Sistemas Operacionais

Processos

Comunicação e Sincronização entre Processos

Especificação de Execução Concorrente

Questão importante na estruturação de Algoritmos paralelos → **Como decompor um problema em um conjunto de processos paralelos**

Algumas formas de se expressar uma execução concorrente (usadas em algumas linguagens e sistemas operacionais

- **Co-rotinas**
- **Declarações FORK/JOIN**
- **Declarações COBEGIN/COEND**
- **Declarações de Processos Concorrentes**

Processos

Co-Rotinas

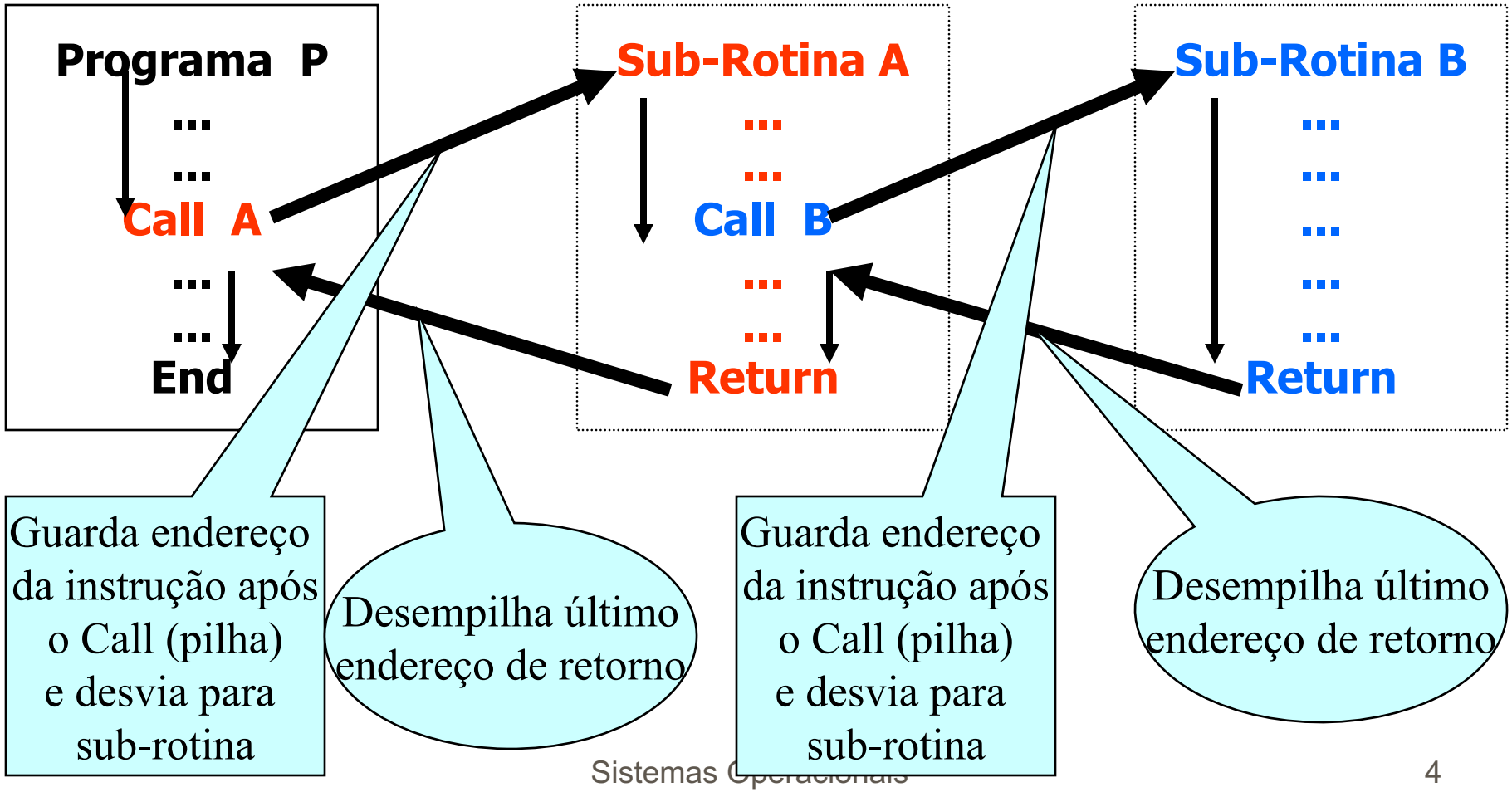
- As co-rotinas são parecidas com sub-rotinas (ou procedimentos), diferindo apenas na forma de transferência de controle, realizada na chamada e no retorno

- As co-rotinas possuem um ponto de entrada, mas pode representar diversos pontos intermediários de entrada e saída

- A transferência de controle entre eles é realizada através do endereçamento explícito e de livre escolha do programador (através de comandos do tipo TRANSFER, do Modula-2)

