Лекция 5

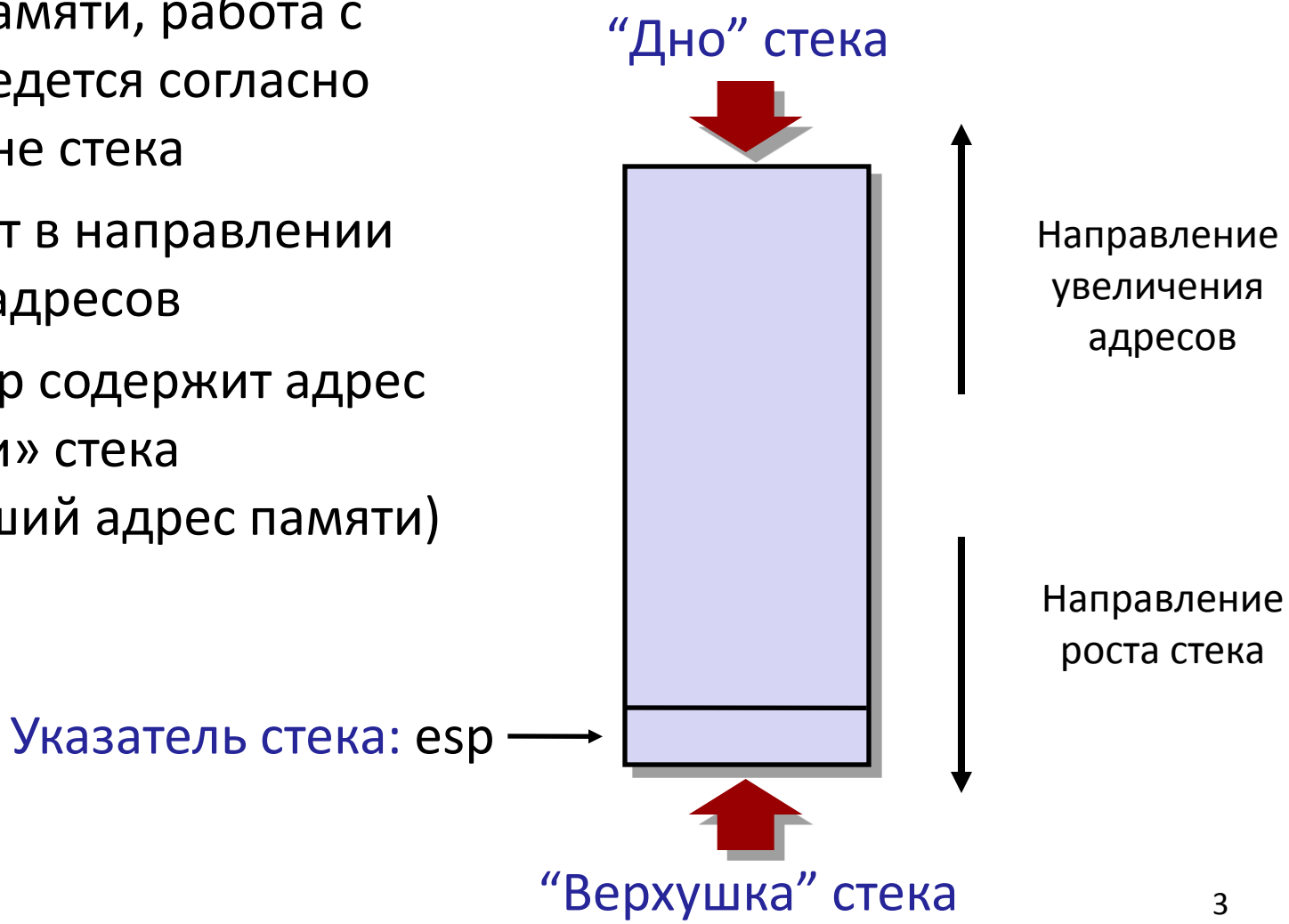
27 февраля

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

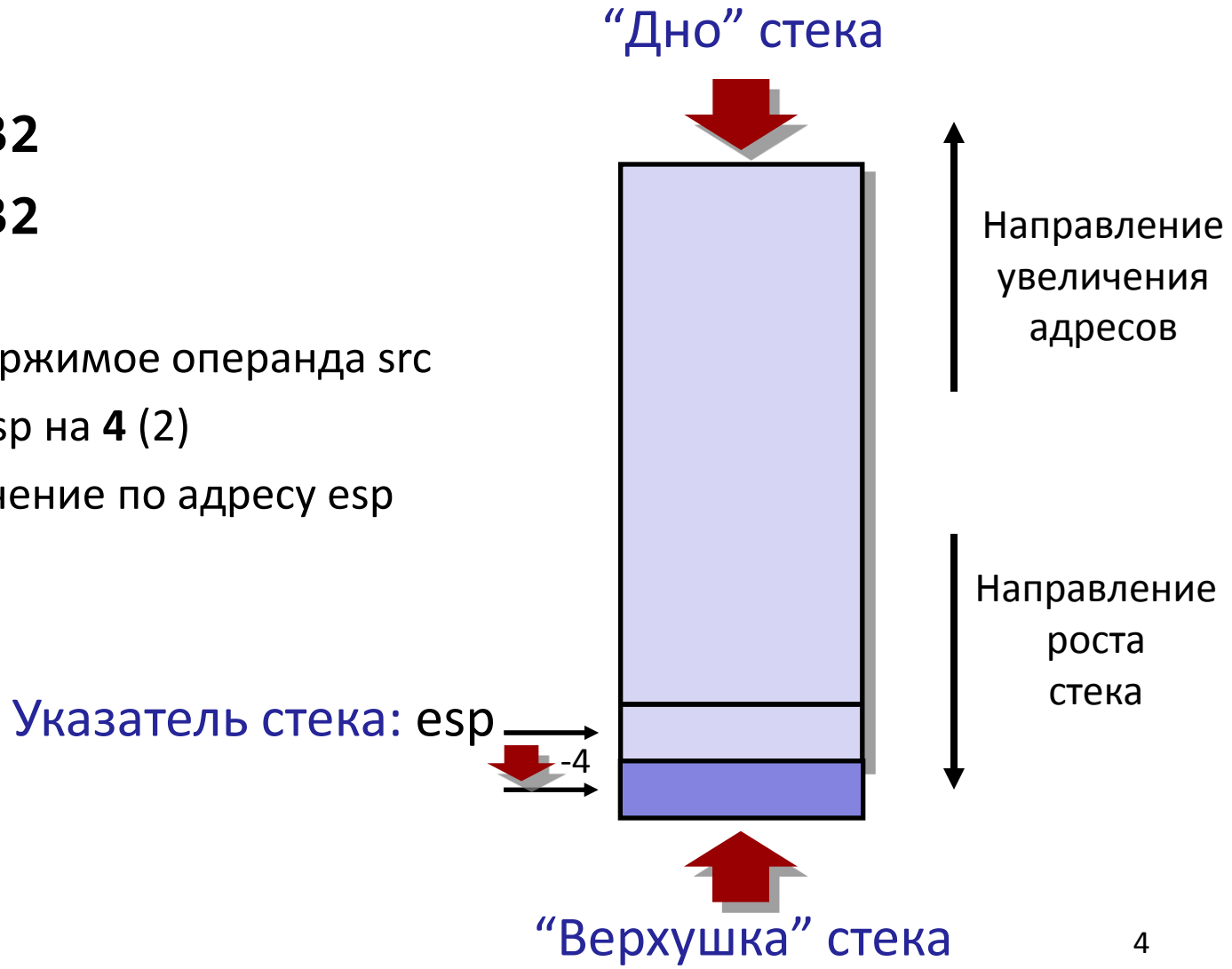
- Область памяти, работа с которой ведется согласно дисциплине стека
- Стек растет в направлении меньших адресов
- Регистр esp содержит адрес «верхушки» стека (наименьший адрес памяти)



Загрузка данных в стек: Push

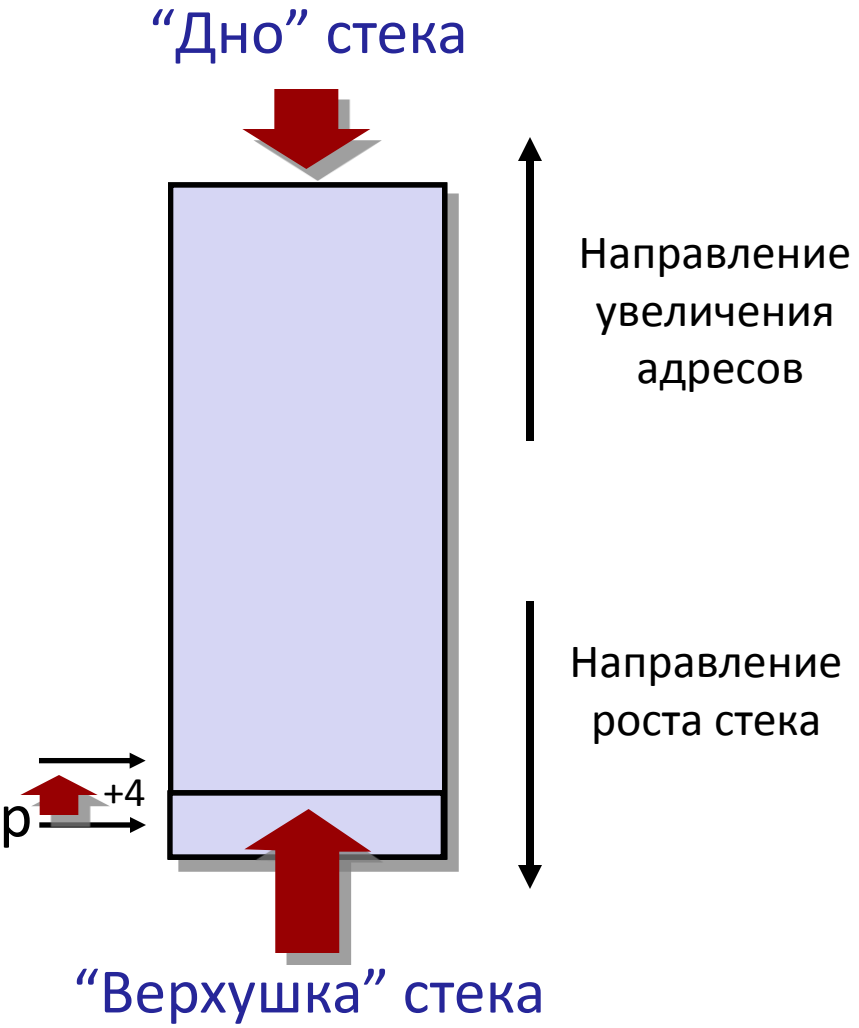
- `push src`
  - `r/m 16/32`
  - `i 8/16/32`

  – 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** (2)
