# HỆ ĐIỀU HÀNH

## BÀI 4: QUẢN LÝ TIẾN TRÌNH

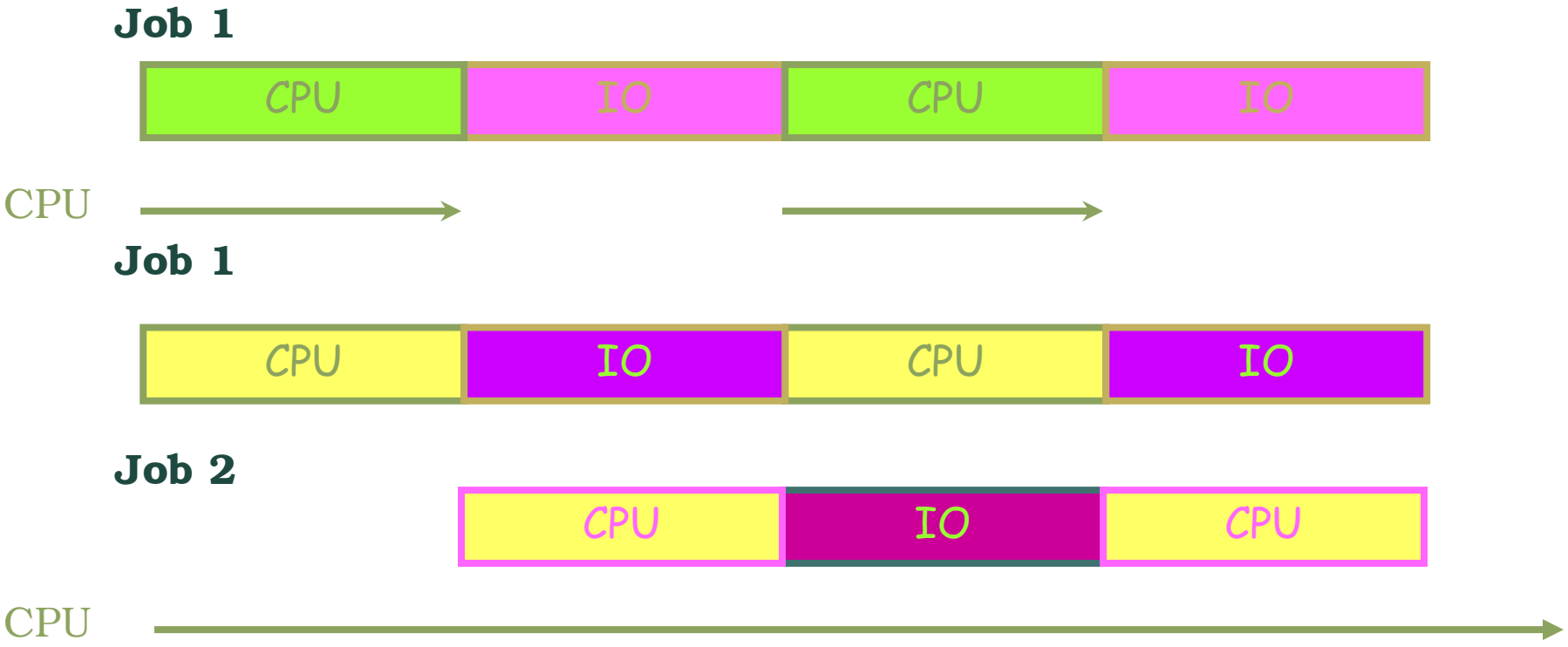
Ths. Lê Viết Long

# Nội dung

- Mô hình Tiến trình
- Trạng thái tiến trình
- Thông tin quản lý tiến trình
- Quá trình điều phối tiến trình
- Các thuật toán điều phối

# Đa nhiệm và đa chương

- Vì sao muốn xử lý đồng thời nhiều công việc trên máy tính ?

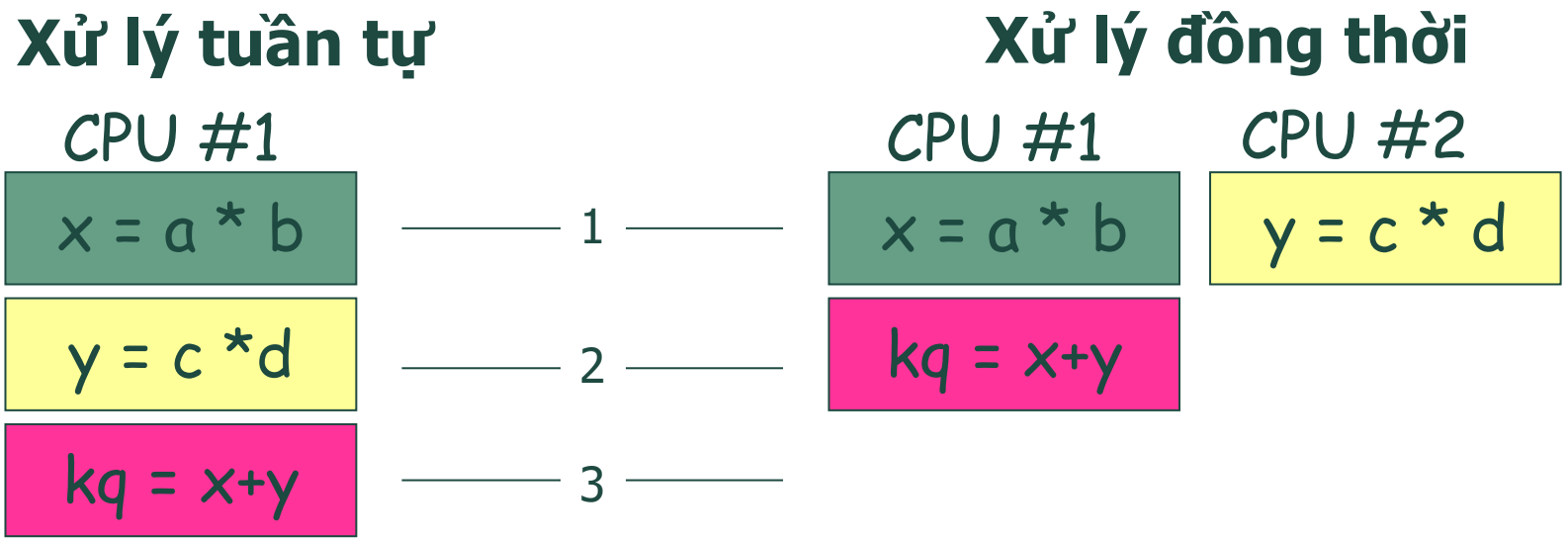



→ **Xử lý đồng thời để tăng hiệu suất sử dụng CPU**

# Đa nhiệm và đa chương ???

- Vì sao muốn xử lý đồng thời nhiều công việc trên máy tính ?

**Job : kq = a*b + c*d;**

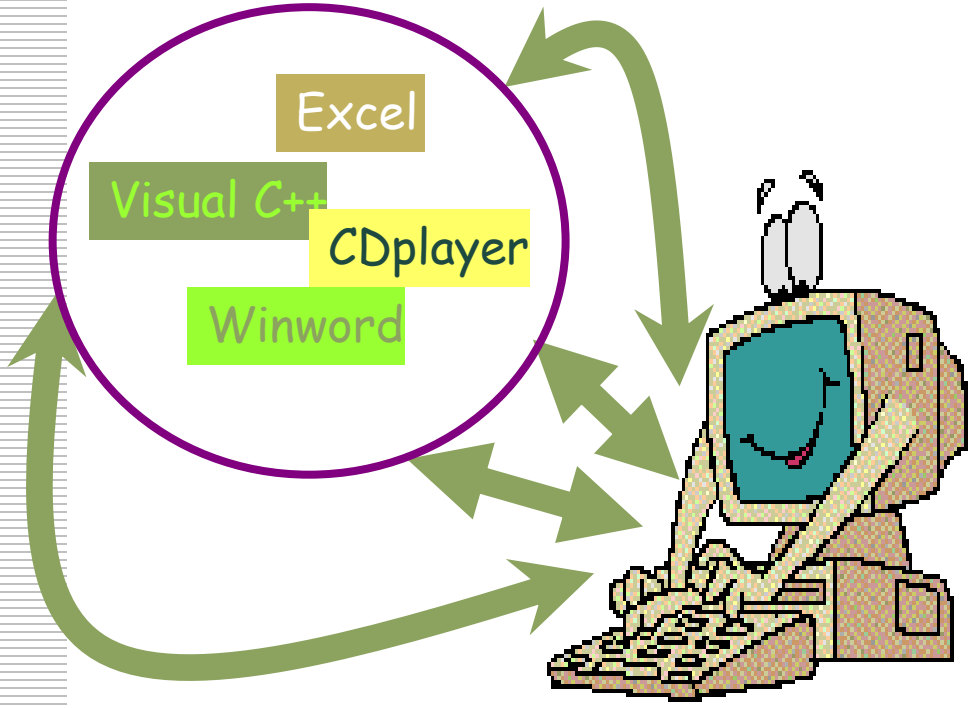



**→ Xử lý đồng thời để tăng tốc độ xử lý**

# Đa nhiệm và đa chương ???

- Multitasking (đa nhiệm): cho phép nhiều tác vụ/ công việc được xử lý đồng thời
- Người dùng luôn mong muốn 1 HĐH đa nhiệm
- Nhưng: Máy tính thường chỉ có 1 CPU?
- Multiprogramming (đa chương): kỹ thuật cho phép nhiều chương trình được thực hiện đồng thời (trên 1 CPU)
- Giả lập nhiều CPU ảo từ 1 CPU thật để cho phép thi hành nhiều chương trình đồng thời.
- Ảo hoá bằng cách nào? Xây dựng các thuật toán để luân chuyển CPU giữa các chương trình ứng dụng.

