软件安全与漏洞分析

2.1 缓冲区溢出漏洞 ---- 栈溢出

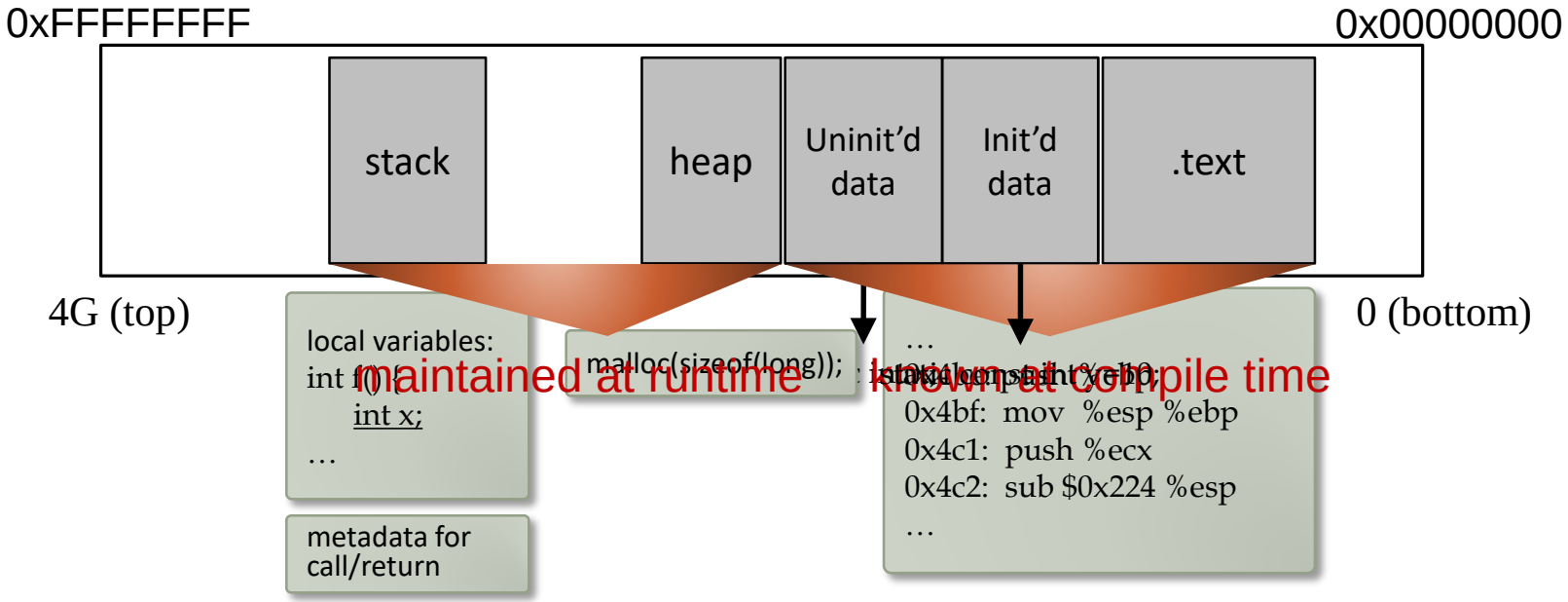
Previously in Software Security

- 典型操作系统中的进程空间布局
- 软件的加载和寻址
- 可执行文件的构造与代码重用

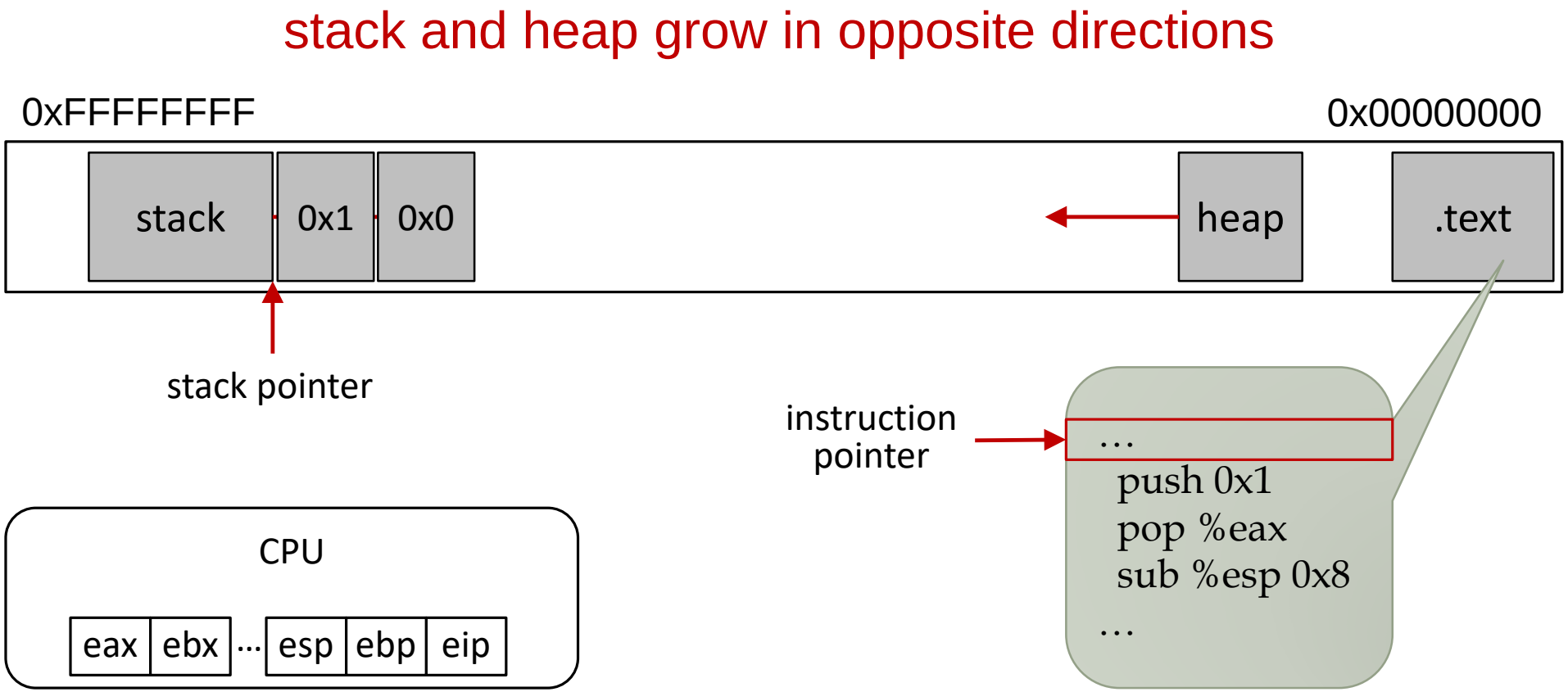
缓冲区溢出 --- 栈溢出

- 本节主题 -- 1. 栈的构造与动态行为
  - 栈帧
  - 函数的调用/返回
- 本节主题 -- 2. 栈溢出
  - 原理和案例
  - 现有应对策略与技术

回顾：进程的内存布局


0xFFFFFFFF
0x00000000
stack
heap
Uninit'd data
Init'd data
.text
4G (top)
0 (bottom)
local variables:
int f() {
int x;
…
malloc(sizeof(long));
…
0x4bf: mov %esp %ebp
0x4c1: push %ecx
0x4c2: sub $0x224 %esp
…
metadata for call/return
maintained at runtime
known at compile time

