ĐỊNH THỜI CPU
Mục tiêu

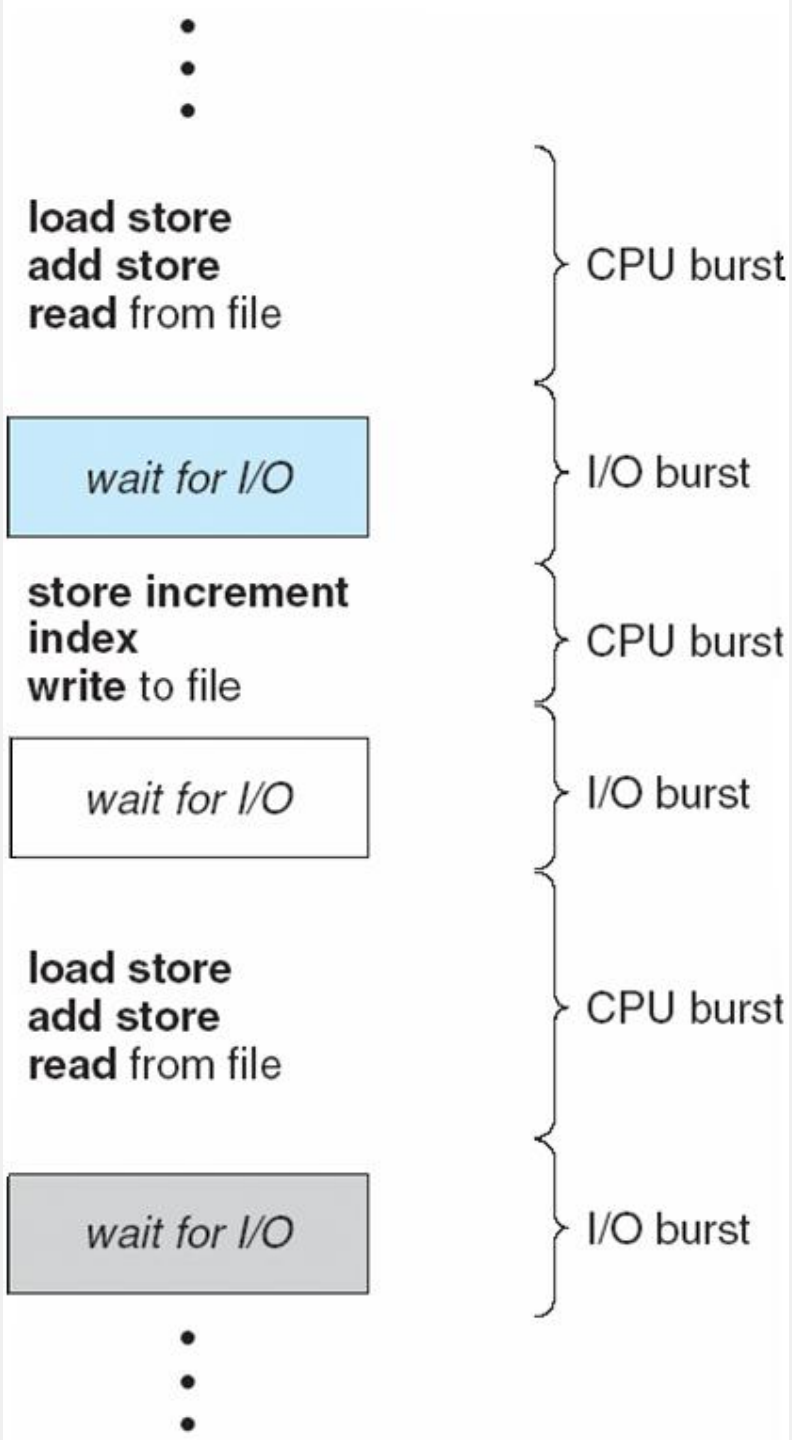
- SV sẽ giải thích được
  - Tại sao cần phải định thời
  - Các tiêu chí định thời
  - Một số giải thuật định thời

Vấn đề cần giải quyết

- Ví dụ: tối đa độ lợi CPU trong hệ thống multiprogramming / multitasking
  - Trong bộ nhớ có nhiều process đồng thời
  - Khi một process phải đợi (vd đợi hoàn tất yêu cầu I/O), hệ điều hành cấp phát CPU cho một quá trình khác thay vì để CPU chạy không (idle)
- Vấn đề: lựa chọn process thực thi sao cho được "hiệu quả" nhất --- tối ưu đối với một tiêu chí cho trước. Cần có chiến lược định thời CPU

Phân loại quá trình

- Quá trình thực thi qua nhiều chu kỳ thực thi CPU và đợi I/O
  - CPU burst
  - I/O burst
- *CPU-bound* process có thời gian sử dụng CPU nhiều hơn thời gian sử dụng I/O
- *I/O-bound* process dùng phần lớn thời gian để đợi I/O