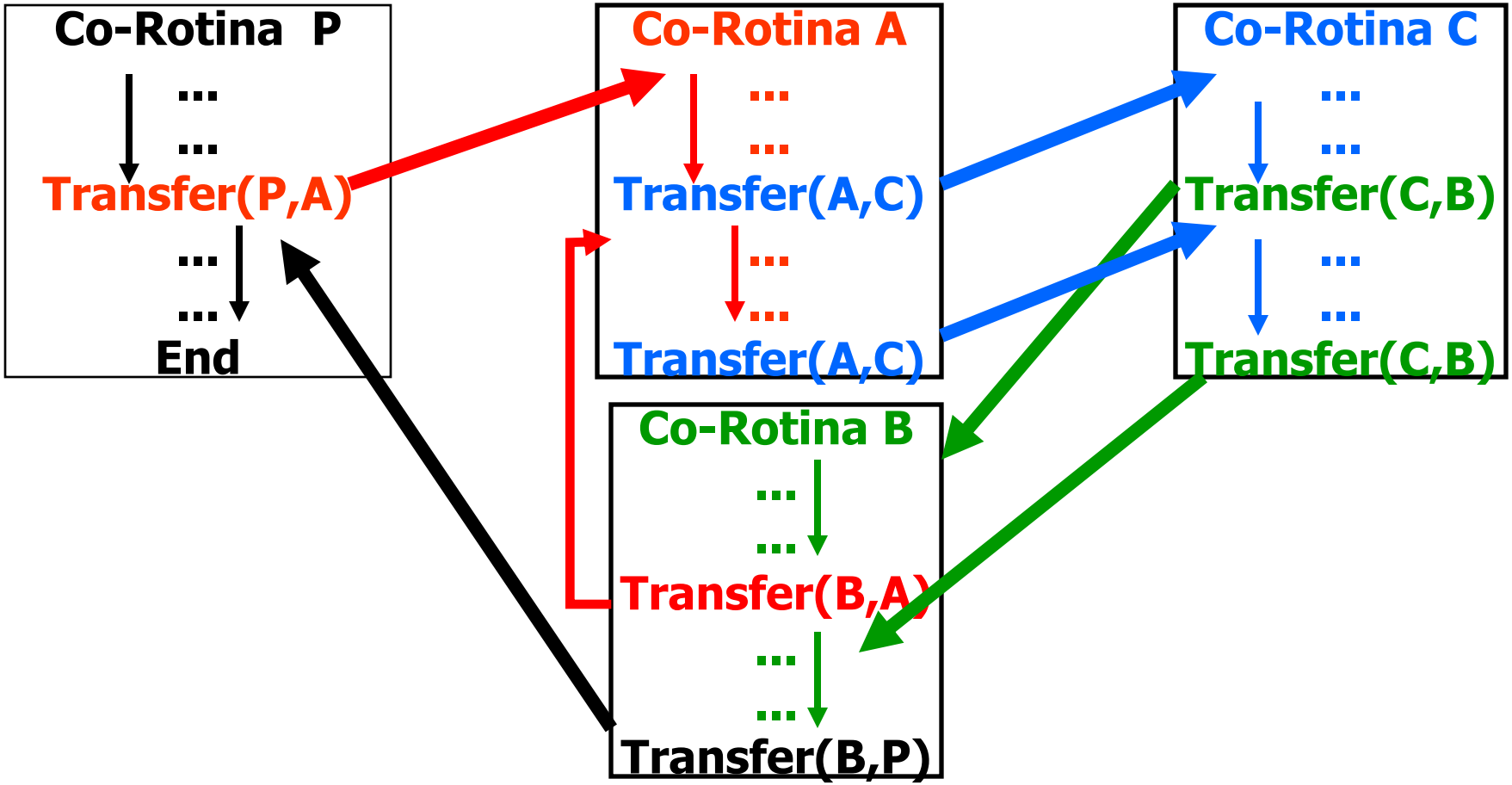
Processos

Funcionamento das Sub-rotinas comuns



Processos

Funcionamento das Co-Rotinas

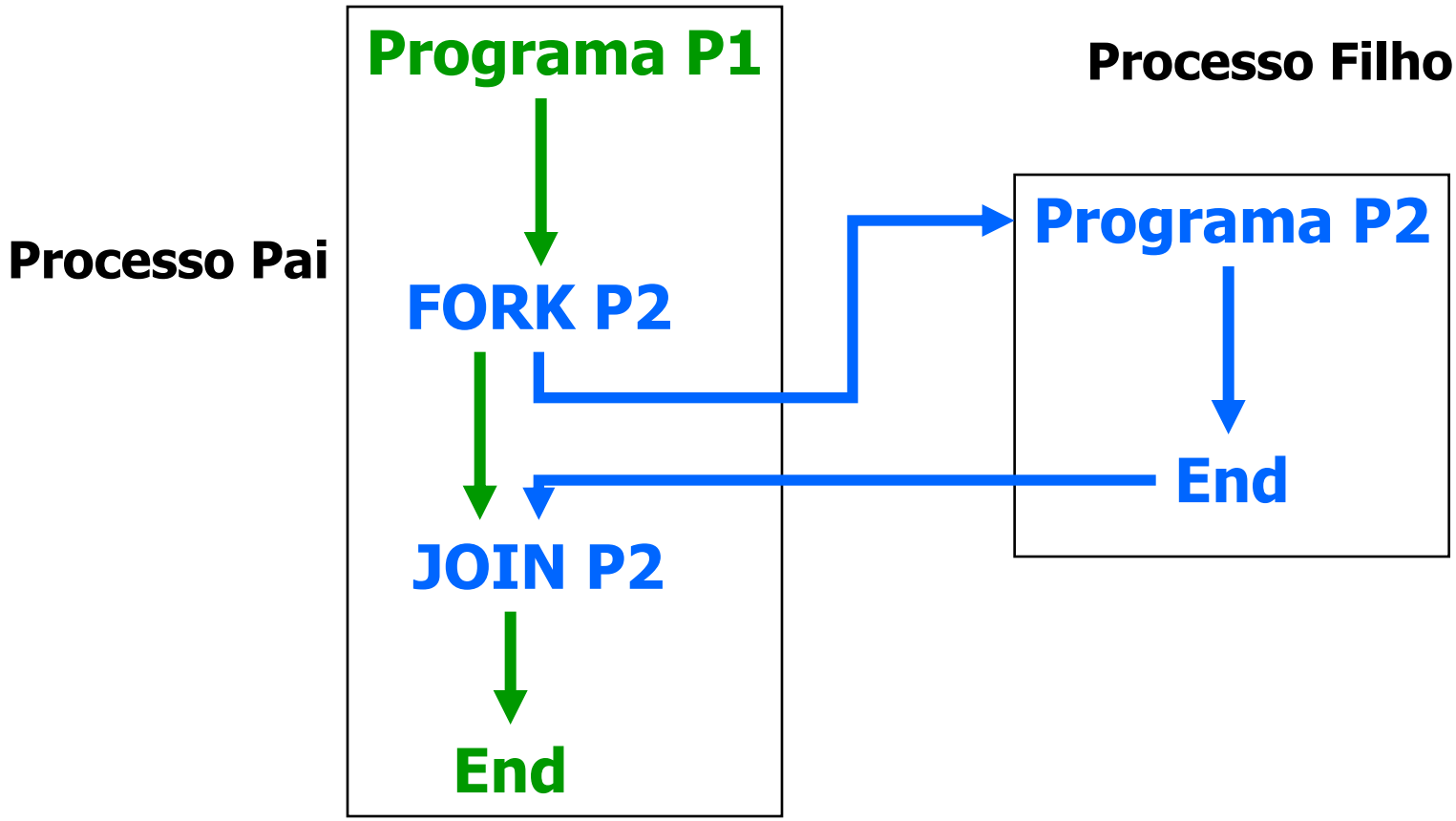


Processos

- **Declarações FORK/JOIN**
  - **A declaração `FORK <nome do programa>` determina o início de execução de um determinado programa, de forma concorrente com o programa sendo executado.**
  - **Para sincronizar-se com o término do programa chamado, o programa chamador deve executar a declaração `JOIN <nome do programa chamado>`.**
