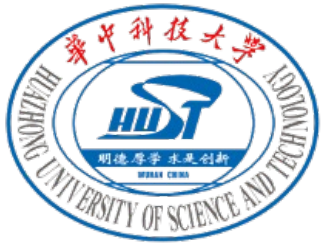


华中科技大学
HUAZHONG UNIVERSITY OF SCIENCE AND TECHNOLOGY

网络空间安全学院

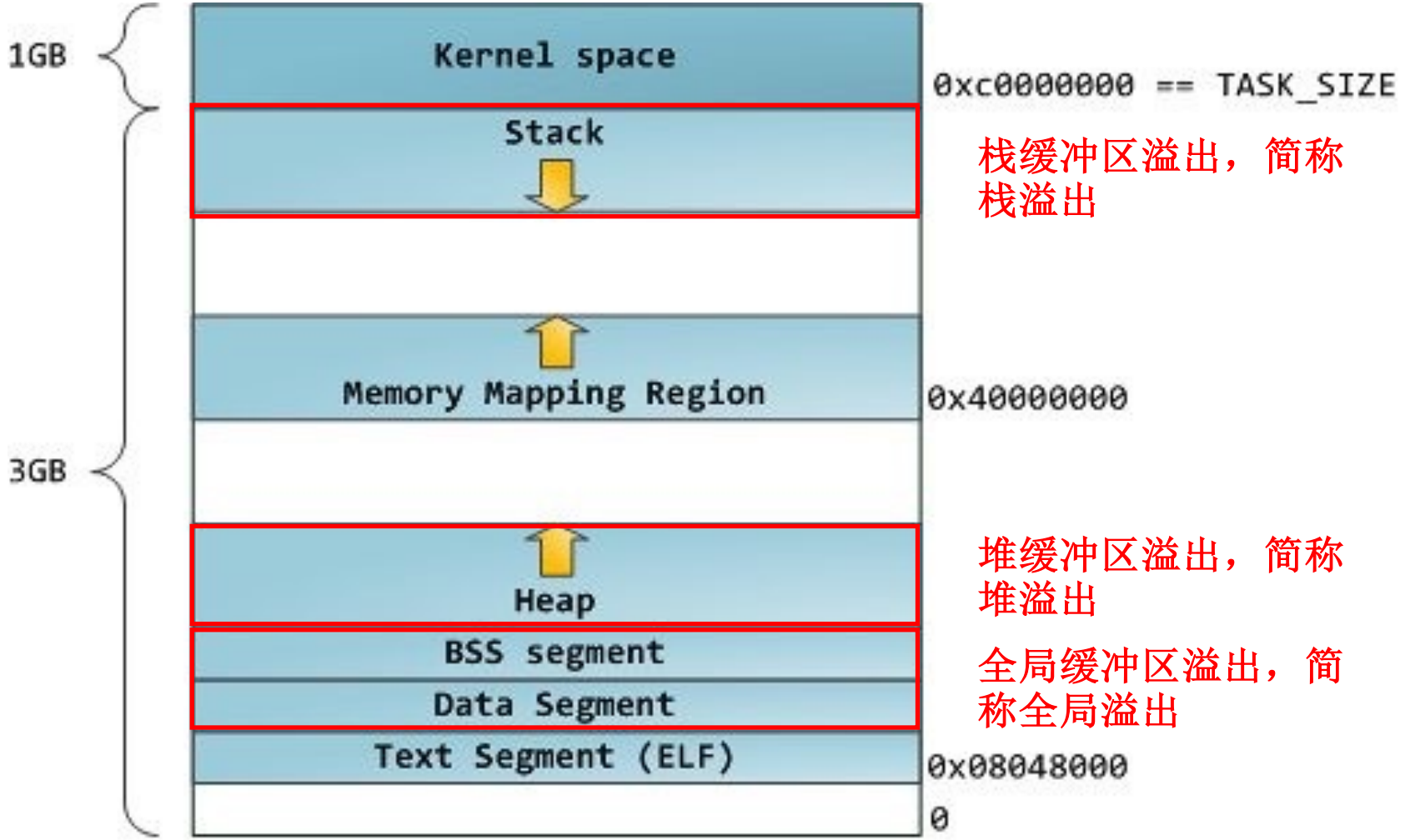
4.2缓冲区溢出之堆溢出

网络空间安全学院 慕冬亮

Email : dzm91@hust.edu.cn

[1] Once upon a free()...

