Deadlocks (Impasses)

Prof. Marcos Ribeiro Quinet de Andrade
Universidade Federal Fluminense - UFF
Instituto de Ciência e Tecnologia - ICT

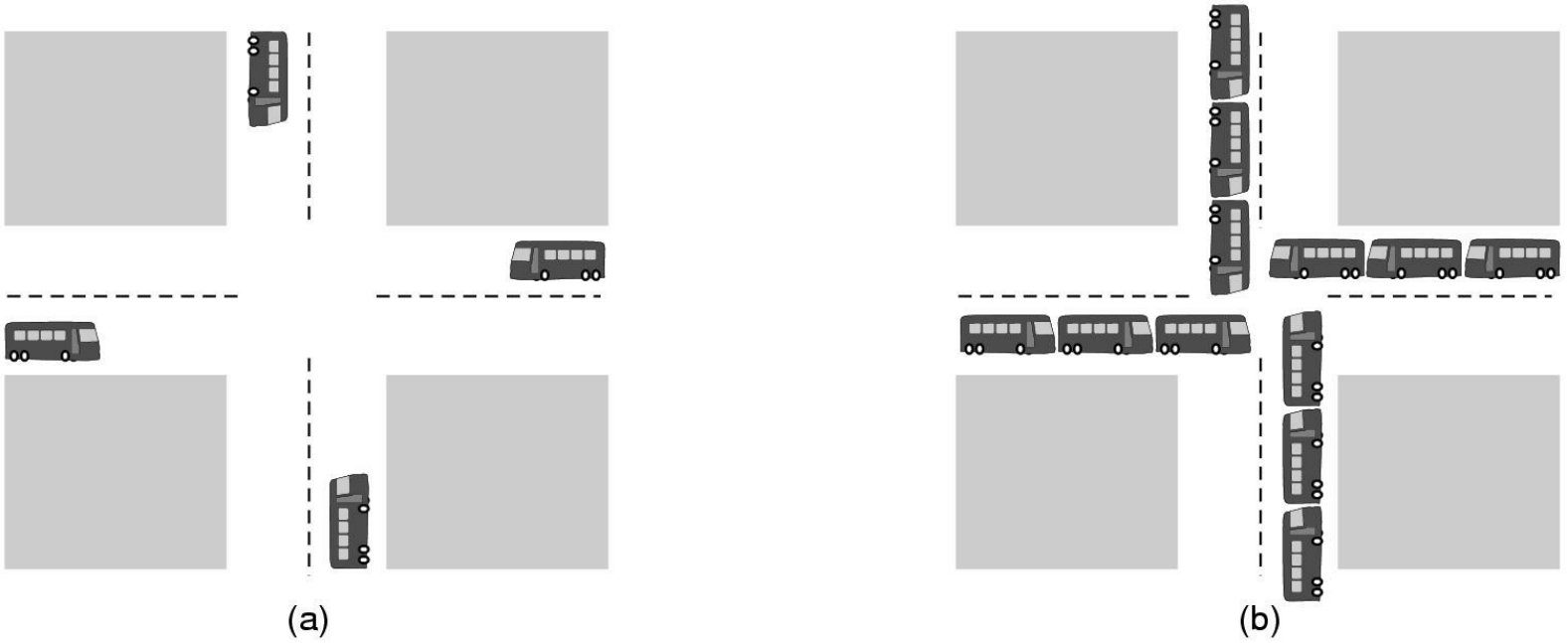
Deadlocks

- Recursos do sistema, como dispositivos em geral, são compartilhados a todo momento: impressora, disco, arquivos, etc.;

- Certos recursos do sistema não podem ser usados por dois ou mais processos simultaneamente, caso contrário, podem ocorrer erros para um ou todos os processos;

- *Deadlock:* processos ficam parados, sem possibilidade de poderem continuar seu processamento;

Deadlocks



Uma situação de *deadlock*

Deadlock na vida real

Recursos

- Recursos: objetos acessados, os quais podem ser tanto *hardware* quanto uma informação
  - Preemptivos: podem ser retirados do processo sem prejuízos;
    - Memória;
    - UCP;
  - Não-preemptivos: não podem ser retirados do processo, pois causam prejuízos;
    - Leitores ópticos (CD, DVD, etc);
    - Unidades de fita;
  - ***Deadlocks* ocorrem com recursos não-preemptivos**;

Recursos

- Operações sobre recursos/dispositivos:
  - Requisição do recurso;
  - Utilização do recurso;
  - Liberação do recurso.
- Se o recurso requerido não está disponível, duas situações podem ocorrer:
  - Processo que requisitou o recurso fica bloqueado até que o recurso seja liberado, ou;
  - Processo que requisitou o recurso falha, e depois de um certo tempo tenta novamente requisitar o recurso.

Recursos

- Aquisição do recurso
  - Para alguns tipos de recursos, os processos dos usuários gerenciam o uso dos recursos, através, por exemplo, de semáforos
    - Exemplo: acesso a registros em um sistema de banco de dados

- Se vários processos tentam acessar os mesmos recursos, podem ocorrer situações onde a ordem de solicitação dos recursos pode conduzir a um *deadlock* ou não

Deadlocks

- Definição formal:

  - “Um conjunto de processos estará em situação de *deadlock* se todo processo pertencente ao conjunto estiver esperando por um evento que somente um outro processo desse mesmo conjunto poderá fazer acontecer.”

