Binary Exploitation - Basic

yuawn

About

- yuawn
- Pwn
- Balsn / DoubleSigma

_yuawn

Outline

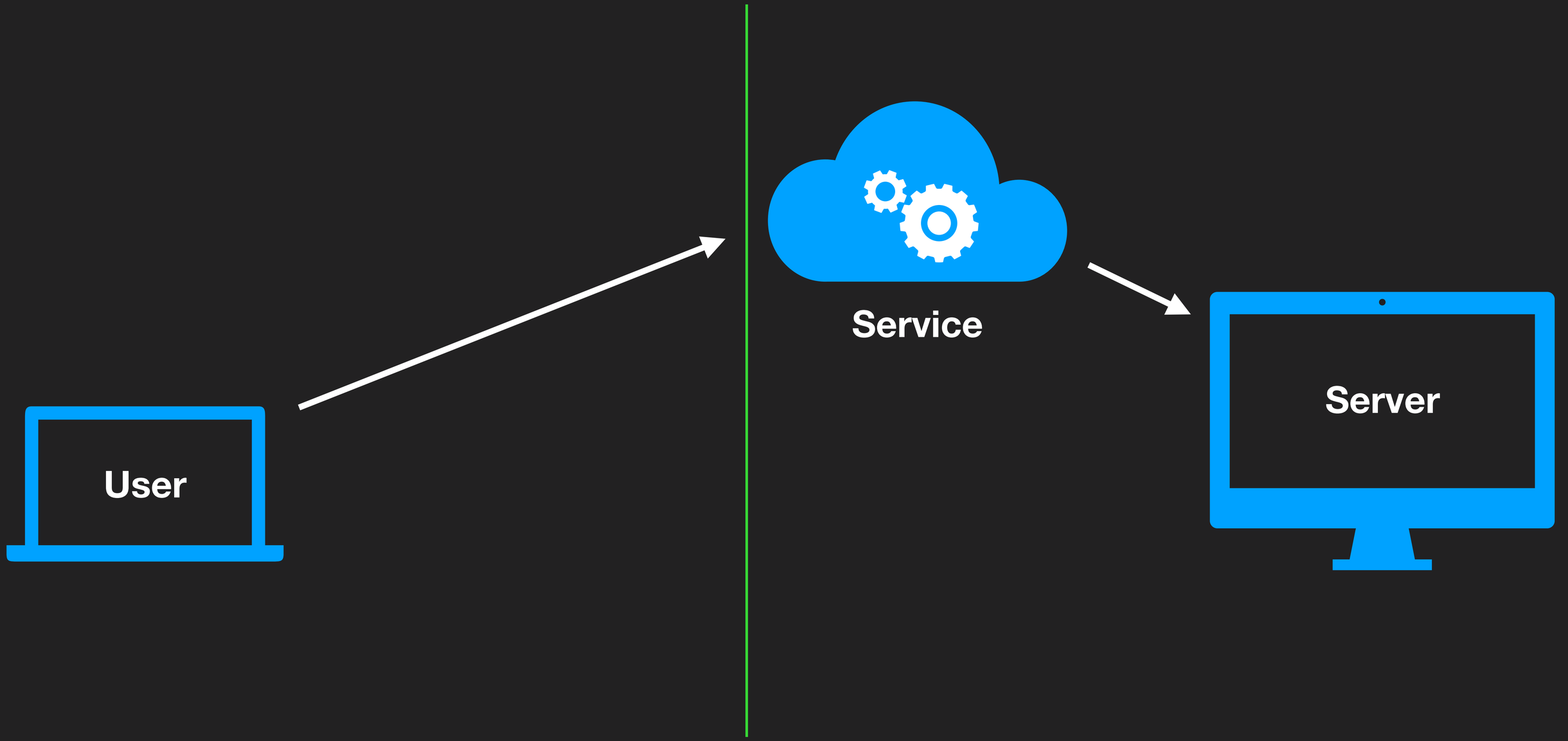
- Binary exploitation
- Basic concepts
  - ELF
  - x64 calling convention
  - Stack frame
- BOF - Buffer Overflow
- Shellcode
- Lazy binding - GOT Hijacking

What is Pwn?

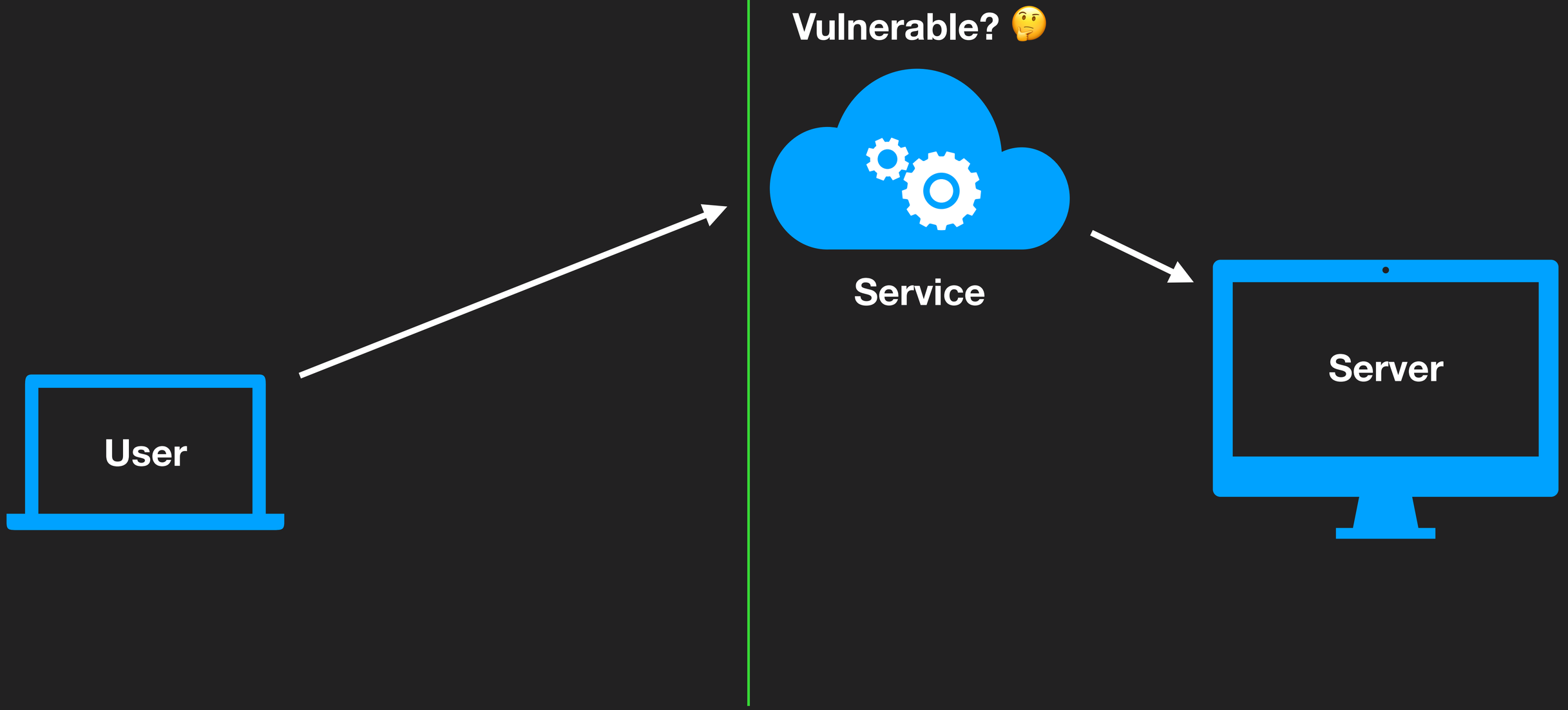
Binary Exploitation

- 透過利用程式 (Binary) 的漏洞 (Vulnerability)，在執行期間控制執行流程 (Control flow)，進而使程式執行特定行為。
- Pwn
  - 胖 or 碰

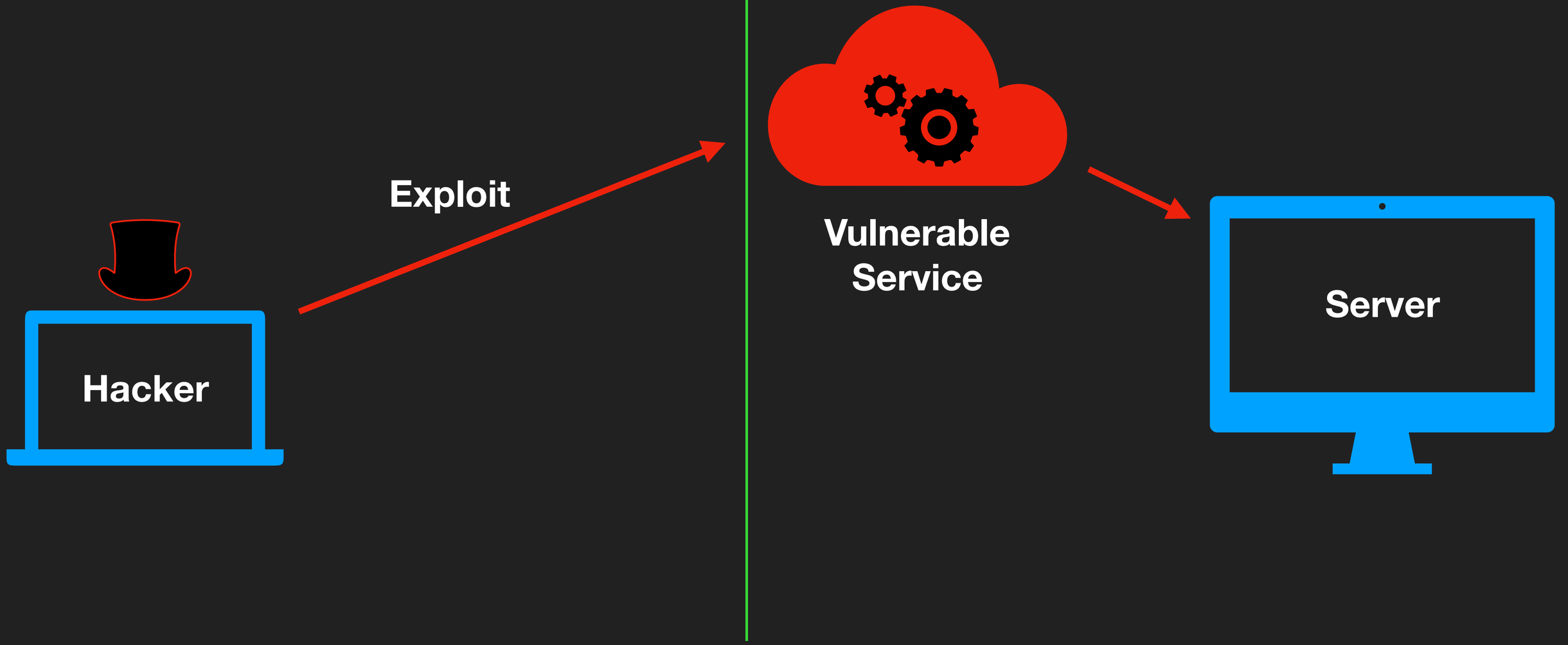
Binary Exploitation



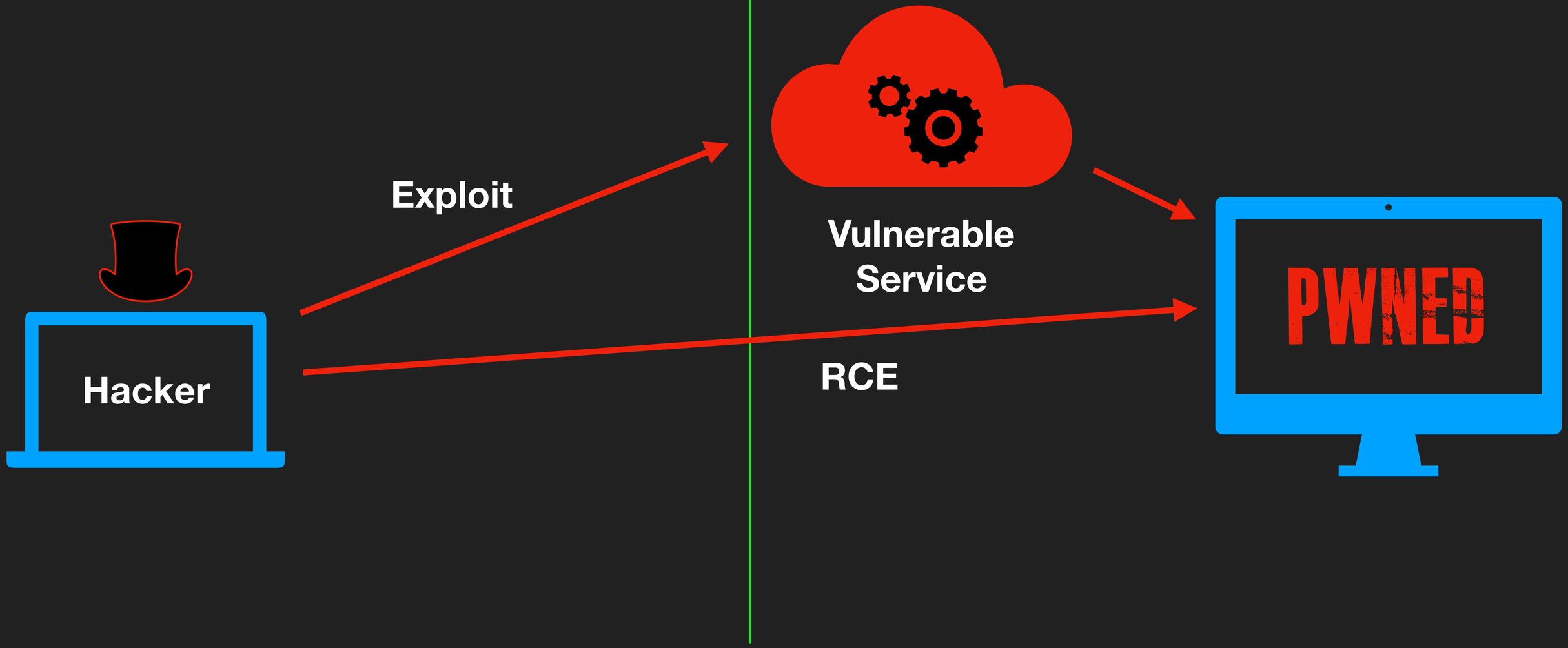
Binary Exploitation



Binary Exploitation



Binary Exploitation



Basic Concepts

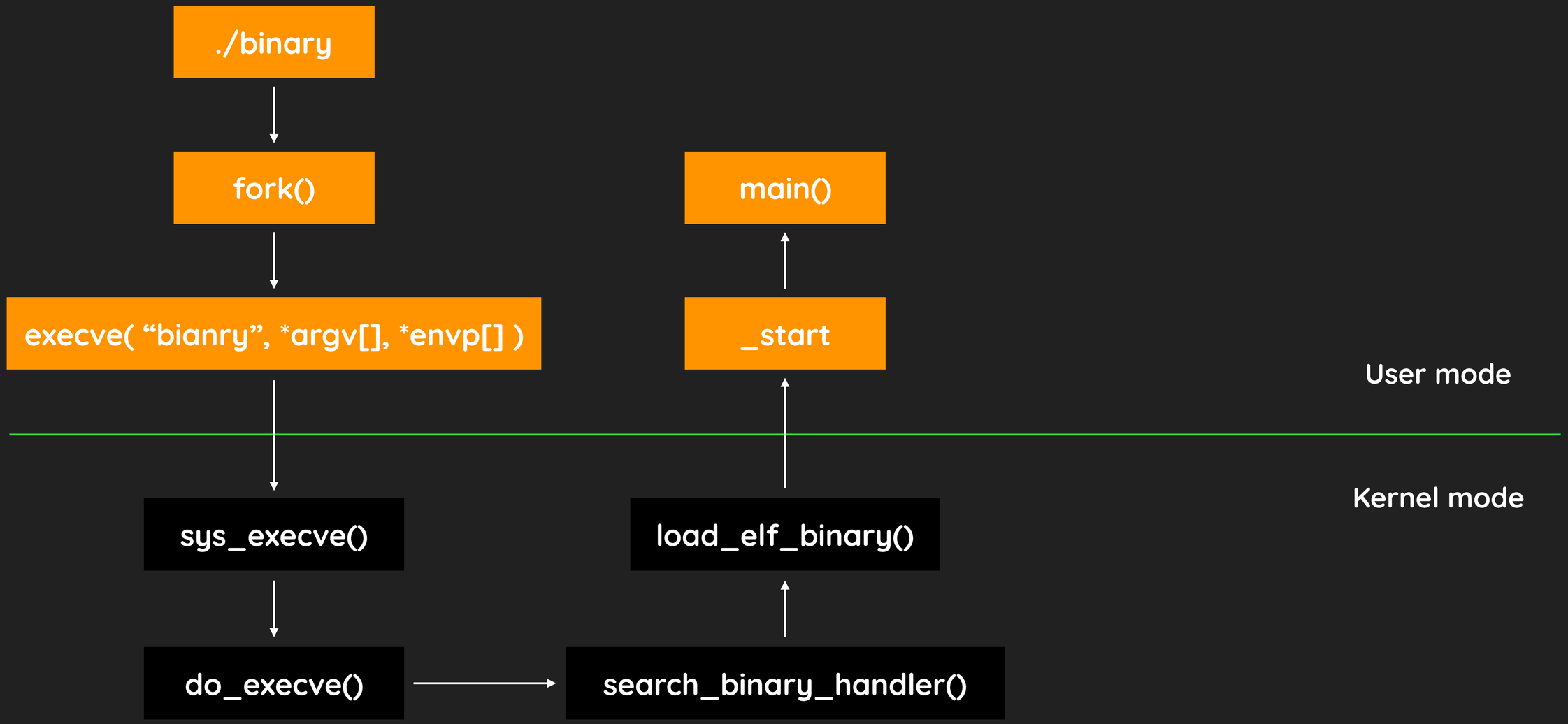
ELF

Executable and Linkable Format

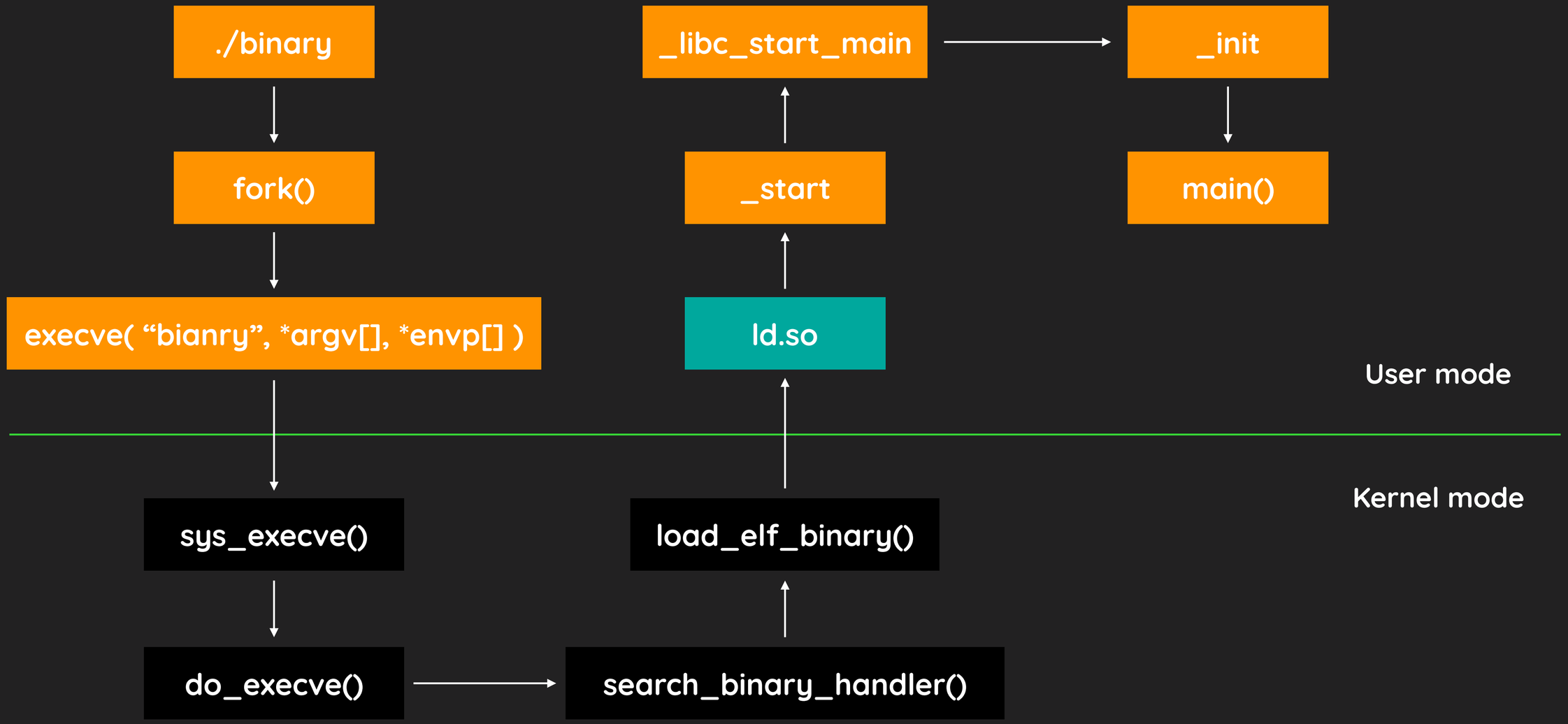
ELF

- Executable and Linkable Format
- 執行檔 ex. exe
- section
  - 執行時會 mapping 到 RAM 上 (virtual memory)
  - .text .bss .data .rodata .got .plt .fini ...

