4. Deadlock

- Mô hình hệ thống
- Resource Allocation Graph (RAG)
- Phương pháp giải quyết deadlock
- Deadlock prevention
- Deadlock avoidance
- Deadlock detection
- Deadlock recovery

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
  - Giả sử hệ thống có một printer và một DVD drive. Quá trình P1 đang giữ DVD drive, quá trình P2 đang giữ printer.

    Bây giờ P1 yêu cầu printer và phải đợi, và P2 yêu cầu DVD drive và phải đợi.

Mô hình hóa hệ thống

- Hệ thống gồm các loại *tài nguyên*, kí hiệu R_1, R_2,…, R_m
  - Tài nguyên: CPU cycle, không gian bộ nhớ, thiết bị I/O, file,…

  Mỗi loại tài nguyên R_i có W_i *thực thể* (instance).

- Process sử dụng tài nguyên theo thứ tự
  - *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 và hoàn trả được gọi qua system call. Ví dụ
  - request/release device
  - open/close file
  - allocate/free memory

Điều kiện cần để xảy ra deadlock (1/2)

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

1. *Mutual exclusion:* một tài nguyên có thể được cấp phát cho nhiều lắm là 1 quá trình (tức là không chia sẻ được)
2. *Hold and wait:* một quá trình đang giữ một tài nguyên được phép yêu cầu thêm tài nguyên khác.

Điều kiện cần để xảy ra deadlock (2/2)

3. *No preemption:* (= no resource preemption) không lấy lại tài nguyên đã cấp phát cho quá trình, ngoại trừ khi quá trình tự hoàn trả nó.

4. *Circular wait:* tồn tại một tập $\{P_1,\ldots,P_n\}$ các quá trình đang đợi sao cho

   P_1 đợi một tài nguyên mà P_1 đang giữ

   P_2 đợi một tài nguyên mà P_2 đang giữ

   …

   P_n đợi một tài nguyên mà P_0 đang giữ

   Để ý: tài nguyên có thể gồm nhiều instance

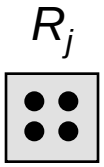
Resource Allocation Graph (1/2)

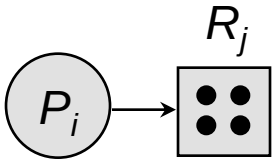
- *Resource allocation graph* (RAG) là đồ thị có hướng, với tập đỉnh *V* và tập cạnh *E*
  - Tập đỉnh *V* gồm 2 loại:
    - $P = \{P_1, P_2, \ldots, P_n\}$ (Tất cả process trong hệ thống)
    - $R = \{R_1, R_2, \ldots, R_m\}$ (Tất cả các loại tài nguyên trong hệ thống)
  - Tập cạnh *E* gồm 2 loại:
 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

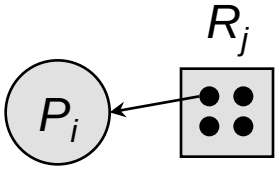
Resource Allocation Graph (2/2)

