# Outline

- Format String
- Read from arbitrary memory
- Write to arbitrary memory
- Advanced Trick

# Format String

- 輸出或字串處理函式中，用來表示輸出格式的字串
- 像是 printf 中常用的 %s %d 等
- 其他常見的 function
  - sprintf
  - fprintf

# Format String

- 正常的情況下，一個 %(s|d|x..) 應該要對應到一個參數，例如 printf(“%d %d”,a,b)
- 若未適當對映好，可能會造成 information leakage
  - printf(“%p %p”,a)
    - 多出來的 %s 會存取到一些記憶體上的資訊

# Format String

```
1 #include <stdio.h>
2
3 int main(){
4     int a = 1;
5     printf("%p,%p\n",a);
6     return 0;
7 }
~
```

```
angelboy@ubuntu:~$ ./ttt
0x1,0x7fff5c9122f8
```

# Format String

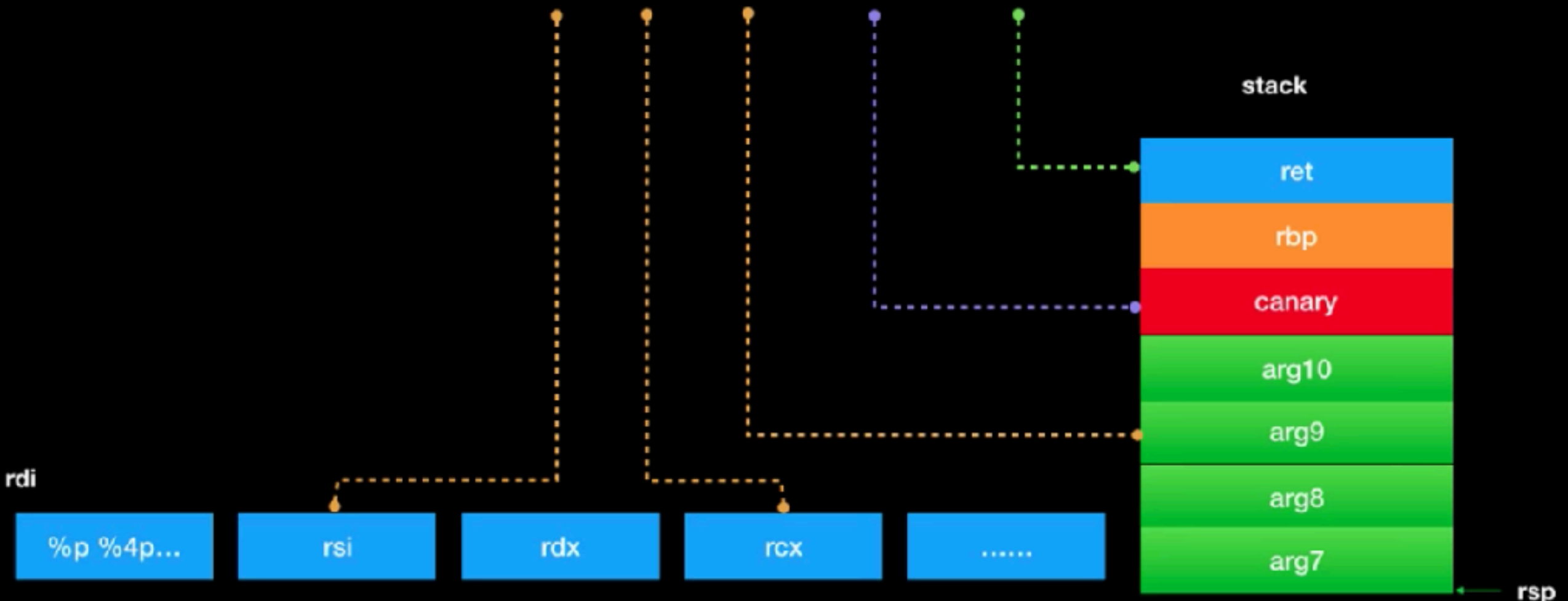
- 錯誤的使用方式
  - format strings 的內容是由使用者可控的，這樣會造成任意記憶體讀寫
  - read(0,buf,0x100);printf(buf)
- 編譯時基本上會有 warnning 但一般使用者通常會忽略他

# Format String

- 利用方式
  - printf(“%7$p”)
    - %*$ 是指定 p 要讀第幾個參數，%7$p 代表要讀 printf 第 8 個參數 (因為第一個參數是 (“%7$p”) )，並用格式 p 印出來
    - printf 也不會檢查參數個數是否有 match ，所以要讀多遠都可以

# Format String

- When we call printf("%p %3p %8$p %10$p %12$p")

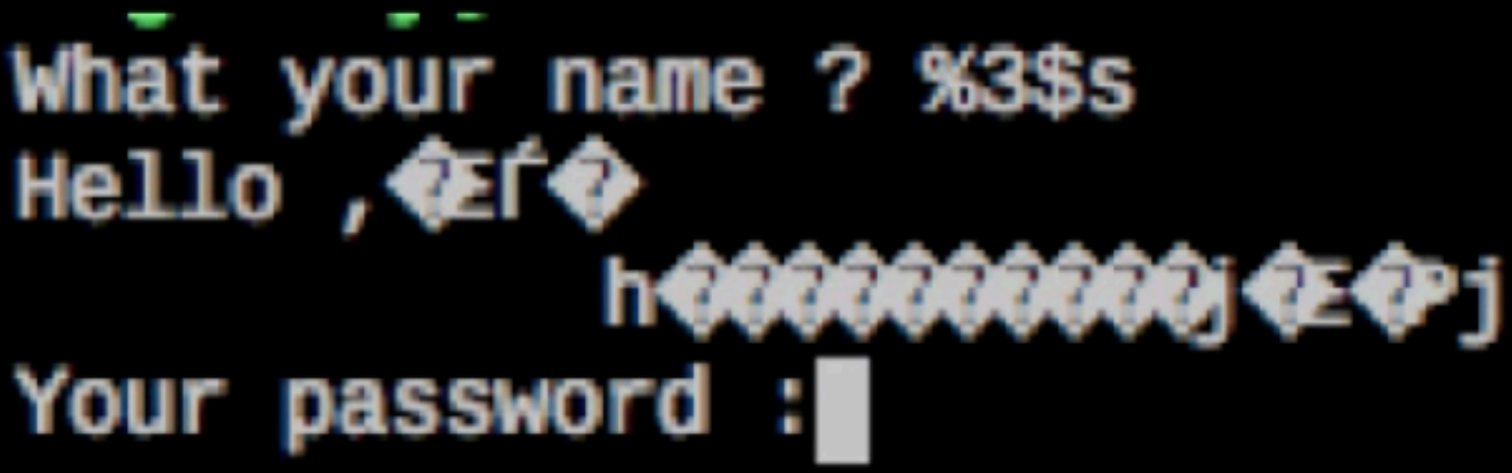


# Outline

- Format String
- Read from arbitrary memory
- Write to arbitrary memory
- Advanced Trick

# Read from arbitrary memory

- 在 stack 上如果有可控 address 的 buffer 時，可以利用 “%s” 來將該 address 做 dereference 將內容當成字串印出來

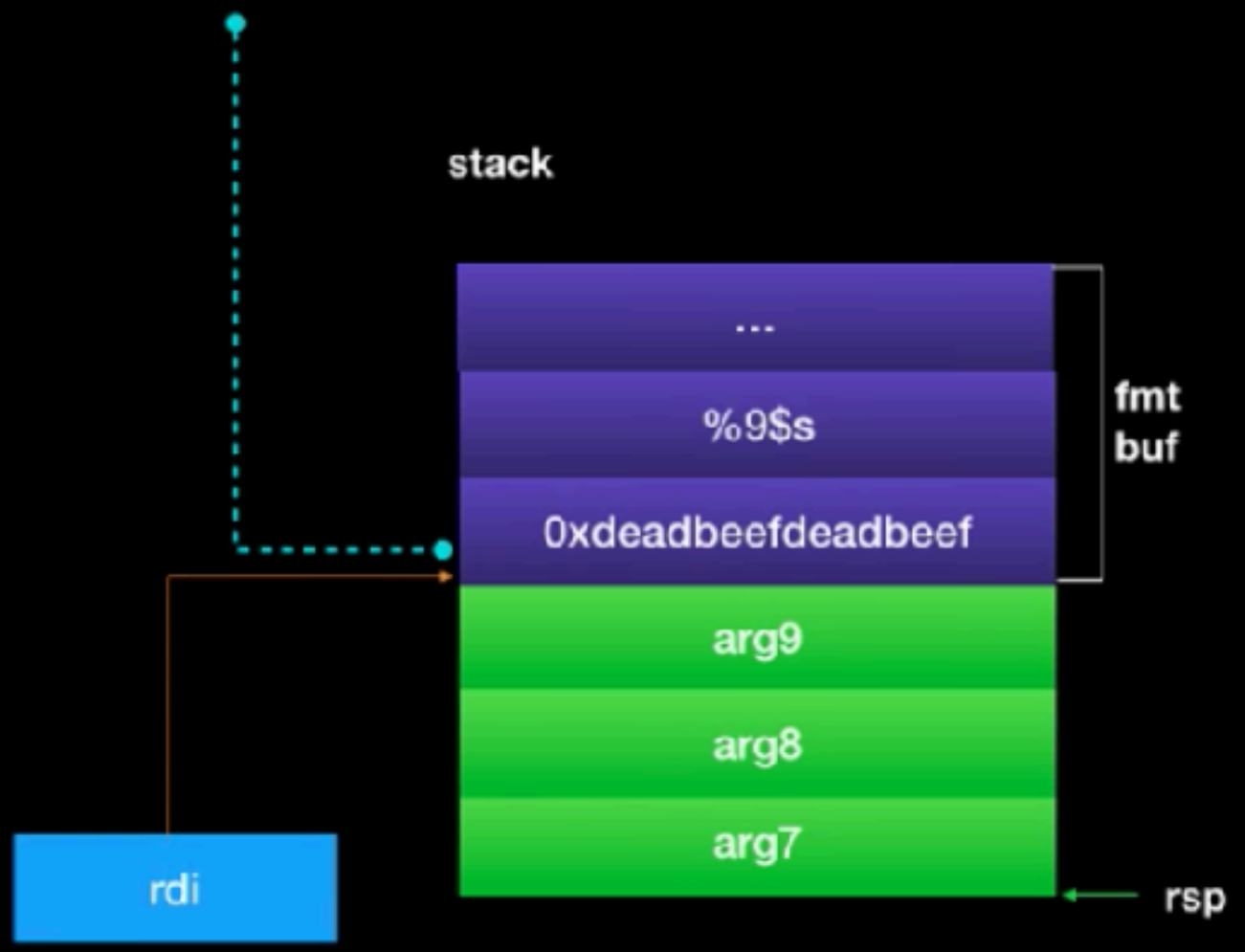


# Read from arbitrary memory

- 若 format string 也在 stack 上，那麼在做 format string 時，可以同時將 address 放入，當成 pointer 使用，這樣就可以做到任意讀寫
- 但 address 內容不可有 NULL byte 否則 format string 會被中斷
  - 可以將 address 改放到後面來避免掉這問題，但 padding 要算好

# Read from arbitrary memory

- printf("\xef\xbe\xad\xde\xef\xbe\xad\xde%9$s")
  - puts *0xdeadbeefdeadbeef
  - 任意記憶體位置讀取



# Read from arbitrary memory

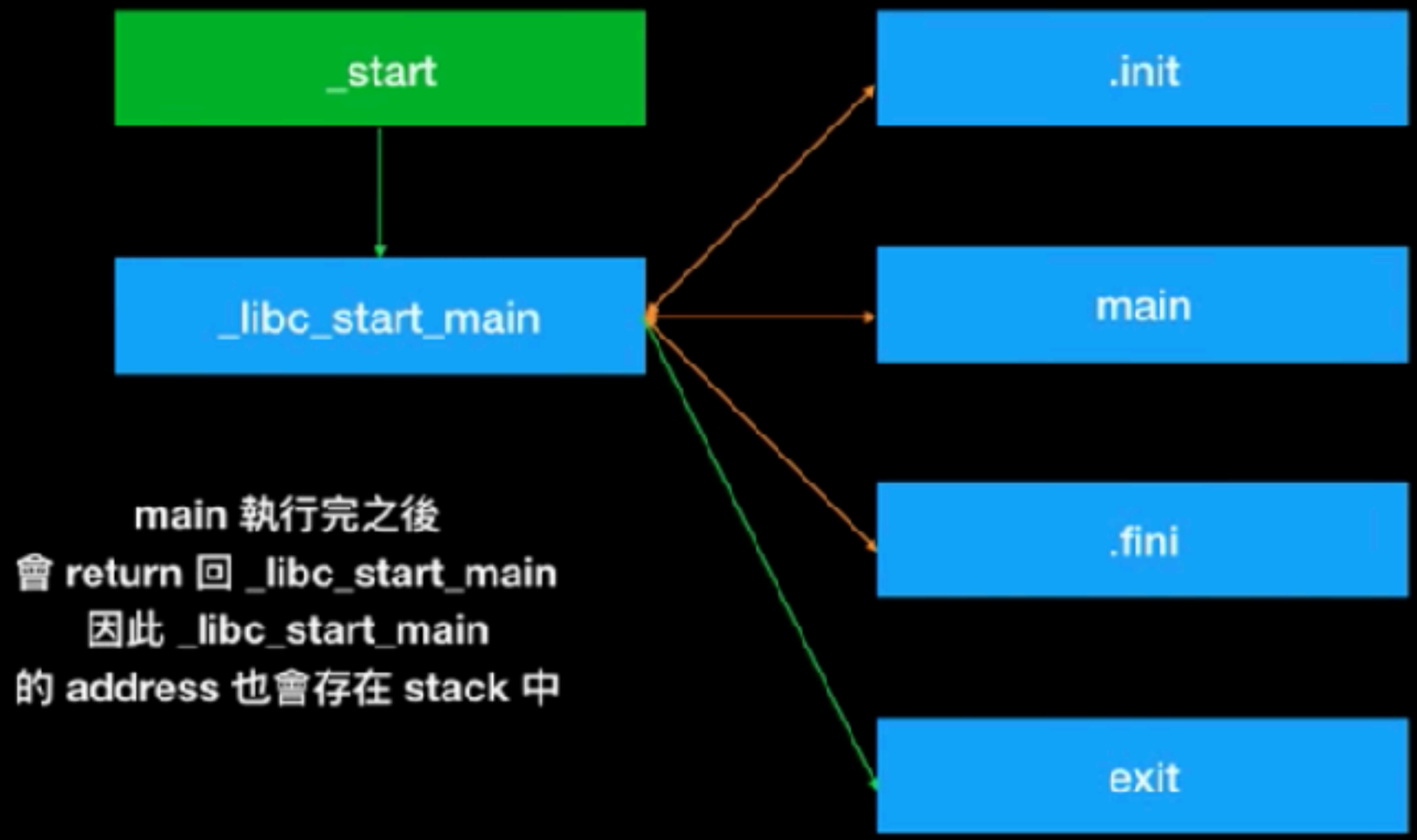
- printf("**%10$s**aaa\xef\xbe\xad\xde\x00\x00\x00\x00")
  - Address 有 null byte
  - 通常 64bit address 一定有 null byte
  - 橘色部分為 padding 補滿 8 倍數這樣參數才對

