Non-Conventional Enter/Leave Sequence

A Return-to-libc Attack via Buffer Overflow

(Steps 1 - 4 are the same)

RECALL

leave = {

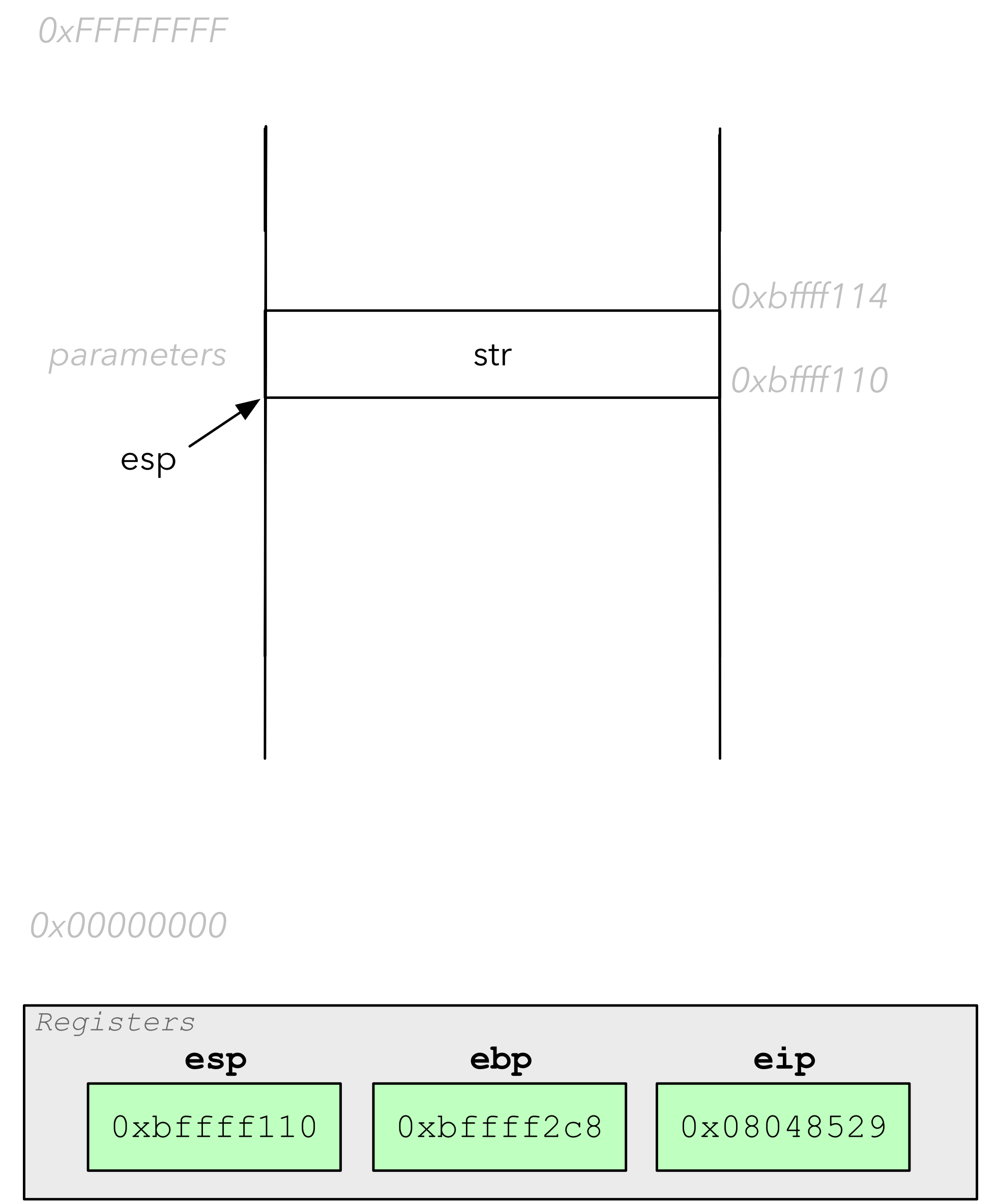
```
Epilogue
movl   %ebp, %esp
popl   %ebp
ret

Prologue
pushl   %ebp
movl    %esp, %ebp
```

