



# Chương 6 : Tắc nghẽn(Deadlock)

---

- ❑ Mô hình hệ thống
- ❑ Định nghĩa
- ❑ Điều kiện cần của deadlock
- ❑ Resource Allocation Graph (RAG)
- ❑ Phương pháp giải quyết deadlock
- ❑ Deadlock prevention
- ❑ Deadlock avoidance
- ❑ Deadlock detection
- ❑ Deadlock recovery
- ❑ Phương pháp kết hợp để giải quyết Deadlock



# Vấn đề deadlock trong hệ thống

- ❑ *Tình huống*: một tập các process bị blocked, mỗi process giữ tài nguyên và đang chờ tài nguyên mà process khác trong tập đang giữ.
- ❑ Ví dụ 1
  - Giả sử hệ thống có 2 file trên đĩa.
  - P1 và P2 mỗi process đang mở một file và yêu cầu mở file kia.
- ❑ Ví dụ 2
  - Semaphore A và B, **khởi tạo bằng 1**

P0	P1
wait(A);	wait(B);
wait(B);	wait(A);



# Mô hình hóa hệ thống

- ❑ Hệ thống gồm các loại *tài nguyên*, kí hiệu  $R_1, R_2, \dots, R_m$ , bao gồm:
  - CPU cycle, không gian bộ nhớ, thiết bị I/O, file, semaphore,...Mỗi loại tài nguyên  $R_i$  có  $W_i$  *thực thể* (instance).
- ❑ Giả sử tài nguyên tái sử dụng theo kỳ (Serially Reusable Resources)
  - *Yêu cầu* (request): process phải chờ nếu yêu cầu không được đáp ứng ngay
  - *Sử dụng* (use): process sử dụng tài nguyên
  - *Hoàn trả* (release): process hoàn trả tài nguyên
- ❑ Các tác vụ yêu cầu (request) và hoàn trả (release) đều là **system call**. Ví dụ
  - request/release device
  - open/close file
  - allocate/free memory
  - wait/signal



# Định nghĩa

---

- ❑ Một tiến trình gọi là *deadlocked* nếu nó đang đợi một sự kiện mà sẽ không bao giờ xảy ra.
  - Thông thường, có nhiều ***hơn một tiến trình*** bị liên quan trong một deadlock.
- ❑ Một tiến trình gọi là trì hoãn vô hạn định (*indefinitely postponed*) nếu nó bị trì hoãn một khoảng thời gian dài lặp đi, lặp lại trong khi hệ thống đáp ứng cho những tiến trình khác .
  - i.e. Một tiến trình sẵn sàng để xử lý nhưng nó ***không bao giờ nhận được CPU***.



# Điều kiện cần để xảy ra deadlock

---

Bốn điều kiện **cần** (necessary condition) để xảy ra deadlock

1. *Loại trừ hồ tương (Mutual exclusion)*: ít nhất một tài nguyên được giữ theo nonsharable mode (ví dụ: printer; ví dụ sharable resource: read-only files).
2. *Giữ và chờ cấp thêm tài nguyên (Hold and wait)*: một process đang giữ ít nhất một tài nguyên và đợi thêm tài nguyên do quá trình khác đang giữ.



## Điều kiện cần để xảy ra deadlock (tt)

---

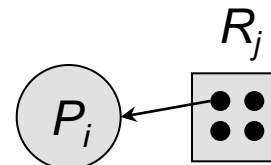
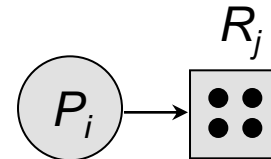
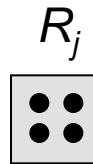
3. *Không trưng dụng (No preemption)*: (= no resource preemption) tài nguyên không thể bị lấy lại, mà chỉ có thể được trả lại từ process đang giữ tài nguyên đó khi nó muốn.
4. *Chu trình đợi (Circular wait)*: tồn tại một tập  $\{P_0, \dots, P_n\}$  các quá trình đang đợi sao cho
  - $P_0$  đợi một tài nguyên mà  $P_1$  đang giữ
  - $P_1$  đợi một tài nguyên mà  $P_2$  đang giữ
  - ...
  - $P_n$  đợi một tài nguyên mà  $P_0$  đang giữ



# Resource Allocation Graph (tt)

## Ký hiệu

- Process:  $P_i$
- Loại tài nguyên với 4 thực thể:
- $P_i$  yêu cầu một thực thể của  $R_j$  :
- $P_i$  đang giữ một thực thể của  $R_j$  :