# Xử lý đồng hành, những khó khăn ?




**Tài nguyên giới hạn, ứng dụng "vô hạn"**

**Nhiều hoạt động đan xen**

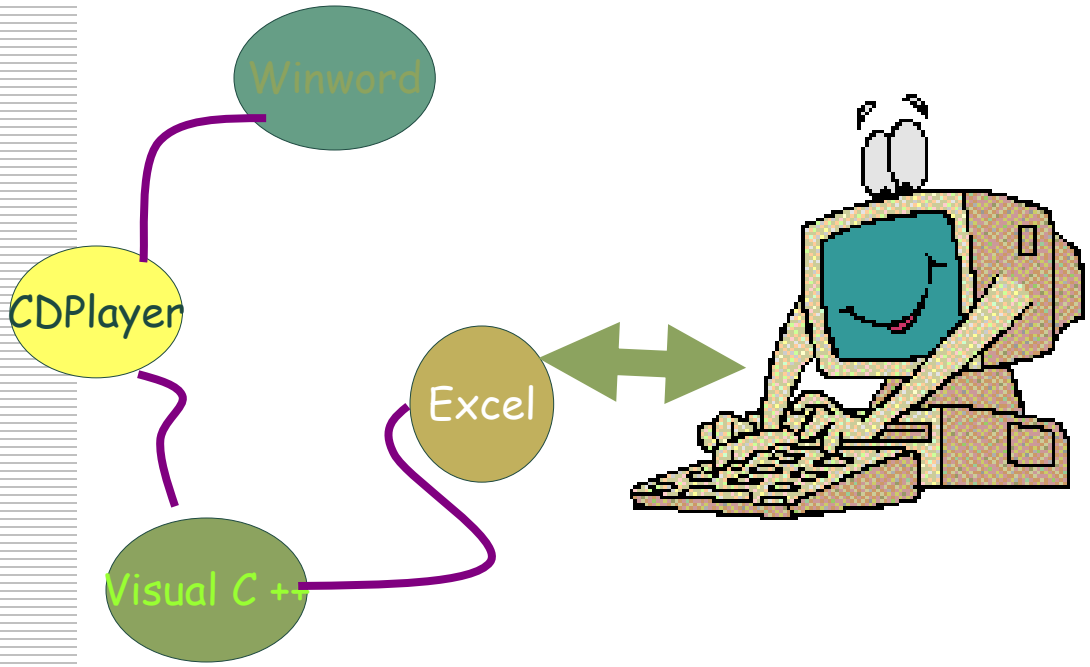
**??? Phân chia tài nguyên ?**

**??? Chia sẻ tài nguyên ?**

**??? Bảo vệ?**

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

# Giải pháp




-"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

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

# Giải pháp

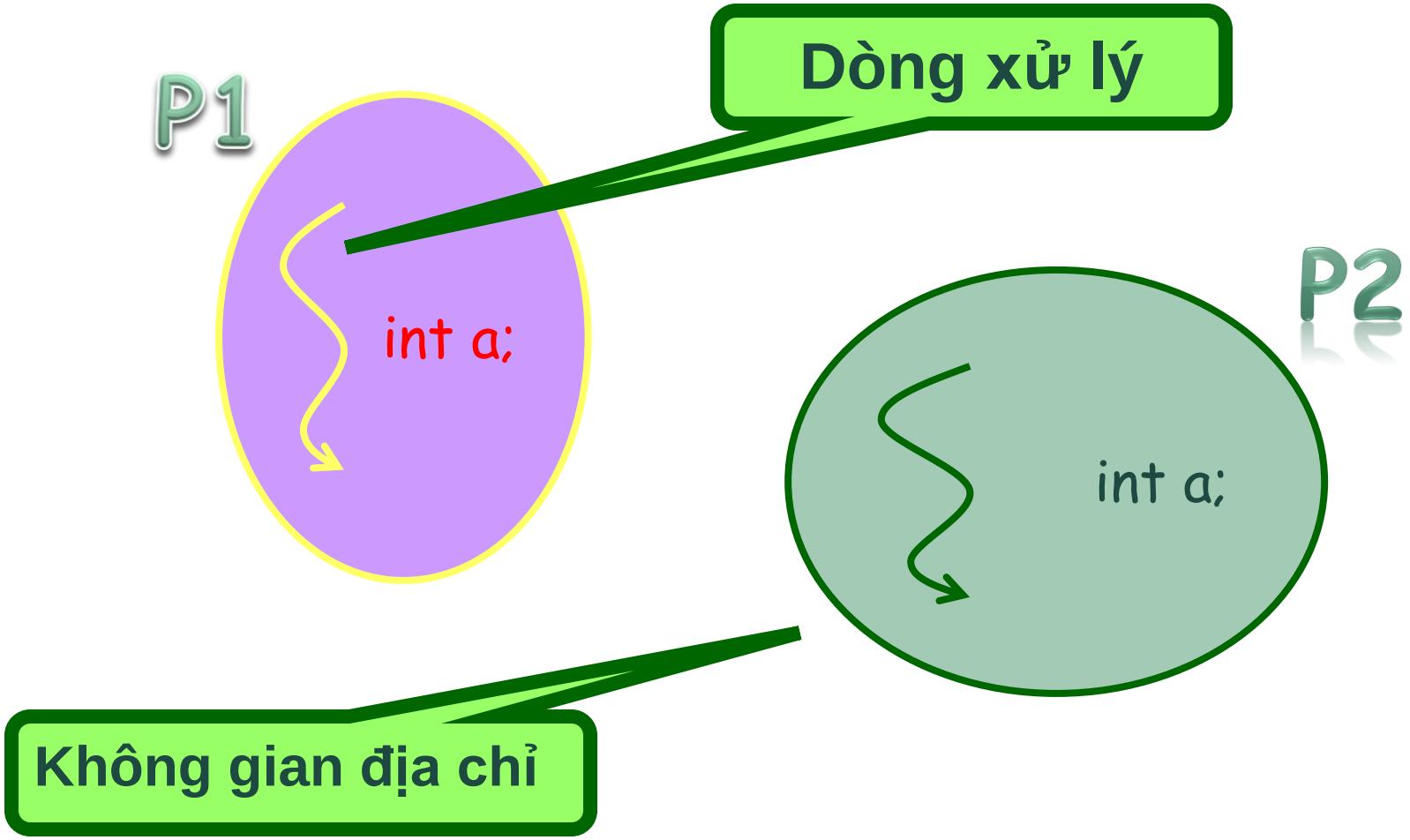


CPU


# Khái niệm tiến trình (Process)

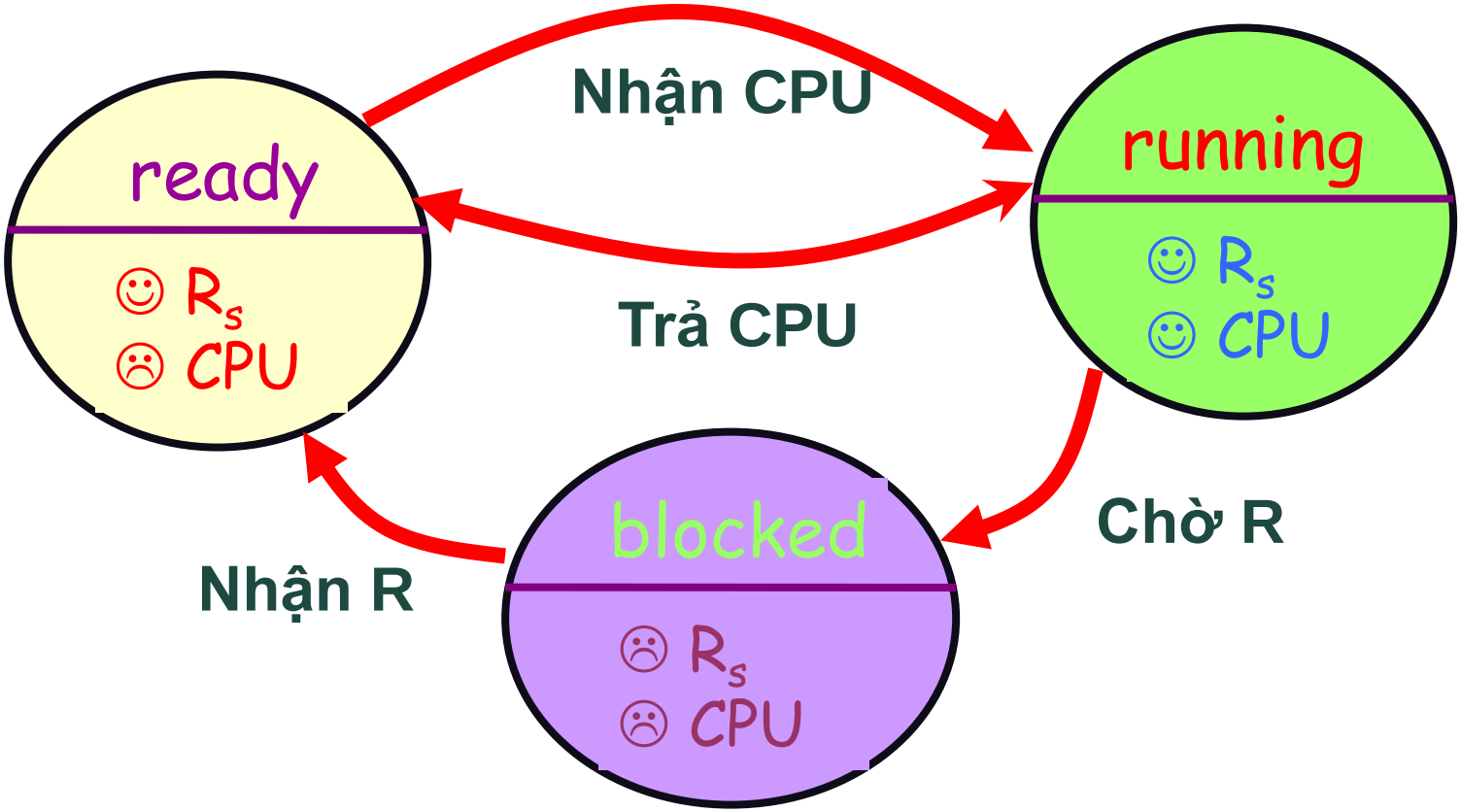
- Tiến trình là một chương trình đang trong quá trình thực hiện
- Mỗi tiến trình sở hữu
- Một CPU (ảo) riêng
- Một không gian nhớ riêng
- Chiếm giữ 1 số tài nguyên của hệ thống
- Vd: Một chương trình Word có thể được chạy 2 lần sẽ tạo ra 2 tiến trình khác nhau:
  - Microsoft Word – [Bai tap1.doc]
  - Microsoft Word – [Bai tap2.doc]

