Linux Binary Exploitation

Return-oriented Programing

angelboy@chroot.org

Outline

- ROP
- Using ROP bypass ASLR
- Stack migration

Outline

- ROP
- Using ROP bypass ASLR
- Stack migration

ROP

- 透過 ret 去執行其他包含 ret 的程式碼片段
- 這些片段又稱為 gadget

```
4027ba:    5b                      pop    rbx
4027bb:    5d                      pop    rbp
4027bc:    41 5c                   pop    r12
4027be:    41 5d                   pop    r13
4027c0:    41 5e                   pop    r14
4027c2:    41 5f                   pop    r15
4027c4:    c3                      ret
```

```
40274c:    48 81 c7 00 06 00 00    add    rdi,0x600
402753:    48 89 f8                mov    rax,rdi
402756:    c3                      ret
```

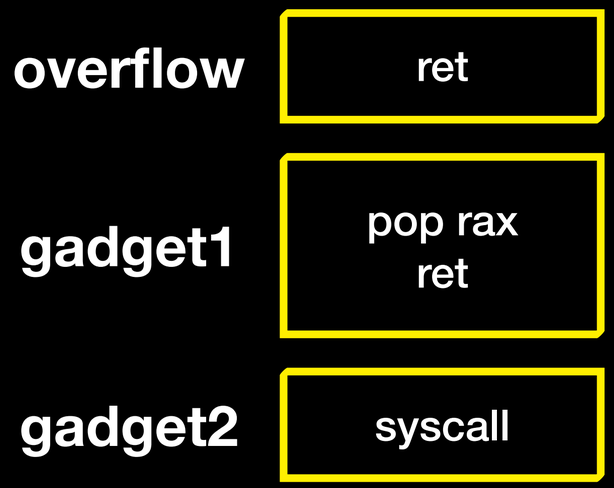
```
4026c5:    5a                      pop    rdx
4026c6:    58                      pop    rax
4026c7:    48 83 c4 08             add    rsp,0x8
4026cb:    5d                      pop    rbp
4026cc:    c3                      ret
```

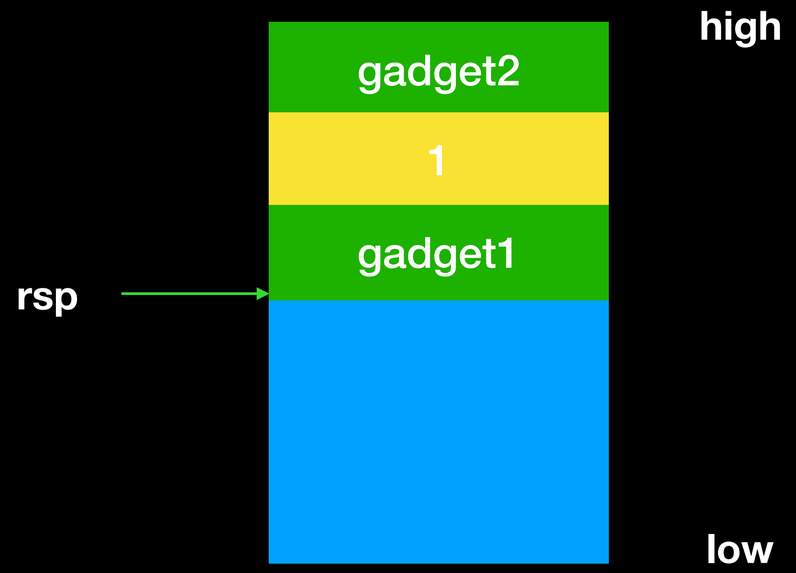
ROP

- Why do we need ROP ?
  - Bypass DEP
  - Static linking can do more thing

ROP

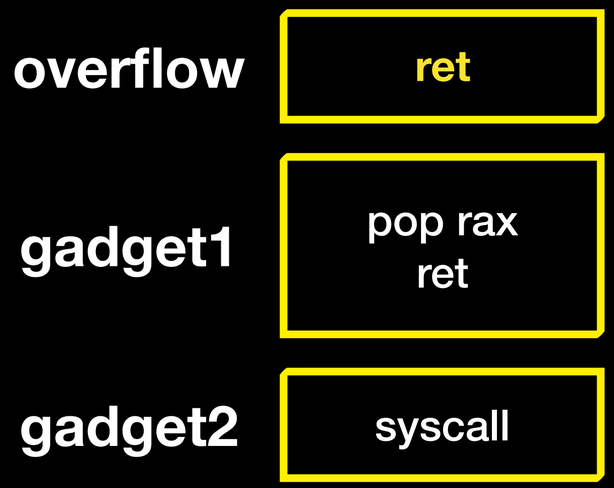
- ROP chain
  - 在夠長的 buffer overflow 後 stack 內容幾乎都由我們控制，我們可以藉由 ret = pop rip 指令，持續的控制 rip

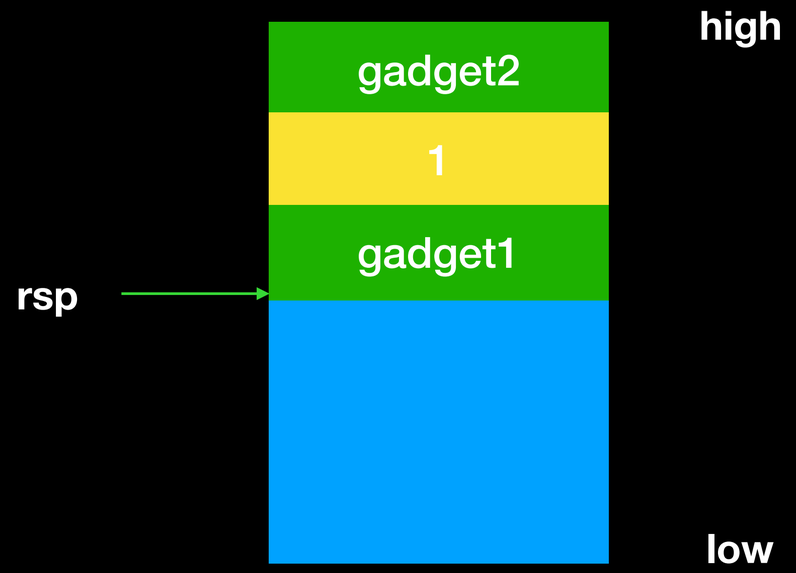




ROP

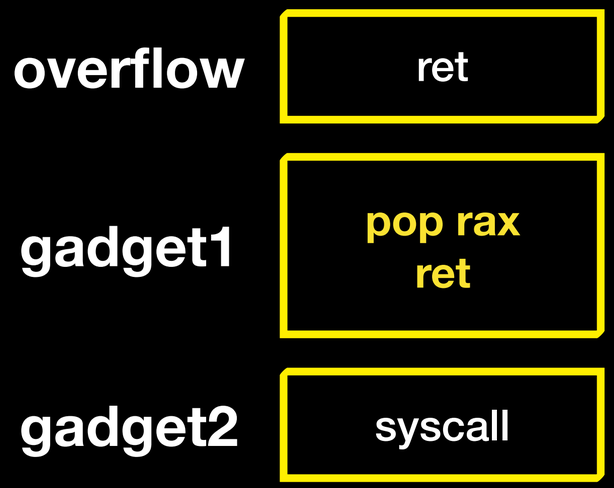
- ROP chain
  - 在夠長的 buffer overflow 後 stack 內容幾乎都由我們控制，我們可以藉由 ret = pop rip 指令，持續的控制 rip

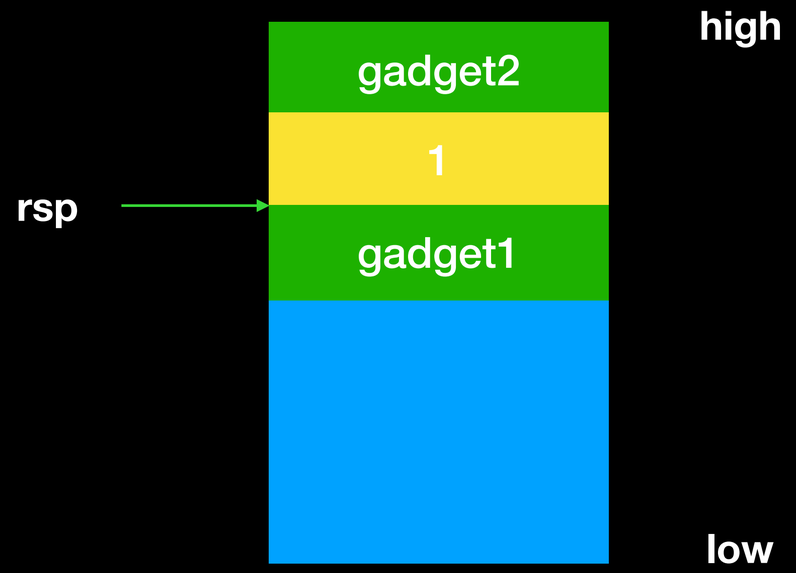




ROP

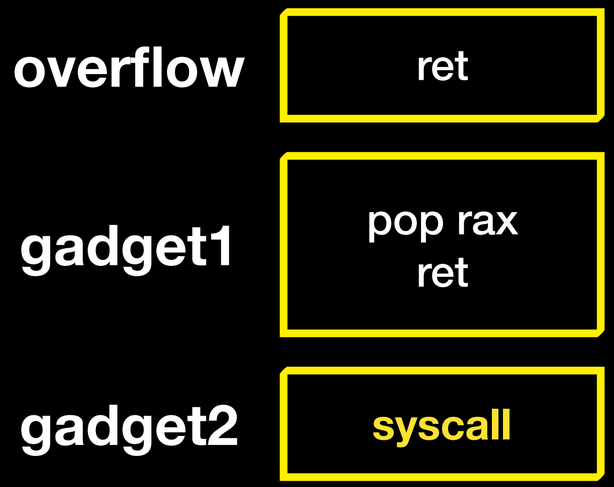
- ROP chain
  - 在夠長的 buffer overflow 後 stack 內容幾乎都由我們控制，我們可以藉由 ret = pop rip 指令，持續的控制 rip

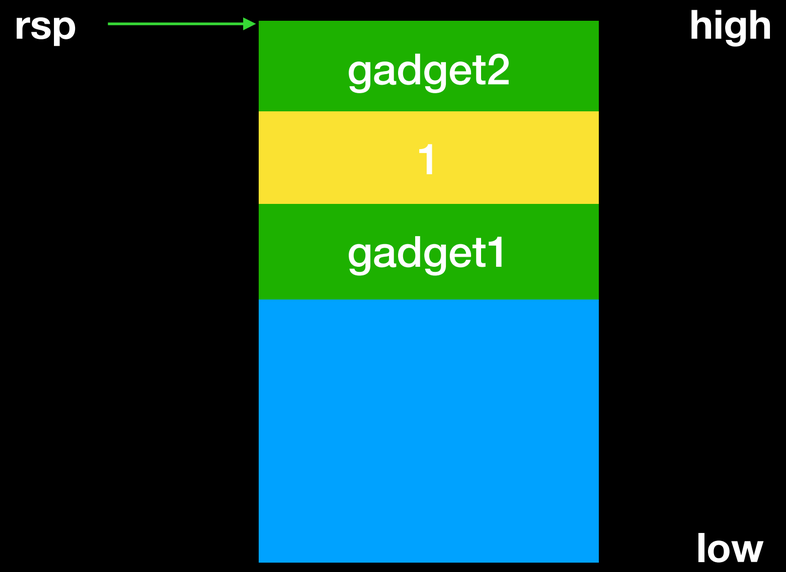




ROP

- ROP chain
  - 在夠長的 buffer overflow 後 stack 內容幾乎都由我們控制，我們可以藉由 ret = pop rip 指令，持續的控制 rip





ROP

- ROP chain
  - 在夠長的 buffer overflow 後 stack 內容幾乎都由我們控制，我們可以藉由 ret = pop rip 指令，持續的控制 rip

ROP

- ROP chain
  - 由眾多的 ROP gadget 組成
  - 藉由不同的 register 及記憶體操作，呼叫 system call 達成任意代碼執行
  - 基本上就是利用 ROP gadget 來串出我們之前寫的 shellcode 的效果

ROP

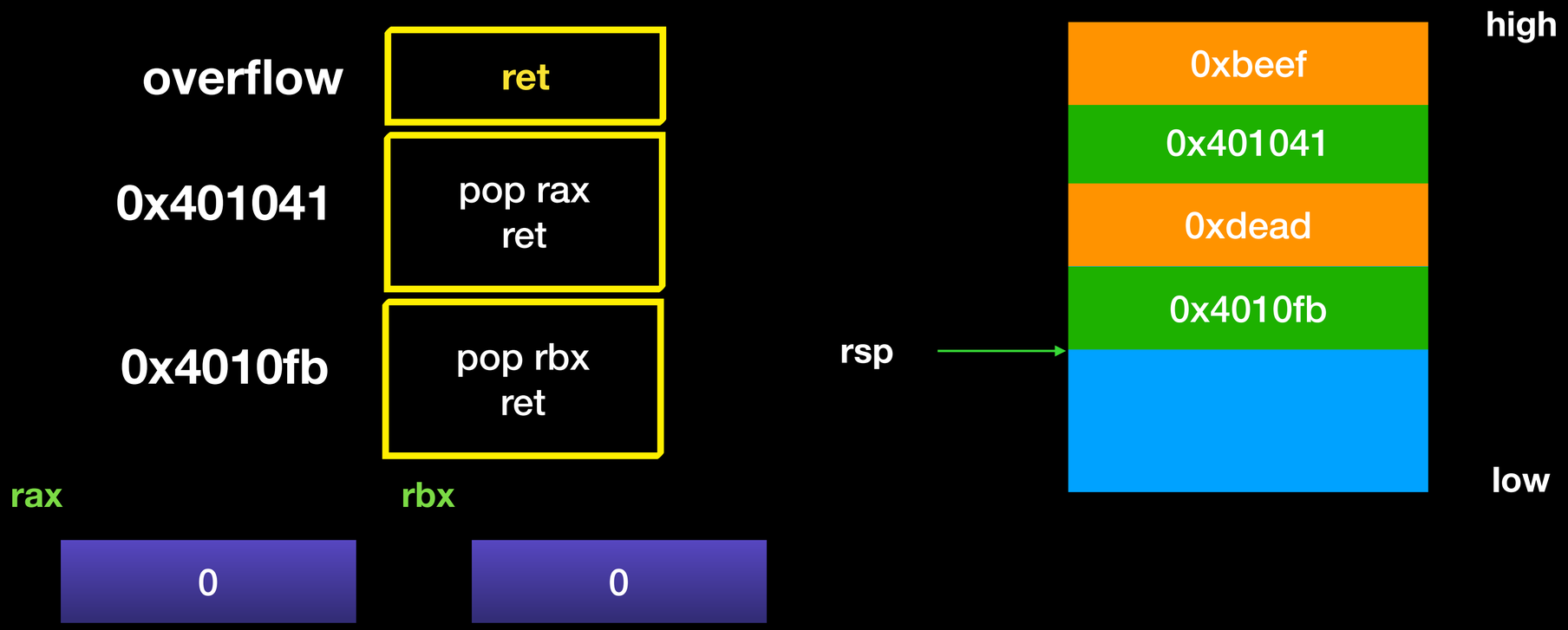
- Gadget
  - read/write register/memory
    - pop rax;pop rcx ; ret
    - mov [rax],rcx ; ret
  - system call
    - syscall
  - change rsp
    - pop rsp ; ret
    - leave ; ret

ROP

- Write to Register
  - pop reg ; ret
  - mov reg, reg ; ret
  - …

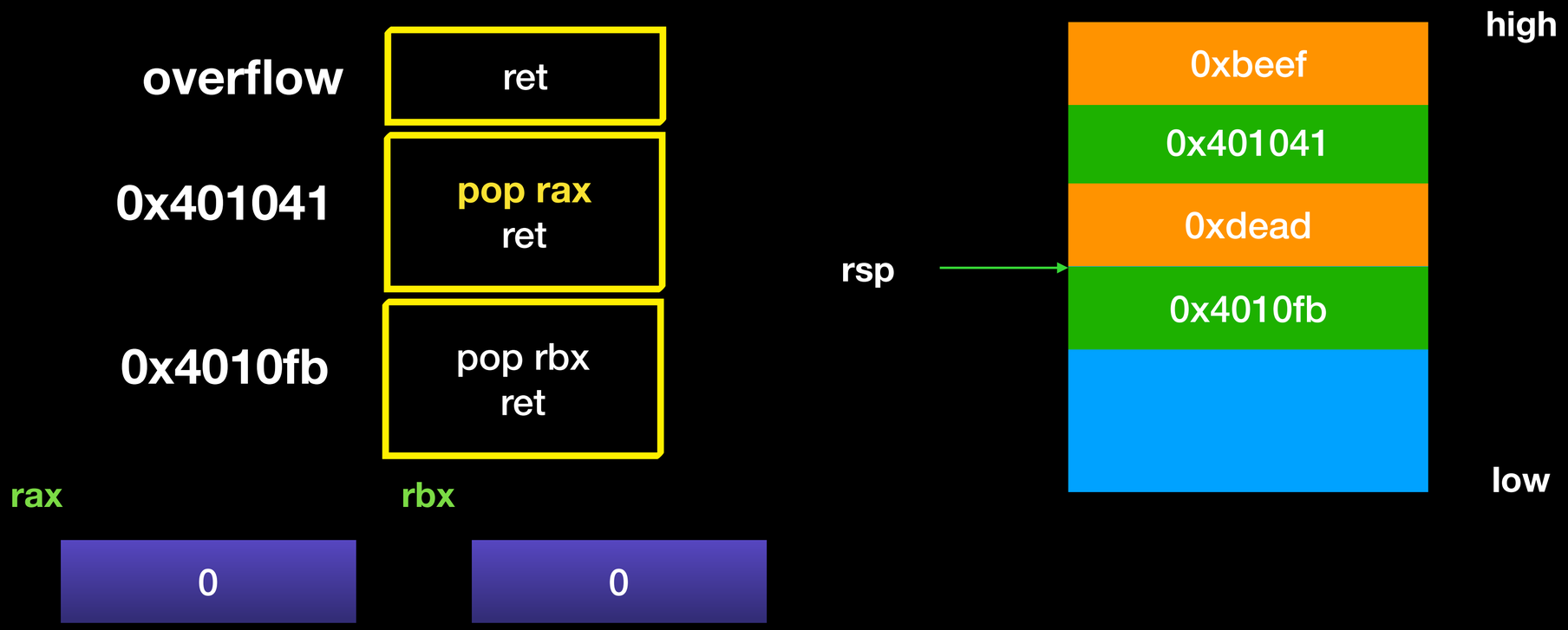
ROP

- Write to Register
  - let rax = 0xdead rbx = 0xbeef



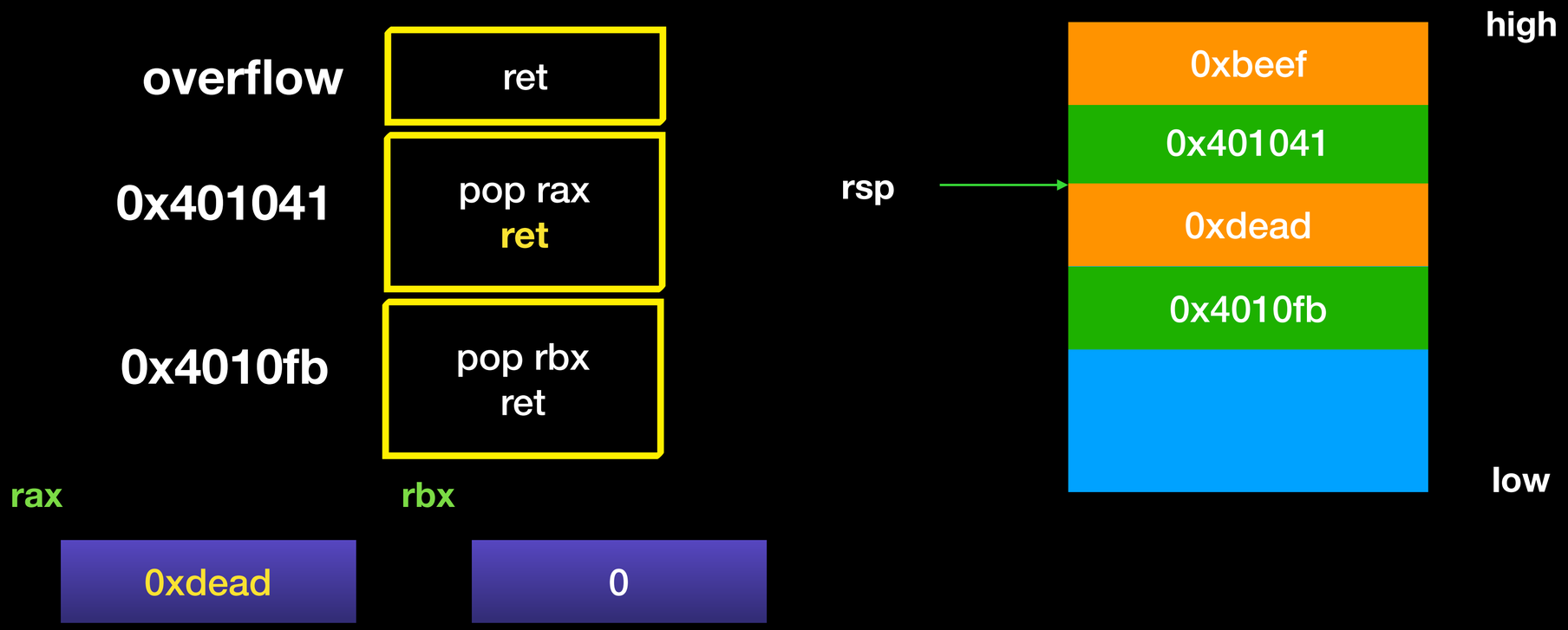
ROP

- Write to Register
  - let rax = 0xdead rbx = 0xbeef



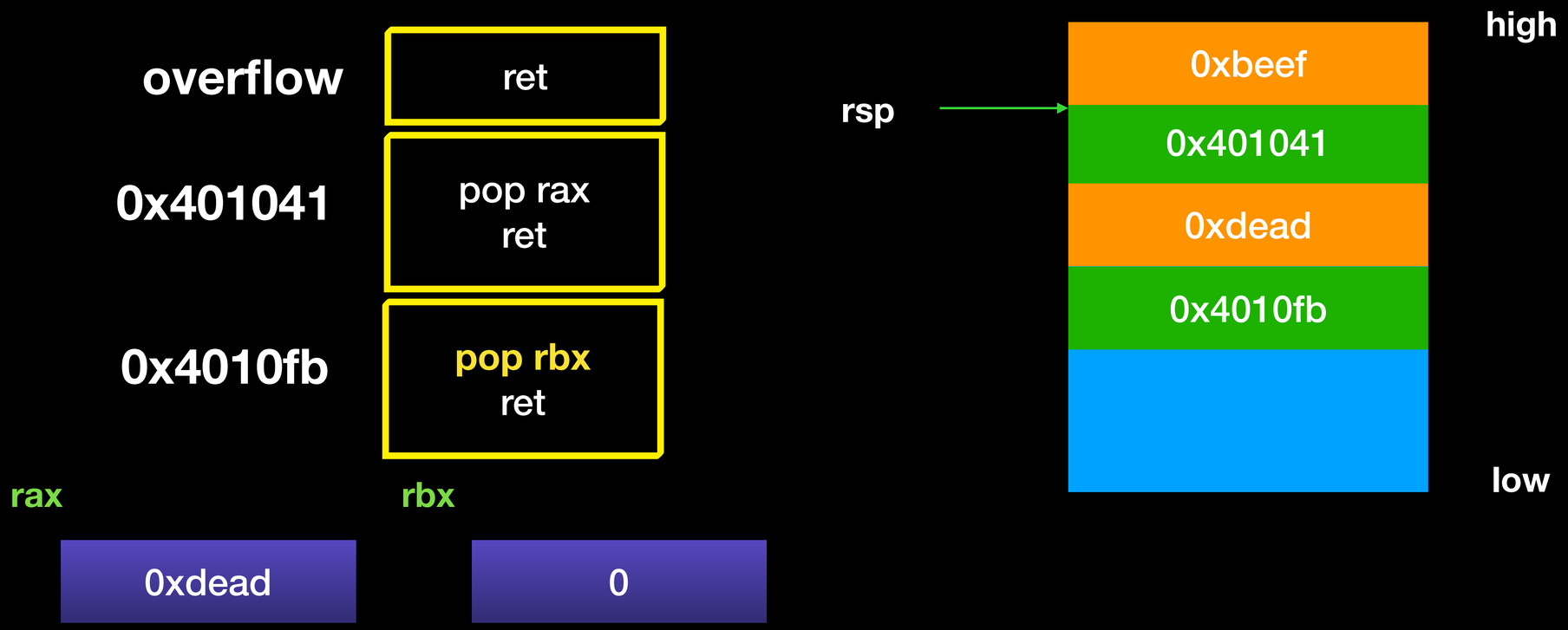
ROP

- Write to Register
  - let rax = 0xdead rbx = 0xbeef



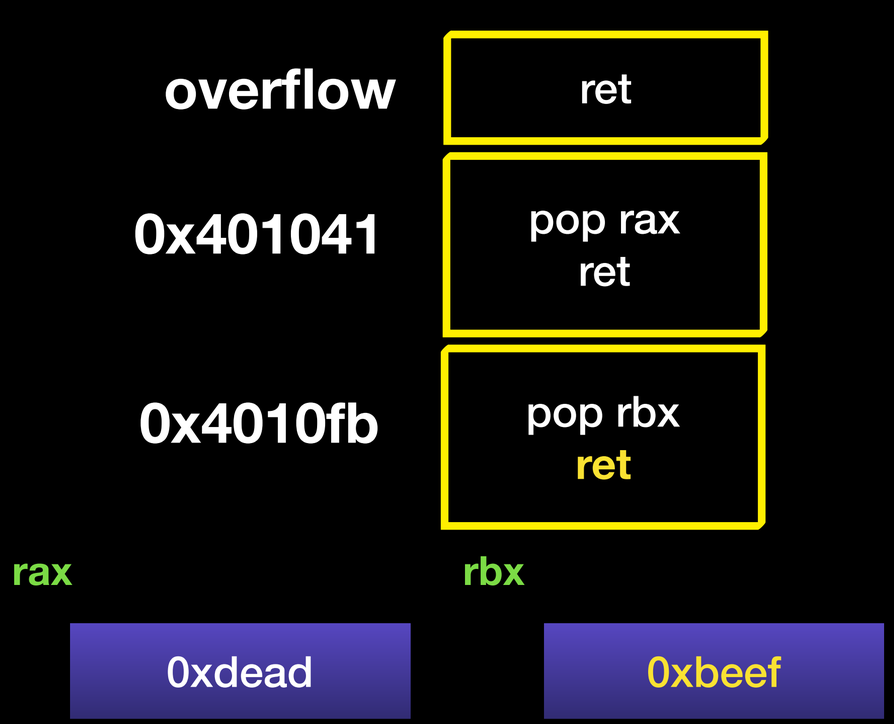
ROP

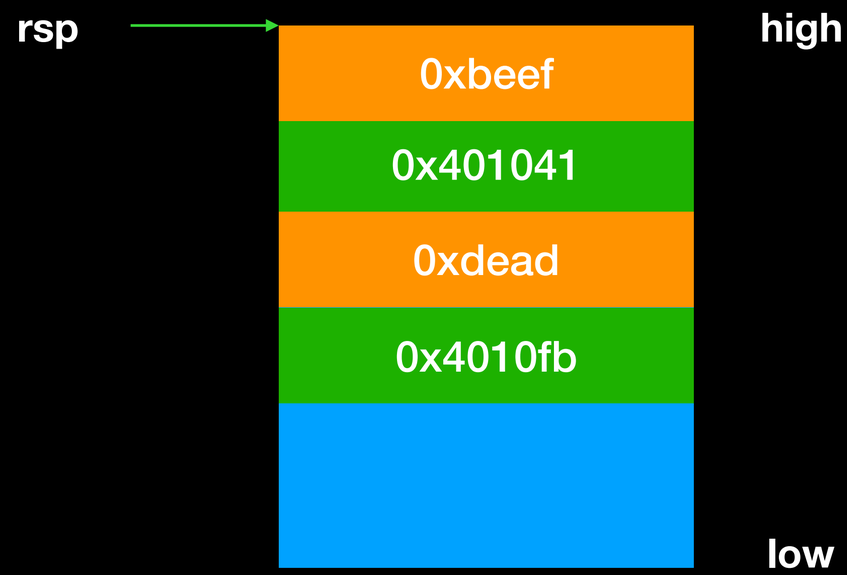
- Write to Register
  - let rax = 0xdead rbx = 0xbeef



