DUE: OCTOBER 30TH

LAB 4: ATTACK LAB

CONTENTS

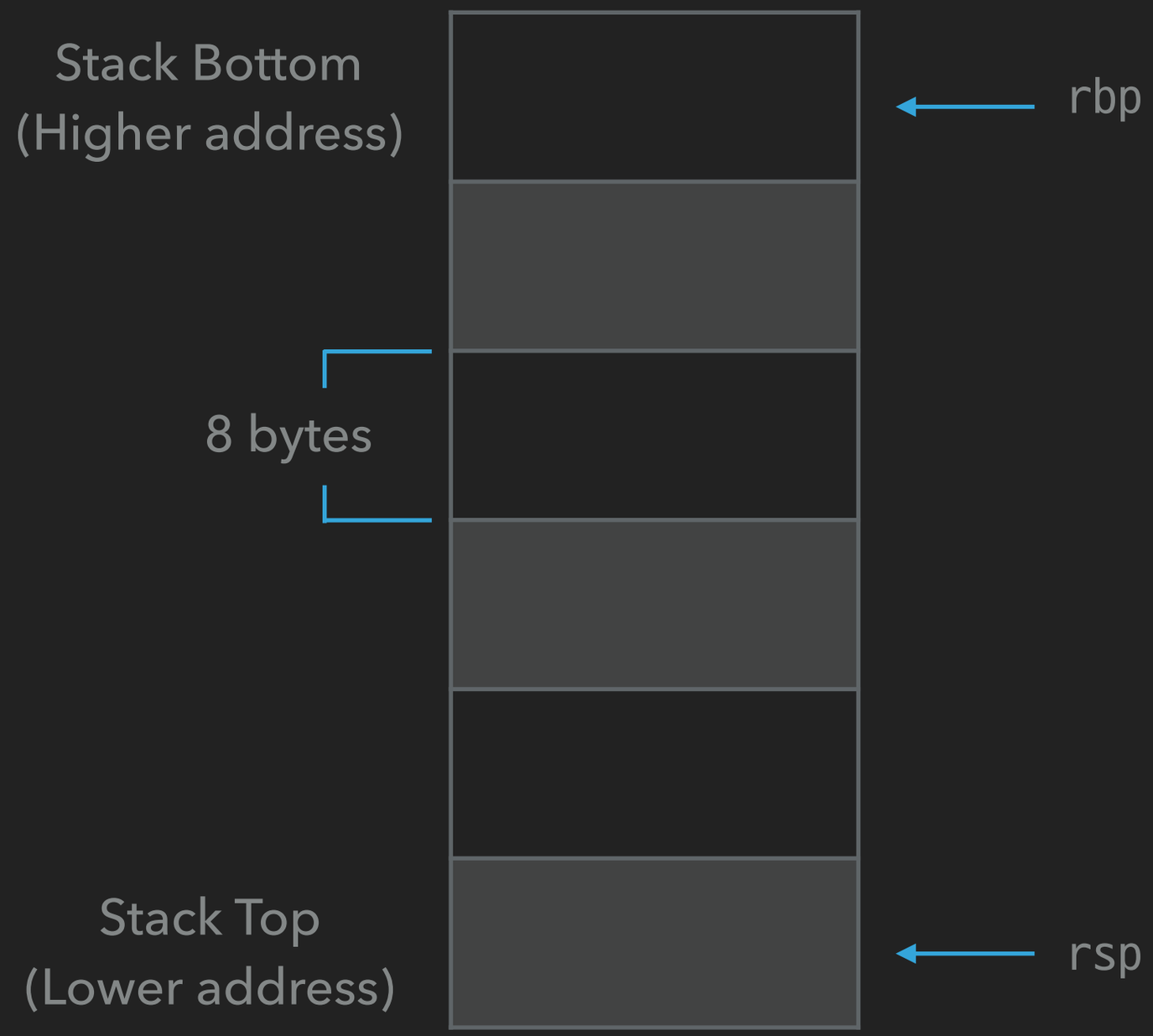
- Objective
- Stack
- Buffer Overflow
- Return Oriented Programming
- Tips

OBJECTIVE

- Attack the binary!
- Total 5 levels to attack
  - Buffer Overflow - 3 levels (`./ctarget`)
  - Return Oriented Programming - 2 levels (`./rtarget`)

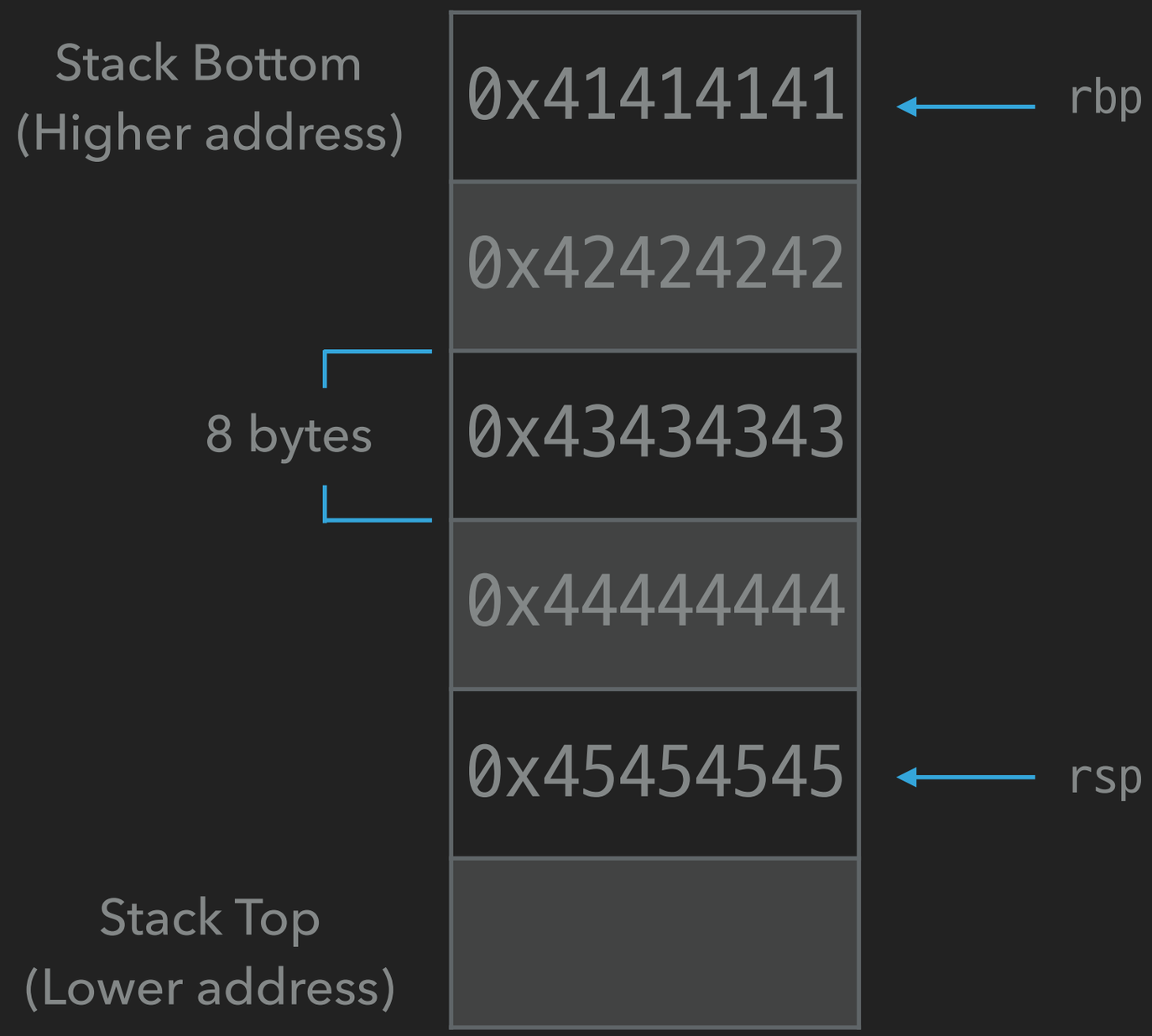
STACK

- Last-in-first-out (LIFO) area in memory that stores data
- Grows from higher to lower address



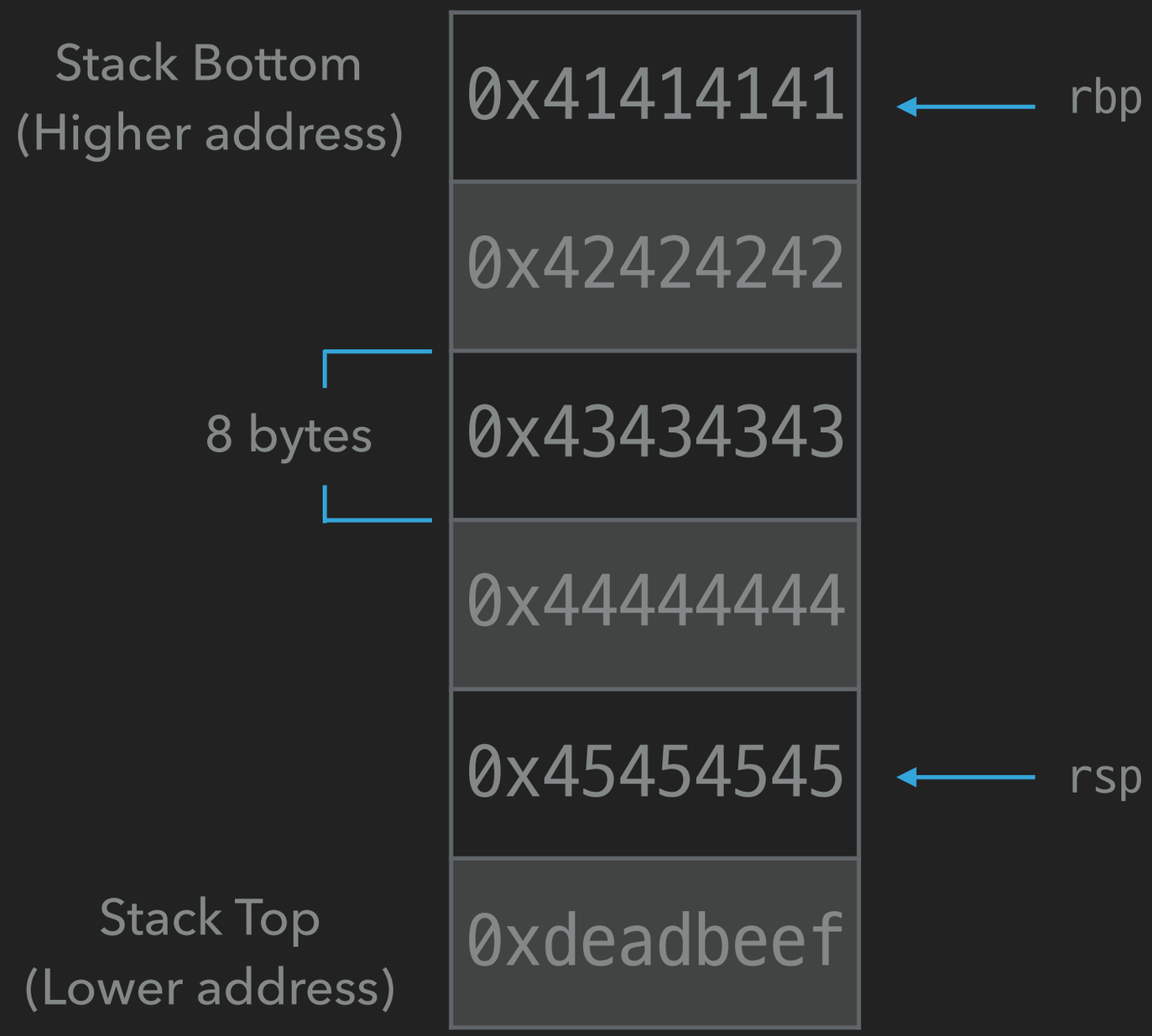
STACK

- `pushq src`
  - Store data from `src` to top of the stack
  - Decrease `rsp` by 8
- Ex) `pushq 0xdeadbeef`