# Read from arbitrary memory

- leak libc 位置
- 在前面的狀況下，可以利用填入 got 後用 %s 來 leak libc 的位置
- 而另一種方式則是利用 _libc_start_main 的 return address 來計算出 libc 的位置

# Read from arbitrary memory

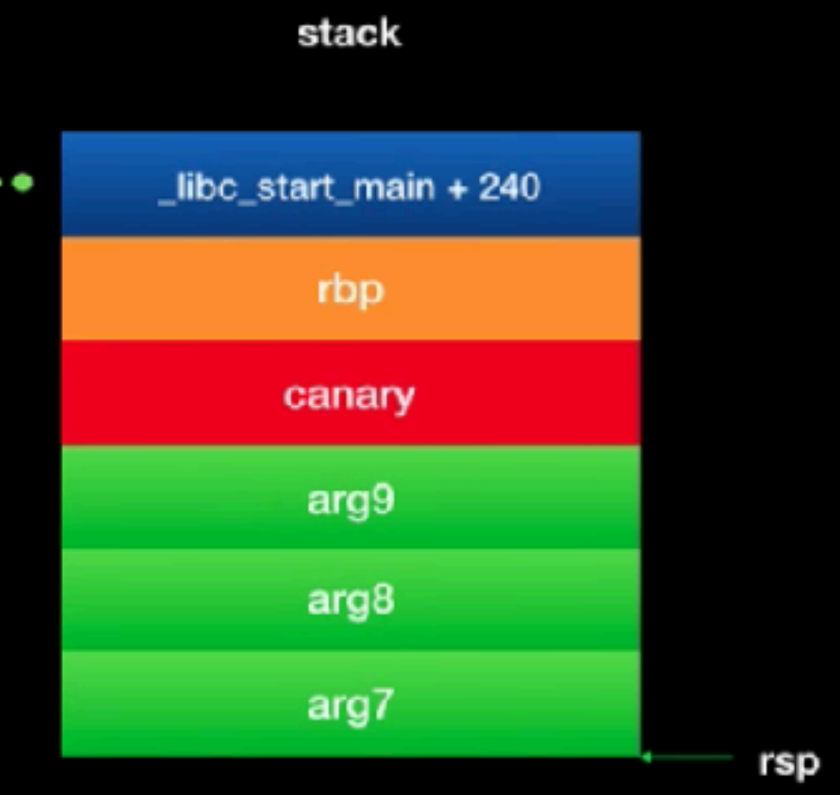
- 回顧先前程式執行的流程




main 執行完之後
會 return 回 _libc_start_main
因此 _libc_start_main
的 address 也會存在 stack 中

# Read from arbitrary memory

- 假設 main 中有 format string
  - printf(“%11$p”)
  - leak 到的資料會是 libc_start_main 的
  - 所以在 leak 完後必須先扣掉那個 offset
  - 然後再用 libc_start_main 的 offset 算出



# Outline

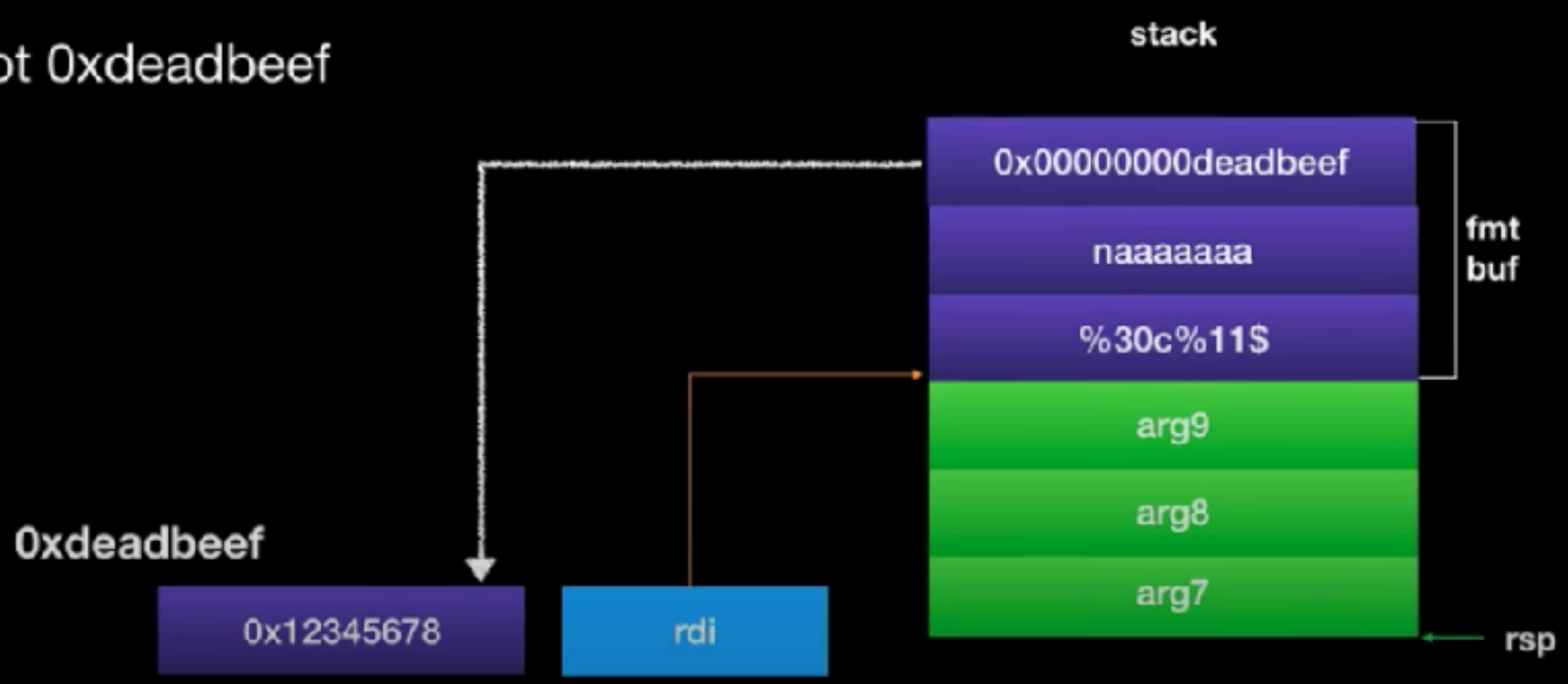
- Format String
- Read from arbitrary memory
- Write to arbitrary memory
- Advanced Trick

# Write to arbitrary memory

- %n 可以對特定參數寫數值，寫入的值為目前已顯示的字元數
  - “12345%3$n” 表示對第四個參數指向位置寫入 len(“12345”) = 5 這個數值
- 可以配合 %c 來做寫入
  - %xxc 為印出 xx 個字元到螢幕上
  - “%123c%3$n” 表示對第四個參數寫入 123 這個數值

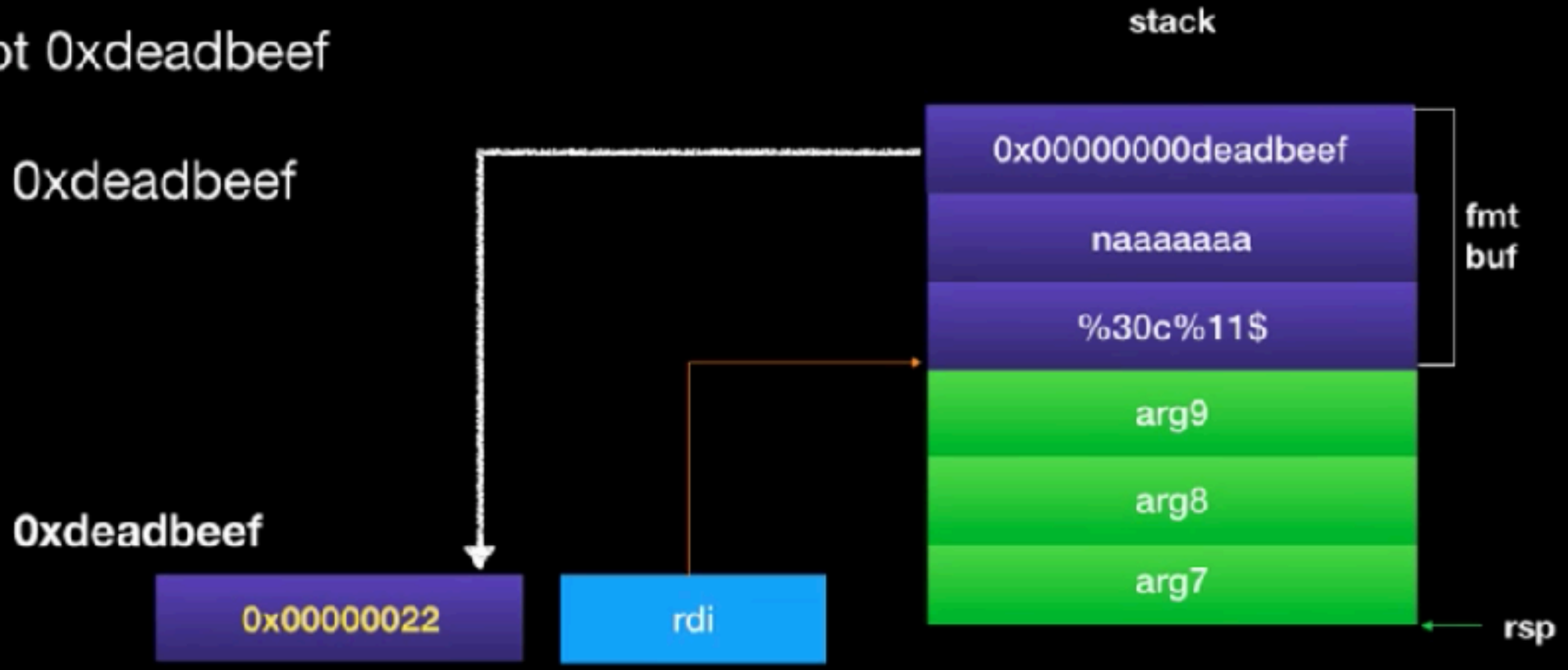
# Write to arbitrary memory

- printf("%30c%11$naaaaaaaa\xef\xbe\xad\xde\x00\x00\x00\x00")
  - Assume got 0xdeadbeef



# Write to arbitrary memory

- printf("%30c%11$naaaaaaaa\xef\xbe\xad\xde\x00\x00\x00\x00")
  - Assume got 0xdeadbeef
  - write 30 to 0xdeadbeef

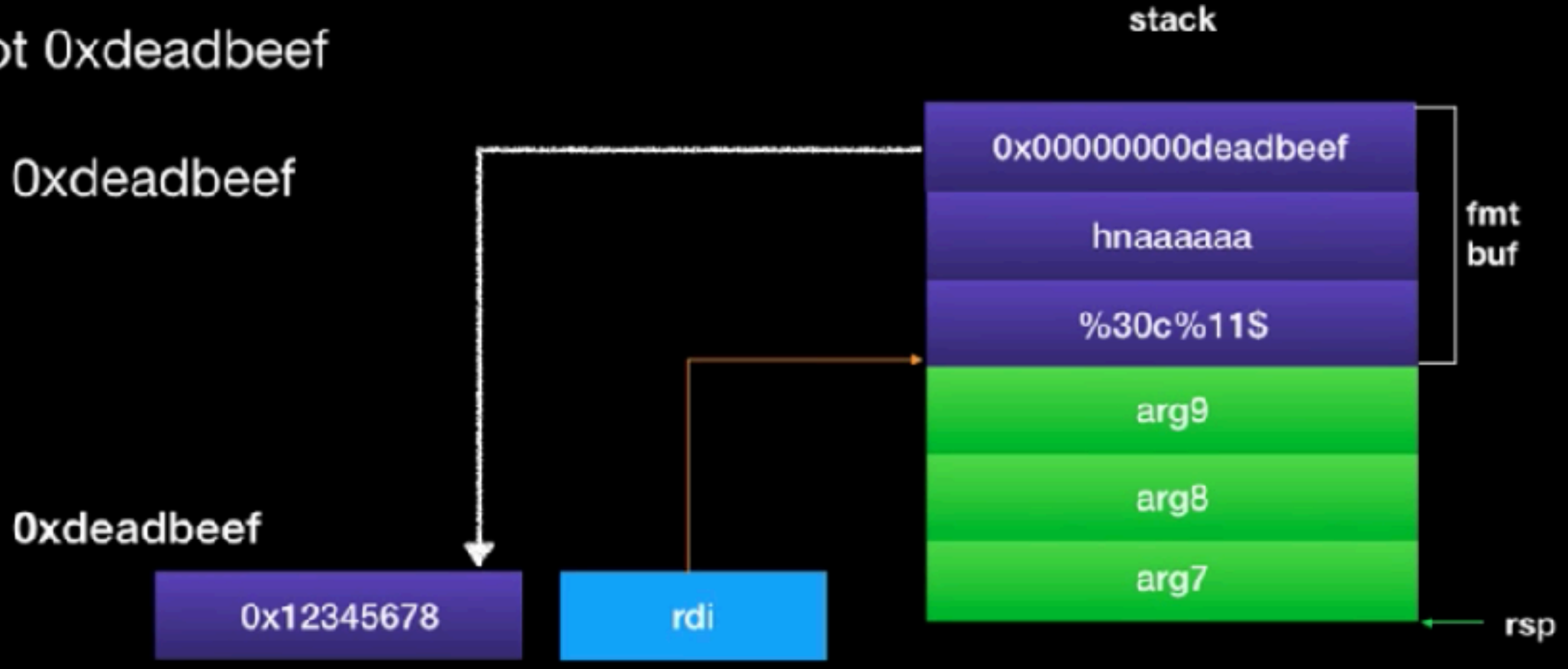


# Write to arbitrary memory

- 但 %n 寫入大小為一個 int 的大小，而我們若要在 GOT 寫上一個 adddress 大小時，需要印出非常大的字元數量，輸出時間也會非常久
- 我們可以利用 %hn 或者是 %hhn 來進行寫值
- %ln 針對參數寫 8 個 byte (long) 的數值
- %hn 針對參數寫 2 個 byte (short) 的數值
- %hhn 針對參數寫 1 個 byte (char) 的數值