SETUP

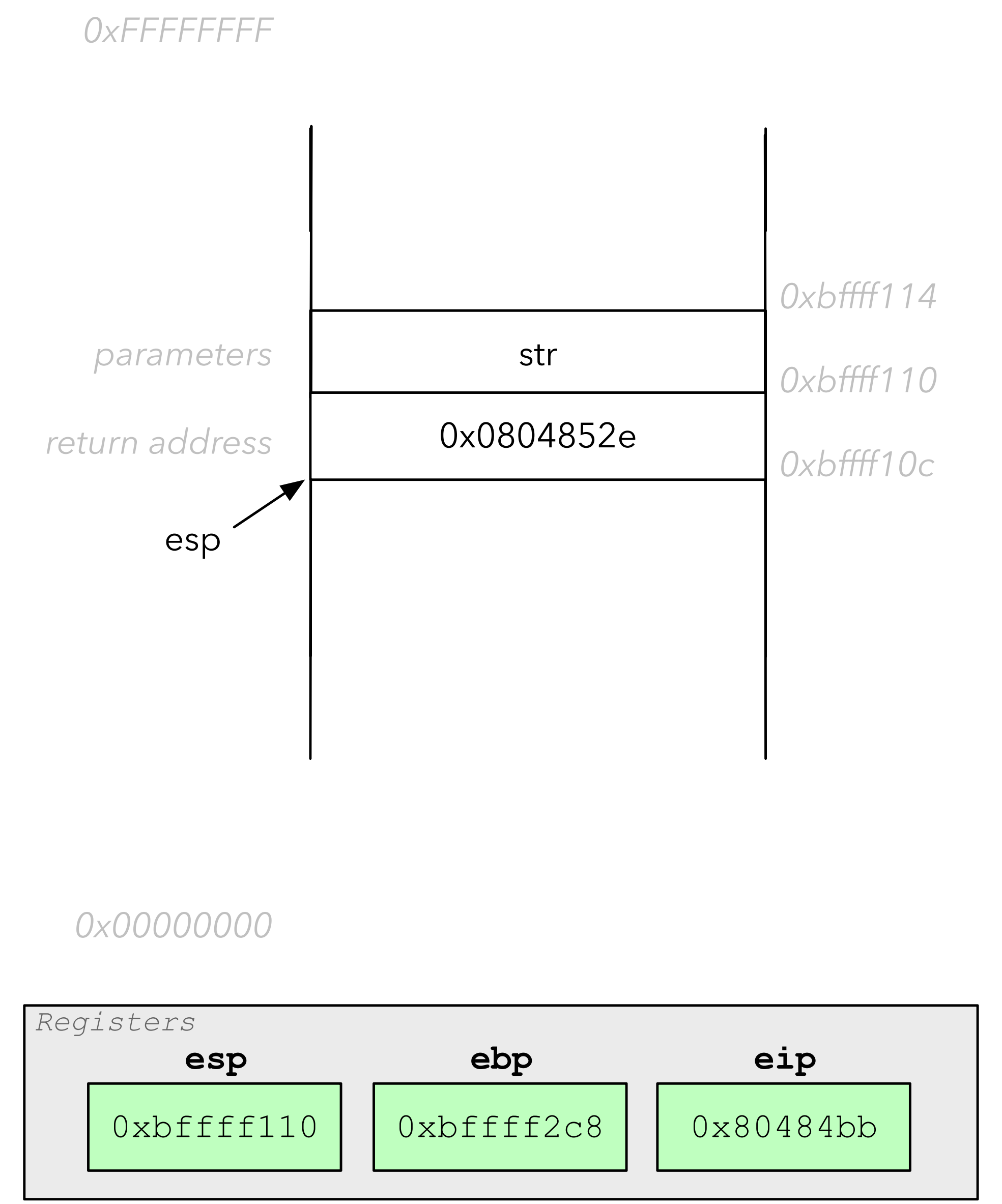
[Step 1]

