

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

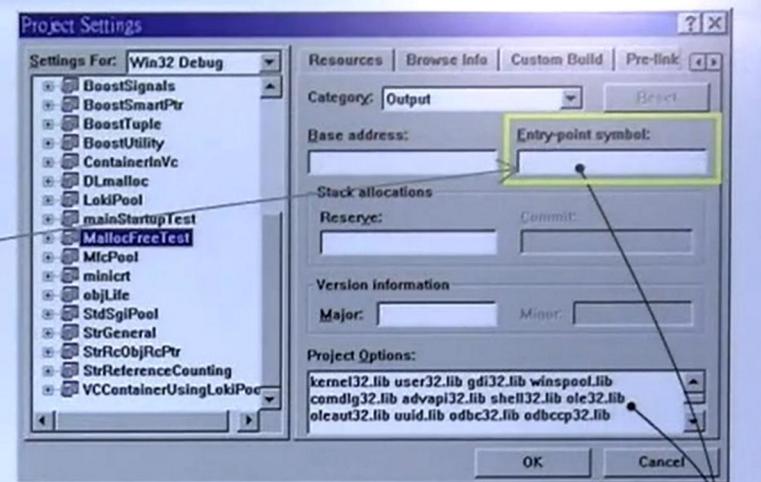
VC6, VC10

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.

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

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

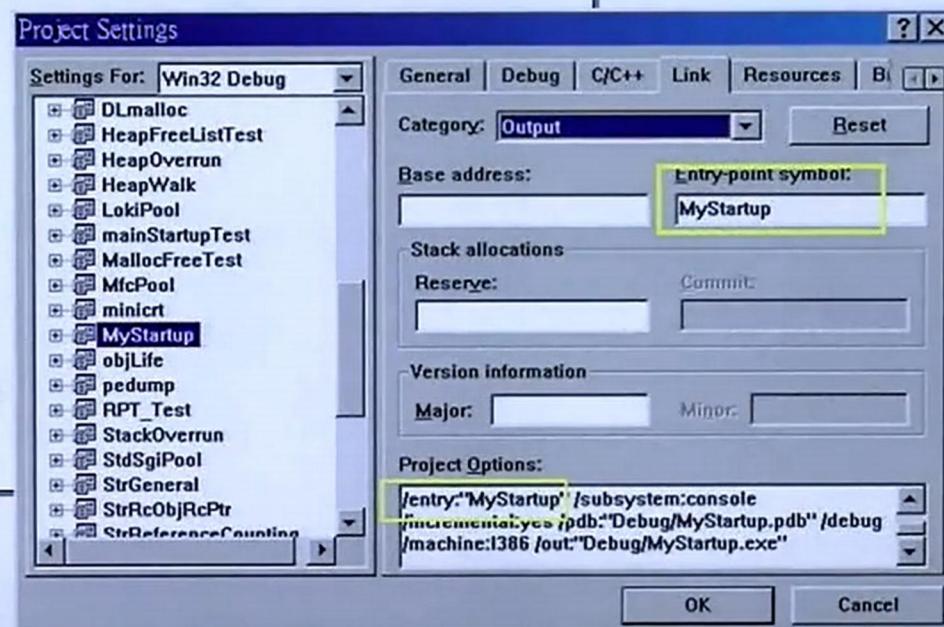
Startup code 的三個可能形式 (函數名稱)

## 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\MyStartup Code调用

main前调用 startup code



## 自定 Startup code

why doesn't a compiler give the address of main() as a starting point? That's because typical libc implementations want to do some initializations before really starting the program.

edit as an example, you can change the entry point like this:

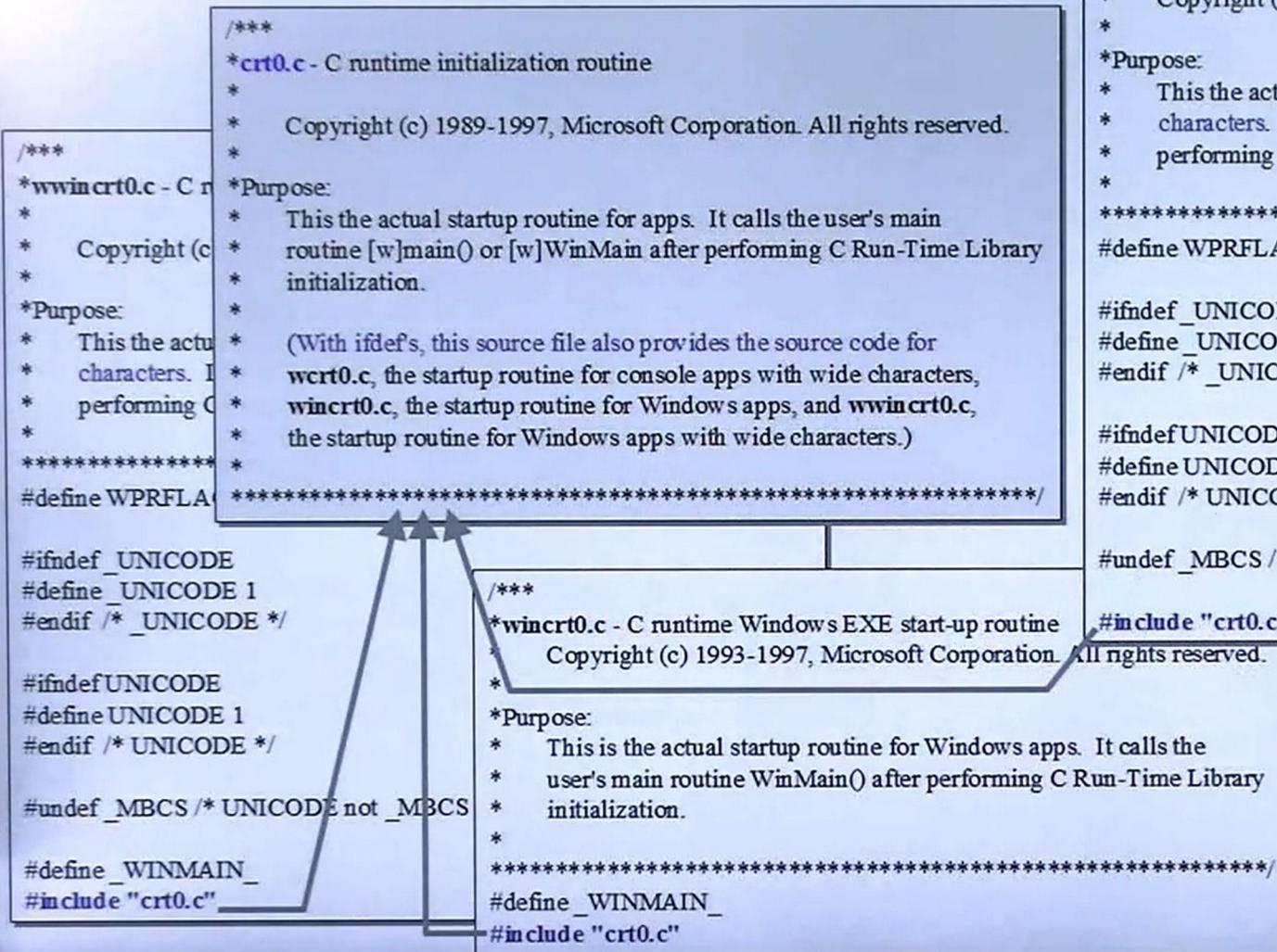
```
$ cat entrypoint.c
int blabla() { printf("Yes it works!\n"); exit(0); }
int main() { printf("not called\n"); }

$ gcc entrypoint.c -e blabla

$ ./a.out
Yes it works!
```

## 2. 默认的Startup code在哪儿

# Startup code 在哪兒



VC6的启动代码



## call stack

```
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() 启动函数
KERNEL32! bff8b6e6()
KERNEL32! bff8b598()
KERNEL32! bff89f5b()
```

从下往上

### 3. Startup code代码摘要

## mainCRTStartup()

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
14     heap初始化
```

VC6

```
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 { ⑩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         _initenv = _environ;
29         mainret = main(_argc, _argv, _environ);
30         ⑨ 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 }
```

# mainCRTStartup()

```
#ifdef _UNICODE
#define _tmain wmain
#define _tWinMain wWinMain
#define _tenviron _wenviron
#define _targv __wargv
#else /* _UNICODE */
#define _tmain main
#define _tWinMain WinMain
#define _tenviron _environ
#define _targv __argv
#endif /* _UNICODE */
```

3个版本 by

很多 #ifdef ,  
有的進入  
WinMain()  
有的進入  
main()

若非呼叫  
WinMain()  
就是呼叫  
main(...),  
又各有 w 版  
和 non-w 版

mainCRTStartup() 局部, in crt0.c

```
#ifdef _WINMAIN_
    StartupInfo.dwFlags = 0;
    GetStartupInfo( &StartupInfo );
#endif WPRFLAG
    lpszCommandLine = _wwincmdln();
    mainret = wWinMain(
#else /* WPRFLAG */
    lpszCommandLine = _wincmdln();
    mainret = WinMain(
#endif /* WPRFLAG */
    GetModuleHandleA(NULL),
    NULL,
    lpszCommandLine,
    StartupInfo.dwFlags &
        STARTF_USESHOWWINDOW
        ? StartupInfo.wShowWindow
        : SW_SHOWDEFAULT
    );
#endif /* _WINMAIN_ */
#endif WPRFLAG
    _winitenv = _wenviron;
    mainret = wmain( __argc, __wargv, _wenviron );
#else /* WPRFLAG */
    _initenv = _environ;
    mainret = main( __argc, __argv, _environ );
#endif /* WPRFLAG */
#endif /* _WINMAIN_ */
```

## 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, 蘆清、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)

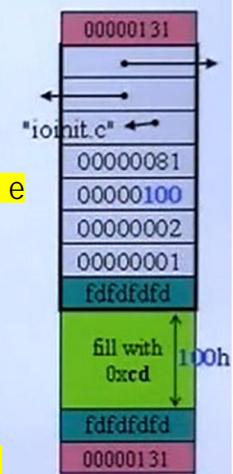
SBH: Small Block Heap  
何謂 small ?

內存塊  
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;      大于1K的让操作系统处理
size = (size + ...) & ~(...);
return HeapAlloc [crtheap], 0, size);
```



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

```

ExitProcess(code)
    _inititem(,) //do terminators
        _endstdio(void)
    _inititem(,) //do pre-terminators
doexit(code, 0, 0)
⑨exit(code)
⑧main()
    _inititem(,) //do C++ initializations
        _initstdio(void)
    _inititem(,) //do initializations
⑦_cinit() // do C data initialize
⑥_setenvp()
⑤_setargv()
④_crtGetEnvironmentStringsA()
③GetCommandLineA()
    __sbh_alloc_new_group(...)
    __sbh_alloc_new_region()
    sbh_alloc_block(...)
    heap_alloc_base(...) ← (highlighted)
    heap_alloc_dbg(...)
    __nh_malloc_dbg(...)
    __malloc_dbg(...)
②_ioinit() // initialize lowio
    __sbh_heap_init()
①_heap_init(...)
mainCRTStartup()
KERNEL32! bff8b6e60
KERNEL32! bff8b5980
KERNEL32! bff89f5b()

```

## VC10\_heap\_init()

所有内存管理  
机制其实仍然  
存在，只不过  
深埋到 O.S. 层  
去了

```
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()
    x [REDACTED] (...)  

    _nh_malloc_dbg(...)  

    _malloc_dbg(...)  

② _ioinit() // initialize lowio
x [REDACTED]()
① heap_init(...)
mainCRTStartup()
KERNEL32! bff8b6e60
KERNEL32! bff8b5980
KERNEL32! bff89f5b0
```

```
24 #ifdef DEBUG
25 #define _heap_alloc _heap_alloc_base
26 #endif /* _DEBUG */
27
28 /**
29 *void *_heap_alloc_base(size_t size) - does actual allocation
30 *
31 *Purpose:
32 *      Same as malloc() except the new handler is not called.
33 *
34 *Entry:
35 *      See malloc
36 *
37 *Exit:
38 *      See malloc
39 *
40 *Exceptions:
41 *
42 *****
43
44 __forceinline void * __cdecl _heap_alloc (size_t size)
45 {
46
47     if (_crtheap == 0) {
48         _FF_MSGBANNER(); /* write run-time error banner */
49         _NMSG_WRITE(_RT_CRT_NOTINIT); /* write message */
50         _crtExitProcess(255); /* normally _exit(255) */
51     }
52
53
54     return HeapAlloc(_crtheap, 0, size ? size : 1);
55 }
```

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()

```
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;  
}
```

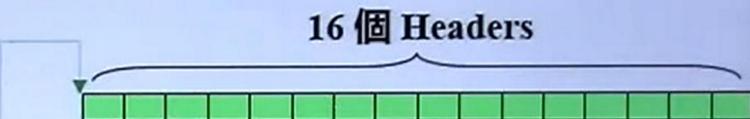
不論 big-block heap or  
small-block heap ,  
只要失敗就 return 0 ,  
別玩了。

heapinit.c

建立一个池塘pool  
从池塘中挖出16个Headers

4096

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() )



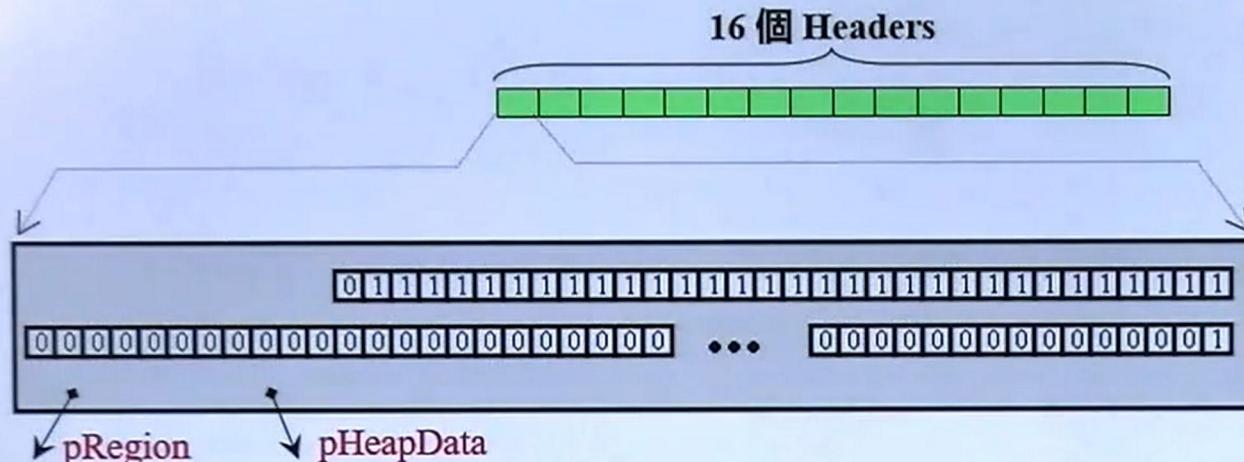
int \_\_cdecl \_\_sbh\_heap\_init( void )

sbheap.c

```
{  
    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;  
}
```