Ký hiệu

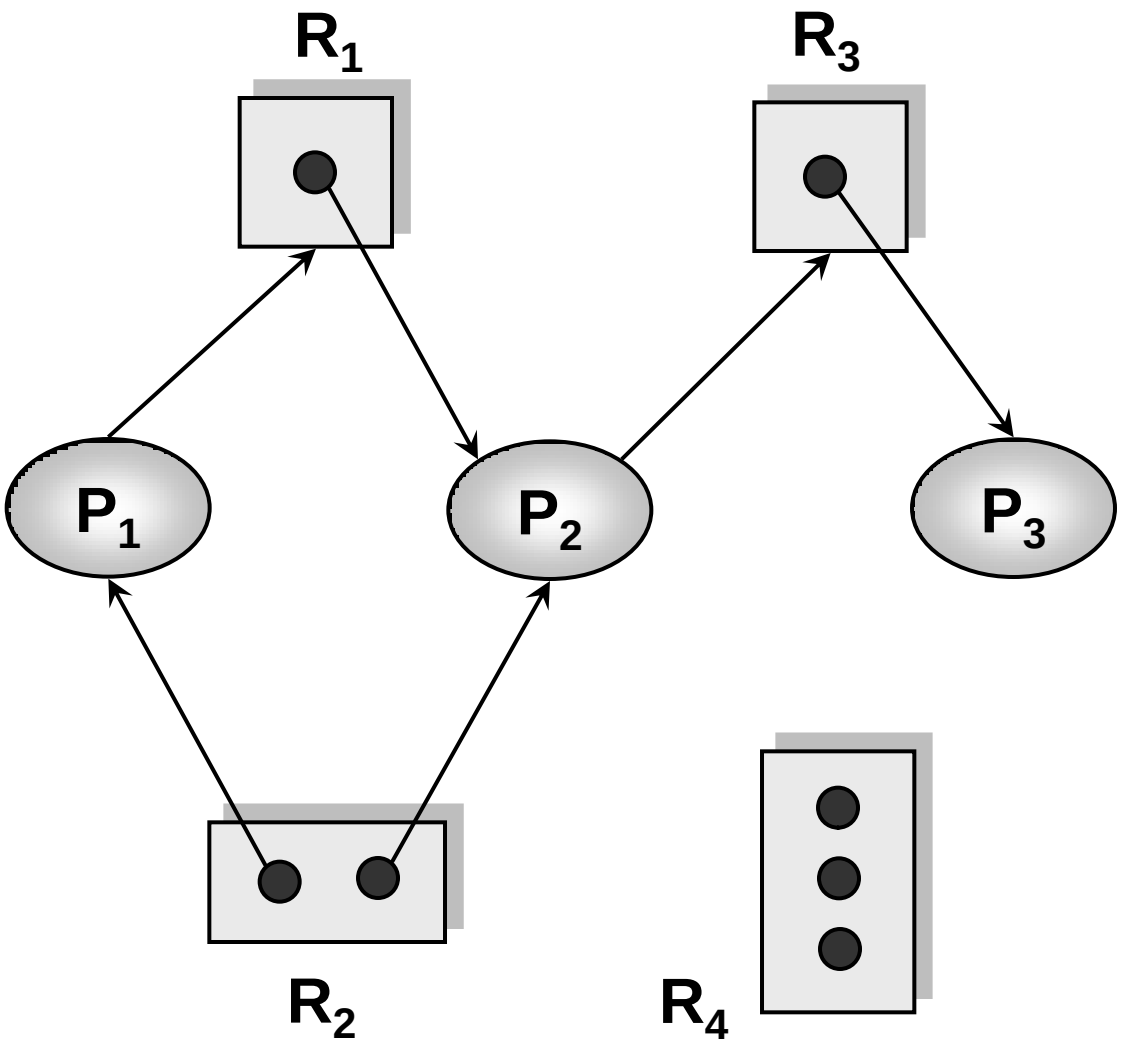
- Process: 

- Loại tài nguyên với 4 thực thể: 