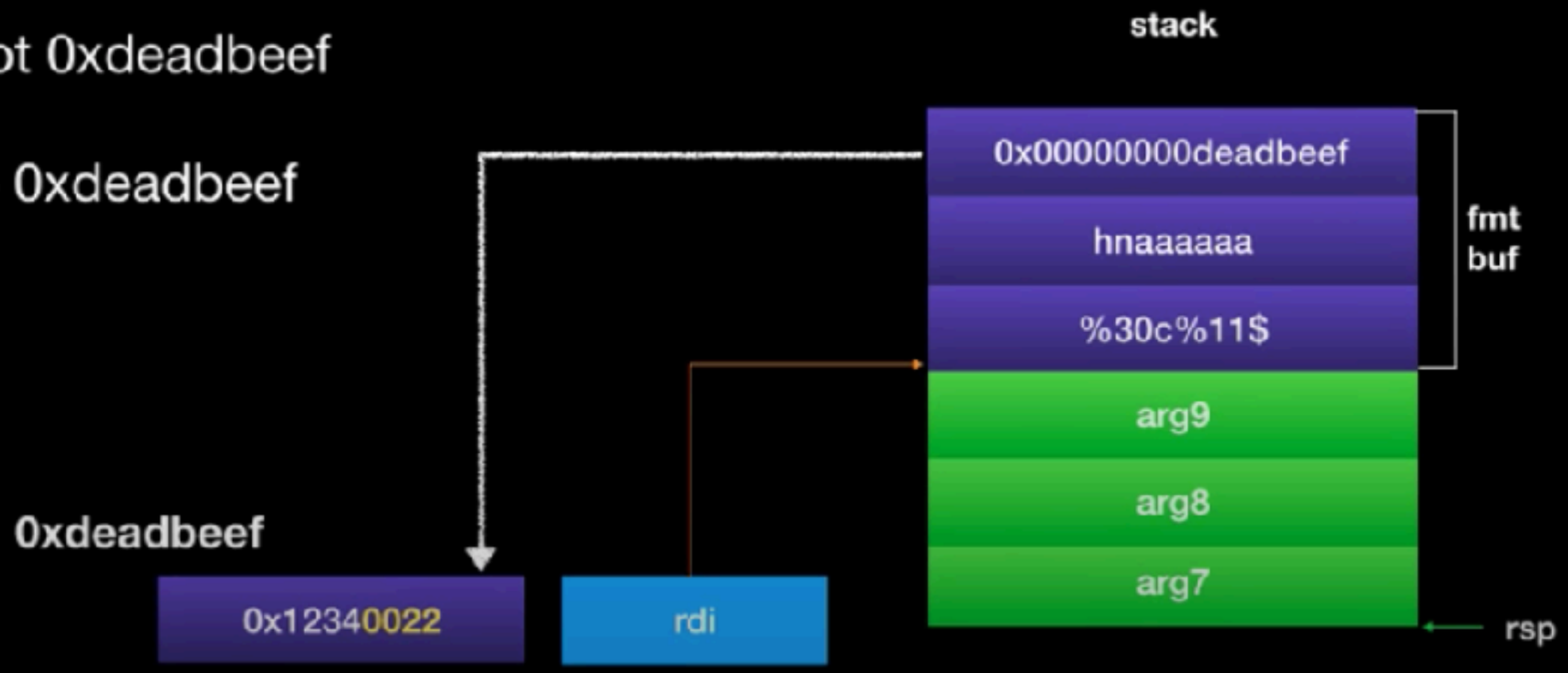
# Write to arbitrary memory

- printf("%30c%11$hnaaaaaaa\xef\xbe\xad\xde\x00\x00\x00\x00")
  - Assume got 0xdeadbeef
  - write 30 to 0xdeadbeef



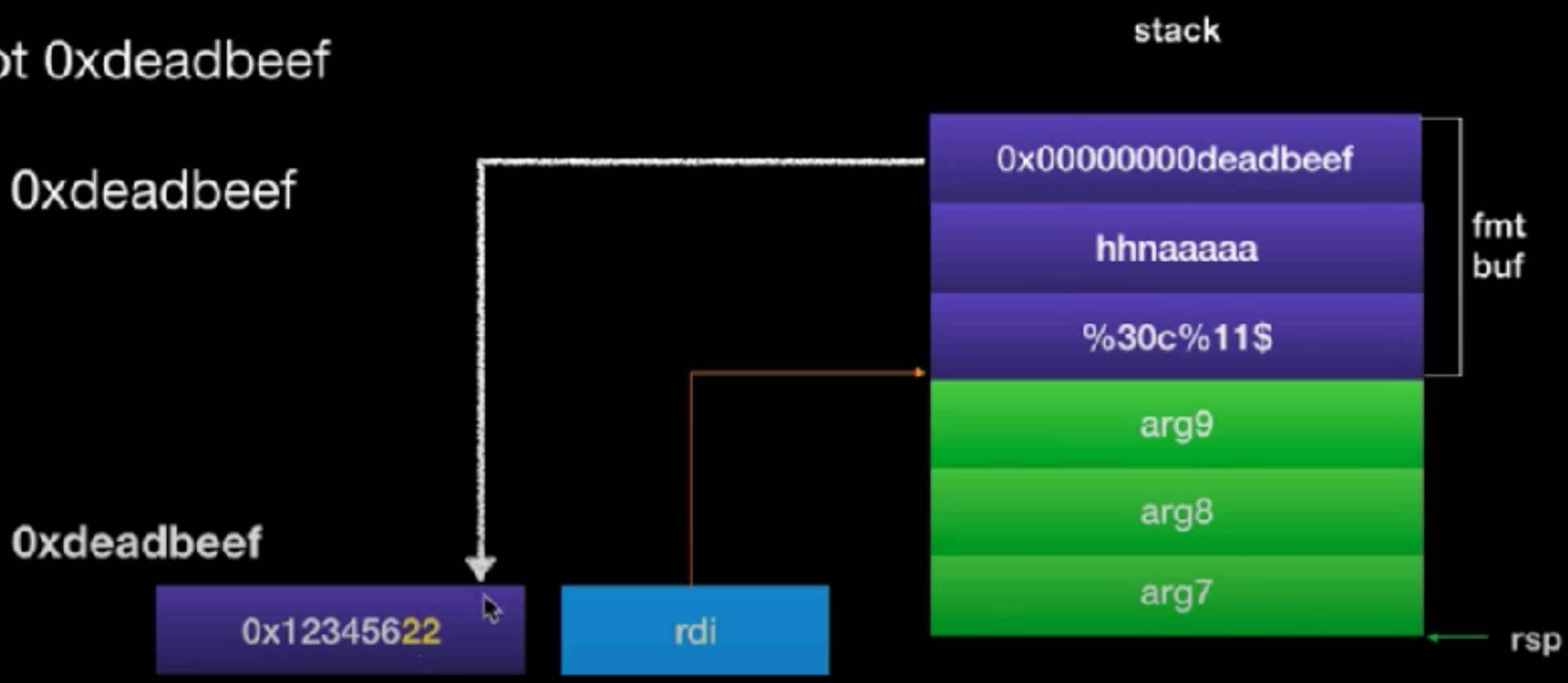
# Write to arbitrary memory

- printf("%30c%11$hnaaaaaaa\xef\xbe\xad\xde\x00\x00\x00\x00")
  - Assume got 0xdeadbeef
  - write 30 to 0xdeadbeef



# Write to arbitrary memory

- printf("%30c%11$hhnaaaaaaa\xef\xbe\xad\xde\x00\x00\x00\x00")
  - Assume got 0xdeadbeef
  - write 30 to 0xdeadbeef



# Write to arbitrary memory

- 通常我們在對 GOT 寫值時，並不會單純一次寫了一個 byte 就結束，因為很有可能再做下一次 format string 時會先呼叫到該 function 的 GOT 有可能就會壞掉

- 所以通常我們會再一次 format string 下把所有 format string 串接起來，去針對一個 GOT 做寫入

# Write to arbitrary memory

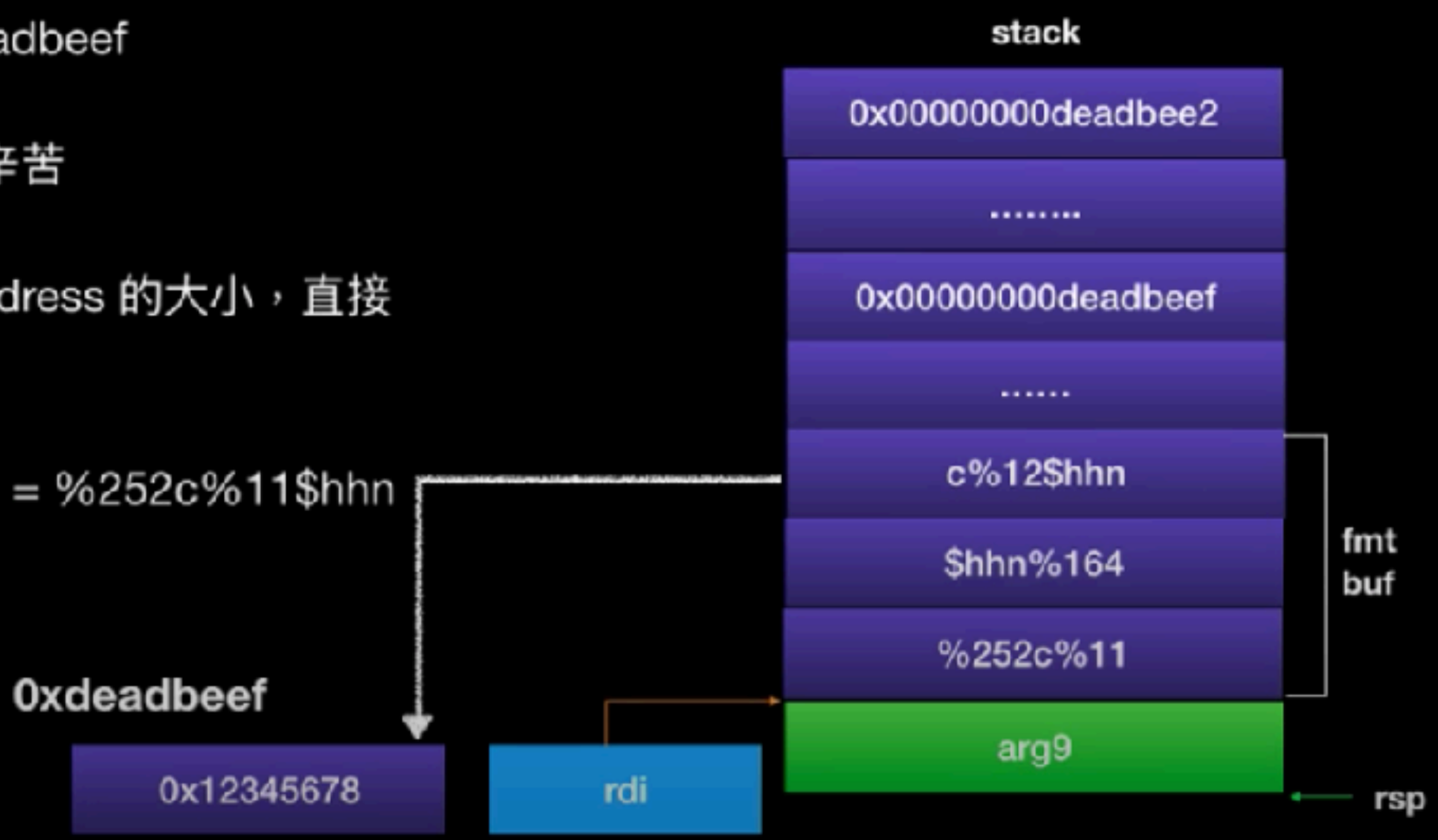
- 而我們在寫入一次寫入多個 format string 時，必須注意到前面已經印出的字元數

- 例如第一次寫值是 %30c%3$n 我們針對第四個參數寫入 30 ，而後面要繼續寫值時必須扣掉 30 ，假設我們要對第五個參數寫 100 那們就是要寫入 100-30 = 70

