



# Quản lý tiến trình

**Môn học: Hệ điều hành**

# Nội dung

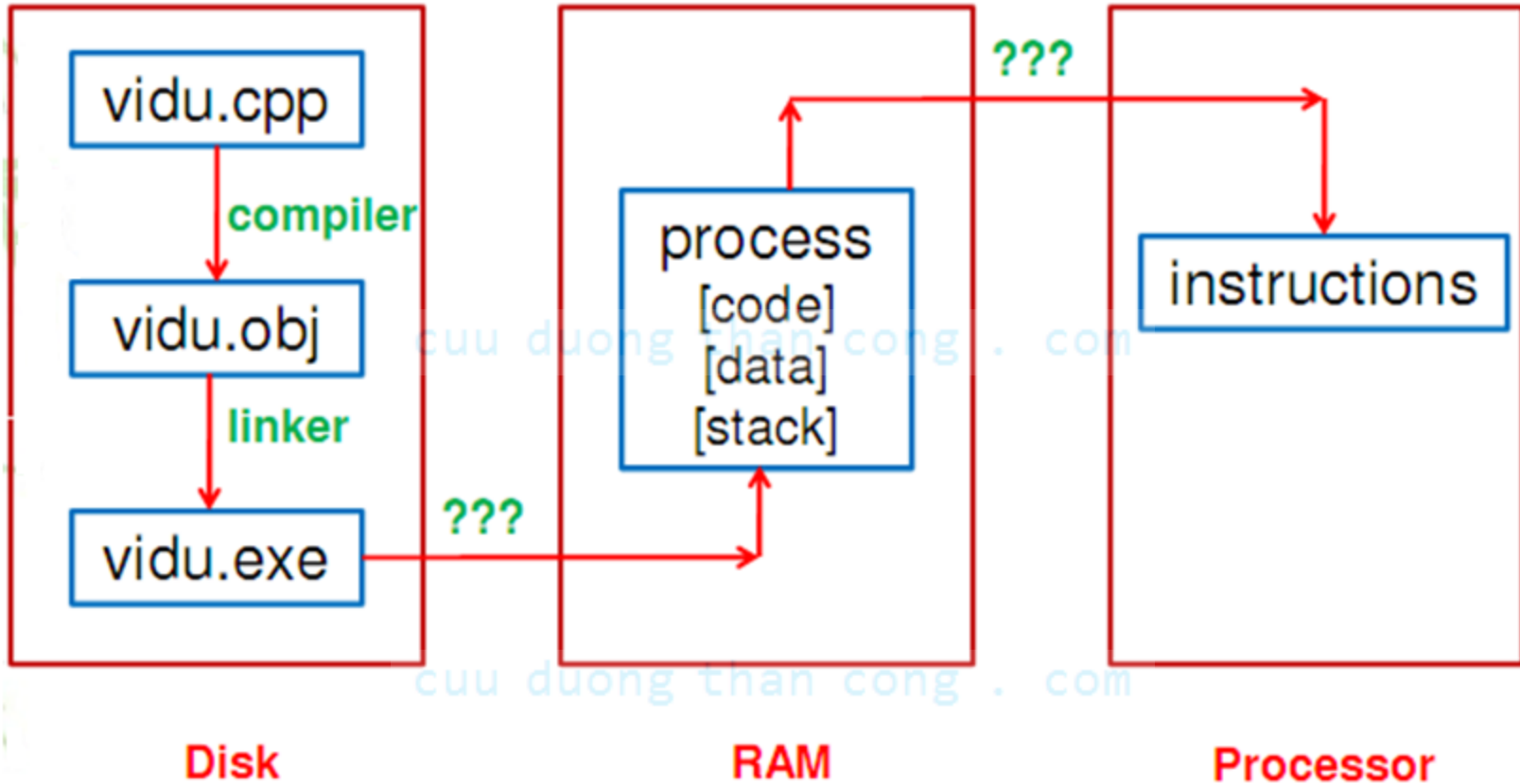
- Phân biệt tiến trình và tiểu trình
- So sánh các thuật toán điều phối tiến trình

cuu duong than cong . com

cuu duong than cong . com



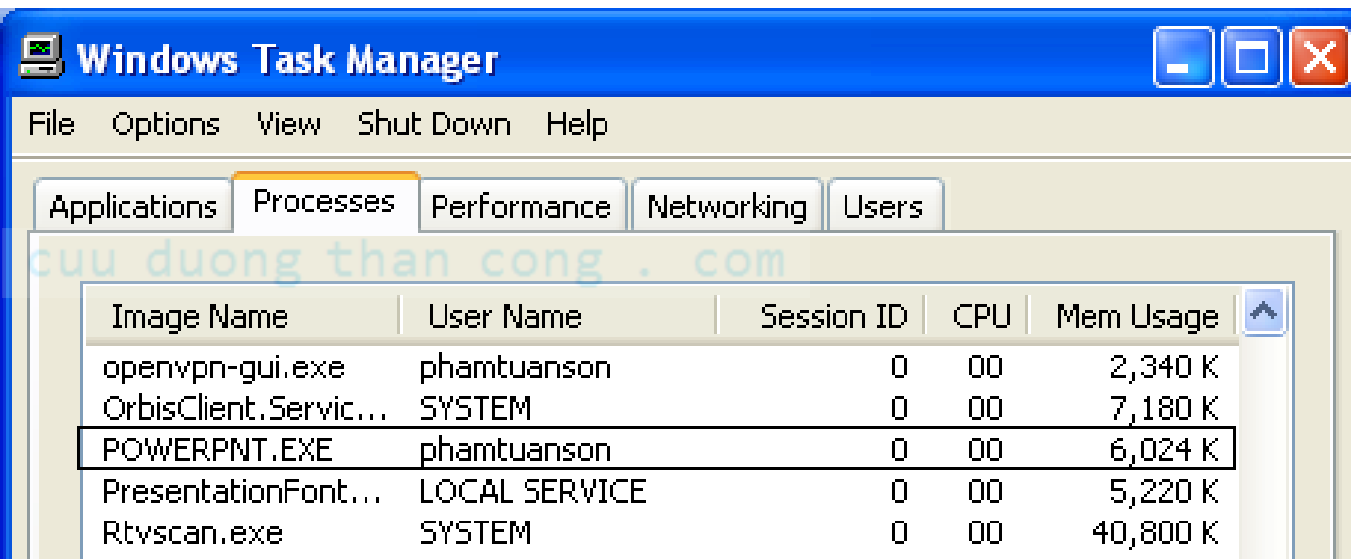
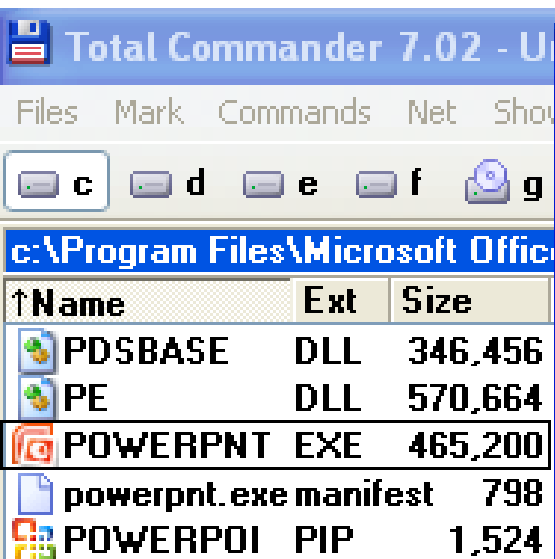
# Thực thi chương trình



# Tiến trình

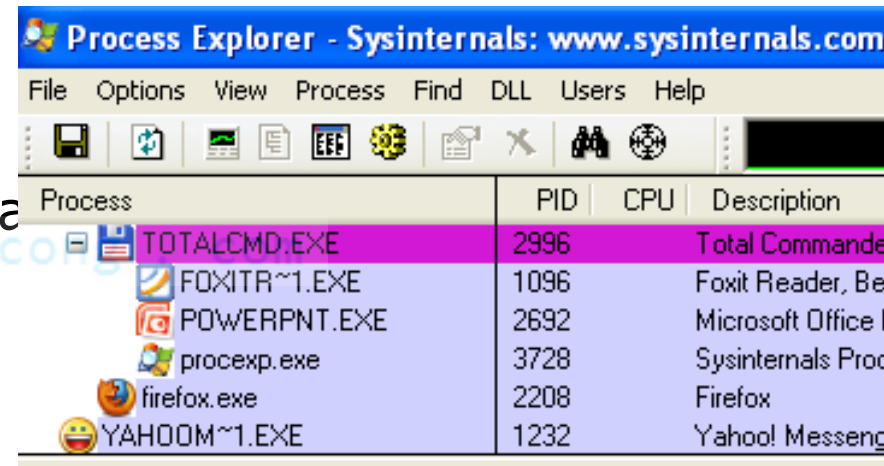
- Tiến trình là một chương trình đang được thực thi
- Một tiến trình cần sử dụng các tài nguyên: CPU, bộ nhớ, tập tin, thiết bị nhập xuất để hoàn tất công việc của nó

cuu duong than cong . com



# 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



The screenshot shows the Process Explorer window from Sysinternals. The title bar reads 'Process Explorer - Sysinternals: www.sysinternals.com'. The menu bar includes File, Options, View, Process, Find, DLL, Users, and Help. The toolbar contains icons for saving, refreshing, and other process management actions. The main table lists running processes with columns for Process, PID, CPU, and Description.