Deadlocks

- Quatro condições para que ocorra um *deadlock*:
  - Exclusão mútua: cada recurso pode estar somente em uma de duas situações: ou associado a um único processo ou disponível;
  - Posse e espera (*hold and wait*): processo que, em um determinado instante, retêm recursos concedidos anteriormente, podem requisitar novos recursos;
  - Não-preempção: recursos já alocados não podem ser forçosamente retirados do processo que os alocou; somente o processo que alocou os recursos pode liberá-los;
  - Espera Circular: um processo pode esperar por recursos alocados a outro processo, que por sua vez espera por recursos alocados a outro processo, e assim por diante, até que o último espera por recursos alocados ao primeiro;

- Todas as condições devem ocorrer para que ocorra um *deadlock*

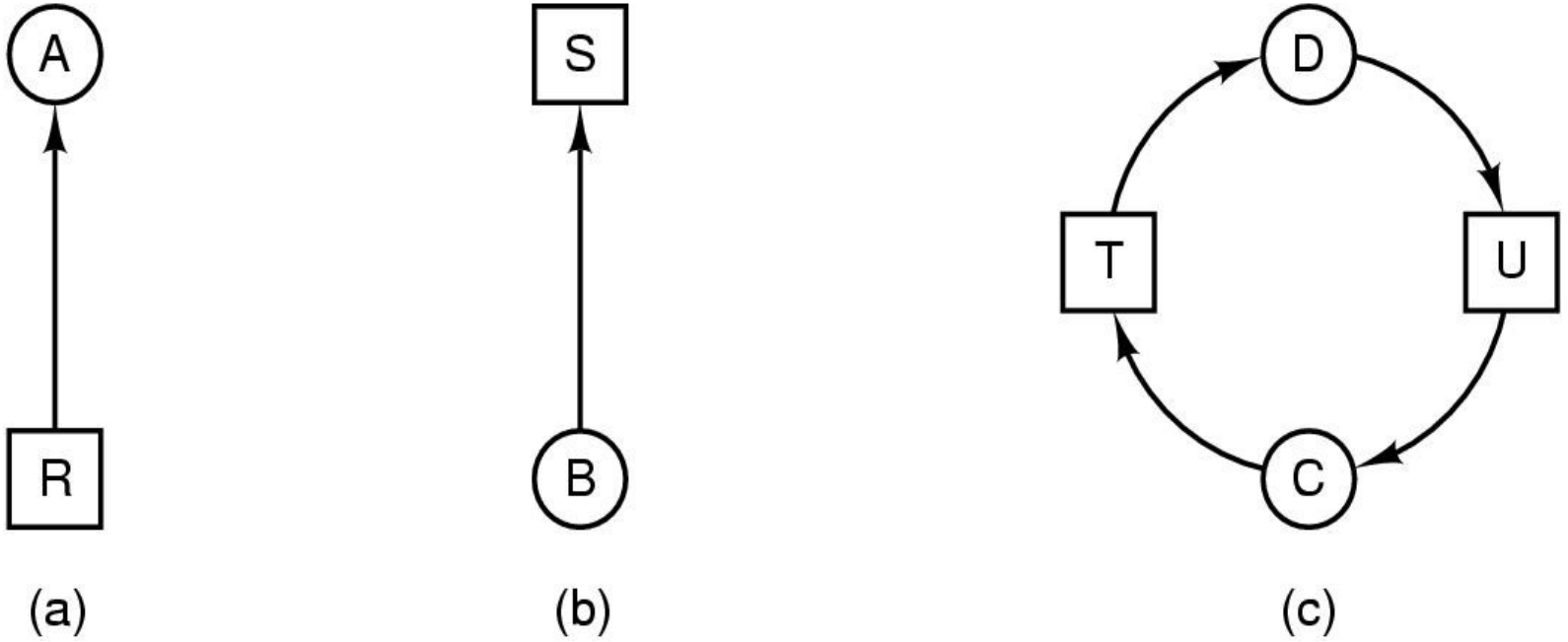
Deadlocks

- Geralmente, *deadlocks* são modelados através de grafos a fim de facilitar sua detecção, prevenção e recuperação

- Processos são simbolizados por círculos, enquanto recursos são representados por quadrados
  - Um arco de um recurso para um processo significa que o recurso está alocado para o processo;
  - Um arco de um processo para um recurso significa que o processo está bloqueado no momento, esperando que este recurso seja liberado;

