Binary Ninja 101

Windows Binary Reversing from Zero to Master

aaaddress1@chroot.org

aaaddress1 // hackingWeekend

馬聖豪（aaaddress1, aka adr）

TDOHacker 資安社群核心成員

Speaker

- TDOHConf 2016 議程組長
- HITCON CMT 2015
- SITCON 2016
- HITCON CMT 2016 閃電秀
- SITCON 2017
- iThome#Chatbot 2017
- BSidesLV
- ICNC
- MC2015
- 全國資安會議
- 資訊安全基礎技術工作坊

C/C++, C#, VB, MASM, Python, Swift, Node.js, Java

專研於 Windows 上平台特性與程式弱點與逆向工程分析

aaaddress1 // hackingWeekend

#murmur

aaaddress1 // hackingWeekend

寫過 C/C++ 的舉個手

aaaddress1 // hackingWeekend

自認會 C/C++ 的舉個手

aaaddress1 // hackingWeekend

智力測驗時間

```
{
  printf("hello Anjali");
 }
```

Output:

Anjali: "I have a boyfriend".

智力測驗時間

```
int arr[] = {
    0x6c6c6568,
    0x6f77206f,
    0x21646c72,
    0x00000000
};
printf("%p", arr[0]);
```

智力測驗時間

```
int arr[] = {
    0x6c6c6568,
    0x6f77206f,
    0x21646c72,
    0x00000000
};
int *ptr  = &arr[0];
printf("%p", *ptr);
```

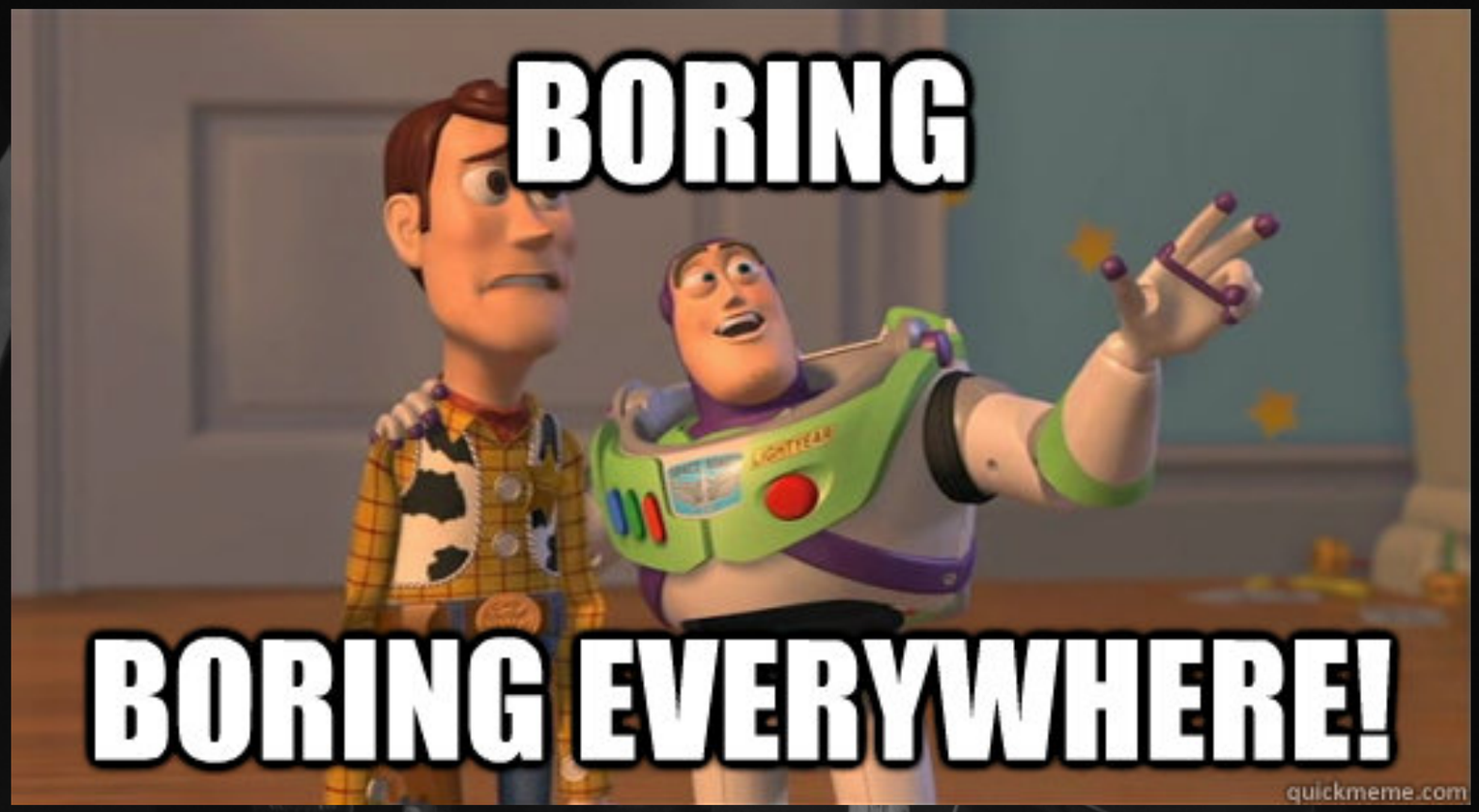
智力測驗時間

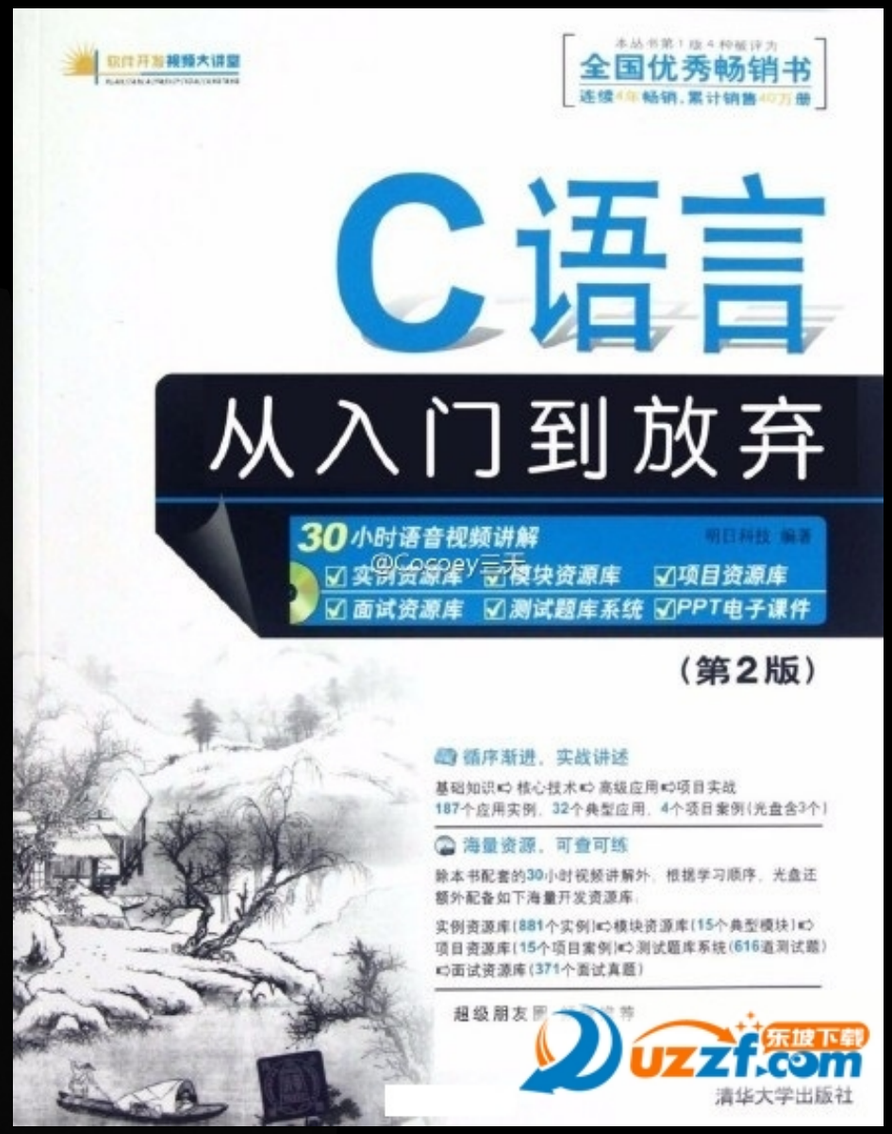
```
int arr[] = {
    0x6c6c6568,
    0x6f77206f,
    0x21646c72,
    0x00000000
};

printf("%s", &arr);
```

智力測驗時間

bash

```
aaaddress1 adr-pc ~ Desktop $ cat main.cpp
#include <stdio.h>

int main(void)
{
        int arr[] = {0x6c6c6568, 0x6f77206f, 0x21646c72, 0};
        printf("%s", &arr);
        return 0;
}
aaaddress1 adr-pc ~ Desktop $ g++ -O main.cpp -o main
main.cpp:6:15: warning: format specifies type 'char *' but the argument has
      type 'int (*)[4]' [-Wformat]
        printf("%s", &arr);
                ~~   ^~~~
1 warning generated.
aaaddress1 adr-pc ~ Desktop $ ./main
hello world! aaaddress1 adr-pc ~ Desktop $ _
```





有沒有特別痛恨資料結構跟作業系統的朋友，

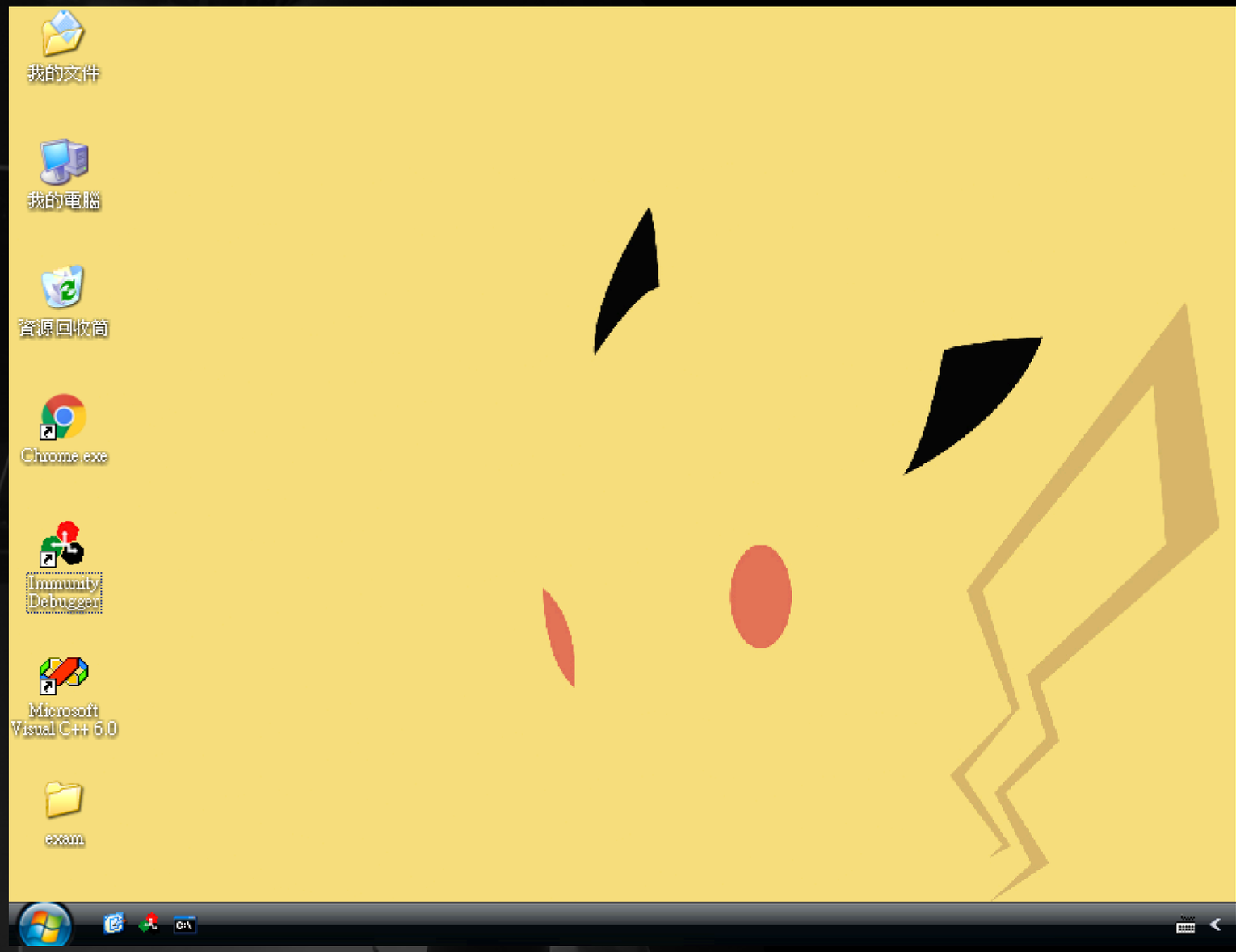
~~想離場的現在趕快喔，不然你會無聊八個小時~~

aaaddress1 // hackingWeekend

Outline

課程環境

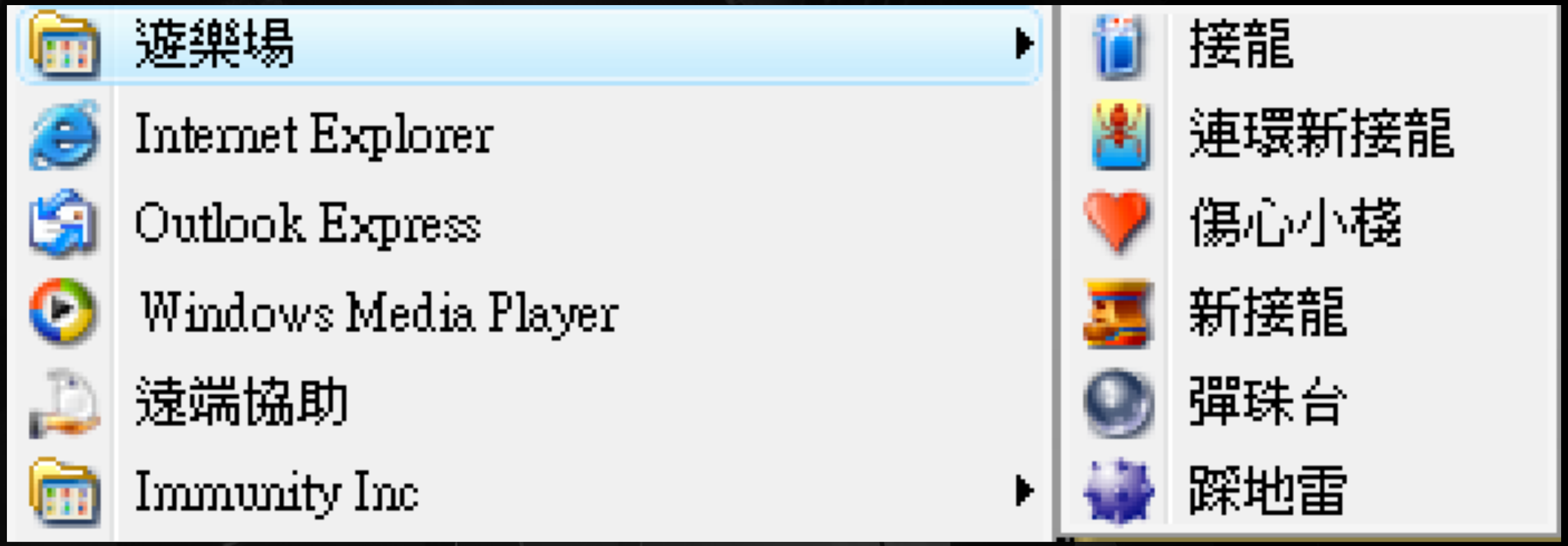
- Windows XP 逆向訓練平台虛擬機一份（*.ova）
- VC 6.0 編譯環境
- Immunity Debugger
- *.exe Binary 題目
- 題目原始碼



課程環境

除此之外虛擬機裡面還有你的童年回憶各式小遊戲，

請不要上課時打電動，謝謝。



逆向工程基礎篇章

1. 逆向工程基礎（一）：記憶體
2. 逆向工程基礎（二）：PE 架構
3. 逆向工程基礎（三）：機械碼
4. 逆向工程基礎（四）：組合語言
5. 逆向工程基礎（五）：Immunity Debugger

＊實驗環境：Windows XP x86 SP3

aaaddress1 // hackingWeekend

調試實戰分析篇章

1. 調試實戰分析 I: babyFirst
2. 調試實戰分析 II: eggHunter
3. 調試實戰分析 III: 1nte1's ATM Login System
4. 調試實戰分析 IV: cesarCipher

＊實驗環境：Windows XP x86 SP3

aaaddress1 // hackingWeekend

Bonus

1. Test v.s. Cmp?
2. main function: return 0; v.s. exit(0)?
3. Unpacker
4. PE Structure: IAT
5. Z3

＊實驗環境：Windows XP x86 SP3

aaaddress1 // hackingWeekend

逆向工程基礎篇章

aaaddress1 // hackingWeekend

逆向工程基礎（一）

記憶體

為什麼要逆向工程

- 你想知道某個程式怎麼運作的
- 寫線上外掛、機器人
- 破解付費驗證、撰寫註冊機
- 挖掘程式漏洞
- 你朋友的程式碼太醜，還不如看組合語言的時候...

編譯/ 中介碼/ 直譯

- 編譯為原生機械碼（Native）
  C、C++、Objective-C、Swfit、Go

- 編譯為中介碼（Intermediate Language, aka IL）
  .NET（C#, VB）、Java、Python（pyc）

- 直譯（interpreting）
  Python、Ruby、Javascript、VBScript

編譯/ 中介碼/ 直譯

- 編譯為原生機械碼（Native）
  C、C++、Objective-C、Swfit、Go

編譯/ 中介碼/ 直譯

- 編譯為原生機械碼（Native）
  C、C++、Objective-C、Swfit、Go

從逆向工程上只能理解大致上運作的過程，
並無法真正取得開發者的原始碼

記憶體

- 記憶體由一個個格子組成，每個格子大小為 1 byte
- 1 byte 可儲存 0 ~ 255 的數值（也就是 0x00 ~ 0xFF）
- 每個格子都有對應的記憶體地址編號

0x11 被存放在 0x100 的格子上

addr	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
100	11															
110																
120																

記憶體

- 記憶體由一個個格子組成，每個格子大小為 1 byte
- 每個格子都有對應的記憶體地址編號

0x22 被存放在 0x10F 的格子上

addr	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
100	11															22
110																
120																

記憶體

- 記憶體由一個個格子組成，每個格子大小為 1 byte
- 每個格子都有對應的記憶體地址編號

0x33 被存放在 0x123 的格子上

addr	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
100	11															22
110																
120				33												

記憶體 (int)

```
printf("size of int = %i\n", sizeof(int));

// size of int = 4
```

addr	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
100																
110																
120																

記憶體 (int)

```
int* ptr = (int *)(0x116);

printf("data = %x\n", *ptr);

// data = 0xDEADBEEF
```

addr	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
100																
110							EF	BE	AD	DE						
120																

記憶體 (int)

```
int* ptr = (int *)(0x116);

printf("data = %x\n", *ptr);

// data = 0xDEADBEEF
```

little-endian

addr	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
100																
110							EF	BE	AD	DE						
120																

記憶體 (int) 問題

```
int* ptr = (int *)(0x117);

*ptr = 0xDEADBEEF;
```

addr	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
100																
110							EF	BE	AD	DE						
120																

記憶體 (int) 問題

```
int* ptr = (int *)(0x117);

*ptr = 0xDEADBEEF;
```

addr	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
100																
110							EF	EF	BE	AD	DE					
120																

記憶體 (array)

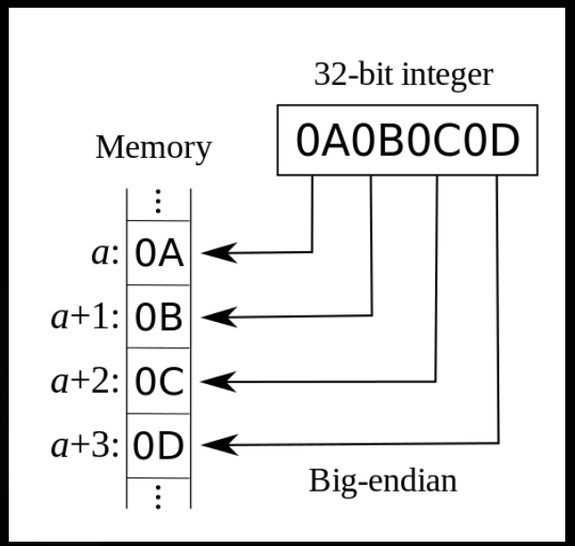
```
int arr[3] = {

0xA1A2A3A4, 0xB4B3B2B1, 0xCCCCCCCC

};
```

addr	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
100																
110							A4	A3	A2	A1	B1	B2	B3	B4	CC	CC
120	CC	CC														

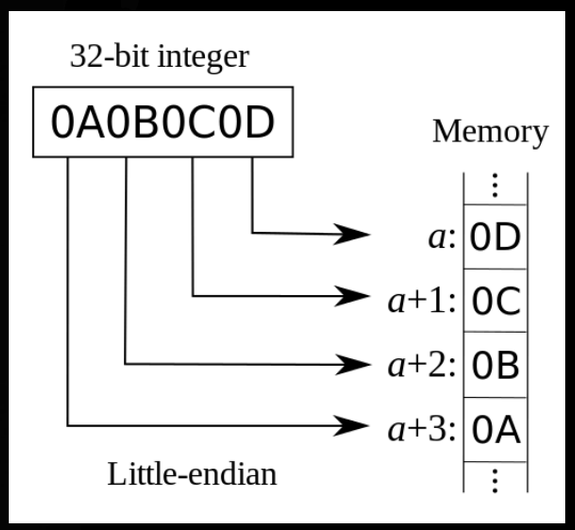
位元組順序

以數值 0xAABBCCDD 為例

- 大端序（big-endian）
  記憶體中擺放: AA BB CC DD



- 小端序（little-endian）
  記憶體中擺放: DD CC BB AA



https://zh.wikipedia.org/wiki/字节序

逆向工程基礎（二）

PE 架構

Portable Executable

Portable Exécutable (PE) 為 Windows 作業系統針對：執行檔（*.exe）、函數庫（*.dll）與驅動程式（*.sys）內，如何存放機械碼所設計的一種結構

例如 macOS 作業系統上採用 mach-O、Linux 上 ELF 等結構

Portable Executable

當一支 *.exe 執行文件被點擊時，
會建立為一個 Process（處理序），內保存 *.exe 的機械碼

Process																
100																
110																
120																
130																
140																
150																
160																
...																

Portable Executable

不過作業系統不會真的直接開一大片記憶體保存 *.exe 的內容

Process																
100	A	A	A	A	A	A	.	.	.							
110																
120																
130																
140																
150																
160																
...	B	B	B	B	B	B	B	B	B	B	B	B	B	B	B	B

Portable Executable

不知道你們小時候有沒有跟我一樣白目，
用解壓縮工具打開過 *.exe 檔案，你會意外發現大秘寶

hello.exe 2017/6/25 上午 0... 應用程式 28 KB

C:\Users\aaaddress1\Desktop\hello_world\hello\Release\hello.exe\

檔案(F) 編輯(E) 檢視(V) 我的最愛(A) 工具(T) 說明(H)

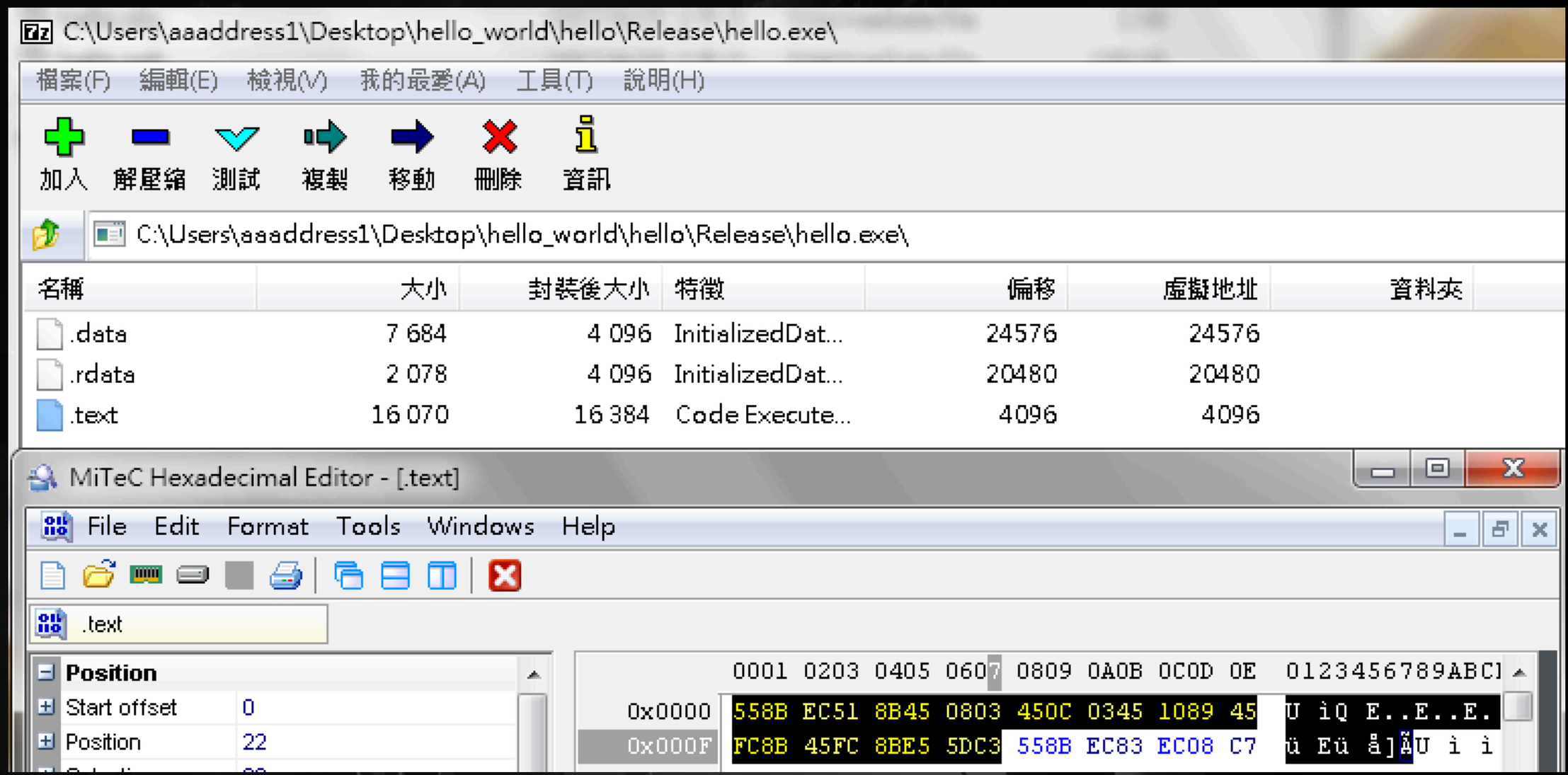
加入 解壓縮 測試 複製 移動 刪除 資訊

C:\Users\aaaddress1\Desktop\hello_world\hello\Release\hello.exe\

名稱	大小	封裝後大小	特徵	偏移	虛擬地址
.data	7 684	4 096	InitializedDat...	24576	24576
.rdata	2 078	4 096	InitializedDat...	20480	20480
.text	16 070	16 384	Code Execute...	4096	4096