  - 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

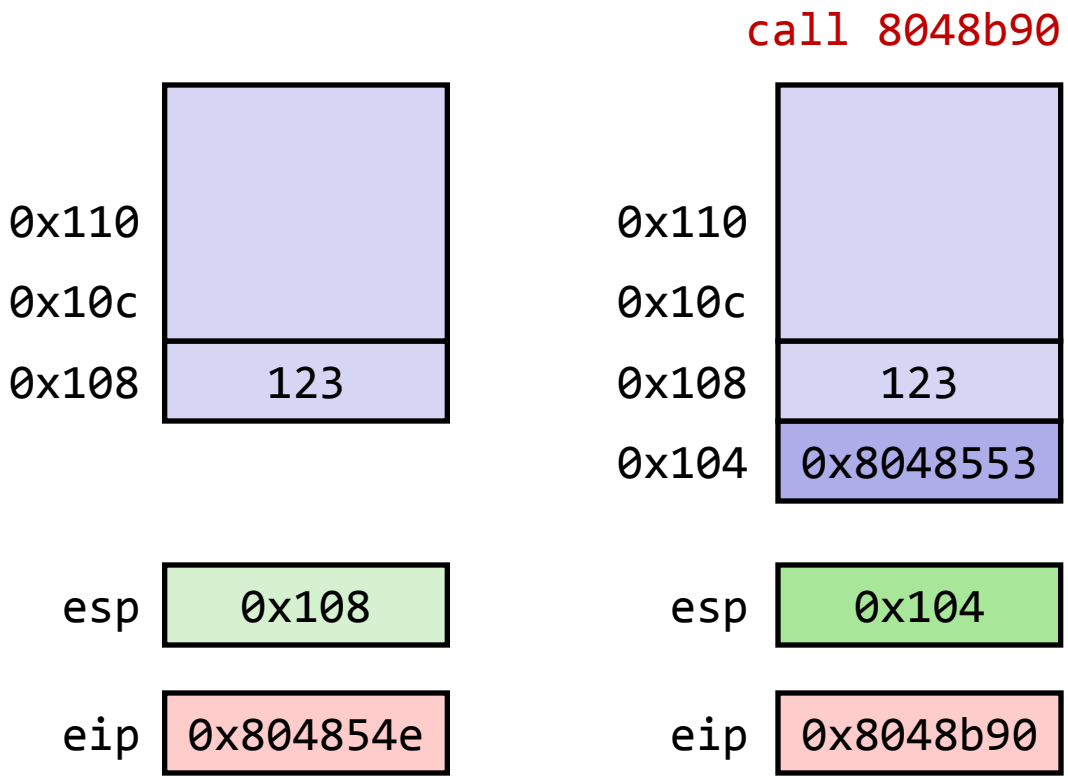
- Аппаратный стек используется для вызова функций и возврата из них
- Вызов функции: **`call label`**
  - На стек помещается адрес возврата
  - Выполняется прыжок на метку *label*
- Адрес возврата:
  - Адрес инструкции непосредственно расположенной за инструкцией call

```
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

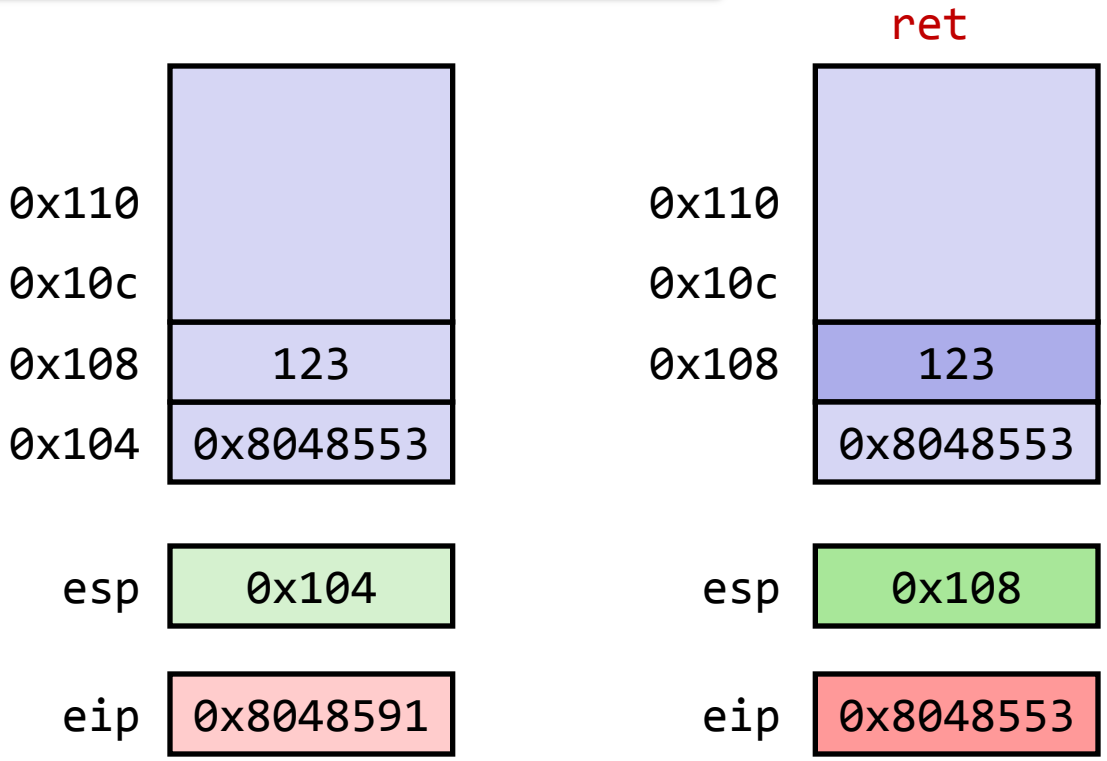
