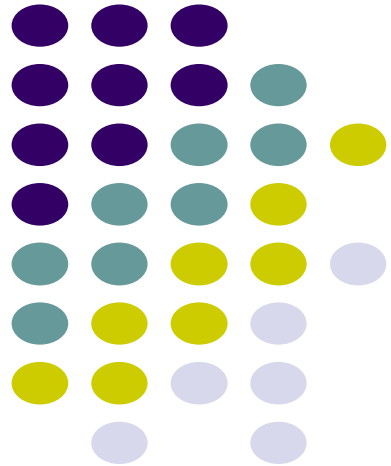
《计算机系统基础（四）：编程与调试实践》

栈和过程调用



栈和过程调用

过程调用的机器级表示

过程调用中栈和栈帧的内容变化

栈和过程调用

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

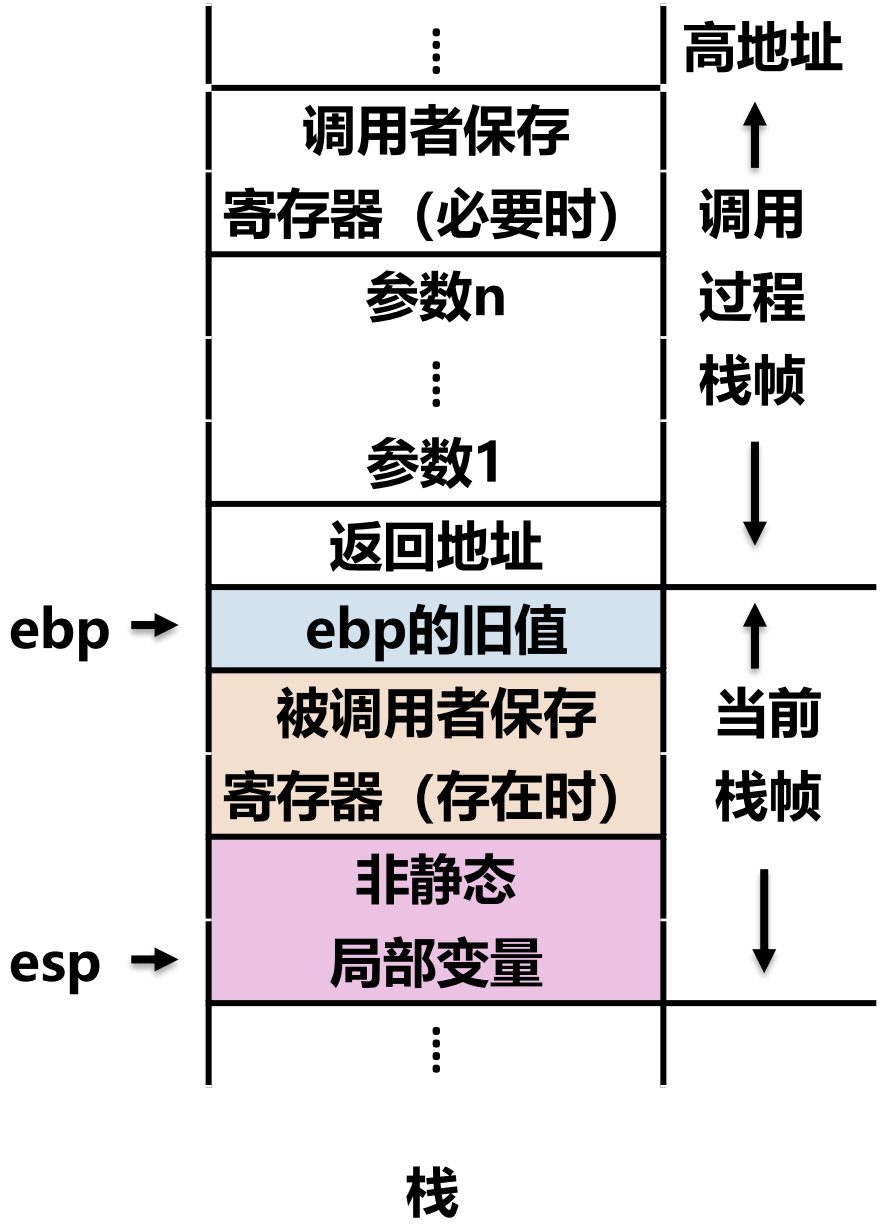
⋮

(ebx等出栈)

P的ebp出栈

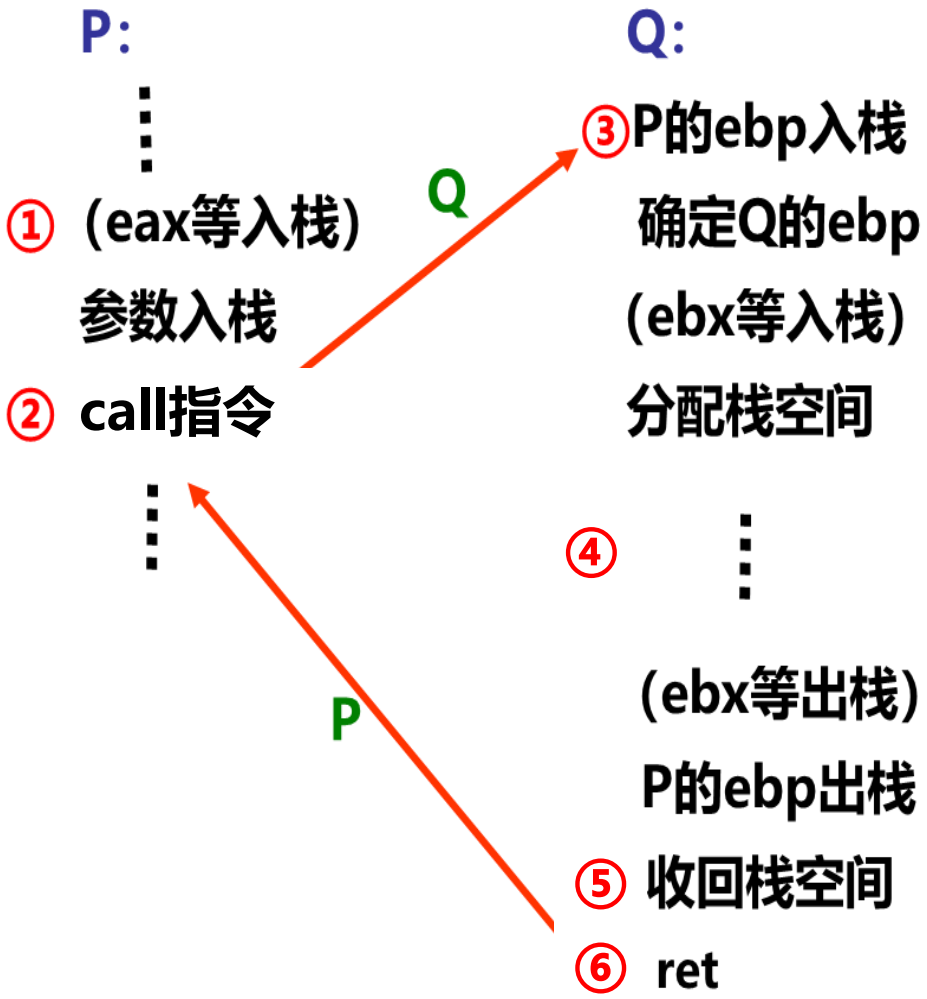
收回栈空间

ret指令



栈和过程调用

过程调用的执行步骤：



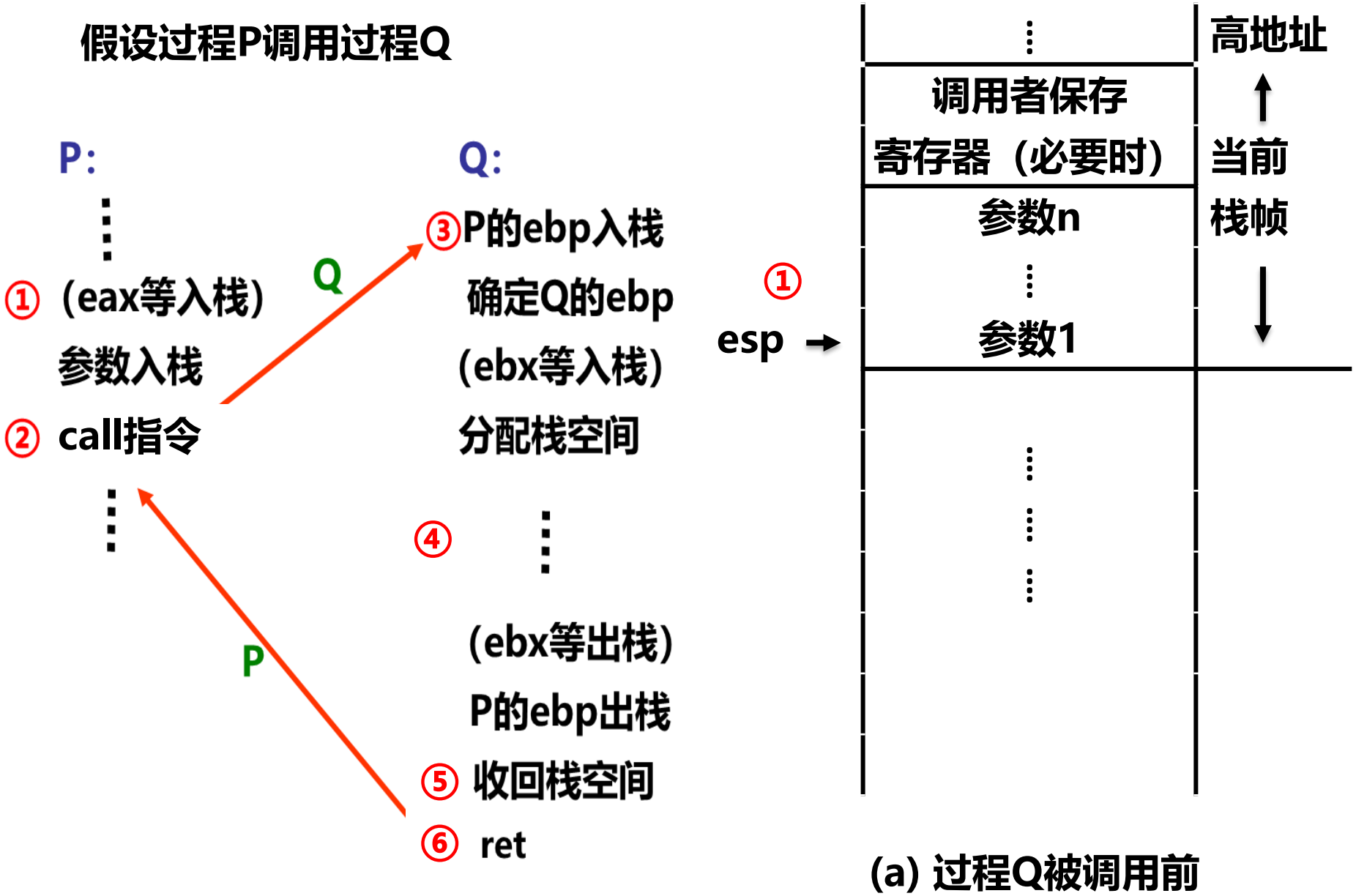
假设过程P调用过程Q

① P的准备阶段
② 从P控制转移到Q：call指令
P

③ Q的准备阶段
④ 执行Q的过程体（函数体）
⑤ Q的恢复阶段
⑥ 从Q返回到P：ret指令
Q

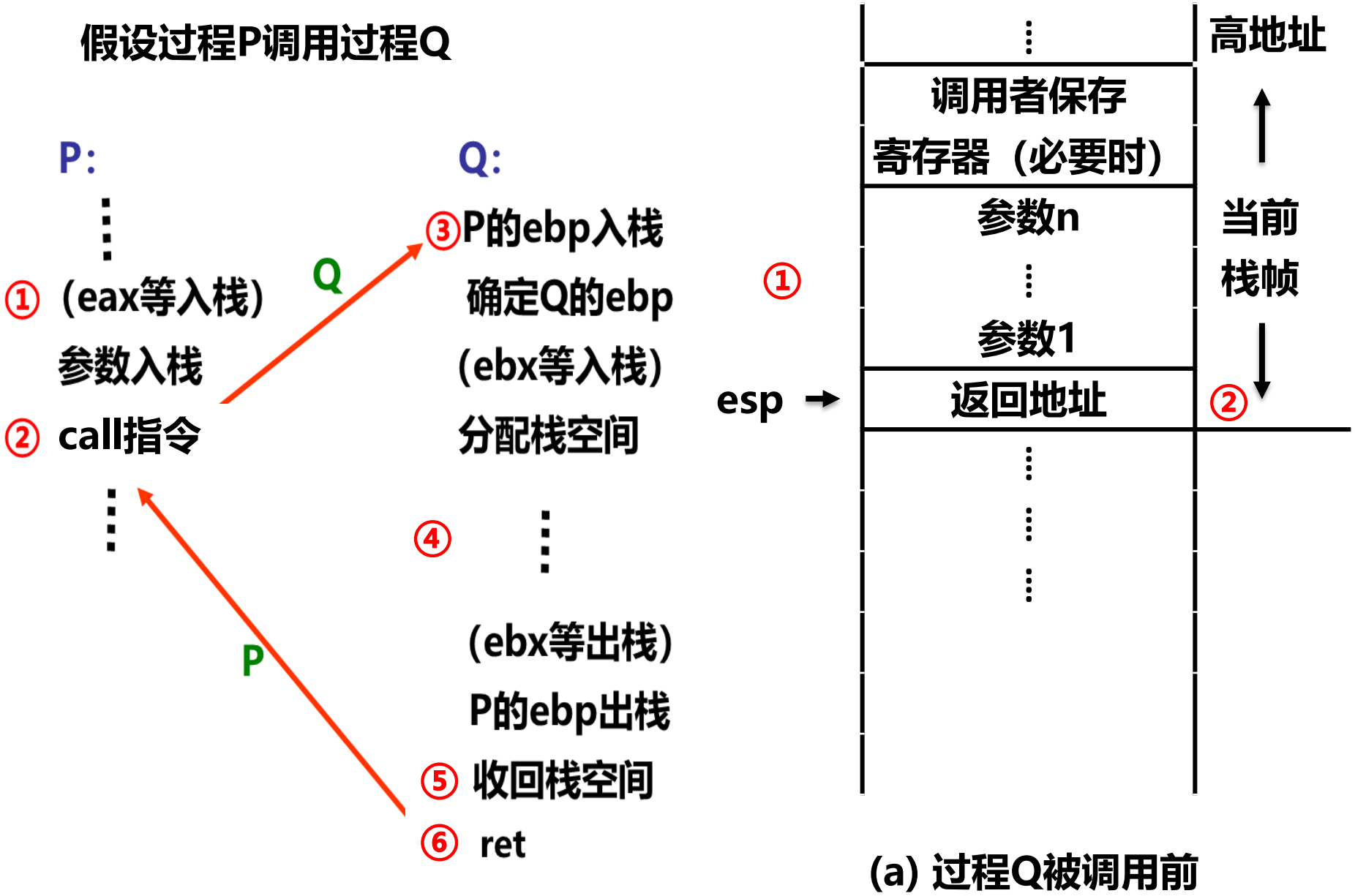
eax、ecx、edx	P 保存
ebx、esi、edi	Q 保存
ebp	Q 保存
esp	动态变化