Portable Executable

把 .text 解壓縮出來，用 HexEditor 之類的工具可以看見十六進位的資料內容，拷貝前幾個十六進位位元組出來



Portable Executable

拿去一些線上 disassembler 反查 opcode 就可以發現
其實它是一個組合語言構成的 C/C++ 函數

Array Literal:

```
{ 0x55, 0x8B, 0xEC, 0x51, 0x8B, 0x45, 0x08, 0x03, 0x45, 0x0C, 0x03, 0x45, 0x10,
0x89, 0x45, 0xFC, 0x8B, 0x45, 0xFC, 0x8B, 0xE5, 0x5D, 0xC3 }
```

Disassembly:

```
0:   55                      push   ebp
1:   8b ec                   mov    ebp,esp
3:   51                      push   ecx
4:   8b 45 08                mov    eax,DWORD PTR [ebp+0x8]
7:   03 45 0c                add    eax,DWORD PTR [ebp+0xc]
a:   03 45 10                add    eax,DWORD PTR [ebp+0x10]
d:   89 45 fc                mov    DWORD PTR [ebp-0x4],eax
10:  8b 45 fc                mov    eax,DWORD PTR [ebp-0x4]
13:  8b e5                   mov    esp,ebp
15:  5d                      pop    ebp
16:  c3                      ret
```

Portable Executable

www.csn.ul.ie/~caolan/pub/winresdump/winresdump/doc/pefile2.html

	0001	0203	0405	0607	0809	0A0B	0C0D	0E
0x0000	558B	EC51	8B45	0803	450C	0345	1089	45
0x000F	FC8B	45FC	8BE5	5DC3	558B	EC83	EC08	C7
0x001E	45F8	0000	0000	C745	FC00	0000	00C7	45
0x002D	F801	0000	00EB	098B	45F8	83C0	0189	45
0x003C	F88B	4DF8	3B4D	087F	0B8B	55FC	0355	F8
0x004B	8955	FCEB	E48B	45FC	5068	3060	4000	E8
0x005A	6F00	0000	83C4	088B	E55D	C355	8BEC	51
0x0069	6A03	6A02	6A01	E88C	FFFF	FF83	C40C	50
0x0078	E89A	FFFF	FF83	C404	A144	6040	0083	E8
0x0087	01A3	4460	4000	833D	4460	4000	007C	21
0x0096	8B0D	4060	4000	0FBE	1181	E2FF	0000	00
0x00A5	8955	FCA1	4060	4000	83C0	01A3	4060	40
0x00B4	00EB	1068	4060	4000	E83D	0000	0083	C4
0x00C3	0489	45FC	33C0	8BE5	5DC3	5356	BE60	60
0x00D2	4000	5756	E8DD	0200	008B	F88D	4424	18
0x00E1	50FF	7424	1856	E896	0300	0056	578B	D8
0x00F0	E850	0300	0083	C418	8BC3	5F5E	5BC3	56
0x00FF	8B74	2408	8B46	0CA8	830F	84C4	0000	00
0x010E	A840	0F85	BC00	0000	A802	740A	0C20	89
0x011D	460C	E9AE	0000	000C	0166	A90C	0189	46
0x012C	0C75	0956	E8F8	0E00	0059	EB05	8B46	08
0x013B	8906	FF76	18FF	7608	FF76	10E8	EC0C	00
0x014A	0083	C40C	8946	0485	C074	6C83	F8FF	74
0x0159	678B	560C	F6C2	8275	348B	4E10	5783	F9
0x0168	FF74	148B	F9C1	FF05	83E1	1F8B	3CBD	E0
0x0177	6C40	008D	3CCF	EB05	BFD0	6240	008A	4F
0x0186	045F	80E1	8280	F982	7506	80CE	2089	56
0x0195	0C81	7E18	0002	0000	7514	8B4E	0CF6	C1
0x01A4	0874	0CF6	C504	7507	C746	1800	1000	00
0x01B3	8B0E	4889	4604	0FB6	0141	890E	5EC3	F7

.text 是用於儲存組合語言機械碼的區段，
整個 .text 塞滿大量的組合語言構成的函數

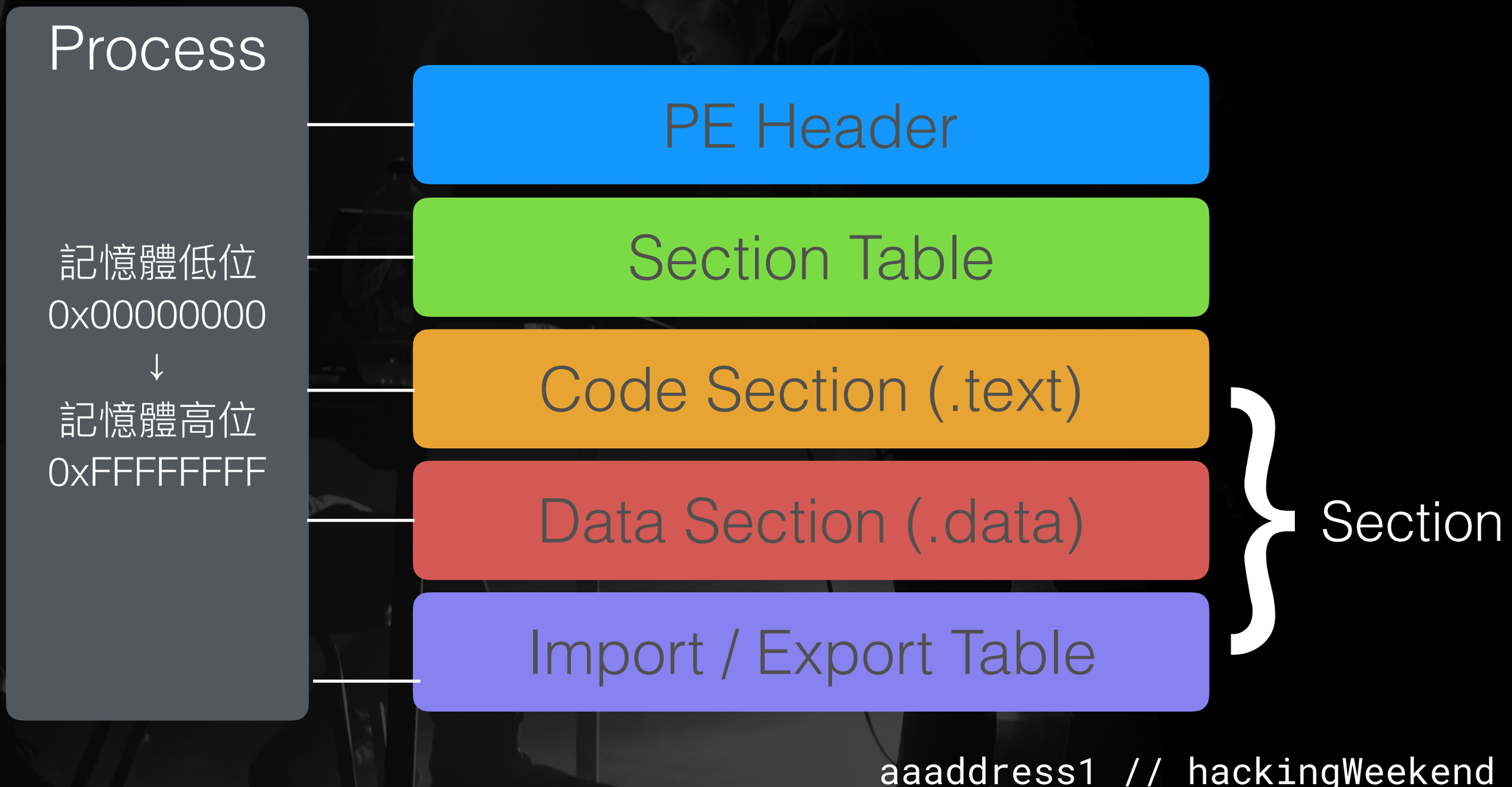
當然區段不止 .text 這一種，
還有比如：

.data 用於存放全域變數的區段
.rdata 存放唯讀的全域變數、常數區段
.bss 存放一些未初始化的變數資訊

Portable Executable

作業系統會根據不同區段（Section）申請記憶體空間來存放

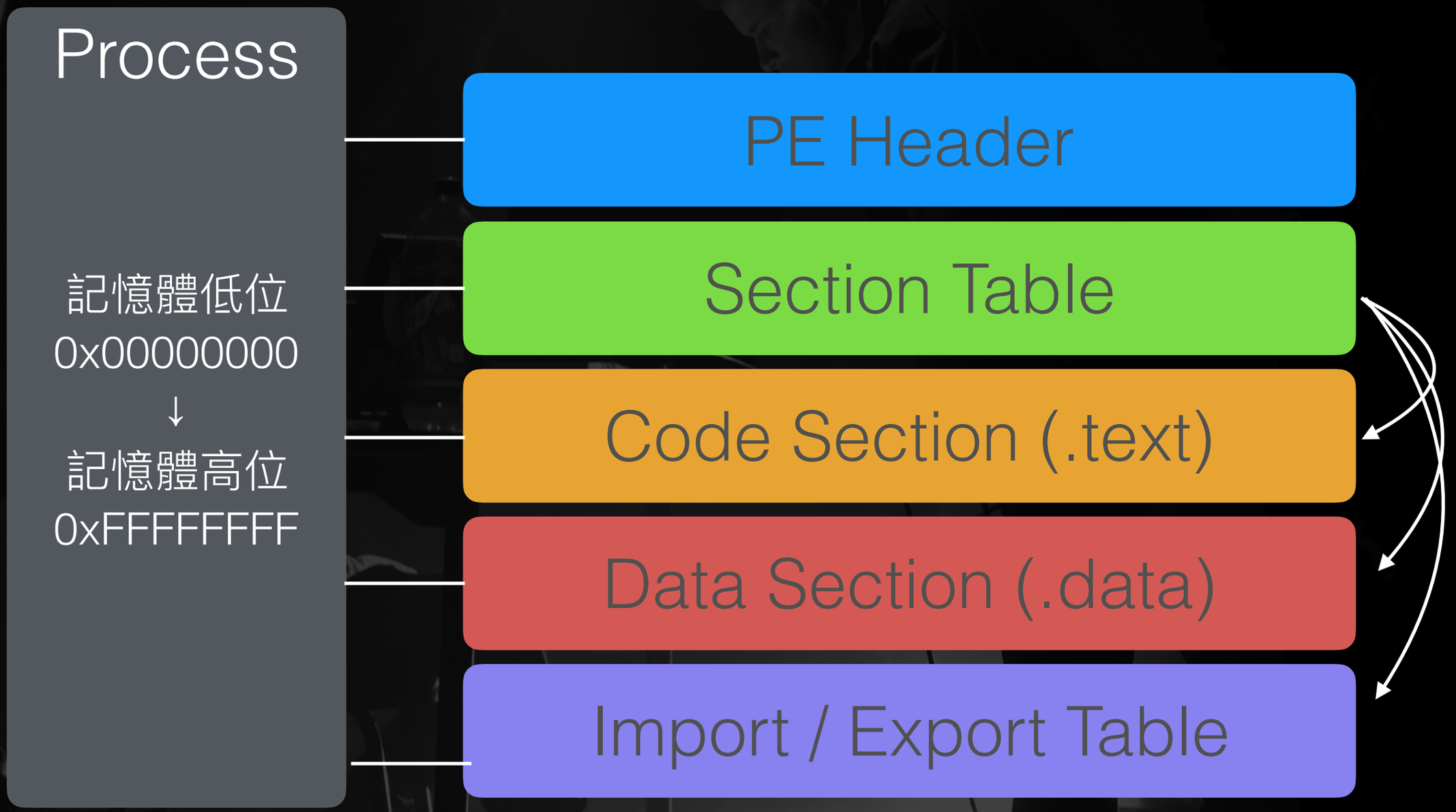
＊當然這是理想上啦...實務上考量效能之類的可能有差異



Portable Executable

作業系統會根據不同區段（Section）申請記憶體空間來存放

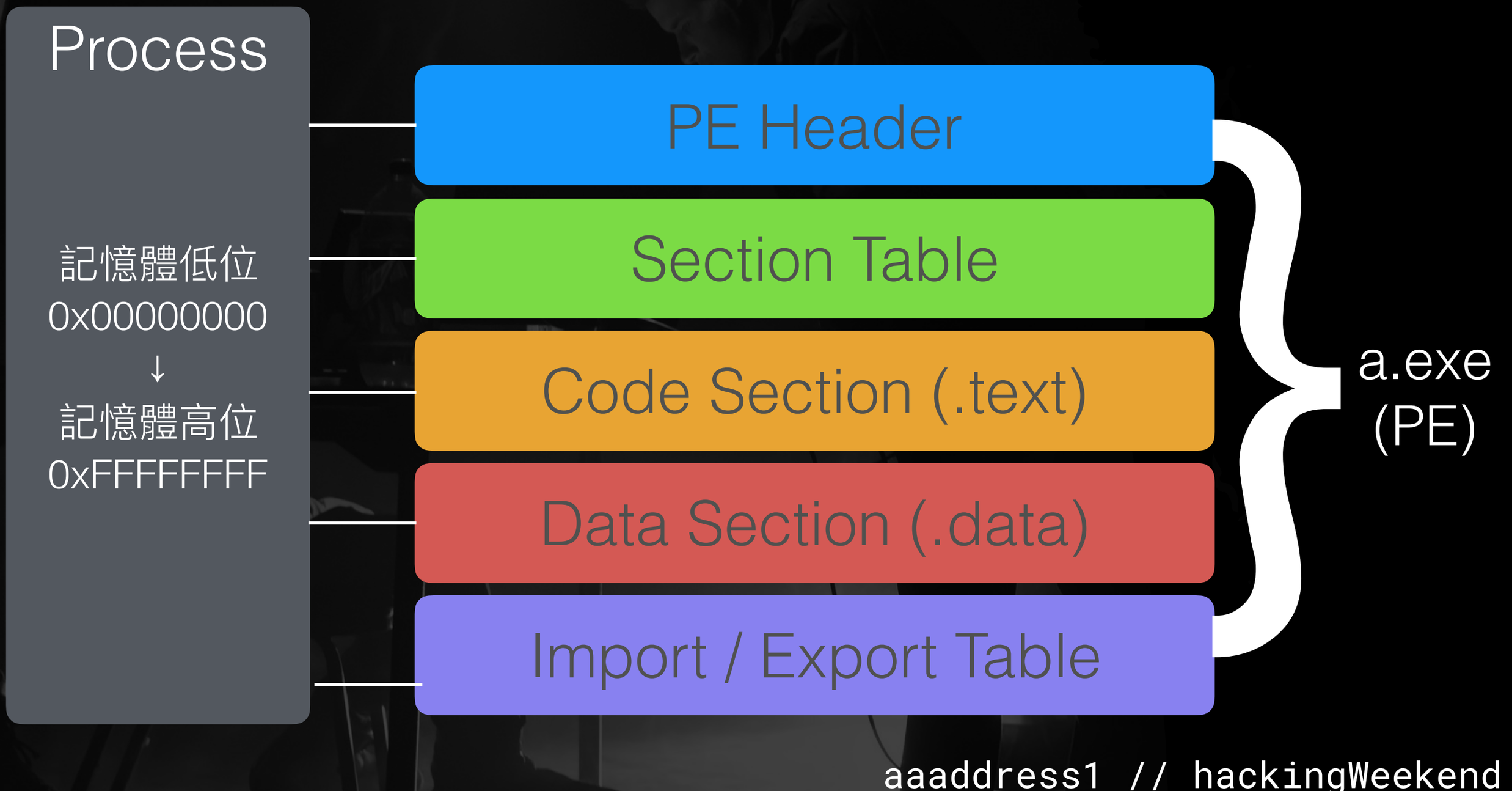
＊當然這是理想上啦...實務上考量效能之類的可能有差異



Portable Executable

作業系統會根據不同區段（Section）申請記憶體空間來存放

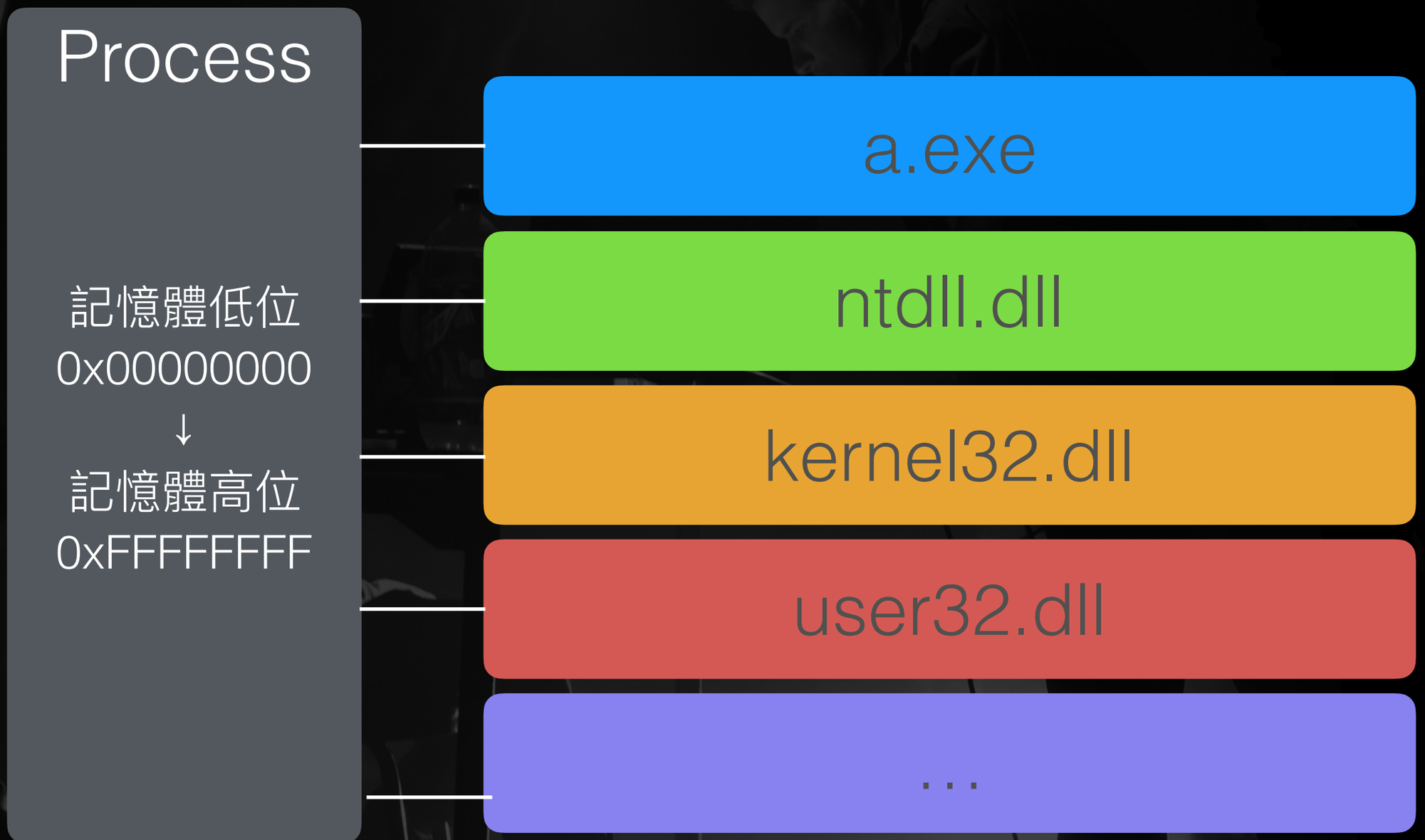
＊當然這是理想上啦...實務上考量效能之類的可能有差異



Portable Executable

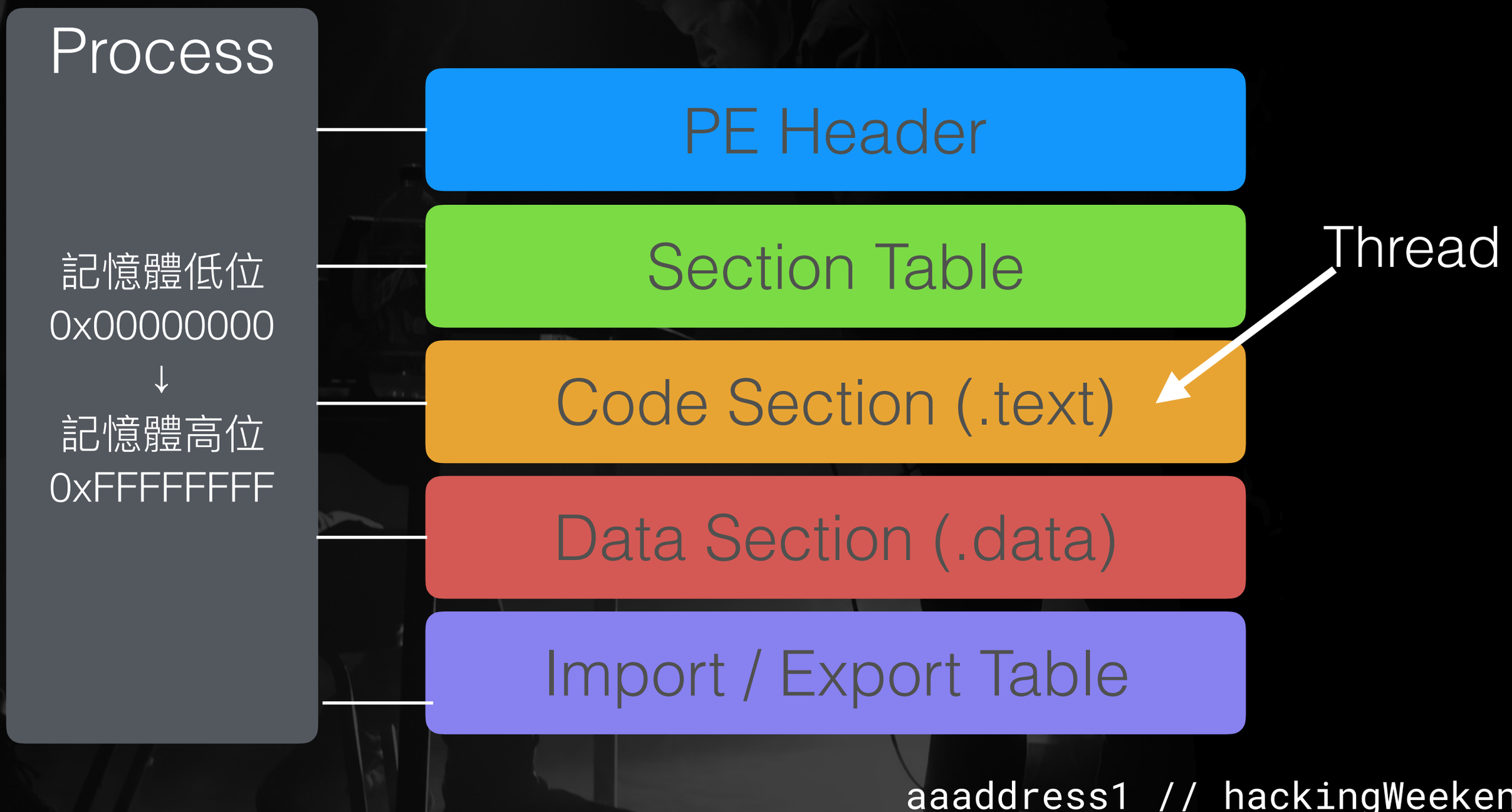
作業系統會根據不同區段（Section）申請記憶體空間來存放

＊當然這是理想上啦...實務上考量效能之類的可能有差異



Portable Executable

完成後，作業系統會創建一個執行緒（Thread）從
.text 區段中定義的起始點開始執行



逆向工程基礎（三）

機械碼

機械碼

在記憶體裡面看得到的機械碼長得像這樣
(沒錯，就是那個經過編譯器處理原始碼過後產生的東西)

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
100	55	8B	EC	83	EC	08	C7	45	F8	00	00	00	00	C7	45	FC
110	00	00	00	00	C7	45	F8	01	00	00	00	EB	09	8B	45	F8
120	83	C0	01	89	45	F8	8B	4D	F8	3B	4D	08	7F	0B	8B	55
130	FC	03	55	F8	89	55	FC	EB	E4	8B	45	FC	50	68	30	60
140	40	00	E8	62	00	00	00	83	C4	08	8B	E5	5D	C3		

機械碼

看不懂嗎？

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
100	55	8B	EC	83	EC	08	C7	45	F8	00	00	00	00	C7	45	FC
110	00	00	00	00	C7	45	F8	01	00	00	00	EB	09	8B	45	F8
120	83	C0	01	89	45	F8	8B	4D	F8	3B	4D	08	7F	0B	8B	55
130	FC	03	55	F8	89	55	FC	EB	E4	8B	45	FC	50	68	30	60
140	40	00	E8	62	00	00	00	83	C4	08	8B	E5	5D	C3		

九九乘法表

背過齁

2 × 1 = 2 2 × 2 = 4 2 × 3 = 6 2 × 4 = 8 2 × 5 = 10 2 × 6 = 12 2 × 7 = 14 2 × 8 = 16 2 × 9 = 18	3 × 1 = 3 3 × 2 = 6 3 × 3 = 9 3 × 4 = 12 3 × 5 = 15 3 × 6 = 18 3 × 7 = 21 3 × 8 = 24 3 × 9 = 27	4 × 1 = 4 4 × 2 = 8 4 × 3 = 12 4 × 4 = 16 4 × 5 = 20 4 × 6 = 24 4 × 7 = 28 4 × 8 = 32 4 × 9 = 36	5 × 1 = 5 5 × 2 = 10 5 × 3 = 15 5 × 4 = 20 5 × 5 = 25 5 × 6 = 30 5 × 7 = 35 5 × 8 = 40 5 × 9 = 45
6 × 1 = 6 6 × 2 = 12 6 × 3 = 18 6 × 4 = 24 6 × 5 = 30 6 × 6 = 36 6 × 7 = 42 6 × 8 = 48 6 × 9 = 54	7 × 1 = 7 7 × 2 = 14 7 × 3 = 21 7 × 4 = 28 7 × 5 = 35 7 × 6 = 42 7 × 7 = 49 7 × 8 = 56 7 × 9 = 63	8 × 1 = 8 8 × 2 = 16 8 × 3 = 24 8 × 4 = 32 8 × 5 = 40 8 × 6 = 48 8 × 7 = 56 8 × 8 = 64 8 × 9 = 72	9 × 1 = 9 9 × 2 = 18 9 × 3 = 27 9 × 4 = 36 9 × 5 = 45 9 × 6 = 54 9 × 7 = 63 9 × 8 = 72 9 × 9 = 81