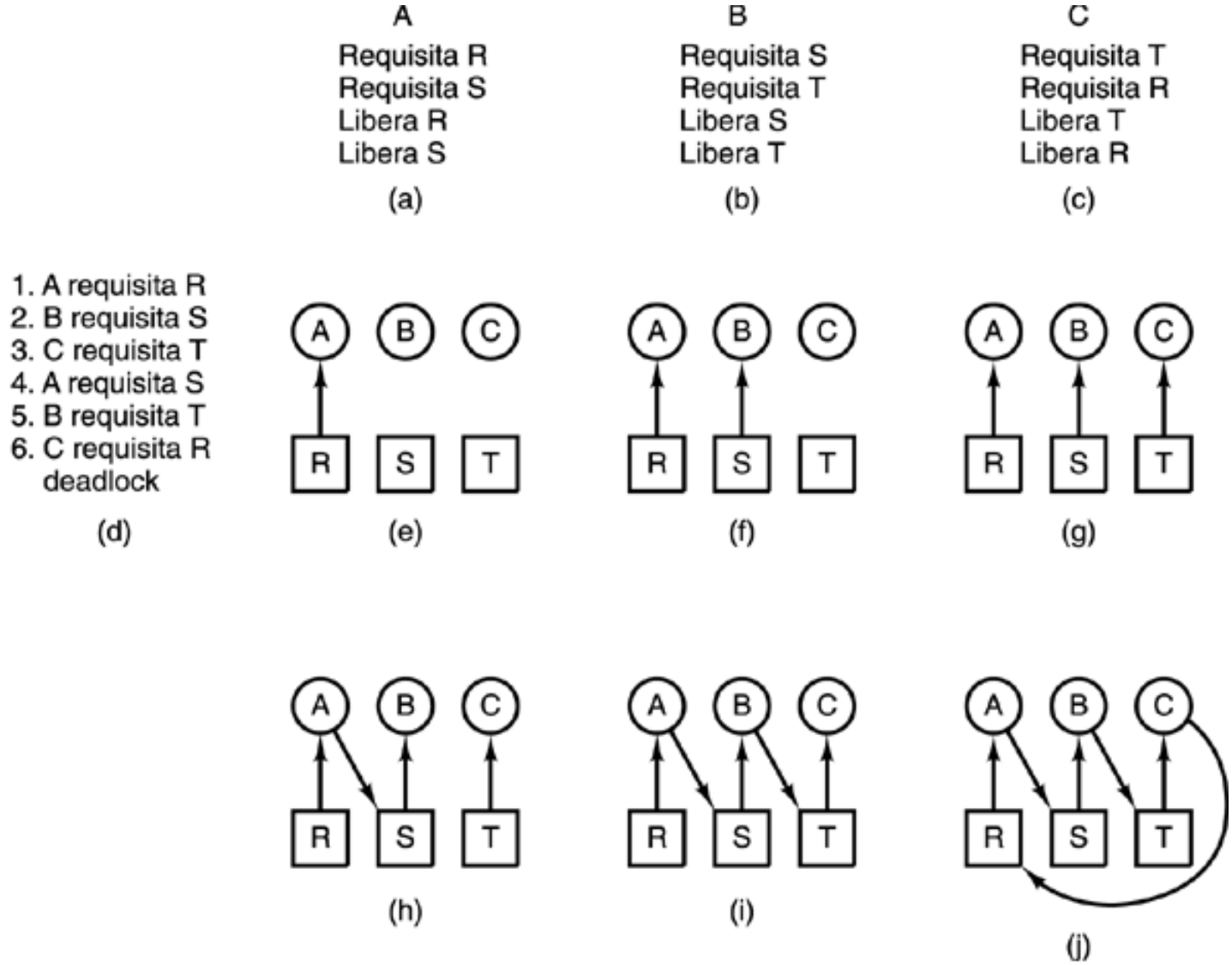
- A ocorrência de ciclos indica um *deadlock*;

Grafos de alocação de recursos



a) **Recurso R** alocado ao **Processo A**
b) **Processo B** requisita **Recurso S**
c) ***Deadlock***

Como ocorre um *deadlock*

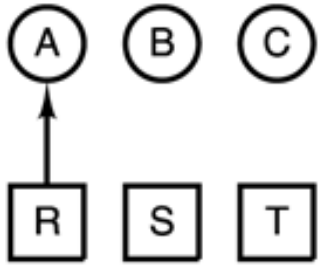


Evitando o *deadlock*

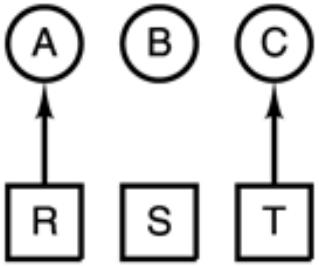
1. A requisita R
2. C requisita T
3. A requisita S
4. C requisita R
5. A libera R
6. A libera S

nenhum deadlock

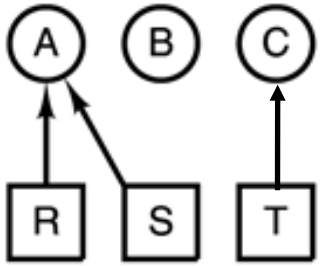
(k)



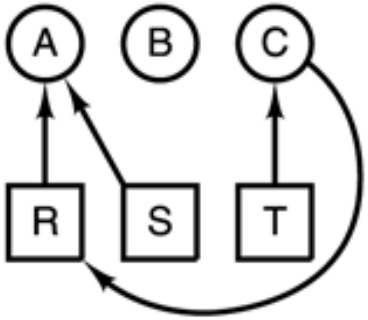
(l)



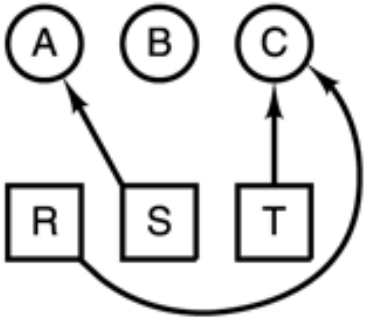
(m)



