

Sistemas Operacionais

Gerência de Memória

Visão Geral

- Sistemas de memória
 - Mapa de memória
 - Hierarquia de memória
- Resolução de endereços
 - Endereços físicos
 - Endereços virtuais
- Estratégias de alocação de memória
 - Partição fixa
 - Contígua
 - Segmentada
 - **Paginada**

Visão Geral

- Memória principal
 - RAM
- Uso do HD
 - Memória virtual
 - Gerência de armazenamento de arquivos - será abordada posteriormente

Gerência de Memória

Memória Principal:

- Fundamental nos sistemas computacionais;
- Constitui o *Espaço de Trabalho* do sistema;
- Processos, Threads, Bibliotecas Compartilhadas, Núcleo do SO.

Hardware Complexo:

- Um ou mais níveis de Cache;
- Unidades de gerenciamento;
- etc.

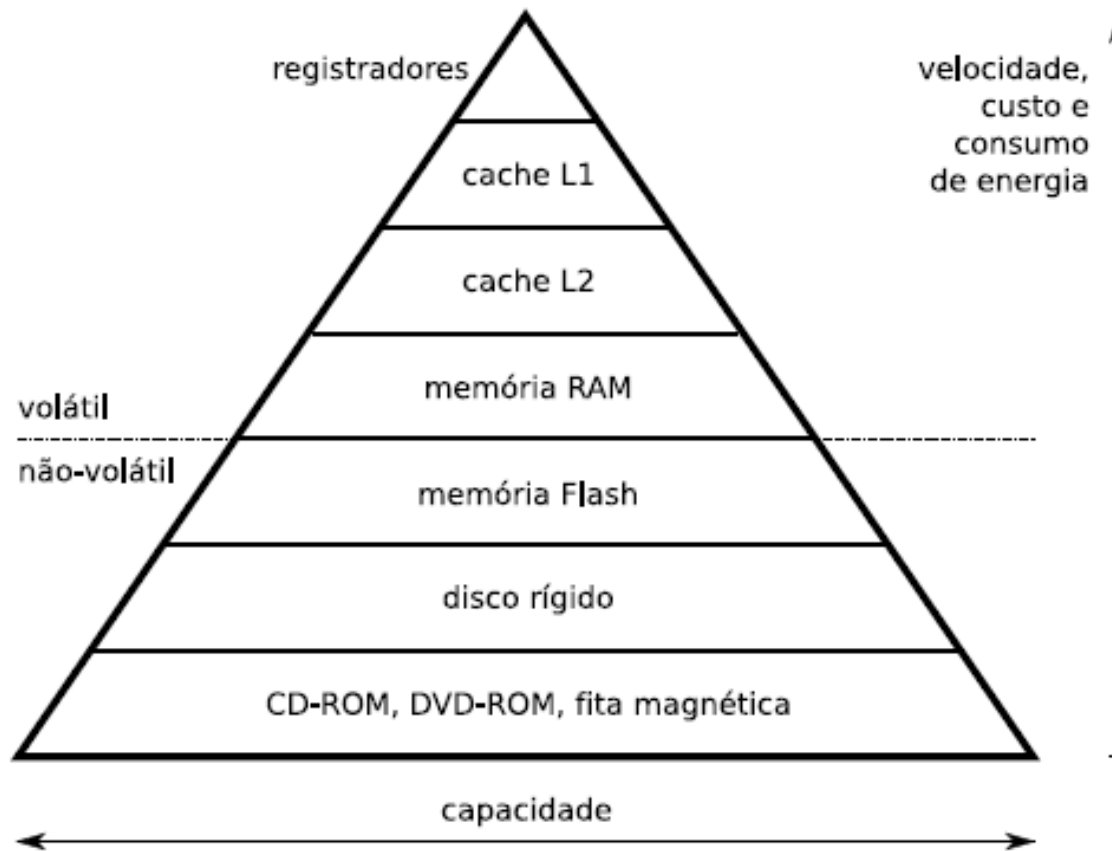
⇒ Esforço de gerência do SO

Gerência de Memória

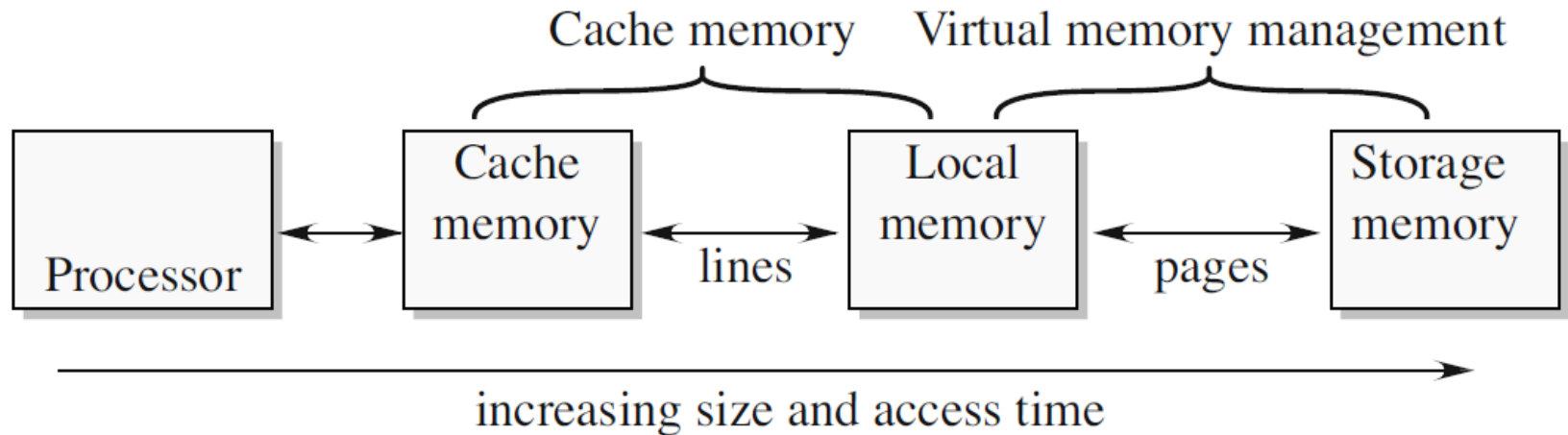


Uma gerência adequada da memória é essencial para o bom **desempenho** e a **segurança** de um computador.

Hierarquia de Memória



Hierarquia de Memória



Velocidade

- Tempo de Acesso
- Taxa de Transferência de Dados

Meio	Tempo de acesso	Taxa de transferência
Cache L2	1 ns	1 GB/s (1 ns/byte)
Memória RAM	60 ns	1 GB/s (1 ns/byte)
Memória <i>flash</i> (NAND)	2 ms	10 MB/s (100 ns/byte)
Disco rígido IDE	10 ms (tempo necessário para o deslocamento da cabeça de leitura e rotação do disco até o setor desejado)	80 MB/s (12 ns/byte)
DVD-ROM	de 100 ms a vários minutos (caso a gaveta do leitor esteja aberta ou o disco não esteja no leitor)	10 MB/s (100 ns/byte)



Definindo Endereços

Endereços, variáveis e funções

Ao escrever um programa usando uma linguagem de alto nível:

- Há apenas referências a entidades abstratas, como **variáveis**, **funções**, **parâmetros** e **valores de retorno**;
- Não há necessidade do programador definir ou manipular **endereços de memória** explicitamente.

Endereços, variáveis e funções

```
#include <stdlib.h>
```

```
#include <stdio.h>
```

```
int main ()
```

```
{
```

```
    int i, soma = 0 ;
```

```
    for (i=0; i< 5; i++)
```

```
    {
```

```
        soma += i ;
```

```
        printf ("i vale %d e soma vale  
                %d\n", i, soma) ;
```

```
    }
```

```
    exit(0) ;
```

```
}
```

Endereços, variáveis e funções

Entretanto, o **processador** acessa endereços de memória para:

- **Buscar** as instruções e seus operandos;
- **Escrever** os resultados do processamento das instruções.

Endereços, variáveis e funções

Portanto, em algum dos seguintes momentos, quando o programa for

- **compilado**;
- **ligado** a bibliotecas;
- **carregado** na memória;
- ou **executado** pelo processador;

cada variável ou trecho de código definido pelo programador deverá ocupar um **espaço específico e exclusivo** na memória.

O código...

```
#include <stdlib.h>
```

```
#include <stdio.h>
```

```
int main ()
```

```
{
```

```
    int i, soma = 0 ;
```

```
    for (i=0; i< 5; i++)
```

```
    {
```

```
        soma += i ;
```

```
        printf ("i vale %d e soma vale  
                %d\n", i, soma) ;
```

```
    }
```

```
    exit(0) ;
```

```
}
```

...se transforma em

00000000 <main>:

0: 8d 4c 24 04

4: 83 e4 f0

7: ff 71 fc

a: 55

b: 89 e5

d: 51

e: 83 ec 14

11: c7 45 f4 00 00 00 00

18: c7 45 f8 00 00 00 00

1f: eb 1f

21: 8b 45 f8

lea 0x4(%esp),%ecx
and \$0xffffffff0,%esp

pushl -0x4(%ecx)

push %ebp

mov %esp,%ebp

push %ecx

sub \$0x14,%esp

movl \$0x0,-0xc(%ebp)

movl \$0x0,-0x8(%ebp)

jmp 40 <main+0x40>

mov -0x8(%ebp),%eax

...se transforma em

00000000 <main>:

0: 8d 4c 24 04

4: 83 e4 f0

7: ff 71 fc

a: 55

b: 89 e5

d: 51

e: 83 ec 14

11: c7 00 00 00

15: f8 00 00 00 00

19: eb 1f

21: 8b 45 f8

lea 0x4(%esp), %ecx

and \$0xffff, %esp

pushl %ecx

movl %ebp, %ecx

mov %esp, %ebp

push %ecx

sub \$0x14, %esp

movl \$0x0, -0xc(%ebp)

movl \$0x0, -0x8(%ebp)

jmp 40 <main+0x40>

mov -0x8(%ebp), %eax

Não há mais referências a nomes simbólicos

Múltiplos processos

```
00000000 <main>:
0: 8d 4c 24 04      lea 0x4(%esp),%ecx
4: 83 e4 f0         and $0xfffff0,%esp
7: ff 71 fc         pushl -0x4(%ecx)
a: 55              push %ebp
b: 89 e5           mov %esp,%ebp
d: 51              push %ecx
e: 83 ec 14         sub $0x14,%esp
11: c7 45 f4 00 00 00 movl $0x0,-0xc(%ebp)
18: c7 45 f8 00 00 00 movl $0x0,-0x8(%ebp)
1f: eb 1f           jmp 40 <main+0x40>
21: 8b 45 f8         mov -0x8(%ebp),%eax
```

```
00000000 <main>:
0: 8d 4c 24 04      lea 0x4(%esp),%ecx
4: 83 e4 f0         and $0xfffff0,%esp
7: ff 71 fc         pushl -0x4(%ecx)
a: 55              push %ebp
b: 89 e5           mov %esp,%ebp
d: 51              push %ecx
e: 83 ec 14         sub $0x14,%esp
11: c7 45 f4 00 00 00 movl $0x0,-0xc(%ebp)
18: c7 45 f8 00 00 00 movl $0x0,-0x8(%ebp)
1f: eb 1f           jmp 40 <main+0x40>
21: 8b 45 f8         mov -0x8(%ebp),%eax
```

```
00000000 <main>:
0: 8d 4c 24 04      lea 0x4(%esp),%ecx
4: 83 e4 f0         and $0xfffff0,%esp
7: ff 71 fc         pushl -0x4(%ecx)
a: 55              push %ebp
b: 89 e5           mov %esp,%ebp
d: 51              push %ecx
e: 83 ec 14         sub $0x14,%esp
11: c7 45 f4 00 00 00 movl $0x0,-0xc(%ebp)
18: c7 45 f8 00 00 00 movl $0x0,-0x8(%ebp)
1f: eb 1f           jmp 40 <main+0x40>
21: 8b 45 f8         mov -0x8(%ebp),%eax
```

?

