

# Sistemas Operacionais

## *Deadlock*

Prof. Otávio Gomes

[otavio.gomes@unifei.edu.br](mailto:otavio.gomes@unifei.edu.br)



## Impasse ou Deadlock

Communications of the ACM, Jan 1988



## Impasse ou *Deadlock*

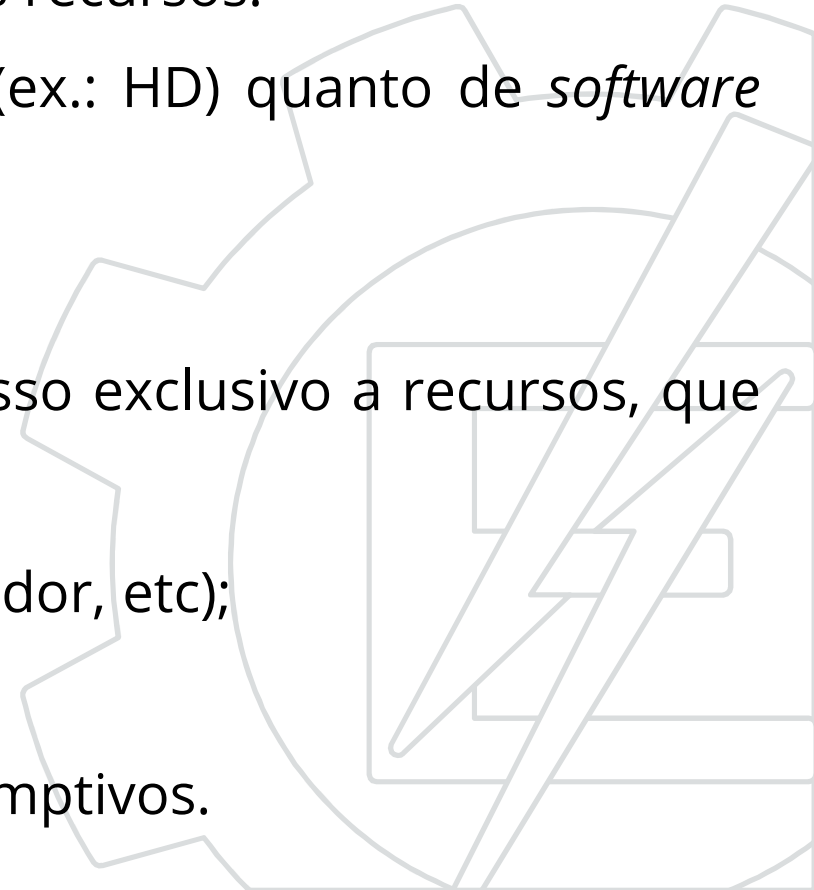
- Ocorrem em virtude do compartilhamento de recursos escassos.





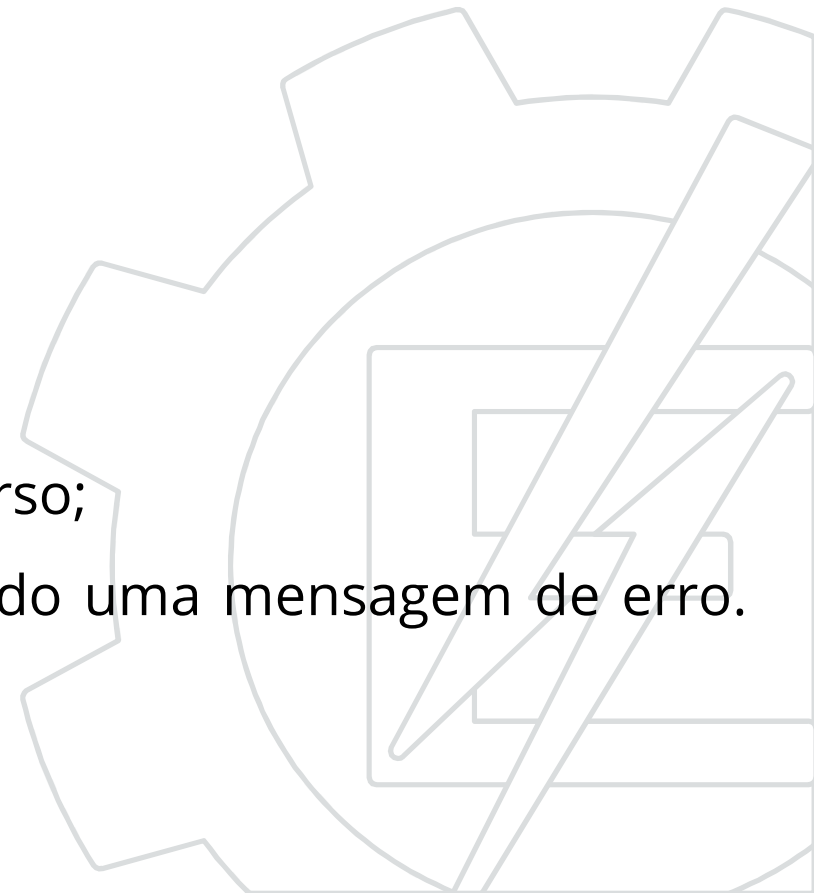
## Impasse ou *Deadlock*

- Ocorrem em virtude do compartilhamento de recursos escassos.
- Sistemas Operacionais devem fornecer acesso aos recursos.
- Podem ocorrer tanto em recursos de *hardware* (ex.: HD) quanto de *software* (ex.: Banco de Dados).
- *Deadlocks* ocorrem quando processos obtêm acesso exclusivo a recursos, que podem ser:
  - Preemptivos (p.ex.: memória, CPU com escalonador, etc);
  - Não-preemptivos (p.ex.: impressora).
    - Geralmente ocorrem com recursos não preemptivos.



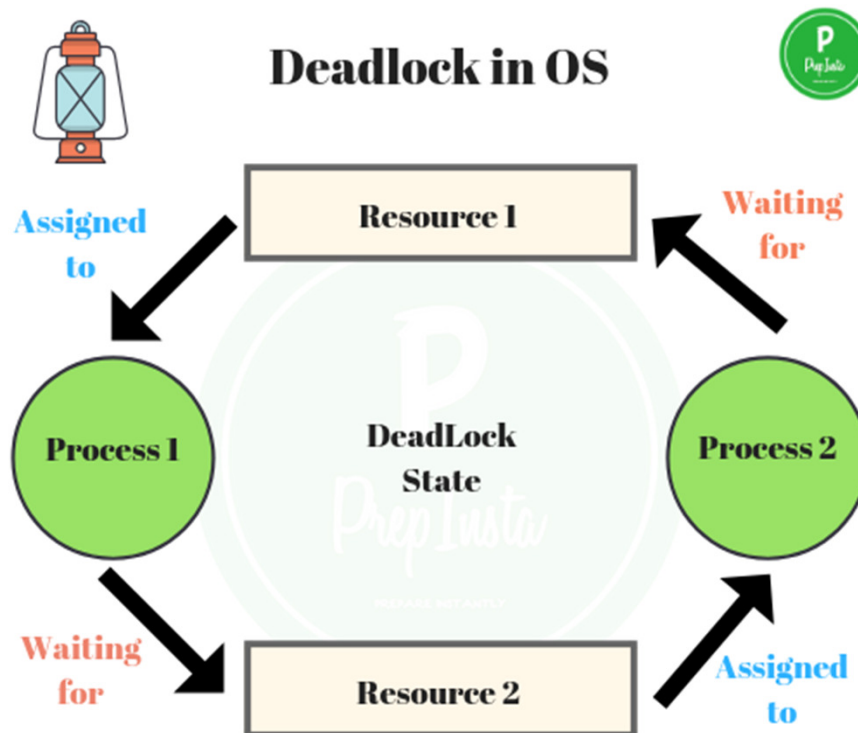
## Impasse ou *Deadlock*

- Eventos para o uso de recursos (pode ser feito por *mutexes*):
  - 1) Requisição;
  - 2) Utilização;
  - 3) Liberação.
- Se o recurso requerido não está disponível:
  - Processo fica **bloqueado** até a liberação do recurso;
  - Processo que requisitou o recurso **falha**, gerando uma mensagem de erro.Após um intervalo de tempo tenta novamente.



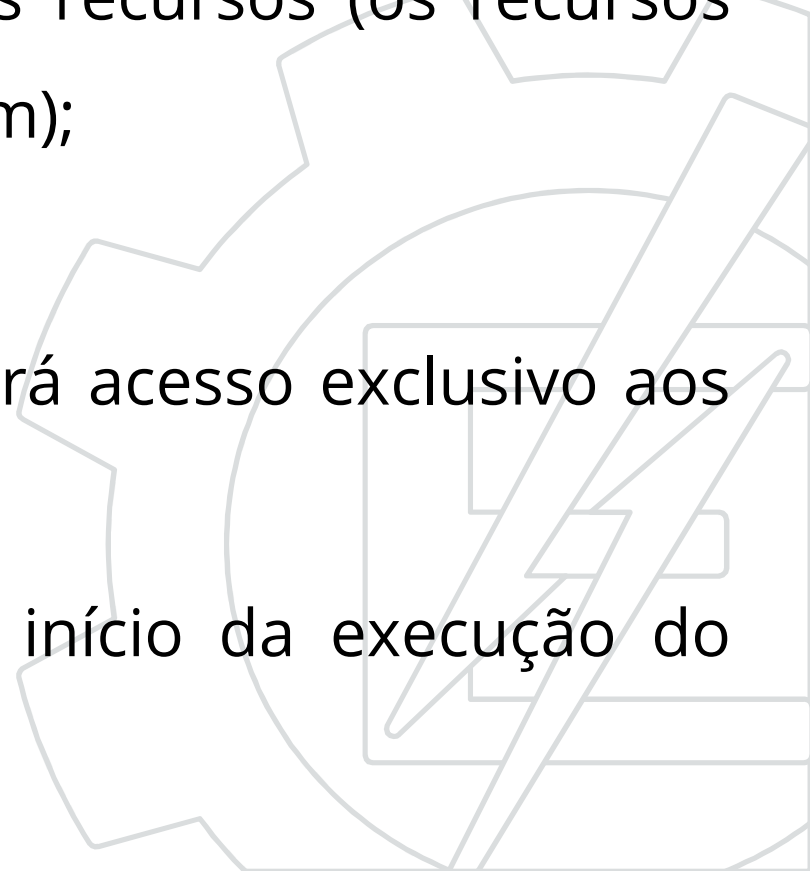
## Impasse ou *Deadlock*

- Processos estão em *deadlock* se cada processo estiver esperando por um evento que somente outro processo no conjunto pode causar.



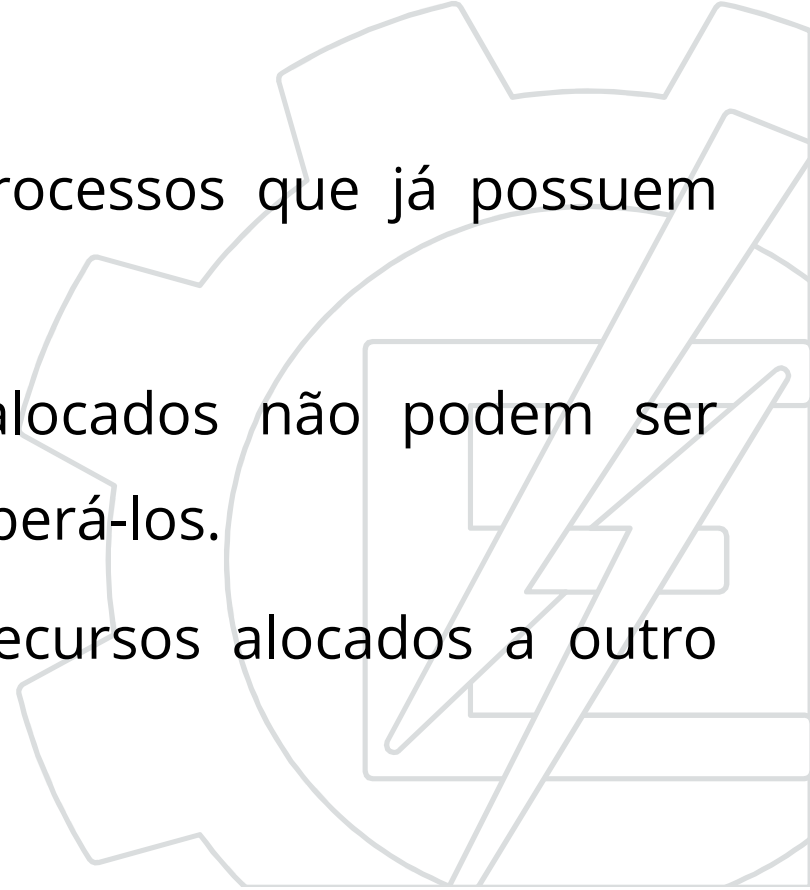
## Impasse ou *Deadlock*

São exemplos de estratégias para prevenir *deadlock*:

- Uso de ordenação por precedência dos recursos (os recursos devem ser alocados seguindo uma ordem);
  - Determinação de tempo de alocação;
  - Determinação que nenhum processo terá acesso exclusivo aos recursos; e
  - Pré-alocação de todos os recursos no início da execução do programa.
- 

## Impasse ou *Deadlock*

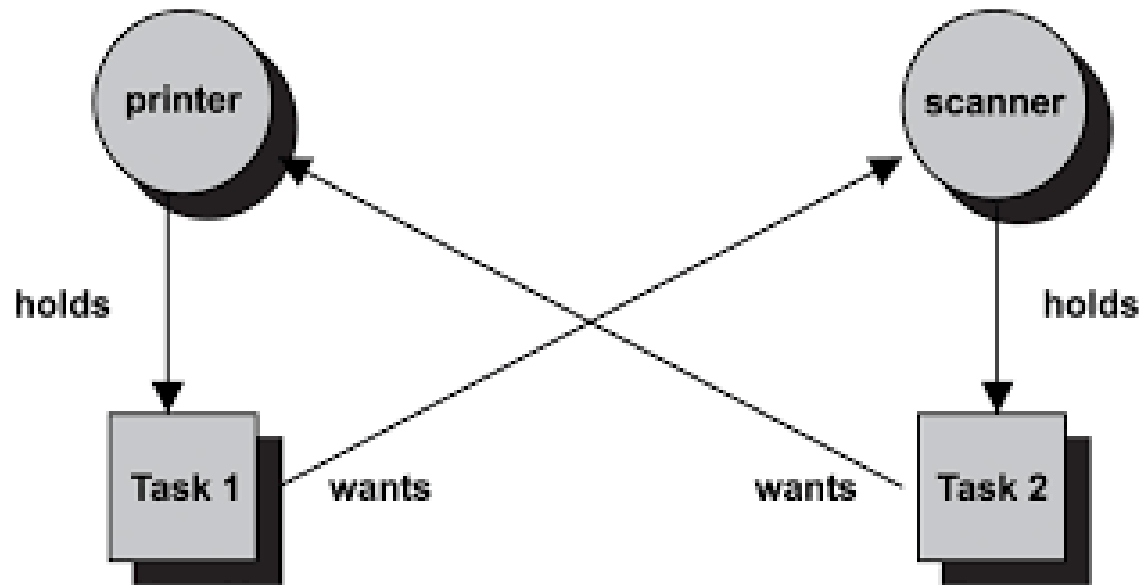
### Condições necessárias:

- **Exclusão mútua:** um recurso só pode estar alocado para um processo em um determinado momento.
  - **Posse durante a espera** (*hold and wait*): processos que já possuem algum recurso podem requerer outros.
  - **Inexistência de preempção:** recursos já alocados não podem ser retirados; somente o próprio processo pode liberá-los.
  - **Espera circular:** um processo espera por recursos alocados a outro processo, em uma cadeia circular.
- 



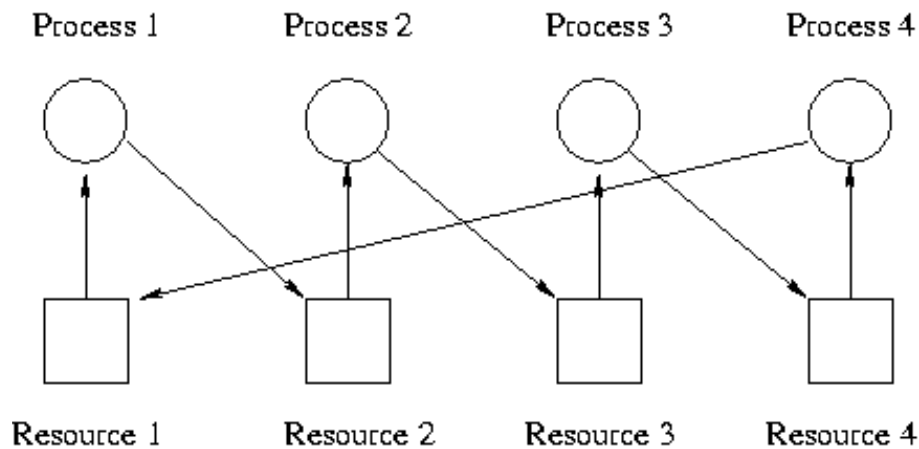
## Impasse ou *Deadlock*

- **Posse durante a espera** (*hold and wait*): processos que já possuem algum recurso podem requerer outros.



## Impasse ou *Deadlock*

- O processo 1 espera pelo processo 2, que espera pelo processo 3, que espera pelo processo 4, que espera pelo processo 1.



- Geralmente, *deadlocks* são representados por grafos dirigidos, a fim de facilitar sua detecção, prevenção e recuperação.