ELF - Workflow (static)



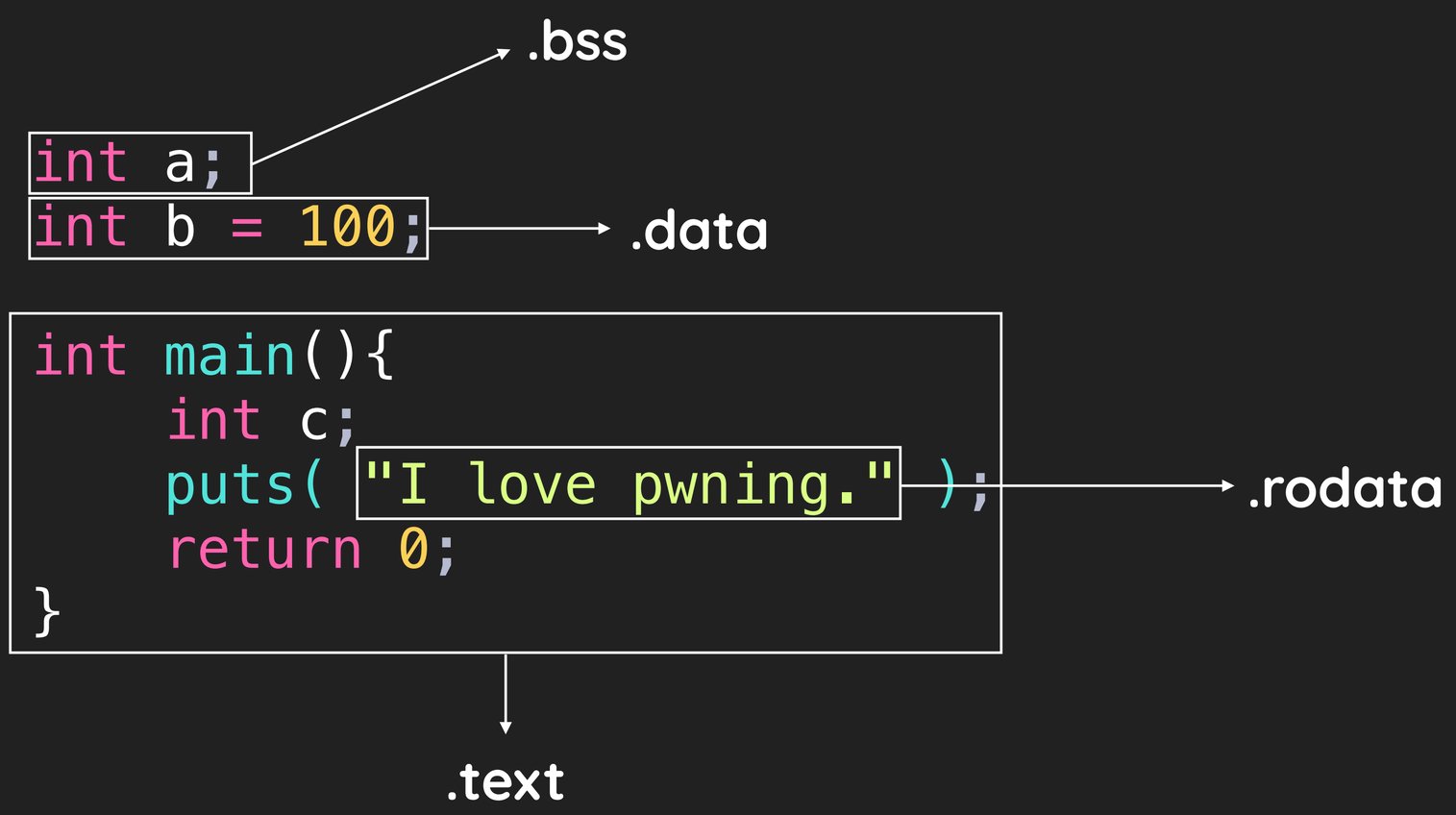
ELF - Workflow (dynamic linking)



ELF - section

- .bss - 存放未初始化值的全域變數 (global variable)
- .data - 存放具初始化值的全域變數
- .rodata - 存放唯讀 (read-only) 資料
- .text - 存放編譯後的 code

ELF - section


.bss
int a;
int b = 100;
.data
int main(){
int c;
puts("I love pwning.");
return 0;
}
.rodata
.text

ELF - Protections

- PIE - Position-Independent Executable
- NX - No-eXecute
- Canary - stack protector
- RELRO - Relocation Read-Only

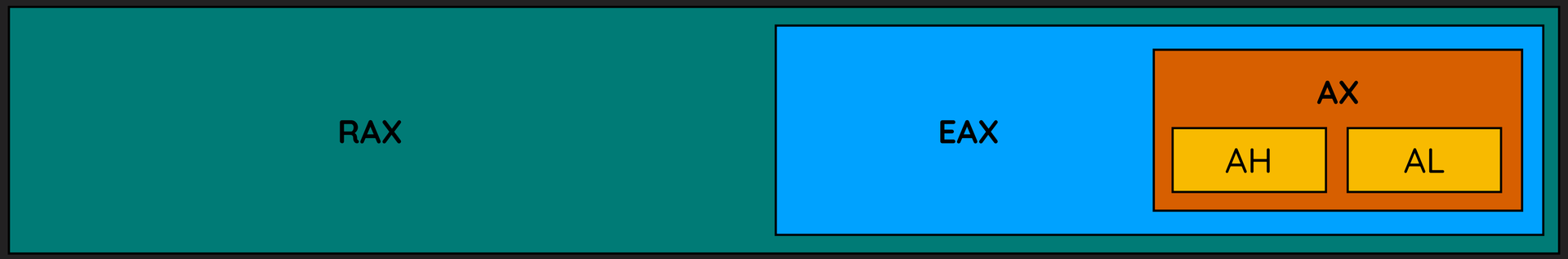
DEMO

x64

- 8 bytes alignment
- Stack 0x10 bytes alignment

x64 Assembly

- Registers
  - RAX RBX RCX RDX RDI RSI - 64 bit
  - EAX EBX ECX EDX EDI ESI - 32 bit
  - AX BX CX DX DI SI - 16 bit
  - AX -> AH AL - 8 bit



x64 Assembly

- Registers
  - RSP - Stack Pointer Register
    - 指向 stack 頂端（頭）
  - RBP - Base Pointer Register
    - 指向 stack 底端（尾）
  - RIP - Program Counter Register
    - 指向當前執行指令instruction位置

x64 Assembly

- jmp (jump)
  - 跳至程式某一地址 A(address) 執行
  - jmp A = mov rip, A
- call
  - 將 call 完後回來緊接著要執行的下一行指令位置 push 到 stack 上儲存起來，再跳過去執行。
  - call A = push next_rip
    mov rip, A

x64 Assembly

- leave
  - 還原至 caller 的 stack frame。
  - mov rsp, rbp
    pop rbp
- ret (return)
  - pop rip

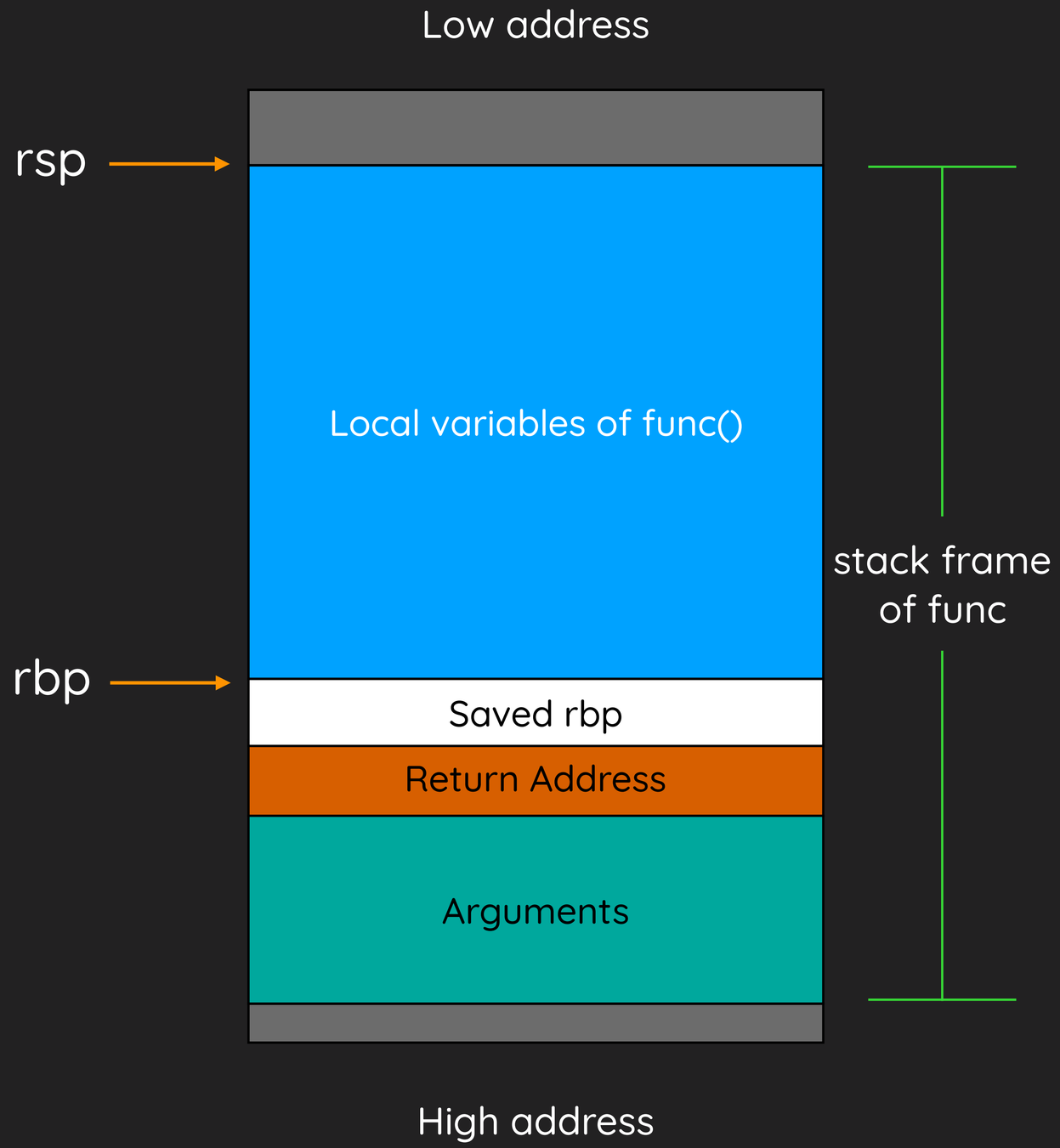
x64 calling convention

- Pass parameters
  - rdi, rsi, rdx, rcx, r8, r9, (stack)
  - rdi, rsi, rdx, r10, r8, r9, (stack)
  - rax - store return value
- x64 - register 傳參
- x86 - stack 傳參

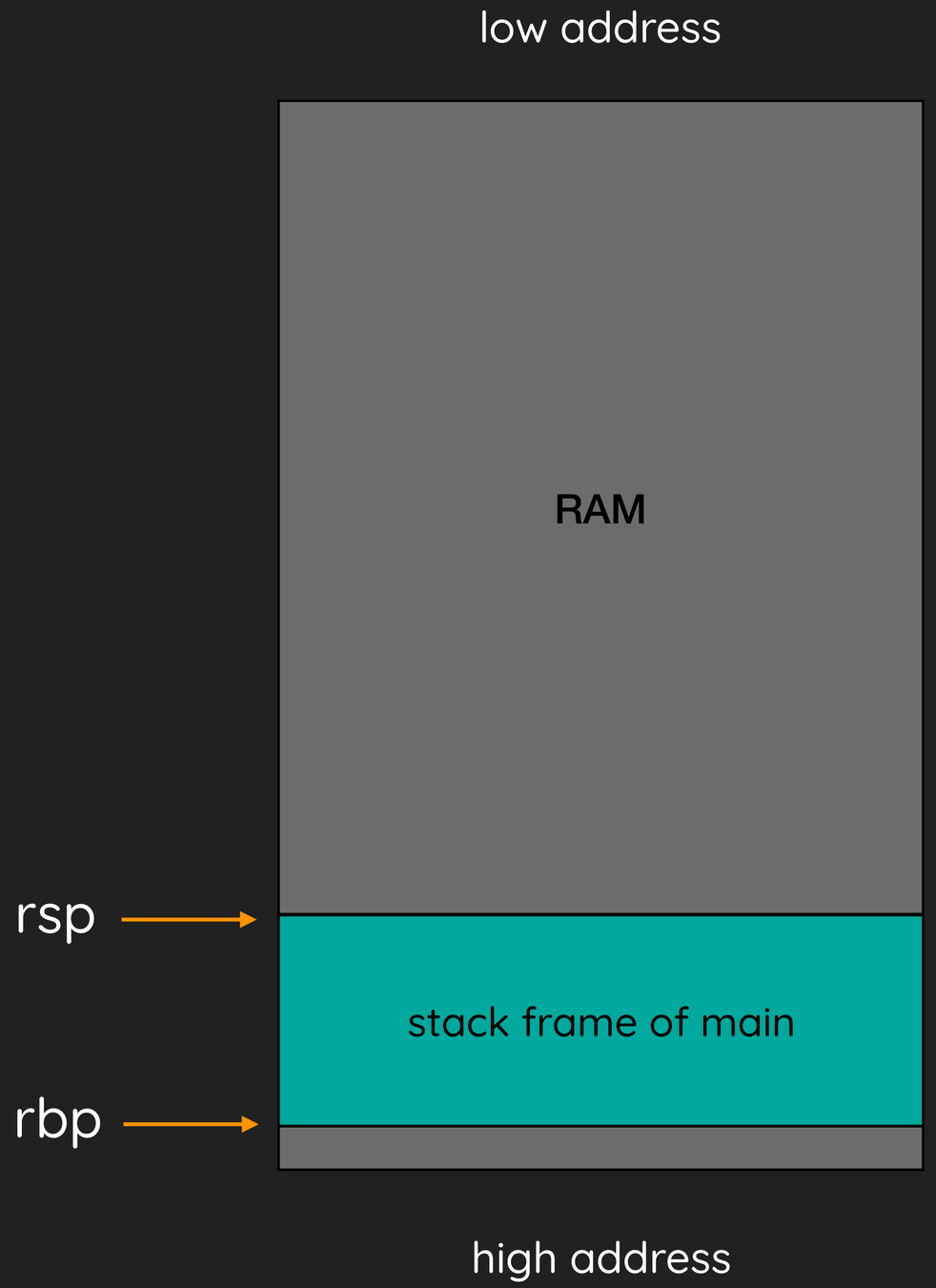
Stack Frame

Stack Frame

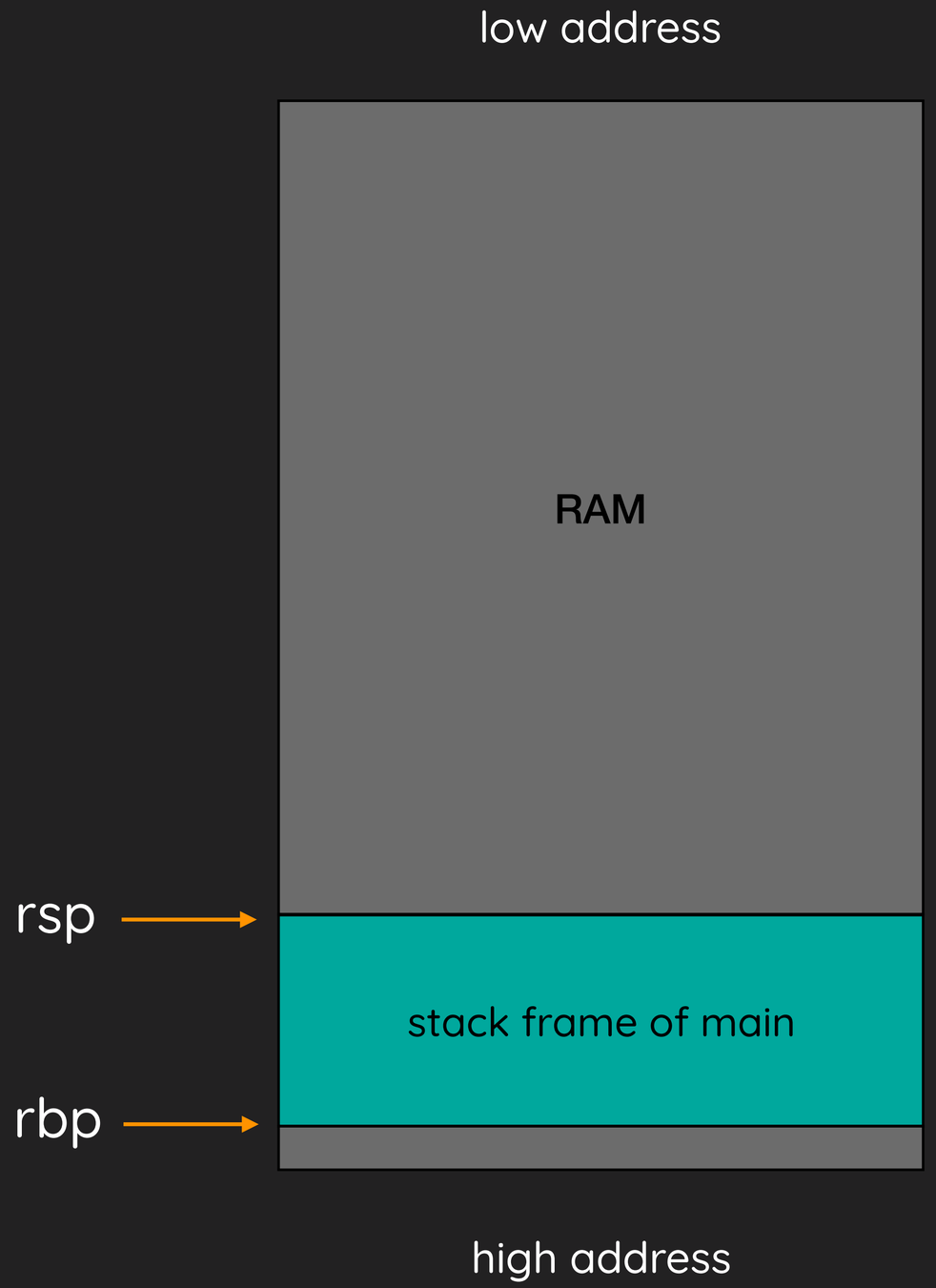
- Function Prologue
- Function Epilogue
- Stack frame
  - local variables
  - [rbp] = old rbp (caller rbp)
  - [rbp + 0x8] = Return Address



```
func:
        push rbp
        mov rbp, rsp
        sub rsp, 0x70

        ...

        mov eax, 0x1
        leave
        ret

main:
        push rbp
        mov rbp, rsp
        mov rdi, 1234
        mov rsi, 666
        call func
        mov eax, 0      // address  = 0x40071A
        leave
        ret
```

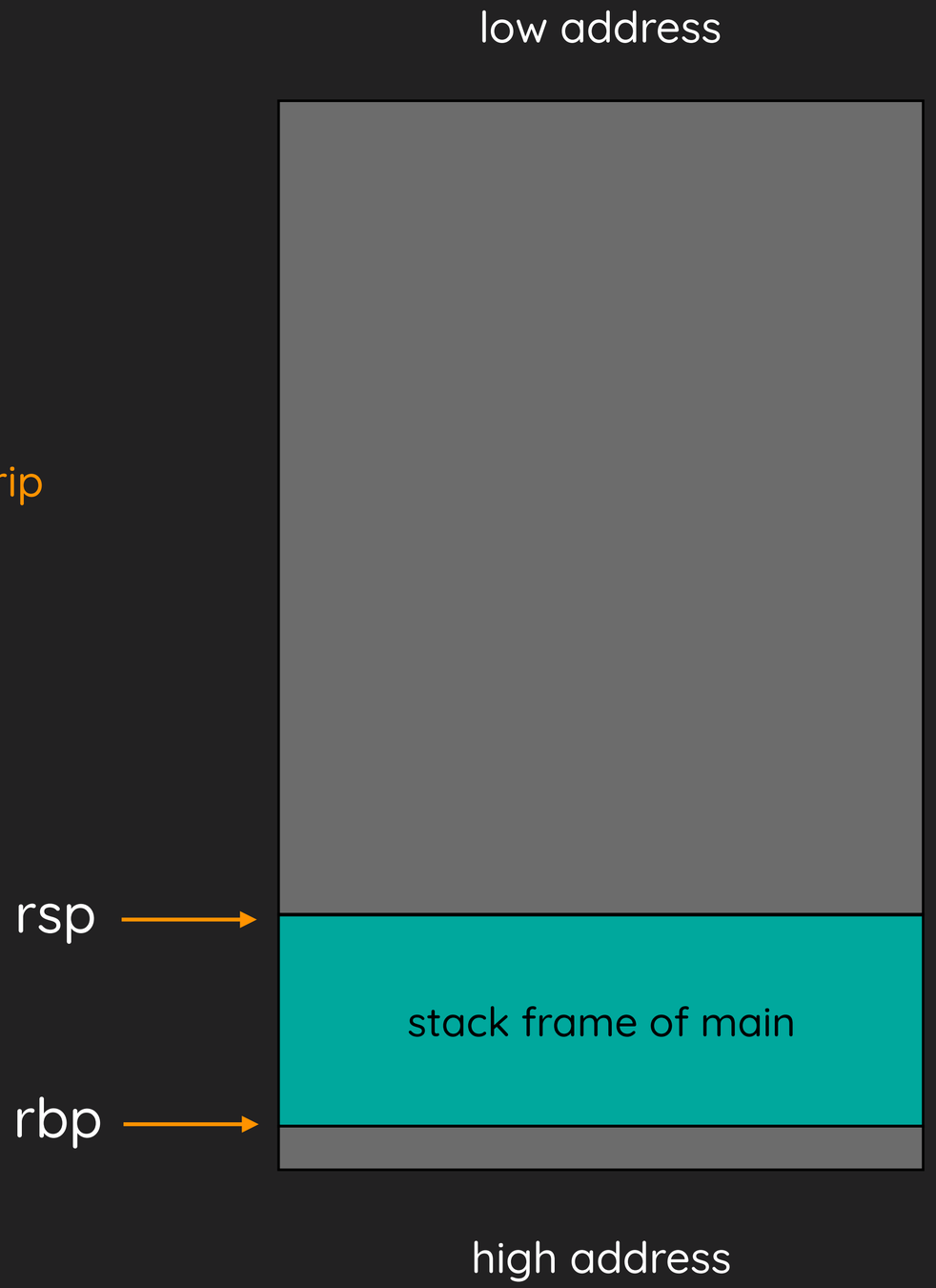


```
func:
        push rbp
        mov rbp, rsp
        sub rsp, 0x70

        ...

        mov eax, 0x1
        leave
        ret

main:
        push rbp
        mov rbp, rsp
        mov rdi, 1234
        mov rsi, 666
rip →   call func
        mov eax, 0      // address  = 0x40071A
        leave
        ret
```



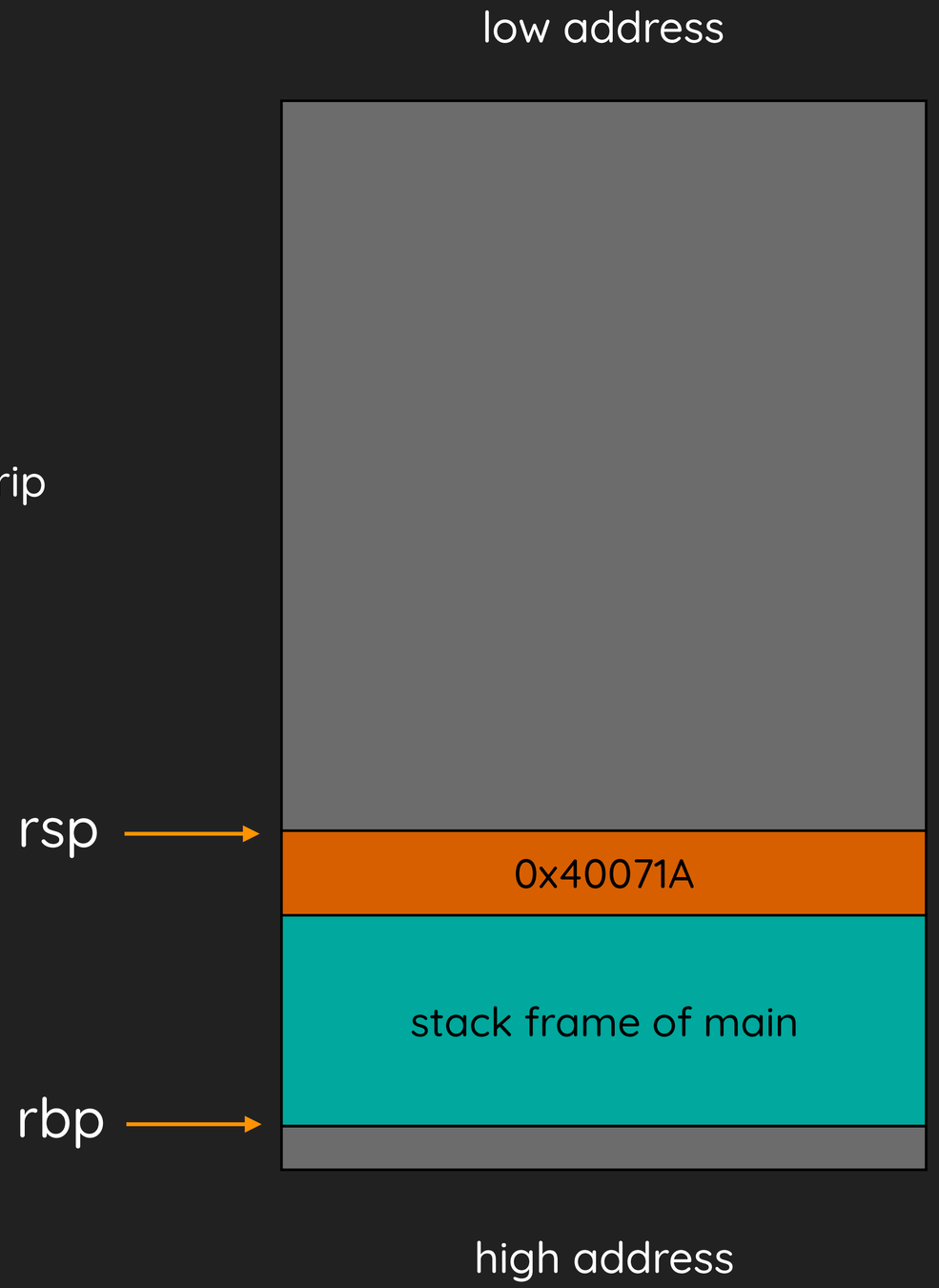
```
func:
        push rbp
        mov rbp, rsp
        sub rsp, 0x70

        ...

        mov eax, 0x1
        leave
        ret

main:
        push rbp
        mov rbp, rsp
        mov rdi, 1234
        mov rsi, 666
rip →   call func
        mov eax, 0      // address  = 0x40071A
        leave
        ret
```

