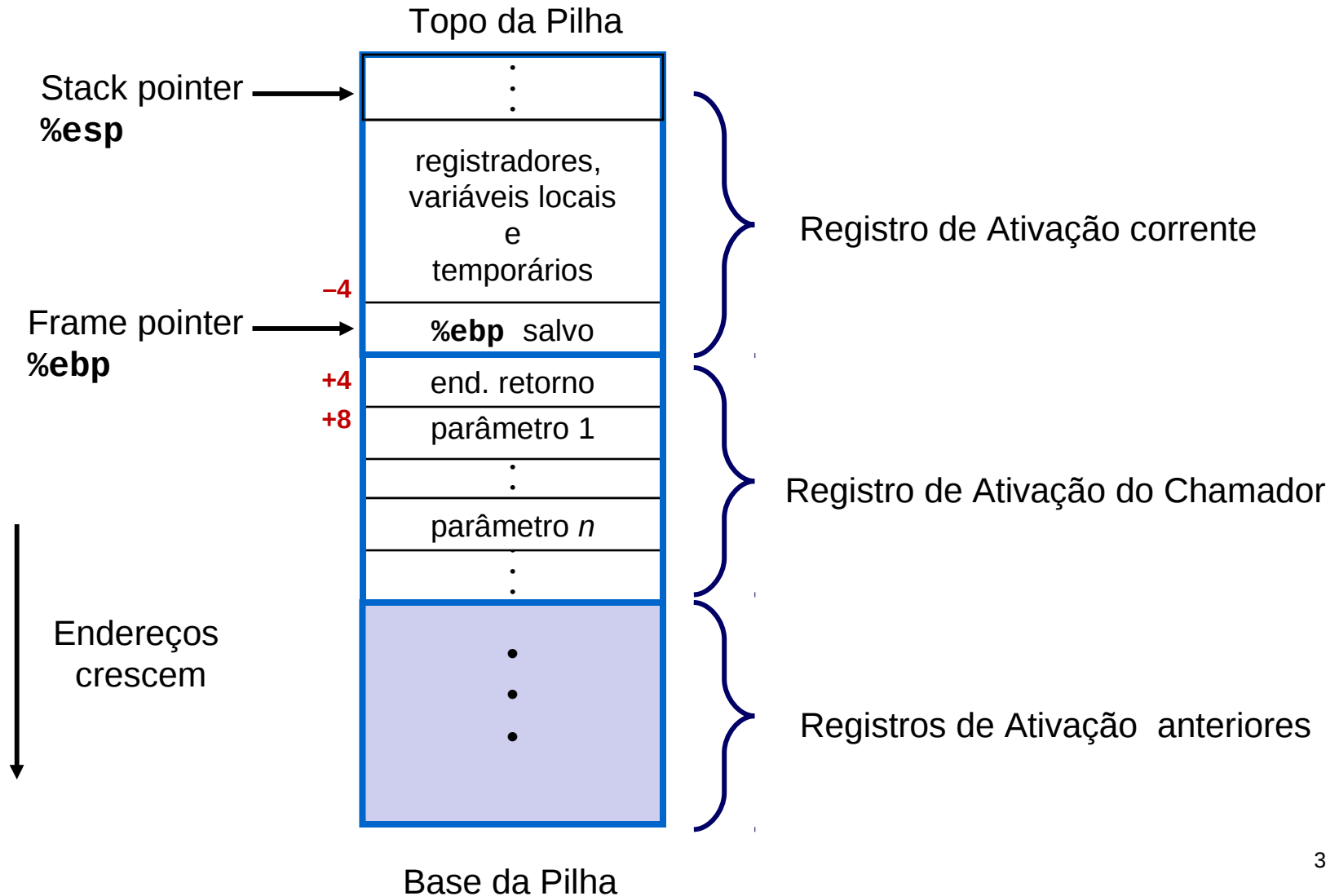


Procedimentos Registro de Ativação e Variáveis Locais

Registro de Ativação

- A implementação de procedimentos tem como base o uso de uma **pilha**
 - passagem de parâmetros, armazenamento do endereço de retorno, salvamento de registradores, armazenamento de **variáveis locais** e **valores temporários**
- Cada chamada de procedimento é associada a uma porção da pilha, que é liberada no retorno desse procedimento
 - Chamada de **registro de ativação** ou *stack frame*
 - o RA corrente (no topo da pilha) é delimitado por dois ponteiros:
 - %ebp: frame pointer → base do registro de ativação (fixa)
 - %esp: stack pointer → topo da pilha (dinâmico)

Estrutura do Registro de Ativação



Acesso ao Registro de Ativação

- Início de procedimento:

pushl %ebp → salva endereço do Registro de Ativação anterior

movl %esp, %ebp → base do novo Registro de Ativação

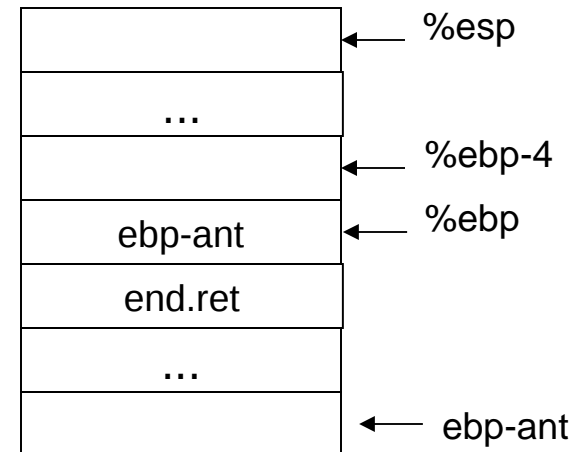
- Fim de procedimento:

movl %ebp, %esp

popl %ebp → restaura ebp-ant

ret

