Sistemas Operacionais

Processos - Parte 1

Prof. Dr. Fábio Rodrigues de la Rocha

UTFPR

Um aspecto importante no estudo sobre sistemas operacionais é o gerenciamento de processos. Mas afinal, o que é um **processo** ?
Quando escrevemos um programa em alguma linguagem, por exemplo C, devemos compilá-lo. Ao compilar um programa em C gera-se um conjunto de instruções em uma linguagem (microinstruções) que o computador consegue entender.

Quando o processador passa a executar o programa, temos um **processo** executando.
Assim sendo, para todos os efeitos um processo é um programa em execução. A este conceito vamos adicionar o seguinte. um processo é um programa em execução, considerando-se todos os valores dos seus registradores e de suas variáveis.

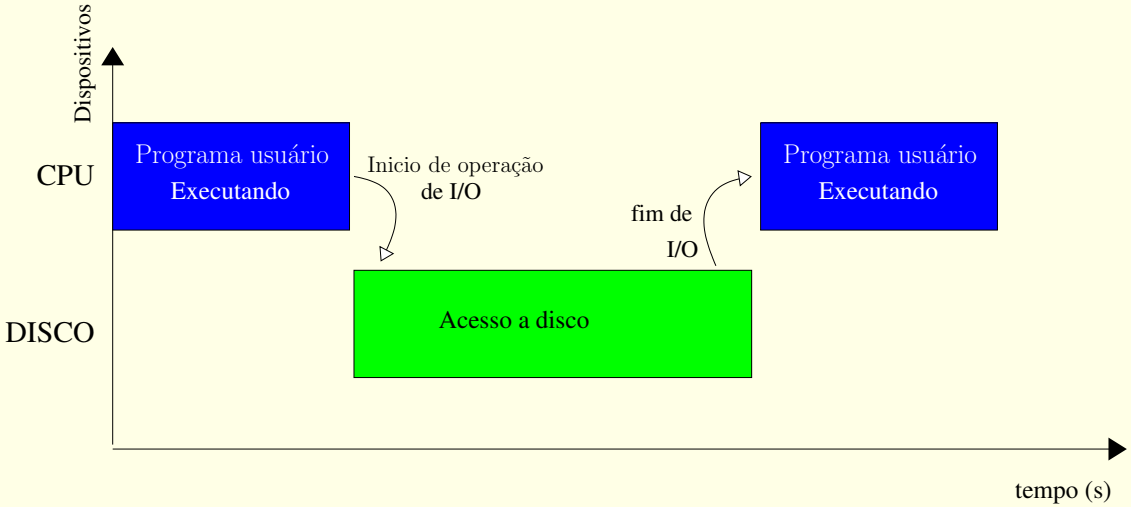
Esse conjunto de instruções é um **programa de computador**. O programa de computador é estático, ou seja composto por um conjunto de instruções como no exemplo abaixo:

```
1 y=0;
2 for (x=0;x<10;x++)
3 {
4    y=y+20;
5 }
```

```
1         CARREGUE #10, A
2         CARREGUE #0,  B
3 REPETE:
4         ADD B,#20
5         DECREMENTA A
6         CMP A,#0
7         SALTA_DIFERENTE REPETE
8 FIM:
```

Monoprogramação

Na monoprogramação temos apenas um processo na memória do computador.

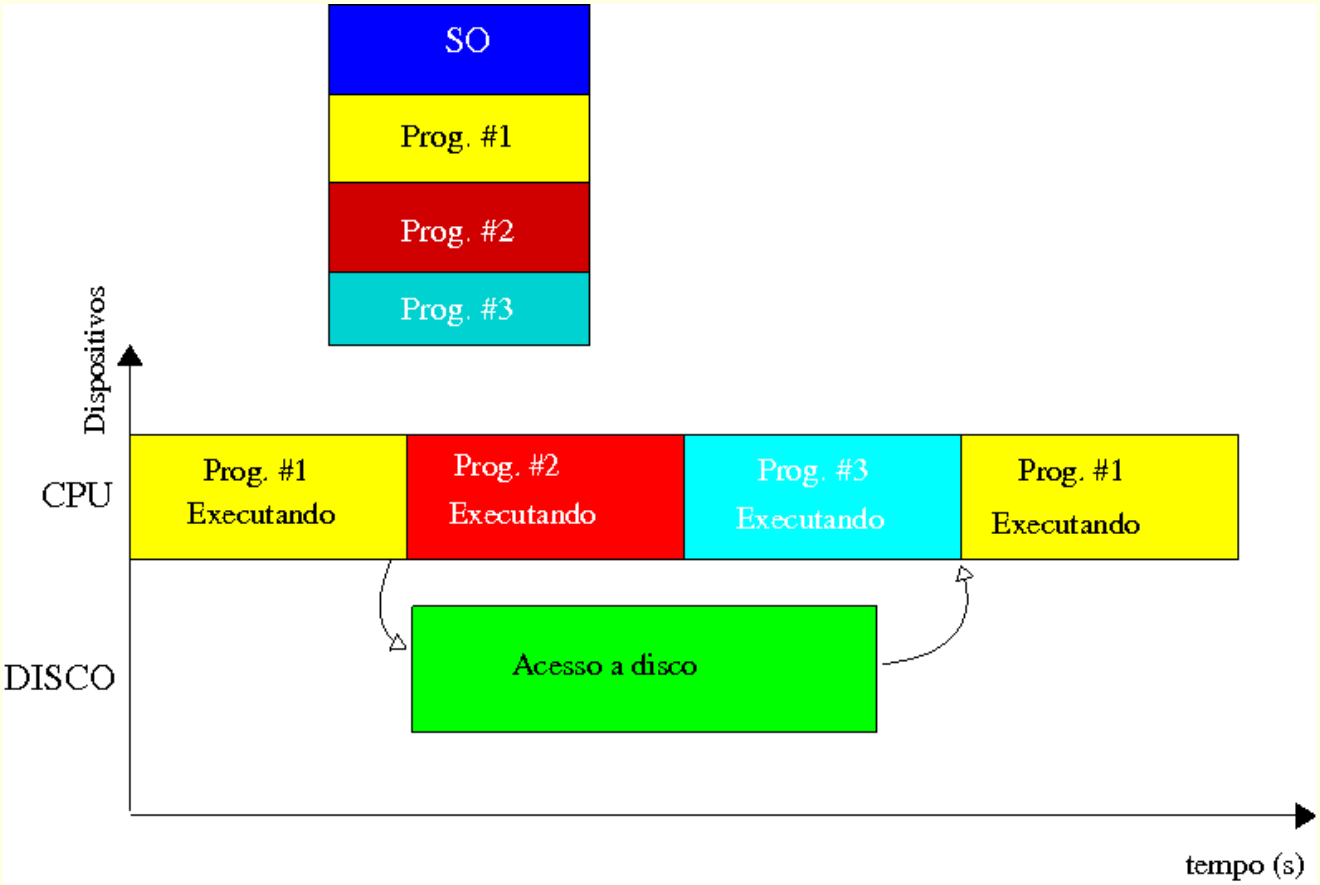



```
1         MOVE A,B
2         MOVE C,D
3         LEIA_DISCO
4 ESPERA:
5         NOP
6         .
7         .
8         NOP
9         ACESSE INFORMACAO
```

Multiprogramação

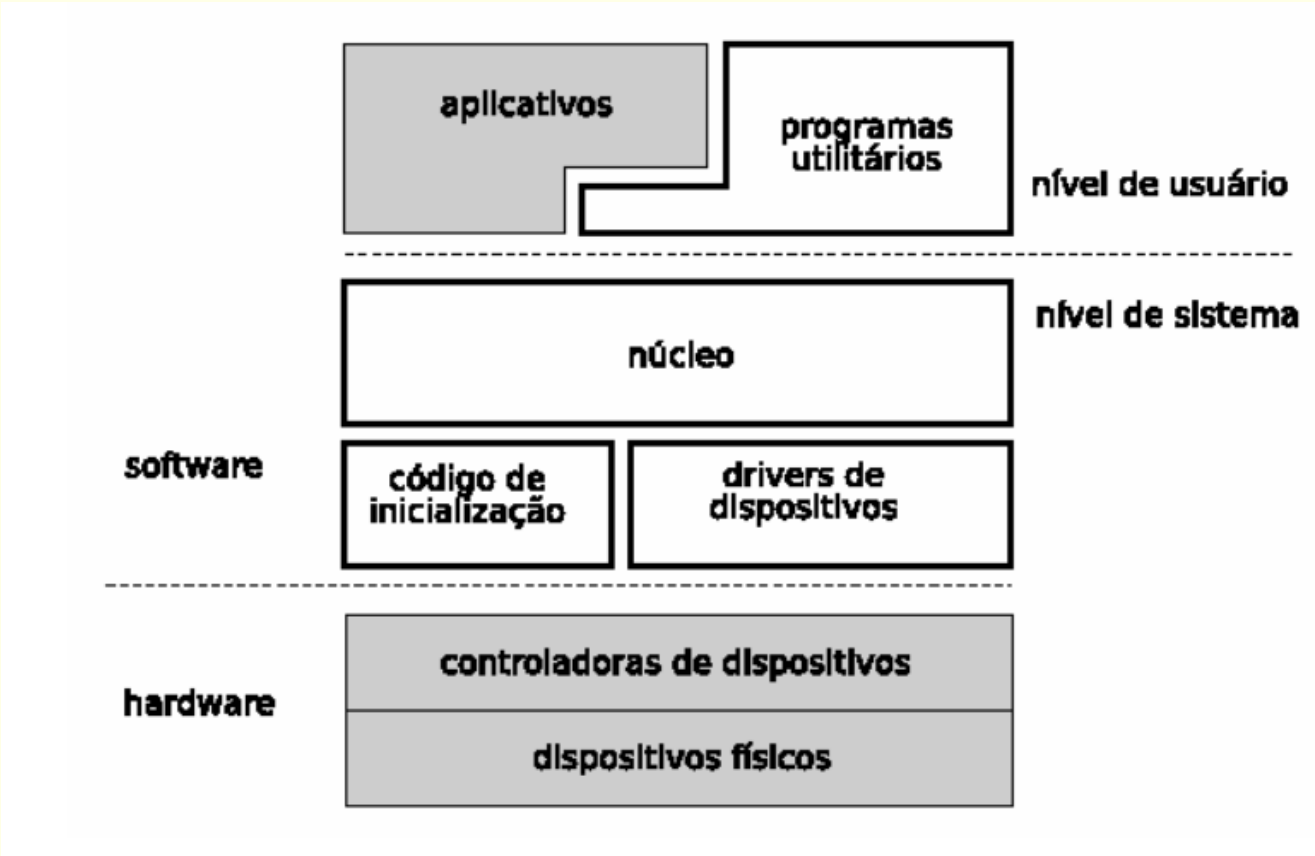
Se existirem vários processos, enquanto um processo espera pelo término da operação de leitura, outro processo pode executar.

