侯捷 C++Startup揭密：C++程序的生前和死后

1. 前言，如何自定Startup code

- C++進入點是 main() 嗎？
- 什麼代碼比 main() 更早被執行？
- 什麼代碼在 main() 結束後才被執行？
- 為什麼上述代碼可以如此行為？
- Heap 的結構如何？
- I/O 的結構如何？

我們的目標

- 徹底解決上頁所提的每一項疑問

你應具備的基礎

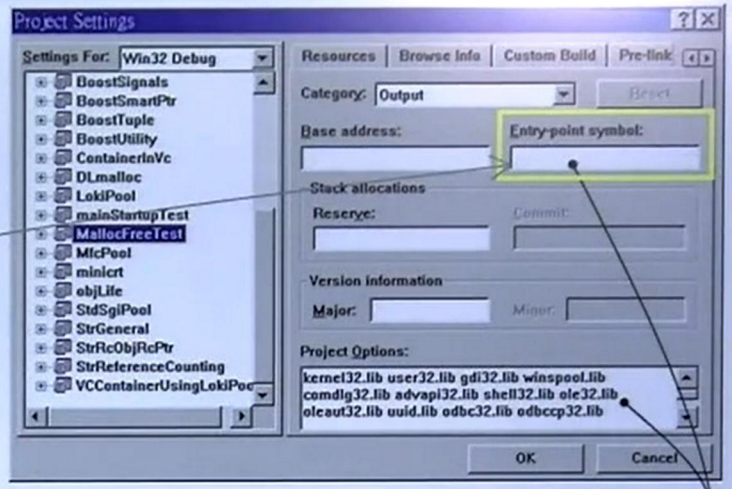
- 是個 C++ programmer

我們所觀察的代碼

VC6, VC10

源碼之前了無秘密

自定 Startup code (Entry-Point Symbol)



/ENTRY (Entry-Point Symbol)

The Entry-Point Symbol (/ENTRY:*function*) option sets the **starting address** for an .EXE file or DLL. (To find this option in the development environment, click **Settings** on the **Project** menu. Then click the **Link** tab, and click **Output** in the **Category** box.)

Type a function name in the **Entry-Point Symbol** text box (or in the *function* argument on the command line). The function must be defined with the **__stdcall** calling convention. The parameters and return value must be defined as documented in the Win32 API for **WinMain** (for an .EXE file) or **DllEntryPoint** (for a DLL). It is recommended that you let the linker set the entry point so that the C run-time library is initialized correctly, and C++ constructors for static objects are executed.

CRT　　這兩件就是 startup code 的主要用途

By default, the starting address is a function name from the C run-time library. The linker selects it according to the attributes of the program, as shown in the following table.

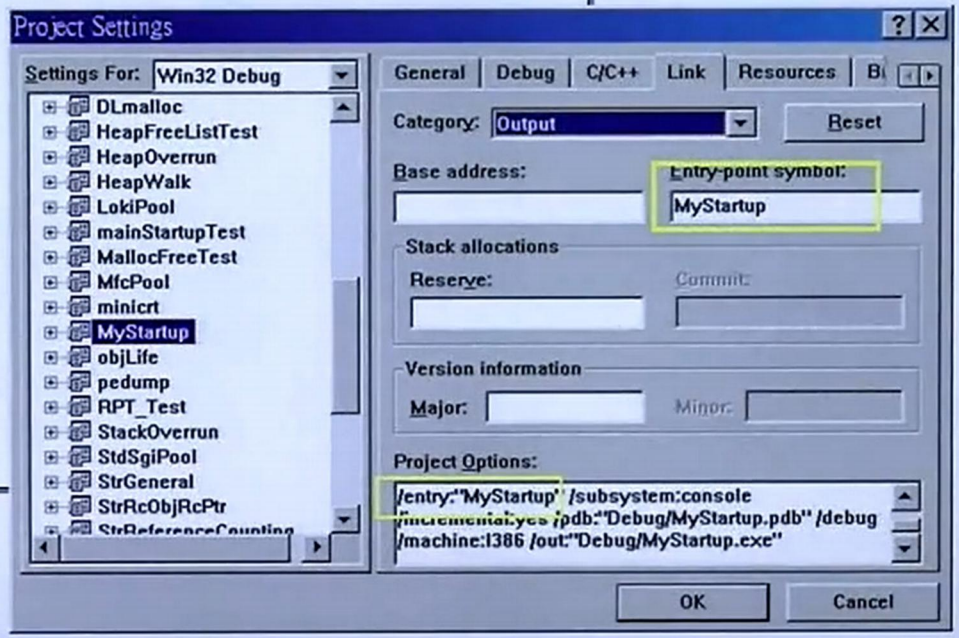
	Function name	Default for
1	**mainCRTStartup** (or **wmainCRTStartup**)	An application using /SUBSYSTEM:**CONSOLE**; calls **main** (or **wmain**)
2	**WinMainCRTStartup** (or **wWinMainCRTStartup**)	An application using /SUBSYSTEM:**WINDOWS**; calls **WinMain** (or **wWinMain**), which must be defined with **__stdcall**
3	**_DllMainCRTStartup**	A DLL; calls **DllMain**, which must be defined with **__stdcall**, if it exists

Startup code 的三個可能形式 (函數名稱)

完成之後 Project Options 會多出 **/entry:"*MyStartup*"**

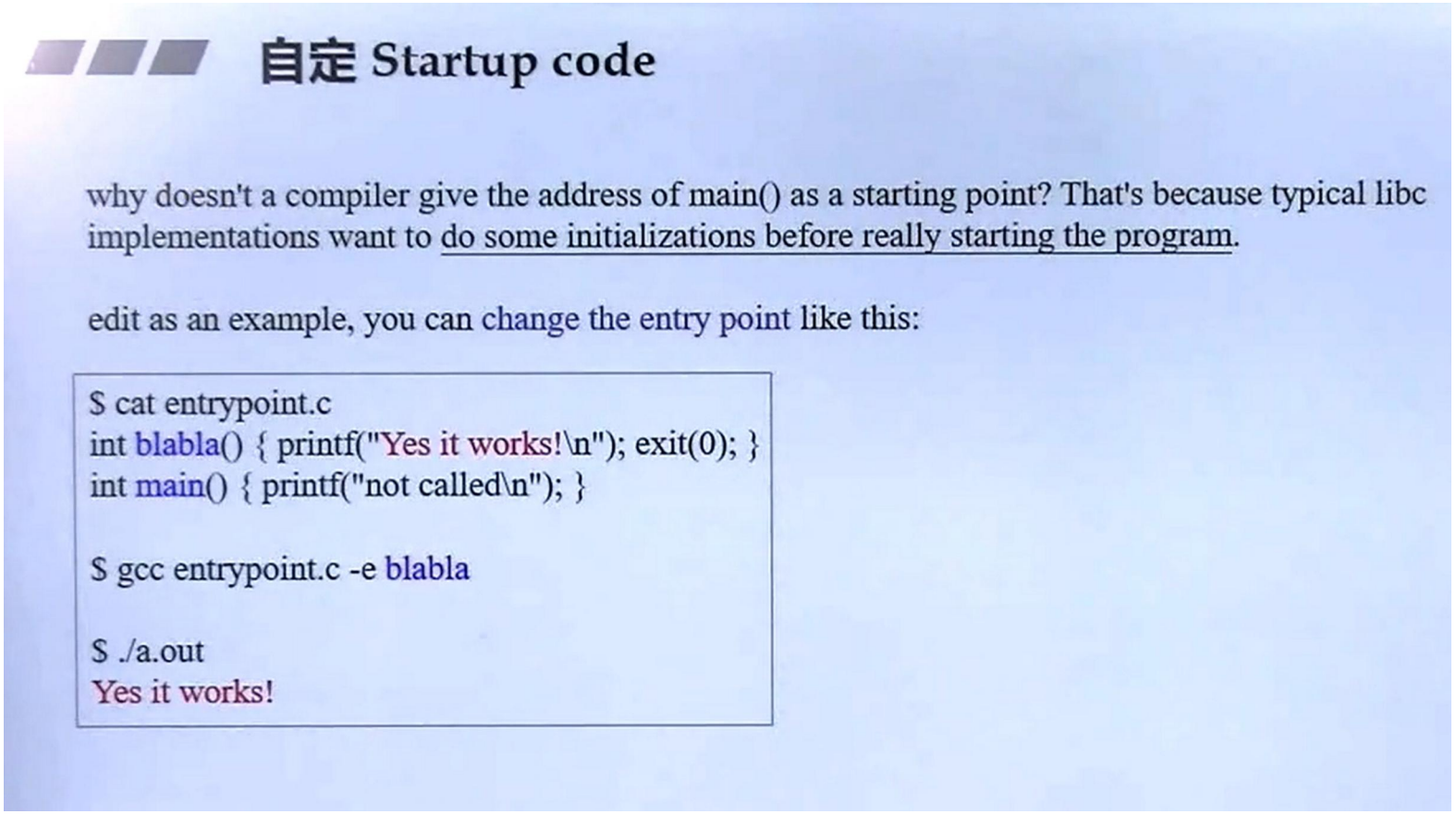
自定 Startup code

```
#include <windows.h>
int MyStartup( void )
{
 int a=10;
 HANDLE crtHeap=HeapCreate(HEAP_NO_SERIALIZE, 0x010, 4000*1024);
 int *p=(int*)HeapAlloc(crtHeap, HEAP_ZERO_MEMORY, 0x010);
 int i,j;

 for(i=0; i<100; i++)
 {
     //int j;
     for(j=0; j<100; j++,p++)
     {
        *p = i*100 + (j+1);
     }
 }

 MessageBoxA(NULL,p,"abcd",MB_OK);
 return 0;
}
```



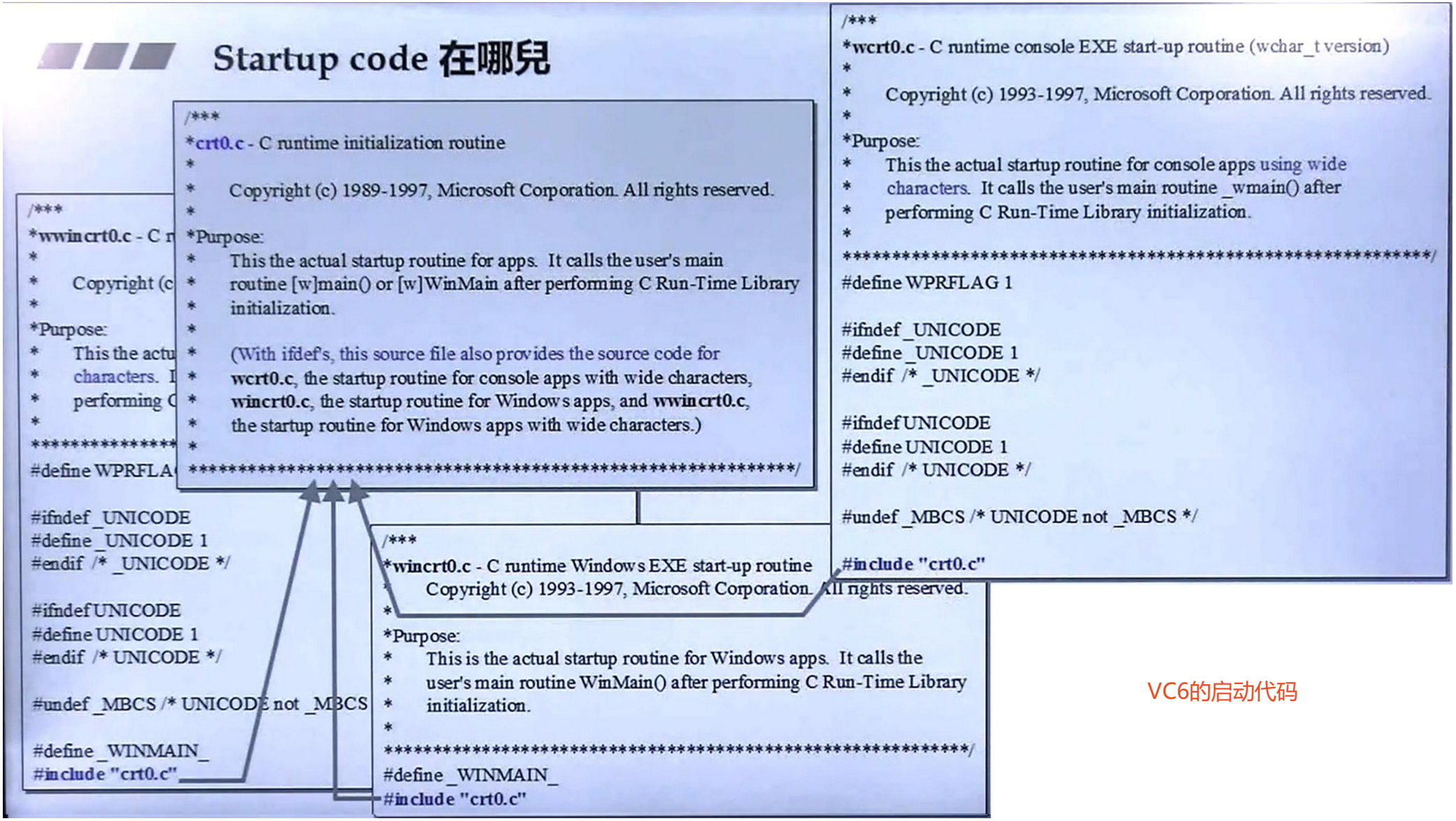
main由Startup code调用

main前调用startup code

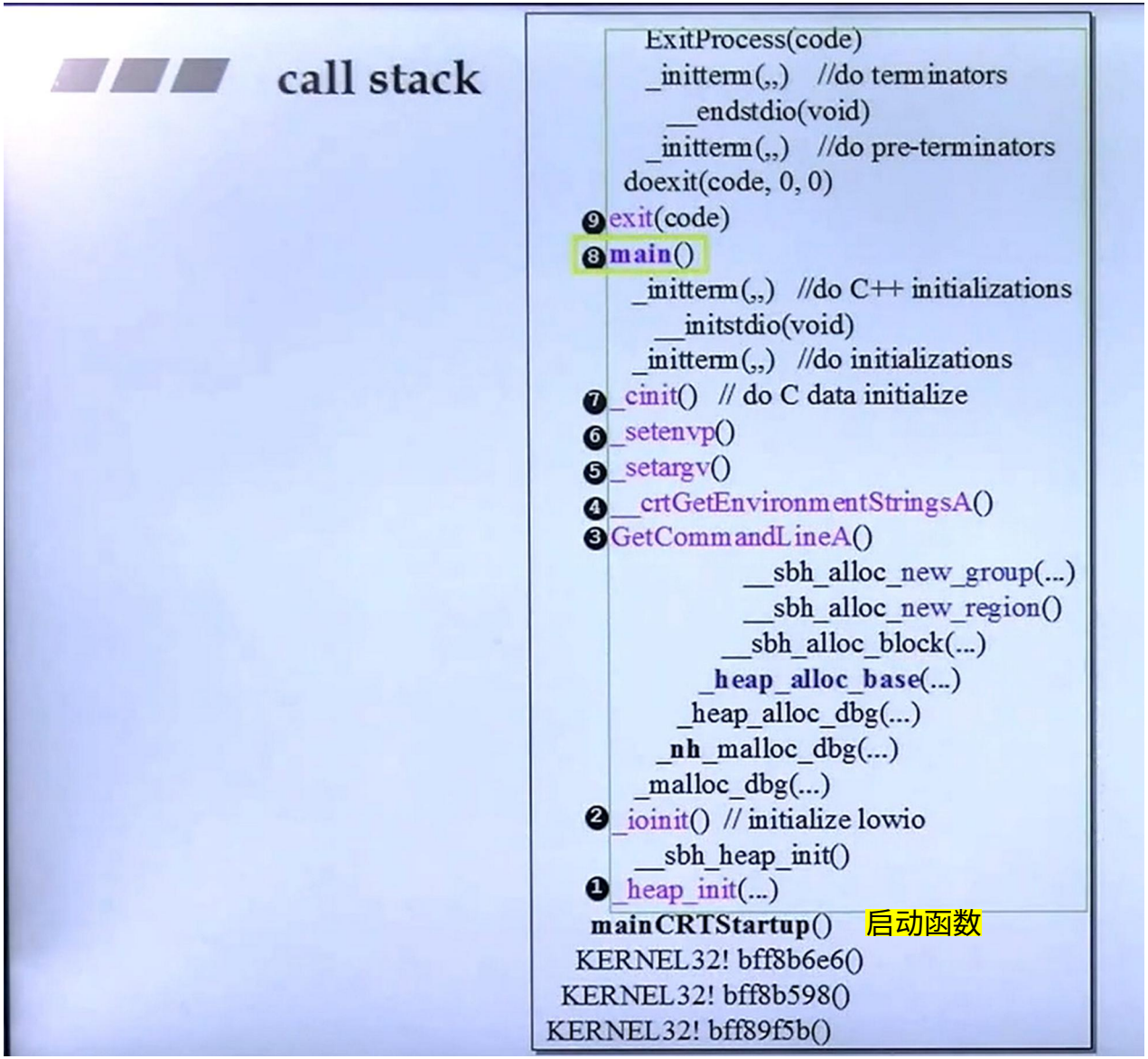


2. 默认的Startup code在哪儿

Startup code 在哪兒



VC6的启动代码



启动函数

从下往上

3. Startup code代码摘要

mainCRTStartup()

VC6

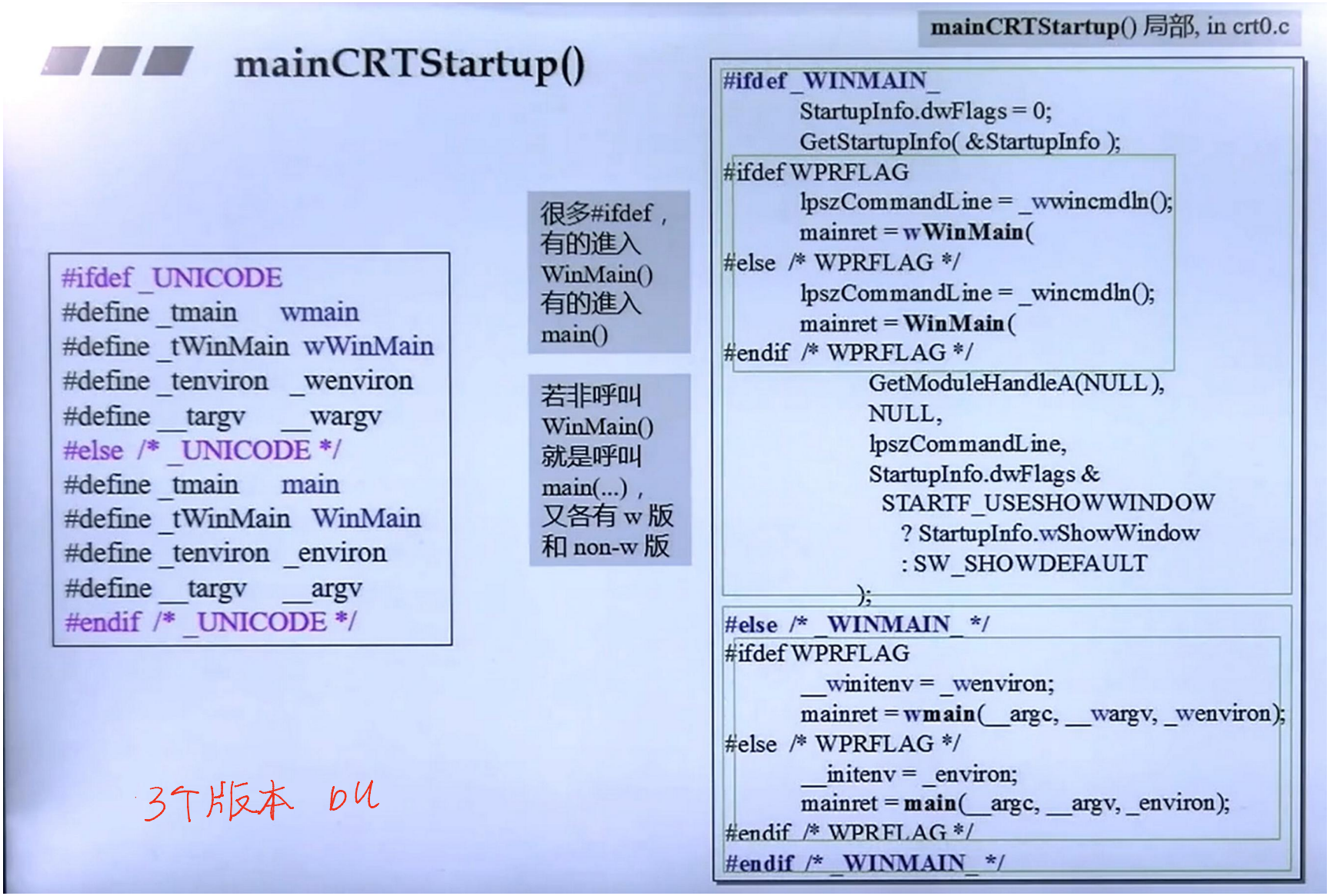
mainCRTStartup() in crt0.c

```
01 void mainCRTStartup(void)
02 {
03     int mainret;
04     // Get the full Win32 version
05     _osver = GetVersion();
06     _winminor = (_osver >> 8) & 0x00FF ;
07     _winmajor = _osver & 0x00FF ;
08     _winver = (_winmajor << 8) + _winminor;
09     _osver = (_osver >> 16) & 0x00FFFF ;
10
11  1  if ( !_heap_init(0) )          /* initialize heap */
12         fast_error_exit(_RT_HEAPINIT);  /* write message and die */
13
```