# Đồ thị cấp phát tài nguyên

## Resource Allocation Graph

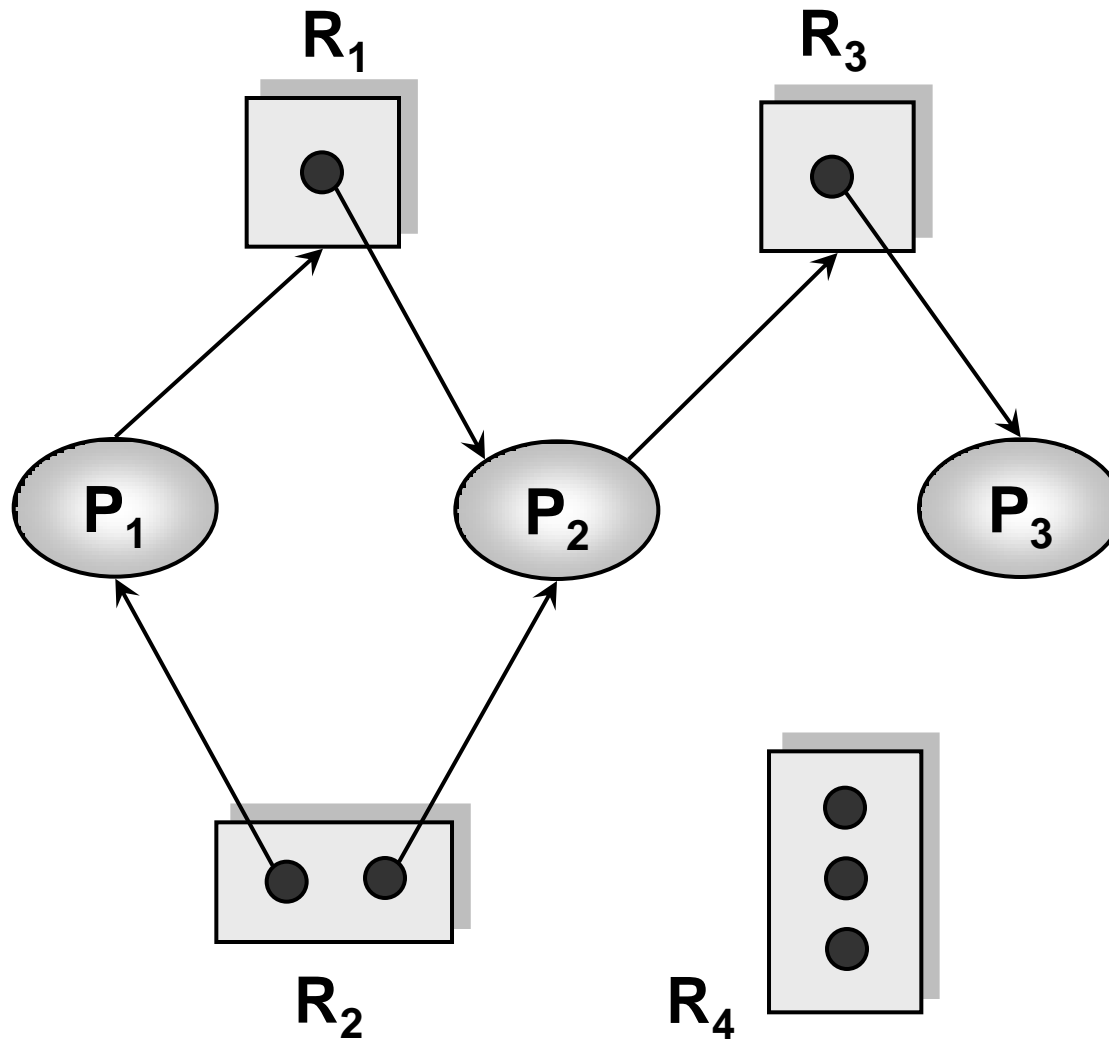
---

- *Resource allocation graph* (RAG) là đồ thị có hướng, với tập đỉnh  $V$  và tập cạnh  $E$ 
  - Tập đỉnh  $V$  gồm 2 loại:
    - $P = \{P_1, P_2, \dots, P_n\}$  (Tất cả process trong hệ thống)
    - $R = \{R_1, R_2, \dots, R_m\}$  (Tất cả các loại tài nguyên trong hệ thống)
  - Tập cạnh  $E$  gồm 2 loại:
    - *Cạnh yêu cầu (Request edge)*:  $P_i \rightarrow R_j$
    - *Cạnh cấp phát (Assignment edge)*:  $R_j \rightarrow P_i$



