

Análise detalhada do exemplo `call_swap`



```
void call_swap()
{
  int zip1 = 15213;
  int zip2 = 91125;
  (...)
  swap(&zip1, &zip2);
  (...)
}
```

```
void swap(int *xp, int *yp)
{
  int t0 = *xp;
  int t1 = *yp;
  *xp = t1;
  *yp = t0;
}
```

Passagem do controlo

`call_swap` invoca `swap`

Assembly IA-32: `call swap` permite passar o controlo a `swap` e regressar a `call_swap` no final de `swap` com `ret`

Passagem de informação

`call_swap`: endereço das variáveis locais `zip1` e `zip2` passado a `swap` (apontadores colocados na pilha antes da chamada)

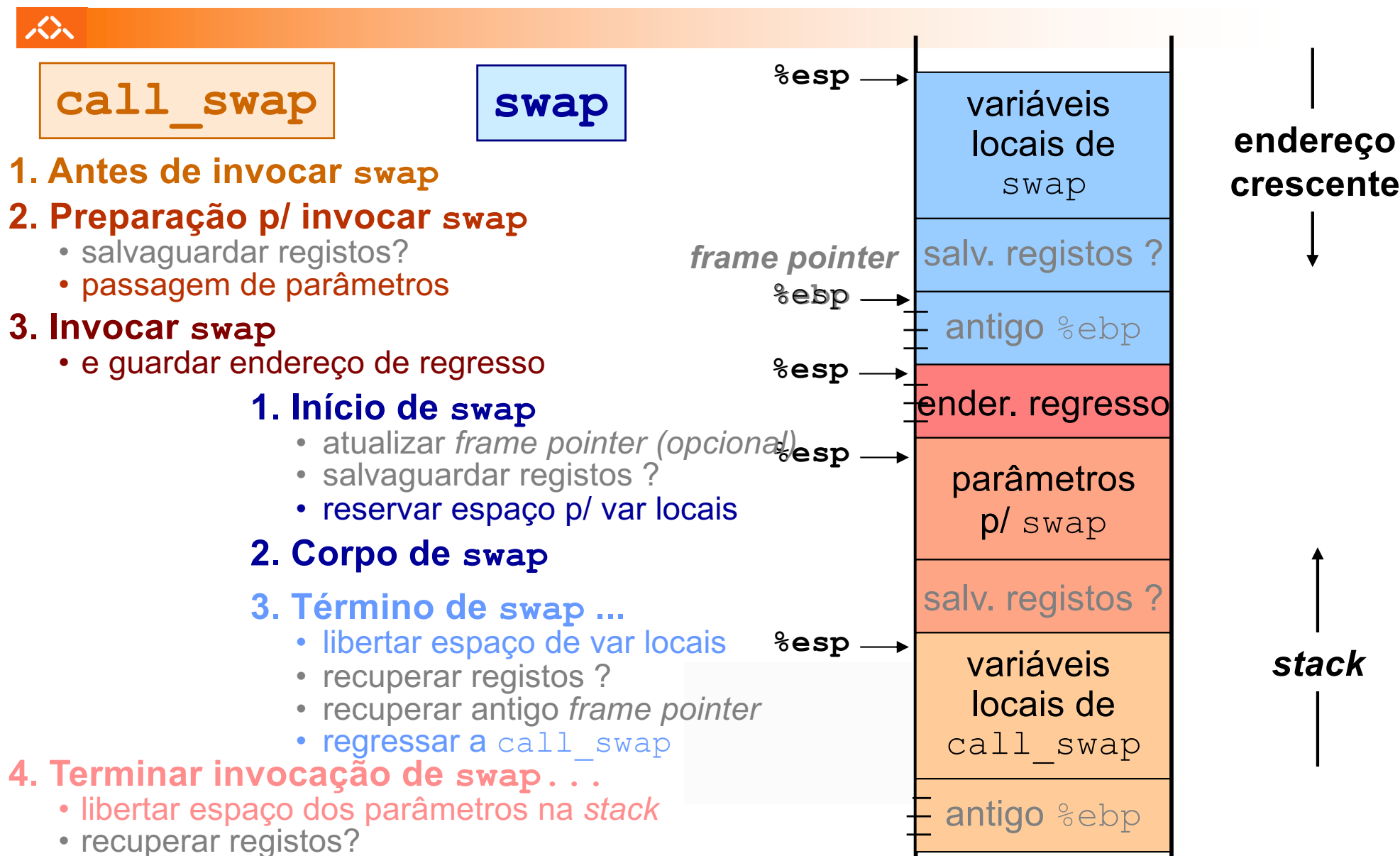
`swap`: acede aos parâmetros `xp` e `yp` colocados na pilha

Gestão da memória

`swap`: usa espaço na pilha para valores temporários (cópia de `%ebp`, etc)

variáveis locais `t0` e `t1` estão em registos (optimização!)

Visão global da criação da informação na stack (IA-32)



Evolução da stack, no IA-32 (1)

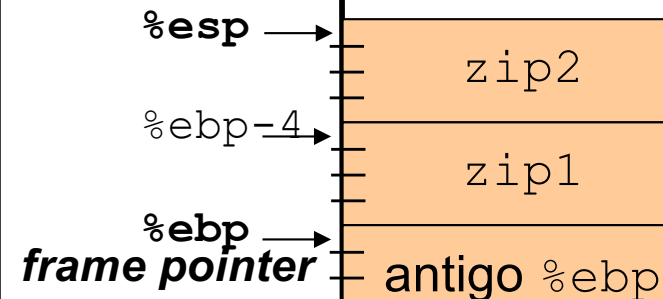


call_swap

1. Antes de invocar swap

```
void swap(int *xp, int *yp)
{
    int t0 = *xp;
    int t1 = *yp;
    *xp = t1;
    *yp = t0;
}
```

```
void call_swap()
{
    int zip1 = 15213;
    int zip2 = 91125;
    (...)
    swap(&zip1, &zip2);
    (...)
}
```



endereço
crescente

stack

Evolução da stack, no IA-32 (2)



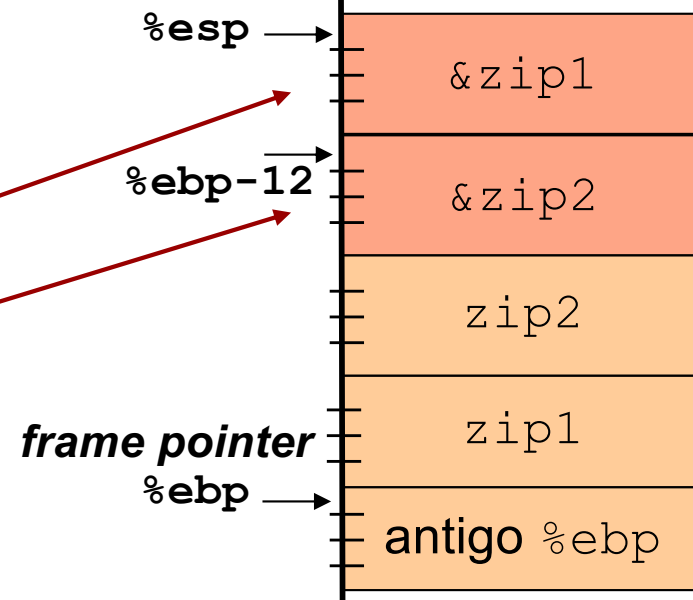
2. Preparação p/ invocar swap

- salvar registos?...**não**...
- passagem de parâmetros

call_swap

```
void swap(int *xp, int *yp)
{
    int t0 = *xp;
    int t1 = *yp;
    *xp = t1;
    *yp = t0;
}
```

```
void call_swap()
{
    int zip1 = 15213;
    int zip2 = 91125;
    (...)
    swap(&zip1, &zip2);
    (...)
}
```



endereço
crescente
↓

↑
stack

Evolução da stack, no IA-32 (3)