  - 最後生成的字串就是 %30c%3$hhn%70c%4$hhn

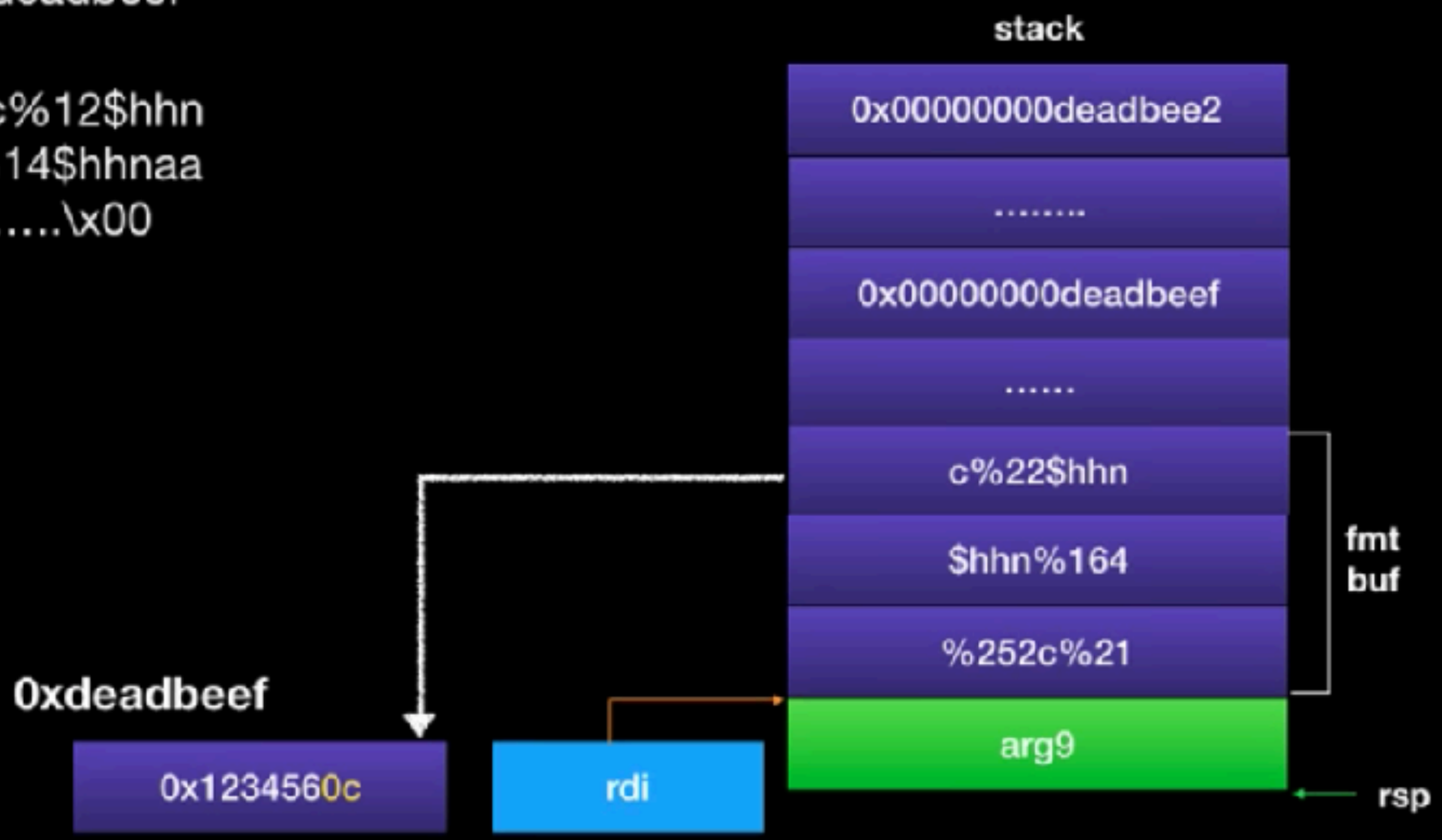
# Write to arbitrary memory

- Write 0xfaceb00c to 0xdeadbeef
- 一個一個算 padding 有點辛苦
  - 可事先算好要預留放 address 的大小，直接 padding 到該大小就好
- %(0xc-16+256)c%11$hhn = %252c%11$hhn
- %(0xb0-0xc)c%12$hhn
- …



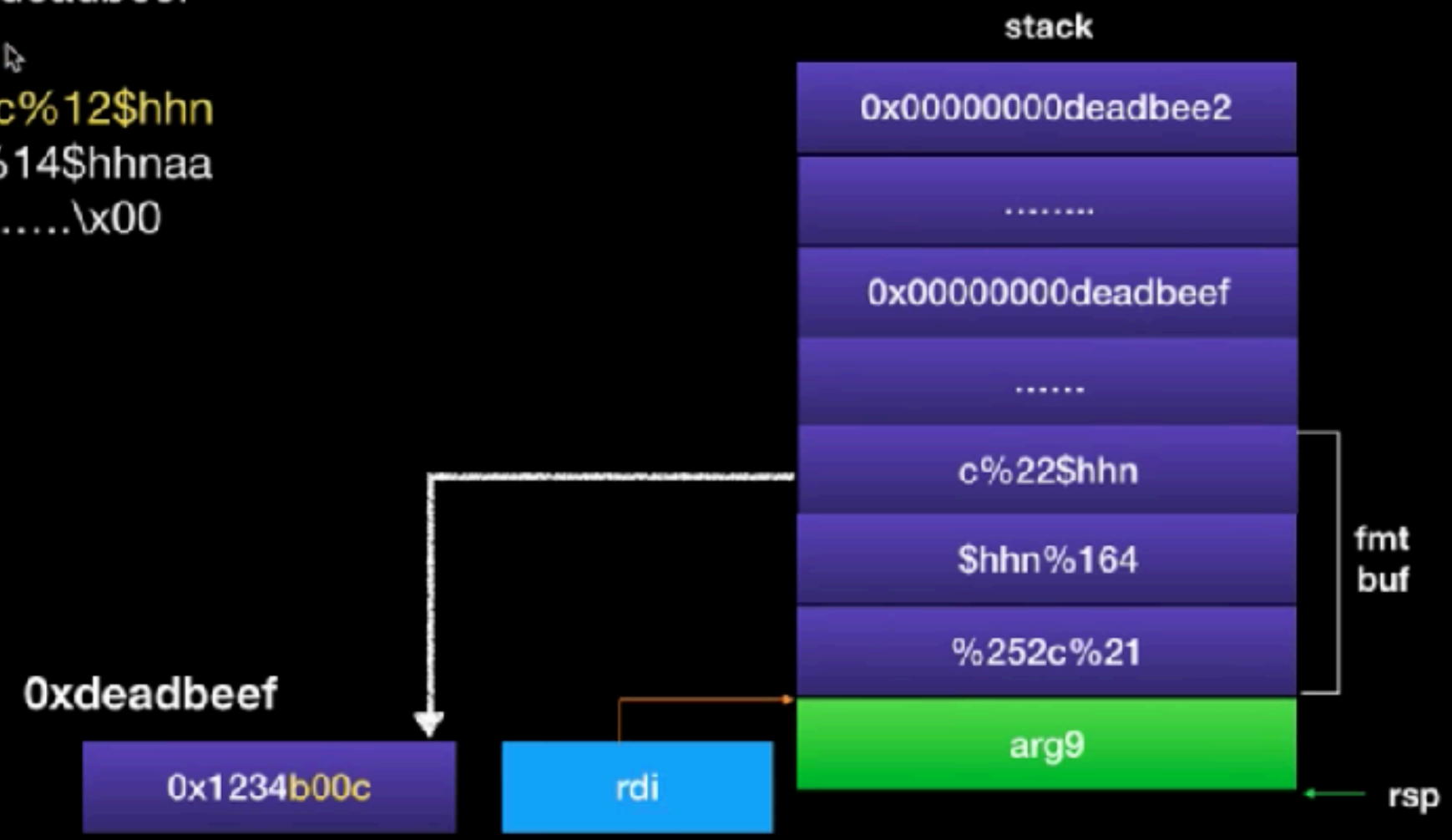
# Write to arbitrary memory

- Write 0xfaceb00c to 0xdeadbeef
- %252c%11$hhn%164c%12$hhn
  %30c%13$hhn%44c%14$hhnaa
  aa\xef\xbe\xad\xde.....\x00



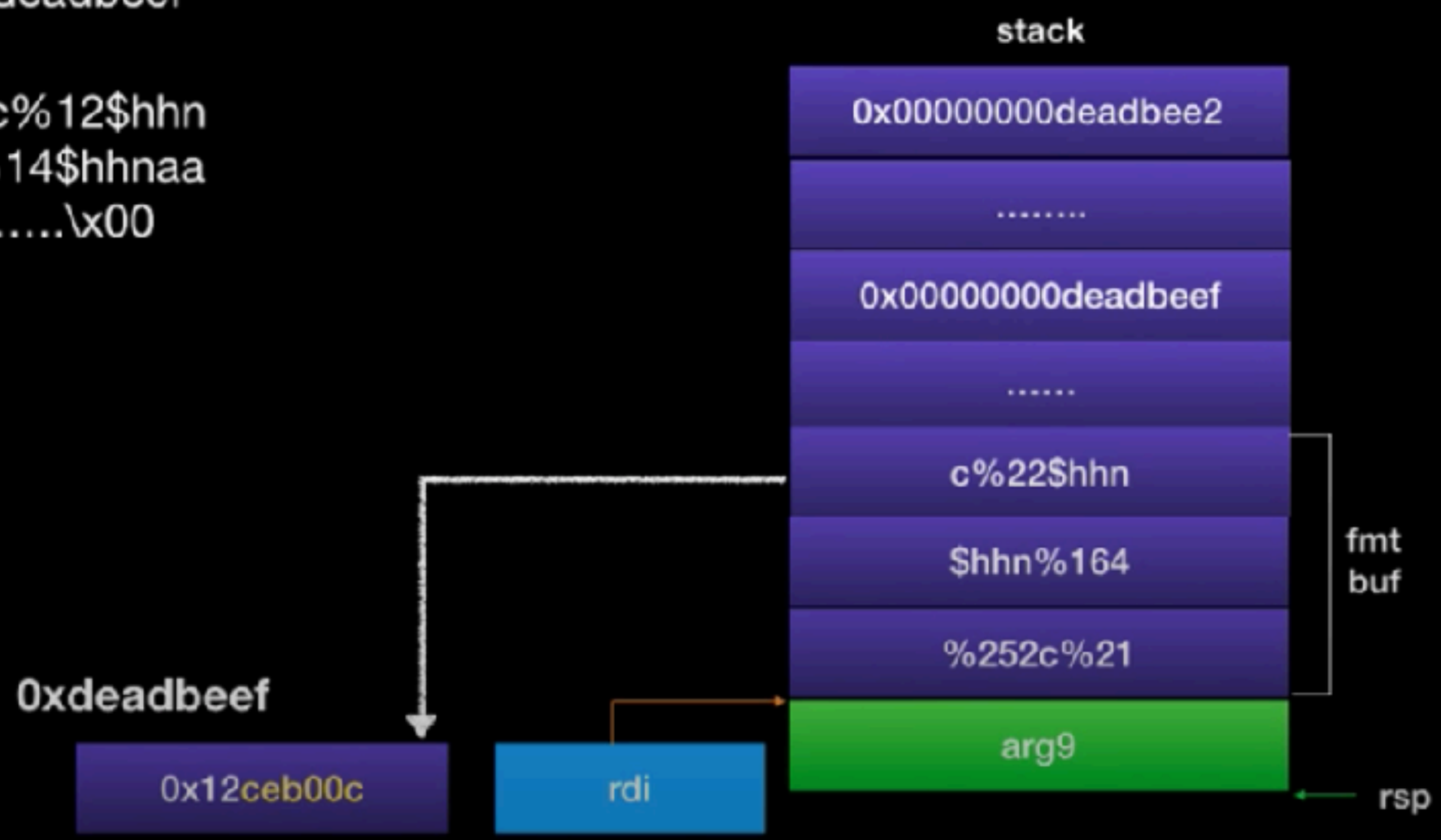
# Write to arbitrary memory

- Write 0xfaceb00c to 0xdeadbeef
- %252c%11$hhn%164c%12$hhn
  %30c%13$hhn%44c%14$hhnaa
  aa\xef\xbe\xad\xde.....\x00



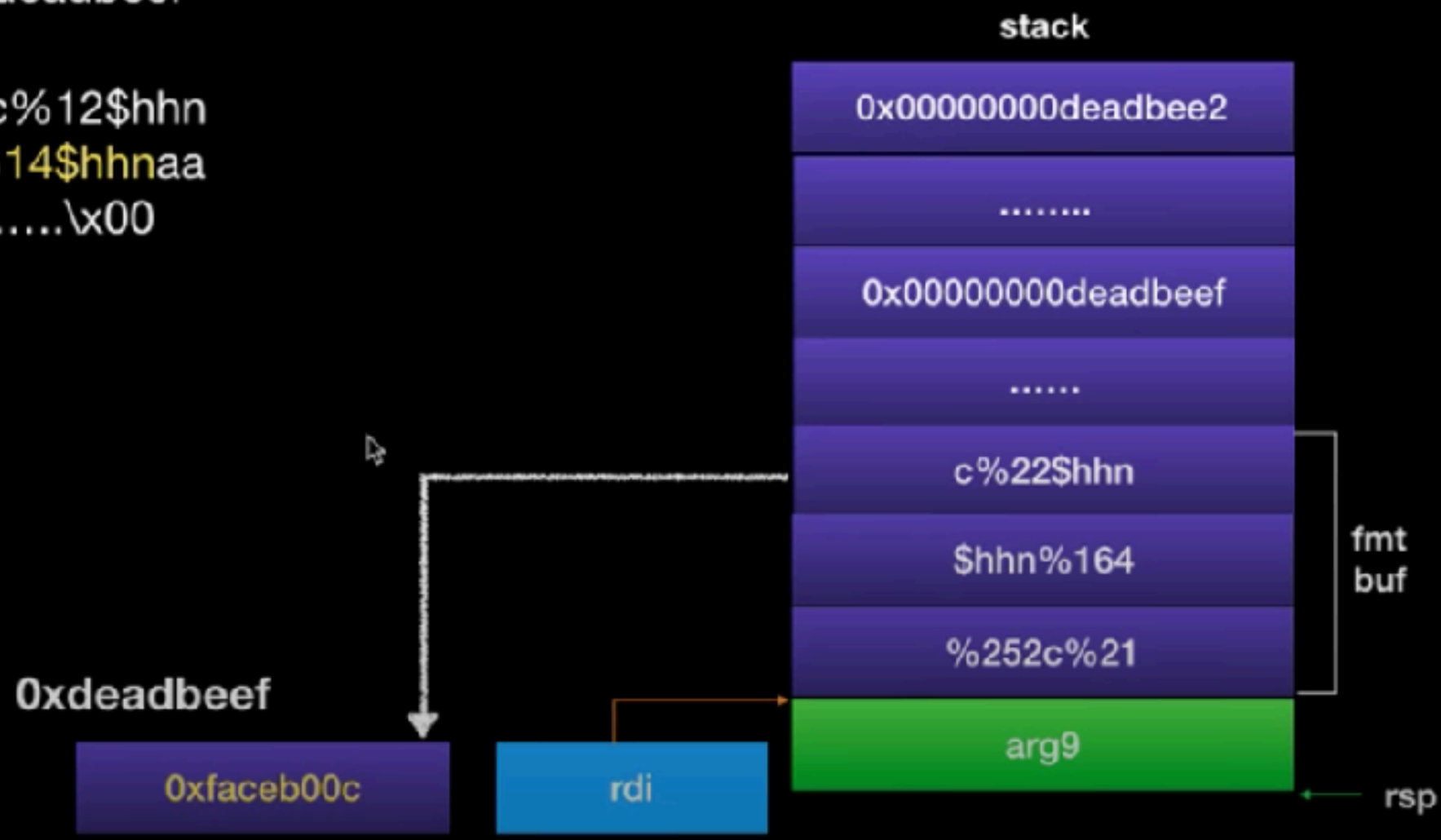
# Write to arbitrary memory

- Write 0xfaceb00c to 0xdeadbeef
- %252c%11$hhn%164c%12$hhn
  %30c%13$hhn%44c%14$hhnaa
  aa\xef\xbe\xad\xde.....\x00