`lea -0x19c(%ebp), %eax`
`push %eax`



[Step 2]

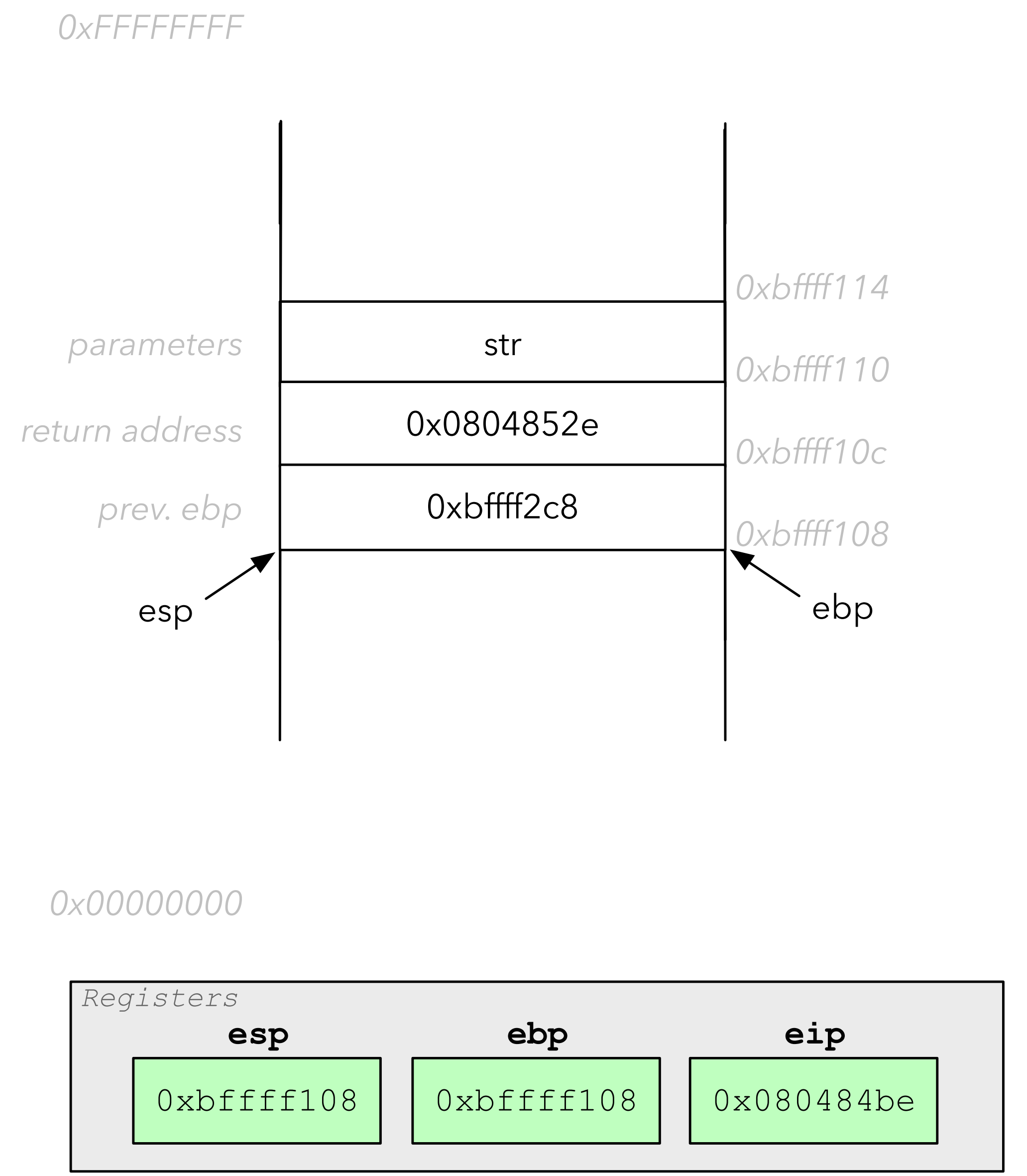
`call foo`



START
PROLOGUE

[Step 3]

