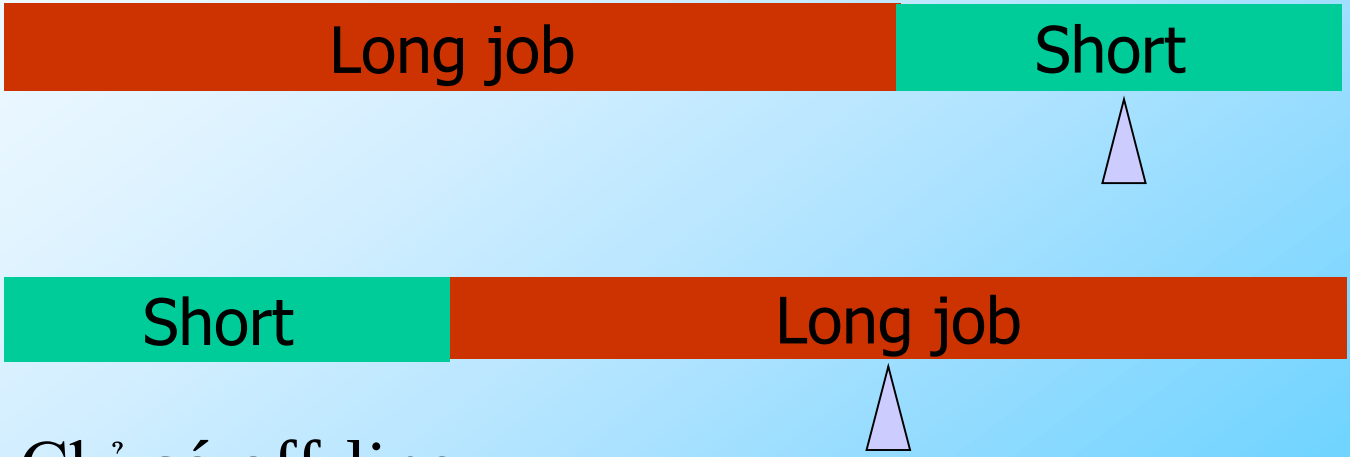
- Thuật toán SST on-line
  - Tương thích với sự thay đổi điều kiện
    - VD: có công việc mới
  - Bổ sung cho việc thiếu thông tin
    - Vd: thời gian chạy
- Sự cưỡng chế theo chu kỳ giúp hệ thống nằm trong tầm kiểm soát
- Cải thiện tính công bằng

Các thuật toán cải tiến FCFS

- Xét trường hợp tốt nhất của FCFS: tiến trình thời gian ngắn vào trước, tiến trình thời gian dài vào sau
- ***Shortest Job First (SJF)***:
  - Chọn tiến trình có thời gian chạy là ít nhất (không phụ thuộc thứ tự vào)
  - Còn gọi là "Shortest Time to Completion First" (STCF)
- ***Shortest Remaining Time First (SRTF)***:
  - Là một phiên bản SJF có cưỡng chế (Preemptive version of SJF): nếu có tiến trình mới vào và thời gian sử dụng CPU ít hơn thời gian còn lại của tiến trình đang chiếm CPU thì dừng tiến trình đang chạy và chuyển quyền cho tiến trình mới vào.
  - Còn gọi là "Shortest Remaining Time to Completion First" (SRTCF)
- Ý tưởng chính
  - Cho phép công việc có thời gian thi hành CPU ngắn ra ngoài CPU càng nhanh càng tốt
  - Kết quả là thời gian phản hồi trung bình sẽ tốt hơn

Shortest Job First (SJF)

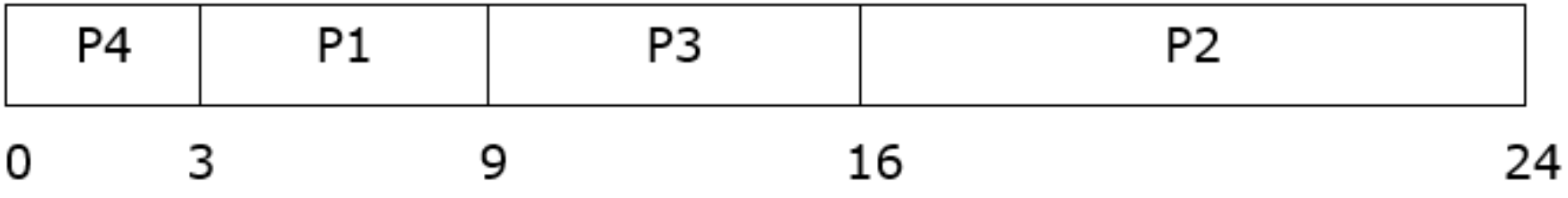
- Công việc có thời gian ít nhất sẽ được thi hành trước
- Độ đo thời gian phản hồi là tốt nhất



- Chỉ có off-line
  - Tất cả các công việc và thời gian thi hành phải được biết trước

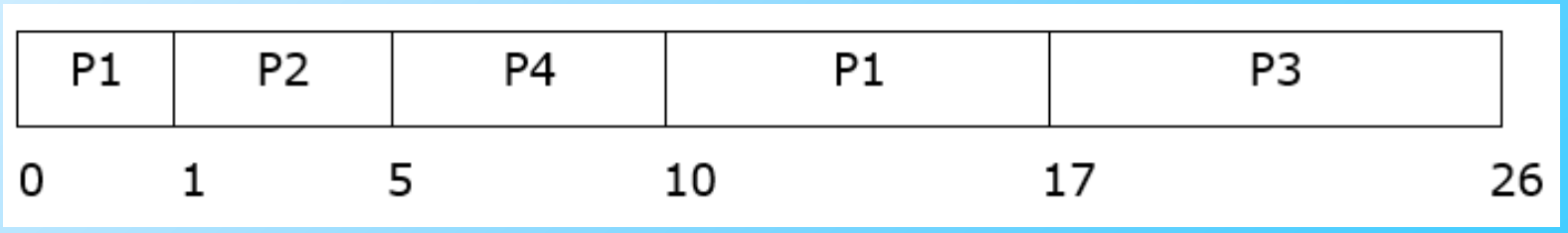
Shortest Job First (2/3)

Tiến trình	Thời gian xử lý
P1	6
P2	8
P3	7
P4	3



Shortest Job First (3/3)

Tiến trình	Thời gian đến	Thời gian xử lý
P1	0	8
P2	1	4
P3	2	9
P4	3	5



Shortest Remaining Time first (SRT)

- Biết: thời gian thi hành công việc
- Không biết: thời điểm công việc bắt đầu (được nạp vào hàng đợi)
- Khi có một công việc mới:

  Nếu thời gian thi hành của nó nhỏ hơn thời gian thi hành còn lại của công việc đang được thi hành hiện tại thì:

  cưỡng chế dừng công việc đang thi hành hiện tại và lập lịch cho công việc vừa được tạo ra

  Ngược lại, tiếp tục công việc hiện tại và chèn công việc mới vào hàng đợi theo thứ tự thời gian còn lại phải thi hành
- Khi công việc hiện tại kết thúc, chọn công việc nằm ở đầu hàng đợi để thi hành

So sánh SJF, SRTF, FCFS, RR

- **SJF vs SRTF**
  - Tốt nhất để tối thiểu hóa thời gian phản hồi trung bình. (SJF: non-preemptive, SRTF: preemptive)
  - SRTF ít nhất là tương đương với SJF
- **SRTF vs FCFS và RR**
  - Nếu thời gian sử dụng của các tiến trình là như nhau → SRTF = FCFS
  - Nếu thời gian sử dụng của các tiến trình là biến động lớn → SRTF, RR giúp cho các tiến trình có thời gian ngắn không chờ quá lâu.

So sánh SJF, SRTF, FCFS, RR (2/3)

- SRTF có thể làm phát sinh trường hợp ***"đói CPU" (starvation)*** cho các tiến trình có thời gian sử dụng CPU tương đối lâu
  - Ví dụ: Trường hợp các tiến trình có thời gian sử dụng ngắn liên tục được đưa vào. → tiến trình có thời gian sử dụng dài sẽ không được phép sử dụng CPU → ***tình trạng đói CPU (starvation).***
- Cả 4 phương pháp đều yêu cầu phải biết thời gian mà một tiến trình sẽ dùng CPU.
  - Làm sao biết?

So sánh SJF, SRTF, FCFS, RR (3/3)

- Dùng SRTF để làm cơ sở đánh giá các phương pháp khác (vì là phương pháp tối ưu) về thời gian phản hồi trung bình.
- Ưu điểm
  - Thời gian phản hồi trung bình của SRTF là tốt nhất.
- Khuyết điểm
  - Phải dự đoán thời gian sử dụng CPU của tiến trình
  - Không công bằng

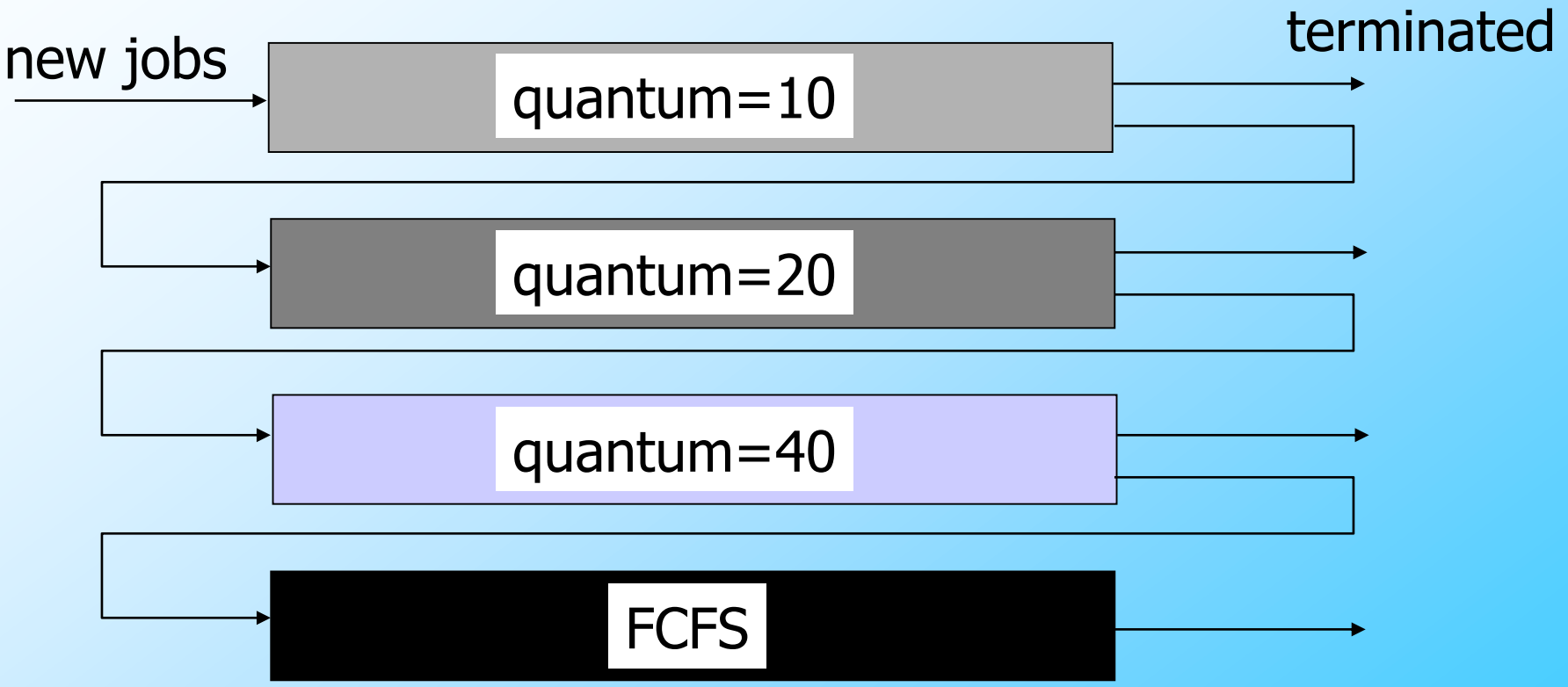
Dự đoán thời gian sử dụng CPU

- Yêu cầu người dùng nhập vào
  - Khó khả thi: người dùng không biết
  - Người dùng có thể đưa vào thời gian thi hành ngắn để mong kết thúc công việc sớm
- ***Adaptive (thích ứng)***: Dự đoán tương lai bằng cách quan sát quá khứ.
  - Nếu trong quá khứ tiến trình (chương trình) thường dùng CPU nhiều (CPU-bound) thì có thể trong tương lai nó sẽ sử dụng nhiều.
  - Nếu trong quá khứ tiến trình (chương trình) thường thao tác I/O (I/O bound) thì có thể trong tương lai nó sẽ sử dụng CPU ít.

Dự đoán thời gian sử dụng CPU (2/2)

- Gọi t_i là thời gian sử dụng CPU tại lần thứ i.
- Ý tưởng thời gian sử dụng CPU tại lần thứ n là:
  - $T_n = f(t_{n-1}, t_{n-2}, ...)$
  - $T_n = \alpha t_{n-1} + (1-\alpha)T_{n-1} \; (\alpha \in [0,1])$

Hàng đợi phản hồi đa mức (Multilevel feedback queues)



Hàng đợi phản hồi đa mức (Multilevel feedback queues)

- Độ ưu tiên được ngầm định trong mô hình này
- Rất linh hoạt
- Tình trạng đói CPU có thể có

Nhiều công việc ngắn vào => công việc dài sẽ bị "đói"

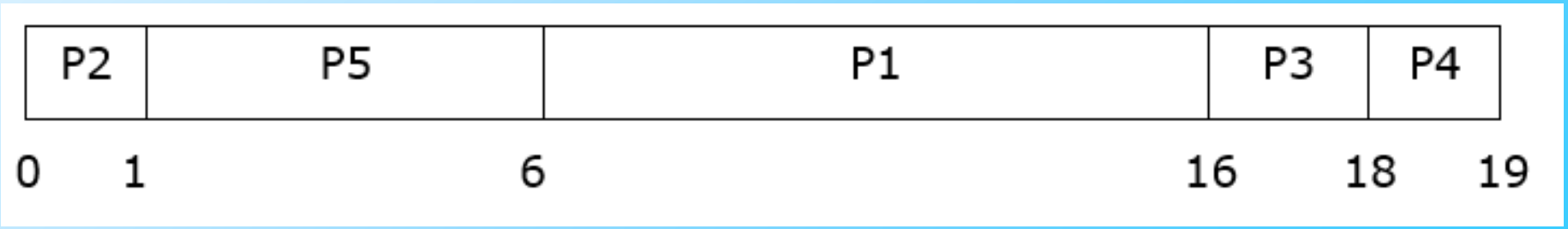
- Giải pháp:
  - Để nguyên
  - Lão hóa (aging)

Điều phối theo độ ưu tiên

- Độ ưu tiên được gán với mỗi tiến trình và CPU được cấp phát tới tiến trình có độ ưu tiên cao nhất.
- Các tiến trình có độ ưu tiên bằng nhau được điều phối theo FCFS.
- Giải thuật SJF là giải thuật ưu tiên đơn giản ở đó độ ưu tiên là nghịch đảo với chu kỳ CPU được đoán tiếp theo. Chu kỳ CPU lớn hơn có độ ưu tiên thấp hơn và ngược lại.

Điều phối theo độ ưu tiên

Tiến trình	Thời gian xử lý	Độ ưu tiên
P1	10	3
P2	1	1
P3	2	4
P4	1	5
P5	5	2



Tiến trình	Thời điểm vào RL	Độ ưu tiên	Thời gian xử lý
P1	0	3	24
P2	1	1	3
P3	2	2	3

Sử dụng thuật giải độc quyền, thứ tự cấp phát CPU như sau :

P1	P2	P3	
0	24	27	30

Thời gian chờ?

Sử dụng thuật giải không độc quyền, thứ tự cấp phát CPU như sau :

P1	P2	P3	P1	
0	1	4	7	30

Thời gian chờ?

Tóm tắt

- Tiến trình = một thể hiện của việc thi hành một chương trình.
- Đa chương = nhiều tiến trình có thể cùng được thi hành. Tại mỗi thời điểm chỉ có một tiến trình ở trạng thái được thi hành.
- Lập lịch = quyết định tiến trình nào sẽ được chuyển trạng thái từ sẵn sàng sang chạy.

Tóm tắt(2/3)

- FCFS: Vào trước sẽ được cấp phát CPU trước
  - Ưu: đơn giản
  - Khuyết: tiến trình ngắn sẽ chờ tiến trình dài.
- Round Robin: Cấp mỗi tiến trình một khoảng thời gian định trước (quantumn) khi nó nhận được CPU.
  - Ưu: Các tiến trình ngắn sẽ kết thúc nhanh chóng
  - Khuyết: tiến trình có thời gian sử dụng CPU gần nhau → không hiệu quả.

Tóm tắt (3/3)

- SJF/SRTF: Cấp phát cho tiến trình có thời gian thi hành/thời gian còn lại là ít nhất.
  - Ưu: thời gian phản hồi trung bình là tốt nhất.
  - Khuyết: khó dự đoán thời gian sử dụng CPU, không công bằng.
- Multi-level feedback: sử dụng nhiều hàng đợi với độ ưu tiên khác nhau. Tự động chuyển đổi mức độ ưu tiên của các tiến trình.

CÂU HỎI ÔN TẬP BÀI 4

1. Mục tiêu của việc điều phối tiến trình.
2. Hãy vẽ sơ đồ tổ chức điều phối tiến trình trong trường hợp các tiến trình tham chiếu tới 3 tài nguyên.
3. Hãy trình bày vắn tắt các chiến lược điều phối tiến trình (không vẽ sơ đồ minh họa).

CÂU HỎI ÔN TẬP BÀI 4

quantum=2, điều phối theo cơ chế không độc quyền, có các tiến trình đang hoạt động như bảng sau:

Tiến trình	**P0**	**P1**	**P2**	**P3**	**P4**
Thời gian CPU xử lý	10	1	2	1	**5**
Thời điểm vào Ready List	0	1	2	3	**4**
Độ ưu tiên	**3**	**1**	**3**	**4**	**2**

Câu hỏi

- Có mấy tiêu chuẩn điều phối tiến trình
  a. 1
  b. 3
  c. 5
  d. 6

Câu hỏi

- Điều phối độc quyền là
  A. Có thể chuyển tiến trình đang chạy đưa về RL
  B. Có thể chuyển tiến trình đang chạy đưa về WL
  C. Không thể dừng tiến trình đang chạy
  D. Không có câu nào đúng

Câu hỏi

Điều phối Round Robin là

- Điều phối độc quyền
- Điều phối theo thứ tự ưu tiên
- Điều phối theo thời gian
- Không có cáu nào đúng

Nhắc lại Kiến thức

- **LIÊN LẠC GIỮA CÁC TIẾN TRÌNH**
  - Nhu cầu liên lạc giữa các tiến trình
  - Các vấn đề nảy sinh trong việc liên lạc
- **Các Cơ Chế Thông Tin Liên lạc**
  - Tín hiệu (Signal)
  - Pipe
  - Vùng nhớ chia sẻ
  - Trao đổi thông điệp (Message)
  - Sockets

LIÊN LẠC GIỮA CÁC TIẾN TRÌNH

- Nhu cầu liên lạc giữa các tiến trình
  - **Chia sẻ thông tin:** nhiều tiến trình có thể cùng quan tâm đến những dữ liệu nào đó, do vậy hệ điều hành cần cung cấp một môi trường cho phép sự truy cập đồng thời đến các dữ liệu chung.
  - **Hợp tác hoàn thành tác vụ:** đôi khi để đạt được một sự xử lý nhanh chóng, người ta phân chia một tác vụ thành các công việc nhỏ có thể tiến hành song song.

Các vấn đề nảy sinh trong việc liên lạc giữa các tiến trình

- *Liên kết tường minh hay tiềm ẩn (explicit naming/implicit naming)* : tiến trình có cần phải biết tiến trình nào đang trao đổi hay chia sẻ thông tin với nó? Mối liên kết được gọi là tường minh khi được thiết lập rõ ràng , trực tiếp giữa các tiến trình, và là tiềm ẩn khi các tiến trình liên lạc với nhau thông qua một qui ước ngầm nào đó.
- *Liên lạc theo chế độ đồng bộ hay không đồng bộ (blocking / non-blocking)*: khi một tiến trình trao đổi thông tin với một tiến trình khác, các tiến trình có cần phải đợi cho thao tác liên lạc hoàn tất rồi mới tiếp tục các xử lý khác ?
- *Liên lạc giữa các tiến trình trong hệ thống tập trung và hệ thống phân tán*: cơ chế liên lạc giữa các tiến trình trong cùng một máy tính có sự khác biệt với việc liên lạc giữa các tiến trình giữa những máy tính khác nhau?

Các Cơ Chế Thông Tin Liên lạc

- **Tín hiệu (Signal)**
- **Pipe**
- **Vùng nhớ chia sẻ**
- **Trao đổi thông điệp (Message)**
- **Sockets**

Tín hiệu (Signal)

Tín hiệu là một cơ chế phần mềm tương tự như các ngắt cứng tác động đến các tiến trình. Một tín hiệu được sử dụng để thông báo cho tiến trình về một sự kiện nào đó xảy ra. Có nhiều tín hiệu được định nghĩa, mỗi một tín hiệu có một ý nghĩa tương ứng với một sự kiện đặc trưng.

Một số tín hiệu của UNIX

Tín hiệu	Mô tả
SIGINT	Người dùng nhấn phím DEL để ngắt xử lý tiến trình
SIGQUIT	Yêu cầu thoát xử lý
SIGILL	Tiến trình xử lý một chỉ thị bất hợp lệ
SIGKILL	Yêu cầu kết thúc một tiến trình
SIGFPT	Lỗi floating – point xảy ra (chia cho 0)
SIGPIPE	Tiến trình ghi dữ liệu vào pipe mà không có reader
SIGSEGV	Tiến trình truy xuất đến một địa chỉ bất hợp lệ
SIGCLD	Tiến trình con kết thúc
SIGUSR1	Tín hiệu 1 do người dùng định nghĩa
SIGUSR2	Tín hiệu 2 do người dùng định nghĩa

- Các tín hiệu được gởi đi bởi
  - Phần cứng (ví dụ lỗi do các phép tính số học).
  - Hạt nhân hệ điều hành gởi đến một tiến trình (ví dụ lưu ý tiến trình khi có một thiết bị nhập/xuất tự do).
  - Một tiến trình gởi đến một tiến trình khác (ví dụ tiến trình cha yêu cầu một tiến trình con kết thúc).
  - Người dùng (ví dụ nhấn phím Ctl-C để ngắt xử lý của tiến trình).
- Khi một tiến trình nhận một tín hiệu, nó có thể xử sự theo một trong các cách sau :
  - Bỏ qua tín hiệu.
  - Xử lý tín hiệu theo kiểu mặc định.
  - **Tiếp nhận tín hiệu và xử lý theo cách đặc biệt của tiến trình.**

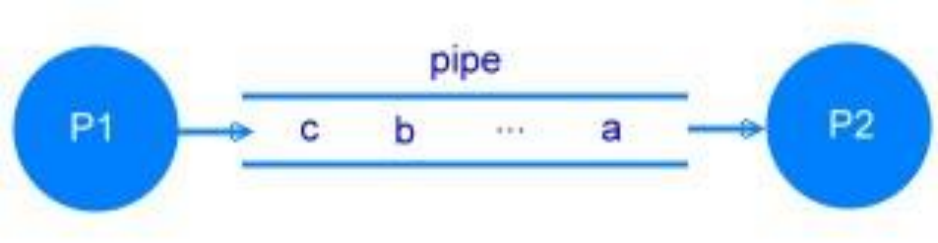
- Liên lạc bằng tín hiệu mang tính chất *không đồng bộ*, nghĩa là một tiến trình nhận tín hiệu không thể xác định trước thời điểm nhận tính hiệu.
- Hơn nữa các tiến trình không thể kiểm tra được sự kiện tương ứng với tín hiệu có thật sự xảy ra?
- Các tiến trình chỉ có thể thông báo cho nhau về một biến cố nào đó, mà không trao đổi dữ liệu theo cơ chế này được

Pipe

•Một pipe là một kênh liên lạc trực tiếp giữa hai tiến trình : dữ liệu xuất của tiến trình này được chuyển đến làm dữ liệu nhập cho tiến trình kia dưới dạng một dòng các byte.

•Khi một pipe được thiết lập giữa hai tiến trình, một trong chúng sẽ ghi dữ liệu vào pipe và tiến trình kia sẽ đọc dữ liệu từ pipe. Thứ tự dữ liệu truyền qua pipe được bảo toàn theo nguyên tắc FIFO.

• Một pipe có kích thước giới hạn (thường là 4096 ký tự).

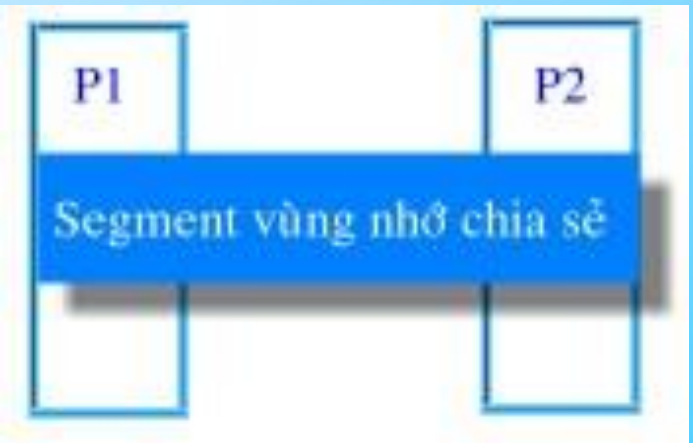


- Một tiến trình chỉ có thể sử dụng một pipe do nó tạo ra hay kế thừa từ tiến trình cha. Hệ điều hành cung cấp các lời gọi hệ thống read/write cho các tiến trình thực hiện thao tác đọc/ghi dữ liệu trong pipe. Hệ điều hành cũng chịu trách nhiệm đồng bộ hóa việc truy xuất pipe trong các tình huống:
  - Tiến trình đọc pipe sẽ bị khóa nếu pipe trống, nó sẽ phải đợi đến khi pipe có dữ liệu để truy xuất.
  - Tiến trình ghi pipe sẽ bị khóa nếu pipe đầy, nó sẽ phải đợi đến khi pipe có chỗ trống để chứa dữ liệu
- Cơ chế này cho phép truyền dữ liệu với cách thức không cấu trúc.
- Ngoài ra, một giới hạn của hình thức liên lạc này là chỉ cho phép kết nối hai tiến trình có quan hệ cha-con, và trên cùng một máy tính.

Vùng nhớ chia sẻ

- Cách tiếp cận của cơ chế này là cho nhiều tiến trình cùng truy xuất đến một vùng nhớ chung gọi là *vùng nhớ chia sẻ (shared memory)*. Không có bất kỳ hành vi truyền dữ liệu nào cần phải thực hiện ở đây, dữ liệu chỉ đơn giản được đặt vào một vùng nhớ mà nhiều tiến trình có thể cùng truy cập được..
- Với phương thức này, các tiến trình chia sẻ một vùng nhớ vật lý thông qua trung gian không gian địa chỉ của chúng. Một vùng nhớ chia sẻ tồn tại độc lập với các tiến trình, và khi một tiến trình muốn truy xuất đến vùng nhớ này, tiến trình phải kết gắn vùng nhớ chung đó vào không gian địa chỉ riêng của từng tiến trình, và thao tác trên đó như một vùng nhớ riêng của mình.

- Đây là phương pháp nhanh nhất để trao đổi dữ liệu giữa các tiến trình. Nhưng phương thức này cũng làm phát sinh các khó khăn trong việc bảo đảm sự toàn vẹn dữ liệu (*coherence*), ví dụ : làm sao biết được dữ liệu mà một tiến trình truy xuất là dữ liệu mới nhất mà tiến trình khác đã ghi? Làm thế nào ngăn cản hai tiến trình cùng đồng thời ghi dữ liệu vào vùng nhớ chung.
- Một khuyết điểm của phương pháp liên lạc này là không thể áp dụng hiệu quả trong các hệ phân tán, để trao đổi thông tin giữa các máy tính khác nhau.



Trao đổi thông điệp (Message)

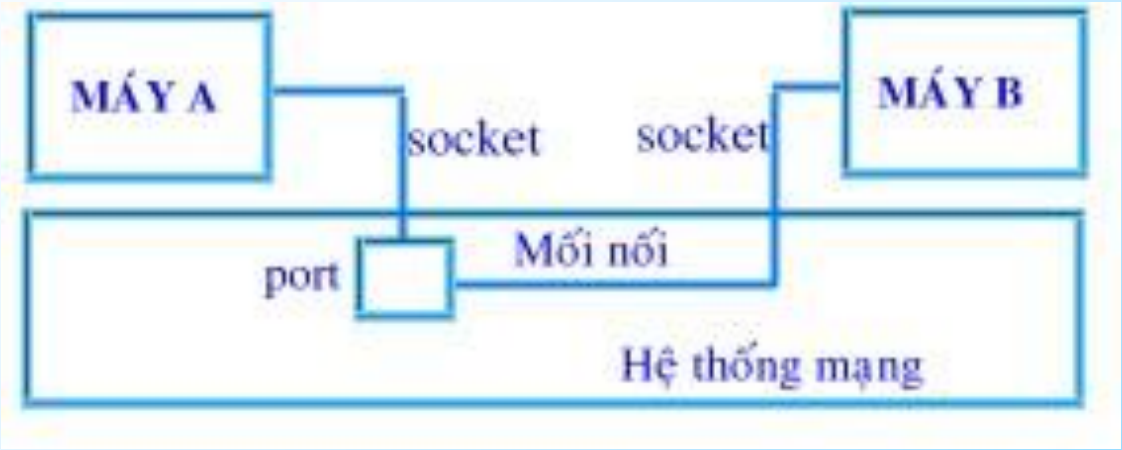
- Hệ điều hành còn cung cấp một cơ chế liên lạc giữa các tiến trình không thông qua việc chia sẻ một tài nguyên chung, mà thông qua việc gởi thông điệp. Để hỗ trợ cơ chế liên lạc bằng thông điệp, hệ điều hành cung cấp các hàm IPC chuẩn (Interprocess communication), cơ bản là hai hàm:
- **Send** (message) : gởi một thông điệp
- **Receive** (message) : nhận một thông điệp

Sockets

- Một socket là một thiết bị truyền thông hai chiều tương tự như tập tin, chúng ta có thể đọc hay ghi lên nó, tuy nhiên mỗi socket là một thành phần trong một mối nối nào đó giữa các máy trên mạng máy tính và các thao tác đọc/ghi chính là sự trao đổi dữ liệu giữa các ứng dụng trên nhiều máy khác nhau.
- Sử dụng socket có thể mô phỏng hai phương thức liên lạc trong thực tế : liên lạc thư tín (socket đóng vai trò bưu cục) và liên lạc điện thoại (socket đóng vai trò tổng đài).

Các thao tác liên lạc bằng socket

- Tạo lập hay mở một socket
- Gắn kết một socket với một địa chỉ
- Liên lạc : có hai kiểu liên lạc tùy thuộc vào chế độ nối kết:
  - ***Liên lạc trong chế độ không liên kết*** **:** liên lạc theo hình thức hộp thư
  - ***Liên lạc trong chế độ nối kết*****:**
- Hủy một socket



Cơ chế socket có thể sử dụng để chuẩn hoá mối liên lạc giữa các tiến trình vốn không liên hệ với nhau, và có thể hoạt động trong những hệ thống khác nhau.