# Write to arbitrary memory

- Write 0xfaceb00c to 0xdeadbeef
- %252c%11$hhn%164c%12$hhn
  %30c%13$hhn%44c%14$hhnaa
  aa\xef\xbe\xad\xde.....\x00



# Write to arbitrary memory

- 在 32 bit 下，把 address 放在最前面會較好定位參數的個數，但就要注意到 address 中沒有 NULL byte ，format string 只會到 NULL byte 就結束了
- 再有 NULL byte 的情況下，一樣可以利用上述的方式將 address 放後面，做好 padding 即可

# Outline

- Format String
- Read from arbitrary memory
- Write to arbitrary memory
- Advanced Trick

# Advanced Trick

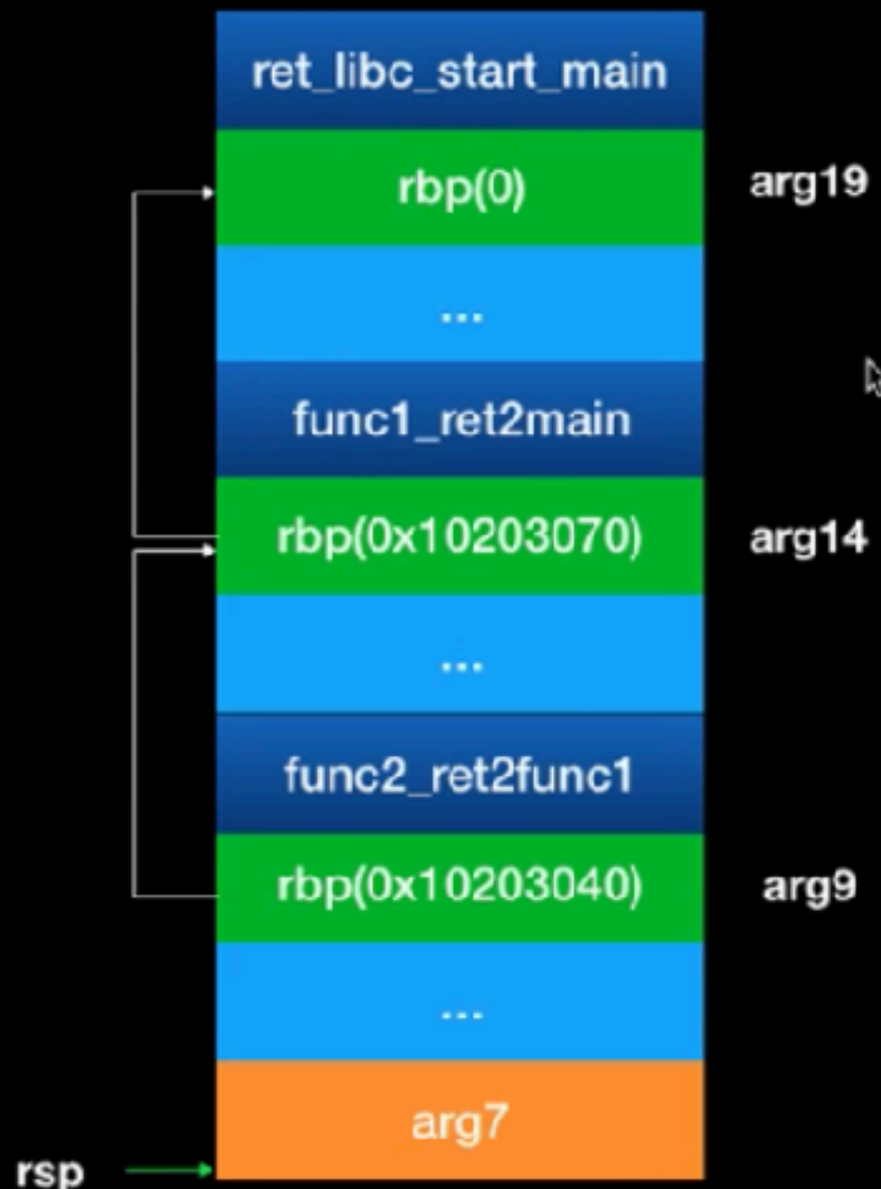
- 有時候並不會這麼剛好 format string 的 buf 也剛好在 stack 上
- 在可以多次 format string 情況下，我們可以用 stack 上現成的 pointer 進行寫值
  - RBP chain
  - argv chain

# Advanced Trick

- RBP Chain
- format string 的漏洞必須在 main function 下兩層中的 function 才可利用
- 利用第二層的 rbp 控制第一層的 rbp 這個 pointer ，再利用第一層的 rbp 來寫值

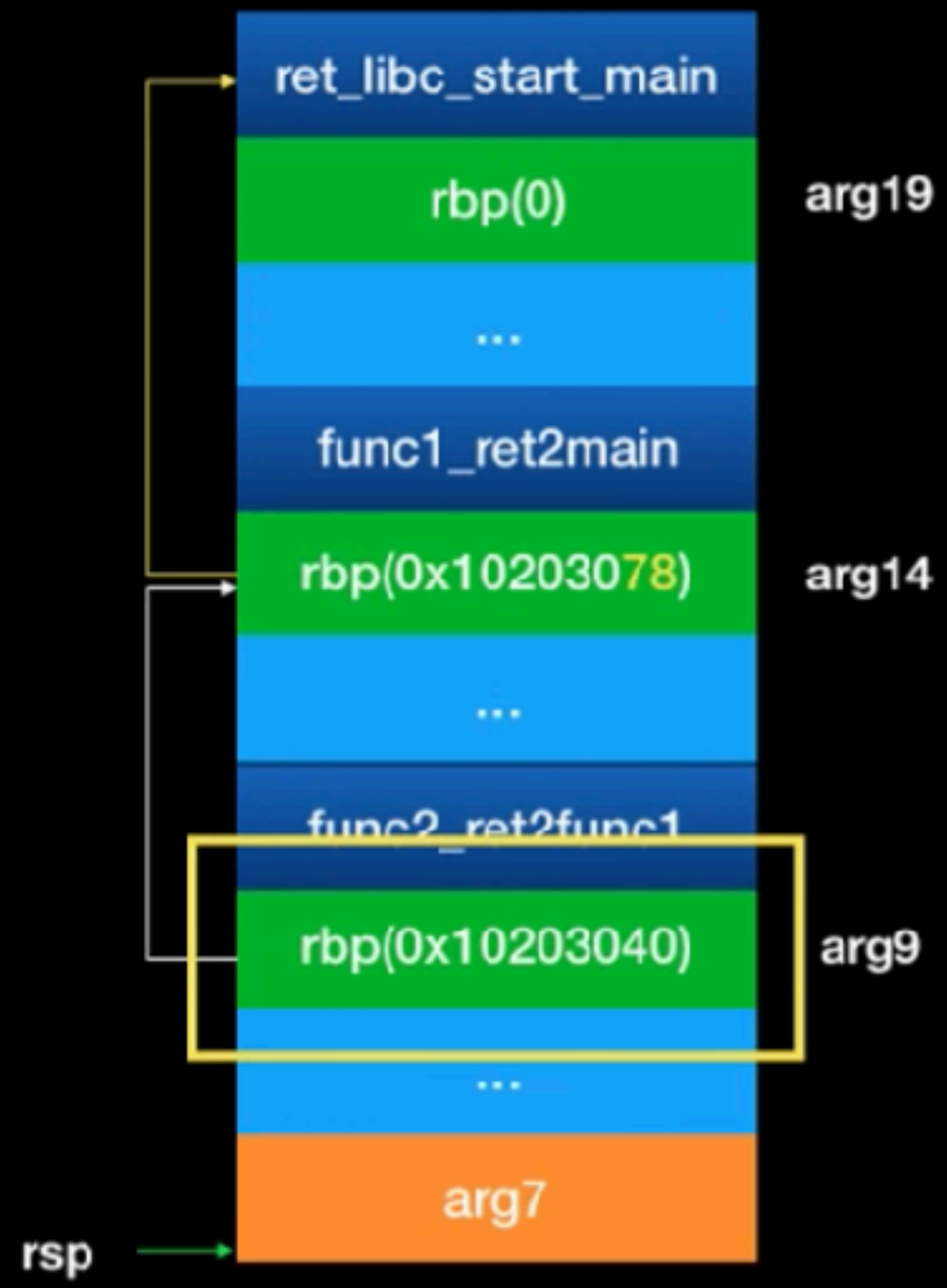
# Advanced Trick

- main->func1->func2>printf
- 在不 return 回 main 情況下



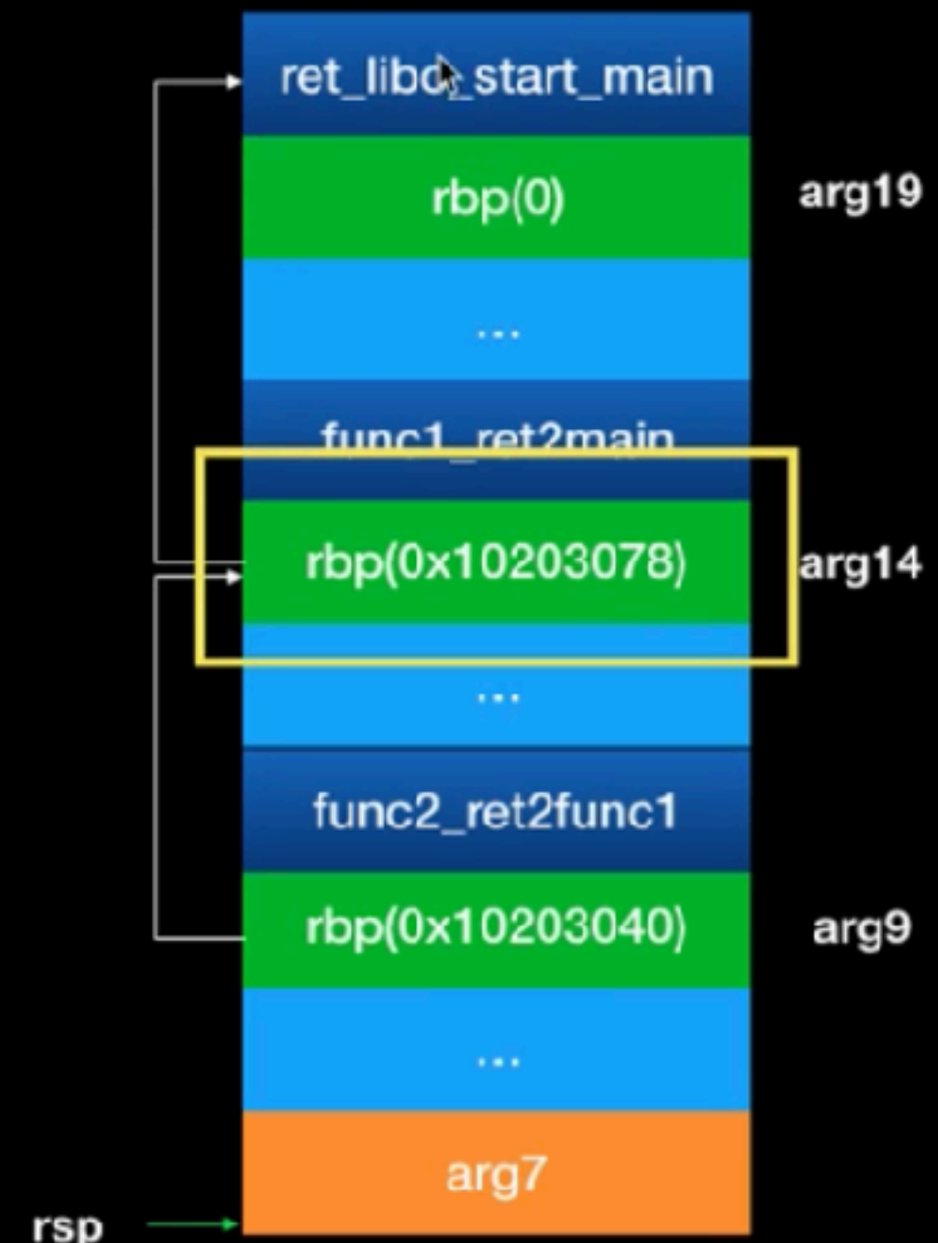
# Advanced Trick

- main->func1->func2>printf
- 在不 return 回 main 情況下
- 第一次先對第二層 rbp 寫值
- 此時會寫到第一層的 rbp
  並將第一層 rbp 往後移 8 byte
- %120c%8$hhn



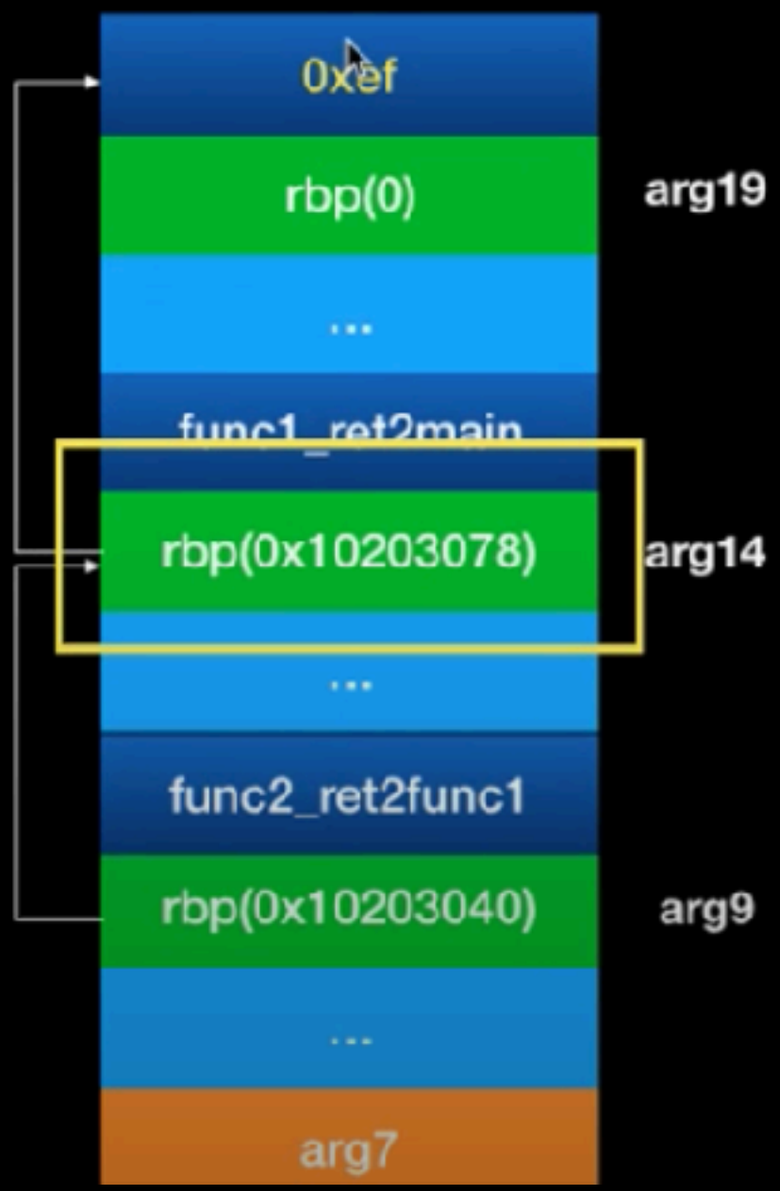
# Advanced Trick

- main->func1->func2>printf
- 在不 return 回 main 情況下
- 第一層 rbp 就會指向 main ret
- 此時利用剛剛更動 rbp 更改 return address
- %239c%13$hhn



