Лекция 4

18 февраля

```
...
SECTION .text
GLOBAL CMAIN
CMAIN:
   MOV  EAX, DWORD [a]     ; (1)
   TEST EAX, EAX           ; (2)
   JE   .1                 ; (3)
   MOV  ECX, DWORD [b]     ; (4)
   TEST ECX, ECX           ; (5)
   JE   .1                 ; (6)
   CDQ                     ; (7)
   IDIV ECX                ; (8)
   SUB  DWORD [a], EDX     ; (9)
 .1:
   XOR  EAX, EAX           ; (10)
   RET                     ; (11)
```

Просмотр содержимого исполняемого файла

```
-bash-2.05b$ ./build_asm.sh backward.asm
-bash-2.05b$ objdump -d -M intel backward
```

```
080483e0 <main>:
 80483e0:       a1 fc 94 04 08          mov    eax,ds:0x80494fc
 80483e5:       85 c0                   test   eax,eax
 80483e7:       74 13                   je     80483fc <main.1>
 80483e9:       8b 0d 00 95 04 08       mov    ecx,ds:0x8049500
 80483ef:       85 c9                   test   ecx,ecx
 80483f1:       74 09                   je     80483fc <main.1>
 80483f3:       99                      cdq
 80483f4:       f7 f9                   idiv   ecx
 80483f6:       29 15 fc 94 04 08       sub    ds:0x80494fc,edx

080483fc <main.1>:
 80483fc:       31 c0                   xor    eax,eax
 80483fe:       c3                      ret
 80483ff:       90                      nop
```

Задача

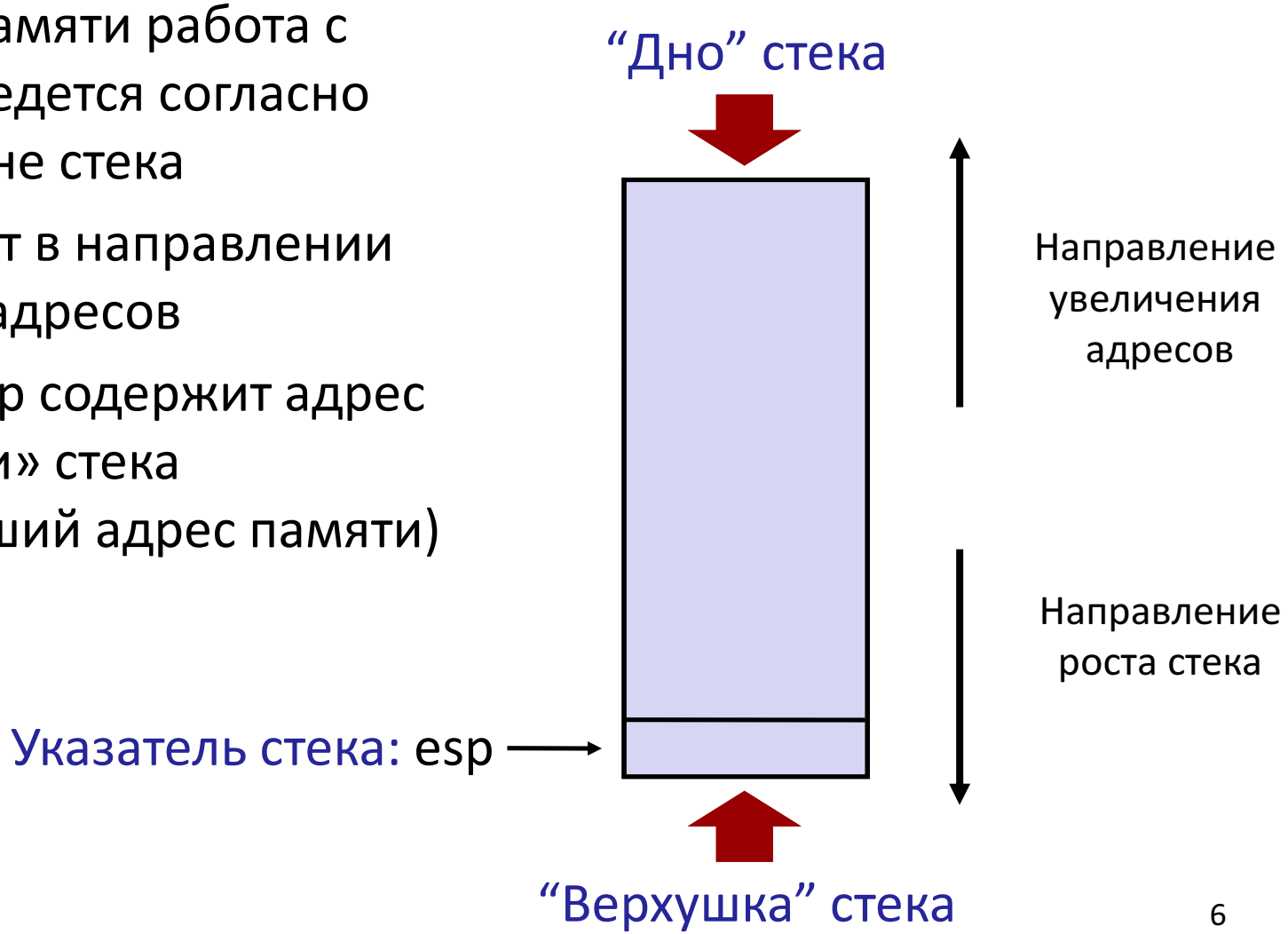
```
080483e0 <main>:
 80483e0:       a1 0c 95 04 08          mov    eax,ds:0x804950c
 80483e5:       8b 0d 14 95 04 08       mov    ecx,ds:0x8049514
 80483eb:       85 c9                   test   ecx,ecx
 80483ed:       75 08                   jne    80483f7 <main.1>
 80483ef:       03 05 10 95 04 08       add    eax,ds:0x8049510
 80483f5:       eb 0a                   jmp    8048401 <main.2>

080483f7 <main.1>:
 80483f7:       0f af 05 10 95 04 08    imul   eax,ds:0x8049510
 80483fe:       99                      cdq
 80483ff:       f7 f9                   idiv   ecx

08048401 <main.2>:
 8048401:       a3 18 95 04 08          mov    ds:0x8049518,eax
 8048406:       31 c0                   xor    eax,eax
 8048408:       c3                      ret
```

Организация вызова функций

- Вопросы
  - Передача управления и возвращение обратно
  - Вычисление значений фактических параметров и их размещение
  - Передача возвращаемого значения
  - Размещение автоматических локальных переменных
  - Порядок использование регистрового файла различными функциями
  - Какие именно машинные команды использовать для поддержки функций
- Ответы – Application Binary Interface (ABI)
  - Соглашение о вызовах (Calling Convention)