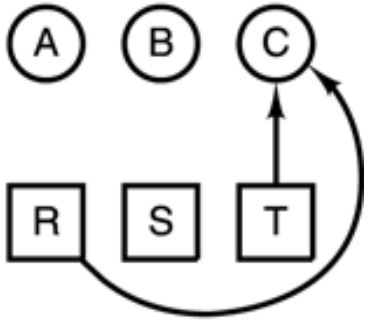
(n)



(o)



(p)



(q)

Deadlocks

- Quatro estratégias para tratar *deadlocks:*
  - Ignorar o problema;
  - Detectar e recuperar o problema;
  - Evitar dinamicamente o problema – alocação cuidadosa de recursos;
  - Prevenir o problema por meio da não satisfação de uma das quatro condições citadas anteriormente;

Deadlocks

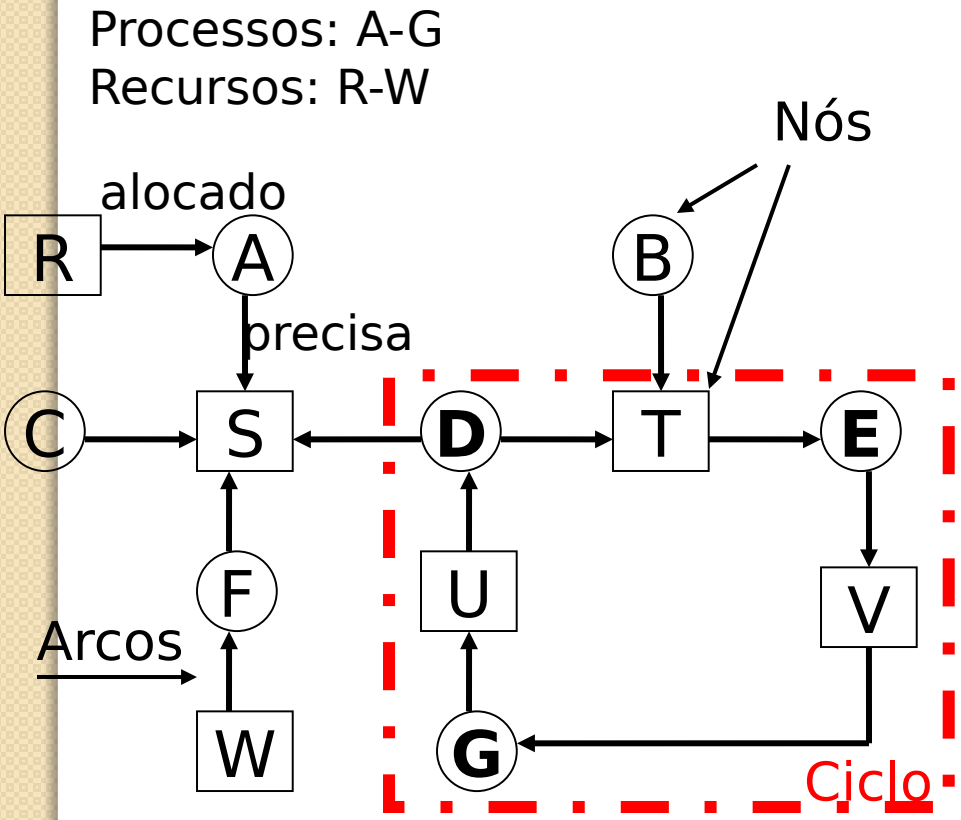
- **(1) Ignorar o problema**:
  - Algoritmo do AVESTRUZ;
  - Com qual frequência ocorre o problema;
  - Matemáticos geralmente não gostam dessa abordagem...
  - Alto custo – estabelecimento de condições para o uso de recursos;
  - Já empregada por sistemas Unix e Windows;

Deadlocks

- **(2) Detectar e Recuperar o problema:**
  - Processos estão com todos os recursos alocados;
  - Procedimento: Permite que os *deadlocks* ocorram, para em seguida tentar detectar as causas e solucionar a situação;
  - Algoritmos:
    - Detecção com um recurso de cada tipo;
    - Detecção com vários recursos de cada tipo;
    - Recuperação por meio de preempção;
    - Recuperação por meio de *rollback* (volta ao passado);
    - Recuperação por meio de eliminação de processos;

Deadlocks

- Detecção com um recurso de cada tipo:
  - Construção de um grafo;
  - Se houver ciclos, existem potenciais *deadlocks*;



Situação:
PA usa R e precisa de S;
PB precisa de T;
PC precisa de S;
PD usa U e precisa de S e T;
PE usa T e precisa de V;
PF usa W e precisa de S;
PG usa V e precisa de U;

Pergunta:
Há possibilidade de *deadlock*?