# Hai phần của tiến trình



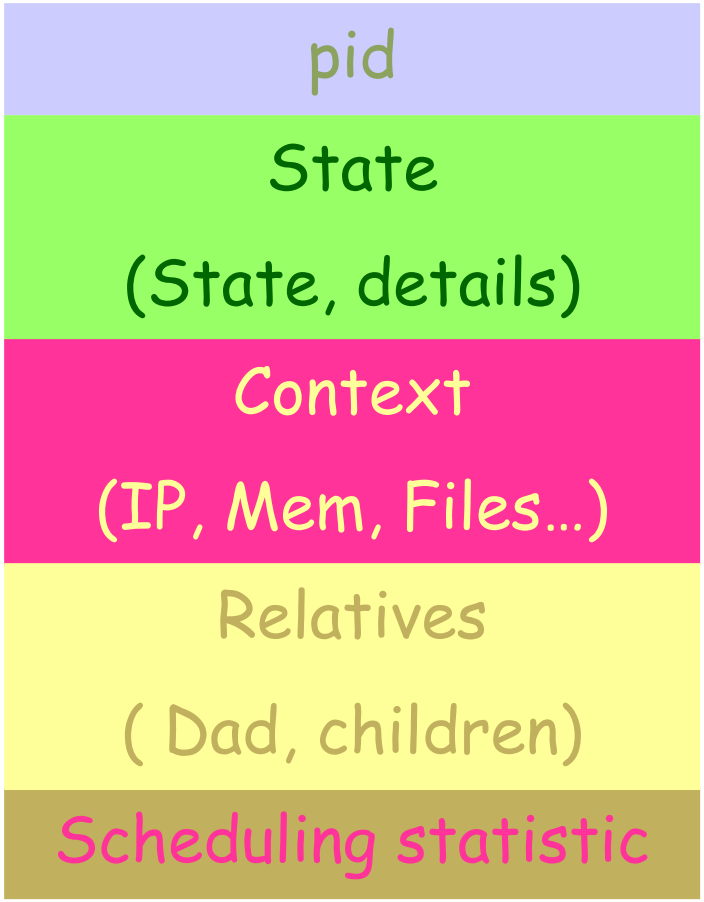
# Trạng thái tiến trình?

Tại 1 thời điểm, tiến trình ở một trong các trạng thái sau:



# Khối quản lý tiến trình

- 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
  - Thời gian sử dụng CPU
  - Thời gian chờ




Process control Block
PCB

# Khối quản lý tiến trình – Mach OS

```
typedef struct machpcb    {
char    mpcb_frame[REGOFF];
struct  regs mpcb_regs;    // user's saved registers
struct  rwindow mpcb_wbuf[MAXWIN]; //user window save buffer
char    *mpcb_spbuf[MAXWIN]; //sp's for each wbuf
int     mpcb_wbcnt;  //number of saved windows in pcb_wbuf
struct  v9_fpu *mpcb_fpu;          // fpu state
struct  fq mpcb_fpu_q[MAXFPQ];     // fpu exception queue
int     mpcb_flags;                // various state flags
int     mpcb_wocnt;                // window overflow count
int     mpcb_wucnt;                // window underflow count
kthread_t *mpcb_thread;            // associated thread
} machpcb_t;
```

**Khối quản lý tiến trình của HĐH MachOS**

# Các thao tác trên tiến trình

- Tạo lập tiến trình
- Kết thúc tiến trình
- Thay đổi trạng thái tiến trình :
  - Assign()
  - Block()
  - Awake()
  - Suspend()
  - Resume()

# Tạo lập tiến trình

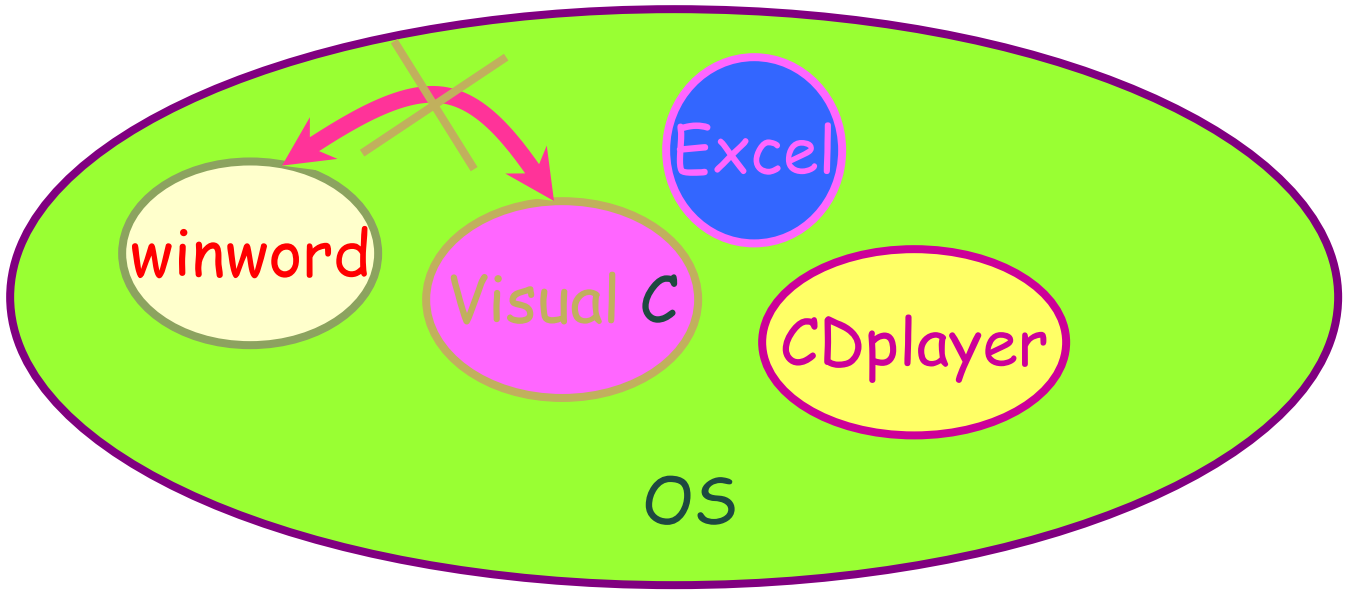
- Các tình huống :
- Khởi động batch job
- User logs on
- Kích hoạt 1 service (print...)
- Process gọi hàm tạo một tiến trình khác
  - Các tiến trình có thể tạo tiến trình con, hình thành cây tiến trình trong hệ thống
  - Các tiến trình mới được tạo có thể thừa hưởng tài nguyên từ cha, hay được cấp tài  nguyên mới

# Kết thúc tiến trình

- Tình huống :
- Tiến trình xử lý xong  lệnh cuối cùng  hay gọi exit ()
- Kết thúc Batch job , *Halt*  instruction
- User logs off
- Do lỗi chương trình
- Một tiến trình có thể kết thúc 1 tiến trình khác nếu có ID (định danh) của tiến trình kia.
- Ví dụ: kill –-s SIGKILL 1234: huỷ tiến trình có ID là 1234

# 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..



# Ví dụ mô hình đa tiến trình

- Giờ thi lý thuyết môn Hệ Điều hành
- Mỗi sinh viên là một tiến trình :
  - Cùng làm bài  => Hoạt động đồng hành
  - Có bài thi , bút, giấy…riêng => Tài nguyên riêng biệt
  - Độc lập  => Không trao đổi  (về nguyên tắc)
- Thực hành môn Hệ Điều hành
  - 2 sinh viên/nhóm
  - Hợp tác đồng hành
  - Nhu cầu trao đổi
  - Dùng tài nguyên chung

# Mô hình đa tiểu trình (MultiThreads)

- Nhiều tình huống cần có nhiều dòng xử lý đồng thời cùng hoạt động trong một không gian địa chỉ => cùng chia sẻ tài nguyên (server, OS, các chương trình tính toán song song : nhân ma trận…)

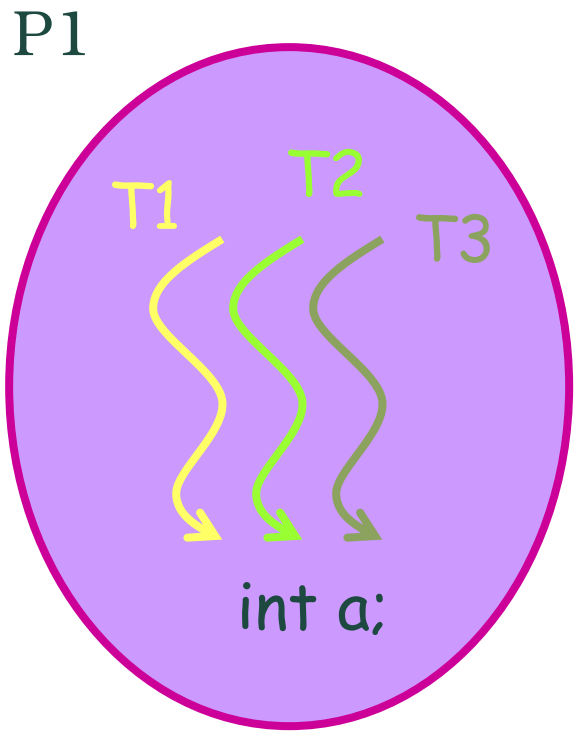

- Khái niệm mới : tiểu trình (thread)

# Ví dụ Mô hình đa tiểu trình

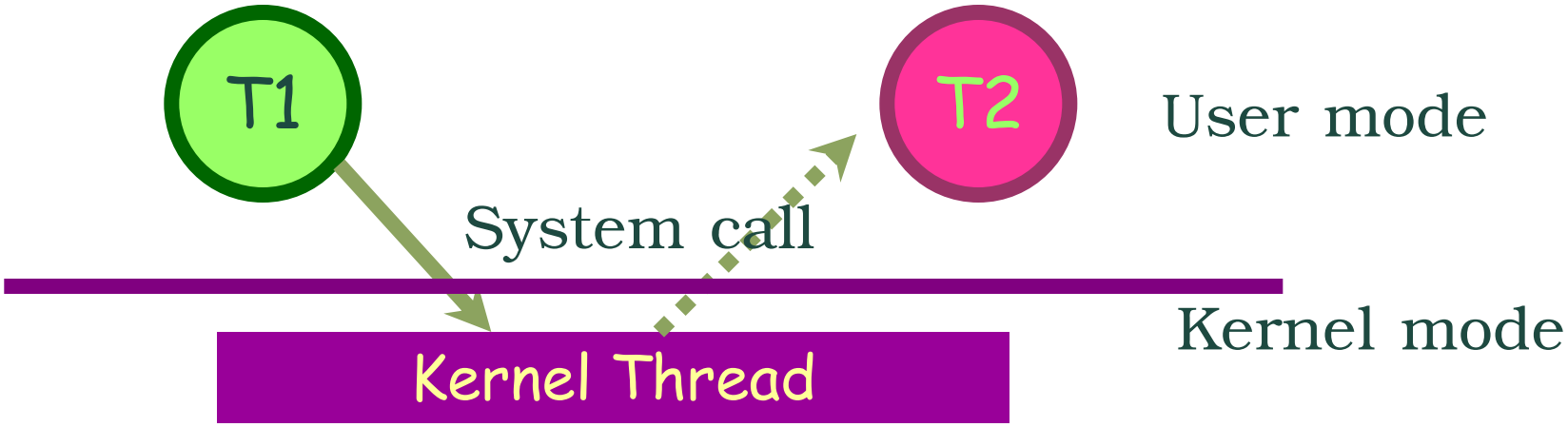
- Thực hành môn Hệ Điều hành
- Mỗi nhóm 2 sinh viên là một tiến trình :
- Mỗi sinh viên là một tiểu trình
  - Cùng làm bài  => Hoạt động đồng hành
  - Có bài thực hành chung => Tài nguyên chung
  - Trao đổi với nhau