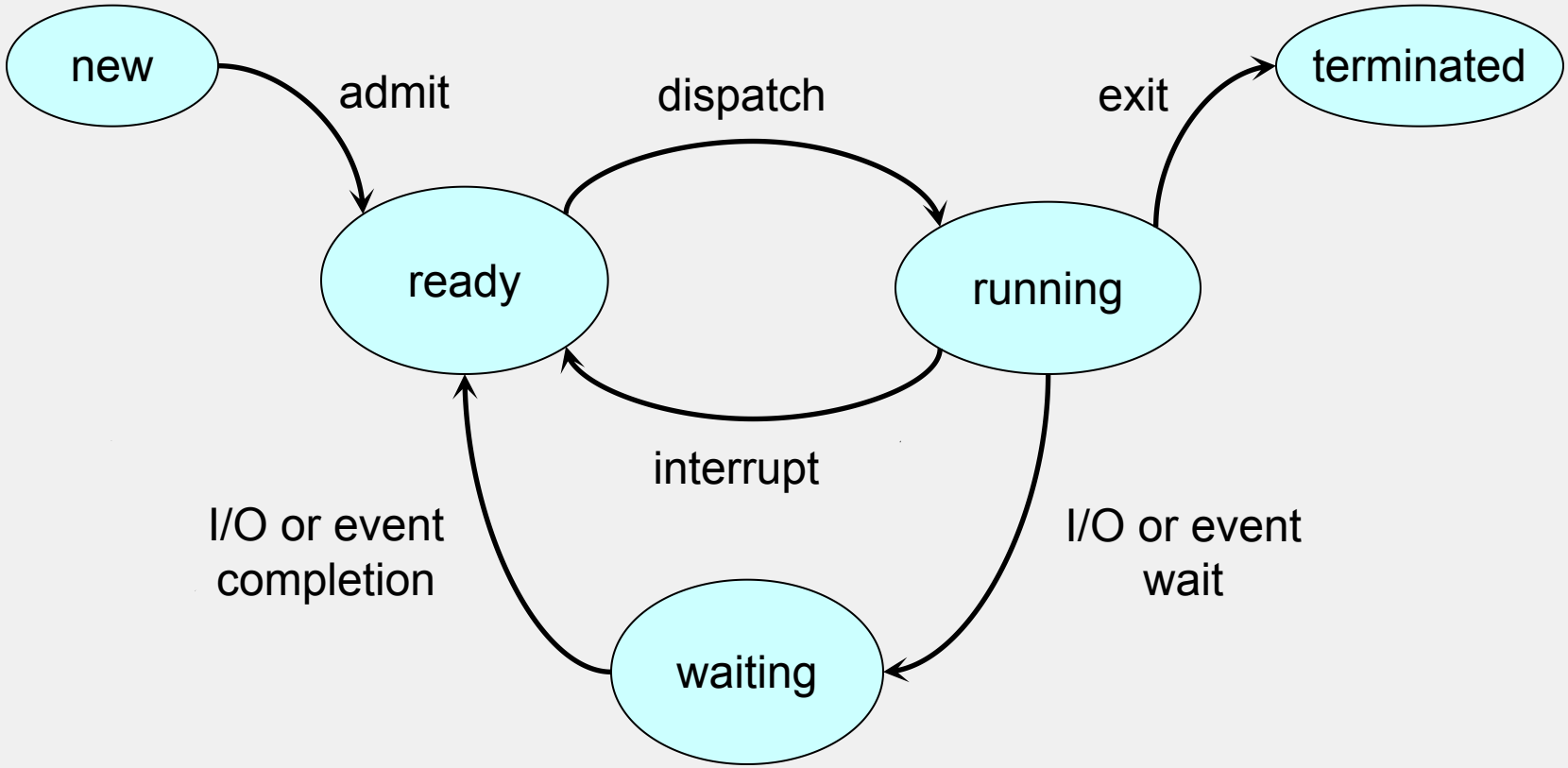


- CPU-bound process: CPU burst dài

- I/O-bound process: CPU burst ngắn

Các trạng thái của quá trình – nhắc lại

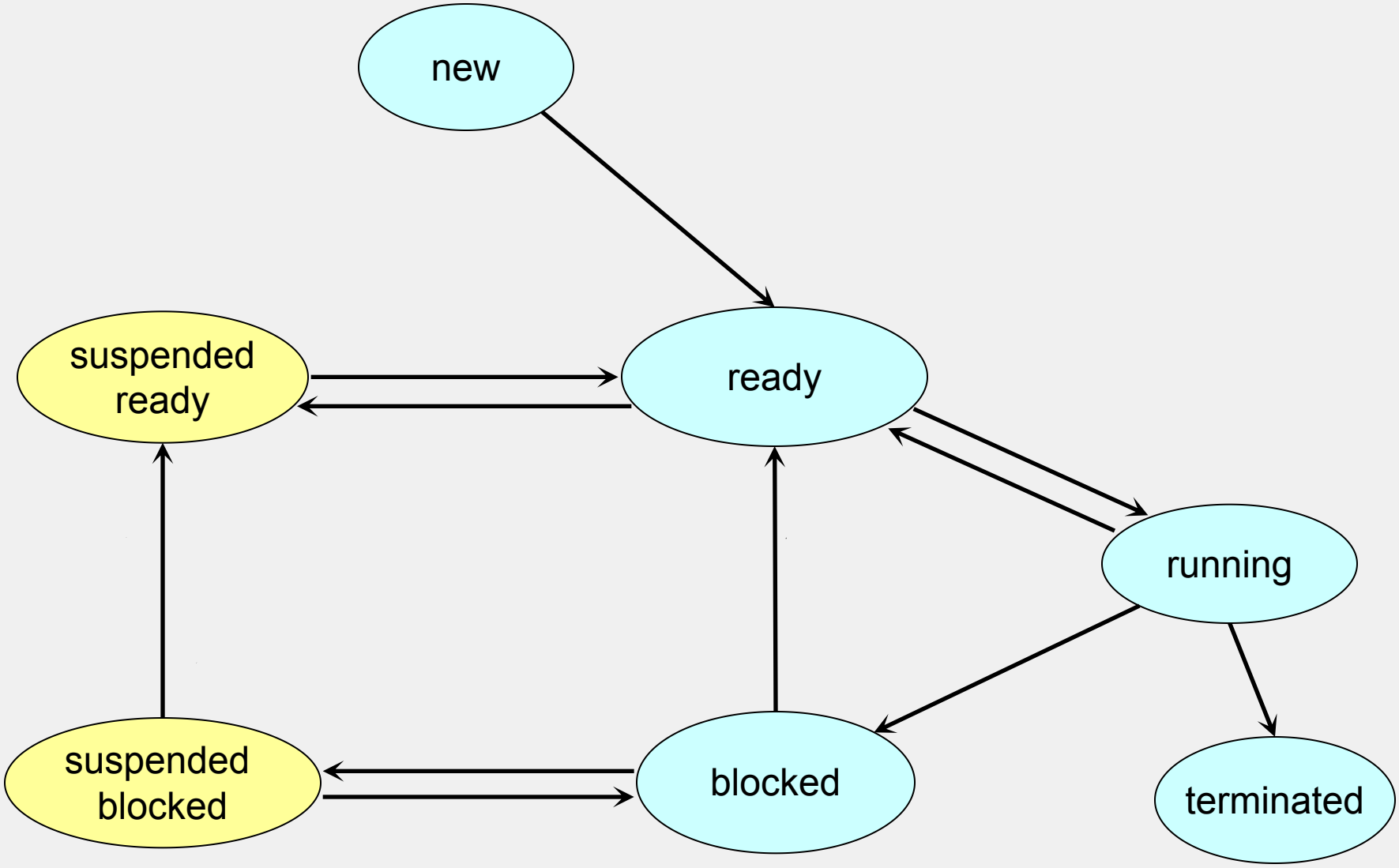
- Sơ đồ chuyển trạng thái của quá trình – sơ đồ 5 trạng thái



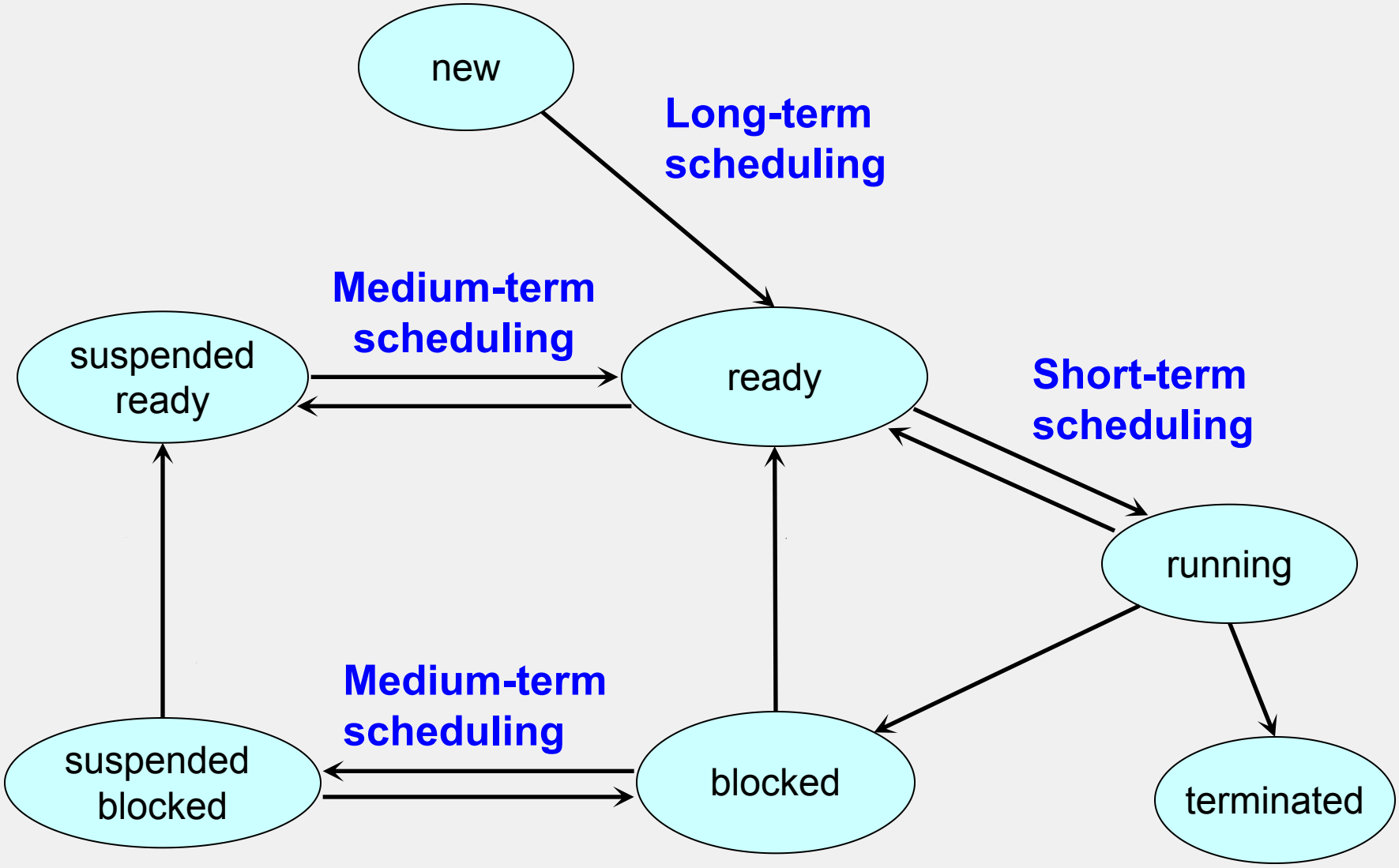
Mở rộng các trạng thái của quá trình

- Làm thế nào để điều chỉnh số lượng quá trình đang ở trong bộ nhớ chính?
  - *Swap out* / *swap in* quá trình (cần phân biệt với paging!) -- Cơ chế
    - Swap out: đem quá trình từ bộ nhớ chính ra đĩa
    - Swap in: đem quá trình từ đĩa vào bộ nhớ chính
  - Chọn quá trình nào để swap out / swap in -- Chính sách
  - Hai trạng thái mới cho quá trình
    - *suspended ready*
    - *suspended blocked*

Mở rộng các trạng thái của quá trình



Phân loại các hoạt động định thời (1/2)



Phân loại các hoạt động định thời (2/2)

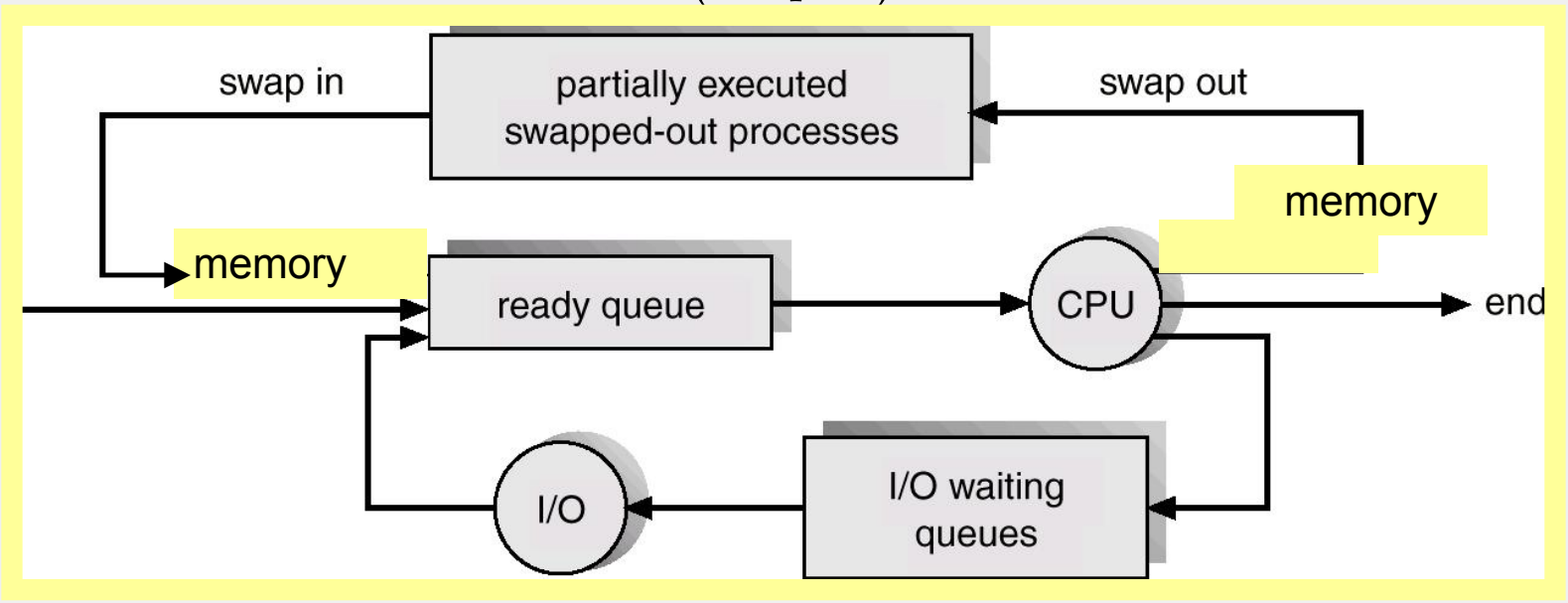
- *Định thời dài hạn* (long-term scheduling): xác định các process mới (new) nào được đưa vào hàng đợi sẵn sàng
  - Thường chỉ có trong batch system
- *Định thời trung hạn* (medium-term scheduling): xác định process nào được đưa ra khỏi (*swap out*) hay được đưa vào (*swap in*) bộ nhớ chính
  - Swap out / in có thể tốn đến vài giây thời gian đ chu kỳ định thời trung hạn có thể là vài phút
- *Định thời ngắn hạn* (short-term scheduling): xác định process nào được thực thi tiếp theo