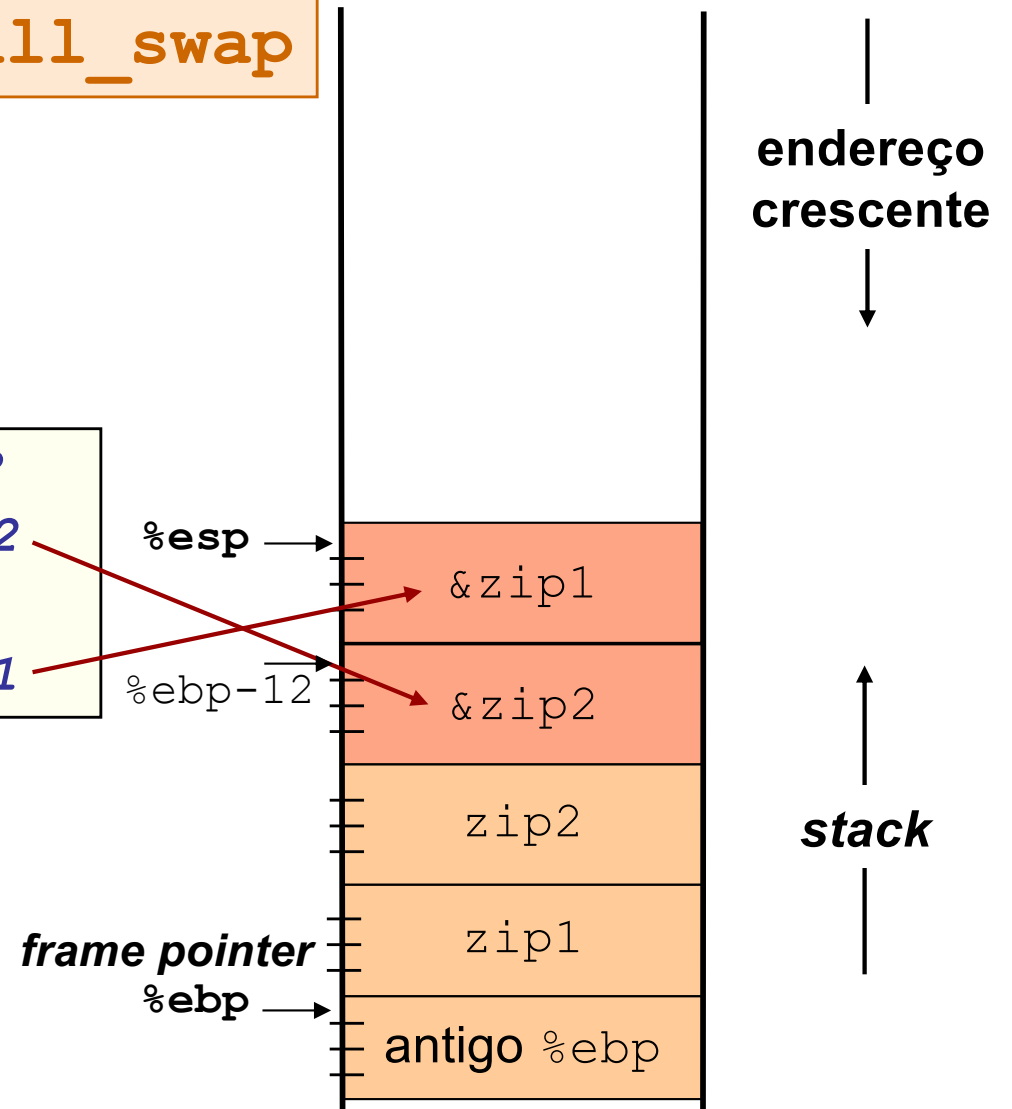


call_swap

2. Preparação p/ invocar swap

- salvar guardar registos?...**não**...
- passagem de parâmetros

```
leal  -8(%ebp), %eax  Calcula &zip2  
pushl %eax           Empilha &zip2  
leal  -4(%ebp), %eax  Calcula &zip1  
pushl %eax           Empilha &zip1
```



Evolução da stack, no IA-32 (4)



call_swap

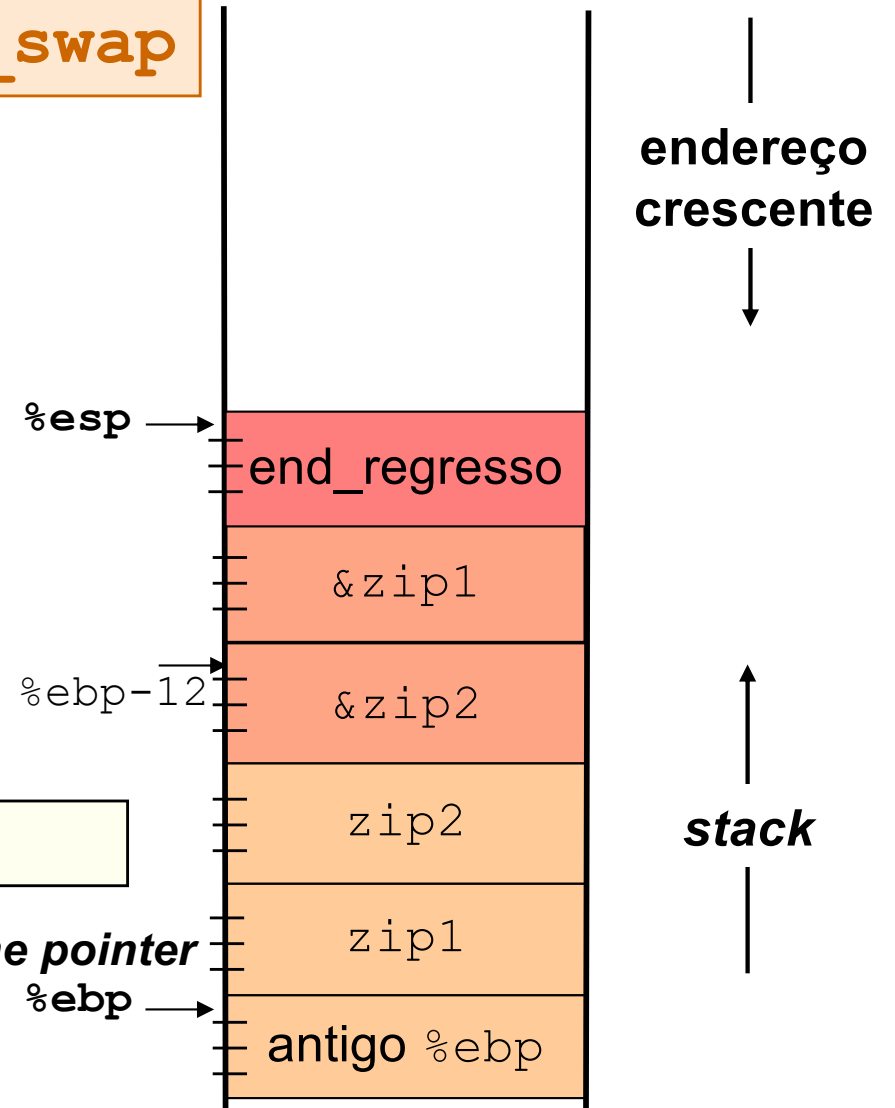
3. Invocar swap

- e guardar endereço de regresso

```
void call_swap()  
{  
  int zip1 = 15213;  
  int zip2 = 91125;  
  (...)  
  swap(&zip1, &zip2);  
  (...)  
}
```

call swap

Invoca função swap



Evolução da stack, no IA-32 (5)



1. Início de swap

- atualizar *frame pointer*
- salvar registos
- reservar espaço p/ locais...*não*...

```
void swap(int *xp, int *yp)
{
    int t0 = *xp;
    int t1 = *yp;
    *xp = t1;
    *yp = t0;
}
```

```
swap:
    pushl    %ebp
    movl     %esp, %ebp
    pushl    %ebx
```

Salv guarda antigo %ebp
Faz %ebp frame pointer
Salv guarda %ebx

swap

frame pointer

%esp →

%ebp →

antigo %ebx

antigo %ebp

end_regresso

*xp

*yp

zip2

zip1

antigo %ebp

endereço
crescente

↑
stack

antigo %ebp

Evolução da stack, no IA-32 (6)

