Deadlocks

Modelo do problema

Caracterização de deadlocks

Métodos para manipulação de deadlock

Prevenção de deadlock

Detecção de deadlock

Recuperação de deadlock

O problema de deadlock

- Um conjunto de processos bloqueados, cada qual mantendo um recurso alocado e esperando para alocar um recurso alocado para outro processo do conjunto.
- Exemplo:
  - Sistema tem duas fitas magnéticas.
  - P_1 e P_2 alocam, cada um, uma fita e necessitam da outra.

  - Exemplo
  - Semáforos *A* e *B*, inicializados em 1

P_0	P_1
wait (A);	*wait(B)*
wait (B);	*wait(A)*

Modelo do problema

- Tipos de recursos $R_1, R_2, \ldots, R_m$

  Ciclos de CPU, espaço de memória, dispositivos de E/S

- Cada recurso do tipo R_i tem W_i instâncias.
- Cada processo utiliza um recurso da seguinte forma:
  - requisita
  - usa
  - libera

Caracterização de deadlock

Para que exista um deadlock o sistema terá que apresentar as quatro condições:

- **Exclusão mútua:** somente um processo pode usar, por vez, um recurso.
- **Alocação e espera:** um processo alocando ao menos um recurso está esperando para conseguir alocar recursos adicionais alocados a outros processos. a
- **Sem preempção:** um recurso pode ser liberado somente por vontade do processo que o está alocando, depois que o processo completou sua tarefa.
- **Espera circular:** Se existe um conjunto $\{P_0, P_1, \ldots, P_n\}$ de processos de tal modo que P_0 está esperando para um recurso alocado por P_1, P_1 está esperando por um recurso alocado para P_2, …, P_{n-1} está esperando por um recurso alocado por P_n, e P_n está esperando por um recurso alocado por P_0.

Grafo de alocação de recursos

Um conjunto de vértives *V* e um conjunto de arestas *E*:

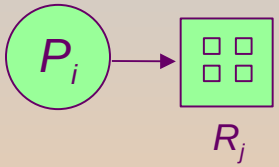
- V está particionado em dois tipos:
- :
  - $P = \{P_1, P_2, \ldots, P_n\}$, o conjunto de todos os processos do sistema.
  - $R = \{R_1, R_2, \ldots, R_m\}$, o conjunto de todos os tipos de recursos do sistema.
- Aresta de requisição – $P_1 \rightarrow R_j$
- Aresta de alocação – $R_j \rightarrow P_i$

Grafo de alocação de recursos (Cont.)