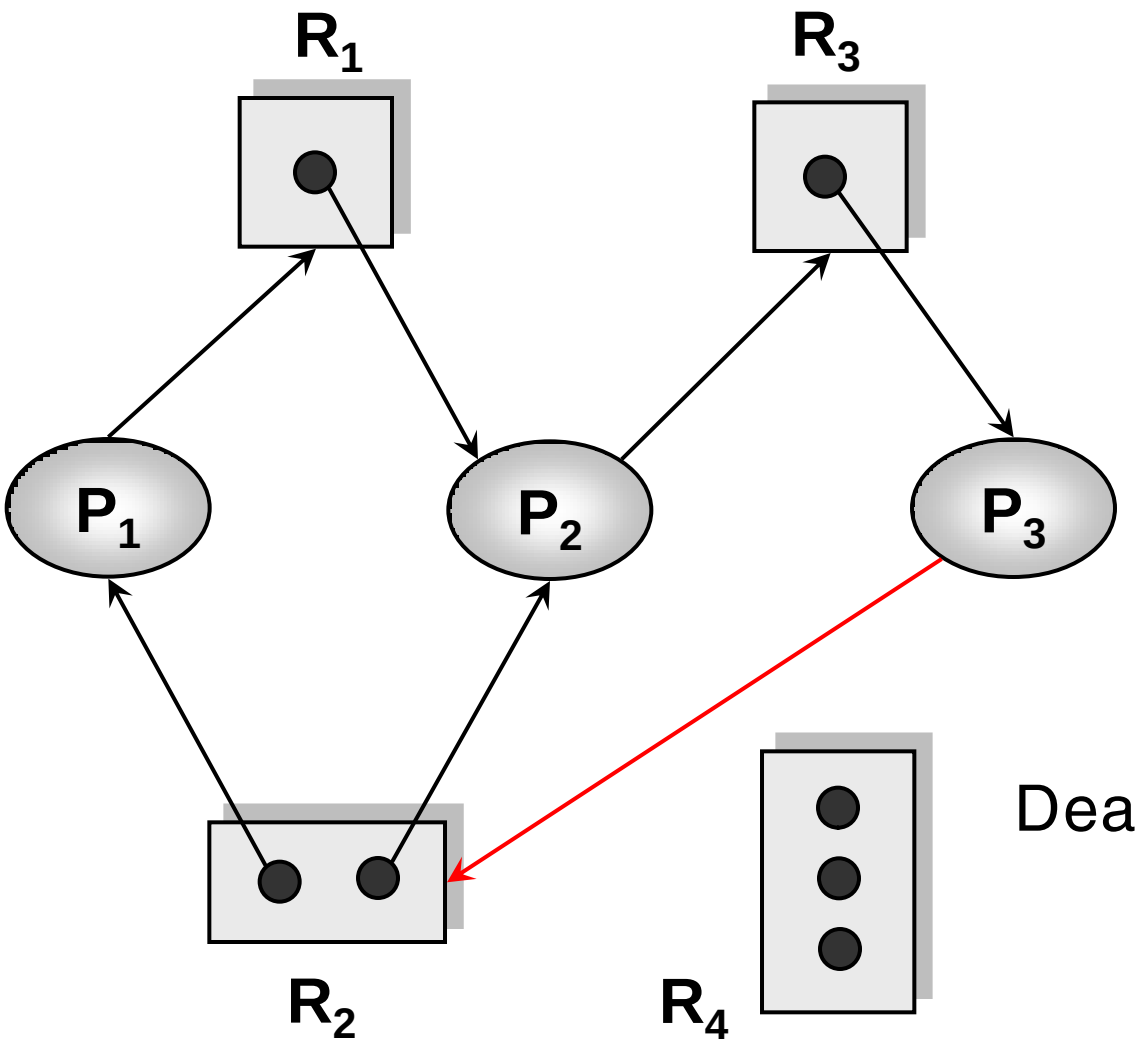
- P_i yêu cầu một thực thể của R_j : 

- P_i đang giữ một thực thể của R_j : 

Ví dụ về RAG (1/2)



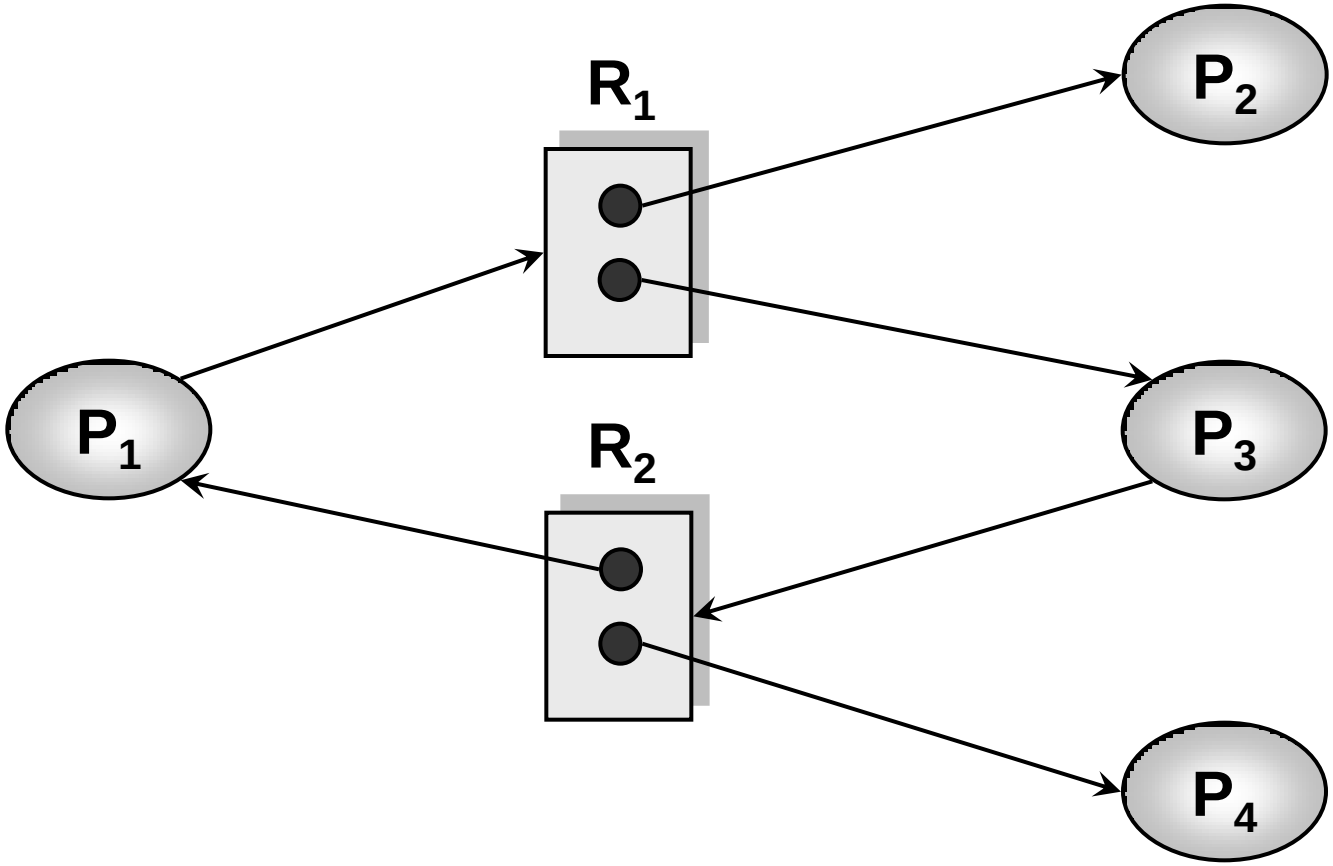
Ví dụ về RAG (2/2)