Вызов функции

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



Выход из функции

```
8048591:    c3                    ret
```

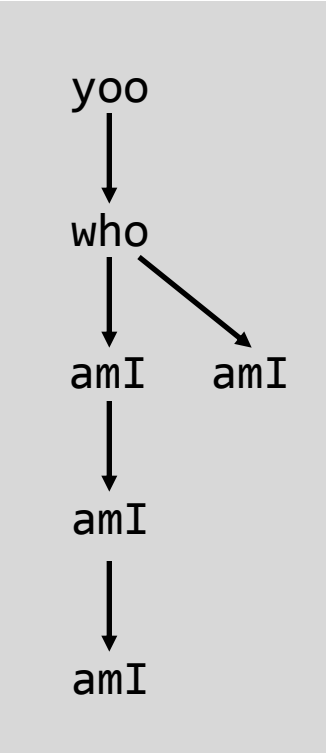


Пример цепочки вызовов

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

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

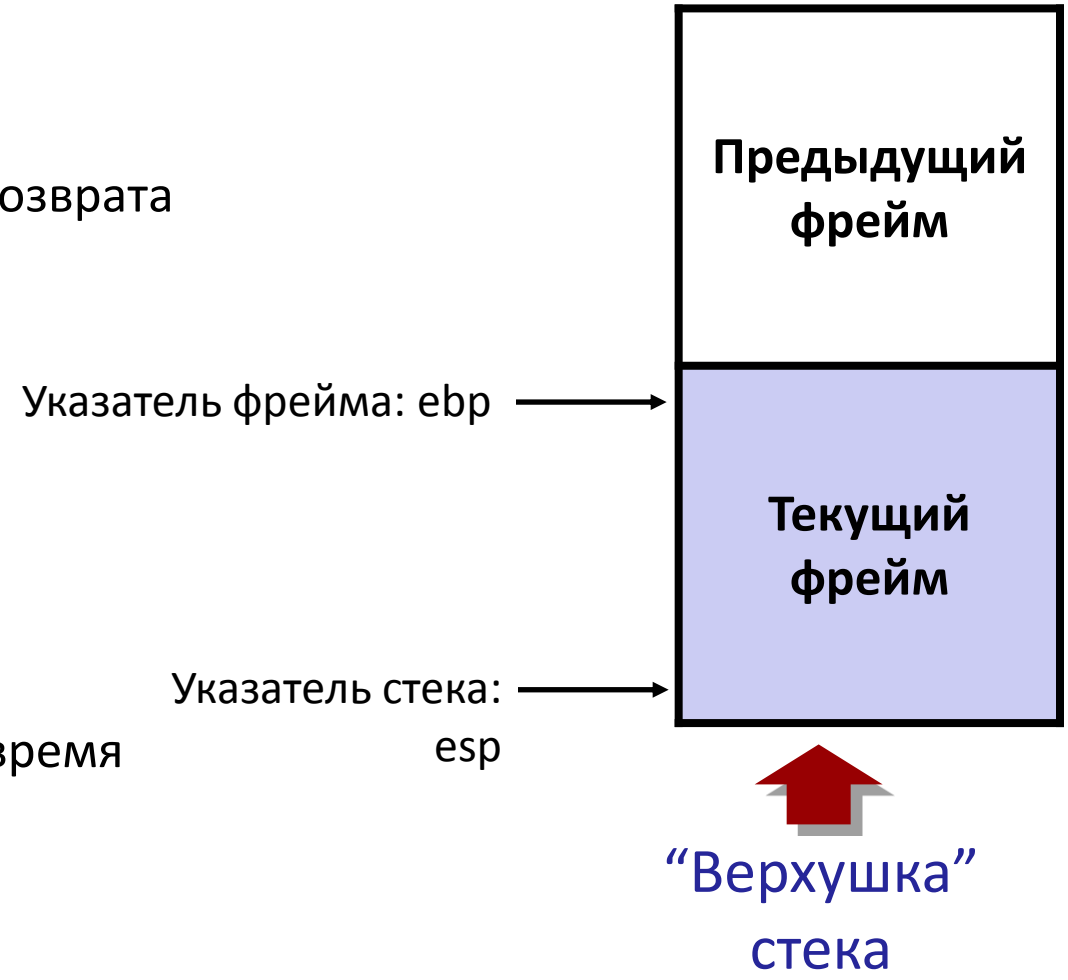
```
amI(…)
{
  •
  •
  amI();
  •
  •
}
```



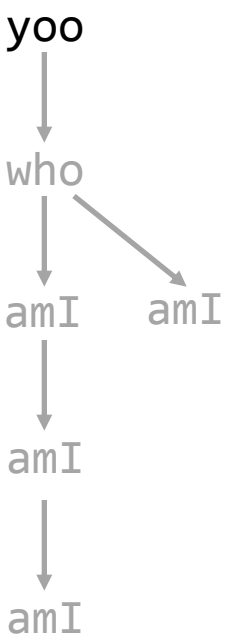
Функция `amI()` рекурсивная

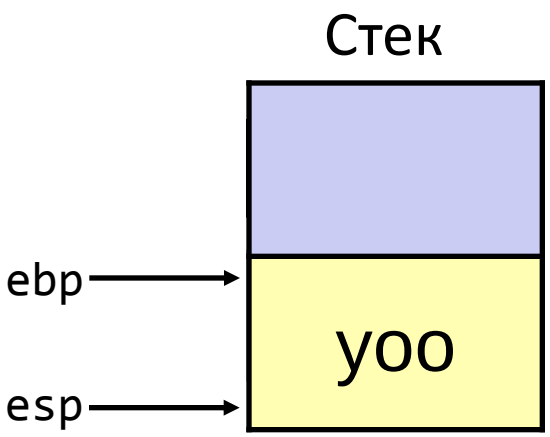
