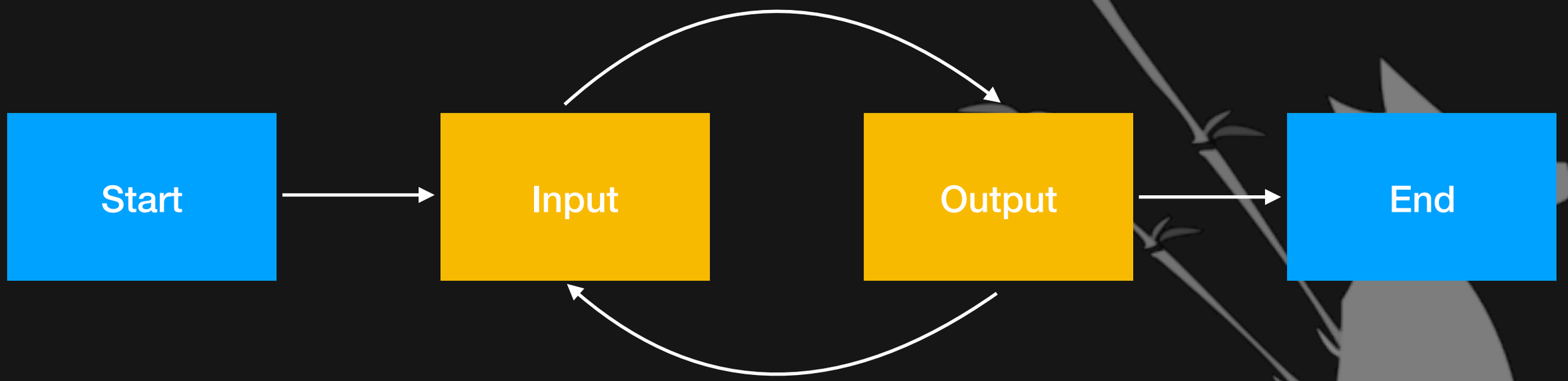
Pwn Introduction

frozenkp@BambooFox

What is Pwn ?

❖ 控制程式流程，進而觸發攻擊

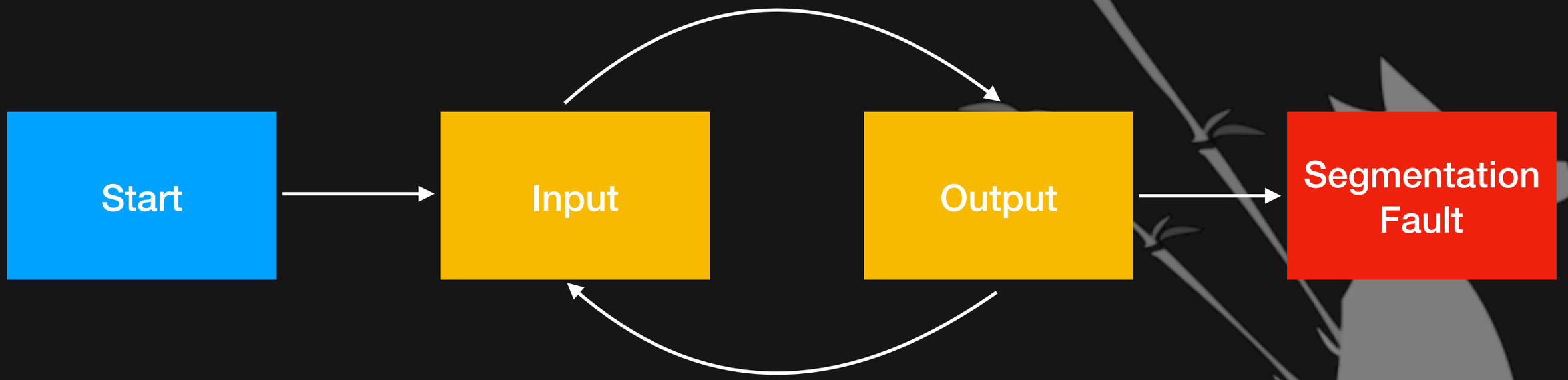


❖ Input: 使用者輸入、操作

❖ Output: 反應 Input 所產生的動作，包含運算、輸出

How to Pwn ?

❖ 尋找漏洞，並進一步利用

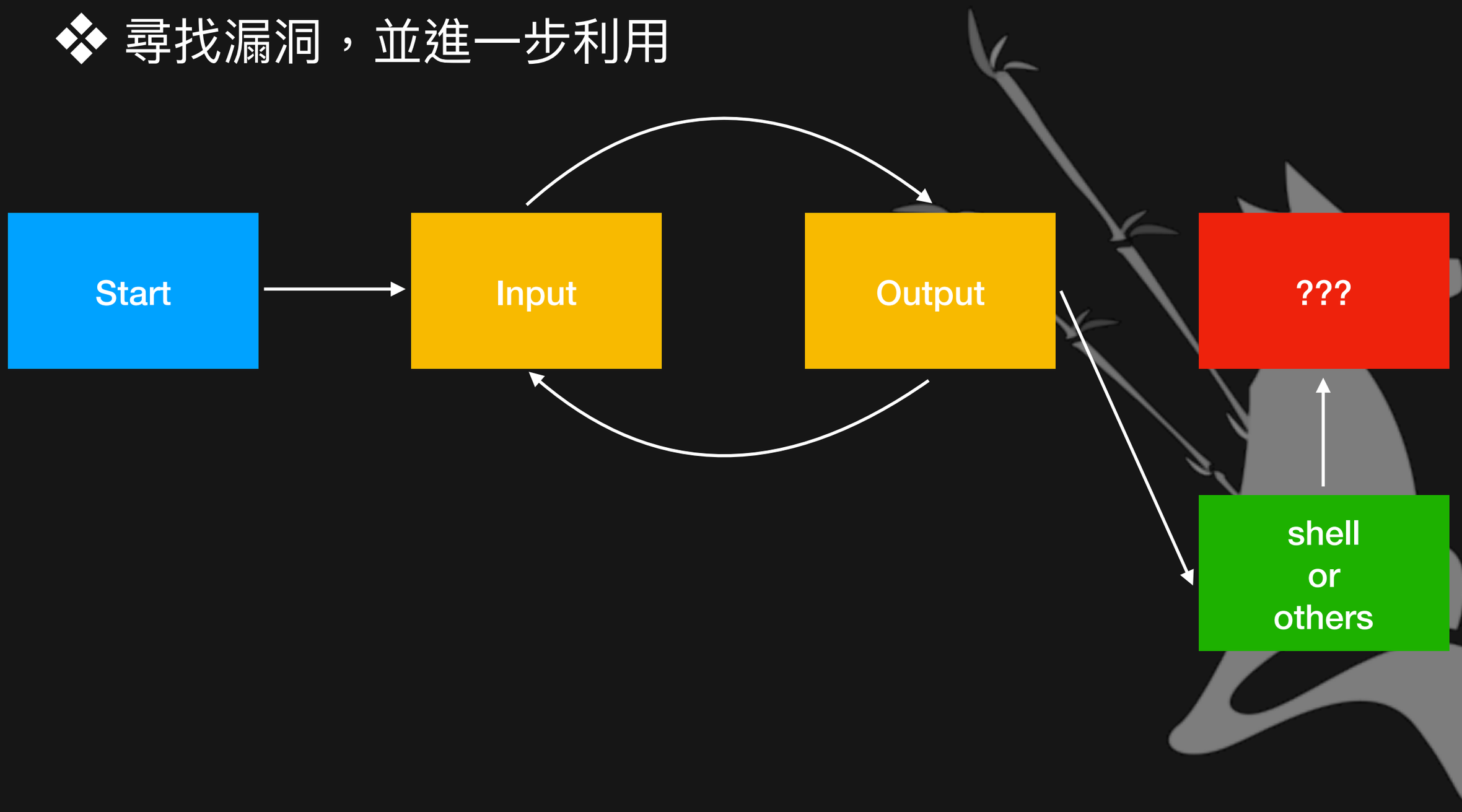


❖ Fuzz 模糊測試: 自動產生隨機輸入來尋找漏洞

❖ 看原始碼、組合語言找漏洞

How to Pwn ?