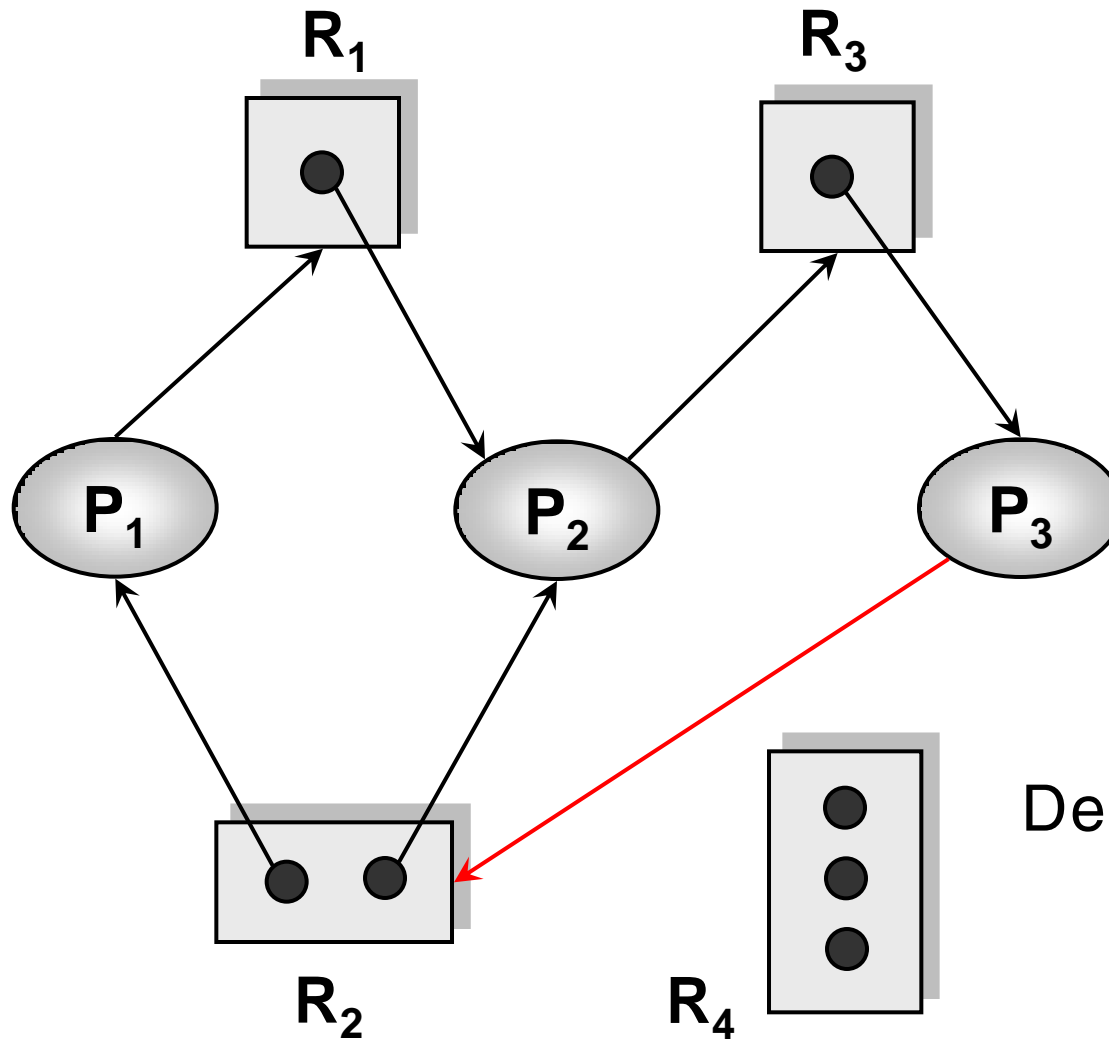


# Ví dụ về RAG





# Ví dụ về RAG (tt)

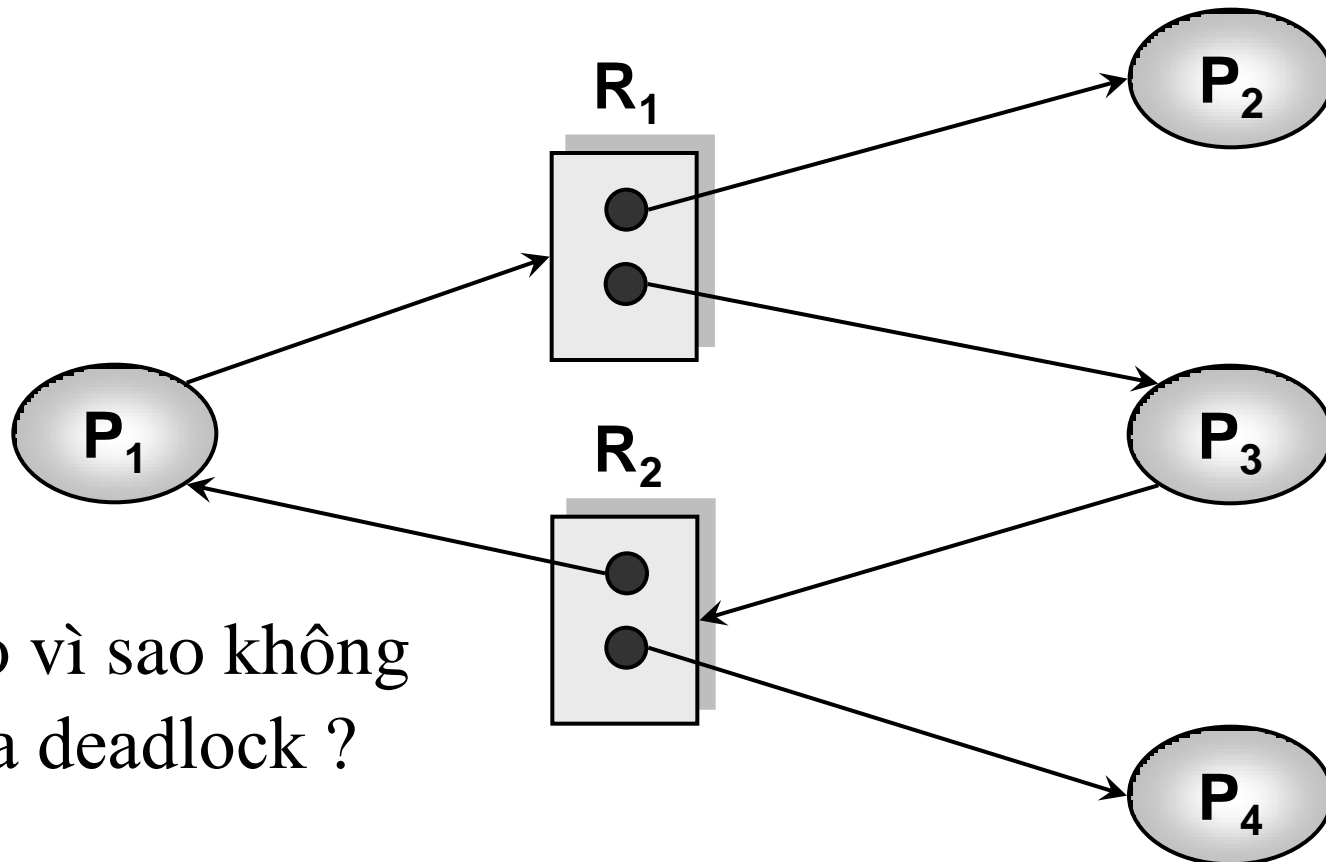


Deadlock xảy ra!



# RAG và deadlock

- Ví dụ một RAG chứa chu trình nhưng không xảy ra deadlock:  $P_4$  có thể trả lại instance của  $R_2$ .



Lý do vì sao không xảy ra deadlock ?



# RAG và deadlock (tt)

---

- ❑ RAG không chứa chu trình (cycle)  $\Rightarrow$  không có deadlock
- ❑ RAG chứa một (hay nhiều) chu trình
  - Nếu mỗi loại tài nguyên chỉ có một thực thể  $\Rightarrow$  deadlock
  - Nếu mỗi loại tài nguyên có nhiều thực thể  $\Rightarrow$  **có thể** xảy ra deadlock



# Các phương pháp giải quyết deadlock (1)

---

Ba phương pháp

1) Bảo đảm rằng hệ thống không rơi vào tình trạng deadlock bằng cách *ngăn* (preventing) hoặc *tránh* (avoiding) deadlock.

Khác biệt

- Ngăn deadlock: không cho phép (ít nhất) một trong 4 điều kiện cần cho deadlock
- Tránh deadlock: các quá trình cần cung cấp thông tin về tài nguyên nó cần để hệ thống cấp phát tài nguyên một cách thích hợp



## Các phương pháp giải quyết deadlock (2)

---

2) Cho phép hệ thống vào trạng thái deadlock, nhưng sau đó phát hiện deadlock và phục hồi hệ thống.

3) Bỏ qua mọi vấn đề, xem như deadlock không bao giờ xảy ra trong hệ thống.

☺ Khá nhiều hệ điều hành sử dụng phương pháp này.

- Deadlock không được phát hiện, dẫn đến việc **giảm hiệu suất** của hệ thống. Cuối cùng, hệ thống có thể ngưng hoạt động và phải được khởi động lại.



# 1. Ngăn deadlock (deadlock prevention)

---

Ngăn deadlock bằng cách ngăn một trong 4 điều kiện cần của deadlock

## 1. Ngăn **mutual exclusion**

- đối với nonsharable resource (vd: printer): không làm được
- đối với sharable resource (vd: read-only file): không cần thiết



# Ngăn deadlock (tt)

## 2. Ngăn Hold and Wait

- Cách 1: mỗi process yêu cầu toàn bộ tài nguyên cần thiết một lần. Nếu có đủ tài nguyên thì hệ thống sẽ cấp phát, nếu không đủ tài nguyên thì process phải bị blocked.
- Cách 2: khi yêu cầu tài nguyên, process không được giữ bất kỳ tài nguyên nào. Nếu đang có thì phải trả lại trước khi yêu cầu.
- Ví dụ để so sánh hai cách trên: một quá trình copy dữ liệu từ tape drive sang disk file, sắp xếp disk file, rồi in kết quả ra printer.
- Khuyết điểm của các cách trên:
  - Hiệu suất sử dụng tài nguyên (resource utilization) thấp
  - Quá trình có thể bị starvation