registro de ativação



liberação do registro de ativação

Variáveis Locais

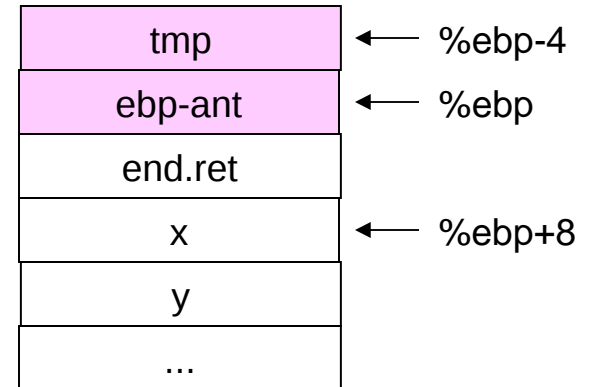
- Variáveis locais de um procedimento podem ser armazenadas na pilha (registro de ativação):
 - quando o número de registradores é insuficiente
 - armazenamento de *arrays* e estruturas
 - quando o operador ‘&’ é aplicado à variável (exige a geração de um endereço)
- A alocação de espaço é tipicamente feita subtraindo-se o espaço a ser alocado (número de bytes) de `%esp`:
 - **`subl $20, %esp` → alocação de 5 palavras...**
- O acesso é realizado com base no *frame pointer* (`%ebp`)
 - **`movl $0, -4(%ebp)`**

Exemplo de variável local

troca:

```
void troca (int *x, int *y){  
  int tmp;  
  
  tmp = *x;  
  
  *x = *y;  
  
  *y = tmp;  
}
```

```
pushl %ebp  
movl %esp, %ebp  
subl $4, %esp  
movl 8(%ebp), %eax  
movl (%eax), %edx  
movl %edx, -4(%ebp)  
movl 12(%ebp), %ecx  
movl (%ecx), %edx  
movl %edx, (%eax)  
movl -4(%ebp), %edx  
movl %edx, (%ecx)  
movl %ebp, %esp  
popl %ebp  
ret
```