這是整理後的結果。實際上有很多#ifdef，有的進入 WinMain() 有的進入 main()

heap初始化

```
14     /*
15      * Guard the remainder of the initialization code and the call
16      * to user's main, or WinMain, function in a __try/__except
17      * statement.
18      */
19     __try {                    IO初始化
20  2  _ioinit();                 /* initialize lowio */
21     /* get cmd line info */
22  3  _acmdln = (char *)GetCommandLineA();
23     /* get environ info */
24  4  _aenvptr = (char *)__crtGetEnvironmentStringsA();
25  5  _setargv();                字符串处理
26  6  _setenvp();
27  7  _cinit();      /* do C data initialize */
28  8  __initenv = _environ;
29  9  mainret = main(__argc, __argv, _environ);
30     exit(mainret);                  ^——>注意 calling convention
31     }
32     __except ( _XcptFilter(GetExceptionCode(),
33                            GetExceptionInformation()) )
34     {
35        // Should never reach here
36        _exit( GetExceptionCode() );
37     } /* end of try - except */
38 }
```



4. Heap_init() - Startup的首要工程

VC6 _heap_init()

我的《C++內存管理機制》(全五講) 中的第三講 malloc/free 對於 CRT 的 SBH 設計和運行有極詳細的說明。本課程此處之 _heap_init() 及其引發的 memory 分配與管理, 是屬相同主題。

本處之特別：

1, 藉 Startup code 處理字符串 (包括命令行參數, 環境變數) 過程中的詳細數據觀察，釐清、確認和加強所有環節。

2, 增加說明 Windows Heap (操作系統層) 的結構和管理。(當 CRT's SBH 不再存在時)

```
    ExitProcess(code)
    _initterm(,,)   //do terminators
      __endstdio(void)
    _initterm(,,)   //do pre-terminators
  doexit(code, 0, 0)
9 exit(code)
8 main()
    _initterm(,,)   //do C++ initializations
      __initstdio(void)
    _initterm(,,)   //do initializations
7 _cinit()  // do C data initialize
6 _setenvp()
5 _setargv()
4 __crtGetEnvironmentStringsA()
3 GetCommandLineA()
          __sbh_alloc_new_group(...)
          __sbh_alloc_new_region()
           sbh_alloc_block(...)
        heap_alloc_base(...)
      _heap_alloc_dbg(...)
    _nh_malloc_dbg(...)
  _malloc_dbg(...)
2 _ioinit()  // initialize lowio
    __sbh_heap_init()
1 _heap_init(...)
  mainCRTStartup()
KERNEL32! bff8b6e6()
KERNEL32! bff8b598()
KERNEL32! bff89f5b()
```

小区块管理

SBH: Small Block Heap

何謂 small ?

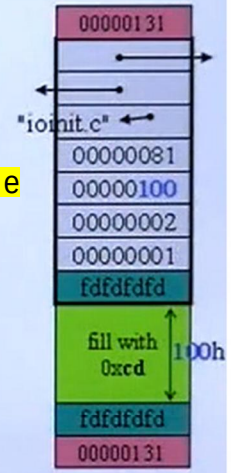
内存塊
1, 從何處來 ?
2, 大小幾何 ?
3, 回收至何處 ?

调用至此时还没有加cookie
所以1024-8 = 1016

门槛

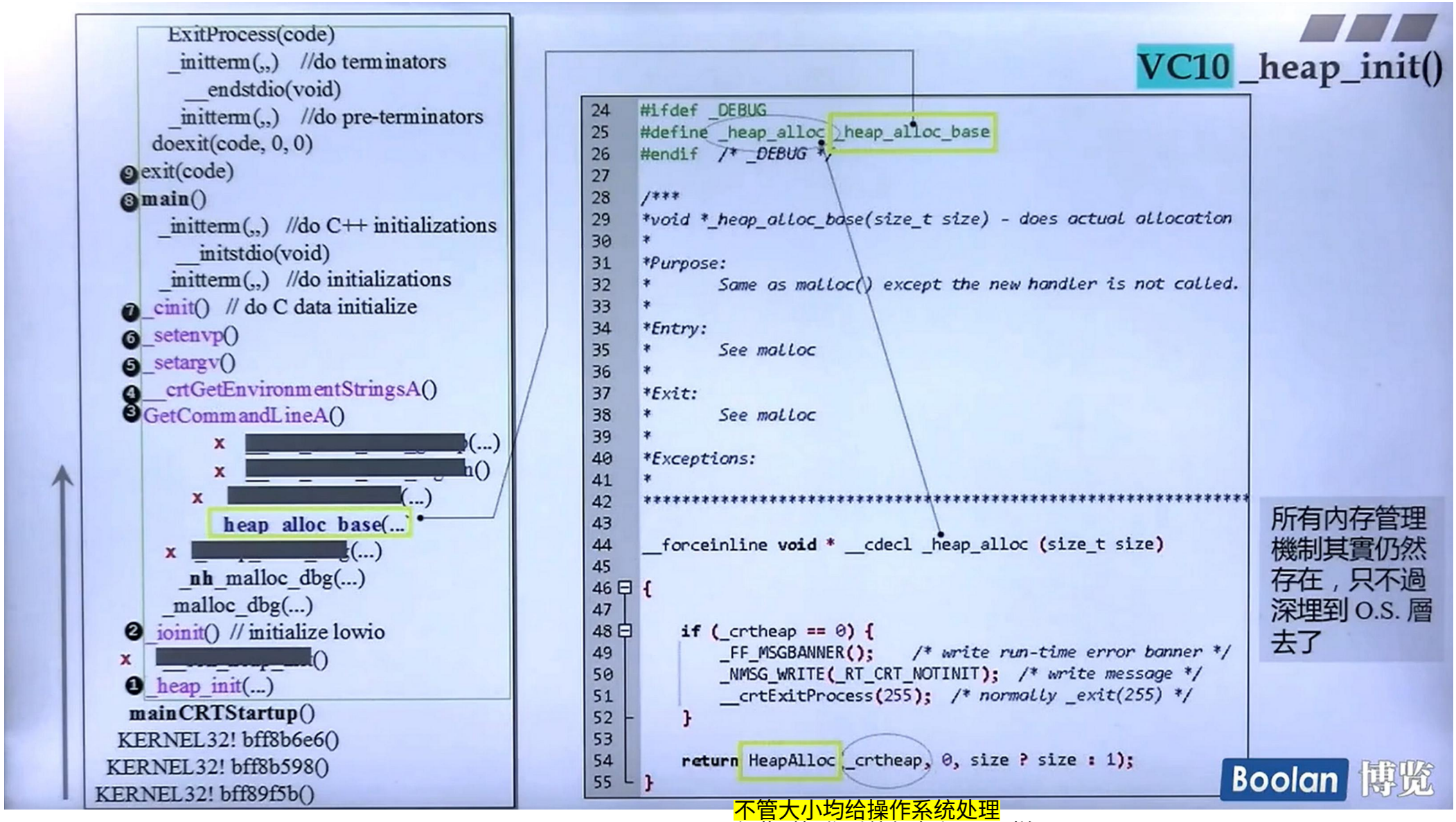
```
if (size <= __sbh_threshold) {    //3F8, i.e. 1016
    pvReturn = __sbh_alloc_block(size);
    if (pvReturn)  return pvReturn;
}
if (size == 0)  size = 1;
size = (size + ...) & ~(...);
return HeapAlloc(_crtheap, 0, size);
```

大于1K的让操作系统处理



_ioinit()包装(Debug模式下)
包装完后加cookie表边界
最后再调整为16的倍数

VC10 _heap_init()



不管大小均给操作系统处理
但此时操作系统行为和VC6一样

SBH 之始 – _heap_init() 和 __sbh_heap_init()

CRT 會先為自己建立一個 _crtheap，然後從中分配 SBH 所需的 headers, regions 做為管理之用。App. 動態分配時若 size > threshold 就調用 HeapAlloc() 從 _crtheap 取。若 size <= threshold 就徑自從 SBH 取 (實際區塊來自 VirtualAlloc())

heapinit.c

```
int __cdecl _heap_init (
        int mtflag
        )
{
        // Initialize the "big-block" heap first.
        if ( (_crtheap = HeapCreate( mtflag ? 0 : HEAP_NO_SERIALIZE,
                                     BYTES_PER_PAGE, 0 )) == NULL )
            return 0;

        // Initialize the small-block heap
        if (__sbh_heap_init() == 0)
        {
            HeapDestroy(_crtheap);
            return 0;
        }
        return 1;
}
```

建立一个池塘pool
从池塘中挖出16个Headers

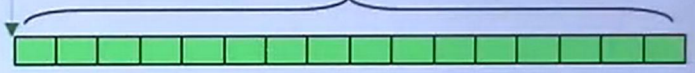
4096

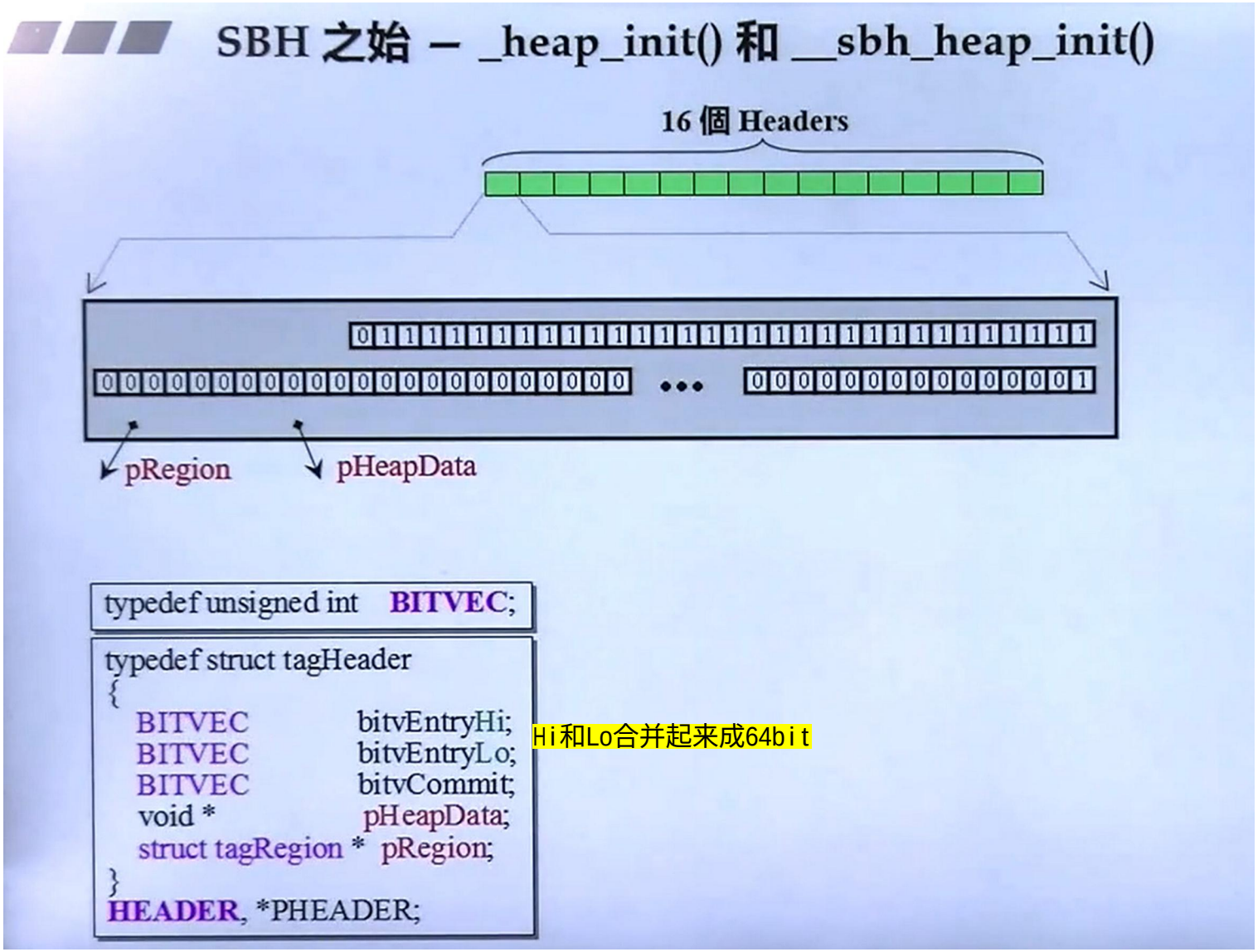
不論 big-block heap or small-block heap，只要失敗就 return 0，別玩了。

sbheap.c

```
int __cdecl __sbh_heap_init (void)
{
    if (!(__sbh_pHeaderList =
          HeapAlloc(_crtheap, 0, 16 * sizeof(HEADER))))
        return FALSE;

    __sbh_pHeaderScan = __sbh_pHeaderList;
    __sbh_pHeaderDefer = NULL;
    __sbh_cntHeaderList = 0;
    __sbh_sizeHeaderList = 16;

    return TRUE;
}
```

16 個 Headers



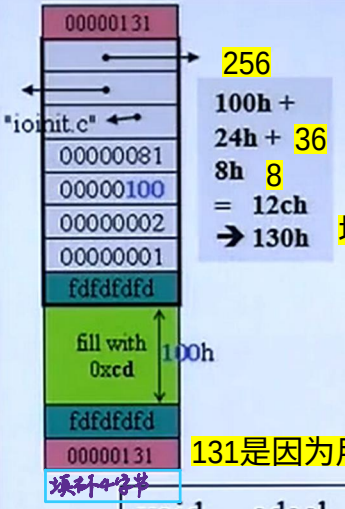

SBH 之始 – _heap_init() 和 __sbh_heap_init()
16 個 Headers
pRegion
pHeapData
typedef unsigned int BITVEC;
typedef struct tagHeader
{
BITVEC bitvEntryHi;
BITVEC bitvEntryLo;
BITVEC bitvCommit;
void * pHeapData;
struct tagRegion * pRegion;
}
HEADER, *PHEADER;
Hi和Lo合并起来成64bit

5. 内存分配精解1

VC6 内存分配

```
ExitProcess(code)
  _initterm(,,)   //do terminators
    __endstdio(void)
  _initterm(,,)   //do pre-terminators
 doexit(code, 0, 0)
9 exit(code)
8 main()
  _initterm(,,)   //do C++ initializations
    __initstdio(void)
  _initterm(,,)   //do initializations
7 _cinit()  // do C data initialize
6 _setenvp()
5 _setargv()
4 __crtGetEnvironmentStringsA()
3 GetCommandLineA()
          __sbh_alloc_new_group(...)
        __sbh_alloc_new_region()
      __sbh_alloc_block(...)
    _heap_alloc_base(...)
  _heap_alloc_dbg(...)
_nh_malloc_dbg(...)
 malloc_dbg(...)
2 ioinit()  // initialize lowio
  __sbh_heap_init()
1 _heap_init(...)
 mainCRTStartup()
KERNEL32! bff8b6e6()
KERNEL32! bff8b598()
KERNEL32! bff89f5b()
```

```
/* Memory block identification */
#define _FREE_BLOCK      0
#define _NORMAL_BLOCK    1
#define _CRT_BLOCK       2
#define _IGNORE_BLOCK    3
#define _CLIENT_BLOCK    4
#define _MAX_BLOCKS      5
```



256
100h + 24h + 36
8h 8
= 12ch
➔ 130h

填补为16倍数

```
#ifndef _DEBUG
#define _malloc_crt   malloc
...
#else  /* _DEBUG */
#define _THISFILE     __FILE__
#define _malloc_crt(s)   _malloc_dbg
                           (s, _CRT_BLOCK,
                            _THISFILE, __LINE__)
...
#endif  /* _DEBUG */
```

登记文件中第几行

131是因为用最后一个bit登记状态

```
typedef struct {
    long osfhnd;
    char osfile;
    char pipech;
} ioinfo;
```

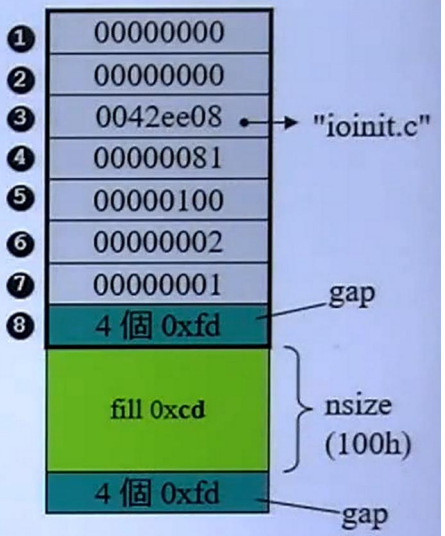
```
void __cdecl _ioinit (void)
{
...
line81  if ( (pio = _malloc_crt( IOINFO_ARRAY_ELTS * sizeof(ioinfo) ))
                                                          32        8
            == NULL )
...
}
```