ROP

- Write to Register
  - let rax = 0xdead rbx = 0xbeef



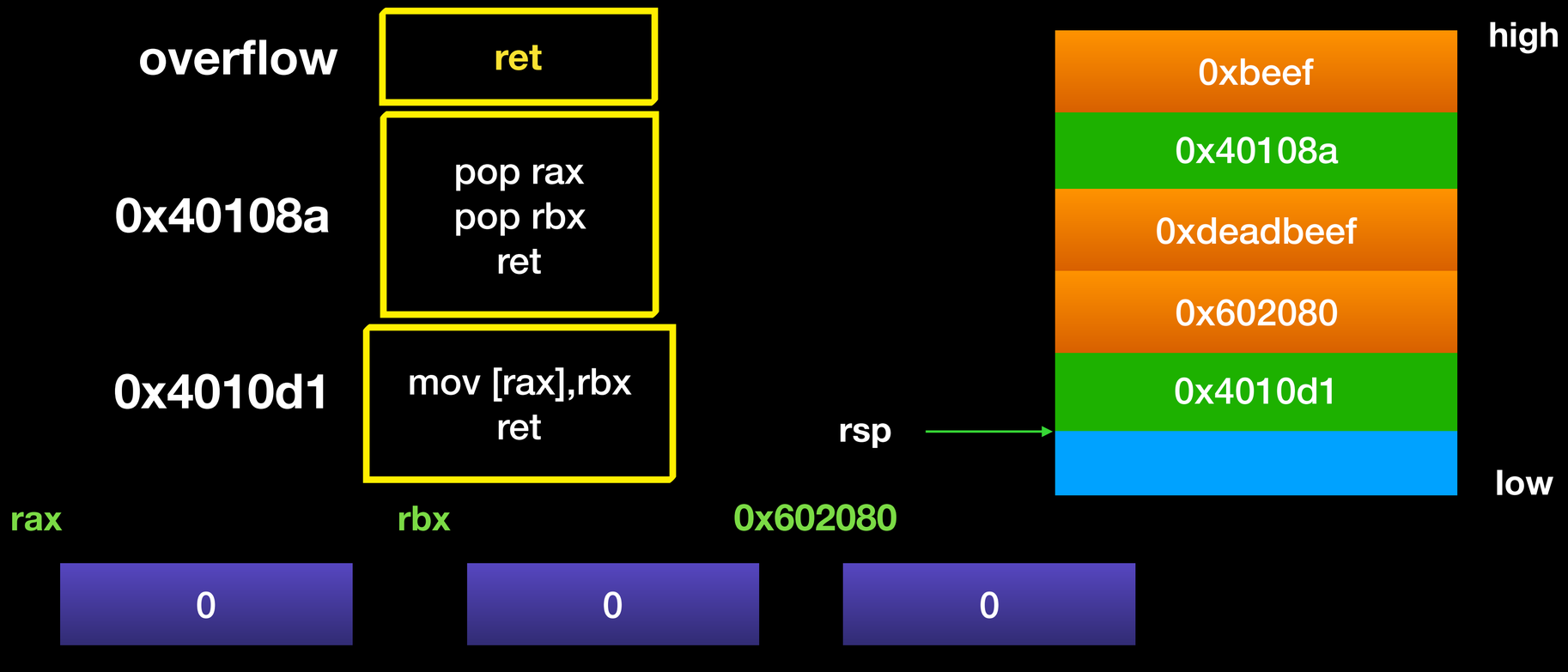


ROP

- Write to Memory
  - mov [reg],reg
  - mov [reg+xx], reg
  - …

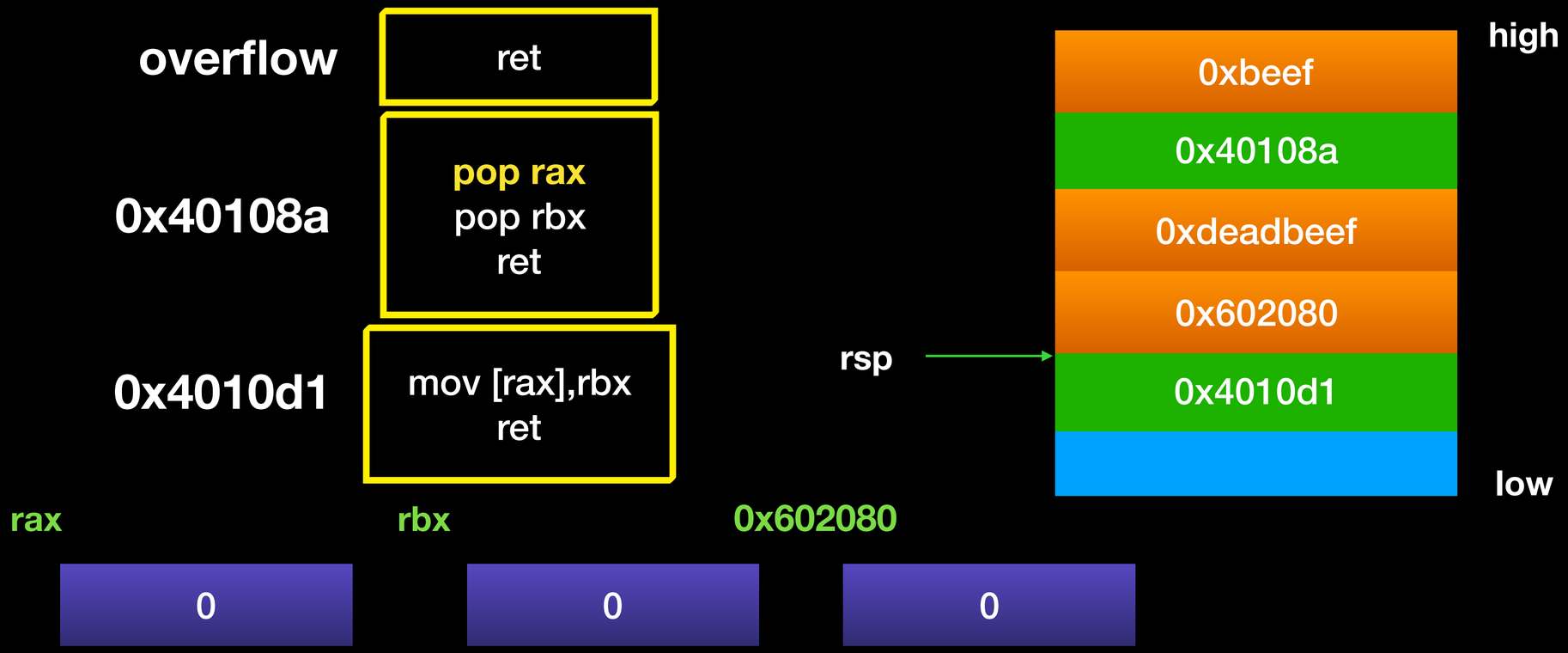
ROP

- Write to Memory
  - let *0x602080 = 0xdeadbeef



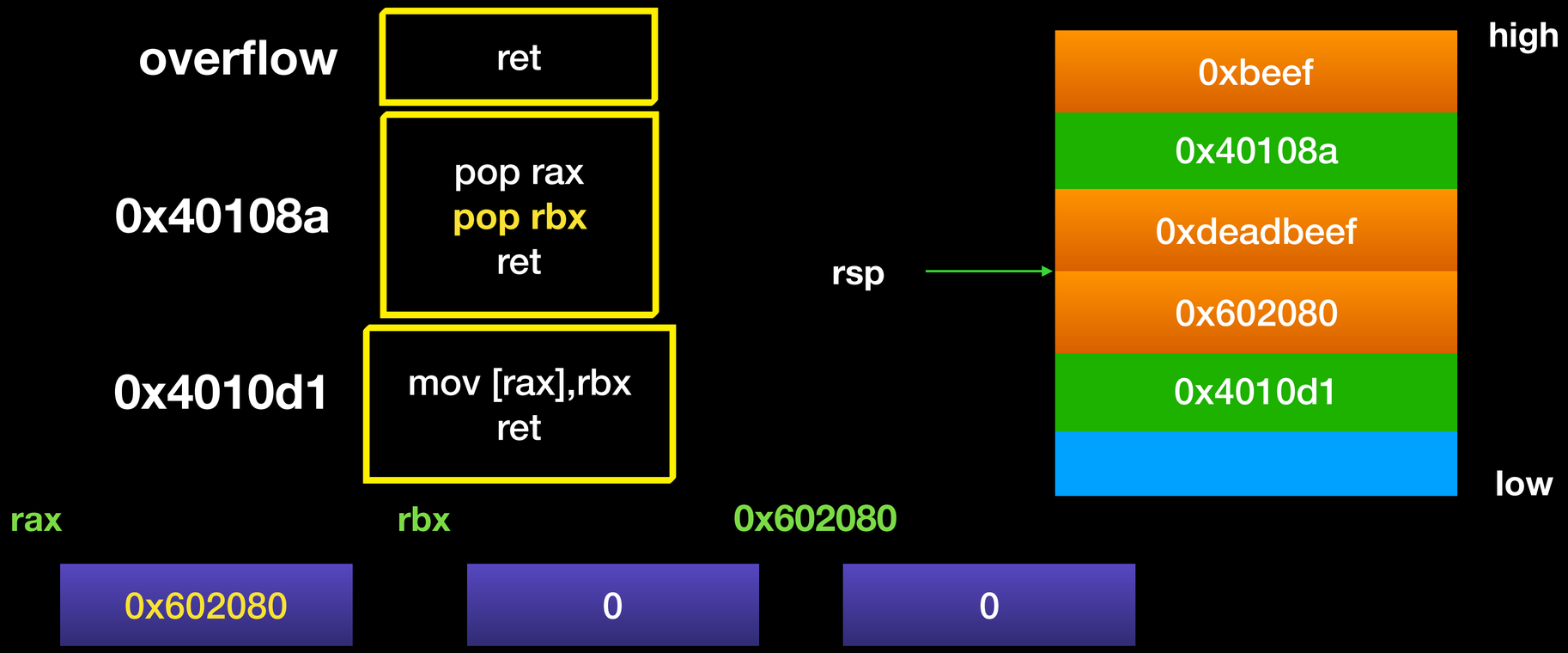
ROP

- Write to Memory
  - let *0x602080 = 0xdeadbeef



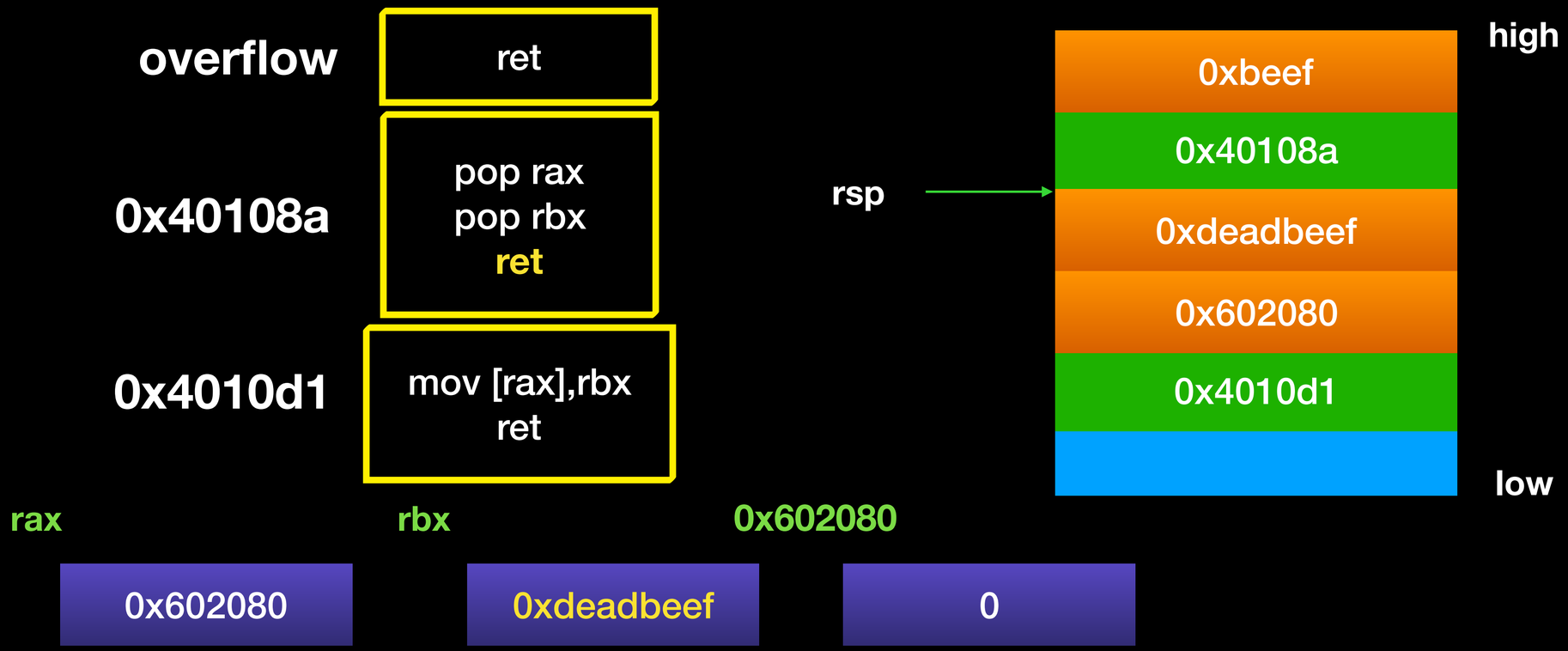
ROP

- Write to Memory
  - let *0x602080 = 0xdeadbeef



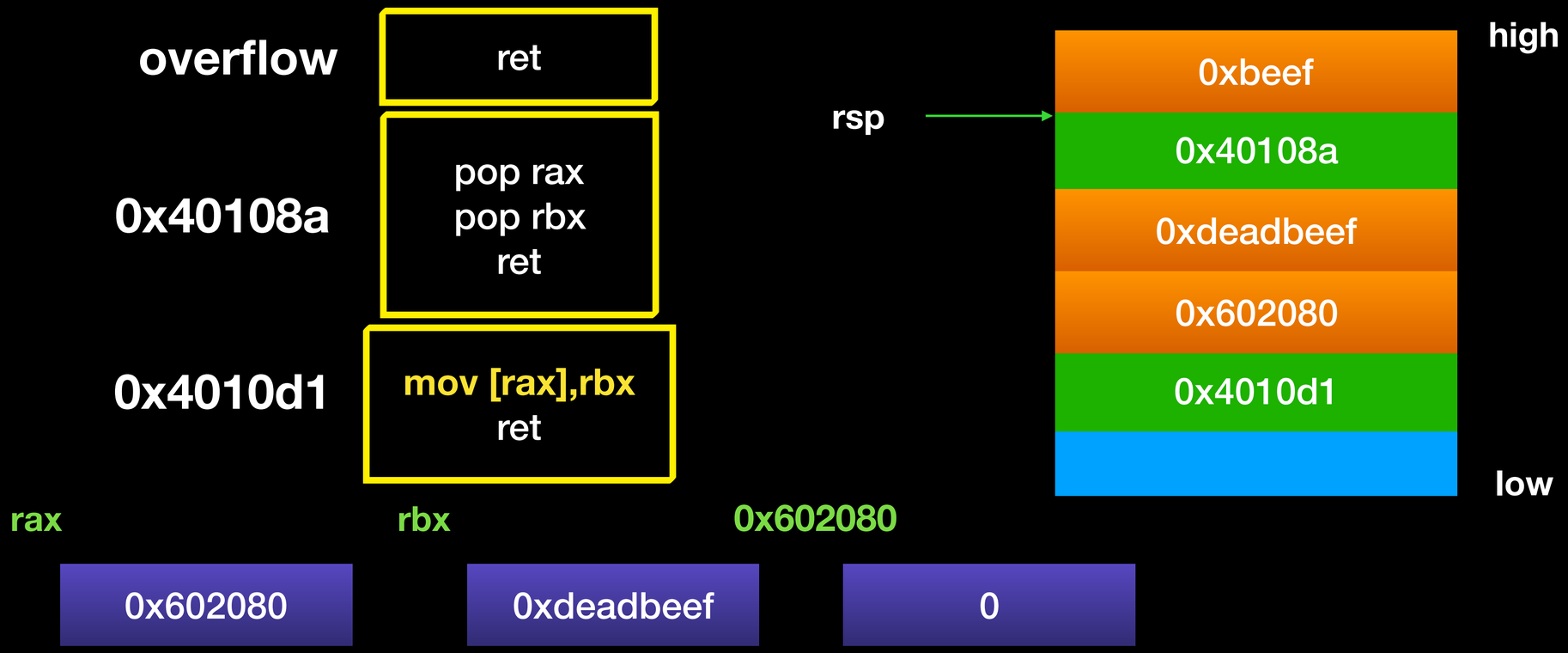
ROP

- Write to Memory
  - let *0x602080 = 0xdeadbeef



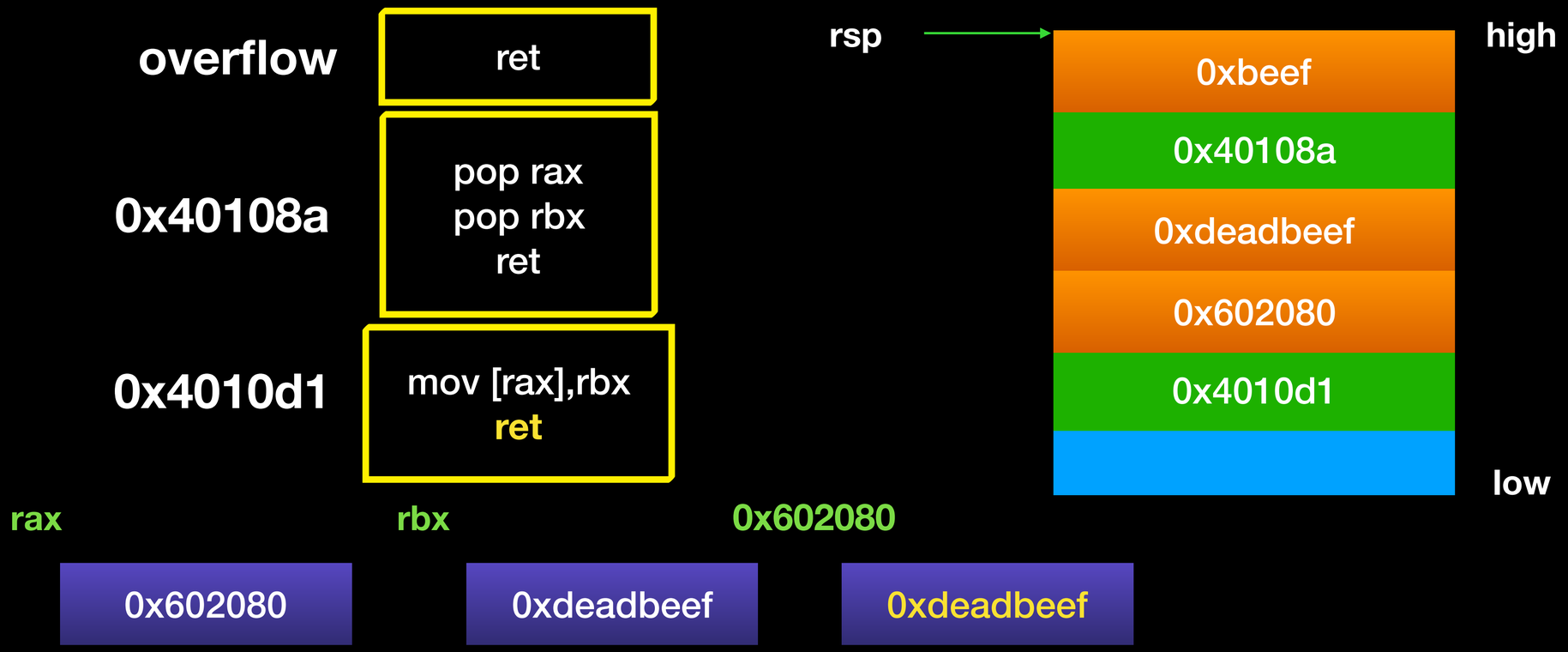
ROP

- Write to Memory
  - let *0x602080 = 0xdeadbeef



ROP

- Write to Memory
  - let *0x602080 = 0xdeadbeef



ROP

- execve(“/bin/sh”,NULL,NULL)
  - write to memory
    - 將 “/bin/sh” 寫入已知位置記憶體中
    - 可分多次將所需字串寫入記憶體中

ROP

- execve(“/bin/sh”,NULL,NULL)
  - write to register
    - rax = 0x3b , rdi = address of “/bin/sh”
    - rsi = 0 , rdx = 0
  - syscall

ROP

- find gadget
  - https://github.com/JonathanSalwan/ROPgadget

```
0x0000000000401947 : pop r12 ; pop r13 ; pop r14 ; pop r15 ; ret
0x00000000004016da : pop r12 ; pop r13 ; pop r14 ; ret
0x0000000000401ee0 : pop r12 ; pop r13 ; ret
0x0000000000401949 : pop r13 ; pop r14 ; pop r15 ; ret
0x00000000004016dc : pop r13 ; pop r14 ; ret
0x0000000000401ee2 : pop r13 ; ret
0x000000000040194b : pop r14 ; pop r15 ; ret
0x00000000004016de : pop r14 ; ret
0x000000000040194d : pop r15 ; ret
0x00000000004026c6 : pop rax ; add rsp, 8 ; pop rbp ; ret
0x000000000040260d : pop rax ; ret
0x0000000000400f92 : pop rbp ; mov byte ptr [rip + 0x20309e], 1 ; ret
0x0000000000400f1f : pop rbp ; mov edi, 0x604018 ; jmp rax
0x0000000000401946 : pop rbp ; pop r12 ; pop r13 ; pop r14 ; pop r15 ; ret
0x00000000004016d9 : pop rbp ; pop r12 ; pop r13 ; pop r14 ; ret
0x0000000000401edf : pop rbp ; pop r12 ; pop r13 ; ret
0x000000000040194a : pop rbp ; pop r14 ; pop r15 ; ret
0x00000000004016dd : pop rbp ; pop r14 ; ret
0x0000000000400f30 : pop rbp ; ret
0x0000000000401ede : pop rbx ; pop rbp ; pop r12 ; pop r13 ; ret
0x0000000000401540 : pop rbx ; pop rbp ; ret
0x000000000040228e : pop rbx ; ret
0x000000000040194e : pop rdi ; ret
```

ROP

- find gadget
  - ROPgadget - - binary binary
  - ROPgadget - - ropchain - - binary binary
    - 在 Static linking 通常可以組成功 execve 的 rop chain 但通常都很長，需要自己找更短的 gadget 來改短一點

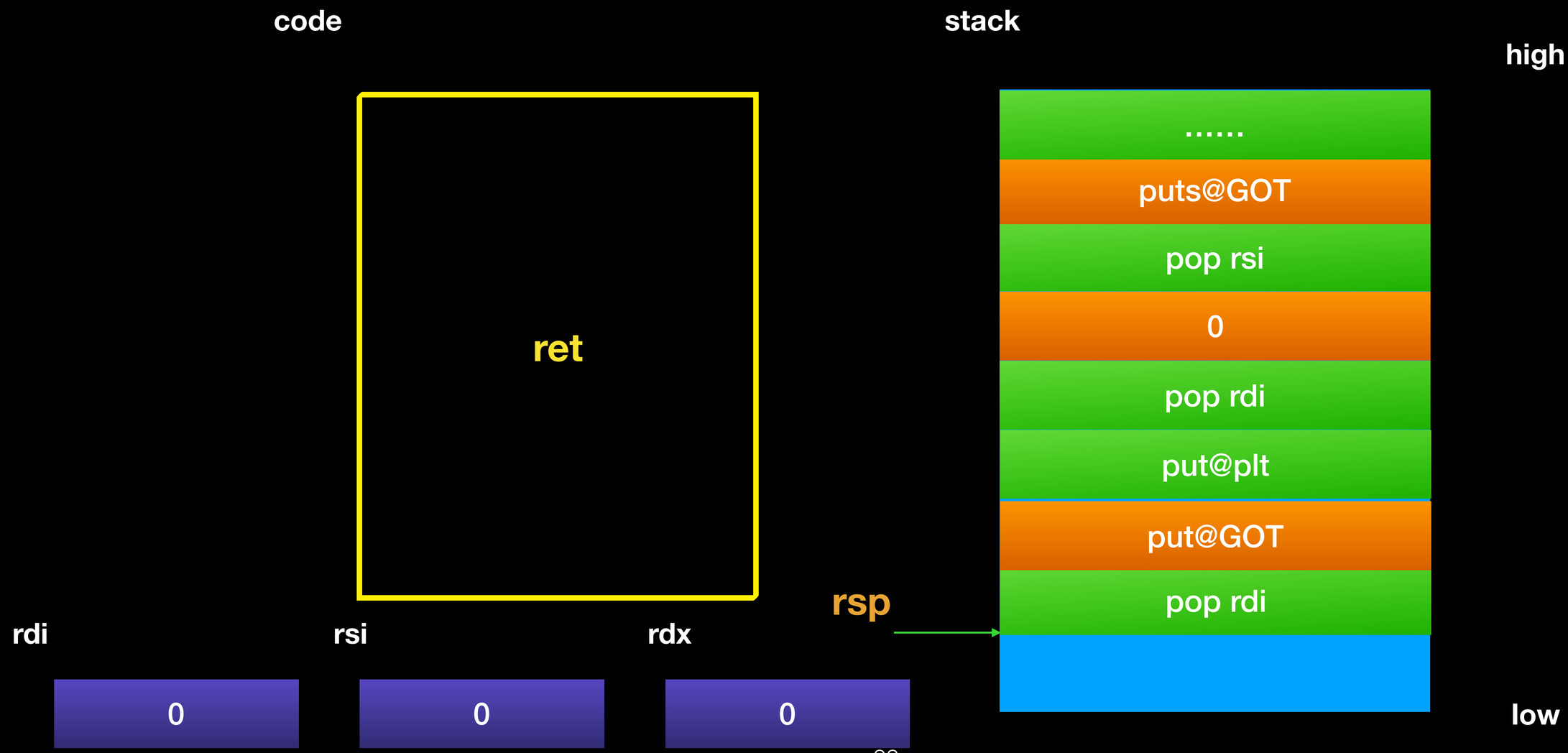
Outline

- ROP
- Using ROP bypass ASLR
- Stack migration

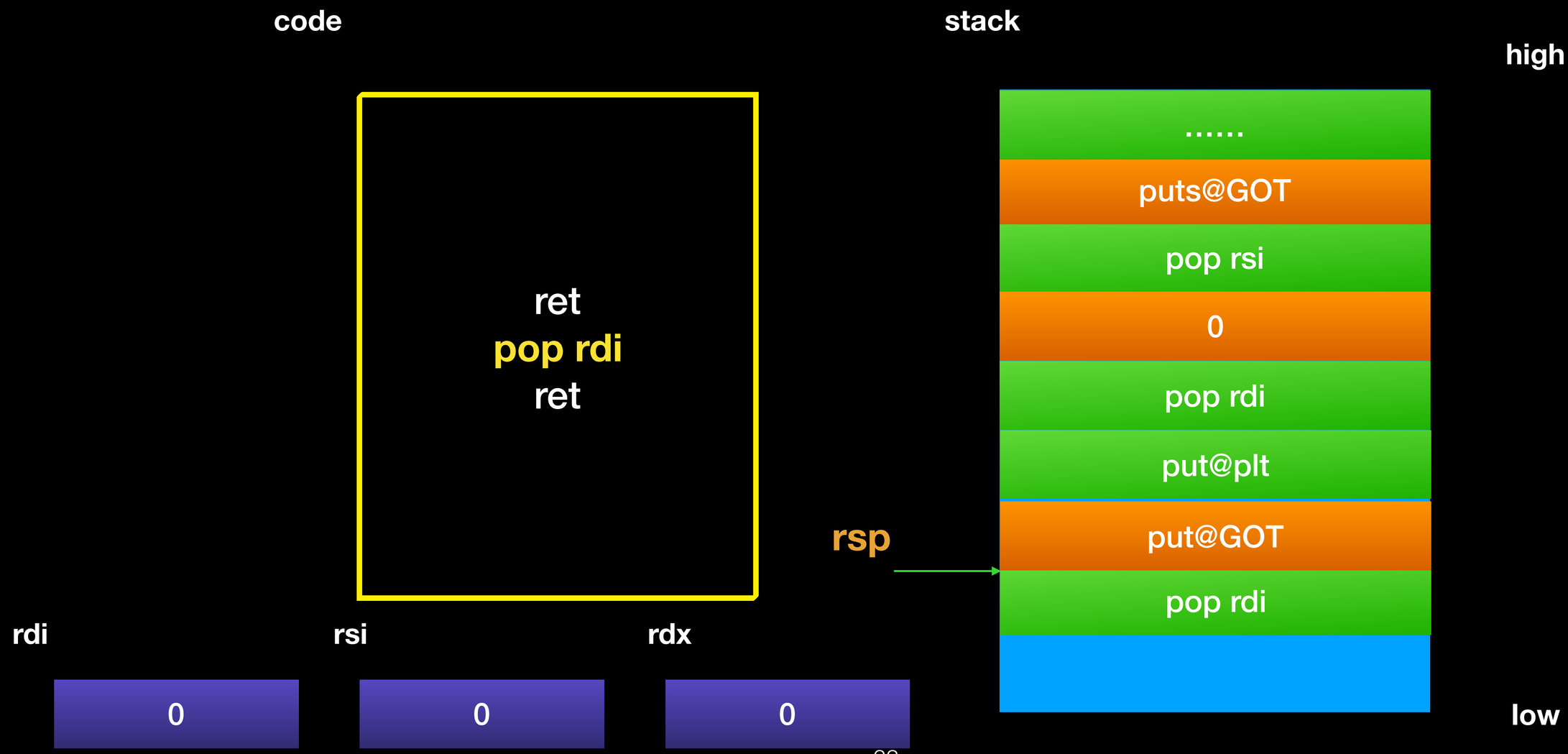
Using ROP bypass ASLR

- 假設 dynamic 編譯的程式中有 Buffer Overflow 的漏洞且在沒 PIE 情況下 (先不考慮 StackGuard 的情況)
- How to bypass ASLR and DEP ?
  - Use .plt section to leak some information
    - ret2plt
  - 通常一般的程式中都會有 put 、 send 、write 等 output function

