Fundamentos de Sistemas de Operação

Unix Windows NT Netware MacOS DOS/VS Vax/VMS Linux Solaris HP/UX AIX Mach Chorus

Parte II: Gestão de Memória

Introdução

Processo: o Conceito

- O Conceito de Processo
  - Uma definição muito simples: um Programa em Execução
  - Usa recursos *hardware* de um sistema computacional:
    - Processador (gasta tempo/ciclos de processador)
    - Memória
    - I/O (periféricos)
  - Usa recursos/serviços *software*:
    - Ficheiros
    - Outros recursos/serviços: acesso a dados remotos; execução remota de procedimentos; memória partilhada; semáforos, *etc*.

O que é um Programa? (visão simplificada)

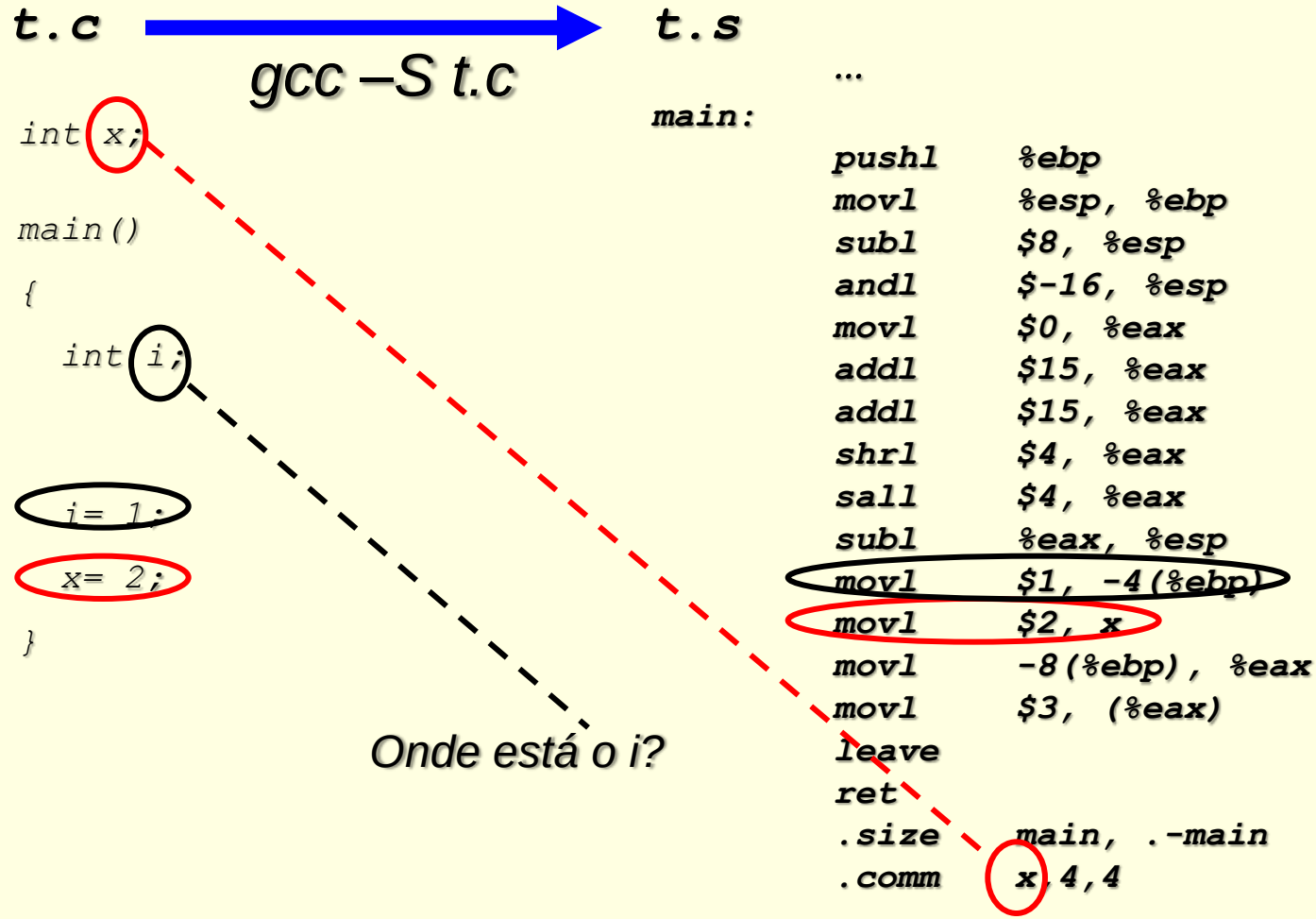
Unix Windows NT Netware MacOS DOS/VS Vax/VMS Linux Solaris HP/UX AIX Mach Chorus

- 1: O Programa Fonte (3GL's: C, FORTRAN, …)
  - Código
    - Programa Principal
    - Rotinas: Funções e Procedimentos
  - Dados
    - Constantes *vs*. Variáveis
    - Dados Globais *vs*. Locais
    - Dados Estáticos *vs*. Dinâmicos

O que é um Programa? (visão simplificada)

- 2: O ficheiro (“programa”) Objecto
  - Código máquina
    - Programa Principal
    - Rotinas do programa fonte: Funções e Procedimentos
  - Dados
    - Constantes
    - Variáveis inicializadas, dimensão da zona que vai conter as variáveis não-inicializadas
  - Referências a “Objectos” externos
    - Rotinas da Biblioteca de Suporte da Linguagem
    - Dados externos

O que é um Programa? (do C ao assembly)



O que é um Programa? (ficheiro objecto)

`gcc -c t.c → t.o` *objdump –D t.o*

```
...
00000000 <main>:
   0:    55                       push   %ebp
   1:    89 e5                    mov    %esp,%ebp
   3:    83 ec 08                 sub    $0x8,%esp
   6:    83 e4 f0                 and    $0xfffffff0,%esp
   9:    b8 00 00 00 00           mov    $0x0,%eax
   e:    83 c0 0f                 add    $0xf,%eax
  11:    83 c0 0f                 add    $0xf,%eax
  14:    c1 e8 04                 shr    $0x4,%eax
  17:    c1 e0 04                 shl    $0x4,%eax
  1a:    29 c4                    sub    %eax,%esp
  1c:    c7 45 fc 01 00 00 00     movl   $0x1,0xfffffffc(%ebp)
  23:    c7 05 dc 95 04 08 02     movl   $0x2,0x0
  2a:    00 00 00
  2d:    8b 45 f8                 mov    0xfffffff8(%ebp),%eax
  30:    c7 00 03 00 00 00        movl   $0x3,(%eax)
  36:    c9                       leave
  37:    c3                       ret
```

O programa começa no endereço 0?

Onde é guardada a variável i?

A variável x é guardada no endereço 0?

O que é um Programa? (ficheiro executável)

`gcc -o t.exe t.c`

```
...
08048374 <main>:
 8048374: 55                    
 8048375: 89 e5                 
 8048377: 83 ec 08              
 804837a: 83 e4 f0              
 804837d: b8 00 00 00 00        
 8048382: 83 c0 0f              
 8048385: 83 c0 0f              
 8048388: c1 e8 04              
 804838b: c1 e0 04              
 804838e: 29 c4                 
 8048390: c7 45 fc 01 00 00 00  
 8048397: c7 05 dc 95 04 08 02  
 804839e: 00 00 00              
 80483a1: 8b 45 f8              
 80483a4: c7 00 03 00 00 00     
 80483aa: c9                    
 80483ab: c3                    
```

O código executável do programa é para ser carregado em memória a partir da posição 0x8048374

objdump –D t.exe

```
push   %ebp
mov    %esp,%ebp
sub    $0x8,%esp
and    $0xfffffff0,%esp
mov    $0x0,%eax
add    $0xf,%eax
add    $0xf,%eax
shr    $0x4,%eax
shl    $0x4,%eax
sub    %eax,%esp
movl   $0x1,0xfffffffc(%ebp)
movl   $0x2,0x80495dc

mov    0xfffffff8(%ebp),%eax
movl   $0x3,(%eax)
leave
ret
```

A variável i é guardada no endereço ebp-4, que é um endereço no stack

A variável x é guardada no endereço 0x80495dc

Execução de um Programa (simulação)

(1) IP contém endereço de arranque do programa.

(2) O endereço é "emitido" para a memória (no address bus) juntamente com a indicação de que se trata de um READ.

(3) A memória entrega os bytes.

(4) A unidade de controlo descodifica a instrução; se é válida, despacha-a para ser executada.

(5) O IP é incrementado; fica a "apontar" (contém o endereço) da instrução seguinte.

(6) Todo o processo é repetido…


```
…

main:
pushl   %ebp
movl    %esp, %ebp
subl    $8, %esp
andl    $-16, %esp
movl    $0, %eax
addl    $15, %eax
addl    $15, %eax
shrl    $4, %eax
sall    $4, %eax
subl    %eax, %esp
movl    $1, -4(%ebp)
movl    $2, x
movl    -8(%ebp), %eax
movl    $3, (%eax)
leave
ret
.size   main, .-main
.comm   x,4,4
```

Execução de um Programa (simulação)

Execução de uma instrução que acede a memória:

`movl $1, -4(%ebp)`

(para a dupla-word (32 bits) que fica localizada quatro posições abaixo da posição apontada pelo ebp é transferido o valor 1). Como?

(1) O endereço efectivo é calculado: é o valor de ebp subtraído de 4.

(2) O endereço efectivo é colocado no address bus.

(3) O valor 1 é colocado no data bus.