Định thời dài hạn

- Ảnh hưởng đến *độ-đa-lập-trình* (degree of multiprogramming: số lượng quá trình đang ở trong bộ nhớ)
- Nếu càng nhiều process đang ở trong bộ nhớ thì khả năng mọi process bị block có xu hướng giảm
  - Sử dụng CPU hiệu quả hơn
  - Nhưng mỗi process được phân chia khoảng thời gian sử dụng CPU nhỏ hơn
- Thường có xu hướng đưa vào một tập lẫn lộn các CPU-bound process và I/O-bound process

Định thời trung hạn

- Đôi khi hệ điều hành (như time-sharing system) có thêm medium-term scheduling
  - *Medium-term scheduler* quyết định việc đưa process (không phải process ở trạng thái new)
    - từ bộ nhớ chính ra đĩa (swap out) hay
    - từ đĩa vào bộ nhớ chính (swap in)



Định thời trung hạn

- Phụ thuộc vào yêu cầu quản lý việc đa-lập-trình (multiprogramming)
  - Cho phép bộ định thời dài hạn chấp nhận (admit) nhiều process hơn số lượng process mà có tổng kích thước được chứa vừa trong bộ nhớ chính (☾ kỹ thuật bộ nhớ ảo)
  - Nhưng nếu có quá nhiều process thì sẽ làm tăng việc truy xuất đĩa (☾ thrashing), do đó cần phải lựa chọn độ-đa-lập-trình cho phù hợp
- Được thực hiện bởi phần mềm quản lý bộ nhớ

Định thời trung hạn

- Tùy theo chiến lược sử dụng bộ nhớ, medium-term scheduler có thể quyết định
  - swap out quá trình chiếm khối lượng lớn bộ nhớ, hay có độ ưu tiên thấp
  - swap in quá trình khi có đủ bộ nhớ…

Định thời ngắn hạn

- Xác định process nào được thực thi tiếp theo, còn gọi là *định thời CPU*
- Tùy hệ thống (định thời non-preemptive, preemptive) mà được kích hoạt khi có một sự kiện dẫn đến khả năng chọn một process để thực thi
  - Ngắt thời gian (clock interrupt)
  - Ngắt ngoại vi (I/O interrupt)
  - Lời gọi hệ thống (operating system call)
  - Signal

Chương này sẽ tập trung vào định thời ngắn hạn

Nội dung cần quan tâm

- Định thời trên hệ thống có một processor (uniprocessor scheduling): quyết định việc sử dụng (một) CPU cho một tập các process trong hệ thống
- Tiêu chí nào?

Tiêu chí định thời (1/4)

- *Độ lợi CPU* (CPU utilization)
  - Khoảng thời gian CPU bận tính toán cho ứng dụng, từ 0% đến 100%
- *Thời gian chờ* (waiting time)
  - Thời gian một process ở trong hàng đợi ready

Tiêu chí định thời (2/4)

- *Thông năng* (throughput)
  - Số lượng process hoàn tất trong một đơn vị thời gian
- *Thời gian đáp ứng* (response time)
  - Thời gian từ lúc có yêu cầu của người dùng (user request) đến khi có đáp ứng đầu tiên đến người dùng
  - Thường là vấn đề với các I/O-bound process

Tiêu chí định thời (3/4)

- *Thời gian quay vòng* (turnaround time)
  - Thời gian để một process hoàn tất, kể từ lúc vào hệ thống (submission) đến lúc kết thúc (termination)
  - Là một trị đặc trưng cần quan tâm đối với các process thuộc dạng CPU-bound
- *Thời gian quay vòng trung bình* (average turnaround time)

Tiêu chí định thời (4/4)

- Độ lợi CPU
  - Tối đa hóa – giữ CPU càng bận tính toán cho ứng dụng càng tốt
- Thông năng – số lượng process kết thúc việc thực thi trong một đơn vị thời gian
  - Tối đa hóa
- Turnaround time – thời gian kể từ lúc đưa vào (submission) đến lúc kết thúc
  - Tối thiểu hóa
- Thời gian chờ – thời gian một process chờ trong hàng đợi ready
  - Tối thiểu hóa
- Thời gian đáp ứng – thời gian từ khi đưa yêu cầu đến khi có đáp ứng đầu tiên
  - Tối thiểu hóa

Có thể làm được?

- Tất cả các tiêu chí không thể được tối ưu đồng thời vì có một số tiêu chí liên quan nhau

Tiêu chí định thời từ các góc nhìn (1/2)

- Hướng đến người sử dụng (user-oriented)
  - Thời gian quay vòng
    - Thời gian từ lúc submission đến lúc process kết thúc
    - Cần quan tâm đối với các hệ thống xử lý bó (batch system)
  - Thời gian đáp ứng
    - Cần quan tâm đối với các hệ thống tương tác (interactive system)

Tiêu chí định thời từ các góc nhìn (2/2)

- Hướng đến hệ thống (system-oriented)
  - Độ lợi CPU
  - Công bằng (fairness)
  - Thông năng: số process hoàn tất trong một đơn vị thời gian

Hai thành phần của chiến lược định thời (1/2)

- *Hàm lựa chọn* (selection function)
  - Xác định process nào trong ready queue sẽ được thực thi tiếp theo. Thường theo các tiêu chí như
    - w = tổng thời gian đợi trong hệ thống
    - e = thời gian đã được phục vụ
    - s = tổng thời gian thực thi của process (bao gồm cả trị e)

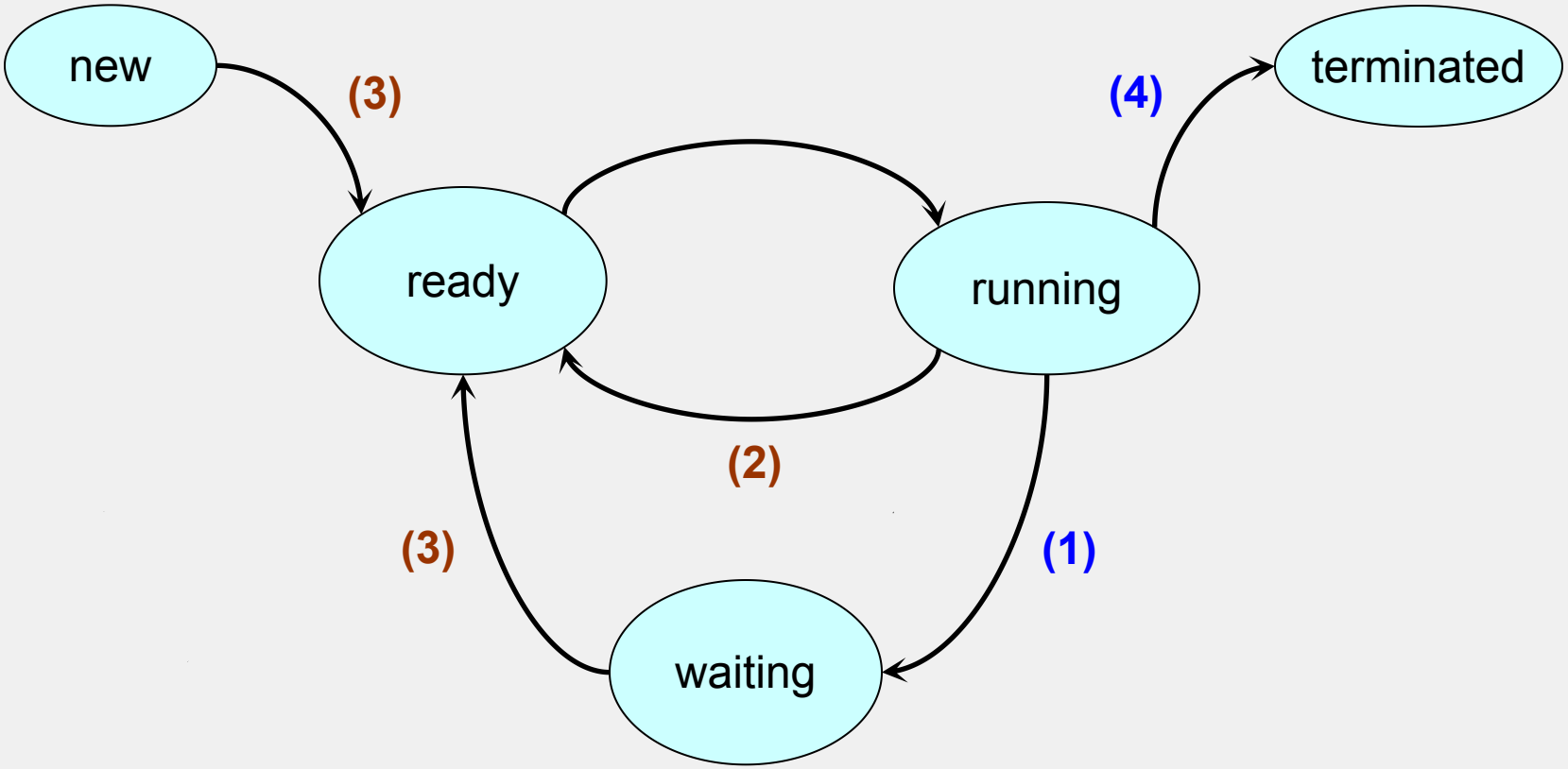
Hai thành phần của chiến lược định thời (2/2)