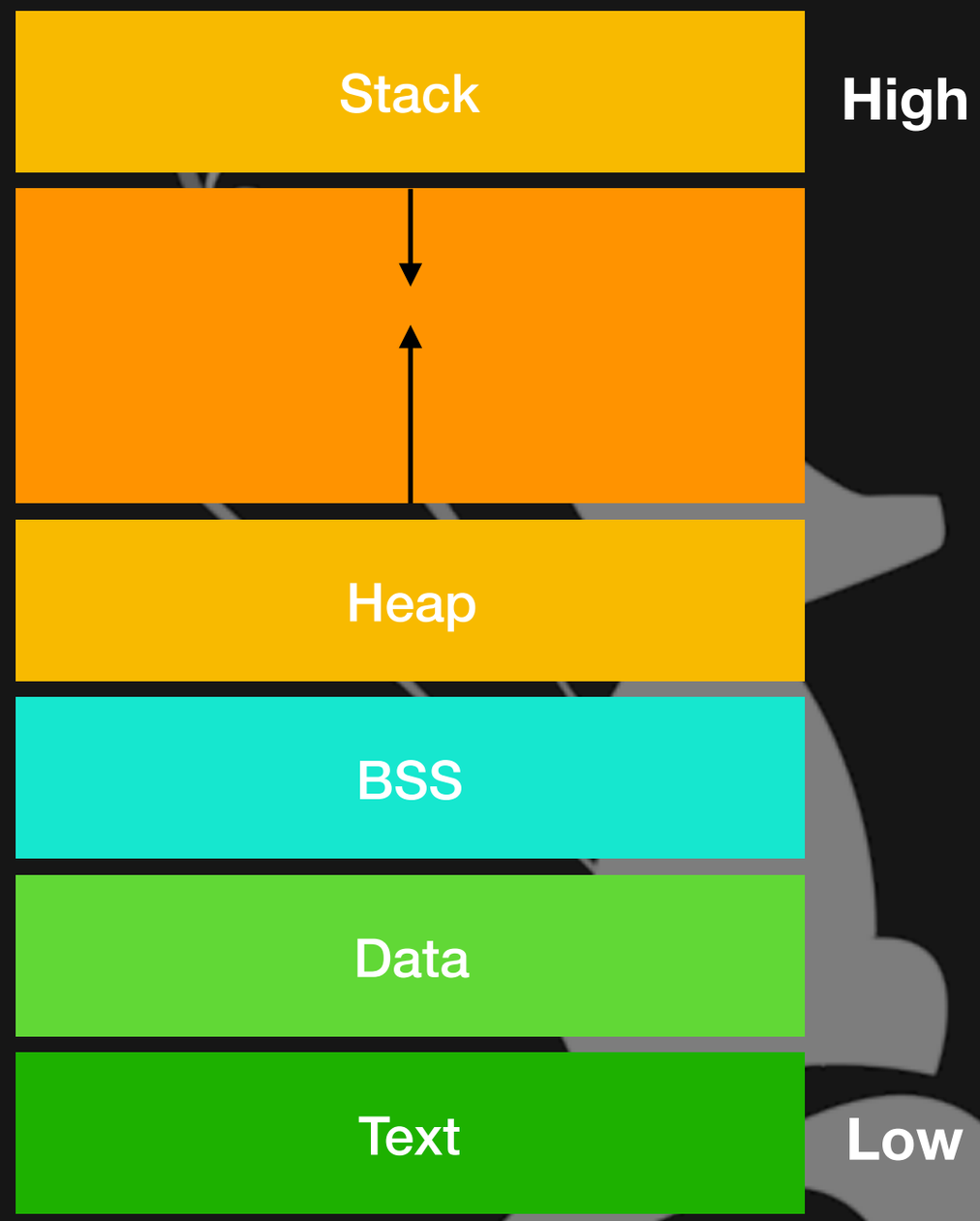
❖ 尋找漏洞，並進一步利用



Program Structure

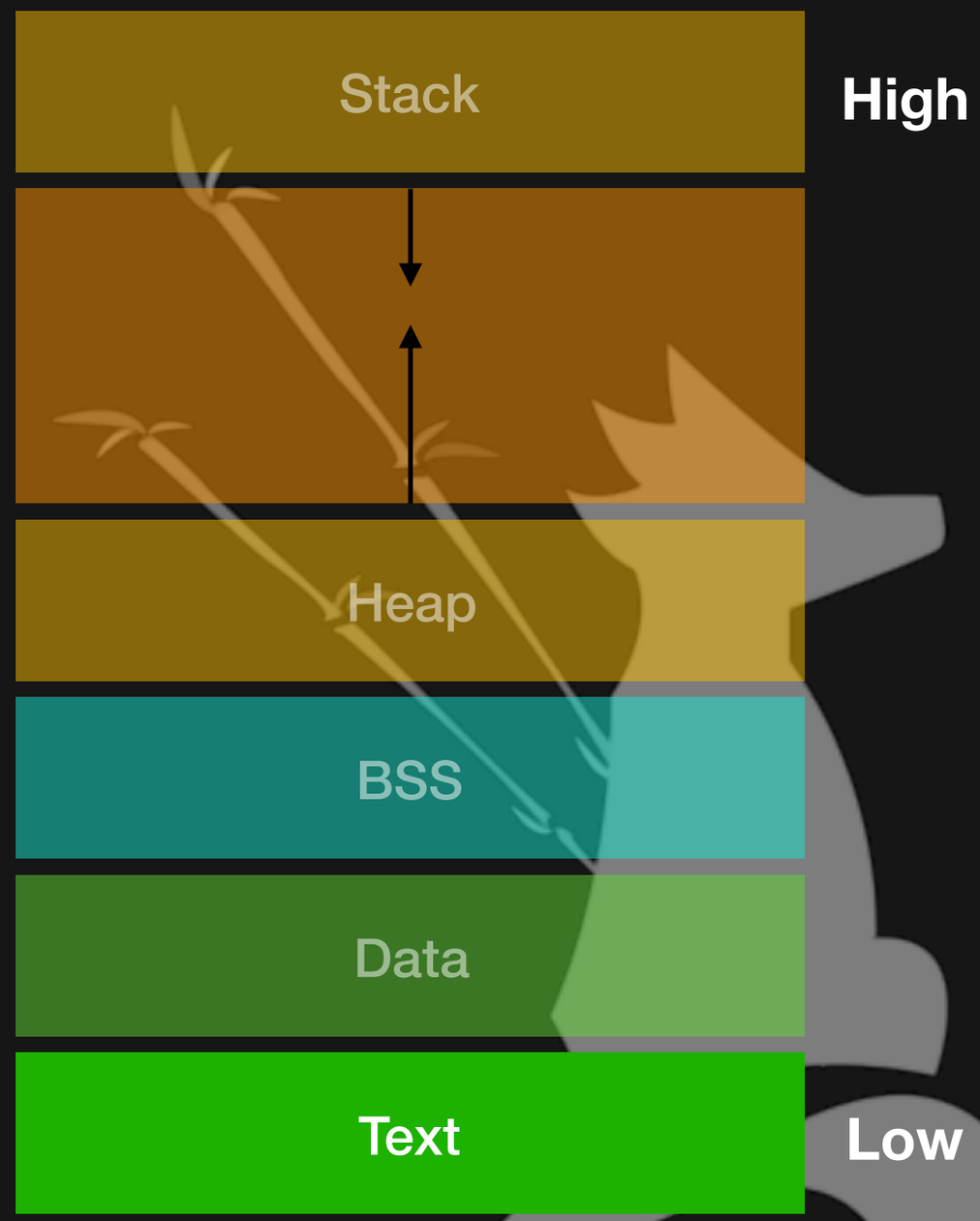
Overview

- Text
- Data
- BSS
- Heap
- Stack



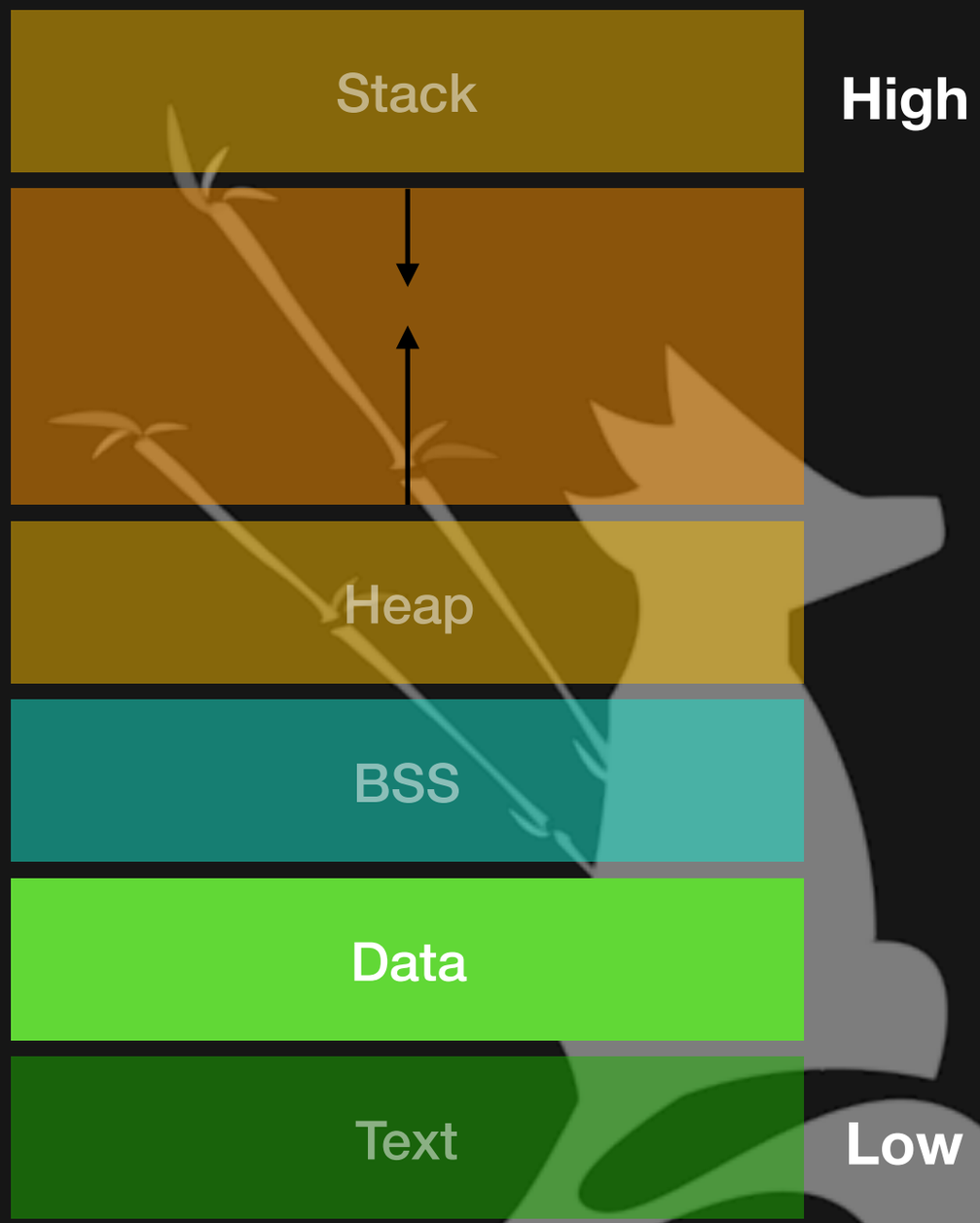
Text

- 程式碼 (binary)
- 可讀 不可寫 可執行 (r-x)