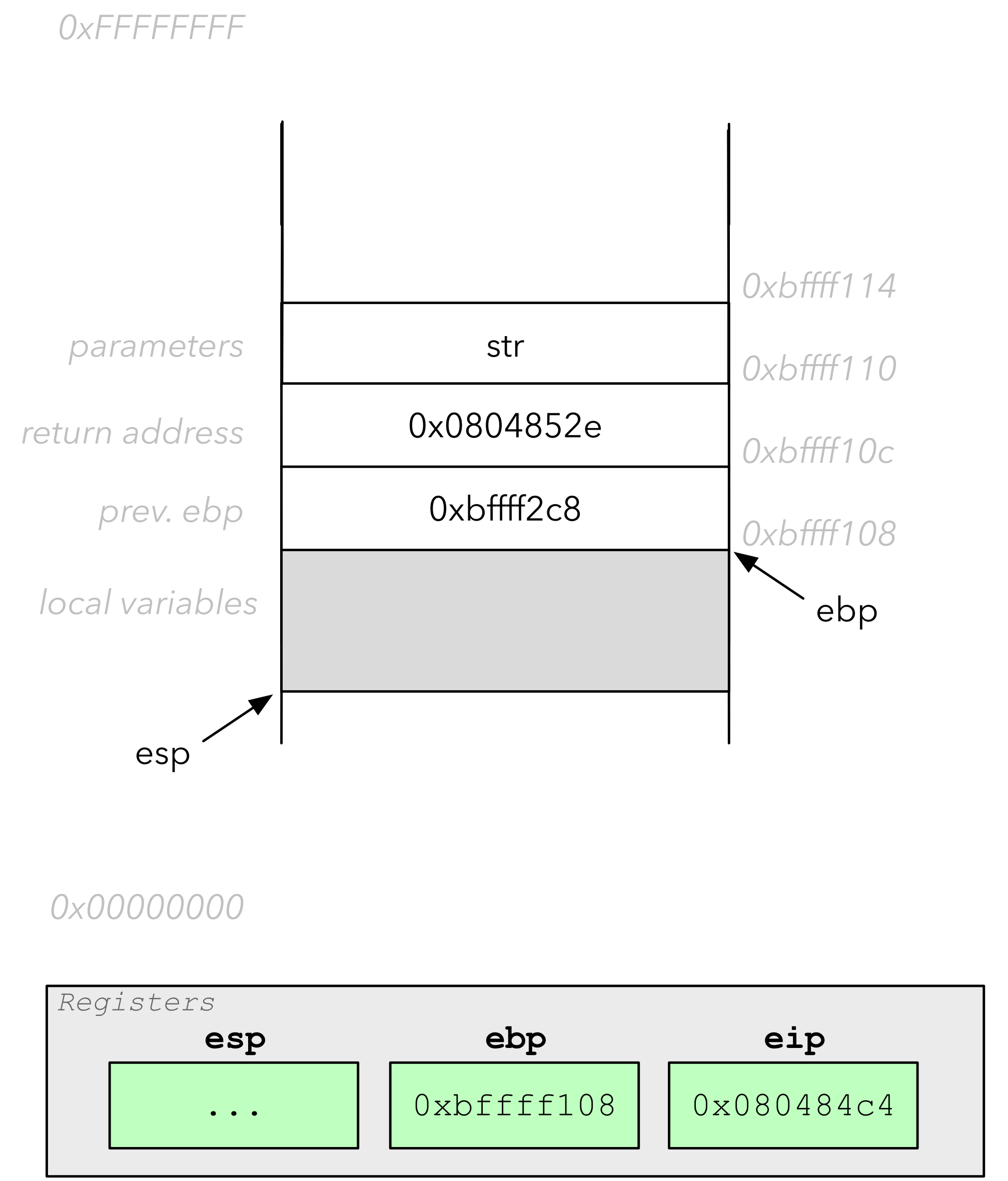
`pushl %ebp`
`movl %esp, %ebp`



[Step 4]

