Open RazzmatazZ Laboratory (OrzLab)

http://orzlab.blogspot.com/

快快樂樂學 GNU Debugger (gdb) Part II - 實務與應用

July 12, 2008

Jim Huang(黃敬群 /**jserv**)
Email: <**jserv.tw@gmail.com**>
Blog: **http://blog.linux.org.tw/jserv/**

注意

- 簡報採用創意公用授權條款 (Creative Commons License: **Attribution-ShareAlike**) 發行
- 議程所用之軟體，依據個別授權方式發行
- 以 x86/IA32 平台為主
- 系統平台
  - Ubuntu interpid (development branch, **8.10**)
  - Linux kernel 2.6.26
  - gcc 4.3.1
  - glibc 2.8
  - gdb 6.8
- 部份基礎概念請參考「深入淺出 Hello World 」系列演講

大綱

- GDB 指令
- Stack Frame
- GDB 巨集處理

GDB 指令

gdb 基本指令

- (gdb) **run**
- (gdb) **next**
- (gdb) **step**
- (gdb) **print**
- (gdb) **continue**
- (gdb) **backtrace**
- (gdb) **finish**
- (gdb) **quit**
- (gdb) **info**

command	format	effect
run	run arg1 arg2 … < stdin	run program as if invoked by the shell
CTL-C	(control-C keystroke)	interrupt the running program
where	where	show the stack with line numbers
list	list	list the source code around the point of current interest
print	print expression	evaluate the expression in the current context and display the result
up	up	move the current context one frame up the stack
down	down	move the current context one frame back down the stack
break	break [function | n]	set a breakpoint at entry to the named function at line n in the current context
watch	watch expression	when begin running, keep evaluating the expression and stop if it becomes true
continue	continue	continue execution (after stopping at a breakpoint or watchpoint)
step	step	continue, but stop after executing just one source line
clear	clear [function | n]	clear the breakpoint set at the function or at line n
info	info break	show information about all breakpoints
help	help	display help information
quit	quit	quit GDB

gdb 指令：查驗記憶體

- (gdb) **print** 變數 | 表示式
- (gdb) **set print pretty** on/off
- (gdb) **display** 變數
- (gdb) **undisplay** 號碼

```
(gdb) p main
$1 = {int ()} 0x8048388 <main>
(gdb) p abs(-1)
$2 = 1
```

```
(gdb) p 100
$1 = 100
(gdb) p/x 100
$2 = 0x64
(gdb) p/o 100
$3 = 0144
(gdb) p/t 100
$4 = 1100100
(gdb) p/u 100
```

十進位

十六進位

八進位

二進位
t=two

十進位，無號數

歷史紀錄 (readline-style)
$ 最新
$$n 倒數第 n 個
(gdb) **show values** [n|+]

$1, $2, $3, … 爲歷史變數

```
(gdb) print exp
$4 = "35*2+10", '\000' <repeats 37 times>, …
(gdb) whatis exp
type = char [500]
(gdb) print exp[0]
$4 = 40 '('
(gdb) print exp[0]@5
$5 = "(20+5"
```

print **a[0] @ 10**
a[0] 與其後 10 個元素的值

```
(gdb) x/4db 0x8048576
0x8048576 <main+86>:    -24     9       -2      -1
(gdb) x/4xb 0x8048576
0x8048576 <main+86>:    0xe8    0x09    0xfe    0xff
(gdb) x/4ub 0x8048576
0x8048576 <main+86>:    232     9       254     255
(gdb) x/4ab 0x8048576
0x8048576 <main+86>:    0xffffffe8      0x9     0xfffffffe      0xffffffff
(gdb) x/2dw 0x8048576
0x8048576 <main+86>:    -128536 281314303
```

x = examine

- (gdb) **x/nfu** 位址
- 格式:
  - 印出 n 個資料項
  - f — 輸出格式

格式字元	輸出格式
x	十六進位
d	有負數之十進位
u	無負數之十進位
o	八進位
t	二進位
a	十六進位之位址格式
c	字元
f	浮點數

資料項單位字元	資料項單位
b	BYTE (1 byte)
n	DBYTE (2 bytes)
w	WORD (4 bytes)
g	DWORD (8 bytes)

```
(gdb) x/10s main
0x8048388 <main>:   "\215L$\004\..."
...
-0x80483d0 <__libc_csu_init>:  "..."
```

gdb 指令：查閱型態

- (gdb) **ptype** 變數

```
struct Rect {
        int x;
        int y;
};

int main()
{
        struct Rect rect = { -10, 20 };
        return 0;
}
```

ptype.c

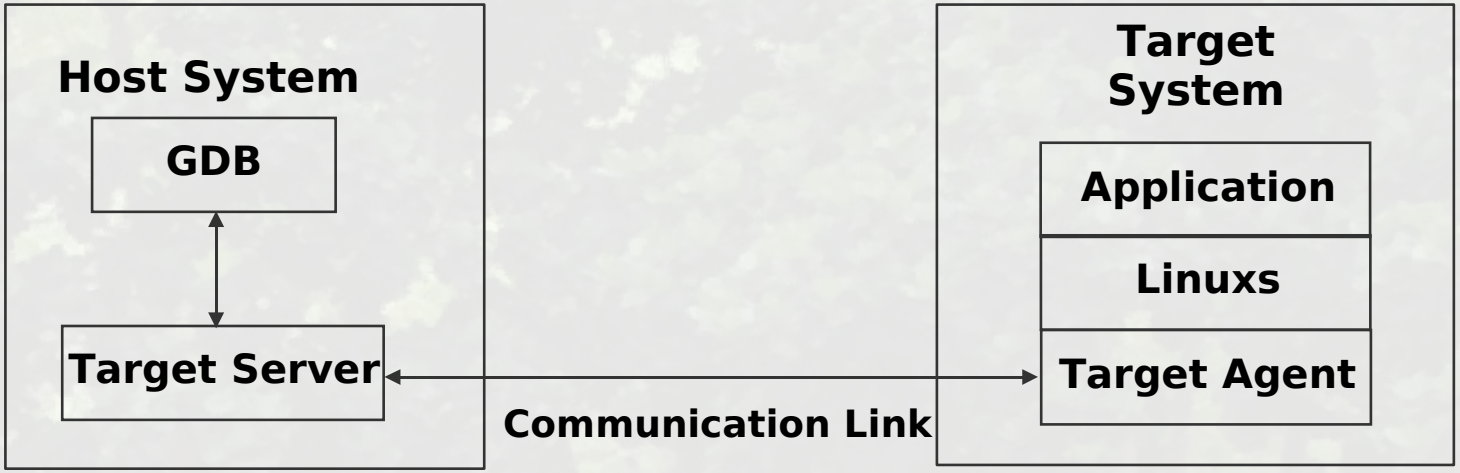
```
(gdb) ptype main
type = int ()
(gdb) ptype rect
type = struct Rect {
    int x;
    int y;
}
(gdb) p rect
$1 = {
  x = -10,
  y = 20
}
```

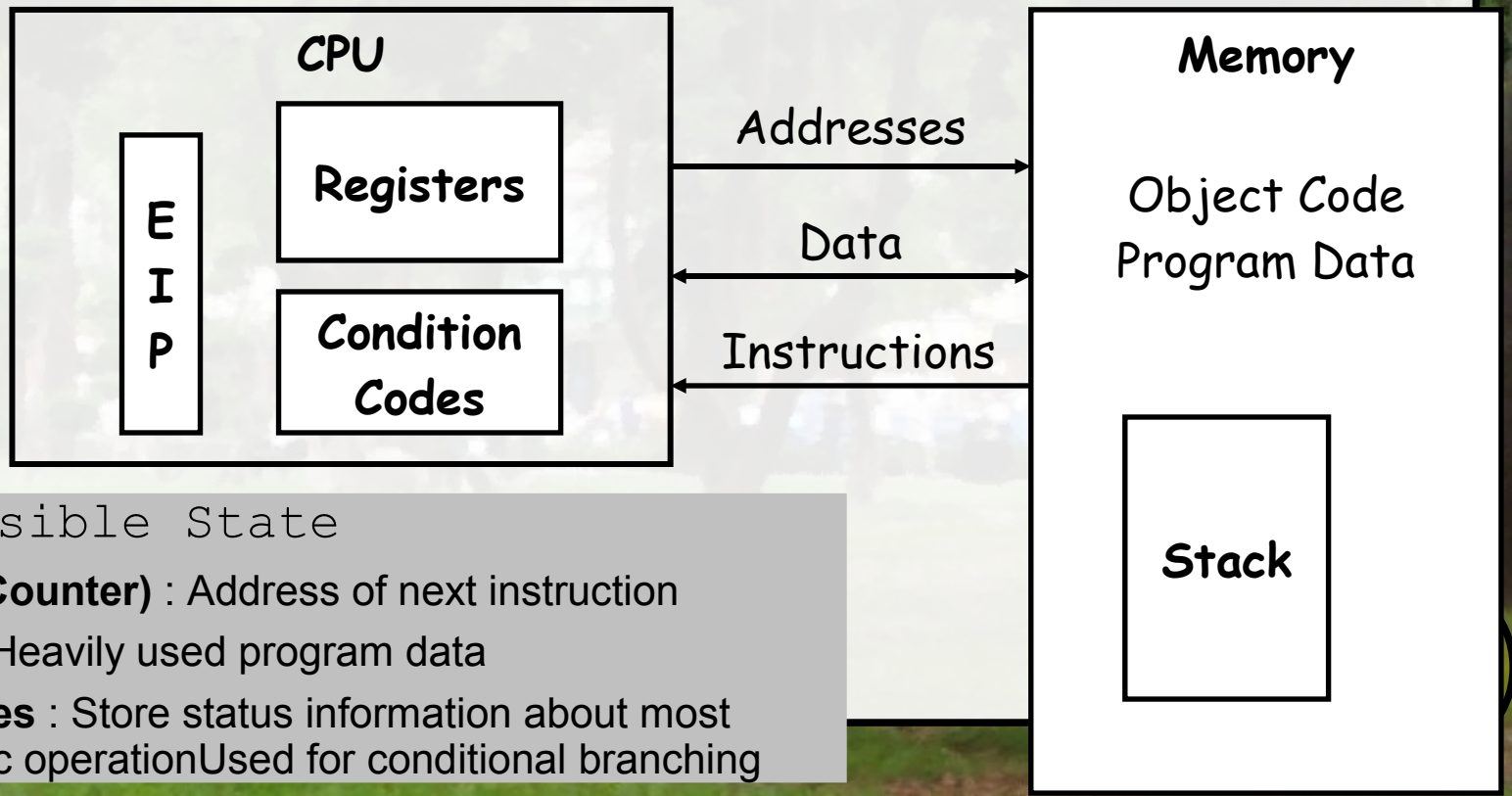
在 set print pretty on 時，
p 與 ptype 有匹配的輸出

gdb 指令：立即變數

- 以 '$' 開頭的立即變數
- 系統內建
  - **$pc** — program counter
  - **$sp** – stack pointer
  - **$fp** – frame pointer
  - **$ps** – processor status
  - **$_** – contains the last examined address
  - **$__** – the value in the last examined address
  - **$_exitcode** – the exit code of the debugged program

機械觀點





Programmer-Visible State

- **EIP (Program Counter)** : Address of next instruction
- **Register File** : Heavily used program data
- **Condition Codes** : Store status information about most recent arithmetic operationUsed for conditional branching

gdb 指令：改變記憶體狀態

- (gdb) **set variable** 變數 = 值
- (gdb) **set** 位址 值

set.c

```
int main()
{
    int a, b, c = 0;
    c = a + b;
    return 0;
}
```

```
(gdb) p c
$1 = -1075837384
(gdb) n
4       c = a + b;
(gdb) set variable a=1
(gdb) set variable b=1
(gdb) n
5       return 0;
(gdb) p c
$2 = 2
(gdb)
```