Call func = push next-rip
jmp func



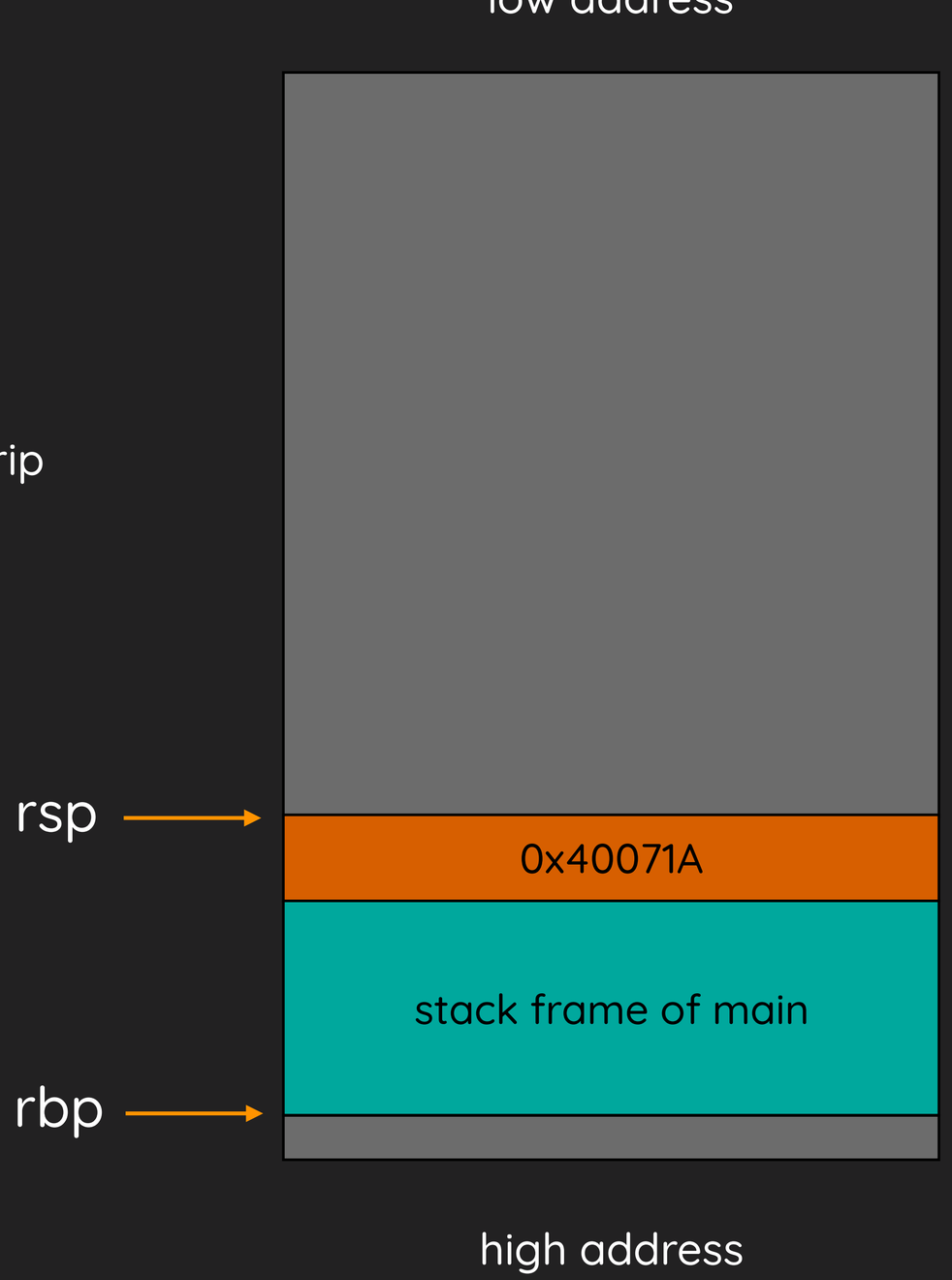
```
func:
        push rbp
        mov rbp, rsp
        sub rsp, 0x70

        ...

        mov eax, 0x1
        leave
        ret

main:
        push rbp
        mov rbp, rsp
        mov rdi, 1234
        mov rsi, 666
rip →   call func
        mov eax, 0      // address  = 0x40071A
        leave
        ret
```

Call func = push next-rip
jmp func

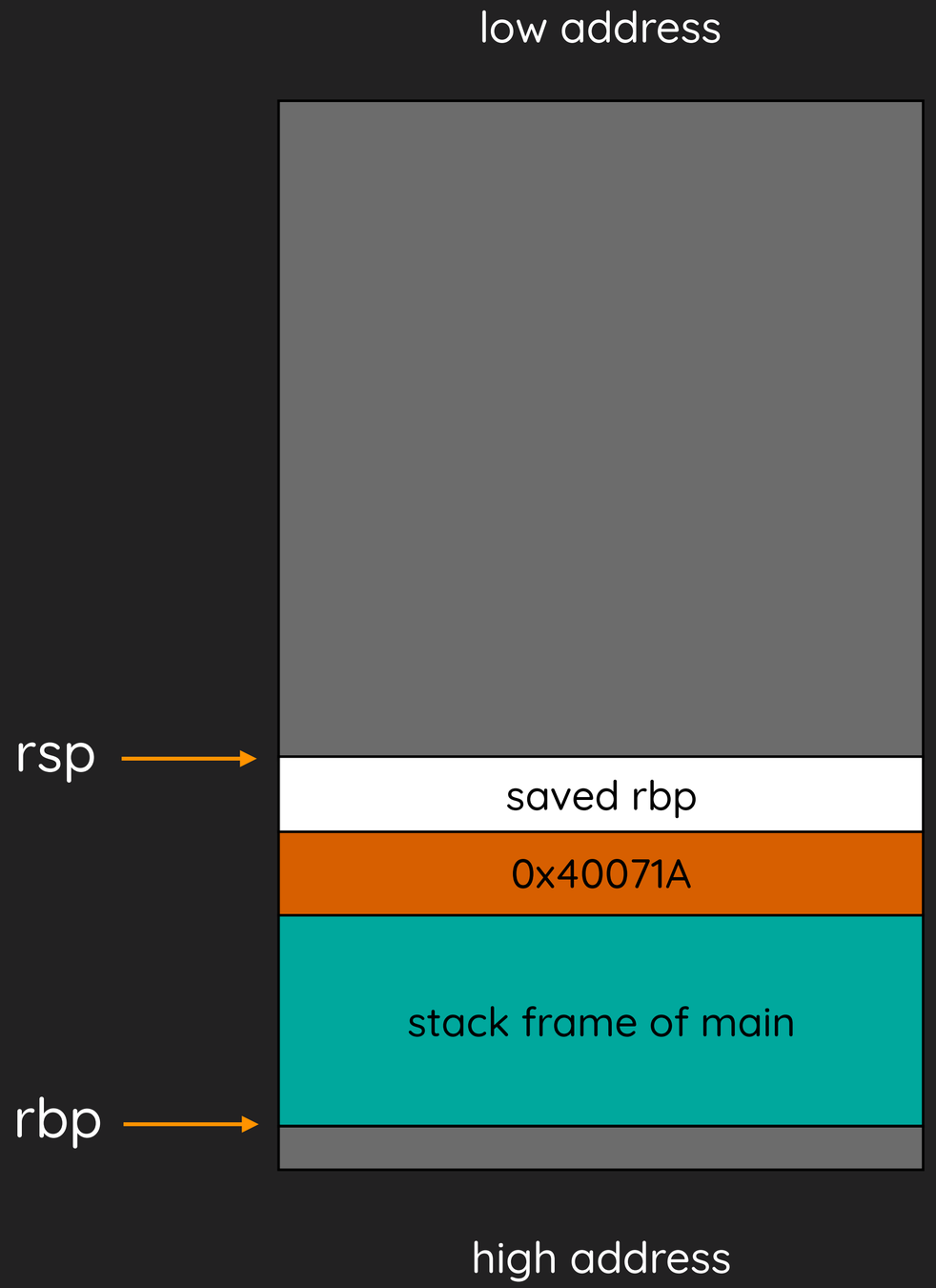


```
func:
 rip → push rbp
       mov rbp, rsp
       sub rsp, 0x70

       ...

       mov eax, 0x1
       leave
       ret

main:
       push rbp
       mov rbp, rsp
       mov rdi, 1234
       mov rsi, 666
       call func
       mov eax, 0      // address  = 0x40071A
       leave
       ret
```

Call func = push next-rip
jmp func

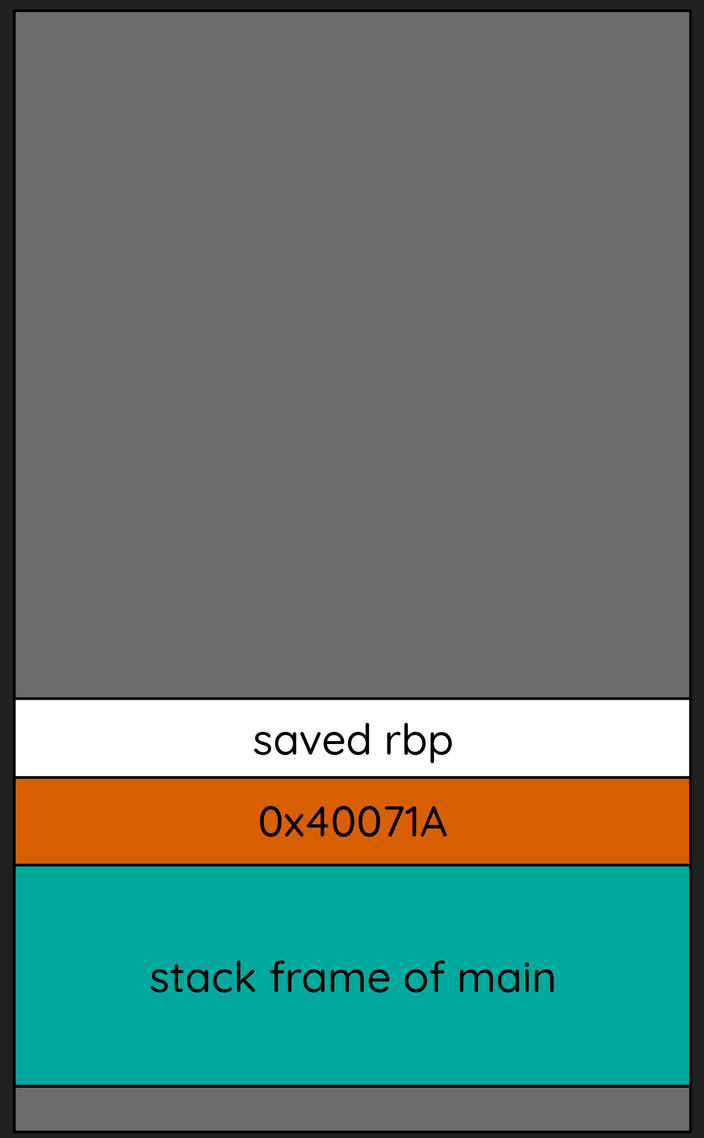


```
func:
            push rbp
rip  ->     mov rbp, rsp
            sub rsp, 0x70

            ...

            mov eax, 0x1
            leave
            ret

main:
            push rbp
            mov rbp, rsp
            mov rdi, 1234
            mov rsi, 666
            call func
            mov eax, 0      // address  = 0x40071A
            leave
            ret
```



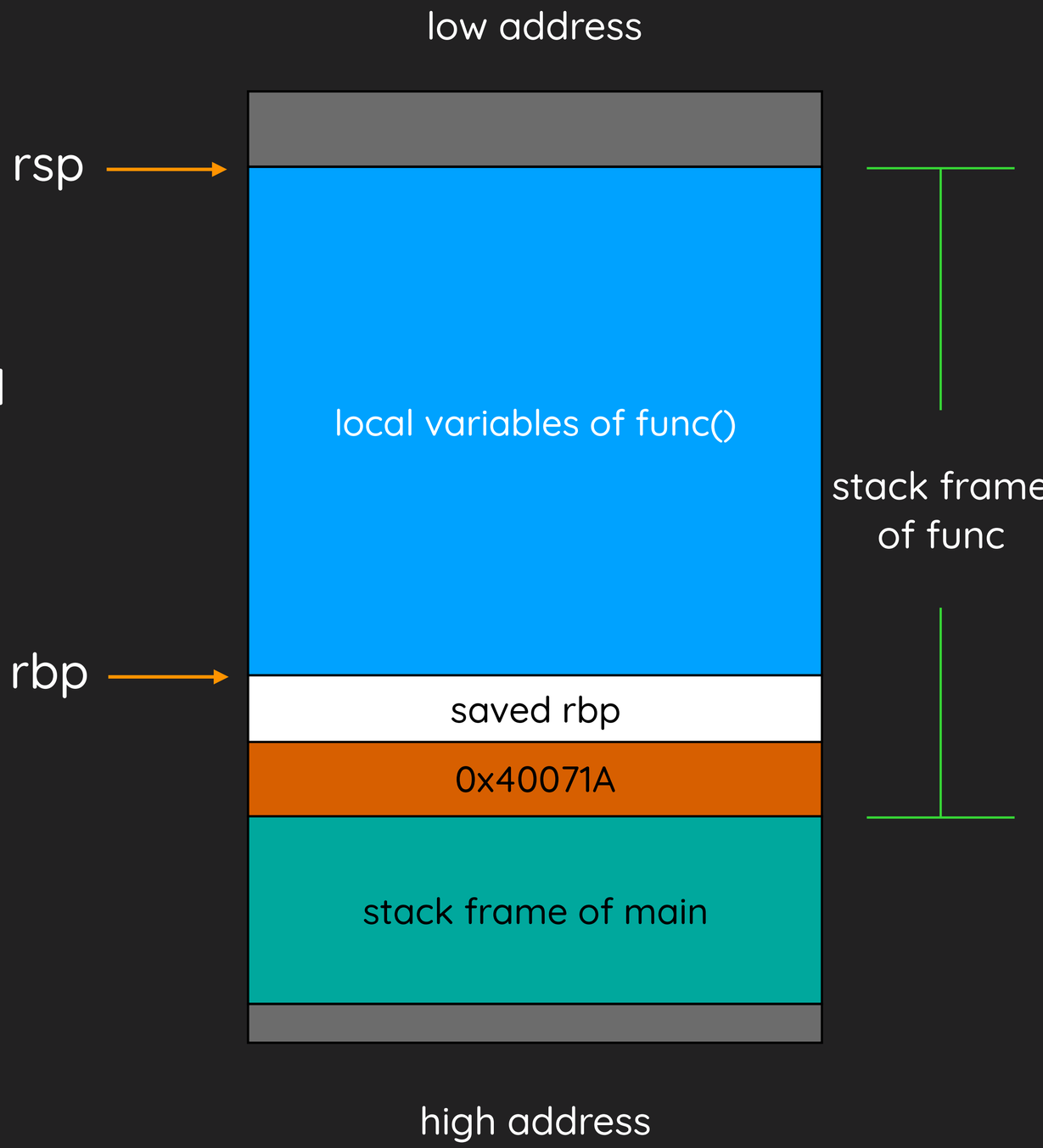
```
func:
        push rbp
        mov rbp, rsp
rip →   sub rsp, 0x70

        ...

        mov eax, 0x1
        leave
        ret

main:
        push rbp
        mov rbp, rsp
        mov rdi, 1234
        mov rsi, 666
        call func
        mov eax, 0      // address  = 0x40071A
        leave
        ret
```

rbp → rsp →

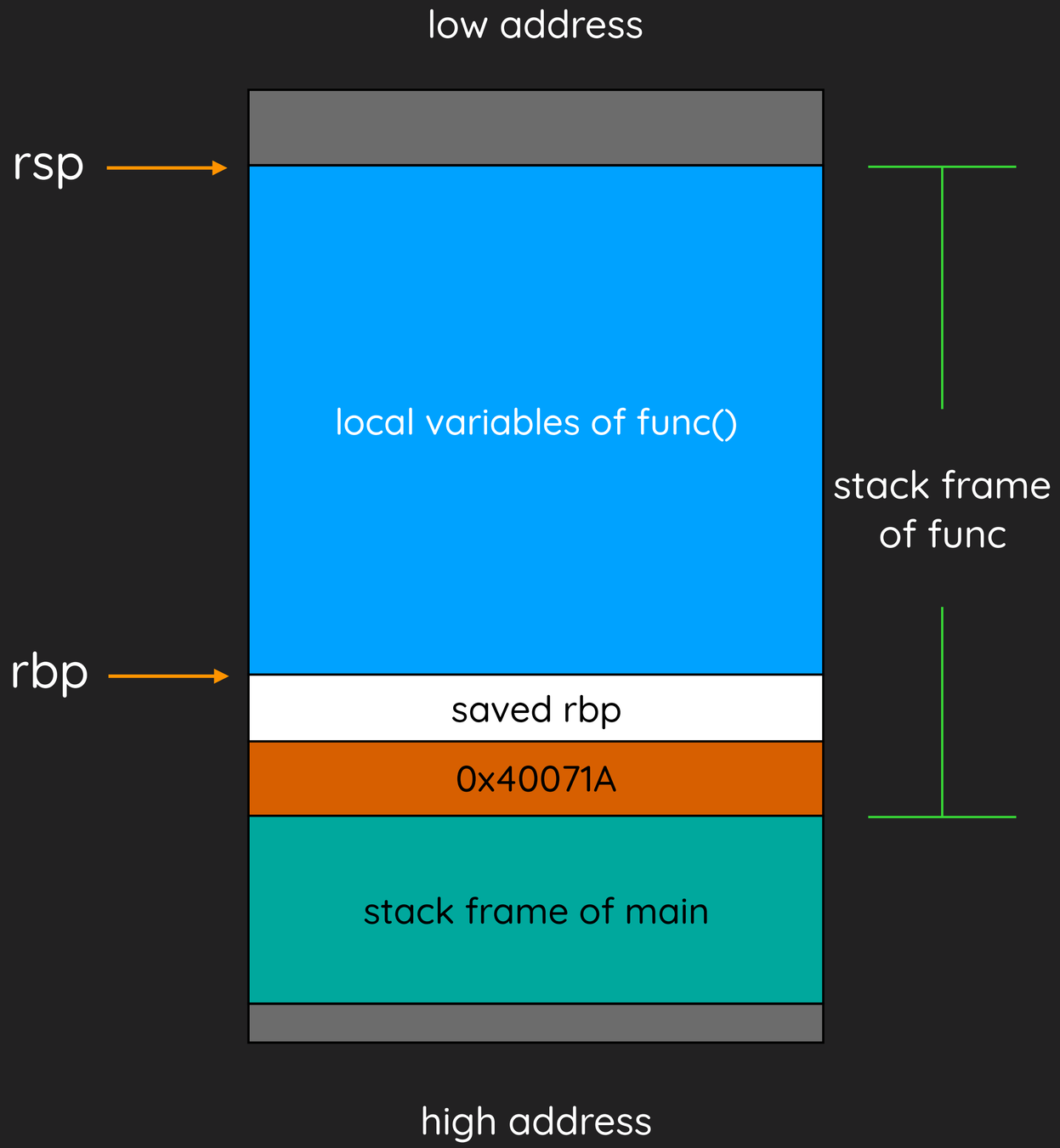


```
func:
        push rbp
        mov rbp, rsp
        sub rsp, 0x70

rip →   ...

        mov eax, 0x1
        leave
        ret

main:
        push rbp
        mov rbp, rsp
        mov rdi, 1234
        mov rsi, 666
        call func
        mov eax, 0      // address  = 0x40071A
        leave
        ret
```

Prologue finished

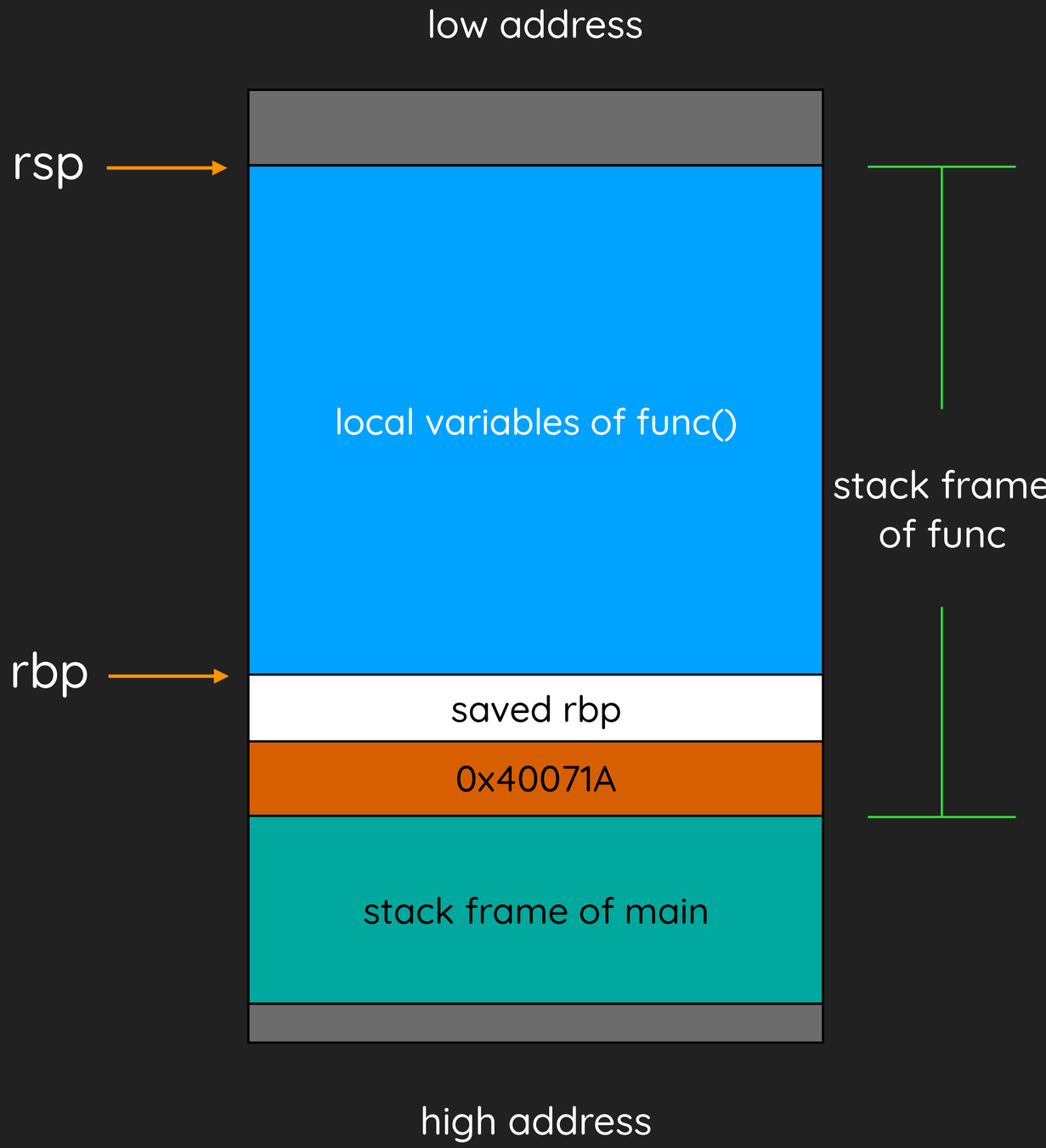


```
func:
        push rbp
        mov rbp, rsp
        sub rsp, 0x70

        ...

rip →   mov eax, 0x1
        leave
        ret

main:
        push rbp
        mov rbp, rsp
        mov rdi, 1234
        mov rsi, 666
        call func
        mov eax, 0      // address  = 0x40071A
        leave
        ret
```



```
func:
        push rbp
        mov rbp, rsp
        sub rsp, 0x70

        ...

        mov eax, 0x1
rip →   leave
        ret

main:
        push rbp
        mov rbp, rsp
        mov rdi, 1234
        mov rsi, 666
        call func
        mov eax, 0      // address  = 0x40071A
        leave
        ret
```