  - **O uso do `FORK/JOIN` permite a concorrência e um mecanismo de criação dinâmica entre processos (criação de múltiplas versões de um mesmo programa -> processo-filho), como no sistema UNIX**

Processos

Declarações FORK/JOIN



Processos

- **Declarações COBEGIN/COEND**
  - **Constituem uma forma estruturada de especificar execução concorrente ou paralela de um conjunto de declarações agrupadas da seguinte maneira:**

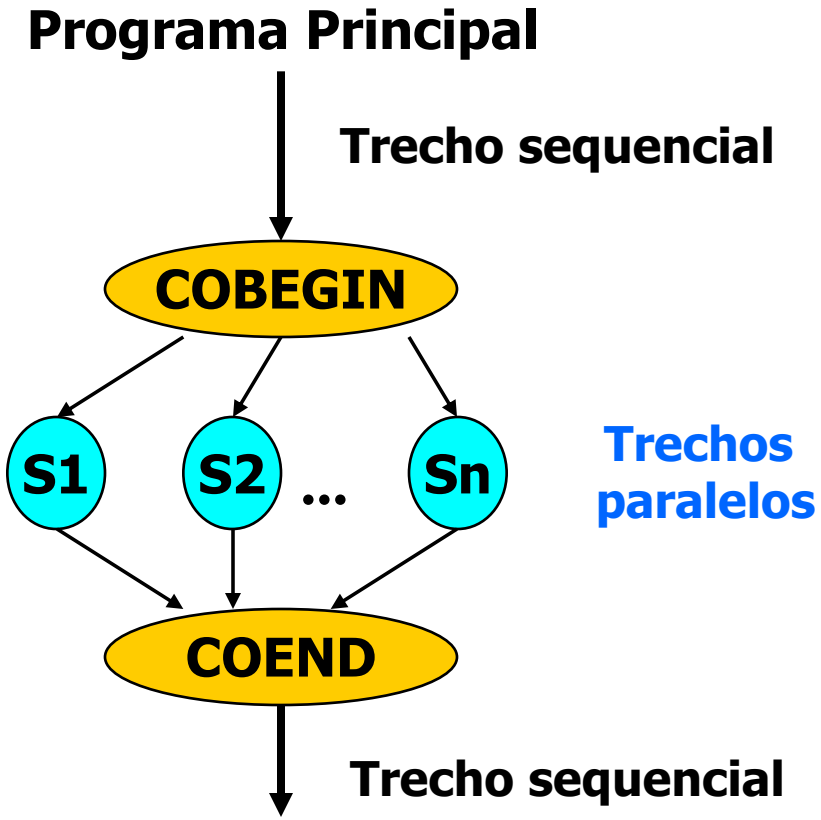
```
COBEGIN
  S1//S2//...//Sn
COEND
```

Onde:

- **A execução deste trecho de programa provoca a execução concorrente das declarações S1, S2, ..., Sn.**
- **Declarações Si podem ser qualquer declaração, incluindo o para COBEGIN/COEND, ou um bloco de declarações locais.**
- **Esta execução só termina, quando todas as declarações Si terminarem.**

Processos

Declarações COBEGIN/COEND




```
Program Paralelo;
/* declaração de var.e const. globais */
Begin
   /* trecho sequencial */
  ...
  COBEGIN /* trechos   paralelos */
       Begin  /* S1 */
          ...
       End;
       ...
       Begin /* Sn */
          ...
       End;
  COEND
  /* trecho sequencial */
  ...
End.
```

Processos

- **Declarações de Processos Concorrentes**
  - **Geralmente, programas de grande porte são estruturados como conjunto de trechos sequenciais de programa, que são executados concorrentemente.**
  - **Poderiam ser utilizadas as co-rotinas, FORK/JOIN, ou Cobegin/Coend, porém a estrutura de um programa será mais clara se a especificação dessas rotinas explicitar que as mesmas são executadas concorrentemente.**
  - **Exemplo de linguagens: DP (Distributed Process): utiliza um único cobegin e coend – apenas 1 instância de programa; ADA: várias instâncias (processos podem ser criados dinamicamente – pode existir um número variável de processos).**

Processos

Declarações de Processos Concorrentes

```
Program Conjunto_Processos;
/* declaração de var.e const. globais */
  ...
Processo Pi;
  /* declaração de var.e const. locais */
   ...
  End;
/* Outros  processos */
Processo Pn;
  /* declaração de var.e const. locais */
   ...
  End;
End.
```

Processos

Mecanismos Simples de Comunicação e Sincronização entre Processos

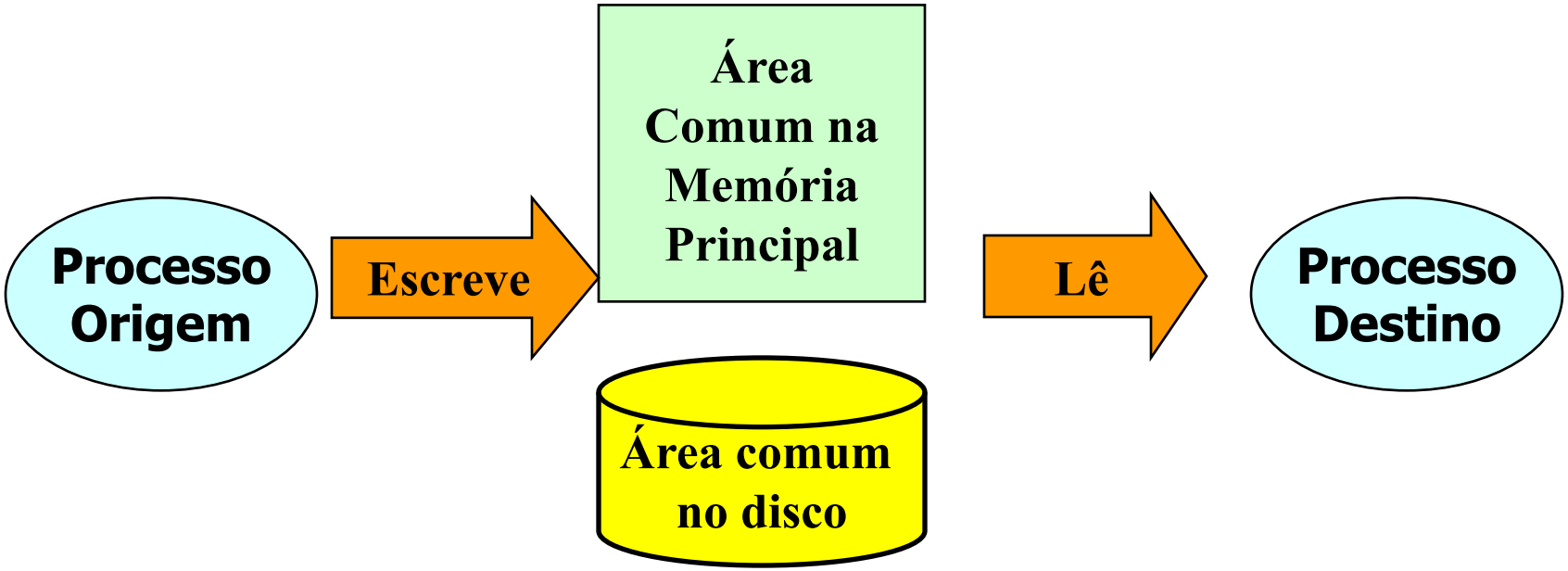
- **Num sistema de multiprocessamento ou multiprogramação, os processos geralmente precisam se comunicar com outros processos.**

- **A comunicação entre processos é mais eficiente se for estruturada e não utilizar interrupções.**

Processos

Condições de Corrida

Em alguns Sistemas Operacionais: os processos se comunicam através de alguma área de armazenamento comum. Esta área pode estar na memória principal ou pode ser um arquivo compartilhado.