Deadlock xảy ra!

RAG và deadlock (1/2)

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



RAG và deadlock (2/2)

- RAG không chứa chu trình $\Rightarrow$ 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 (1/2)

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)

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.

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

Ngăn deadlock (1/4)

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 và tác vụ cho phép lên file chỉ là đọc): không cần thiết

Ngăn deadlock (2/4)

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 sẽ bị blocked.
   - Cách 2: khi yêu cầu tài nguyên, process không đang giữ bất kỳ tài nguyên nào. Nếu đang giữ 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 (3/4)

3. Ngăn No Preemption: cho phép lấy lại tài nguyên đã cấp phát cho quá trình $\Rightarrow$

   Chỉ thích hợp cho loại tài nguyên dễ dàng lưu và phục hồi như
   - CPU
   - Register
   - Vùng nhớ

   Không thích hợp cho loại tài nguyên như printer, DVD drive.

3. Ngăn No Preemption (tt):
   Một hiện thực: Nếu process A có giữ tài nguyên và yêu cầu tài nguyên khác nhưng tài nguyên này chưa cấp phát ngay được, thì hệ thống
   - lấy lại (preempt) mọi tài nguyên mà A đang giữ,
   - ghi nhận những tài nguyên của A đã bị lấy lại và tài nguyên mà A đã yêu cầu thêm,
   - tiếp tục A khi có đủ tài nguyên cho nó.

Ngăn deadlock (4/4)

4. Ngăn Circular Wait: tập các loại tài nguyên trong hệ thống được gán một thứ tự hoàn toàn.
   - Ví dụ: F(tape drive) = 1, F(disk drive) = 5, F(printer) = 12
     - F là hàm định nghĩa thứ tự trên tập các loại tài nguyên.

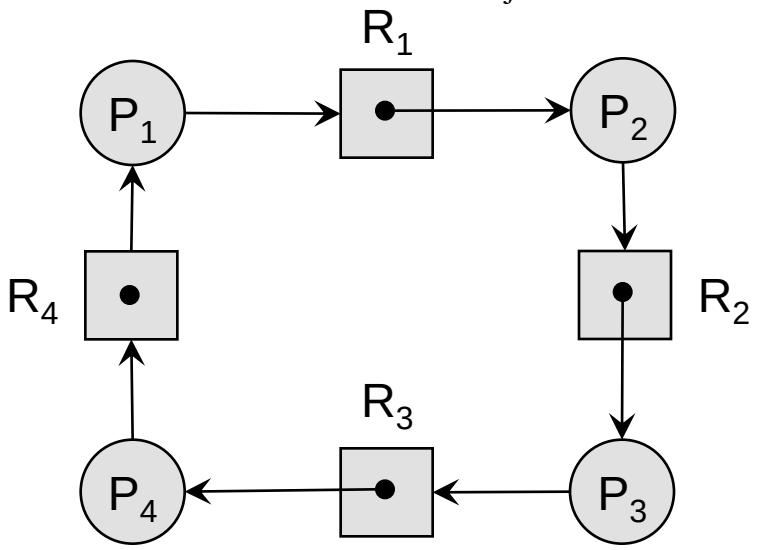
4. Ngăn Circular Wait (tt)