## **Modelo do sistema**

Grafo de alocação de recursos

- **Recursos** têm vários tipos  $R_1, R_2, \dots, R_m$ 
  - ex.: ciclos de CPU, espaço de memória, dispositivo de E/S
- Cada recurso  $R_i$  tem  $W_i$  **instâncias**
  - ex.:  $R_1$ =impressora,  $W_1$ ={imprA, imprB, imprC}
- Processos acessam recursos da mesma forma:
  - **Solicitação\*** → **Uso** → **Liberação**
    - \* se não puder ser atendida imediatamente, espera
- Solicitação e Liberação são **chamadas de sistema**, ex.:
  - arquivo: *open()* e *close()*
  - memória: *allocate()* e *free()*



## **Modelo do sistema**

Grafo de alocação de recursos

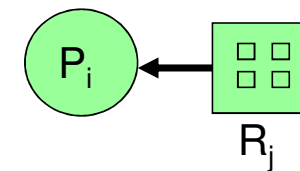
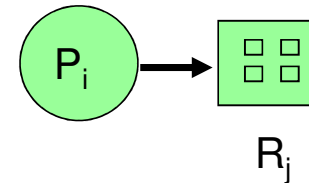
- Permite descrever com mais precisão *deadlocks*
- Vértices são divididos em dois tipos:
  - $P = \{P_1, P_2, \dots, P_n\}$ , os processos no sistema
  - $R = \{R_1, R_2, \dots, R_m\}$ , os recursos do sistema
- Arestas são também de dois tipos:
  - Solicitação: aresta direcionada  $P_i \rightarrow R_j$
  - Atribuição: aresta direcionada  $R_i \rightarrow P_j$



## Modelo do sistema

Grafo de alocação de recursos

- Um processo:
- Tipo de recurso com 4 instâncias
- $P_i$  requisita instância de  $R_j$
- $P_i$  detém uma instância de  $R_j$

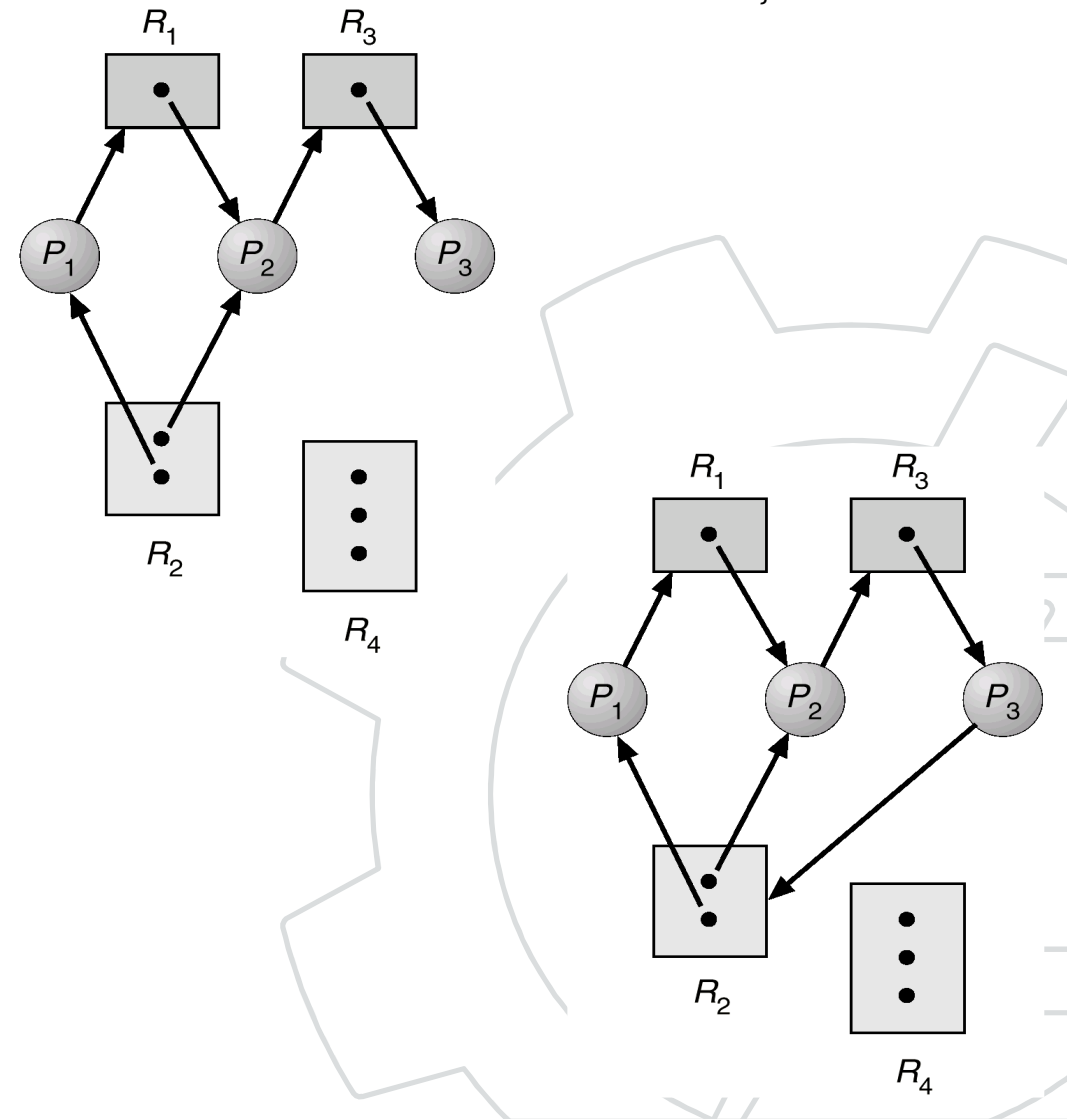




- Se não há ciclos no grafo  
→ **não há** *deadlock*
- Se o grafo contém ciclos, depende:
  - Se recursos só têm uma instância  
→ **há** *deadlock*
    - $P_1 \rightarrow R_1 \rightarrow P_2 \rightarrow R_2 \rightarrow P_1$
  - Se há mais de uma instância  
→ **possível** *deadlock*

## Modelo do sistema

Grafo de alocação de recursos



## Impasse ou *Deadlock*

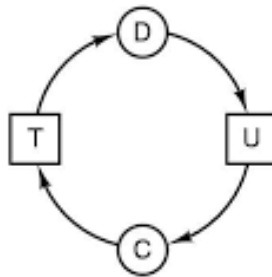
Grafo de alocação de recursos



(a)



(b)

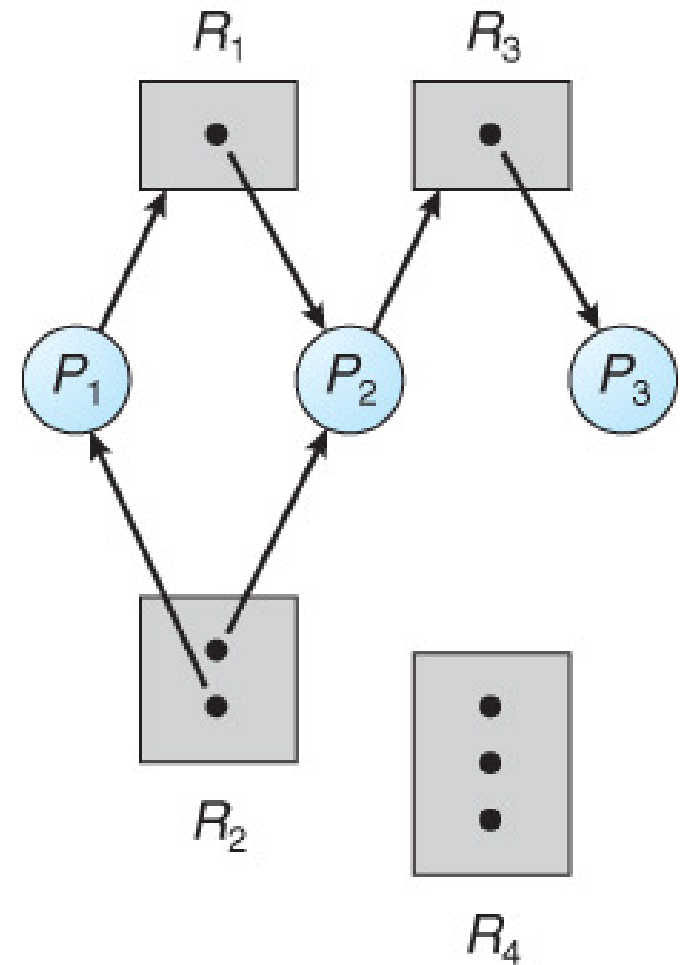


(c)

**(a)** Recurso **R** **alocado** ao **Processo A**

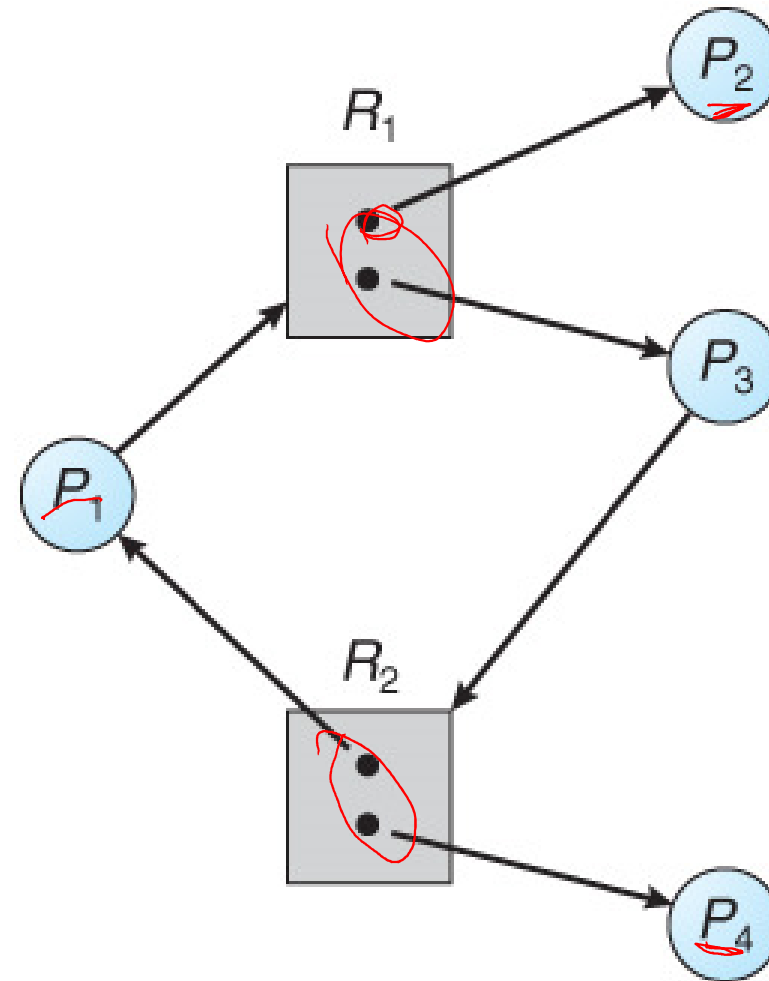
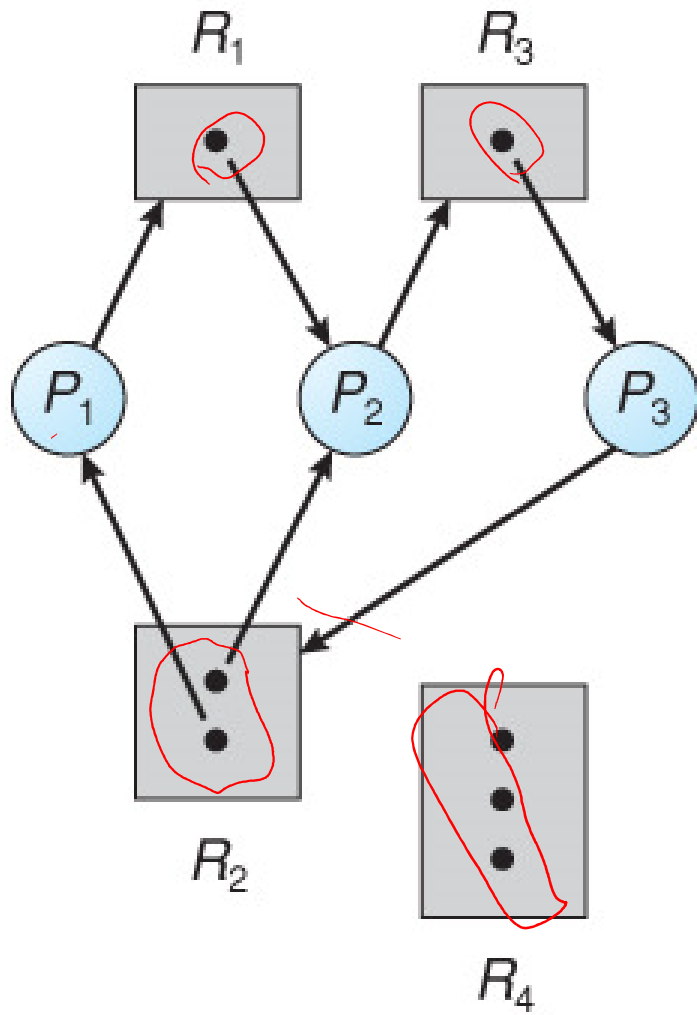
**(b)** Processo **B** **requisita** o **Recurso S**

**(c)** *Deadlock*



## Impasse ou *Deadlock*

Grafo de alocação de recursos



## **Impasse (*Deadlock*)**

Algoritmo de detecção

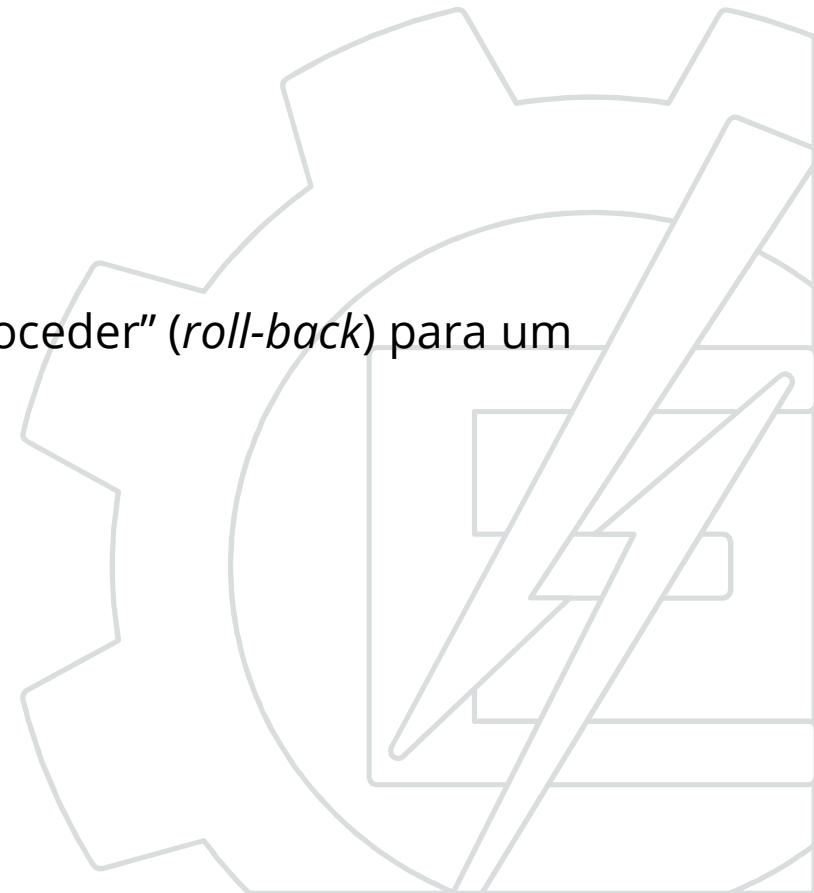
- Considerações a serem feitas:
  - Quão frequentes são os *deadlocks*?
  - Quantos processos são afetados?
- Detecção **frequente**:
  - Pouco tempo de espera
  - Pouca chance de “propagação” do travamento
- Detecção **esporádica**:
  - Menor *overhead* de detecção
  - Pode encontrar muitos ciclos



## Impasse (**Deadlock**)

Recuperação

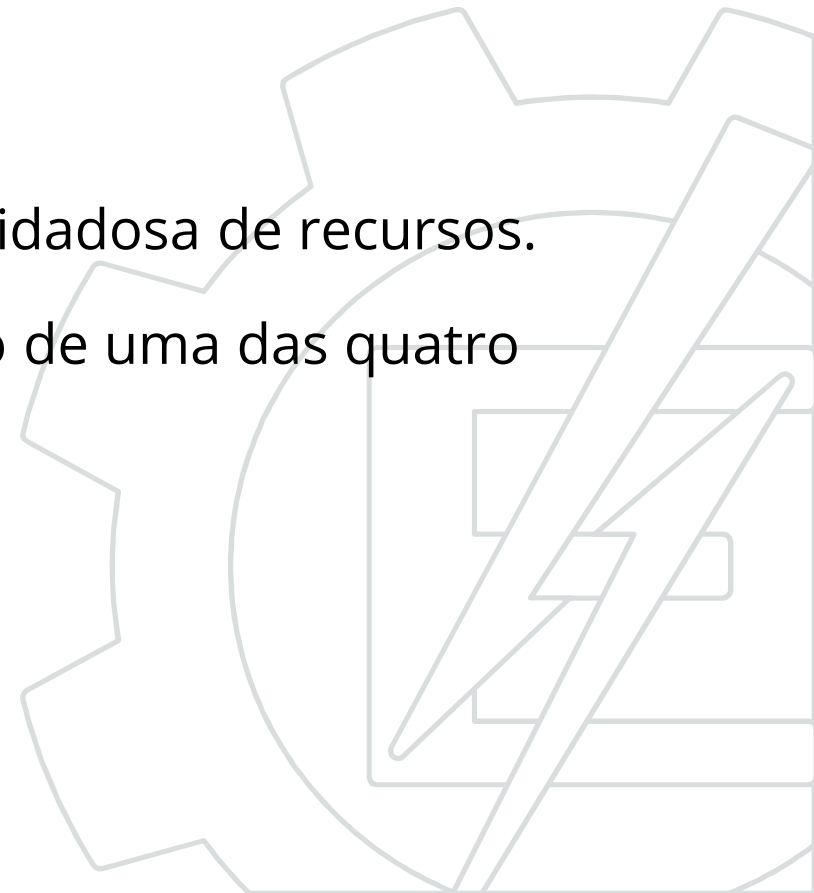
- É preciso quebrar os ciclos no grafo:
  - **Abortando** um ou mais processos
    - Processo termina com erro
    - Estado do sistema pode ficar inconsistente
  - Realizando a **preempção** de recursos
    - Processos que sofrem preempção precisam “retroceder” (*roll-back*) para um ponto anterior
- Considerações:
  - Como escolher o(s) processo(s) vítima(s)?
  - Como distribuir os recursos reclamados?
  - Como evitar a inanição?





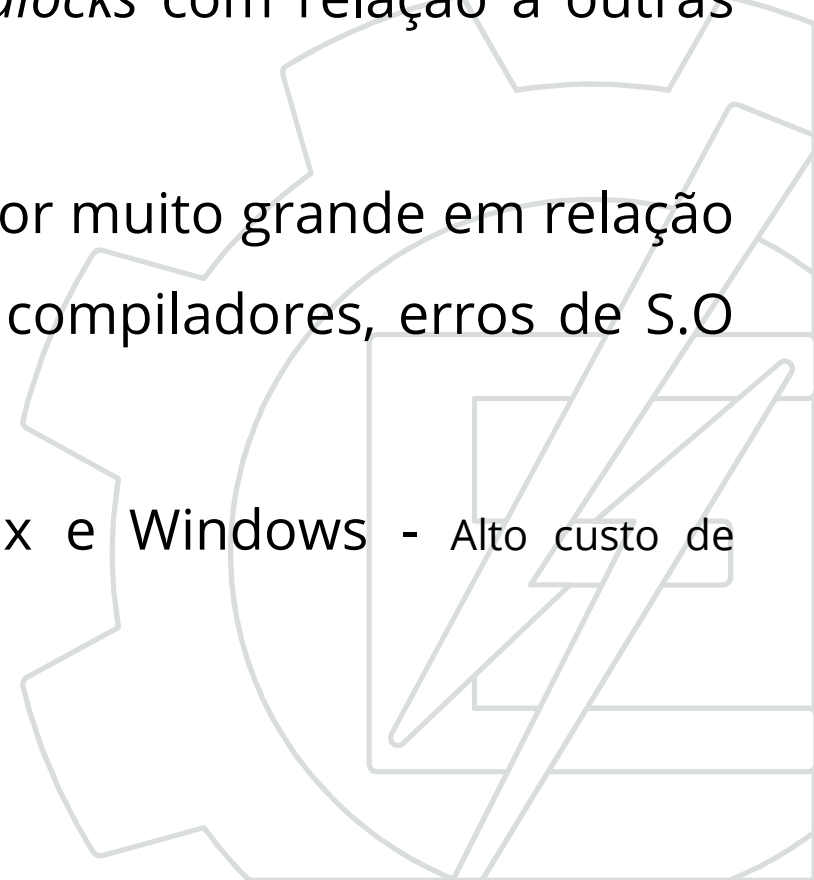
### **Estratégias para prevenção e/ou tratamento de *deadlocks*:**

- 1) Ignorar o problema.
- 2) Detectar e recuperar o problema.
- 3) Evitar dinamicamente o problema – alocação cuidadosa de recursos.
- 4) Prevenir o problema por meio da não-satisfação de uma das quatro condições apresentadas:
  - Exclusão mútua;
  - Posse durante a espera;
  - Inexistência de preempção;
  - Espera circular.