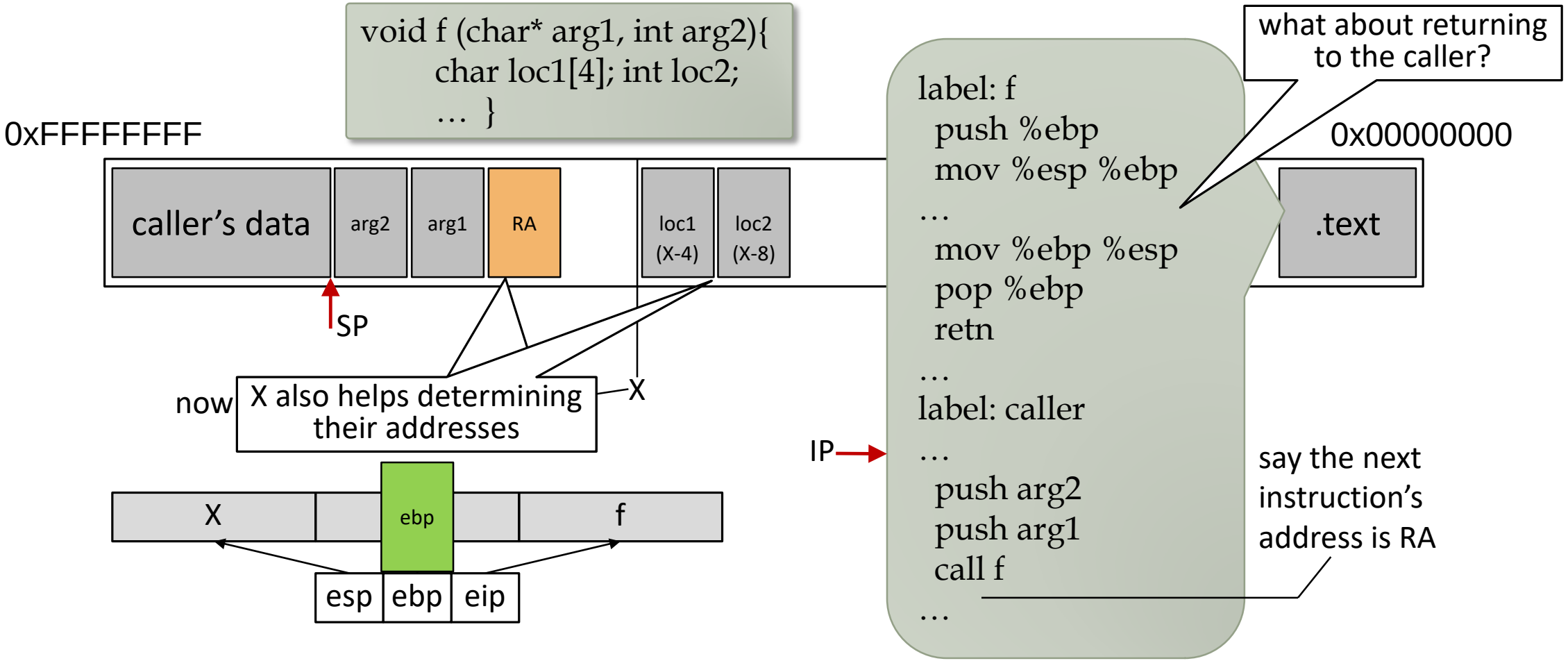
栈的工作原理


stack and heap grow in opposite directions
0xFFFFFFFF
0x00000000
stack
0x1
0x0
heap
.text
stack pointer
instruction pointer
…
push 0x1
pop %eax
sub %esp 0x8
…
CPU
eax
ebx
…
esp
ebp
eip