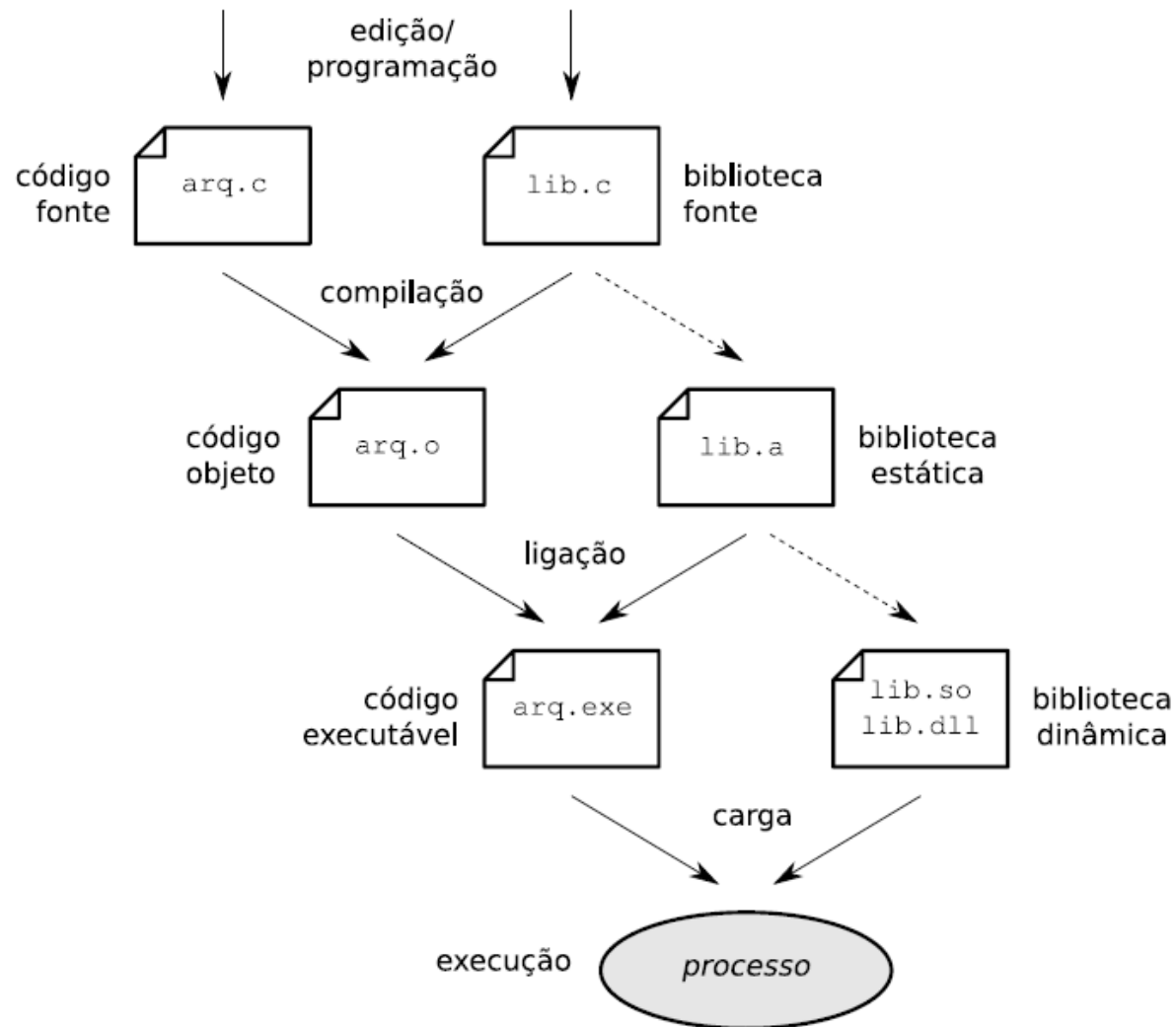
Memória
Principal

Endereços, variáveis e funções

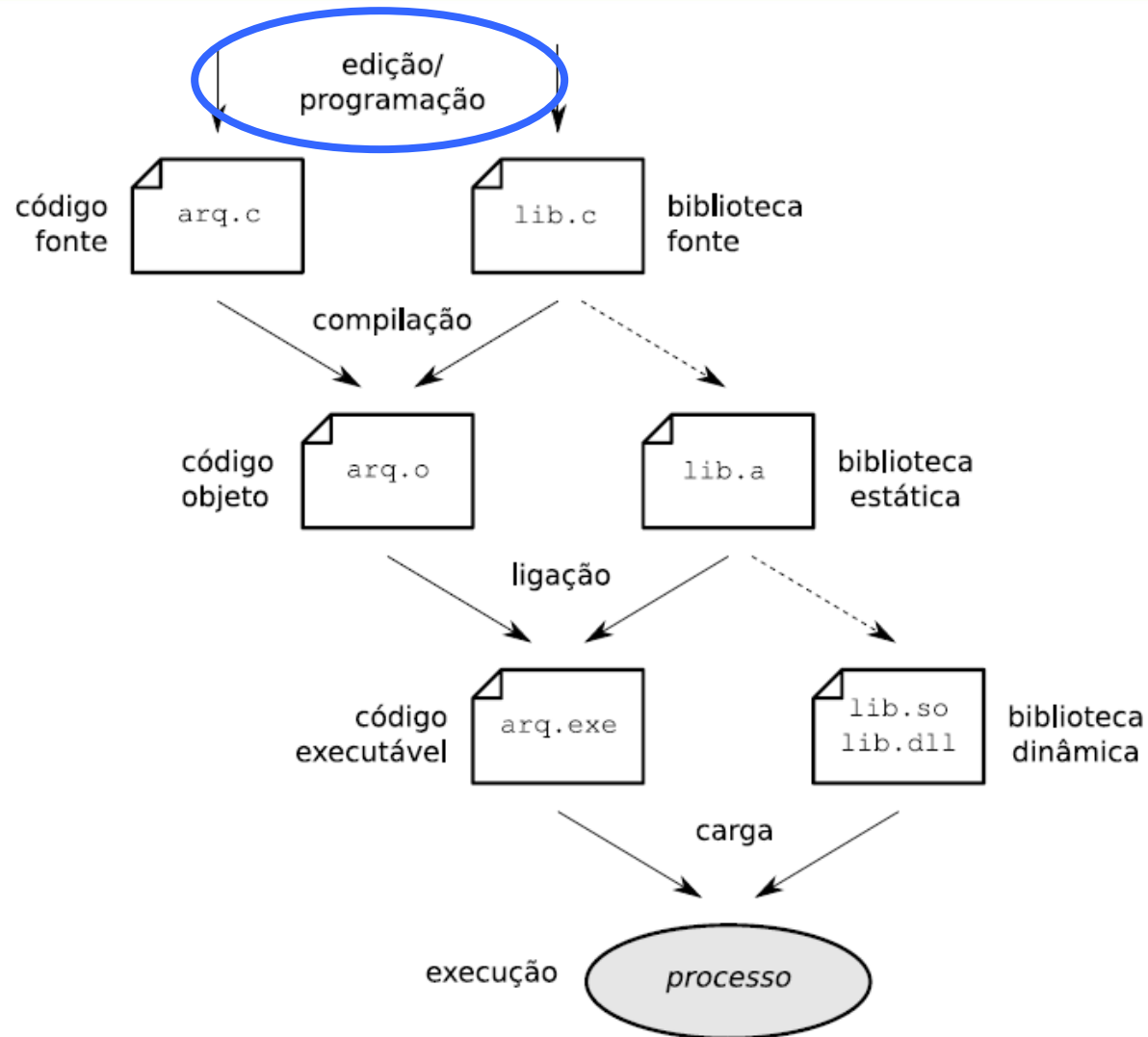
Os **endereços** das **variáveis** e trechos de **código** usados por um programa devem ser definidos em algum momento entre a **escrita do código** e sua **execução** pelo processador.

Resolução dos endereços

Resolução dos endereços



Resolução dos endereços

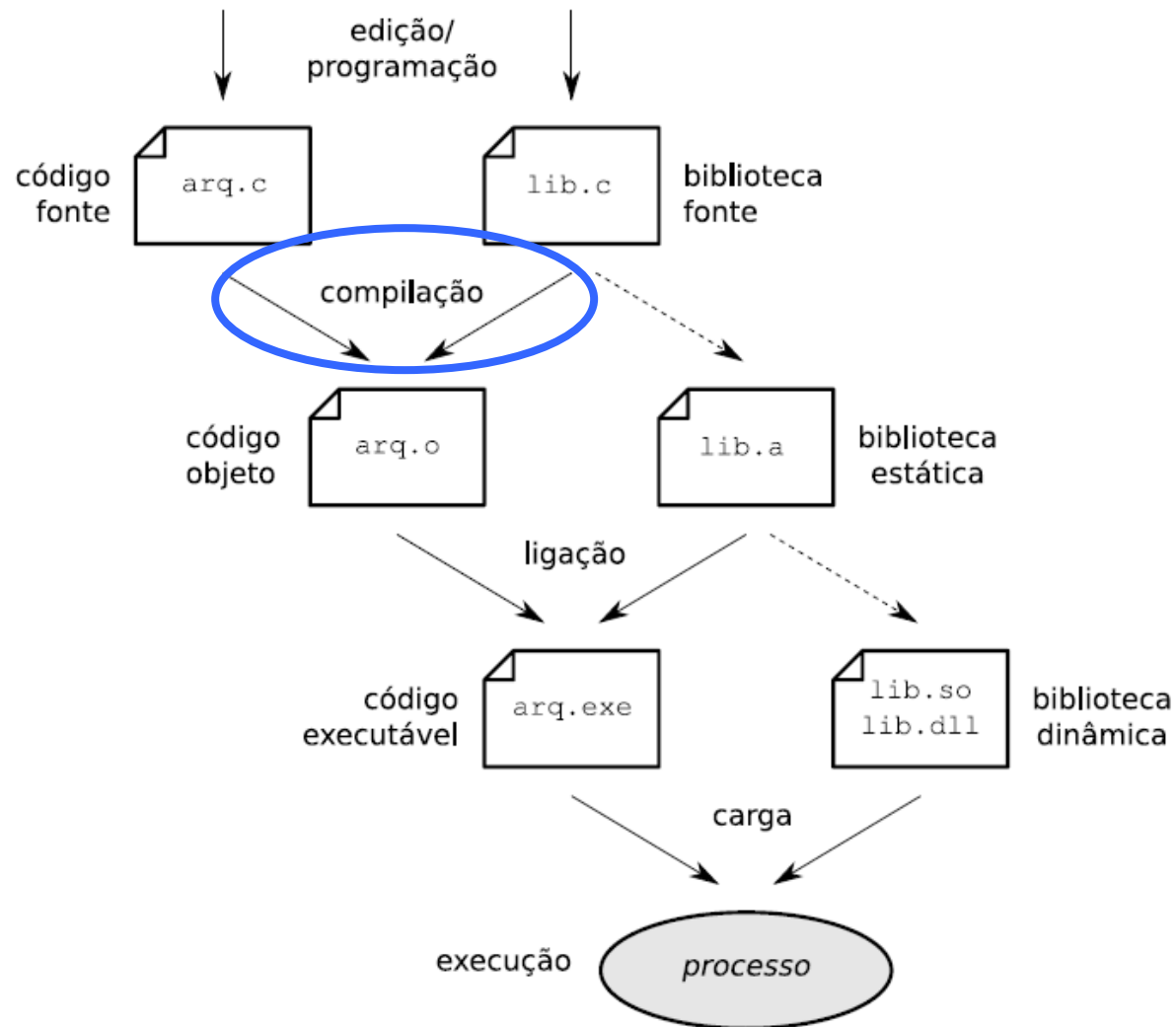


Resolução dos endereços

Durante a edição : O **programador** escolhe a posição de cada uma das **variáveis e do código** do programa na memória.

Abordagem normalmente usada na programação de **sistemas embarcados (muito) simples**, programados diretamente em linguagem de máquina.

Resolução dos endereços

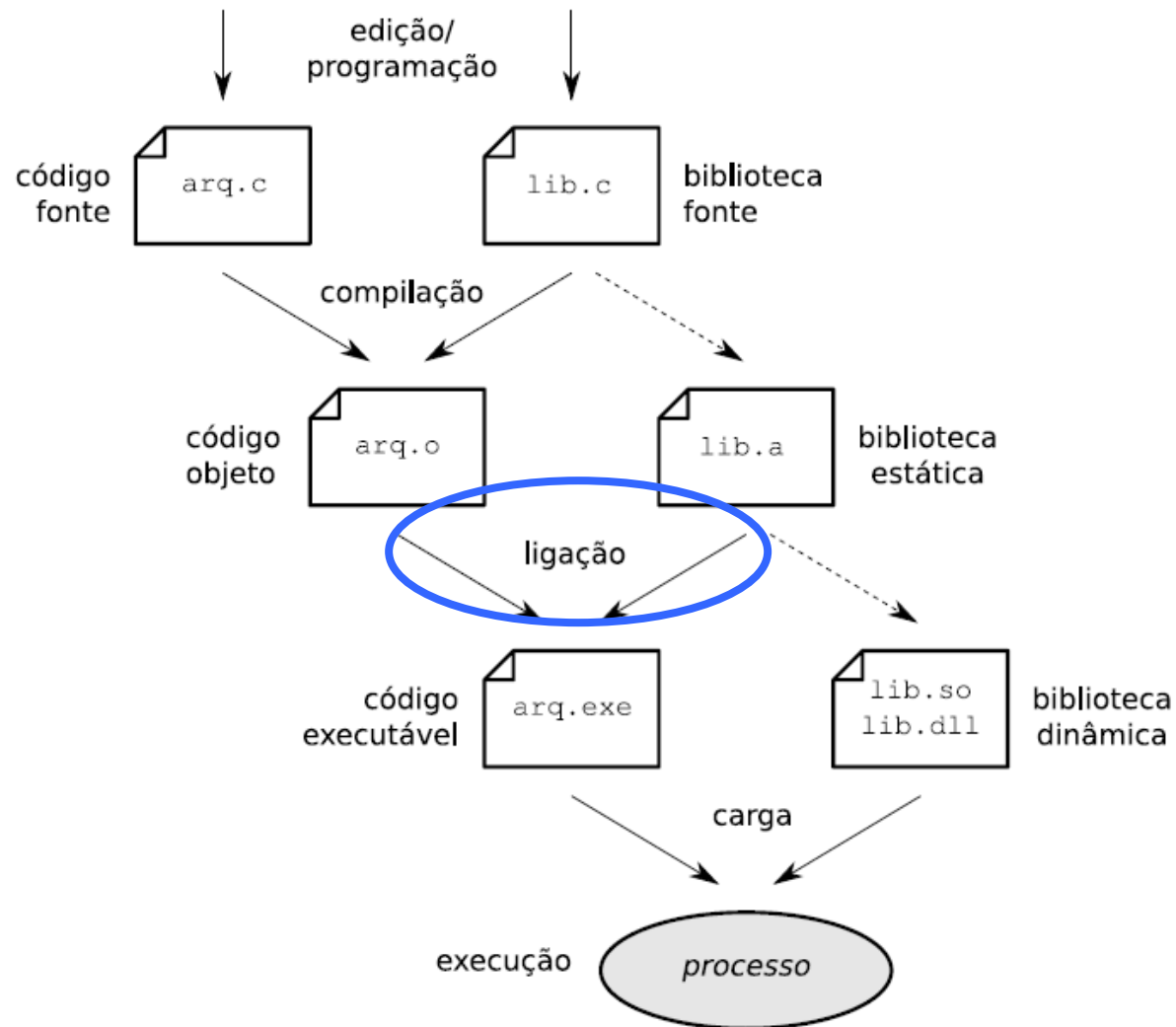


Resolução dos endereços

Durante a compilação : O compilador escolhe as posições das variáveis na memória.