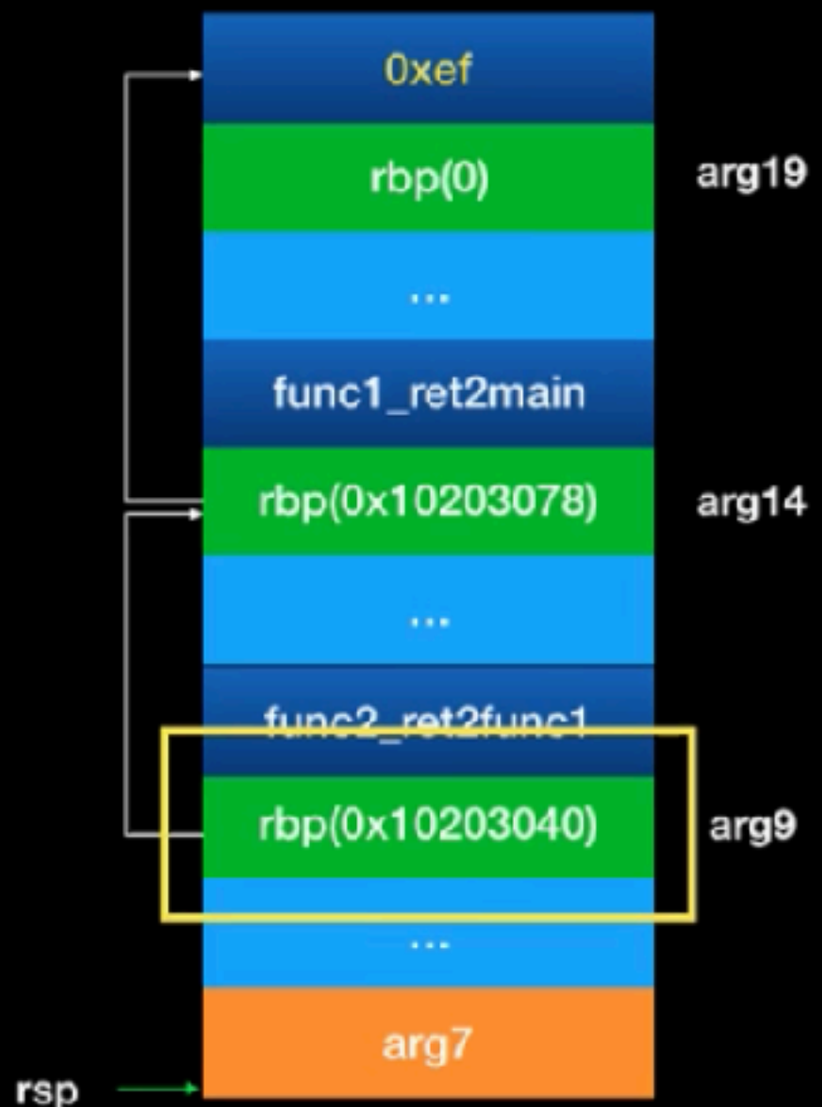
# Advanced Trick

- main->func1->func2>printf
- 在不 return 回 main 情況下
- 第一層 rbp 就會指向 main ret
- 此時利用剛剛更動 rbp 更改 return address
- %239c%13$hhn



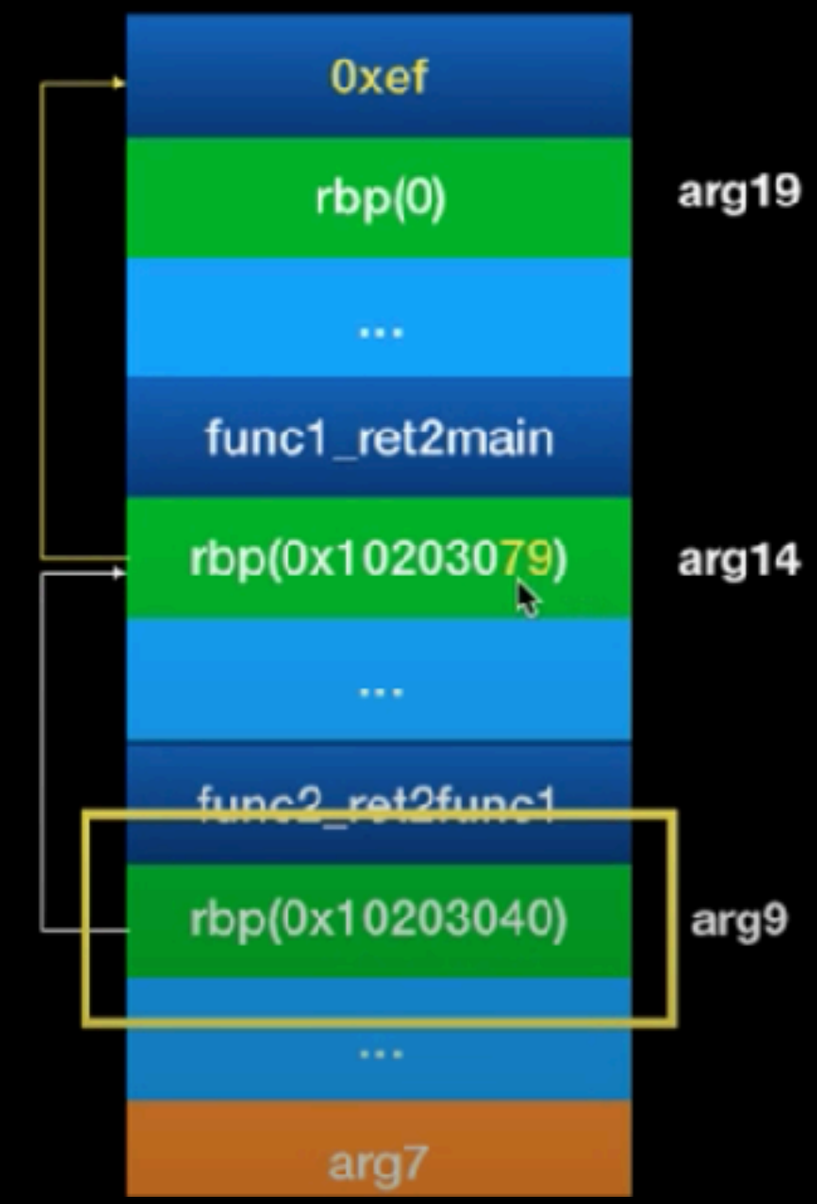
# Advanced Trick

- main->func1->func2>printf
- 在不 return 回 main 情況下
- 然後再回來更改第二層 rbp
  讓他指向下一個 byte



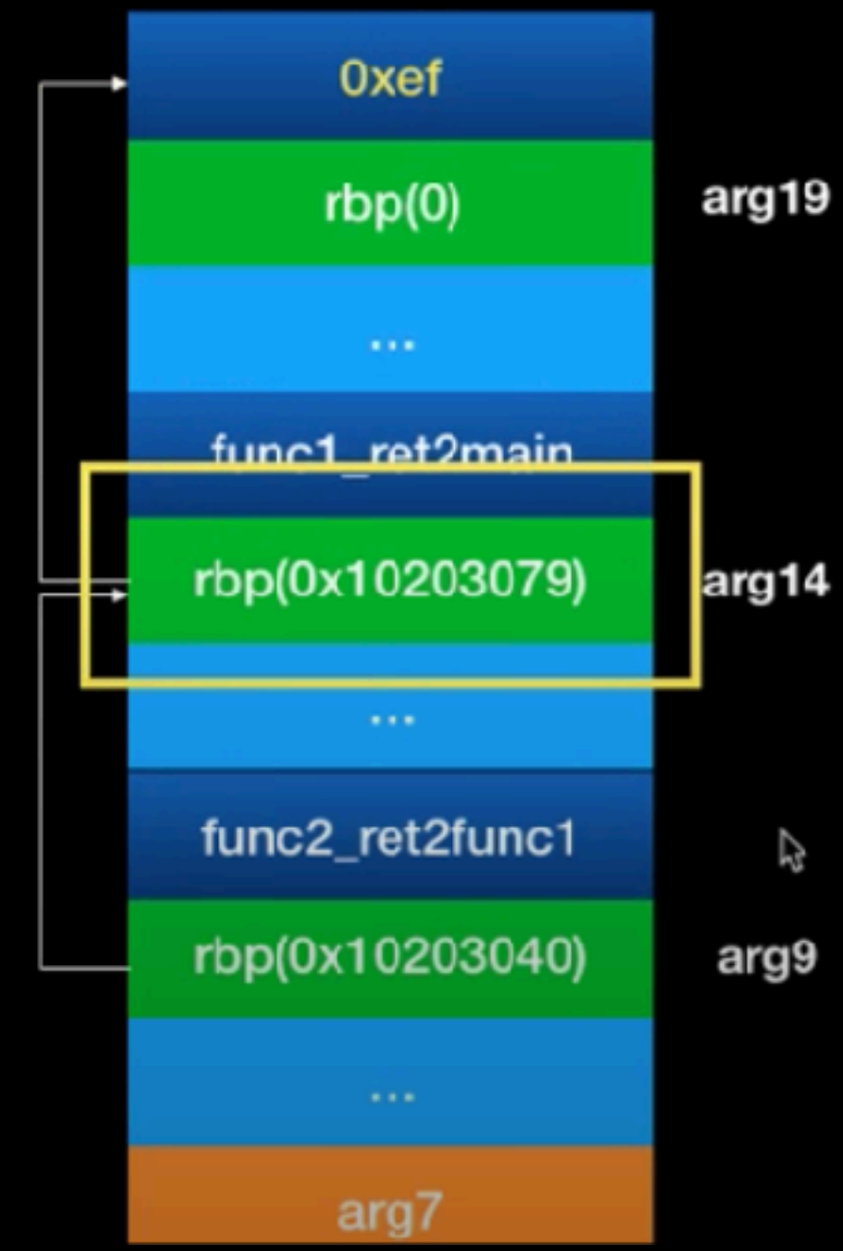
# Advanced Trick

- main->func1->func2>printf
- 在不 return 回 main 情況下
- 然後再回來更改第二層 rbp
  讓他指向下一個 byte
- %121c%8$hhn



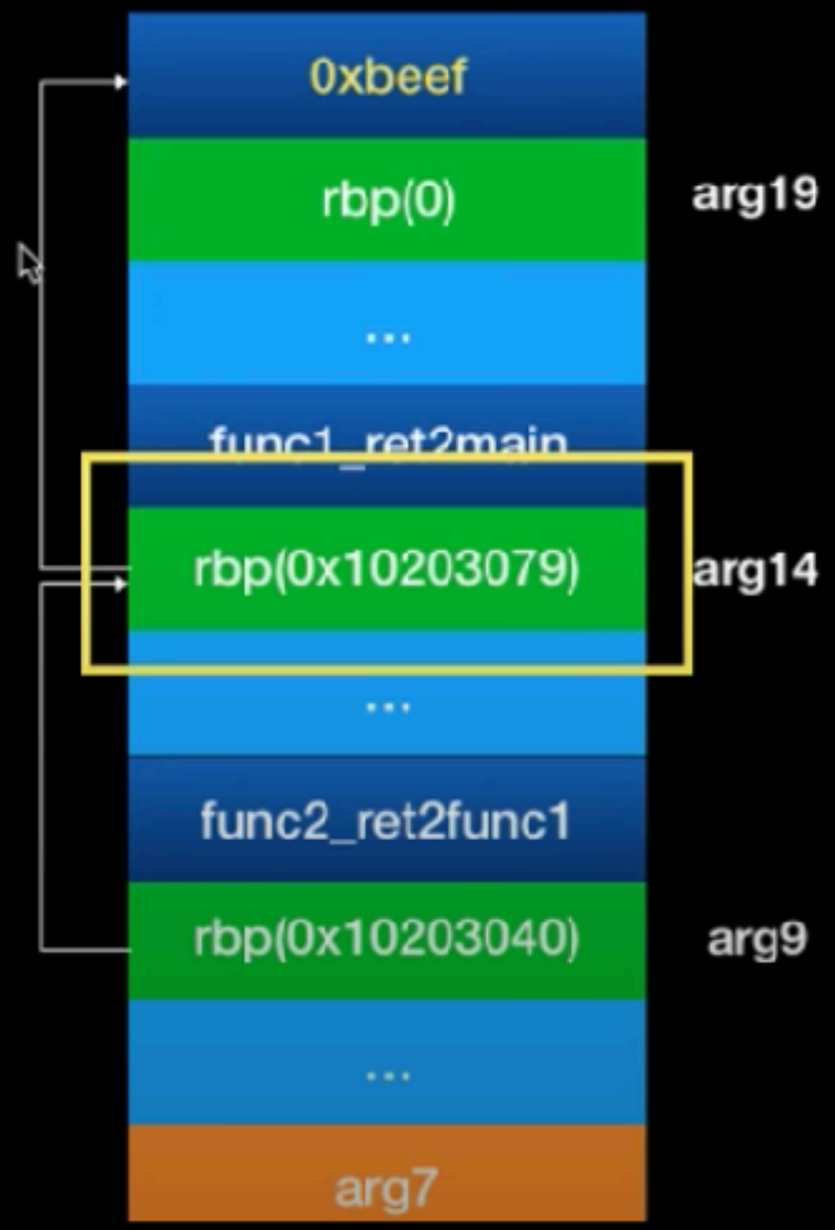
# Advanced Trick

- main->func1->func2>printf
- 在不 return 回 main 情況下
- 接著我們再次回去更改第一層 rbp
- %190c%13$hhn



