2019-2020学年秋季学期

漏洞利用与攻防实践

Exploiting Software Vulnerability-Techniques and Practice

授课教师：霍玮

助　　教：邹燕燕

漏洞利用与攻防实践

Exploiting Software Vulnerability-Techniques and Practice

[第一次课] 课程引言

授课教师：霍玮

授课时间：2019-9-3

课程大纲

课程大纲

首席教师

- 霍玮(huowei@iie.ac.cn)
  - 国科大网安学院教师，中科院信工所研究员，博导
  - 中科院青促会成员
  - 软件安全理论及分析技术
  - 带领团队研制国内先进的漏洞分析平台VARAS，挖掘近百个零日漏洞，申请CVE编号100余个

课程助教

- 邹燕燕(zouyanyan@iie.ac.cn)
  - 中科院信工所助理研究员，在职博士生
  - 毕业于中国科学技术大学，获硕士学位
  - 模糊测试漏洞挖掘、程序分析

课程定位

漏洞发现与利用具体方法及实践

发现及分析漏洞的基础方法及技术

软件安全理论及概念方法

专业研讨课：漏洞利用与攻防实践 | 基于大数据的软件安全 | 面向虚拟化技术的攻防前沿问题研究 | 恶意代码前沿技术

专业普及课：软件安全漏洞分析与发现 | 网络攻防 | 移动安全与测评 | 恶意软件发现与分析

专业核心课：软件安全原理

课程定位


windows
A fatal exception OE has occurred at 0167:BFF9E463.
The current applications will be terminated.
*Press any key to terminate the current applications
*Press CTRL+ALT+DEL again to restart your computer. You will
lose any unsaved information in all application.
Press Enter to continue.


your important files are enc
Wana DecryptOr 2.0
Ooops, your files have been encrypted!
Chinese (simpl...
我的电脑出了什么问题？
您的一些重要文件被我加密保存了。
照片、图片、文档、压缩包、音频、视频文件、exe文件等，几乎所有类型的文件都被加密了，因此不能正常打开。
这和一般文件损坏有本质上的区别。您大可在网上找找恢复文件的方法，我敢保证，没有我们的解密服务，就算老天爷来了也不能恢复这些文档。
有没有恢复这些文档的方法？
当然有可恢复的方法。只能通过我们的解密服务才能恢复。我以人格担保，能够提供安全有效的恢复服务。
但这是收费的，也不能无限期的推迟。
请点击 <Decrypt> 按钮，就可以免费恢复一些文档。请您放心，我是绝不会骗你的。
但想要恢复全部文档，需要付款点费用。
是否随时都可以固定金额付款，就会恢复的吗，当然不是，推迟付款时间越长对你不利。
最好3天之内付款费用，过了三天费用就会翻倍。
还有，一个礼拜之内未付款，将会永远恢复不了。
对了，忘了告诉你，对半年以上没钱付款的穷人，会有活动免费恢复，能否轮
Payment will be raised on
5/16/2017 10:12:00
Time Left
02:23:47:24
Your files will be lost on
5/20/2017 10:12:00
Time Left
06:23:47:24
About bitcoin
How to buy bitcoins?
Contact Us
bitcoin
ACCEPTED HERE
Send $300 worth of bitcoin to this address:
12t9YDPgwueZ9NyMgw519p7AA8isjr6SMw
Copy
Check Payment
Decrypt
processha...
the instructions!


WHY?

课程定位

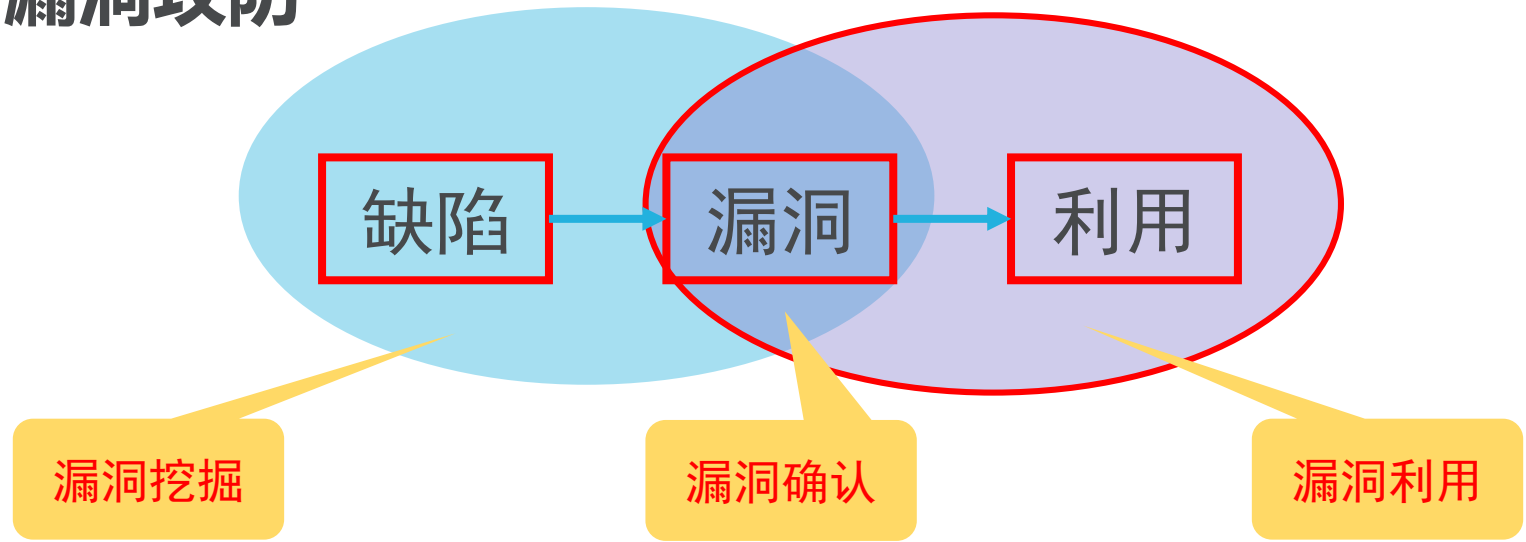
- 重点围绕内存破坏类漏洞的挖掘及控制流劫持攻防方法

破坏相对于什么？
为什么会破坏？
如何有效挖掘？

程序有哪几种控制流？
控制流劫持有哪几种方式？
控制流劫持的效果是什么？

核心概念

- 漏洞
  - 一个错误(mistake)如果可以被攻击者用于违反目标系统的一个合理的安全策略，它就是一个漏洞
- 漏洞攻防



课程组成-1

主题参考资料参见课程网站

源代码漏洞分析

基于中间表示的分析

- 数据流分析
- 符号执行
- 污点分析

基于逻辑推理的分析

- 模型检测
- 定理证明

二进制漏洞分析

二进制静态分析

- 基于模式的漏洞分析
- 基于二进制代码比对的漏洞分析

二进制动静结合分析

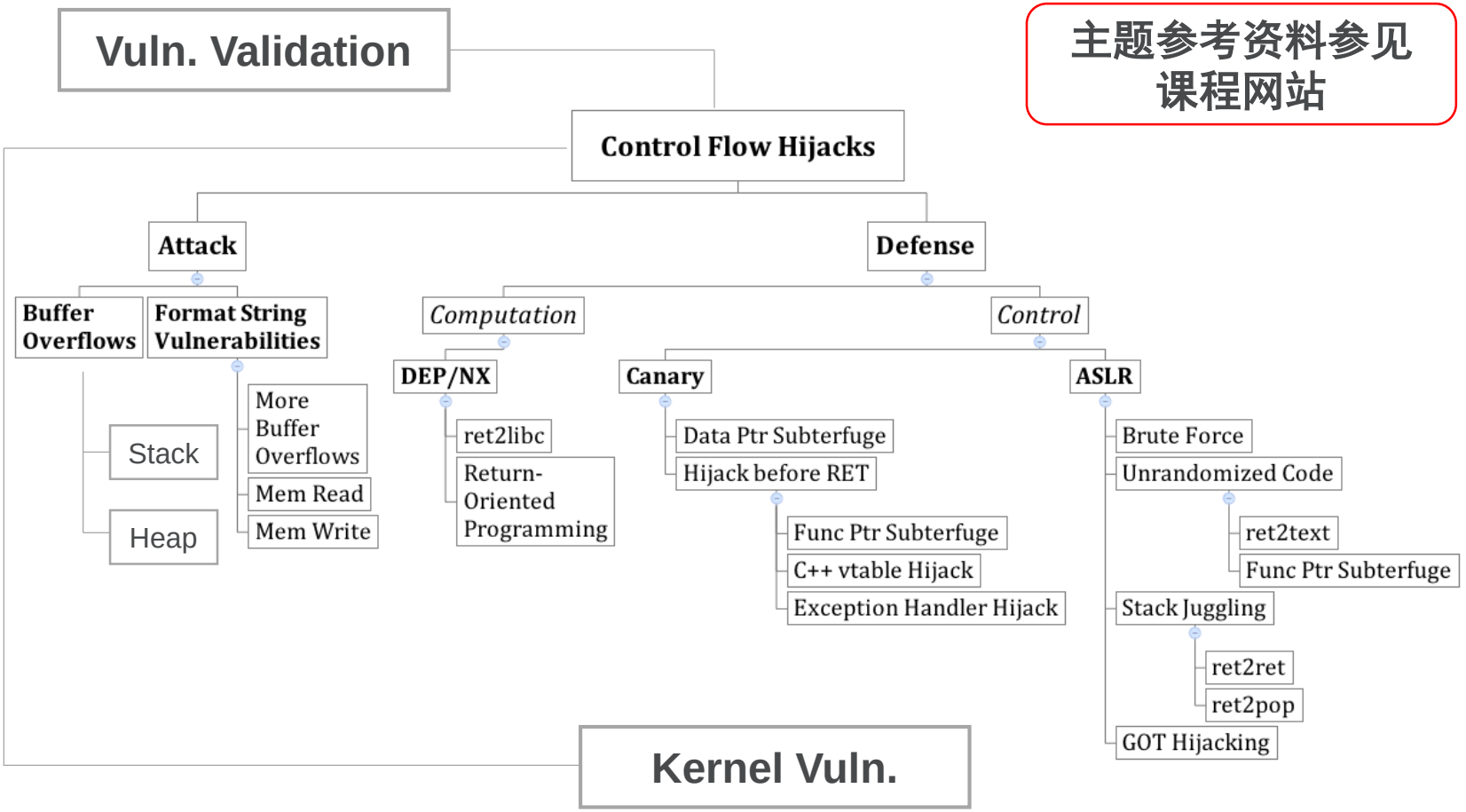
- 智能灰盒测试
- 动态污点分析

二进制动态分析

- 随机模糊测试
- 智能模糊测试

课程组成-2

主题参考资料参见
课程网站



课程安排

一	二	三	四	五	六	七	八	九	十	十一
课程概况	模糊测试基本方法	栈溢出及格式化字符串	堆溢出及其他内存破坏	代码重用方法	内核漏洞利用技术介绍	模糊测试高级研讨1	模糊测试高级研讨2	模糊测试高级研讨3	模糊测试高级研讨4	课程总结

讲授　　研讨+实践

课程要求

课程以分组报告和实验的形式共同学习讨论漏洞攻防的基础知识和基本原理，对漏洞挖掘及利用方法进行初步实践。

研讨方式

- 分成10组，每组学习并报告1个主题中的内容
- 每个主题分配参考文献，形成1个报告以及1个实验
- 每个报告内容由小组确定，报告45分钟，实验25分钟，报告人（2选1）及实验人（2选1）均当堂随机挑选，中间不得更换
- 分组及主题选择由助教课下组织

研讨要求

- 报告得分因素
  - 报告内容划分合理、PPT美观
  - 原理、方法介绍清楚直观易懂，提供参考文献之外内容
  - 回答问题准确合适
  - 报告举止得体大方，时间控制合理
- 实验得分因素
  - 现场演示实验过程及结果
  - 详细介绍实验原理及实现

课程要求

课程以分组报告和实验的形式共同学习讨论漏洞攻防的基础知识和基本原理，对漏洞挖掘及利用方法进行初步实践。

课程得分

- 报告分（满分50分）
- 实验分（满分25分）
- 互评分（满分20分）
- 考勤分（满分5分）
  - 多种形式考查出勤

得分方法

- 报告分和实验分由教师在结课后统一给出(百分制)
- 互评分由除报告小组外的全体同学匿名给出分数的平均分(百分制)

加分

- 提交内存破坏类漏洞利用报告及演示视频得1~10分
  - 已公开漏洞(CVE编号)的演示及报告得10分
  - 自己动手实践或复现，严禁拷贝！（不及格）
- 利用AFL/libfuzz等工具获得CVE编号，每个5分

课程大纲

一 课程总览

- 1.1 授课团队介绍
- 1.2 课程定位及内容
- 1.3 课程要求

二 背景知识

- 2.1 典型二进制文件格式
- 2.2 存储模型
- 2.3 执行模型

三 内容概述

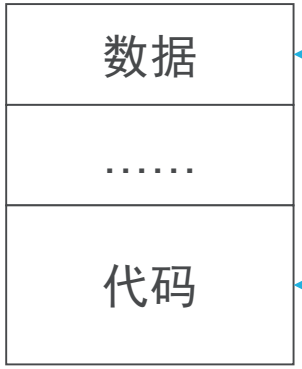
程序运行概览

当我们双击一个window桌面的快捷方式时，计算机内部都发生了什么

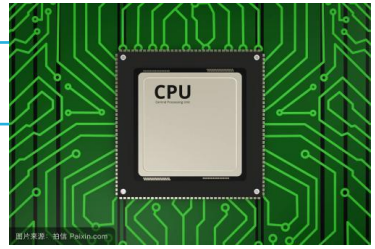
外存 —加载→ 内存 —运行→ 中央处理器

典型二进制格式

4G空间(32位)