(4) É dada uma ordem de WRITE para a memória.

```
                …
main:
        pushl   %ebp
        movl    %esp, %ebp
        subl    $8, %esp
        andl    $-16, %esp
        movl    $0, %eax
        addl    $15, %eax
        addl    $15, %eax
        shrl    $4, %eax
        sall    $4, %eax
        subl    %eax, %esp
        movl    $1, -4(%ebp)
        movl    $2, x
        movl    -8(%ebp), %eax
        movl    $3, (%eax)
        leave
        ret
        .size   main, .-main
        .comm   x,4,4
```

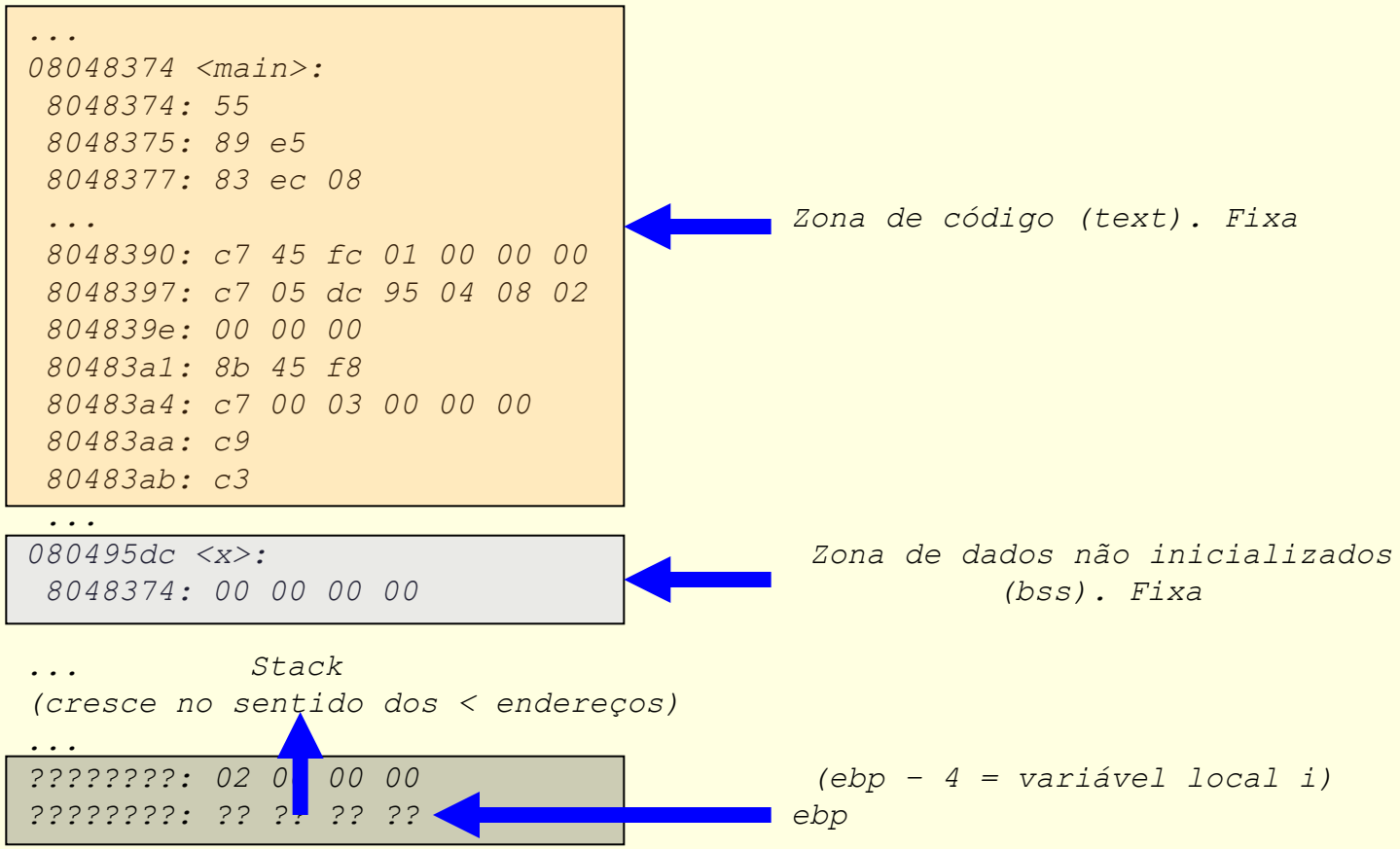
Execução de um Programa (simulação)


```
…
08048374 <main>:
 8048374: 55                          push   %ebp
 8048375: 89 e5                       mov    %esp,%ebp
 8048377: 83 ec 08                    sub    $0x8,%esp
 804837a: 83 e4 f0                    and    $0xfffffff0,%esp
 804837d: b8 00 00 00 00              mov    $0x0,%eax
 8048382: 83 c0 0f                    add    $0xf,%eax
 8048385: 83 c0 0f                    add    $0xf,%eax
 8048388: c1 e8 04                    shr    $0x4,%eax
 804838b: c1 e0 04                    shl    $0x4,%eax
 804838e: 29 c4                       sub    %eax,%esp
 8048390: c7 45 fc 01 00 00 00        movl   $0x1,0xfffffffc(%ebp)
 8048397: c7 05 dc 95 04 08 02        movl   $0x2,0x80495dc
 804839e: 00 00 00
 80483a1: 8b 45 f8                    mov    0xfffffff8(%ebp),%eax
 80483a4: c7 00 03 00 00 00           movl   $0x3,(%eax)
 80483aa: c9                          leave
 80483ab: c3                          ret
```

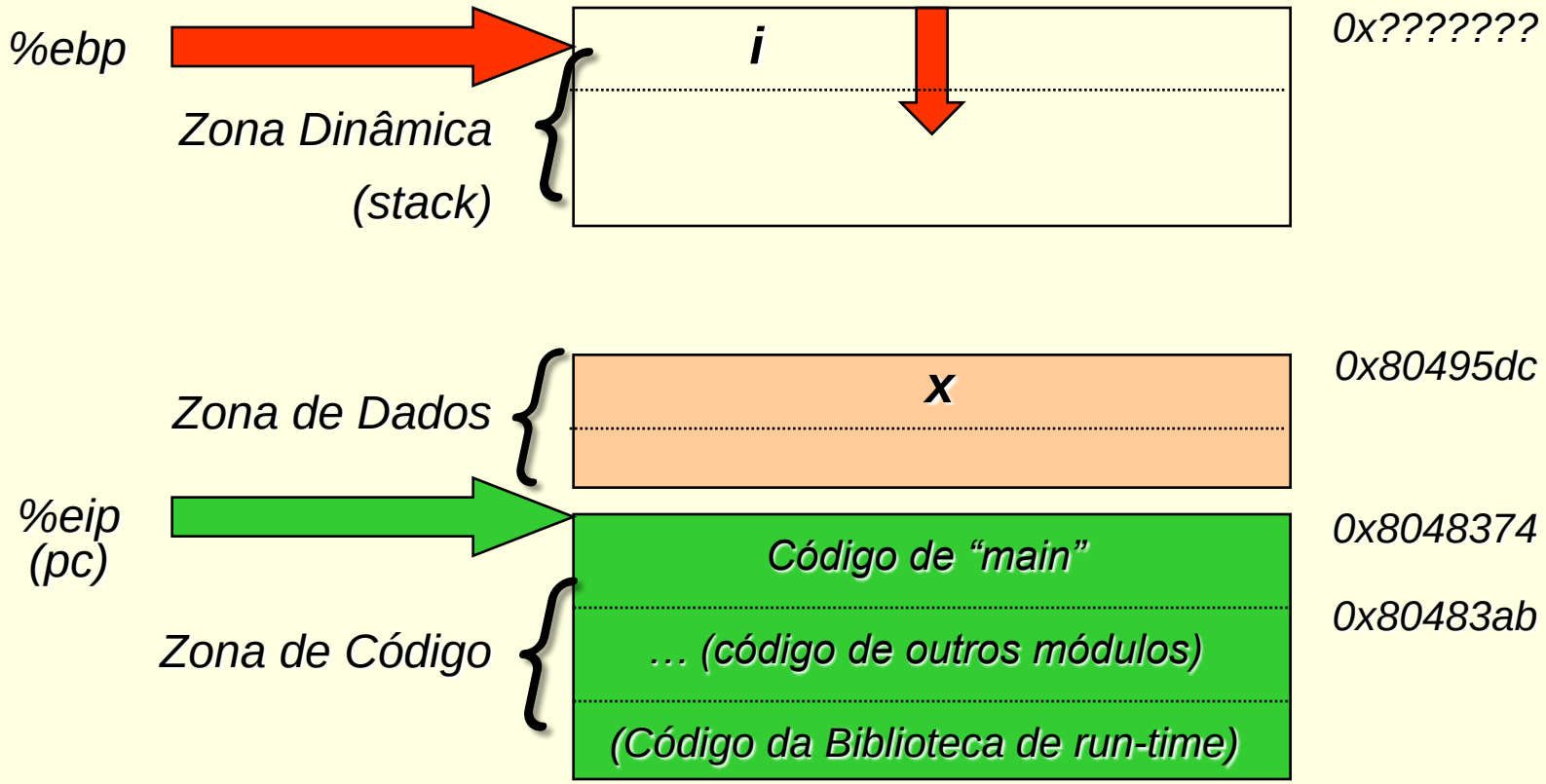
O IP "percorre" estes endereços (i.e., injecta-os no bus de endereços) para que o CPU "receba" as instruções…

