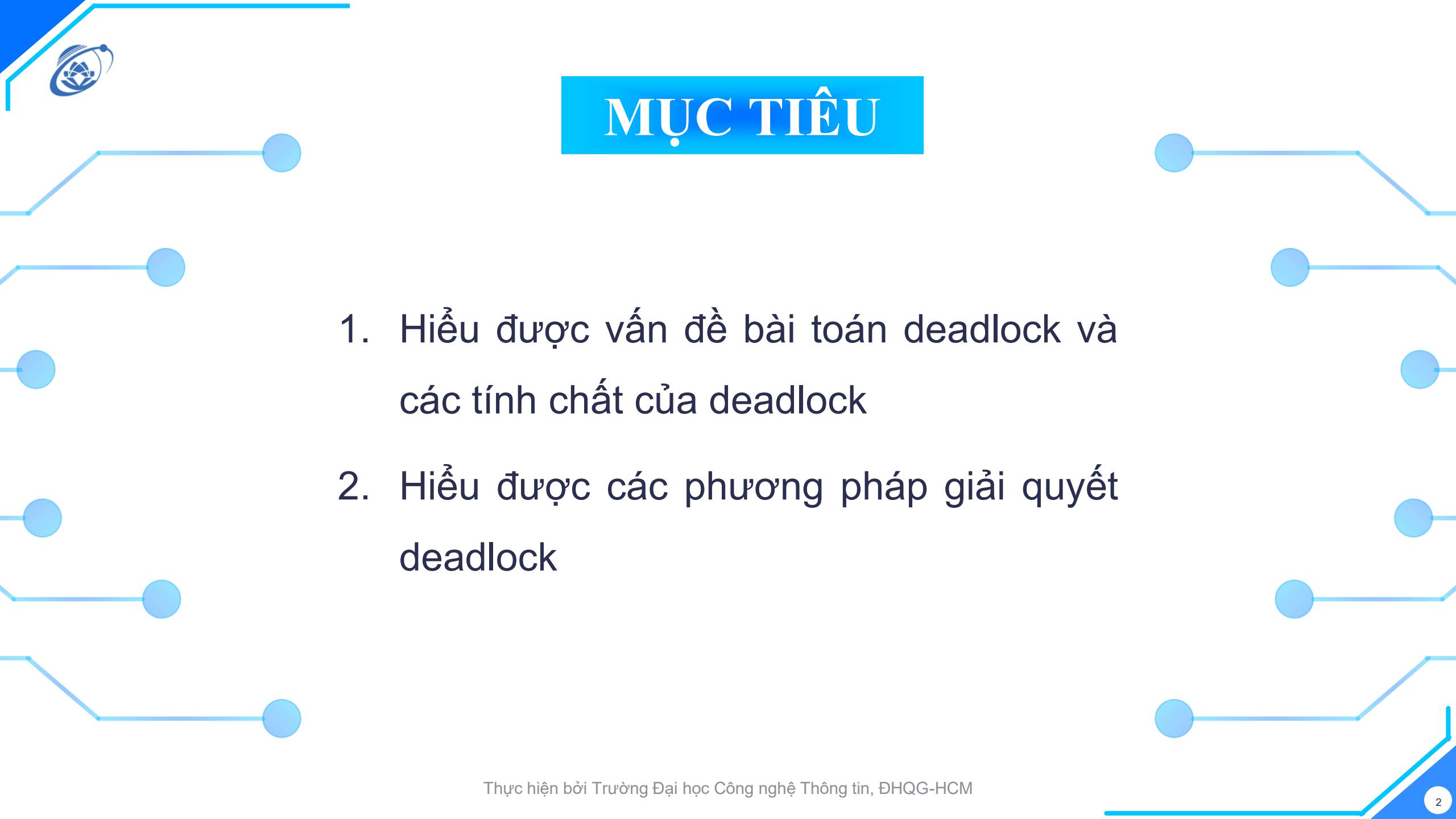




# HỆ ĐIỀU HÀNH

## CHƯƠNG 6: DEADLOCK

Trình bày các khái niệm cơ bản về deadlock, các phương pháp giải quyết deadlock



# MỤC TIÊU

1. Hiểu được vấn đề bài toán deadlock và các tính chất của deadlock
2. Hiểu được các phương pháp giải quyết deadlock



# NỘI DUNG

1. Vấn đề deadlock
2. Mô hình hệ thống
3. Phương pháp giải quyết deadlock



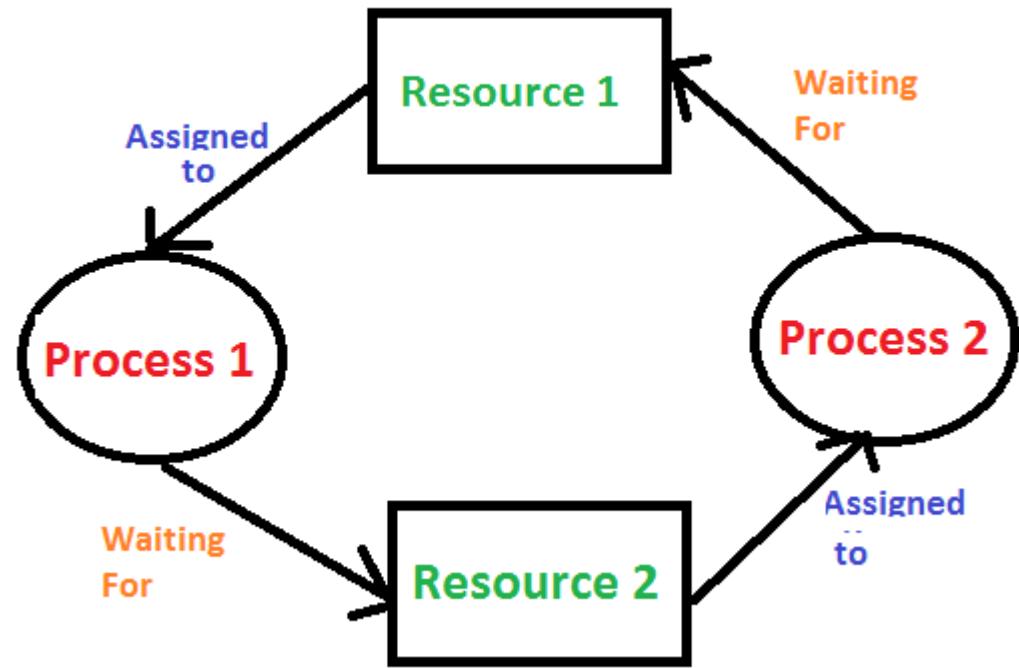
# VẤN ĐỀ DEADLOCK

## 6.1.1. Vấn đề Deadlock

01.



