



## Chương 3: Lập lịch - Scheduling

---

*Tìm hiểu về: khái niệm lập lịch, các thuật toán lập lịch sử dụng trong hệ điều hành*



# Nội dung

---

- Khái niệm
- Tiêu chuẩn lập lịch
- Giải thuật lập lịch
- Lập lịch multiprocessor
- Lập lịch thời gian thực
- Lựa chọn giải thuật



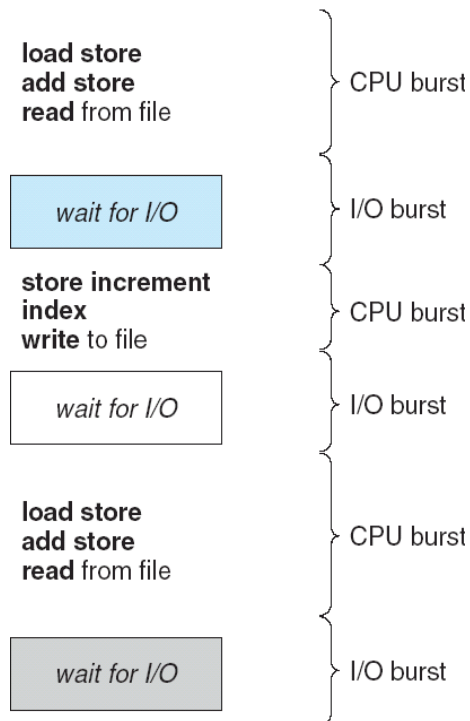
# 1. Khái niệm(1)

---

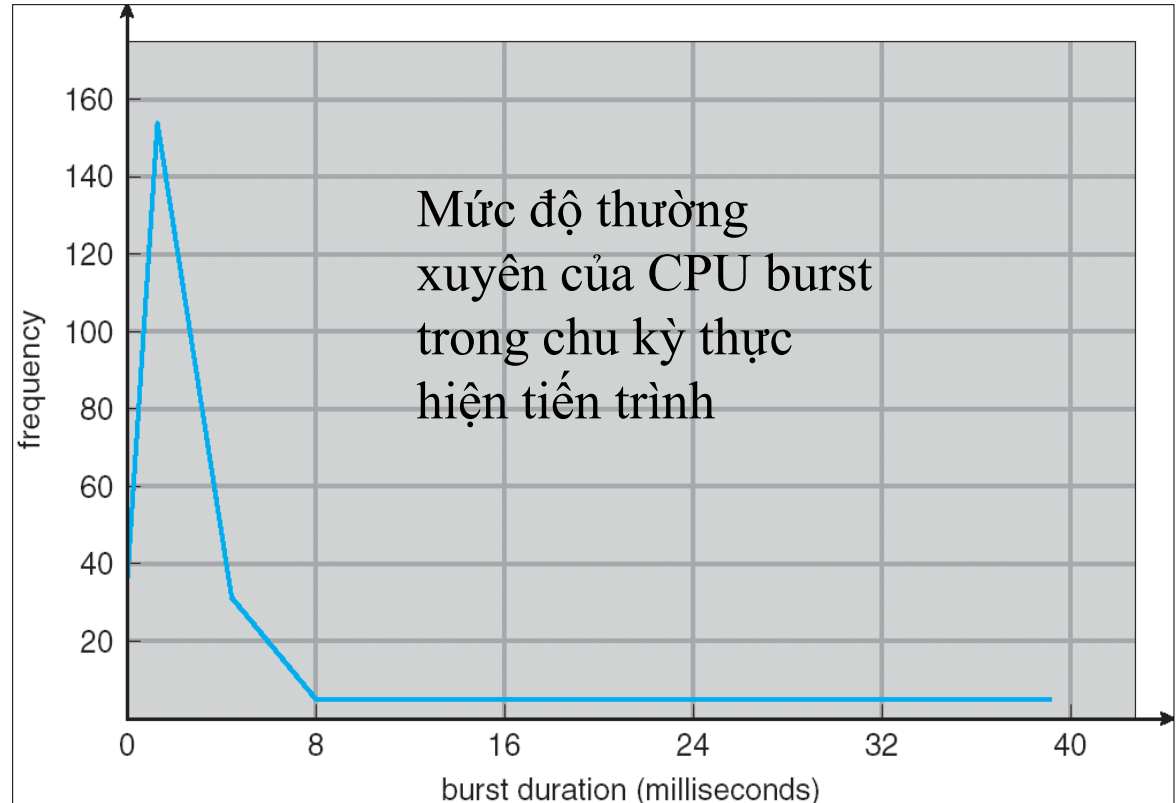
- Multi-programming (chế độ đa chương trình) giúp tăng hiệu quả sử dụng CPU
- Chu kỳ sử dụng CPU–I/O (CPU–I/O Burst Cycle):
  - Sự thực hiện tiến trình luôn chứa một chu kỳ thực hiện của CPU và chu kỳ chờ I/O.
  - CPU burst và I/O burst luân phiên nhau
- Sự phân phối sử dụng CPU giúp lựa chọn giải thuật lập lịch CPU