# Ngăn deadlock (tt)

3. Ngăn **No Preemption**: nếu process A có giữ tài nguyên và đang yêu cầu tài nguyên khác nhưng tài nguyên này chưa cấp phát ngay được thì
- Cách 1: Hệ thống lấy lại mọi tài nguyên mà A đang giữ
    - A chỉ bắt đầu lại được khi có được các tài nguyên đã bị lấy lại cùng với tài nguyên đang yêu cầu
  - Cách 2: Hệ thống sẽ xem tài nguyên mà A yêu cầu
    - Nếu tài nguyên được giữ bởi một process khác **đang đợi thêm tài nguyên**, tài nguyên này được hệ thống lấy lại và cấp phát cho A.
    - Nếu tài nguyên được giữ bởi process **không đợi tài nguyên**, A phải đợi và tài nguyên của A bị lấy lại. Tuy nhiên hệ thống chỉ lấy lại các tài nguyên mà process khác yêu cầu



# Ngăn deadlock (tt)

4. Ngăn **Circular Wait**: gán một thứ tự cho tất cả các tài nguyên trong hệ thống.
- Tập hợp loại tài nguyên:  $R = \{R_1, R_2, \dots, R_m\}$   
Hàm ánh xạ:  $F: R \rightarrow \mathbb{N}$
  - Ví dụ:  $F(\text{tape drive}) = 1, F(\text{disk drive}) = 5, F(\text{printer}) = 12$ 
    - $F$  là hàm định nghĩa thứ tự trên tập các loại tài nguyên.



# Ngăn deadlock (tt)

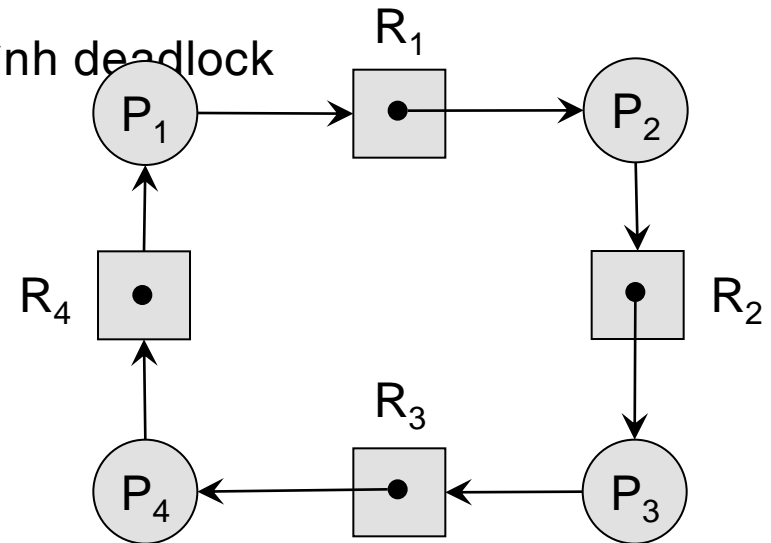
## 4. Ngăn Circular Wait (tt)

- Mỗi process chỉ có thể yêu cầu thực thể của một loại tài nguyên theo thứ tự tăng dần (định nghĩa bởi hàm  $F$ ) của loại tài nguyên. Ví dụ
  - Chuỗi yêu cầu thực thể hợp lệ: tape drive  $\rightarrow$  disk drive  $\rightarrow$  printer
  - Chuỗi yêu cầu thực thể không hợp lệ: disk drive  $\rightarrow$  tape drive
- Khi một process yêu cầu một thực thể của loại tài nguyên  $R_j$  thì nó phải trả lại các tài nguyên  $R_i$  với  $F(R_i) > F(R_j)$ .

- “Chứng minh” giả sử tồn tại một chu trình deadlock

- $F(R_4) < F(R_1)$
- $F(R_1) < F(R_2)$
- $F(R_2) < F(R_3)$
- $F(R_3) < F(R_4)$

Vậy  $F(R_4) < F(R_4)$ , mâu thuẫn!





## 2. Tránh tắc nghẽn

### Deadlock avoidance

---

- ❑ Deadlock prevention sử dụng tài nguyên không hiệu quả.
- ❑ Deadlock avoidance vẫn đảm bảo hiệu suất sử dụng tài nguyên tối đa đến mức có thể.
- ❑ Yêu cầu mỗi process khai báo số lượng tài nguyên tối đa cần để thực hiện công việc
- ❑ Giải thuật deadlock-avoidance sẽ kiểm tra *trạng thái cấp phát tài nguyên* (resource-allocation state) để bảo đảm hệ thống không rơi vào deadlock.

Trạng thái cấp phát tài nguyên được định nghĩa dựa trên số tài nguyên còn lại, số tài nguyên đã được cấp phát và yêu cầu tối đa của các process.



# Trạng thái safe và unsafe

- ❑ Một trạng thái của hệ thống được gọi là *an toàn* (safe) nếu tồn tại một *chuỗi (thứ tự) an toàn* (safe sequence).
- ❑ Một chuỗi quá trình  $\langle P_1, P_2, \dots, P_n \rangle$  là một *chuỗi an toàn* nếu
  - Với mọi  $i = 1, \dots, n$ , yêu cầu tối đa về tài nguyên của  $P_i$  có thể được thỏa bởi
    - tài nguyên mà hệ thống đang có sẵn sàng (available)
    - cùng với tài nguyên mà tất cả  $P_j, j < i$ , đang giữ.
- ❑ Một trạng thái của hệ thống được gọi là *không an toàn* (unsafe) nếu không tồn tại một chuỗi an toàn.