Injectando este valor no bus de endereços acede-se à variável x

Espaço de Endereçamento (deduzido a partir do programa executável)



Mapa (incompleto) do Espaço de Endereçamento
(Linux, conclusões obtidas observando o ficheiro executável)



Fundamentos de Sistemas de Operação

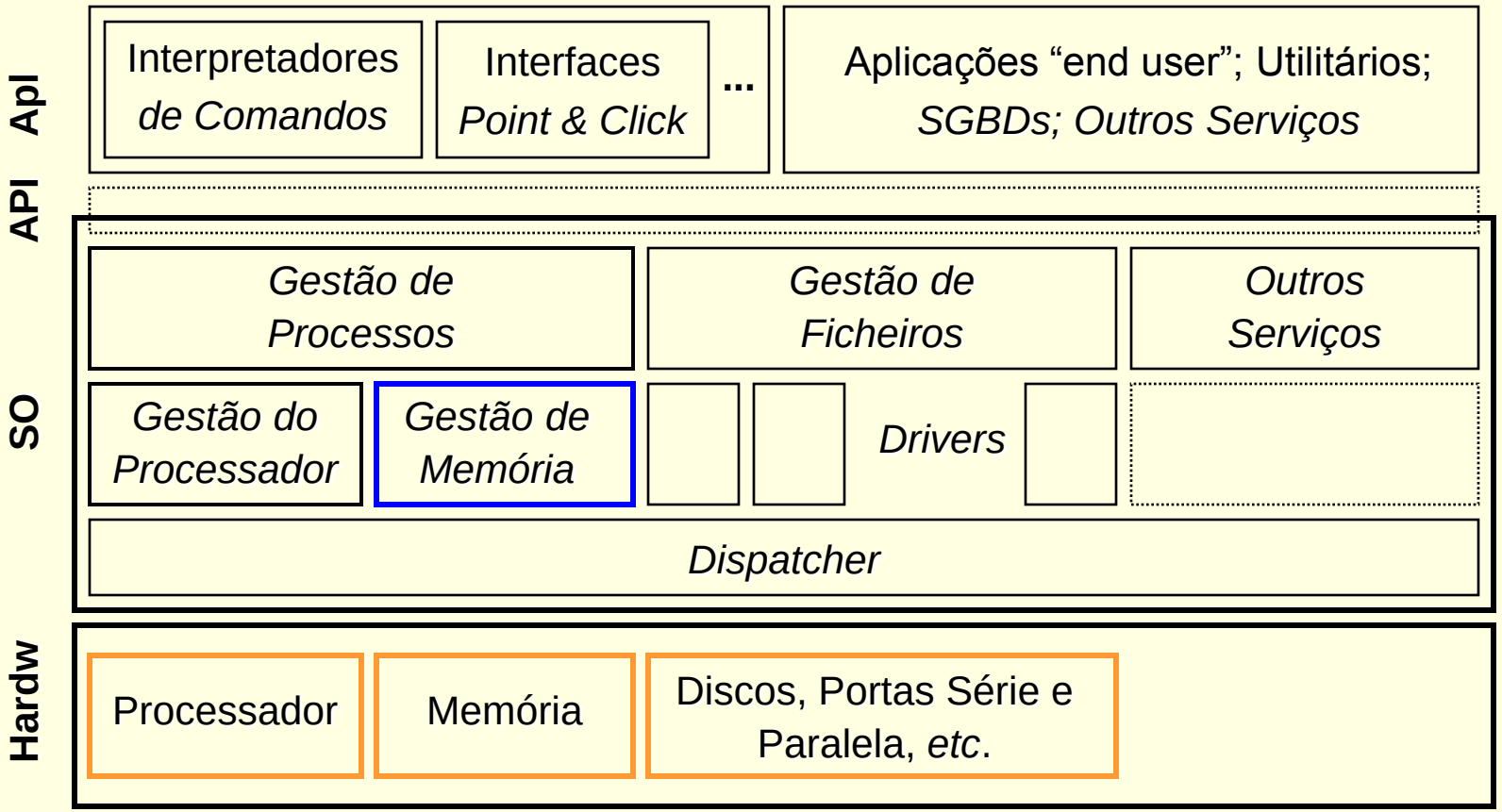
Unix Windows NT Netware MacOS DOS/VS Vax/VMS Linux Solaris HP/UX AIX Mach Chorus

