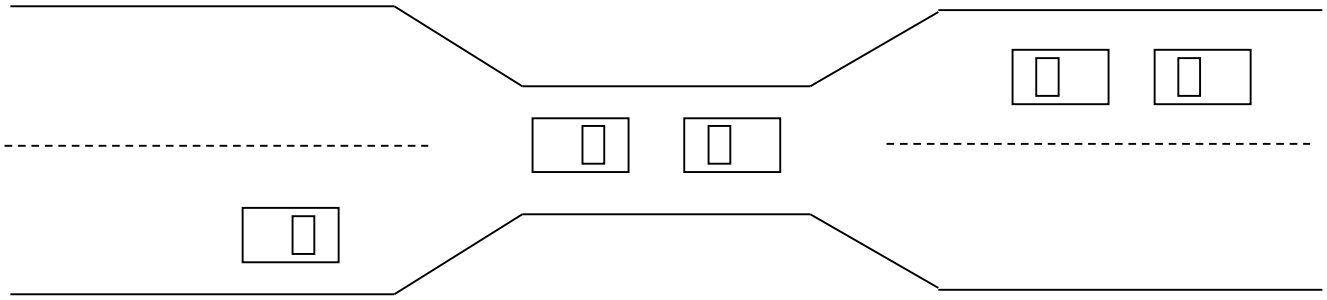
Deadlock x Starvation

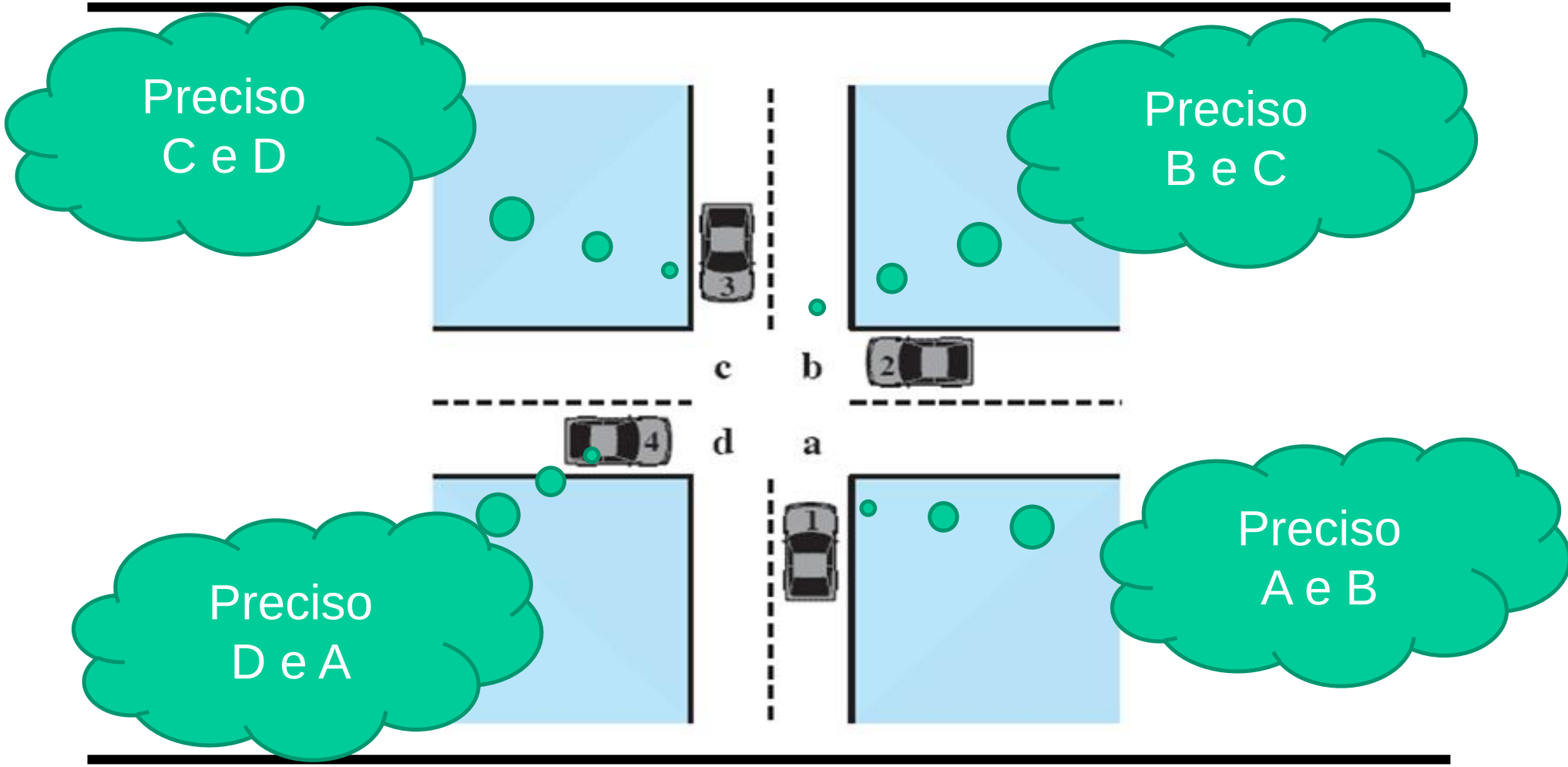
- Deadlock (impasse)
  - *todos os processos de um conjunto esperando por um evento que só pode ser causado por um processo do próprio conjunto*
  - processos travam seus recursos e buscam por outros que também estão travados por outros processos, gerando uma situação de congestionamento
- Starvation (postergação indefinida/abandono)
  - espera indefinida por recursos do sistema
- Deadlock: mais grave que starvation, pois trava o sistema e pode inviabilizar sua execução