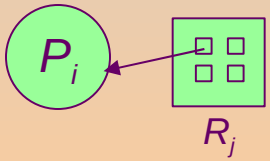
- Processo

- Tipo de recurso com 4 instâncias

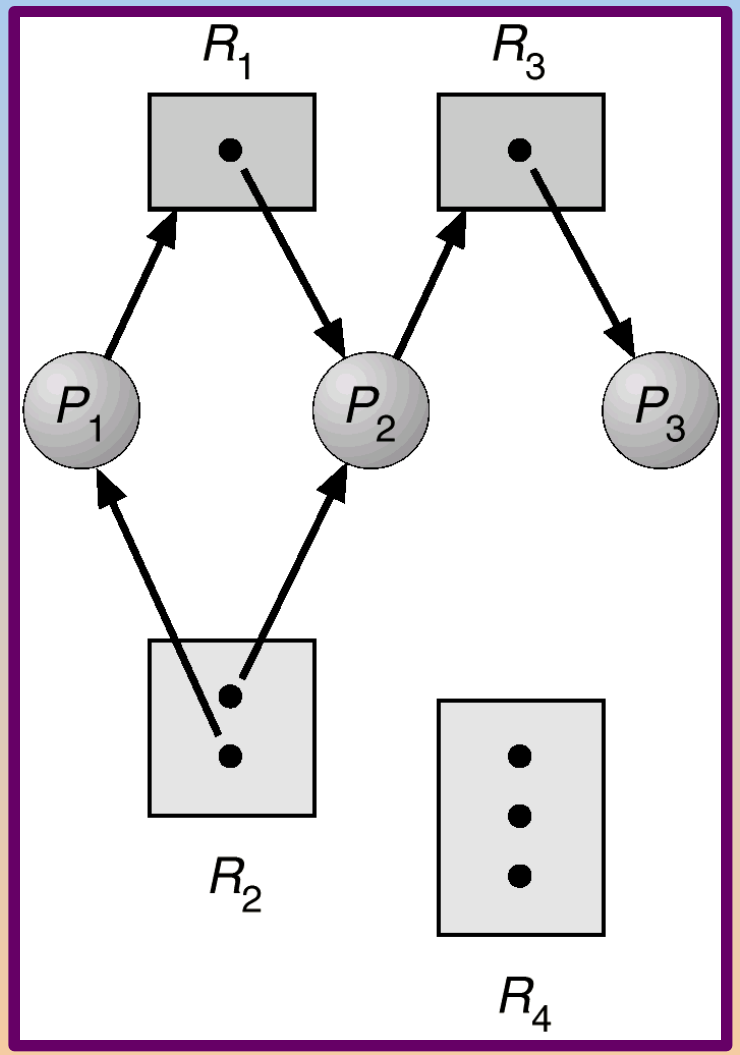
- P_i requisita uma instância de R_j



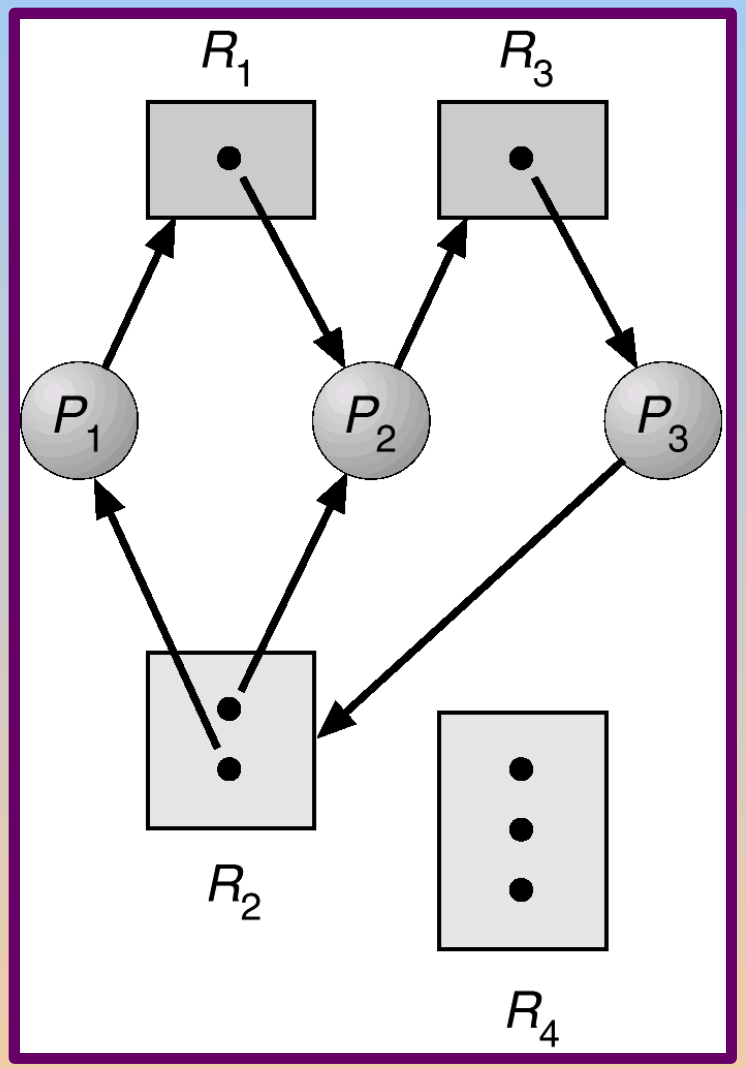
- P_i está alocando uma instância de R_j



