

Chương 4

TẮC NGHẼN

(Deadlock)

Nội Dung Chương 4

- ❑ 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 Deadlock

- ❑ 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 lẫn nhau (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

Đồ thị cấp phát tài nguyên (Resource Allocation Graph – RAG)

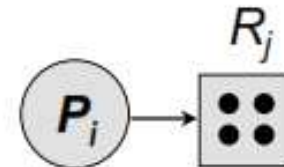
Ký hiệu

□ Process: 

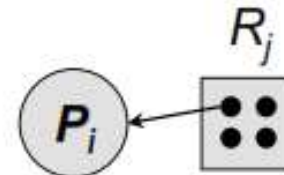
□ Loại tài nguyên với 4 thực thể:



□ P_i yêu cầu một thực thể của R_j :



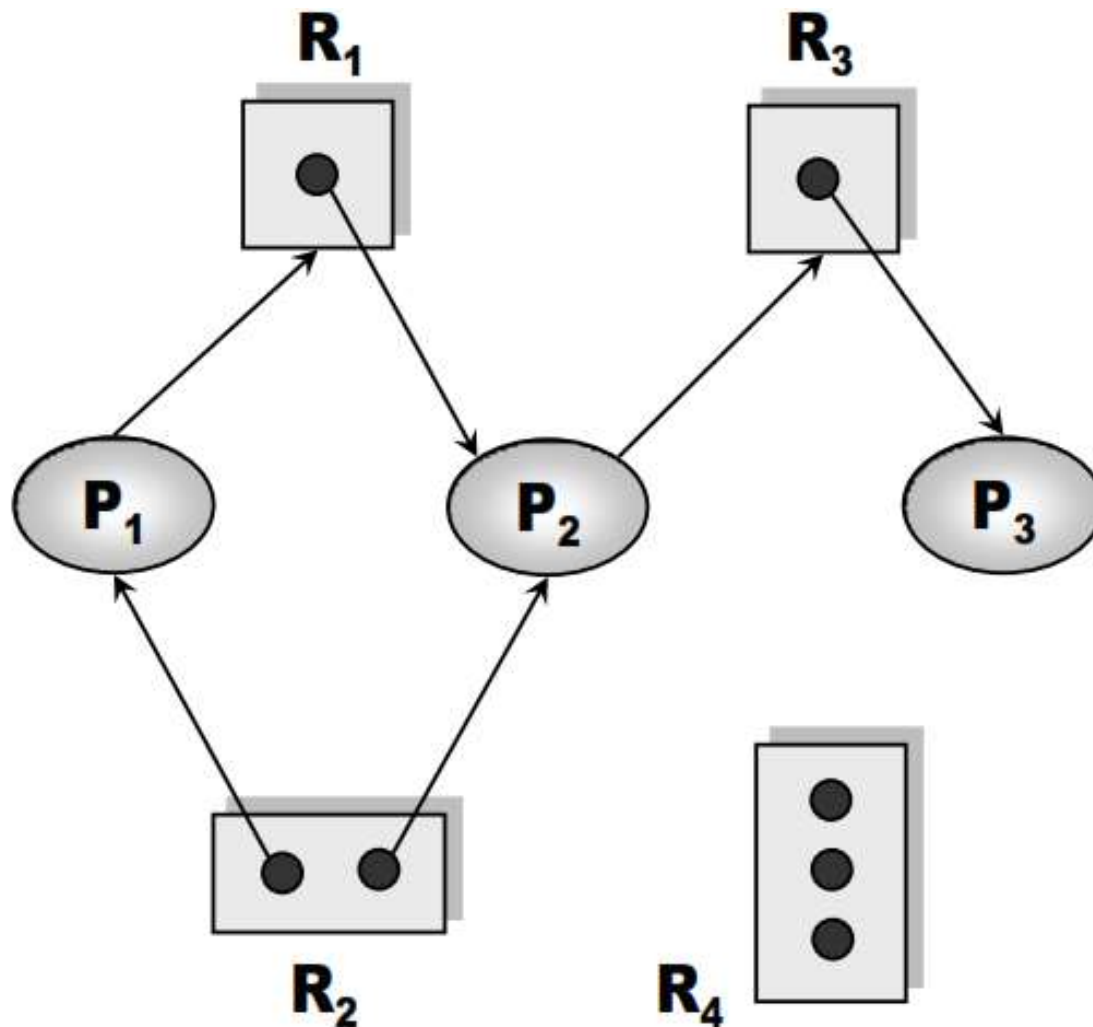
□ P_i đang giữ một thực thể của R_j :