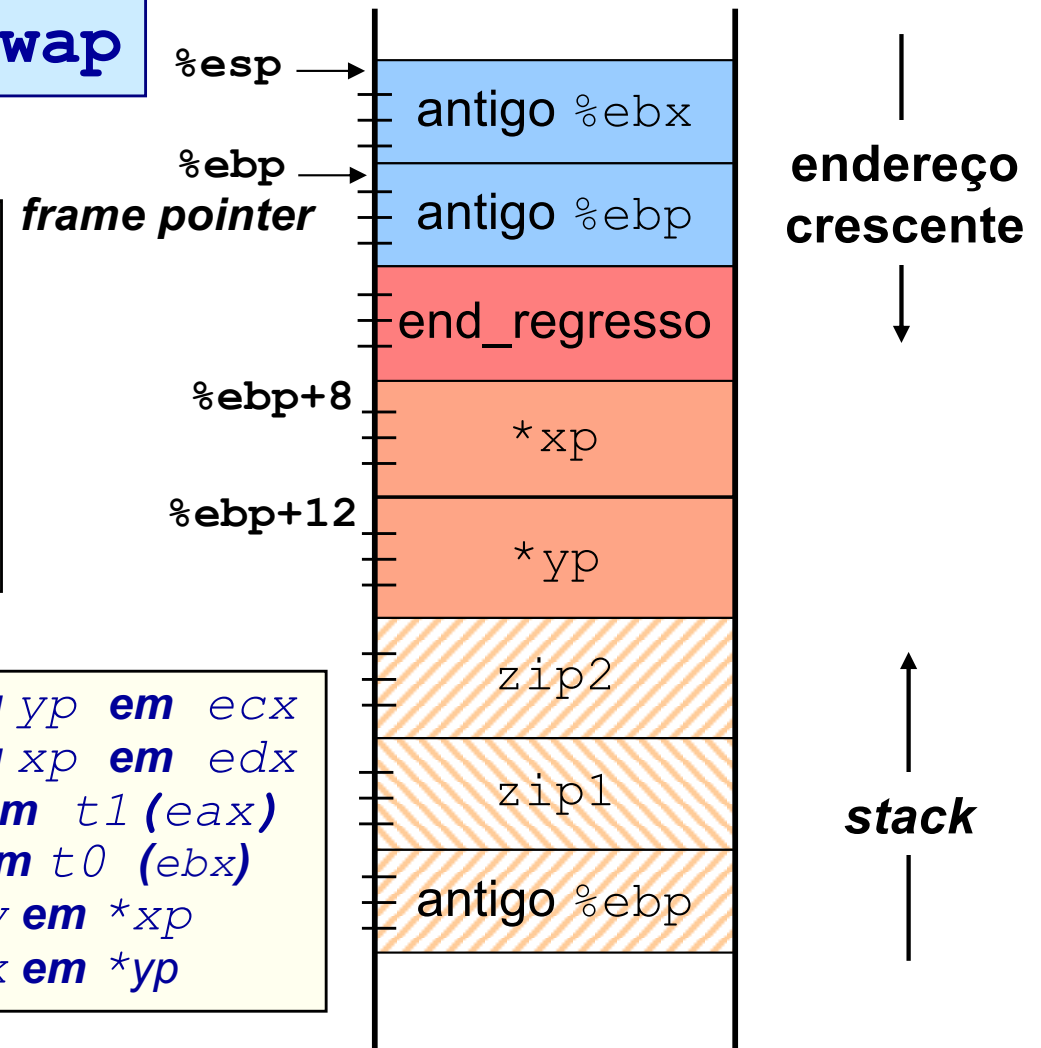


2. Corpo de swap

```
void swap(int *xp, int *yp)
{
    int t0 = *xp;
    int t1 = *yp;
    *xp = t1;
    *yp = t0;
}
```

<code>movl 12(%ebp), %ecx</code>	Carrega arg yp em ecx
<code>movl 8(%ebp), %edx</code>	Carrega arg xp em edx
<code>movl (%ecx), %eax</code>	Coloca y em t1 (eax)
<code>movl (%edx), %ebx</code>	Coloca x em t0 (ebx)
<code>movl %eax, (%edx)</code>	Armazena y em *xp
<code>movl %ebx, (%ecx)</code>	Armazena x em *yp

swap



Evolução da stack, no IA-32 (7)



3. Término de swap ...

- libertar espaço de var locais...**não**...
- recuperar registos
- recuperar antigo *frame pointer*
- regressar a `call_swap`

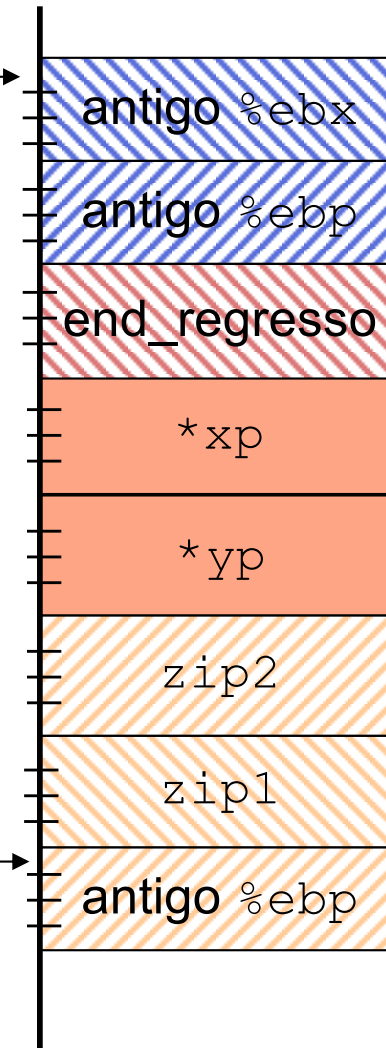
```
void swap(int *xp, int *yp)
{
    (...)
}
```

```
popl %ebx          Recupera %ebx
movl %ebp, %esp    Recupera %esp
popl %ebp          Recupera %ebp
ou
leave              Recupera %esp, %ebp
ret                Regressa à f. chamadora
```

swap

%esp →

%ebp →
frame
pointer



endereço
crescente

↑
stack

Evolução da stack, no IA-32 (8)



call_swap

4. Terminar invocação de swap...

- libertar espaço de parâmetros na *stack*...
- recuperar registos?...**não**...

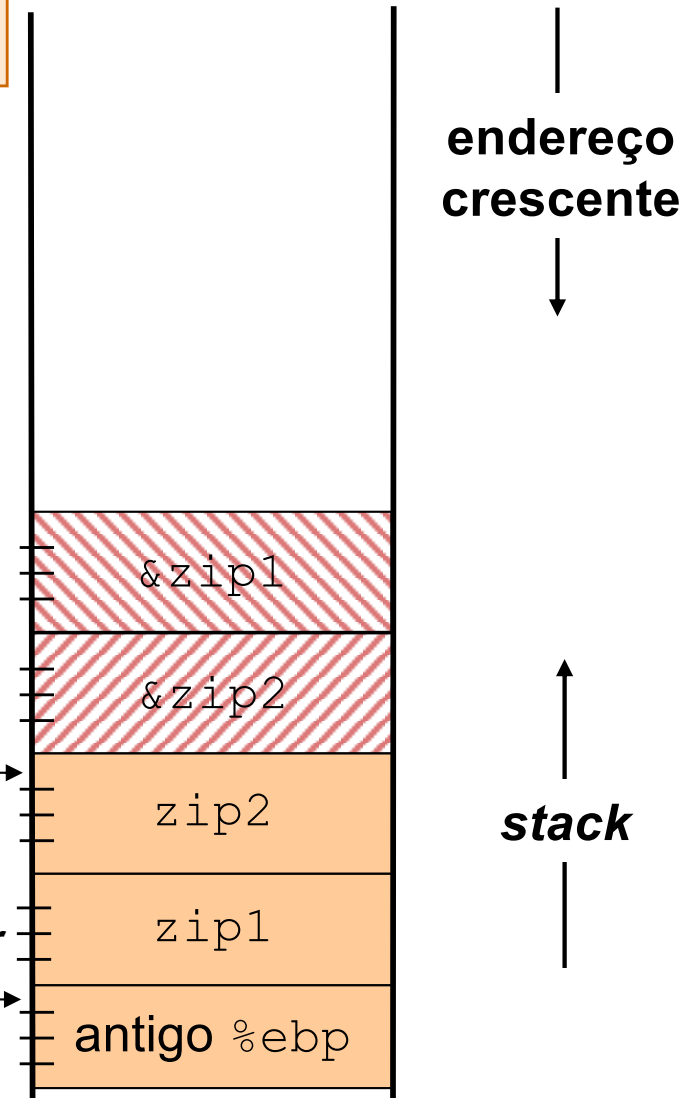
```
void call_swap()  
{  
  int zip1 = 15213;  
  int zip2 = 91125;  
  (...)  
  swap(&zip1, &zip2);  
  (...)  
}
```

```
addl $8, %esp
```