### 1) Ignorar o problema.

- Utilizado se a frequência de ocorrência de *deadlocks* com relação a outras falhas do sistema não é tão expressiva.
- Ignorar se o esforço em solucionar o problema for muito grande em relação à sua frequência. Falhas de *hardware*, erros de compiladores, erros de S.O terão o foco do maior esforço.
- Usado na maioria dos sistemas, inclusive Unix e Windows - Alto custo de tratamento e baixa frequência de ocorrência.



### 2) Detectar e recuperar o problema:

- Permite que os *deadlocks* ocorram. Tenta 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*;
    - Recuperação por meio de eliminação de processos.

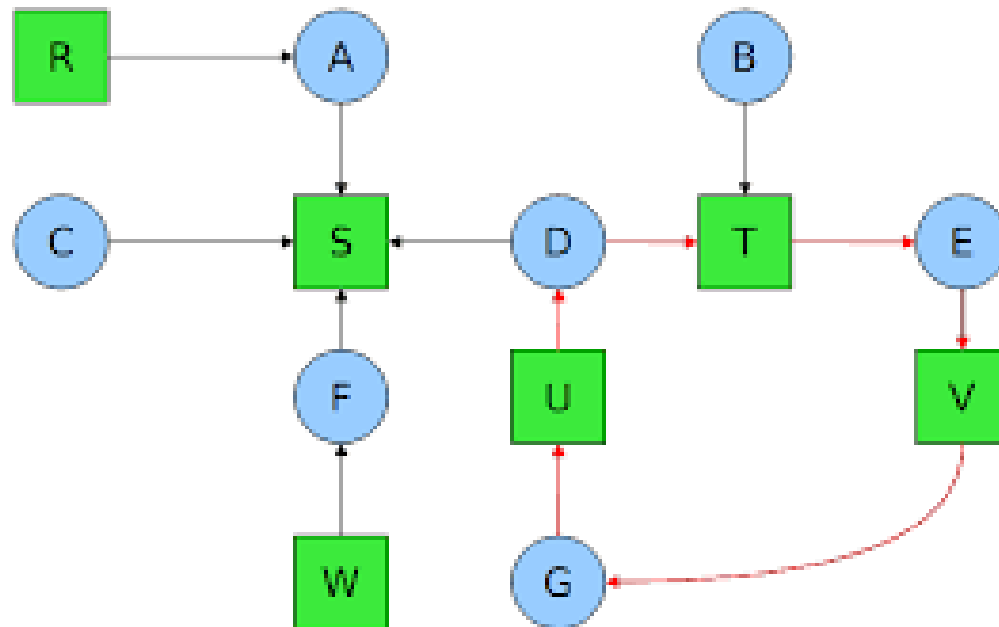


## Impasse (*Deadlock*)

Estratégias

### 2) Detectar e recuperar o problema:

- Detecção com **um recurso** de cada tipo



### 2) Detectar e recuperar o problema:

- Detecção com **um recurso** de cada tipo
  - Se todos os recursos têm apenas uma única instância, então podemos definir um algoritmo de detecção de *deadlocks* que use uma variante do grafo de alocação de recursos, chamado grafo de espera (*waitfor*).
  - Existe um *deadlock* no sistema se e somente se o grafo de espera contiver um ciclo. Para detectar *deadlocks*, o sistema precisa manter o grafo de espera e, periodicamente, invocar um algoritmo que procure um ciclo no grafo.

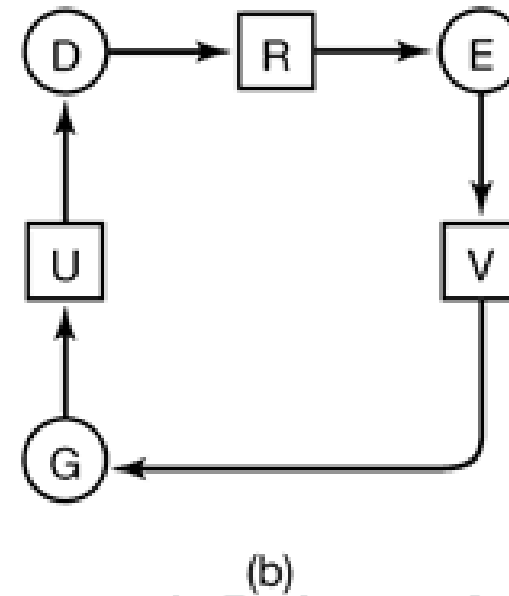
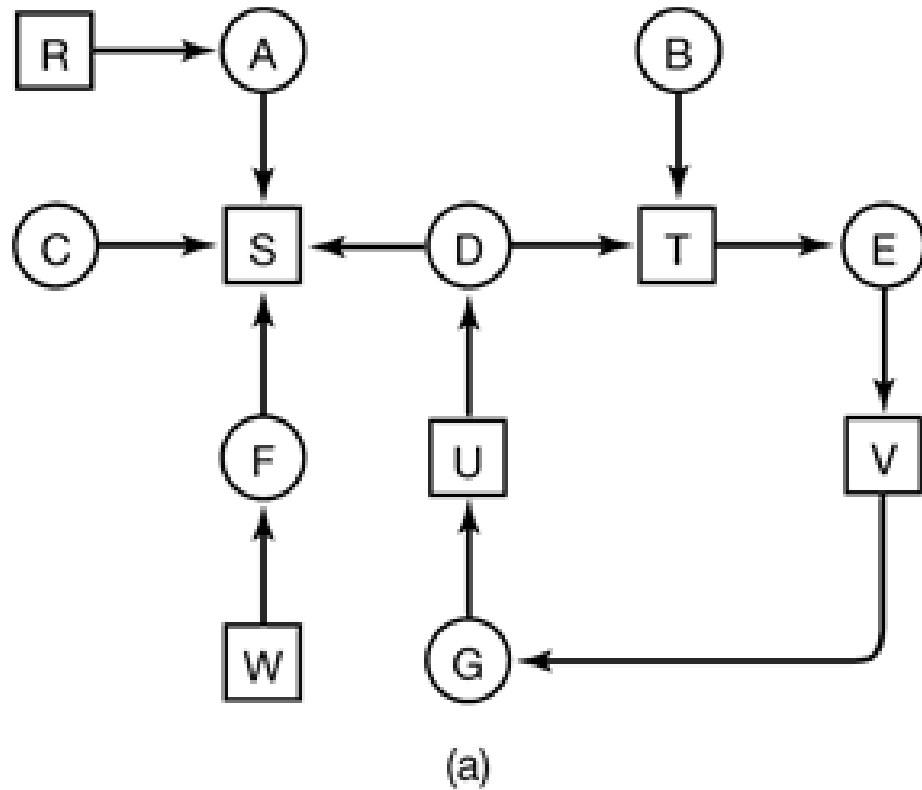


## Impasse (*Deadlock*)

Estratégias

### 2) Detectar e recuperar o problema:

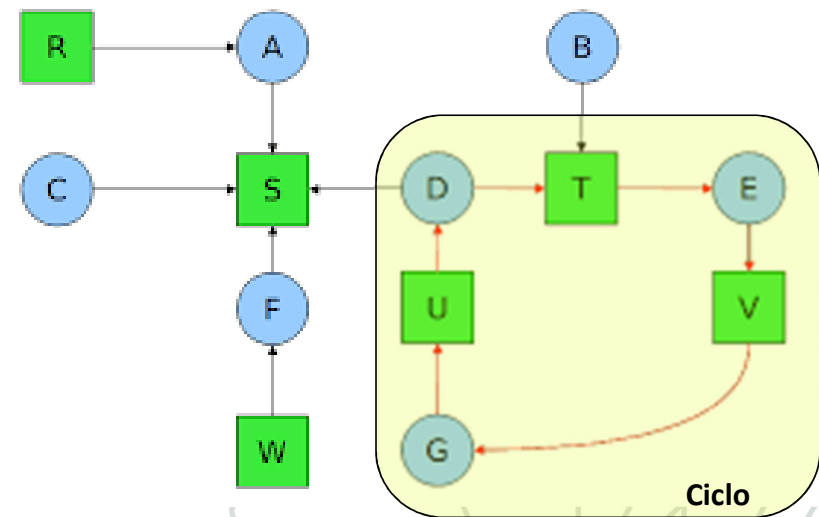
- Detecção com **um recurso** de cada tipo



## Estratégias

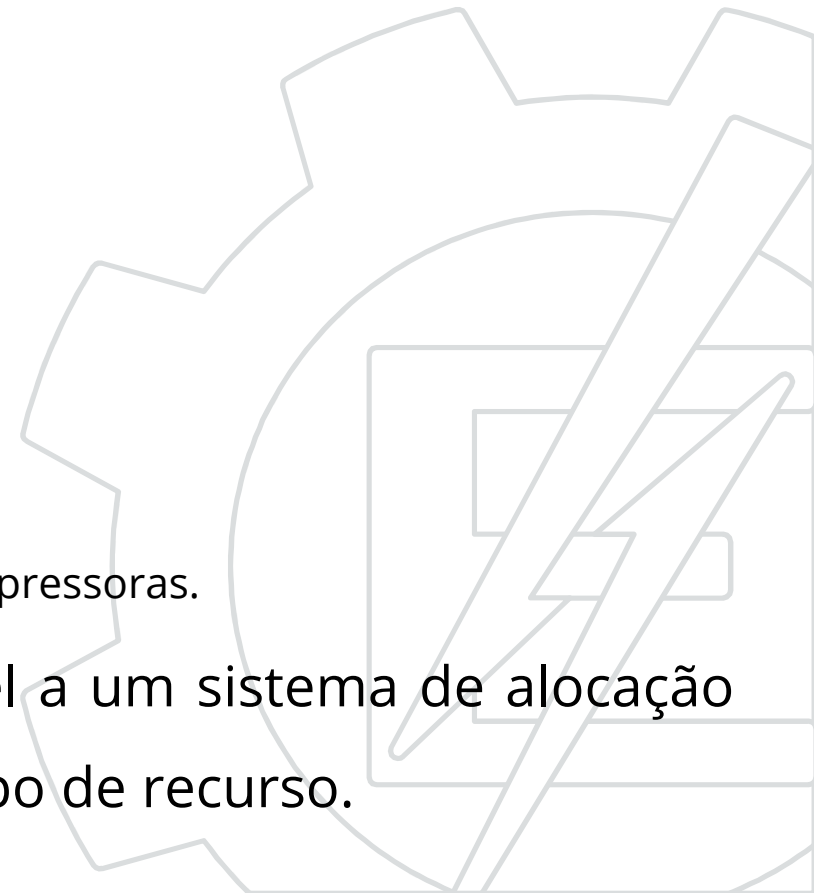
- Detecção com **um recurso** de cada tipo

The diagram illustrates the relationship between system states and safety. A diagonal line divides the space into two regions: a light blue region labeled 'unsafe' and a light gray region labeled 'safe'. A gray rectangular box labeled 'deadlock' is positioned within the 'unsafe' region, indicating that a deadlock state is inherently unsafe.



### 2) Detectar e recuperar o problema:

- Detecção com **vários recursos** de cada tipo:
  - n processos: P1 a Pn
  - M classes diferentes de recursos
  - Usamos um **vetor de recursos existentes - E**
    - Contém o número total de cada recurso existente
    - Ei recursos da classe i ( $1 \leq i \leq m$ )
    - Se a classe 1 for impressora e E1=2, então existem duas impressoras.
  - O esquema do grafo de espera não é aplicável a um sistema de alocação de recursos com múltiplas instâncias de cada tipo de recurso.



### 2) Detectar e recuperar o problema:

- Detecção com **vários recursos** de cada tipo:
  - **Vetor de recursos disponíveis (A)**
    - Quantidade do recurso  $i$  disponível no momento-  $A_i$
    - Se ambas as impressoras estiverem alocadas,  $A_1=0$
  - Duas matrizes
    - **C – matriz de alocação corrente**
      - $C_{ij}$  – número de recurso  $j$  mantido pelo processo  $i$
    - **R – matriz de requisições**
      - $R_{ij}$  – número do recurso  $j$  que o processo  $i$  deseja



## 2) Detectar e recuperar o problema:

- Detecção com **vários recursos** de cada tipo:

Resources in existence  
( $E_1, E_2, E_3, \dots, E_m$ )

Current allocation matrix

$$\begin{bmatrix} C_{11} & C_{12} & C_{13} & \dots & C_{1m} \\ C_{21} & C_{22} & C_{23} & \dots & C_{2m} \\ \vdots & \vdots & \vdots & & \vdots \\ C_{n1} & C_{n2} & C_{n3} & \dots & C_{nm} \end{bmatrix}$$

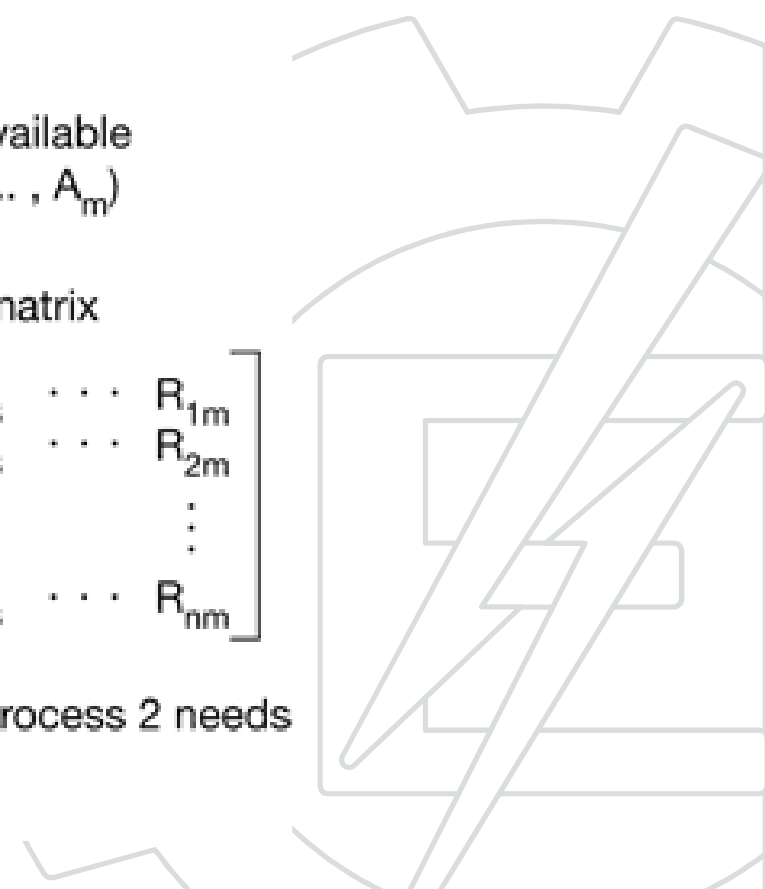
Row n is current allocation  
to process n

Resources available  
( $A_1, A_2, A_3, \dots, A_m$ )

Request matrix

$$\begin{bmatrix} R_{11} & R_{12} & R_{13} & \dots & R_{1m} \\ R_{21} & R_{22} & R_{23} & \dots & R_{2m} \\ \vdots & \vdots & \vdots & & \vdots \\ R_{n1} & R_{n2} & R_{n3} & \dots & R_{nm} \end{bmatrix}$$

Row 2 is what process 2 needs





## Impasse (*Deadlock*)

Estratégias

### 2) Detectar e recuperar o problema:

- Detecção com **vários recursos** de cada tipo:
  - E: Vetor de recursos existentes
  - A: Vetor de recursos disponíveis
  - C: Matriz de alocação
- Três processos:
  - **P1** usa um *scanner*
  - P2 usa duas unidades de fita e uma de CD-ROM
  - **P3** usa um plotter e dois *scanners*
  - Cada processo pode precisar de outros recursos

$$\begin{array}{c} \text{Tape drives} \\ \text{Plotters} \\ \text{Scanners} \\ \text{CD Roms} \end{array} \\ E = (4 \quad 2 \quad 3 \quad 1)$$

$$\begin{array}{c} \text{Tape drives} \\ \text{Plotters} \\ \text{Scanners} \\ \text{CD Roms} \end{array} \\ A = (2 \quad 1 \quad 0 \quad 0)$$

Current allocation matrix

$$C = \begin{array}{|c|c|c|c|c|} \hline 0 & 0 & 1 & 0 \\ \hline 2 & 0 & 0 & 1 \\ \hline 0 & 1 & 2 & 0 \\ \hline \end{array}$$

## Impasse (*Deadlock*)

Estratégias

### 2) Detectar e recuperar o problema:

- Detecção com **vários recursos** de cada tipo:
  - E: Vetor de recursos existentes
  - A: Vetor de recursos disponíveis
  - C: Matriz de alocação
- Três processos:
  - **P1** usa um scanner
  - P2 usa duas unidades de fita e uma de CD-ROM
  - **P3** usa um plotter e dois scanners
  - Cada processo pode precisar de outros recursos

$$\begin{array}{c} \text{Tape drives} \\ \text{Plotters} \\ \text{Scanners} \\ \text{CD Roms} \end{array} \\ E = (4 \quad 2 \quad 3 \quad 1)$$

$$\begin{array}{c} \text{Tape drives} \\ \text{Plotters} \\ \text{Scanners} \\ \text{CD Roms} \end{array} \\ A = (2 \quad 1 \quad 0 \quad 0)$$

Current allocation matrix

$$C = \begin{bmatrix} 0 & 0 & 1 & 0 \\ 2 & 0 & 0 & 1 \\ 0 & 1 & 2 & 0 \end{bmatrix}$$

Request matrix

$$R = \begin{bmatrix} 2 & 0 & 0 & 1 \\ 1 & 0 & 1 & 0 \\ 2 & 1 & 0 & 0 \end{bmatrix}$$

## Impasse (*Deadlock*)

Estratégias

### 2) Detectar e recuperar o problema:

- Detecção com **vários recursos** de cada tipo:

$$\begin{array}{c} \text{Tape drives} \\ \text{Plotters} \\ \text{Scanners} \\ \text{CD Roms} \end{array} \\ E = (4 \quad 2 \quad 3 \quad 1)$$

$$\begin{array}{c} \text{Tape drives} \\ \text{Plotters} \\ \text{Scanners} \\ \text{CD Roms} \end{array} \\ A = (2 \quad 1 \quad 0 \quad 0)$$

Comparamos cada processo em R com os recursos em A, de modo a encontrar um que possa ser executado. Qual processo pode ser concluído?