Deadlocks

- Detecção com vários recursos de cada tipo:
  - Classes diferentes de recursos – vetor de recursos existentes (E):
    - Se classe1=unidade de fita e E_1=2, então existem duas unidades de fita;
  - Vetor de recursos disponíveis (A):
    - Se ambas as unidades de fita estiverem alocadas, A_1=0;
  - Duas matrizes:
    - C: matriz de alocação corrente;
      - C_{ij}: número de instâncias do recurso j entregues ao processo i;
    - R: matriz de requisições;
      - R_{ij}: número de instâncias do recurso j que o processo i precisa;

Deadlocks

4 unidades de fita;
2 *plotters*;
3 *scanners*;
1 unidade de CD-ROM

Três processos:
P_1 usa um *scanner*;
P_2 usa duas unid. de fita e uma de CD-ROM;
P_3 usa um *plotter* e dois *scanners*;

Cada processo precisa de outros recursos (R);

Recursos existentes
E = (4 2 3 1)

UF P S UCD

Recursos disponíveis
A = (2 1 0 0)

UF P S UCD

Matriz de alocação

$$C = \begin{array}{c} \text{UF P S UCD} \\ \left[\begin{array}{cccc} 0 & 0 & 1 & 0 \\ 2 & 0 & 0 & 1 \\ 0 & 1 & 2 & 0 \end{array}\right] \\ \text{Recursos} \end{array} \begin{array}{l} \leftarrow P_1 \\ \leftarrow P_2 \\ \leftarrow P_3 \end{array}$$

Matriz de requisições

$$R = \begin{array}{c} \text{UF P S UCD} \\ \left[\begin{array}{cccc} 2 & 0 & 0 & 1 \\ 1 & 0 & 1 & 0 \\ 2 & 1 & 0 & 0 \end{array}\right] \end{array} \begin{array}{l} \leftarrow P_1 \\ \leftarrow P_2 \\ \leftarrow P_3 \end{array}$$

Deadlocks

4 unidades de fita;
2 *plotters*;
3 *scanners*;
1 unidade de CD-ROM

<u>Requisições</u>:
P_1 requisita duas unidades de fita e um CD-ROM;
P_2 requisita uma unidade de fita e um *scanner*;
P_3 requisita duas unidades de fita e um *plotter*;

Recursos existentes
E = (4 2 3 1)

Recursos disponíveis
A = (2 1 0 0) **P_3 pode executar**
A = (**0 0 0 0**)

Matriz de alocação

$$C = \begin{bmatrix} 0 & 0 & 1 & 0 \\ 2 & 0 & 0 & 1 \\ \mathbf{2} & \mathbf{2} & \mathbf{2} & \mathbf{0} \end{bmatrix} \begin{matrix} \leftarrow P_1 \\ \leftarrow P_2 \\ \leftarrow P_3 \end{matrix}$$

Matriz de requisições

$$R = \begin{bmatrix} 2 & 0 & 0 & 1 \\ 1 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix} \begin{matrix} \leftarrow P_1 \\ \leftarrow P_2 \\ \leftarrow \mathbf{P_3} \end{matrix}$$

Deadlocks

4 unidades de fita;
2 *plotters*;
3 *scanners*;
1 unidade de CD-ROM

Requisições:
P_1 requisita duas unidades de fita e um CD-ROM;
P_2 requisita uma unidade de fita e um *scanner*;
P_3 requisita duas unidades de fita e um *plotter*;

Recursos existentes
E = (4 2 3 1)

Recursos disponíveis
A = (2 1 0 0) **P_3 pode executar**
A = (**2 2 2 0**)

Matriz de alocação

$$C = \begin{bmatrix} 0 & 0 & 1 & 0 \\ 2 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \end{bmatrix} \begin{matrix} \leftarrow P_1 \\ \leftarrow P_2 \\ \leftarrow P_3 \end{matrix}$$

Matriz de requisições

$$R = \begin{bmatrix} 2 & 0 & 0 & 1 \\ 1 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix} \begin{matrix} \leftarrow P_1 \\ \leftarrow P_2 \\ \leftarrow \mathbf{P_3} \end{matrix}$$

Deadlocks

4 unidades de fita;
2 *plotters*;
3 *scanners*;
1 unidade de CD-ROM

Requisições:
P_1 requisita duas unidades de fita e um CD-ROM;
P_2 requisita uma unidade de fita e um *scanner*;
P_3 requisita duas unidades de fita e um *plotter*;

Recursos existentes
E = (4 2 3 1)

Recursos disponíveis
A = (2 1 0 0)
A = (2 2 2 0) **P_2 pode executar**
A = (**1 2 1 0**)

Matriz de alocação

$$C = \begin{bmatrix} 0 & 0 & 1 & 0 \\ \mathbf{3} & \mathbf{0} & \mathbf{1} & \mathbf{1} \\ 0 & 0 & 0 & 0 \end{bmatrix} \begin{matrix} \leftarrow P_1 \\ \leftarrow P_2 \\ \leftarrow P_3 \end{matrix}$$

Matriz de requisições

$$R = \begin{bmatrix} 2 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix} \begin{matrix} \leftarrow P_1 \\ \leftarrow \mathbf{P_2} \\ \leftarrow P_3 \end{matrix}$$

Deadlocks

4 unidades de fita;
2 *plotters*;
3 *scanners*;
1 unidade de CD-ROM

<u>Requisições</u>:
P_1 requisita duas unidades de fita e um CD-ROM;
P_2 requisita uma unidade de fita e um *scanner*;
P_3 requisita duas unidades de fita e um *plotter*;

Recursos existentes
E = (4 2 3 1)

Recursos disponíveis
A = (2 1 0 0)
A = (2 2 2 0) **P_2 pode executar**
A = (**4 2 2 1**)

Matriz de alocação

$$C = \begin{bmatrix} 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix} \begin{matrix} \leftarrow P_1 \\ \leftarrow P_2 \\ \leftarrow P_3 \end{matrix}$$