## SBH 之始 – \_heap\_init() 和 \_\_sbh\_heap\_init()



```
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

```

    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()

```

登记文件中第几行

```

/* 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

00000131
"ioinit.c" ←→
256
100h +
24h + 36
8h 8
= 12ch
→ 130h 填补为16倍数

fdffffdf
fill with 0xcd
fdffffdf
00000131 填补为16字节

131是因为用最后一个bit登记状态
void __cdecl _ioinit (void)
{
...
line81 if ( (pio = _malloc_crt( IOINFO_ARRAY_ELTS * sizeof(ioinfo) ) )
            = NULL )
...
} 32 8 第1次分配内存大小-256

```

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

# VC6 内存分配

```

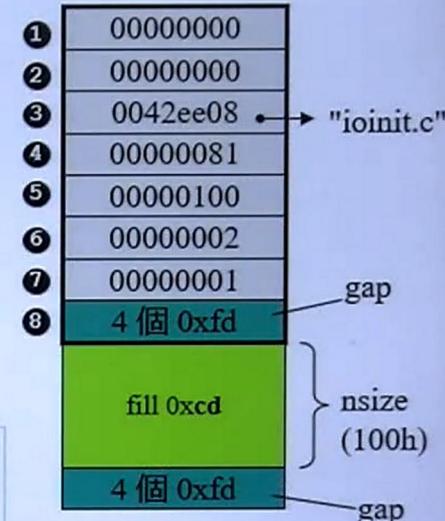
ExitProcess(code)
    _initterm(,,) //do terminators
        _endstdio(void)
        _initterm(,,) //do pre-terminators
    doexit(code, 0, 0)
    exit(code)
    main()
        _initterm(,,) //do C++ initializations
            _initstdio(void)
            _initterm(,,) //do initializations
    cinit() // do C data initialize
    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() // initialize lowio
        sbh_heap_init()
    heap_init(...)
    mainCRTStartup()
KERNEL32! bff8b6e60
KERNEL32! bff8b5980
KERNEL32! bff89f5b0

```

```

#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 SBRH 執取一內存塊  
... (續下頁) → 設妥 Debug Heap

Boolan 博览

# VC6 内存分配

```

ExitProcess(code)
    _initem(,,) //do terminators
    _endstdio(void)
    _initem(,,) //do pre-terminators
doexit(code, 0, 0)
⑨ exit(code)
⑧ main()
    _initem(,,) //do C++ initializations
    _initstdio(void)
    _initem(,,) //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! bff8b6e60
KERNEL32! bff8b5980
KERNEL32! bff89f5b0

```

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)

... (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;

```

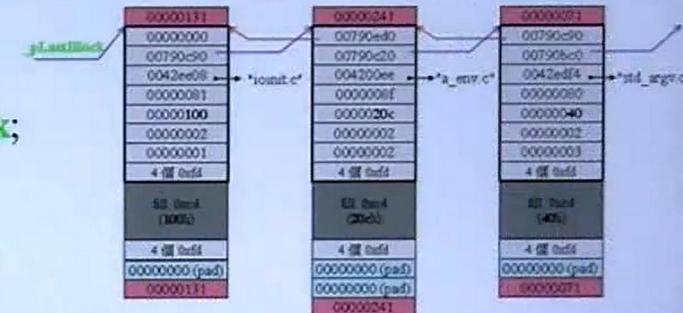
```

/* 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

Debug Heap



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(...)

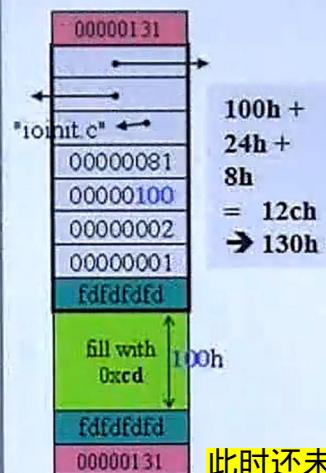
    heap alloc base(..) 算大小
        heap alloc_dbg(...)
        nh_malloc_dbg(...)
        malloc_dbg(...)

    ② ioinit() // initialize lowio
        sbh_heap_init()

    ① heap init(...)

mainCRTStartup()
KERNEL32! bff8b6e6()
KERNEL32! bff8b598()
KERNEL32! bff89f5b()

```



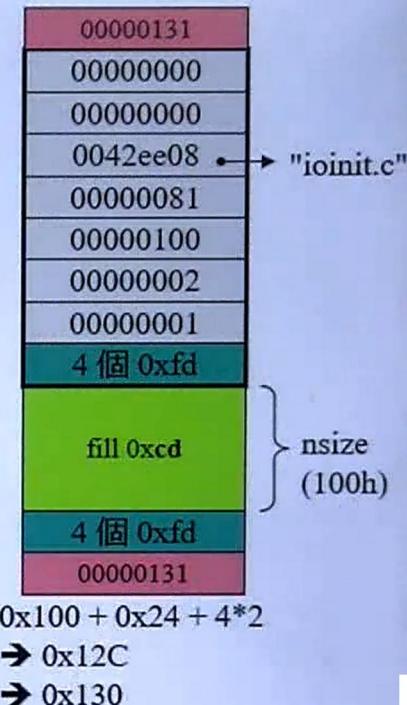
```

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

## 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()

```

```

// 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

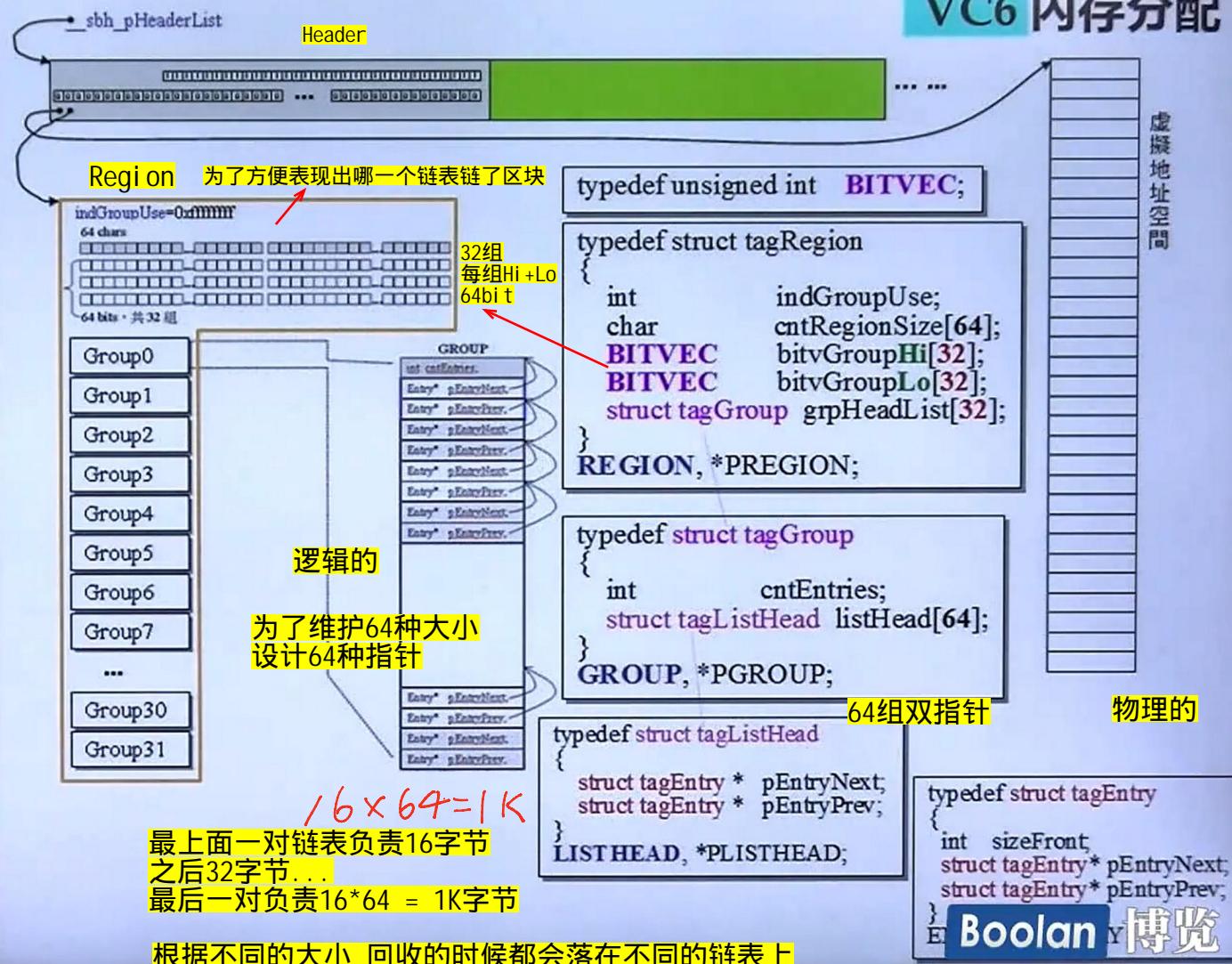
```

# VC6 内存分配

虚  
拟  
地  
址  
空  
间

```

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() // do C data initialize
    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() // initialize lowio
        sbh_heap_init()
        heap_init(...)
        mainCRTStartup()
        KERNEL32! bff8b6e60
        KERNEL32! bff8b5980
        KERNEL32! bff89f5b0
    
```

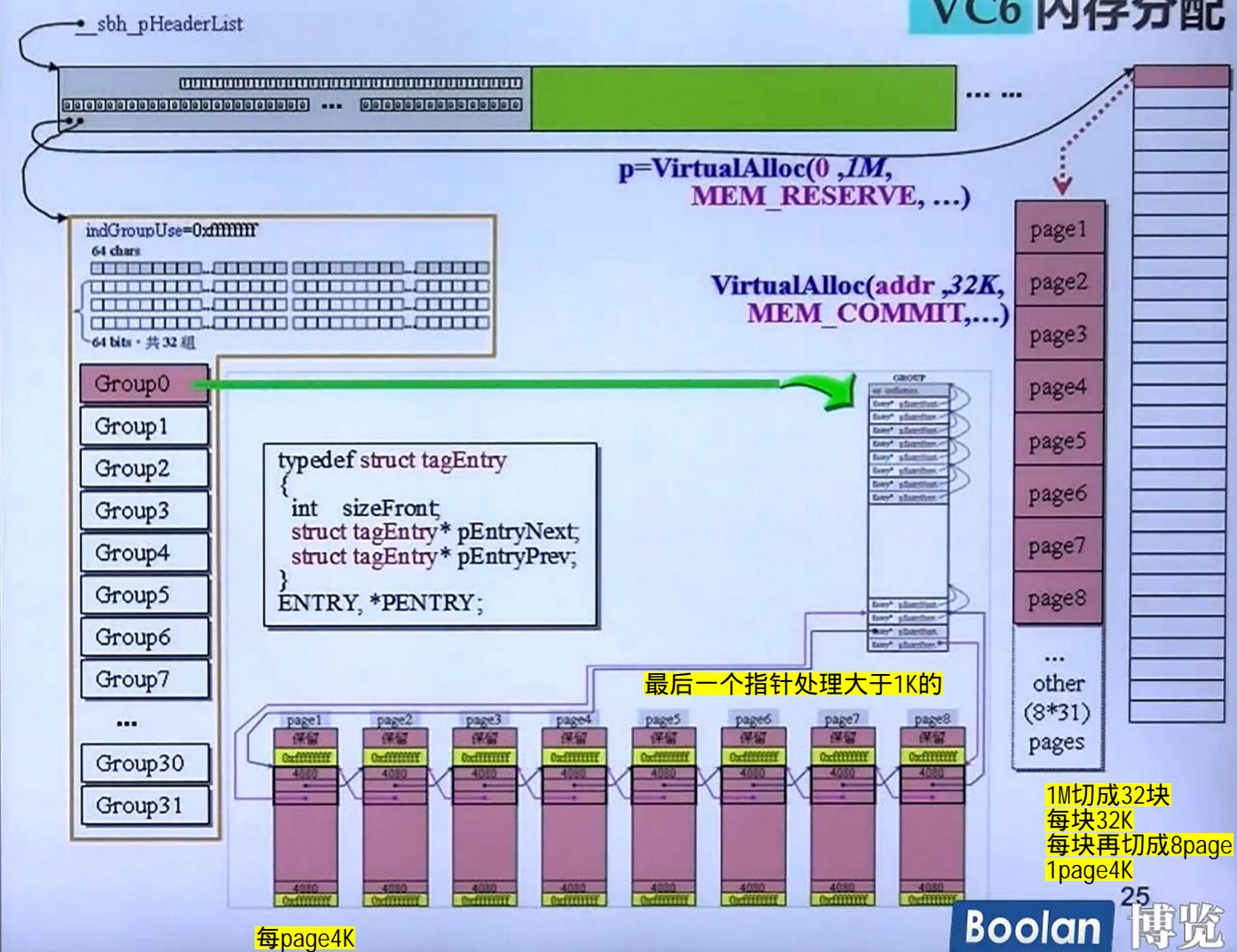


# VC6 内存分配

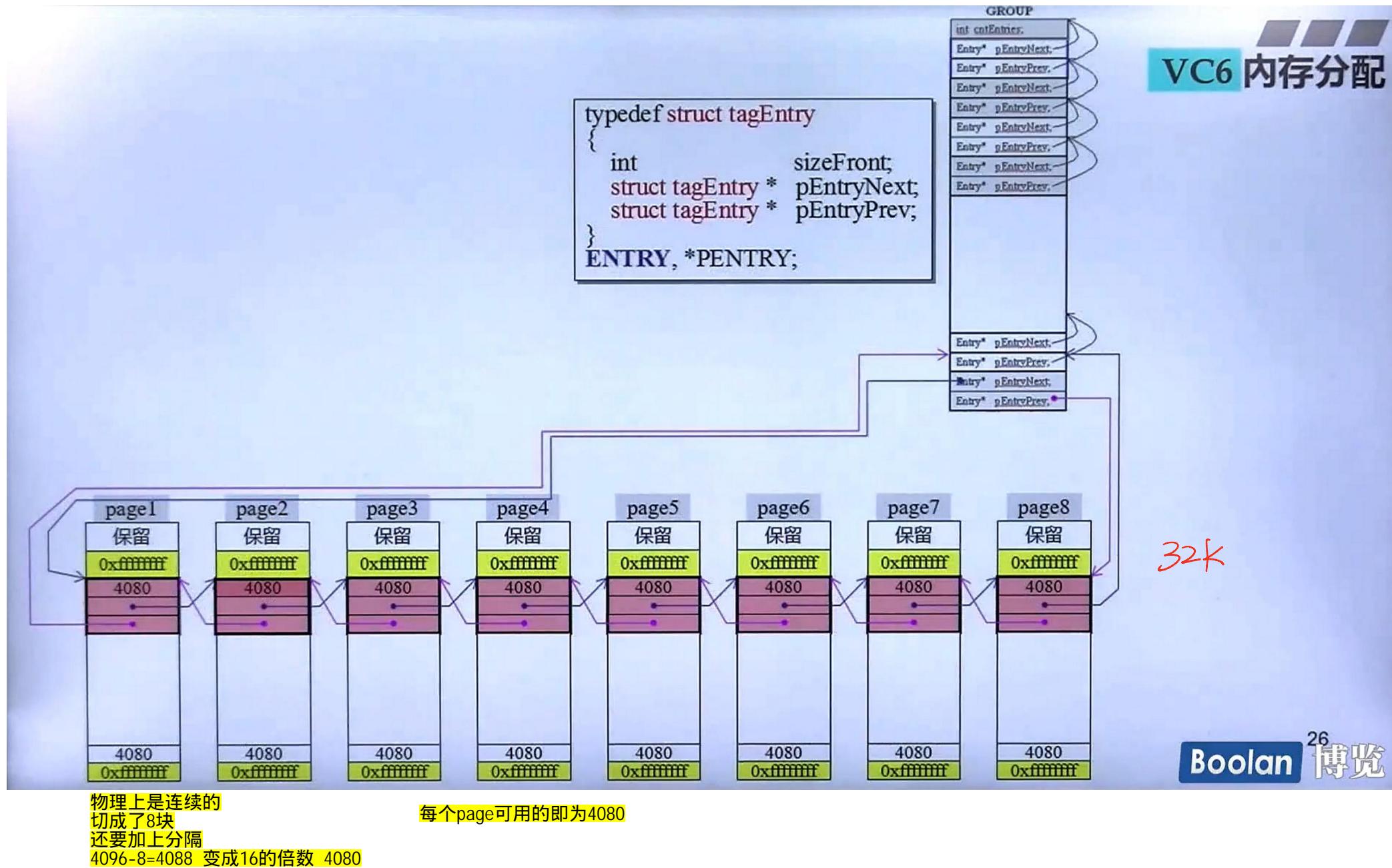
```