Current allocation matrix

$$C = \begin{bmatrix} 0 & 0 & 1 & 0 \\ 2 & 0 & 0 & 1 \\ 0 & 1 & 2 & 0 \end{bmatrix}$$

Request matrix

$$R = \begin{bmatrix} 2 & 0 & 0 & 1 \\ 1 & 0 & 1 & 0 \\ 2 & 1 & 0 & 0 \end{bmatrix}$$

## Impasse (*Deadlock*)

Estratégias

### 2) Detectar e recuperar o problema:

- Detecção com **vários recursos** de cada tipo:

$$\begin{array}{c} \text{Tape drives} \\ \text{Plotters} \\ \text{Scanners} \\ \text{CD Roms} \end{array} \\ E = (4 \quad 2 \quad 3 \quad 1)$$

$$\begin{array}{c} \text{Tape drives} \\ \text{Plotters} \\ \text{Scanners} \\ \text{CD Roms} \end{array} \\ A = (2 \quad 1 \quad 0 \quad 0)$$

Comparamos cada processo em R com os recursos em A, de modo a encontrar um que possa ser executado. Qual processo pode ser concluído?

Current allocation matrix

$$C = \begin{bmatrix} 0 & 0 & 1 & 0 \\ 2 & 0 & 0 & 1 \\ 0 & 1 & 2 & 0 \end{bmatrix}$$

Request matrix

$$R = \begin{bmatrix} 2 & 0 & 0 & 1 \\ 1 & 0 & 1 & 0 \\ 2 & 1 & 0 & 0 \end{bmatrix}$$



## Impasse (*Deadlock*)

Estratégias

### 2) Detectar e recuperar o problema:

- Detecção com **vários recursos** de cada tipo:

$$\begin{array}{c} \text{Tape drives} \\ \text{Plotters} \\ \text{Scanners} \\ \text{CD Roms} \end{array} \\ E = (4 \quad 2 \quad 3 \quad 1)$$

$$\begin{array}{c} \text{Tape drives} \\ \text{Plotters} \\ \text{Scanners} \\ \text{CD Roms} \end{array} \\ A = (2 \quad 1 \quad 0 \quad 0)$$

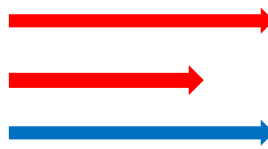
Comparamos cada processo em R com os recursos em A, de modo a encontrar um que possa ser executado. Qual processo pode ser concluído?

Current allocation matrix

$$C = \begin{bmatrix} 0 & 0 & 1 & 0 \\ 2 & 0 & 0 & 1 \\ 0 & 1 & 2 & 0 \end{bmatrix}$$

Request matrix

$$R = \begin{bmatrix} 2 & 0 & 0 & 1 \\ 1 & 0 & 1 & 0 \\ 2 & 1 & 0 & 0 \end{bmatrix}$$



## Impasse (*Deadlock*)

Estratégias

### 2) Detectar e recuperar o problema:

- Detecção com **vários recursos** de cada tipo:

$$\begin{array}{c} \text{Tape drives} \\ \text{Plotters} \\ \text{Scanners} \\ \text{CD Roms} \end{array} \\ E = (4 \quad 2 \quad 3 \quad 1)$$

$$\begin{array}{c} \text{Tape drives} \\ \text{Plotters} \\ \text{Scanners} \\ \text{CD Roms} \end{array} \\ A = (2 \quad 1 \quad 0 \quad 0)$$

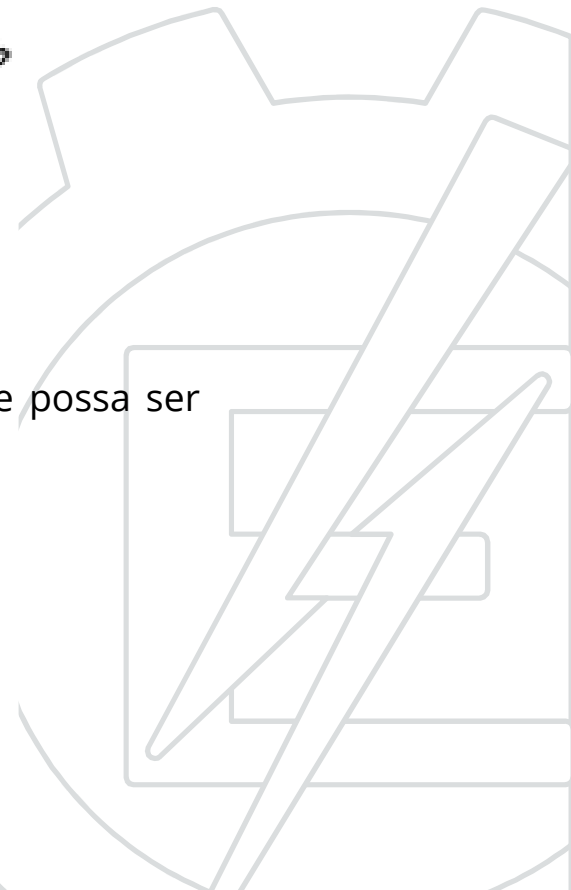
Comparamos cada processo em R com os recursos em A, de modo a encontrar um que possa ser executado. Qual processo pode ser concluído?

Current allocation matrix

$$C = \begin{bmatrix} 0 & 0 & 1 & 0 \\ 2 & 0 & 0 & 1 \\ 0 & 1 & 2 & 0 \end{bmatrix}$$

Request matrix

$$R = \begin{bmatrix} 2 & 0 & 0 & 1 \\ 1 & 0 & 1 & 0 \\ 2 & 1 & 0 & 0 \end{bmatrix}$$



## Impasse (*Deadlock*)

Estratégias

### 2) Detectar e recuperar o problema:

- Detecção com **vários recursos** de cada tipo:

$$\begin{array}{c} \text{Tape drives} \\ \text{Plotters} \\ \text{Scanners} \\ \text{CD Roms} \end{array}$$
$$E = (4 \quad 2 \quad 3 \quad 1)$$

$$\begin{array}{c} \text{Tape drives} \\ \text{Plotters} \\ \text{Scanners} \\ \text{CD Roms} \end{array}$$
$$A = ( \textcolor{red}{2} \quad \textcolor{red}{2} \quad \textcolor{red}{2} \quad 0 )$$

Após o término do processo P3:

Current allocation matrix

$$C = \begin{bmatrix} 0 & 0 & 1 & 0 \\ 2 & 0 & 0 & 1 \\ 0 & \textcolor{red}{0} & \textcolor{red}{0} & 0 \end{bmatrix}$$

Request matrix

$$R = \begin{bmatrix} 2 & 0 & 0 & 1 \\ 1 & 0 & 1 & 0 \\ \textcolor{red}{0} & \textcolor{red}{0} & 0 & 0 \end{bmatrix}$$



# Impasse (*Deadlock*)

Estratégias

## 2) Detectar e recuperar o problema:

- Detecção com **vários recursos** de cada tipo:

$$E = (4 \quad 2 \quad 3 \quad 1)$$

Tape drives  
Plotters  
Scanners  
CD Roms

$$A = (2 \quad 2 \quad 2 \quad 0)$$

Tape drives  
Plotters  
Scanners  
CD Roms

Seleção do próximo processo a ser executado:

- P1 necessita de recursos que não estão disponíveis.
- P2 pode ser executado

Current allocation matrix

$$C = \begin{bmatrix} 0 & 0 & 1 & 0 \\ 2 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \end{bmatrix}$$

Request matrix

$$R = \begin{bmatrix} 2 & 0 & 0 & 1 \\ 1 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix}$$





## Impasse (*Deadlock*)

Estratégias

### 2) Detectar e recuperar o problema:

- Detecção com **vários recursos** de cada tipo:

$$\begin{array}{c} \text{Tape drives} \\ \text{Plotters} \\ \text{Scanners} \\ \text{CD Roms} \end{array} \\ E = ( 4 \quad 2 \quad 3 \quad 1 )$$

$$\begin{array}{c} \text{Tape drives} \\ \text{Plotters} \\ \text{Scanners} \\ \text{CD Roms} \end{array} \\ A = ( 4 \quad 2 \quad 2 \quad 1 )$$

Após a execução de P2:

Current allocation matrix

$$C = \begin{bmatrix} 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix}$$

Request matrix

$$R = \begin{bmatrix} 2 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix}$$



## Impasse (*Deadlock*)

Estratégias

### 2) Detectar e recuperar o problema:

- Detecção com **vários recursos** de cada tipo:

$$\begin{array}{c} \text{Tape drives} \\ \text{Plotters} \\ \text{Scanners} \\ \text{CD Roms} \end{array} \\ E = ( 4 \quad 2 \quad 3 \quad 1 )$$

$$\begin{array}{c} \text{Tape drives} \\ \text{Plotters} \\ \text{Scanners} \\ \text{CD Roms} \end{array} \\ A = ( 4 \quad 2 \quad 3 \quad 1 )$$

Após a execução de P1:

Current allocation matrix

$$C = \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix}$$

Request matrix

$$R = \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix}$$

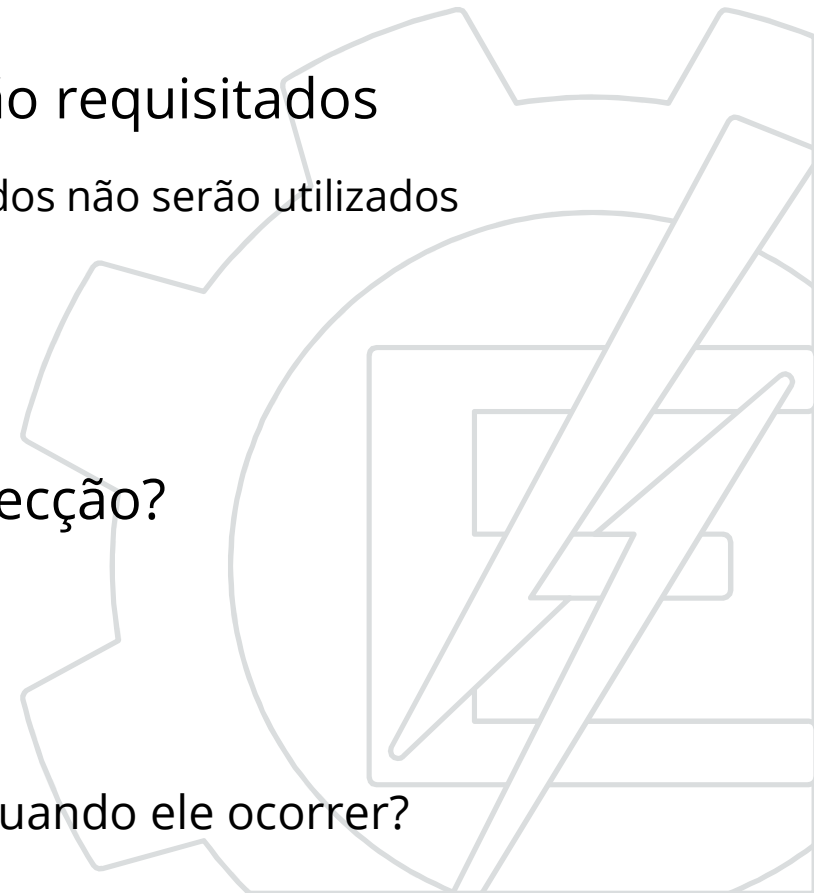


### 2) Detectar e recuperar o problema:

- Detecção com **vários recursos** de cada tipo:
  - Deve-se saber de antemão que recursos serão requisitados
    - Processos podem ser abortados e os recursos requisitados não serão utilizados
    - É um cenário desafiador
- Quando devemos invocar o algoritmo de detecção?

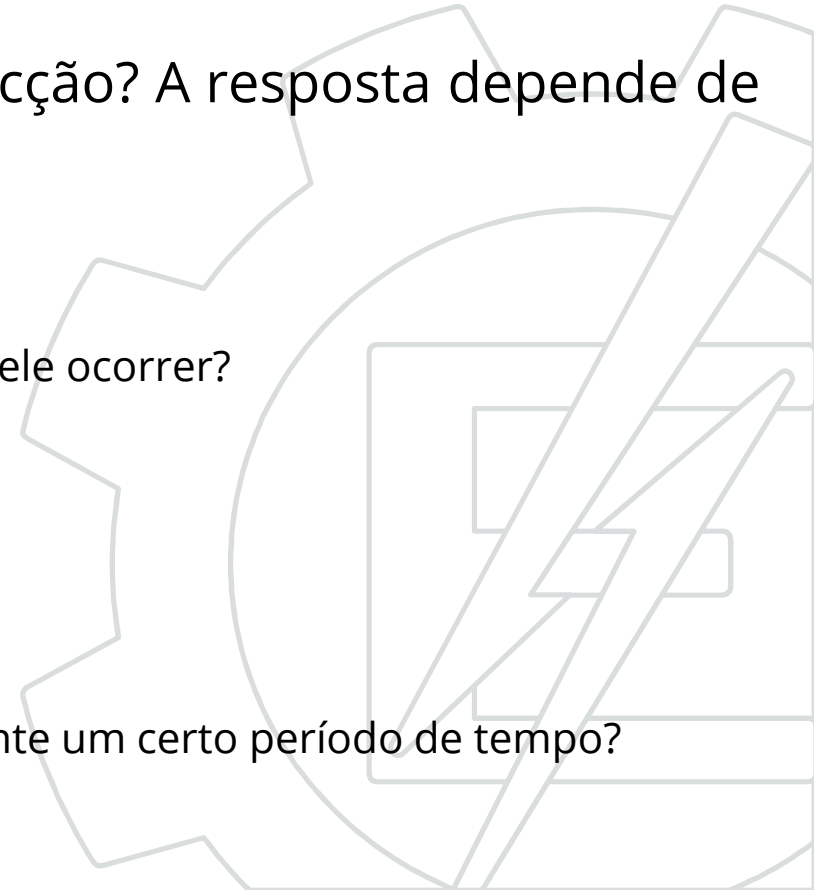
A resposta depende de dois fatores:

- Com que frequência um *deadlock* pode ocorrer?
- Quantos processos serão afetados pelo *deadlock* quando ele ocorrer?



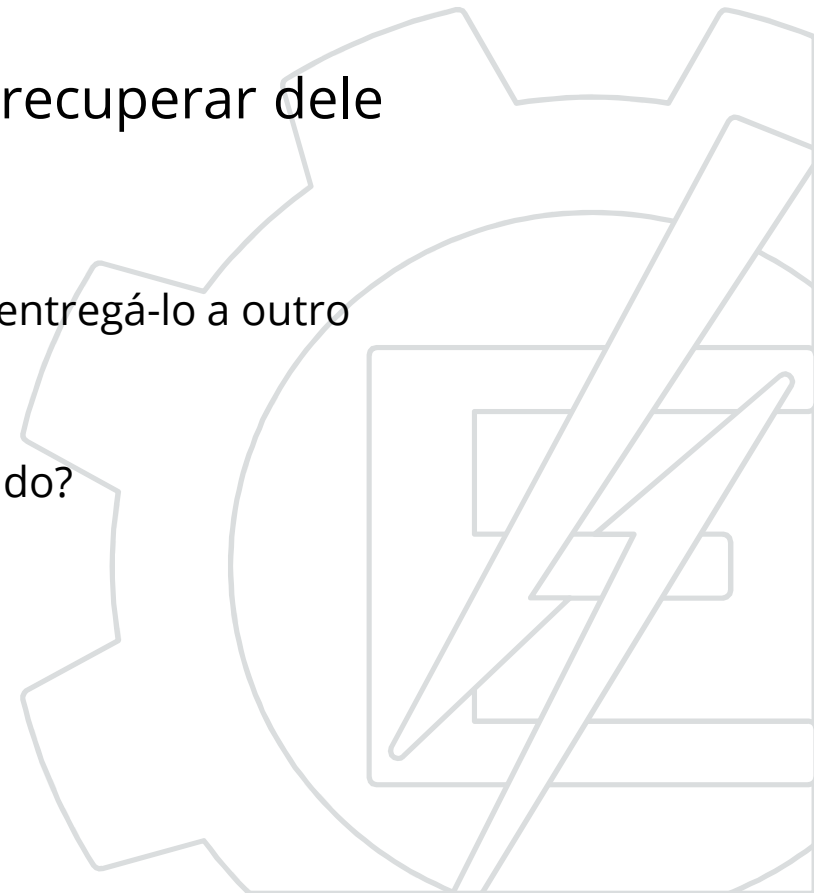
### 2) Detectar e recuperar o problema:

- Detecção com **vários recursos** de cada tipo:
  - Quando devemos invocar o algoritmo de detecção? A resposta depende de dois fatores:
    - Com que frequência um *deadlock* pode ocorrer?
    - Quantos processos serão afetados pelo *deadlock* quando ele ocorrer?
  - Quando devemos procurar por *deadlocks*?
    - a) Toda vez que uma requisição é realizada?
    - b) A cada k minutos?
    - c) Quando a utilização da CPU cai abaixo de um valor durante um certo período de tempo?