`subl $N, %esp`

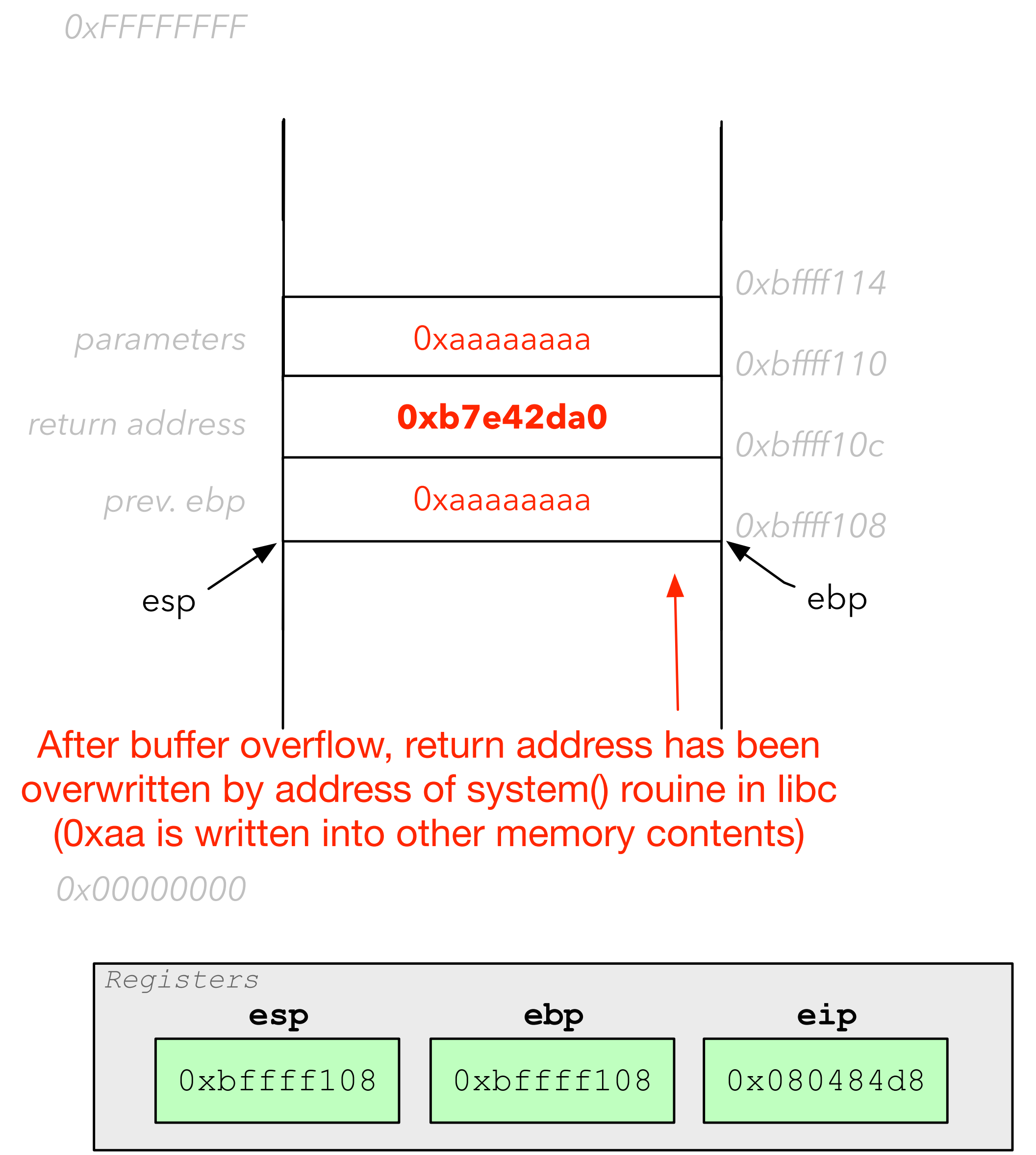
END
PROLOGUE



START
EPILOGUE

[Step 5a]