Process	PID	CPU	Description
TOTALCMD.EXE	2996		Total Commander
FOXITR~1.EXE	1096		Foxit Reader, Be
POWERPNT.EXE	2692		Microsoft Office
procexp.exe	3728		Sysinternals Proc
firefox.exe	2208		Firefox
YAHOOM~1.EXE	1232		Yahoo! Messeng

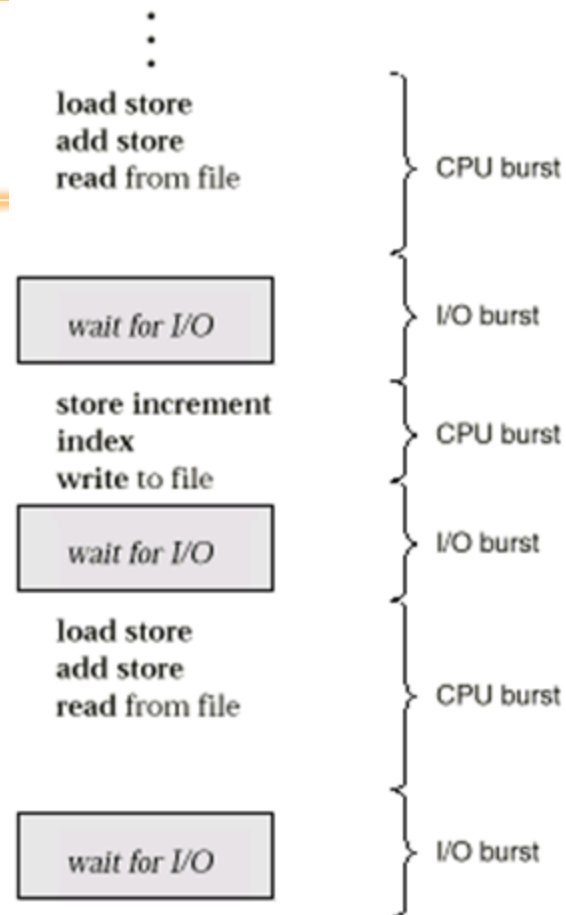
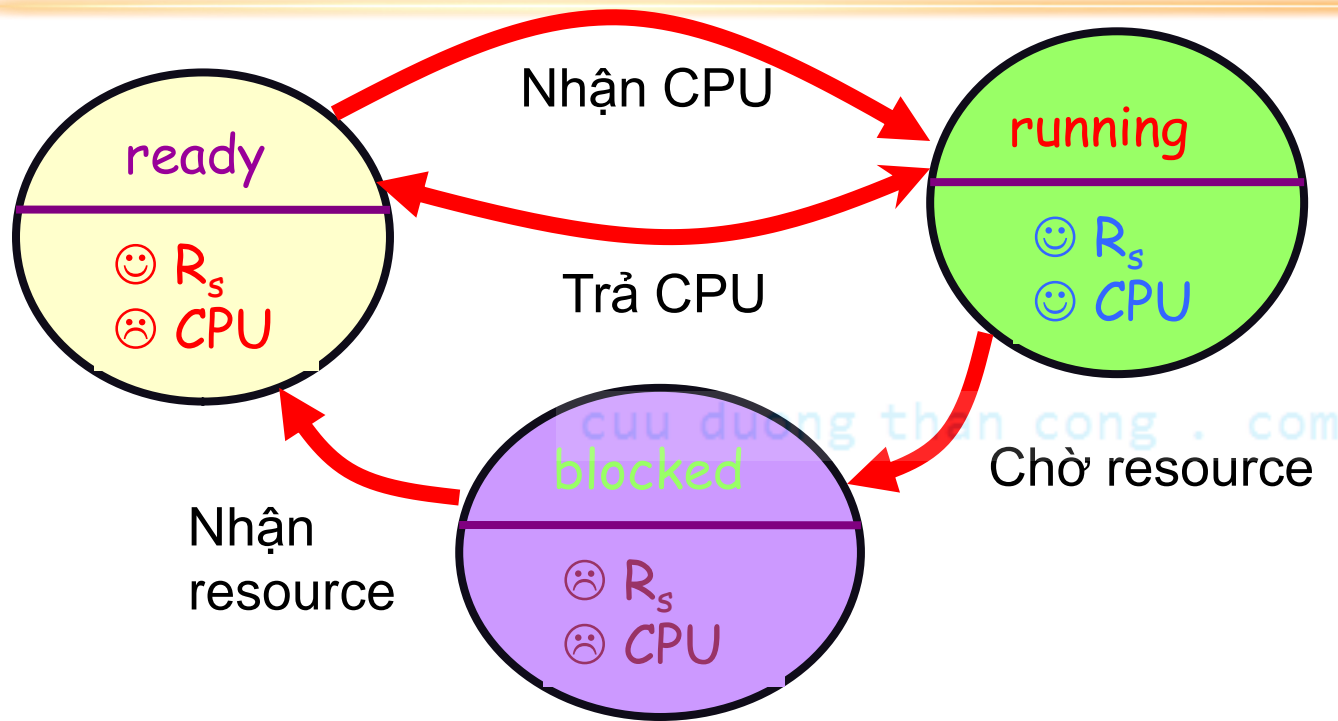
# Thao tác trên tiến trình (2/2)