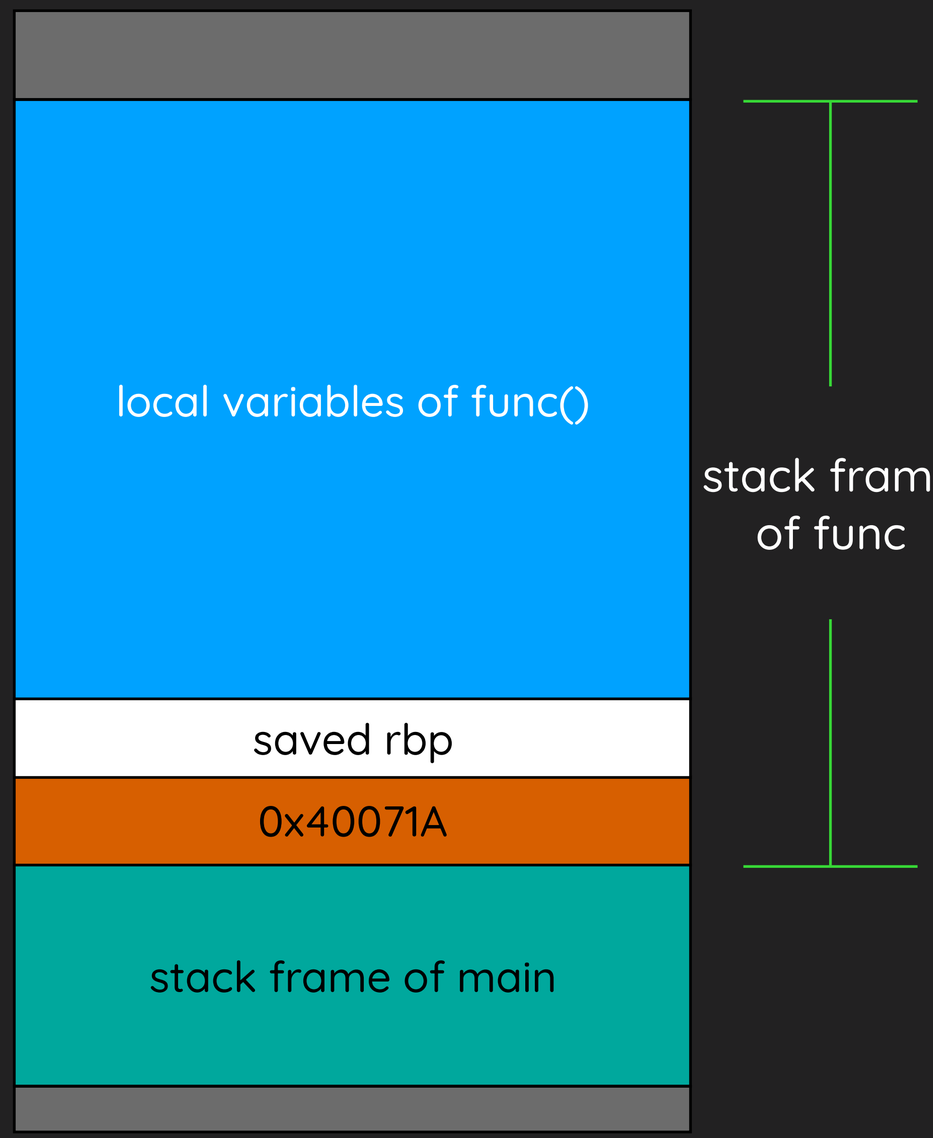
leave = mov rsp, rbp
pop rbp



```
func:
        push rbp
        mov rbp, rsp
        sub rsp, 0x70

        ...

        mov eax, 0x1
rip →   leave
        ret

main:
        push rbp
        mov rbp, rsp
        mov rdi, 1234
        mov rsi, 666
        call func
        mov eax, 0      // address  = 0x40071A
        leave
        ret
```

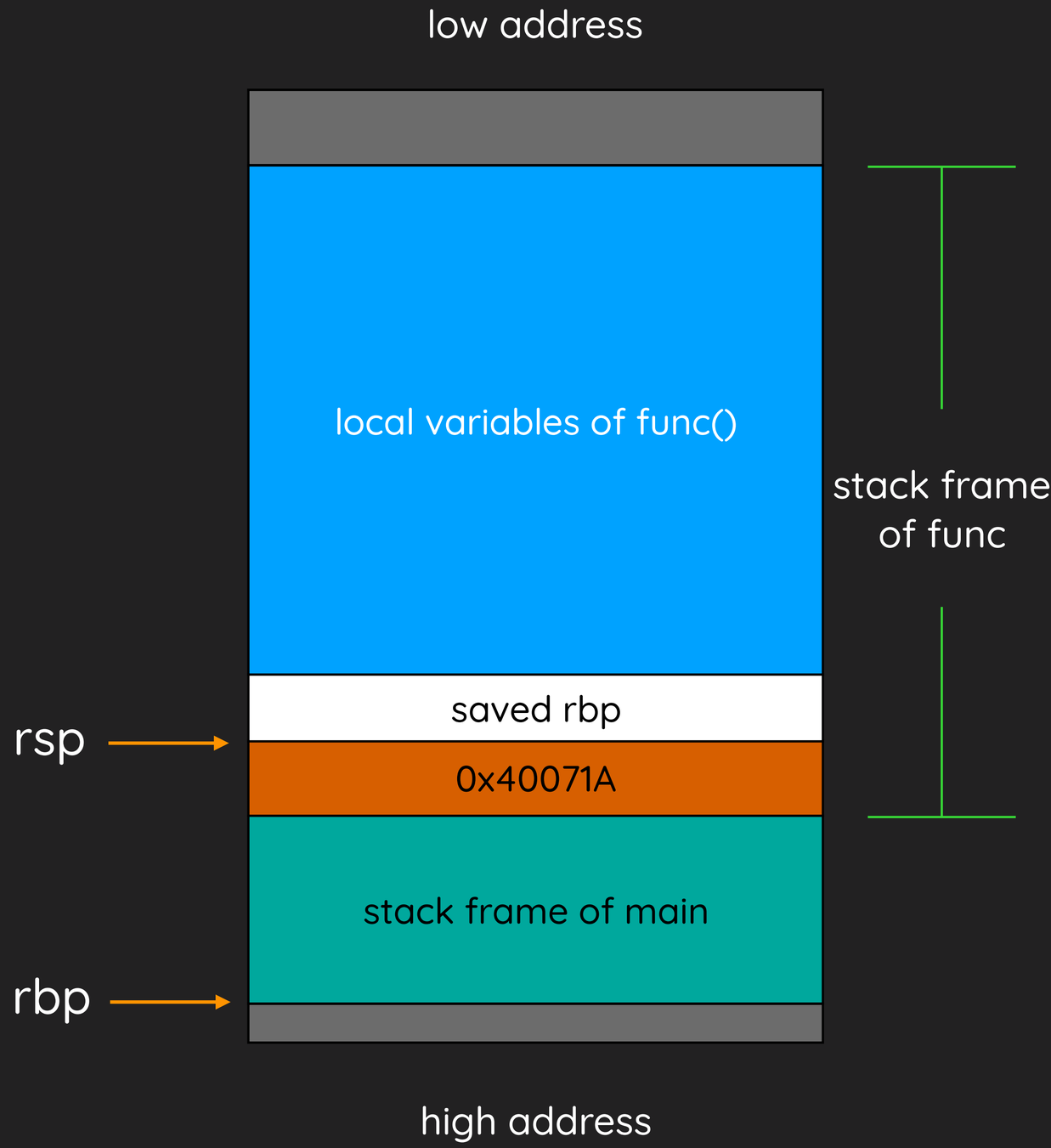
leave = mov rsp, rbp
pop rbp

rbp → rsp →



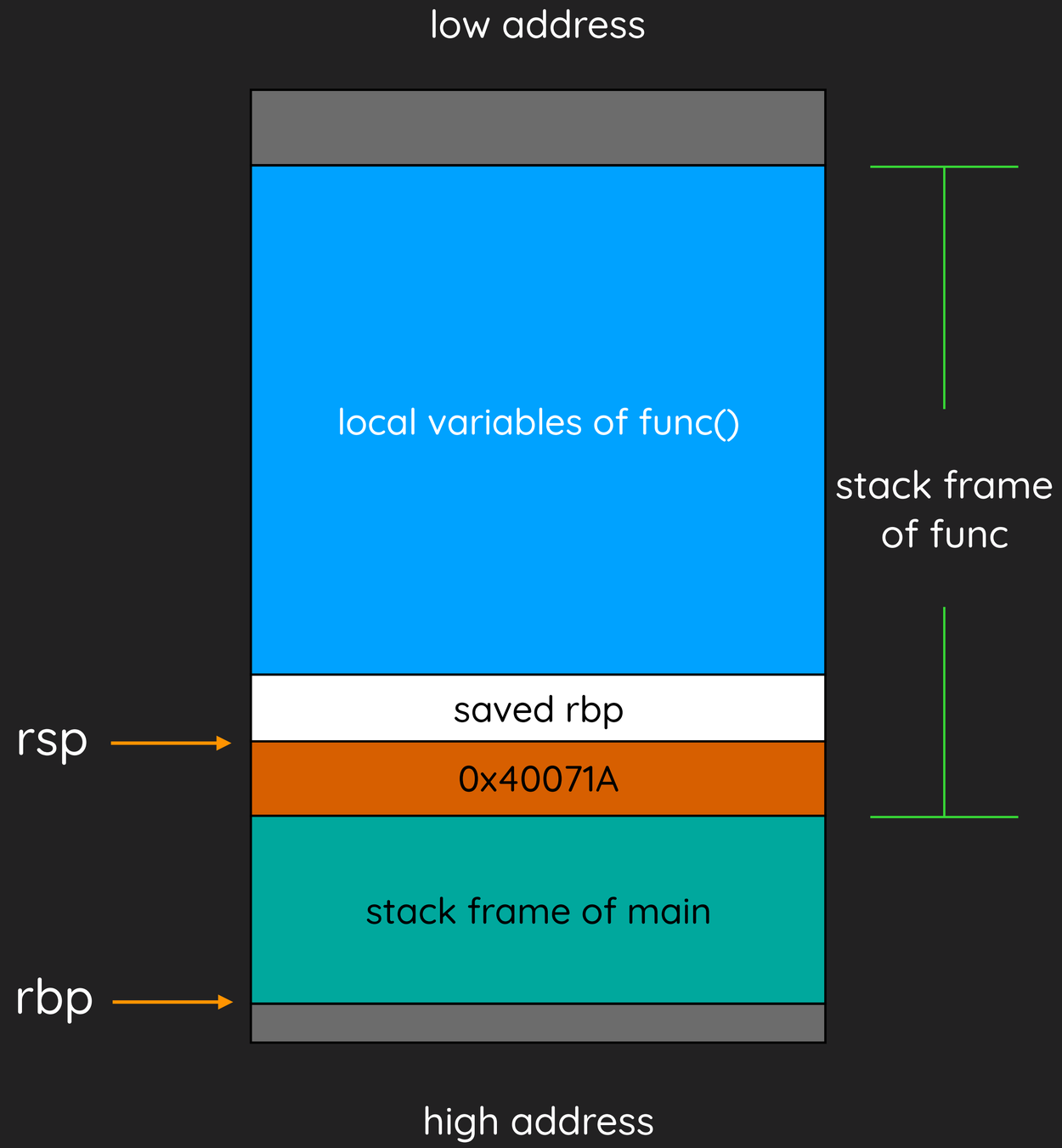
```
func:
        push rbp
        mov rbp, rsp
        sub rsp, 0x70

        ...

        mov eax, 0x1
        leave
rip →   ret

main:
        push rbp
        mov rbp, rsp
        mov rdi, 1234
        mov rsi, 666
        call func
        mov eax, 0      // address  = 0x40071A
        leave
        ret
```

leave = mov rsp, rbp
pop rbp



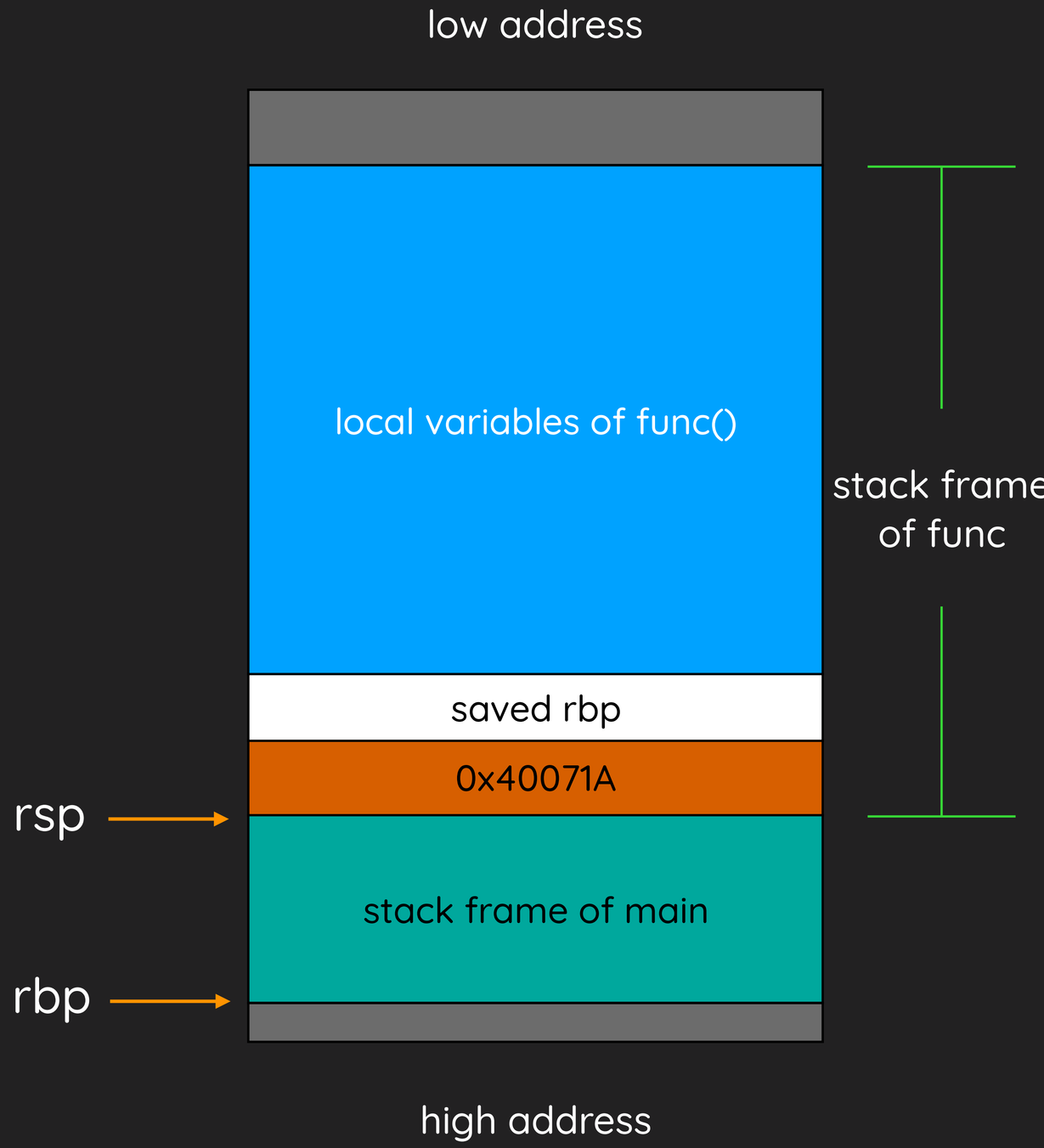
```
func:
        push rbp
        mov rbp, rsp
        sub rsp, 0x70

        ...

        mov eax, 0x1
        leave
rip →   ret

main:
        push rbp
        mov rbp, rsp
        mov rdi, 1234
        mov rsi, 666
        call func
        mov eax, 0      // address  = 0x40071A
        leave
        ret
```

ret = pop rip



```
func:
        push rbp
        mov rbp, rsp
        sub rsp, 0x70

        ...

        mov eax, 0x1
        leave
        ret

main:
        push rbp
        mov rbp, rsp
        mov rdi, 1234
        mov rsi, 666
        call func
rip →   mov eax, 0      // address  = 0x40071A
        leave
        ret
```

Epilogue finished



DEMO

Pwn

Binary exploitation

Environment

- Ubuntu 18.04
- libc-2.27
- x64

Overflow

Overflow

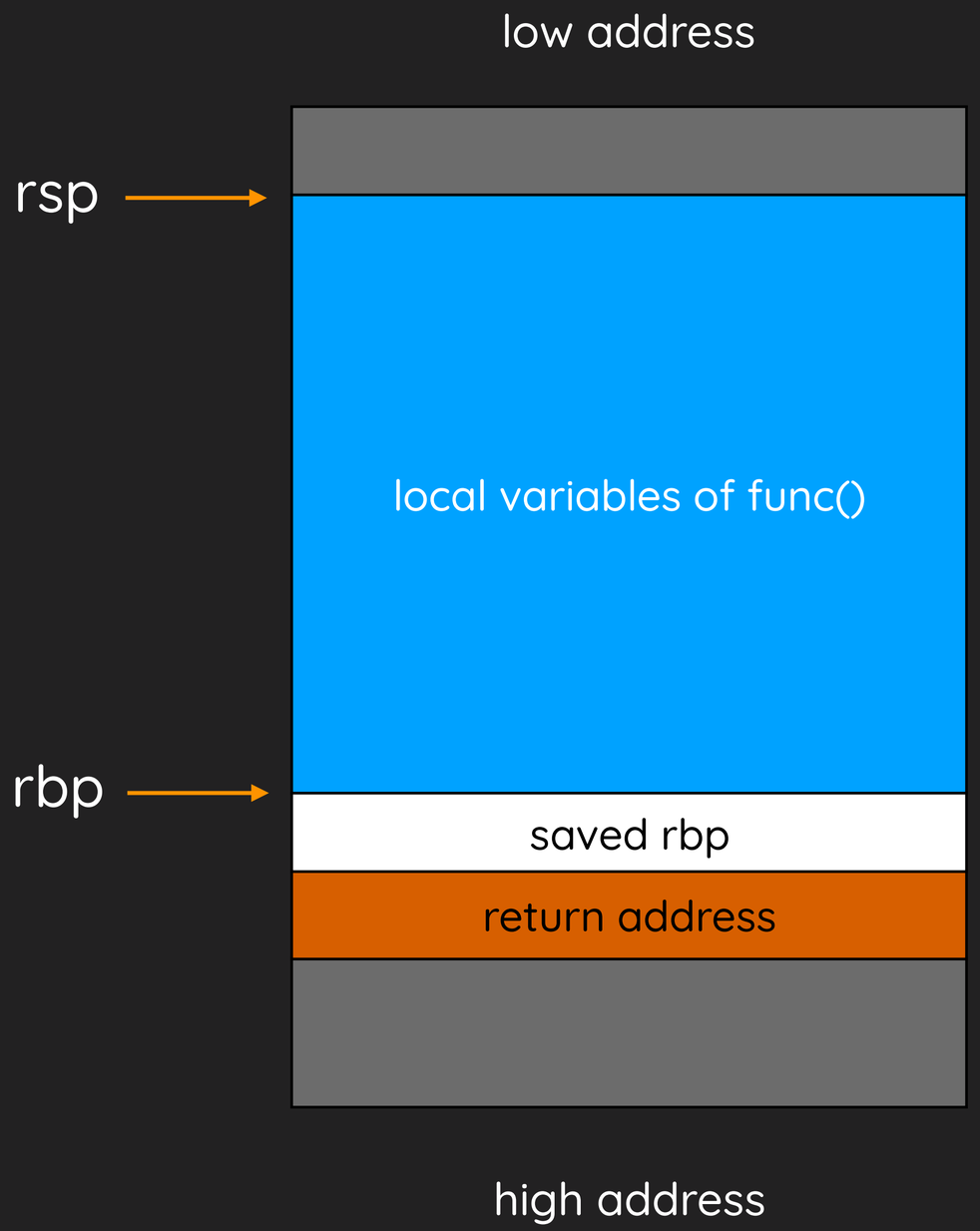
- Buffer Overflow
- Stack Overflow
- Heap Overflow
- 覆蓋到理論上不應該被修改到的資料
  - Important data, Secret
  - Return address