- Cách 1: mỗi process yêu cầu thực thể của 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 → disk drive → printer
  - Chuỗi yêu cầu thực thể không hợp lệ: disk drive → tape drive

- Mở rộng cách 1: 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": phản chứng
  - $F(R_4) < F(R_1)$
  - $F(R_1) < F(R_2)$
  - $F(R_2) < F(R_3)$
  - $F(R_3) < F(R_4)$

  Vậy $F(R_4) < F(R_4)$, mâu thuẩn!



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 nó cần.
- Giải thuật deadlock-avoidance sẽ điều khiển *trạng thái cấp phát tài nguyên* (resource-allocation state) sao cho hệ thống không rơi vào deadlock.

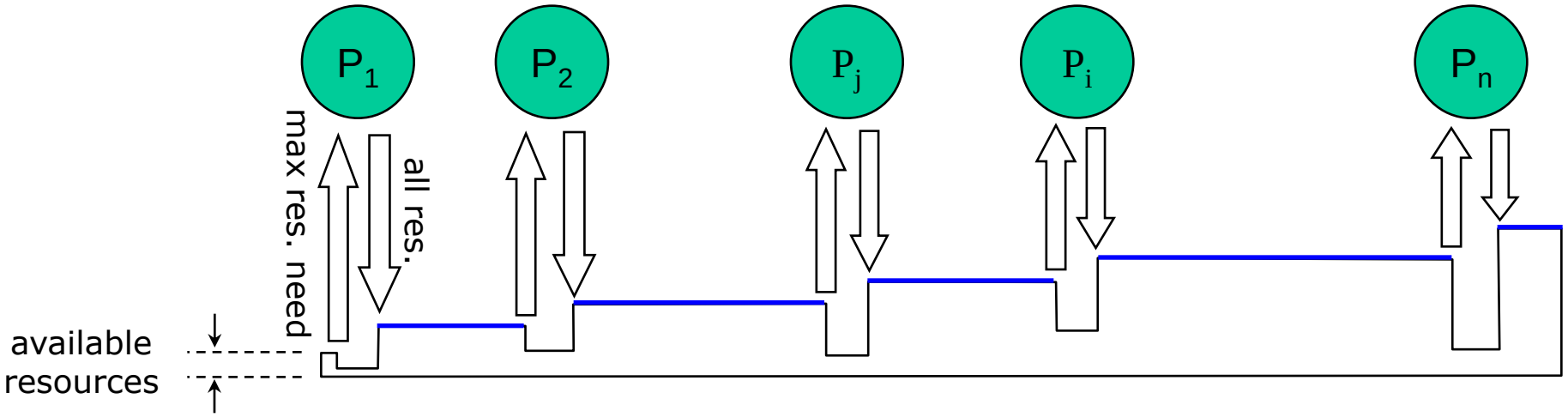
  Trạng thái cấp phát tài nguyên được định nghĩa dựa trên số tài nguyên còn lại, số tài nguyên đã được cấp phát và yêu cầu tối đa của mỗi process.

Trạng thái safe và unsafe

- Một trạng thái của hệ thống được gọi là *an toàn* (safe) nếu tồn tại một *chuỗi an toàn* (safe sequence).

Chuỗi an toàn (1/4)

- Một chuỗi quá trình $\langle P_1, P_2, \ldots, P_n \rangle$ là một *chuỗi an toàn* nếu
  - Với mọi $i = 1,\ldots,n$, 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.



Chuỗi an toàn (2/4)

- Một trạng thái của hệ thống được gọi là *không an toàn* (unsafe) nếu không tồn tại một chuỗi an toàn.