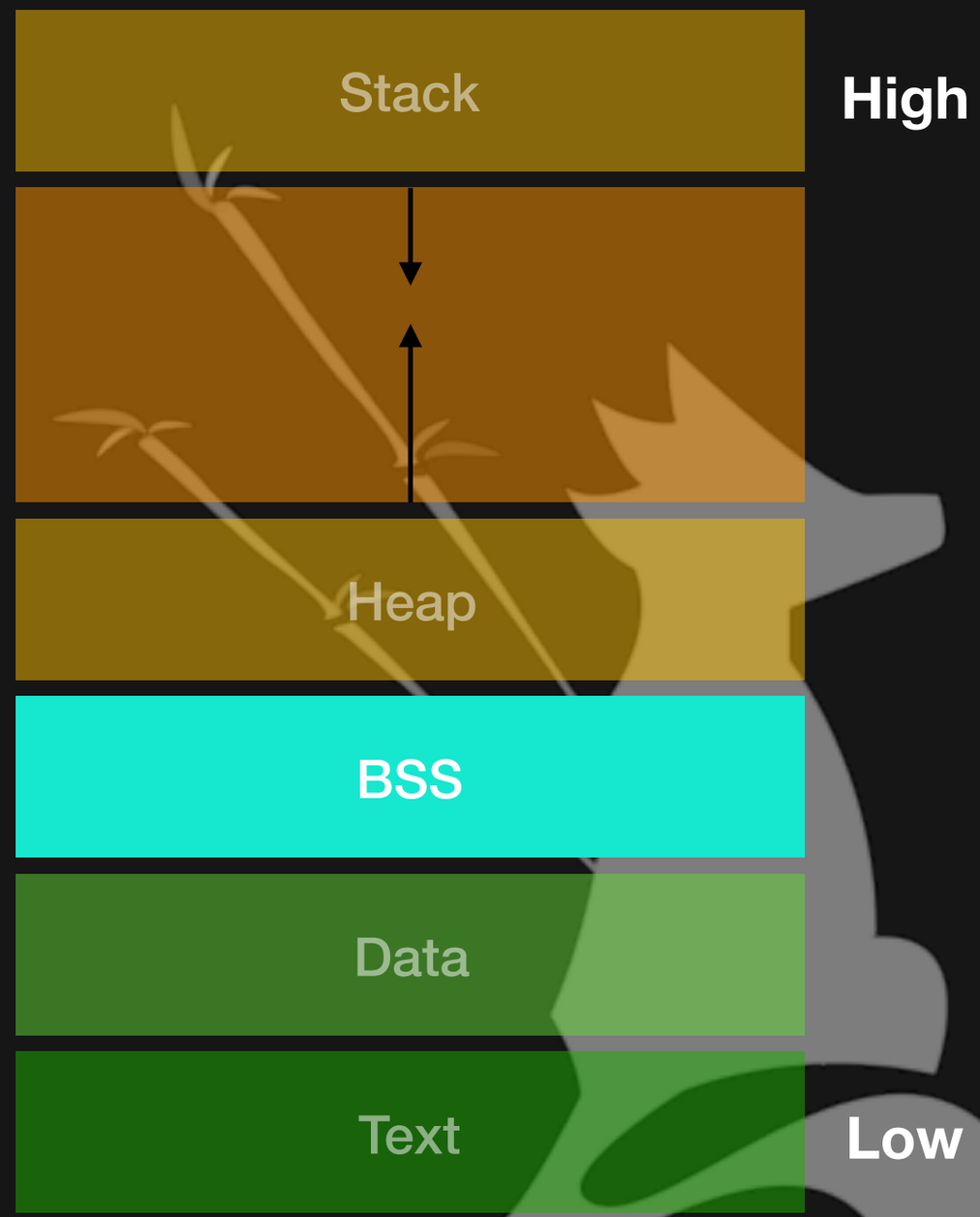
Data

- 已初始化的全域變數



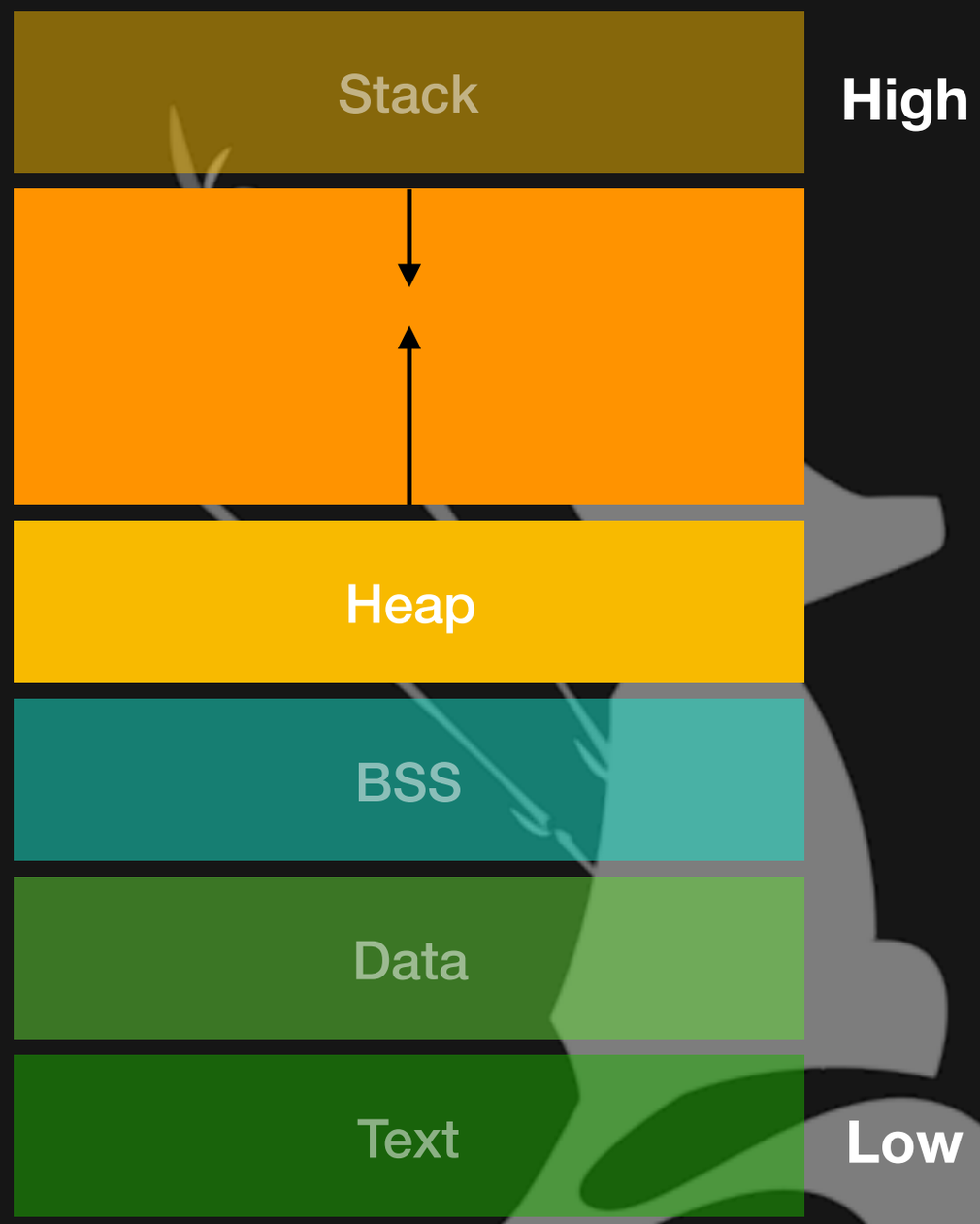
BSS

❖ 未初始化的全域變數



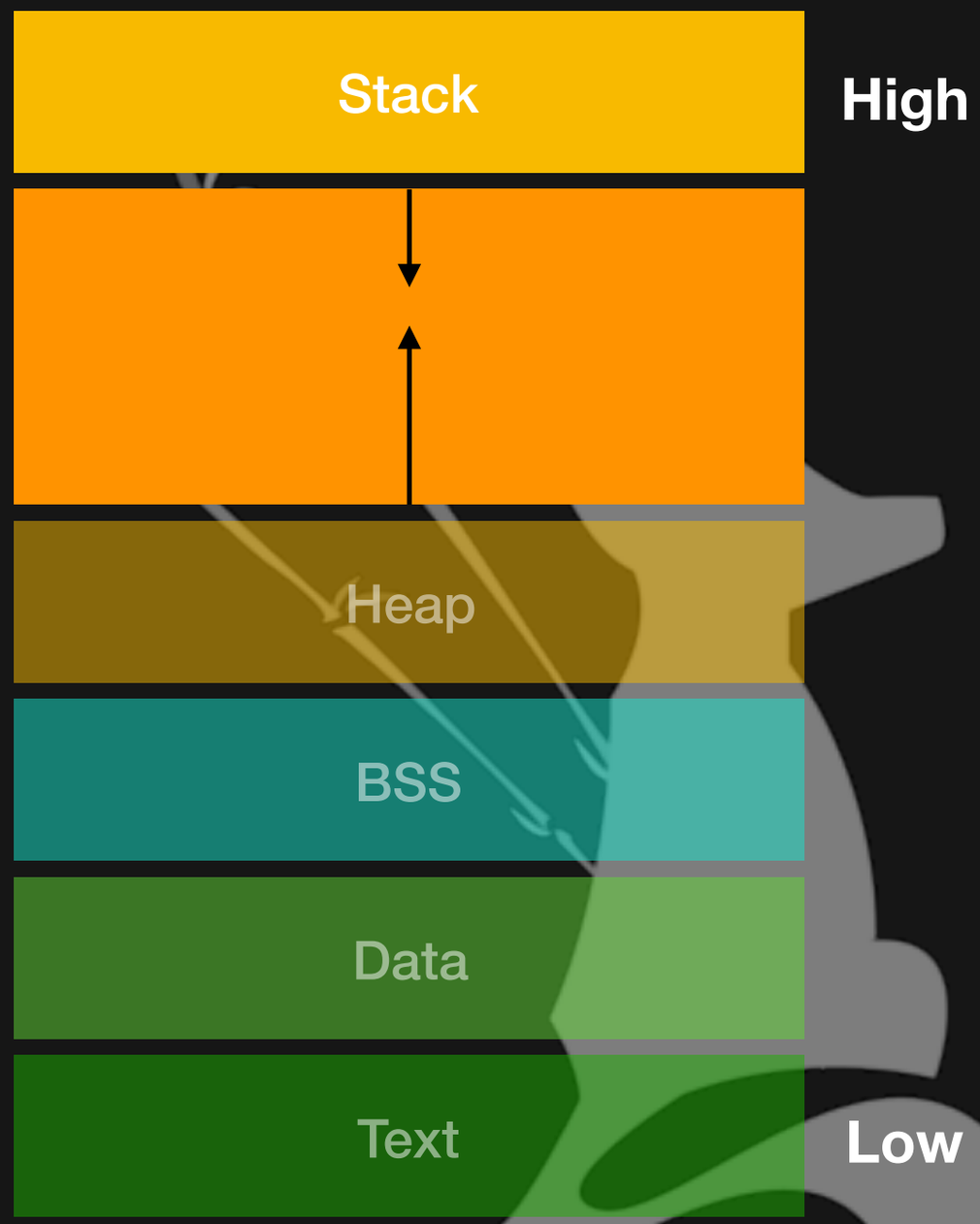
Heap

- 動態記憶體空間
- malloc( ) / free( )
- 由低位往高位長



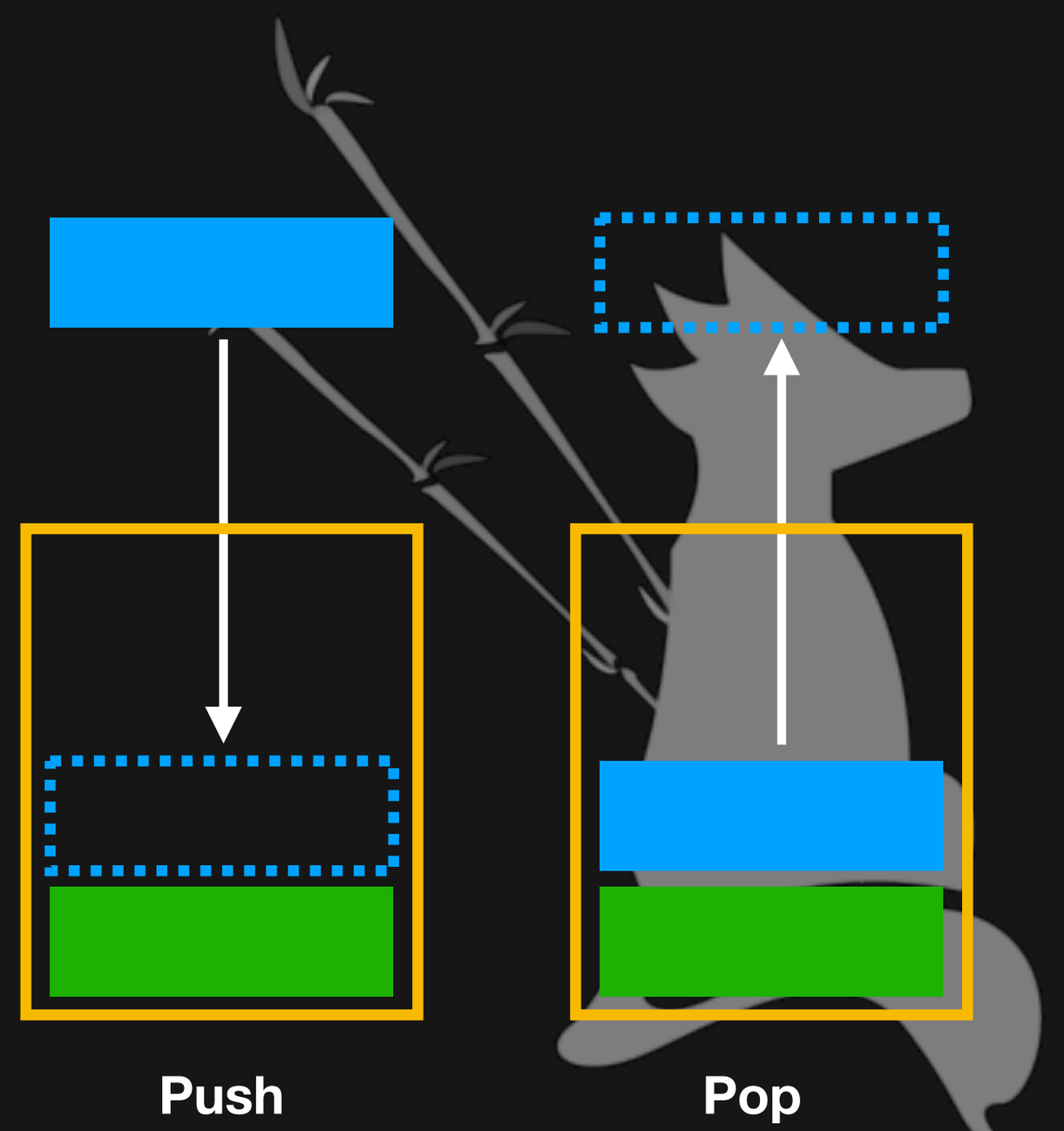
Stack

- ❖ 存放暫存資料
  - ▸ 區域變數
  - ▸ return address
  - ▸ 參數
  - ▸ 回傳值
- ❖ 由高位往低位長
- ❖ stack top 存在 rsp



Stack - Data Structure

- 先進後出 (first in last out)
- 進 (push)
- 出 (pop)

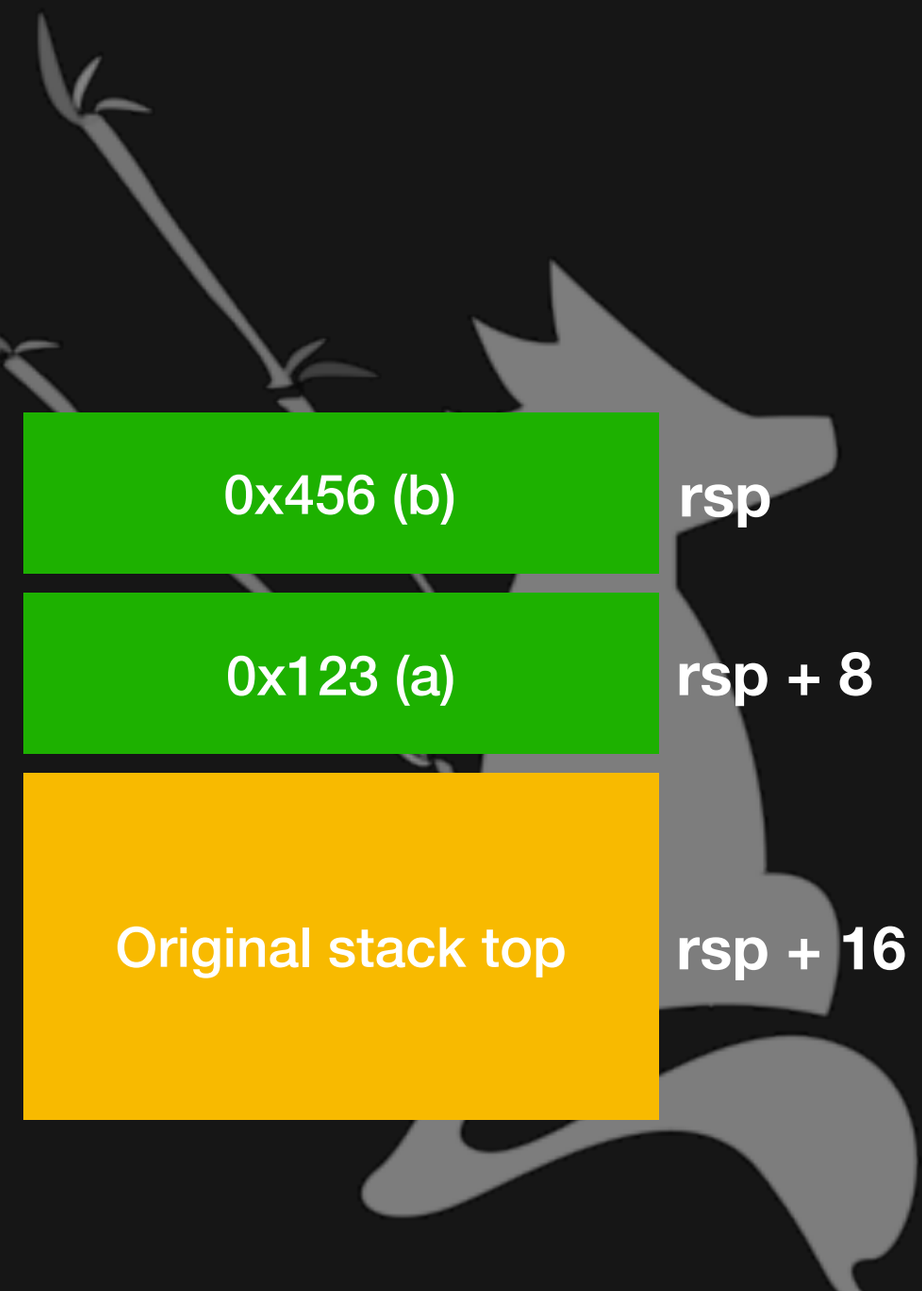