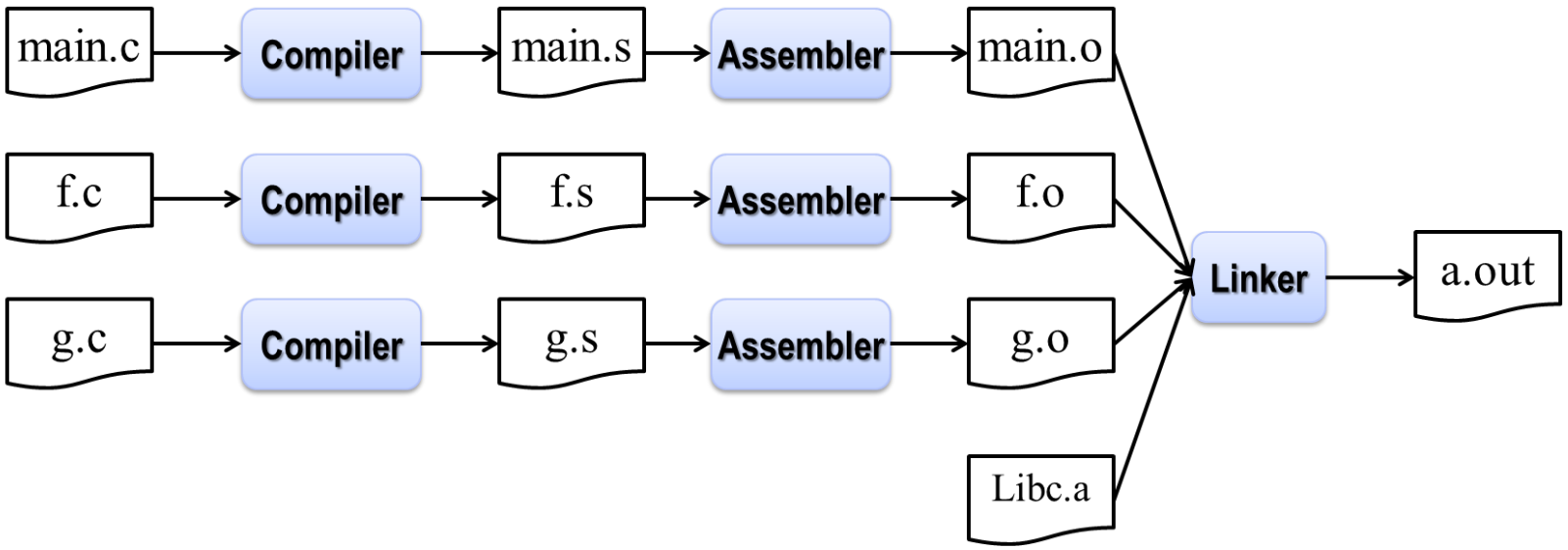
存储模型



执行模型

二进制文件的产生

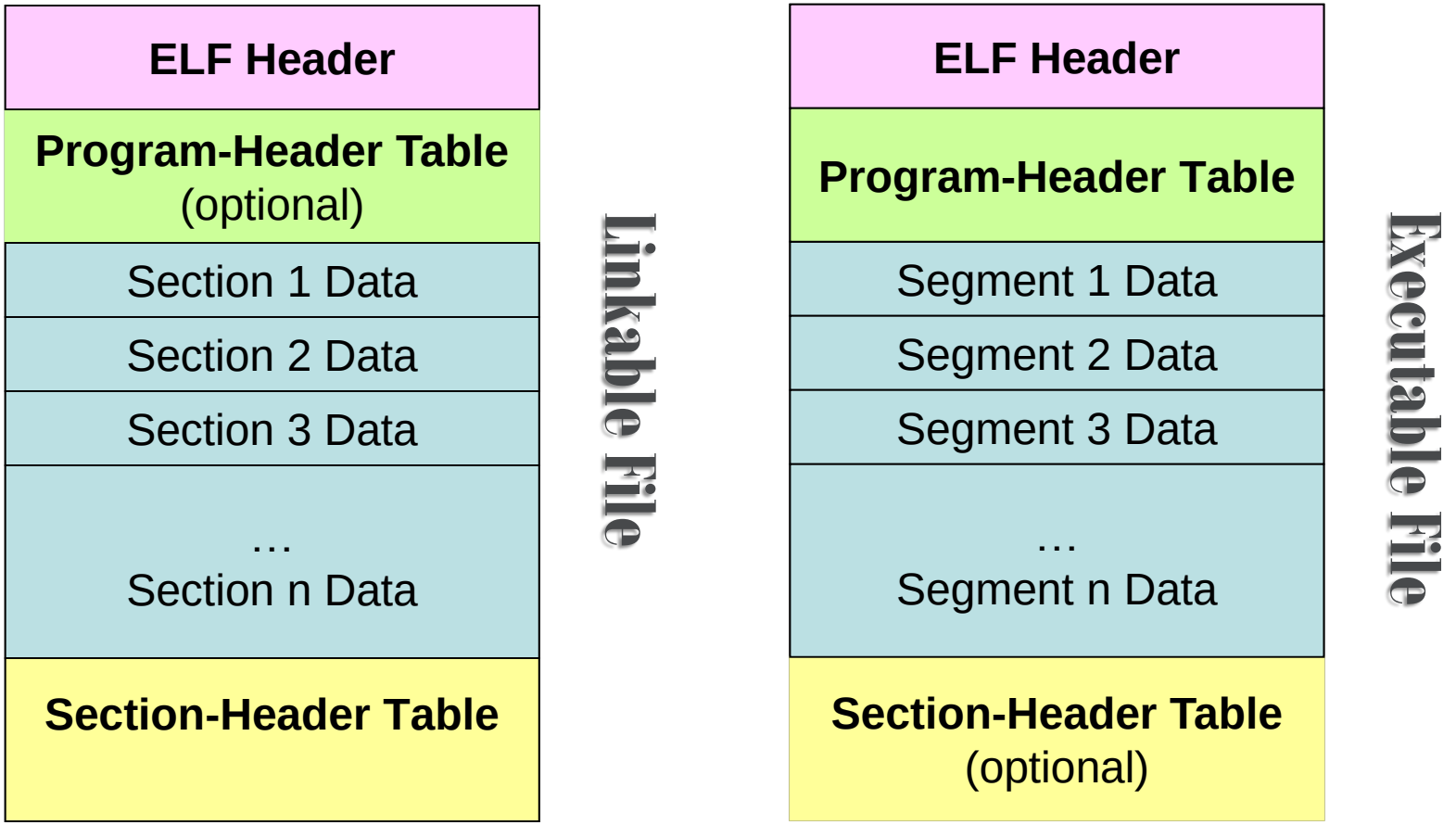
Linux平台使用**ELF**格式，Windows平台常使用**PE COFF**格式(简称PE)



Linux平台为例

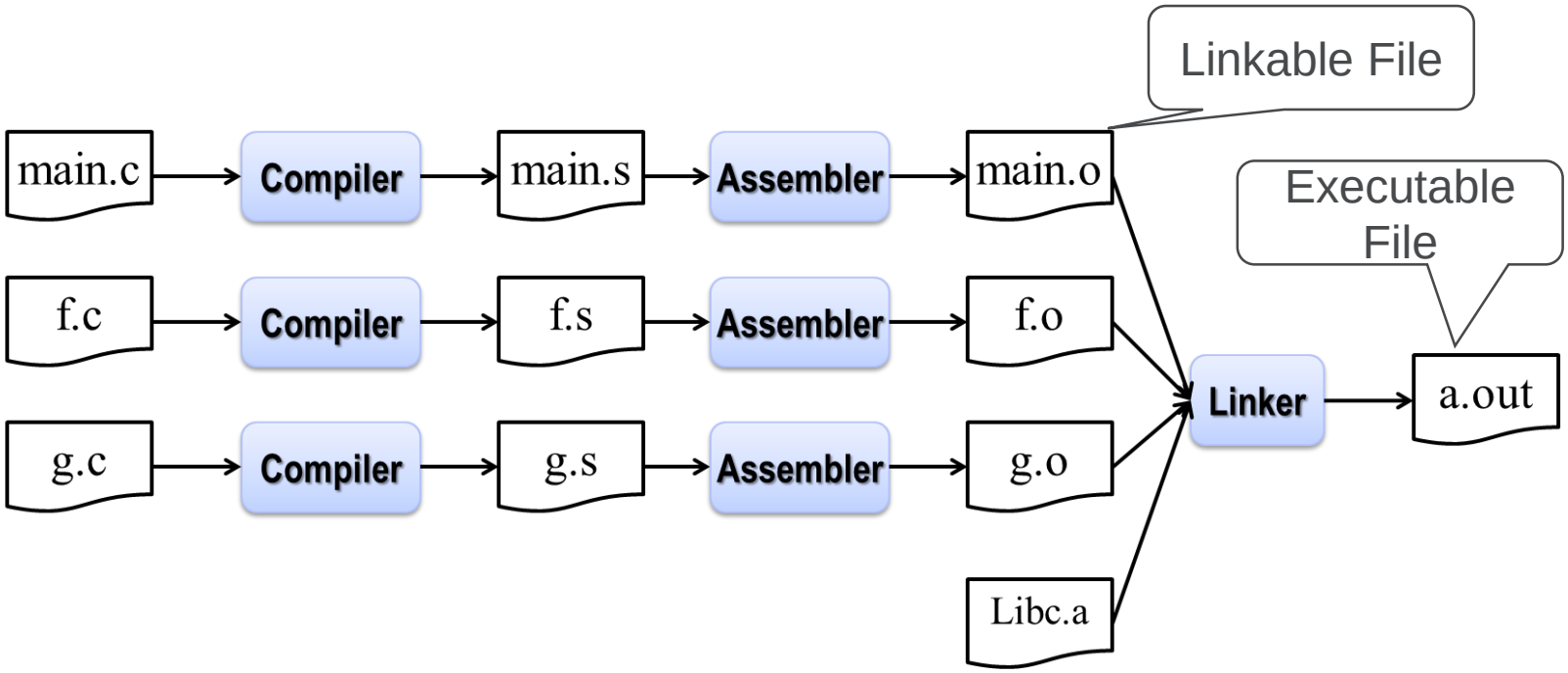
典型二进制格式-ELF

ELF = Executable and Linking File



典型二进制格式-ELF

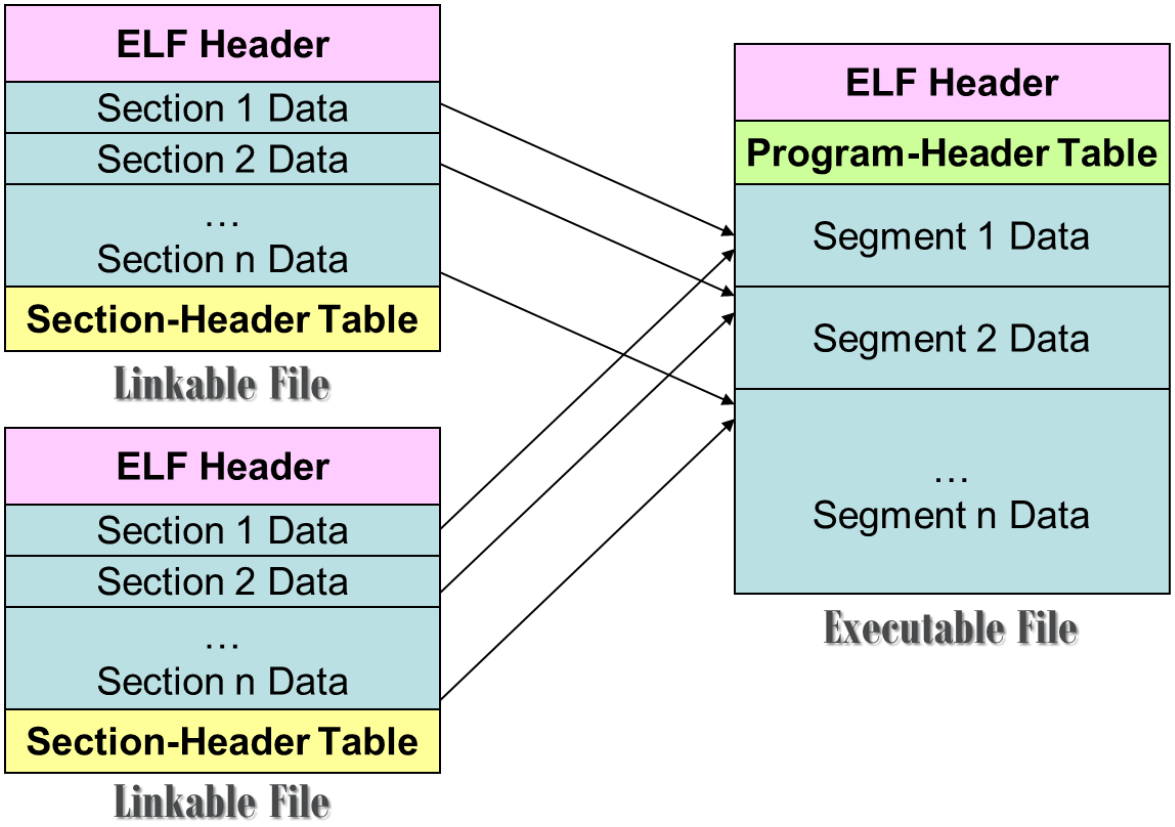
Linux平台使用**ELF**格式，Windows平台常使用**PE COFF**格式(简称PE)



Linux平台为例

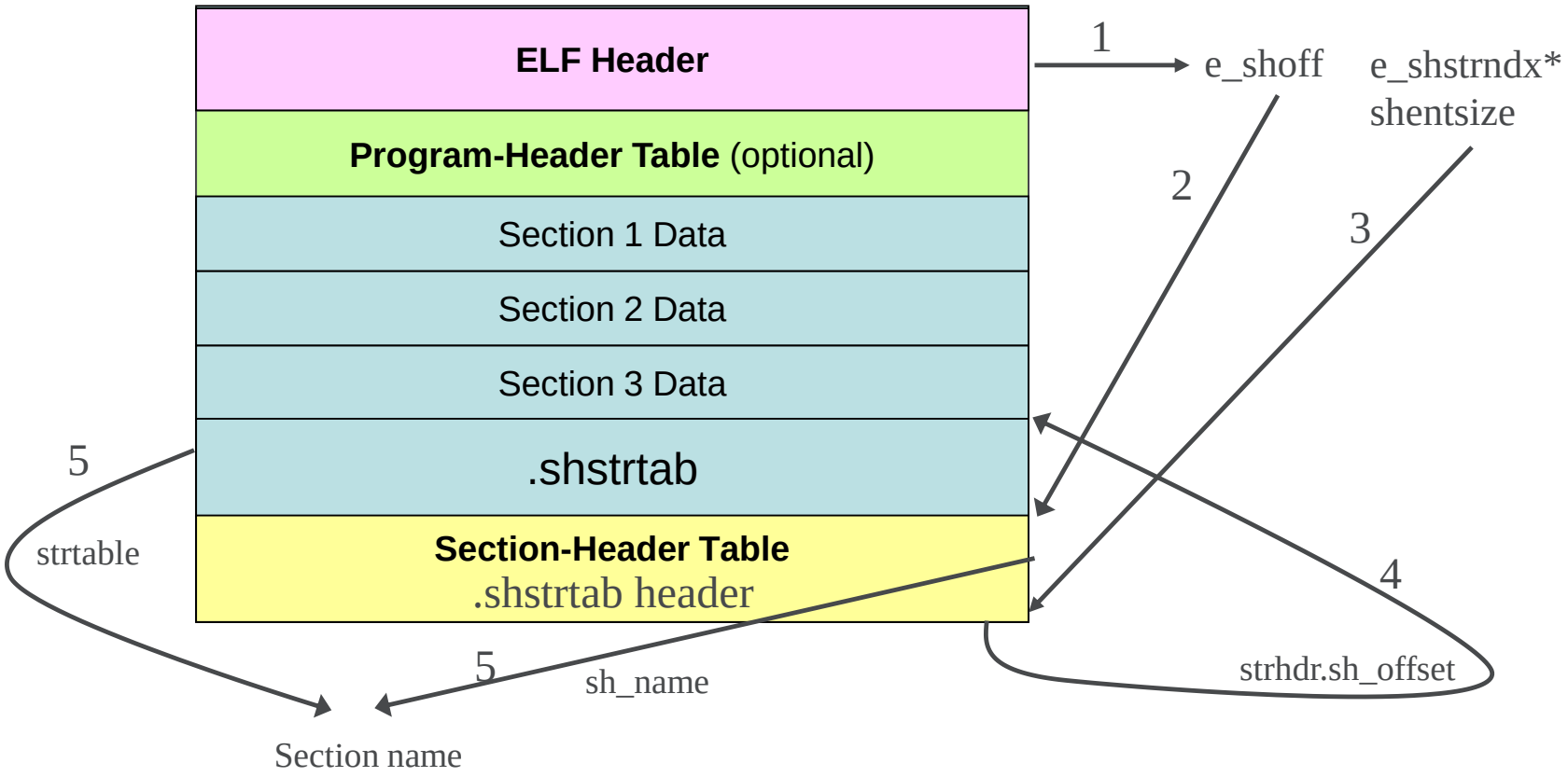
典型二进制格式-ELF

ELF = Executable and Linking File



典型二进制格式-ELF

ELF = Executable and Linking File



典型二进制格式-ELF

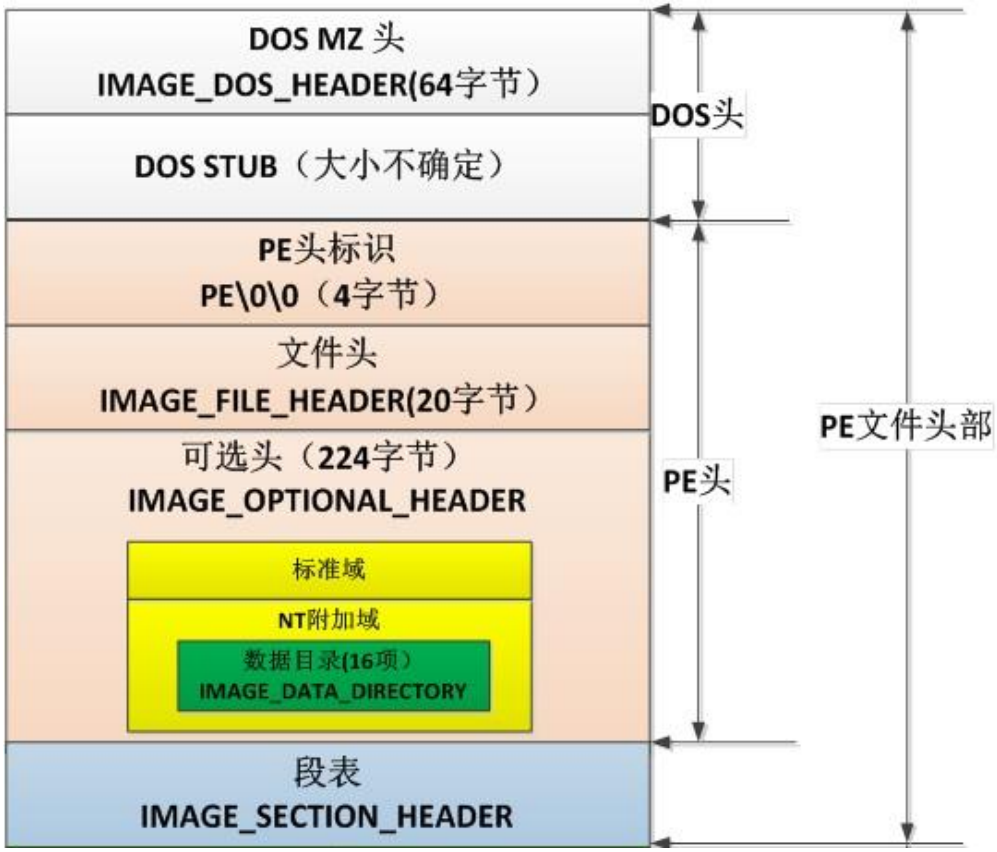
ELF = Executable and Linking File

Section	sh_type	sh_flags	含义
.bss	SHT_NOBITS	SHF_ALLOC +SHF_WRITE	包含将出现在程序内存映像的未初始化数据 当程序开始时，系统将这些数据初始化为0 不占用文件空间
.data	SHT_PROGBITS	SHF_ALLOC +SHF_WRITE	包含将出现在程序内存映像中已初始化数据 占用文件空间
.init	SHT_PROGBITS	SHF_ALLOC + SHF_EXECINSTR	进程初始化代码 程序在调用main函数前调用这些代码
.plt	SHT_PROGBITS		过程链接表
.rel	SHT_REL	SHF_ALLOC	重定位信息
.text	SHT_PROGBITS	SHF_ALLOC + SHF_EXECINSTR	代码段