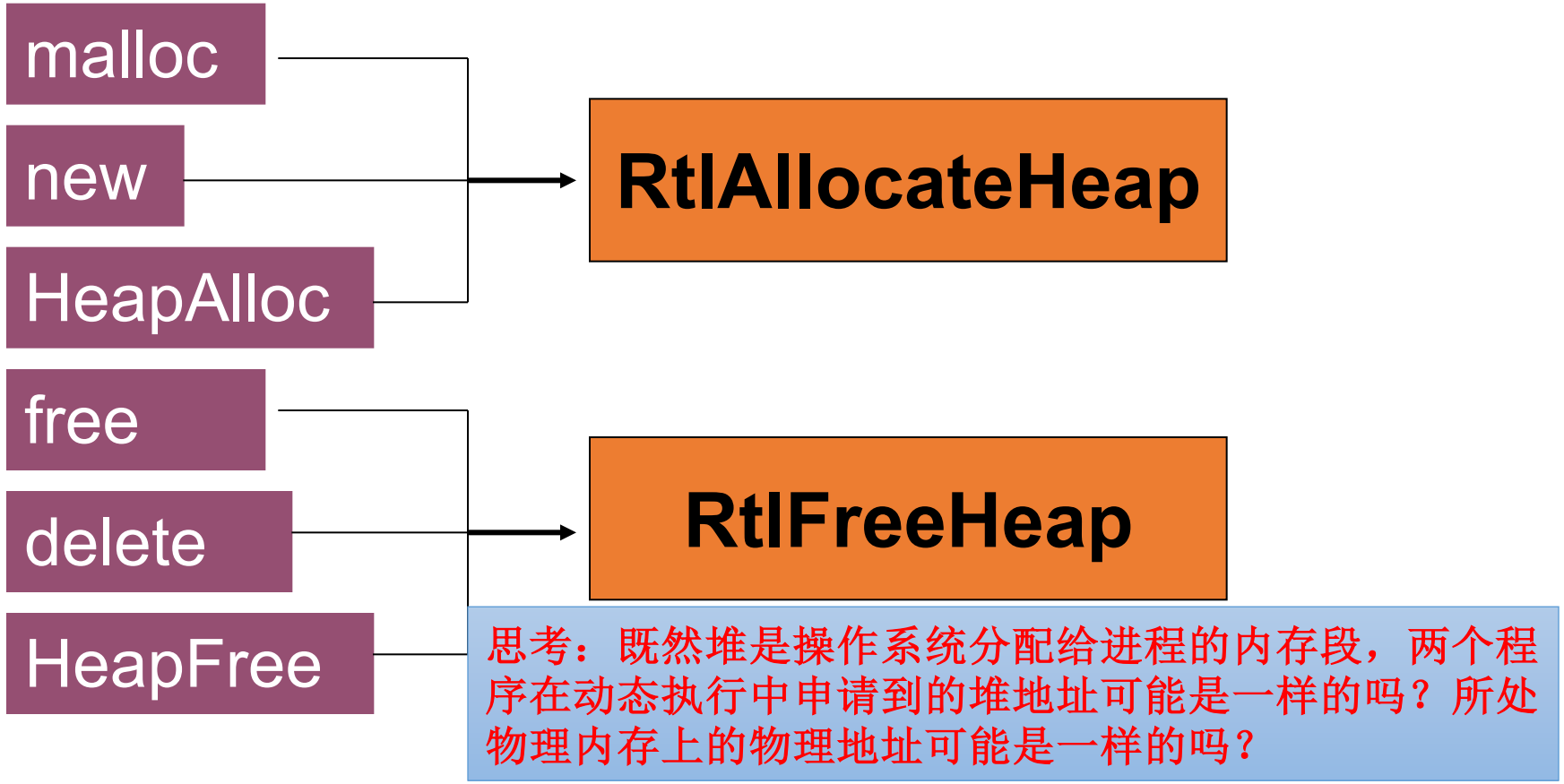
进程地址空间分布


1GB
3GB
Kernel space
0xc0000000 == TASK_SIZE
Stack
栈缓冲区溢出，简称栈溢出
Memory Mapping Region
0x40000000
Heap
堆缓冲区溢出，简称堆溢出
BSS segment
Data Segment
全局缓冲区溢出，简称全局溢出
Text Segment (ELF)
0x08048000
0