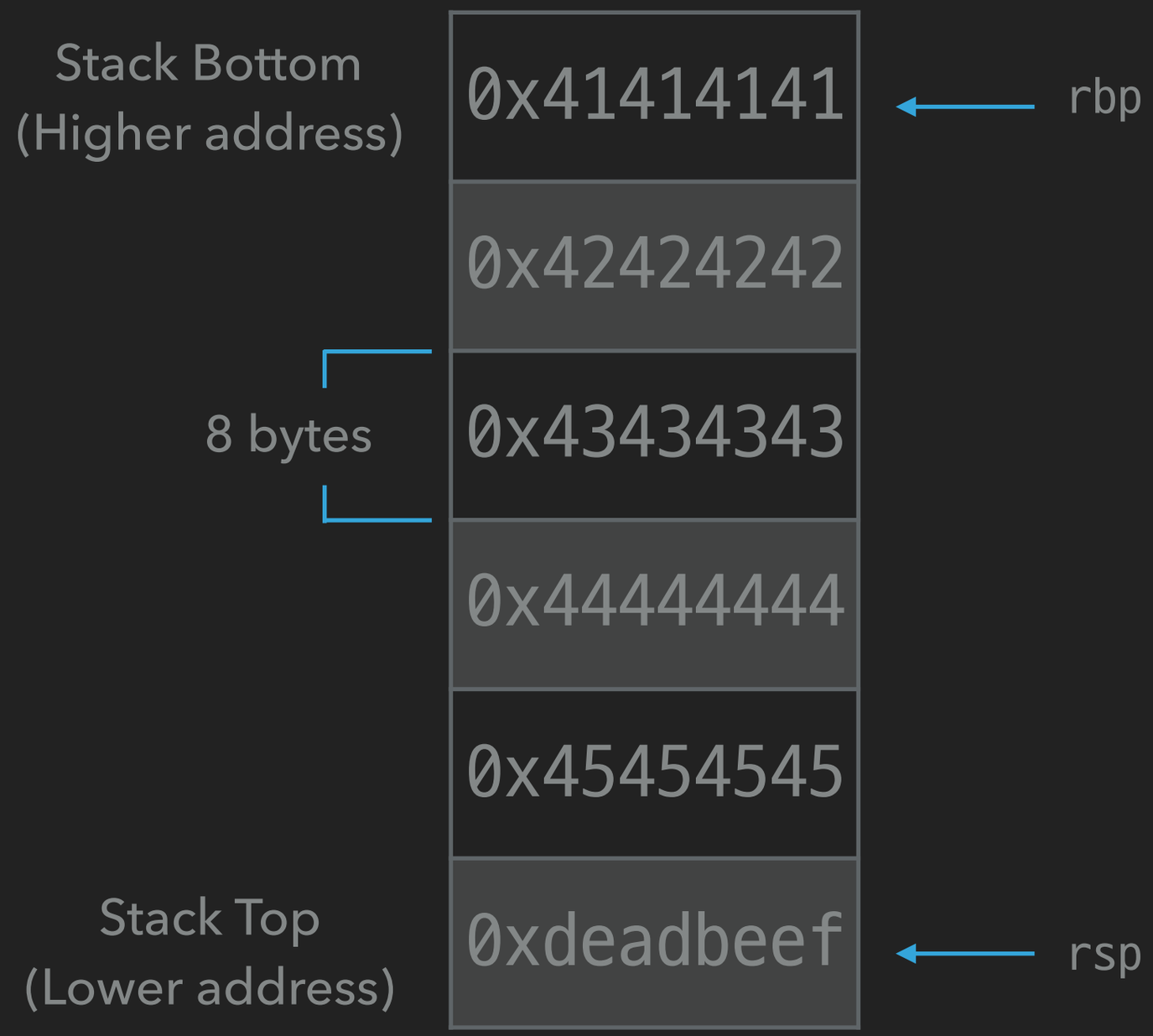
STACK

- `pushq src`
  - Store data from `src` to top of the stack
  - Decrease `rsp` by 8
- Ex) `pushq 0xdeadbeef`



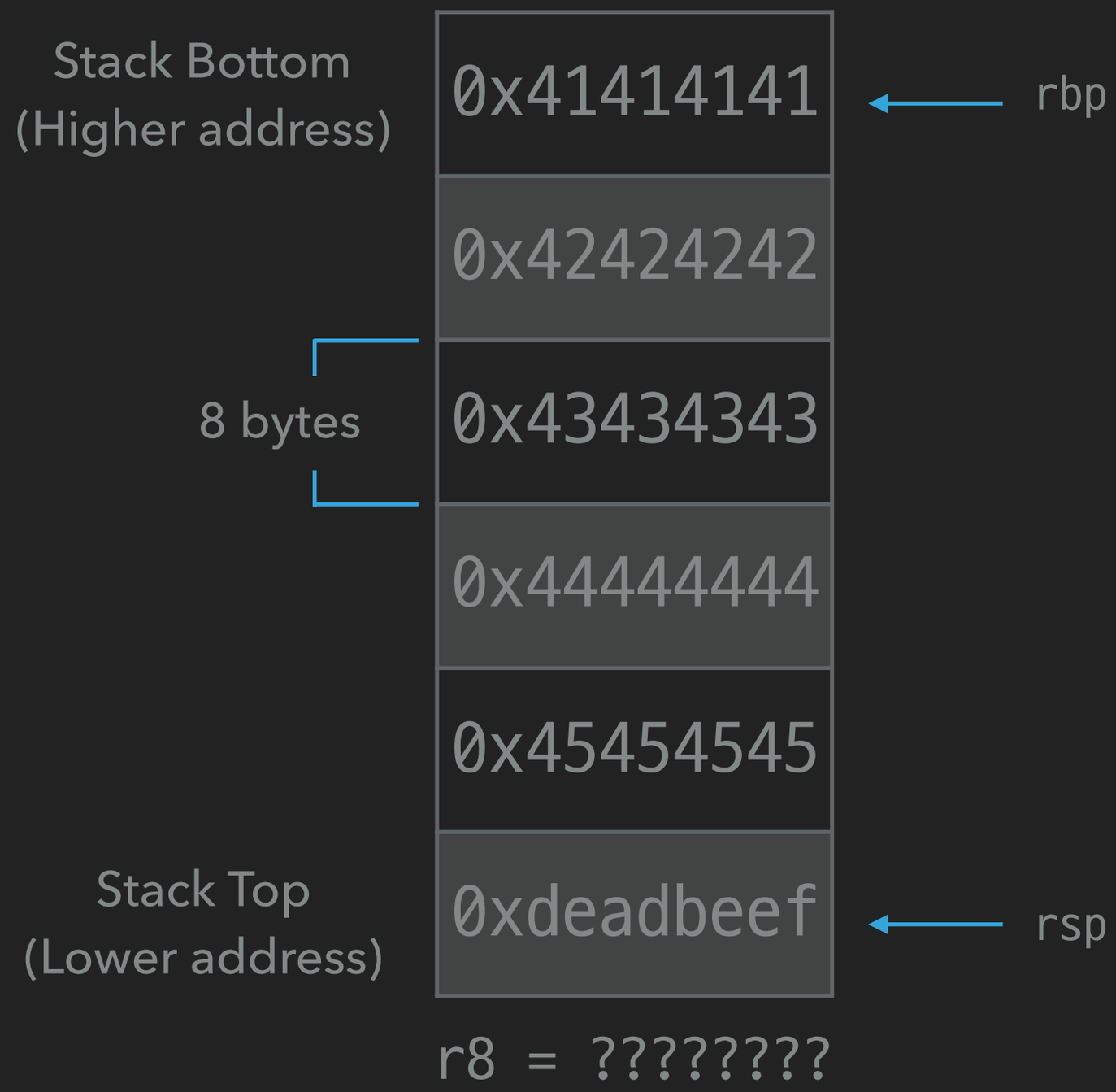
STACK

- `pushq src`
  - Store data from `src` to top of the stack
  - Decrease `rsp` by 8
- Ex) `pushq 0xdeadbeef`



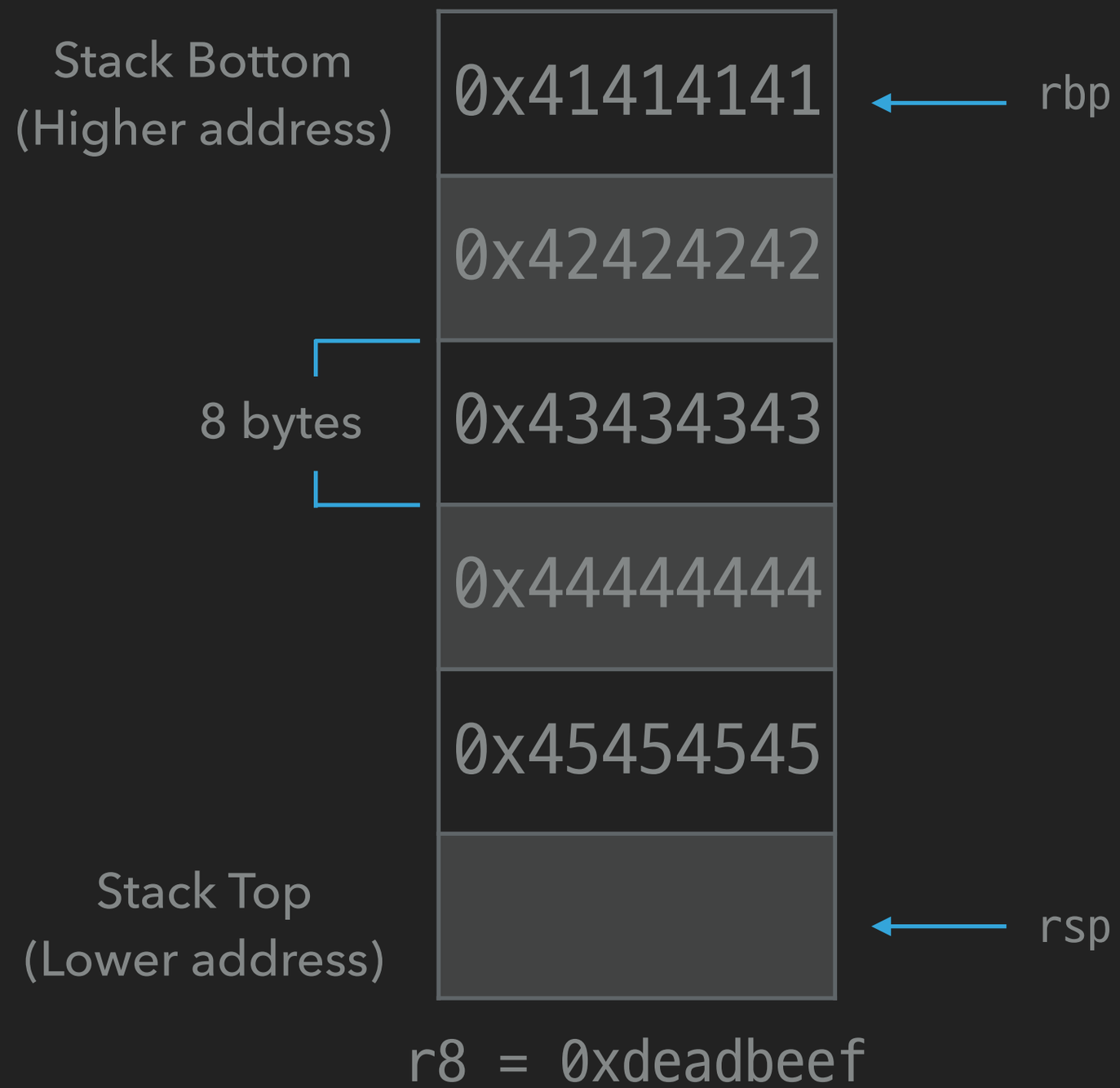
STACK

- `popq src`
  - Store data from the top of the stack to src
  - Increase `rsp` by 8
- Ex) `popq %r8`