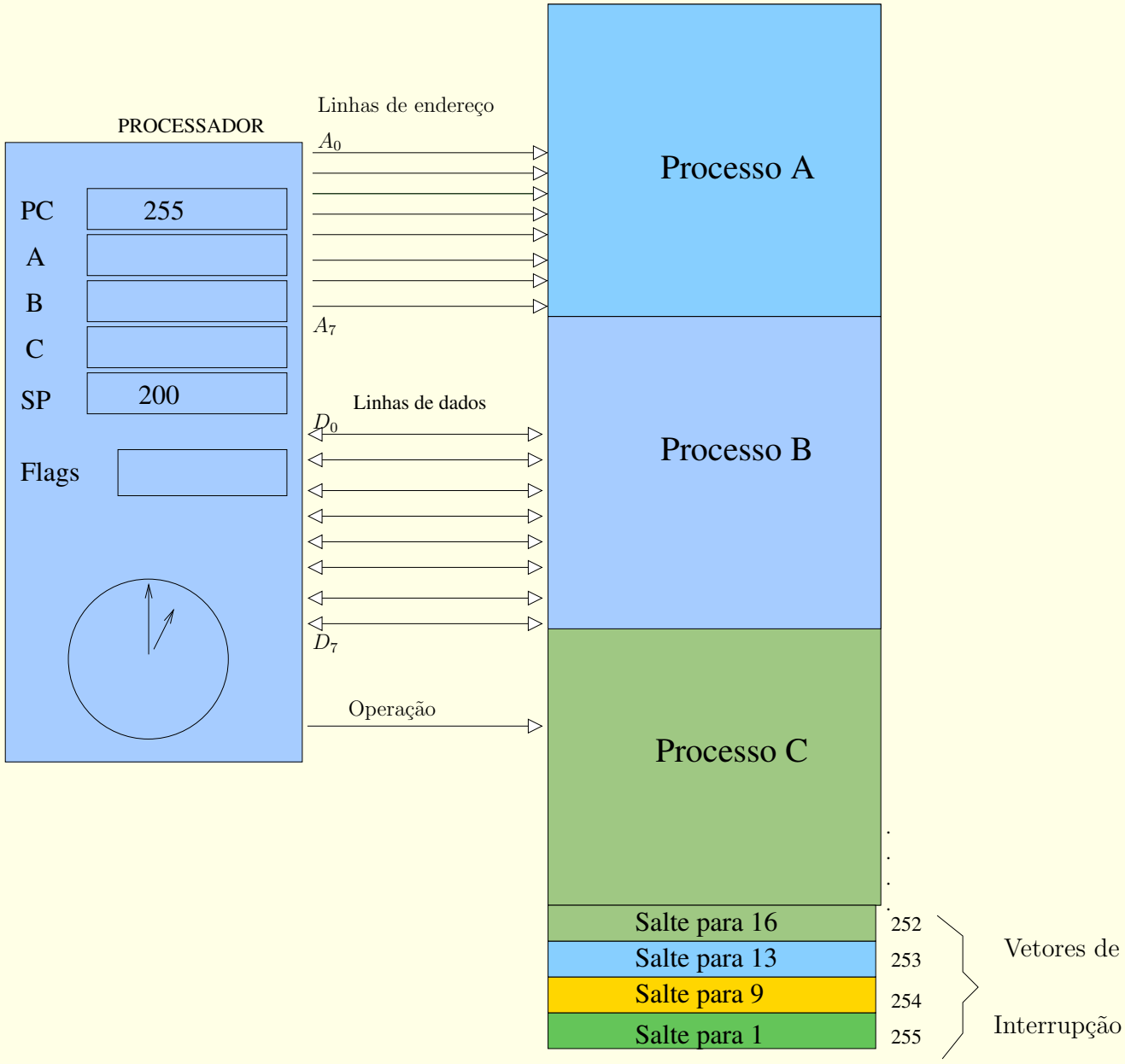
Multiprogramação



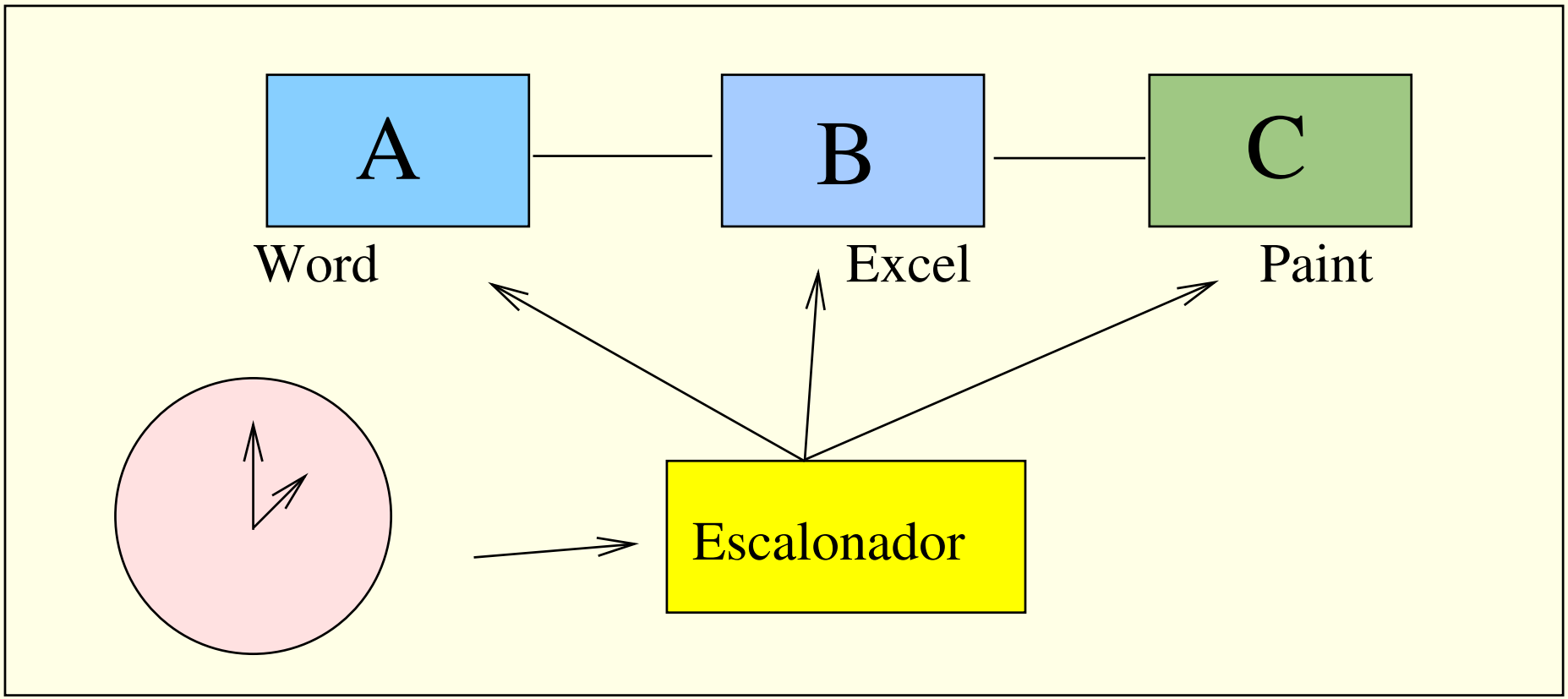
Como é possível a multiprogramação ?

- Interrupções;
- Timer;

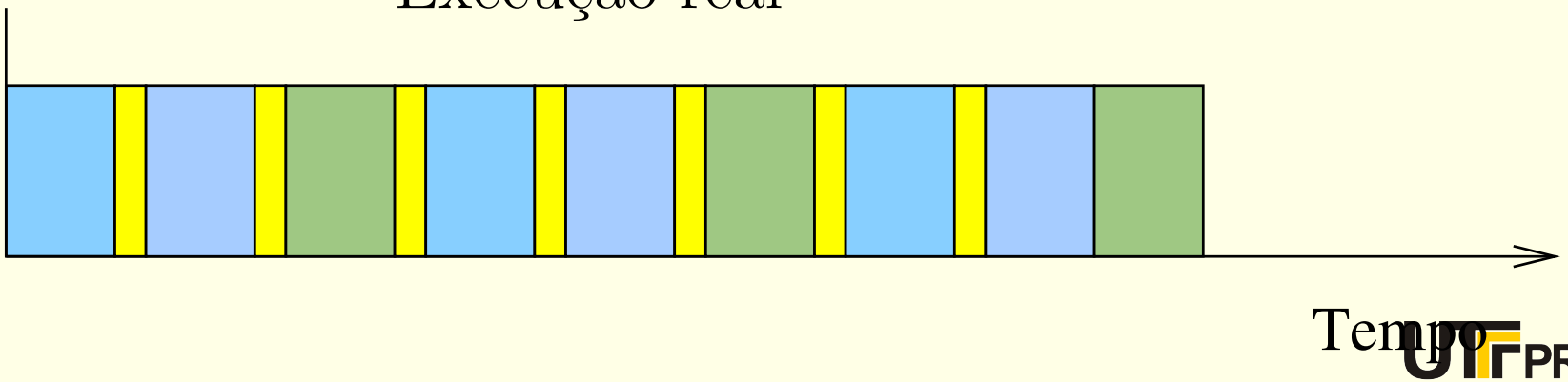



PROCESSADOR
PC
255
A
B
C
SP
200
Flags
Linhas de endereço
A_0
A_7
Linhas de dados
D_0
D_7
Operação
Processo A
Processo B
Processo C
Salte para 16
Salte para 13
Salte para 9
Salte para 1
252
253
254
255
Vetores de
Interrupção