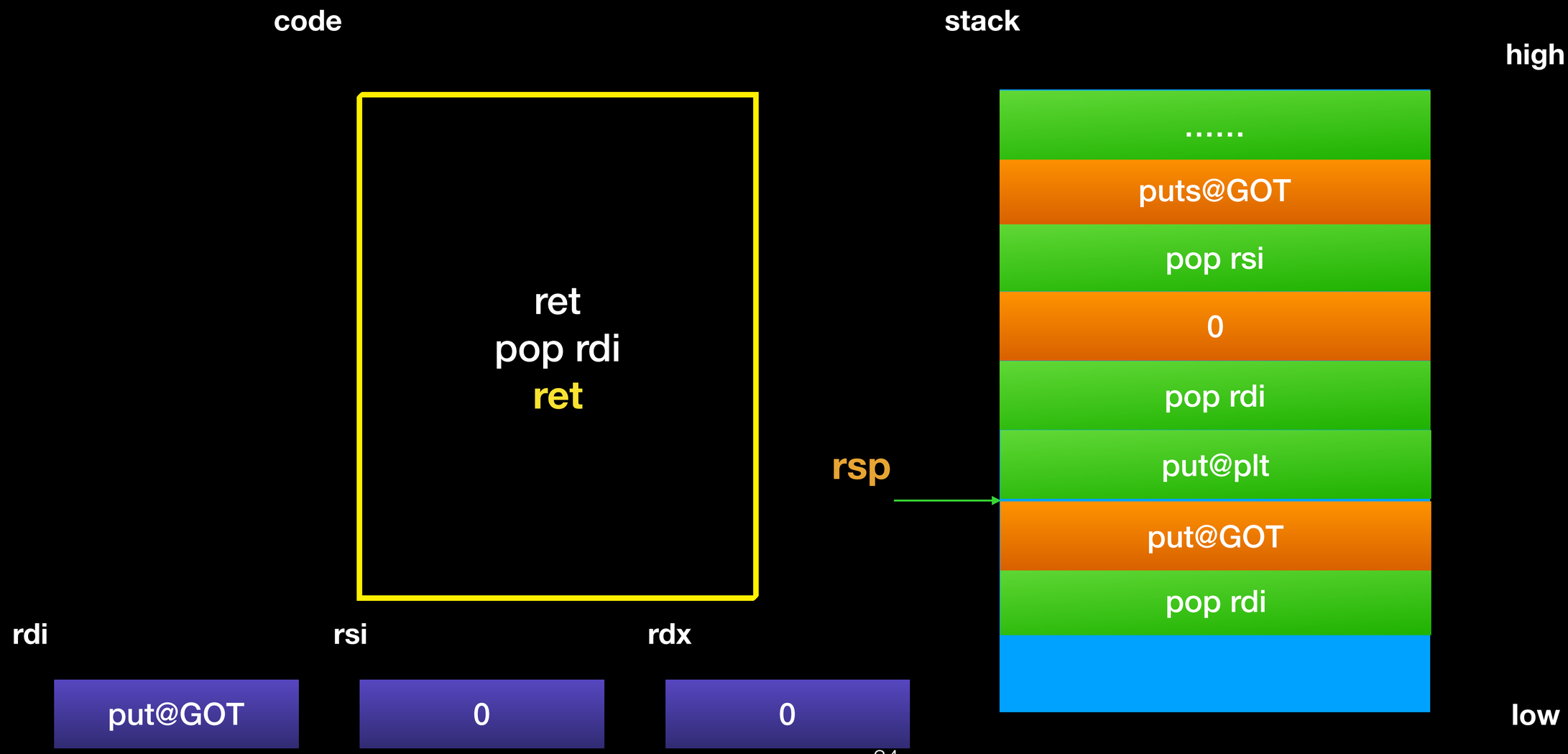
Using ROP bypass ASLR



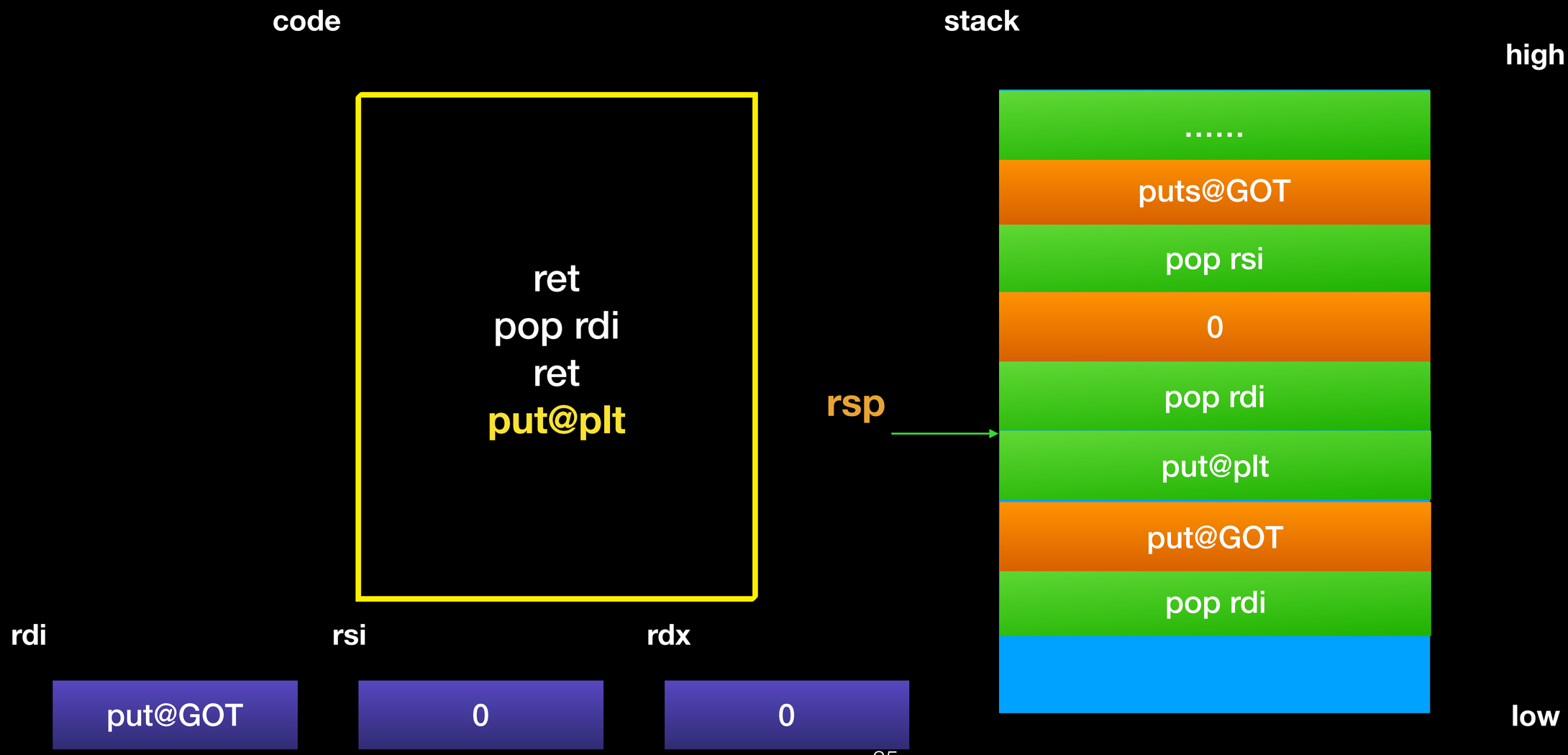
Using ROP bypass ASLR



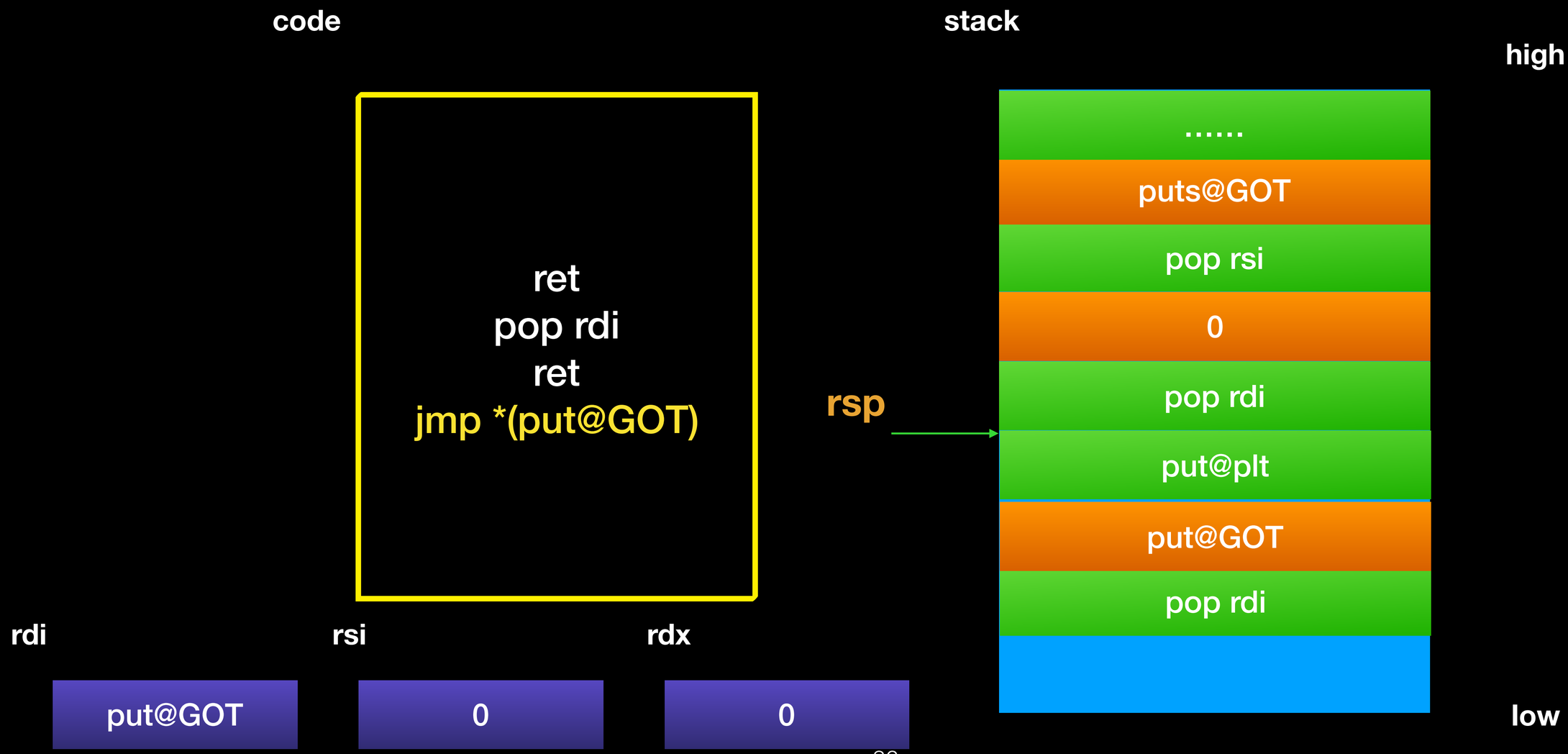
Using ROP bypass ASLR



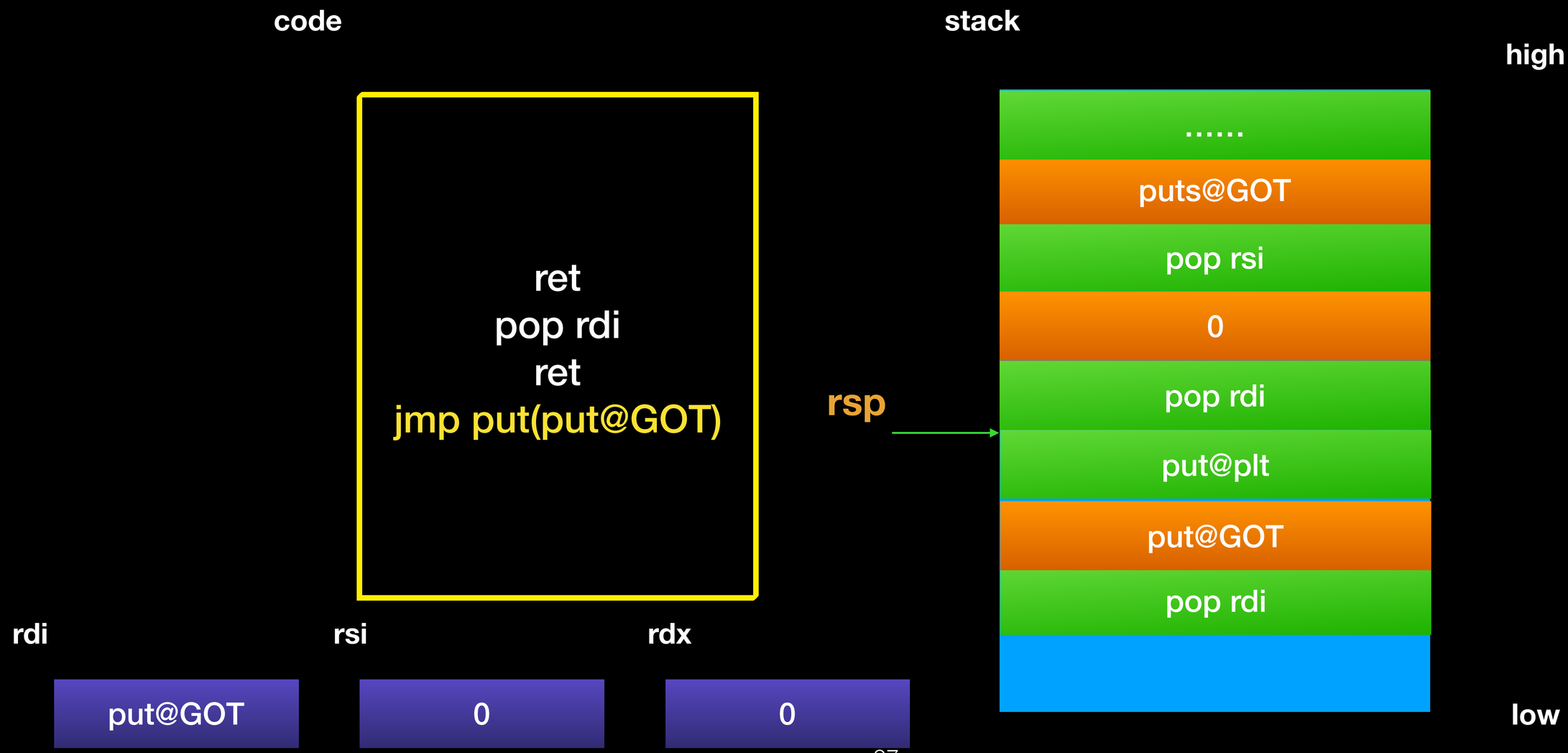
Using ROP bypass ASLR



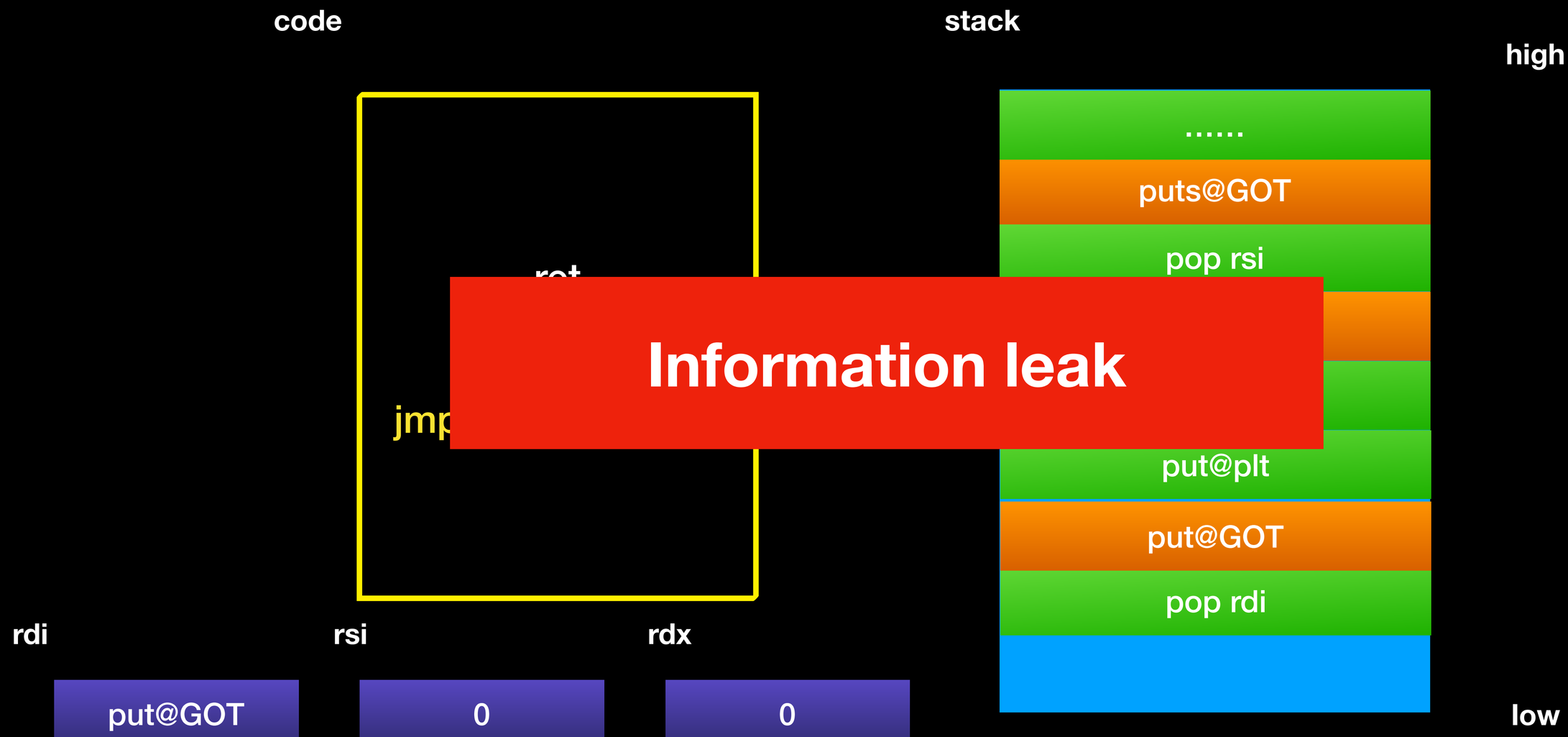
Using ROP bypass ASLR



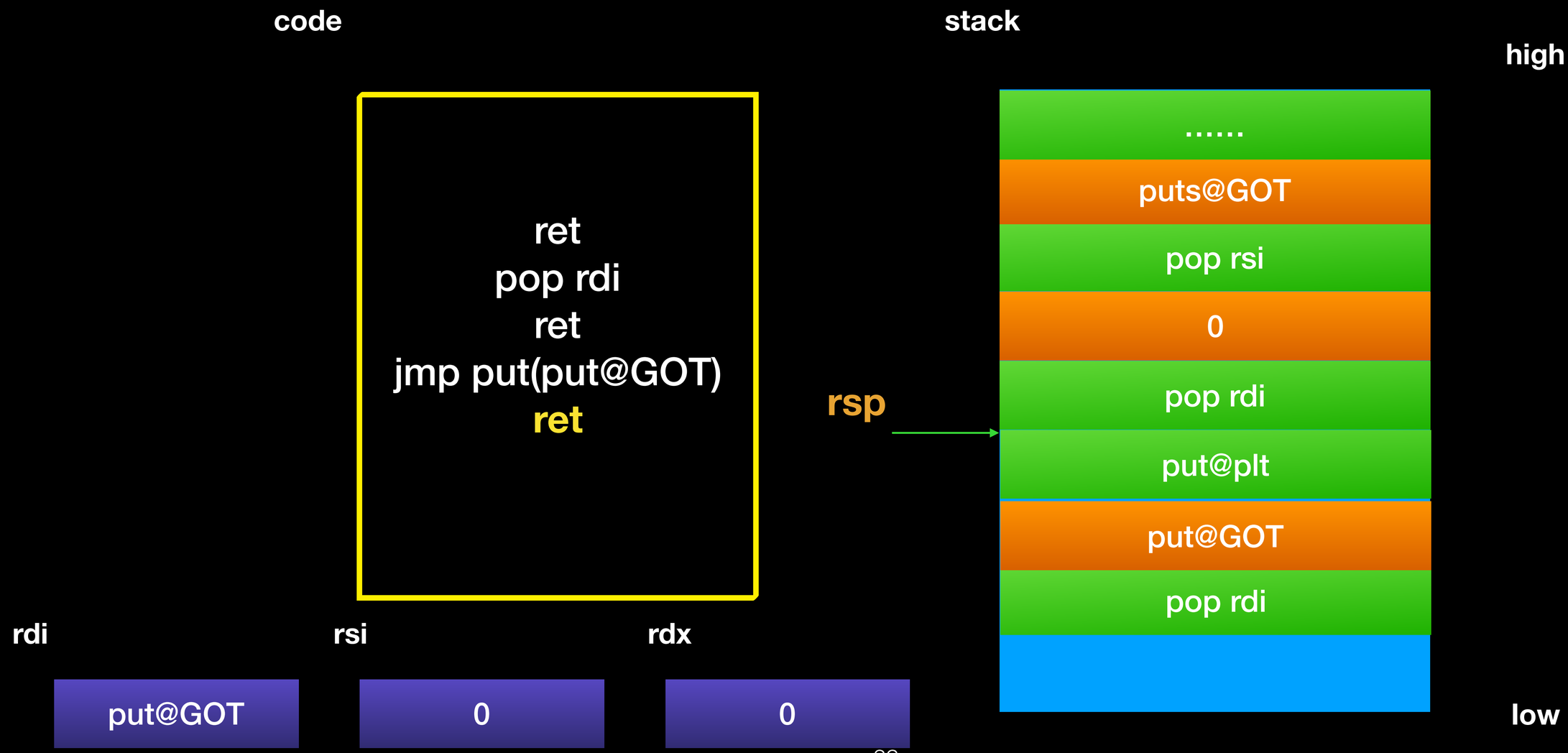
Using ROP bypass ASLR



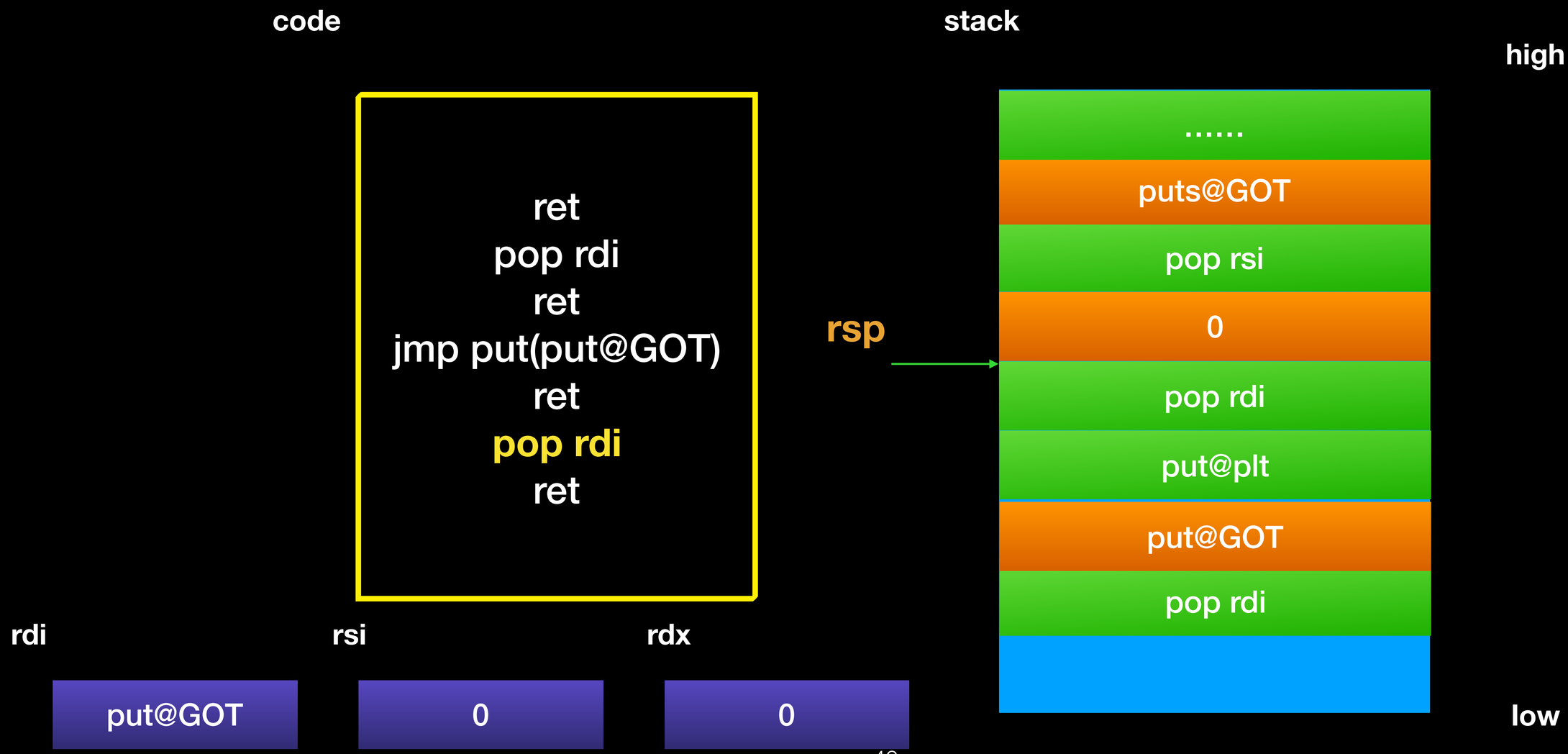
Using ROP bypass ASLR



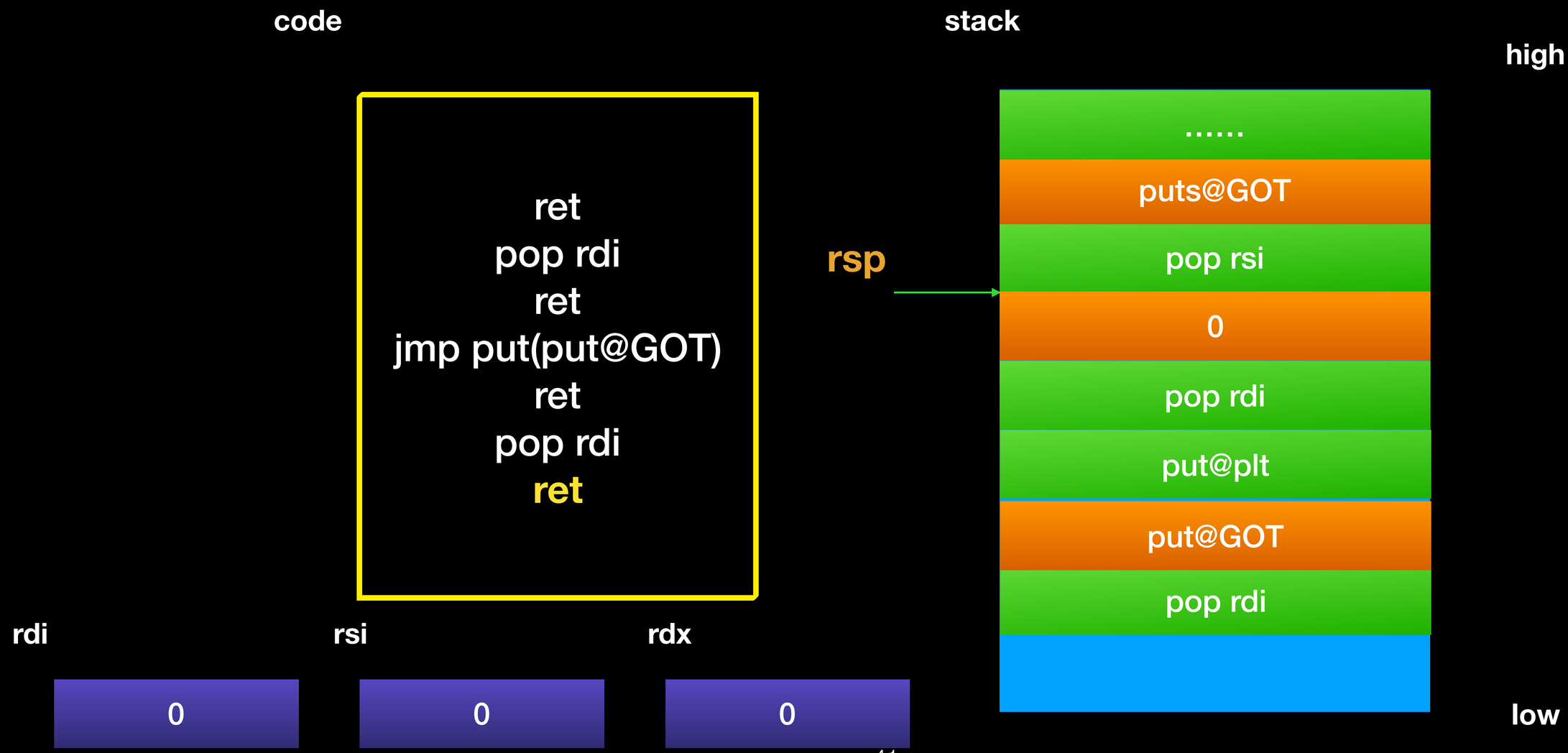
Using ROP bypass ASLR



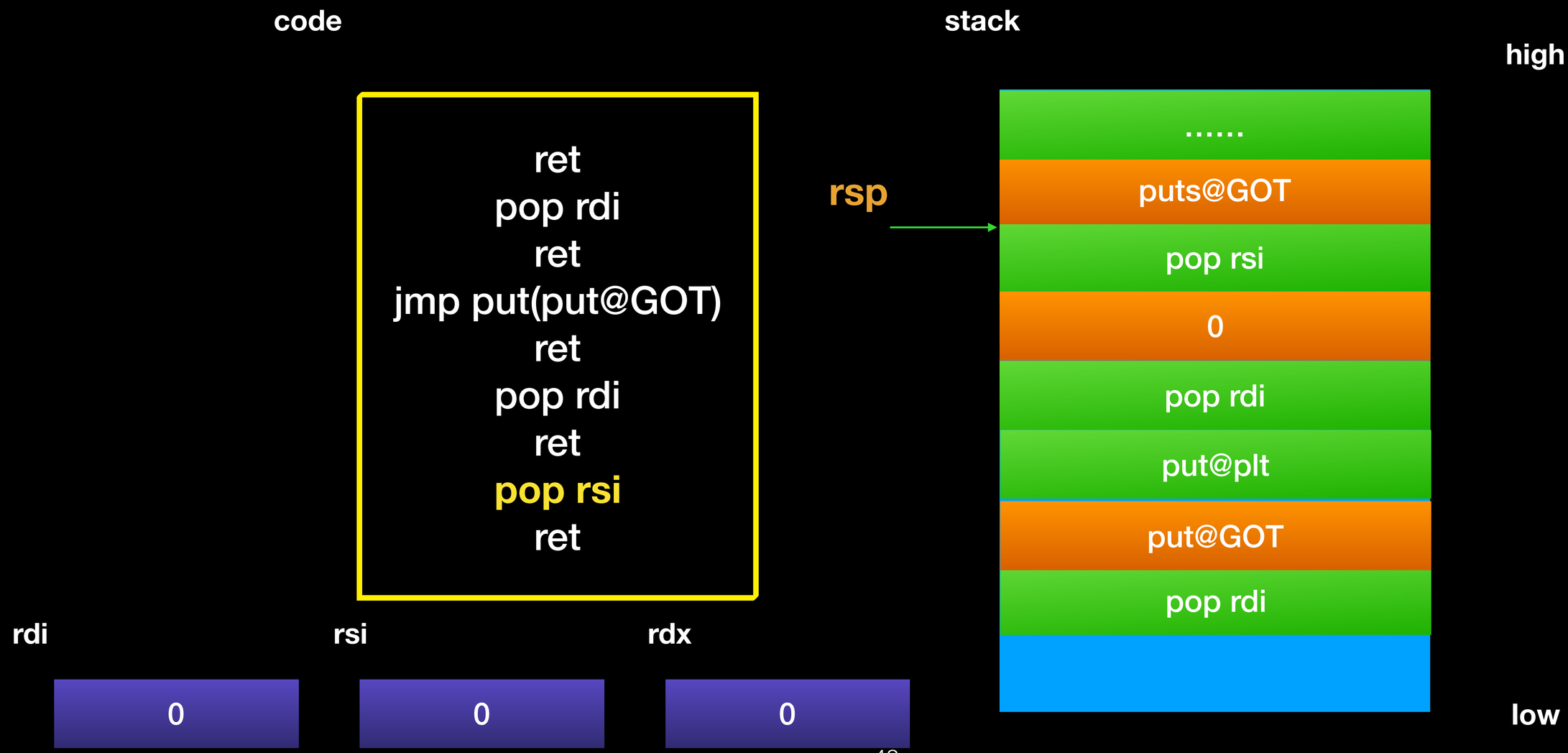
Using ROP bypass ASLR



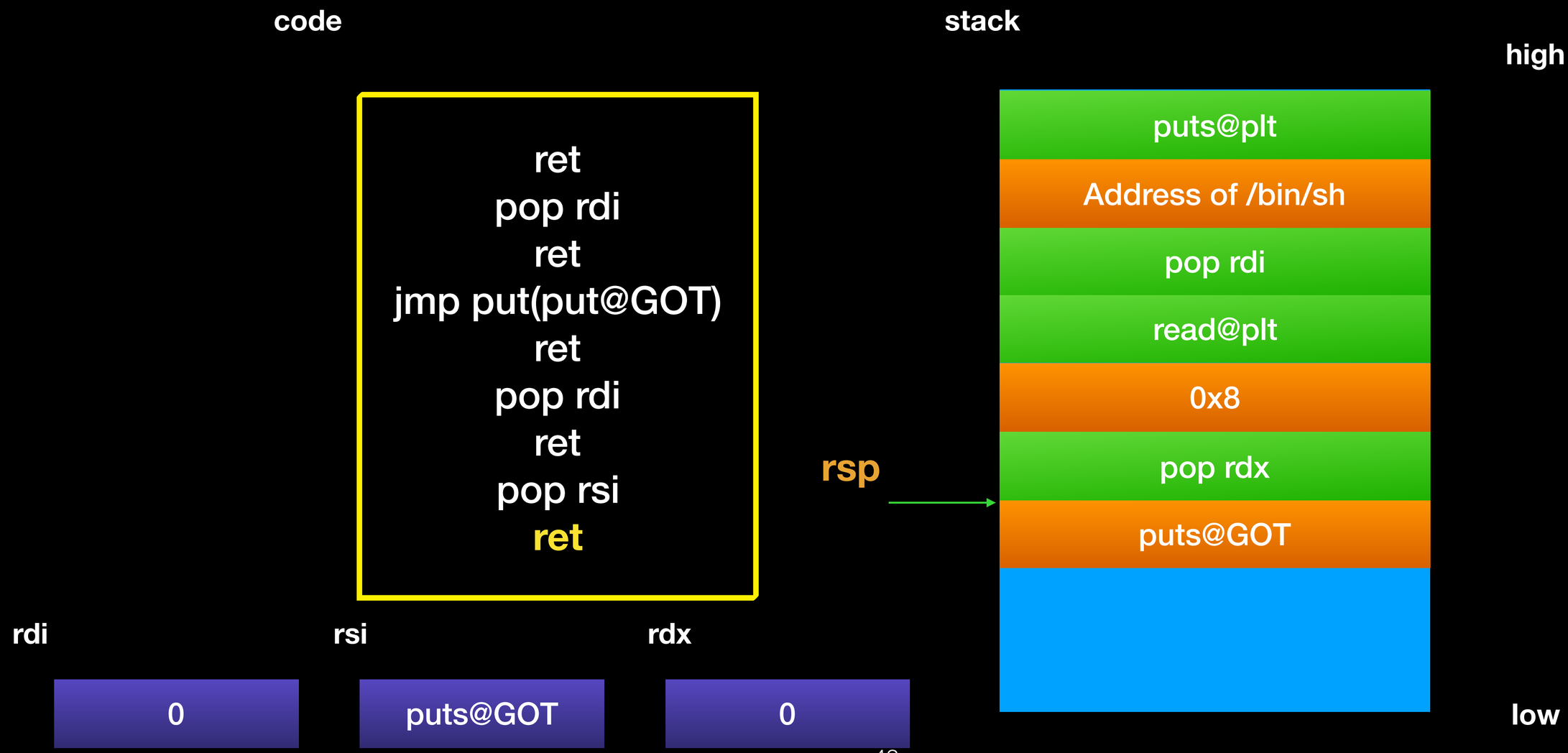
Using ROP bypass ASLR



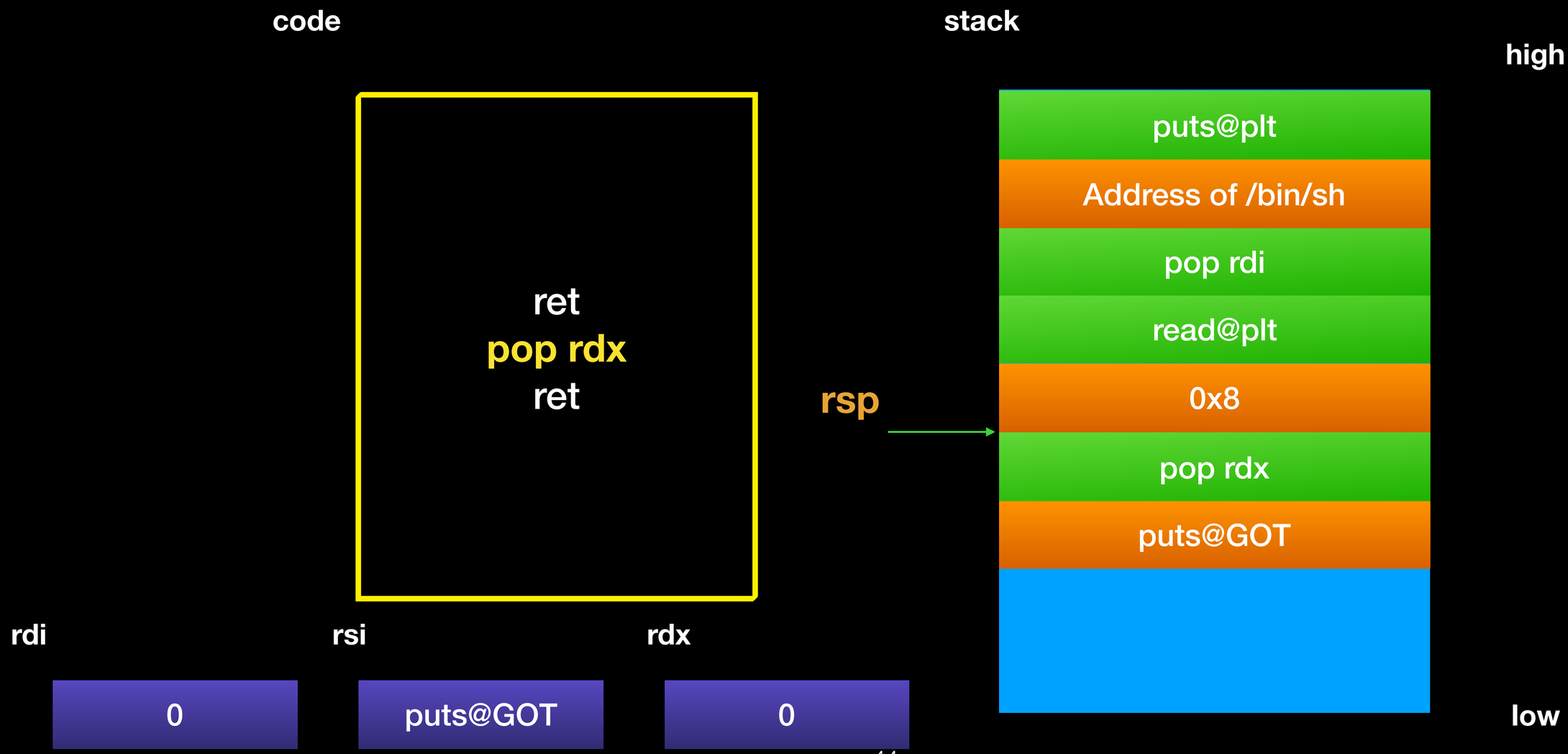
Using ROP bypass ASLR