- 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





# Hoạt động của tiến trình

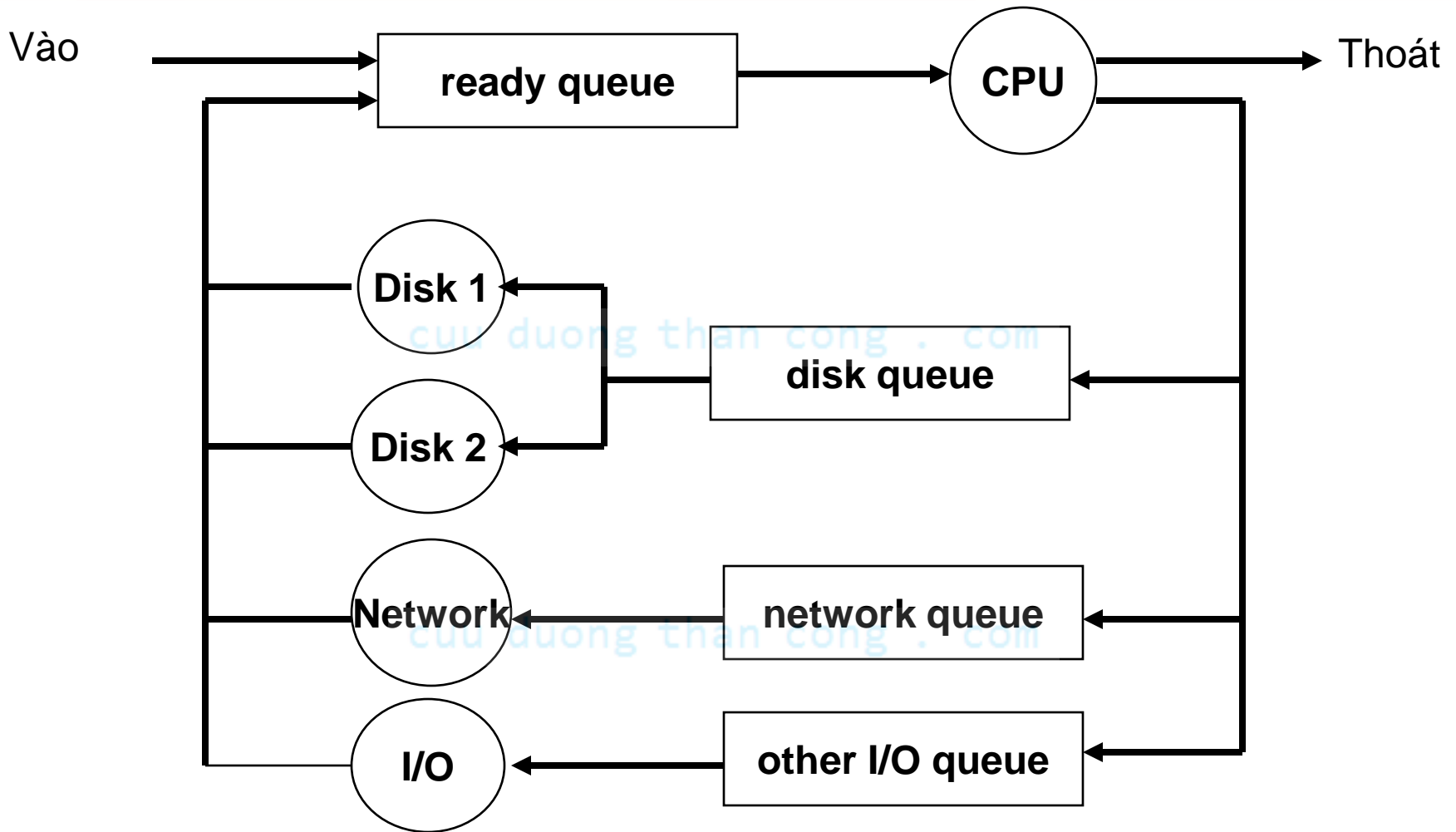


CPU-bound process



IO-bound process

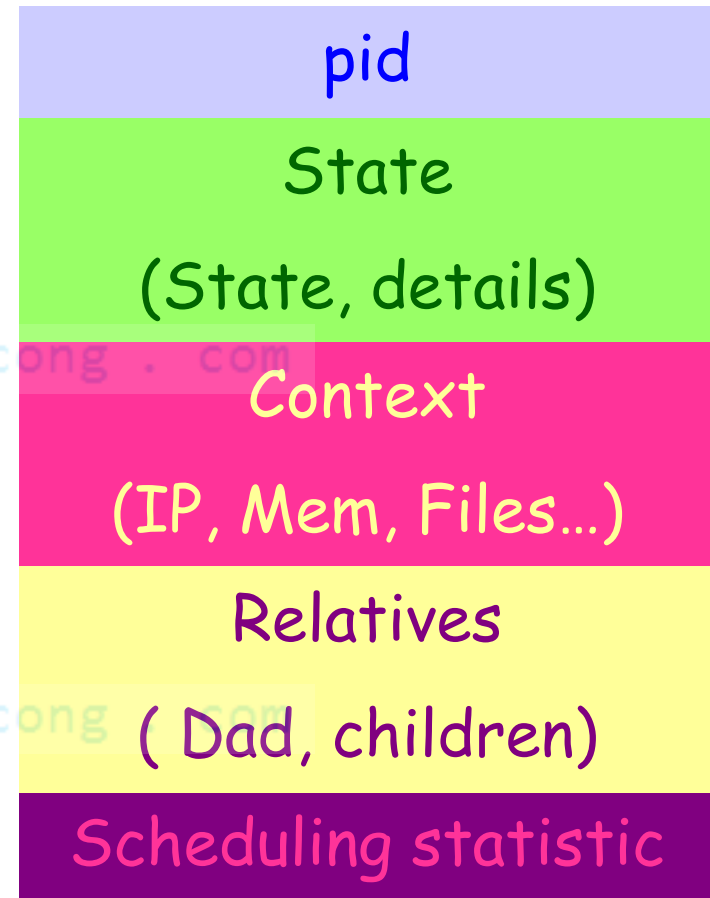
# Sơ đồ hàng đợi





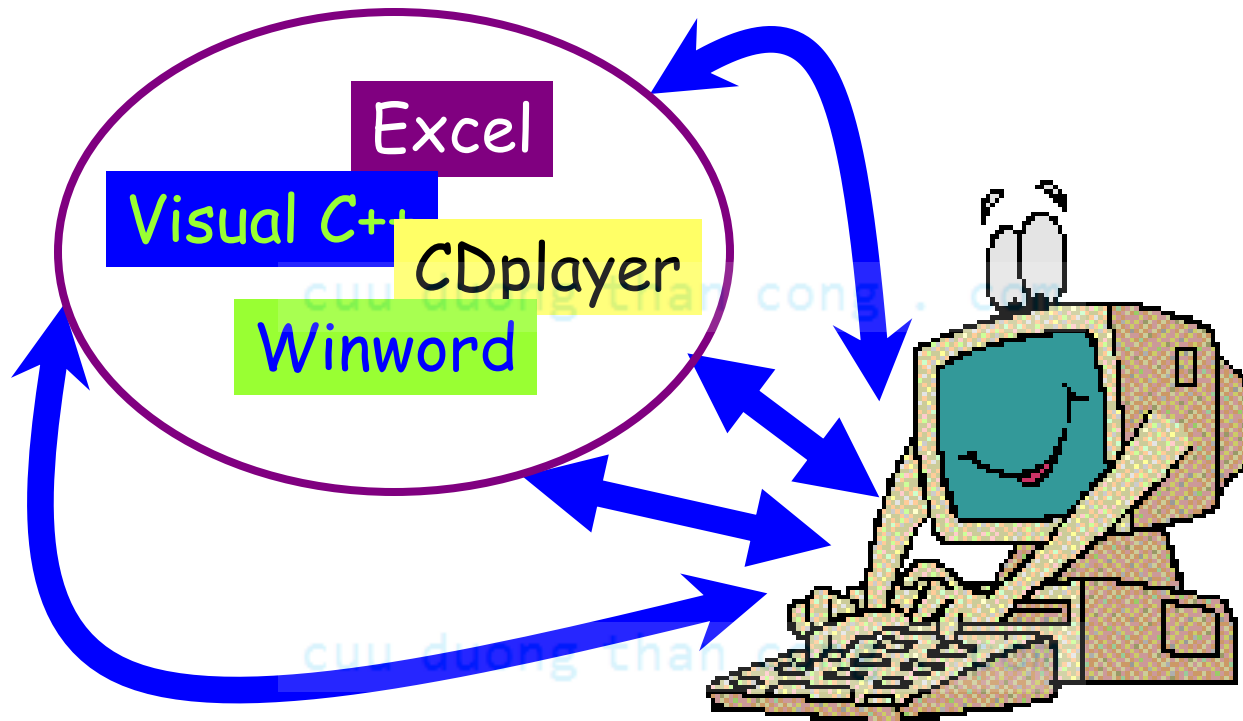
# Khởi quản lý tiến trình (PCB – Process Control Block)

- Định danh (Process ID)
- Trạng thái tiến trình
- Ngưỡng 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

# Môi trường đa chương



# Giải pháp

Tiến trình 1



Tiến trình 2



Tiến trình 3



Điều phối  
như thế nào  
???



Hệ  
điều  
hành

# Điều phối tiến trình (Scheduling)

- Mục tiêu điều phối
- Lựa chọn tiến trình
  - Tiêu chuẩn lựa chọn
  - Thời điểm lựa chọn
- Chuyển đổi giữa các tiến trình
- Các chiến lược điều phối
  - FCFS ☐ Multiple Queues
  - Round Robin ☐ Guaranteed Scheduling
  - Priority ☐ Lottery Scheduling
  - SJF ☐ Fair-Share Scheduling



# Mục tiêu điều phối

- Mục tiêu chung
  - Công bằng sử dụng CPU
  - Cân bằng sử dụng các thành phần của hệ thống
- Hệ thống theo lô
  - Tối ưu throughput
  - Giảm thiểu turnaround time:  $T_{\text{quit}} - T_{\text{arrive}}$
  - Tận dụng CPU
- Hệ thống tương tác
  - Giảm thiểu thời gian chờ (Tối ưu thời gian hồi đáp):  $T_{\text{in ReadyQueue}}$
  - Cân đối mong muốn của người dùng
- Hệ thống thời gian thực
  - Thời hạn hoàn thành công việc



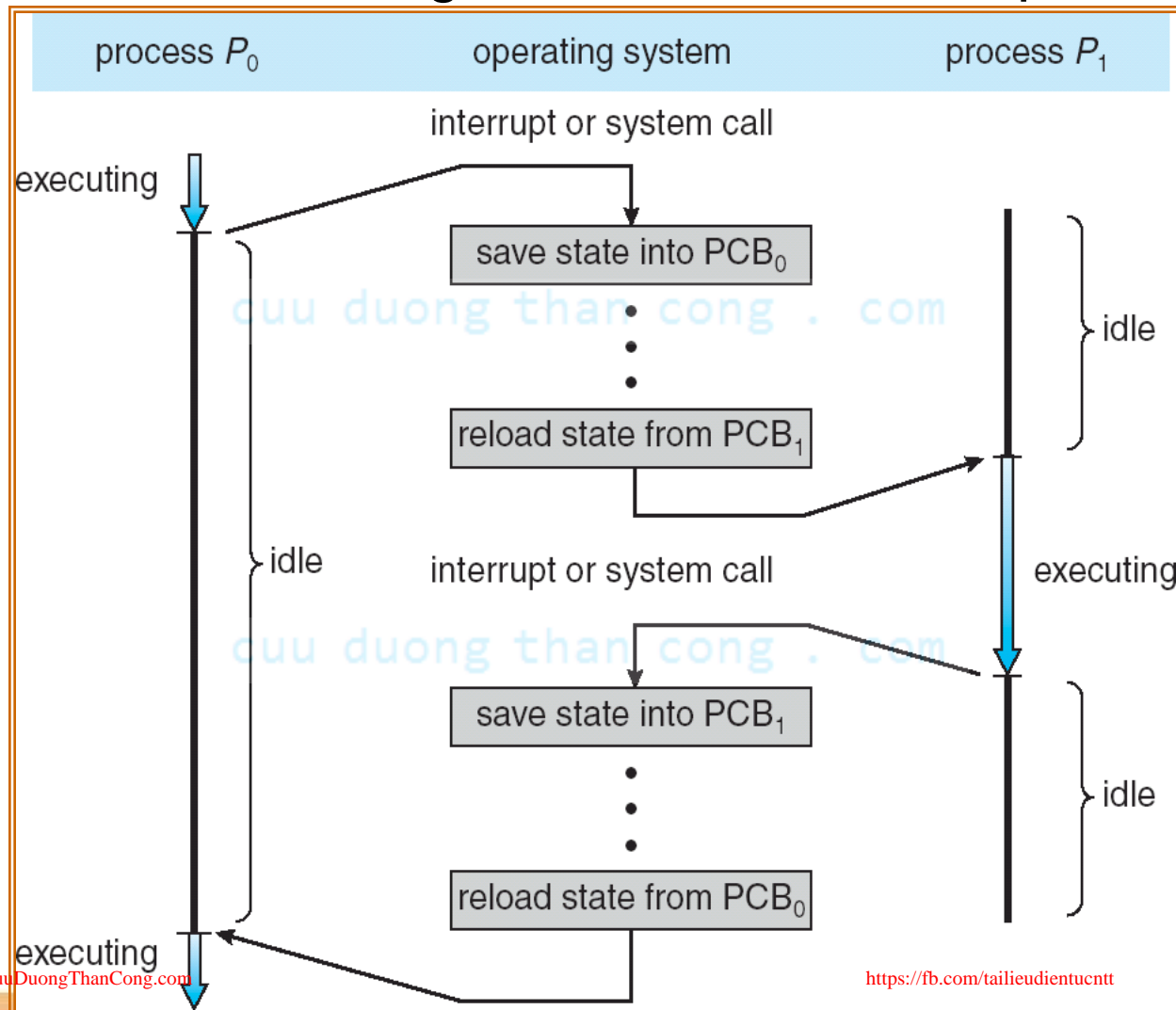
# Lựa chọn tiến trình

- Tiêu chí lựa chọn
  - Chọn tiến trình vào RQ trước
  - Chọn tiến trình có độ ưu tiên cao hơn
- Thời điểm lựa chọn
  - Điều phối độc quyền (non-preemptive scheduling): một khi tiến trình đang ở trạng thái Running, nó sẽ tiếp tục sử dụng CPU cho đến khi kết thúc hoặc bị block vì I/O hay các dịch vụ của hệ thống (độc chiếm CPU)
    - $P_{cur}$  kết thúc
    - $P_{cur}$  : running  $\rightarrow$  blocked
  - Điều phối không độc quyền (preemptive scheduling): ngoài thời điểm lựa chọn như điều phối độc quyền, tiến trình đang sử dụng CPU có thể bị ngắt (chuyển sang trạng thái Ready) khi hết thời gian qui định hoặc có tiến trình có độ ưu tiên hơn vào ReadyQueue
    - Q : blocked / new  $\rightarrow$  ready



# Chuyển đổi giữa các tiến trình (1/2)

## Context switching – Nhiệm vụ của Dispatcher





# Chuyển đổi giữa các tiến trình (2/2)

- 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)



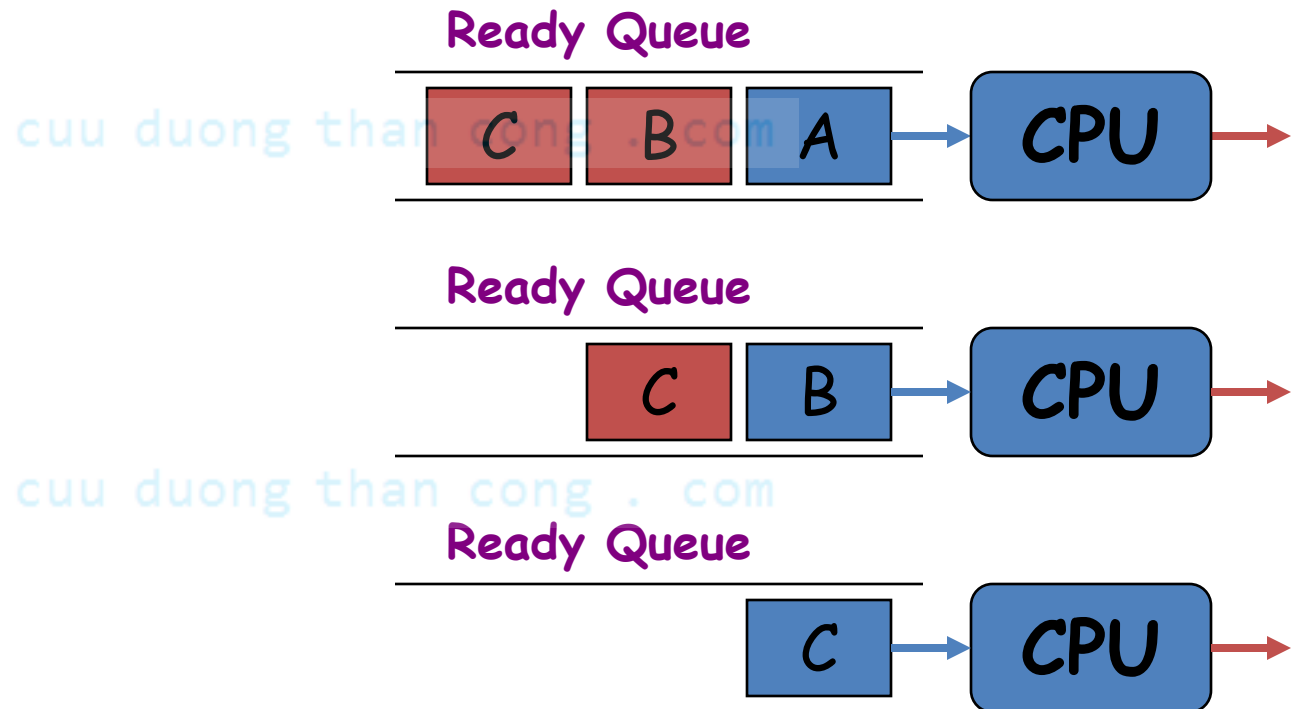
# Chiến lược điều phối tiến trình

- FCFS
- Round Robin
- Priority
- SJF
- Multiple Queues
- Guaranteed Scheduling
- Lottery Scheduling
- Fair-Share Scheduling