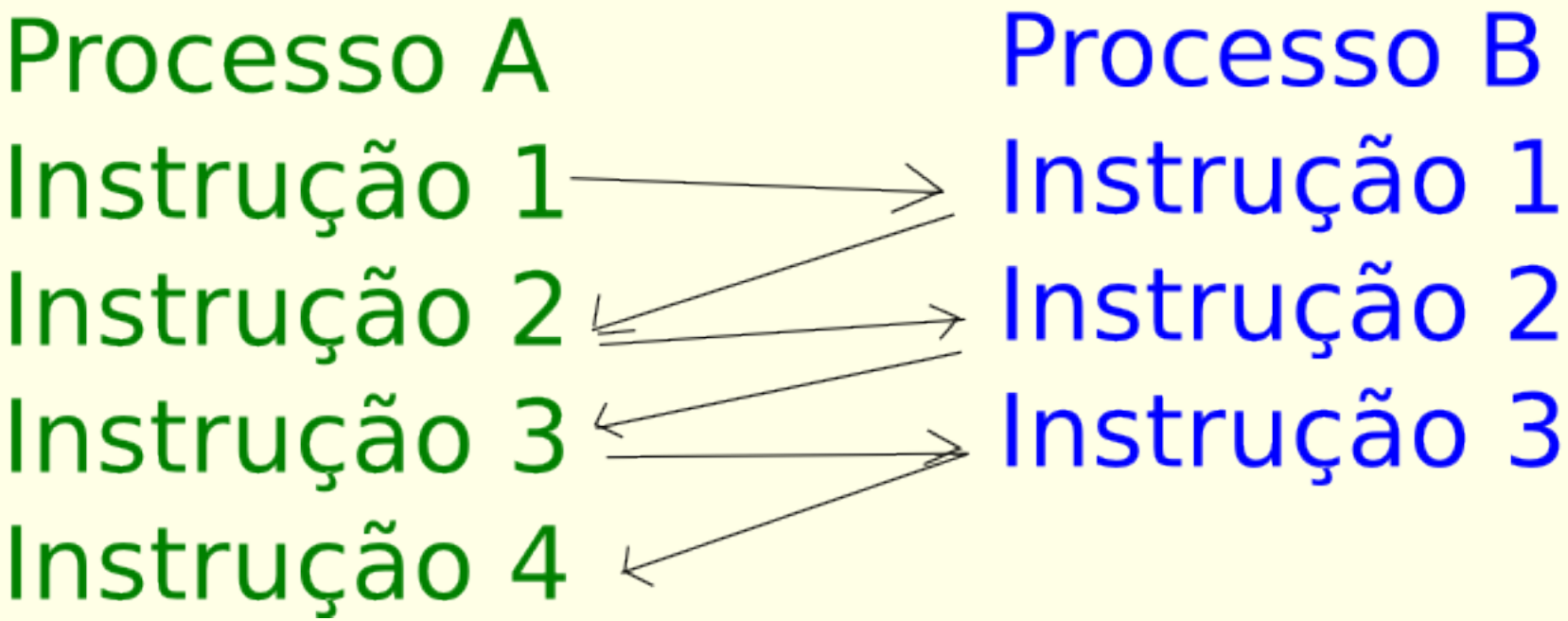
Idealização



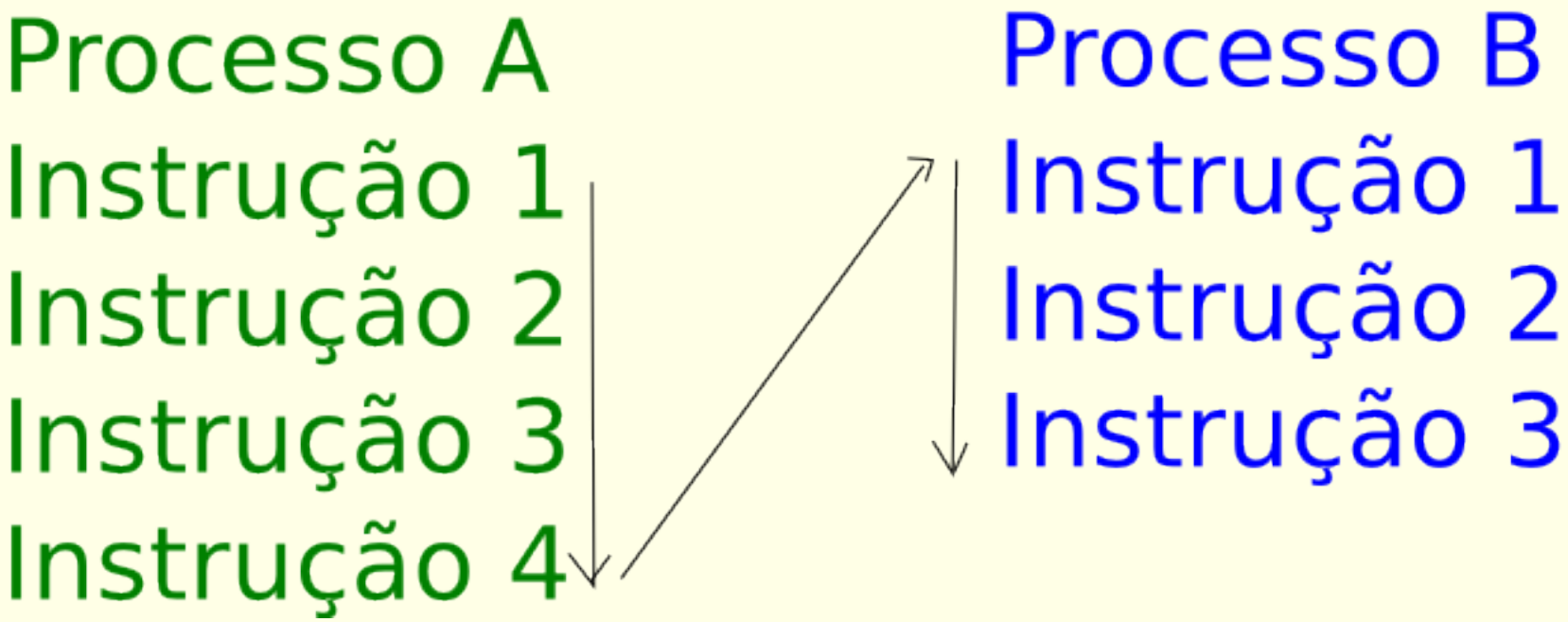
Execução real



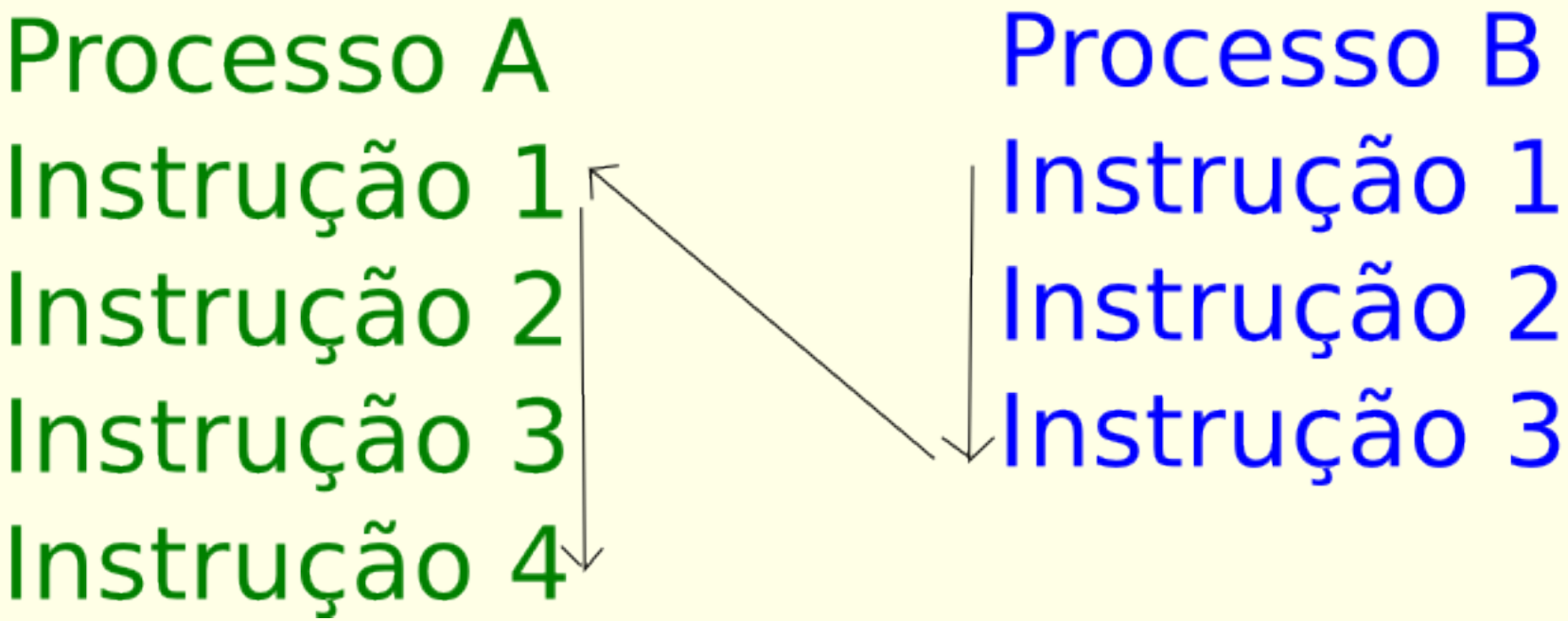
Ordem de execução das instruções



Ordem de execução das instruções



Ordem de execução das instruções



Resumo:

- Diversos processos podem existir na memória do computador, mesmo com apenas um processador;
- Pode-se executar um pouco um processo A e depois passar a executar o processo B e assim por diante;
- Realizando-se essa troca entre processos (**chaveamento de processos**) rapidamente dá a ilusão de vários processos executando simultaneamente.
- Processos não executam sobre controle direto do SO. O sistema operacional (escalonador) só assume quando ocorre uma interrupção ou quando um processo faz uma chamada de sistema (por exemplo ao requisitar uma leitura do disco).