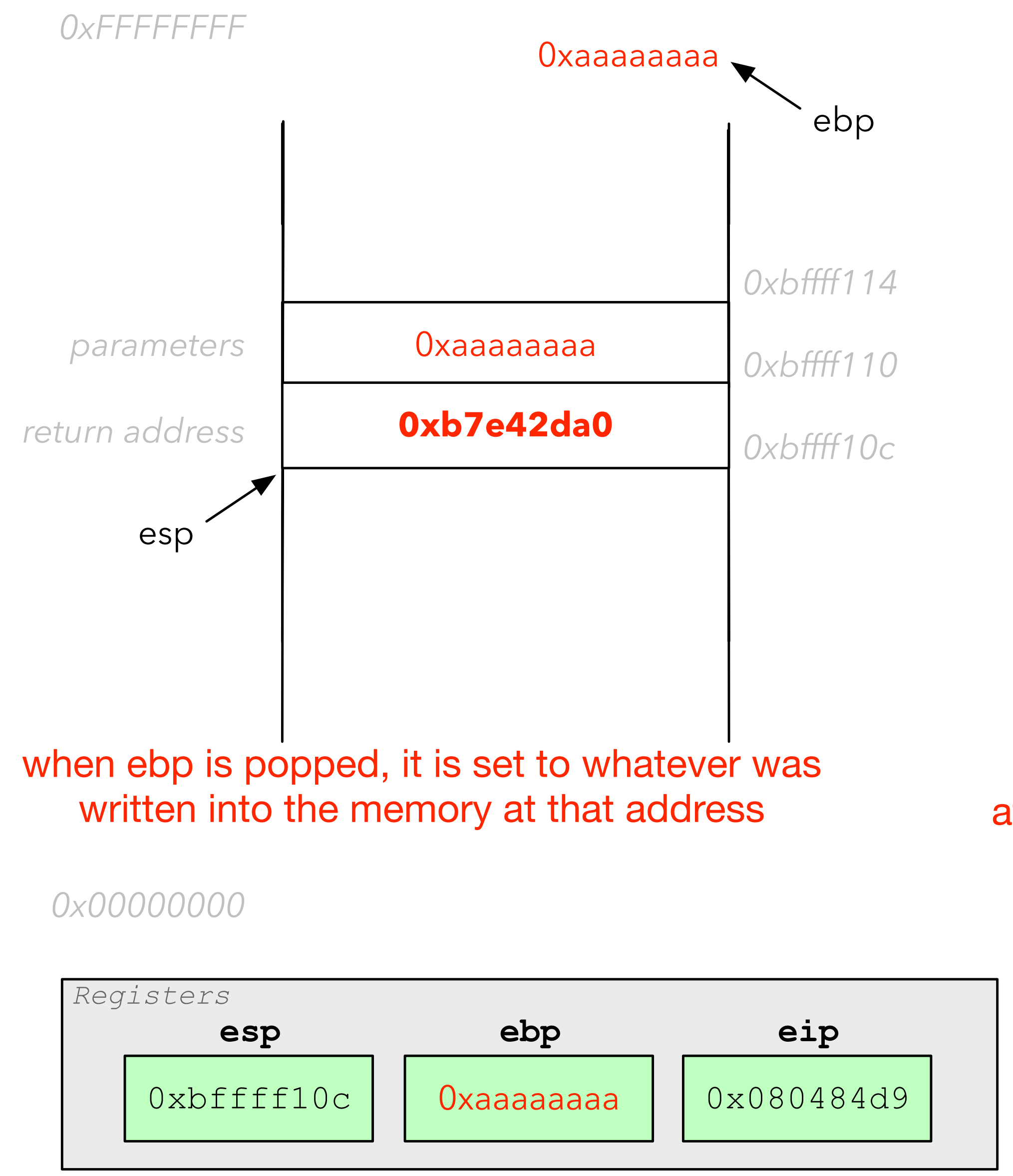
`movl %ebp, %esp`



After buffer overflow, return address has been overwritten by address of system() rouine in libc (0xaa is written into other memory contents)

[Step 5b]