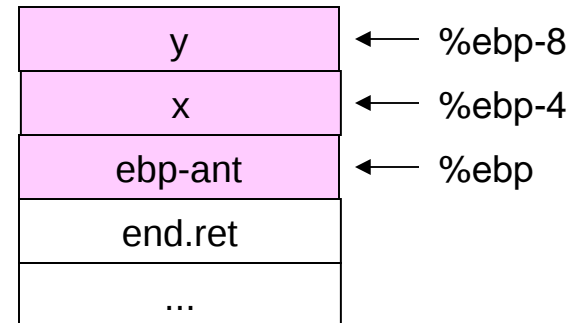


Outro exemplo

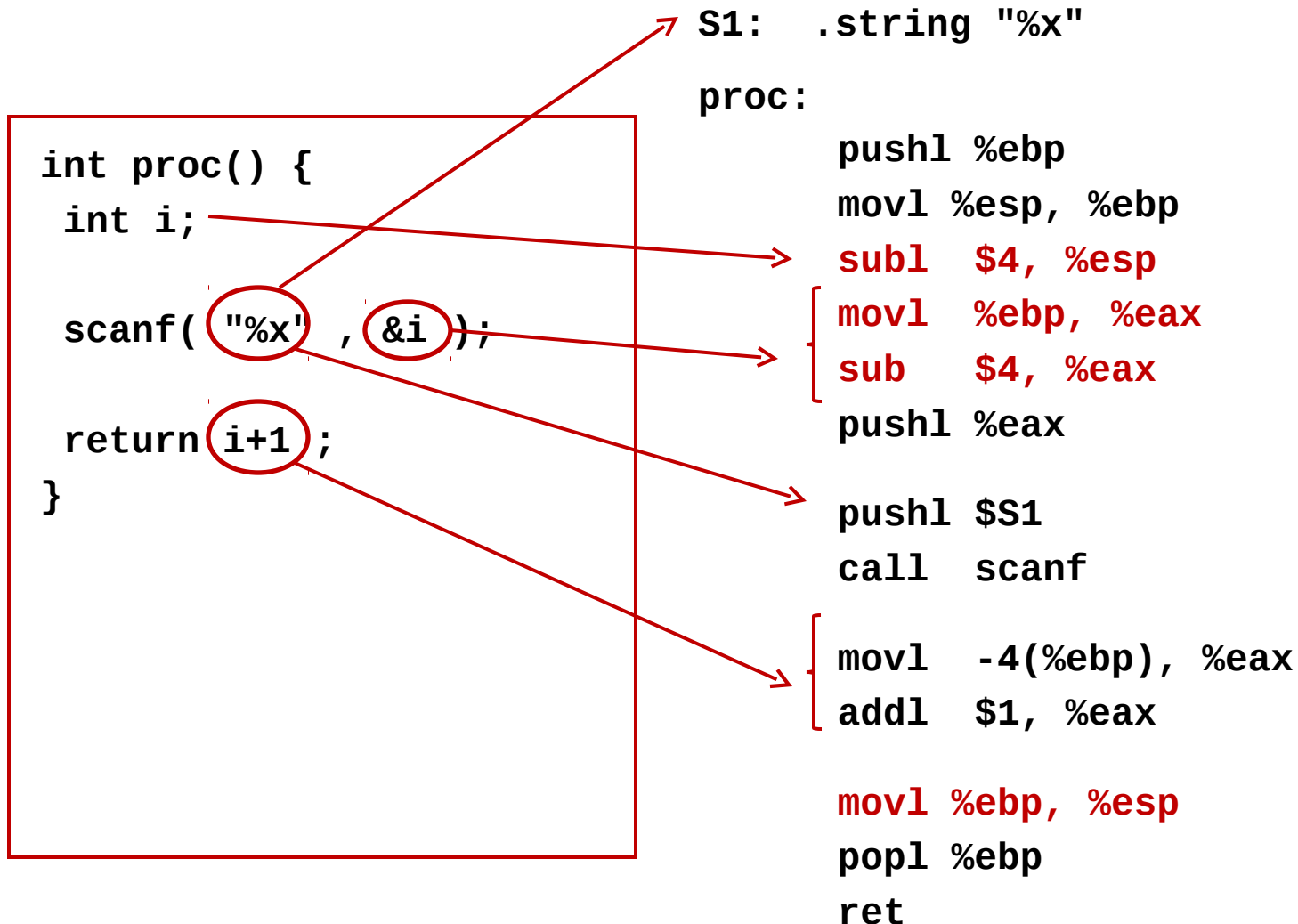
caller:

```
int caller() {  
    int x = 534;  
    int y = 1057;  
  
    ...  
  
    return f(&x, y);  
}
```