UTFPR

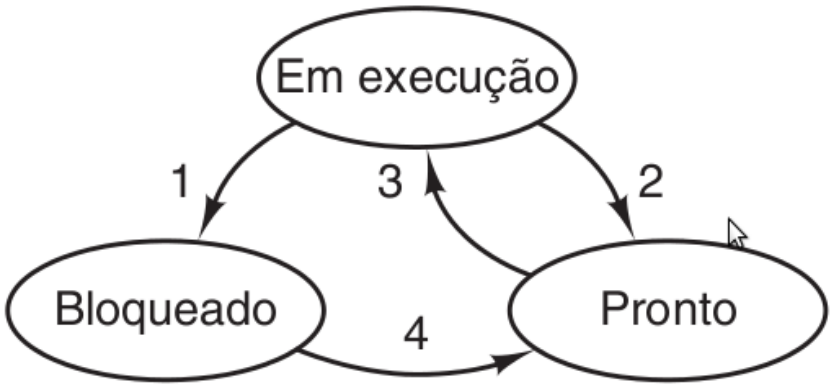
Estado dos processos

Na multiprogramação existem diversos processos na memória e algumas questões podem surgir:

- O que acontece quando um processo faz uma solicitação de chamada de sistema ?
- O que ocorre quando o SO escolhe que um processo deve executar no lugar de outro ?

UTFPR

Estado dos processos



1. O processo bloqueia aguardando uma entrada
2. O escalonador seleciona outro processo
3. O escalonador seleciona esse processo
4. A entrada torna-se disponível

Comunicação entre processos

Até agora, assumimos que os processos são independentes entre si. Isto é, eles não se comunicam.
Onde comunicação pode ser:

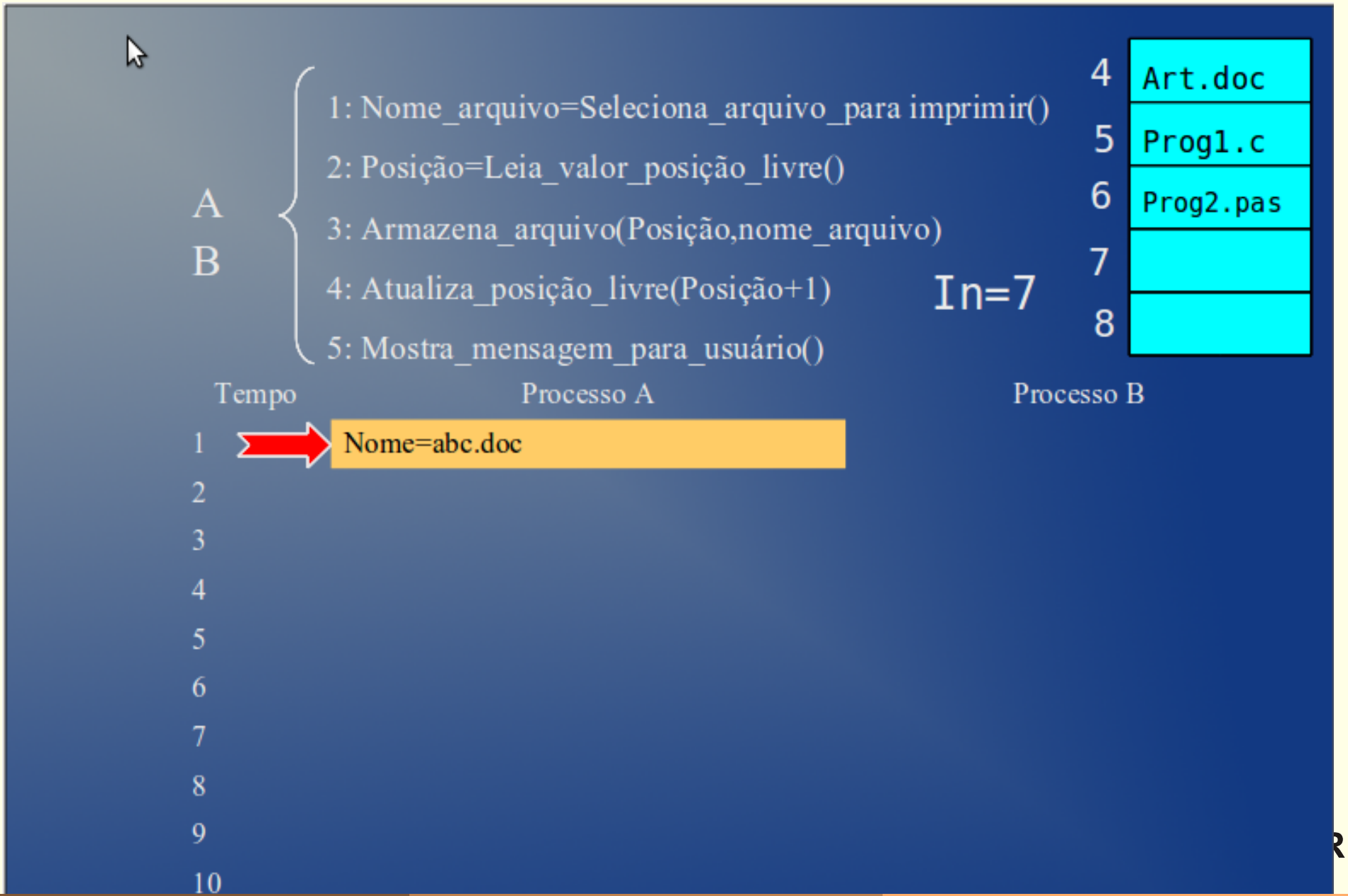
- Quando dois processos acessam a mesma posição de memória (**memória compartilhada**);
- Quando dois processos acessam o mesmo dispositivo.

UTFPR

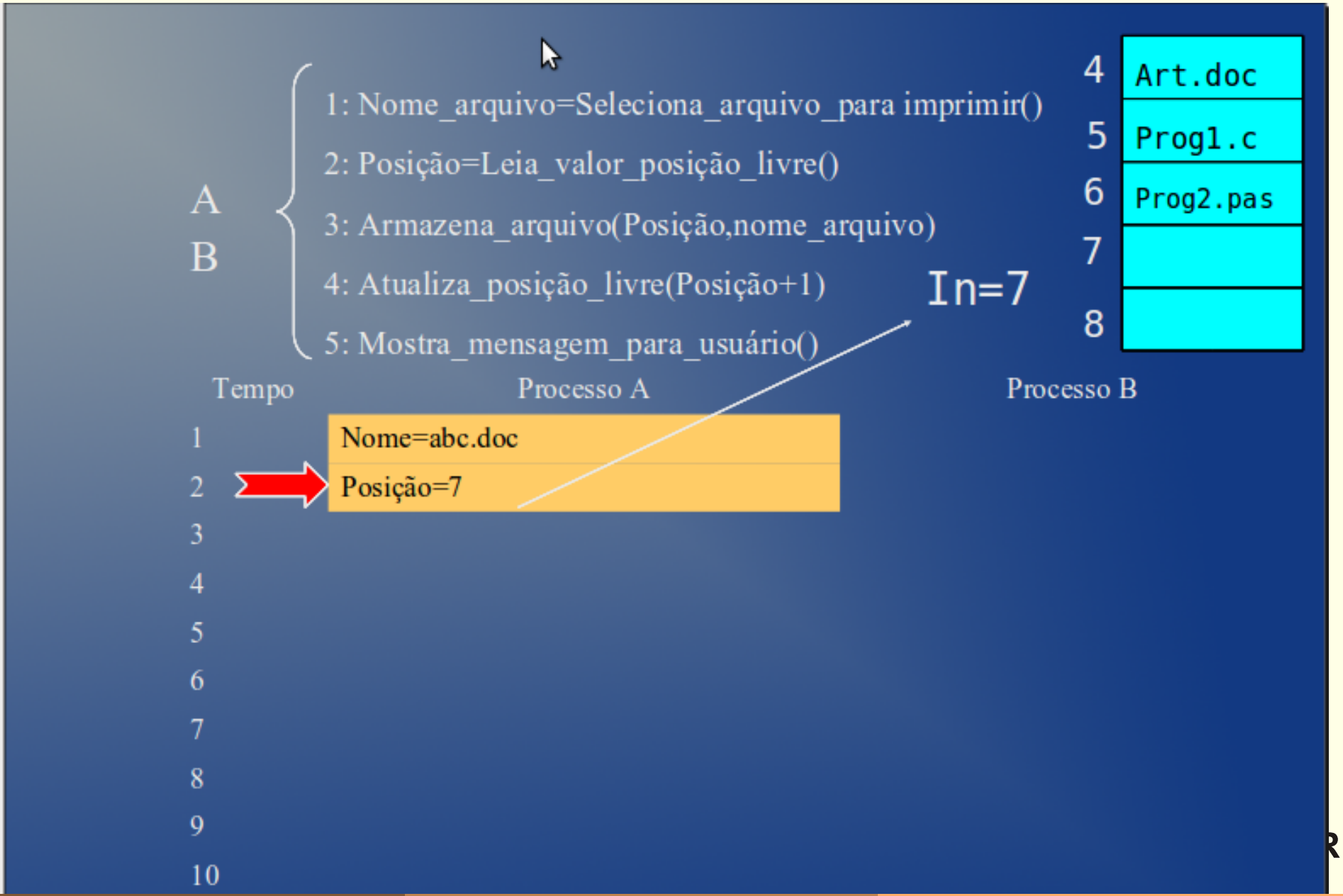
Comunicação entre processos

O que ocorre é que muitas vezes os processos compartilham memória ou um dispositivo. Isso pode fazer com que o resultado deles dependa da ordem em que o escalonador decidiu que eles deveriam executar.