BOF

Buffer Overflow

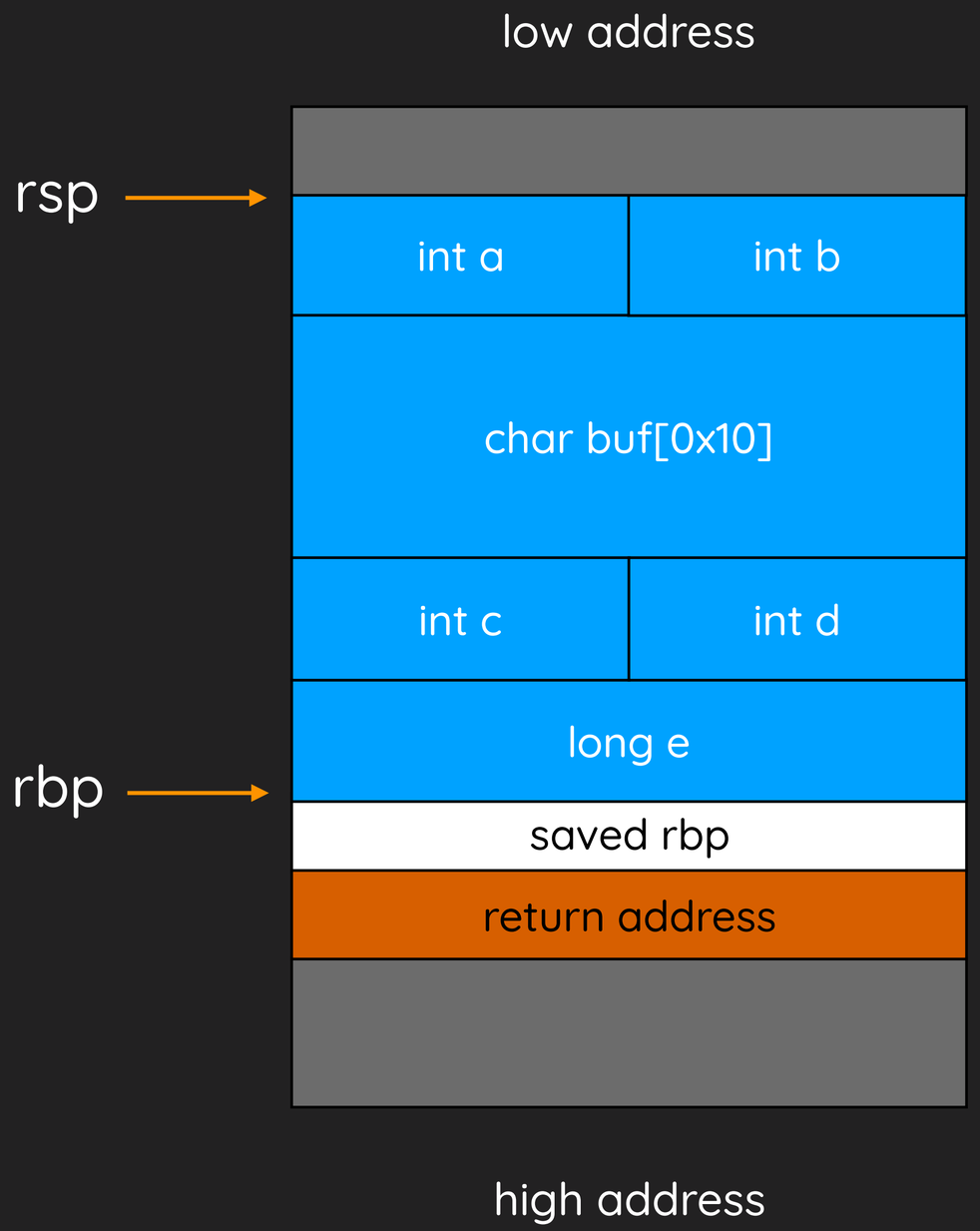
Buffer Overflow

- Local variables
- Data on stack



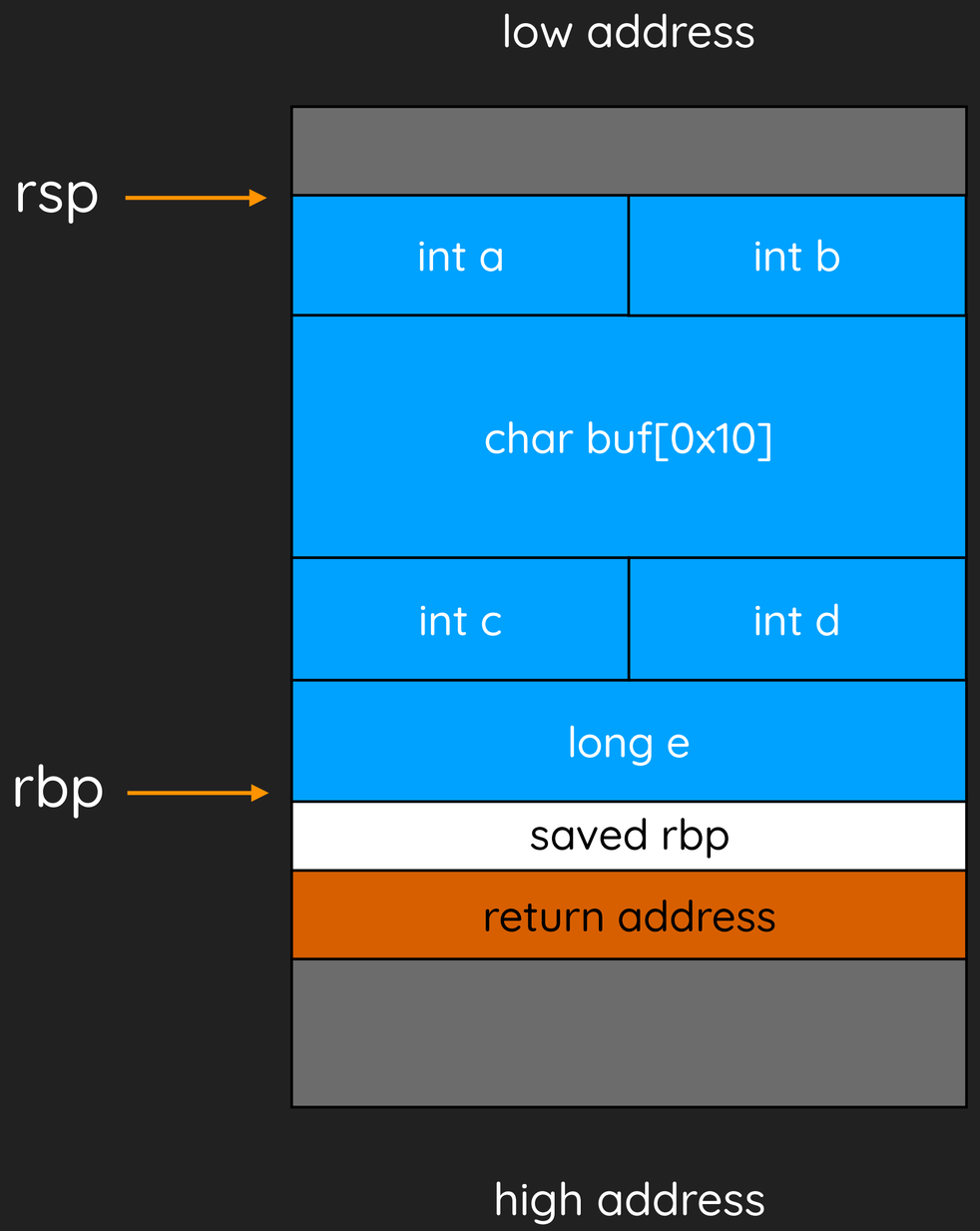
Buffer Overflow

- Local variables
- Data on stack



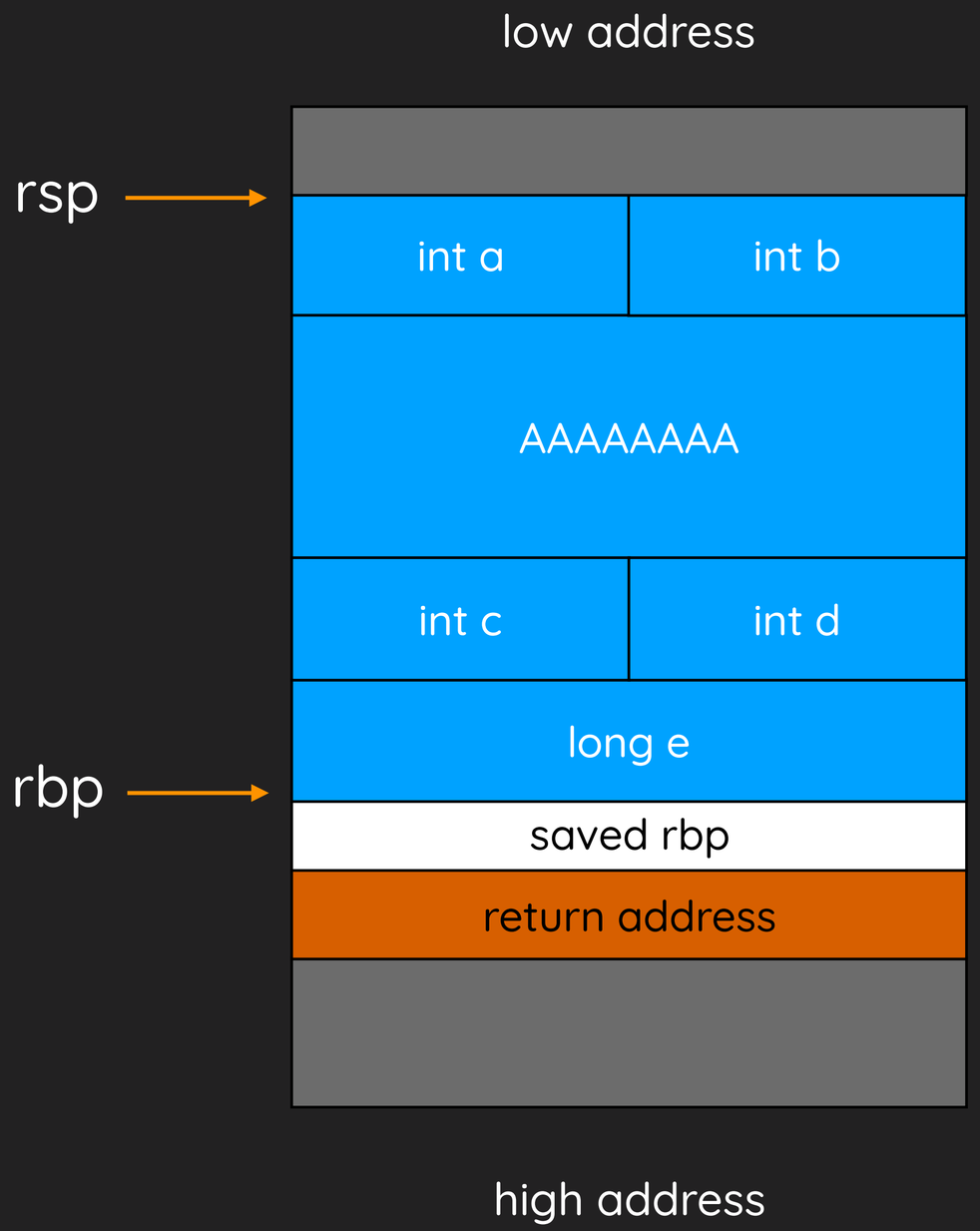
Buffer Overflow

- gets(buf)
- gets() 並不會檢查輸入長度



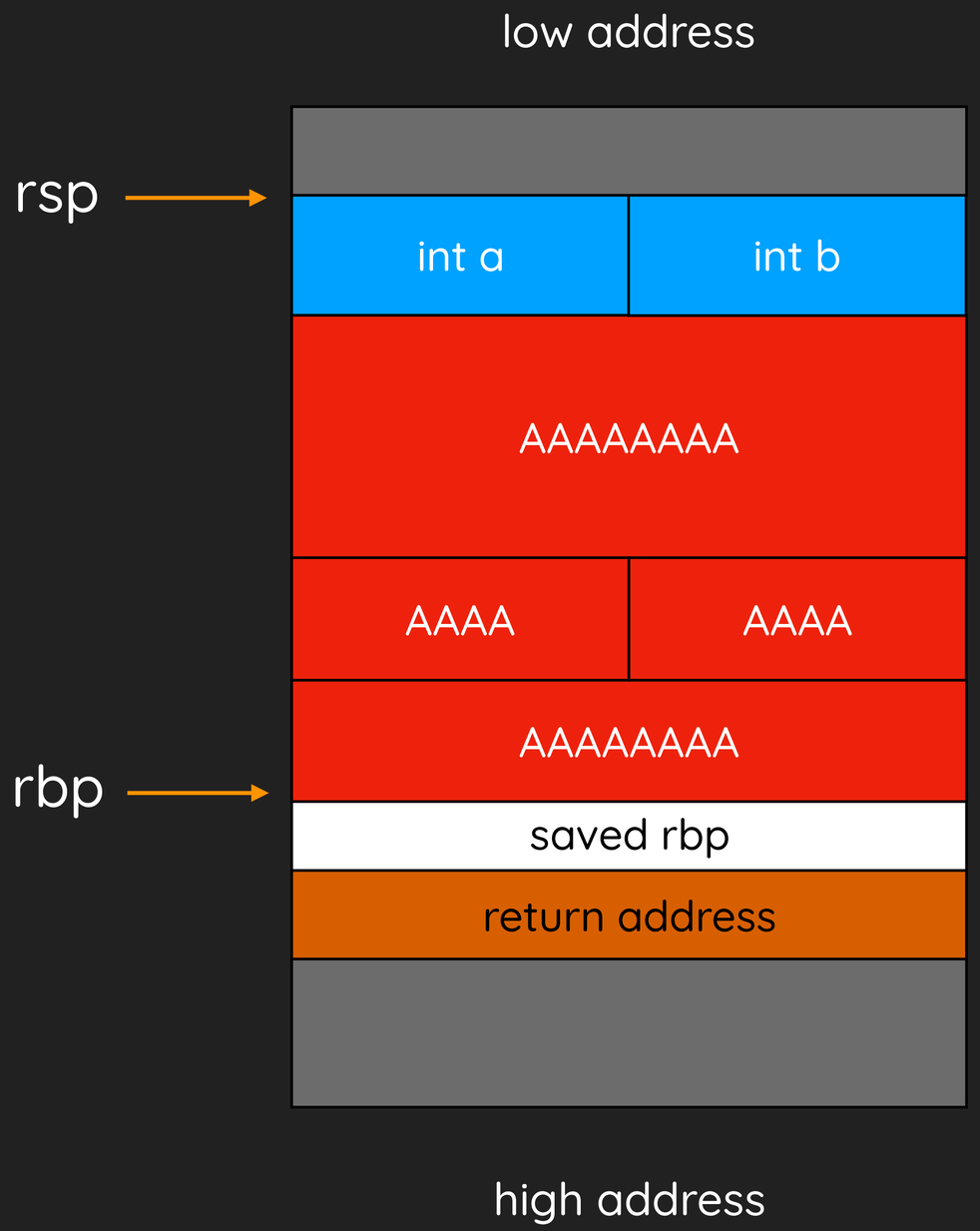
Buffer Overflow

- gets(buf)



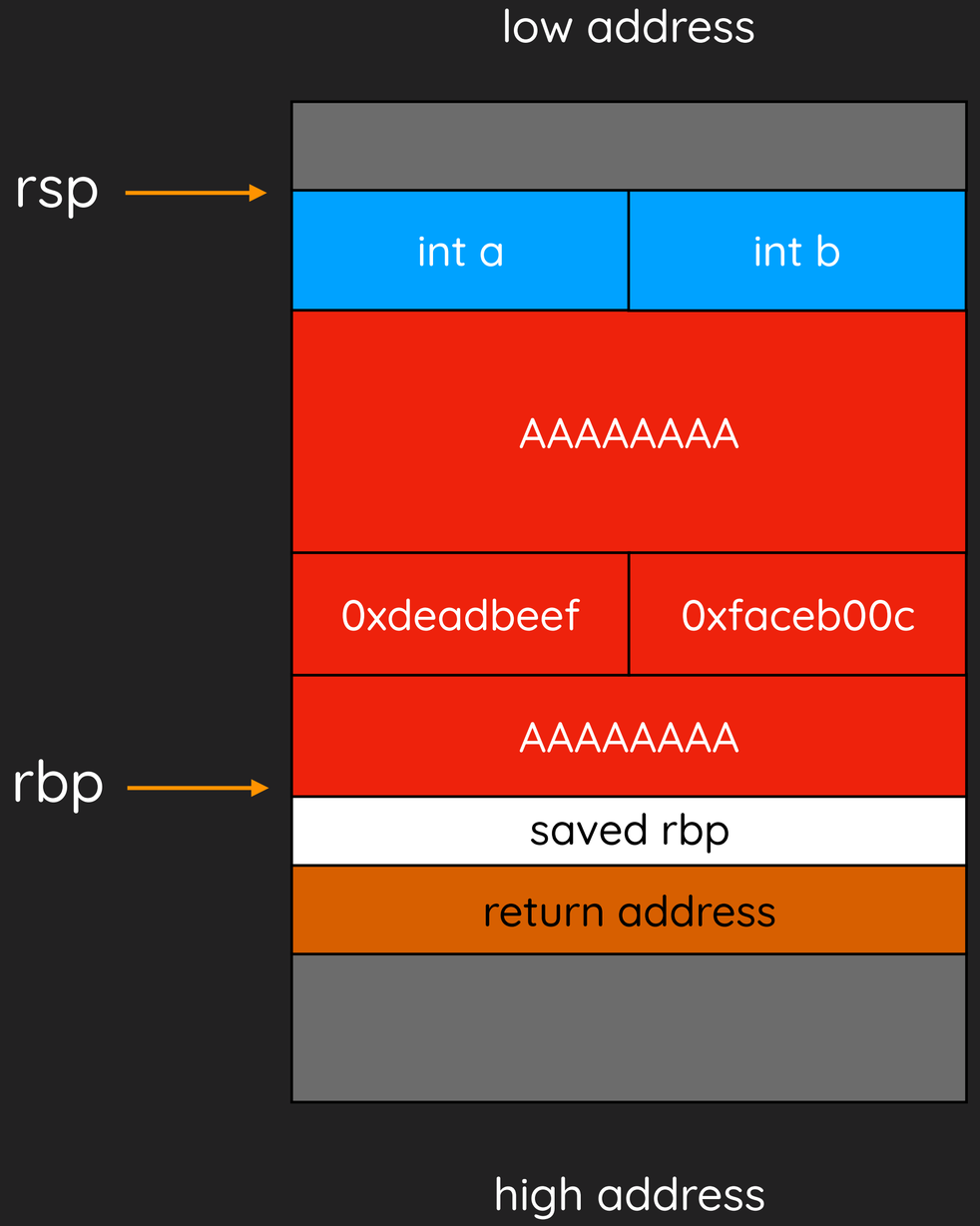
Buffer Overflow

- gets(buf)
- Overflow!



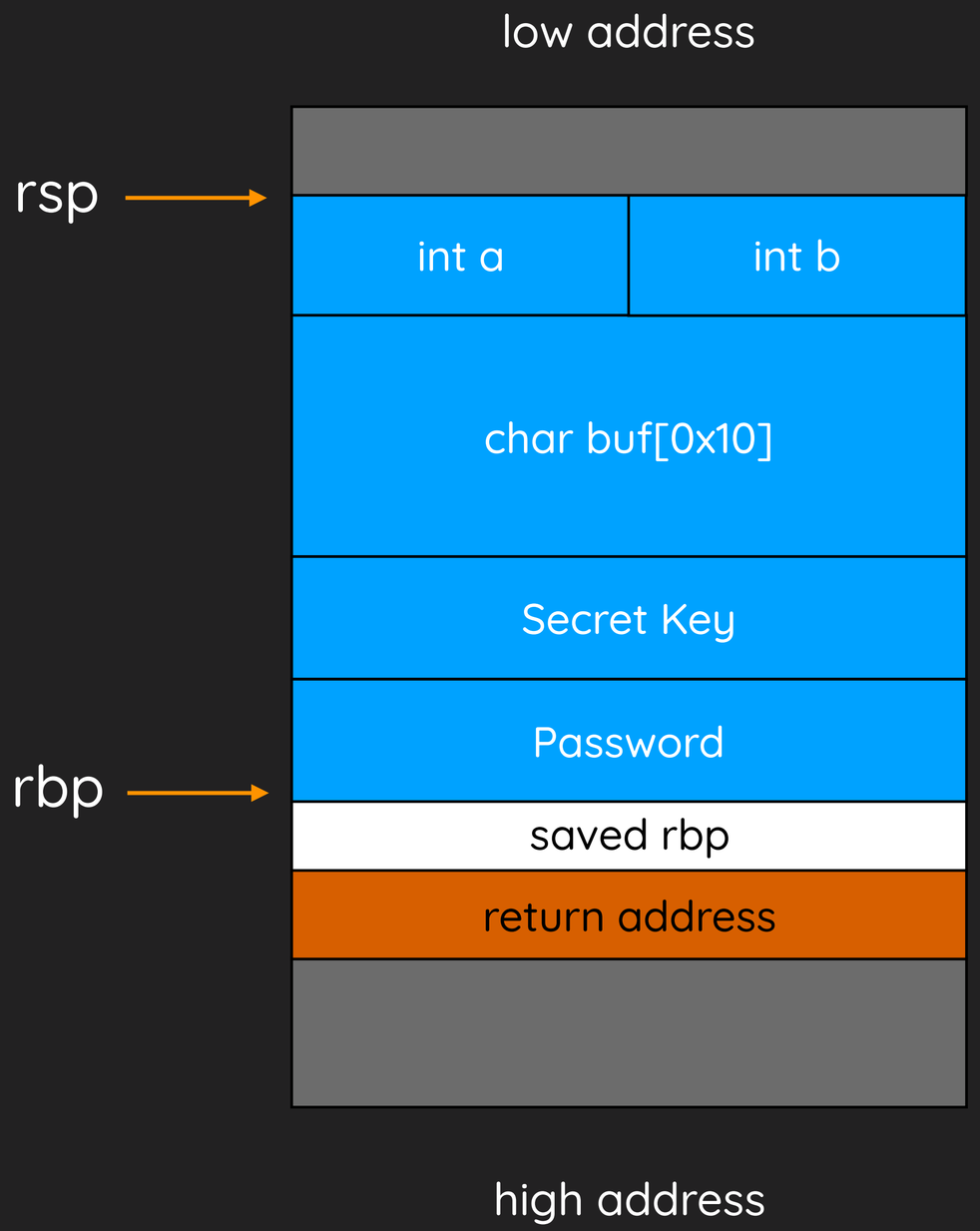
Buffer Overflow

- 控制位於 stack 上的變數值
- int c = 0xdeadbeef
- int d = 0xfaceb00c
- long e = 0x4141414141414141