Atualiza stack pointer

frame pointer

%ebp



Utilização de registos em funções: regras seguidas pelos compiladores para IA-32



Utilização dos registos (de inteiros)

–Três do tipo *caller-save*

%eax, %edx, %ecx

- *save/restore*: função chamadora

–Três do tipo *callee-save*

%ebx, %esi, %edi

- *save/restore*: função chamada

–Dois apontadores (para a *stack*)

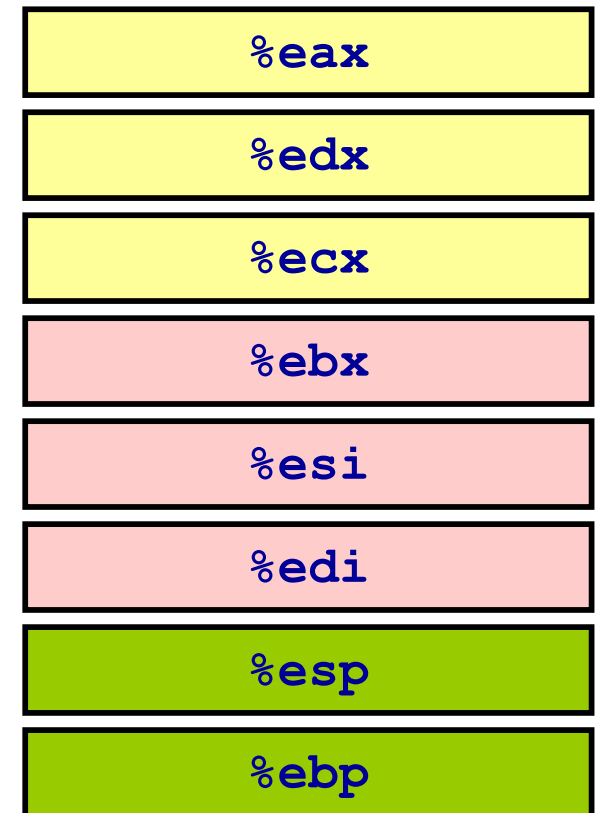
%esp, %ebp

- topo da *stack*, base/referência na *stack*

Caller-Save

Callee-Save

Pointers



Nota: a utilização do registo %ebp (*frame-pointer*) como base da estrutura local na pilha é opcional e simplifica o acesso à informação na pilha (variáveis locais)



Análise de exemplos

– revisão do exemplo swap

- análise das fases: arranque/inicialização, corpo, término
- análise dos contextos (IA-32)
- evolução dos contextos na *stack* (IA-32)

– aninhamento e recursividade

- evolução dos contextos na *stack*

Exemplo de cadeia de invocações no IA-32 (1)



Estrutura do código

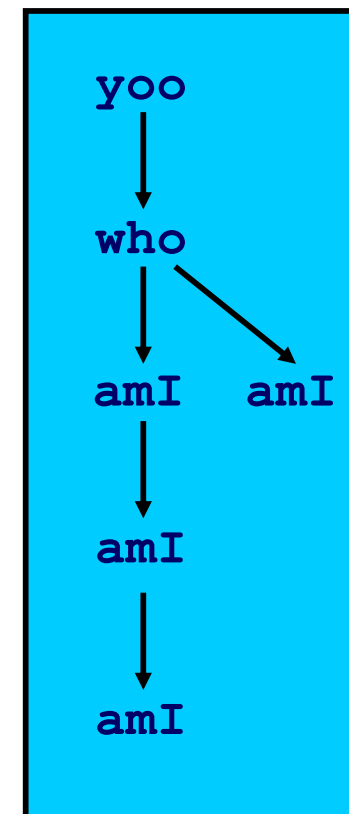
```
yoo (...)  
{  
  .  
  .  
  who () ;  
  .  
  .  
}
```

```
who (...)  
{  
  . . .  
  amI () ;  
  . . .  
  amI () ;  
  . . .  
}
```

```
amI (...)  
{  
  .  
  .  
  amI () ;  
  .  
  .  
}
```

Função **amI** é recursiva

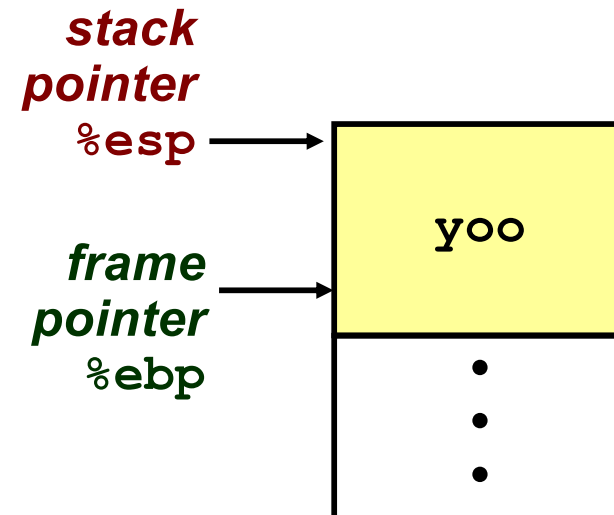
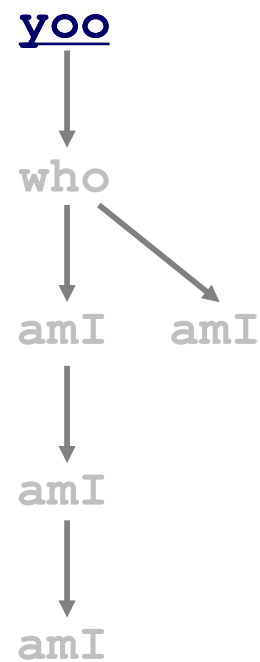
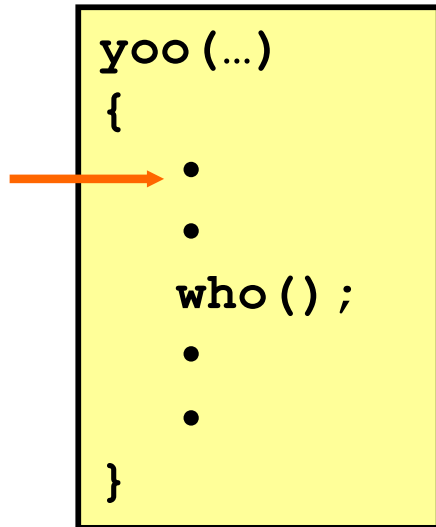
Cadeia de *call*



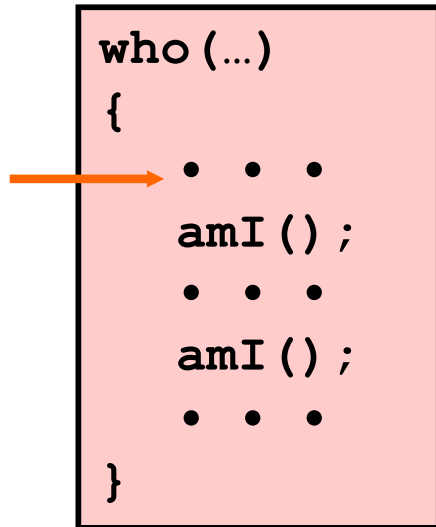
Exemplo de cadeia de invocações no IA-32 (2)