第1次分配内存大小256

VC6 內存分配

```
ExitProcess(code)
_initterm(,,)   //do terminators
  __endstdio(void)
_initterm(,,)   //do pre-terminators
doexit(code, 0, 0)
9 exit(code)
8 main()
  _initterm(,,)   //do C++ initializations
    __initstdio(void)
  _initterm(,,)   //do initializations
7 _cinit()  // do C data initialize
6 _setenvp()
5 _setargv()
4 __crtGetEnvironmentStringsA()
3 GetCommandLineA()
          __sbh_alloc_new_group(...)
          __sbh_alloc_new_region()
          __sbh_alloc_block(...)
        heap_alloc_base(...)
      heap_alloc_dbg(...)
    _nh_malloc_dbg(...)
  _malloc_dbg(...)
2 _ioinit()  // initialize lowio
    __sbh_heap_init()
1 _heap_init(...)
mainCRTStartup()
KERNEL32! bff8b6e6()
KERNEL32! bff8b598()
KERNEL32! bff89f5b()
```

```
#define nNoMansLandSize 4
typedef struct _CrtMemBlockHeader
{
    struct _CrtMemBlockHeader * pBlockHeaderNext;
    struct _CrtMemBlockHeader * pBlockHeaderPrev;
    char *          szFileName;
    int             nLine;
    size_t          nDataSize;
    int             nBlockUse;
    long            lRequest;
    unsigned char   gap[nNoMansLandSize];
    /* followed by:
     *  unsigned char  data[nDataSize];
     *  unsigned char  anotherGap[nNoMansLandSize];
     */
} _CrtMemBlockHeader;
```




```
…
blockSize = sizeof(_CrtMemBlockHeader) + nSize + nNoMansLandSize;   ➔ 調整大小
…
pHead = (_CrtMemBlockHeader *)_heap_alloc_base(blockSize);   ➔ 從 SBH 擷取一內存塊
… (續下頁)   ➔ 設妥 Debug Heap
```

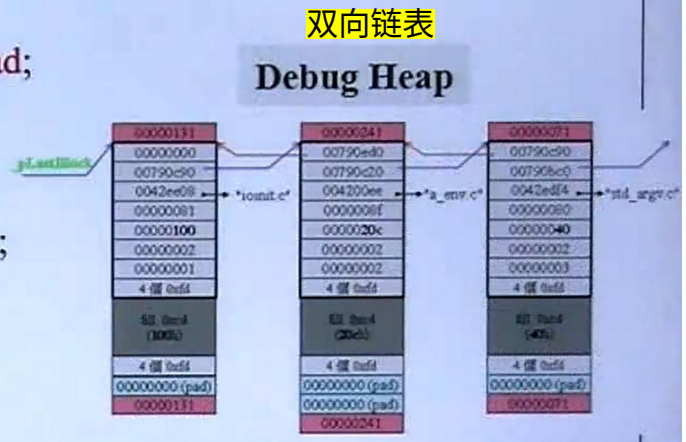
VC6 内存分配

每个Debug模式下的区块都会被串起来(即使已经分配给了客户

```
ExitProcess(code)
  _initterm(,,)   //do terminators
    __endstdio(void)
  _initterm(,,)   //do pre-terminators
 doexit(code, 0, 0)
9 exit(code)
8 main()
  _initterm(,,)   //do C++ initializations
    __initstdio(void)
  _initterm(,,)   //do initializations
7 _cinit()  // do C data initialize
6 _setenvp()
5 _setargv()
4 __crtGetEnvironmentStringsA()
3 GetCommandLineA()
          __sbh_alloc_new_group(...)
          __sbh_alloc_new_region()
          __sbh_alloc_block(...)
        heap_alloc_base(...)
      heap_alloc_dbg(...)
    _nh_malloc_dbg(...)
  _malloc_dbg(...)
2 _ioinit()  // initialize lowio
    __sbh_heap_init()
1 _heap_init(...)
  mainCRTStartup()
KERNEL32! bff8b6e6()
KERNEL32! bff8b598()
KERNEL32! bff89f5b()
```

```
... (承上頁)
if ( _pFirstBlock)
    _pFirstBlock->pBlockHeaderPrev = pHead;
else
    _pLastBlock = pHead;

pHead->pBlockHeaderNext =  _pFirstBlock;
pHead->pBlockHeaderPrev = NULL;
pHead->szFileName = (char*)szFileName;
pHead->nLine = nLine;
pHead->nDataSize = nSize;
pHead->nBlockUse = nBlockUse;
pHead->lRequest = lRequest;

/* link blocks together */
_pFirstBlock = pHead;

/* fill in gap before and after real block */
memset((void *)pHead->gap, _bNoMansLandFill, nNoMansLandSize);
memset((void *)(pbData(pHead) + nSize), _bNoMansLandFill, nNoMansLandSize);
/* fill data with silly value (but non-zero) */
memset((void *)pbData(pHead), _bCleanLandFill, nSize);
return (void *)pbData(pHead);
```

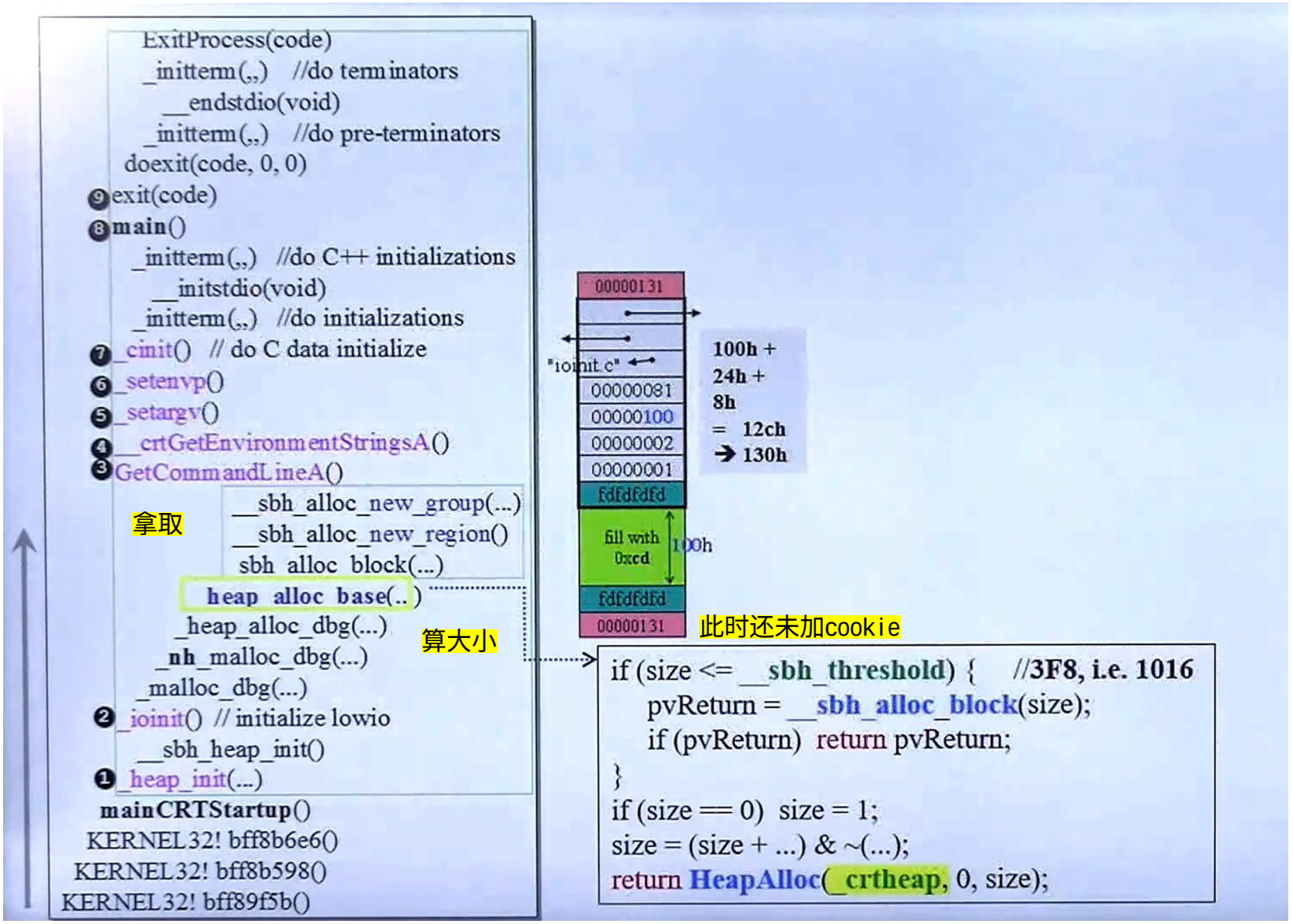
双向链表



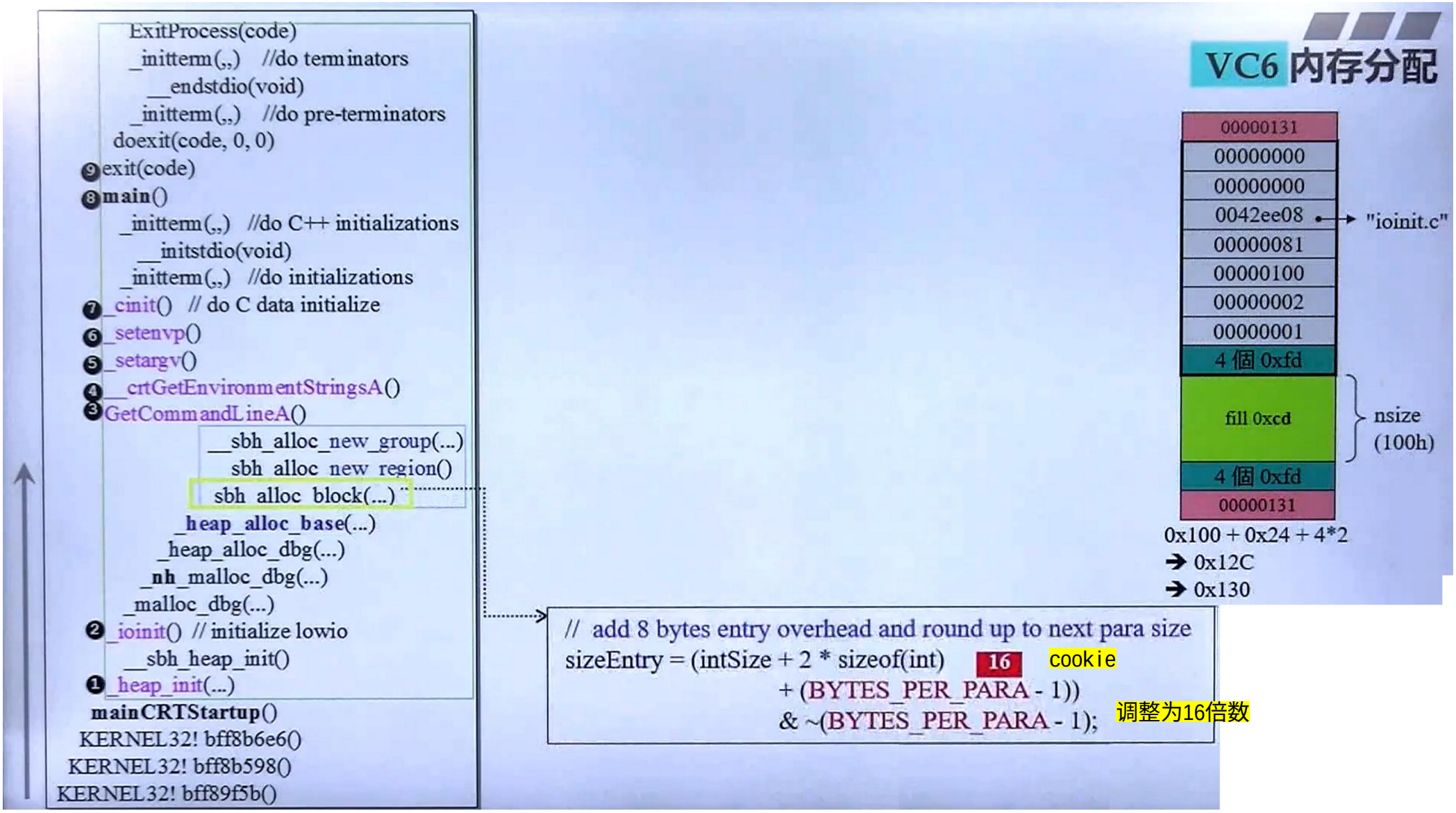
设定的初值

```
static unsigned char _bNoMansLandFill = 0xFD;
static unsigned char _bDeadLandFill   = 0xDD;
static unsigned char _bCleanLandFill  = 0xCD;

static _CrtMemBlockHeader * _pFirstBlock;
static _CrtMemBlockHeader * _pLastBlock;
```

Boolan 博览

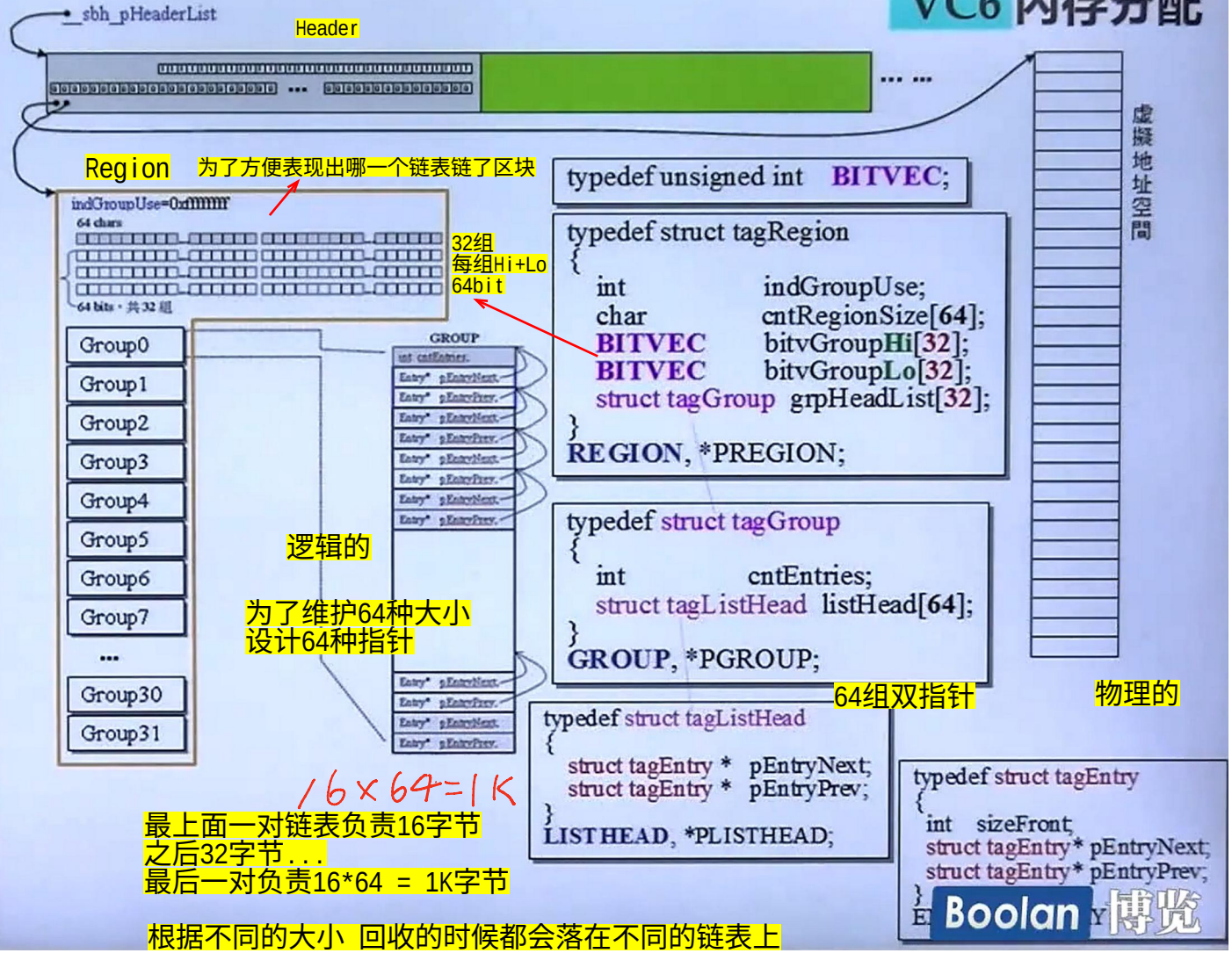

ExitProcess(code)
_initterm(,,) //do terminators
__endstdio(void)
_initterm(,,) //do pre-terminators
doexit(code, 0, 0)
9 exit(code)
8 main()
_initterm(,,) //do C++ initializations
__initstdio(void)
_initterm(,,) //do initializations
7 _cinit() // do C data initialize
6 _setenvp()
5 _setargv()
4 __crtGetEnvironmentStringsA()
3 GetCommandLineA()
拿取
__sbh_alloc_new_group(...)
__sbh_alloc_new_region()
sbh_alloc_block(...)
heap_alloc_base(..)
_heap_alloc_dbg(...)
_nh_malloc_dbg(...)
_malloc_dbg(...)
2 _ioinit() // initialize lowio
__sbh_heap_init()
1 _heap_init(...)
mainCRTStartup()
KERNEL32! bff8b6e6()
KERNEL32! bff8b598()
KERNEL32! bff89f5b()
算大小
00000131
"ioinit.c"
00000081
00000100
00000002
00000001
fdfdfdfd
fill with 0xcd
100h
fdfdfdfd
00000131
100h +
24h +
8h
= 12ch
➔ 130h
此时还未加cookie
if (size <= __sbh_threshold) { //3F8, i.e. 1016
pvReturn = __sbh_alloc_block(size);
if (pvReturn) return pvReturn;
}
if (size == 0) size = 1;
size = (size + ...) & ~(...);
return HeapAlloc(_crtheap, 0, size);

6. 内存分配精解2


ExitProcess(code)
_initterm(,,) //do terminators
__endstdio(void)
_initterm(,,) //do pre-terminators
doexit(code, 0, 0)
9 exit(code)
8 main()
_initterm(,,) //do C++ initializations
__initstdio(void)
_initterm(,,) //do initializations
7 _cinit() // do C data initialize
6 _setenvp()
5 _setargv()
4 __crtGetEnvironmentStringsA()
3 GetCommandLineA()
__sbh_alloc_new_group(...)
sbh_alloc_new_region()
sbh_alloc_block(...)
_heap_alloc_base(...)
_heap_alloc_dbg(...)
_nh_malloc_dbg(...)
_malloc_dbg(...)
2 _ioinit() // initialize lowio
__sbh_heap_init()
1 _heap_init(...)
mainCRTStartup()
KERNEL32! bff8b6e6()
KERNEL32! bff8b598()
KERNEL32! bff89f5b()
// add 8 bytes entry overhead and round up to next para size
sizeEntry = (intSize + 2 * sizeof(int) 16 cookie
+ (BYTES_PER_PARA - 1))
& ~(BYTES_PER_PARA - 1); 调整为16倍数
VC6 内存分配
00000131
00000000
00000000
0042ee08
"ioinit.c"
00000081
00000100
00000002
00000001
4 個 0xfd
fill 0xcd
nsize
(100h)
4 個 0xfd
00000131
0x100 + 0x24 + 4*2
0x12C
0x130