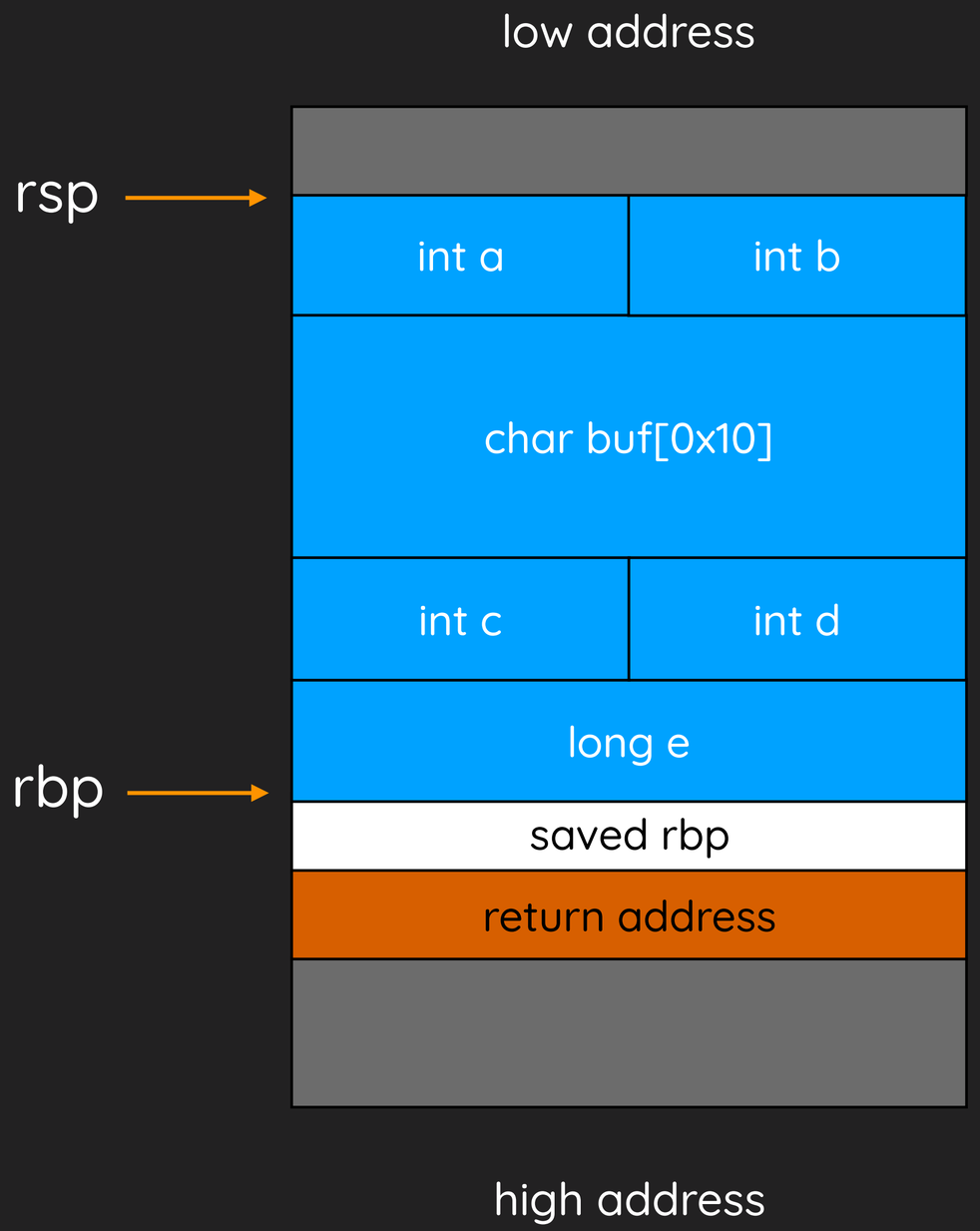
Buffer Overflow

- 控制敏感資料



Buffer Overflow

- Control Flow
- Return address



Buffer Overflow

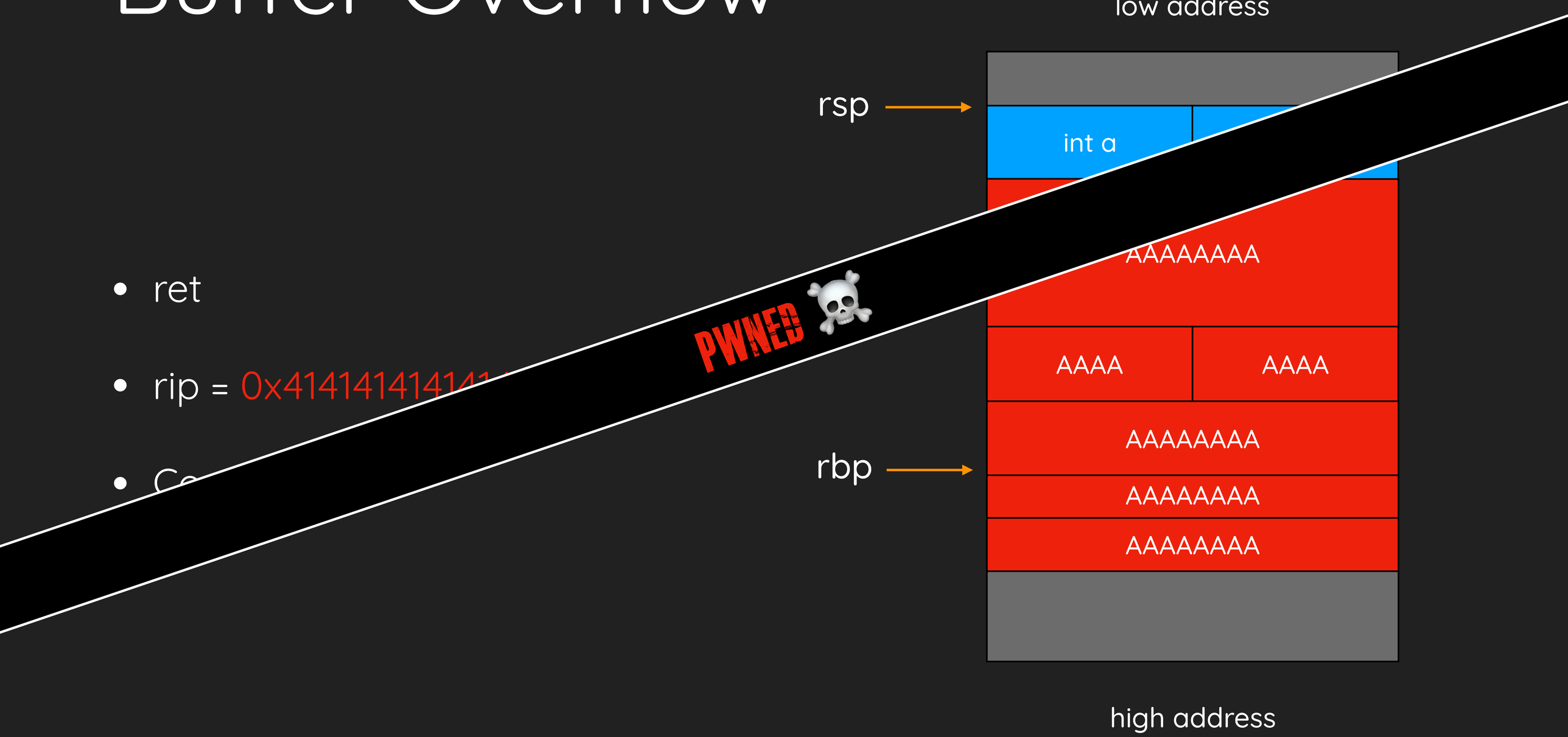
- Control Flow
- Return address

Buffer Overflow

- ret
- rip = 0x4141414141414141
- Control rip

Buffer Overflow

- ret
- rip = 0x414141414141
- Co



Canary

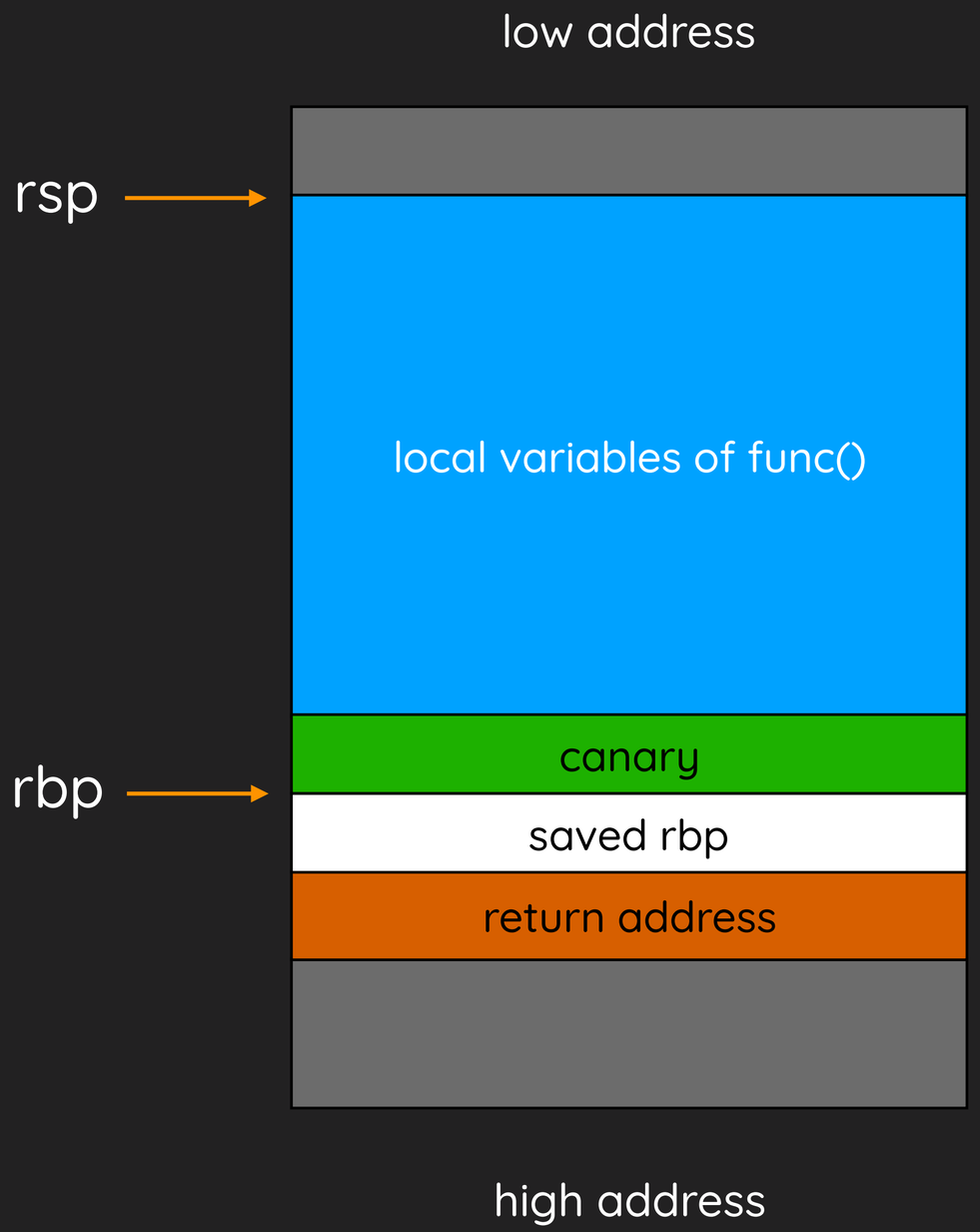
stack protector

Canary

- Function prologue 時在 stack 上放置程式執行時隨機生成得 8 bytes 在 saved rbp 前，第一個 byte 為 null byte

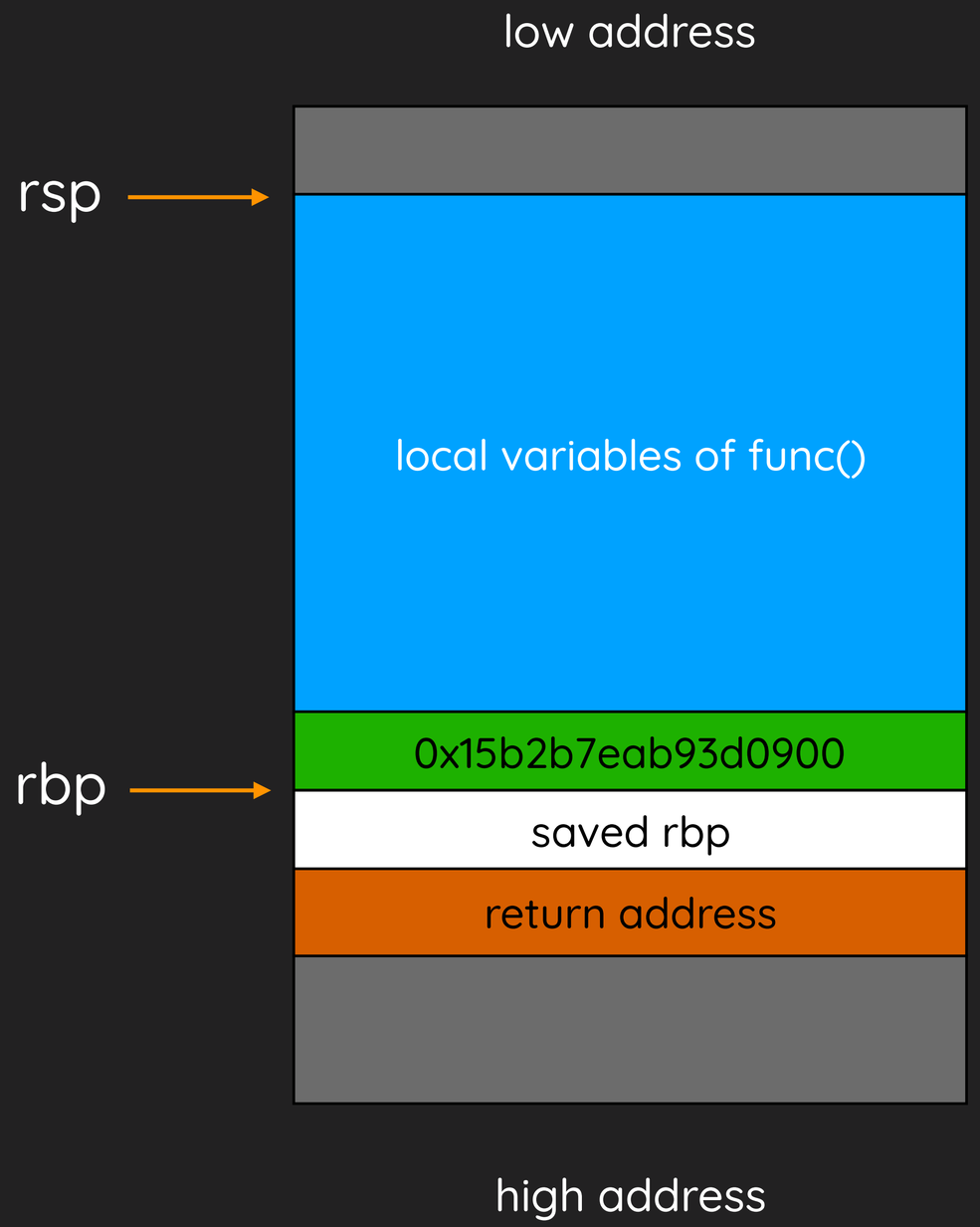
- Function epilogue 時會拿儲存在另一 segment 的值檢查 canary 值是否相同(被修改)來檢測是否發生 overflow，若相同才正常 return，否則直接終止程式 (Abort)

- 每次執行 canary 不同，同一次的 canary 固定



Canary

0x15b2b7eab93d0900



Canary

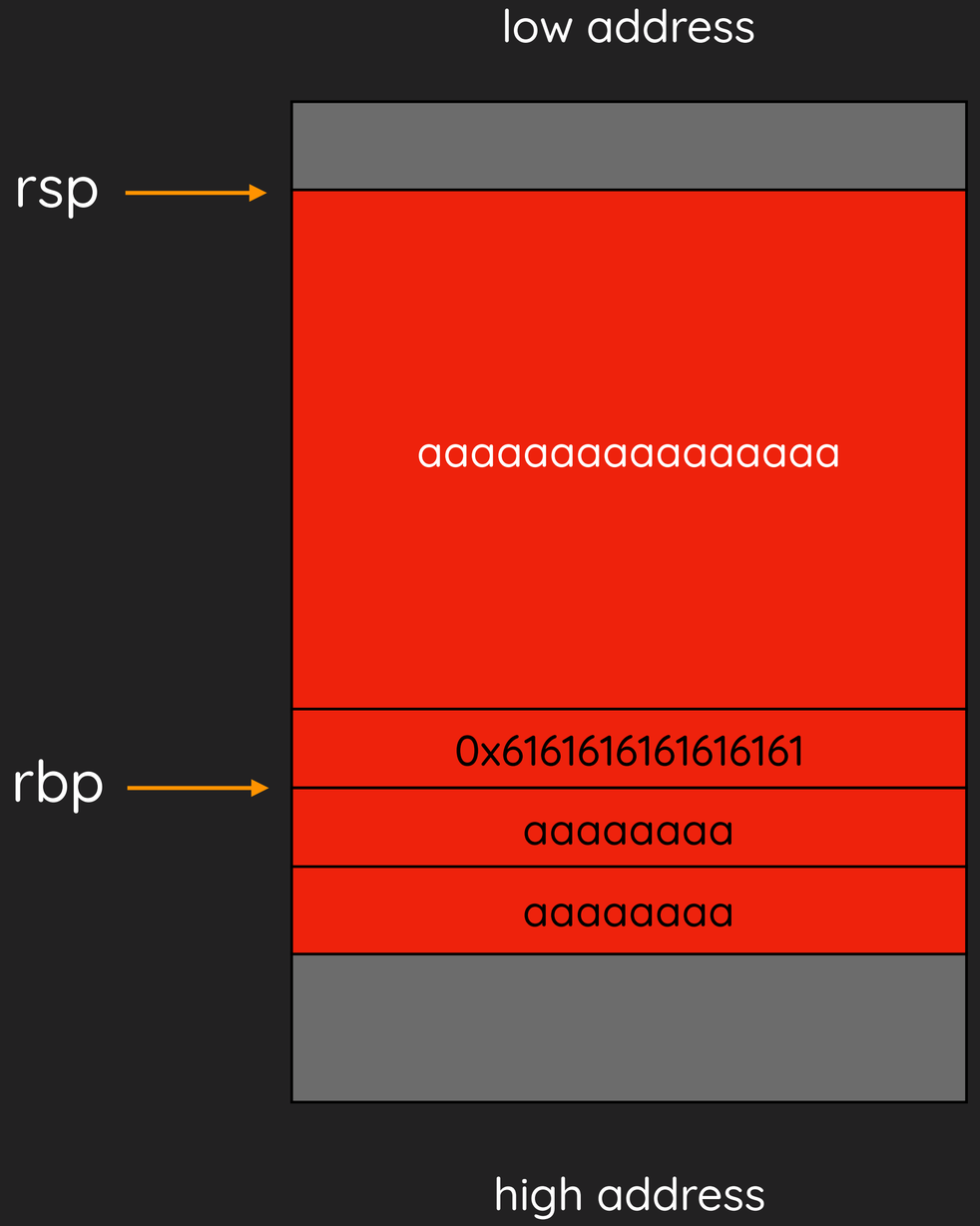
0x15b2b7eab93d0900

- Overflow

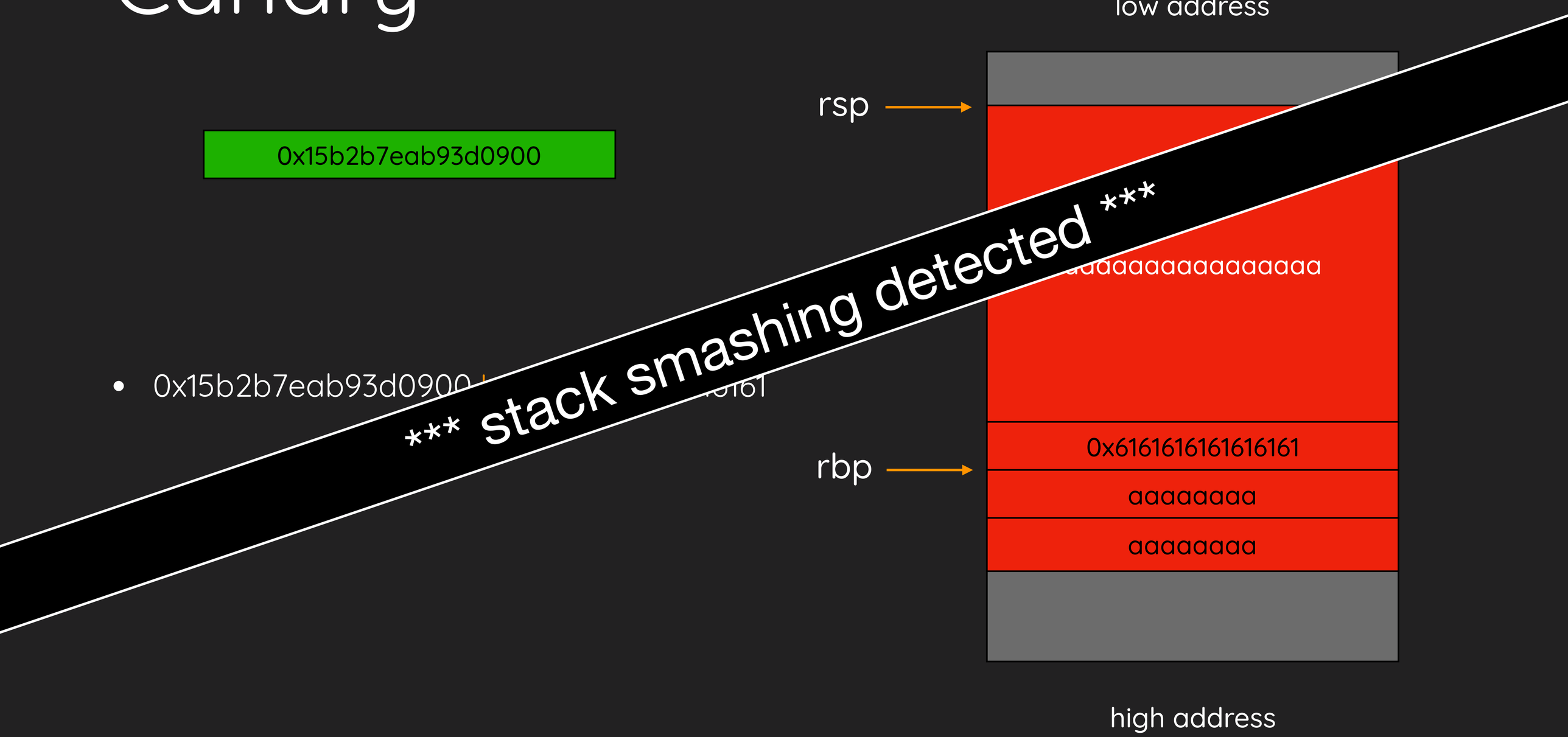
Canary

0x15b2b7eab93d0900

- 0x15b2b7eab93d0900 != 0x6161616161616161



Canary



- 0x15b2b7eab93d0900

DEMO

LAB

Shellcode

Shellcode

```c
int main(){

    puts( "Hello World!" );

    return 0;
}
```

Compiler

Machine Code

```
55 48 89 e5
48 8d 3d 9f
00 00 00 e8
a3 fe ff ff
b8 00 00 00
00 5d c3
```

Shellcode

Compiler

Assembler

```
int main(){

    puts( "Hello World!" );

    return 0;
}
```



```
55                      push   rbp
48 89 e5                mov    rbp,rsp
48 8d 3d 9f 00 00 00    lea    rdi,[rip+0x9f]
e8 a3 fe ff ff          call   550 <puts@plt>
b8 00 00 00 00          mov    eax,0x0
5d                      pop    rbp
c3                      ret
```

Shellcode

Compiler

Assembler

```c
int main(){

    puts( "Hello World!" );

    return 0;
}
```



```
55                      push   rbp
48 89 e5                mov    rbp,rsp
48 8d 3d 9f 00 00 00    lea    rdi,[rip+0x9f]
e8 a3 fe ff ff          call   550 <puts@plt>
b8 00 00 00 00          mov    eax,0x0
5d                      pop    rbp
c3                      ret
```

Let’s write this!

Shellcode

Assembler

```
push   rbp
mov    rbp,rsp
lea    rdi,[rip+0x9f]
call   550 <puts@plt>
mov    eax,0x0
pop    rbp
ret
```

```
55
48 89 e5
48 8d 3d 9f 00 00 00
e8 a3 fe ff ff
b8 00 00 00 00
5d
c3
```

Linux syscall

Syscall

- System call
- 跟 kernel 做溝通的 interface
- x86 - https://syscalls.kernelgrok.com/
- x64 - https://blog.rchapman.org/posts/Linux_System_Call_Table_for_x86_64/

Syscall

- Instruction - syscall
- rax - syscall number
- Arguments - rdi rsi rdx r10 r8 r9
- Return value - rax

read(0 , buf , 0x100)

```
xor     rdi, rdi
mov     rsi, 0x601000 // buf
mov     rdx, 0x100
mov     eax, 0
syscall
```

Shellcode

- execve
- int execve(const char *pathname, char *const argv[], char *const envp[]);
- Spawn a shell!
  - execve(“/bin/sh” , NULL, NULL)

Shellcode

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

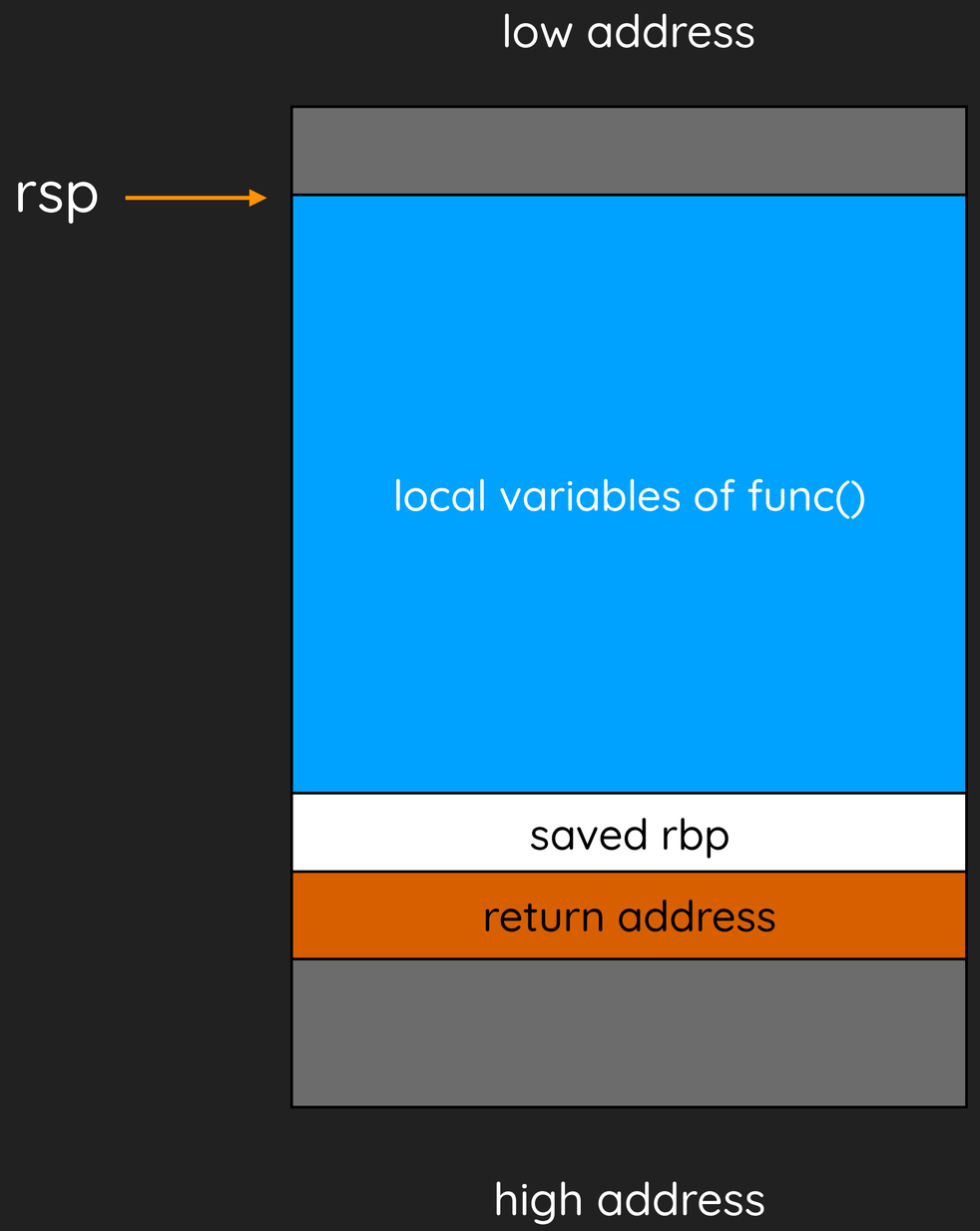


```
rax = 0x3b
```

```
mov     rax, 0x68732f6e69622f // “/bin/sh\0”
push    rax
mov     rdi, rsp
xor     rsi, rsi
xor     rdx, rdx
mov     rax, 0x3b
syscall
```