提示：未初始化的記憶體內容

gdb 指令：儲存 / 回復記憶內容

- (gdb) **dump memory** 檔名 起始位置 終止位置
- (gdb) **restore** 檔名 **binary** 起始位置
- 可搭配外部工具使用

```
(gdb) p str
$1 = 0x8048460 "1234567890"
(gdb) dump memory clone-mem-file 0x8048460 0x804846a
```

```
(gdb) p str
$1 = 0x8048460 "0987654321"
(gdb) restore clone-mem-file binary
0x8048460
Restoring binary file clone-mem-file into
memory (0x8048460 to 0x804846a)
(gdb) p str
$2 = 0x8048460 "1234567890"
```

save-restore.c

```
#include <stdio.h>

char * str =
#if 1
    "1234567890";
#else
    "0987654321";
#endif

int main()
{
    puts(str);
    return 0;
}
```

gdb 指令：對已執行的行程偵錯

- (gdb) **attach** PID
- (gdb) **dettach**
- $ **gdb -pid=**PID
- 若已執行的行程無 Debug info，需要額外指定 image file

gdb 指令：中斷點

- (gdb) break *lines-number*
- (gdb) break *function-name*
- (gdb) break *line-or-function if condition*
- 設定發生中斷點的條件

prefix.c

```
(gdb) list priority
1       #include <stdio.h>
2       #include <string.h>
3
4       int priority(char x)
5       {
6           switch (x) {
7               case '+':
8               case '-': return 0;
9               case '*':
10              case '/': return 1;
(gdb) break 4
Breakpoint 1 at 0x80484d0: file prefix.c, line 4.
```

```
(gdb) break priority
Breakpoint 1 at 0x80484dc: file prefix.c, line
    6.
(gdb) run
Starting program: /tmp/src/prefix
Input expression: 3+5

Breakpoint 1, priority (x=35 '#') at prefix.c:6
6           switch (x) {
```

gdb 指令：中斷點

- (gdb) break *lines-number*
- (gdb) break *function-name*
- (gdb) break *line-or-function if condition*
- 設定發生中斷點的條件

```
(gdb) break 4 if x == '+'
Breakpoint 1 at 0x80484dc: file prefix.c, line 6.
(gdb) run
Starting program: /tmp/src/prefix
Input expression: 3-2
Prefix: -32

Program exited with code 014.
(gdb) run
Starting program: /tmp/src/prefix
Input expression: 3+2

Breakpoint 1, priority (x=43 '+') at prefix.c:6
6           switch (x) {
```

gdb 指令： stack backtrace

- (gdb) **bt**
- (gdb) **up**
- (gdb) **down**
- (gdb) **fx**

```
#include <stdio.h>
void func4() { puts("Output"); }
void func3() { func4(); }
void func2() { func3(); }
void func1() { func2(); }

int main()
{
    int ret = 0;
    func1();
    return ret;
}
```

stack-backtrace.c

```
(gdb) until 11
Output
main () at stack-backtrace.c:11
11      return ret;
(gdb) info locals
ret = 0
(gdb) info f
Stack level 0, frame at 0xbf812830:
 eip = 0x80483cc in main (stack-backtrace.c:11); saved eip 0xb7e1c450
 source language c.
 Arglist at 0xbf812828, args:
 Locals at 0xbf812828, Previous frame's sp at 0xbf812824
 Saved registers:
  ebp at 0xbf812828, eip at 0xbf81282c
```

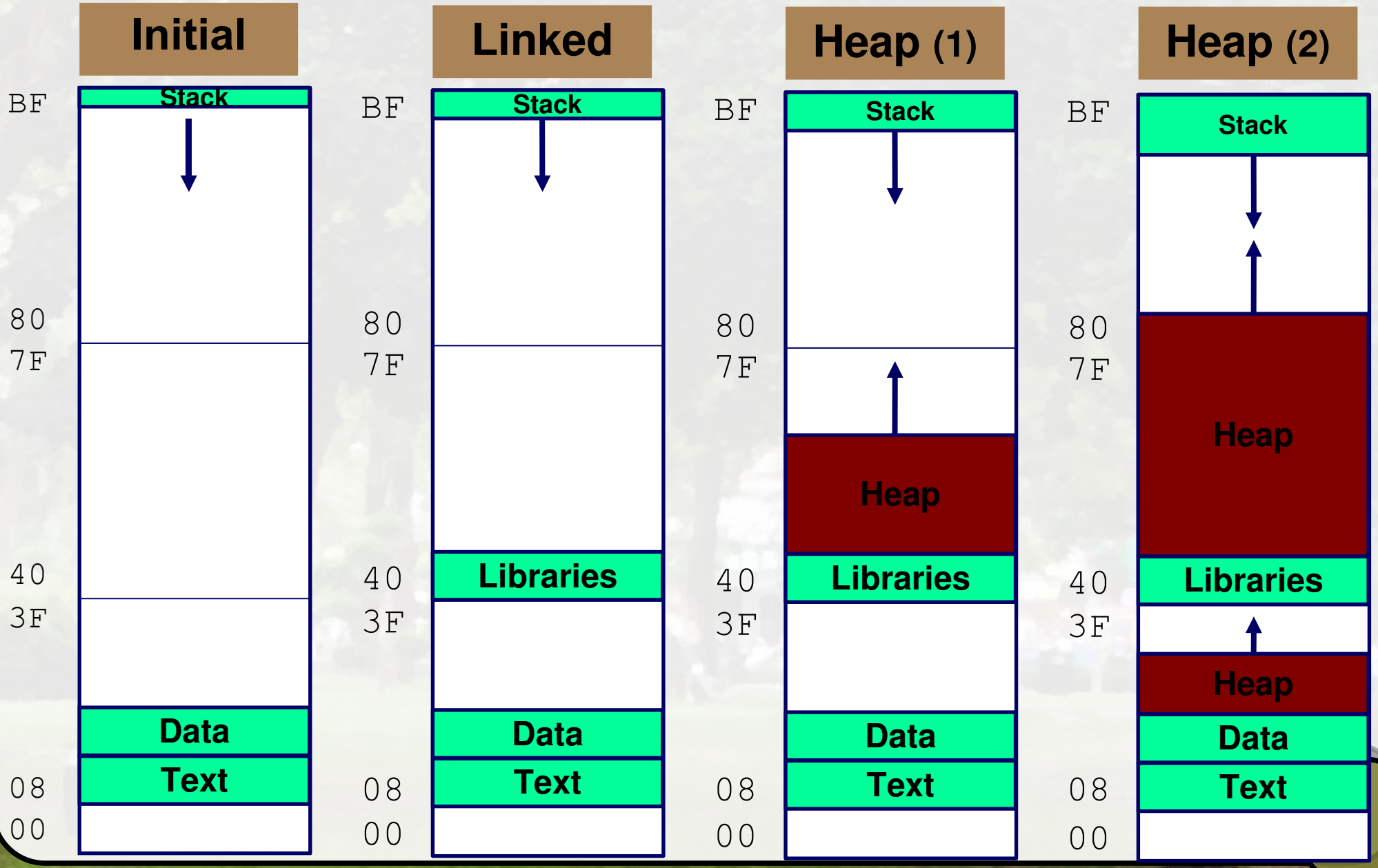
Stack Frame

Stack Frame 概念

Stack Frame 是深入理解系統程式的重要概念

- 記憶體操作模式
- IA32 的 stack 處理
- 知悉 IA32 的 stack frame 與 GDB 的對應

Linux Memory Allocation



觀察 Text 與 Stack

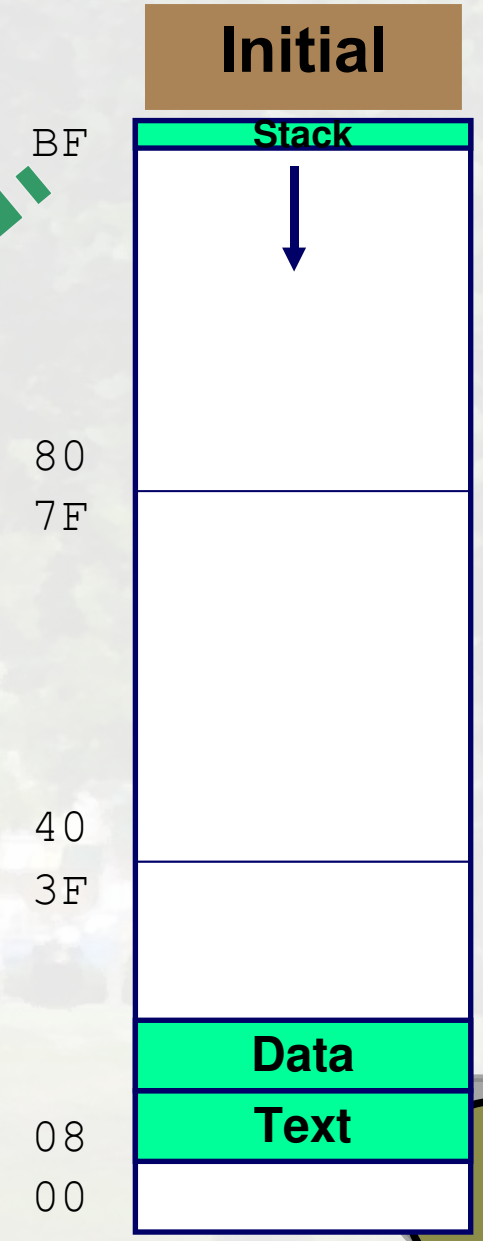
```
(gdb) break main
(gdb) run
  Breakpoint 1, 0x804856f in main ()
(gdb) print $esp
  $3 = (void *) 0xbffffc78
```

main (位於 text)

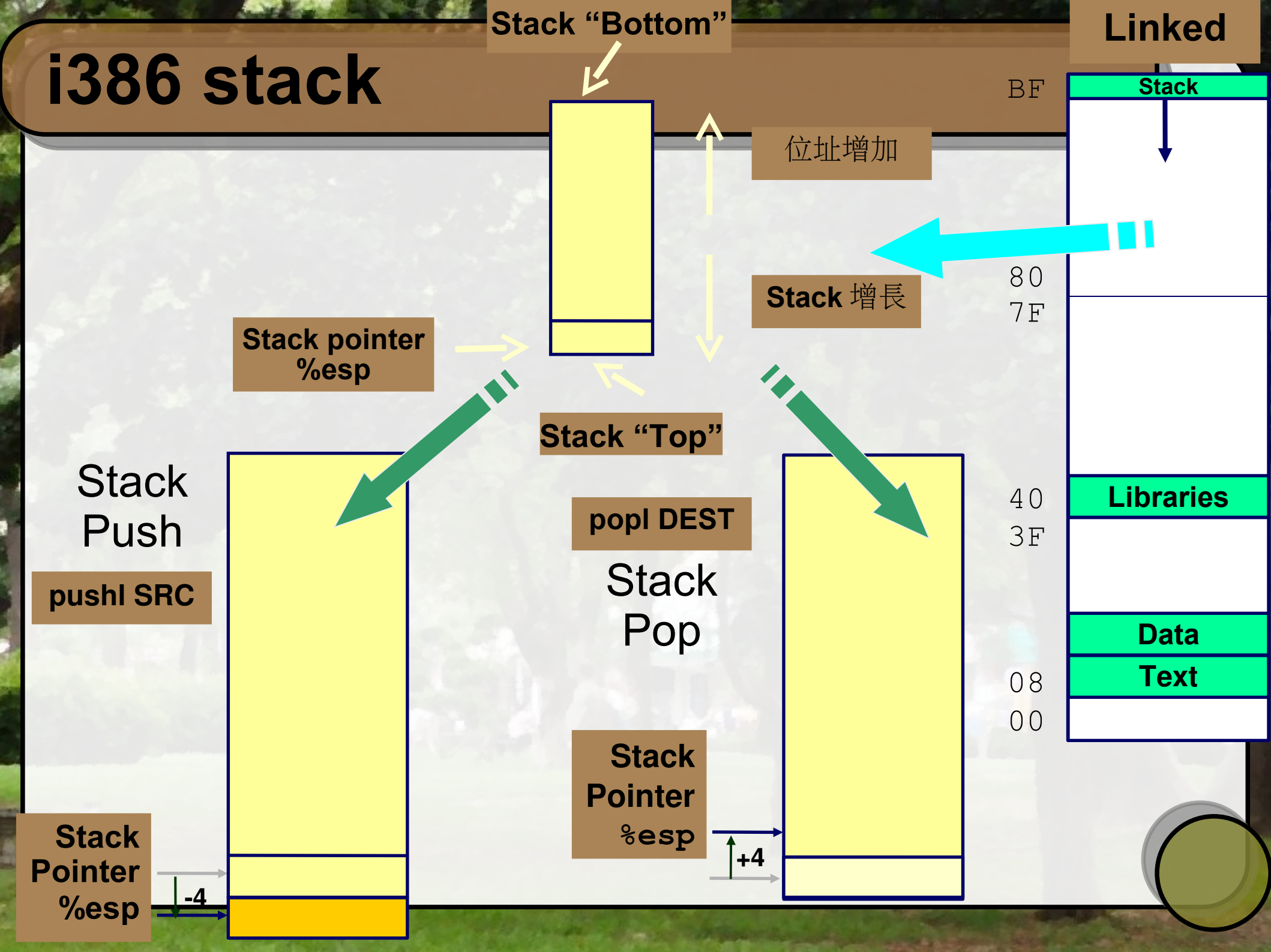
- 位址 0x804856f 即 0x08 04856f

Stack

- 位址 0xbfffc78



i386 stack


Stack "Bottom"
Linked
BF
Stack
位址增加
80
7F
Stack 增長
Stack pointer
%esp
Stack "Top"
Stack
Push
pushl SRC
popl DEST
Stack
Pop
40
3F
Libraries
Data
Text
08
00
Stack
Pointer
%esp
-4
Stack
Pointer
%esp
+4