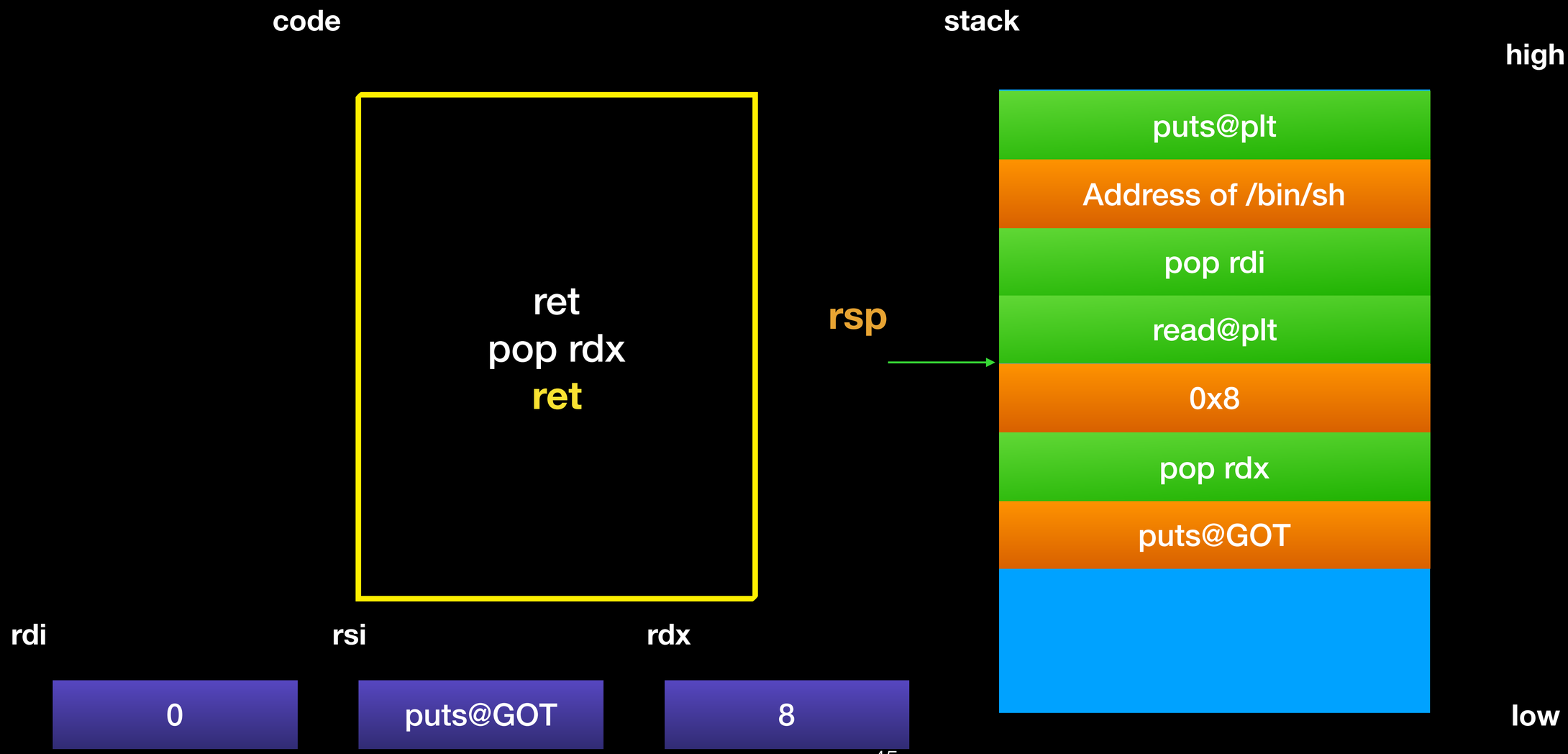
Using ROP bypass ASLR



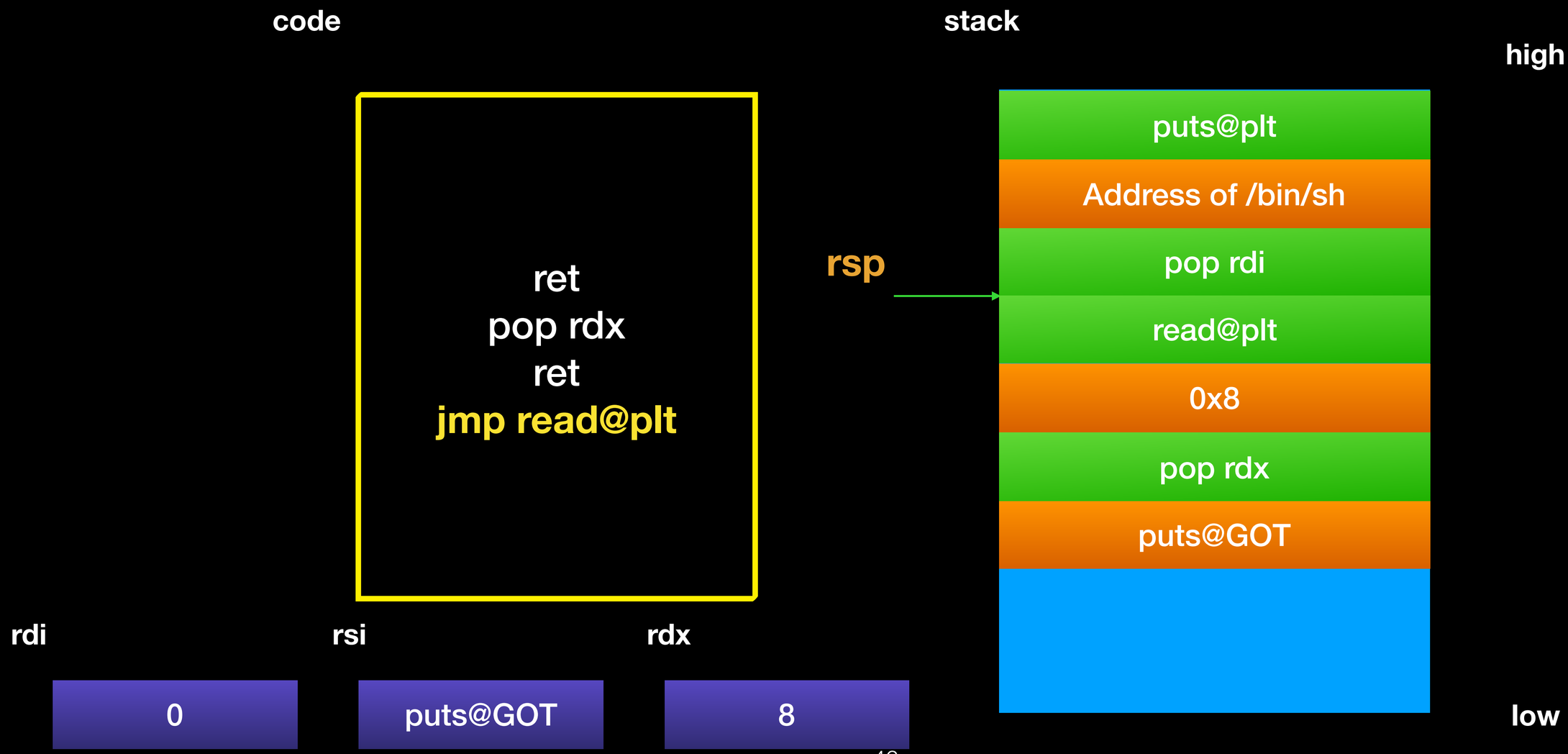
Using ROP bypass ASLR



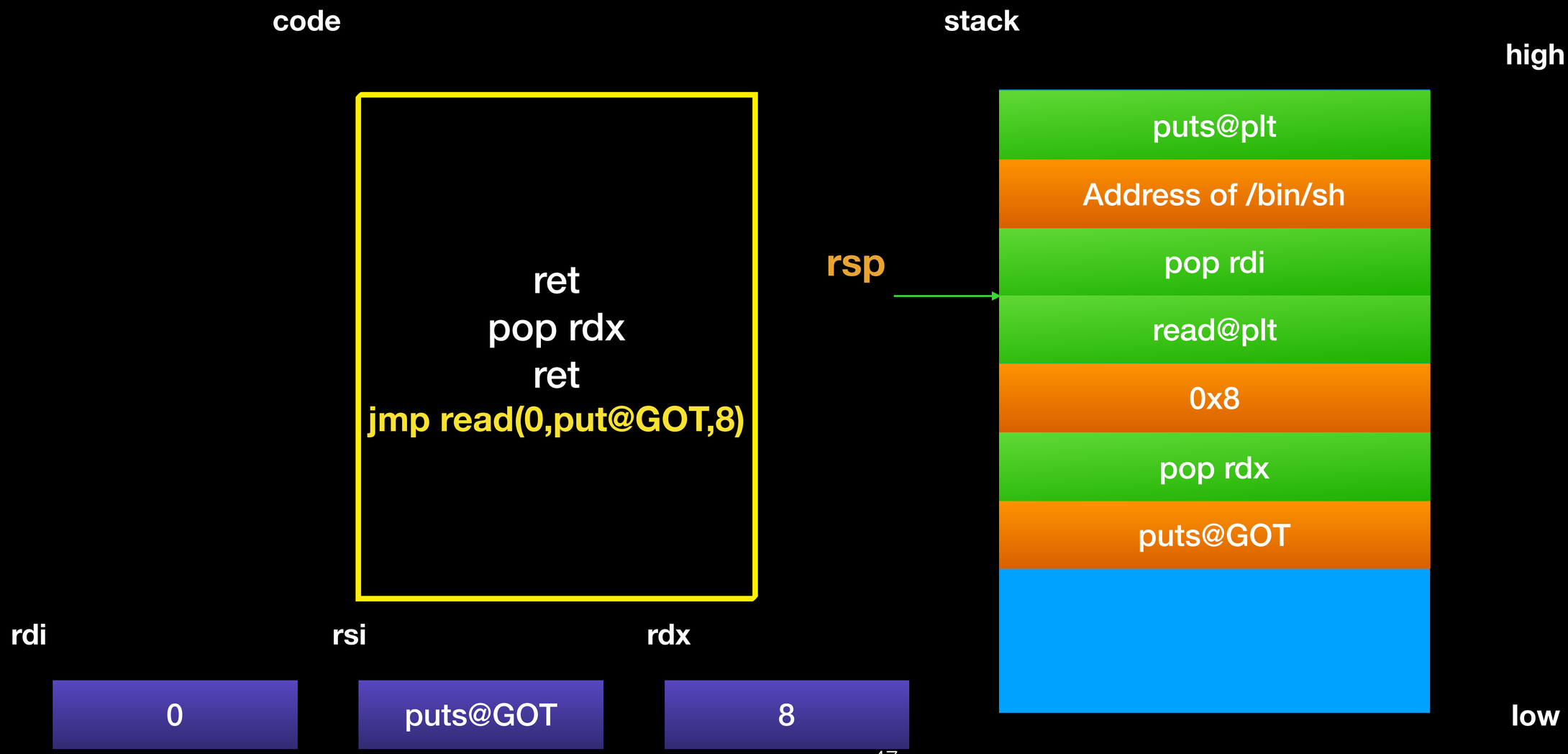
Using ROP bypass ASLR



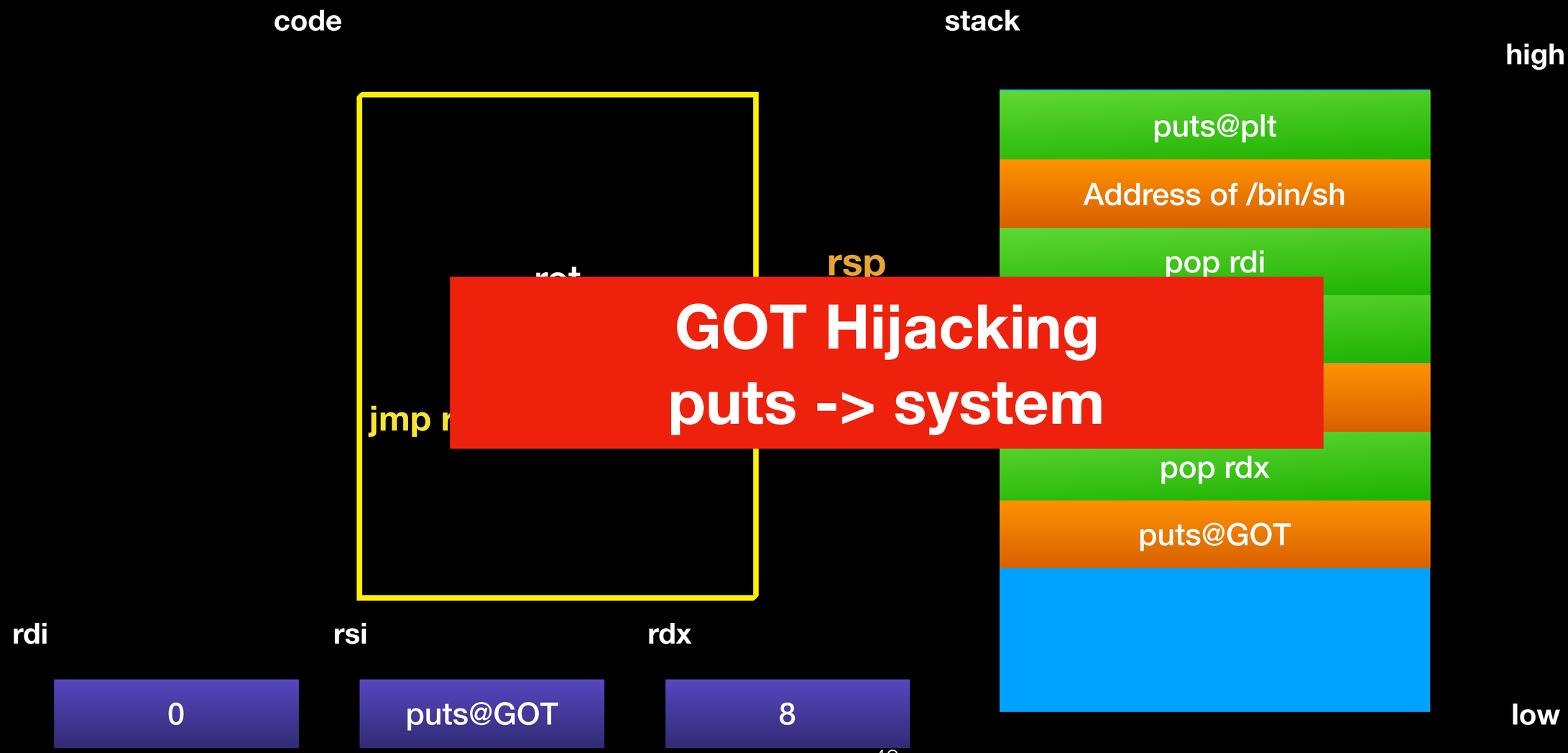
Using ROP bypass ASLR



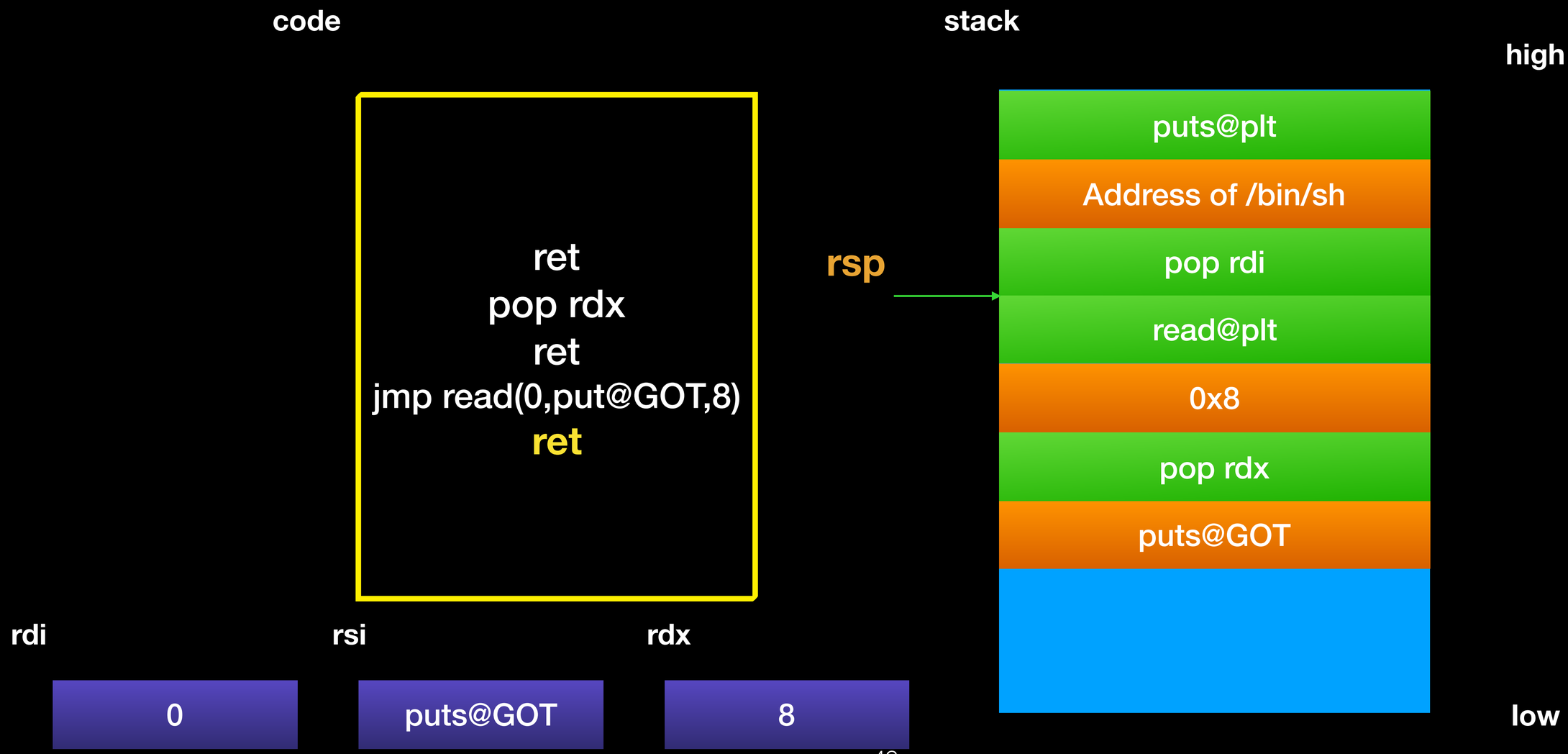
Using ROP bypass ASLR



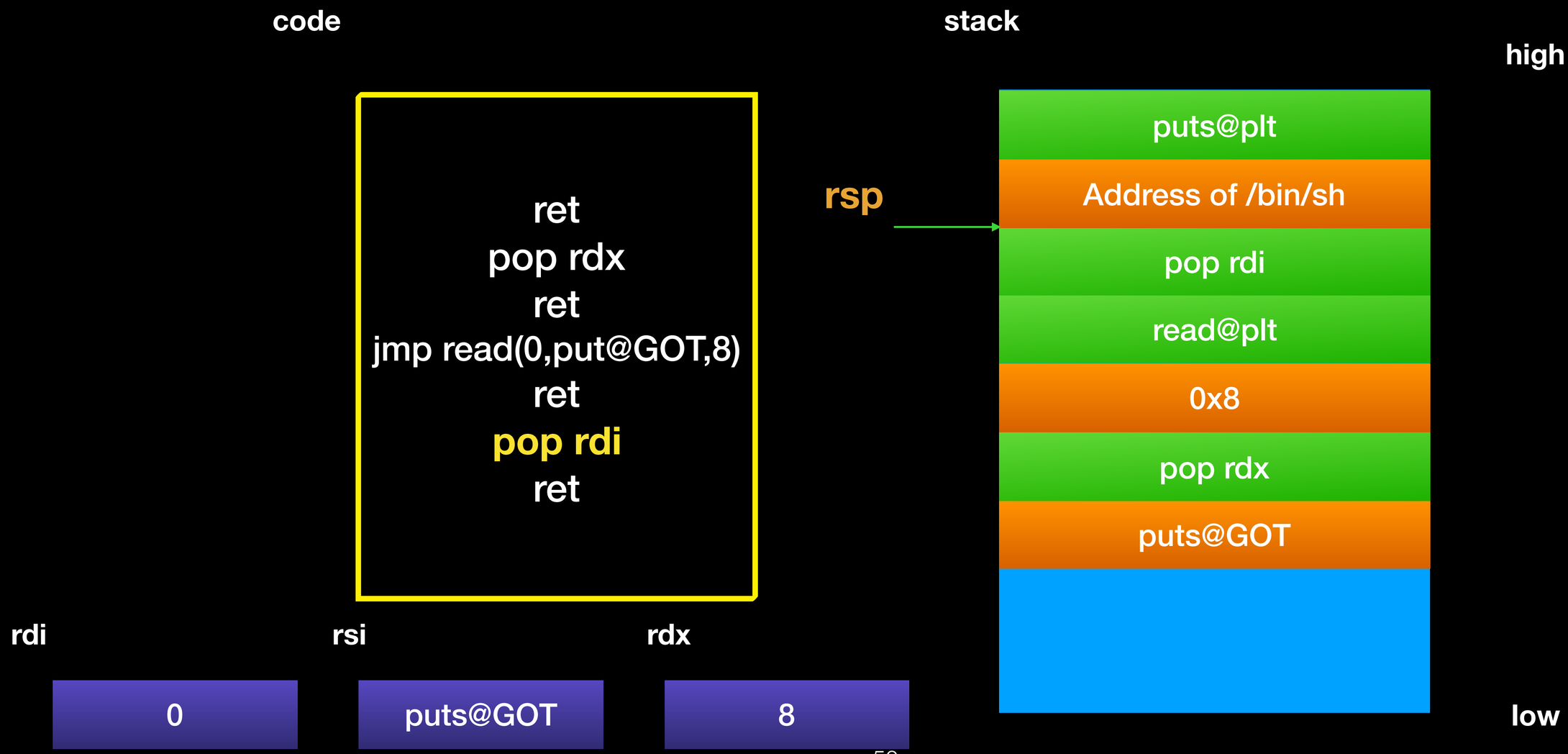
Using ROP bypass ASLR



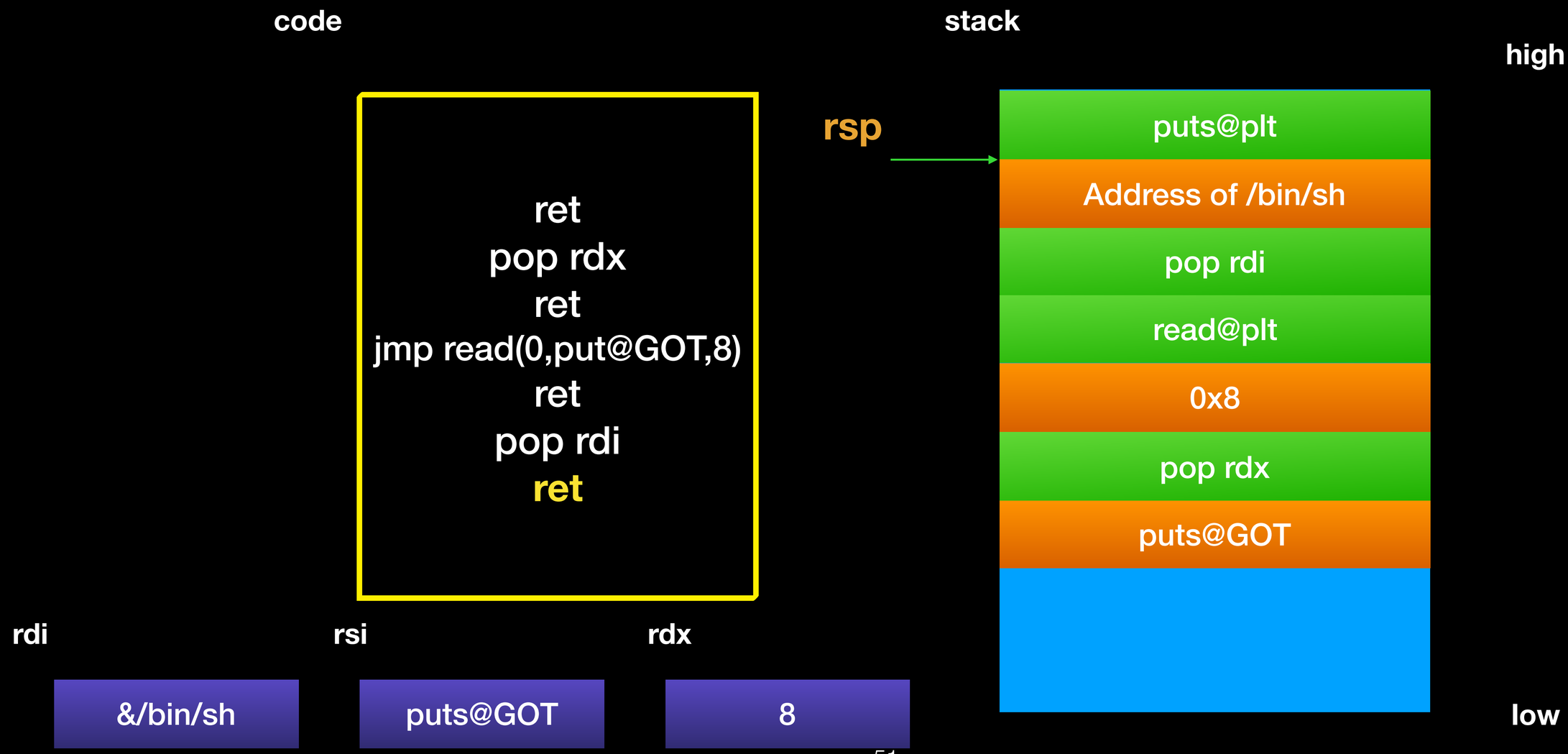
Using ROP bypass ASLR



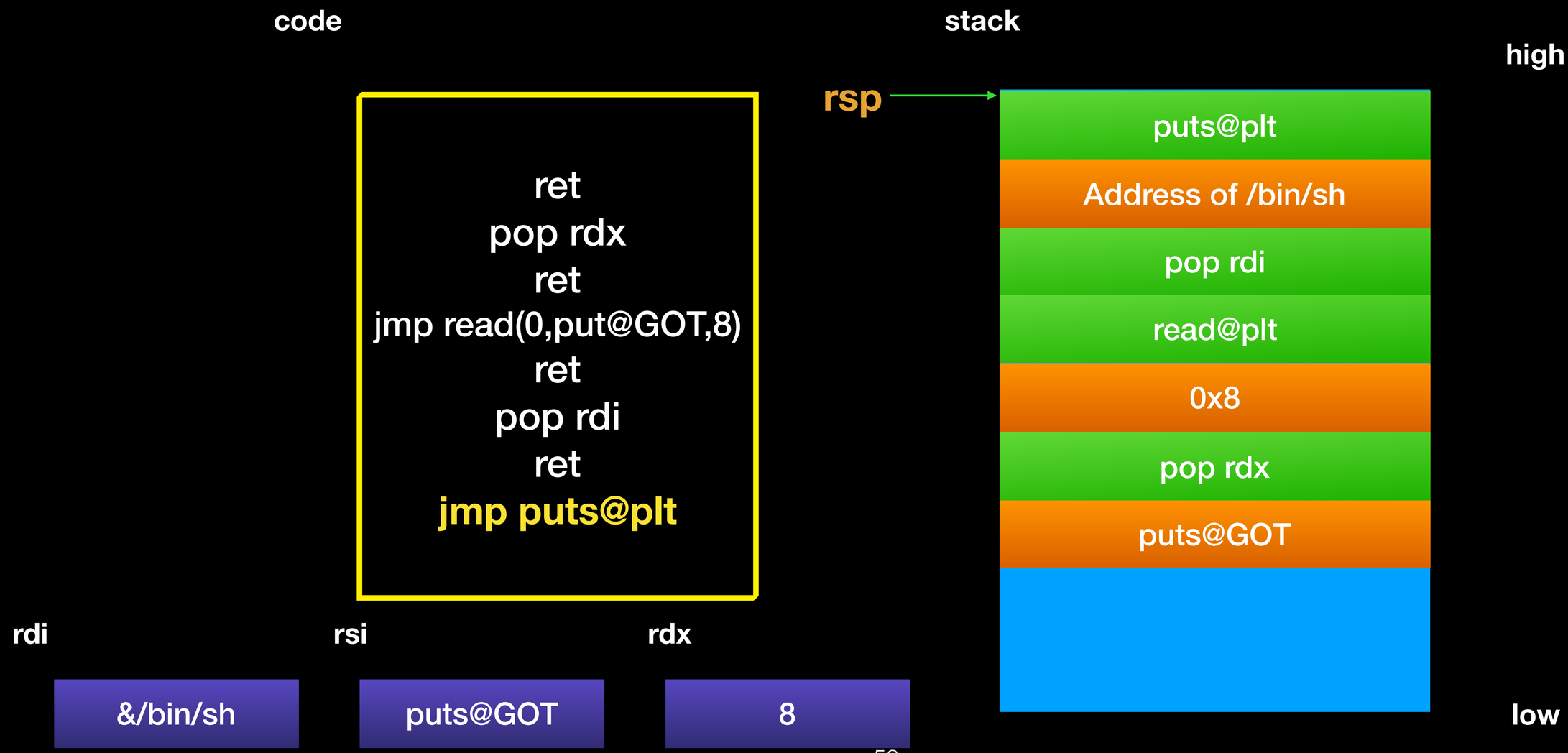
Using ROP bypass ASLR



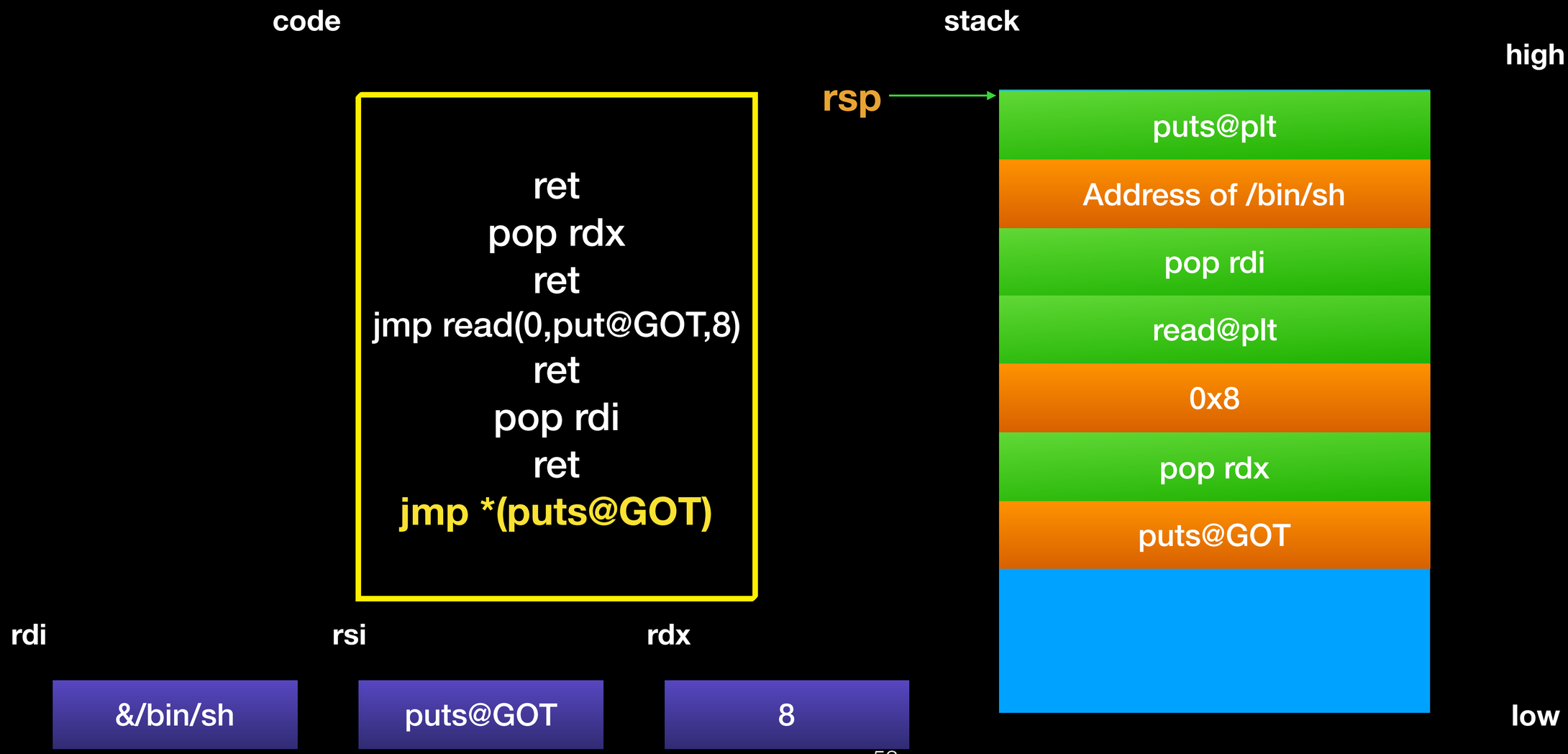
Using ROP bypass ASLR



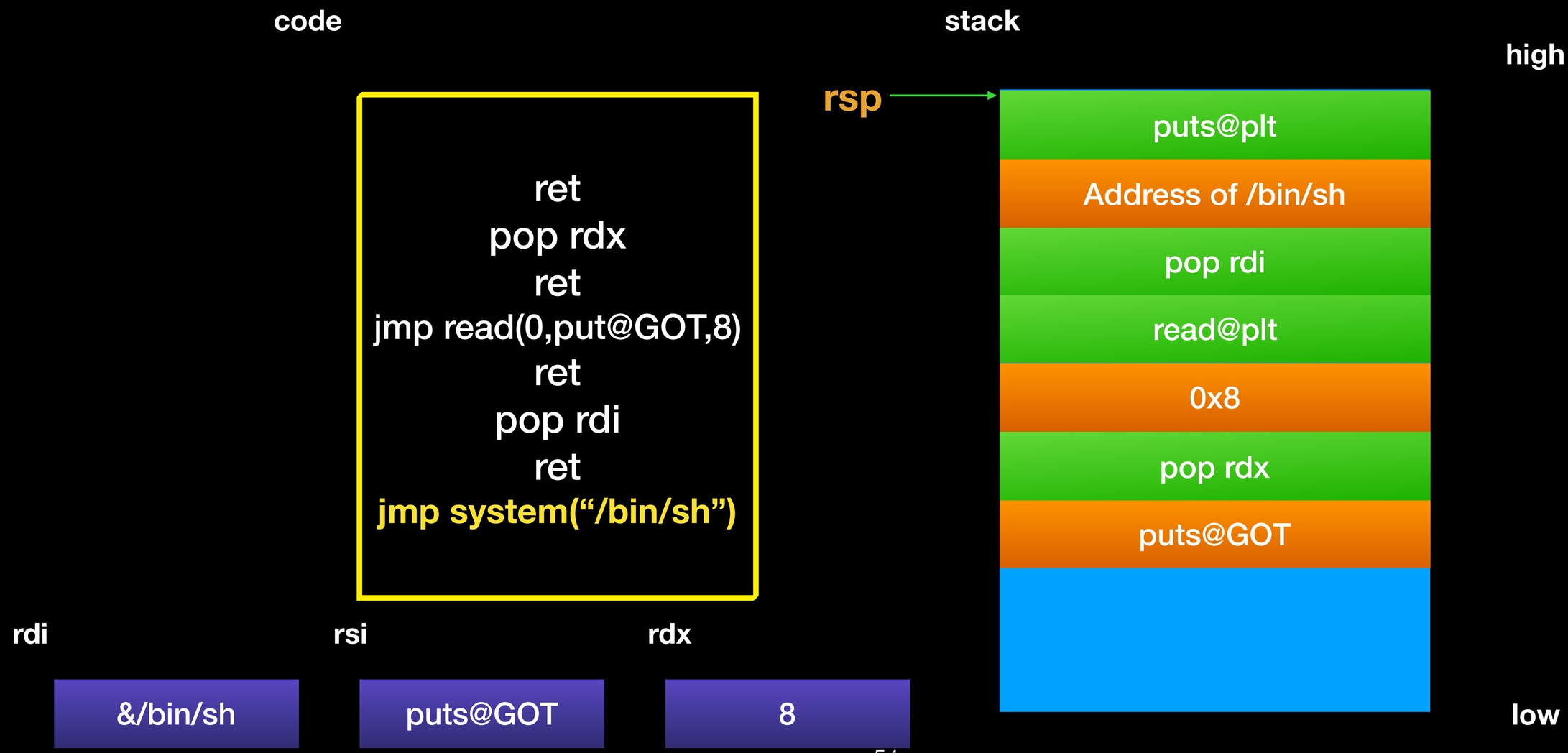
Using ROP bypass ASLR



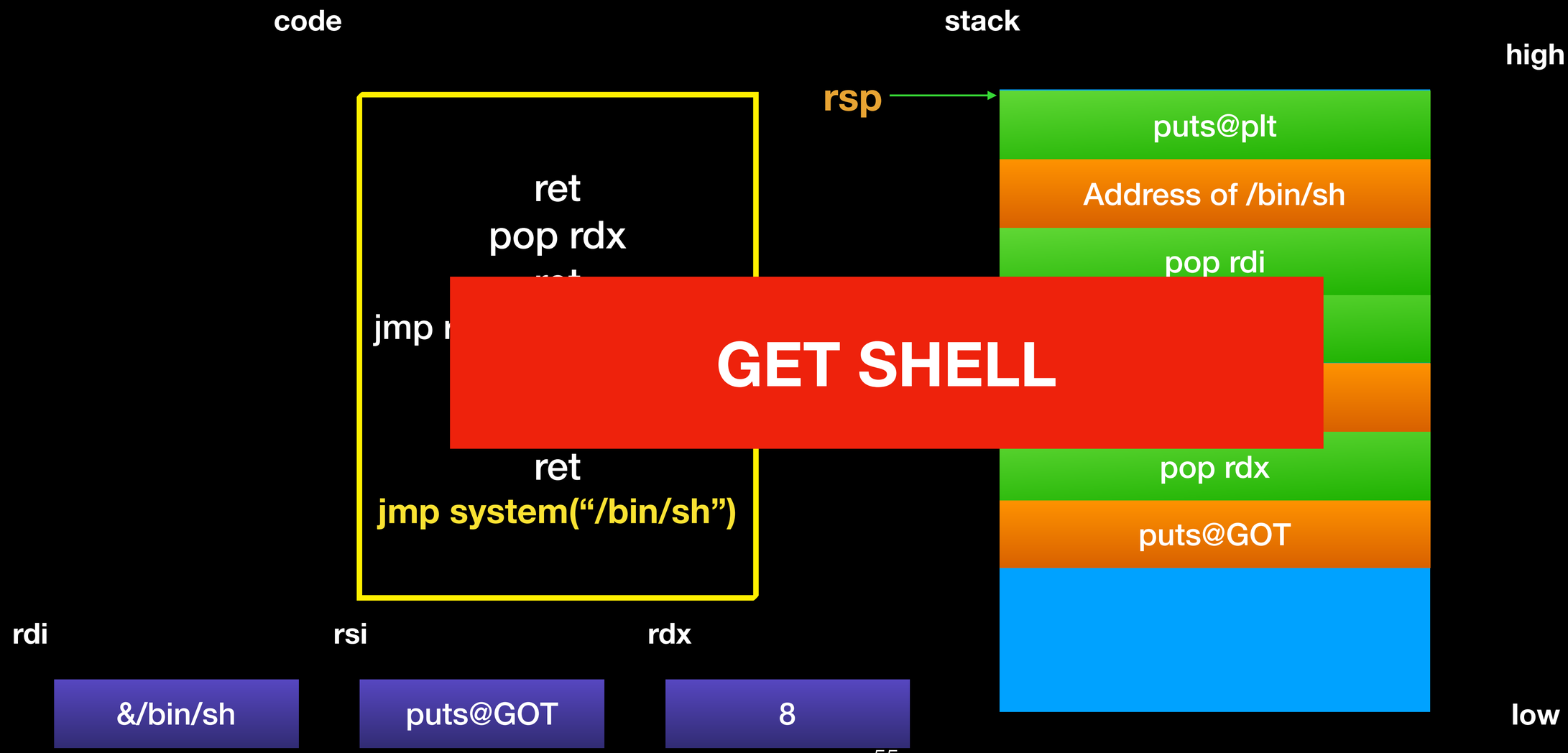
Using ROP bypass ASLR



Using ROP bypass ASLR