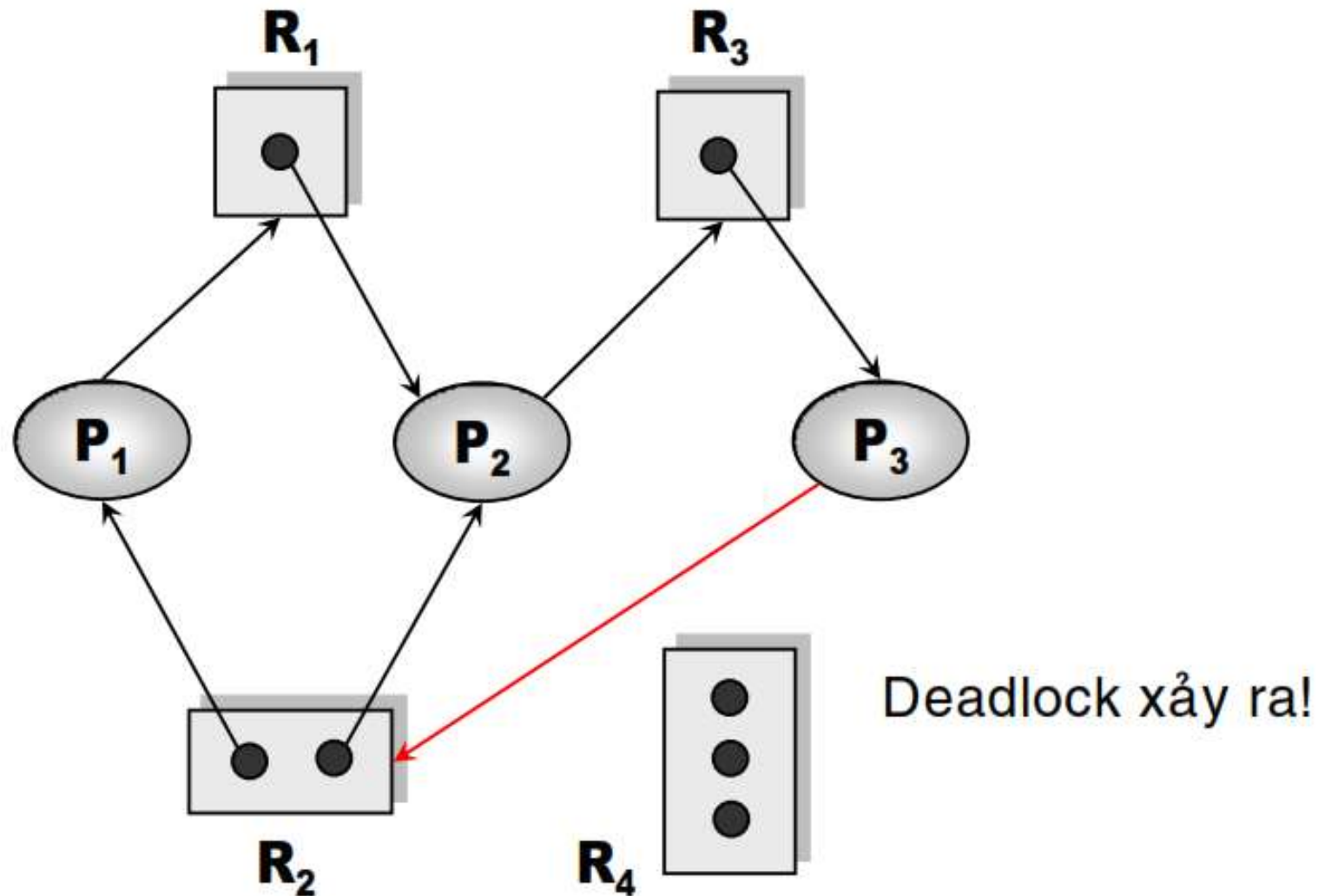
Đồ thị cấp phát tài nguyên (Resource Allocation Graph – RAG)

- *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: Đồ thị cấp phát tài nguyên
của 3 tiến trình P_1 , P_2 , P_3

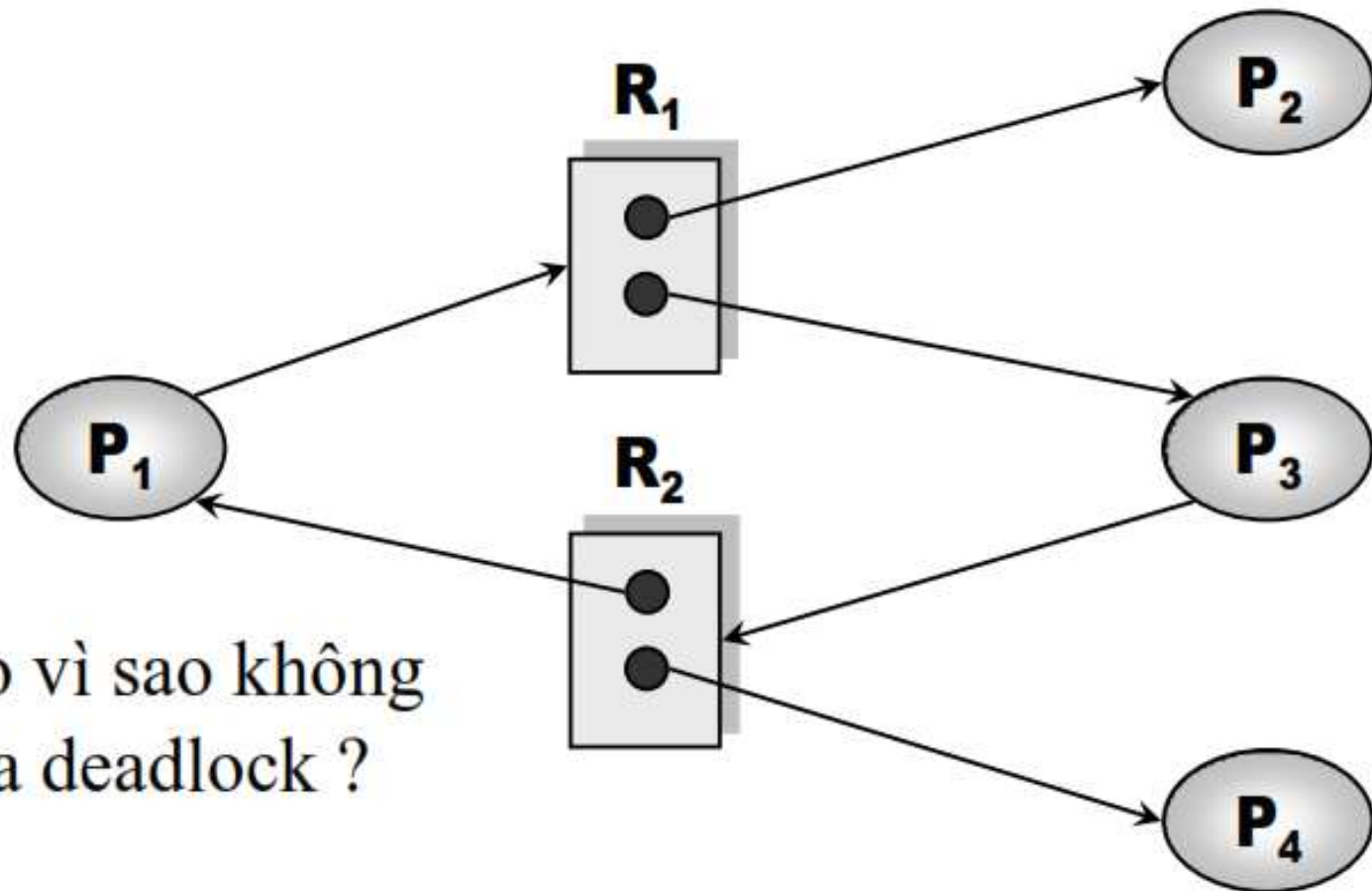


Ví dụ: Đồ thị cấp phát tài nguyên của 3 tiến trình P_1 , P_2 , P_3



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 .



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.