Section Header

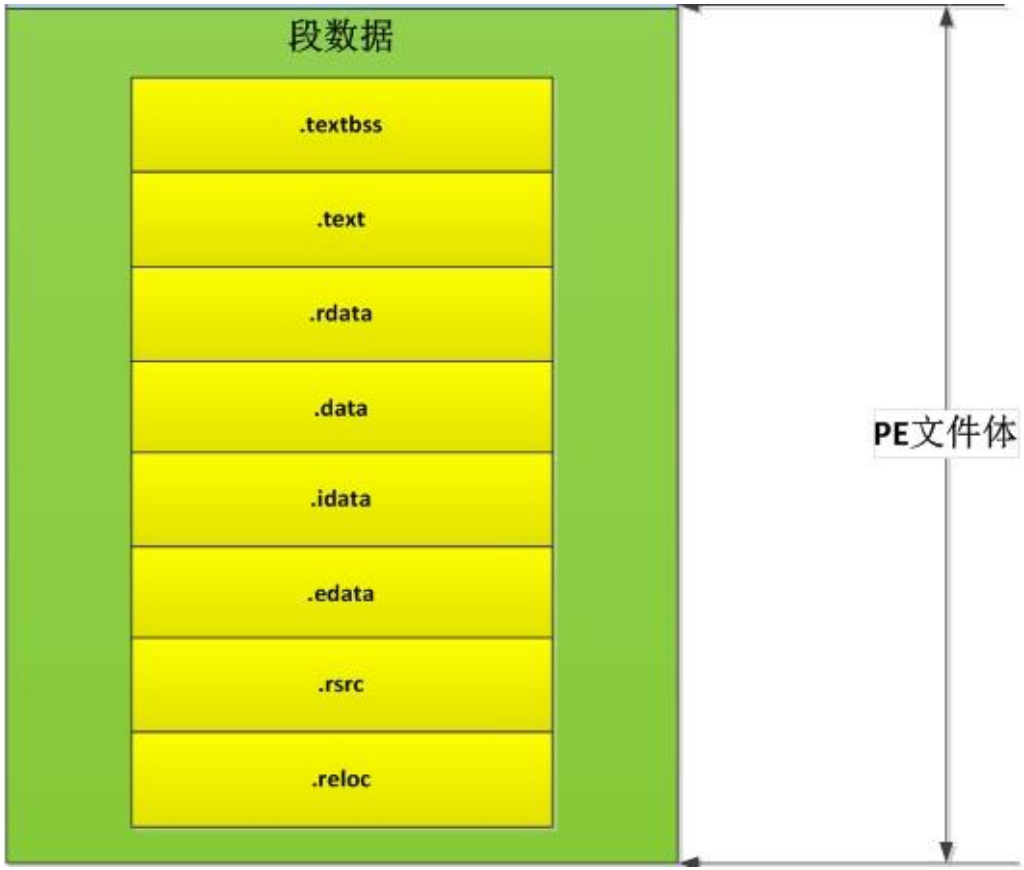
典型二进制格式-PE(1)

PE COFF=Portable Executable Common Object Format File

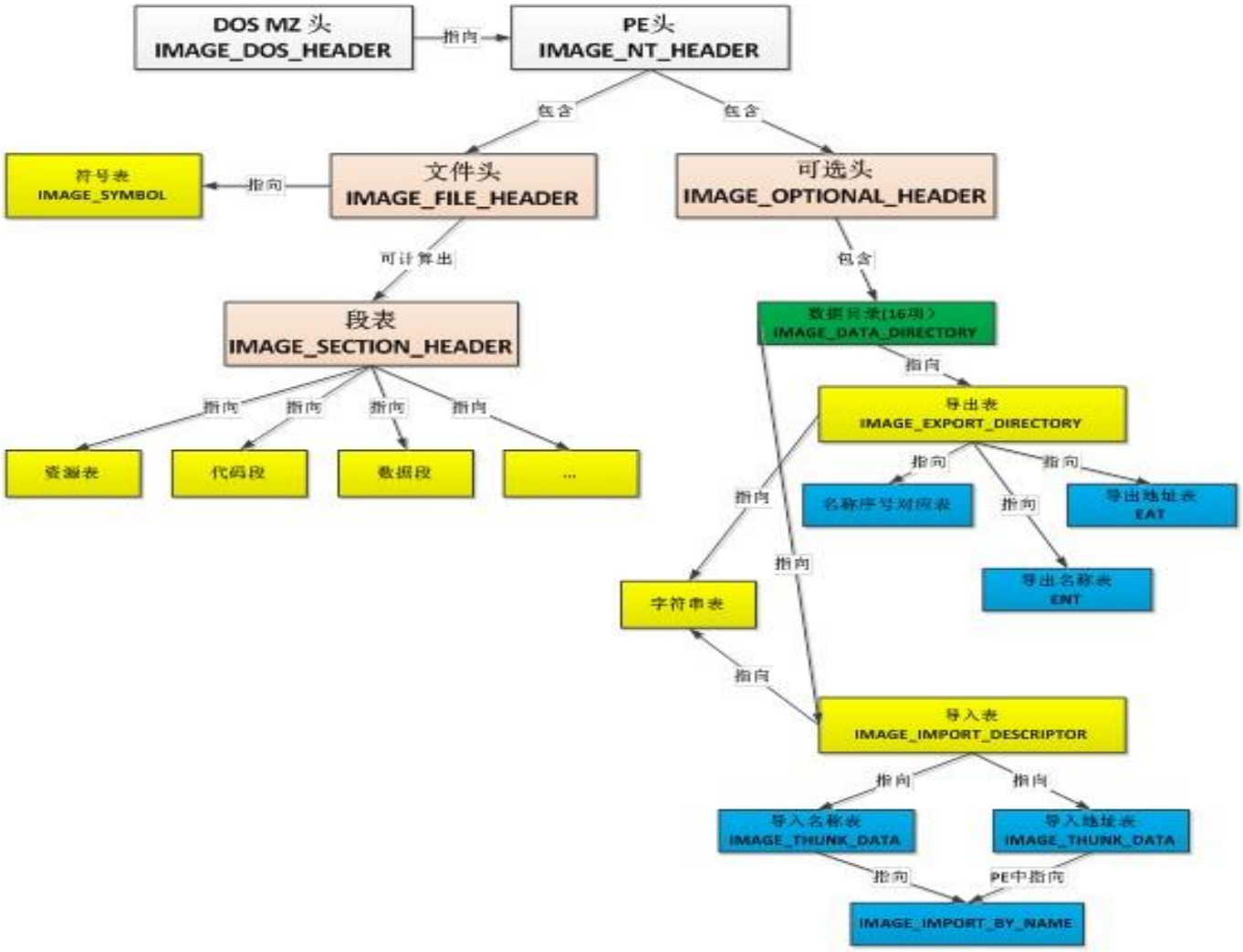


典型二进制格式-PE(2)

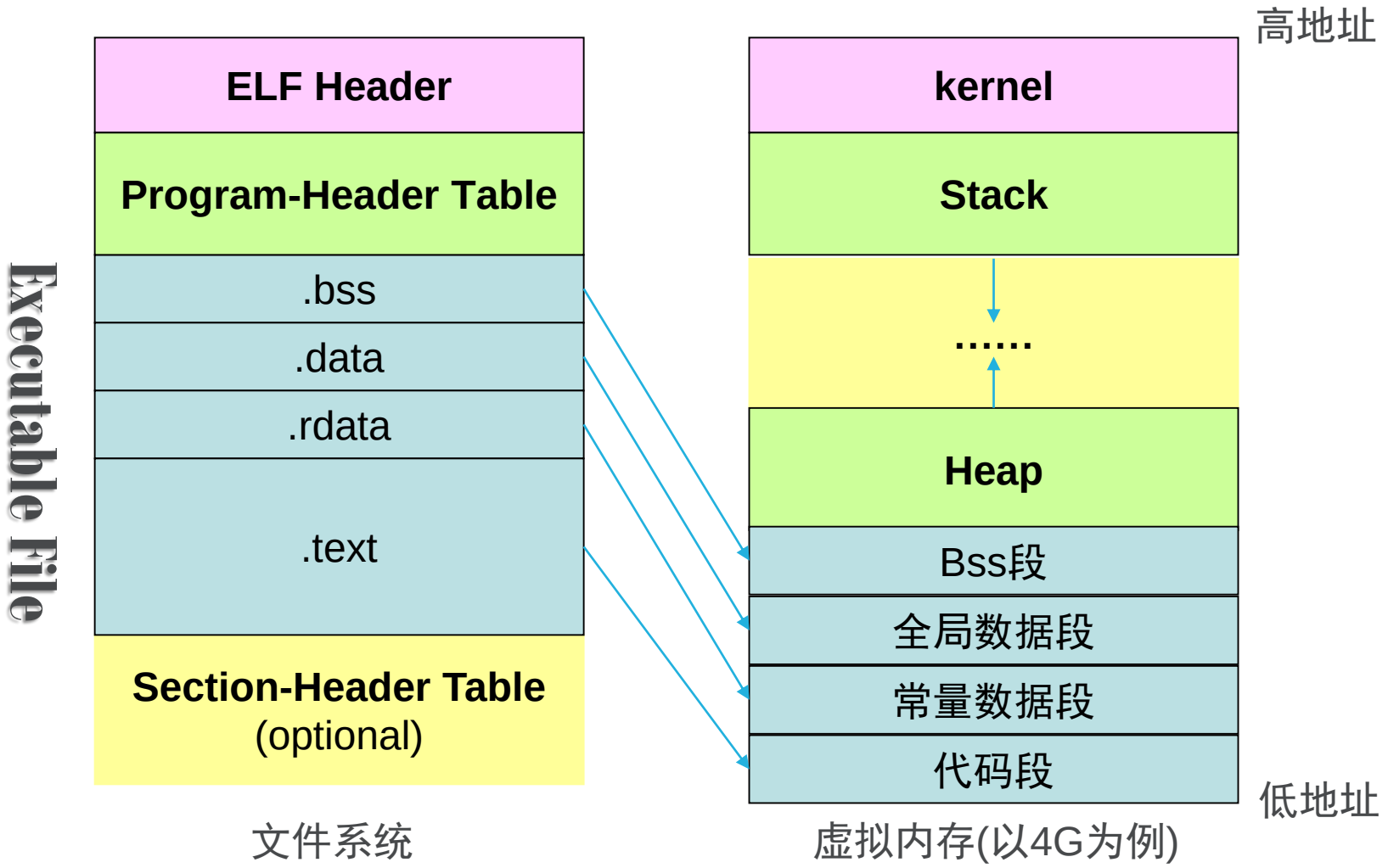
PE COFF=Portable Executable Common Object Format File



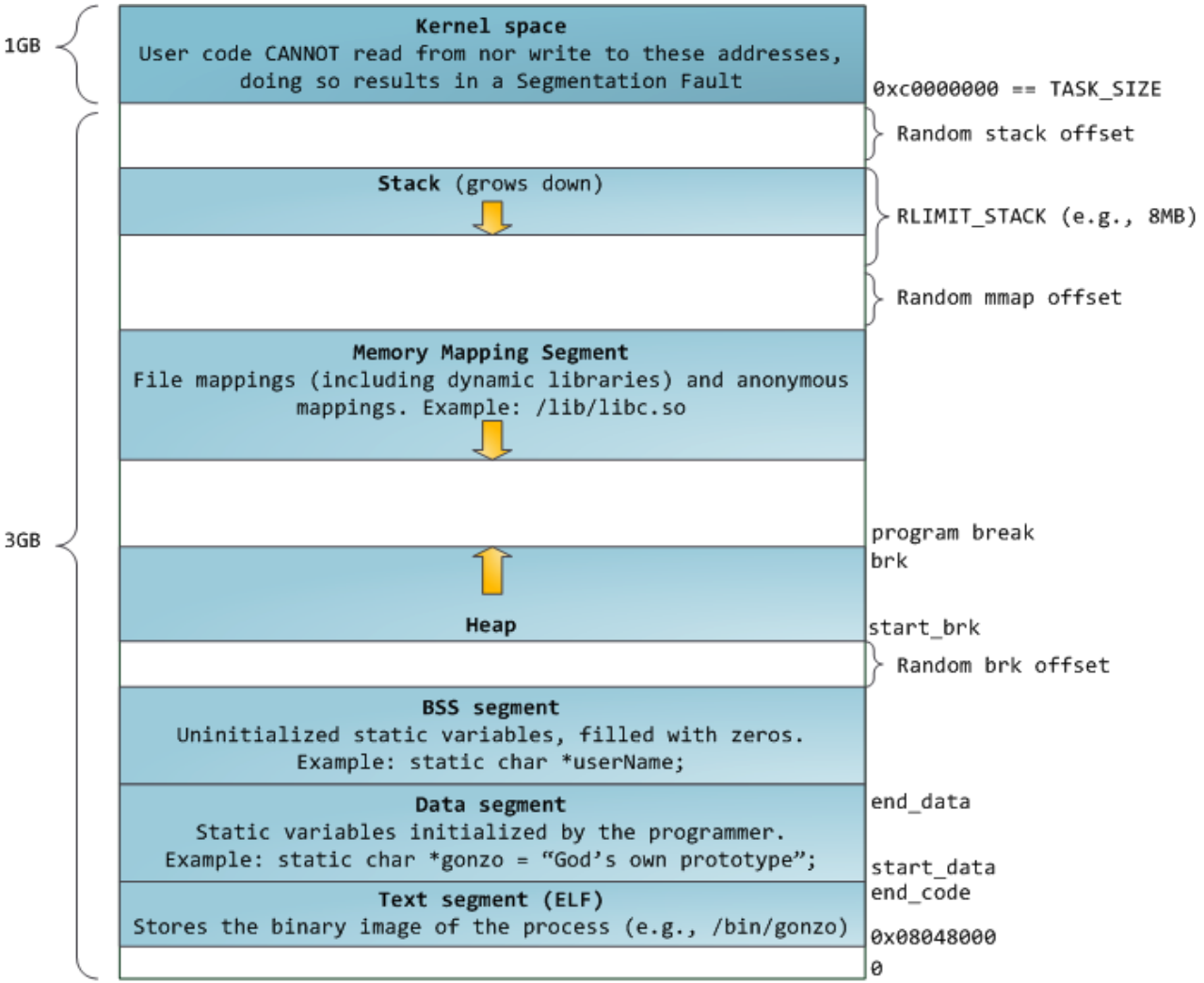
典型二进制格式-PE


DOS MZ 头
IMAGE_DOS_HEADER
指向
PE头
IMAGE_NT_HEADER
包含
包含
符号表
IMAGE_SYMBOL
指向
文件头
IMAGE_FILE_HEADER
可选头
IMAGE_OPTIONAL_HEADER
可计算出
包含
段表
IMAGE_SECTION_HEADER
数据目录(16项)
IMAGE_DATA_DIRECTORY
指向
指向
指向
指向
指向
资源表
代码段
数据段
...
导出表
IMAGE_EXPORT_DIRECTORY
指向
指向
指向
名称序号对应表
导出地址表
EAT
指向
指向
导出名称表
ENT
字符串表
指向
指向
导入表
IMAGE_IMPORT_DESCRIPTOR
指向
指向
导入名称表
IMAGE_THUNK_DATA
导入地址表
IMAGE_THUNK_DATA
指向
PE中指向
IMAGE_IMPORT_BY_NAME

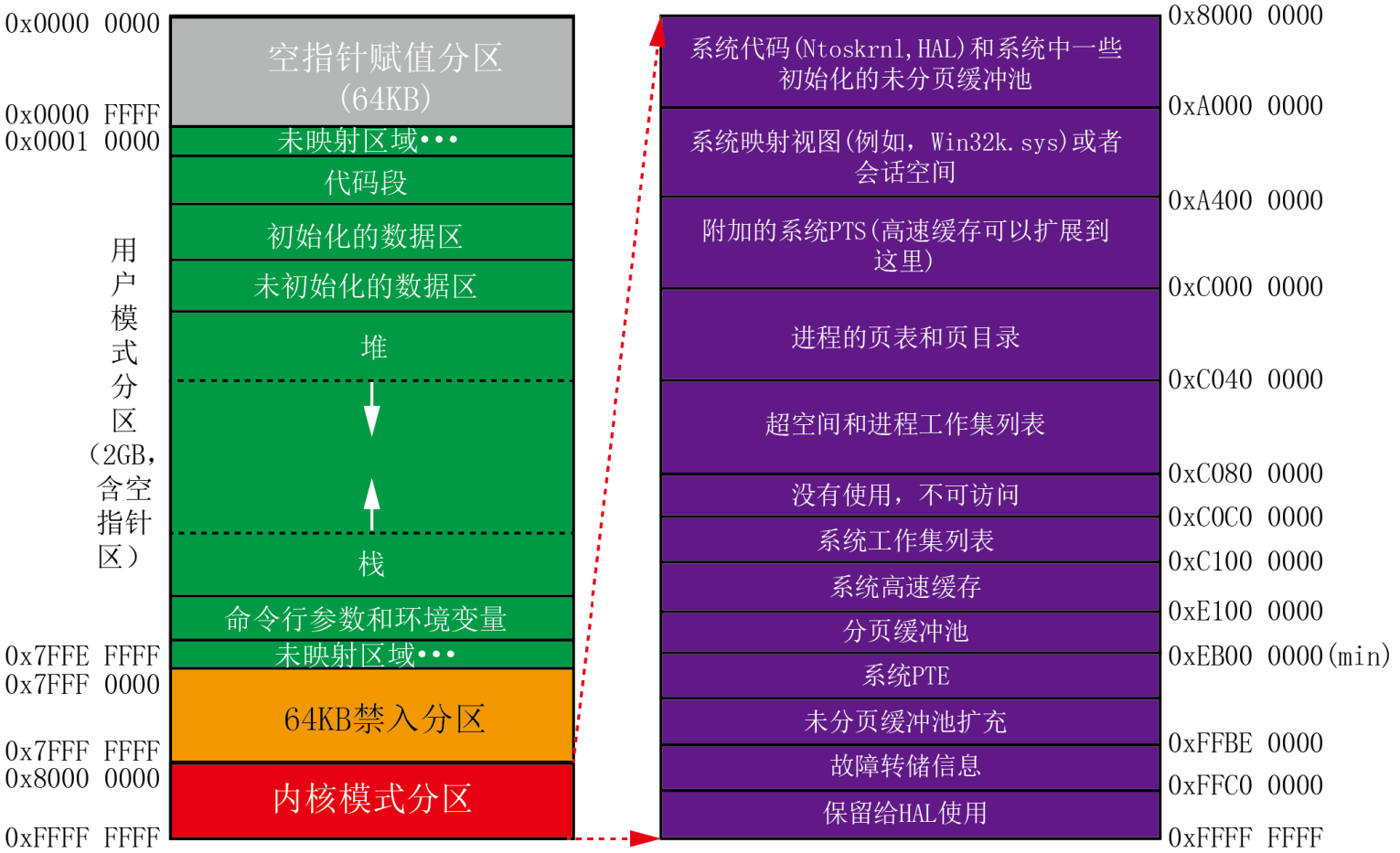
存储模型


高地址
ELF Header
Program-Header Table
.bss
.data
.rdata
.text
Section-Header Table
(optional)
Executable File
文件系统
kernel
Stack
.....
Heap
Bss段
全局数据段
常量数据段
代码段
低地址
虚拟内存(以4G为例)