# Tiểu trình vs 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)



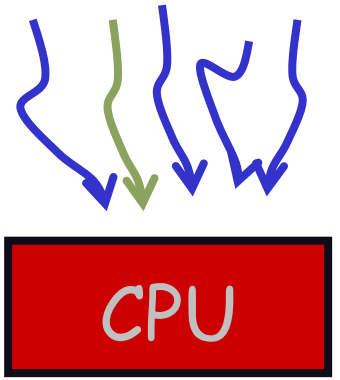

- Khái niệm tiểu trình được xây dựng bên trong hạt nhân
- Đơn vị xử lý là tiểu trình
- Ví dụ :
  - Windows 95/98/NT/2000
  - Solaris, Tru64 UNIX, BeOS, Linux

# Phân chia CPU ?

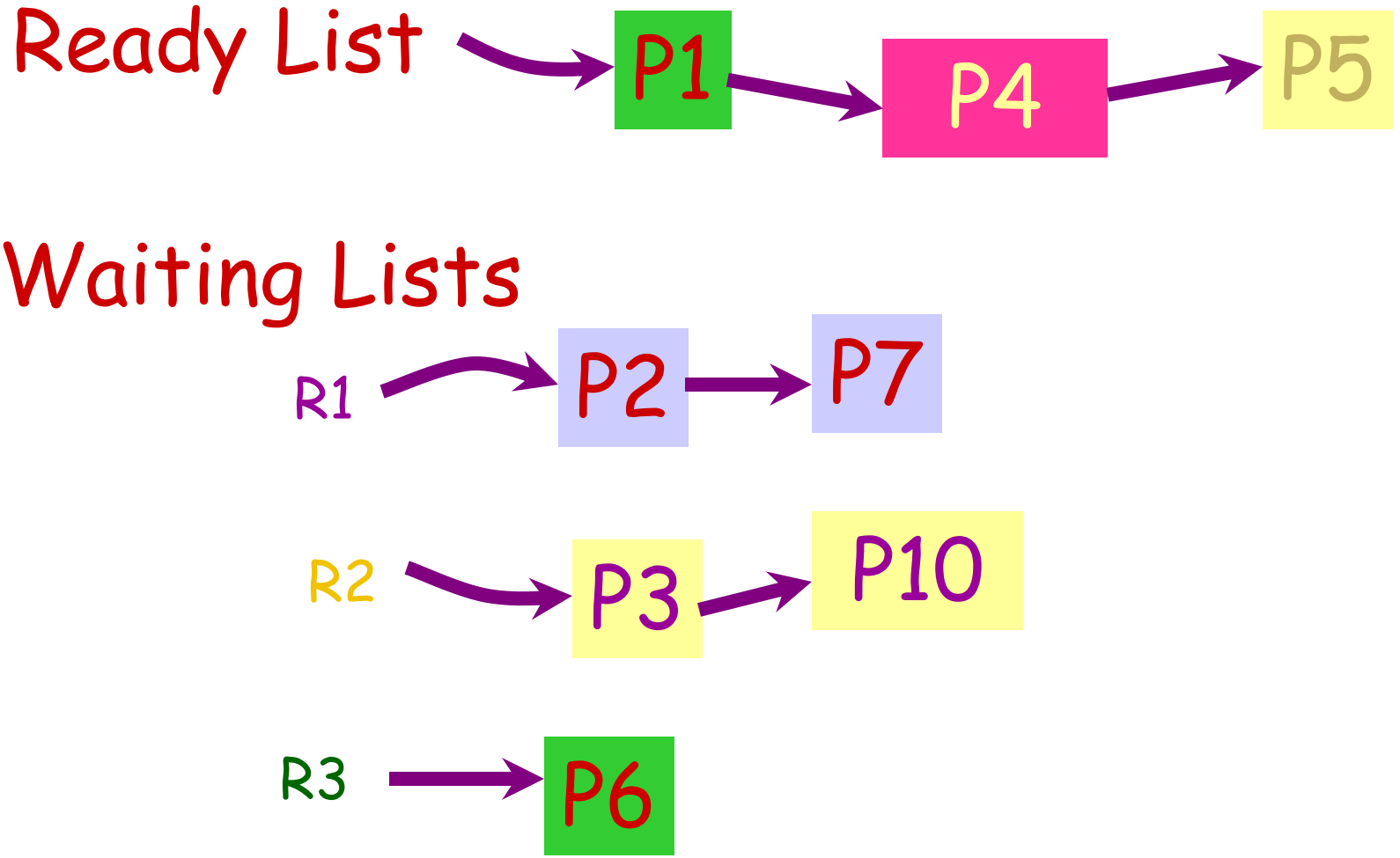
- 1 CPU vật lý : làm thế nào để tạo ảo giác mỗi tiến trình sở hữu CPU riêng của mình ?

  → Luân chuyển CPU giữa các tiến trình

- 2 thành phần đảm nhiệm vai trò điều phối:
  - Scheduler chọn 1 tiến trình
  - Dispatcher chuyển CPU cho tiến trình được chọn



# Các danh sách tiến trình



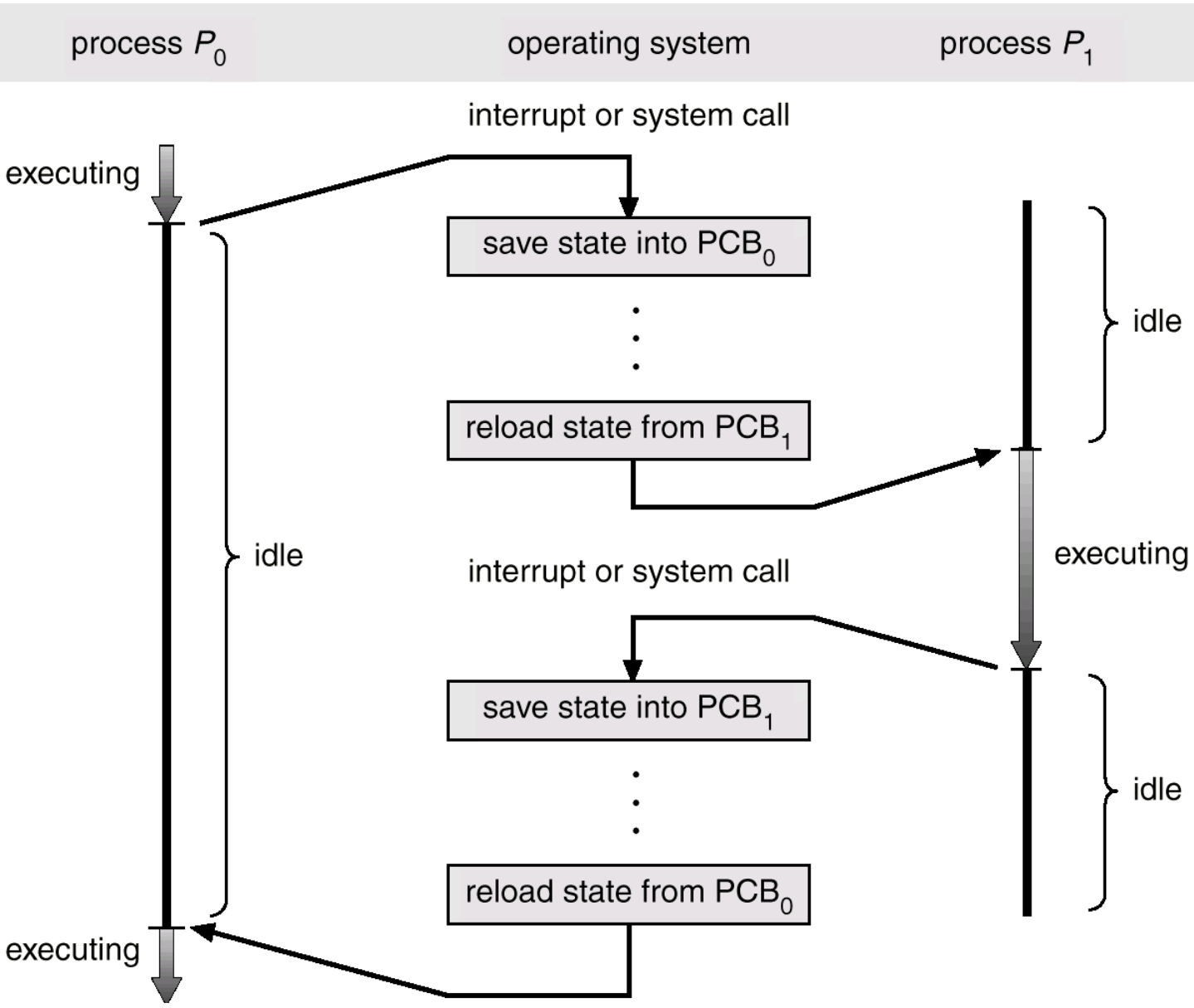
# Scheduler - Nhiệm vụ

- Ra quyết định chọn một tiến trình để cấp phát CPU :
- Ứng cử viên = {Các tiến trình ready list}
  - 0 tiến trình : CPU rảnh rỗi (idle)!
  - 1 tiến trình : không cần suy nghĩ nhiều, đúng không ?
  - >1 : chọn ai bây giờ ? ➔ Cần có thuật toán điều phối

# Dispatcher - Nhiệm vụ

- Nhiệm vụ của Dispatcher: Chuyển đổi ngữ cảnh
- Xét ví dụ
  - Tiến trình A đang dùng CPU 1 chút thì bị HĐH thu hồi CPU
  - HĐH cấp CPU cho B dùng 1 chút, HĐH thu hồi lại CPU.
  - HĐH cấp CPU trở lại cho A.

