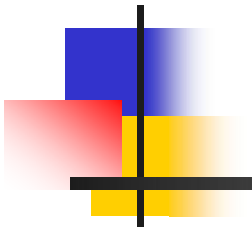
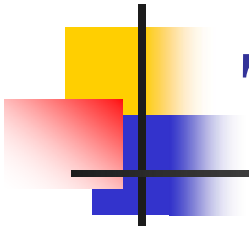


HỆ ĐIỀU HÀNH NÂNG CAO

Trường đại học Khoa học tự nhiên

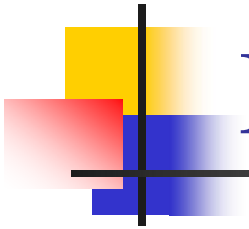
Khoa Công nghệ Thông tin

Trần Hạnh Nhi



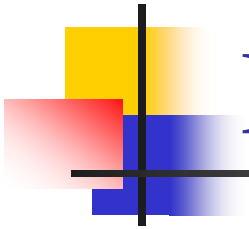
Tổ chức

- Phụ trách Lý thuyết :
  - **Trần Hạnh Nhi**
- Phụ trách thực hành:
  - **Phạm Nguyễn Anh Huy**
  - **Trần Anh Tuấn**
  - **Lê Thụy Anh**
  - **Đinh Bá Tiến**
- Trang web của môn học :



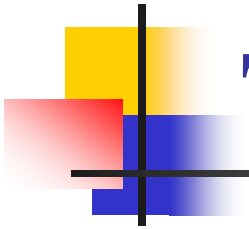
Mục tiêu

- Kết quả mong đợi về lý thuyết :
  - Hiểu được cách thức Hệ điều hành làm việc
  - Nắm được các nguyên lý thiết kế Hệ điều hành
  - Biết được một số cơ chế, chiến lược cơ bản để giải quyết các nhiệm vụ của Hệ điều hành
- Kết quả cần đạt được về thực hành
  - Vận dụng được các kiến thức lý thuyết để cài đặt giả lặp một số module của Hệ điều hành
  - Sử dụng được các cơ chế hỗ trợ của một Hệ điều hành cụ thể (Windows NT) để giải quyết các bài toán cơ bản.



Kiến thức yêu cầu

- Kiến trúc Máy tính
- Hệ điều hành cơ bản
- Lập trình C/C++

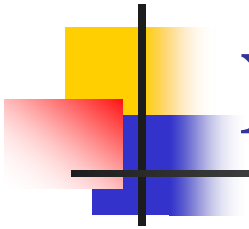


Tính điểm

- 70% Lý thuyết + 30% Thực hành
- Lý thuyết :
  - 1 bài thi cuối khoá (không tham khảo tài liệu)
  - Mỗi sinh viên làm bài độc lập
- Thực hành: 2 bài tập lớn
  - Thời hạn và cách thức nộp bài sẽ do giáo viên phụ trách thực hành qui định
  - Mỗi nhóm thực hành gồm 2 sinh viên
- Bắt buộc có nộp bài thực hành mới được thi lý thuyết

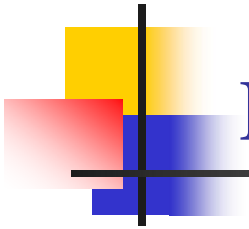
Tài liệu tham khảo

- *Trần Hạnh Nhi* : Giáo trình Hệ điều hành Nâng cao
- *A.Silberschatz & P/Galvin* : OS concepts (5e)
  - Slides :
- *W. Stallings* : Operating Systems
- *A.Tanenbaum et al* : OS Design and Implementation
  - Minix :
- *R.Finkel*:: An OS vade mecum
  - Book online :
- *Jeffrey Richter* : Advanced Windows
- *Tiến Huy- Đan Thư- Hạnh Nhi* : Kỹ thuật lập trình trên Windows NT



Nội dung

- Chương 1 : Tổ chức Hệ điều hành
- Chương 2 : Quản lý tiến trình
- Chương 3 : Liên lạc giữa các tiến trình
- Chương 4 : Quản lý bộ nhớ chính
- Chương 5 : An toàn hệ thống



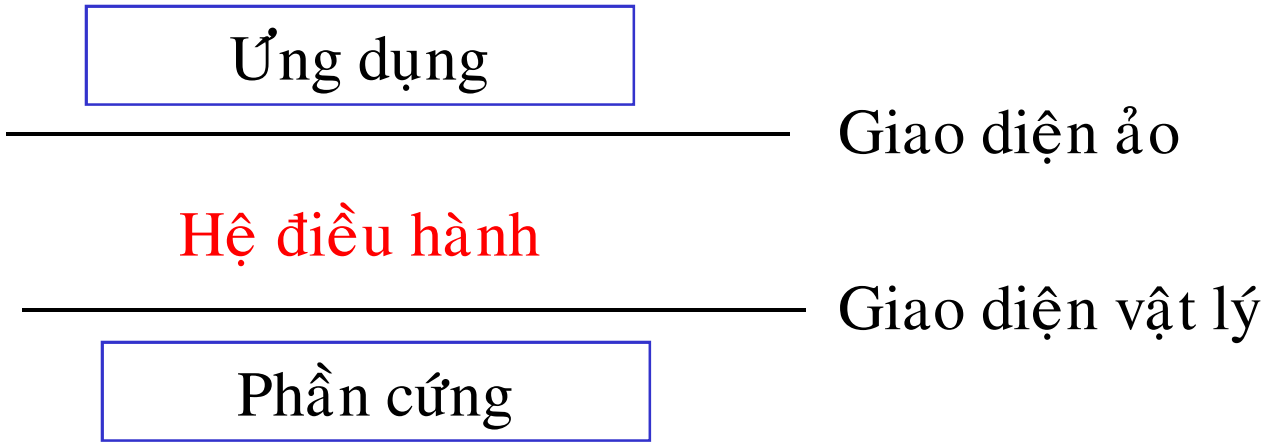
Bài giảng 1 : Giới thiệu

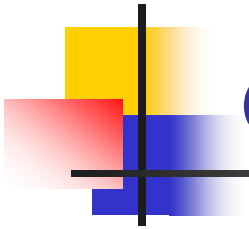
- Tại sao phải tìm hiểu về Hệ điều hành ?
- Hệ điều hành là gì ?
  - Vai trò trong hệ thống ?
  - Chức năng ?
  - Kiến trúc ?
- Các nguyên lý thiết kế Hệ điều hành

Tại sao cần tìm hiểu Hệ điều hành ?

- Để phá vỡ sự "bí ẩn" của hệ thống :
  - Tại sao máy tính có thể "biết" được nội dung đĩa ?
  - Tại sao có thể vừa soạn thảo, vừa nghe nhạc trên cùng 1 máy tính (có 1 CPU ?)
  - Tại sao 1 ứng dụng kích thước 1 M có thể hoạt động trên Windows mà bị báo "Not enough memory" trên DOS ?
- Để khai thác tốt hơn môi trường làm việc :
  - Lập trình trên môi trường đa nhiệm (multitask), đa xử lý(multiprocessing) với các mô hình multiprocess, multithreads..
  - Sử dụng bộ nhớ hiệu quả
  - sử dụng các cơ chế Thông tin liên lạc, an toàn & bảo mật…
- Vì là môn học bắt buộc ☺

Hệ điều hành, anh là ai ?





Chức năng của Hệ điều hành

- Quản trị tài nguyên (resource principle) :
  - Tài nguyên : CPU, Mem, IO; Files, ports, mailboxes…
  - Đối tượng sử dụng tài nguyên : Process, Thread
  - Nhiệm vụ : Cung cấp các giải thuật cấp phát, quản lý tài nguyên.cho các đối tượng hoạt động trong hệ thống
  - Mục tiêu : Cấp phát đầy đủ, công bằng R cho Ps; Sử dụng hiệu quả Rs, Nâng cao thông lượng Ps…
- Trừu tượng hoá hệ thống (beautification principle)
  - Nhiệm vụ : Cung cấp các giải thuật để che dấu chi tiết phần cứng, tạo 1 môi trường dễ làm việc hơn (hope) cho user
  - Mục tiêu : tạo môi trường an toàn, tạo sự trừu tượng hoá, độc lập thiết bị
  - Ví dụ : device driver

Các thành phần

Quản lý bộ nhớ phụ

Quản lý tiến trình

Quản lý nhập xuất

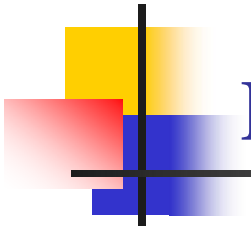
Hệ thống tập tin

Quản lý bộ nhớ chính

Hệ thống bảo vệ

Bộ thông dịch lệnh

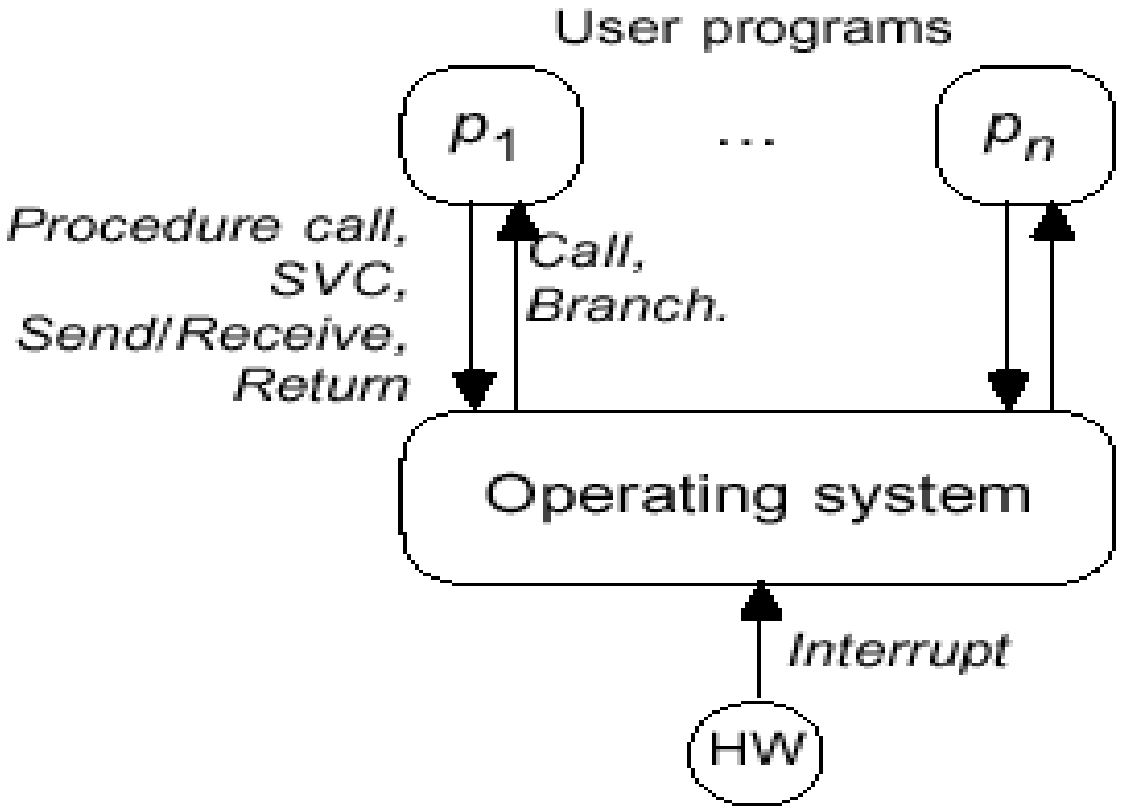
Giao tiếp mạng

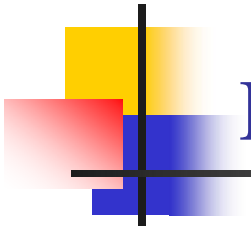


Kiến trúc Hệ điều hành

- Đơn giản (Monolithic)
- Hạt nhân (Kernel)
- Phân lớp (Layered)
- Máy ảo (Virtual Machine)
- Hướng đối tượng (OOOS)
- Exokernel

Monolithic



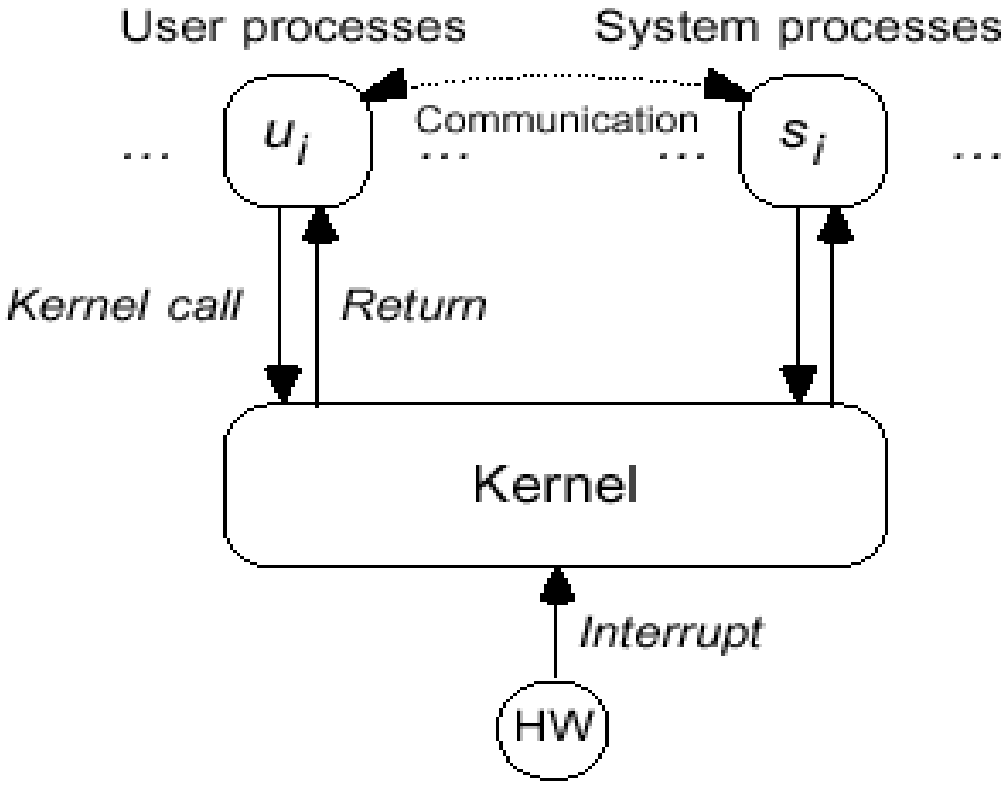


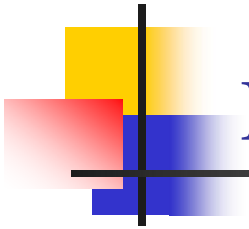
Monolithic

- OS = Thư viện tiện ích
- Có thể tổ chức thành nhiều module : CPU scheduling, Mem Management, Device management…nhưng chỉ có 1 trong những module này hoạt động tại một thời điểm
- Đơn nhiệm
- Quyền điều khiển được chuyển đổi thông qua lời gọi hàm

  Khi tầm vóc phát triển hệ thống trở nên thiếu tin cậy.
- Ví dụ : MS-DOS, Ultrix (mature Unix)

Kernel





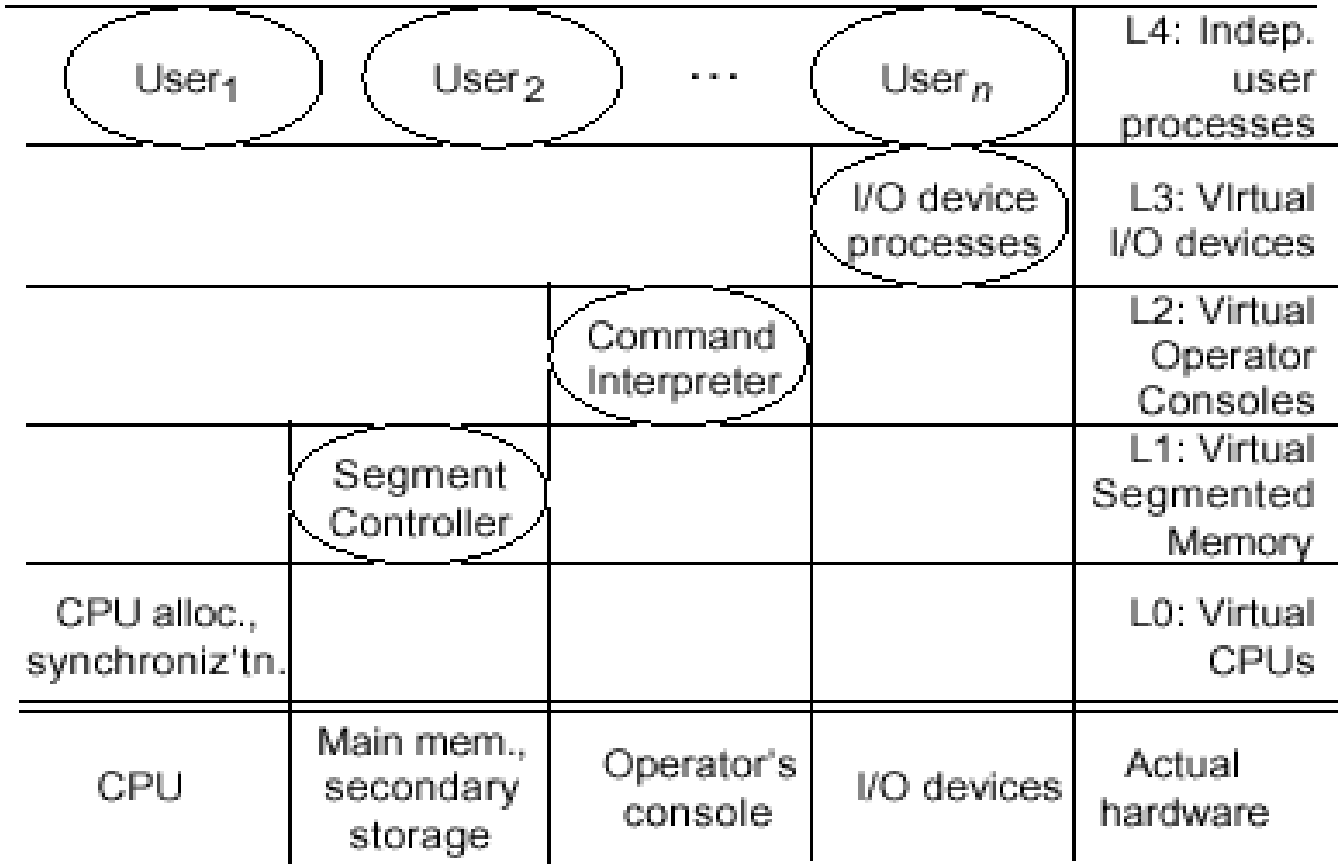
Kernel

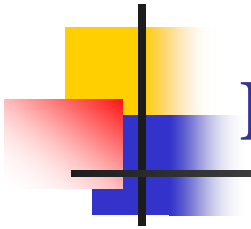
- OS = Kernel + System processes
- Kernel được bảo vệ
- Đa nhiệm
- Kernel chịu trách nhiệm phân chia thời gian sử dụng CPU, Giao tiếp giữa các tiến trình

  Chỉ có 2 mức kernel/non-kernel =>kernel lớn, thiếu tin cậy như trước

  Định nghĩa cứng các giao tiếp với ứng dụng trong kernel
- Ví dụ : Windows NT

Layered





Layered

- OS = các lớp trừu tượng hoá một tác vụ quản lý
- Lớp trên được sử dụng các hàm xử lý tài nguyên thuộc tác vụ do lớp dưới cung cấp

  Khó xác định được các lớp xử lý rạch ròi, thứ tự lớp ?

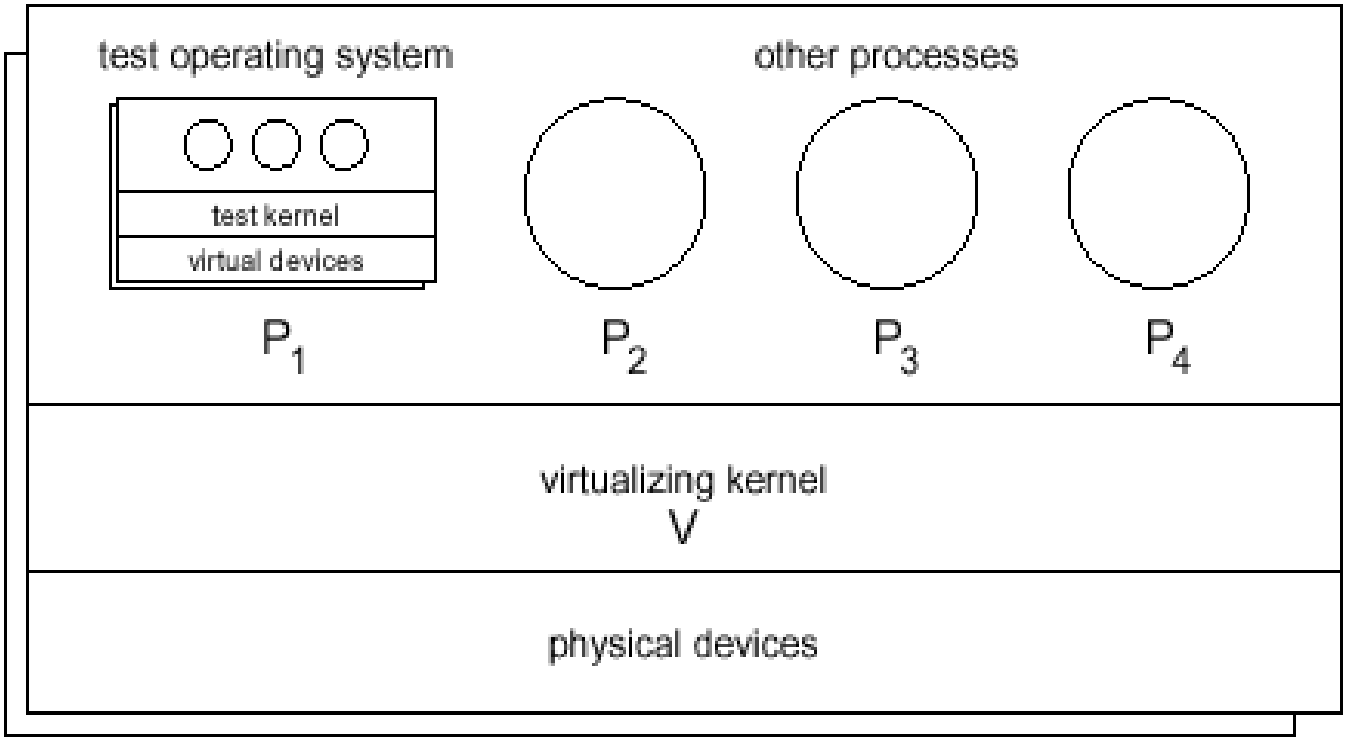
  Tạo tiến trình -> PM gọi MM

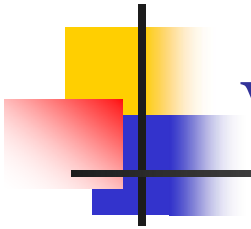
  Bộ nhớ đầy -> MM gọi PM

  Xếp lớp theo hàm xử lý , thay vì tác vụ

  Seg management- P scheduling- Seg creation- P creation
- Ví dụ : THE , MULTICS

Virtual Machine





Virtual Machine

- OS = Virtualizing kernel + virtual machines
- Virtual machine = physical hardware
- Virtualizing kernel tạo ra nhiều VM trên 1 máy tính.
- Process interface = hardware interface

  Ưu điểm :

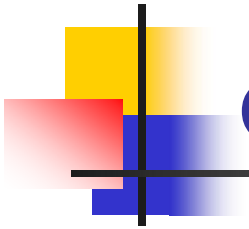
  Môi trường thuận lợi cho sự tương thích (compatibility)

  Tăng tính an toàn hệ thống do cung cấp các VM độc lập.