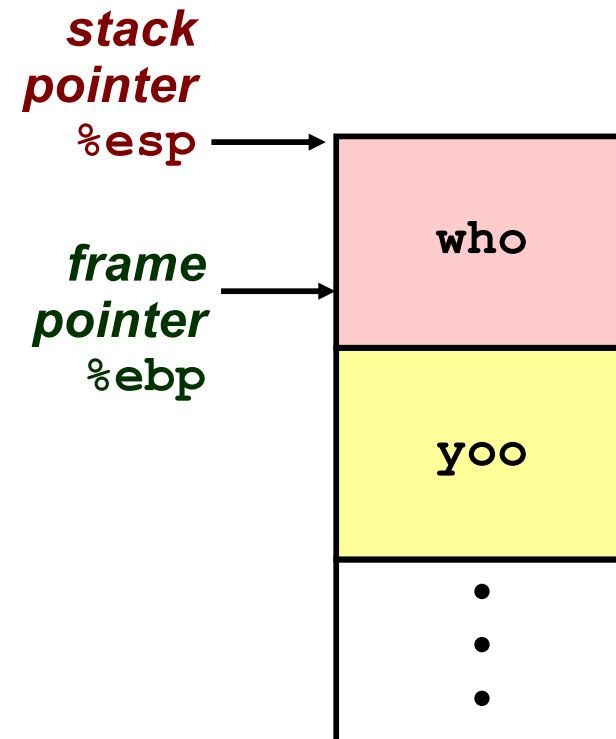
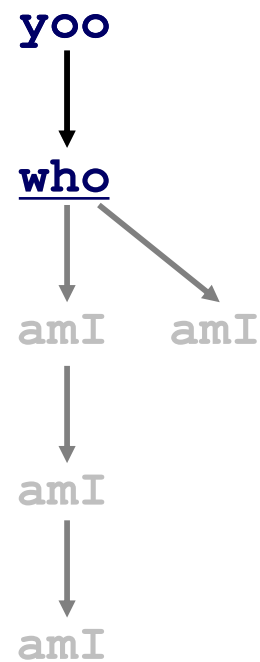
Cadeia de *call*



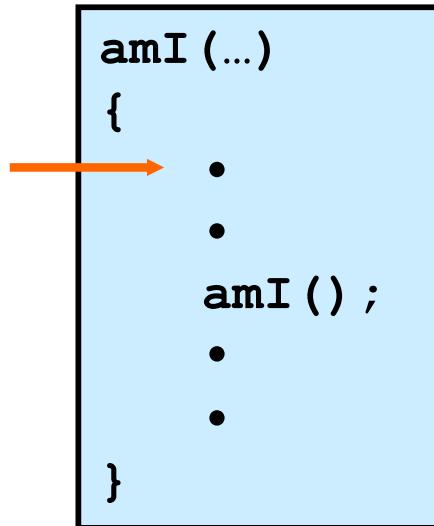
Exemplo de cadeia de invocações no IA-32 (3)



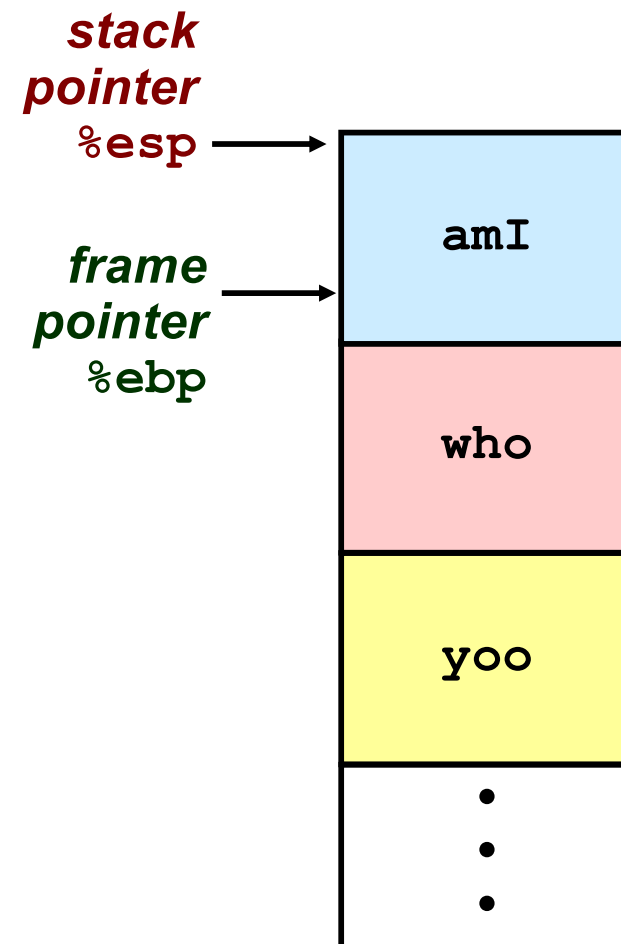
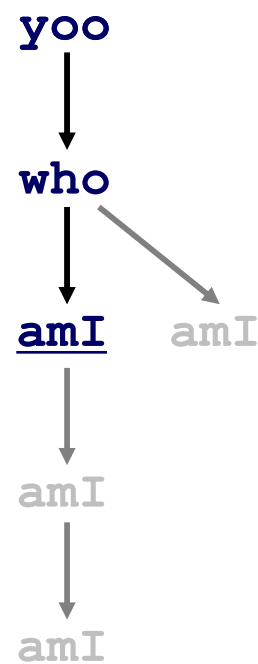
Cadeia de *call*



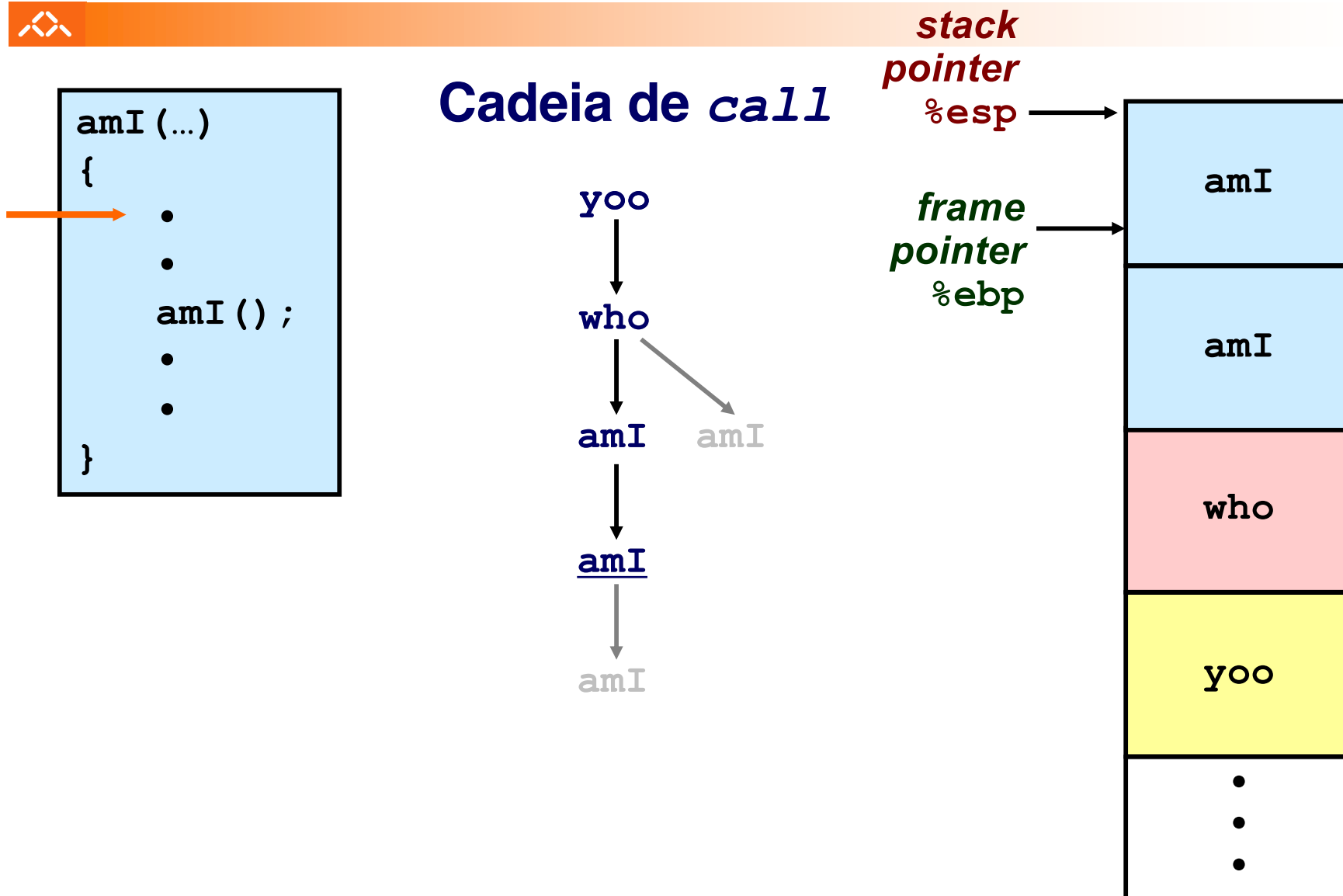
Exemplo de cadeia de invocações no IA-32 (4)