Chuỗi an toàn (3/4)

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

- Giả sử hệ thống còn 3 tape drive sẵn sàng và giả sử trạng thái hệ thống là

	cần tối đa	đang giữ
P_0	10	5
P_1	4	2
P_2	9	2

  - Chuỗi $\langle P_1, P_0, P_2 \rangle$ là chuỗi an toàn $\Rightarrow$ hệ thống là an toàn

Chuỗi an toàn (4/4)

- Giả sử hệ thống còn 2 tape drive sẵn sàng và giả sử trạng thái hệ thống là

	cần tối đa	đang giữ
P_0	10	5
P_1	4	2
P_2	9	3

  - Hệ thống là không an toàn

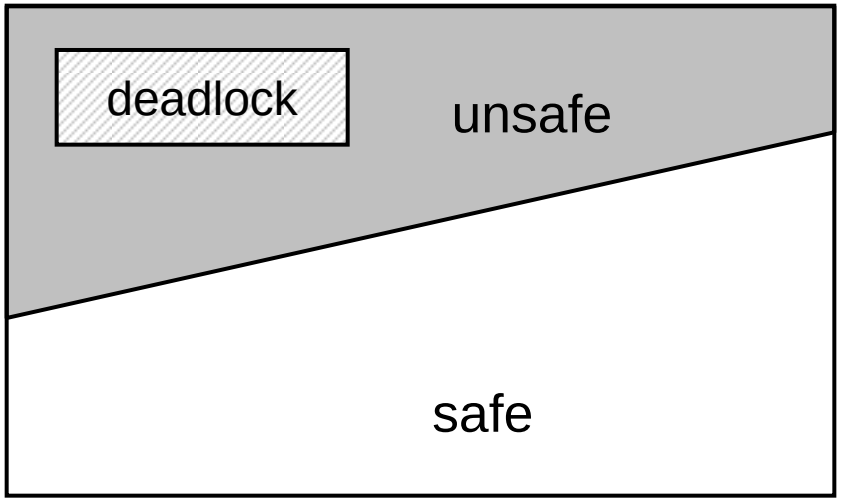
Trạng thái safe/unsafe và deadlock (1/2)

- Ý tưởng cho giải pháp tránh deadlock:

  Khi một process yêu cầu một tài nguyên đang sẵn sàng, hệ thống sẽ kiểm tra: nếu việc cấp phát này không dẫn đến tình trạng unsafe thì sẽ cấp phát ngay.

Trạng thái safe/unsafe và deadlock (2/2)

- 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 cấp phát tài nguyên sao cho hệ thống không đi vào vùng unsafe.



Giải thuật banker

- Áp dụng cho hệ thống cấp phát tài nguyên trong đó mỗi loại tài nguyên có thể có nhiều instance.
- 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 gồm
  - Giải thuật kiểm tra trạng thái an toàn
  - Giải thuật cấp phát tài nguyên
  - Giải thuật phát hiện deadlock

Giải thuật kiểm tra trạng thái an toàn (1/4)

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

Các cấu trúc dữ liệu

Available: vector độ dài m

Available[j] = k ⇔ loại tài nguyên R_j có k instance sẵn sàng

Max: ma trận n × m

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

Allocation: ma trận n × m

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

Need: ma trận n × m

Need[i, j] = k ⇔ P_i có thể yêu cầu 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 kiểm tra trạng thái an toàn (2/4)

Tìm một chuỗi an toàn

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

   Work $\leftarrow$ Available

   Finish[i] $\leftarrow$ false, i = 1,…, n

2. Tìm i thỏa

   (a) Finish[i] = false

   (b) $Need_i \leq$ Work (hàng thứ i của Need)

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

3. Work $\leftarrow$ Work + $Allocation_i$

   Finish[i] $\leftarrow$ true

   quay về bước 2.

4. Nếu Finish[i] = true, i = 1,…, n, thì hệ thống đang ở trạng thái safe

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

Giải thuật kiểm tra trạng thái an toàn (3/4)

- 5 process P_0 ,…, P_4
- 3 loại tài nguyên: A, gồm 10 instance; *B*, 5 instance; và *C*, 7 instance.
- Trạng thái cấp phát tài nguyên của 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	❺
P_1	2	0	0	3	2	2				1	2	2	❶
P_2	3	0	2	9	0	2				6	0	0	❹
P_3	2	1	1	2	2	2				0	1	1	❷
P_4	0	0	2	4	3	3				4	3	1	❸

Giải thuật kiểm tra trạng thái an toàn (4/4)

Dùng giải thuật kiểm tra trạng thái an toàn, tìm được một chuỗi an toàn là $\langle P_1, P_3, P_4, P_2, P_0 \rangle$:

	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

Giải thuật cấp phát tài nguyên (1/5)

Gọi $Request_i$ (độ dài m) là request vector của process P_i .