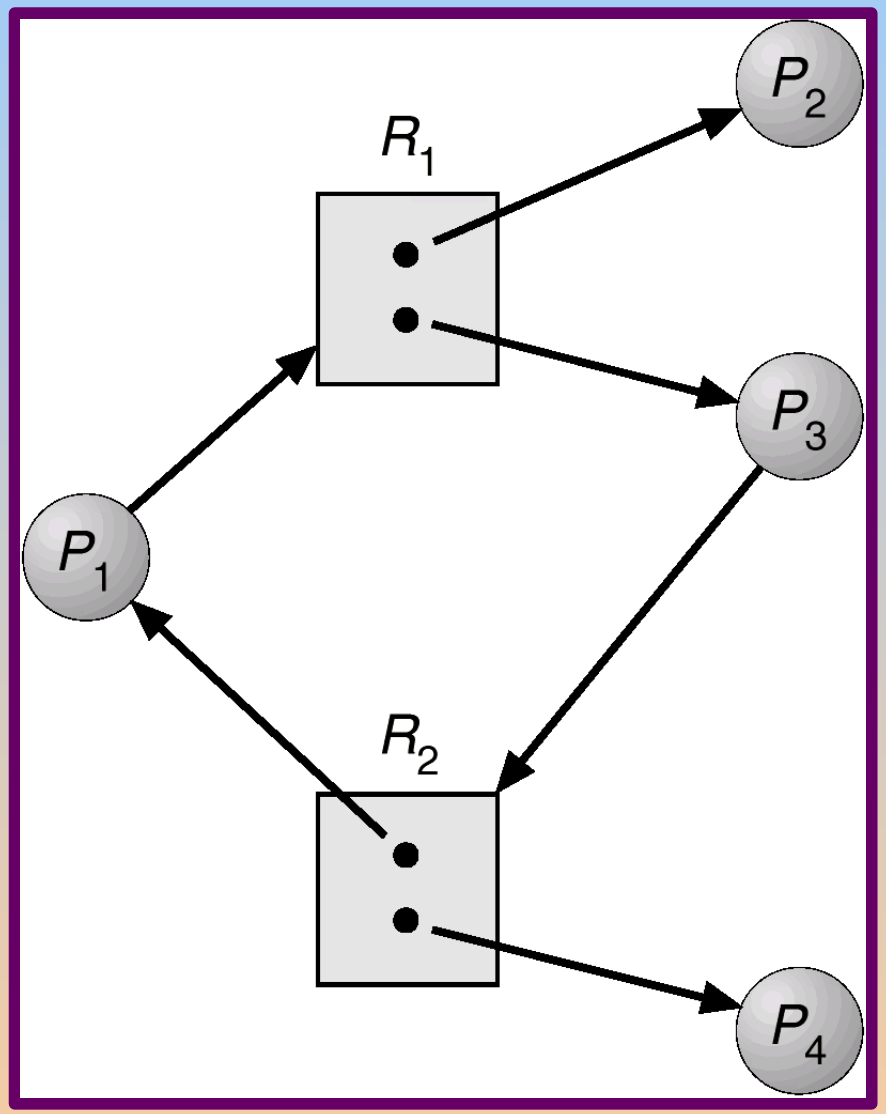
Exemplo de um grafo de alocação de recursos



Grafo de alocação de recursos com deadlock



Grafo de alocação de recursos com um ciclo, mas sem deadlock



Fatos básicos

- Se o grafo não contém ciclos $\Rightarrow$ sem deadlock.
- Se o grafo contém um ciclo $\Rightarrow$
  - Se uma instância por tipo de recurso, então existe deadlock.
  - Se várias instâncias por tipo de recursos, possibilidade de deadlock.

Métodos para manipulação de deadlocks

- Garanta que o sistema nunca entrará num estado de deadlock.
- Permita que o sistema entre num estado de deadlock e então realize um esquema de recuperação.
- Ignore o problema e assuma que deadlocks nunca ocorrem no sistema; usado pela maioria dos sistemas operacionais, incluindo UNIX.

Prevenção de deadlock

- Quando um processo requisita um recurso disponível, o sistema precisa decidir se alocação imediata deixa o sistema num estado seguro.