我們來背FF指令表吧，耶比

Intel x86 Assembler Instruction Set Opcode Table

ADD Eb Gb *00*	**ADD** Ev Gv *01*	**ADD** Gb Eb *02*	**ADD** Gv Ev *03*	**ADD** AL Ib *04*	**ADD** eAX Iv *05*	**PUSH** ES *06*	**POP** ES *07*	**OR** Eb Gb *08*	**OR** Ev Gv *09*	**OR** Gb Eb *0A*	**OR** Gv Ev *0B*	**OR** AL Ib *0C*	**OR** eAX Iv *0D*	**PUSH** CS *0E*	**TWOBYTE** *0F*
ADC Eb Gb *10*	**ADC** Ev Gv *11*	**ADC** Gb Eb *12*	**ADC** Gv Ev *13*	**ADC** AL Ib *14*	**ADC** eAX Iv *15*	**PUSH** SS *16*	**POP** SS *17*	**SBB** Eb Gb *18*	**SBB** Ev Gv *19*	**SBB** Gb Eb *1A*	**SBB** Gv Ev *1B*	**SBB** AL Ib *1C*	**SBB** eAX Iv *1D*	**PUSH** DS *1E*	**POP** DS *1F*
AND Eb Gb *20*	**AND** Ev Gv *21*	**AND** Gb Eb *22*	**AND** Gv Ev *23*	**AND** AL Ib *24*	**AND** eAX Iv *25*	**ES:** *26*	**DAA** *27*	**SUB** Eb Gb *28*	**SUB** Ev Gv *29*	**SUB** Gb Eb *2A*	**SUB** Gv Ev *2B*	**SUB** AL Ib *2C*	**SUB** eAX Iv *2D*	**CS:** *2E*	**DAS** *2F*
XOR Eb Gb *30*	**XOR** Ev Gv *31*	**XOR** Gb Eb *32*	**XOR** Gv Ev *33*	**XOR** AL Ib *34*	**XOR** eAX Iv *35*	**SS:** *36*	**AAA** *37*	**CMP** Eb Gb *38*	**CMP** Ev Gv *39*	**CMP** Gb Eb *3A*	**CMP** Gv Ev *3B*	**CMP** AL Ib *3C*	**CMP** eAX Iv *3D*	**DS:** *3E*	**AAS** *3F*
INC eAX *40*	**INC** eCX *41*	**INC** eDX *42*	**INC** eBX *43*	**INC** eSP *44*	**INC** eBP *45*	**INC** eSI *46*	**INC** eDI *47*	**DEC** eAX *48*	**DEC** eCX *49*	**DEC** eDX *4A*	**DEC** eBX *4B*	**DEC** eSP *4C*	**DEC** eBP *4D*	**DEC** eSI *4E*	**DEC** eDI *4F*
PUSH eAX *50*	**PUSH** eCX *51*	**PUSH** eDX *52*	**PUSH** eBX *53*	**PUSH** eSP *54*	**PUSH** eBP *55*	**PUSH** eSI *56*	**PUSH** eDI *57*	**POP** eAX *58*	**POP** eCX *59*	**POP** eDX *5A*	**POP** eBX *5B*	**POP** eSP *5C*	**POP** eBP *5D*	**POP** eSI *5E*	**POP** eDI *5F*
PUSHA *60*	**POPA** *61*	**BOUND** Gv Ma *62*	**ARPL** Ew Gw *63*	**FS:** *64*	**GS:** *65*	**OPSIZE:** *66*	**ADSIZE:** *67*	**PUSH** Iv *68*	**IMUL** Gv Ev Iv *69*	**PUSH** Ib *6A*	**IMUL** Gv Ev Ib *6B*	**INSB** Yb DX *6C*	**INSW** Yz DX *6D*	**OUTSB** DX Xb *6E*	**OUTSW** DX Xv *6F*
JO Jb *70*	**JNO** Jb *71*	**JB** Jb *72*	**JNB** Jb *73*	**JZ** Jb *74*	**JNZ** Jb *75*	**JBE** Jb *76*	**JA** Jb *77*	**JS** Jb *78*	**JNS** Jb *79*	**JP** Jb *7A*	**JNP** Jb *7B*	**JL** Jb *7C*	**JNL** Jb *7D*	**JLE** Jb *7E*	**JNLE** Jb *7F*
ADD Eb Ib *80*	**ADD** Ev Iv *81*	**SUB** Eb Ib *82*	**SUB** Ev Ib *83*	**TEST** Eb Gb *84*	**TEST** Ev Gv *85*	**XCHG** Eb Gb *86*	**XCHG** Ev Gv *87*	**MOV** Eb Gb *88*	**MOV** Ev Gv *89*	**MOV** Gb Eb *8A*	**MOV** Gv Ev *8B*	**MOV** Ew Sw *8C*	**LEA** Gv M *8D*	**MOV** Sw Ew *8E*	**POP** Ev *8F*
NOP *90*	**XCHG** eAX eCX *91*	**XCHG** eAX eDX *92*	**XCHG** eAX eBX *93*	**XCHG** eAX eSP *94*	**XCHG** eAX eBP *95*	**XCHG** eAX eSI *96*	**XCHG** eAX eDI *97*	**CBW** *98*	**CWD** *99*	**CALL** Ap *9A*	**WAIT** *9B*	**PUSHF** Fv *9C*	**POPF** Fv *9D*	**SAHF** *9E*	**LAHF** *9F*
MOV AL Ob *A0*	**MOV** eAX Ov *A1*	**MOV** Ob AL *A2*	**MOV** Ov eAX *A3*	**MOVSB** Xb Yb *A4*	**MOVSW** Xv Yv *A5*	**CMPSB** Xb Yb *A6*	**CMPSW** Xv Yv *A7*	**TEST** AL Ib *A8*	**TEST** eAX Iv *A9*	**STOSB** Yb AL *AA*	**STOSW** Yv eAX *AB*	**LODSB** AL Xb *AC*	**LODSW** eAX Xv *AD*	**SCASB** AL Yb *AE*	**SCASW** eAX Yv *AF*
MOV AL Ib *B0*	**MOV** CL Ib *B1*	**MOV** DL Ib *B2*	**MOV** BL Ib *B3*	**MOV** AH Ib *B4*	**MOV** CH Ib *B5*	**MOV** DH Ib *B6*	**MOV** BH Ib *B7*	**MOV** eAX Iv *B8*	**MOV** eCX Iv *B9*	**MOV** eDX Iv *BA*	**MOV** eBX Iv *BB*	**MOV** eSP Iv *BC*	**MOV** eBP Iv *BD*	**MOV** eSI Iv *BE*	**MOV** eDI Iv *BF*
#2 Eb Ib *C0*	**#2** Ev Ib *C1*	**RETN** Iw *C2*	**RETN** *C3*	**LES** Gv Mp *C4*	**LDS** Gv Mp *C5*	**MOV** Eb Ib *C6*	**MOV** Ev Iv *C7*	**ENTER** Iw Ib *C8*	**LEAVE** *C9*	**RETF** Iw *CA*	**RETF** *CB*	**INT3** *CC*	**INT** Ib *CD*	**INTO** *CE*	**IRET** *CF*
#2 Eb 1 *D0*	**#2** Ev 1 *D1*	**#2** Eb CL *D2*	**#2** Ev CL *D3*	**AAM** Ib *D4*	**AAD** Ib *D5*	**SALC** *D6*	**XLAT** *D7*	**ESC** 0 *D8*	**ESC** 1 *D9*	**ESC** 2 *DA*	**ESC** 3 *DB*	**ESC** 4 *DC*	**ESC** 5 *DD*	**ESC** 6 *DE*	**ESC** 7 *DF*
LOOPNZ Jb *E0*	**LOOPZ** Jb *E1*	**LOOP** Jb *E2*	**JCXZ** Jb *E3*	**IN** AL Ib *E4*	**IN** eAX Ib *E5*	**OUT** Ib AL *E6*	**OUT** Ib eAX *E7*	**CALL** Jz *E8*	**JMP** Jz *E9*	**JMP** Ap *EA*	**JMP** Jb *EB*	**IN** AL DX *EC*	**IN** eAX DX *ED*	**OUT** DX AL *EE*	**OUT** DX eAX *EF*
LOCK: *F0*	**INT1** *F1*	**REPNE:** *F2*	**REP:** *F3*	**HLT** *F4*	**CMC** *F5*	**#3** Eb *F6*	**#3** Ev *F7*	**CLC** *F8*	**STC** *F9*	**CLI** *FA*	**STI** *FB*	**CLD** *FC*	**STD** *FD*	**#4** INC/DEC *FE*	**#5** INC/DEC *FF*

機械碼

執行緒由 0x100 開始運行，fetch 到第一個 byte 為 0x55

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
100	55	8B	EC	83	EC	08	C7	45	F8	00	00	00	00	C7	45	FC
110	00	00	00	00	C7	45	F8	01	00	00	00	EB	09	8B	45	F8
120	83	C0	01	89	45	F8	8B	4D	F8	3B	4D	08	7F	0B	8B	55
130	FC	03	55	F8	89	55	FC	EB	E4	8B	45	FC	50	68	30	60
140	40	00	E8	62	00	00	00	83	C4	08	8B	E5	5D	C3		

機械碼

反查 0x55 在 FF 乘法表上為 push ebp，接著讀下一個指令

	0	1	2	3	4	5	6	7	8	9	A	B
100	55	8B	EC	83	EC	08	C7	45	F8	00	00	00
110	00	00	00	00	C7	45	F8	01	00	00	00	EB
120	83	C0	01	89	45	F8	8B	4D	F8	3B	4D	08
130	FC	03	55	F8	89	55	FC	EB	E4	8B	45	FC
140	40	00	E8	62	00	00	00	83	C4	08	8B	E5

機械碼

反查 0x55 在 FF 乘法表上為 push ebp，接著讀下一個指令

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
100	55	8B	EC	83	EC	08	C7	45	F8	00	00	00	00	C7	45	FC
110	00	00	00	00	C7	45	F8	01	00	00	00	EB	09	8B	45	F8
120	83	C0	01	89	45	F8	8B	4D	F8	3B	4D	08	7F	0B	8B	55
130	FC	03	55	F8	89	55	FC	EB	E4	8B	45	FC	50	68	30	60
140	40	00	E8	62	00	00	00	83	C4	08	8B	E5	5D	C3		

機械碼

fetch 到 0x8B 為 mov Gv, Ev，讀取到下一個 byte 為 0xEC，
可以得知 8B, EC 的意思對應為：mov ebp, esp

	0	1	2	3	4	5	6	7	8	9	A	B	C
100	55	8B	EC	83	EC	08	C7	45	F8	00	00	00	0(
110	00	00	00	00	C7	45	F8	01	00	00	00	EB	09
120	83	C0	01	89	45	F8	8B	4D	F8	3B	4D	08	7F
130	FC	03	55	F8	89	55	FC	EB	E4	8B	45	FC	50
140	40	00	E8	62	00	00	00	83	C4	08	8B	E5	5D

MOV
Gv Ev
8B

機械碼

逆向工程就是熟背 FF 指令表，
肉眼讀機械碼，今天就教到這邊，
回去大家記得多背 FF 指令表，結束

(哽？

善用工具，工具很棒

www.onlinedisassembler.com/odaweb

Live View

Set the platform below. Then watch the disassembly window update as you type hex bytes in the text area. You can also upload an ELF, PE, COFF, Mach-O, or other executable file from the *File* menu.

Platform: i386

```
55 8B EC 83 EC 08 C7 45  F8 00 00 00 00 C7 45 FC
00 00 00 00 C7 45 F8 01  00 00 00 EB 09 8B 45 F8
83 C0 01 89 45 F8 8B 4D  F8 3B 4D 08 7F 0B 8B 55
FC 03 55 F8 89 55 FC EB  E4 8B 45 FC 50 68 30 60
40 00 E8 62 00 00 00 83  C4 08 8B E5 5D C3
```

Hex Sections File Info

```
.data:00000000 55                  push ebp
.data:00000001 8bec                mov ebp,esp
.data:00000003 83ec08              sub esp,0x8
.data:00000006 c745f800000000      mov DWORD PTR [ebp-0x8],0x0
.data:0000000d c745fc00000000      mov DWORD PTR [ebp-0x4],0x0
.data:00000014 c745f801000000      mov DWORD PTR [ebp-0x8],0x1
.data:0000001b eb09                jmp loc_00000026
.data:0000001d
.data:0000001d                   loc_0000001d:
.data:0000001d 8b45f8                mov eax,DWORD PTR [ebp-0x8]
.data:00000020 83c001                add eax,0x1
.data:00000023 8945f8                mov DWORD PTR [ebp-0x8],eax
.data:00000026
.data:00000026                   loc_00000026:
.data:00000026 8b4df8                mov ecx,DWORD PTR [ebp-0x8]
.data:00000029 3b4d08                cmp ecx,DWORD PTR [ebp+0x8]
.data:0000002c 7f0b                  jg loc_00000039
.data:0000002e 8b55fc                mov edx,DWORD PTR [ebp-0x4]
.data:00000031 0355f8                add edx,DWORD PTR [ebp-0x8]
.data:00000034 8955fc                mov DWORD PTR [ebp-0x4],edx
.data:00000037 ebe4                  jmp loc_0000001d
.data:00000039
.data:00000039                   loc_00000039:
.data:00000039 8b45fc                mov eax,DWORD PTR [ebp-0x4]
.data:0000003c 50                    push eax
.data:0000003d 6830604000            push 0x406030
.data:00000042 e862000000            call loc_000000a9
.data:00000047 83c408                add esp,0x8
.data:0000004a 8be5                  mov esp,ebp
.data:0000004c 5d                    pop ebp
.data:0000004d c3                    ret
```

aaaddress1 // hackingWeekend

開發者的程式碼

```
void print_sum(int num_in) {
  int i = 0, sum = 0;

  for (i = 1; i <= num_in; i++)
    sum += i;

  printf("sum: %i\n", sum);
}
```

組合語言

```
.text:00401006                          mov     [ebp+var_8], 0
.text:0040100D                          mov     [ebp+var_4], 0
.text:00401014                          mov     [ebp+var_8], 1
.text:0040101B                          jmp     short loc_401026
.text:0040101D loc_40101D:
.text:0040101D                          mov     eax, [ebp+var_8]
.text:00401020                          add     eax, 1
.text:00401023                          mov     [ebp+var_8], eax
.text:00401026 loc_401026:
.text:00401026                          mov     ecx, [ebp+var_8]
.text:00401029                          cmp     ecx, [ebp+arg_0]
.text:0040102C                          jg      short loc_401039
.text:0040102E                          mov     edx, [ebp+var_4]
.text:00401031                          add     edx, [ebp+var_8]
.text:00401034                          mov     [ebp+var_4], edx
.text:00401037                          jmp     short loc_40101D
.text:00401039 loc_401039:
.text:00401039                          mov     eax, [ebp+var_4]
.text:0040103C                          push    eax
.text:0040103D                          push    offset aSumI    ; "sum: %i\n"
.text:00401042                          call    _printf
```

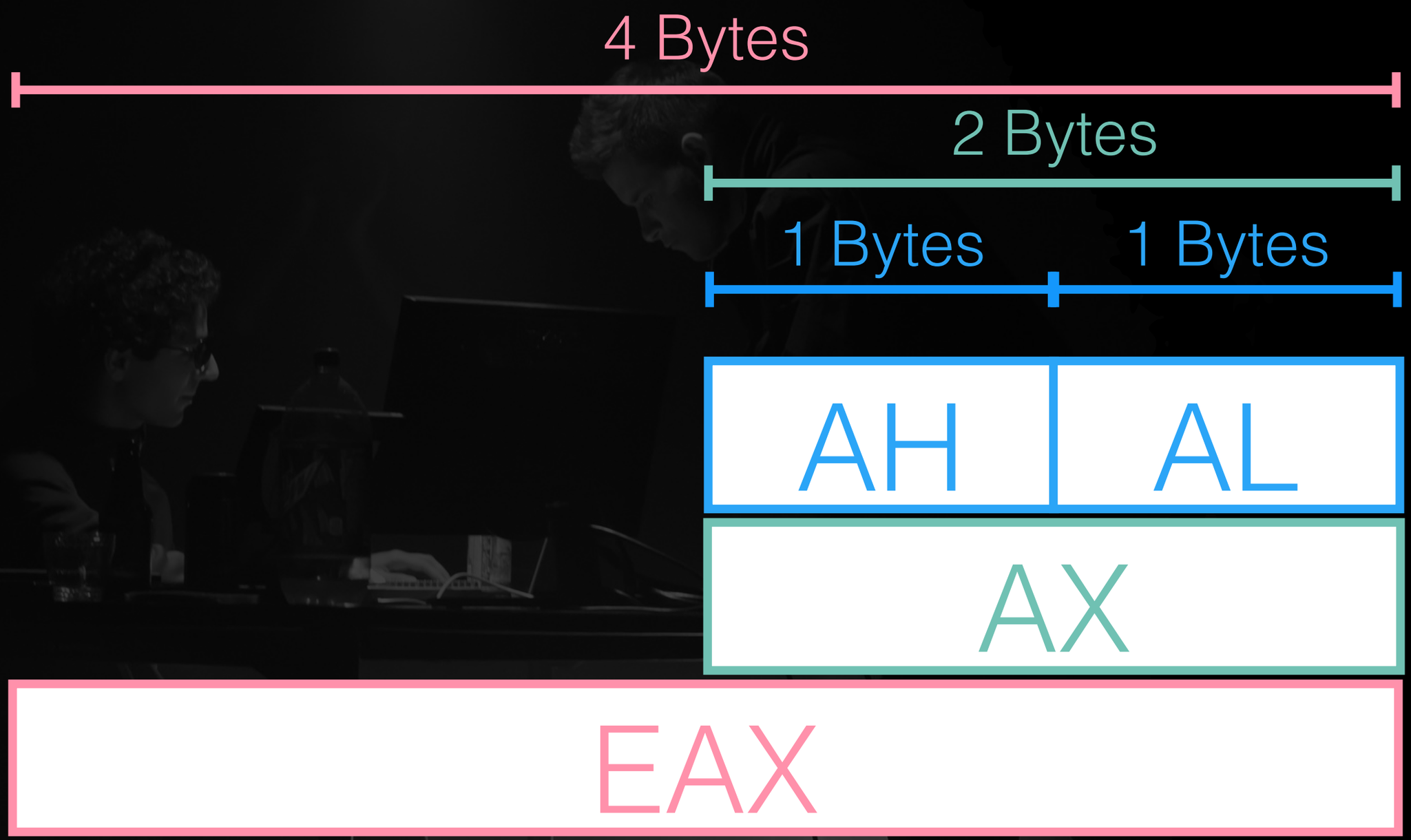
逆向工程基礎（四）

組合語言

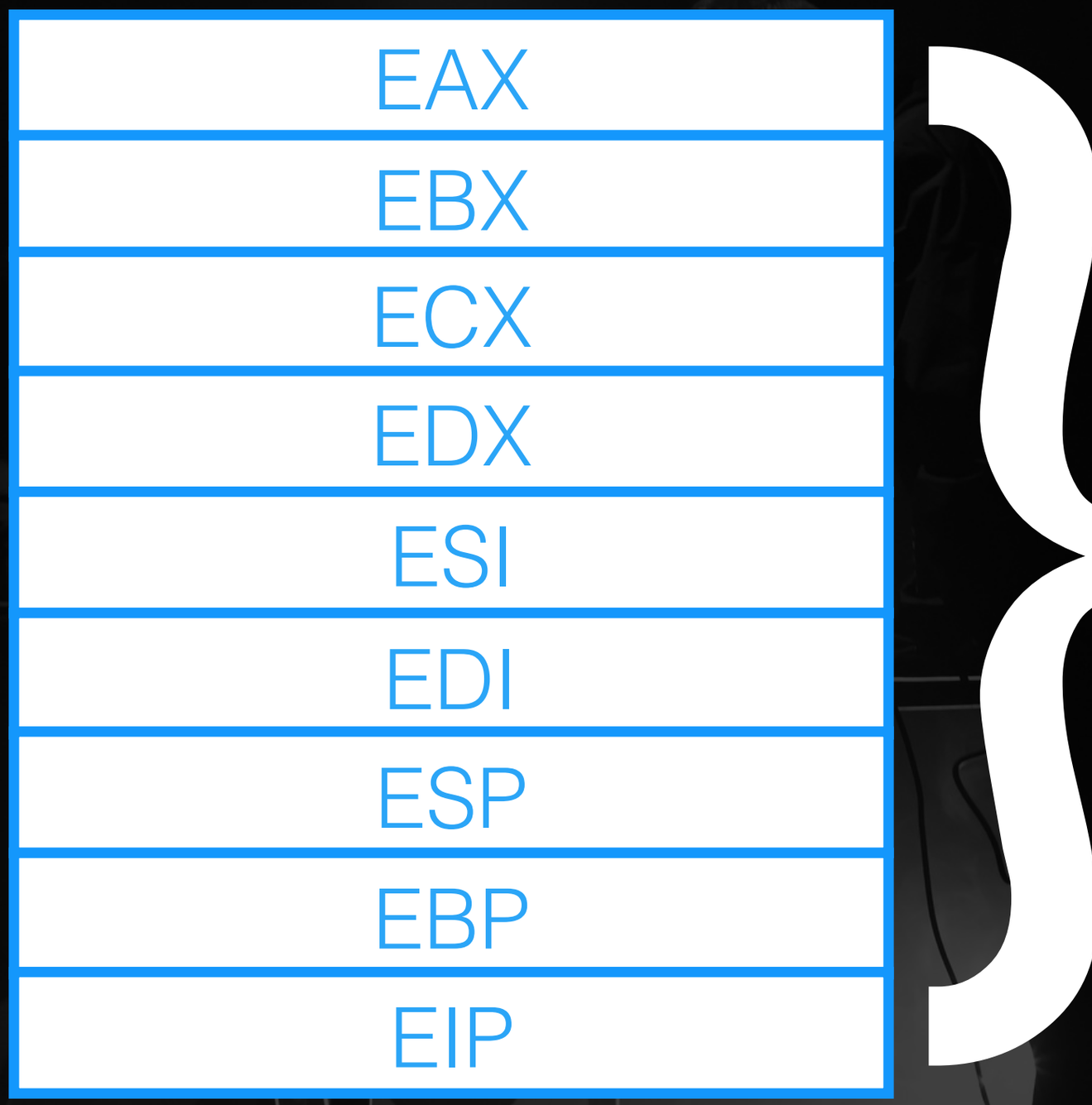
暫存器

- 每個執行緒（Thread）運行時會配發一組暫存器：eax、ebx、ecx、edx、esi、edi、ebp、esp、eip
- 每個暫存器可以儲存 4 bytes 大小的資料，也就是一個 Integer
- esp 指向一組堆疊（一大片記憶體空間）

暫存器



一組暫存器



暫存器公定用法

大家約好必須這樣做（ㄩㄇ）

EAX
EBX
ECX
EDX
ESI
EDI
ESP
EBP
EIP

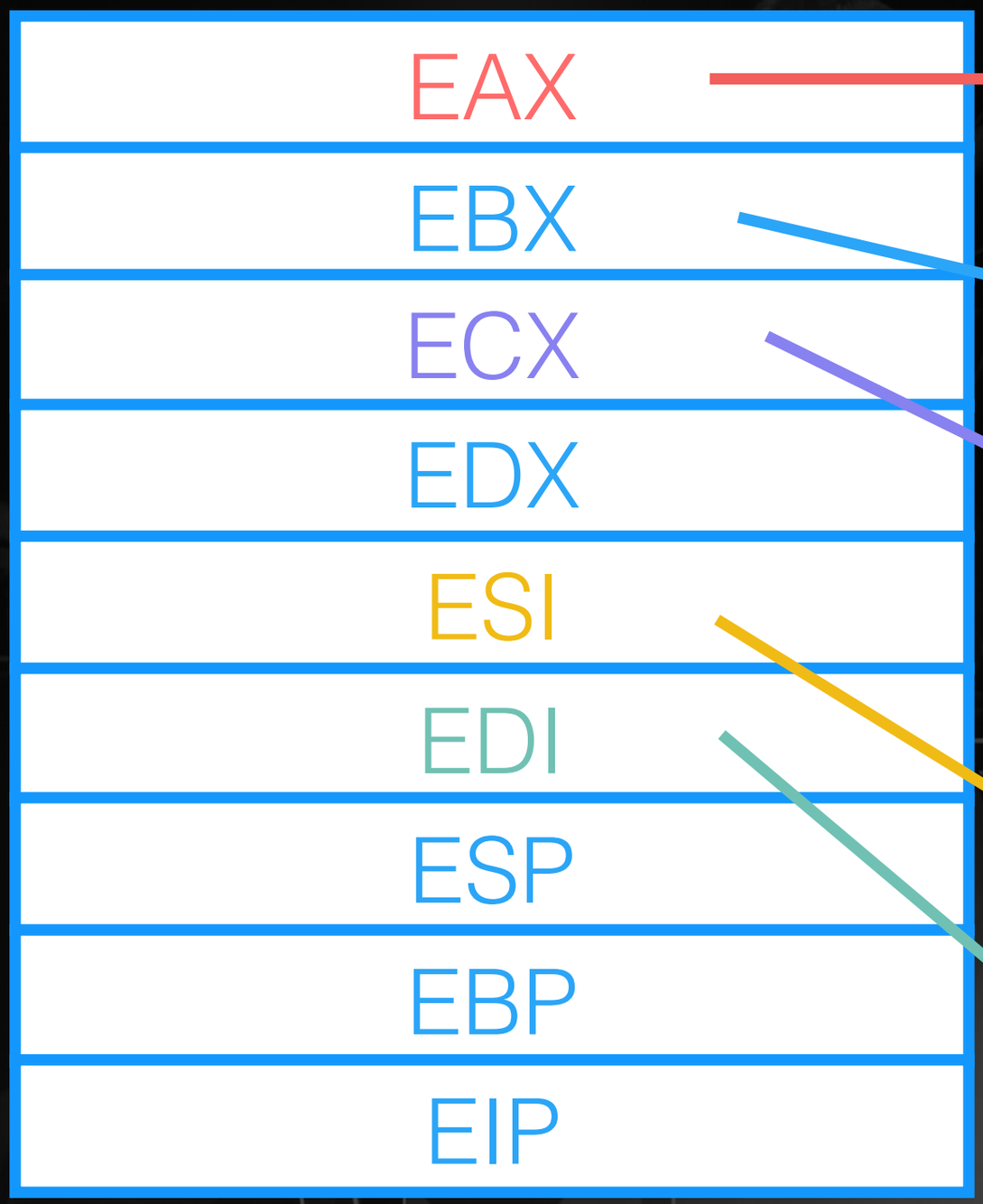
指向堆疊（Stack）

當層函數的堆疊基礎指標

指向下一行指令地址

暫存器不成文約定...