栈和过程调用

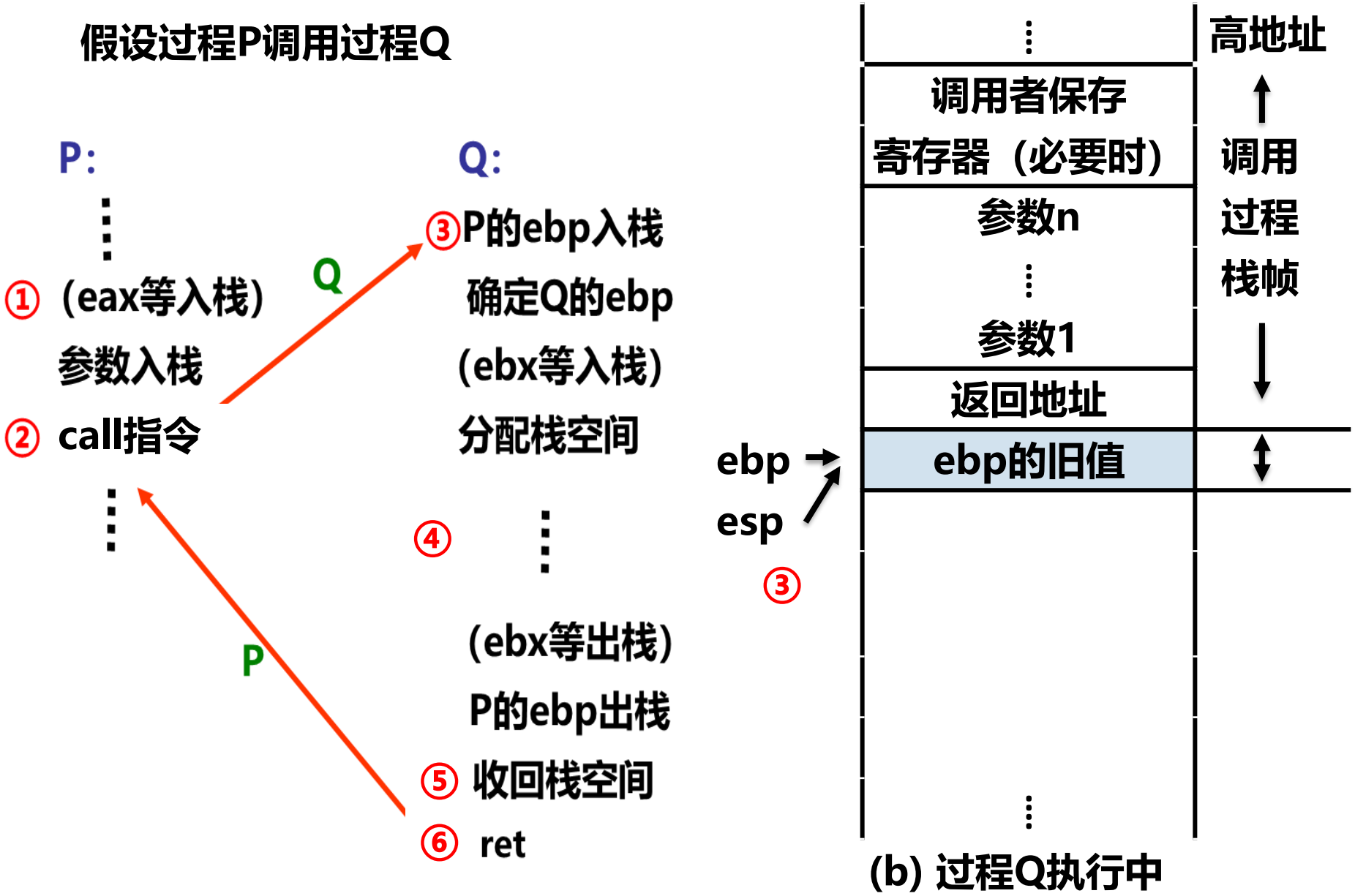


(a) 过程Q被调用前

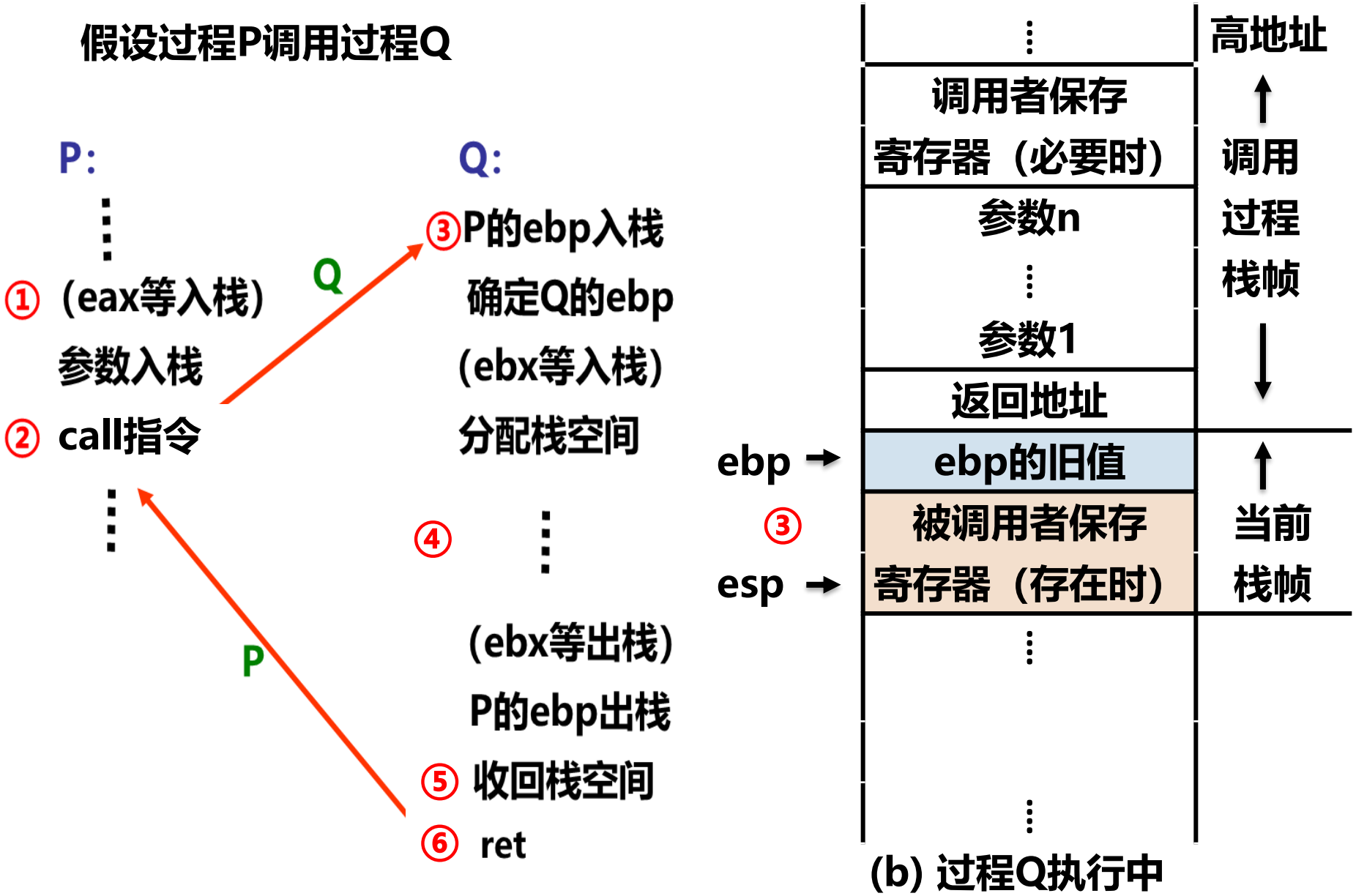
栈和过程调用


假设过程P调用过程Q
P:
①（eax等入栈）
参数入栈
② call指令
Q
③P的ebp入栈
确定Q的ebp
(ebx等入栈)
分配栈空间
④
(ebx等出栈)
P的ebp出栈
⑤ 收回栈空间
⑥ ret
P
高地址
调用者保存
寄存器（必要时）
参数n
参数1
返回地址
当前
栈帧
①
②
esp
(a) 过程Q被调用前

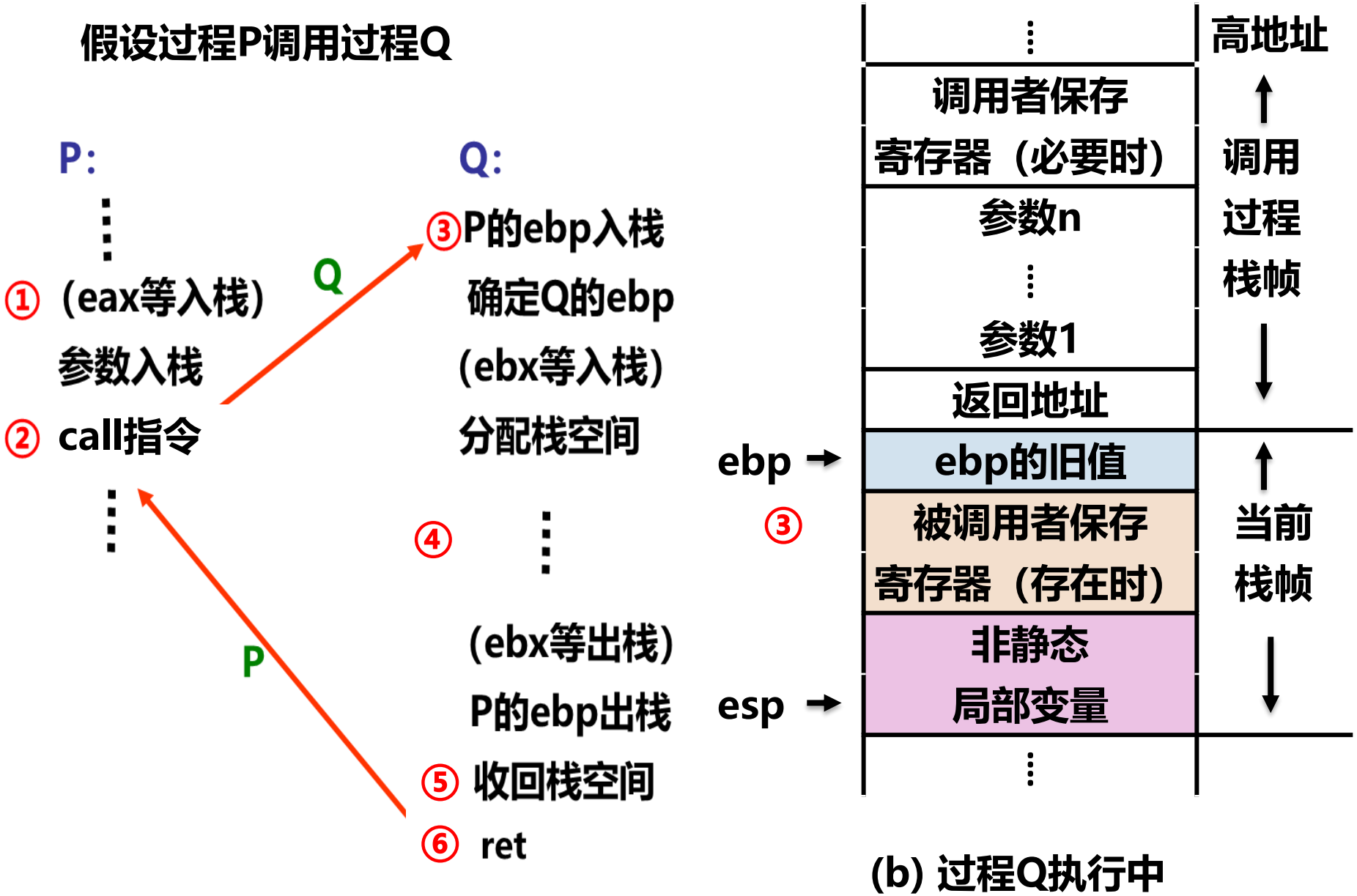
栈和过程调用


假设过程P调用过程Q
P:
①（eax等入栈）
参数入栈
② call指令
Q
③P的ebp入栈
确定Q的ebp
(ebx等入栈)
分配栈空间
④
(ebx等出栈)
P的ebp出栈
⑤ 收回栈空间
⑥ ret
P
高地址
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
调用
过程
栈帧
ebp
esp
③
(b) 过程Q执行中

栈和过程调用

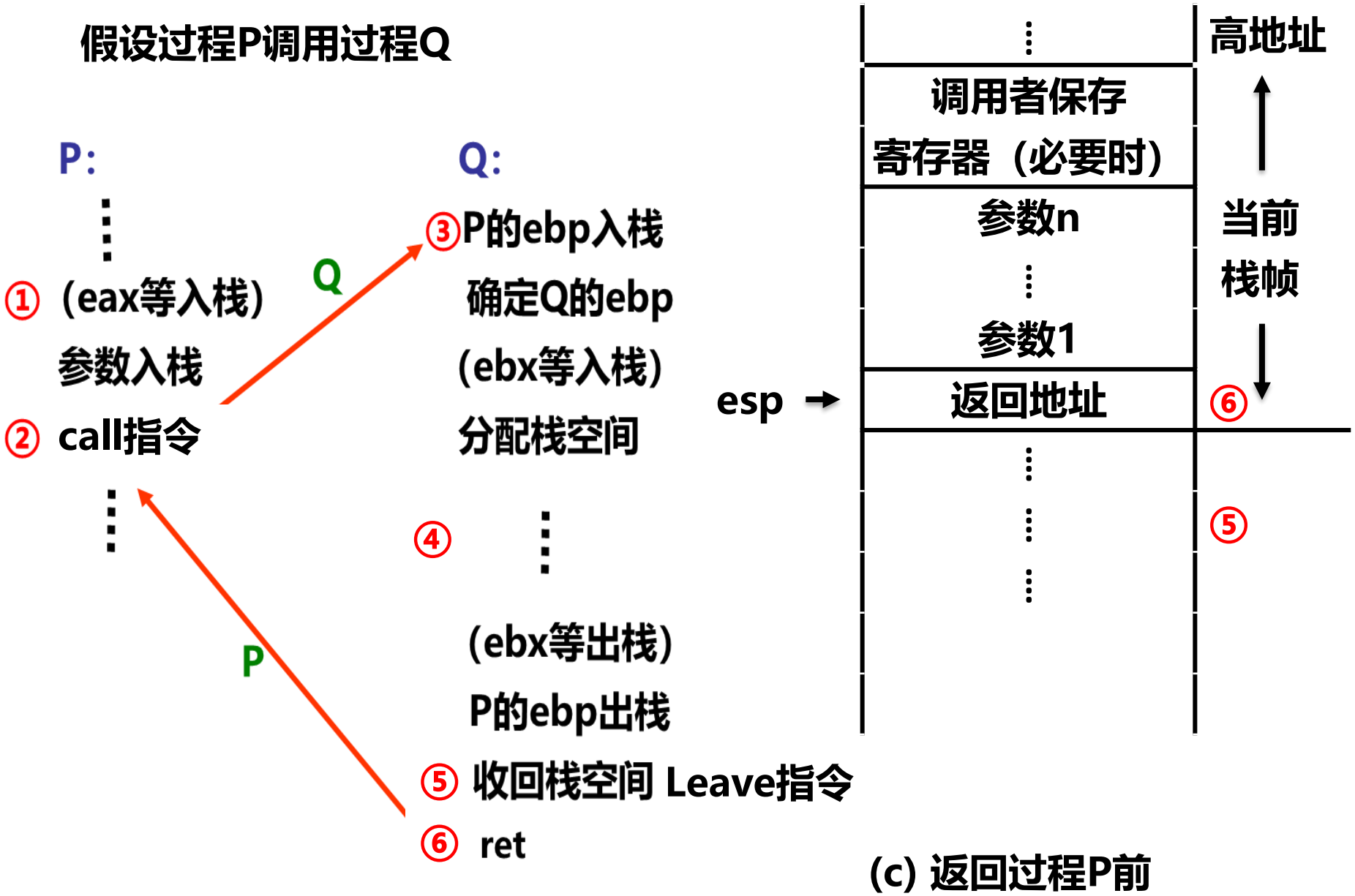

假设过程P调用过程Q
P:
①（eax等入栈）
参数入栈
② call指令
Q
P
Q:
③P的ebp入栈
确定Q的ebp
(ebx等入栈)
分配栈空间
④
(ebx等出栈)
P的ebp出栈
⑤ 收回栈空间
⑥ ret
高地址
调用者保存
寄存器（必要时）
参数n
参数1
返回地址
ebp
ebp的旧值
③
被调用者保存
esp
寄存器（存在时）
调用
过程
栈帧
当前
栈帧
(b) 过程Q执行中

栈和过程调用



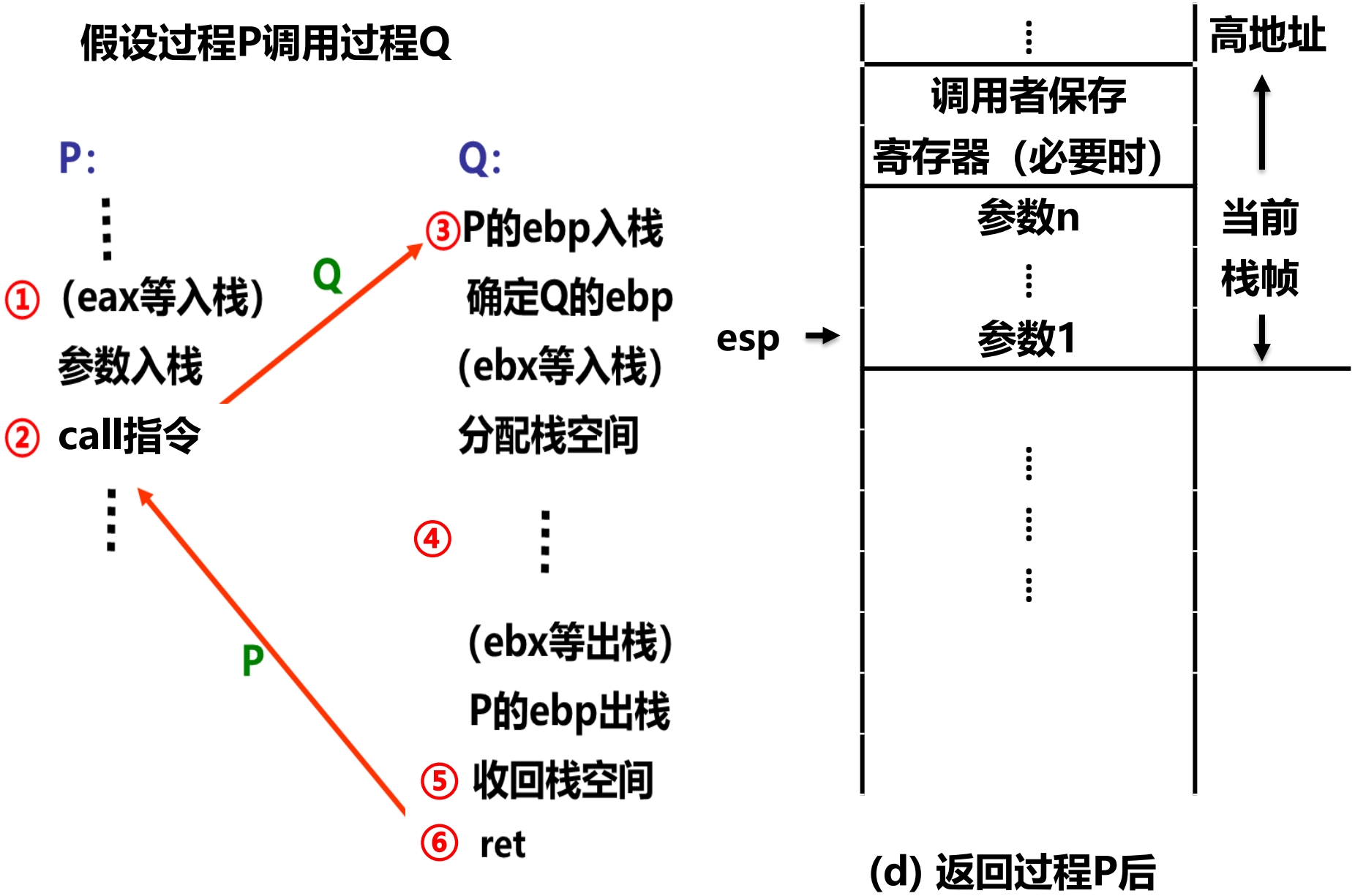
(b) 过程Q执行中

栈和过程调用



(c) 返回过程P前

栈和过程调用


假设过程P调用过程Q
P:
①（eax等入栈）
参数入栈
②call指令
Q
③P的ebp入栈
确定Q的ebp
(ebx等入栈)
分配栈空间
④
(ebx等出栈)
P的ebp出栈
⑤ 收回栈空间
⑥ ret
P
esp
调用者保存
寄存器（必要时）
参数n
参数1
高地址
当前
栈帧
(d) 返回过程P后

