Aula 07
Sistemas Operacionais I

Comunicação e Sincronismo de Processos

– Parte 01

Prof. Julio Cezar Estrella
jcezar@icmc.usp.br
Material adaptado de
Sarita Mazzini Bruschi

Processos

- Introdução
- Escalonamento de Processos
- **Comunicação entre Processos**
  - **Condição de Disputa**
  - **Região Crítica**
  - **Formas de Exclusão Mútua**
  - **Problemas Clássicos**
- Threads
- Deadlock

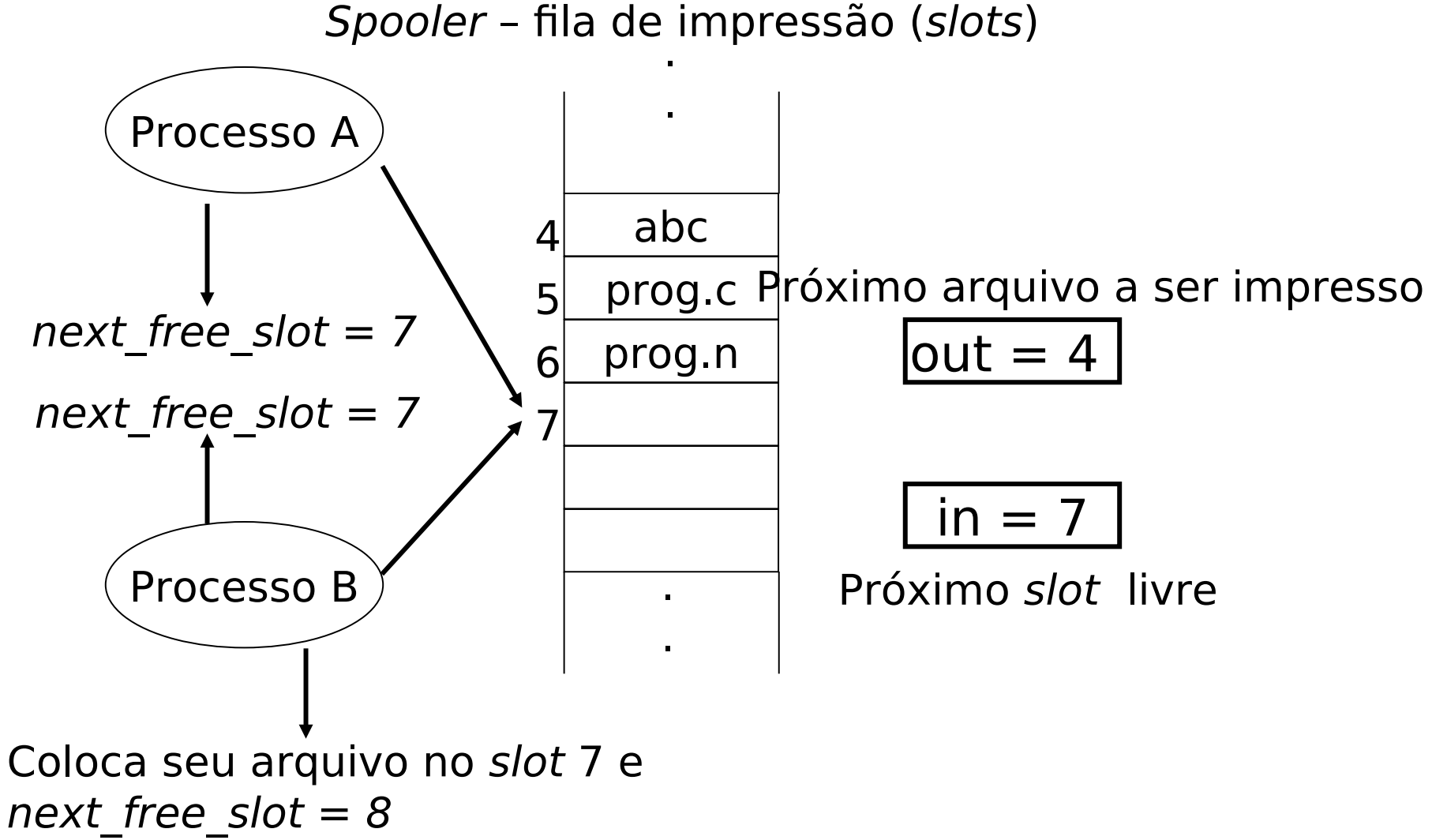
Comunicação de Processos

- Processos precisam se comunicar;
- Processos competem por recursos
- Três aspectos importantes:
  - Como um processo passa informação para outro processo;
  - Como garantir que processos não invadam espaços uns dos outros;
  - Dependência entre processos: seqüência adequada;

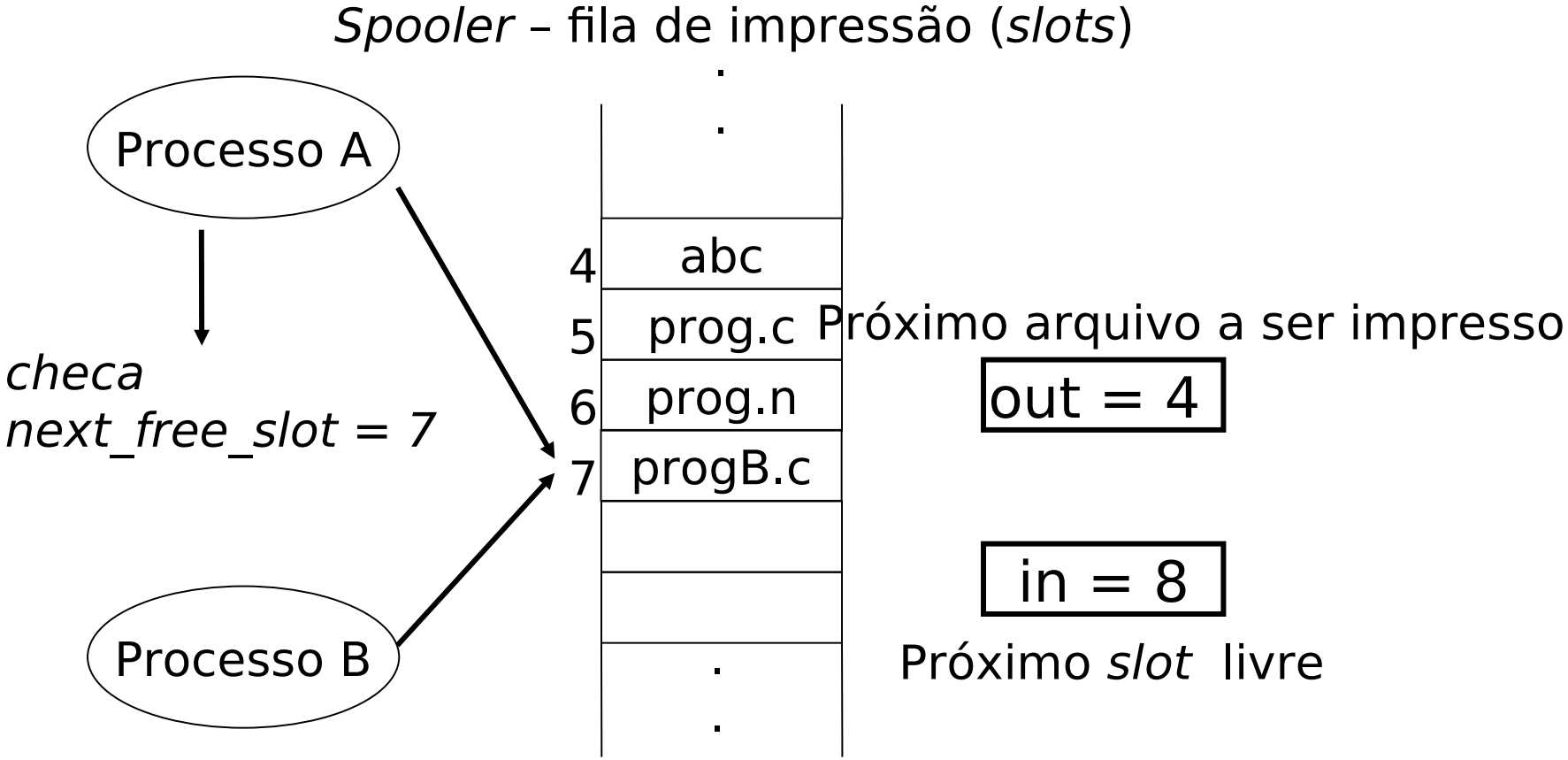
Comunicação de Processos – *Race Conditions*