# First-Come First-Served (FCFS)

- Tiêu chí lựa chọn tiến trình
  - Thứ tự vào hàng đợi Ready Queue
- Thời điểm lựa chọn tiến trình
  - Độc quyền



# Minh họa FCFS

P	T <sub>arriveRQ</sub>	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 RQ  
P1 dùng CPU

0:01 P2 vào RQ

0:02 P3 vào RQ

0:24 P1 kết thúc  
P2 dùng CPU

0:27 P2 kết thúc  
P3 dùng CPU

# Nhận xét FCFS

- Đơn giản, dễ cài đặt
- 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
- Không phù hợp với hệ thống tương tác người dùng

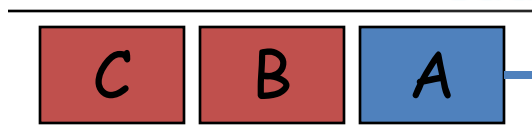


# Round Robin (RR)

- Mỗi tiến trình chỉ sử dụng một lượng  $q$  cho mỗi lần sử dụng CPU
- Tiêu chí lựa chọn tiến trình
  - Thứ tự vào hàng đợi Ready Queue
- Thời điểm lựa chọn tiến trình
  - Không độc quyền (không có độ ưu tiên)

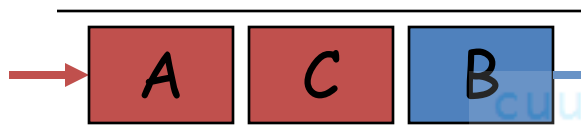
Quantum/  
Time slice

Ready Queue



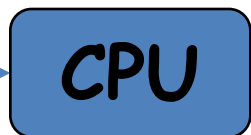
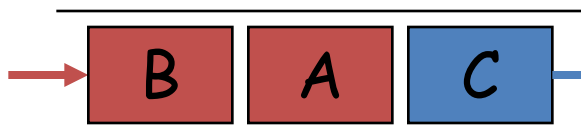
A chỉ chiếm CPU trong  $q$  ms

Ready Queue



B được giao quyền sử dụng CPU trong  $q$  ms kế tiếp

Ready Queue



C được giao quyền sử dụng CPU trong  $q$  ms kế tiếp

# Minh họa RR, q=4

P	T <sub>arriveRQ</sub>	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

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

P1	P2	P3	P1	P1	P1	P1	P1	
0	4	7	10	14	18	22	26	30

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

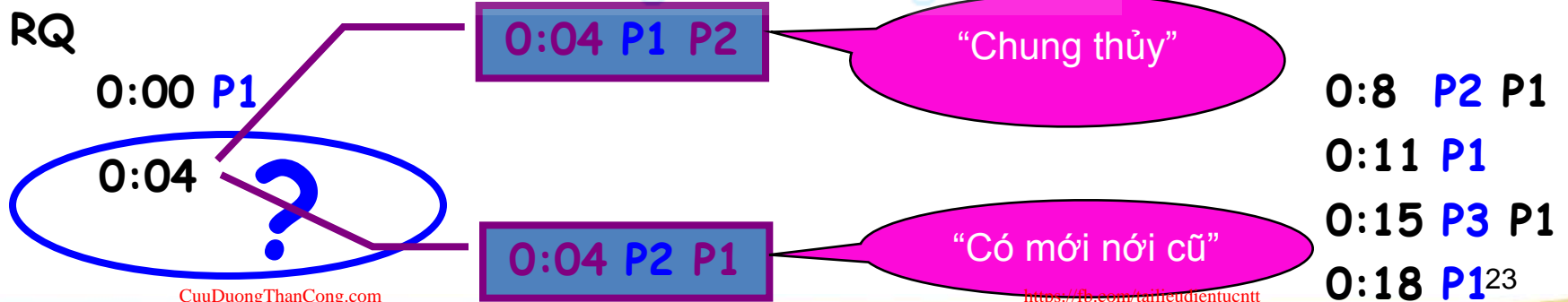
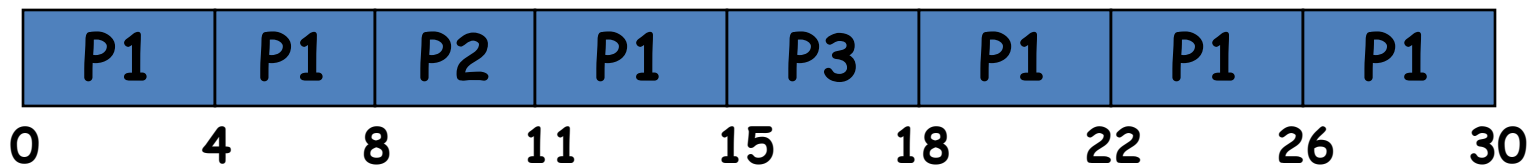
...



# Minh họa RR, q=4

P	T <sub>arriveRQ</sub>	CPU burst
P1	0	24
P2	4	3
P3	12	3

- Tranh chấp vị trí trong RQ:  
“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...



# Nhận xét RR

- Loại bỏ hiện tượng độc chiếm CPU
- Phù hợp với hệ thống tương tác người dùng
- Hiệu quả ? Phụ thuộc vào việc lựa chọn quantum  $q$ 
  - $q$  quá lớn  $\Rightarrow$  FCFS (giảm tính tương tác)
  - $q$  quá nhỏ  $\Rightarrow$  chủ yếu thực hiện chuyển đổi ngữ cảnh (context switching)
  - Thường  $q = 10\text{-}100$  milliseconds



# Điều phối với độ ưu tiên

- Một độ ưu tiên (integer) được gán vào mỗi tiến trình
- Phân biệt tiến trình quan trọng với tiến trình bình thường
- Tiêu chí lựa chọn tiến trình
  - Tiến trình có độ ưu tiên cao nhất
- Thời điểm lựa chọn tiến trình
  - Độc quyền
  - Không độc quyền (có độ ưu tiên)



# Minh họa điều phối với độ ưu tiên (không độc quyền)

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

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

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



0:00 P1 vào, P1 dùng CPU

0:01 P2 vào (độ ưu tiên cao hơn P1)

P2 dành quyền dùng CPU

0:02 P3 vào (độ ưu tiên thấp hơn P2)

P3 không dành được quyền dùng CPU

0:4 P2 kết thúc, P3 dùng CPU

0:7 P3 dùng, P1 dùng CPU

0:30 P1 dùng

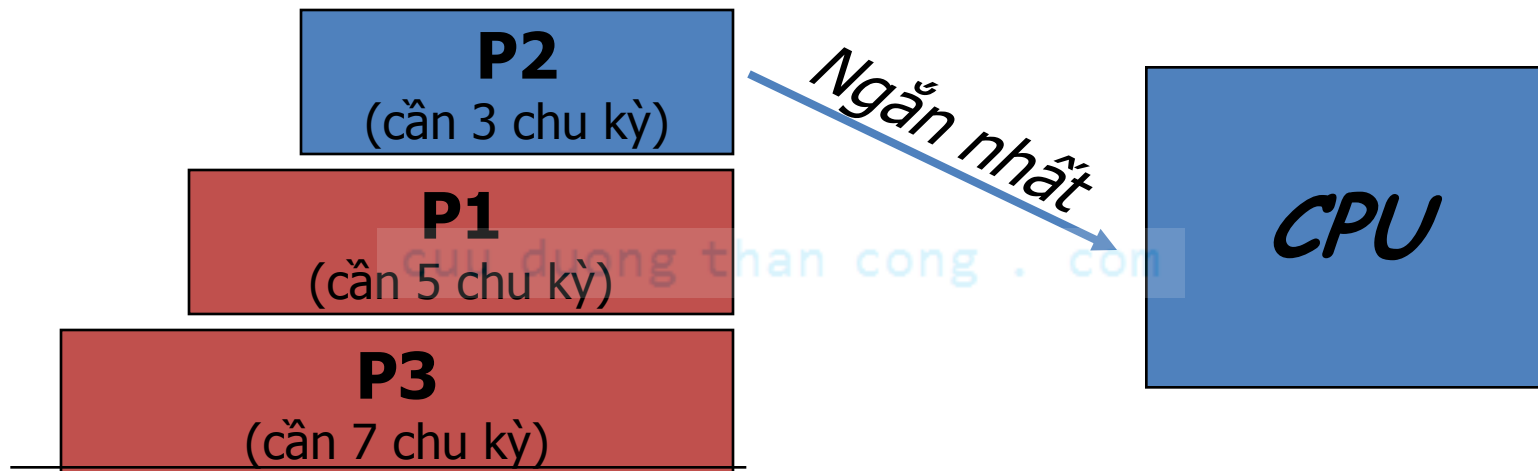
# Nhận xét điều phối với độ ưu tiên

- Cách tính độ ưu tiên ?
  - Hệ thống gán
  - Người dùng gán
- Tính chất độ ưu tiên
  - Tĩnh
    - Vấn đề Starvation: các tiến trình độ ưu tiên thấp có thể không bao giờ thực thi được
    - Giải pháp Aging – tăng độ ưu tiên cho tiến trình chờ lâu trong hệ thống (sống lâu lên lão làng...)
  - Động



# Shortest Job First (SJF)

## Ready Queue



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 (không độc quyền)

P	$T_{\text{arriveRQ}}$	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

$$\text{Avg}_{WT} = (6 + 0 + 2) / 3 = 2.66$$



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



# Nhận xét SJF

- Tối ưu thời gian chờ
- Làm sao biết được thời gian còn lại cần thực thi của một tiến trình
  - Không khả thi ?
  - Ước lượng – sử dụng thời gian sử dụng CPU ngay trước, dùng qui luật trung bình giảm theo hàm mũ.

$$\tau_{n+1} = \alpha t_n + (1 - \alpha) \tau_n$$

length of the  $n$ th  
CPU burst

$0 \leq \alpha \leq 1$

predicted value for  
the  $n$ th CPU burst

most recent  
information

relative weight

past history

# Ước lượng thời gian sử dụng CPU

- $\alpha = 0$ 
  - $\tau_{n+1} = \tau_n$
  - Không xét các giá trị thực đã dùng CPU.
- $\alpha = 1$ 
  - $\tau_{n+1} = t_n$
  - Chỉ dùng giá trị thực mới dùng CPU.
- Nếu mở rộng công thức, ta được:
  - $\tau_{n+1} = \alpha t_n + (1 - \alpha) \alpha t_n - 1 + \dots$
  - $+ (1 - \alpha)^j \alpha t_n - 1 + \dots$
  - $+ (1 - \alpha)^{n=1} t_n \tau_0$
- Vì  $\alpha$  và  $(1 - \alpha)$  nhỏ hơn hay bằng 1, mỗi giá trị kế tiếp sẽ nhỏ dần.