Stack - Local variable

❖ 區域變數存在 stack 上

❖ 先宣告的先存

```c
1 int main(){
2     int a = 0x123;
3     int b = 0x456;
4     …
5     return 0;
6 }
```



Stack - Return address

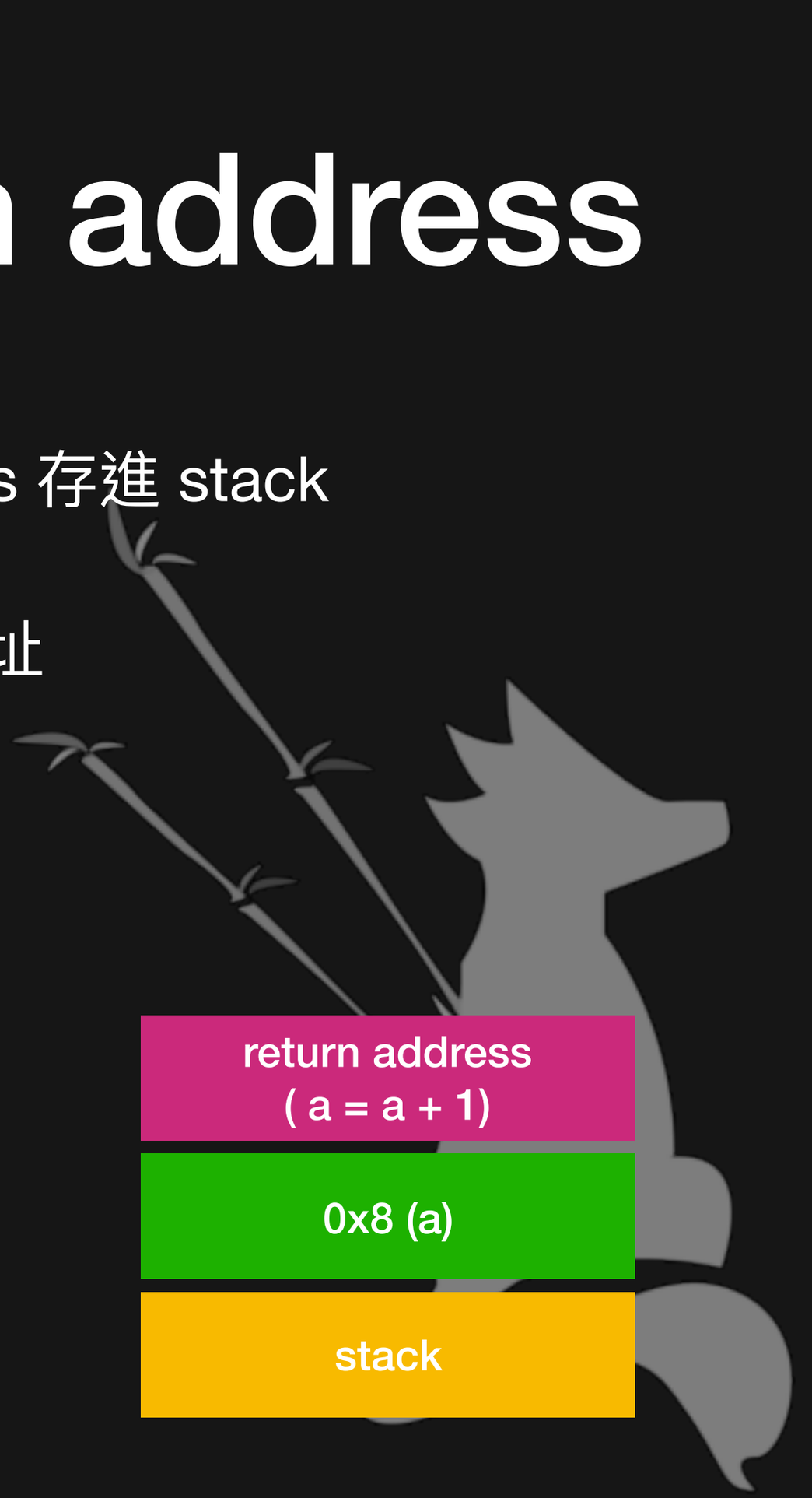
- call function 前，將 return address 存進 stack
- return 時，回到 stack 中所存的位址

```
1  void process(){
2      …
3      return;
4  }
5
6  int main(){
7      int a = 8;
8      process();
9      a = a + 1;
10     …
11     return 0;
12 }
```