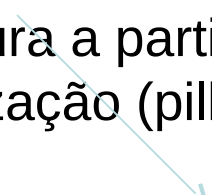
```
pushl %ebp  
movl %esp, %ebp  
subl $8, %esp  
movl $534, -4(%ebp)  
movl $1057, -8(%ebp)  
  
...  
[ movl -8(%ebp), %eax  
  pushl %eax  
]  
[ movl %ebp, %eax  
  subl $4, %eax  
  pushl %eax  
]  
call f  
movl %ebp, %esp  
popl %ebp  
ret
```



Mais um exemplo



Alocação de Variáveis Locais

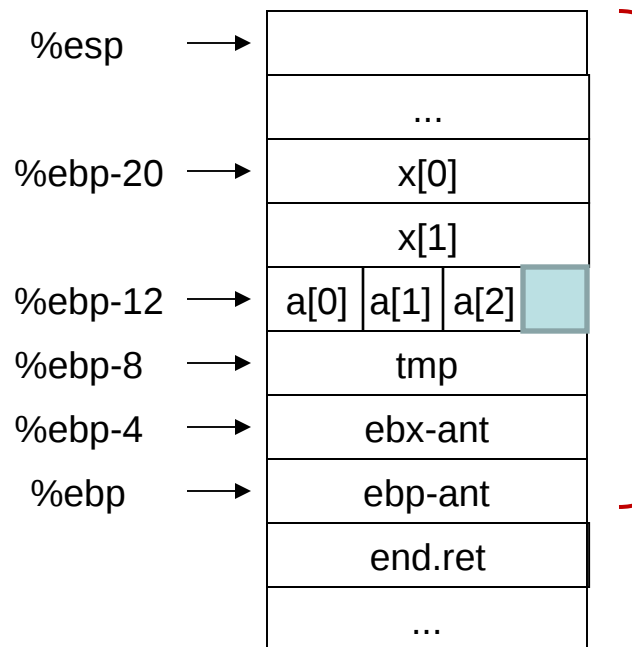
- Não existe convenção para a ordenação de variáveis na pilha!
 - apenas a própria função obtém os endereços dessas variáveis
 - Devem ser respeitadas as convenções de alinhamento para *arrays* e estruturas!
 - o acesso aos elementos de um *array* a partir de seu endereço inicial deve depender de sua localização (pilha ou memória global)
 - o acesso aos campos de uma estrutura a partir do seu endereço inicial deve depender de sua localização (pilha ou memória global)
- 

Alocação de arrays na pilha

```
void f() {  
    int tmp;  
    char a[3];  
    int x[2];  
    ...  
}
```

16 bytes: 4 de tmp, 4 de a, 8 de x

```
pushl %ebp  
movl  %esp, %ebp  
subl  $20, %esp  
movl  %ebx, -4(%ebp)  
...  
mov  %ebp, %ecx  
sub  $12, %ecx  
...  
mov  %ebp, %ecx  
sub  $20, %ecx  
...  
movl -4(%ebp), %ebx  
movl %ebp, %esp  
popl %ebp  
ret
```



registro de ativação de "f"