Ngăn chặn tắc nghẽn (Deadlock prevention)

Hệ điều hành thực hiện một số quy định để một trong bốn điều kiện tắc nghẽn không xảy ra.

- Cấm "Mutual Exclusion":
 - Đối với một số tài nguyên có thể dùng cơ chế spooling. Ví dụ: spooling máy in.
- Cấm "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..

Nhược điểm của Hold and Wait:

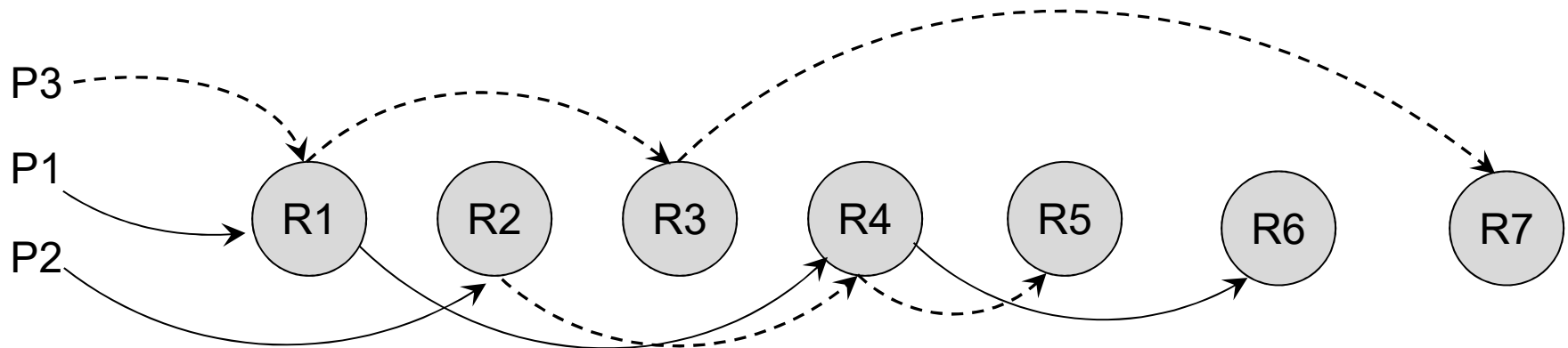
- Hiệu suất sử dụng tài nguyên (resource utilization) thấp
- Quá trình có thể bị starvation

Ngăn chặn tắc nghẽn (tt)

- Cấm **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 lấy đã bị 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 chặn tắc nghẽn (tt)

- **Cấm "Circular wait":**
 - Mỗi tài nguyên được đánh một số thứ tự.
 - Nếu tiến trình đang chiếm giữ tài nguyên có STT là a thì chỉ được phép yêu cầu thêm các tài nguyên có STT b với điều kiện $b > a$.



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

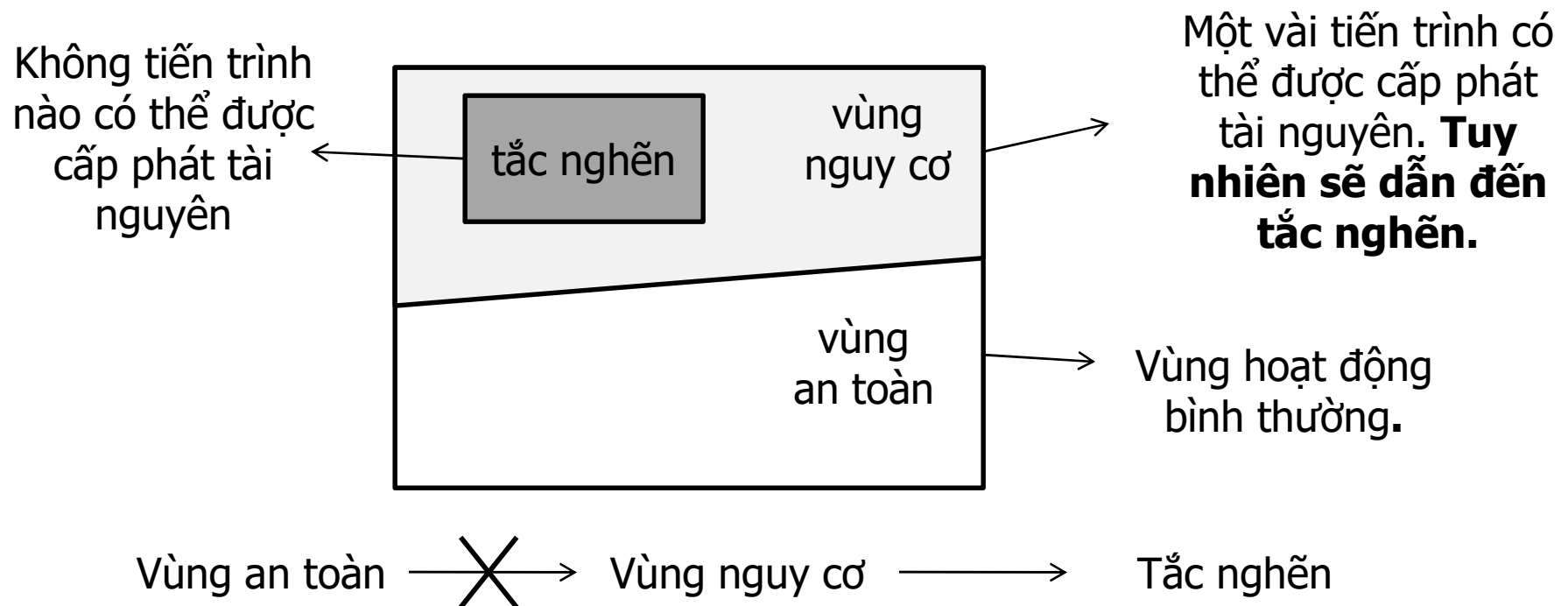
cần tối đa đang giữ

P_0	10	5
P_1	4	2
P_2	9	3

□ Hệ thống còn an toàn không?