# Chuỗi an toàn (tt)

Ví dụ: Hệ thống có 12 tape drives và 3 quá trình  $P_0$ ,  $P_1$ ,  $P_2$

□ Tại thời điểm  $t_0$

	Maximum needs	Current needs
$P_0$	10	5
$P_1$	4	2
$P_2$	9	2

- Còn 3 tape drive sẵn sàng.
- Chuỗi  $\langle P_1, P_0, P_2 \rangle$  là chuỗi an toàn  $\Rightarrow$  hệ thống là an toàn



# Chuỗi an toàn (tt)

- ❑ Giả sử tại thời điểm  $t_1$ ,  $P_2$  yêu cầu và được cấp phát 1 tape drive
  - còn 2 tape drive sẵn sàng

cần tối đa      đang giữ

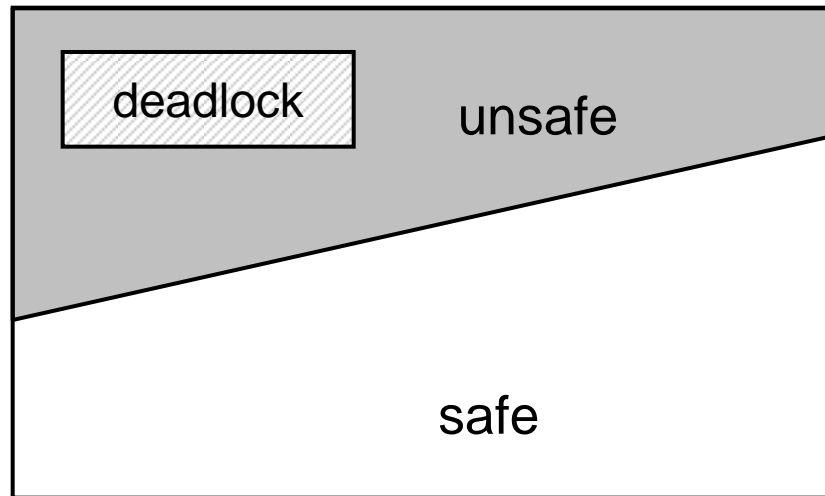
$P_0$	10	5
$P_1$	4	2
$P_2$	9	3

- ❑ Hệ thống còn an toàn không?



# Trạng thái safe/unsafe và deadlock

- ❑ Nếu hệ thống đang ở trạng thái safe  $\Rightarrow$  không deadlock.
- ❑ Nếu hệ thống đang ở trạng thái unsafe  $\Rightarrow$  **có thể** dẫn đến deadlock.
- ❑ Tránh deadlock bằng cách bảo đảm hệ thống không đi đến trạng thái unsafe.

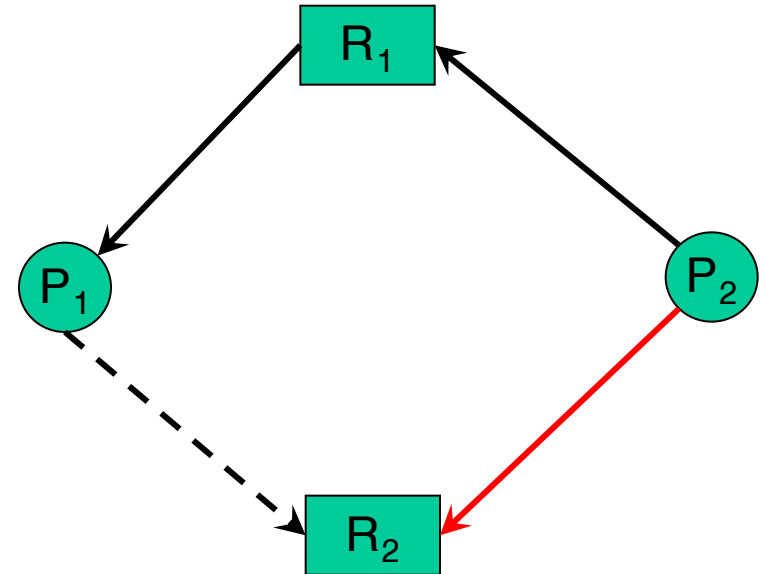
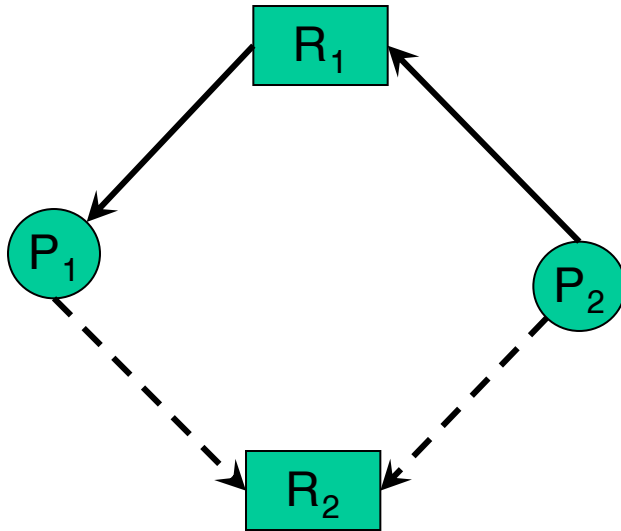






# Giải thuật đồ thị cấp phát tài nguyên

## □ Khái niệm cạnh thỉnh cầu





# Giải thuật banker

---

- ❑ Áp dụng cho hệ thống cấp phát tài nguyên trong đó mỗi loại tài nguyên có thể có nhiều instance.
- ❑ Bắt chước nghiệp vụ ngân hàng (banking)
- ❑ Điều kiện
  - Mỗi process phải khai báo số lượng thực thể (instance) **tối đa** của mỗi loại tài nguyên mà nó cần
  - Khi process yêu cầu tài nguyên thì có thể phải đợi mặc dù tài nguyên được yêu cầu đang có sẵn
  - Khi process đã có được đầy đủ tài nguyên thì phải hoàn trả trong một khoảng thời gian hữu hạn nào đó.



# Giải thuật banker (tt)

$n$ : số process,  $m$ : số loại tài nguyên

Các cấu trúc dữ liệu

**Available**: vector độ dài  $m$

$\text{Available}[j] = k \Leftrightarrow$  loại tài nguyên  $R_j$  có  $k$  instance sẵn sàng

**Max**: ma trận  $n \times m$

$\text{Max}[i, j] = k \Leftrightarrow$  quá trình  $P_i$  yêu cầu tối đa  $k$  instance của loại tài nguyên  $R_j$

**Allocation**: ma trận  $n \times m$

$\text{Allocation}[i, j] = k \Leftrightarrow P_i$  đã được cấp phát  $k$  instance của  $R_j$

**Need**: ma trận  $n \times m$

$\text{Need}[i, j] = k \Leftrightarrow P_i$  cần thêm  $k$  instance của  $R_j$

Nhận xét:  $\text{Need}[i, j] = \text{Max}[i, j] - \text{Allocation}[i, j]$

Ký hiệu  $Y \leq X \Leftrightarrow Y[i] \leq X[i]$ , ví dụ  $(0, 3, 2, 1) \leq (1, 7, 3, 2)$



# Giải thuật banker (tt)

## 1. Giải thuật an toàn

---

Tìm một chuỗi an toàn

1. Gọi **Work** và **Finish** là hai vector độ dài là  $m$  và  $n$ . Khởi tạo

$\text{Work} := \text{Available}$

$\text{Finish}[i] := \text{false}, i = 1, \dots, n$

2. Tìm  $i$  thỏa

(a)  $\text{Finish}[i] = \text{false}$

(b)  $\text{Need}_i \leq \text{Work}$  (hàng thứ  $i$  của  $\text{Need}$ )

Nếu không tồn tại  $i$  như vậy, đến bước 4.