Processos

Condições de Corrida

- **Definição de condições de corrida: situações onde dois os mais processos estão lendo ou escrevendo algum dado compartilhado e o resultado depende de quem processa no momento propício.**

- **Depurar programas que contém condições de corrida não é fácil, pois não é possível prever quando o processo será suspenso.**

Processos

Condições de Corrida

- Um exemplo: Print Spooler

- Quando um processo deseja imprimir um arquivo, ele coloca o nome do arquivo em uma lista de impressão (spooler directory).

- Um processo chamado "printer daemon", verifica a lista periodicamente para ver se existe algum arquivo para ser impresso, e se existir, ele os imprime e remove seus nomes da lista.

Processos

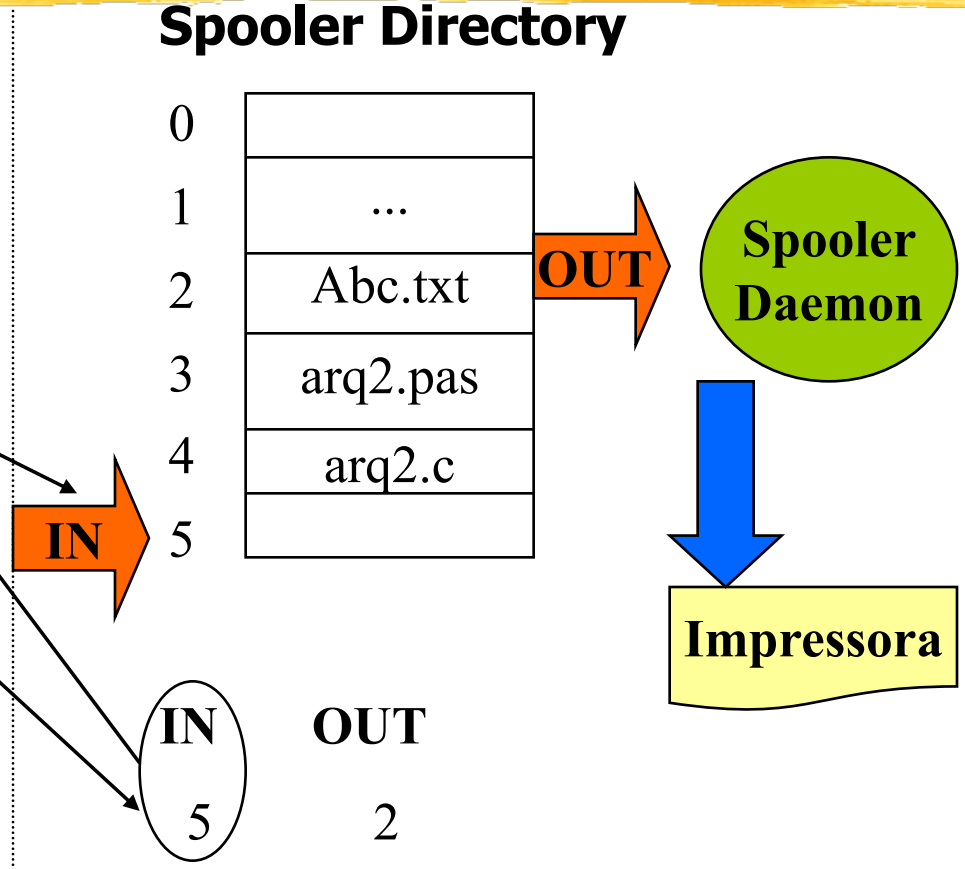
Condições de Corrida

Processo A

```
...
get_prox_slot(slot);
set_nomearq(slot,nomearq);
slot++;
set_prox_slot(slot);
...
```

Processo B

```
...
get_prox_slot(sl);
set_nomearq(sl,nomearq);
slot++;
set_prox_slot(slot);
...
```