存储模型-Linux(32位)

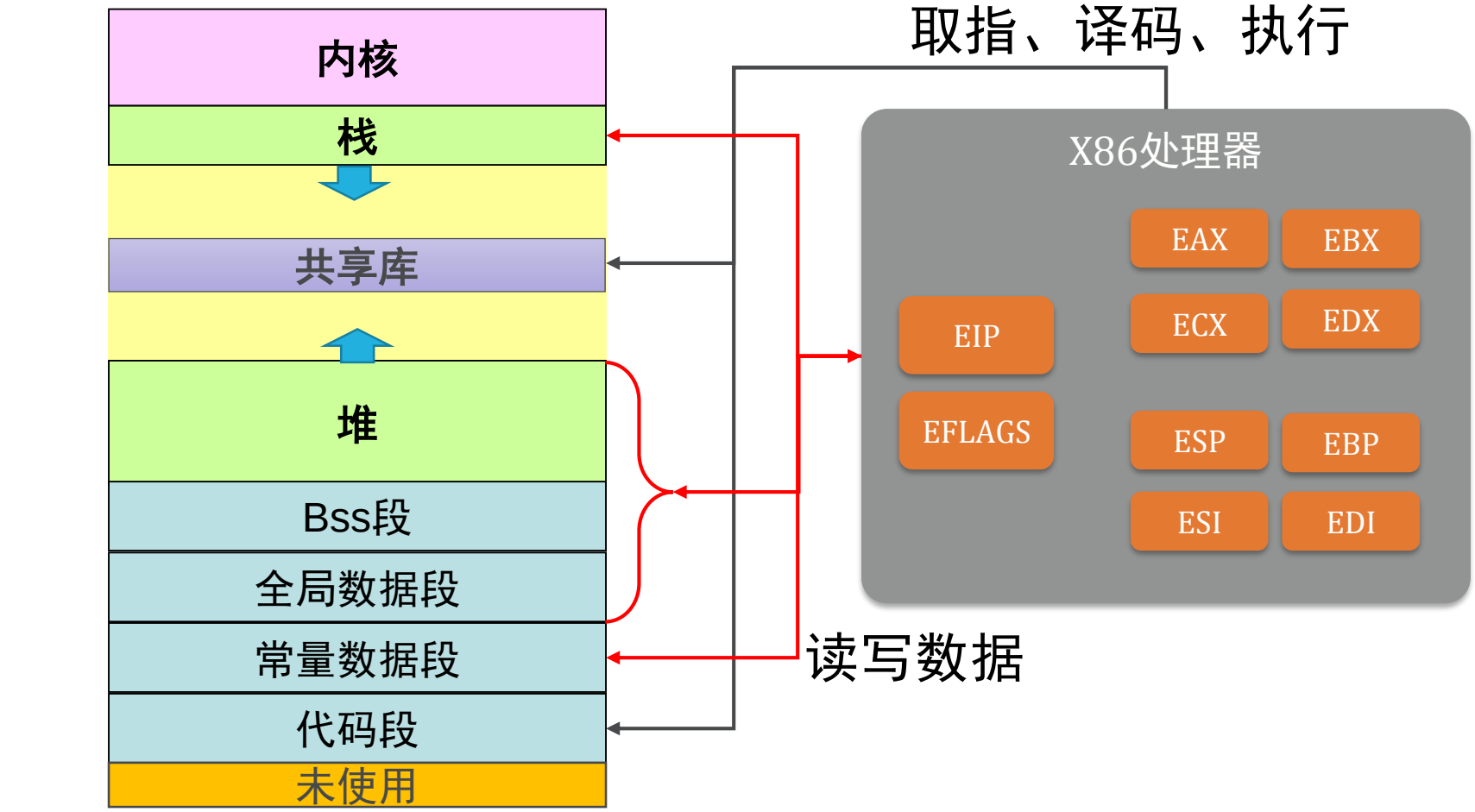


存储模型-Windows(32位)

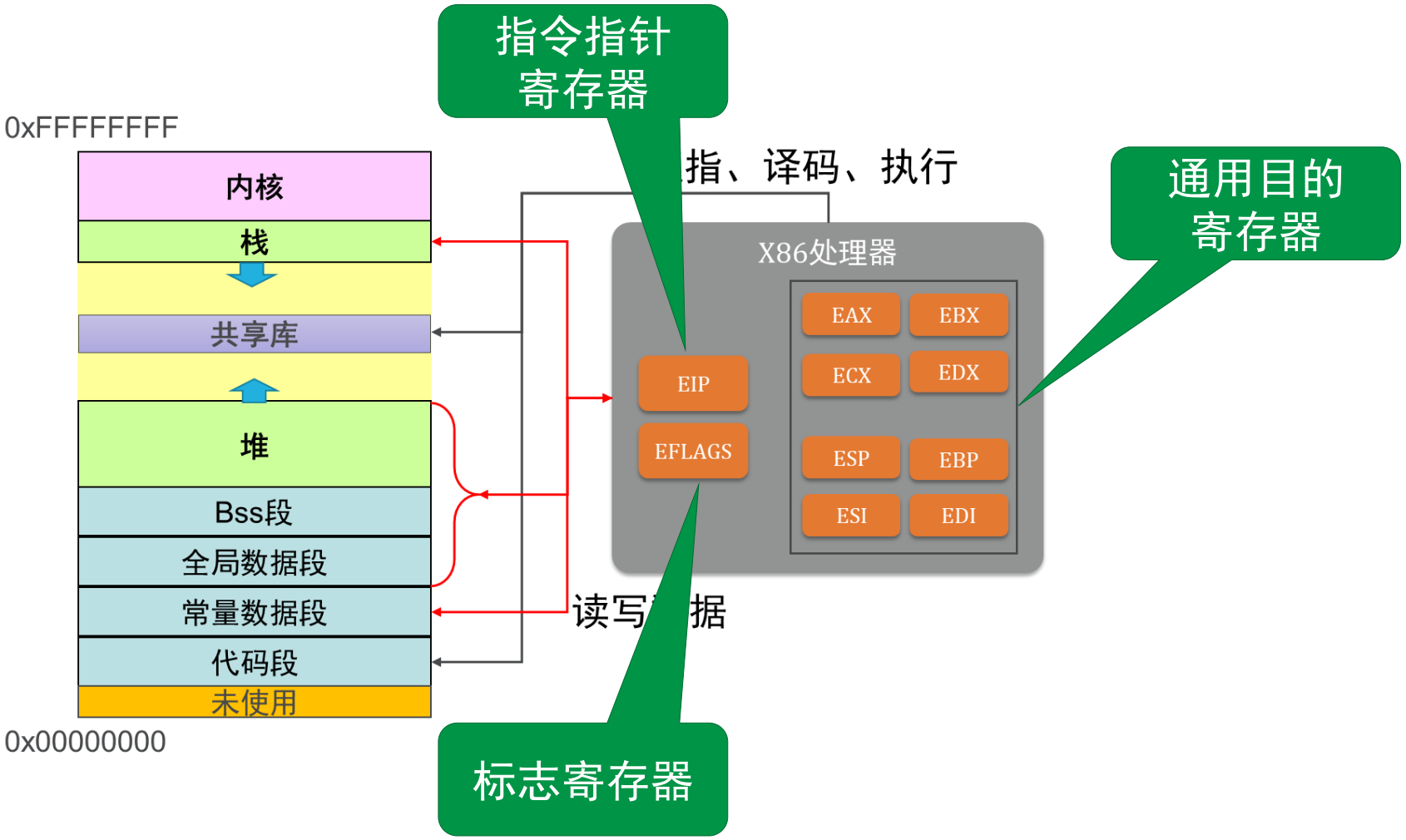


x86系统32位Windows内存空间布局

执行模型


0xFFFFFFFF
内核
栈
共享库
堆
Bss段
全局数据段
常量数据段
代码段
未使用
0x00000000
取指、译码、执行
X86处理器
EAX
EBX
ECX
EDX
EIP
EFLAGS
ESP
EBP
ESI
EDI
读写数据

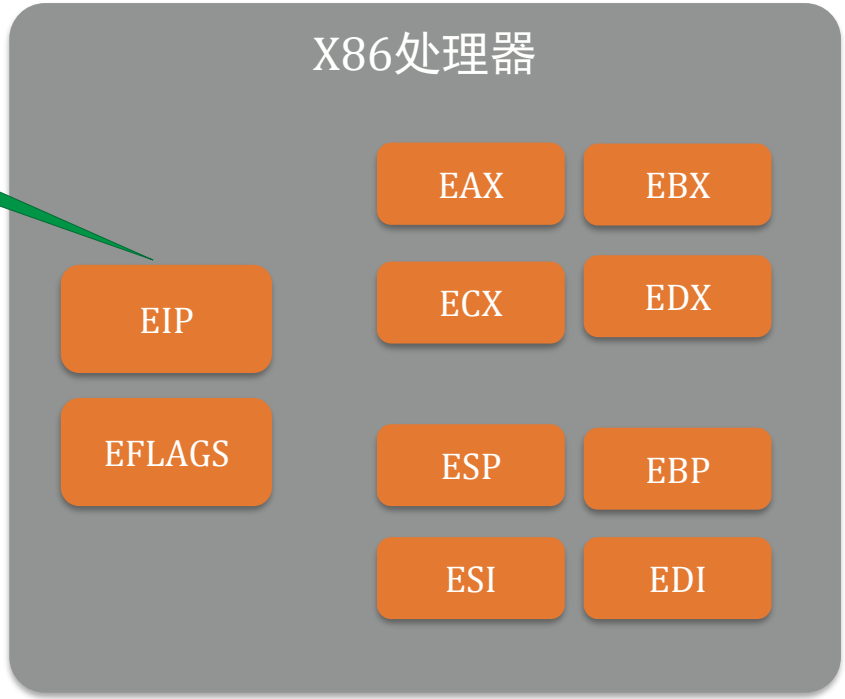
执行模型


指令指针
寄存器
0xFFFFFFFF
内核
栈
共享库
堆
Bss段
全局数据段
常量数据段
代码段
未使用
0x00000000
指、译码、执行
X86处理器
EIP
EFLAGS
EAX
EBX
ECX
EDX
ESP
EBP
ESI
EDI
通用目的
寄存器
读写
据
标志寄存器

控制操作

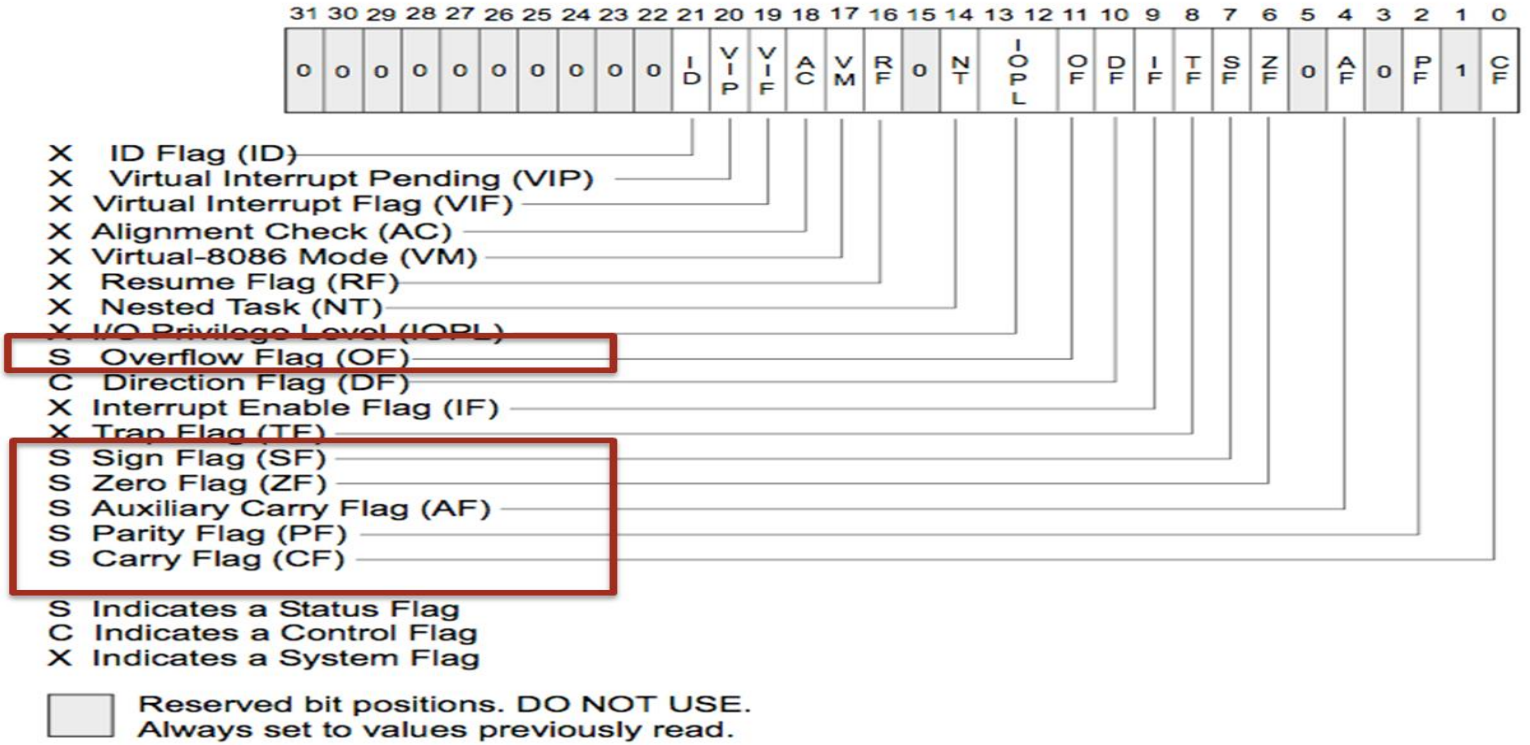
EIP无法直接读取或者设置

- 跳转
  - jmp 0x45, 直接跳转
  - jmp *eax , 间接跳转
- 分支
  - if (EFLAG) jmp x
  - 由EFLAG的某一位决定是否跳转



EFLAGS

- eflag可由指令设置
- 常由比较和算数运算指令设置



实例分析

C

```
1. if(x <= y)
2.     z = x;
3. else
4.     z = y;
```

2条汇编指令
1. 设置eflag
2. 测试eflag并分支

伪汇编

1. 计算x – y. 设置eflags:
   1. 如果x < y，则CF =1
   2. 如果x==y，则ZF =1
2. 测试EFLAGS:如果CF和ZF同时未被设置，则跳转至E
3. 将x赋值给z
4. 跳转至F

E. 将y赋值给z

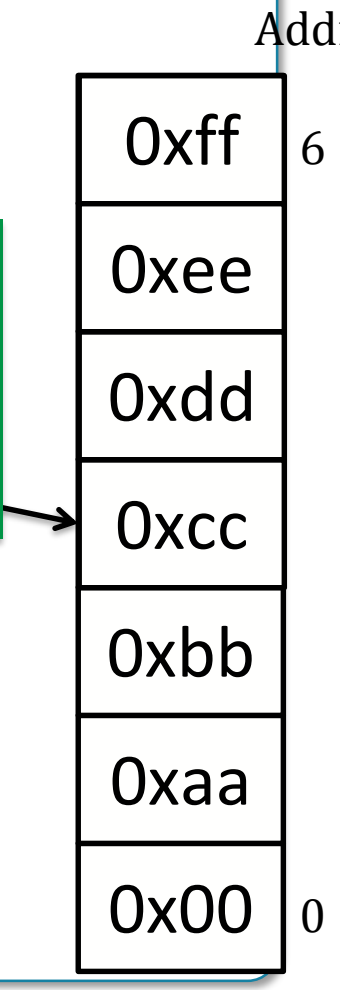
F.