Exemplo da travessia da ponte

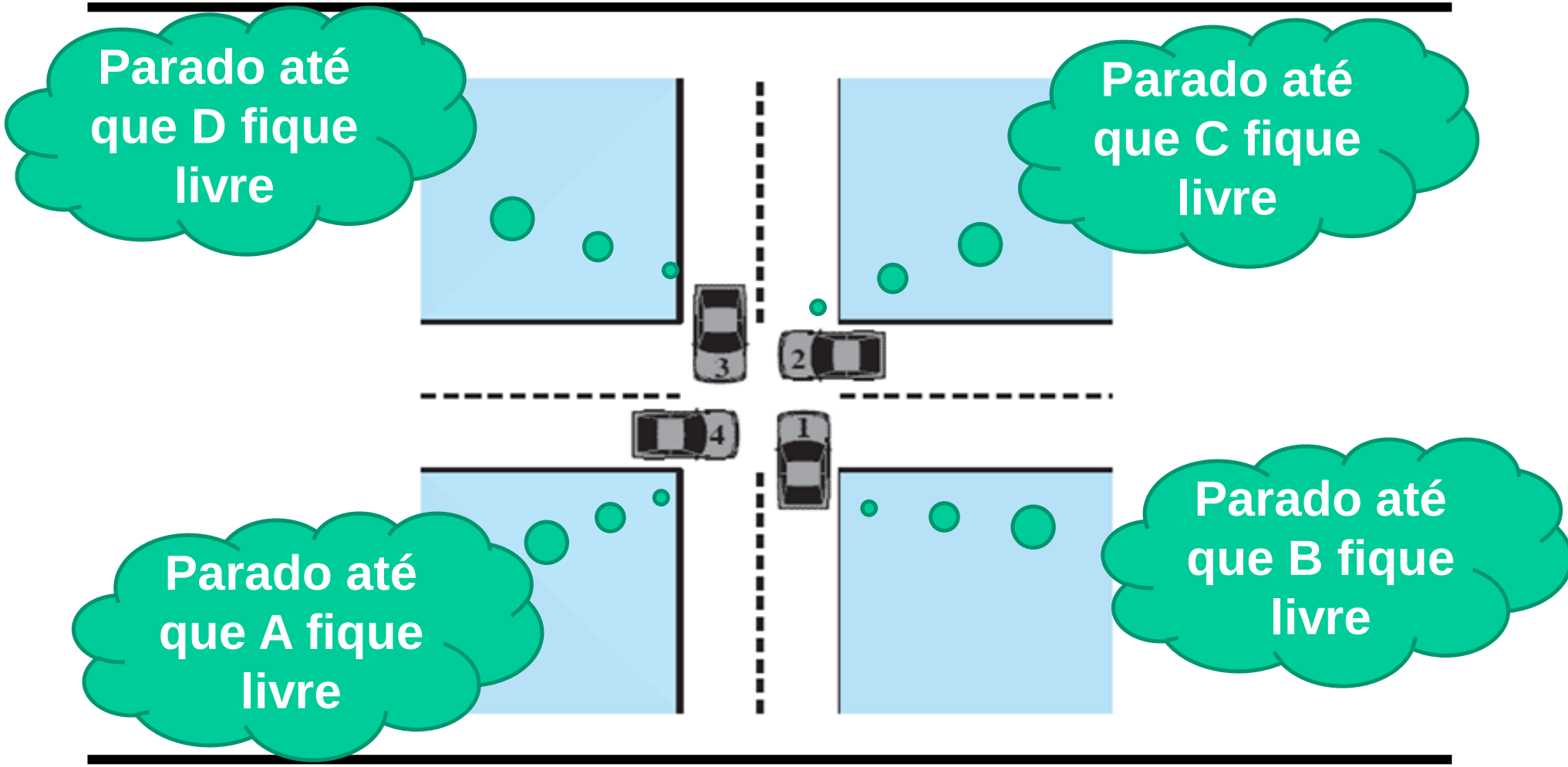


- tráfego em apenas uma direção
- cada seção da ponte pode ser vista como um recurso
- se um deadlock ocorre, pode ser resolvido se um carro dá ré (preempta recursos e retorna (*rollback*))
- vários carros podem ter que retornar se um deadlock ocorre
- starvation é possível

Deadlock em potencial



Deadlock


Parado até que D fique livre
Parado até que C fique livre
Parado até que A fique livre
Parado até que B fique livre
3
2
4
1

Recursos (1)

- deadlock ocorre quando...
  - processos têm acesso exclusivo a recursos
    - recursos “preemptáveis”
      - podem ser retirados do processo sem prejuízo
    - recursos “não preemptáveis”
      - causam a falha do processo quando retirado