## 6.1.1. Vấn đề deadlock





## 6.1.1. Vấn đề deadlock

- Gọi S và Q là hai biến semaphore được khởi tạo  $S.v = Q.v = 1$

P0

wait(S); ( $S.v=0$ )

wait(Q); ( $Q.v=-1$ )

P1

wait(Q); ( $Q.v=0$ )

wait(S); ( $S.v=-1$ )

signal(S);

signal(Q);

signal(Q);

signal(S);

- Giả sử HĐH sử dụng giải thuật RR và quantum time bằng thời gian thực thi hàm wait(), khi đó P0 thực thi wait(S), rồi P1 thực thi wait(Q), rồi P0 thực thi wait(Q) bị blocked, P1 thực thi wait(S) bị blocked.



## 6.1.1. Vấn đề deadlock

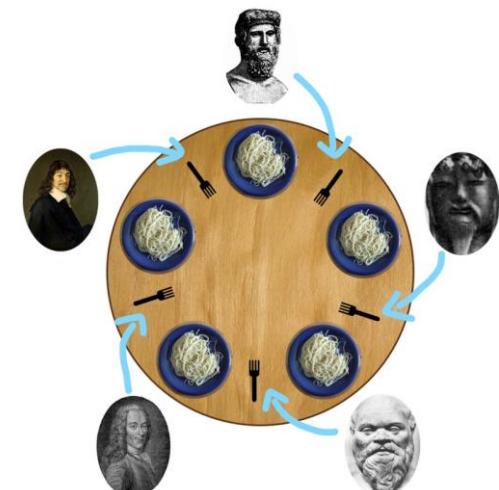
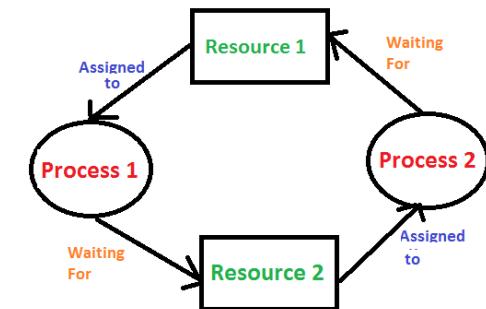
☐ Tình huống: Một tập các tiến trình bị block, mỗi tiến trình giữ tài nguyên và đang chờ tài nguyên mà tiến trình khác trong tập đang giữ.

☐ Ví dụ 1:

- Hệ thống có 2 file trên đĩa.
- P1 và P2 mỗi tiến trình mở một file và yêu cầu mở file kia

☐ Ví dụ 2:

- Bài toán các triết gia ăn tối.
- Mỗi người cầm 1 chiếc đũa và chờ chiếc còn lại.





# VẤN ĐỀ DEADLOCK

---

## 6.1.2. Định nghĩa

01.



## 6.1.2. Định nghĩa

- Một tiến trình gọi là deadlock 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 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.
  - Ví dụ: Một tiến trình sẵn sàng để xử lý nhưng nó không bao giờ nhận được CPU.



# VẤN ĐỀ DEADLOCK

## 6.1.3. Điều kiện cần để xảy ra deadlock

01.



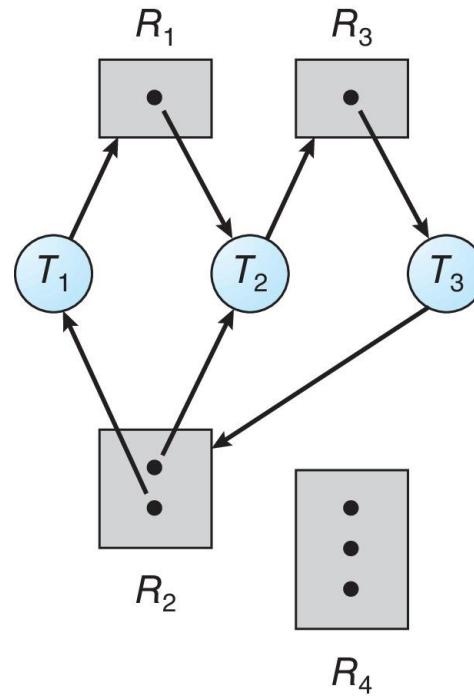
## 6.1.3. Điều kiện cần để xảy ra deadlock

- **Loại trừ tương hỗ:** ít nhất một tài nguyên được giữ theo nonshareable mode.
  - Ví dụ: printer <> read-only files (sharable)
- **Giữ và chờ cấp thêm tài nguyên:** Một tiến trình đang giữ ít nhất một tài nguyên và đợi thêm tài nguyên do tiến trình khác giữ.



## 6.1.3. Điều kiện cần để xảy ra deadlock

- **Không trung dụng:** tài nguyên không thể bị lấy lại mà chỉ có thể được trả lại từ tiến trình đang giữ tài nguyên đó khi nó muốn.
- **Chu trình đợi:** tồn tại một tập  $\{P_0, \dots, P_n\}$  các tiến trình đang đợi sao cho
  - $P_0$  đợi một tài nguyên mà  $P_1$  giữ.
  - $P_1$  đợi một tài nguyên mà  $P_2$  giữ.
  - ...
  - $P_n$  đợi một tài nguyên mà  $P_0$  giữ.





# MÔ HÌNH HÓA HỆ THỐNG

02.



## 6.2. Mô hình hóa hệ thống

- Các loại tài nguyên, kí hiệu R1, R2,...,Rm, bao gồm:
  - CPU cycle, không gian bộ nhớ, thiết bị I/O, file, semaphore,..
  - Mỗi loại tài nguyên Ri có Wi thực thể.
- Giả sử tài nguyên tái sử dụng theo chu kỳ:
  - Yêu cầu: tiến trình phải chờ nếu yêu cầu không được đáp ứng ngay.
  - Sử dụng: tiến trình sử dụng tài nguyên.
  - Hoàn trả: tiến trình hoàn trả tài nguyên.
- Các tác vụ yêu cầu và hoàn trả đều là system call. Ví dụ:
  - Request/ release device.
  - Open / close file.
  - Allocate/ free memory.
  - Wait/ signal.