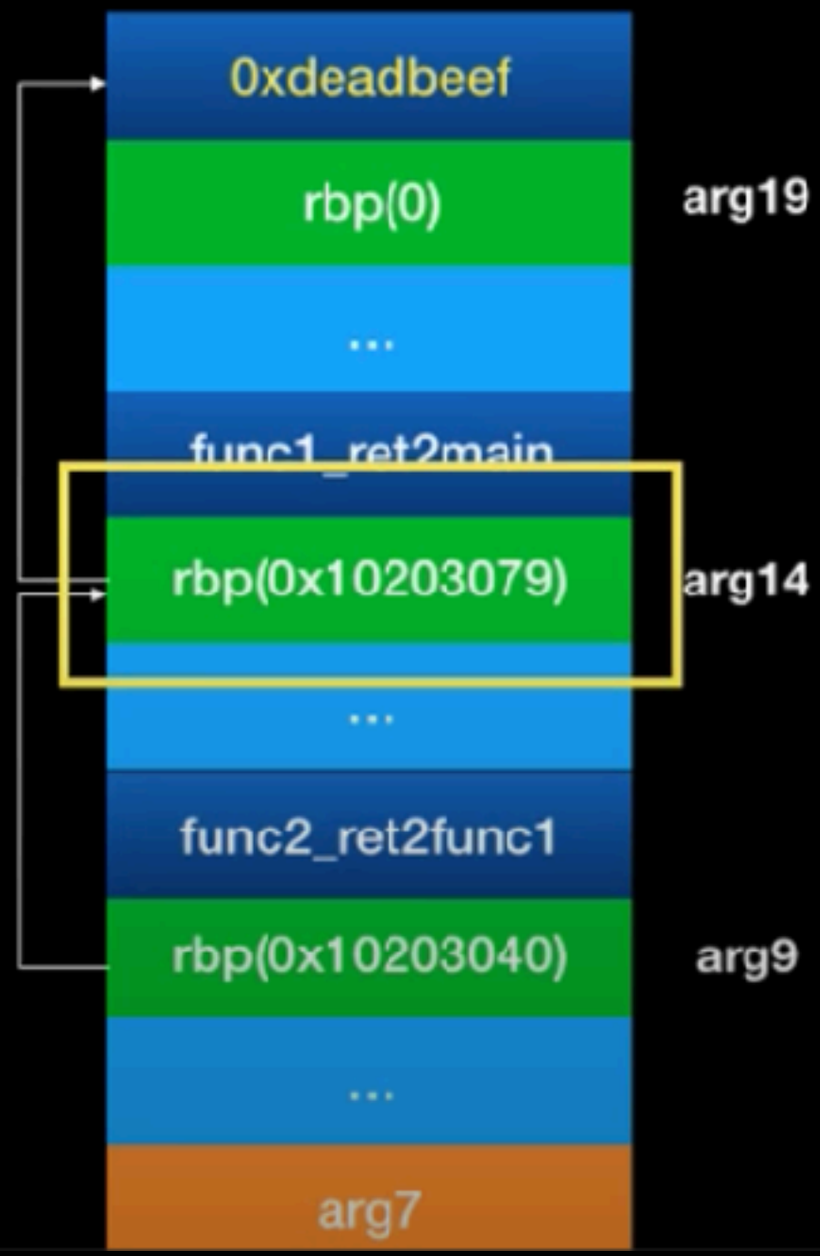
# Advanced Trick

- main->func1->func2>printf
- 在不 return 回 main 情況下
- 接著我們再次回去更改第一層 rbp
- %190c%13$hhn



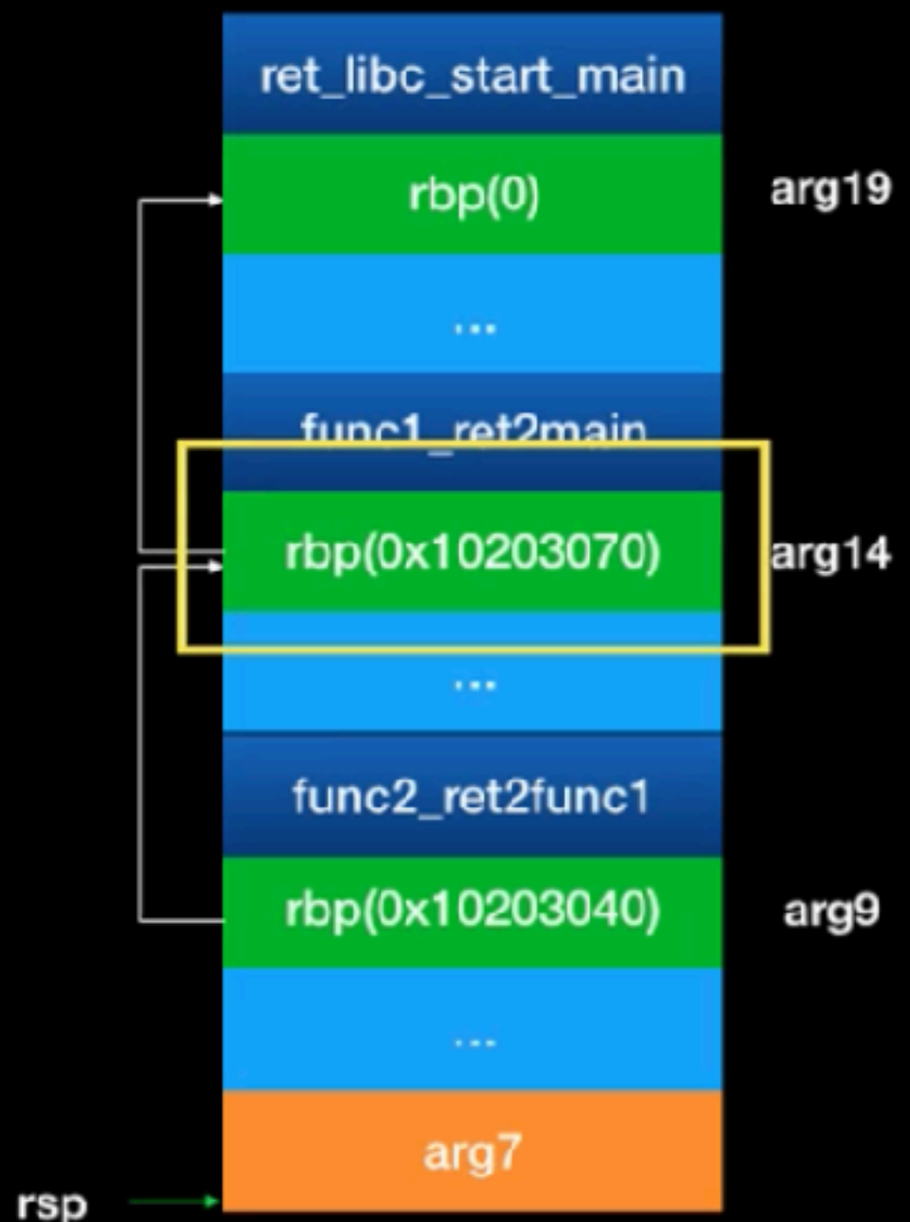
# Advanced Trick

- main->func1->func2>printf
- 在不 return 回 main 情況下
- 後面不斷重複一樣動作
  便可將整個 return address 改掉
- 最後在觸發 main return 就可
  - 記得要把 rbp 修回來，不然 stack frame 會跑掉