- *Chế độ quyết định* (decision mode)

  Xác định thời điểm hàm lựa chọn được thực thi. Hai chế độ quyết định

  - *Non-preemptive*
    - Một process sẽ ở trạng thái running cho đến khi nó bị block hoặc nó kết thúc
  - *Preemptive*
    - Process đang thực thi có thể bị ngắt và chuyển về trạng thái ready
    - Tránh trường hợp một process độc chiếm CPU

Thời điểm thực thi hàm lựa chọn



Non-preemption và preemption (1/2)

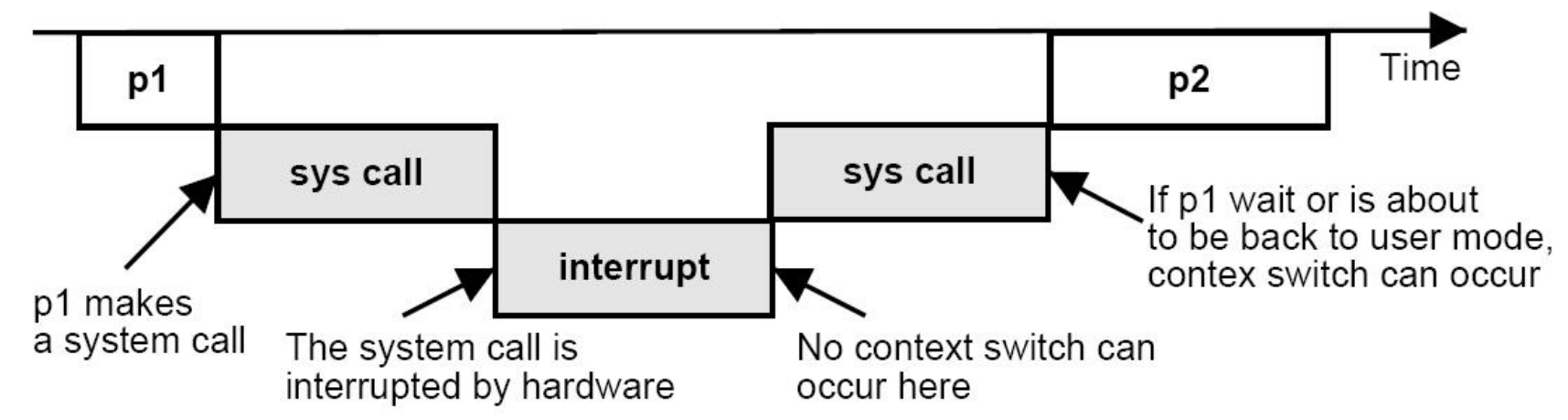
- Hàm lựa chọn có thể được thực thi khi có quá trình
  - **(1)** chuyển từ trạng thái running sang waiting
  - **(2)** chuyển từ trạng thái running sang ready
  - **(3)** chuyển từ trạng thái waiting, new sang ready

  } Có quá trình trở nên ready

  - **(4)** kết thúc thực thi
- Chiến lược định thời *non-preemptive*: chỉ thực thi hàm lựa chọn trong trường hợp 1 và 4 (quá trình running nếu bị ngắt sẽ tiếp tục running sau đó)
  - MS Windows 3.x
- Chiến lược định thời *preemptive*: ngoài trường hợp 1 và 4 còn thực thi thêm hàm lựa chọn trong trường hợp 2 hoặc 3 (hoặc cả hai)
  - Windows 95 (hay trễ hơn)

Non-preemption và preemption (2/2)

- Hiện thực chế độ quyết định nào khó hơn? Tại sao?
- Preemptive scheduling: không phải lúc nào preempt quá trình cũng được, còn tùy thuộc vào kernel
  - Non-preemptive kernel
    - Lời gọi hệ thống của quá trình được thực thi đến khi xong (trở về user space) hoặc block



from Isaac POS(0230A)

Non-preemption và preemption (2'/2)

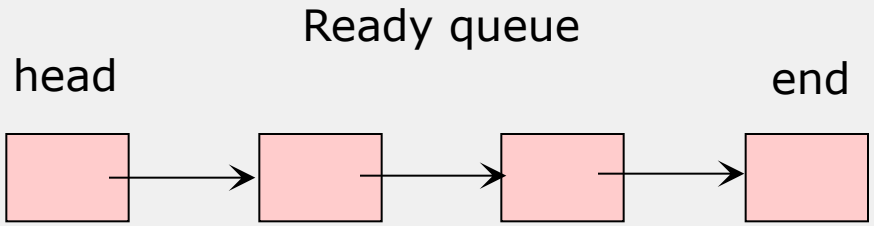
- (tt)
  - Preemptive kernel
    - Có thể preempt quá trình khi kernel đang thực thi một lời gọi hệ thống của nó
    - Vấn đề: giữ nhất quán các dữ liệu trong kernel (ví dụ các hàng đợi I/O)
  - Linux
    - Non-preemptive kernel: Linux 2.4
    - Preemptive kernel: Linux 2.6

Dispatcher

- Chức năng: chuyển quyền điều khiển CPU sang process được chọn bởi bộ định thời ngắn hạn
- Công việc bao gồm:
  - Chuyển ngữ cảnh (sử dụng thông tin ngữ cảnh trong PCB)
  - Chuyển về user mode
  - Nhảy đến địa chỉ thích hợp (chính là program counter trong PCB) trong vùng code của quá trình được tiếp tục thực thi
- Công việc này gây ra phí tổn
  - *Dispatch latency*: thời gian từ lúc dừng một process đến lúc một process khác tiếp tục chạy

First Come First Served (FCFS) (1/5)

- Hàm lựa chọn: chọn process ở trong hàng đợi ready lâu nhất

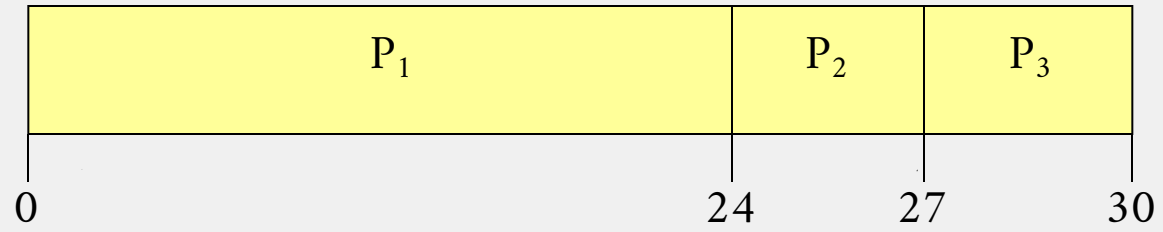


- Chế độ quyết định: non-preemptive
  - Một process sẽ được thực thi cho đến khi nó block hoặc kết thúc
- FCFS thường được hiện thực bằng một FIFO queue

First Come First Served (2/5)

Process	Burst time (ms)
P_1	24
P_2	3
P_3	3

- Giả sử các process đến theo thứ tự P_1 , P_2 , P_3
- *Giản đồ Gantt* cho việc định thời là:



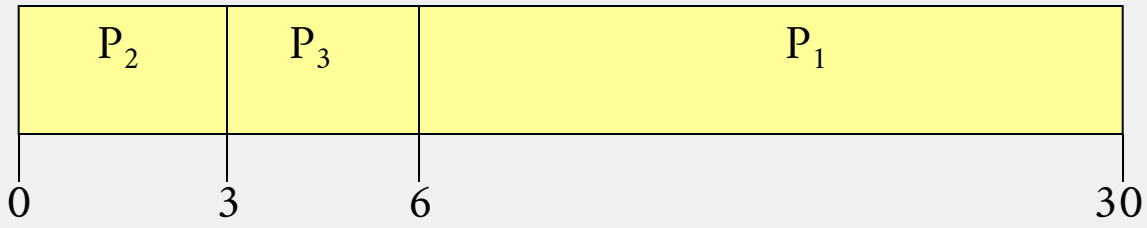
- Thời gian đợi cho $P_1 = 0$, $P_2 = 24$, $P_3 = 27$
  - Thời gian đợi trung bình: $(0 + 24 + 27) / 3 = 17$

First Come First Served (4/5)

- Giả sử các process đến theo thứ tự:

  P_2 , P_3 , P_1

- Giản đồ Gantt cho việc định thời là:



- Thời gian đợi của $P_1 = 6, P_2 = 0, P_3 = 3$
  - Thời gian đợi trung bình: $(6 + 0 + 3) / 3 = 3$
    - Tốt hơn rất nhiều so với trường hợp trước

First Come First Served (5/5)

Nhận xét

- FCFS "không công bằng" với process có CPU burst ngắn vì quá trình phải chờ trong thời gian dài (so với thời gian mà nó cần phục vụ) thì mới được sử dụng CPU. Điều này đồng nghĩa với việc FCFS 'ưu tiên' các process thuộc dạng CPU bound.