  Dể phát triển các HDH đơn nhiệm cho mỗi VM

  Khuyết điểm:

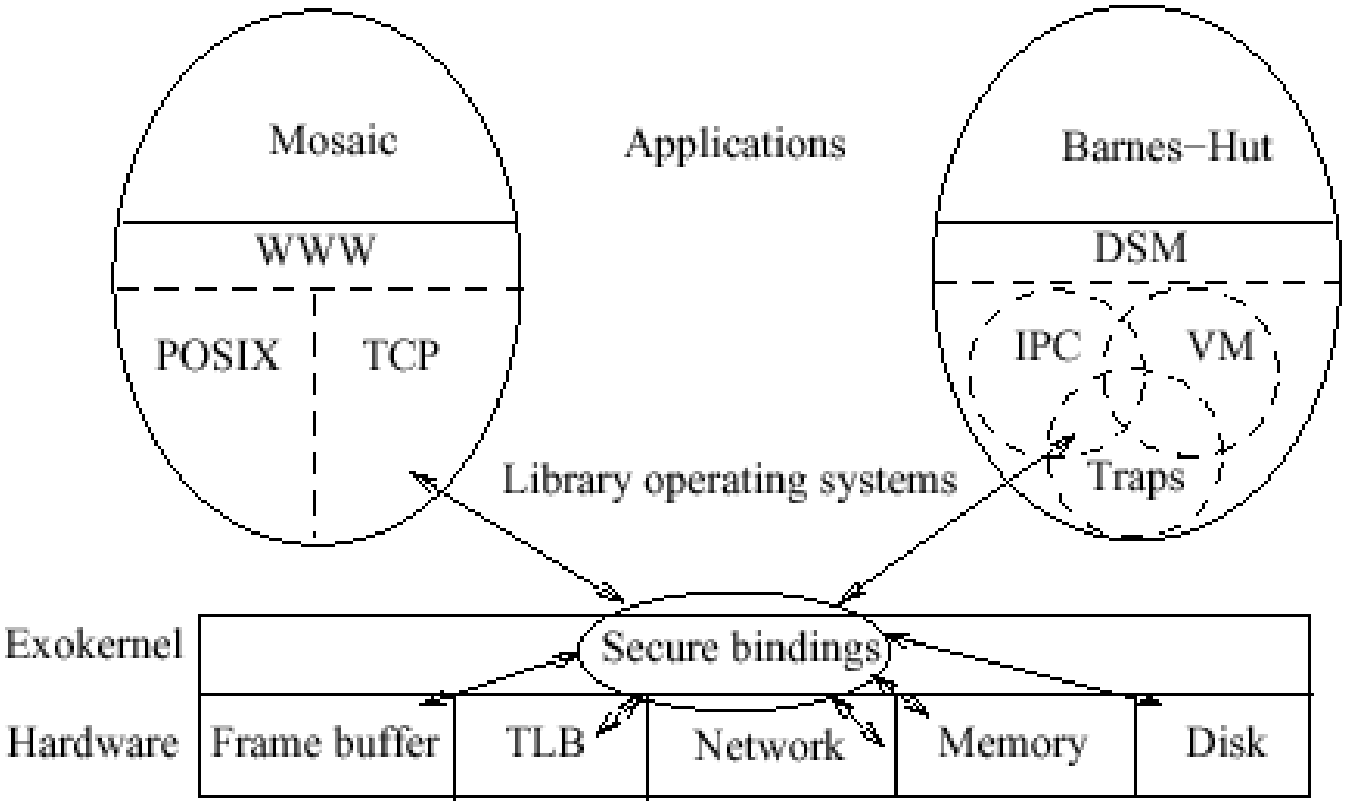
  Phức tạp cho việc giả lặp (transput, add translation…)
- Ví dụ : CMS(conversational Monitor System) trên VM/370 (hỗ trợ hardware)

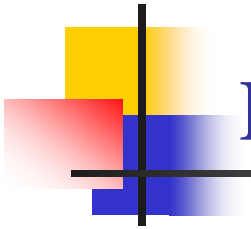


OOOS

- OS = tập các đối tượng
- Tiến trình, tập tin, hàm, khối nhớ…
- Một hàm xử lý (kernel/non-kernel mode) thao tác trên một tập các đối tượng.
- Che dấu thông tin
- Ví dụ :CAP, StarOS, iMAX432

Exokernel



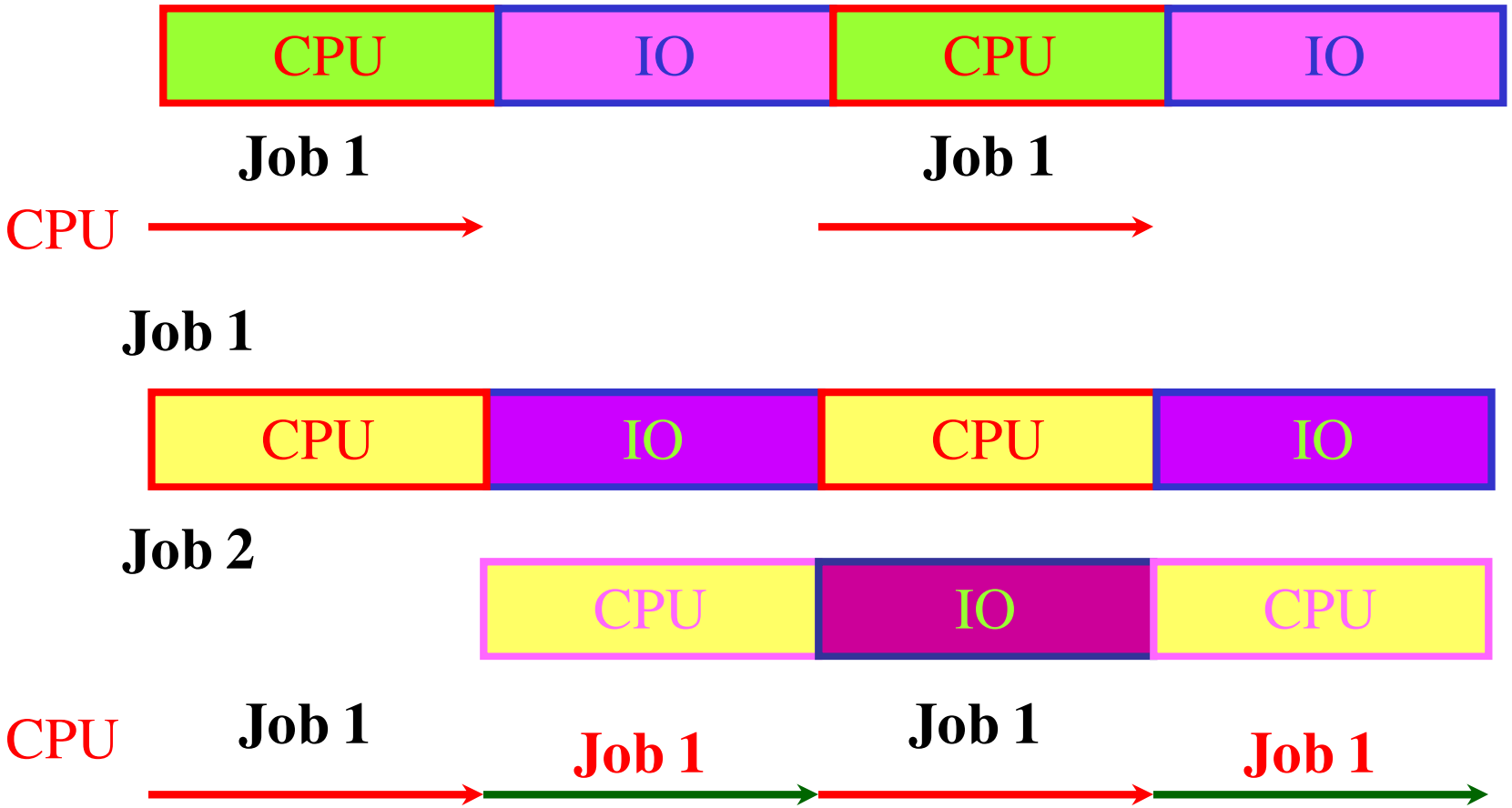


Exokernel

- Hướng đến một HDH linh đông trong giao tiếp với ứng dụng, cho phép ứng dụng chuyên biệt hoá hệ điều hành theo nhu cầu đặc thù một cách dễ dàng
- OS = Exokernel + Library OS
- Ứng dụng có thể phát triển các mô hình tổ chức VM, IPC theo nhu cầu riêng
- Ví dụ : ý tưởng của project do Dawson R Engler et al phát triển tại MIT

Bài 2 : CÁC MÔ HÌNH XỬ LÝ ĐỒNG HÀNH

Xử lý đồng hành, để tăng hiệu suất sử dụng CPU



Xử lý đồng hành, để tăng tốc độ xử lý

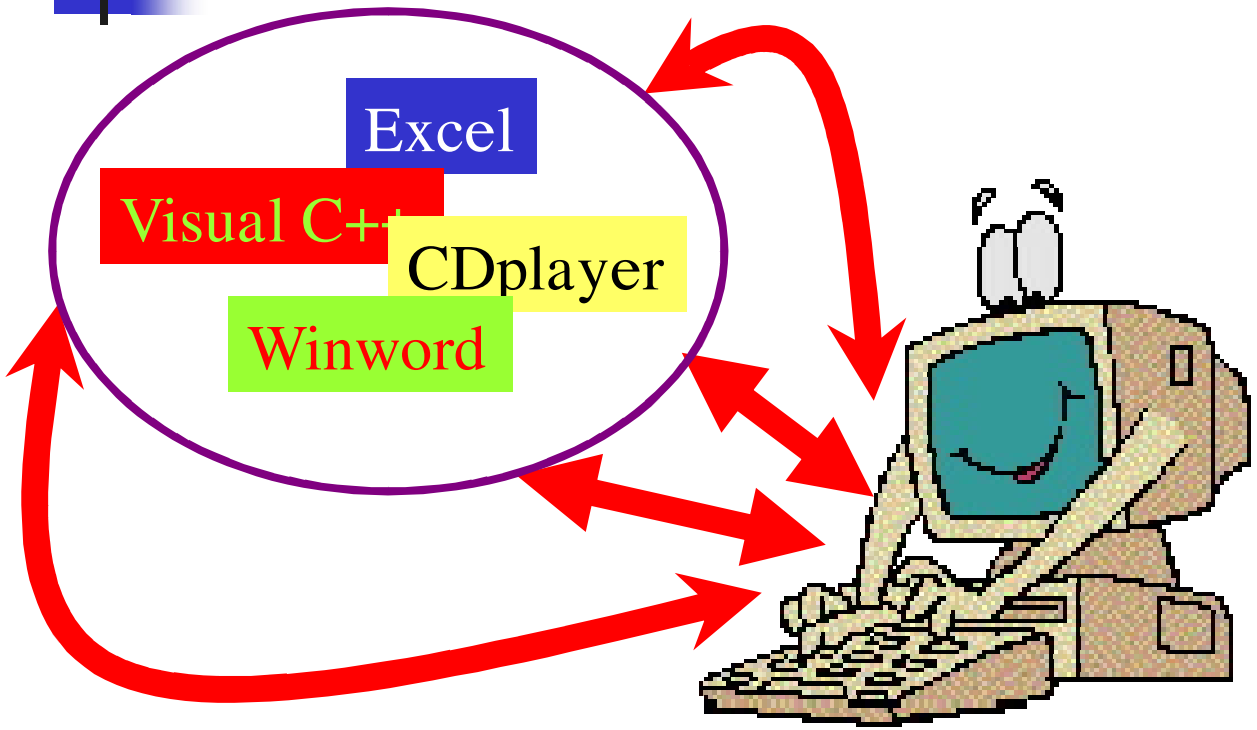
- **Job : kq = a*b + c*d;**
- **Xử lý tuần tự :**

```
kq1 = a * b;
kq2 = c* d;
kq = kq1 + kq2;
```

- **Xử lý đồng hành :**

kq1 = a*b;	kq2 = c*d;
kq = kq1 + kq2;	

Xử lý đồng hành, những khó khăn ?



HĐH : " Giải quyết nhiều công việc đồng thời, đâu có dễ !

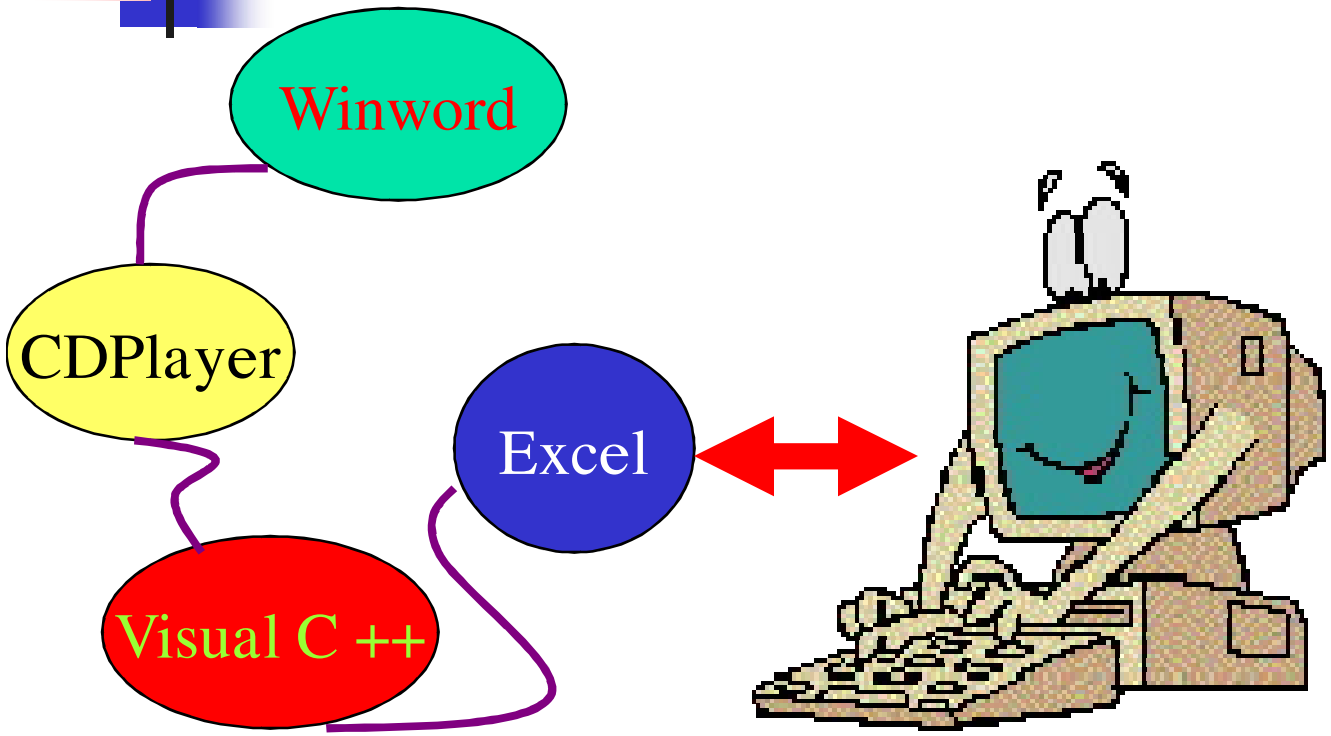
- **Tài nguyên giới hạn, ứng dụng "vô hạn"**
- **Nhiều hoạt động đan xen**

??? Phân chia tài nguyên ?

??? Chia sẻ tài nguyên ?

??? Bảo vệ?

Giải pháp

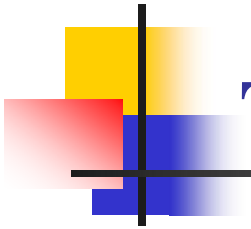


HĐH : " Ai cũng có phần khi đến lượt mà ! "

-"Chia để trị", cô lập các hoạt động.

- Mỗi thời điểm chỉ giải quyết 1 yêu cầu.

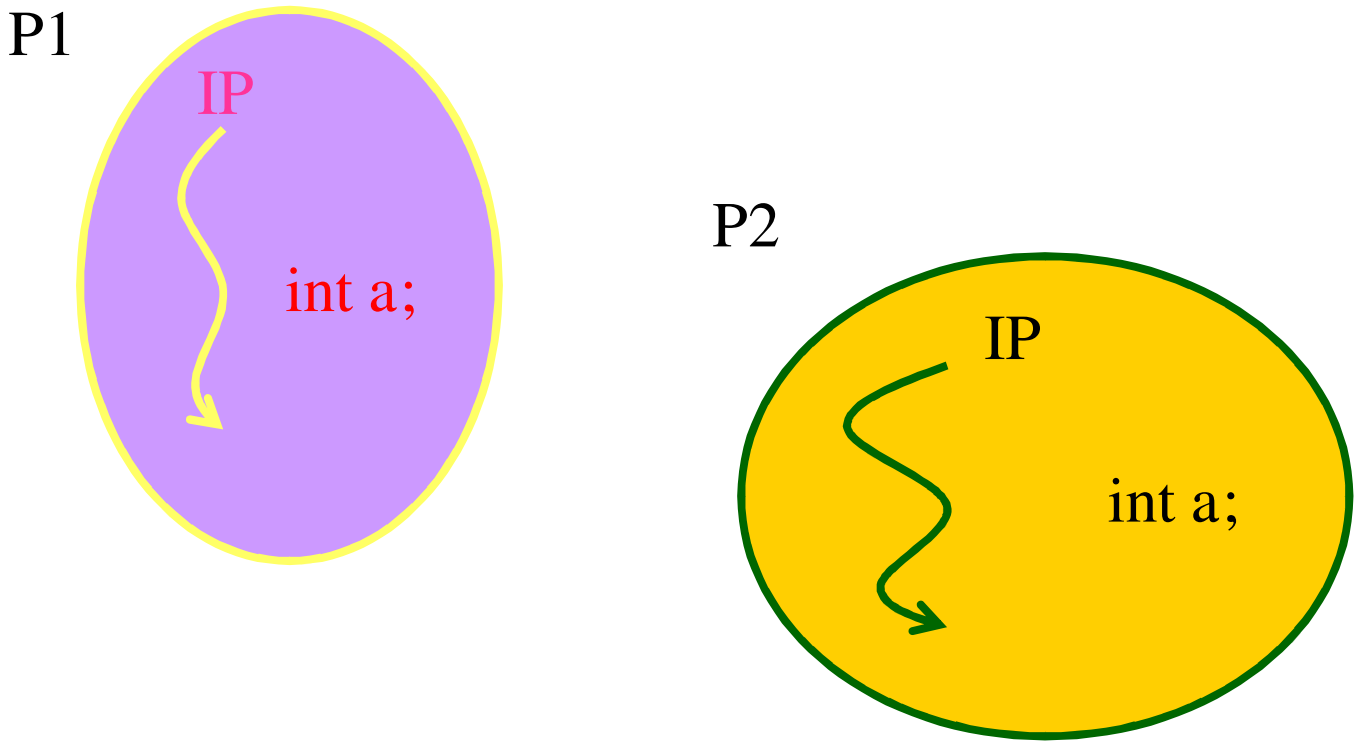
- Ảo hoá tài nguyên : biến ít thành nhiều



Thuật ngữ

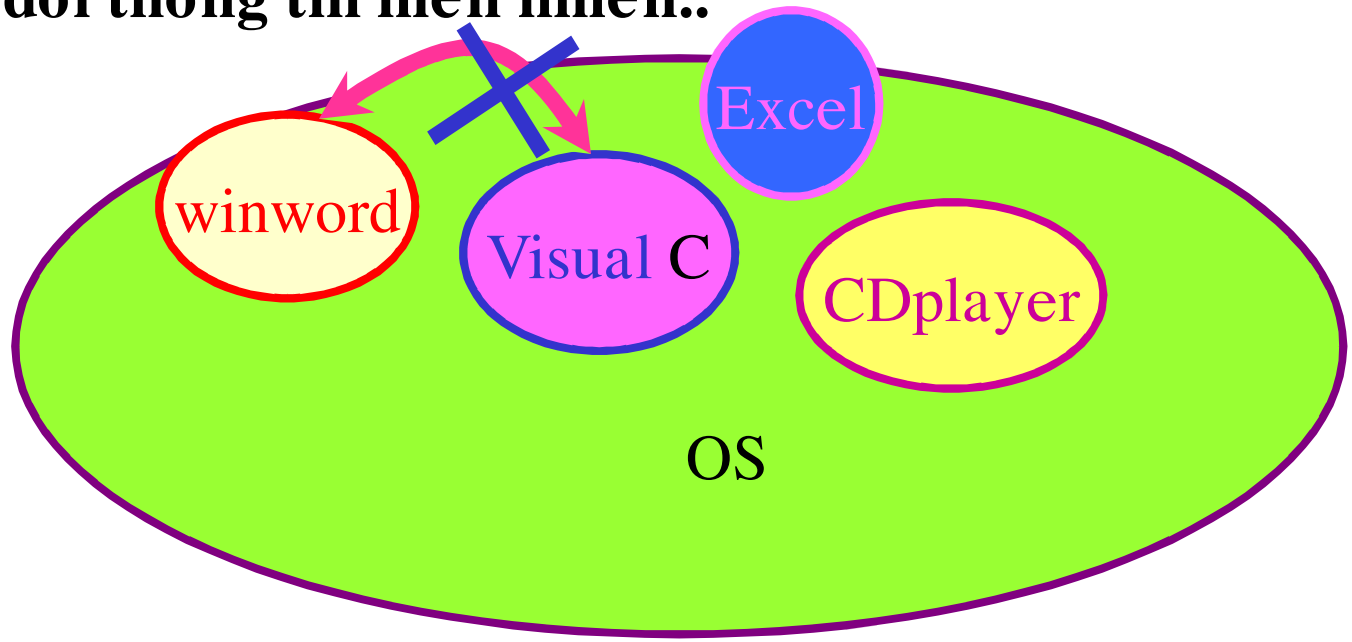
- **Concurrency** (đồng hành): mô hình xử lý nhiều tác vụ đồng thời.
- **Multitasking** (đa nhiệm) : cho phép nhiều tác vụ/ công việc được xử lý đồng thời
- **Multiprogramming** (đa chương) : cho phép nhiều chương trình được thực hiện đồng thời (trên 1 CPU)
- **Multiprocessing** (đa xử lý): nhiều bộ xử lý làm việc đồng thời

Khái niệm tiến trình



Mô hình đa tiến trình (MultiProcesses)

- **Hệ thống là một tập các tiến trình hoạt động đồng thời**
- **Các tiến trình độc lập với nhau => không có sự trao đổi thông tin hiển nhiên..**



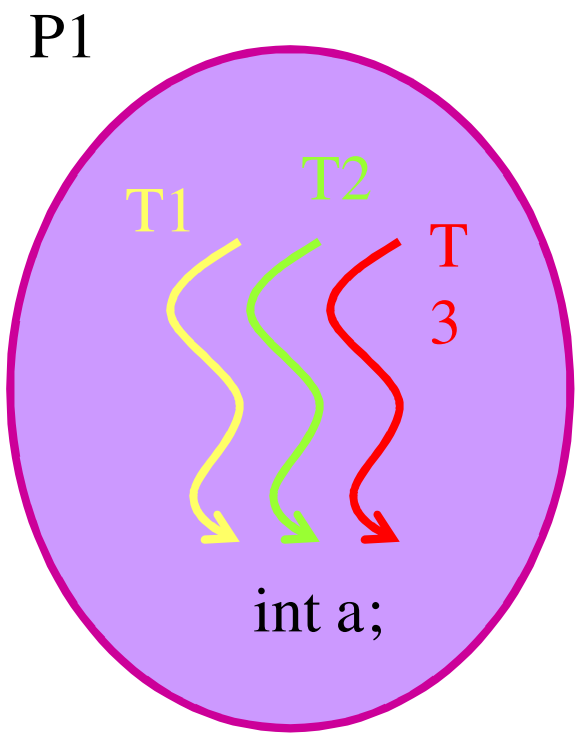
Mô hình đa tiểu trình (MultiThreads)

- **Muốn nhiều dòng xử lý đồng thời cùng chia sẻ tài nguyên (server, OS, các chương trình tính toán song song)**

☞ **TIỂU TRÌNH (THREAD)**

Khác biệt giữa Tiểu trình & Tiến trình

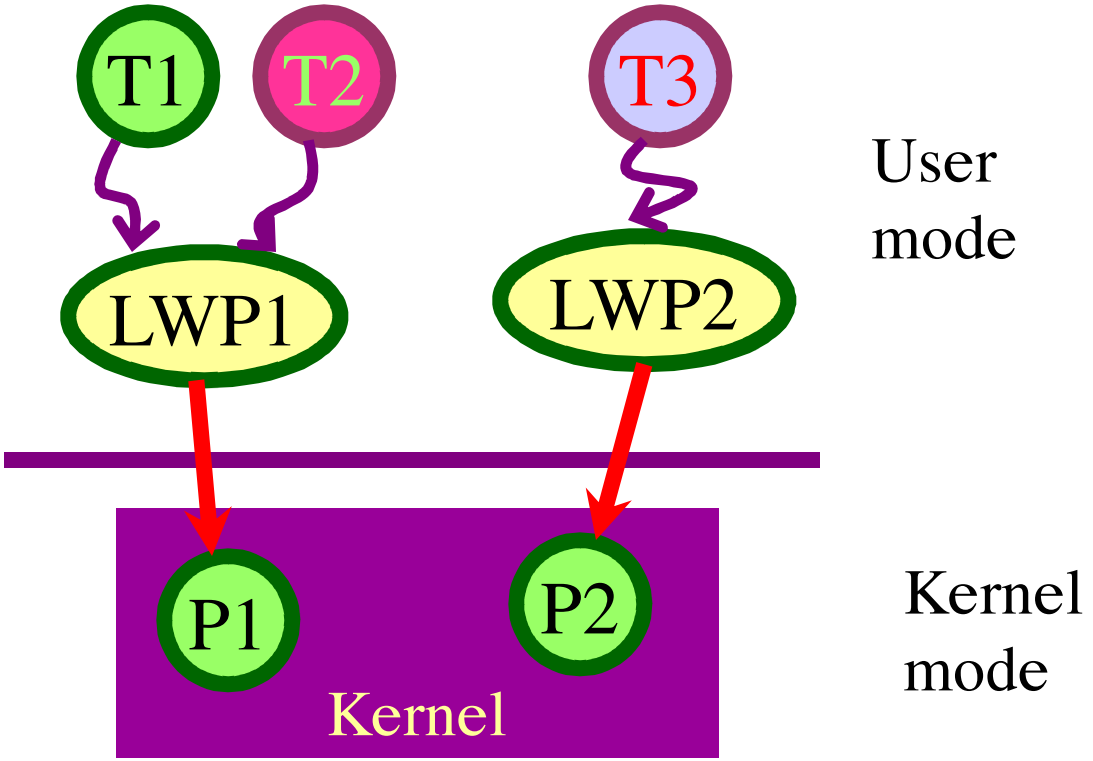
- Tiểu trình : 1 dòng xử lý
- Tiến trình :
  - 1 không gian địa chỉ
  - 1 hoặc nhiều tiểu trình
- Các tiến trình là độc lập
- Các tiểu trình trong cùng 1 tiến trình không có sự bảo vệ lẫn nhau (cần thiết ?).