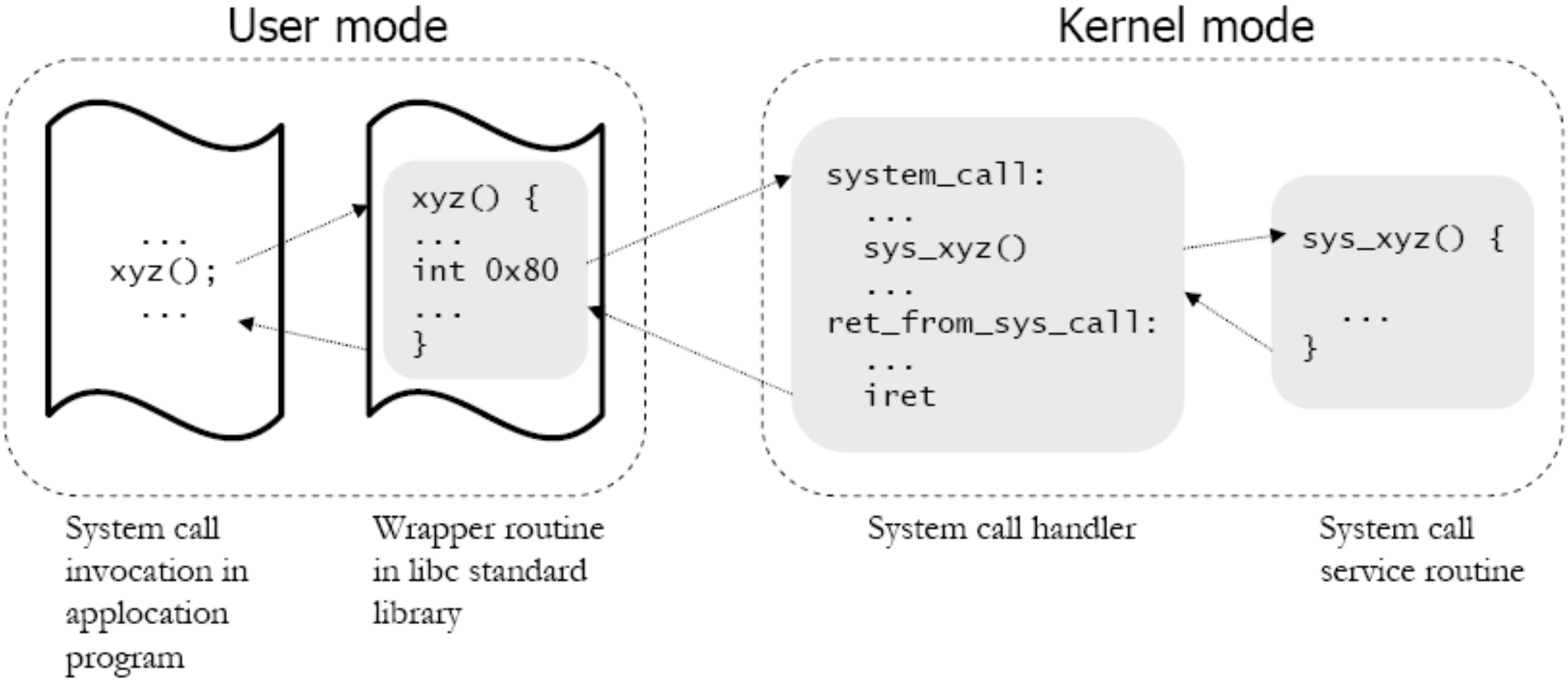
i386/Linux register

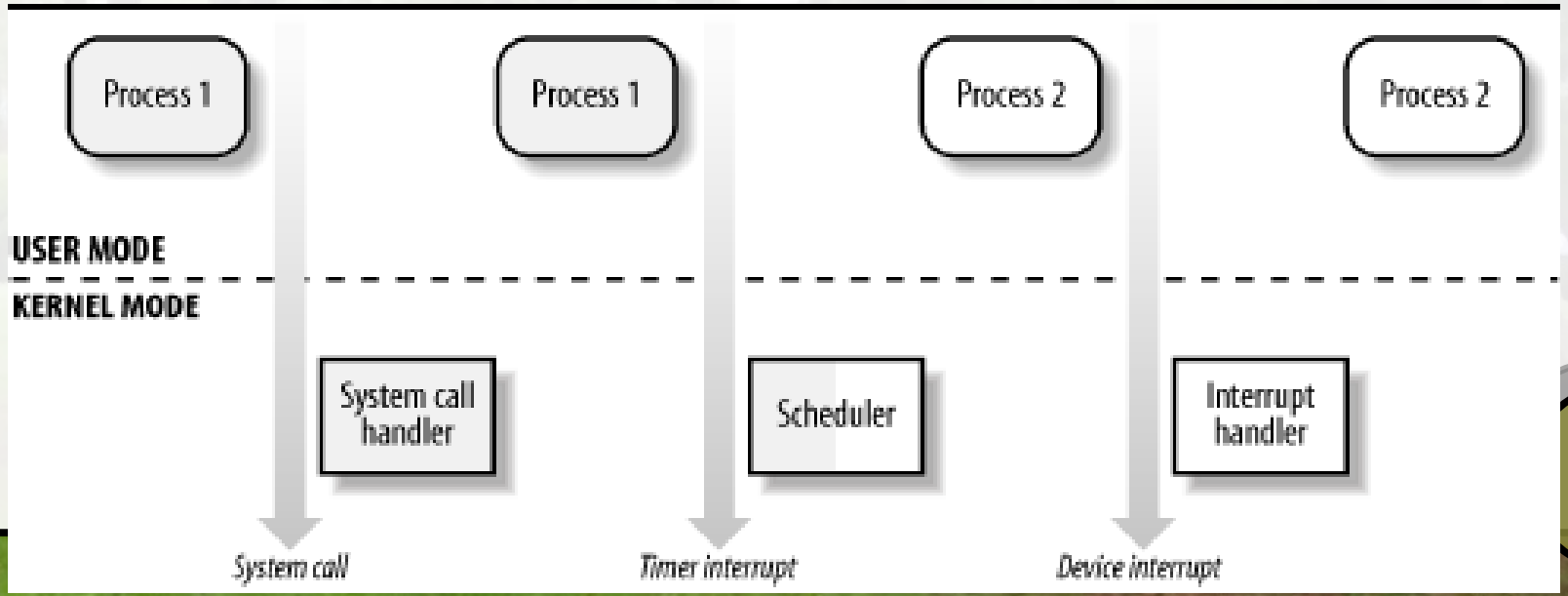
也作為保存 return address 使用

```
#include <stdio.h>
char message[] = "Hello, world!\n";
int main(void)
{
  long _res;
  __asm__ volatile (
    "int $0x80"
    : "=a" (_res)
    : "a" ((long) 4),
      "b" ((long) 1),
      "c" ((long) message),
      "d" ((long) sizeof(message)));
    return 0;
}
```

```
.text
message:
.ascii "Hello World!\0"
.align 4
.globl main
main:
        pushl %ebp
        movl %esp,%ebp
        pushl $message
        call puts
        addl $4,%esp
        xorl %eax,%eax
        movl %ebp,%esp
        popl %ebp
        ret
```

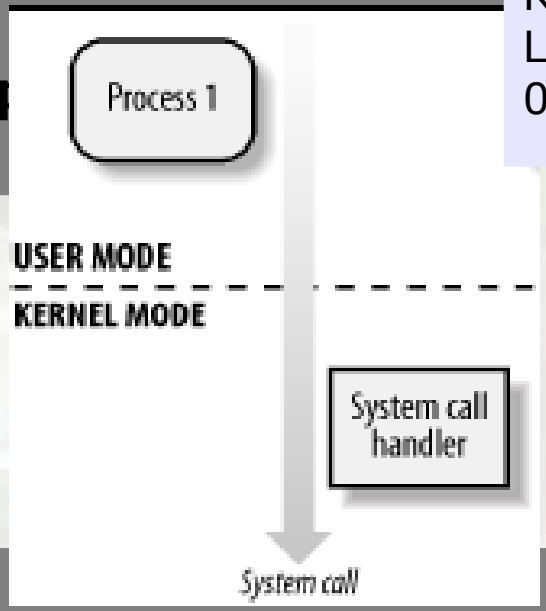
Caller-Save Temporaries	%eax, %edx, %ecx
Callee-Save Temporaries	%ebx, %esi, %edi
Special	%esp, %ebp


User mode
Kernel mode
...
xyz();
...
xyz() {
...
int 0x80
...
}
system_call:
...
sys_xyz()
...
ret_from_sys_call:
...
iret
sys_xyz() {
...
}
System call invocation in applocation program
Wrapper routine in libc standard library
System call handler
System call service routine


Process 1
Process 1
Process 2
Process 2
USER MODE
KERNEL MODE
System call handler
Scheduler
Interrupt handler
System call
Timer interrupt
Device interrupt

```
$ cat hello-loop.c
#include <stdio.h>
#include <unistd.h>
int main(int argc, char **argv)
{
        printf("Hello World!\n");
        while (1) {
                usleep(10000);
        }
        return 0;
}
$ ./hello-loop
Hello World!
```

```
$ pidof hello-loop
6987
$ gdb
(gdb) attach 6987
Attaching to process 6987
Reading symbols from
/home/jserv/HelloWorld/samples/hello-loop...done.
Using host libthread_db library
"/lib/tls/i686/cmov/libthread_db.so.1".
Reading symbols from
/lib/tls/i686/cmov/libc.so.6...done.
Loaded symbols for /lib/tls/i686/cmov/libc.so.6
Reading symbols from /lib/ld-linux.so.2...done.
Loaded symbols for /lib/ld-linux.so.2
0xffffe410 in __kernel_vsyscall ()
```



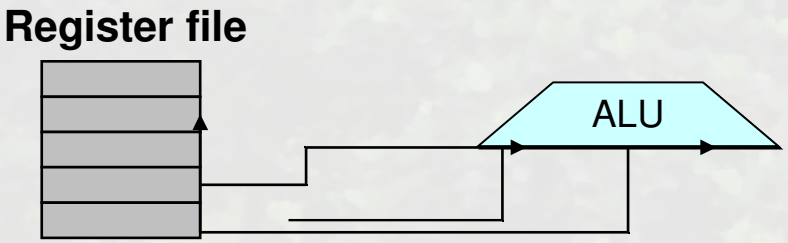

```
(gdb) bt
#0  0xffffe410 in __kernel_vsyscall ()
#1  0xb7e37ef0 in nanosleep () from /lib/tls/i686/cmov/libc.so.6
#2  0xb7e6f93a in usleep () from /lib/tls/i686/cmov/libc.so.6
#3  0x080483ad in main () at hello-loop.c:7
```