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! bff8b6e60
KERNEL32! bff8b5980
KERNEL32! bff89f5b0

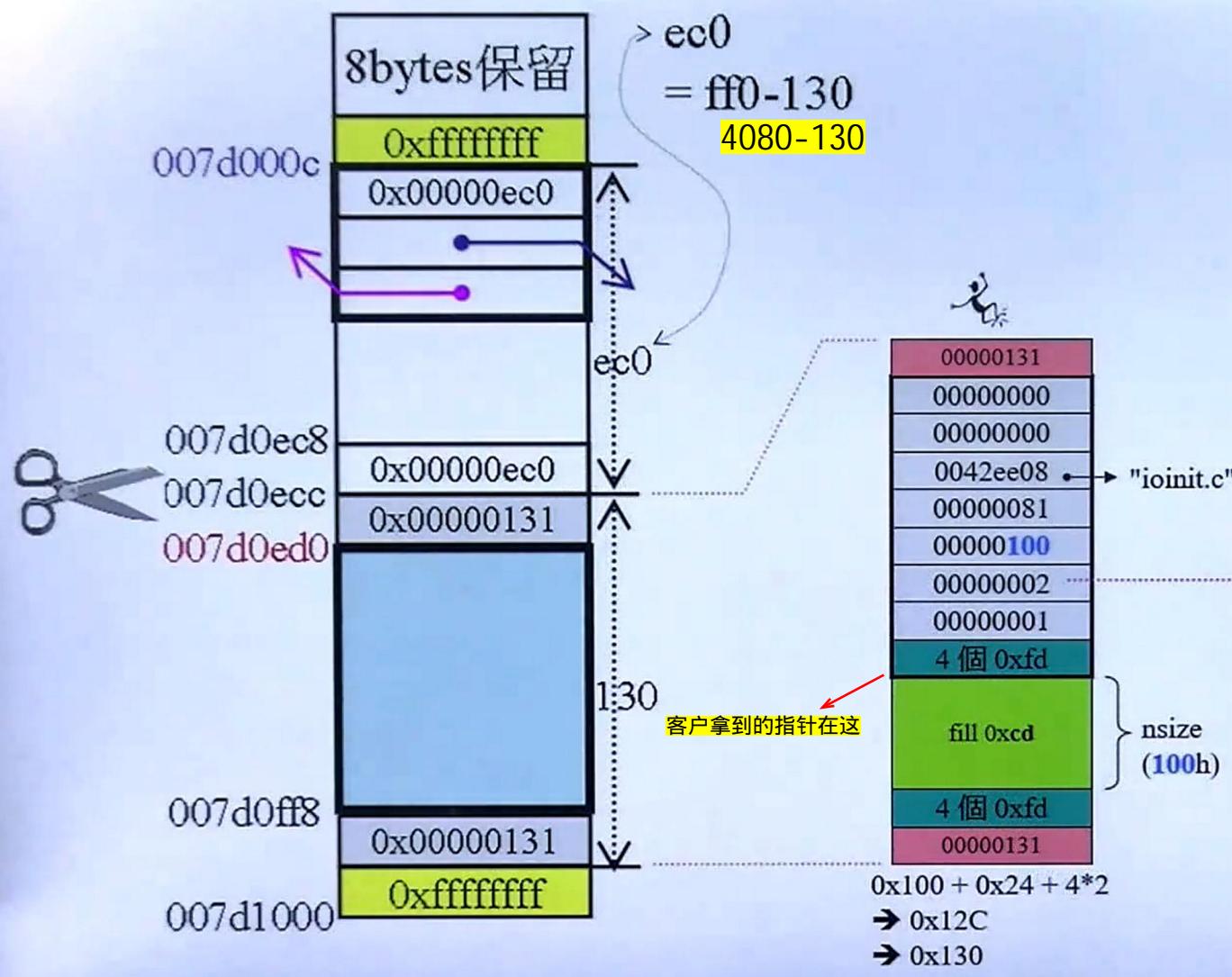
```



32K



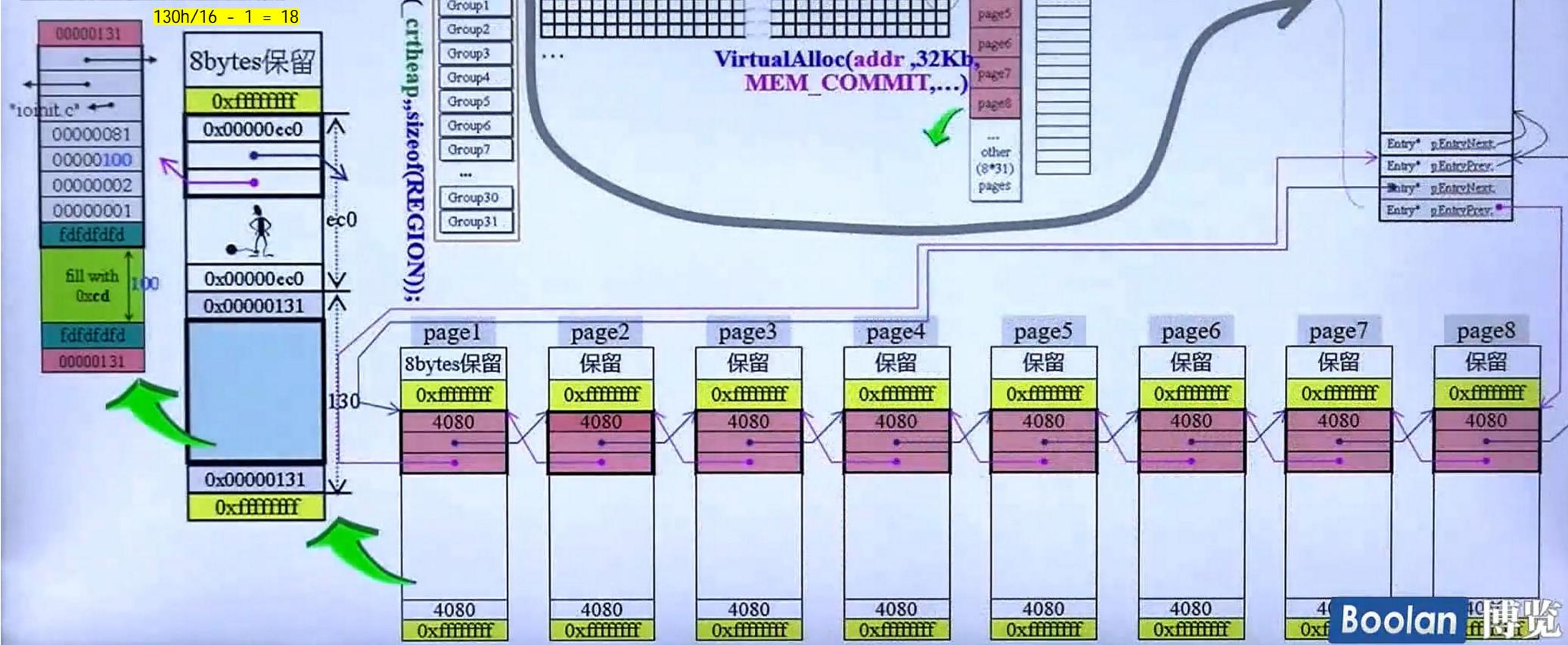
## 7. 内存分配精解3



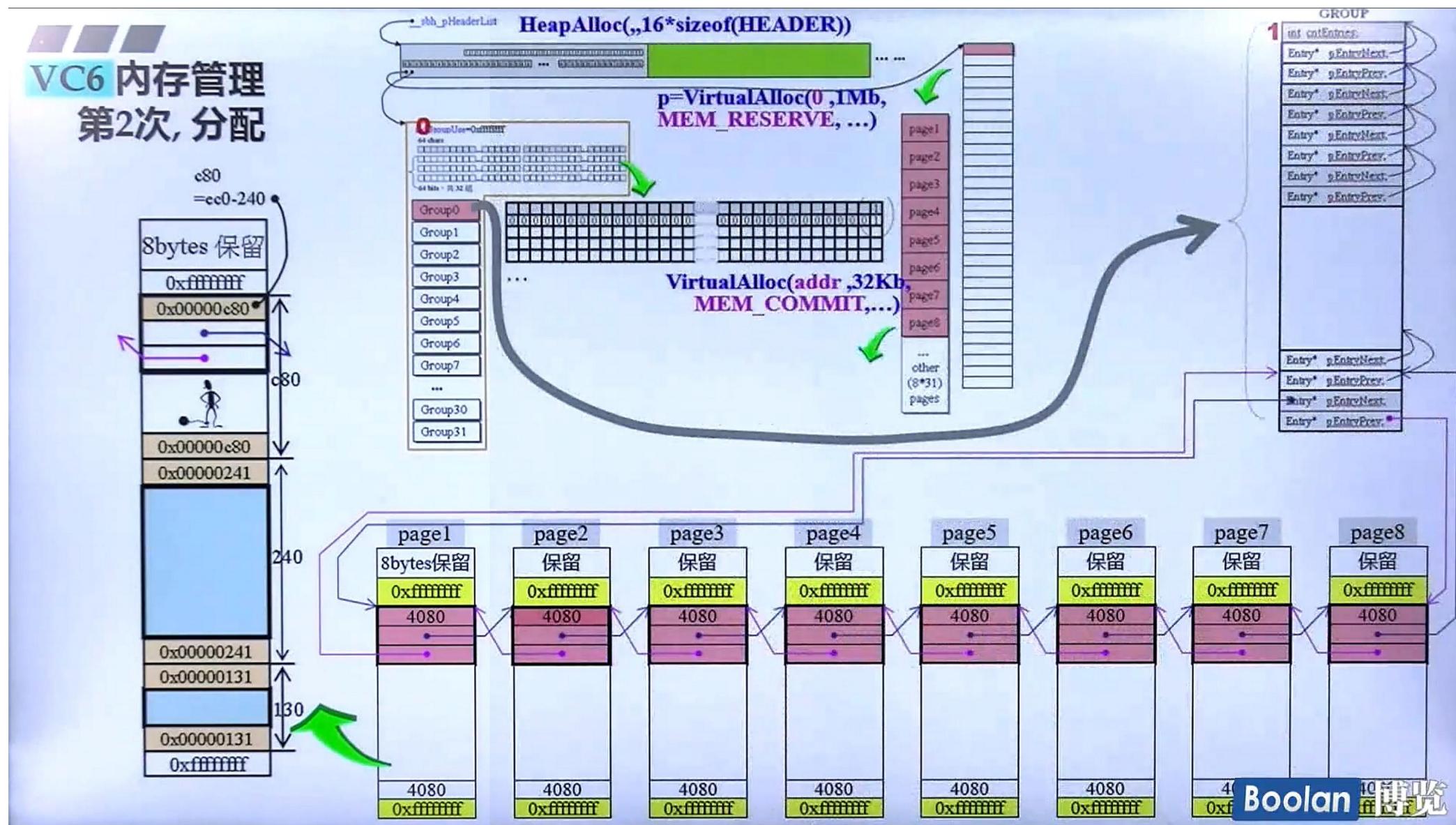
```
/* Memory block identification */
#define _FREE_BLOCK    0
#define _NORMAL_BLOCK  1
#define _CRT_BLOCK     2
#define _IGNORE_BLOCK   3
#define _CLIENT_BLOCK   4
#define _MAX_BLOCKS    5
```

# VC6 内存管理 首次分配

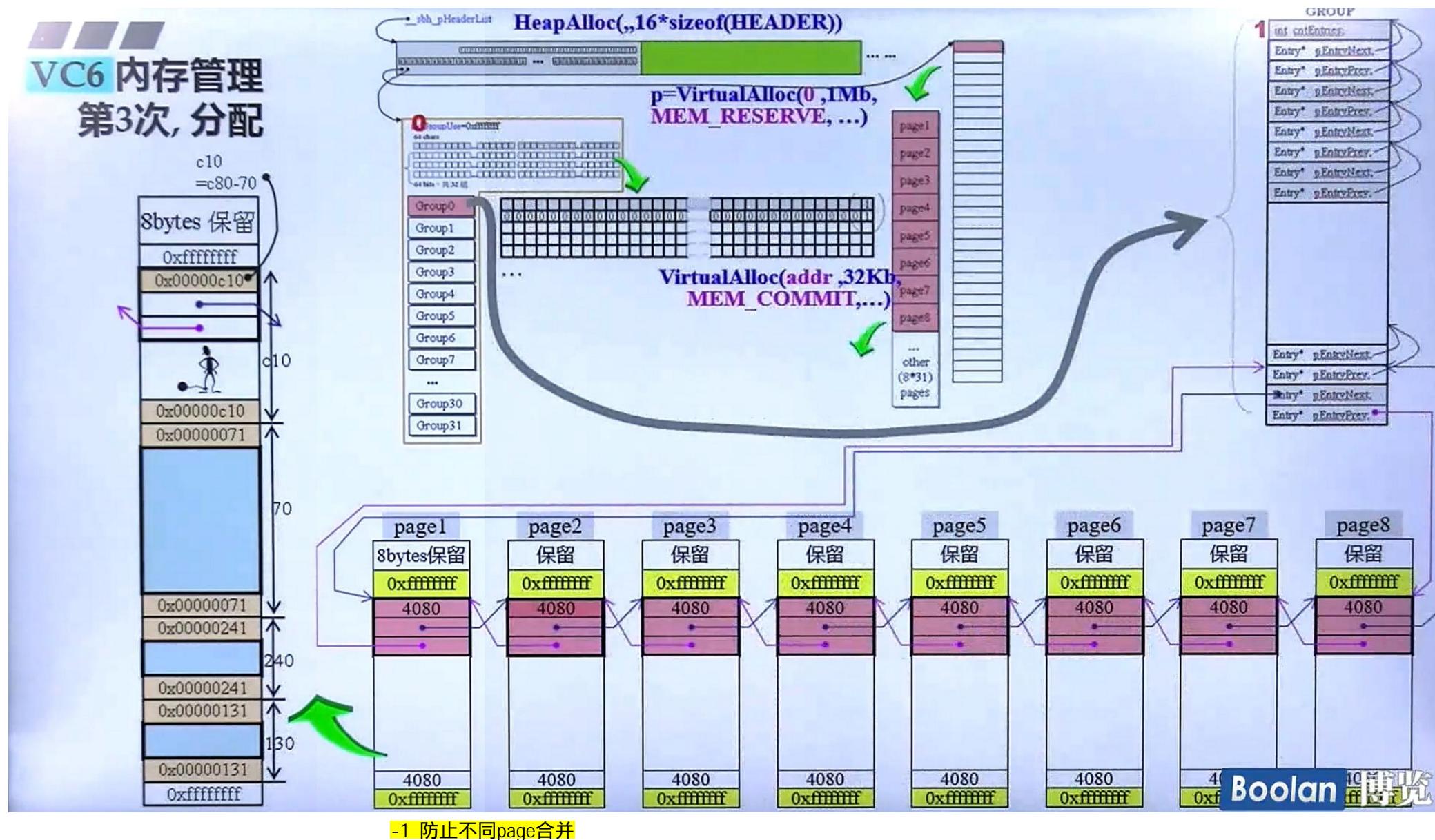
由 ioinit.c, line#81 申请  
100h, 区块大小130h,  
理应由 #18 lists 供应



# VC6 内存管理 第2次, 分配



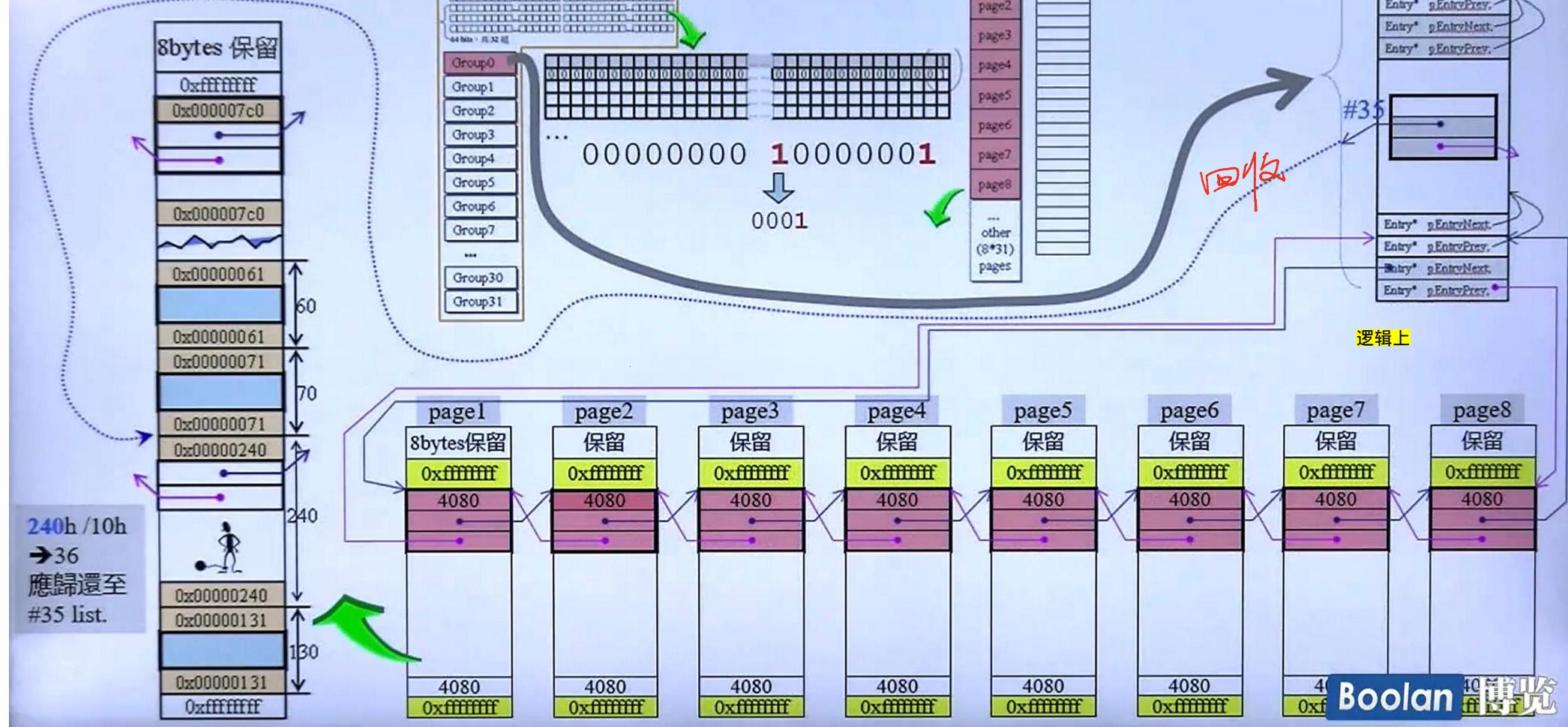
# VC6 内存管理 第3次, 分配



-1 防止不同page合并

# VC6 内存管理

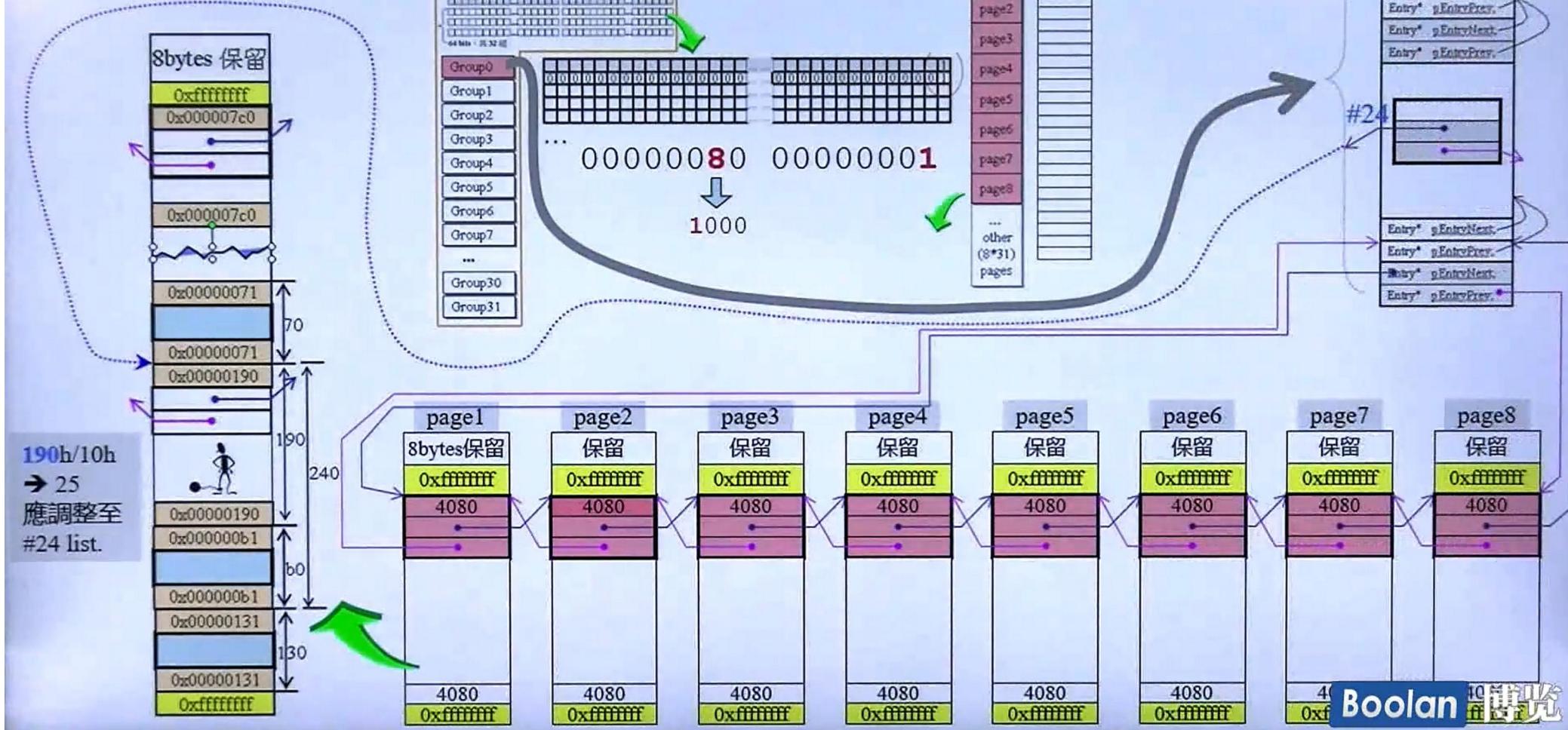
## 第15次，释还



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

# VC6 内存管理

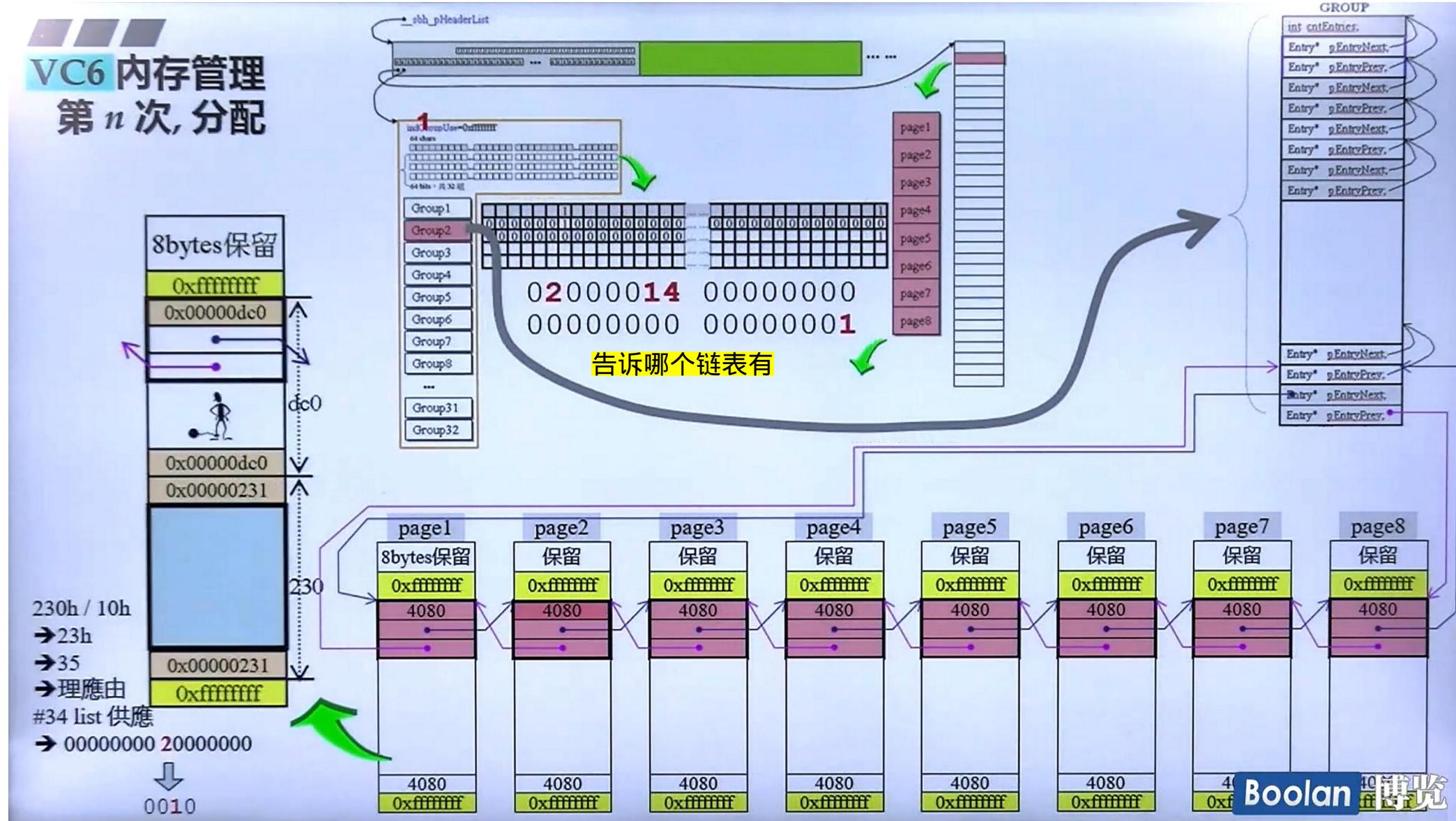
## 第16次, 分配



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



## 8. 内存分配精解4

# VC6 內存管理

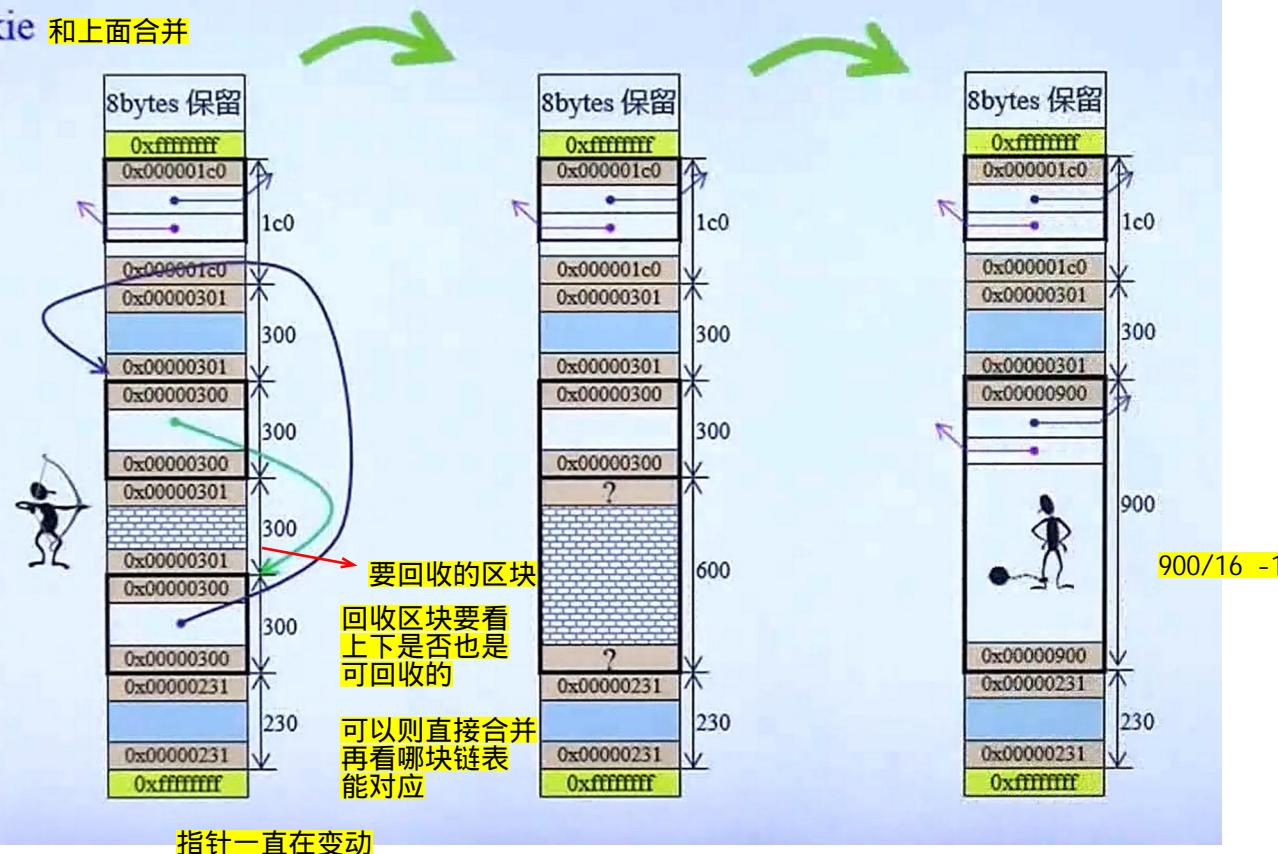
## 區塊之合併

上 cookie  
與  
下 cookie 和上面合并  
的作用

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）。

下方區塊若為 free ,  
合併之  
( unlink 下方區塊 )

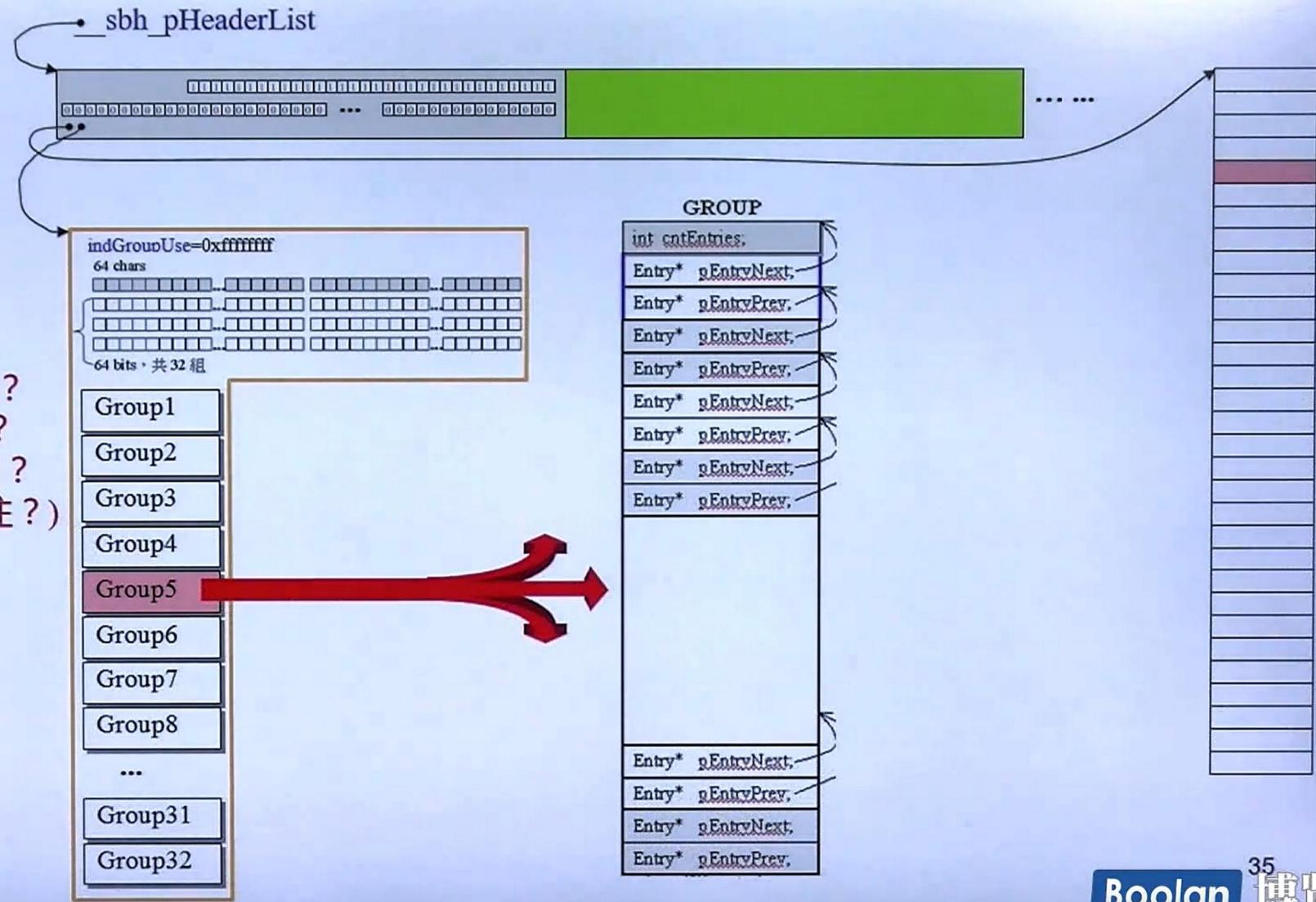
上方區塊若為 free ,  
合併之  
( unlink 上方區塊 )



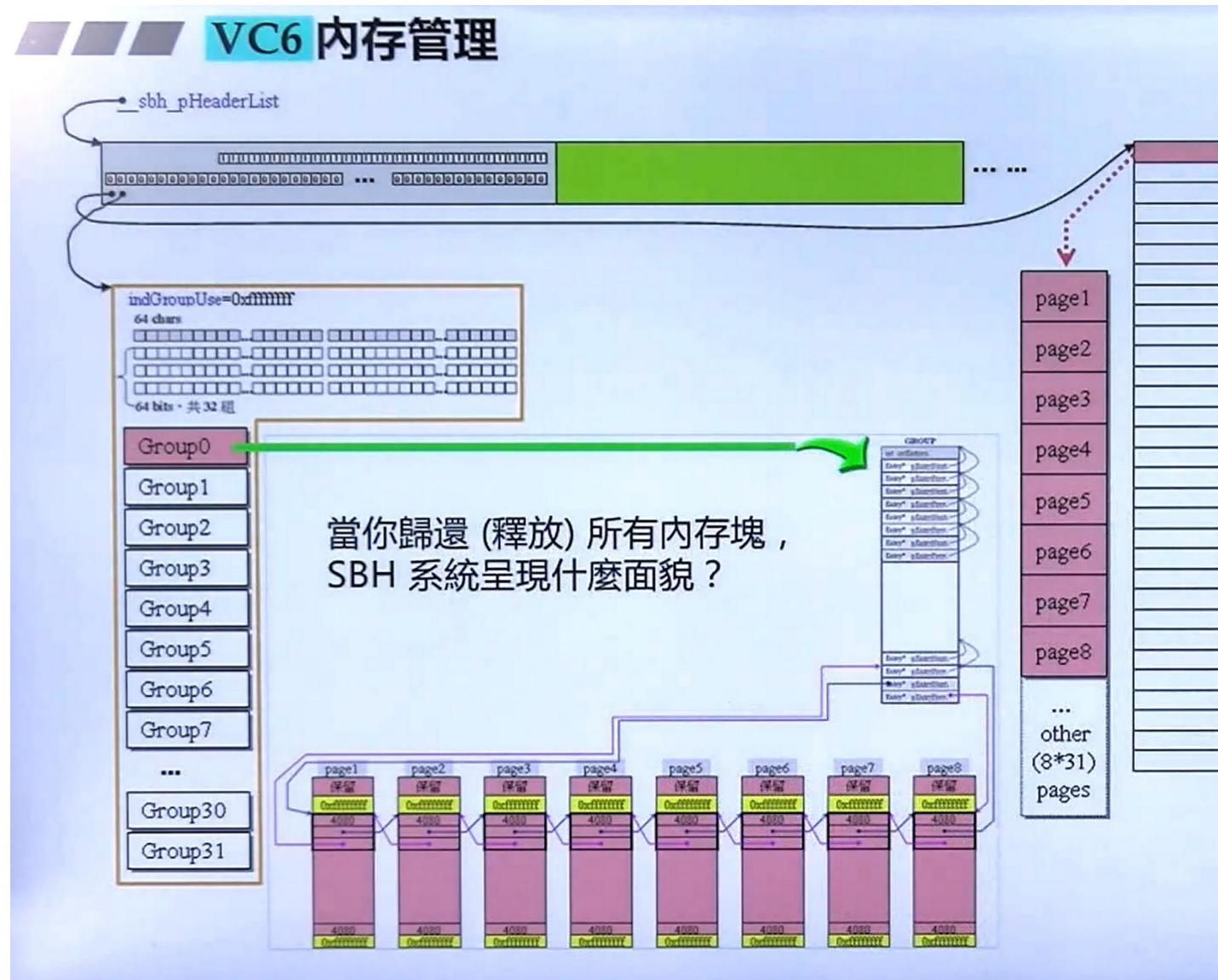
free(p)

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？)



# VC6 内存管理



9. Main()生前所有内存分配之实例观察与解释

blocks  
allocated  
before  
main()

`environ` is pointer to pointer table, table  
中的每個 entry 都是 pointer to string which  
represents an environment variable.

`environ` : 0x007d0be0

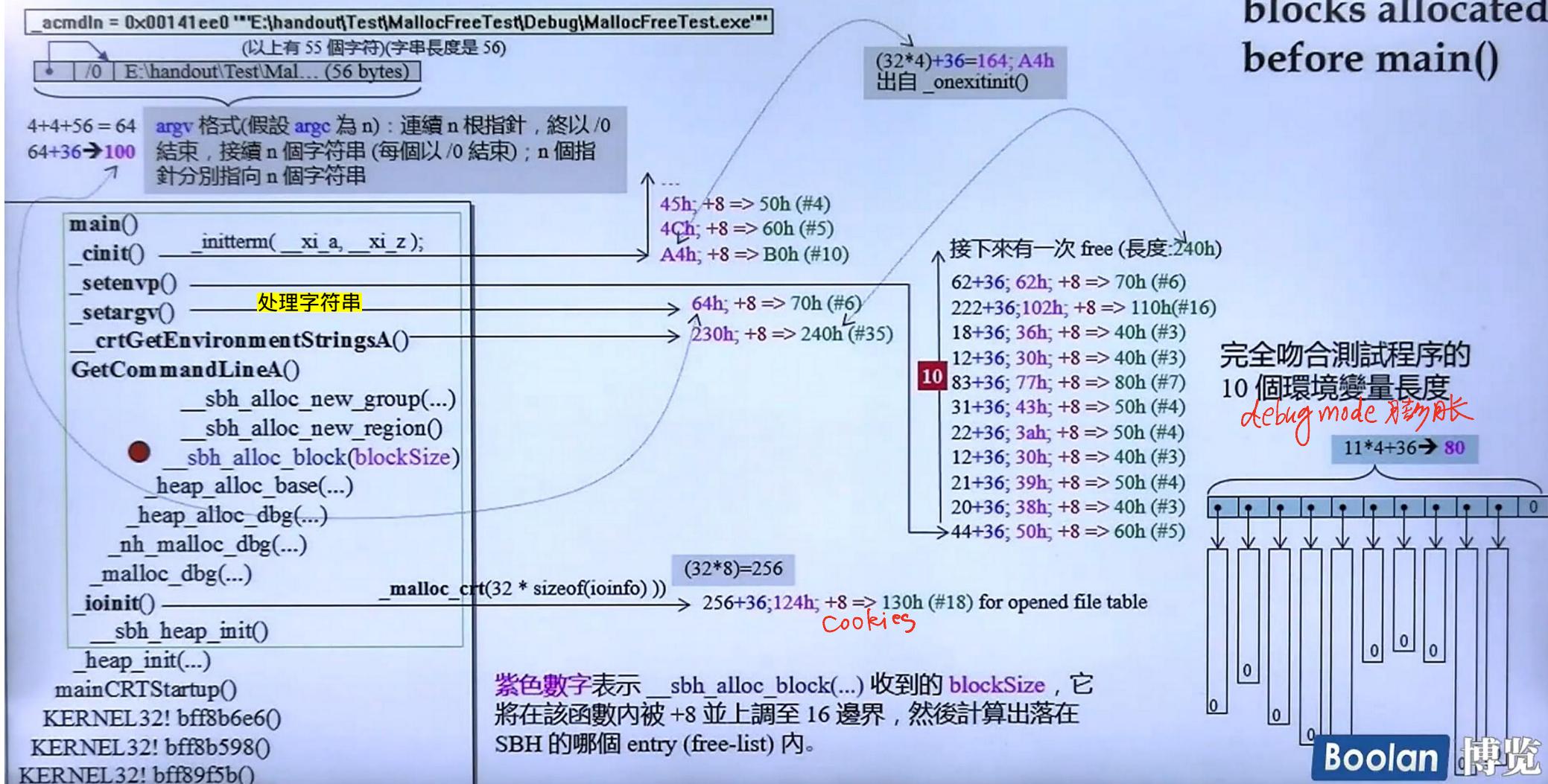
	Address:	Value	Description
0	007D0BE0	A0 08 7D 00 50 08 7D 00 ..\P..	
1	007D0BE8	10 08 7D 00 C0 0A 7D 00 ..\....	
2	007D0BF0	70 0A 7D 00 F0 09 7D 00 p.\..	
3	007D0BF8	80 09 7D 00 70 09 7D 00 ..\p..	
4	007D0C00	60 08 7D 00 F0 07 7D 00 ..\....	
5	007D0C08	00 00 00 00 FD FD FD FD .....	
6	007D0C10	00 00 00 00 00 00 00 00 .....	
7	007D0C18	61 00 00 00 71 00 00 00 a...q...	
8	007D0C20	D0 0E 7D 00 C0 0B 7D 00 ..\....	
9	007D0C28	24 D7 46 00 80 00 00 00 \$膜.....	
10	007D0C30	40 00 00 00 A2 00 00 00 a.....	
11	007D0BA0	54 4D 50 3D 43 3A 5C 57 TMP=C:\W	
12	007D0BA8	49 4E 44 4F 57 53 5C 54 INDOWS\T	
13	007D0BB0	45 4D 50 00 FD FD FD FD EMP. 長度 20	
14	007D0B50	54 45 4D 50 3D 43 3A 5C TEMP=C:\	
15	007D0B58	57 49 4E 44 4F 57 53 5C WINDOWS\	
16	007D0B60	54 45 4D 50 00 FD FD FD TEMP 長度 21	
17	007D0B10	50 52 4F 4D 50 54 3D 24 PROMPT=\$	
18	007D0B18	70 24 67 00 FD FD FD FD p\$g. 長度 12	
19	007D0AC0	77 69 6E 62 6F 6F 74 64 winbootd	
20	007D0AC8	69 72 3D 43 3A 5C 57 49 ir=C:\WI	
21	007D0AD0	4E 44 4F 57 53 00 FD FD NDOWS.	
22	007D0AD8	FD FD 00 00 00 00 00 00 .... 長度 22	
23	007D0A70	43 4F 4D 53 50 45 43 3D COMSPEC=	
24	007D0A78	43 3A 5C 57 49 4E 44 4F C:\WINDO	
25	007D0A80	57 53 5C 43 4F 4D 40 41 WS\COMMA	
26	007D0A88	4E 44 2E 43 4F 4D 00 FD ND.COM.	
27	007D0A90	FD FD FD 00 00 00 00 00 ..... 長度 31	

5	007D09F0	50 41 54 48 3D 43 3A 5C PATH=C:\	長度83
6	007D09F8	57 49 4E 44 4F 57 53 3B WINDOWS;	
7	007D0A00	43 3A 5C 57 49 4E 44 4F C:\WINDO	
8	007D0A08	57 53 5C 43 4F 4D 40 41 WS\COMMA	
9	007D0A10	4E 44 3B 45 3A 5C 55 54 ND;E:\UT	
10	007D0A18	49 4C 49 54 59 3B 43 3A ILLITY;C:	
11	007D0A20	5C 50 52 4F 47 52 41 7E \PROGRA~	
12	007D0A28	31 5C 43 4F 4D 40 4F 4E 1\COMMON	
13	007D0A30	7E 31 5C 4D 55 56 45 45 ~1\MUVEE	
14	007D0A38	54 7E 31 5C 30 33 30 36 T~1\0306	
15	007D0A40	32 35 AA FD FD FD FD AA 25. -	
16	007D09B0	43 4D 44 46 49 4E 45 3D CMDLINE=	長度12
17	007D09B8	57 49 4E 00 FD FD FD FD WIN.	
18	007D0970	77 69 6E 64 69 72 3D 43 windir=C	長度18
19	007D0978	3A 5C 57 49 4E 44 4F 57 :WINDOW	
20	007D0980	53 00 FD FD FD FD 00 00 S. ..	
21	007D0860	5F 41 43 50 5F 50 41 54 _ACP_PAT	長度??
22	007D0868	48 3D 43 3A 5C 4D 53 44 H=C:\MSD	
23	007D0870	45 56 5C 43 4F 4D 40 4F EU\COMM0	
24	007D0878	4E 5C 4D 53 44 45 56 39 N\MSDEV9	
25	007D0880	38 5C 42 49 4E 3B 43 3A 8\BIN;C:	
26	007D0888	5C 6D 73 64 65 76 5C 56 \msdev\U	
27	007D0890	43 39 38 5C 42 49 4E 3B C98\BIN;	