  → Giá trị các thanh ghi giữa những lần chuyển đổi CPU ?
- Kịch bản :
  - Lưu ngữ cảnh tiến trình hiện hành
  - Nạp ngữ cảnh tiến trình được chọn kế tiếp


process $P_0$
operating system
process $P_1$
interrupt or system call
executing
save state into $PCB_0$
reload state from $PCB_1$
idle
idle
executing
interrupt or system call
save state into $PCB_1$
reload state from $PCB_0$
idle
executing

# Dispatcher – Thảo luận

- 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 NSD 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)

# Mục tiêu điều phối

- Hiệu qủa (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

# Hai nguyên tắc điều phối CPU

## Độc quyền

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

## Không độc quyền

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

# Thời điểm ra quyết định điều phối

- Điều phối độc quyền (non-preemptive scheduling): tiến trình được chọn có quyền độc chiếm CPU
- Các thời điểm kích hoạt Scheduler
  - $P_{cur}$ kết thúc
  - $P_{cur}$ : running ->blocked
- 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 thời điểm kích hoạt Scheduler
  - $P_{cur}$ kết thúc
  - $P_{cur}$ : Running -> Blocked
  - Q  :  Blocked / New  -> Ready

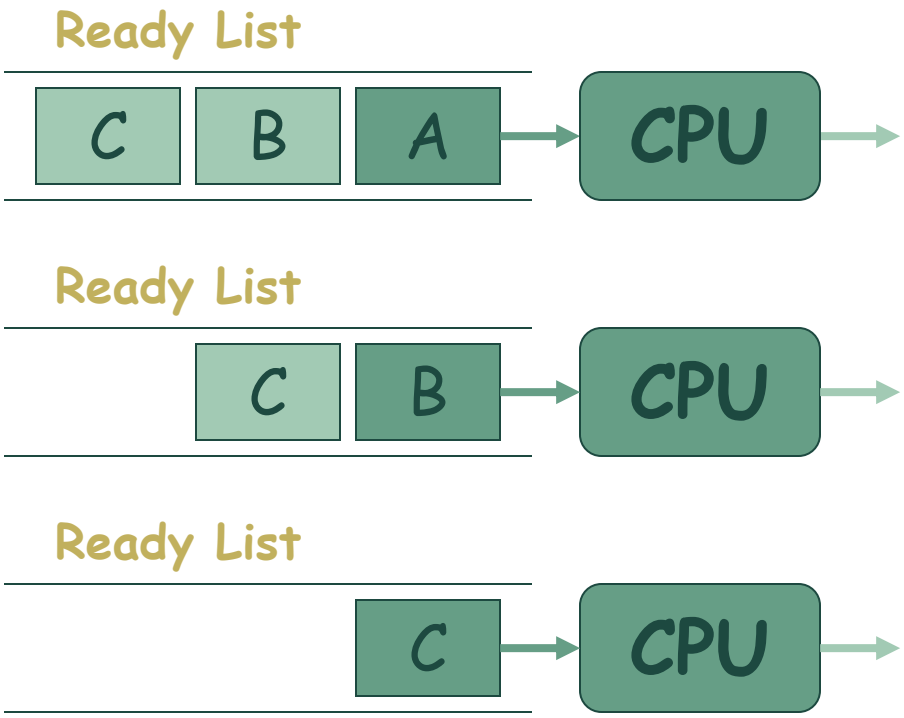
# Đánh giá chiến lược điều phối

- Sử dụng 2 đại lượng đo :
  - Turn- around time = $T_{quit} - T_{arrive}$: từ lúc vào HT đến khi hoàn tất
  - Waiting time = $T_{\text{in Ready}}$
- Xét trường hợp trung bình
  - N tiến trình
  - Avg $_{\text{Turn- around time}}$ = ($f$° Turn- around time $_{Pi}$ )/N
  - Avg $_{\text{Waiting time}}$ = ($f$° Waiting time $_{Pi}$ )/N

# Các chiến lược điều phối

- FIFO (FCFS)
- Xoay vòng (Round Robin)
- Theo độ ưu tiên
- Công việc ngắn nhất (SJF)
- Nhiều mức độ ưu tiên

# FCFS (First comes first served)




- Tiến trình vào RL lâu nhất được chọn trước
- Theo thứ tự vào RL
- Độc quyền

# Minh họa FCFS

| P | $T_{arriveRL}$ | CPU burst |
|---|---|---|
| P1 | 0 | 24 |
| P2 | 1 | 3 |
| P3 | 2 | 3 |

| P | TT | WT |
|---|---|---|
| P1 | 24 | 0 |
| P2 | 27-1 | 24-1 |
| P3 | 30-2 | 27-2 |

$Avg_{WT} = (23+25)/3 = 16$


0: P1 vào RL
P1 dùng CPU

1: P2 vào RL

2: P3 vào RL

24: P1 kết thúc
P2 dùng CPU

27: P2 kết thúc
P3 dùng CPU

# Nhận xét FCFS

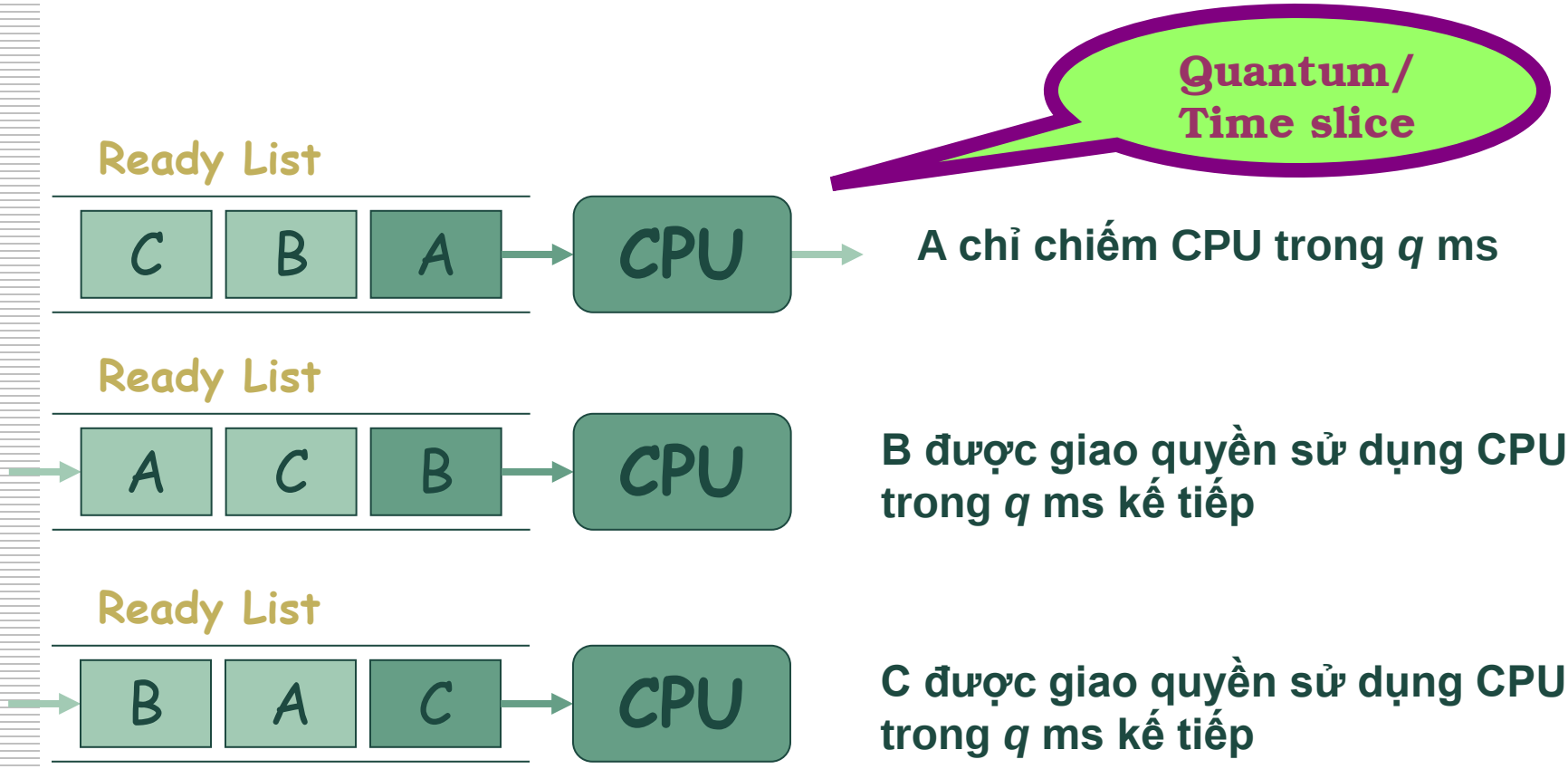
- Đơn giản
- Chịu đựng hiện tượng tích lũy thời gian chờ
- Tiến trình có thời gian xử lý ngắn đợi tiến trình có thời gian xử lý dài

➔ Ưu tiên tiến trình cpu-bounded

- Có thể xảy ra tình trạng độc chiếm CPU

# Điều phối XOAY VÒNG Round Robin (RR)

- Điều phối theo nguyên tắc FCFS
- Mỗi tiến trình chỉ sử dụng một lượng q cho mỗi lần sử dụng CPU