Стек фреймов

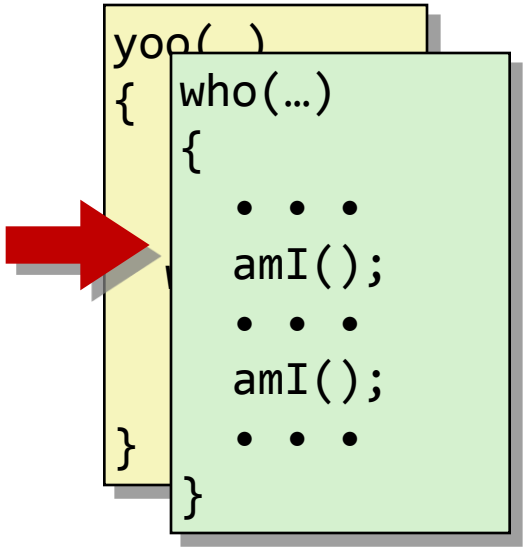
- Во фрейме размещаются
  - Локальные переменные
  - Данные, необходимые для возврата из функции
  - Временные переменные
- Управление фреймами
  - Пространство выделятся во время входа в функцию
    - «пролог» функции
  - Освобождается на выходе
    - «эпилог» функции

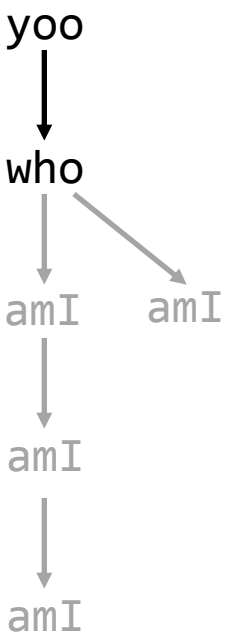


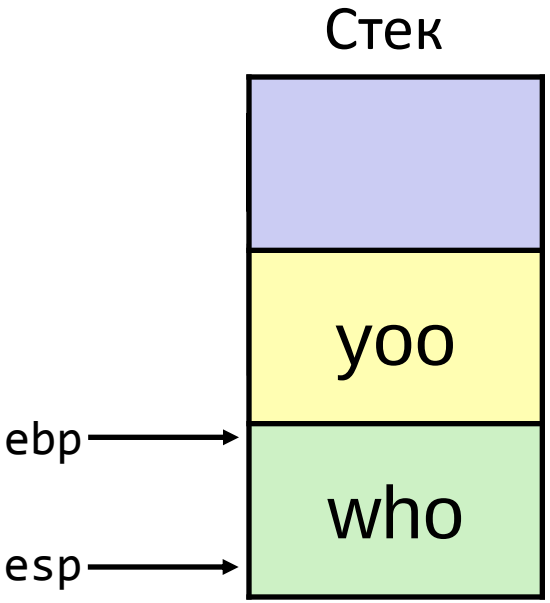
```
yoo(…)
{
   •
   •
   who();
   •
   •
}
```

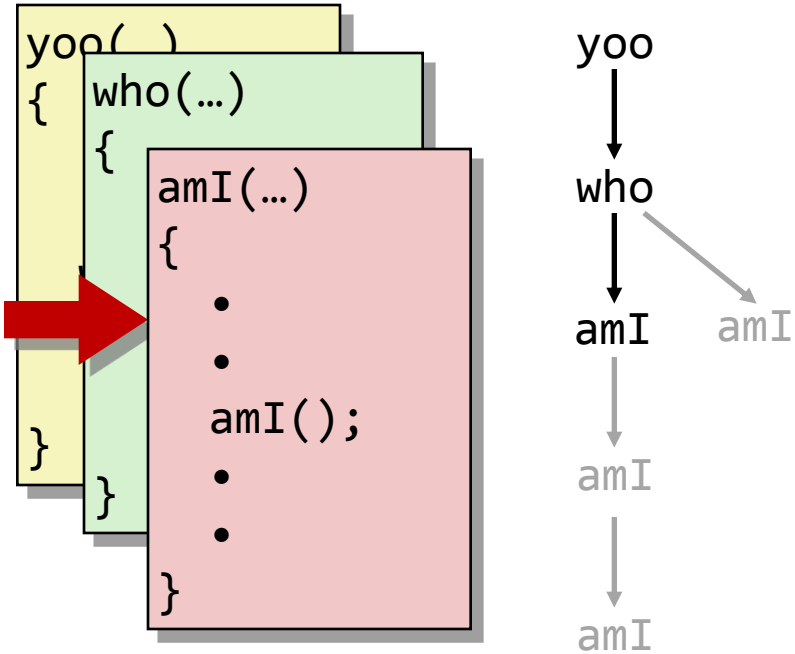