Stack - Return address

- call function 前，將 return address 存進 stack
- return 時，回到 stack 中所存的位址

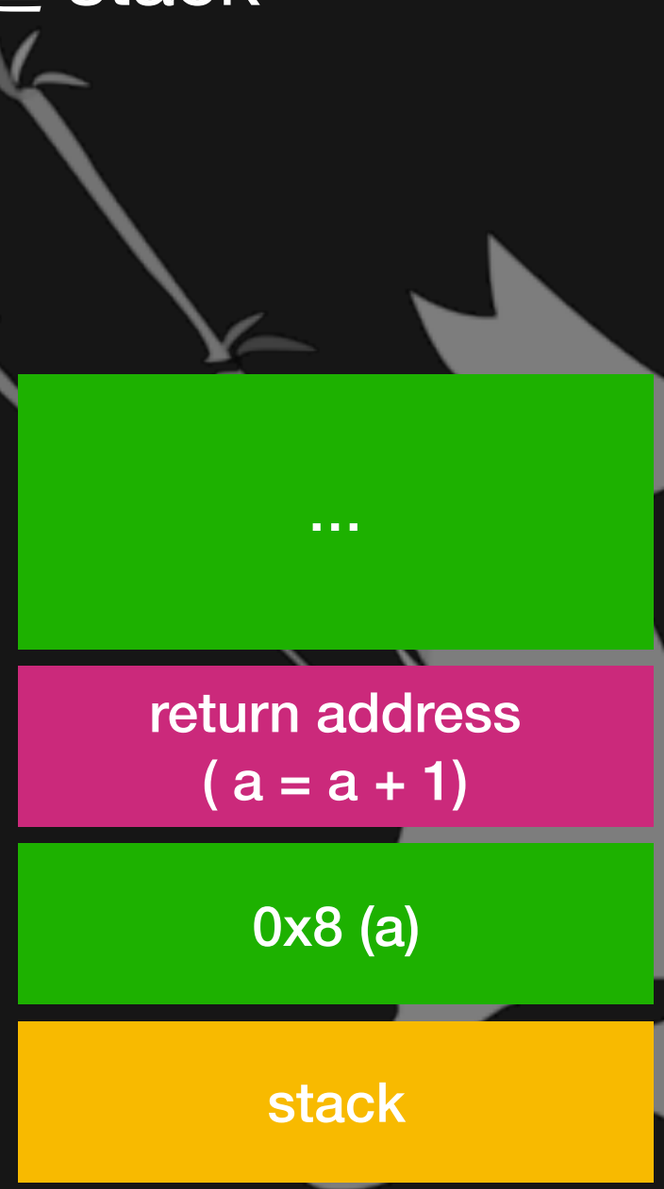
```
1  void process(){
2       …
3       return;
4  }
5
6  int main(){
7       int a = 8;
8       process();
9       a = a + 1;
10      …
11      return 0;
12 }
```



Stack - Return address

- call function 前，將 return address 存進 stack
- return 時，回到 stack 中所存的位址

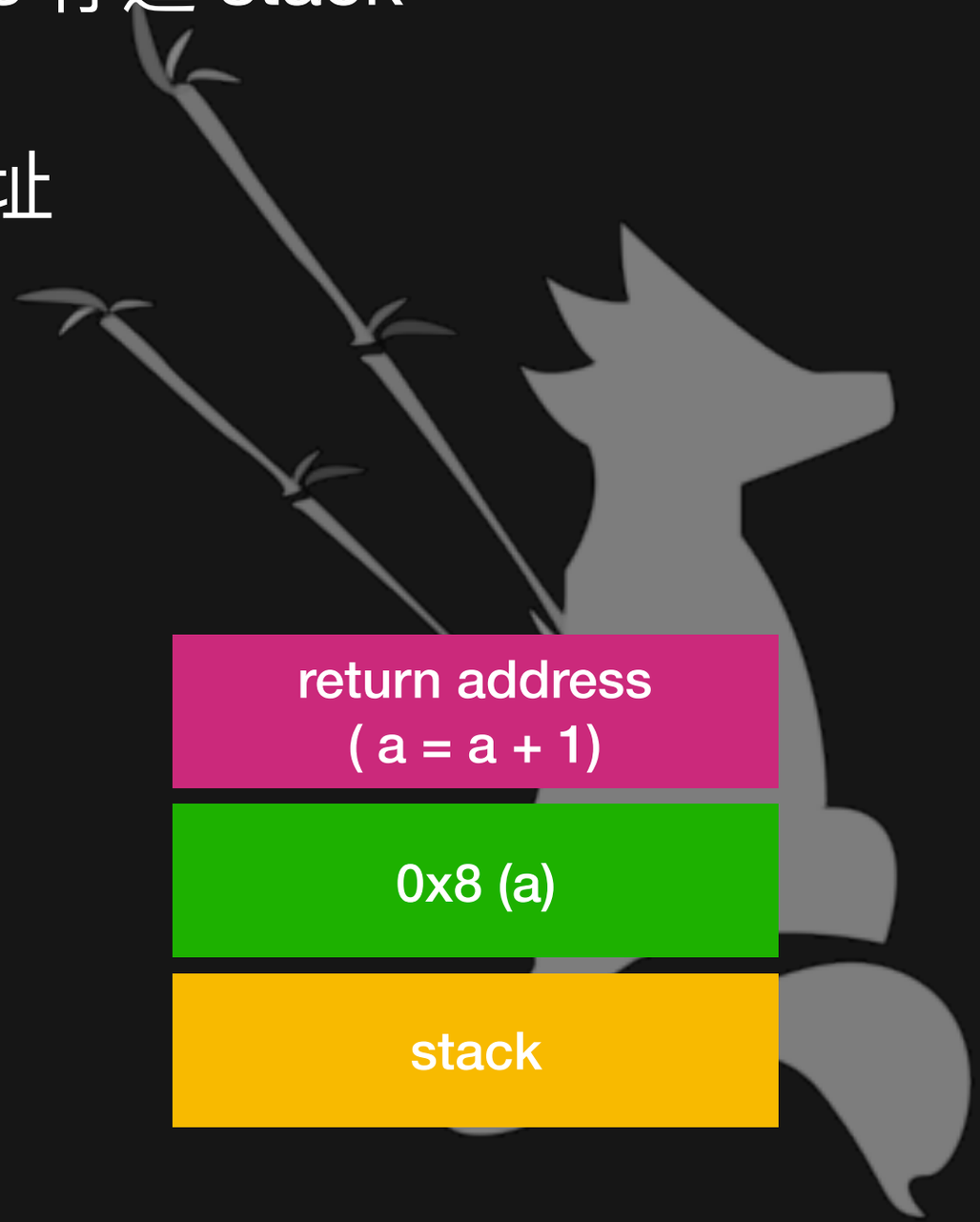
```
1  void process(){
2      …
3      return;
4  }
5
6  int main(){
7      int a = 8;
8      process();
9      a = a + 1;
10     …
11     return 0;
12 }
```



Stack - Return address

- call function 前，將 return address 存進 stack
- return 時，回到 stack 中所存的位址

```
1  void process(){
2       …
3       return;
4  }
5
6  int main(){
7       int a = 8;
8       process();
9       a = a + 1;
10      …
11      return 0;
12 }
```



Stack - Return address

- call function 前，將 return address 存進 stack
- return 時，回到 stack 中所存的位址

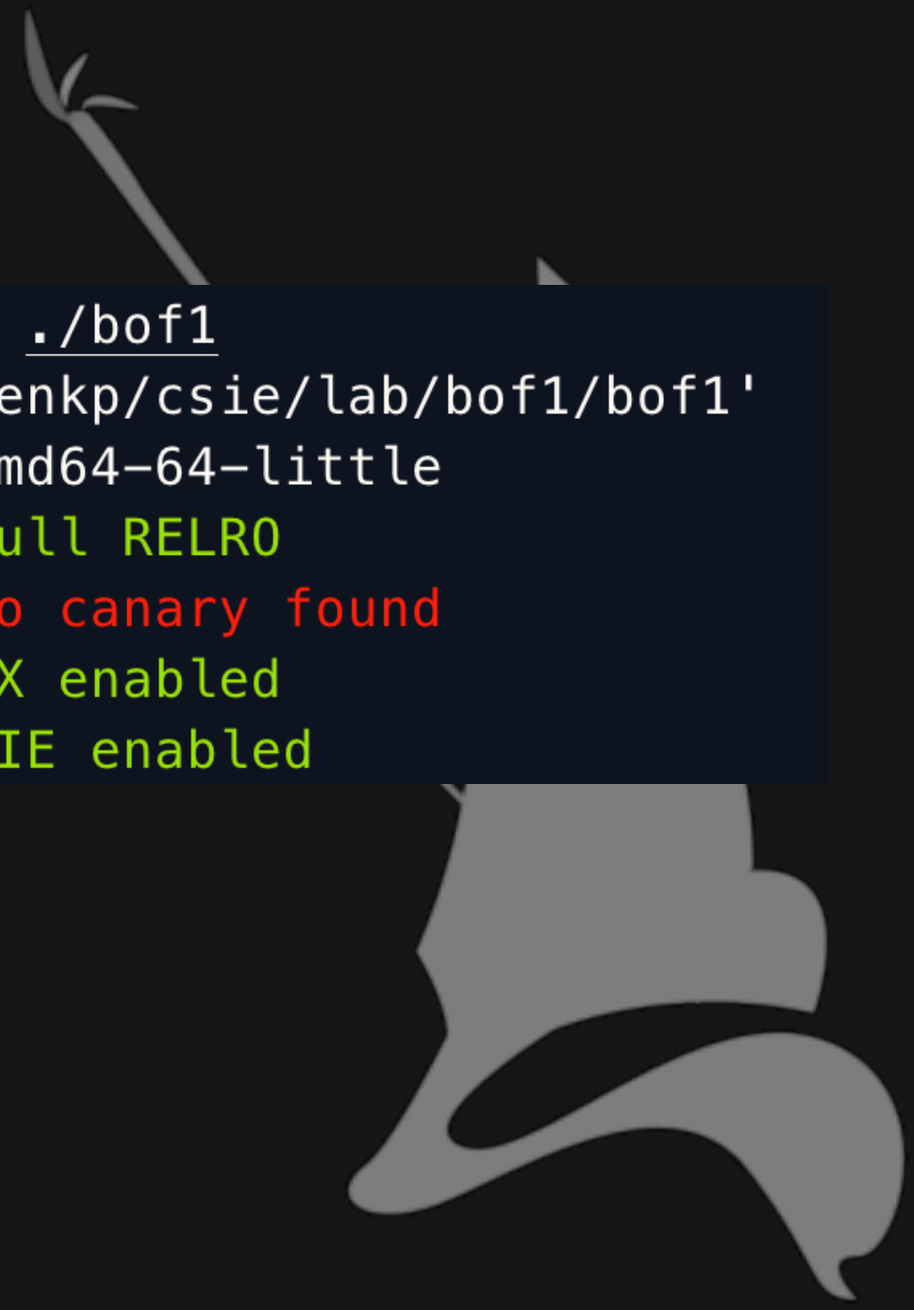
```
1  void process(){
2      …
3      return;
4  }
5
6  int main(){
7      int a = 8;
8      process();
9      a = a + 1;
10     …
11     return 0;
12 }
```

Security Options

Overview

- RELRO
- Stack Canary
- NX
- PIE
- ASLR

```
[XD] % checksec ./bof1
[*] '/home/frozenkp/csie/lab/bof1/bof1'
    Arch:     amd64-64-little
    RELRO:    Full RELRO
    Stack:    No canary found
    NX:       NX enabled
    PIE:      PIE enabled
```