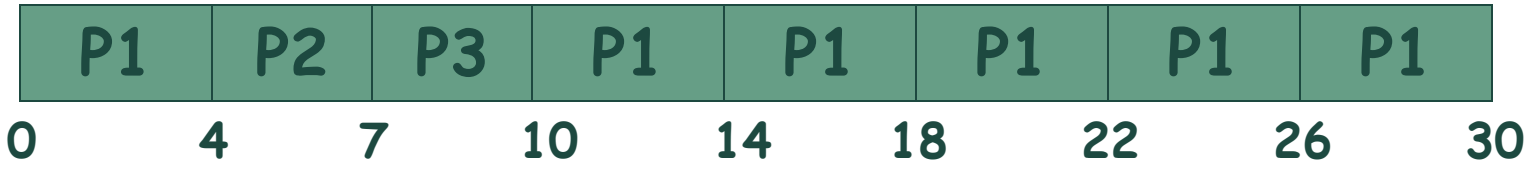


# Minh họa RR, q=4

| P | $T_{arriveRL}$ | CPU burst |
|---|---|---|
| P1 | 0 | 24 |
| P2 | 1 | 3 |
| P3 | 2 | 3 |

| P | TT | WT |
|---|---|---|
| P1 | 30 | 0+(10-4) |
| P2 | 7-1 | 4-1 |
| P3 | 10-2 | 7-2 |

$Avg_{WT} = (6+3+5)/3 = 4.66$




**0:00 P1 vào, P1 dùng CPU**

**0:01 P2 vào (đợi)**

**0:02 P3 vào (đợi)**

**0:04 P1 hết lượt, P2 dùng CPU**

**0:07 P2 dừng, P3 dùng CPU**

**0:10 P3 dừng, P1 dùng CPU**

**0:14 P1 vẫn chiếm CPU**

**...**

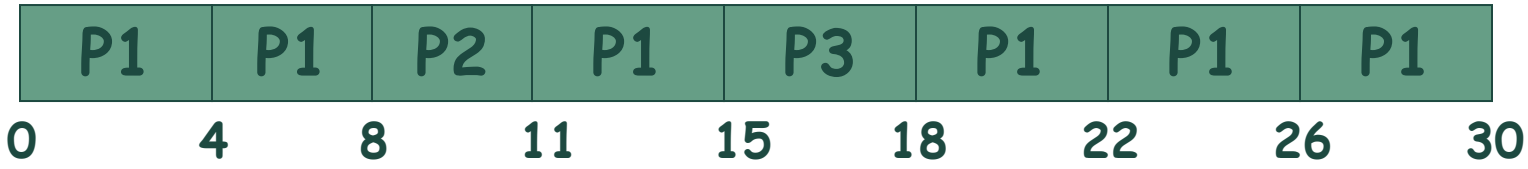
# Minh họa RR, q=4

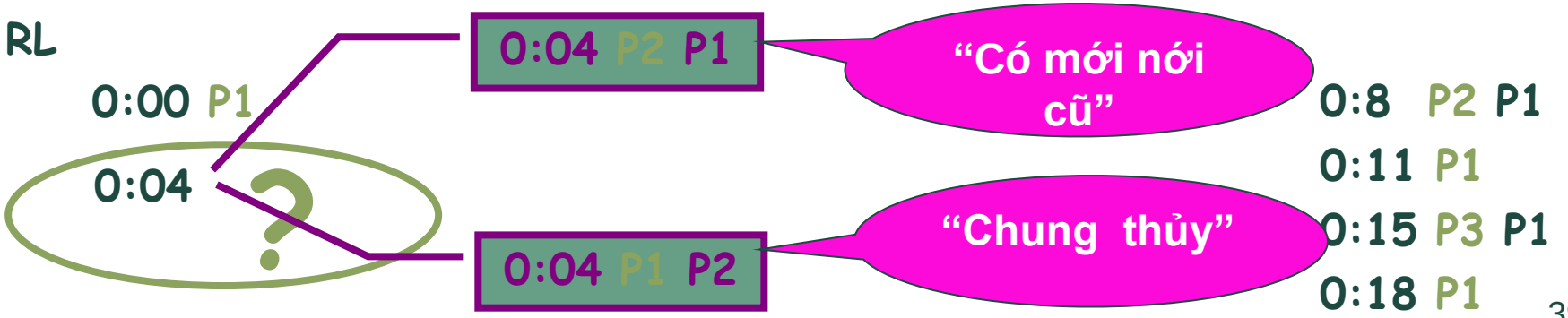
| P | $T_{arriveRL}$ | CPU burst |
|---|---|---|
| P1 | 0 | 24 |
| P2 | 4 | 3 |
| P3 | 12 | 3 |

**Tranh chấp vị trí trong RL : "Chung thủy"**

- **P : running -> ready**
- **P : blocked -> ready**
- **P: new ->ready**

**Không phải luôn luôn có thứ tự điều phối P1 P2 P3 P4 P1 P2 P3 P4...**



# Round Robin

- Khi nào kết thúc 1 lượt sử dụng CPU
- Hết thời lượng q ms (quantum) cho phép
- Tiến trình kết thúc
- Tiến trình bị khóa
  - Chờ Rs
  - Chờ biến cố

# Round Robin – Nhận xét

- Sử dụng cơ chế không độc quyền
- Mỗi tiến trình không phải đợi quá lâu
- Loại bỏ hiện tượng độc chiếm CPU
- Hiệu quả ?
- Phụ thuộc vào việc chọn lựa quantum *q*
  - *q* quá lớn ???
  - *q* quá nhỏ ???

**Bao lâu ?**

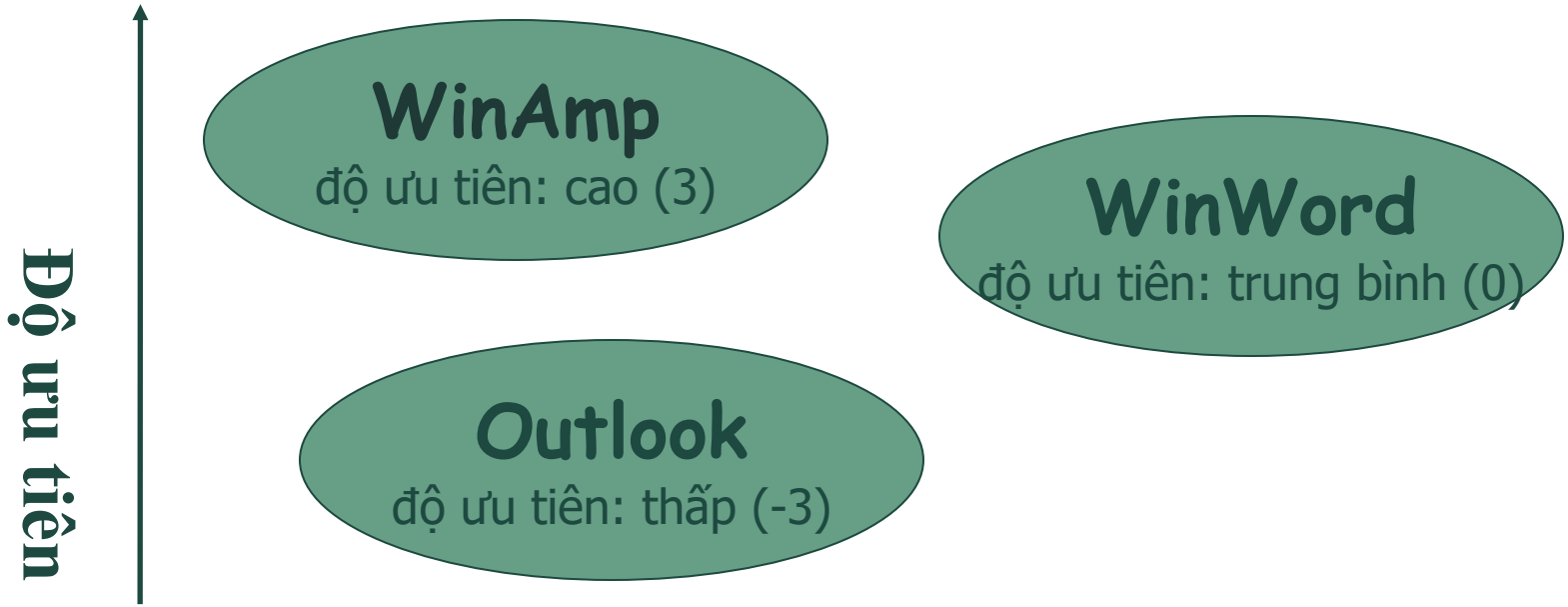
**Giảm tính tương tác**

**Tăng chi phí chuyển đổi ngữ cảnh**

- Trường hợp xấu nhất của RR ?

# Điều phối với độ ưu tiên

Phân biệt tiến trình quan trọng >< tiến trình bình thường?