大家很愛這麼做，但事實上沒特別規定這樣做



慣性被拿來當暫時用變數與函數回傳值

很常被拿來儲存某個指標

被當計數器用

很常被拿來當來源 Index 用
(特別是字串拷貝)

很常拿來被當目標 Index 用
(特別是字串拷貝)

aaaddress1 // hackingWeekend

暫存器不成文約定...

不要問我為什麼，我真的不知道XD（不用死記）

組合語言

- 指令集定義在那張 FF 指令表上（大部分啦）
- 指令形式如下

```
opcode arg1, arg2, arg3
```

- 常見指令如 add、sub、mul、div（加減乘除）
- `add eax, ebx //eax = (eax + ebx)`
- [arg] 即為將 arg 當作 pointer 用（*arg）

基本運算

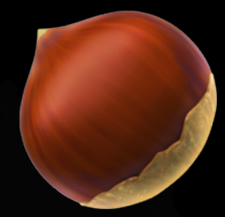
搬移

- mov eax, ebx
  //eax = ebx

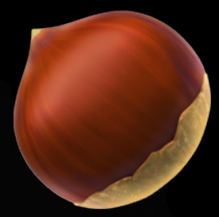
- mov [eax], ebx
  mov dword ptr [eax], ebx
  // *(dword*)eax = ebx

- mov [ax], bx
  mov word ptr [ax], bx
  // *(word*)eax = ebx

- mov [al], bl
  mov byte ptr [al], bl
  // *((byte*)eax) = ebx

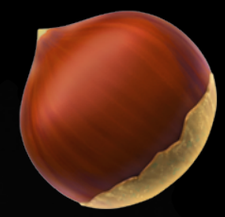
舉個 

```
mov  eax , 0xDEADBEEF
mov [eax], 0xEA7C0FEE
```

舉個 

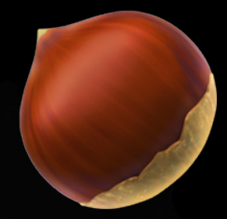
暫存器

```
mov  eax , 0xDEADBEEF
mov [eax], 0xEA7C0FEE
```

舉個 

指令

```
mov   eax , 0xDEADBEEF
mov [eax], 0xEA7C0FEE
```

舉個 

addr	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
DEADBEEF	EE	0F	7C	EA												

```
mov  eax , 0xDEADBEEF
mov [eax], 0xEA7C0FEE
```

基本運算

加法、減法

- `add eax, ebx` `//eax += ebx`
- `sub eax, ebx` `//eax -= ebx`
- `inc eax` `//eax ++`
- `dec eax` `//eax --`

基本運算

比對、跳轉

- `cmp eax, ebx  // T = eax - ebx`
- `je  0xdead  // if (T==0) EIP=0xdead`
- `jl  0xbeef  // if (T <0) EIP=0xbeef`
- `jg  0xf00d  // if (T >0) EIP=0xf00d`
- `jle 0xb00c  // if (T<=0) EIP=0xb00c`
- `jmp 0xcafe  // EIP = 0xcafe`

基本運算

無號數乘法

mul 將 eax 乘 乘數 後，答案為 8bytes
高 4bytes 寫入 edx、低 4bytes 寫入 eax

```
mov eax, 0x02
mov edx, 0x03
mul edx
```

結果：

```
eax = 6, edx = 0
```

```
mov eax, 0x02
mov edx, 0x03
mul eax
```

結果：

```
eax = 4, edx = 0
```

基本運算

無號數乘法

mul 將 eax 乘 乘數 後，答案為 8bytes
高 4bytes 寫入 edx、低 4bytes 寫入 eax

```
mov eax, 0x02

mov edx, 0xFFFFFFFF

mul edx
```

結果：

```
eax = 0xFFFFFFFE,
edx = 0x00000001,
Carry = 1,
Overflow = 1
```

基本運算

有號數乘法

imul 將 eax 乘 乘數 後，答案為 8bytes
高 4bytes 寫入 edx、低 4bytes 寫入 eax

```
mov eax, 0x02
mov edx, 0x03
imul edx
```

結果：

```
eax = 6, edx = 0
```

```
mov eax, -2
mov edx, 3
imul eax
```

結果：

```
eax = 0xFFFFFFFA,
edx = 0xFFFFFFFF
```

基本運算

無號數除法

div 將 eax 除 除數 後，答案為 8bytes
結果寫入 eax，餘數寫入 edx
＊使用 div 前記得將 edx 清為 0

```
mov eax, 0x7
mov ecx, 0x2
mov edx, 0x0
div ecx
```

結果：

```
eax = 3, edx = 1, ecx = 2
```

基本運算

有號數除法

idiv 將 eax 除 除數 後，答案為 8bytes
結果寫入 eax，餘數寫入 edx
＊使用 idiv 前記得將 edx 清為 0

```
mov eax, 0x7
mov ecx, 0x2
mov edx, 0x0
idiv ecx
```

結果：

eax = 3, edx = 1, ecx = 2

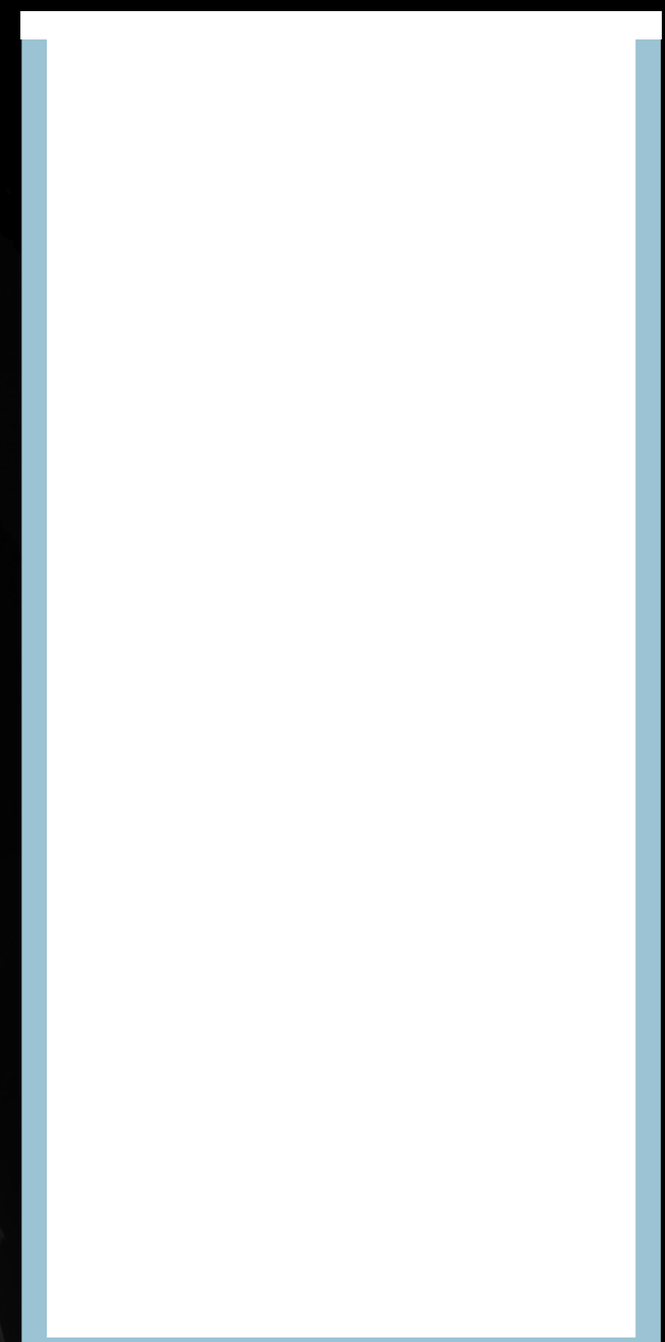
練習 I:

計算 (99 * 99) / 123

http://carlosrafaelgn.com.br/asm86

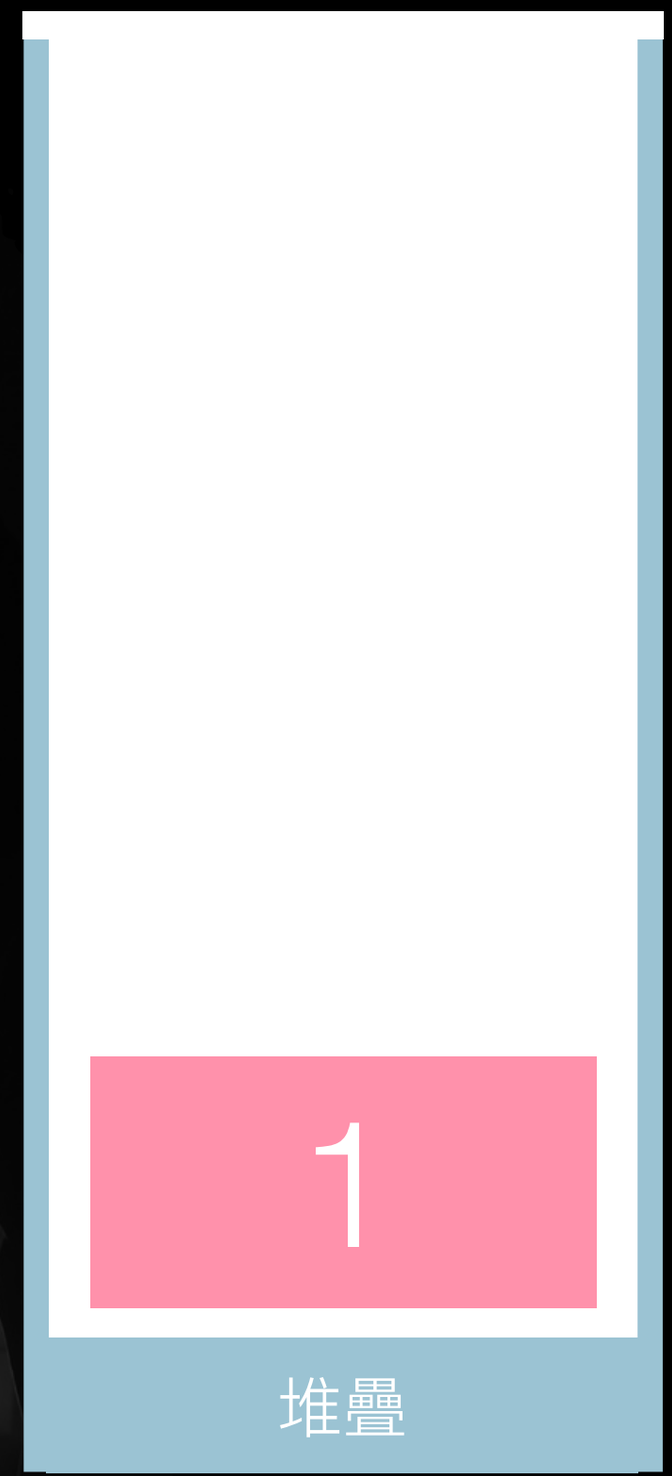
堆疊 (Stack)

```
push 1
push 2
pop  ebx
push 3
pop  eax
```



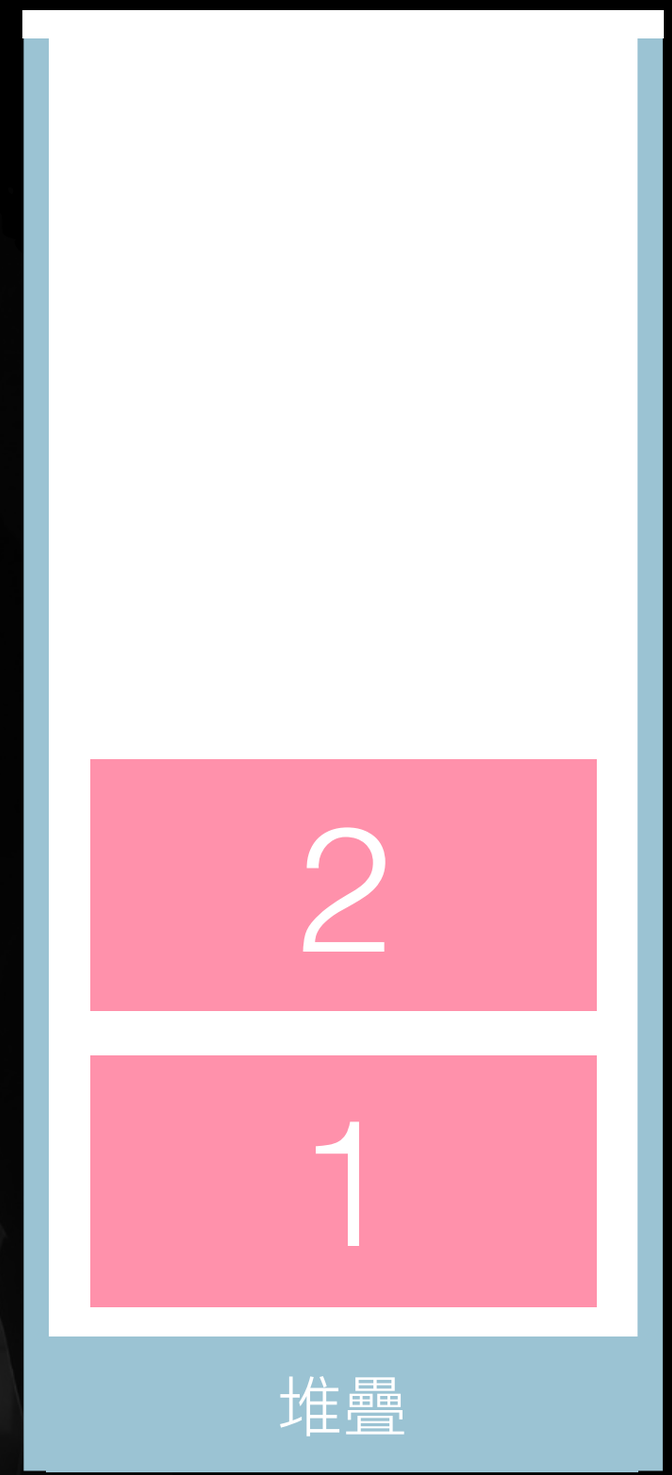
堆疊 (Stack)

```
push 1
push 2
pop  ebx
push 3
pop  eax
```



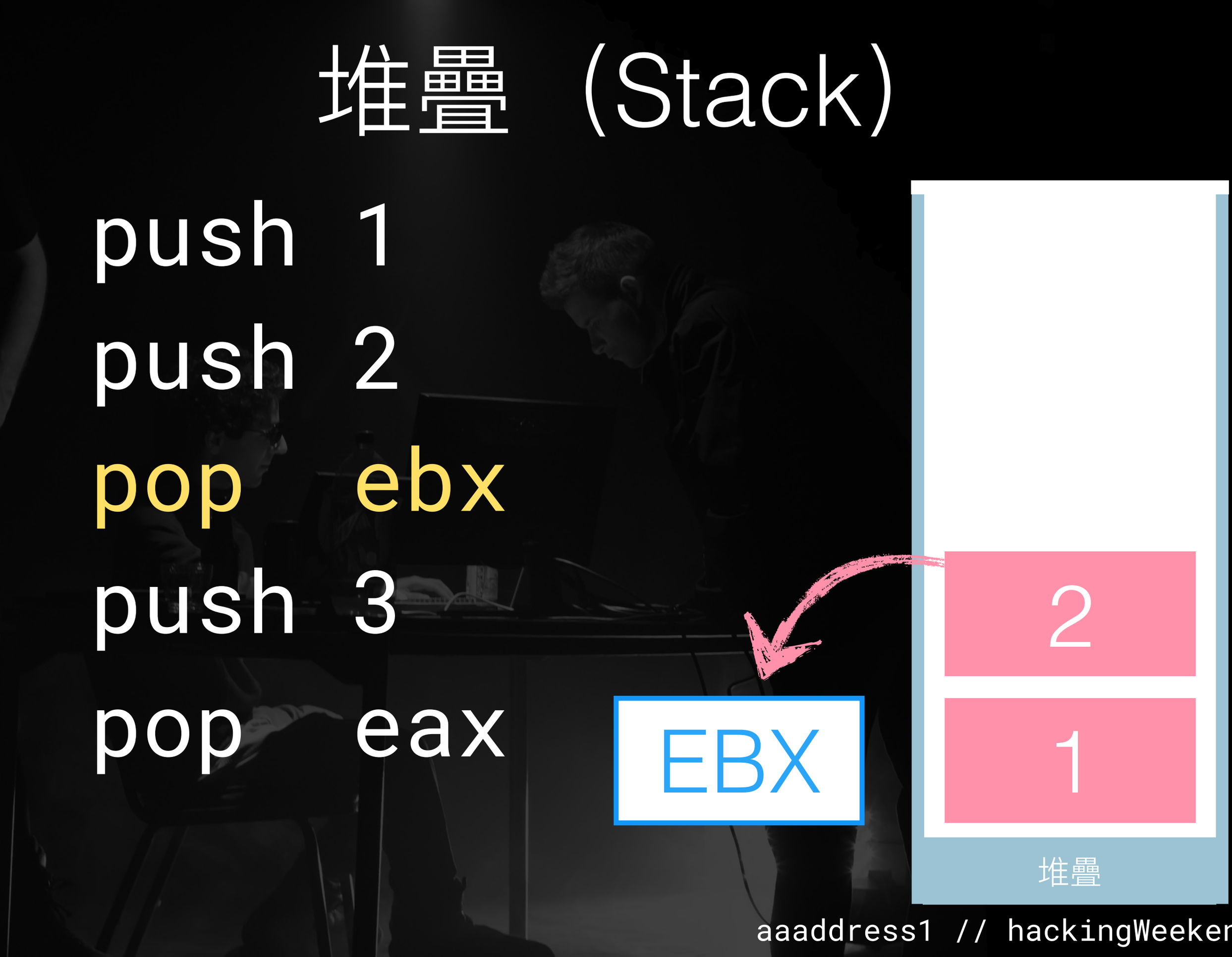
堆疊（Stack）

```
push 1
push 2
pop  ebx
push 3
pop  eax
```



堆疊 (Stack)

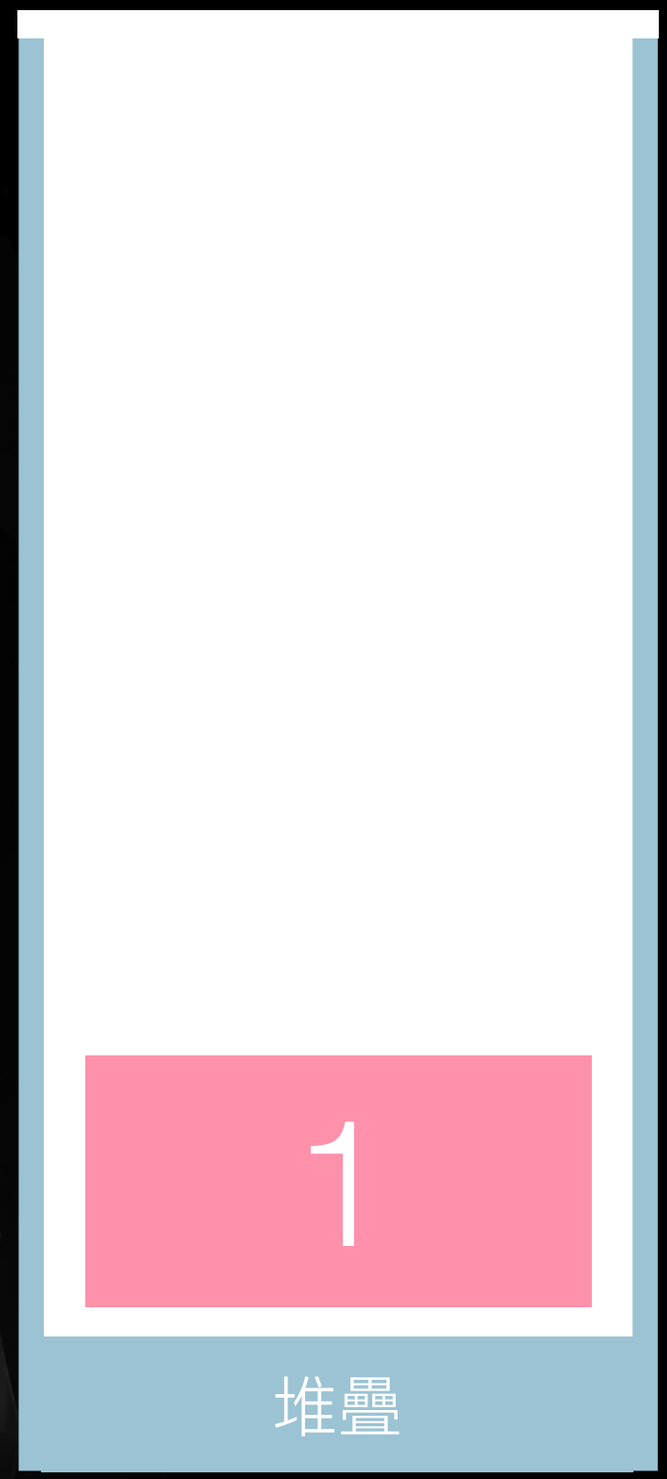
```
push 1
push 2
pop  ebx
push 3
pop  eax
```