栈的工作原理

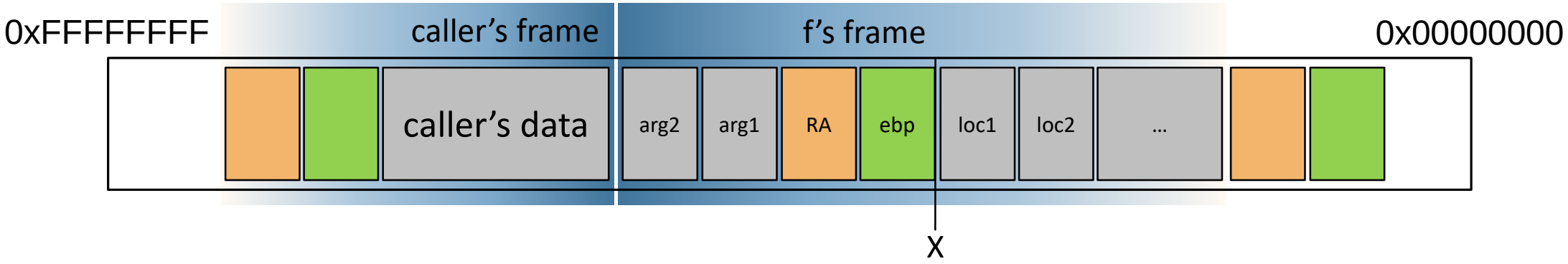
- 栈的作用：为函数的调用和执行维护存储结构
- 函数调用时
  - 需要**存储**哪些数据？
  - 这些数据存储在什么位置？
- 函数返回时
  - 需要**恢复**什么数据？
  - 从何处找到这些待恢复的数据？

栈的工作原理


void f (char* arg1, int arg2){
char loc1[4]; int loc2;
… }
0xFFFFFFFF
caller's data
arg2
arg1
RA
loc1
(X-4)
loc2
(X-8)
SP
now
X also helps determining
their addresses
X
X
ebp
f
esp
ebp
eip
label: f
push %ebp
mov %esp %ebp
…
mov %ebp %esp
pop %ebp
retn
…
label: caller
…
push arg2
push arg1
call f
…
IP
what about returning
to the caller?
0x00000000
.text
say the next
instruction's
address is RA

栈的工作原理

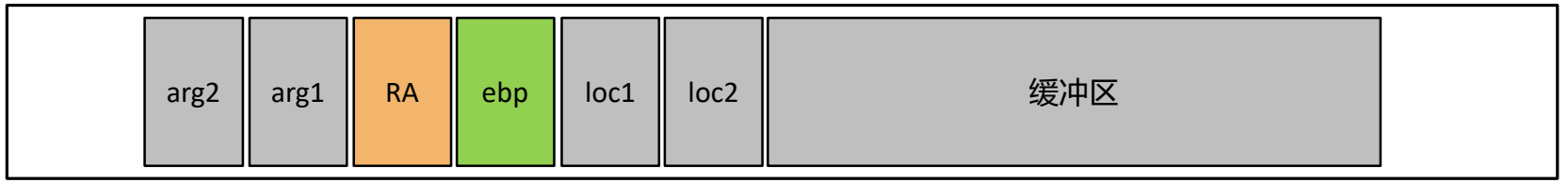
- 可见：**调用函数将在栈上创建一个结构化空间**
  - 称为**栈帧**
  - **锚点**：函数被调用时的栈指针位置为
  - 可用于对函数的**局部数据寻址 / 返回到调用者**



栈溢出

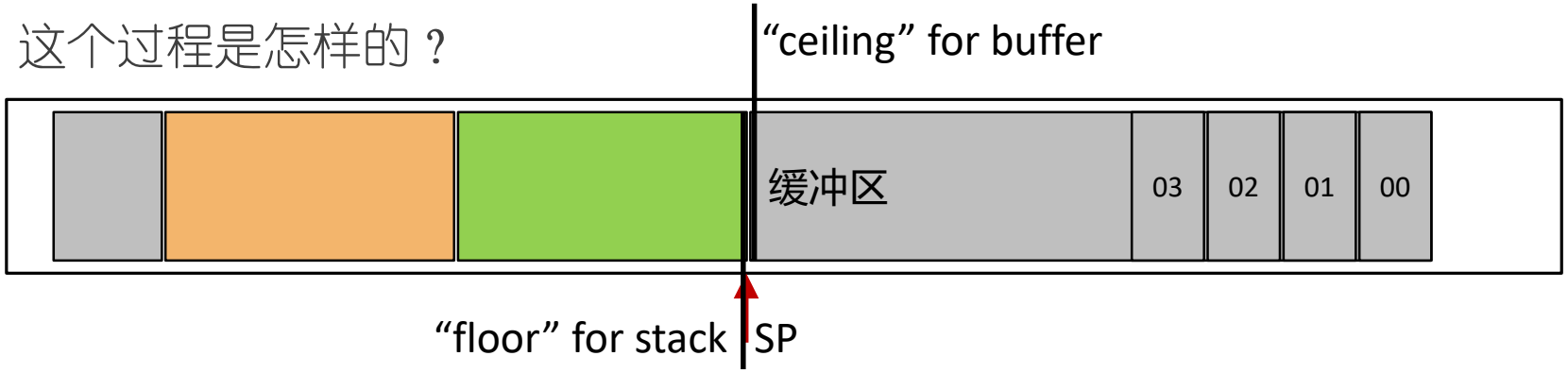
- 从**缓冲区**开始
  - 记得**“copy-on-write”**？
  - Assumed case：对一份文件进行压缩