### 2) Detectar e recuperar o problema:

- Detecção com **vários recursos** de cada tipo:
  - Após a detecção do *deadlock* é necessário se recuperar dele
- a) Por meio de preempção
  - Retirar temporariamente um recurso de um processo e entregá-lo a outro
  - Depende da natureza do recurso
  - Que processo deverá receber o recurso que foi desalocado?
  - Frequentemente muito difícil – gera prejuízos

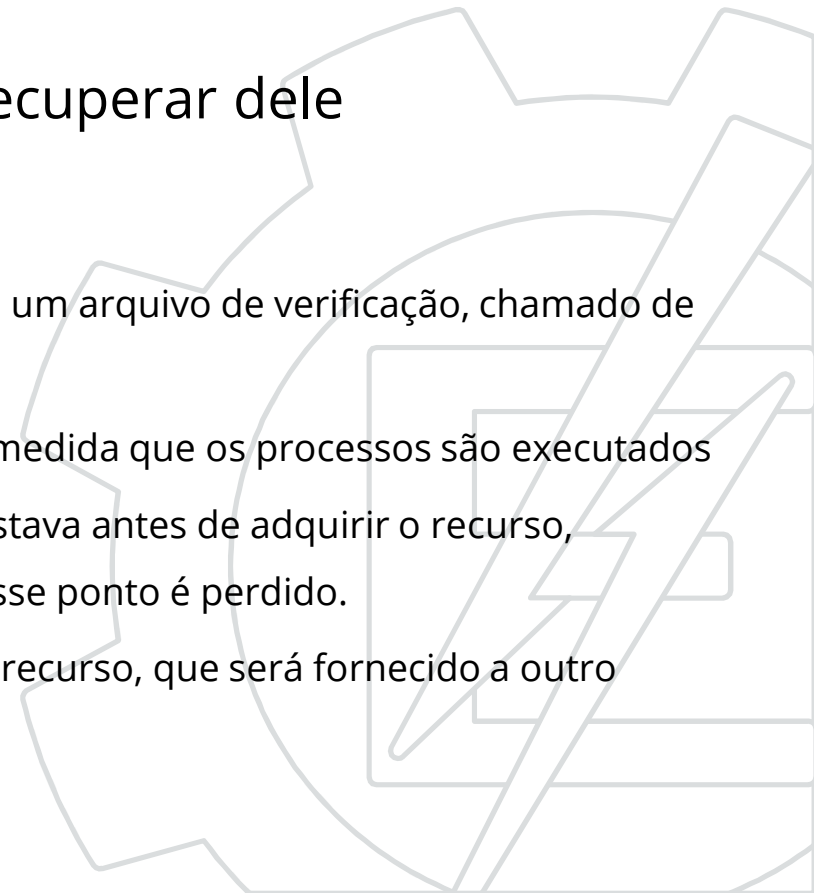


### 2) Detectar e recuperar o problema:

- Detecção com **vários recursos** de cada tipo:
  - Após a detecção do *deadlock* é necessário se recuperar dele

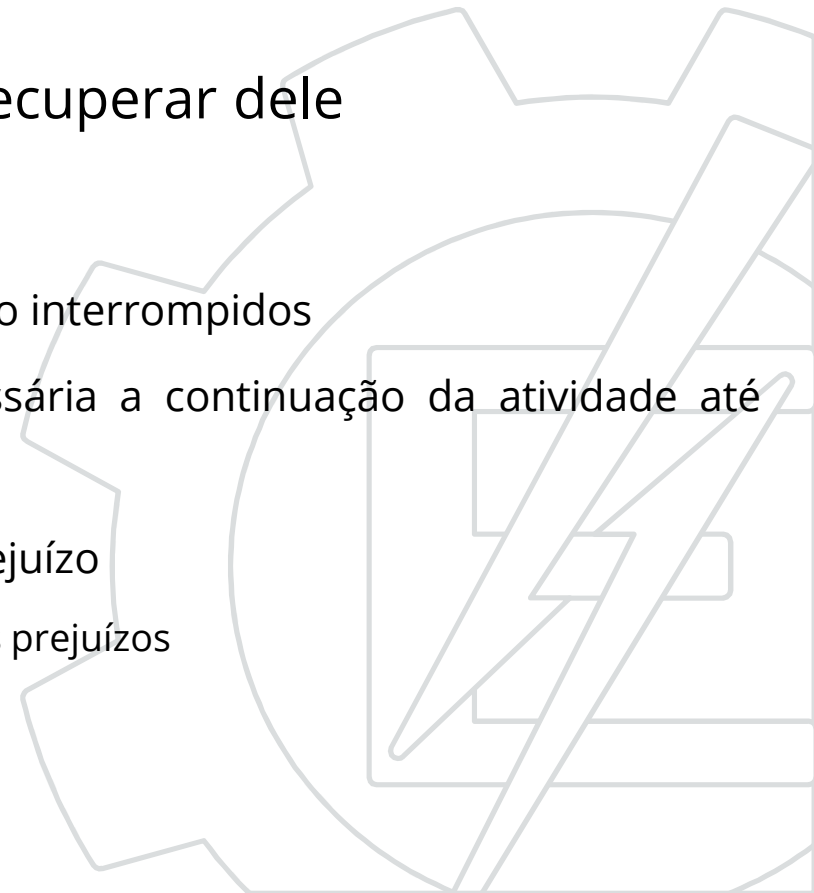
#### b) Por meio de *Rollback*

- O estado de cada processo e o uso de recursos é armazenado em um arquivo de verificação, chamado de *checkpoint file*.
- Novas verificações e informações são armazenadas ao arquivo à medida que os processos são executados
- Quando ocorre um *deadlock*, o processo volta ao ponto em que estava antes de adquirir o recurso, utilizando o *checkpoint file* apropriado. Todo trabalho feito após esse ponto é perdido.
  - O processo retrocede a um momento em que não possuía o recurso, que será fornecido a outro processo.



### 2) Detectar e recuperar o problema:

- Detecção com **vários recursos** de cada tipo:
  - Após a detecção do *deadlock* é necessário se recuperar dele
- c) Por eliminação do processo
  - Um ou mais processos que estão no ciclo com *deadlock* são interrompidos
  - Não garante a eliminação de *deadlock*. Pode ser necessária a continuação da atividade até quebrar o ciclo
  - Melhor solução para processos que não causam muito prejuízo
    - Exemplo 1: Compilação – sem problemas, não causa grandes prejuízos
    - Exemplo 2: Atualização de base de dados - problemas



### 3) Evitar dinamicamente o problema:

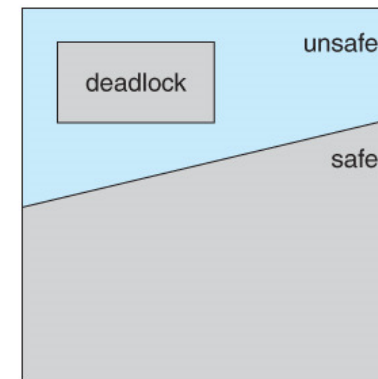
- Alocação de recursos na medida em que se fazem necessários (*runtime*)
- Escalonamento cuidadoso – alto custo
- Prévio conhecimento dos recursos que serão utilizados
- Algoritmo do Banqueiro
  - Para um único tipo de recurso
  - Para vários tipos de recursos
- Utilizam a noção de Estados Seguros e Inseguros





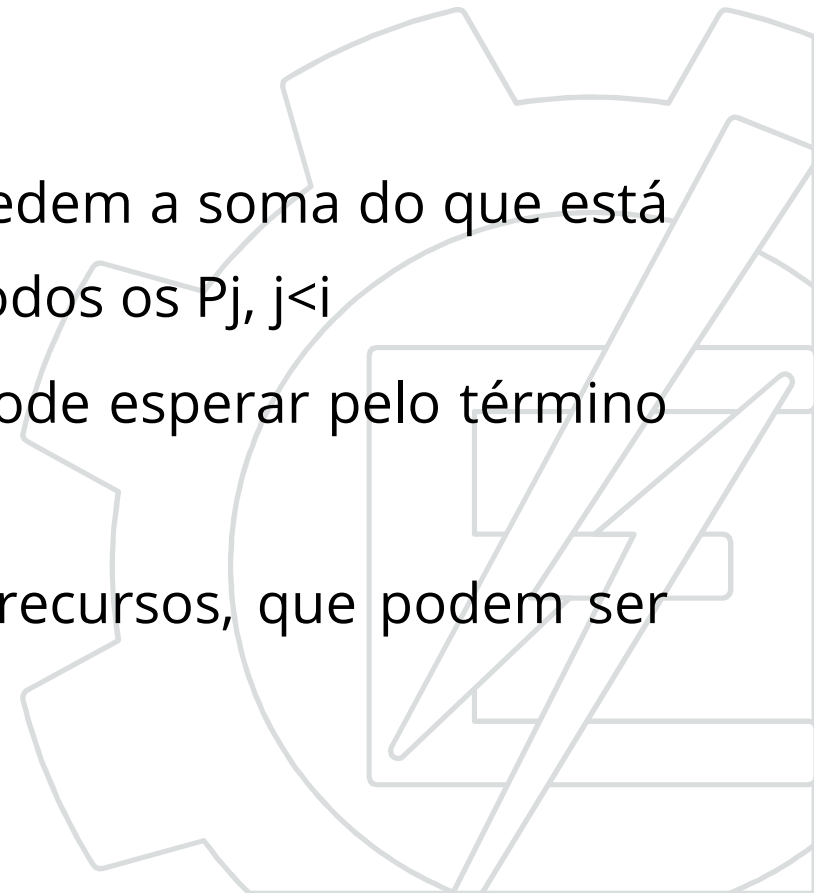
### 3) Evitar dinamicamente o problema:

- Estados Seguros: não provocam *deadlocks*
  - Há uma maneira de atender a todas as requisições
  - A partir de um estado seguro existe a garantia de que os processos terminarão sua execução.
- Estados Inseguros: podem provocar *deadlocks*
  - Não necessariamente provocam/ocorrem *deadlocks*
  - A partir de um estado inseguro não é possível garantir que os processos terminarão corretamente sua execução.



- **Estado Seguro**

- Uma seq.  $\langle P_1, P_2, \dots, P_n \rangle$  é segura se para  $P_i$ :
  - Os recursos que  $P_i$  pode requisitar não excedem a soma do que está disponível com as demandas máximas de todos os  $P_j, j < i$
  - Se recursos não estiverem disponíveis,  $P_i$  pode esperar pelo término de todos os  $P_j$
  - Quando  $P_i$  terminar, ele liberará todos os recursos, que podem ser usados por  $P_{i+1}$



## •Estado Seguro

- Exemplo: 12 acionadores de fitas, processos P0, P1, P2:

	P0	P1	P2
Necessidade máxima	10	4	9
Necessidade atual	5	2	2

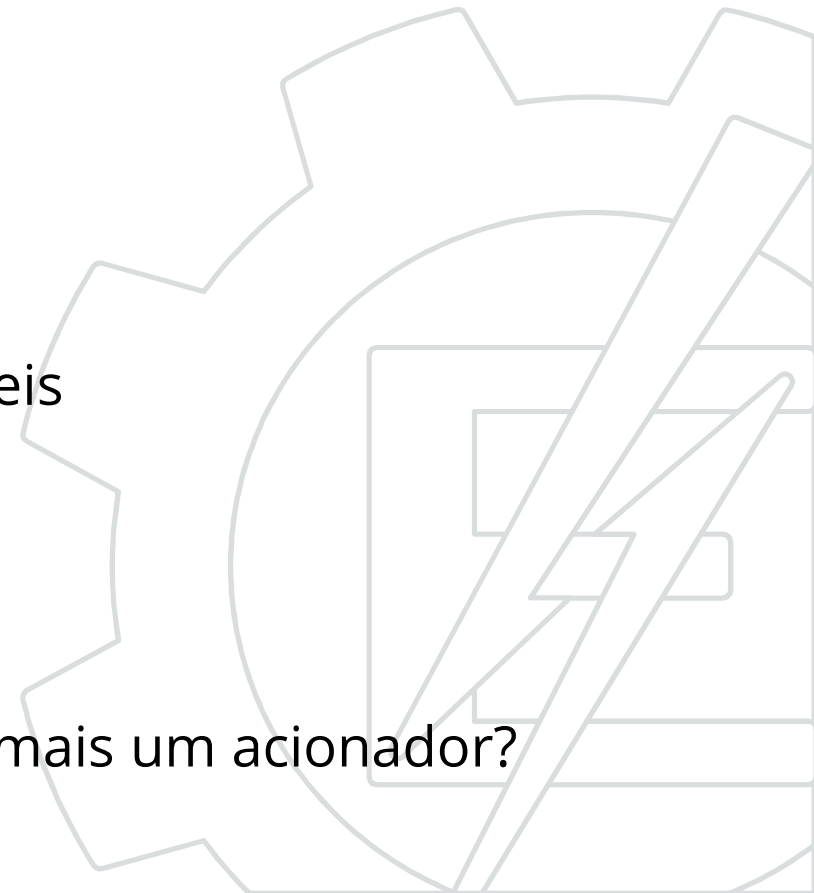
- Atualmente, **há somente 3** acionadores disponíveis:
  - $12 - (5+2+2)$
- Estado seguro:**  $\langle P1, P0, P2 \rangle$ 
  - P1 pode pedir no máximo mais  $2 \leq 3$
  - P0 pode pedir no máximo mais  $5 \leq 3 + 2$
  - P2 pode pedir no máximo mais  $7 \leq 3 + 2 + 5$
- O que aconteceria se o S.O. alocasse P0 ou P2 antes de P1?
  - $\langle P0, \dots \rangle$  ou  $\langle P2, \dots \rangle$  são estados inseguros e que podem gerar deadlock

## •Estado Seguro?

- Exemplo: 12 acionadores de fitas, 3 processos

	P0	P1	P2
Necessidade máxima	10	4	9
Necessidade atual	5	2	2

- Estado seguro:  $\langle P1, P0, P2 \rangle$ 
  - Há  $12 - (5+2+2) = 3$  acionadores disponíveis
  - P1 pode pedir no máximo mais 2
  - P0 pode pedir no máximo  $5 = 3 + 2$
  - P2 pode pedir no máximo  $7 < 5 + 2 + 3$
- O que acontece se P2 requisitar (e receber) mais um acionador?

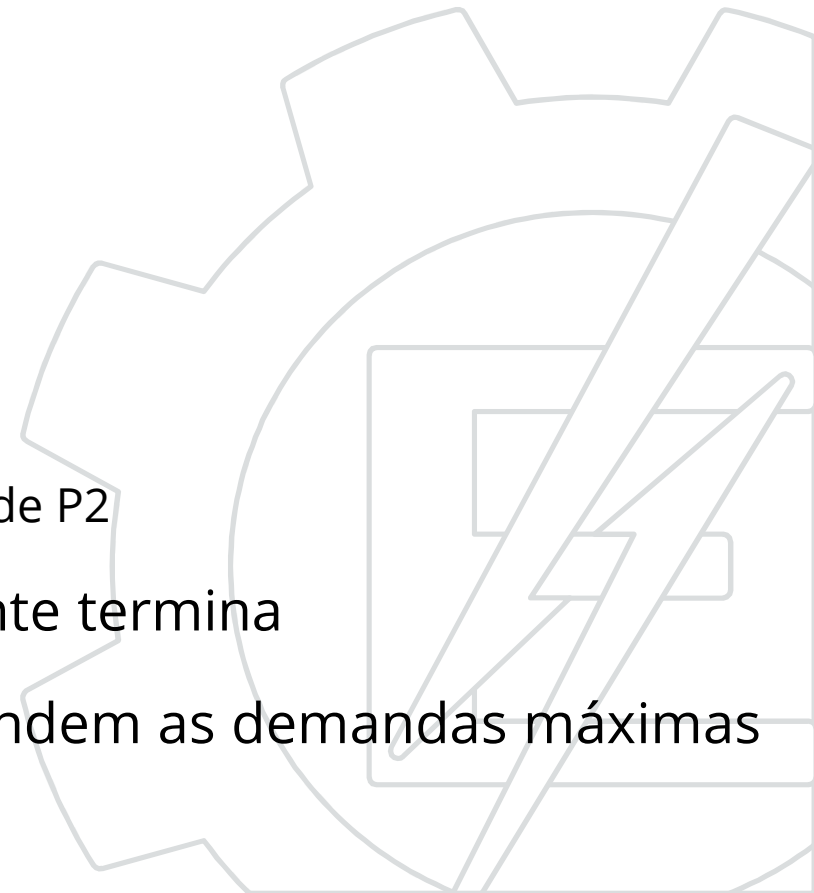


### •Estado Inseguro

- Exemplo: 12 acionadores de fitas, 3 processos

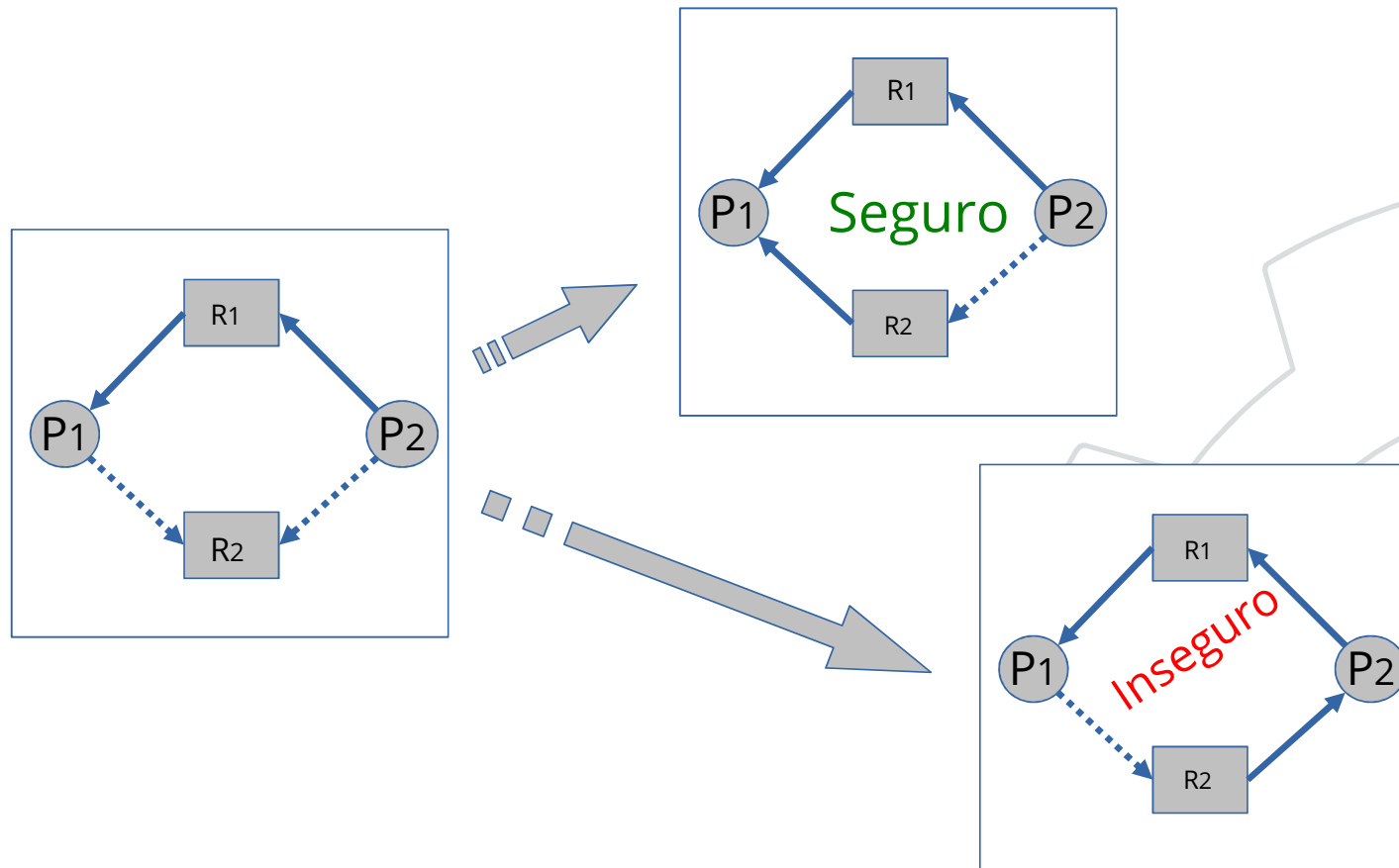
	P0	P1	P2
Necessidade máxima	10	4	9
Necessidade atual	5	2	3

- Estado inseguro
  - Há apenas dois acionadores disponíveis
    - Isso não atende a demanda máxima de P0, nem de P2
  - P1 só pode pedir mais 2, então eventualmente termina
  - Restariam 4 acionadores livres, que não atendem as demandas máximas de P0 nem de P2