3.  $\text{Work} := \text{Work} + \text{Allocation}_i$

$\text{Finish}[i] := \text{true}$

quay về bước 2.

4. Nếu  $\text{Finish}[i] = \text{true}, i = 1, \dots, n$ , thì hệ thống đang ở trạng thái safe

Thời gian chạy của giải thuật là  $O(m \cdot n^2)$



## Giải thuật banker (tt)

### 2. Giải thuật yêu cầu (cấp phát) tài nguyên

---

Gọi  $\text{Request}_i$  là request vector của process  $P_i$ .

$\text{Request}_i[j] = k \Leftrightarrow P_i$  cần  $k$  instance của tài nguyên  $R_j$ .

1. Nếu  $\text{Request}_i \leq \text{Need}_i$  thì đến bước 2. Nếu không, báo lỗi vì process đã vượt yêu cầu tối đa.
2. Nếu  $\text{Request}_i \leq \text{Available}$  thì qua bước 3. Nếu không,  $P_i$  phải chờ vì tài nguyên không còn đủ để cấp phát.
3. Giả định cấp phát tài nguyên đáp ứng yêu cầu của  $P_i$  bằng cách cập nhật trạng thái hệ thống như sau:

$\text{Available} := \text{Available} - \text{Request}_i$

$\text{Allocation}_i := \text{Allocation}_i + \text{Request}_i$

$\text{Need}_i := \text{Need}_i - \text{Request}_i$



# Giải thuật banker (tt)

## 2. Giải thuật yêu cầu tài nguyên

---

Áp dụng giải thuật kiểm tra trạng thái an toàn lên trạng thái trên

- Nếu trạng thái là safe thì tài nguyên được cấp thực sự cho  $P_i$ .
- Nếu trạng thái là unsafe thì  $P_i$  phải đợi, và phục hồi trạng thái:

Available := Available + Request<sub>i</sub>

Allocation<sub>i</sub> := Allocation<sub>i</sub> – Request<sub>i</sub>

Need<sub>i</sub> := Need<sub>i</sub> + Request<sub>i</sub>



# Giải thuật kiểm tra trạng thái an toàn – Ví dụ

- Có 5 process  $P_0, \dots, P_4$
- Có 3 loại tài nguyên:  $A$  (có 10 instance),  $B$  (5 instance) và  $C$  (7 instance).
- Sơ đồ cấp phát trong hệ thống tại thời điểm  $T_0$

	Allocation			Max			Available			Need			
	$A$	$B$	$C$	$A$	$B$	$C$	$A$	$B$	$C$	$A$	$B$	$C$	
$P_0$	0	1	0	7	5	3	3	3	2	7	4	3	5
$P_1$	2	0	0	3	2	2				1	2	2	1
$P_2$	3	0	2	9	0	2				6	0	0	4
$P_3$	2	1	1	2	2	2				0	1	1	2
$P_4$	0	0	2	4	3	3				4	3	1	3



# GT (kiểm tra trạng thái) an toàn – Vd (tt)

Chuỗi an toàn  $\langle P_1, P_3, P_4, P_2, P_0 \rangle$

	Allocation	Need	Work
	A B C	A B C	A B C
$P_0$	0 1 0	7 4 3	3 3 2
$P_1$	2 0 0	1 2 2	5 3 2
$P_2$	3 0 2	6 0 0	7 4 3
$P_3$	2 1 1	0 1 1	7 4 5
$P_4$	0 0 2	4 3 1	10 4 7
			→ 10 5 7





# GT cấp phát tài nguyên – Ví dụ

- ❑ Yêu cầu (1, 0, 2) của  $P_1$  có thỏa được không?
  - Kiểm tra điều kiện  $\text{Request}_1 \leq \text{Available}$ :
    - $(1, 0, 2) \leq (3, 3, 2)$  là đúng
  - Giả định thỏa yêu cầu, kiểm tra trạng thái mới có phải là safe hay không.

	Allocation			Need			Available		
	A	B	C	A	B	C	A	B	C
$P_0$	0	1	0	7	4	3	2	3	0
$P_1$	3	0	2	0	2	0			
$P_2$	3	0	2	6	0	0			
$P_3$	2	1	1	0	1	1			
$P_4$	0	0	2	4	3	1			

$P_4$  (3, 3, 0) ?

$P_0$  (0, 2, 0) ?

$P_3$  (0, 2, 1) ?

- Trạng thái mới là safe (chuỗi an toàn là  $\langle P_1, P_3, P_4, P_0, P_2 \rangle$ ), vậy có thể cấp phát tài nguyên cho  $P_1$ .