28	007D0898	43 3A 5C 4D 53 44 45 56 C:\MSDEV	
29	007D08A0	5C 43 4F 4D 40 4F 4E 5C \COMMON\	
30	007D08A8	54 4F 4C 53 3B 43 3A TOOLS;C:	
31	007D08A0	5C 4D 53 44 45 56 5C 43 \MSDEV\U	
32	007D07F0	5F 41 43 50 5F 4C 49 42 _ACP_LIB	長度62
33	007D07F8	3D 43 3A 5C 6D 73 64 65 =C:\msde	
34	007D0800	76 5C 56 43 39 38 5C 4C v\VC98\L	
35	007D0808	49 42 3B 43 3A 5C 6D 73 IB;C:\ms	
36	007D0810	64 65 76 5C 56 43 39 38 dev\VC98	
37	007D0818	5C 4D 46 43 5C 4C 49 42 \MFC\LIB	
38	007D0820	3B 63 3A 5C 62 6F 6F 73 ;c:\boos	
39	007D0828	74 5C 6C 69 62 00 FD FD t\lib.	
40	007D0830	FD FD 00 00 00 00 00 00 .....	

進入main()之前，CRT 已經做了許多工作，其中需要若干 memory，因此當 main() 首次調用 malloc() 時，已有若干 blocks 掛在 sbh (small blocks heap) 所維護的 free lists 上頭。



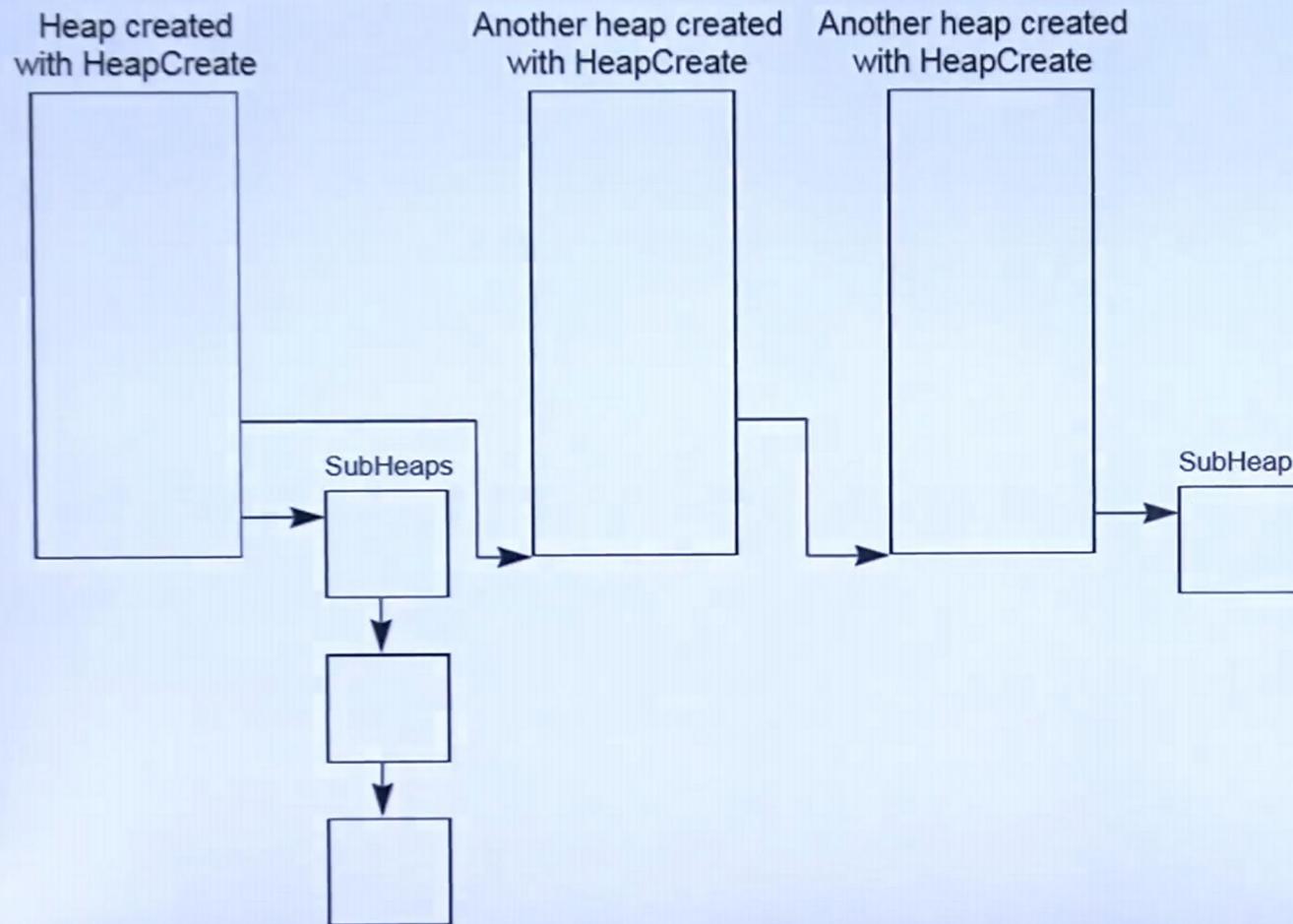
## blocks allocated before main()



## 10. HeapAlloc()角色与影响



# Windows Heap Management



# Windows XP sp2 Heap 觀察結果

The screenshot shows the Windows Task Manager with a memory dump loaded. The left pane displays the context of the main() function, showing variable values:

Name	Value
hMyHeap	0x00390000
hHeap	0x00140000
lp3	0xffffffff
lp2	0xffffffff
lp	0x00142cc0 X
hMyHeap2	0xffffffff

The right pane shows a memory dump starting at address 0x00390000. A yellow box highlights the value at offset 178 (0x003901F0), which is 0x0139. A green box highlights the text "free[0], 位於 heap 的 offset 178 處." above the dump area.

Address	Value
00390000	C8 00 00 00 8D 01 00 00 FF EE FF EE 62 10 00 50
00390010	60 00 00 40 00 FE 00 00 00 10 00 00 20 00 00
00390020	00 02 00 00 00 20 00 00 2D 02 00 00 FF EF FD 7F
00390030	05 00 08 06 00 00 00 00 00 00 00 00 00 00 00 00
00390040	00 00 00 00 98 05 39 00 17 00 00 00 F8 FF FF FF
00390050	50 00 39 00 50 00 39 00 40 06 39 00 00 00 00 00
00390060	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00390070	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00390080	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00390090	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
003900A0	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
003900B0	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
003900C0	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
003900D0	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
003900E0	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
003900F0	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00390100	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00390110	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00390120	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00390130	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00390140	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00390150	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00390160	00 00 00 00 00 00 00 00 00 00 00 00 00 01 00 00
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
00390190	90 01 39 00 90 01 39 00 98 01 39 00 98 01 39 00
003901A0	A0 01 39 00 A0 01 39 00 A0 01 39 00 A0 01 39 00
003901B0	B0 01 39 00 B0 01 39 00 B0 01 39 00 B0 01 39 00
003901C0	C0 01 39 00 C0 01 39 00 C0 01 39 00 C0 01 39 00
003901D0	D0 01 39 00 D0 01 39 00 D0 01 39 00 D0 01 39 00
003901E0	E0 01 39 00 E0 01 39 00 E8 01 39 00 E8 01 39 00
003901F0	F0 01 39 00 F0 01 39 00 F8 01 39 00 F8 01 39 00

free[0], 位於 heap 的 offset 178 處.

190160	00 00 00 00 00 00 00 00 00 00 00 00 00 01 00 00 00
190170	00 00 00 00 00 00 00 00 A0 1E 39 00 A0 1E 39 00
190180	80 01 39 00 80 01 39 00 88 01 39 00 88 01 39 00
190190	90 01 39 00 90 01 39 00 98 01 39 00 98 01 39 00
1901A0	A0 01 39 00 A0 01 39 00 A8 01 39 00 A8 01 39 00
1901B0	B0 01 39 00 B0 01 39 00 B8 01 39 00 B8 01 39 00
1901C0	C0 01 39 00 C0 01 39 00 C8 01 39 00 C8 01 39 00
1901D0	D0 01 39 00 D0 01 39 00 D8 01 39 00 D8 01 39 00
1901E0	E0 01 39 00 E0 01 39 00 E8 01 39 00 E8 01 39 00
1901F0	F0 01 39 00 F0 01 39 00 F8 01 39 00 F8 01 39 00
190200	00 02 39 00 00 02 39 00 08 02 39 00 08 02 39 00
190210	10 02 39 00 10 02 39 00 18 02 39 00 18 02 39 00