    Solution = 反复读写 / 按块内存处理+少量读写？
  - 很多时候，函数执行中需要局部分配较大的线性内存空间



栈溢出

- 设**char** num[4]={0x00, 0x01, 0x02, 0x03}是某函数的局部变量
  - 如果建立一个缓冲区，然后将num写入缓冲区
  - 问：这个过程是怎样的？



- 所以
  - 栈增长方向 --- High → Low
  - 缓冲区增长方向 --- Low → High

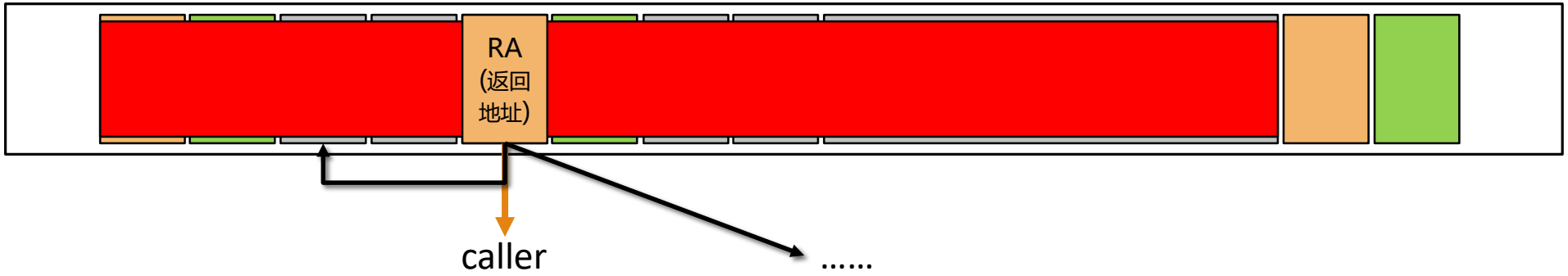
栈溢出

- “蜾蠃以为器，当其元，有器之用。凿户牖以为室，当其无，有室之用”
  - 缓冲区应该**足够大**，且写入数据前应该**先检查是否仍有剩余空间可用**
  - 然而：**思维盲区 + 设计赶不上发展**