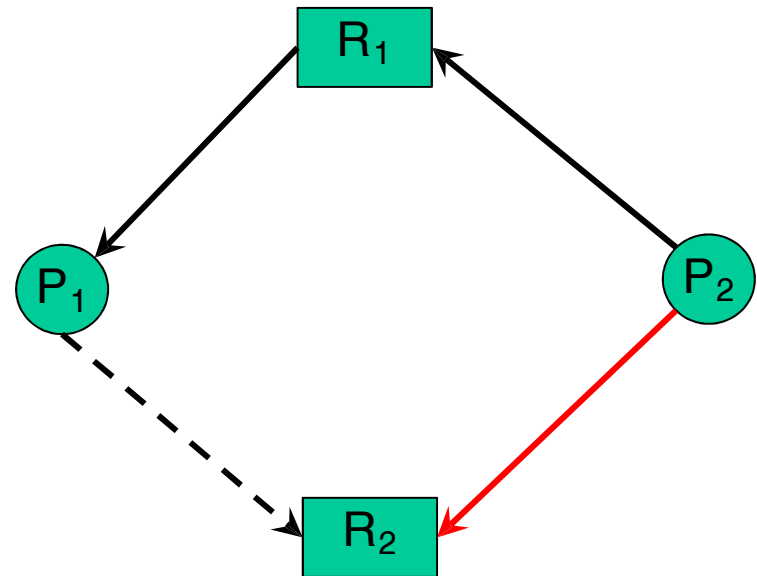
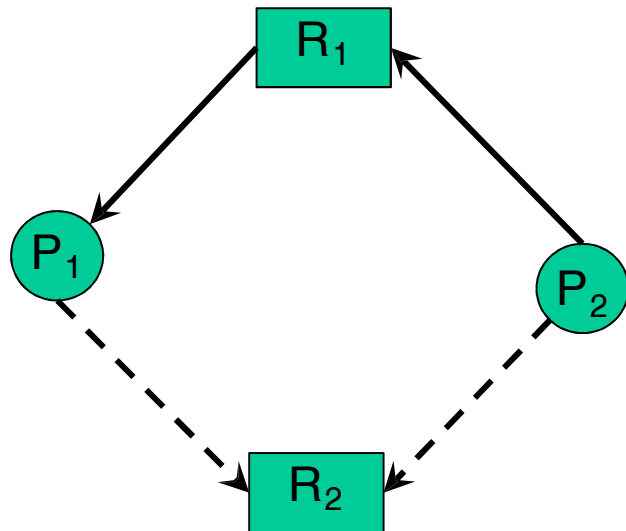
Trạng thái safe/unsafe và deadlock

- ❑ Nếu hệ thống đang ở trạng thái safe \Rightarrow không deadlock.
- ❑ Nếu hệ thống đang ở trạng thái unsafe ~~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

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

Max: ma trận $n \times m$

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

Allocation: ma trận $n \times m$

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

Need: ma trận $n \times m$

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

Nhận xét: $Need[i, j] = Max[i, j] - Allocation[i, j]$

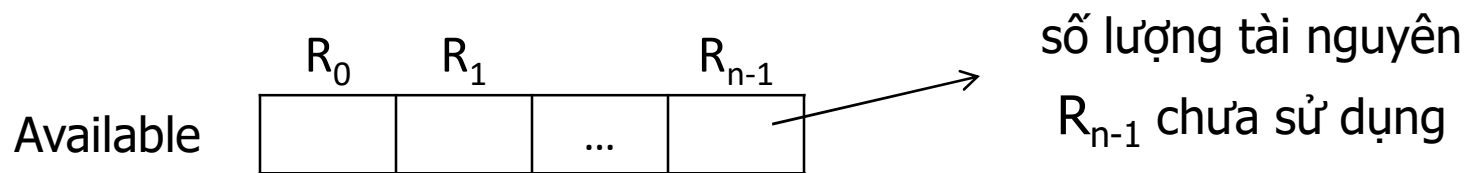
Ký hiệu $Y \leq X \Leftrightarrow Y[i] \leq X[i]$, ví dụ $(0, 3, 2, 1) \leq (1, 7, 3, 2)$

Giải thuật xác định vùng an toàn: Giải thuật Banker.

- Các dữ liệu: n : số process, m : số loại tài nguyên

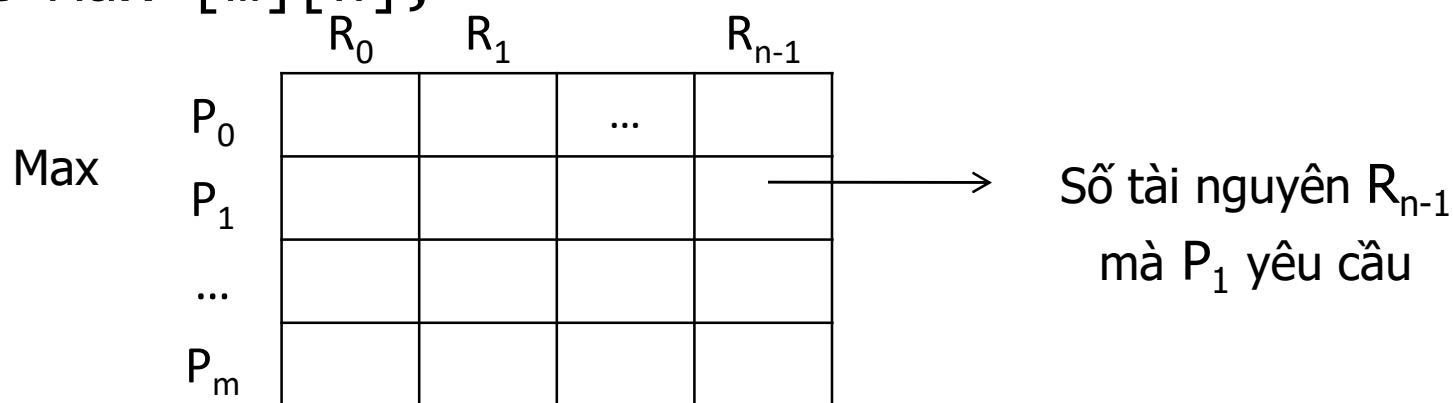
- Bảng tài nguyên chưa sử dụng:

`int Available[n];`

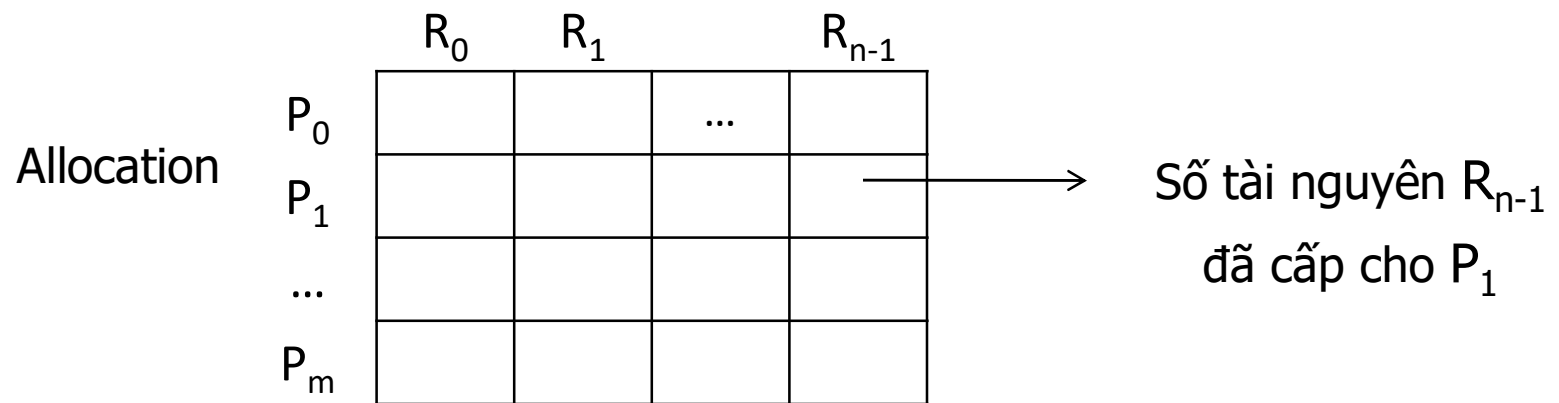


- Bảng tài nguyên đăng ký của từng tiến trình:

`int Max [m][n];`



- Bảng tài nguyên đã cấp phát cho từng tiến trình:
`int Allocation[m][n];`



- Bảng tài nguyên đã chưa cấp phát cho từng tiến trình:
`int Need[m][n];`
 $Need[i][j] = Max[i][j] - Allocation[i][j]$

Giải thuật Banker (tt): GT xác định chuỗi cấp phát 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 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): GT yêu cầu cấp phát tài nguyên

Gọi 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): GT yêu cầu cấp phát 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:

$Available := Available + Request_i$

$Allocation_i := Allocation_i - Request_i$

$Need_i := Need_i + Request_i$

- Ví dụ 1: Cho hệ thống có trạng thái như sau, xác định chuỗi cấp phát an toàn:

	Max			Allocation			Available		
	R0	R1	R2	R0	R1	R2	R0	R1	R2
P0	3	2	2	1	0	0	4	1	2
P1	6	1	3	2	1	1			
P2	3	1	5	2	1	1			
P3	7	2	2	0	0	2			

Tính bảng Need

	Need		
	R0	R1	R2
P0	2	2	2
P1	4	0	2
P2	1	0	4
P3	7	2	0

	Need			Allocation			Work		
	R0	R1	R2	R0	R1	R2	R0	R1	R2
P0	2	2	2	1	0	0	4	1	2
P1	4	0	2	2	1	1			
P2	1	0	4	2	1	1			
P3	7	2	0	0	0	2			

→ Chọn P1 để cấp phát và P1 hoàn tất

	Need			Allocation			Work		
	R0	R1	R2	R0	R1	R2	R0	R1	R2
P0	2	2	2	1	0	0	6	2	3
P1									
P2	1	0	4	2	1	1			
P3	7	2	0	0	0	2			

→ Chọn P0 để cấp phát và P0 hoàn tất

	Need			Allocation			Work		
	R0	R1	R2	R0	R1	R2	R0	R1	R2
P0							7	2	3
P1									
P2	1	0	4	2	1	1			
P3	7	2	0	0	0	2			

→ Chọn P3 để cấp phát và P3 hoàn tất

	Need			Allocation			Work		
	R0	R1	R2	R0	R1	R2	R0	R1	R2
P0							7	2	5
P1									
P2	1	0	4	2	1	1			
P3									

→ Chọn P2 để cấp phát và P2 hoàn tất

→ Chuỗi cấp phát: P1, P0, P3, P2. Hệ thống an toàn

Ví dụ 2: Yêu cầu cấp phát tài nguyên

Một hệ thống có 3 loại tài nguyên (A, B, C) và 5 tiến trình (P0, P1, P2, P3, P4) kèm theo các thông số được mô tả trong bảng sau.

	Allocation			Max			Available		
	A	B	C	A	B	C	A	B	C
P0	3	0	1	10	7	4	6	2	2
P1	3	2	1	8	5	3			
P2	2	1	3	6	3	4			
P3	0	3	0	9	6	3			
P4	1	1	2	7	4	5			

Tiến trình P1 yêu cầu tài nguyên là (2, 0, 1). Sử dụng giải thuật Banker, cho biết có thể thực hiện yêu cầu cấp phát tài nguyên này hay không?

Bài giải

Bước 1: Kiểm tra Request \leq Available

$$2\ 0\ 1 \leq 6\ 2\ 2 \quad \text{True}$$

Yêu cầu là hợp lệ.

Thử kiểm tra việc cấp phát có an toàn không

Bước 2: Work = Available - Request = 6 2 2 - 2 0 1 = 4 2 1

Cập nhật Allocation cho P1 = 3 2 1 + 2 0 1 = 5 2 2

Bước 3: Tính Need = Max - Allocation

$$P0: 10\ 7\ 4 - 3\ 0\ 1 = 7\ 7\ 3$$

$$P1: 8\ 5\ 3 - 5\ 2\ 2 = 3\ 3\ 1$$

$$P2: 6\ 3\ 4 - 2\ 1\ 3 = 4\ 2\ 1$$

$$P3: 9\ 6\ 3 - 0\ 3\ 0 = 9\ 3\ 3$$

$$P4: 7\ 4\ 5 - 1\ 1\ 2 = 6\ 3\ 3$$

Bước 4: Xác định Need (i) \leq Work

Với P0: 7 7 3 \leq 4 2 1 \rightarrow False

Với P1: 3 3 1 \leq 4 2 1 \rightarrow False

Với P2: 4 2 1 \leq 4 2 1 \rightarrow True

Bài giải (tt)