# MÔ HÌNH HÓA HỆ THỐNG

## 6.2.1. Đồ thị cấp phát tài nguyên RAG

02.



## 6.2.1. Đồ thị cấp phát tài nguyên - RAG

- Là đồ thị có hướng, với tập đỉnh V và tập cạnh E.
- Tập đỉnh V gồm 2 loại:
  - $P = \{P_1, P_2, \dots, P_n\}$  (All process)
  - $R = \{R_1, R_2, \dots, R_n\}$  (All resource)
- Tập cạnh E gồm 2 loại:
  - Cạnh yêu cầu:  $P_i \rightarrow R_j$
  - Cạnh cấp phát:  $R_j \rightarrow P_i$

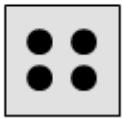


## 6.2.1. Đồ thị cấp phát tài nguyên - RAG

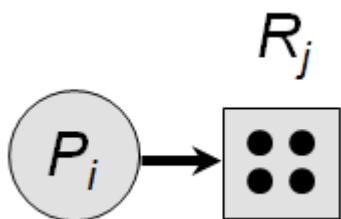
- Process i



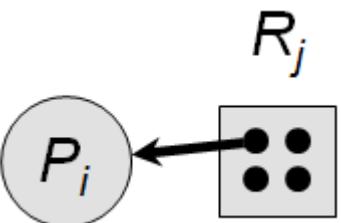
- Loại tài nguyên  $R_j$  với 4 thực thể

 $R_j$ 

- $P_i$  yêu cầu một thực thể của  $R_j$



- $P_i$  đang giữ một thực thể của  $R_j$





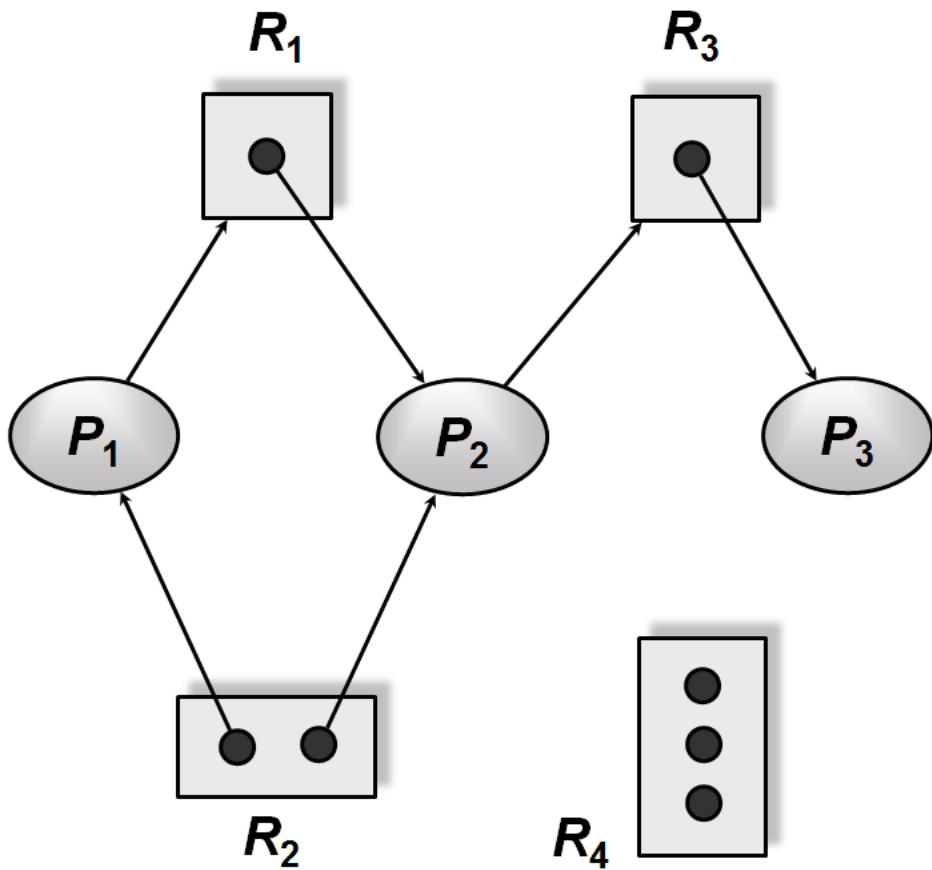
# MÔ HÌNH HÓA HỆ THỐNG

## 6.2.2. Các ví dụ

02.