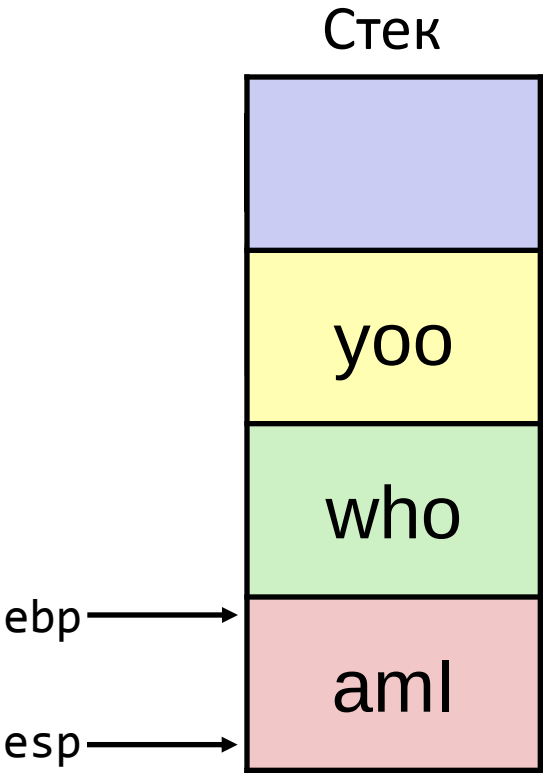


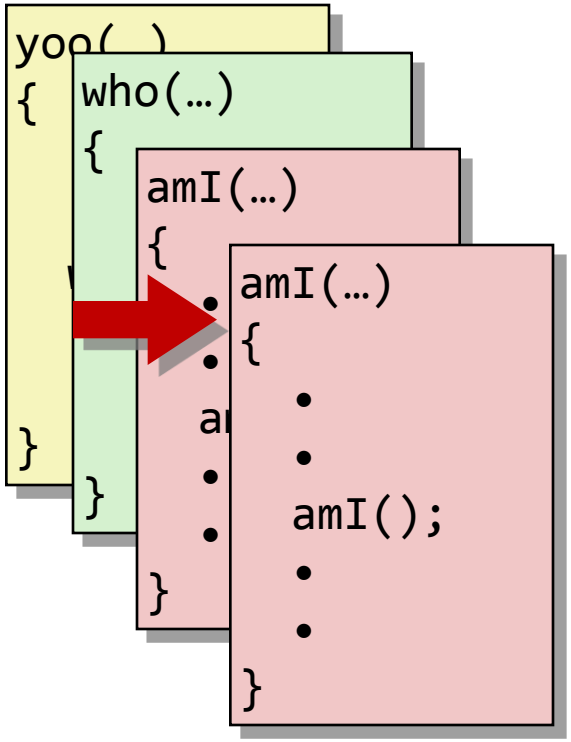


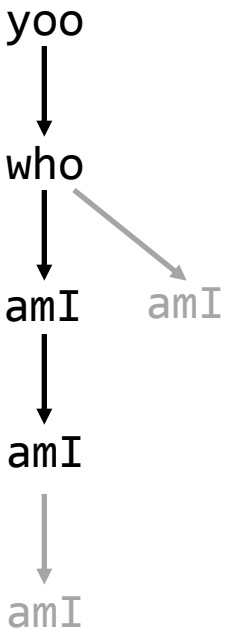


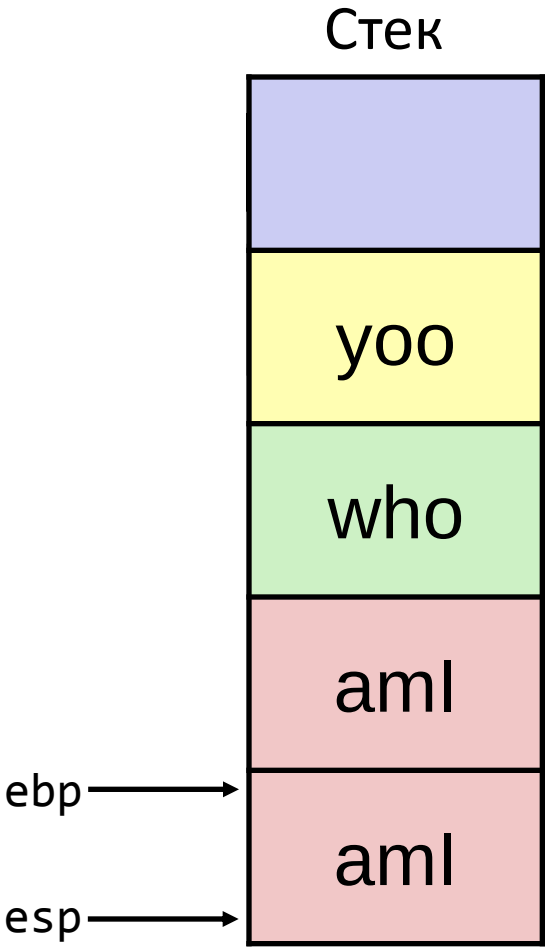


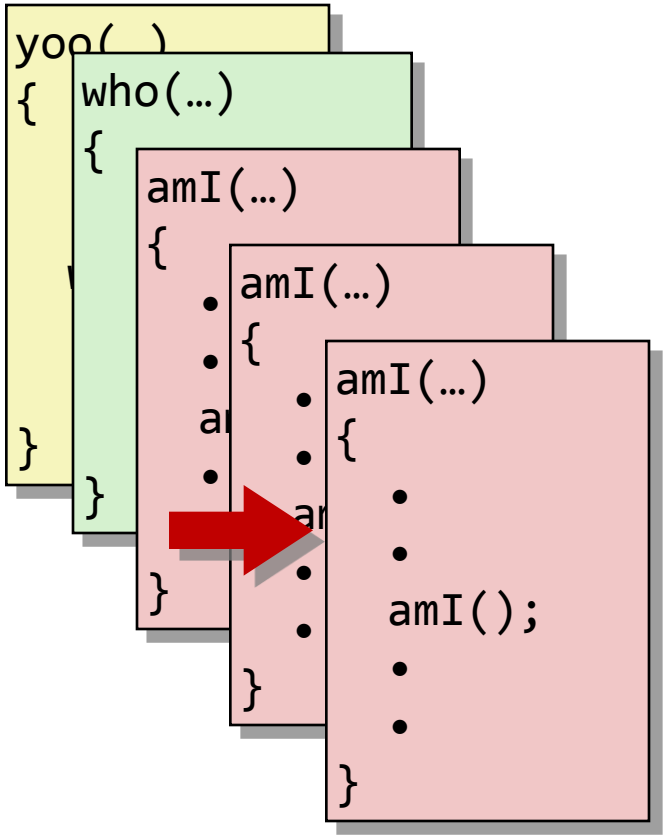


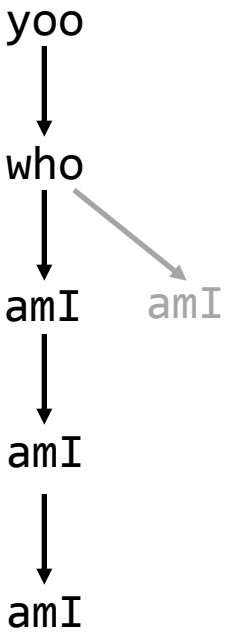


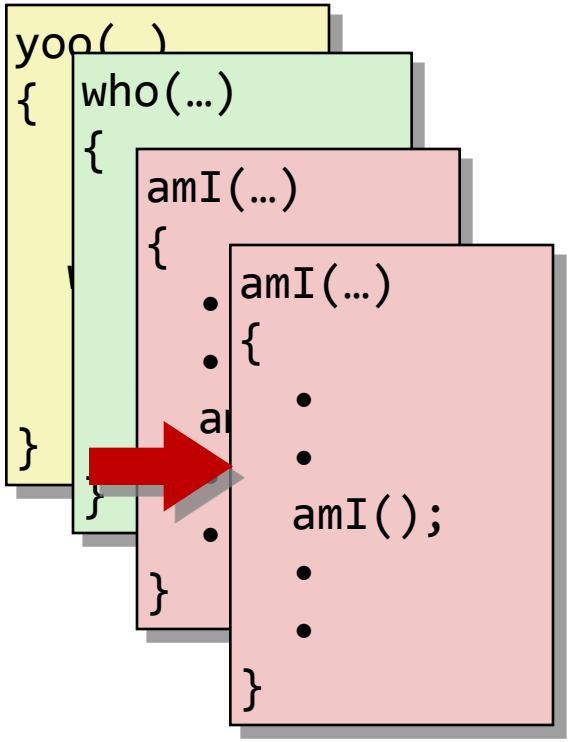


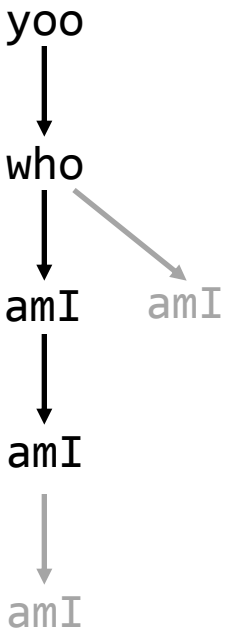


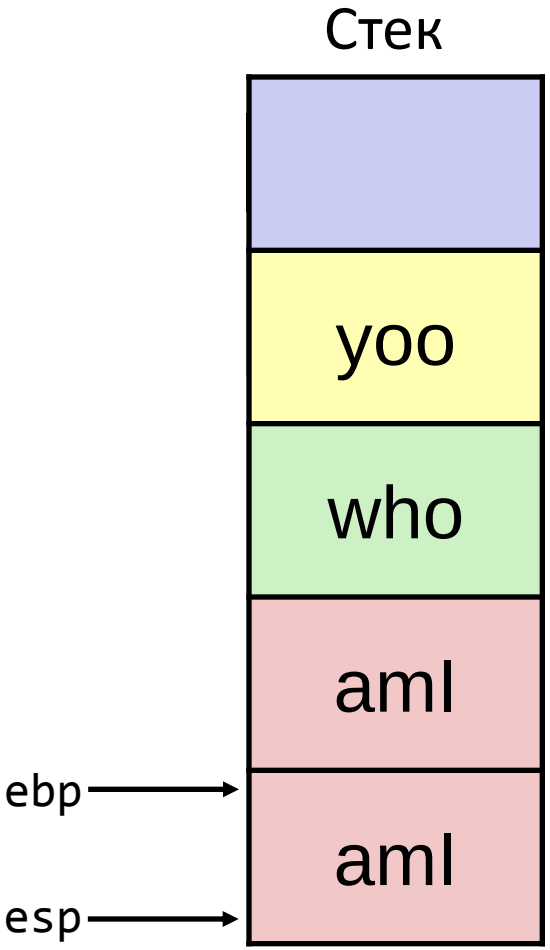


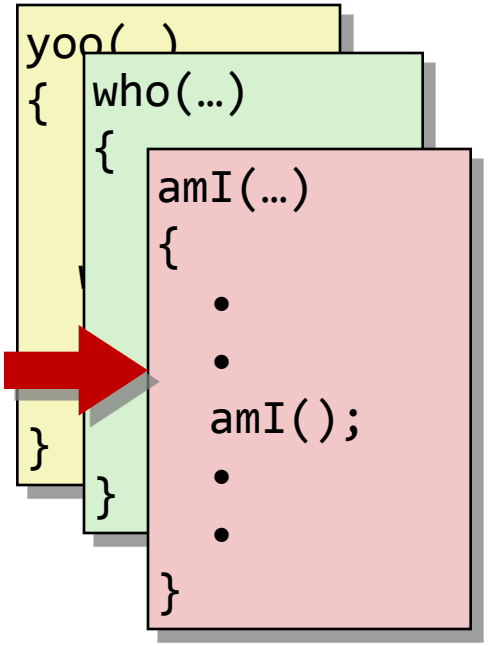


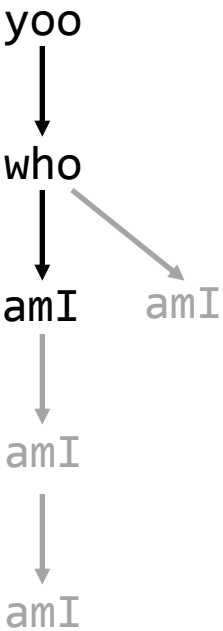