190220	20 02 39 00 20 02 39 00 28 02 39 00 28 02 39 00
190230	30 02 39 00 30 02 39 00 38 02 39 00 38 02 39 00
190240	40 02 39 00 40 02 39 00 48 02 39 00 48 02 39 00
190250	50 02 39 00 50 02 39 00 58 02 39 00 58 02 39 00
190260	60 02 39 00 60 02 39 00 68 02 39 00 68 02 39 00
190270	70 02 39 00 70 02 39 00 78 02 39 00 78 02 39 00
190280	80 02 39 00 80 02 39 00 88 02 39 00 88 02 39 00
190290	90 02 39 00 90 02 39 00 98 02 39 00 98 02 39 00
1902A0	A0 02 39 00 A0 02 39 00 A8 02 39 00 A8 02 39 00
1902B0	B0 02 39 00 B0 02 39 00 B8 02 39 00 B8 02 39 00
1902C0	C0 02 39 00 C0 02 39 00 C8 02 39 00 C8 02 39 00
1902D0	D0 02 39 00 D0 02 39 00 D8 02 39 00 D8 02 39 00
1902E0	E0 02 39 00 E0 02 39 00 E8 02 39 00 E8 02 39 00
1902F0	F0 02 39 00 F0 02 39 00 F8 02 39 00 F8 02 39 00
190300	00 03 39 00 00 03 39 00 08 03 39 00 08 03 39 00
190310	10 03 39 00 10 03 39 00 18 03 39 00 18 03 39 00
190320	20 03 39 00 20 03 39 00 28 03 39 00 28 03 39 00
190330	30 03 39 00 30 03 39 00 38 03 39 00 38 03 39 00
190340	40 03 39 00 40 03 39 00 48 03 39 00 48 03 39 00
190350	50 03 39 00 50 03 39 00 58 03 39 00 58 03 39 00

free[0], 位於 heap 的 offset 178 處.

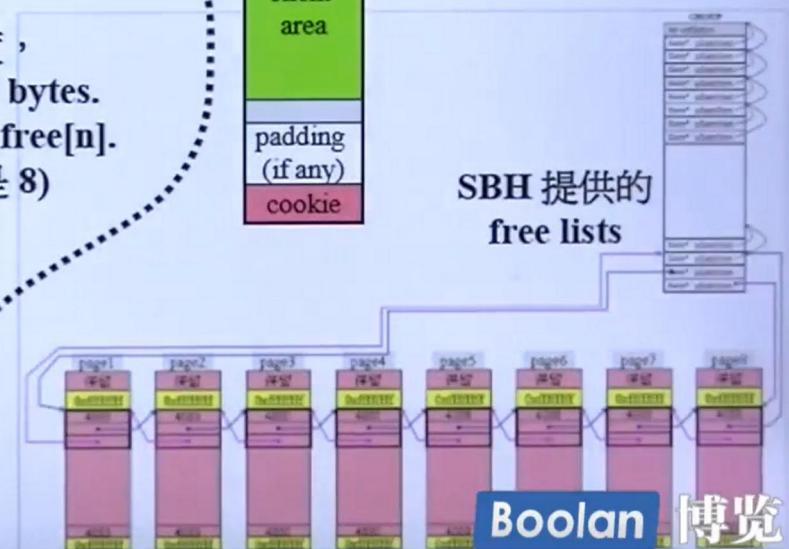
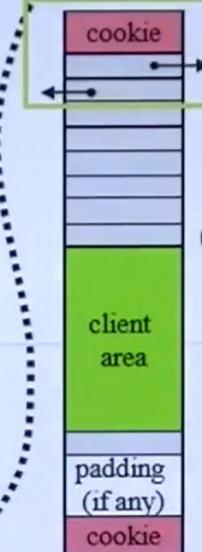
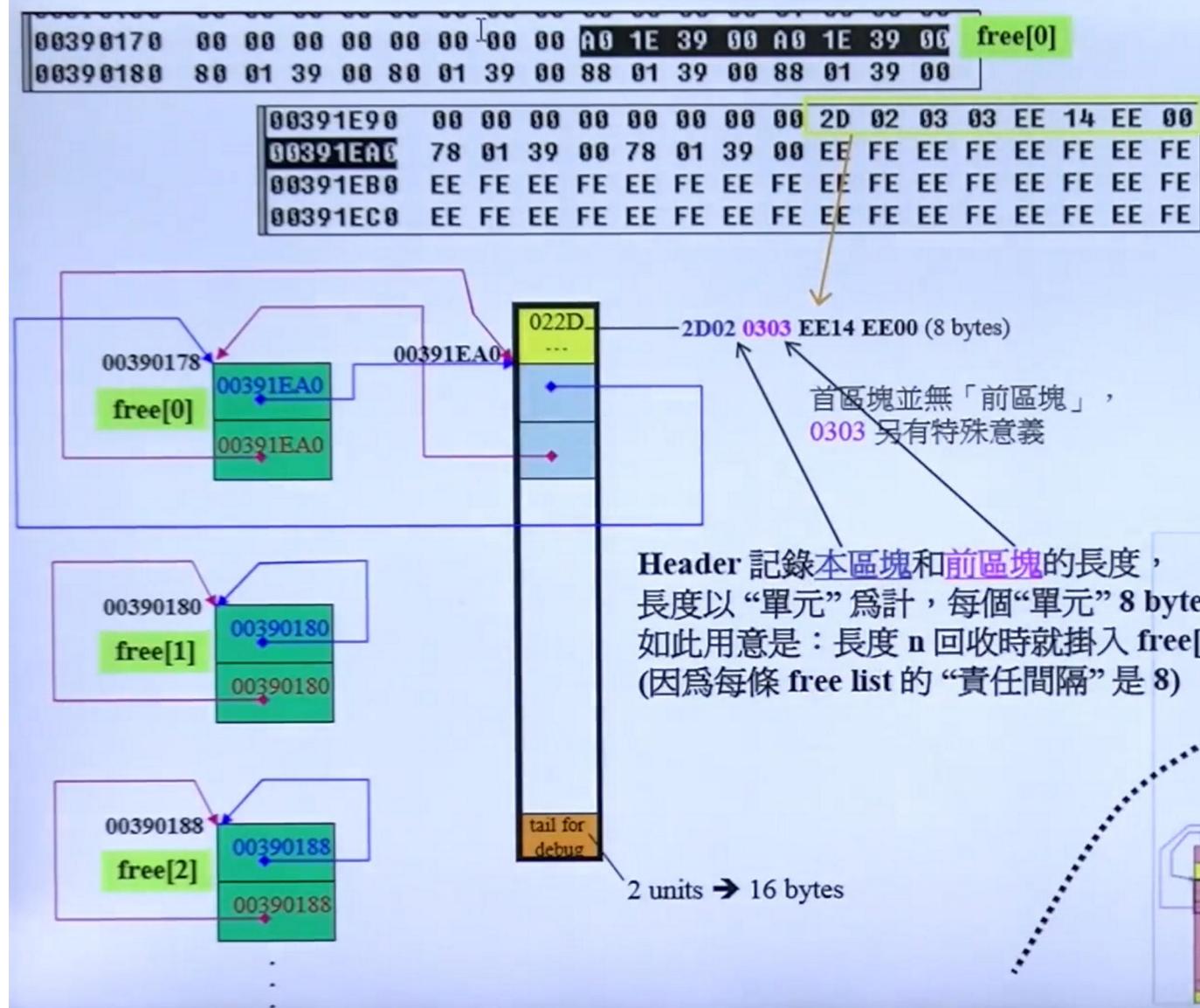
00390350	50 03 39 00 50 03 39 00 58 03 39 00 58 03 39 00
00390360	60 03 39 00 60 03 39 00 68 03 39 00 68 03 39 00
00390370	70 03 39 00 70 03 39 00 78 03 39 00 78 03 39 00
00390380	80 03 39 00 80 03 39 00 88 03 39 00 88 03 39 00
00390390	90 03 39 00 90 03 39 00 98 03 39 00 98 03 39 00
003903A0	A0 03 39 00 A0 03 39 00 A8 03 39 00 A8 03 39 00
003903B0	B0 03 39 00 B0 03 39 00 B8 03 39 00 B8 03 39 00
003903C0	C0 03 39 00 C0 03 39 00 C8 03 39 00 C8 03 39 00
003903D0	D0 03 39 00 D0 03 39 00 D8 03 39 00 D8 03 39 00
003903E0	E0 03 39 00 E0 03 39 00 E8 03 39 00 E8 03 39 00
003903F0	F0 03 39 00 F0 03 39 00 F8 03 39 00 F8 03 39 00
00390400	00 04 39 00 00 04 39 00 08 04 39 00 08 04 39 00
00390410	10 04 39 00 10 04 39 00 18 04 39 00 18 04 39 00
00390420	20 04 39 00 20 04 39 00 28 04 39 00 28 04 39 00
00390430	30 04 39 00 30 04 39 00 38 04 39 00 38 04 39 00
00390440	40 04 39 00 40 04 39 00 48 04 39 00 48 04 39 00
00390450	50 04 39 00 50 04 39 00 58 04 39 00 58 04 39 00
00390460	60 04 39 00 60 04 39 00 68 04 39 00 68 04 39 00
00390470	70 04 39 00 70 04 39 00 78 04 39 00 78 04 39 00
00390480	80 04 39 00 80 04 39 00 88 04 39 00 88 04 39 00
00390490	90 04 39 00 90 04 39 00 98 04 39 00 98 04 39 00
003904A0	A0 04 39 00 A0 04 39 00 A8 04 39 00 A8 04 39 00
003904B0	B0 04 39 00 B0 04 39 00 B8 04 39 00 B8 04 39 00
003904C0	C0 04 39 00 C0 04 39 00 C8 04 39 00 C8 04 39 00
003904D0	D0 04 39 00 D0 04 39 00 D8 04 39 00 D8 04 39 00
003904E0	E0 04 39 00 E0 04 39 00 E8 04 39 00 E8 04 39 00
003904F0	F0 04 39 00 F0 04 39 00 F8 04 39 00 F8 04 39 00
00390500	00 05 39 00 00 05 39 00 08 05 39 00 08 05 39 00
00390510	10 05 39 00 10 05 39 00 18 05 39 00 18 05 39 00
00390520	20 05 39 00 20 05 39 00 28 05 39 00 28 05 39 00
00390530	30 05 39 00 30 05 39 00 38 05 39 00 38 05 39 00
00390540	40 05 39 00 40 05 39 00 48 05 39 00 48 05 39 00
00390550	50 05 39 00 50 05 39 00 58 05 39 00 58 05 39 00
00390560	60 05 39 00 60 05 39 00 68 05 39 00 68 05 39 00
00390570	70 05 39 00 70 05 39 00 78 05 39 00 78 05 39 00
00390580	88 06 free[127] 01 00 00 00 00 00 00 30 39 00
00390590	00 D0

## Windows XP sp2 Heap 觀察結果

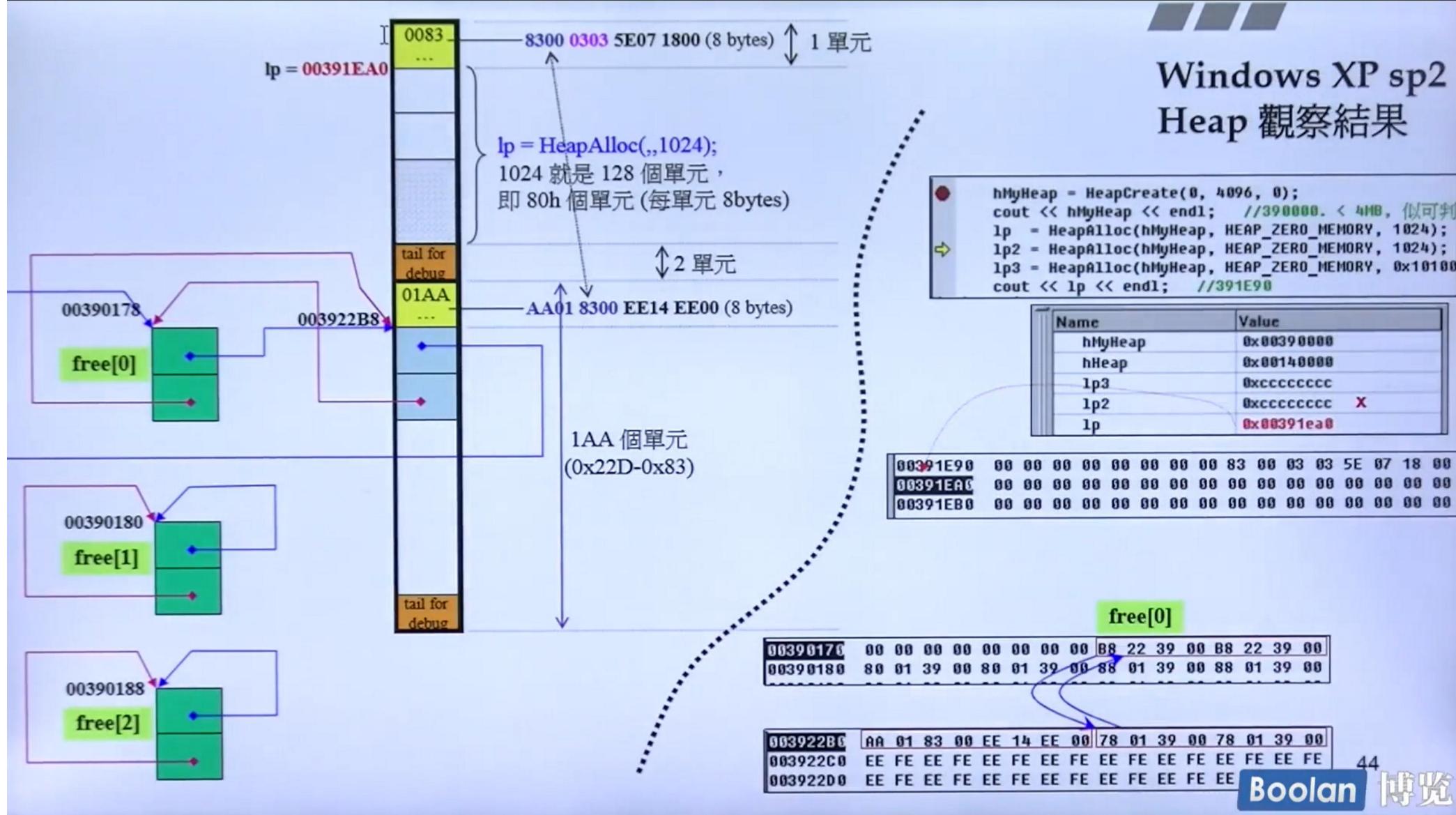
128 個 free lists，  
也就是 128 對指針，  
 $0x390178 + 128 * 8$   
 $= 0x390578$

191E70	00 00 00 00 00 00 00 00 size: 022D 10 00 00 00
191E80	00 00 00 00 00 00 00 00
191E90	40 00 00 00 00 00 00 00 2D 02 03 03 EE 14 EE 00
191EA0	78 01 39 00 78 01 39 00 EE FE EE FE EE FE EE FE EE FE
191EB0	EE FE
191EC0	EE FE

# Windows XP sp2 Heap 觀察結果



# Windows XP sp2 Heap 觀察結果



11. io\_init() - startup 的第二项大工程

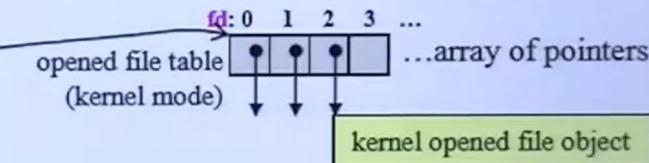
## ■■■ \_io\_init()

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 理解為 file 的任何事務 — file 在此是一個廣義概念。