BÀI 5 : ĐỒNG BỘ HÓA TIẾN TRÌNH

5.1. Tài nguyên găng và đoạn găng

5.2 Giải pháp Peterson

5.3 Các giải pháp phần cứng

5.4 Semaphore

5.5 Monitors

5.6 Giải pháp trao đổi thông điệp

5.7 Ví dụ kinh điển

5.1 Tài nguyên găng và đoạn găng

Tổng quan về giao tiếp tiến trình

- *Tiến trình độc lập* không ảnh hưởng và không bị ảnh hưởng bởi việc thực thi của các tiến trình khác.
- *Tiến trình hợp tác (không độc lập)* có thể ảnh hưởng và bị ảnh hưởng bởi việc thực thi của các tiến trình khác.
- Ưu điểm của việc hợp tác tiến trình:
  - Chia sẻ thông tin
  - Tăng tốc tính toán (xử lý song song): thời gian I/O và thời gian CPU
  - Tính module hóa
  - Tiện lợi

Hợp tác bằng việc chia sẻ

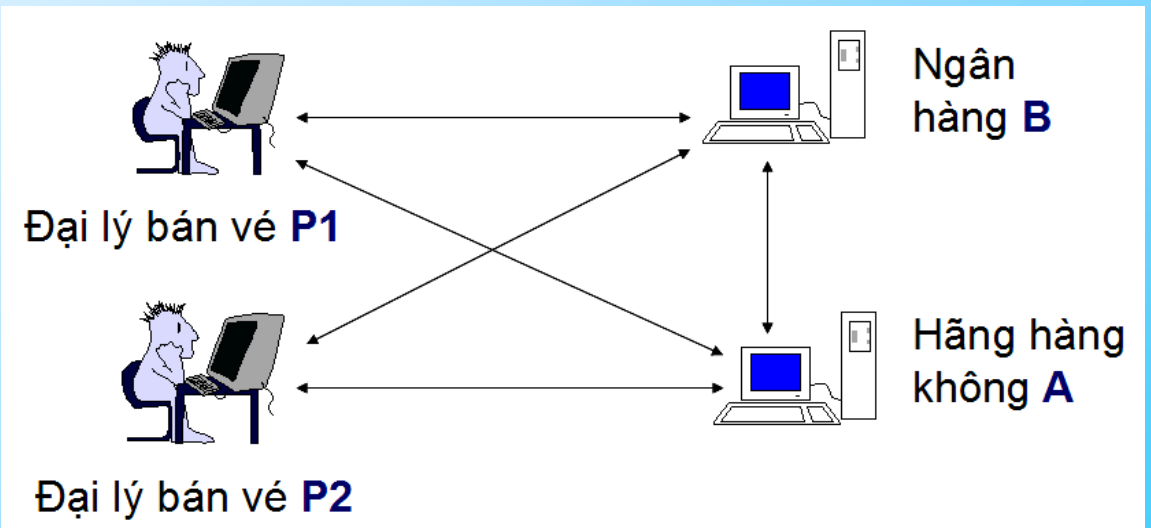
- Các tiến trình sử dụng và cập nhập dữ liệu chia sẻ như các biến, file và cơ sở dữ liệu dùng chung.
- Thao tác ghi phải độc lập từng đôi một để ngăn ngừa tình trạng đụng độ, có thể dẫn đến tính không toàn vẹn dữ liệu.
- Các miền găng dùng để cung cấp sự toàn vẹn dữ liệu.
- Một tiến trình đòi hỏi miền găng phải không bị chờ mãi mãi: deadlock hoặc starvation.

Hợp tác bằng việc giao tiếp

- Giao tiếp cung cấp phương cách để đồng bộ hóa nhiều hoạt động.
- Có khả năng deadlock
  - Mỗi tiến trình đều chờ thông điệp từ một tiến trình khác.
- Có khả năng xảy ra tình trạng đói (starvation)
  - Hai tiến trình gởi thông điệp cho nhau trong khi một tiến trình khác chờ thông điệp.

Tài nguyên găng

- Những tài nguyên có nguy cơ bị hư hỏng, sai lệch khi được hệ điều hành chia sẻ đồng thời cho nhiều tiến trình được gọi là **tài nguyên găng** (critical resource).
- Ví dụ:



Hệ thống bán vé máy bay online

Tài nguyên găng

- Trường hợp chỉ còn 1 vé (SCA=1):

Time	P1	P2	A
0			SCA=1
1	b1: N:=SCA		
2	b2: (N=1)	b1: N:=SCA	
3	b3:	b2: (N=1)	
4	b4:	b3:	
5	b5:	b4:	SCA=SCA-1=0
6		b5:	SCA=SCA-1=-1

Tài nguyên găng là biến SCA đã bị tranh chấp.

Đoạn găng

- Đoạn găng (Critical Section) hay miền găng là đoạn mã có tác động đến các tài nguyên găng, chỉ cho phép một tiểu trình (tiến trình) thi hành tại một thời điểm.
  - Tiểu trình (tiến trình) được gọi là đi vào miền găng.
  - Loại trừ hỗ tương và miền găng là hai khái niệm cùng một mục đích.

Cấu trúc chương trình khi có đoạn găng

<**noncritical section**>

<enter critical section> {kiểm tra và xác lập quyền vào đoạn găng}

<**critical section**>

<exit critical section> {xác nhận khi rời đoạn găng}

<**noncritical section**>

Yêu cầu của đồng bộ hóa tiến trình

Cần thoả mản 4 điều kiện sau:

- **Mutual Exclusion**: Không có hai tiến trình cùng ở trong miền găng cùng lúc.
- **Progess**: Một tiến trình tạm dừng bên ngoài miền găng không được ngăn cản các tiến trình khác vào miền găng
- **Bounded Waiting**: Không có tiến trình nào phải chờ vô hạn để được vào miền găng.
- Không có giả thiết nào đặt ra cho sự liên hệ về tốc độ của các tiến trình, cũng như về số lượng bộ xử lý trong hệ thống.

5.2 Giải pháp Peterson

Giải pháp Peterson

- Giải quyết P0, P1 chia sẽ 2 biến chung
  - int turn; *//đến phiên ai*
  - int interest[2] = FALSE; *//interest[i] = T : Pi muốn vào CS*

Pi NonCS;

```
j = 1 - i;
interest[i] = TRUE;
turn = j;
while (turn==j && interest[j]==TRUE);
```

CS;

```
interest[i] = FALSE;
```

NonCS;

Giải pháp Peterson

P.j

NonCS;

```
i = 1 - j;
interest[j] = TRUE;
turn = i;
while (turn==i &&
interest[i]==TRUE);
```

CS;

```
interest[j] = FALSE;
```

NonCS;

Giải pháp Peterson

- Là giải pháp phần mềm đáp ứng được cả 3 đk:
  - Mutual Exclusion :
    - Pi chỉ có thể vào CS khi: *interest[j] == F* hay *turn == i*
    - Nếu cả 2 muốn về thì do *turn* chỉ có thể nhận giá trị 0 hay 1 nên chỉ có 1 tiến trình vào CS
  - Progress
    - Sử dụng 2 biến *interest[i]* riêng biệt => trạng thái đối phương không khóa được mình.
  - Bounded Wait : *interest[i]* và *turn* đều có thay đổi giá trị.
- Không thể mở rộng cho N tiến trình

5.3 Các giải pháp phần cứng

Giải pháp 1 : Cấm ngắt

NonCS;

```
Disable Interrupt;
```

CS;

```
Enable Interrupt;
```

NonCS;

- ***Disable Interrupt***: *Cấm mọi ngắt, kể cả ngắt đồng hồ*
- ***Enable Interrupt***: *Cho phép ngắt*

Giải pháp 1 : Cấm ngắt

- Thiếu thận trọng
  - Nếu tiến trình bị khóa trong CS ?
    - System Halt
  - Cho phép tiến sử dụng một đặc quyền
    - Quá ...liều !
- Máy có N CPUs ?
  - Không bảo đảm được Mutual Exclusion

Giải pháp 2: Chỉ thị TSL()

- CPU hỗ trợ primitive Test and Set Lock
  - Trả về giá trị hiện hành của 1 biến, và đặt lại giá trị True cho biến
  - Thực hiện một cách không thể phân chia

```
TSL (boolean &target)
{
        TSL = target;
        target = TRUE;
}
```

Áp dụng TSL

int lock = 0

Pi

NonCS;

while (TSL(lock)); // wait

CS;

lock = 0;

NonCS;

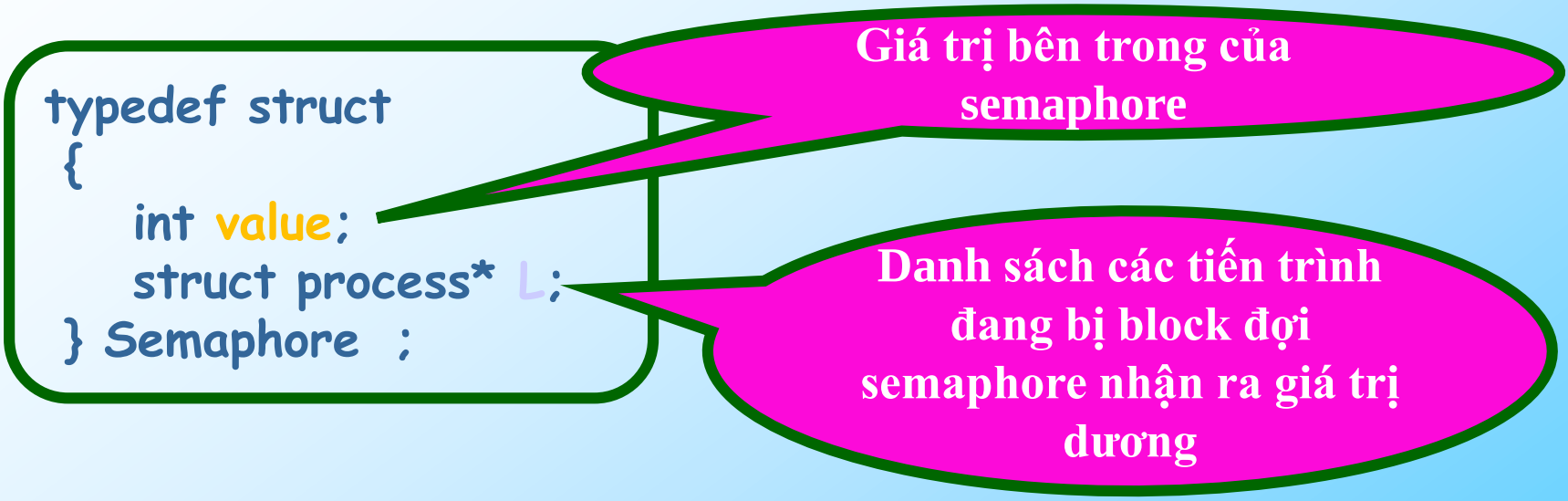
5.4 Semaphore

Semaphore

- Được đề nghị bởi Dijkstra năm 1965
- Các đặc tính: Semaphore s;
  - Có 1 giá trị nguyên dương
  - Chỉ được thao tác bởi 2 primitives :
    - **Down(s)**
    - **Up(s)**
  - Các primitive Down và Up được thực hiện không thể phân chia

```
Semaphore s; // s >=0
```

Semaphore (Sleep & Wakeup)



- Semaphore được xem như là một resource
  - Các tiến trình "yêu cầu" semaphore : gọi Down(s)
    - Nếu không hoàn tất được Down(s) : chưa được cấp resource
      - Blocked, được đưa vào s.L
- Cần có sự hỗ trợ của HĐH
  - Sleep() & Wakeup()

Semaphore (Sleep & Wakeup)

```
Down (S)
{
  S.value --;
    if S.value < 0
    {
       Add(P,S.L);
      Sleep();
    }
}
```

```
Up(S)
{
   S.value ++;
   if S.value ≤ 0
    {
      Remove(P,S.L);
      Wakeup(P);
     }
}
```

Sử dụng Semaphore

- Tổ chức "độc quyền truy xuất"

Semaphore s = 1

P_i

Down (s)
CS;
Up(s)

- Tổ chức "hò hẹn"

Semaphore s = 0

P_1 :
Job1;
Up(s)

P_2:
Down (s);
Job2;

Nhận xét Semaphores

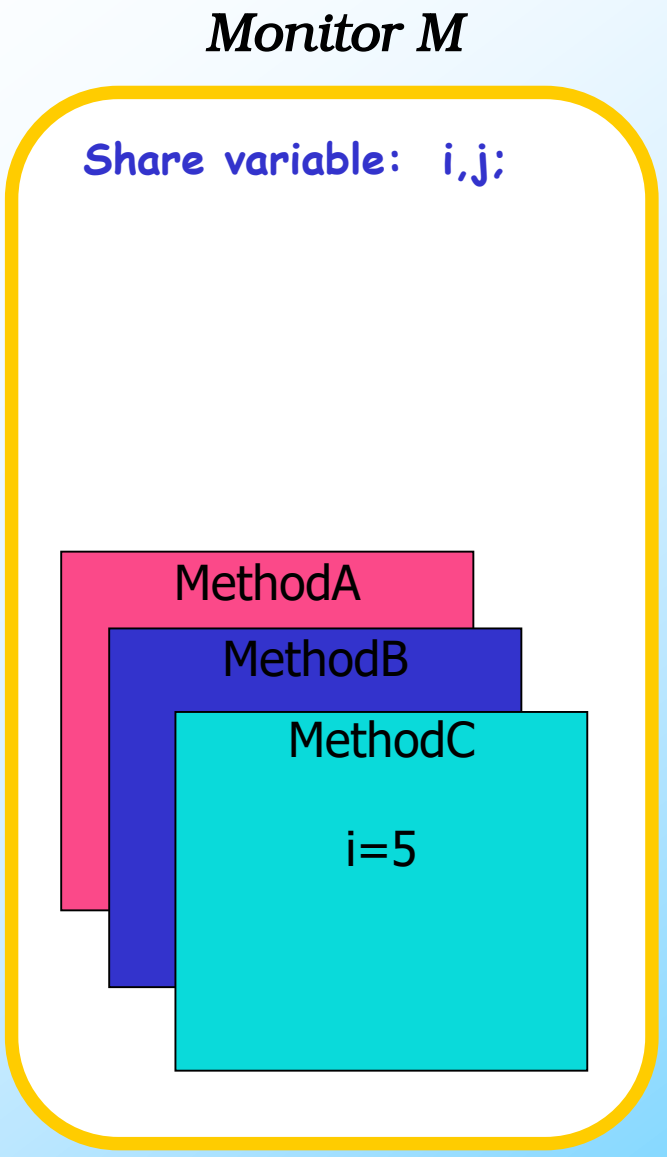
- Là một cơ chế tốt để thực hiện đồng bộ
  - Dễ dùng cho N tiến trình
- Nhưng ý nghĩa sử dụng không rõ ràng
  - MutualExclusion : Down & Up
  - Rendez-vous : Down & Up
  - Chưa phân biệt qua mô hình
- Khó sử dụng đúng
  - Nhầm lẫn

5.5 Monitors

Monitor

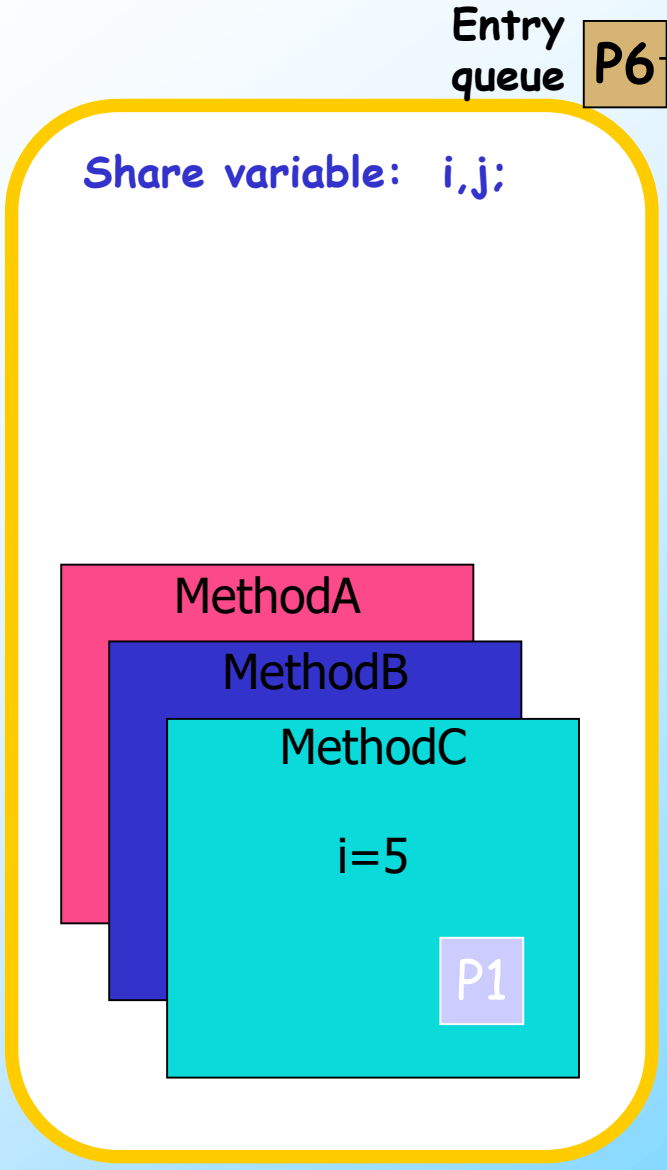
- Đề xuất bởi Hoare(1974) & Brinch (1975)
- Là cơ chế đồng bộ hóa do NNLT cung cấp
  - Hỗ trợ các chức năng như Semaphore
  - Dễ sử dụng và kiểm soát hơn Semaphore
    - Đảm bảo Mutual Exclusion một cách tự động
    - Sử dụng biến điều kiện để thực hiện Synchronization

Monitor : Ngữ nghĩa và tính chất(1)



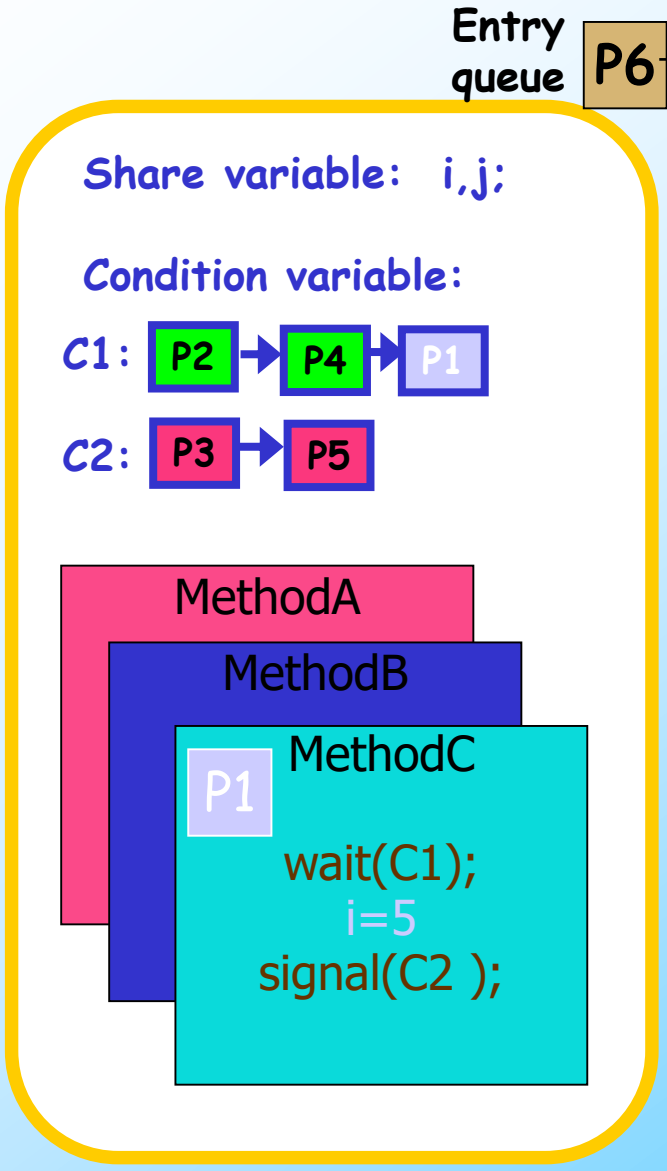
- Là một module chương trình định nghĩa
  - Các CTDL, đối tượng dùng chung
  - Các phương thức xử lý các đối tượng này
  - Đảm bảo tính encapsulation
- Các tiến trình muốn truy xuất dữ liệu bên trong monitor phải dung các phương thức của monitor :
  - P1 : M.C() // i=5
  - P2: M.B() // printf(j)

Monitor : Ngữ nghĩa và tính chất(2)



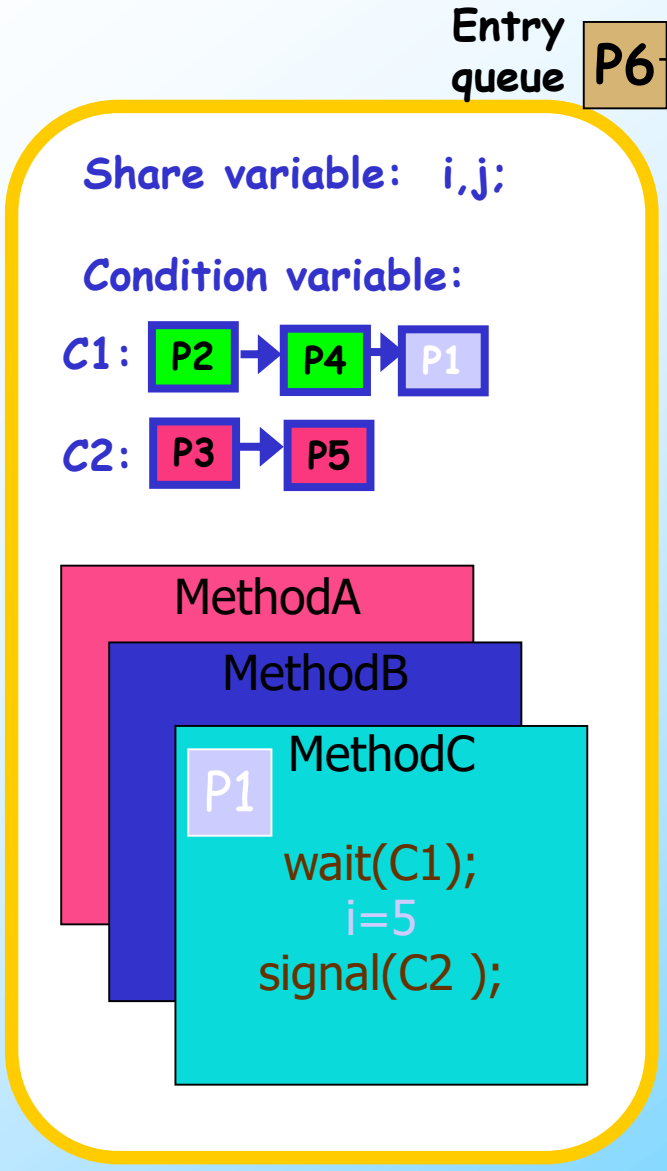
- Tự động bảo đảm Mutual Exclusion
  - Tại 1 thời điểm chỉ có 1 tiến trình thực hiện các phương thức của Monitor
  - Các tiến trình không vào được Monitor sẽ được đưa vào Entry queue của Monitor
- Ví dụ
  - P1 : M.A();
  - P6 : M.B();
  - P7 : M.A();
  - P8 : M.C();

Monitor : Ngữ nghĩa và tính chất(3)



- Hỗ trợ Synchronization với các condition variables
  - Wait(c) : Tiến trình gọi hàm sẽ bị blocked
  - Signal(c): Giải phóng 1 tiến trình đang bị blocked trên biến điều kiện c
  - C.queue : danh sách các tiến trình blocked trên c

Monitor : Ngữ nghĩa và tính chất(3)



- Trạng thái tiến trình sau khi gọi Signal?
  - Blocked. Nhường quyền vào monitor cho tiến trình được đánh thức.
  - Tiếp tục xử lý hết chu kỳ, rồi blocked

Sử dụng Monitor

- *Tổ chức "độc quyền truy xuất"*

```
Monitor       M
<resource type> RC;
Function   AccessMutual
                    CS; // access RC
```

```
Pi
  M.AccessMutual(); //CS
```

- *Tổ chức "hò hẹn"*

```
Monitor    M
Condition  c;
Function   F1
           Job1;
           Signal(c);
Function F2
           Wait(c);
           Job2;
```

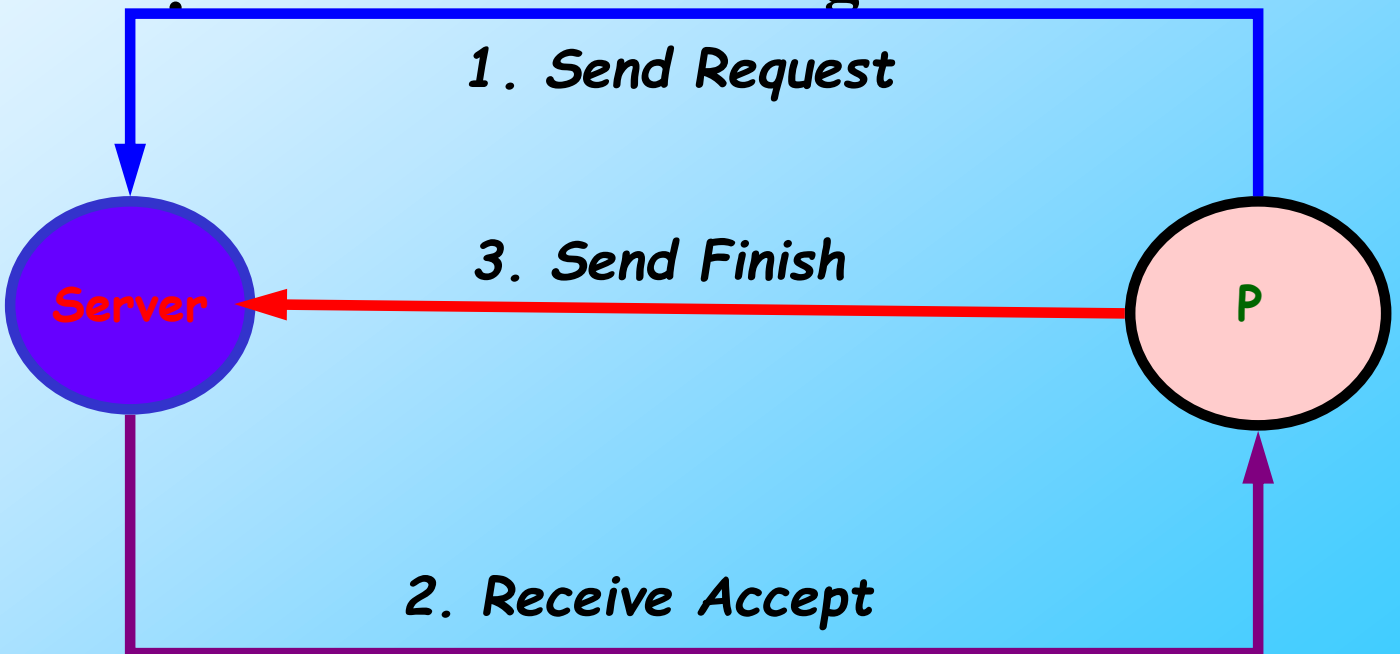
```
P1 :
M.F1();
```

→

```
P2:
M.F2();
```

5.6 Giải pháp trao đổi thông điệp

- **Được hỗ trợ bởi HĐH**
- **Đồng bộ hóa trên môi trường phân tán**
- **2 primitive Send & Receive**
  - **Cài đặt theo mode blocking**



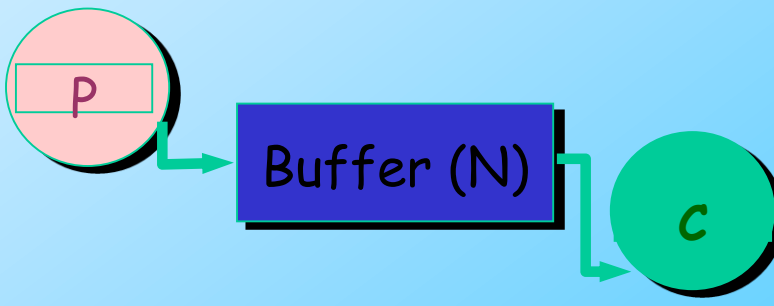
5.7 Ví dụ kinh điển

Các bài toán đồng bộ hóa kinh điển

- Producer – Consumer
- Readers – Writers
- Dinning Philosophers

Producer - Consumer (Bounded-Buffer Problem)

- Mô tả : 2 tiến trình P và C hoạt động đồng hành
  - P sản xuất hàng và đặt vào Buffer
  - C lấy hàng từ Buffer đi tiêu thụ
  - Buffer có kích thước giới hạn
- Tình huống
  - P và C đồng thời truy cập Buffer ?
  - P thêm hàng vào Buffer đầy ?
  - C lấy hàng từ Buffer trống ?