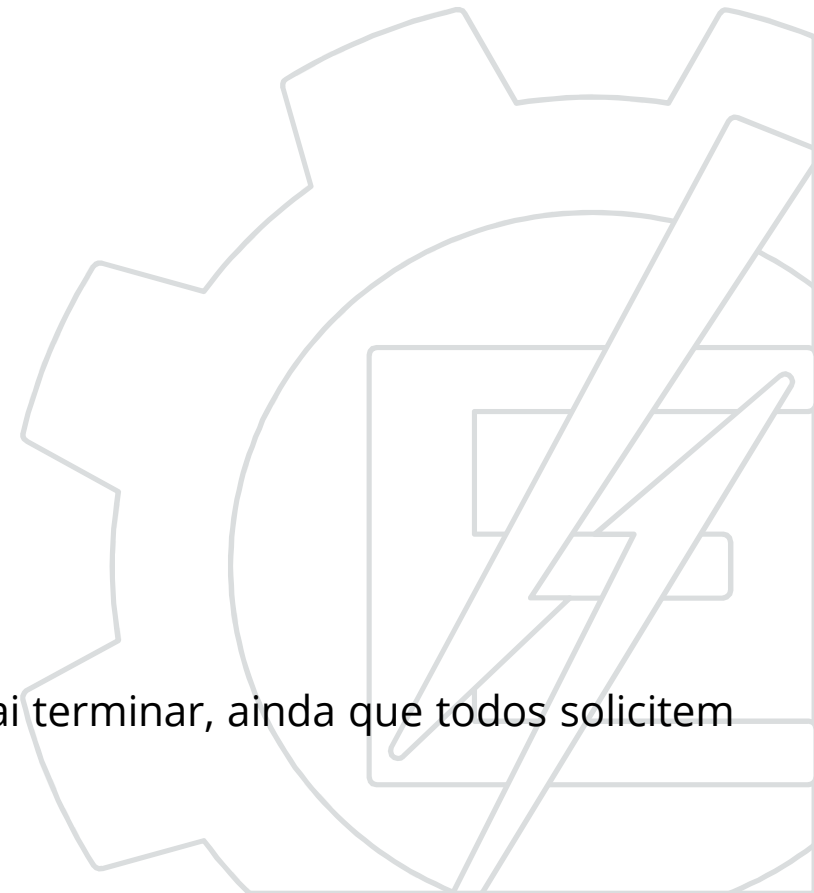
# Impasse (*Deadlock*)

Estratégias



### 3) Evitar dinamicamente o problema:

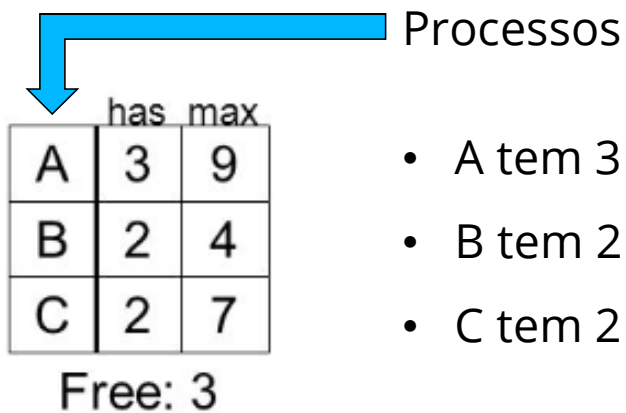
- Mecanismos:
  - Utiliza as mesmas estruturas de detecção com vários recursos.
  - Um estado consiste no conjunto de estruturas:
    - E (*existing*);
    - A (*available*);
    - C (*current*); e
    - R (*requirement*).
- Estado Seguro:
  - Aquele no qual existe alguma ordem em que todo processo vai terminar, ainda que todos solicitem seu número máximo de recursos.



## Impasse (*Deadlock*)

Estratégias

### 3) Evitar dinamicamente o problema:

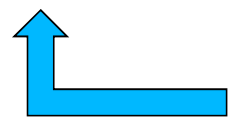


	has	max
A	3	9
B	2	4
C	2	7

Free: 3

- A tem 3 instâncias, mas pode precisar de até 9
- B tem 2 instâncias e pode precisar de 4
- C tem 2 instâncias e pode precisar de 7

Há um total de 10 instâncias do recurso.



Diante da primeira distribuição, sobram 3 recursos.



## Impasse (*Deadlock*)

Estratégias

### 3) Evitar dinamicamente o problema:

- Este estado é seguro ou inseguro?

	has	max
A	3	9
B	2	4
C	2	7

Free: 3



## Impasse (*Deadlock*)


Estratégias

### 3) Evitar dinamicamente o problema:

- Este estado é seguro ou inseguro?

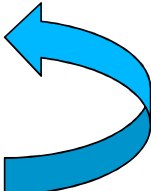
	has	max
A	3	9
B	2	4
C	2	7

Free: 3



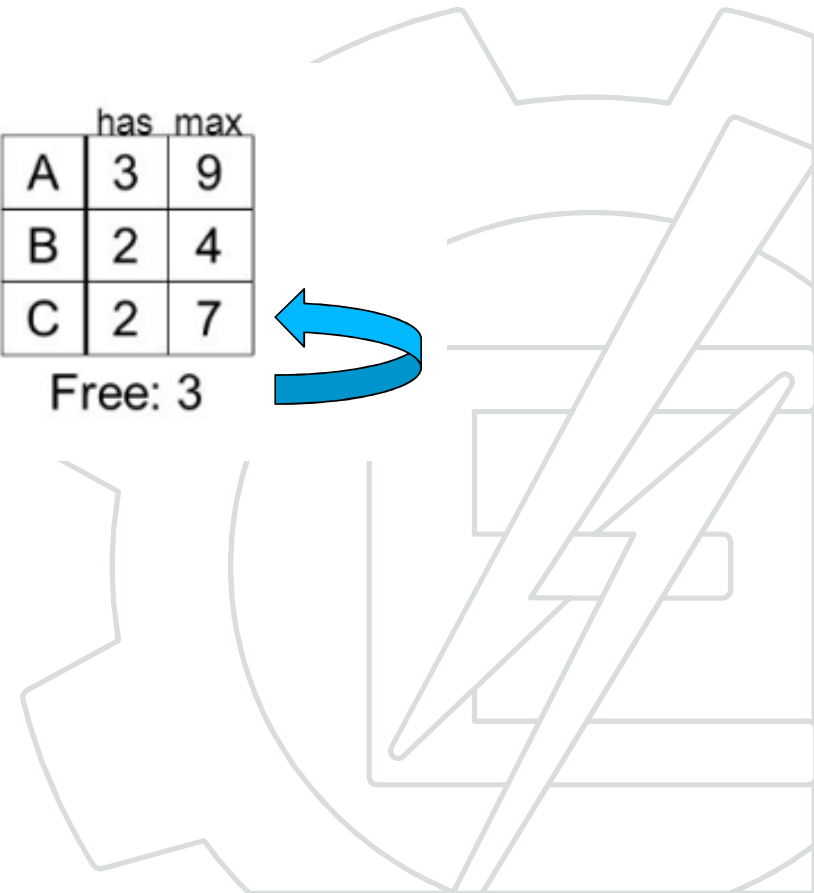
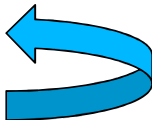
	has	max
A	3	9
B	2	4
C	2	7

Free: 3



	has	max
A	3	9
B	2	4
C	2	7

Free: 3



## Impasse (*Deadlock*)

Estratégias

### 3) Evitar dinamicamente o problema:

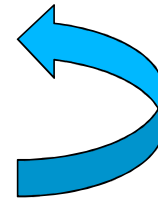
- Este estado é seguro ou inseguro?

	has	max
A	3	9
B	2	4
C	2	7

Free: 3

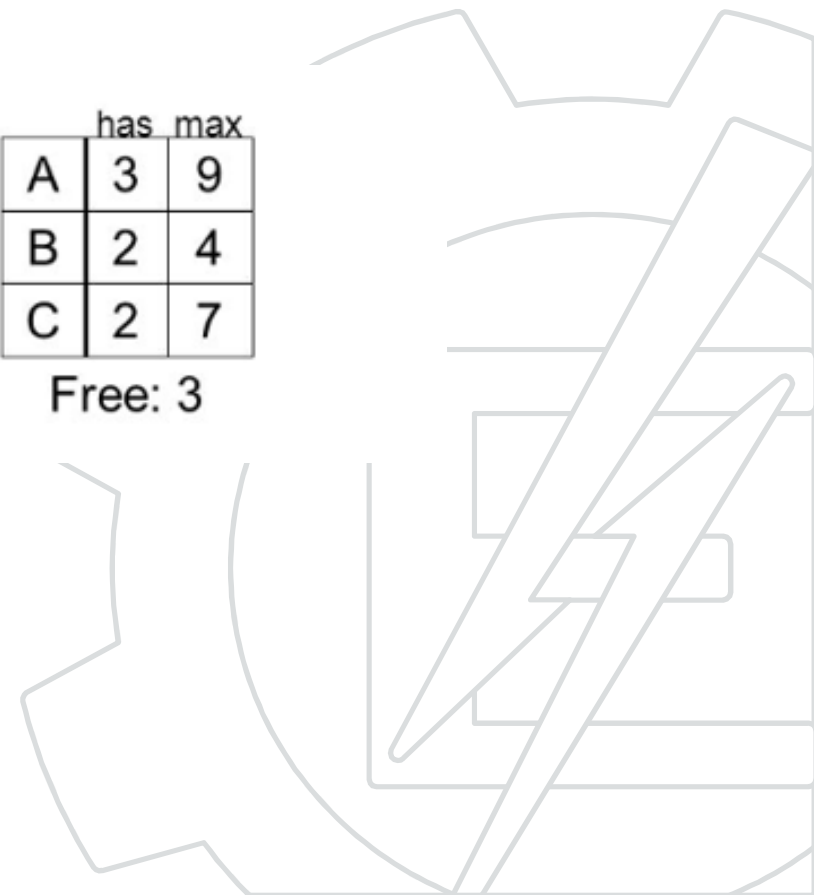
	has	max
A	3	9
B	2	4
C	2	7

Free: 3



	has	max
A	3	9
B	2	4
C	2	7

Free: 3



## Impasse (*Deadlock*)

Estratégias

### 3) Evitar dinamicamente o problema:

	has	max
A	3	9
B	2	4
C	2	7
Free: 3		

	has	max
A	3	9
B	4	4
C	2	7
Free: 1		

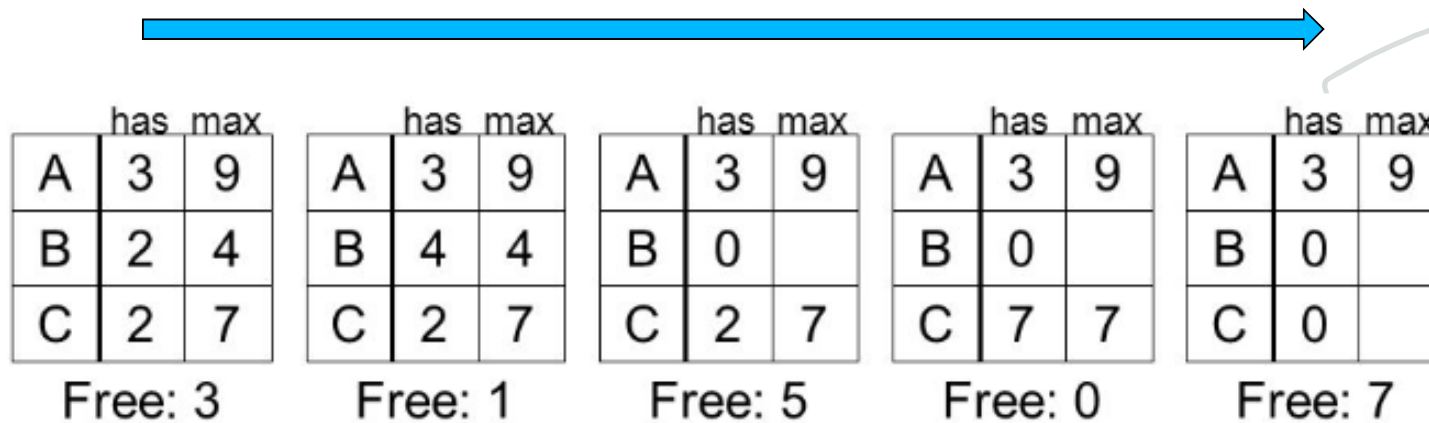


## Impasse (*Deadlock*)

Estratégias

### 3) Evitar dinamicamente o problema:

- Sequência para que todos os processos terminem:




## Impasse (*Deadlock*)

Estratégias

### 3) Evitar dinamicamente o problema:

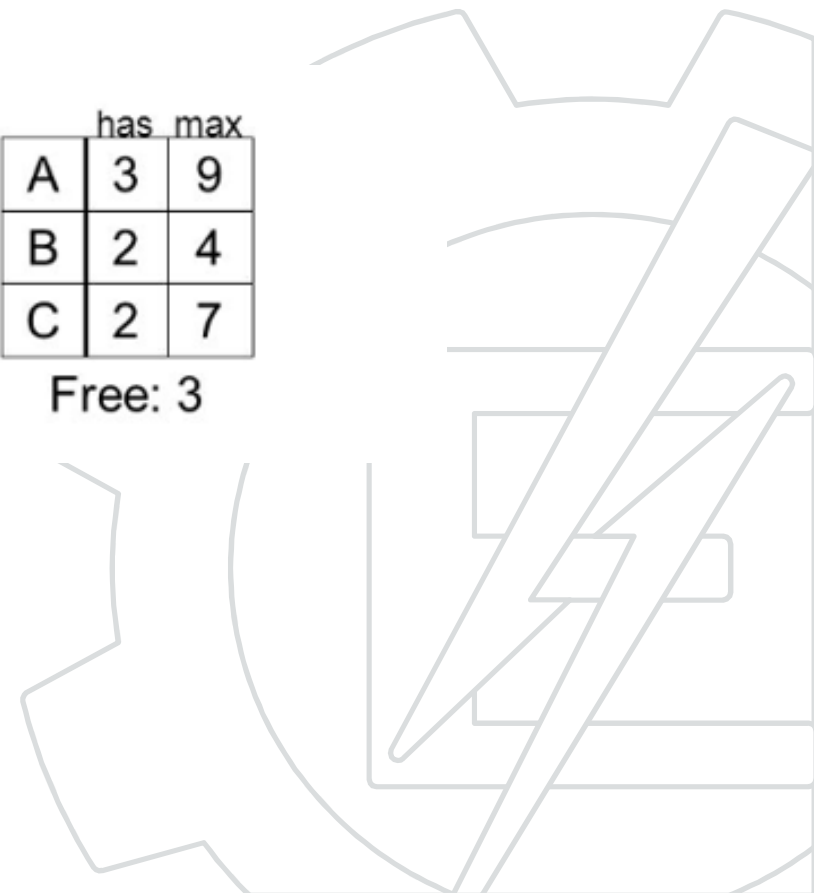
- Este estado é seguro ou inseguro?

	has	max
A	3	9
B	2	4
C	2	7
Free: 3		



	has	max
A	3	9
B	2	4
C	2	7
Free: 3		

	has	max
A	3	9
B	2	4
C	2	7
Free: 3		



## Impasse (*Deadlock*)

Estratégias

### 3) Evitar dinamicamente o problema:

- Este estado é seguro ou inseguro?

	has	max		has	max
A	3	9	A	4	9
B	2	4	B	2	4
C	2	7	C	2	7
Free: 3			Free: 2		



## Impasse (*Deadlock*)

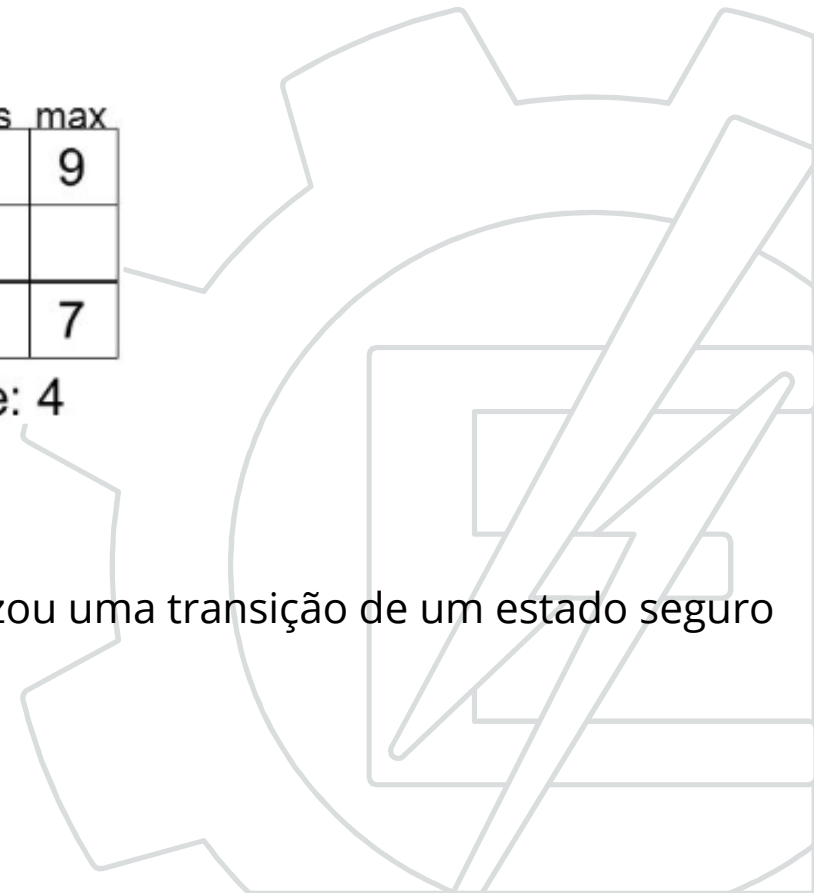
Estratégias

### 3) Evitar dinamicamente o problema:

- Estado inseguro

	has	max		has	max		has	max		has	max
A	3	9	A	4	9	A	4	9	A	3	9
B	2	4	B	2	4	B	4	4	B	0	
C	2	7	C	2	7	C	2	7	C	2	7
Free: 3			Free: 2			Free: 0			Free: 4		

- Não há sequência capaz de garantir que terminem.
- A decisão entre o primeiro e o segundo estado realizou uma transição de um estado seguro para um estado inseguro.
- A decisão deve ser tomada em tempo de execução.