Аппаратный стек IA-32

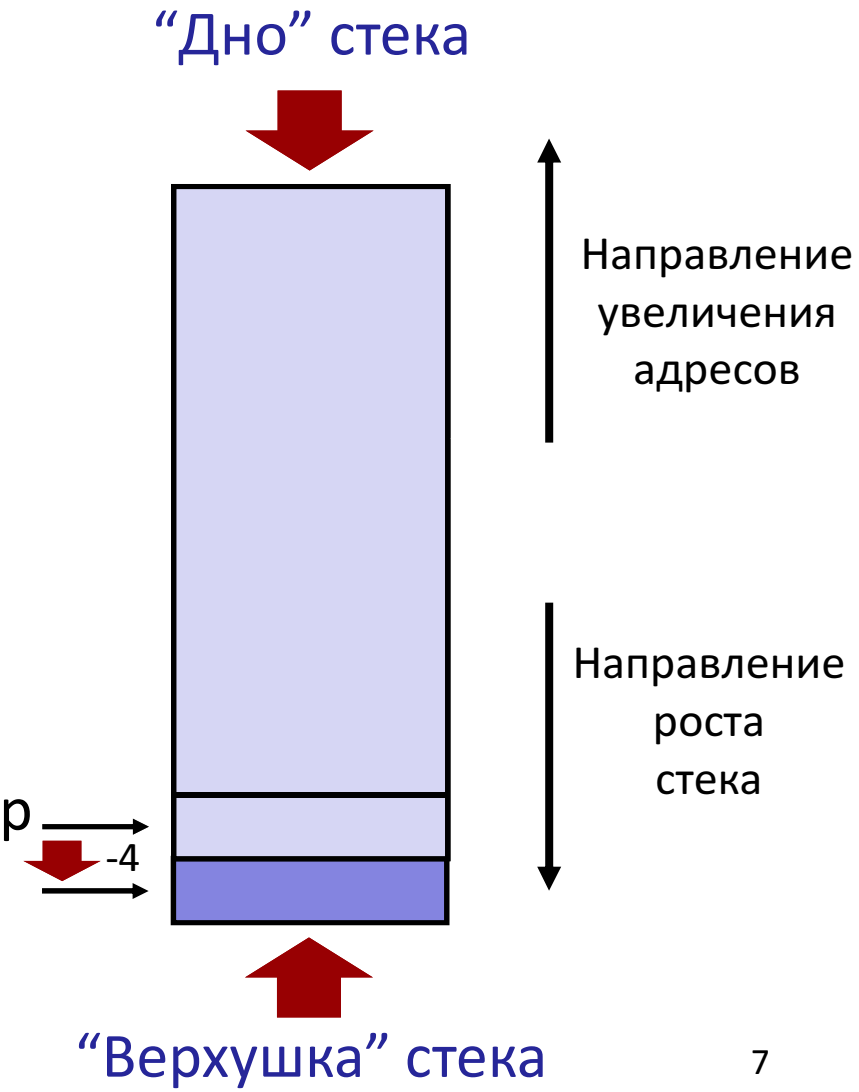
- Область памяти работа с которой ведется согласно дисциплине стека
- Стек растет в направлении меньших адресов
- Регистр esp содержит адрес «верхушки» стека (наименьший адрес памяти)



Загрузка данных в стек: Push

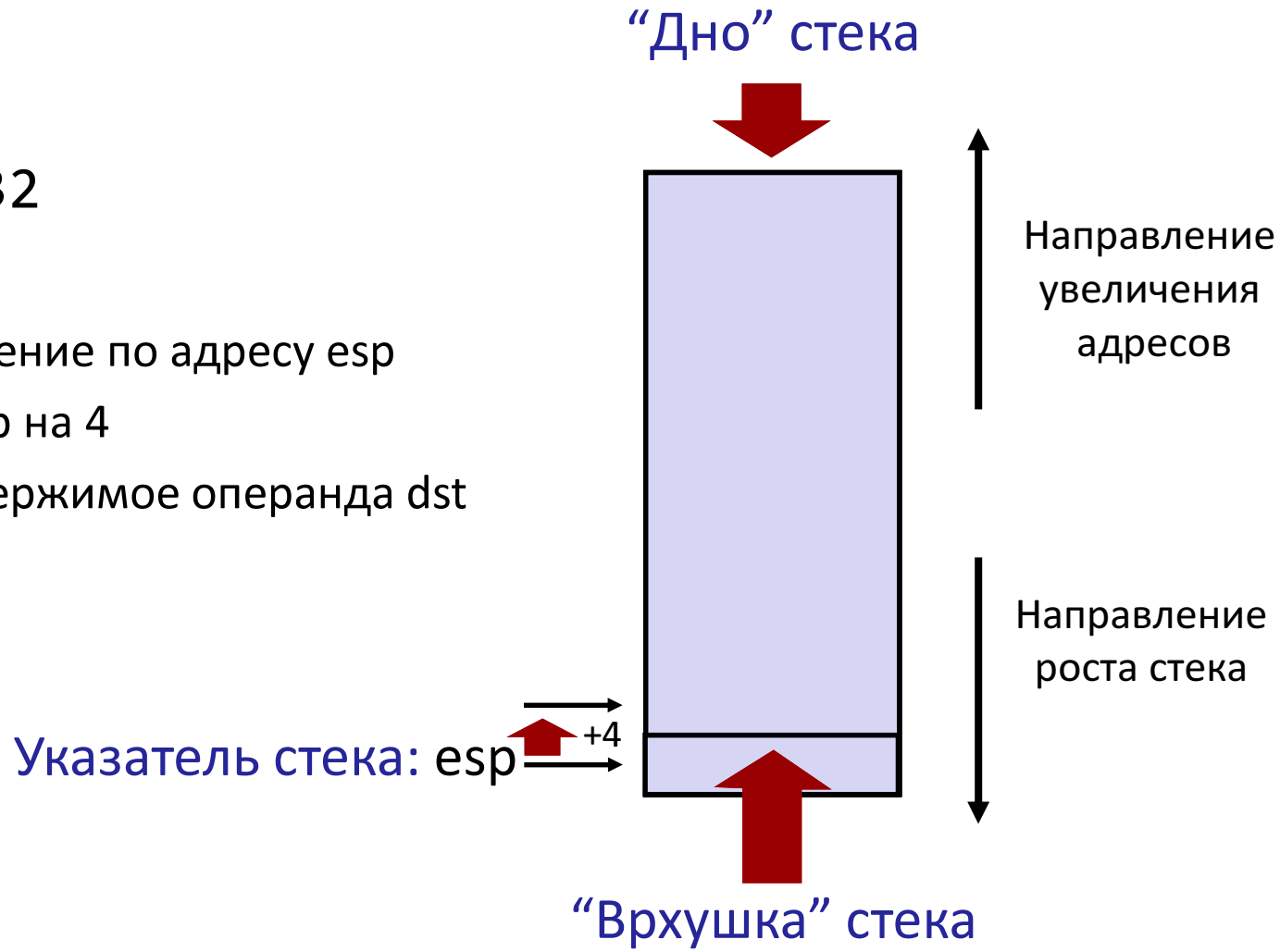
- `push src`
  - `r/m 16/32`
  - `imm 8/16/32`

  – Извлечь содержимое операнда src
  – Уменьшить esp на 4 (2)
  – Записать значение по адресу esp



Выгрузка данных из стека: Pop

- `pop dst`
  - `r/m 16/32`
    - Извлечь значение по адресу esp
    - Увеличить esp на 4
    - Записать содержимое операнда dst



Языки программирования (ЯП), базирующиеся на стеке вызовов

- ЯП с поддержкой рекурсии
  - C, Pascal, Java
  - Код функции можно вызывать повторно (“Reentrant”)
    - Одновременно могут выполняться несколько вызовов функции
  - Необходимо выделять память под сохранение состояния каждого работающего вызова
    - Аргументы
    - Локальные переменные
    - Адрес возврата
- Стек
  - Сохранять состояние вызова функции надо в ограниченный период времени: от момента вызова до момент выхода
  - Вызываемая функция всегда завершается до вызывающей
- Стек выделяется **Фреймами**
  - Состояние отдельного вызова функции