堆疊 (Stack)

```
push 1
push 2
pop  ebx
push 3
pop  eax
```

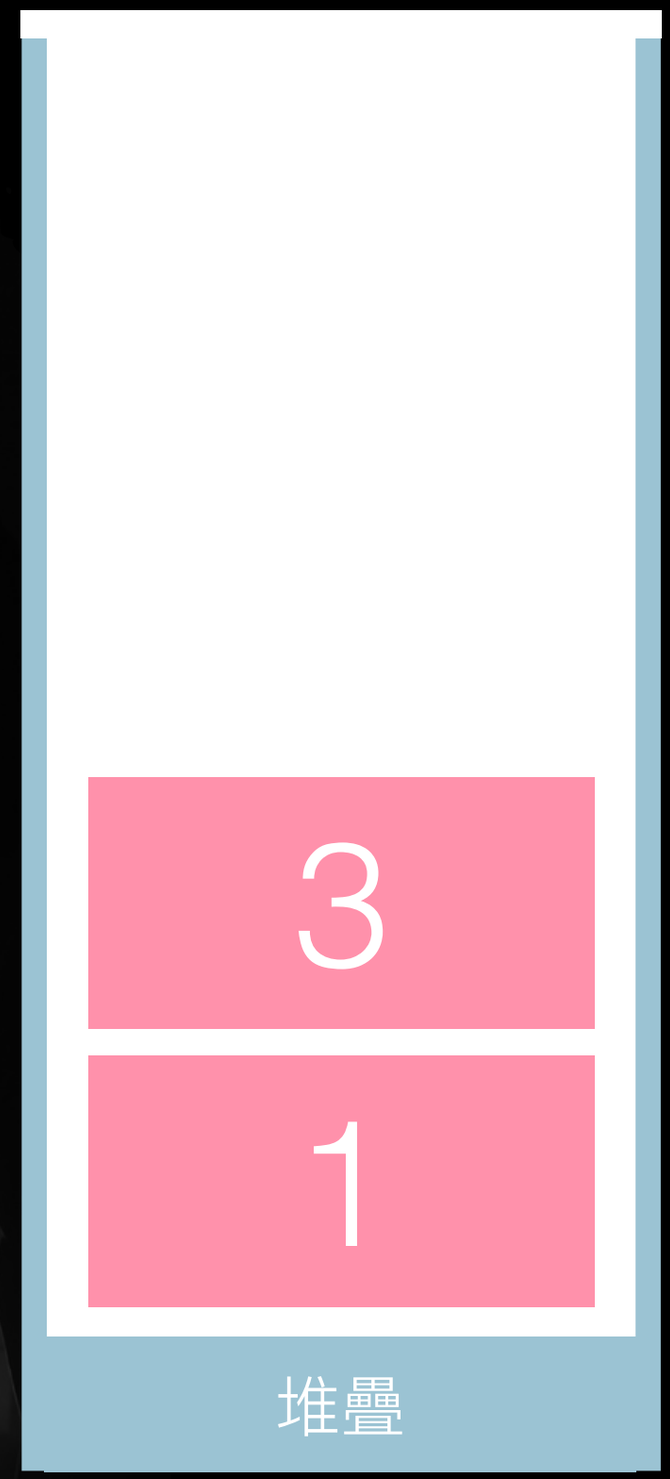
EBX: 2



堆疊（Stack）

```
push 1
push 2
pop  ebx
push 3
pop  eax
```

EBX: 2

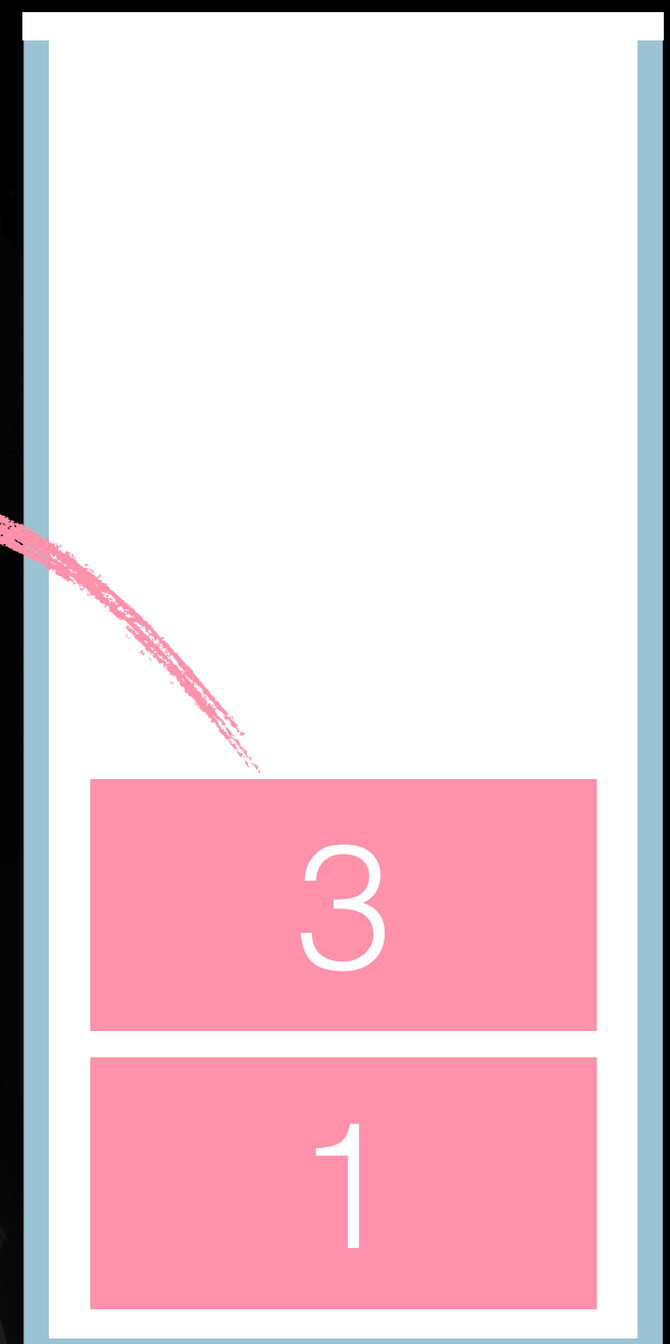


堆疊（Stack）

```
push 1
push 2
pop  ebx
push 3
pop  eax
```

EAX

EBX: 2

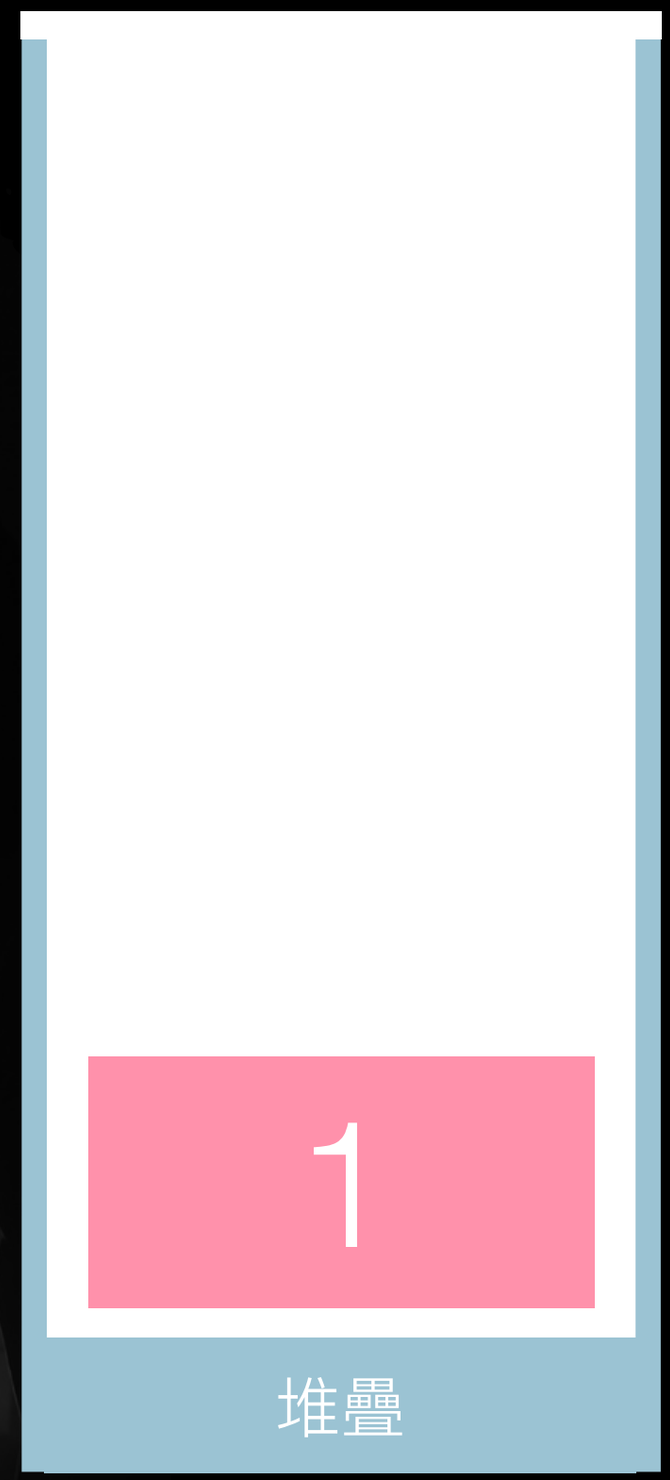


堆疊（Stack）

```
push 1
push 2
pop  ebx
push 3
pop  eax
```

EAX: 3

EBX: 2

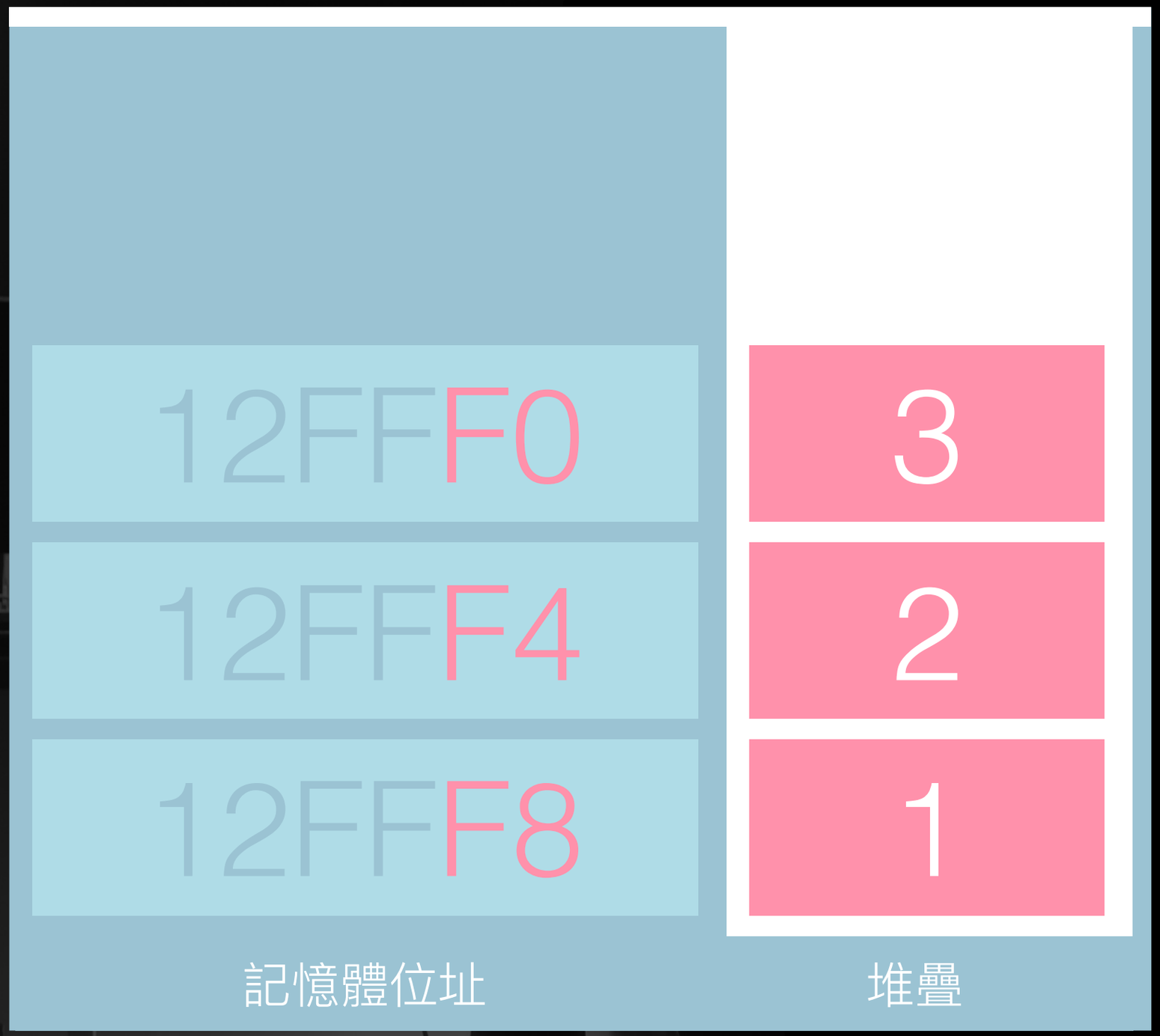


堆疊（Stack）

push 1

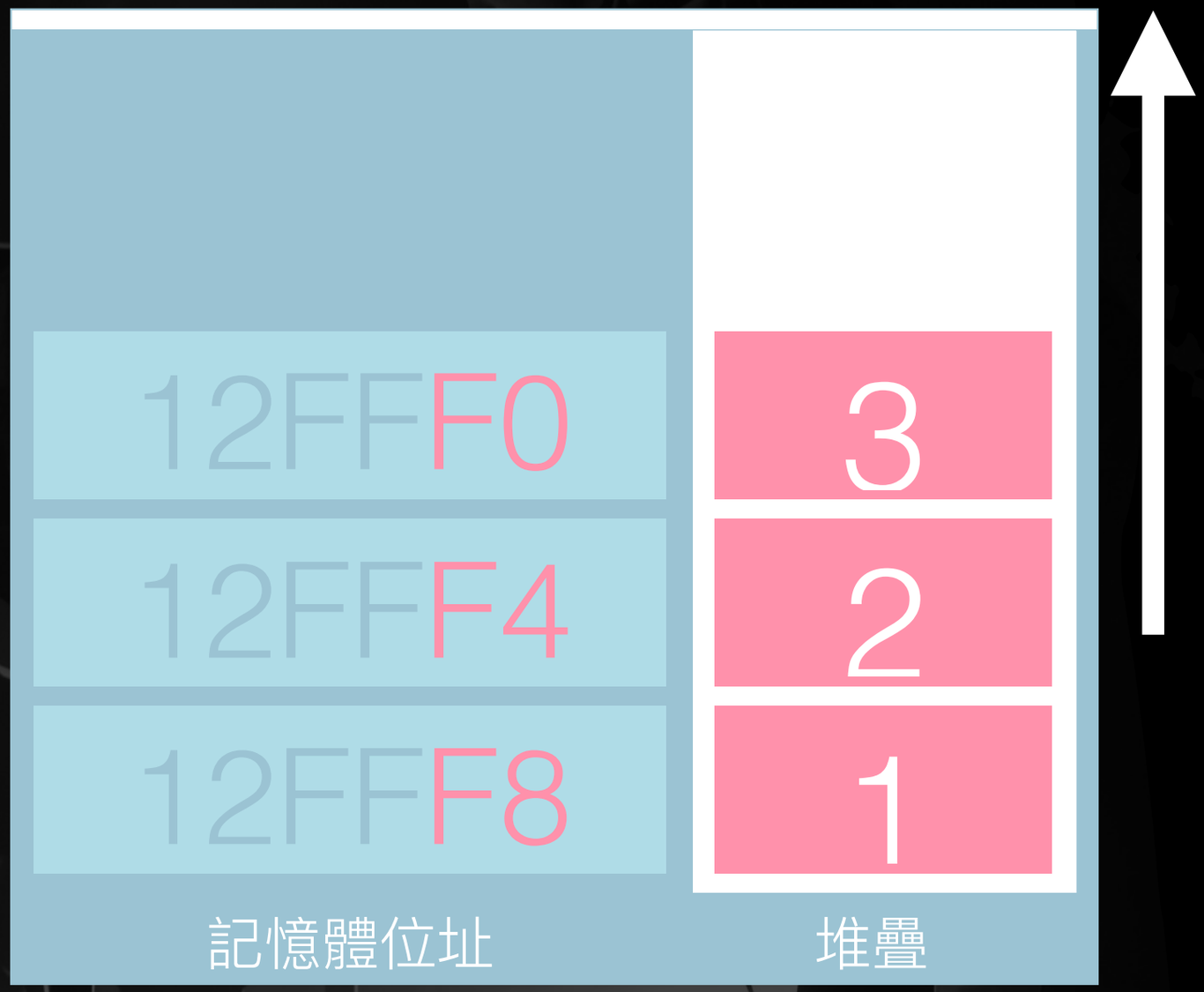
push 2

push 3



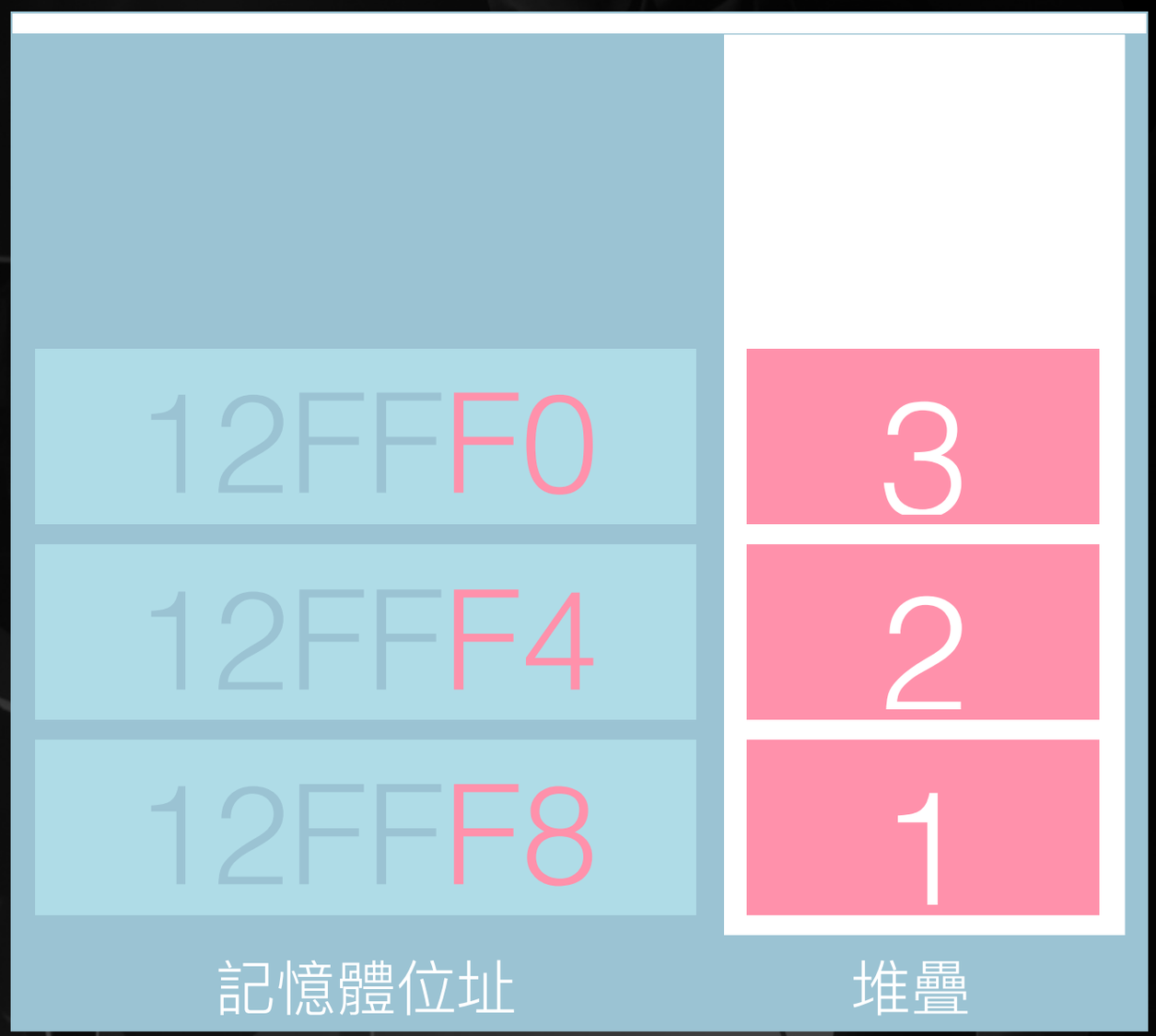
堆疊（Stack）

越早推入堆疊的資料所在的記憶體地址越高；
越晚推入堆疊的資料所在的記憶體地址越低。



堆疊 (Stack)

```
push 01 = sub esp, 4; mov [esp], 01
pop eax = mov eax,[esp]; add esp, 04
```



呼叫約制
Calling Convention

```
add(1, 2, 3)
```

```
int add(int a, int b, int c)
{
  int tmp = (a + b + c);
  return tmp;
}
```

呼叫約制

Calling Convention

```
push 3
push 2
push 1
call add
add  esp,0x0C
//add(1, 2, 3)
```

```
int add(int a, int b, int c)
{
  int tmp = (a + b + c);
  return tmp;
}
```

1. 參數由右至左推入堆疊中
2. 呼叫者負責清理堆疊（stdcall）

呼叫約制

堆疊清理問題

X86調用約定 [編輯]

維基百科，自由的百科全書

這裡描述了在x86晶片架構上的調用約定(calling conventions)。 調用約定描述了被調用代碼的接口：

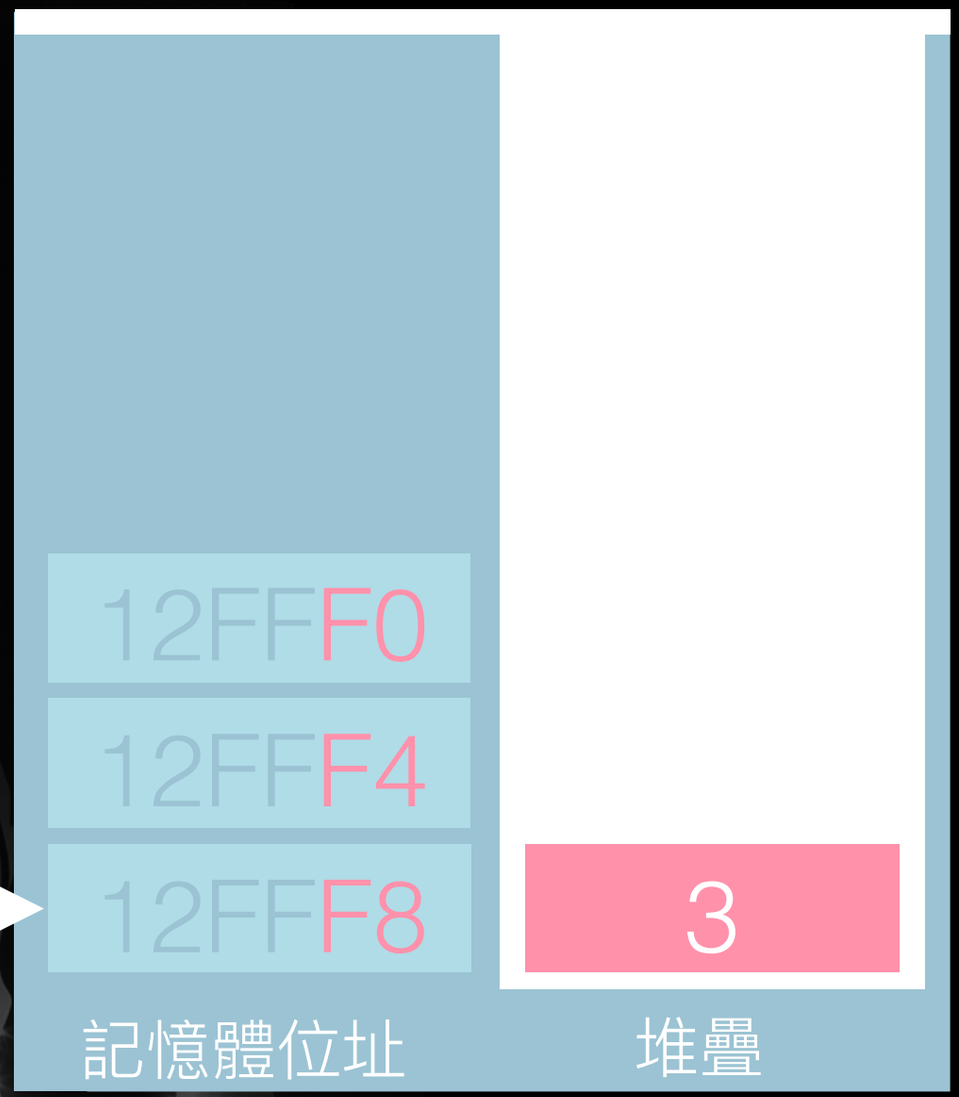
- 原子(純量)參數，或複雜參數獨立部分的分配順序;
- 參數是如何被傳遞的(放置在堆疊上，或是暫存器中，亦或兩者混合);
- 被調用者應保存調用者的哪個暫存器;
- 調用函數時如何為任務準備堆疊，以及任務完成如何恢復;

zh.wikipedia.org/wiki/X86调用约定

呼叫約制

Calling Convention

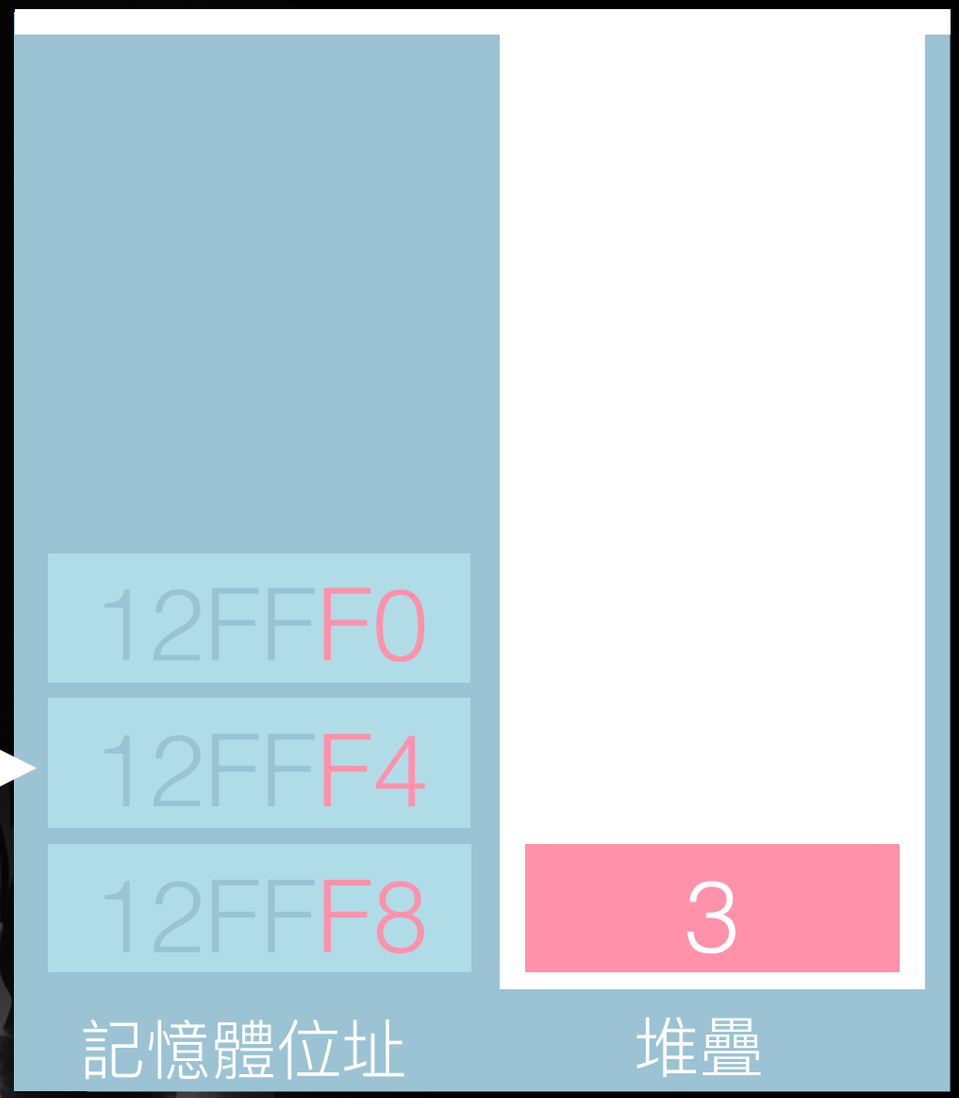
```
push 3
push 2
push 1
call add
add  esp,0x0C
//add(1, 2, 3)
```



呼叫約制

Calling Convention

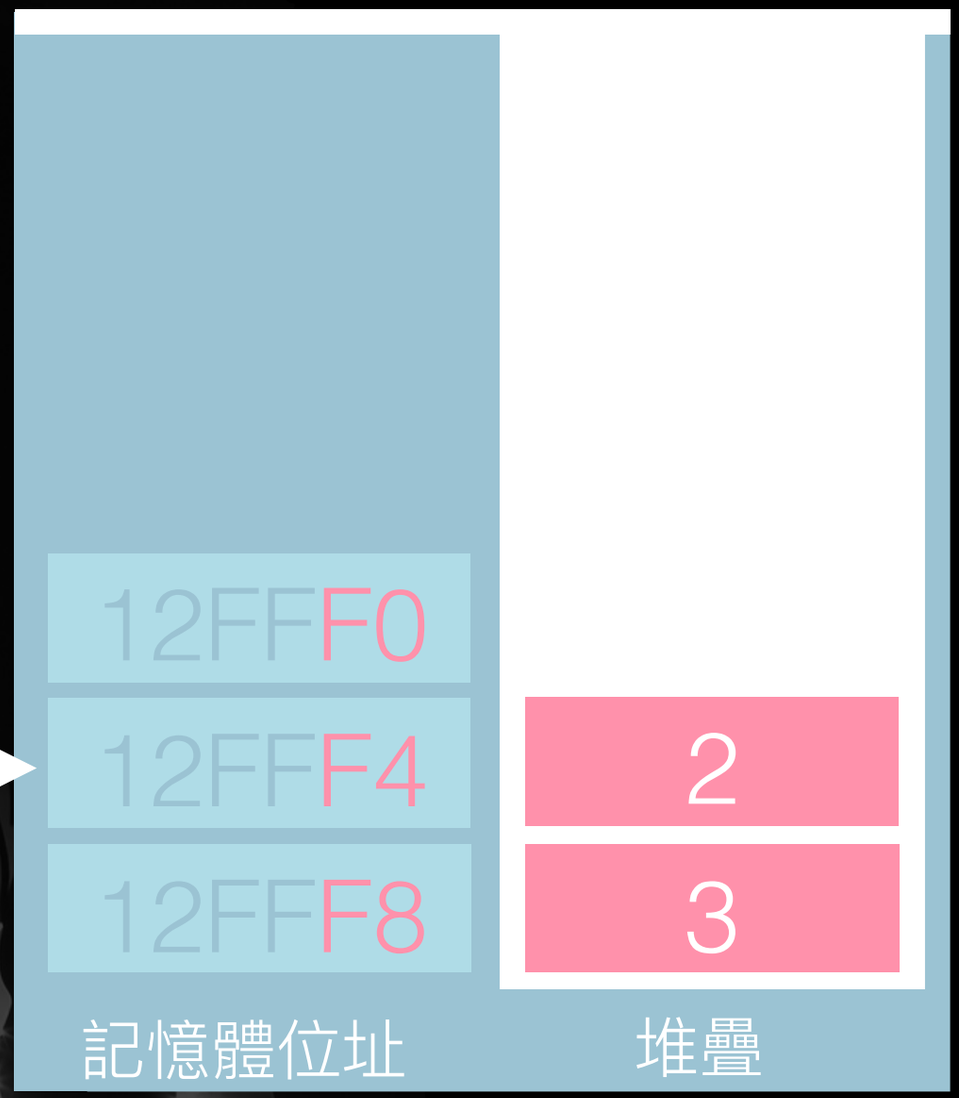
```
push 3
push 2
push 1
call add
add  esp,0x0C
//add(1, 2, 3)
```