- *P không được ghi dữ liệu vào buffer đã đầy (Rendez-vous)*
- *C không được đọc dữ liệu từ buffer đang trống (Rendez-vous)*
- *P và C không được thao tác trên buffer cùng lúc (Mutual Exclusion)*

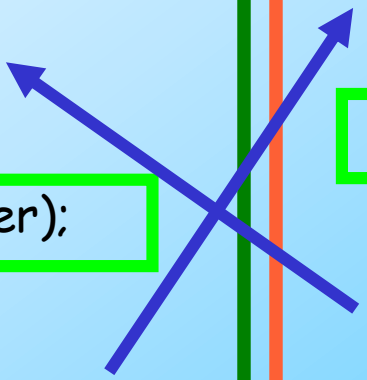
Producer – Consummer: Giải pháp Semaphore

- Các biến chung giữa P và C
  - BufferSize = N; // số chỗ trong bộ đệm
  - semaphore mutex = 1 ; // kiểm soát truy xuất độc quyền
  - semaphore empty = BufferSize; // số chỗ trống
  - semaphore full = 0; // số chỗ đầy
  - int Buffer[BufferSize]; // bộ đệm dùng chung

Producer – Consummer: Giải pháp Semaphore

```
Producer()
{
  int item;
  while (TRUE)
  {
    produce_item(&item);
    down(&empty);
    down(&mutex)
    enter_item(item,Buffer);
    up(&mutex);
    up(&full);
  }
}
```

```
Consumer()
{
  int item;
  while (TRUE)
  {
    down(&full);
    down(&mutex);
    remove_item(&item,Buffer);
    up(&mutex);
    up(&empty);
    consume_item(item);
  }
}
```



Producer – Consummer: Giải pháp Semaphore

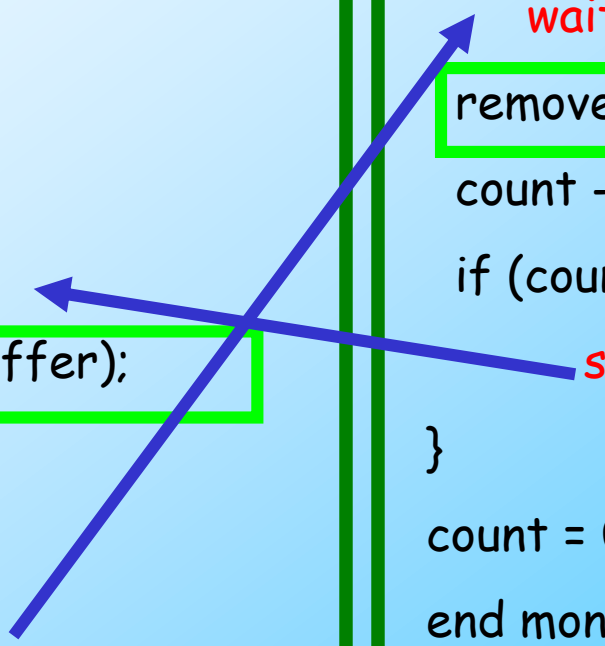
```
Producer()
{
  int item;
  while (TRUE)
  {
    produce_item(&item);
    down(&mutex)
    down(&empty);
    enter_item(item,Buffer);
    up(&mutex);
    up(&full);
  }
}
```

```
Consumer()
{
  int item;
  while (TRUE)
  {
    down(&mutex);
    down(&full);
    remove_item(&item,Buffer);
    up(&mutex);
    up(&empty);
    consume_item(item);
  }
}
```

Producer – Consummer : Giải pháp Monitor

```
monitor ProducerConsumer
 condition full, empty;
 int Buffer[N], count;
 procedure enter();
 {
   if (count == N)
         wait(full);
  enter_item(item,Buffer);
  count ++;
  if (count == 1)
       signal(empty);
}
```

```
procedure remove();
 {
   if (count == 0)
      wait(empty)
  remove_item(&item,Buffer);
  count --;
  if (count == N-1)
         signal(full);
 }
 count = 0;
 end monitor;
```



Producer – Consummer : Giải pháp Monitor

```
Producer()
{
 int item;
while (TRUE)
 {
     produce_item(&item);
     ProducerConsumer.enter;
 }
}
```

```
Consumer();
{
 int item;
  while (TRUE)
  {
   ProducerConsumer.remove;
     consume_item(item);
 }
}
```

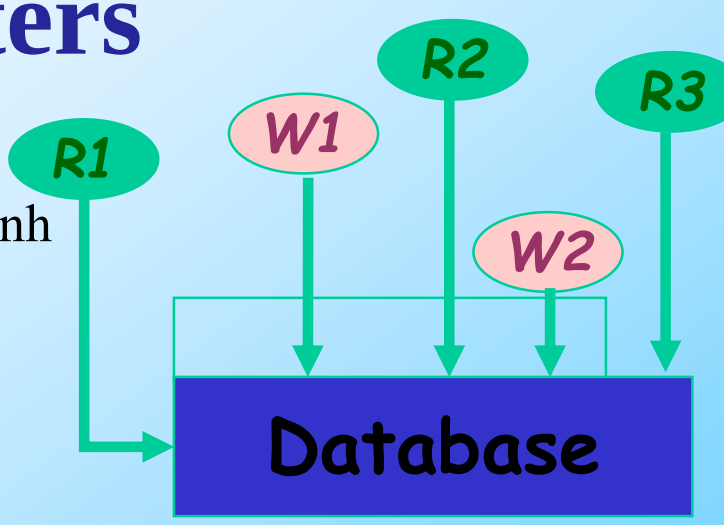
Producer – Consummer: Giải pháp Message

```
Producer()
 {
  int item;
  message m;

  while (TRUE)
  {
    produce_item(&item);
    receive(consumer, Request);
    create_message(&m, item);
    send(consumer,&m);
  }
}
```


```
Consumer();
 {
   int item;
   message m;
   for(0 to N)
        send(producer, Request);
   while (TRUE)
   {
     receive(producer, &m);
     remove_item(&m,&item);
     send(producer, Request);
     consumer_item(item);
    }
 }
```

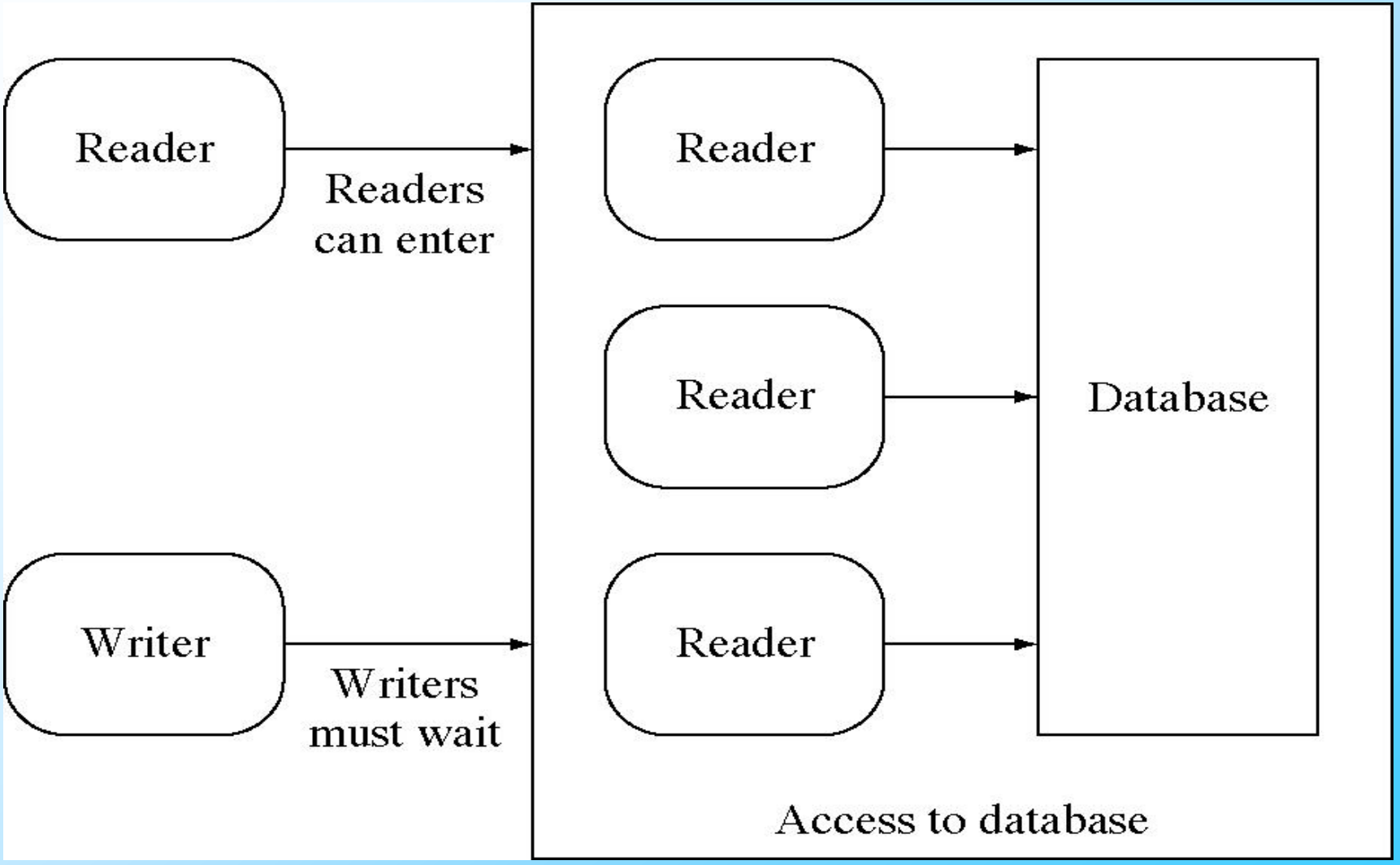
Readers & Writers



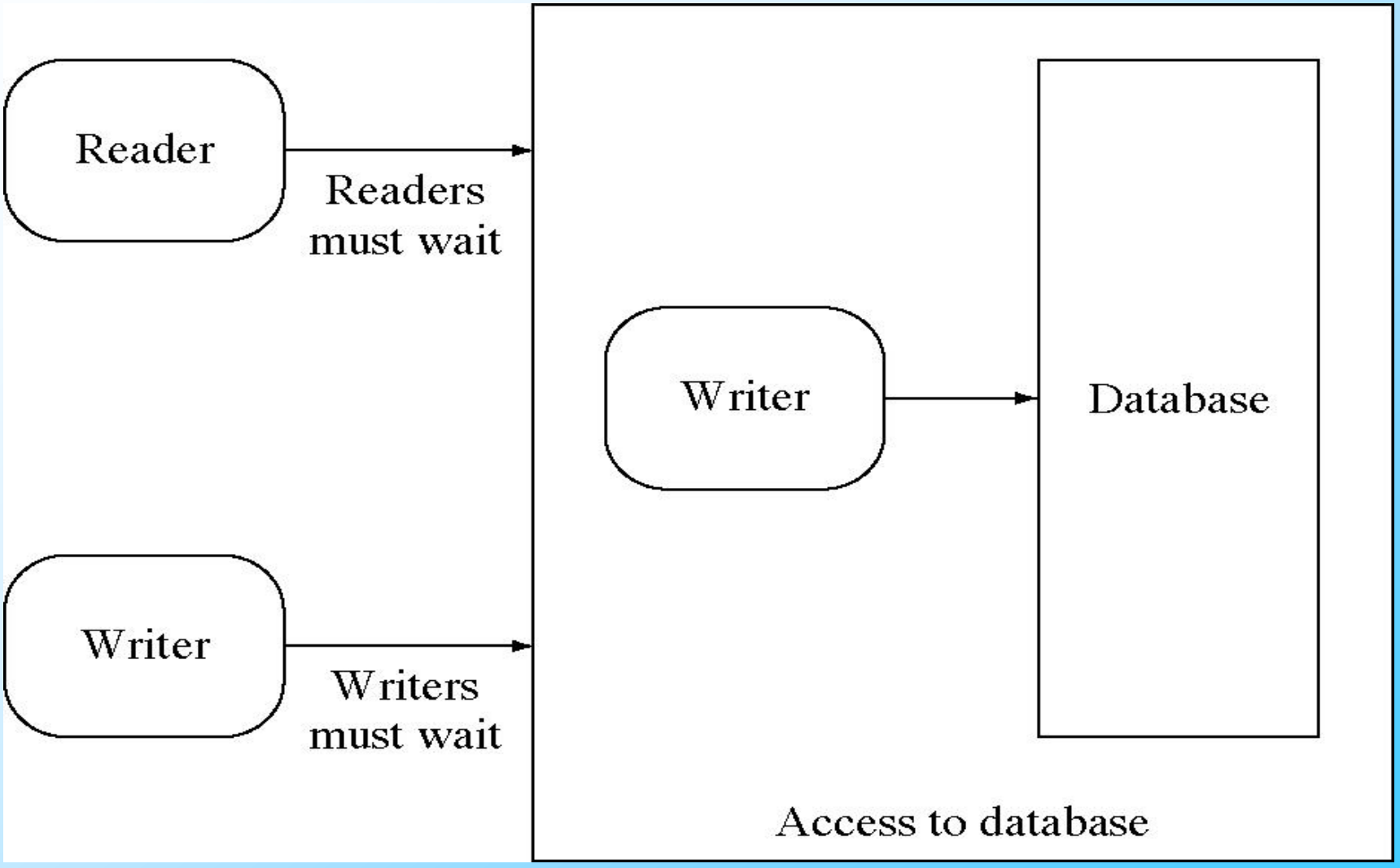
- Mô tả: N tiến trình Ws và Rs hoạt động đồng hành
  - Rs và Ws chia sẽ CSDL
  - W cập nhật nội dung CSDL
  - Rs truy cập nội dung CSDL
- Tình huống
  - Các Rs cùng truy cập CSDL ?
  - W đang cập nhật CSDL thì các Rs truy cập CSDL ?
  - Các Rs đang truy caäp CSDL thì W muốn cập nhật CSDL ?

- *W không được cập nhật dữ liệu khi có ít nhất một R đang truy xuất CSDL (ME)*
- *Rs không được truy cập CSDL khi một W đang cập nhật nội dung CSDL (ME)*
- *Tại một thời điểm , chỉ cho phép một W được sửa đổi nội dung CSDL (ME)*

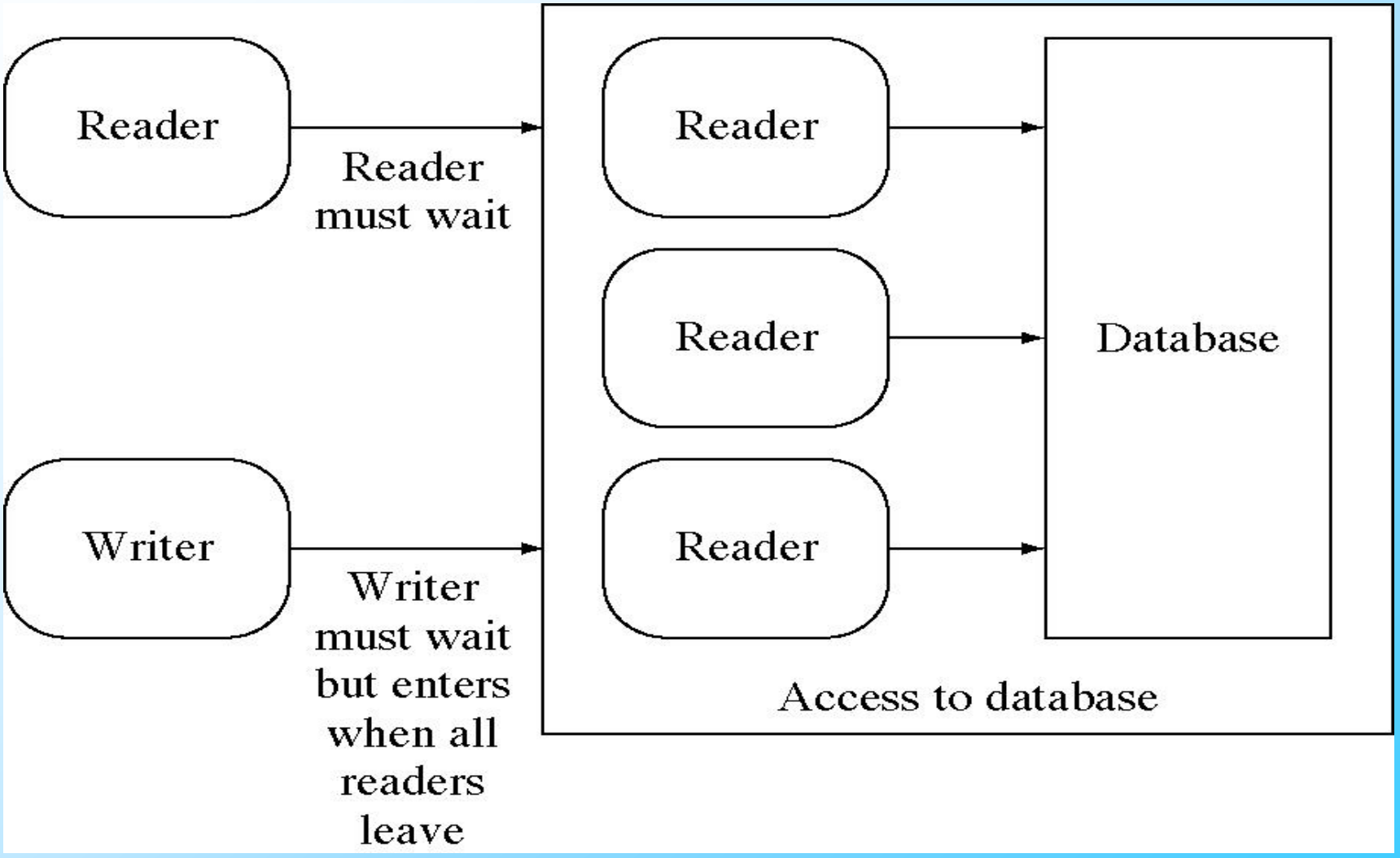
Readers-Writers với “active readers”



Readers-writers với một "active writer"



Ưu tiên ai hơn đây?



Readers & Writers

- W độc quyền truy xuất CSDL
- W hiện tại kết thúc cập nhật CSDL : ai vào?
  - Cho W khác vào, các Rs phải đợi
    - Ưu tiên Writer, Reader có thể starvation
  - Cho các Rs vào, Ws khác phải đợi
    - Ưu tiên Reader, Writer có thể starvation

Readers & Writers : Giải pháp Semaphore

- Các biến dùng chung giữa Rs và Ws
  - semaphore db = 1; // Kiểm tra truy xuất CSDL

R&W : Giải pháp Semaphore (1)

```
Reader()
 {
    down(&db);
    read-db(Database);
    up(&db);
 }
```

```
Writer()
 {
    down(&db);
    write-db(Database);
    up(&db);
}
```

- *Chuyện gì xảy ra ?*
  - *Chỉ có 1 Reader được đọc CSDL tại 1 thời điểm !*

R&W : Giải pháp Semaphore (2)

```
Reader()
 {
    rc = rc +1;
    if (rc ==1)
       down(&db);
    read-db(Database);
    rc = rc – 1;
    if (rc == 0)
       up(&db);
 }
```

```
Writer()
 {
    down(&db);
    write-db(Database);
    up(&db);
}
```

- *Đúng chưa ?*
  - *rc là biến dùng chung giữa các Reader…*
    - *CS đó ☹*

Readers & Writers : Giải pháp Semaphore

- Các biến dùng chung giữa Rs và Ws
  - semaphore db = 1; // Kiểm tra truy xuất CSDL
- Các biến dùng chung giữa Rs
  - int rc; // Số lượng tiến trình Reader
  - semaphore mutex = 1; // Kiểm tra truy xuất rc

R&W : Giải pháp Semaphore (3)

```
Reader()
 {
    down(&mutex);
    rc = rc +1;
    if (rc ==1)
       down(&db);
    up(mutex);
    read-db(Database);
    down(mutex);
    rc = rc – 1;
    if (rc == 0)
       up(&db);
    up(mutex);
 }
```

```
Writer()
 {
    down(&db);
    write-db(Database);
    up(&db);
}
```

Ai được ưu tiên ?

R&W : Giải pháp Semaphore (Thinking...)

```
Reader()
 {
    down(&mutex);
    rc = rc +1;
    up(mutex);
    if (rc ==1)
       down(&db);
    read-db(Database);
    down(mutex);
    rc = rc – 1;
    up(mutex);
    if (rc == 0)
       up(&db);
}
```

```
Writer()
 {
    down(&db);
    write-db(Database);
    up(&db);
}
```

??? hê, hê, hê ☺

R&W: Giải pháp Monitor

```
monitor ReaderWriter
?    Database;
 procedure R1();
 {
 }
procedure R...();
 {
 }
```

```
procedure W1();
 {
 }
procedure W...();
 {
 }
```

```
monitor ReaderWriter

condition  OKWrite, OKRead;
int        rc = 0;
Boolean    busy = false;

procedure BeginRead()
 {
   if (busy)
        wait(OKRead);
   rc++;
   signal(OKRead);
 }
procedure FinishRead()
 {
    rc--;
    if (rc == 0)
      signal(OKWrite);
 }
```

```
procedure BeginWrite()
 {
    if (busy || rc != 0)
      wait(OKWrite);
    busy = true;
 }

procedure FinishWrite()
 {
   busy = false;
   if (OKRead.Queue)
    signal(OKRead);
   else
     signal(OKWrite);
 }
end monitor;
```

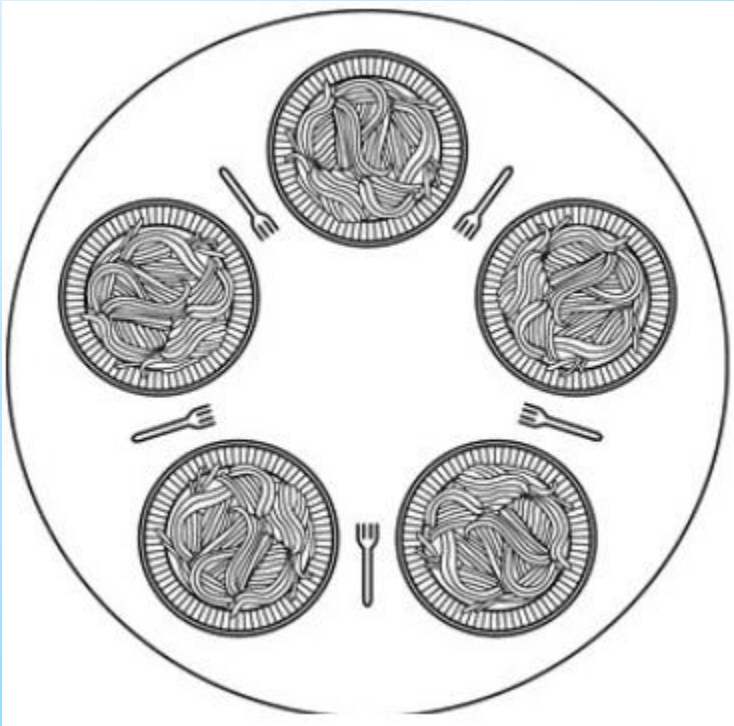
Reader&Writer : Giải pháp Monitor

```
Reader()
{
    RW.BeginRead();
    Read-db(Database);
    RW.FinishRead();
}
```

```
Writer();
{
    RW.BeginWrite();
    Write-db(Database);
    RW.FinishWrite();
}
```

Dining Philosophers

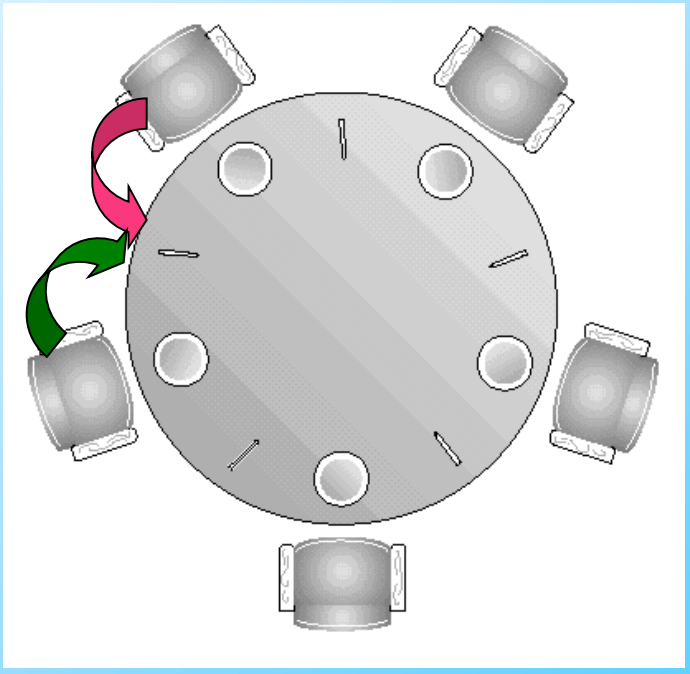
- Năm triết gia ngồi chung quanh bàn ăn món spaghetti
  - Trên bán có 5 cái nĩa được đặt giữa 5 cái đĩa (xem hình)
  - Để ăn món spaghetti mỗi người cần có 2 cái nĩa
- Triết gia thứ i:
  - Thinking...
  - Eating...



Chuyện gì có thể xảy ra ?

Dining Philosophers: Tình huống nguy hiểm

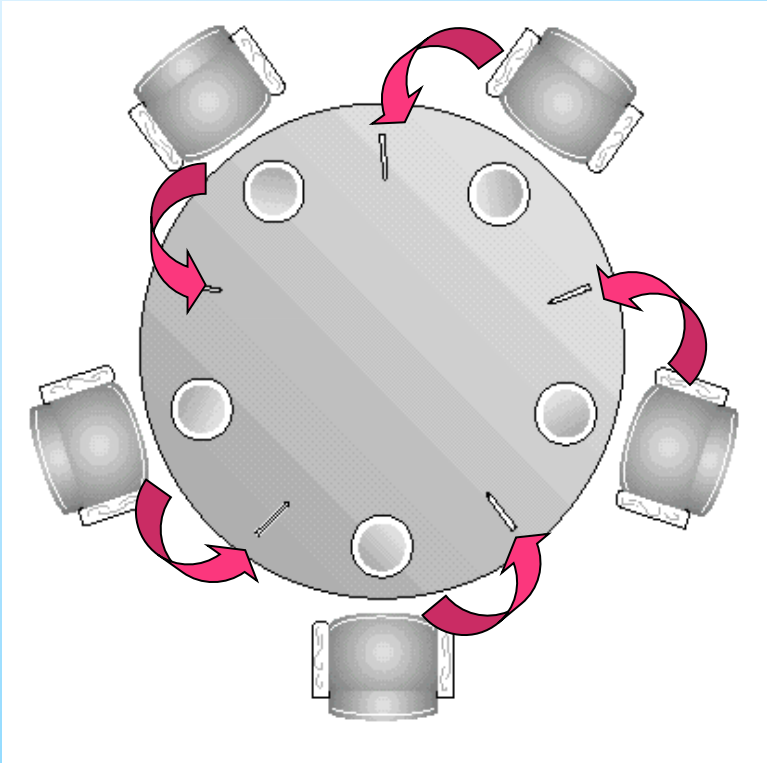
- 2 triết gia "giành giật" cùng 1 cái nĩa
  - Tranh chấp
- Cần đồng bộ hóa hoạt động của các triết gia



Dining Philosophers : Giải pháp đồng bộ

```
semaphore fork[5] = 1;

Philosopher (i)
{
  while(true)
    {
          down(fork[i]);
          down(fork[i+1 mod 5])
          eat;
          up(fork[i]);
          up(fork[i+1 mod 5]);
          think;
    }
```



Deadlock

Dining Philosophers : Thách thức

- Cần đồng bộ sao cho:
  - Không có deadlock
  - Không có starvation

CÂU HỎI ÔN TẬP BÀI 5

1. Cơ chế liên lạc giữa các tiến trình?
2. Nêu ví dụ về một tình huống xung đột.
3. Khái niệm miền găng và các yêu cầu để giải quyết đồng bộ hóa?
4. Trình bày giải pháp Peterson.
5. Trình bày giải pháp Semaphores.

Câu hỏi

Miền găng (critical section) là

A. Bộ nhớ dùng chung cho 2 hoặc nhiều tiến trình
B. CPU
C. Đoạn mã lệnh trong hai tiến trình cùng truy xuất một tài nguyên
D. Không có câu nào đúng

Bài 6: Tắc nghẽn

6.1 Khái niệm

6.2 Điều kiện cần của tắc nghẽn

6.3 Ngăn chặn tắc nghẽn

6.4 Tránh tắc nghẽn

6.5 Phát hiện tắc nghẽn

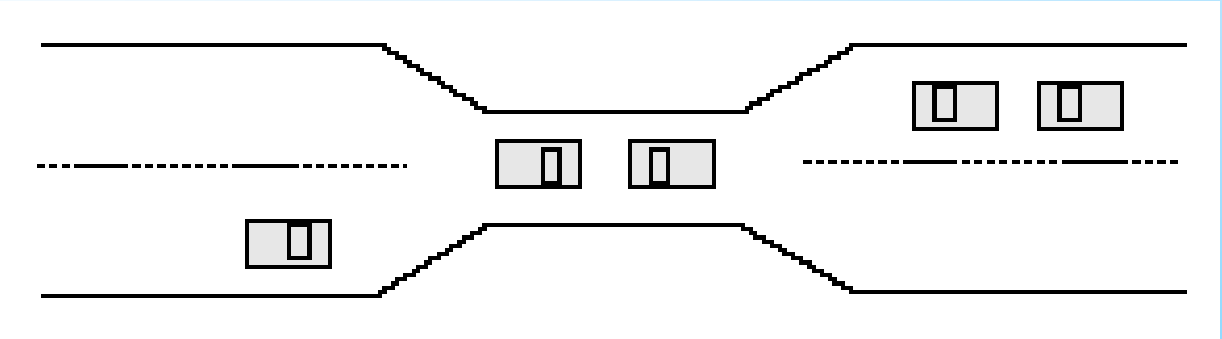
6.6 Phục hồi tắc nghẽn

Vấn đề Deadlock

- Trong môi trường multiprogramming 1 số process có thể tranh nhau 1 số tài nguyên hạn chế.
- 1 process yêu cầu các tài nguyên. Nếu tài nguyên không thể đáp ứng tại thời điểm đó thì process sẽ chuyển sang trạng thái chờ.
- Các process chờ có thể sẽ không bao giờ thay đổi lại trạng thái được vì các tài nguyên mà nó yêu cầu bị giữ bởi các process khác.

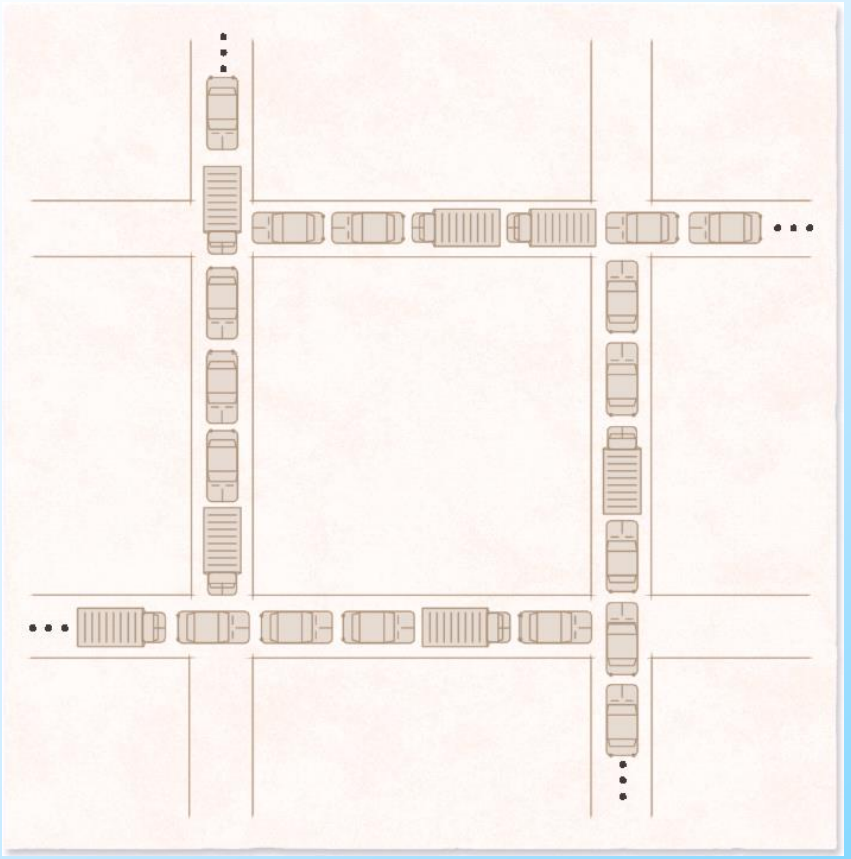
→ Ví dụ: tắc nghẽn trên cầu.

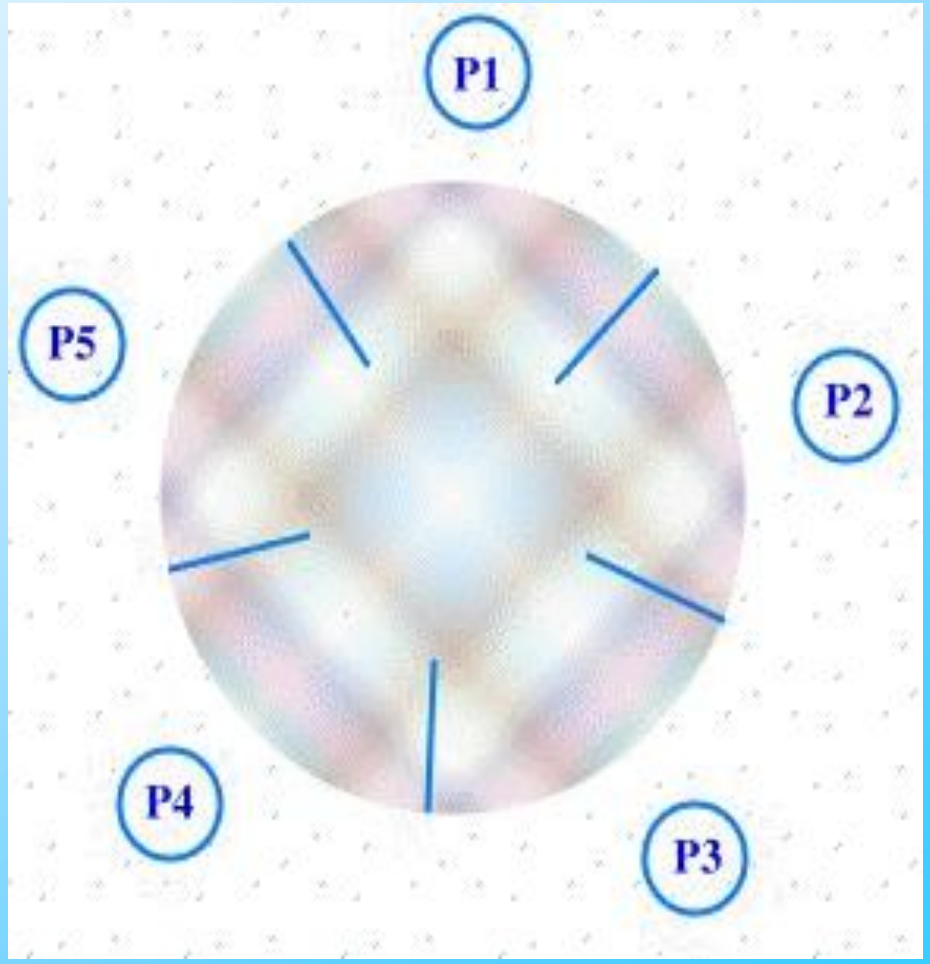
Ví dụ qua cầu



- Hai (hay nhiều) ô tô đối đầu nhau trên 1 cây cầu hẹp chỉ đủ độ rộng cho 1 chiếc.
- Mỗi đoạn của cây cầu có thể xem như 1 tài nguyên
- Nếu deadlock xuất hiện: nó có thể được giải quyết nếu 1 hay 1 số ô tô lùi lại nhường đường rồi lên sau.

Bài toán xảy ra Deadlock

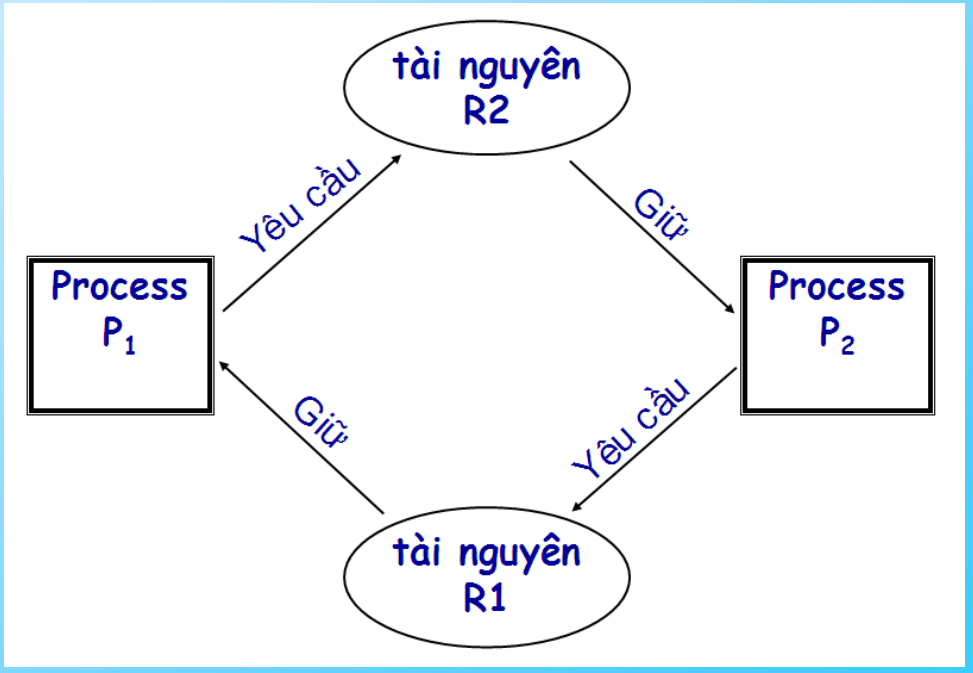




6.1 – Khái niệm Tắc nghẽn

- *Tắc nghẽn* (DeadLock) là trạng thái trong hệ thống có ít nhất hai tiến trình đang dừng chờ lẫn nhau và chúng không thể chạy tiếp được. Sự chờ đợi này có thể kéo dài vô hạn nếu không có sự tác động từ bên ngoài.