VC6 内存分配

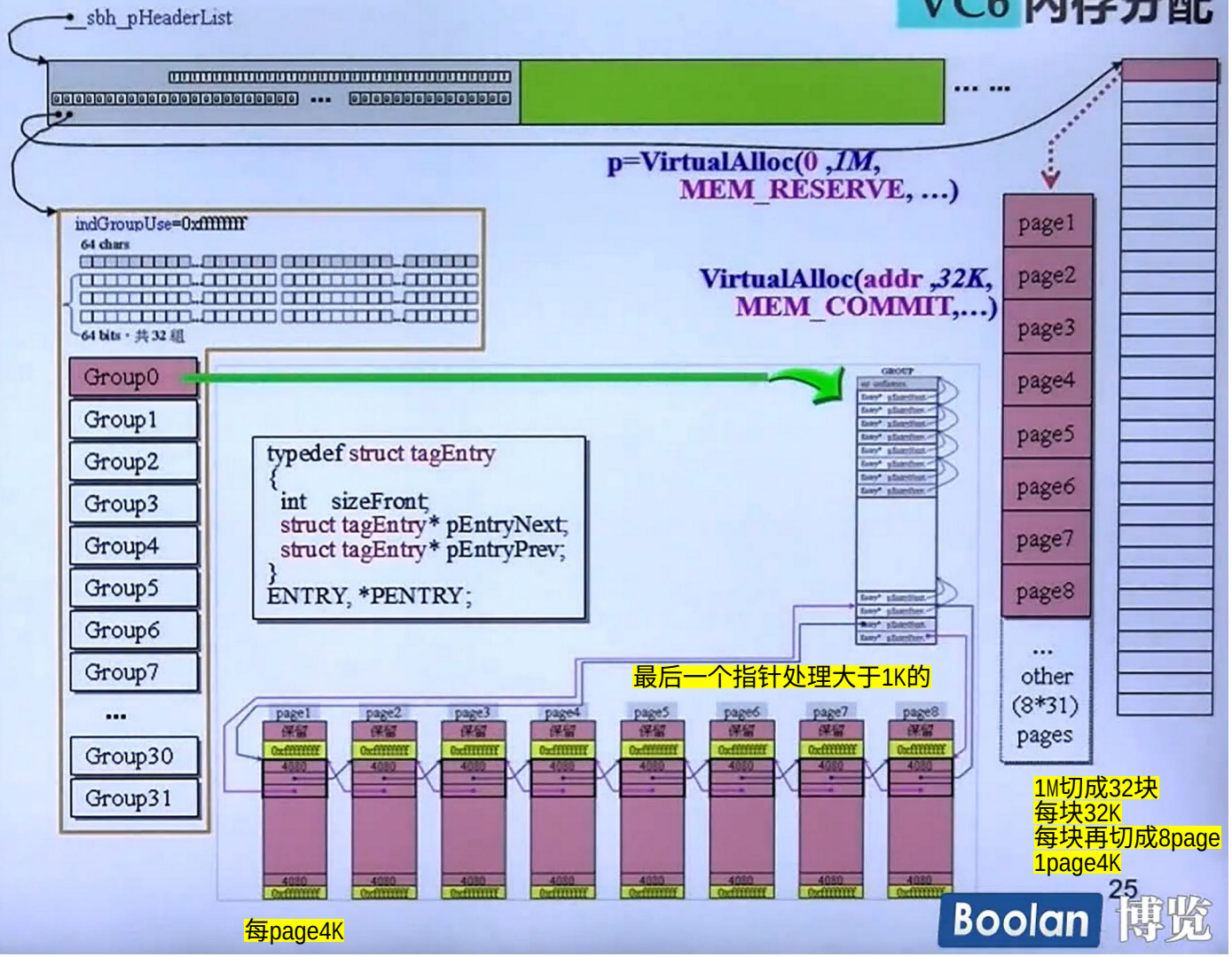
```
ExitProcess(code)
  _initterm(,,)   //do terminators
    __endstdio(void)
  _initterm(,,)   //do pre-terminators
doexit(code, 0, 0)
9 exit(code)
8 main()
  _initterm(,,)   //do C++ initializations
    __initstdio(void)
  _initterm(,,)   //do initializations
7 _cinit()  // do C data initialize
6 _setenvp()
5 _setargv()
4 __crtGetEnvironmentStringsA()
3 GetCommandLineA()
            sbh_alloc_new_group(...)
            sbh_alloc_new_region()
          __sbh_alloc_block(...)
        _heap_alloc_base(...)
      _heap_alloc_dbg(...)
    _nh_malloc_dbg(...)
  _malloc_dbg(...)
2 _ioinit()  // initialize lowio
    __sbh_heap_init()
1 _heap_init(...)
  mainCRTStartup()
KERNEL32! bff8b6e6()
KERNEL32! bff8b598()
KERNEL32! bff89f5b()
```

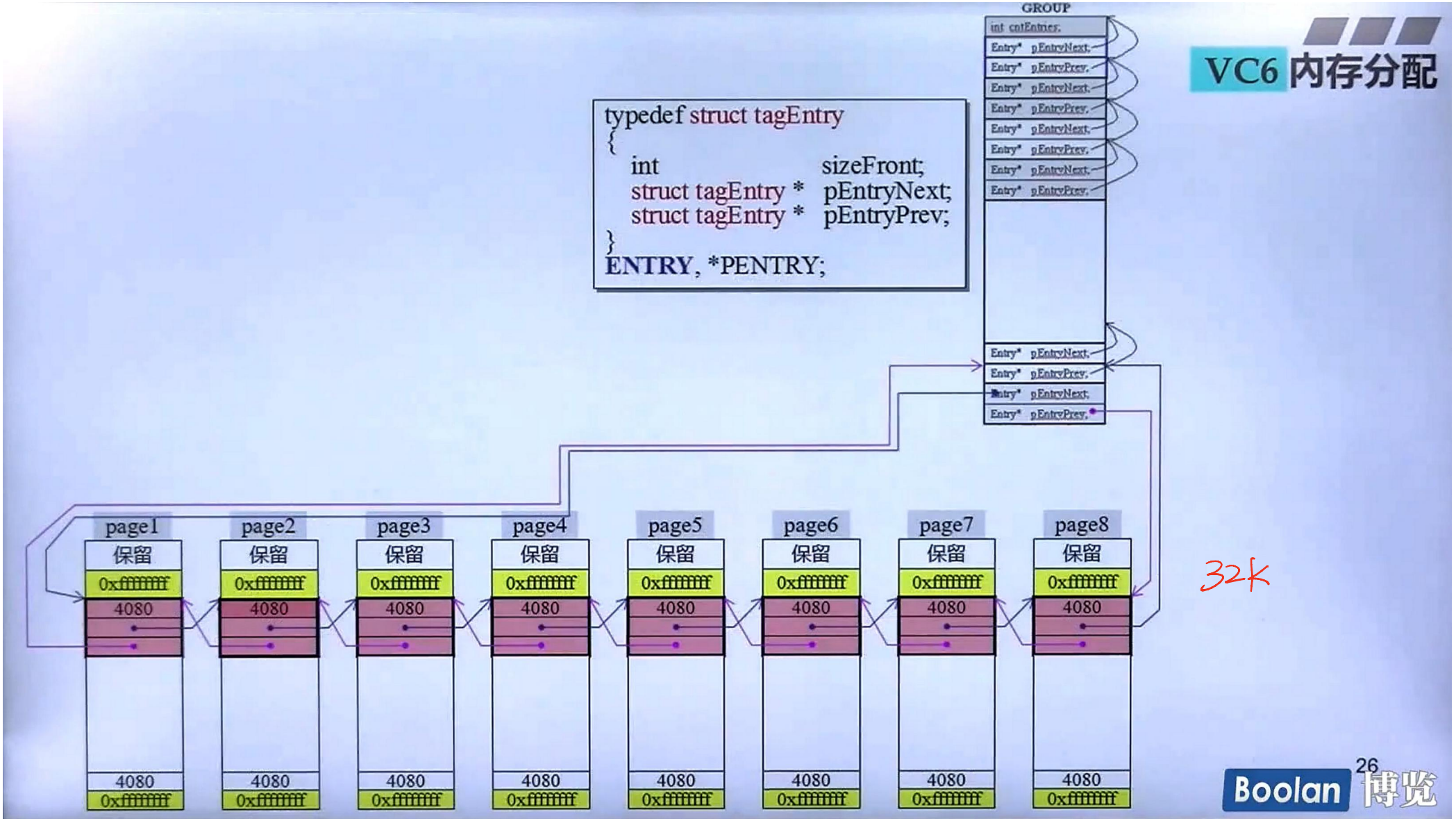



```
typedef unsigned int    BITVEC;

typedef struct tagRegion
{
    int             indGroupUse;
    char            cntRegionSize[64];
    BITVEC          bitvGroupHi[32];
    BITVEC          bitvGroupLo[32];
    struct tagGroup grpHeadList[32];
}
REGION, *PREGION;

typedef struct tagGroup
{
    int             cntEntries;
    struct tagListHead listHead[64];
}
GROUP, *PGROUP;

typedef struct tagListHead
{
    struct tagEntry *  pEntryNext;
    struct tagEntry *  pEntryPrev;
}
LISTHEAD, *PLISTHEAD;

typedef struct tagEntry
{
    int    sizeFront;
    struct tagEntry* pEntryNext;
    struct tagEntry* pEntryPrev;
}
```

VC6 内存分配

```
ExitProcess(code)
  _initterm(,,)   //do terminators
    __endstdio(void)
  _initterm(,,)   //do pre-terminators
 doexit(code, 0, 0)
9 exit(code)
8 main()
  _initterm(,,)   //do C++ initializations
    __initstdio(void)
  _initterm(,,)   //do initializations
7 _cinit()  // do C data initialize
6 _setenvp()
5 _setargv()
4 __crtGetEnvironmentStringsA()
3 GetCommandLineA()
          __sbh_alloc_new_group(...)
        __sbh_alloc_new_region()
      __sbh_alloc_block(...)
    _heap_alloc_base(...)
  _heap_alloc_dbg(...)
 _nh_malloc_dbg(...)
 _malloc_dbg(...)
2 _ioinit()  // initialize lowio
    __sbh_heap_init()
1 _heap_init(...)
 mainCRTStartup()
KERNEL32! bff8b6e6()
KERNEL32! bff8b598()
KERNEL32! bff89f5b()
```



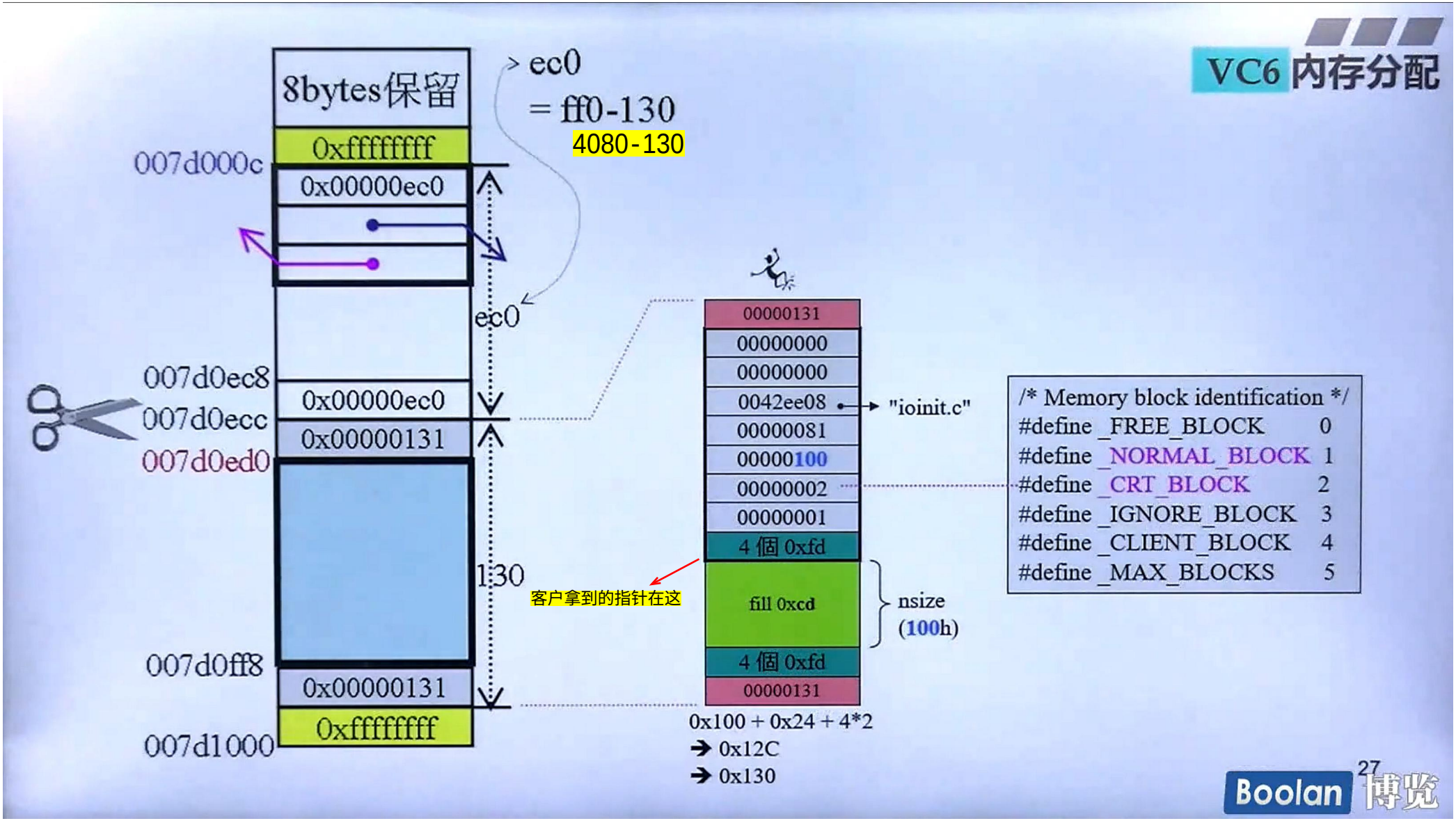


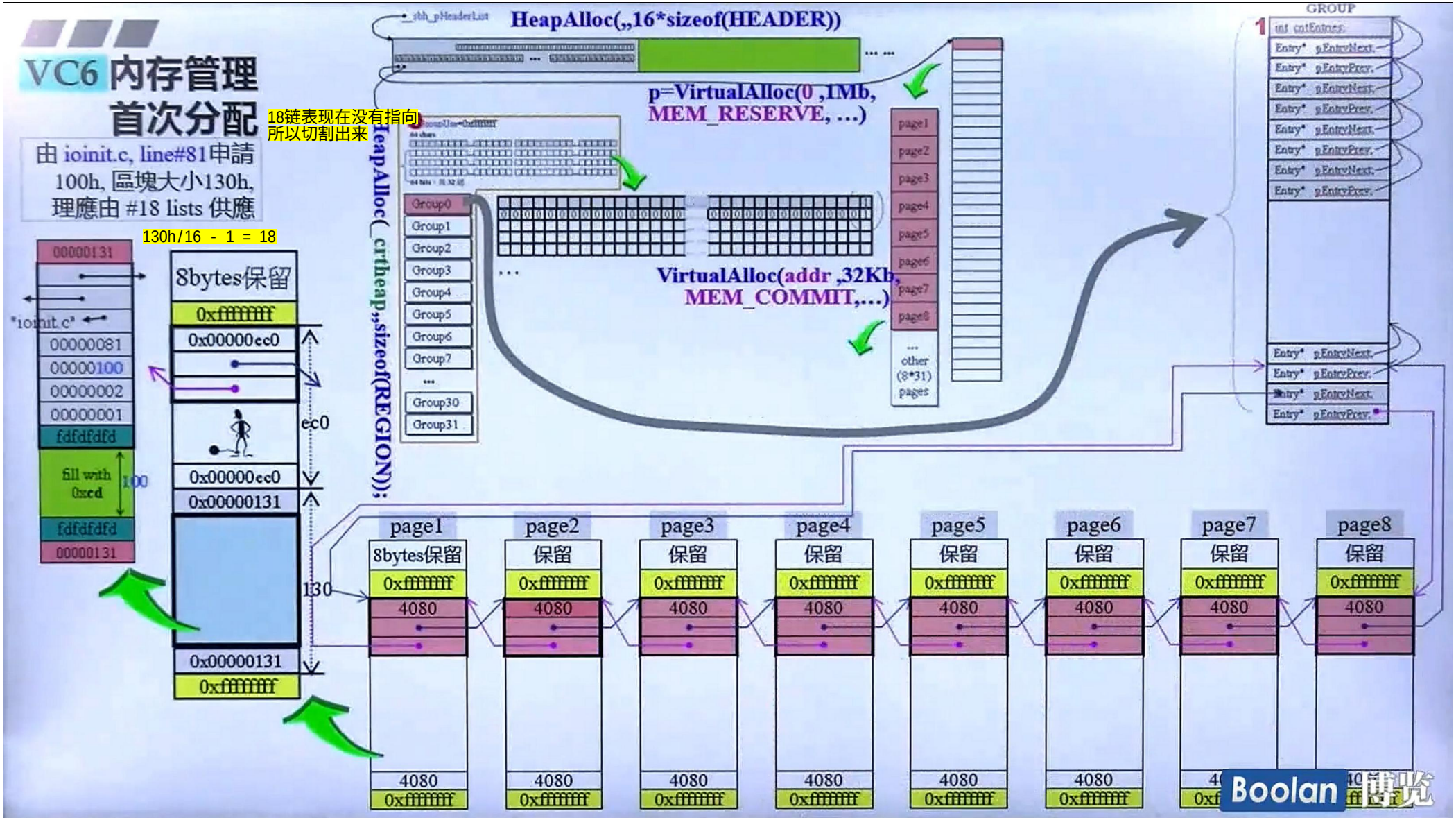
物理上是连续的
切成了8块
还要加上分隔
4096-8=4088 变成16的倍数 4080