数据操作

EDX

Register	Value
eax	
edx	
ebx	

EDX = 0xffeeddcc!



mov edx, [eax]

数据操作

从源到目的

赋值语句的镜像

含义	AT&T语法	Intel语法
ebx = eax	movl %eax, %ebx	mov ebx, eax
eax = eax + ebx	addl %ebx, %eax	add eax, ebx
ecx = ecx << 2	shl $2, %ecx	shl ecx, 2

寻址模式

格式	含义(M表示内存)
imm (r)	M[r + imm]
imm (r_1, r_2)	M[r_1 + r_2 + imm]
imm (r_1, r_2, s)	M[r_1 + r_2*s + imm]
imm	M[imm]

内存操作

从内存中读/写值: mov

```
<eax> = *buf;
```

```
mov   -0x38(%ebp),%eax (I)
mov   eax, [ebp-0x38] (A)
```

读取内存地址: lea

```
<eax> = buf;
```

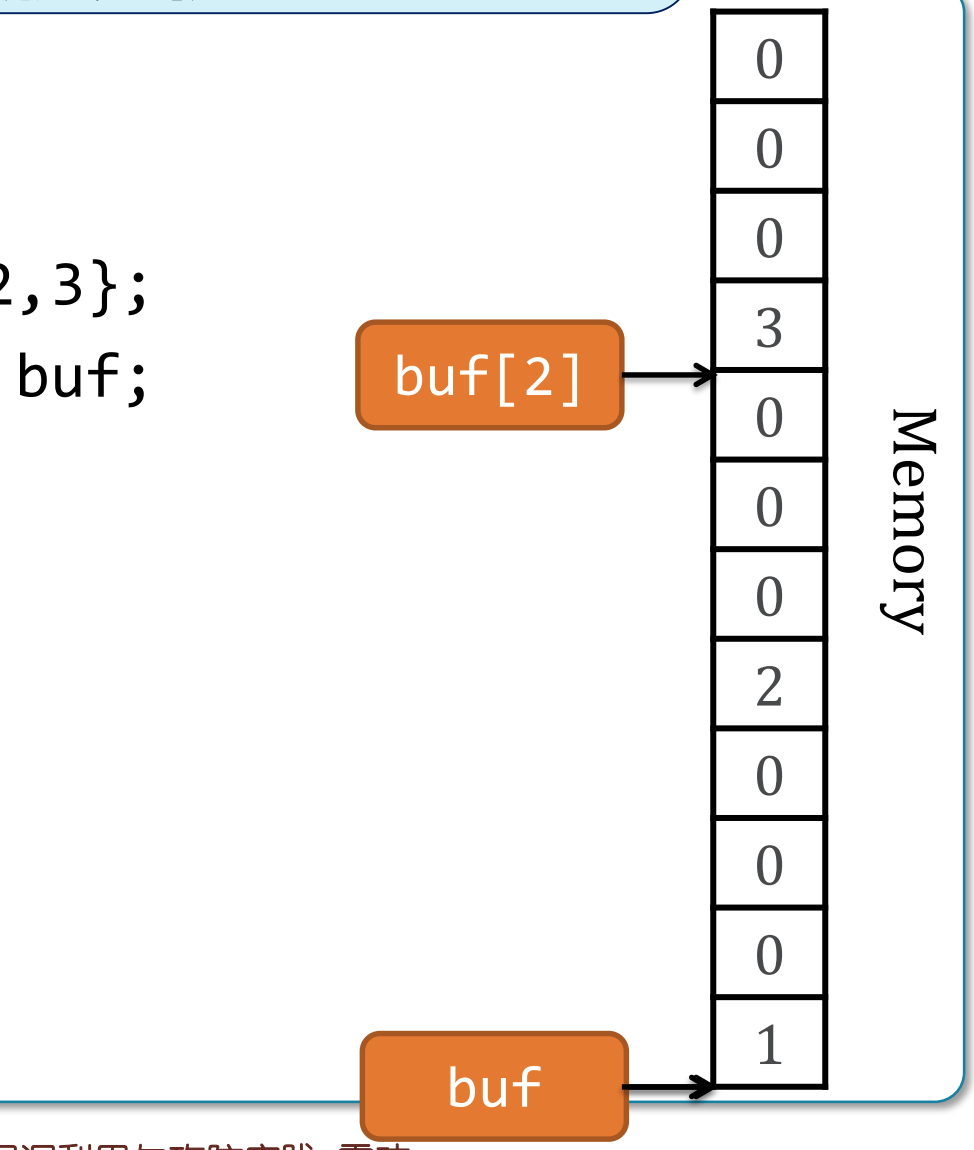
```
lea   -0x38(%ebp),%eax (I)
lea   eax, [ebp-0x38] (A)
```

实例分析

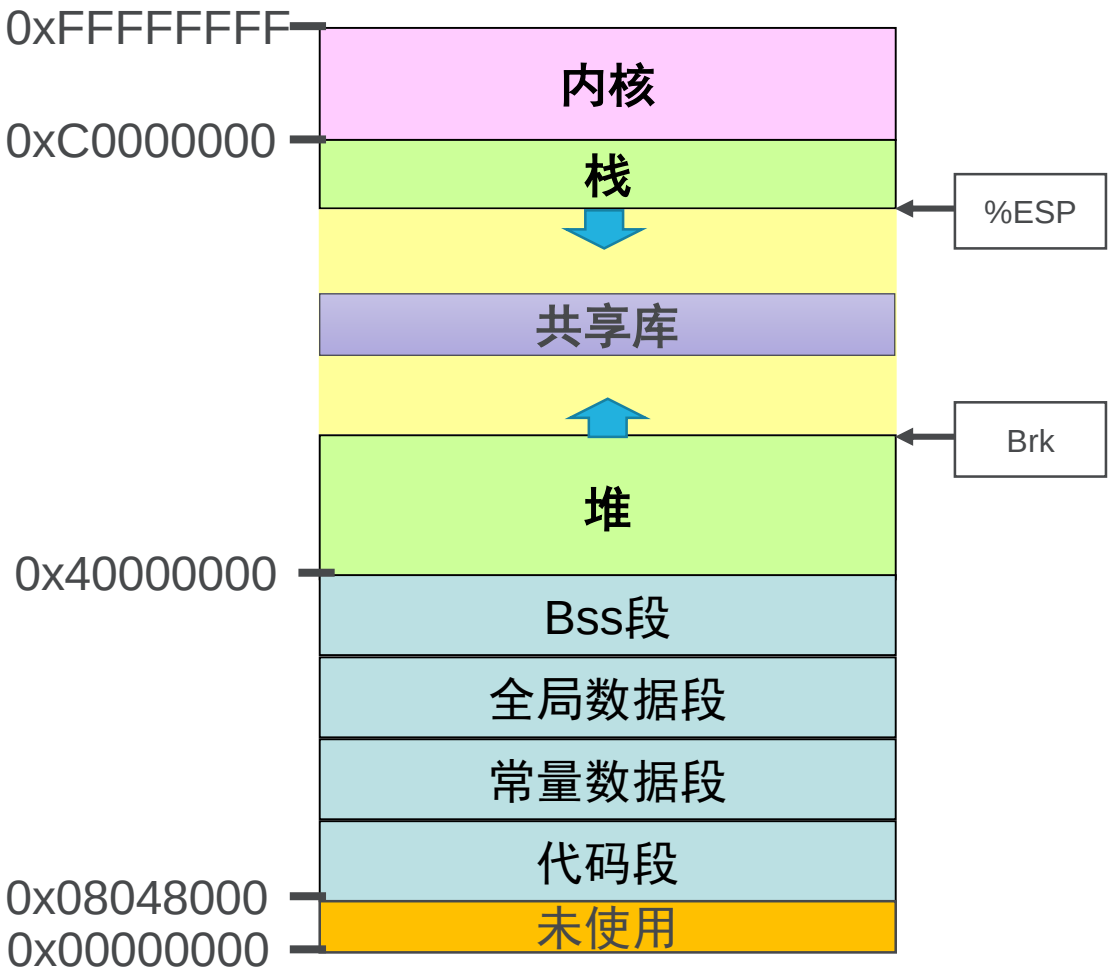
```
typedef char *addr_t;
uint32_t w, x, y, z;
uint32_t buf[3] = {1,2,3};
addr_t ptr = (addr_t) buf;

w = buf[2];
x = *(buf + 2);
```

X是多少？指向的内存单位是哪个？



执行过程

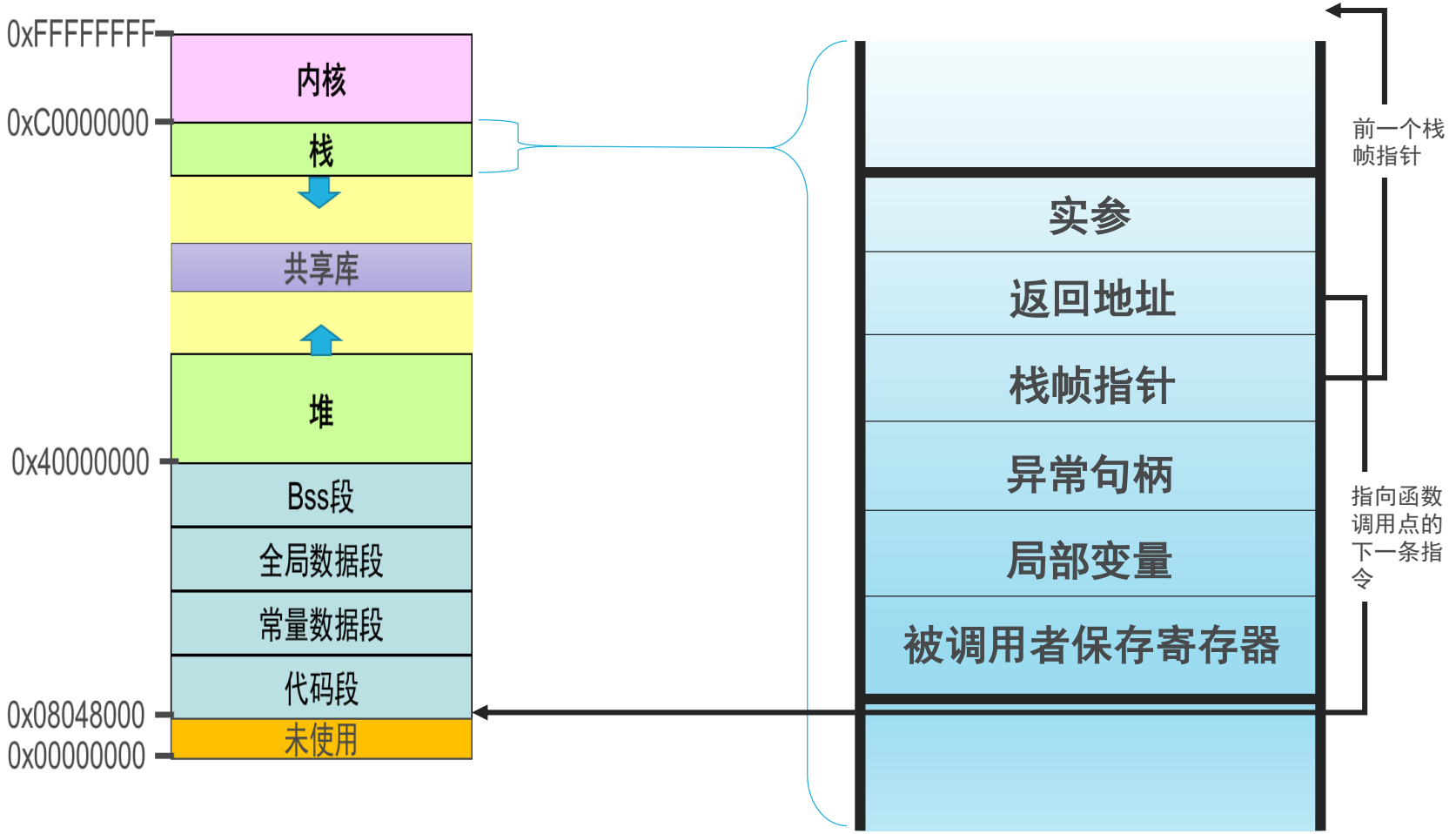


- 栈
  - 局部变量
  - 生命期: 栈帧
- 堆
  - 动态分配内存
  - 生命期:被释放之前

栈帧作用

- 函数执行时面临的关键问题
  - 如何为局部变量分配空间
  - 如何传递参数
  - 如何传递返回值
  - 如何让可能任意多个局部变量共享8个寄存器
- 一个栈帧提供了必要的内存空间
  - 每次函数调用都拥有自己的栈帧
  - 栈帧要按照后进先出的顺序操作
    - 如果函数A调用函数B，B的栈帧要在A的栈帧之前退出

栈帧内容


0xFFFFFFFF
内核
0xC0000000
栈
共享库
堆
0x40000000
Bss段
全局数据段
常量数据段
代码段
0x08048000
未使用
0x00000000
实参
返回地址
栈帧指针
异常句柄
局部变量
被调用者保存寄存器
前一个栈帧指针
指向函数调用点的下一条指令

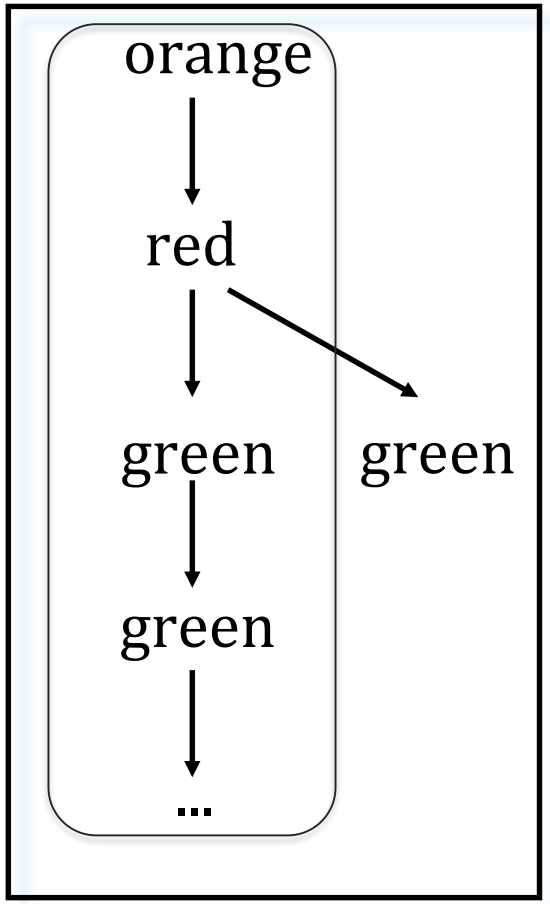
示例分析

```
orange(...)
{
    ...
    red()
    ...
}
```

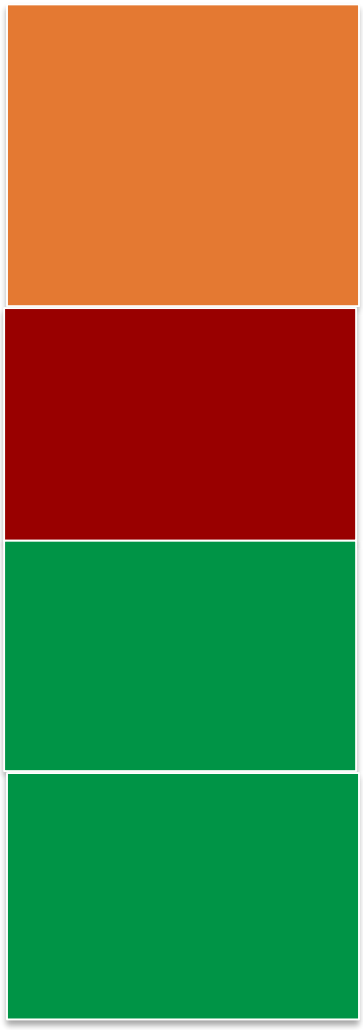
```
red(...)
{
    ...
    green()
    ...
    green()
}
```

```
green(...)
{
...
green()
...
}
```