=> Thu hồi tài nguyên $Work = Work + Allocation(P2) = (4, 2, 1) + (2, 1, 3) = (6, 3, 4)$

=> Xét lại vòng lặp

Với P0: 7 7 3 <= 6 3 4-> False

Với P1: 3 3 1 <= 6 3 4-> True

=> Thu hồi tài nguyên $Work = Work + Allocation(P1) = (6, 3, 4) + (5, 2, 2) = (11, 5, 6)$

=> Xét lại vòng lặp

Với P0: 7 7 3 <= 11 5 6-> False

Với P3: 9 3 3 <= 11 5 6-> True

=> Thu hồi tài nguyên $Work = Work + Allocation(P3) = 11\ 5\ 6 + 0\ 3\ 0 = 11\ 8\ 6$

=> Xét lại vòng lặp

Với P0: 7 7 3 <= 11 8 6-> True

=> Thu hồi tài nguyên $Work = Work + Allocation(P0) = 11\ 8\ 6 + 3\ 0\ 1 = 14\ 8\ 7$

=> Xét lại vòng lặp

Với P4: 6 3 3 <= 14 8 7-> True

=> Thu hồi tài nguyên $Work = Work + Allocation(P4) = 14\ 8\ 7 + 1\ 1\ 2 = 15\ 9\ 9$

Tìm thấy chuỗi cấp phát an toàn {P2, P1, P3, P0, P4} nên có thể thực hiện cấp phát tài nguyên cho P1 được.

Ví dụ 3:

- 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

Ví dụ 3:

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	P1
P_1	2 0 0	1 2 2	↓	
P_2	3 0 2	6 0 0	5 3 2	P3
P_3	2 1 1	0 1 1	↓	
P_4	0 0 2	4 3 1	7 4 3	P4
			↓	
			7 4 5	P2
			↓	
			10 4 7	
			→ 10 5 7	

Ví dụ 3:

- 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 .

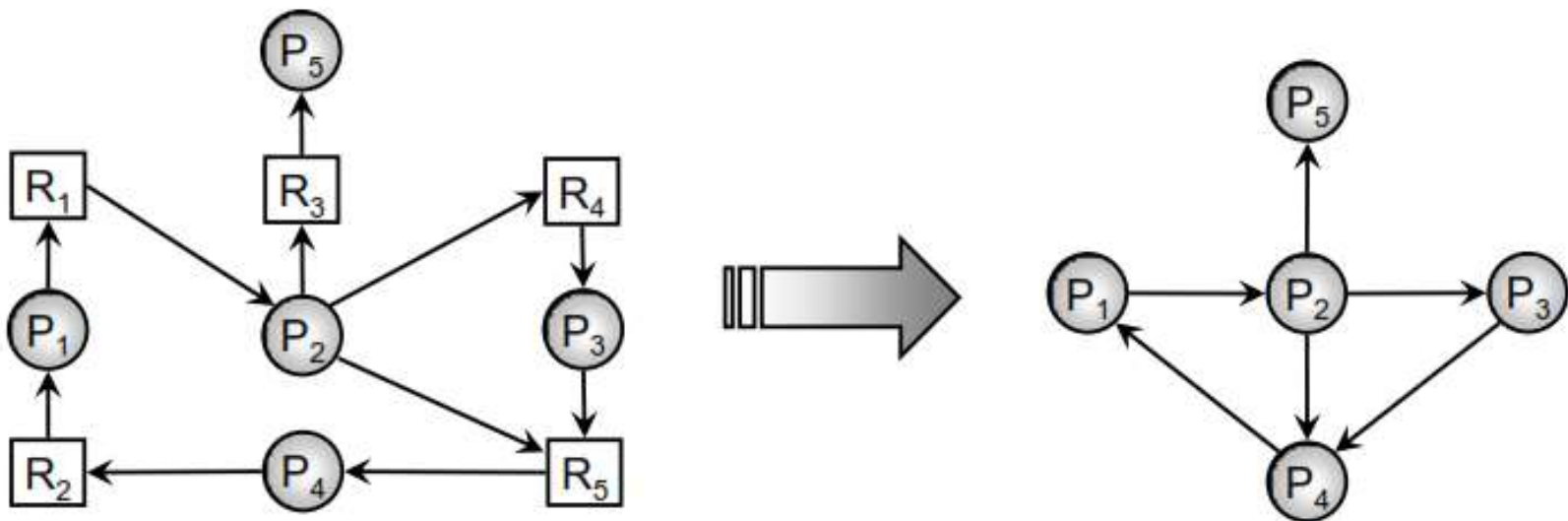
Phát hiện deadlock (Deadlock detection)

- ❑ 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 01 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 găng 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.

Q & A



Câu hỏi ôn tập

1. Nêu khái niệm tranh đoạt điều khiển và độc quyền truy xuất.
2. Miền găng tránh vấn đề tranh đoạt điều khiển như thế nào?
3. Nêu 3 yêu cầu của miền găng.
4. Giải pháp đồng bộ Busy waiting khác giải pháp Sleep & Wakeup ở điểm nào? Phương pháp nào tối ưu hơn?
5. Các hàm TestAndSet, up, down phải thực hiện trong điều kiện đặc biệt nào? Tại sao?

6. Trong giải pháp semaphore áp dụng cho miền găng, nếu khởi động giá trị value là 2 thay vì 1 thì điều gì xảy ra?
7. Ngoài vấn đề miền găng, semaphore có thể được sử dụng trong tình huống nào?
8. Nêu khái niệm tắc nghẽn.
9. Trạng thái an toàn là gì?

Bài tập

1. Xét giải pháp đồng bộ Peterson sửa đổi như sau:

```
while(TRUE)
```

```
{
```

```
    flag[0]=TRUE;
```

```
    turn= 0;
```

```
    while (turn==1 && flag[1] );
```

Critical section

```
    flag[0]= FALSE;
```

Non-critical section

```
}
```

```
while(TRUE)
```

```
{
```

```
    flag[1]=TRUE;
```

```
    turn= 1;
```

```
    while (turn==0 && flag[0] );
```

Critical section

```
    flag[1]= FALSE;
```

Non-critical section

```
}
```

Đây có phải là giải pháp đảm bảo độc quyền truy xuất không?