Порядок вызова функции

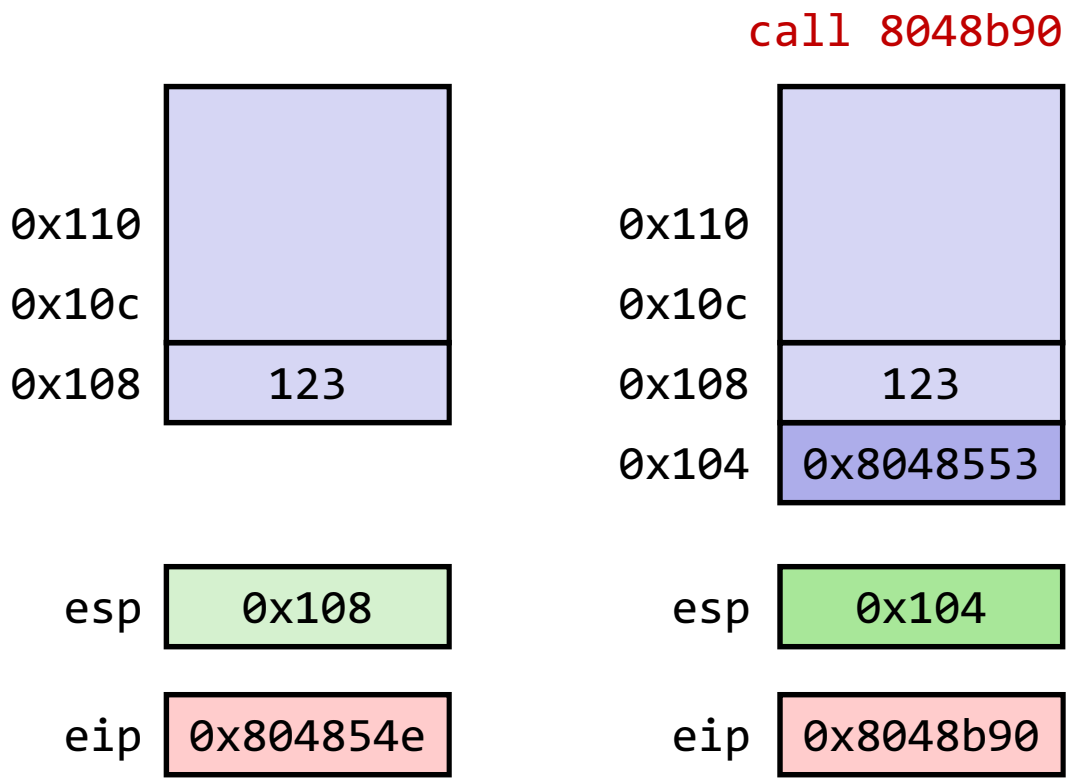
- Аппаратный стек используется для вызова функций и возврата из них
- Вызов функции: **call label**
  - На стек помещается адрес возврата
  - Выполняется прыжок на метку *label*
- Адрес возврата:
  - Адрес инструкции непосредственно расположенной за инструкцией call

```
804854e:   e8 3d 06 00 00      call   8048b90 <main>
8048553:   50                  push   eax
```

  - Адрес возврата = 0x8048553
- Возврат из функции: **ret**
  - Выгрузка адреса из стека
  - Прыжок на этот адрес

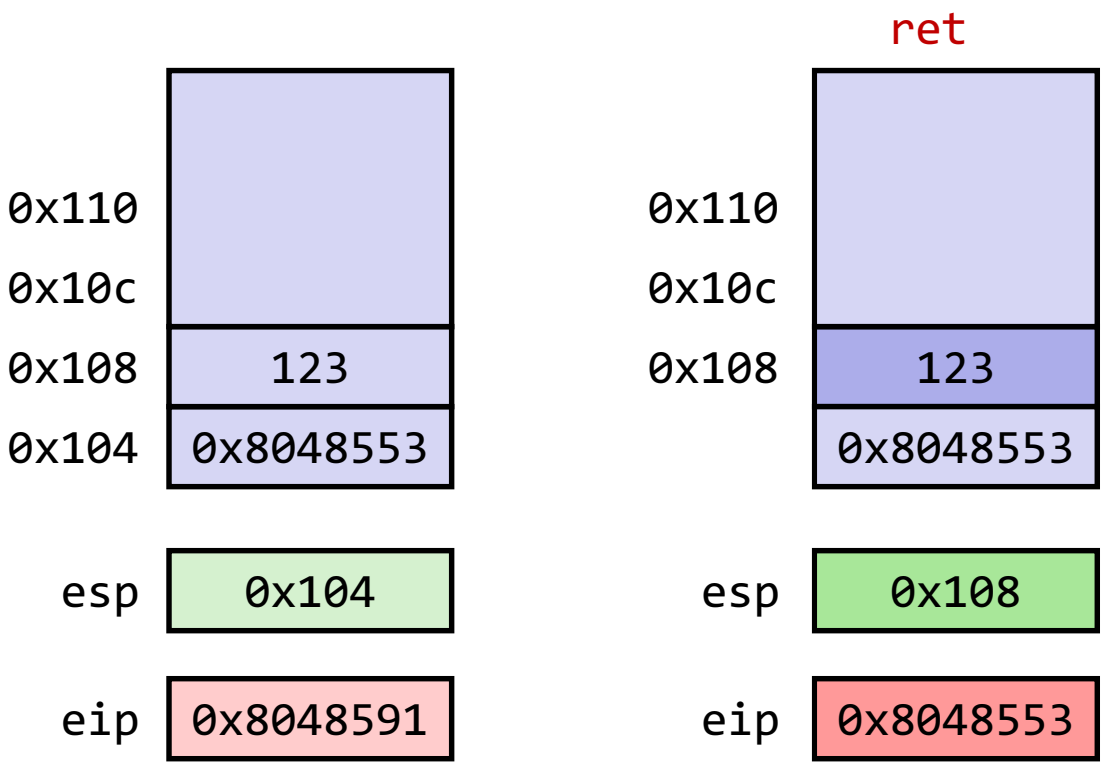
Вызов функции

```
804854e:      e8 3d 06 00 00          call   8048b90 <main>
8048553:      50                      push   eax
```



Выход из функции

```
8048591:      c3                        ret
```

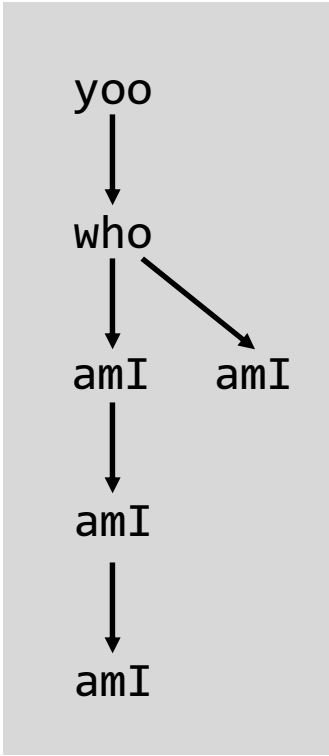


Пример цепочки вызовов

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

```
who(…)
{
   • • •
   amI();
   • • •
   amI();
   • • •
}
```