呼叫約制

Calling Convention

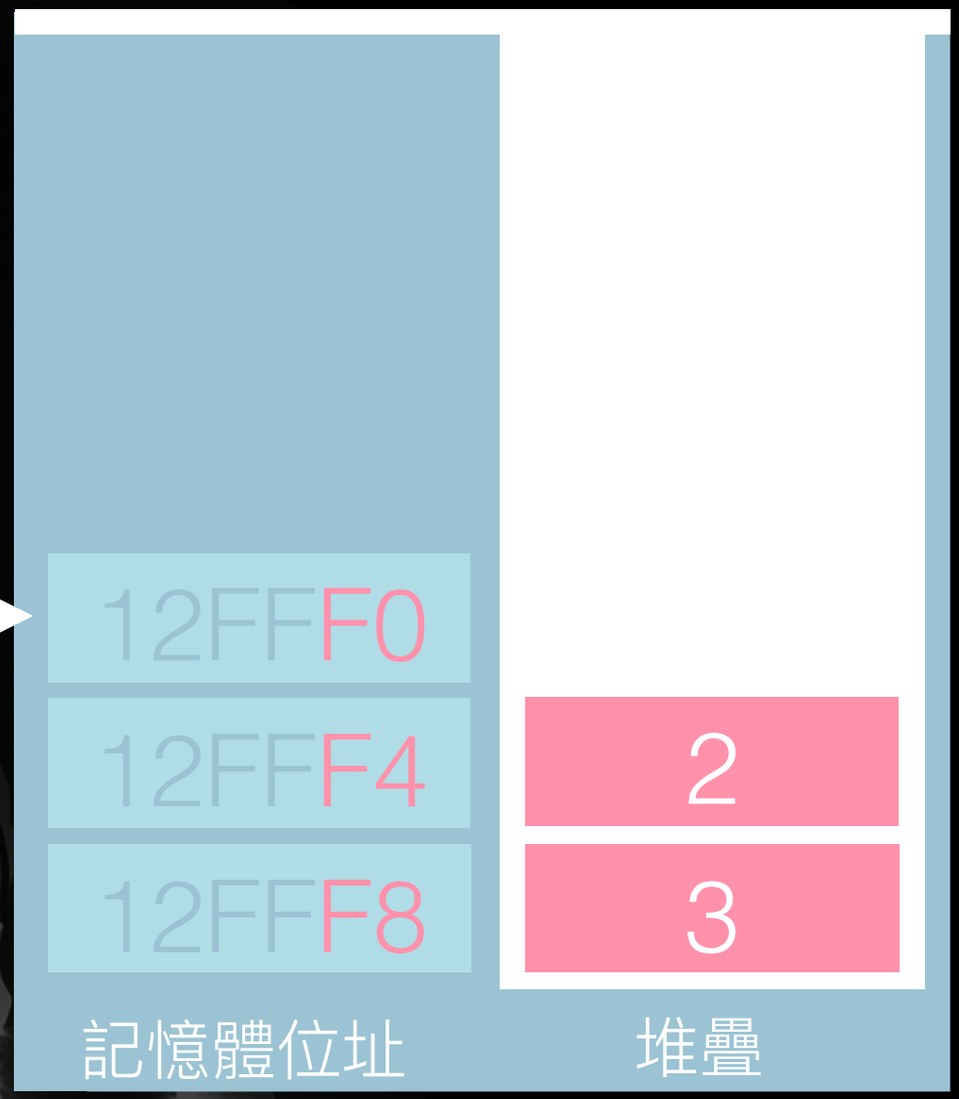
```
push 3
push 2
push 1
call add
add  esp,0x0C
//add(1, 2, 3)
```



呼叫約制

Calling Convention

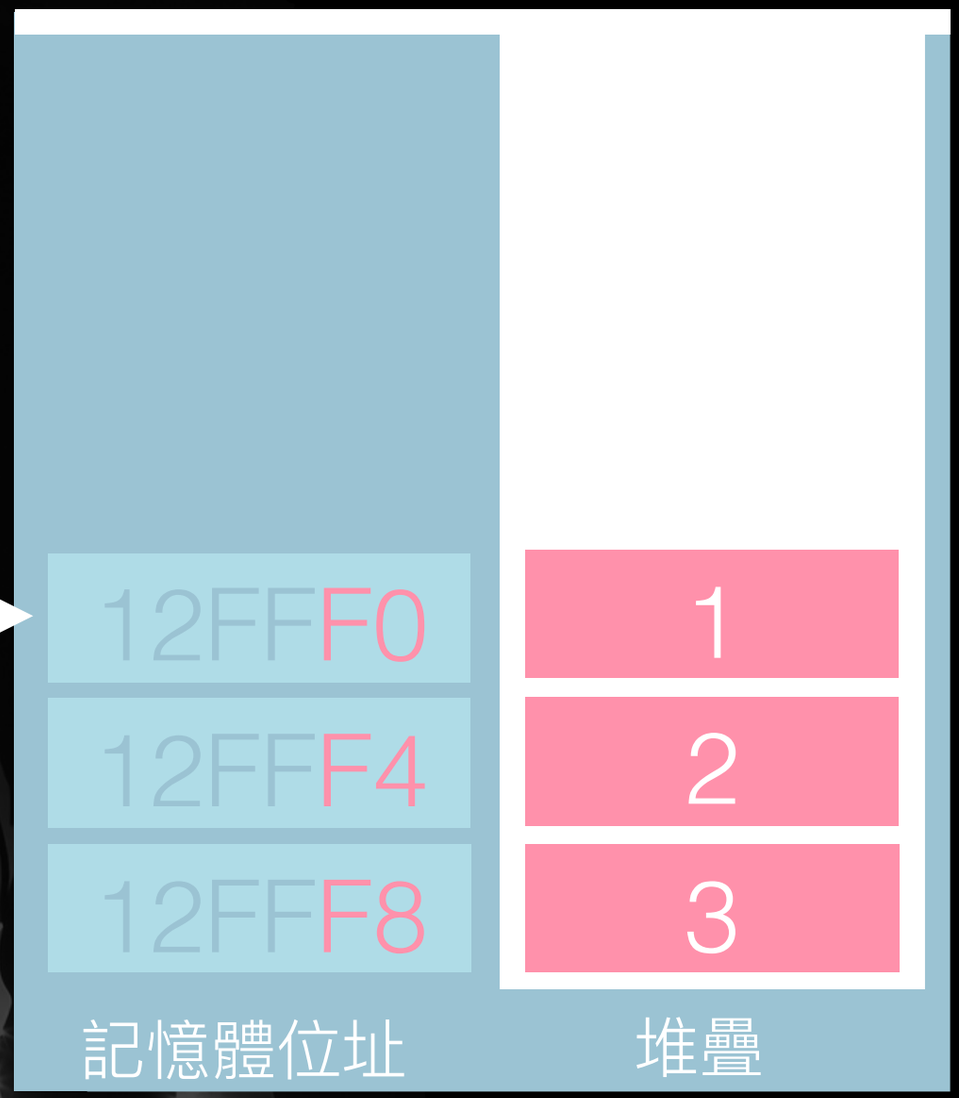
```
push 3
push 2
push 1
call add
add  esp,0x0C
//add(1, 2, 3)
```



呼叫約制

Calling Convention

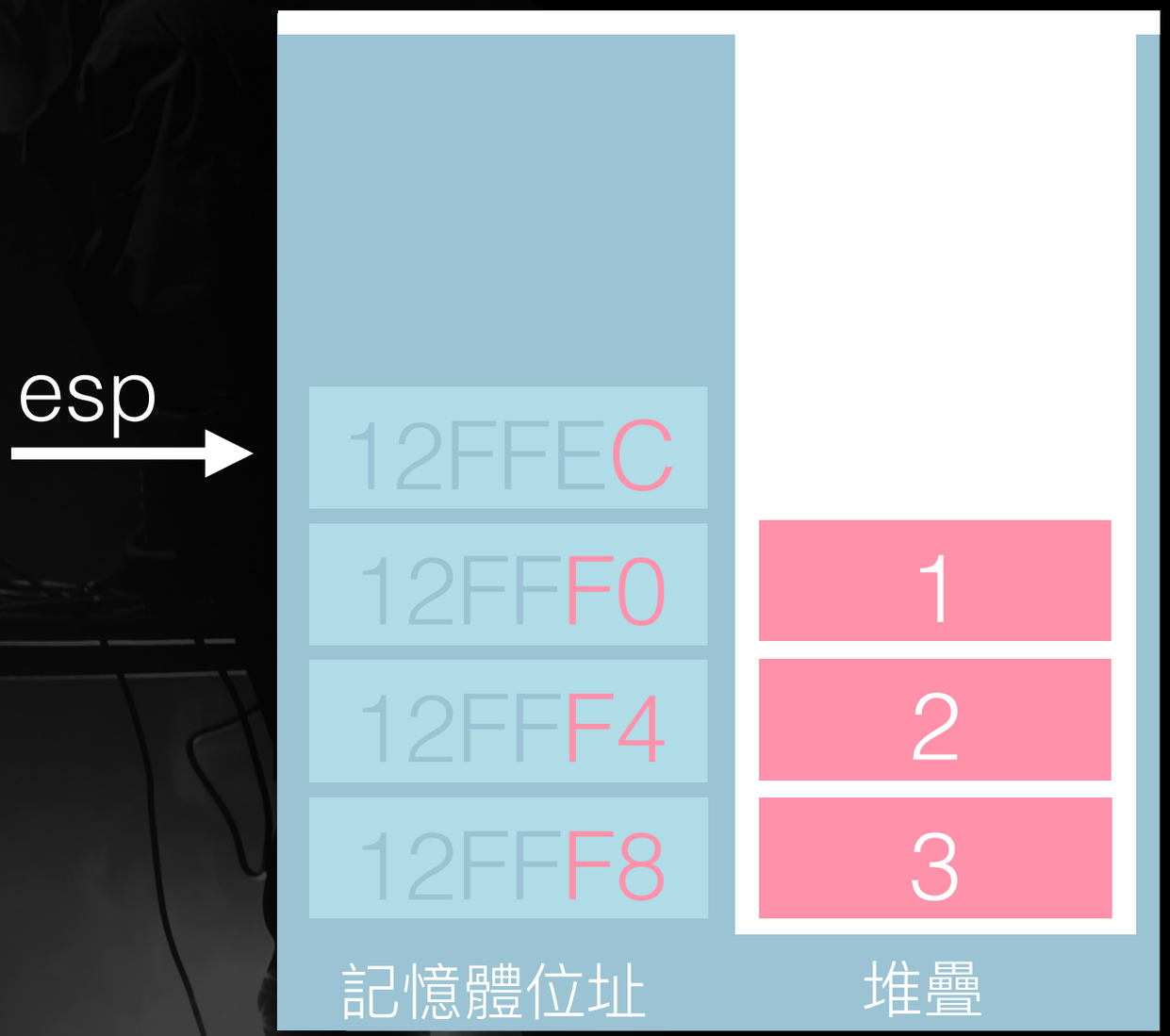
```
push 3
push 2
push 1
call add
add  esp,0x0C
//add(1, 2, 3)
```



呼叫約制

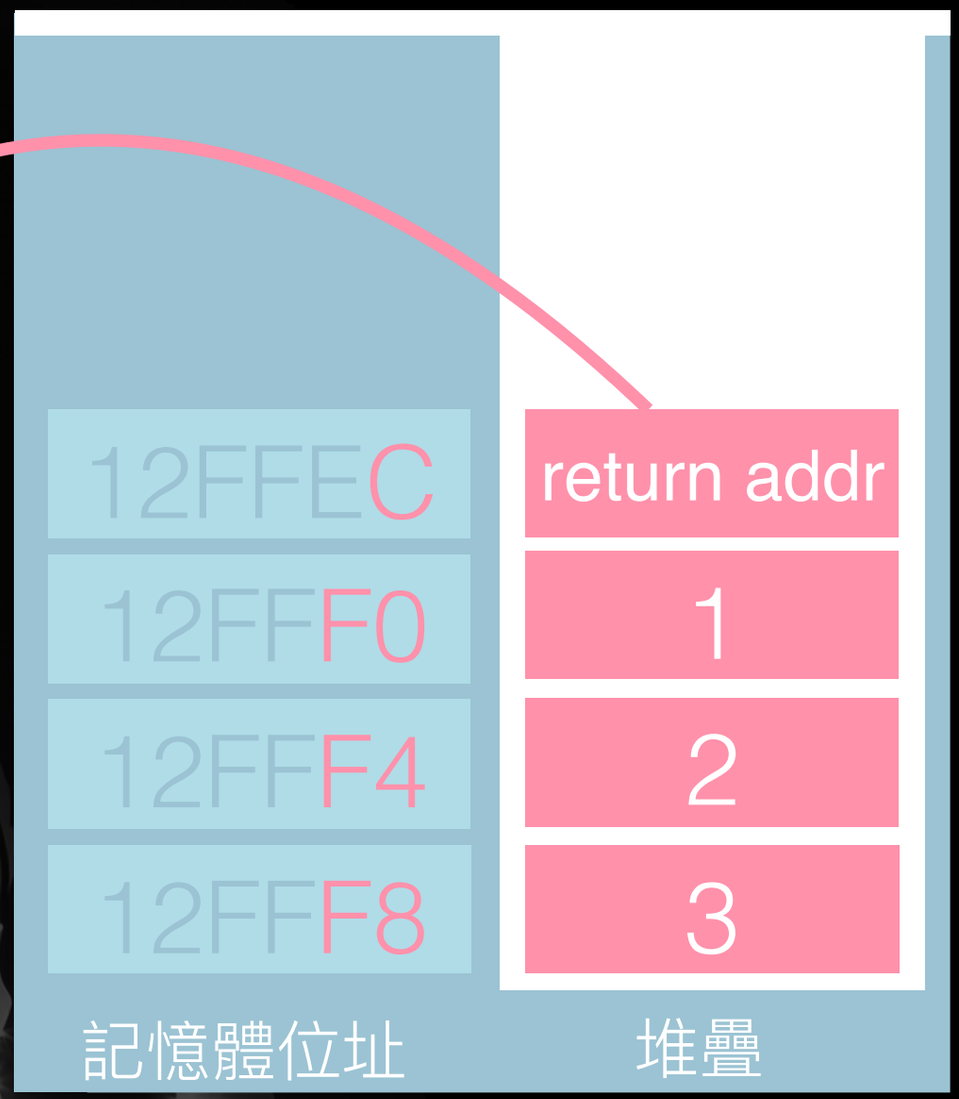
Calling Convention

```
push 3
push 2
push 1
call add
add  esp,0x0C
//add(1, 2, 3)
```



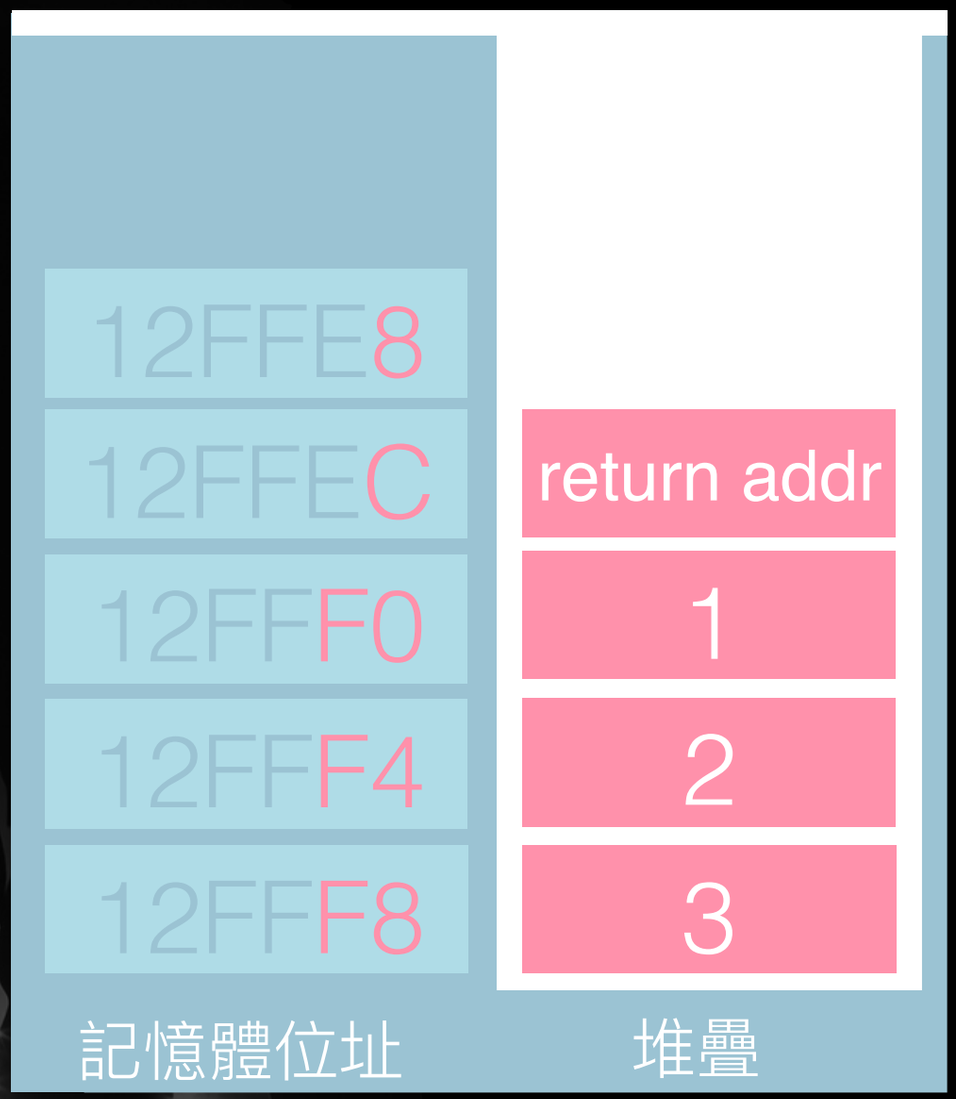
呼叫約制
Calling Convention

```
push 3
push 2
push 1
call add
add  esp,0x0C
//add(1, 2, 3)
```



呼叫約制
Calling Convention

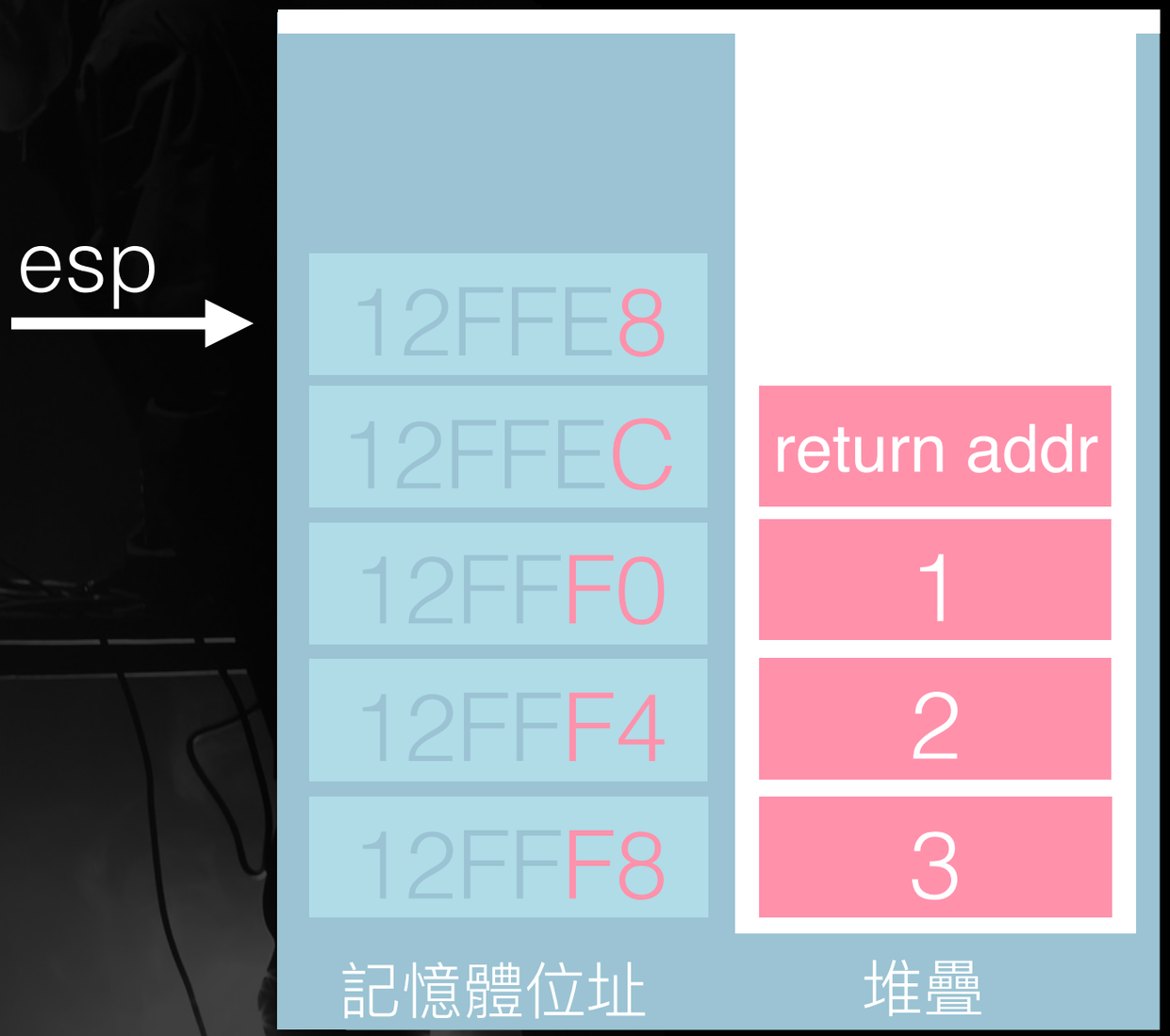
```
push  3
push  2
push  1
call  add
add   esp,0x0C
//add(1, 2, 3)
```



呼叫約制

Calling Convention

```
add:
push ebp
mov  ebp, esp
sub  esp, 0x04
mov  eax, [ebp+0x08]
add  eax, [ebp+0x0C]
add  eax, [ebp+0x10]
mov  [ebp-0x04], eax
mov  eax, [ebp-0x04]
mov  esp, ebp
pop  ebp
ret
```



呼叫約制

Calling Convention

```
add:
push ebp
mov  ebp, esp
sub  esp, 0x04
mov  eax, [ebp+0x08]
add  eax, [ebp+0x0C]
add  eax, [ebp+0x10]
mov  [ebp-0x04], eax
mov  eax, [ebp-0x04]
mov  esp, ebp
pop  ebp
ret
```

呼叫約制

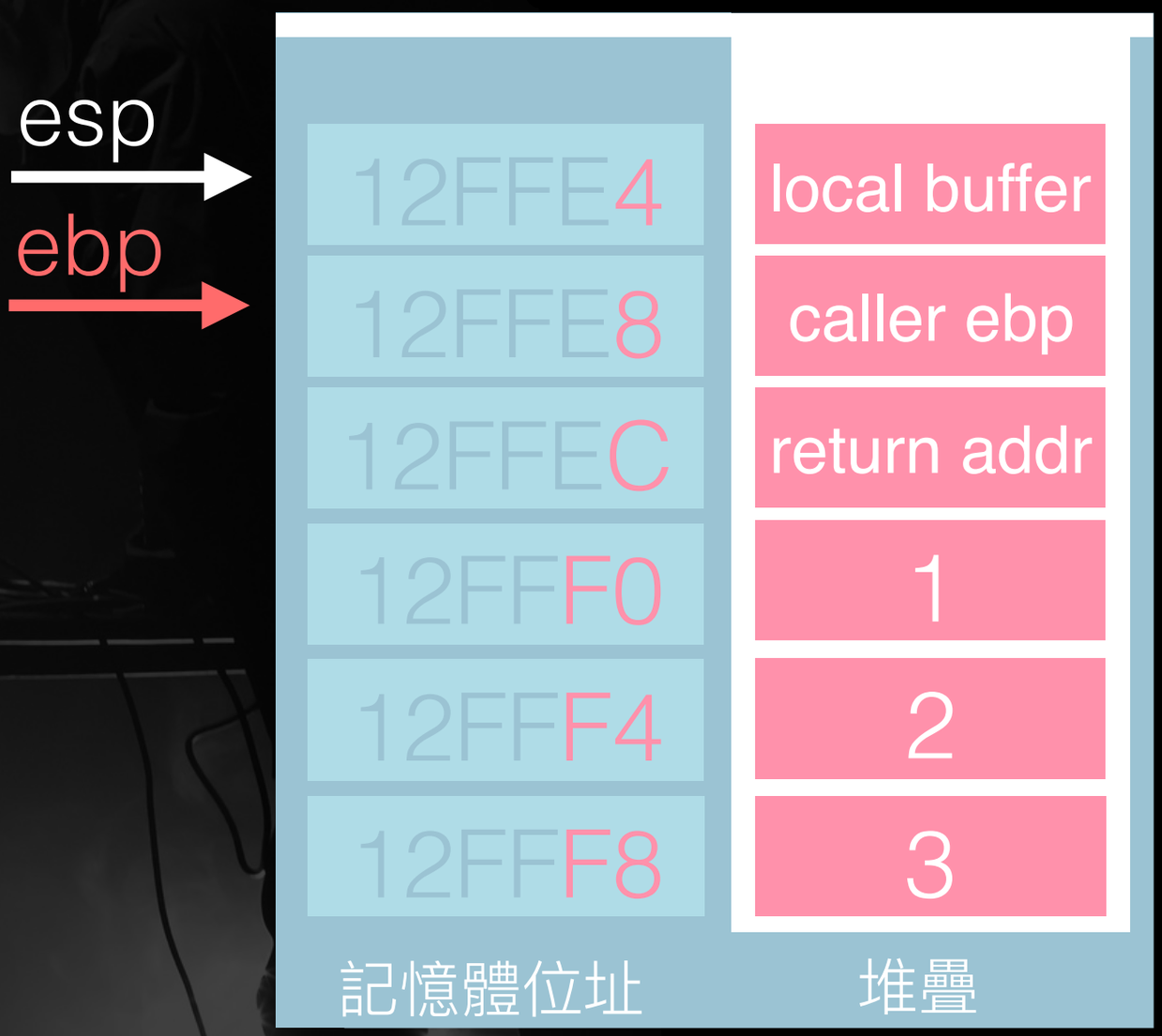
Calling Convention

```
add:
push ebp
mov  ebp, esp
sub  esp, 0x04
mov  eax, [ebp+0x08]
add  eax, [ebp+0x0C]
add  eax, [ebp+0x10]
mov  [ebp-0x04], eax
mov  eax, [ebp-0x04]
mov  esp, ebp
pop  ebp
ret
```