Lazy Binding

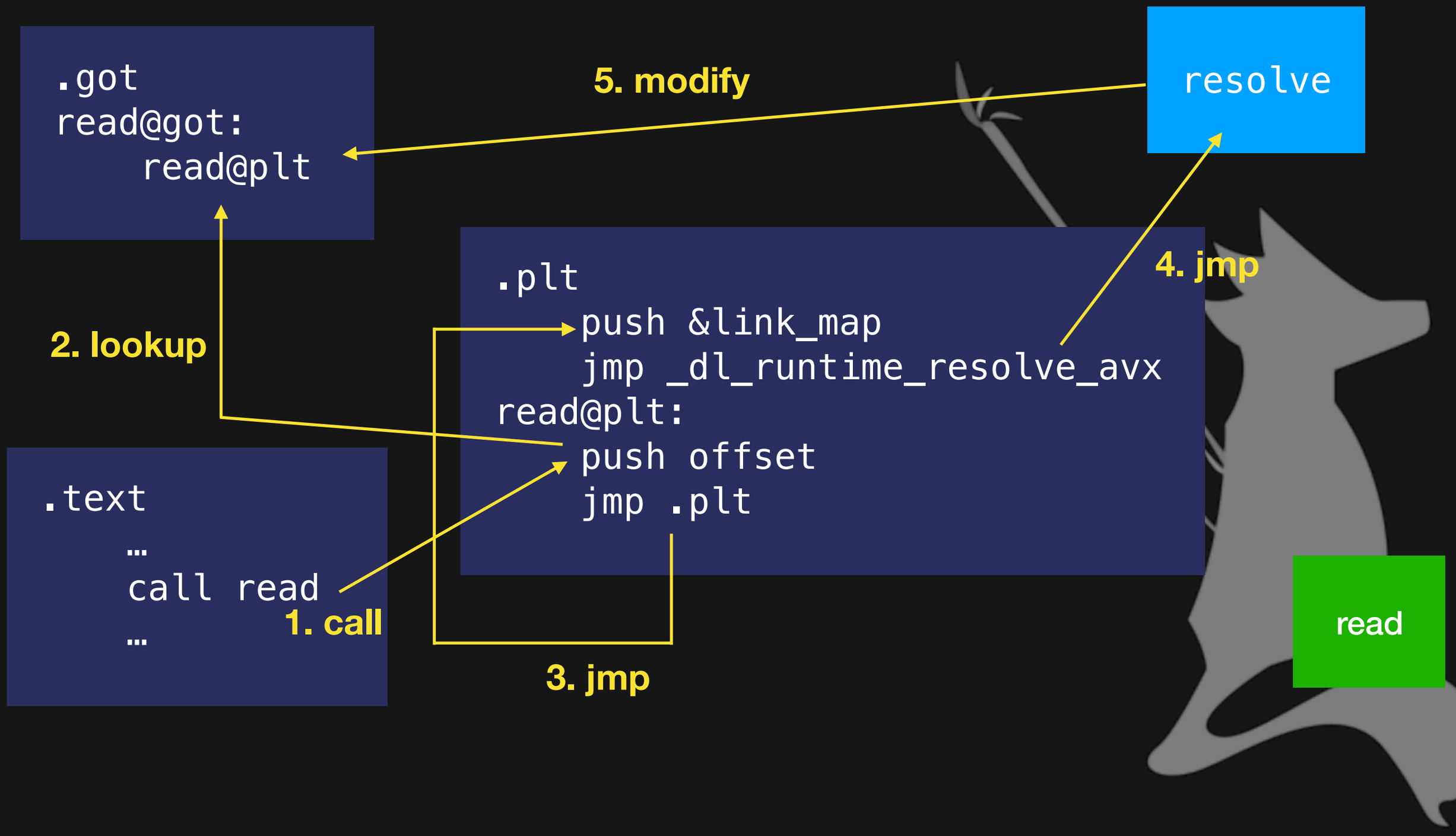
- Dynamic linking 的程式在執行過程中，有些 library 的函式可能到結束都不會執行到

- ELF 採取 Lazy binding 的機制，在第一次 call library 函式時，才會去尋找函式真正的位置進行 binding

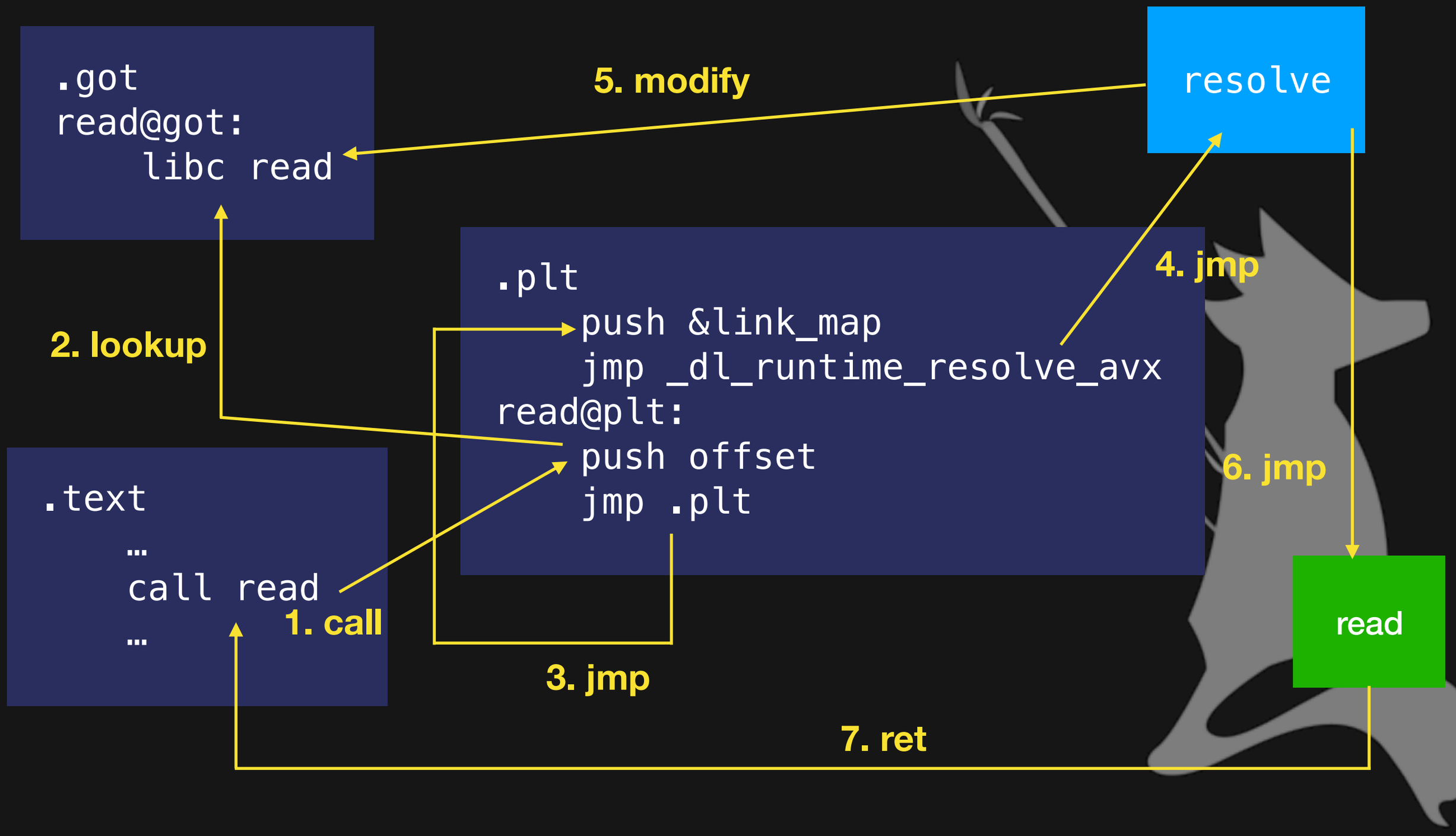
GOT & PLT

- Global Offset Table
- GOT 為一個函式指標陣列，存了其他 library 中 function 的位置，因為 Lazy binding 的機制，所以一開始只會填上一段 plt 位置的 code
- 第一次執行時，plt 會呼叫 _dl_fixup( )，才會去尋找真正的 function 並填入 GOT
- 第二次以後執行時，直接透過 GOT 找到 function 位置

GOT & PLT

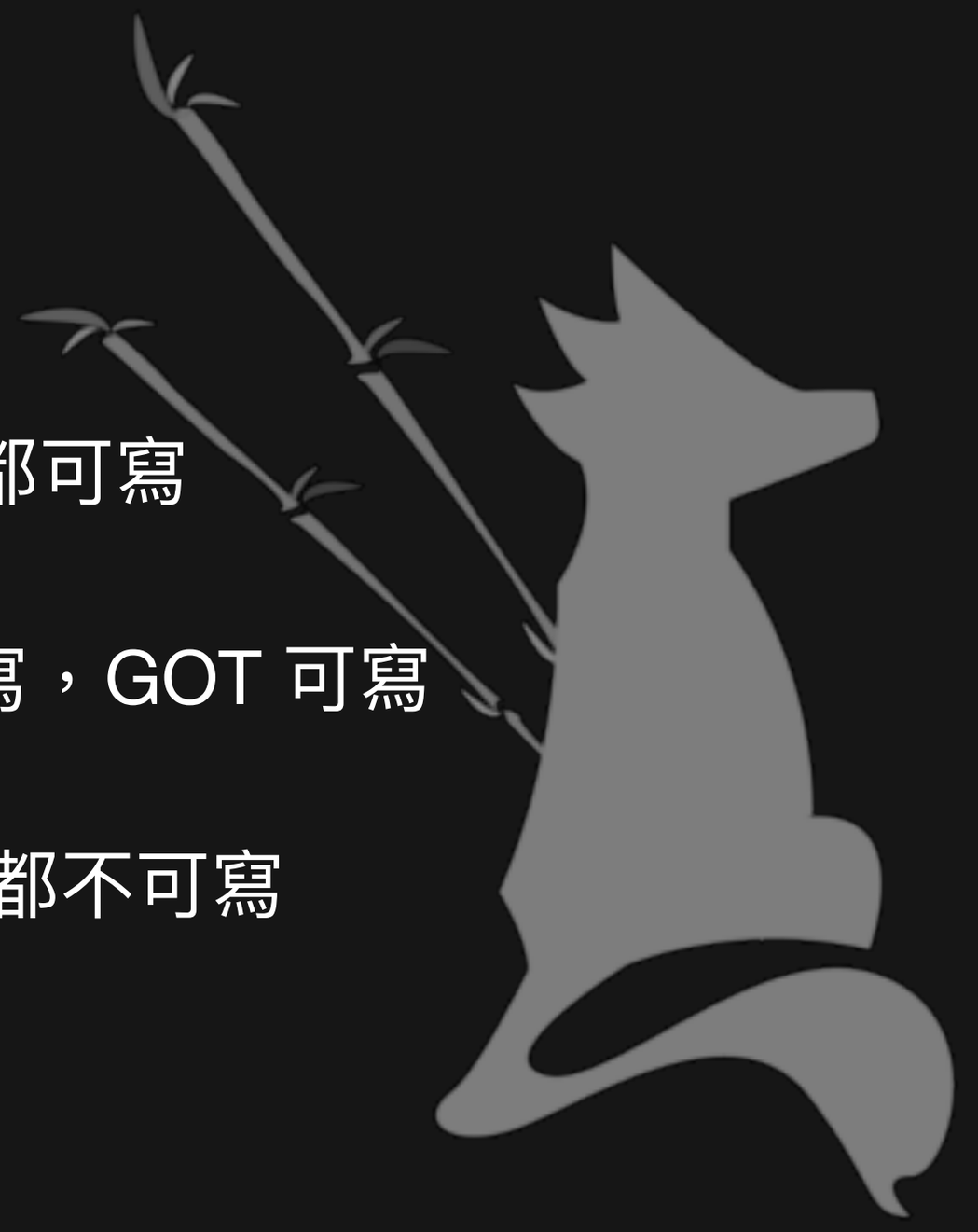

.got
read@got:
read@plt
5. modify
resolve
.plt
push &link_map
jmp _dl_runtime_resolve_avx
read@plt:
push offset
jmp .plt
4. jmp
2. lookup
.text
…
call read
…
1. call
3. jmp
read