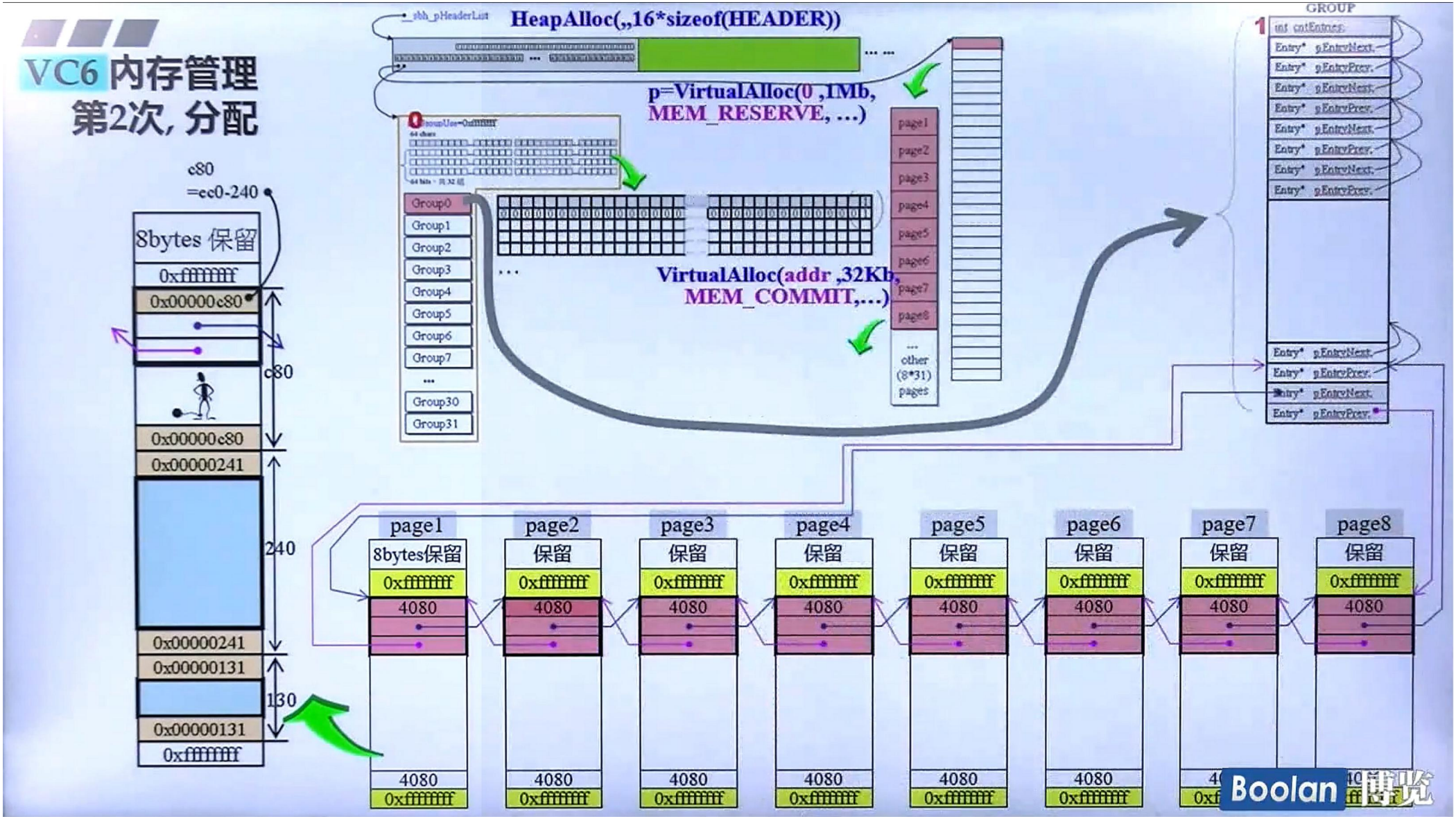
每个page可用的即为4080

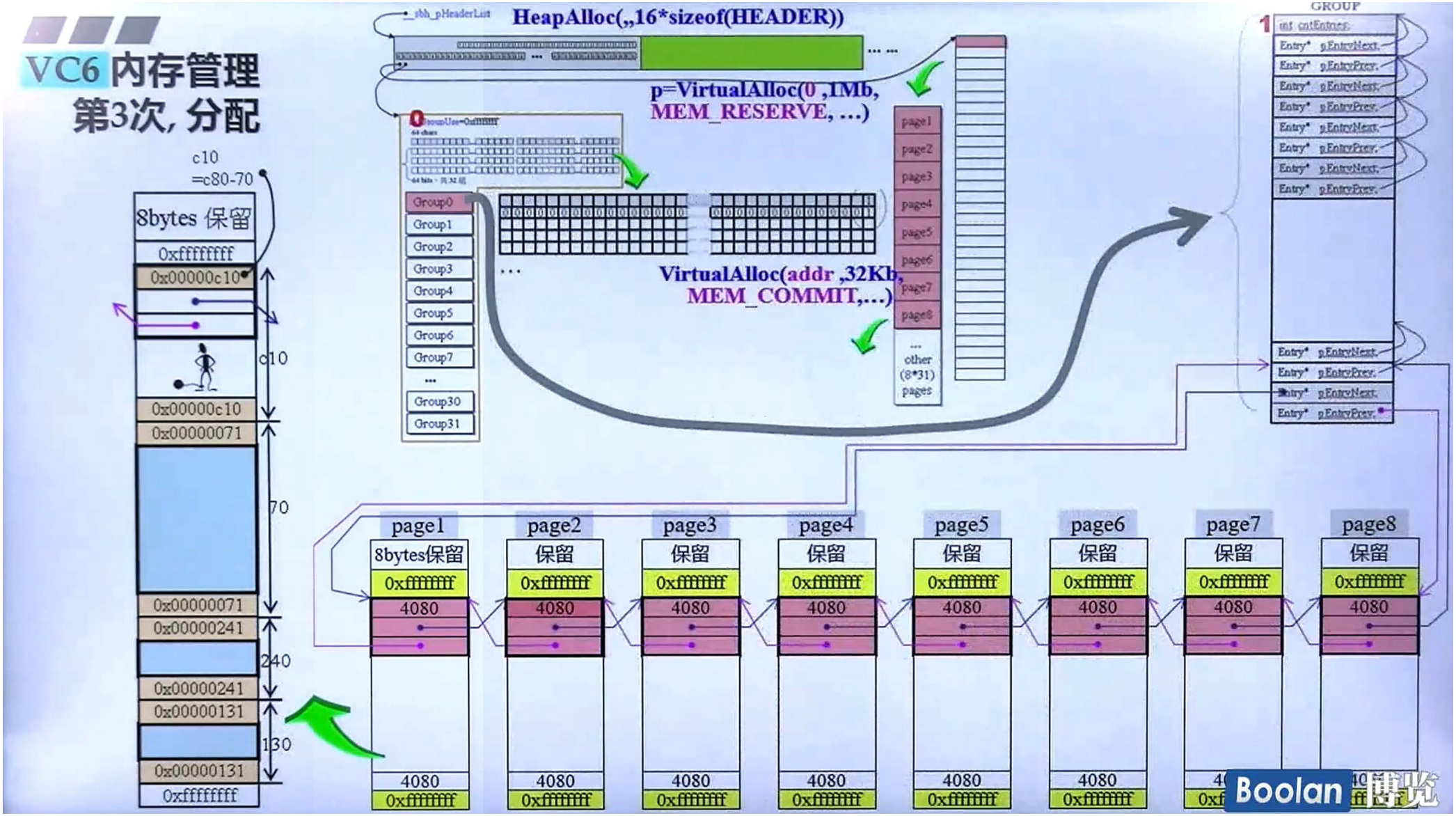
7. 内存分配精解3

怎么切

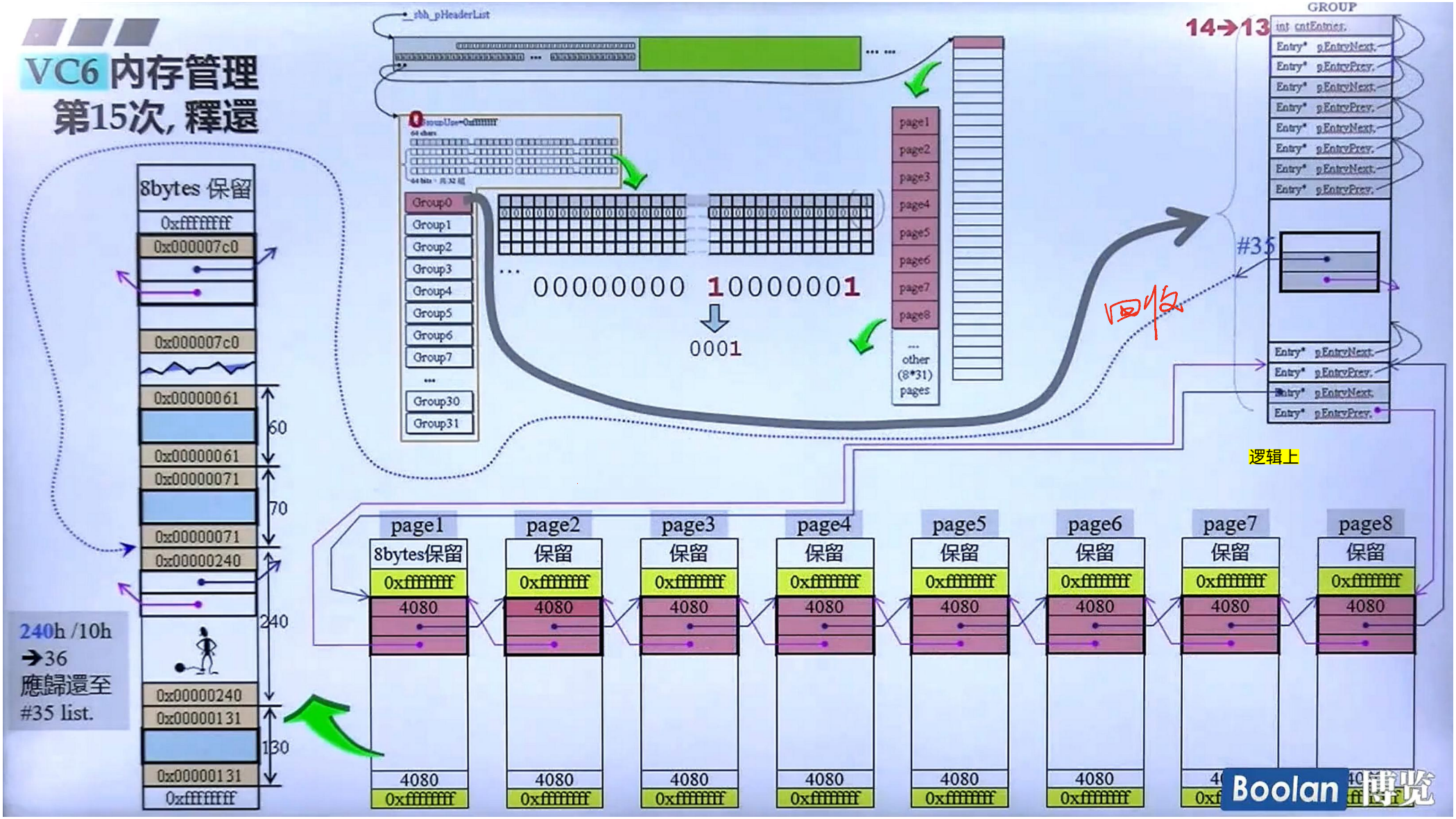

VC6 内存分配
8bytes保留
0xffffffff
007d000c
0x00000ec0
ec0
= ff0-130
4080-130
ec0
007d0ec8
0x00000ec0
007d0ecc
0x00000131
007d0ed0
130
007d0ff8
0x00000131
0xffffffff
007d1000
客户拿到的指针在这
00000131
00000000
00000000
0042ee08
"ioinit.c"
00000081
00000100
00000002
00000001
4 個 0xfd
fill 0xcd
nsize
(100h)
4 個 0xfd
00000131
0x100 + 0x24 + 4*2
➔ 0x12C
➔ 0x130
/* Memory block identification */
#define _FREE_BLOCK 0
#define _NORMAL_BLOCK 1
#define _CRT_BLOCK 2
#define _IGNORE_BLOCK 3
#define _CLIENT_BLOCK 4
#define _MAX_BLOCKS 5
27
Boolan 博览


VC6 内存管理
首次分配
由 ioinit.c, line#81申請
100h, 區塊大小130h,
理應由 #18 lists 供應
18链表现在没有指向
所以切割出来
130h/16 - 1 = 18
HeapAlloc(,,16*sizeof(HEADER))
p=VirtualAlloc(0 ,1Mb,
MEM_RESERVE, ...)
VirtualAlloc(addr ,32Kb,
MEM_COMMIT,...)
HeapAlloc(crtheap,,sizeof(REGION));
GROUP
8bytes保留
0xffffffff
0x00000ec0
0x00000131
ec0
130
100
fill with 0xcd
page1
page2
page3
page4
page5
page6
page7
page8
保留
4080
Group0
Group1
Group2
Group3
Group4
Group5
Group6
Group7
Group30
Group31
other (8*31) pages
Boolan 博览

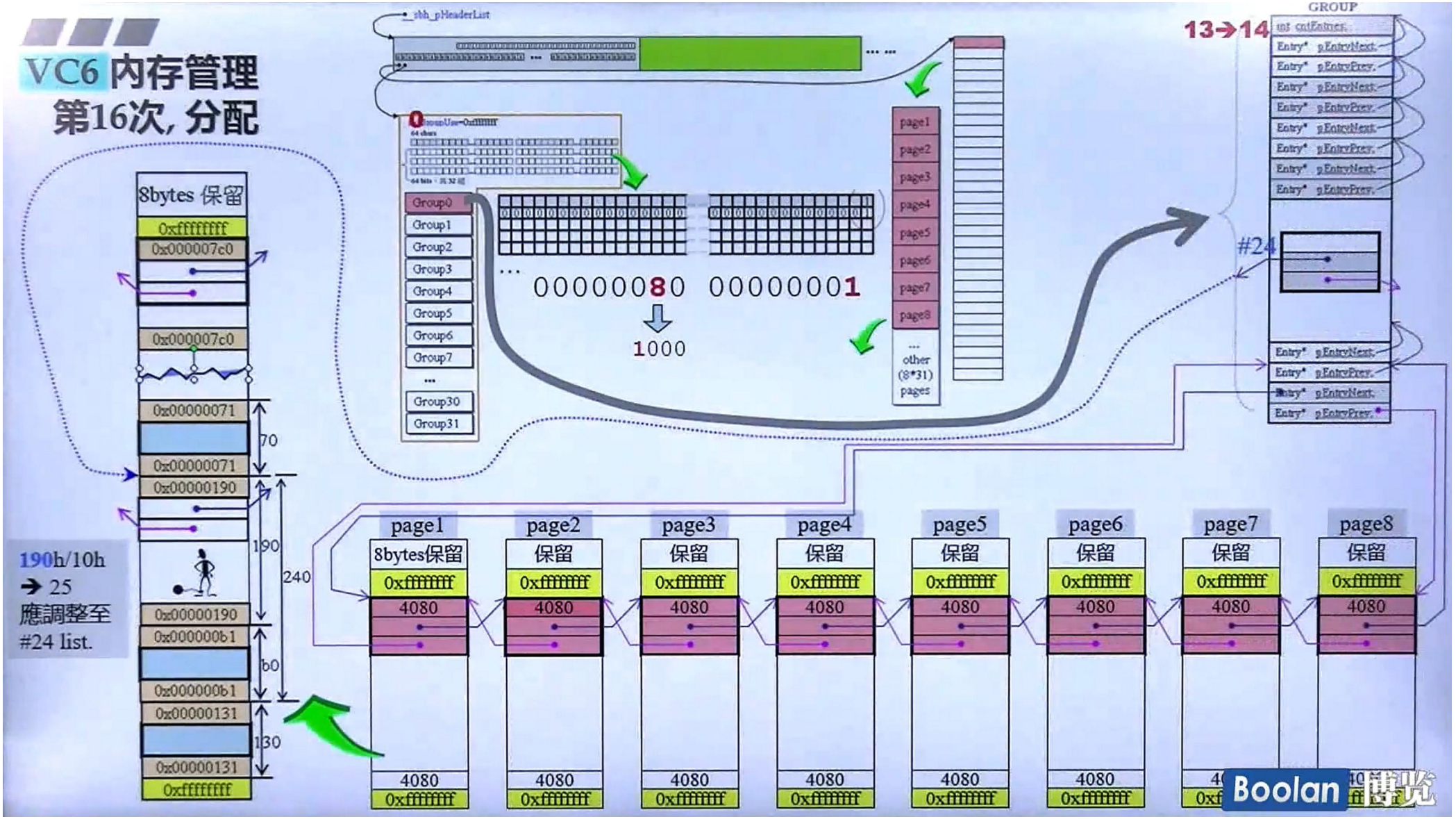

VC6 内存管理
第2次, 分配
c80
=ec0-240
8bytes 保留
0xffffffff
0x00000c80
c80
0x00000c80
0x00000241
240
0x00000241
0x00000131
130
0x00000131
0xffffffff
HeapAlloc(,,16*sizeof(HEADER))
p=VirtualAlloc(0 ,1Mb,
MEM_RESERVE, ...)
VirtualAlloc(addr ,32Kb,
MEM_COMMIT,...)
Group0
Group1
Group2
Group3
Group4
Group5
Group6
Group7
Group30
Group31
page1
page2
page3
page4
page5
page6
page7
page8
other
(8*31)
pages
GROUP
Entry* pEntryNext;
Entry* pEntryPrev;
page1
page2
page3
page4
page5
page6
page7
page8
8bytes保留
保留
0xffffffff
4080
Boolan 博览



-1 防止不同page合并

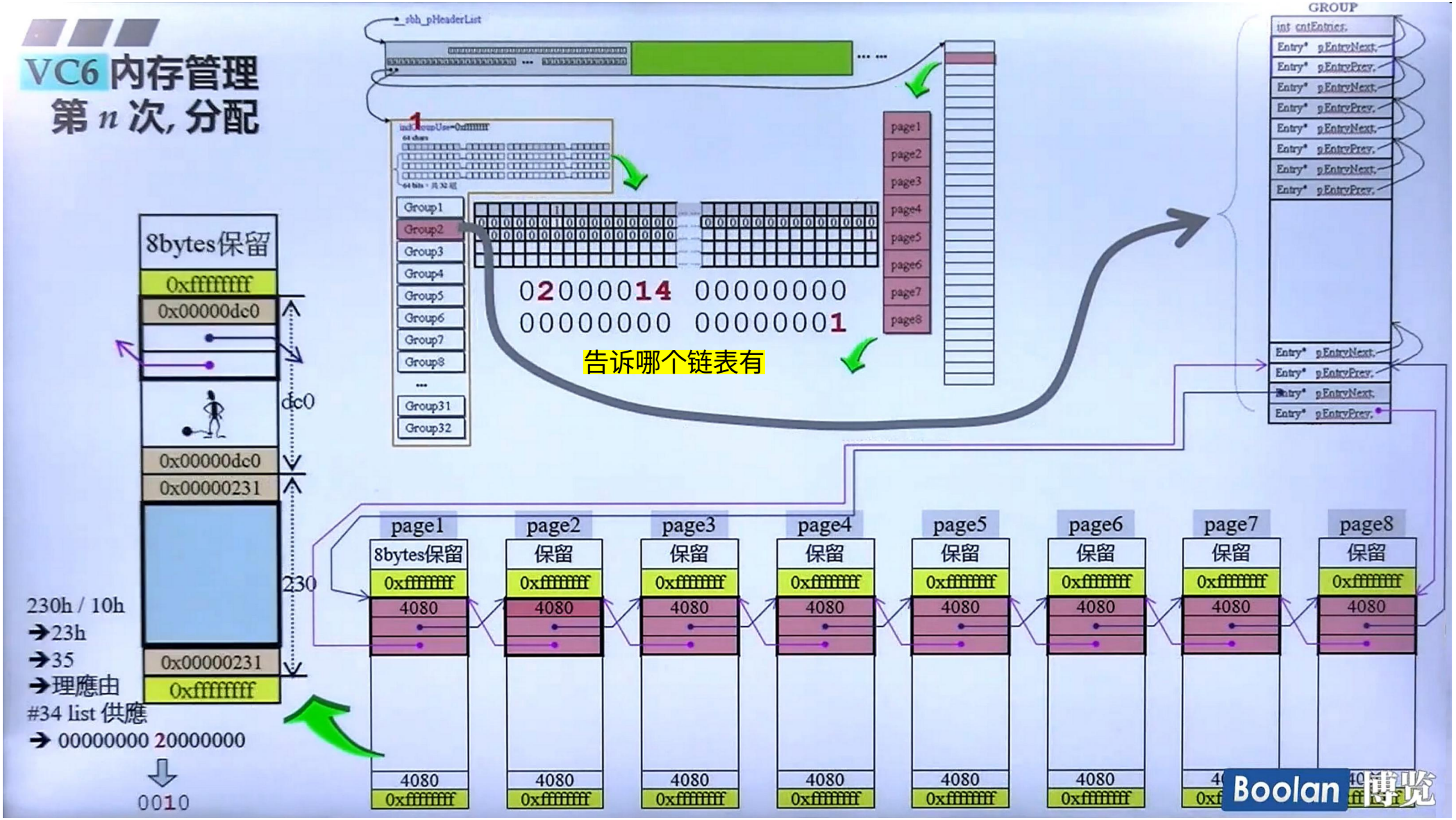


回收后下一次若还要相同大小的区块
则直接由链表指出
不用重新切割之类的操作



取最靠近需求的去切割
所以直接取上一个回收的240切

链表不断在调整


VC6 内存管理
第 n 次, 分配
8bytes保留
0xffffffff
0x00000dc0
dc0
0x00000dc0
0x00000231
230
0x00000231
0xffffffff
230h / 10h
➔23h
➔35
➔理應由
#34 list 供應
➔ 00000000 20000000
0010
Group1
Group2
Group3
Group4
Group5
Group6
Group7
Group8
Group31
Group32
page1
page2
page3
page4
page5
page6
page7
page8
02000014 00000000
00000000 00000001
告诉哪个链表有
GROUP
int cntEntries;
Entry* pEntryNext;
Entry* pEntryPrev;
保留
4080
Boolan 博览