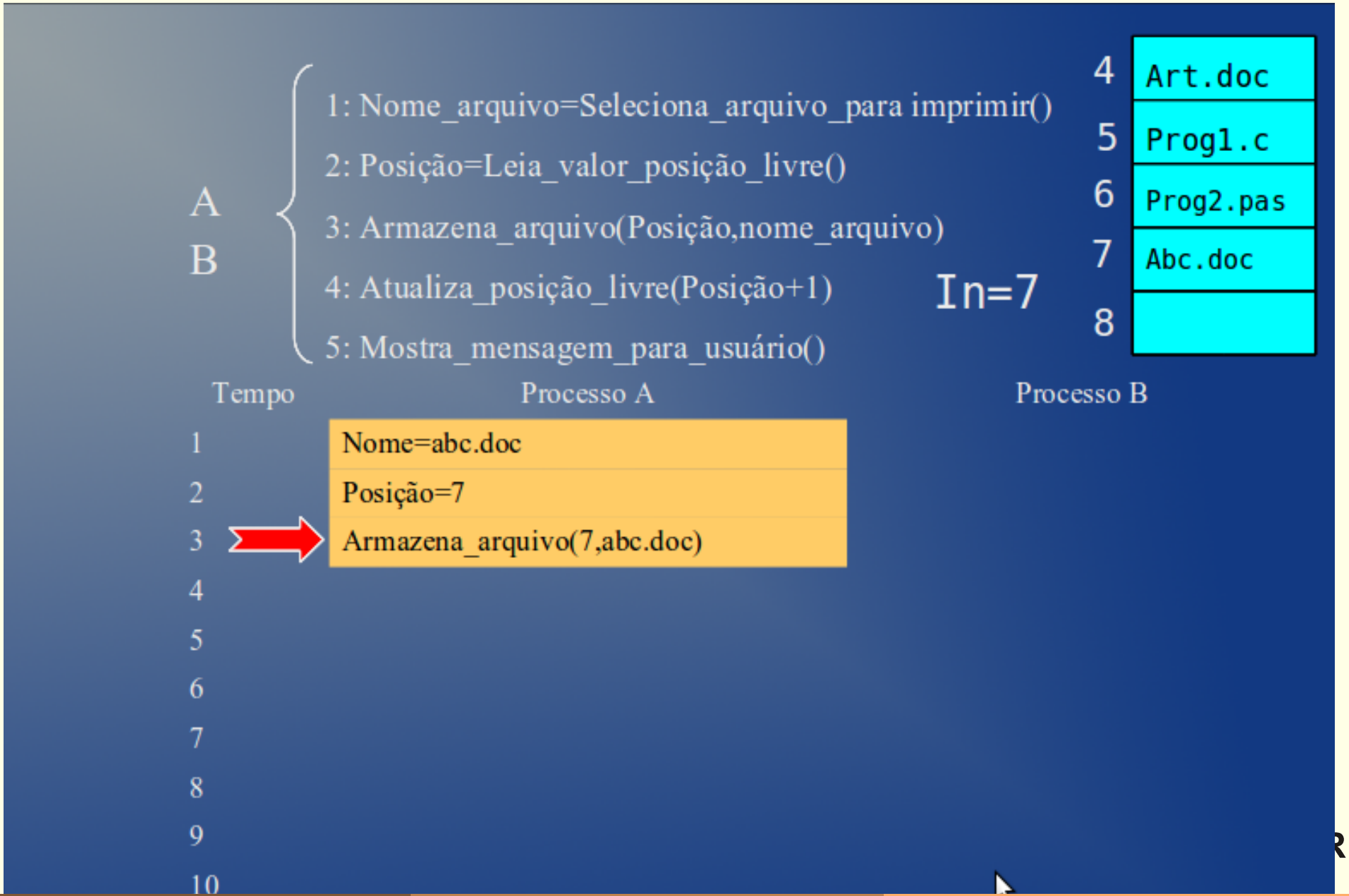
Comunicação entre processos



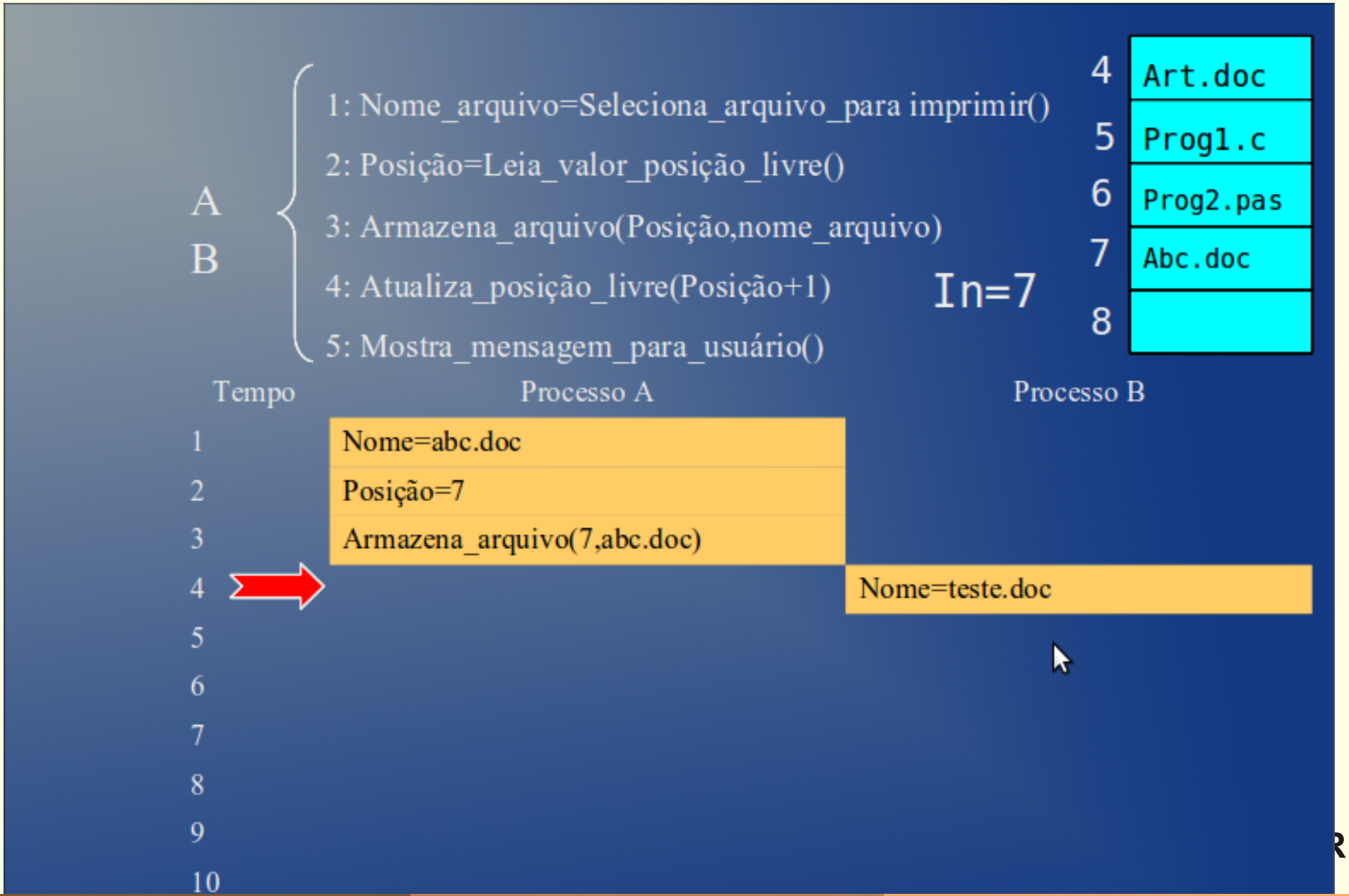
Comunicação entre processos



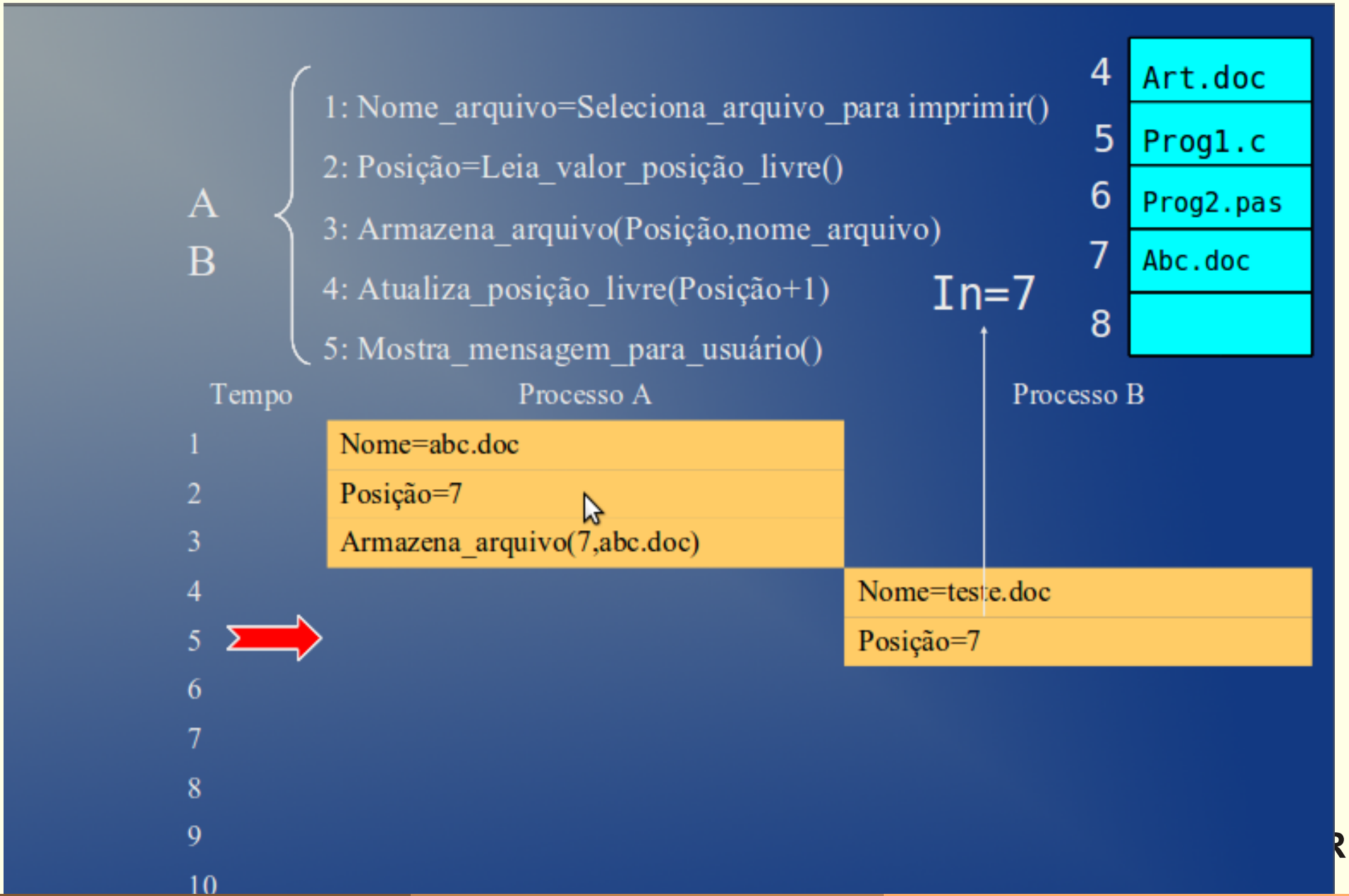
Comunicação entre processos



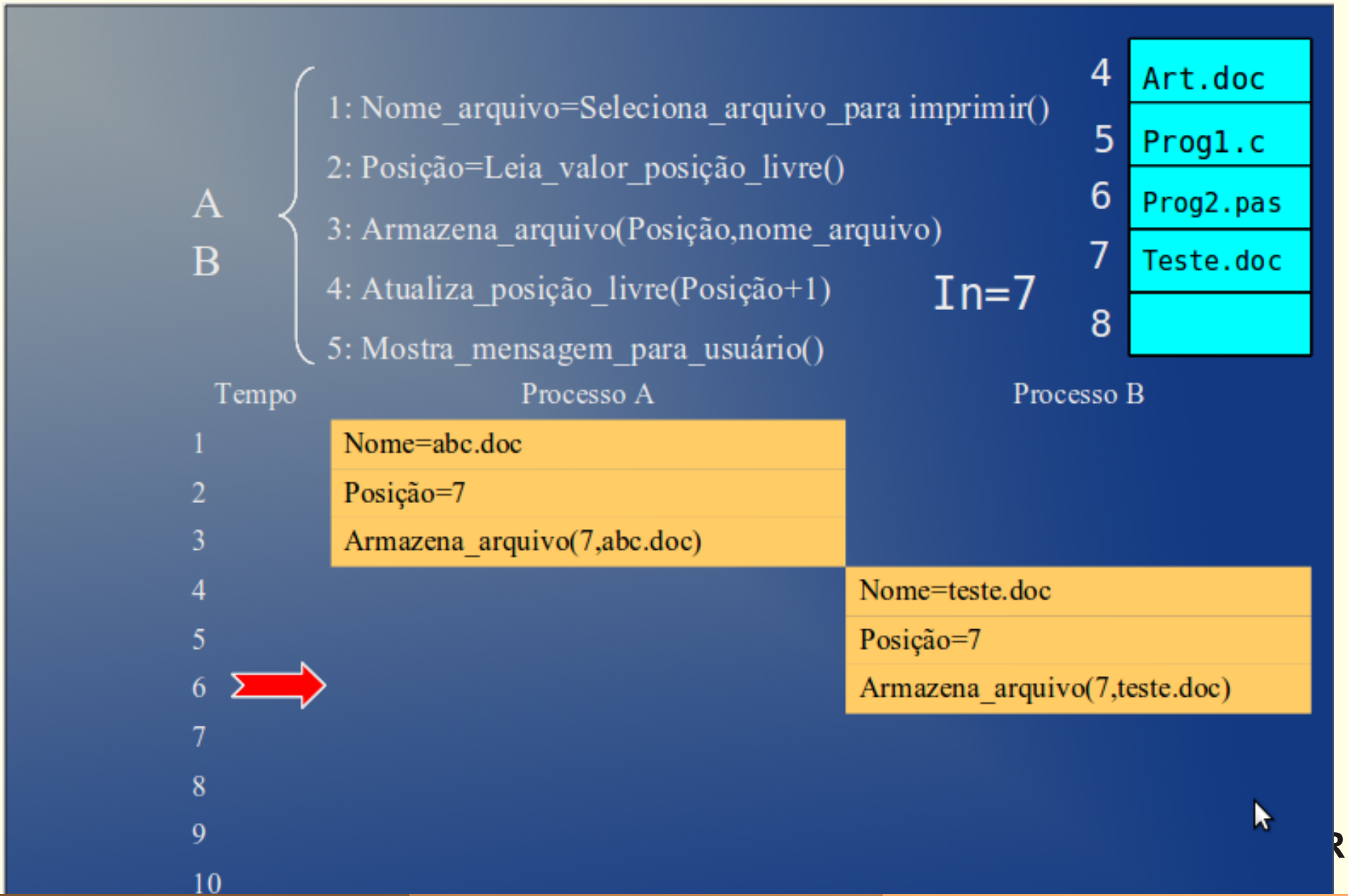
Comunicação entre processos



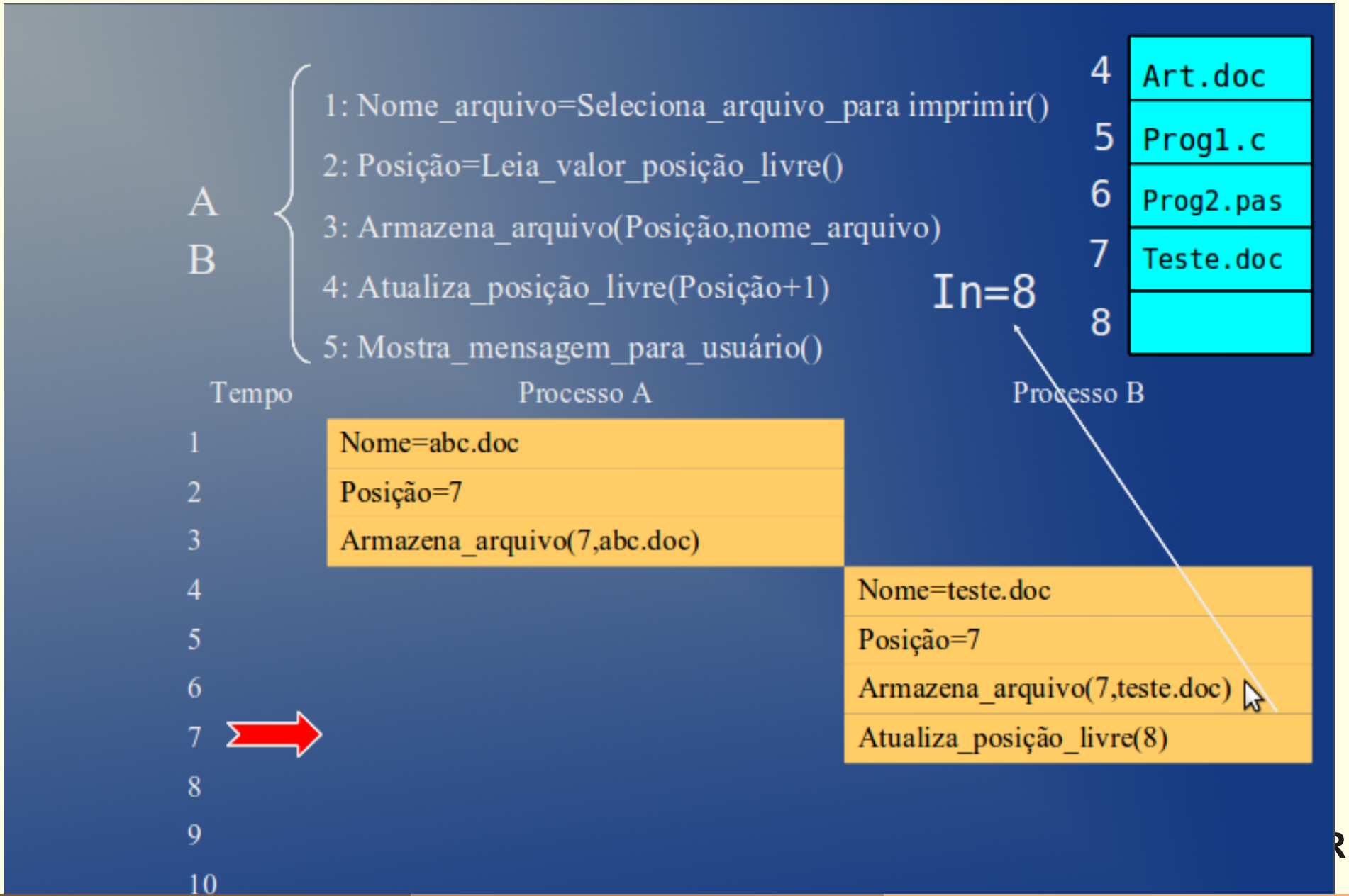
Comunicação entre processos



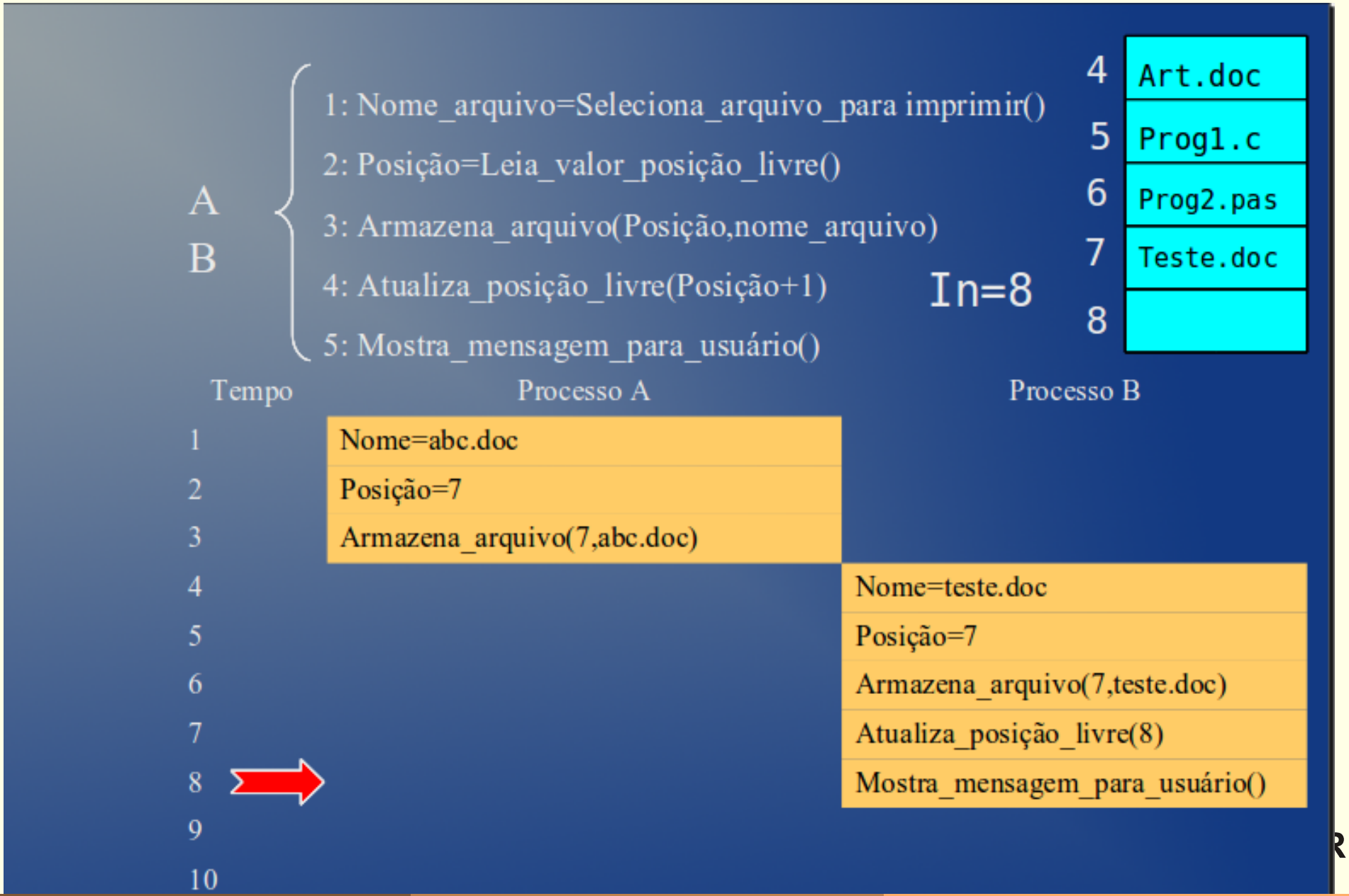
Comunicação entre processos



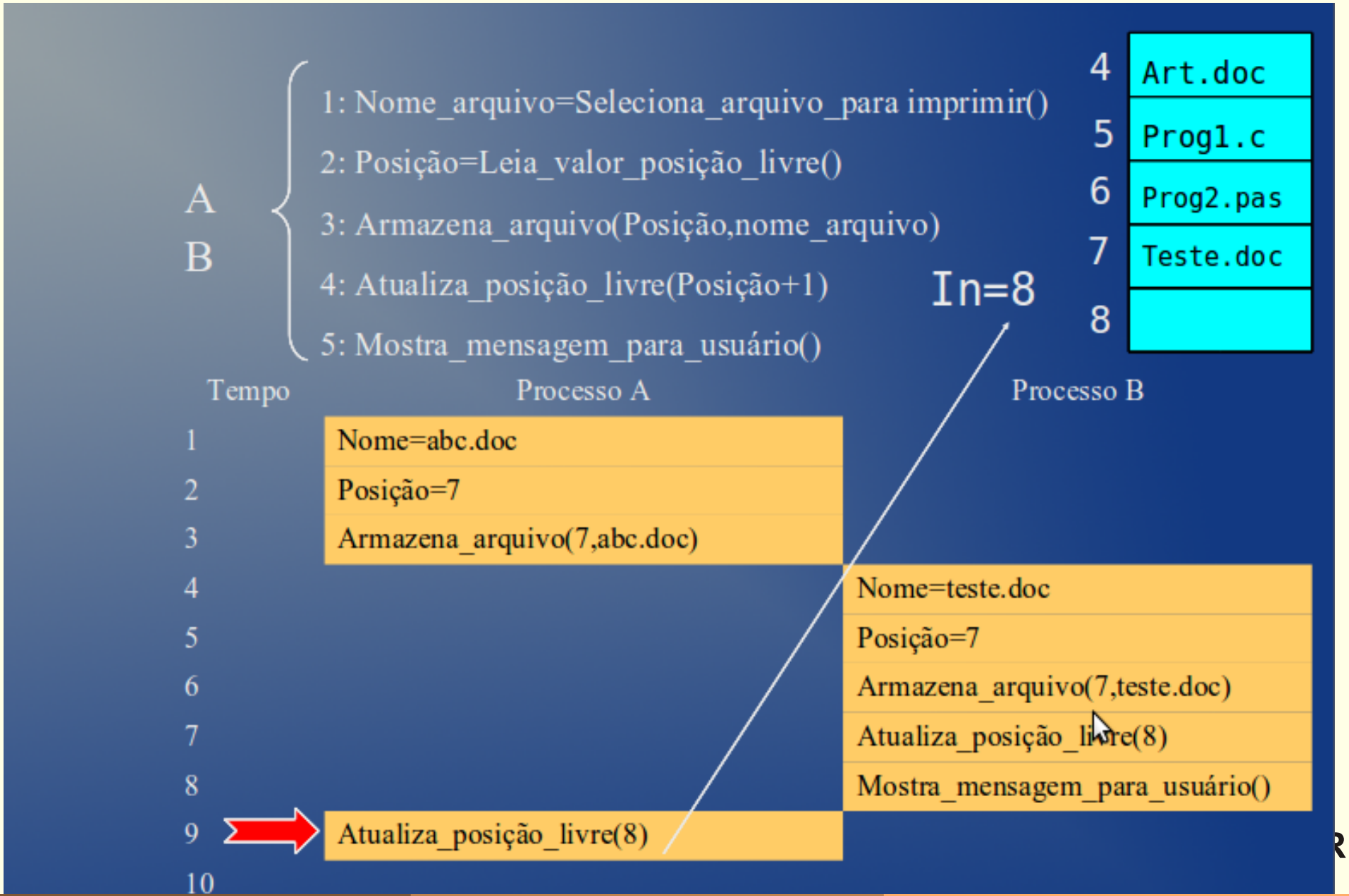
Comunicação entre processos



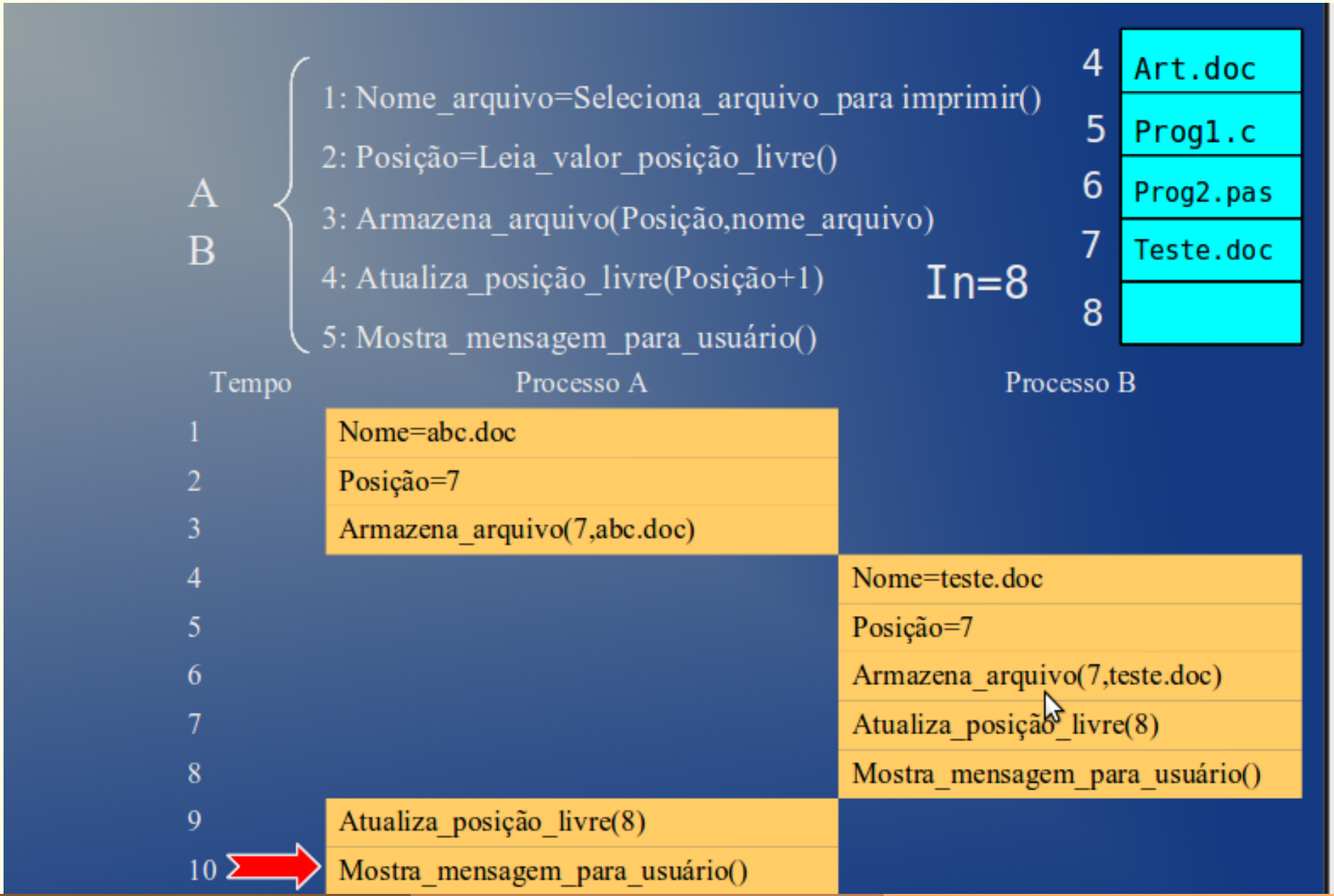
Comunicação entre processos



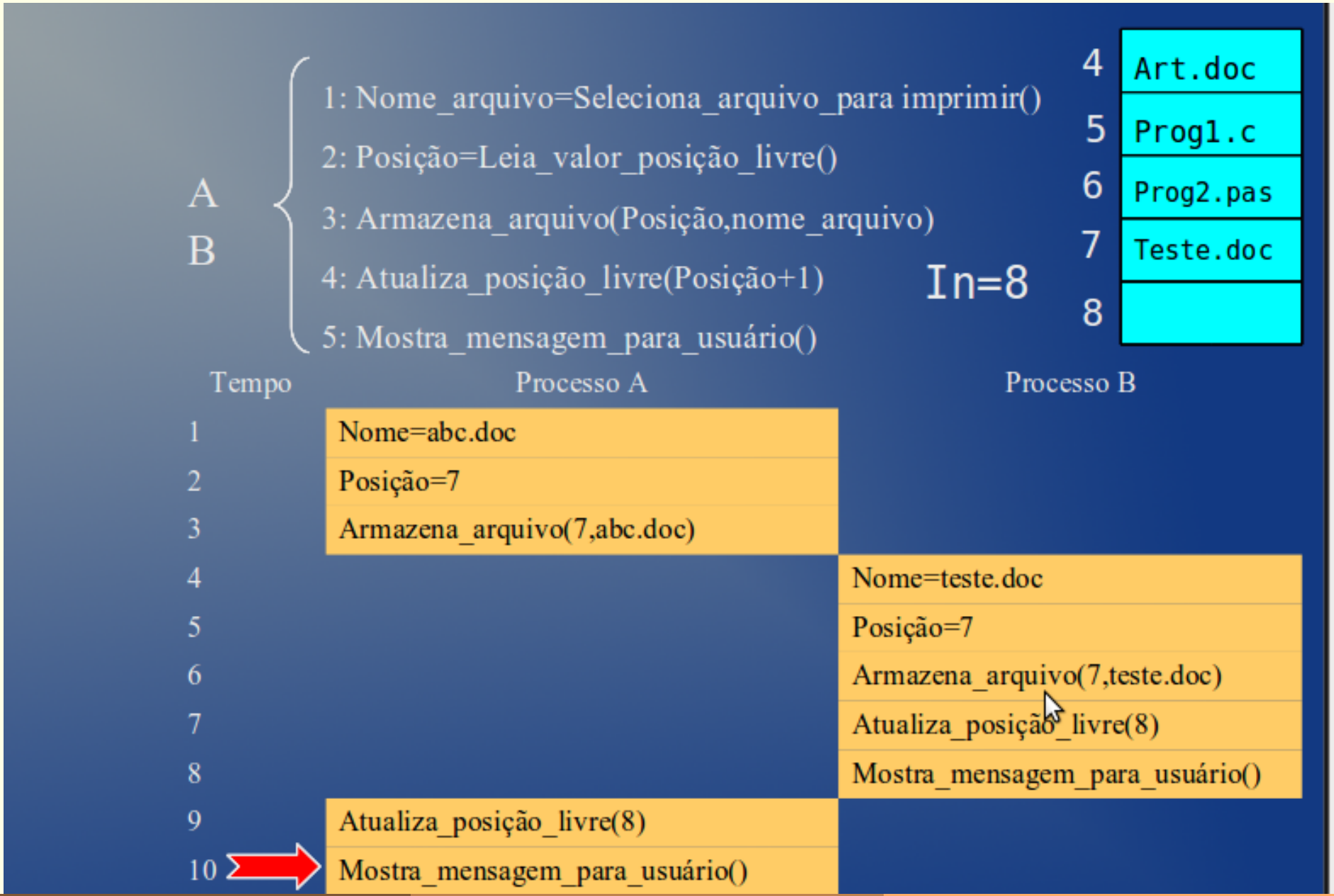
Comunicação entre processos



Comunicação entre processos



Comunicação entre processos



Comunicação entre processos

- No exemplo anterior dois processos (A e B) acessaram a variável **In**;
- A variável **In** é chamada variável compartilhada;
- Como o acesso foi feito sem nenhum cuidado, a execução do processo A não teve o resultado esperado.

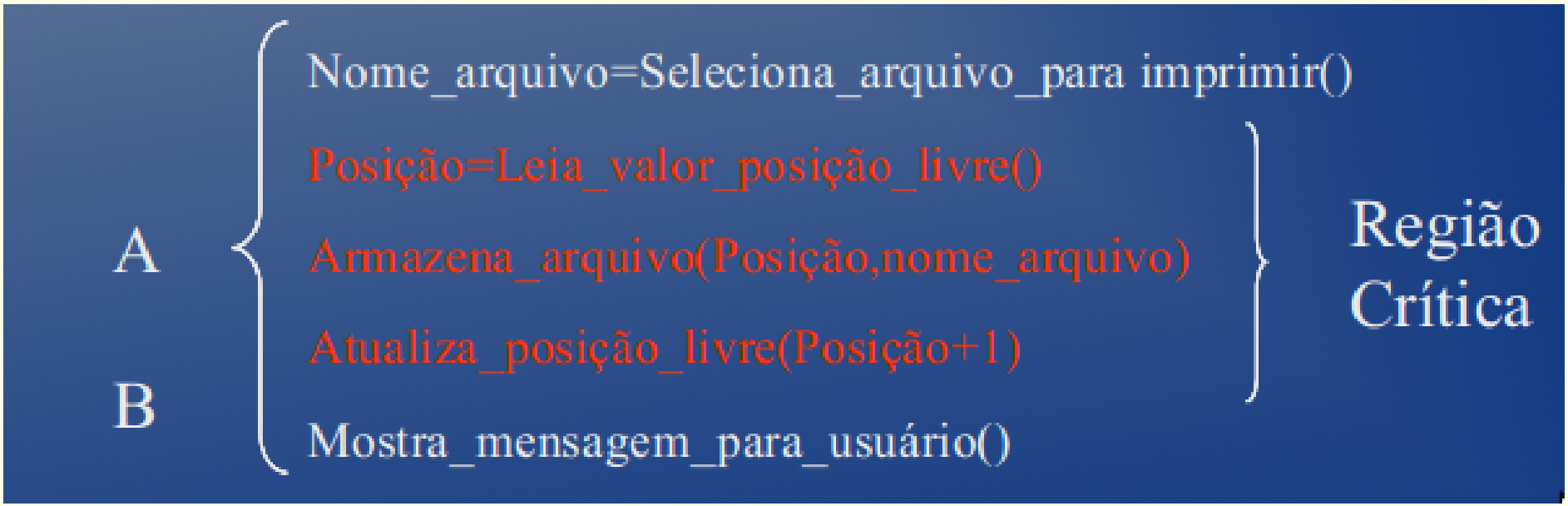
Condição de disputa

Define-se como **Condição de disputa** a situação onde dois ou mais processos estão acessando dados compartilhados (variável **In** no exemplo) e o resultado final do processamento depende de quem roda quando.

Região crítica