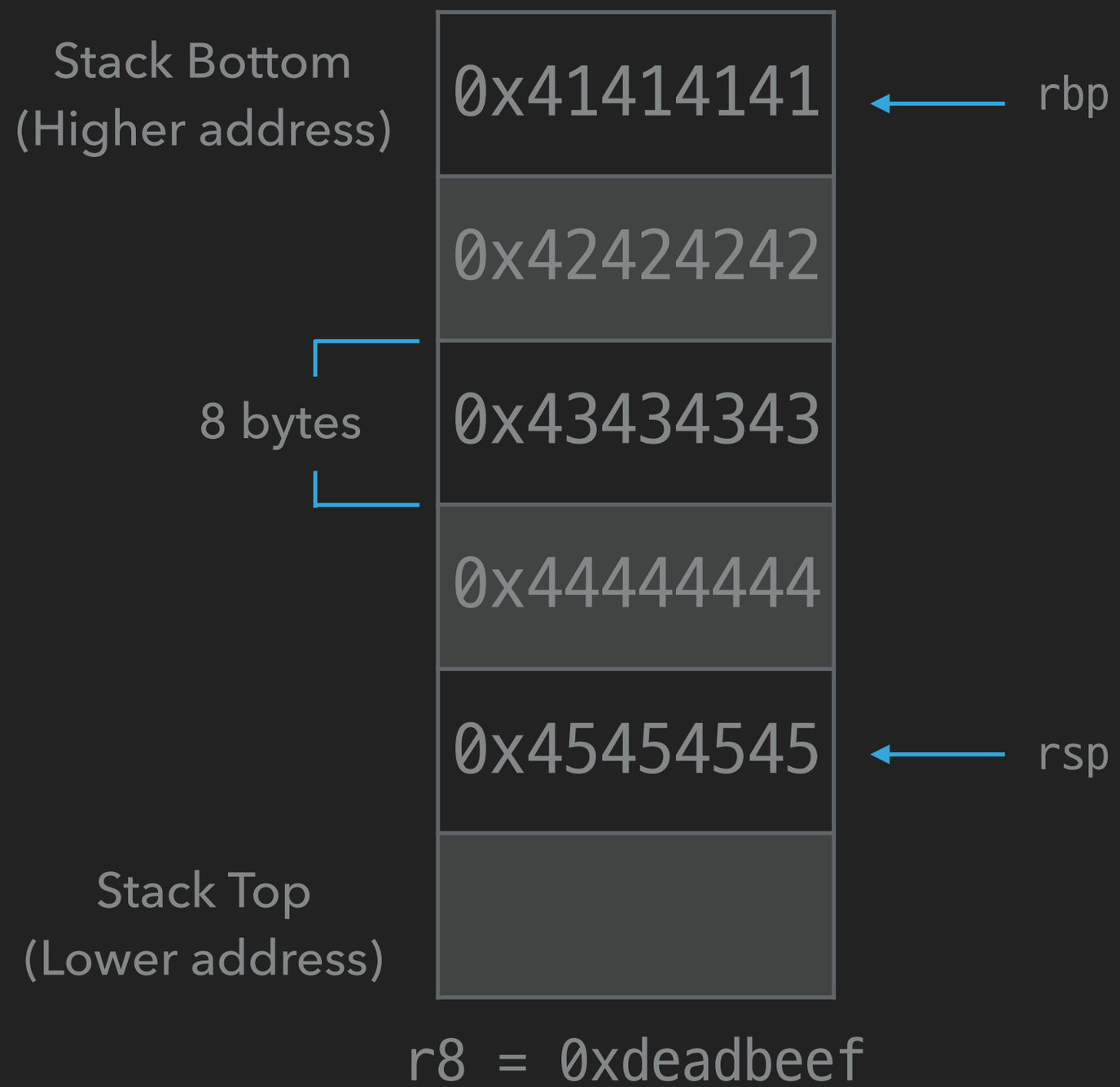
STACK

- `popq src`
  - Store data from the top of the stack to src
  - Increase `rsp` by 8
- Ex) `popq %r8`



STACK

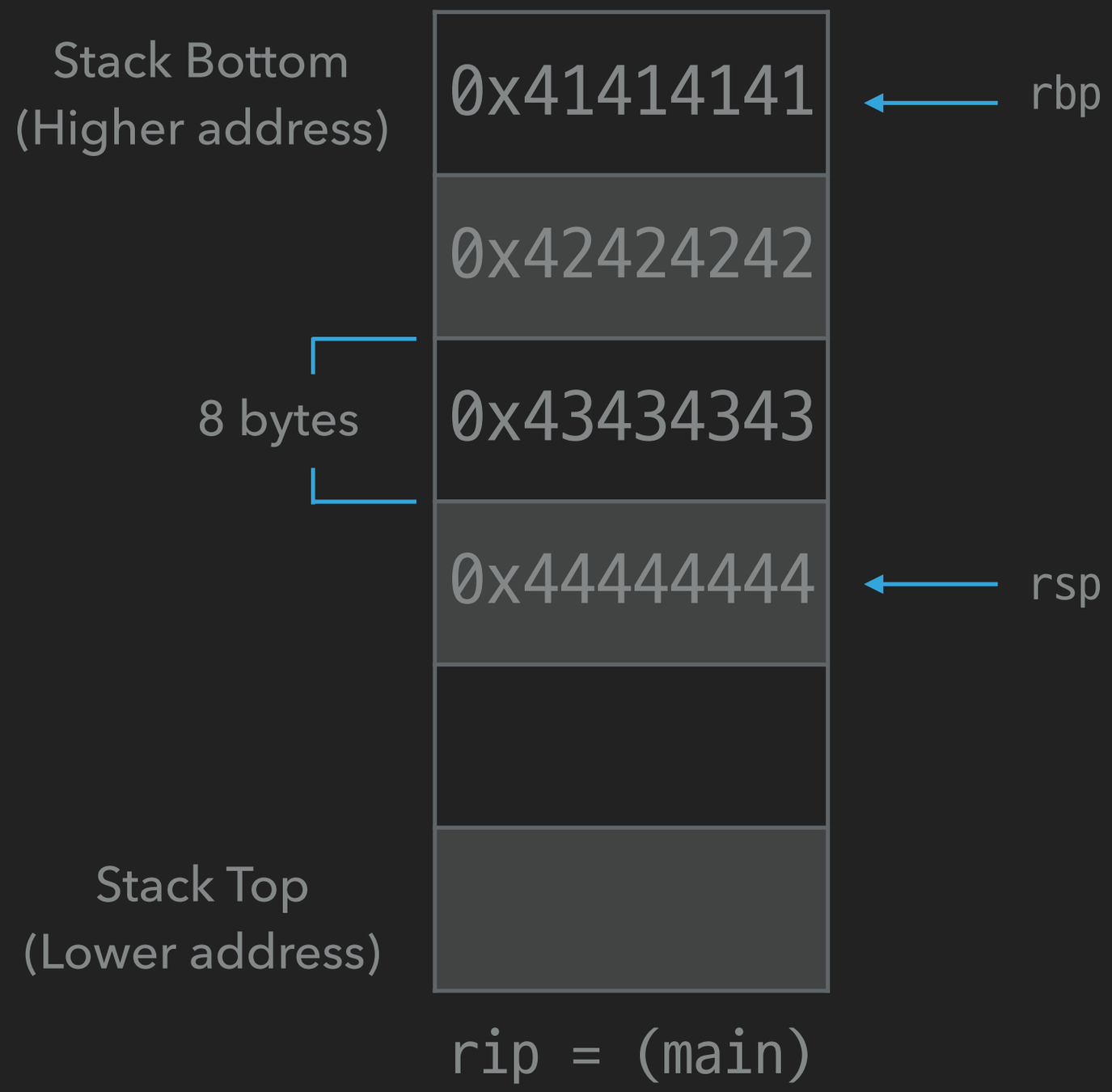
- `popq src`
  - Store data from the top of the stack to src
  - Increase `rsp` by 8
- Ex) `popq %r8`



STACK

- `call label`
  - Push return address to stack and jump to `label`
- Ex) From function `main`:

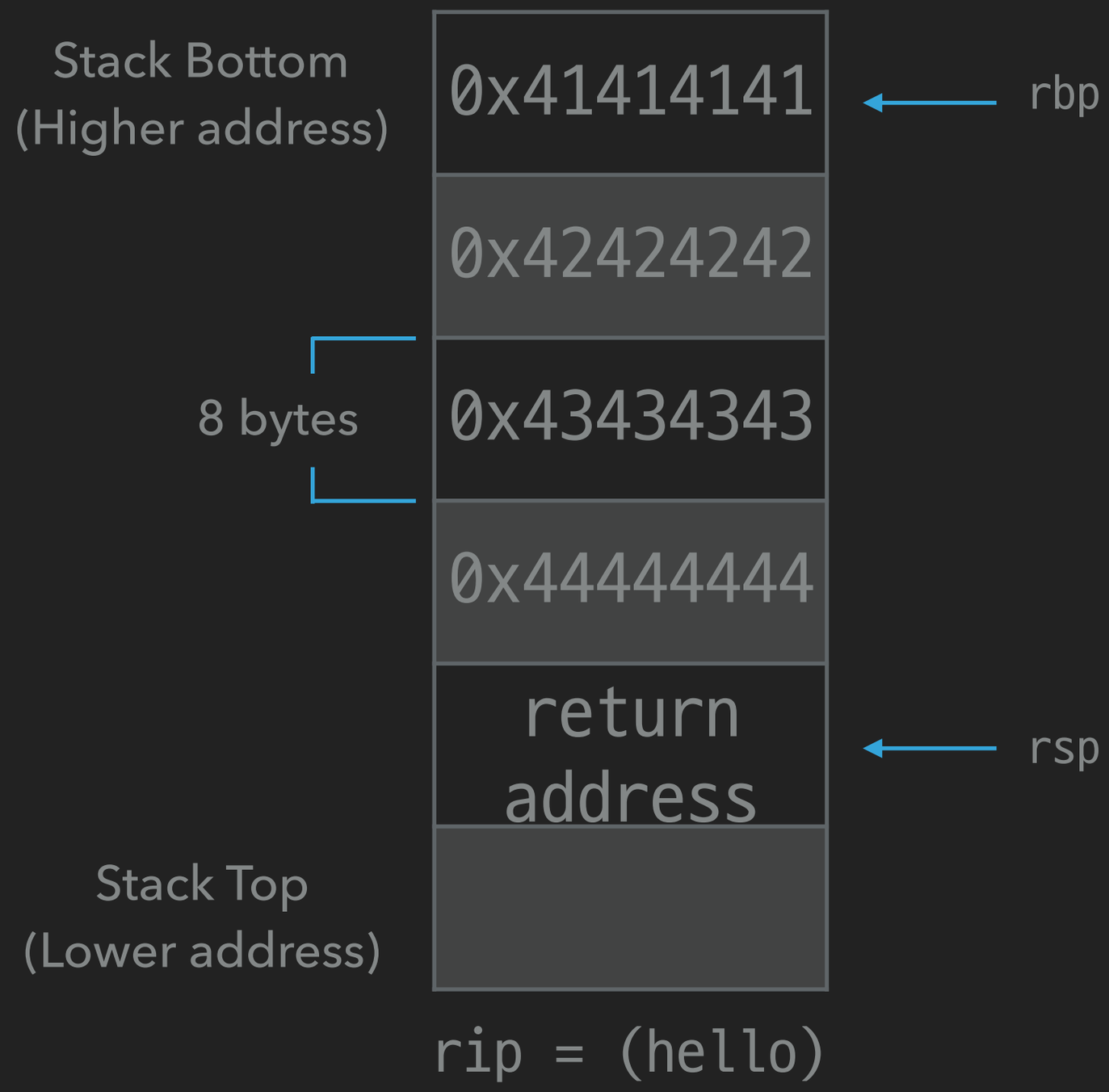
```
call hello
```



STACK

- `call label`
  - Push return address to stack and jump to `label`
- Ex) From function `main`:

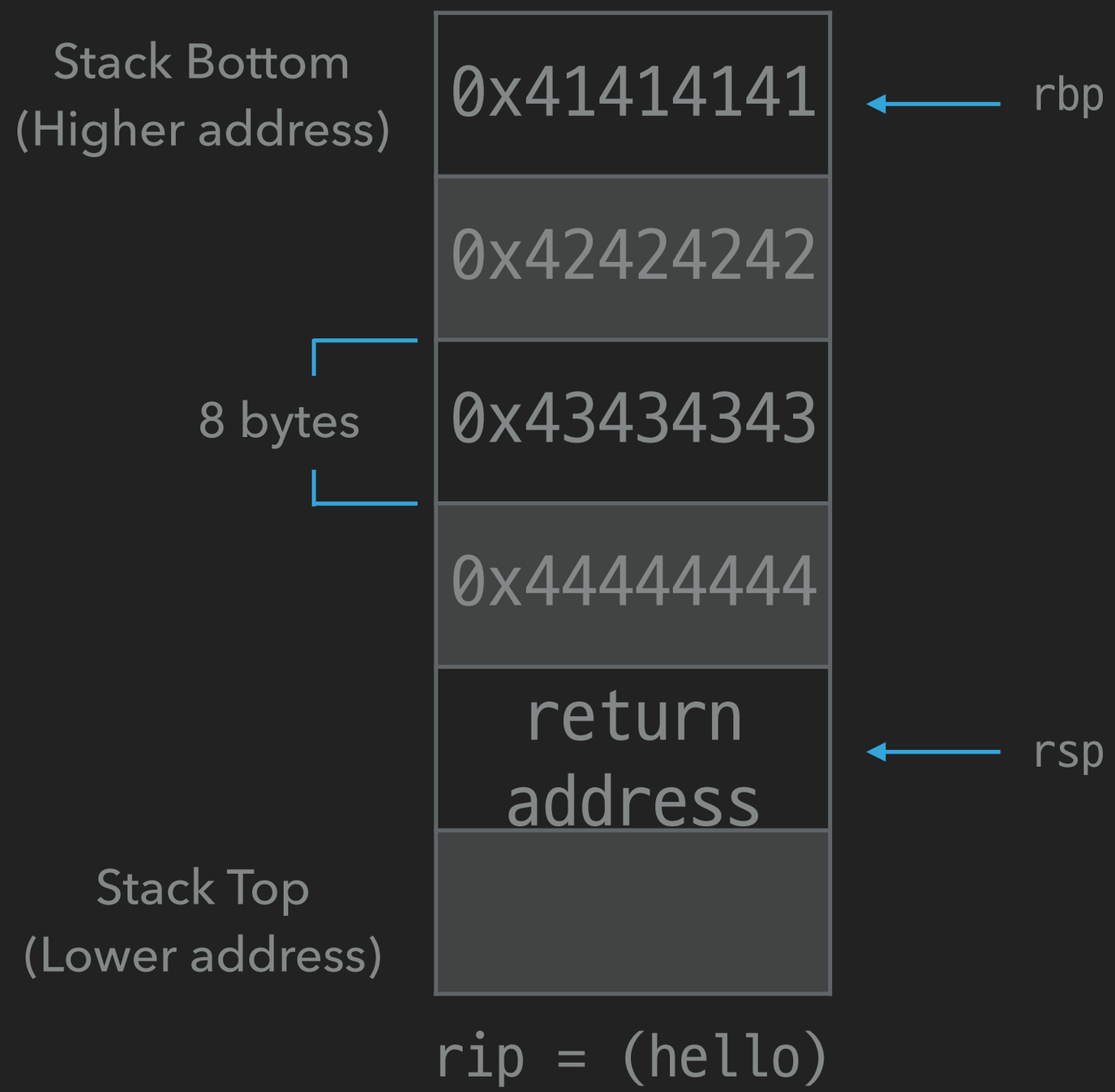
```
call hello
```



STACK

- `ret`
  - Pop return address to stack and jump to the address
- Ex) From function `hello`:

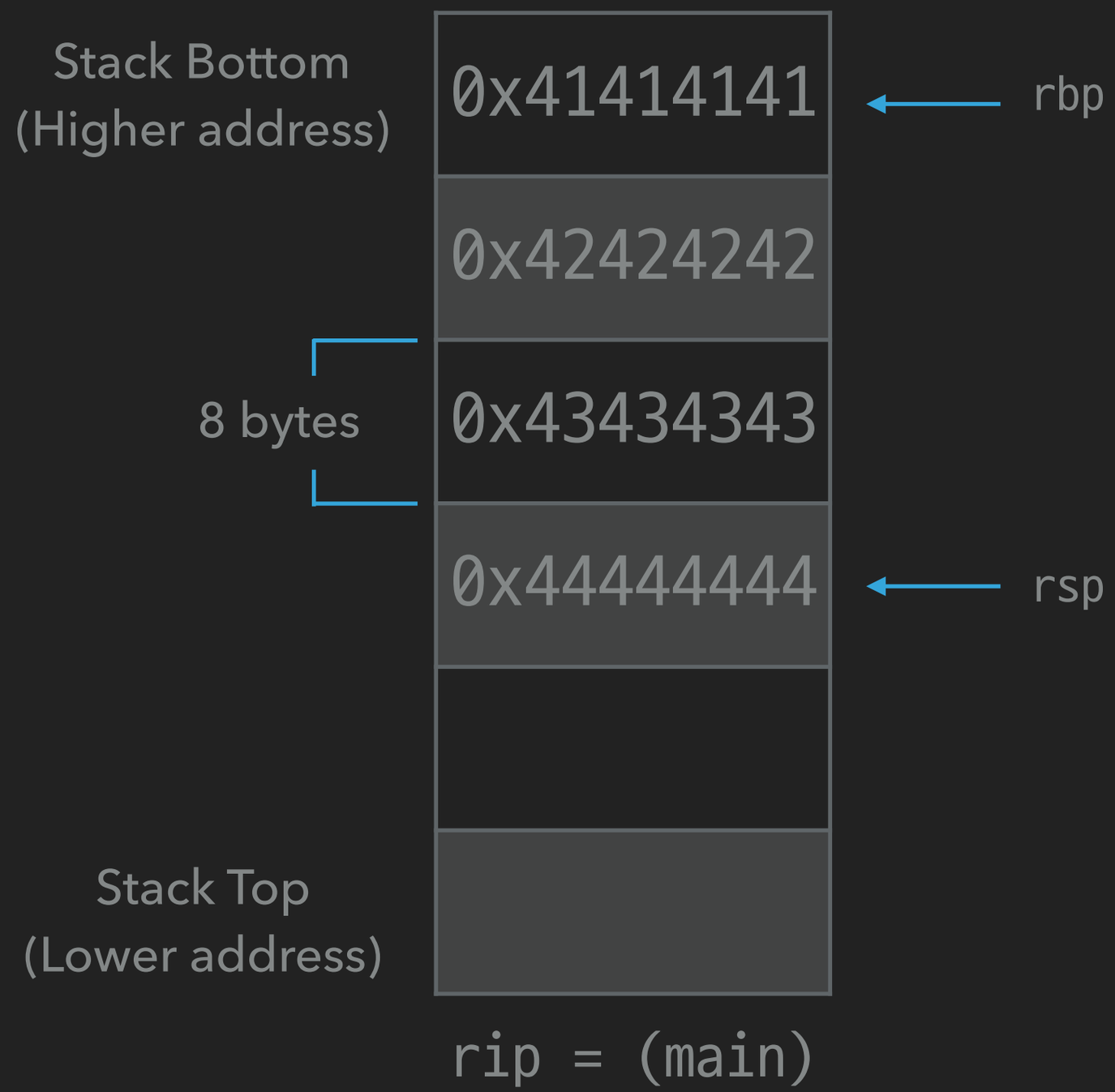
```
ret
```



STACK

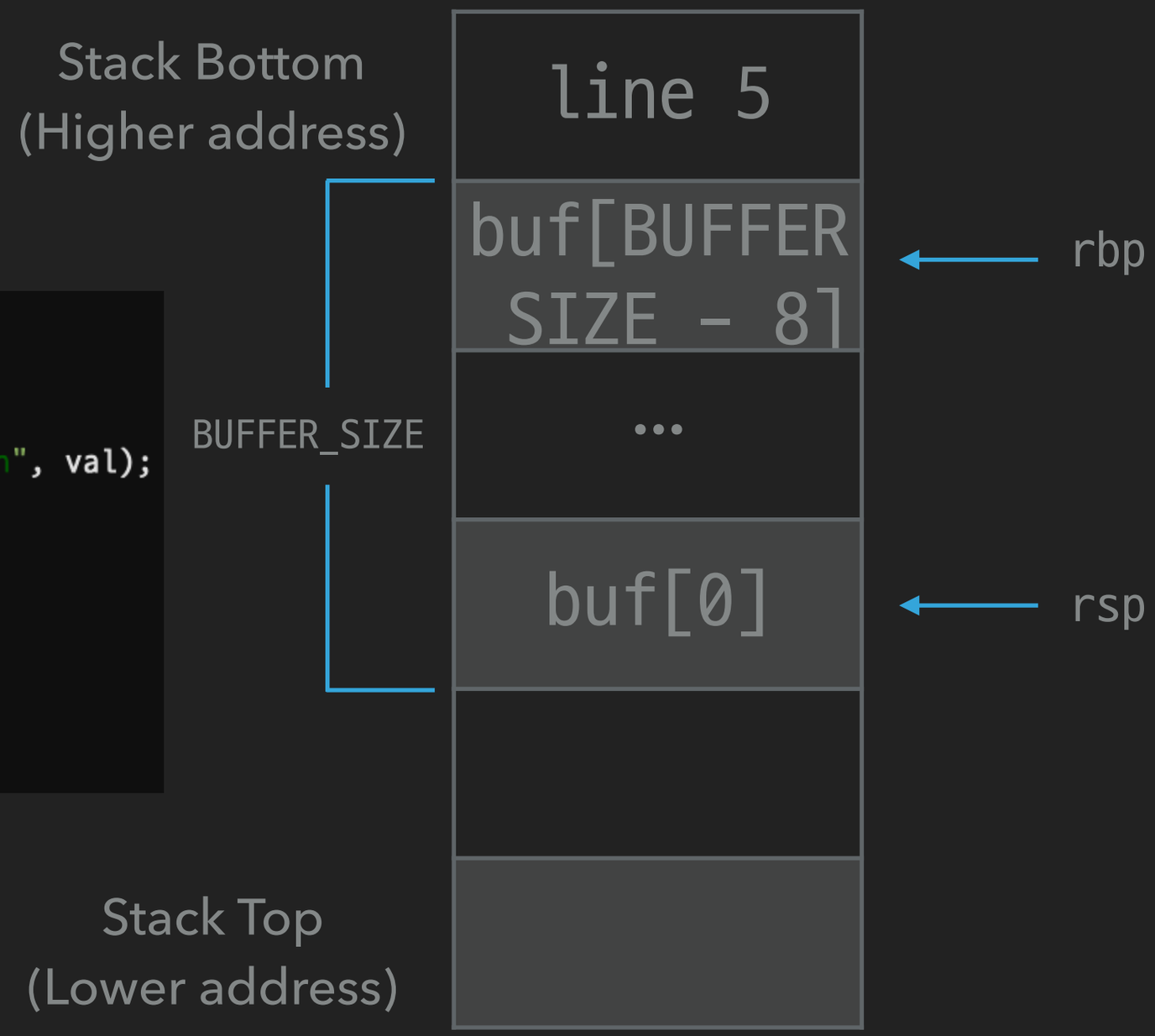
- `ret`
  - Pop return address to stack and jump to the address
- Ex) From function `hello`:

```
ret
```



BUFFER OVERFLOW

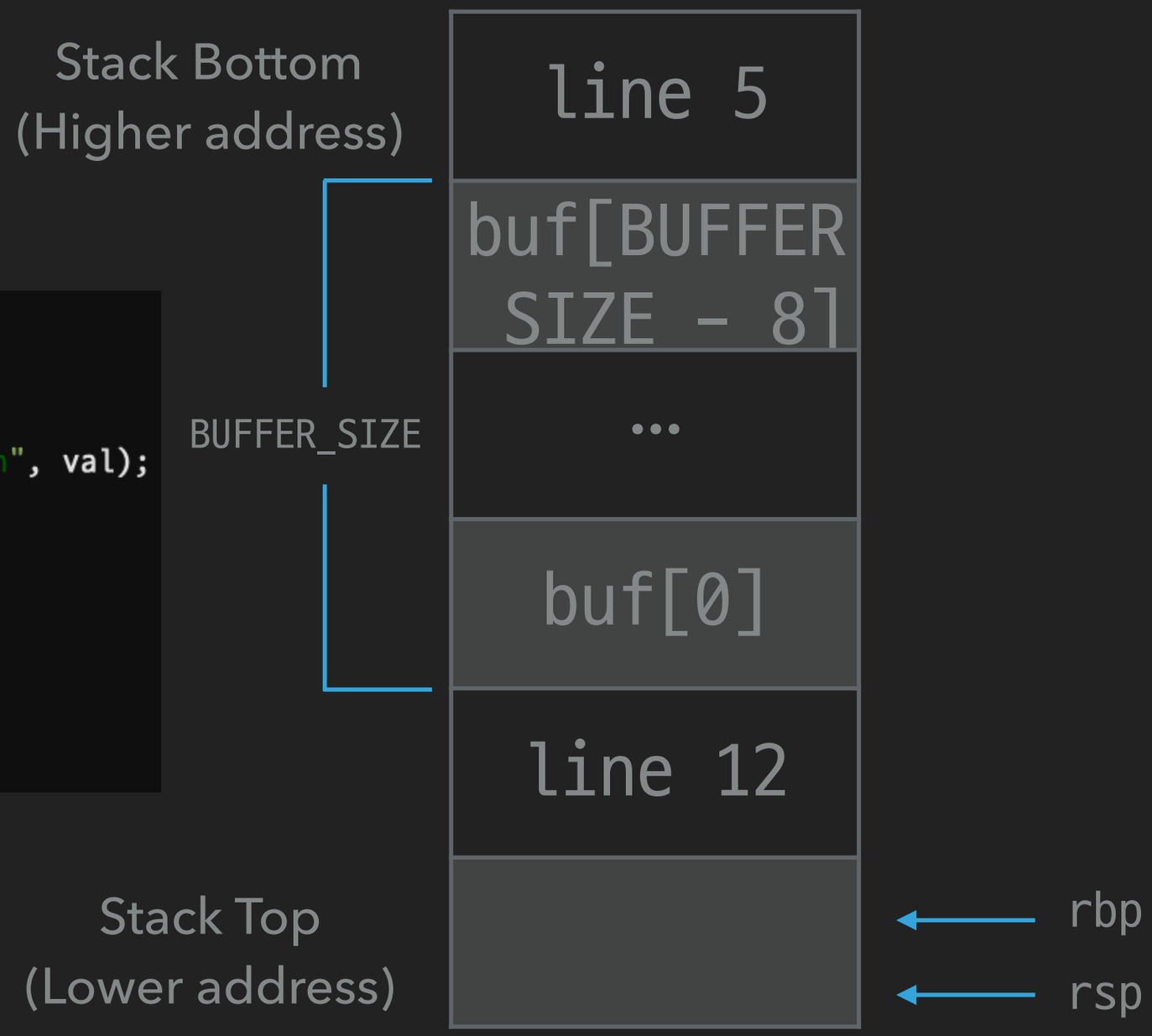
```
1  void test()
2  {
3      int val;
4      val = getbuf();
5      printf("No exploit. Getbuf returned 0x%x\n", val);
6  }
7
8  unsigned getbuf()
9  {
10     char buf[BUFFER_SIZE];
11     Gets(buf);
12     return 1;
13 }
```



This is an approximation. You need to disassemble the functions to get correct offsets.

BUFFER OVERFLOW

```
1  void test()
2  {
3      int val;
4      val = getbuf();
5      printf("No exploit. Getbuf returned 0x%x\n", val);
6  }
7
8  unsigned getbuf()
9  {
10     char buf[BUFFER_SIZE];
11     Gets(buf);
12     return 1;
13 }
```



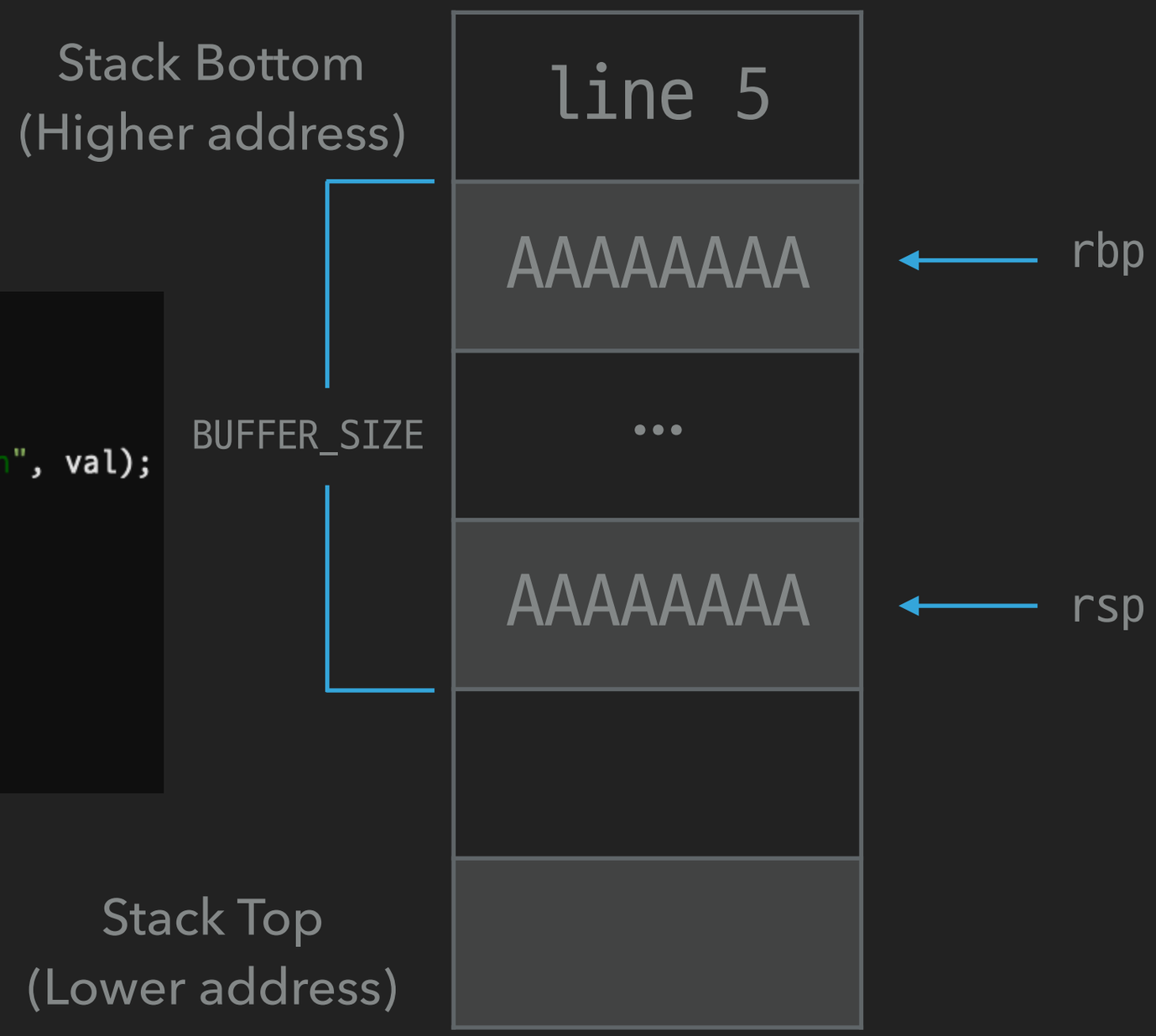
In getbuf - Calls Gets

This is an approximation. You need to disassemble the functions to get correct offsets.