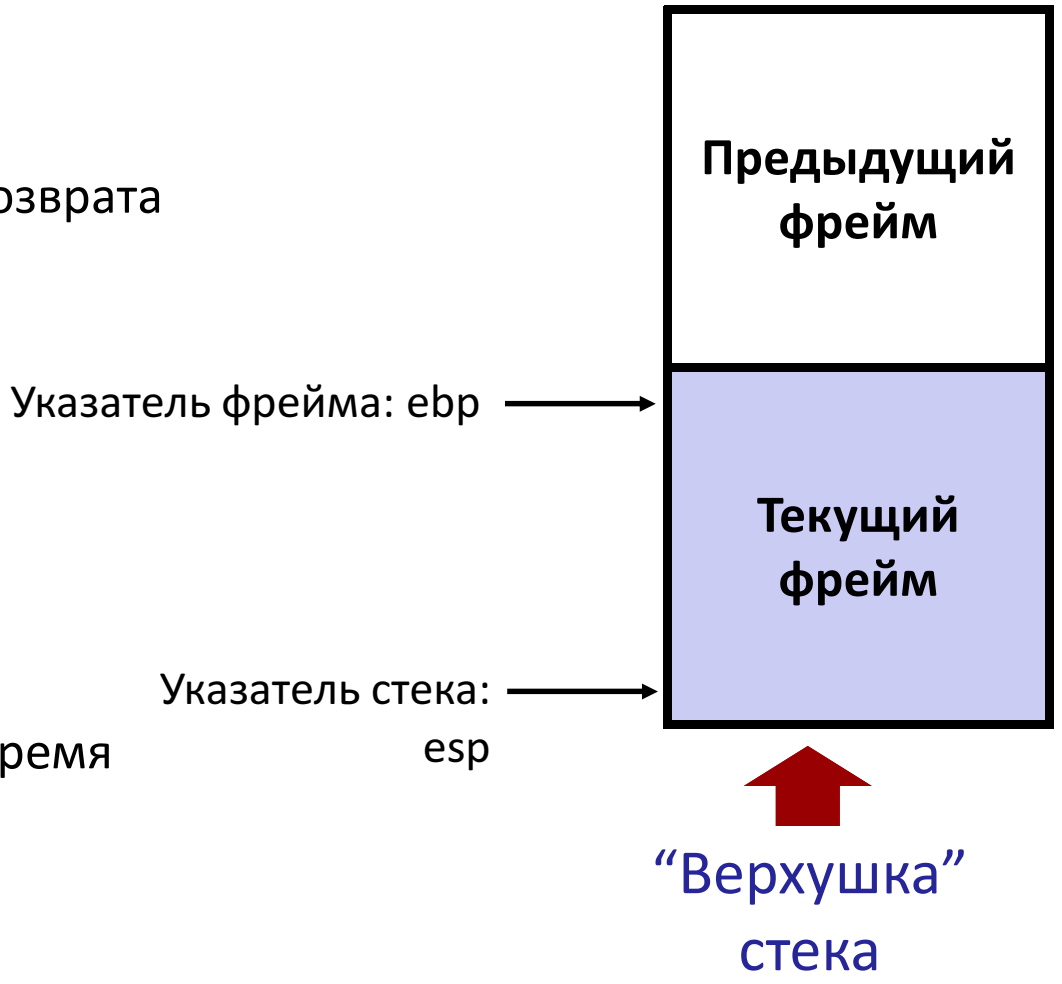
```
amI(…)
{
   •
   •
   amI();
   •
   •
}
```

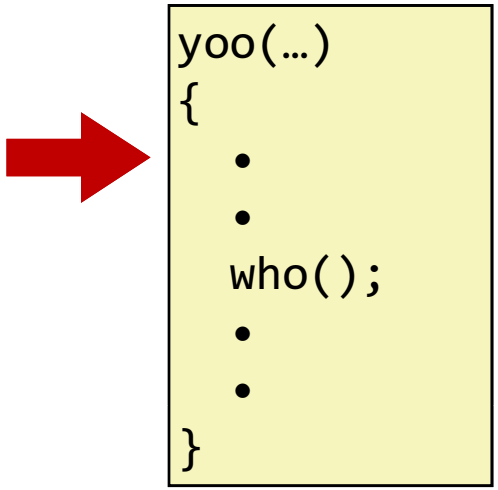


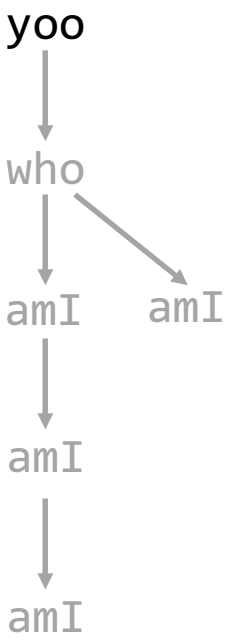
Функция amI() рекурсивная

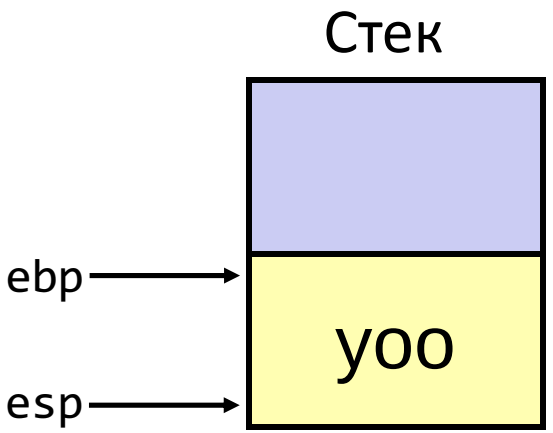
Стек фреймов

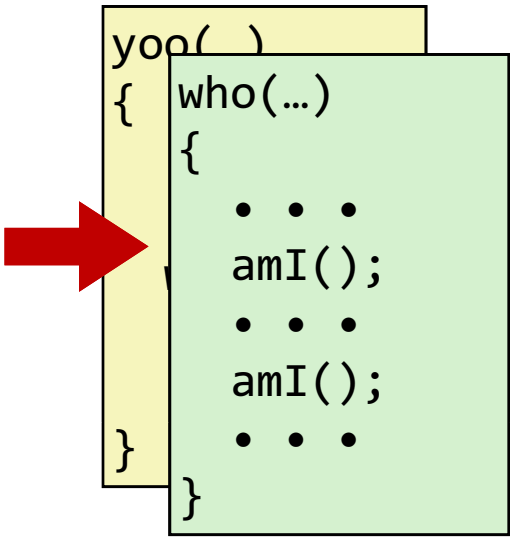
- Во фрейме размещаются
  - Локальные переменные
  - Данные, необходимые для возврата из функции
  - Временные переменные
- Управление фреймами
  - Пространство выделятся во время входа в функцию
    - «пролог» функции
  - Освобождается на выходе
    - «эпилог» функции