8. 内存分配精解4

VC6 内存管理

區塊之合併

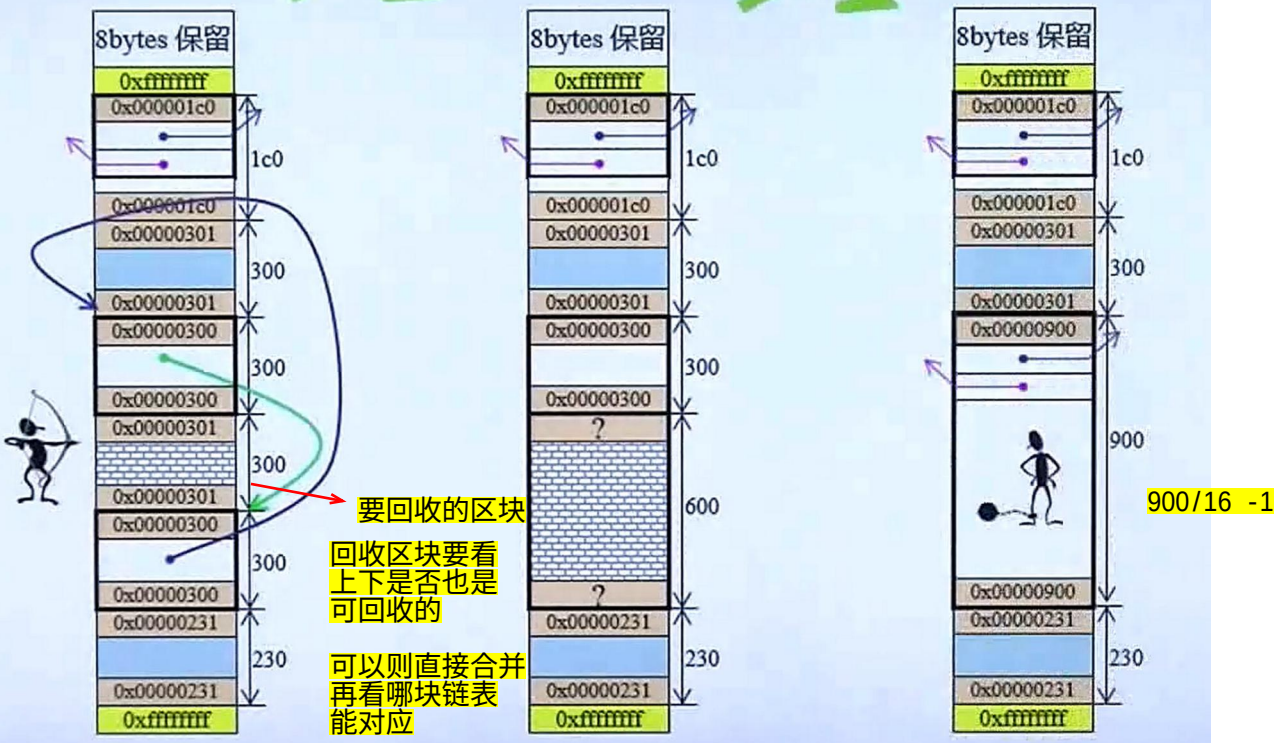
上 cookie
與
下 cookie
的作用

下方區塊若為 free，
合併之
（unlink 下方區塊）

上方區塊若為 free，
合併之
（unlink 上方區塊）

和上面合并

注意：腦袋裡要有兩張畫面 (兩個意象)，一是 memory block 處於線性空間 (物理意象)，一是 memory block 處於自由鏈表 (邏輯意象)。

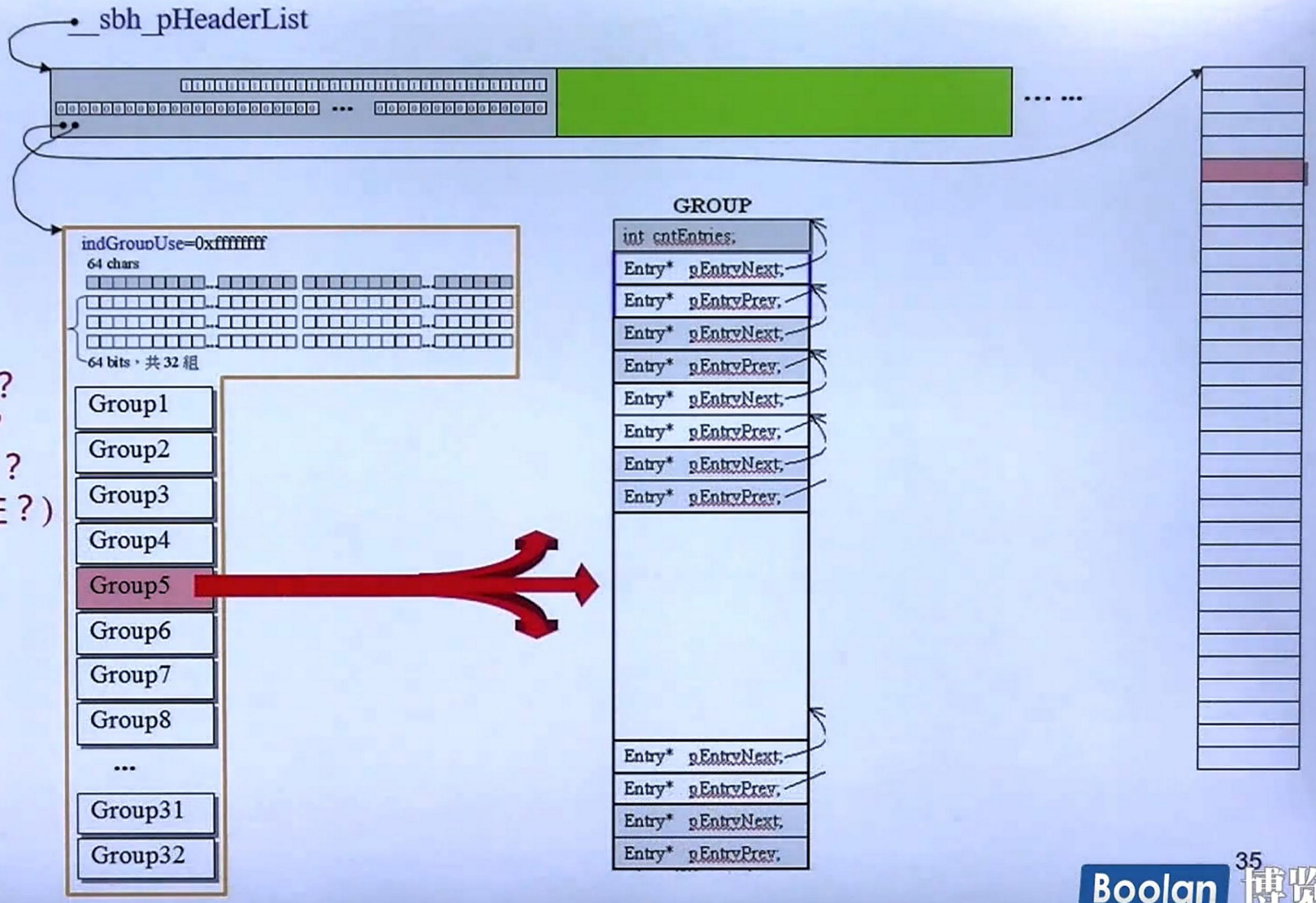


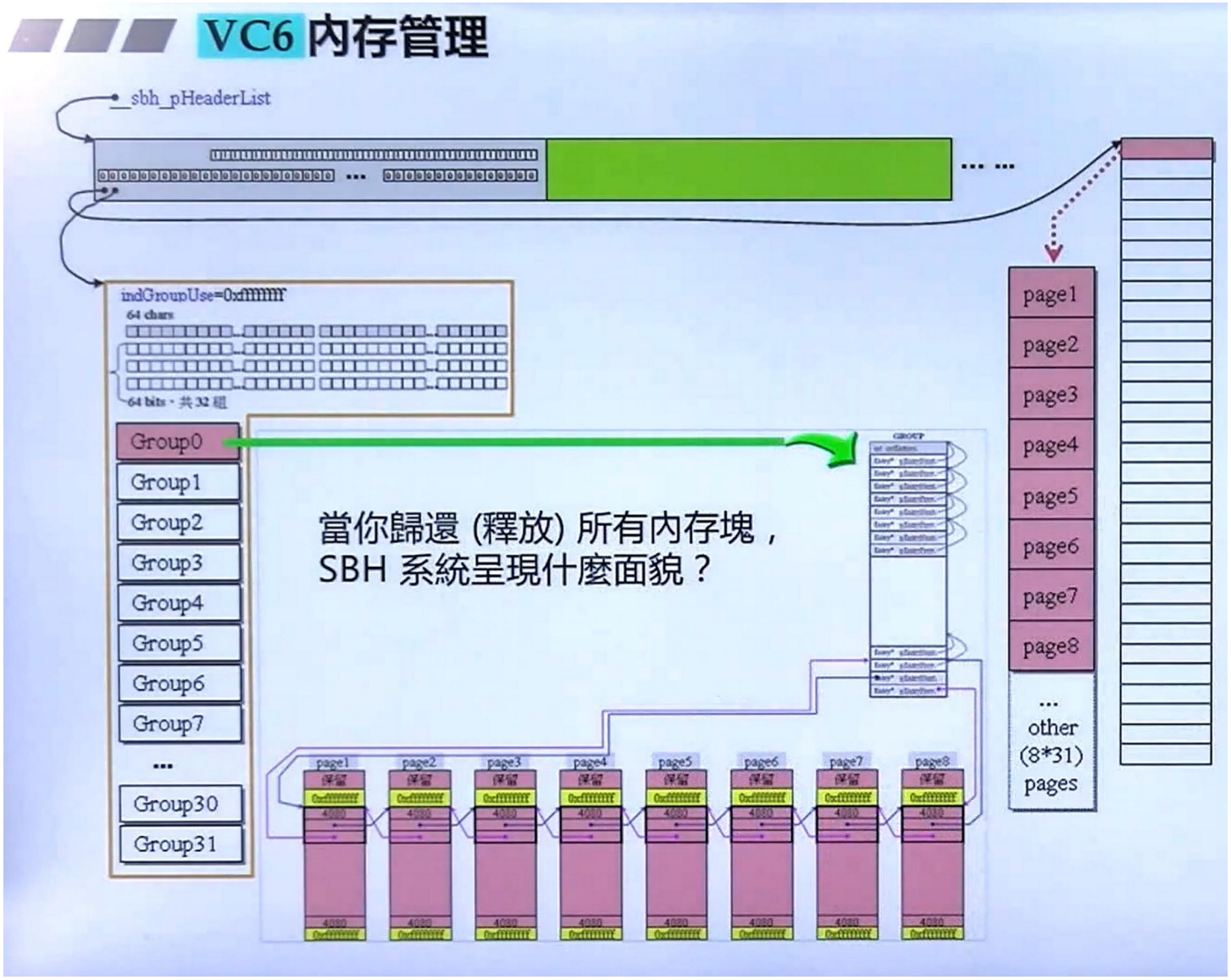
VC6 內存管理

free(p)

p 花落誰家？

落在哪個 Header 內？
落在哪個 Group 內？
落在哪個 free-list 內？
(被哪個 free-list 鏈住？)

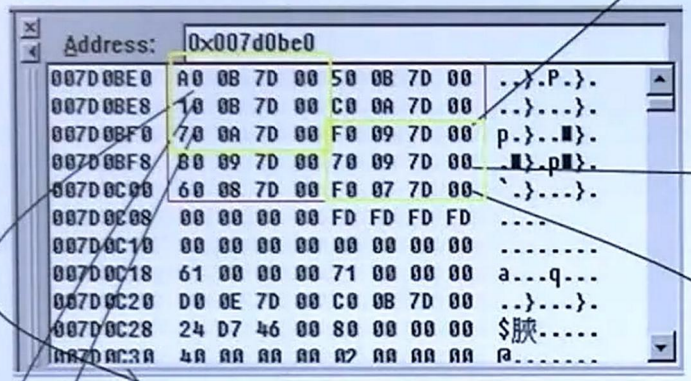




9. Main()生前所有内存分配之实例观察与解释

_environ 是 **pointer to pointer table**, table 中的每個 entry 都是 pointer to string which represents an environment variable.

_environ : 0x007d0be0



0

```
007D0BA0  54 4D 50 3D 43 3A 5C 57  TMP=C:\W
007D0BA8  49 4E 44 4F 57 53 5C 54  INDOWS\T
007D0BB0  45 4D 50 00 FD FD FD FD  EMP.
```

長度 20

1

```
007D0B50  54 45 4D 50 3D 43 3A 5C  TEMP=C:\
007D0B58  57 49 4E 44 4F 57 53 5C  WINDOWS\
007D0B60  54 45 4D 50 00 FD FD FD  TEMP
```

長度21

2

```
007D0B10  50 52 4F 4D 50 54 3D 24  PROMPT=$
007D0B18  70 24 67 00 FD FD FD FD  p$g.
```

長度12

3

```
007D0AC0  77 69 6E 62 6F 6F 74 64  winbootd
007D0AC8  69 72 3D 43 3A 5C 57 49  ir=C:\WI
007D0AD0  4E 44 4F 57 53 00 FD FD  NDOWS.
007D0AD8  FD FD 00 00 00 00 00 00  ....
```

長度22

4

```
007D0A70  43 4F 4D 53 50 45 43 3D  COMSPEC=
007D0A78  43 3A 5C 57 49 4E 44 4F  C:\WINDO
007D0A80  57 53 5C 43 4F 4D 4D 41  WS\COMMA
007D0A88  4E 44 2E 43 4F 4D 00 FD  ND.COM.
007D0A90  FD FD FD 00 00 00 00 00  .....
```

長度31

5

```
007D09F0  50 41 54 48 3D 43 3A 5C  PATH=C:\
007D09F8  57 49 4E 44 4F 57 53 3B  WINDOWS;
007D0A00  43 3A 5C 57 49 4E 44 4F  C:\WINDO
007D0A08  57 53 5C 43 4F 4D 4D 41  WS\COMMA
007D0A10  4E 44 3B 45 3A 5C 55 54  ND;E:\UT
007D0A18  49 4C 49 54 59 3B 43 3A  ILITY;C:
007D0A20  5C 50 52 4F 47 52 41 7E  \PROGRA~
007D0A28  31 5C 43 4F 4D 4D 4F 4E  1\COMMON
007D0A30  7E 31 5C 4D 55 56 45 45  ~1\MUVEE
007D0A38  54 7E 31 5C 30 33 30 36  T~1\0306
007D0A40  32 35 00 FD FD FD FD 00  25.
```

長度83

6

```
007D09B0  43 4D 44 4C 49 4E 45 3D  CMDLINE=
007D09B8  57 49 4E 00 FD FD FD FD  WIN.
```

長度12

7

```
007D0970  77 69 6E 64 69 72 3D 43  windir=C
007D0978  3A 5C 57 49 4E 44 4F 57  :\WINDOW
007D0980  53 00 FD FD FD FD 00 00  S.
```

長度18

8

```
007D0860  5F 41 43 50 5F 50 41 54  _ACP_PAT
007D0868  48 3D 43 3A 5C 4D 53 44  H=C:\MSD
007D0870  45 56 5C 43 4F 4D 4D 4F  EV\COMMO
007D0878  4E 5C 4D 53 44 45 56 39  N\MSDEV9
007D0880  38 5C 42 49 4E 3B 43 3A  8\BIN;C:
007D0888  5C 6D 73 64 65 76 5C 56  \msdev\V
007D0890  43 39 38 5C 42 49 4E 3B  C98\BIN;
007D0898  43 3A 5C 4D 53 44 45 56  C:\MSDEV
007D08A0  5C 43 4F 4D 4D 4F 4E 5C  \COMMON\
007D08A8  54 4F 4F 4C 53 3B 43 3A  TOOLS;C:
```

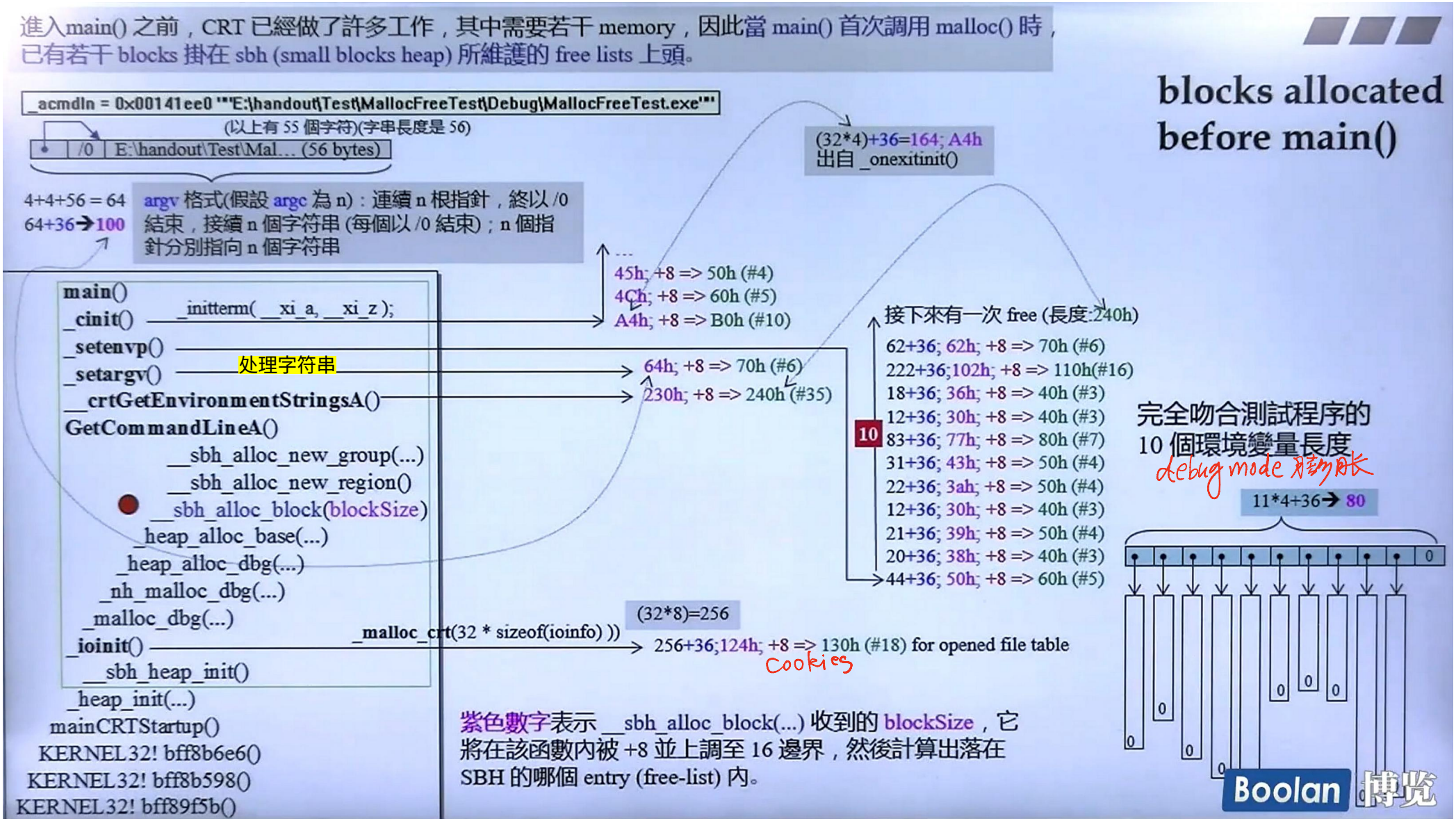
長度??