Todos os códigos-fonte que fazem parte do programa **devem** ser conhecidos no momento da compilação, para evitar **conflitos** de endereços entre variáveis.

Resolução dos endereços



Resolução dos endereços

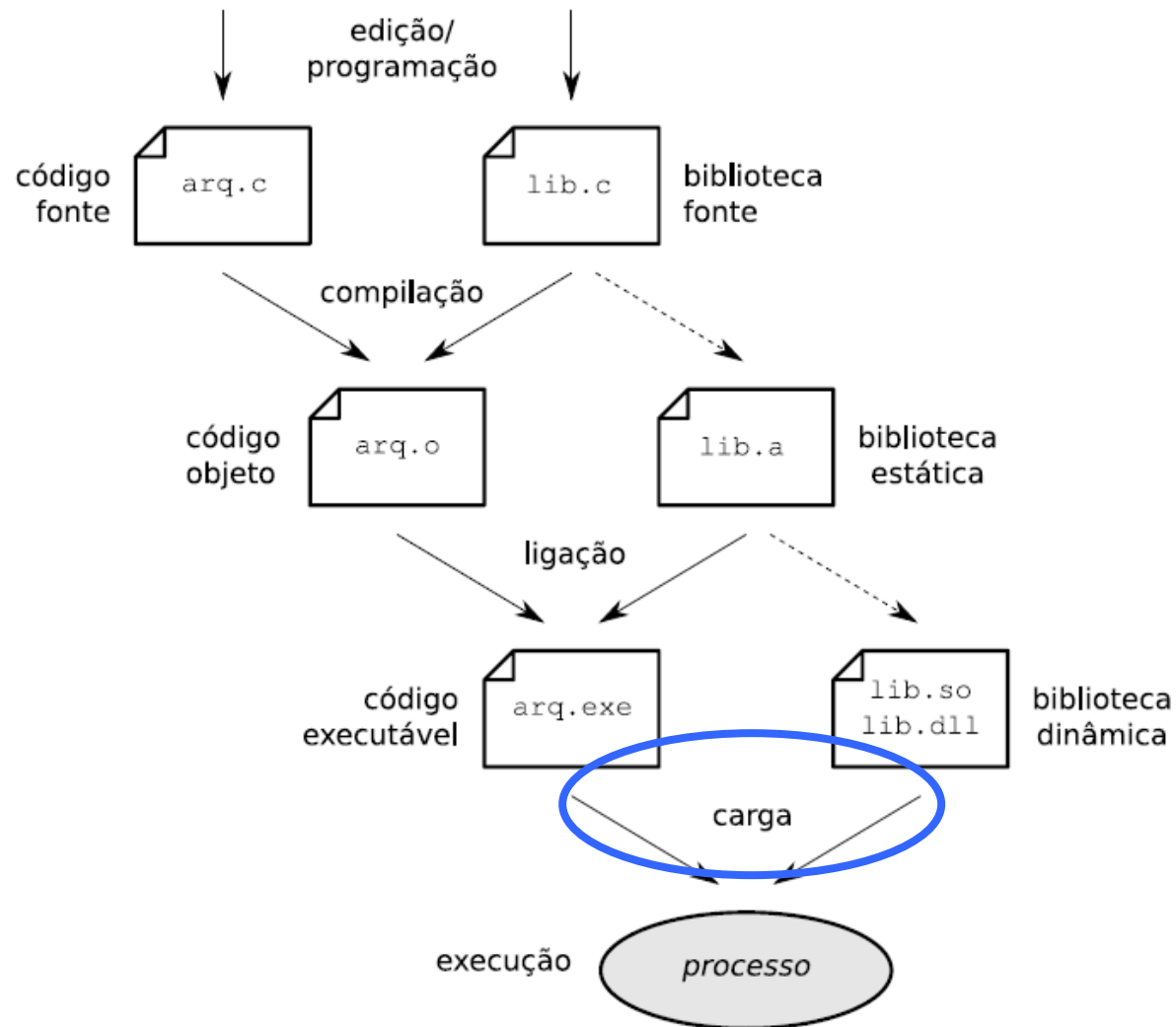
Durante a ligação : O **compilador** gera símbolos que representam as **variáveis** mas não define seus endereços finais.

É gerado um **arquivo-objeto** que contém as instruções em linguagem de máquina e as definições das variáveis utilizadas.

Resolução dos endereços

O *linker* lê todos os arquivos-objeto e as bibliotecas e gera um arquivo executável, no qual os **endereços** de todas as variáveis estão definidos.

Resolução dos endereços



Resolução dos endereços

O *linker* lê todos os arquivos-objeto e as bibliotecas e gera um arquivo executável, no qual os **endereços** de todas as variáveis são relativos.

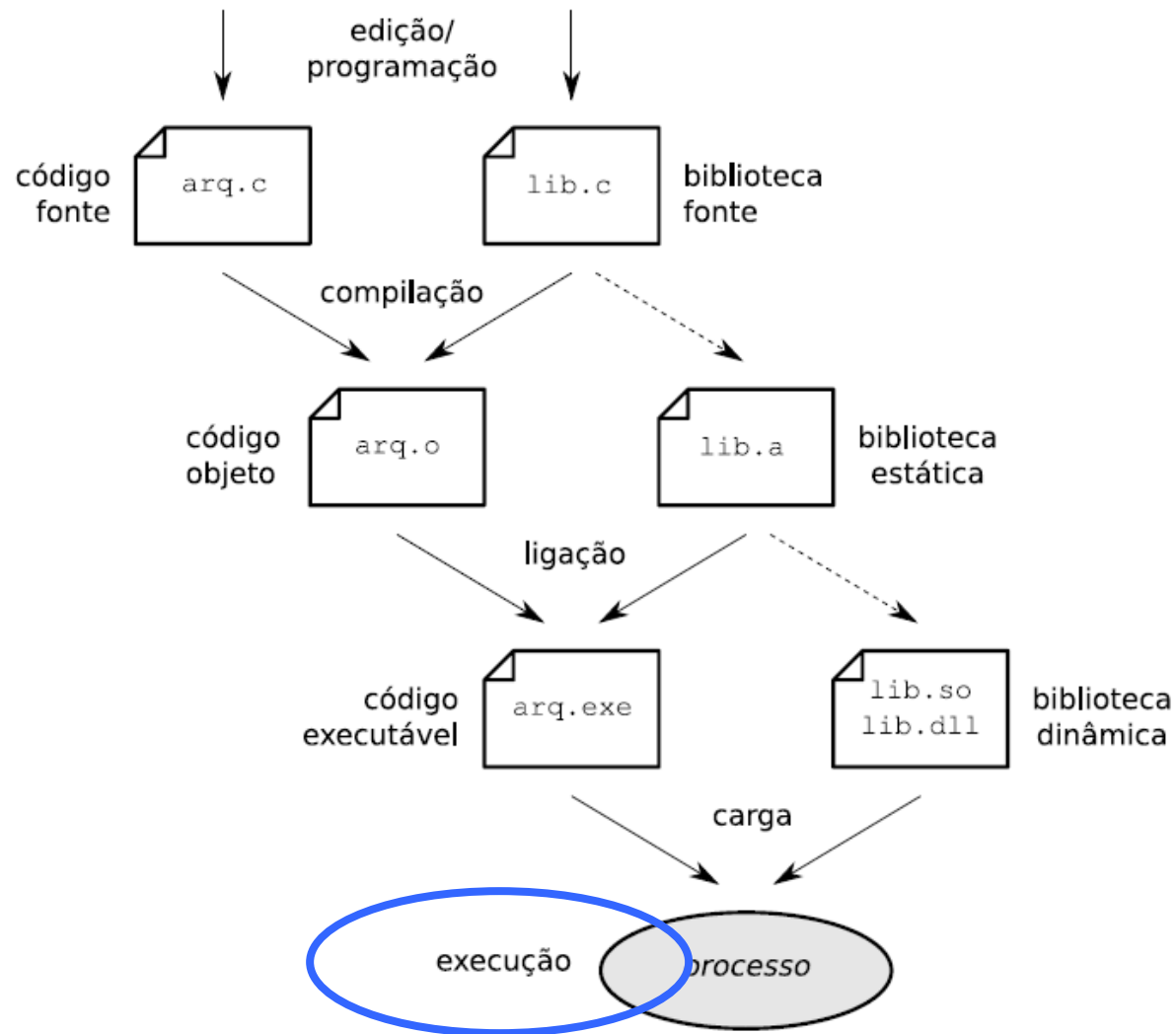
Resolução dos endereços

Durante a carga : Um *carregador* (*loader*) carrega o código do processo na memória e define os endereços de memória.

O *carregador* pode ser parte do *núcleo* do sistema operacional ou uma *biblioteca* ligada ao executável, ou ambos.

Mecanismo normalmente usado na carga das bibliotecas dinâmicas (*DLL* - *Dynamic Linking Libraries*).

Resolução dos endereços




Resolução dos endereços

Durante a execução : Os **endereços** emitidos pelo processador durante a **execução** do processo são analisados e **convertidos** nos endereços efetivos a serem acessados na memória real.

Resolução dos endereços

Durante a execução : Os **endereços** emitidos pelo processador durante a **execução** do processo são analisados e **convertidos** nos endereços efetivos a serem acessados na memória real.

Só é viável com o uso de **hardware** dedicado (MMU) para esse tratamento*.
→ Abordagem **usada na maioria** dos sistemas computacionais atuais**.



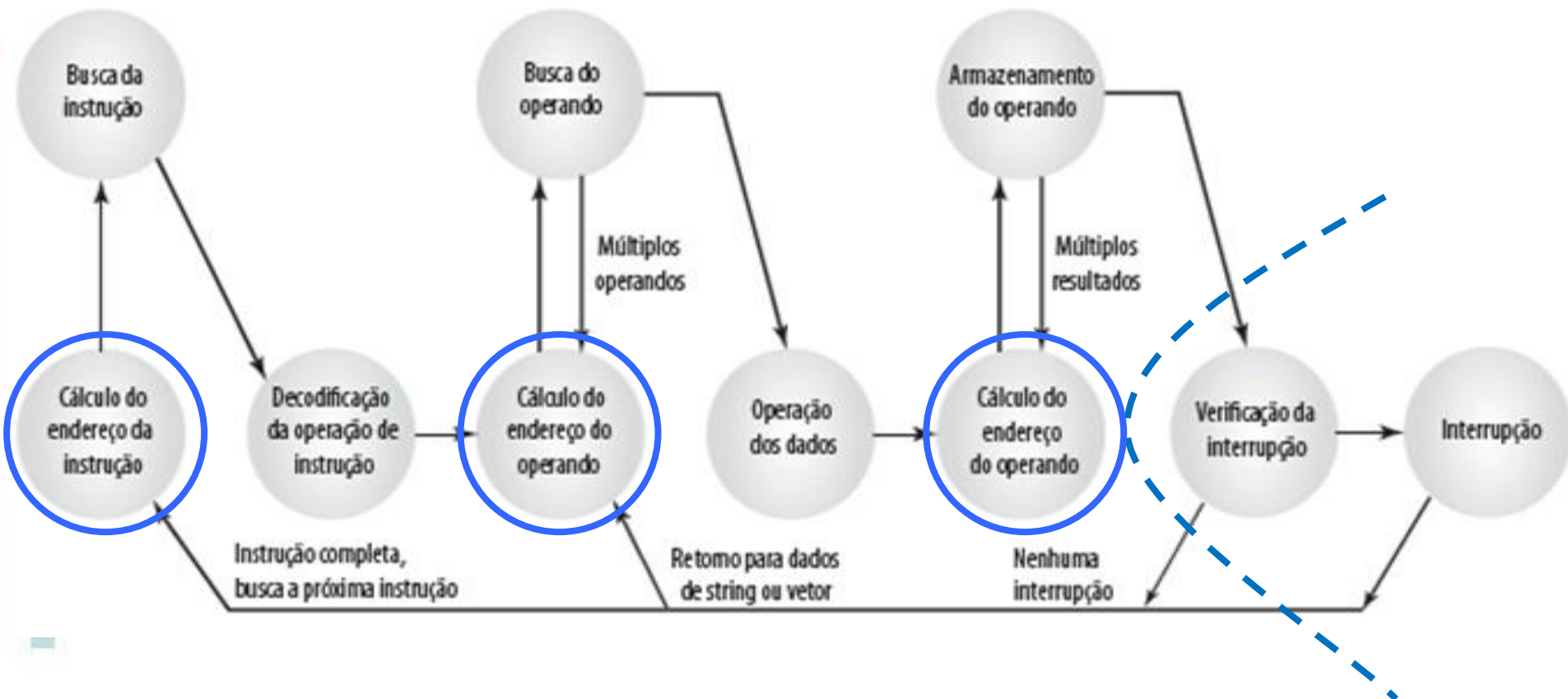
Endereços: lógicos e físicos (MMU)

Endereços lógicos e físicos

Os **endereços de memória** gerados pelo processador são chamados de ***endereços lógicos***

Não são necessariamente iguais aos **endereços reais** das instruções e variáveis na memória real do computador, que são chamados de ***endereços físicos***.