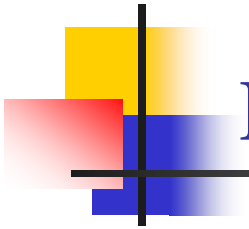
Tiểu trình hạt nhân (Kernel thread)

Khái niệm tiểu trình được xây dựng bên trong hạt nhân

Tiểu trình người dùng (User thread)



Khái niệm tiểu trình được hỗ trợ bởi một thư viện hoạt động trong user mode

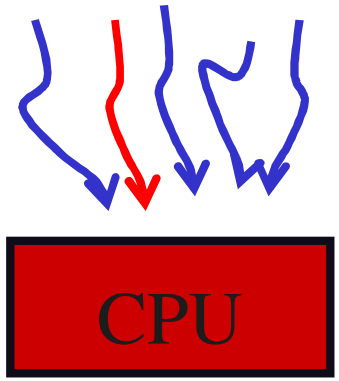


Bài 3 : QUẢN LÝ TIẾN TRÌNH

- Phân chia CPU cho các tiến trình ?
  - Tiếp cận
  - Mục tiêu ?
  - Tổ chức ?
  - Chiến lược ?
- Trạng thái tiến trình ?
- Lưu trữ thông tin tiến trình ?
- Các thao tác trên tiến trình ?
- Bảo vệ tiến trình ?
- Trao đổi thông tin giữa các tiến trình ?

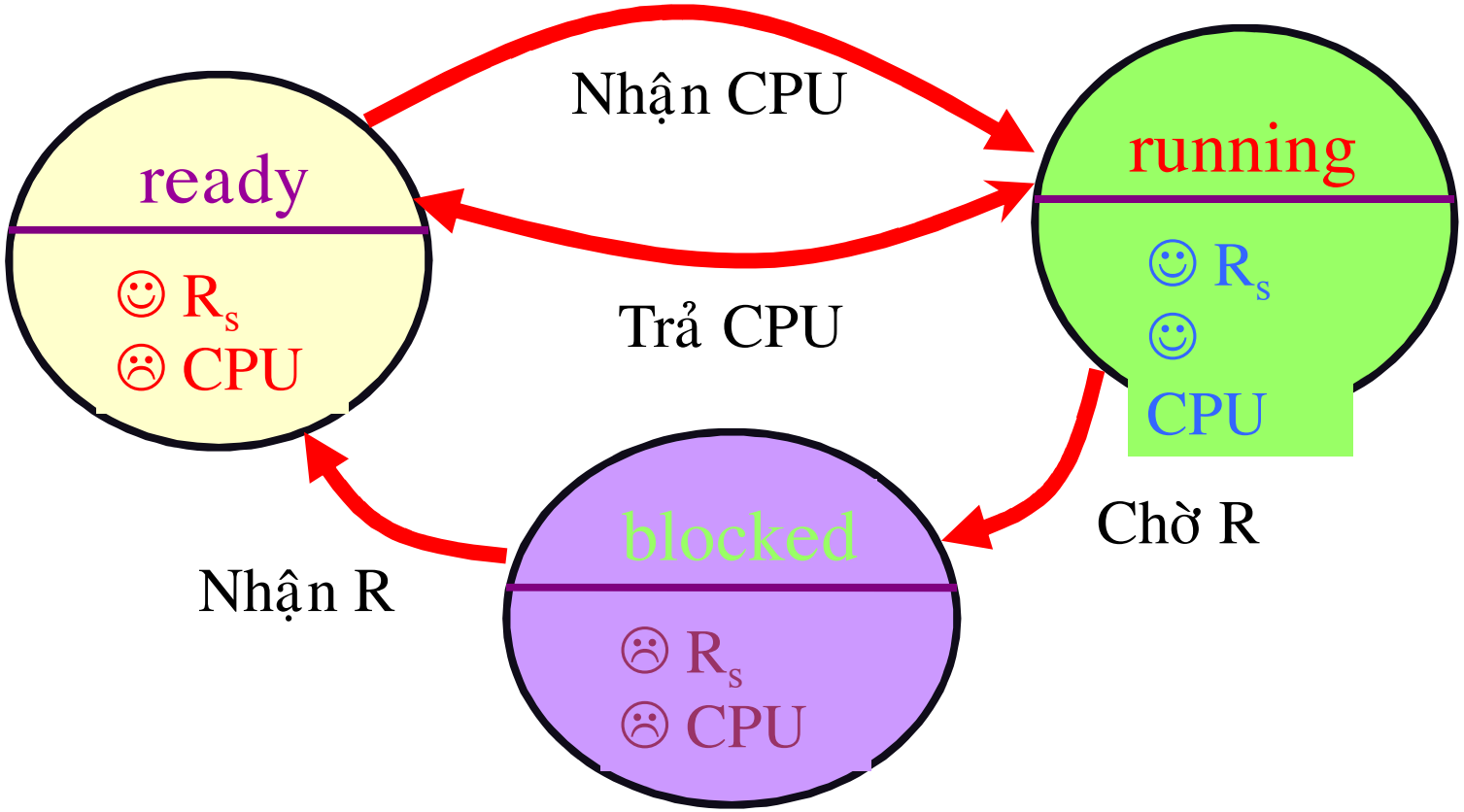
Phân chia CPU ?

- **1 CPU vật lý** : làm thế nào để tạo ảo giác mỗi tiến trình sở hữu CPU riêng của mình ?
- **Dispatcher** luân chuyển CPU giữa các tiến trình:
  - Ngữ cảnh xử lý riêng biệt cho mỗi tiến trình (PCB)
  - Dispatching loop :



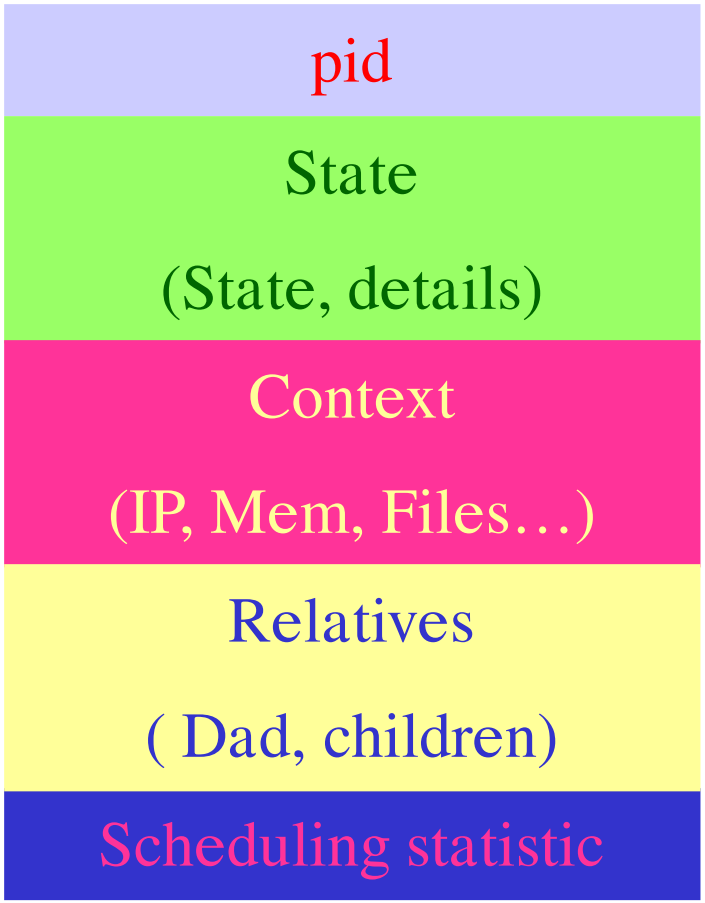

```
while(1)
{
   interrupt P_cur
   save state P_cur
   Scheduler gets P_next
   load state P_next
   jump to it
}
```

Trạng thái tiến trình ?



Khối quản lý tiến trình trong mô hình multiprocesses

Process control Block
PCB



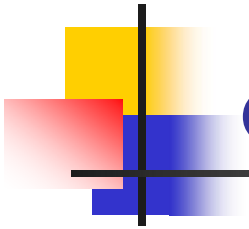
PCB và TCB trong mô hình multithreads

PCB

pid
Threads list
Context (Mem, global ressources…)
Relatives (Dad, children)
Scheduling statistic

Thread Control Block
TCB

tid
State (State, details)
Context (IP, local stack…)



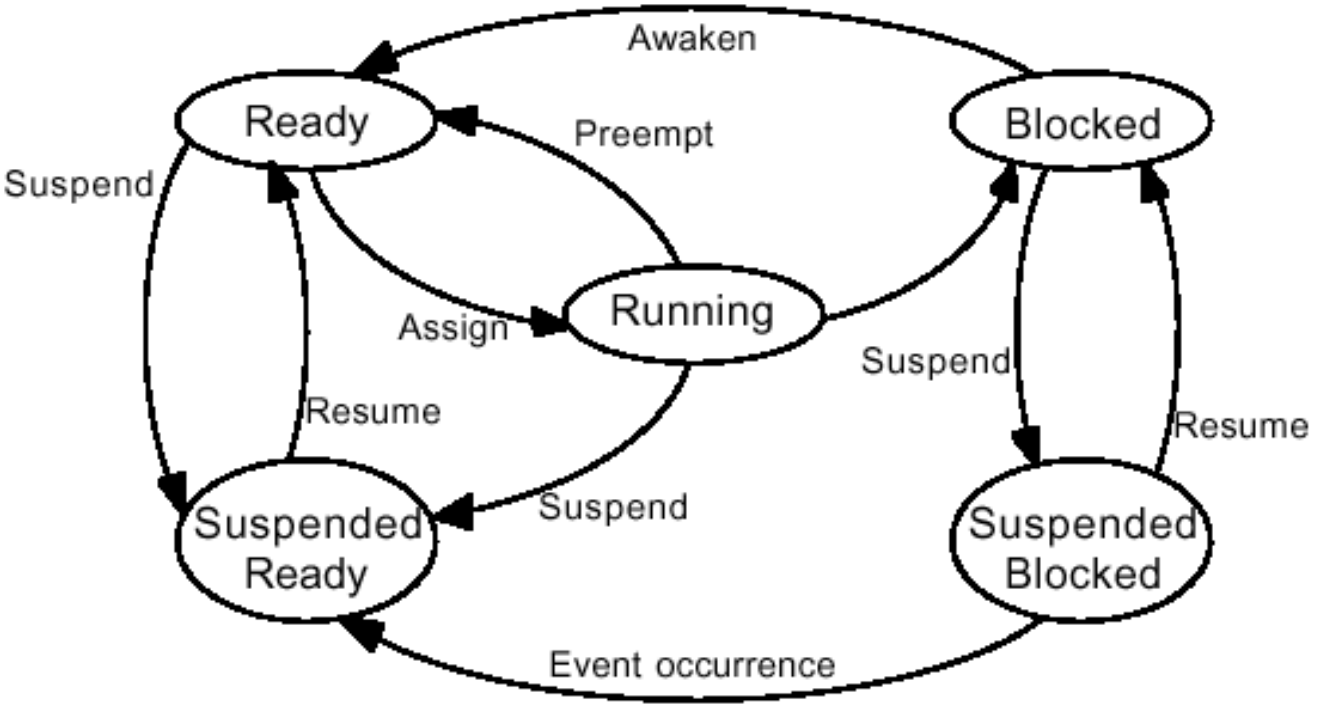
Các thao tác trên tiến trình

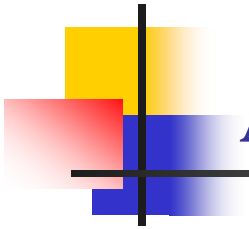
- **Tạo lập tiến trình :**
  - **Cấp phát tài nguyên cho tiến trình con ?**
  - **Hoạt động của cha và con độc lập**
- **Kết thúc tiến trình :**
  - **Thu hồi tài nguyên ?**
  - **Ép buộc kết thúc ?**
- **Thay đổi trạng thái tiến trình :**

  Assign(), Block(), Awake(), Resume(), Suspend()

Trạng thái tiến trình ?

- Có nhu cầu Suspend &Resume :
  - Hệ thống quá tải
  - Kiểm soát hoạt động của tiến trình con

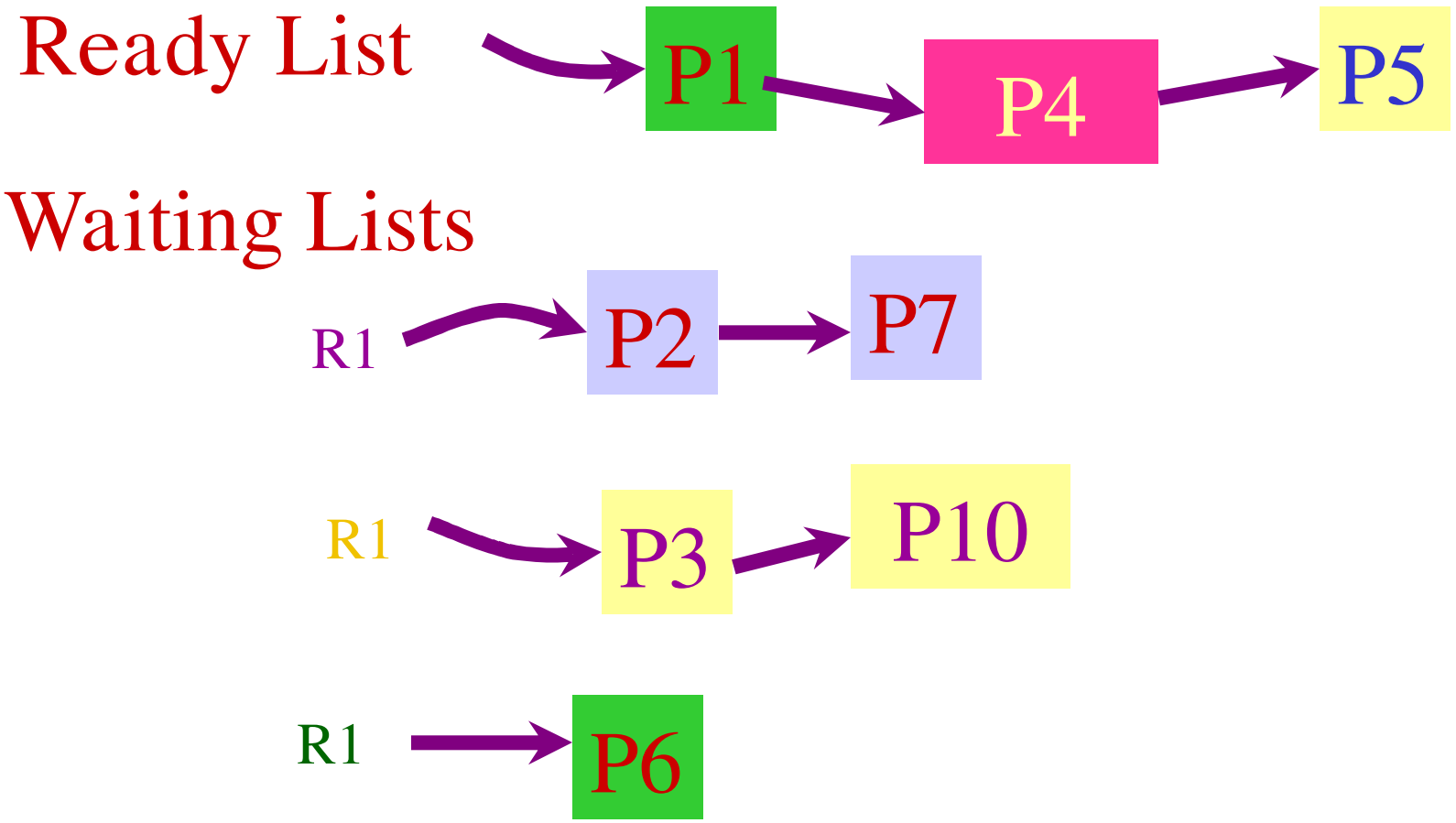


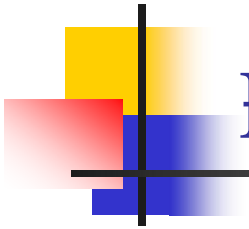


An ninh trật tự cho môi trường đa tiến trình !

- **Bảo vệ tiến trình :**
  - **Ngăn cản các tiến trình xâm phạm tài nguyên, can thiệp vàoxử lý của nhau => KGĐC riêng biệt, 2 mode xử lý**
  - **Bảo đảm quyền tiến triển xử lý cho mỗi tiến trình => công bằng trong các chiến lược phân phối tài nguyên.**
- **Trao đổi thông tin , phối hợp hoạt động ?**
  - **Nhu cầu ?**
  - **Vấn đề ? => Chương kế tiếp**
  - **Giải pháp ?**

Các danh sách tiến trình





Điều phối tiến trình

- Mục tiêu ?
- Các cấp độ điều phối
- Thời điểm ra quyết định điều phối ?
- Đánh giá chiến lược điều phối ?
- Một số chiến lược điều phối

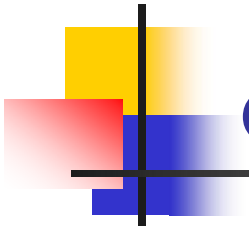
Điều phối tiến trình

SCHEDULER

chọn một tiến trình nhận cpu

DISPATCHER

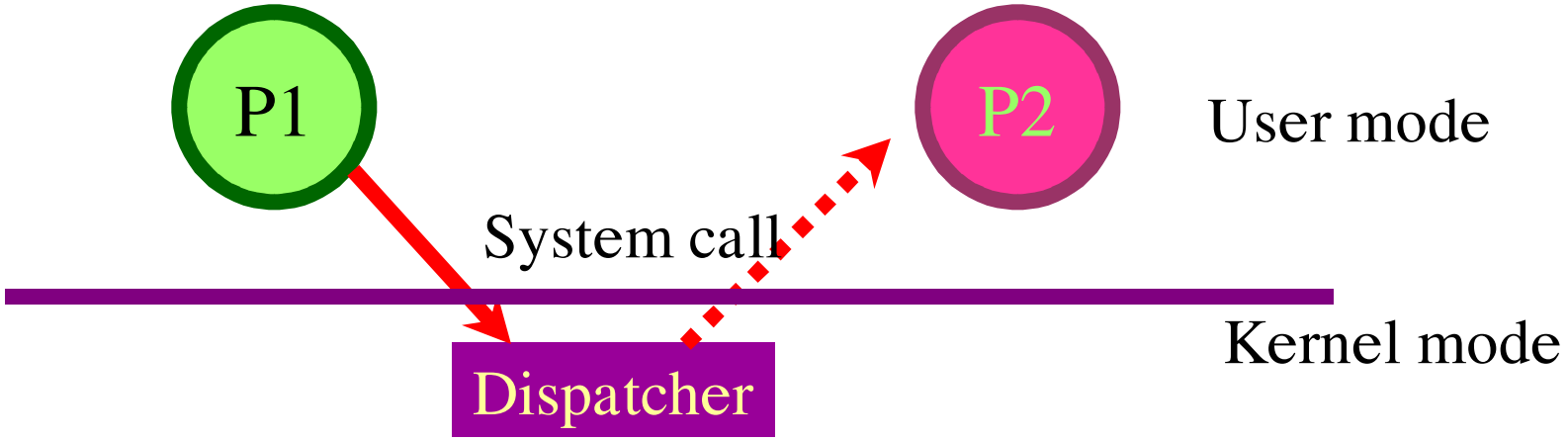
chuyển đổi ngữ cảnh



Chuyển đổi ngữ cảnh (context switching)

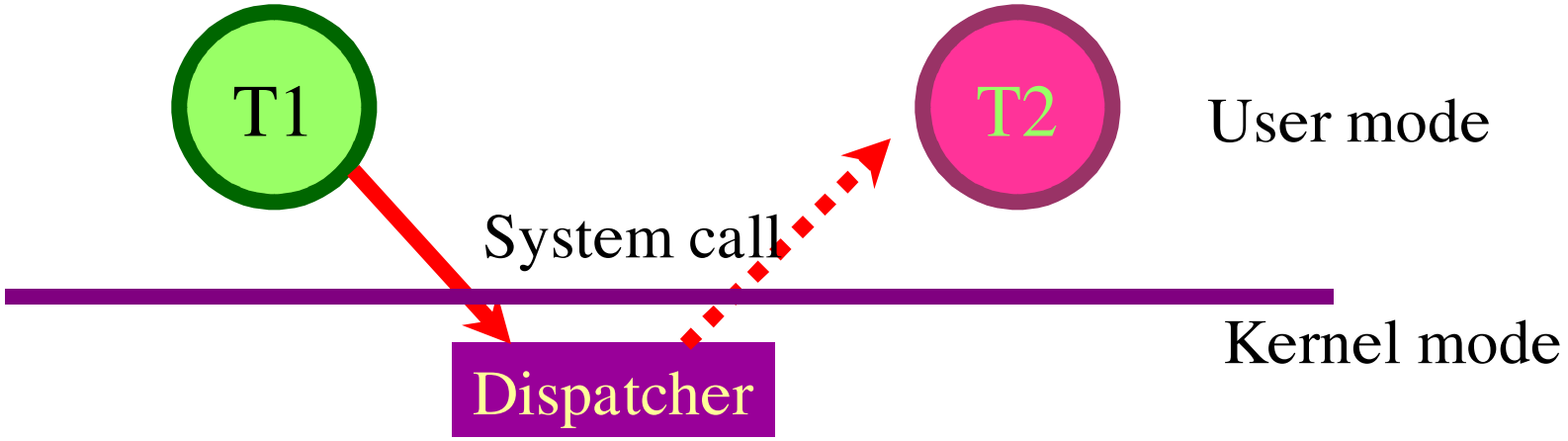
- **Kịch bản :**
  - **Lưu ngữ cảnh tiến trình hiện hành**
  - **Nạp ngữ cảnh tiến trình được chọn kế tiếp**
- **Chi tiết cụ thể phụ thuộc vào phần cứng**
  - **general-purpose & floating point registers, co-processor state...**
- **Chi phí chuyển đổi ngữ cảnh :**
  - **Giữa các tiến trình ?**
  - **Giữa các tiểu trình ?**

Chuyển đổi ngữ cảnh giữa các tiến trình



- **Chuyển đổi mode xử lý**
- **Chuyển đổi IP và các thanh ghi khác của CPU**
- **Chuyển đổi không gian địa chỉ**

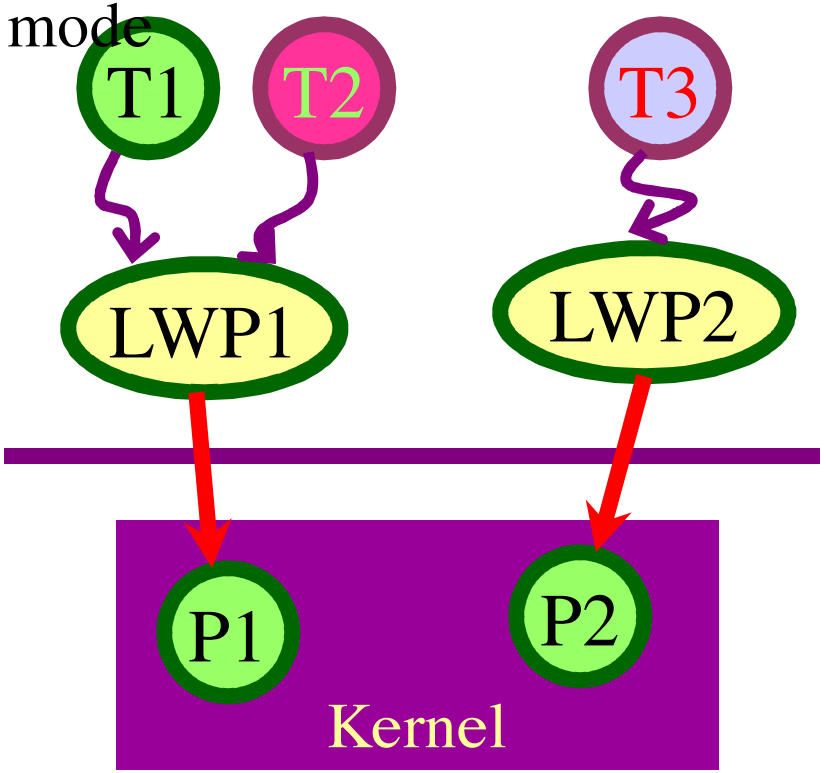
Tiểu trình hạt nhân (Kernel thread)

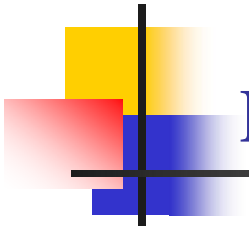


- Khái niệm tiểu trình được xây dựng bên trong hạt nhân
- Distpatcher làm việc với đơn vị là tiểu trình

Tiểu trình người dùng (User thread)

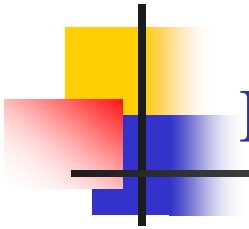
- Khái niệm tiểu trình được hỗ trợ bởi một thư viện hoạt động trong user mode
- Distpatcher của hạt nhân làm việc với đơn vị là tiến trình
- ThreadDistpatcher làm việc với đơn vị là tiểu trình
  - P –– LWP - T
- Không cần chuyển đổi chế độ xử lý khi chuyển đổi các tiểu trình cùng thuộc 1 tiến trình.