Define-se como **Região crítica** a parte do código de um processo cuja execução pode levar a uma condição de disputa. No exemplo seria a parte do código que corresponde as linhas de código 2 e 4.

Comunicação entre processos



Comunicação entre processos

Processos que compartilham dados devem obedecer algumas regras para que sua execução seja um sucesso.

- Dois ou mais processos não podem estar simultaneamente dentro de suas regiões críticas;
- Não podemos assumir nada sobre a velocidade em que cada um dos processos executa suas instruções;

Comunicação entre processos

- Nenhum processo que está executando fora de sua região crítica pode bloquear outro processo;
- Nenhum processo pode ser obrigado a esperar indefinidamente para entrar na sua região crítica.

Comunicação entre processos

Existem algumas soluções para lidar com o problema das regiões críticas:
1-Desabilitar interrupções;
Quais os problemas ?

Comunicação entre processos

```
1 Nome_arquivo=Seleciona_arquivo_para_imprimir();
2 DESABILITA_INTERRUPCOES()
3 Posicao=Leia_valor_posicao_livre()
4 Armazena_arquivo(Posicao,nome_arquivo)
5 Atualiza_posicao_livre(Posicao+1)
6 HABILITA_INTERRUPCOES();
7 Mostra_mensagem_para_usuario()
```

Comunicação entre processos

Existem algumas soluções para lidar com o problema das regiões críticas:
2-Variáveis de travamento;

Comunicação entre processos

```
1  Nome_arquivo=Seleciona_arquivo_para imprimir();
2
3  While (variavel_travamento==1);
4
5      // Processo consegue acesso a regiao critica
6      variavel_travamento=1;
7      Posi   o=Leia_valor_posicao_livre()
8      Armazena_arquivo(Posicao,nome_arquivo)
9      Atualiza_posicao_livre(Posicao+1)
10     variavel_travamento = 0;
11     Mostra_mensagem_para_usuario()
```

Existem algumas soluções para lidar com o problema das regiões críticas:
3-Estrita alternância;

Comunicação entre processos

```
1  //Codigo do processo A
2  while (VEZ==’B’);
3  // Processo A entra regiao
       critica
4
5  .
6  .
7  .
8  .
9  //sai da regiao critica
10 VEZ=’B’;
```

```
1  //Codigo do processo B
2  while (VEZ==’A’);
3  // Processo B entra regiao
       critica
4
5  .
6  .
7  .
8  .
9  //sai da regiao critica
10 VEZ=’A’;
```

Comunicação entre processos

Existem algumas soluções para lidar com o problema das regiões críticas:
4-Instrução TSL;

TSL (Test and Set Locked)

A instrução TSL é uma instrução atômica, ou seja ela é composta internamente por algumas ações mas estas ações são executadas completamente como se fossem apenas uma. Ou seja, não existe como o escalonador do SO executar apenas parte da instrução e dar o processador para outro processo executar.