BUFFER OVERFLOW

Gets: A * BUFFER_SIZE

```
1  void test()
2  {
3      int val;
4      val = getbuf();
5      printf("No exploit. Getbuf returned 0x%x\n", val);
6  }
7
8  unsigned getbuf()
9  {
10     char buf[BUFFER_SIZE];
11     Gets(buf);
12     return 1;
13 }
```



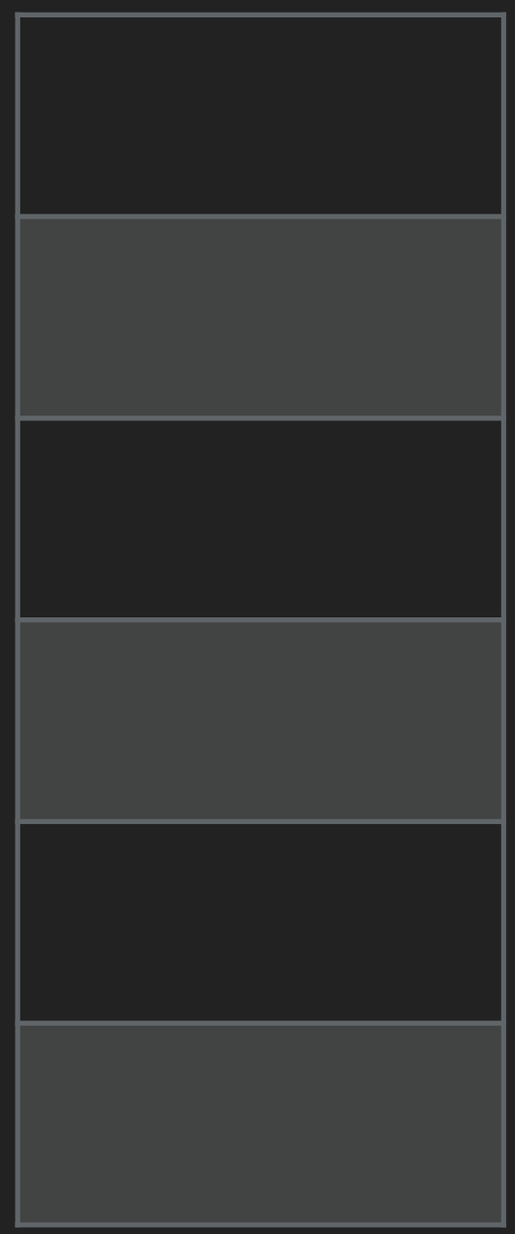
This is an approximation. You need to disassemble the functions to get correct offsets.

In getbuf - Line 12

BUFFER OVERFLOW

ret

```
1 void test()
2 {
3     int val;
4     val = getbuf();
5     printf("No exploit. Getbuf returned 0x%x\n", val);
6 }
7
8 unsigned getbuf()
9 {
10     char buf[BUFFER_SIZE];
11     Gets(buf);
12     return 1;
13 }
```



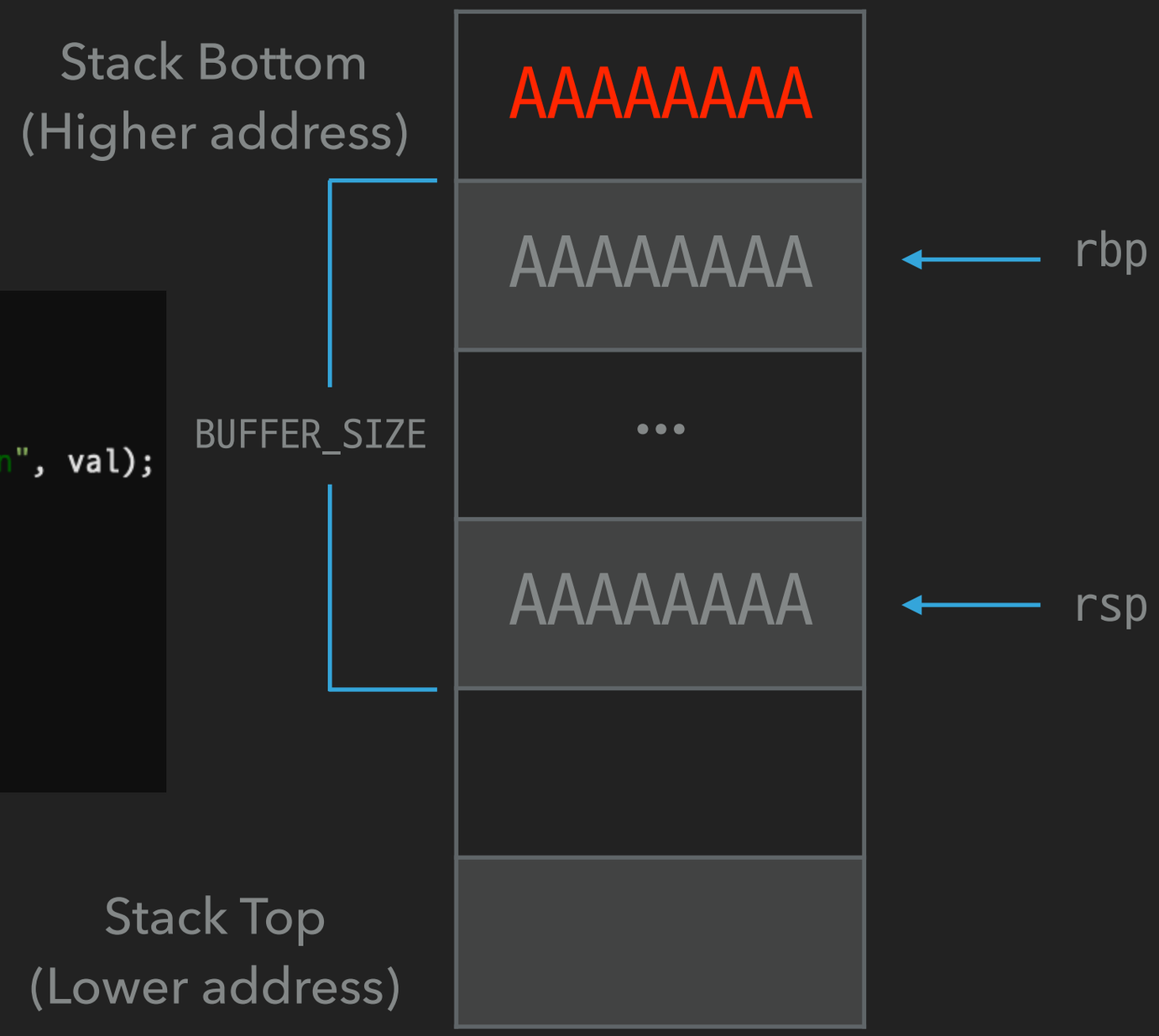
This is an approximation. You need to disassemble the functions to get correct offsets.

In test - rip: Line 5

BUFFER OVERFLOW

Gets: A *
(BUFFER_SIZE + 8)

```
1  void test()
2  {
3      int val;
4      val = getbuf();
5      printf("No exploit. Getbuf returned 0x%x\n", val);
6  }
7
8  unsigned getbuf()
9  {
10     char buf[BUFFER_SIZE];
11     Gets(buf);
12     return 1;
13 }
```



This is an approximation. You need to disassemble the functions to get correct offsets.

In getbuf - Line 12

BUFFER OVERFLOW

ret

```
1 void test()
2 {
3     int val;
4     val = getbuf();
5     printf("No exploit. Getbuf returned 0x%x\n", val);
6 }
7
8 unsigned getbuf()
9 {
10     char buf[BUFFER_SIZE];
11     Gets(buf);
12     return 1;
13 }
```

(ASCII) A == 0x41 (Hex)

This is an approximation. You need to disassemble the functions to get correct offsets.

Stack Bottom
(Higher address)

Stack top
(Lower address)

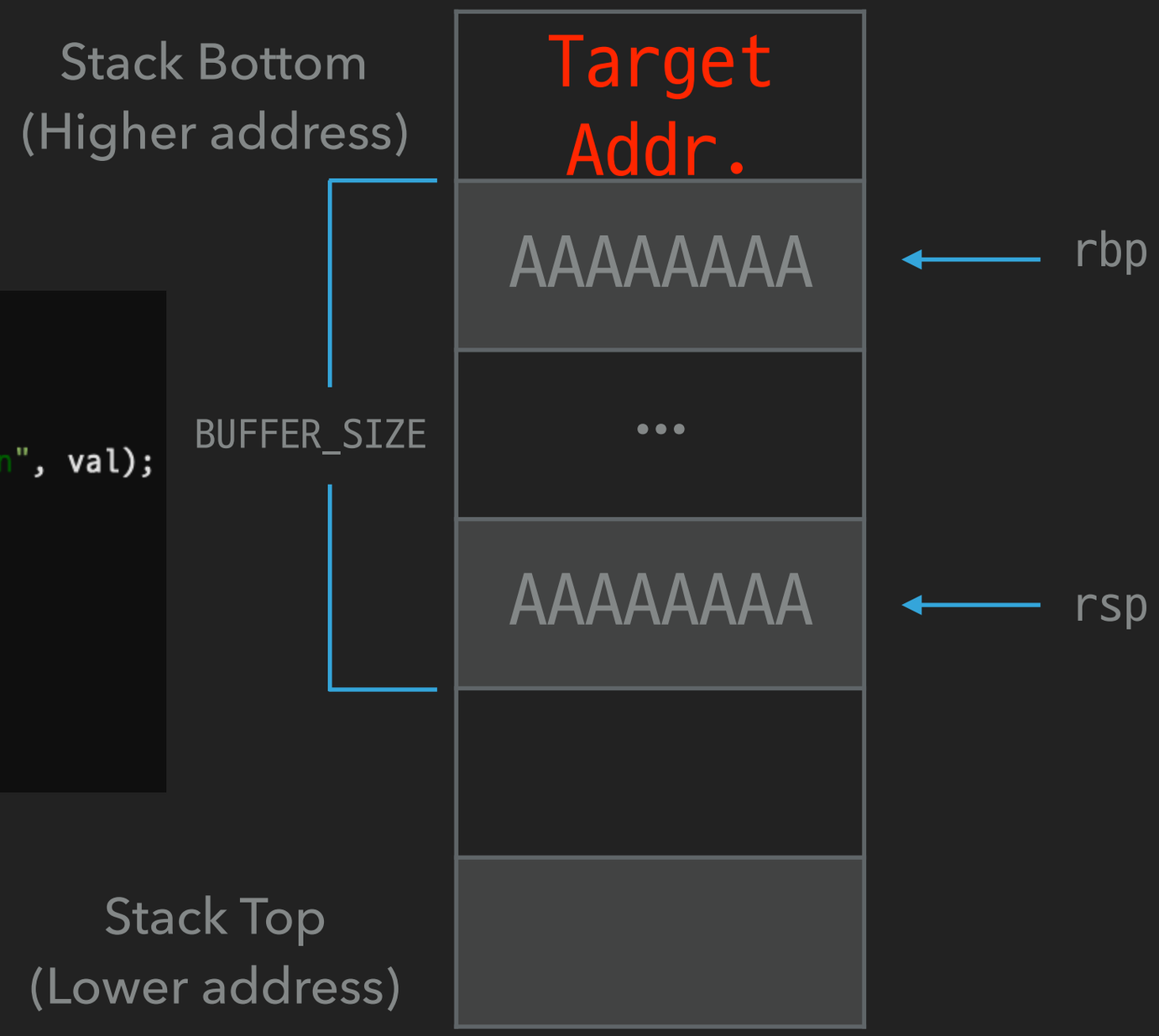
In (??????)