$Request_i [j] = k \Leftrightarrow P_i$ cần k instance của tài nguyên R_j .

1. Nếu $Request_i \leq Need_i$ thì đến bước 2. Nếu không, báo lỗi vì process đã vượt yêu cầu tối đa.

2. Nếu $Request_i \leq Available$ thì qua bước 3. Nếu không, P_i phải chờ vì tài nguyên không còn đủ để cấp phát.

Giải thuật cấp phát tài nguyên (2/5)

3. Giả định cấp phát tài nguyên đáp ứng yêu cầu của P_i bằng cách cập nhật trạng thái hệ thống như sau:

$$\begin{aligned} Available &\leftarrow Available - Request_i \\ Allocation_i &\leftarrow Allocation_i + Request_i \\ Need_i &\leftarrow Need_i - Request_i \end{aligned}$$

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:

$$\begin{aligned} Available &\leftarrow Available + Request_i \\ Allocation_i &\leftarrow Allocation_i - Request_i \\ Need_i &\leftarrow Need_i + Request_i \end{aligned}$$

Giải thuật cấp phát tài nguyên (3/5)

- (tiếp ví dụ) request$_1$(1, 0, 2) của P_1 có thỏa đượckhông ?
  - Kiểm tra điều kiện Request$_1$ ≤ Available:
    - (1, 0, 2) ≤ (3, 3, 2) là đúng

	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
P_1	2	0	0	3	2	2				1	2	2
P_2	3	0	2	9	0	2				6	0	0
P_3	2	1	1	2	2	2				0	1	1
P_4	0	0	2	4	3	3				4	3	1

Giải thuật cấp phát tài nguyên (4/5)

- (tiếp ví dụ) request$_1$(1, 0, 2) của P_1 có thỏa đượckhông?
  - Giả sử đáp ứng 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			

  - Trạng thái mới là **safe**, với 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.

Giải thuật cấp phát tài nguyên (4/4)

	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 yêu cầu (3, 3, 0) hoặc P_0 yêu cầu (0, 2, 0) thì theo giải thuật cấp phát tài nguyên có thỏa mãn được hay không?

Phát hiện deadlock

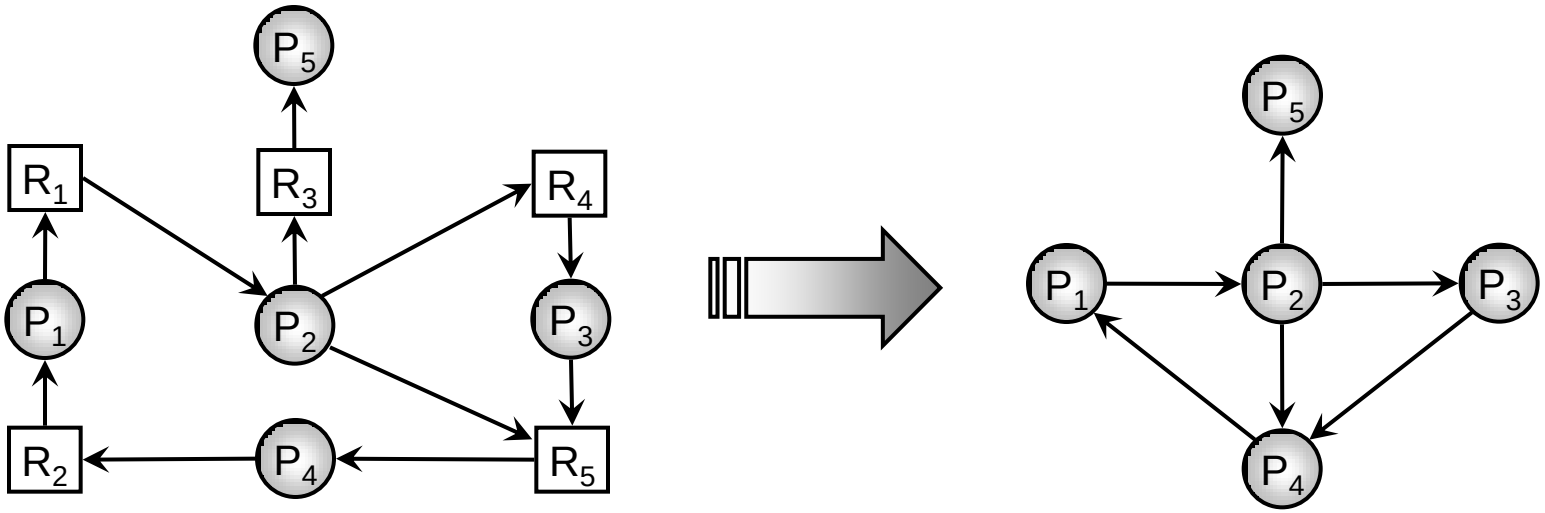
- Chấp nhận xảy ra deadlock trong hệ thống, kiểm tra trạng thái hệ thống bằng giải thuật phát hiện deadlock. Nếu có deadlock thì tiến hành phục hồi hệ thống