Using ROP bypass ASLR



Using ROP bypass ASLR

- Bypass PIE
  - 必須比平常多 leak 一個 code 段的位置，藉由這個值算出 code base 進而推出所有 GOT 等資訊
  - 有了 code base 之後其他就跟沒有 PIE 的情況下一樣

Using ROP bypass ASLR

- Bypass StackGuard
  - canary 只有在 function return 時做檢查
  - 只檢查 canary 值時否一樣
  - 所以可以先想辦法 leak 出 canary 的值，塞一模一樣的內容就可 bypass，或是想辦法不要改到 canary 也可以

Using ROP bypass ASLR

- Weakness in fork
  - canary and memory mappings are same as parent.

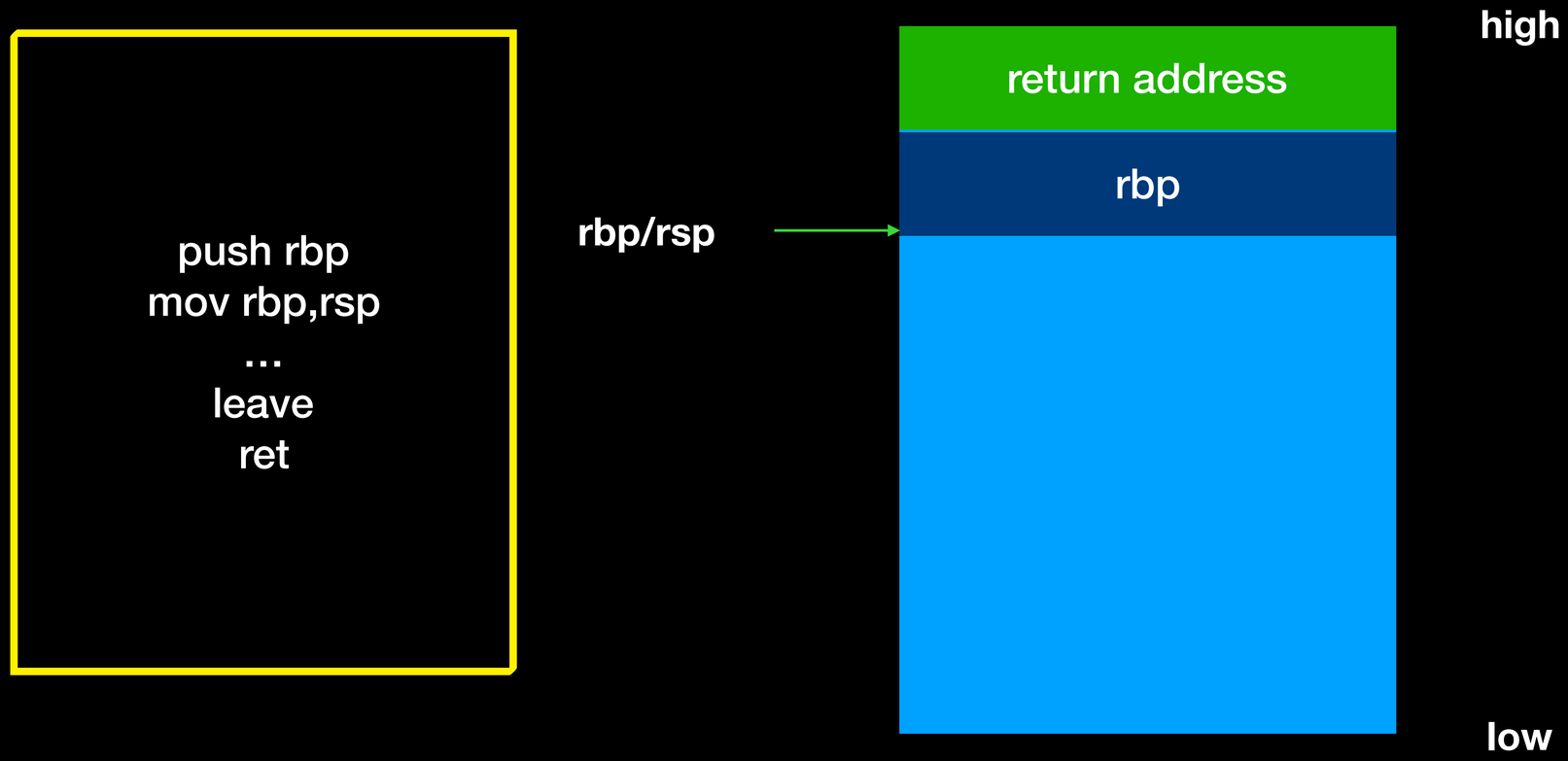
Outline

- ROP
- Using ROP bypass ASLR
- Stack migration

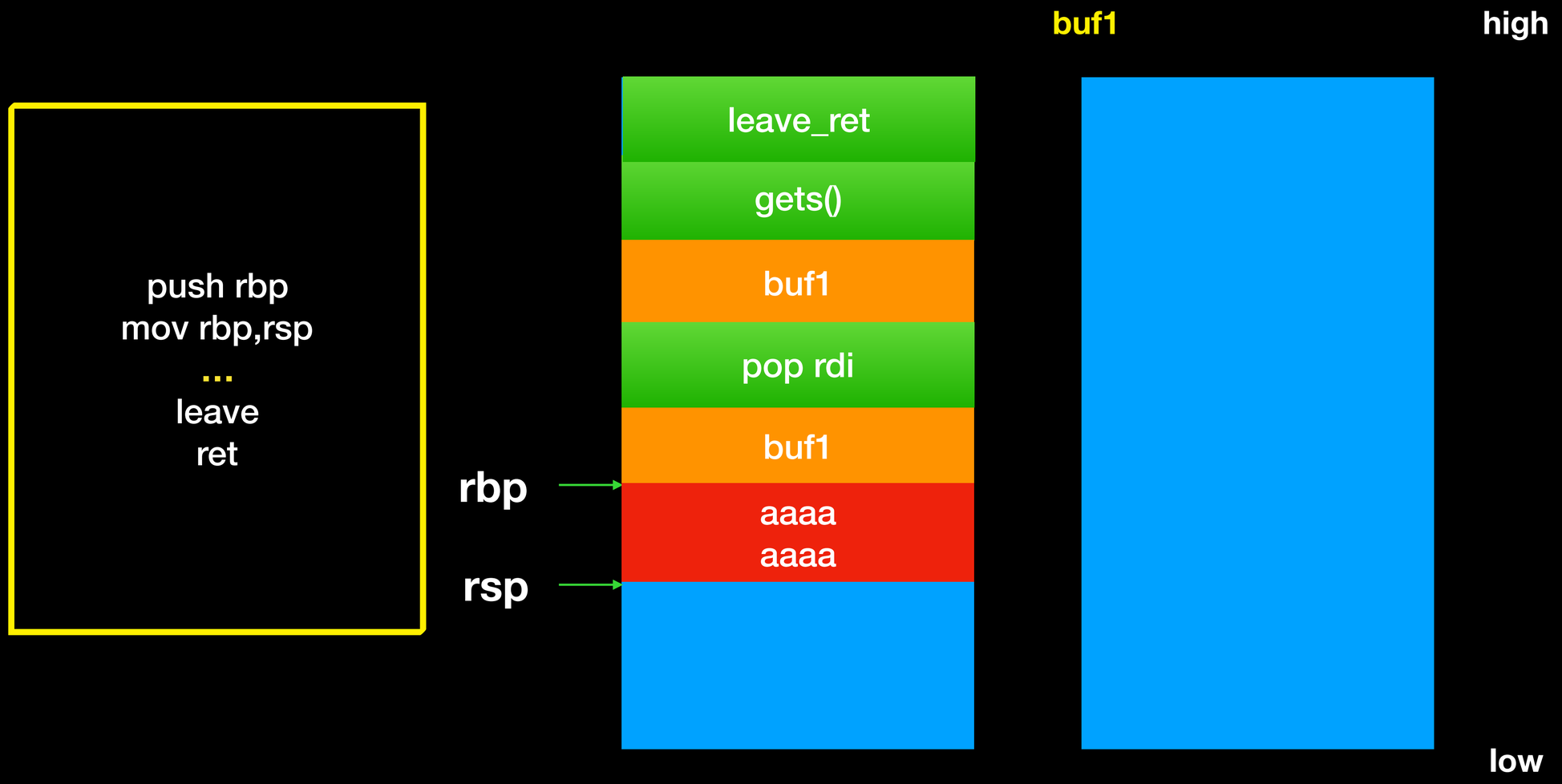
Stack Migration

- 將 ROP Chain 寫在已知固定位置上
- 再利用 leave 搬移 Stack 位置到已知位置
- 可無限接 ROP Chain
- 必須注意到 Migration 之後 stack 要留大一點，有些 function 可能會需要很大的 stack frame ，太小可能會存取到唯獨區域，導致 Segmentation Fault