- P1, P2 hoạt động đồng thời trong hệ thống. P1 đang giữ tài nguyên găng R1, cần tài nguyên găng R2 để tiếp tục hoạt động; P2 đang giữ tài nguyên R2 và cần R1 để tiếp tục hoạt động →P1 và P2 sẽ không tiếp tục hoạt động được. → trạng thái tắc nghẽn



- Ví dụ 2: Trong các ứng dụng CSDL, một chương trình có thể khóa một vài record mà nó sử dụng để tiến hành cập nhật dữ liệu. Nếu tiến trình P1 khóa record R1, tiến trình P2 khóa record R2, và rồi sau đó mỗi tiến trình lại cố gắng khóa record của một tiến trình kia, tắc nghẽn sẽ xảy ra.

6.2 – Điều kiện cần của tắc nghẽn

Deadlock có thể xảy ra nếu 4 điều kiện sau tồn tại:

- ***Độc quyền sử dụng*** *(mutual exclusion):* ít nhất một tài nguyên được giữ theo nonsharable mode (ví dụ: printer; ví dụ sharable resource: read-only files)
- ***Giữ và đợi*** *(hold and wait):* 1 process đang sở hữu tài nguyên đã được cấp phép trong khi vẫn yêu cầu xin thêm tài nguyên khác.
- ***Không có ưu tiên*** *(no preemption):* 1 tài nguyên chỉ có thể được process (tự nguyện) giải phóng khi nó đã hoàn thành công việc.
- ***Chờ đợi vòng tròn*** *(circular wait):* tồn tại 1 chu kỳ đóng các yêu cầu tài nguyên.

6.3 – Ngăn chặn tắc nghẽn

- *Ngăn chặn tắc nghẽn* (Deadlock Prevention) là cung cấp các phương thức đảm bảo rằng ít nhất 1 trong 4 điều kiện cần của tắc nghẽn không xảy ra.
- Các phương thức ngăn chặn tắc nghẽn sau đây đều khó thực hiện và có thể làm cho việc sử dụng tài nguyên bị chậm và thông lượng hệ thống bị giảm.

Ngăn chặn điều kiện loại trừ lẫn nhau

- Đảm bảo hệ thống không có các file không thể chia sẻ.
- Một process không bao giờ chờ tài nguyên chia sẻ (shareble resource).
  Ví dụ: read-only file
- Nhưng có 1 số tài nguyên không chia sẻ được.
  Ví dụ : chế độ toàn màn hình

Ngăn chặn điều kiện giữ và đợi

- Cách 1: Phải đảm bảo rằng bất cứ khi nào một tiến trình yêu cầu tài nguyên, nó phải không giữ bất cứ tài nguyên nào khác.
- Cách 2: Mỗi process yêu cầu toàn bộ tài nguyên cần thiết một lần. Nếu có đủ tài nguyên thì hệ thống sẽ cấp phát, nếu không đủ tài nguyên thì process phải bị blocked.
- Khuyết điểm của các cách trên:
    - Hiệu suất sử dụng tài nguyên
    - Quá trình có thể bị starvation

Ngăn chặn điều kiện Không thể thu hồi

- Nếu process A có giữ tài nguyên và đang yêu cầu tài nguyên khác nhưng tài nguyên này chưa cấp phát ngay được thì:
  - Cách 1: Hệ thống lấy lại mọi tài nguyên mà A đang giữ
    - A chỉ bắt đầu lại được khi có được các tài nguyên đã bị lấy lại cùng với tài nguyên đang yêu cầu.
  - Cách 2: Hệ thống sẽ xem tài nguyên mà A yêu cầu
    - Nếu tài nguyên được giữ bởi một process khác đang đợi thêm tài nguyên, tài nguyên này được hệ thống lấy lại và cấp phát cho A.
    - Nếu tài nguyên được giữ bởi process không đợi tài nguyên, A phải đợi và tài nguyên của A bị lấy lại. Tuy nhiên hệ thống chỉ lấy lại các tài nguyên mà process khác yêu cầu.

Ngăn chặn điều kiện Chờ đợi luẩn quẩn

- Sắp xếp thứ tự cho tất cả loại tài nguyên và đòi hỏi mỗi tiến trình phải yêu cầu tài nguyên theo thứ tự đó.
- Ví dụ, một tiến trình muốn dùng ổ đĩa và máy in tại cùng một lúc thì trước tiên phải yêu cầu ổ đĩa, nếu được cấp ổ đĩa thì mới yêu cầu máy in.

6.4 – Trách tắc ngẽn

- *Tránh tắc nghẽn* (Deadlock Avoidance) đòi hỏi hệ điều hành có trước thông tin bổ sung liên quan đến tài nguyên mà một tiến trình sẽ yêu cầu và sử dụng trong suốt cuộc đời của nó.
- Yêu cầu mỗi tiến trình khai báo số lượng tối đa mỗi loại tài nguyên mà nó cần.
- Giải thuật tránh tắc nghẽn sẽ kiểm tra *trạng thái cấp phát tài nguyên* (resource-allocation state) để bảo đảm hệ thống không rơi vào deadlock.
- Trạng thái cấp phát tài nguyên được định nghĩa dựa trên số tài nguyên còn lại, số tài nguyên đã được cấp phát và yêu cầu tối đa của các process.

Trạng thái an toàn

- *Trạng thái an toàn* (safe) là trạng thái trong đó hệ thống có thể thỏa mãn tất cả các nhu cầu tài nguyên của mỗi tiến trình theo một thứ tự nào đó mà vẫn không xảy ra tắc nghẽn. Thứ tự này còn gọi là *thứ tự an toàn*.
- Một trạng thái của hệ thống được gọi là *không an toàn* (unsafe) nếu không tồn tại một chuỗi an toàn.

Trạng thái an toàn

- Nếu hệ thống ở trạng thái an toàn => không có deadlock.
- Nếu hệ thống ở trạng thái không an toàn => có thể có deadlock.
- Sự tránh khỏi deadlock => đảm bảo rằng hệ thống sẽ không bao giờ bước vào trạng thái không an toàn:
  - Mỗi loại tài nguyên có 1 instance giải thuật đồ thị phân phối tài nguyên.
  - Mỗi loại tài nguyên có nhiều instance: giải thuật chủ nhà băng (banker).

Ví dụ1:
Trạng thái an toàn

	Max		Chiếm		Còn	
	R1	R2	R1	R2	R1	R2
P1	3	2	1	0	4	1
P2	6	1	2	1		
P3	3	1	2	1		

- Cột *Max* chỉ số lượng tối đa của mỗi loại tài nguyên mà mỗi tiến trình yêu cầu.
- Cột *Chiếm* chỉ số lượng mỗi loại tài nguyên mà mỗi tiến trình đang chiếm giữ (tức là đã được cấp).
- Cột *Còn* chỉ số lượng mỗi loại tài nguyên hiện đang còn rảnh rỗi trong hệ thống, có thể cấp ngay cho các tiến trình có yêu cầu.

Ví dụ 2:
Trạng thái an toàn

- Hệ thống với 12 ổ băng từ và 3 tiến trình: P1, P2, P3. P1 yêu cầu 10 ổ băng từ, P2 cần 4 và P3 cần tới 9 ổ băng từ. Giả sử rằng tại thời điểm hiện tại, P1 đang giữ 5 ổ băng từ, P2 giữ 2 và P3 giữ 2 ổ băng từ. Do đó, có 3 ổ băng từ còn rảnh.

	Max	Chiếm	Còn
	R	R	R
P1	10	5	3
P2	4	2	
P3	9	2	

Hiện tại, **<P2,P1,P3>** là một trạng thái an toàn

Ví dụ 2:
Trạng thái an toàn

- Giả sử P3 được cấp thêm 1 ổ băng từ nữa.

	Max	Chiếm	Còn
	R	R	R
P1	10	5	2
P2	4	2	
P3	9	3	

Hệ thống không còn trong trạng thái an toàn

Giải thuật nhà băng

- Khi tiến trình mới đưa vào hệ thống, nó phải khai báo số tối đa các thể hiện của mỗi loại tài nguyên mà nó cần. Số này có thể không vượt quá tổng số tài nguyên trong hệ thống.
- Khi một tiến trình yêu cầu cấp phát tài nguyên, hệ thống phải xác định sau khi cấp phát các tài nguyên này hệ thống có vẫn ở trong trạng thái an toàn hay không. Nếu trạng thái hệ thống sẽ vẫn là an toàn, tài nguyên sẽ được cấp, ngược lại, tiến trình phải chờ cho tới khi một vài tiến trình giải phóng đủ tài nguyên.
- Giải thuật nhà băng dùng để **xác định trạng thái hiện tại có an toàn hay không**?

Giải thuật nhà băng

Bước 1 : Từ bảng trạng thái lập *bảng nhu cầu* trong đó thay cột *Max* bằng cột *Cần* với công thức tính toán Cần = Max – Chiếm. Cột *Cần* thể hiện số lượng mỗi loại tài nguyên cần cung cấp thêm cho mỗi tiến trình.

Bước 2 :

While ∃i : (Cần(Pi)<>0) and (Cần(Pi)<=Còn)

Begin

Còn = Còn+Chiếm(Pi);

Cần(Pi)=0; Chiếm(Pi)=0;

End

If ∀i : Cần(Pi)=0

Then “Trạng thái an toàn”

Else “Trạng thái không an toàn”

Giải thuật tránh tắc nghẽn

if Not(Request(P)<= Còn) Then

"Không cấp được"

Else

Begin

1. Lập bảng trạng thái sau khi cấp tài nguyên cho P:

Còn = Còn – Request(P);

Chiếm(P)= Chiếm(P)+ Request(P);

Max(P)=Max(P);

Các số liệu ứng với các tiến trình khác giữ nguyên;

2. Kiểm tra trạng thái trên có an toàn không

3. If (An toàn) then "Cấp ngay" else "Không cấp ngay"

end

6.5 – Phát hiện Deadlock

Mỗi khi tắc nghẽn được phát hiện, hệ điều hành thực hiện một vài giải pháp để thoát khỏi tắc nghẽn. Một vài giải pháp có thể:

- Thoát tất cả các tiến trình bị tắc nghẽn. Đây là một giải pháp đơn giản nhất, thường được các hệ điều hành sử dụng nhất.
- Sao lưu lại mỗi tiến trình bị tắc nghẽn tại một vài điểm kiểm tra được định nghĩa trước, sau đó khởi động lại tất cả các tiến trình. Giải pháp này yêu cầu hệ điều hành phải lưu lại các thông tin cần thiết tại điểm dừng của tiến trình, đặc biệt là con trỏ lệnh và các tài nguyên tiến trình đang sử dụng, để có thể khởi động lại tiến trình được → nguy cơ xuất hiện tắc nghẽn trở lại là rất cao, vì khi tất cả các tiến trình đều được reset trở lại thì việc tranh chấp tài nguyên là khó tránh khỏi. Ngoài ra hệ điều hành thường phải chi phí rất cao cho việc tạm dừng và tái kích hoạt tiến trình.

Phát hiện Deadlock(2/3)

- Kết thúc một tiến trình trong tập tiến trình bị tắc nghẽn, thu hồi tài nguyên của tiến trình này, để cấp phát cho một tiến trình nào đó trong tập tiến trình tắc nghẽn. Sau đó, gọi lại thuật toán kiểm tra tắc nghẽn để xem hệ thống đã ra khỏi tắc nghẽn hay chưa, nếu rồi thì dừng, nếu chưa thì tiếp tục giải phóng thêm tiến trình khác
  - chọn tiến trình nào để giải phóng đầu tiên?
  - dựa vào tiêu chuẩn nào để chọn lựa sao cho chi phí để giải phóng tắc nghẽn là thấp nhất?

Phát hiện Deadlock(3/3)

- Tập trung toàn bộ quyền ưu tiên sử dụng tài nguyên cho một tiến trình, để tiến trình này ra khỏi tắc nghẽn, và rồi kiểm tra xem hệ thống đã ra khỏi tắc nghẽn hay chưa?
  - nếu rồi thì dừng lại,
  - nếu chưa thì tiếp tục.

Lần lượt như thế cho đến khi hệ thống ra khỏi tắc nghẽn.

6.6 – Phục hồi tắc ngẽn

Khi deadlock xảy ra, để phục hồi:

- báo người vận hành (operator)

hoặc

- hệ thống tự động phục hồi bằng cách bẻ gãy chu trình deadlock:
  - chấm dứt một hay nhiều tiến trình
  - lấy lại tài nguyên từ một hay nhiều tiến trình

Chấm dứt tiến trình

- Chấm dứt tất cả process bị deadlocked, hoặc
- Chấm dứt lần lượt từng process cho đến khi không còn deadlock
  - Sử dụng giải thuật phát hiện deadlock để xác định còn deadlock hay không

Dựa trên yếu tố nào để chọn process cần được chấm dứt?

- Độ ưu tiên của process
- Thời gian đã thực thi của process và thời gian còn lại
- Loại tài nguyên mà process đã sử dụng
- Tài nguyên mà process cần thêm để hoàn tất công việc
- Số lượng process cần được chấm dứt
- Process là interactive process hay batch process

Lấy lại tài nguyên

- Lấy lại tài nguyên từ một process, cấp phát cho process khác cho đến khi không còn deadlock nữa.
- Các vấn đề trong chiến lược thu hồi tài nguyên:
  - Chọn "nạn nhân" để tối thiểu chi phí (có thể dựa trên số tài nguyên sở hữu, thời gian CPU đã tiêu tốn,...)
  - Trở lại trạng thái trước deadlock (Rollback): rollback process bị lấy lại tài nguyên trở về trạng thái safe, tiếp tục process từ trạng thái đó. Hệ thống cần lưu giữ một số thông tin về trạng thái các process đang thực thi.
  - Đói tài nguyên (Starvation): để tránh starvation, phải bảo đảm không có process sẽ luôn luôn bị lấy lại tài nguyên mỗi khi deadlock xảy ra.

CÂU HỎI ÔN TẬP BÀI 6

1. Hãy định nghĩa trạng thái tắc nghẽn của hệ thống
2. Hãy nêu 4 điều kiện cần để xuất hiện trạng thái tắc nghẽn.
3. Hãy cho một ví dụ về trạng thái tắc nghẽn.
4. Hãy định nghĩa trạng thái an toàn (hiểu theo nghĩa không gây ra tình trạng tắc nghẽn) của hệ thống.
5. Hãy dùng giải thuật nhà băng để xác định xem trạng thái sau có an toàn hay không?

	Max		Chiếm		Còn	
	R1	R2	R1	R2	R1	R2
P1	4	10	1	6	2	1
P2	6	3	4	2		
P3	8	5	6	1		

Bài 7: Quản lý bộ nhớ

7.1 Mở đầu

7.2 Cấp phát bộ nhớ liên tục

7.3 Cấp phát bộ nhớ không liên tục

7.1 – Mở đầu

- Nhiệm vụ của quản lý bộ nhớ:
  - Tổ chức và quản lý bộ nhớ vật lý
  - Tổ chức và quản lý bộ nhớ logic
  - Định vị và tái định vị các tiến trình
  - Chia sẻ bộ nhớ cho các tiến trình
  - Bảo vệ vùng nhớ của các tiến trình

Vì sao phải quản lý bộ nhớ

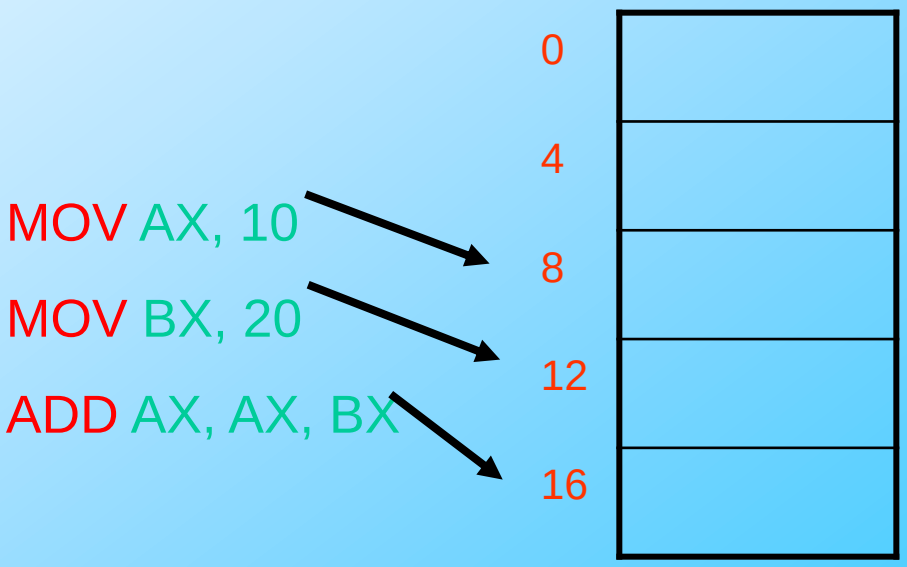
- Một chương trình muốn chạy thì phải được nạp vào trong bộ nhớ chính.
  - Vấn đề:
    - Khi nào nạp?
    - Nạp vào đâu?
    - Nạp những phần nào?
- Quản lý bộ nhớ giúp tối ưu hóa hoạt động của bộ nhớ

  →Tối ưu hóa số tiến trình cùng lúc ở trong bộ nhớ chính nhằm nâng cao tính đa chương

  →Tận dụng tối đa bộ nhớ của máy tính

Bộ nhớ

- Là một dãy các ô nhớ liên tục nhau
- Mỗi ô nhớ (một word) có một địa chỉ
- Chương trình = tập các câu lệnh (chỉ thị máy) + dữ liệu
- Nạp chương trình vào bộ nhớ ⇔ đặt các chỉ thị và dữ liệu vào các ô nhớ ⇔ xác định ánh xạ giữa các chỉ thị, dữ liệu vào địa chỉ trong bộ nhớ



- Sự tương ứng giữa địa chỉ *logic* và địa chỉ vật lý (*physic*) : làm cách nào để chuyển đổi một địa chỉ tượng trưng (symbolic) trong chương trình thành một địa chỉ thực trong bộ nhớ chính?
- Quản lý bộ nhớ vật lý: làm cách nào để mở rộng bộ nhớ có sẵn nhằm lưu trữ được nhiều tiến trình đồng thời?
- Chia sẻ thông tin: làm thế nào để cho phép hai tiến trình có thể chia sẻ thông tin trong bộ nhớ?
- Bảo vệ: làm thế nào để ngăn chặn các tiến trình xâm phạm đến vùng nhớ được cấp phát cho tiến trình khác?

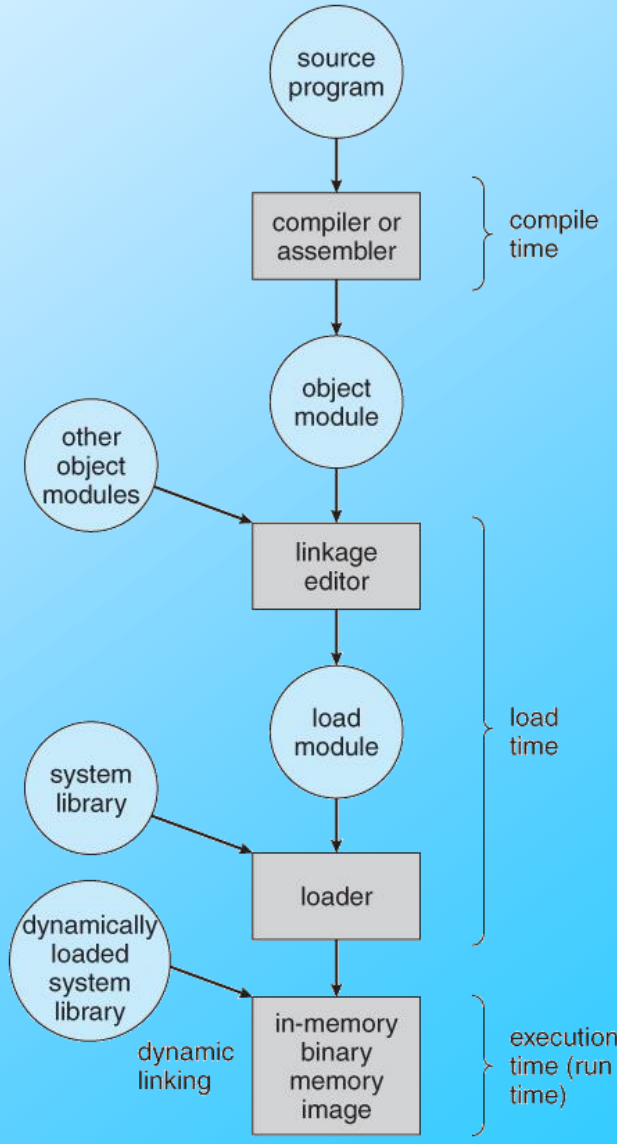
Không gian địa chỉ lôgic và vật lý

- Khái niệm một *không gian địa chỉ lôgic* được kết buộc với một *không gian địa chỉ vật lý* đóng vai trò trung tâm trong một việc quản lý bộ nhớ tốt.
  - *Địa chỉ lôgic* – được phát sinh bởi bộ xử lý; còn được gọi là *địa chỉ ảo*.
  - *Địa chỉ vật lý* – địa chỉ được nhìn thấy bởi đơn vị quản lý bộ nhớ.
  - *Không gian địa chỉ* – là tập hợp tất cả các địa chỉ ảo phát sinh bởi một chương trình.
  - *Không gian vật lý* – là tập hợp tất cả các địa chỉ vật lý tương ứng với các địa chỉ ảo.
- Địa chỉ lôgic và địa chỉ vật lý như nhau trong mô hình kết buộc địa chỉ tại thời điểm biên dịch và nạp;
- Địa chỉ lôgic và địa chỉ vật lý khác nhau trong mô hình kết buộc địa chỉ tại thời điểm thi hành.

Ánh xạ chỉ thị, dữ liệu vào địa chỉ bộ nhớ

Việc ánh xạ chỉ thị, dữ liệu vào địa chỉ bộ nhớ có thể xảy ra tại 3 thời điểm:

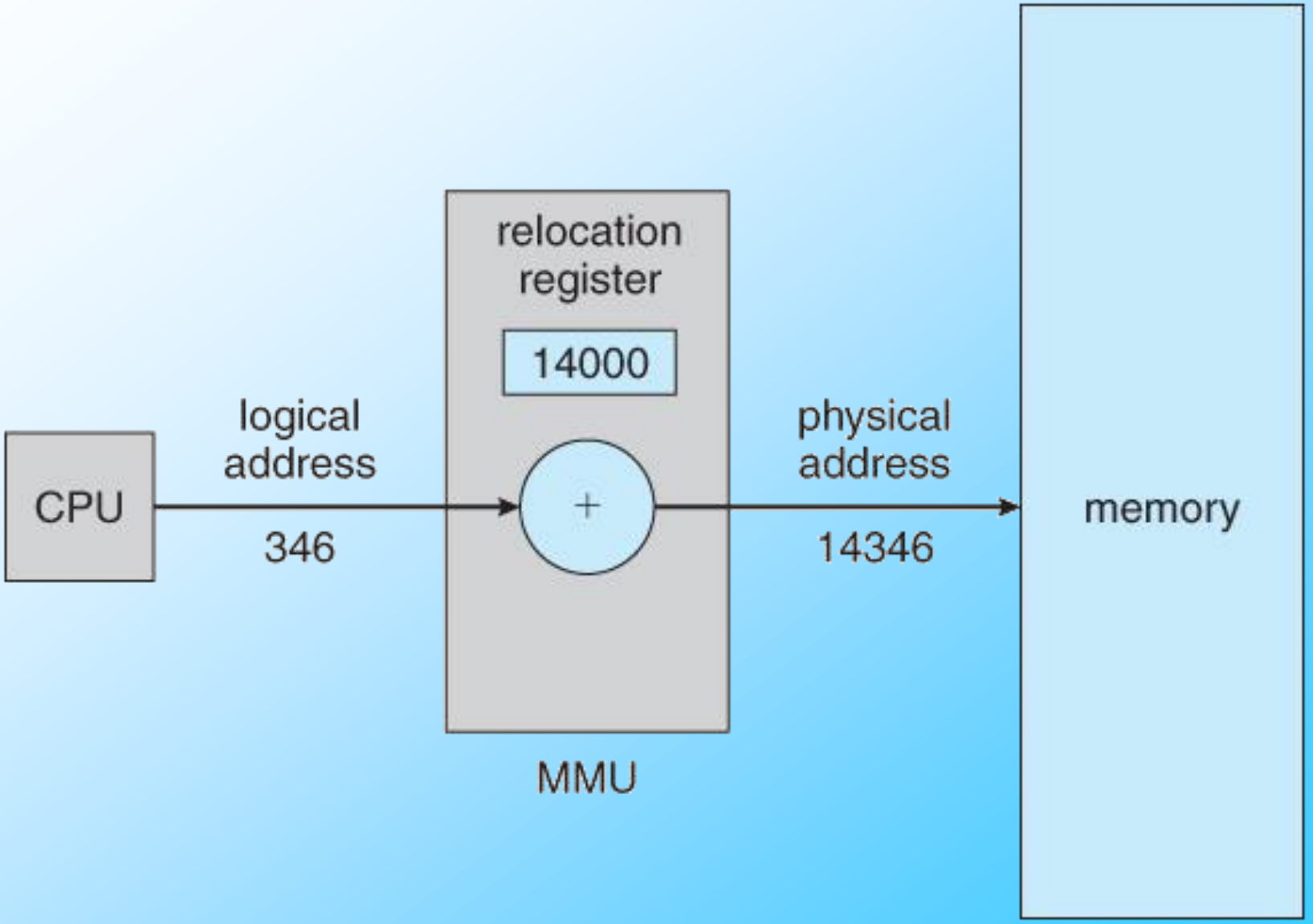
- **Thời điểm biên dịch**: nếu địa chỉ vùng nhớ được biết trước thì *mã lệnh tuyệt đối (có địa chỉ tuyệt đối)* có thể được tạo ra ngay tại thời điểm biên dịch. Nếu địa chỉ bắt đầu của vùng nhớ bị thay đổi thì sẽ phải biên dịch lại.
- **Thời điểm nạp**: Tạo ra các *mã lệnh có thể tái định vị* (*relocatable* code) nếu địa chỉ vùng nhớ không thể biết tại thời điểm biên dịch.
- **Thời điểm thi hành**: Việc kết hợp mã lệnh và địa chỉ sẽ được trì hoãn cho đến lúc chạy chương trình nếu tiến trình đó có thể bị di chuyển từ phân đoạn nhớ này đến phân đoạn nhớ khác. Cần phải có hỗ trợ từ phần cứng để ánh xạ địa chỉ (ví dụ: thanh ghi cơ sở và thanh ghi giới hạn (*base registers*, *limit registers*)).



Đơn vị quản lý Bộ nhớ
Memory-Management Unit (MMU)

- Là một cơ chế phần cứng để ánh xạ địa chỉ ảo thành địa chỉ vật lý vào thời điểm xử lý.
- Trong mô hình MMU, mỗi địa chỉ phát sinh bởi một tiến trình được cộng thêm giá trị của thanh ghi tái định vị (relocation register) tại thời điểm nó truy xuất đến bộ nhớ.
- Chương trình người dùng chỉ quan tâm đến *địa chỉ lôgic*; nó không thấy *địa chỉ vật lý thật sự*.

MMU


relocation
register
14000
CPU
logical
address
346
+
physical
address
14346
memory
MMU

7.2 – Cấp phát liên tục

Cấp phát liên tục

- Mỗi tiến trình được nạp vào một vùng nhớ liên tục đủ lớn để chứa toàn bộ tiến trình.
- Ưu điểm : việc chuyển đổi địa chỉ logic thành địa chỉ vật lý và ngược lại chỉ cần dựa vào một công thức đơn giản ***<địa chỉ vật lý> = <địa chỉ bắt đầu> + <địa chỉ logic>***.

Hiện tượng phân mảnh bộ nhớ

- Cấp phát liên tục có nhược điểm lớn nhất là không sử dụng hiệu quả bộ nhớ do *hiện tượng phân mảnh bộ nhớ*.
- Không thể nạp được một tiến trình nào đó do không có một vùng nhớ trống liên tục đủ lớn trong khi tổng kích thước các vùng nhớ trống đủ để thỏa mãn yêu cầu.
- Ví dụ, nếu bộ nhớ có ba vùng nhớ trống liên tục với kích thước 1MB, 3MB, 5MB thì không thể nào nạp một chương trình có kích thước **6**MB mặc dù tổng kích thước bộ nhớ trống là **9**MB.

Giải quyết phân mảnh bộ nhớ

- Đề ra chiến lược cấp phát hợp lý
- Tái định vị các tiến trình
- Sử dụng kỹ thuật hoán vị (swapping)
- Sử dụng kỹ thuật phủ lấp (overlay)

Đề ra chiến lược cấp phát hợp lý

Chọn vùng nhớ trống nào để cấp phát cho một tiến trình khi có nhu cầu.

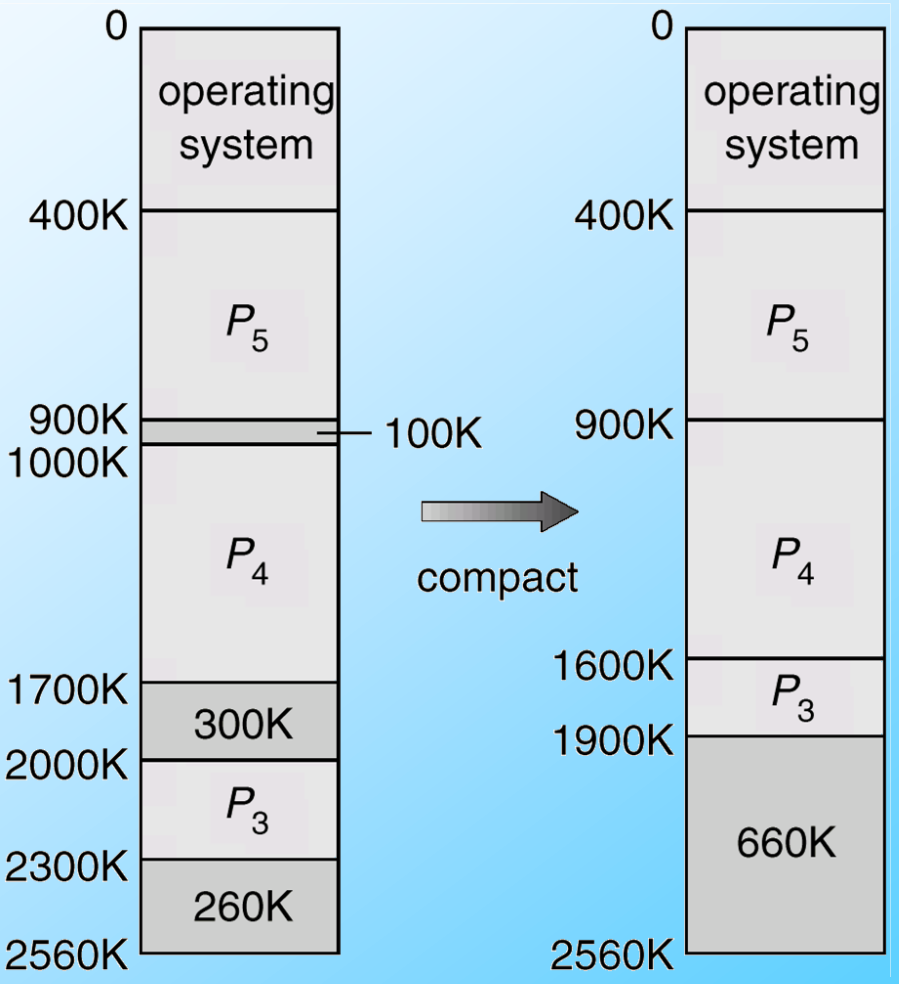
- **First-fit**: Cấp phát vùng nhớ trống liên tục đầu tiên đủ lớn.
- **Best-fit**: Cấp phát vùng nhớ trống liên tục nhỏ nhất đủ lớn. Chiến lược này tạo ra lỗ trống nhỏ nhất còn thừa lại → phải tìm kiếm trên toàn bộ danh sách các vùng trống.
- **Worst-fit**: cấp phát vùng nhớ trống liên tục lớn nhất đủ lớn → phải tìm kiếm trên toàn bộ danh sách.