rip: 0x4141414141414141

BUFFER OVERFLOW

Gets: A * (BUFFER_SIZE) + (Target address)

```
1  void test()
2  {
3      int val;
4      val = getbuf();
5      printf("No exploit. Getbuf returned 0x%x\n", val);
6  }
7
8  unsigned getbuf()
9  {
10     char buf[BUFFER_SIZE];
11     Gets(buf);
12     return 1;
13 }
```



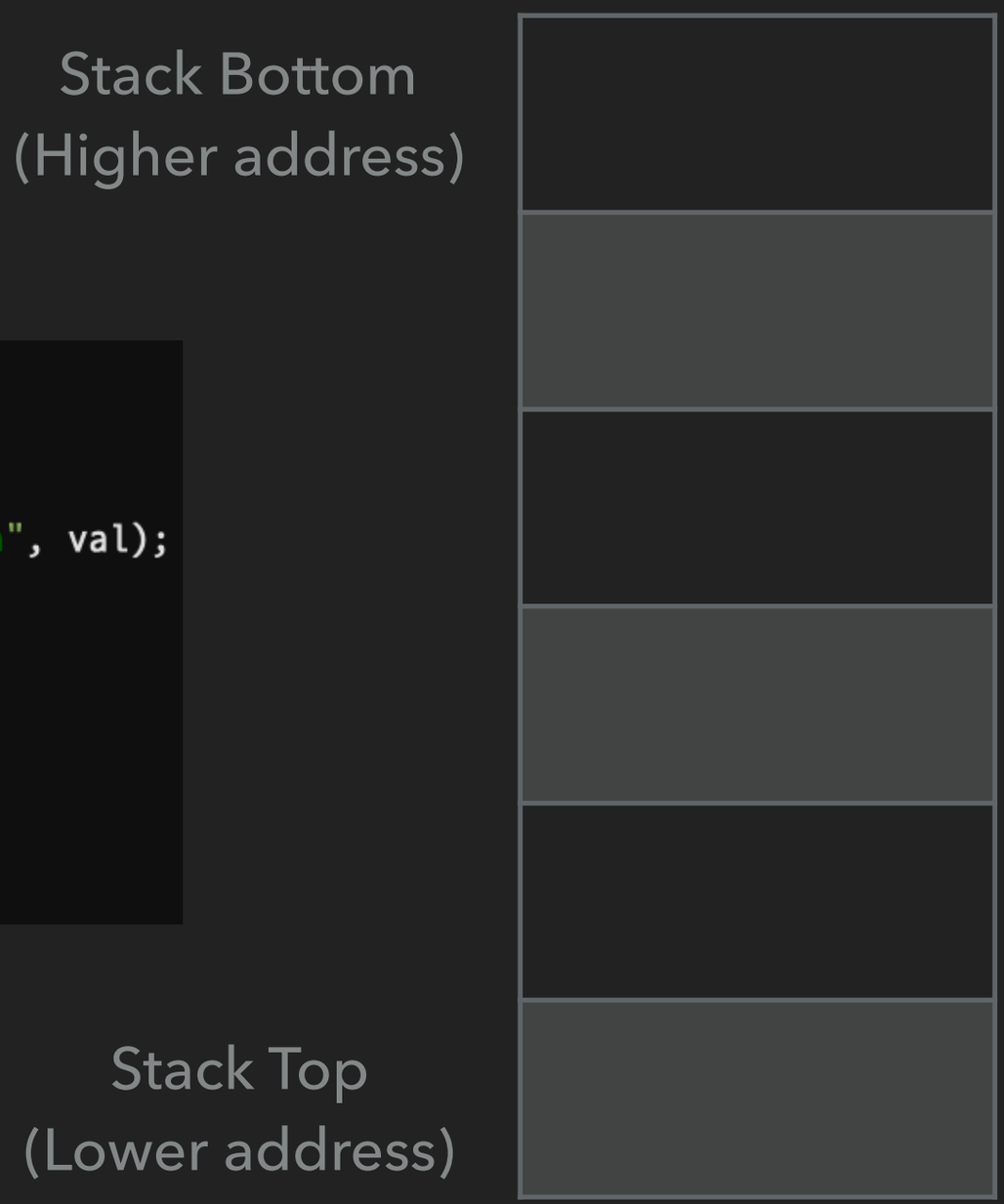
This is an approximation. You need to disassemble the functions to get correct offsets.

BUFFER OVERFLOW

ret

```
1 void test()
2 {
3     int val;
4     val = getbuf();
5     printf("No exploit. Getbuf returned 0x%x\n", val);
6 }
7
8 unsigned getbuf()
9 {
10     char buf[BUFFER_SIZE];
11     Gets(buf);
12     return 1;
13 }
```

This is an approximation. You need to disassemble the functions to get correct offsets.



BUFFER OVERFLOW

- Why does it work?
  - No boundary checks!
- Prevention?
  - Address space layout randomization (ASLR)
  - Non-eXecutable stack (NX)

BUFFER OVERFLOW

- ./ctarget: Executable stack (x present in stack permission)

```
ta@canis01:~/target1$ ps aux | grep ctarget
ta        8333  0.0  0.0   5372   872 pts/10   S+   22:03   0:00 ./ctarget
ta        8341  0.0  0.0  11748   908 pts/27   S+   22:03   0:00 grep --color=auto ctarget
ta@canis01:~/target1$ cat /proc/8333/maps
00400000-00404000 r-xp 00000000 08:01 55445836                           /home/ta/target1/ctarget
00603000-00604000 r--p 00003000 08:01 55445836                           /home/ta/target1/ctarget
00604000-00605000 rw-p 00004000 08:01 55445836                           /home/ta/target1/ctarget
00605000-00606000 rw-p 00000000 00:00 0
01d41000-01d62000 rw-p 00000000 00:00 0                                  [heap]
55586000-55686000 rwxp 00000000 00:00 0                                  [stack:8333]
7fb05c258000-7fb05c416000 r-xp 00000000 08:01 39323766                   /lib/x86_64-linux-gnu/libc-2.19.so
7fb05c416000-7fb05c616000 ---p 001be000 08:01 39323766                   /lib/x86_64-linux-gnu/libc-2.19.so
7fb05c616000-7fb05c61a000 r--p 001be000 08:01 39323766                   /lib/x86_64-linux-gnu/libc-2.19.so
7fb05c61a000-7fb05c61c000 rw-p 001c2000 08:01 39323766                   /lib/x86_64-linux-gnu/libc-2.19.so
7fb05c61c000-7fb05c621000 rw-p 00000000 00:00 0
7fb05c621000-7fb05c644000 r-xp 00000000 08:01 39323754                   /lib/x86_64-linux-gnu/ld-2.19.so
7fb05c826000-7fb05c829000 rw-p 00000000 00:00 0
7fb05c841000-7fb05c843000 rw-p 00000000 00:00 0
7fb05c843000-7fb05c844000 r--p 00022000 08:01 39323754                   /lib/x86_64-linux-gnu/ld-2.19.so
7fb05c844000-7fb05c845000 rw-p 00023000 08:01 39323754                   /lib/x86_64-linux-gnu/ld-2.19.so
7fb05c845000-7fb05c846000 rw-p 00000000 00:00 0
7ffd0f989000-7ffd0f9aa000 rw-p 00000000 00:00 0
7ffd0f9e2000-7ffd0f9e4000 r-xp 00000000 00:00 0                          [vdso]
ffffffffff600000-ffffffffff601000 r-xp 00000000 00:00 0                  [vsyscall]
```

BUFFER OVERFLOW

- ./ctarget: Address space not randomized

```
ta@canis01:~/target1$ ps aux | grep ctarget
ta        8403  0.0  0.0   5372   868 pts/10   S+   22:05   0:00 ./ctarget
ta        8406  0.0  0.0  11748   908 pts/27   S+   22:05   0:00 grep --color=auto ctarget
ta@canis01:~/target1$ cat /proc/8403/maps
00400000-00404000 r-xp 00000000 08:01 55445836                           /home/ta/target1/ctarget
00603000-00604000 r--p 00003000 08:01 55445836                           /home/ta/target1/ctarget
00604000-00605000 rw-p 00004000 08:01 55445836                           /home/ta/target1/ctarget
00605000-00606000 rw-p 00000000 00:00 0
0204a000-0206b000 rw-p 00000000 00:00 0                                  [heap]
55586000-55686000 rwxp 00000000 00:00 0                                  [stack:8403]
7f5db63f5000-7f5db65b3000 r-xp 00000000 08:01 39323766                   /lib/x86_64-linux-gnu/libc-2.19.so
7f5db65b3000-7f5db67b3000 ---p 001be000 08:01 39323766                   /lib/x86_64-linux-gnu/libc-2.19.so
7f5db67b3000-7f5db67b7000 r--p 001be000 08:01 39323766                   /lib/x86_64-linux-gnu/libc-2.19.so
7f5db67b7000-7f5db67b9000 rw-p 001c2000 08:01 39323766                   /lib/x86_64-linux-gnu/libc-2.19.so
7f5db67b9000-7f5db67be000 rw-p 00000000 00:00 0
7f5db67be000-7f5db67e1000 r-xp 00000000 08:01 39323754                   /lib/x86_64-linux-gnu/ld-2.19.so
7f5db69c3000-7f5db69c6000 rw-p 00000000 00:00 0
7f5db69de000-7f5db69e0000 rw-p 00000000 00:00 0
7f5db69e0000-7f5db69e1000 r--p 00022000 08:01 39323754                   /lib/x86_64-linux-gnu/ld-2.19.so
7f5db69e1000-7f5db69e2000 rw-p 00023000 08:01 39323754                   /lib/x86_64-linux-gnu/ld-2.19.so
7f5db69e2000-7f5db69e3000 rw-p 00000000 00:00 0
7fff8a60b000-7fff8a62c000 rw-p 00000000 00:00 0
7fff8a7e0000-7fff8a7e2000 r-xp 00000000 00:00 0                          [vdso]
ffffffffff600000-ffffffffff601000 r-xp 00000000 00:00 0                  [vsyscall]
```