# 1. Khái niệm(2): CPU-I/O burst

## Alternating Sequence of CPU And I/O Bursts

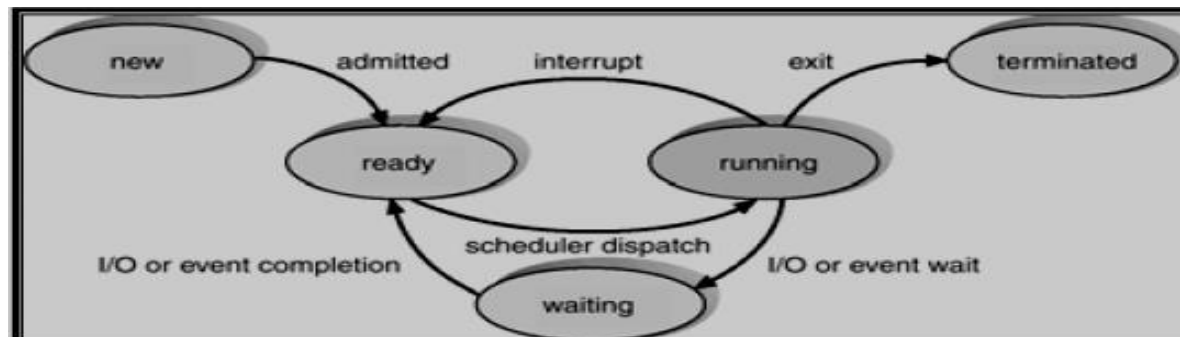


## Histogram of CPU-burst Times



# 1.1. Trình lập lịch CPU - CPU Scheduler

- Mỗi khi CPU rỗi, HĐH cần chọn trong số các tiến trình ở trạng thái sẵn sàng( ready) thực hiện trong bộ nhớ và phân phối CPU cho một trong số đó.
- Tiến trình được thực hiện bởi trình lập lịch ngắn kỳ (short-term scheduler, CPU scheduler)
- Các quyết định lập lịch CPU có thể xảy ra khi một tiến trình:
  - 1. Chuyển từ trạng thái chạy sang trạng thái chờ (vd: I/O request)
  - 2. Chuyển từ trạng thái chạy sang trạng thái sẵn sàng (vd: khi một ngắt xuất hiện)
  - 3. Chuyển từ trạng thái đợi sang trạng thái sẵn sàng (vd: I/O hoàn thành)
  - 4. Kết thúc





## 1.2. Preemptive/nonpreemptive Scheduling

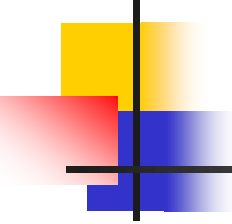
- Lập lịch CPU khi (1) và (4) là *không được ưu tiên trước (nonpreemptive)*-**độc quyền**:
  - Không có sự lựa chọn: phải chọn 1 tiến trình mới để thực hiện.
  - Tiến trình được phân phối CPU, nó sẽ sử dụng CPU cho đến khi nó giải phóng CPU bằng cách kết thúc hoặc chuyển sang trạng thái chờ.
  - Các tiến trình không sẵn sàng nhường điều khiển của CPU.
- Lập lịch khi (2) và (3) là *được ưu tiên trước (preemptive)*-**không độc quyền**
  - Khi (2): tiến trình giải phóng CPU -> Cần phải chọn tiến trình kế tiếp.
  - Khi (3): tiến trình có thể đẩy tiến trình khác để dành quyền điều khiển CPU.



## 1.3. Trình điều vận- Dispatcher

---

- Môđun trình điều vận *trao quyền điều khiển CPU cho tiến trình* được lựa chọn bởi trình lập lịch CPU. Các bước:
  - 1. Chuyển ngữ cảnh
  - 2. Chuyển sang user mode
  - 3. Nhảy tới vị trí thích hợp trong chương trình của người sử dụng để khởi động lại chương trình đó
- *Trễ điều vận (Dispatch latency)* – thời gian cần thiết để trình điều vận dừng một tiến trình và khởi động một tiến trình khác chạy.



## 2. Tiêu chuẩn lập lịch

---

- **CPU utilization(tận dụng)** – giữ cho CPU càng bận càng tốt (0-100%)
- **Throughput(thông lượng tối đa)** – số tiến trình được hoàn thành trong một đơn vị thời gian
- **Turnaround time** – tổng lượng thời gian để thực hiện một tiến trình: t/g chờ được đưa vào bộ nhớ + t/g chờ trong ready queue + t/g thực hiện bởi CPU + t/g thực hiện vào-ra
- **Waiting time** – thời gian mà một tiến trình chờ đợi ở trong *ready queue*
- **Response time** – lượng thời gian tính từ khi có một yêu cầu được gửi đến khi có sự trả lời đầu tiên được phát ra, không phải là thời gian đưa ra kết quả của sự trả lời đó.  
→ là tiêu chuẩn tốt.