C 函數：機械觀點

```
int subtract(int a, int b)
{
   return (a - b);
}
```

Gcc -S

```
_subtract:
        pushl   %ebp
        movl    %esp, %ebp
        movl    12(%ebp), %edx
        movl    8(%ebp), %eax
        subl    %edx, %eax
        popl    %ebp
        ret
```

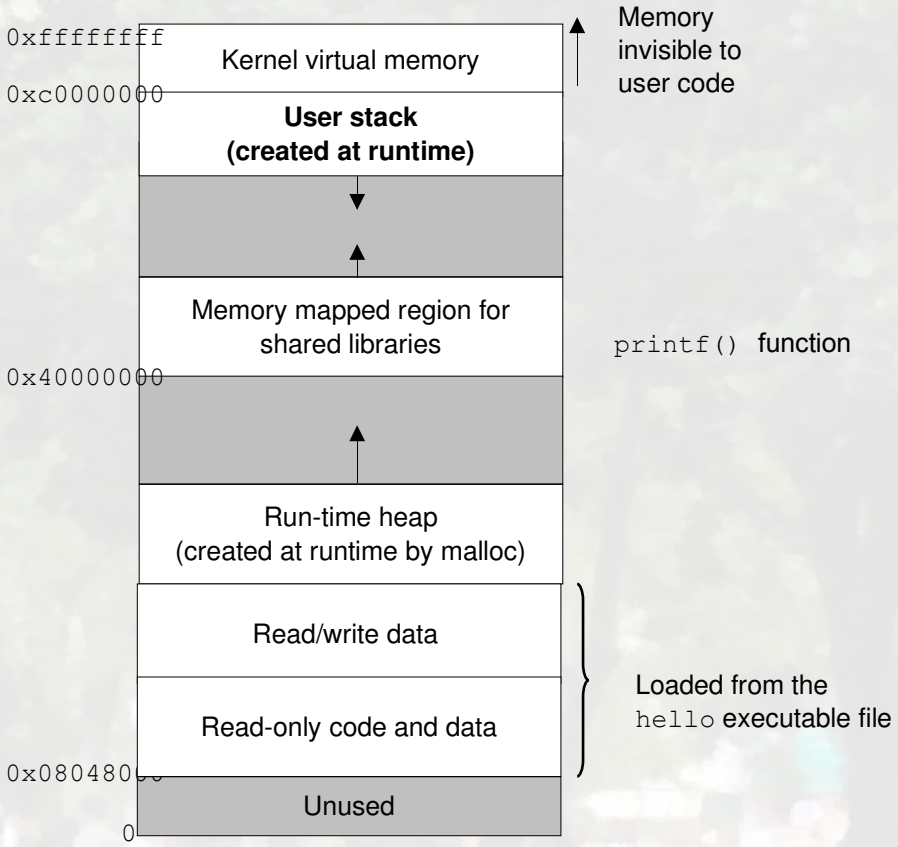


C 函數的引數 **(a, b)** 自記憶體
被搬移到兩個暫存器 **(%ax,%edx)** 中，
並透過 **subl** 指令作運算

C 函式呼叫：機械觀點

Calling procedure

push parameter2 on stack
Push parameter1 on stack
Call subroutine
Clean parameters off stack



- **stack** 是虛擬記憶體的一部分
- 每個被呼叫的函式在 stack 擁有 **frame**

i386 call(1)

_start 為該 Image 的 **entry point**

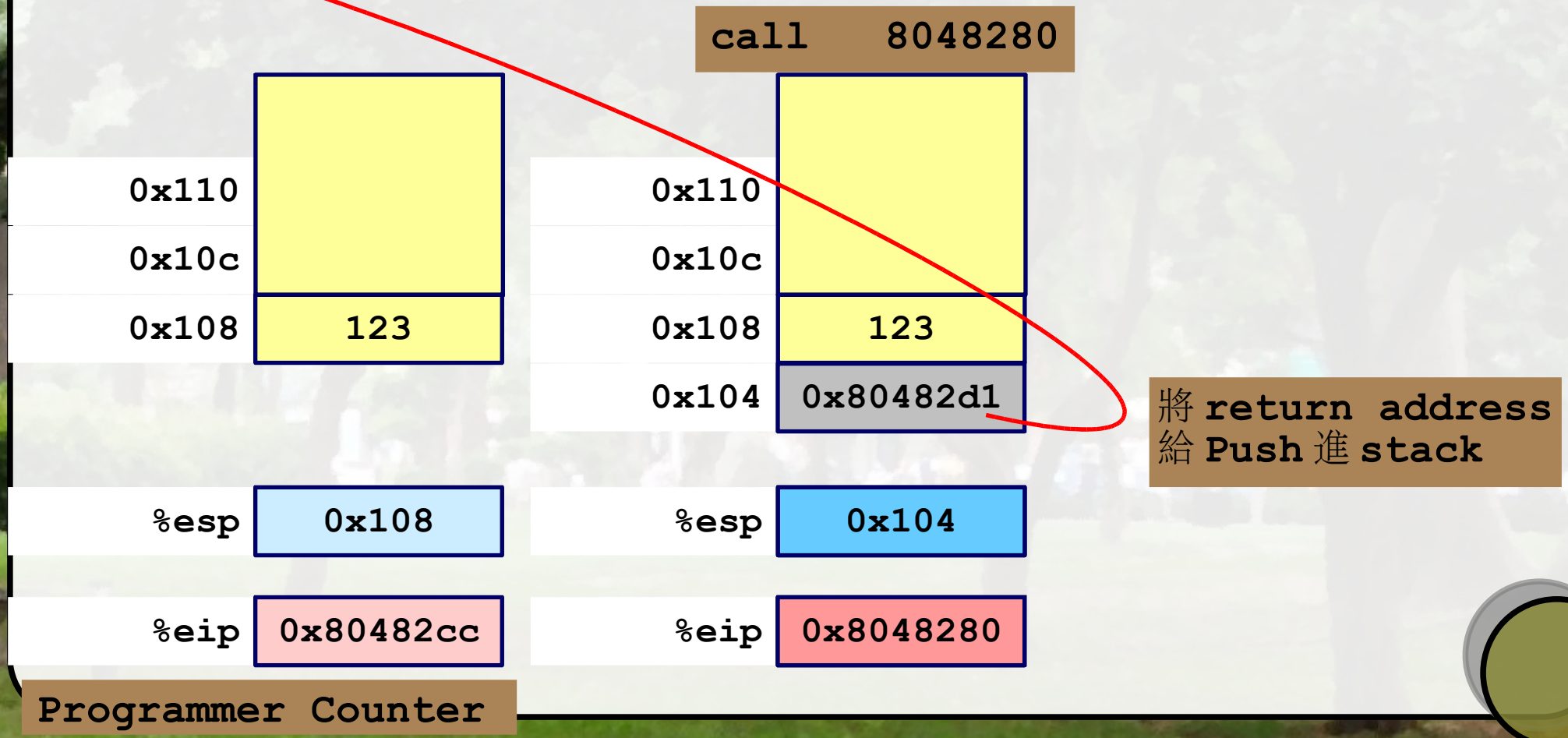
```
Disassembly of section .text:

080482b0 <_start>:
 80482b0:       31 ed           xor    %ebp,%ebp
 80482b2:       5e              pop    %esi
 80482b3:       89 e1           mov    %esp,%ecx
 80482b5:       83 e4 f0        and    $0xfffffff0,%esp
 80482b8:       50              push   %eax
 80482b9:       54              push   %esp
 80482ba:       52              push   %edx
 80482bb:       68 84 83 04 08  push   $0x8048384
 80482c0:       68 89 83 04 08  push   $0x8048389
 80482c5:       51              push   %ecx
 80482c6:       56              push   %esi
 80482c7:       68 50 83 04 08  push   $0x8048350
 80482cc:       e8 af ff ff ff  call   8048280 <__libc_start_main@plt>
 80482d1:       f4              hlt
 80482d2:       90              nop
 80482d3:       90
```

```
08048280 <__libc_start_main@plt>:
 8048280:       ff 25 44 95 04 08 jmp    *0x8049544
 8048286:       68 00 00 00 00    push   $0x0
 804828b:       e9 e0 ff ff ff    jmp    8048270 <_init+0x18>
```

- call LABLE
  - 使用 stack 來實現 procedure call
  - 先 PUSH 返回位址，然後 JUMP 到 LABLE

i386 call(2)

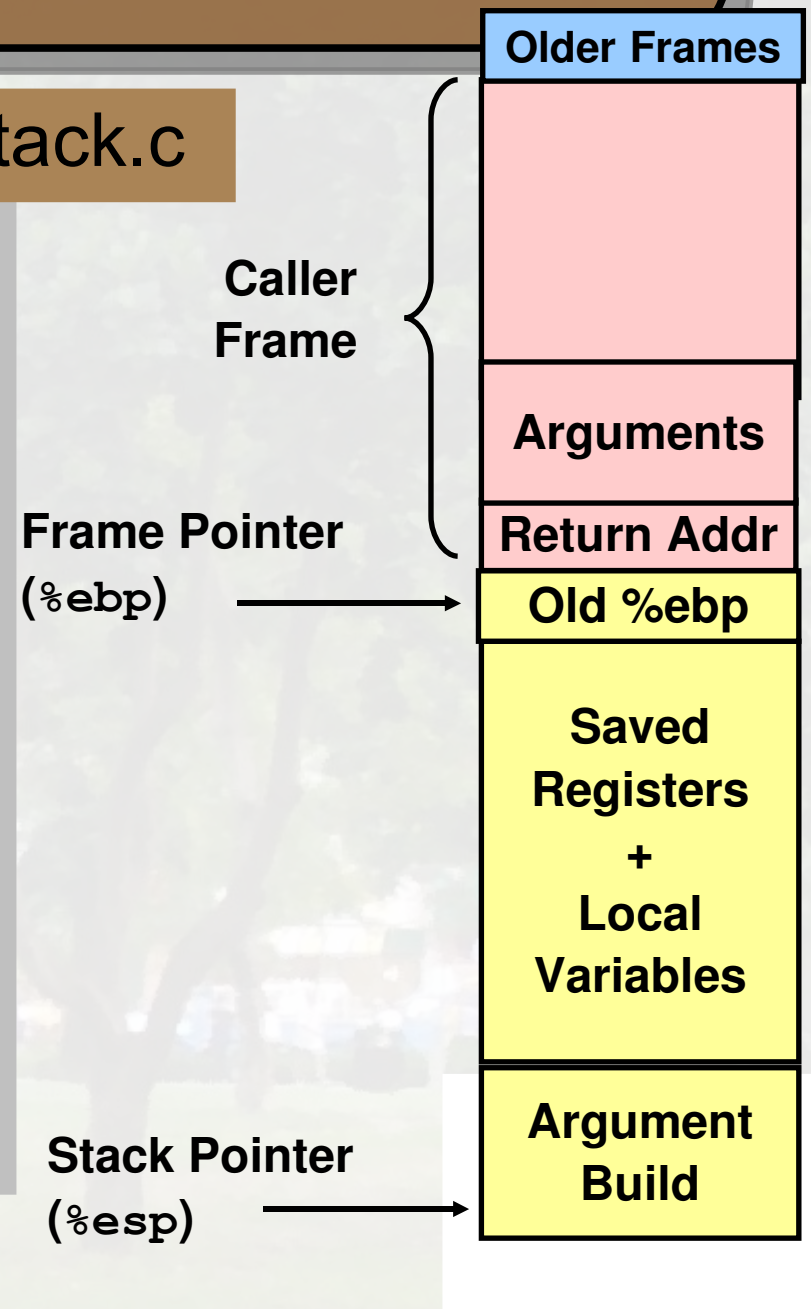
```
Disassembly of section .text:

080482b0 <_start>:
...
80482cc:        e8 af ff ff ff        call   8048280 <__libc_start_main@plt>
80482d1:        f4                    hlt
```