Relembrando: ciclo de instrução



Endereços lógicos e físicos





Alocação

Conversão

Endereços lógicos e físicos

0: 8d 4c 24 04

4: 83 e4 f0

7: ff 71 fc

a: 55

b: 89 e5

d: 51

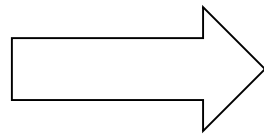
e: 83 ec 14

11: c7 45 f4 00 00 00 00

18: c7 45 f8 00 00 00 00

1f: eb 1f

21: 8b 45 f8



Alocação



Memória
Principal

Endereço lógico

Endereço físico

Endereços lógicos e físicos

0: 8d 4c 24 04

4: 83 e4 f0

7: ff 71 fc

a: 55

b: 89 e5

d: 51

e: 83 ec 14

11: c7 45 f4 00 00 00 00

18: c7 45 f8 00 00 00 00

1f: eb 1f

21: 8b 45 f8

Conversão

Memória
Principal

Endereço lógico

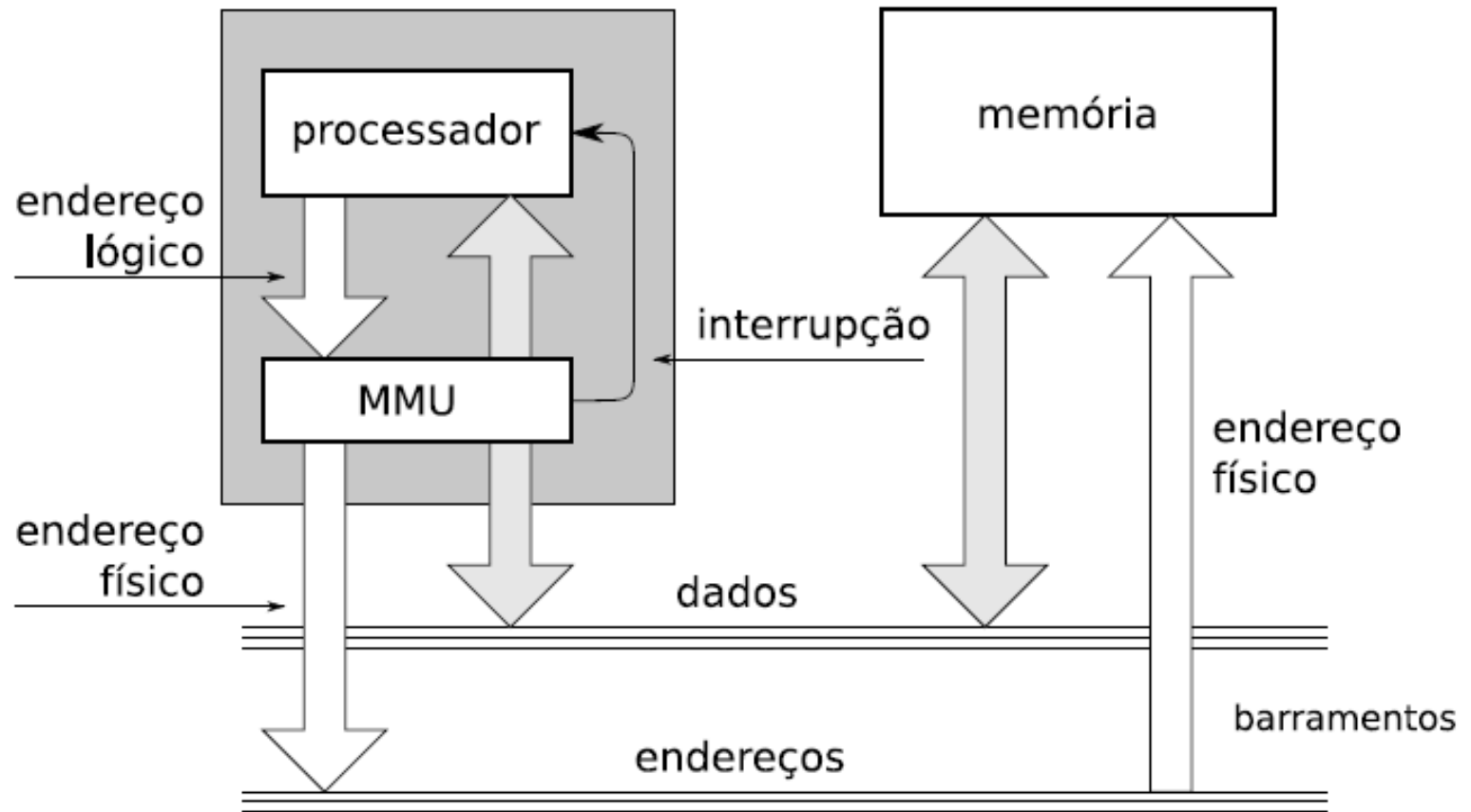
Endereço físico

Endereços lógicos e físicos

Os **endereços lógicos** são interceptados por um hardware especial denominado *Unidade de Gerência de Memória* (MMU-*Memory Management Unit*)

A MMU pode fazer parte do próprio processador ou constituir um dispositivo separado (**máquinas mais antigas**)

MMU

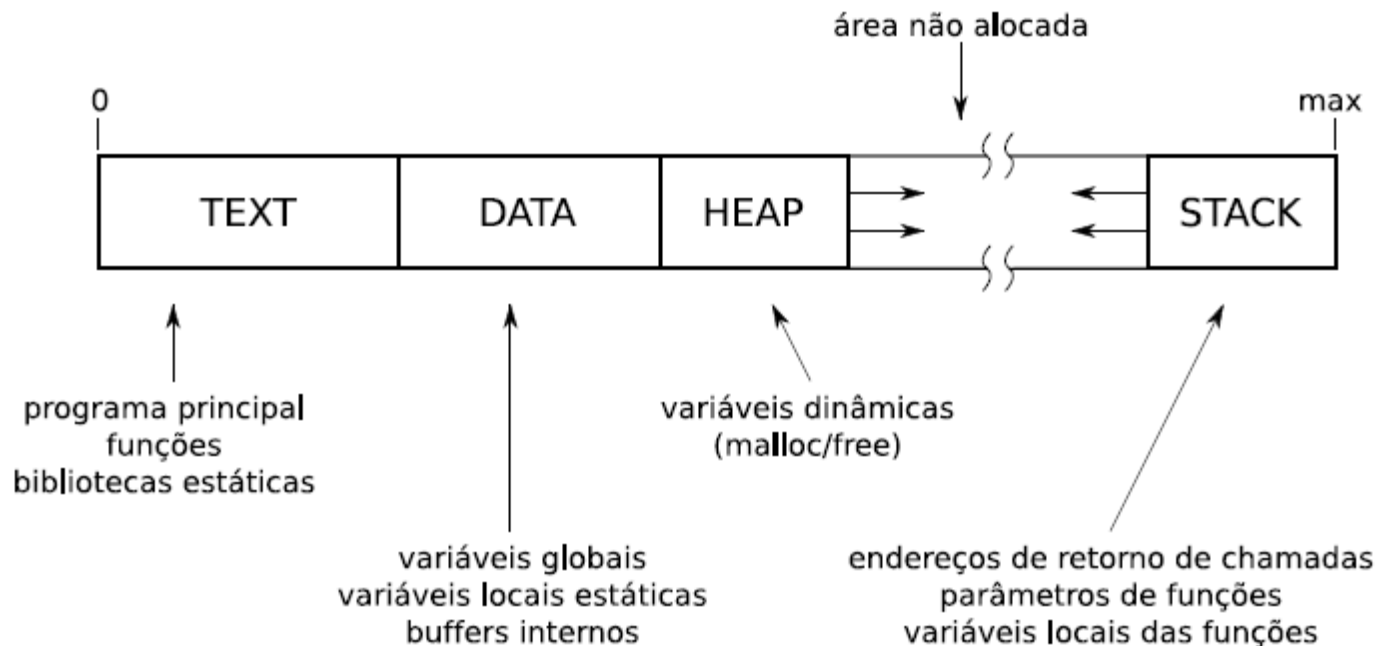




Alocação de Processos

Modelo de memória dos processos

Cada processo é visto pelo **sistema operacional** como uma **área de memória** exclusiva que só ele e o núcleo do sistema podem acessar.



Modelo de memória dos processos

```
#include <stdio.h>
#include <stdlib.h>
int main(int argc, char *argv[]) {
    printf("location of code : %p\n", (void *) main);
    printf("location of heap : %p\n", (void *) malloc(1));
    int x = 3;
    printf("location of stack : %p\n", (void *) &x);
    return x;
}
```

Modelo de memória dos processos

```
#include <stdio.h>
#include <stdlib.h>
int main(int argc, char *argv[]) {
    printf("location of code : %p\n", (void *) main);
    printf("location of heap : %p\n", (void *) malloc(1));
    int x = 3;
    printf("location of stack : %p\n", (void *) &x);
    return x;
}
```

Resultado para um PC Ubuntu 64-bit:

location of code : 0x564fe99086fa

location of heap : 0x564feab3f670

location of stack : 0x7ffec82c6434

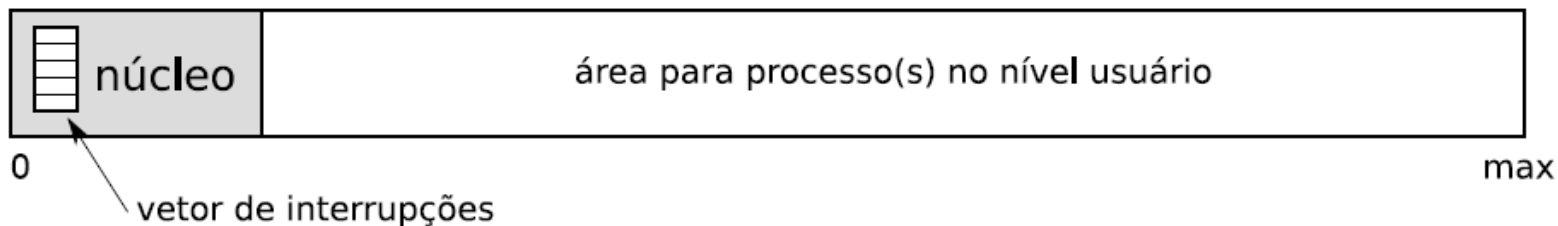
Estratégias de alocação

- Sistemas **Mono-tarefa**
- Partição fixa
- Alocação contígua
- Alocação por segmentos
- Alocação paginada

Estratégias de alocação

Sistemas **Mono-tarefa**:

Problema simples: basta reservar uma área de memória para o núcleo do sistema operacional e alocar o processo na memória restante.



Estratégias de alocação

Sistemas **Multi-tarefa**:

- *Vários processos* podem ser carregados simultaneamente na memória para execução;
- O espaço de memória destinado aos processos deve ser **dividido** entre eles usando uma **estratégia** que permita **eficiência** e **flexibilidade**.