## 3. Các giải thuật lập lịch

---

- Giải thuật First-Come, First-Served
- Giải thuật Shortest-Job-First
- Giải thuật Lập lịch có ưu tiên - Priority Scheduling
- Giải thuật Round-Robin (RR)
- Giải thuật Lập lịch hàng đợi đa mức Multilevel Queue
- Giải thuật Hàng đợi phản hồi đa mức - Multilevel Feedback Queue

# 3.1. Giải thuật First-Come, First-Served (FCFS)(1)

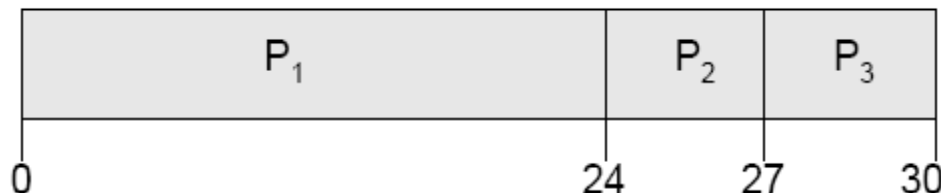
- Tiến trình nào yêu cầu CPU trước sẽ được phân phối CPU trước → Giải thuật FCFS là không được ưu tiên
- Là giải thuật đơn giản nhất
- Process Burst Time (thời gian xử lý - *thời gian sử dụng CPU*, ms)

$P_1$       24

$P_2$       3

$P_3$       3

- Giả định rằng các tiến trình đến theo thứ tự:  $P_1, P_2, P_3$  thì biểu đồ Gantt (Gantt Chart) của lịch biểu như sau:



- Thời gian chờ đợi của các tiến trình:  $P_1 = 0$ ;  $P_2 = 24$ ;  $P_3 = 27$
- Thời gian chờ đợi trung bình:  $(0 + 24 + 27)/3 = 17$

# 3.1. Giải thuật First-Come, First-Served (FCFS)(2)

Giả sử các tiến trình đến theo thứ tự  $P_2$ ,  $P_3$ ,  $P_1$

- Biểu đồ Gantt của lịch biểu như sau:



- Thời gian chờ đợi của các tiến trình:  $P_1 = 6$ ;  $P_2 = 0$ ;  $P_3 = 3$
- Thời gian chờ đợi trung bình:  $(6 + 0 + 3)/3 = 3$
- Tốt hơn nhiều so với trường hợp trước
- *Convoy effect* (hiệu ứng hộ tống): tiến trình ngắn đứng sau tiến trình dài  $\rightarrow$  tăng thời gian đợi của các tiến trình

## 3.2. Giải thuật Shortest-Job-First (SJF)(1)

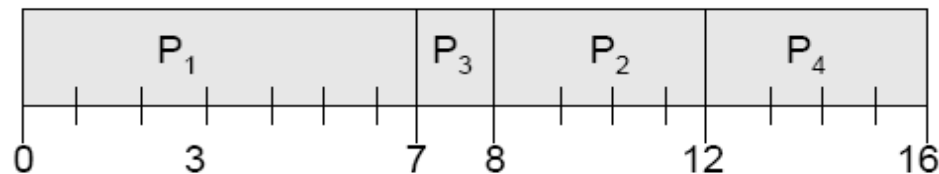
- Gắn với mỗi tiến trình là thời gian sử dụng CPU *tiếp sau* của nó.
- Thời gian này được sử dụng để lập lịch các tiến trình với thời gian đợi ngắn nhất.
- Hai phương pháp:
  - Không ưu tiên trước (non-preemptive) – một tiến trình nếu sử dụng CPU thì **không nhường cho tiến trình khác** cho đến khi nó kết thúc.
  - Có ưu tiên trước – nếu một tiến trình đến có thời gian sử dụng CPU *ngắn hơn* **thời gian còn lại của tiến trình đang thực hiện** thì ưu tiên tiến trình mới đến trước. Phương pháp này còn được gọi là **Shortest-Remaining-Time-First (SRTF)**
- **SJF** là tối ưu – cho thời gian chờ đợi trung bình của các tiến trình là nhỏ nhất

## 3.2. Giải thuật Shortest-Job-First (SJF)(2)

Ví dụ SJF không ưu tiên trước (đọc quyền)

<u>Process</u>	<u>Arrival Time</u>	<u>Burst Time</u>
$P_1$	0.0	7
$P_2$	2.0	4
$P_3$	4.0	1
$P_4$	5.0	4

- SJF (non-preemptive)