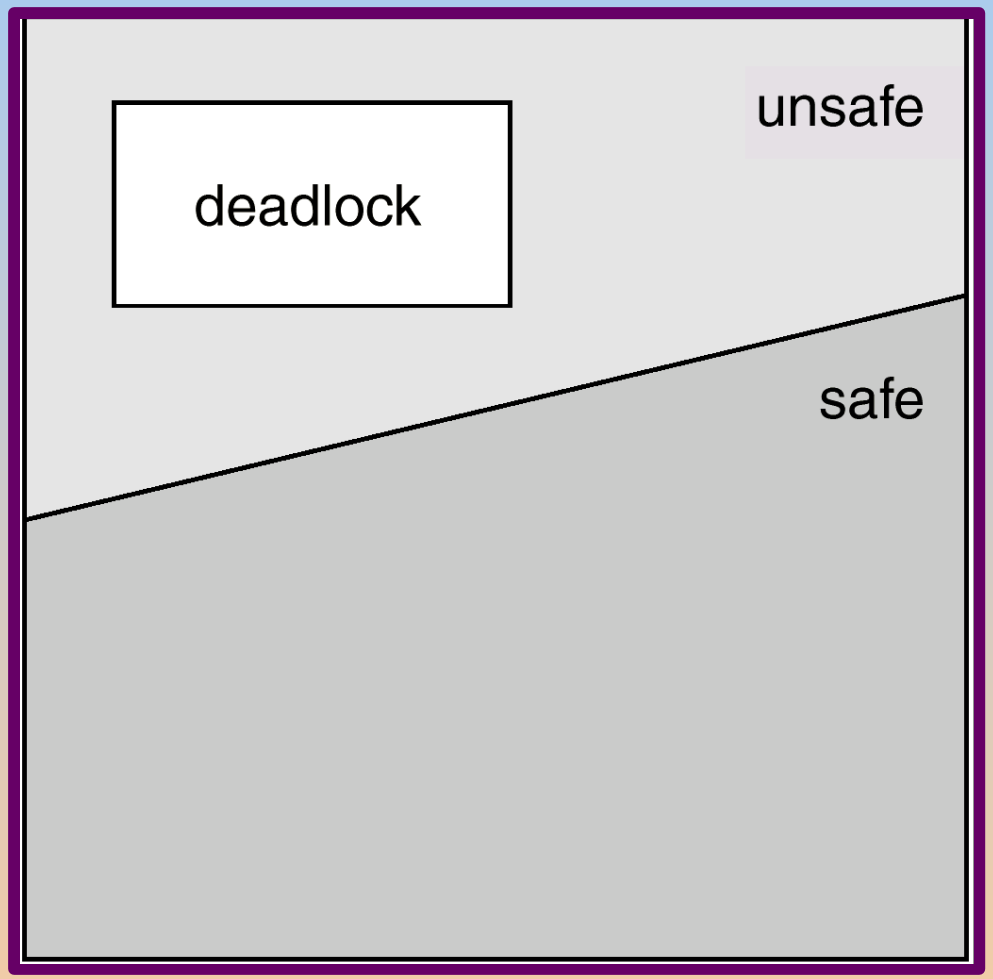
- Sistema está num estado seguro se existe uma sequência segura de todos os processos.

- Seqüência <P_1, P_2, …, P_n> está segura se para cada P_i, os recursos que P_i pode ainda requisitar podem ser disponibilizados pelos recursos atualmente disponíveis mais os recursos alocados por todos P_j, com $j<i$.

Fatos básicos - Prevenção

- Se o sistema está em estado seguro $\Rightarrow$ sem dedalocks.
- Se o sistema está em estado inseguro $\Rightarrow$ possibilidade de deadlock.
- Prevenção $\Rightarrow$ garantir que o sistema nunca entre num estado inseguro.