函数调用链



示例分析

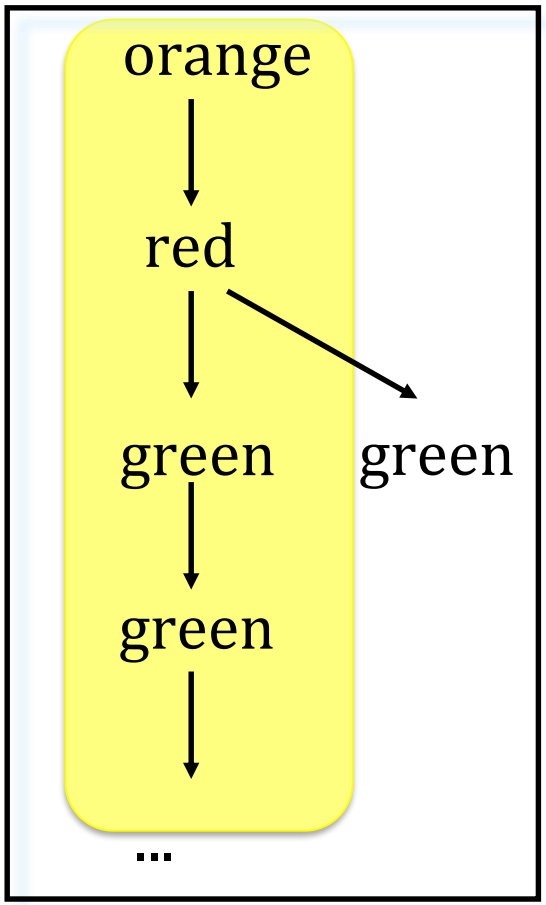


栈帧包含
- 局部变量
- 实际参数
- 临时空间

调用red时压新的栈帧

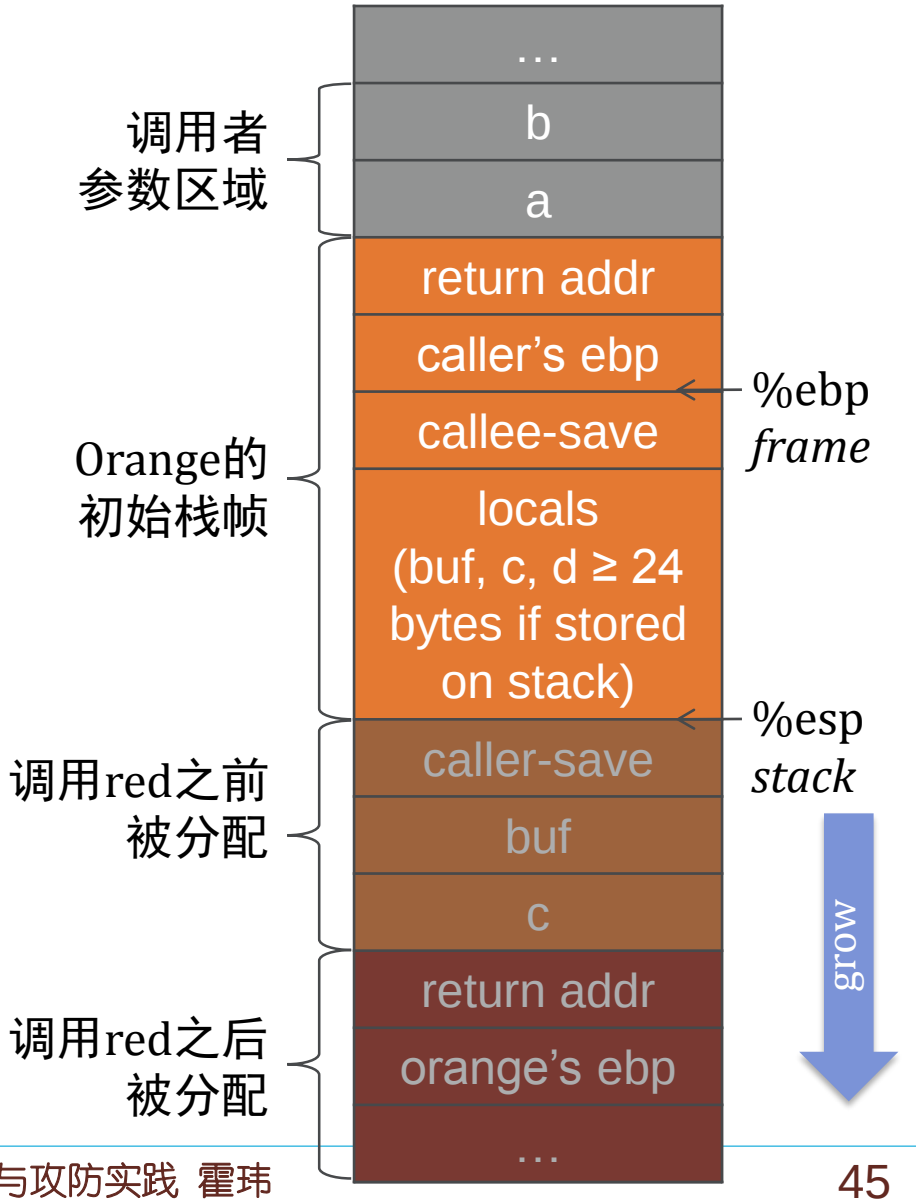
当green返回时，弹出它的栈帧

函数调用链



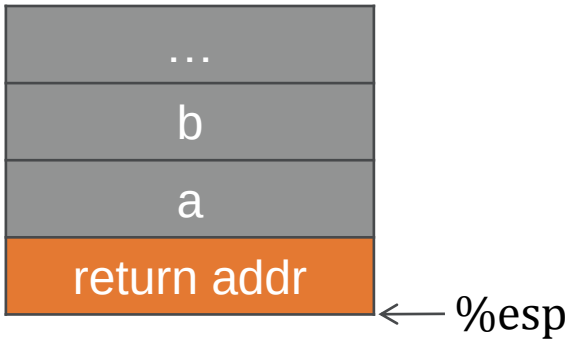
Linux&Gcc调用过程分析

```
int orange(int a, int b)
{
   char buf[16];
   int c, d;
   if(a > b)
      c = a;
   else
      c = b;
   d = red(c, buf);
   return d;
}
```



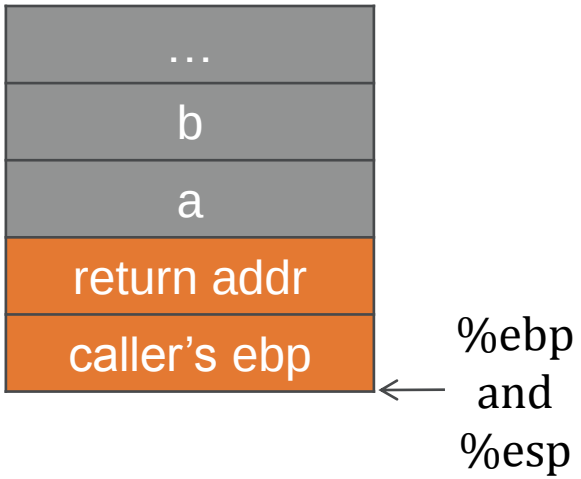
当orange获得控制权时

1. 返回地址已经被调用者压栈



当orange获得控制权时

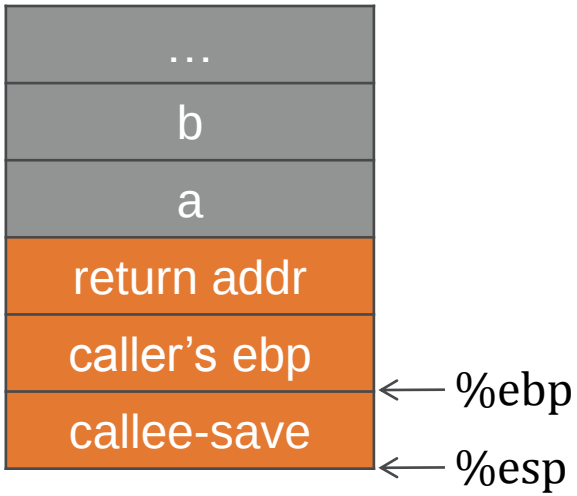
1. 返回地址已经被调用者压栈
2. 获得栈帧指针
   - 将调用者的ebp压栈
   - 复制esp到ebp
   - 第一个参数在ebp+8



二 背景知识

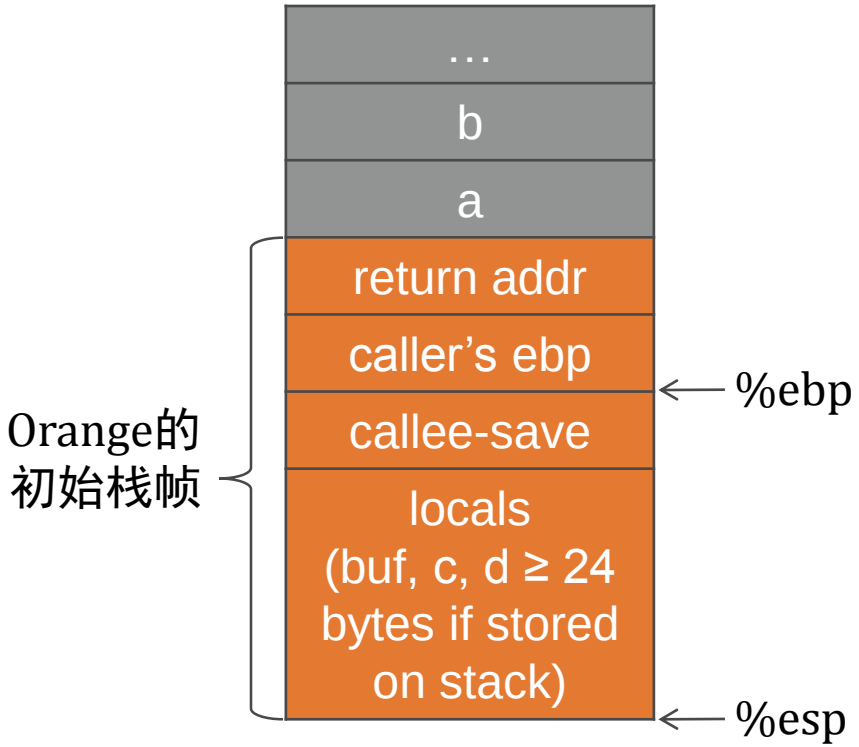
当orange获得控制权时

1. 返回地址已经被调用者压栈
2. 获得栈帧指针
   - 将调用者的ebp压栈
   - 复制esp到ebp
   - 第一个参数在ebp+8
3. 保存将要使用的被调用者保存寄存器
   - edi, esi, ebx
   - esp可以通过运算恢复

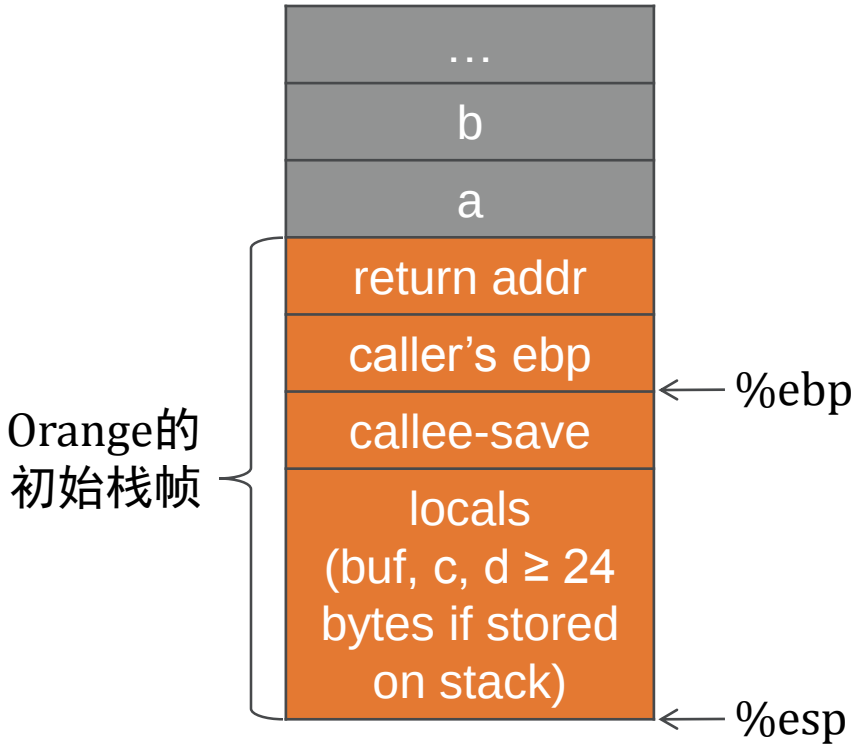


当orange获得控制权时

1. 返回地址已经被调用者压栈
2. 获得栈帧指针
   - 将调用者的ebp压栈
   - 复制esp到ebp
   - 第一个参数在ebp+8
3. 保存将要使用的被调用者保存寄存器
   - edi, esi, ebx
   - esp可以通过运算恢复
4. 为局部变量分配空间
   - 减小esp
   - “*spilled*”寄存器中的活跃变量



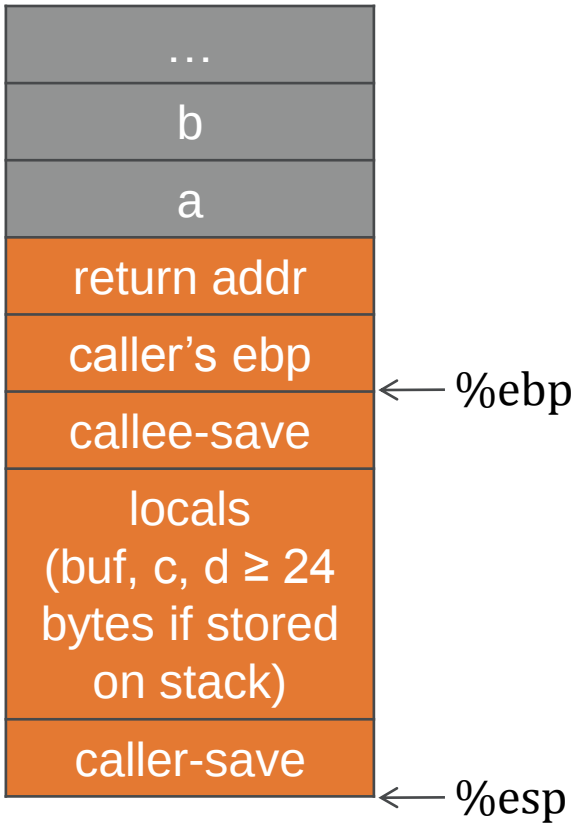
当调用者orange调用被调用者red时



当调用者orange调用被调用者red时

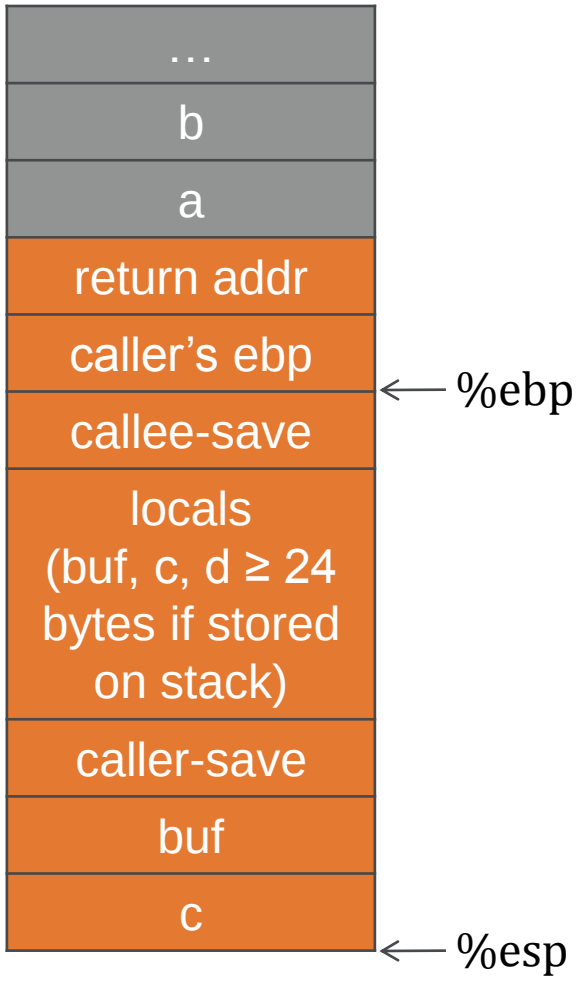
1. 对调用者保存寄存器压栈

- eax, edx, ecx



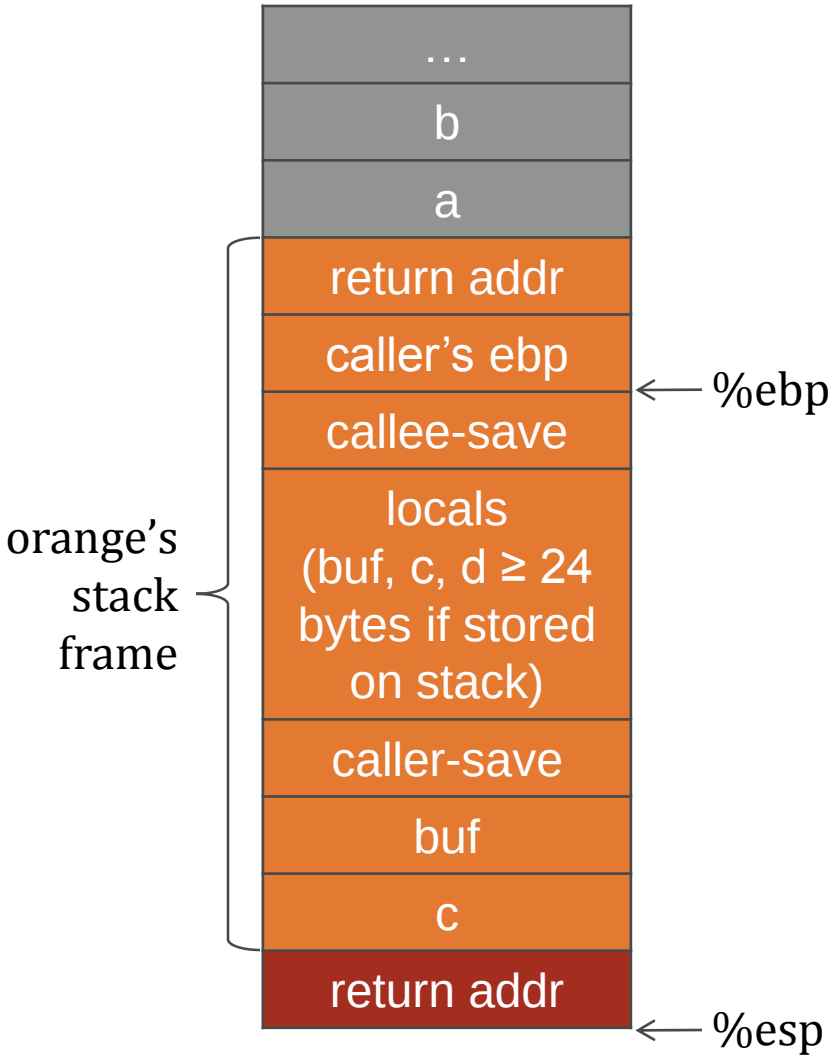
当调用者orange调用被调用者red时

1. 对调用者保持寄存器压栈
   - eax, edx, ecx
2. 自右向左将red的实参压栈
   - 从被调用者的角度，第一个实参距离栈顶最近



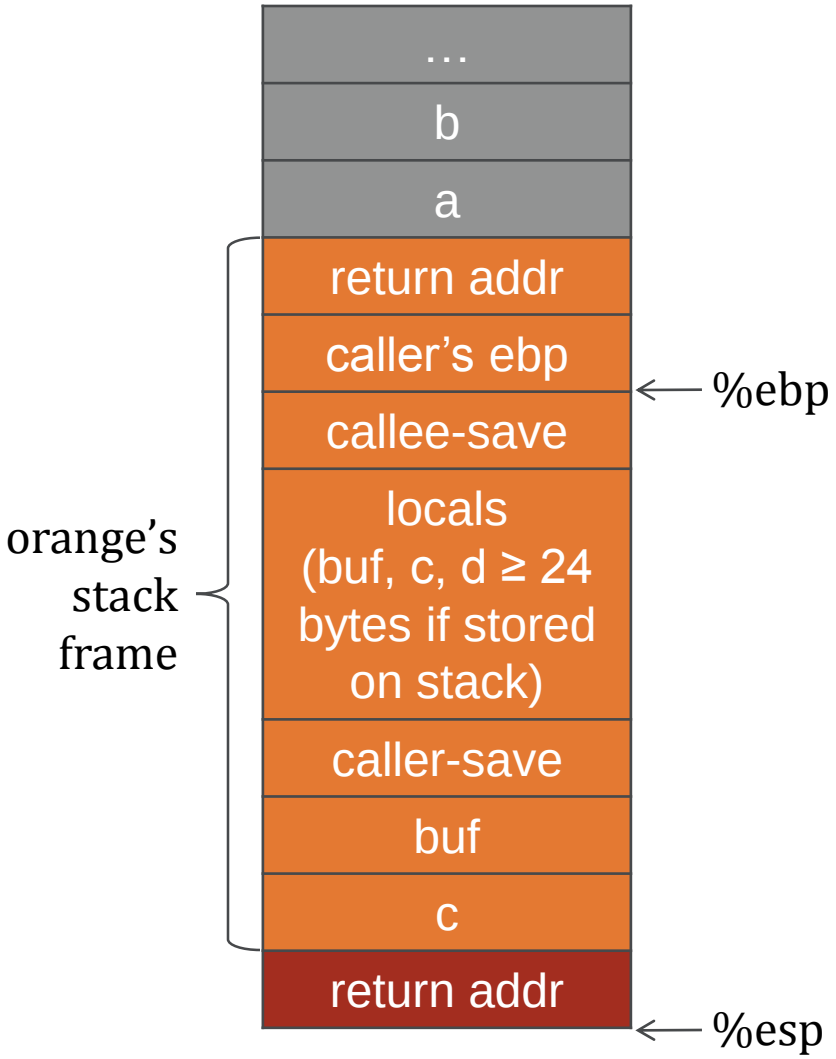
当调用者orange调用被调用者red时

1. 对调用者保持寄存器压栈
   - eax, edx, ecx
2. 自右向左将red的实参压栈
   - 从被调用者的角度，第一个实参距离栈顶最近
3. 返回地址压栈，即red返回后orange中将要执行的下一条指令地址



