ECE 6504

Embedded and Controls Systems Security: Stack Smashing II

Reading:

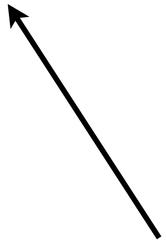
T: *Return to LIBC exploits and Advanced return-into-lib(c) exploits*
N: *The Geometry of Innocent Flesh on the Bone*

“Start every day off with a smile and get it over with.”

–W.C. Fields

preventing code injection: $W \oplus X$

preventing code injection: $W \oplus X$



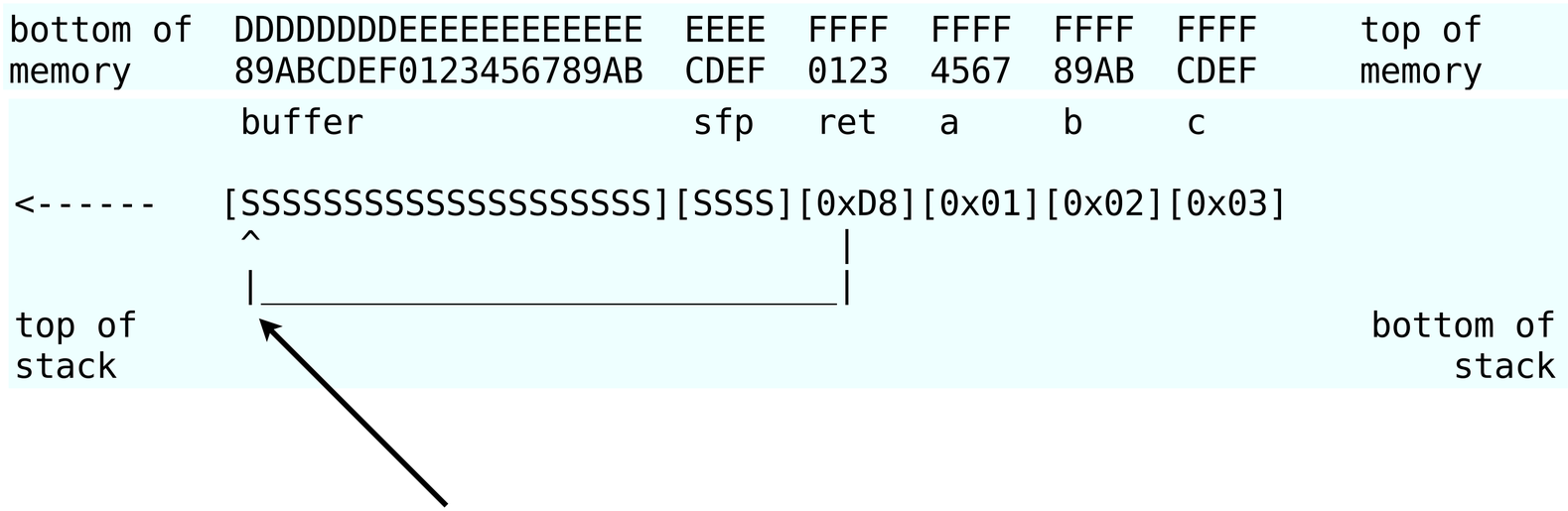
memory marked either writable or executable, not both

```
bottom of                                                          top of
memory                                                             memory
          buffer2       buffer1   sfp   ret   a     b     c
<------   [           ][        ][   ][   ][   ][   ][   ]

top of                                                          bottom of
stack          W     W   W  W                                      stack
```

$W \oplus X$

memory after buffer overflow:



jump causes seg. fault

(no execute data)

circumventing $W \oplus X$

attacker:

return-into-libc:

you don’t like my code, I’ll use yours

circumventing $W \oplus X$

attacker:

return-into-libc:

you don’t like my code, I’ll use yours

return-into-libc

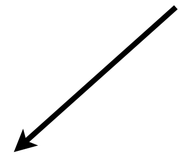
libc:

1. linked by (almost) all C programs
2. interesting functions: file and console-based I/O

return-into-libc

libc:

1. linked by (almost) all C programs
2. interesting functions: file and console-based I/O