# Bài tập

- Thực hiện điều phối theo chiến lược FCFS, RR với  $q = 4$  và SJF
- Tính WT

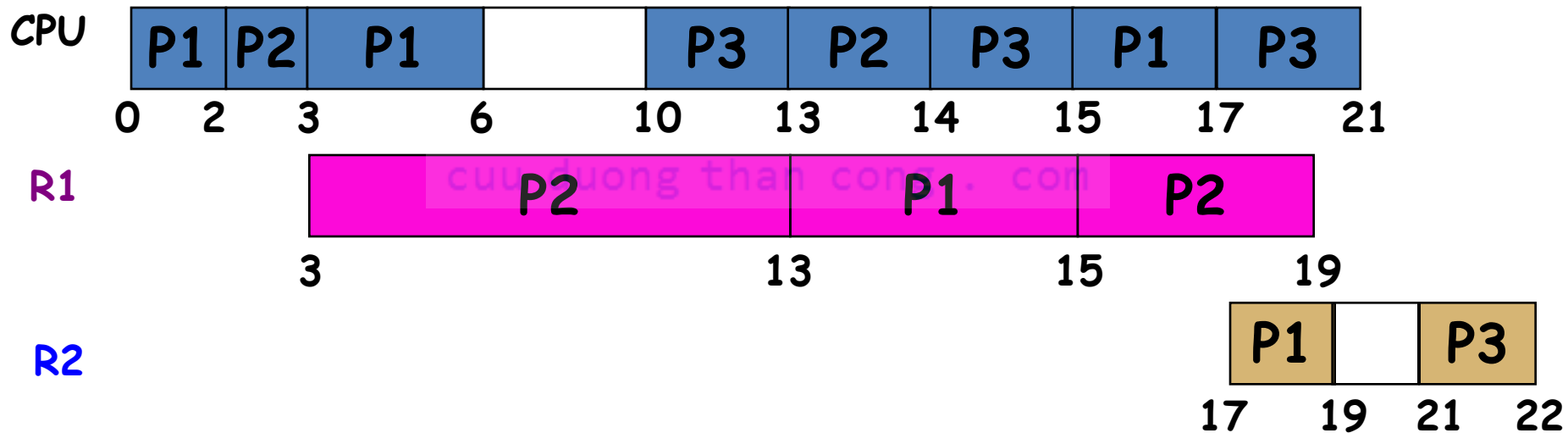
P	$T_{arriveRL}$	CPU burst
P1	0	20
P2	1	5
P3	2	3
P4	12	6



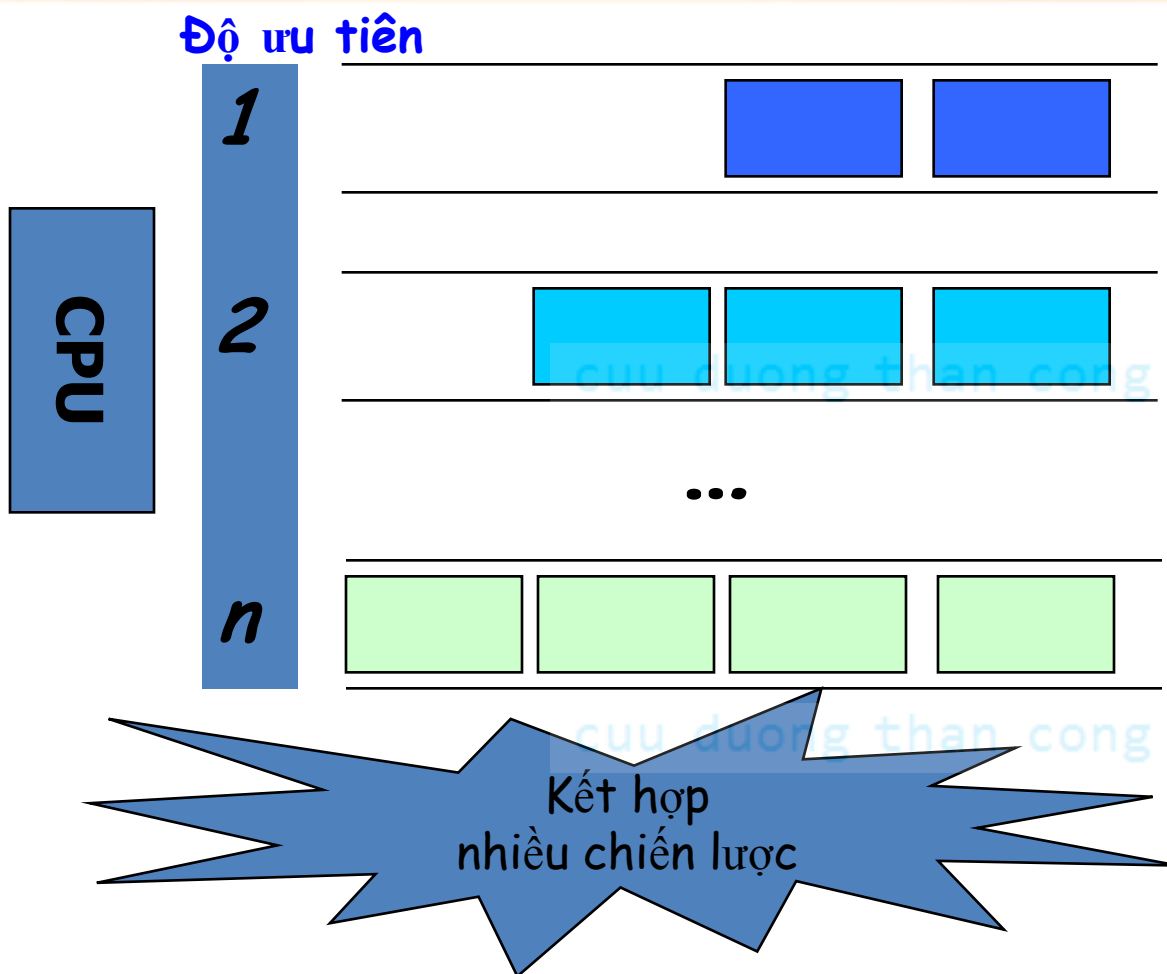
# Minh họa điều phối nhiều chu kỳ CPU

## CPU: SJF; IO:FCFS

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



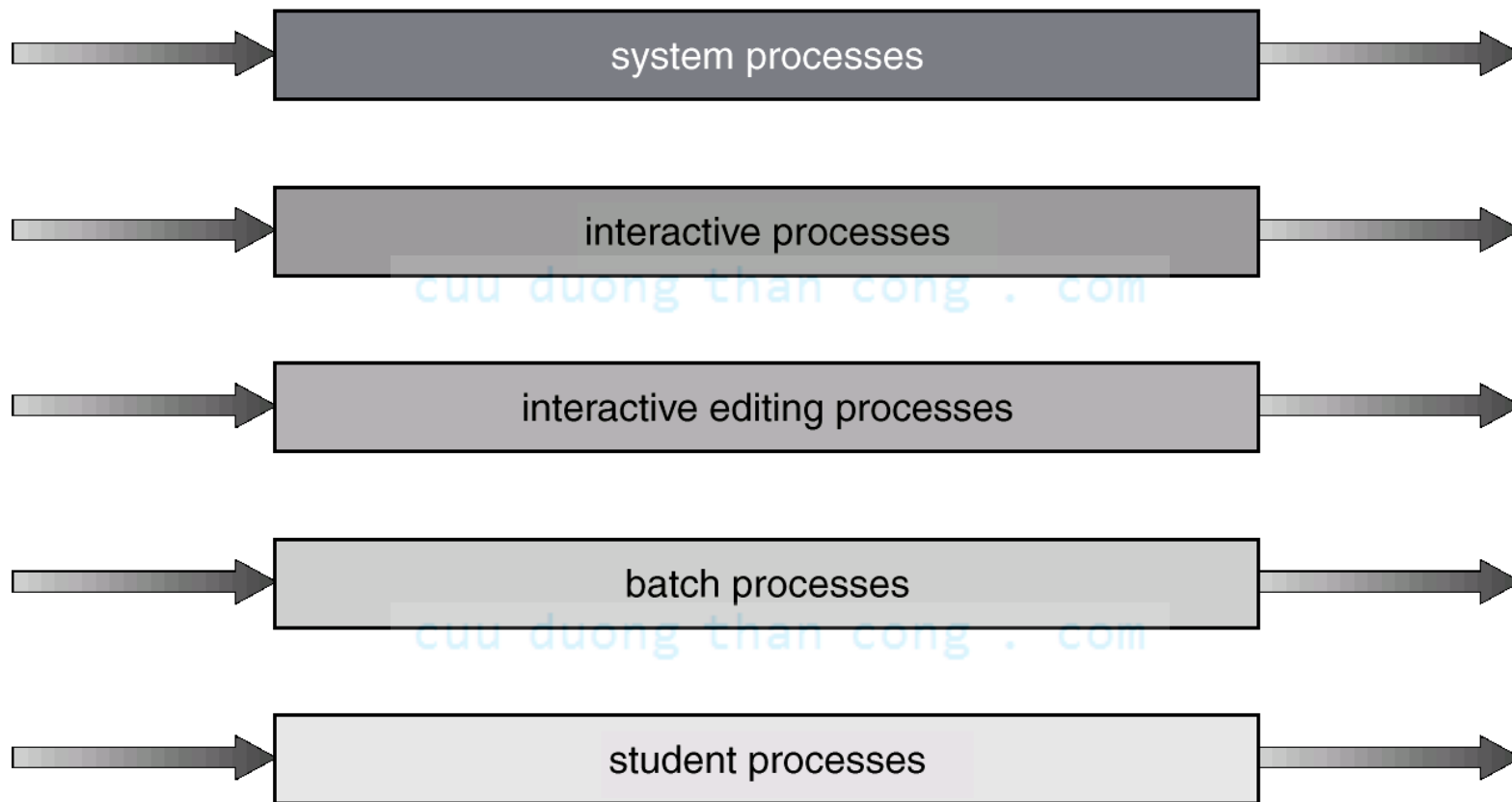
# Điều phối sử dụng nhiều hàng đợi



- Tổ chức  $n$  RQ ứng với nhiều mức ưu tiên
- Mỗi  $RQ_i$  có thể áp dụng một chiến lược điều phối riêng
- Điều phối giữa các RQ:
  - Điều phối theo độ ưu tiên của RQ
    - Có thể xảy ra starvation
    - Giải pháp Aging :
      - Chờ lâu quá : chuyển lên RQ với độ ưu tiên cao hơn
      - Chiếm CPU lâu quá : chuyển xuống RQ với độ ưu tiên thấp hơn
  - Time slice – mỗi hàng đợi nhận chiếm CPU một khoảng thời gian