- deadlock surge de recursos “não preemptáveis”

Recursos (2)

- cada processo utiliza um recurso obedecendo a sequência:
  - solicita
  - usa
  - libera

- pedido é negado:
  - processo pode bloquear e esperar
  - processo pode falhar, retornando um código de erro

Condições p/ ocorrência de deadlock (simultâneas !!!)

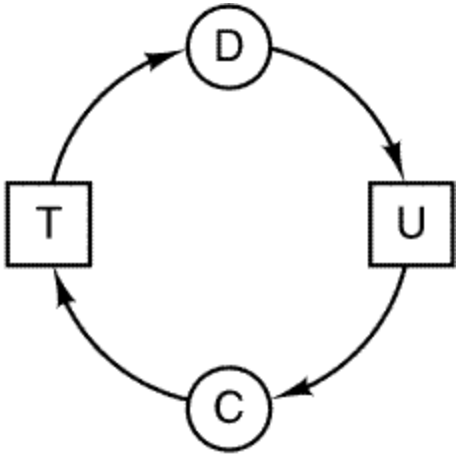
- exclusão mútua
  - recurso está atribuído a um processo ou está disponível
- segura e espera ("*hold and wait*")
  - processos possuindo recursos podem pedir mais recursos
- não preempção
  - recursos já atribuídos não podem ser retirados "a força"
- espera circular
  - cadeia circular de 2 ou mais processos, onde cada um espera por um recurso do seguinte na cadeia

Modelagem de deadlock (1)

- Grafo de alocação de recursos

(a)

(b)