9

```
007D07F0  5F 41 43 50 5F 4C 49 42  _ACP_LIB
007D07F8  3D 43 3A 5C 6D 73 64 65  =C:\msde
007D0800  76 5C 56 43 39 38 5C 4C  v\VC98\L
007D0808  49 42 3B 43 3A 5C 6D 73  IB;C:\ms
007D0810  64 65 76 5C 56 43 39 38  dev\VC98
007D0818  5C 4D 46 43 5C 4C 49 42  \MFC\LIB
007D0820  3B 63 3A 5C 62 6F 6F 73  ;c:\boos
007D0828  74 5C 6C 69 62 00 FD FD  t\lib.
007D0830  FD FD 00 00 00 00 00 00  ......
```

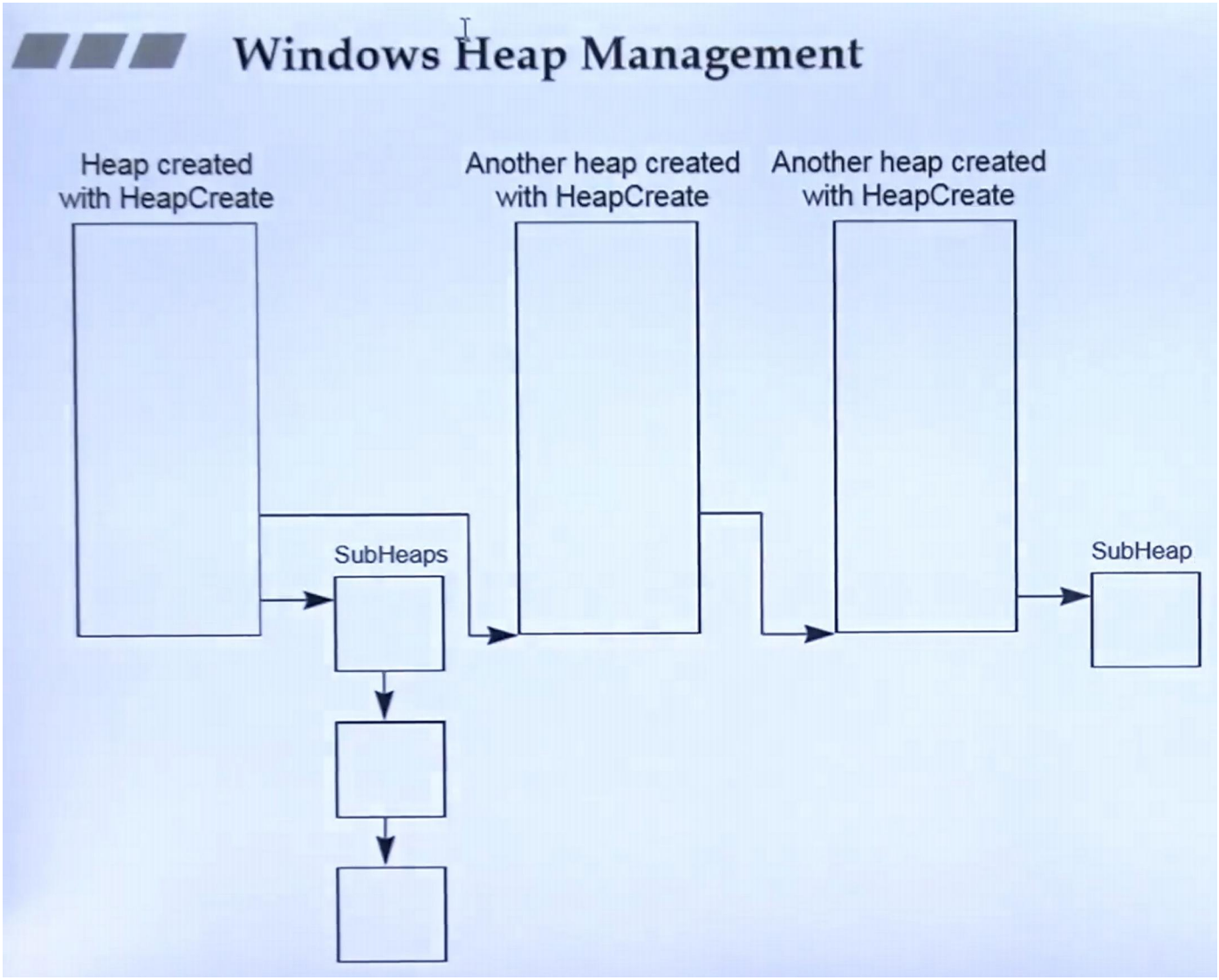
長度62

blocks allocated before main()

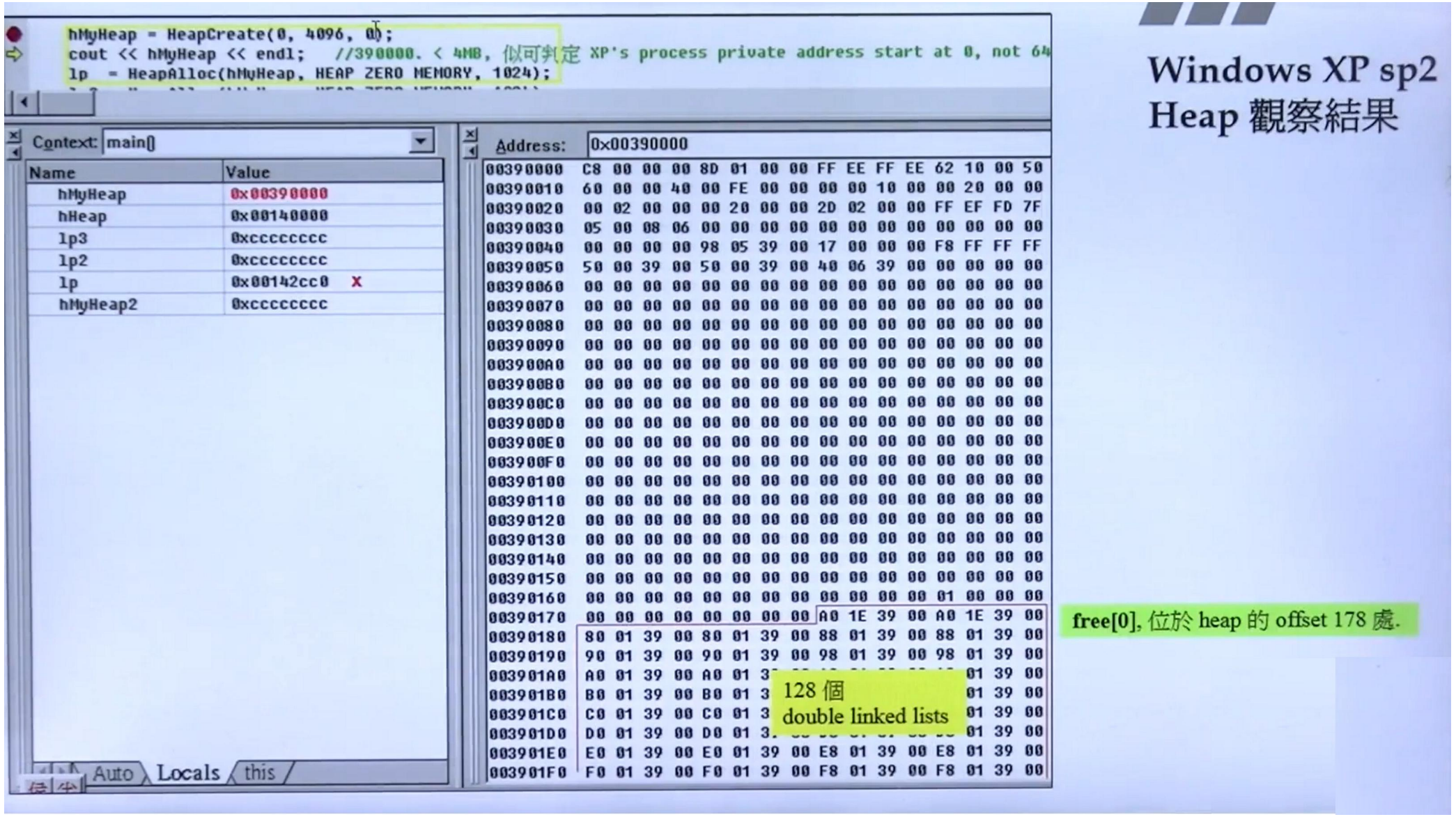


10. HeapAlloc()角色与影响

Windows Heap Management

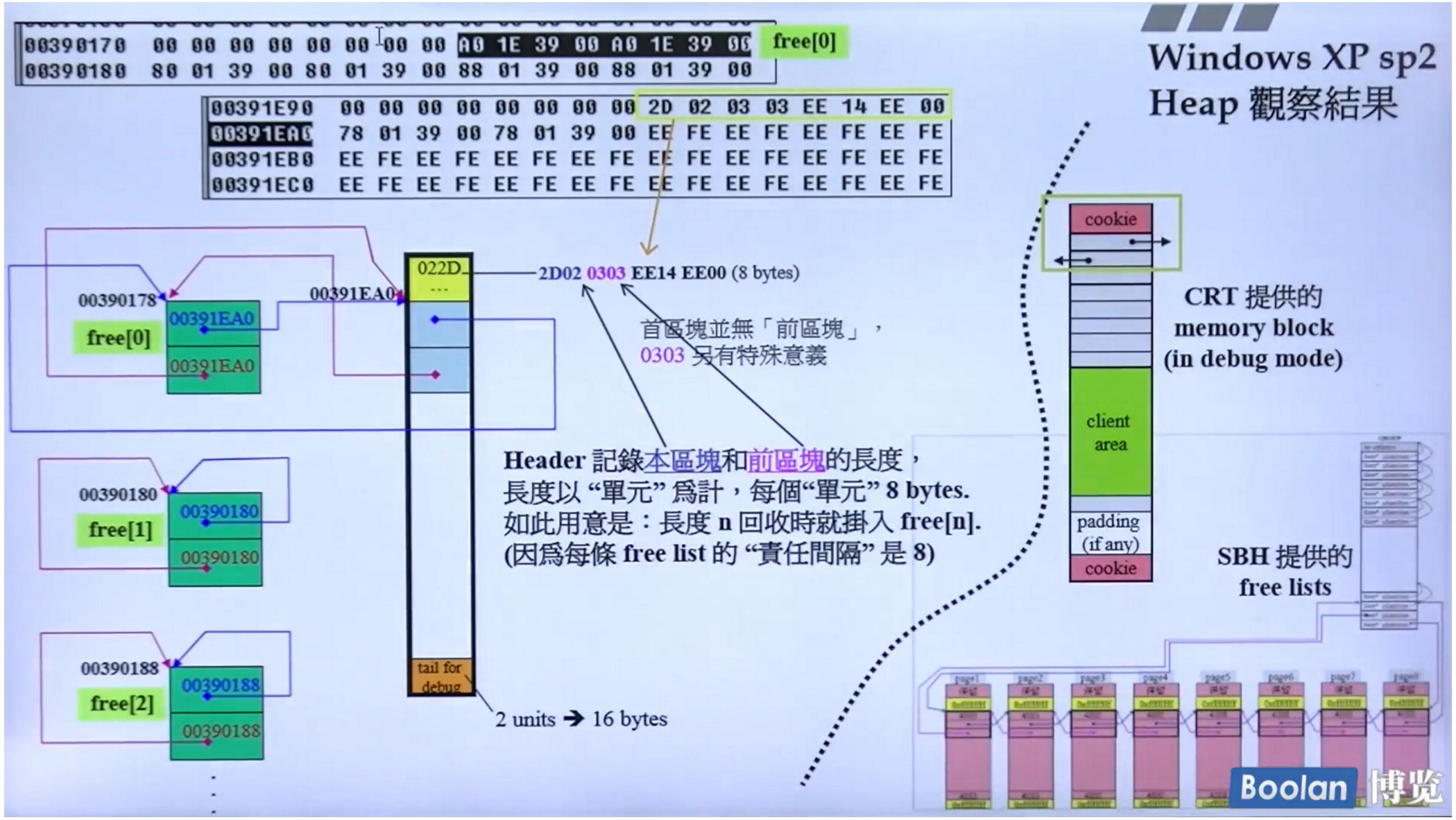


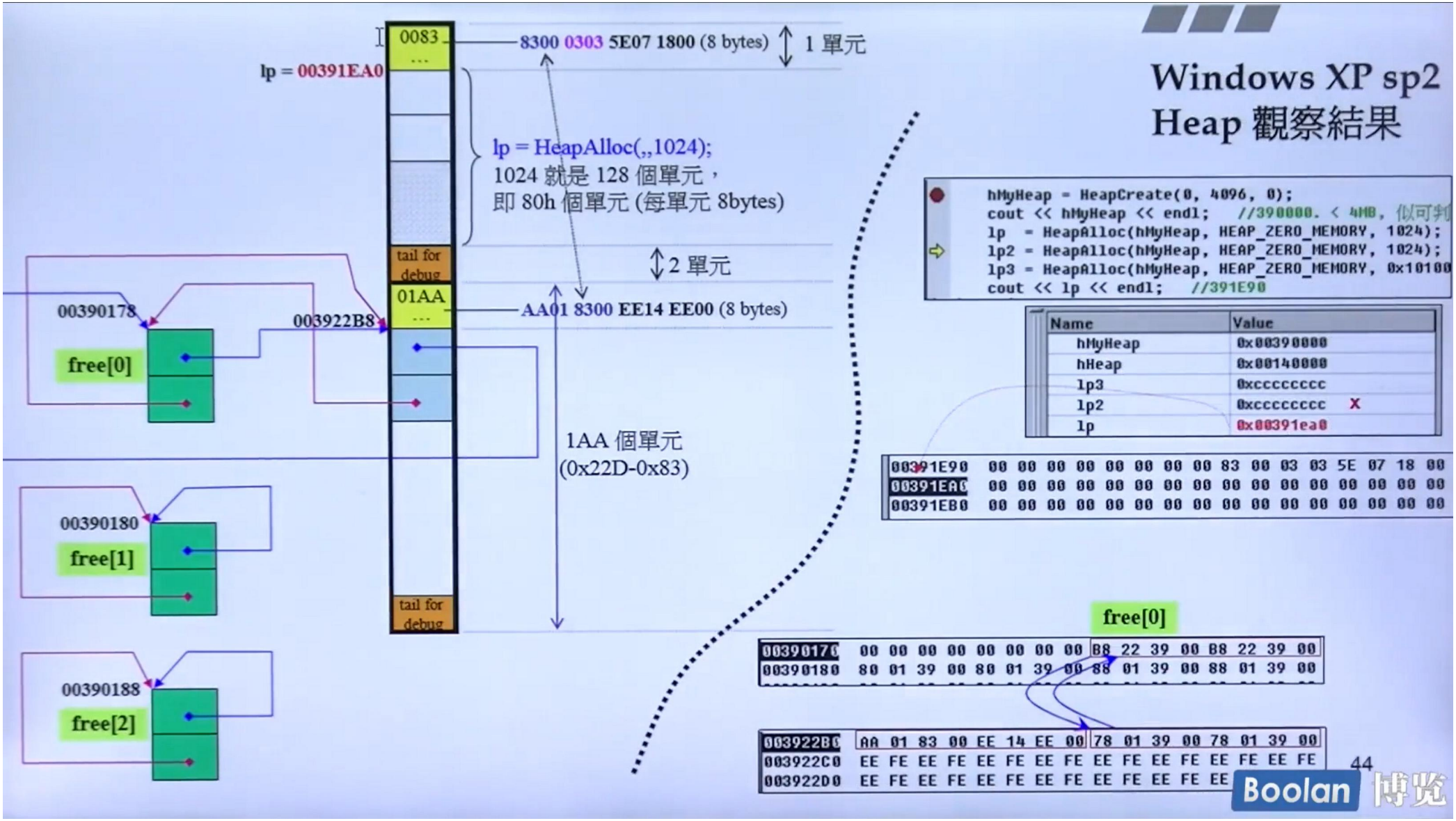
Windows XP sp2 Heap 觀察結果



free[0], 位於 heap 的 offset 178 處.


free[0], 位於 heap 的 offset 178 處.
Windows XP sp2
Heap 觀察結果
size: 022D
free[127]
128 個 free lists，
也就是 128 對指針，
0x390178 + 128 * 8
= 0x390578
42
Boolan 博


00390170 00 00 00 00 00 00 00 00 A0 1E 39 00 A0 1E 39 00
00390180 80 01 39 00 80 01 39 00 88 01 39 00 88 01 39 00
free[0]
00391E90 00 00 00 00 00 00 00 00 2D 02 03 03 EE 14 EE 00
00391EA0 78 01 39 00 78 01 39 00 EE FE EE FE EE FE EE FE
00391EB0 EE FE EE FE EE FE EE FE EE FE EE FE EE FE EE FE
00391EC0 EE FE EE FE EE FE EE FE EE FE EE FE EE FE EE FE
Windows XP sp2
Heap 觀察結果
00390178
free[0]
00391EA0
00391EA0
00391EA0
022D
...
2D02 0303 EE14 EE00 (8 bytes)
首區塊並無「前區塊」，
0303 只有特殊意義
00390180
free[1]
00390180
00390180
00390188
free[2]
00390188
00390188
tail for debug
2 units → 16 bytes
Header 記錄本區塊和前區塊的長度，
長度以 “單元” 爲計，每個“單元” 8 bytes.
如此用意是：長度 n 回收時就掛入 free[n].
(因爲每條 free list 的 “責任間隔” 是 8)
cookie
client area
padding (if any)
cookie
CRT 提供的
memory block
(in debug mode)
SBH 提供的
free lists
Boolan 博览



11. Io_init() - startup 的第二项大工程

_io_init()

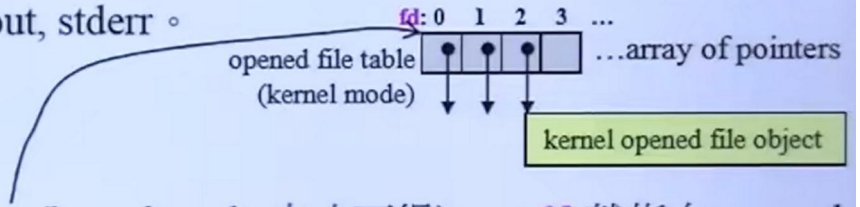
I/O 泛指程序和外界的各種交互作用，包括 file, pipe, network, console, semaphore 等等，或泛指能被 OS 理解爲 file 的任何事務 — file 在此是一個廣義概念。

```
FILE* fp = fopen("xxx.dat", "wb");
```

```
struct _iobuf { … };
typedef struct _iobuf FILE;
```

C 通過一個 pointer to **FILE** 來進行 file 操作。在 OS 層面，Linux 對應於 **FILE** 的是 File Descriptor (**fd**)，Windows 對應的則是 **file handle**. 二者都用來映射 kernel file object。

fd 具體是個 index of **opened file table** —process 個別擁有的這個 table 是個 array of pointers，每個 pointer 指向一個 kernel opened object。當 client 開啓一個 file，OS 會建立一個 (kernel) **opened file object** 並找到上述 table 中的一個 idle entry 指向之，然後以該 entry 的 index 做爲 **fd**。此 table 位於 kernel mode，因此 client 即使擁有 **fd** 亦無法獲得 table address. Linux 的 **fd** 0,1,2 分別代表 stdin, stdout, stderr。



C 的 **FILE** 與 Linux 的 **fd** 必有一對一關係。只要有 table address p (kernel mode 中才可得)，p+**fd** 就指向 opened file table 的某個 entry，從而可得 kernel file object。

Windows 的 **file handle** 和 Linux 的 **fd** 大同小異，但 **handle** 不是 index，而是 index 經某種變換後的結果。