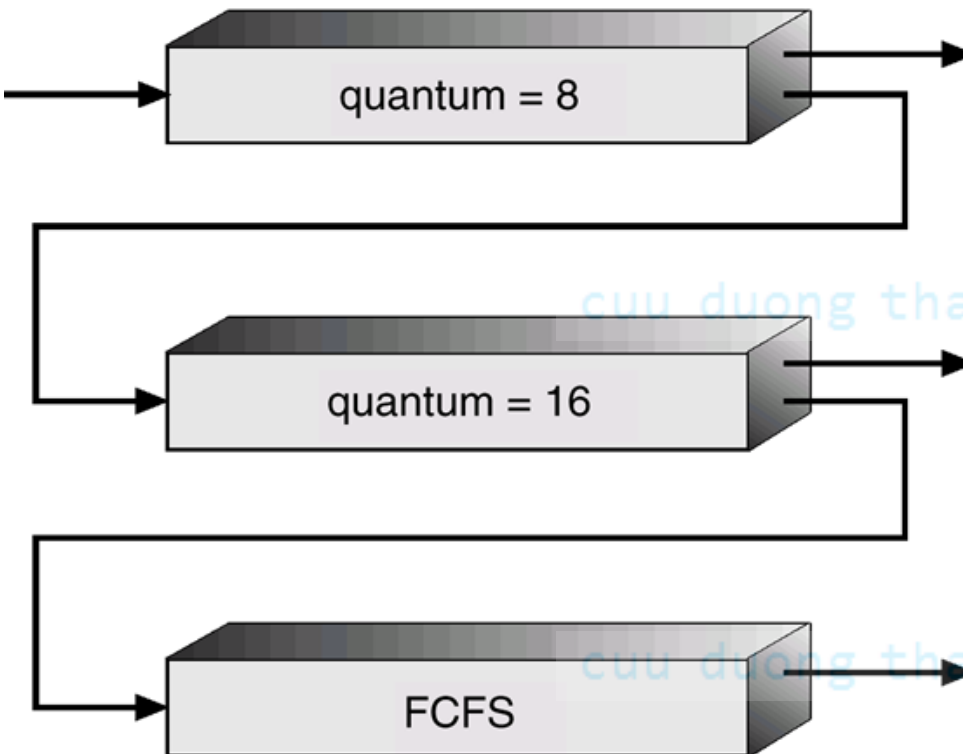
# Một mô hình điều phối nhiều hàng đợi

highest priority



lowest priority

# Ví dụ điều phối nhiều hàng đợi



- Ba hàng đợi:

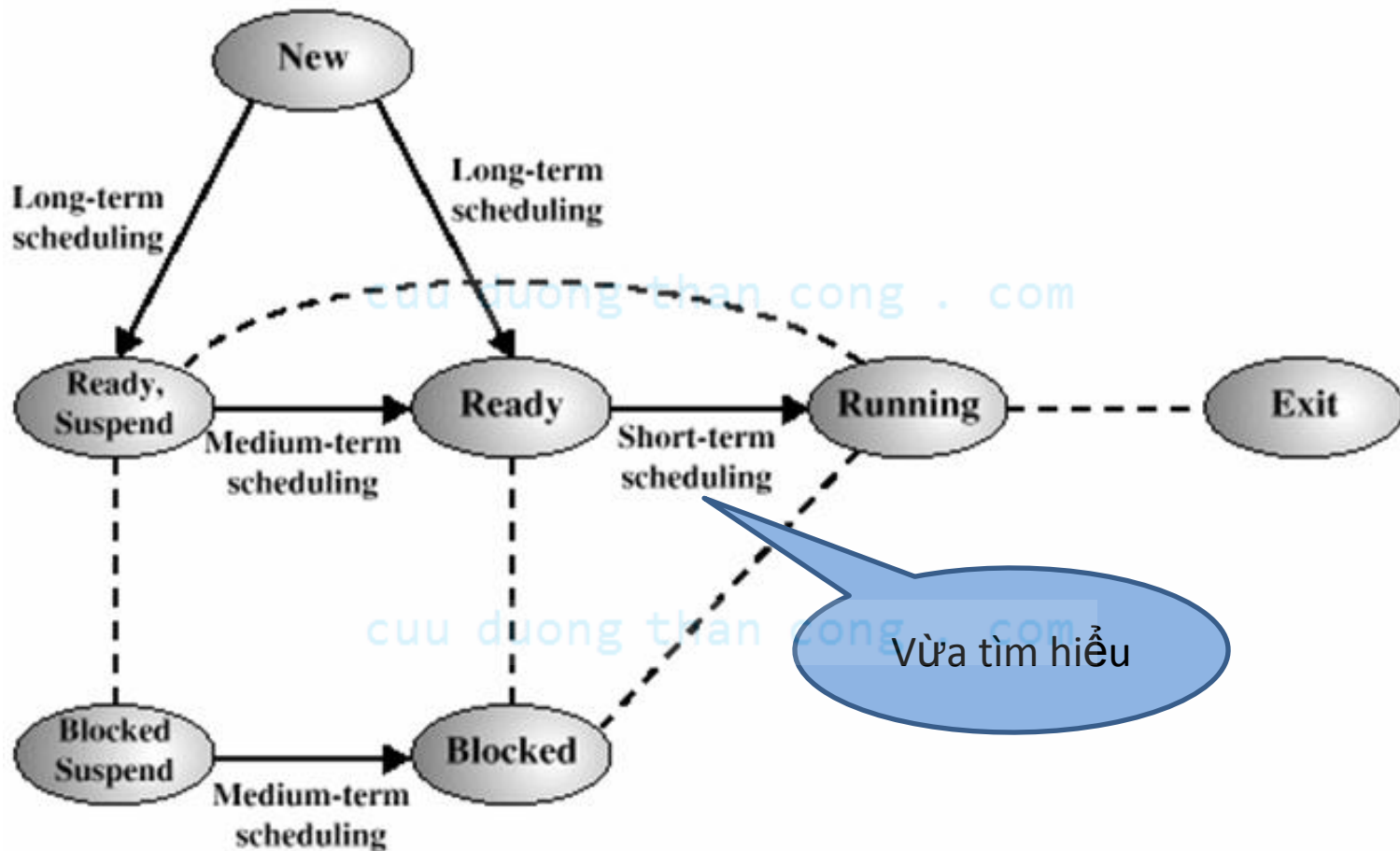
- $Q_0$  – time quantum 8 mili giây
- $Q_1$  – time quantum 16 mili giây
- $Q_2$  – FCFS

## Lập lịch

- Một việc mới vào queue  $Q_0$  nó được điều phối theo FCFS. Khi nó nhận CPU, chỉ dùng trong 8 mili giây. Nếu chưa hoàn tất trong 8 mili giây, công việc chuyển sang queue  $Q_1$ .
- Tại  $Q_1$  cv được điều phối theo FCFS và nhận CPU thêm 16 mili giây. Nếu vẫn chưa hoàn tất, nó sẽ bị đẩy qua queue  $Q_2$ .



# Một mô hình điều phối và chuyển trạng thái của tiến trình



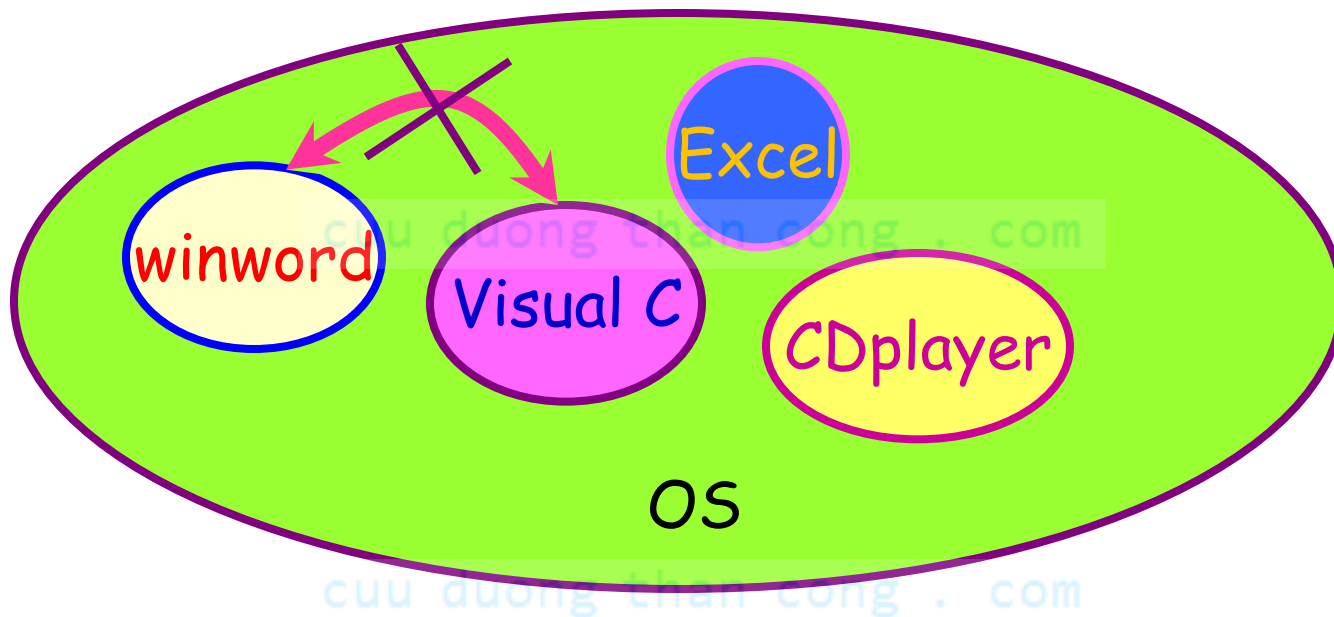
# Bài toán xử lý đồng thời

- Đơn giản
  - Viết chương trình theo kiểu bấm giờ bằng cách mỗi lần nhấn một phím thì sẽ in ra số nhip đã trôi qua từ khi chương trình bắt đầu chạy, với mỗi nhip bằng 1 giây, chương trình sẽ kết thúc khi nhấn ESC.
- Phức tạp hơn
  - Viết chương trình hiển thị 10 ký tự trên màn hình, mỗi ký tự di chuyển ngẫu nhiên, liên tục với tốc độ tùy ý. Khi di chuyển nếu chạm biên màn hình thì ký tự sẽ xuất hiện lại tại giữa màn hình. Chương trình kết thúc khi người dùng nhấn phím bất kỳ.
- Bài toán thực tế:
  - Ứng dụng web phục vụ cùng lúc nhiều yêu cầu người dùng