栈和过程调用

总结：

1. IA-32中使用栈支持过程调用

   入口参数、返回地址、被保存的寄存器内容、非静态局部变量。

2. 正在执行的过程都有自己的栈帧，过程执行结束会回收栈空间。

3. 当前栈帧的范围在ebp和esp指向的区域。

4. 过程调用的机器级表示：过程调用时call指令前后的指令

   过程开始和结束的指令

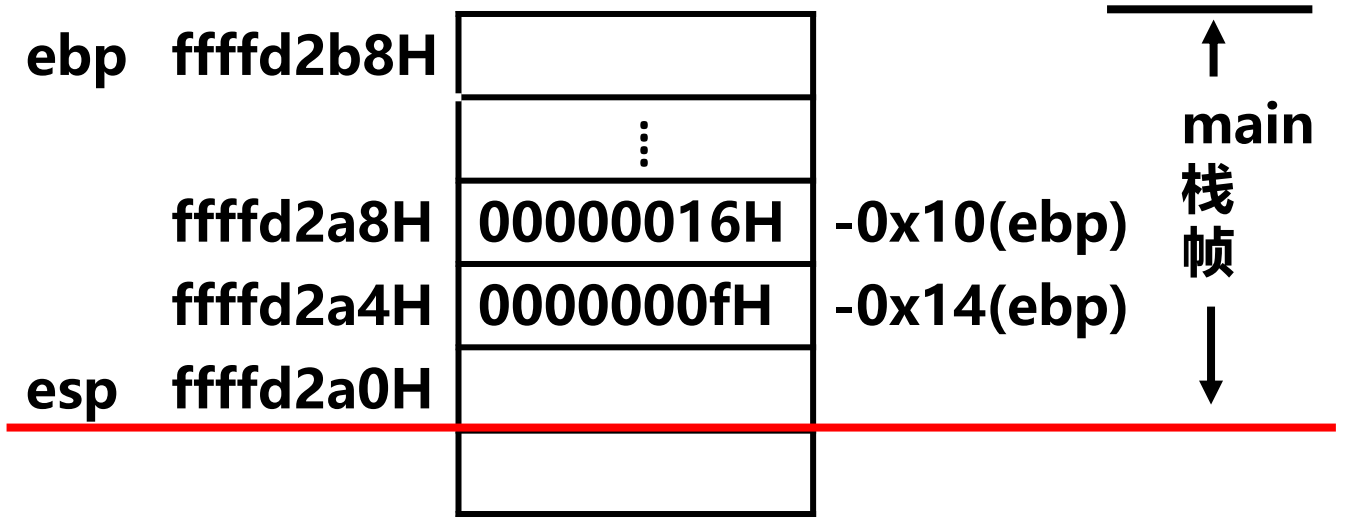
栈和过程调用–按地址传递参数示例

```
#include <stdio.h>
int swap (int *x, int *y )
{
   int t=*x;
       *x=*y;
       *y=t;
}

void main ( )
{  int a=15, b=22;
   swap (&a, &b);
   printf ("a=%d\tb=%d\n", a, b);
}
```

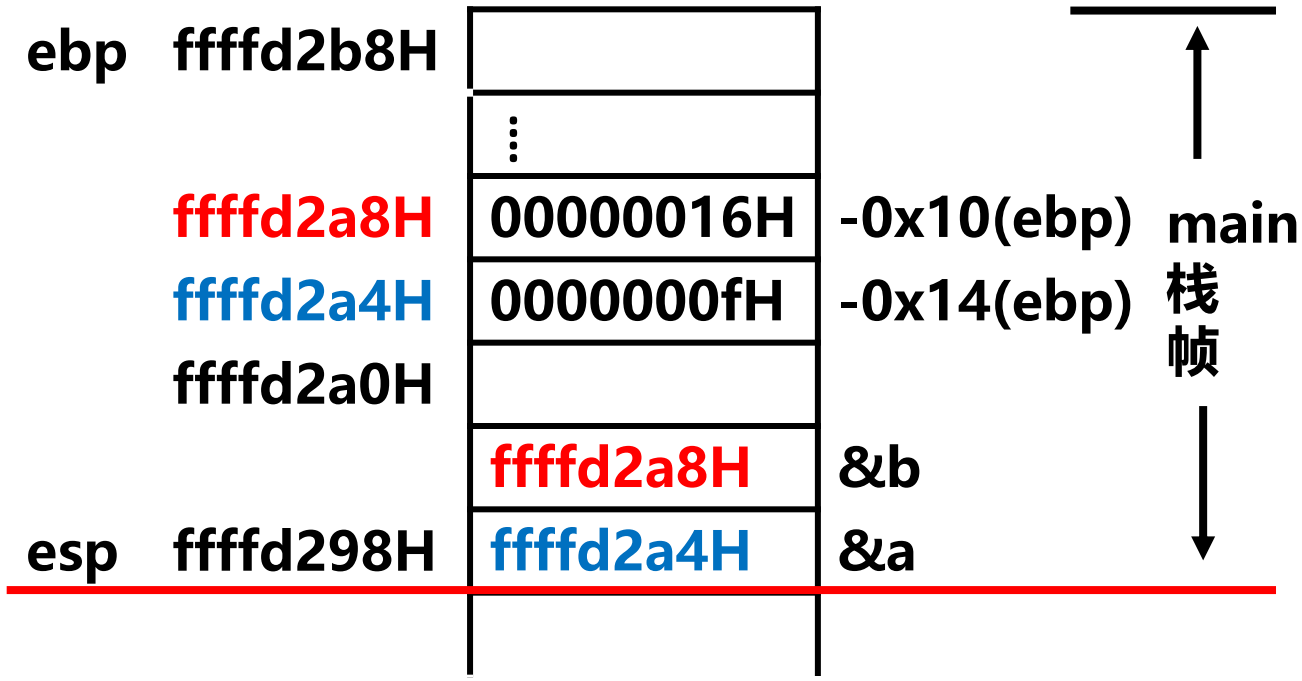
1. 打开反汇编后的文档，找出过程调用中的相关语句
2. 调试执行程序，画出过程调用中栈帧结构图，理解栈和过程调用
3. 理解参数的按地址传递含义

```
(gdb) i r ebp esp
ebp            0xffffd2b8
esp            0xffffd2a0
(gdb) x/7xw $esp
0xffffd2a0:    0x00000001    0x0000000f    0x00000016    0x0ebdd200
0xffffd2b0:    0xf7fb63fc    0xffffd2d0    0x00000000
```

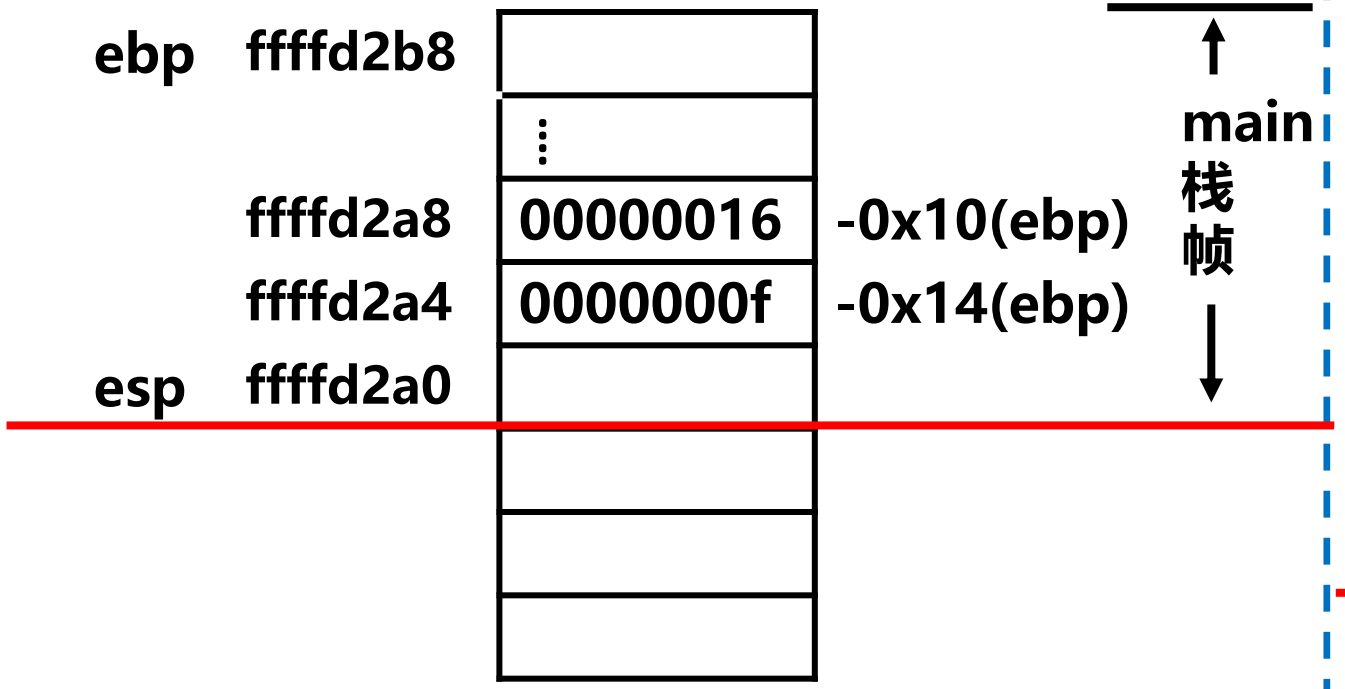


四条指令执行前的ebp、esp内容和栈帧结构

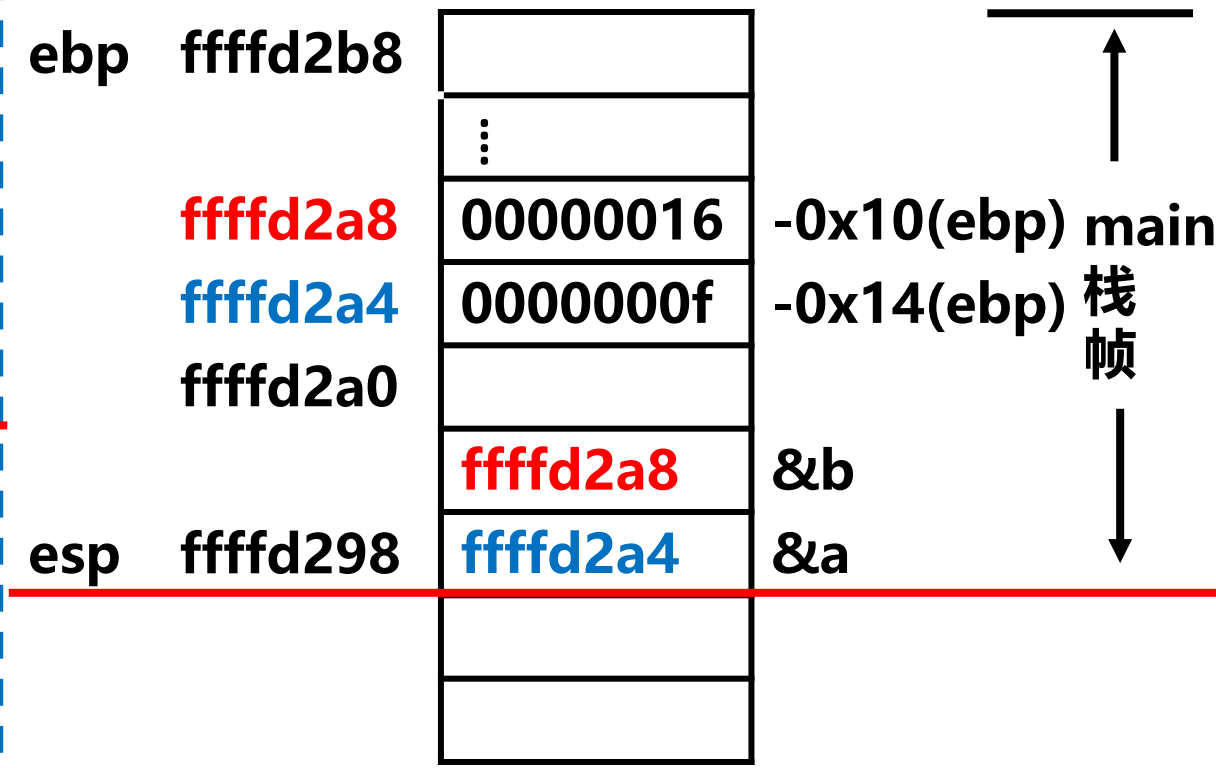
```
(gdb) i r ebp esp
ebp            0xffffd2b8
esp            0xffffd298
(gdb) x/9xw $esp
 0xffffd298:    0xffffd2a4    0xffffd2a8    0x00000001    0x0000000f
0xffffd2a8:     0x00000016    0x0ebdd200    0xf7fb63fc    0xffffd2d0
0xffffd2b8:     0x00000000
```