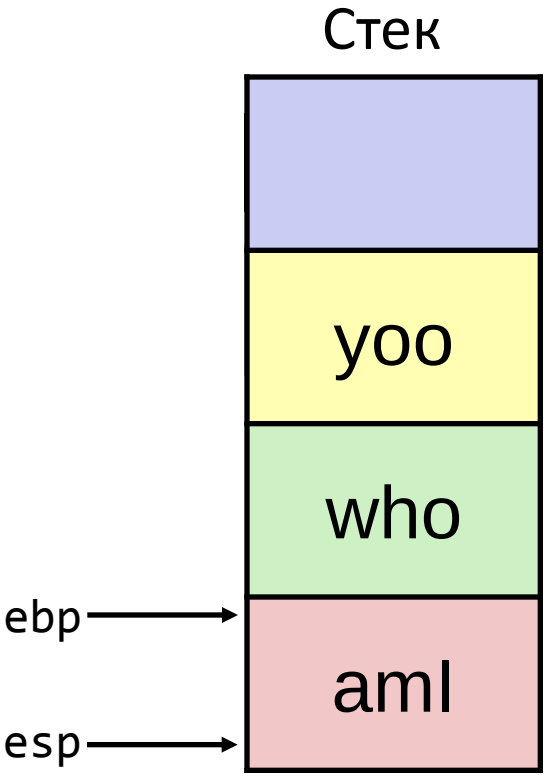


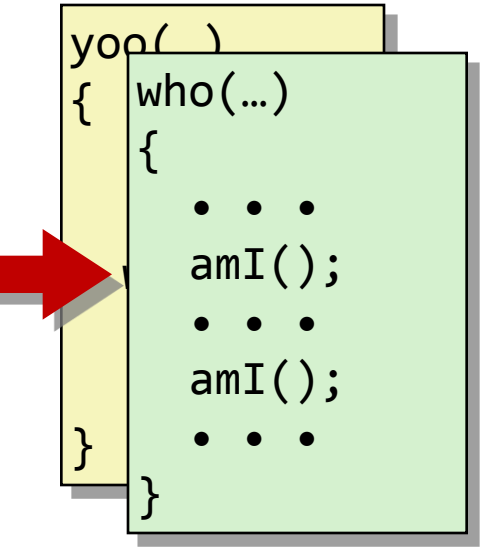


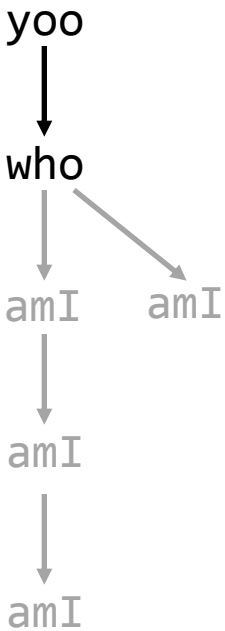


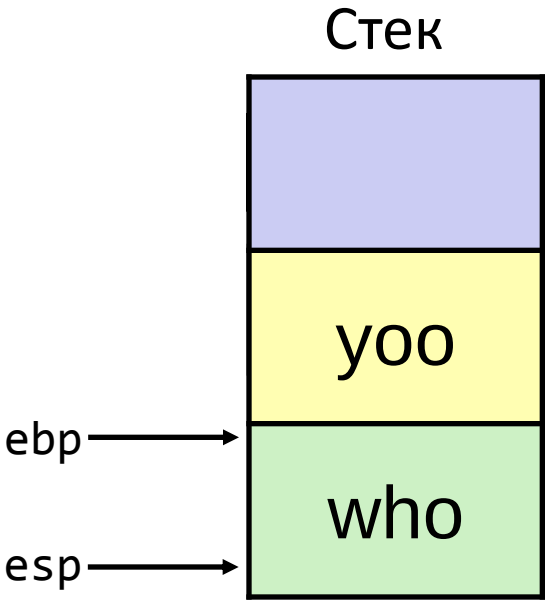


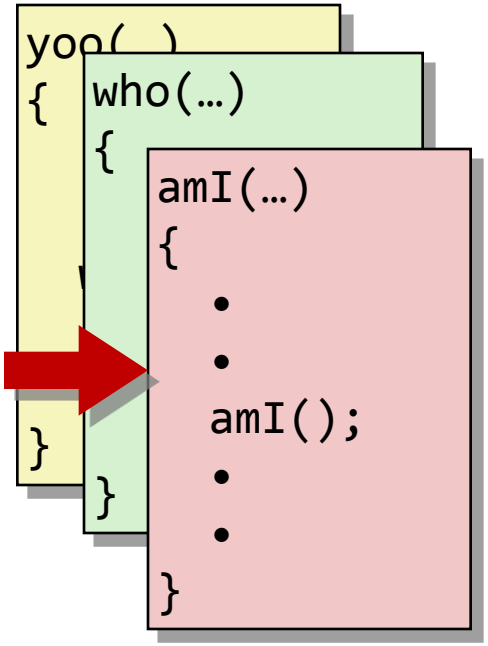


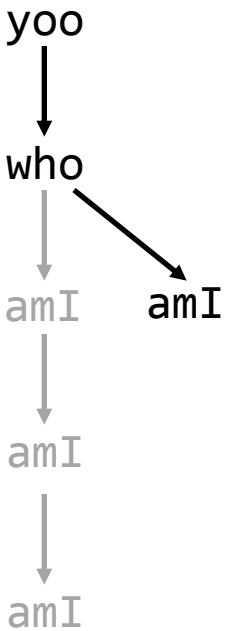


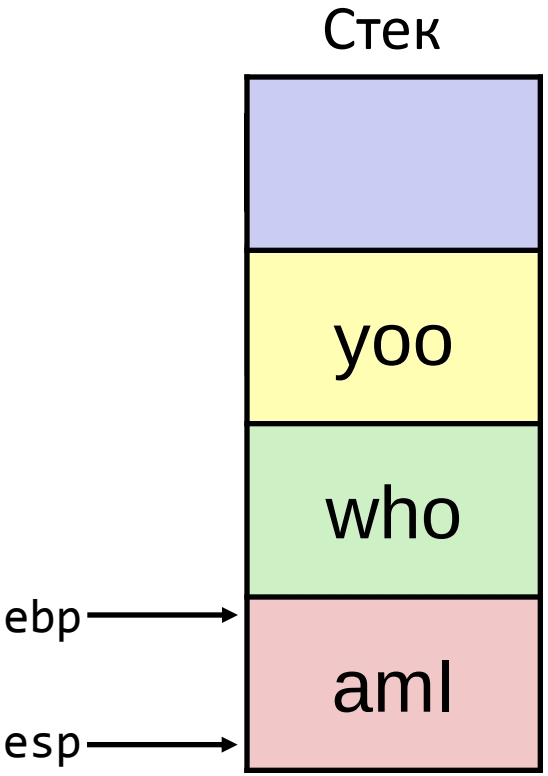


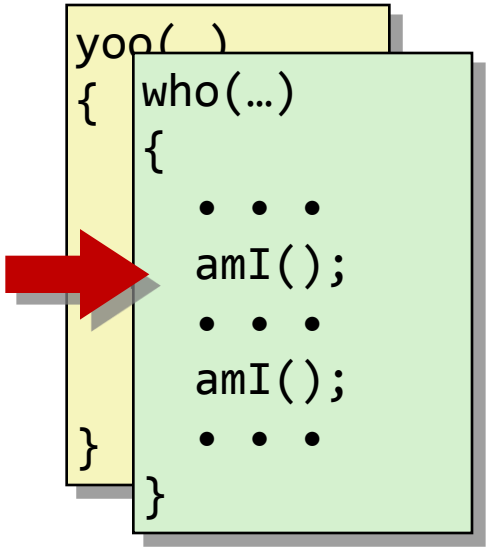


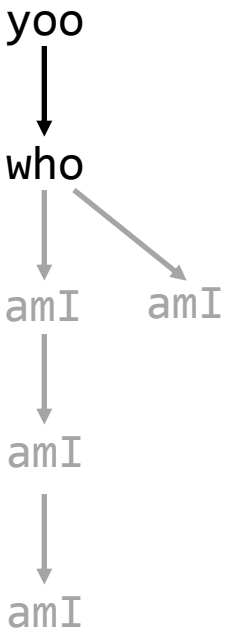


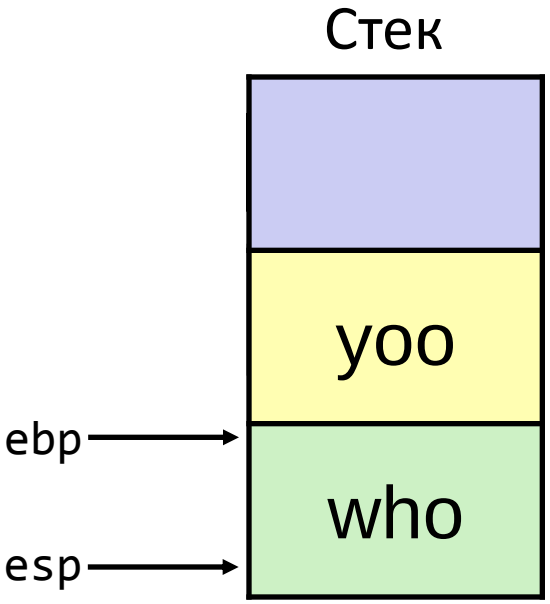




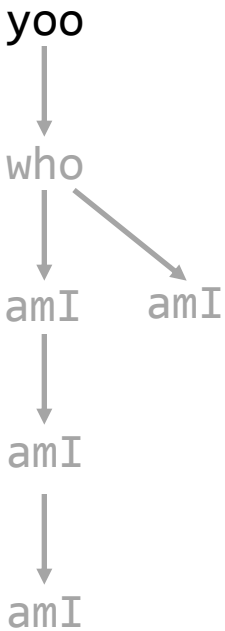


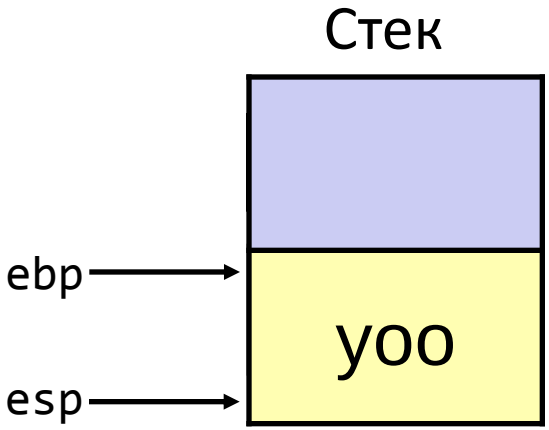