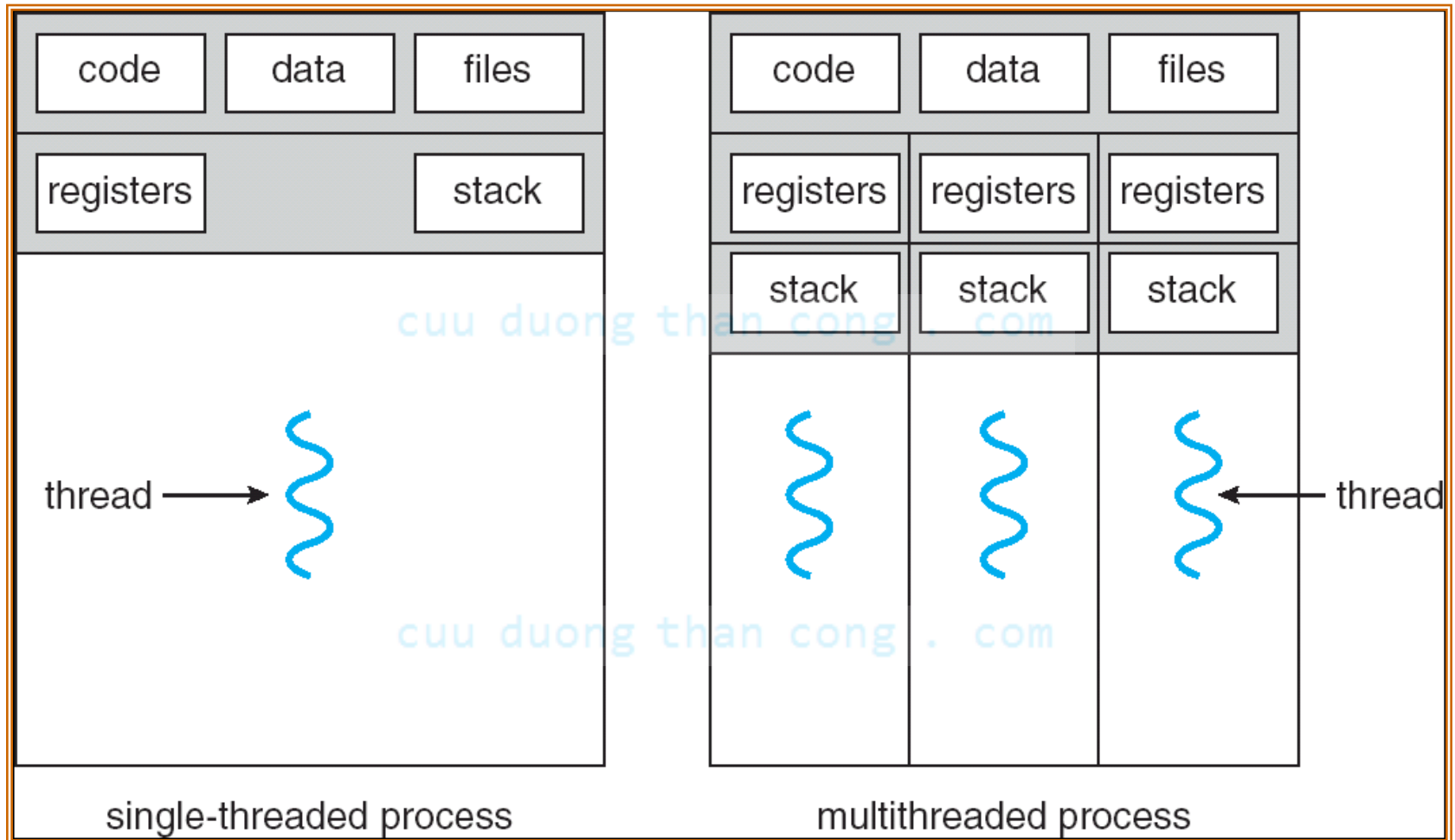
# Giải pháp đa tiến trình

- Các tiến trình độc lập, không có sự liên lạc với nhau



- Muốn trao đổi thông tin với nhau, các chương trình cần được xây dựng theo mô hình liên lạc đa tiến trình (IPC – Inter-Process Communication) → Phức tạp, chi phí cao

# Giải pháp đa tiêu trình



# Tiểu trình

- Tiểu trình là một dòng xử lý trong một tiến trình
- Mỗi tiến trình luôn có một tiểu trình chính (dòng xử lý cho hàm main())
- Ngoài tiểu trình chính, tiến trình còn có thể có nhiều tiểu trình con khác
- Các tiểu trình của một tiến trình
  - Chia sẻ không gian vùng code và data
  - Có vùng stack riêng