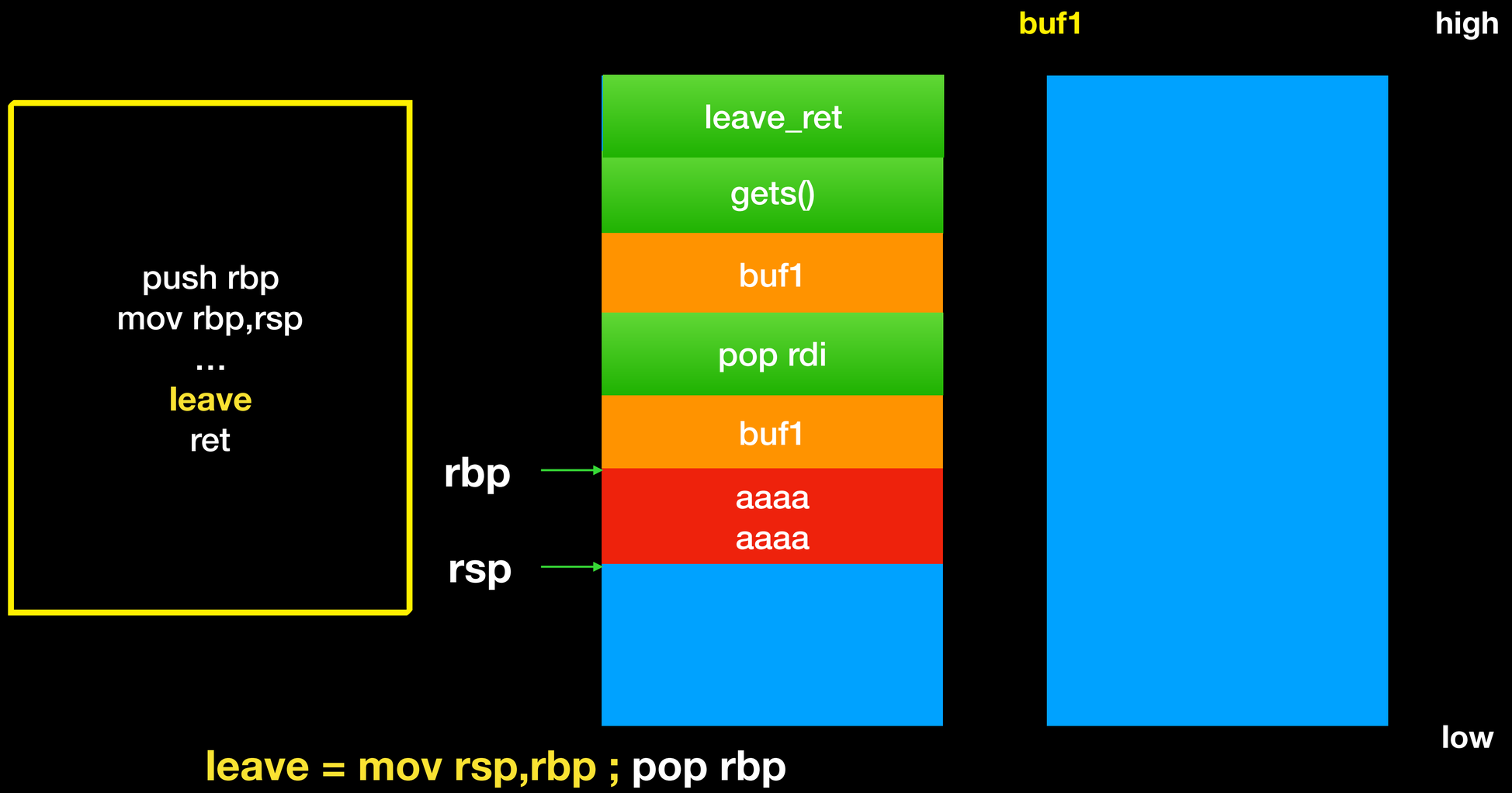
Stack Migration



Stack Migration

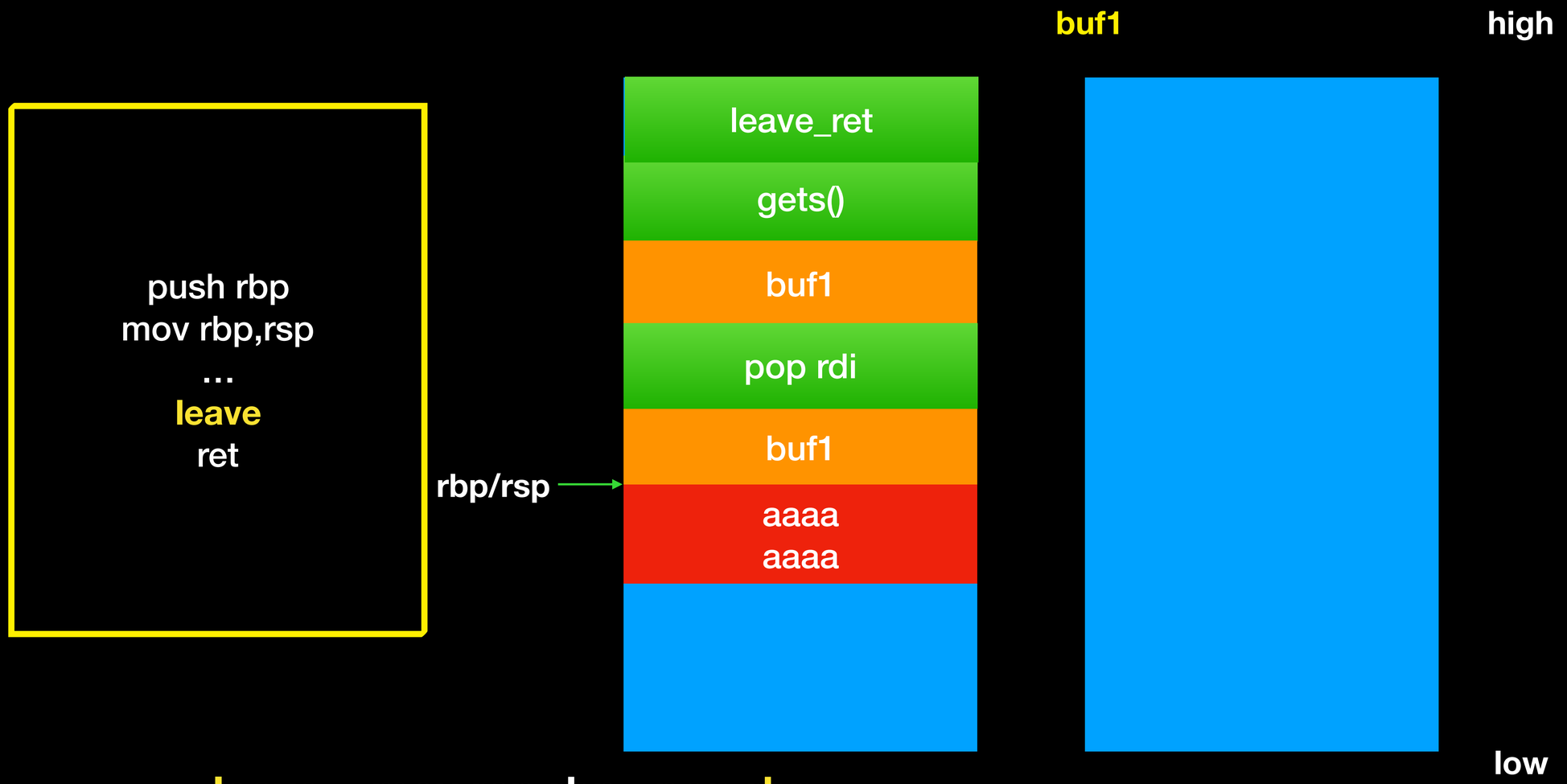


Stack Migration



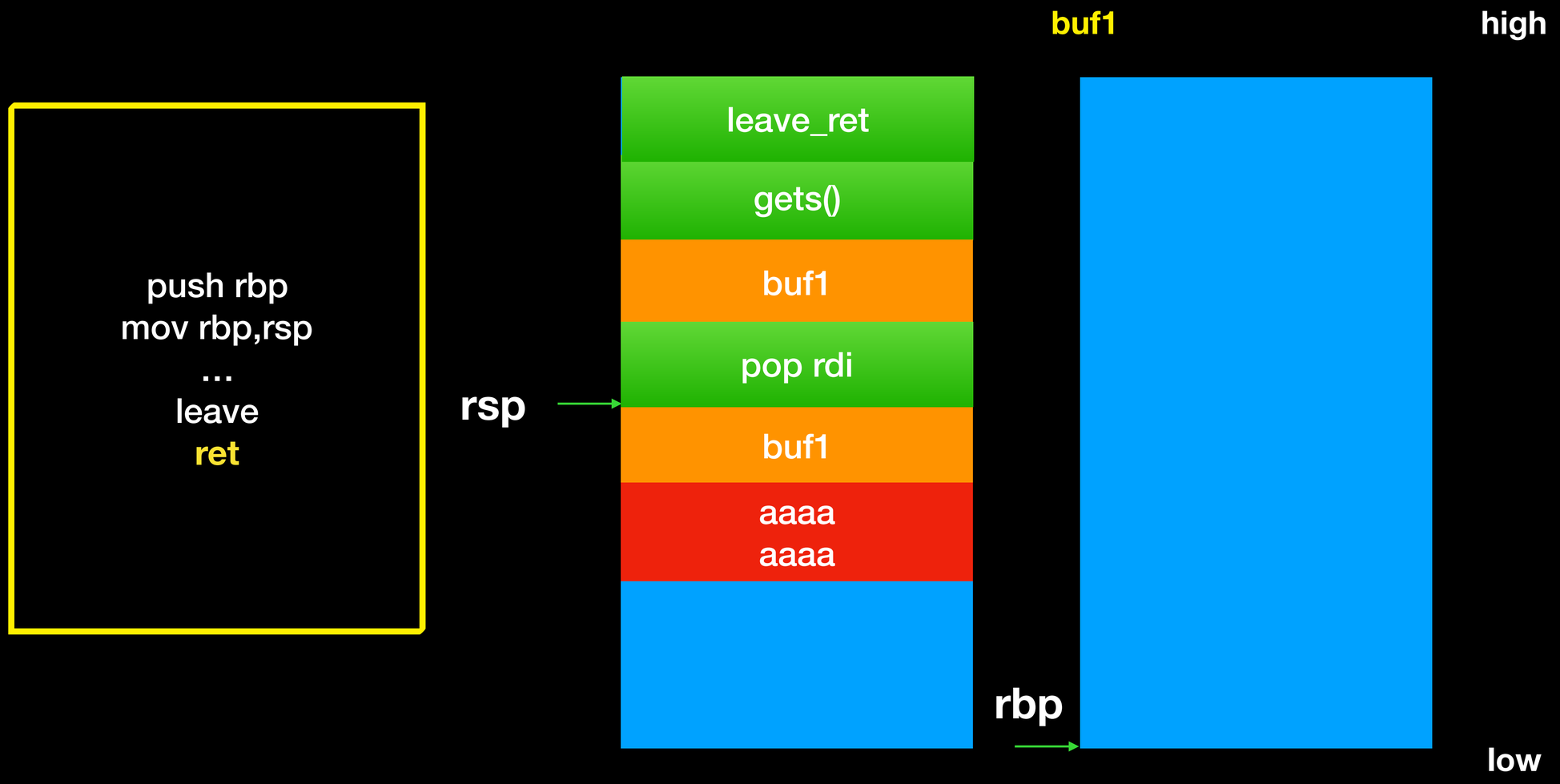
leave = mov rsp,rbp ; pop rbp

Stack Migration

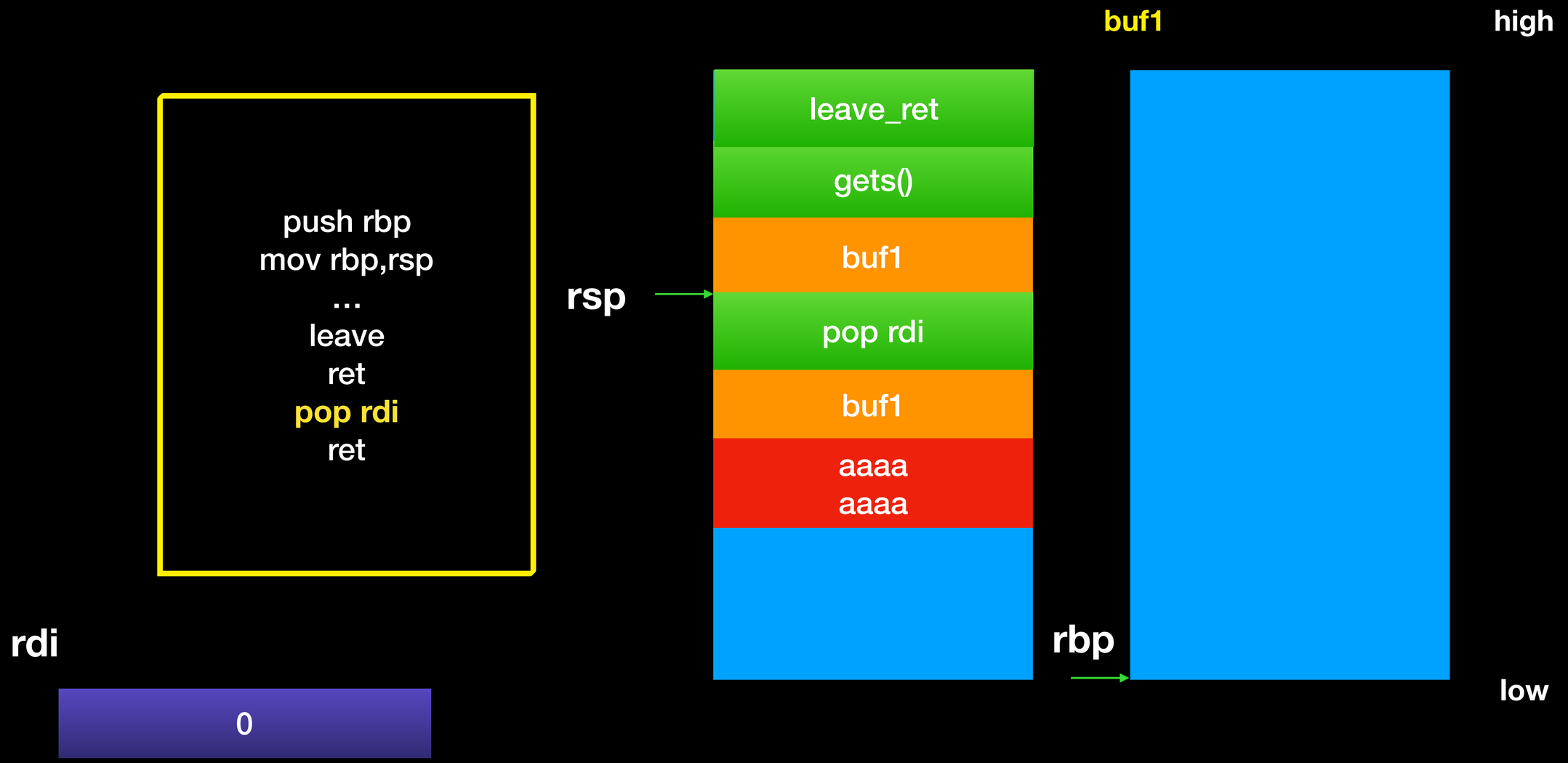


leave = mov rsp,rbp ; pop rbp

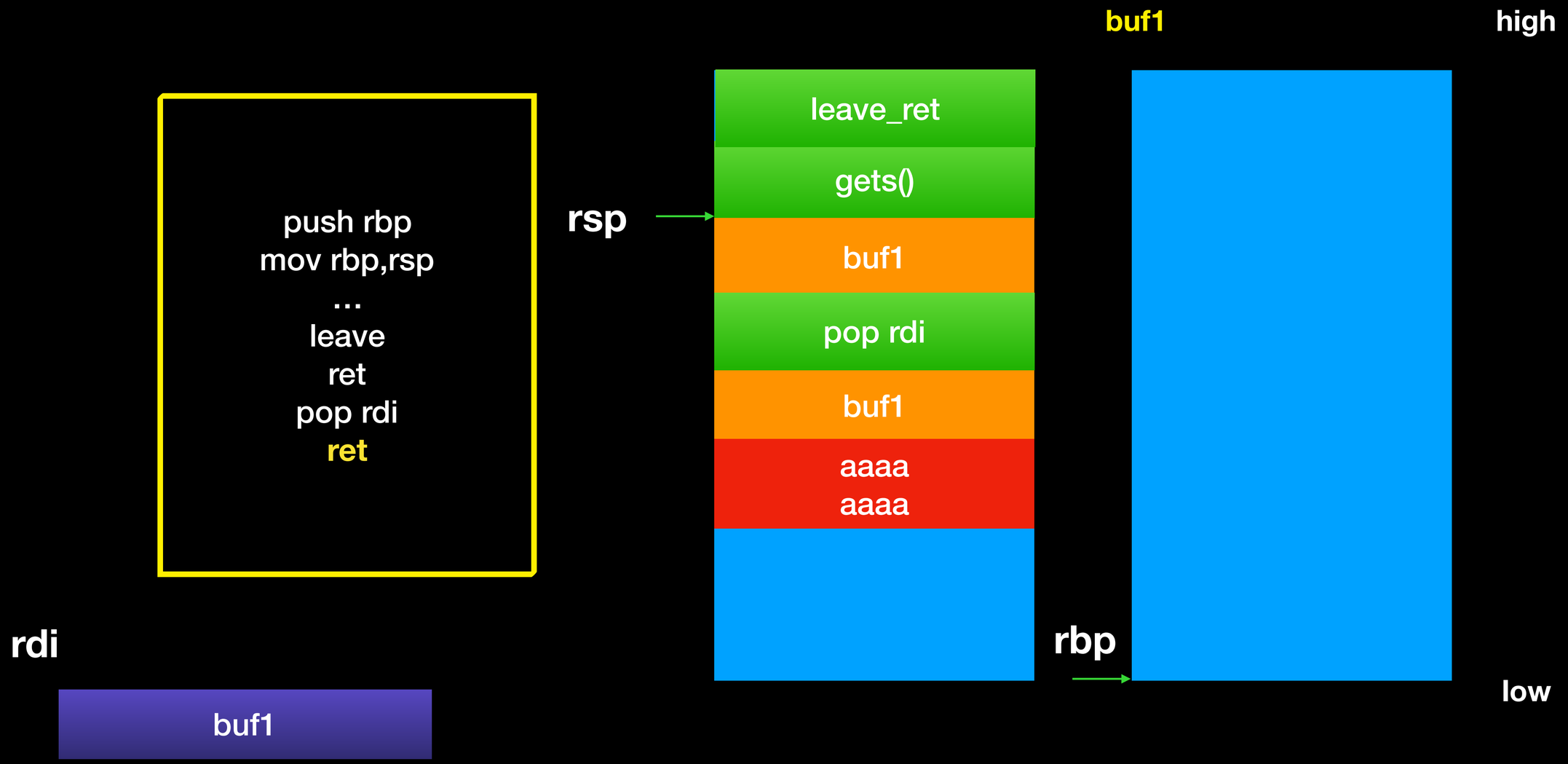
Stack Migration



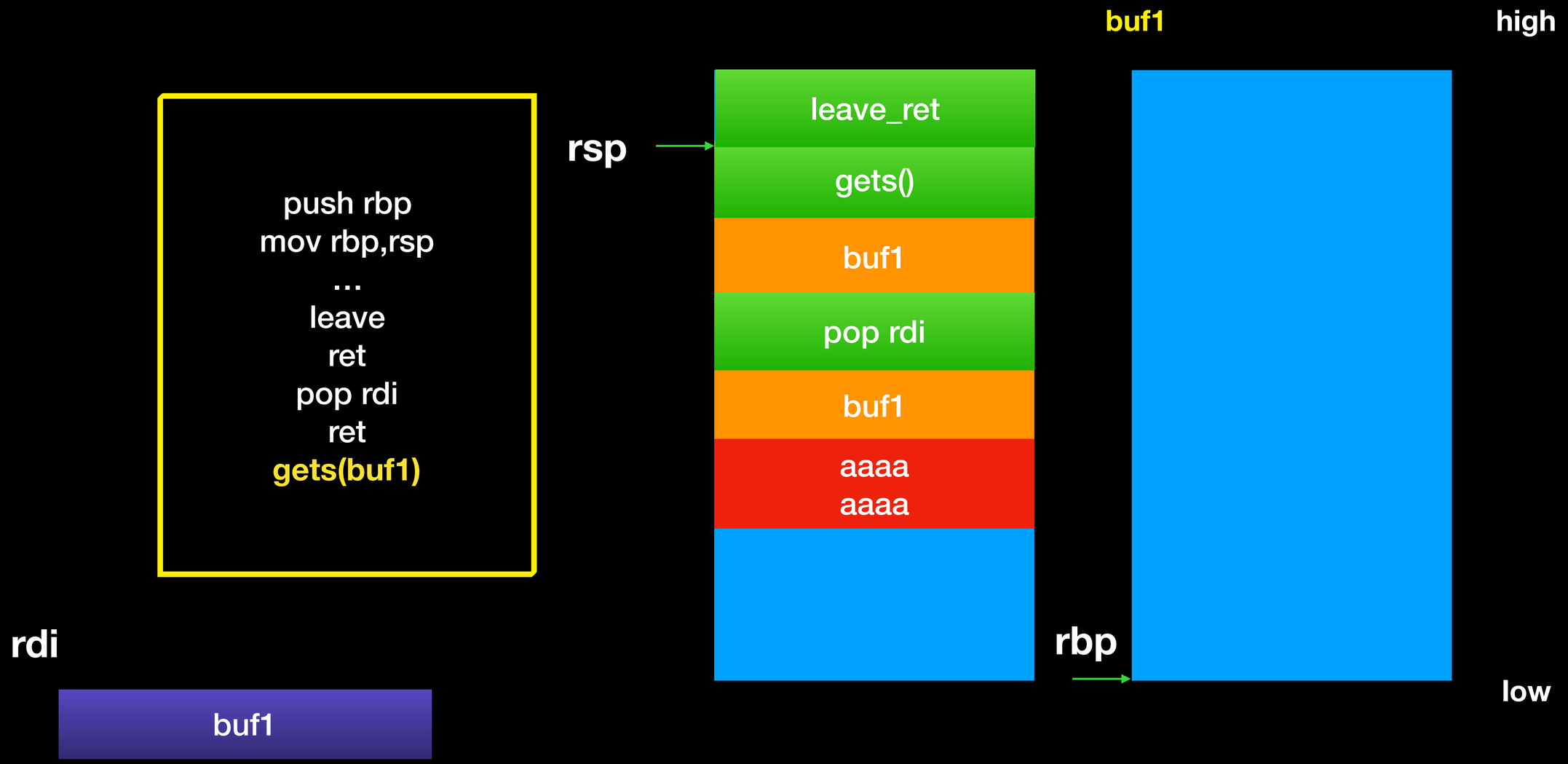
Stack Migration



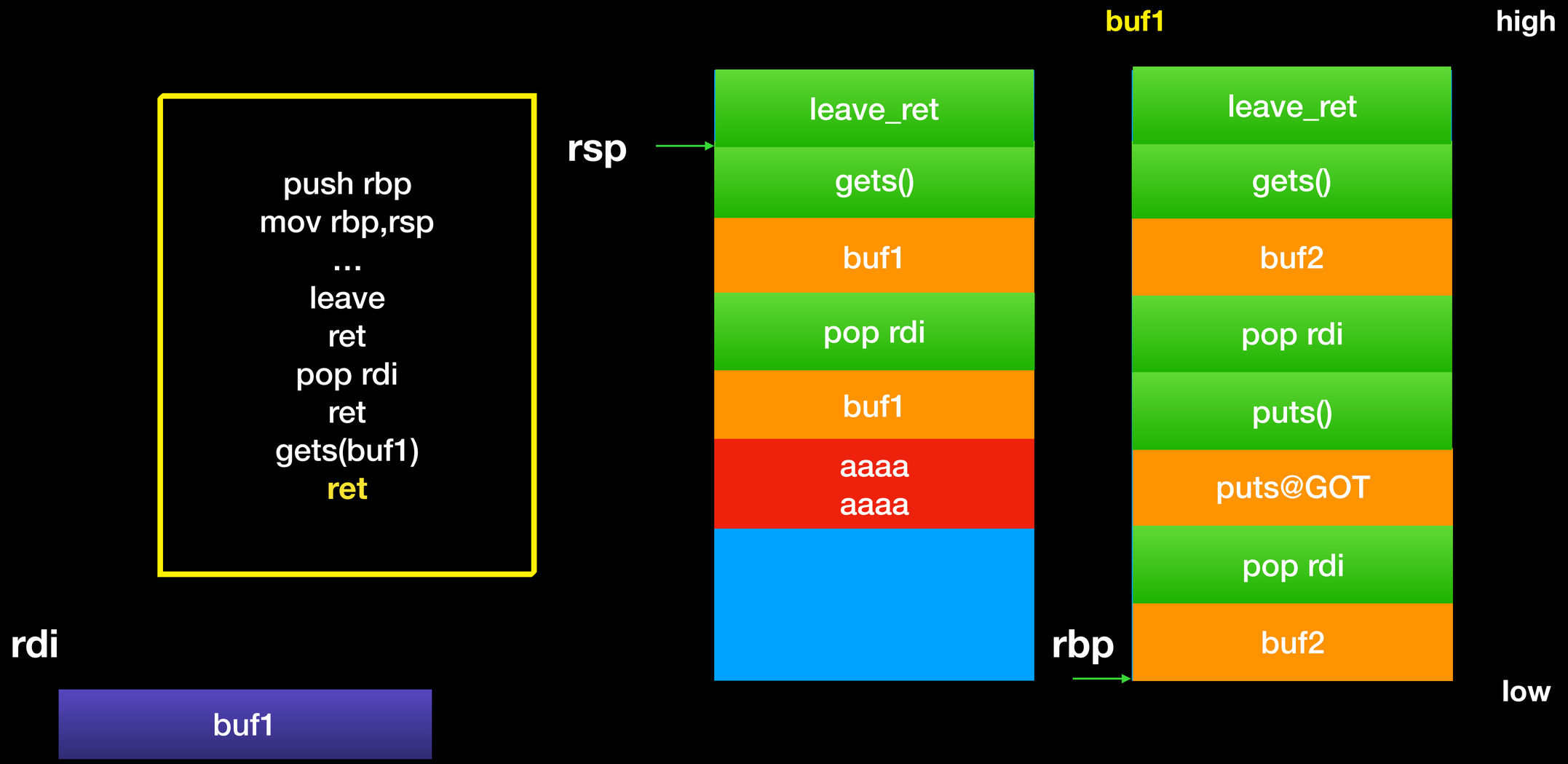
Stack Migration



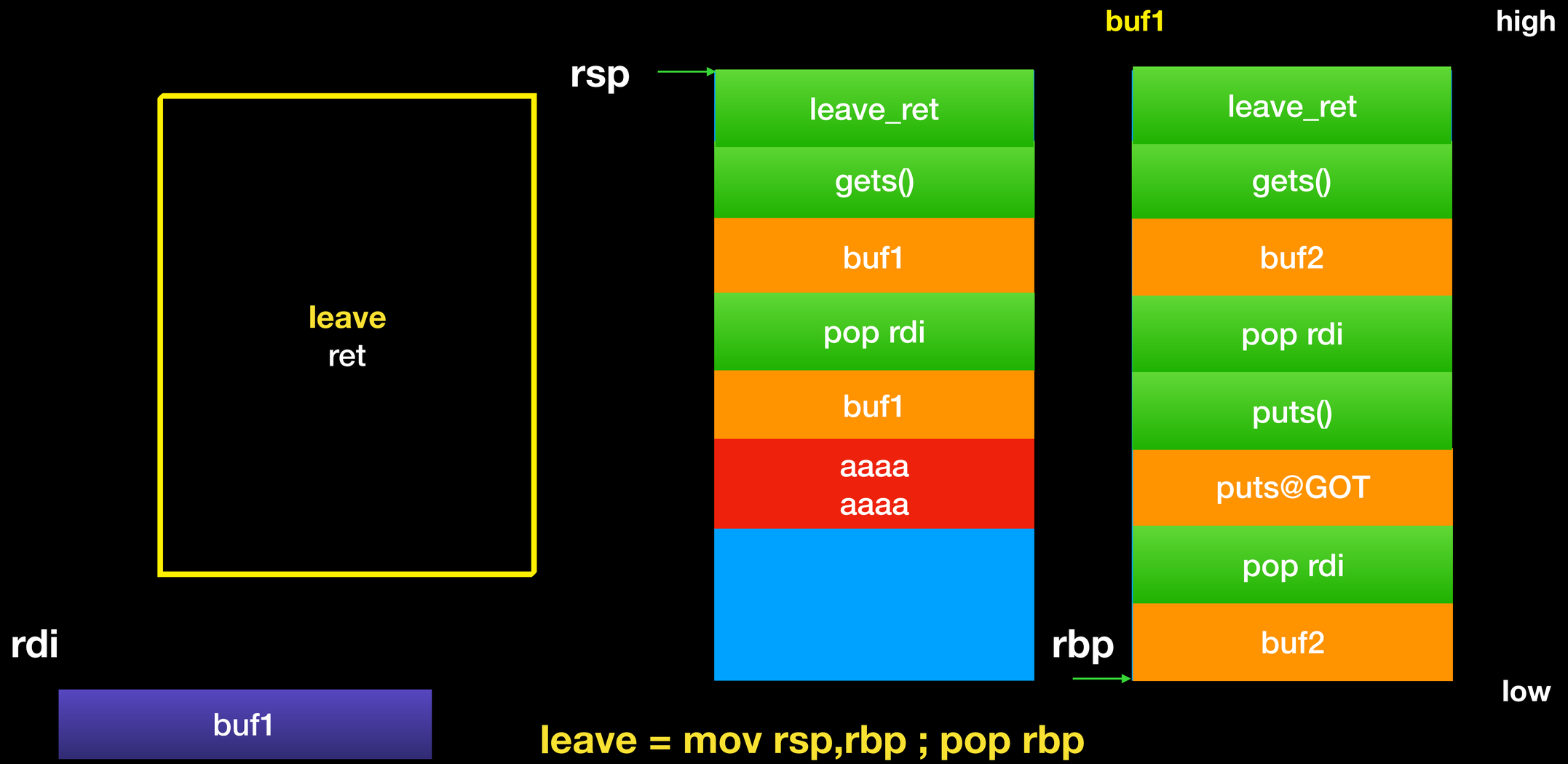
Stack Migration



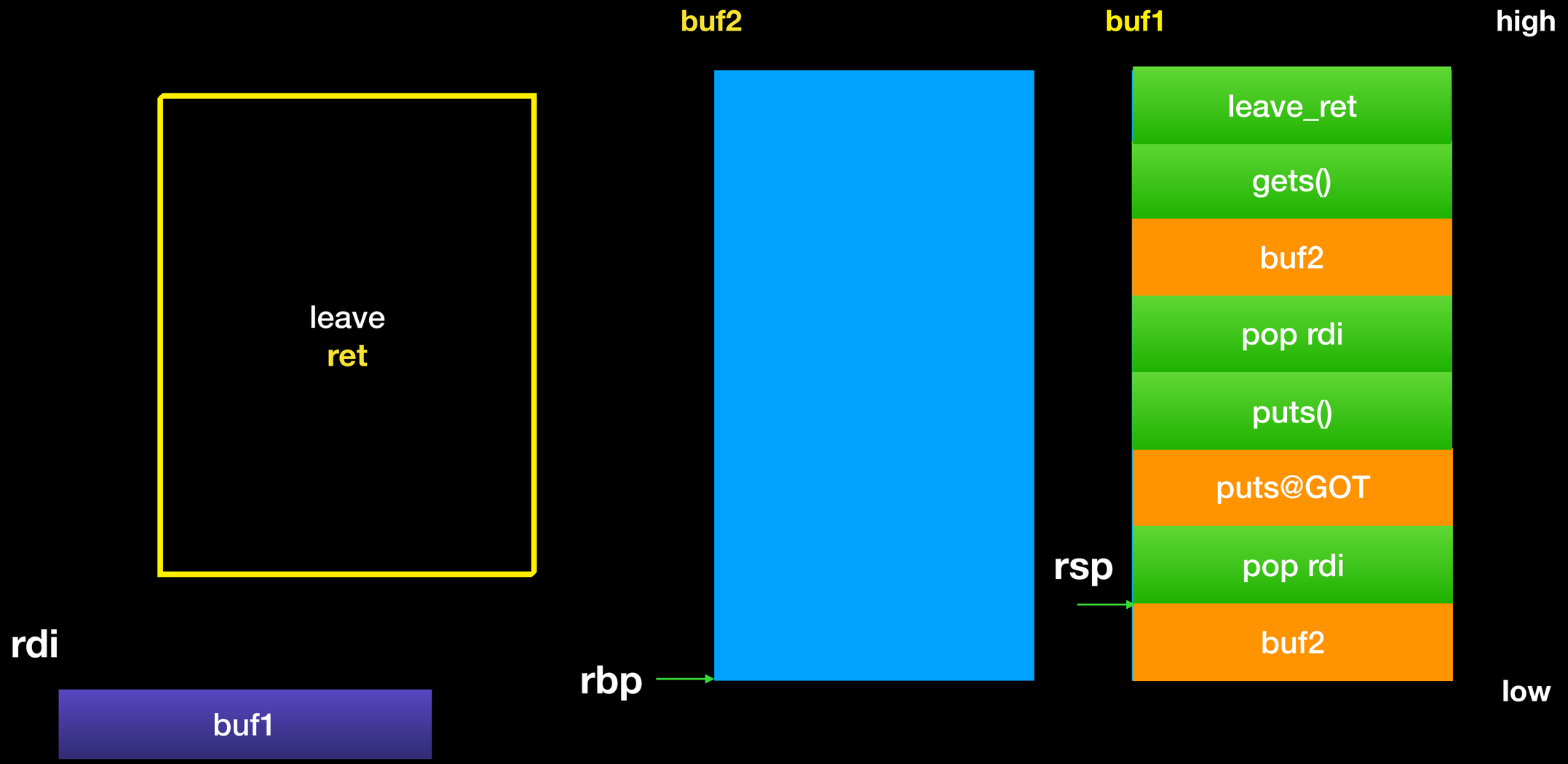
Stack Migration



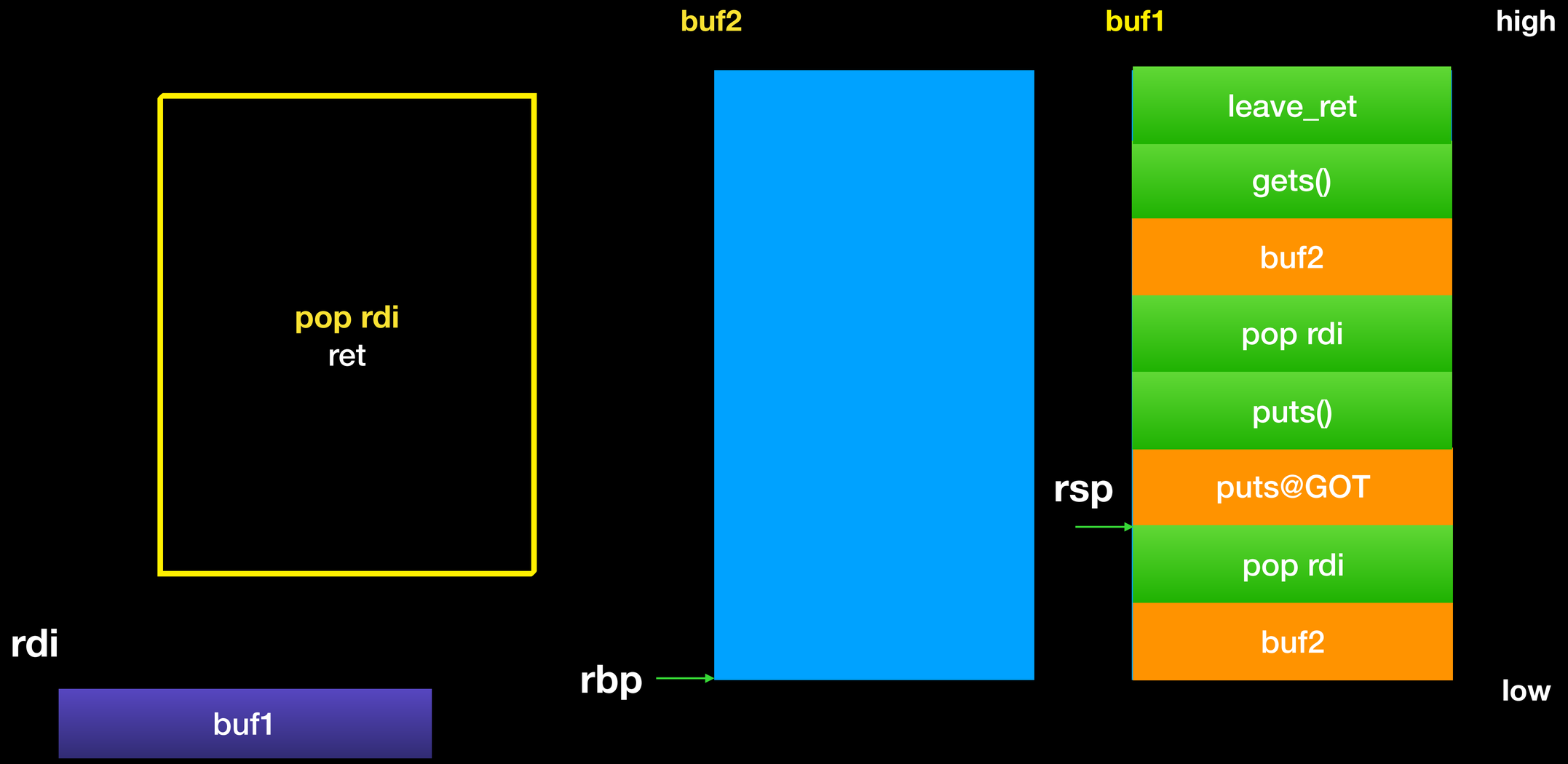
Stack Migration