**Tiến trình có độ ưu tiên cao nhất được chọn cấp CPU trước**

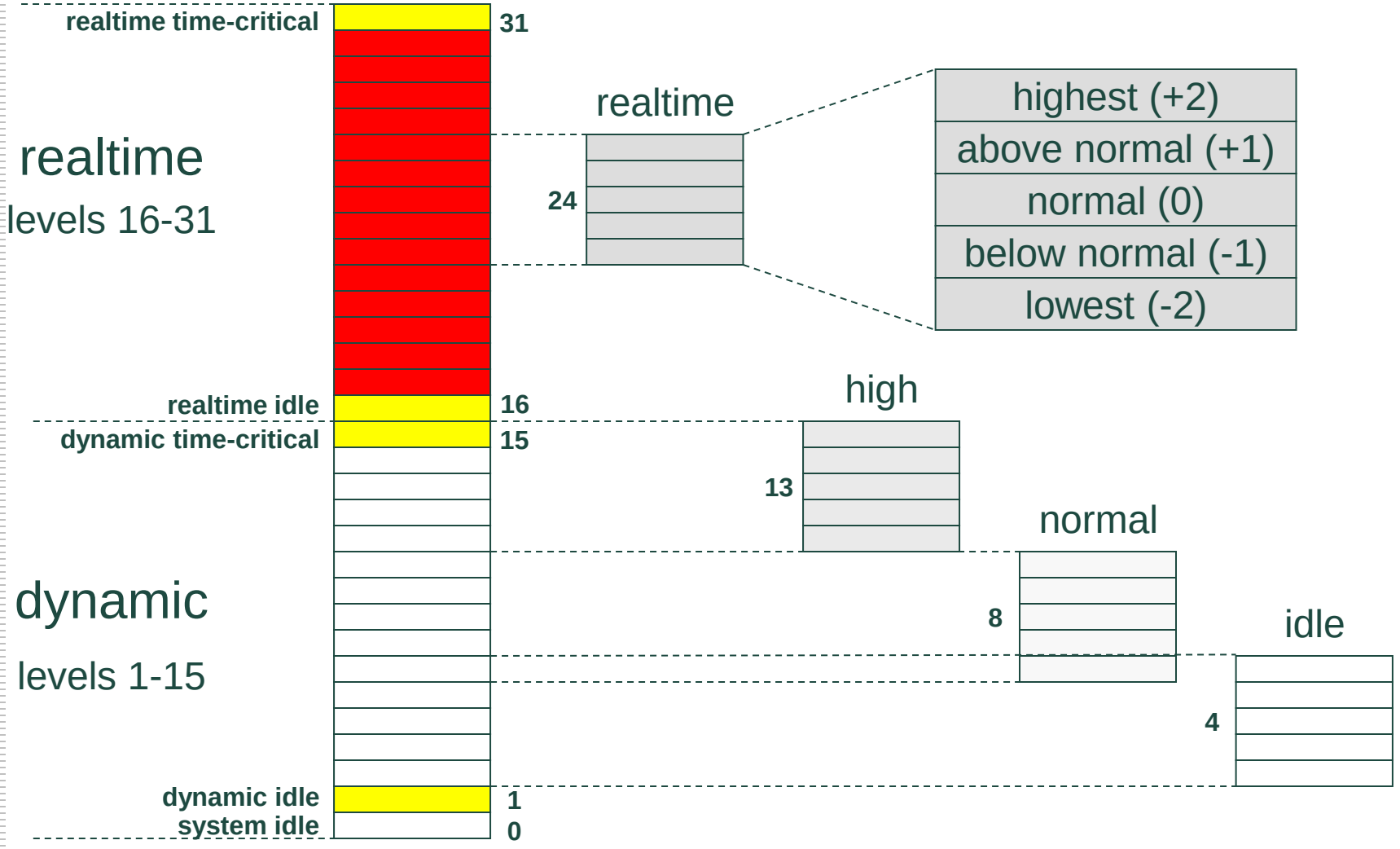
# Điều phối với độ ưu tiên

- Cách tính độ ưu tiên?
- Hệ thống gán: CPU times,…
- Người dùng gán tường minh
- Tính chất độ ưu tiên :
- Tĩnh
- Động

# Ví dụ: Độ ưu tiên của HĐH WinNT

- WinNT gán cho mỗi tiến trình độ ưu tiên có giá trị giữa 0 & 31
- 0 (độ ưu tiên nhỏ nhất): dành riêng cho trạng thái *system idle*
- Độ ưu tiên được phân theo nhóm:
- *Realtime* : (16 - 31)
  - Thích hợp cho các tiến trình thời gian thực
  - Dành riêng cho các tiến trình của người quản trị hệ thống
- *Dynamic* : (0 - 15)
  - Thích hợp cho các tiến trình của người dùng thường
  - *Chia thành 3 mức :*
    - *high* (11 - 15)
    - *normal* (6 - 10)
    - *idle* (2 - 6)

# Biểu đồ phân bố độ ưu tiên của WinNT



# Nguyên tắc điều phối

- Độc quyền
- Lượt sử dụng CPU kết thúc khi:
  - Tiến trình kết thúc,
  - Tiến trình bị khóa
- Không độc quyền
- Lượt sử dụng CPU kết thúc khi:
  - Tiến trình kết thúc,
  - Tiến trình bị khóa,
  - Có tiến trình với độ ưu tiên cao hơn vào RL

# Độ ưu tiên – không độc quyền

| P | $T_{RL}$ | Priority | CPU burst |
|---|---|---|---|
| P1 | 0 | 0 | 24 |
| P2 | 1 | 2 | 3 |
| P3 | 2 | 1 | 3 |

| P | TT | WT |
|---|---|---|
| P1 | 30 | 0+(7-1) |
| P2 | 4-1 | 0 |
| P3 | 7-2 | 4-2 |

$Avg_{WT} = (6+0+2)/3 = 2.66$


**0: P1 vào, P1 dùng CPU**

**1: P2 vào (độ ưu tiên cao hơn P1)**

**P2 dành quyền dùng CPU**

**2: P3 vào (độ ưu tiên thấp hơn P2)**

**P3 không dành được quyền dùng CPU**

**4: P2 kết thúc, P3 dùng CPU**

**7: P3 dừng, P1 dùng CPU**

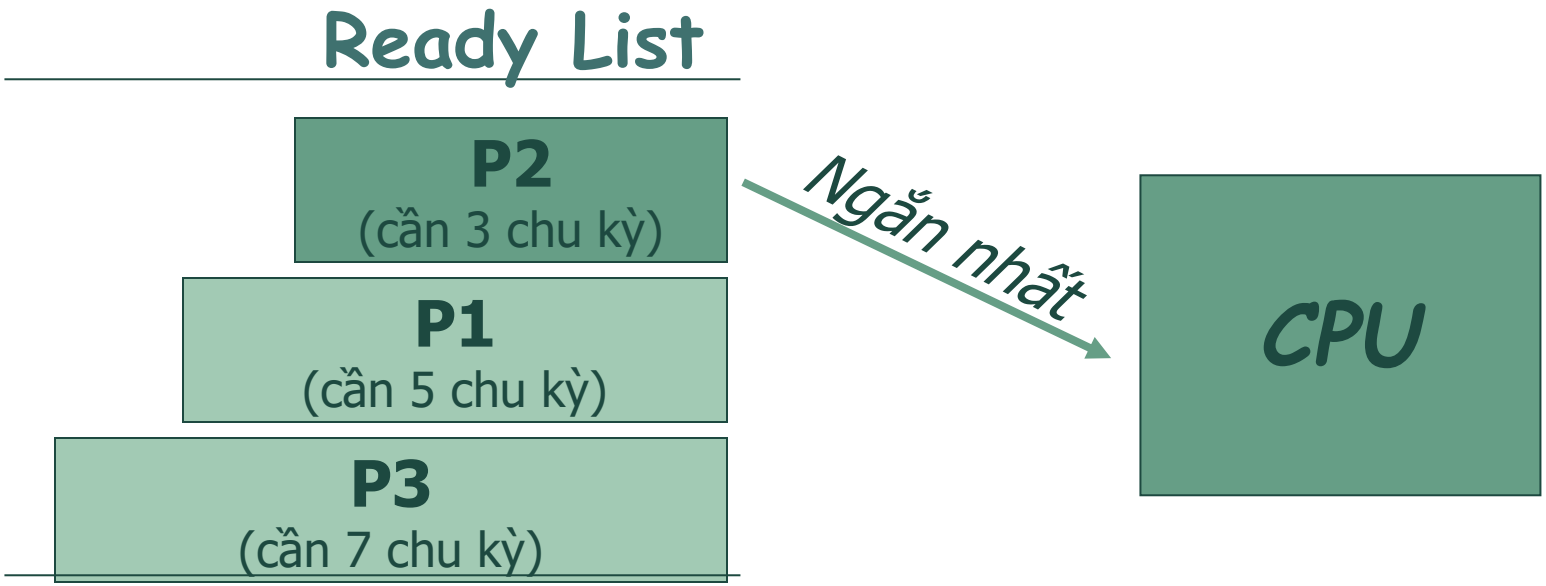
**30: P1 dừng**

# Độ ưu tiên – không độc quyền - Nhận xét

- Số phận tiến trình có độ ưu tiên thấp?
- Chờ lâu, lâu, lâu ...

➔ Giải quyết: tăng độ ưu tiên cho những tiến trình chờ lâu trong hệ thống (Aging)

# Shortest Job First (SJF)




Là một dạng độ ưu tiên đặc biệt với độ ưu tiên

$$p_i = \text{thời\_gian\_còn\_lại}(\text{Process}_i)$$

➔ Có thể cài đặt độc quyền hoặc không độc quyền

# Minh họa SJF (độc quyền)(1)

| P | $T_{arriveRL}$ | CPU burst |
|---|---|---|
| P1 | 0 | 24 |
| P2 | 1 | 3 |
| P3 | 2 | 3 |

| P | TT | WT |
|---|---|---|
| P1 | 24 | 0 |
| P2 | 27-1 | 24-1 |
| P3 | 30-2 | 27-2 |

$Avg_{WT} = (23+25)/3 = 16$


**0:00 P1 vào, P1 dùng CPU**

**0:01 P2 vào RL**

**0:02 P3 vào RL**

**0:24 P1 kết thúc, P2 dùng CPU**

**0:27 P2 dừng, P3 dùng CPU**

**0:30 P3 dừng**

# Minh họa SJF (độc quyền)(2)

| P | $T_{arriveRL}$ | CPU burst |
|---|---|---|
| P1 | 0 | 24 |
| P2 | 1 | 3 |
| P3 | 1 | 2 |

| P | TT | WT |
|---|---|---|
| P1 | 24 | 0 |
| P2 | 29-1 | 26-1 |
| P3 | 26-1 | 24-1 |

$Avg_{WT} = (24+22)/3 = 15.33$


**0:00 P1 vào, P1 dùng CPU**

**0:01 P2 vào**

**0:01 P3 vào**

**0:24 P1 kết thúc, P3 dùng CPU**

**0:26 P3 dừng, P2 dùng CPU**

**0:29 P2 dừng**

# Minh họa SJF (khôngđộc quyền) (1)

| P | $T_{arriveRL}$ | CPU burst |
|---|---|---|
| P1 | 0 | 24 |
| P2 | 1 | 3 |
| P3 | 2 | 3 |

| P | TT | WT |
|---|---|---|
| P1 | 30 | 0+(7-1) |
| P2 | 4-1 | 0 |
| P3 | 7-2 | 4-2 |

$Avg_{WT}$ = (6+0+2)/3 = 2.66


**0:00 P1 vào, P1 dùng CPU**

**0:01 P2 vào (độ ưu tiên cao hơn P1)**

**P2 dành quyền dùng CPU**

**0:4 P2 kết thúc, P3 dùng CPU**

**0:7 P3 dừng, P1 dùng CPU**

**0:30 P1 dừng**

# Minh họa SJF (khôngđộc quyền) (2)

| P | $T_{arriveRL}$ | CPU burst |
|---|---|---|
| P1 | 0 | 24 |
| P2 | 1 | 5 |
| P3 | 3 | 4 |