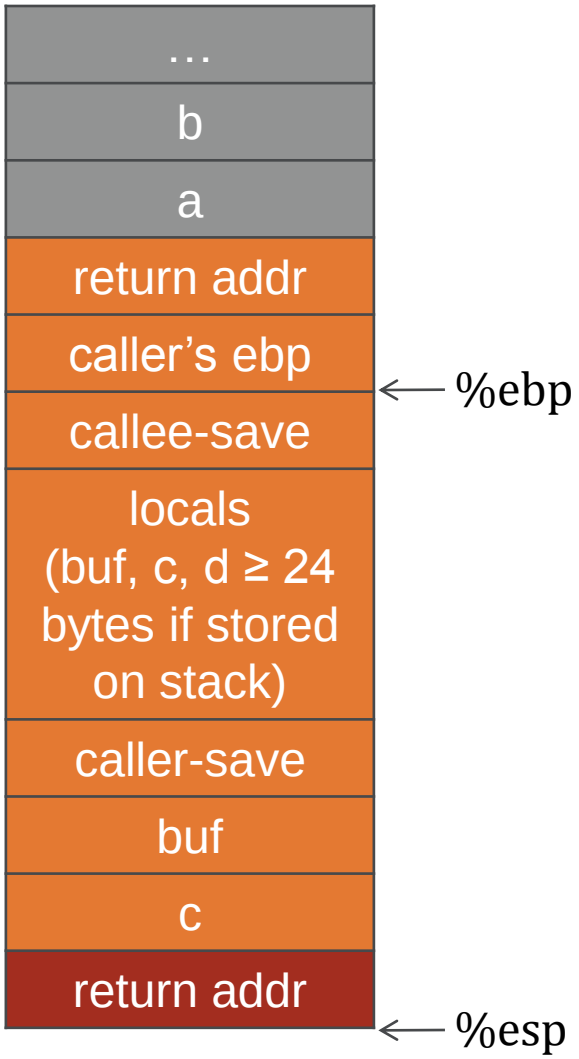
当调用者orange调用被调用者red时

1. 对调用者保持寄存器压栈
   - eax, edx, ecx
2. 自右向左将red的实参压栈
   - 从被调用者的角度，第一个实参距离栈顶最近
3. 返回地址压栈，即red返回后orange中将要执行的下一条指令地址
4. 当控制权传给 red
   - 通常用call指令与第3步同时完成



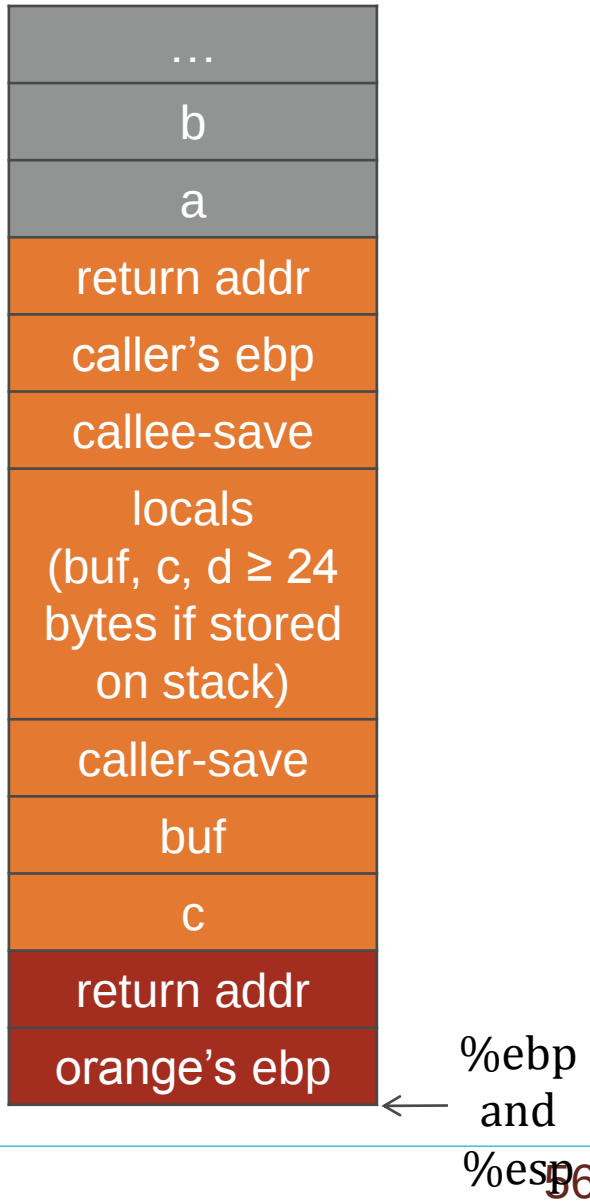
当red获得控制权时

1. 返回地址已经由orange压栈



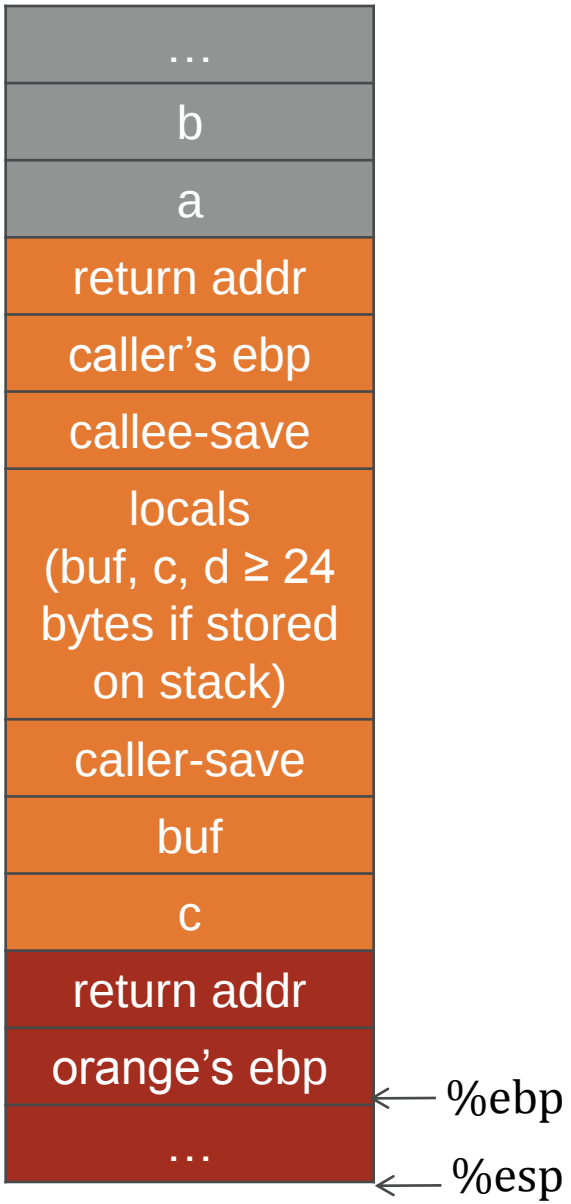
当red获得控制权时

1. 返回地址已经由orange压栈
2. 获得自己的栈帧指针



当red获得控制权时

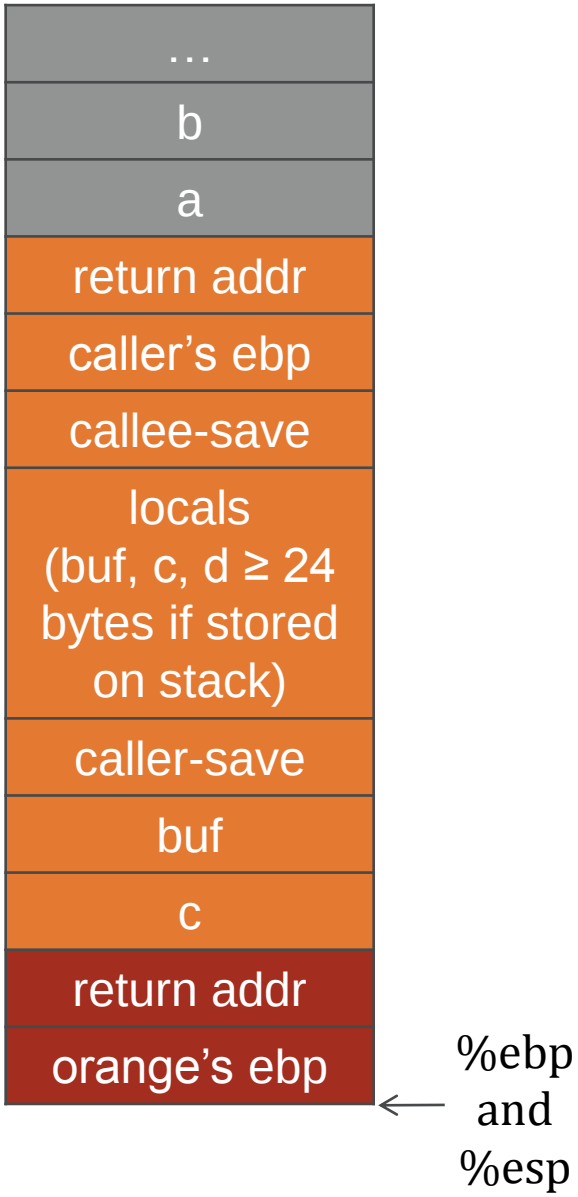
1. 返回地址已经由orange压栈
2. 获得自己的栈帧指针
3. … (red 重复上述过程) …



二 背景知识

当red获得控制权时

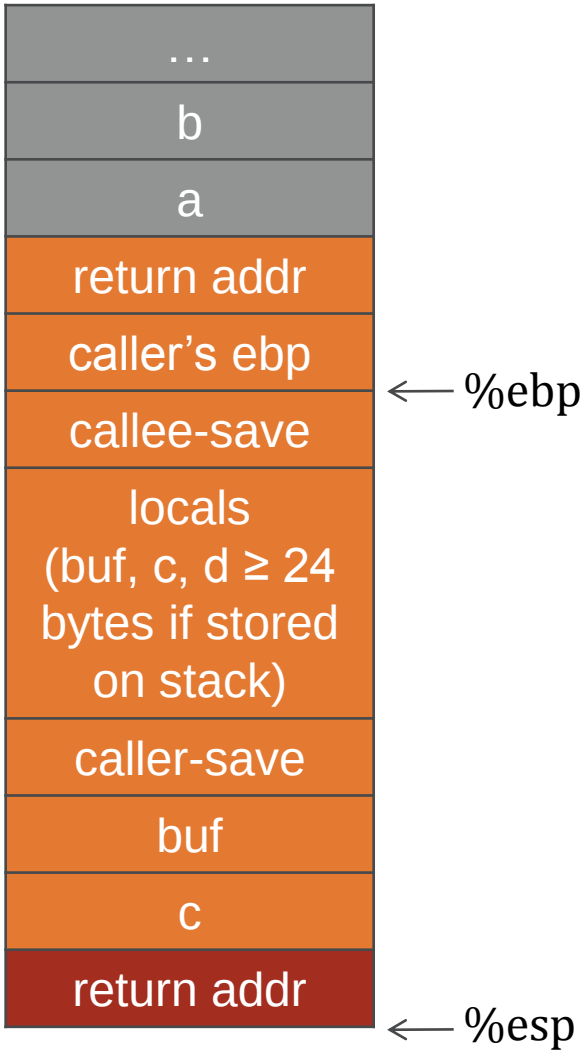
1. 返回地址已经由orange压栈
2. 获得自己的栈帧指针
3. … (red 重复上述过程) …
4. 将返回值存储在eax中
5. 释放局部变量空间
   - 增加esp
6. 恢复被调用者保存寄存器



二 背景知识

当red获得控制权时

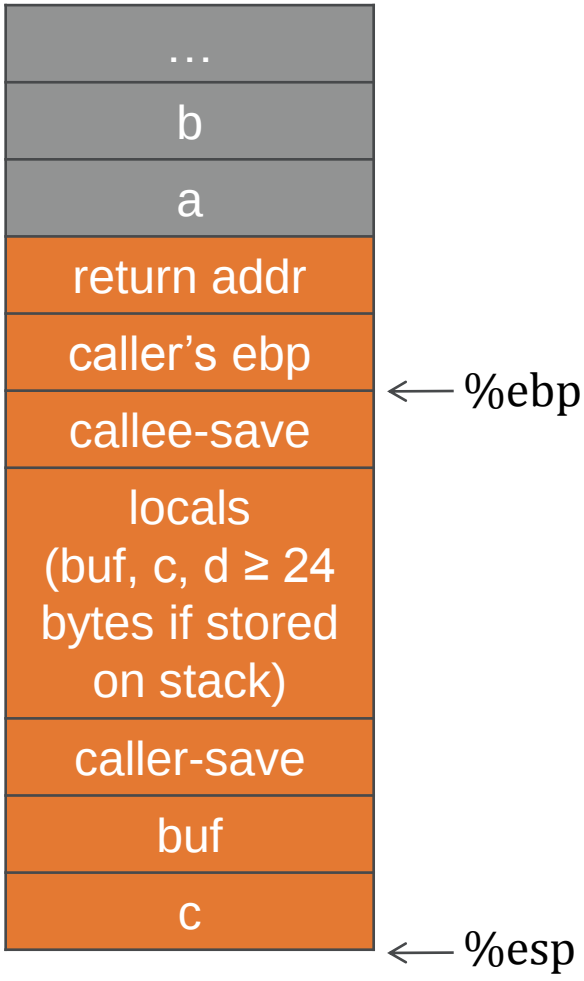
1. 返回地址已经由orange压栈
2. 获得自己的栈帧指针
3. … (red 重复上述过程) …
4. 将返回值存储在eax中
5. 释放局部变量空间
   - 增加esp
6. 恢复被调用者保存寄存器
7. 恢复orange栈帧指针
   - pop %ebp



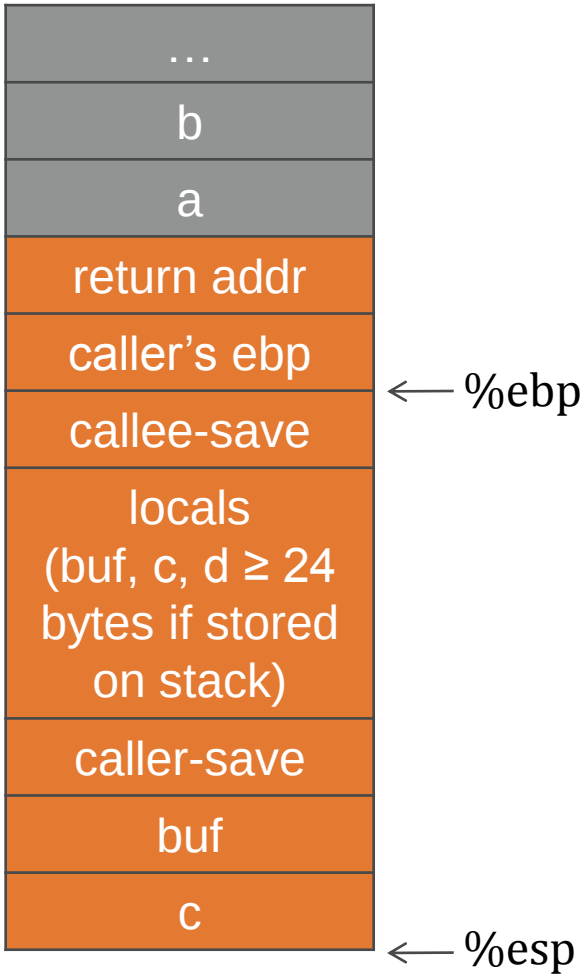
二 背景知识

当red获得控制权时

1. 返回地址已经由orange压栈
2. 获得自己的栈帧指针
3. … (red 重复上述过程) …
4. 将返回值存储在eax中
5. 释放局部变量空间
   - 增加esp
6. 恢复被调用者保存寄存器
7. 恢复orange栈帧指针
   - pop %ebp
8. 将控制权交回orange
   - ret
   - 从栈中弹出返回地址并跳转到相应指令