1.堆定义

- 堆（Heap）是用于存放程序运行中请求操作系统分配给自己的内存段
  - 大小并不固定，可动态扩张或缩减
  - 操作系统采用动态链表管理
  - 内存不一定连续
- 每一个进程有自己的堆
  - 提供一个进程生命周期存放数据的区域
  - 默认堆与私有堆
- 用new/malloc/HeapAlloc…指令来申请堆空间
- 用delete/free/HeapFree…指令释放堆内存

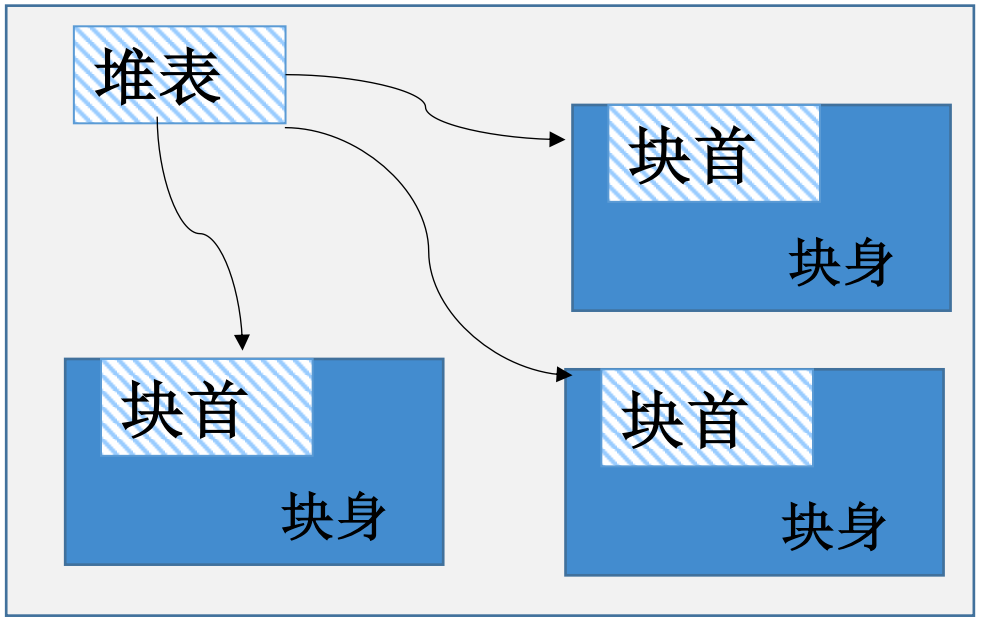
2.堆操作



- 堆操作最终都转化为 RtlAllocateHeap/RtlFreeHeap

3.堆的数据结构与管理

- 堆结构：堆表+堆块
  - 堆表
    - 位于堆区起始位置,用于索引堆区中所有堆块的重要信息
    - 分为两类：快表与空表
  - 堆块
    - 块首+块身



堆表

- 堆表有很多种，且随OS的不同而不同，Windows常见的有：
  - FreeList（空表）
  - Lookaside（快表）

两类重要堆表：空表

- 空表
  - 有128项，每项标识指定大小的空闲块
  - 空闲块大小=索引项（ID）*8
  - Free[0]标识大于等于1024 Byte的空闲块
  - 双向链表