Processos

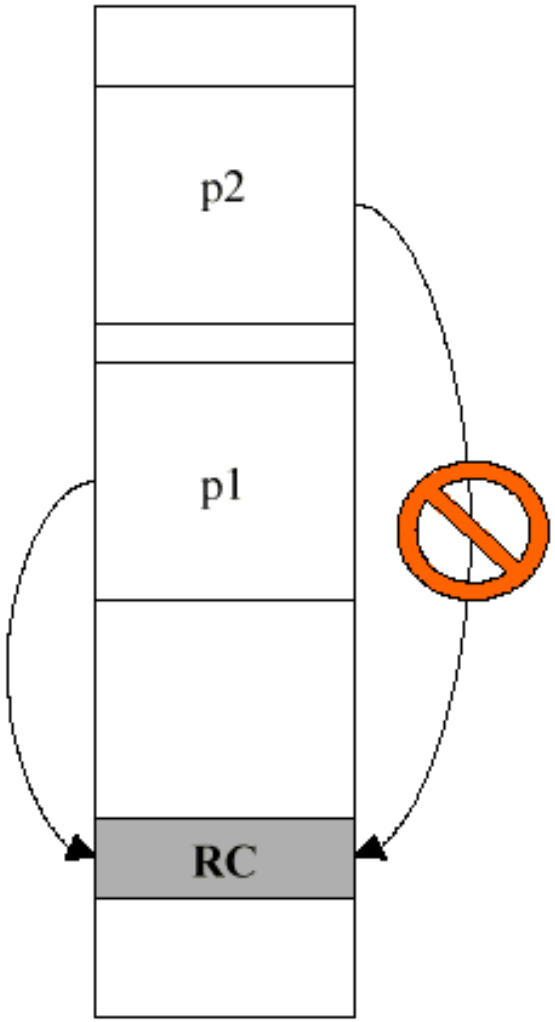
Regiões Críticas

- Uma solução para as condições de corrida é proibir que mais de um processo leia ou escreva em uma variável compartilhada ao mesmo tempo.

- Esta restrição é conhecida como **exclusão mútua**, e os trechos de programa de cada processo que usam um recurso compartilhado e são executados um por vez, são denominados **seções críticas** ou **regiões críticas (R.C.).**

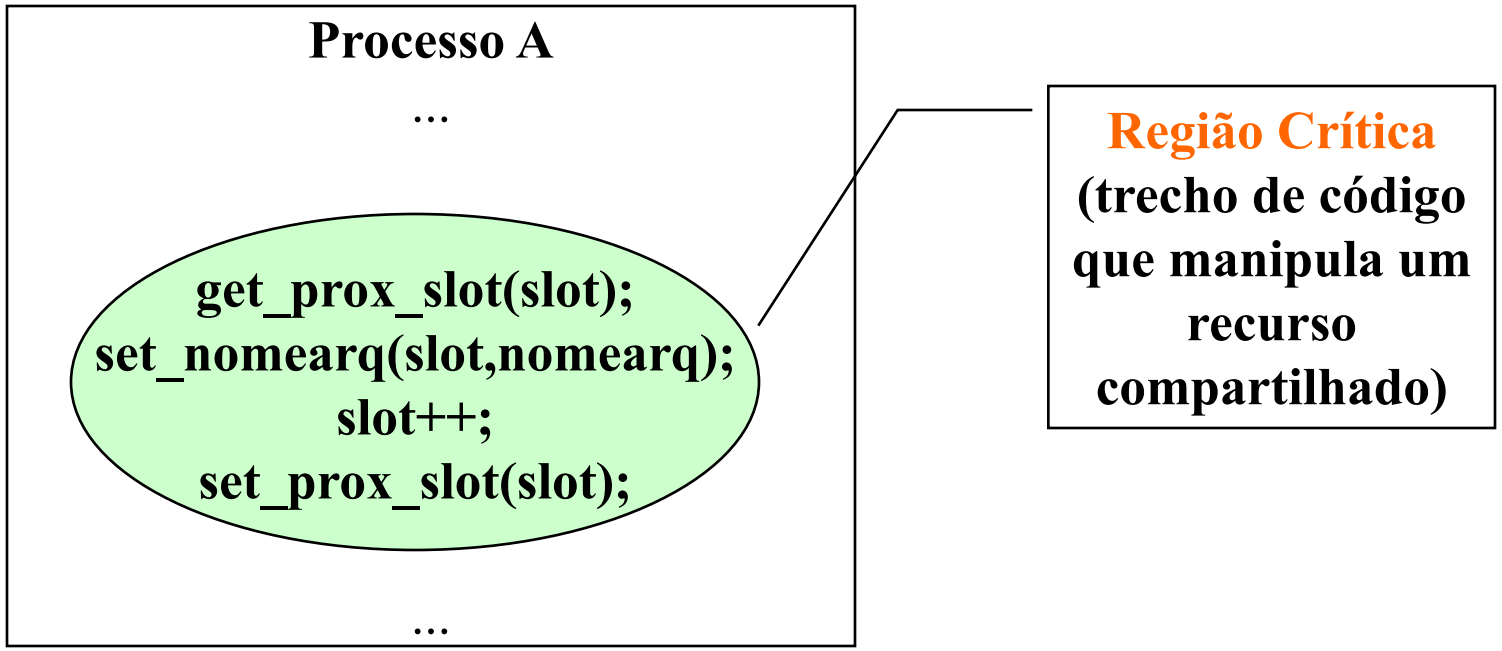
Processos

Regiões Críticas



Processos

Regiões Críticas



Processos

Exclusão Mútua

- **Condições para uma boa solução de exclusão mútua:**
  - dois processos não podem estar simultaneamente dentro de suas R.C.s.
  - nenhuma suposição é feita quanto à velocidade relativa dos processos ou o número de UCPs.
  - nenhum processo parado fora de sua R.C. pode bloquear outros processos de entrarem na R.C.
  - nenhum processo deve esperar um tempo arbitrariamente longo em sua região crítica (R.C. longa afeta o desempenho do sistema).

Processos

Exclusão Mútua com Espera Ocupada

Desabilitando as Interrupções

SOLUÇÃO MAIS SIMPLES: cada processo desabilita todas as interrupções (inclusive a do relógio) após entrar em sua região crítica, e as reabilita antes de deixá-la.

DESVANTAGENS:

- Processo pode esquecer de reabilitar as interrupções;
- Em sistemas com várias UCPs, desabilitar interrupções em uma UCP não evita que as outras acessem a memória compartilhada.