Matriz de requisições

$$R = \begin{bmatrix} 2 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix} \begin{matrix} \leftarrow P_1 \\ \leftarrow \mathbf{P_2} \\ \leftarrow P_3 \end{matrix}$$

Deadlocks

4 unidades de fita;
2 *plotters*;
3 *scanners*;
1 unidade de CD-ROM

Requisições:
P_1 requisita duas unidades de fita e um CD-ROM;
P_2 requisita uma unidade de fita e um *scanner*;
P_3 requisita duas unidades de fita e um *plotter*;

Recursos existentes
E = (4 2 3 1)

Recursos disponíveis
A = (2 1 0 0)
A = (2 2 2 0)
A = (2 2 2 0) **P_1 pode executar**

Matriz de alocação

$$C = \begin{bmatrix} \mathbf{2} & \mathbf{0} & \mathbf{1} & \mathbf{1} \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix} \begin{matrix} \leftarrow P_1 \\ \leftarrow P_2 \\ \leftarrow P_3 \end{matrix}$$

Matriz de requisições

$$R = \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix} \begin{matrix} \leftarrow \mathbf{P_1} \\ \leftarrow P_2 \\ \leftarrow P_3 \end{matrix}$$

Deadlocks

Ao final da execução, temos:

4 unidades de fita;
2 *plotters*;
3 *scanners*;
1 unidade de CD-ROM

Recursos existentes
E = (4 2 3 1)

Recursos disponíveis
A = (4 2 3 1)

Matriz de alocação

$$C = \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix} \begin{matrix} \leftarrow P_1 \\ \leftarrow P_2 \\ \leftarrow P_3 \end{matrix}$$

Matriz de requisições

$$R = \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix} \begin{matrix} \leftarrow P_1 \\ \leftarrow P_2 \\ \leftarrow P_3 \end{matrix}$$

Deadlocks – Situação 1

4 unidades de fita;
2 *plotters*;
3 *scanners*;
1 unidade de CD-ROM

Requisições:
P_1 requisita duas unidades de fita e um CD-ROM;
P_2 requisita uma unidade de fita e um *scanner*;
P_3 requisita uma unidade de fita e um *plotter*;

Recursos existentes
$E = (4\ \ 2\ \ 3\ \ 1)$

Recursos disponíveis
$A = (0\ \ 1\ \ 2\ \ 0)$

Matriz de alocação

$$C = \begin{bmatrix} 2 & 0 & 0 & 1 \\ 1 & 0 & 1 & 0 \\ 1 & 1 & 0 & 0 \end{bmatrix} \begin{matrix} \leftarrow P_1 \\ \leftarrow P_2 \\ \leftarrow P_3 \end{matrix}$$

Matriz de requisições

$$R = \begin{bmatrix} 2 & 0 & 0 & 1 \\ 2 & 0 & 1 & 1 \\ 1 & 0 & 0 & 0 \end{bmatrix} \begin{matrix} \leftarrow P_1 \\ \leftarrow P_2 \\ \leftarrow P_3 \end{matrix}$$

Nessa situação, nenhum processo pode ser atendido!
DEADLOCK

Deadlocks

- Detecção com vários recursos de cada tipo:
  - Nesse algoritmo, o sistema procura periodicamente por *deadlocks*;
  - CUIDADO:
    - Evitar ociosidade da UCP: quando se tem muitos processos em situação de *deadlock*, poucos processos estão em execução;

Deadlocks

- Recuperação de *Deadlocks*:
  - Por meio de preempção: possibilidade de retirar temporariamente um recurso de seu atual dono (processo) e entregá-lo a outro processo;
  - Por meio de retrocesso (*rollback)*: recursos alocados a um processo são armazenados em arquivos de verificação; quando ocorre um *deadlock*, os processos voltam ao estado no qual estavam antes do *deadlock*: **solução cara**;

Deadlocks

- Recuperação de *Deadlocks*:
  - Por meio de eliminação de processos: processos que estão no ciclo com *deadlock* são retirados do ciclo;
  - Melhor solução para processos que não causam algum efeito negativo ao sistema;
    - Ex1.: compilação – sem problemas;
    - Ex2.: atualização de um base de dados – problemas;

Deadlocks

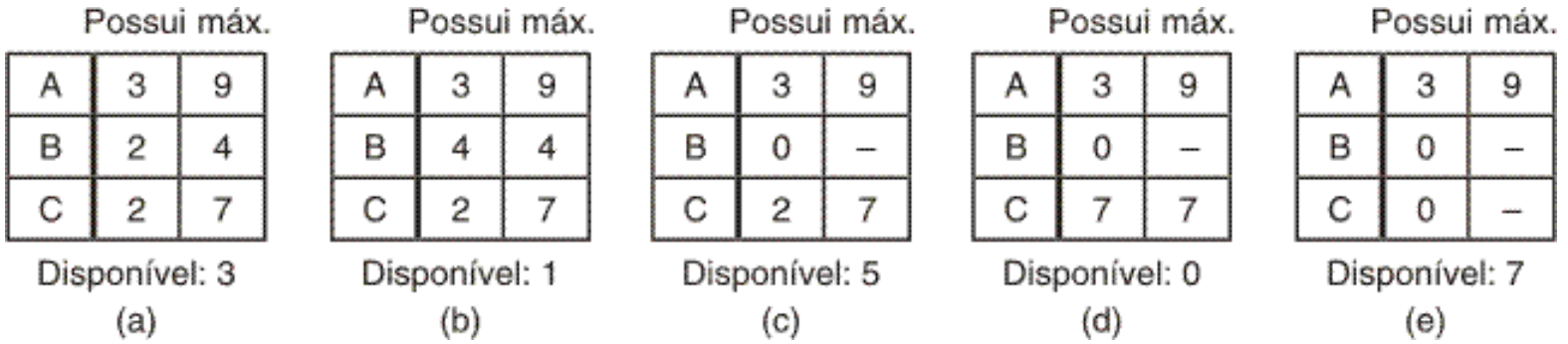
- **(3) Evitar dinamicamente o problema**:
  - Alocação individual de recursos à medida que o processo necessita;
  - Soluções também utilizam matrizes;
  - Escalonamento cuidadoso: alto custo;
    - Conhecimento prévio dos recursos que serão utilizados;
  - Algoritmos:
    - Banqueiro para um único tipo de recurso;
    - Banqueiro para vários tipos de recursos;
  - Definição de Estados Seguros e Inseguros;