设计一个用于输入身份证的输入框 → 18位 + ~~数字0~9~~ + ~~输入不可能更长~~

栈溢出

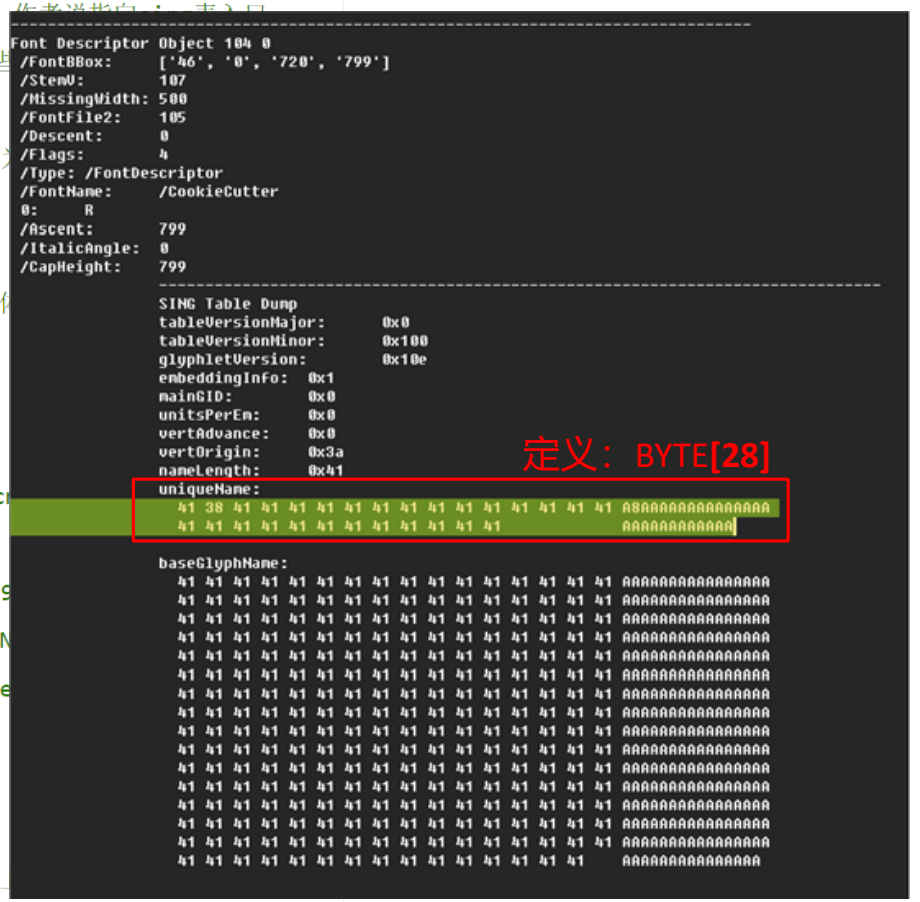
- 一旦缓冲区数据“飞跃天花板”……



- 像忽略边界检查这样的“愚蠢错误”难道不是很容易避免的吗？
  - 开发者有时**只考虑到**了“被操作对象的结构是合法的”这**一种情况**
  - 二进制代码的底层细节较为复杂，容易使人**忽略问题的存在**
  - 一些重要的**遗产代码（legacy code）**在最初的设计中存在此类缺陷

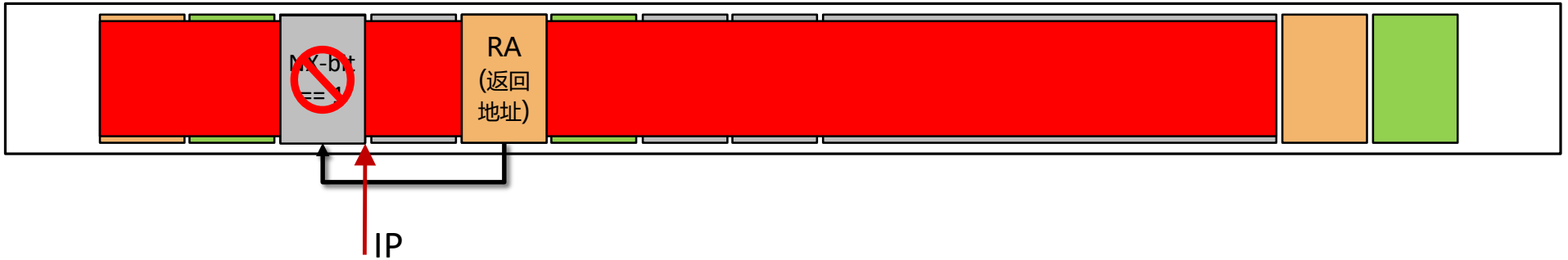
栈溢出

```
34  .text:0803DD7A              lea     ecx, [ebp+108h+var_12C] ;这个是this指针，
35  .text:0803DD7D              call    sub_8021B06       ;里面好像是将"SING"跟一些
36  .text:0803DD82              mov     eax, [ebp+108h+var_12C]
37  .text:0803DD85              cmp     eax, esi    ;调用完上面的函数var_12C已经不
38  .text:0803DD87              mov     byte ptr [ebp+108h+var_10C], 2
39  .text:0803DD8B              jz      short loc_803DDC4 ;比较结果不为0，不跳
40  .text:0803DD8D              mov     ecx, [eax] ;获取sing表的第一个4字节，即字
41  .text:0803DD8F              and     ecx, 0FFFFh ;and的结果为0
42  .text:0803DD95              jz      short loc_803DD9F ;所以这里跳转
43  .text:0803DD97              cmp     ecx, 100h
44  .text:0803DD9D              jnz     short loc_803DDC0 ;字节逆序是版本是100，c
45  .text:0803DD9F
46  .text:0803DD9F loc_803DD9F:                           ; CODE XREF: sub_803DCF9
47  .text:0803DD9F              add     eax, 10h          ;偏移0x10字节，为uniqueN
48  .text:0803DDA2              push    eax               ;  源地址是指向uniqueName
49  .text:0803DDA3              lea     eax, [ebp+108h+var_108]
50  .text:0803DDA6              push    eax               ;  目标地址是栈上
51  .text:0803DDA7              mov     [ebp+108h+var_108], 0
52  .text:0803DDAB              call    strcat            ;将uniqueName域连接到栈
```