Estratégias de alocação

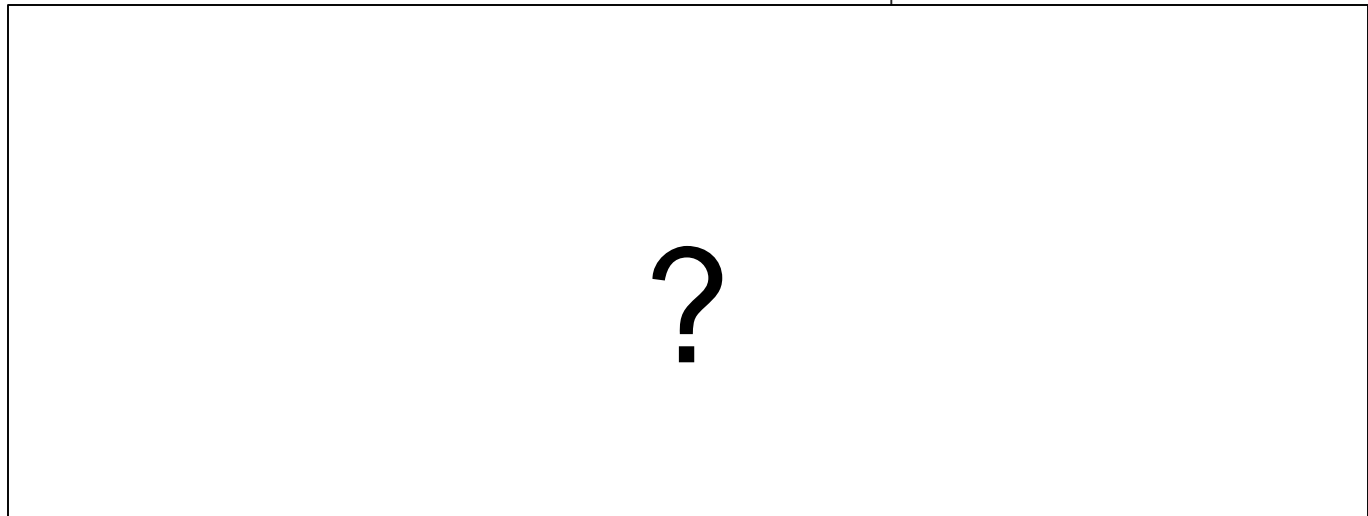
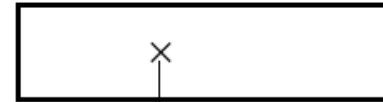
- Sistemas Mono-tarefa
- Partição fixa
- Alocação contígua
- Alocação por segmentos
- Alocação paginada

Partições fixas

- A forma *mais simples* de alocação de memória.
- Consiste em **dividir** a memória destinada aos processos em **N partições fixas**, de tamanhos iguais ou distintos.

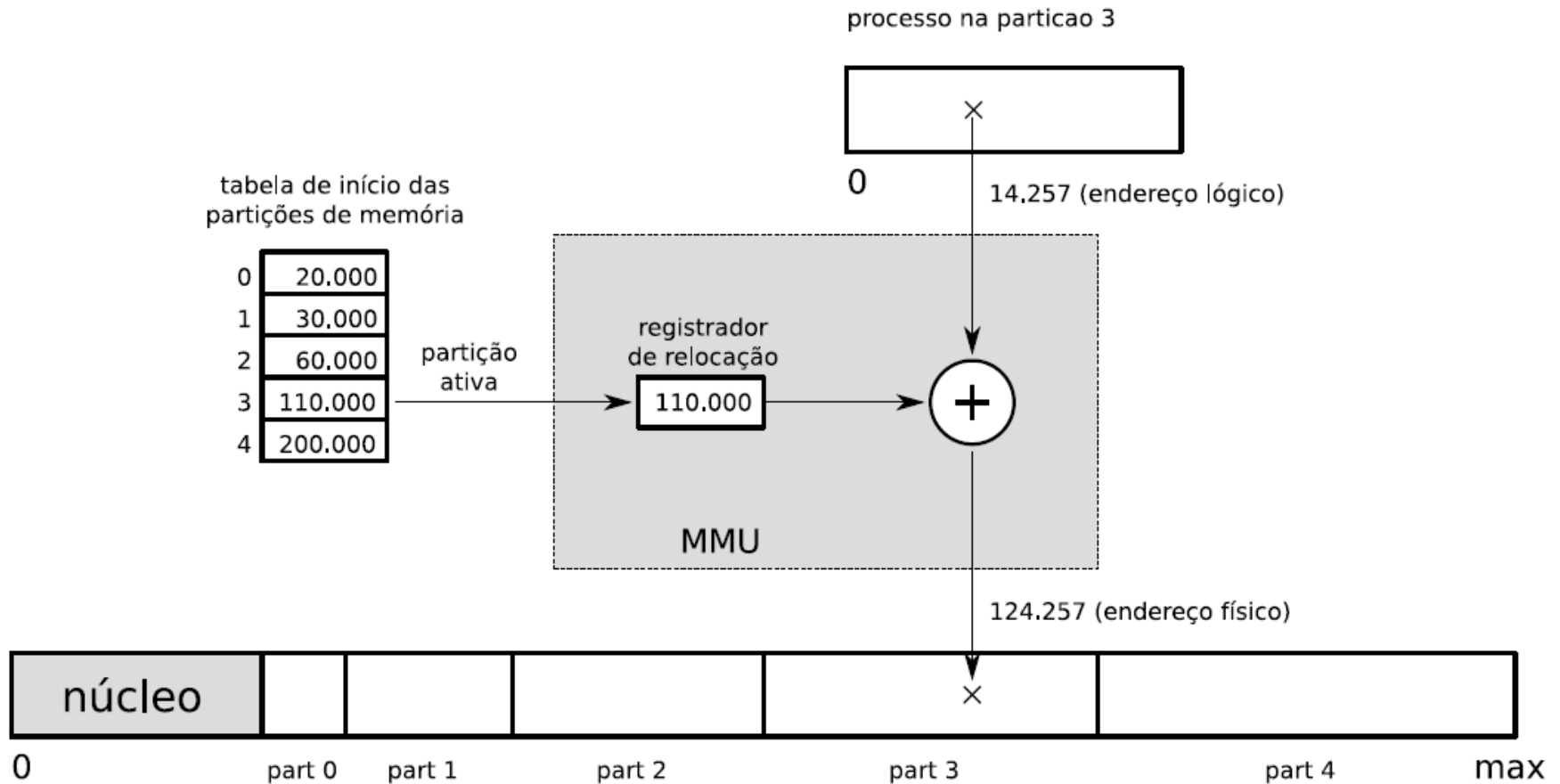
Partições fixas – Endereços

processo na particao 3



0 part 0 part 1 part 2 part 3 part 4 max

Partições fixas – Endereços



Partições fixas - Desvantagens

- Processos podem ter **tamanhos distintos** dos tamanhos das partições \Rightarrow áreas de memória **sem uso** no final de cada partição (fragmentação interna).

Partições fixas - Desvantagens

- Processos podem ter **tamanhos distintos** dos tamanhos das partições \Rightarrow áreas de memória **sem uso** no final de cada partição (fragmentação interna).
- O número máximo de processos na memória é **limitado** ao número de partições, mesmo que os processos sejam pequenos.

Partições fixas - Desvantagens

- Processos maiores que o tamanho da maior partição **não poderão ser carregados** na memória, mesmo se todas as partições estiverem livres.

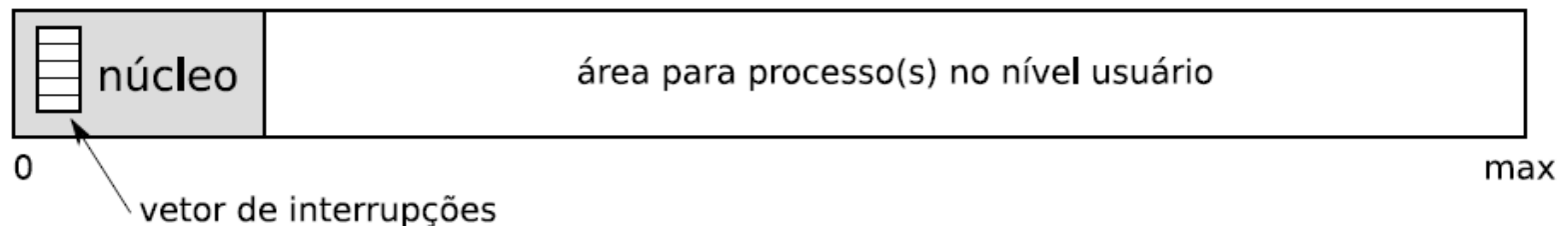
⇒ Muito usada no **OS/360**, sistema operacional da IBM das décadas de 1960-70.

Estratégias de alocação

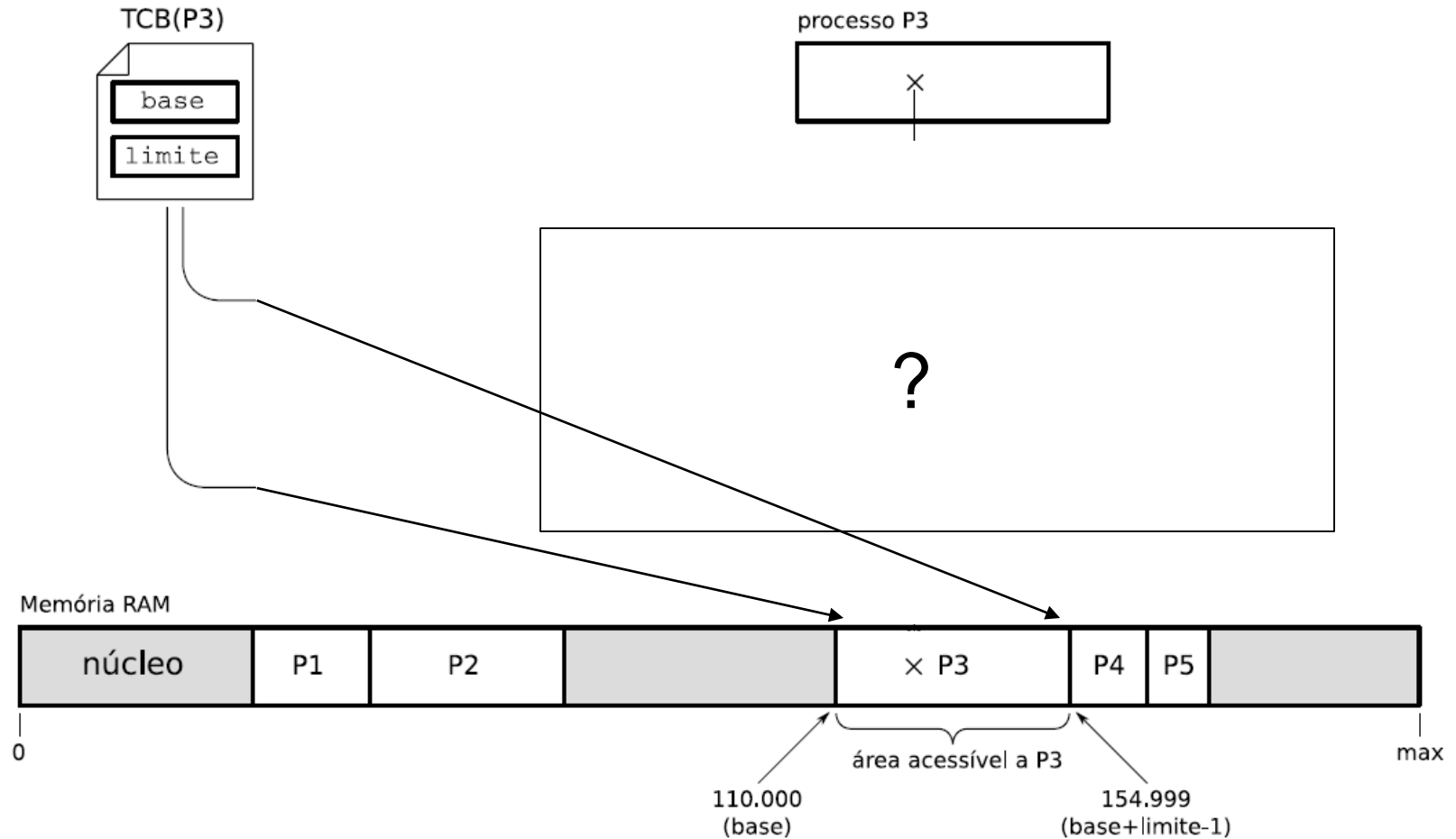
- Sistemas Mono-tarefa
- Partição fixa
- **Alocação contígua**
- Alocação por segmentos
- Alocação paginada