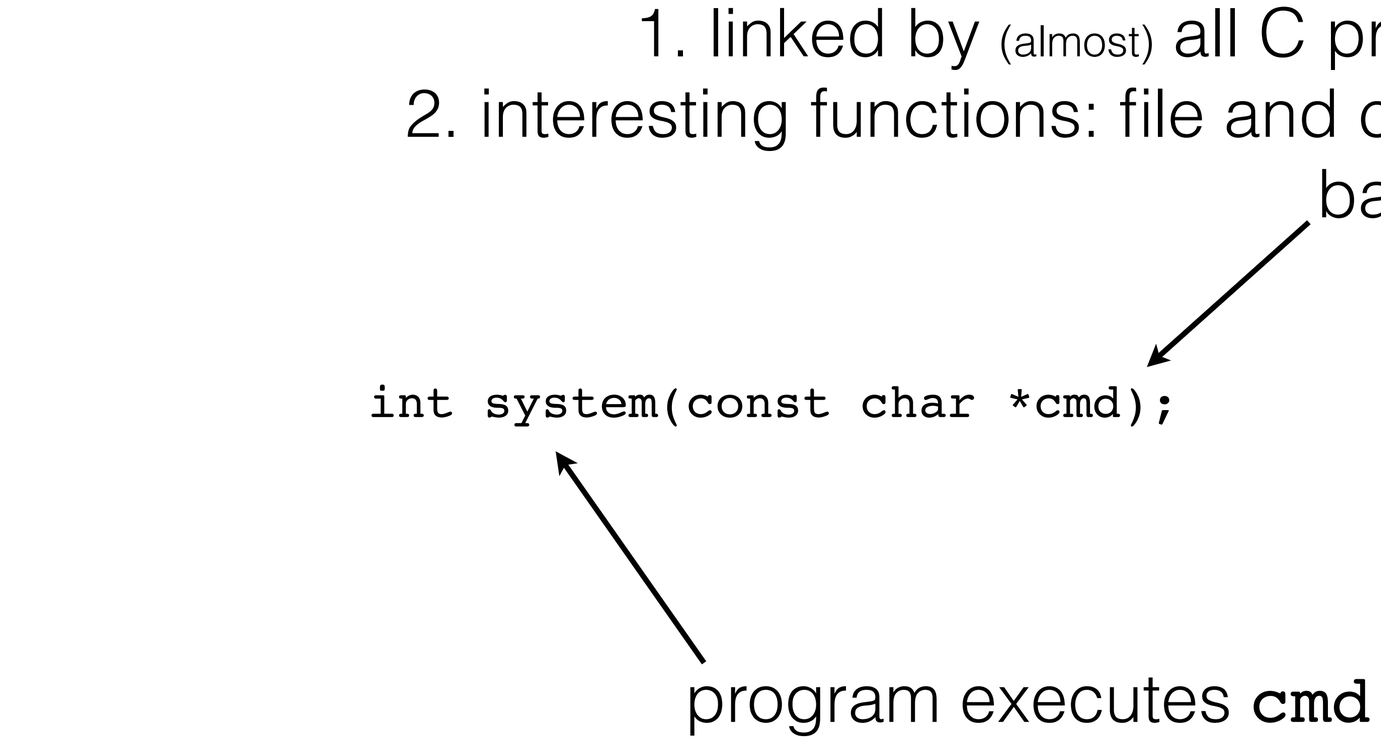


```
int system(const char *cmd);
```

return-into-libc

libc:

1. linked by (almost) all C programs
2. interesting functions: file and console-based I/O



attack:

make function
return to libc function

attack:

make function
return to libc function

Q: is this possible?

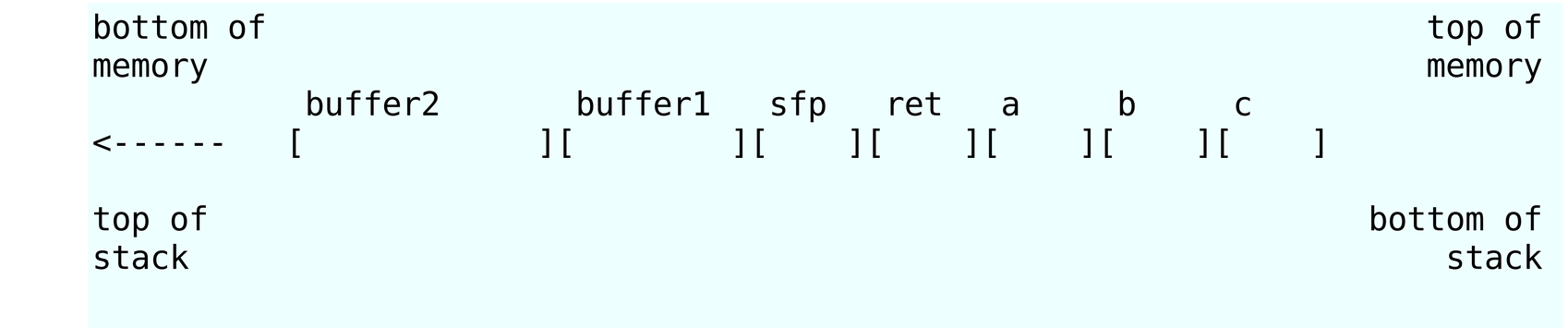
A: remember state w/r/t function call

stack after function called

```
void function(int a, int b, int c) {
   char buffer1[5];
   char buffer2[10];
}

void main() {
  function(1,2,3);
}
```

```
pushl %ebp
movl %esp,%ebp
subl $20,%esp
```

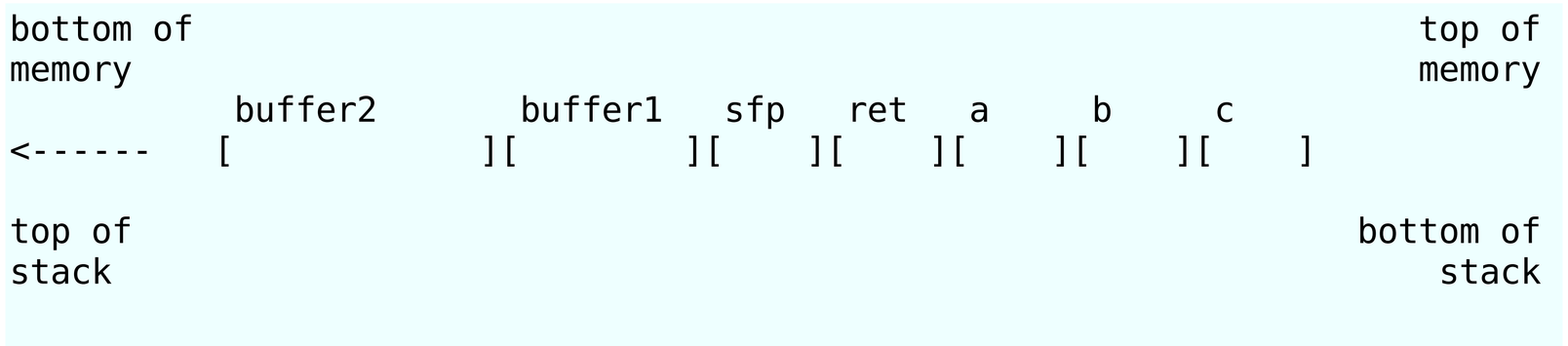


stack after function called

```
void function(int a, int b, int c) {
   char buffer1[5];
   char buffer2[10];
}

void main() {
  function(1,2,3);
}
```

```
pushl %ebp
movl %esp,%ebp
subl $20,%esp
```



to return-to-libc:

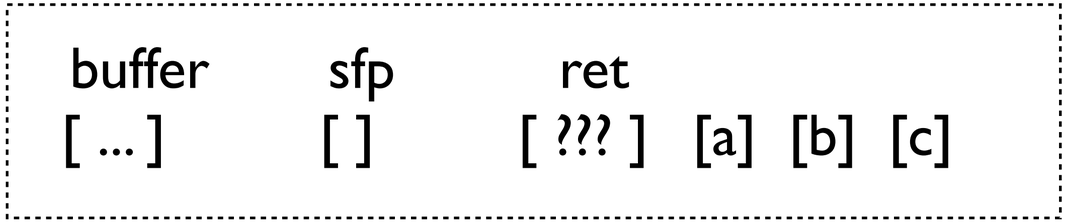
1. ret => addr. of libc function
2. a,b,c => arguments of libc function

return-into-libc: stack after buffer overflow

memory addr grow ⟶

stack after call:

```
[function(int a, int b, int c)]
```

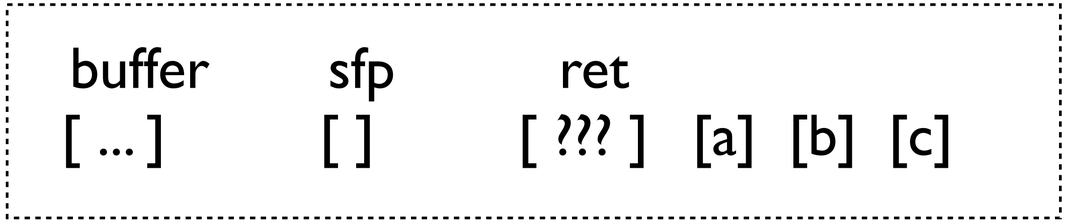


return-into-libc: stack after buffer overflow

memory addr grow ⟶

stack after call:

```
[function(int a, int b, int c)]
```



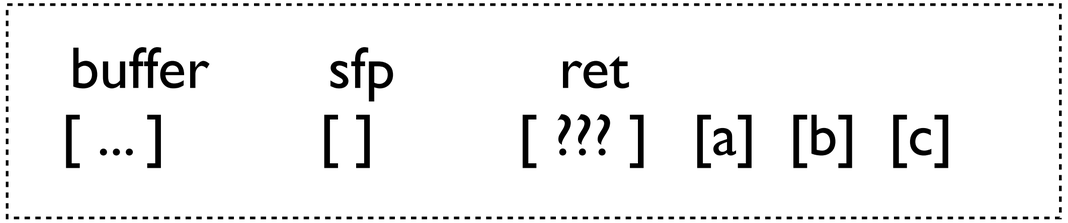
after overflowing buffer:

return-into-libc: stack after buffer overflow

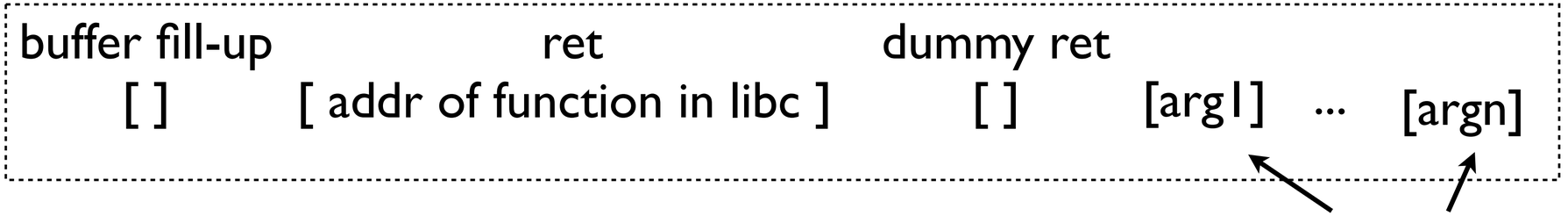
memory addr grow ⟶

stack after call:

```
[function(int a, int b, int c)]
```



after overflowing buffer:

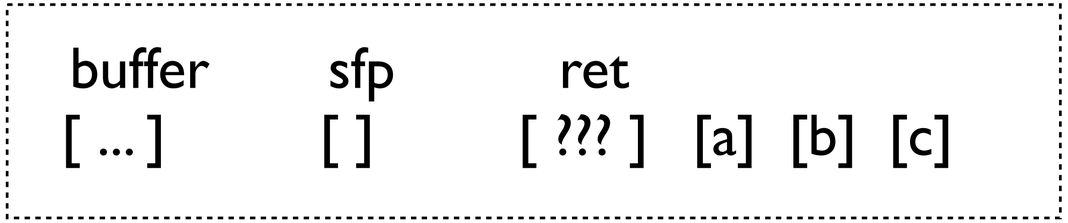


return-into-libc: stack after buffer overflow

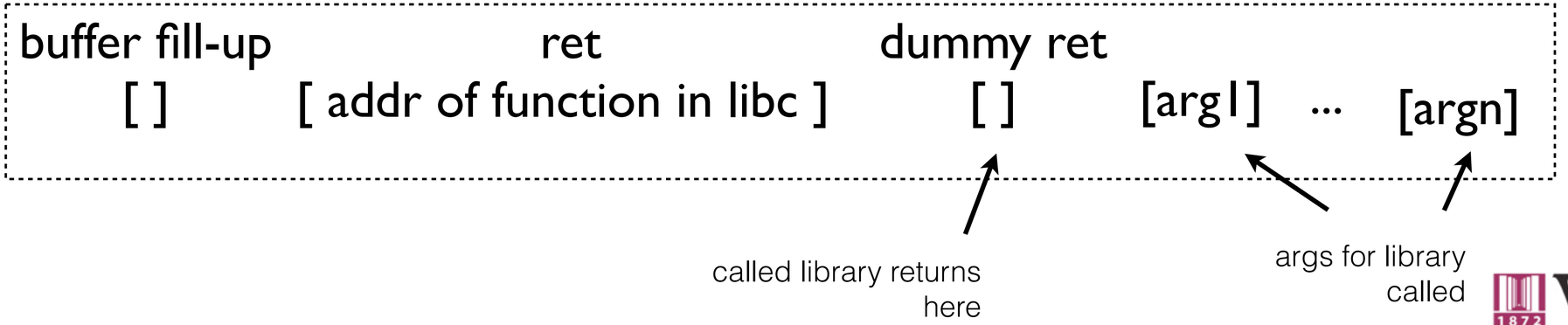
memory addr grow ⟶

stack after call:

```
[function(int a, int b, int c)]
```



after overflowing buffer:

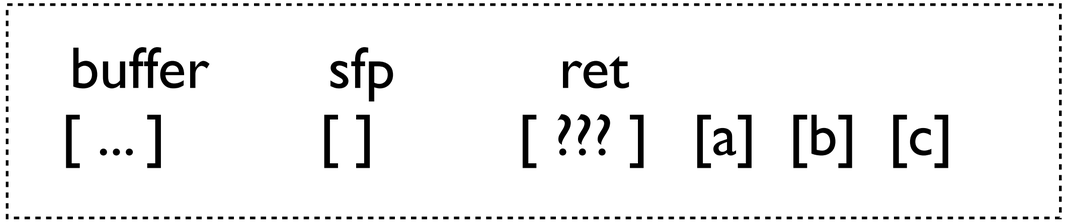


return-into-libc: stack after buffer overflow

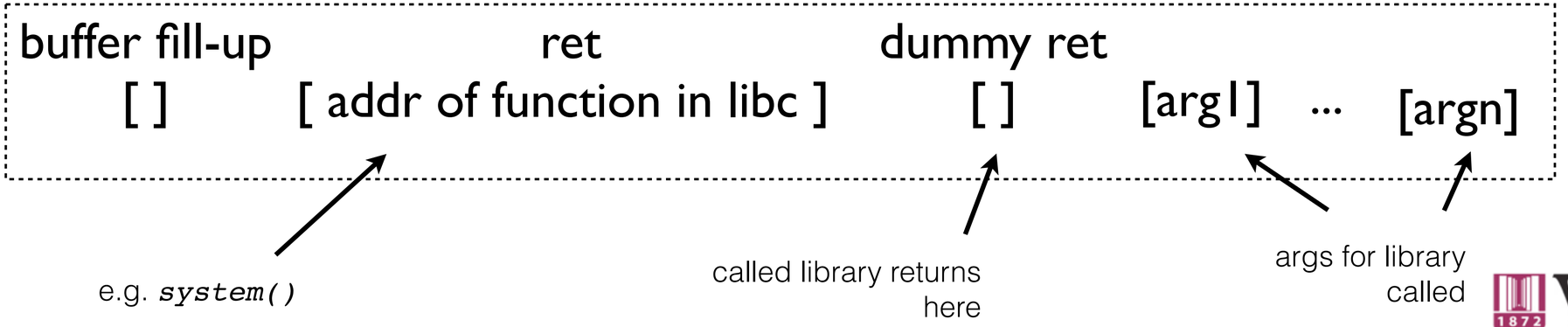
memory addr grow ⟶

stack after call:

```
[function(int a, int b, int c)]
```



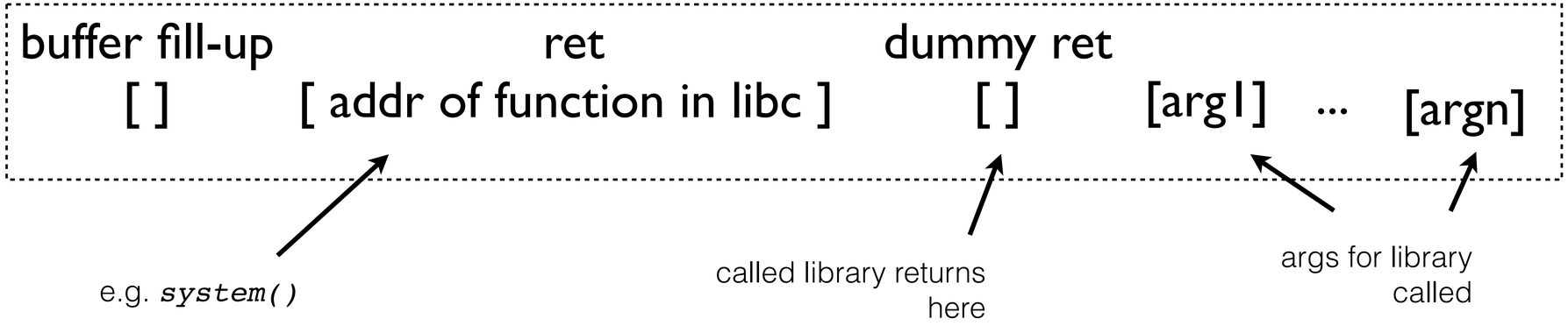
after overflowing buffer:



return-into-libc: stack after buffer overflow

memory addr grow ⟶

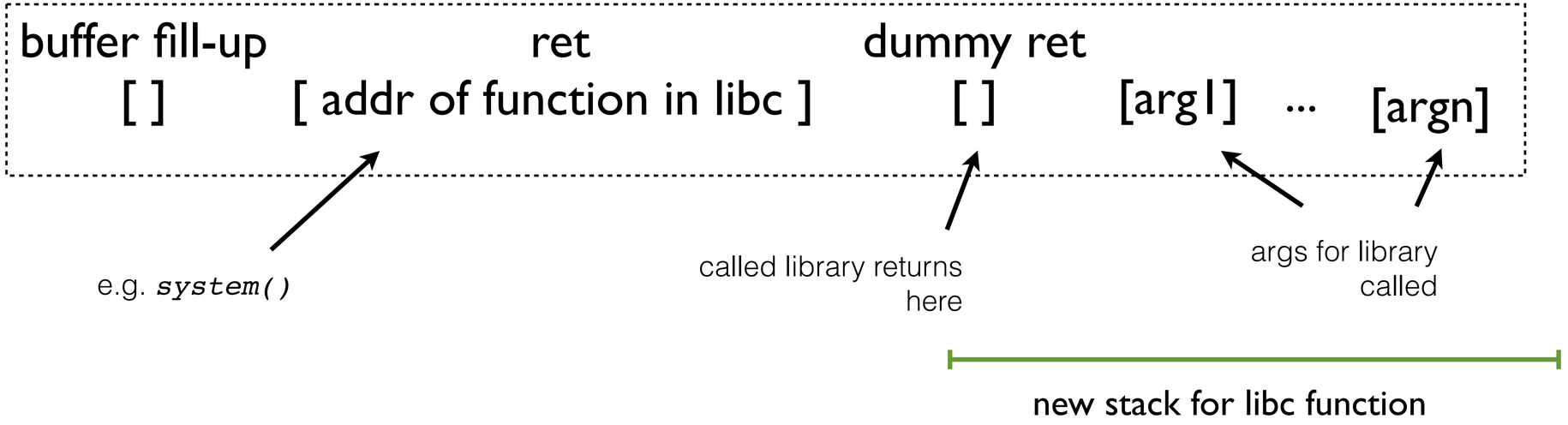
after overflowing buffer:



return-into-libc: stack after buffer overflow

memory addr grow ⟶

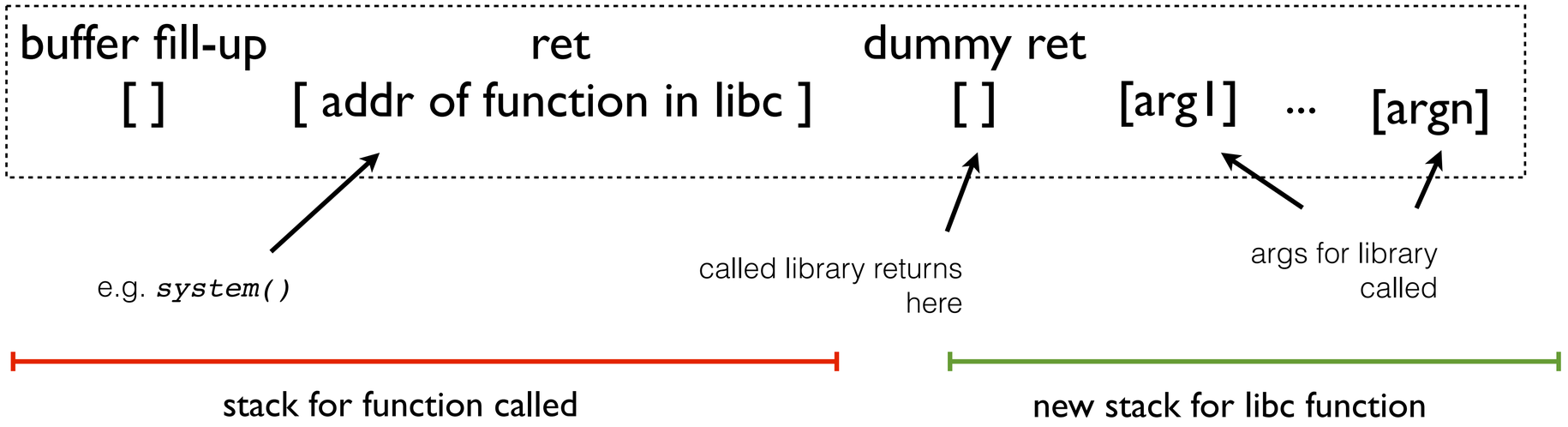
after overflowing buffer:



return-into-libc: stack after buffer overflow

memory addr grow ⟶

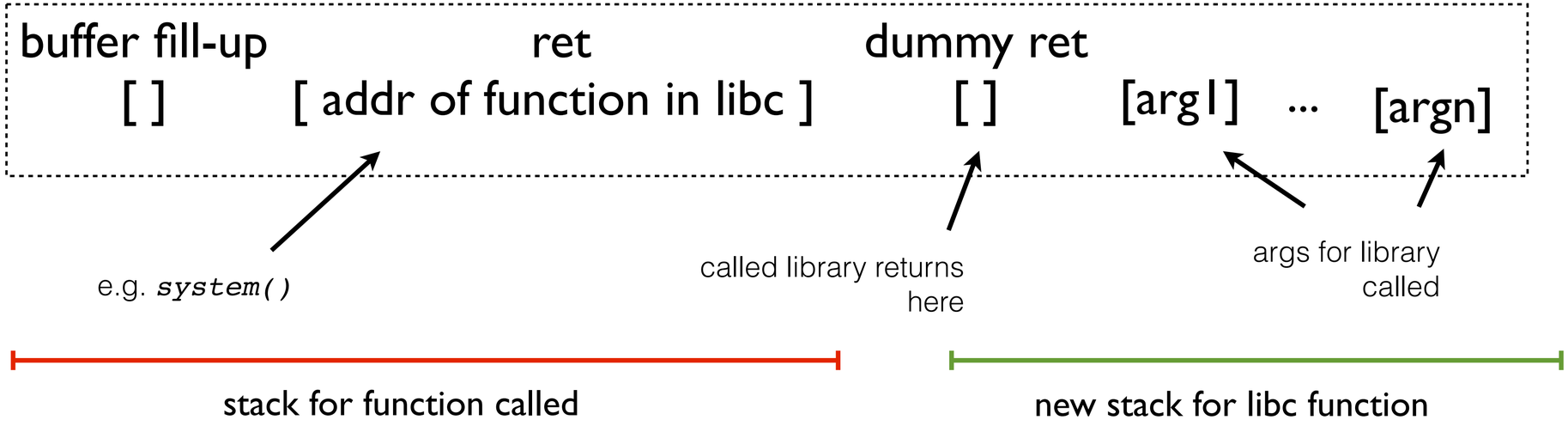
after overflowing buffer:



return-into-libc: stack after buffer overflow

memory addr grow ⟶

after overflowing buffer:

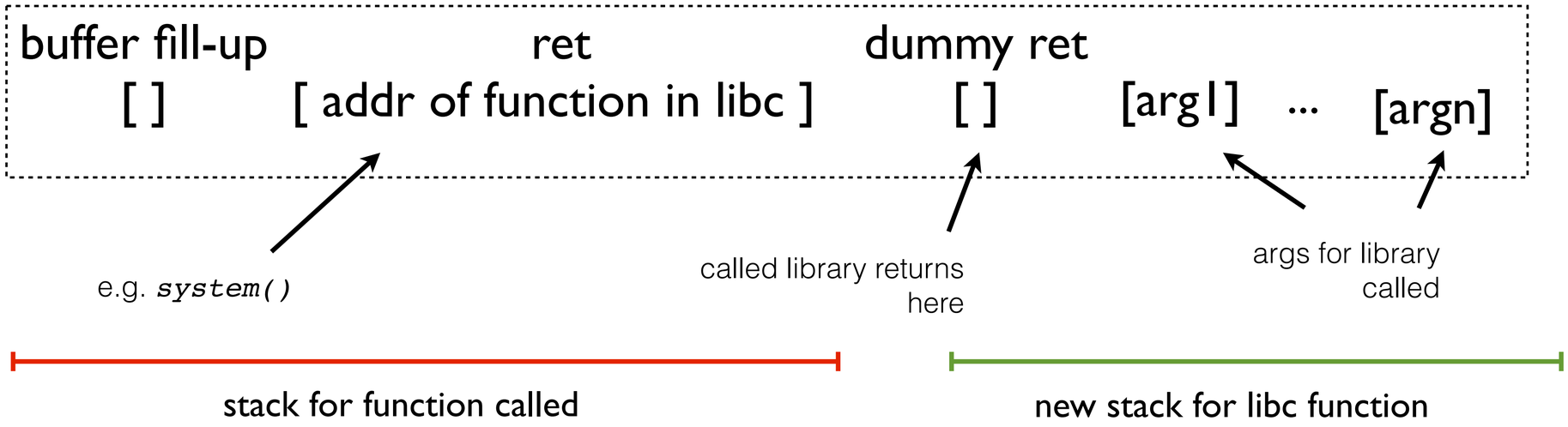


Q: limitation?

return-into-libc: stack after buffer overflow

memory addr grow ⟶

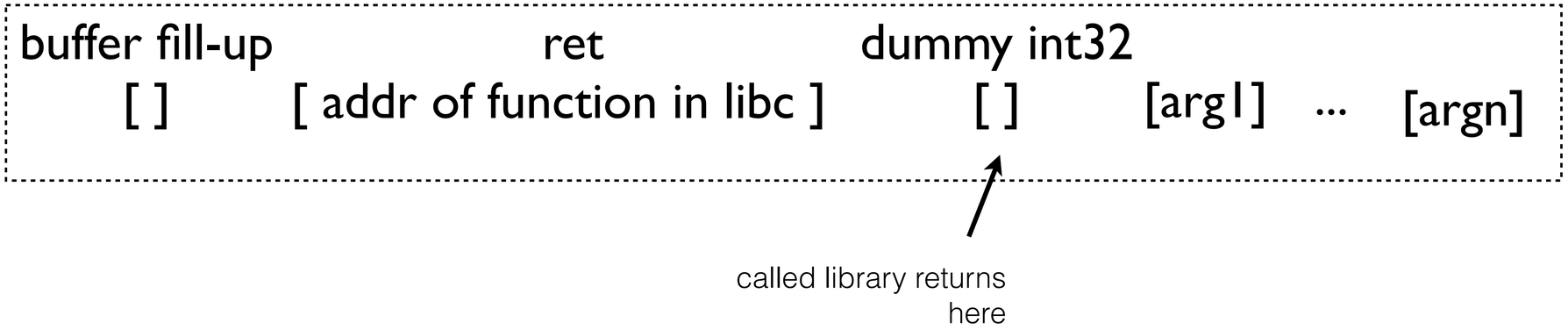
after overflowing buffer:



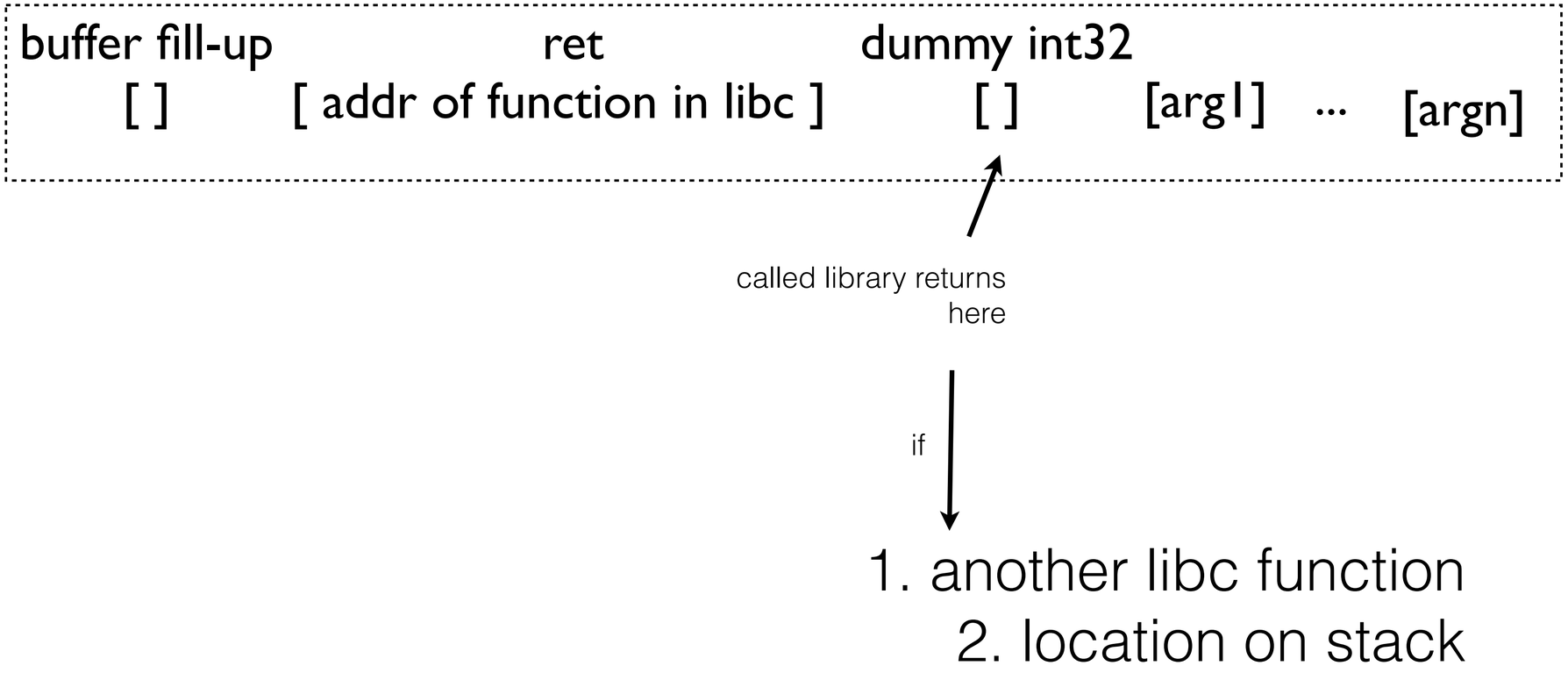
Q: limitation?