- *Race Conditions*: processos acessam recursos compartilhados concorrentemente;
  - Recursos: memória, arquivos, impressoras, discos, variáveis;
- Ex.: Impressão: quando um processo deseja imprimir um arquivo, ele coloca o arquivo em um local especial chamado ***spooler*** (tabela). Um outro processo, chamado ***printer spooler***, checa se existe algum arquivo a ser impresso. Se existe, esse arquivo é impresso e retirado do *spooler*. Imagine dois processos que desejam ao mesmo tempo imprimir um arquivo...

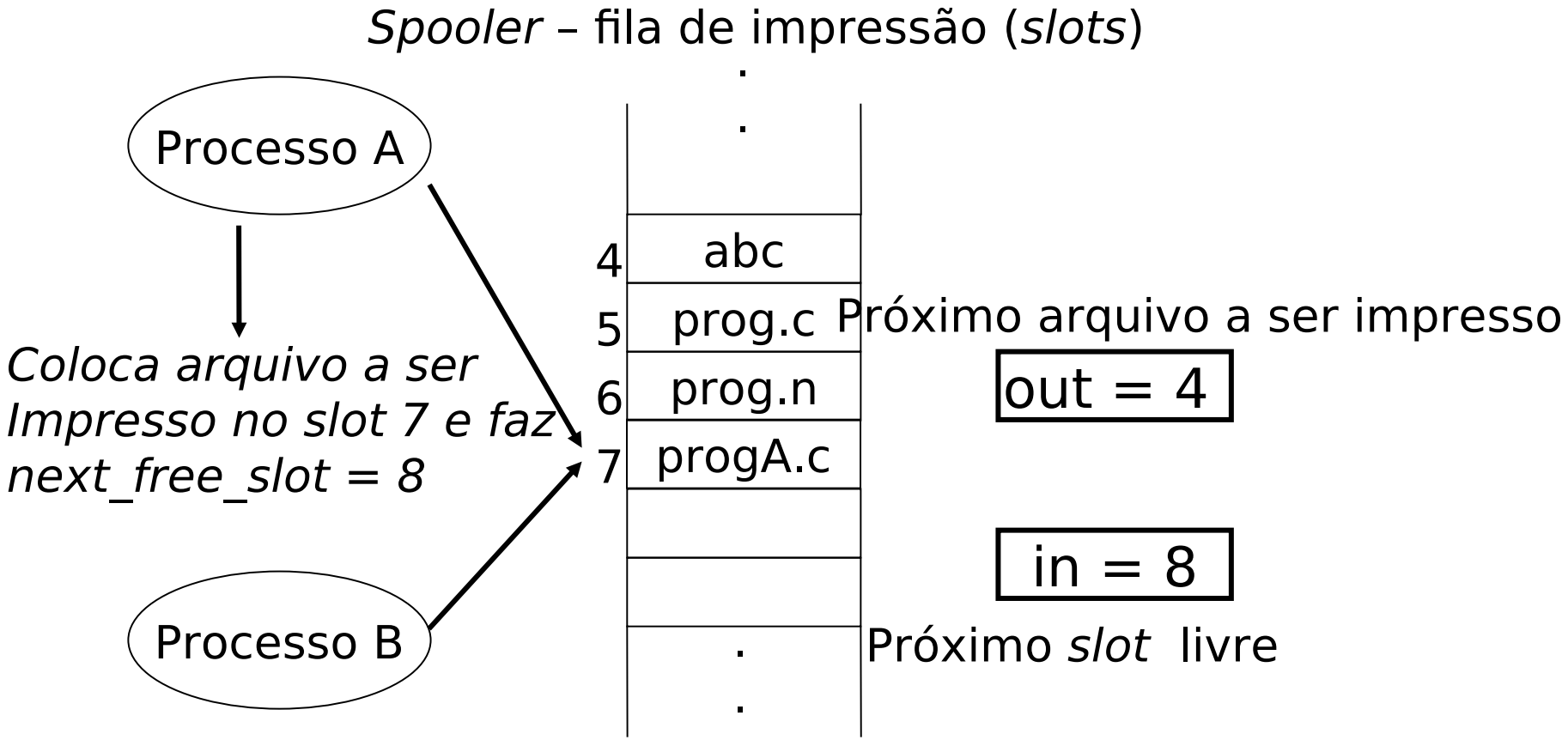
Comunicação de Processos - *Race Conditions*