First-fit tốt hơn về tốc độ và Best-fit tối ưu hóa việc sử dụng bộ nhớ.

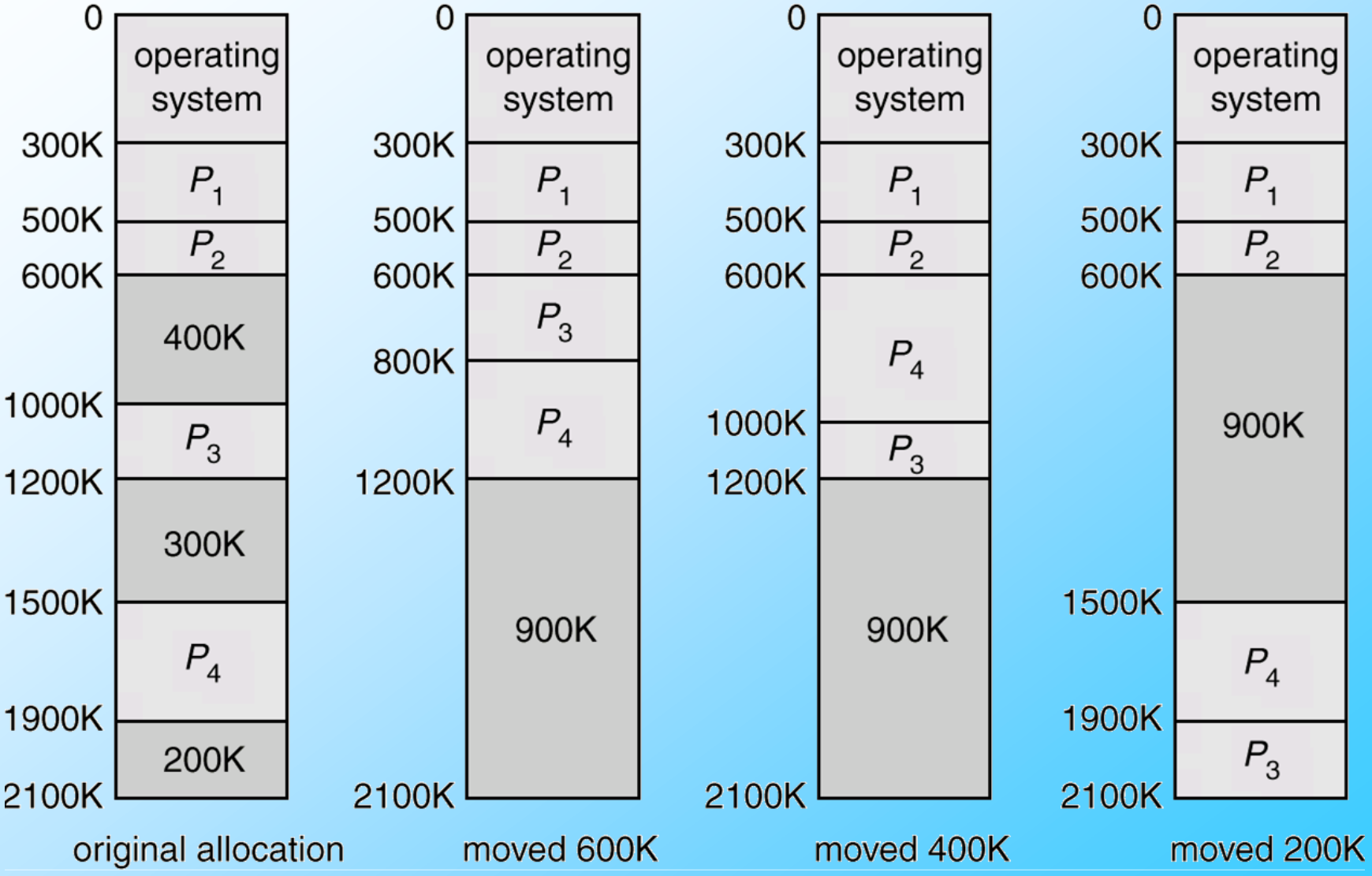
Tái định vị các tiến trình

- Kết hợp các mảnh bộ nhớ trống nhỏ rời rạc thành một vùng nhớ trống lớn liên tục.
- Đòi hỏi nhiều thời gian xử lý, ngoài ra sự kết buộc địa chỉ phải thực hiện vào thời điểm xử lý vì các tiến trình có thể bị di chuyển trong quá trình dồn bộ nhớ.

Relocation


0
operating system
400K
P_5
900K
100K
1000K
P_4
1700K
300K
2000K
P_3
2300K
260K
2560K
compact
0
operating system
400K
P_5
900K
P_4
1600K
P_3
1900K
660K
2560K

Relocation


0
operating system
300K
P_1
500K
P_2
600K
400K
1000K
P_3
1200K
300K
1500K
P_4
1900K
200K
2100K
original allocation
0
operating system
300K
P_1
500K
P_2
600K
P_3
800K
P_4
1200K
900K
2100K
moved 600K
0
operating system
300K
P_1
500K
P_2
600K
P_4
1000K
P_3
1200K
900K
2100K
moved 400K
0
operating system
300K
P_1
500K
P_2
600K
900K
1500K
P_4
1900K
P_3
2100K
moved 200K

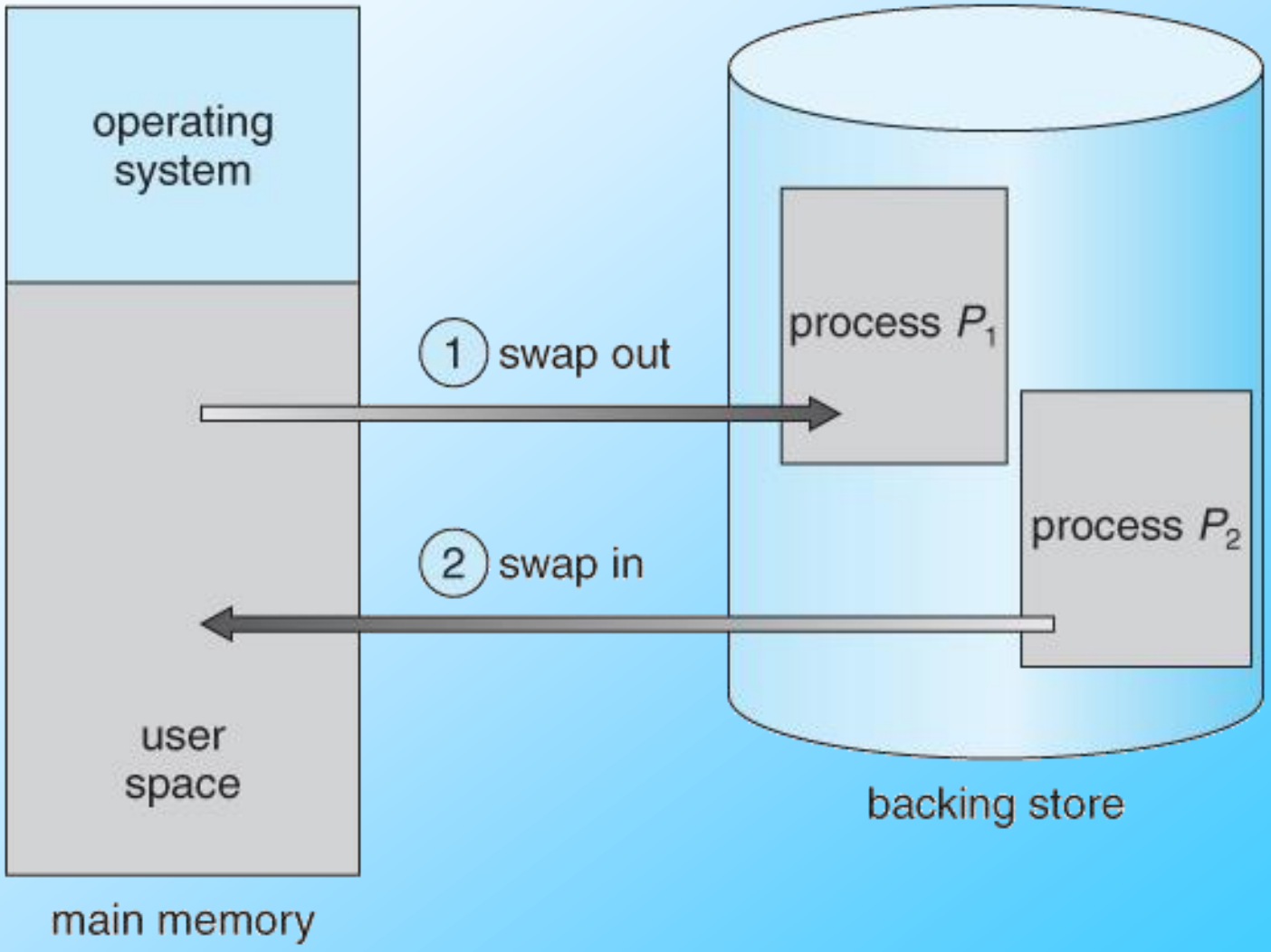
Sử dụng kỹ thuật hoán vị (swapping)

- Chuyển tạm một vài tiến trình đang trong trạng thái blocked hoặc ready ra bộ nhớ phụ, giải phóng bộ nhớ chính để có vùng nhớ trống liên tục đủ lớn cho việc nạp chương trình mới (swap out).
- Sau này chương trình bị chuyển tạm ra bộ nhớ phụ sẽ được nạp trở lại vào bộ nhớ chính để tiếp tục thực thi (swap in).
- Hệ thống bị chậm lại do thời gian hoán vị một chương trình cũng khá lớn.

Swapping

- Nâng cao mức độ đa chương
- Khi tiến trình được mang lại bộ nhớ chính, nó sẽ được nạp vào vùng nhớ nào?
  - Kết buộc lúc nạp → phải đưa vào vùng nhớ cũ của nó
  - Kết buộc lúc thi hành → thay đổi thanh ghi tái định vị
- Cần sử dụng bộ nhớ phụ đủ lớn để lưu các tiến trình bị khóa.
- Thời gian swap chủ yếu là thời gian chuyển tiến trình từ vùng nhớ chính đến bộ nhớ phụ → một tiến trình cần phải có khoảng thời gian xử lý đủ lớn.

Mô hình thao tác swapping



Sử dụng kỹ thuật phủ lấp (overlay)

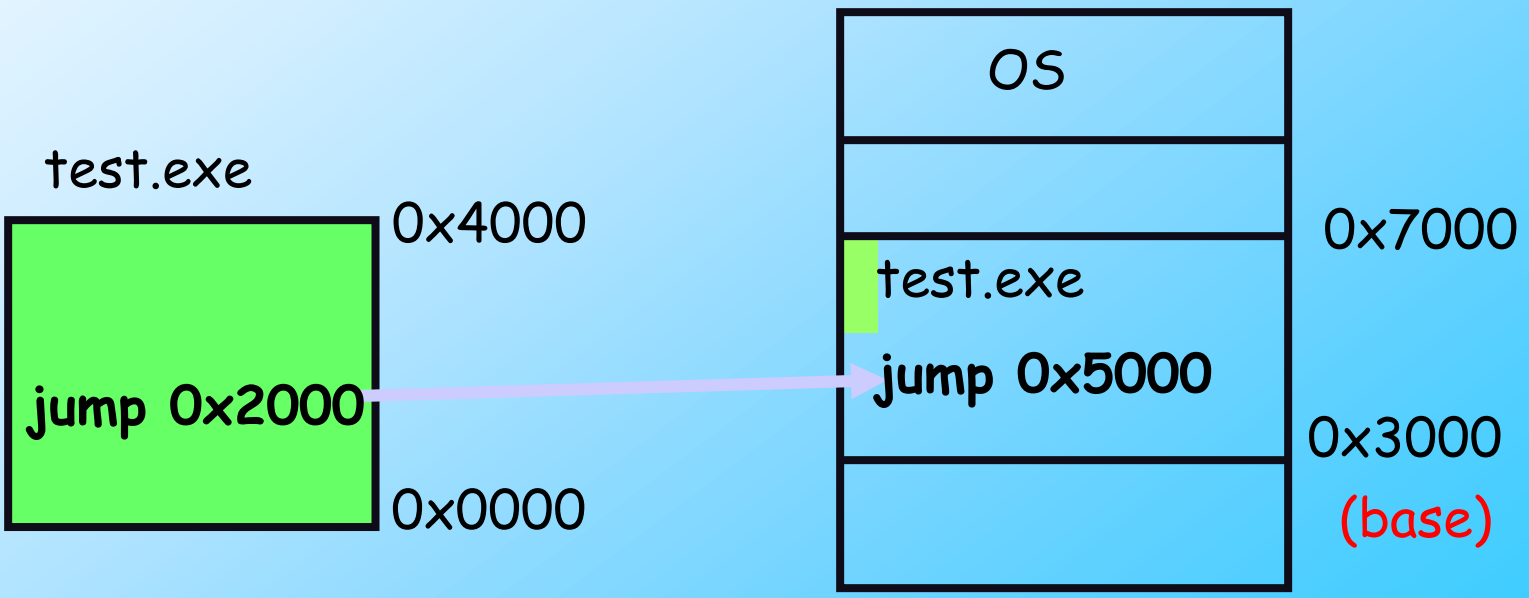
- Chia chương trình (process) thành nhiều phần nhỏ hơn bộ nhớ, mỗi phần là một overlay.
- Chỉ lưu trong bộ nhớ chỉ thị và dữ liệu đang cần.
- Sử dụng khi tiến trình có yêu cầu bộ nhớ lớn hơn dung lượng nhớ có thể cấp phát cho nó.
- đòi hỏi sự hỗ trợ của ngôn ngữ lập trình và người lập trình phải quan tâm đến kích thước bộ nhớ ngay khi lập trình.

Ánh xạ bộ nhớ tại thời điểm nạp chương trình

- Hệ điều hành sẽ trả về địa chỉ bắt đầu nạp tiến trình và thay các địa chỉ tham chiếu trong tiến trình (đang là địa chỉ logic) bằng địa chỉ vật lý theo công thức :

  (địa chỉ vật lý) = (địa chỉ bắt đầu) + (địa chỉ logic)

- Mô hình linker - loader



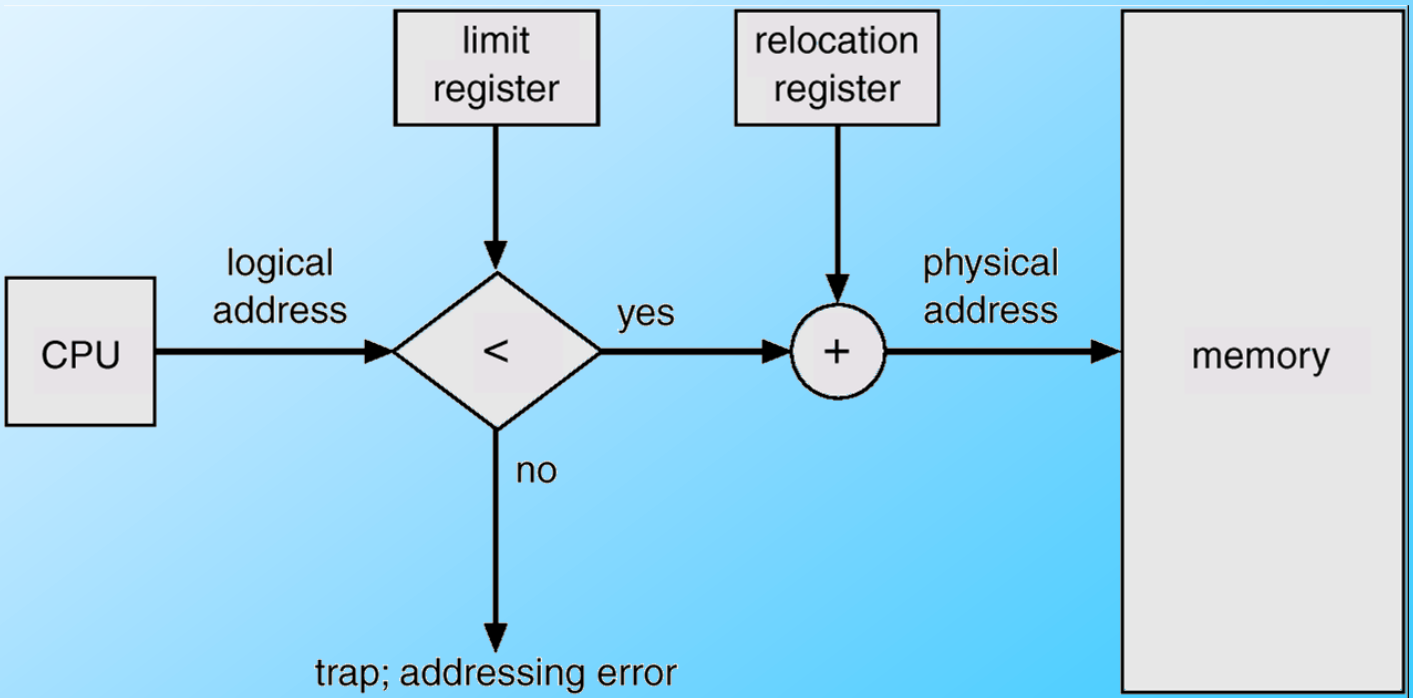
Ánh xạ bộ nhớ tại thời điểm thực thi chương trình

if (địa chỉ logic) < (giá trị thanh ghi giới hạn) then

(địa chỉ vật lý) = (giá trị thanh ghi nền) + (địa chỉ logic)

else báo lỗi

Mô hình Base and Bound:



7.3 – Cấp phát không liên tục

Cấp phát không liên tục

- Kỹ thuật phân trang
- Kỹ thuật phân đoạn
- Phân trang kết hợp phân đoạn

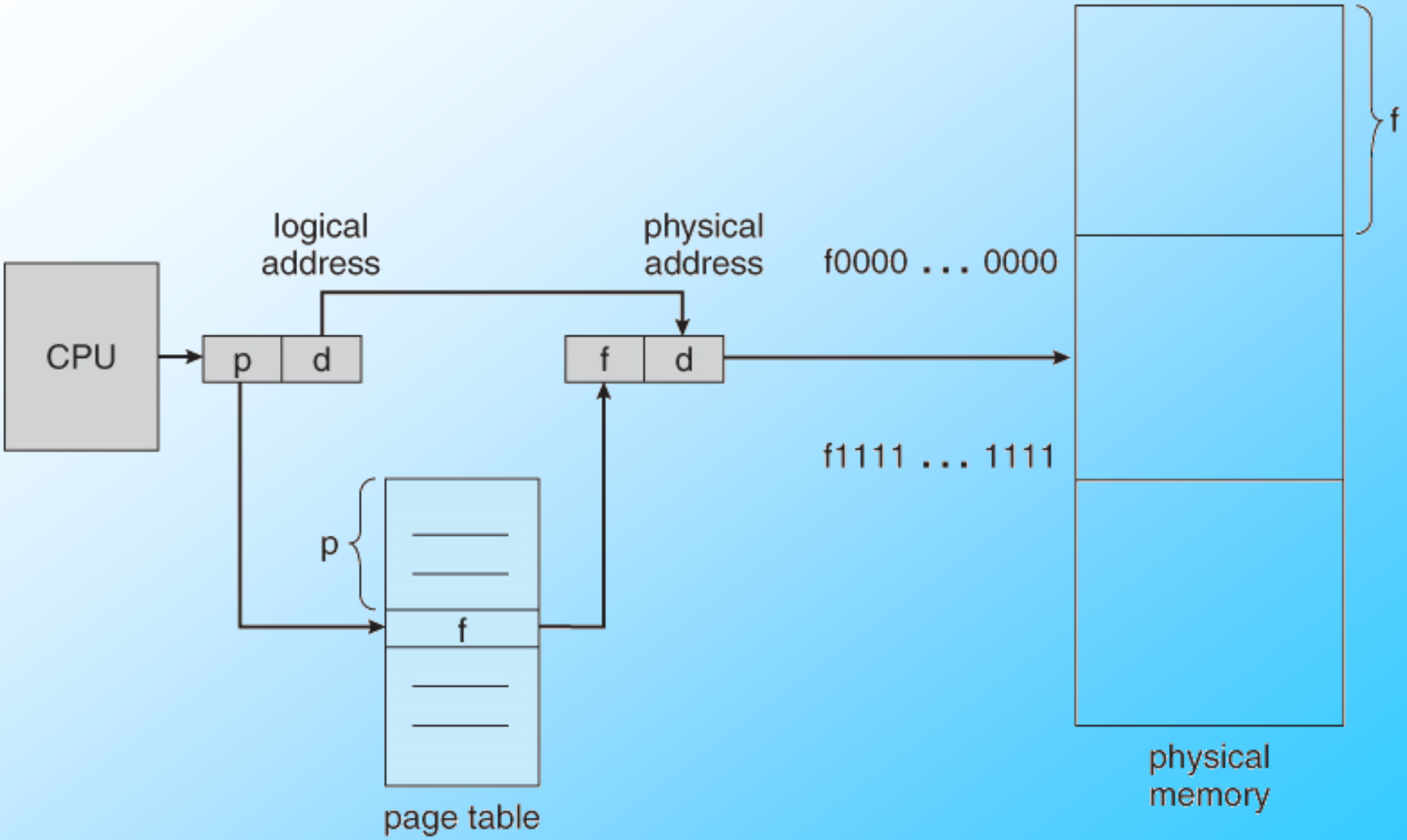
Cấp phát bộ nhớ bằng pp phân trang (Paging)

- Không gian địa chỉ lôgic của một tiến trình có thể không liên tục.
- Chia bộ nhớ vật lý thành các khối có kích thước cố định gọi là khung (frame) (kích thước là số mũ của 2, từ 512 đến 8192 bytes).
- Chia bộ nhớ lôgic thành các khối có cùng kích thước và gọi là trang (pages).
- Lưu trạng thái của tất cả các khung (frame).
- Để chạy một chương trình có kích thước n trang, cần phải tìm n khung trống và nạp chương trình vào.
- Tạo một bảng trang để chuyển đổi địa chỉ lôgic sang địa chỉ vật lý.
- Có hiện tượng phân mảnh bộ nhớ nội vi.

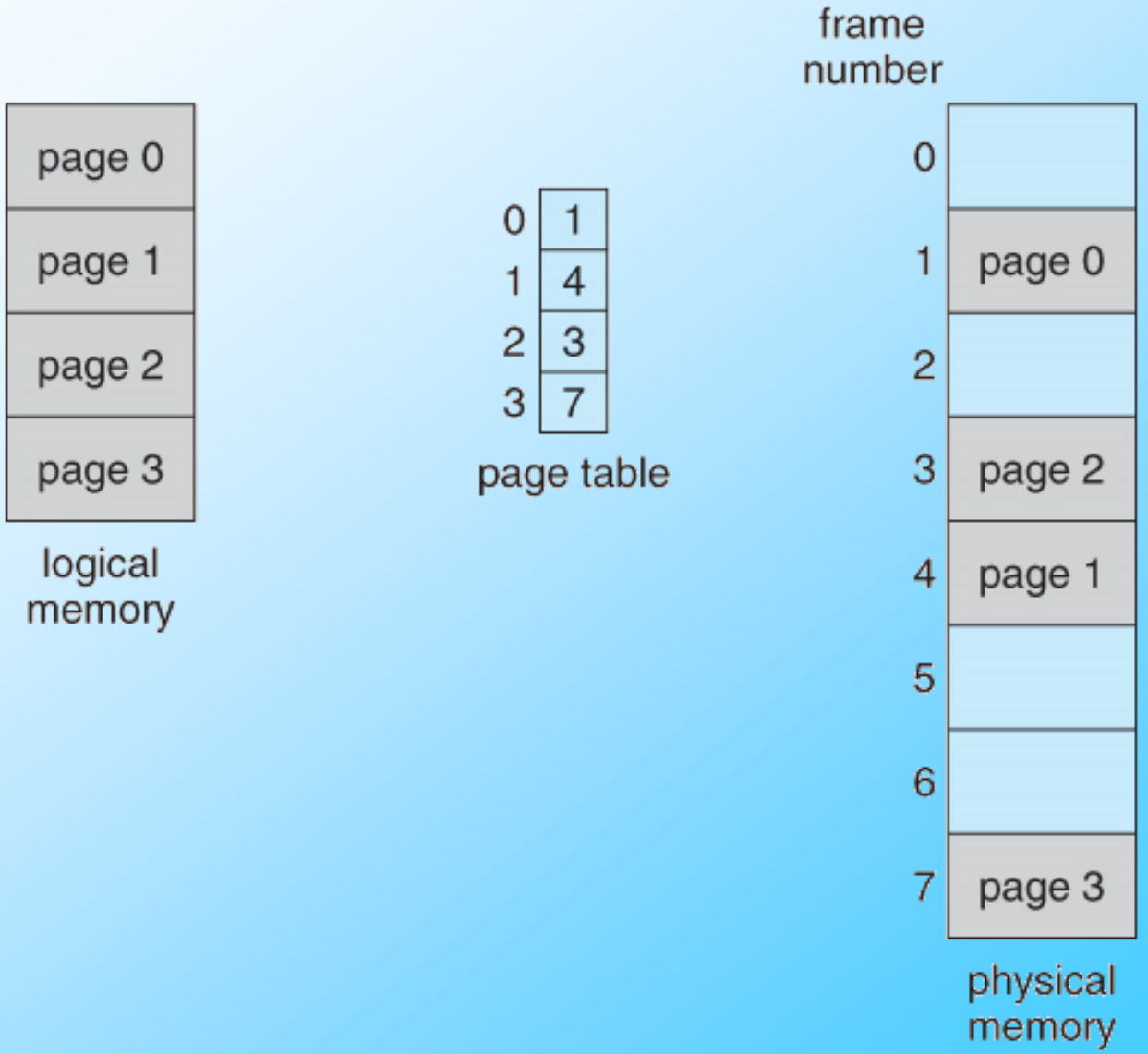
Mô hình chuyển đổi địa chỉ

- Địa chỉ được tạo ra bởi CPU gồm có hai phần:
  - *Số trang (Page number) (p)* – được dùng như là một chỉ số của một bảng trang chứa địa chỉ cơ sở của mỗi trang trong bộ nhớ vật lý.
  - *Page offset (d)* – kết hợp với địa chỉ cơ sở để định ra không gian địa chỉ vật lý được gởi đến bộ nhớ.

Kiến trúc chuyển đổi địa chỉ

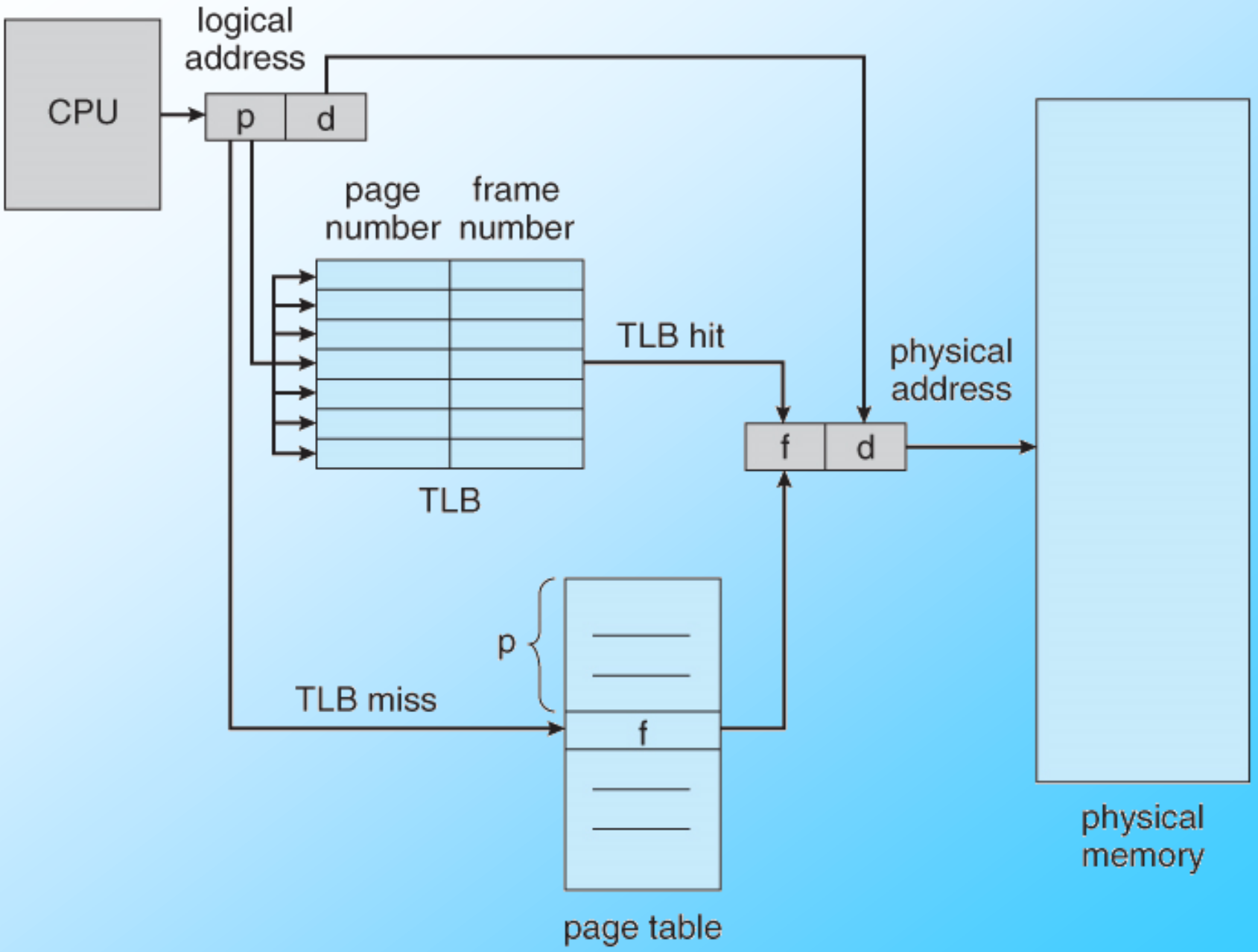


Ví dụ trang


page 0
page 1
page 2
page 3
logical memory
0 1
1 4
2 3
3 7
page table
frame number
0
1 page 0
2
3 page 2
4 page 1
5
6
7 page 3
physical memory

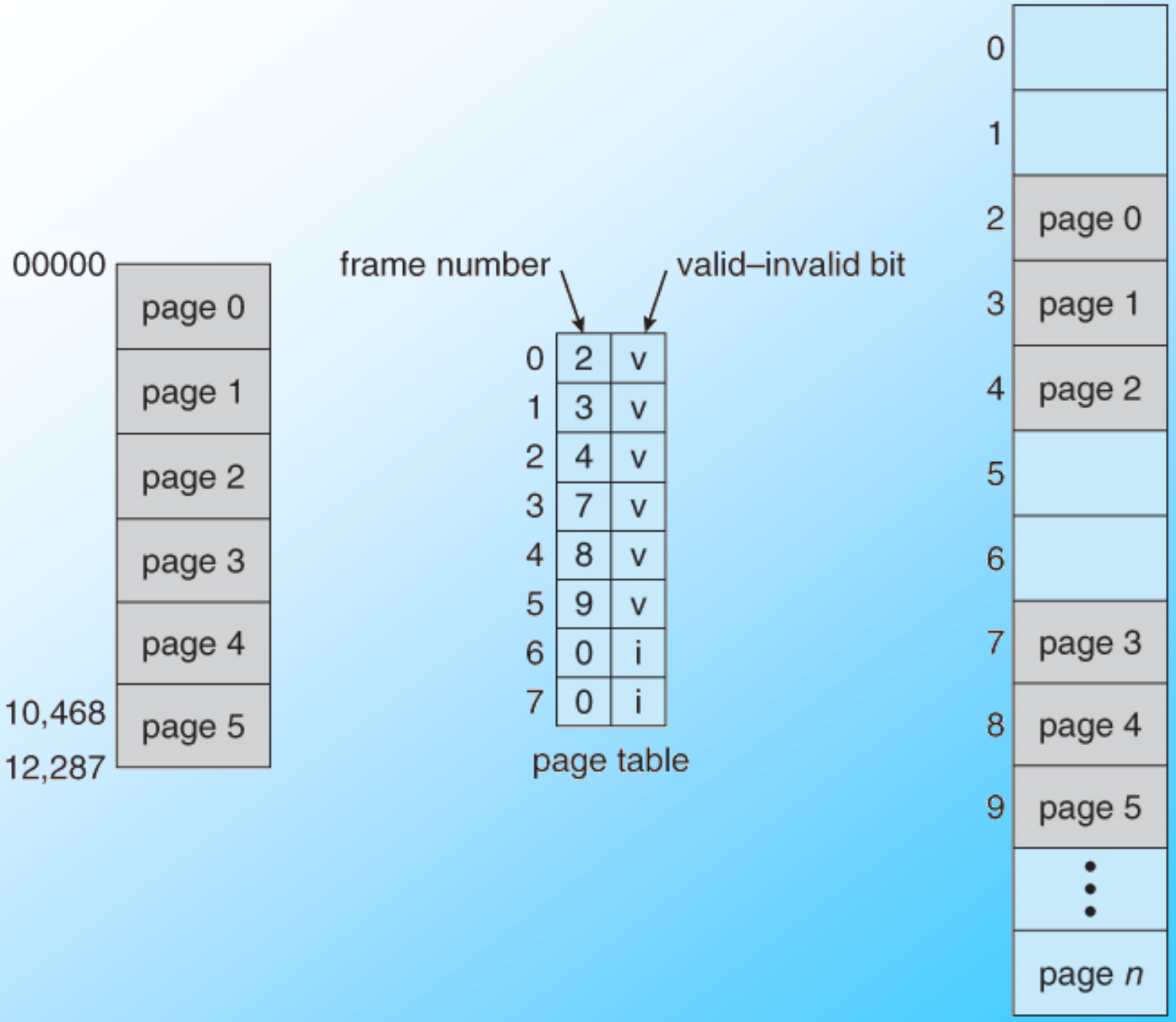
Cài đặt bảng trang

- Bảng trang được đặt trong bộ nhớ.
- *Page-table base register (*PTBR) chỉ đến bảng trang.
- *Page-table length register* (PRLR) cho biết kích thước của bảng trang.
- Với mô hình này, mọi sự truy cập chỉ thị/dữ liệu đều đòi hỏi hai lần truy cập vùng nhớ: 1) truy cập bảng trang; 2) chỉ thị hoặc dữ liệu → có vẻ chậm.
- Khắc phục vấn đề hai lần truy cập vùng nhớ bằng cách sử dụng một vùng đệm phần cứng tra cứu nhanh đặc biệt (special fast-lookup hardware) gọi là *associative registers* hoặc *translation look-aside buffers (TLBs)*


CPU
logical address
p
d
page number
frame number
TLB hit
physical address
f
d
TLB
p
TLB miss
f
page table
physical memory

Bảo vệ vùng nhớ

- Làm sao biết trang nào của tiến trình nào? Cần bảo vệ các tiến trình truy xuất vào trang không phải của mình.
- Việc bảo vệ vùng nhớ được cài đặt bằng cách liên kết một khung với một bit, gọi là bit kiểm tra hợp lệ (valid-invalid bit).
- *Valid-invalid* bit được đính kèm vào mỗi ô trang bảng trang:
  - "valid" chỉ ra rằng trang đi kèm là nằm trong không gian địa chỉ lôgic của tiến trình vì vậy truy xuất trang này là hợp lệ.
  - "invalid" chỉ ra rằng trang đi kèm không nằm trong không gian địa chỉ lôgic của tiến trình.