Parte II: Gestão de Memória

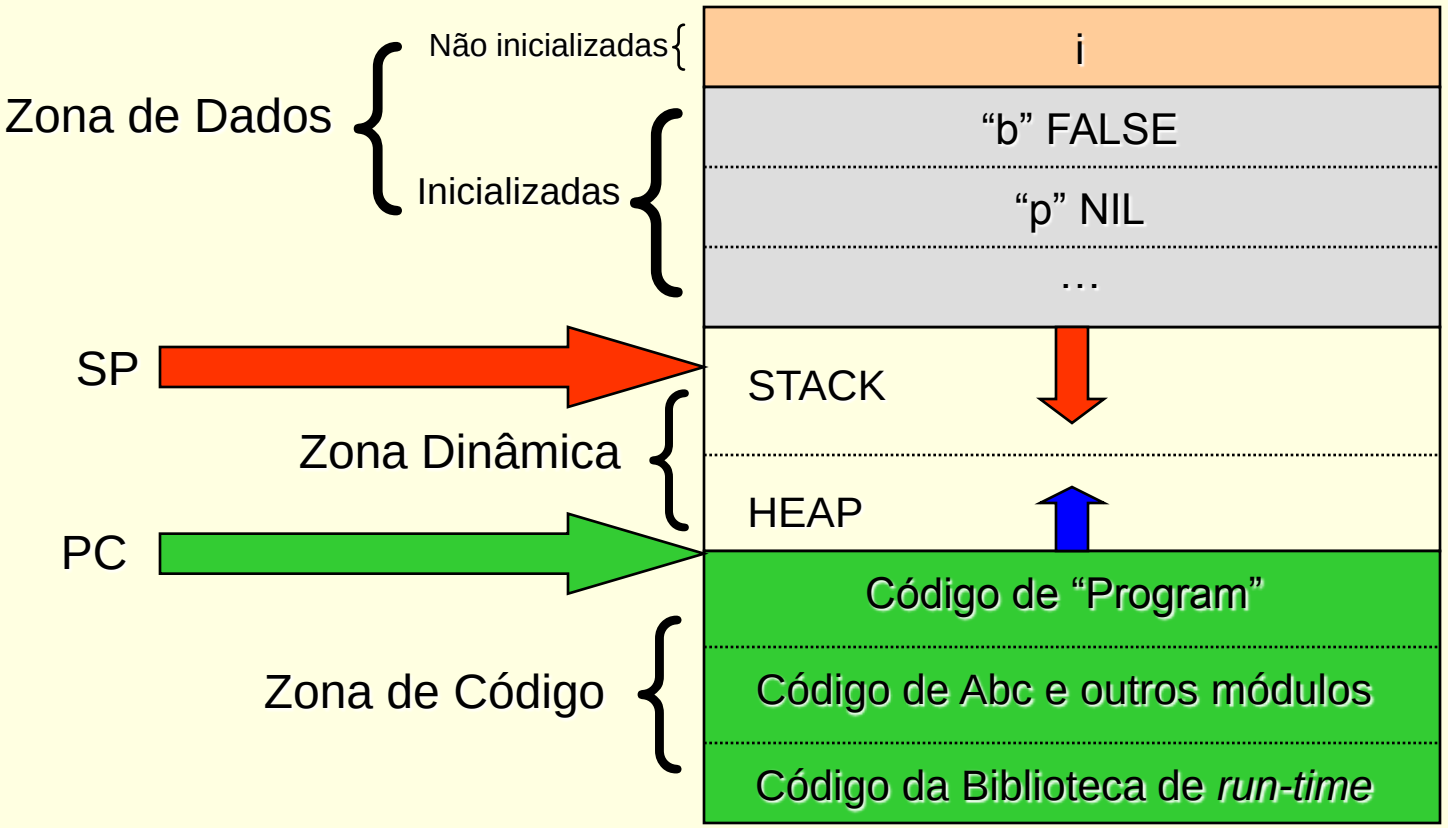
a) Partição única

A Gestão de Memória

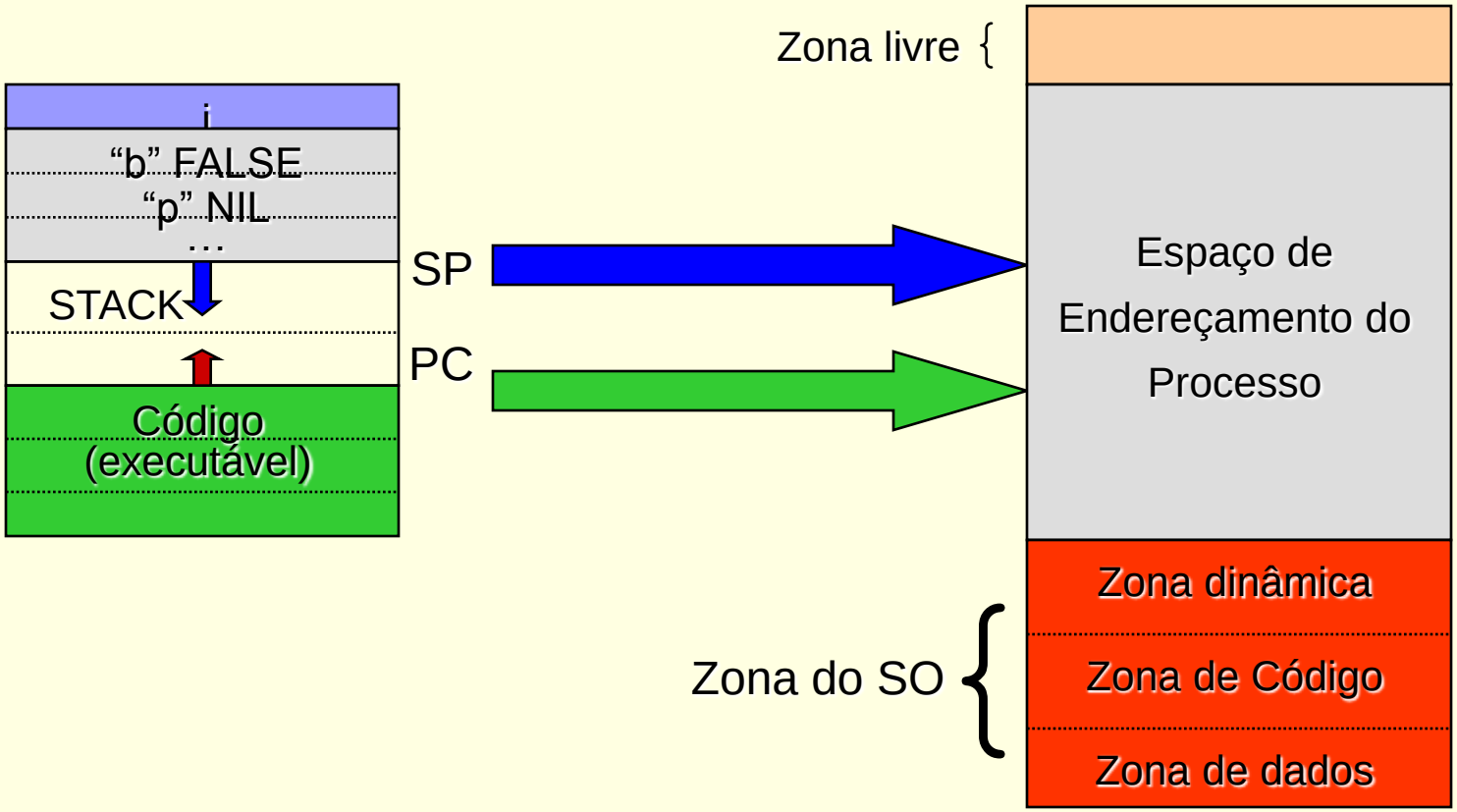
Unix Windows NT Netware MacOS DOS/VS Vax/VMS
Linux Solaris HP/UX AIX Mach
Chorus



Como colocar em RAM o Processo?



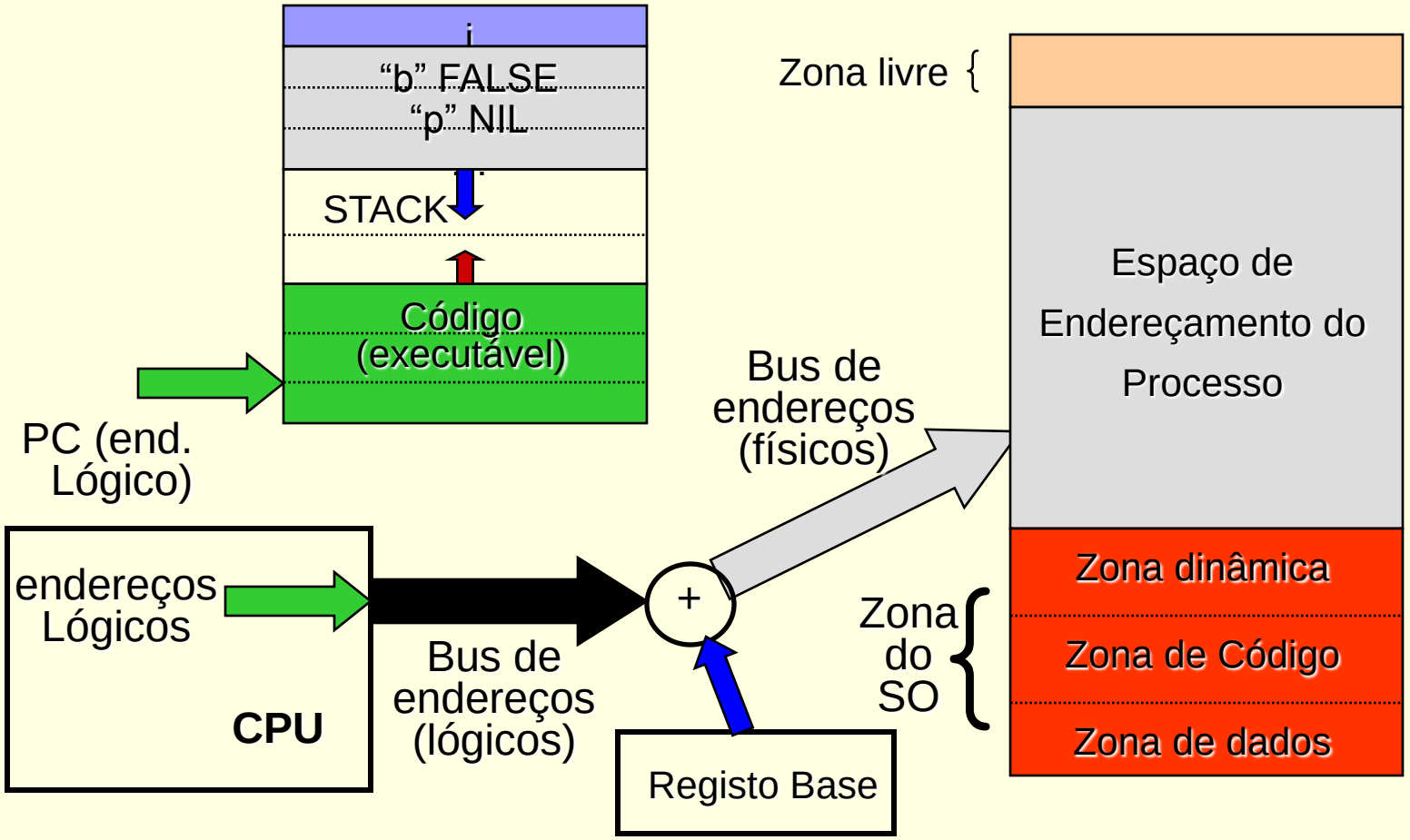
Gestão de Memória por Partição única



Gestão de Memória por Partição única

- Vantagens:
  - Algoritmo de GM muito simples: ler cabeçalho do programa executável para obter dimensão do EE; se dimensão <= RAM livre, carregar e executar; senão, mensagem de erro.
- Desvantagens:
  - Se a dimensão da zona ocupada pelo SO muda, os programas exe têm de ser re-criados (*link* de novo).
  - Nada a fazer quando executável não cabe na RAM livre.
- Questão
  - Como proteger a zona do SO de um processo errado ou "mal comportado" (que tenta escrever e/ou ler na zona do SO)?