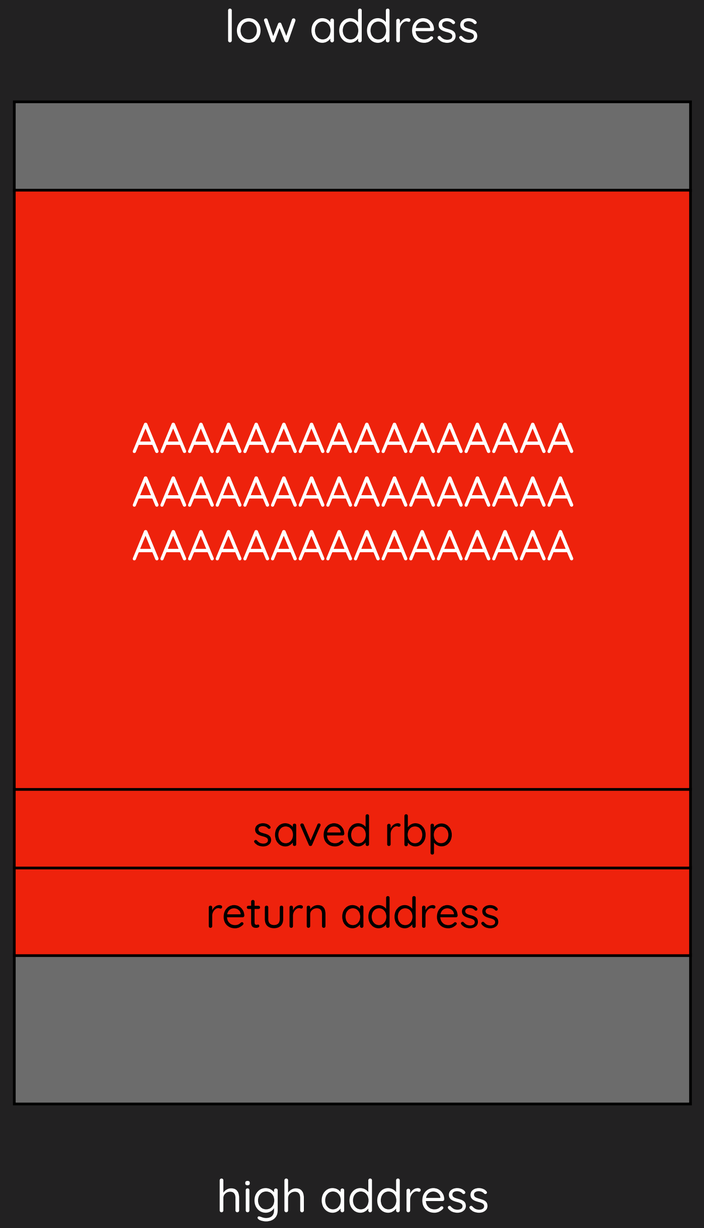
Shellcode

- NX - disable
- Return to shellcode



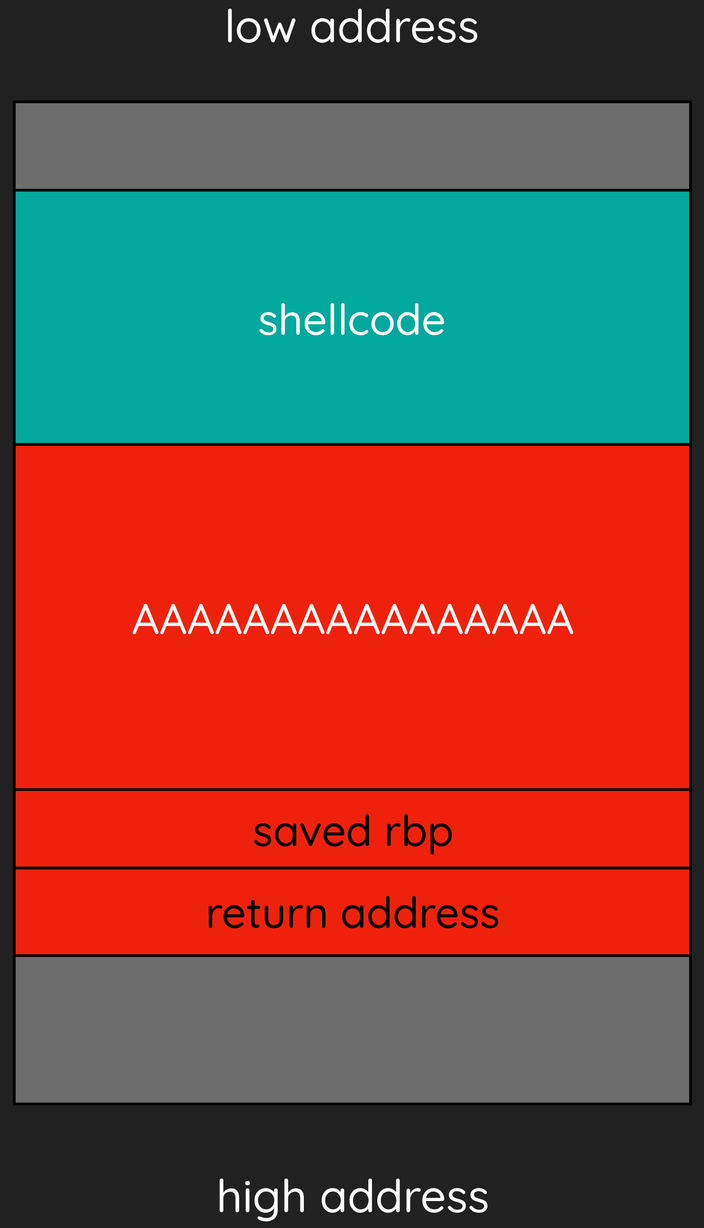
Shellcode

- Overflow



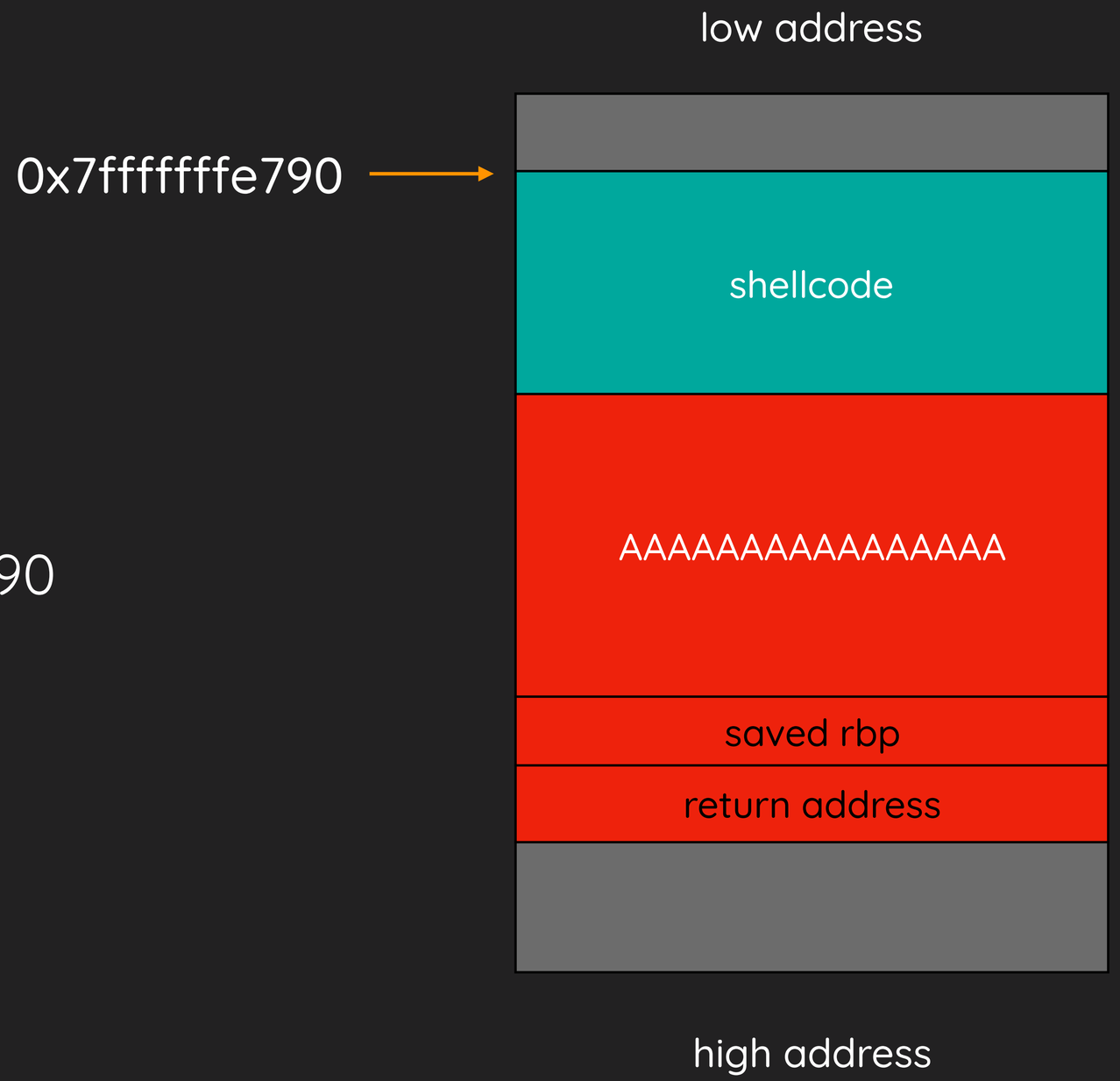
Shellcode

- 放置恶意 payload (shellcode)
- Stack is executable



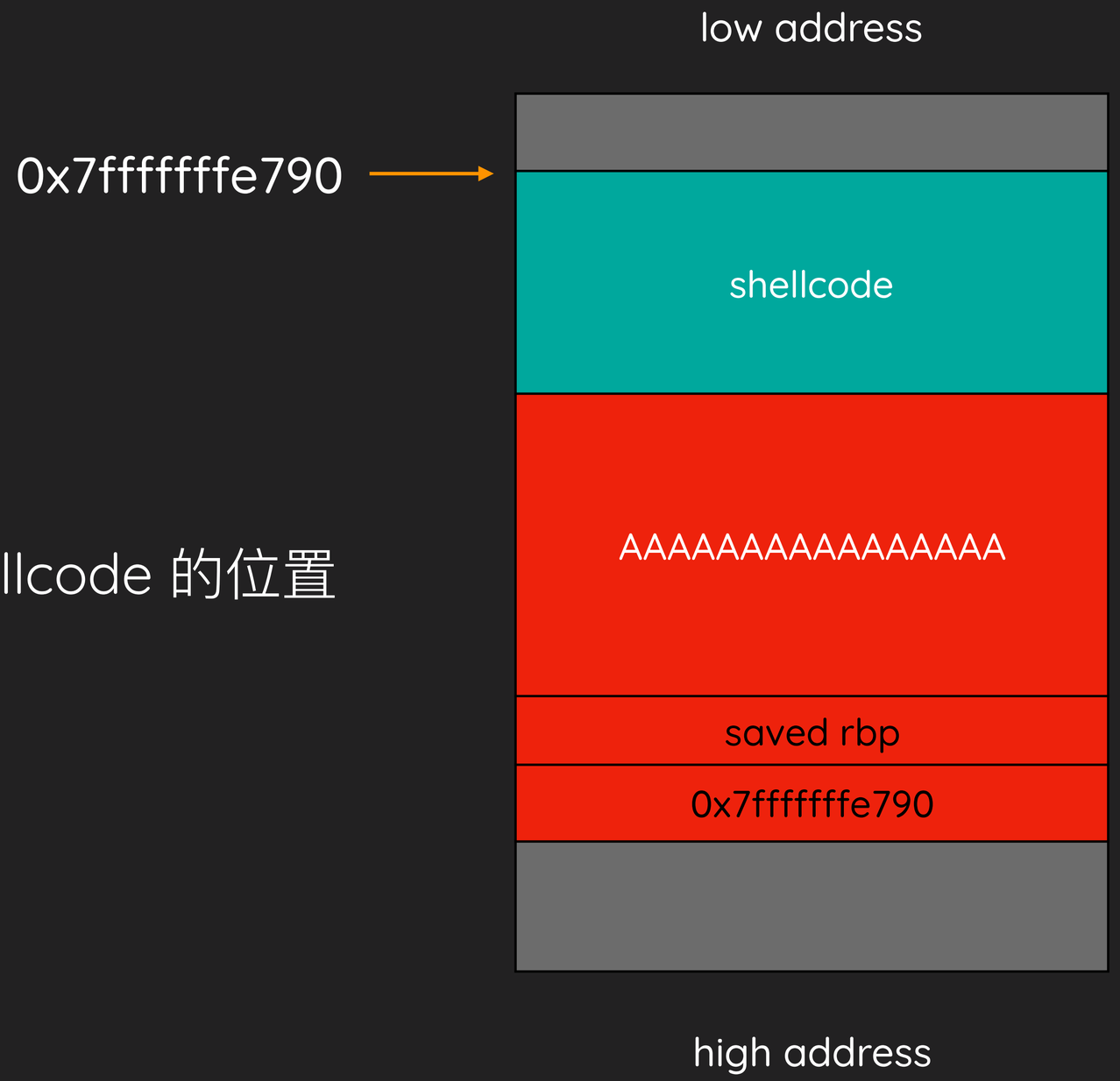
Shellcode

- Shellcode is at 0x7fffffffe790



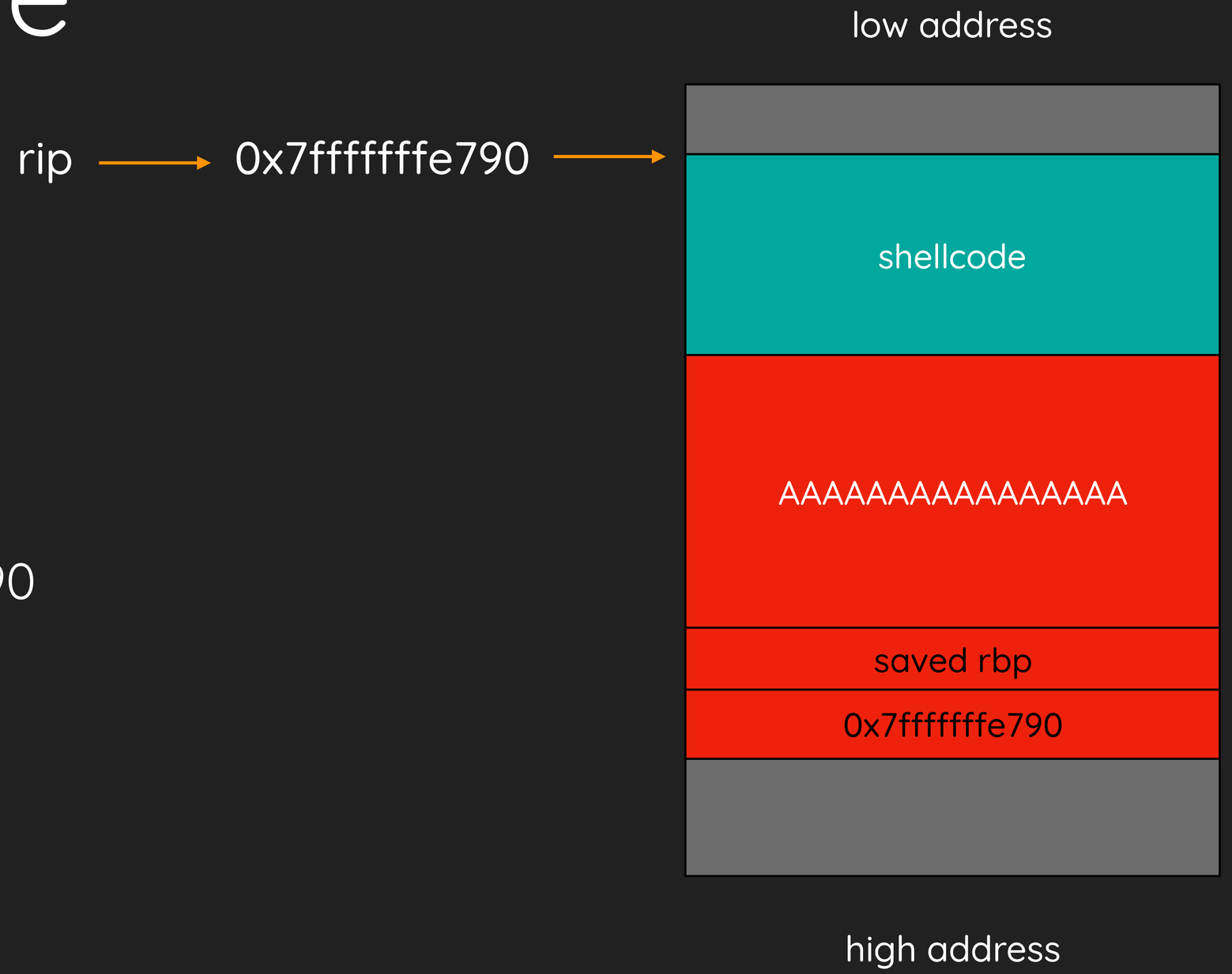
Shellcode

- 覆蓋 return address 成 shellcode 的位置



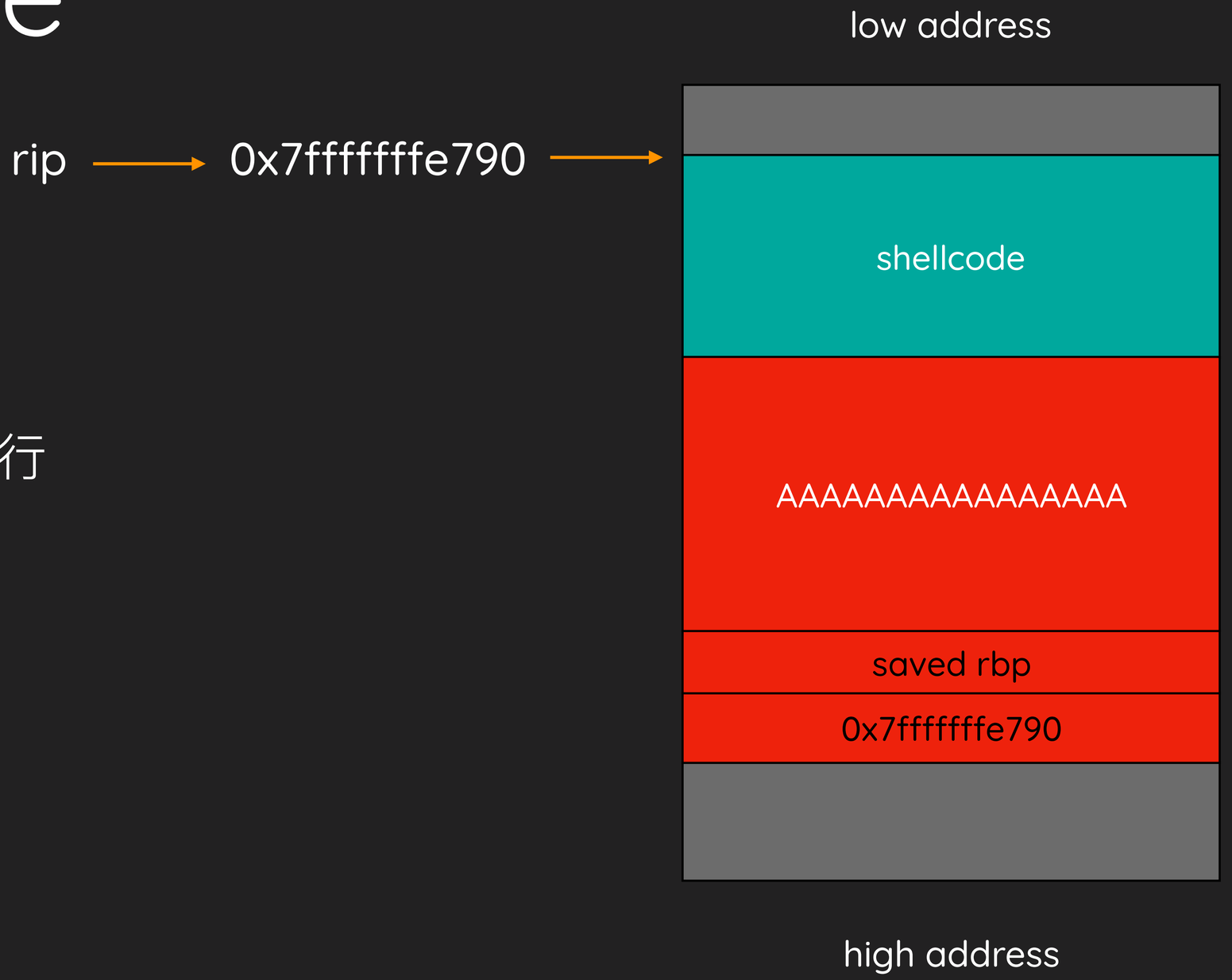
Shellcode

- ret
- rip = 0x7fffffffe790

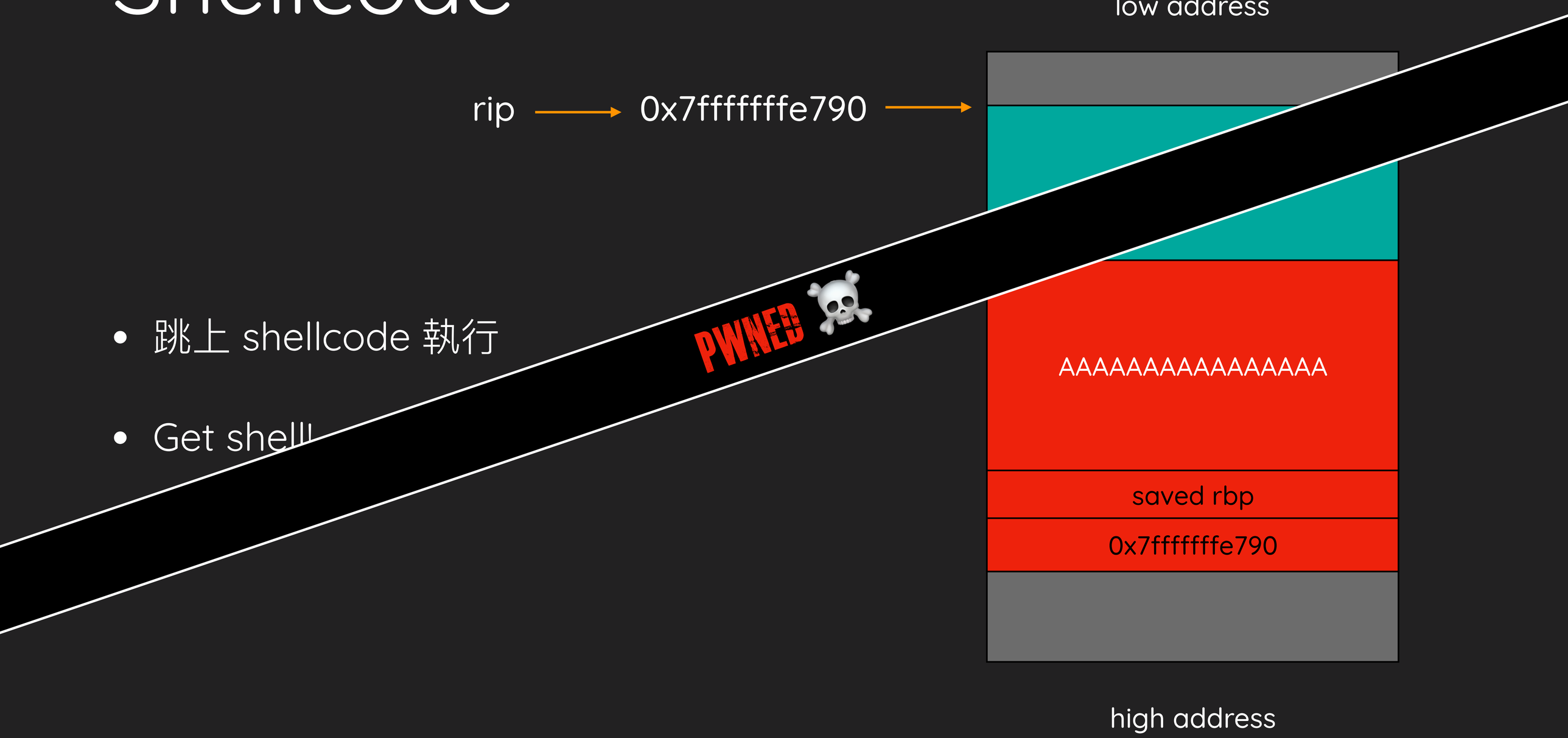


Shellcode

- 跳上 shellcode 執行
- Get shell!