```
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 做為 **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 FILEs，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() [highlight]
    __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, _osfhndl and _osfile for the first three ioinfo structs,
*corrsponding to handles 0, 1 and 2, are initialized as follows:
*    If the value in _osfhndl 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:
******/
```

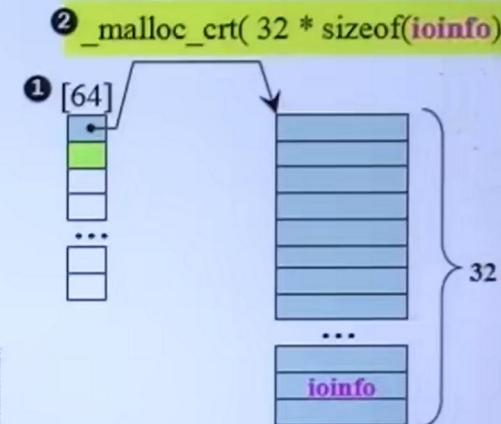
(詳圖見下頁)

```
#define IOINFO_ARRAYS 64
① _CRTIMP ioinfo * __pioinfo[IOINFO_ARRAYS];
```

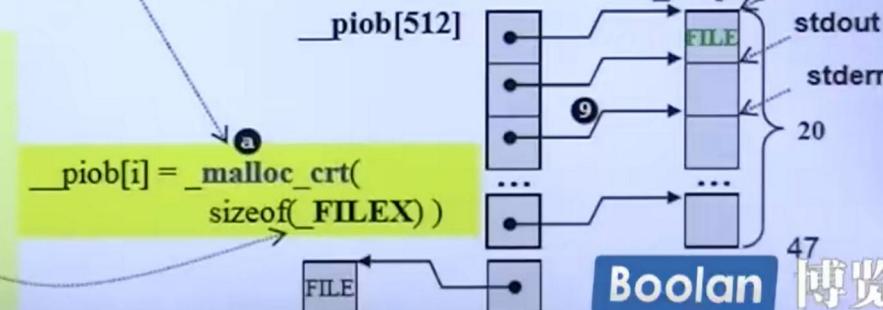
- CRT (that is in user mode) 維護一個靜態的 opened file table [64]，其內皆為 0，爾後依需要
- ② 每次動態分配 32 個 struct **ioinfo**. **\_malloc\_crt( 32 \* sizeof(ioinfo) )**
- **\_ioinit()** 經由 **GetStartupInfo()** 取得 inherited opened file handles，把它們逐一拷貝到 struct **ioinfo** 內 (在 text mode 之下前三個 opened files 必是 stdin, stdout, stderr)
- CRT 維護一個靜態的 **\_iob [20]**，元素都是 **FILE**，前三元素是：
- **FILE** 中的 **int \_file** 就是 index of opened file table，其最高 5 bits 是第二維索引，後 6 bits 是第一維索引。
- **\_initstdio()** 動態分配 **\_piob[512]**，前 20 個元素 (ptrs) 指向 **\_iob[20]** 中的對應元素
- **fopen()** 首先在 **\_piob[512]** 中找出一個 free entry, 若找不到就分配一個 new entry.
- 接下來找出 arrays of **ioinfo** 中的一個 free entry，並將 fh (file handle) 設正確 (如 ⑥ ⑦ 所言)  
這個 fh 將賦值給 **a FILE** 中的 **int \_file**.
- 然後呼叫 WinAPI **CreateFile()** 獲得 osfh (OS file handle)  
這個 osfh 將賦值給 **b** 中的 **long osfhnd**