## 6.2.2. Các ví dụ

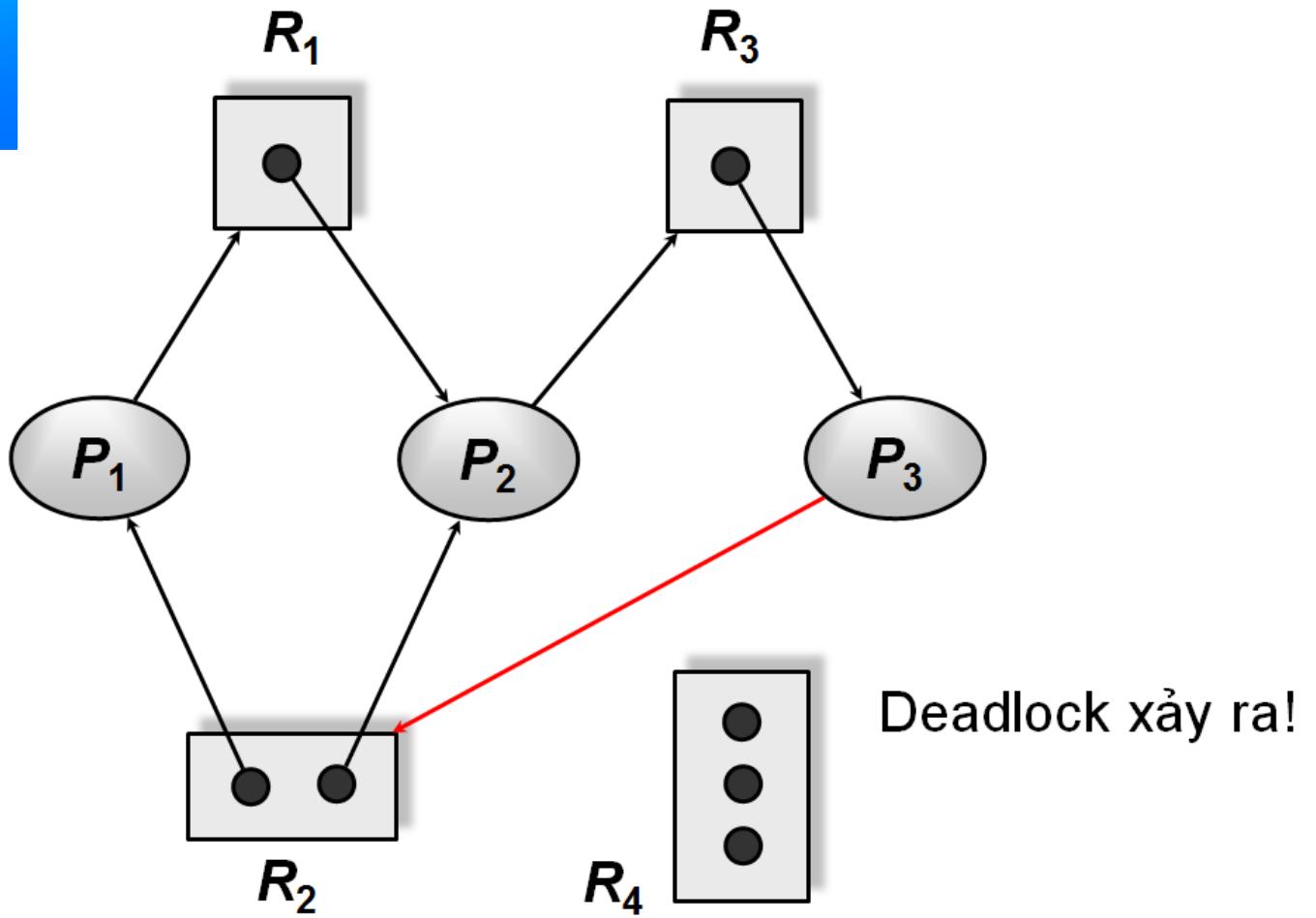
Cho 1 hệ thống có 3 tiến trình P1  
đến P3 và 4 loại tài nguyên R1 (1),  
R2 (2), R3 (1) và R4 (4). P1 giữ 1  
R2 và yêu cầu 1 R1; P2 giữ 1 R2,  
1 R1 và yêu cầu 1 R3; P3 giữ 1  
R3.





## 6.2.2. Các ví dụ

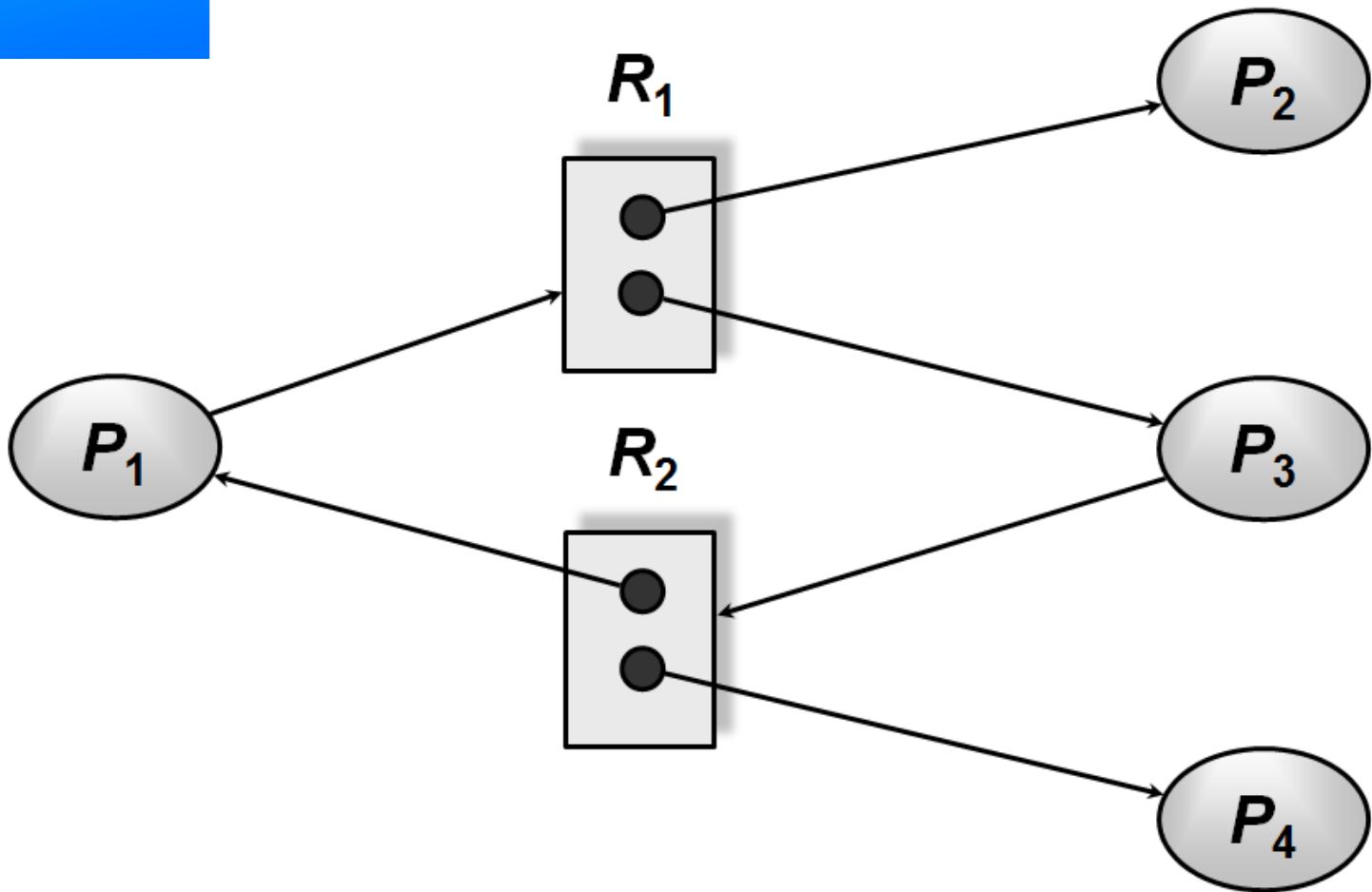
Đồ thị cấp phát tài nguyên  
với một deadlock





## 6.2.2. Các ví dụ

Đồ thị chứa chu trình nhưng  
không deadlock





# MÔ HÌNH HÓA HỆ THỐNG

## 6.2.3. RAG và deadlock

02.