2. Một biến X được chia sẻ bởi hai tiến trình cùng thực hiện đoạn code sau:

```
do{  
    X = X+1;  
    if (X==20) X = 0;  
}while (TRUE);
```

Bắt đầu với giá trị $X=0$, chứng tỏ rằng X có thể vượt quá 20. Hãy sửa đoạn chương trình trên để X không vượt quá 20

3. Xét hai tiến trình xử lý đoạn chương trình sau:

```
process P1 {    A1();  A2();  }  
process P2 {    B1();  B2();  }
```

Đồng bộ hóa sự hoạt động của hai tiến trình này sao cho cả A_1 và B_1 đều hoàn tất trước khi A_2 hay B_2 bắt đầu

4. Tổng quát hóa câu 2 cho các tiến trình xử lý đoạn chương trình sau

```
process P1 { for(i=1; i<=100; i++) Ai(); }
```

```
process P2 { for(j=1; j<=100; j++) Bj(); }
```

Đồng bộ hóa hoạt động của hai tiến trình này sao cho với mọi $2 \leq k \leq 100$, A_k chỉ có thể bắt đầu khi B_{k-1} đã kết thúc và B_k chỉ có thể bắt đầu khi A_{k-1} kết thúc.

5. Xét hai tiến trình sau:

```
process A {  
    while(TRUE)  
        na = na+1;  
}
```

na và nb khởi tạo là 1

```
process B {  
    while (TRUE)  
        nb = nb+1;  
}
```

- a) Đồng bộ hóa hai tiến trình trên dùng semaphore sao cho tại bất cứ thời điểm nào đều có $na \leq nb + 10$
- b) Thực hiện câu a với điều kiện $nb \leq na \leq nb + 10$

6. Xét trạng thái hệ thống:

	Max			Allocation			Available		
	R0	R1	R2	R0	R1	R2	R0	R1	R2
P0	4	2	2	1	0	0	4	1	2
P1	6	1	3	2	1	1			
P2	3	2	5	3	1	1			
P3	7	2	2	0	0	2			

- Cho biết nội dung bảng Need.
- Hệ thống có ở trong trạng thái an toàn không?
- Nếu tiến trình P1 yêu cầu 4 cho R0, 1 cho R1, 1 cho R2, hãy xác định xem sau khi thực hiện yêu cầu này thì hệ thống có an toàn không?

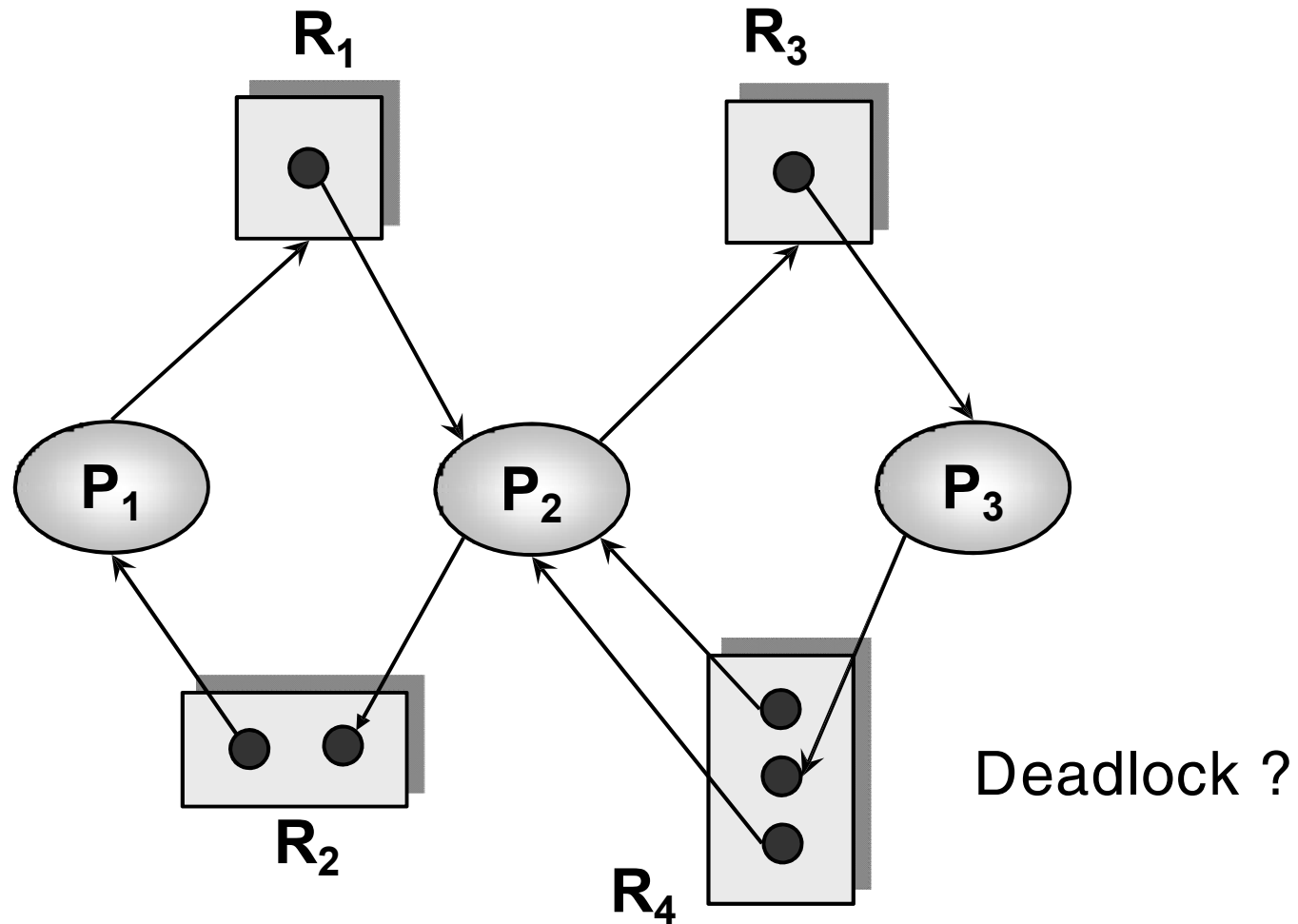
7. Xét trạng thái hệ thống:

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

- Cho biết nội dung bảng Need.
- Hệ thống có ở trong trạng thái an toàn không?
- Nếu P1 có yêu cầu tài nguyên (0, 4, 2, 0) thì yêu cầu này có được đáp ứng không?