## Impasse (*Deadlock*)

Estratégias

### 3) Evitar dinamicamente o problema:

- Apresente escolhas para que se chegue a um estado seguro e a um inseguro:

has max		
A	0	6
B	0	5
C	0	4
D	0	7

Free: 10



## Impasse (*Deadlock*)

Estratégias

### 3) Evitar dinamicamente o problema:

- Apresente escolhas para que se chegue a um estado seguro e a um inseguro:

has max

A	0	6
B	0	5
C	0	4
D	0	7

Free: 10

has max

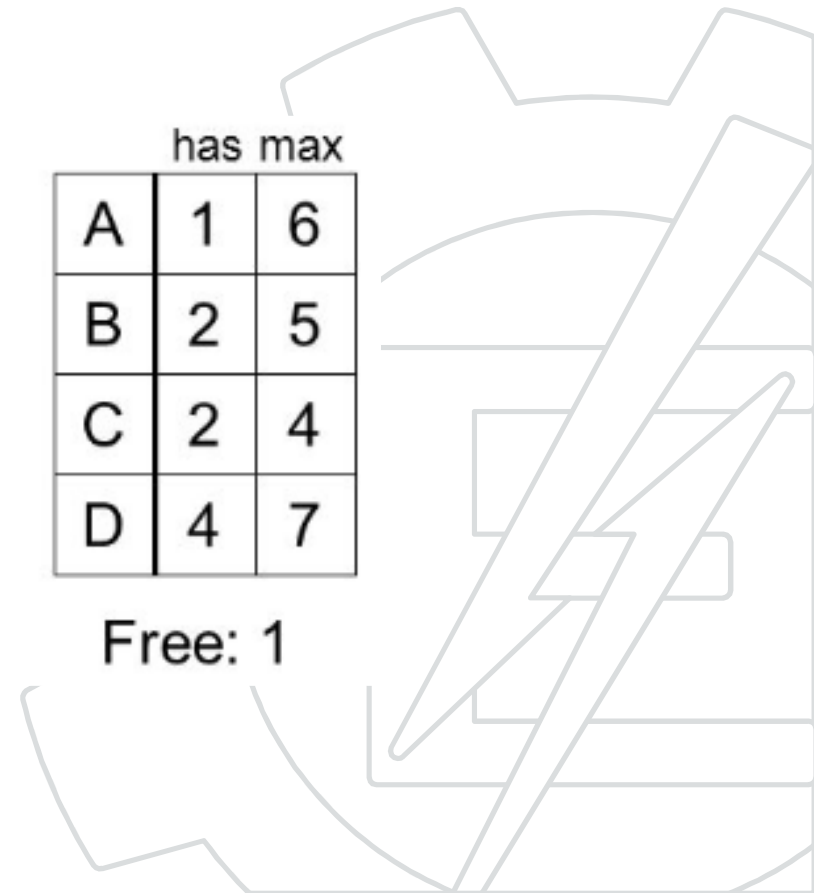
A	1	6
B	1	5
C	2	4
D	4	7

Free: 2

has max

A	1	6
B	2	5
C	2	4
D	4	7

Free: 1



## Impasse (*Deadlock*)

Estratégias

### 3) Evitar dinamicamente o problema:

- Apresente escolhas para que se chegue a um estado seguro e a um inseguro:

has max

A	0	6
B	0	5
C	0	4
D	0	7

Free: 10

has max

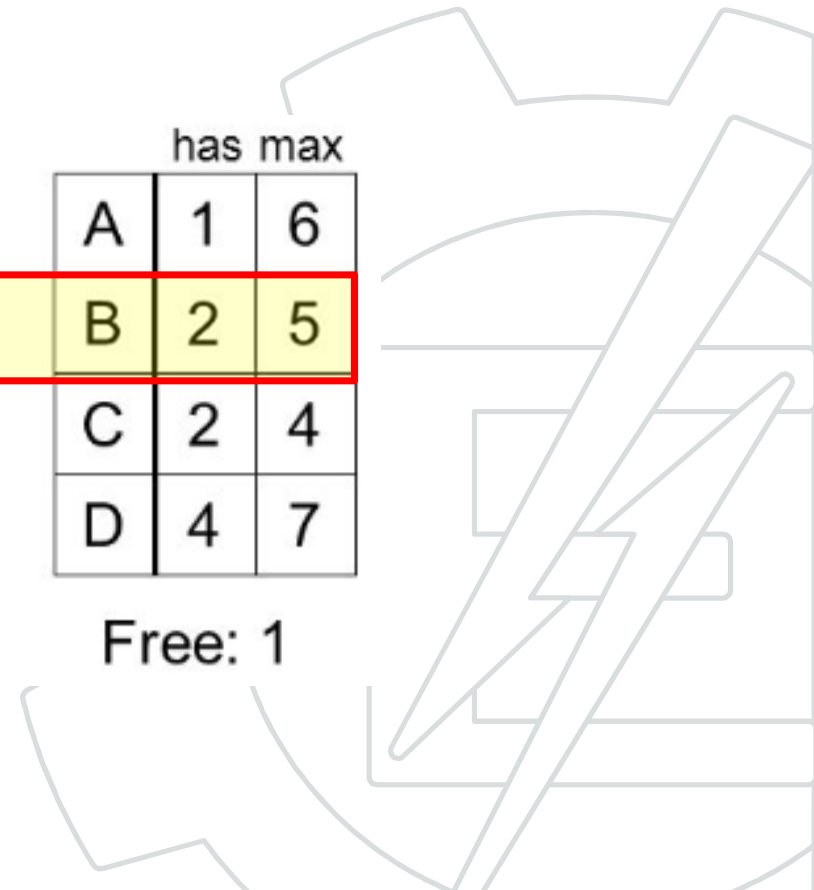
A	1	6
B	1	5
C	2	4
D	4	7

Free: 2

has max

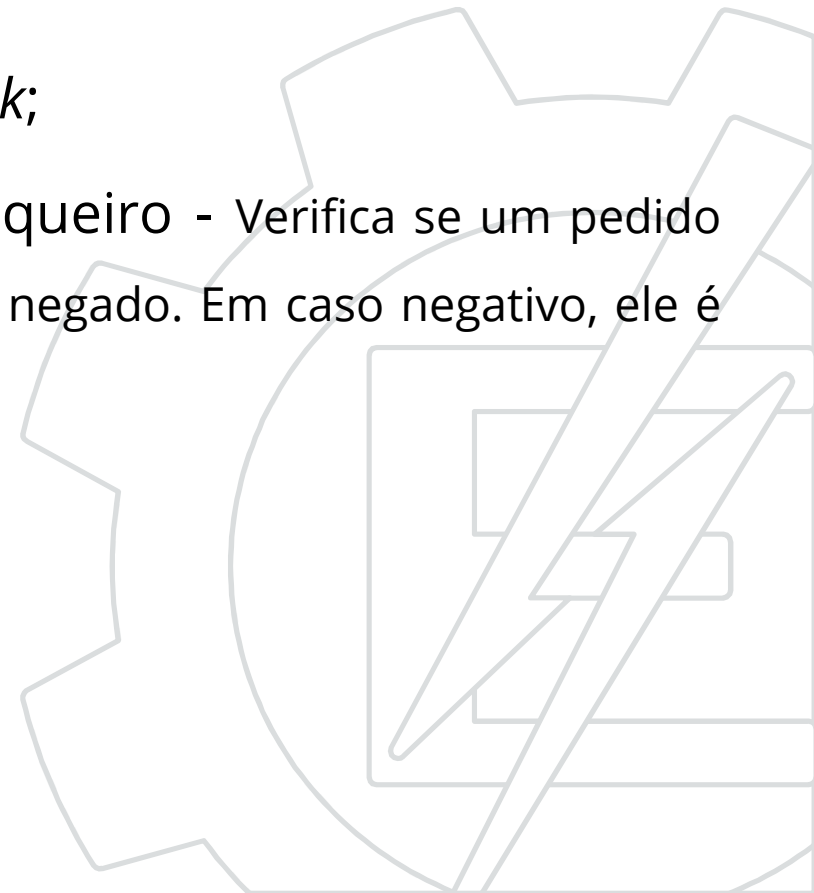
A	1	6
B	2	5
C	2	4
D	4	7

Free: 1



### Algoritmo do Banqueiro:

- Idealizado por Dijkstra (1965);
- Algoritmo de escalonamento para evitar *deadlock*;
- O Sistema Operacional funciona como um banqueiro - Verifica se um pedido leva a um estado inseguro. Em caso positivo, o pedido é negado. Em caso negativo, ele é executado.



## Impasse (*Deadlock*)

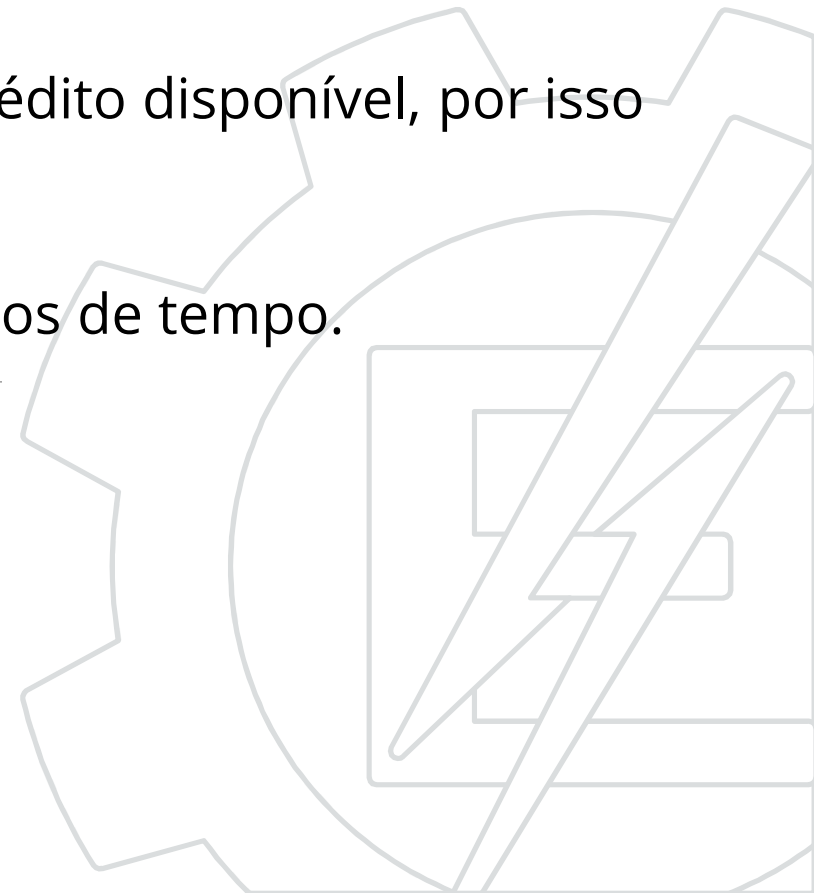
Estratégias

### Algoritmo do Banqueiro:

- 4 clientes: A, B, C e D;
- O banqueiro sabe que não precisarão de todo crédito disponível, por isso reservou 10 dos 22 para distribuir.
- Solicitações de crédito são realizadas em intervalos de tempo.
- Máxima linha de crédito: 22.

has max		
A	0	6
B	0	5
C	0	4
D	0	7

Free: 10



## Impasse (*Deadlock*)

Estratégias

### Algoritmo do Banqueiro:

- Estados seguros:

has max		
A	0	6
B	0	5
C	0	4
D	0	7

Free: 10

has max		
A	1	6
B	1	5
C	2	4
D	4	7

Free: 2

- O processo C pode ser executado seguramente.



## Impasse (*Deadlock*)

Estratégias

### Algoritmo do Banqueiro:

- Atendendo inicialmente a B com uma unidade a mais

has max		
A	0	6
B	0	5
C	0	4
D	0	7

Free: 10



has max		
A	1	6
B	2	5
C	2	4
D	4	7

Free: 1

o banqueiro não poderia suprir os outros clientes (estado inseguro).

## Impasse (*Deadlock*)

Estratégias

### Algoritmo do Banqueiro para vários recursos:

- E: Vetor de recursos existentes
- P: Vetor de recursos alocados
- A: Vetor de recursos disponíveis
- C: Matriz de alocação
- R: Matriz de requisição

Process  
Tape drives  
Plotters  
Scanners  
CD ROMs

A	3	0	1	1
B	0	1	0	0
C	1	1	1	0
D	1	1	0	1
E	0	0	0	0

Resources assigned

Process  
Tape drives  
Plotters  
Scanners  
CD ROMs

A	1	1	0	0
B	0	1	1	2
C	3	1	0	0
D	0	0	1	0
E	2	1	1	0

Resources still needed

E = (6342)  
P = (5322)  
A = (1020)



## Impasse (*Deadlock*)

Estratégias

### Algoritmo do Banqueiro para vários recursos:

- O que aconteceria se atendêssemos a B?

Process	Tape drives	Plotters	Scanners	CD ROMs
A	3	0	1	1
B	0	1	0	0
C	1	1	1	0
D	1	1	0	1
E	0	0	0	0

Resources assigned

Process	Tape drives	Plotters	Scanners	CD ROMs
A	1	1	0	0
B	0	1	1	2
C	3	1	0	0
D	0	0	1	0
E	2	1	1	0

Resources still needed

E = (6342)  
P = (5322)  
A = (1020)



## Impasse (*Deadlock*)

Estratégias

### Algoritmo do Banqueiro para vários recursos:

- Qual seria uma ordem de distribuição para estados seguros?

Process	Tape drives	Plotters	Scanners	CD ROMs
A	3	0	1	1
B	0	1	0	0
C	1	1	1	0
D	1	1	0	1
E	0	0	0	0

Resources assigned

Process	Tape drives	Plotters	Scanners	CD ROMs
A	1	1	0	0
B	0	1	1	2
C	3	1	0	0
D	0	0	1	0
E	2	1	1	0

Resources still needed

E = (6342)  
P = (5322)  
A = (1020)



## Algoritmo do Banqueiro

- **Desvantagens:**

- Pouco utilizado, pois é difícil saber quais recursos serão necessários.
- O escalonamento cuidadoso é muito caro para o sistema.
- O número de processos é dinâmico e pode variar constantemente.

- **Vantagem:**

- Na teoria, o algoritmo é ótimo.



### 4) Prevenção:

- **Exclusão mútua:** alocar recursos utilizando *spooling* (somente o *printer daemon* tem acesso direto à impressora);
- **Posse durante a espera:** processos requisitam todos os recursos que precisam antes da execução. Difícil controle e gera sobrecarga;
- **Inexistência de preempção:** retirada de recursos dos processos – praticamente não implementável, pois traz prejuízos;
- **Espera circular:** ordenar numericamente os recursos e realizar a solicitação em ordem – não há ciclos. Permitir que o processo utilize apenas um recurso por vez – se quiser um segundo recurso deve liberar o primeiro.

### 4) Prevenção:

- **Exclusão mútua** - Pelo menos um recurso deve ser não compartilhável.
  - Recursos compartilháveis, por outro lado, não requerem acesso mutuamente exclusivo e, portanto, não podem estar envolvidos em um *deadlock*.
  - Arquivos somente de leitura são um bom exemplo de recurso compartilhável. Se vários processos tentam abrir um arquivo somente de leitura ao mesmo tempo, podem conseguir acesso simultâneo ao arquivo.

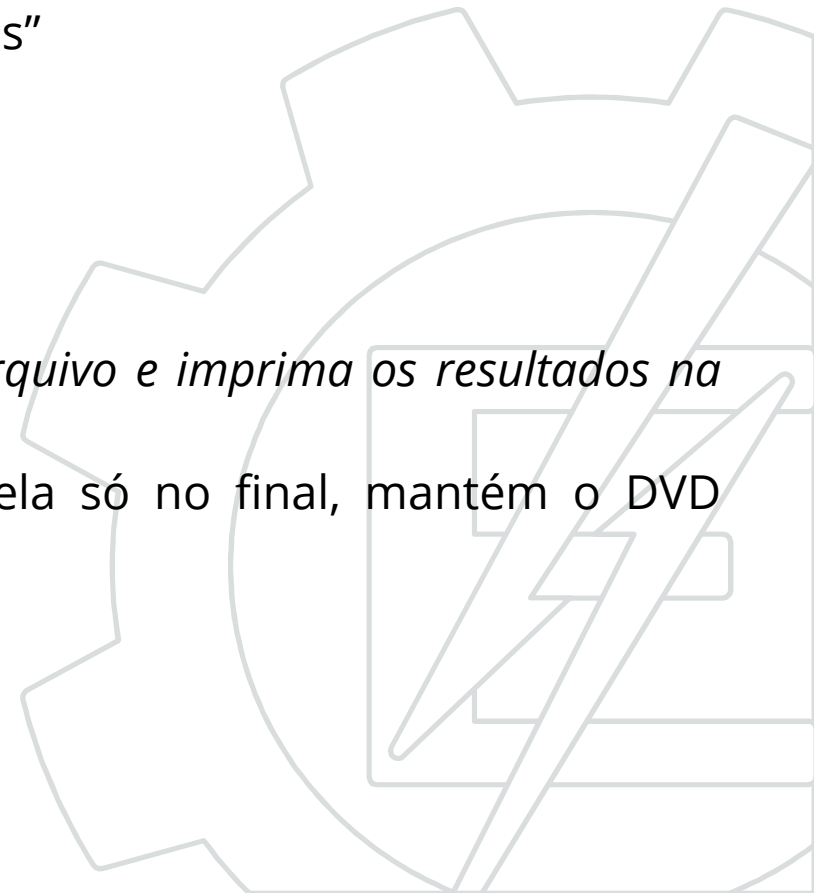
### 4) Prevenção:

- **Exclusão mútua** - Pelo menos um recurso deve ser não compartilhável.
  - Um processo nunca precisa esperar por um recurso compartilhável.
  - Em geral, porém, não podemos prevenir a ocorrência de *deadlocks* negando a condição de exclusão mútua, porque alguns recursos são intrinsecamente não-compartilháveis. Por exemplo, um *lock mutex* não pode ser compartilhado simultaneamente por vários processos.