### 3. Phát hiện deadlock (Deadlock detection)

---

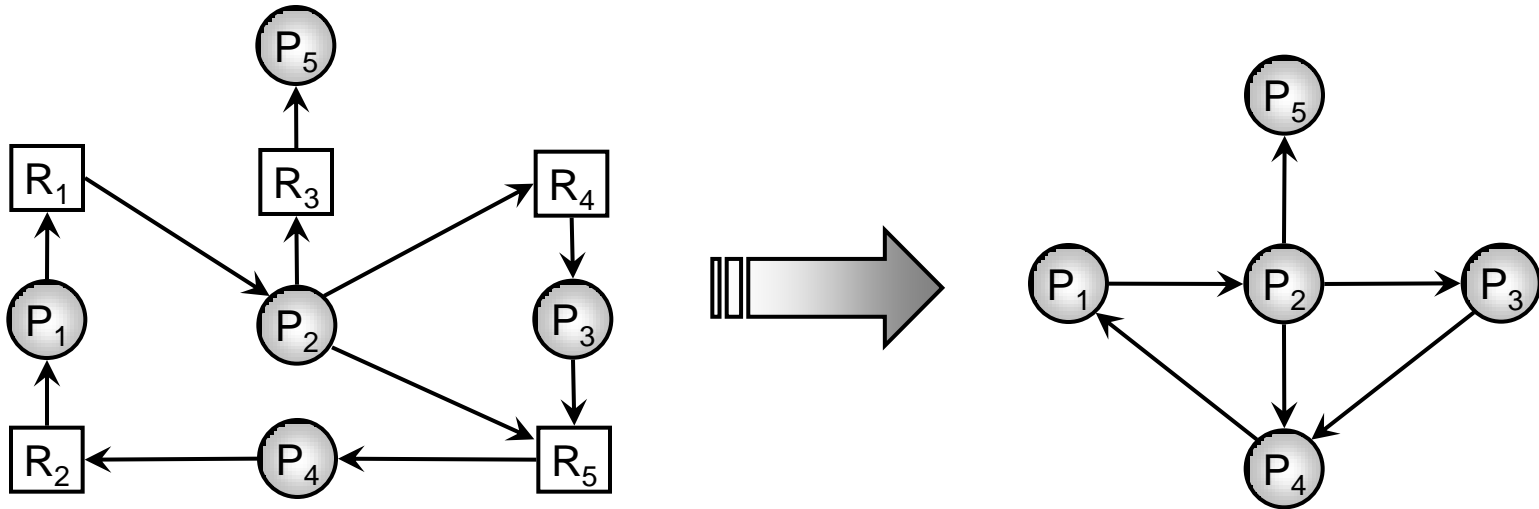
- ❑ Chấp nhận xảy ra deadlock trong hệ thống, kiểm tra trạng thái hệ thống bằng giải thuật phát hiện deadlock.
- ❑ Nếu có deadlock thì tiến hành phục hồi hệ thống
- ❑ Các giải thuật phát hiện deadlock thường sử dụng mô hình RAG.
- ❑ Hệ thống cấp phát tài nguyên được khảo sát trong mỗi trường hợp sau
  1. Mỗi loại tài nguyên chỉ có một thực thể (instance)
  2. Mỗi loại tài nguyên có thể có nhiều thực thể



# Mỗi loại tài nguyên chỉ có một thực thể

## ❑ Sử dụng *wait-for graph*

- Wait-for graph được dẫn xuất từ RAG bằng cách bỏ các node biểu diễn tài nguyên và ghép các cạnh tương ứng.
  - Có cạnh từ  $P_i$  đến  $P_j \Leftrightarrow P_i$  đang chờ tài nguyên từ  $P_j$



- ❑ Một giải thuật kiểm tra có tồn tại chu trình trong wait-for graph hay không sẽ được gọi **định kỳ**. Giải thuật phát hiện chu trình có thời gian chạy là  $O(n^2)$ , với  $n$  là số đỉnh của graph.



# Mỗi loại tài nguyên có nhiều thực thể

---

- ❑ Phương pháp dùng wait-for graph không áp dụng được cho trường hợp mỗi loại tài nguyên có nhiều instance.
- ❑ Các cấu trúc dữ liệu dùng trong giải thuật phát hiện deadlock

*Available*: vector độ dài  $m$

số instance sẵn sàng của mỗi loại tài nguyên

*Allocation*: ma trận  $n \times m$

số instance của mỗi loại tài nguyên đã cấp phát cho mỗi process

*Request*: ma trận  $n \times m$

yêu cầu hiện tại của mỗi process.

$\text{Request}[i, j] = k \Leftrightarrow P_i \text{ đang yêu cầu thêm } k \text{ instance của } R_j$



# Giải thuật phát hiện deadlock

1. Gọi **Work** và **Finish** là vector kích thước  $m$  và  $n$ . Khởi tạo:

$Work := Available$

$i = 1, 2, \dots, n$ , nếu  $Allocation_i \neq 0$  thì  $Finish[i] := false$   
còn không thì  $Finish[i] := true$

2. Tìm  $i$  thỏa mãn:

$Finish[i] := false$  và

$Request_i \leq Work$

Nếu không tồn tại  $i$  như thế, đến bước 4.

thời gian chạy  
của giải thuật

$O(m \cdot n^2)$

3.  $Work := Work + Allocation_i$

$Finish[i] := true$

quay về bước 2.

4. Nếu  $Finish[i] = false$ , với một  $i = 1, \dots, n$ , thì hệ thống đang ở trạng thái **deadlock**. Hơn thế nữa,  $Finish[i] = false$  thì  $P_i$  bị **deadlocked**.



# Giải thuật phát hiện deadlock – Ví dụ

- Hệ thống có 5 quá trình  $P_0, \dots, P_4$   
3 loại tài nguyên:  $A$  (7 instance),  $B$  (2 instance),  $C$  (6 instance).

	Allocation			Request			Available		
	$A$	$B$	$C$	$A$	$B$	$C$	$A$	$B$	$C$
$P_0$	0	1	0	0	0	0	0	0	0
$P_1$	2	0	0	2	0	2			
$P_2$	3	0	3	0	0	0			
$P_3$	2	1	1	1	0	0			
$P_4$	0	0	2	0	0	2			

Chạy giải thuật, tìm được chuỗi  $\langle P_0, P_2, P_3, P_1, P_4 \rangle$  với  $\text{Finish}[i] = \text{true}$ ,  $i = 1, \dots, n$ , vậy hệ thống không bị deadlock.



# Giải thuật phát hiện deadlock – Ví dụ (tt)

- $P_2$  yêu cầu thêm một instance của  $C$ . Ma trận Request như sau:

	Request		
	$A$	$B$	$C$
$P_0$	0	0	0
$P_1$	2	0	2
$P_2$	0	0	1
$P_3$	1	0	0
$P_4$	0	0	2

- Trạng thái của hệ thống là gì?
  - Có thể thu hồi tài nguyên đang sở hữu bởi process  $P_0$  nhưng vẫn không đủ đáp ứng yêu cầu của các process khác.  
Vậy tồn tại deadlock, bao gồm các process  $P_1$ ,  $P_2$ ,  $P_3$ , và  $P_4$ .