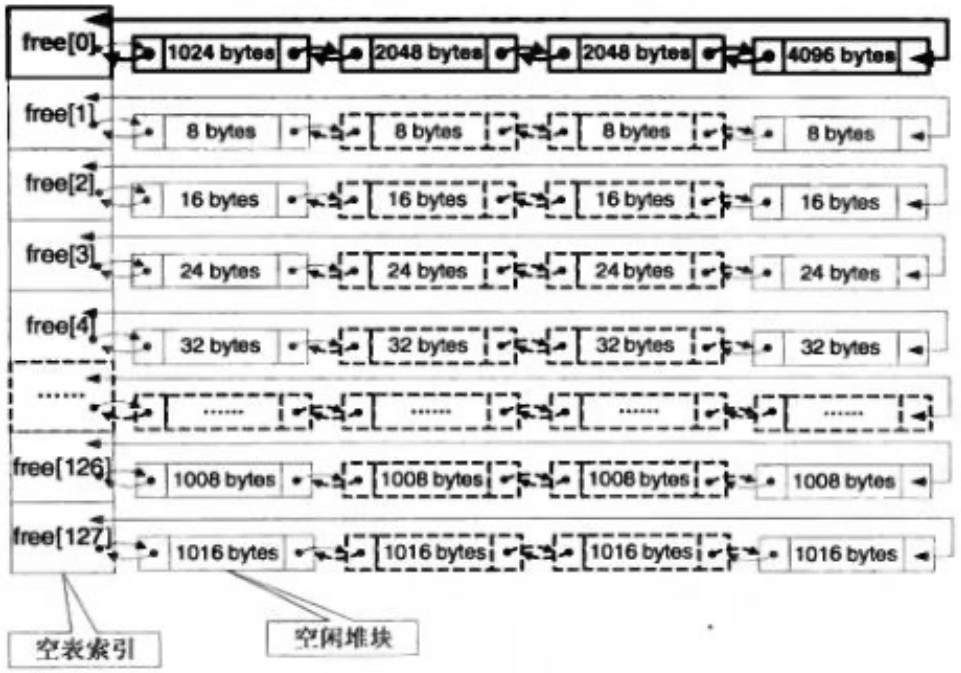


图 5.1.2　空闲双向链表（Freelist）

两类重要堆表：快表

- 快表
  - 128项
  - 采用单向链表
  - 链中的堆从不发生合并
  - 每项最多4个节点

两类重要堆表：快表

- 快表:快速单向链表Lookaside
  - 单向链表,128条
  - 目的:加快堆块分配
  - 不进行堆块合并
  - 初始化为空,每条快表最多4个节点
  - 1-127项:以8字节空间为基础,递增至1016字节

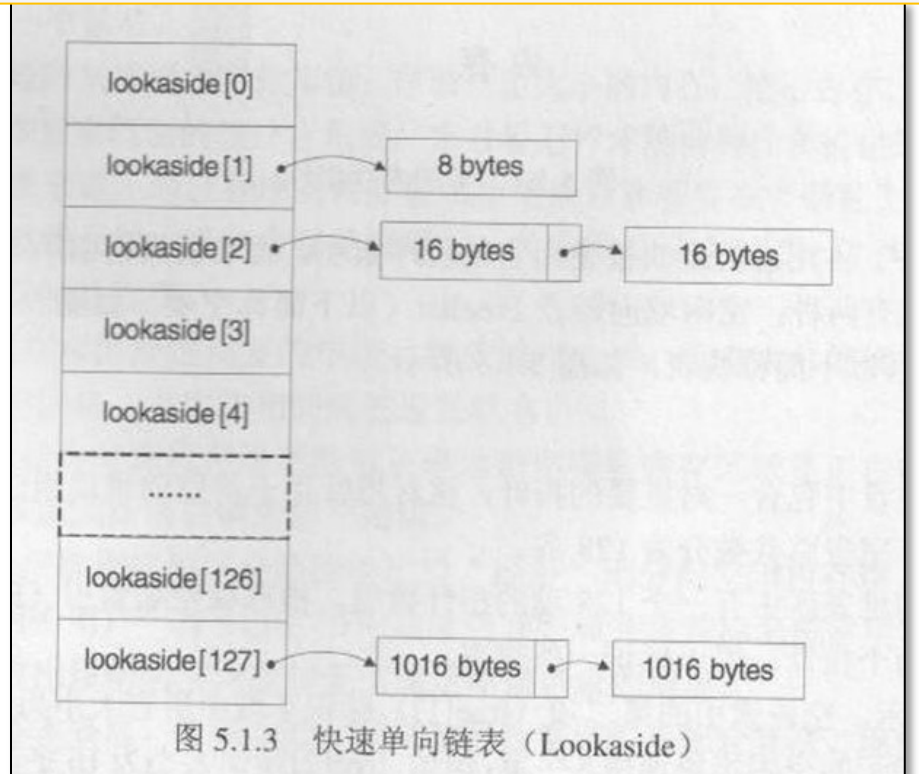
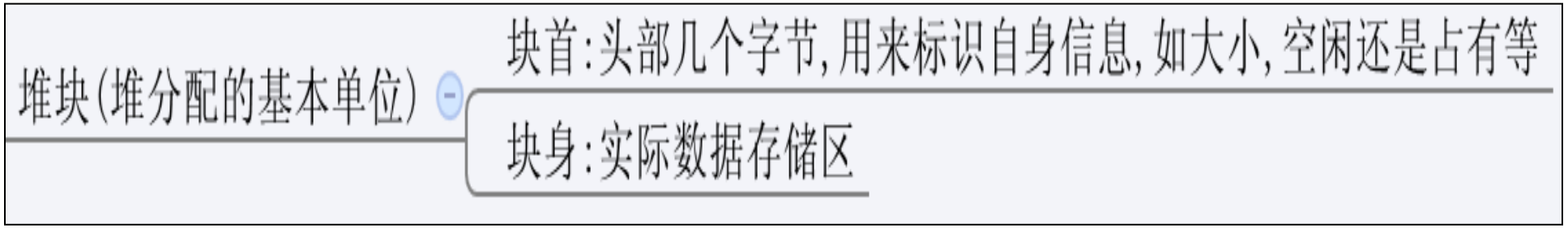