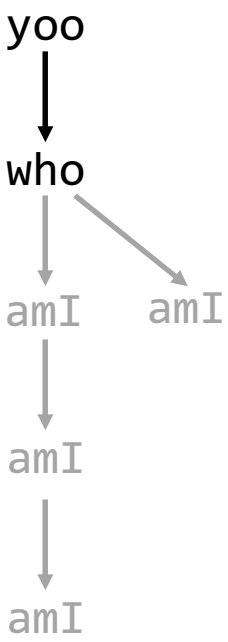


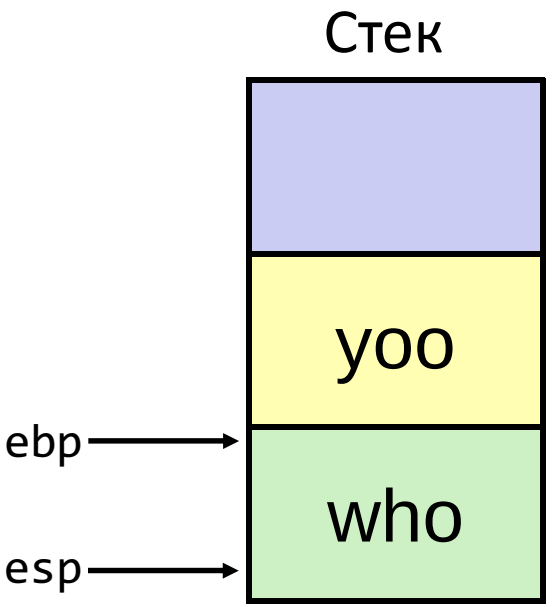


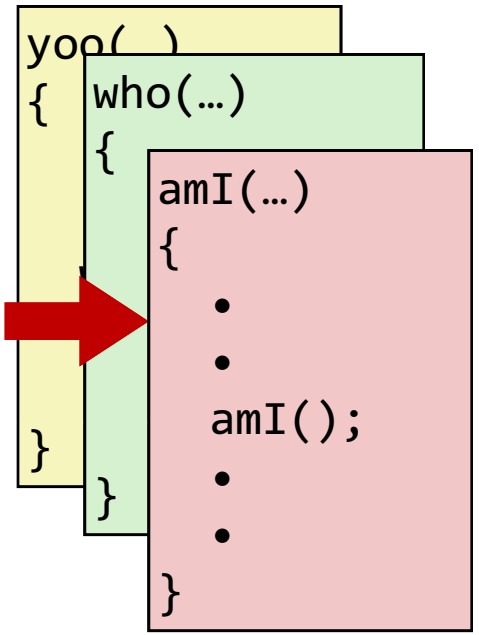


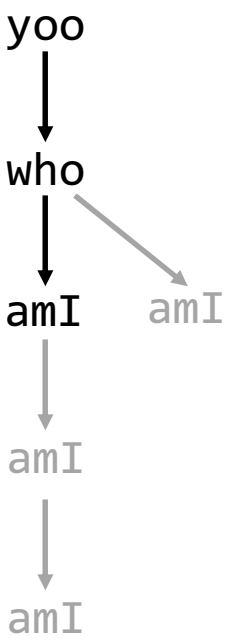


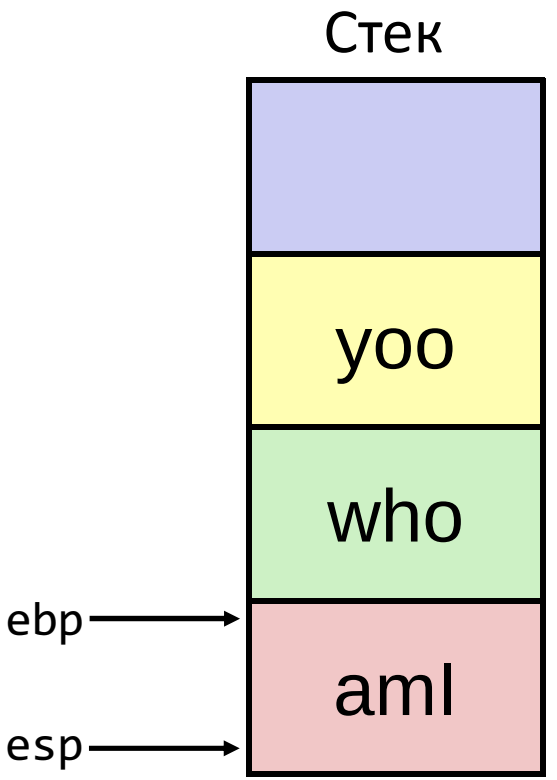


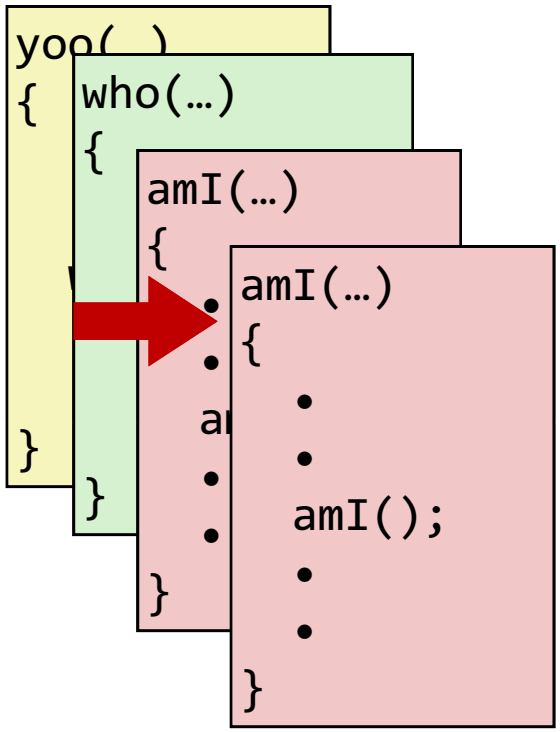


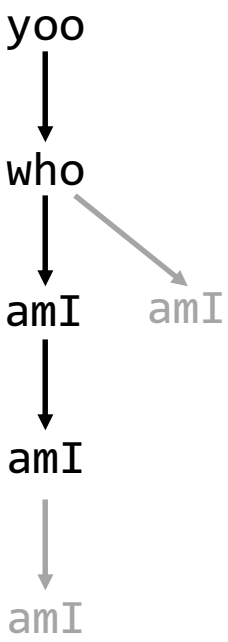


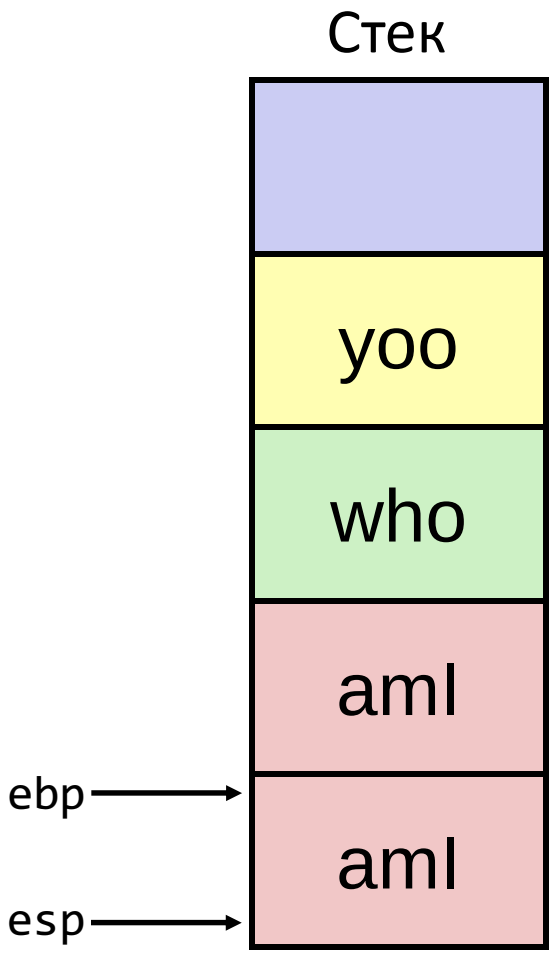


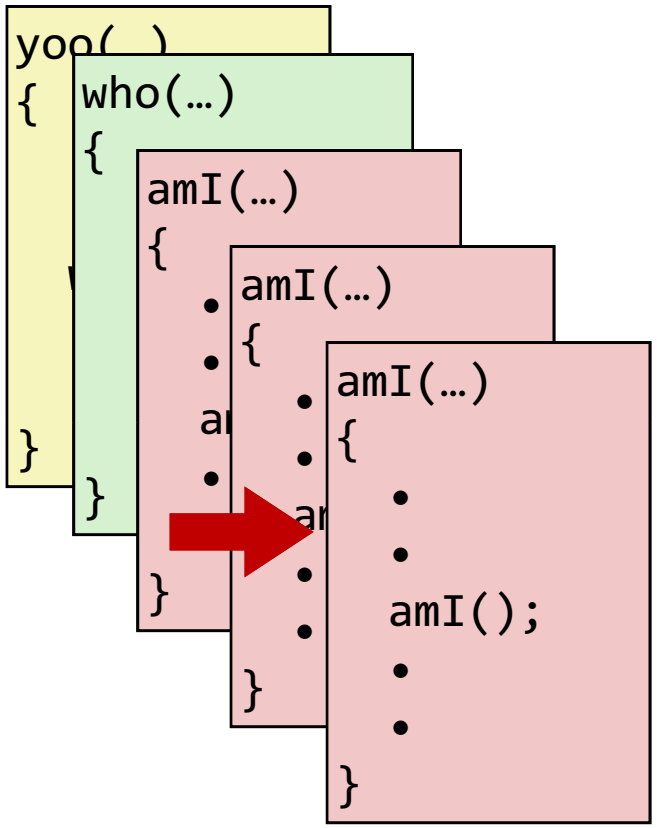


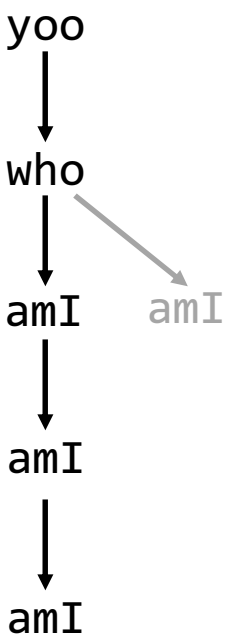


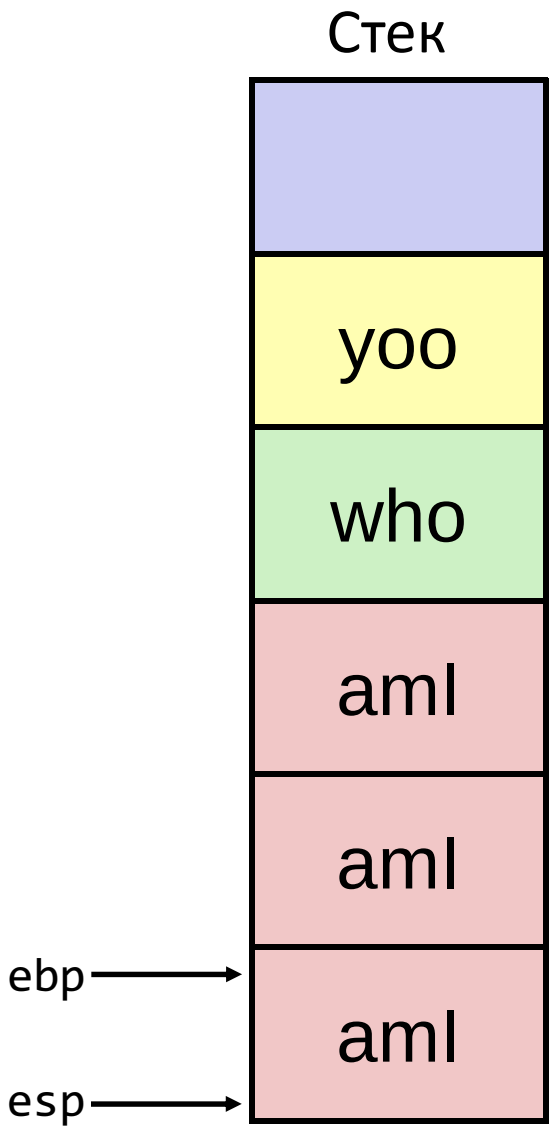


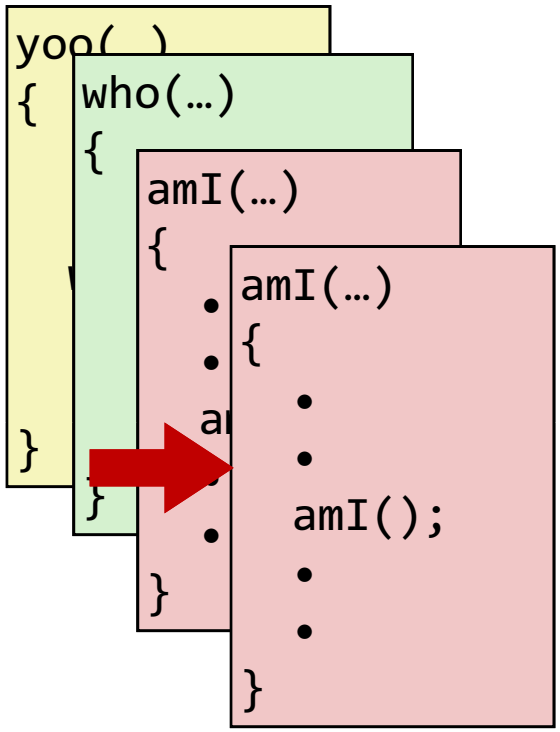


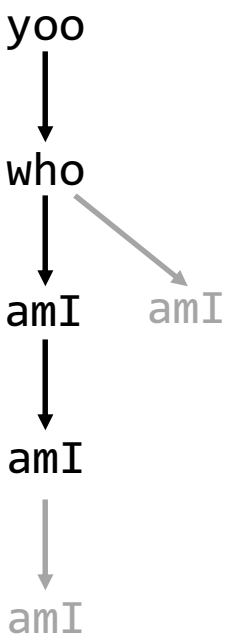


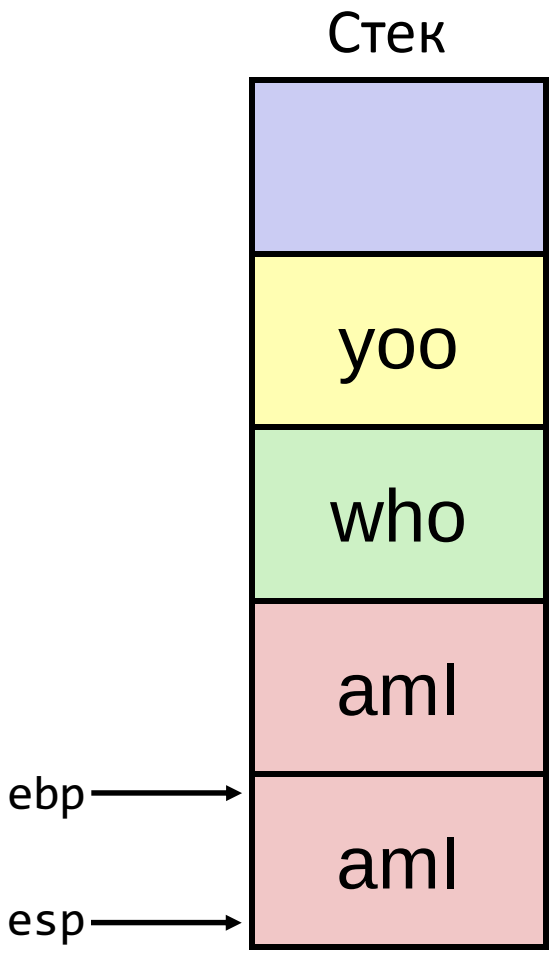


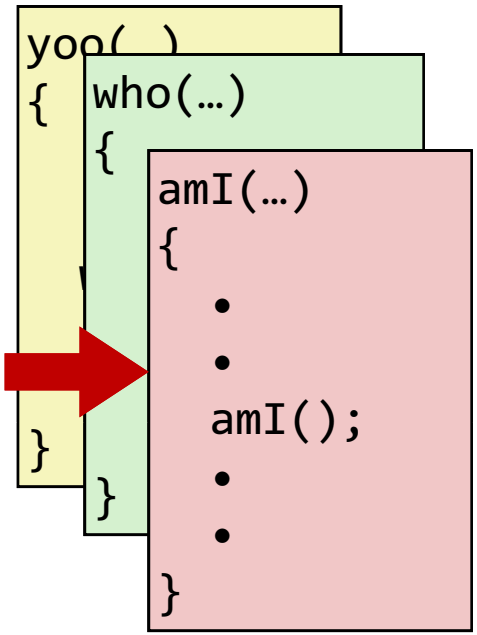