□ Bài 8: Liệt kê 3 trường hợp xảy ra deadlock trong đời sống

□ Bài 9:



Giải thích ý nghĩa của đồ thị cấp phát tài nguyên trên.

Bài tập 10

□ Cho 1 hệ thống có 4 tiến trình, P1 đến P4, và 3 loại tài nguyên, R1 (3 thực thể), R2 (2 thực thể), R3 (2 thực thể). Tiến trình P1 giữ 1 R1, và yêu cầu 1 R2. Tiến trình 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.

Có nguy cơ deadlock không?

Nếu có nguy cơ deadlock, có chuỗi an toàn không, chuỗi nào?

Bài tập 11

- ❑ Xét hệ thống yêu cầu tài nguyên như sau: Tiến trình P1 yêu cầu sử dụng cả CPU và màn hình (display). Tiến trình P2 yêu cầu cả disk và màn hình. Tiến trình P3 yêu cầu cả disk và network. Tiến trình P4 yêu cầu cả network và màn hình. Tài nguyên được phân chia cho tiến trình theo thứ tự yêu cầu. Mỗi loại tài nguyên chỉ có 1 thực thể. Disk, màn hình và network là tài nguyên không thể lấy lại khi tiến trình sở hữu chưa kết thúc. Xác định deadlock có thể xảy ra trong trường hợp nào bằng cách sử dụng Resource Allocation Graph và Wait-For Graph.

Bài tập 12

- Xét trạng thái hệ thống với các loại tài nguyên A, B, C, và D như sau:

	Max				Allocation			
	A	B	C	D	A	B	C	D
P0	4	4	1	3	2	0	1	2
P1	1	6	5	0	1	0	4	0
P2	5	4	5	6	1	3	5	2
P3	0	6	5	2	0	6	3	2
P4	0	6	6	6	0	0	1	2

Available			
A	B	C	D
2	6	2	1

- Xác định nội dung bảng Need
- Hệ thống có ở trạng thái an toàn không?
- Nếu tiến trình P2 có yêu cầu thêm tài nguyên (4,0,0,4), yêu cầu này có được đáp ứng ngay lập tức hay không?

Bài tập 13

- ❑ Cho hệ thống 5 tiến trình và 3 loại tài nguyên (A, B, C). Giả sử hệ thống đang ở trạng thái như sau:

	Yêu cầu ban đầu			Đã cấp phát			Tài nguyên còn lại		
	A	B	C	A	B	C	A	B	C
P1	4	7	5	1	1	2	2	6	3
P2	6	9	5	2	2	1			
P3	3	2	2	1	0	0			
P4	3	6	4	0	1	1			
P5	2	3	2	0	0	1			

- ❑ Tính nhu cầu còn lại của mỗi tiến trình và số tài nguyên của mỗi loại hệ thống.
- ❑ Hãy tìm 1 trạng thái an toàn.
- ❑ Nếu tiến trình P2 có yêu cầu thêm tài nguyên (A: 0; B: 2; C: 1), hãy cho biết yêu cầu này có thể đáp ứng mà bảo đảm không xảy ra tình trạng deadlock hay không?

Bài tập 14

- ❑ Xét trạng thái hệ thống với các loại tài nguyên A, B, C, và D như sau:

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

Available			
A	B	C	D
2	6	2	2

- ❑ Xác định nội dung bảng Need
- ❑ Hệ thống có ở trạng thái an toàn không?
- ❑ Nếu tiến trình P2 có yêu cầu thêm tài nguyên (2,1,0,2), yêu cầu này có được đáp ứng ngay lập tức hay không?

Bài tập 15

- Cho 5 process $P_0 \dots P_4$; Hệ thống có 3 loại tài nguyên: A (10 instance), B (5 instance), và C (7 instance). Tại thời điểm T_0 :

	<u>Allocation</u>	<u>Max</u>	<u>Available</u>
	A B C	A B C	A B C
P_0	0 1 0	7 5 3	3 3 2
P_1	2 0 0	3 2 2	
P_2	3 0 2	9 0 2	
P_3	2 1 1	2 2 2	
P_4	0 0 2	4 3 3	

- Hệ thống có ở trạng thái an toàn không?
- Tại thời điểm T_1 ; P_0 có yêu cầu thêm tài nguyên (2,1,0), hệ thống sẽ như thế nào?

Bài tập 16

- Một hệ thống có 5 tiến trình với tình trạng tài nguyên như sau:

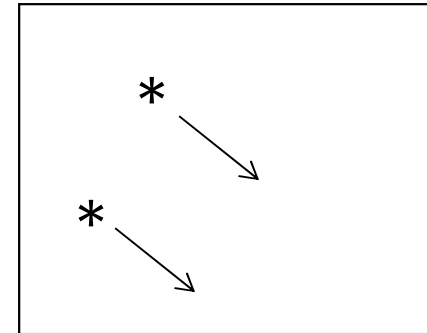
Process	Allocation				Max				Available			
	A	B	C	D	A	B	C	D	A	B	C	D
P ₀	0	0	1	2	0	0	1	2	1	5	2	0
P ₁	1	0	0	0	1	7	5	0				
P ₂	1	3	5	4	2	3	5	6				
P ₃	0	6	3	2	0	6	5	2				
P ₄	0	0	1	4	0	6	5	6				

- Dùng giải thuật banking, để trả lời các câu hỏi sau:
- Xác định nội dung bảng Need
 - Hệ thống có ở trạng thái an toàn không? (Nếu có hãy cho biết chuỗi an toàn)?
 - Nếu tiến trình P₁ có yêu cầu thêm tài nguyên (0, 4, 3, 0), thì yêu cầu này có được đáp ứng ngay lập tức hay không?

Bài tập thực hành

1. Viết chương trình đồng thời in ra 1000 chữ "World" và 1000 chữ "Earth". Trong đó từ "World" có màu đỏ và từ Earth có màu xanh. Quan sát thấy hiện tượng gì? Giải thích lý do.
2. Dùng mutex sửa lại bài 1 để khắc phục hiện tượng trên.

3. Viết chương trình cho phép 2 (và tổng quát là n) trái banh bay cùng lúc trong màn hình.



4. Áp dụng Semaphore, sửa chương trình trái banh bay trên, sao cho tại một thời điểm chỉ có tối đa 2 quả banh đi vào 1/3 phải màn hình.