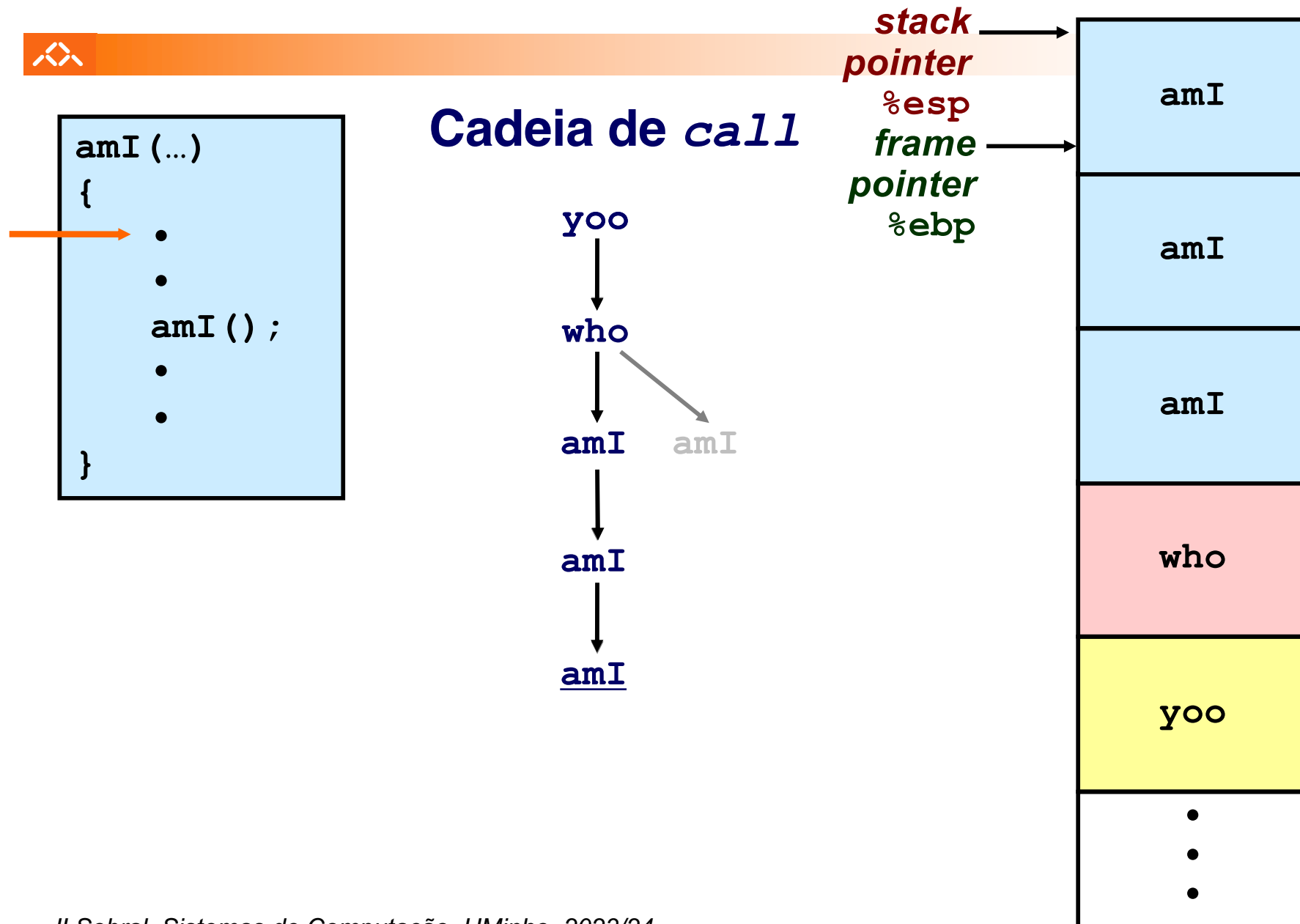
Cadeia de *call*



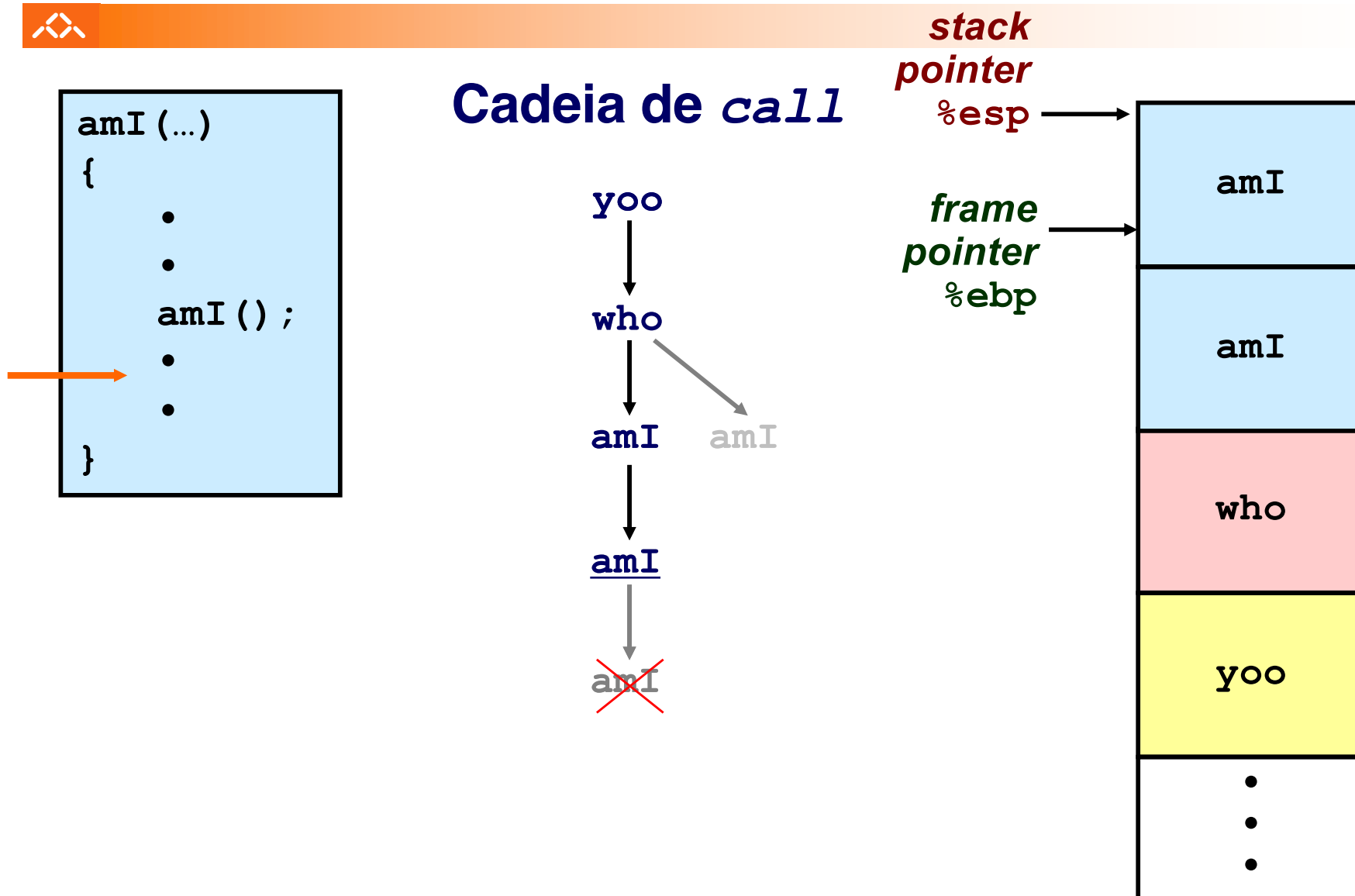
Exemplo de cadeia de invocações no IA-32 (5)



Exemplo de cadeia de invocações no IA-32 (6)