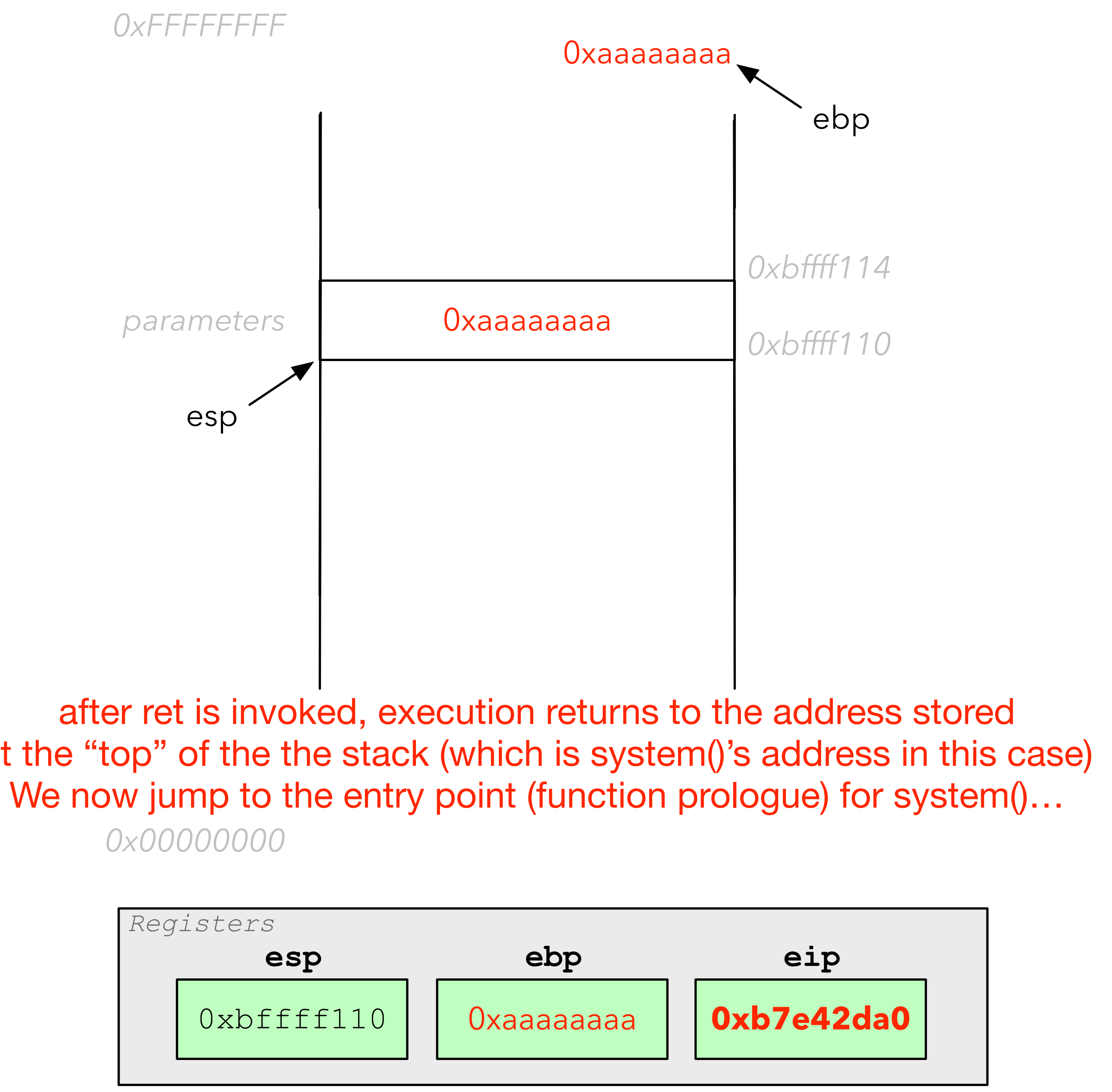
`popl %ebp`



when ebp is popped, it is set to whatever was written into the memory at that address

[Step 6]