- Câu hỏi: Liệu có xảy ra trường hợp *trì hoãn vô hạn định* (starvation hay indefinite blocking) với giải thuật FCFS?

- FCFS thường được sử dụng trong các hệ thống bó (batch system)

Ví dụ thực tế

- Việc phục vụ khách trong nhà hàng
  - Thực khách sẽ đến và gọi món ăn cho mình
  - Mỗi món ăn cần thời gian chuẩn bị khác nhau
- Mục tiêu:
  - Giảm thời gian đợi trung bình của các thực khách
- Cách làm nào sẽ phù hợp?
  - Thông thường các nhà hàng sẽ phục vụ theo kiểu FCFS (!)

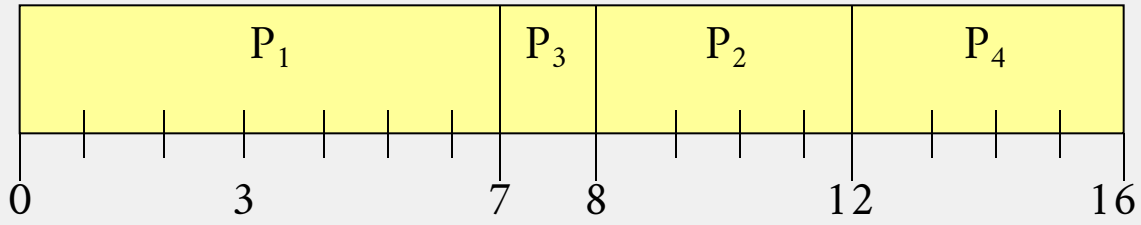
Shortest Job First (SJF) (1/3)

- Đối với mỗi process, cần biết độ dài của CPU burst
- Hàm lựa chọn: chọn process có độ dài CPU burst nhỏ nhất
- Chế độ quyết định: non-preemptive
- Chứng minh được: SJF là tối ưu trong việc giảm thời gian đợi trung bình
- Vấn đề: Cần phải ước lượng CPU burst tiếp theo của process
  - Giải pháp?

Shortest Job First (2/3)

Process	Thời điểm đến	Burst time (ms)
P_1	0,0	7
P_2	2,0	4
P_3	4,0	1
P_4	5,0	4

- Giản đồ Gantt khi định thời theo SJF



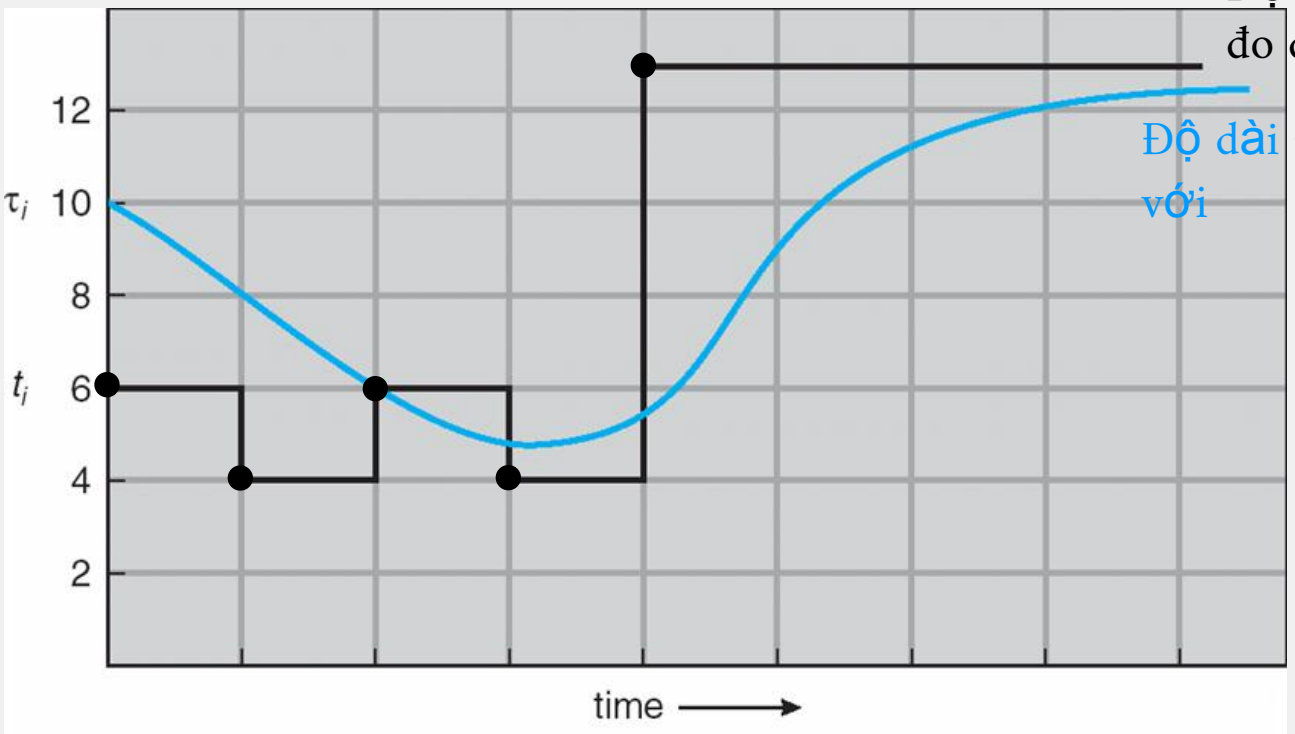
- Thời gian đợi trung bình = $(0 + 6 + 3 + 7)/4 = 4$

Dự đoán thời gian sử dụng CPU (1/2)

- Phương pháp: Lấy trung bình có trọng số các CPU burst đo được trong quá khứ
- Giả thiết: những CPU burst càng mới càng phản ánh gần hành vi của process trong tương lai
- Phương pháp *trung bình hàm mũ* (exponential averaging)
  - $\tau_{n+1} = at_n + (1 - a)\tau_n$, $0 \le a \le 1$
    - t là trị đo được, τ là trị dự đoán
  - Suy ra:

    $\tau_{n+1} = at_n + (1 - a)at_{n-1} + \ldots + (1 - a)^j at_{n-j} + \ldots + (1 - a)^{n+1} a\tau_0$
  - Ví dụ: Nếu chọn $a = \frac{1}{2}$ thì có nghĩa là trị đo được t_n và trị đã dự đoán τ_n được xem quan trọng như nhau

Dự đoán thời gian sử dụng CPU (2/2)



Độ dài CPU burst đo được

Độ dài CPU burst dự đoán, với § = ½ và ¶ 0 = 10

CPU burst (t_i)		6	4	6	4	13	13	13 ...
"guess" (τ_i)	10	8	6	6	5	9	11	12 ...

Shortest Job First

Nhận xét

- SJF sử dụng ưu tiên ngầm định: công việc ngắn nhất được ưu tiên trước
  - Những công việc thuộc loại I/O bound thường có CPU burst ngắn
- Process có thời gian thực thi dài có thể bị trì hoãn vô hạn định nếu các process có thời gian thực thi ngắn liên tục vào
- Không thích hợp cho môi trường tương tác
  - Các CPU bound process có "độ ưu tiên" thấp, nhưng có thể chiếm CPU lâu một khi nó được lựa chọn thực thi

Shortest Remaining Time First (SRTF) (1/4)

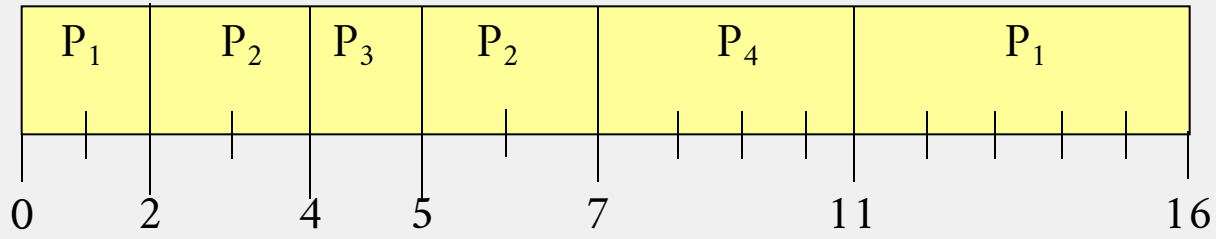
- Chế độ quyết định của SJF: non-preemptive
- Phiên bản preemptive của SJF:
  - Nếu một process mới đến mà có độ dài CPU burst nhỏ hơn thời gian cần CPU còn lại của process đang thực thi, thì preempt process đang thực thi
  - Cách làm này còn được gọi là *Shortest-Remaining-Time-First* (SRTF)

Shortest Remaining Time First (2/4)

(Ví dụ giống vd cho SJF)

Process	Thời điểm đến	Burst time (ms)
P_1	0,0	7
P_2	2,0	4
P_3	4,0	1
P_4	5,0	4

- Giản đồ Gantt khi định thời theo SRTF



- Thời gian đợi trung bình = (9 + 1 + 0 + 2) / 4 = 3
  - Tốt hơn giải thuật SJF (4)

Shortest Remaining Time First (4/4)

Nhận xét

- Tránh trường hợp process có thời gian thực thi dài độc chiếm CPU
- Cần phải quản lý thời gian thực thi còn lại của các process
- Có thời gian quay vòng tốt hơn SJF
- Process có thời gian thực thi ngắn có "độ ưu tiên" cao
- Có thể dẫn đến starvation