# Phục hồi deadlock (Deadlock Recovery)

---

- ❑ Khi deadlock xảy ra, để phục hồi
  - báo người vận hành (operator)hoặc
  - hệ thống tự động phục hồi bằng cách bỏ gậy chu trình deadlock:
    - chấm dứt một hay nhiều quá trình
    - lấy lại tài nguyên từ một hay nhiều quá trình





# Deadlock Recovery: Chấm dứt quá trình

---

- ❑ Phục hồi hệ thống bị deadlock bằng cách chấm dứt quá trình
  - Chấm dứt tất cả process bị deadlocked, hoặc
  - Chấm dứt lần lượt từng process cho đến khi không còn deadlock
    - Sử dụng giải thuật phát hiện deadlock để xác định còn deadlock hay không
  
- ❑ Dựa trên yếu tố nào để chọn process cần được chấm dứt?
  - Độ ưu tiên của process
  - Thời gian đã thực thi của process và thời gian còn lại
  - Loại tài nguyên mà process đã sử dụng
  - Tài nguyên mà process cần thêm để hoàn tất công việc
  - Số lượng process cần được chấm dứt
  - Process là interactive process hay batch process



# Deadlock recovery: Lấy lại tài nguyên

---

- ❑ Lấy lại tài nguyên từ một process, cấp phát cho process khác cho đến khi không còn deadlock nữa.
- ❑ Các vấn đề trong chiến lược thu hồi tài nguyên:
  - Chọn “nạn nhân” để tối thiểu chi phí (có thể dựa trên số tài nguyên sở hữu, thời gian CPU đã tiêu tốn,...)
  - Trở lại trạng thái trước deadlock (Rollback): rollback process bị lấy lại tài nguyên trở về trạng thái safe, tiếp tục process từ trạng thái đó. Hệ thống cần lưu giữ một số thông tin về trạng thái các process đang thực thi.
  - Đói tài nguyên (Starvation): để tránh starvation, phải bảo đảm không có process sẽ luôn luôn bị lấy lại tài nguyên mỗi khi deadlock xảy ra.



# Phương pháp kết hợp để giải quyết Deadlock

---

## ❑ Kết hợp 3 phương pháp cơ bản

- Ngăn chặn (Prevention)
- Tránh (Avoidance)
- Phát hiện (Detection)

Cho phép sử dụng cách giải quyết tối ưu cho mỗi lớp tài nguyên trong hệ thống.

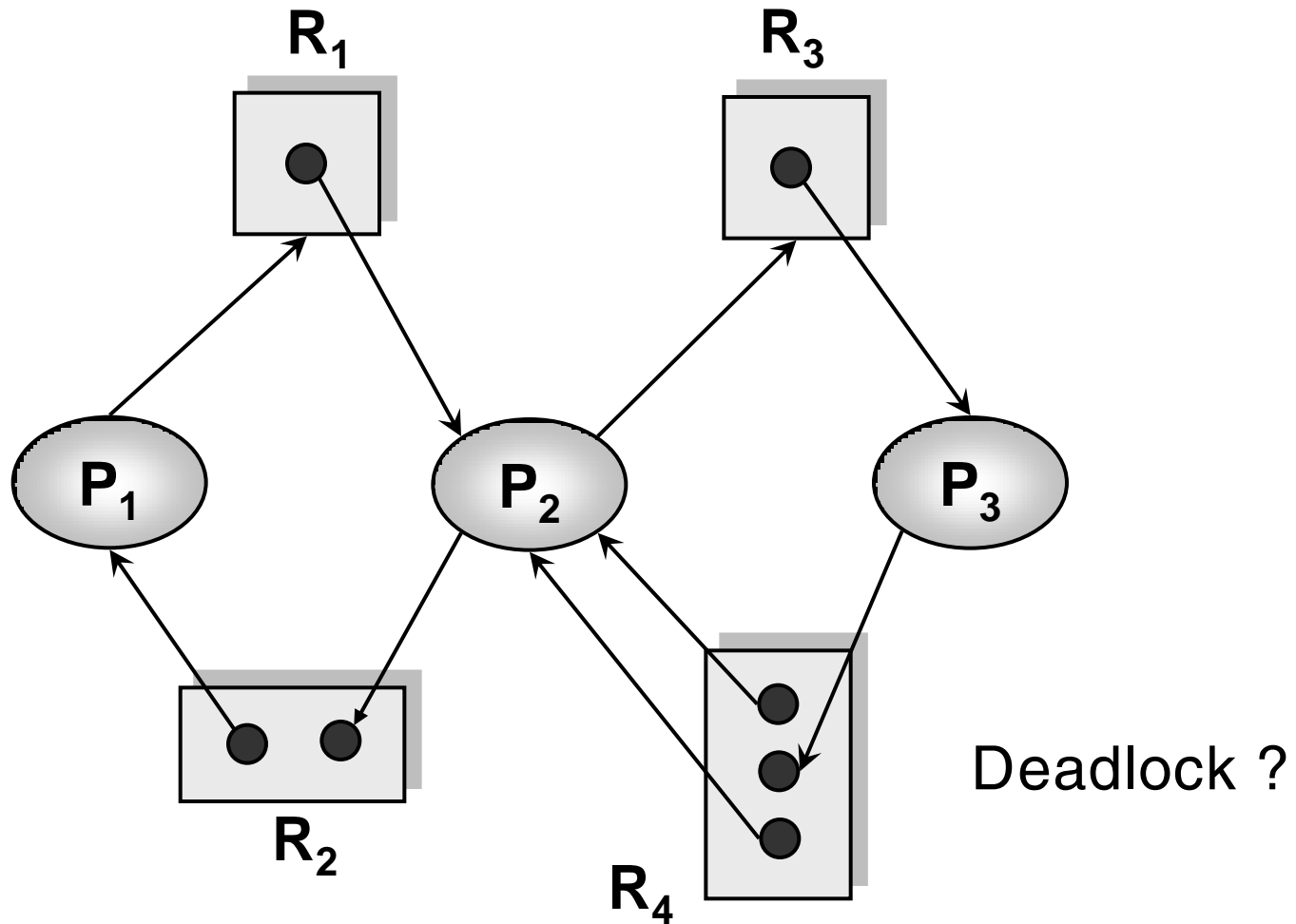
## ❑ Phân chia tài nguyên thành các lớp theo thứ bậc.

- Sử dụng kỹ thuật thích hợp nhất cho việc quản lý deadlock trong mỗi lớp này.



# Bài tập

- ❑ Bài 01: Liệt kê 3 trường hợp xảy ra deadlock trong đời sống
- ❑ Bài 02:





# Bài tập

## □ Bài 03:

	<u>Allocation</u>	<u>Max</u>	<u>Available</u>
	<i>A B C D</i>	<i>A B C D</i>	<i>A B C D</i>
$P_0$	0 0 1 2	0 0 1 2	1 5 2 0
$P_1$	1 0 0 0	1 7 5 0	
$P_2$	1 3 5 4	2 3 5 6	
$P_3$	0 6 3 2	0 6 5 2	
$P_4$	0 0 1 4	0 6 5 6	

- A) Tìm Need
- B) Hệ thống có an toàn không
- C) Nếu  $P_1$  yêu cầu (0,4,2,0) thì có thể cấp phát cho nó ngay không?