`ret`

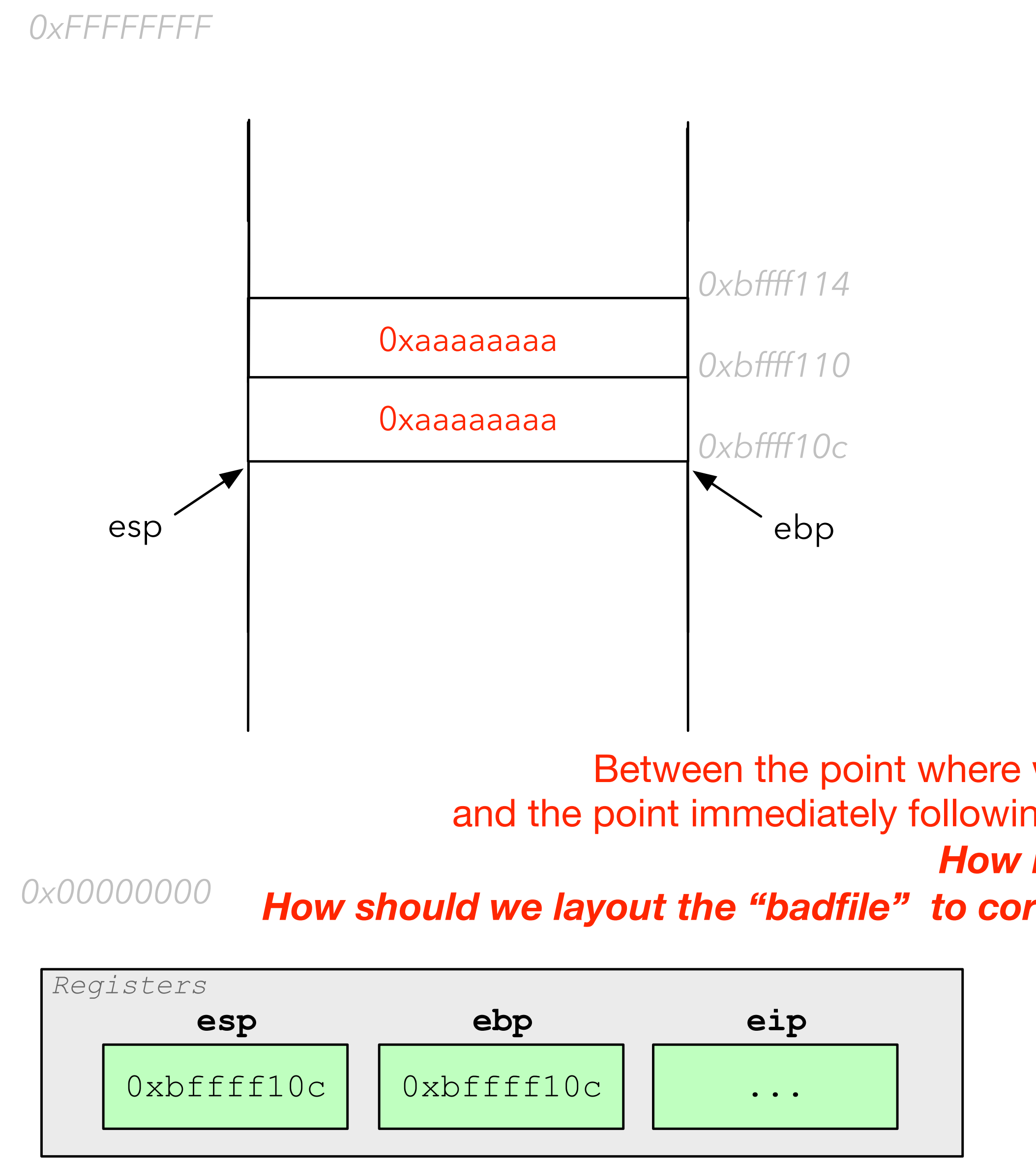


after ret is invoked, execution returns to the address stored at the "top" of the the stack (which is system()'s address in this case) We now jump to the entry point (function prologue) for system()…

START
PROLOGUE

[Step 7]

`pushl %ebp`
`movl %esp, %ebp`



Between the point where we overwrite the buffer in foo() (Step 5a) and the point immediately following the execution o system()'s prologue (step 7)…

How has ebp changed?

How should we layout the "badfile" to correctly place the target addresses on the victim's stack?