Priority Scheduling

- Mỗi process được gán một độ ưu tiên
- CPU sẽ được cấp cho process có độ ưu tiên cao nhất
- Chế độ quyết định:
  - Preemptive
  - Non-preemptive

Gán độ ưu tiên

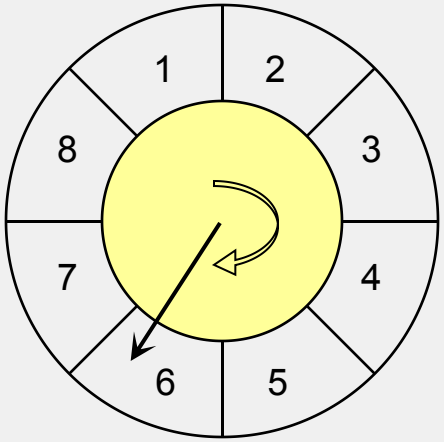
- SJF: độ ưu tiên là thời-gian-sử-dụng-CPU-dự-đoán
- Gán độ ưu tiên còn có thể dựa vào:
  - Yêu cầu về bộ nhớ
  - Số lượng file được mở
  - Tỉ lệ thời gian dùng cho I/O trên thời gian sử dụng CPU
  - Các yêu cầu bên ngoài như: số tiền người dùng trả khi thực thi công việc

Priority Scheduling

- Vấn đề: trì hoãn vô hạn định – process có độ ưu tiên thấp có thể không bao giờ được thực thi
- Giải pháp: *aging* – độ ưu tiên của process được tăng theo thời gian

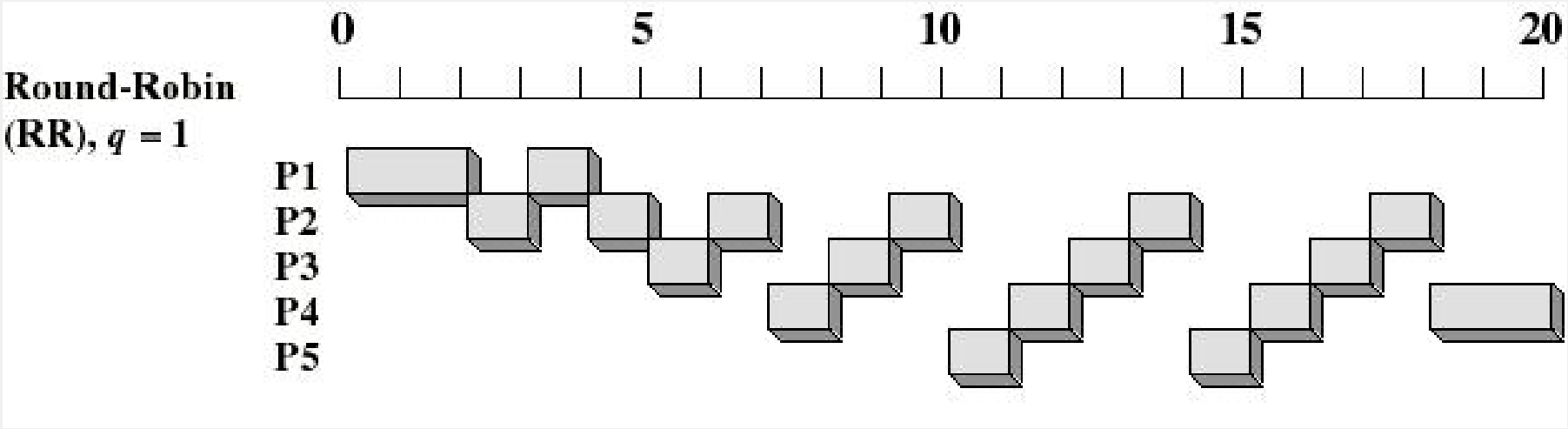
Round Robin (RR) (1/4)

- Hàm lựa chọn: giống FCFS



Round Robin (2/4)

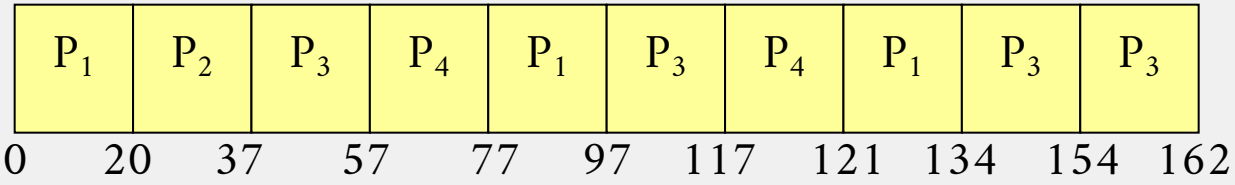
- Chế độ quyết định: preemptive
  - Khoảng thời gian tối đa cho phép (*quantum time*, thường 10 - 100 ms) được đảm bảo bằng việc sử dụng timer interrupt
  - Process đang chạy, khi hết thời gian (quantum time) sẽ được chuyển về cuối của hàng đợi ready



Round Robin (3/4)

Process	Burst time (ms)
P_1	53
P_2	17
P_3	68
P_4	24

- Quantum time = 20 ms
- Giản đồ Gantt:

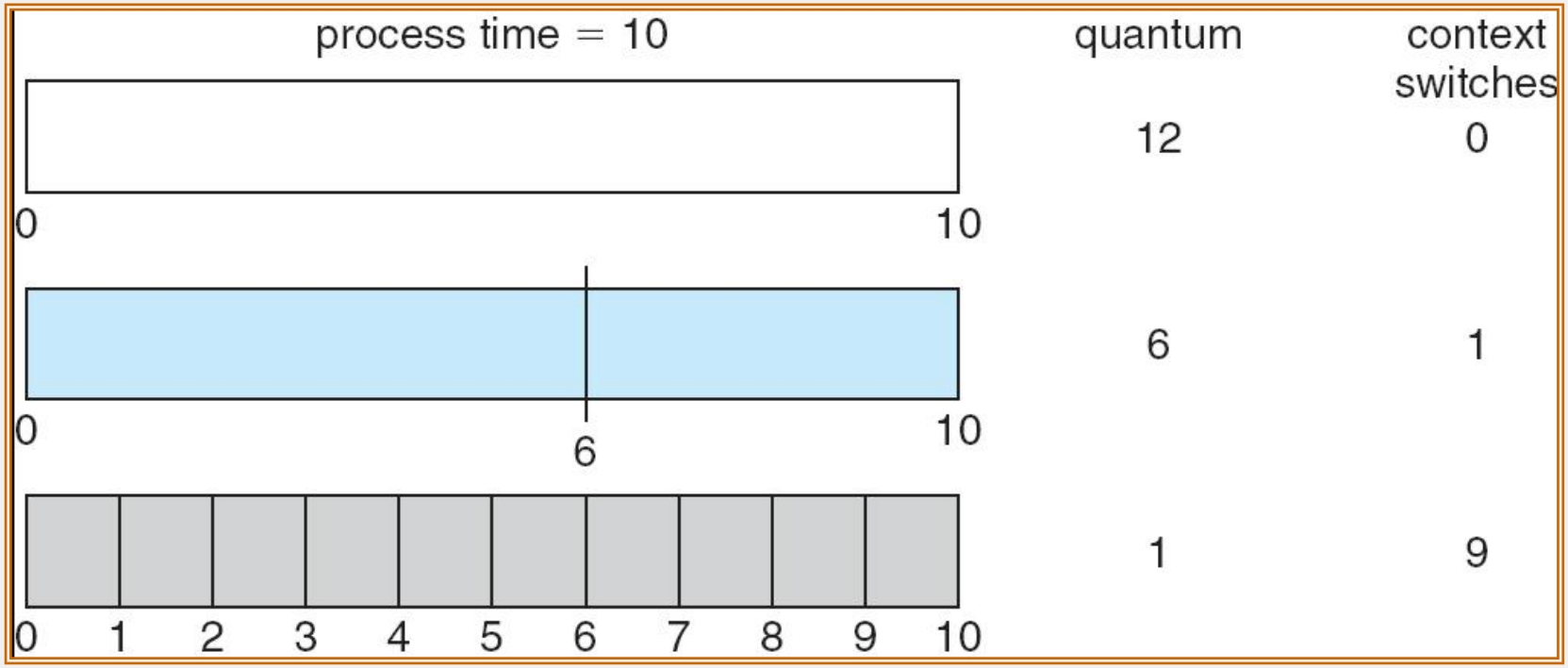


- Thường có thời gian quay vòng cao hơn SJF, nhưng lại có thời gian đáp ứng tốt hơn