```
#ifdef _MT
typedef struct {
    FILE f;
    CRITICAL_SECTION lock;
} _FILEX;
#else /* _MT */
typedef FILE _FILEX;
#endif /* _MT */
```

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



**\_piob = (void \*\*)\_calloc\_crt( 512, sizeof(void \*) )**

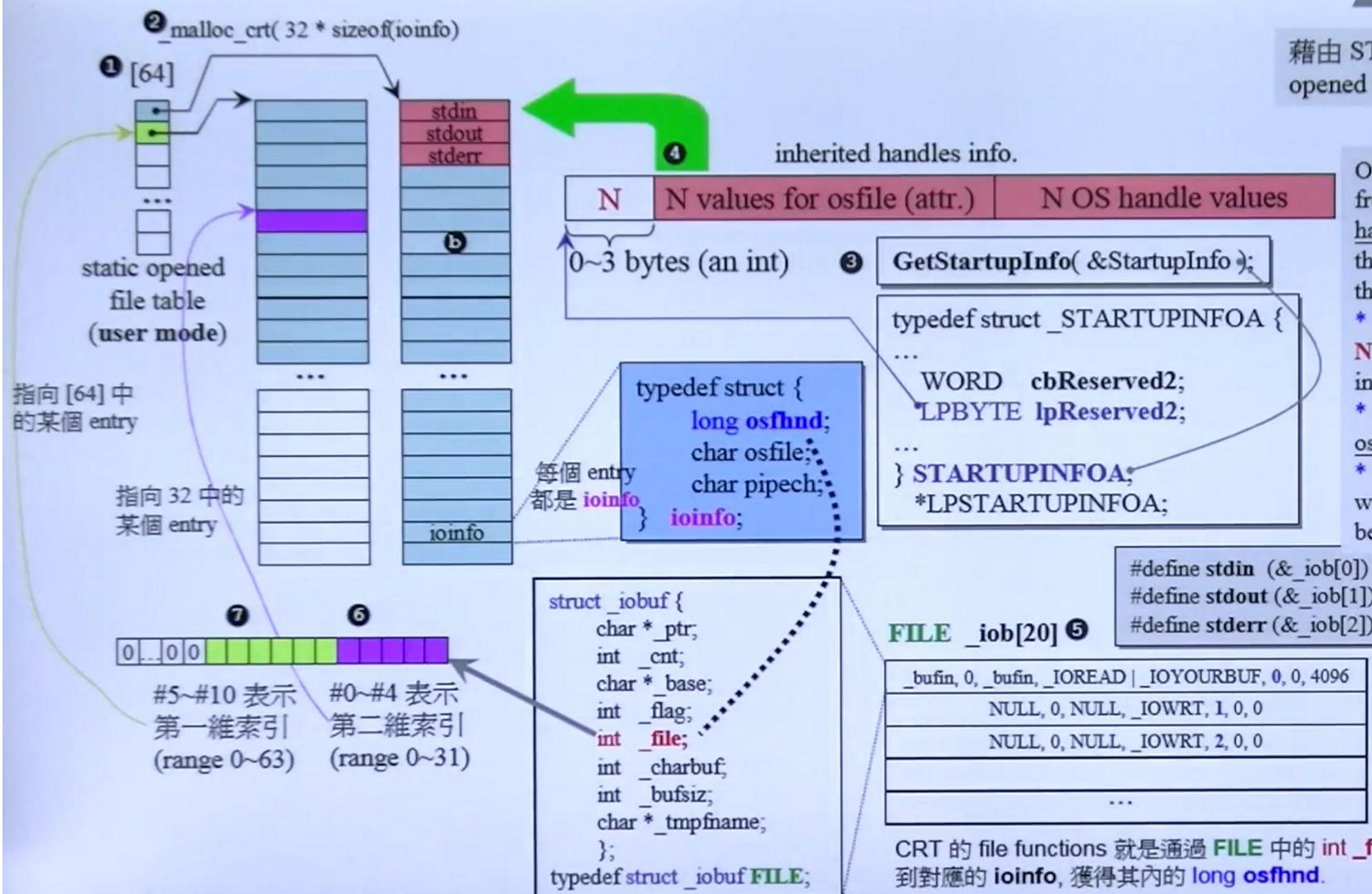


47 Boolean 博覽

## 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



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:

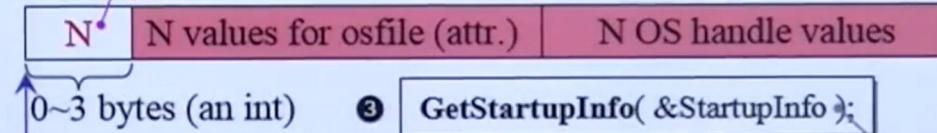
- \* bytes 0 thru 3 - integer value, say **N**, which is the number of handles information is passed about
- \* bytes 4 thru **N+3** - the N values for osfile

- \* bytes **N+4** thru **5\*N+3** - N double-words, the N OS HANDLE values being passed

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



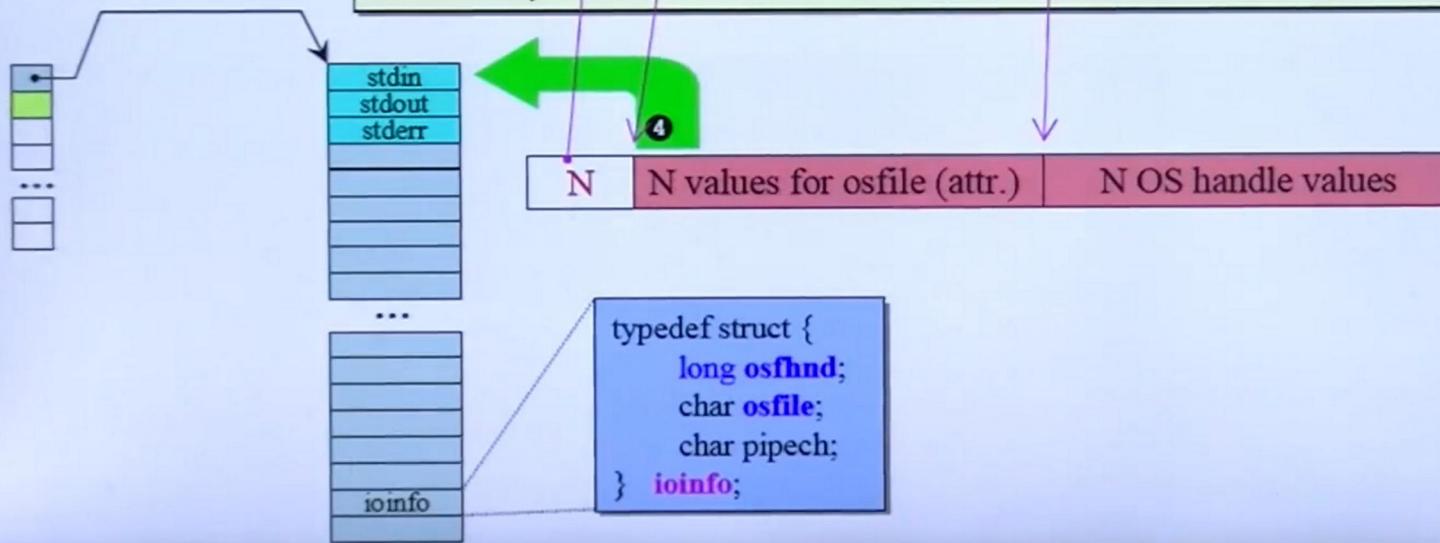
```
#0080 /* * Process inherited file handle information, if any */
#0081 GetStartupInfo( &StartupInfo );
#0082 if ( (StartupInfo.cbReserved2 != 0) &&
#0083     (StartupInfo.lpReserved2 != NULL) )
#0084 {
#0085     /* * Get the number of handles inherited. */
#0086     cfi_len = *(UNALIGNED int *) (StartupInfo.lpReserved2);
#0087
#0088     /* * Set pointers to the start of the passed file info and OS
#0089      * HANDLE values. */
#0090     posfile = (char *) (StartupInfo.lpReserved2) + sizeof( int );
#0091     posfhnd = (UNALIGNED long *) (posfile + cfi_len);
```



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