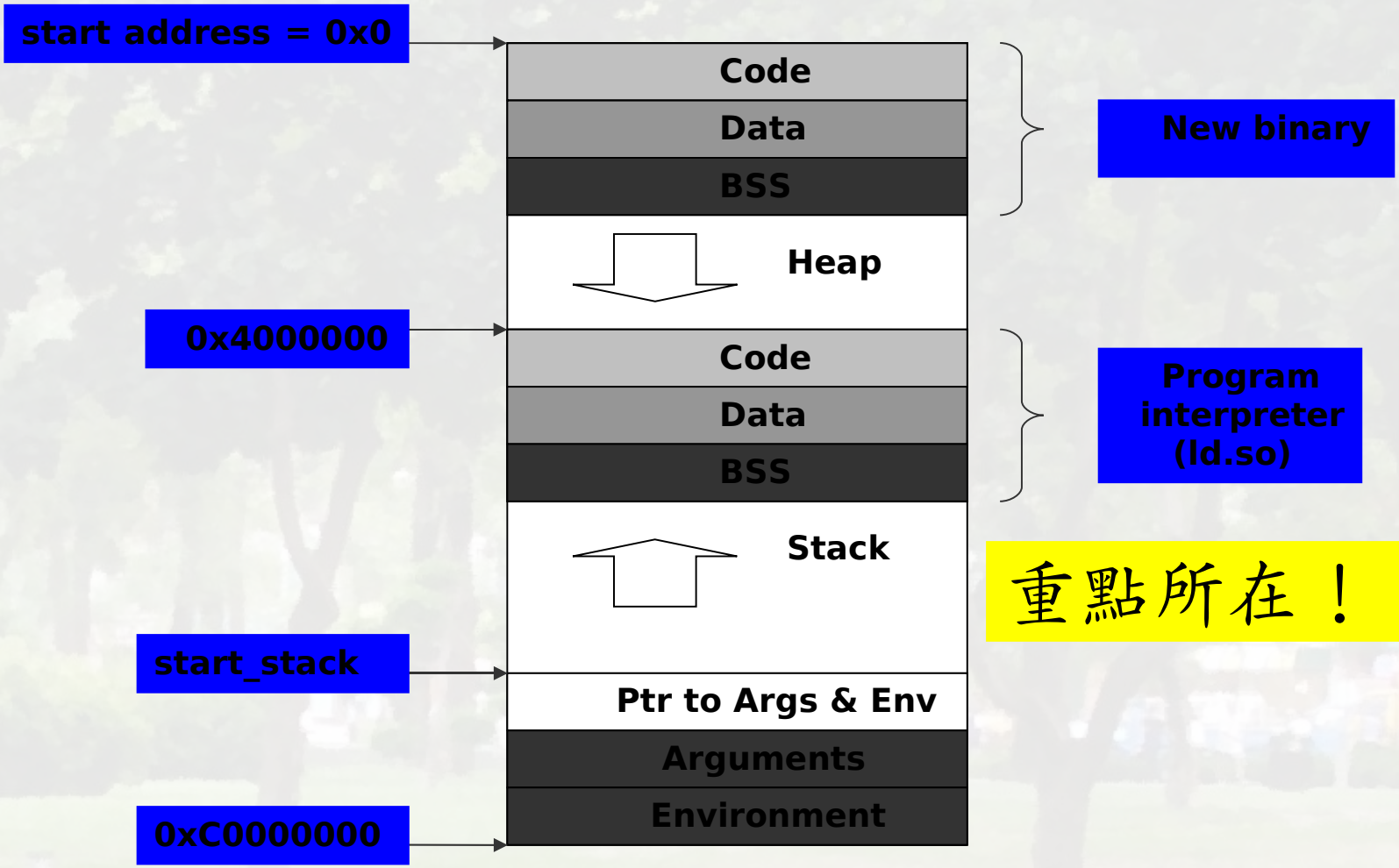
初步觀察

```
(gdb) b subtract
Breakpoint 1 at 0x8048377: file stack.c,
line 5.
(gdb) r
Starting program: stack
Breakpoint 1, subtract (a=3, b=2) at
stack.c:5
5           return (a - b);
(gdb) info stack
#0  subtract (a=3, b=2) at stack.c:5
#1  0x080483a6 in main () at stack.c:10
(gdb) up
#1  0x080483a6 in main () at stack.c:10
10      printf("out = %d\n", subtract(3, 2));
(gdb) down
#0  subtract (a=3, b=2) at stack.c:5
5           return (a - b);
```

stack.c



Process Address Space



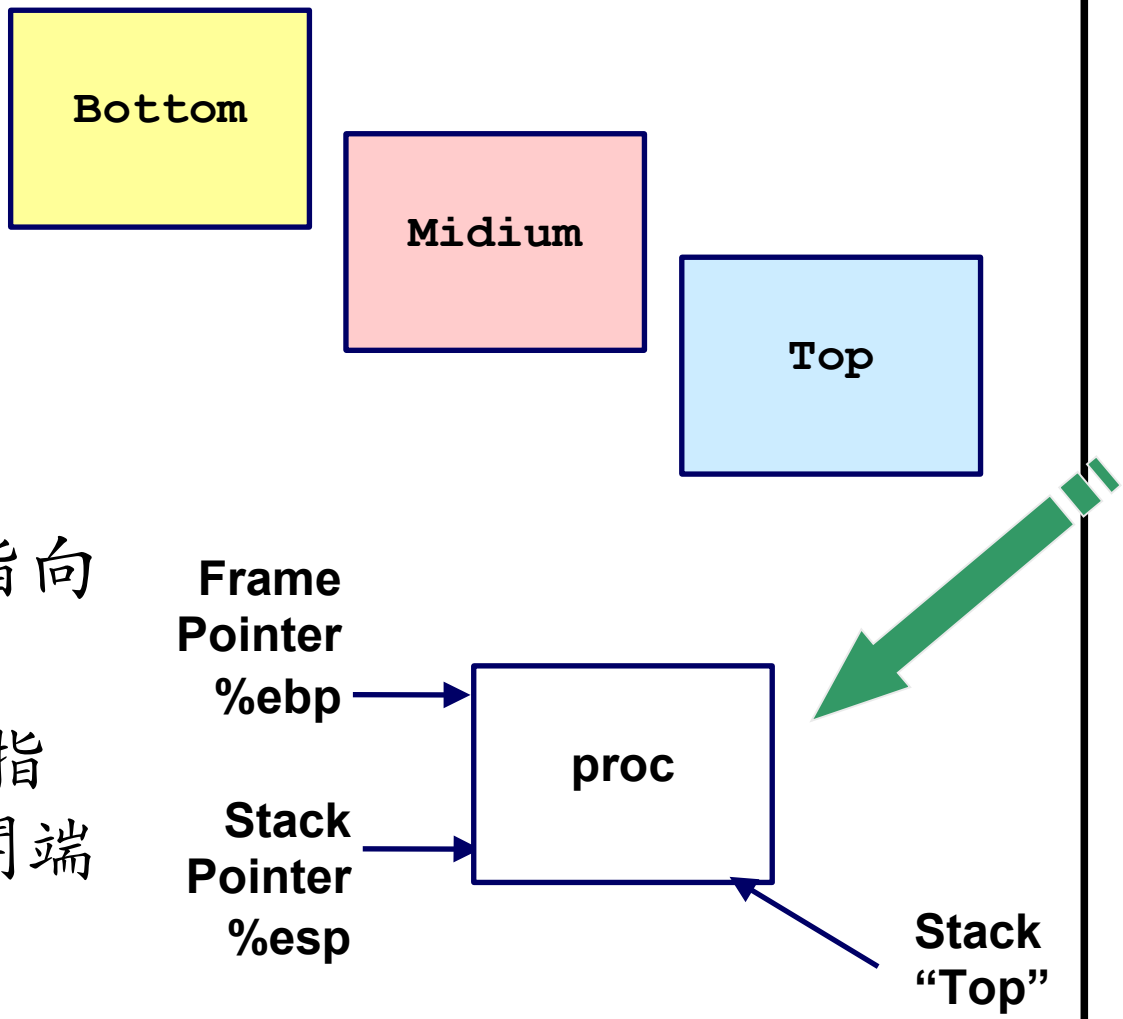
Stack Frame 與 C 程式

```
void function(int a, int b){
  printf("hello");
  return;
}

void main() {
  function(1,2);
}
```