呼叫約制

Calling Convention

```
add:
push ebp
mov  ebp, esp
sub  esp, 0x04
mov  eax, [ebp+0x08]
add  eax, [ebp+0x0C]
add  eax, [ebp+0x10]
mov  [ebp-0x04], eax
mov  eax, [ebp-0x04]
mov  esp, ebp
pop  ebp
ret
```



呼叫約制
Calling Convention

```
add:
push ebp
mov  ebp, esp
sub  esp, 0x04
mov  eax, [ebp+0x08]
add  eax, [ebp+0x0C]
add  eax, [ebp+0x10]
mov  [ebp-0x04], eax
mov  eax, [ebp-0x04]
mov  esp, ebp
pop  ebp
ret
```

呼叫約制

Calling Convention

```
add:
push ebp
mov  ebp, esp
sub  esp, 0x04
mov  eax, [ebp+0x08]
add  eax, [ebp+0x0C]
add  eax, [ebp+0x10]
mov  [ebp-0x04], eax
mov  eax, [ebp-0x04]
mov  esp, ebp
pop  ebp
ret
```

呼叫約制

Calling Convention

```
add:
push ebp
mov  ebp, esp
sub  esp, 0x04
mov  eax, [ebp+0x08]
add  eax, [ebp+0x0C]
add  eax, [ebp+0x10]
mov  [ebp-0x04], eax
mov  eax, [ebp-0x04]
mov  esp, ebp
pop  ebp
ret
```

呼叫約制

Calling Convention

```
add:
push ebp
mov  ebp, esp
sub  esp, 0x04
mov  eax, [ebp+0x08]
add  eax, [ebp+0x0C]
add  eax, [ebp+0x10]
mov  [ebp-0x04], eax
mov  eax, [ebp-0x04]
mov  esp, ebp
pop  ebp
ret
```

呼叫約制

Calling Convention

```
add:
push ebp
mov  ebp, esp
sub  esp, 0x04
mov  eax, [ebp+0x08]
add  eax, [ebp+0x0C]
add  eax, [ebp+0x10]
mov  [ebp-0x04], eax
mov  eax, [ebp-0x04]
mov  esp, ebp
pop  ebp
ret
```

呼叫約制

Calling Convention

```
add:
push ebp
mov  ebp, esp
sub  esp, 0x04
mov  eax, [ebp+0x08]
add  eax, [ebp+0x0C]
add  eax, [ebp+0x10]
mov  [ebp-0x04], eax
mov  eax, [ebp-0x04]
mov  esp, ebp
pop  ebp
ret
```

EAX: 1+2+3

記憶體位址	堆疊
12FFE4	6
12FFE8	caller ebp
12FFEC	return addr
12FFF0	1
12FFF4	2
12FFF8	3

呼叫約制

Calling Convention

```
add:
push ebp
mov  ebp, esp
sub  esp, 0x04
mov  eax, [ebp+0x08]
add  eax, [ebp+0x0C]
add  eax, [ebp+0x10]
mov  [ebp-0x04], eax
mov  eax, [ebp-0x04]
mov  esp, ebp
pop  ebp
ret
```

EAX: 1+2+3

記憶體位址	堆疊
12FFE4	6
12FFE8	caller ebp
12FFEC	return addr
12FFF0	1
12FFF4	2
12FFF8	3

呼叫約制

Calling Convention

```
add:
push ebp
mov  ebp, esp
sub  esp, 0x04
mov  eax, [ebp+0x08]
add  eax, [ebp+0x0C]
add  eax, [ebp+0x10]
mov  [ebp-0x04], eax
mov  eax, [ebp-0x04]
mov  esp, ebp
pop  ebp
ret
```

EBP: caller ebp

EAX: 1+2+3

記憶體位址	堆疊
12FFE4	6
12FFE8	caller ebp
12FFEC	return addr
12FFF0	1
12FFF4	2
12FFF8	3

呼叫約制

Calling Convention

EIP: return addr | EBP: caller ebp | EAX: 1+2+3

```
add:
push ebp
mov  ebp, esp
sub  esp, 0x04
mov  eax, [ebp+0x08]
add  eax, [ebp+0x0C]
add  eax, [ebp+0x10]
mov  [ebp-0x04], eax
mov  eax, [ebp-0x04]
mov  esp, ebp
pop  ebp
ret
```

esp →

呼叫約制

Calling Convention

```
push  3
push  2
push  1
call  add
add   esp,0x0C
//add(1, 2, 3)
```

呼叫約制

Calling Convention

```
push 3
push 2
push 1
call add
add  esp,0x0C
//add(1, 2, 3)
```

EAX: 1+2+3

記憶體位址	堆疊
12FFE4	6
12FFE8	caller ebp
12FFEC	return addr
12FFF0	1
12FFF4	2
12FFF8	3

呼叫約制

Calling Convention

```
push 3
push 2
push 1
call add
add  esp,0x0C
//add(1, 2, 3)
```

懶人筆記

1. esp 永遠指向 Stack 最高處
2. [ebp] 保存呼叫者 base pointer
3. [ebp+4] 指向 return address
4. [ebp+8+4*0] 為第一個參數
   [ebp+8+4*1] 為第二個參數
   [ebp+8+4*2] 為第三個參數
   ...etc
5. 若區域變數佔用的空間必須
   padding 為 4 的倍數

懶人筆記

```
void func
(
  int arg1,
  int* arg2,
  char* arg3
){
  int buffer = 0;
  int arr[3];
  char data[2];
  /* do something */
}
```

```c
int add(int a, int b) {
  int tmp = a + b;
  return tmp;
}
add(1, 2);
```

練習 II:
嘗試撰寫出組合語言

http://carlosrafaelgn.com.br/asm86

看懂了嗎 :)

```
.text:00401006                          mov     [ebp+var_8], 0
.text:0040100D                          mov     [ebp+var_4], 0
.text:00401014                          mov     [ebp+var_8], 1
.text:0040101B                          jmp     short loc_401026
.text:0040101D loc_40101D:
.text:0040101D                          mov     eax, [ebp+var_8]
.text:00401020                          add     eax, 1
.text:00401023                          mov     [ebp+var_8], eax
.text:00401026 loc_401026:
.text:00401026                          mov     ecx, [ebp+var_8]
.text:00401029                          cmp     ecx, [ebp+arg_0]
.text:0040102C                          jg      short loc_401039
.text:0040102E                          mov     edx, [ebp+var_4]
.text:00401031                          add     edx, [ebp+var_8]
.text:00401034                          mov     [ebp+var_4], edx
.text:00401037                          jmp     short loc_40101D
.text:00401039 loc_401039:
.text:00401039                          mov     eax, [ebp+var_4]
.text:0040103C                          push    eax
.text:0040103D                          push    offset aSumI    ; "sum: %i\n"
.text:00401042                          call    _printf
```

逆向工程基礎（五）

Immunity Debugger

aaaddress1 // hackingWeekend

Immunity Debugger

The best of both worlds

GUI **and** Command line

IMMUNITY
DEBUGGER

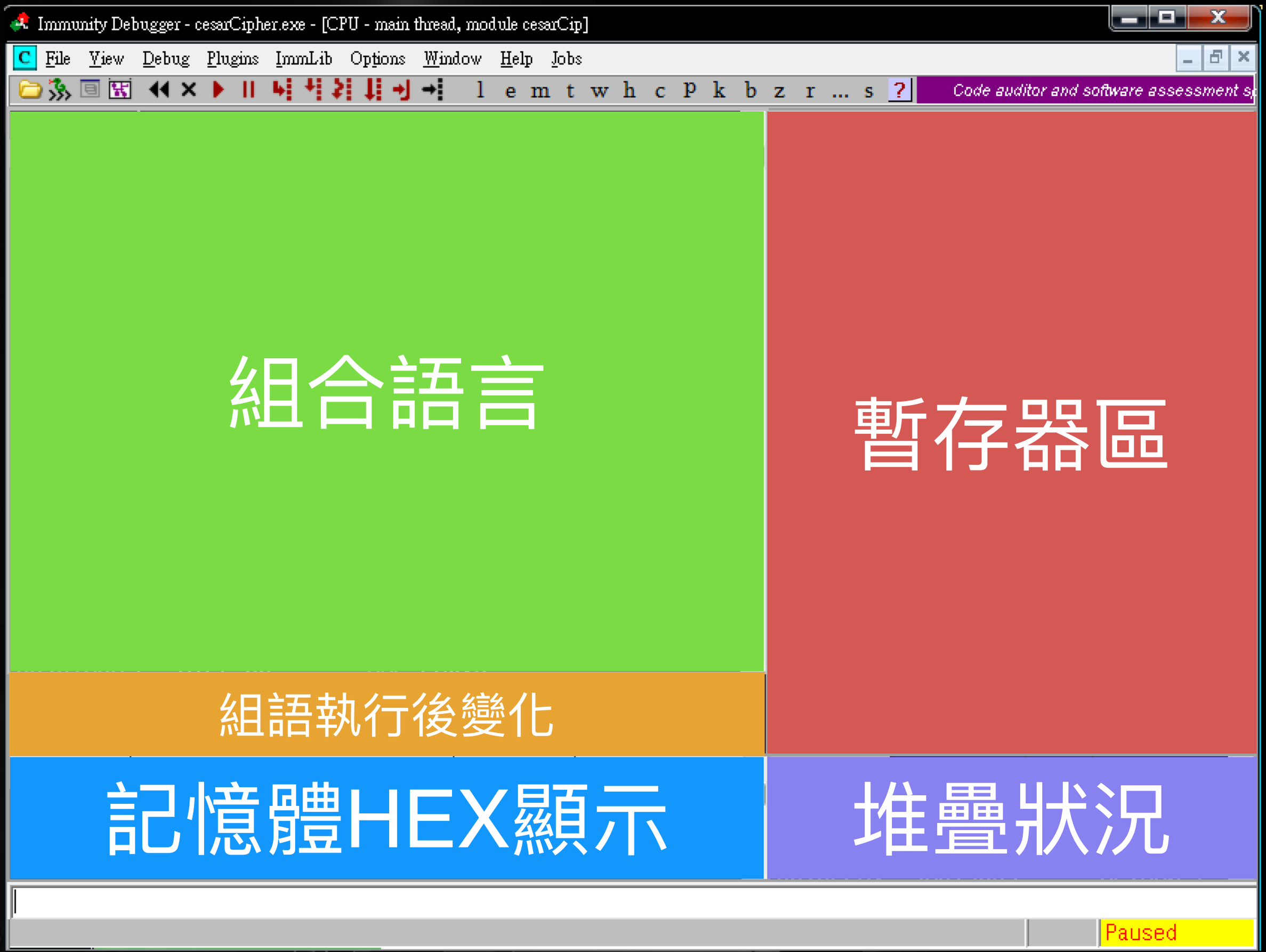
Debugger Overview
Job Ads in Debugger

DEBUGGER

Immunity Debugger is a powerful new way to write exploits, analyze malware, and reverse engineer binary files. It builds on a solid user interface with function graphing, the

Immunity Debugger

- 由 OllyDBG v1 改來的
- 支持 Python 的 Debugger，並且可以運行 PyCommand。允許自撰 Python 插件，Immunity 有提供 API 給 Python 用
- 原生對於 Exploit 上分析記憶體提供的功能比起 OllyDBG 好用很多（比如說：SEH Chain）
- 很多奇怪的 OllyDBG 的 BUG 都被修掉了
- ~~爽啦，介面漂亮就是爽，我才不屑用 OllyDBG，我們要教潮到出水的東西，才不要用老古董。~~


Immunity Debugger - cesarCipher.exe - [CPU - main thread, module cesarCip]
File View Debug Plugins ImmLib Options Window Help Jobs
Code auditor and software assessment s
組合語言
暫存器區
組語執行後變化
記憶體HEX顯示
堆疊狀況
Paused

Immunity Debugger - cesarCipher.exe - [CPU - main thread, module cesarCip]

File View Debug Plugins ImmLib Options Window Help Jobs

```
00401CBA  . 68 E48C4000     PUSH cesarCip.00408CE4
00401CBF  . 64:A1 0000000(  MOV EAX,DWORD PTR FS:[0]
00401CC5  . 50              PUSH EAX
00401CC6  . 64:8925 00000(  MOV DWORD PTR FS:[0],ESP
00401CCD  . 83C4 F0         ADD ESP,-10
00401CD0  . 53              PUSH EBX
00401CD1  . 56              PUSH ESI
00401CD2  . 57              PUSH EDI
00401CD3  . 8965 E8         MOV DWORD PTR SS:[EBP-18],ES
00401CD6  . FF15 7CC14300   CALL DWORD PTR DS:[<&KERNEL3
00401CDC  . A3 B09F4300     MOV DWORD PTR DS:[439FB0],EA
00401CE1  . A1 B09F4300     MOV EAX,DWORD PTR DS:[439FB0
00401CE6  . C1E8 08         SHR EAX,8
00401CE9  . 25 FF000000     AND EAX,0FF
00401CEE  . A3 BC9F4300     MOV DWORD PTR DS:[439FBC],EA
00401CF3  . 8B0D B09F4300   MOV ECX,DWORD PTR DS:[439FB0
00401CF9  . 81E1 FF000000   AND ECX,0FF
00401CFF  . 890D B89F4300   MOV DWORD PTR DS:[439FB8],EC
00401D05  . 8B15 B89F4300   MOV EDX,DWORD PTR DS:[439FB8
00401D0B    C1E2 08         SHL EDX,8
```

```
Registers (FPU)
EAX 00000000
ECX 0012FFB0
EDX 7C92E4F4 ntdll.KiFastSystemCallR
EBX 7FFDA000
ESP 0012FFB4
EBP 0012FFC0
ESI 00790074
EDI 0069006E

EIP 00401CBF cesarCip.00401CBF

C 0  ES 0023 32bit 0(FFFFFFFF)
P 1  CS 001B 32bit 0(FFFFFFFF)
A 0  SS 0023 32bit 0(FFFFFFFF)
Z 1  DS 0023 32bit 0(FFFFFFFF)
S 0  FS 003B 32bit 7FFDF000(FFF)
T 0  GS 0000 NULL
D 0
O 0  LastErr ERROR_SUCCESS (0000000(

EFL 00000246 (NO,NB,E,BE,NS,PE,GE,LE

ST0 empty
```

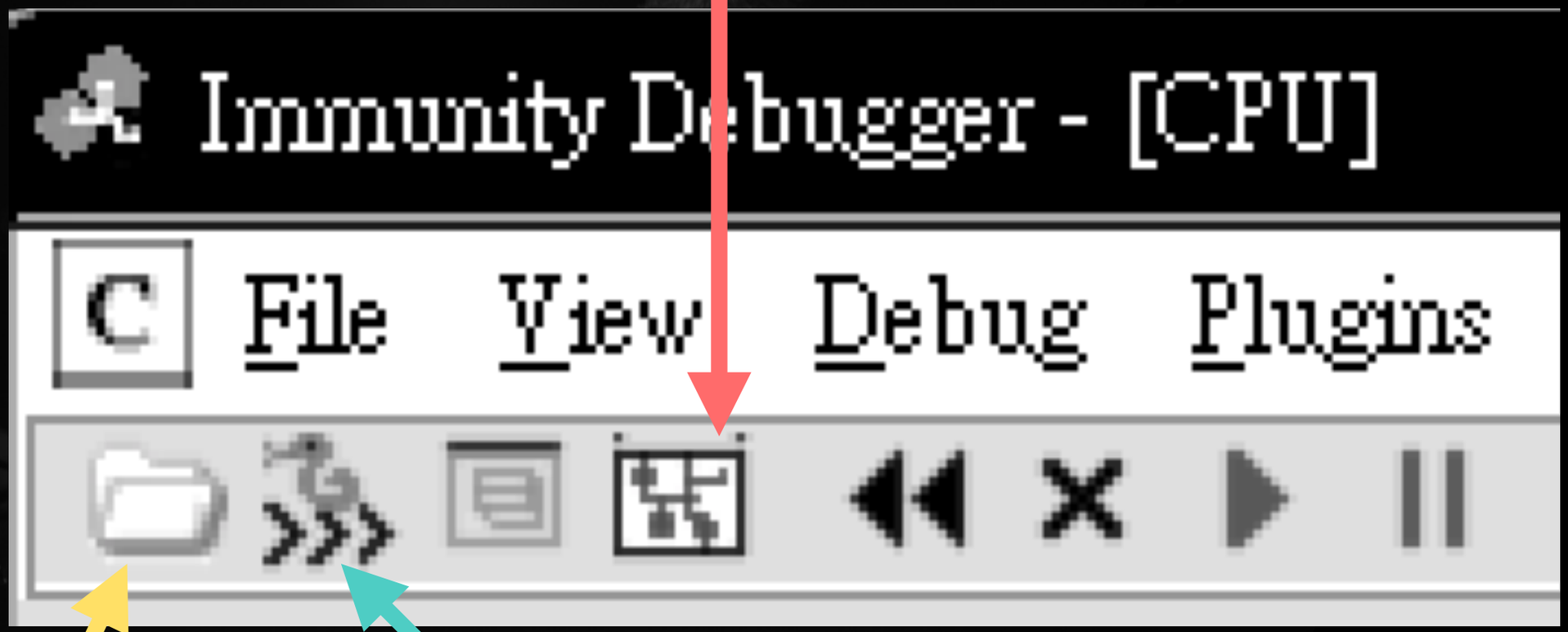
```
FS:[00000000]=[7FFDF000]=0012FFE0
EAX=00000000
```

Address	Hex dump	ASCII
00438000	00 00 00 00 00 00 00 00	....
00438008	00 00 00 00 00 00 00 00	....
00438010	00 00 00 00 00 00 00 00	....

```
0012FFB4  00408CE4  ?@.  Entry addr
0012FFB8  00434340  @CC. cesarCip.(
0012FFBC  FFFFFFFF  ████
0012FFC0  0012FFF0  ?↕.
```

Paused

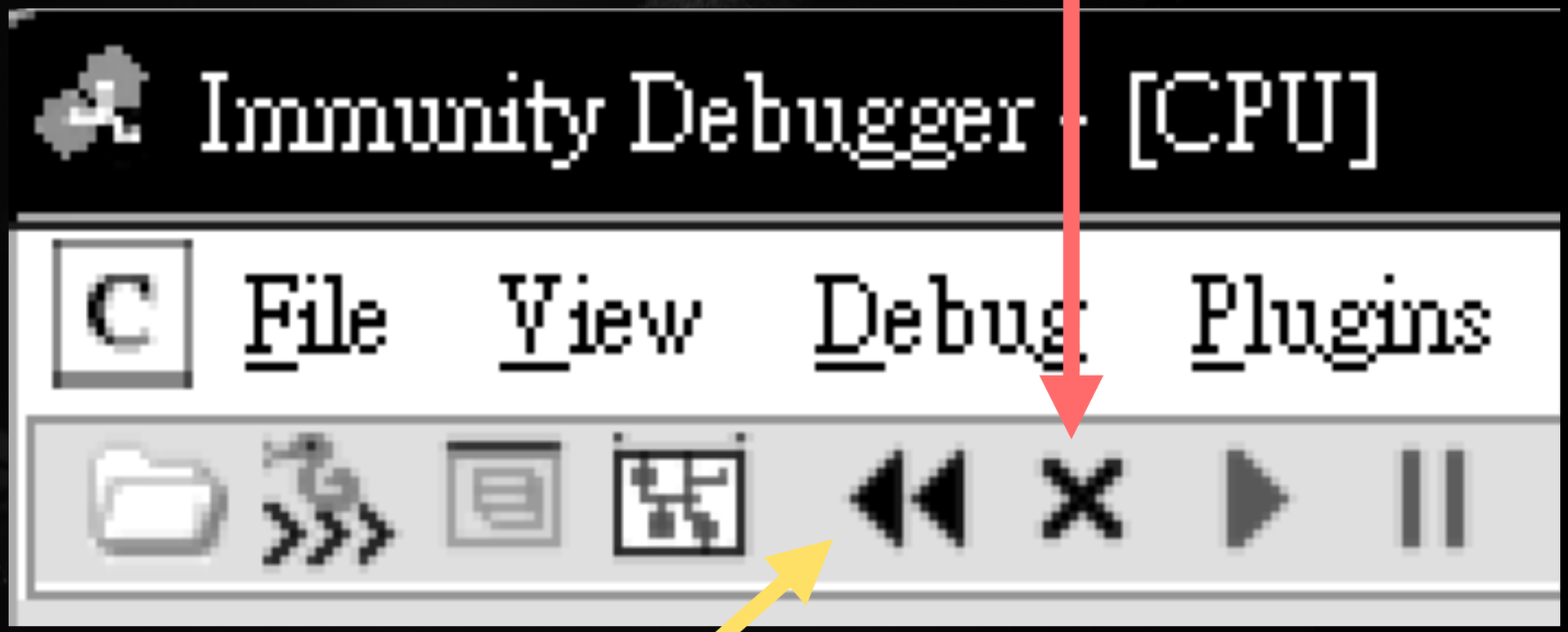
繪出當層函數的流程圖



打開 Python 終端機

創建一支 *.exe 為 Process

Kill 掉當前 Process



重新創一個 Process

調試 DEBUG

```
00401CBA  . 68 E48C4000     PUSH cesarCip.00408CE4
00401CBF  . 64:A1 0000000(  MOV EAX,DWORD PTR FS:[0]
00401CC5  . 50              PUSH EAX
00401CC6  . 64:8925 00000(  MOV DWORD PTR FS:[0],ESP
00401CCD  . 83C4 F0         ADD ESP,-10
00401CD0  . 53              PUSH EBX
00401CD1  . 56              PUSH ESI
00401CD2  . 57              PUSH EDI
00401CD3  . 8965 E8         MOV DWORD PTR SS:[EBP-18],ES
00401CD6  . FF15 7CC14300   CALL DWORD PTR DS:[<&KERNEL3
00401CDC  . A3 B09F4300     MOV DWORD PTR DS:[439FB0],EA
00401CE1  . A1 B09F4300     MOV EAX,DWORD PTR DS:[439FB0
00401CE6  . C1E8 08         SHR EAX,8
00401CE9  . 25 FF000000     AND EAX,0FF
00401CEE  . A3 BC9F4300     MOV DWORD PTR DS:[439FBC],EA
00401CF3  . 8B0D B09F4300   MOV ECX,DWORD PTR DS:[439FB0
00401CF9  . 81E1 FF000000   AND ECX,0FF
00401CFF  . 890D B89F4300   MOV DWORD PTR DS:[439FB8],EC
00401D05  . 8B15 B89F4300   MOV EDX,DWORD PTR DS:[439FB8
00401D0B    C1E2 08         SHL EDX,8
```

1. F7 單步：
   執行下一行

2. F8 躍過：
   執行下一行（遇到 call 不會執行進去）

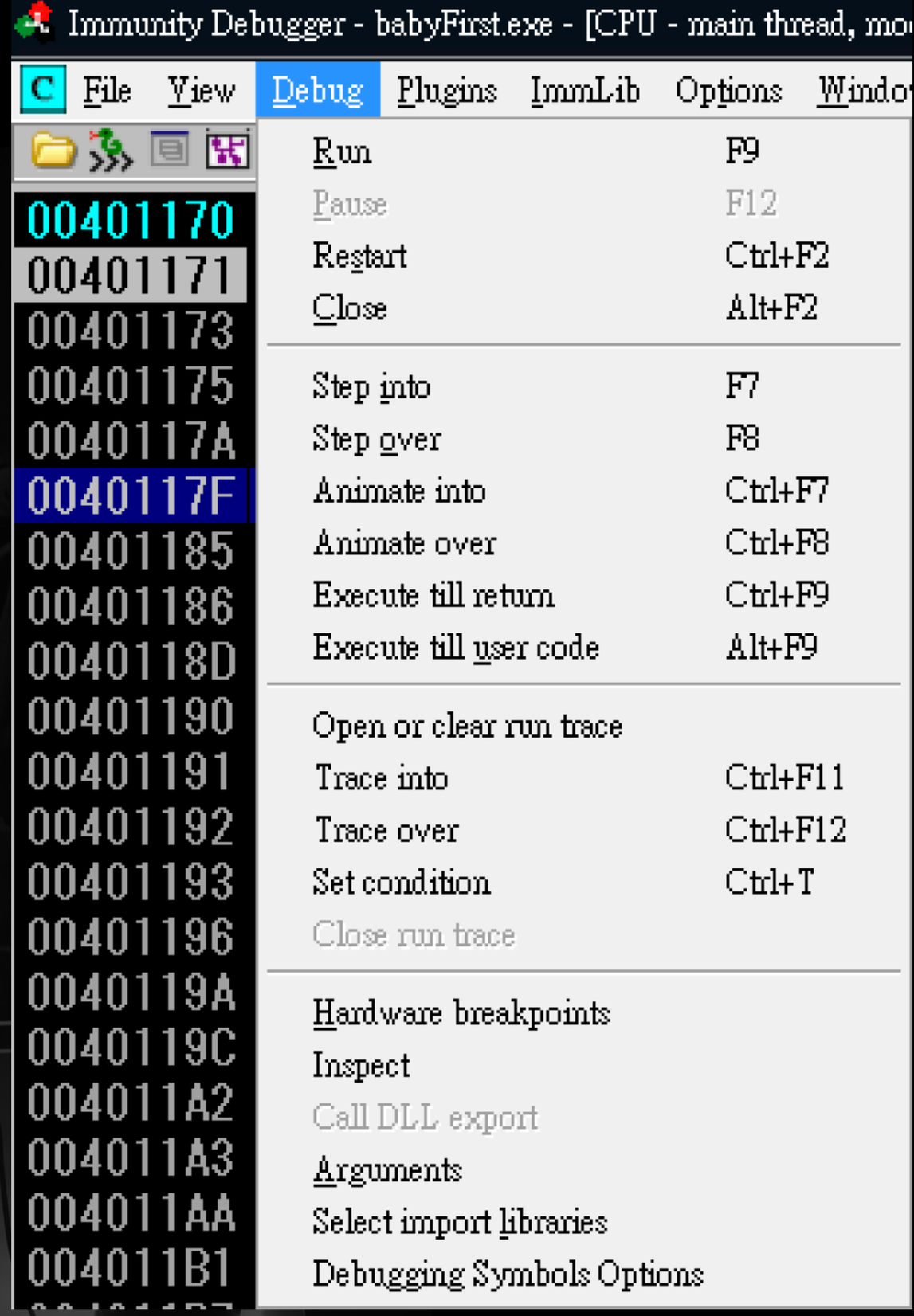
3. F2 設置一個斷點：
   當 Thread 運行到斷點時，會暫停，給你接手處理