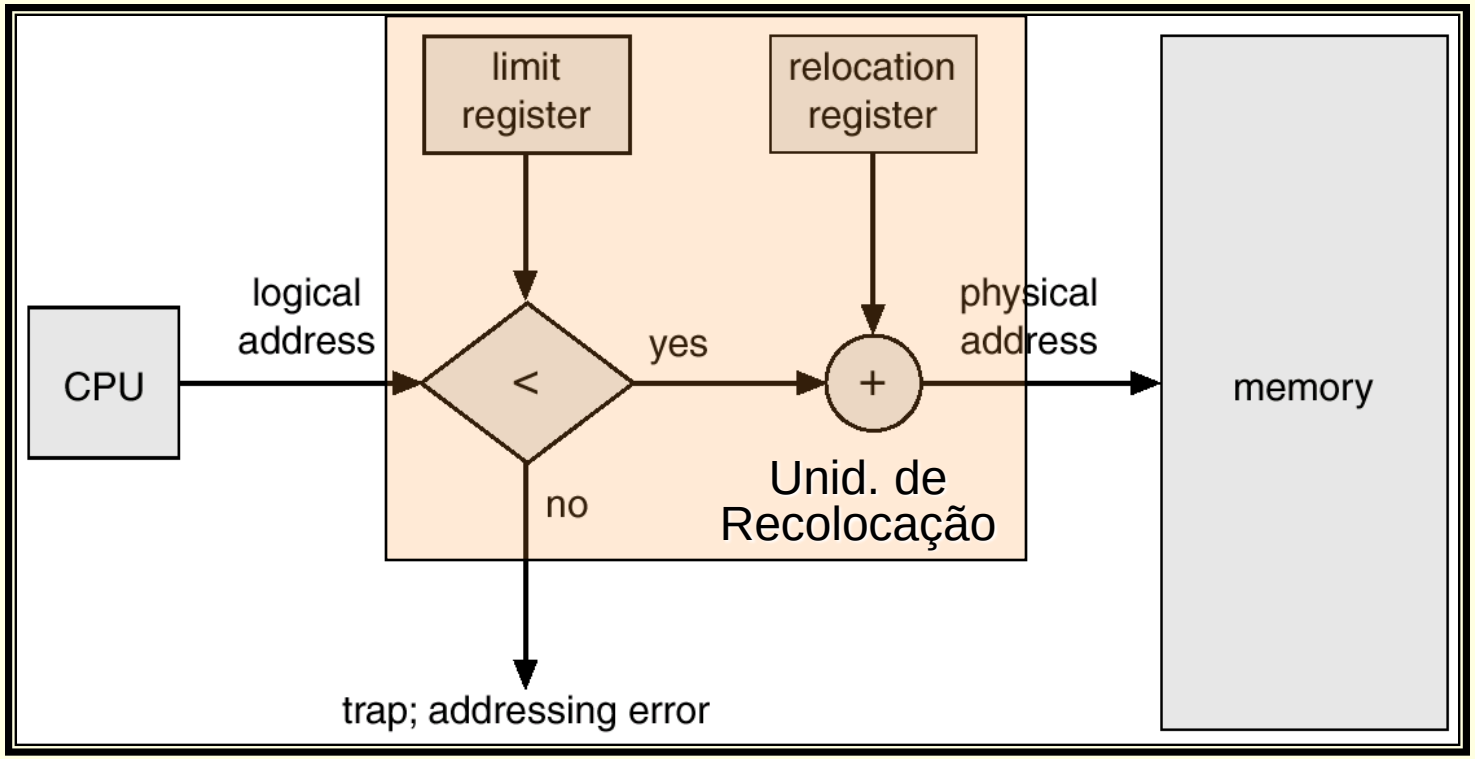
GM com Recolocação: Registo Base



GM com Recolocação: Registo Base

- Espaço de Endereçamento Lógico (ou virtual):
  - O CPU vê/referencia os endereços tal qual como estes se apresentam no espaço do processo – endereços lógicos.
- Unidade de Recolocação:
  - Adiciona um *valor,* guardado num registo chamado de *recolocação* ou *base* a cada endereço "emitido" pelo CPU.
- Espaço de Endereçamento Físico (ou real):
  - O espaço dos endereços efectivamente "comunicados" à memória.

GM com Registo Base e Registo Limite



Fonte: OSC

GM com Registo Base e Registo Limite: vantagens

- Proteger o Espaço de Endereçamento Físico:
  - O Limite – endereço "mais alto" do espaço de endereçamento é guardado num registo limite (RL) da UR.
- Unidade de Recolocação de Endereços:
  - Se o endereço lógico é superior a RL, não efectuar a tradução e gerar uma interrupção.
  - Senão, gerar o endereço físico por adição do *valor base* ao endereço "emitido" pelo CPU.
- Protege o EE dos outros processos (nas técnicas que iremos estudar em seguida) e do SO; ajuda o *debugging*!

GM com Registo Base e Registo Limite: desvantagens

- Acesso a memória mais lento:
  - A Unidade de Recolocação leva algum tempo a comparar o valor do endereço emitido pelo CPU com o RL,
  - E leva mais algum tempo a fazer a adição do RB para gerar o endereço físico.
  - Mas, sendo essas operações MUITO simples de implementar (em termos de circuitos electrónicos, ou *gates*), o acréscimo de tempo pode ser muito pequeno face ao tempo total gasto num acesso à memória.

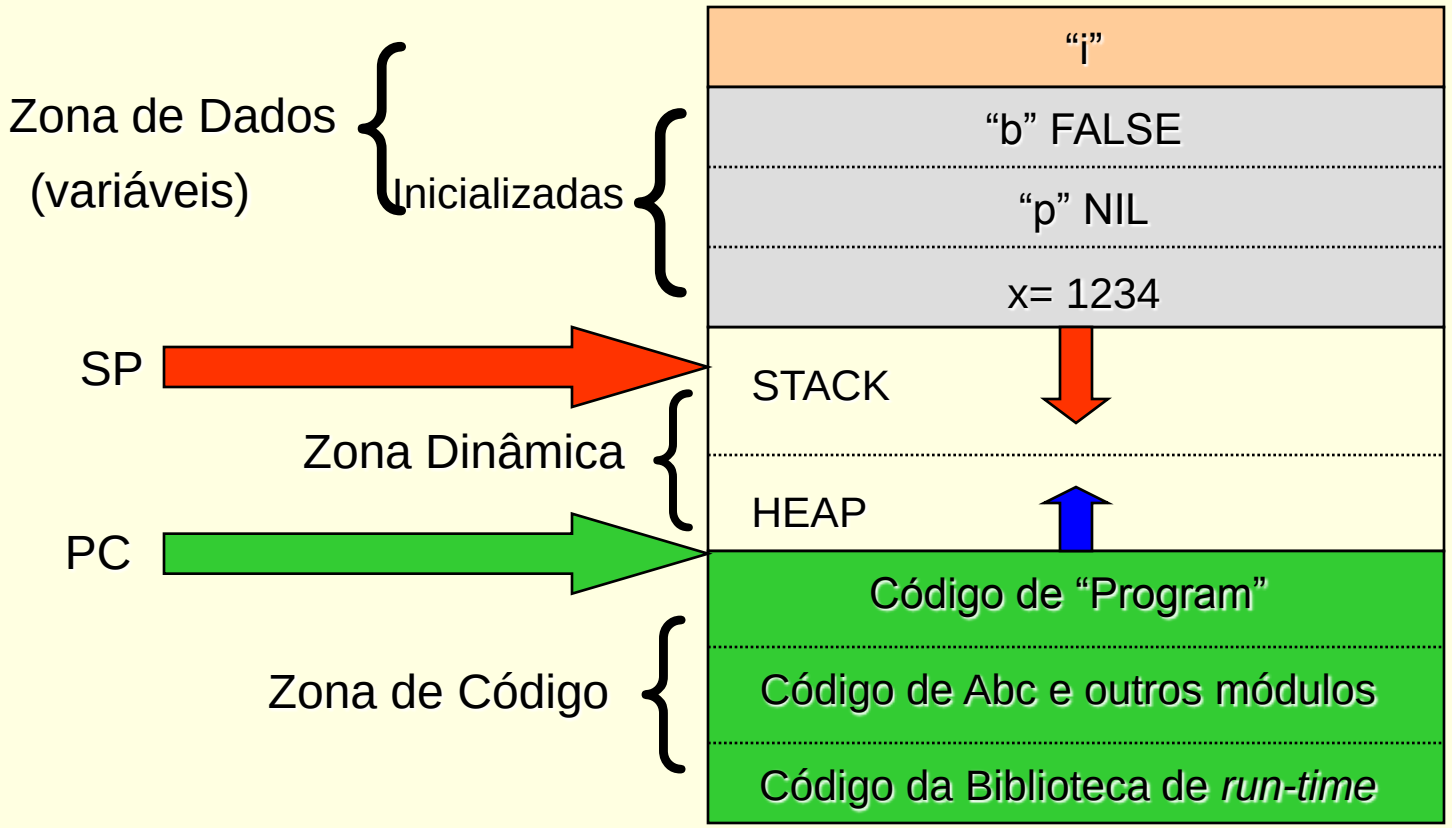
Fundamentos de Sistemas de Operação

Unix Windows NT Netware MacOS DOS/VS Vax/VMS Linux Solaris HP/UX AIX Mach Chorus

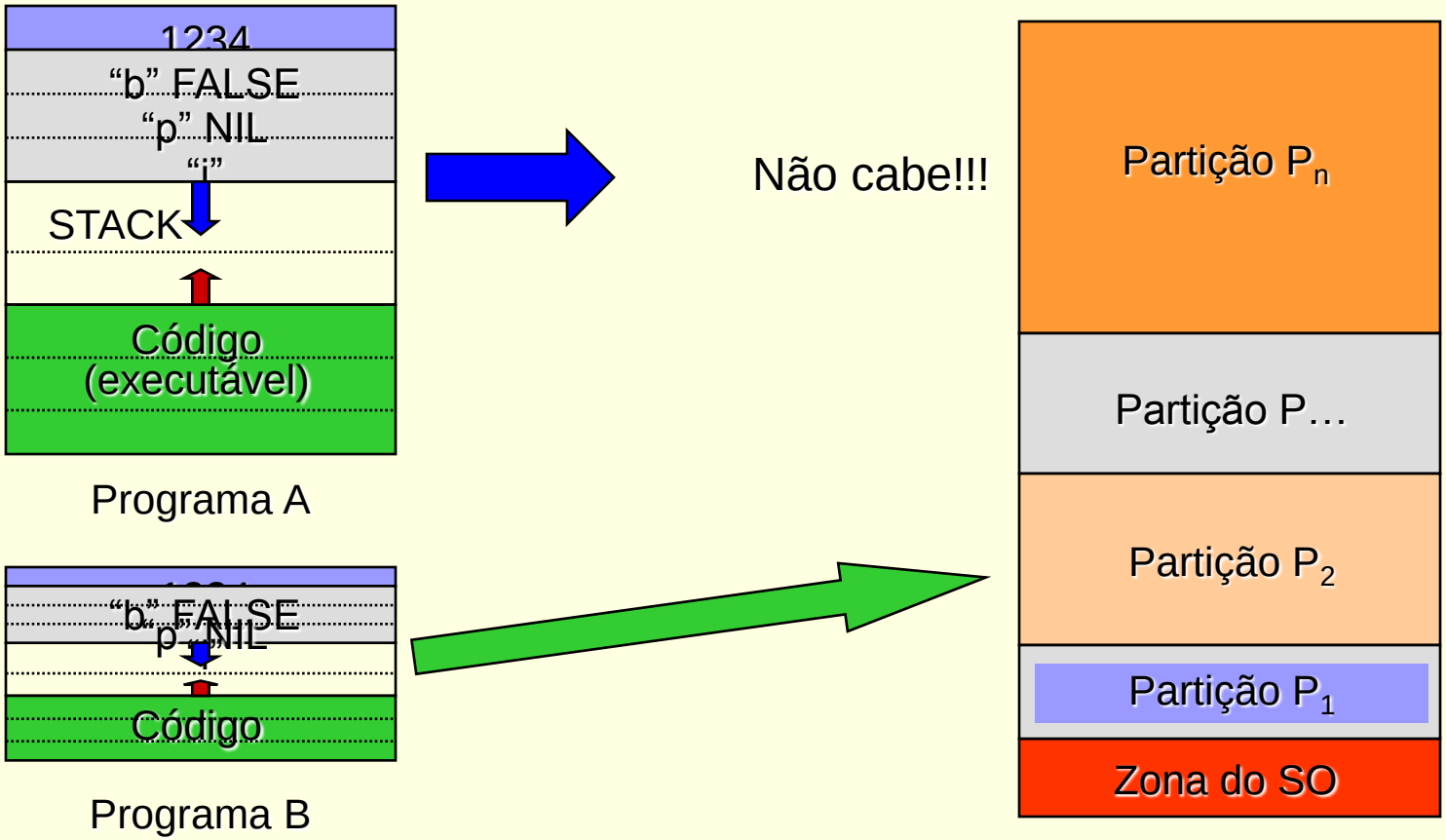
Parte II: Gestão de Memória

b) Partições fixas, contíguas

Como colocar em RAM múltiplos Processos?