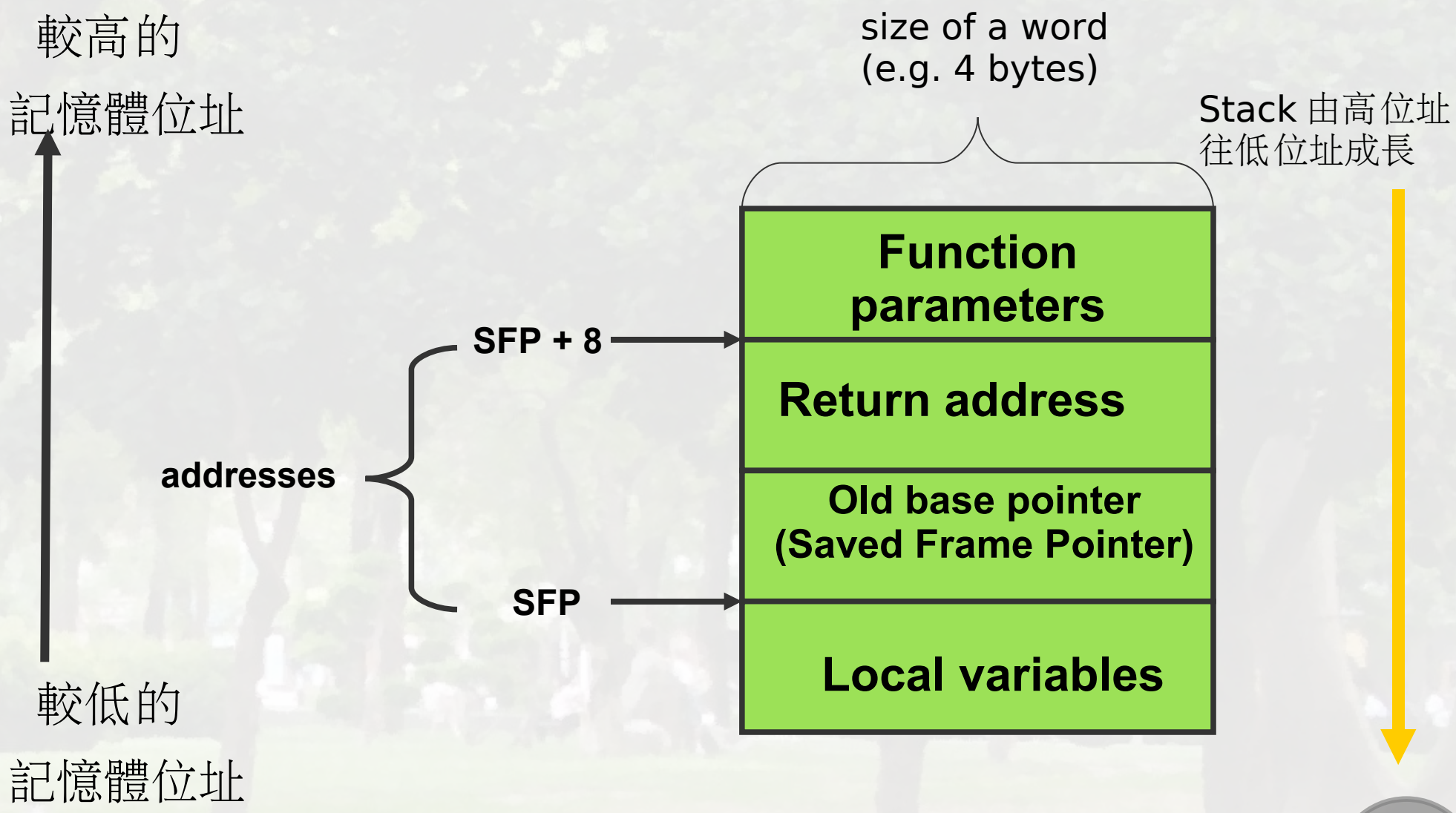
細節！

Stack Frame

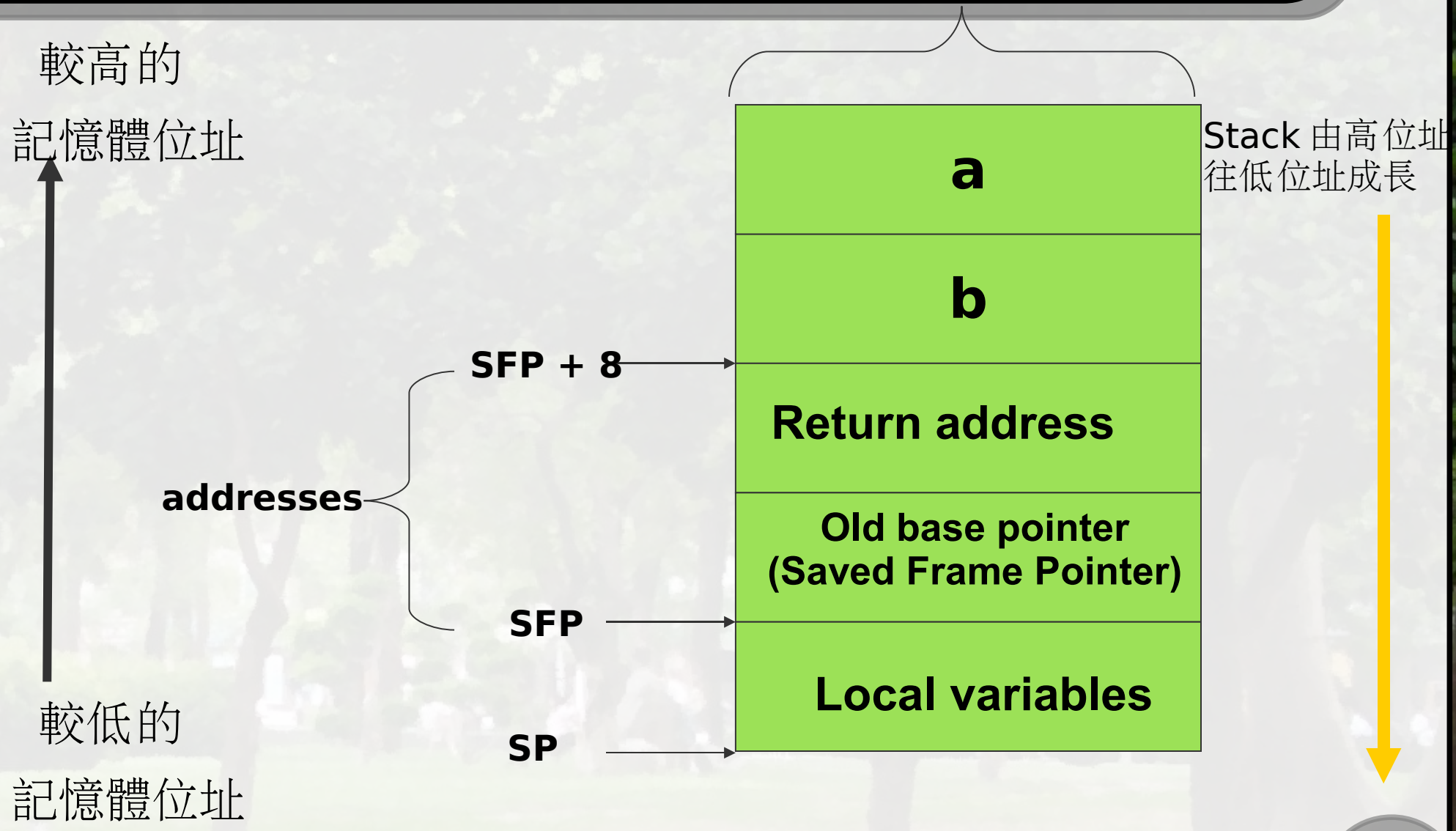
- 要素
  - Local variables
  - Return information
  - Temporary space
- Pointers
  - Stack Pointer %esp 指向 Stack 的頂端
  - Frame pointer %ebp 指向 Current Frame 的開端



Stack Frame


較高的
記憶體位址
size of a word
(e.g. 4 bytes)
Stack 由高位址
往低位址成長
Function
parameters
SFP + 8
Return address
addresses
Old base pointer
(Saved Frame Pointer)
SFP
Local variables
較低的
記憶體位址

Stack Frame



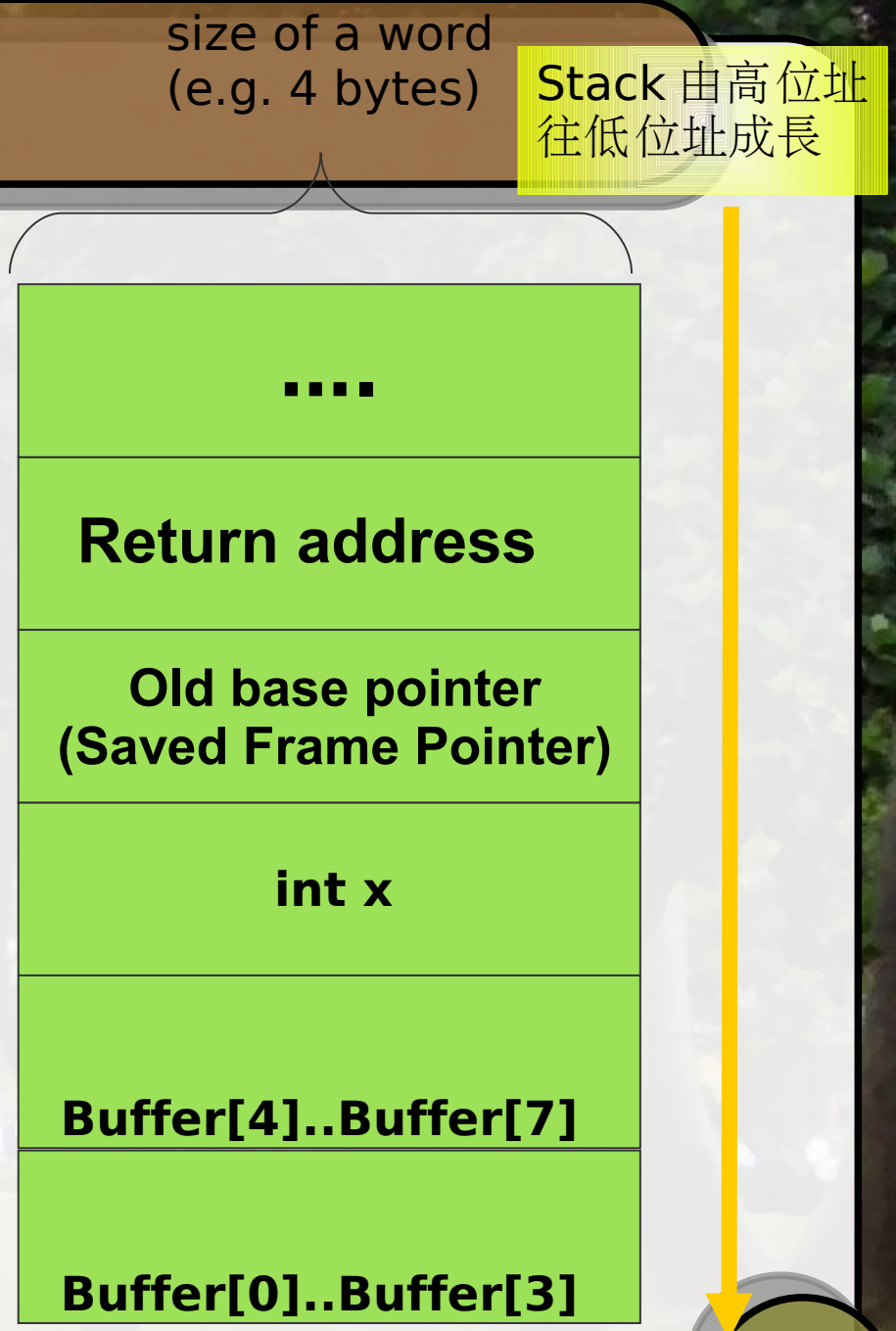
當呼叫 `void function(int a, int b)`

觀察 (1)

```
void function(){
  int x = 0;
  char buffer[8];

  memcpy(buffer,"abcdefg",8);

  printf( "%s %d", buffer, x );
}
```

Output:

```
...
```

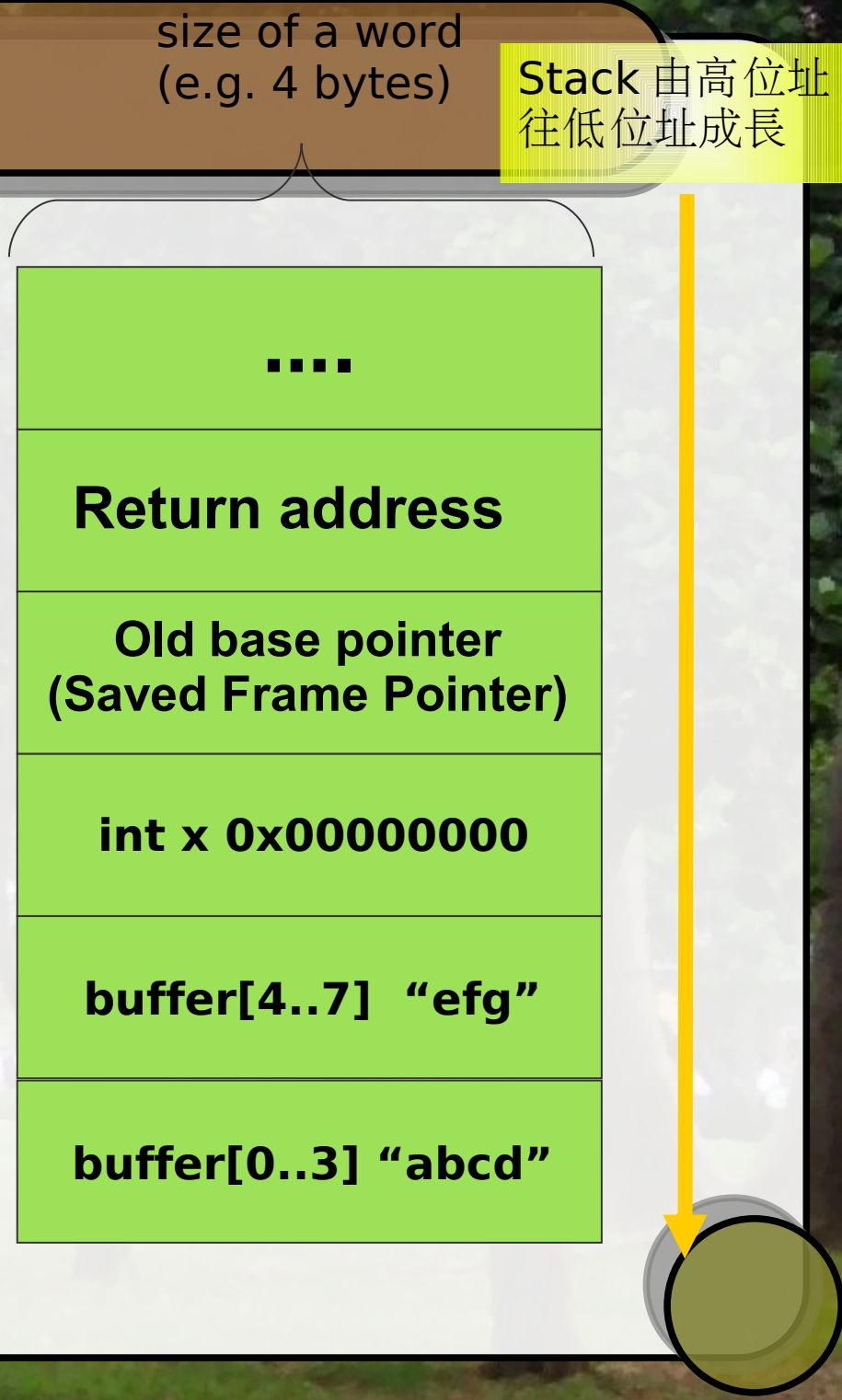


觀察 (1)

```
void function(){
  int x = 0;
  char buffer[8];

  memcpy(buffer,"abcdefg",8);

  printf( "%s %d", buffer, x );
}
```