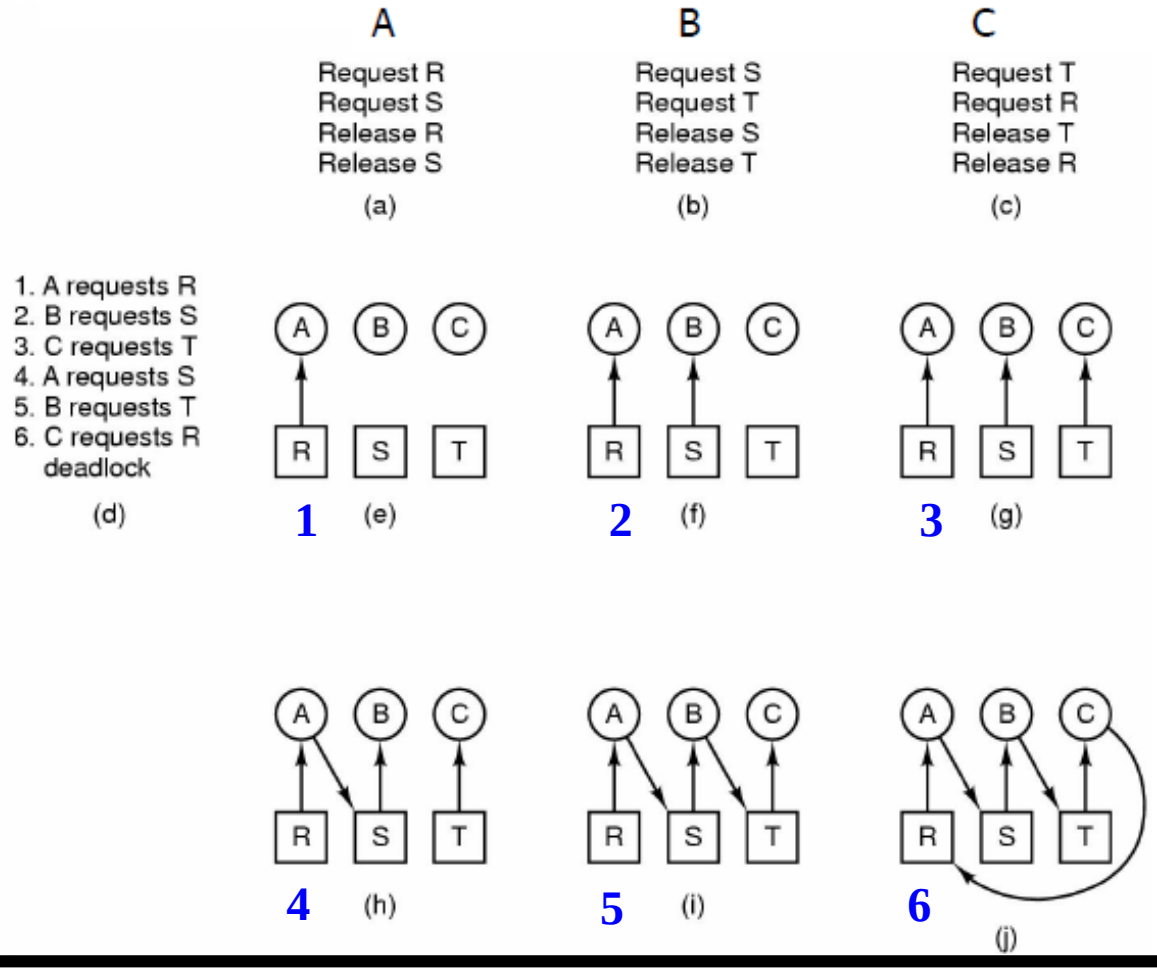
(c)

pA detém R

pB requisita S

deadlock

Deadlock: ocorrendo … (1)