Comunicação de Processos - *Race Conditions*



Comunicação de Processos - *Race Conditions*



Processo B nunca receberá sua impressão!!!!!

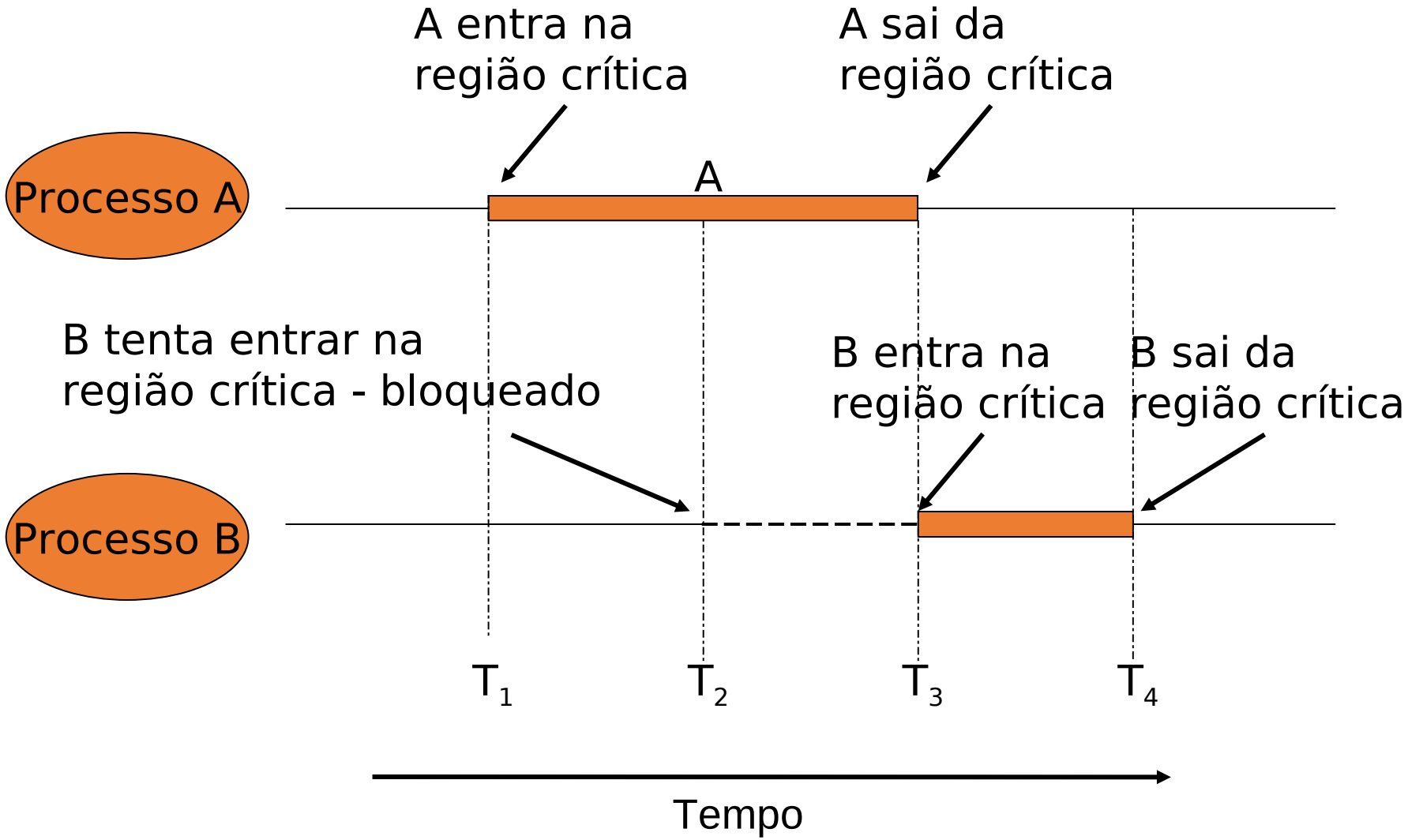
Comunicação de Processos – Regiões Críticas