四条指令执行后的ebp、esp内容和栈帧结构

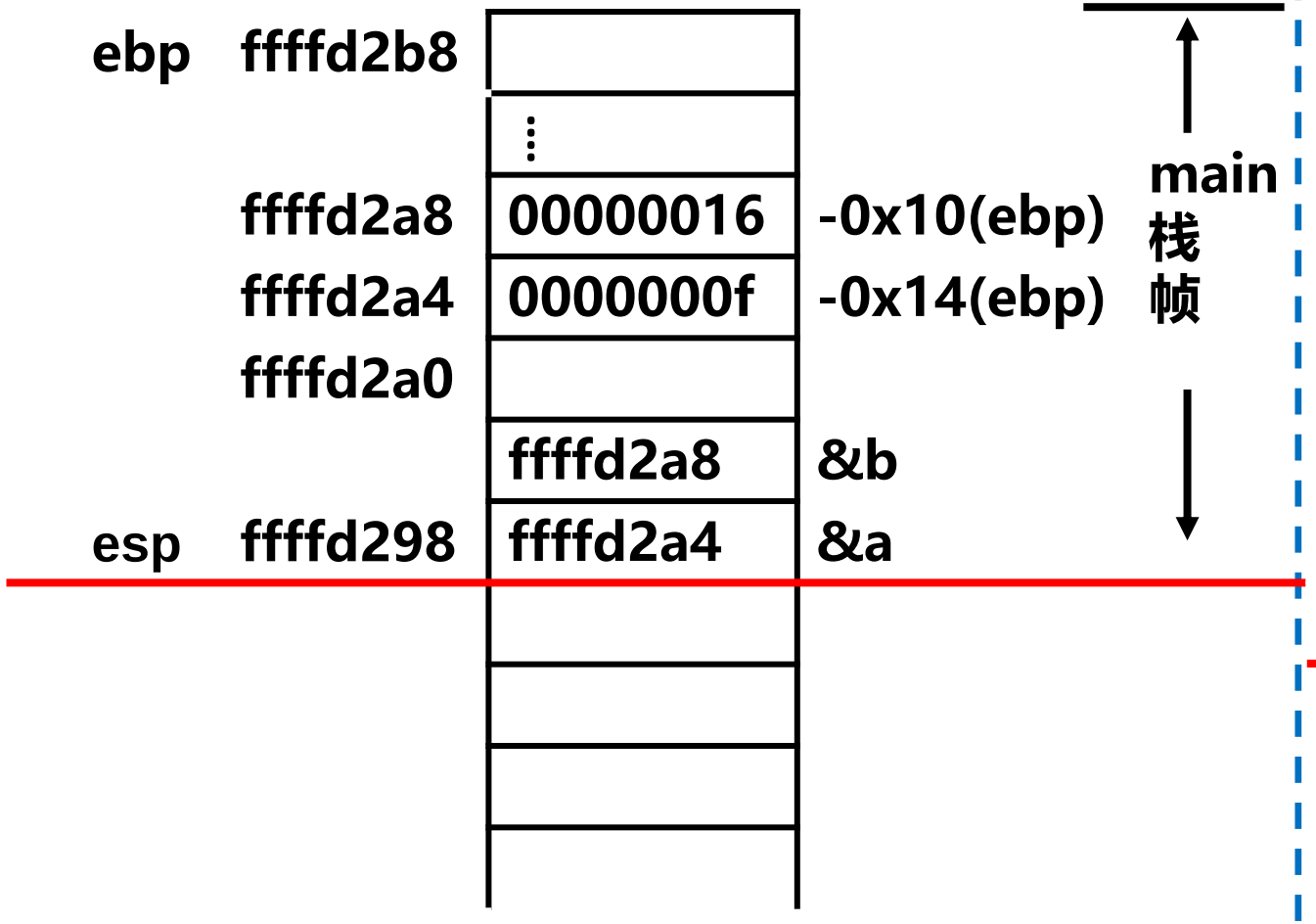


main中“swap（&a，&b）”执行前

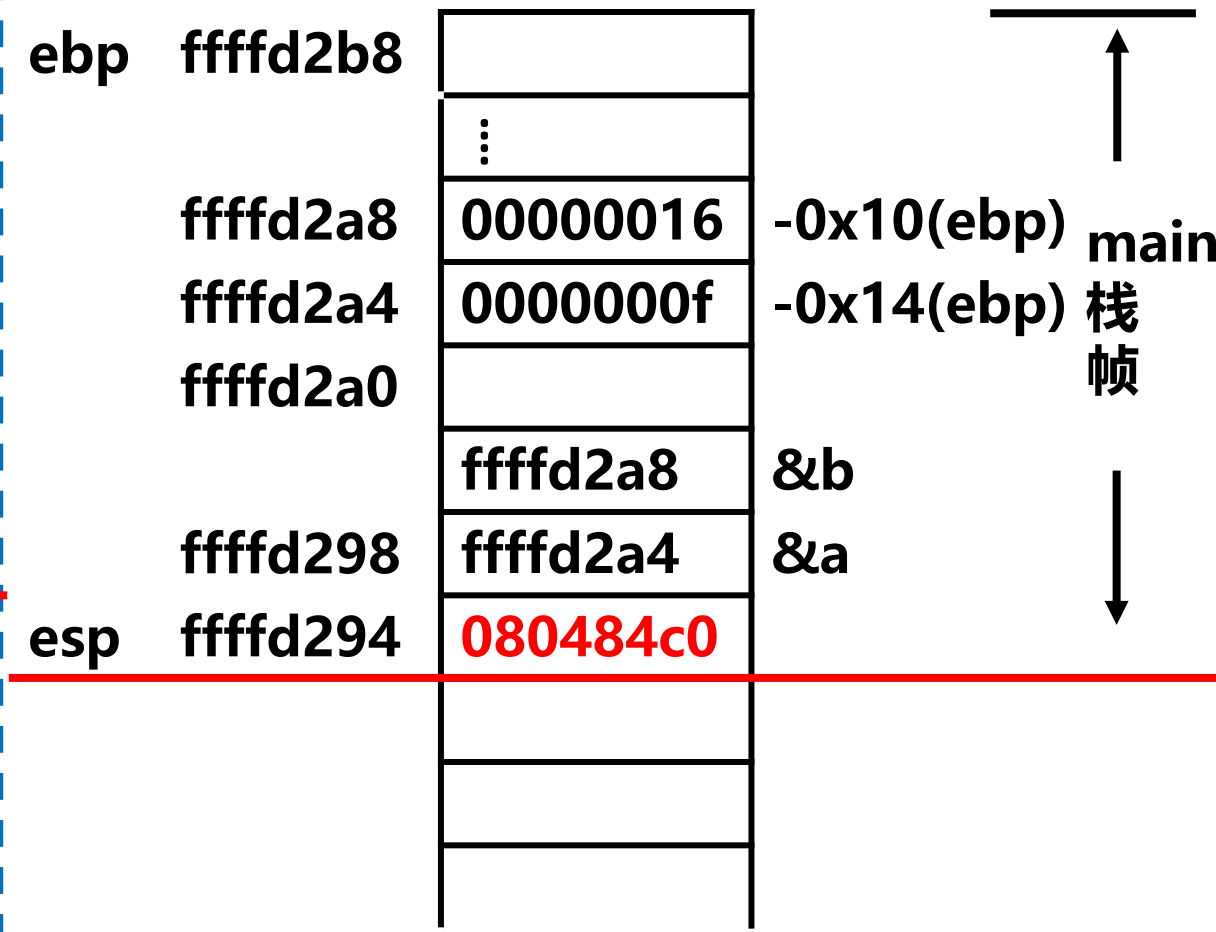


swap（&a，&b）

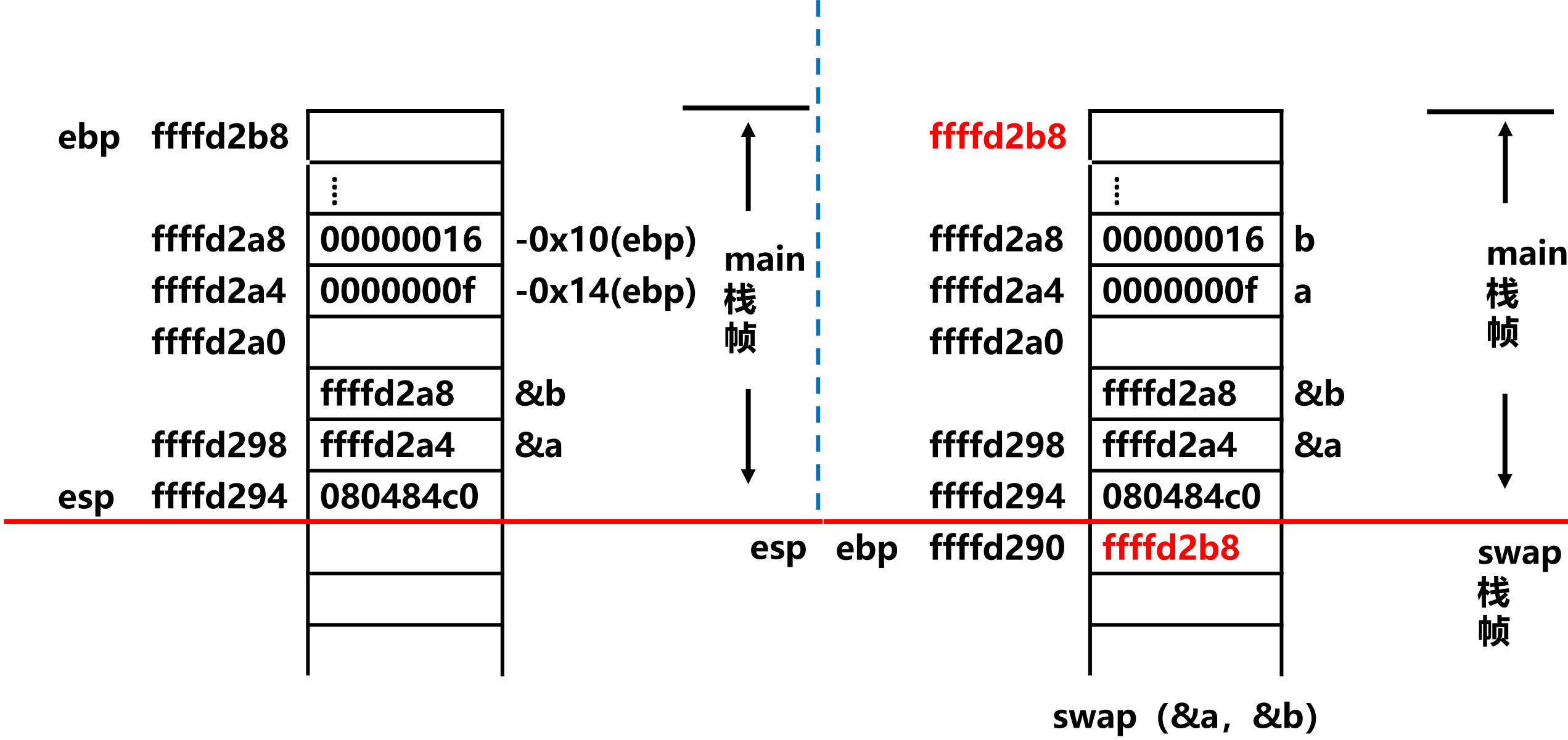
① main()准备工作：参数入栈



① main()准备工作：参数入栈

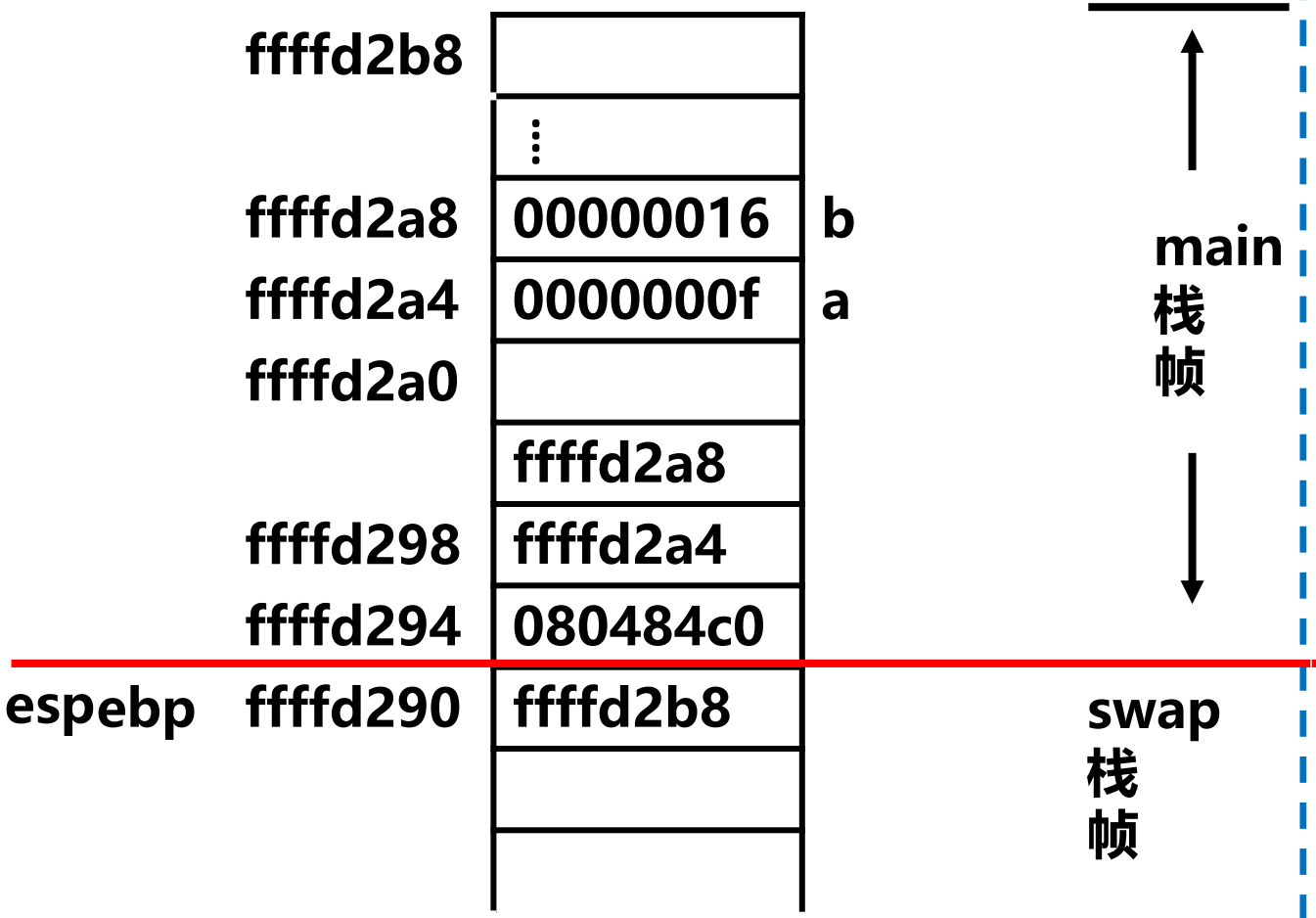


② main()执行call指令：返回地址入栈

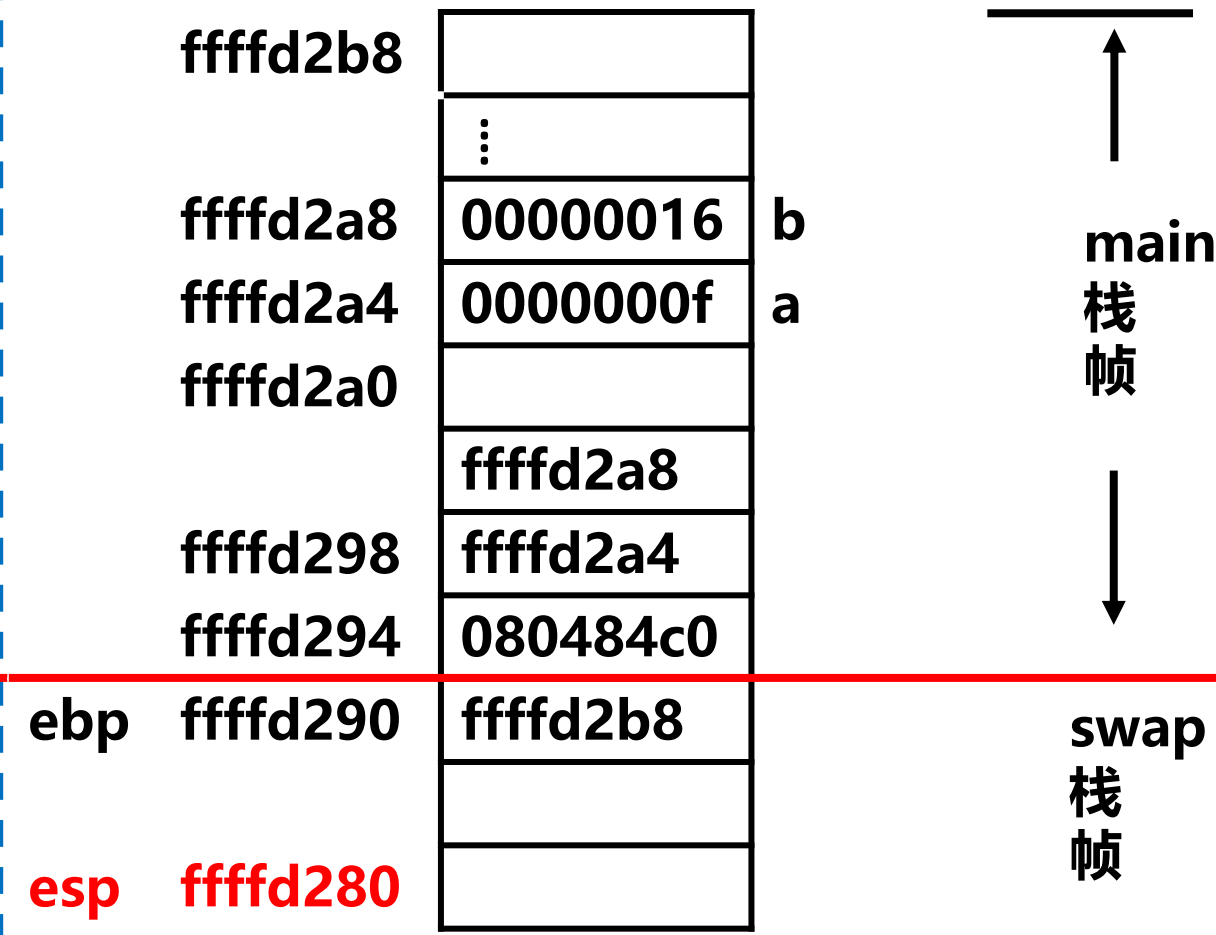


② main()执行call指令：返回地址入栈

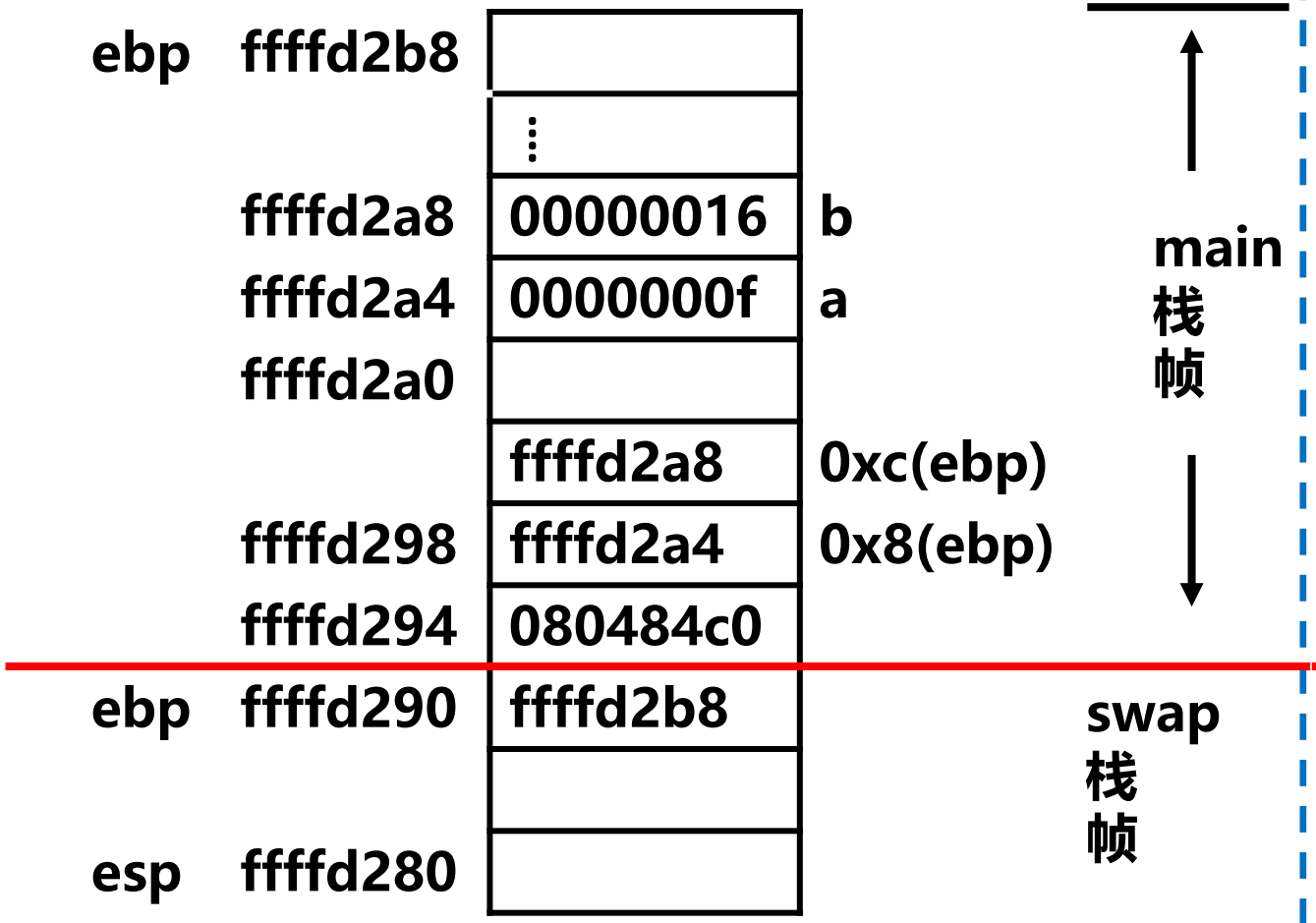