00000
page 0
page 1
page 2
page 3
page 4
10,468
page 5
12,287
frame number
valid–invalid bit
0 2 v
1 3 v
2 4 v
3 7 v
4 8 v
5 9 v
6 0 i
7 0 i
page table
0
1
2 page 0
3 page 1
4 page 2
5
6
7 page 3
8 page 4
9 page 5
page n

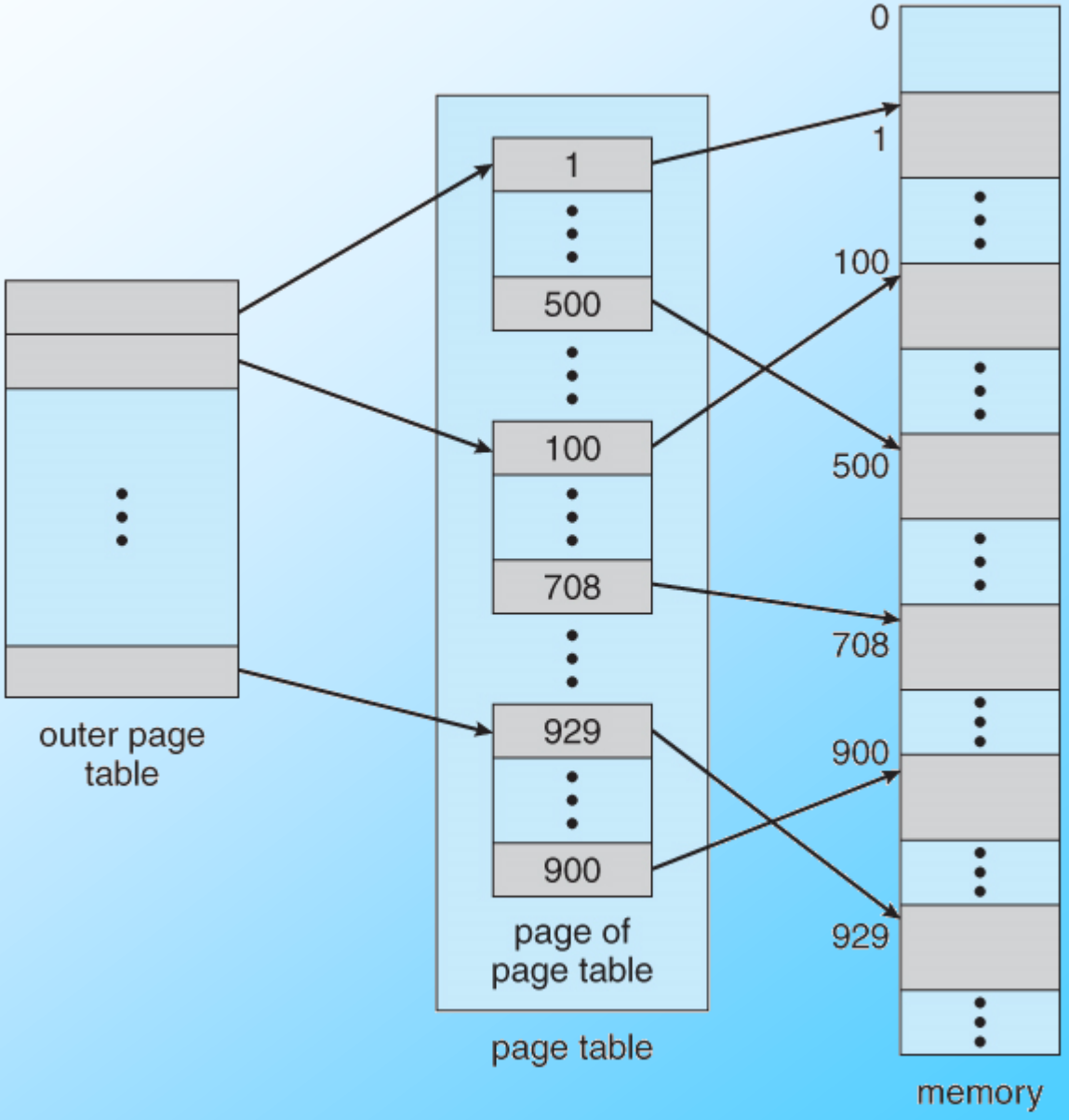
Tổ chức bảng trang

- Chỉ một bảng trang (cho mỗi tiến trình)
- Tiến trình nhiều trang → quản lý bộ nhớ rất lớn → số trang nhiều → kích thước của bảng trang phải lớn → không tối ưu.
- Giải pháp:
  - Phân trang đa cấp (multilevel paging)
  - Bảng trang nghịch đảo

Phân trang đa cấp

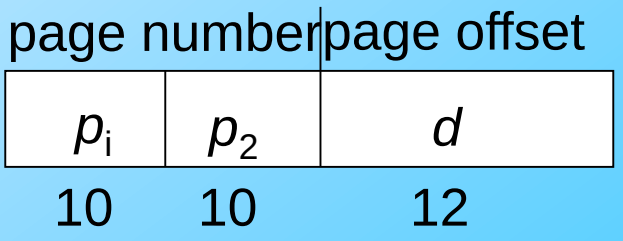
- Phân chia bảng trang thành các phần nhỏ
- Bản thân bảng trang cũng được phân trang

Mô hình phân trang hai cấp



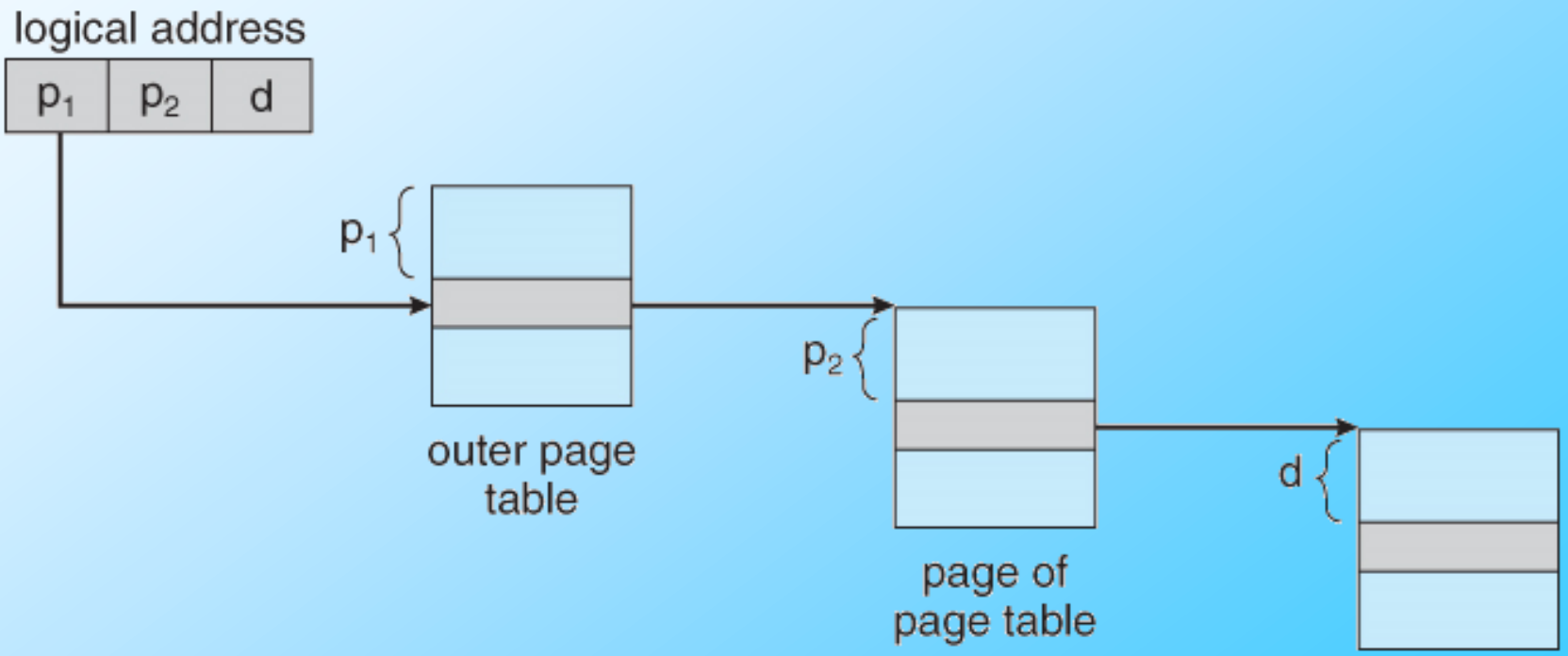
Ví dụ phân trang 2 cấp

- Một địa chỉ lôgic (trên máy 32 bit với kích thước trang là 4K) được chia thành:
  - Page number: 20 bit.
  - Page offset: 12 bit.
- Bởi vì bảng trang được phân trang, số trang tiếp tục được phân chia thành:
  - Page number: 10-bit.
  - Page offset: 10 bit.
- Địa chỉ lôgic sẽ có cấu trúc như sau:



Chuyển đổi địa chỉ

- Address-translation scheme for a two-level 32-bit paging architecture



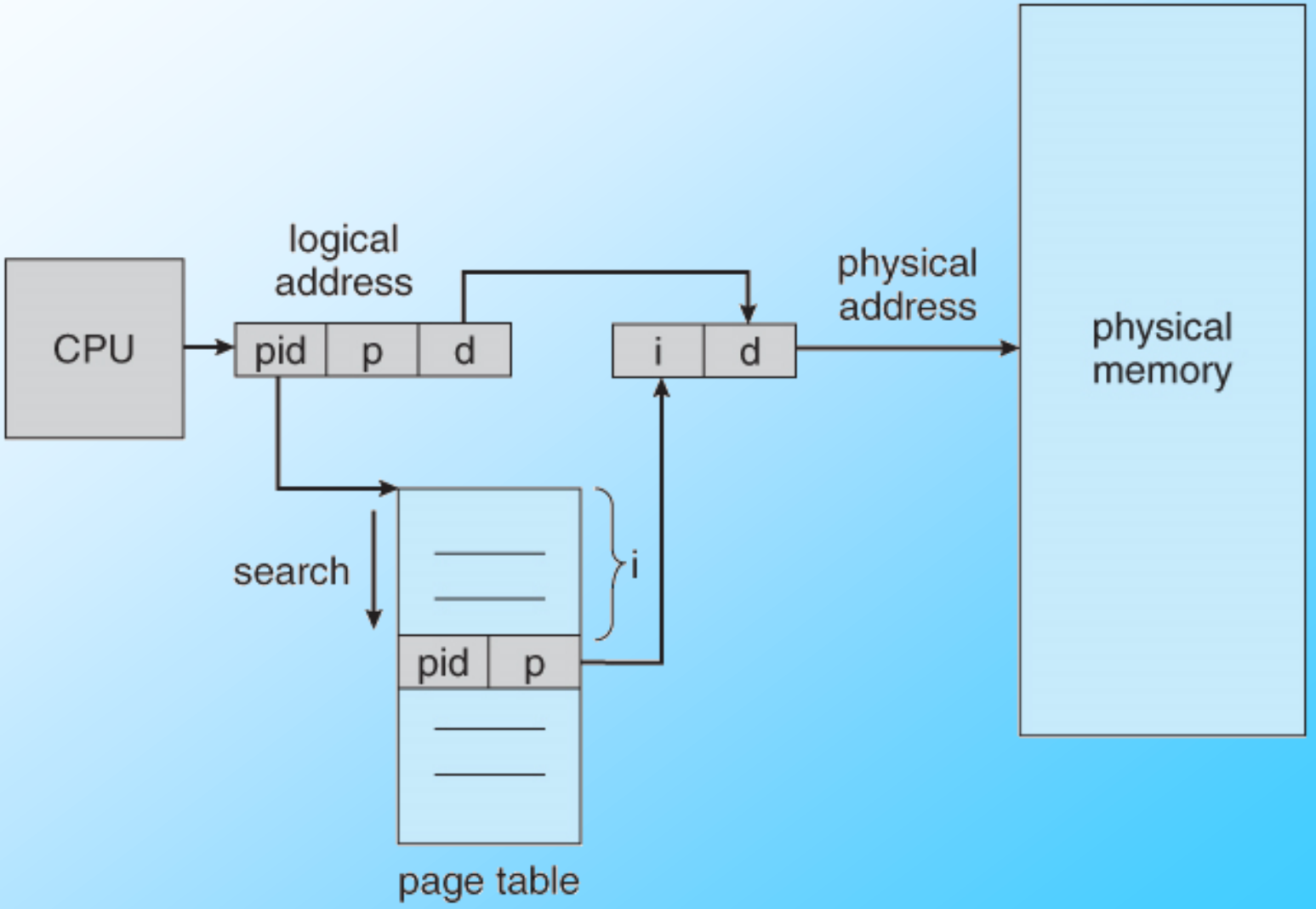
Phân tích

- Nếu chỉ dùng một trang
  - bảng trang sẽ có 2^{20} phần tử.
- Nếu dùng phân trang 2 cấp
  - Bảng trang ngoài cần 2^{10} phần tử
  - Bảng trang trong vẫn có 2^{20} phần tử nhưng có thể được đặt ở vùng nhớ phụ

Bảng trang nghịch đảo

- Sử dụng một bảng trang duy nhất cho mọi tiến trình
- Một entry cho mỗi khung (bộ nhớ vật lý).
- Một entry bao gồm địa chỉ ảo của khung và tiến trình sở hữu khung đó.
- Giảm kích thước lưu trữ bảng trang, nhưng tăng thời gian tìm kiếm bảng trang.
- Dùng bảng băm để tăng tốc tìm kiếm.

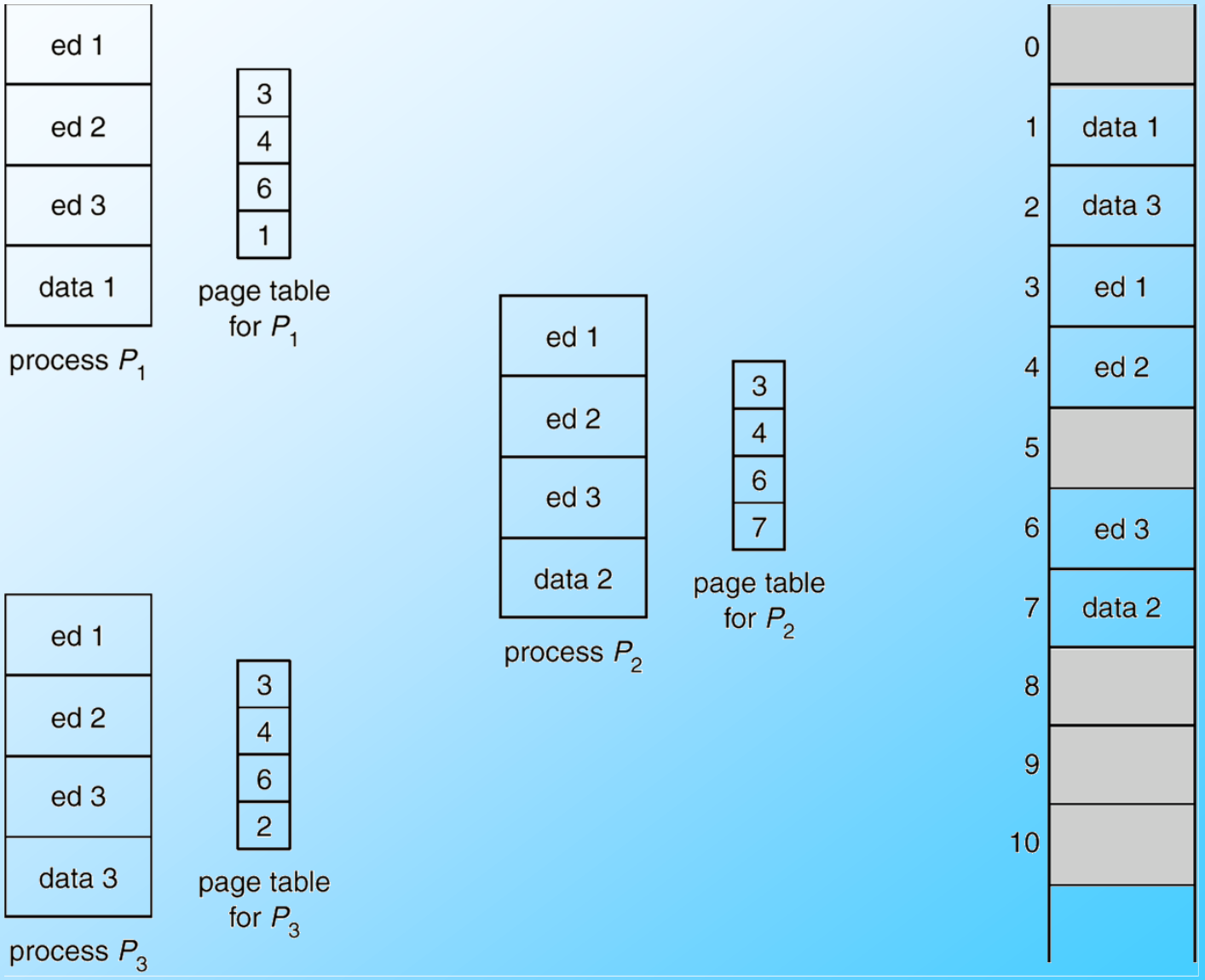
Kiến trúc bảng trang nghịch đảo



Chia sẻ và bảo vệ

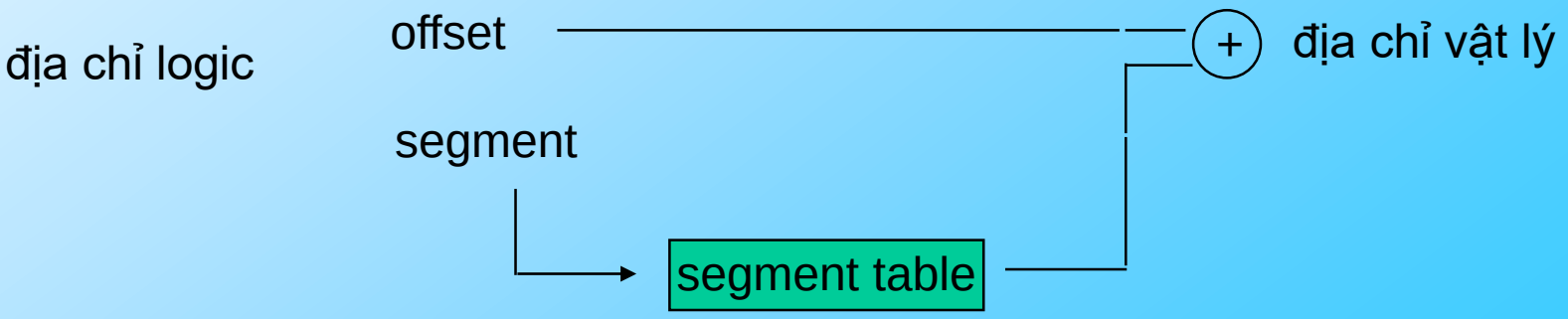
- Chia sẻ và bảo vệ
  - Ở cấp độ trang
  - Đứng ở khía cạnh người dùng, khá bất tiện
    - Vd: Đoạn mã nằm trên nhiều trang → phải chia sẻ tất cả các trang đó với nhau.
  - Đoạn mã phải nằm ở một vị trí giống nhau trông tất cả các không gian địa chỉ của của tiến trình muốn chia sẻ

Ví dụ chia sẻ trang

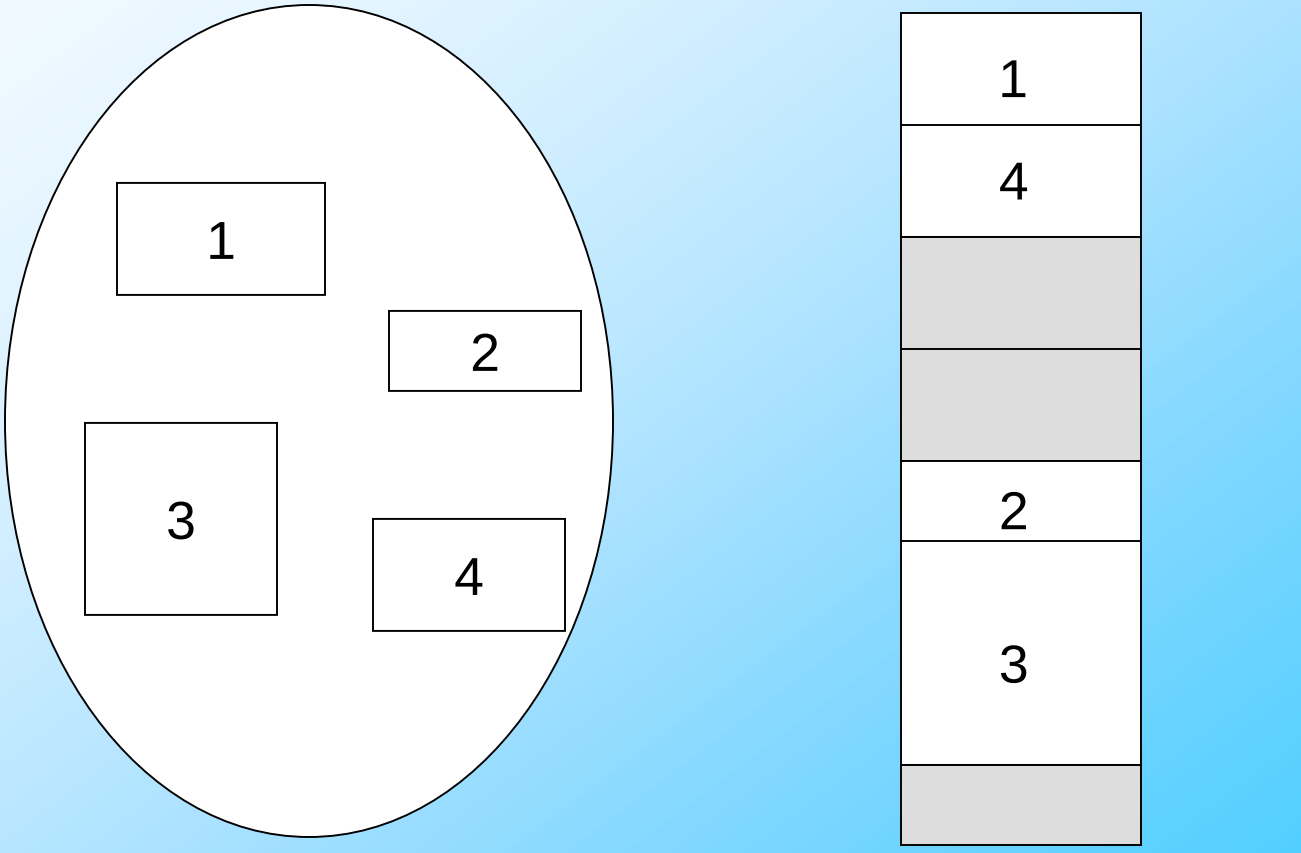


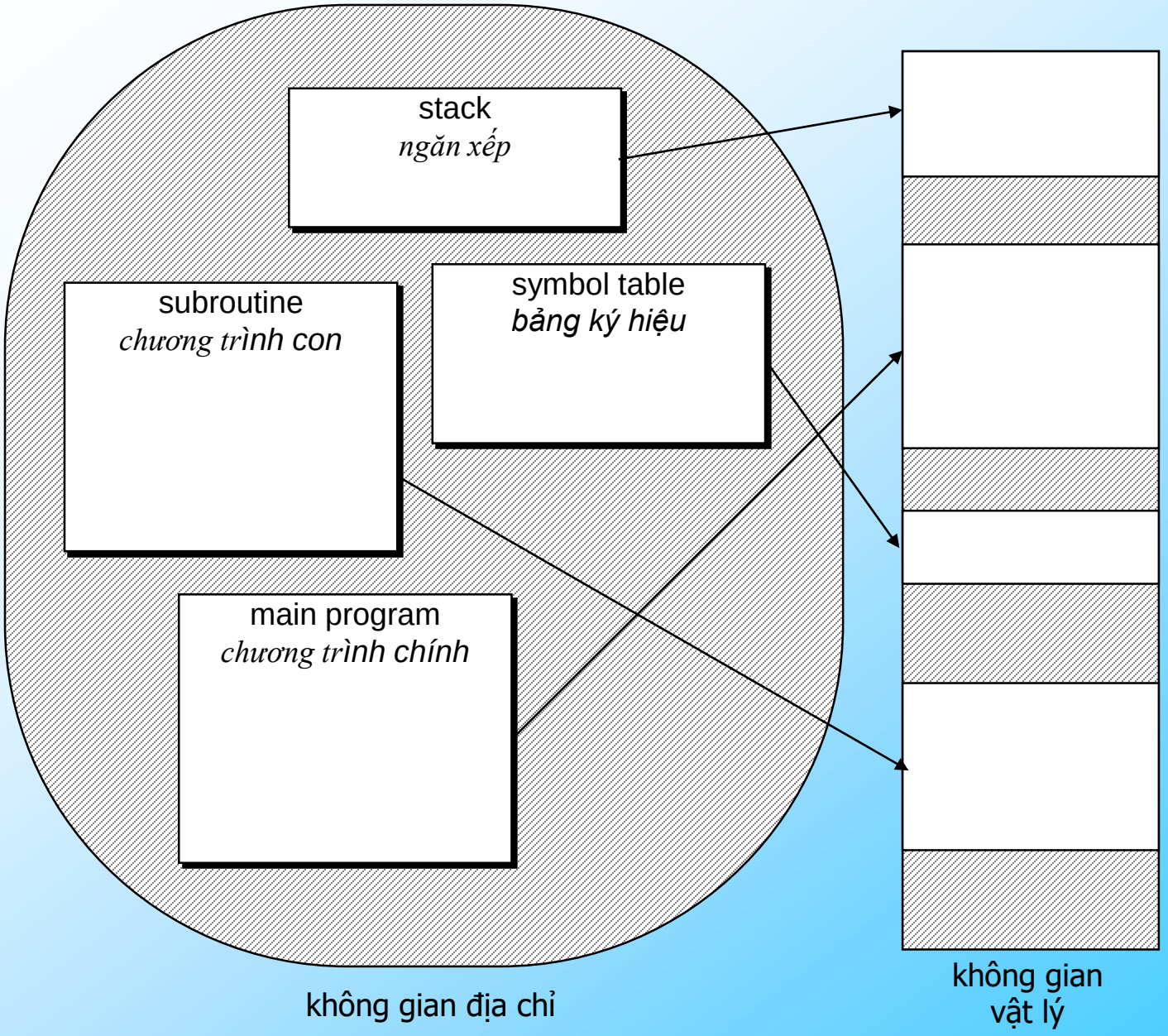
Phân đoạn (Segmentation)

- Hỗ trợ quản lý bộ nhớ theo góc độ người dùng.
- Một chương trình bao gồm nhiều phân đoạn (segment):
  - Đoạn mã
  - Biến toàn cục, dữ liệu
  - stack
  - heap



Góc độ người dùng




stack
ngăn xếp
subroutine
chương trình con
symbol table
bảng ký hiệu
main program
chương trình chính
không gian địa chỉ
không gian
vật lý