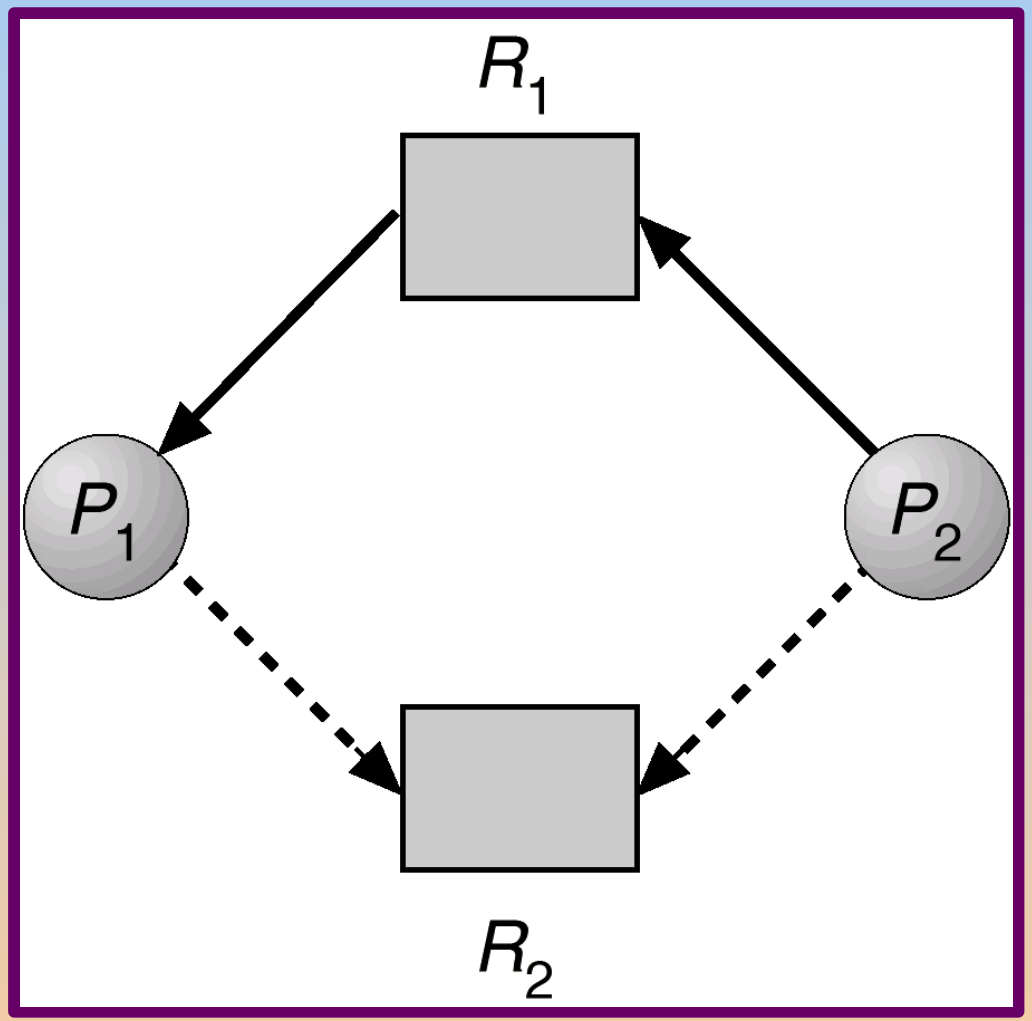
Estados seguro, inseguro e deadlock



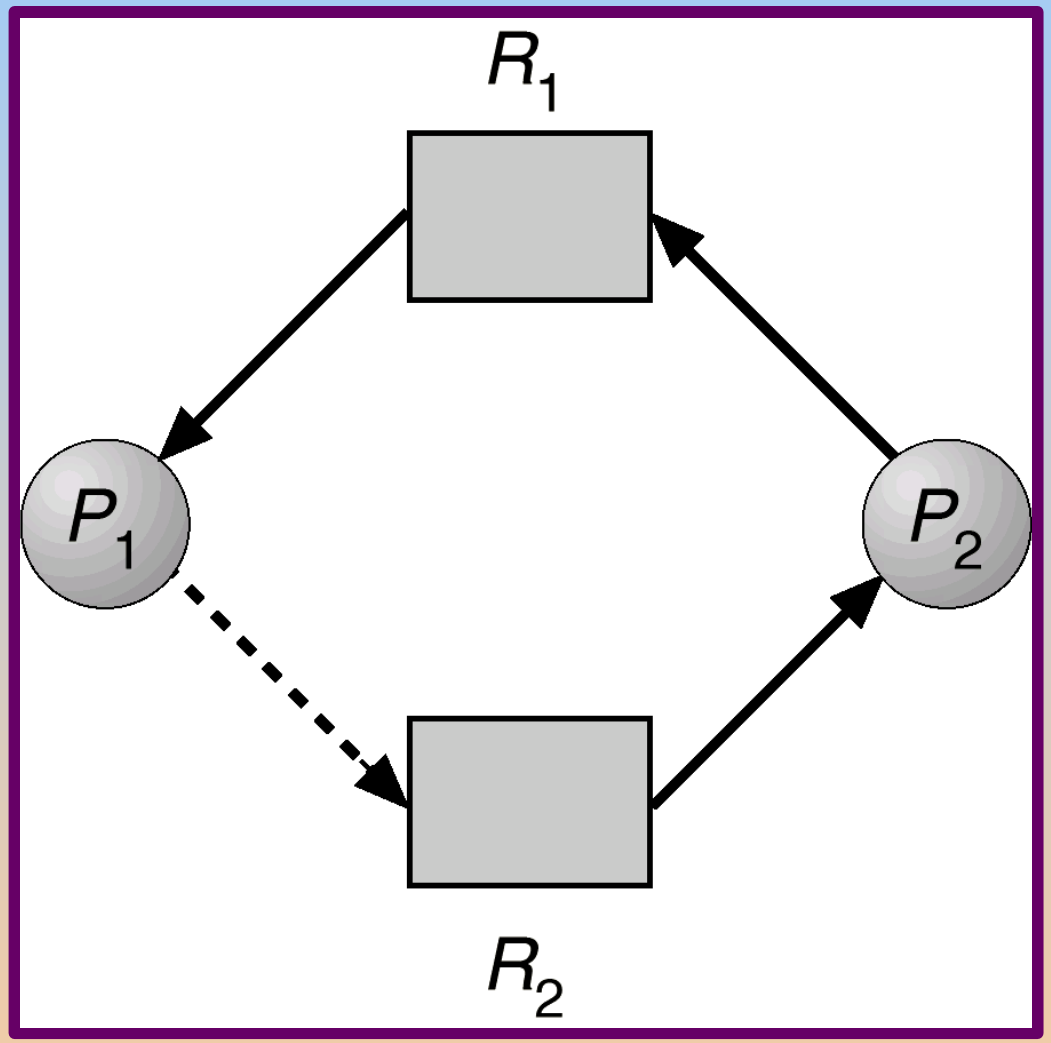
Algoritmo do grafo de alocação de recursos