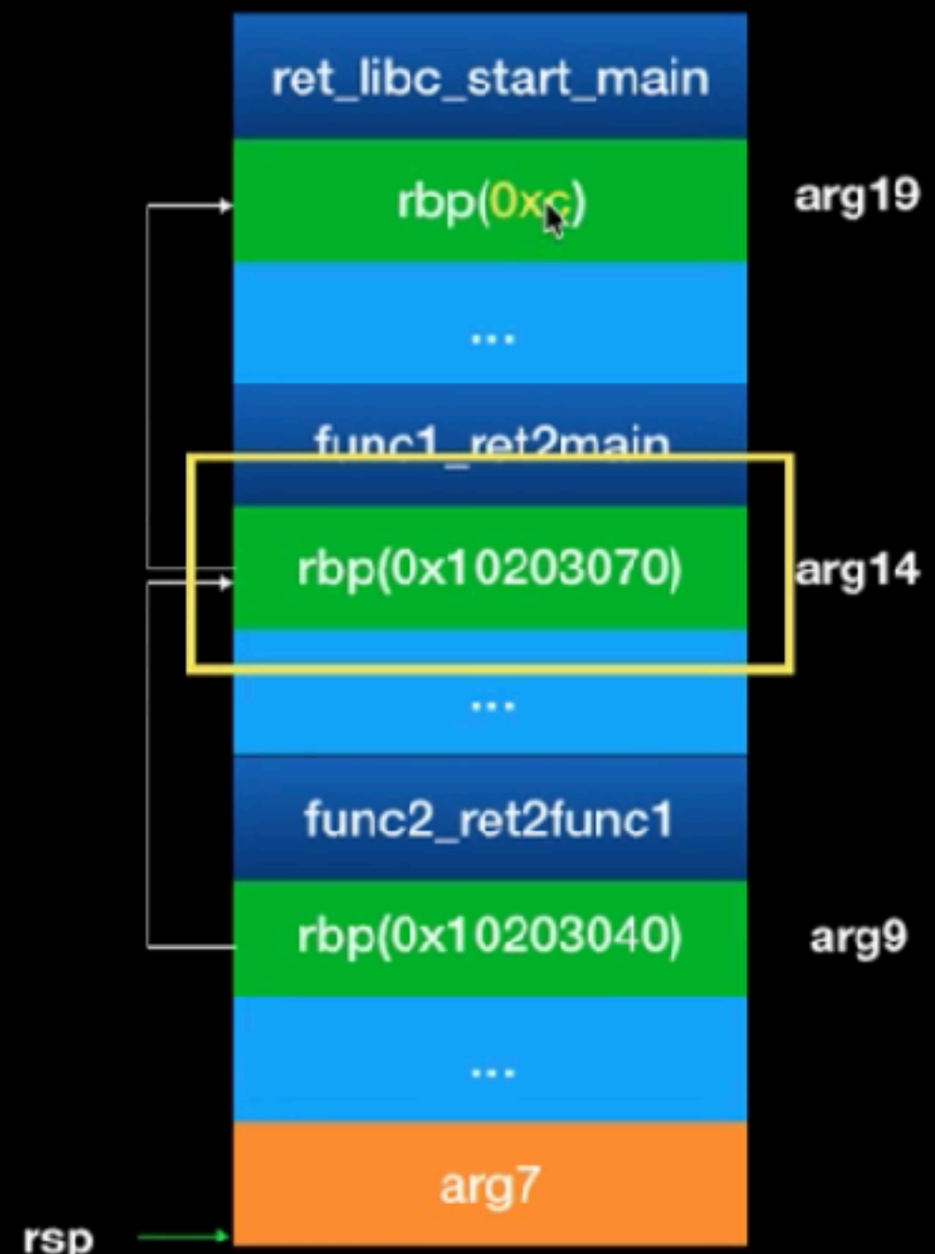
# Advanced Trick

- main->func1->func2>printf
- 我們也可利用第一層 rbp 製造 pointer 來做任意位置寫入
- %12c%13$hhn



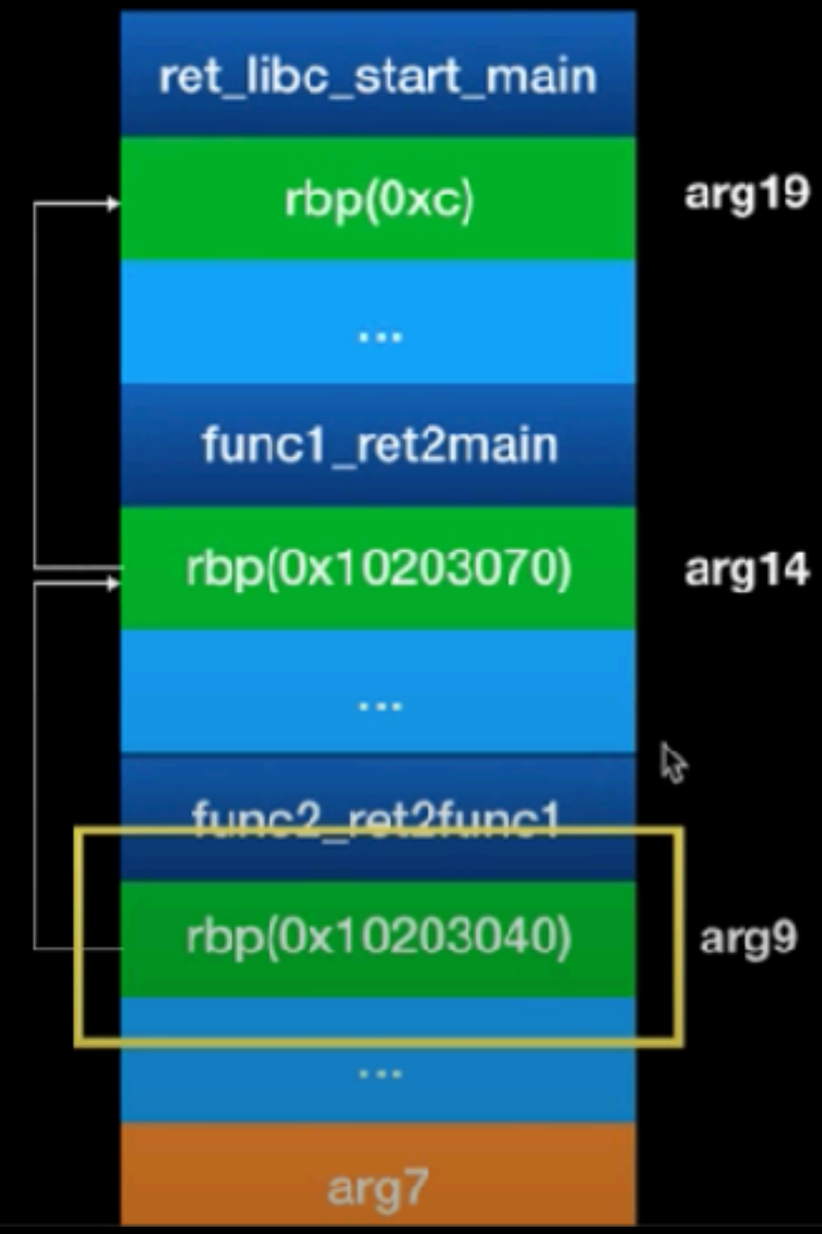
# Advanced Trick

- main->func1->func2>printf
- 我們也可利用第一層 rbp 製造 pointer 來做任意位置寫入
- %12c%13$hhn



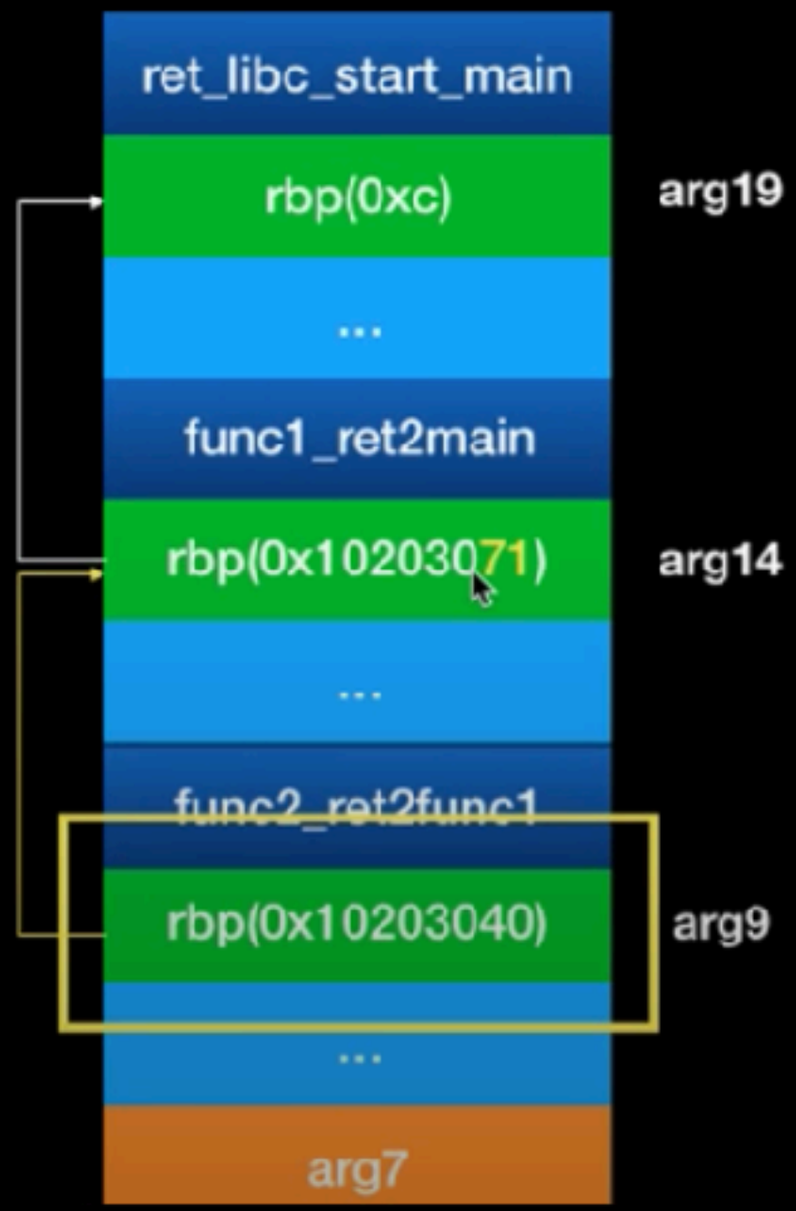
# Advanced Trick

- main->func1->func2>printf
- 一樣利用第二層 rbp 來更改第一層 rbp
- %113c%8$hhn



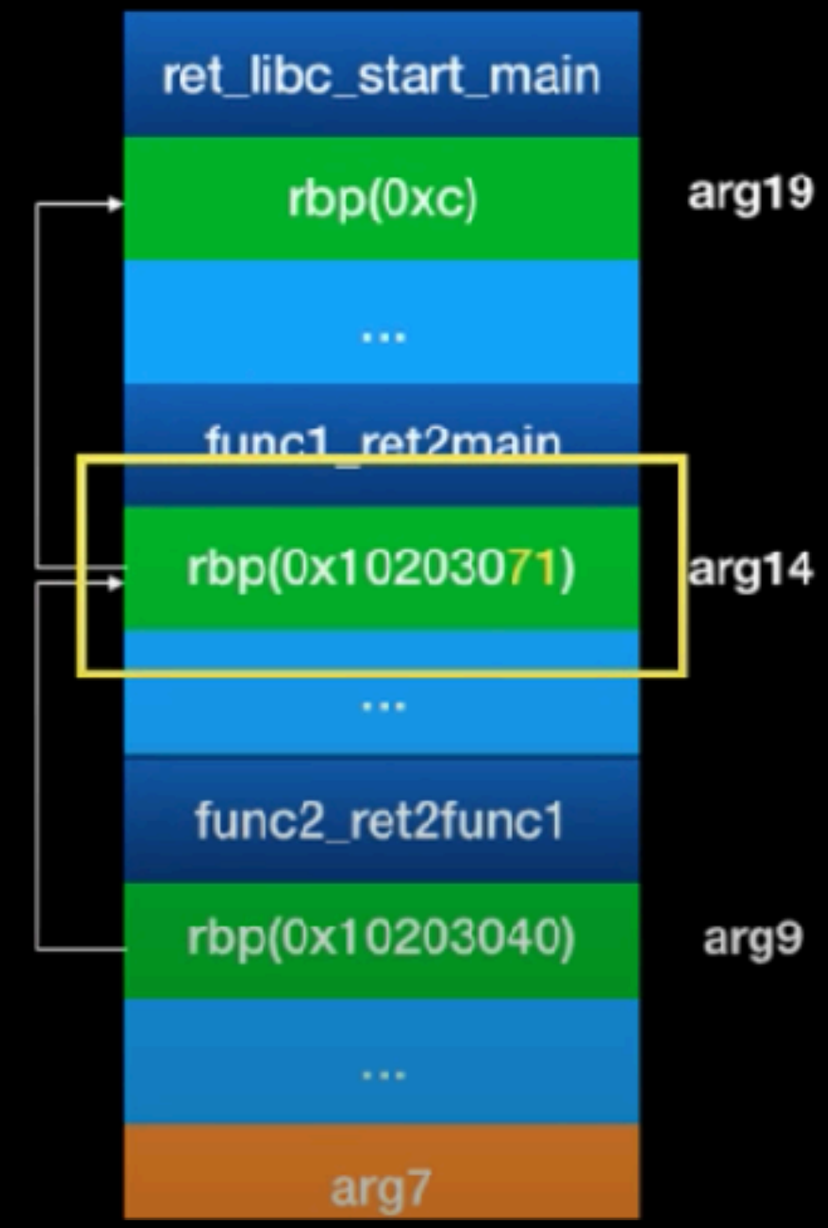
# Advanced Trick

- main->func1->func2>printf
- 一樣利用第二層 rbp 來更改第一層 rbp
- %113c%8$hhn



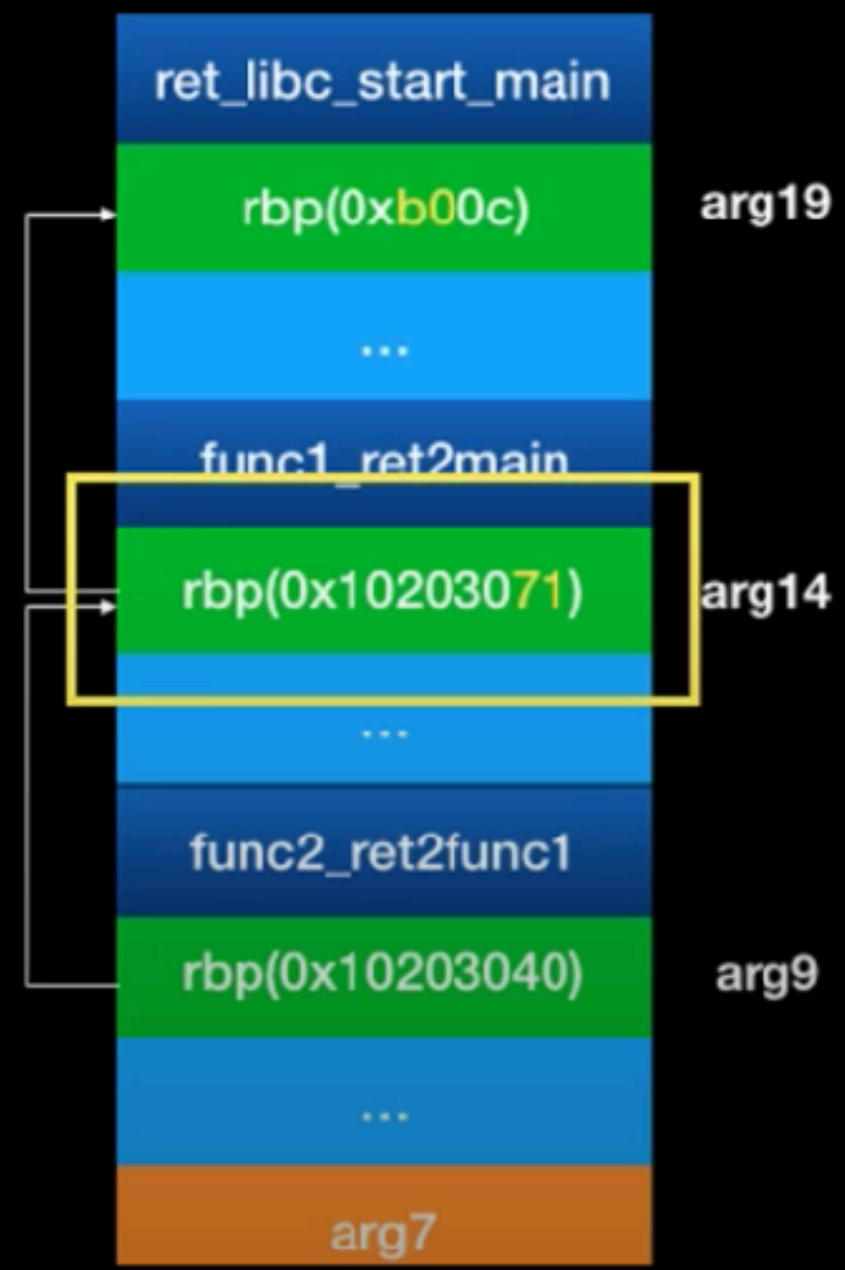
# Advanced Trick

- main->func1->func2>printf
- 再利用第一層繼續寫值
- %176c%13$hhn



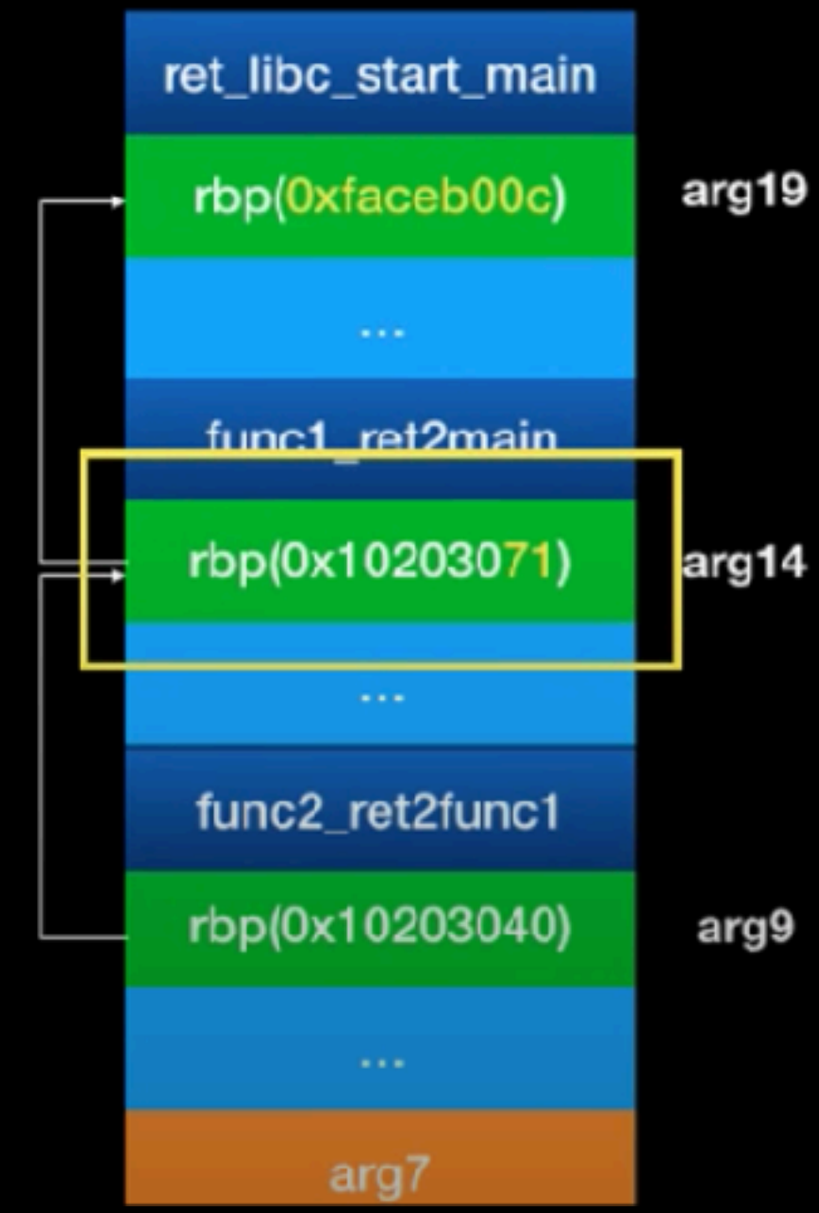
# Advanced Trick

- main->func1->func2>printf
- 再利用第一層繼續寫值
- %176c%13$hhn



# Advanced Trick

- main->func1->func2>printf
- 持續同樣的動作就可以
  將 0xfaceb00c 指標塞入 rbp
- 也就是我們可以針對 0xfaceb00c 寫值
- %xxc%18$hhn



# Advanced Trick

- Argv Chain
- 做法非常類似 rbp chain ，不過 argv 利用的是 main function 傳遞的 argv 來控制指標
- &argv->argv->argv[0]
- 但要注意 argv[0] 每次 offset 都不固定，需要先 leak 來確認參數位置