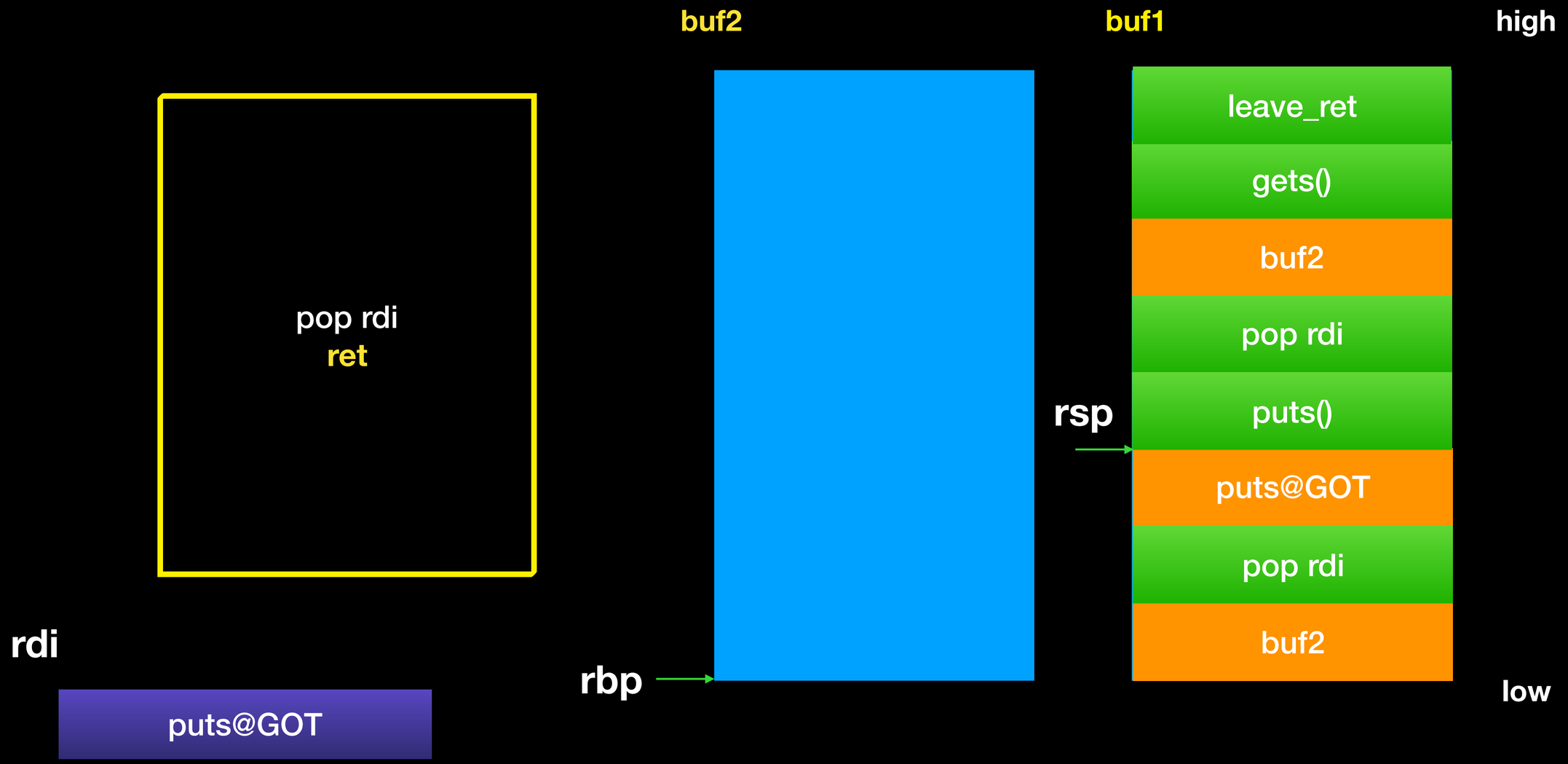
Stack Migration



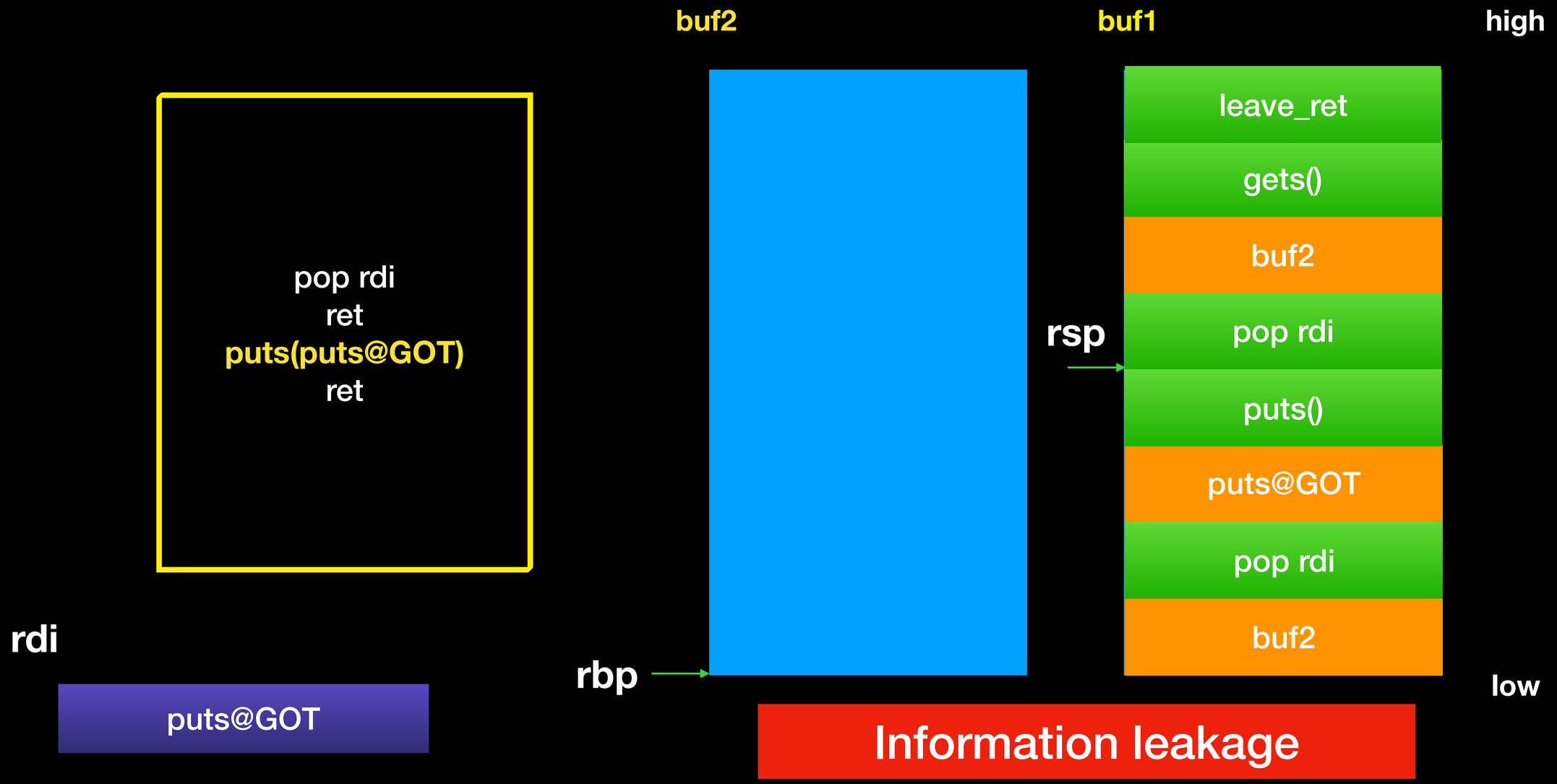
Stack Migration



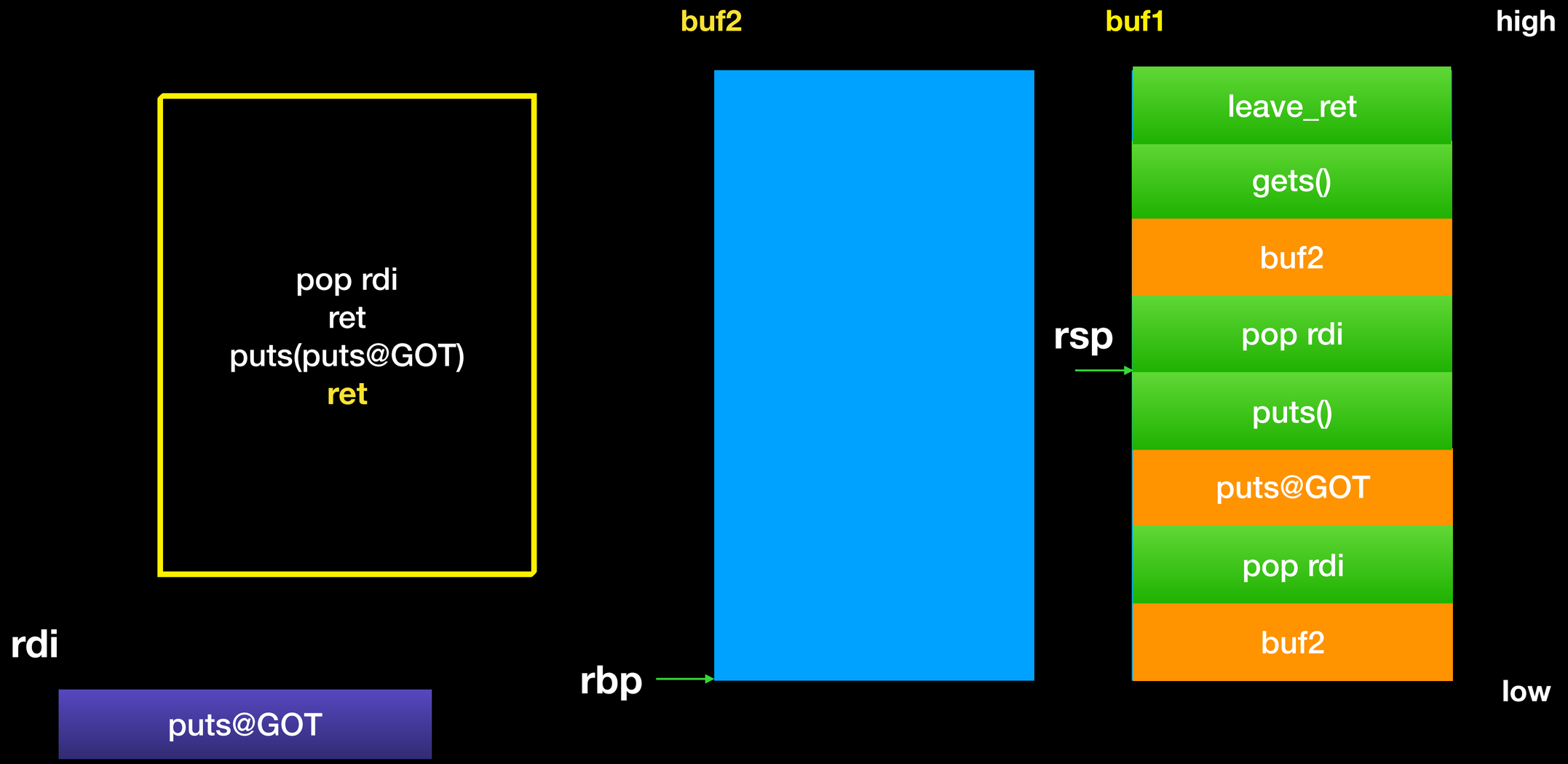
Stack Migration



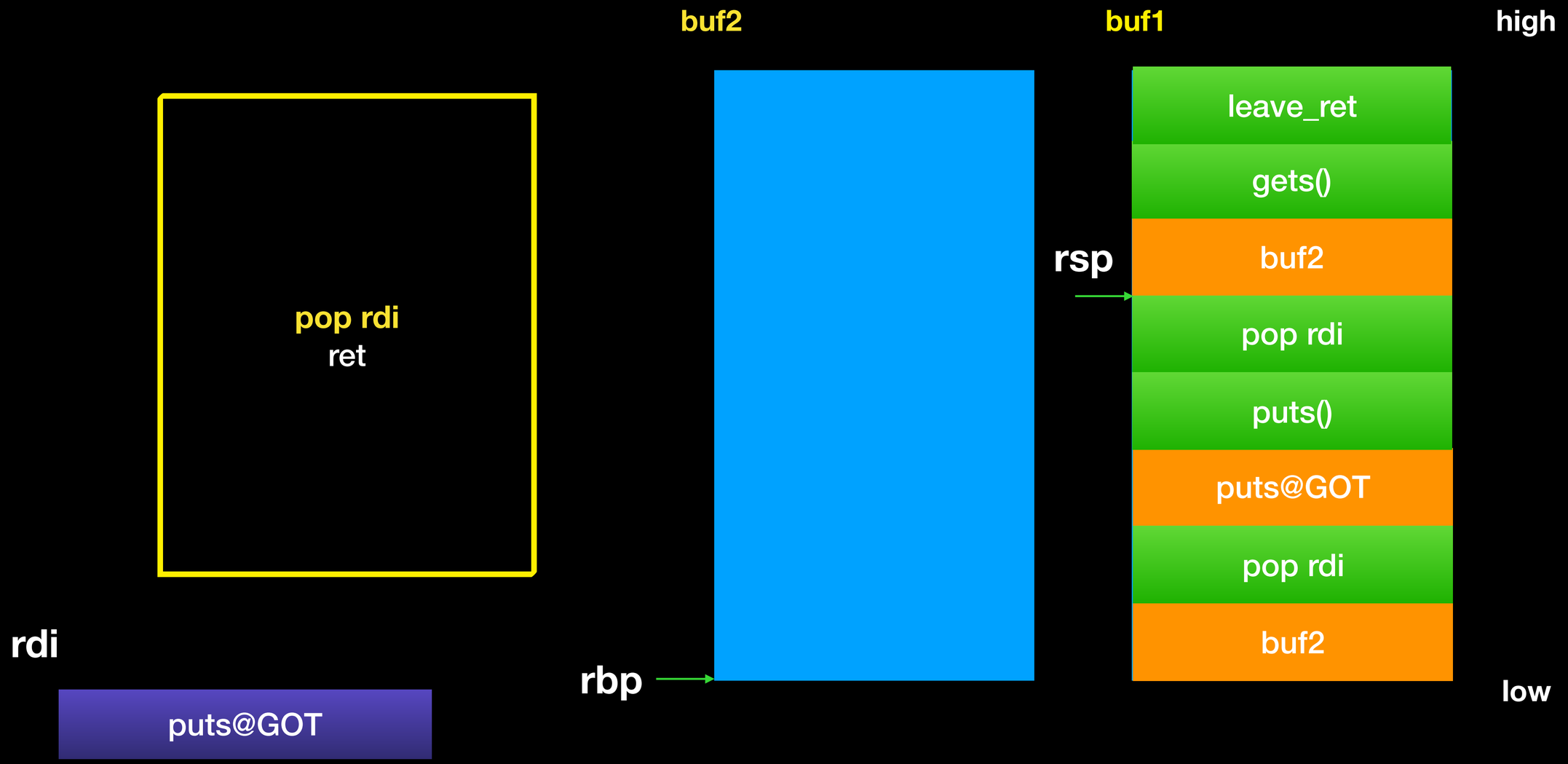
Stack Migration



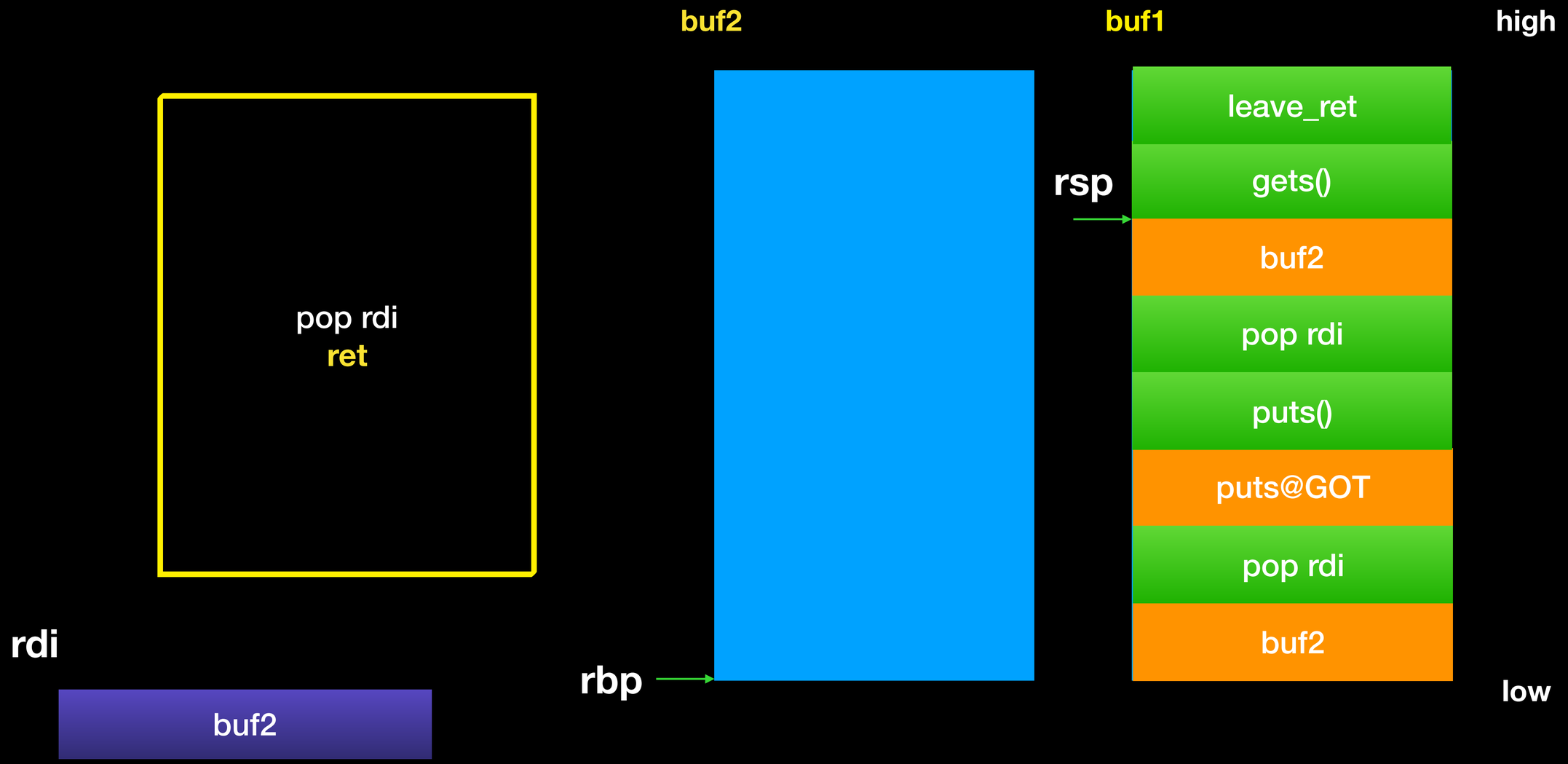
Stack Migration



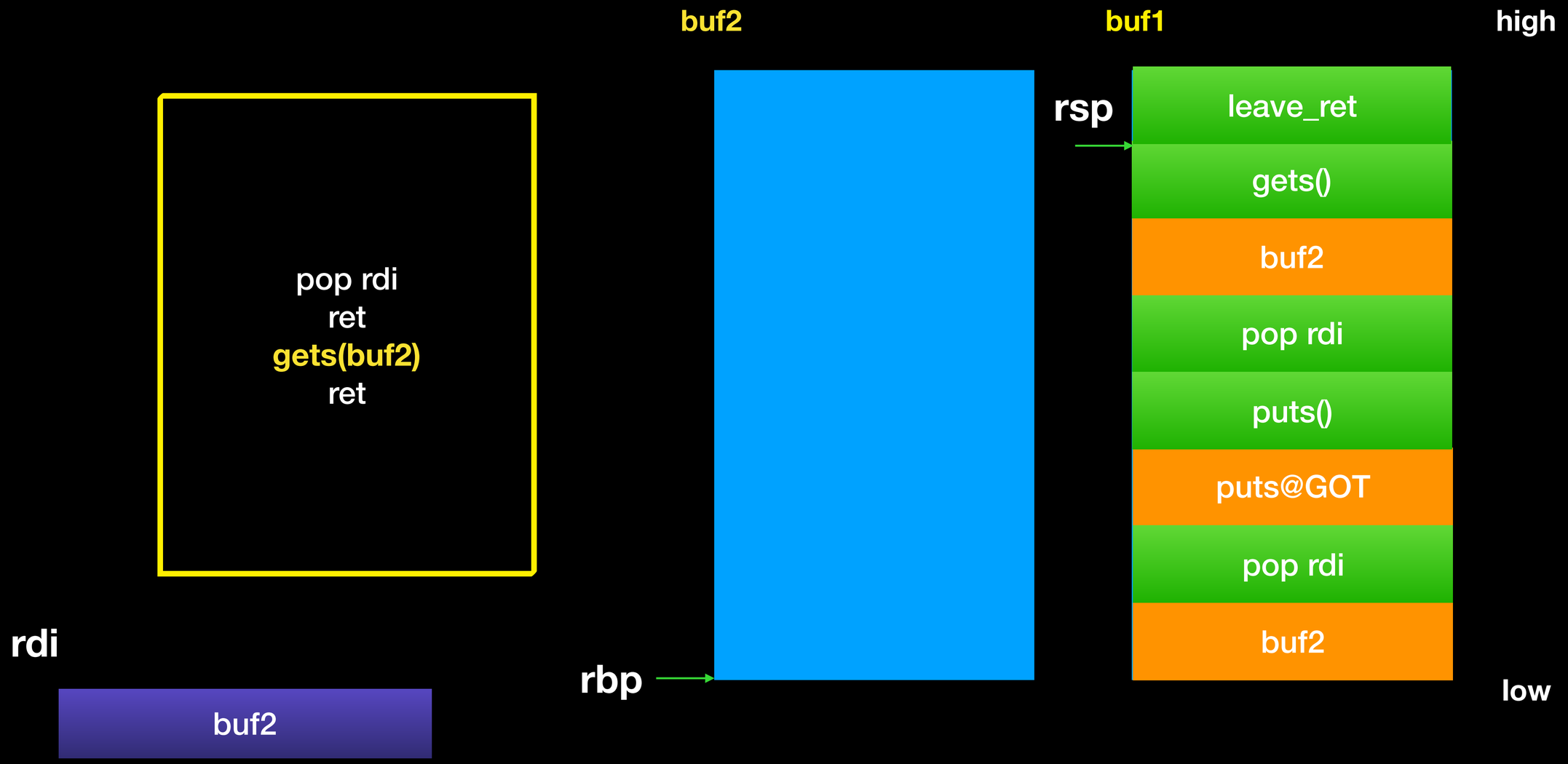
Stack Migration



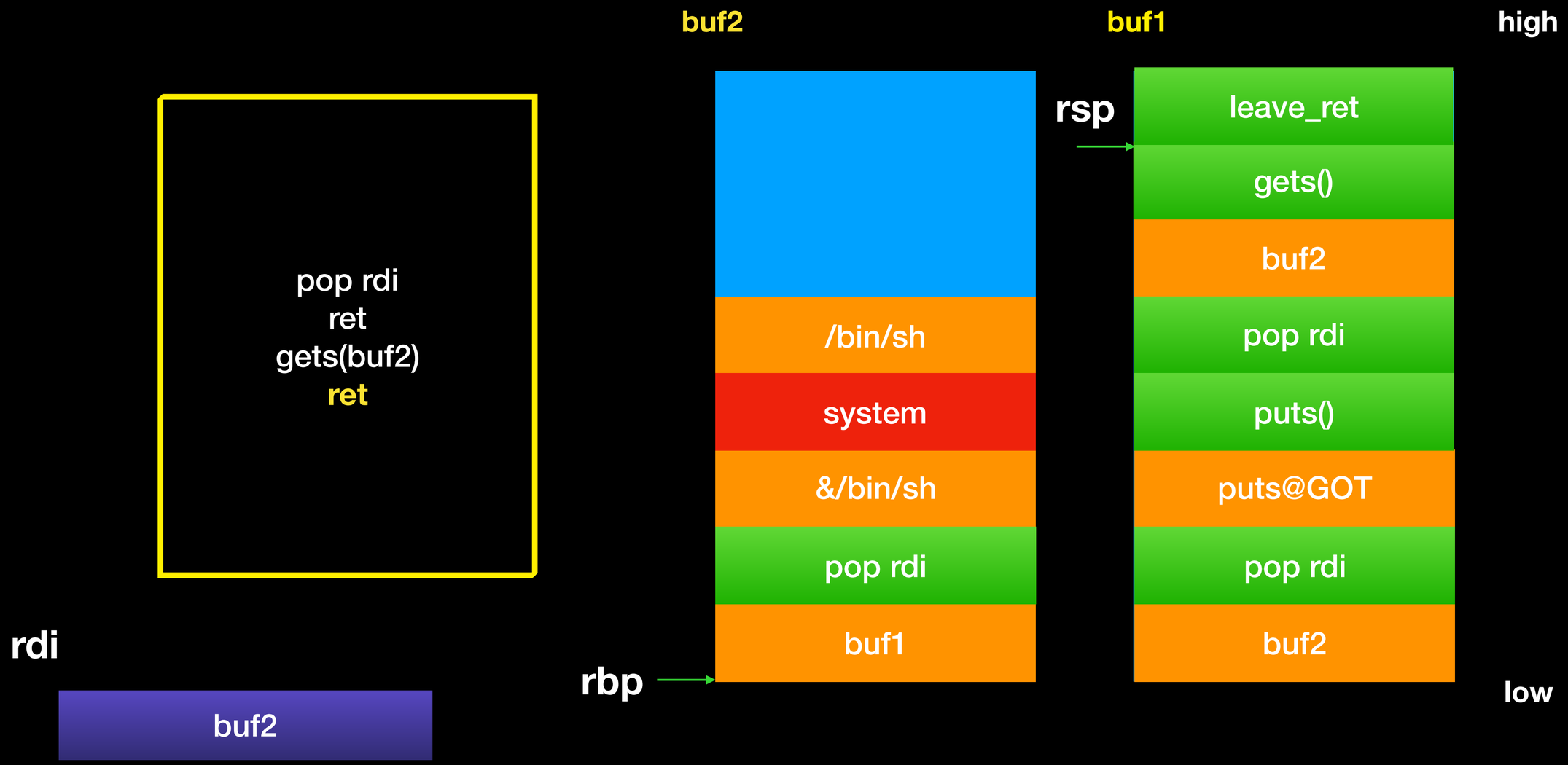
Stack Migration



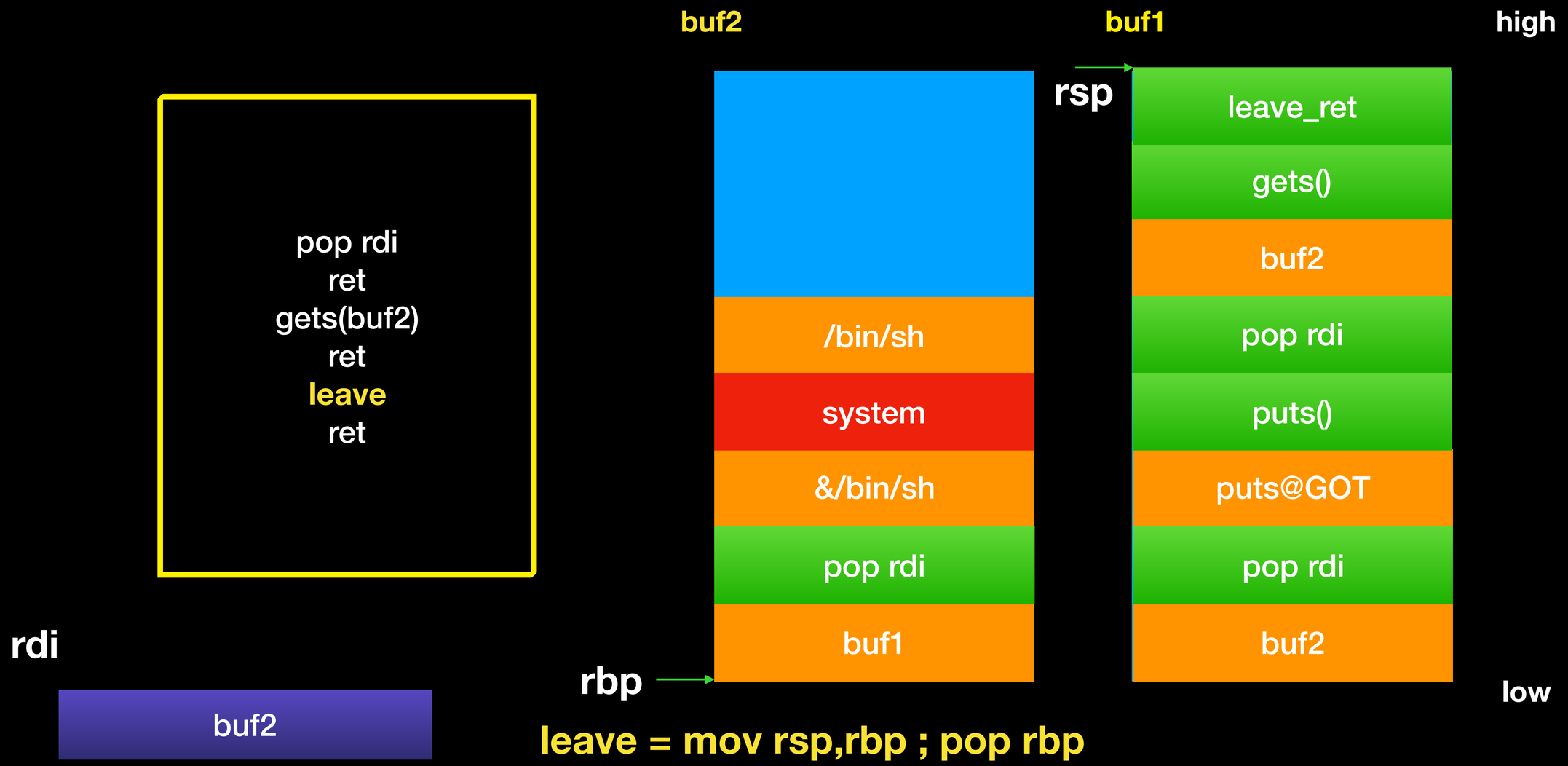
Stack Migration



Stack Migration

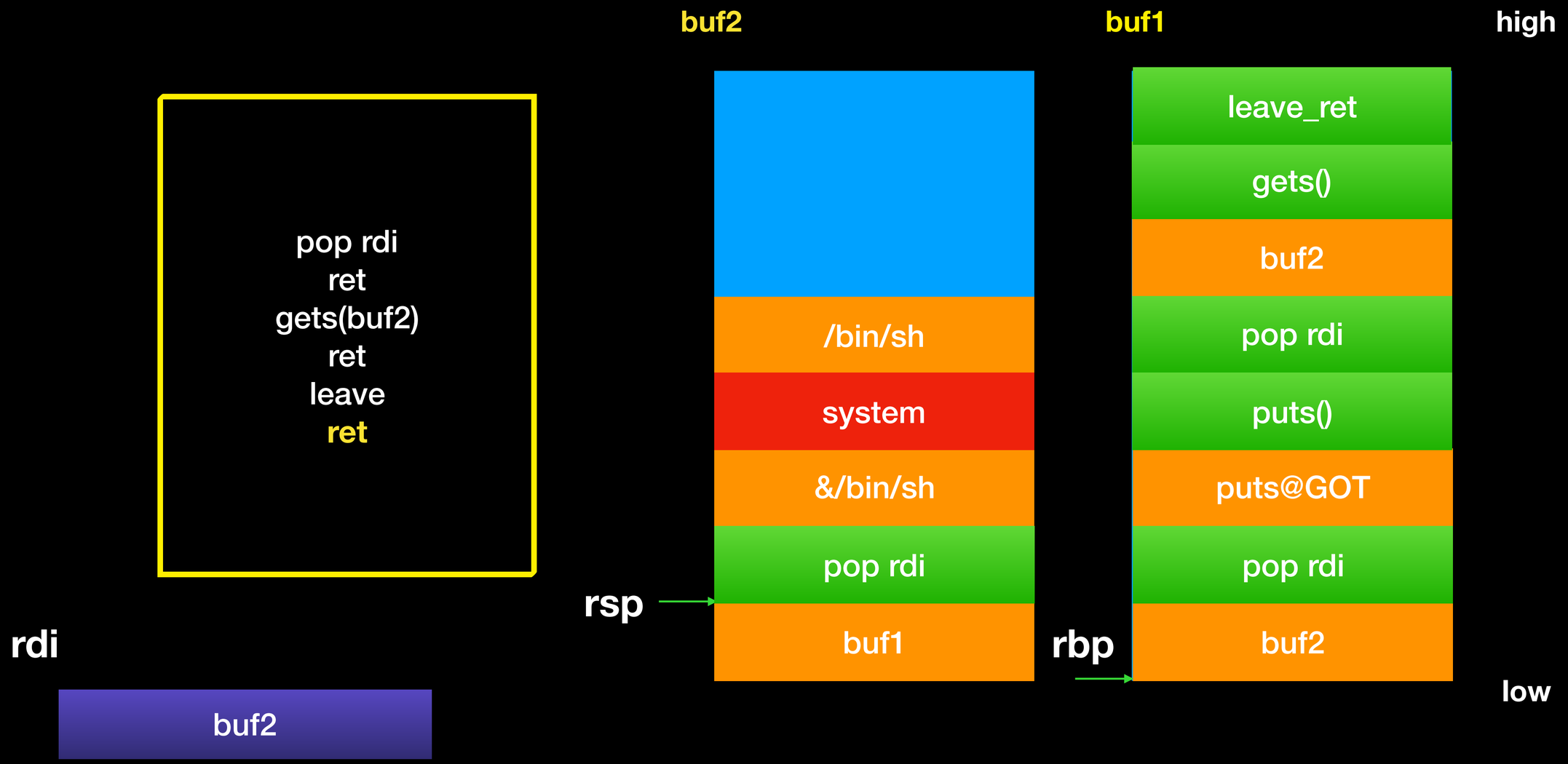


Stack Migration

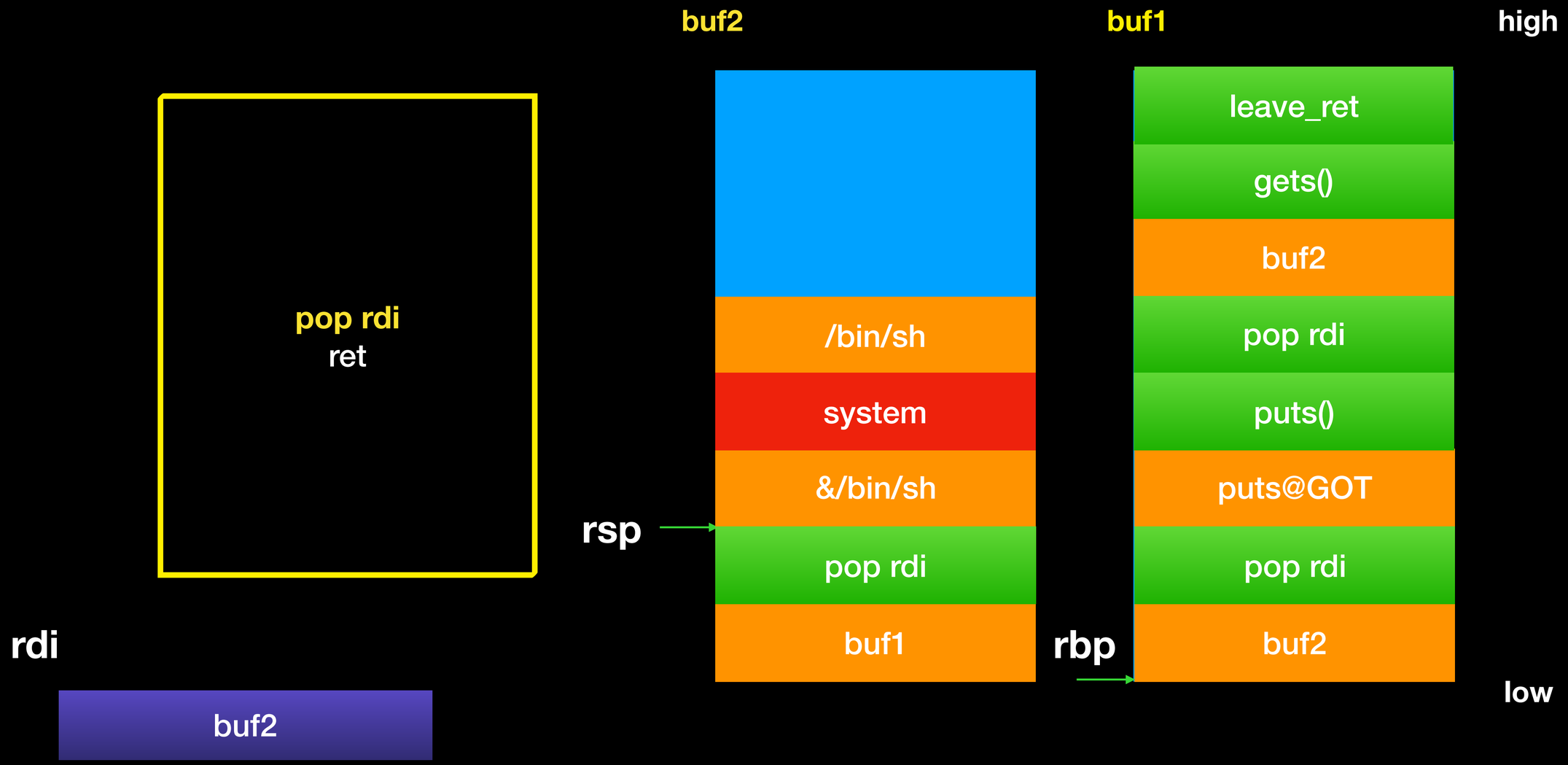


leave = mov rsp,rbp ; pop rbp

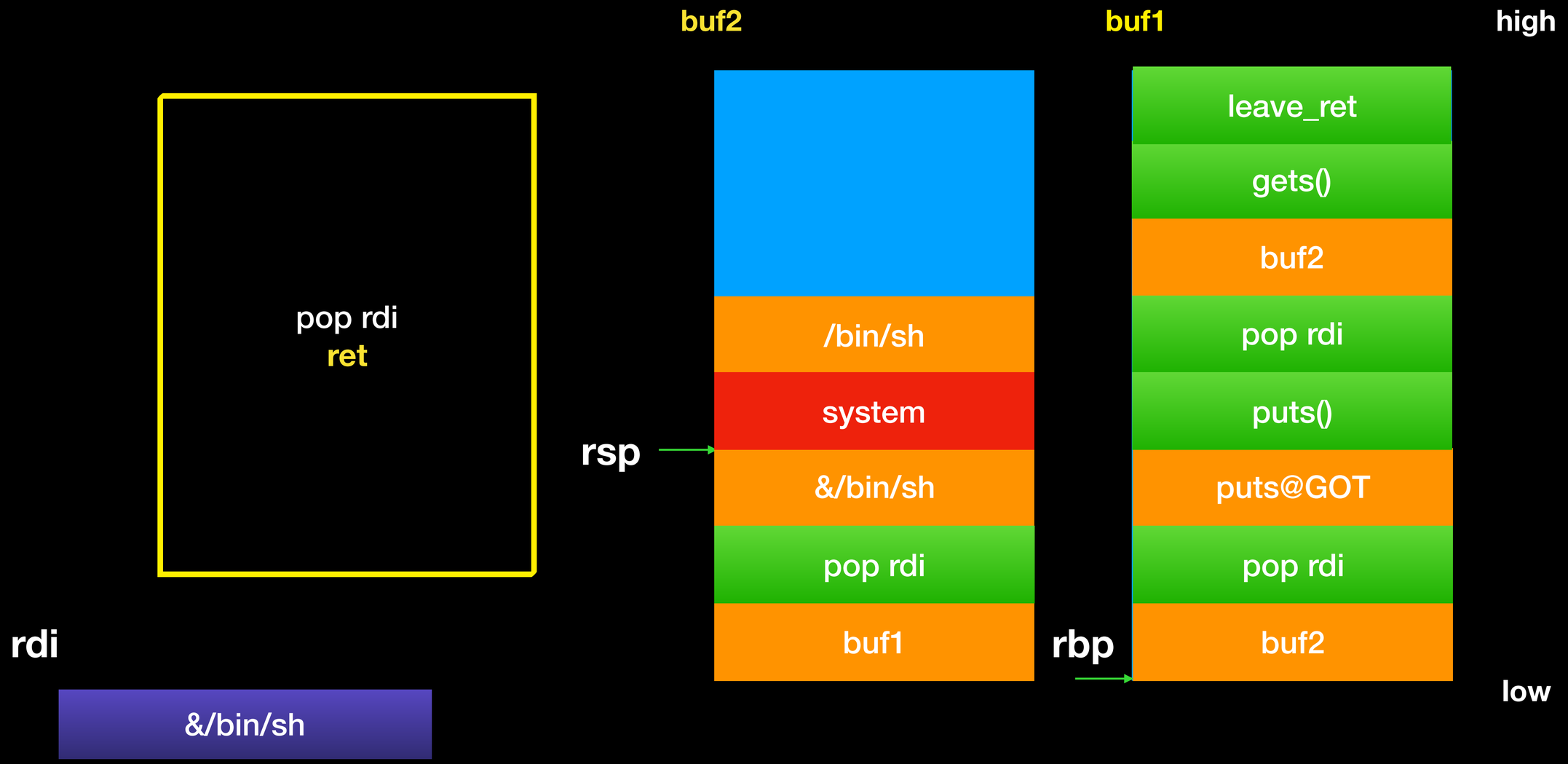