GM por Partições fixas, contíguas



GM por Partições fixas, contíguas

- Benefícios:
  - Multiprogramação – vários processos em memória, podem ser “simultaneamente” executados pelo CPU.
- Estratégia de Colocação:
  - **Best Fit**: programa colocado na partição “mais ajustada” à sua dimensão.
  - **Worst Fit**: programa colocado na maior partição livre!
  - **First Fit**: programa colocado na primeira partição que se sabe livre e de dimensão suficiente!
  - Na GM por partições fixas usava-se sempre a estratégia Best Fit.

GM por Partições fixas, contíguas

- Desvantagens:
  - Nada a fazer quando o exe não cabe, ou quando nº de processos é superior ao nº de partições.
  - Nº e dimensão das partições definido no arranque do SO; para alterar era preciso fazer shutdown e reboot.
  - Fragmentação interna – memória que fica livre no "interior" de cada partição não pode ser usada para nada!
- Suporte hardware necessário:
  - UR com RB e RL.
  - Carregamento da UR é uma instrução privilegiada (Pense: porquê?)

GM por Partições fixas, contíguas

- Estruturas de dados para a GM por P. Fixas:
  - Tabela com uma linha por partição para manter: indicação da dimensão da partição; RB e RL do processo que ocupar a partição (RB=RL=0, partição livre);...
- Algoritmo para a GM por Partições Fixas:
  - Estratégia de colocação Best fit!
- Escalonamento dos Processos – um exemplo:
  - Há processos “em fila” à espera de “entrar na memória”
  - Sempre que há uma partição livre, procurar o mais antigo que cabe nessa partição, e carregá-lo!