③ swap()中的准备工作：保存旧ebp值，建立新栈帧



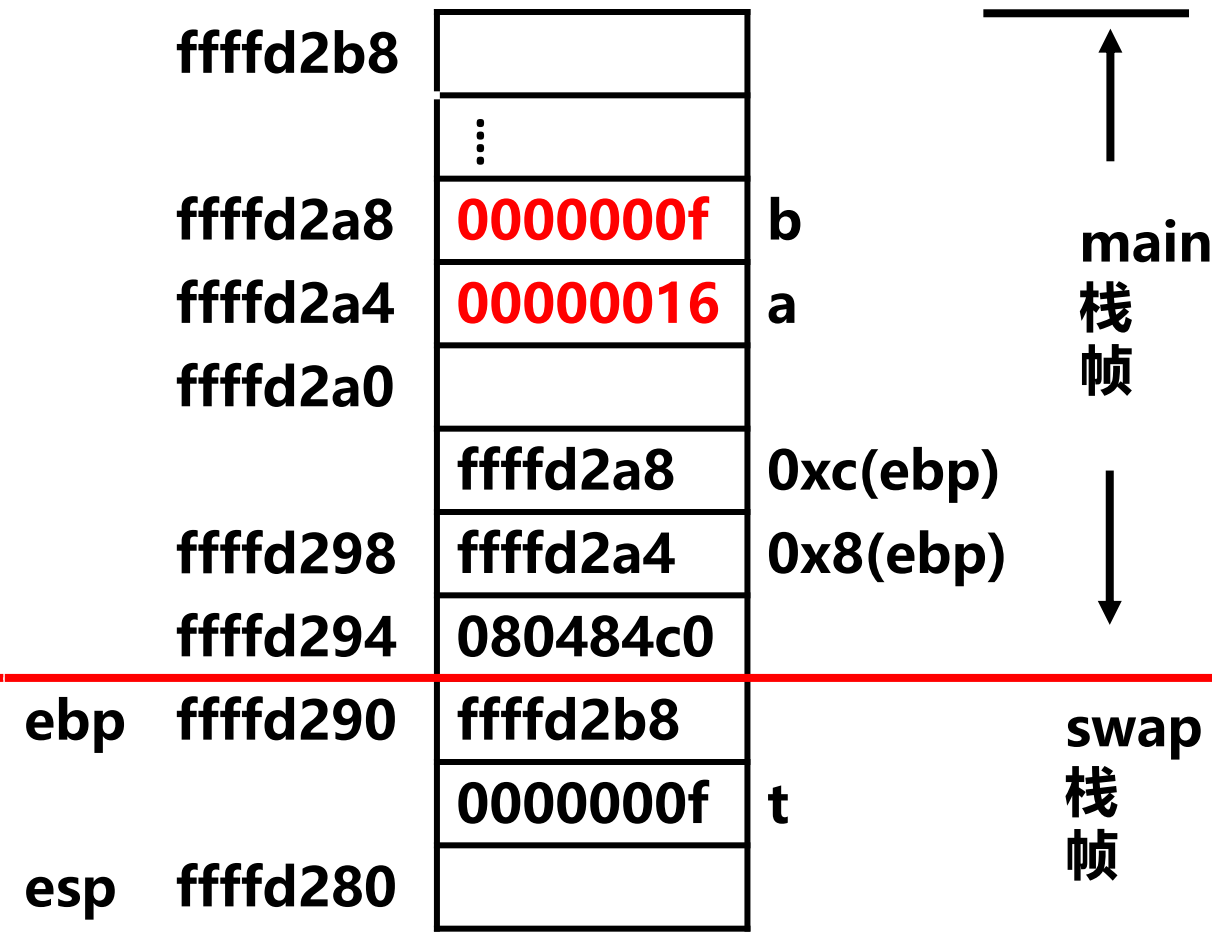
③ swap()的准备工作：保存旧ebp值，建立新栈帧



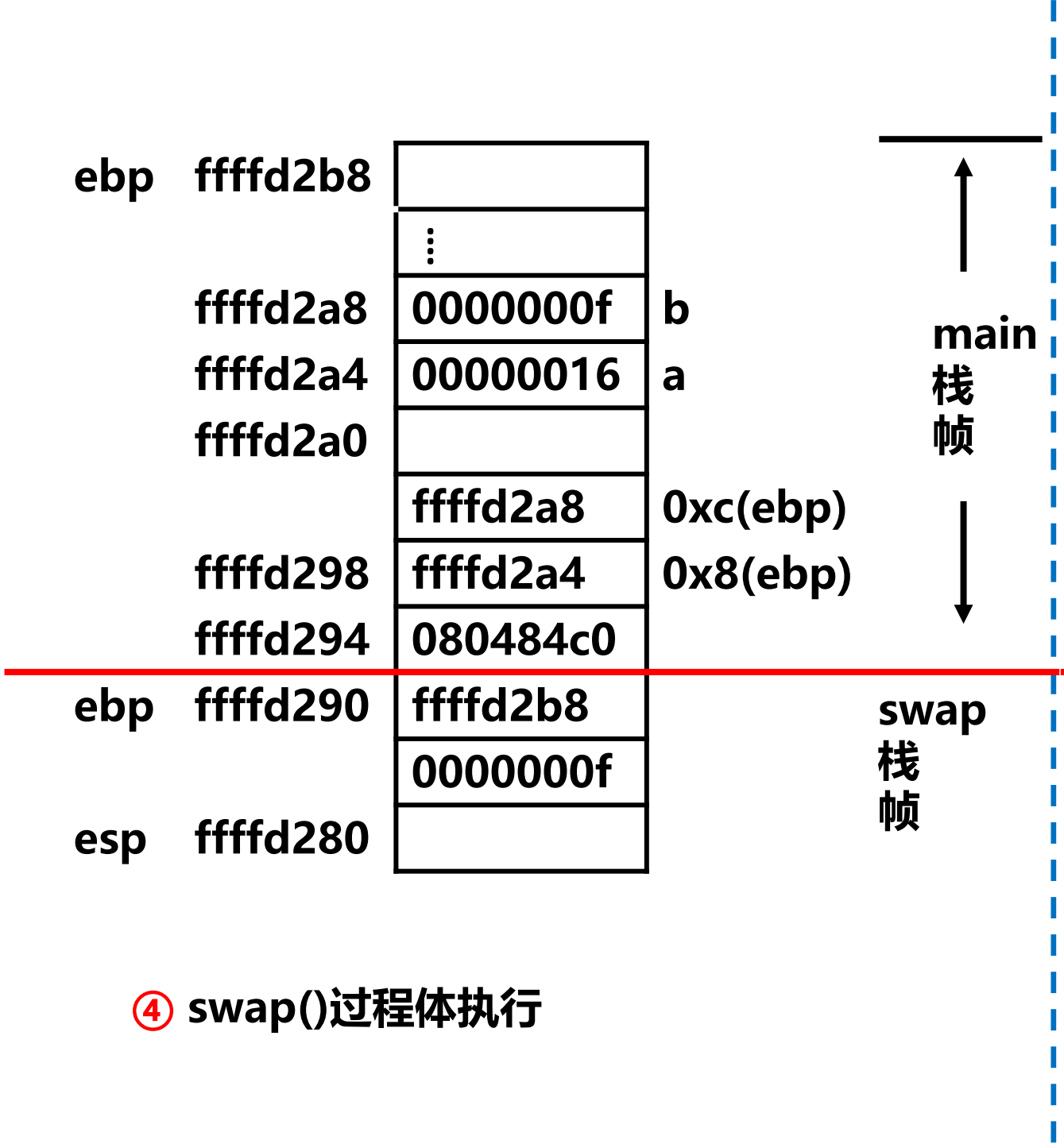
③ swap()的准备工作：分配栈空间



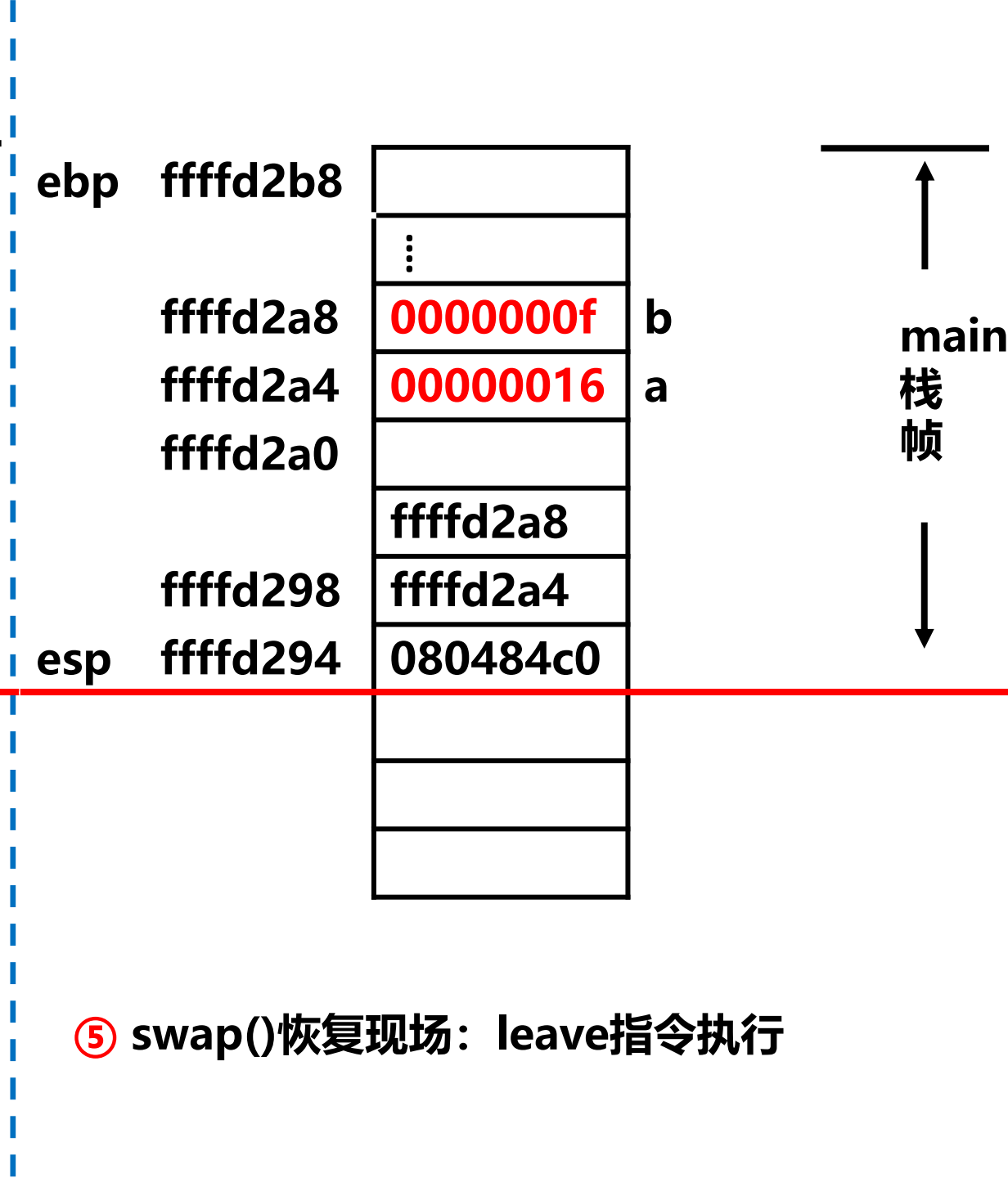
③ swap()的准备工作：分配栈空间



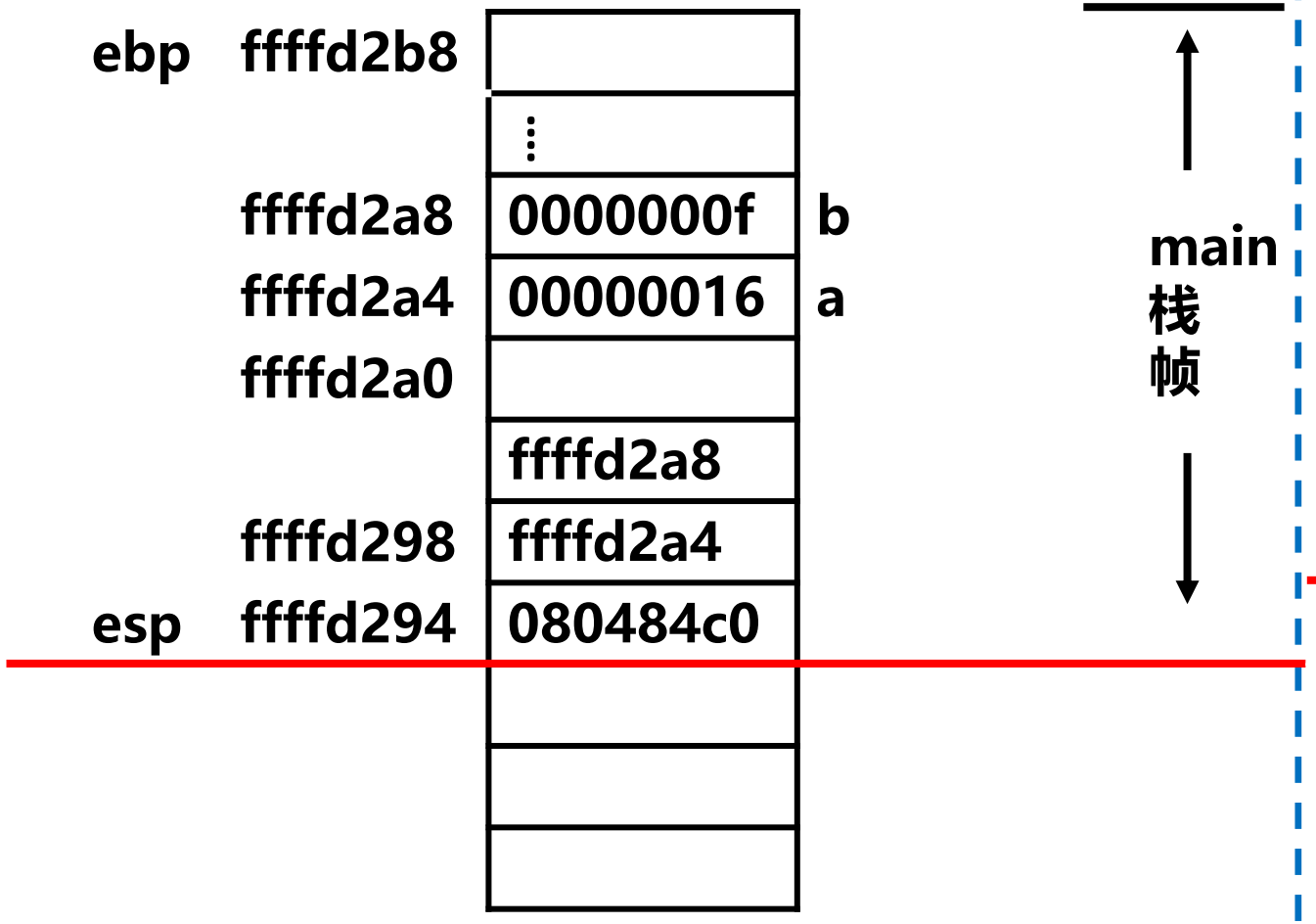
④ swap()执行过程体



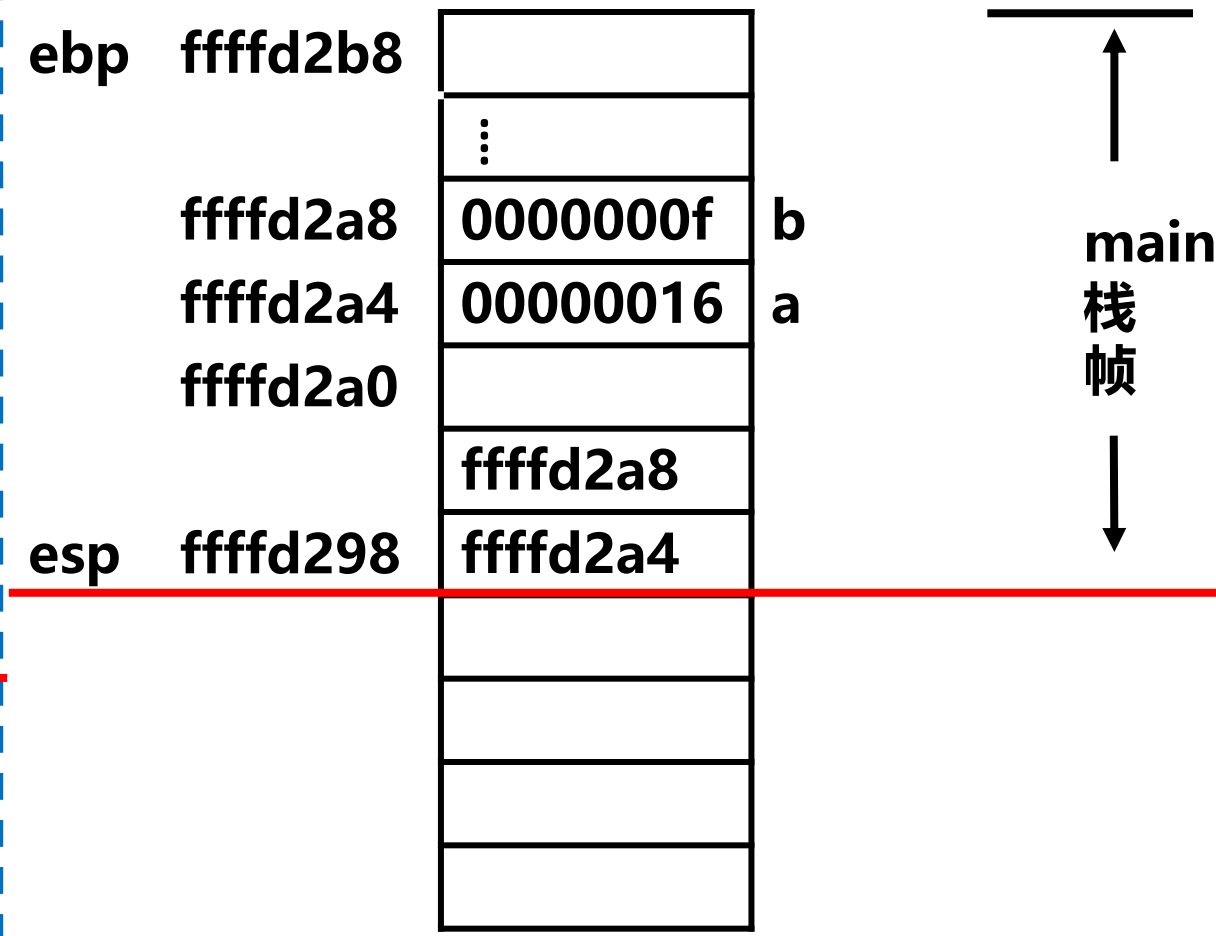
④ swap()过程体执行



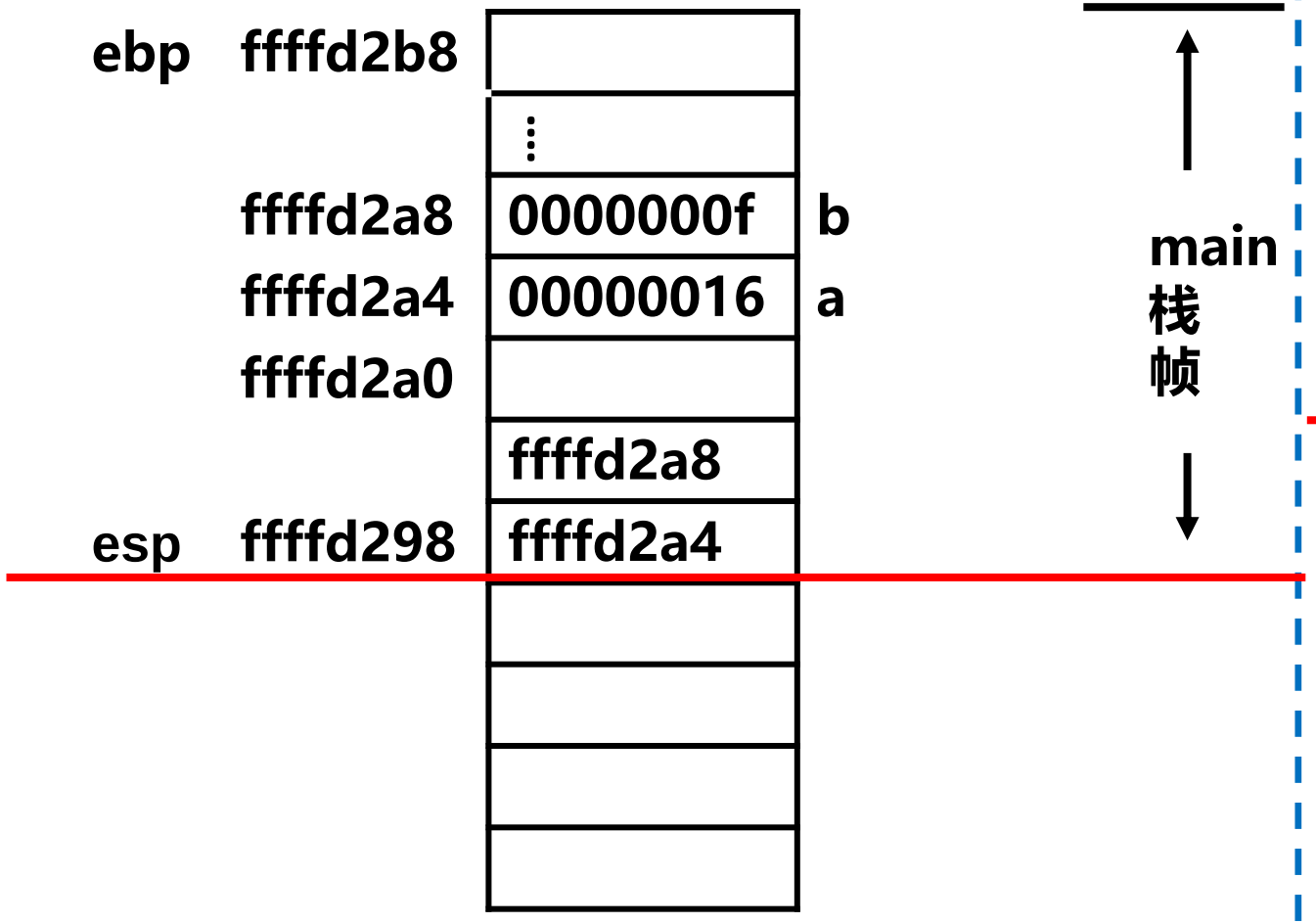