Alocação contígua

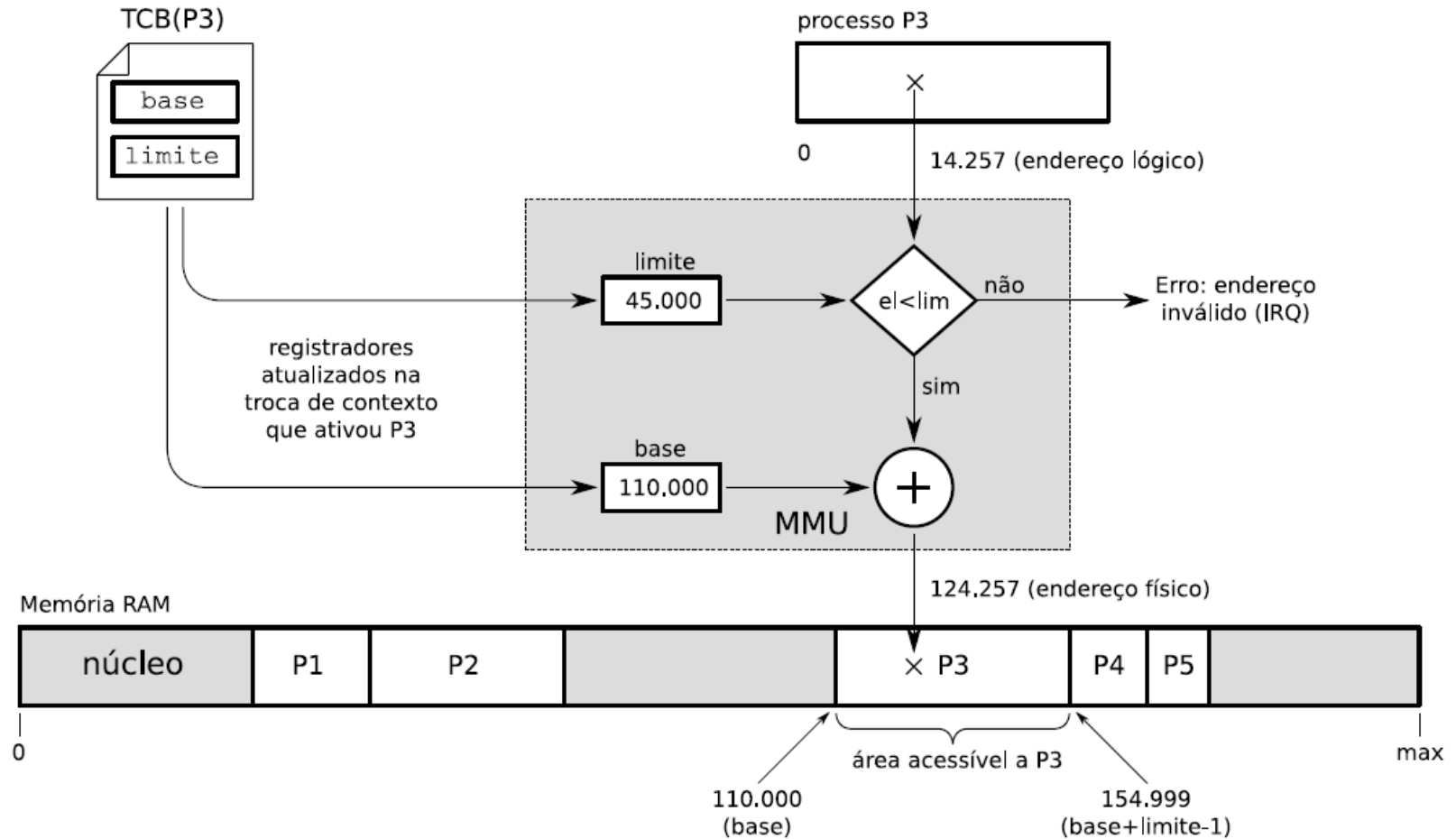
Partições fixas podem se tornar bem mais flexíveis caso o tamanho de cada **partição** possa ser **ajustado** para se adequar à demanda específica de cada processo.



Tradução de Endereços



Tradução de Endereços



Núcleo → Base = 0 e Limite = Max

Alocação contígua – Vantagens e Desvantagens

Simplicidade: depende apenas de dois registradores e de uma lógica simples.

Pode ser implementada em hardware de baixo custo.

Alocação contígua – Vantagens e Desvantagens

Simplicidade: depende apenas de dois registradores e de uma lógica simples.

Pode ser implementada em hardware de baixo custo.

Todavia, é uma estratégia pouco flexível e está muito sujeita à fragmentação externa.

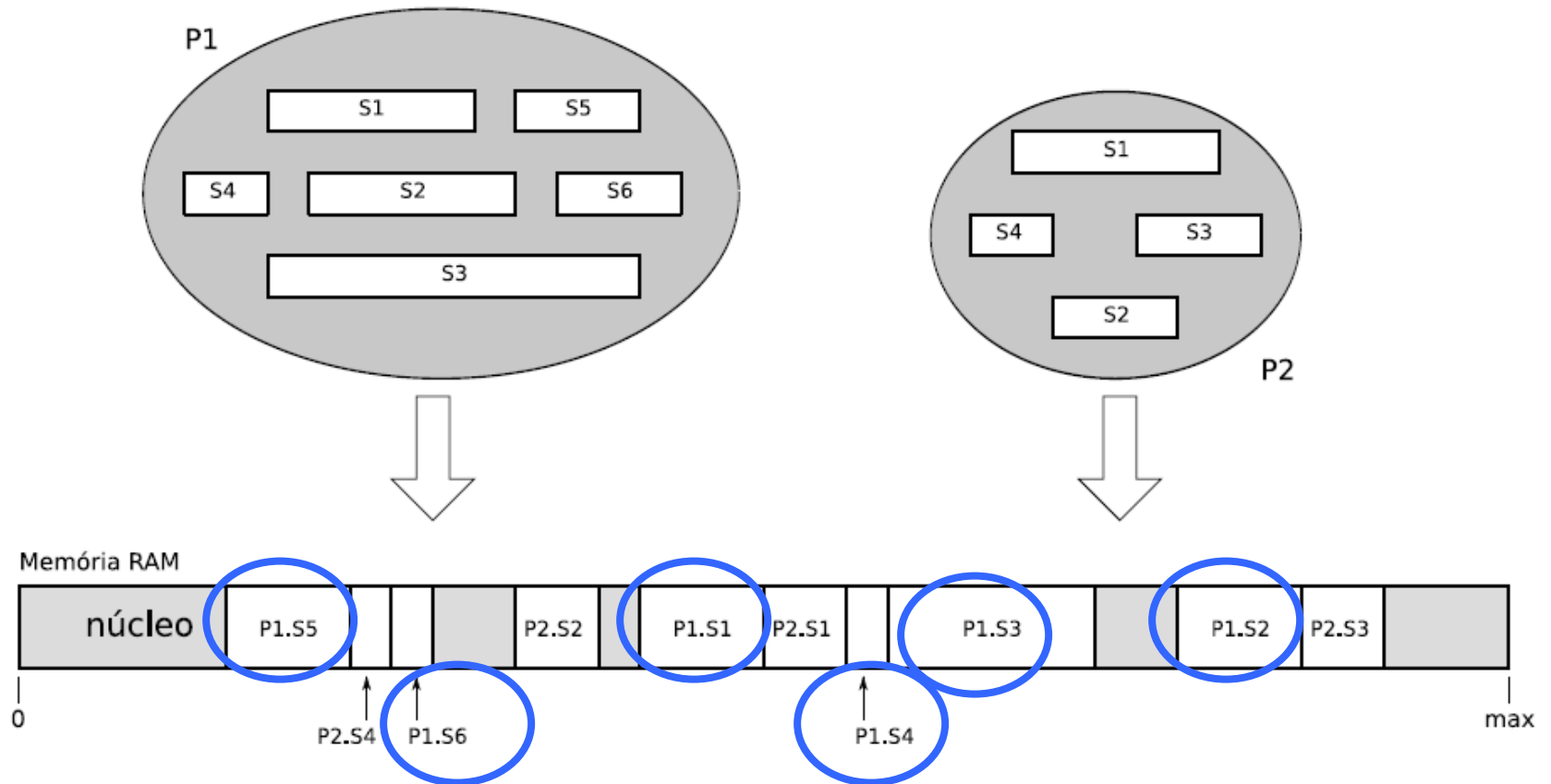
Estratégias de alocação

- Sistemas Mono-tarefa
- Partição fixa
- Alocação contígua
- **Alocação por segmentos**
- Alocação paginada

Alocação por segmentos

- É uma **extensão da alocação contígua**
- O espaço de memória de um **processo** é **fracionado segmentos**
- **Segmentos** podem ser alocados separadamente na memória física.

Alocação por segmentos



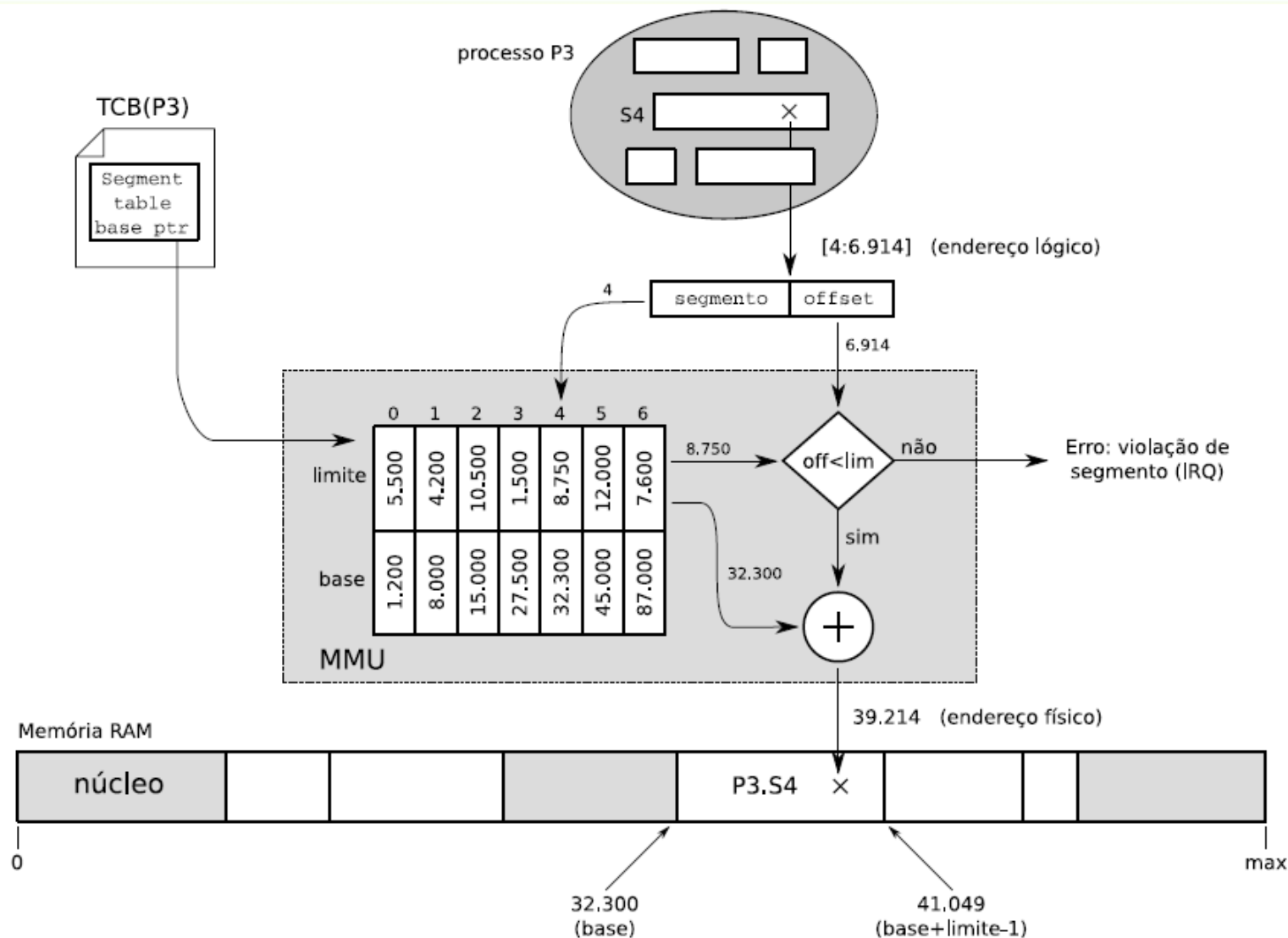
Endereços

Este modelo usa endereços lógicos ***bidimensionais*** - [*segmento:offset*]

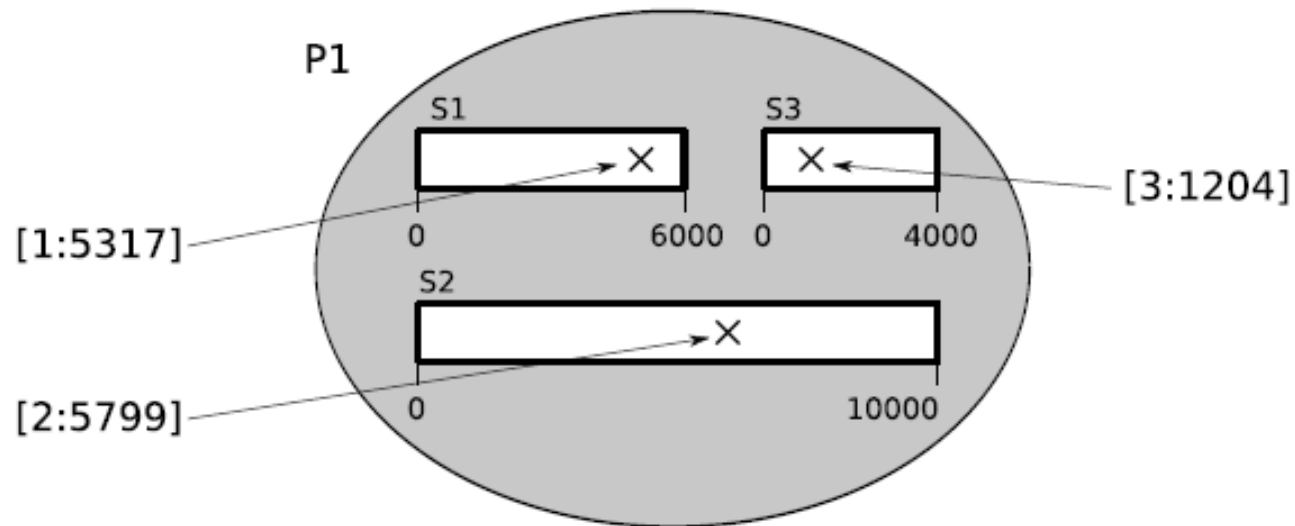
segmento indica o número do *segmento* desejado.

offset indica a posição desejada dentro do *segmento*.