现有栈溢出防护技术

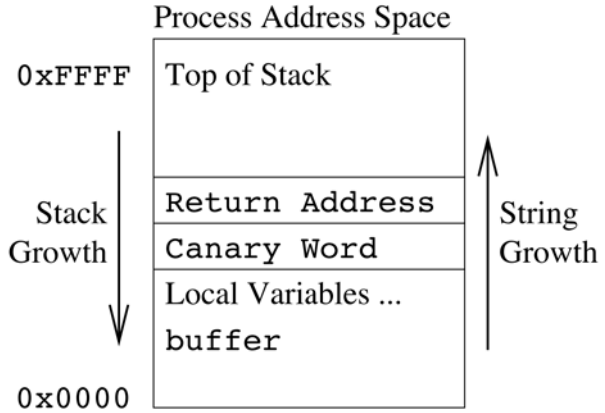
- 数据执行保护（**D**ata **E**xecution **P**revention）
  - 一些别名：W⊕X，NX-bit
  - 机制：对内存页面增加一个标识bit，使之**要么可改写，要么可执行**



  - **Shortcoming**：只要不试图执行数据区内容，则既不能阻止溢出，也不能发现溢出

现有栈溢出防护技术

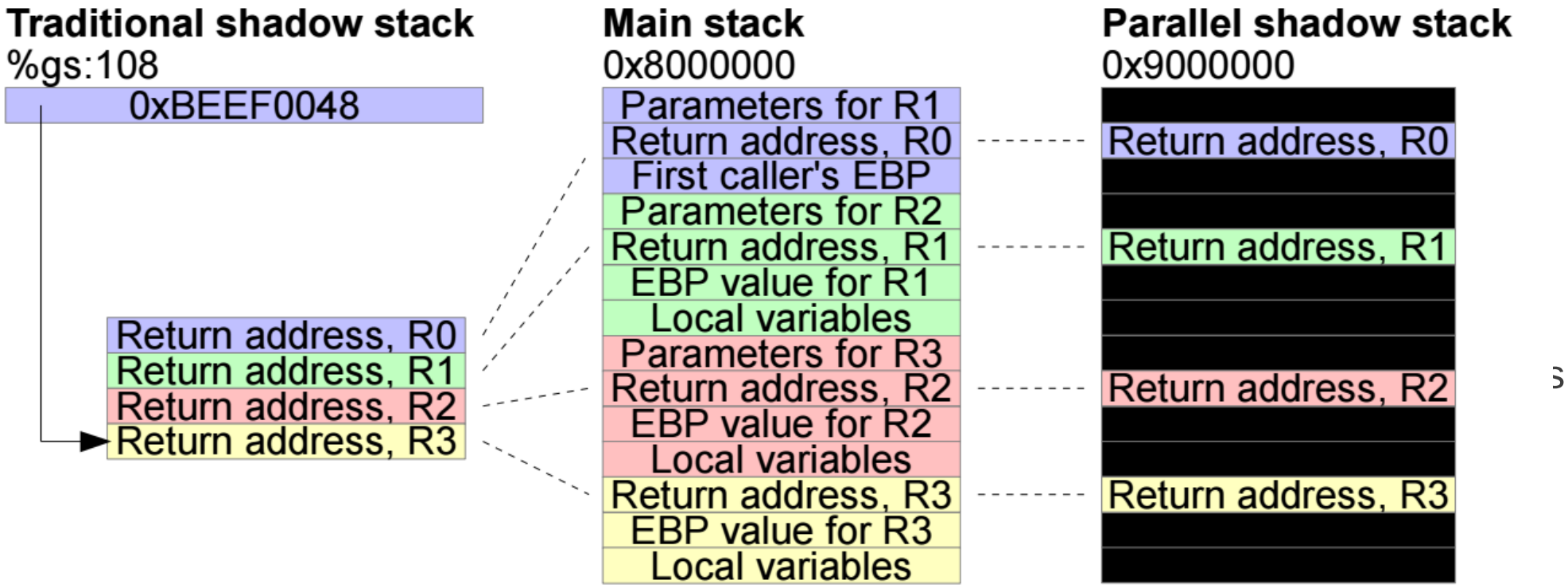
- 栈帧上的“金丝雀”（stack canaries）
  - 机制：栈上设置检查点，**检查点数值异常 = 栈溢出发生**