GOT & PLT


.got
read@got:
libc read
5. modify
resolve
.plt
push &link_map
jmp _dl_runtime_resolve_avx
read@plt:
push offset
jmp .plt
4. jmp
2. lookup
.text
…
call read
…
1. call
3. jmp
6. jmp
read
7. ret

RELRO

- RELocation Read Only
- No / Partial / Full
  - No RELRO - link map 和 GOT 都可寫
  - Partial RELRO - link map 不可寫，GOT 可寫
  - Full RELRO - link map 和 GOT 都不可寫

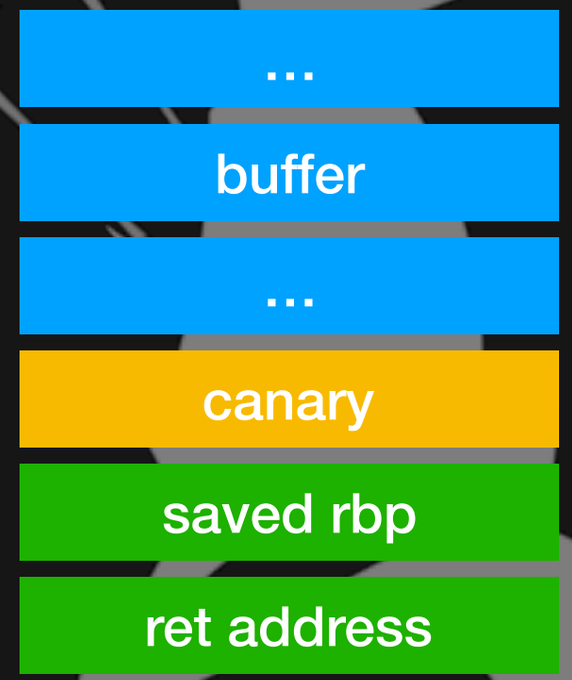


Stack Canary

- 在 rbp 之前塞一個 random 值，ret 之前檢查是否相同，不同的話就會 abort

- 有 canary 的話不能蓋到 return address、rbp

```
% ./test
aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
*** stack smashing detected ***: <unknown>
terminated
zsh: abort (core dumped)  ./test
```

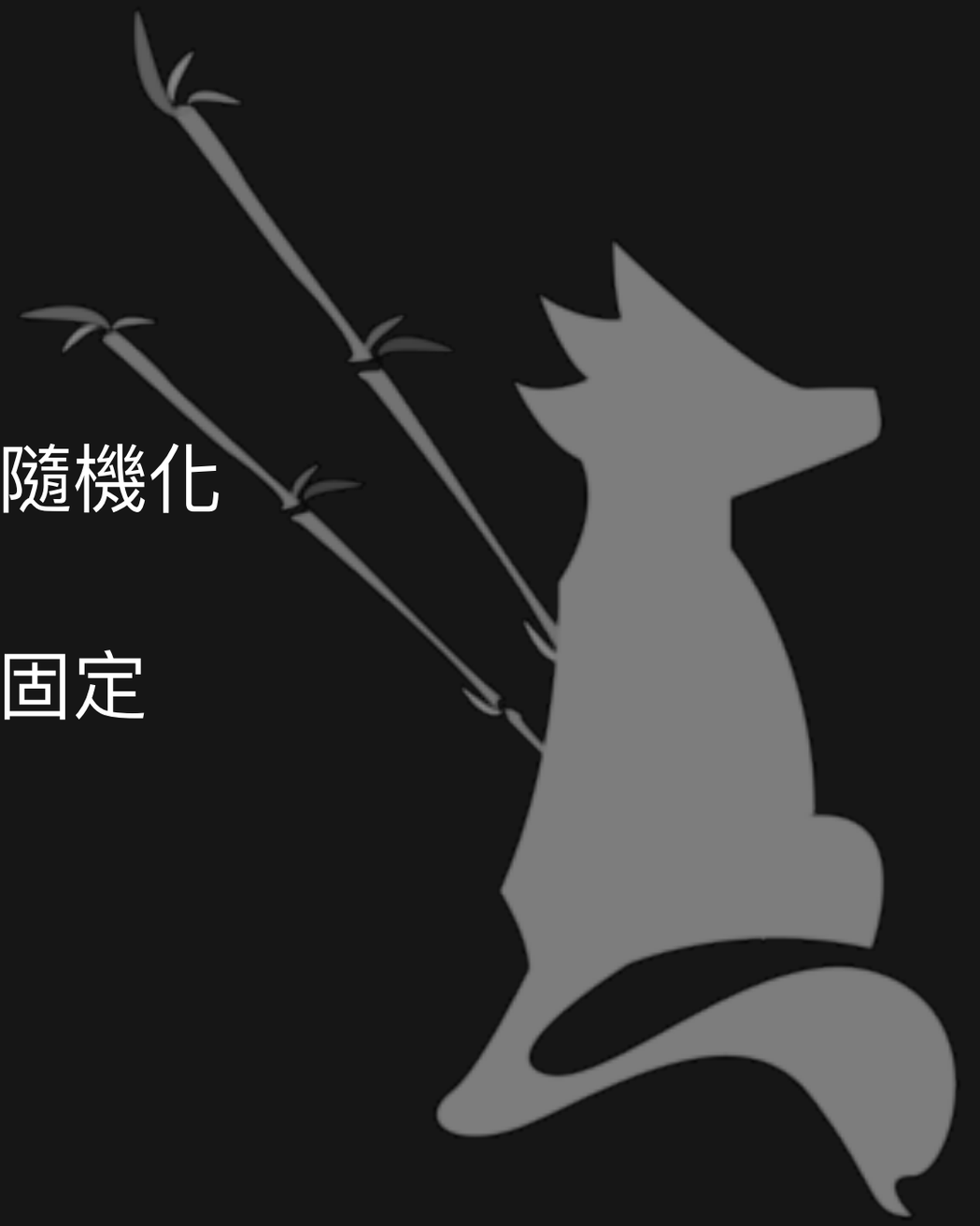


NX

- No eXecute
- 又稱 DEP (Data Execution Prevention)
- 可寫得不可執行，可執行的不可寫

```
gdb-peda$ vmmap
Start              End                Perm      Name
0x00400000         0x00401000         r-xp      /home/frozenkp/csie/class/a.out
0x00600000         0x00601000         r-xp      /home/frozenkp/csie/class/a.out
0x00601000         0x00602000         rwxp      /home/frozenkp/csie/class/a.out
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-xp      /lib/x86_64-linux-gnu/libc-2.27.so
0x00007ffff7dcf000 0x00007ffff7dd1000 rwxp      /lib/x86_64-linux-gnu/libc-2.27.so
0x00007ffff7dd1000 0x00007ffff7dd5000 rwxp      mapped
0x00007ffff7dd5000 0x00007ffff7dfc000 r-xp      /lib/x86_64-linux-gnu/ld-2.27.so
0x00007ffff7fe0000 0x00007ffff7fe2000 rwxp      mapped
0x00007ffff7ff7000 0x00007ffff7ffa000 r--p      [vvar]
0x00007ffff7ffa000 0x00007ffff7ffc000 r-xp      [vdso]
0x00007ffff7ffc000 0x00007ffff7ffd000 r-xp      /lib/x86_64-linux-gnu/ld-2.27.so
0x00007ffff7ffd000 0x00007ffff7ffe000 rwxp      /lib/x86_64-linux-gnu/ld-2.27.so
0x00007ffff7ffe000 0x00007ffff7fff000 rwxp      mapped
0x00007ffffffde000 0x00007ffffffff000 rwxp      [stack]
0xffffffffff600000 0xffffffffff601000 r-xp      [vsyscall]
```