## 6.2.3. RAG và deadlock

- RAG không chứa chu trình => không có deadlock.
- RAG chứa một (hay nhiều) chu trình:
  - Nếu mỗi loại tài nguyên chỉ có một thực thể  
=> deadlock
  - Nếu mỗi loại tài nguyên có nhiều thực thể  
=> có thể xảy ra deadlock



# Bài tập 1

- Cho 1 hệ thống có 4 tiến trình P1 đến P4 và 3 loại tài nguyên R1 (3), R2 (2) R3 (2). P1 giữ 1 R1 và yêu cầu 1 R2; P2 giữ 2 R2 và yêu cầu 1 R1 và 1 R3; P3 giữ 1 R1 và yêu cầu 1 R2; P4 giữ 2 R3 và yêu cầu 1 R1.
  - Vẽ đồ thị tài nguyên cho hệ thống này?
  - Deadlock?
  - Chuỗi an toàn? (nếu có)



# CÁC PHƯƠNG PHÁP GIẢI QUYẾT DEADLOCK

03.



## 6.3. Các phương pháp giải quyết deadlock

1. Ngăn deadlock: không cho phép (ít nhất) một trong 4 điều kiện cần cho deadlock.
2. Tránh deadlock: các tiến trình cần cung cấp thông tin về tài nguyên nó cần để hệ thống cấp phát tài nguyên một cách thích hợp.
3. Cho phép hệ thống vào trạng thái deadlock, nhưng sau đó phát hiện deadlock và phục hồi hệ thống.
4. Bỏ qua mọi vấn đề, xem như deadlock không bao giờ xảy ra trong hệ thống.
  - 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 khởi động lại.



# CÁC PHƯƠNG PHÁP GIẢI QUYẾT DEADLOCK

## 6.3.1. Ngăn deadlock

03.



## 6.3.1. Ngăn deadlock

- Ngăn deadlock bằng cách ngăn một trong 4 điều kiện cần của deadlock.
- **Ngăn mutual exclusion**
  - Đối với tài nguyên không chia sẻ (printer): không làm được.
  - Đối với tài nguyên chia sẻ (read-only file): không cần thiết.
- **Hold and wait**
  - Cách 1: Mỗi tiến trình 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 tiến trình phải bị block.
  - Cách 2: Khi yêu cầu tài nguyên, tiến trình không được giữ tài nguyên nào. Nếu đang có thì phải trả lại trước khi yêu cầu.



## 6.3.1. Ngăn deadlock

- **Ngăn no preemption:** nếu tiến trình A có giữ tài nguyên và đang yêu cầu tài nguyên khác nhưng tài nguyên này chưa được cấp phát ngay 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 tiến trình 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 tiến trình 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 tiến trình khác yêu cầu.



## 6.3.1. Ngăn deadlock

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



## 6.3.1. Ngăn deadlock

- **Ngăn chu trình đợi:**

- Mỗi tiến trình 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: file → disk → printer
  - Khi một tiến trình yêu cầu một thực thể của loại tài nguyên  $R_j$  thì nó phải trả lại các tài nguyên  $R_i$  với  $F(R_i) > F(R_j)$ .



# CÁC PHƯƠNG PHÁP GIẢI QUYẾT DEADLOCK

## 6.3.2. Tránh deadlock

03.



## 6.3.2. Tránh deadlock

- Ngăn deadlock sử dụng tài nguyên không hiệu quả.
- Tránh deadlock vẫn đảm bảo hiệu suất sử dụng tài nguyên tối đa đến mức có thể.
- Yêu cầu mỗi tiến trình khai báo số lượng tài nguyên tối đa cần để thực hiện công việc.
- Giải thuật tránh deadlock sẽ kiểm tra trạng thái cấp phát tài nguyên để đảm bảo 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 tiến trình.



## 6.3.2.1. 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**.
- Một chuỗi tiến 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.
    - Cùng với tài nguyên mà tất cả các  $P_j$  ( $j < i$ ) đang giữ.
- Một trạng thái của hệ thống được gọi là **không an toàn** (unsafe) nếu không tồn tại một chuỗi an toàn.



## 6.3.2.1. Trạng thái safe và unsafe

- Ví dụ: hệ thống có 12 file và 3 tiến trình P0, P1, P2
  - Tại thời điểm  $t_0$

	Cần tối đa	Đang giữ	Cần thêm
P0	10	5	5
P1	4	2	2
P2	9	2	7

- Còn 3 file sẵn sàng
- Chuỗi  $\langle P1, P0, P2 \rangle$  là chuỗi an toàn  $\rightarrow$  hệ thống là an toàn



## 6.3.2.1. Trạng thái safe và unsafe

- Giả sử tại thời điểm  $t_1$ , P2 yêu cầu và được cấp phát 1 file
  - Còn 2 file sẵn sàng

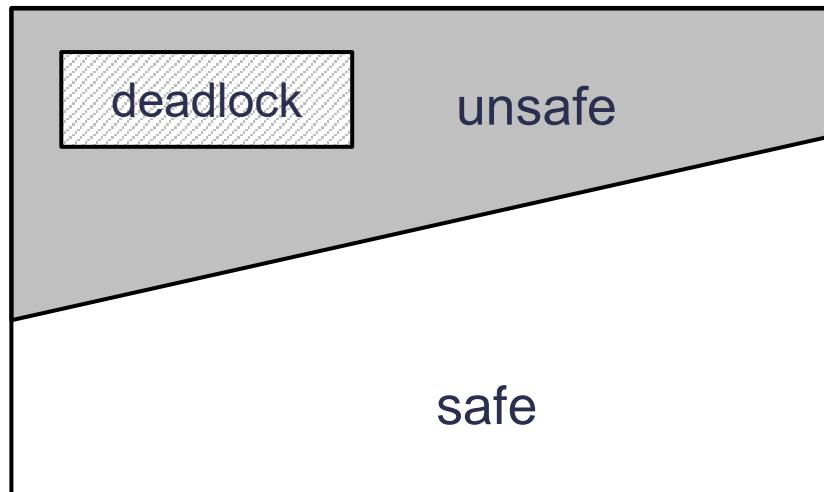
	Cần tối đa	Đang giữ
P0	10	5
P1	4	2
P2	9	3

- Hệ thống còn an toàn không?