- *Aresta de necessidade* $P_i \rightarrow R_j$ indica que o processo P_j pode requisitar o recurso R_j; representada por uma linha pontilhada.
- A resta de necessidade é convertida para uma aresta de requisição quando o processo solicita o recurso.
- Quando o processo libera o recurso, a aresta de requisição torna-se uma aresta de necessidade.
- Recursos podem ser uma necessidade *a priori* no sistema.

Grafo de alocação de recursos para prevenção de deadlock



Estado inseguro no grafo de alocação de recursos



Algoritmo do banqueiro (Banker's Algorithm)

- Múltiplas instâncias.
- Cada processo precisa declarar uma necessidade máxima a priori.
- Quando um processo requisita um recurso, ele pode ter que esperar.
- Quando um processo aloca todos os recursos, ele precisa liberá-los numa quantidade finita de tempo.

Estruturas de dados para o algoritmo do banqueiro

Seja n=número de processos e m=número de tipos de recursos

- *Disponibilidade:* Vetor de tamanho *m*. Se Disponibilidade [*j*] = *k*, existem k instâncias do recurso R_j disponíveis.
- *Max: matriz n x m*. Se *Max* [*i,j*] = *k*, então o processo P_i pode requisitar, no máximo, *k* instâncias do recurso R_j.
- *Alocação: Matriz n* x *m*. Se Alocação[*i,j*] = *k* então P_i tem alocadas *k* instâncias of R_j.
- *Necessidade: matriz n* x *m*. Se Necessidade[*i,j*] = *k*, então o processo P_i pode necessitar mais *k* instâncias do recurso R_j para completar suas tarefas.

Necessidade [*i,j*] = *Max*[*i,j*] – *Alocação* [*i,j*].

Algoritmo Estado Seguro(Safety)

1. Inicialização:

   Trabalho = *Disponilidade*

   Terminado [*i*] = *false* for *i* = 1,2, …, *n.*

2. Encontre *i* tal que:

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

   (b) *Necessidade[i,]* ≤ *Trabalho*

   Se não existe tal processo, vá para o passo 4.

3. *Trabalho* = *Trabalho* + *Alocação[i,]*

   Terminado[*i*] = *true*

   Vá para passo 2.

4. Se *Terminado* [*i*] == true para todo i, então o sistema está num estado seguro. Caso contrário, está num estado inseguro.

Algoritmo de requisição de recursos para P_i

Requisição = vetor de requisição do processos P_i. Se $Requisição_i[j] = k$ então o processo P_i quer k instâncias do tipo de recurso R_j.