## 4) Prevenção:

### •Posse durante a espera

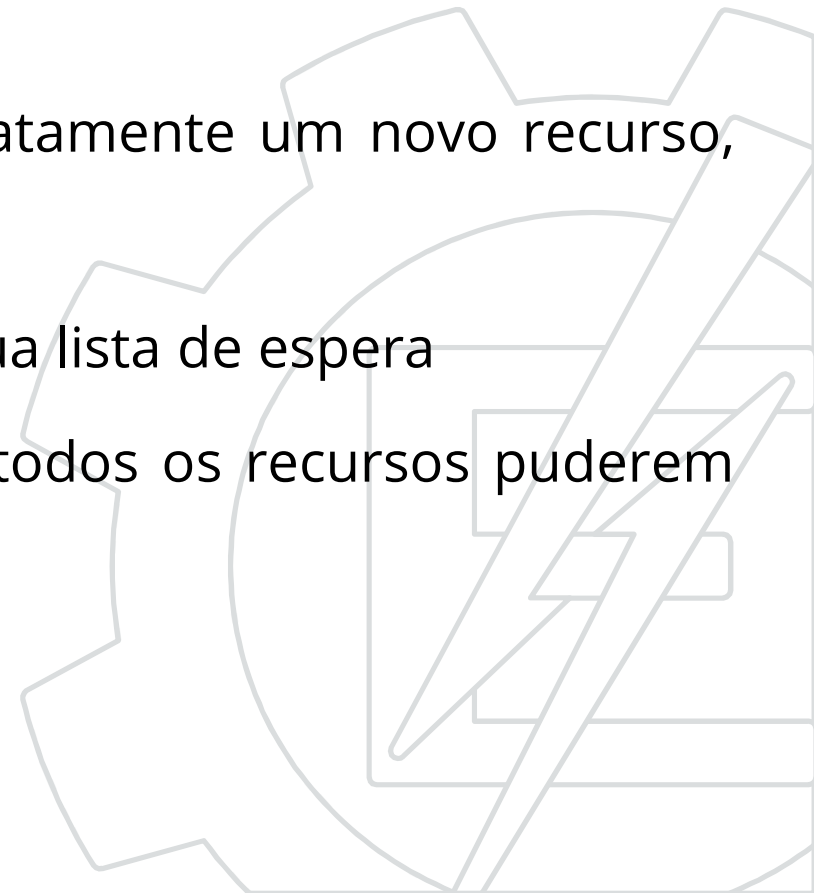
- Não permitir que processos peçam recursos “aos poucos”
  - A) **pedem todos** de uma vez ou
  - B) **liberam todos** os que detêm antes de pedir outros
- Pode levar à **baixa utilização dos recursos** ou **inanição**
  - ex.: *copie do DVD para arquivo no HD, classifique o arquivo e imprima os resultados na impressora*
    - °A) mantém a impressora apesar de precisar dela só no final, mantém o DVD apesar de precisar só no início
    - °B) pode demorar a conseguir o HD pela 2ª vez  
–1ª vez DVD(**r**)→HD(**w**) ; 2ª vez HD(**r**)→IMPR(**w**)



### 4) Prevenção:

- **Inexistência de preempção**

- Se um processo não conseguir alocar imediatamente um novo recurso, **deve abrir mão** dos que já detém (**intercepta**)
  - Implicitamente, eles serão adicionados à sua lista de espera
  - O processo só poderá continuar quando todos os recursos puderem ser obtidos (antigos e novos)
- Pode levar à inanição

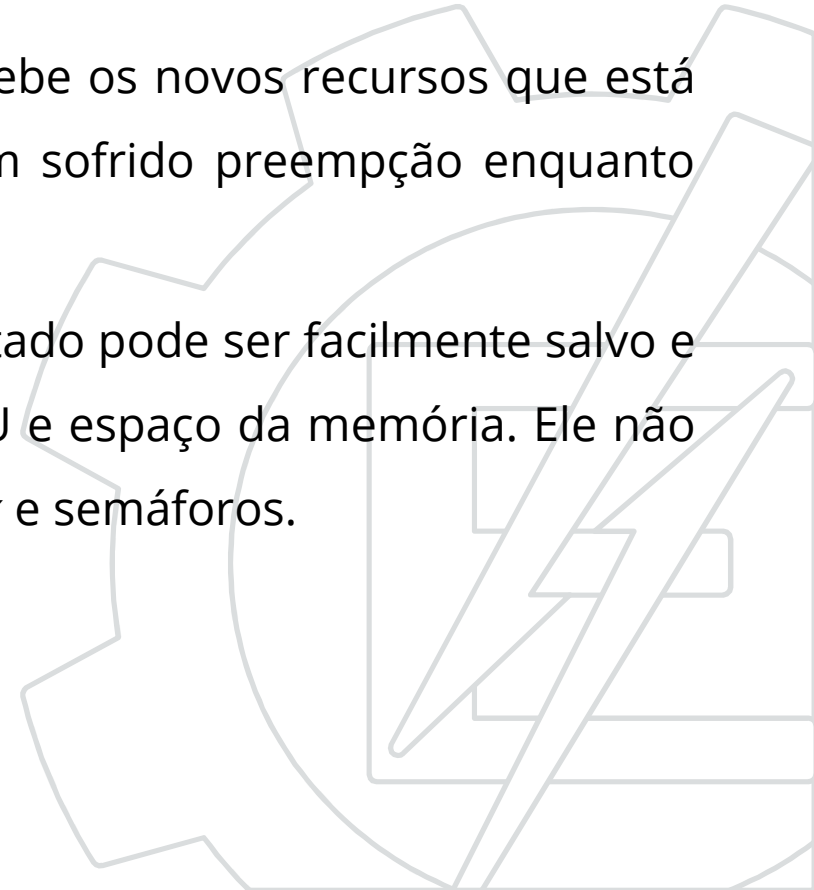




### 4) Prevenção:

- **Inexistência de preempção**

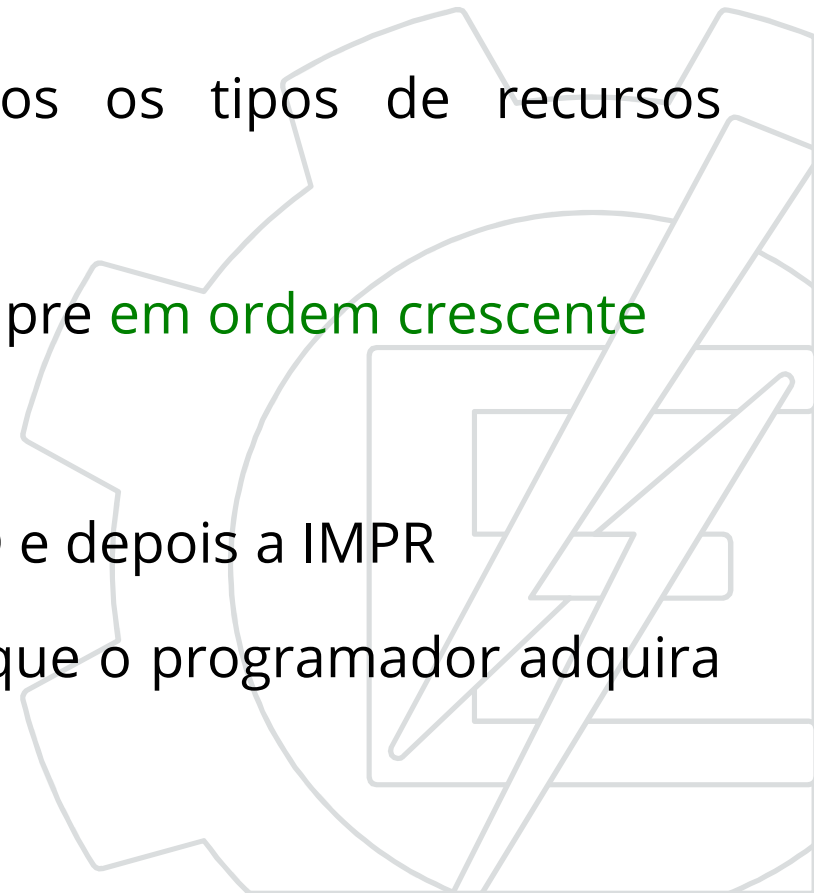
- Um processo pode ser reiniciado somente quando recebe os novos recursos que está solicitando e recupera quaisquer recursos que tenham sofrido preempção enquanto ele estava esperando.
- Esse protocolo costuma ser aplicado a recursos cujo estado pode ser facilmente salvo e restaurado posteriormente, como registradores da CPU e espaço da memória. Ele não pode ser aplicado em geral a recursos como *locks mutex* e semáforos.



### 4) Prevenção:

- **Espera circular**

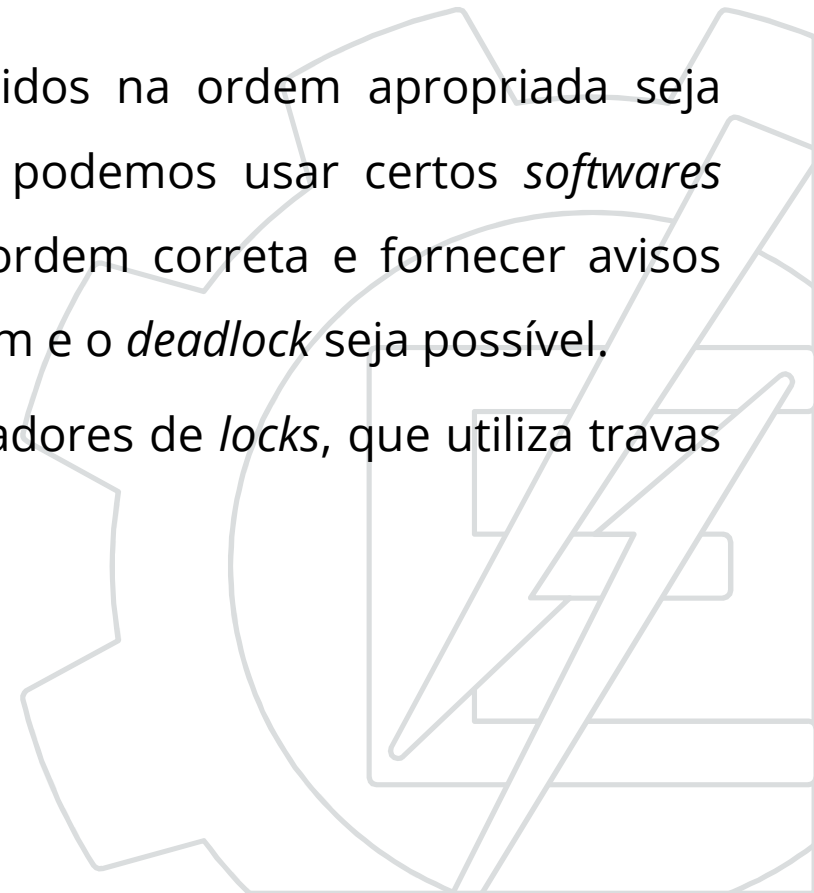
- Defina uma ordenação absoluta para todos os tipos de recursos disponíveis
- Exija que todo processo requisiite recursos sempre **em ordem crescente**
  - ex.: DVD(1), HD(5), IMPR(12)
    - DVD+IMPR: deve solicitar primeiro o DVD e depois a IMPR
- Impede a formação de ciclos no grafo, desde que o programador adquira os recursos na ordem apropriada



### 4) Prevenção:

- **Espera circular**

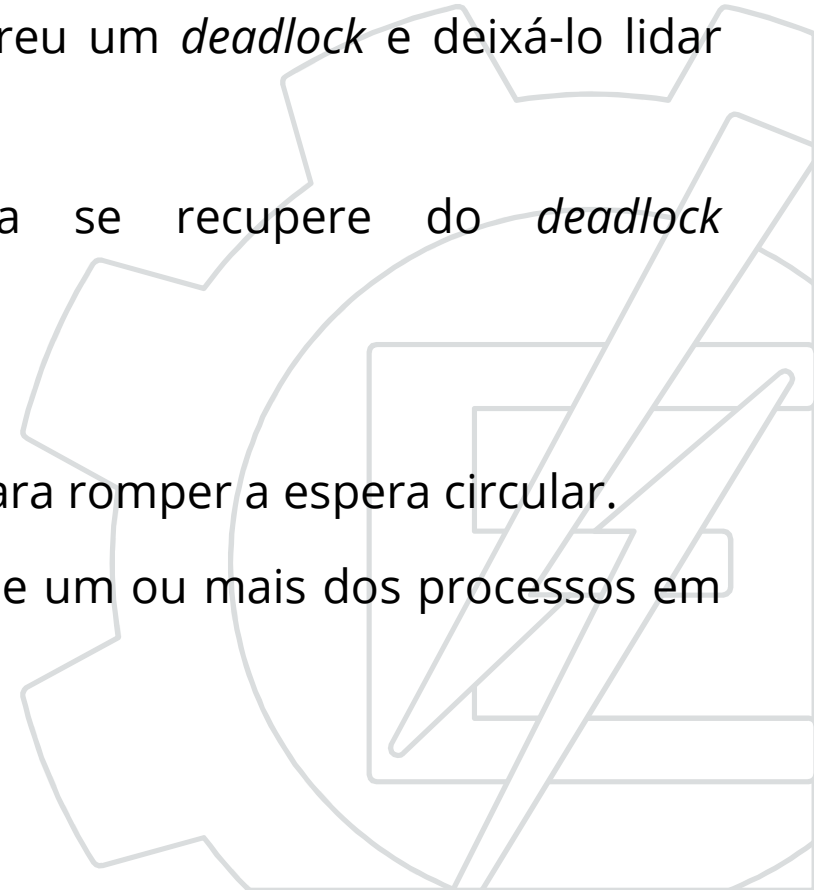
- Embora a garantia de que os recursos sejam adquiridos na ordem apropriada seja responsabilidade dos desenvolvedores de aplicações, podemos usar certos *softwares* para verificar se os *locks* estão sendo adquiridos na ordem correta e fornecer avisos apropriados quando eles forem adquiridos fora de ordem e o *deadlock* seja possível.
- Versões BSD do UNIX como o FreeBSD possuem verificadores de *locks*, que utiliza travas de exclusão mútua para proteger seções críticas.



## Impasse (*Deadlock*)

Estratégias

- Quando um algoritmo de detecção determina que existe um *deadlock*, várias alternativas estão disponíveis:
  - a) Uma possibilidade é informar ao operador que ocorreu um *deadlock* e deixá-lo lidar com o problema manualmente.
  - b) Outra possibilidade é permitir que o sistema se recupere do *deadlock* automaticamente.
- Há duas opções para a interrupção de um *deadlock*:
  - a) Uma é simplesmente abortar um ou mais processos para romper a espera circular.
  - b) A outra é provocar a preempção de alguns recursos de um ou mais dos processos em *deadlock*.



## Impasse (*Deadlock*)

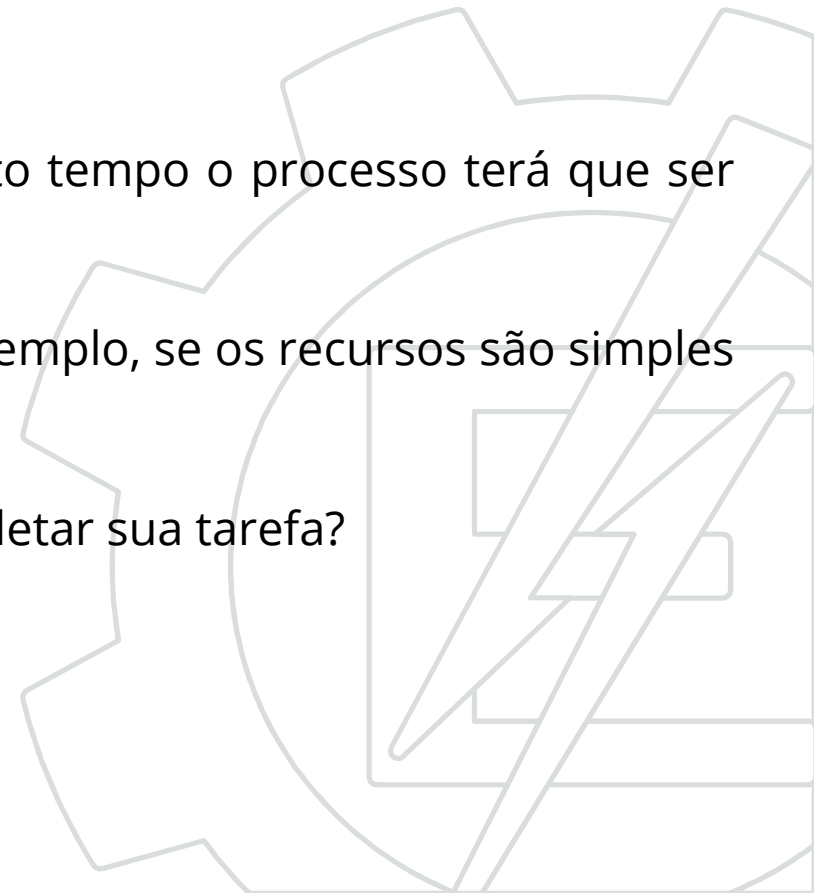
Estratégias

- **Abortar um processo de cada vez até que o ciclo do *deadlock* seja eliminado** - Esse método incorre em sobrecarga considerável, já que após cada processo ser abortado, um algoritmo de detecção de *deadlocks* deve ser invocado para determinar se algum processo ainda está em *deadlock*.
- **Pode não ser fácil abortar um processo** - Se o processo estava no meio da atualização de um arquivo, seu encerramento deixará esse arquivo em um estado incorreto. Da mesma forma, se o processo estava no meio da impressão de dados em uma impressora, o sistema deve reposicionar a impressora para um estado correto antes de imprimir o próximo *job*.

## **Impasse (*Deadlock*)**

Estratégias

- Muitos fatores podem afetar a decisão de qual processo deve ser selecionado para o encerramento, incluindo:
  - a) Qual é a prioridade do processo?
  - b) Por quanto tempo o processo foi executado e por quanto tempo o processo terá que ser executado para completar sua tarefa?
  - c) Quantos e que tipos de recursos o processo usou (por exemplo, se os recursos são simples de serem interceptados)?
  - d) De quantos recursos o processo ainda precisa para completar sua tarefa?
  - e) Quantos processos terão que ser encerrados?
  - f) O processo é interativo ou *batch*?



- **Preempção de Recursos**

- Para eliminar *deadlocks* usando a preempção de recursos, provocamos a preempção sucessiva de alguns recursos dos processos e damos esses recursos a outros processos até que o ciclo do *deadlock* seja rompido.
  - **Seleção de uma vítima:** Que recursos e que processos devem sofrer preempção?
  - **Reversão:** Se provocarmos a preempção de um recurso de um processo, o que deve ser feito com esse processo? Retornar sua execução a um ponto seguro ou reiniciá-lo?
  - **Inanição:** Como assegurar que a inanição não ocorrerá? Isto é, como podemos garantir que os recursos interceptados não serão sempre do mesmo processo?

1) Considere a situação a seguir:

	<u>Allocation</u>	<u>Max</u>	<u>Available</u>	<u>Need</u>
	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

a) Agora, considere o que ocorre se o processo  $P_1$  requisitar 1 instância de A e 2 instâncias de C  $\rightarrow Request[1] = (1, 0, 2)$

b) Considere os casos a seguir e analise a possibilidade de execução e os estados encontrados:

- $Request[4] = (3, 3, 0)$
- $Request[0] = (0, 2, 0)$



## Exercícios

de fixação

2) Considere a situação a seguir e verifique se está em *deadlock*:

	<u>Allocation</u>	<u>Request</u>	<u>Available</u>
	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	

3) Considere a situação a seguir e verifique se está em *deadlock*:

	<u>Allocation</u>	<u>Request</u>	<u>Available</u>
	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 1	
$P_3$	2 1 1	1 0 0	
$P_4$	0 0 2	0 0 2	



## Exercícios de fixação

4) Apresente as possíveis sequências de estados seguros para a execução dos processos a seguir a partir do cenário apresentado:

Total Resources	R1	R2	R3
	10	5	7

Process	Allocation			Max		
	R1	R2	R3	R1	R2	R3
P1	0	1	0	7	5	3
P2	2	0	0	3	2	2
P3	3	0	2	9	0	2
P4	2	1	1	2	2	2



## Exercícios

de fixação

5) Um sistema possui 4 processos e 5 recursos que permitem alocação. A alocação atual e as necessidades máximas são as seguintes:

	Alocado	Máximo	Disponível
Processo A	1 0 2 1 1	1 1 2 1 3	00x11
Processo B	2 0 1 1 0	2 2 2 1 0	
Processo C	1 1 0 1 0	2 1 3 1 0	
Processo D	1 1 1 1 0	1 1 2 2 1	

Qual é o menor valor de **x** que produz um Estado Seguro?



# Bibliografia

- TANENBAUM, Andrew S; BOS, Herbert. Sistemas operacionais modernos. 4a ed. São Paulo: Pearson Education do Brasil, 2016.

## Capítulo 6.

<https://plataforma.bvirtual.com.br/Acervo/Publicacao/1233>

- DEITEL, H.M; DEITEL, P.J; CHOFFNES,D.R. Sistemas Operacionais. 3a ed. São Paulo: Pearson Prentice Hall, 2005. **Capítulo 7.**

<https://plataforma.bvirtual.com.br/Acervo/Publicacao/315>



# Sistemas Operacionais

Prof. Otávio Gomes

[otavio.gomes@unifei.edu.br](mailto:otavio.gomes@unifei.edu.br)