```

#0119     for ( fh = 0 ; fh < cfi_len ; fh++, posfile++, posfhnd++ ) {
#0120         /*
#0121         ...
#0122         */
#0123         if( (*posfhnd != (long)INVALID_HANDLE_VALUE) &&
#0124             (*posfile & FOPEN) &&
#0125             ((*posfile & FPIPE) ||
#0126              (GetFileType( (HANDLE)*posfhnd ) != FILE_TYPE_UNKNOWN)) )
#0127         {
#0128             pio = _pioinfo( fh );
#0129             pio->osfhnd = *posfhnd;
#0130             pio->osfile = *posfile;
#0131         }
#0132     }
#0133 }
#0134 }
#0135 }
#0136 }
#0137 }
#0138 }
```



```
#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!

sizeof(FILE): 0x20

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

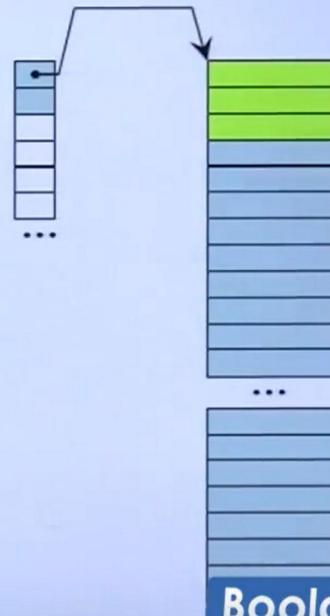
所以 stdin, stdout, stderr (都是指針) 相差 0x20

```
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 分別為 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:

\*

\*\*\*\*\*