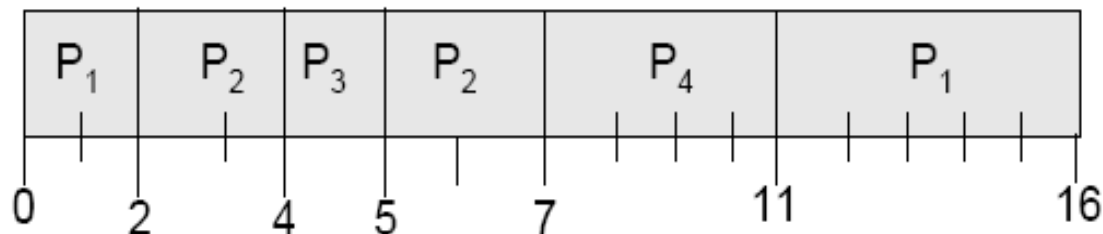
- Thời gian chờ đợi của các tiến trình:  $P1 = 0$ ;  $P2 = 6$ ;  $P3 = 3$ ,  $P4 = 7$
- Thời gian chờ đợi trung bình =  $(0 + 6 + 3 + 7)/4 = 4$

## 3.2. Giải thuật Shortest-Job-First (SJF)(3)

- Ví dụ Preemptive SJF

<u>Process</u>	<u>Arrival Time</u>	<u>Burst Time</u>
$P_1$	0.0	7
$P_2$	2.0	4
$P_3$	4.0	1
$P_4$	5.0	4

- SJF (preemptive)



- Thời gian chờ đợi trung bình =  $(9 + 1 + 0 + 2)/4 = 3$

## 3.3. Xác định thời gian sử dụng CPU kế tiếp(1)

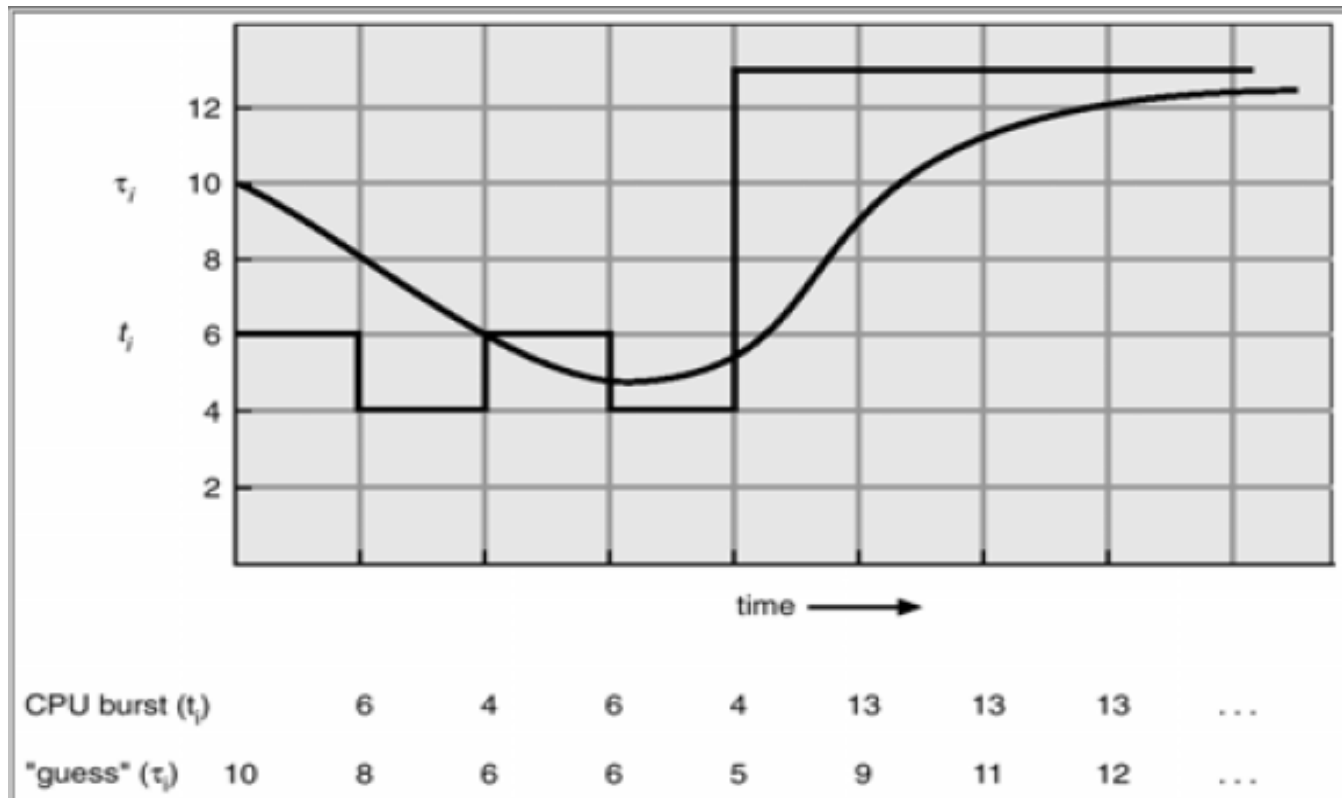
- Không thể biết chính xác thời gian sử dụng CPU tiếp sau của tiến trình nhưng có thể đoán giá trị xấp xỉ của nó dựa vào thời gian sử dụng CPU trước đó và sử dụng công thức đệ quy:

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

- $\tau_{n+1}$  = giá trị dự đoán cho thời gian sử dụng CPU tiếp sau
- $t_n$  = thời gian thực tế của sự sử dụng CPU thứ n
- $\alpha$ ,  $0 \leq \alpha \leq 1$
- $\tau_0$  là một hằng số
- $\alpha = 0$ :  $\tau_{n+1} = \tau_n = \tau_0$ .
  - Thời gian thực tế sử dụng CPU gần đây không có tác dụng gì cả.
- $\alpha = 1$ :  $\tau_{n+1} = t_n$ .
  - Chỉ tính đến thời gian sử dụng CPU thực tế ngay trước đó.

## 3.3. Xác định thời gian sử dụng CPU kế tiếp(2)

- Minh họa khi  $\alpha = 1/2$  và  $\tau_0 = 10$





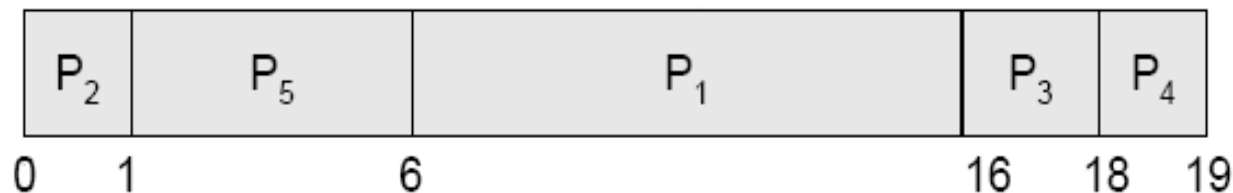
## 3.4. Lập lịch có ưu tiên - Priority Scheduling(1)

- Mỗi tiến trình được gán một số ưu tiên (số nguyên). VD: 0-127
- CPU được phân phối cho tiến trình có mức ưu tiên cao nhất (có số ưu tiên nhỏ nhất)
  - Preemptive
  - nonpreemptive
- SJF là trường hợp riêng của lập lịch có ưu tiên: mức ưu tiên chính là thời gian sử dụng CPU tiếp sau dự đoán được.
- Vấn đề: những tiến trình có mức ưu tiên thấp có thể không bao giờ được thực hiện (starvation).
- Giải pháp  $\equiv$  Aging: kỹ thuật tăng mức ưu tiên của các tiến trình chờ đợi lâu trong hệ thống.
  - VD: Sau 1-15 phút giảm số ưu tiên một lần.

## 3.4. Lập lịch có ưu tiên - Priority Scheduling(2): Ví dụ

<u>Process</u>	<u>Burst Time</u>	<u>Priority</u>
$P_1$	10	3
$P_2$	1	1
$P_3$	2	4
$P_4$	1	5
$P_5$	5	2

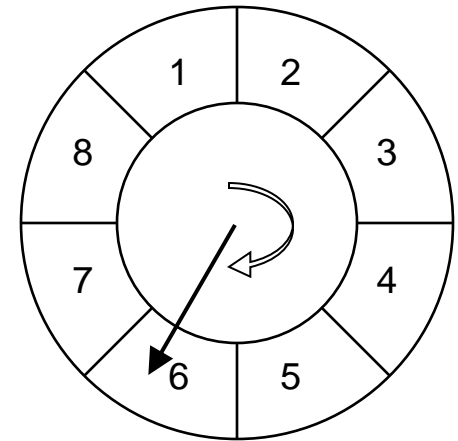
- Preemptive:



- T/gian chờ đợi trung bình =  $(0 + 1 + 6 + 16 + 18)/5 = 8.2$

## 3.5. Giải thuật Round-Robin (RR)(1)

- Mỗi tiến trình sử dụng một lượng nhỏ thời gian của CPU (*time quantum* – *thời gian định lượng*,  $q$ ), thường là 10-100 ms.
- Sau thời gian thực hiện  $q$ , tiến trình đưa vào cuối của *ready queue*.
- *Ready queue* được tổ chức dạng FIFO (FCFS)
- Nếu tiến trình có thời gian sử dụng CPU còn lại  $< q$  thì tiến trình sẽ giải phóng CPU khi kết thúc và không có mặt trong *ready queue*. Trình lập lịch sẽ chọn tiến trình kế tiếp trong *ready queue*.
- Nếu tiến trình có thời gian sử dụng CPU còn lại  $> q$  thì bộ định thời (timer) sẽ đếm lùi và gây ngắt HĐH khi nó = 0. Việc chuyển ngữ cảnh được thực hiện để chuyển điều khiển CPU cho tiến trình ở đầu hàng đợi, và tiến trình hiện tại được đưa xuống cuối *ready queue*.