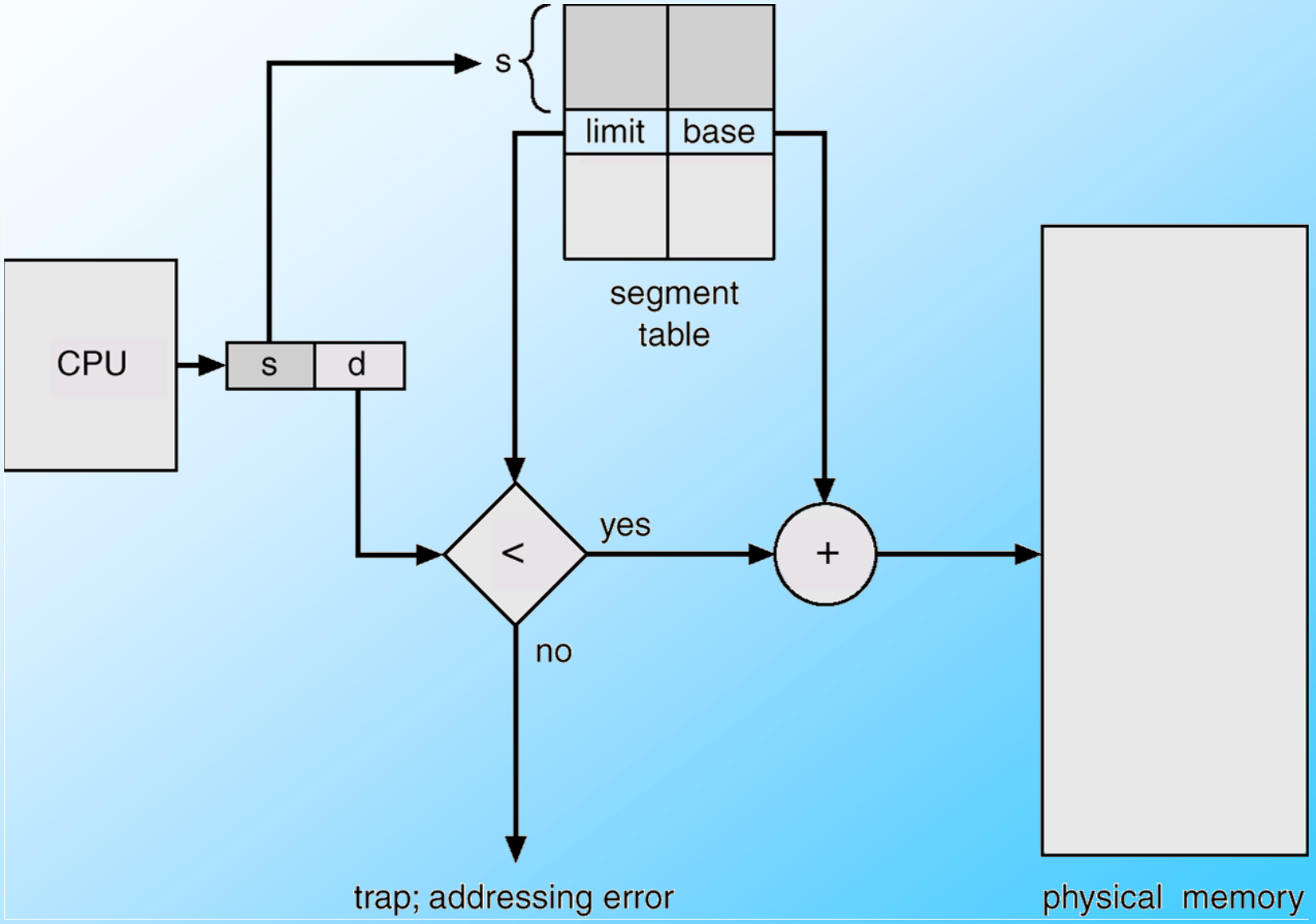
Kiến trúc phân đoạn

- Địa chỉ lôgic = <**segment-number, offset**>
- *Bảng phân đoạn (Segment table)* – chuyển đổi địa chỉ hai chiều thành địa chỉ vật lý một chiều. Mỗi ô trong bảng gồm có:
  - Thanh ghi nền (base) – chứa địa chỉ vật lý bắt đầu của phân đoạn trong bộ nhớ chính.
  - *Thanh ghi giới hạn (limit)* – chỉ kích thước của phân đoạn.
- *Segment-table base register (STBR)* lưu trữ địa chỉ của bảng phân đoạn trong vùng nhớ.
- *Segment-table length register (STLR)* chỉ số segment được sử dụng bởi 1 chương trình

Chuyển đổi địa chỉ

- Mỗi địa chỉ ảo là một bộ <***s,d***> :
- *số hiệu phân đoạn s* : được sử dụng như chỉ mục đến bảng phân đoạn
- *địa chỉ tương đối d* : có giá trị trong khoảng từ 0 đến giới hạn chiều dài của phân đoạn. Nếu địa chỉ tương đối hợp lệ, nó sẽ được cộng với giá trị chứa trong thanh ghi nền để phát sinh địa chỉ vật lý tương ứng

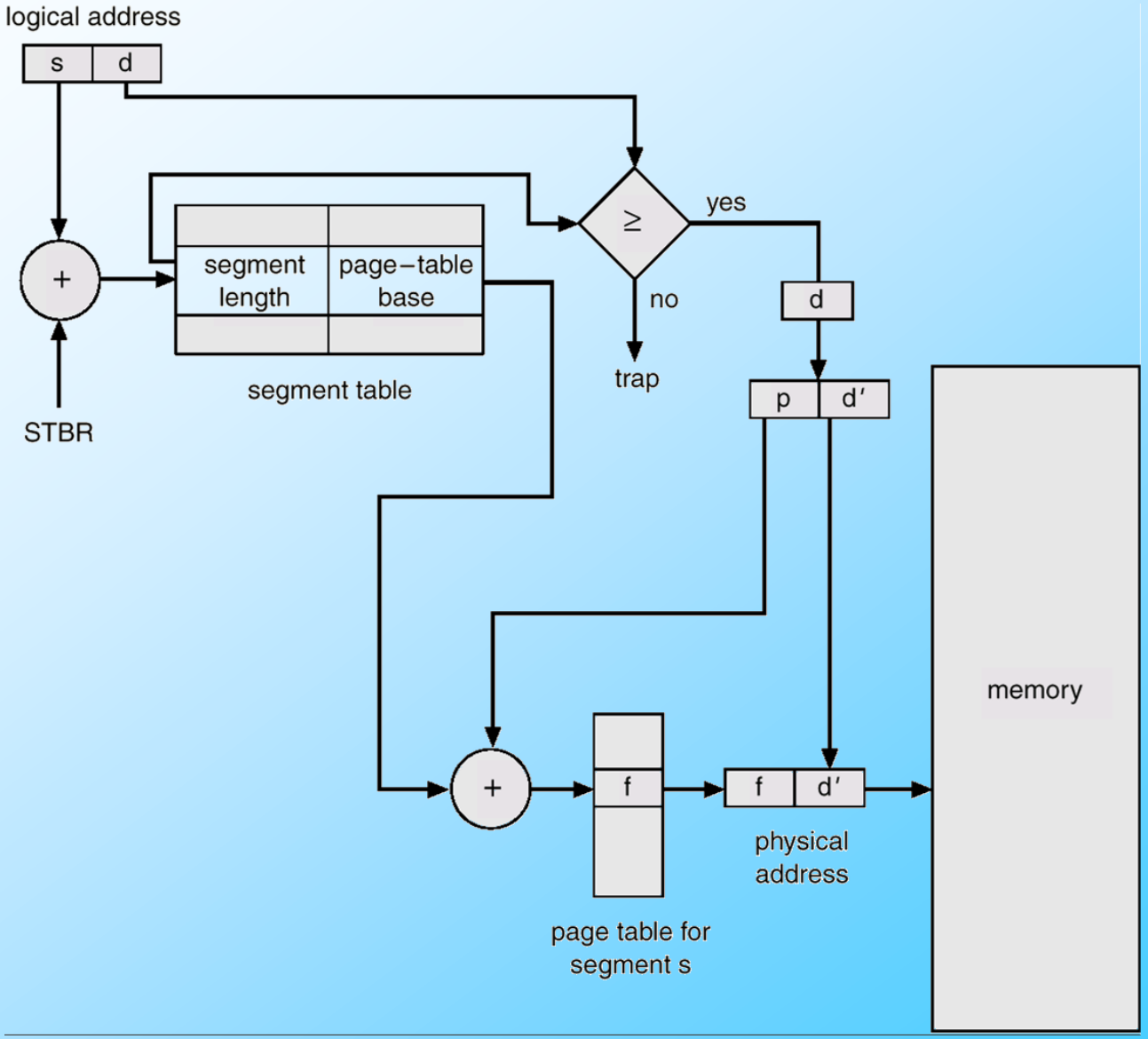
Chuyển đổi địa chỉ



Kiến trúc phân đoạn (tt)

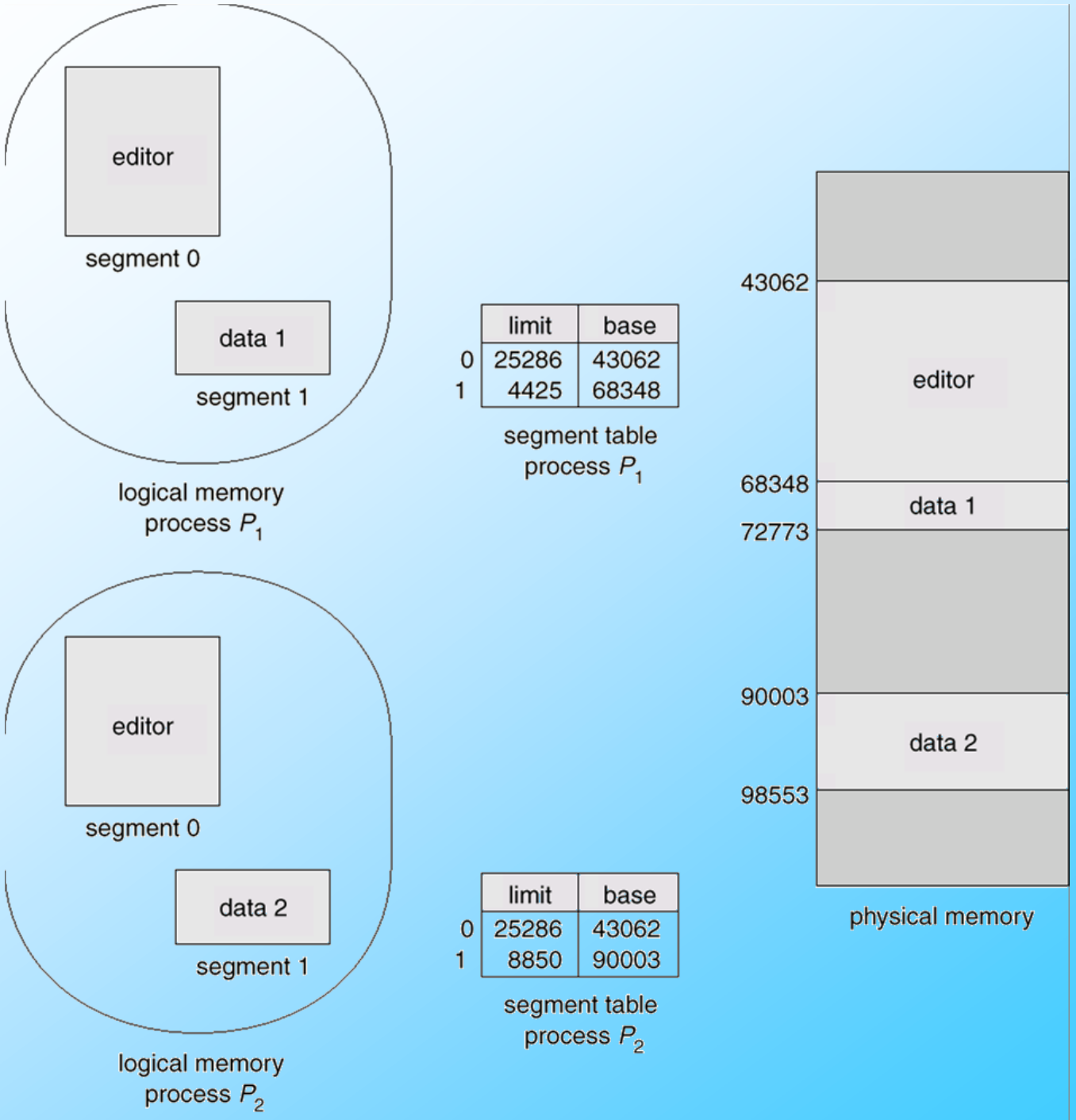
- Tái định vị.
  - Động (dynamic partition)
  - Thông qua bảng phân đoạn
- Chia sẻ.
  - Có thể chia sẻ các phân đoạn giữa các chương trình
  - Sử dụng chung chỉ số segment
- Cấp phát.
  - first fit/best fit
  - Có hiện tượng phân mảnh ngoại vi

Chuyển đổi địa chỉ



Kiến trúc phân đoạn (tt)

- Bảo vệ
  - Mỗi entry thêm một bit "valid bit". Nếu valid bit = 0 $\Rightarrow$ truy cập phân đoạn không hợp lệ
  - Hỗ trợ phân quyền theo từng thao tác read/write/execute

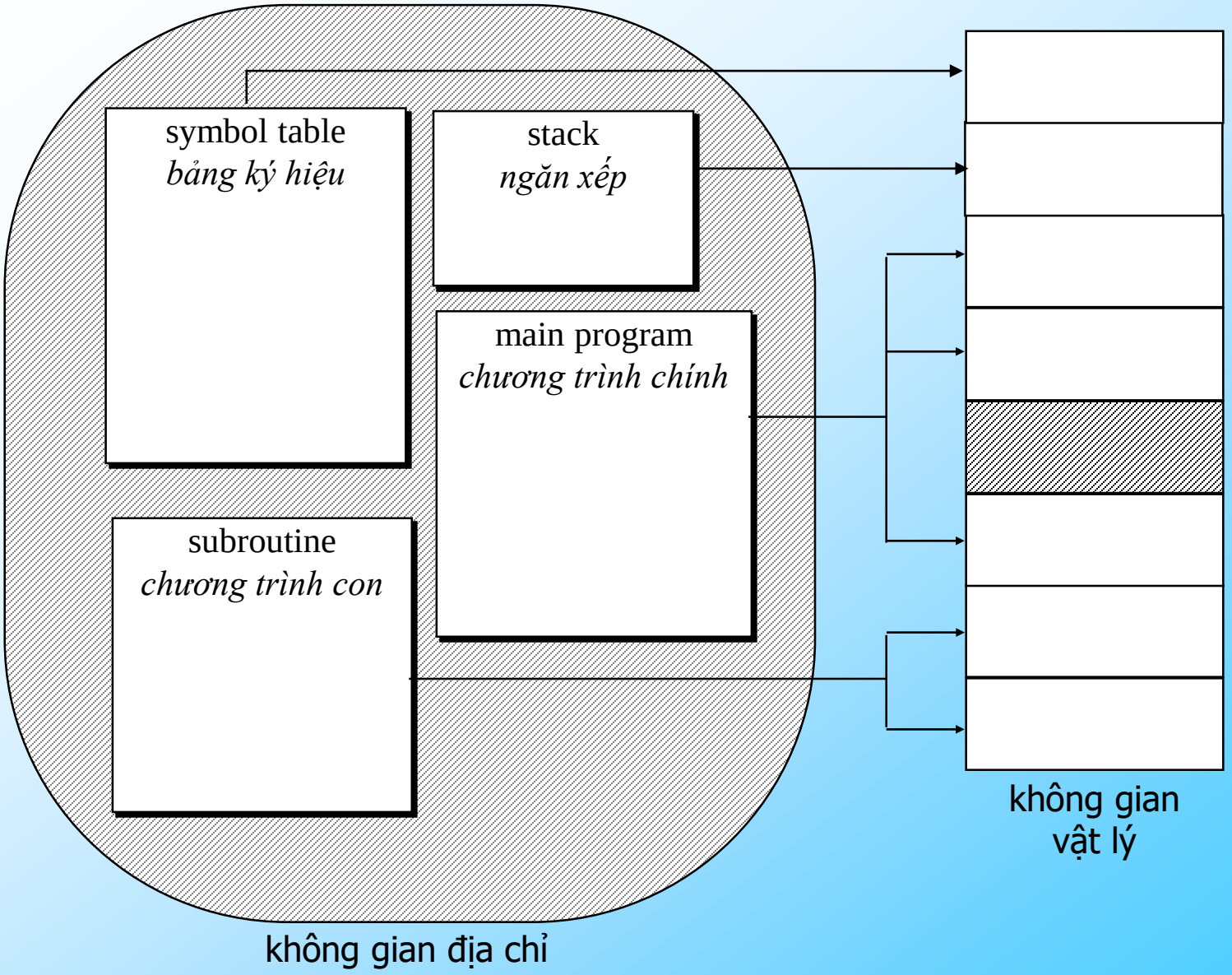

editor
segment 0
data 1
segment 1
logical memory
process P_1
limit
base
0
25286
43062
1
4425
68348
segment table
process P_1
43062
editor
68348
data 1
72773
90003
data 2
98553
physical memory
editor
segment 0
data 2
segment 1
logical memory
process P_2
limit
base
0
25286
43062
1
8850
90003
segment table
process P_2

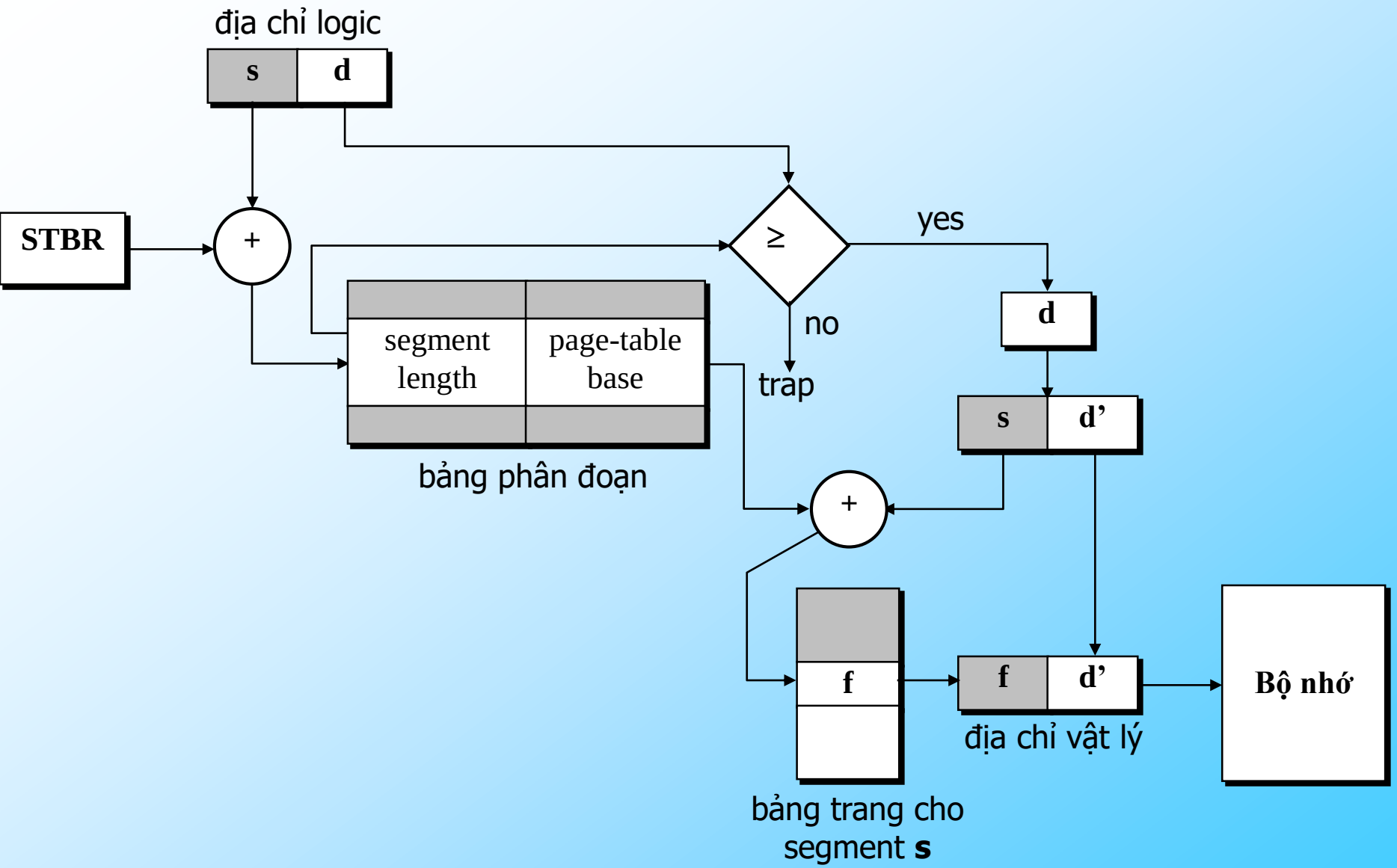
Kết hợp phân trang và phân đoạn

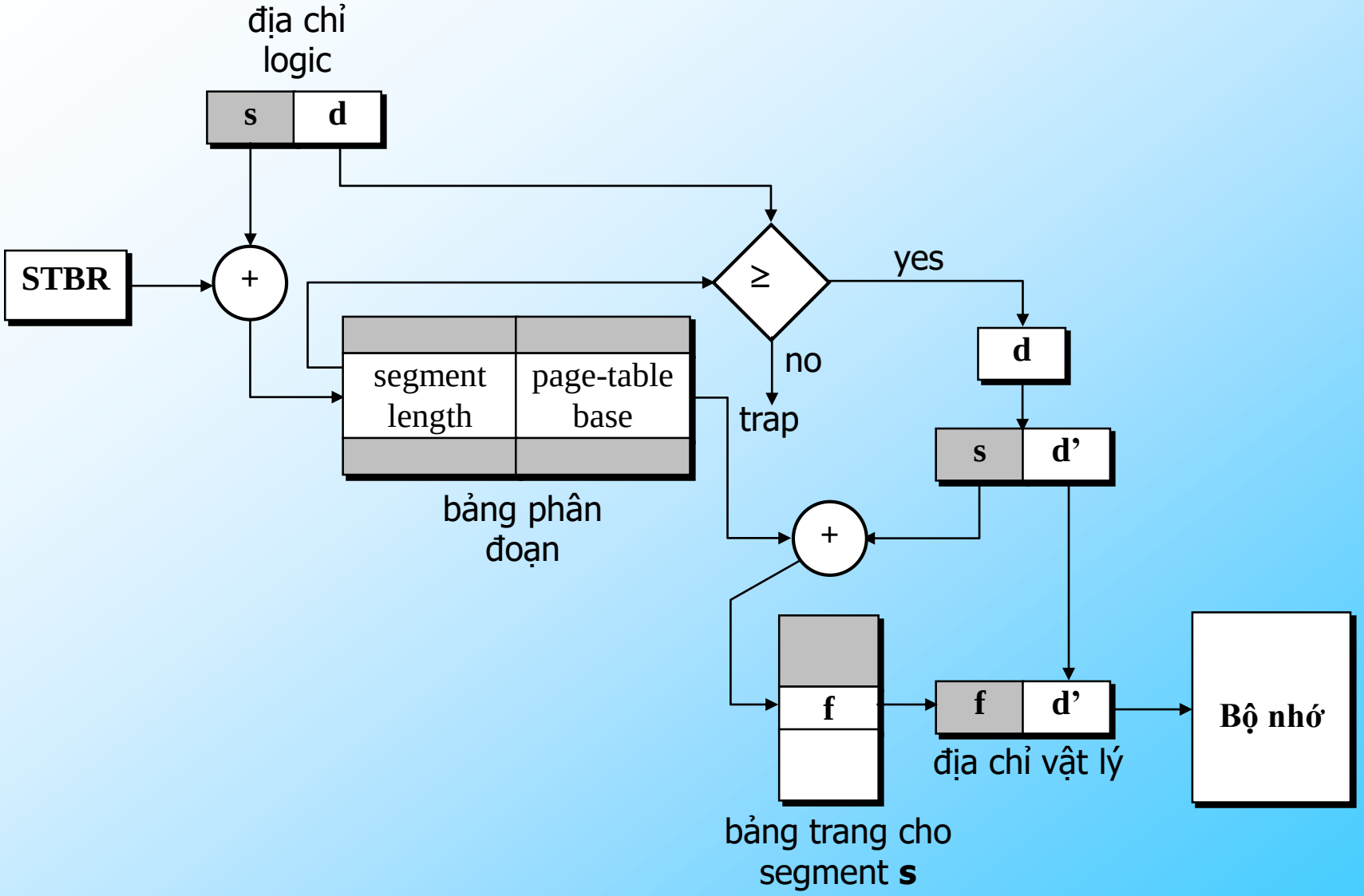
- Ý tưởng:
  - Phân trang trong mỗi phân đoạn
  - Bộ nhớ = nhiều phân đoạn
  - Phân đoạn = nhiều trang
- Giải quyết tình trạng phân mảnh ngoại vi
- Mỗi phần tử của bảng phân đoạn gồm hai thành phần:
  - Thanh ghi giới hạn (limit): kích thước của phân đoạn (giống với phân đoạn thuần)
  - Thanh ghi cơ sở (base): chứa địa chỉ của bảng trang của phân đoạn đó (khác với phân đoạn thuần)

Chuyển đổi địa chỉ

- Mỗi địa chỉ logic là một bộ ba: <***s,p,d***>
  - *số hiệu phân đoạn (s)*: sử dụng như chỉ mục đến phần tử tương ứng trong bảng phân đoạn.
  - *số hiệu trang (p)*: sử dụng như chỉ mục đến phần tử tương ứng trong bảng trang của phân đoạn.
  - *địa chỉ tương đối trong trang (d): kết hợp với địa chỉ bắt đầu của trang để tạo ra địa* chỉ vật lý mà trình quản lý bộ nhớ sử dụng.


symbol table
bảng ký hiệu
stack
ngăn xếp
main program
chương trình chính
subroutine
chương trình con
không gian địa chỉ
không gian
vật lý


địa chỉ logic
s
d
STBR
+
≥
yes
no
trap
segment length
page-table base
bảng phân đoạn
d
s
d'
+
f
f
d'
địa chỉ vật lý
Bộ nhớ
bảng trang cho segment s


địa chỉ
logic
s
d
STBR
+
≥
yes
no
trap
segment
length
page-table
base
bảng phân
đoạn
d
s
d'
+
f
f
d'
địa chỉ vật lý
Bộ nhớ
bảng trang cho
segment s

CÂU HỎI ÔN TẬP BÀI 7

1. Phân biệt địa chỉ logic và địa chỉ vật lý? Cách chuyển đổi địa chỉ logic sang địa chỉ vật lý.
2. Vẽ sơ đồ ánh xạ và bảo vệ bộ nhớ trong kỹ thuật cấp phát bộ nhớ liên tục.
3. Giả sử giá trị của thanh ghi nền là 8192, thanh ghi giới hạn là 16384. Hãy tính địa chỉ logic của ô nhớ có địa chỉ vật lý là 12288
4. Cho biết sự khác nhau giữa phương pháp cấp phát liên tục và phương pháp cấp phát không liên tục. Ưu và nhược điểm của phương pháp cấp phát không liên tục so với kỹ thuật cấp phát liên tục.

CÂU HỎI ÔN TẬP BÀI 7

5. Hãy trình bày phương pháp cấp phát theo kỹ thuật phân trang. Hãy vẽ sơ đồ mô hình cơ chế chuyển đổi địa chỉ logic sang địa chỉ vật lý trong kỹ thuật phân trang.
6. Hãy trình bày mục đích, ý tưởng và sơ đồ phương pháp cấp phát TLB trong kỹ thuật phân trang.
7. Hãy trình bày ý tưởng và sơ đồ mô hình của bảng trang nghịch đảo.
8. Hãy trình bày phương pháp cấp phát kết hợp kỹ thuật phân trang và kỹ thuật phân đoạn.

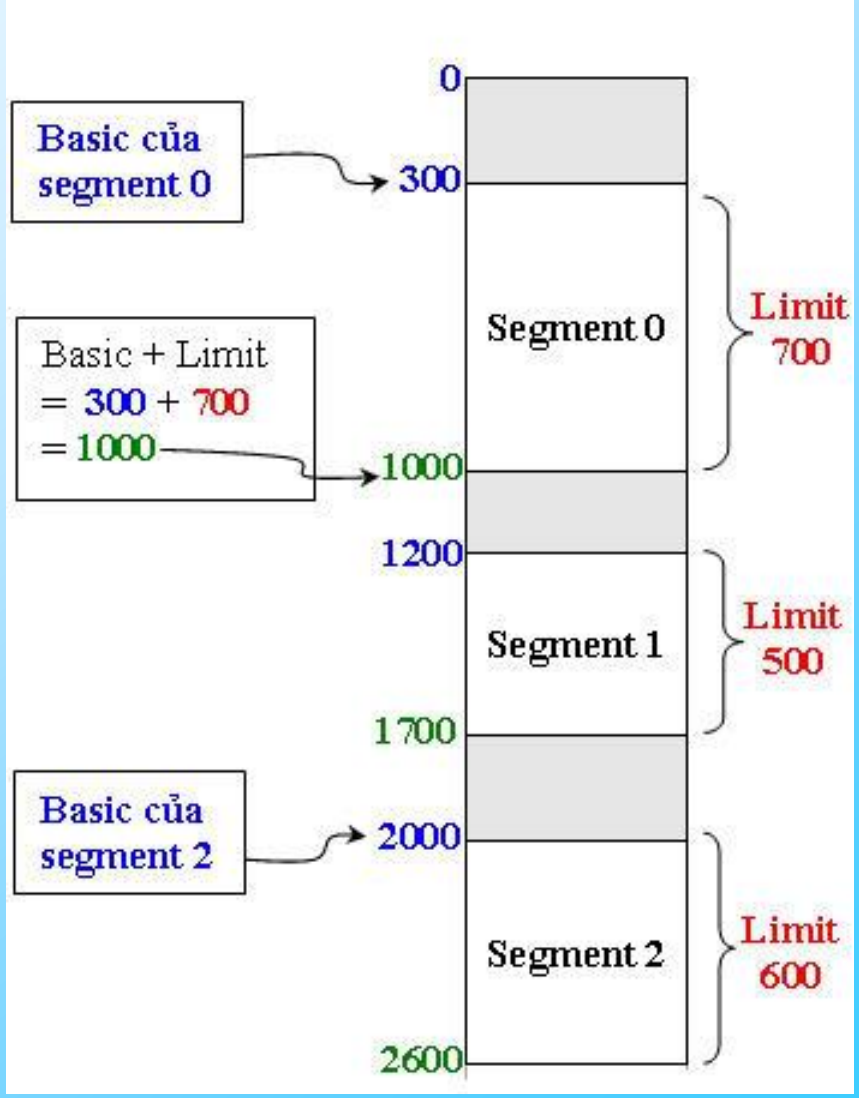
Bài tập

- Giả sử trong quá trình quản lý bộ nhớ ảo dạng phân đoạn, hệ điều hành duy trì bảng đoạn bên:
- Hãy tính địa chỉ vật lý cho mỗi địa chỉ logic:
  - (1,200)
  - (1,0)
  - (0,700)

Segment	Base	Limit
0	300	700
1	1200	500
2	2000	600

Bài giải

- Vùng bộ nhớ vật lý
- Segment 0:
  - Base: 300
  - Limit: 700
  - -> địa chỉ vật lý: 300->1000
- Segment 1:
  - Base: 1200
  - Limit: 500
  - -> địa chỉ vật lý: 1200->1700
- Segment 2:
  - Base: 2000
  - Limit: 600
  - -> địa chỉ vật lý: 2600



Bài giải

- (1,200):
  - Tính địa chỉ vật lý của segment 1, địa chỉ logic là 200
  - Vậy (1,200)=1200+200=1400 (hợp lệ vì <1700)
- Tương tự cho (1,0), (0,700)

Bài 8: Quản lý Bộ nhớ ảo

8.1 Mở đầu

8.2 Phân trang theo yêu cầu

8.3 Thay thế trang

8.4 Cấp phát khung trang

8.5 Trì trệ toàn hệ thống

8.1 – Mở đầu

- Bộ nhớ ảo là một kỹ thuật cho phép một không gian địa chỉ logic lớn có thể được ánh xạ vào một bộ nhớ vật lý nhỏ hơn.
- Bộ nhớ ảo có thể được triển khai bằng cách phân trang hoặc phân đoạn, hiện tại phân trang thông dụng hơn.
- Bộ nhớ ảo cho phép chạy những tiến trình cực lớn và cũng cho phép gia tăng mức độ đa chương được, tăng hiệu suất sử dụng CPU. Ngoài ra, nó giải phóng người lập trình ứng dụng khỏi việc lo lắng về khả năng sẵn có của bộ nhớ.

Bộ nhớ ảo: Ý tưởng

- Hai đặc trưng quan trọng của kiến trúc phân đoạn và phân trang:
  - Mọi sự truy xuất vùng nhớ của một tiến trình đều được chuyển đổi địa chỉ lúc thi hành (run-time) → có thể swap-in, swap-out.
  - Một tiến trình được phân ra thành một số phần (trang hoặc đoạn) và không nhất thiết phải nằm liên tục nhau

Bộ nhớ ảo: Ý tưởng (tt)

- Nếu hai tính chất trên được bảo đảm thì không nhất thiết tất cả các trang hoặc phân đoạn phải nằm trong bộ nhớ chính lúc thi hành.
- Ưu điểm:
  - ✓ Có nhiều tiến trình trong bộ nhớ hơn → giải thuật lập lịch sẽ tối ưu hơn → nâng cao mức độ đa chương.
  - ✓ Một tiến trình có thể lớn hơn kích thước của bộ nhớ chính.

Nguyên lý cục bộ

- Các thao tác truy cập vùng nhớ có khuynh hướng cụm lại (cluster).
  Sau một khoảng thời gian đủ dài, cụm này có thể sẽ thay đổi, nhưng trong một khoảng thời gian ngắn, bộ xử lý chủ yếu chỉ làm việc trên một số cụm nhất định.

Giải thích

Các câu lệnh cơ bản chủ yếu là tuần tự (thi hành từ trên xuống dưới). Câu lệnh không tuần tự là câu lệnh rẽ nhánh (câu lệnh điều kiện) thường chiếm tỉ lệ khá ít.

Trong một khoảng thời gian ngắn, các chỉ thị thông thường nằm trong một số hàm, thủ tục nhất định.

Hầu hết các câu lệnh lặp chứa một số ít các chỉ thị và lặp lại nhiều lần. Do đó trong suốt thời gian lặp, việc tính toán hầu như chỉ diễn ra trong một vùng nhỏ liên tục.