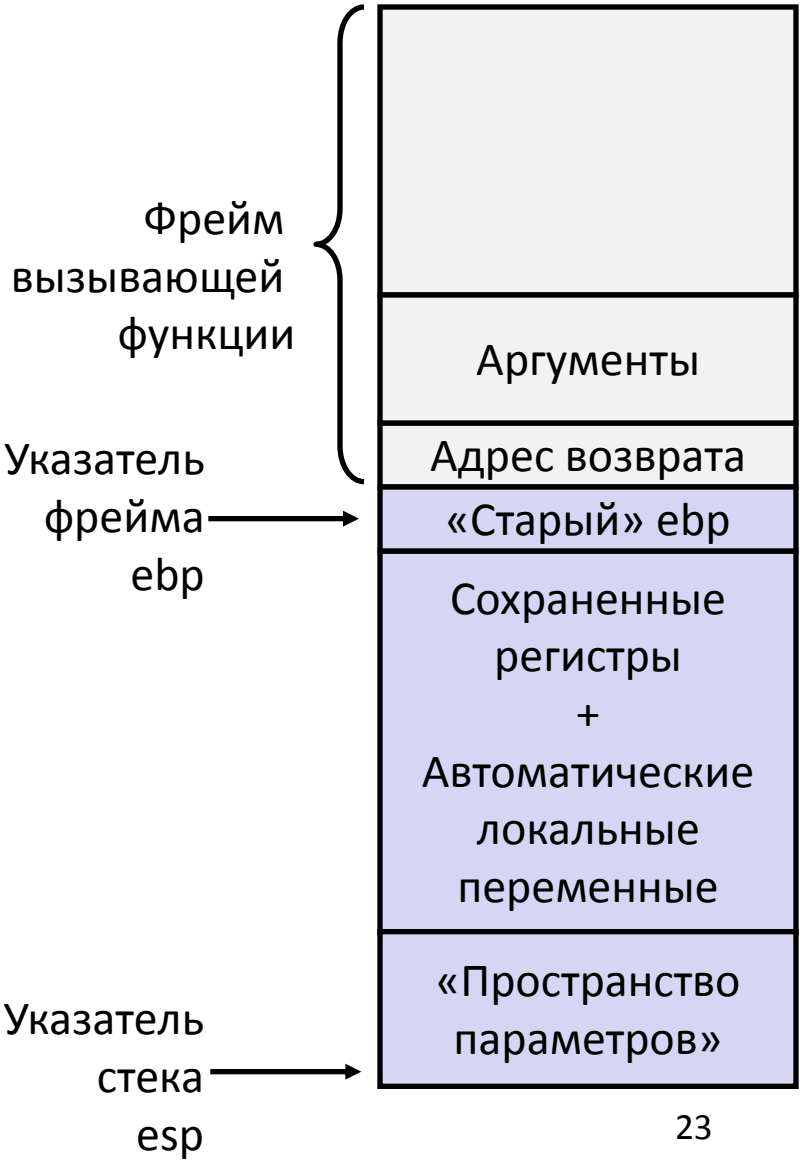
```
yoo(…)
{
   •
   •
   who();
   •
   •
}
```





Организация фрейма в IA-32/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

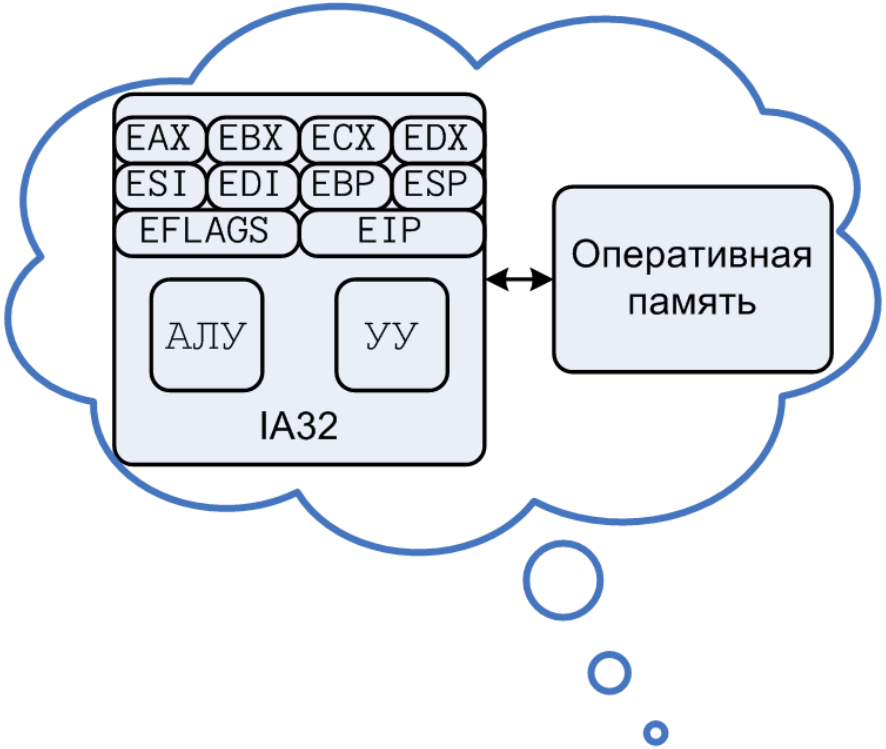


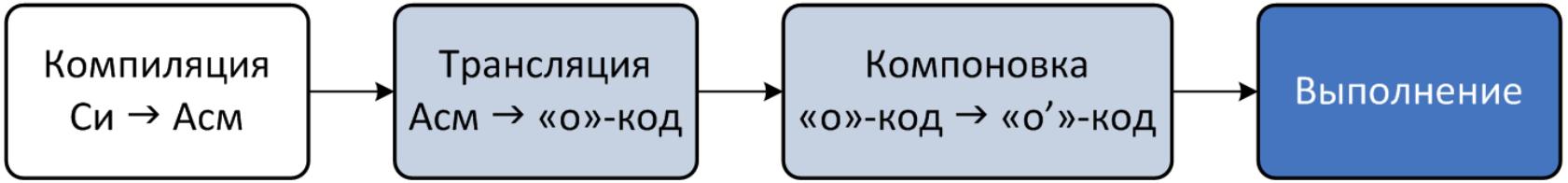
```c
int main() {
   int a = 1, b = 2, c;
   c = sum(a, b);
   return 0;
}

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)
```

Промежуточные итоги





Дальнейший материал

- Взаимосвязь языка Си, языка ассемблера и особенностей архитектуры IA32
  - Операции над целыми числами и битовыми векторами
    - «быстрая» арифметика и обработка 64 разрядных чисел
    - побитовые операции, сдвиги, вращения
  - реализация управляющих конструкций языка Си
  - адресная арифметика
  - массивы
  - структуры и объединения, выравнивание данных
  - соглашение вызова
    - cdecl, stdcall, fastcall
    - выравнивание стека
    - ускорение вызова функций
    - переменное число параметров
  - числа с плавающей точкой
  - …