Lựa chọn tiến trình ?

- Tác vụ của Scheduler
- Mục tiêu ?
  - Sử dụng CPU hiệu quả
  - Đảm bảo tất cả các tiến trình đều tiến triển xử lý
- Tiêu chuẩn lựa chọn ?
  - Tất cả các tiến trình đều như nhau ?
  - Đề xuất một độ ưu tiên cho mỗi tiến trình ?
- Thời điểm lựa chọn ? (Thời điểm kích hoạt Scheduler())

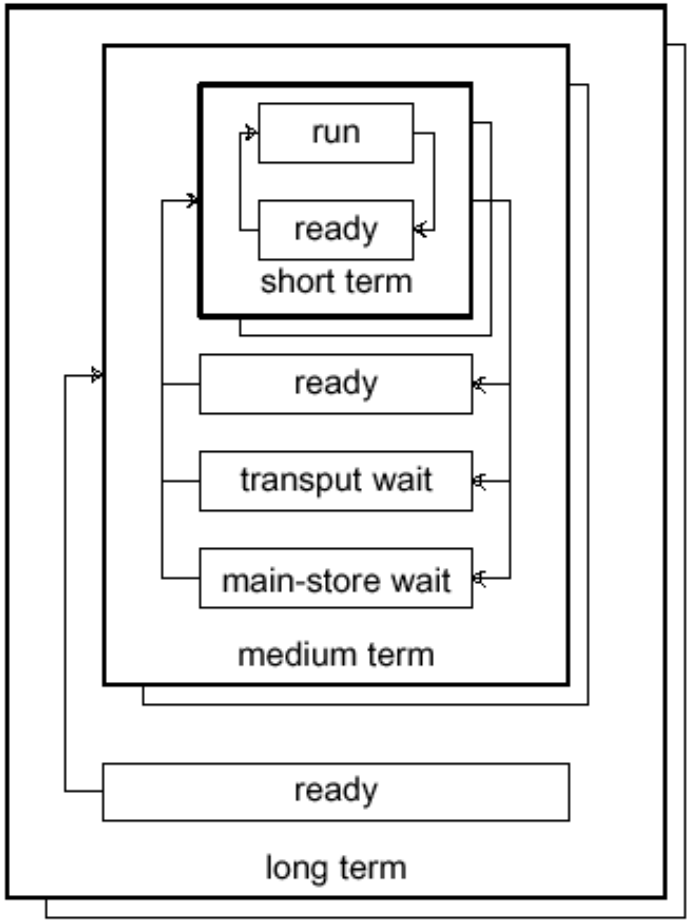


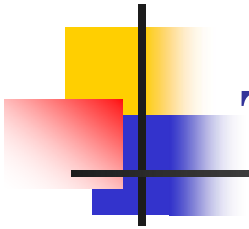
Mục tiêu điều phối

- **Hiệu qủa (Efficiency)**
  - **Thời gian**
    - **Đáp ứng (Response time)**
    - **Hoàn tất(Turnaround Time = T_{quit} -T_{arrive}):**
    - **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 cấp độ điều phối

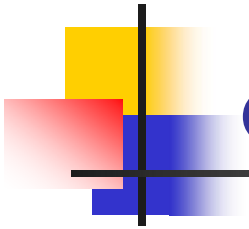
- **Longterm scheduling** : chọn tiến trình kế tiếp được khởi động (mang vao bộ nhớ và nhận trạng thái ready)
- **Mediumterm scheduling** : quyết định chuyển tiến trình đang running sang trạng thái blocked.
- **Shorterm scheduling** : chọn 1 tiến trình ở trạng thái ready để chuyển sang trạng thái running.
- Không có sự phân biệt rõ





Thời điểm ra quyết định điều phối

- **Điều phối độc quyền (non-preemptive scheduling)**: tiến trình được chọn độc chiếm CPU
- **Điều phối không độc quyền (preemptive scheduling)**: tiến trình được chọn có thể bị « cướp » CPU bởi tiến trình có độ ưu tiên cao hơn



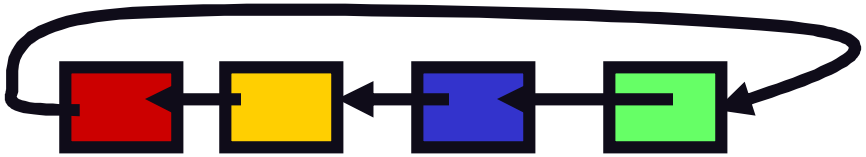
Các chiến lược điều phối

- **FIFO**
- **RR**
- **SJF**
- **MULTILEVELFEEDBACK**
- **LOTTERY**

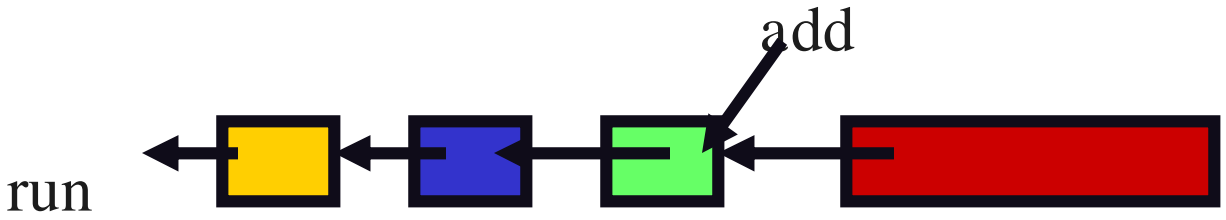
FIFO – RR -SJF

- FIFO

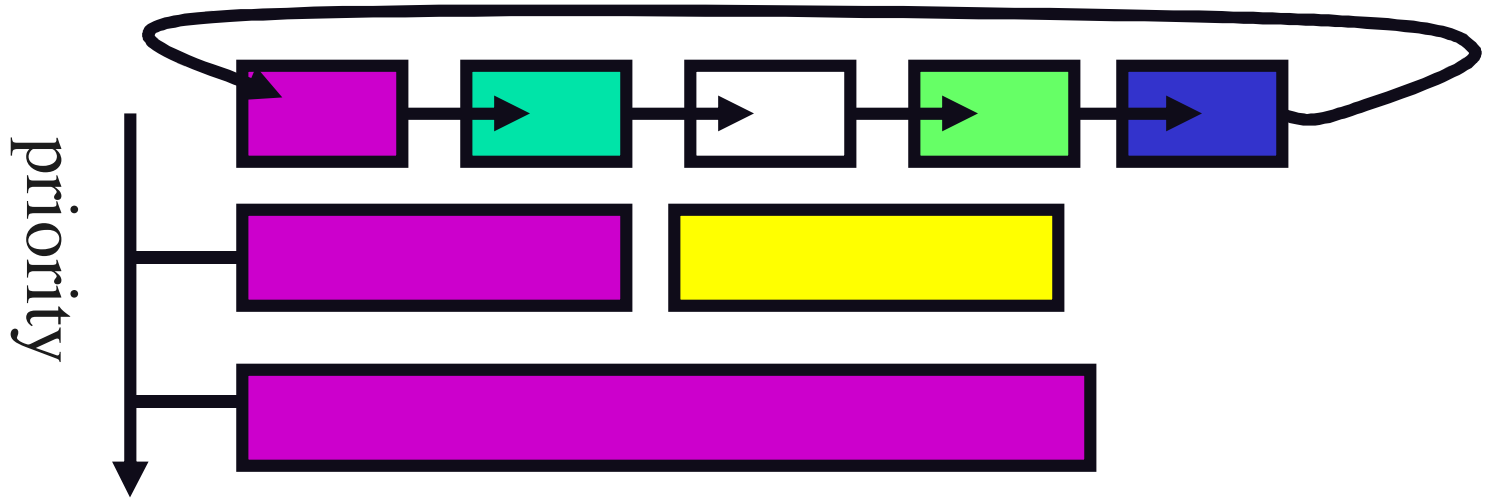
- RR



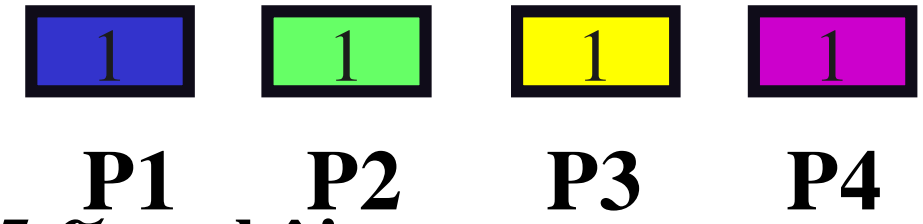
- SJF



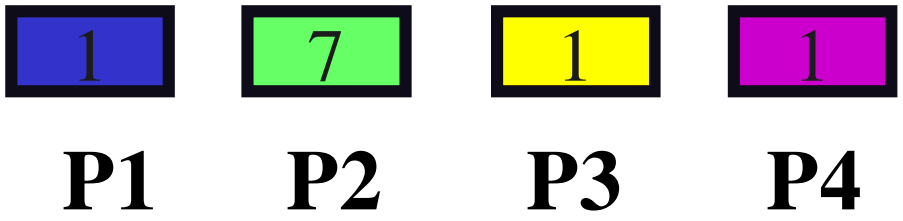
Multilevel Feedback



Lottery



P2 có 25 % cơ hội

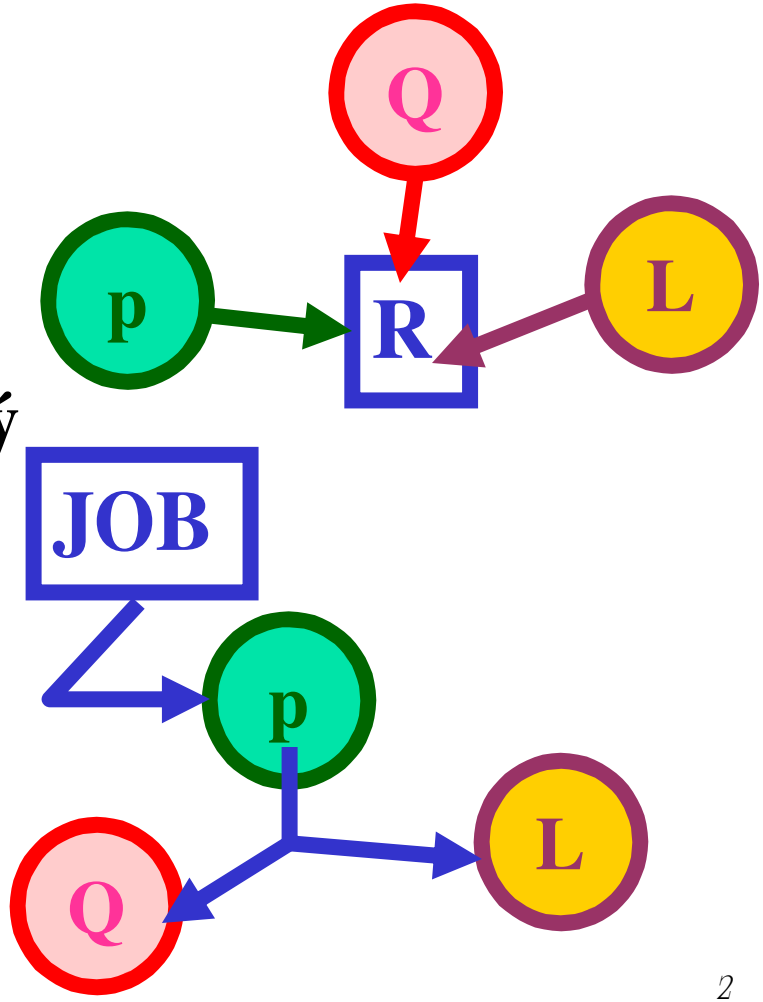


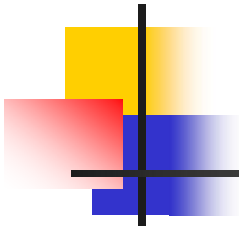
P2 có 70 % cơ hội

BÀI 4 : LIÊN LẠC GIỮA CÁC TIẾN TRÌNH & VẤN ĐỀ ĐỒNG BỘ HOÁ

Nhu Cầu Liên Lạc

- **Chia sẻ thông tin**
- **Phối hợp tăng tốc độ xử lý**



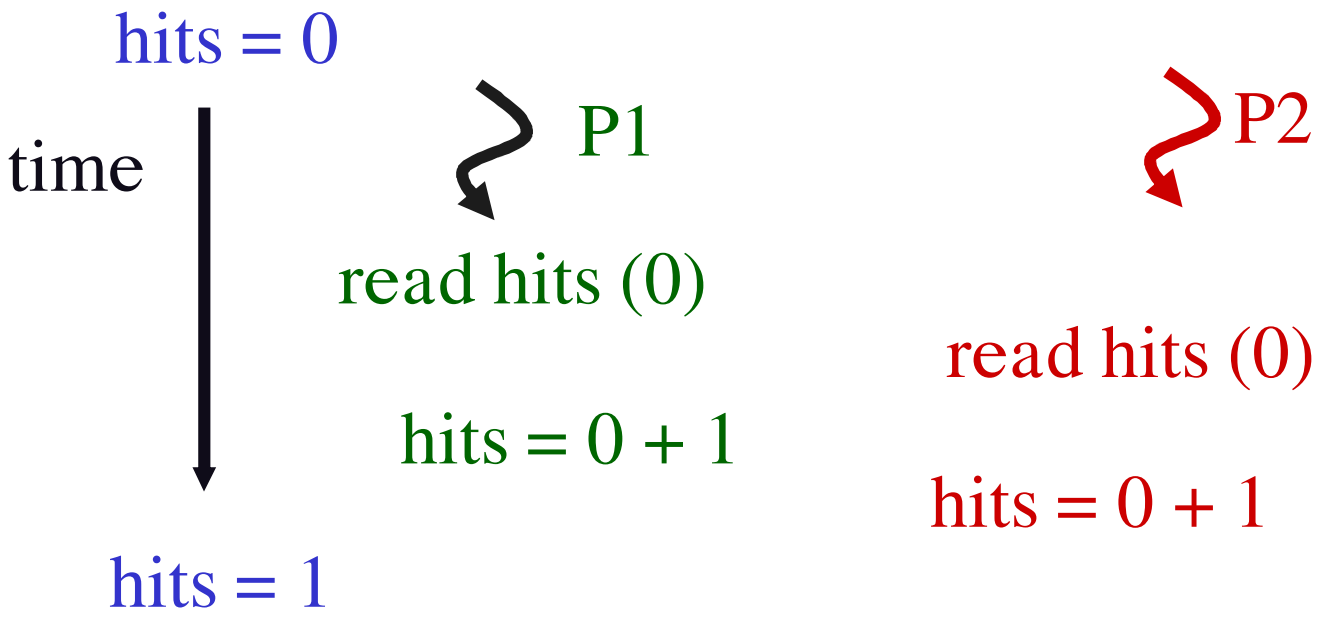


Các Cơ Chế Liên Lạc

- **Signal**
  - ☹Không truyền được dữ liệu
- **Pipe**
  - ☹ Truyền dữ liệu không cấu trúc
- **Shared Memory**
  - ☺ Broadcast
  - ☹ Mâu thuẫn truy xuất => nhu cầu đồng bộ hoá
- **Message**
  - ☺ Liên lạc trên môi trường phân tán
- **Socket**
  - ☺ Liên lạc trên nhiều môi trường khác biệt

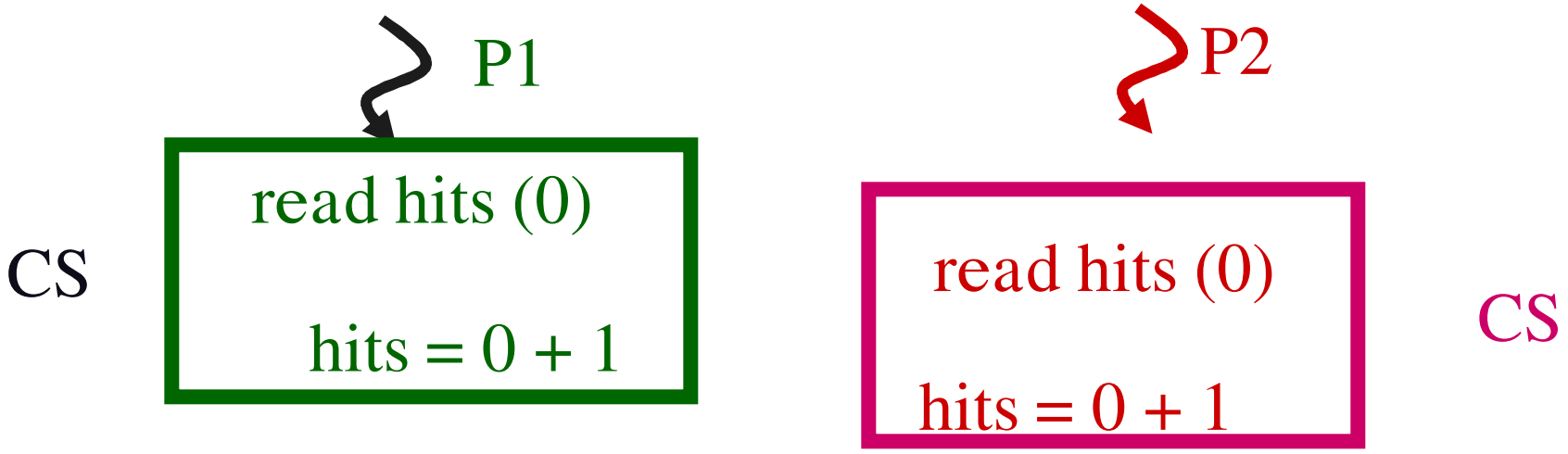
Race condition

- **P1 và P2 chia sẻ biến chung hits**



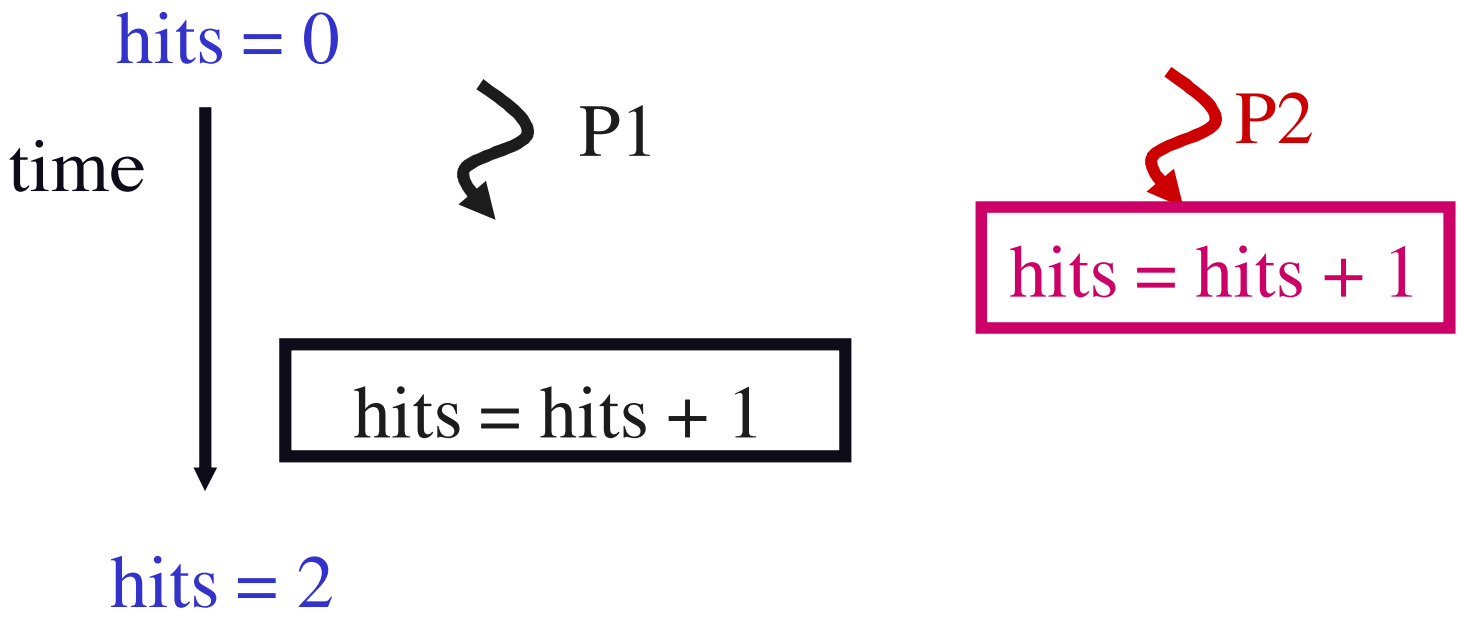
☹ **Kết quả cuối cùng không dự đoán được !**

Miền găng (critical section)

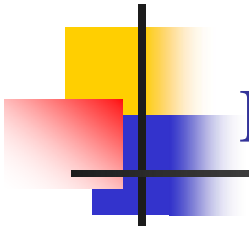


CS là đoạn chương trình có khả năng gây ra hiện tượng race condition

Giải pháp tổng quát



Bảo đảm tính "độc quyền truy xuất" miền găng tại một thời điểm



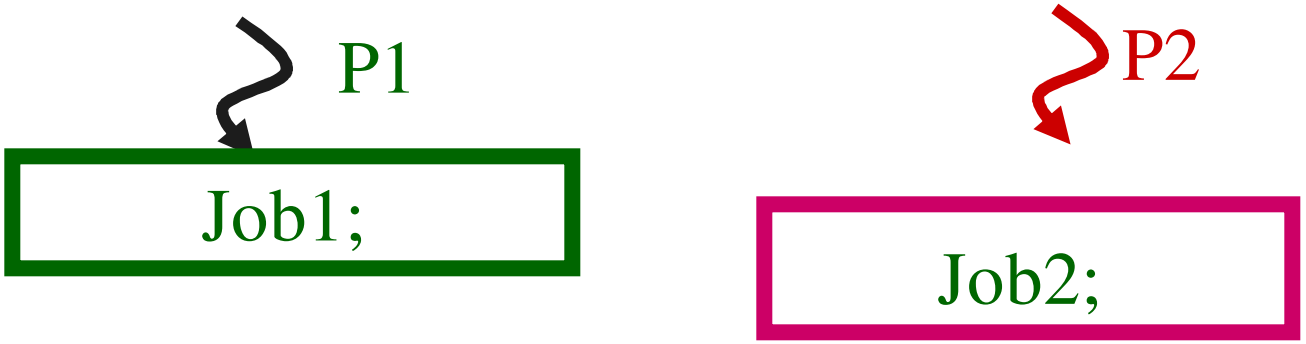
Mô hình đảm bảo độc quyền truy xuất

Kiểm tra và dành quyền vào CS

CS;

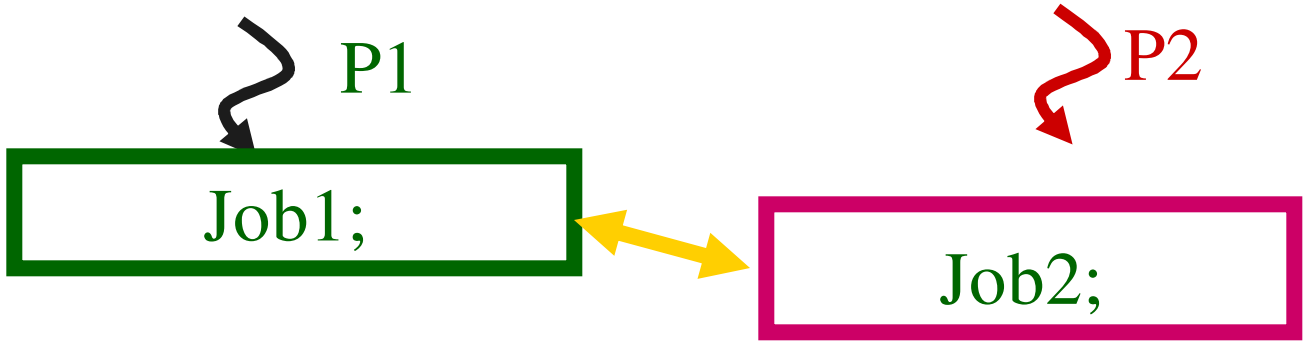
Từ bỏ quyền sử dụng CS

Rendez-Vous



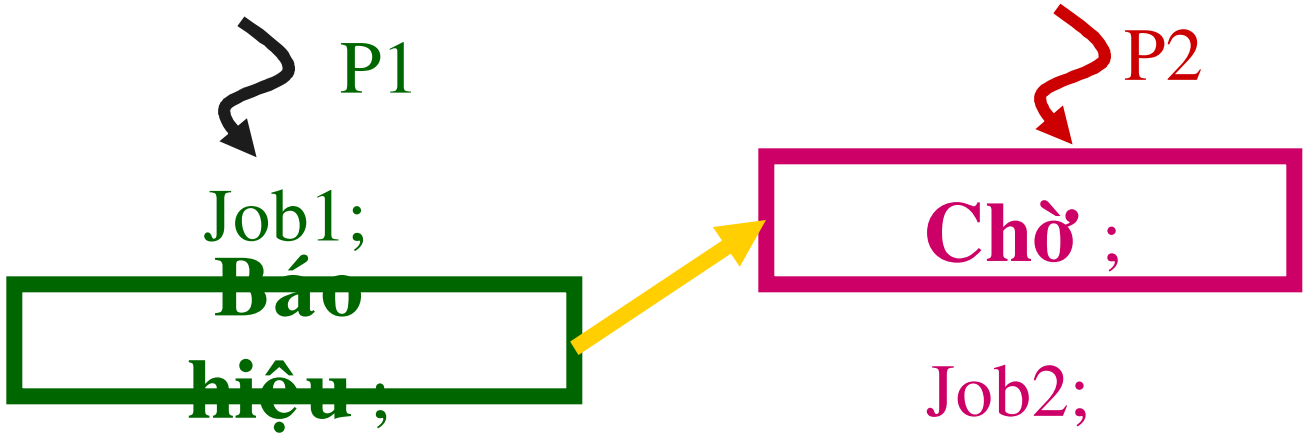
Làm thế nào bảo đảm trình tự thực hiện Job1 - Job2 ?