Khi truy cập vào một cấu trúc dữ liệu trước đó, thông thường các câu lệnh đặt liền nhau sẽ truy cập đến các thành phần khác nhau của cùng một cấu trúc dữ liệu.

Các vấn đề liên quan đến bộ nhớ ảo

- Cần có sự hỗ trợ phần cứng về kiến trúc phân trang và phân đoạn
  - Đã khảo sát
- Cần có thuật toán hiệu quả để quản lý việc chuyển đổi các trang, phân đoạn từ bộ nhớ chính vào bộ nhớ phụ và ngược lại
  - Nguyên lý cục bộ
  - Đĩa cứng hoạt động theo khối
  - Dự đoán được các trang và phân đoạn dựa vào lịch sử truy xuất vùng nhớ trước đó.

Quản lý việc chuyển đổi giữa vùng nhớ chính và vùng nhớ phụ

- Các chính sách cần xét:
  - Chính sách **nạp** (fetch policy): khi nào thì một trang được nạp vào bộ nhớ?
  - Chính sách **đặt** (placement policy): trang hoặc phân đoạn sẽ được đặt ở đâu trong bộ nhớ chính?
  - Chính sách **thay thế** (replacement policy): chọn trang nào đưa ra khỏi bộ nhớ phụ khi cần nạp một trang mới vào bộ nhớ chính?

Cài đặt bộ nhớ ảo

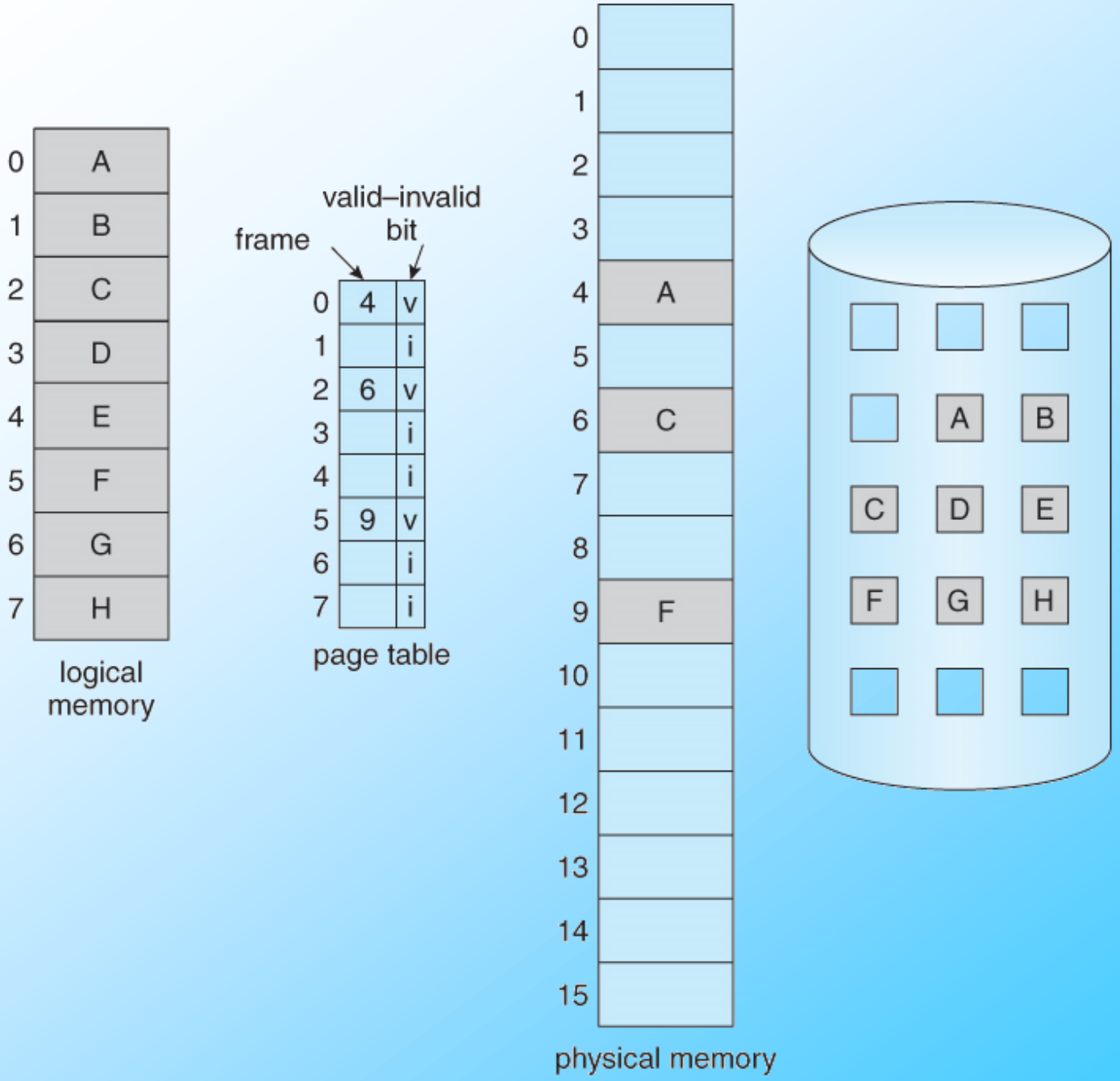
- Kỹ thuật phân trang theo yêu cầu (demand paging)
- Kỹ thuật phân đoạn theo yêu cầu (demand segmentation)
  - Khó vì kích thước không đồng nhất

8.2 - Phân trang theo yêu cầu

- Phân trang theo yêu cầu = Phân trang + swapping
- Tiến trình là một tập các trang thường trú trên bộ nhớ phụ.
- Một trang chỉ được nạp vào bộ nhớ chính khi có yêu cầu.
- Khi có yêu cầu về một trang nào đó, cần có cơ chế cho biết trang đó đang ở trên đó hoặc ở trong bộ nhớ
  - Sử dụng bit valid/invalid
  - Valid: có trong bộ nhớ chính
  - Invalid: trang không hợp lệ hoặc trang đang nằm trong bộ nhớ phụ

Cơ chế phần cứng

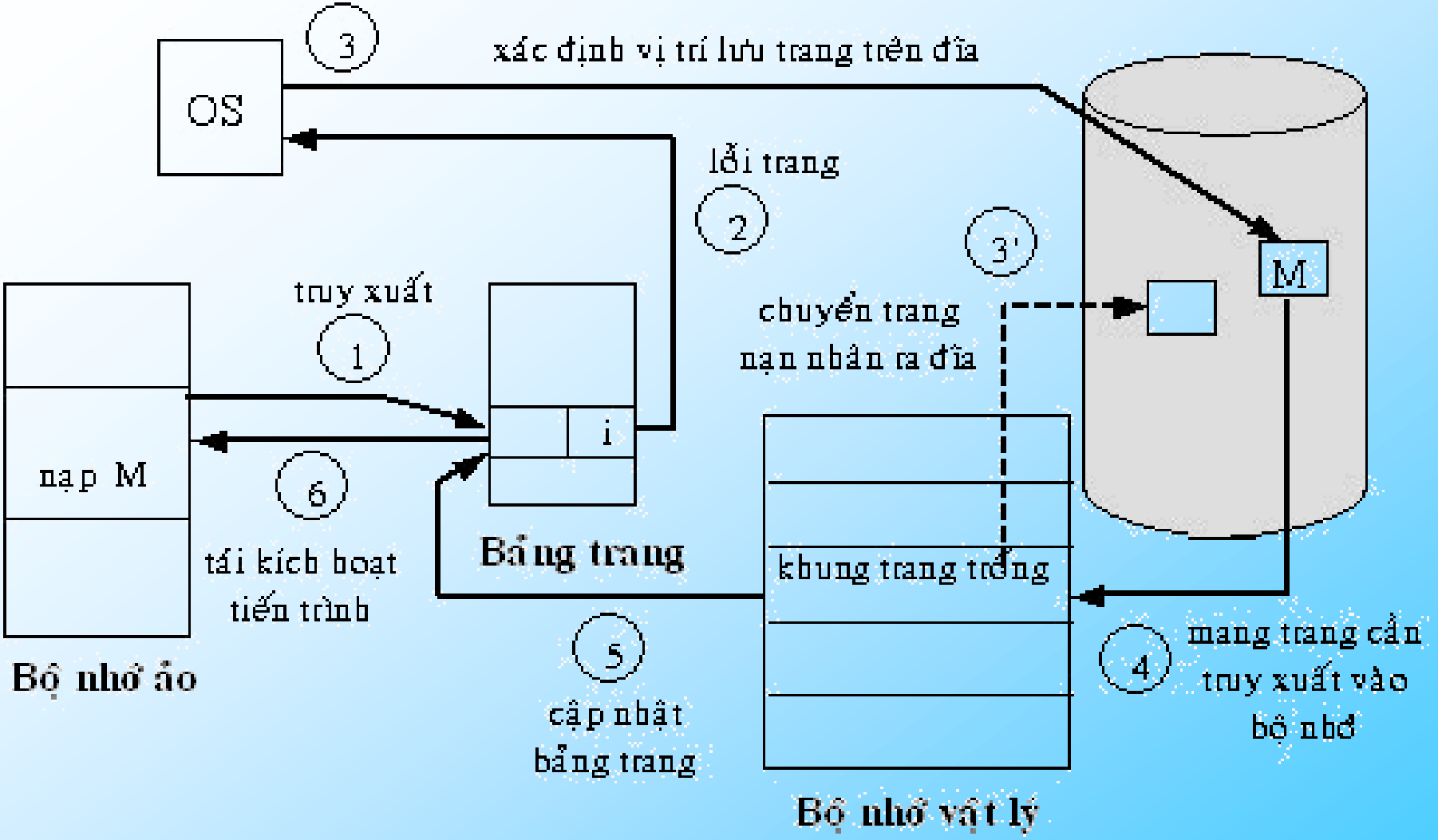
- Bảng trang
  - Phải phản ánh được một trang đang nằm trong bộ nhớ chính hay bộ nhớ phụ và tương ứng đang nằm ở vị trí nào (trong bộ nhớ chính hoặc bộ nhớ phụ)
- Bộ nhớ phụ
  - Dùng một không gian trên đĩa cứng thường gọi là không gian swapping.



Cơ chế hỗ trợ phần cứng cho kỹ thuật phân trang

Lỗi trang

- Truy xuất đến một trang được đánh dấu bất hợp lệ sẽ làm phát sinh một *lỗi trang* (*page fault*). Khi dò tìm trong bảng trang để lấy các thông tin cần thiết cho việc chuyển đổi địa chỉ, nếu nhận thấy trang đang được yêu cầu truy xuất là bất hợp lệ, cơ chế phần cứng sẽ phát sinh một ngắt để báo cho hệ điều hành.


3
xác định vị trí lưu trang trên đĩa
OS
lỗi trang
2
3'
truy xuất
1
chuyển trang
nạn nhân ra đĩa
M
nạp M
i
6
tái kích hoạt
tiến trình
Bảng trang
khung trang trống
5
cập nhật
bảng trang
4
mang trang cần
truy xuất vào
bộ nhớ
Bộ nhớ ảo
Bộ nhớ vật lý

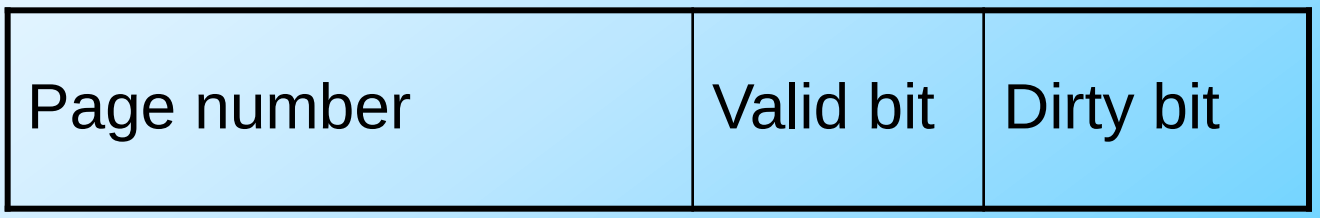
Quá trình xử lý một trang không có trong bộ nhớ chính (lỗi trang)

1. Kiểm tra trang được truy xuất có hợp lệ hay không?
2. Nếu truy xuất không hợp lệ → kết thúc.
   Ngược lại → bước 3.
3. Tìm vị trí chứa trang muốn truy xuất trên đĩa cứng.
4. Tìm một khung trang trống trên bộ nhớ chính
   a) Nếu tìm thấy → bước 5
   b) Nếu không tìm thấy khung trang trống, tìm một khung trang "nạn nhân" và chuyển nó ra bộ nhớ phụ, cập nhật bảng trang.
5. Chuyển trang muốn truy xuất từ bộ nhớ phụ vào bộ nhớ chính, cập nhật bảng trang, bảng khung trang.
6. Tái kích hoạt tiến trình tại chỉ thị truy xuất đến trang.

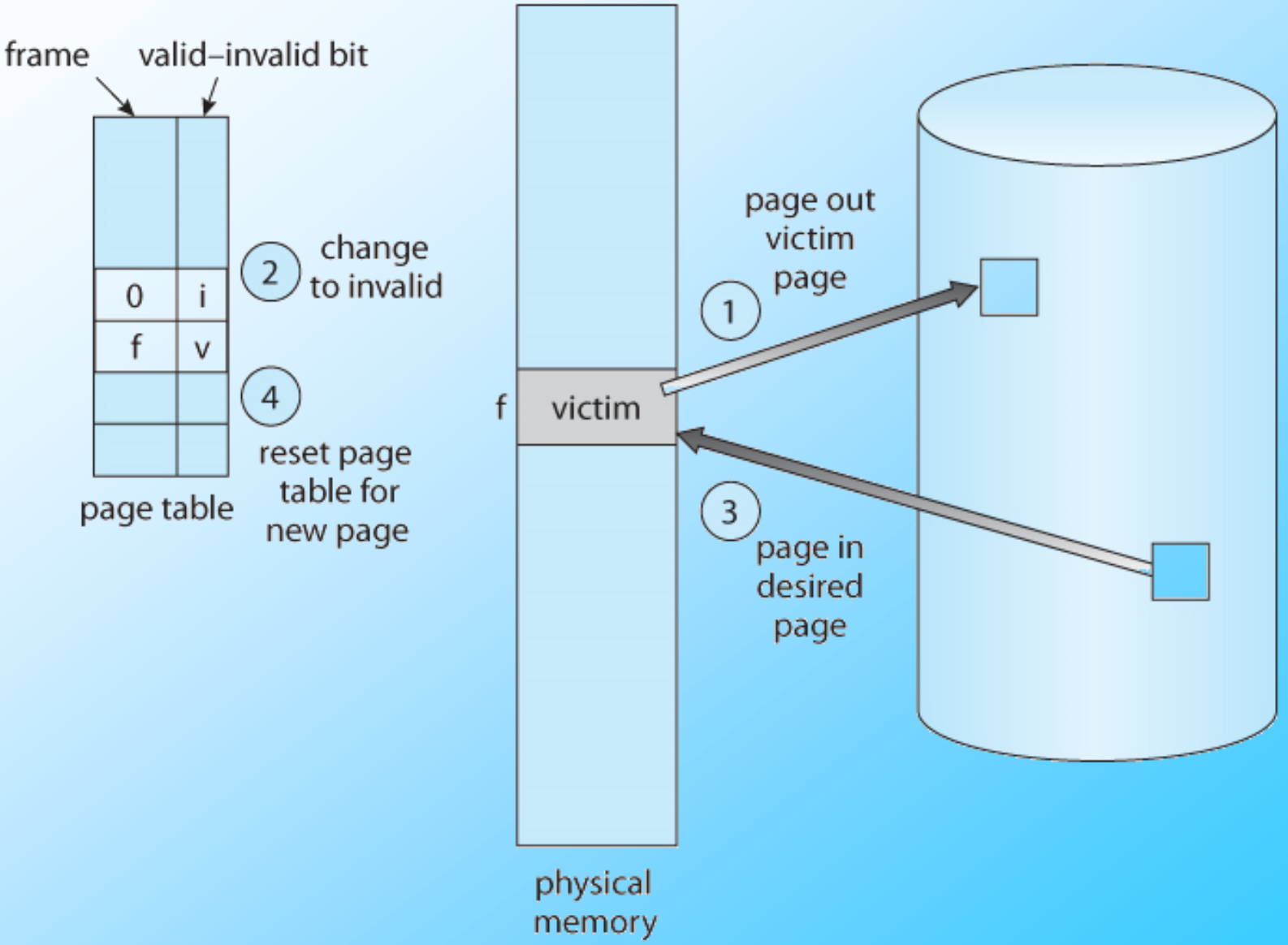
8.3 - Thay thế trang

- Là cơ chế thay thế một trang đang nằm trong bộ nhớ nhưng chưa cần sử dụng bằng một trang đang nằm trong đĩa (không gian swapping) đang được yêu cầu.
- Hai thao tác:
  - Chuyển trang từ bộ nhớ chính ra bộ nhớ phụ.
  - Mang trang từ bộ nhớ phụ vào vào nhớ chính
- Giảm số lần thao tác bằng bit cập nhập (dirty bit)
  - Bit cập nhật = 1: nội dung trang đã bị thay đổi → cần lưu lại trên đĩa
  - Bit cập nhật = 0: nội dung trang không bị thay đổi → không cần lưu lại trên đĩa

Một phần tử trong bảng trang



Các bước thay thế trang



Thuật toán thay thế trang

- Ý tưởng chính:
  - Chọn trang nạn nhân là trang mà sau khi thay thế sẽ gây ra ít lỗi trang nhất.
- Các thuật toán:
  - FIFO
  - Tối ưu (ít sử dụng nhất trong tương lai)
  - LRU (trang lâu nhất chưa được truy xuất)
  - Xấp xỉ LRU

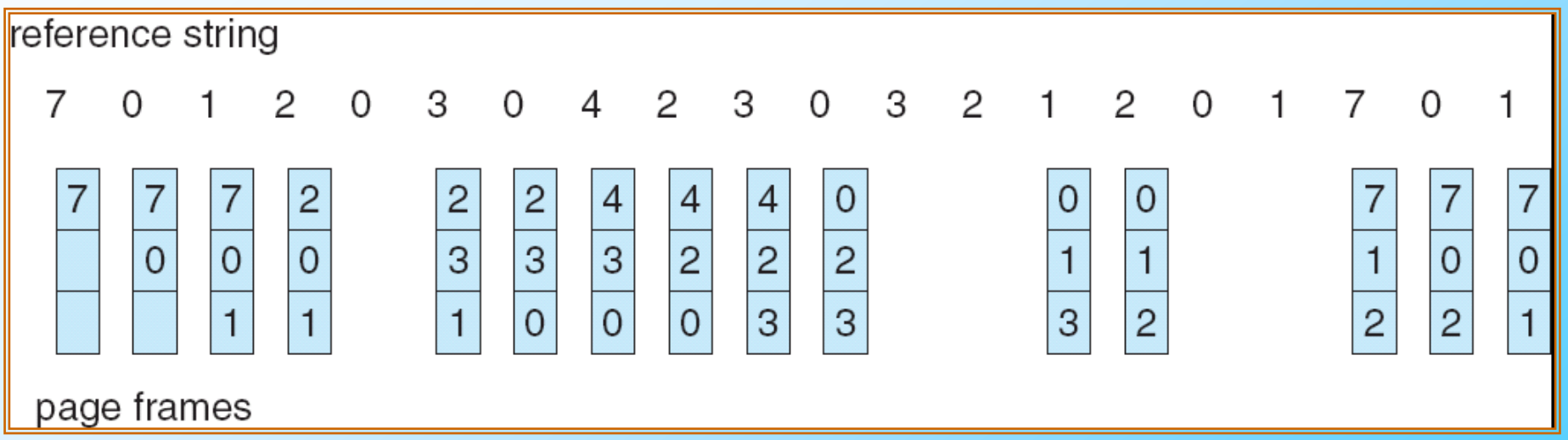
Thuật toán FIFO

- Ý tưởng:
  - Ghi nhận thời điểm một trang được đưa vào bộ nhớ
  - Thay thế trang ở trong bộ nhớ lâu nhất
- Có thể không cần ghi nhận thời điểm đưa một trang vào bộ nhớ. Sử dụng danh sách trang theo kiểu FIFO → trang thay thế luôn là trang đầu
- Dễ hiểu, dễ cài đặt, nhưng không lôgic trong trường hợp những trang đầu tiên được nạp vào thường những trang quan trọng, chứa dữ liệu truy xuất thường xuyên

→chuyển nó ra sẽ gây lỗi trang cho những lần truy xuất sau

- Nghịch lý Belady: *số lượng lỗi trang sẽ tăng lên nếu số lượng khung trang tăng lên*.

Thay thế trang FIFO



Ví dụ khác FIFO

Reference string: 1, 2, 3, 4, 1, 2, 5, 1, 2, 3, 4, 5

3 frames (3 trang có thể đồng thời trong bộ nhớ tại mỗi thời điểm)

1	1	4	5	
2	2	1	3	9 page faults
3	3	2	4	

Trang	1	2	3	4	1	2	5	1	2	3	4	5
Khung trang	1	1	1	4	4	4	5	5	5	5	5	5
		2	2	2	1	1	1	1	1	3	3	3
			3	3	3	2	2	2	2	2	4	4
Lỗi	*	*	*	*	*	*	*			*	*	

Ví dụ khác FIFO

4 frames

1	1	5	4	
2	2	1	5	10 page faults
3	3	2		
4	4	3		

Trang	1	2	3	4	1	2	5	1	2	3	4	5
Khung trang	1	1	1	1	1	1	5	5	5	5	4	4
		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
Lỗi	*	*	*	*			*	*	*	*	*	*

Belady's Anomaly: more frames ⇒ more page faults

Thuật toán tối ưu

- Ý tưởng:
  - Thay thế trang sẽ được lâu sử dụng nhất trong tương lai.
- Hoàn hảo về mặt ý tưởng nhưng không khả thi về mặt thực tế
  - Làm sao dự đoán được chuỗi các trang truy xuất trong tương lai.

Ví dụ

Trang	7	0	1	2	0	3	0	4	2	3	0	3	2	1	2	0	1	7	0	1
Khung trang	7	7	7	2	2	2	2	2	2	2	2	2	2	2	2	2	2	7	7	7
		0	0	0	0	0	0	4	4	4	0	0	0	0	0	0	0	0	0	0
			1	1	1	3	3	3	3	3	3	3	3	1	1	1	1	1	1	1
Lỗi	*	*	*	*		*		*			*			*				*		

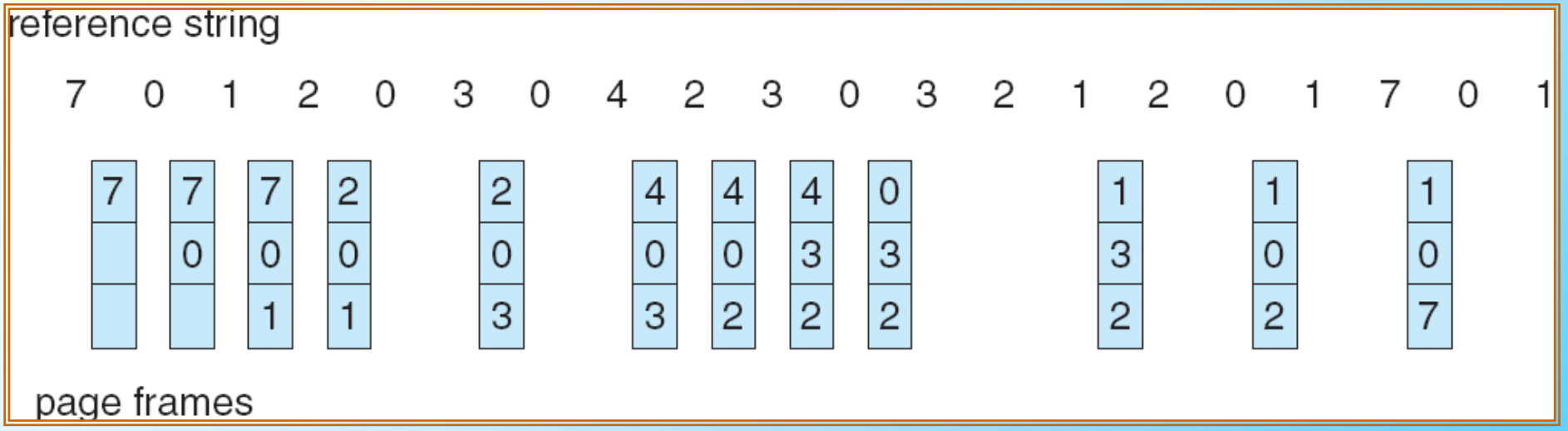
Thuật toán "Lâu nhất chưa sử dụng" (Least-recently-used LRU)

- Ý tưởng:
  - Ghi nhận thời điểm cuối cùng trang được truy cập
  - Thay thế trang chưa được truy cập lâu nhất
- Dùng quá khứ gần để dự đoán tương lai
  - FIFO: thời điểm nạp vào
  - Tối ưu: thời điểm sẽ truy cập

Least-recently-used LRU

- Các cách cài đặt:
  - Sử dụng bộ đếm
    - Mỗi phần tử trong bảng trang có một thành phần ghi nhận thời điểm truy xuất mới nhất.
    - CPU có một bộ đếm, tăng khi có một truy xuất đến bộ nhớ
    - Cập nhật thời điểm theo bộ đếm
    - Trang có thời điểm truy xuất nhỏ nhất sẽ bị thay thế
  - Sử dụng stack
    - Lưu các số hiệu trang
    - Khi một trang được truy xuất → chuyển số hiệu trang lên đầu stack
    - Thay thế trang có số hiệu ở đáy stack

Ví dụ LRU



Trang	7	0	1	2	0	3	0	4	2	3	0	3	2	1	2	0	1	7	0	1
Khung trang	7	7	7	2	2	2	2	4	4	4	0	0	0	1	1	1	1	1	1	1
		0	0	0	0	0	0	0	0	3	3	3	3	3	3	0	0	0	0	0
			1	1	1	3	3	3	2	2	2	2	2	2	2	2	2	7	7	7
Lỗi	*	*	*	*		*		*	*	*	*			*		*		*		

- LRU đòi hỏi phần cứng hỗ trợ khá nhiều
  - Biến bộ đếm
  - Stack
- Tìm các thuật toán xấp xỉ LRU

Thuật toán xấp xỉ LRU

- Có 3 thuật toán
  - Sử dụng nhiều bit tham khảo (reference bit)
  - Cơ hội thứ hai
  - Cơ hội thứ hai cải tiến

- Ý tưởng chính: bit tham khảo được thêm vào mỗi phần tử trong bảng trang
  - Ban đầu = 0
  - Có truy xuất → 1
  - Sau mỗi chu kỳ qui định trước, kiểm tra bit này và gán nó trở lại là 0.
  - Biết được trang nào đã được truy xuất gần đây nhưng không biết được thứ tự truy xuất.

Thuật toán nhiều bit tham khảo

- Ý tưởng:
  - 1 bit tham khảo chỉ biết được thông tin của 1 chu kỳ
  - Nhiều bit tham khảo sẽ biết được thông tin của nhiều chu kỳ.
- Sử dụng thêm 8 bit tham khảo cho mỗi phần tử trong bảng trang
- Sau một chu kỳ, một ngắt được phát sinh, HĐH sẽ đặt bit tham khảo của trang đó (0 hoặc 1) vào bit cao nhất trong 8 bit, loại bỏ bit cuối cùng (thấp nhất)
- 8 bit sẽ lưu trữ tình hình truy xuất đến trang trong 8 chu kỳ gần nhất
- 10001000 sẽ tốt hơn 01111111
- Nếu xem là số nguyên không dấu thì trang được thay thế là trang có số tương ứng nhỏ nhất.

Thuật toán cơ hội thứ hai

- Ý tưởng:
  - Sử dụng một một bit tham khảo duy nhất
  - Ý tưởng như FIFO có cải tiến
    - Nếu bit tham khảo = 0 □ thay thế trang
    - Ngược lại, cho trang này cơ hội thứ hai đặt bit tham khảo về 0, chọn trang FIFO kế tiếp. Trang được cho cơ hội thứ hai đặt vào cuối hàng đợi.
- Một trang đã được cho cơ hội thứ hai sẽ không bị thay thế trước khi các trang còn lại bị thay thế
- Có thể cài đặt bằng một xâu vòng (danh sách liên kết vòng).

Thuật toán cơ hội thứ hai nâng cao

- Ý tưởng:
  - Xét cặp bit: reference bit và dirty bit
  - (0,0): không truy xuất, không sửa đổi → trang tốt nhất để thay thế.
  - (0,1): không truy xuất, có sửa đổi → cần lưu lại trang thay thế.
  - (1,0): có truy xuất, chưa sửa đổi → có khả năng sẽ được sử dụng tiếp.
  - (1,1): có truy xuất, có sửa đổi → có khả năng được sử dụng tiếp và nếu thay thế cần lưu lại.
  - Lớp đầu tiên có độ ưu tiên thấp nhất và lớp cuối cùng có độ ưu tiên cao nhất.

8.4 - Cấp phát khung trang

- Trả lời câu hỏi:
  - Mỗi tiến trình sẽ được cấp phát bao nhiêu khung trang?
- Các hướng tiếp cận:
  - Cấp phát cố định:
    - Cấp phát công bằng
    - Cấp phát theo tỉ lệ
  - Cấp phát theo độ ưu tiên

Cấp phát cố định

- Mỗi tiến trình sẽ được cấp phát một số lượng khung trang cố định ngay từ đầu cho đến khi kết thúc thi hành.
- Có 2 hướng
  - Cấp phát công bằng
    - M khung trang, n tiến trình → mỗi tiến trình m/n
  - Cấp phát theo tỉ lệ
    - S_i: kích thước bộ nhớ ảo của tiến trình I
    - S = sum(S_i)
    - M khung trang
    - Tiến trình I sẽ có: (S_i/S)*M khung trang

Cấp phát theo độ ưu tiên

- Số khung trang dành cho mỗi tiến trình phụ thuộc vào độ ưu tiên của tiến trình tại thời điểm xác định.
- Nếu tiến trình p_i phát sinh lỗi trang, chọn một trong các khung trang của nó để thay thế hoặc một khung trang của các tiến trình có độ ưu tiên thấp hơn.

Thay thế toàn cục và Thay thế cục bộ

- Thay thế toàn cục
  - ✓ Trang "nạn nhân" có thể là bất cứ khung trang nào của hệ thống, không nhất thiết phải là khung trang của tiến trình đó.
- Thay thế cục bộ
  - ✓ Trang nạn nhân là một trong số khung trang của tiến trình đó.
- Có vẻ thay thế toàn cục sẽ linh hoạt hơn nhưng có thể gây ra hiệu ứng trì trệ hệ thống (thrashing)

8.5 – Trì trệ toàn hệ thống

- *Sự trì trệ* (thrashing) là hiện tượng tiến trình thường xuyên phát sinh lỗi trang và vì thế phải dùng rất nhiều thời gian sử dụng CPU để thực hiện việc thay thế trang → thời gian dành cho xử lý công việc còn rất ít →hệ thống gần như mất khả năng xử lý công việc.
- Tốc độ phát sinh lỗi trang tăng rất cao, không công việc nào có thể kết thúc vì tất cả tiến trình đều bận rộn với việc thay thế trang → tình trạng *trì trệ toàn bộ hệ thống*.
- Nguyên nhân là do tiến trình không có đủ các khung trang để chứa những trang cần thiết cho xử lý công việc

Giải pháp

- Để tránh tình trạng trì trệ toàn hệ thống mà vẫn duy trì được mức độ đa chương cao, cần phải có các giải pháp xác định và điều chỉnh mức độ cấp phát khung trang cho các tiến trình sao cho không thừa không thiếu.
- Hai trong số các giải pháp đó là mô hình tập làm việc và kiểm soát tần suất lỗi trang.

CÂU HỎI ÔN TẬP BÀI 8

1. Hãy trình bày khái niệm, ý tưởng và lợi ích của kỹ thuật bộ nhớ ảo.
2. Hãy trình bày khái niệm lỗi trang và các bước xử lý lỗi trang. Vẽ sơ đồ xử lý lỗi trang.
3. Công thức tính thời gian trung bình truy xuất cho một truy xuất trang. Vì sao phải giảm thiểu xác suất xảy ra lỗi trang.
4. Hãy trình bày giải thuật tối ưu cho việc thay thế trang. Vì sao giải thuật này không áp dụng được trong thực tế.
5. Hãy trình bày giải thuật LRU cho việc thay thế trang.

NGUỒN THAM KHẢO

1. Giáo trình Hệ điều hành, ĐH HUTECH, 2015.
2. Slide bài giảng Hệ điều hành của các trường Đại học trong nước và quốc tế.
3. https://www.cs.uic.edu/~jbell/CourseNotes/OperatingSystems/index.html

Q & A