图 5.1.3 快速单向链表（Lookaside）

堆块

- 块首
  - 头部8个字节,用来标识自身信息（如大小,空闲还是占有等）
- 块身
  - 数据存储区域，紧跟块首



占用态vs空闲态 堆块结构

占用态:

Self Size	Previous Chunk size	Segment Index	Flags	Unused bytes	Tag Index (Debug)
Data					

空闲态:

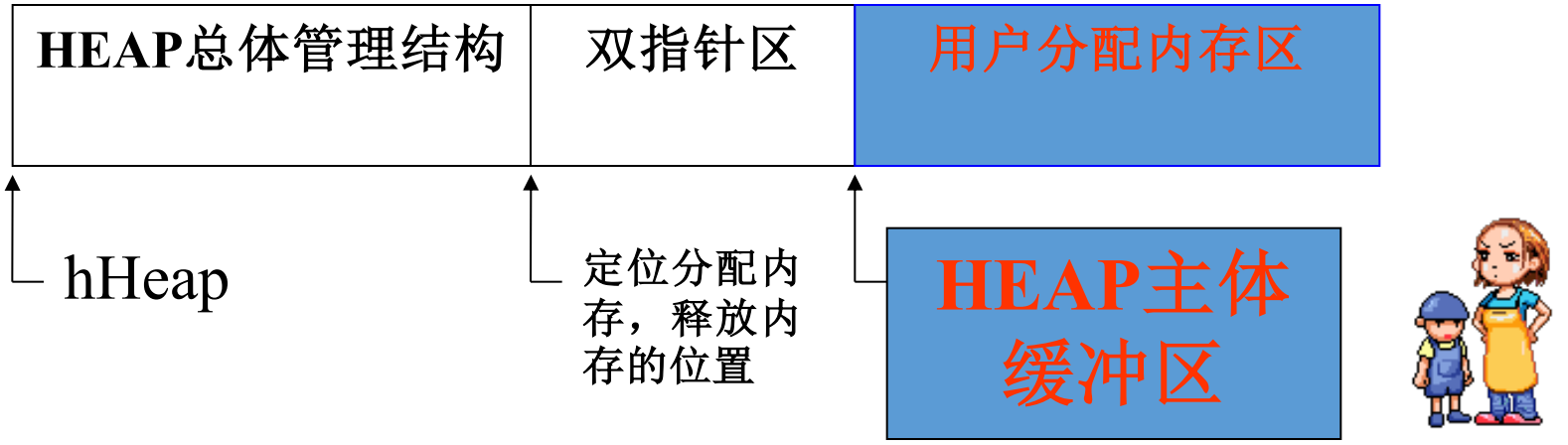
占用态与空闲态堆块完全可能连接在一起!