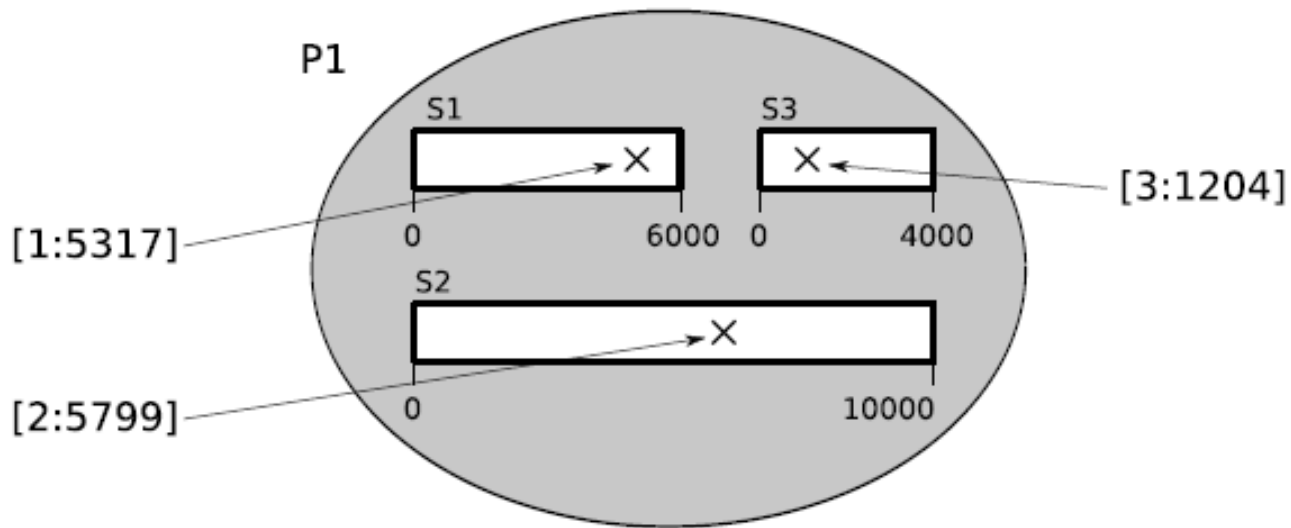
Tradução de endereços



Quem segmenta o processo?



Quem segmenta o processo?



Cabe ao **compilador** colocar os **diversos trechos** do código-fonte de cada programa em **segmentos** separados.

A decorative graphic on the left side of the slide, consisting of a vertical grey rectangle, a horizontal blue line, and two vertical bars (one light blue, one orange) at the bottom.

Tabela de Segmentos: Implementação

Tabela de Segmentos

- Caso o número de segmentos usados por cada processo seja pequeno
 - Tabela reside em registradores especializados do processador - MMU

Tabela de Segmentos

- Caso o número de segmentos usados por cada processo seja pequeno
 - Tabela reside em registradores especializados do processador - MMU
- Caso o número de segmentos por processo seja elevado
 - Necessário alocar as tabelas na memória RAM

Tabela de Segmentos - RAM

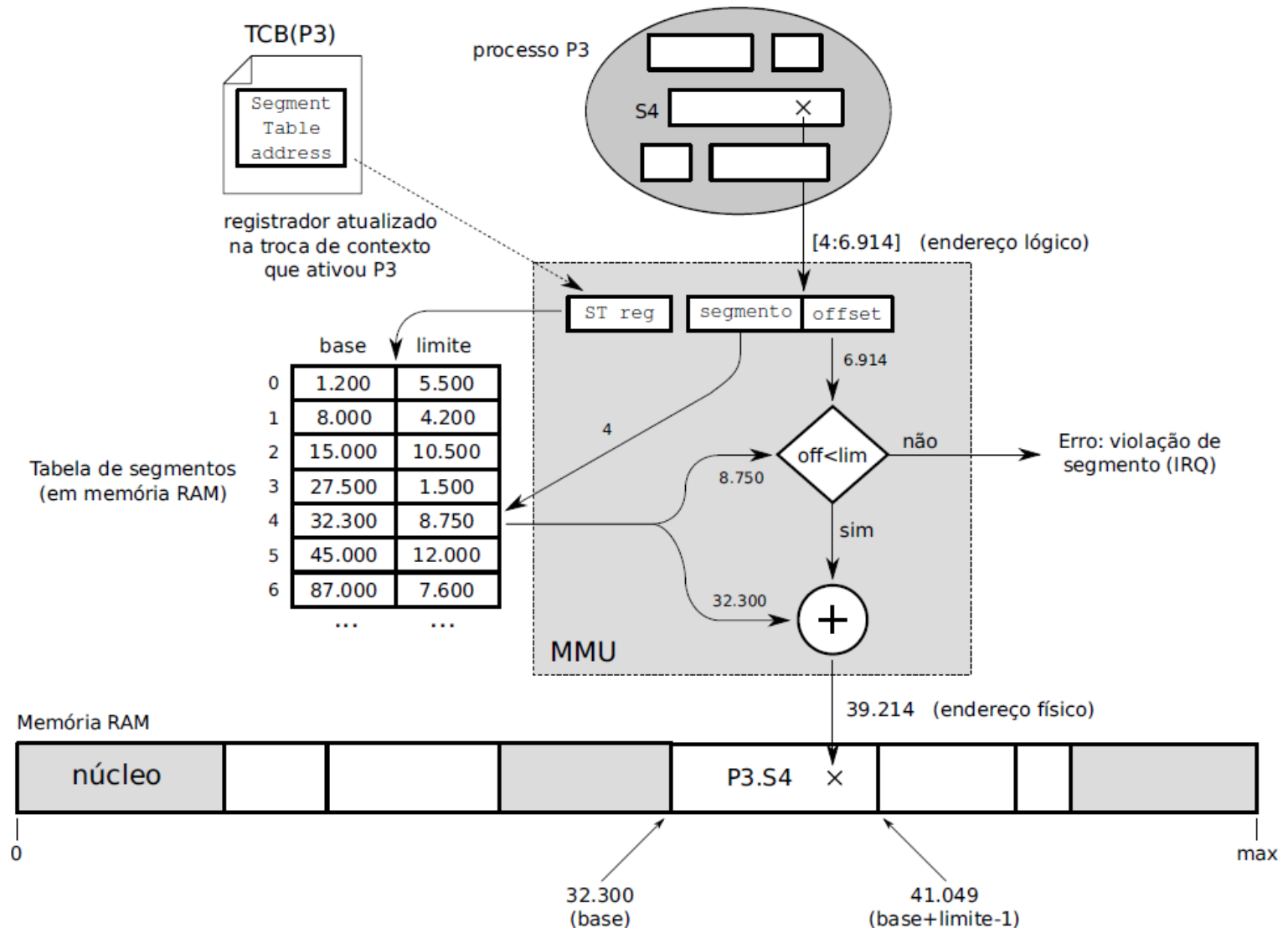
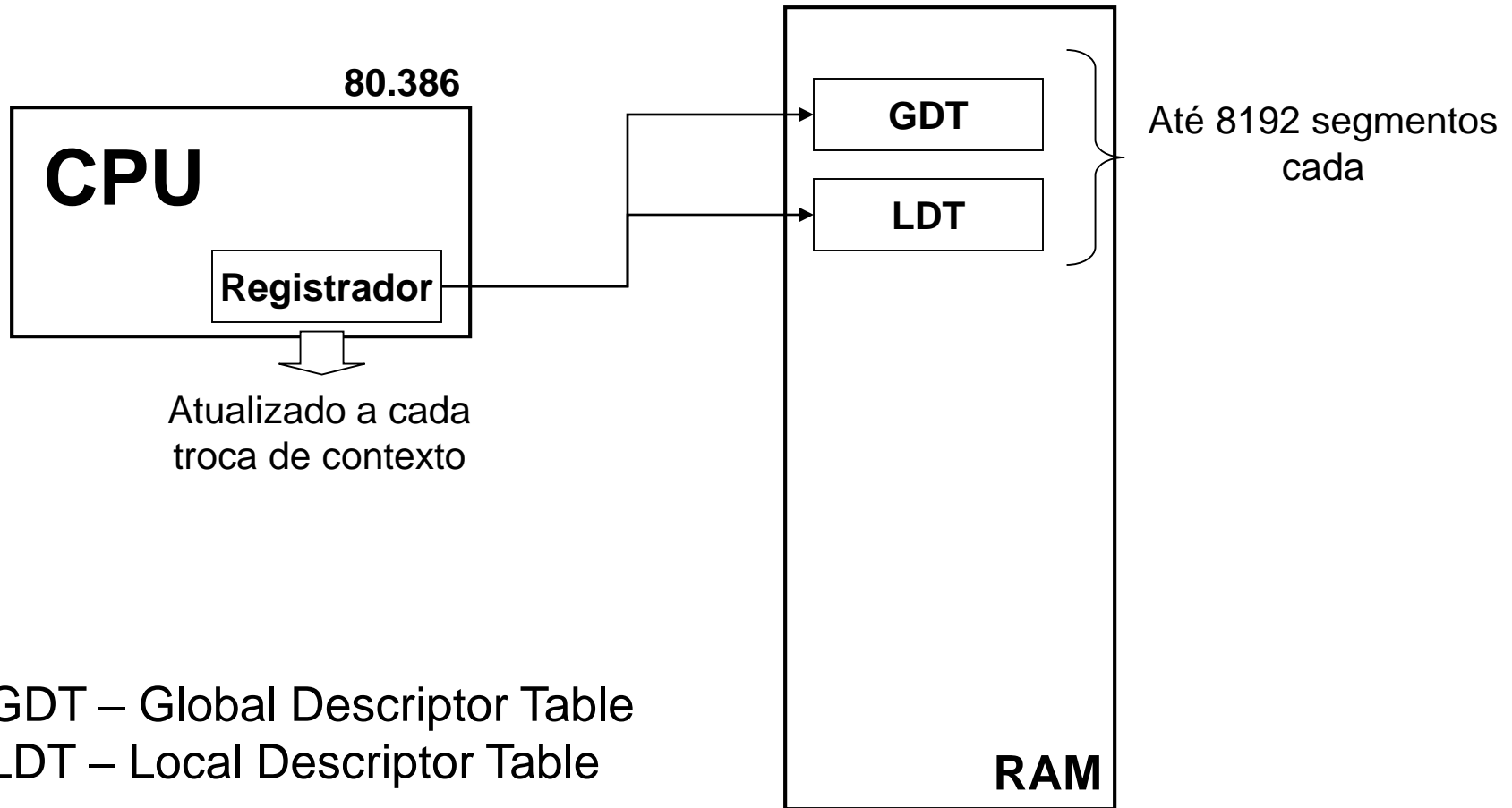
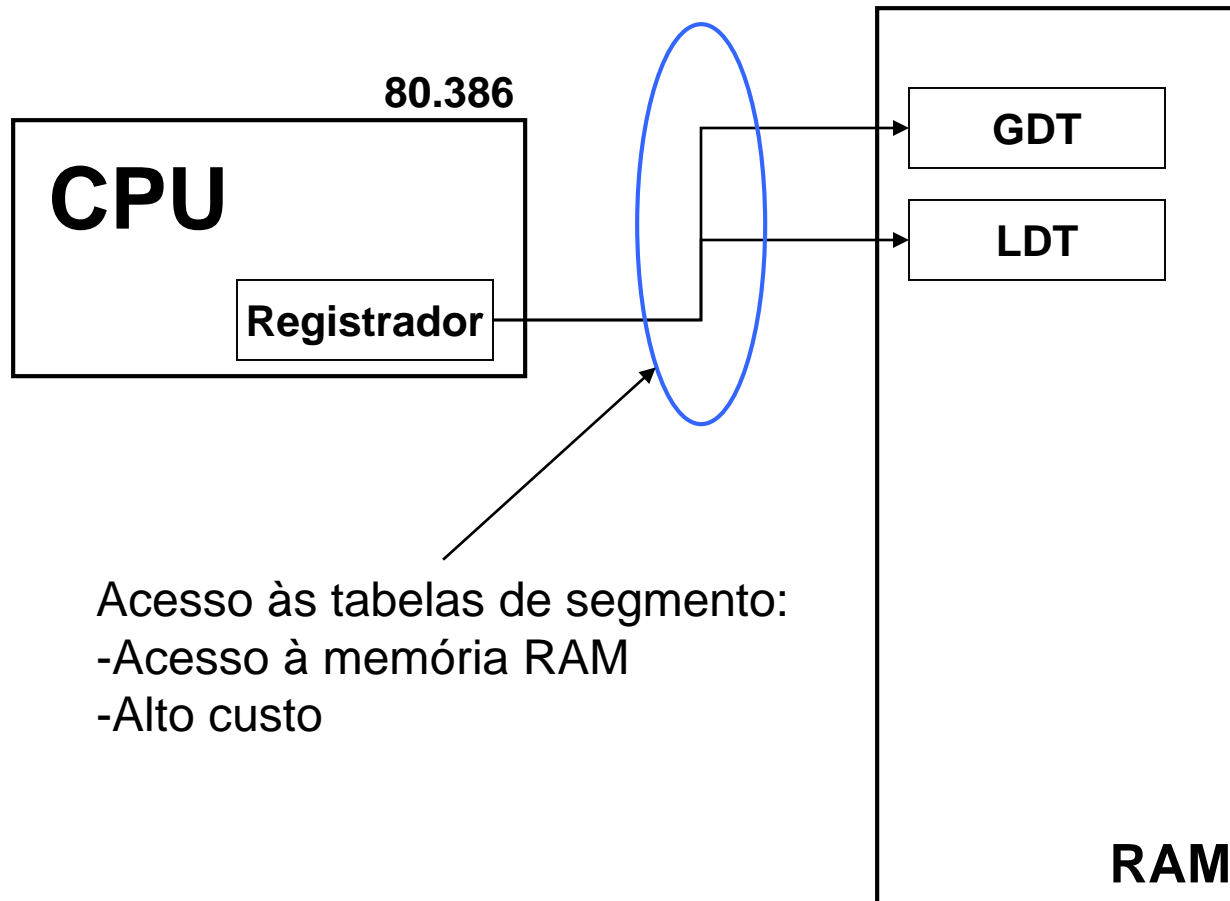