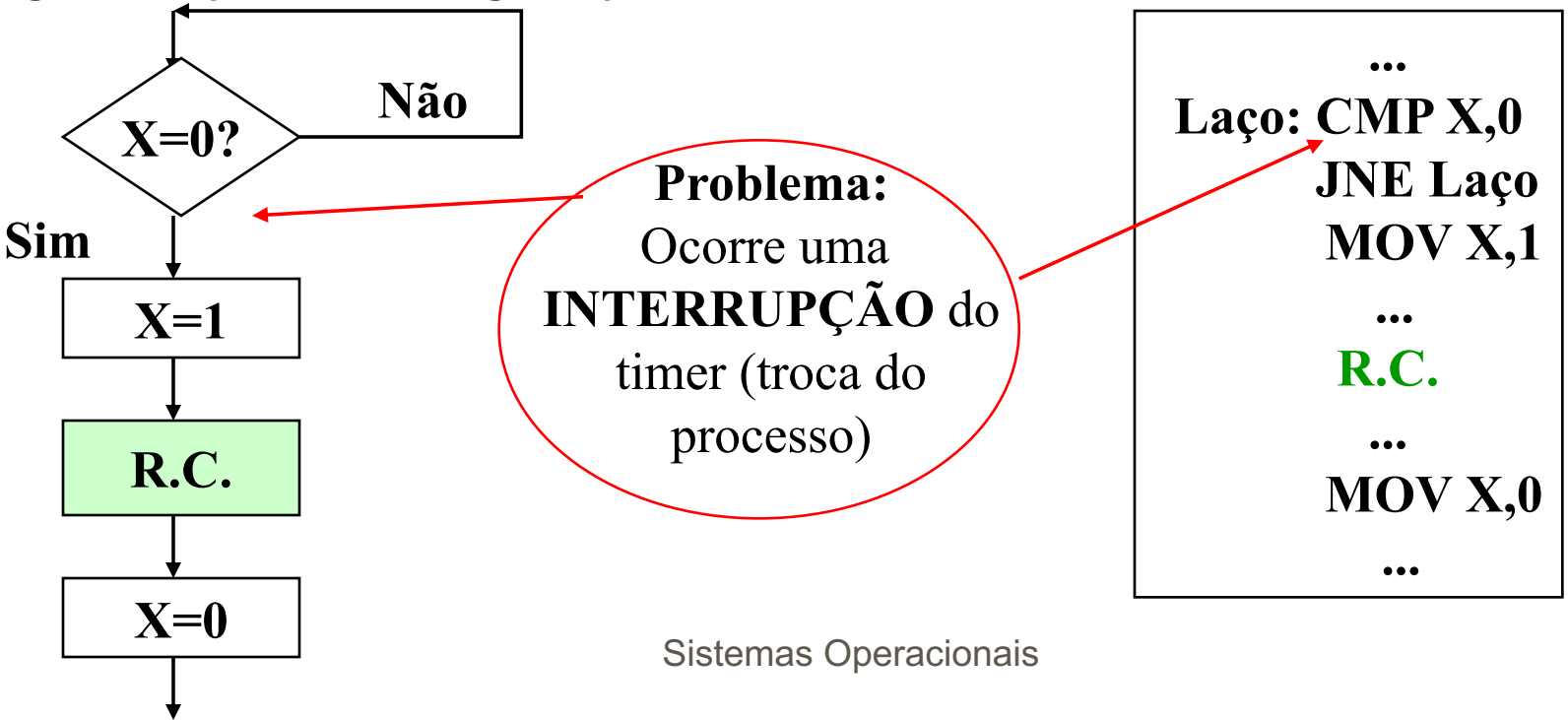
CONCLUSÃO: é útil que o kernel tenha o poder de desabilitar interrupções, mas não é apropriado que os processos de usuário usem este método de exclusão mútua.

Processos

Exclusão Mútua com Espera Ocupada

Variáveis de Trava

Consiste no uso de uma **variável, compartilhada, de trava**. Se a variável está em zero, significa que nenhum processo está na R.C., e "1" significa que existe algum processo na R.C.



Processos

Exclusão Mútua com Espera Ocupada

Instrução TSL (Test and Set Lock)

- Esta solução é implementada com **uso do hardware**.

- Muitos computadores possuem uma instrução especial, chamada **TSL (test and set lock)**, que funciona assim: ela lê o conteúdo de uma palavra de memória e armazena um valor diferente de zero naquela posição.

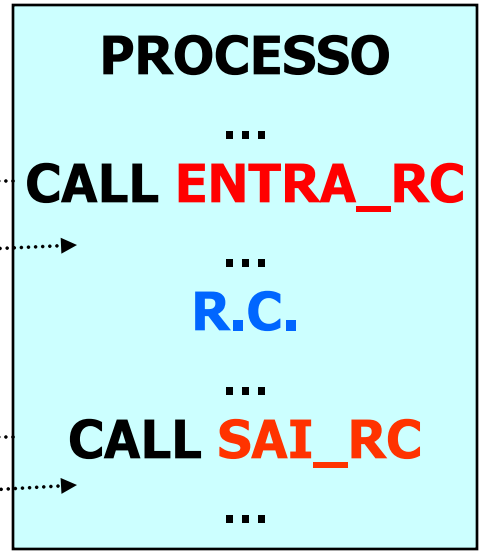
- **Em sistemas multiprocessados:** esta instrução trava o barramento de memória, proibindo outras UCPs de acessar a memória até ela terminar.