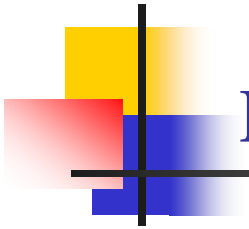
Giải pháp



Hai tiến trình cần trao đổi thông tin về diễn tiến xử lý

Mô hình tổ chức phối hợp hoạt động giữa hài tiến trình



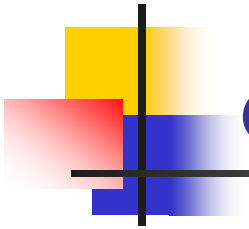


Bài toán đồng bộ hoá

- **Nhiều tiến trình chia sẻ tài nguyên chung đồng thời :**
  - **Tranh chấp ?**
  - **Nhu cầu "độc quyền truy xuất" (mutual exclusion)**
- **Các tiến trình phối hợp hoạt động :**
  - **Tương quan diễn tiến xử lý ?**
  - **Nhu cầu "hò hẹn" (rendez-vous)**

BÀI 5 : CÁC GIẢI PHÁP ĐỒNG BỘ HOÁ

- Nhóm giải pháp Busy Waiting
  - Sử dụng các biến cờ hiệu
  - Sử dụng việc kiểm tra luân phiên
  - Giải pháp của Peterson
  - Cấm ngắt
  - Chỉ thị TSL
- Nhóm giải pháp Sleep & Wakeup
  - Semaphore
  - Monitor
  - Message



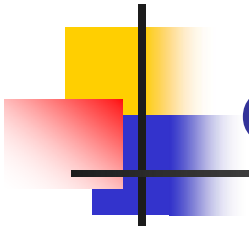
Các giải pháp "Busy waiting"

While (chưa có quyền) donothing() ;

CS;

Từ bỏ quyền sử dụng CS

- **Tiếp tục tiêu thụ CPU trong khi chờ đợi vào miền gă**
- **Không đòi hỏi sự trợ giúp của Hệ điều hành**



Các giải pháp "Sleep & Wake up"

if (chưa có quyền) Sleep() ;

CS;

Wakeup(somebody);

- **Từ bỏ CPU khi chưa được vào miền găng**
- **Cần được Hệ điều hành hỗ trợ**

Semaphore

Semaphore s; // s >=0
Down (s) & Up(s)

- **Được hỗ trợ bởi HĐH**
- **Tổ chức độc quyền truy xuất**

Down (s)
CS;
Up(s)

P1 :
Job1;
Up(s)

P2:
Down (s);
Job2;

- **Tổ chức "hò hẹn"**

Monitor

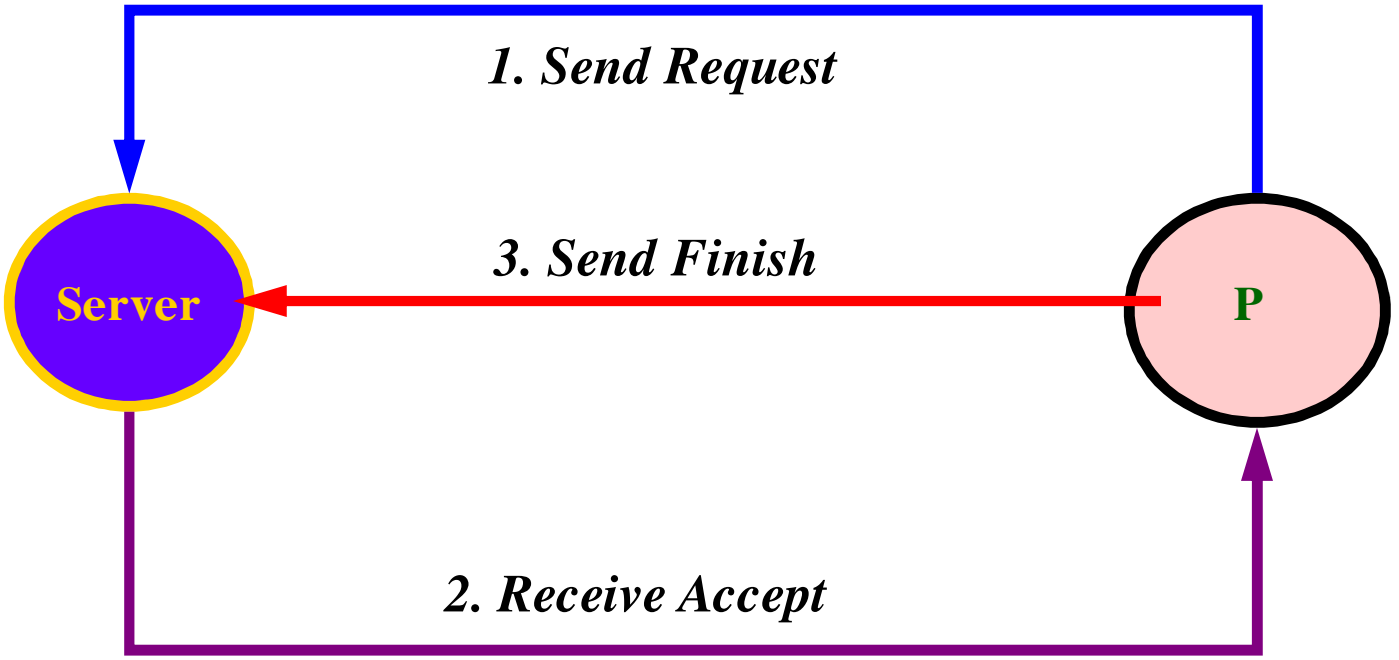
```
Monitor m
int x;
Condition c;

Function F1()
{  ….wait(c); …}

Function F2()
{  ….signal(c); …}
```

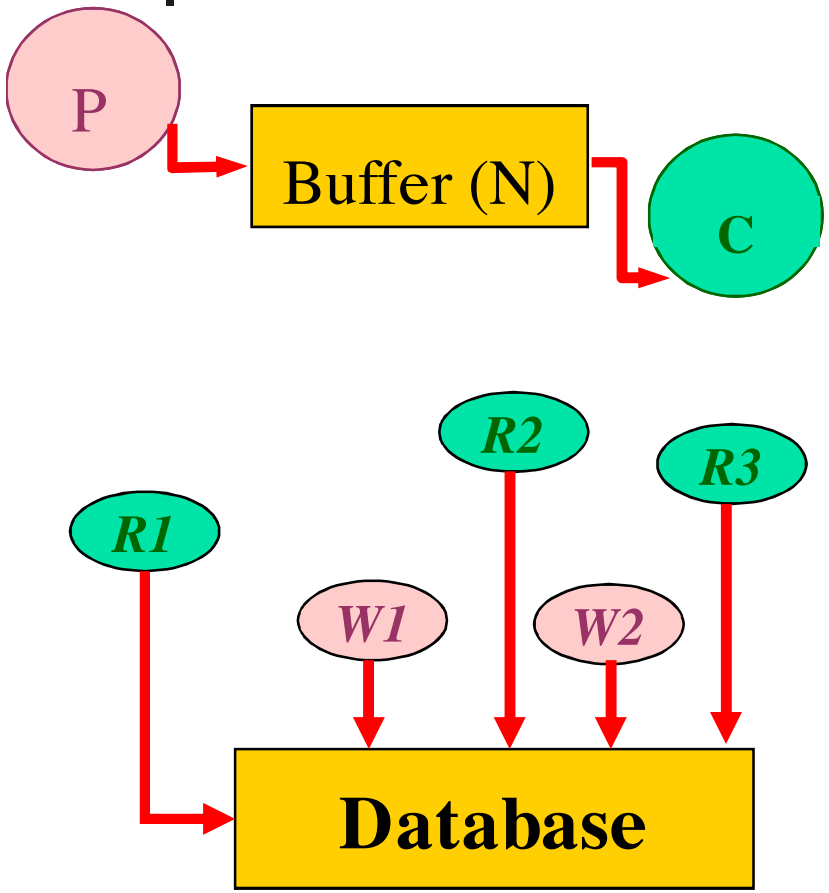
- **Được hỗ trợ bởi NNLT**
- **Bảo đảm độc quyền truy xuất tự động**
- **Sử dụng biến điều kiện để thực hiện "Hò hẹn"**

Message



- Được hỗ trợ bởi HĐH
- Đồng bộ hóa trên môi trường phân tán

Các bài toán đồng bộ hoá kinh điển

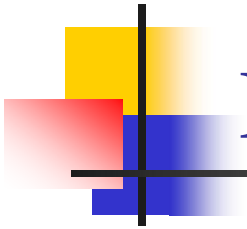


Producer-Consumer

- P không được ghi dữ liệu vào buffer đã đầy
- C không được đọc dữ liệu từ buffer đang trống
- P và C không được thao tác trên buffer cùng lúc

Readers - Writers

- W không được cập nhật dữ liệu khi có một R đang truy xuất CSDL .
- Tại một thời điểm , chỉ cho phép một Wđược sửa đổi nội dung CSDL.

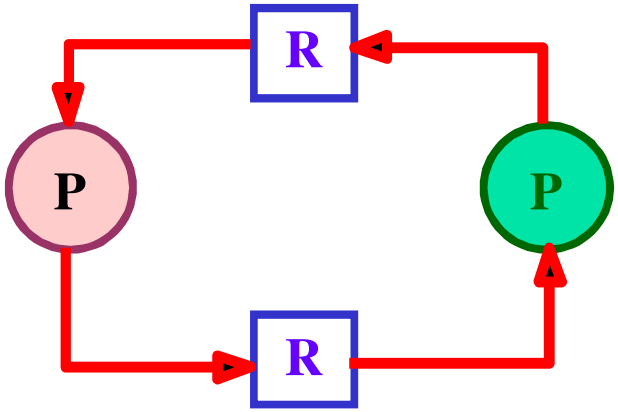


BÀI 6 : HIỆN TƯỢNG DEALOCK

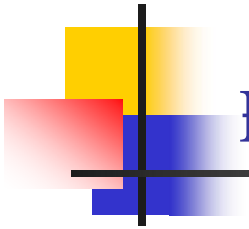
Deadlock

: P đang giữ tài nguyên R

: P đang yêu cầu tài nguyên R

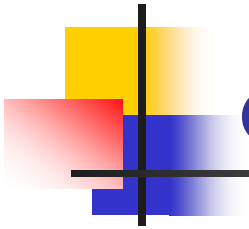


: một tình huống tắt nghẽn



Điều kiện phát sinh Deadlock

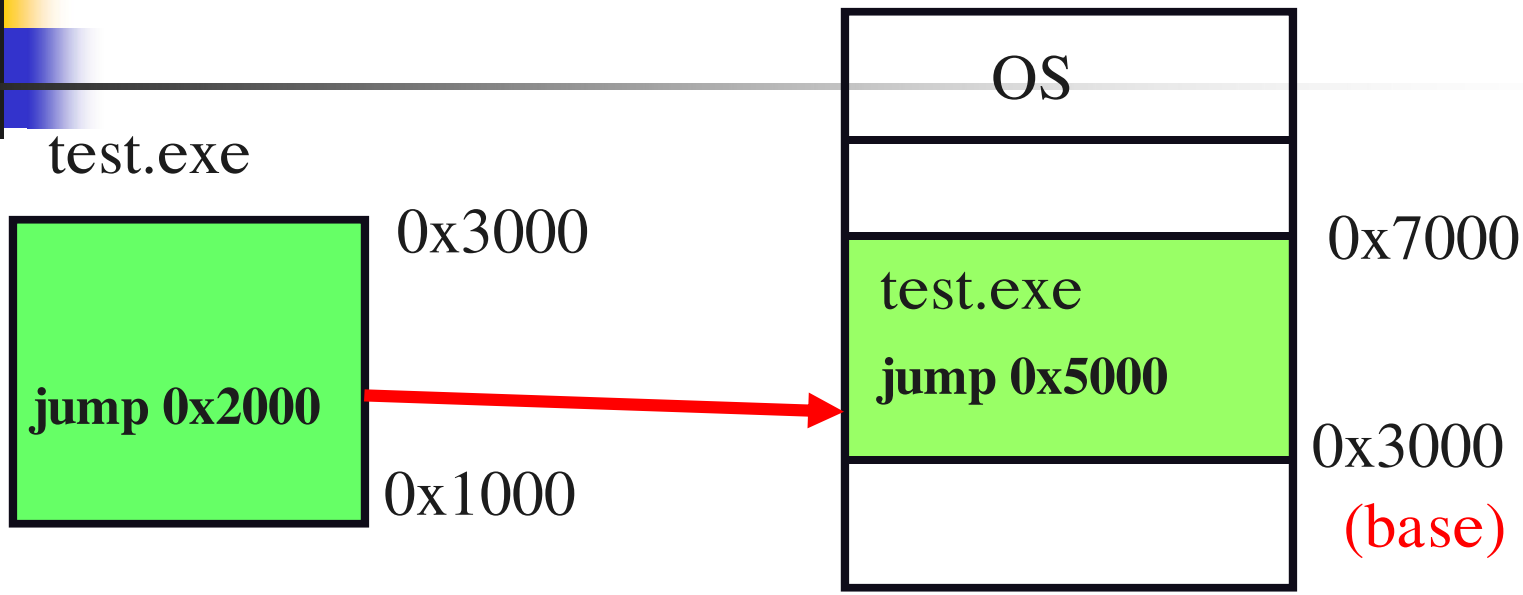
- Sử dụng tài nguyên không thể chia sẻ
- Tiến trình chiếm giữ và yêu cầu thêm tài nguyên
- Hệ thống không thể thu hồi tài nguyên nếu tiến trình không từ bỏ
- Tồn tại một chu trình trong đồ thị cấp phát



Giải quyết Deadlock

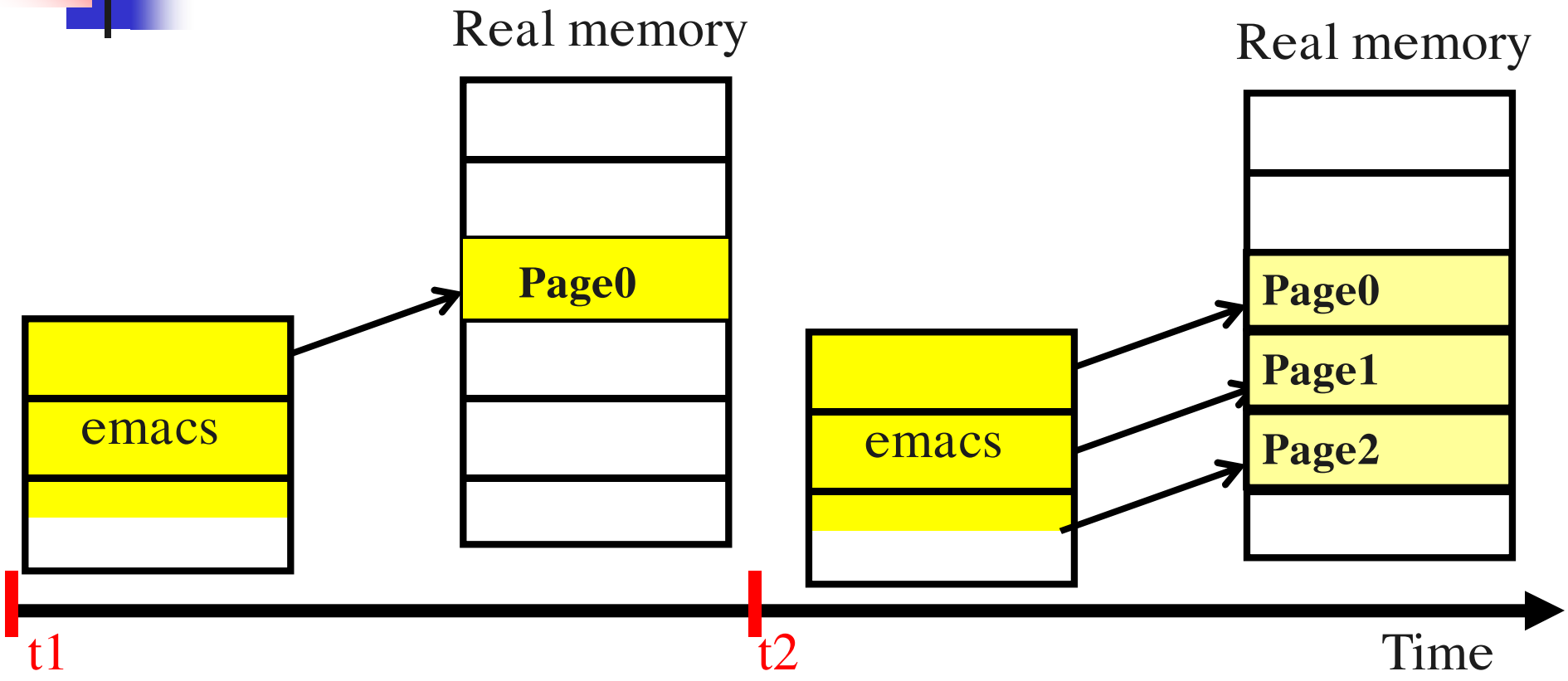
- Bỏ qua
- Phòng ngừa
- Phát hiện và hiệu chỉnh

BÀI 7 : BỘ NHỚ ẢO



- **Cho đến nay : Nạp toàn bộ tiến trình vào bộ nhớ rồi thực hiện nó...**
  - **Chậm, lãng phí bộ nhớ**
  - **Nếu kích thước tiến trình lớn hơn dung lương bộ nhớ chính ?**
  - **Lưu ý : tại 1 thời điểm chỉ có một chỉ thị được thực hiện**

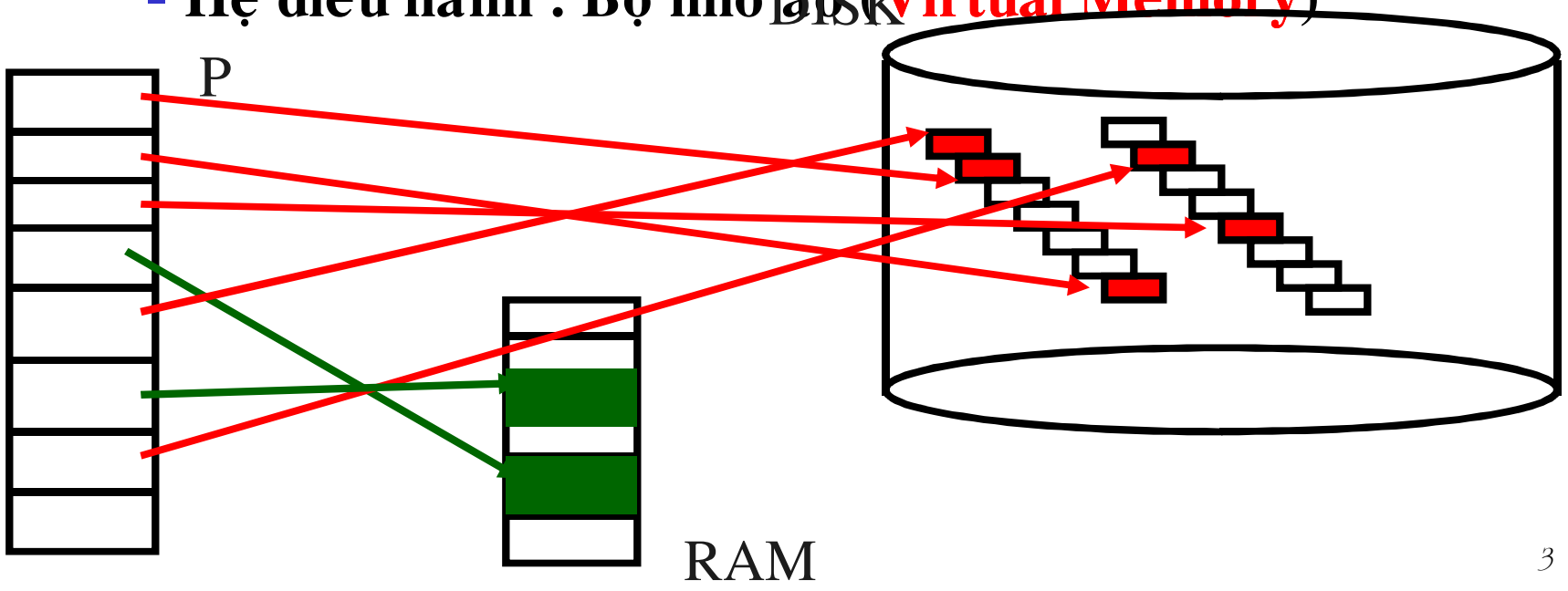
Giải pháp



- **Nạp từng phần chương trình khi cần thiết**
- **Demand paging**

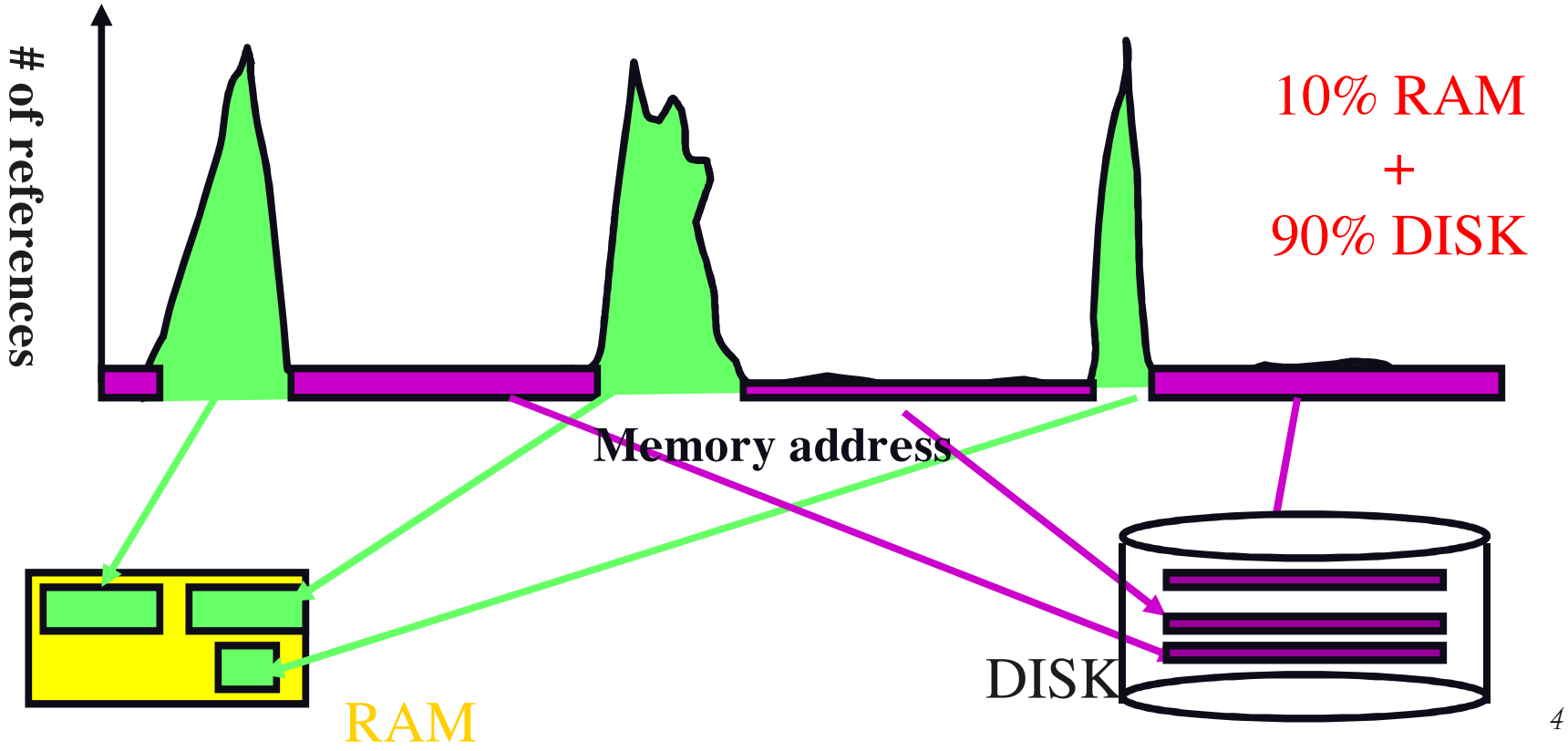
Cơ chế

- Sử dụng bộ nhớ phụ để lưu trữ tạm thời các trang chưa sử dụng
- Ai chịu trách nhiệm chuyển đổi ?
  - Lập trình viên : Overlay
  - Hệ điều hành : Bộ nhớ ảo (Virtual Memory)