Tabela de Segmentos

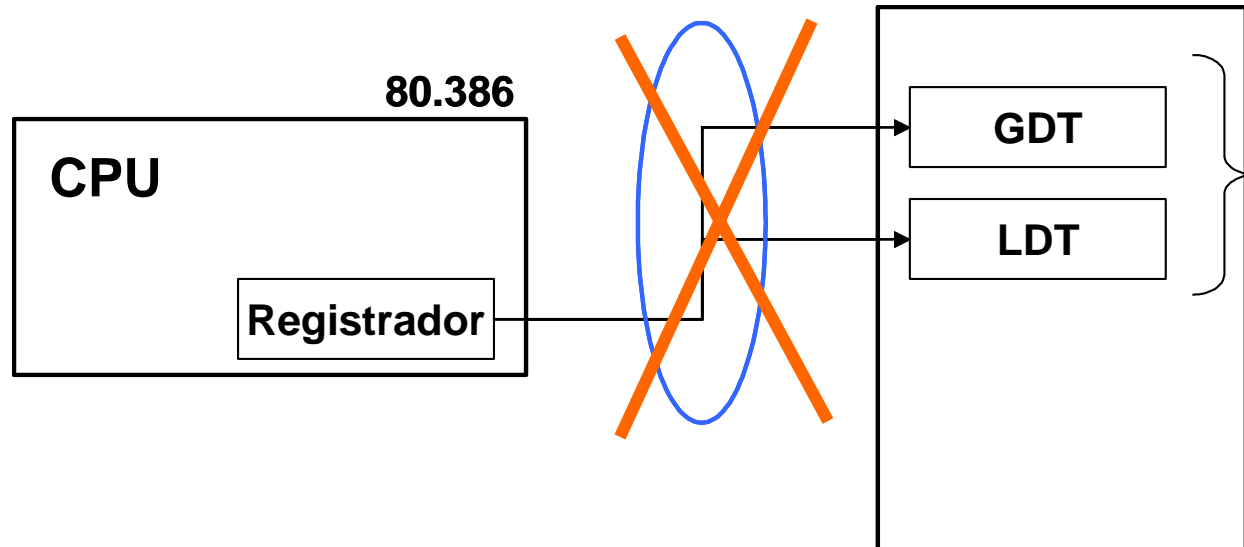


Alto Custo

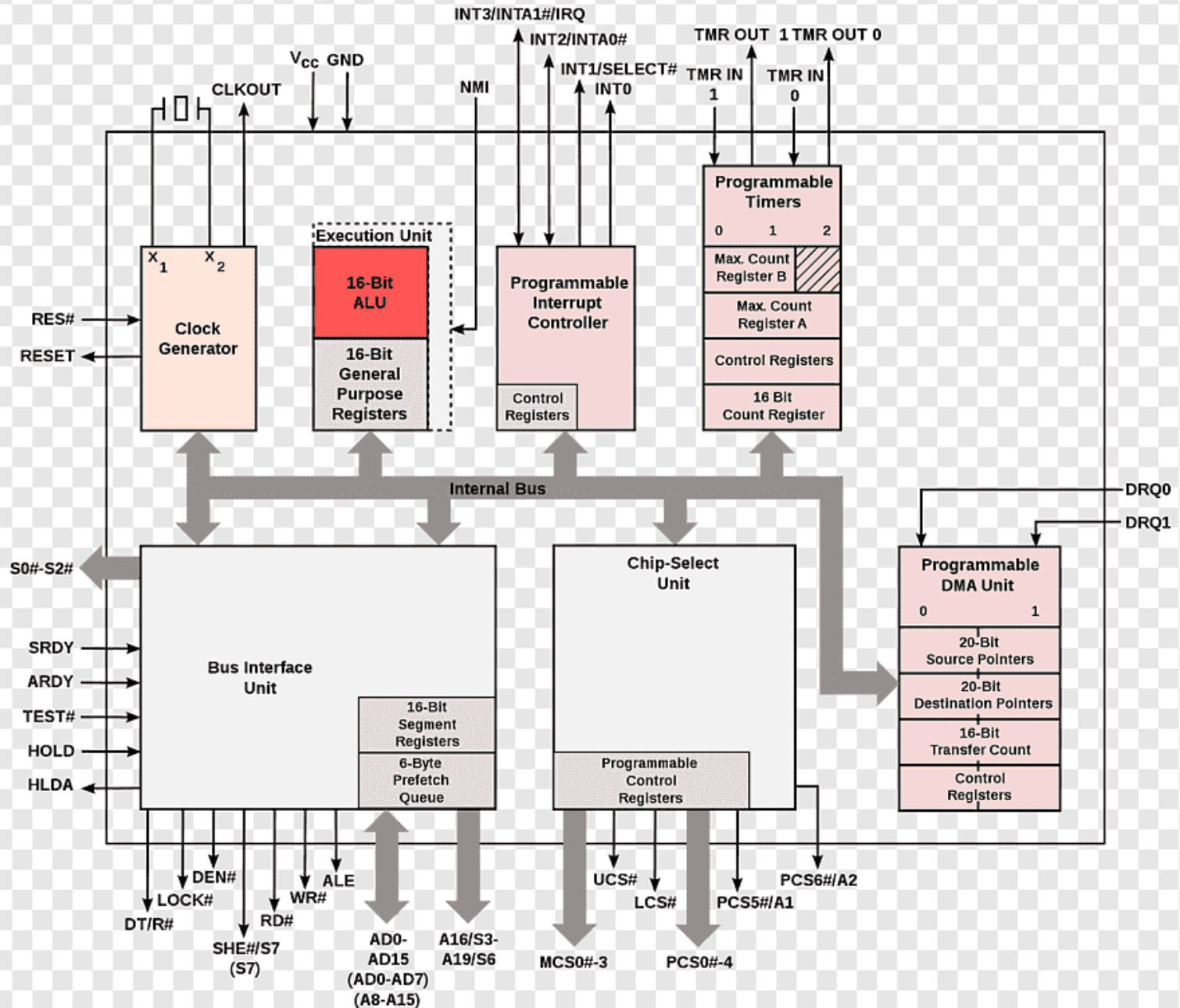


Registradores de Segmentos

Processadores usam **registradores de segmentos** \Rightarrow permitem armazenar os valores de base e limite dos **segmentos mais usados** pelo processo ativo.



Intel 80186 / 80188 architecture



Registradores de Segmentos

EX:. O **80.386** define os seguintes registradores de segmento:

CS: *Code Segment*, indica o segmento onde se encontra o código atualmente em execução;

SS: *Stack Segment*, indica o segmento onde se encontra a pilha em uso pelo processo atual;

DS, ES, FS e GS: *Data Segments*, indicam quatro segmentos com dados usados pelo processo atual, que podem conter variáveis globais, vetores ou áreas alocadas dinamicamente.

Registradores de Segmentos

- O conteúdo desses registradores é preservado no TCB (*Task Control Block*) de cada processo.
- Acesso à memória bastante eficiente caso poucos segmentos sejam usados simultaneamente.

Registradores de Segmentos

- O conteúdo desses registradores é preservado no TCB (*Task Control Block*) de cada processo.
- Acesso à memória bastante eficiente caso poucos segmentos sejam usados simultaneamente.
- Compilador tem uma grande responsabilidade na geração de código executável:
 - **minimizar** o número de segmentos necessários à execução do processo a “cada instante”
⇒ localidade de referências

Desvantagens da alocação segmentada

- Uso de endereço bidimensional
- Suscetível à fragmentação externa
- Armazenamento da tabela de segmentos na RAM

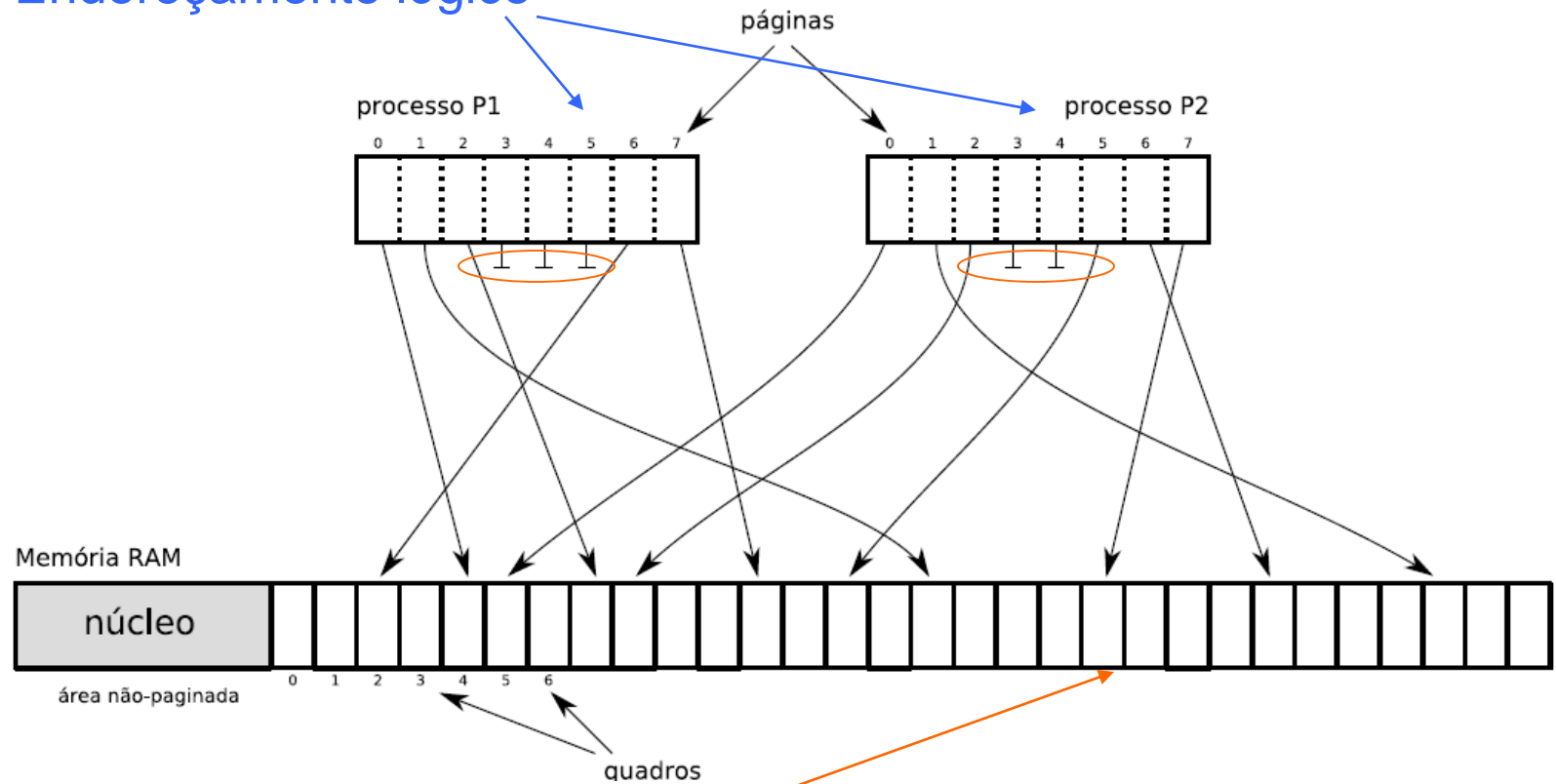
Desvantagens da alocação segmentada

- Uso de endereço bidimensional
- Suscetível à fragmentação externa
- Armazenamento da tabela de segmentos na RAM*

Solução → **Alocação Paginada**

Alocação Paginada

Endereçamento lógico



Endereçamento físico