Deadlocks

- Estados seguros: não provocam *deadlocks* e há uma maneira de atender a todas as requisições pendentes finalizando normalmente todos os processos;
  - A partir de um estado seguro, existe a garantia de que os processos terminarão;

- Estados inseguros: podem provocar *deadlocks,* mas não necessariamente provocam;
  - A partir de um estudo inseguro, não é possível garantir que os processos terminarão corretamente;

Estados seguros e inseguros



	Possui	máx.
A	3	9
B	2	4
C	2	7

Disponível: 3
(a)

	Possui	máx.
A	3	9
B	4	4
C	2	7

Disponível: 1
(b)

	Possui	máx.
A	3	9
B	0	–
C	2	7

Disponível: 5
(c)

	Possui	máx.
A	3	9
B	0	–
C	7	7

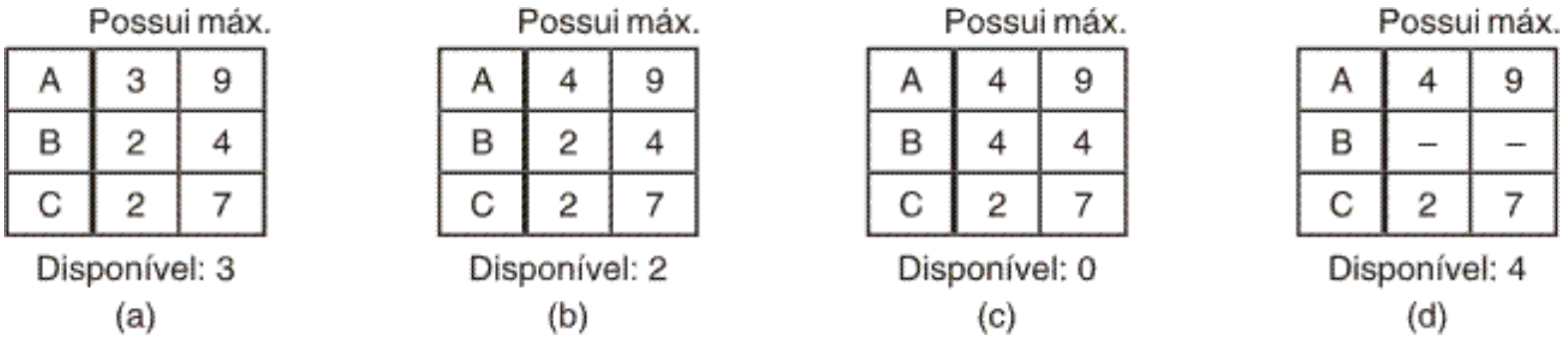
Disponível: 0
(d)

	Possui	máx.
A	3	9
B	0	–
C	0	–

Disponível: 7
(e)

- Demonstração de que o estado em (a) é seguro

Estados seguros e inseguros



	Possui	máx.
A	3	9
B	2	4
C	2	7

Disponível: 3
(a)

	Possui	máx.
A	4	9
B	2	4
C	2	7

Disponível: 2
(b)

	Possui	máx.
A	4	9
B	4	4
C	2	7

Disponível: 0
(c)

	Possui	máx.
A	4	9
B	–	–
C	2	7

Disponível: 4
(d)

- Demonstração de que o estado em (b) é inseguro

Deadlocks

- Algoritmos do Banqueiro:
  - Idealizado por Dijkstra (1965);
  - Considera cada requisição no momento em que ela ocorre, verificando se essa requisição leva a um estado seguro; Se sim, a requisição é atendida, se não o atendimento é adiado para um outro momento;
  - Premissas adotadas por um banqueiro (SO) para garantir ou não crédito (recursos) para seus clientes (processos);
  - Nem todos os clientes (processos) precisam de toda a linha de crédito (recursos) disponível para eles;