Self Size	Previous Chunk size	Segment Index	Flags	Unused bytes	Tag Index (Debug)
Flink(前向指针)		**Blink(后向指针)**			
Data					

思考：为什么占用态不用前后向指针？

Windows堆结构

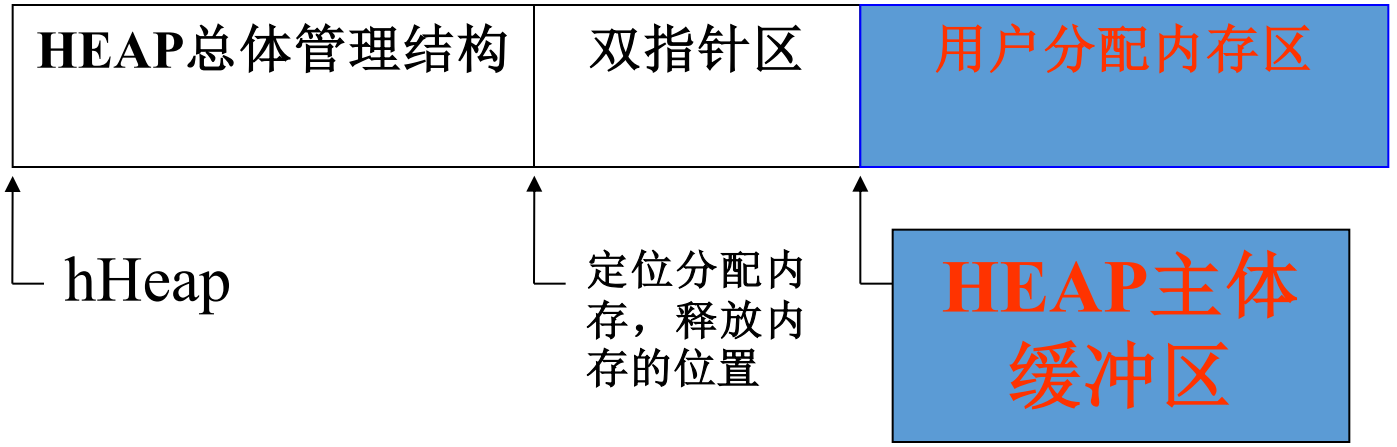
每个HEAP区的结构



- 对于一个进程来说可以有多个HEAP区，每一个HEAP的首地址以句柄hHeap来表示，这也是RtlAllocateHeap的第一个参数
- heap总体管理结构区存放着一些用于HEAP总体管理的结构，该结构与溢出无关
- 双指针区存放着一些成对出现的指针，用于定位分配内存以及释放内存的位置

Windows堆结构

如果有堆溢出…

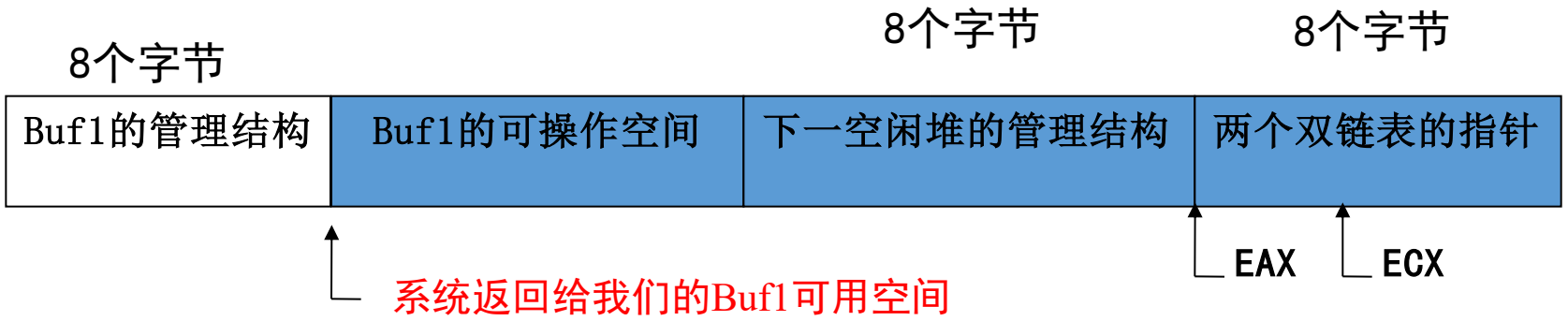


- buf1 =(char*)HeapAlloc(hHeap, 0, 16);
- memcpy(buf1,mybuf,32);
- buf2 =(char*)HeapAlloc(hHeap, 0, 16);