**Có  $n$  tiến trình thì  $t$  đợi tối đa là  $(n-1)*q$**

Ex: 8 tiến trình,  
 $q=10$  thì  $t_{\text{Max}}=70$   
(Hình trên)

## 3.5. Giải thuật Round-Robin (RR)(2): $q=4$

<u>Process</u>	<u>Burst Time</u>
$P_1$	24
$P_2$	3
$P_3$	3

- Biểu đồ Gantt:



- T/gian chờ đợi trung bình =  $(4 + 7 + 6)/3 = 5.67$

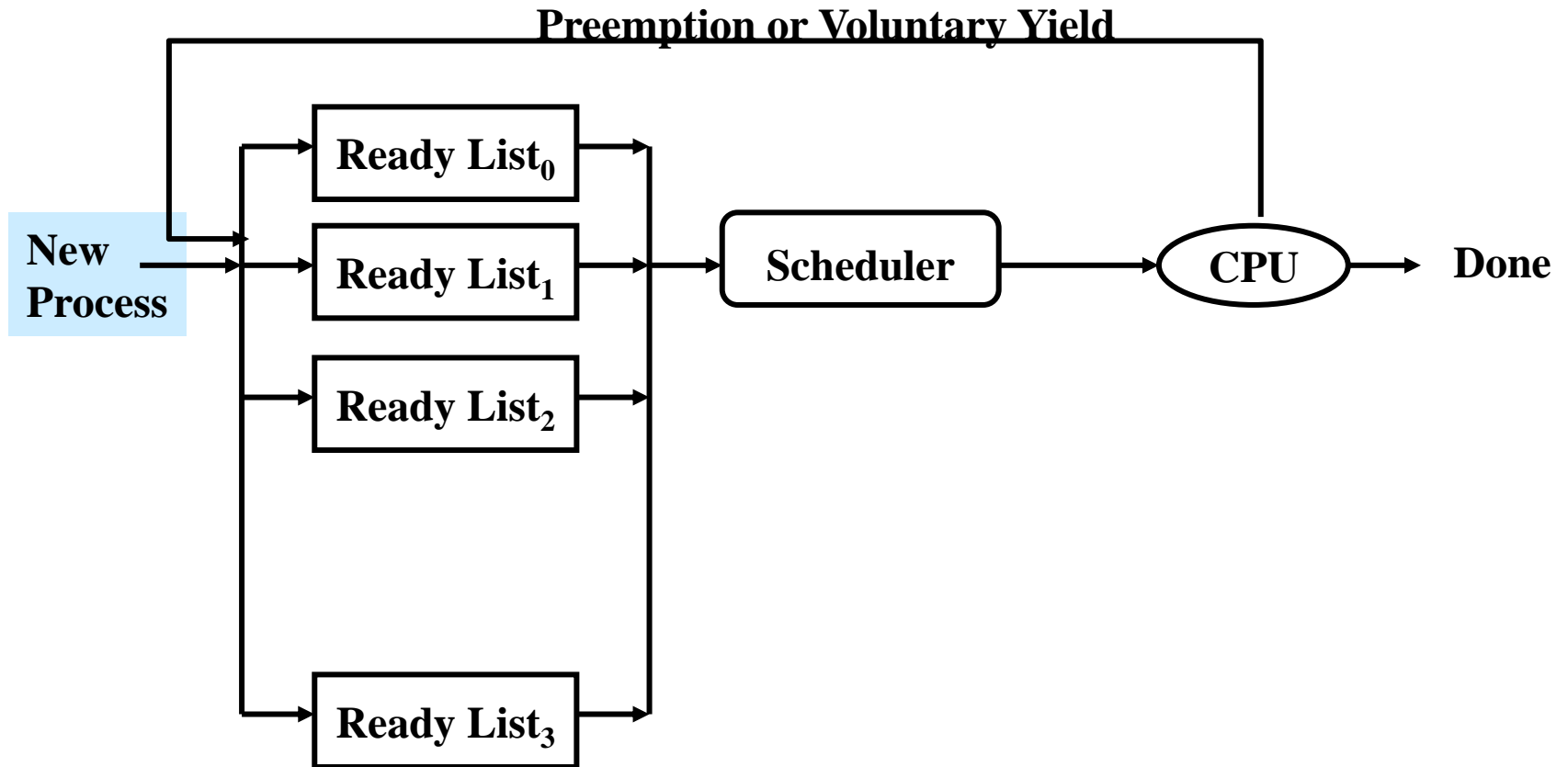
## 3.6. Lập lịch hàng đợi đa mức

### Multilevel Queue(1)

- **Ý tưởng:** chia ready queue thành nhiều queue, các tiến trình trên cùng queue có cùng độ ưu tiên
- **Phổ biến:**
  - foreground (chứa các tiến trình ảnh hưởng lẫn nhau *interactive process*)
  - background (chứa các tiến trình bó *batch process*)
- **Mỗi hàng đợi có giải thuật lập lịch riêng:**
  - foreground – RR
  - background – FCFS
- **Phải có sự lập lịch giữa các queue:**
  - Lập lịch với mức ưu tiên cố định; vd: phục vụ tất cả tiến trình từ foreground, tiếp theo từ background (có thể xảy ra starvation).
  - Phân chia thời gian: mỗi queue nhận được một lượng thời gian CPU nào đó mà nó có thể lập lịch các tiến trình của nó.
    - vd: 80% cho foreground và 20% cho background queue