Stack Migration



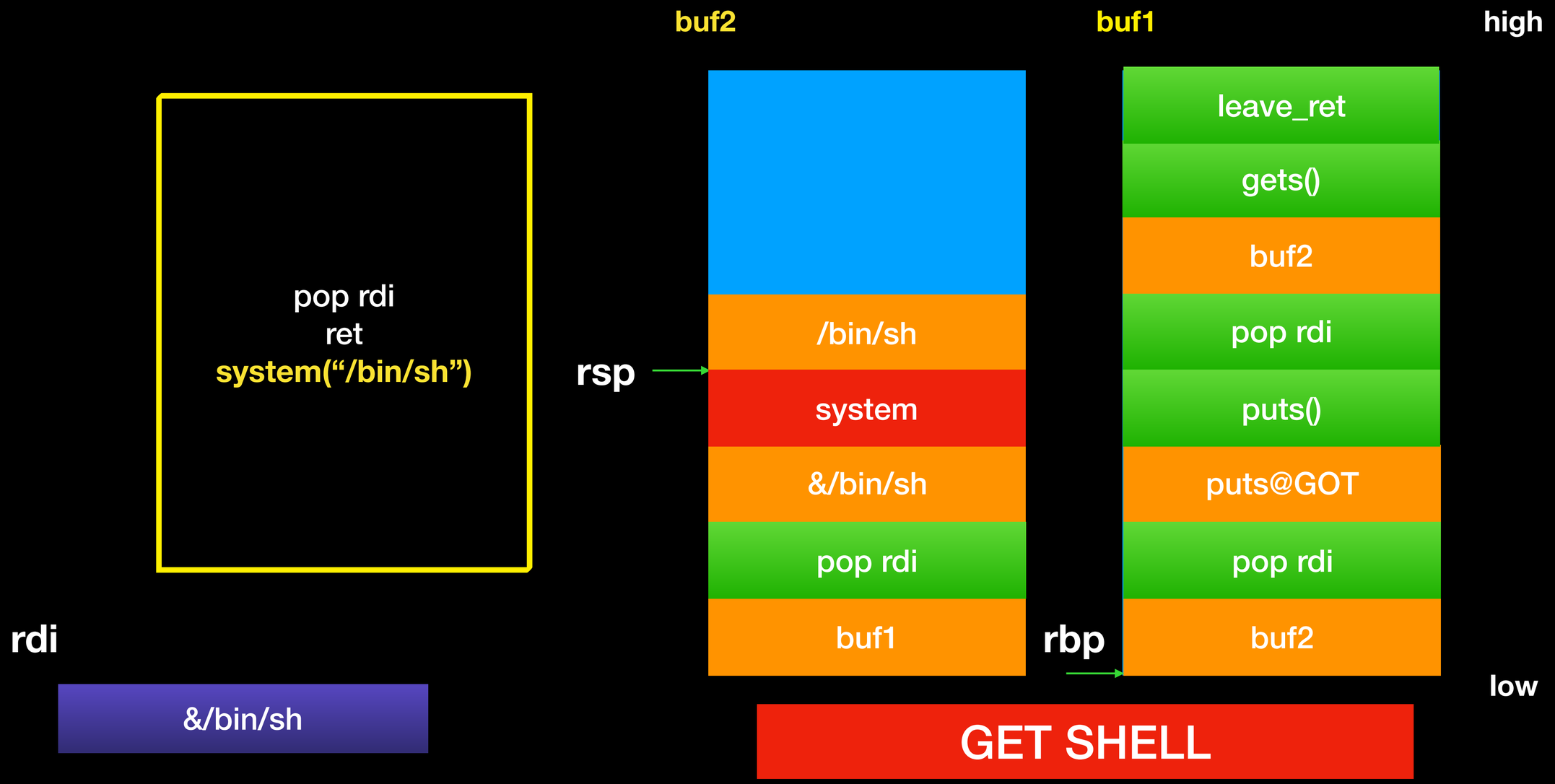
Stack Migration



Stack Migration



Stack Migration



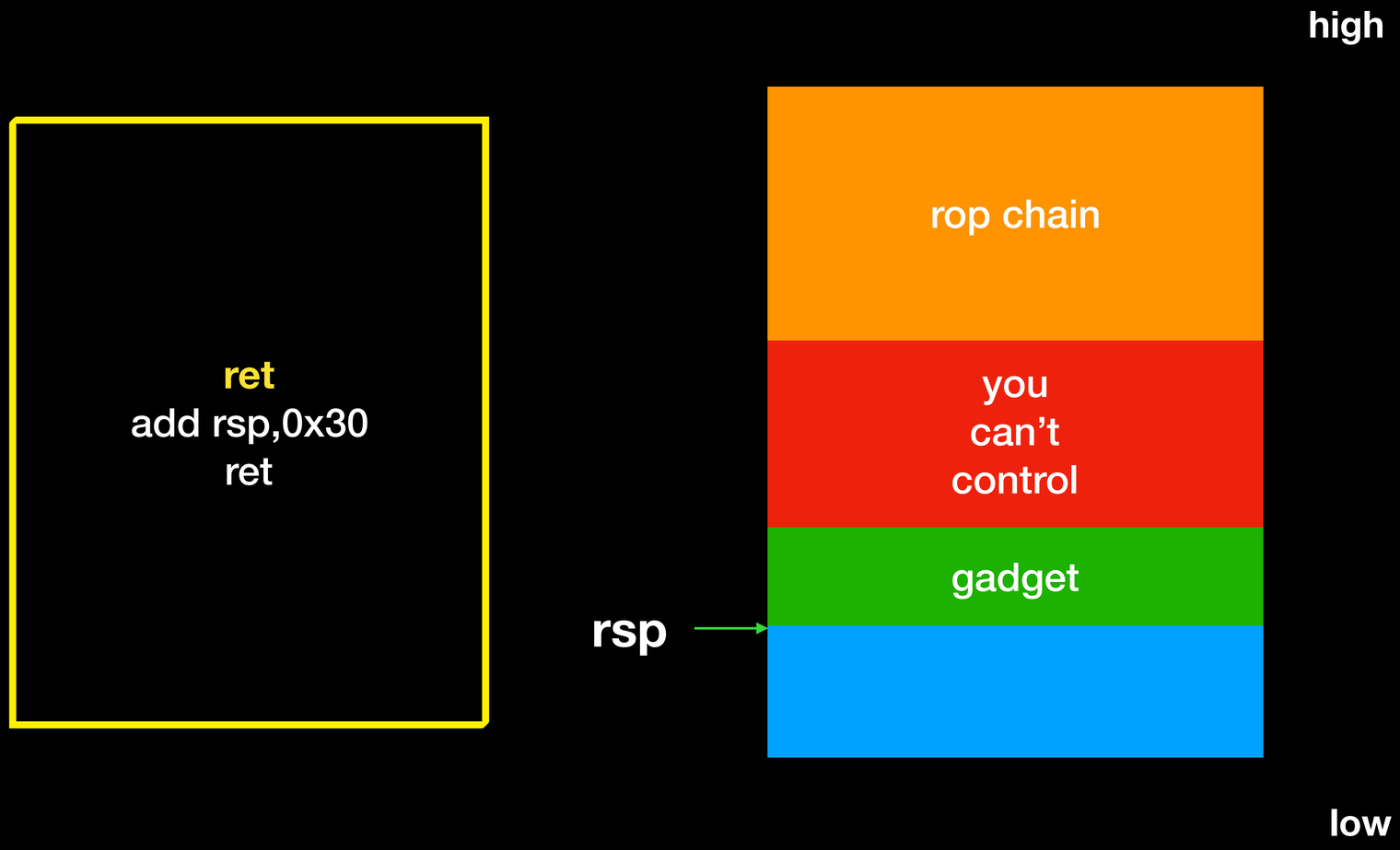
Stack Migration

- 若能妥善利用，在沒有 libc 的情況下，有機會將整個 libc 給 dump 出來，更有機會直接找出 system 的位置
- 無限 ROP ，幾乎可以做出所有事情，但唯一要注意的是 buf 大小要控制好，盡量選 bss 後半段位置，否則可能因為 stack 不夠大而 segfault

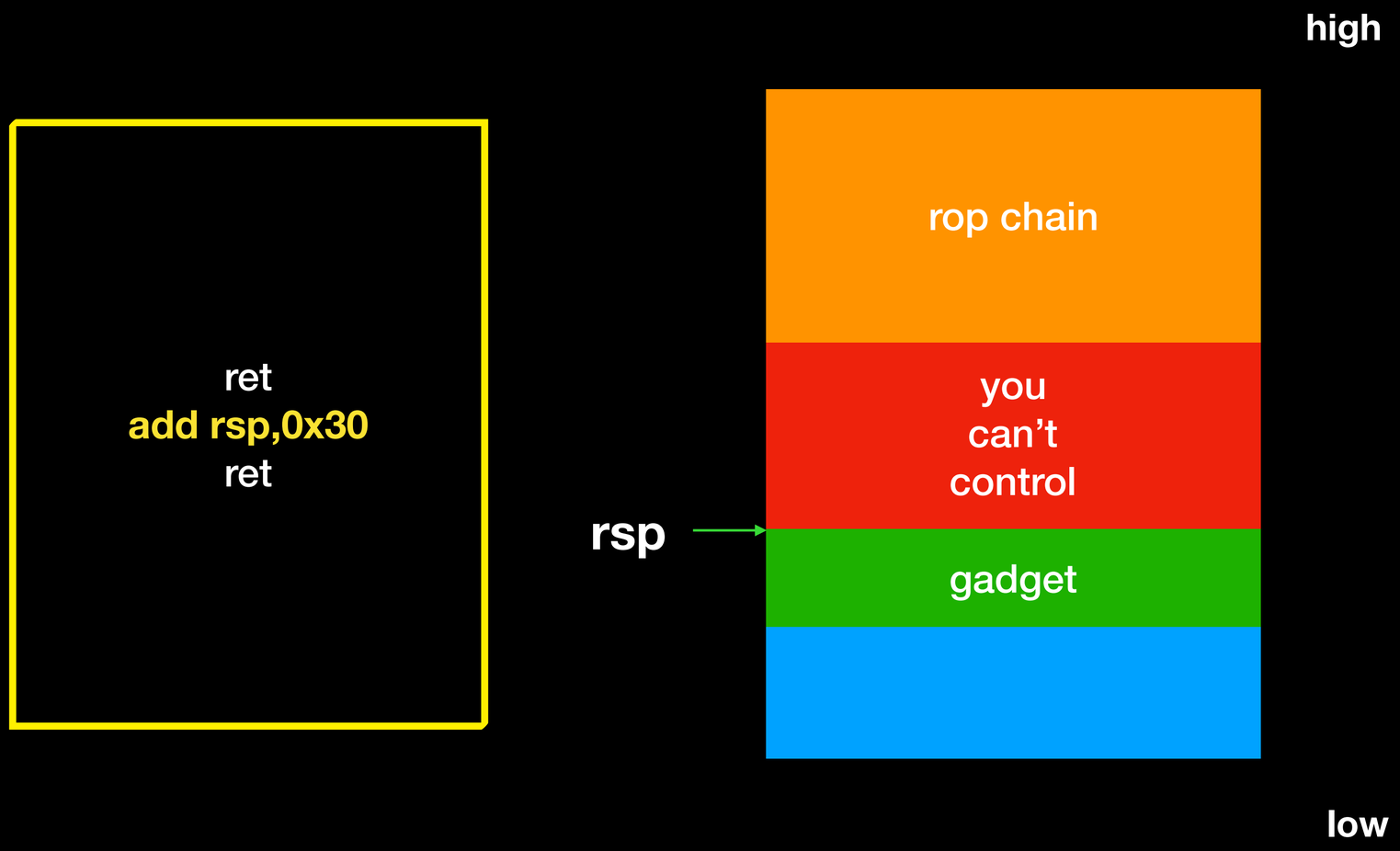
Stack Migration

- Other migration gadget
  - add rsp,0xNN ; ret
  - sub rsp,0xNN ; ret
  - ret 0xNN
  - xchg rsp,exx ; ret
  - partial overwrite rbp

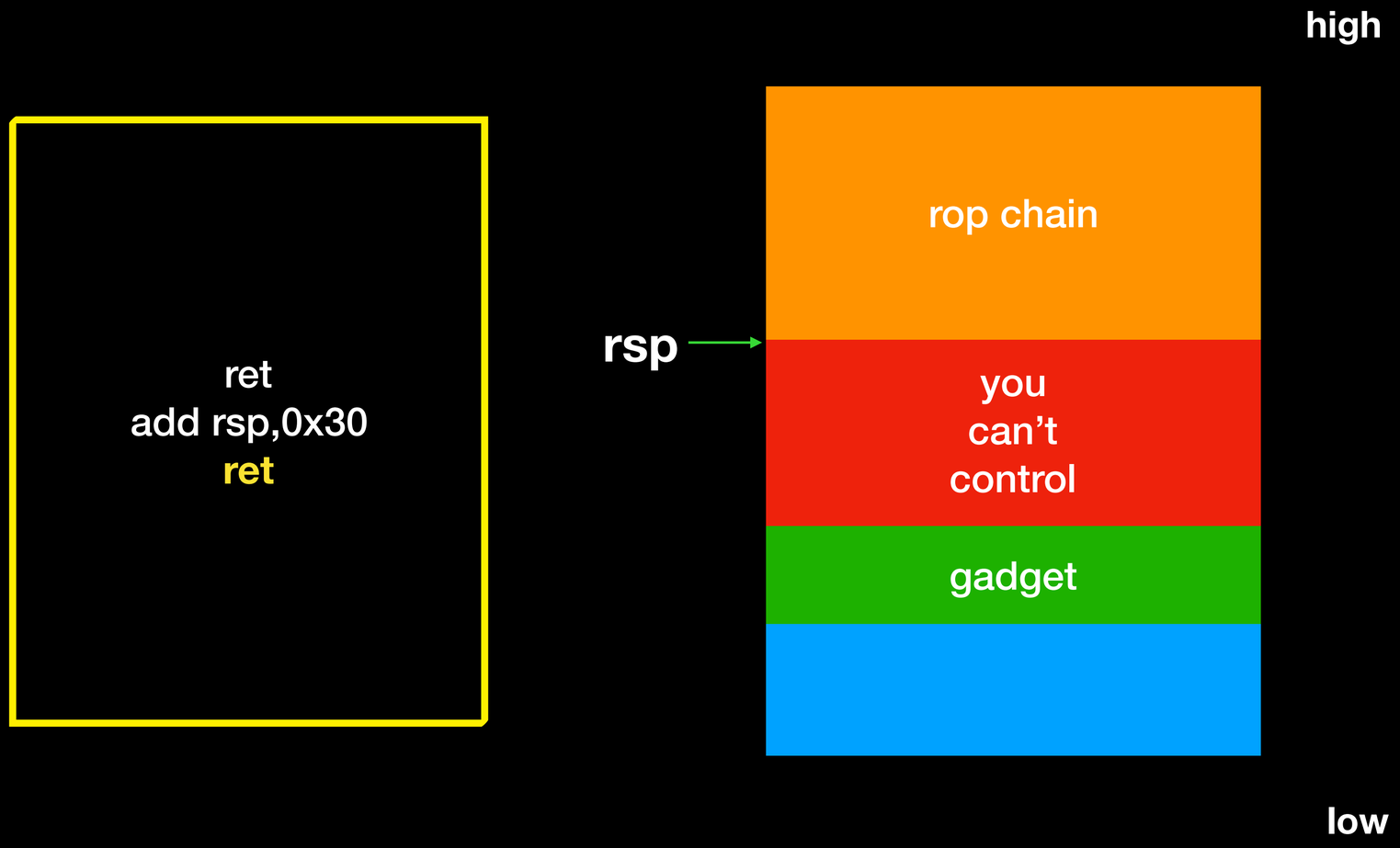
Stack Migration



Stack Migration



Stack Migration



Reference

- http://www.slideshare.net/hackstuff/rop-40525248

Q & A