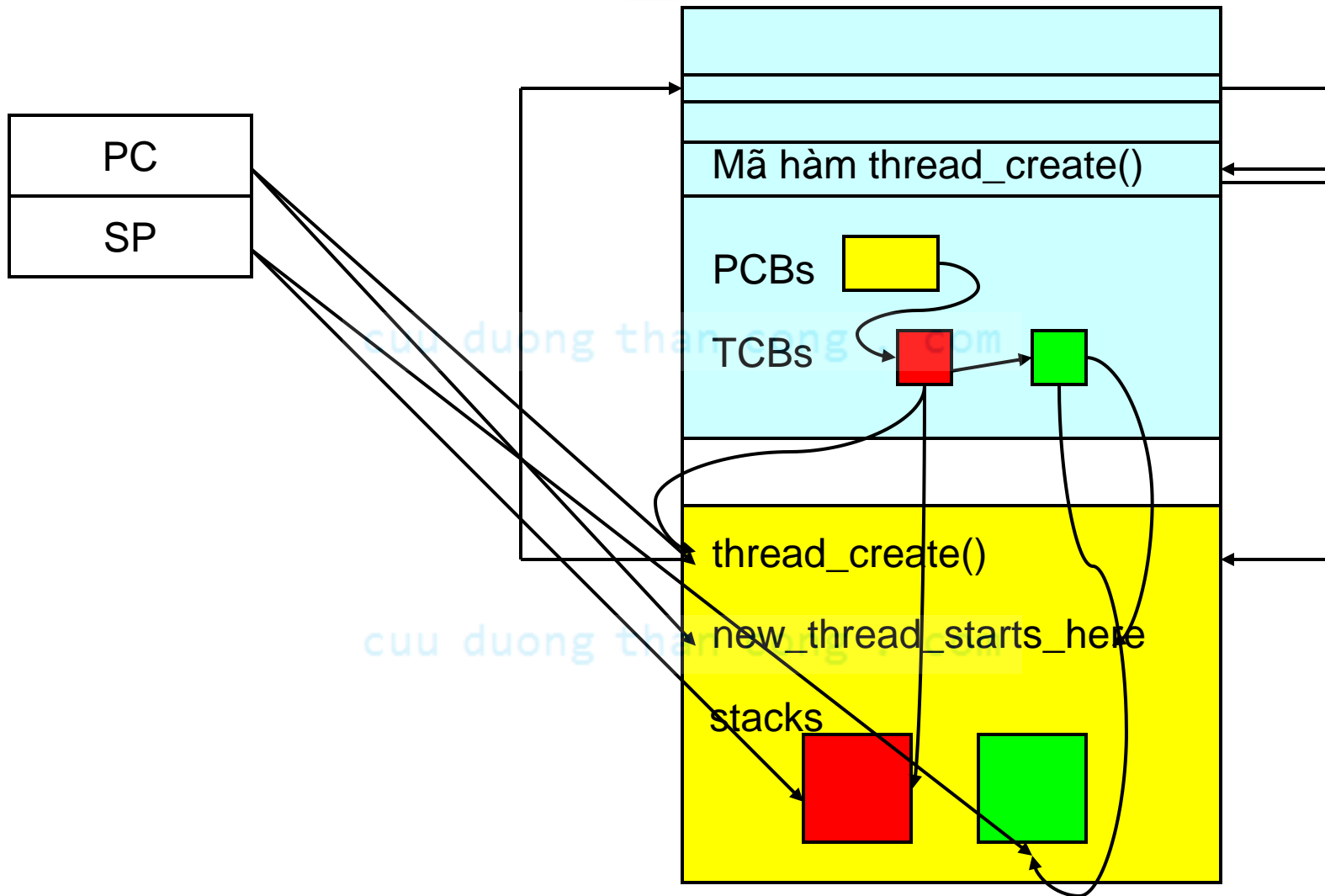


# Khối quản lý tiểu trình (Thread Control Block – TCB)

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

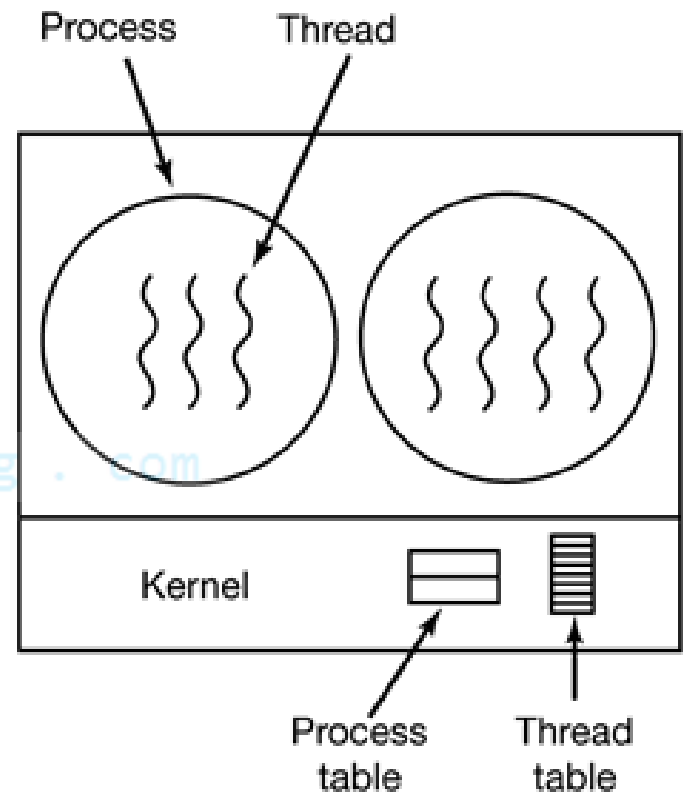
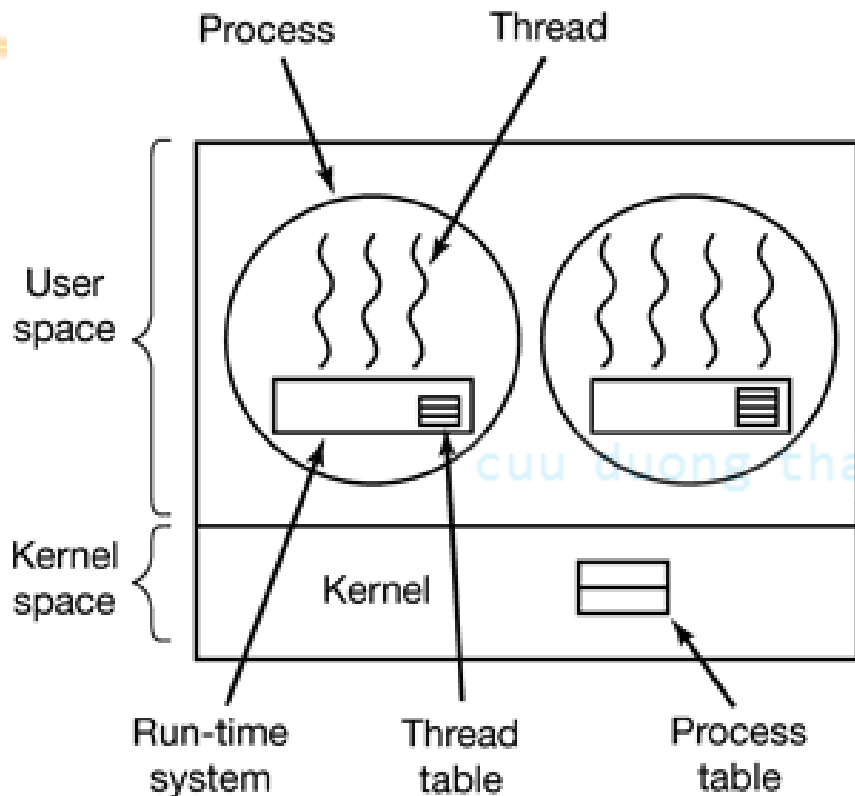


# Tạo tiêu trình





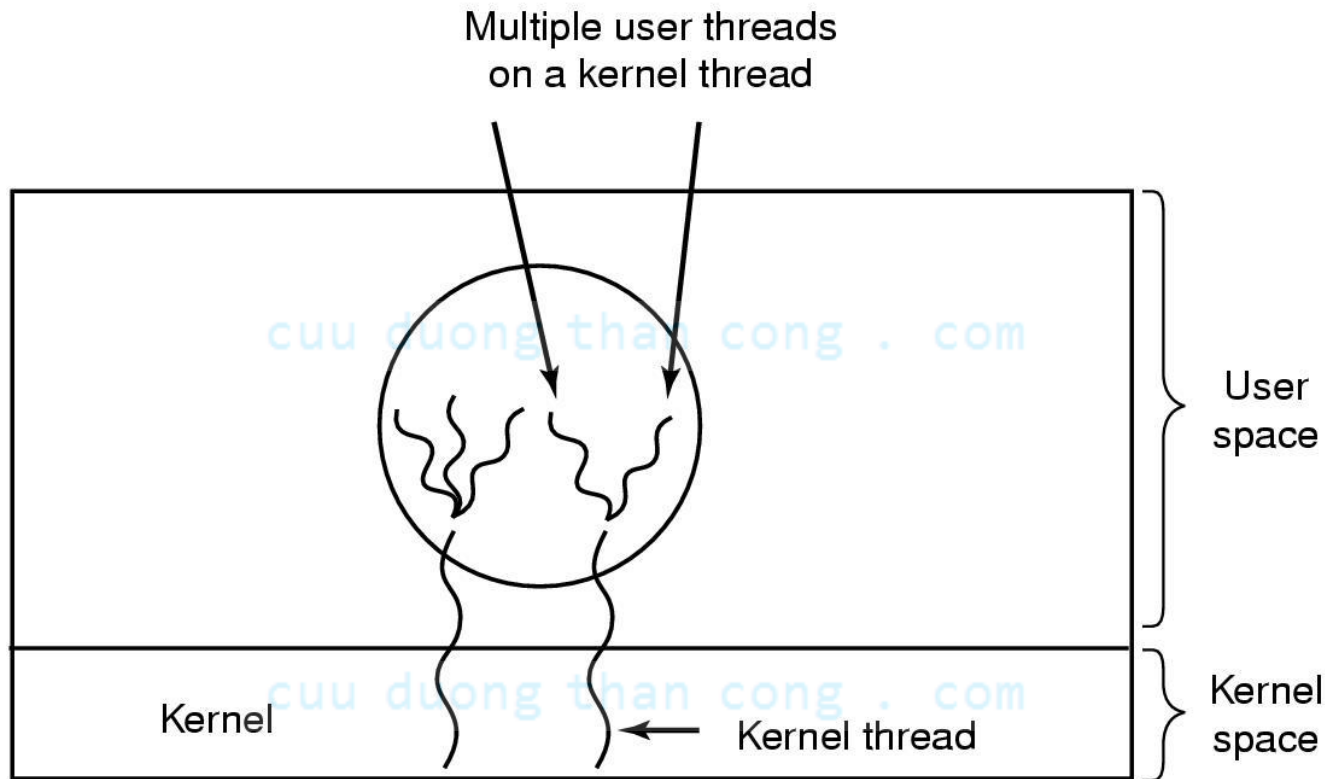
# Tổ chức tiến trình



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

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

# Mô hình kết hợp



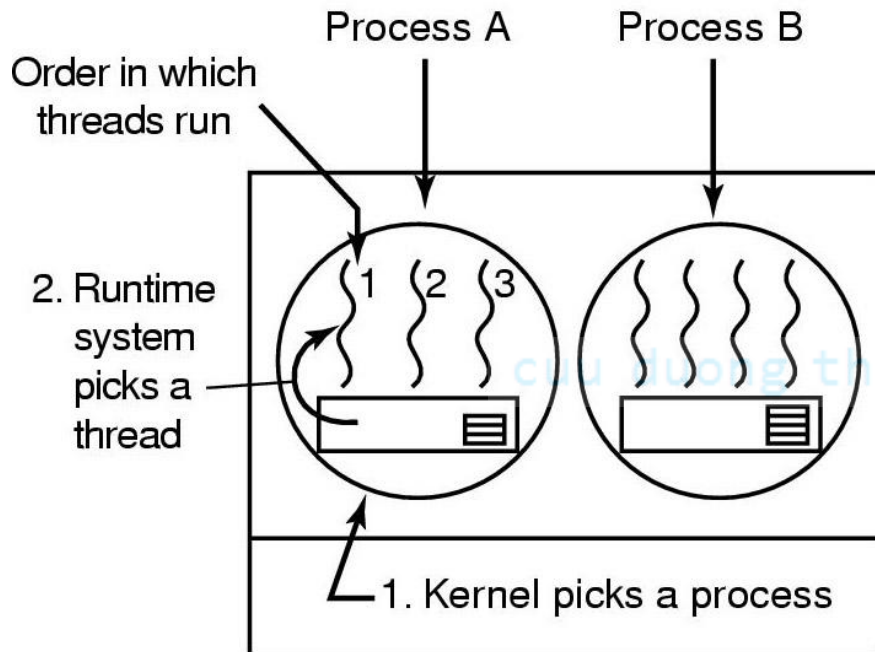
# So sánh tiểu trình và tiến trình

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

cuu duong than cong . com



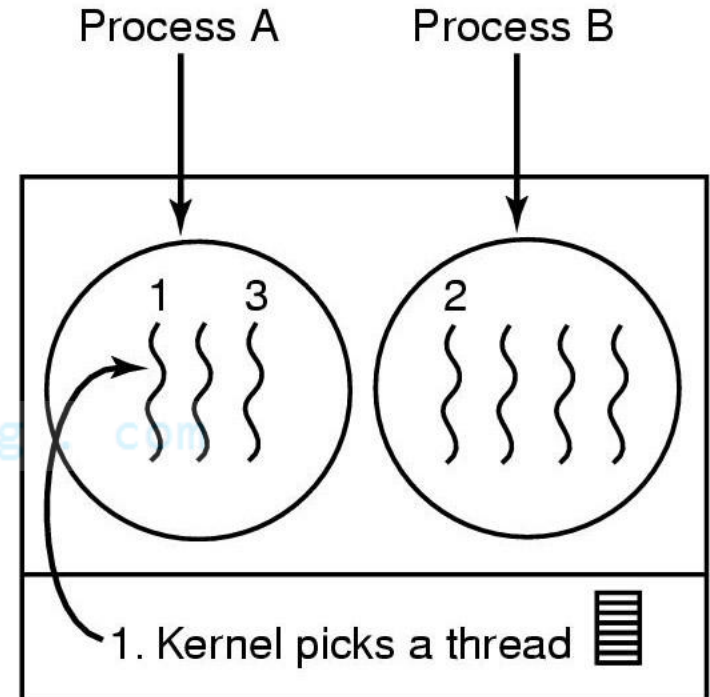
# Điều phối tiến trình



Possible: A1, A2, A3, A1, A2, A3

Not possible: A1, B1, A2, B2, A3, B3

Quản lý tiến trình mức người dùng



Possible: A1, A2, A3, A1, A2, A3

Also possible: A1, B1, A2, B2, A3, B3

Quản lý tiến trình mức hệ thống



- Tự tìm hiểu  
(*Modern Operating System - Tanenbaum*)
  - Guaranteed Scheduling
  - Lottery Scheduling
  - Fair-Share Scheduling
- Tham khảo thêm
  - Tổ chức, quản lý tiến trình của hệ điều hành Windows
  - Tổ chức, quản lý tiến trình của hệ điều hành Unix/Linux