Processos

- **Exclusão Mútua com Espera Ocupada**
- **Instrução TSL (Test and Set Lock) - Exemplo**

```
ENTRA_RC:
    TSL reg, flag ; copia flag para reg
                  ; e coloca 1 em flag
    CMP reg,0     ; flag era zero?
    JNZ ENTRA_RC  ; se a trava não
                  ; estava ligada,
                  ; volta ao laço
    RET

SAI_RC:
    MOV flag,0  ;   desliga flag
    RET
```



Processos

Exclusão Mútua com Espera Ocupada

Considerações Finais

- **Espera Ocupada:** quando um processo deseja entrar na sua região crítica, ele verifica se a entrada é permitida. Se não for, o processo ficará em um laço de espera, até entrar.

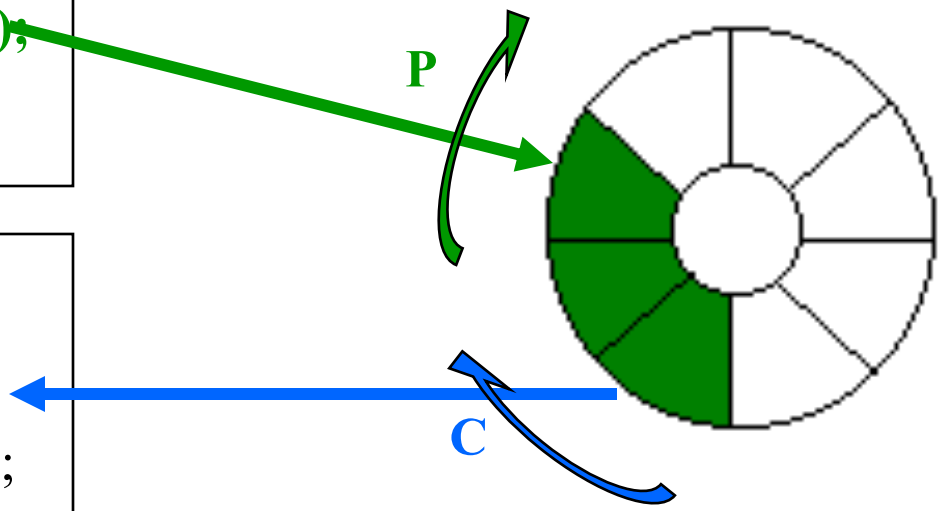
- **Desvantagens:**
  - desperdiça tempo de UCP;
  - pode provocar "**bloqueio perpétuo**" (deadlock) em sistemas com prioridades.

Processos

- **O Problema do Produtor/Consumidor atuando sobre um Buffer Circular**

Produtor
Ciclo
produz(mensagem);
deposita(mensagem);
...
Fim-ciclo

Consumidor
Ciclo
retira(mensagem);
consome(mensagem);
...
Fim-ciclo



Buffer circular com

N elementos

Processos

- **O Problema do Produtor/Consumidor atuando sobre um Buffer Circular**

Restrições do Problema:

- o produtor não deve exceder a capacidade finita do buffer;
- o consumidor não poderá consumir mensagens mais rapidamente do que forem produzidas;
- as mensagens devem ser retiradas do buffer na mesma ordem que forem colocadas;
- restrição de exclusão mútua no acesso ao buffer circular.

Processos

Exemplo do Problema do Produtor/Consumidor usando Sleep e Wakeup

Para os casos extremos de ocupação do buffer (cheio/vazio), deverão funcionar as seguintes **regras de sincronização**:

- se o produtor tentar depositar uma mensagem no **buffer cheio**, ele será suspenso até que o consumidor retire pelo menos uma mensagem do buffer;
- se o consumidor tenta retirar uma mensagem do **buffer vazio**, ele será suspenso até que o produtor deposite pelo menos uma mensagem no buffer.

Processos

Exemplo do Problema do Produtor/Consumidor usando Sleep e Wakeup

```
#define N 100
int contador = 0;
```

```
produtor()
{
  while(TRUE)
   {
     produz_item();
     if (contador==N)    Sleep();
     deposita_item();
     contador + = 1;
     if (contador==1)
                Wakeup(consumidor);
    }
}
```

```
consumidor()
{
  while(TRUE)
   {
     if (contador==0)      Sleep();
     retira_item();
     contador - = 1;
     if (contador==N-1)
                Wakeup(produtor);
    consome_item();
   }
}
```

interrupção

Processos

- **Exemplo do Problema do Produtor/Consumidor usando Sleep e Wakeup**

Problema: pode ocorrer uma condição de corrida, se a variável contador for utilizada sem restrições.

Solução: Criar-se um **"bit de wakeup".** Quando um Wakeup é mandado à um processo já acordado, este bit é setado. Depois, quando o processo tenta ir dormir, se o bit de espera de Wakeup estiver ligado, este bit será desligado, e o processo será mantido acordado.