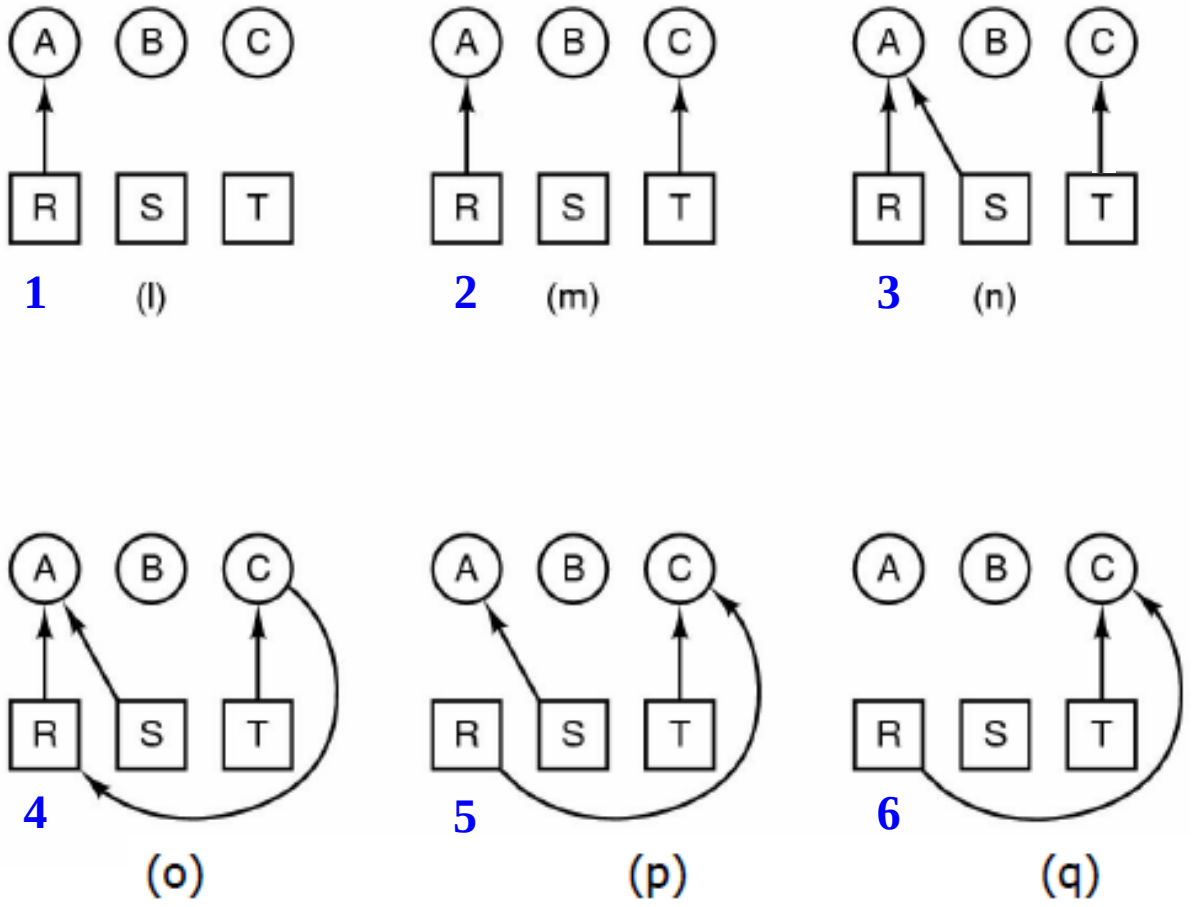


Deadlock: evitando … (2)

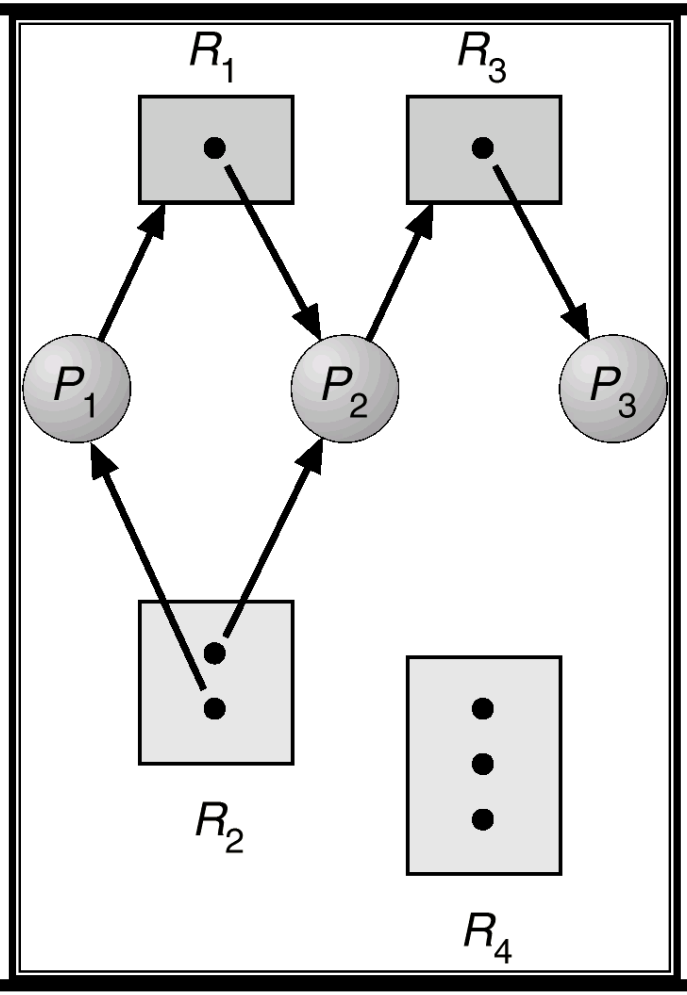
1. A requests R
2. C requests T
3. A requests S
4. C requests R
5. A releases R
6. A releases S