Output:

```
abcdefg 0
```

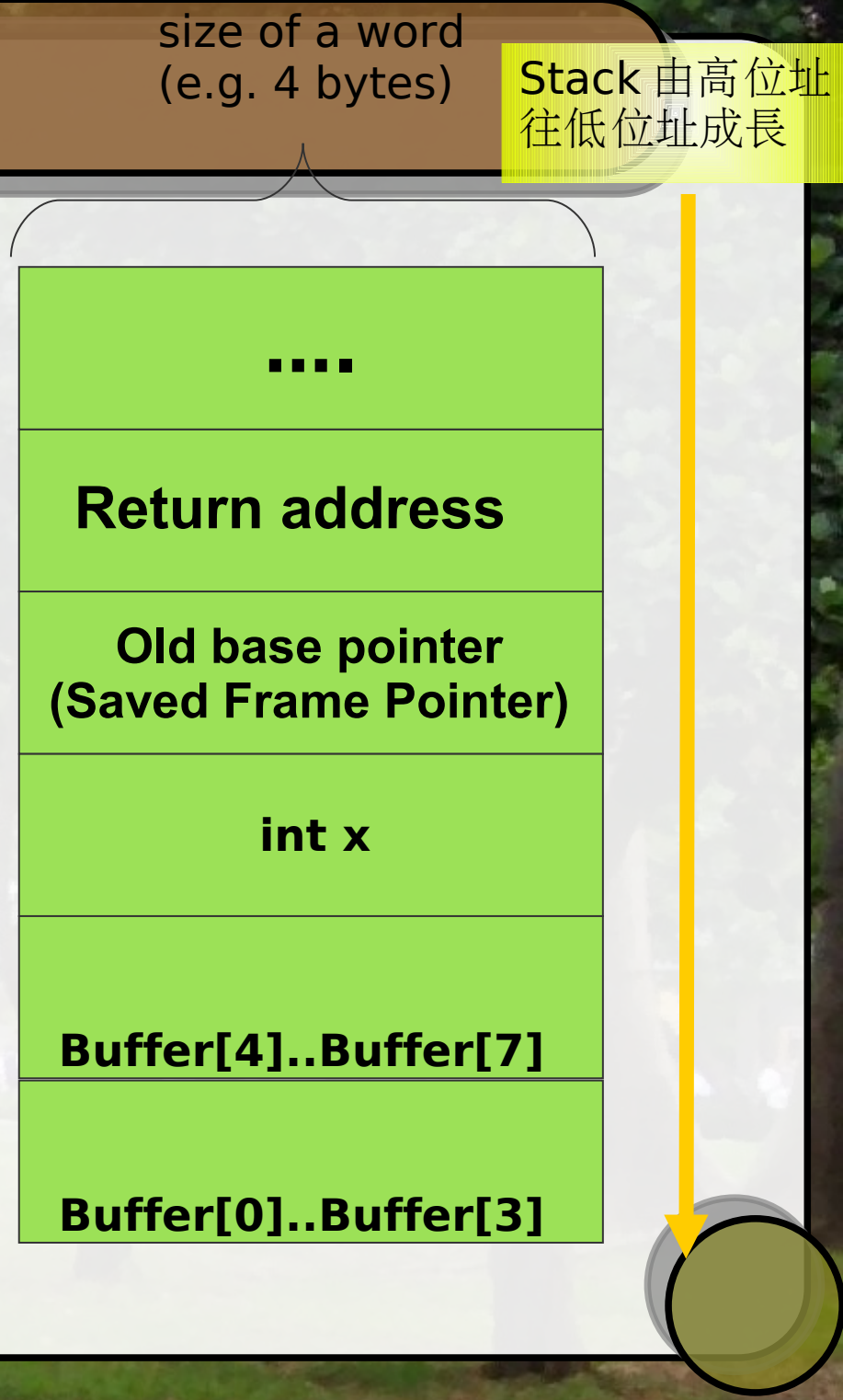


觀察 (2)

```
void function(){
  int x = 0;
  char buffer[8];

  memcopy(buffer,
          "abcdefghijk",12);

  printf( "%s %d", buffer, x );
}
```

Output:

```
...
```

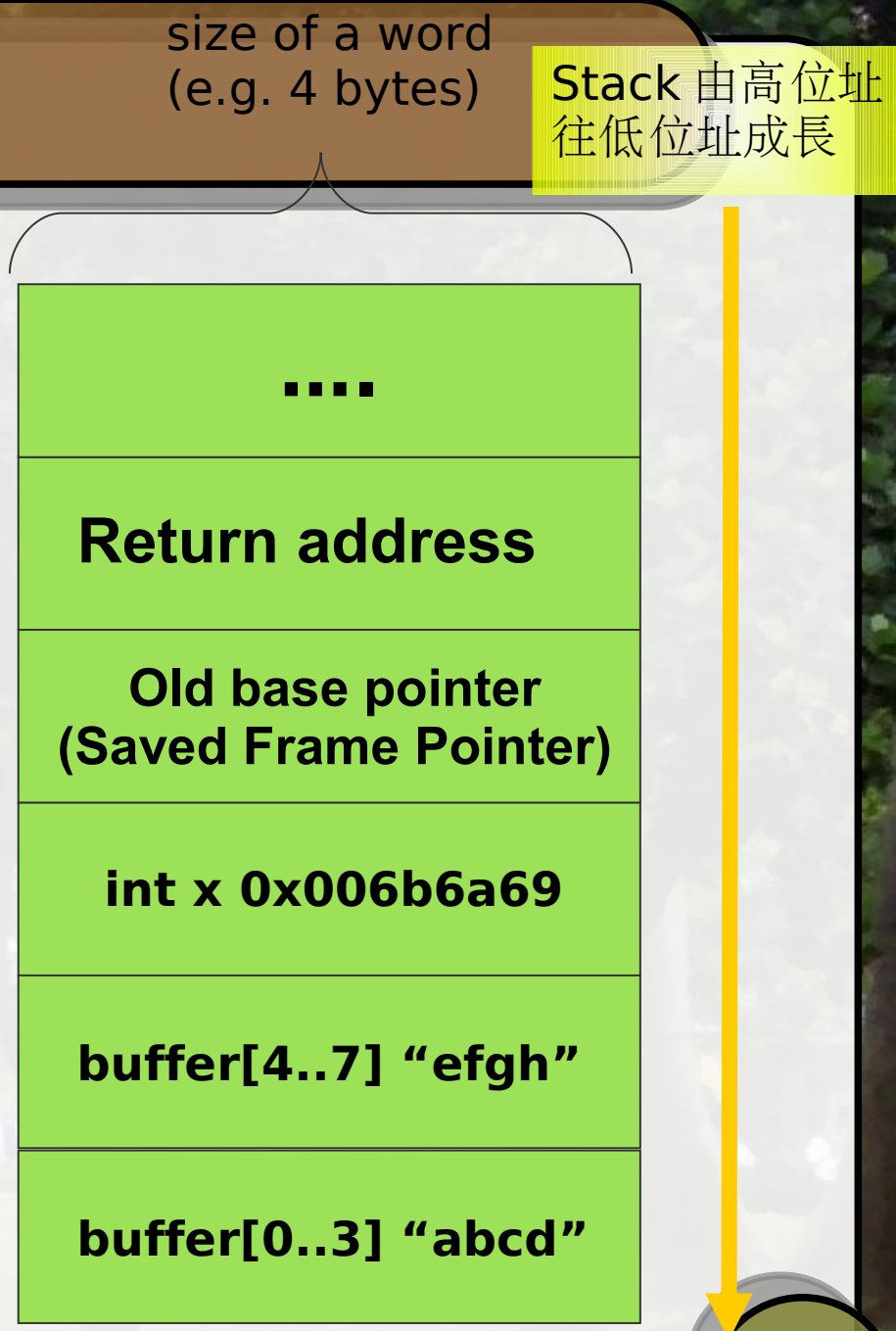


觀察 (2)

```
void function(){
  int x = 0;
  char buffer[8];

  memcopy(buffer,
          "abcdefghijk",12);

  printf( "%s %d", buffer, x );
}
```

Output:

```
abcdefghijkl 7039593
```



範例：遞迴 *Fibonacci Numbers*

```
int fib (int n)
{
    int result;

    if (n <= 2)
        result = 1;
    else
        result = fib(n-2) +
             fib(n-1);

    return result;
}

int main()
{
    int ret = fib(5);
    return ret;
}
```