- Como solucionar problemas de *Race Conditions*???
  - Proibir que mais de um processo leia ou escreva em recursos compartilhados concorrentemente (ao “mesmo tempo”)
    - **Recursos compartilhados → regiões críticas**;
  - Exclusão mútua: garantir que um processo não terá acesso à uma região crítica quando outro processo está utilizando essa região;

Comunicação de Processos – Exclusão Mútua

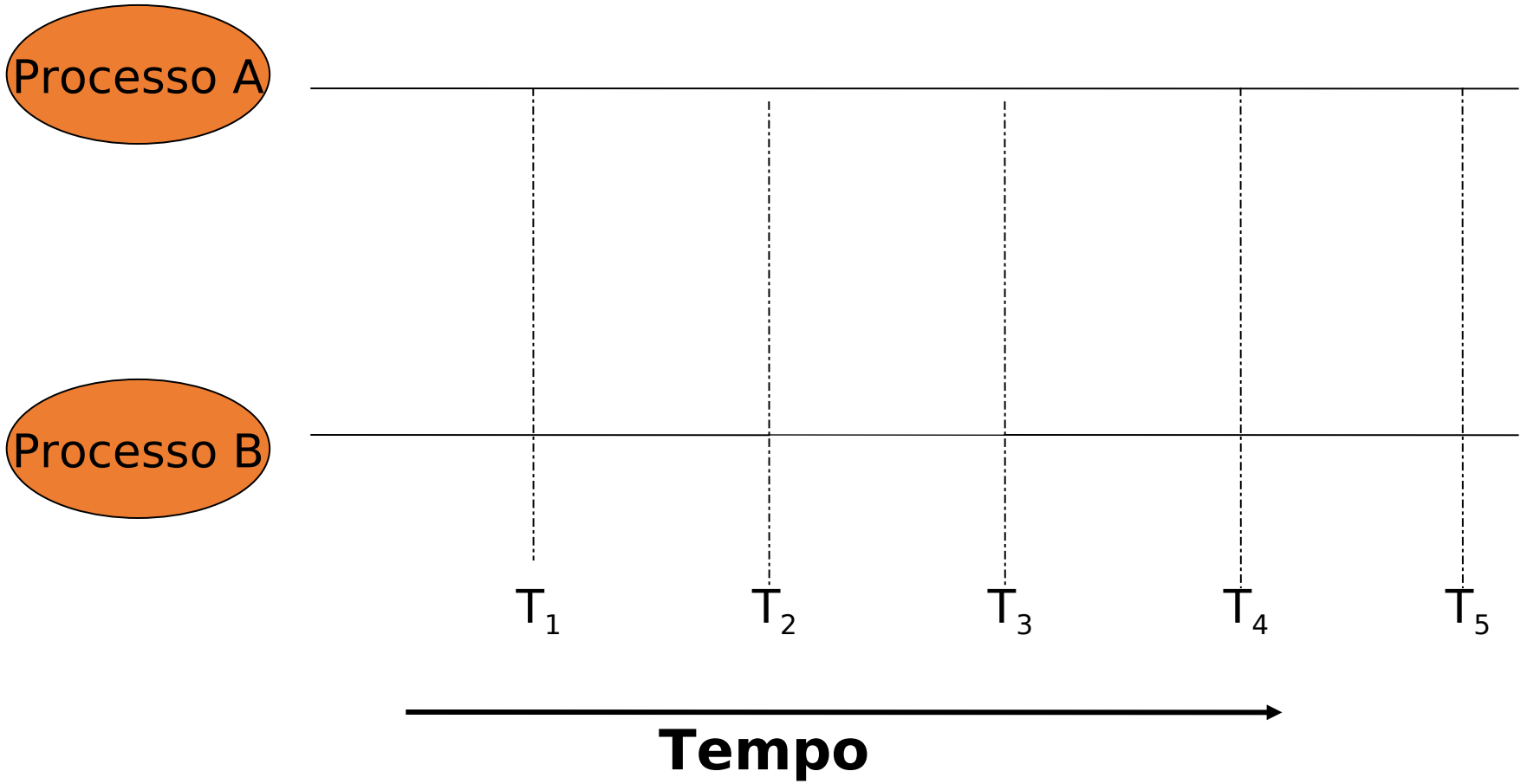
- Quatro condições para uma boa solução:
  1. Dois processos não podem estar simultaneamente na mesma região crítica;
  2. Nenhuma restrição deve ser feita com relação à CPU;
  3. Processos que não estão em regiões críticas não podem bloquear outros processos que desejam utilizar regiões críticas;
  4. Processos não podem esperar para sempre para acessarem regiões críticas;