## 6.3.2.2. Trạng thái safe/unsafe và deadlock

- Nếu hệ thống đang ở trạng thái safe => không deadlock.
- Nếu hệ thống đang ở trạng thái unsafe => có thể dẫn đến deadlock.
- Tránh deadlock bằng cách bảo đảm hệ thống không đi đến trạng thái unsafe.





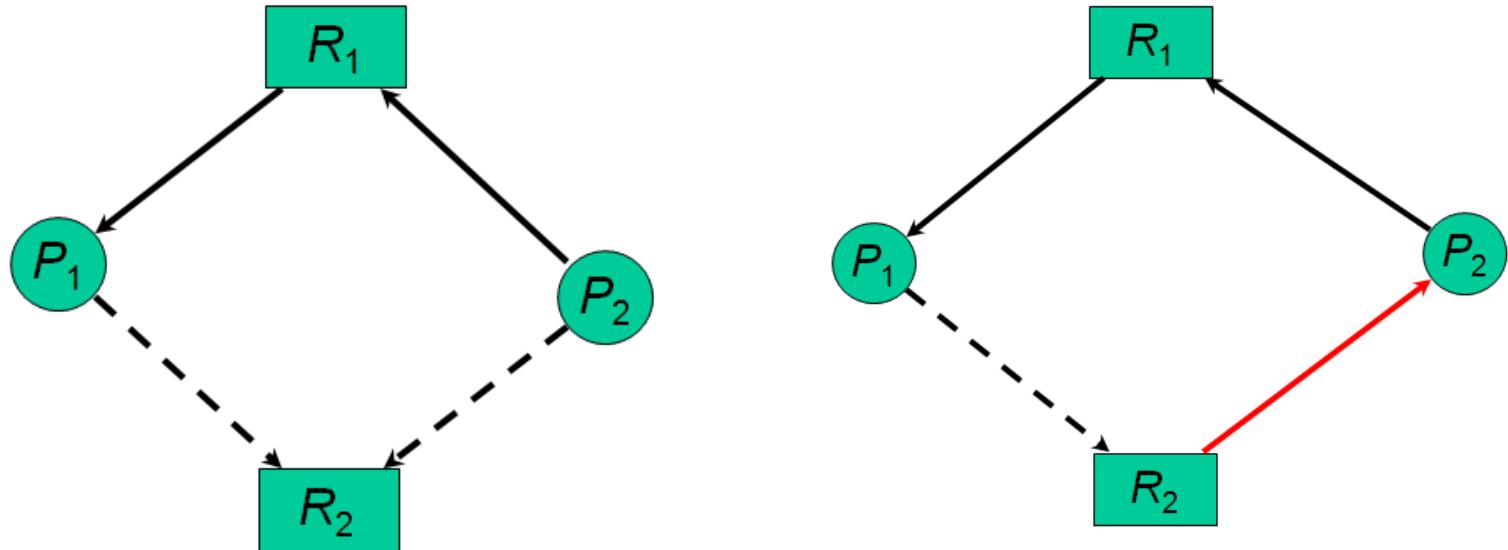
### 6.3.2.3. Các giải thuật tránh deadlock

- Mỗi tài nguyên chỉ có một thực thể
  - Giải thuật đồ thị cấp phát tài nguyên
- Mỗi tài nguyên có nhiều thực thể
  - Giải thuật Banker



### 6.3.2.3. Các giải thuật tránh deadlock

#### Giải thuật đồ thị cấp phát tài nguyên





## 6.3.2.3. Các giải thuật tránh deadlock

### Giải thuật Banker

- Mỗi loại tài nguyên có nhiều thực thể.
- Bắt chước nghiệp vụ ngân hàng.
- Điều kiện:
  - Mỗi tiến trình phải khai báo số lượng thực thể tối đa của mỗi loại tài nguyên mà nó cần.
  - Khi tiến trình yêu cầu tài nguyên thì có thể phải đợi.
  - Khi tiến trình đã có được đầy đủ tài nguyên thì phải hoàn trả trong một khoảng thời gian hữu hạn nào đó.



# Cấu trúc dữ liệu cho giải thuật Banker

n: số tiến trình; m: số loại tài nguyên

- **Available**: vector độ dài m
  - $\text{Available}[j] = k \Rightarrow$  loại tài nguyên  $R_j$  có  $k$  instance sẵn sàng
- **Max**: ma trận  $n \times m$ 
  - $\text{Max}[i, j] = k \Rightarrow$  tiến trình  $P_i$  yêu cầu tối đa  $k$  instance của loại tài nguyên  $R_j$
- **Allocation**: ma trận  $n \times m$ 
  - $\text{Allocation}[i, j] = k \Rightarrow P_i$  đã được cấp phát  $k$  instance của  $R_j$
- **Need**: ma trận  $n \times m$ 
  - $\text{Need}[i, j] = k \Rightarrow P_i$  cần thêm  $k$  instance của  $R_j$
  - $\Rightarrow \text{Need}[i, j] = \text{Max}[i, j] - \text{Allocation}[i, j]$

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



# Giải thuật an toàn

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

Work = Available

Finish[i] = **false**,  $i = 0, 1, \dots, n-1$

2. Tìm  $i$  thỏa

(a) Finish[i] = **false**

(b)  $\text{Need}_{:,i} \leq \text{Work}$  (hàng thứ  $i$  của Need)

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

3.  $\text{Work} = \text{Work} + \text{Allocation}_{:,i}$

Finish[i] = **true**

quay về bước 2

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



# Giải thuật an toàn - Ví dụ

- 5 tiến trình P0,...,P4
- 3 loại tài nguyên:
  - A (10 thực thể), B (5 thực thể), C (7 thực thể)
- Sơ đồ cấp phát trong hệ thống tại thời điểm T0