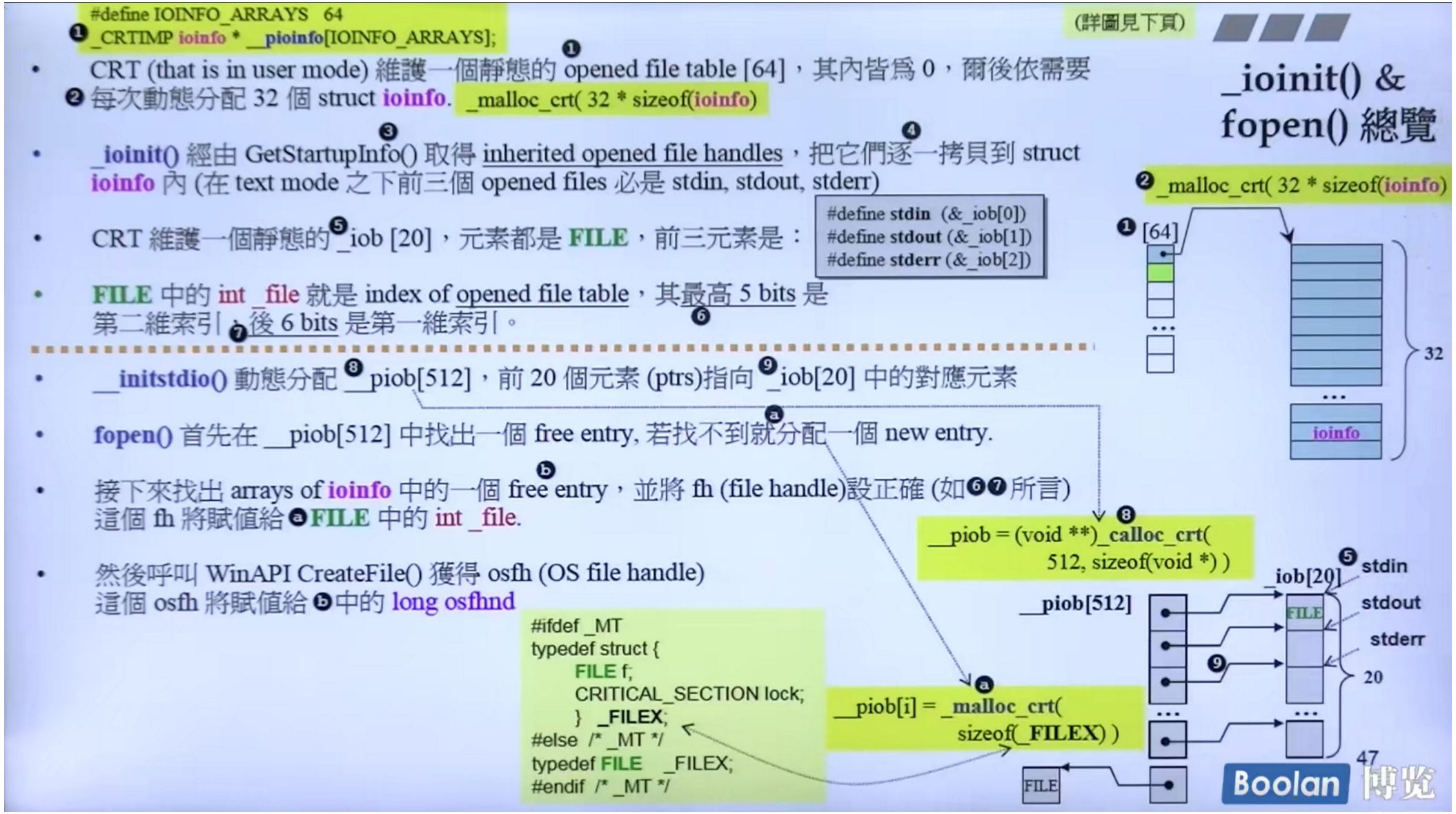
I/O initialization 就是要在 **client space** 中建立起 stdin, stdout, stderr 及其對應的 **FILE**s，使程式進入 main() 之後立即可用 printf(), scanf() 等函式。

_io_init()

```
    ExitProcess(code)
    _initterm(,,)   //do terminators
      __endstdio(void)
    _initterm(,,)   //do pre-terminators
  doexit(code, 0, 0)
exit(code)
main()
  _initterm(,,) //do C++ initializations
    __initstdio(void)
  _initterm(,,) //do initializations
_cinit()
_setenvp()
_setargv()
__crtGetEnvironmentStringsA()
GetCommandLineA()
            __sbh_alloc_new_group(...)
            __sbh_alloc_new_region()
          __sbh_alloc_block(...)
        _heap_alloc_base(...) //size:292
      _heap_alloc_dbg(...)
    _nh_malloc_dbg(...)
  _malloc_dbg(...) //size:256
_ioinit()
  __sbh_heap_init()
_heap_init(...)
mainCRTStartup()
```

ioinit.c

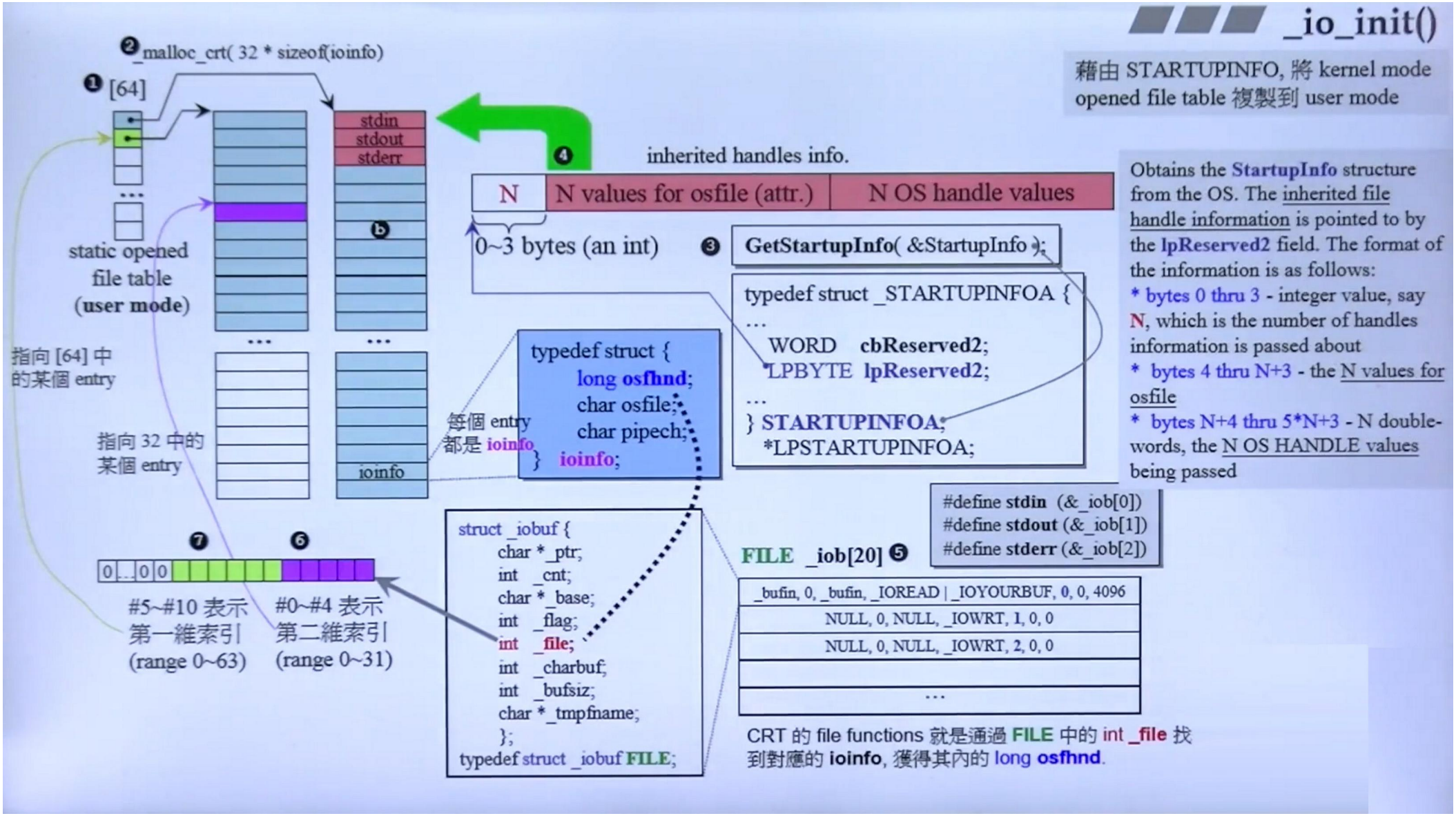
```
/***
*Purpose:
*       Allocates and initializes initial array(s) of ioinfo structs. Then,
*       obtains and processes information on inherited file handles from the
*       parent process (e.g., cmd.exe).
*
*       Obtains the StartupInfo structure from the OS. The inherited file
*       handle information is pointed to by the lpReserved2 field. The format
*       of the information is as follows:
*           bytes 0 thru 3 - integer value, say N, which is the
*                            number of handles information is passed about
*           bytes 4 thru N+3   - the N values for osfile
*           bytes N+4 thru 5*N+3  - N double-words, the N OS HANDLE values
*                            being passed
*       Next, osfhnd and osfile for the first three ioinfo structs,
*       corrsponding to handles 0, 1 and 2, are initialized as follows:
*           If the value in osfhnd is INVALID_HANDLE_VALUE, then try to
*           obtain a HANDLE by calling GetStdHandle, and call GetFileType to
*           help set osfile. Otherwise, assume _osfhndl and _osfile are
*           valid, but force it to text mode (standard input/output/error
*           are to always start out in text mode).
*       Notes:
*           1. In general, not all of the passed info from the parent process
*              will describe open handles! If, for example, only C handle 1
*              (STDOUT) and C handle 6 are open in the parent, info for C
*              handles 0 thru 6 is passed to the the child.
*           2. Care is taken not to 'overflow' the arrays of ioinfo structs.
*           3. See exec\dospawn.c for the encoding of the file handle info
*              to be passed to a child process.
*Entry:
*       No parameters: reads the STARTUPINFO structure.
*Exit:
*       No return value.
*Exceptions:
*******************************************************************************/
```



12. C_init() startup的第三大工程

_io_init()

藉由 STARTUPINFO, 將 kernel mode opened file table 複製到 user mode

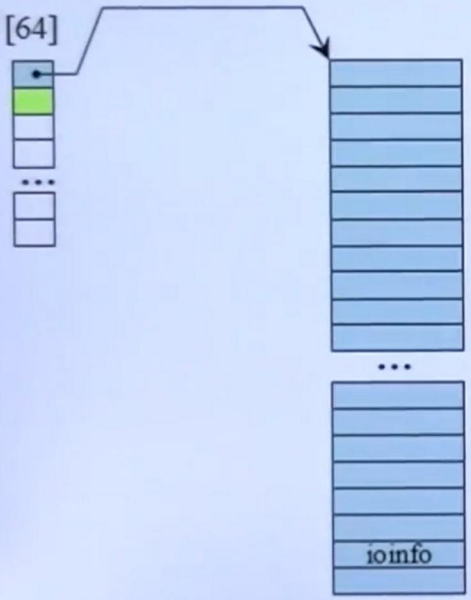


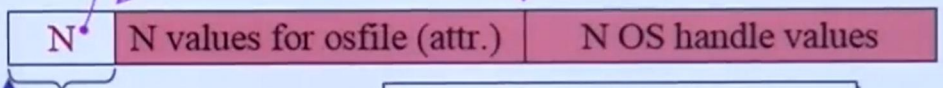
```
#0080    /* * Process inherited file handle information, if any  */
#0081    GetStartupInfo( &StartupInfo );
#0082    if ( (StartupInfo.cbReserved2 != 0) &&
#0083         (StartupInfo.lpReserved2 != NULL) )
#0084    {
#0085        /* * Get the number of handles inherited. */
#0086        cfi_len = *(UNALIGNED int *)(StartupInfo.lpReserved2);

#0087        /* * Set pointers to the start of the passed file info and OS
#0088         * HANDLE values.  */
#0089        posfile = (char *)(StartupInfo.lpReserved2) + sizeof( int );
#0090        posfhnd = (UNALIGNED long *)(posfile + cfi_len);
```

```
#define IOINFO_ARRAYS   64
_CRTIMP ioinfo * __pioinfo[IOINFO_ARRAYS];
```

❷ _malloc_crt(32 * sizeof(ioinfo)

❶ [64]





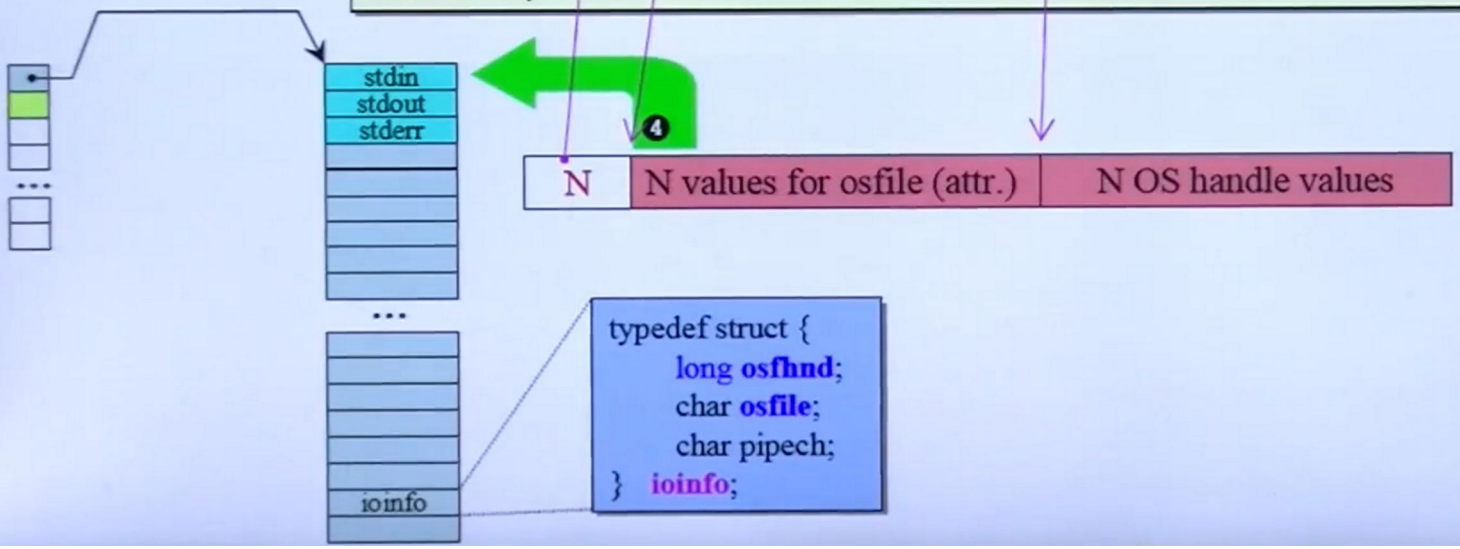
0~3 bytes (an int)

❸ **GetStartupInfo**(&StartupInfo);

```
typedef struct _STARTUPINFOA {
…
    WORD    cbReserved2;
    LPBYTE  lpReserved2;
…
} STARTUPINFOA;
  *LPSTARTUPINFOA;
```

_io_init()

```
#0119        for ( fh = 0 ; fh < cfi_len ; fh++, posfile++, posfhnd++ ) {
#0120          /*
                ...
#0127           */
#0128          if ( (*posfhnd != (long)INVALID_HANDLE_VALUE) &&
#0129              (*posfile & FOPEN) &&
#0130              ((*posfile & FPIPE) ||
#0131               (GetFileType( (HANDLE)*posfhnd ) != FILE_TYPE_UNKNOWN)) )
#0132          {
#0133            pio = _pioinfo( fh );
#0134            pio->osfhnd = *posfhnd;
#0135            pio->osfile = *posfile;
#0136          }
#0137        }
#0138      }
```



file test

```
#include <stdio.h>
#include <iostream>
using namespace std;

int main()
{
  cout << stdin << endl;          //00414538
  cout << stdin->_file << endl;   //0
  cout << stdout << endl;         //00414558
  cout << stdout->_file << endl;  //1
  cout << stderr << endl;         //00414578
  cout << stderr->_file << endl;  //2

  FILE* fp = fopen("xxx.dat", "wb");
  if (fp == NULL) return -1;
  cout << fp << endl;             //00414598
  cout << fp->_file << endl;      //3

  fwrite("abcde", 5, 1, fp);
  fclose(fp);
  cout << fp << endl;             //00414598
  cout << fp->_file << endl;      //3
```

fclose() 會歸還 file handle!

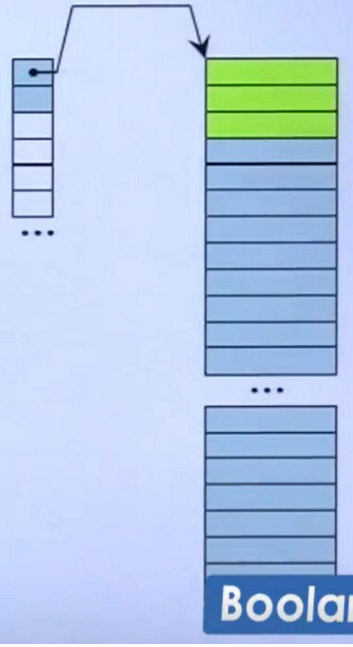
```
#define stdin  (&_iob[0])
#define stdout (&_iob[1])
#define stderr (&_iob[2])
```

sizeof(FILE): 0x20
所以 stdin, stdout, stderr (都是指針) 相差 0x20

```
  fp = fopen("yyy.dat", "wb");
  if (fp == NULL) return -1;
  cout << fp << endl;             //00414598
  cout << fp->_file << endl;      //3
```

```
  fwrite("fghij", 5, 1, fp);
  fclose(fp);
  cout << fp << endl;             //00414598
  cout << fp->_file << endl;      //3

  int i;
  FILE* f[60];
  char buffer[20];
  for (i=0; i< 60; ++i) {
     f[i] = fopen(_itoa(i,buffer,10), "wb");
     cout << f[i]->_file << endl;      //3 4 5 6 ... 62
  }
  return 0;
}
```

本例先檢驗 std I/O 的 handle 分別爲 0,1,2. 然後 open/close 二次。然後 open 60 次。故總共佔用 3+60 = 63 個 handles。



_cinit()

```
    ExitProcess(code)
    _initterm(,,)   //do terminators
      __endstdio(void)
    _initterm(,,)   //do pre-terminators
  doexit(code, 0, 0)
exit(code)
main()
  _initterm(,,) //do C++ initializations
    __initstdio(void)
  _initterm(,,) //do initializations
  _cinit()
_setenvp()
_setargv()
__crtGetEnvironmentStringsA()
GetCommandLineA()
            __sbh_alloc_new_group(...)
            __sbh_alloc_new_region()
          __sbh_alloc_block(...)
        _heap_alloc_base(...)
      _heap_alloc_dbg(...)
    _nh_malloc_dbg(...)
  _malloc_dbg(...)
_ioinit()
  __sbh_heap_init()
_heap_init(...)
mainCRTStartup()
```

```
/***
*_cinit - C initialization
*
*Purpose:
*       This routine performs the shared DOS and Windows initialization.
*       The following order of initialization must be preserved -
*
*       1.  Check for devices for file handles 0 - 2
*       2.  Integer divide interrupt vector setup
*       3.  General C initializer routines
*
*Entry:
*       No parameters: Called from __crtstart and assumes data
*       set up correctly there.
*
*Exit:
*       Initializes C runtime data.
*
*Exceptions:
*
*******************************************************/
```