Comunicação de Processos – Exclusão Mútua



Comunicação de Processos – Exclusão Mútua

Volta a situação inicial!!!!



Soluções

- Exclusão Mútua:
  - **Espera Ocupada**;
  - Primitivas *Sleep/Wakeup*;
  - Semáforos;
  - Monitores;
  - Passagem de Mensagem;

Comunicação de Processos – Exclusão Mútua

- Espera Ocupada (*Busy Waiting)*: constante checagem por algum valor;
- Algumas soluções para Exclusão Mútua com Espera Ocupada:
  - Desabilitar interrupções;
  - Variáveis de Travamento (*Lock)*;
  - Estrita Alternância (*Strict Alternation)*;
  - Solução de Peterson e Instrução TSL;

Comunicação de Processos – Exclusão Mútua

- Desabilitar interrupções:
  - Processo desabilita todas as suas interrupções ao entrar na região crítica e habilita essas interrupções ao sair da região crítica;
  - Com as interrupções desabilitadas, a CPU não realiza chaveamento entre os processos;
    - Viola condição 2 (Nenhuma restrição deve ser feita com relação à CPU);
  - Não é uma solução segura, pois um processo pode não habilitar novamente suas interrupções e não ser finalizado;
    - Viola condição 4 (Processos não podem esperar para sempre para acessarem regiões críticas);

Comunicação de Processos – Exclusão Mútua

- Variáveis *Lock*:
  - O processo que deseja utilizar uma região crítica atribuí um valor a uma variável chamada *lock*;
  - Se a variável está com valor 0 (zero) significa que nenhum processo está na região crítica; Se a variável está com valor 1 (um) significa que existe um processo na região crítica;
  - Apresenta o mesmo problema do exemplo do *spooler de impressão*;