Deadlocks

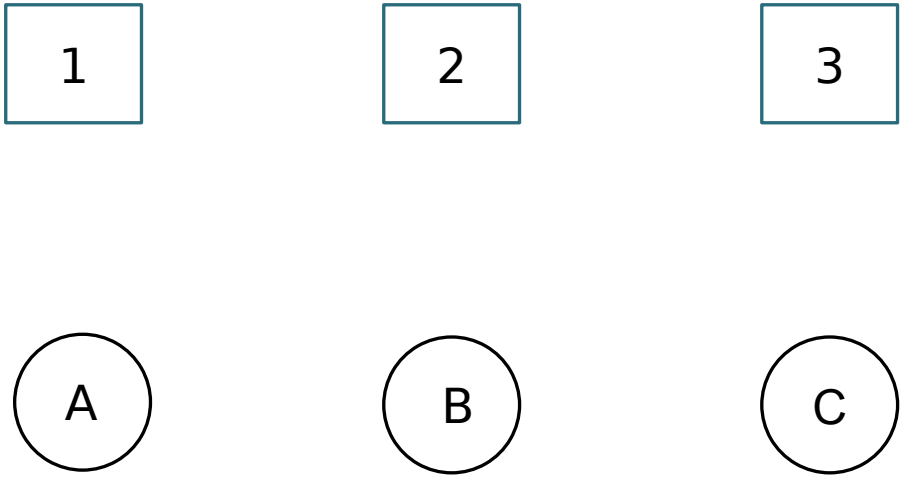
- Algoritmo do Banqueiro:
  - Desvantagens
    - Pouco utilizado, pois é difícil saber quais recursos serão necessários;
    - Escalonamento cuidadoso é caro para o sistema;
    - O número de processos é dinâmico e pode variar constantemente, tornando o algoritmo custoso;
  - Vantagem
    - Na teoria, o algoritmo do banqueiro é considerado ótimo;

Deadlocks

- Prevenir *Deadlocks*:
  - Atacar uma das quatro condições:

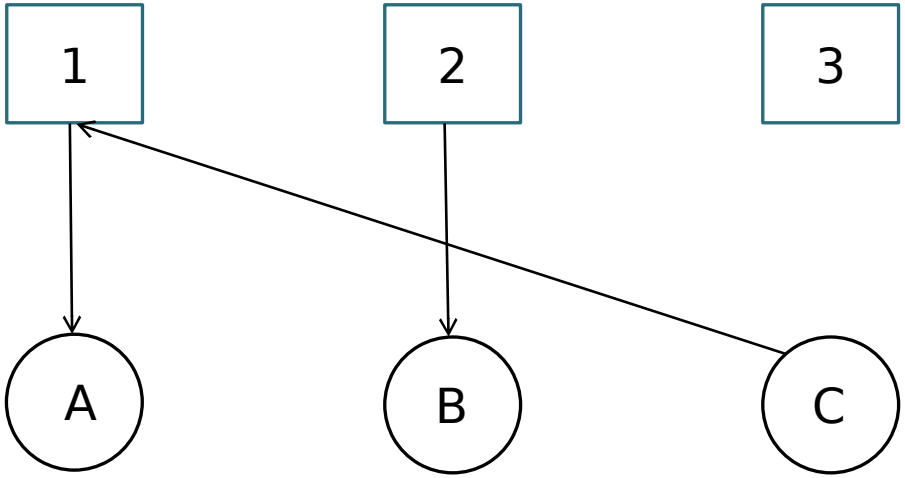
Condição	**Abordagem**
Exclusão Mútua	Alocar todos os recursos usando um *spool*
Posse e Espera	Requisitar todos os recursos inicialmente para execução – difícil saber; sobrecarga do sistema
Não-preempção	Retirar recursos dos processos – pode ser ruim dependendo do tipo de recurso; praticamente não implementável
Espera Circular	Ordenar numericamente os recursos, e realizar solicitações em ordem numérica Permitir que o processo utilize apenas um recurso por vez

Espera Circular



- O processo A precisa dos recursos 1 e 2 para executar;
- O processo B precisa dos recursos 2 e 3 para executar;
- O processo C precisa dos recursos 1 e 3 para executar.

Espera Circular



- Se os processo forem executados na ordem A, B e C:
  - A recebe o recurso 1, B recebe o recurso 2 e C é bloqueado, pois se não conseguir garantir o recurso 1, não pode solicitar ao SO a alocação do recurso 3, que permanece disponível. O que acontece a seguir?

Deadlocks

- *Deadlocks* podem ocorrer sem o envolvimento de recursos, por exemplo, se semáforos forem implementados erroneamente;

```
..                  ..
down(&empty);       down(&mutex);
down(&mutex);       down(&empty);
    ..                  ..
```

- *Inanição (Starvation)*
  - *Todos os processos devem conseguir utilizar os recursos que precisam, sem ter que esperar indefinidamente;*
  - *Alocação usando FIFO;*

Deadlocks

Potencial *deadlock*

```
semaphore resource_1;
semaphore resource_2;

void Process_A (void){
   down(&resource_1);
   down(&resource_2);
   use_both_resources();
   up(&resource_2);
   up(&resource_1);
}
 void Process_B (void){
   down(&resource_2);
   down(&resource_1);
   use_both_resources();
   up(&resource_1);
   up(&resource_2);}
```

Livre de *deadlock*

```
semaphore resource_1;
semaphore resource_2;

void Process_A (void){
   down(&resource_1);
   down(&resource_2);
   use_both_resources();
   up(&resource_2);
   up(&resource_1);
}
 void Process_B (void){
   down(&resource_1);
   down(&resource_2);
   use_both_resources();
   up(&resource_1);
   up(&resource_2);}
```