$$\text{TSL Registrador, valor} = \begin{cases} \textit{Registrador} \leftarrow \textit{valor} \\ \textit{valor} \leftarrow 1 \end{cases}$$

Comunicação entre processos

```
    //valor esta
        compartilhada
    // entre todos os
        processos
    MOV valor,#0
Entra_Regiao:
    TSL Registrador,valor
    CMP Registrador,#0
    SALTA_NZERO Entra_Regiao
    RET
Sai_Regiao_Critica:
    MOV flag, #0
    RET
```

```
1 //Codigo do processo A
    CALL Entra_Regiao
3   // Executa o codigo
    // dentro da regiao
        critica
5   CALL Sai_Regiao_Critica
```

```
1 //Codigo do processo B
    CALL Entra_Regiao
3   // Executa o codigo
    // dentro da regiao
        critica
5   CALL Sai_Regiao_Critica
```

5-Solução de Peterson;

Comunicação entre processos

```
//flag[2] é booleana; e turn é um inteiro
flag[0]   = 0;
flag[1]   = 0;
turn;
```

```
P0: flag[0] = 1;
    turn = 1;
    while (flag[1] == 1 && turn == 1)
    {
        // ocupado, espere
    }
    // parte crítica
    ...
    // fim da parte crítica
    flag[0] = 0;
```

```
P1: flag[1] = 1;
    turn = 0;
    while (flag[0] == 1 && turn == 0)
    {
        // ocupado, espere
    }
    // parte crítica
    ...
    // fim da parte crítica
    flag[1] = 0;
```