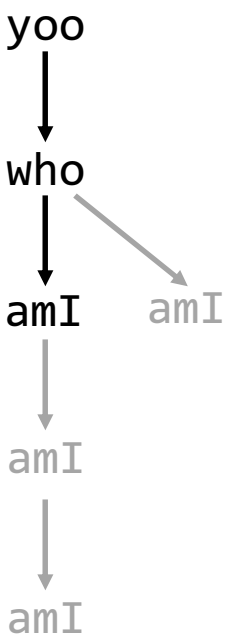


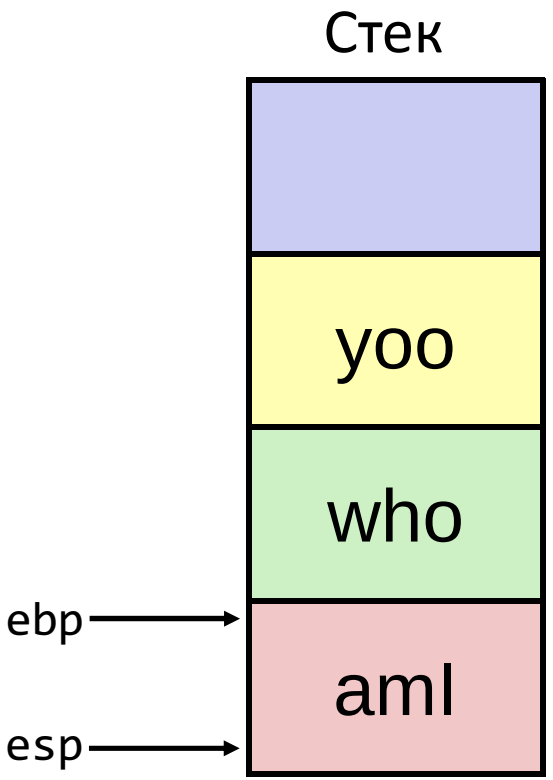


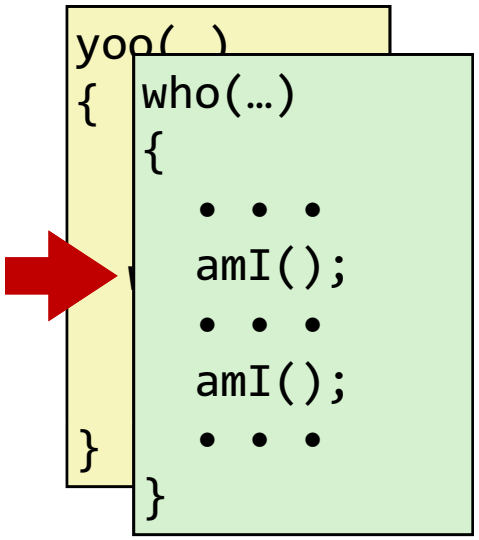


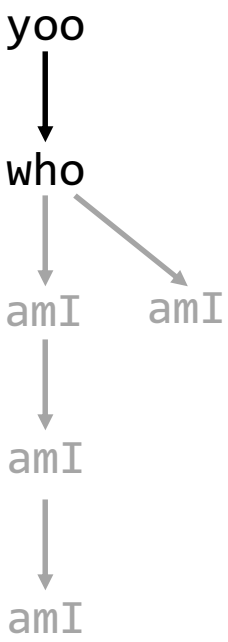


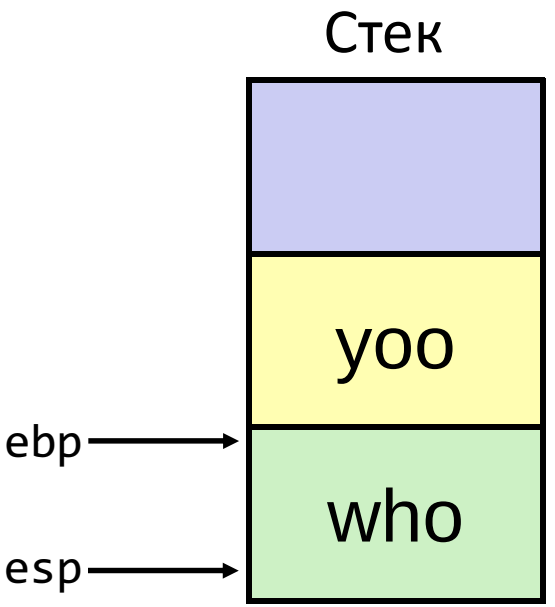


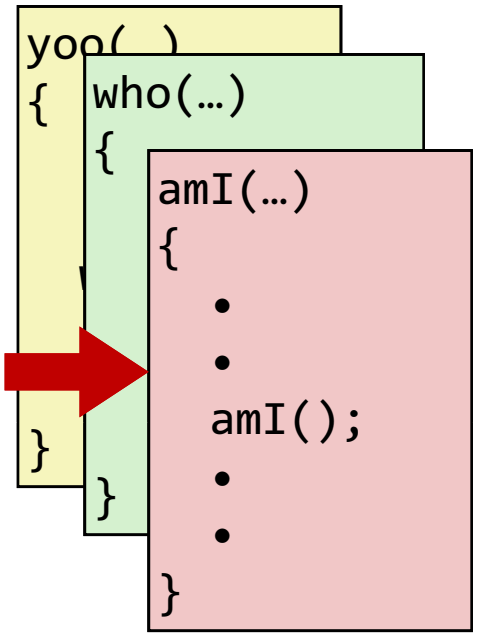


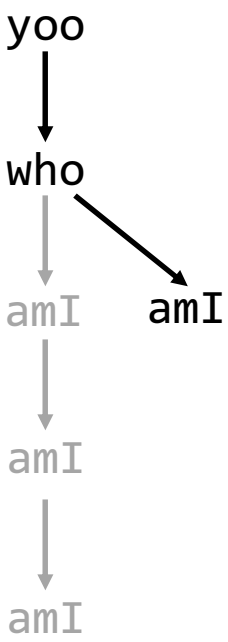


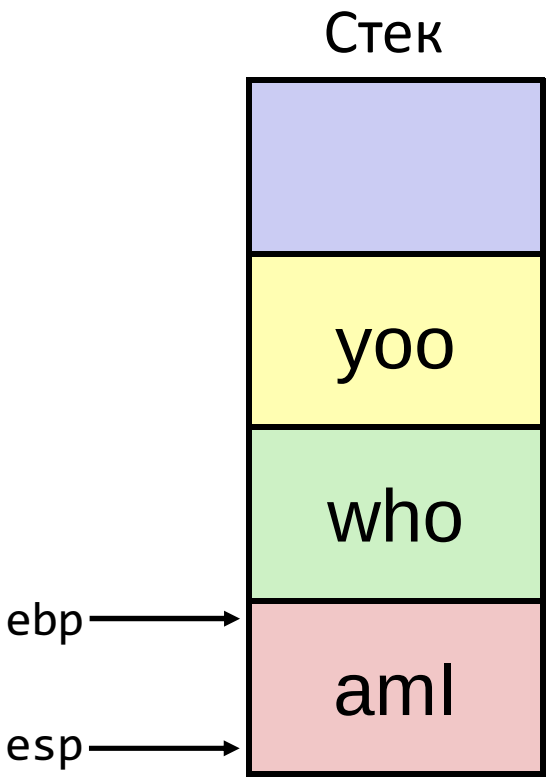


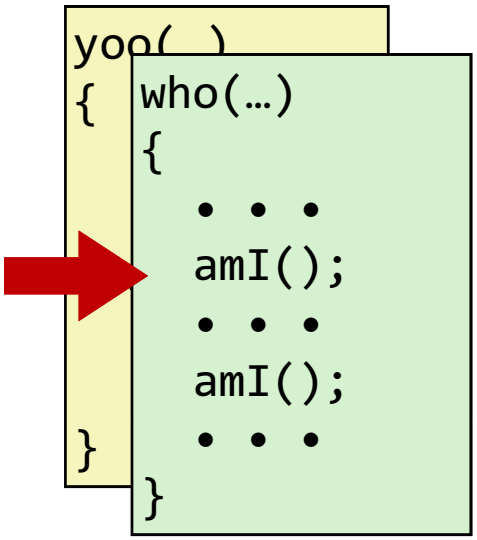


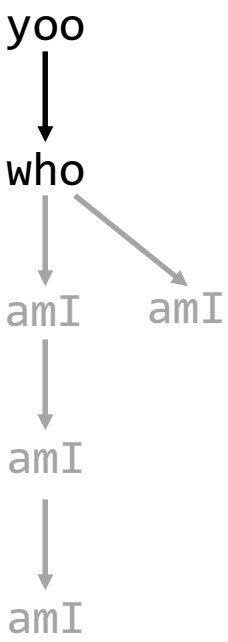


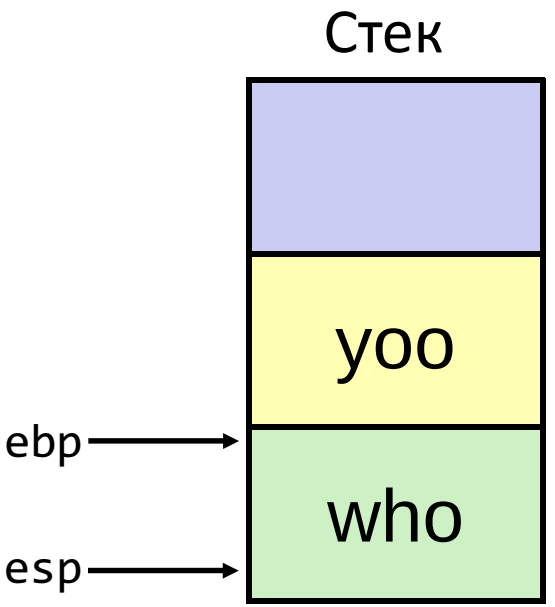


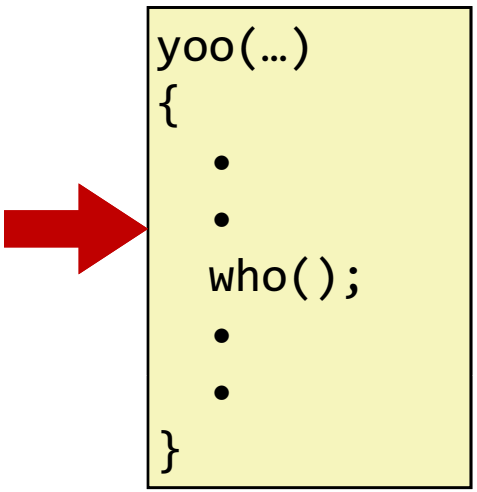


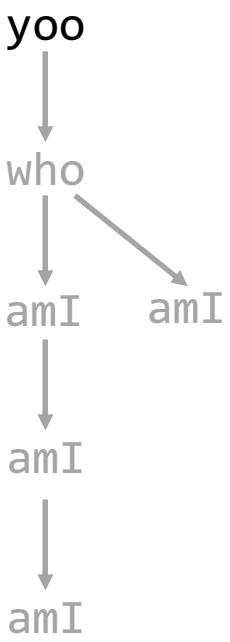


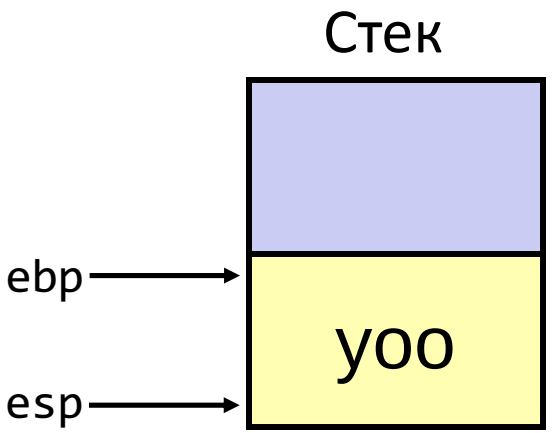






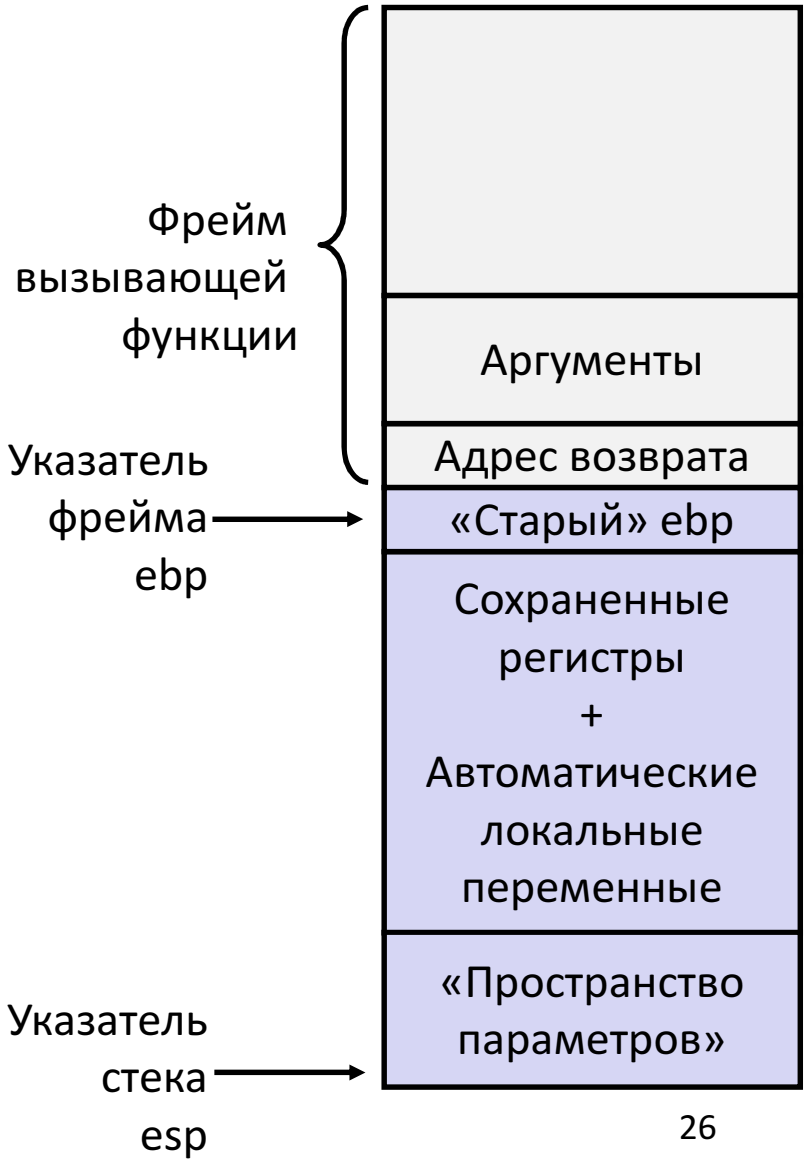