	Allocation			Max			Available			Need		
	A	B	C	A	B	C	A	B	C	A	B	C
P0	0	1	0	7	5	3	3	3	2	7	4	3
P1	2	0	0	3	2	2				1	2	2
P2	3	0	2	9	0	2				6	0	0
P3	2	1	1	2	2	2				0	1	1
P4	0	0	2	4	3	3				4	3	1



# Giải thuật an toàn - Ví dụ

- 5 tiến trình P0,...,P4
- 3 loại tài nguyên:
  - A (10 thực thể), B (5 thực thể), C (7 thực thể)
- Sơ đồ cấp phát trong hệ thống tại thời điểm T0

	Allocation			Max			Work			Need			Finish
	A	B	C	A	B	C	A	B	C	A	B	C	
P0	0	1	0	7	5	3	3	3	2	7	4	3	
P1	2	0	0	3	2	2				1	2	2	
P2	3	0	2	9	0	2				6	0	0	
P3	2	1	1	2	2	2				0	1	1	
P4	0	0	2	4	3	3				4	3	1	



# Giải thuật yêu cầu tài nguyên cho tiến trình Pi

*Request<sub>i</sub>* 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 tiến trình 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 yêu cầu tài nguyên cho tiến trình Pi

- Áp dụng giải thuật kiểm tra trạng thái an toàn lên trạng thái trên hệ thống mới:

- 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:

$$\text{Available} = \text{Available} + \text{Request}_i$$

$$\text{Allocation}_i = \text{Allocation}_i - \text{Request}_i$$

$$\text{Need}_i = \text{Need}_i + \text{Request}_i$$



# Ví dụ: P1 yêu cầu (1, 0, 2)

- Kiểm tra Request  $1 \leq$  Need 1:  $(1, 0, 2) \leq (1, 1, 2) \Rightarrow$  Đúng
- Kiểm tra Request  $1 \leq$  Available:  $(1, 0, 2) \leq (3, 3, 2) \Rightarrow$  Đúng
- Giả định cấp phát tài nguyên đáp ứng yêu cầu của  $P_1$

	Allocation			Max			Available			Need		
	A	B	C	A	B	C	A	B	C	A	B	C
P0	0	1	0	7	5	3	3	3	2	7	4	3
P1	2	0	0	3	2	2				1	2	2
P2	3	0	2	9	0	2				6	0	0
P3	2	1	1	2	2	2				0	1	1
P4	0	0	2	4	3	3				4	3	1



# Ví dụ: P1 yêu cầu (1, 0, 2)

- Trạng thái mới

	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			

- 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  $P_1$ .



# Ví dụ: P4 yêu cầu (3, 3, 0)

- Kiểm tra Request  $4 \leq Available$ :

- $(3, 3, 0) \leq (3, 3, 2) \Rightarrow Đúng$

- Trạng thái mới

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

- Trạng thái mới là unsafe vậy không thể cấp phát tài nguyên cho P4.



## Các phương pháp giải quyết deadlock

### 6.3.3. Phát hiện deadlock

03.



### 6.3.3. Phát hiện deadlock

- Chấp nhận xảy ra deadlock trong hệ thống
- Giải thuật phát hiện deadlock
- Cơ chế phục hồi



### 6.3.3. Phát hiện deadlock

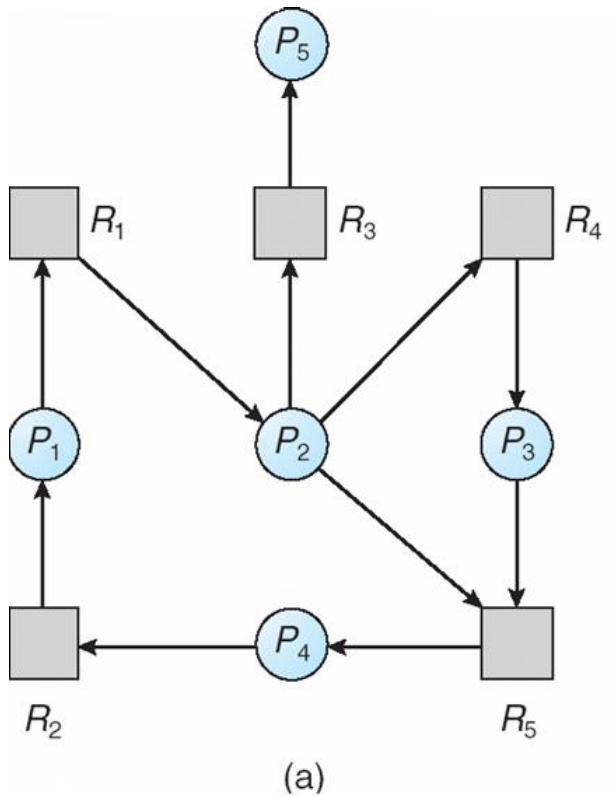
**Mỗi loại tài nguyên chỉ có một thực thể**

- Sử dụng wait-for graph
  - Các Node là các tiến trình
  - $P_i \rightarrow P_j$  nếu  $P_i$  chờ tài nguyên từ  $P_j$
- Mỗi giải thuật kiểm tra có tồn tại chu trình trong wait-for graph hay không sẽ được gọi định kỳ. Nếu có chu trình thì tồn tại deadlock.
- Giải thuật phát hiện chu trình có thời gian chạy là  $O(n^2)$ , với  $n$  là số đỉnh của graph.