⑤ swap()恢复现场：leave指令执行



⑤ swap()恢复现场：leave指令执行



⑥ swap()返回：ret指令执行



swap()返回：ret指令执行

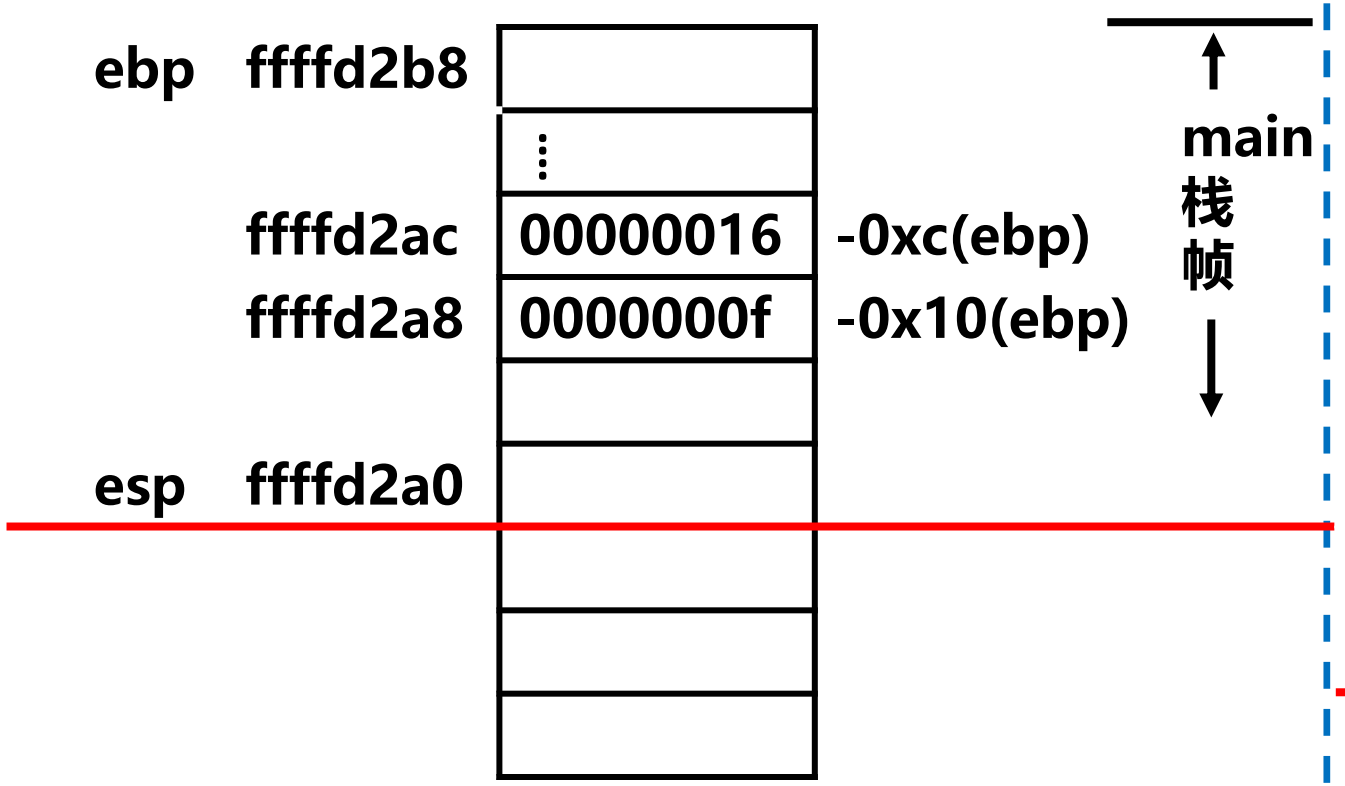
main()：add指令执行，回收部分栈空间

栈和过程调用–按值传递参数示例

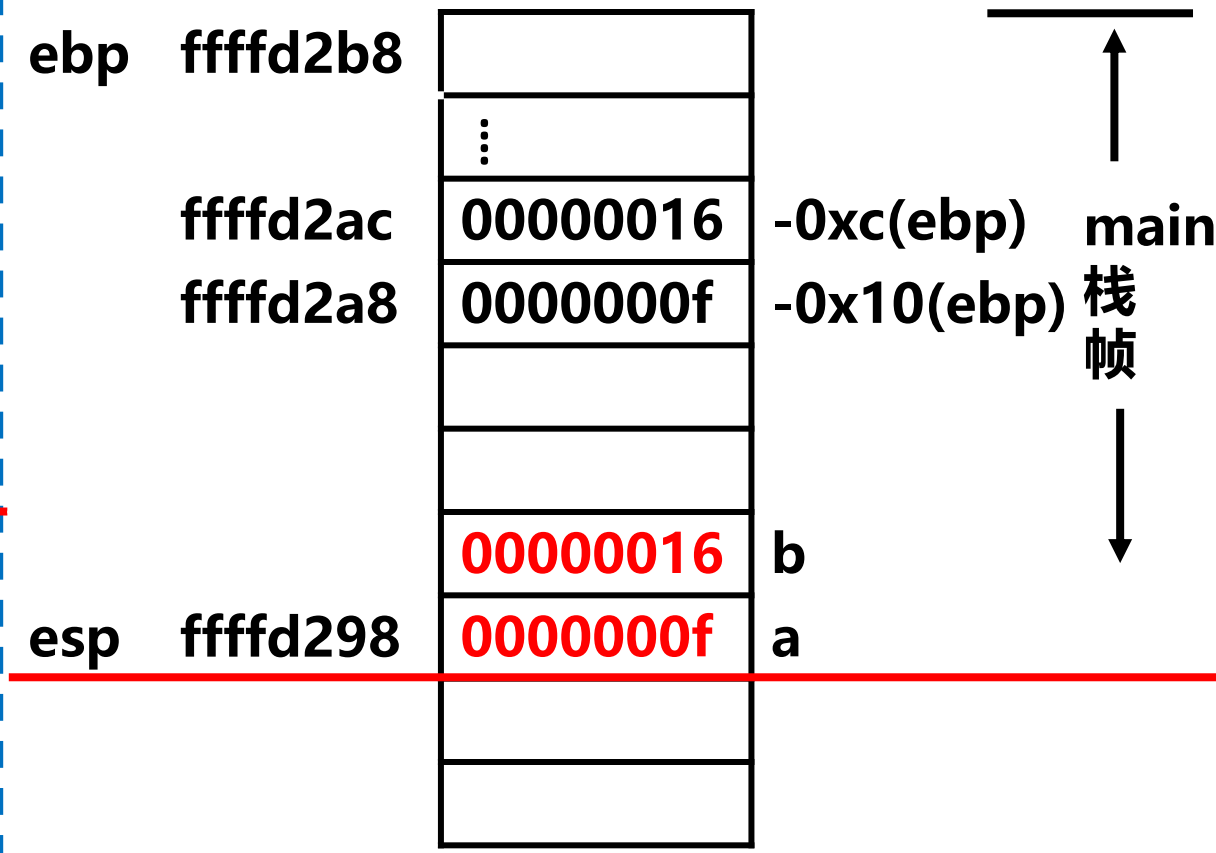
```
#include <stdio.h>
int swap (int x, int y )
{
   int t=x;
       x=y;
       y=t;
}

void main ( )
{  int a=15, b=22;
   swap (a, b);
   printf ("a=%d\tb=%d\n", a, b);
}
```