| P | TT | WT |
|---|---|---|
| P1 | 33 | 0+(10-1) |
| P2 | 5 | 0 |
| P3 | 7 | 6-3 |

$Avg_{WT} = (9+0+3)/3 = 4$


**0:00 P1 vào, P1 dùng CPU**

**0:01 P2 vào (độ ưu tiên cao hơn P1)**

**P2 dành quyền dùng CPU**

**0:03 P3 vào (độ ưu tiên < P2)**

**P2 dành quyền dùng CPU**

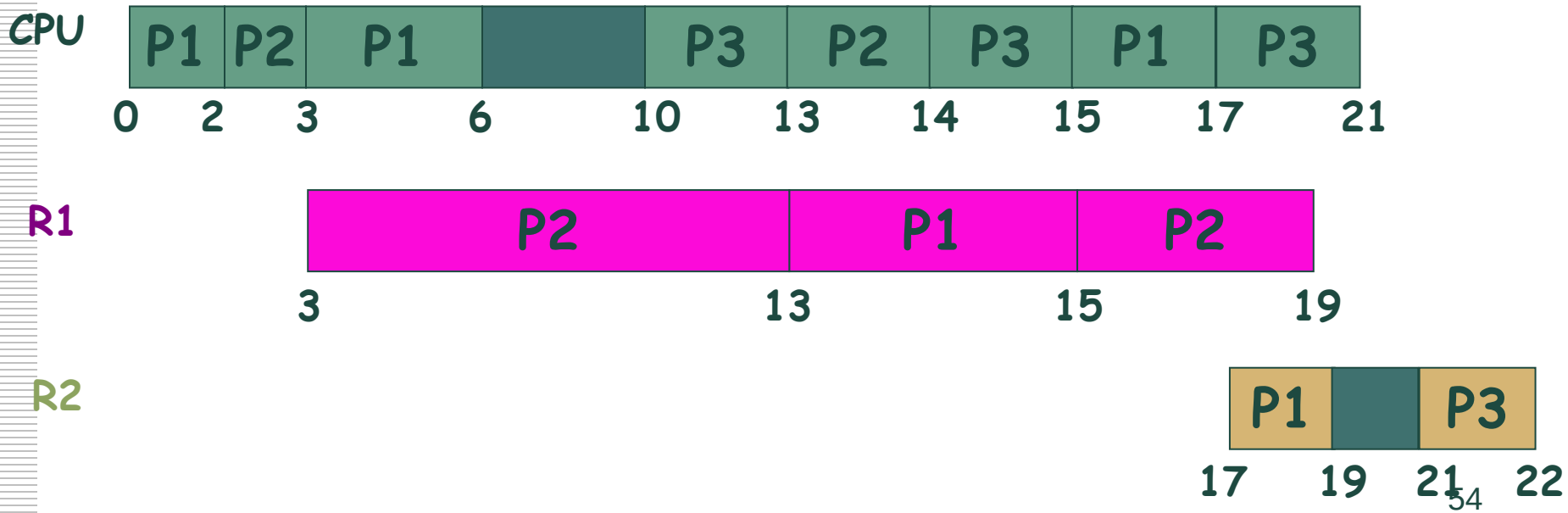
**0:6 P2 kết thúc, P3 dùng CPU**

**0:10 P3 dừng, P1 dùng CPU**

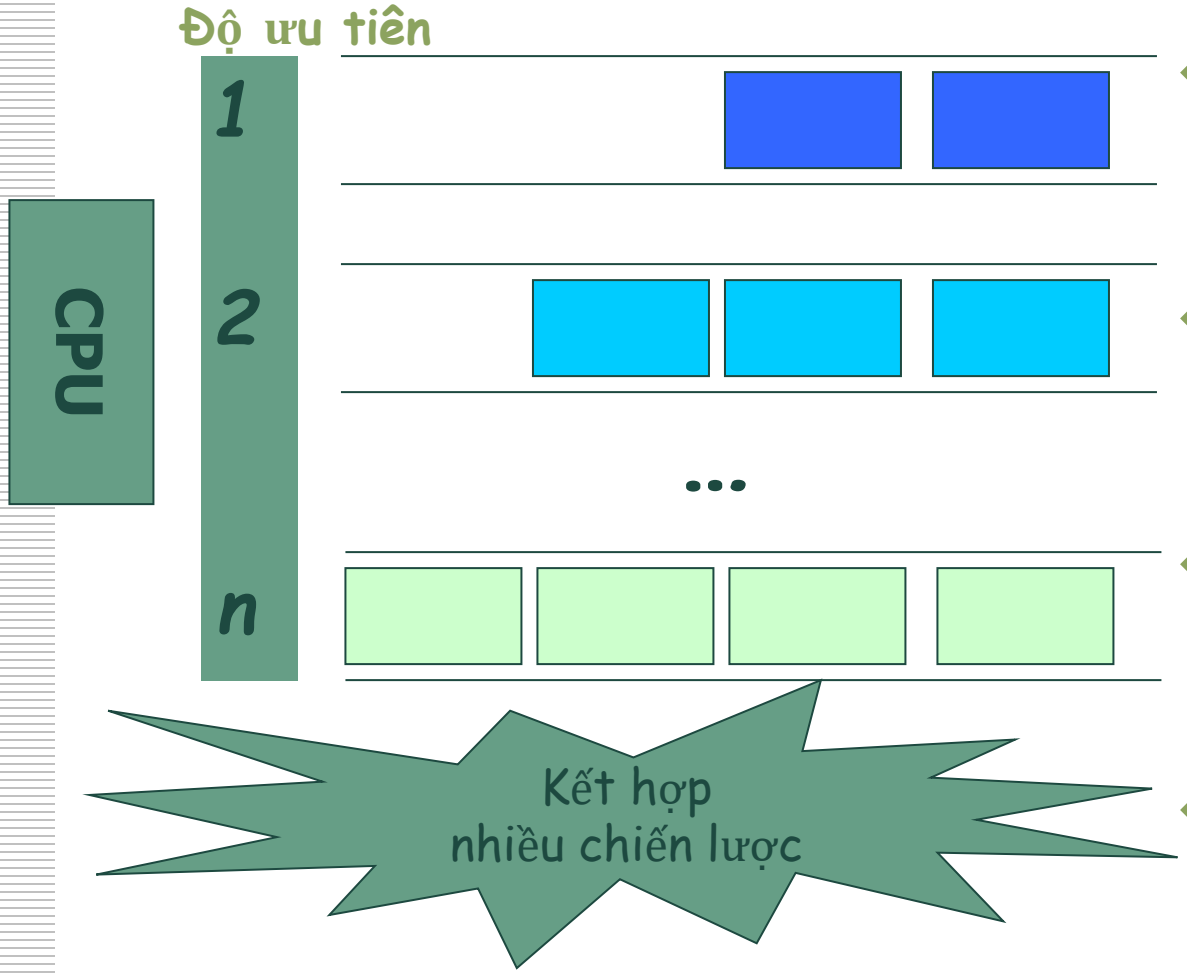
**0:33 P1 dừng**

# Minh họa SJF (nhiều chu kỳ CPU)

| P | $T_{arriveRL}$ | CPU1 burst | IO1 R | IO1 T | CPU2 burst | IO2 R | IO2 T |
|---|---|---|---|---|---|---|---|
| P1 | 0 | 5 | R1 | 2 | 2 | R2 | 2 |
| P2 | 2 | 1 | R1 | 10 | 1 | R1 | 4 |
| P3 | 10 | 8 | R2 | 1 | 0 | Null | 0 |

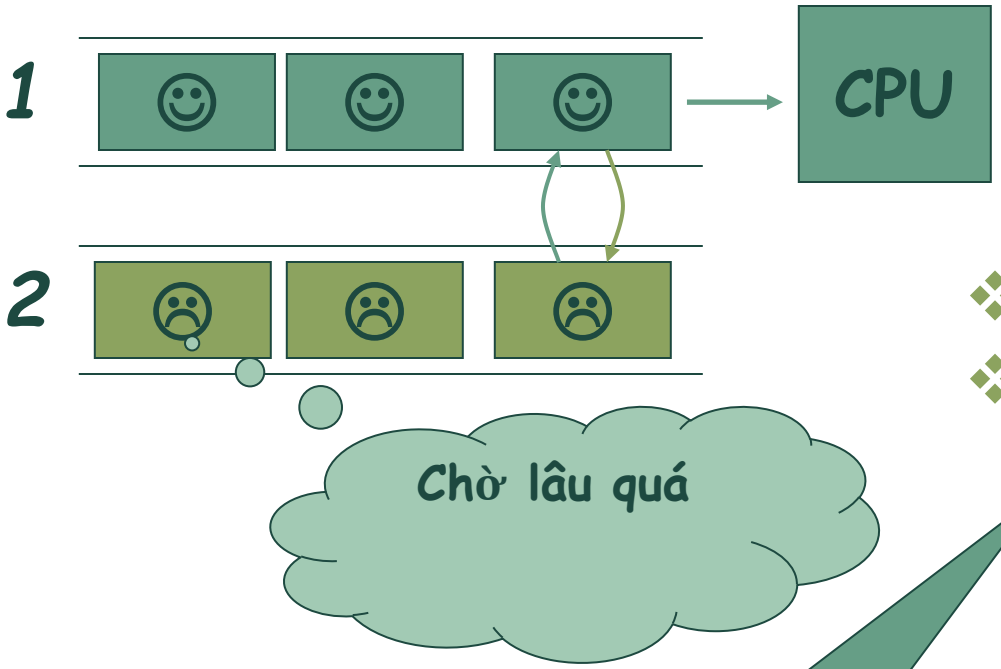


# Điều phối với nhiều mức ưu tiên




- Tổ chức N RL ứng với nhiều mức ưu tiên
- Mỗi $RL_i$ áp dụng một chiến lược điều phối thích hợp
- Giữa các RL áp dụng điều phối theo độ ưu tiên :
- $RL_i$ rỗng mới điều phối $RL_{i+1}$

# Khuyết điểm




- Starvation !!!
- Giải pháp Aging :
  - Chờ lâu quá : chuyển lên RL với độ ưu tiên cao hơn
  - Chiếm CPU lâu quá : chuyển xuống RL với độ ưu tiên thấp hơn

**Khi nào thực hiện aging?**
**Aging tiến trình nào?**

# Bài tập

**Hãy điều phối: CPU: SJF không độc quyền. R1,R2: FIFO**

| Tiến trình | Thời điểm vào Ready list | CPU1 | IO lần 1 | | CPU2 | IO lần 2 | |
|---|---|---|---|---|---|---|---|
| | | | Thời gian | Thiết bị | | Thời gian | Thiết bị |
| P1 | 0 | 8 | 5 | R1 | 1 | 0 | Null |
| P2 | 2 | 1 | 8 | R2 | 2 | 5 | R1 |
| P3 | 10 | 6 | 5 | R1 | 2 | 3 | R2 |
| P4 | 11 | 3 | 20 | R2 | 0 | 0 | Null |