no deadlock

(k)



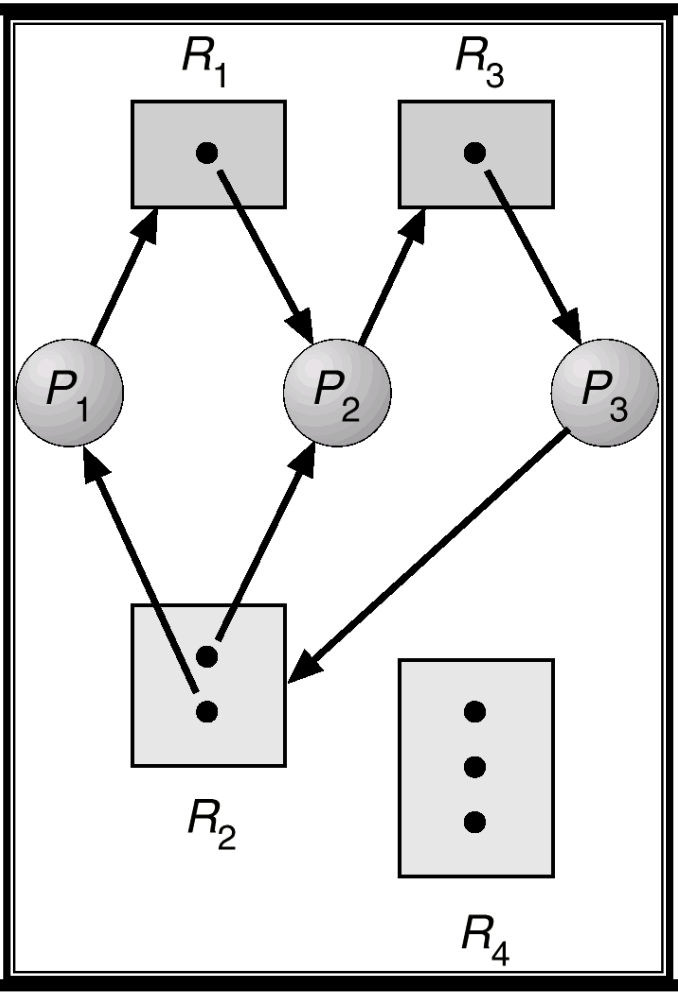
Exemplo de grafo de alocação de recursos



sem ciclo → sem deadlock

com ciclo → pode haver deadlock

Grafo de alocação de recursos com deadlock



2 ciclos:

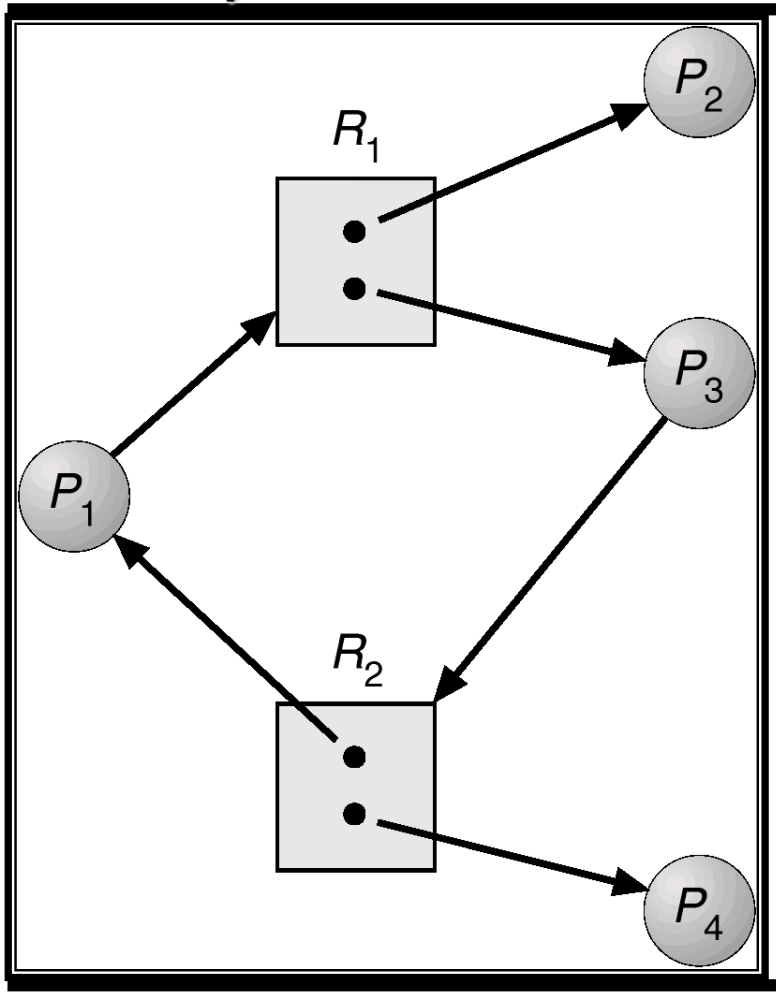
P1 → R1 → P2 → R3 → P3 → R2 → P1

P2 → R3 → P3 → R2 → P2

P1, P2 e P3 estão em deadlock:

P1 espera recurso R1 (mantido por P2)
P2 espera recurso R3 (mantido por P3)
P3 espera recurso R2 (mantidos por P1 e P2)

Grafo de alocação de recursos com um ciclo, sem deadlock



1 ciclo:

P1 → R1 → P3 → R2 → P1

não há deadlock

P4 pode liberar recurso R2, que pode ser alocado para P3, quebrando o ciclo

Fatos básicos

- grafo sem ciclos
  - ➢ não há deadlock

- grafo com ciclo
  - – se há apenas uma instância por tipo de recurso
    - ➢ há deadlock
  - – se há várias instâncias por tipo de recurso
    - ➢ há possibilidade de deadlock

Métodos para Manipulação de Deadlocks

1. Algoritmo do avestruz
   - simplesmente ignora o problema
2. Detecção e recuperação
   - deixa ocorrer, detecta e age de acordo
3. Evitar deadlocks
   - alocação cuidadosa de recursos
4. Prevenir deadlocks
   - evitar uma das quatro condições necessárias

1. Algoritmo do avestruz

- finge que não há problema
- aceitável se
  - deadlocks são raros
  - custo da prevenção for elevado

- UNIX e Windows seguem esta abordagem
  - Suposição: a maioria dos usuários prefere um deadlock ocasional a uma imposição de regras (limite no nº de processos criados, arquivos abertos, recursos solicitados, ...)