1. 打开反汇编后的文档，找出过程调用中的相关语句
2. 调试执行程序，画出过程调用中栈帧结构图，理解栈和过程调用
3. 理解参数的按值传递含义

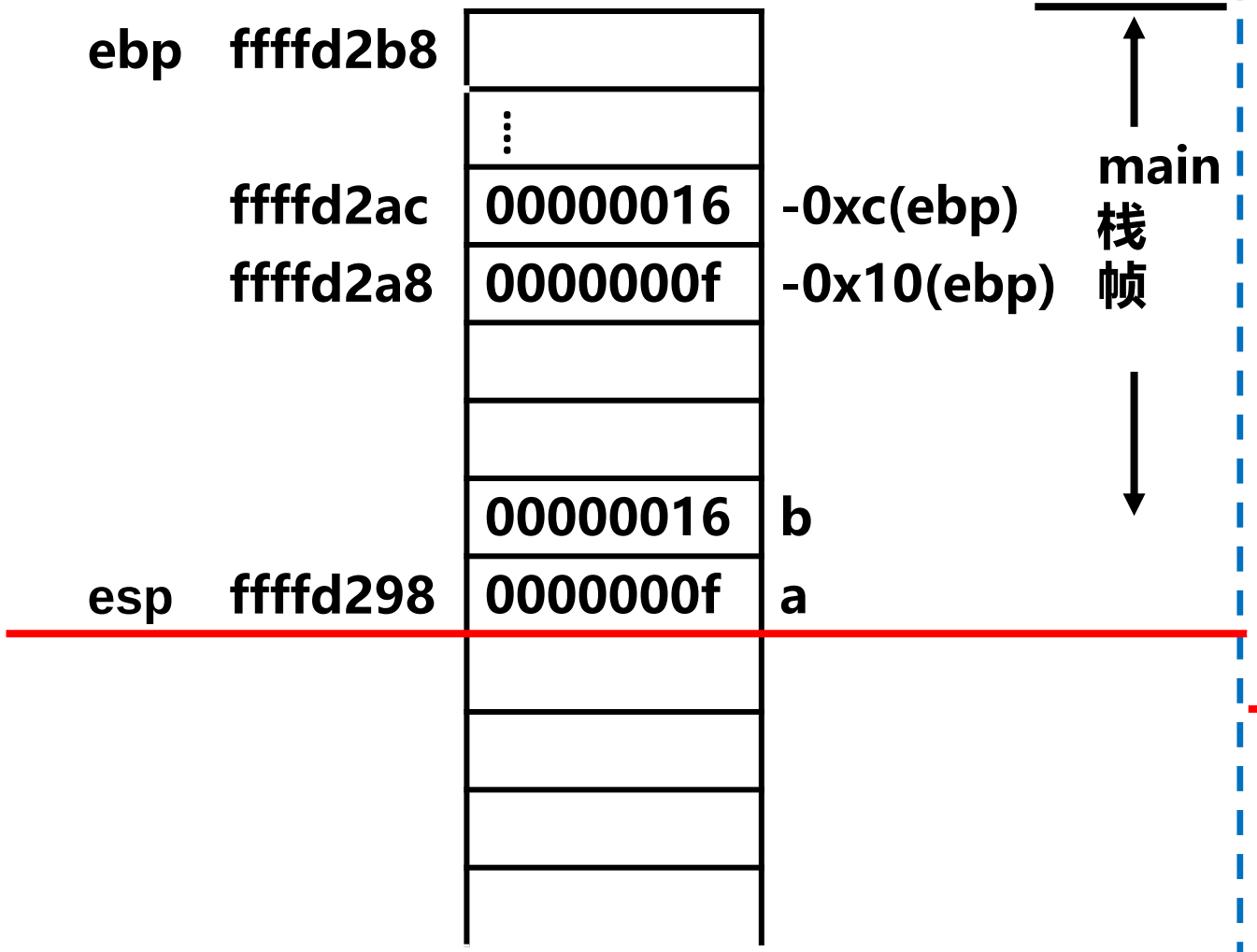


main中“swap（a，b）”执行前

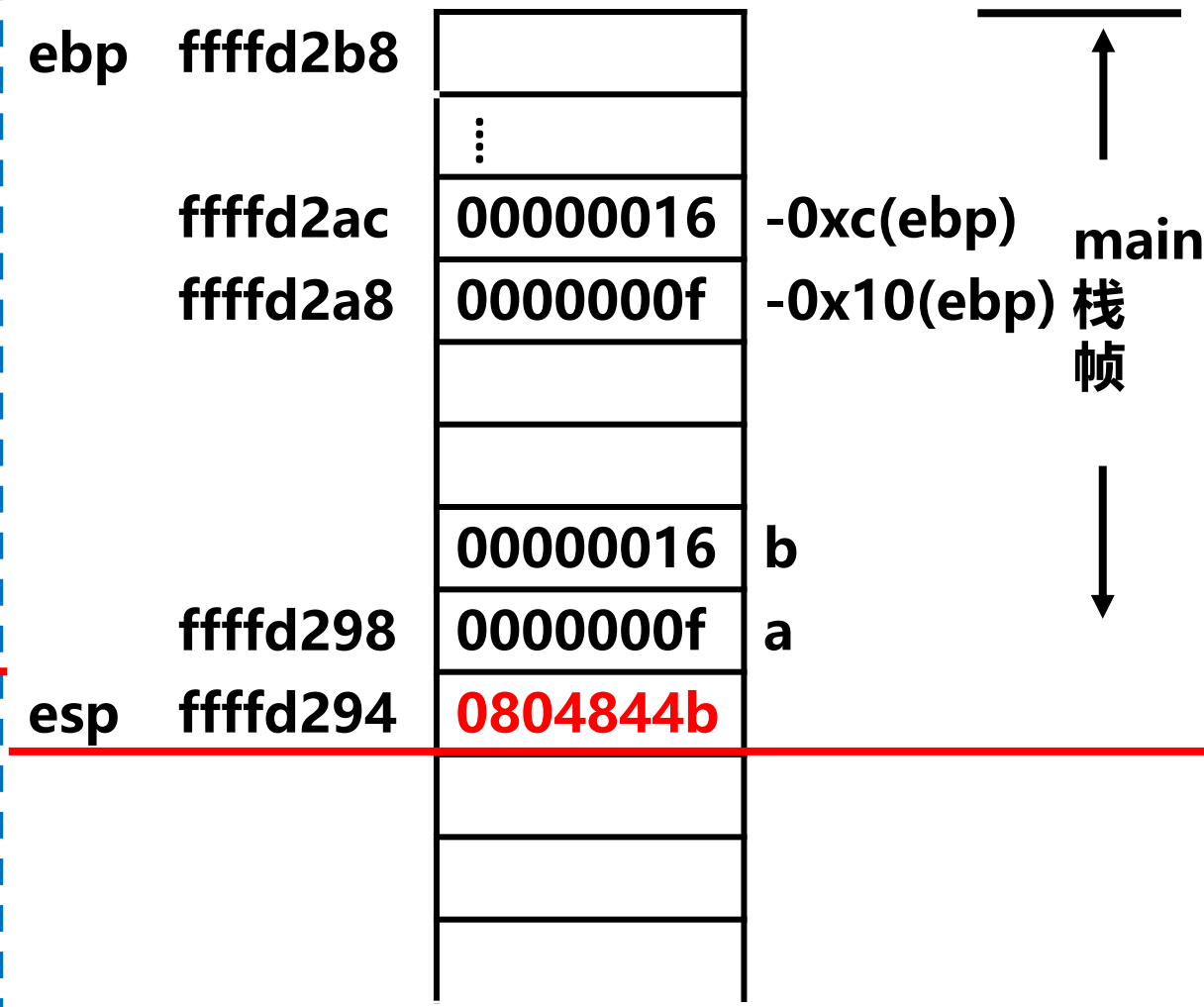


swap（a，b）

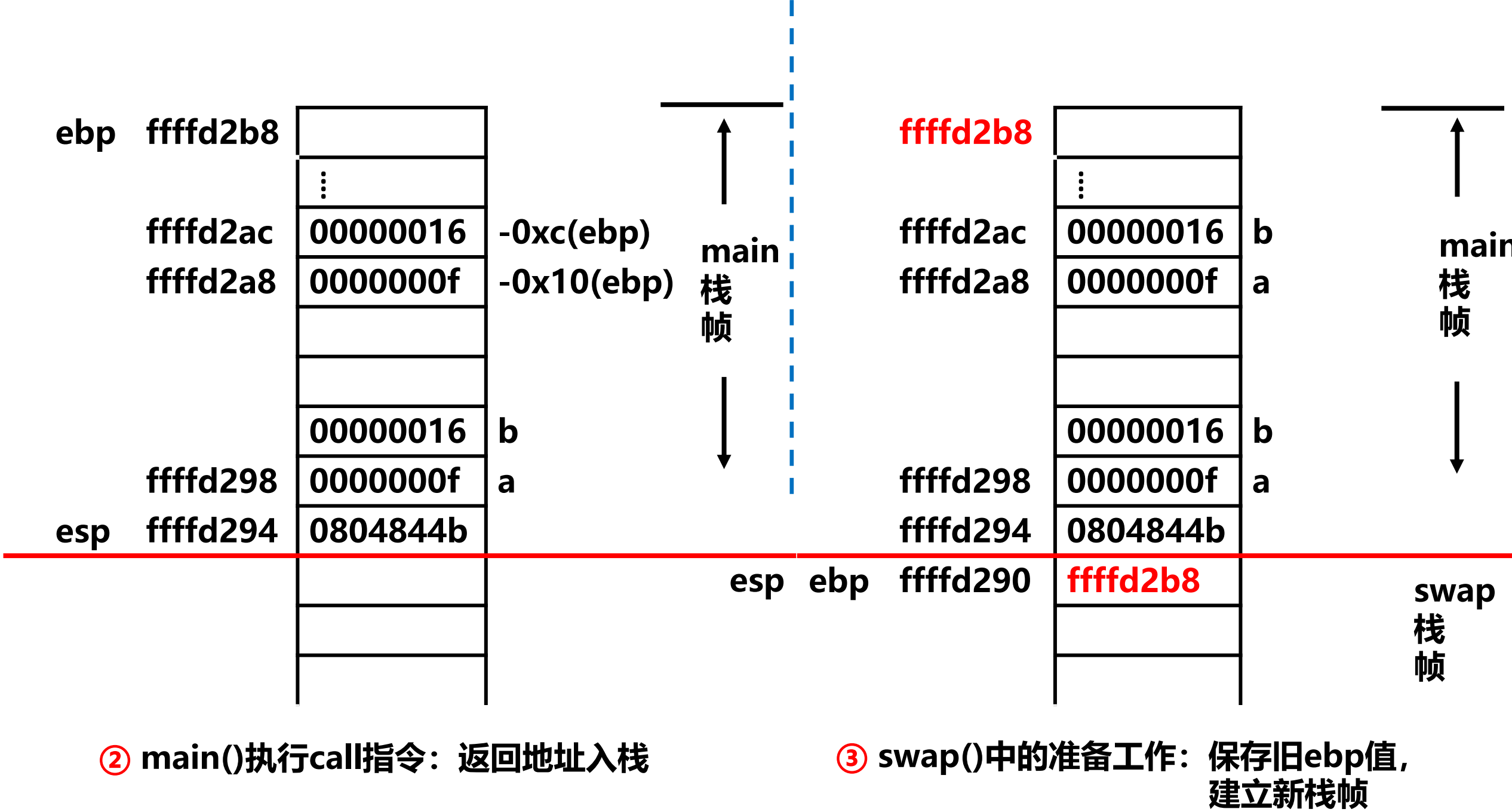
① main()准备工作：参数入栈



① main()准备工作：参数入栈

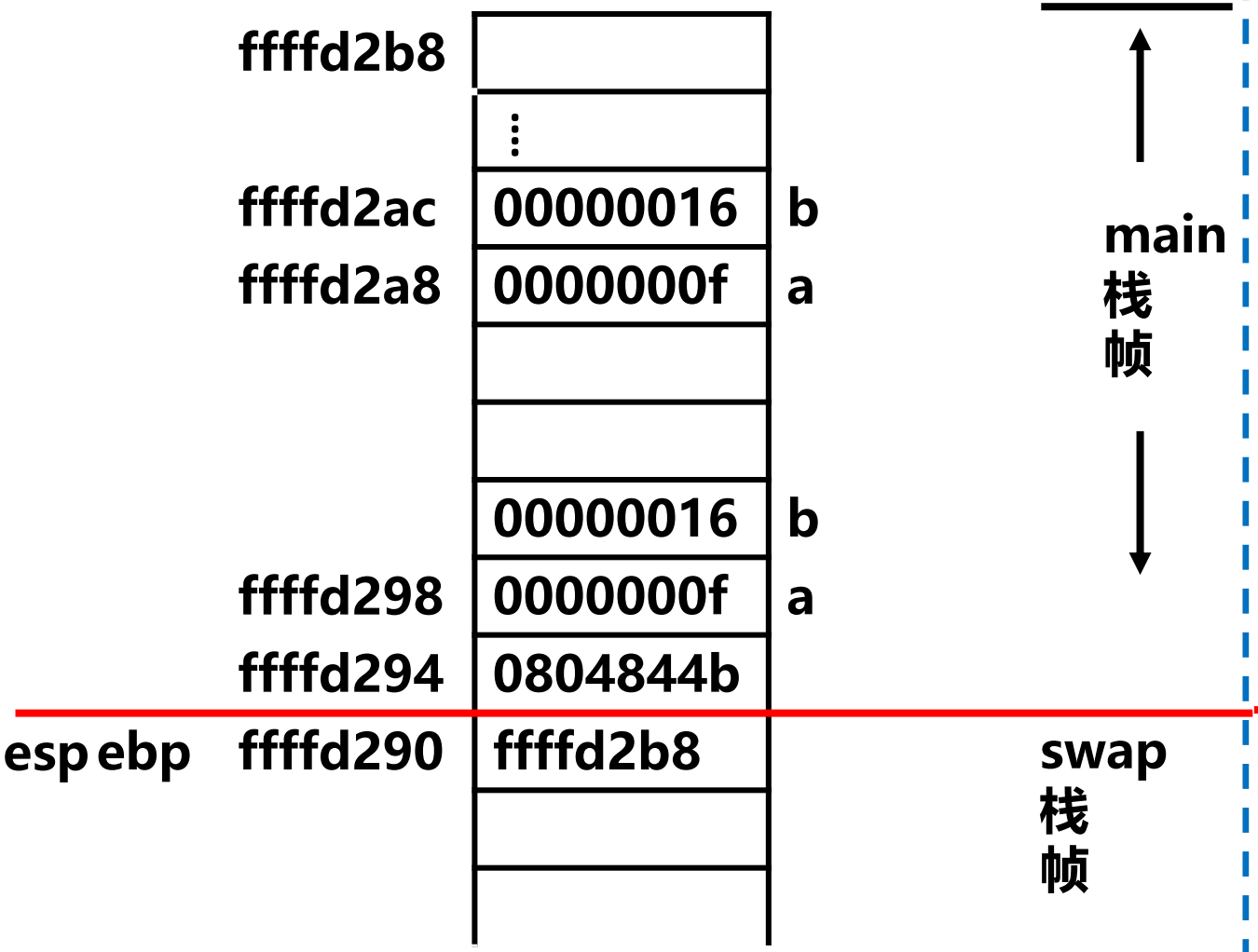


② main()执行call指令：返回地址入栈

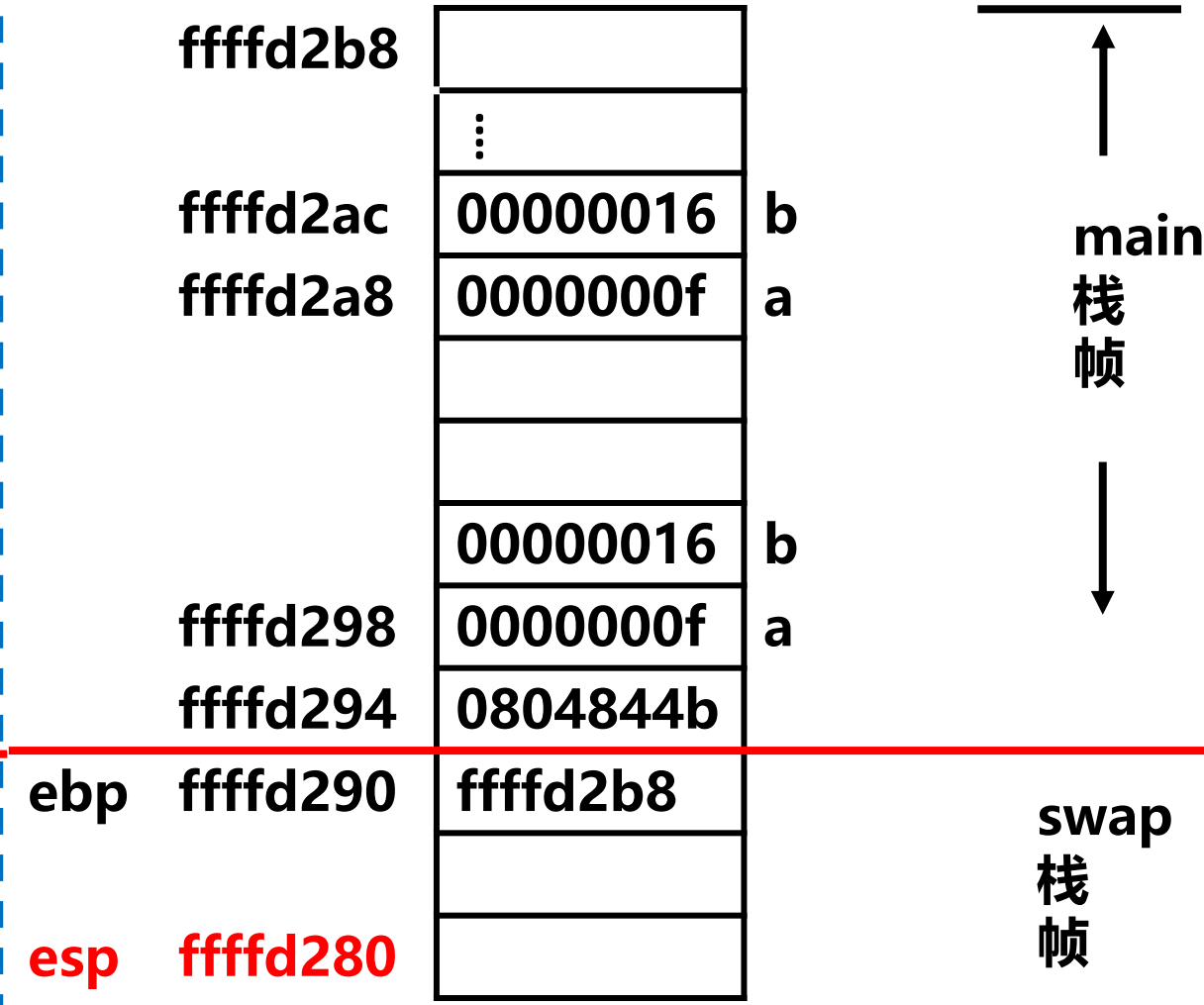


② main()执行call指令：返回地址入栈

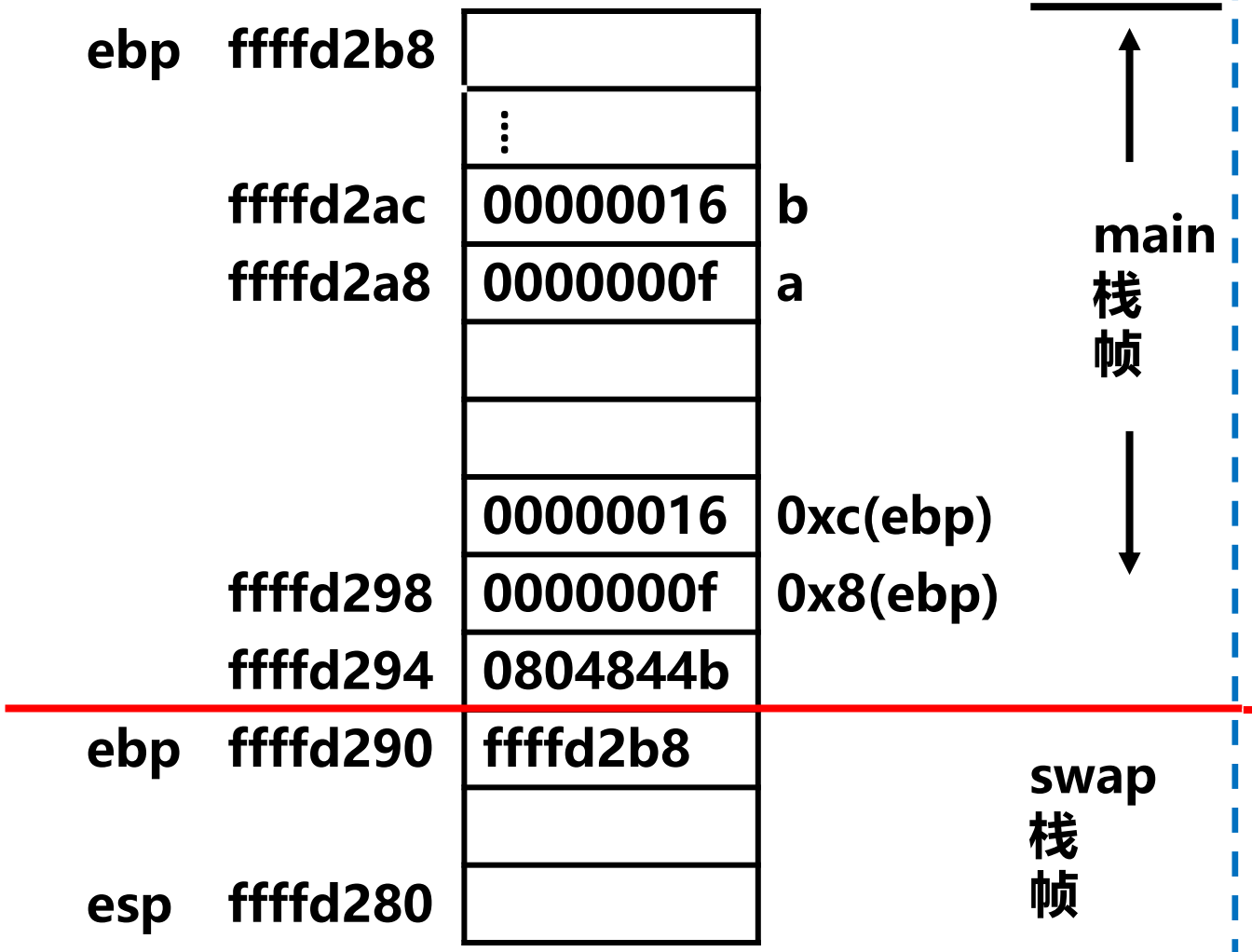
③ swap()中的准备工作：保存旧ebp值，建立新栈帧