Windows堆结构：用户内存区 1

Ntdll.dll中的RtlAllocateHeap来分配堆

第一次：Buf1=HeapAlloc(hHeap,0,16);

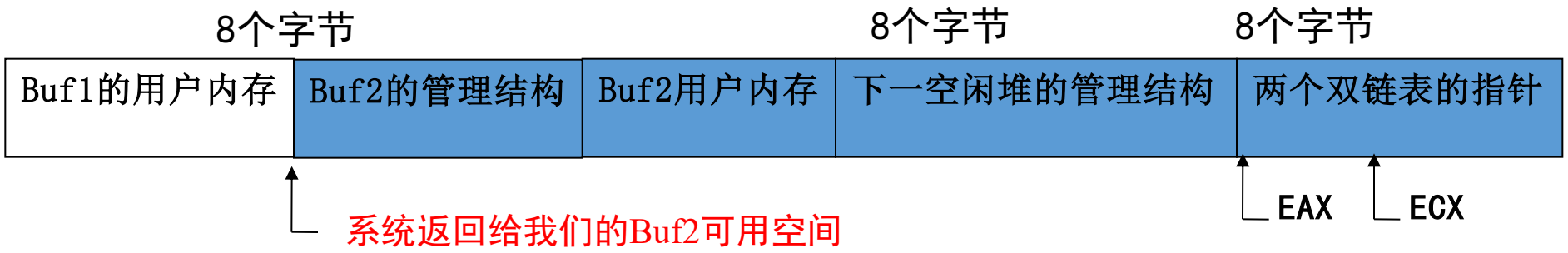


- 系统实际多申请8字节管理空间
- 两个指针将空闲块连接起来

Windows堆结构：用户内存区 2

Ntdll.dll中的RtlAllocateHeap来分配堆

第二次：Buf2=HeapAlloc(hHeap,0,16);