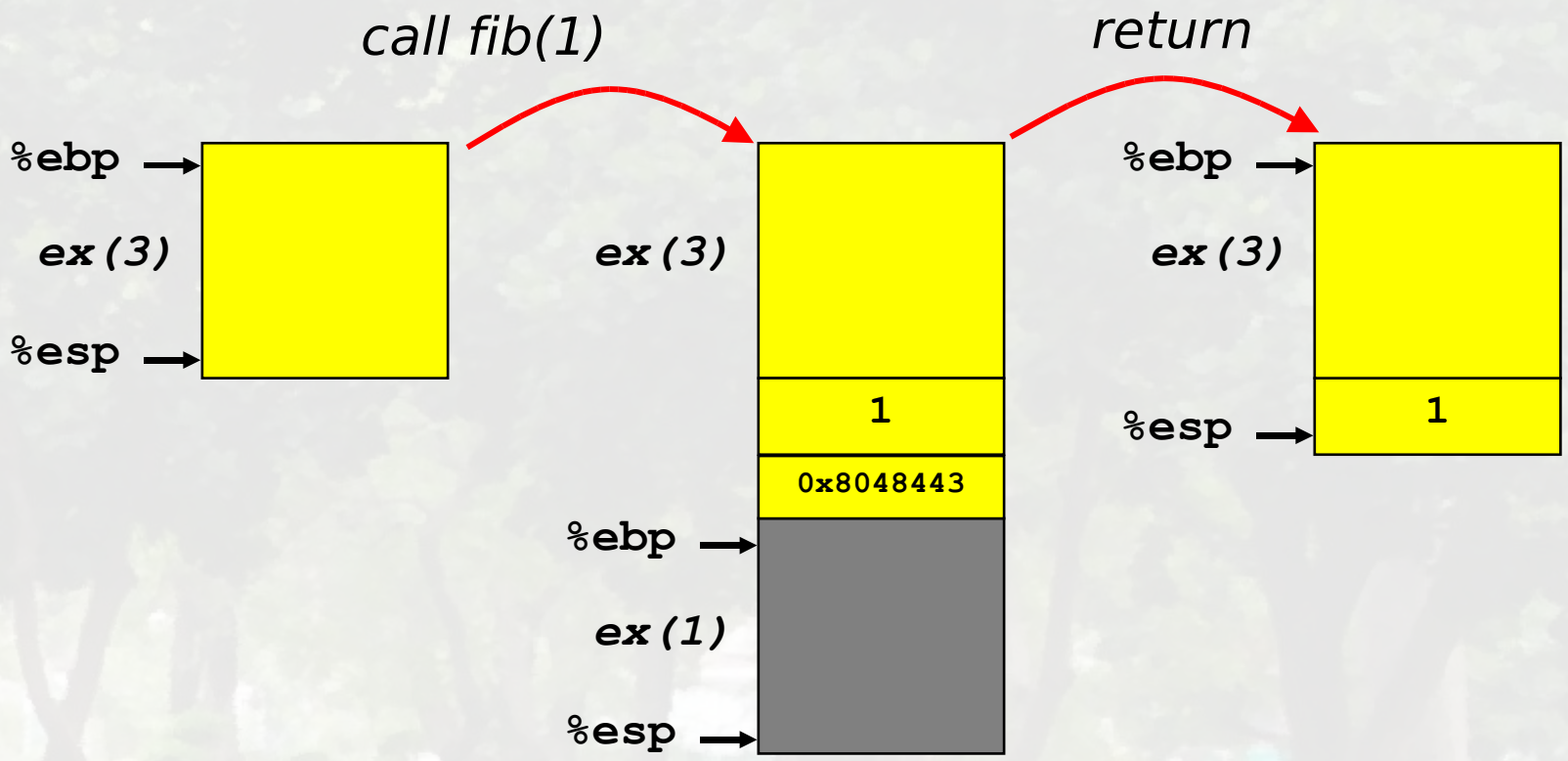
遞迴

fib.c

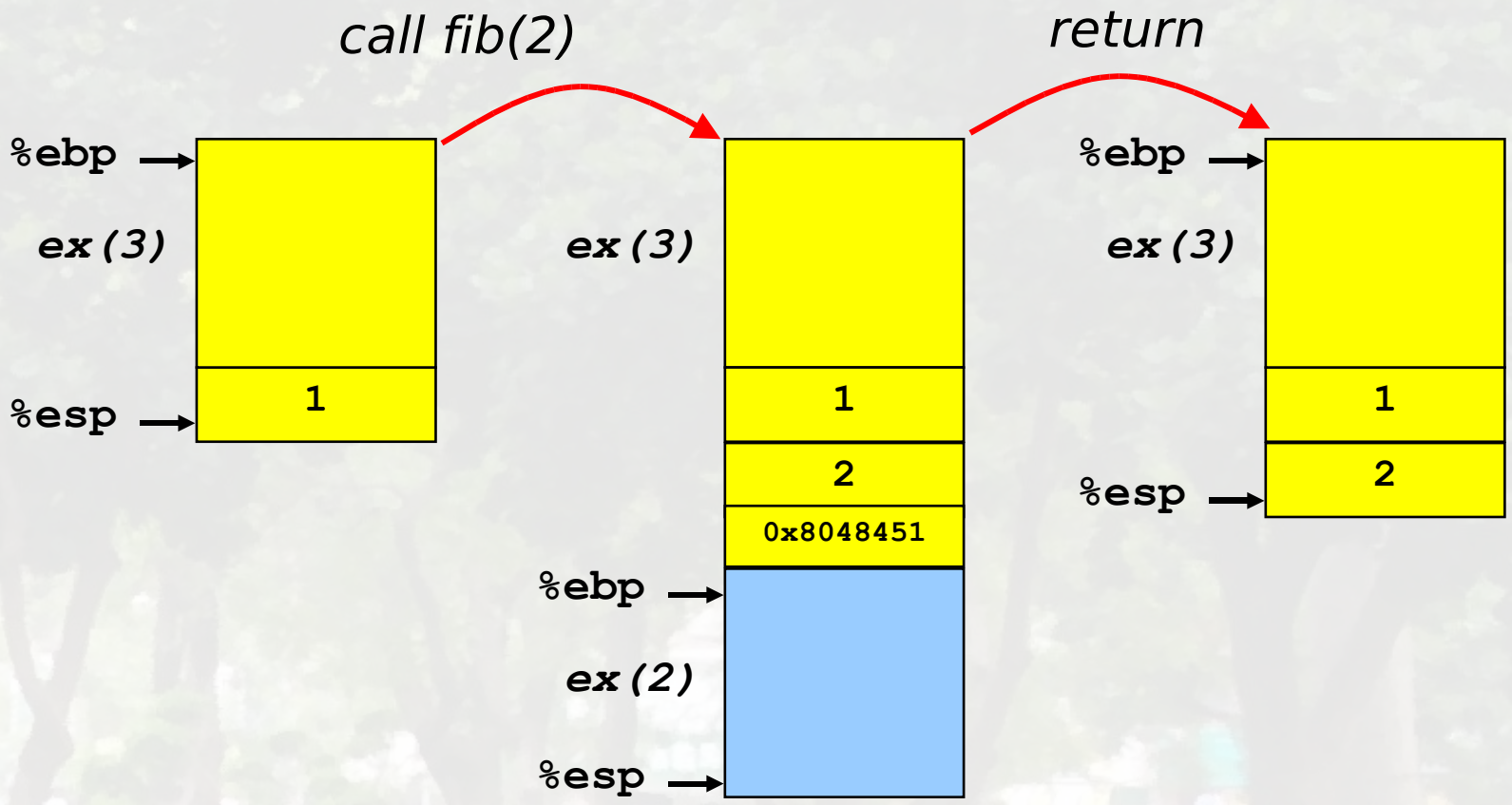
```
(gdb) b fib
Breakpoint 1 at 0x804834b: file fib.c, line 5.
(gdb) r
Breakpoint 1, fib (n=5) at fib.c:5
5       if (n <= 2)
(gdb) c
Continuing.

Breakpoint 1, fib (n=3) at fib.c:5
5       if (n <= 2)
(gdb) c
Continuing.

Breakpoint 1, fib (n=1) at fib.c:5
5       if (n <= 2)
(gdb) info stack
#0  fib (n=1) at fib.c:5
#1  0x08048368 in fib (n=3) at fib.c:8
#2  0x08048368 in fib (n=5) at fib.c:8
#3  0x080483a4 in main () at fib.c:16
```

stack frame 推入的同名但實際上不同 context 的 stack

範例：遞迴 stack frame 變化 (1)

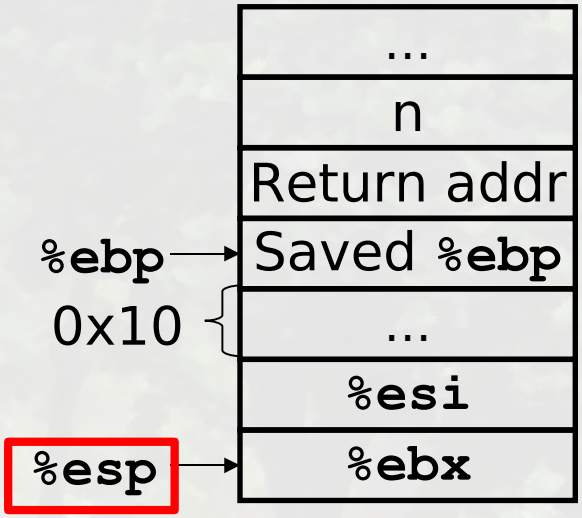


範例：遞迴 stack frame 變化 (2)



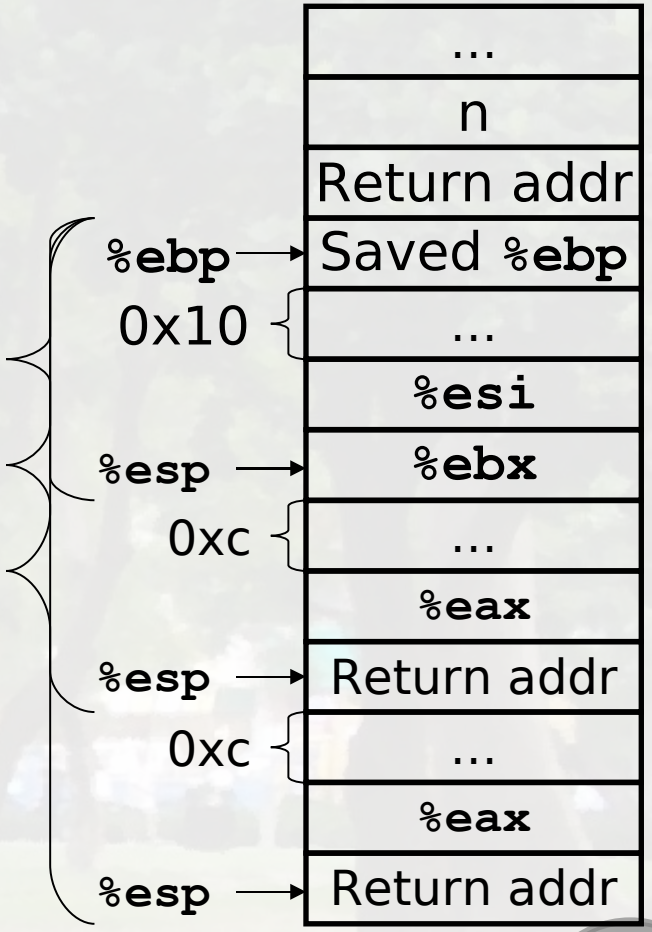
範例：遞迴機械觀點（頭）

```
0x8048420 <fib>:
  push    %ebp
  mov     %esp,%ebp
  sub     $0x10,%esp
  push    %esi
  push    %ebx

# first part of body
  mov     0x8(%ebp),%ebx
  cmp     $0x2,%ebx
  jg      0x8048437 <fib+23>
  mov     $0x1,%eax
  jmp     0x8048453 <fib+51>
```



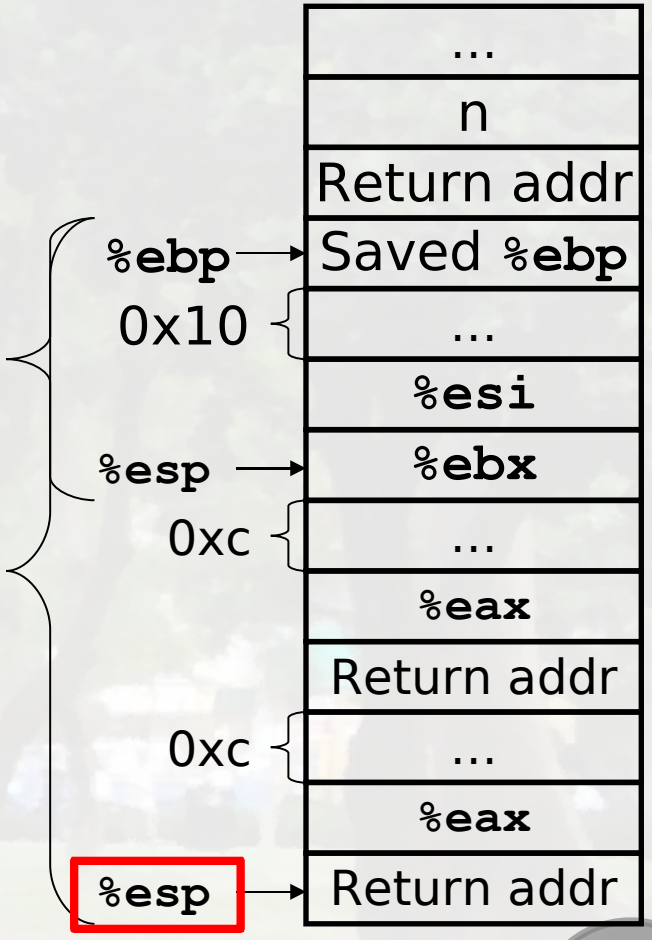
範例：遞迴機械觀點（軀體）

```
0x8048437 <fib+23>
  add     $0xfffffff4,%esp
  lea     0xfffffffe(%ebx),%eax
  push    %eax
  call    0x8048420 <fib>
  mov     %eax,%esi
  add     $0xfffffff4,%esp
  lea     0xffffffff(%ebx),%eax
  push    %eax
  call    0x8048420 <fib>
  add     %esi,%eax
```



範例：遞迴機械觀點(尾)

```
0x8048453 <fib+51>:
  lea     0xffffffe8(%ebp),%esp
  pop     %ebx
  pop     %esi
  mov     %ebp,%esp
  pop     %ebp
  ret
```



GDB 巨集處理

gdb macro

- (gdb) **show user**
  - 查看使用者定義的 macro
- $HOME/.gdbinit
- gdb **-command=**.gdbrc

```
set history save on
set history size 10000
set history filename ~/.gdb_history
set print pretty on
set print static-members off
set charset ASCII
def prun
    tbreak main
    run
    set auto-solib-add 0
    cont
end
def pmail
    tbreak main
    run -mail
    set auto-solib-add 0
    cont
end
```

set

def

gdb macro

- (gdb) **show user**
  - 查看使用者定義的 macro

```
# .gdbinit
# some project specific gdb macros
  document proj_request
  Print a request block
  end
  define proj_request
    print *request_p
  end
```

gdb macro

```
document proj_array
Print a project defined array
Usage: proj_array
end
define proj_array
    set $i = 0
    while ( $i < arrayMax )
        printf “array index %u, value %u\n”, $i, array[$i]
        set $i = $i + 1
    end
end
```

```
document proj_list
Print a project defined array,
Usage proj_list NUMBER
end
define proj_list
    set $i = 0
    while ( $i < $arg0 )
        printf “array index %u, value %u\n”, $i, array[$i]
        set $i = $i + 1
    end
end
```

參考資料

- 「深入淺出 Hello World 」系列演講
  - http://wiki.debian.org.tw/HackingHelloWorld
- 用 Open Source 工具開發軟體： 新軟體開發觀念
  - http://www.study-area.org/cyril/opentools/
- Using the 'gdb' Debugger, Doug Toppin (2006)