Организация фрейма в IA32/Linux

- Текущий фрейм (от “верхушки” ко «дну»)
  - “Пространство параметров”: фактические параметры вызываемых функций
  - Локальные переменные
  - Сохраненные регистры
  - Прежнее значение указателя фрейма
- Фрейм вызывающей функции
  - Адрес возврата
    - Помещается на стек инструкцией call
  - Значения фактических аргументов для текущего вызова



```
int main() {
   int a = 1, b = 2, c;
   c = sum(a, b);
   return 0;
}

int sum(int x, int y) {
   int t = x + y;
   return t;
}
```

```
%include ‘io.inc’
section .text

global CMAIN
CMAIN:
   mov    dword [ebp-16], 0x1  ; (1)
   mov    dword [ebp-12], 0x2  ; (2)
   mov    eax, dword [ebp-12]  ; (3)
   mov    dword [esp+4], eax   ; (4)
   mov    eax, dword [ebp-16]  ; (5)
   mov    dword [esp], eax     ; (6)
   call   sum                  ; (7)
   mov    dword [ebp-8], eax   ; (8)
...
global sum
sum:
   push   ebp                  ; (9)
   mov    ebp, esp             ; (10)
   sub    esp, 0x10            ; (11)
   mov    edx, dword [ebp+12]  ; (12)
   mov    eax, dword [ebp+8]   ; (13)
   add    eax, edx             ; (14)
   mov    dword [ebp-4], eax   ; (15)
   mov    eax, dword [ebp-4]   ; (16)
   mov    esp, ebp             ; (17)
   pop    ebp                  ; (18)
   ret                         ; (19)
```