1. Se $Requisição_i \leq Necessidade_i$ vá para o passo 2. Caso contrário, lançar uma condição de erro, pois o processo excedeu seu limite de necessidade declarada.
2. Se $Requisição_i \leq Disponibilidade$, vá para o passo 3. Caso contrário, P_i precisa esperar, pois os recursos não estão disponíveis.
3. Modificar o estado das estruturas de dados para:

$$Disponibilidade = Disponibilidade - Requisição_i$$

$$Alocação_i = Alocação_i + Requisição_i;$$

$$Necessidade_i = Necessidade_i – Requisição_{i;;}$$

- *Se sistema em estado seguro $\Rightarrow$ os recursos são alocados para P_i.*
- *Se não seguro $\Rightarrow$ P_i precisa esperar, e o estado anterior das estruturas de dados precisa ser recuperado.*

Exemplo do algoritmo do Banqueiro

- 5 processos; 3 tipos de recursos
- *A*(10 Instâncias), *B* (5 instâncias), e *C* (7 instâncias).
- Estado no tempo T_0:

	Alocação	*Max*	*Disponibilidade*
	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	

Exemplo (Cont.)

- O conteúdo da matriz Necessidade é mostrado abaixo:

	Necessidade
	A B C
P_0	7 4 3
P_1	1 2 2
P_2	6 0 0
P_3	0 1 1
P_4	4 3 1

- O sistema está num estado seguro pois a seqüência < P_1, P_3, P_4, P_2, P_0> satisfaz o critério de segurança.

P_1 Requisita (1,0,2) (Cont.)

- Checar se Requisição ≤ Disponibilidade (isto é, (1,0,2) ≤ (3,3,2) ⇒ *true.*

	Alocação	*Necessidade*	*Disponibilidade*
	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 1	6 0 0	
P_3	2 1 1	0 1 1	
P_4	0 0 2	4 3 1	

- A execução do algoritmo de estado seguro mostra que $<P_1, P_3, P_4, P_0, P_2>$ satisfaz o requisito de segurança. Assim, os recursos são alocados para P_1.