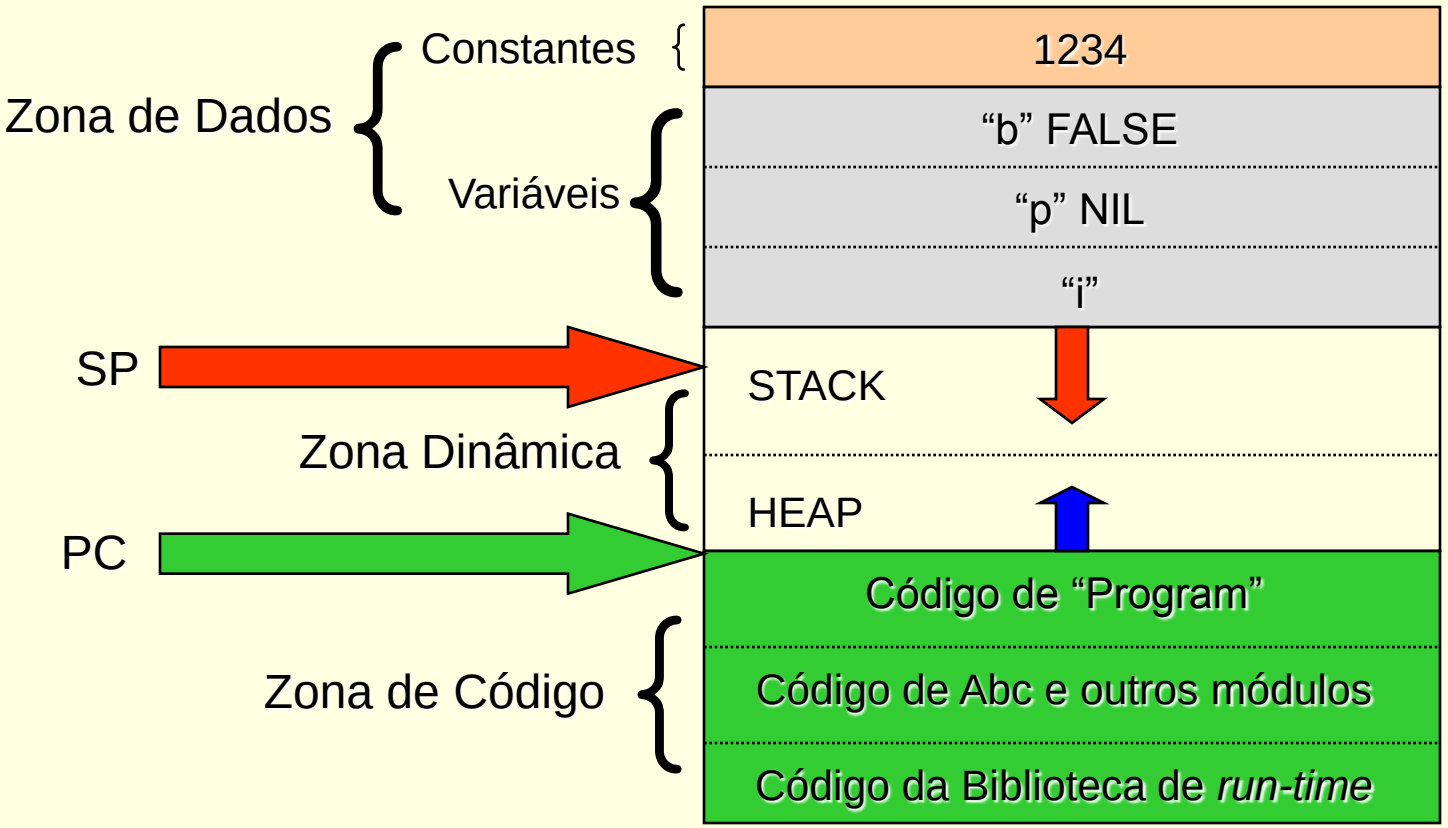
Fundamentos de Sistemas de Operação

Unix Windows NT Netware MacOS DOS/VS Vax/VMS Linux Solaris HP/UX AIX Mach Chorus

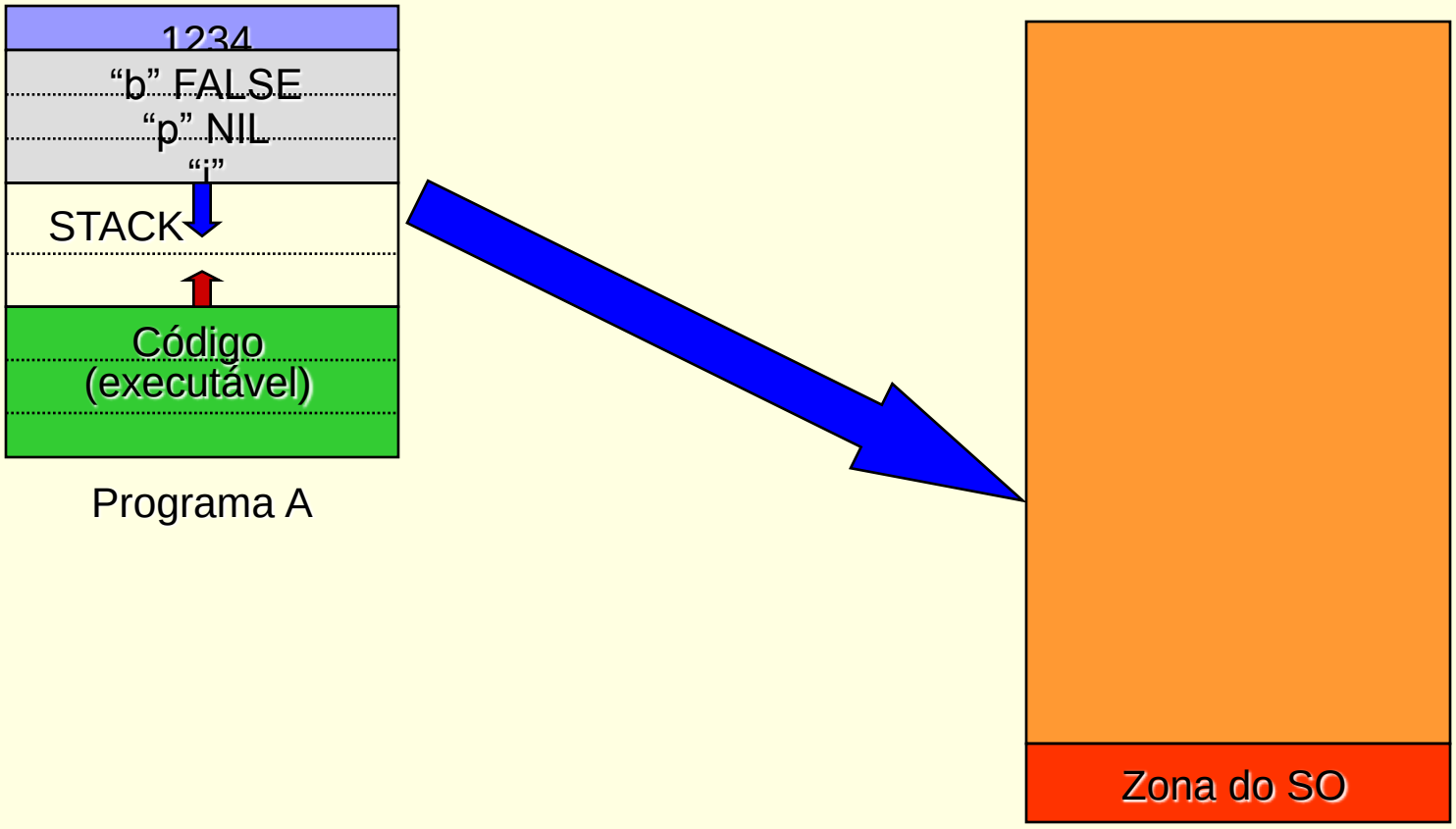
Parte II: Gestão de Memória

c) Partições dinâmicas, contíguas

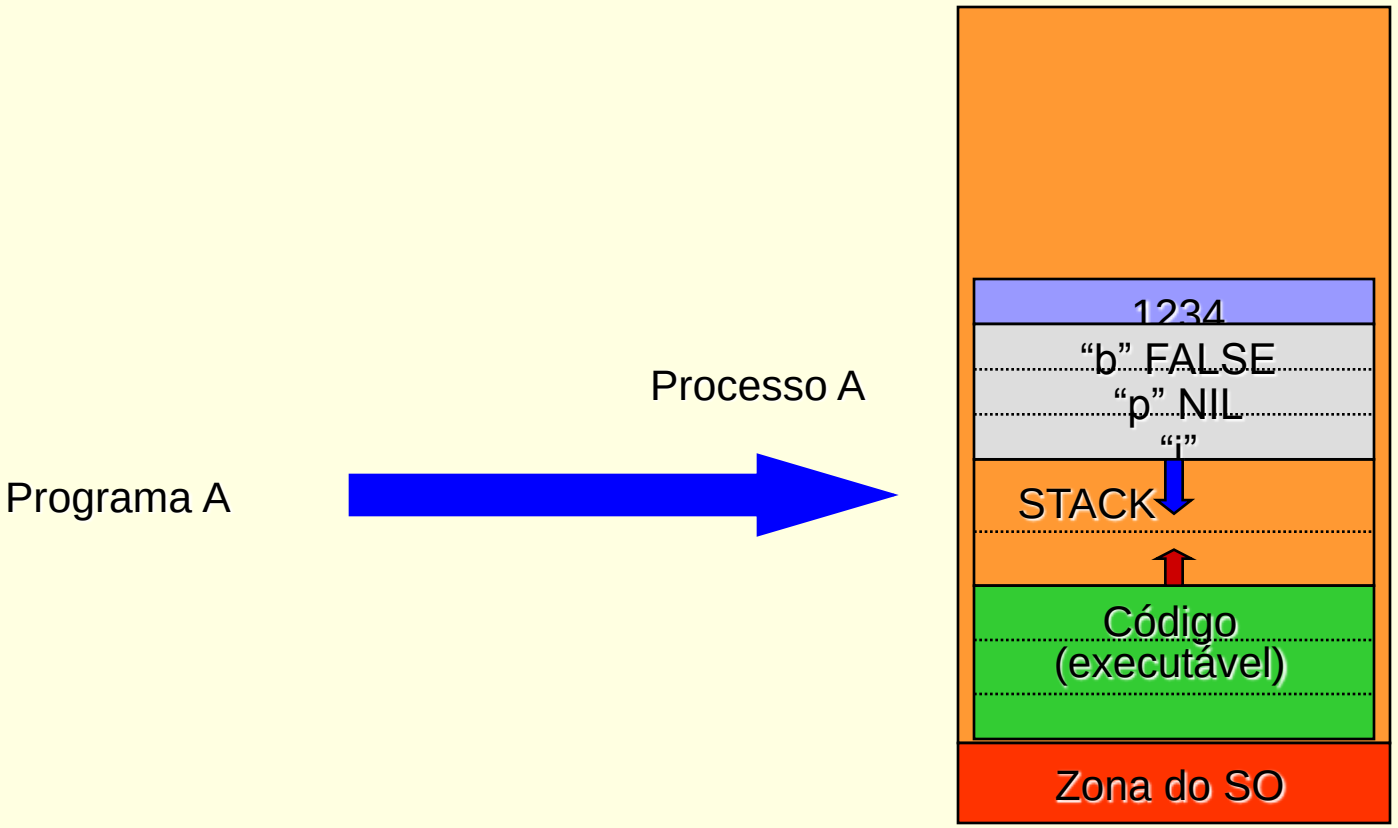
Como colocar em RAM múltiplos Processos?