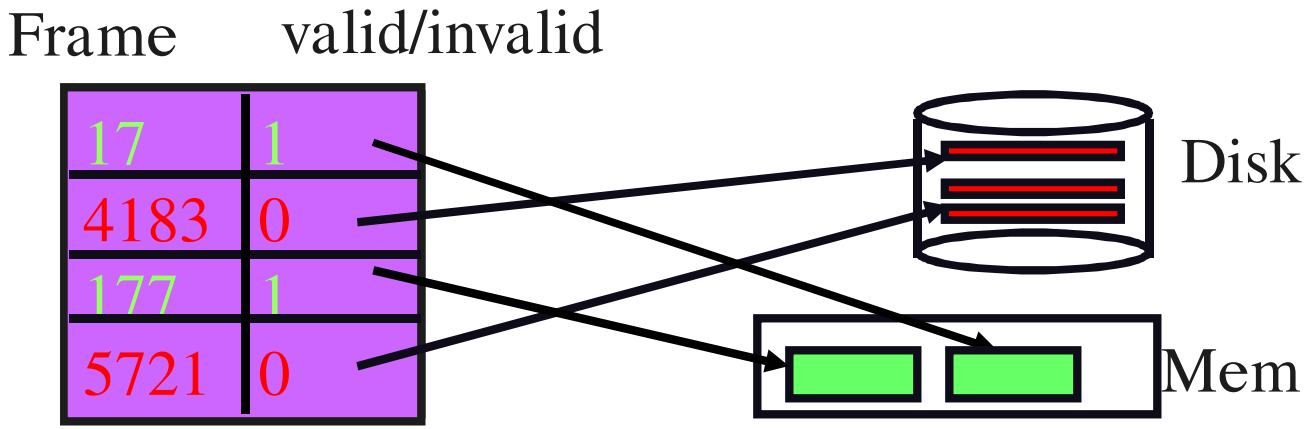
Bộ nhớ ảo = "lời nói dối vĩ đại"

- **Người dùng : sở hữu bộ nhớ "vô hạn", "riêng biệt"**
- **Hệ điều hành : "thầm lặng" thực hiện quá trình swapping**



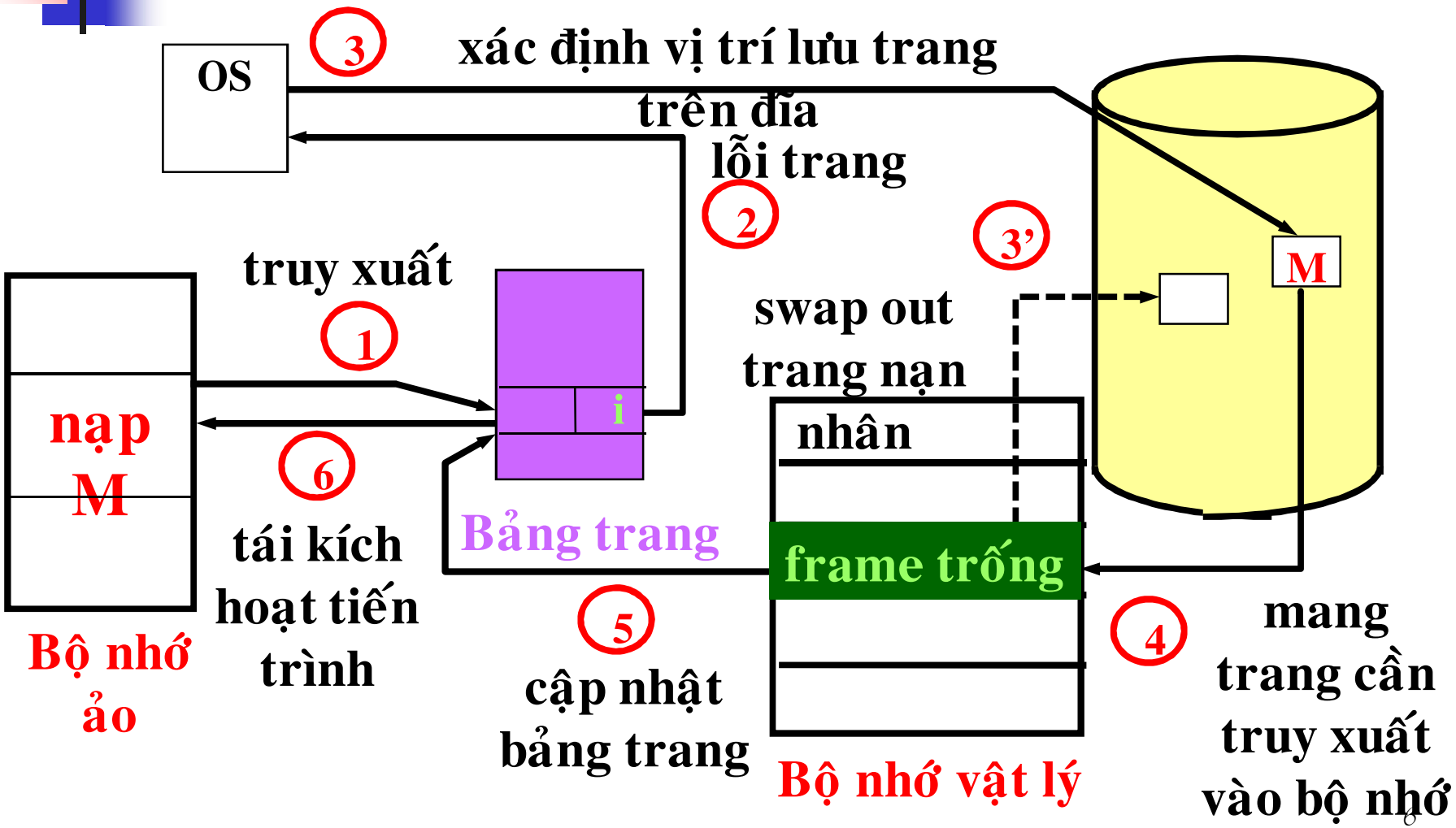
Thực hiện Bộ nhớ ảo

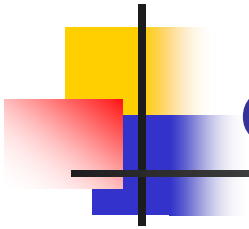
- **Bảng trang : thêm 1 bit valid/invalid để nhận diện trang đã hay chưa được nạp vào RAM**



- **Truy xuất đến một trang chưa được nạp vào bộ nhớ : lỗi trang (page fault)**

Xử lý lỗi trang





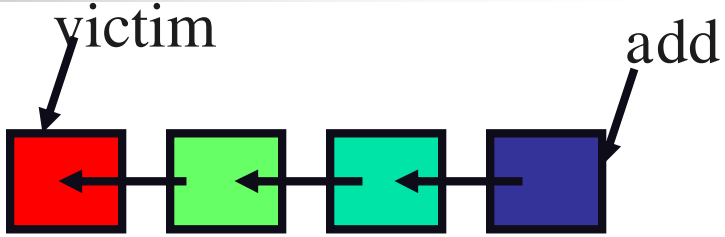
Các câu hỏi

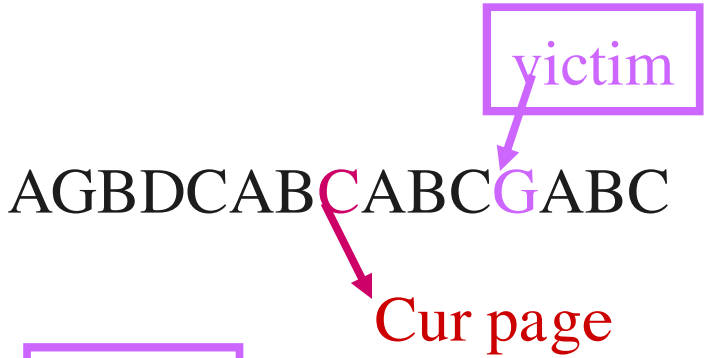
1. **Chọn trang nạn nhân ? => Chiến lược thay thế trang**

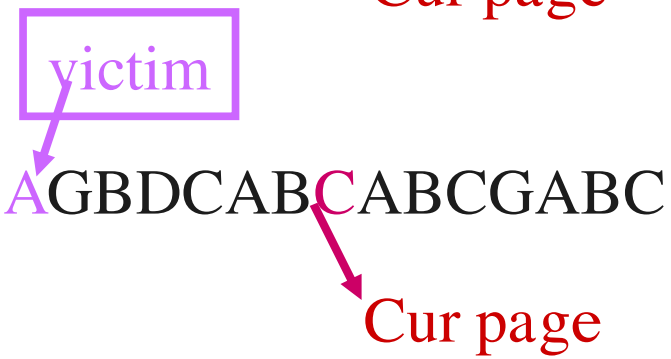
2. **Chọn trang nào để nạp ? => Chiến lược nạp**

Chiến lược thay thế trang

- **FIFO: trang "già " nhất**
  - **Công bằng ?**
  - **Không xét đến tính sủ dụng !**
- **TỐI ƯU : trang lâu sử dụng đến nhất trong tương lai**
  - **Tần suất lỗi trang thấp nhất**
  - **Không khả thi !**
- **LRU :trang lâu nhất chưa sử dụng đến trong quá khứ**
  - **Dự đoán tương lai LRU = MIN ?**





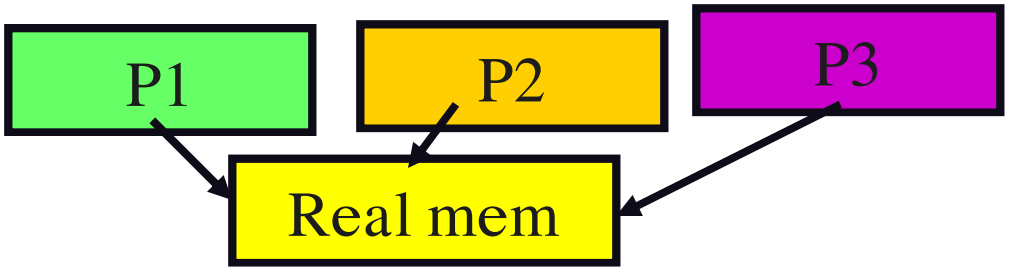


Chiến lược nạp

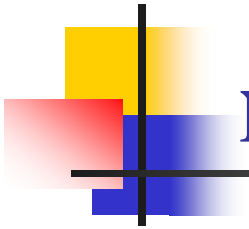
- **Demand paging : nạp trang được yêu cầu**
  - **Khi nào ?**
  - **Nạp sau : tần suất lỗi trang cao ? => pure demand paging**
  - **Nạp trước : làm sao biết ? =>prepaging**

Thrashing = ảo tưởng sụp đổ !

- **Các tiến trình trong hệ thống yêu cầu bộ nhớ nhiều hơn khả năng cung cấp của hệ thống !**



- **Tất cả tiến trình đầu bận rộn xử lý lỗi trang !**
- **IO hoạt động 100 %, CPU rảnh !**
- **Hệ thống ngừng trệ**

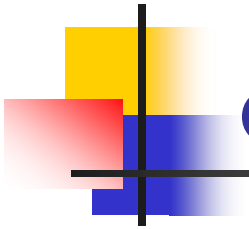


Nguyên nhân Thrashing

1. **Tiến trình không tái sử dụng bộ nhớ (quá khứ != tương lai)**
2. **Tiến trình tái sử dụng bộ nhớ, nhưng với kích thước lớn hơn**

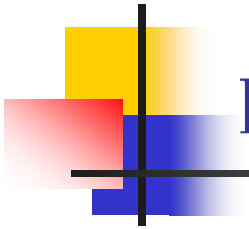
- **Chỉ có thể kiểm soát thrashing do nguyên nhân 3.**

3. **Quá nhiều tiến trình trong hệ thống**



Giải quyết thrasing với mô hình Working set

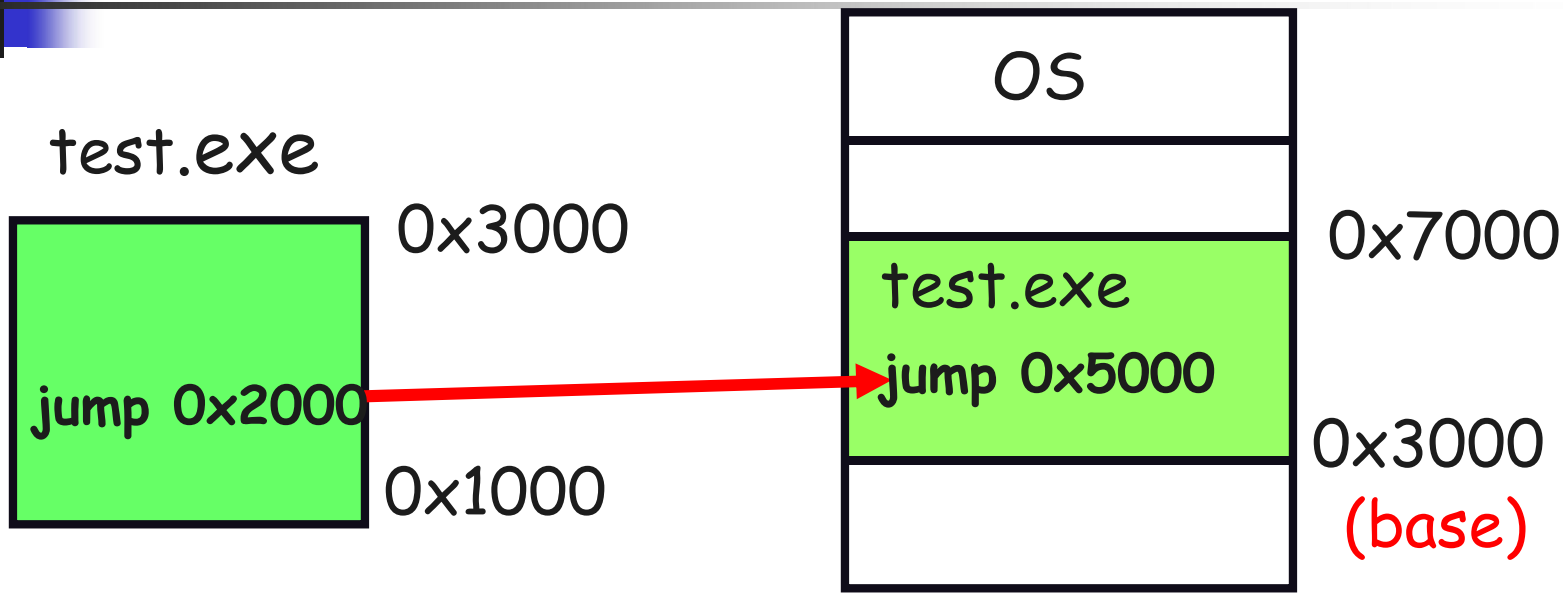
- **Working set = tập hợp các trang tiến trình đang truy xuất tại 1 thời điểm.**
- **Hệ điều hành :**
  - **Chỉ nạp một tiến trình khi có đủ khung trang tự do cho working set của nó.**
  - **Kiểm soát mức độ đa chương của hệ thống : Nếu tổng số khung trang yêu cầu của các tiến trình trong hệ thống vượt quá các khung trang có thể sử dụng, chọn một tiến trình để tạm dừng, ngược lại, khi tổng working set bé hơn số khung trang tự do, nạp thêm tiến trình**



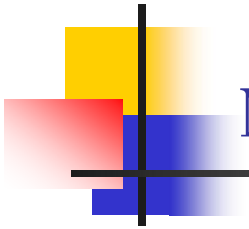
BÀI 8 : CÁC MÔ HÌNH BỘ NHỚ ĐƠN GIẢN

- *Cấp phát liên tục :*
  - *Linker-Loader*
  - *Base & Bound*

Mô hình Linker-Loader



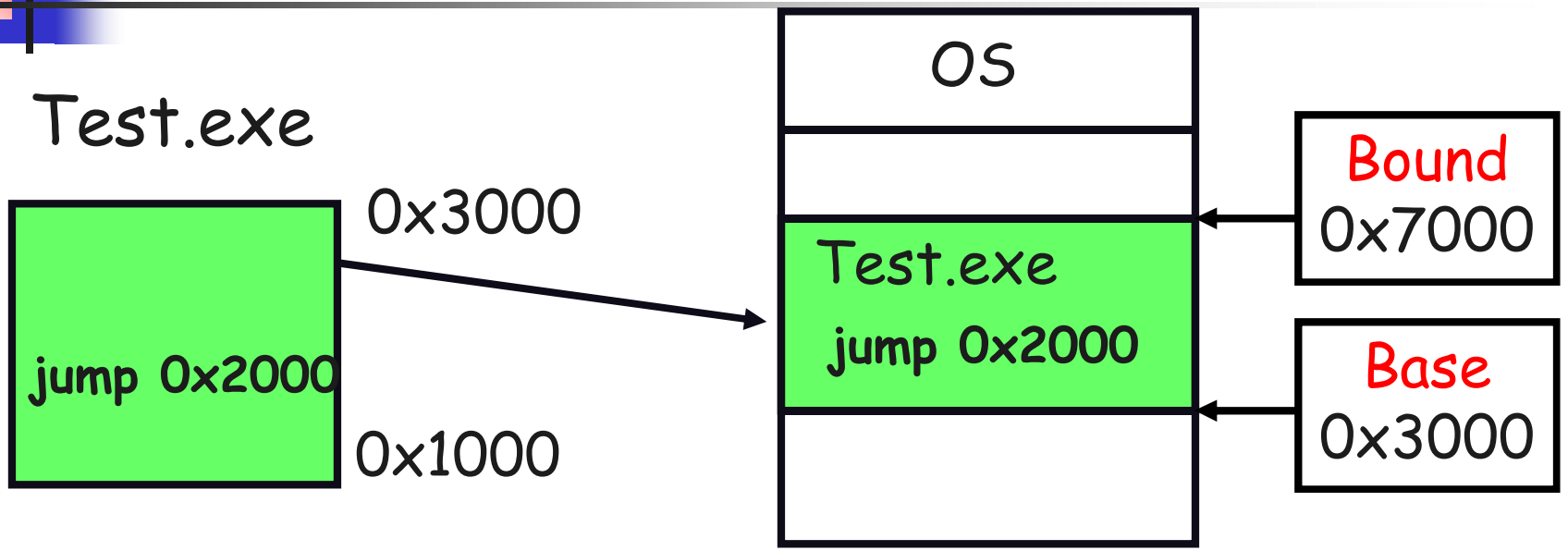
- Tại thời điểm Link, giữ lại các địa chỉ logic
- Vị trí base của tiến trình trong bộ nhớ xác định được vào thời điểm nạp : địa chỉ physic = địa chỉ logic + base



Mô hình Linker-Loader

- Bảo vệ ?
- Dời chuyển sau khi nạp ?
- Không có vùng nhớ liên tục đủ lớn để chứa toàn bộ tiến trình ?

Mô hình Base & Bound

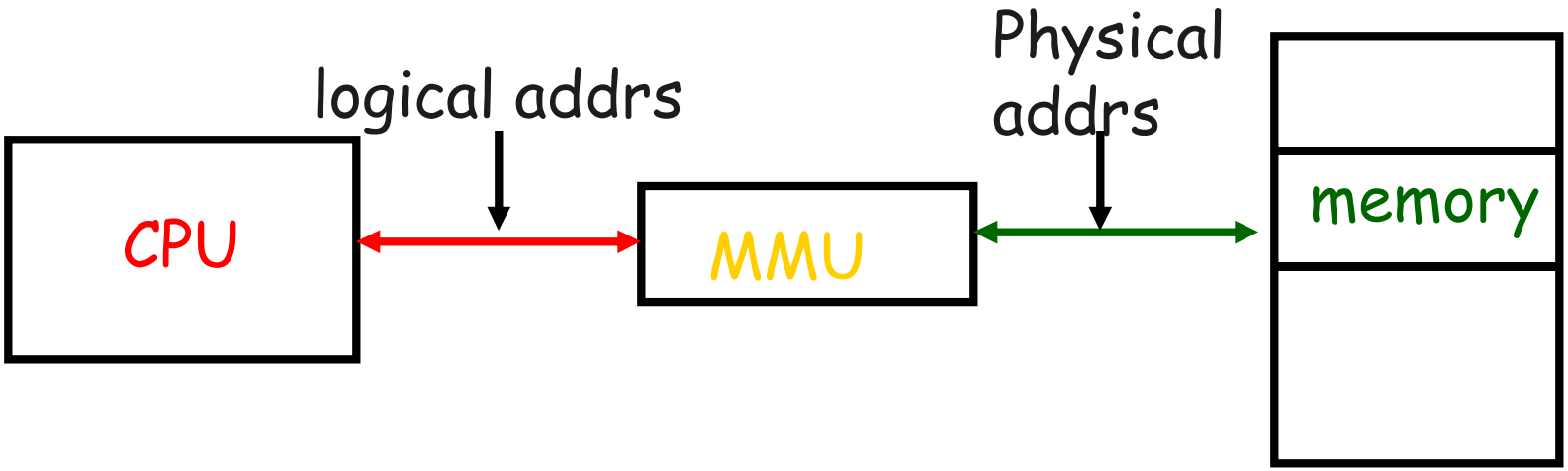


- Tại thời điểm Link, giữ lại các địa chỉ logic
- Vị trí base , bound được ghi nhận vào 2 thanh ghi:
- Kết buộc địa chỉ vào thời điểm thi hành => tái định vị được :

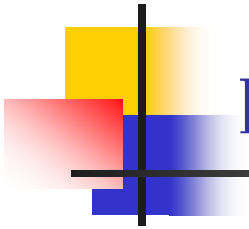
  địa chỉ physic = địa chỉ logic + base register
- Bảo vệ : địa chỉ hợp lệ ⊆ [base, bound]

Mô hình Base & Bound

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



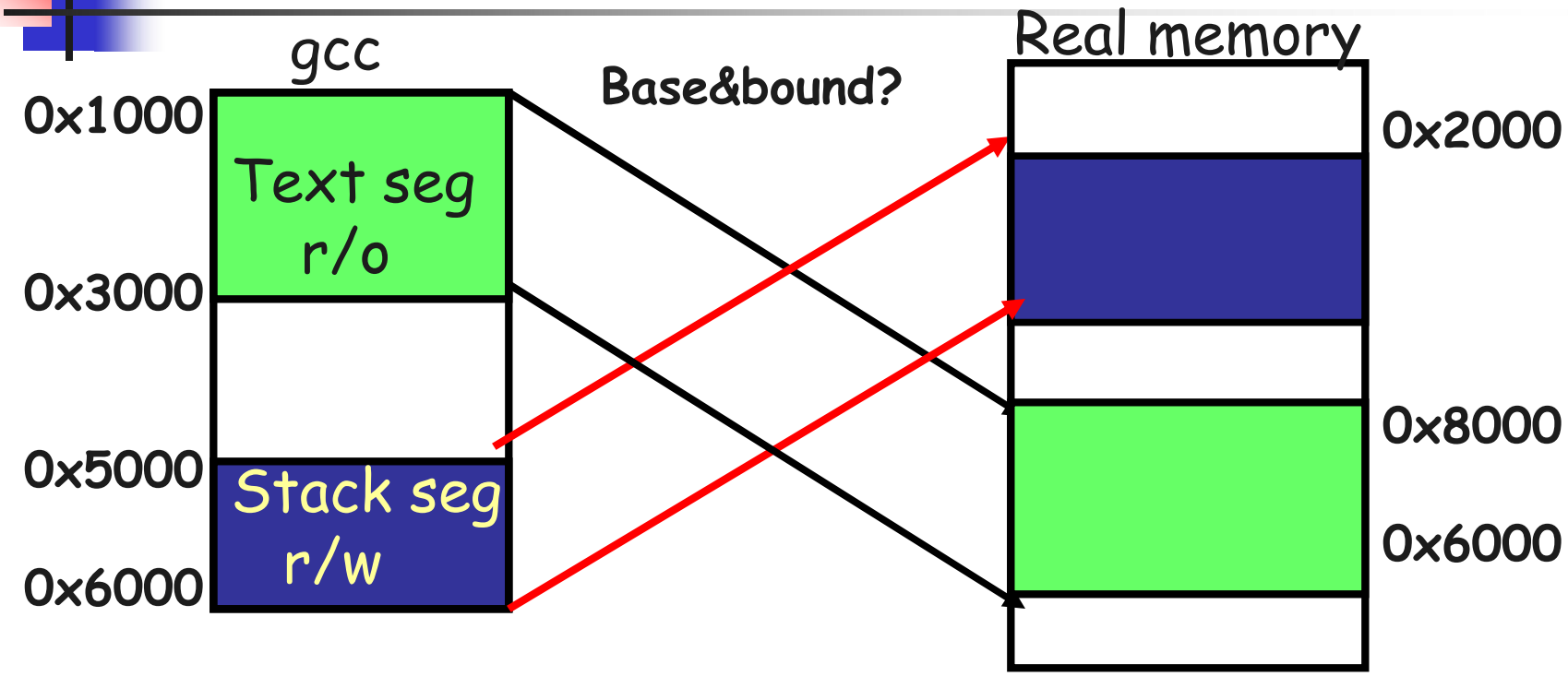
- Tiến trình tăng trưởng ? Vẫn là vấn đề cấp phát liên tục !
- Chia sẻ ?
- Phân biệt code và data ?