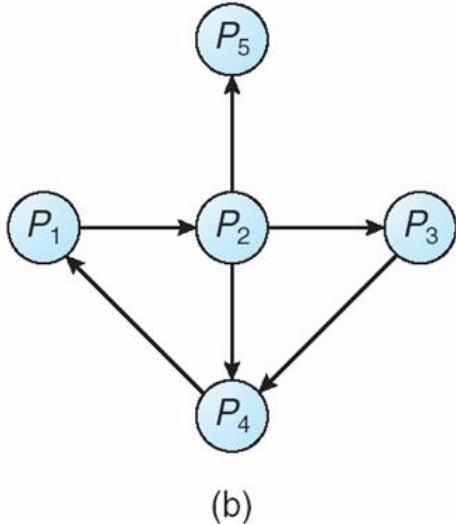


## 6.3.3. Phát hiện deadlock

### Đồ thị cấp phát tài nguyên và đồ thị wait-for



Resource-Allocation Graph



Corresponding wait-for graph



### 6.3.3. Phát hiện deadlock

Mỗi loại tài nguyên có nhiều thực thể

- Gọi n: số tiến trình; m: số loại tài nguyên.
- Available: vector độ dài m thể hiện số thực thể sẵn sàng của mỗi loại tài nguyên.
- Allocation: ma trận  $n \times m$  định nghĩa số thực thể của mỗi loại tài nguyên đã cấp phát cho mỗi tiến trình.
- Request: ma trận  $n \times m$  xác định yêu cầu hiện tại của mỗi tiến trình.
  - $\text{Request}[i,j] = k \Leftrightarrow P_i$  đang yêu cầu thêm k thực thể của  $R_j$ .



## 6.3.3.1. Giải thuật phát hiện deadlock

1. Gọi *Work* và *Finish* là vector kích thước  $m$  và  $n$ . Khởi tạo:
  - a. *Work* = Available
  - b. For  $i = 1, 2, \dots, n$ , nếu  $\text{Allocation}_i \neq 0$  thì  $\text{Finish}[i] := \text{false}$   
còn không thì  $\text{Finish}[i] := \text{true}$
2. Tìm  $i$  thỏa mãn:
  - a.  $\text{Finish}[i] = \text{false}$
  - b.  $\text{Request}_i \leq \text{Work}$

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



## 6.3.3.1. Giải thuật phát hiện deadlock

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

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

quay về bước 2.

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

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



## 6.3.3.1. Giải thuật phát hiện deadlock - Ví dụ

- 5 quá trình P0 ,..., P4 3 loại tài nguyên:
  - A (7 instance), B (2 instance), C (6 instance).
- Tại thời điểm T0

	Allocation			Request			Available		
	A	B	C	A	B	C	A	B	C
P0	0	1	0	0	0	0	0	0	0
P1	2	0	0	2	0	2			
P2	3	0	3	0	0	0			
P3	2	1	1	1	0	0			
P4	0	0	2	0	0	2			

Chuỗi  $\langle P_0, P_2, P_3, P_1, P_4 \rangle$  sẽ cho kết quả  $\text{Finish}[i] = \text{true}$ ,  $i = 1, \dots, n$



## 6.3.3.1. Giải thuật phát hiện deadlock - Ví dụ

- P2 yêu cầu thêm một instance của C. Ma trận Request như sau:

	Allocation			Request			Available		
	A	B	C	A	B	C	A	B	C
P0	0	1	0	0	0	0	0	0	0
P1	2	0	0	2	0	2			
P2	3	0	3	0	0	1			
P3	2	1	1	1	0	0			
P4	0	0	2	0	0	2			



## 6.3.3.2. Phục hồi deadlock

- Khi deadlock xảy ra, để phục hồi:
  - Báo người vận hành.
  - Hệ thống tự động phục hồi bằng cách bẻ gãy chu trình deadlock:
    - Chấm dứt một hay nhiều tiến trình.
    - Lấy lại tài nguyên từ một hay nhiều tiến trình.



## 6.3.3.2. Phục hồi deadlock

### Chấm dứt tiến trình bị deadlock

- Chấm dứt lần lượt từng tiến trình 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 chấm dứt?
  - Độ ưu tiên của tiến trình
  - Thời gian đã thực thi của tiến trình và thời gian còn lại
  - Loại tài nguyên mà tiến trình đã sử dụng
  - Tài nguyên mà tiến trình cần thêm để hoàn tất công việc
  - Số lượng tiến trình cần được chấm dứt
  - Tiến trình là interactive hay batch



## 6.3.3.2. Phục hồi deadlock

### Lấy lại tài nguyên

- Lấy lại tài nguyên từ một tiến trình, cấp phát cho tiến trình khác cho đến khi không còn deadlock nữa.
- **Chọn “nạn nhân”** để tối thiểu chi phí (có thể dựa trên số tài nguyên sở hữu, thời gian CPU đã tiêu tốn,...).



# Lấy lại tài nguyên

## Chấm dứt tiến trình bị deadlock

- **Trở lại trạng thái trước deadlock (Rollback):**
  - Rollback tiến trình bị lấy lại tài nguyên trở về trạng thái safe, tiếp tục tiến trình từ trạng thái đó.
  - Hệ thống cần lưu giữ một số thông tin về trạng thái các tiến trình đang thực thi.
- **Đói tài nguyên (Starvation):** để tránh starvation, phải bảo đảm không có tiến trình sẽ luôn luôn bị lấy lại tài nguyên mỗi khi deadlock xảy ra.



# Tóm tắt lại nội dung buổi học

- Vấn đề deadlock
- Mô hình hệ thống
- Các tính chất của deadlock
- Phương pháp giải quyết deadlock



## Bài tập 2

- Tìm Need?
- Hệ thống có an toàn không?
- Nếu P1 yêu cầu (0,4,2,0) thì có thể cấp phát cho nó ngay không?

	<i>Allocation</i>				<i>Max</i>				<i>Available</i>			
	<i>A</i>	<i>B</i>	<i>C</i>	<i>D</i>	<i>A</i>	<i>B</i>	<i>C</i>	<i>D</i>	<i>A</i>	<i>B</i>	<i>C</i>	<i>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				



## Bài tập 3

- Sử dụng thuật toán Banker xem các trạng thái sau có an toàn hay không? Nếu có thì đưa ra chuỗi thực thi an toàn, nếu không thì nêu rõ lý do không an toàn?

a. **Available** = (0,3,0,1)

b. **Available** = (1,0,0,2)

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



## Bài tập 4

- Trả lời các câu hỏi sau sử dụng giải thuật Banker:
  - Hệ thống có an toàn không? Đưa ra chuỗi an toàn nếu có?
  - Nếu  $P_1$  yêu cầu  $(1,1,0,0)$  thì có thể cấp phát cho nó ngay không?
  - Nếu  $P_4$  yêu cầu  $(0,0,2,0)$  thì có thể cấp phát cho nó ngay không?

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



# THẢO LUẬN