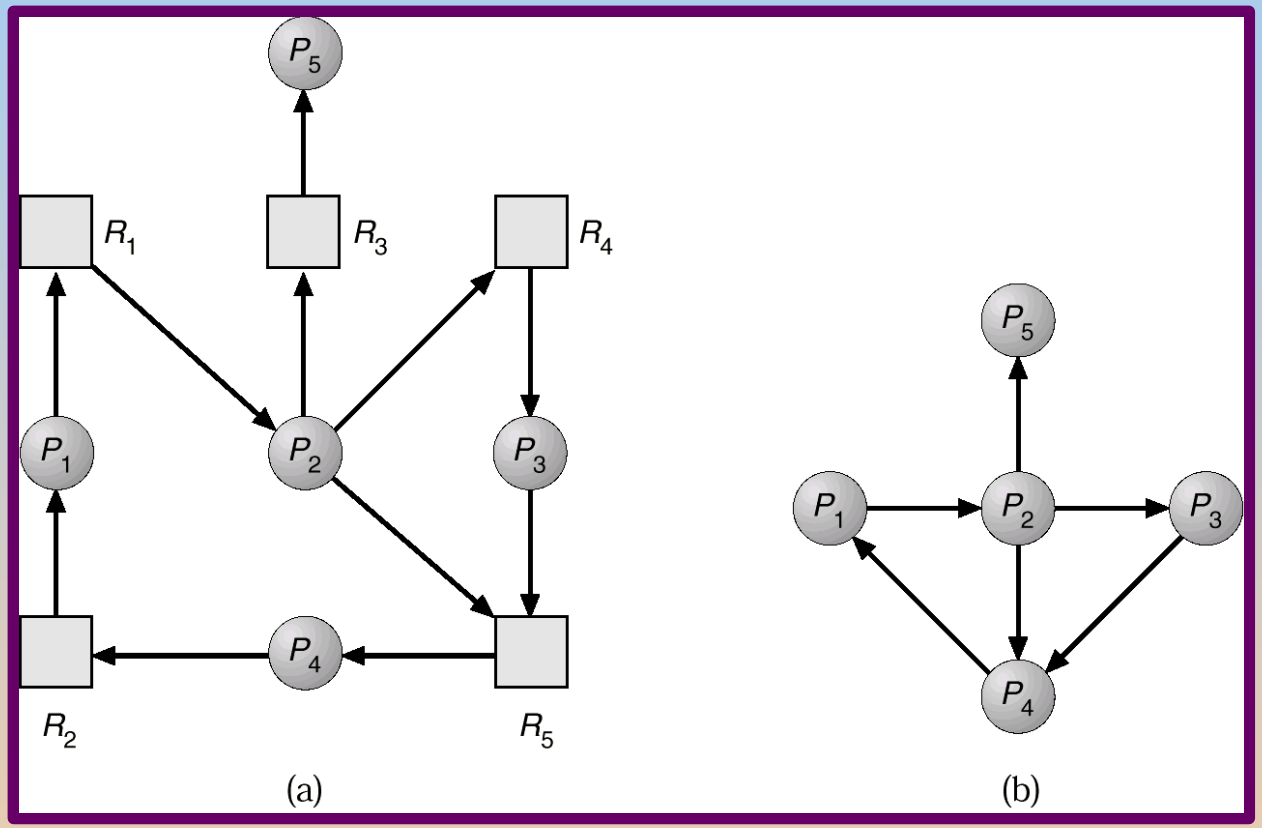
Detecção de deadlock

- Permite que o sistema entre num estado de deadlock
- Algoritmo de detecção
- Esquema de recuperação

Caso I: Única instância de cada tipo de recurso

- Manter um grafo de espera (wait-for)
  - Vértices são processos.
  - $P_i \rightarrow P_j$ se P_i está esperando por P_j.
- Periodicamente, o sistema invoca um algoritmo para detectar circuitos no grafo.
-
- Um algoritmo para detectar um ciclo num grafo necessita de uma ordem de n^2 operações, onde n é o número de vértices do grafo.

Grafo de alocação de recursos e grafo de espera



Grafo de alocação de recursos Grafo de espera correspondente

Caso 2: Várias instância de um tipo de recurso

- *Disponibilidade:* Um vetor de temanho m indica o número de recursos disponíveis de cada tipo.
- *Alocação:* Uma matriz *n* x *m* define o número de recursos de cada tipo alocado para um processo.
- *Requisição:* Uma matriz *n* x *m* que indica a requisição de cada processo. If $Requisição_i[\ j\ ] = k$, então o proceso P_i está requisitando mais k instância do tipo de recurso R_j.

Algoritmo de detecção

1. Inicialização:
   (a) *Trabalho* = *Disponibilidade*
   (b) Para *i* = 1,2, …, *n*, se *Alocação*$_i$ ≠ 0, então *Terminado*[i] = false;caso contrário, *Terminado*[i] = *true*.

2. Encontre um índice *i* tal que:
   (a) *Terminado*[*i*] == *false*
   (b) *Requisição*$_i$ ≤ *Trabalho*

   Se não existe tal I , vá para o passo 4.

3. *Trabalho* = *Trabalho* + *Alocação*$_i$

   Terminado[*i*] = *true*

   Vá para o passo 2.

4. Se *Terminado*[*i*] == false, algum *i*, 1 ≤ *i* ≤ *n*, então o sistema está em deadlock state. Ainda mais, se *Terminando*[*i*] == *false*, então P_i está no conjunto de processos em deadlock.

Exemplo de algoritmo de detecção

- 5 processos; três tipos de recursos
- A (7 instâncias), *B* (2 instâncias), e *C* (6 instâncias).
- Estado no tempo T_0:

	Alocação	*Requisição*	*Disponibilidade*
	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	

- Seqüência <P_0, P_2, P_3, P_1, P_4> irá resultar em *Terminado*[*i*] = true para todo *i*.

Exemplo (Cont.)

- P_2 requisita uma instância adicional do tipo C.

	Requisição
	A B C
P_0	0 0 0
P_1	2 0 1
P_2	0 0 1
P_3	1 0 0
P_4	0 0 2

- Estado do sistema?
  - Deadlock existe, consistindo dos P_1, P_2, P_3, and P_4.

Recuperação de deadlock: Término dos processos

- Terminar todos os processos em deadlock ou
- Abortar um processo de cada vez, até que o ciclo seja eliminado.