- Các giải thuật phát hiện deadlock thường sử dụng RAG

- Giải thuật phát hiện deadlock được thiết kế cho mỗi trường hợp sau
  1. Mỗi loại tài nguyên chỉ có một thực thể
  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



- Gọi định kỳ một giải thuật kiểm tra có tồn tại chu trình trong wait-for graph hay không. 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.
- Giả thiết: sau khi được đáp ứng yêu cầu tài nguyên, process sẽ hoàn tất và trả lại tất cả tài nguyên → giải thuật optimistic!
- Giải thuật phát hiện deadlock trường hợp mỗi loại tài nguyên có nhiều instance: các cấu trúc dữ liệu

  Available: vector độ dài m

  số instance sẵn sàng của mỗi loại tài nguyên

  Allocation: ma trận n × m

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

  Request: ma trận n × m

  yêu cầu hiện tại của mỗi process.

  Request [i, j] = k ⇔ P_i đang yêu cầu thêm k instance của R_j

Giải thuật phát hiện deadlock (1/4)

1. Các biến *Work* và *Finish* là vector kích thước *m* và *n*. Khởi tạo:

   Work ← Available

   i = 1, 2,…, 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 tồn tại i với Finish[i] = false, thì hệ thống đang ở trạng thái deadlock. Hơn thế nữa, nếu Finish[i] = false thì P_i bị deadlocked.

Giải thuật phát hiện deadlock (2/4)

- Nhận xét:
  - Khi giải thuật phát hiện deadlock không thấy hệ thống đang deadlock, chưa chắc trong tương lai hệ thống vẫn không deadlock.

Giải thuật phát hiện deadlock (3/4)

- Hệ thống có 5 quá trình P_0 ,…, P_4
  3 loại tài nguyên: *A*, gồm 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 Finish[i] = true cho mọi i, vậy hệ thống hiện không bị deadlocked.

Giải thuật phát hiện deadlock (4/4)

- Nhưng nếu thêm vào đó P_2 yêu cầu một instance của *C*, nghĩa là *có* ma trận Request:

	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

  - Hệ thống có đang bị deadlocked?
    - Có thể thu hồi tài nguyên đang giữ bởi process P_0 nhưng vẫn không đủ đáp ứng yêu cầu của các process khác.

      Vậy tồn tại deadlock, gây bởi các process P_1, P_2, P_3, và P_4.

Phục hồi khỏi deadlock

- Các giải pháp khi phát hiện deadlock
  - báo người vận hành (operator), người này sẽ xử lý tiếp

  hoặc
  - hệ thống tự động phục hồi bằng cách phá deadlock:
    - Giải pháp chấm dứt quá trình
    - Giải pháp lấy lại tài nguyên
    - Giải pháp Rollback

Phục hồi khỏi deadlock: Chấm dứt quá trình

- Phục hồi hệ thống khỏi deadlock bằng cách
  - 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

Phục hồi khỏi deadlock: Lấy lại tài nguyên

- Các bước
  - Tạm dừng process *P* cần lấy lại tài nguyên, lấy lại tài nguyên từ *P*, cấp phát chúng cho process khác.
  - Khi tài nguyên được trả lại, cấp phát chúng lại cho *P*, rồi tiếp tục *P*.
- Giải pháp thường khó hoặc không thể thực hiện được.

Phục hồi khỏi deadlock: Rollback

- Xác định một process *P* đang giữ tài nguyên mà một process *Q* đang deadlocked cần
- Rollback process *P* về một thời điểm mà nó chưa có tài nguyên này.
- Cấp phát tài nguyên này cho quá trình *Q*