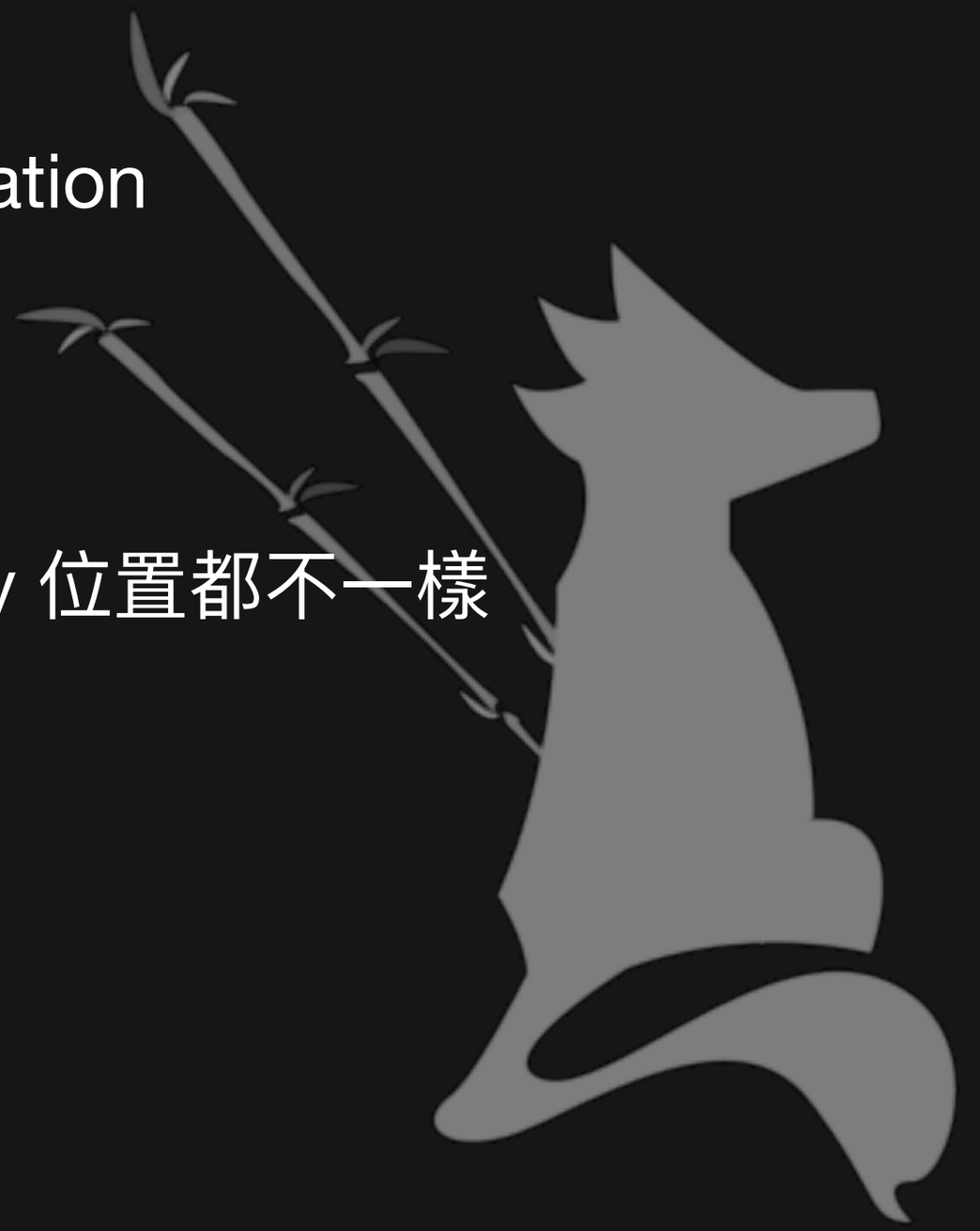
PIE



- Position Independent Executable
- 開啟時，data 段以及 code 段位址隨機化
- 關閉時，data 段以及 code 段位址固定

ASLR

- Address Space Layout Randomization
- 記憶體位址隨機變化
- 每次執行時，stack、heap、library 位置都不一樣
- ASLR 是系統設定，非程式的設定



Tools

Overview

- nc / ncat
- objdump
- gdb
- checksec
- gdb-vmmap
- pwntools
- readelf
- ROPgadget
- one_gadget
- radare2

nc / ncat

- pwn 題常用的遠端連線工具
- 使用 ncat 將程式在遠端架起來，接著使用 nc 連線
- 範例

```
% ncat -vc $binary -kl $ip $port
```

```
% nc $ip $port
```

objdump

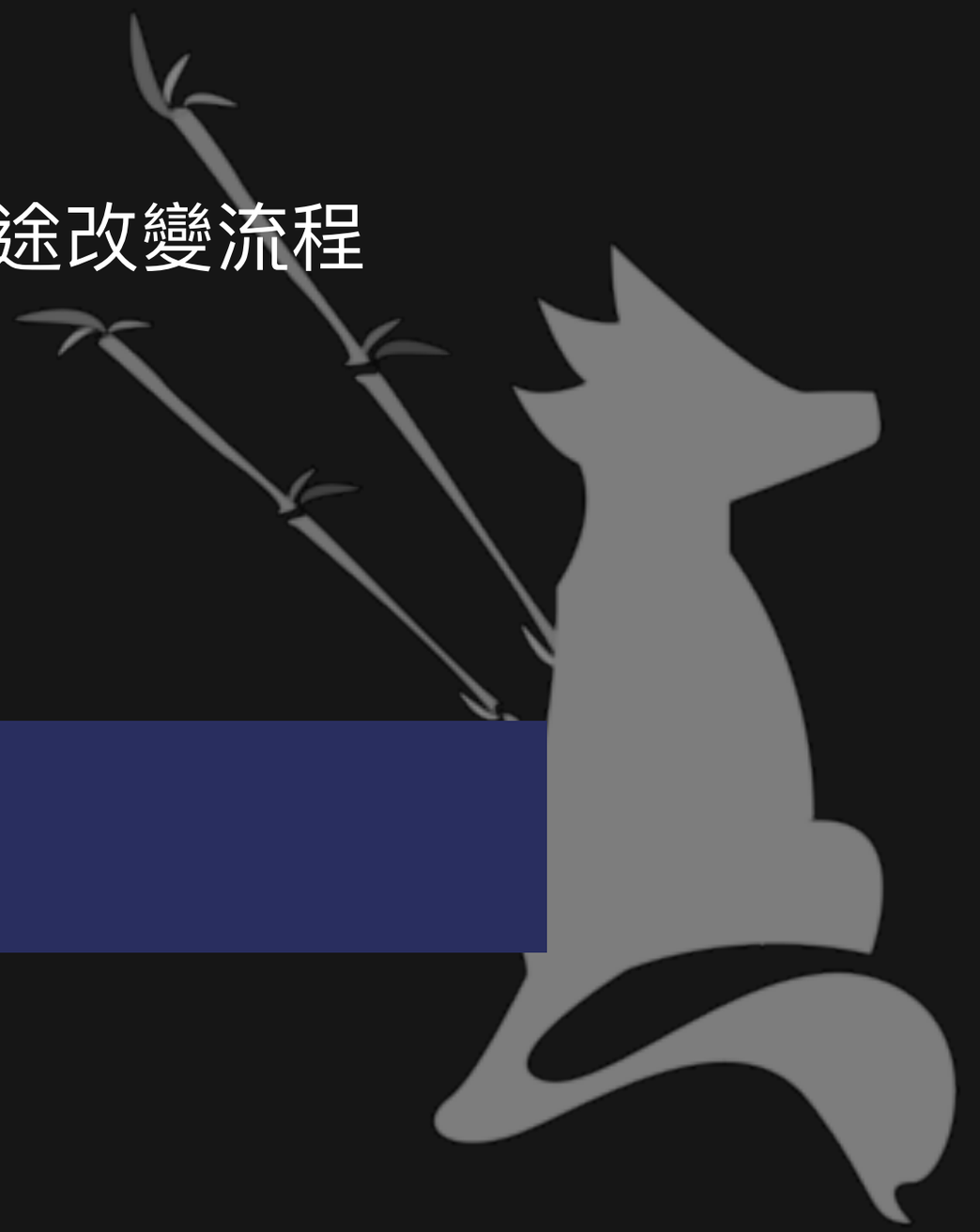
- dump 出執行檔中的組合語言
- 可以搭配 less / grep 使用
- 範例

```
% objdump -M intel -d $binary | less
```

gdb

- 追蹤程式流程
- 可以追蹤每一行組語，也可以在中途改變流程
- 範例

```
% gdb $binary
```



gdb

❖ 設斷點 (break)

```
% b main
% b *0x4896aa
```

❖ 執行 (run)

```
% r
```

❖ 繼續執行 (continue)

```
% c
```

gdb

❖ 下一行指令 (next instruction)

- 不會進入 call function，直接跳到同一區塊的下個指令

```
% ni
```

❖ 下個執行的指令 (step instruction)

- 會進入 call function

```
% si
```

gdb

❖ 跳到某個位址 (jump)

```
% j *0x4896aa
```

❖ 秀出某位址的值

```
% x/10gx 0x400686
```

❖ 設定某個位址 / 暫存器得值

```
% set $rax=0x5
```

checksec

❖ 查看某個程式的安全性保護

```
% checksec $binary
```

```
[XD] % checksec ./bof1
[*] '/home/frozenkp/csie/lab/bof1/bof1'
    Arch:     amd64-64-little
    RELRO:    Full RELRO
    Stack:    No canary found
    NX:       NX enabled
    PIE:      PIE enabled
```

gdb-vmmap

❖ 查看目前程式的記憶體分佈，以及 rwx 權限設定

```
% vmmap
```

```
gdb-peda$ vmmap
Start              End                Perm      Name
0x0000555555554000 0x0000555555555000 r-xp      /home/frozenkp/csie/lab/bof1/bof1
0x0000555555754000 0x0000555555755000 r--p      /home/frozenkp/csie/lab/bof1/bof1
0x0000555555755000 0x0000555555756000 rw-p      /home/frozenkp/csie/lab/bof1/bof1
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
0x00007ffff7fdb000 0x00007ffff7fdd000 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
0x00007ffffffde000 0x00007ffffffff000 rw-p      [stack]
0xffffffffff600000 0xffffffffff601000 r-xp      [vsyscall]
```

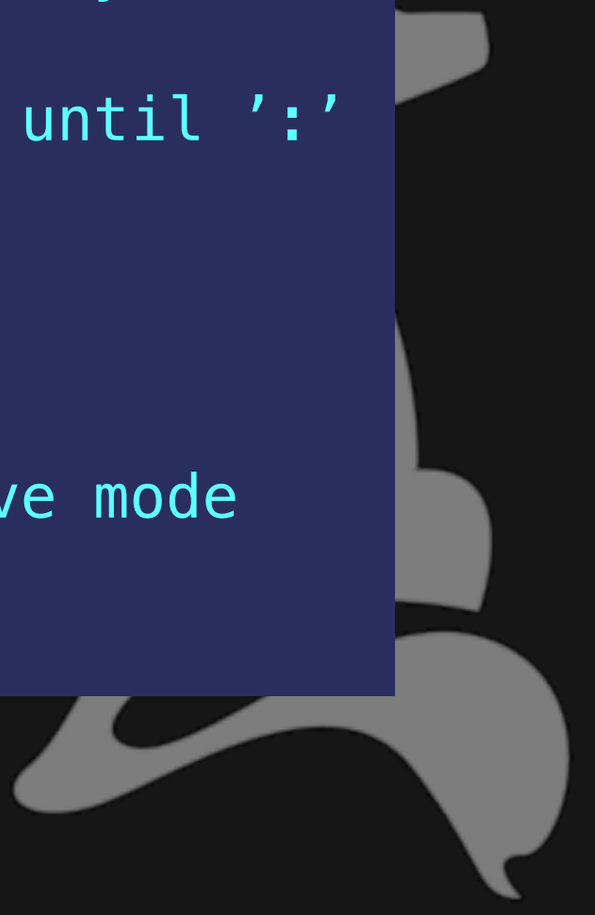
pwntools

- 用來和遠端程式互動的 python 套件
- github 連結
- 範例

```
% pip install pwntools
```

pwntools

```
1  from pwn import *
2
3  # connect to server
4  r = process('./add')                  # localhost binary
5  r = remote('140.113.0.3', 8080)  # remote binary
6
7  s = r.recvuntil(':')    # receive from binary until ':'
8  print '1: ' + s
9
10 r.sendline('3 5')       # send to server
11
12 r.interactive()         # switch to interactive mode
13
```



readelf

- 分析 libc 的工具
- 範例

```
% readelf -a $libc | less
```

ROPgadget

- 列出 binary 中可以使用的 ROP gadget
- github 連結
- 範例

```
% ROPgadget --binary $binary
```

one_gadget

- 在 libc 中尋找可以一步開啟 shell 的 gadget
- 限制
  - 須有 libc base
  - 須滿足必要條件 (constraints)
- github 連結
- 範例

```
% one_gadget $libc
```

radare2

- 動態、靜態分析都可以用的工具
- github 連結
- 使用教學

```
% r2 $binary
```

Thanks for listening.