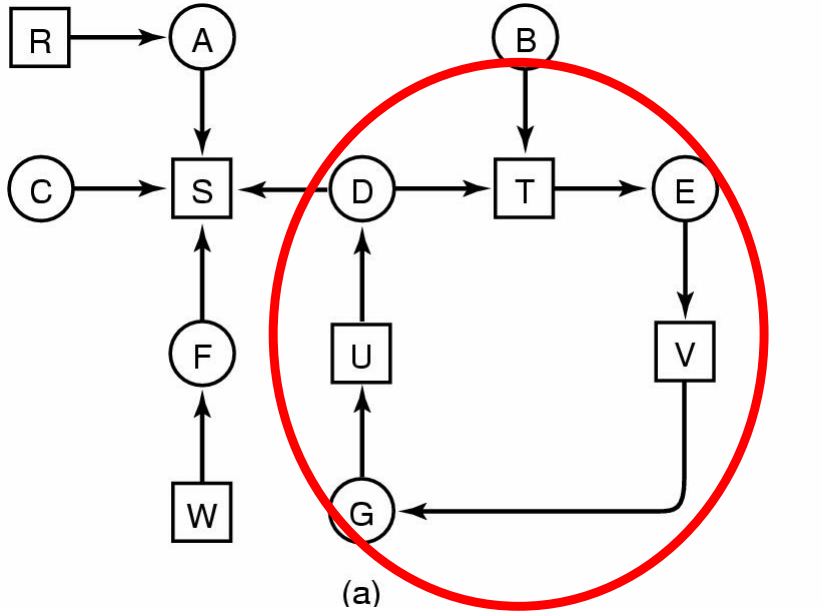
2. Detecção e recuperação

- processos estão com todos os recursos alocados
- permite que os *deadlocks* ocorram, tenta detectar as causas e solucionar a situação

  - Detecção com um recurso de cada tipo
  - Recuperação por preempção
  - Recuperação por *rollback*
  - Recuperação por eliminação de processos

2. Detecção e recuperação: detecção com um recurso de cada tipo

- Construção de um grafo
- Se houver ciclos, existem potenciais *deadlocks*



(a)

Situação:
A detém R e solicita S
B solicita T
C solicita S
D detém U e solicita S e T
E detém T e solicita V
F detém W e solicita S
G detém V e solicita U

2. Detecção e recuperação

- Como detectar ? → ok

- Quando procurar deadlock ?
  - a cada requisição → custo !!!
  - verificação periódica ???

- O que fazer após a detecção ?
  - Algoritmos de recuperação

2. Detecção e recuperação: algoritmos de recuperação (1)

- por preempção
  - retira temporariamente recurso de um processo sem que ele perceba e aloca a outro
  - depende do recurso
    - preemptável x não preemptável

- por rollback – reversão de estados
  - exige salvar checkpoints
  - reverte para estado anterior
    - perde trabalho já efetuado

2. Detecção e recuperação: algoritmos de recuperação (2)

- por eliminação de processos (radical, porém simples)
  - escolha de vítimas:
    - termina todos os processos ativos
    - termina apenas os processos envolvidos no deadlock
    - identifica os processos envolvidos e termina um a um verificando se a situação é resolvida
    - escolhe processo fora do ciclo p/ terminar, liberando seus recursos (critério ??)
  - problema: inconsistência
    - Ex.1) compilação – ok, reinício não sofre influência de execução anterior
    - Ex.2) atualização de BD – operações não podem ser desfeitas

3. Evitar deadlocks

- se o SO souber a priori a sequência de requisições dos processos
  - cada processo deve declarar o nº máximo de recursos de cada tipo de que pode precisar
  - algoritmo examina solicitações para garantir que não se forme espera circular
    - risco de deadlock → acesso negado

- soluções usam matrizes (ex. algoritmo do Banqueiro)
- definição de estados seguros e inseguros

4. Prevenir deadlocks (1)

- garantir que pelo menos uma das 4 condições necessárias p/ deadlock não ocorra:
- ***exclusão mútua***
  - evitar o uso de recursos exclusivos ???
    - não há como → necessária para recursos não compartilháveis
- ***segura e espera (hold and wait)***
  - não permitir que processos solicitem recursos aos poucos:
    - solicitam tudo de uma vez ou
    - liberam todos os que detêm antes de solicitar outros

4. Prevenir deadlocks (2)

- ***não preempção***
  - se um processo não conseguir alocar um novo recurso, deve abrir mão dos que já detém
  - recursos podem ser “tomados à força” de processos que estejam esperando por outros recursos

- ***espera circular***
  - define uma ordem total para todos os tipos de recursos
  - exige que os processos requisitem recursos nessa ordem (impede ciclos)