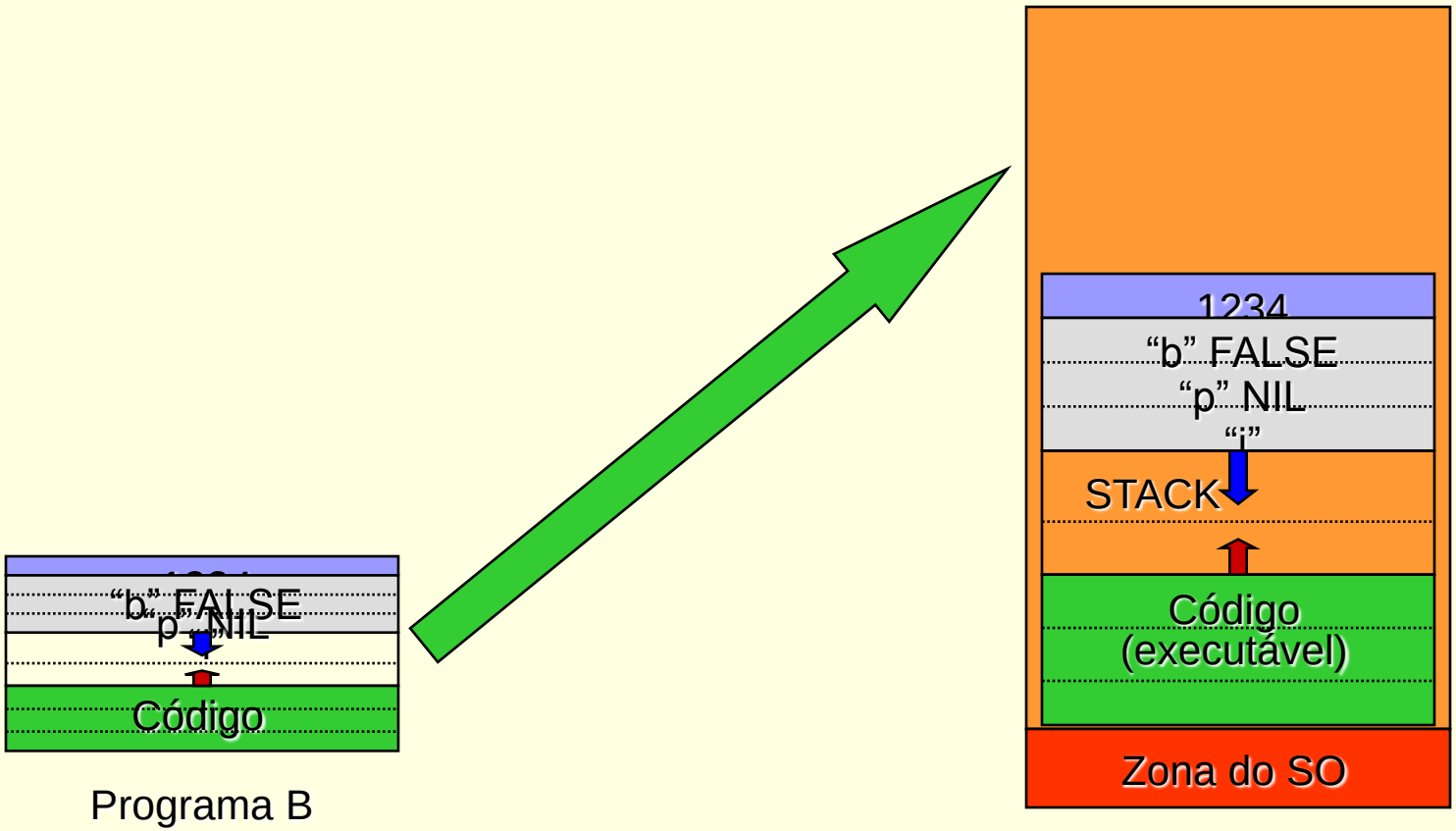
Partições dinâmicas (contíguas): I



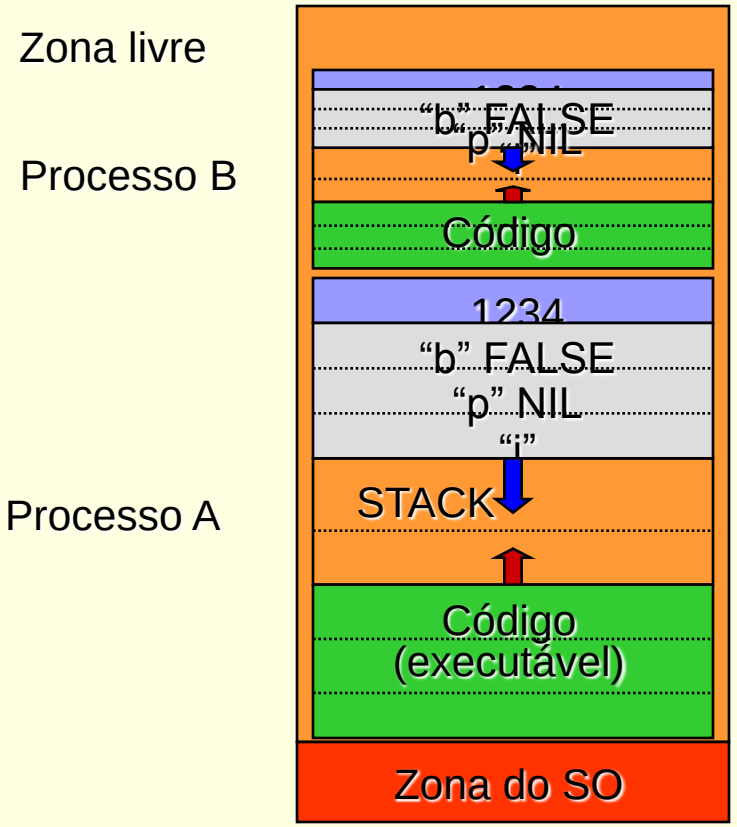
Partições dinâmicas (contíguas): II



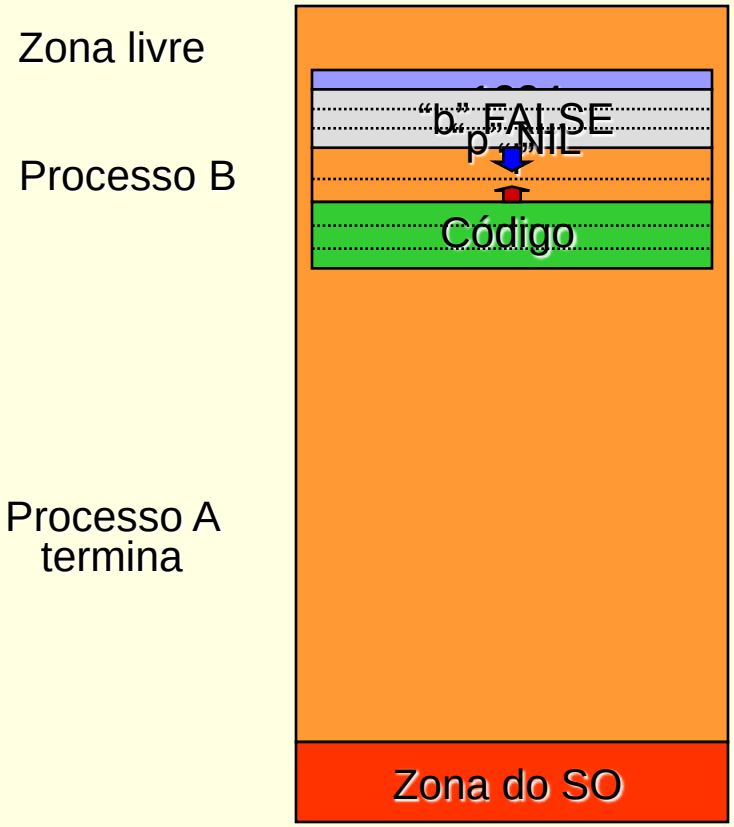
Partições dinâmicas (contíguas): III

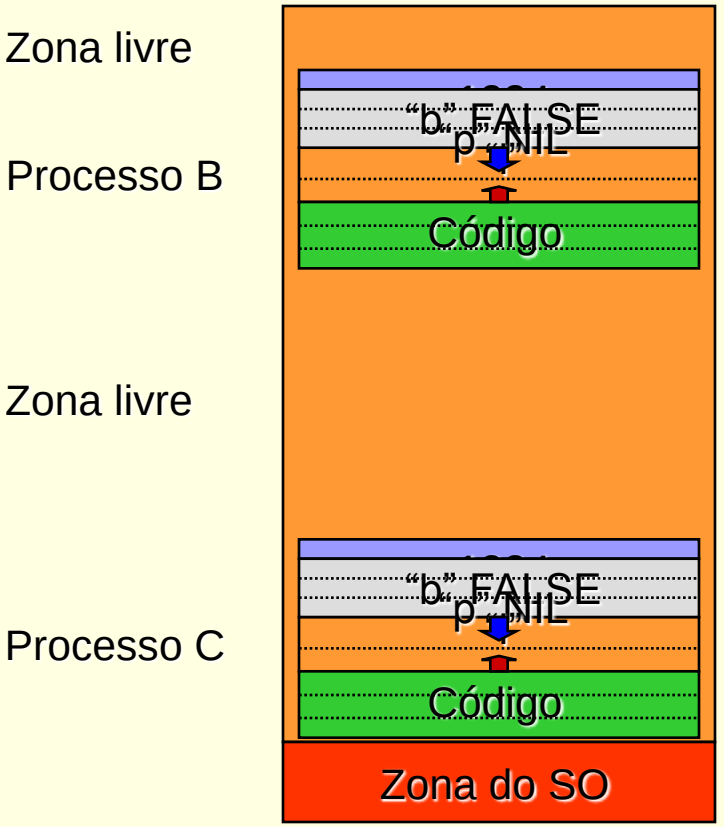


Partições dinâmicas (contíguas): IV

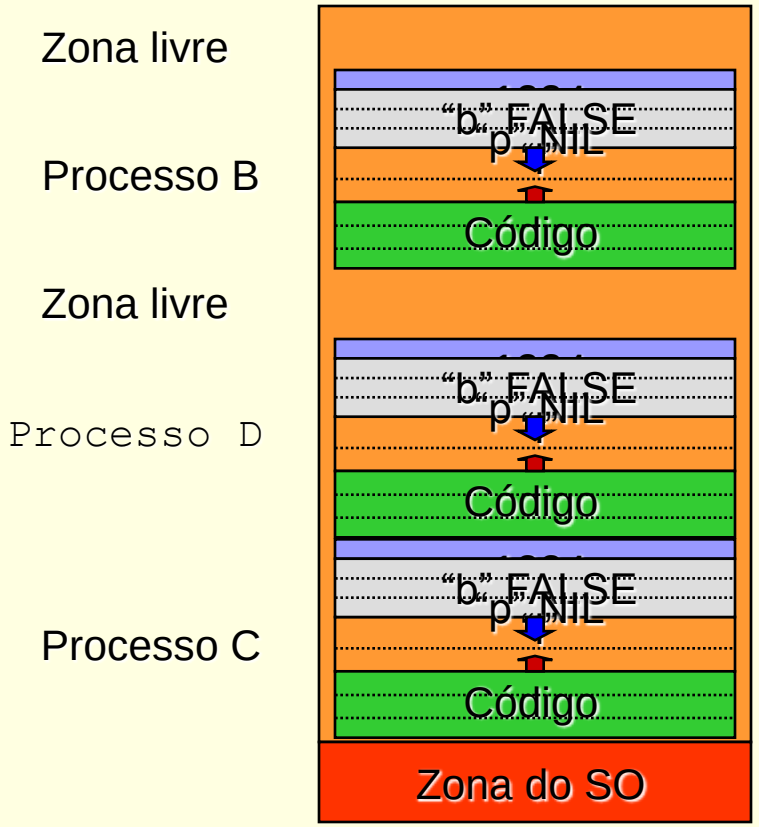


Partições dinâmicas (contíguas): V, VI





Partições dinâmicas (contíguas): VII



GM por Partições dinâmicas contíguas

- Benefícios:
  - Multiprogramação – vários processos em memória, podem ser “simultaneamente” executados pelo CPU.
- Estratégia de Colocação:
  - **Best Fit**: programa colocado na região livre “mais ajustada” à sua dimensão.
  - **Worst Fit**: programa colocado na maior região livre!
  - **First Fit**: programa colocado na primeira região que se sabe livre e de dimensão suficiente!

GM por Partições dinâmicas contíguas

- Desvantagens:
  - Nada a fazer quando exe não cabe.
  - Fragmentação externa – “fragmentos” livres de memória que ficam entre regiões ocupadas e não têm dimensão suficiente para albergar programas que se querem correr. (este é um argumento em favor do Worst Fit, que tende a deixar fragmentos grandes)
  - Má GM se a fragmentação for excessiva.
- Suporte hardware necessário:
  - UR com RB e RL.
  - Carregamento da UR é instrução privilegiada.

GM por Partições dinâmicas contíguas

- Solução: Operações de compactação reduzem o problema da fragmentação externa.
  - Juntar a memória livre num bloco único.
  - Operação feita em tempo de execução; só possível com recolocação dos programas (registos base e limite).
  - Exige muitas operações de I/O e provoca a paragem temporária de todos os programas envolvidos (pode haver *swap out/in* associado: colocar uma cópia da zona de memória em disco – swap out; libertar, compactar, e trazer de novo de disco para memória – swap in).

GM por Partições dinâmicas contíguas

- Estruturas de dados para a GM:
  - Lista ligada (?) para gerir as regiões livres, ordenada por dimensão da região (?)
  - Tabela (?) com uma linha por processo activo para manter: indicação da dimensão da região; RB e RL do processo;...
  - Que tal lembrar-se de alguns algoritmos que estudou em AED?
- Algoritmos para a GM:
  - Estratégia de colocação First, Best ou Worst fit
  - Best fit geralmente não usada: tende a criar numerosos pequenos fragmentos.