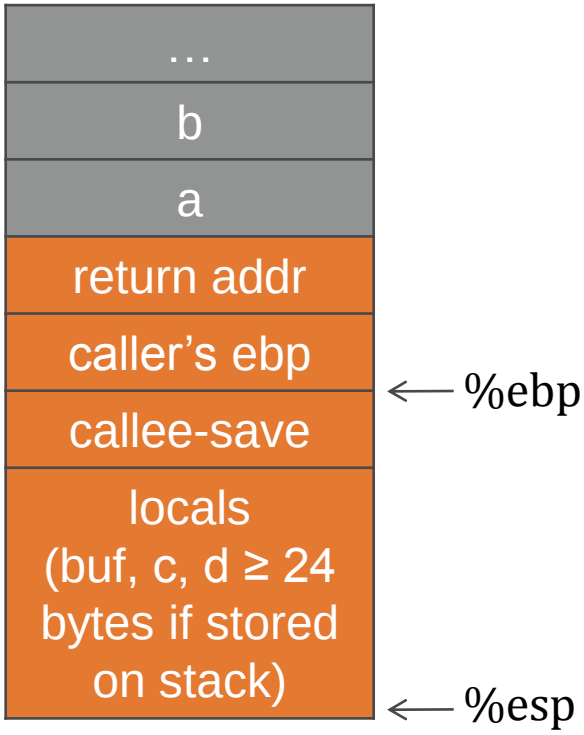


当orange再次获得控制权时



当orange再次获得控制权时

1. 清理传给red的实参
   - 增加esp
2. 恢复调用者保存寄存器
   - Pops
3. …



课程大纲

一 课程总览

- 1.1 授课团队介绍
- 1.2 课程定位及内容
- 1.3 课程要求

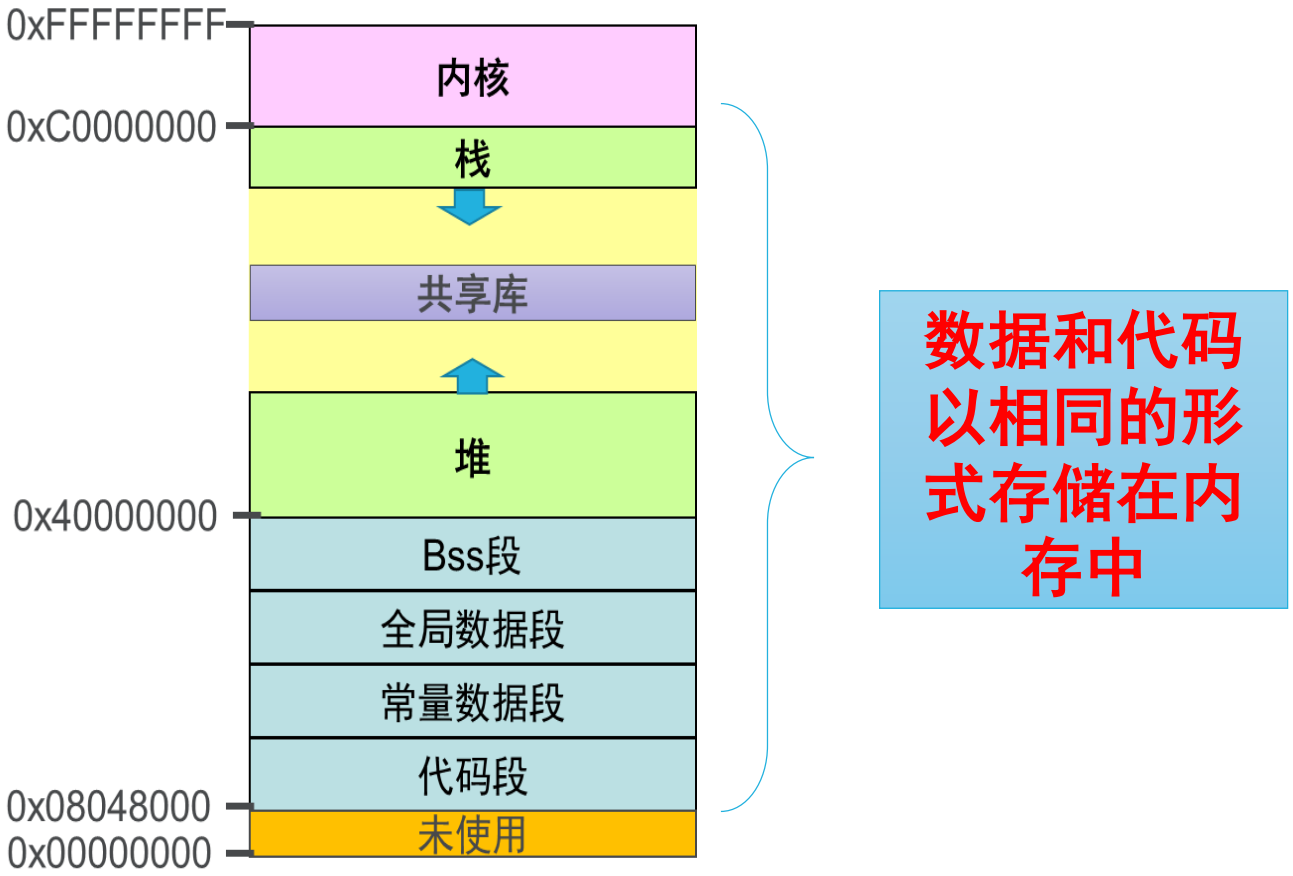
二 背景知识

- 2.1 典型二进制文件格式
- 2.2 存储模型
- 2.3 执行模型

三 内容概述

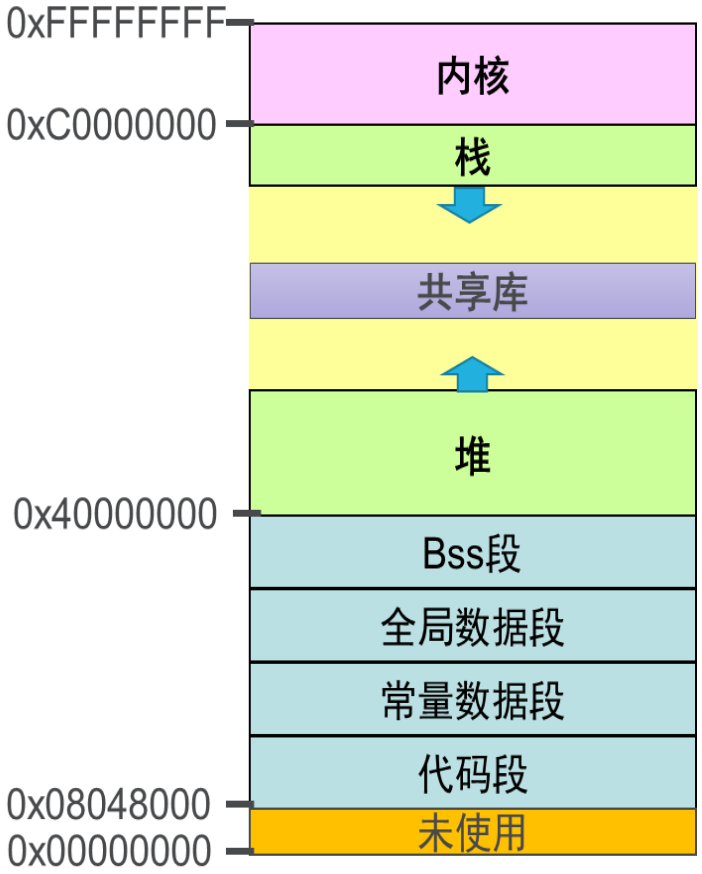
内存破坏类漏洞

冯 诺依曼体系结构是内存破坏类漏洞的具有利用性的根源



控制流劫持

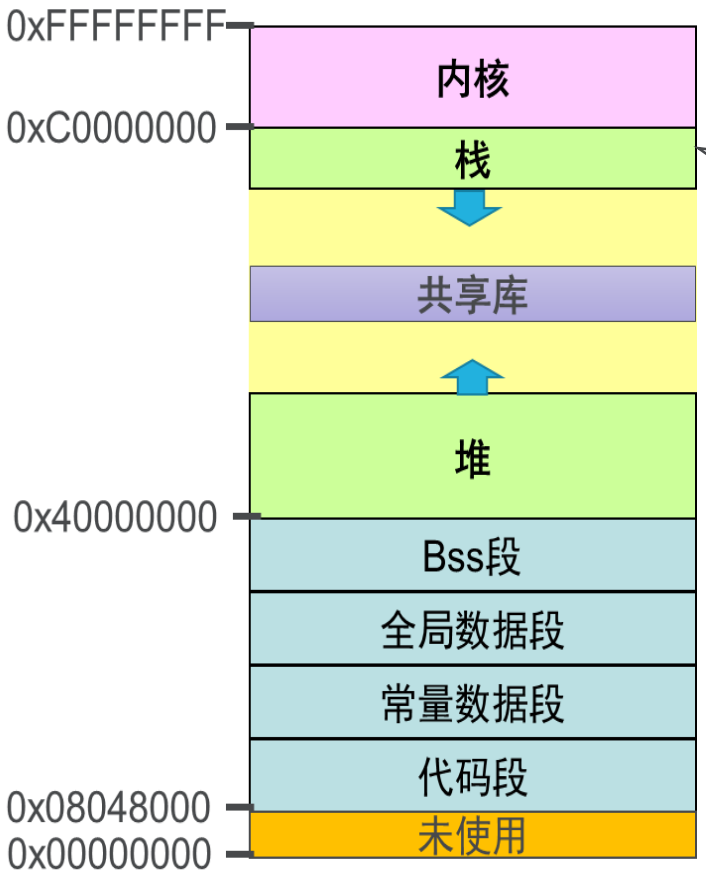
如何劫持控制流



因为需要通过控制输入而控制程序控制流，只能在程序数据区寻找控制程序控制流的渠道

控制流劫持

如何劫持控制流

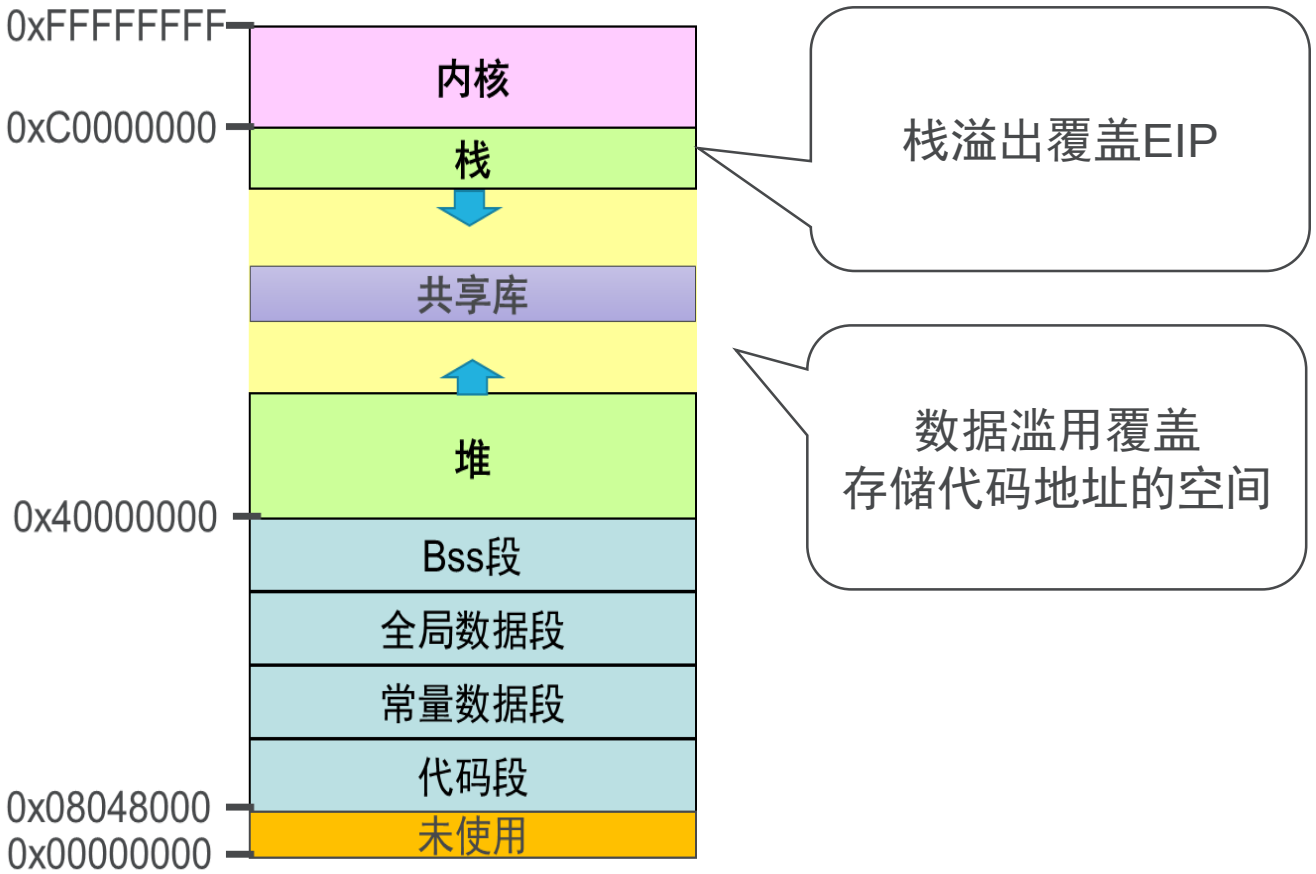


“返回地址”数据借助CPU自动的控制函数返回时将要执行的指令

数据区存储的代码地址，可以借助间接跳转指令实现控制流的改变

控制流劫持

如何劫持控制流



课程内容-1

主题参考资料参见课程网站

源代码漏洞分析

基于中间表示的分析

- 数据流分析
- 符号执行
- 污点分析

基于逻辑推理的分析

- 模型检测
- 定理证明

二进制漏洞分析

二进制静态分析

- 基于模式的漏洞分析
- 基于二进制代码比对的漏洞分析

二进制动静结合分析

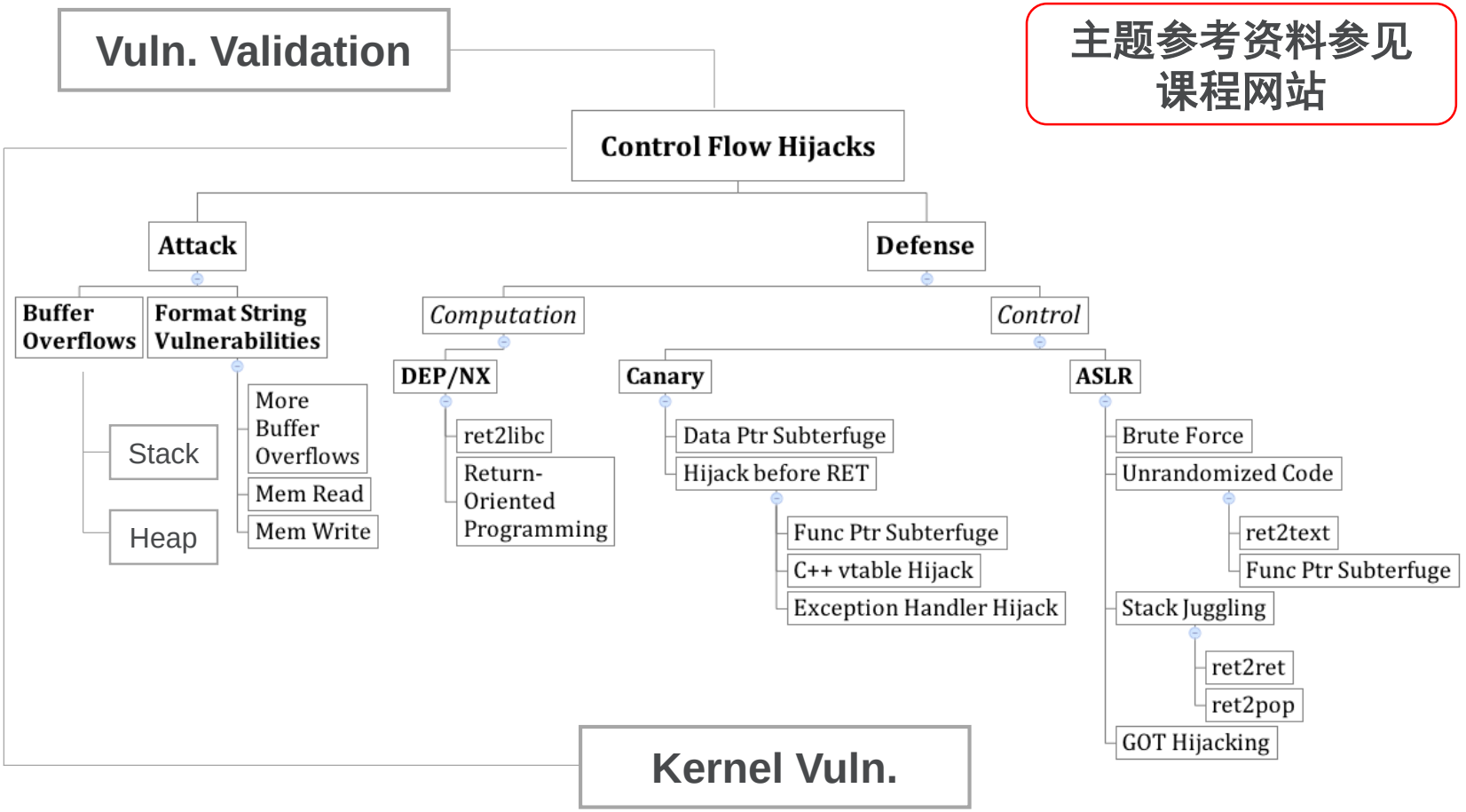
- 智能灰盒测试
- 动态污点分析

二进制动态分析

- 随机模糊测试
- 智能模糊测试

课程内容-2

主题参考资料参见
课程网站



漏洞利用与攻防实践