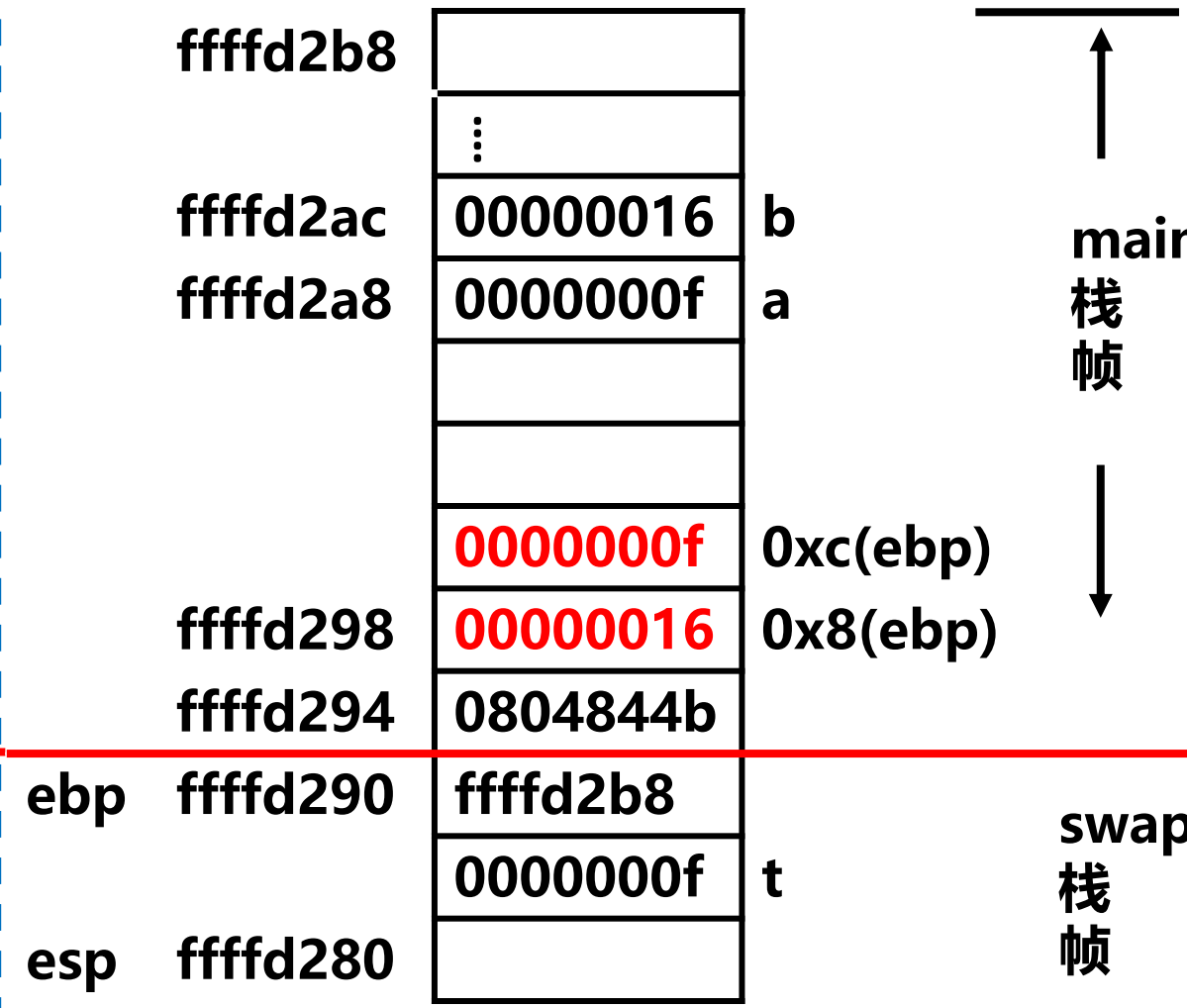
③ swap()的准备工作：保存旧ebp值，建立新栈帧



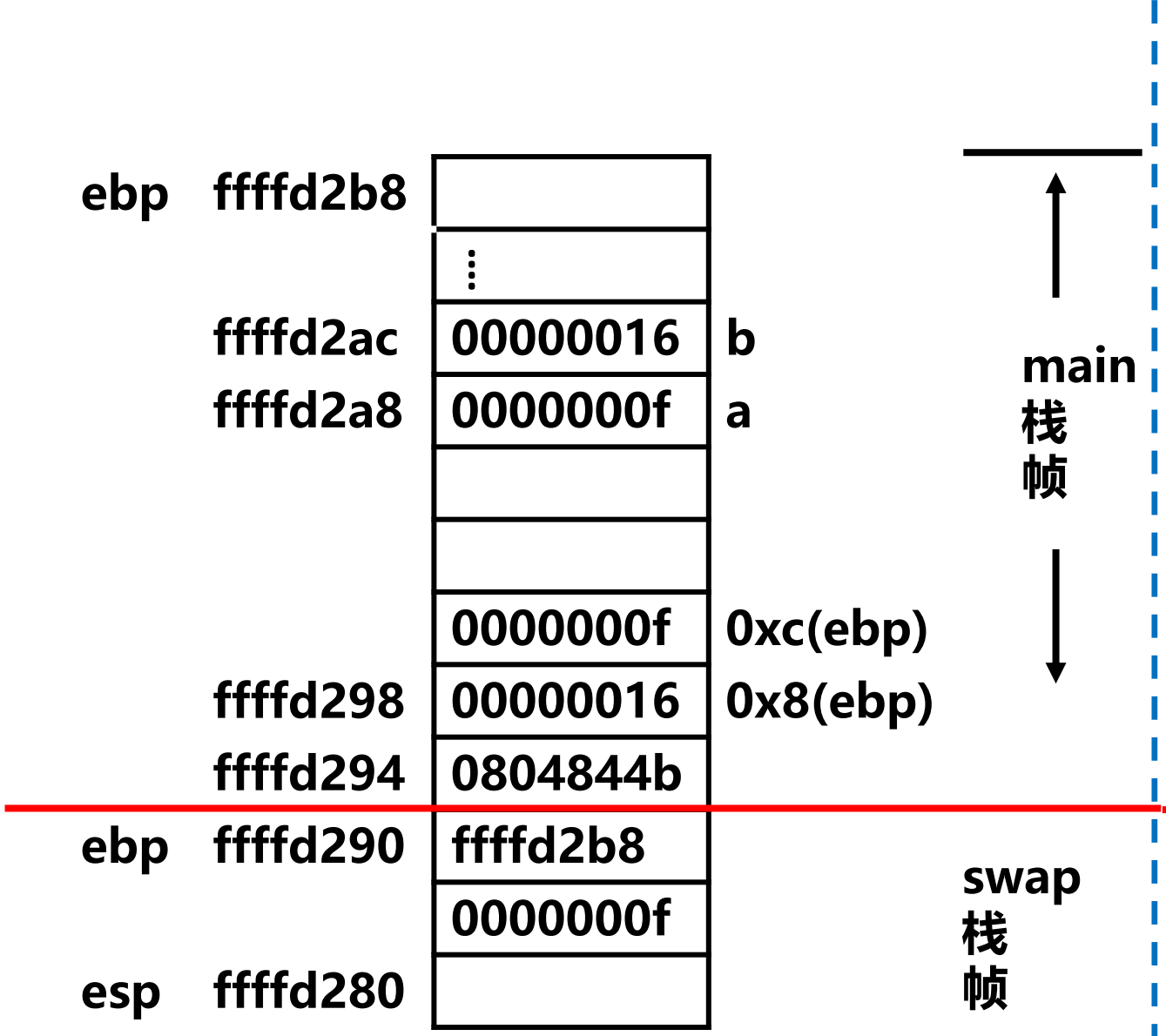
③ swap()的准备工作：分配栈空间



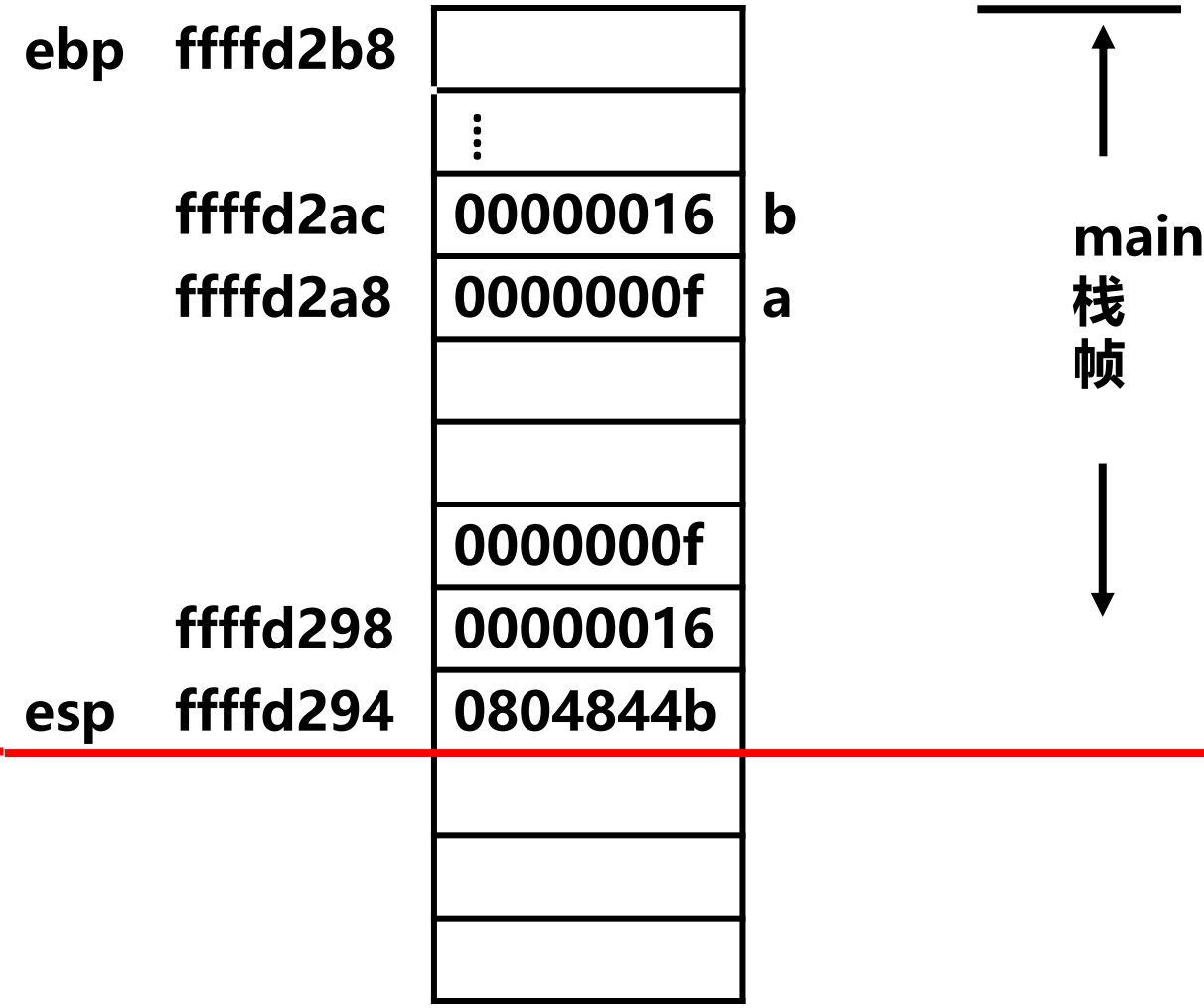
③ swap()的准备工作：分配栈空间



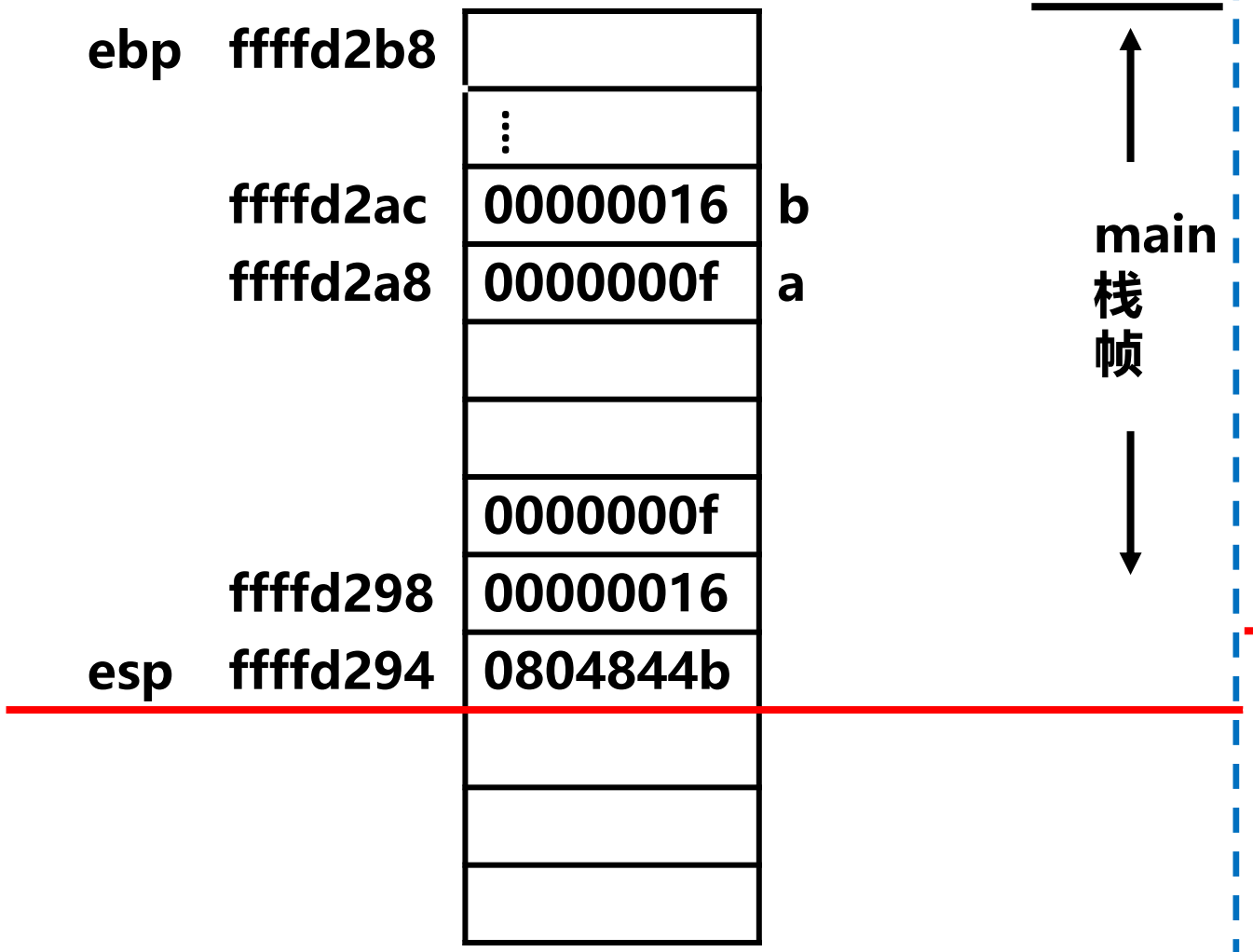
④ swap()执行过程体



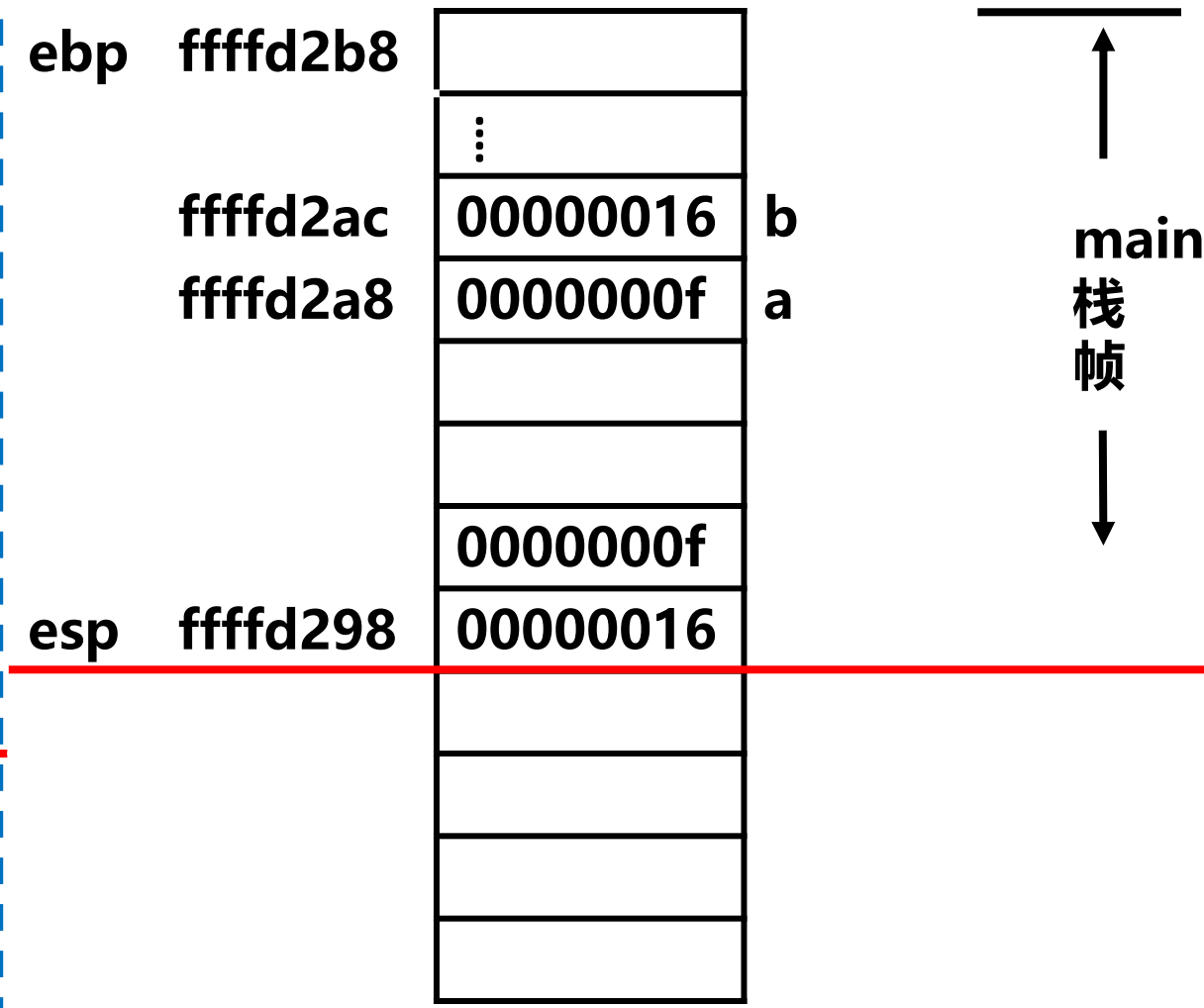
④ swap()过程体执行



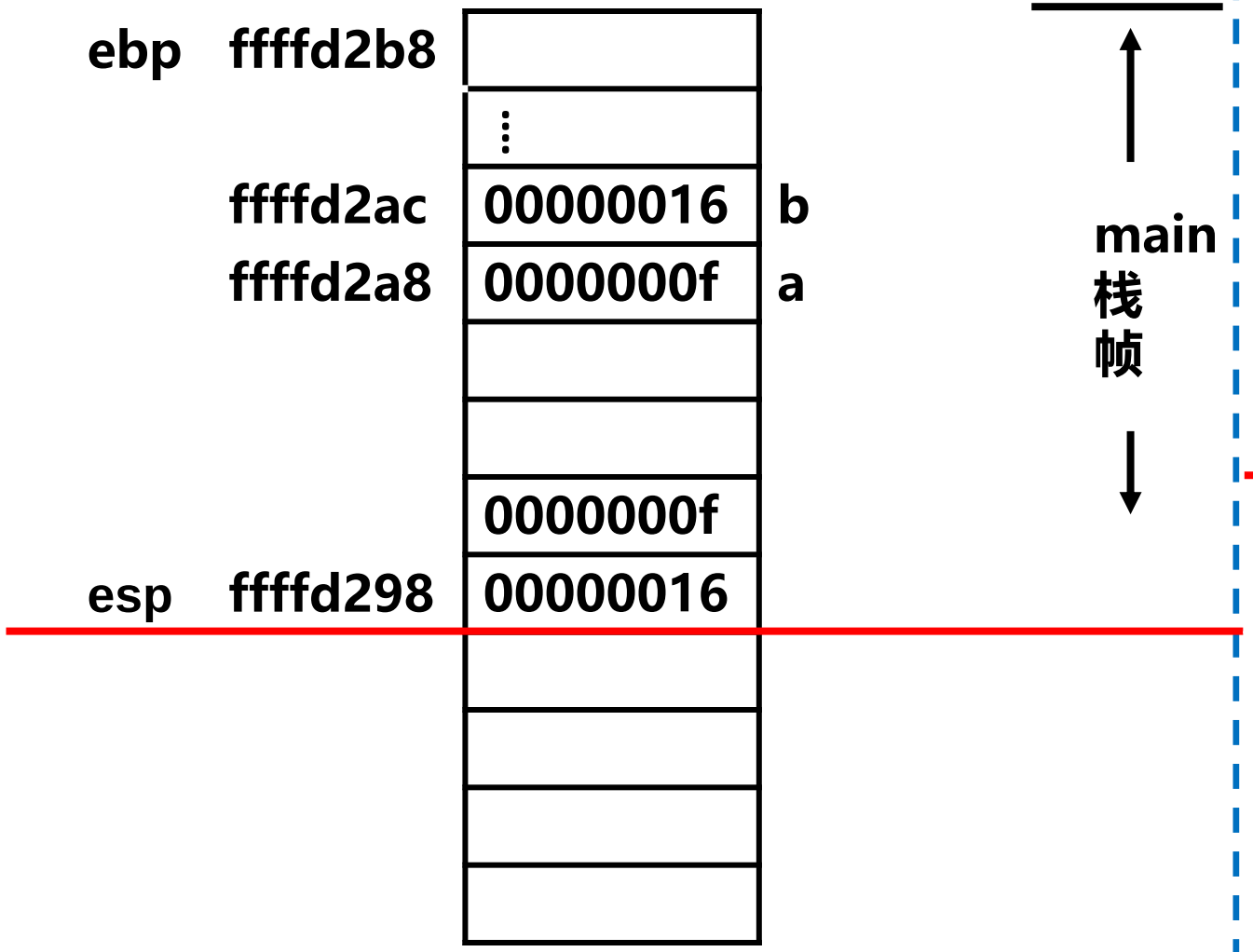
⑤ swap()恢复现场：leave指令执行



⑤ swap()恢复现场：leave指令执行

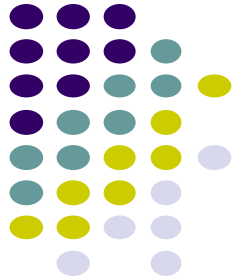


⑥ swap()返回：ret指令执行



swap()返回：ret指令执行

main()：add指令执行，回收部分栈空间



谢谢！