BÀI 9 : PHÂN ĐOẠN VÀ PHÂN TRANG

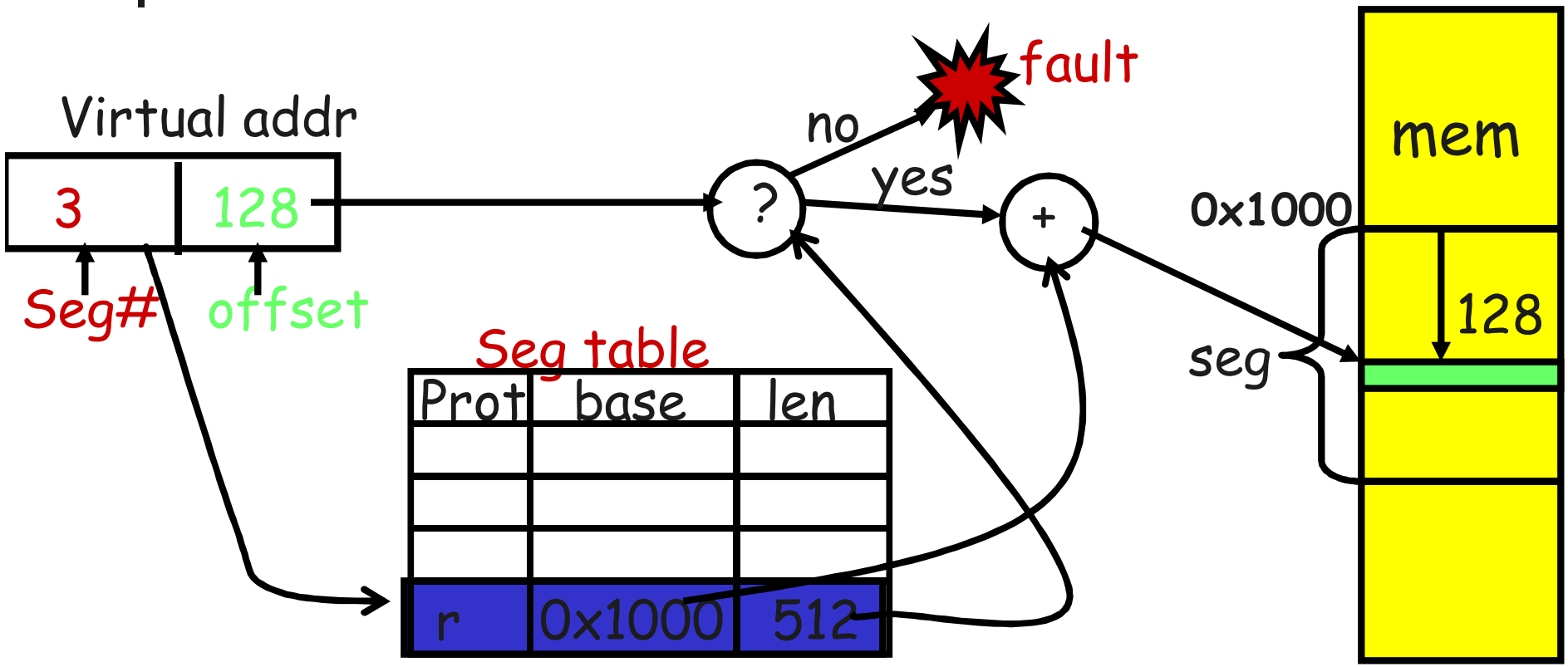
- Cấp phát không liên tục :
  - Segmentation
  - Paging

Mô hình Segmentation



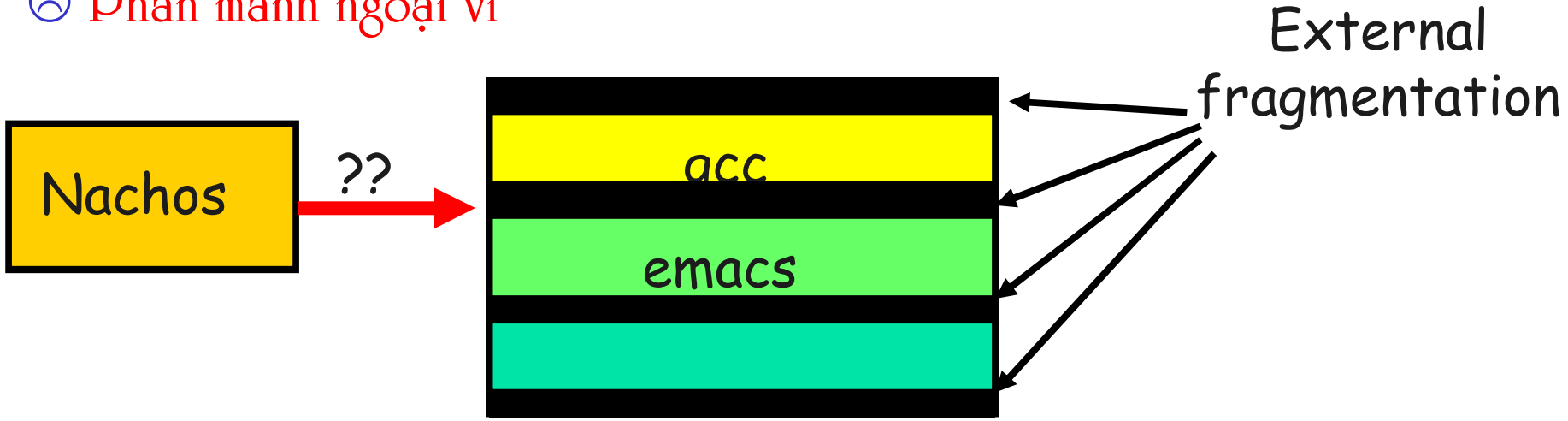
- Tiến trình gồm nhiều segment, áp dụng base bound cho từng segment
- Phân chia không gian địa chỉ thành các segment ?

Chuyển đổi địa chỉ trong mô hình Segmentation

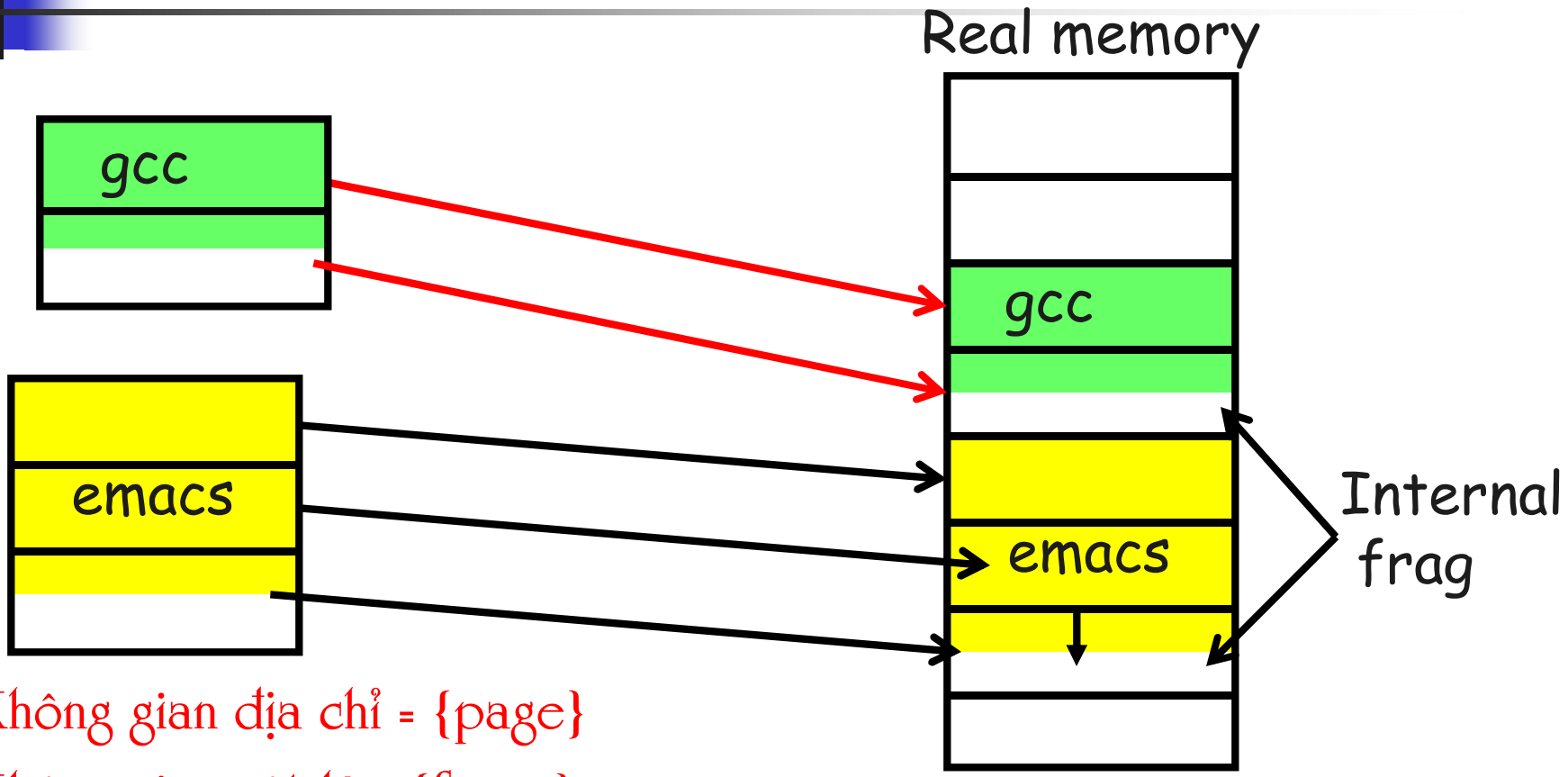


Mô hình Segmentation

- Cấp phát không liên tục => tận dụng bộ nhớ hiệu quả
- Chia sẻ được ở mức module
- ☹ Chuyển đổi địa chỉ phức tạp
- ☹ Vấn đề "Cấp phát động" : lựa chọn vùng nhớ liên tục cho 1 segment ?
- ☹ Phân mảnh ngoại vi

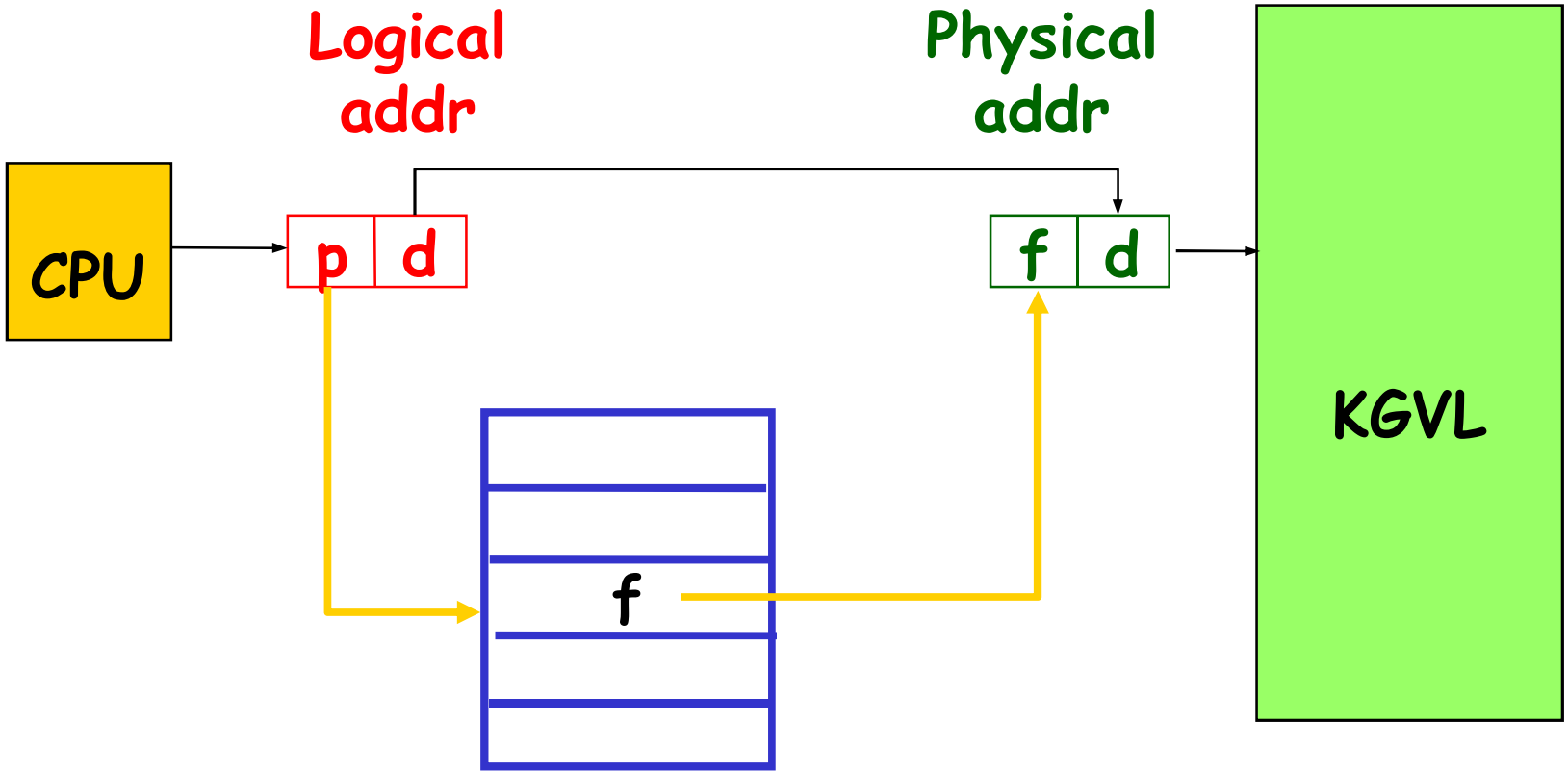


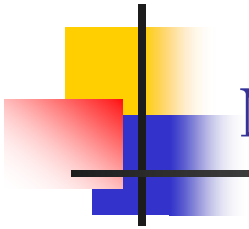
Mô hình Paging



- Không gian địa chỉ = {page}
- Không gian vật lý = {frame}
- Kích thước trang ?

Chuyển đổi địa chỉ trong mô hình Paging





Mô hình Paging

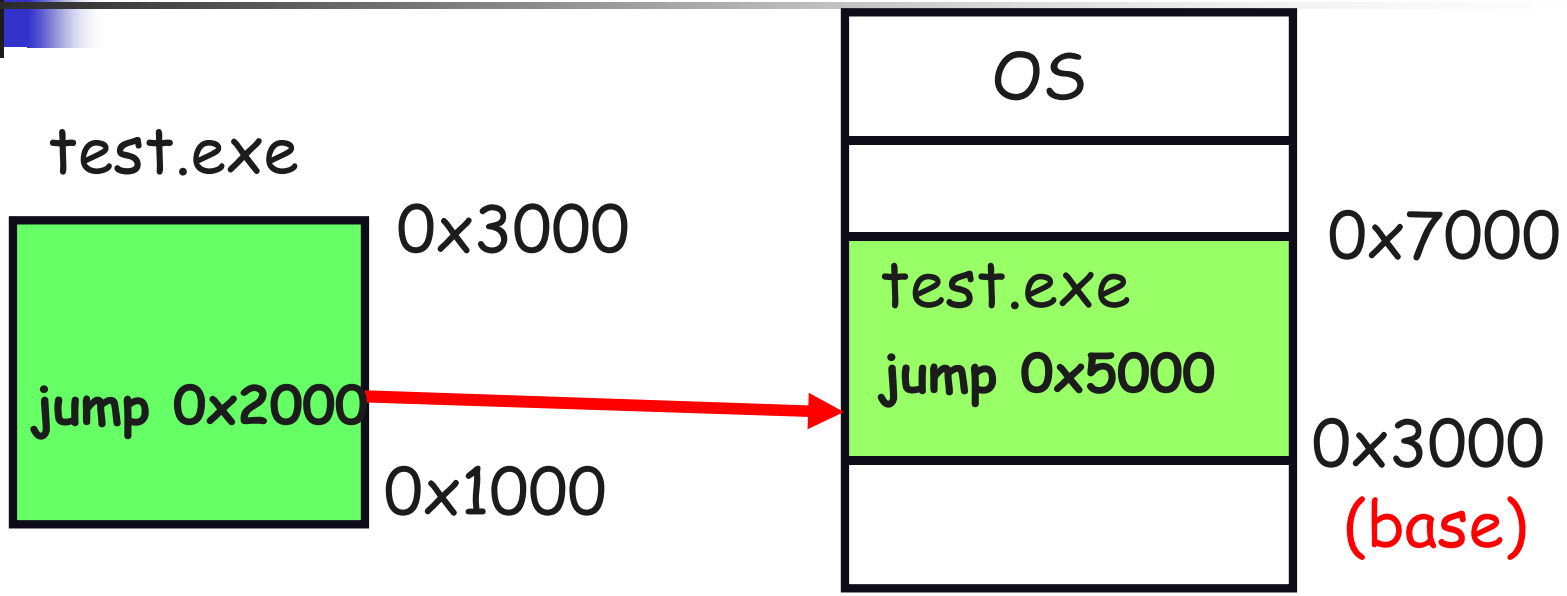
- Cấp phát bộ nhớ đơn giản
- Không còn phân mảnh ngoại vi

☹ Không chia sẻ ở mức module

☹ Phân mảnh nội vi

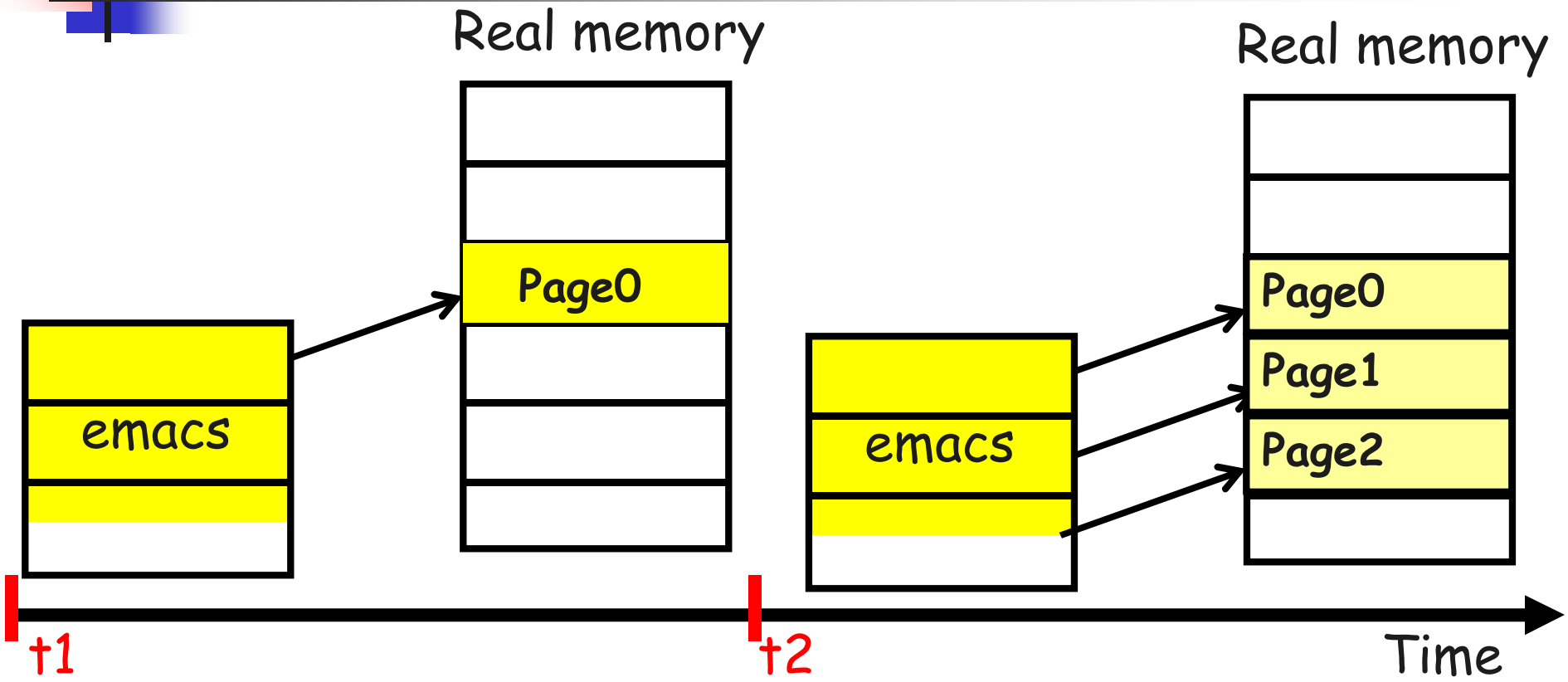
☹ Lưu trữ bảng trang ?