  - **Shortcomings**：开销大，对遗产代码不起作用
  - 参考文献：Cowan C, Pu C, Maier D, et al. Stackguard: automatic adaptive detection and prevention of buffer-overflow attacks[C]//Usenix Security. 1998, 98: 63-78.

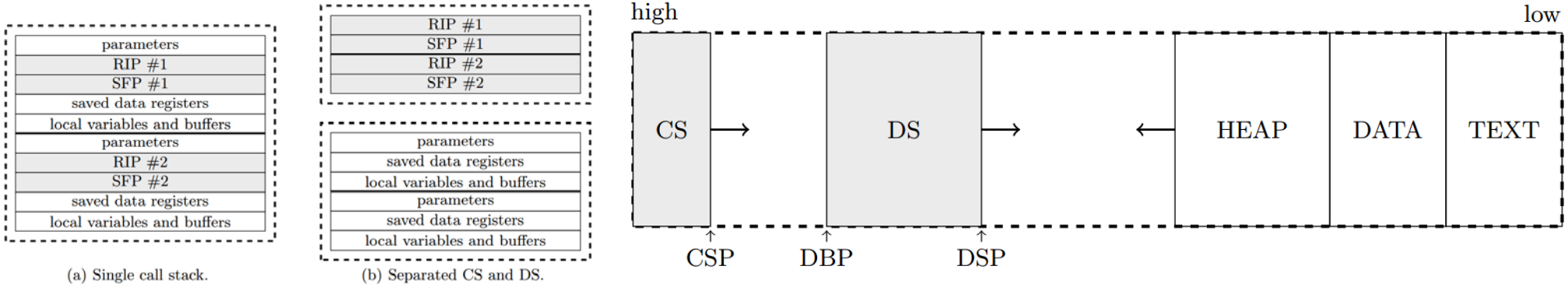
现有栈溢出防护技术

- 影子栈（shadow stack）



现有栈溢出防护技术

- 分离的控制/数据栈（Separated Control- and Data-Stacks）
  - 机制：修改操作系统的进程内存布局，分配**两个栈空间**，分别用于**控制参数**和**数据**的存储



  - **Shortcomings**：无法保护遗产代码，不适用于完全由汇编语言构成的函数
  - 参考文献：Kugler C, Müller T. SCADS[C]//International Conference on Security and Privacy in Communication Systems. Springer International Publishing, 2014: 323-340.

What’s next?

- 堆溢出漏洞
  - 堆的工作原理
  - 堆溢出原理
  - 相关防护技术