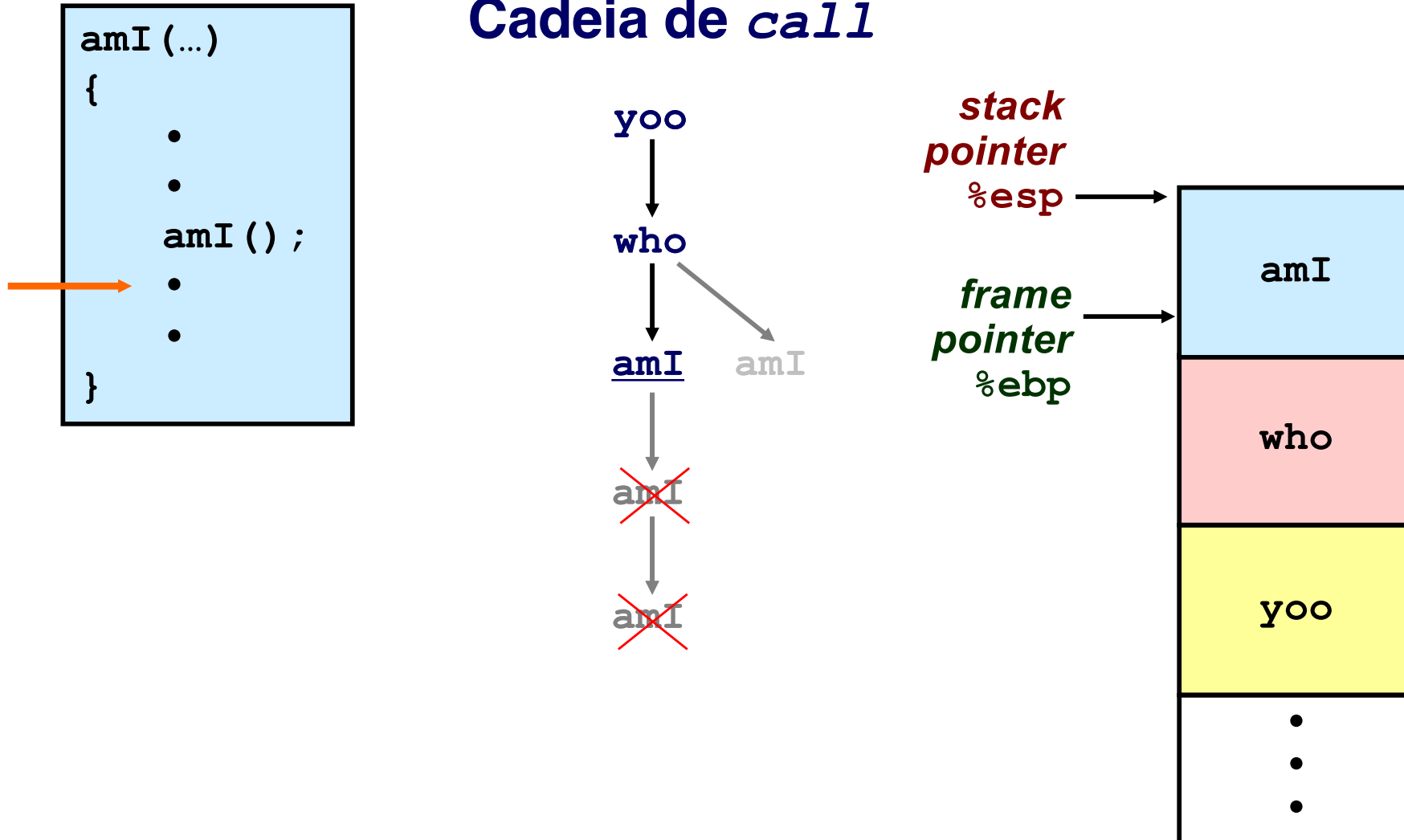
Exemplo de cadeia de invocações no IA-32 (7)



Exemplo de cadeia de invocações no IA-32 (8)



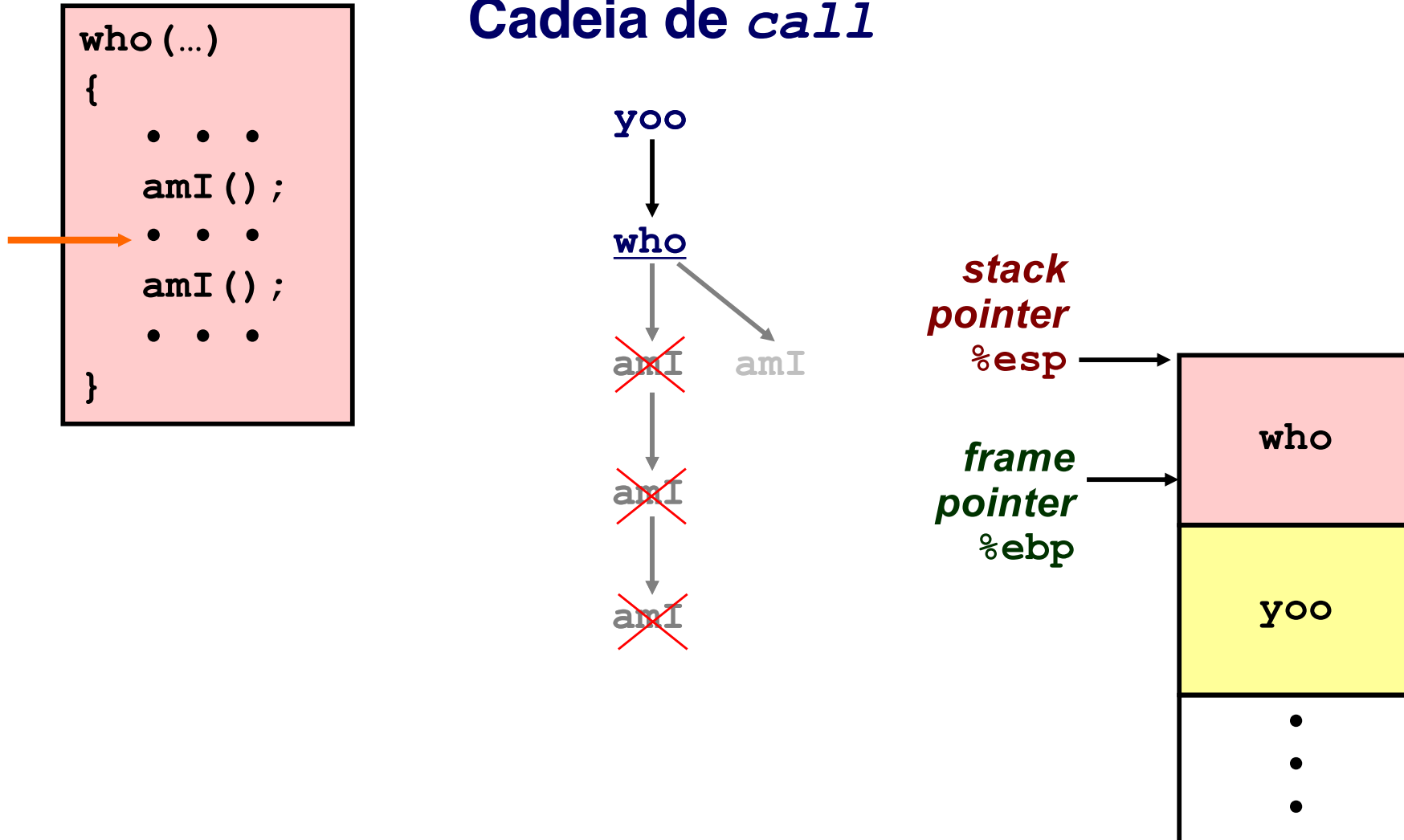
Cadeia de *call*



Exemplo de cadeia de invocações no IA-32 (9)