BÀI 10 : BỘ NHỚ ẢO



- Cho đến nay : Nạp toàn bộ tiến trình vào bộ nhơ rồi thực hiện nó...
  - Chậm, lãng phí bộ nhớ
  - Nếu kích thước tiến trình lớn hơn dung lương bộ nhớ chính ?
  - Lưu ý : tại 1 thời điểm chỉ có một chỉ thị được thực hiện

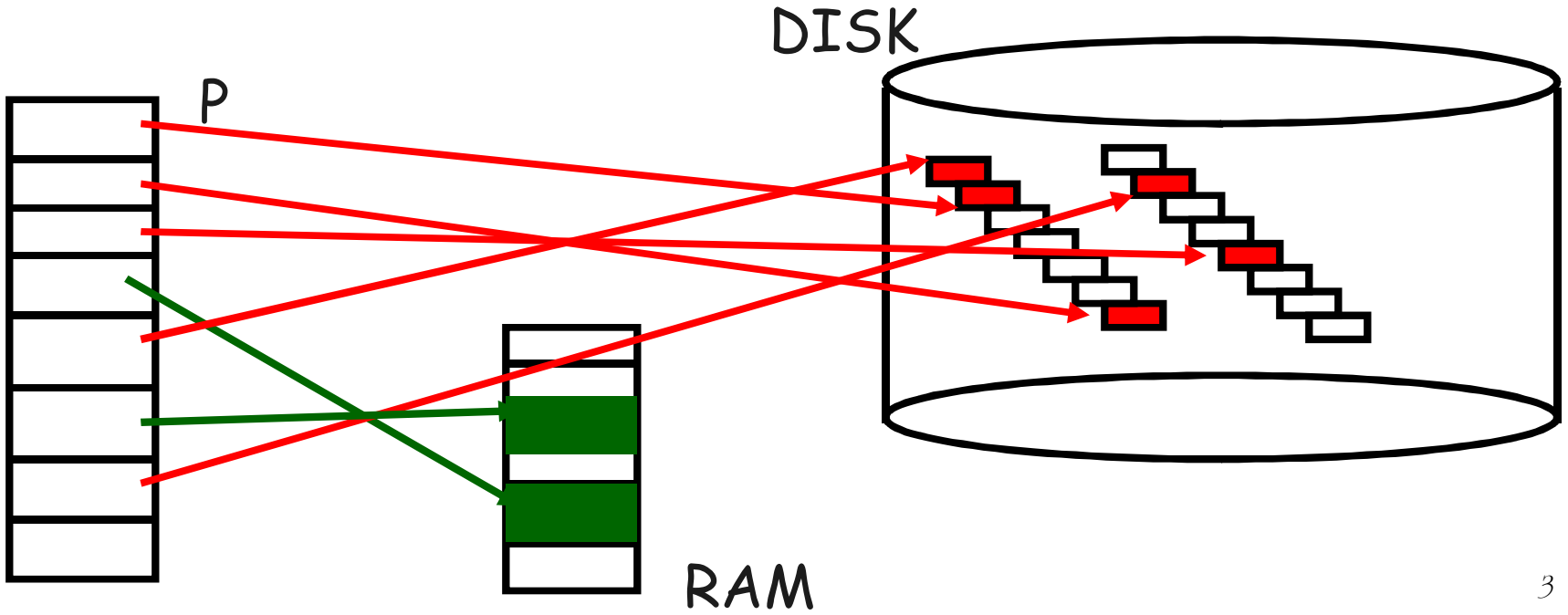
Giải pháp



- Nạp từng phần chương trình khi cần thiết
- Demand paging

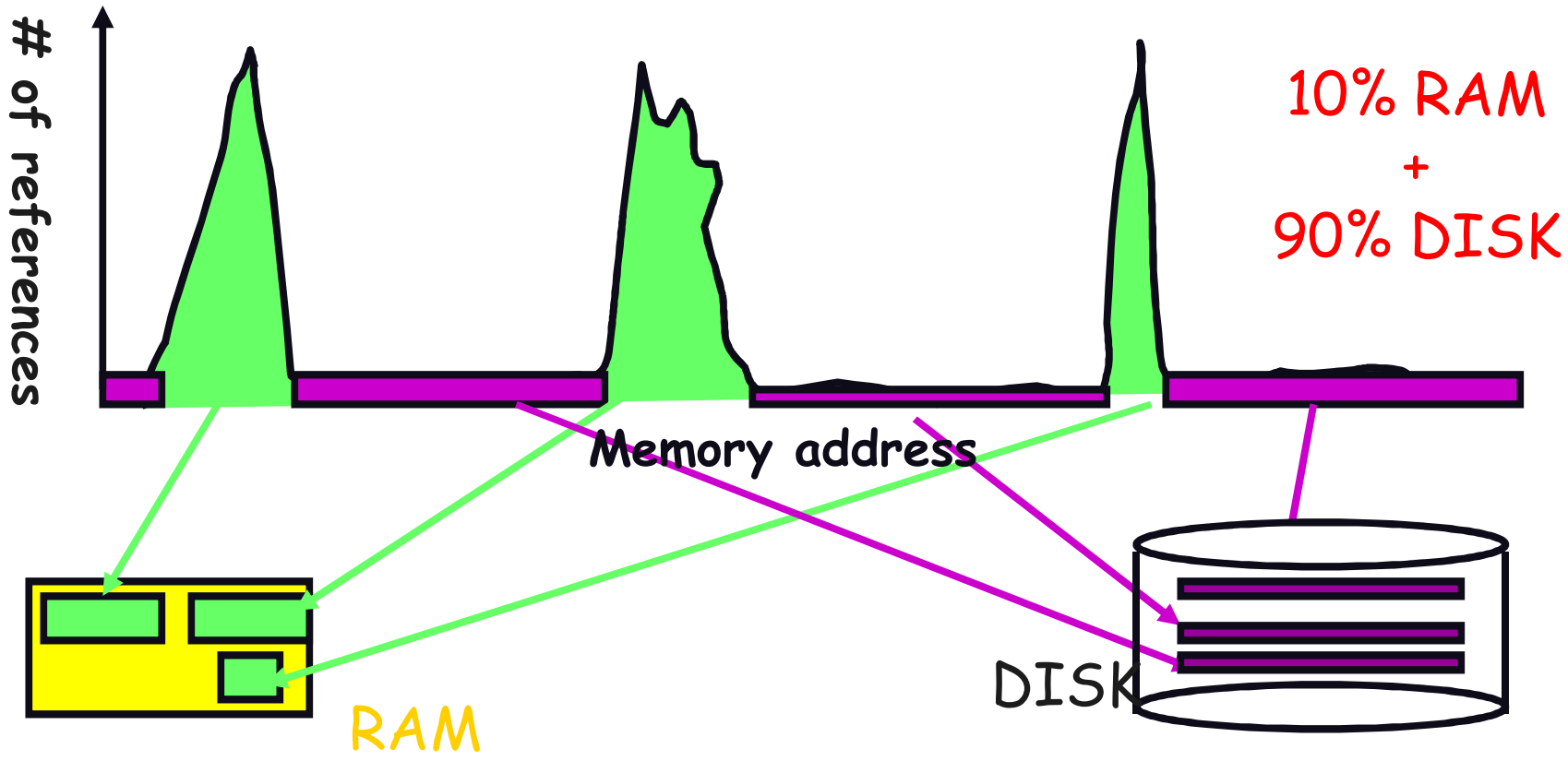
Cơ chế

- Sử dụng bộ nhớ phụ để lưu trữ tạm thời các trang chưa sử dụng
- Ai chịu trách nhiệm chuyển đổi ?
  - Lập trình viên : Overlay
  - Hệ điều hành : Bộ nhớ ảo (Virtual Memory)



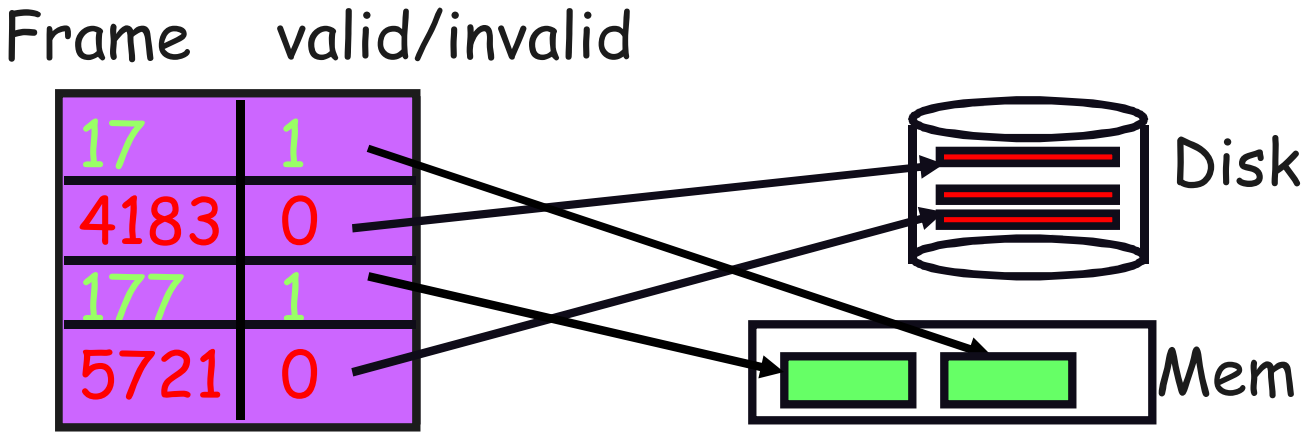
Bộ nhớ ảo = "lời nói dối vĩ đại"

- Người dùng : sở hữu bộ nhớ "vô hạn", "riêng biệt"
- Hệ điều hành : "thầm lặng" thực hiện quá trình swapping



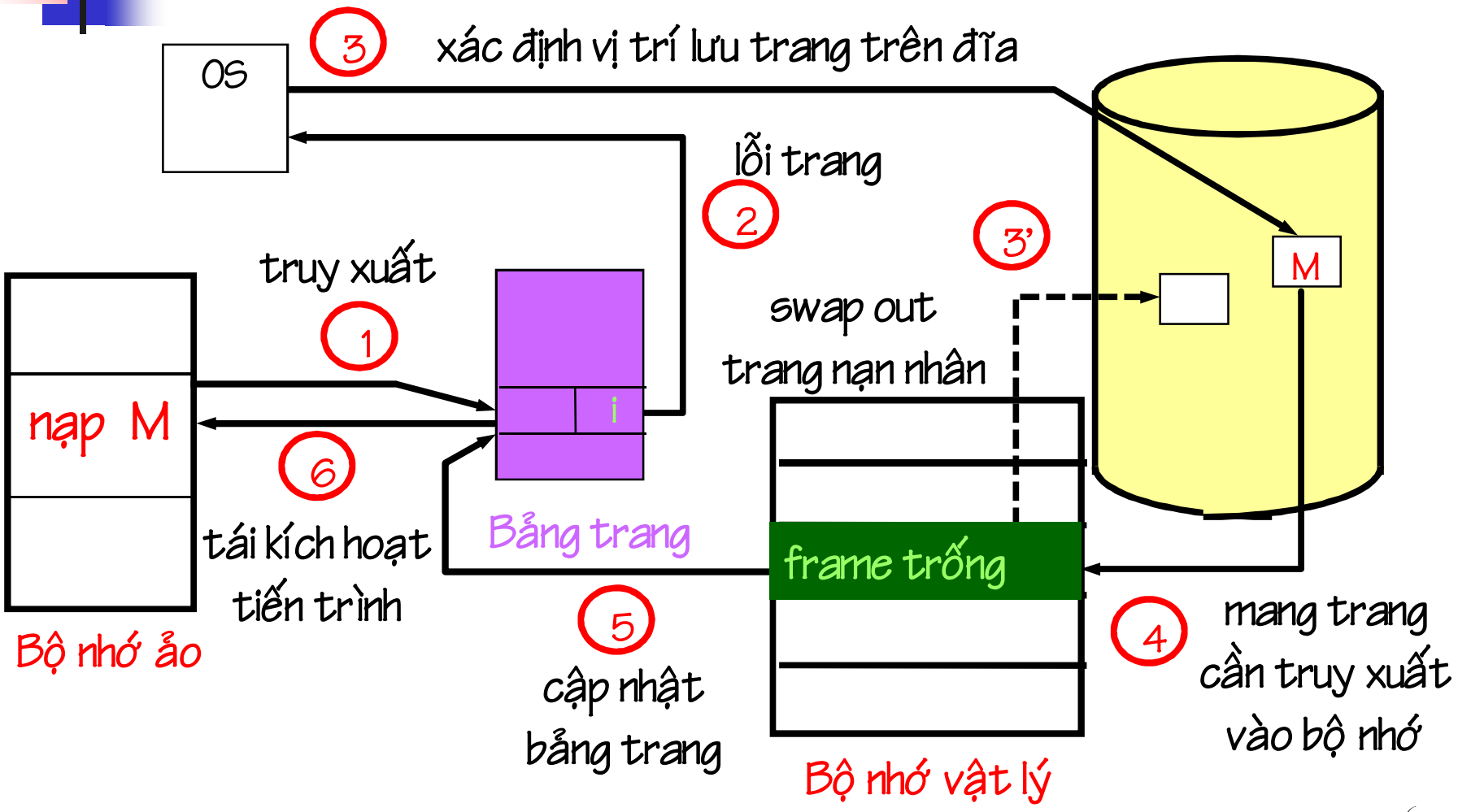
Thực hiện Bộ nhớ ảo

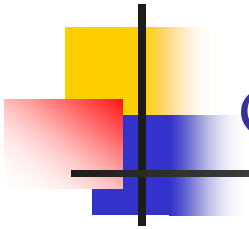
- Bảng trang : thêm 1 bit valid/invalid để nhận diện trang đã hay chưa được nạp vào RAM



- Truy xuất đến một trang chưa được nạp vào bộ nhớ : lỗi trang (page fault)

Xử lý lỗi trang





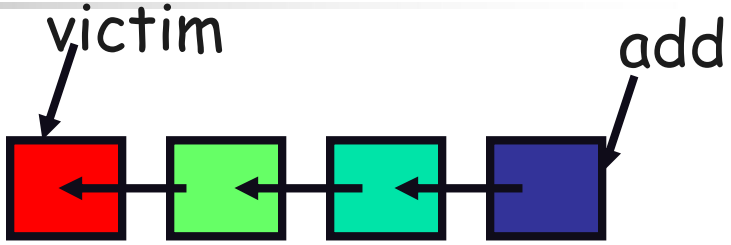
Các câu hỏi

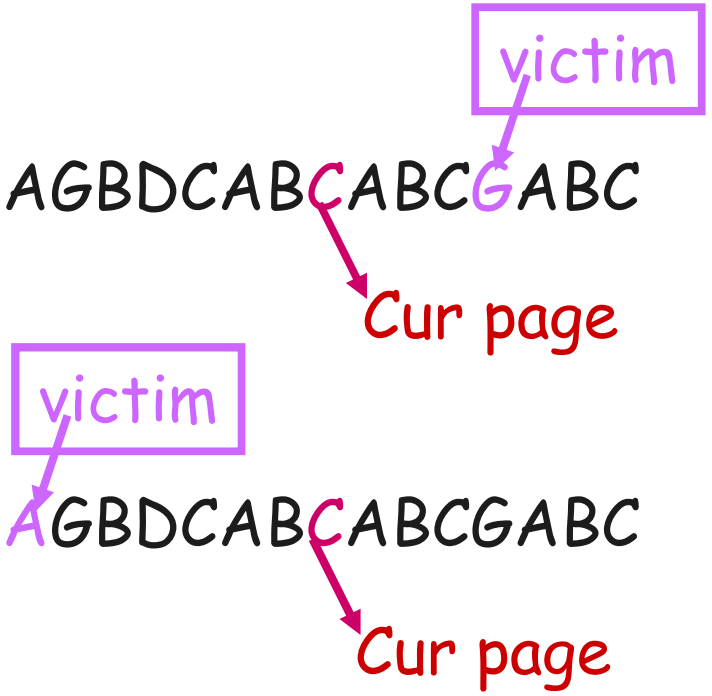
1. Chọn trang nạn nhân ? => Chiến lược thay thế trang

2. Chọn trang nào để nạp ? => Chiến lược nạp

Chiến lược thay thế trang

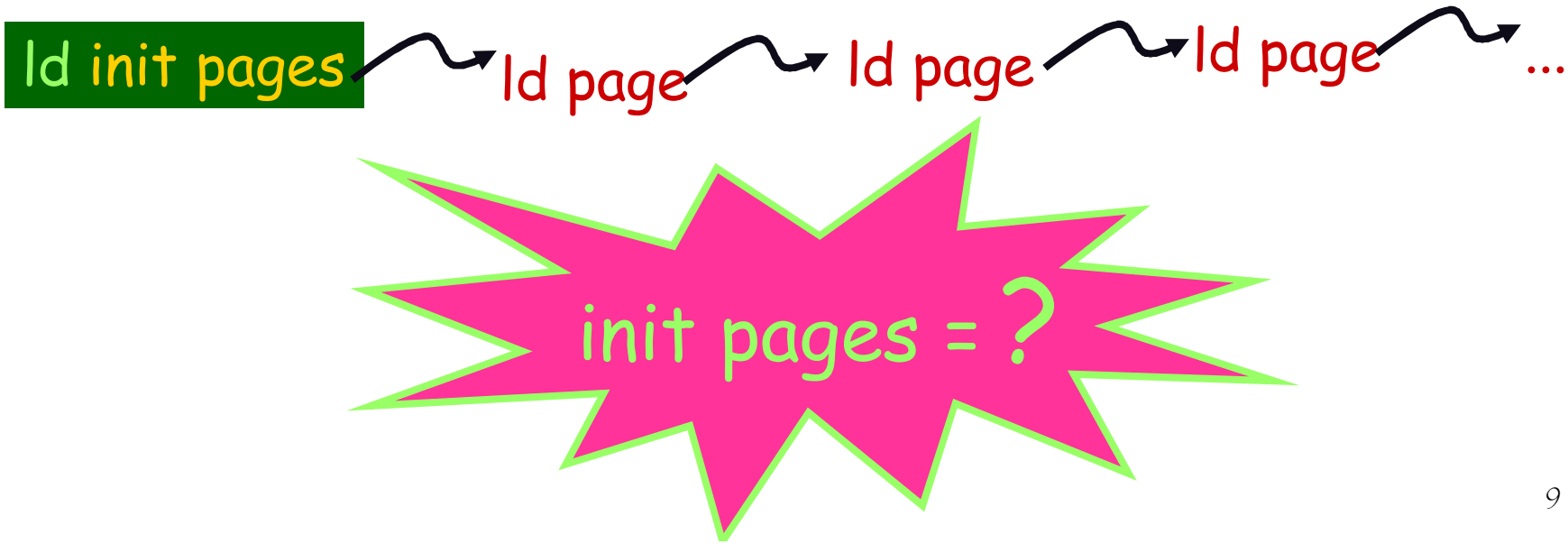
- FIFO: trang "già" nhất
  - Công bằng ?
  - Không xét đến tính sủ dụng !
- TỐI ƯU : trang lâu sử dụng đến nhất trong tương lai
  - Tần suất lỗi trang thấp nhất
  - Không khả thi !
- LRU :trang lâu nhất chưa sử dụng đến trong quá khứ
  - Dự đoán tương lai LRU = MIN ?





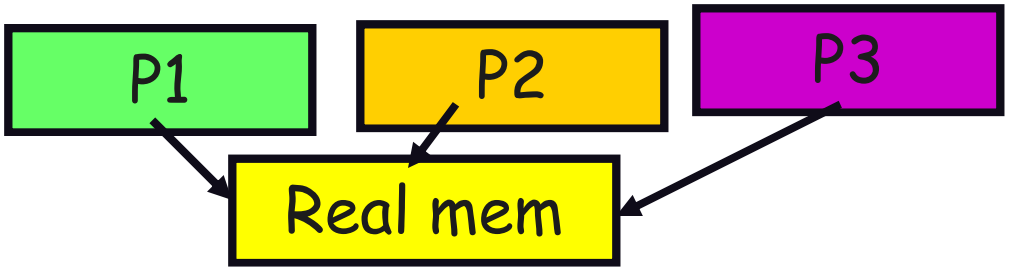
Chiến lược nạp

- Demand paging : nạp trang được yêu cầu
  - Khi nào ?
  - Nạp sau : tần suất lỗi trang cao ? => pure demand paging
  - Nạp trước : làm sao biết ? =>prepaging

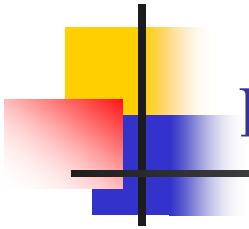


Thrashing = ảo tưởng sụp đổ !

- Các tiến trình trong hệ thống yêu cầu bộ nhớ nhiều hơn khả năng cung cấp của hệ thống !



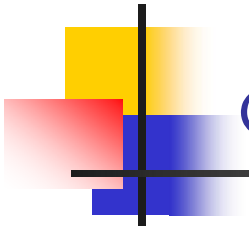
- Tất cả tiến trình đều bận rộn xử lý lỗi trang !
- IO hoạt động 100 %, CPU rảnh !
- Hệ thống ngừng trệ



Nguyên nhân Thrashing

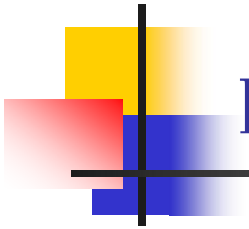
1. Tiến trình không tái sử dụng bộ nhớ (quá khứ != tương lai)
2. Tiến trình tái sử dụng bộ nhớ, nhưng với kích thước lớn hơn
3. Quá nhiều tiến trình trong hệ thống

- Chỉ có thể kiểm soát thrashing do nguyên nhân 3.



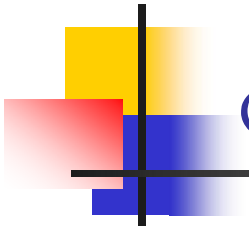
Giải quyết thrasing với mô hình Working set

- Working set = tập hợp các trang tiến trình đang truy xuất tại 1 thời điểm.
- Hệ điều hành :
  - Chỉ nạp một tiến trình khi có đủ khung trang tự do cho working set của nó.
  - Kiểm soát mức độ đa chương của hệ thống : Nếu tổng số khung trang yêu cầu của các tiến trình trong hệ thống vượt quá các khung trang có thể sử dụng, chọn một tiến trình để tạm dừng, ngược lại, khi tổng working set bé hơn số khung trang tự do, nạp thêm tiến trình.



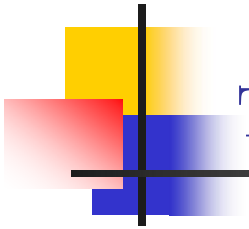
BÀI 11 : An toàn và bảo vệ hệ thống

- An toàn hệ thống (security):
  - Bảo vệ cái gì ?
  - Chiến lược ?
- Bảo vệ hệ thống (protection)
  - Cơ chế kỹ thuật hỗ trợ thiết lập an toàn hệ thống



Các mối nguy hiểm

- Truy xuất bất hợp lệ
  - thâm nhập
  - thao tác lạm quyền
- "Núp bóng" truy xuất hợp lệ để phá hoại
  - "trojan horse
- "Kẻ xấu thật sự"
  - virus
  - worm



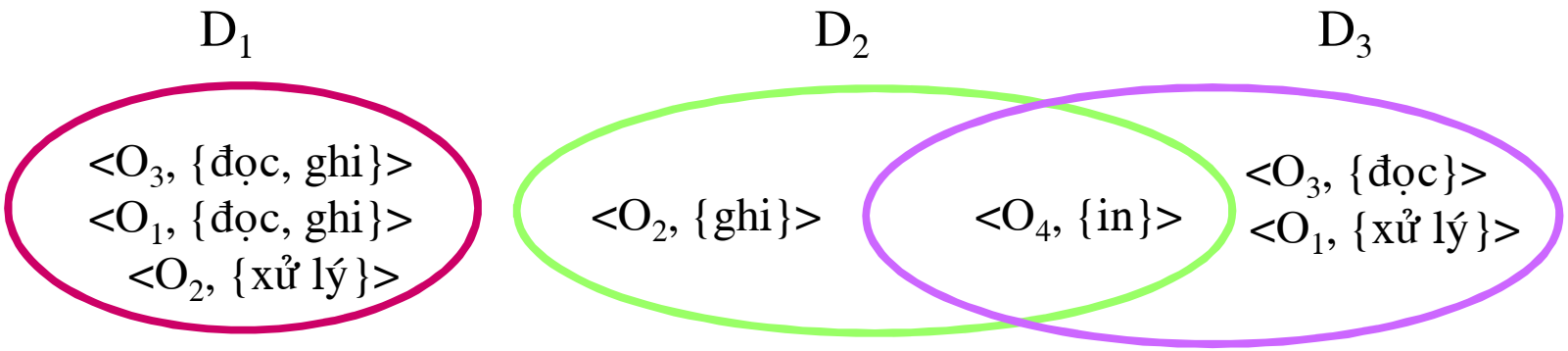
Thiết lập an toàn cho hệ thống

- Kiểm định danh tính (Authentication)
  - Xác định quyền hạn của *người dùng(authorized)*
  - password ?
- Sử dụng cơ chế nào để thực hiện các chiến lược kiểm tra an toàn?

Thuật ngữ

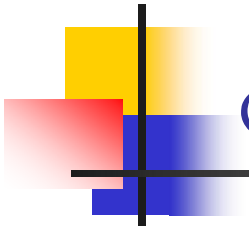
- *objects* : đối tượng cần được kiểm soát truy xuất
- *rights* : **Các khả năng thao tác trên một đối tượng**
- *domains* : **tập các quyền truy xuất,**

quyền truy xuất = <*đối tượng, {quyền thao tác}* >.



Ma trận quyền truy xuất

object / domain	F_1	F_2	F_3	Máy in
D_1	đọc		đọc	
D_2				in
D_3		đọc	xử lý	
D_4	đọc ghi		đọc ghi	



Các cơ chế bảo vệ

- Cài đặt ma trận quyền truy xuất :
  - Access Control List:
    - Mỗi Object cómột ACL <domains,rights>
  - Capabilities
    - Mỗi Domain có một capabilities <objects,rights>