Quantum time và chuyển ngữ cảnh

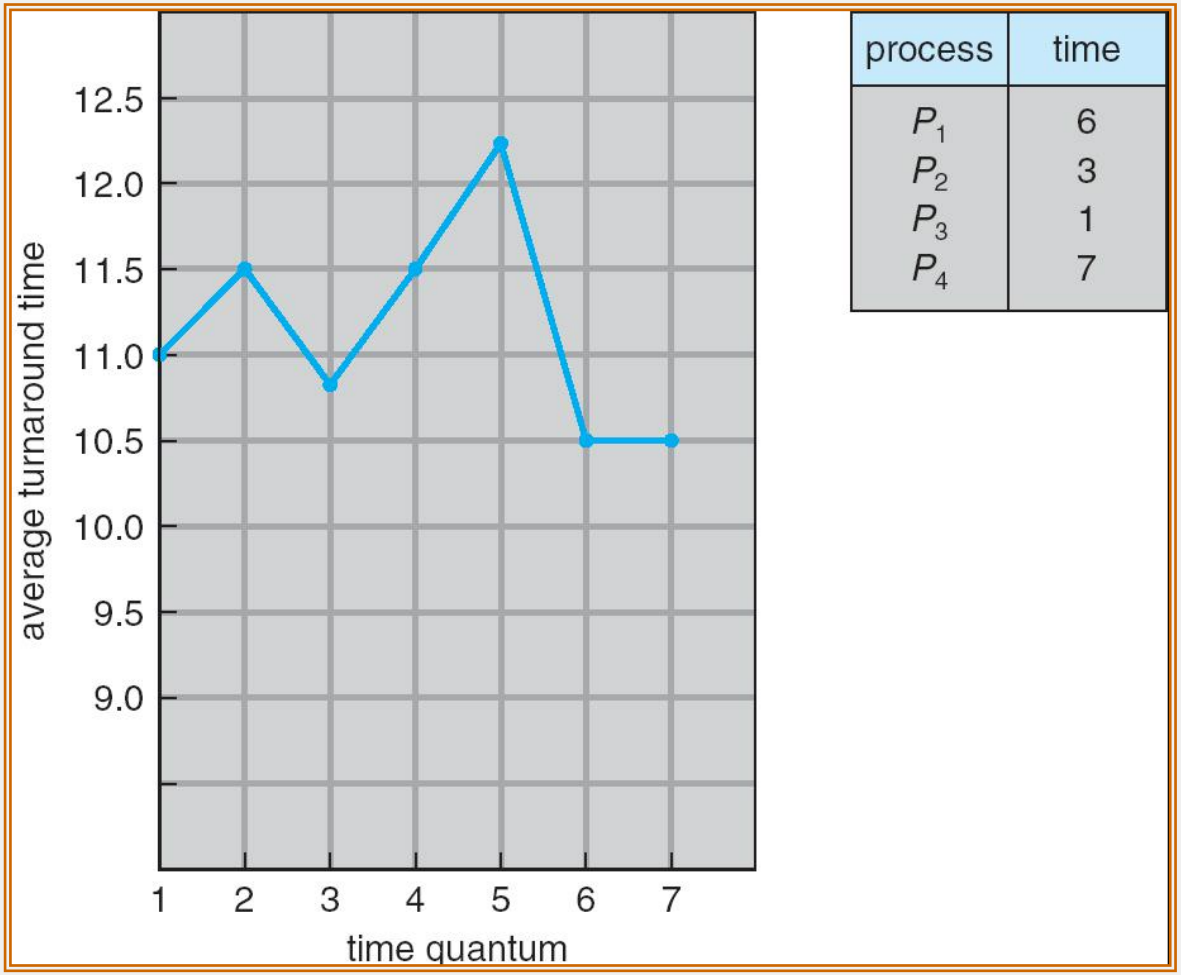
- Quantum time càng nhỏ thì càng có nhiều lần chuyển ngữ cảnh (context switch) trong khi thực thi

Số lần ngưng/tiếp tục quá trình



Quantum time và thời gian quay vòng

- Thời gian quay vòng trung bình không chắc sẽ được cải thiện khi quantum lớn



process	time
P_1	6
P_2	3
P_3	1
P_4	7

Quantum time cho Round Robin

Nhận xét

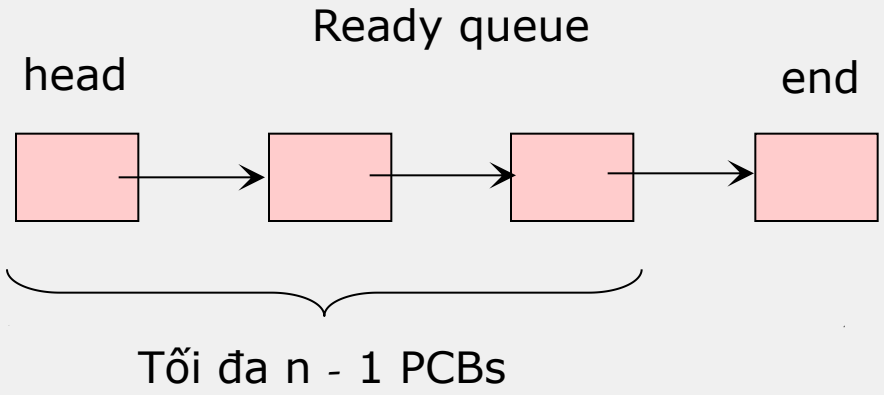
- Khi thực hiện process switch thì OS sẽ sử dụng CPU chứ không phải process của người dùng (*OS overhead*, phí tổn):
  - Dừng thực thi process, lưu tất cả thông tin, nạp thông tin của process sắp thực thi
- Performance tùy thuộc vào kích thước của quantum time (còn gọi là time slice), và hàm phụ thuộc này không đơn giản
  - Time slice ngắn thì đáp ứng nhanh
    - Vấn đề: có nhiều chuyển ngữ cảnh. Phí tổn sẽ cao
  - Time slice dài hơn thì throughput tốt hơn (do giảm OS overhead) nhưng thời gian đáp ứng lớn
  - Nếu time slice quá lớn, RR trở thành FCFS

Quantum time cho Round Robin

- Quantum time và thời gian cho process switch: ví dụ thời gian cho process switch = 5 ms
  - Nếu quantum time = 20 ms, thì phí tổn (OS overhead) chiếm 5 / (20 + 5) = 20%
  - Nếu quantum = 500 ms, thì phí tổn chỉ còn Ü 1%
    - Nhưng nếu có nhiều ứng dụng interactive thì sẽ thấy đáp ứng rất chậm
- Tùy thuộc vào tập công việc mà lựa chọn quantum time
- Quantum time nên lớn trong tương quan so sánh với thời gian cho process switch
  - Ví dụ với 4.3 BSD UNIX, quantum time là 1 s

Round Robin

- Nếu có n process trong hàng đợi ready, và quantum time là q, như vậy mỗi process sẽ được cấp phát thời gian CPU theo từng khối có kích thước lớn nhất là q
  - Sẽ không có process nào chờ lâu hơn $(n - 1)\,q$ đơn vị thời gian



- RR sử dụng một giả thiết ngầm là tất cả các process đều có tầm quan trọng ngang nhau
  - Không sử dụng RR nếu muốn các process khác nhau có độ ưu tiên khác nhau

Round Robin: nhược điểm

- Các process dạng CPU-bound vẫn còn được 'ưu tiên'
  - Ví dụ:
    - Một I/O-bound process sử dụng CPU trong thời gian ngắn hơn quantum time và block để đợi I/O. Và
    - Một CPU-bound process chạy hết time slice và liên tục quay trở về hàng đợi ready, thường ở trước các I/O bound process đang block.

Multilevel Queue Scheduling (1/3)

Được áp dụng trong trường hợp các quá trình có thể được phân thành nhóm, ví dụ: interactive và batch

- Hàng đợi ready sẽ được chia thành nhiều hàng đợi riêng rẽ
  - Ví dụ: *foreground* (cho công việc cần tương tác) và *background* (cho công việc dạng bó)
  - Không di chuyển quá trình sang hàng đợi ready khác

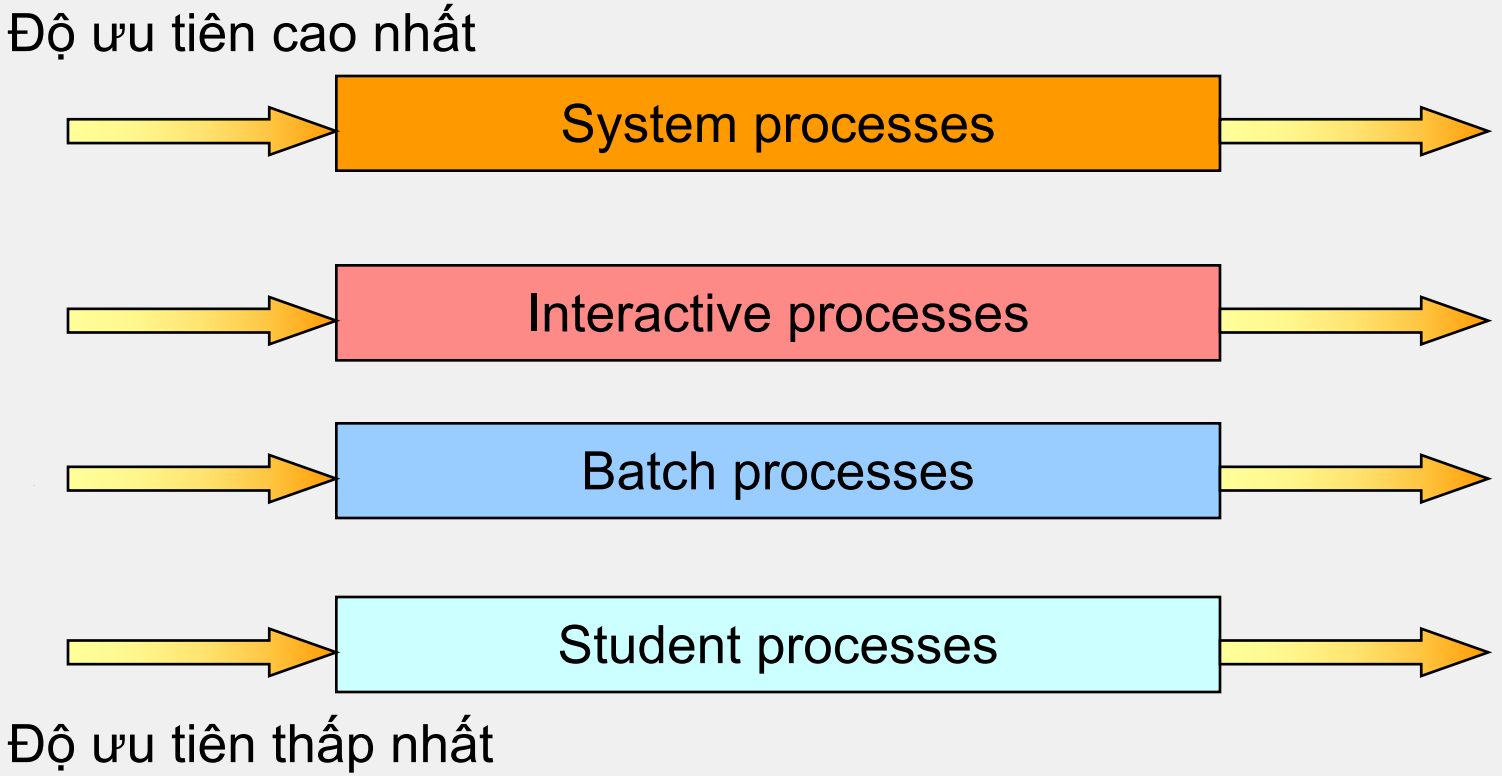
- Mỗi hàng đợi ready sẽ có giải thuật định thời riêng. Ví dụ:
  - foreground: dùng RR
  - background: dùng FCFS