BUFFER OVERFLOW

- ./rtarget: Stack not executable (x not present in stack permission)

```
ta@canis01:~/target1$ ps aux | grep rtarget
ta        8468  0.0  0.0   4412   608 pts/10   S+   22:09   0:00 ./rtarget
ta        8474  0.0  0.0  11748   904 pts/27   S+   22:09   0:00 grep --color=auto rtarget
ta@canis01:~/target1$ cat /proc/8468/maps
00400000-00405000 r-xp 00000000 08:01 55445837                           /home/ta/target1/rtarget
00604000-00605000 r--p 00004000 08:01 55445837                           /home/ta/target1/rtarget
00605000-00606000 rw-p 00005000 08:01 55445837                           /home/ta/target1/rtarget
00606000-00607000 rw-p 00000000 00:00 0
0091f000-00940000 rw-p 00000000 00:00 0                                  [heap]
7f93c317a000-7f93c3338000 r-xp 00000000 08:01 39323766                   /lib/x86_64-linux-gnu/libc-2.19.so
7f93c3338000-7f93c3538000 ---p 001be000 08:01 39323766                   /lib/x86_64-linux-gnu/libc-2.19.so
7f93c3538000-7f93c353c000 r--p 001be000 08:01 39323766                   /lib/x86_64-linux-gnu/libc-2.19.so
7f93c353c000-7f93c353e000 rw-p 001c2000 08:01 39323766                   /lib/x86_64-linux-gnu/libc-2.19.so
7f93c353e000-7f93c3543000 rw-p 00000000 00:00 0
7f93c3543000-7f93c3566000 r-xp 00000000 08:01 39323754                   /lib/x86_64-linux-gnu/ld-2.19.so
7f93c3748000-7f93c374b000 rw-p 00000000 00:00 0
7f93c3763000-7f93c3765000 rw-p 00000000 00:00 0
7f93c3765000-7f93c3766000 r--p 00022000 08:01 39323754                   /lib/x86_64-linux-gnu/ld-2.19.so
7f93c3766000-7f93c3767000 rw-p 00023000 08:01 39323754                   /lib/x86_64-linux-gnu/ld-2.19.so
7f93c3767000-7f93c3768000 rw-p 00000000 00:00 0
7ffd3b27b000-7ffd3b2ab000 rw-p 00000000 00:00 0                          [stack]
7ffd3b3c7000-7ffd3b3c9000 r-xp 00000000 00:00 0                          [vdso]
ffffffffff600000-ffffffffff601000 r-xp 00000000 00:00 0                  [vsyscall]
```

BUFFER OVERFLOW

- ./rtarget: Address space randomized

```
ta@canis01:~/target1$ ps aux ¦ grep rtarget
ta        8509  0.0  0.0   4640   868 pts/10   S+   22:10   0:00 ./rtarget
ta        8512  0.0  0.0  11748   904 pts/27   S+   22:10   0:00 grep --color=auto rtarget
ta@canis01:~/target1$ cat /proc/8509/maps
00400000-00405000 r-xp 00000000 08:01 55445837                           /home/ta/target1/rtarget
00604000-00605000 r--p 00004000 08:01 55445837                           /home/ta/target1/rtarget
00605000-00606000 rw-p 00005000 08:01 55445837                           /home/ta/target1/rtarget
00606000-00607000 rw-p 00000000 00:00 0
02382000-023a3000 rw-p 00000000 00:00 0                                  [heap]
7f4644ceb000-7f4644ea9000 r-xp 00000000 08:01 39323766                   /lib/x86_64-linux-gnu/libc-2.19.so
7f4644ea9000-7f46450a9000 ---p 001be000 08:01 39323766                   /lib/x86_64-linux-gnu/libc-2.19.so
7f46450a9000-7f46450ad000 r--p 001be000 08:01 39323766                   /lib/x86_64-linux-gnu/libc-2.19.so
7f46450ad000-7f46450af000 rw-p 001c2000 08:01 39323766                   /lib/x86_64-linux-gnu/libc-2.19.so
7f46450af000-7f46450b4000 rw-p 00000000 00:00 0
7f46450b4000-7f46450d7000 r-xp 00000000 08:01 39323754                   /lib/x86_64-linux-gnu/ld-2.19.so
7f46452b9000-7f46452bc000 rw-p 00000000 00:00 0
7f46452d4000-7f46452d6000 rw-p 00000000 00:00 0
7f46452d6000-7f46452d7000 r--p 00022000 08:01 39323754                   /lib/x86_64-linux-gnu/ld-2.19.so
7f46452d7000-7f46452d8000 rw-p 00023000 08:01 39323754                   /lib/x86_64-linux-gnu/ld-2.19.so
7f46452d8000-7f46452d9000 rw-p 00000000 00:00 0
7ffe3e5f9000-7ffe3e662000 rw-p 00000000 00:00 0                          [stack]
7ffe3e6e1000-7ffe3e6e3000 r-xp 00000000 00:00 0                          [vdso]
ffffffffff600000-ffffffffff601000 r-xp 00000000 00:00 0                  [vsyscall]
```

RETURN ORIENTED PROGRAMMING

- What space is not randomized and executable?
  - Text section! (Binary instructions)
- How do we exploit text section?
  - Think about ret instruction!
- Similar to coding in assembly
  - Need to find instructions that do what you desire

RETURN ORIENTED PROGRAMMING

- Gadgets
  - Piece of machine instruction
  - Usually ends with `ret` instruction
- Need to disassemble functions to look for gadgets

RETURN ORIENTED PROGRAMMING

- Gadgets - Example

```
void setval_210(unsigned *p)
{
    *p = 3347663060U;
}
```

```
movq %rax, %rdi
```

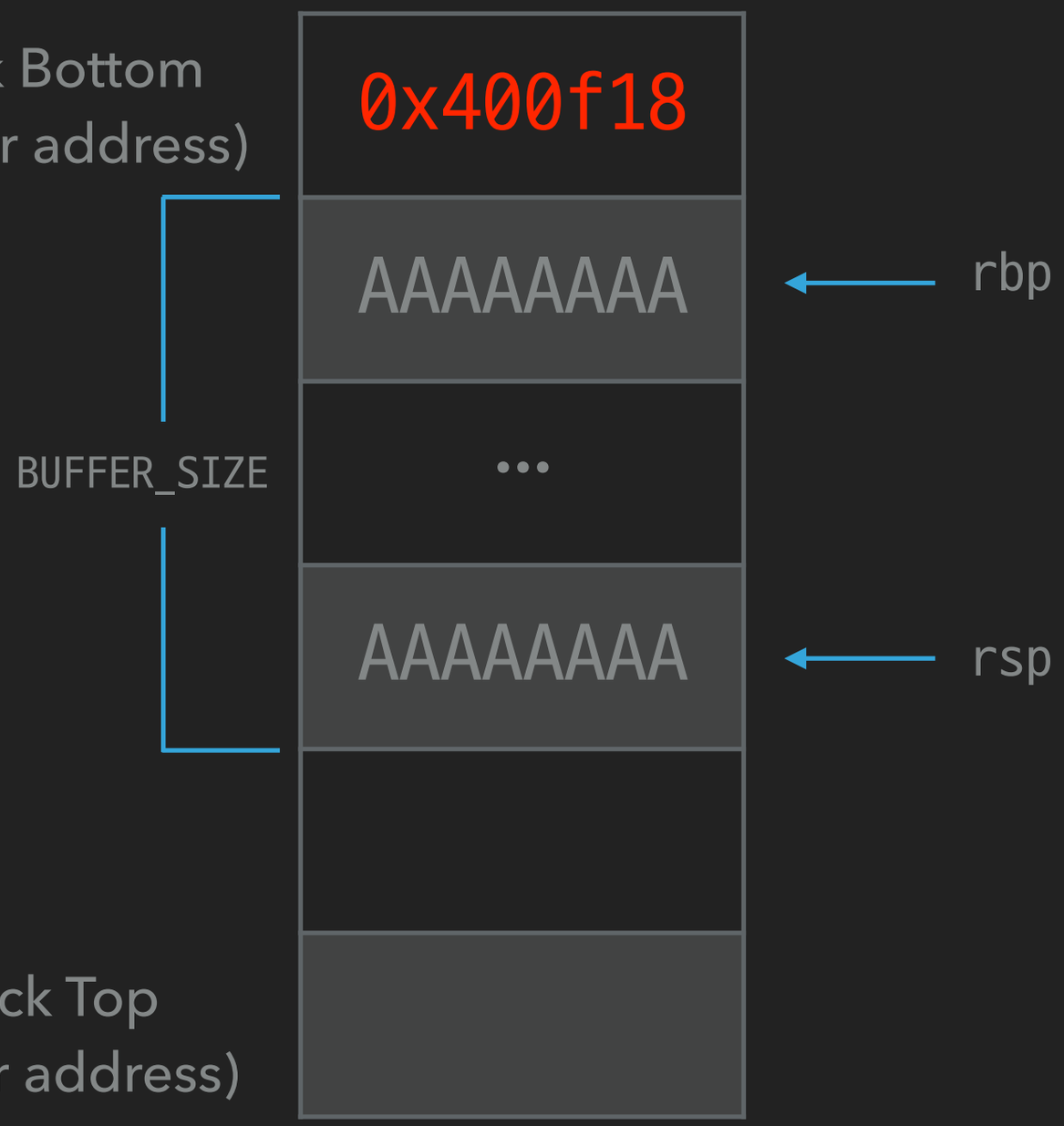
```
0000000000400f15 <setval_210>:
  400f15:       c7 07 d4 48 89 c7       movl   $0xc78948d4,(%rdi)
  400f1b:       c3                      retq
```

0x400f18: movq %rax, %rdi; retq;

RETURN ORIENTED PROGRAMMING

Gets: A * (BUFFER_SIZE) + (\x18\x0f\x40\x00\x00\x00\x00\x00)

```
1  void test()
2  {
3      int val;
4      val = getbuf();
5      printf("No exploit. Getbuf returned 0x%x\n", val);
6  }
7
8  unsigned getbuf()
9  {
10     char buf[BUFFER_SIZE];
11     Gets(buf);
12     return 1;
13 }
```



In getbuf - Line 12

This is an approximation. You need to disassemble the functions to get correct offsets.

RETURN ORIENTED PROGRAMMING

`ret`

```
1  void test()
2  {
3      int val;
4      val = getbuf();
5      printf("No exploit. Getbuf returned 0x%x\n", val);
6  }
7
8  unsigned getbuf()
9  {
10     char buf[BUFFER_SIZE];
11     Gets(buf);
12     return 1;
13 }
```

Stack Bottom
(Higher address)

Stack Top
(Lower address)

This is an approximation. You need to disassemble the functions to get correct offsets.

`In gadget - 0x400f18`

RETURN ORIENTED PROGRAMMING

- You can chain multiple gadgets
  - Gadgets end with ret instruction (`ret` ≈ `pop %rip`)
  - Put another gadget's return address to appropriate position in order to create ROP chain!

TIPS

- `hex2raw`
  - Your answer must be in binary data, but you can only type hex-formatted strings
- `hex2raw` is provided for this lab to convert a hex-formatted string to binary data

TIPS

- hex2raw - Example
  - answer.txt -> `ee ff c0 00 00 00 00 00`
- Convert to binary string
  - `./hex2raw < answer.txt`
- Convert to binary string and save to `answer_raw.txt`
  - `./hex2raw < answer.txt > answer_raw.txt`

TIPS

- hex2raw - Example
  - answer.txt -> `ee ff c0 00 00 00 00 00`
- Feed the converted string to target binary
  - `./ctarget -i answer_raw.txt`
  - `./rtarget -i answer_raw.txt`

TIPS

- Be careful of the endianness of your code!
  - x86, x86_64 - Little endian
- Ex) Providing `0xdeadbeef` to the machine
  - If you give `de ad be ef` to `hex2raw`, it will be interpreted as `0xefbeadde` by the machine
  - In order to provide proper input, you must give `ef be ad de`