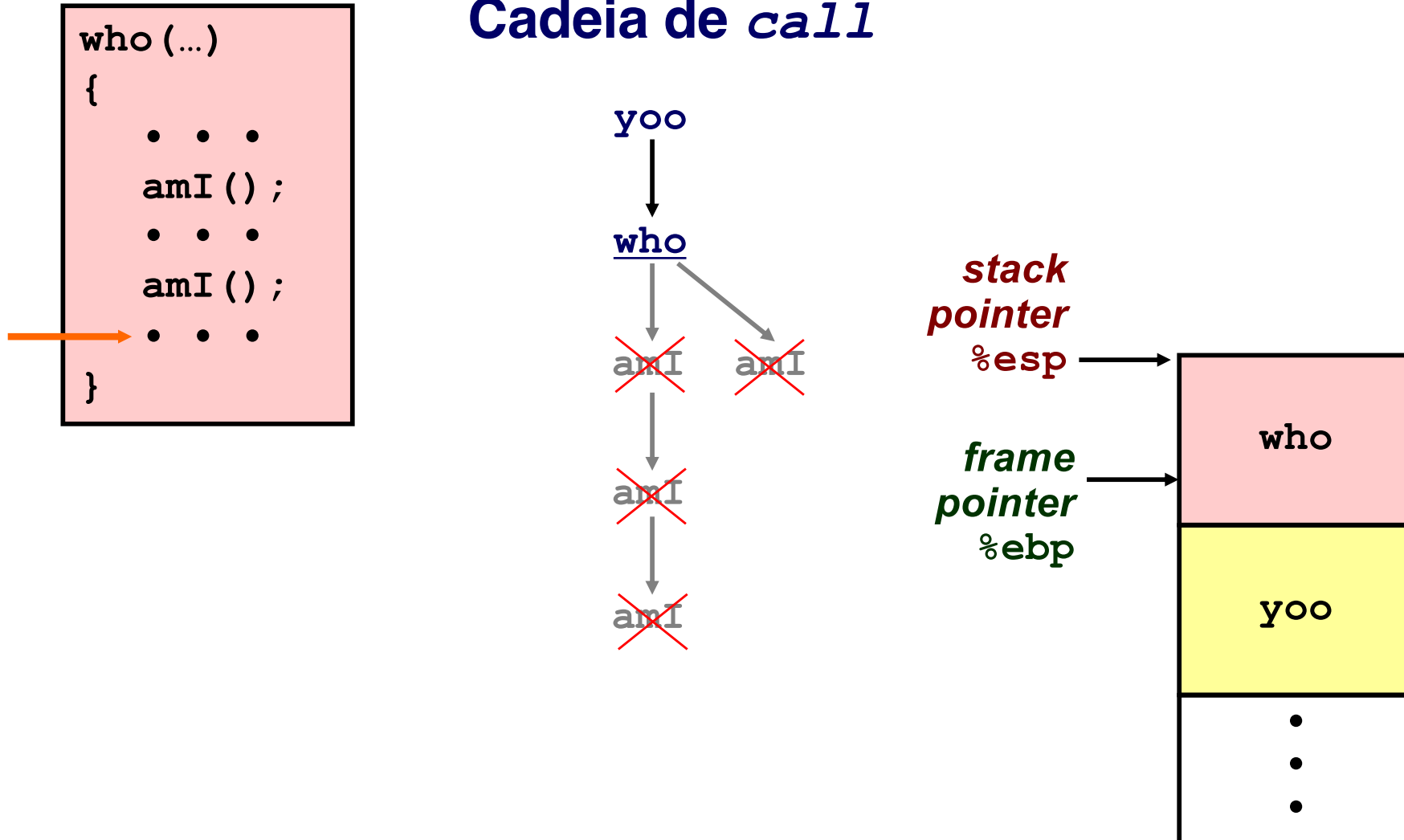
Cadeia de *call*



Exemplo de cadeia de invocações no IA-32 (11)



Cadeia de *call*

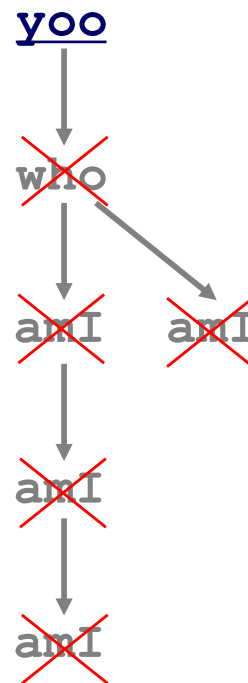


Exemplo de cadeia de invocações no IA-32 (12)



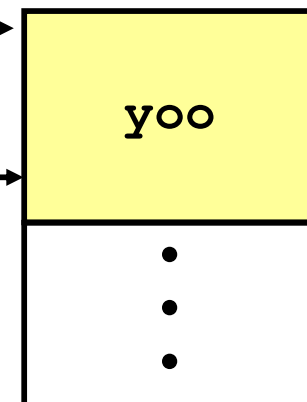
Cadeia de *call*

```
yoo (...)  
{  
  •  
  •  
  who ( ) ;  
  •  
  •  
}
```



**stack
pointer**
%esp

**frame
pointer**
%ebp

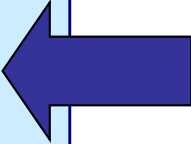


A série de Fibonacci no IA-32 (2)



função recursiva

```
int fib_rec (int n)
{
    int prev_val, val;
    if (n ≤ 2)
        return (1);
    prev_val = fib_rec (n-2);
    val = fib_rec (n-1);
    return (prev_val+val);
}
```



_fib_rec:

```
    pushl %ebp
    movl  %esp, %ebp

    subl  $12, %esp
    movl  %ebx, -8(%ebp)
    movl  %esi, -4(%ebp)

    movl  8(%ebp), %esi
```

Atualiza frame pointer

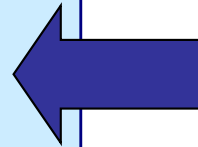
*Reserva espaço na stack para 3 int's
Salvaguarda os 2 reg's que vão ser usados;
de notar a forma de usar a stack...*

A série de Fibonacci no IA-32 (3)



função recursiva

```
int fib_rec (int n)
{
    int prev_val, val;
    if (n ≤ 2)
        return (1);
    prev_val = fib_rec (n-2);
    val = fib_rec (n-1);
    return (prev_val+val);
}
```



```
...
movl    %esi, -4(%ebp)
movl    8(%ebp), %esi
movl    $1, %eax
cmpl    $2, %esi
jle     L1
leal    -2(%esi), %eax
...
L1:
movl    -8(%ebp), %ebx
```

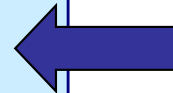
Coloca o argumento n em %esi
Coloca já o valor a devolver em %eax
Compara n:2
Se n ≤ 2, salta para o fim
Se não, ...

A série de Fibonacci no IA-32 (4)



função recursiva

```
int fib_rec (int n)
{
    int prev_val, val;
    if (n ≤ 2)
        return (1);
    prev_val = fib_rec (n-2);
    val = fib_rec (n-1);
    return (prev_val+val);
}
```



```
...
jle     L1
leal    -2(%esi), %eax
movl    %eax, (%esp)
call    _fib_rec
movl    %eax, %ebx
leal    -1(%esi), %eax
...
```

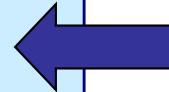
Se $n \leq 2$, salta para o fim
Se não, ... calcula $n-2$, e...
... coloca-o no topo da stack (argumento)
Invoca a função `fib_rec` e ...
... guarda o valor de `prev_val` em `%ebx`

A série de Fibonacci no IA-32 (5)



função recursiva

```
int fib_rec (int n)
{
    int prev_val, val;
    if (n ≤ 2)
        return (1);
    prev_val = fib_rec (n-2);
    val = fib_rec (n-1);
    return (prev_val+val);
}
```



```
...
movl    %eax, %ebx
leal    -1(%esi), %eax
movl    %eax, (%esp)
call    _fib_rec
leal    (%eax,%ebx), %eax
...
```

Calcula n-1, e...

... coloca-o no topo da stack (argumento)

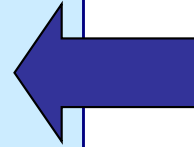
Chama de novo a função fib_rec

A série de Fibonacci no IA-32 (6)



função recursiva

```
int fib_rec (int n)
{
    int prev_val, val;
    if (n ≤ 2)
        return (1);
    prev_val = fib_rec (n-2);
    val = fib_rec (n-1);
    return (prev_val+val);
}
```



```
call    fib_rec
leal    (%eax,%ebx), %eax    Calcula e coloca em %eax o valor a devolver

L1:
movl    -8(%ebp), %ebx
movl    -4(%ebp), %esi       Recupera o valor dos 2 reg's usados
movl    %ebp, %esp          Atualiza o valor do stack pointer
popl    %ebp                Recupera o valor anterior do frame pointer
ret
```