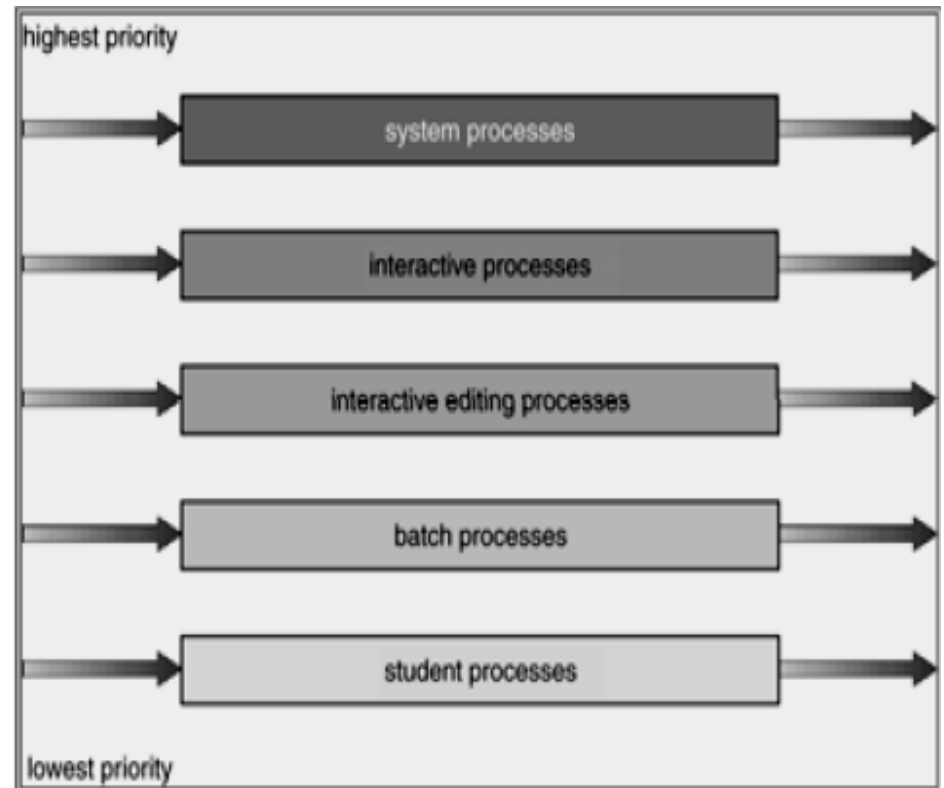
# 3.6. Lập lịch hàng đợi đa mức

## Multilevel Queue(1)



## 3.6. Lập lịch hàng đợi đa mức Multilevel Queue(2)

- Tiến trình trong queue có mức ưu tiên thấp hơn chỉ có thể chạy khi các *queue có mức ưu tiên cao hơn rỗng*.
- Tiến trình có mức ưu tiên cao hơn **khi vào** ready queue **không ảnh hưởng** đến tiến trình đang chạy có mức ưu tiên thấp hơn.
- Tiến trình mức ưu tiên thấp có thể *đói* CPU



## 3.7. Hàng đợi phản hồi đa mức

### Multilevel Feedback Queue(1)

---

- Hạn chế của multilevel queue:
  - Không linh động(một tiến trình không thể di chuyển giữa các hàng đợi)
  - Dễ xảy ra trường hợp *đói* CPU
- Multilevel Feedback Queue:
  - Tiến trình có thể di chuyển giữa các queue
  - Tiến trình sử dụng nhiều CPU burst có thể di chuyển từ hàng đợi có mức ưu tiên cao xuống hàng đợi có mức ưu tiên thấp hơn



# 3.7. Hàng đợi phản hồi đa mức

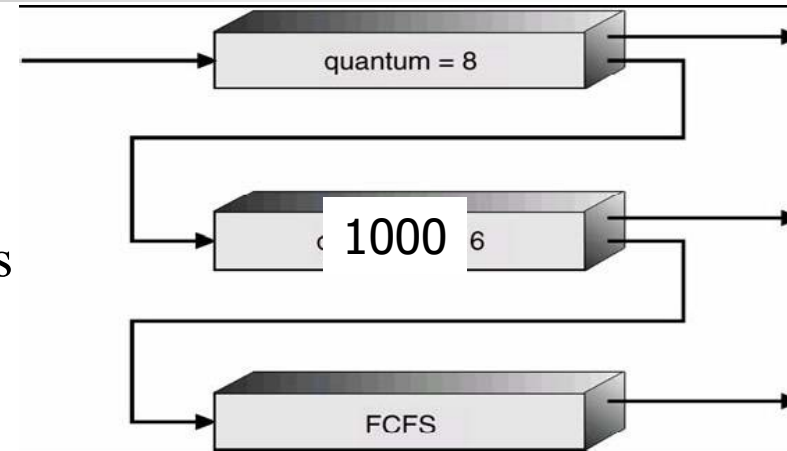
## Multilevel Feedback Queue(2)