Multilevel Queue Scheduling (2/3)

- Định thời trong Multilevel Queue Scheduling còn cần phải thực hiện giữa các hàng đợi với nhau
  - *Theo cách cố định* (fixed priority scheduling) – ví dụ: phục vụ tất cả các process của foreground rồi mới đến background
    - Process ở hàng đợi "cao" preempt process ở hàng đợi "thấp"
    - Có khả năng xảy ra trì hoãn vô hạn định (starvation) đối với process trong hàng đợi có độ ưu tiên thấp
  - *Chia thời gian* (time slice) – mỗi hàng đợi sẽ được lấy một khoảng sử dụng CPU nhất định để định thời cho các process của mình. Ví dụ:
    - 80% cho foreground (dùng RR)
    - 20% cho background (dùng FCFS)

Multilevel Queue Scheduling (3/3)

- Ví dụ phân nhóm các quá trình



Multilevel Feedback Queue (1/3)

- Trong hệ thống sử dụng Multilevel Feedback Queue, bộ định thời có thể di chuyển process giữa các ready queue tùy theo đặc tính của nó được quan sát.

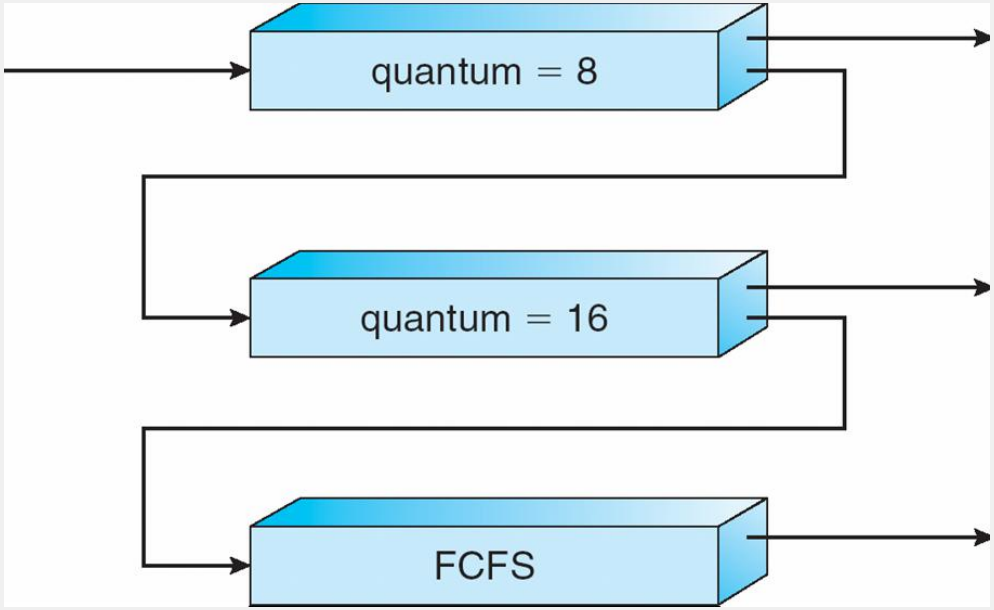
  Ví dụ:

  - Nếu một process sử dụng CPU quá lâu, nó sẽ bị di chuyển sang một hàng đợi có độ ưu tiên thấp hơn
  - Nếu một process chờ quá lâu trong một hàng đợi có độ ưu tiên thấp, nó sẽ được di chuyển lên hàng đợi có độ ưu tiên cao hơn (*aging*, giúp tránh starvation)

- Ví dụ: Windows NT/XP/Vista

Multilevel Feedback Queue (2/3)

- Ví dụ: Có 3 hàng đợi
  - Q_0 , dùng RR với q = 8 ms
  - Q_1 , dùng RR với q = 16 ms
  - Q_2 , dùng FCFS



- Giải thuật
  - Process mới sẽ vào hàng đợi Q_0 . Khi đến lượt, process sẽ được một quantum là 8 ms. Nếu không trả CPU trong vòng 8 ms, process sẽ được đưa xuống cuối hàng đợi Q_1
  - Tại Q_1 , process sẽ được cho một quantum là 16 ms. Nếu nó không trả CPU trước khi hết quantum thì sẽ bị chuyển xuống Q_2
  - Công việc ở hàng đợi "cao" preempt công việc ở hàng đợi "thấp"

Multilevel Feedback Queue (3/3)

- Multilevel Feedback Queue được xác định bởi các thông số
  - Có bao nhiêu hàng đợi?
  - Với mỗi queue sử dụng giải thuật định thời nào?
  - Khi nào thăng cấp một process?
  - Khi nào giáng cấp một process?

Chính sách và cơ chế (1/2)

Một nguyên lý thiết kế hệ điều hành: Tách biệt chính sách (policy) và cơ chế (mechanism)

- *Chính sách*
  - Lược đồ ra quyết định cần phải làm gì (ví dụ để tối ưu một tiêu chí nào đó)
- *Cơ chế*
  - Công cụ để hiện thực các chính sách

Chính sách và cơ chế (2/2)

- Định thời quá trình
  - Chính sách
    - Khi nào thì định thời quá trình?
    - Quá trình nào được chạy?
  - Cơ chế
    - Làm thế nào để chuyển ngữ cảnh? Dispatcher
- Ví dụ:
  - Chính sách: FCFS
  - Cơ chế: dispatcher
- Ưu điểm: ví dụ
  - Thay đổi loại tải công việc (workload) thì chỉ cần thay đổi chính sách
  - Thay đổi HW platform thì chỉ cần thay đổi dispatcher

Tổng kết

- Sự thực thi của một process
- Bộ định thời chọn một process từ hàng đợi ready
  - Dispatcher thực hiện switching
- Các tiêu chí định thời (thường xung đột nhau)
  - Độ lợi CPU, thời gian chờ, thời gian đáp ứng, thông năng,…
- Các giải thuật định thời
  - FCFS, SJF, Priority, RR, Multilevel Feedback Queue

Bài tập 1

- Hệ thống đa chương gồm có 2 quá trình A & B và hệ thống thực hiện các tác vụ sau:

- Nạp A vào bộ nhớ mất 10s, sau đó quá trình A thực hiện thao tác tính toán mất 5s, và kế đến mất 1 phút để thực hiện in kết quả.
- Nạp B vào bộ nhớ mất 15s, B thực hiện tính toán mất 20s, và mất 62 s để in.

Biết quá trình A được nạp vào bộ nhớ đầu tiên.

Hãy tính tỷ lệ % thời gian sử dụng đĩa, CPU và máy in.

Bài tập 2

Process	Burst Time
P_1	10
P_2	29
P_3	3
P_4	7
P_5	12

- Tất cả đều đến ở thời điểm 0
- Xét các giải thuật FCFS, SJF, và RR với quantum time = 10
- Giải thuật nào cho
  - thời gian đợi trung bình nhỏ nhất?
  - thông năng cao nhất?
  - thời gian quay vòng trung bình của process nhỏ nhất?

Bài tập 3

- Có 3 quá trình A, B,C thực hiện các tác vụ theo thứ tự như sau: thực hiện tính toán, yêu cầu I/O và lặp lại.
- Thời gian mỗi lần thực thi của A,B,C lần lượt à 10ms, 10ms, và 45ms.
- Thời gian mỗi lần thực hiện tác vụ I/O của mỗi quá trình A,B,C là 20ms.
- Quan sát sự thực thi của quá trình trong 150ms đầu tiên và hãy tính độ lợi sử dụng CPU với hai giải thuật Round Robin (quantum = 20ms) và FCFS. Biết rằng thứ tự xuất hiện tại hàng đợi ready lần lượt là A,B,C.

Bài tập 4

Process	Burst Time	Arrival time	Priority
P_1	10	0	3
P_2	29	2	2
P_3	3	5	1
P_4	7	3	3
P_5	12	6	0 (cao nhat)

Xét các giải thuật FCFS, SJF, SRTF, preemptive/non-preemptive Priority, và RR với quantum time = 10

Hãy tính thời gian đợi trung bình của các quá trình?

Tính thời gian đợi trung bình khi thực hiện giải thuật RR với q = 5.

Bài tập 5

Process	Burst Time	Arrival time
P_1	11	0
P_2	12	3
P_3	13	9

- Sử dụng MLF 3 hàng đợi với thông tin cho mỗi hàng đợi như sau:
  - Q0: RR (4ms)
  - Q1: RR (6ms)
  - Q2: FCFS

  Hãy tính thời gian đợi của các quá trình.