Comunicação de Processos – Exclusão Mútua

- Variáveis *Lock* - Problema:
  - Suponha que um processo A leia a variável *lock* com valor 0;
  - Antes que o processo A posso alterar a variável para o valor 1, um processo B é escalonado e altera o valor de *lock* para 1;
  - Quando o processo A for escalonado novamente, ele altera o valor de *lock* para 1, e ambos os processos estão na região crítica;
    - Viola condição 1 (Dois processos não podem estar simultaneamente em regiões críticas);

Comunicação de Processos – Exclusão Mútua

- Variáveis *Lock:* ***lock==0;***

```
while(true){
    while(lock!=0); //loop
    lock=1;
    critical_region();
    lock=0;
    non-critical_region();
}
```

Processo A

```
while(true){
    while(lock!=0); //loop
    lock=1;
    critical_region();
    lock=0;
    non-critical_region();
}
```

Processo B

Comunicação de Processos – Exclusão Mútua

- *Strict Alternation*:
  - Fragmentos de programa controlam o acesso às regiões críticas;
  - Variável `turn`, inicialmente em `0`, estabelece qual processo pode entrar na região crítica;

```
while (TRUE) {
    while (turn!=0); //loop
        critical_region();
        turn = 1;
        noncritical region();}
```

(Processo A)
`turn` **0**

```
while (TRUE){
    while (turn!=1); //loop
        critical_region();
        turn = 0;
        noncritical region();}
```

(Processo B)
`turn` **1**

Comunicação de Processos – Exclusão Mútua

- Problema do *Strict Alternation*:
  1. Suponha que o Processo B é mais rápido e sai da região crítica;
  2. Ambos os processos estão fora da região crítica e `turn` com valor 0;
  3. O processo A termina antes de executar sua região não crítica e retorna ao início do *loop*; Como o `turn` está com valor zero, o processo A entra novamente na região crítica, enquanto o processo B ainda está na região não crítica;
  4. Ao sair da região crítica, o processo A atribui o valor 1 à variável `turn` e entra na sua região não crítica;

Comunicação de Processos – Exclusão Mútua

- Problema do *Strict Alternation*:
  5. Novamente ambos os processos estão na região não crítica e a variável `turn` está com valor 1;
  6. Quando o processo A tenta novamente entrar na região crítica, não consegue, pois `turn` ainda está com valor 1;
  7. **Assim, o processo A fica bloqueado pelo processo B que <u>NÃO</u> está na sua região crítica, violando a condição 3**;

Comunicação de Processos – Exclusão Mútua

- Solução de Peterson e Instrução TSL (*Test and Set Lock*):
  - Uma variável (ou programa) é utilizada para bloquear a entrada de um processo na região crítica quando um outro processo está na região;
  - Essa variável é compartilhada pelos processos que concorrem pelo uso da região crítica;
  - Ambas as soluções possuem fragmentos de programas que controlam a entrada e a saída da região crítica;

Comunicação de Processos – Exclusão Mútua

- Solução de Peterson

```
#define FALSE 0
#define TRUE 1
#define N 2

int turn;
int interested[N];

void enter_region(int process)
{
  int other;

  other = 1 - process;
  interested[process] = TRUE;
  turn = process;
  while (turn ==  process) && interested[other] == TRUE) ;
}

void leave_region(int process)
{
  interested[process] = FALSE;
}
```

Comunicação de Processos – Exclusão Mútua

- Instrução TSL: utiliza registradores do hardware;
  - TSL RX, LOCK; (lê o conteúdo de ***lock*** em RX, e armazena um valor diferente de zero (0) em ***lock*** – operação indivisível);
  - ***Lock*** é compartilhada
    - Se ***lock***==0, então região crítica “liberada”.
    - Se ***lock***<>0, então região crítica “ocupada”.

```
enter_region:
    TSL REGISTER, LOCK | Copia lock para reg. e lock=1
    CMP REGISTER, #0   | lock valia zero?
    JNE enter_region   | Se sim, entra na região crítica,
                       | Se não, continua no laço
    RET        | Retorna para o processo chamador

leave_region
     MOVE LOCK, #0 | lock=0
     RET       | Retorna para o processo chamador
```