Shellcode



- 跳上 shellcode 執行
- Get shell!

NX

No-execute

NX

- No-Execute
- Data segment 不應該具有執行權限
  - stack heap
  - rw-
- Code segment 具執行權限，但不具寫入權限
  - r-x

NX

```
Start              End                Perm  Name
0x00400000         0x00401000         r-xp  /home/yuawn/binary  .text
0x00600000         0x00601000         r--p  /home/yuawn/binary
0x00601000         0x00602000         rw-p  /home/yuawn/binary  .bss
0x00007f5c39819000 0x00007f5c39a00000 r-xp  /lib/x86_64-linux-gnu/libc-2.27.so
0x00007f5c39a00000 0x00007f5c39c00000 ---p  /lib/x86_64-linux-gnu/libc-2.27.so
0x00007f5c39c00000 0x00007f5c39c04000 r--p  /lib/x86_64-linux-gnu/libc-2.27.so
0x00007f5c39c04000 0x00007f5c39c06000 rw-p  /lib/x86_64-linux-gnu/libc-2.27.so
0x00007f5c39c06000 0x00007f5c39c0a000 rw-p  mapped
0x00007f5c39c0a000 0x00007f5c39c31000 r-xp  /lib/x86_64-linux-gnu/ld-2.27.so
0x00007f5c39e26000 0x00007f5c39e28000 rw-p  mapped
0x00007f5c39e31000 0x00007f5c39e32000 r--p  /lib/x86_64-linux-gnu/ld-2.27.so
0x00007f5c39e32000 0x00007f5c39e33000 rw-p  /lib/x86_64-linux-gnu/ld-2.27.so
0x00007f5c39e33000 0x00007f5c39e34000 rw-p  mapped
0x00007ffdb0f84000 0x00007ffdb0fa5000 rw-p  [stack]
0x00007ffdb0ff6000 0x00007ffdb0ff8000 r--p  [vvar]
0x00007ffdb0ff8000 0x00007ffdb0ffa000 r-xp  [vdso]
0xffffffffff600000 0xffffffffff601000 r-xp  [vsyscall]
```

DEMO

ASLR

Address Space Layout Randomization

ASLR

- Address Space Layout Randomization
- kernel
- 每次動態載入時，base 都是隨機的
  - library
  - stack
  - heap

PIE

Position-Independent Executable

PIE

- 可以看成是 ELF code & data section map 到 virtual address 時的 ASLR。
- PIE 開啟時，每次執行程式 code base 都會不同，否則固定 0x400000
- 紀錄在 ELF file 中

Lazy Binding

Lazy Binding

- Dynamic linking 的程式，有些使用到的 library function 可能因執行流程到結束都不會被執行到。
- Lazy binding 機制為當程式第一次呼叫 library function 時，才會去第一次尋找 libc function 的位置(function address)進行 binding，並填入 GOT 表中，後續呼叫此 function 則直接從 GOT 表中獲取位置。

GOT

Global Offset Table

GOT

- library 的位置在載入時才決定，compiler 在編譯時期亦無法得知執行時期的 library function address。
- GOT 為儲存 library function 位置的指標陣列，而 lazy binding 的機制，一開始不會得知真實位置，而是先填入位於 plt 的 code。

Lazy Binding

```
00000000004003e0 <.plt>:
  4003e0:   push   QWORD PTR [rip+0x200c22]        # 601008 <got+0x8>
  4003e6:   jmp    QWORD PTR [rip+0x200c24]        # 601010 <got+0x10>

00000000004003f0 <puts@plt>:
  4003f0:   jmp    QWORD PTR [rip+0x200c22]        # 601018 <puts@got>
  4003f6:   push   0x0
  4003fb:   jmp    4003e0 <.plt>


  ..

  4004f2:   call   4003f0 <puts@plt>

  ..

  400700:   call   4003f0 <puts@plt>
```

GOT

0x601010	0x0	puts@plt+6
0x601020	printf@plt+6	read@plt+6
0x601030	write@plt+6	system@plt+6

Lazy Binding

```
00000000004003e0 <.plt>:
  4003e0:   push   QWORD PTR [rip+0x200c22]        # 601008 <got+0x8>
  4003e6:   jmp    QWORD PTR [rip+0x200c24]        # 601010 <got+0x10>

00000000004003f0 <puts@plt>:
  4003f0:   jmp    QWORD PTR [rip+0x200c22]        # 601018 <puts@got>
  4003f6:   push   0x0
  4003fb:   jmp    4003e0 <.plt>


  ..

  4004f2:   call   4003f0 <puts@plt>

  ..

  400700:   call   4003f0 <puts@plt>
```

GOT

0x601010	0x0	puts@plt+6
0x601020	printf@plt+6	read@plt+6
0x601030	write@plt+6	system@plt+6

Lazy Binding

```
00000000004003e0 <.plt>:
  4003e0:   push   QWORD PTR [rip+0x200c22]        # 601008 <got+0x8>
  4003e6:   jmp    QWORD PTR [rip+0x200c24]        # 601010 <got+0x10>

00000000004003f0 <puts@plt>:
  4003f0:   jmp    QWORD PTR [rip+0x200c22]        # 601018 <puts@got>
  4003f6:   push   0x0
  4003fb:   jmp    4003e0 <.plt>


  ..

  4004f2:   call   4003f0 <puts@plt>

  ..

  400700:   call   4003f0 <puts@plt>
```

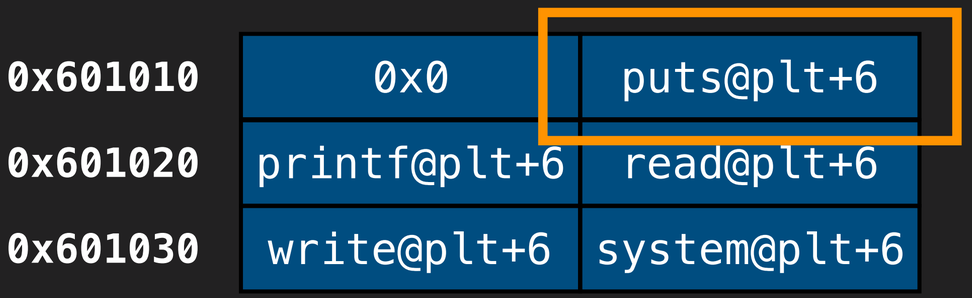
GOT

0x601010	0x0	puts@plt+6
0x601020	printf@plt+6	read@plt+6
0x601030	write@plt+6	system@plt+6

Lazy Binding

```
00000000004003e0 <.plt>:
  4003e0:   push   QWORD PTR [rip+0x200c22]        # 601008 <got+0x8>
  4003e6:   jmp    QWORD PTR [rip+0x200c24]        # 601010 <got+0x10>

00000000004003f0 <puts@plt>:
  4003f0:   jmp    QWORD PTR [rip+0x200c22]        # 601018 <puts@got>
  4003f6:   push   0x0
  4003fb:   jmp    4003e0 <.plt>


  ..

  4004f2:   call   4003f0 <puts@plt>

  ..

  400700:   call   4003f0 <puts@plt>
```

GOT



Lazy Binding

```
00000000004003e0 <.plt>:
  4003e0:   push   QWORD PTR [rip+0x200c22]        # 601008 <got+0x8>
  4003e6:   jmp    QWORD PTR [rip+0x200c24]        # 601010 <got+0x10>

00000000004003f0 <puts@plt>:
  4003f0:   jmp    QWORD PTR [rip+0x200c22]        # 601018 <puts@got>
  4003f6:   push   0x0
  4003fb:   jmp    4003e0 <.plt>


  ..

  4004f2:   call   4003f0 <puts@plt>

  ..

  400700:   call   4003f0 <puts@plt>
```

GOT

0x601010	0x0	puts@plt+6
0x601020	printf@plt+6	read@plt+6
0x601030	write@plt+6	system@plt+6

Lazy Binding

```
00000000004003e0 <.plt>:
  4003e0:   push   QWORD PTR [rip+0x200c22]        # 601008 <got+0x8>
  4003e6:   jmp    QWORD PTR [rip+0x200c24]        # 601010 <got+0x10>

00000000004003f0 <puts@plt>:
  4003f0:   jmp    QWORD PTR [rip+0x200c22]        # 601018 <puts@got>
  4003f6:   push   0x0
  4003fb:   jmp    4003e0 <.plt>


  ..

  4004f2:   call   4003f0 <puts@plt>

  ..

  400700:   call   4003f0 <puts@plt>
```

GOT

0x601010	0x0	puts@plt+6
0x601020	printf@plt+6	read@plt+6
0x601030	write@plt+6	system@plt+6

Lazy Binding

```
00000000004003e0 <.plt>:
  4003e0:   push   QWORD PTR [rip+0x200c22]        # 601008 <got+0x8>
  4003e6:   jmp    QWORD PTR [rip+0x200c24]        # 601010 <got+0x10>

00000000004003f0 <puts@plt>:
  4003f0:   jmp    QWORD PTR [rip+0x200c22]        # 601018 <puts@got>
  4003f6:   push   0x0
  4003fb:   jmp    4003e0 <.plt>


  ..

  4004f2:   call   4003f0 <puts@plt>

  ..

  400700:   call   4003f0 <puts@plt>
```

GOT

0x601010	0x0	puts@plt+6
0x601020	printf@plt+6	read@plt+6
0x601030	write@plt+6	system@plt+6

Lazy Binding

```
00000000004003e0 <.plt>:
  4003e0:   push   QWORD PTR [rip+0x200c22]        # 601008 <got+0x8>
  4003e6:   jmp    QWORD PTR [rip+0x200c24]        # 601010 <got+0x10>

00000000004003f0 <puts@plt>:
  4003f0:   jmp    QWORD PTR [rip+0x200c22]        # 601018 <puts@got>
  4003f6:   push   0x0
  4003fb:   jmp    4003e0 <.plt>


  ..

  4004f2:   call   4003f0 <puts@plt>

  ..

  400700:   call   4003f0 <puts@plt>
```

GOT

0x601010	0x0	puts@plt+6
0x601020	printf@plt+6	read@plt+6
0x601030	write@plt+6	system@plt+6

Lazy Binding

```
00000000004003e0 <.plt>:
  4003e0:   push   QWORD PTR [rip+0x200c22]        # 601008 <got+0x8>
  4003e6:   jmp    QWORD PTR [rip+0x200c24]        # 601010 <got+0x10>

00000000004003f0 <puts@plt>:
  4003f0:   jmp    QWORD PTR [rip+0x200c22]        # 601018 <puts@got>
  4003f6:   push   0x0
  4003fb:   jmp    4003e0 <.plt>


  ..

  4004f2:   call   4003f0 <puts@plt>

  ..

  400700:   call   4003f0 <puts@plt>
```

GOT

0x601010	0x0	puts@plt+6
0x601020	printf@plt+6	read@plt+6
0x601030	write@plt+6	system@plt+6

Lazy Binding

```
00000000004003e0 <.plt>:
  4003e0:   push   QWORD PTR [rip+0x200c22]        # 601008 <got+0x8>
  4003e6:   jmp    QWORD PTR [rip+0x200c24]        # 601010 <got+0x10>

00000000004003f0 <puts@plt>:
  4003f0:   jmp    QWORD PTR [rip+0x200c22]        # 601018 <puts@got>
  4003f6:   push   0x0
  4003fb:   jmp    4003e0 <.plt>


  ..

  4004f2:   call   4003f0 <puts@plt>

  ..

  400700:   call   4003f0 <puts@plt>
```

_dl_runtime_resolve_xsave

GOT

0x601010	0x0	puts@plt+6
0x601020	printf@plt+6	read@plt+6
0x601030	write@plt+6	system@plt+6

Lazy Binding

```
00000000004003e0 <.plt>:
  4003e0:   push   QWORD PTR [rip+0x200c22]        # 601008 <got+0x8>
  4003e6:   jmp    QWORD PTR [rip+0x200c24]        # 601010 <got+0x10>

00000000004003f0 <puts@plt>:
  4003f0:   jmp    QWORD PTR [rip+0x200c22]        # 601018 <puts@got>
  4003f6:   push   0x0
  4003fb:   jmp    4003e0 <.plt>


  ..

  4004f2:   call   4003f0 <puts@plt>

  ..

  400700:   call   4003f0 <puts@plt>
```

_dl_runtime_resolve_xsave

GOT

0x601010	0x0	0x7ffff7a649c0
0x601020	printf@plt+6	read@plt+6
0x601030	write@plt+6	system@plt+6

Lazy Binding

```
00000000004003e0 <.plt>:
  4003e0:   push   QWORD PTR [rip+0x200c22]        # 601008 <got+0x8>
  4003e6:   jmp    QWORD PTR [rip+0x200c24]        # 601010 <got+0x10>

00000000004003f0 <puts@plt>:
  4003f0:   jmp    QWORD PTR [rip+0x200c22]        # 601018 <puts@got>
  4003f6:   push   0x0
  4003fb:   jmp    4003e0 <.plt>


  ..

  4004f2:   call   4003f0 <puts@plt>

  ..

  400700:   call   4003f0 <puts@plt>
```

_dl_runtime_resolve_xsave

GOT

0x601010	0x0	<_IO_puts>
0x601020	printf@plt+6	read@plt+6
0x601030	write@plt+6	system@plt+6

Lazy Binding

```
00000000004003e0 <.plt>:
  4003e0:   push   QWORD PTR [rip+0x200c22]        # 601008 <got+0x8>
  4003e6:   jmp    QWORD PTR [rip+0x200c24]        # 601010 <got+0x10>

00000000004003f0 <puts@plt>:
  4003f0:   jmp    QWORD PTR [rip+0x200c22]        # 601018 <puts@got>
  4003f6:   push   0x0
  4003fb:   jmp    4003e0 <.plt>


  ..

  4004f2:   call   4003f0 <puts@plt>

  ..

  400700:   call   4003f0 <puts@plt>
```

GOT

0x601010	0x0	<_IO_puts>
0x601020	printf@plt+6	read@plt+6
0x601030	write@plt+6	system@plt+6

Lazy Binding

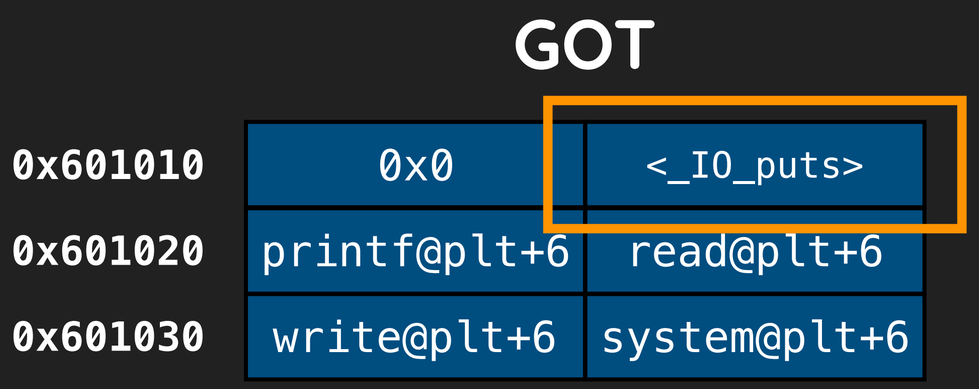
```
00000000004003e0 <.plt>:
  4003e0:   push   QWORD PTR [rip+0x200c22]        # 601008 <got+0x8>
  4003e6:   jmp    QWORD PTR [rip+0x200c24]        # 601010 <got+0x10>

00000000004003f0 <puts@plt>:
  4003f0:   jmp    QWORD PTR [rip+0x200c22]        # 601018 <puts@got>
  4003f6:   push   0x0
  4003fb:   jmp    4003e0 <.plt>


  ..

  4004f2:   call   4003f0 <puts@plt>

  ..

  400700:   call   4003f0 <puts@plt>
```

GOT

0x601010	0x0	<_IO_puts>
0x601020	printf@plt+6	read@plt+6
0x601030	write@plt+6	system@plt+6

Lazy Binding

```
00000000004003e0 <.plt>:
  4003e0:   push   QWORD PTR [rip+0x200c22]        # 601008 <got+0x8>
  4003e6:   jmp    QWORD PTR [rip+0x200c24]        # 601010 <got+0x10>

00000000004003f0 <puts@plt>:
  4003f0:   jmp    QWORD PTR [rip+0x200c22]        # 601018 <puts@got>
  4003f6:   push   0x0
  4003fb:   jmp    4003e0 <.plt>


  ..

  4004f2:   call   4003f0 <puts@plt>

  ..

  400700:   call   4003f0 <puts@plt>
```



Lazy Binding

```
00000000004003e0 <.plt>:
  4003e0:   push   QWORD PTR [rip+0x200c22]        # 601008 <got+0x8>
  4003e6:   jmp    QWORD PTR [rip+0x200c24]        # 601010 <got+0x10>

00000000004003f0 <puts@plt>:
  4003f0:   jmp    QWORD PTR [rip+0x200c22]        # 601018 <puts@got>
  4003f6:   push   0x0
  4003fb:   jmp    4003e0 <.plt>
```

0x7ffff7a649c0 <_IO_puts>

```
  ..

  4004f2:   call   4003f0 <puts@plt>

  ..

  400700:   call   4003f0 <puts@plt>
```

GOT

0x601010	0x0	<_IO_puts>
0x601020	printf@plt+6	read@plt+6
0x601030	write@plt+6	system@plt+6

DEMO

GOTHijacking

GOT Hijacking

- 因為 Lazy Binding 的機制，GOT 為可寫區域
- 假設程式有漏洞可以造成對 GOT 做寫入覆蓋其值，下一次呼叫對應的 library function 時則可以從中劫持，任意控制將要執行的 funtion pointer。

GOT Hijacking

```
00000000004003e0 <.plt>:
  4003e0:   push   QWORD PTR [rip+0x200c22]        # 601008 <got+0x8>
  4003e6:   jmp    QWORD PTR [rip+0x200c24]        # 601010 <got+0x10>

00000000004003f0 <puts@plt>:
  4003f0:   jmp    QWORD PTR [rip+0x200c22]        # 601018 <puts@got>
  4003f6:   push   0x0
  4003fb:   jmp    4003e0 <.plt>


  ..

  4004f2:   call   4003f0 <puts@plt>
```

GOT

0x601010	0x0	puts@plt+6
0x601020	printf@plt+6	read@plt+6
0x601030	write@plt+6	system@plt+6

GOT Hijacking

```
00000000004003e0 <.plt>:
  4003e0:   push   QWORD PTR [rip+0x200c22]        # 601008 <got+0x8>
  4003e6:   jmp    QWORD PTR [rip+0x200c24]        # 601010 <got+0x10>

00000000004003f0 <puts@plt>:
  4003f0:   jmp    QWORD PTR [rip+0x200c22]        # 601018 <puts@got>
  4003f6:   push   0x0
  4003fb:   jmp    4003e0 <.plt>


  ..

  4004f2:   call   4003f0 <puts@plt>
```

GOT

0x601010	0x0	0xdeadbeef
0x601020	printf@plt+6	read@plt+6
0x601030	write@plt+6	system@plt+6

GOT Hijacking

```
00000000004003e0 <.plt>:
  4003e0:   push   QWORD PTR [rip+0x200c22]        # 601008 <got+0x8>
  4003e6:   jmp    QWORD PTR [rip+0x200c24]        # 601010 <got+0x10>

00000000004003f0 <puts@plt>:
  4003f0:   jmp    QWORD PTR [rip+0x200c22]        # 601018 <puts@got>
  4003f6:   push   0x0
  4003fb:   jmp    4003e0 <.plt>


  ..

  4004f2:   call   4003f0 <puts@plt>
```

GOT

0x601010	0x0	0xdeadbeef
0x601020	printf@plt+6	read@plt+6
0x601030	write@plt+6	system@plt+6

GOT Hijacking

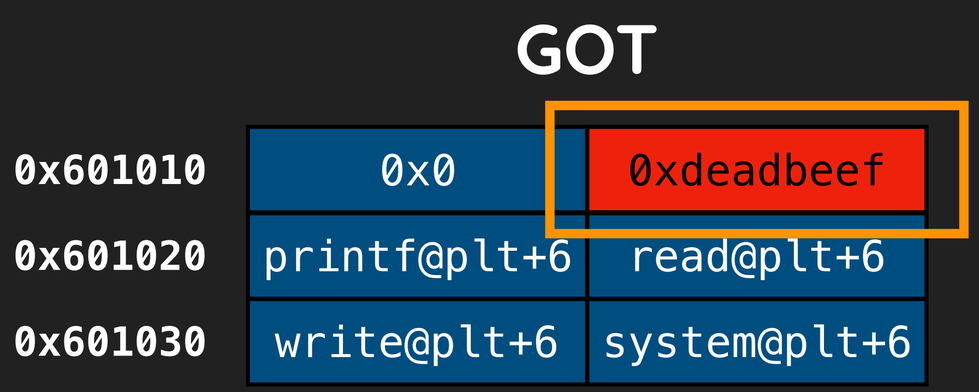
```
00000000004003e0 <.plt>:
  4003e0:   push   QWORD PTR [rip+0x200c22]        # 601008 <got+0x8>
  4003e6:   jmp    QWORD PTR [rip+0x200c24]        # 601010 <got+0x10>

00000000004003f0 <puts@plt>:
  4003f0:   jmp    QWORD PTR [rip+0x200c22]        # 601018 <puts@got>
  4003f6:   push   0x0
  4003fb:   jmp    4003e0 <.plt>


  ..

  4004f2:   call   4003f0 <puts@plt>
```

GOT

0x601010	0x0	0xdeadbeef
0x601020	printf@plt+6	read@plt+6
0x601030	write@plt+6	system@plt+6

GOT Hijacking

```
00000000004003e0 <.plt>:
  4003e0:   push   QWORD PTR [rip+0x200c22]        # 601008 <got+0x8>
  4003e6:   jmp    QWORD PTR [rip+0x200c24]        # 601010 <got+0x10>

00000000004003f0 <puts@plt>:
  4003f0:   jmp    QWORD PTR [rip+0x200c22]        # 601018 <puts@got>
  4003f6:   push   0x0
  4003fb:   jmp    4003e0 <.plt>


  ..

  4004f2:   call   4003f0 <puts@plt>
```



GOT Hijacking

```
00000000004003e0 <.plt>:
  4003e0:   push   QWORD PTR [rip+0x200c22]        # 601008 <got+0x8>
  4003e6:   jmp    QWORD PTR [rip+0x200c24]        # 601010 <got+0x10>

00000000004003f0 <puts@plt>:
  4003f0:   jmp    QWORD PTR [rip+0x200c22]        # 601018 <puts@got>
  4003f6:   push   0x0
  4003fb:   jmp    4003e0 <.plt>


  ..

  4004f2:   call   4003f0 <puts@plt>
```

Jump to 0xdeadbeef!
rip = 0xdeadbeef

GOT

0x601010	0x0	0xdeadbeef
0x601020	printf@plt+6	read@plt+6
0x601030	write@plt+6	system@plt+6

GOT Hijacking

```
00000000004003e0 <.plt>:
  4003e0:   push    QWORD PTR [rip+0x200c22]        # 601008 <got+0x8>
  4003e6:   jmp     QWORD PTR [rip+0x200c24]        # 601010 <got+0x10>

00000000004003f0 <puts@plt>:
  4003f0:   jmp     QWORD PTR [rip+0x200c22]        # 6
  4003f6:   push    0x0
  4003fb:   jmp     4003e0 <.plt>


  ..

  4004f2:                                 <puts@plt>
```

Jump to 0xdeadbeef!
rip = 0xdeadbeef

PWNED

GOT

0x601010	0x0	0xdeadbeef
0x601020	printf@plt+6	read@plt+6
0x601030	write@plt+6	system@plt+6

DEMO

Casino

HW

HW - Casino

- Just Pwn It!

Thanks!

 _yuawn

 yuawn