4. 我猜你們應該今天只用到這麼多（？）

aaaddress1 // hackingWeekend

調試 DEBUG

忘記熱鍵的話，從選單列中
的 Debug 就可以看到熱鍵了
(我很忘記熱鍵XD)



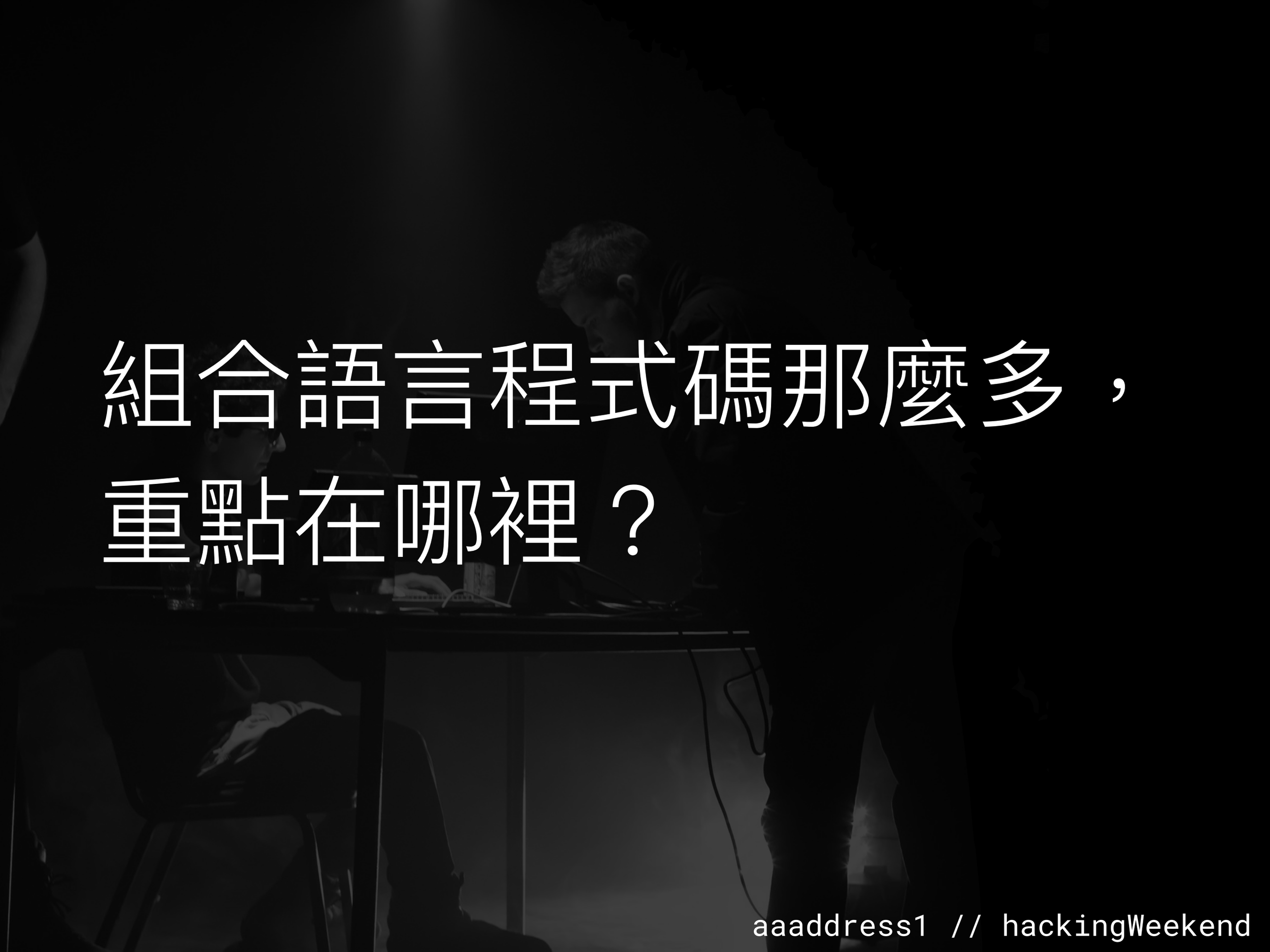
調試實戰分析篇章

aaaddress1 // hackingWeekend

調試實戰分析 I: babyFirst

aaaddress1 // hackingWeekend

大家第一次用調試器後，
即使剛剛把所有功能介紹完，
但你們一定會立刻發現一個很嚴重的問題



組合語言程式碼那麼多，重點在哪裡？

main function

```
0040120C | . 8945 D8         MOV DWORD PTR SS:[EBP-28],EAX
0040120F | . 8D45 D8         LEA EAX,DWORD PTR SS:[EBP-28]
00401212 | . 50              PUSH EAX
00401213 | . FF35 44304000   PUSH DWORD PTR DS:[403044]
00401219 | . 8D45 E0         LEA EAX,DWORD PTR SS:[EBP-20]
0040121C | . 50              PUSH EAX
0040121D | . 8D45 D4         LEA EAX,DWORD PTR SS:[EBP-2C]
00401220 | . 50              PUSH EAX
00401221 | . 8D45 E4         LEA EAX,DWORD PTR SS:[EBP-1C]
00401224 | . 50              PUSH EAX
00401225 | . FF15 34204000   CALL DWORD PTR DS:[<&MSVCRT.__getmainar  msvcrt.__getmainargs
0040122B | . 68 0C304000     PUSH babyFirs.0040300C
00401230 | . 68 00304000     PUSH babyFirs.00403000
00401235 | . E8 52000000     CALL <JMP.&MSVCRT._initterm>
0040123A | . FF15 30204000   CALL DWORD PTR DS:[<&MSVCRT.__p___initer  msvcrt.__p___initenv
00401240 | . 8B4D E0         MOV ECX,DWORD PTR SS:[EBP-20]
00401243 | . 8908            MOV DWORD PTR DS:[EAX],ECX
00401245 | . FF75 E0         PUSH DWORD PTR SS:[EBP-20]                Arg3
00401248 | . FF75 D4         PUSH DWORD PTR SS:[EBP-2C]                Arg2
0040124B | . FF75 E4         PUSH DWORD PTR SS:[EBP-1C]                Arg1
0040124E | . E8 82FEFFFF     CALL babyFirs.004010D5                    babyFirs.004010D5
```

函數頭、尾巴

```
004010D5  $ 55            PUSH EBP
004010D6  . 8BEC          MOV EBP,ESP
004010D8  . 6A 03         PUSH 3                          ┌Arg3 = 00000003
004010DA  . 6A 02         PUSH 2                          │Arg2 = 00000002
004010DC  . 6A 01         PUSH 1                          │Arg1 = 00000001
004010DE  . E8 9BFFFFFF   CALL babyFirs.0040107E          └babyFirs.0040107E
004010E3  . 83C4 0C       ADD ESP,0C
004010E6  . 50            PUSH EAX                        ┌Arg1
004010E7  . E8 A9FFFFFF   CALL babyFirs.00401095          └babyFirs.00401095
004010EC  . 83C4 04       ADD ESP,4
004010EF  . 50            PUSH EAX                        ┌<%i>
004010F0  . 68 20304000   PUSH babyFirs.00403020          │format = "sum: %i "
004010F5  . E8 70000000   CALL <JMP.&MSVCRT.printf>       └printf
004010FA  . 83C4 08       ADD ESP,8
004010FD  . 68 2C304000   PUSH babyFirs.0040302C          ┌command = "PAUSE"
00401102  . E8 5D000000   CALL <JMP.&MSVCRT.system>       └system
00401107  . 83C4 04       ADD ESP,4
0040110A  . 33C0          XOR EAX,EAX
0040110C  . 5D            POP EBP
0040110D  . C3            RETN
```

函數內容

```
004010D5  $ 55          PUSH EBP
004010D6  . 8BEC        MOV EBP,ESP
004010D8  . 6A 03       PUSH 3                                  ; Arg3 = 00000003
004010DA  . 6A 02       PUSH 2                                  ; Arg2 = 00000002
004010DC  . 6A 01       PUSH 1                                  ; Arg1 = 00000001
004010DE  . E8 9BFFFFFF CALL babyFirs.0040107E                  ; babyFirs.0040107E
004010E3  . 83C4 0C     ADD ESP,0C
004010E6  . 50          PUSH EAX                                ; Arg1
004010E7  . E8 A9FFFFFF CALL babyFirs.00401095                  ; babyFirs.00401095
004010EC  . 83C4 04     ADD ESP,4
004010EF  . 50          PUSH EAX                                ; <%i>
004010F0  . 68 20304000 PUSH babyFirs.00403020                  ; format = "sum: %i "
004010F5  . E8 70000000 CALL <JMP.&MSVCRT.printf>               ; printf
004010FA  . 83C4 08     ADD ESP,8
004010FD  . 68 2C304000 PUSH babyFirs.0040302C                  ; command = "PAUSE"
00401102  . E8 5D000000 CALL <JMP.&MSVCRT.system>               ; system
00401107  . 83C4 04     ADD ESP,4
0040110A  . 33C0        XOR EAX,EAX
0040110C  . 5D          POP EBP
0040110D  . C3          RETN
```

40107E

```
0040107E  ┌$ 55          PUSH EBP
0040107F  │. 8BEC        MOV EBP,ESP
00401081  │. 51          PUSH ECX
00401082  │. 8B45 08     MOV EAX,DWORD PTR SS:[EBP+8]
00401085  │. 0345 0C     ADD EAX,DWORD PTR SS:[EBP+C]
00401088  │. 0345 10     ADD EAX,DWORD PTR SS:[EBP+10]
0040108B  │. 8945 FC     MOV DWORD PTR SS:[EBP-4],EAX
0040108E  │. 8B45 FC     MOV EAX,DWORD PTR SS:[EBP-4]
00401091  │. 8BE5        MOV ESP,EBP
00401093  │. 5D          POP EBP
00401094  └. C3          RETN
```

401095

```
00401095 ┌$ 55              PUSH EBP
00401096 │. 8BEC            MOV EBP,ESP
00401098 │. 83EC 08         SUB ESP,8
0040109B │. C745 F8 00000(  MOV DWORD PTR SS:[EBP-8],0
004010A2 │. C745 FC 00000(  MOV DWORD PTR SS:[EBP-4],0
004010A9 │. C745 F8 01000(  MOV DWORD PTR SS:[EBP-8],1
004010B0 │. EB 09           JMP SHORT babyFirs.004010BB
004010B2 │> 8B45 F8         ┌MOV EAX,DWORD PTR SS:[EBP-8]
004010B5 │. 83C0 01         │ADD EAX,1
004010B8 │. 8945 F8         │MOV DWORD PTR SS:[EBP-8],EAX
004010BB │> 8B4D F8         │MOV ECX,DWORD PTR SS:[EBP-8]
004010BE │. 3B4D 08         │CMP ECX,DWORD PTR SS:[EBP+8]
004010C1 │. 7F 0B           │JG SHORT babyFirs.004010CE
004010C3 │. 8B55 FC         │MOV EDX,DWORD PTR SS:[EBP-4]
004010C6 │. 0355 F8         │ADD EDX,DWORD PTR SS:[EBP-8]
004010C9 │. 8955 FC         │MOV DWORD PTR SS:[EBP-4],EDX
004010CC │.^EB E4           └JMP SHORT babyFirs.004010B2
004010CE │> 8B45 FC         MOV EAX,DWORD PTR SS:[EBP-4]
004010D1 │. 8BE5            MOV ESP,EBP
004010D3 │. 5D              POP EBP
004010D4 └. C3              RETN
```

print and pause

```
004010D5  ┌$ 55            PUSH EBP
004010D6  │. 8BEC          MOV EBP,ESP
004010D8  │. 6A 03         PUSH 3                                  ┌Arg3 = 00000003
004010DA  │. 6A 02         PUSH 2                                  │Arg2 = 00000002
004010DC  │. 6A 01         PUSH 1                                  │Arg1 = 00000001
004010DE  │. E8 9BFFFFFF   CALL babyFirs.0040107E                  └babyFirs.0040107E
004010E3  │. 83C4 0C       ADD ESP,0C
004010E6  │. 50            PUSH EAX                                ┌Arg1
004010E7  │. E8 A9FFFFFF   CALL babyFirs.00401095                  └babyFirs.00401095
004010EC  │. 83C4 04       ADD ESP,4
004010EF  │. 50            PUSH EAX                                ┌<%i>
004010F0  │. 68 20304000   PUSH babyFirs.00403020                  │format = "sum: %i "
004010F5  │. E8 70000000   CALL <JMP.&MSVCRT.printf>               └printf
004010FA  │. 83C4 08       ADD ESP,8
004010FD  │. 68 2C304000   PUSH babyFirs.0040302C                  ┌command = "PAUSE"
00401102  │. E8 5D000000   CALL <JMP.&MSVCRT.system>               └system
00401107  │. 83C4 04       ADD ESP,4
0040110A  │. 33C0          XOR EAX,EAX
0040110C  │. 5D            POP EBP
0040110D  └. C3            RETN
```

Live Demo

aaaddress1 // hackingWeekend

調試實戰分析 II: eggHunter

aaaddress1 // hackingWeekend

fopen != NULL

```
004010B2  PUSH eggHunte.00403020                 ┌mode = "w"
004010B7  PUSH eggHunte.00403024                 │path = "./4re_u_a_bunny/egg.txt"
004010BC  CALL DWORD PTR DS:[<&MSVCRT.fopen>]    └fopen
004010C2  ADD ESP,8
004010C5  MOV DWORD PTR SS:[EBP-1C],EAX
004010C8  CMP DWORD PTR SS:[EBP-1C],0
004010CC  JE SHORT eggHunte.00401115
004010CE  PUSH eggHunte.0040303C                 ┌s = "good job! you found my egg."
004010D3  CALL DWORD PTR DS:[<&MSVCRT.puts>]     └puts
004010D9  ADD ESP,4
```

Live Demo

aaaddress1 // hackingWeekend

調試實戰分析 III:

1nte1's AMT Login System

嘗試在不使用 Debugger 情況下登入成功

aaaddress1 // hackingWeekend

fget(user, 0x40)

```
004011B0 | . 68 8C314000    PUSH loginSys.0040318C                  ┌format = "User: "
004011B5 | . FF15 18204000 CALL DWORD PTR DS:[<&MSVCRT.printf>]    └printf
004011BB | . 83C4 04        ADD ESP,4
004011BE | . A1 28204000    MOV EAX,DWORD PTR DS:[<&MSVCRT._iob>]
004011C3 | . 50             PUSH EAX                               ┌stream => OFFSET MSVCRT._iob
004011C4 | . 6A 40          PUSH 40                                │n = 40 (64.)
004011C6 | . 8D4D C0        LEA ECX,DWORD PTR SS:[EBP-40]          │
004011C9 | . 51             PUSH ECX                               │s
004011CA | . FF15 24204000 CALL DWORD PTR DS:[<&MSVCRT.fgets>]     └fgets
004011D0 | . 83C4 0C        ADD ESP,0C
004011D3 | . 8D55 C0        LEA EDX,DWORD PTR SS:[EBP-40]
004011D6 | . 52             PUSH EDX                               ┌s
004011D7 | . E8 B0000000    CALL <JMP.&MSVCRT.strlen>              └strlen
004011DC | . 83C4 04        ADD ESP,4
004011DF | . C64405 BF 00   MOV BYTE PTR SS:[EBP+EAX-41],0
```

fget(password, 0x40)

```
004011E4 | . 68 94314000    PUSH loginSys.00403194                  ┌format = "Password: "
004011E9 | . FF15 18204000 CALL DWORD PTR DS:[<&MSVCRT.printf>]     └printf
004011EF | . 83C4 04        ADD ESP,4
004011F2 | . A1 28204000    MOV EAX,DWORD PTR DS:[<&MSVCRT._iob>]
004011F7 | . 50             PUSH EAX                                ┌stream => OFFSET MSVCRT._iob
004011F8 | . 6A 40          PUSH 40                                 │n = 40 (64.)
004011FA | . 8D4D 80        LEA ECX,DWORD PTR SS:[EBP-80]           │
004011FD | . 51             PUSH ECX                                │s
004011FE | . FF15 24204000 CALL DWORD PTR DS:[<&MSVCRT.fgets>]     └fgets
00401204 | . 83C4 0C        ADD ESP,0C
00401207 | . 8D55 80        LEA EDX,DWORD PTR SS:[EBP-80]
0040120A | . 52             PUSH EDX                                ┌s
0040120B | . E8 7C000000    CALL <JMP.&MSVCRT.strlen>               └strlen
00401210 | . 83C4 04        ADD ESP,4
00401213 | . C68405 7FFFFFI MOV BYTE PTR SS:[EBP+EAX-81],0
```

004010CD([ebp-40], [ebp-80])

```
LEA EAX,DWORD PTR SS:[EBP-80]
PUSH EAX
LEA ECX,DWORD PTR SS:[EBP-40]
PUSH ECX
CALL loginSys.004010CD
ADD ESP,8
```

```
┌Arg2
│
└Arg1
 loginSys.004010CD
```

strncmp(user, "root", 4)

```
004010D3 | . 6A 04          PUSH 4                                  ┌maxlen = 4
004010D5 | . 68 B0304000    PUSH loginSys.004030B0                  | s2 = "root"
004010DA | . 8B45 08        MOV EAX,DWORD PTR SS:[EBP+8]            |
004010DD | . 50             PUSH EAX                                | s1
004010DE | . FF15 1C204000  CALL DWORD PTR DS:[<&MSVCRT.strncmp>]   └strncmp
004010E4 | . 83C4 0C        ADD ESP,0C
004010E7 | . 85C0           TEST EAX,EAX
004010E9 | . 74 25          JE SHORT loginSys.00401110
```

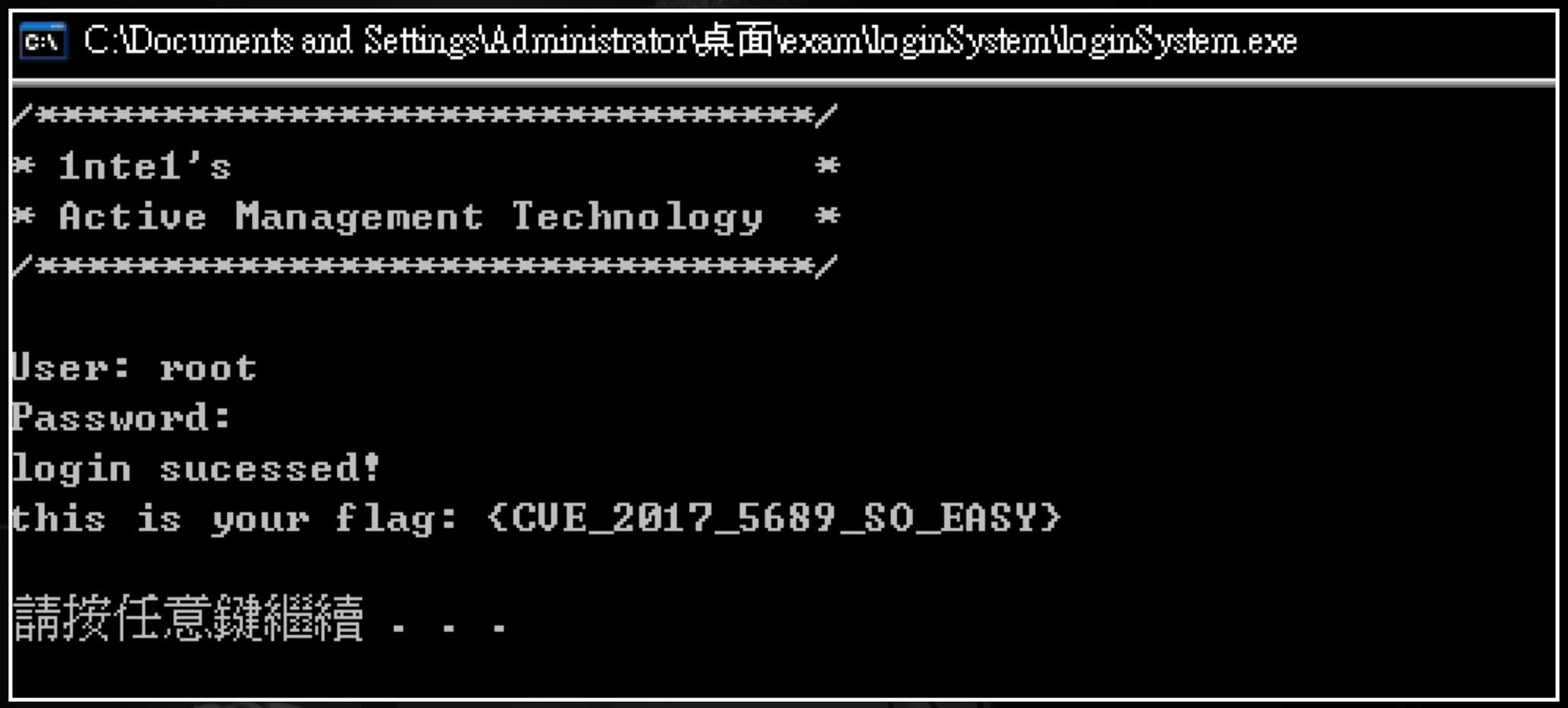
srand(time(NULL))
sprintf([ebp-8], "%i", rand())

```
00401110 | > 6A 00          PUSH 0                                  ┌timer = NULL
00401112 | . FF15 14204000 CALL DWORD PTR DS:[<&MSVCRT.time>]      └time
00401118 | . 83C4 04        ADD ESP,4
0040111B | . 50             PUSH EAX                                ┌seed
0040111C | . FF15 3C204000 CALL DWORD PTR DS:[<&MSVCRT.srand>]     └srand
00401122 | . 83C4 04        ADD ESP,4
00401125 | . FF15 68204000 CALL DWORD PTR DS:[<&MSVCRT.rand>]      ┌rand
0040112B | . 50             PUSH EAX                                ┌<%i>
0040112C | . 68 08314000    PUSH loginSys.00403108                  │format = "%i"
00401131 | . 8D55 F8        LEA EDX,DWORD PTR SS:[EBP-8]            │
00401134 | . 52             PUSH EDX                                │s
00401135 | . FF15 6C204000 CALL DWORD PTR DS:[<&MSVCRT.sprintf>]   └sprintf
0040113B | . 83C4 0C        ADD ESP,0C
```

strncmp(password, [ebp-8], strlen(password))

```
0040113E | . 8B45 0C        MOV EAX,DWORD PTR SS:[EBP+C]
00401141 | . 50             PUSH EAX                                 ┌s
00401142 | . E8 45010000    CALL <JMP.&MSVCRT.strlen>                └strlen
00401147 | . 83C4 04        ADD ESP,4
0040114A | . 50             PUSH EAX                                 ┌maxlen
0040114B | . 8D4D F8        LEA ECX,DWORD PTR SS:[EBP-8]             │
0040114E | . 51             PUSH ECX                                 │s2
0040114F | . 8B55 0C        MOV EDX,DWORD PTR SS:[EBP+C]             │
00401152 | . 52             PUSH EDX                                 │s1
00401153 | . FF15 1C204000  CALL DWORD PTR DS:[<&MSVCRT.strncmp>]    └strncmp
00401159 | . 83C4 0C        ADD ESP,0C
0040115C | . 85C0           TEST EAX,EAX
0040115E | . 74 22          JE SHORT loginSys.00401182
```

即使每次密碼都不一樣，
但只要輸入空密碼就進去惹



Live Demo

aaaddress1 // hackingWeekend

你覺得這麼智障的漏洞

怎麼會有人寫出來呢

aaaddress1 // hackingWeekend

這可是好蚌蚌 Intel 今年五月爆發的 0day 呢

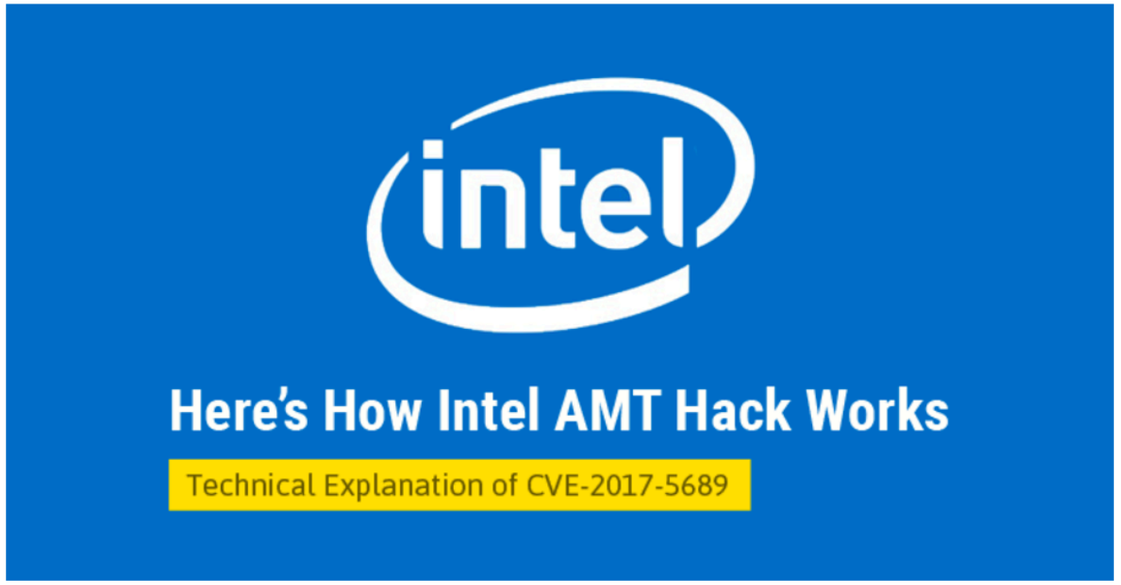
（搞不好你現在手上的筆電就可以被我當遙控飛機玩喔）

The Hacker News™
Security in a serious way

Explained — How Intel AMT Vulnerability Allows to Hack Computers Remotely

Friday, May 05, 2017 Swati Khandelwal

G+1 69 Like 3.2 Share 9013 Tweet 4503 in Share 457 Share 14.4K



Earlier this week Intel announced a critical escalation of privilege bug that affects its remote management features shipping with Intel Server chipsets for past 7 years, which, if exploited, would allow a remote attacker to take control of vulnerable PCs, laptops, or servers.

The vulnerability, labeled **CVE-2017-5689**, affects Intel remote management technologies, including Active Management Technology (AMT), Intel Standard Manageability (ISM), and Intel Small Business Technology (SBT) software, versions 6 through 11.6.

The flaw was originally discovered by Maksim Malyutin, a member of Embedi research team, in mid-February, who then responsibly disclosed it to the Intel security team.

調試實戰分析 IV:

cesarCipher

!! 注意 !! Key 形式為 FLAG{XXXXXXXXXXXXXXX}

aaaddress1 // hackingWeekend

Live Demo

aaaddress1 // hackingWeekend

Live Demo

aaaddress1 // hackingWeekend

Bonus

~~一堆不是很重要但我很想講的東西~~

QA

aaaddress1@chroot.org