- Ví dụ: có 3 queue:

- $Q0$  – RR, thời gian định lượng 8 ms
- $Q1$  – RR, thời gian định lượng 16 ms
- $Q2$  – FCFS

- Lập lịch:

- Một tiến trình vào queue  $Q0$  và được phục vụ RR. Khi nó giành được CPU, tiến trình nhận được 8 ms. Nếu nó không hoàn thành trong 8 ms, tiến trình được chuyển tới queue  $Q1$ .
- Tại  $Q1$  tiến trình tiếp tục được phục vụ RR với 16 ms nữa. Nếu nó vẫn chưa hoàn thành thì nó được ưu tiên và được chuyển đến queue  $Q2$ .





## 3.8. Điều phối Lottery(xổ số)

---

- **Nguyên tắc :**

- Ý tưởng chính của giải thuật là phát hành một số vé số và phân phối cho các tiến trình trong hệ thống.
- Khi đến thời điểm ra quyết định điều phối, sẽ **tiến hành chọn 1 vé "trúng giải", tiến trình nào sở hữu vé này sẽ được nhận CPU(chọn ngẫu nhiên)**



## 4. Lập lịch multiprocessor

---

- Lập lịch CPU khi có nhiều processor phức tạp hơn nhiều
- Các loại processor trong multiprocessor
  - *Đồng nhất (Homogeneous)*: tất cả có cùng kiến trúc.
  - *Không đồng nhất (Heterogeneous)*: một số tiến trình có thể không tương thích với kiến trúc của các CPU.
  - *Cân bằng tải (Load balancing/sharing)*: một ready queue cho tất cả các processor, CPU nhận rồi được gán cho tiến trình ở đầu queue (tránh trường hợp 1 CPU quá tải-**ready queue** gán với nó liên tục đầy trong khi các **ready queue** gán với các CPU khác có thể luôn rỗng).
  - *Đa xử lý không đối xứng - Asymmetric multiprocessing*: chỉ một processor (master processor) truy nhập các cấu trúc dữ liệu hệ thống, làm giảm sự chia sẻ dữ liệu; có thể gây hiệu ứng *thắt cổ chai* khi processor master (CPU master) phải thực hiện quá nhiều việc.



## 5. Lập lịch thời gian thực

---

- *Hard real-time* systems – yêu cầu hoàn thành một tác vụ gấp (critical task) trong thời gian được đảm bảo.
  - **Resource reservation:** khi tiến trình được gửi đến cùng với lệnh cho biết thời gian cần thiết của nó, trình lập lịch có thể chấp nhận và đảm bảo nó sẽ kết thúc đúng hạn, hoặc từ chối tiến trình.
- *Soft real-time* computing – yêu cầu các tiến trình gấp nhận mức ưu tiên lớn hơn các tiến trình **non real-time**.
  - Có thể phân phối tài nguyên không hợp lý, thời gian trễ lâu, starvation. → phải cẩn thận trong thiết kế trình lập lịch và các khía cạnh liên quan của HĐH:
    - Lập lịch có ưu tiên, các tiến trình thời gian thực có **mức ưu tiên cao nhất** và phải không giảm theo thời gian
    - Trễ điều vận (dispatch latency) phải nhỏ.



## 6. Lựa chọn giải thuật

---

- Chọn giải thuật lập lịch CPU nào cho hệ thống cụ thể?
- Trước tiên, xác định sử dụng tiêu chuẩn nào? Ví dụ:
  - Tối đa CPU utilization với ràng buộc response time lớn nhất là 1s
  - Tối đa throughput (thông lượng tối đa) để turnaround time (t hoàn thành) là tỷ lệ tuyến tính với thời gian thực hiện
- 1. Phân tích hiệu năng của từng giải thuật đối với các tiến trình
- 2. Sử dụng chuẩn hàng đợi: công thức Little:  $n = \lambda \times W$ 
  - $n$ : độ dài queue trung bình
  - $W$ : thời gian chờ đợi trung bình trong queue
  - $\lambda$ : tốc độ đến queue của tiến trình (số tiến trình/giây)
- 3. Mô phỏng: lập trình mô hình hệ thống để đánh giá
- 4. Thực hiện: đặt giải thuật cụ thể trong hệ thống thực để đánh giá



# Q & A

---

■ ...