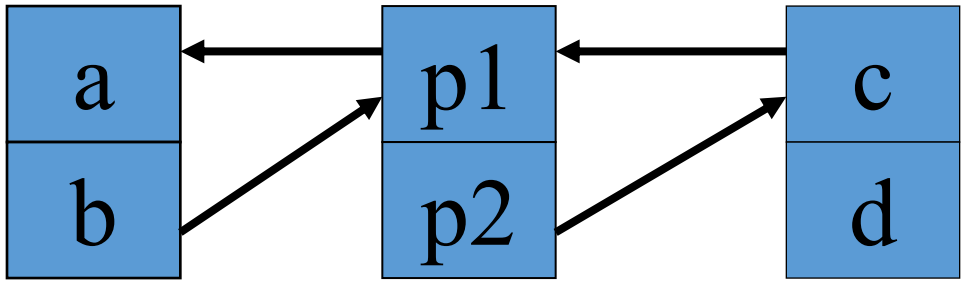


- Buf1后面是Buf2管理结构
- Buf2后面是空闲堆管理结构，然后是链表指针

申请Buf2,会发生什么？

堆块操作进一步示意图

双向链表的删除操作

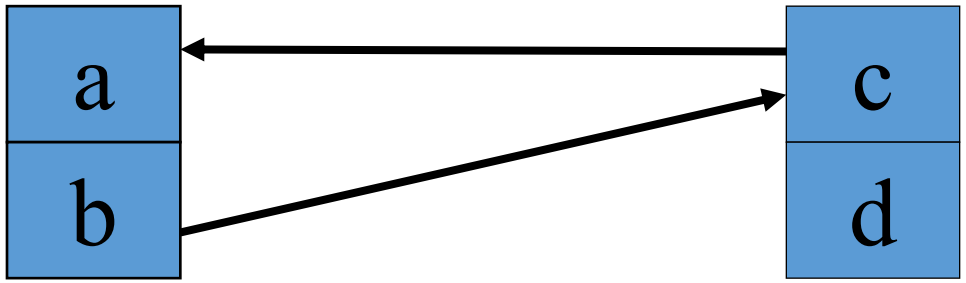


其中，a，b，c，d，p1，p2 表示对应内存空间的值

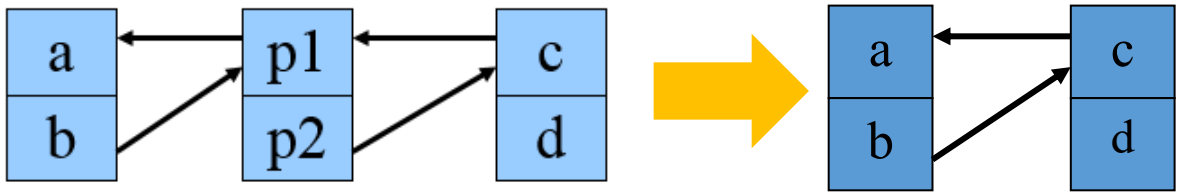
a = *p1, b = *(p1+1), c = *p2, d = *(p2+1)

申请使用p1、p2 所在的空闲堆块，或者

如果是使用内存链表，释放p1、p2指向的内存



堆块操作进一步示意图2




```
c = &a
b = &c
```

```
a = *p1
b = *(p1+1)
c = *p2
d = *(p2+1)
```

```
*p2=p1
*(p1+1)=p2
```

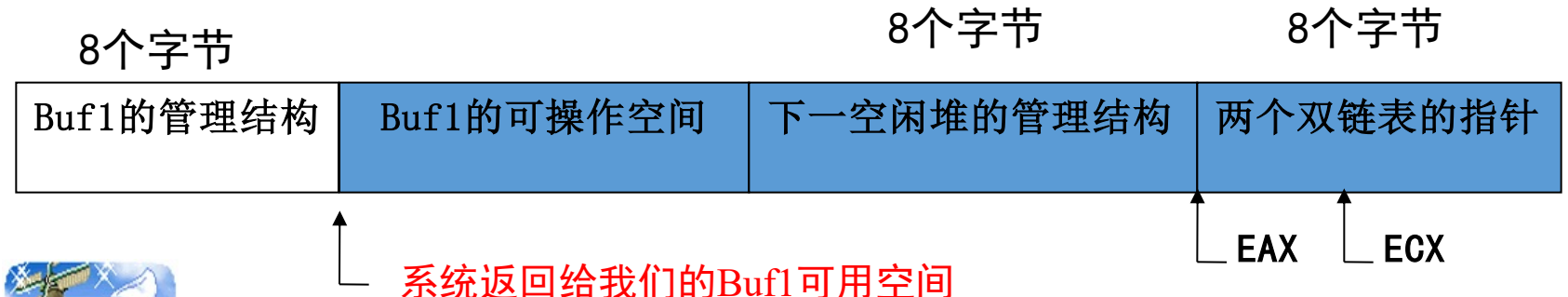
p1 p2
指链表里指的内容

在堆分配和堆回收时利用的具体的实现指令

```
MOV [ECX],EAX
MOV [EAX+4],ECX
```

- 只需要能够控制ECX EAX的值，就可以实现ShellCode

4.堆溢出与利用



- 分配完buf1之后向其中拷贝内容，拷贝的内容大小超过buf1的大小，即16字节，就会发生溢出，如果覆盖掉两个4字节的指针，而下一次分配buf2之前又没有把buf1释放掉的话，就会把一个4字节的内容写入一个地址当中，而这个内容和地址都是能够控制的，这样就可以控制函数的流程转向shellcode。

漏洞界称之为what→where操作
or Dword Shoot

堆溢出利用思路

- 利用 MOV [ECX],EAX
  MOV [EAX+4]， ECX
  完成任意地址任意值的控制
- 利用ESI+0x4c 指向下一个空闲块头部结构

8个字节		8个字节	8个字节
Buf1的管理结构	Buf1的可操作空间	下一空闲堆的管理结构	两个双链表的指针

当有不能处理的异常发生时，系统调用UnhandledExceptionFilter函数，它其实就是call [0x77EC044c]，即执行0x77EC044c指向的异常处理程序.

可以把where赋成0x77EC044c, what覆盖成ShellCode的地址

总结

- 堆漏洞利用

下一节继续...