A: one function call

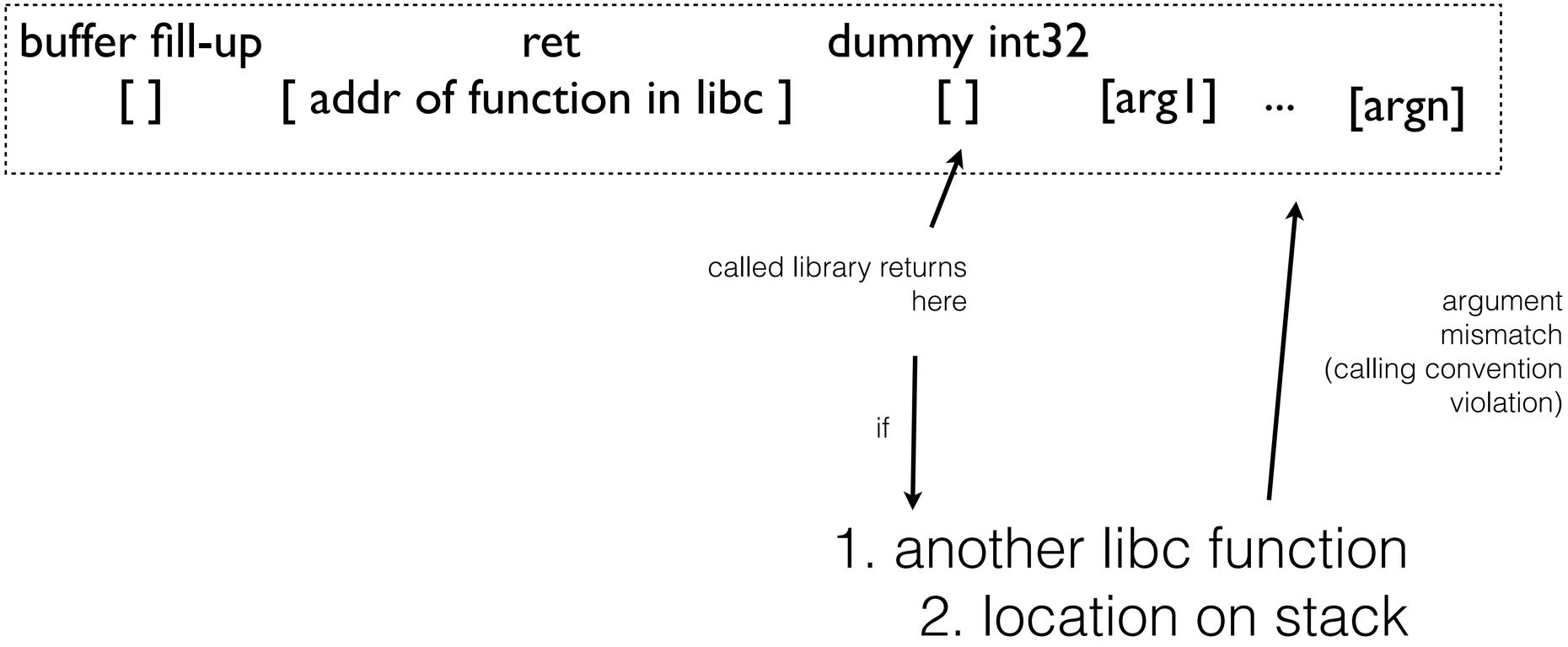
return-into-libc: one function call



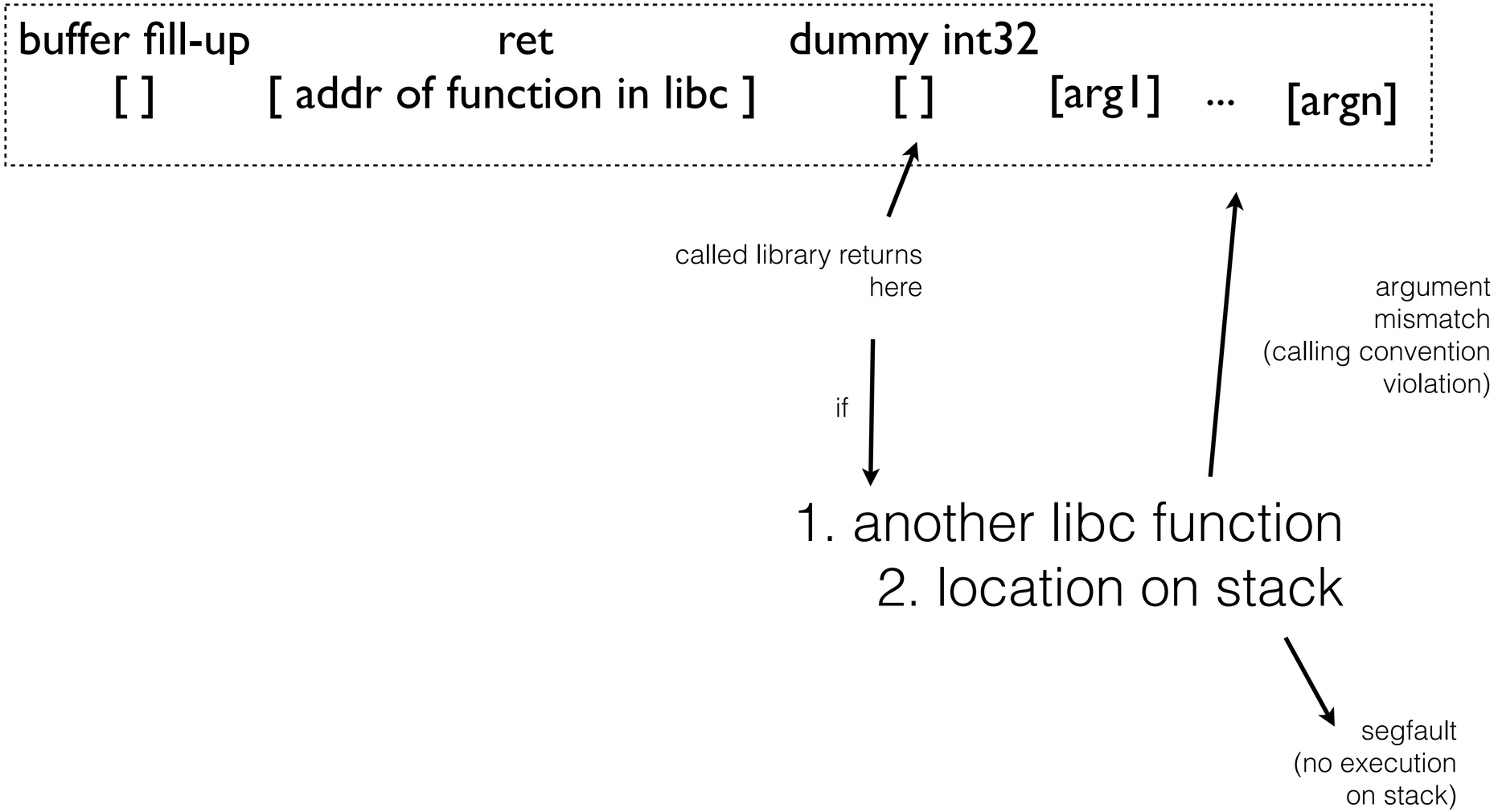
return-into-libc: one function call



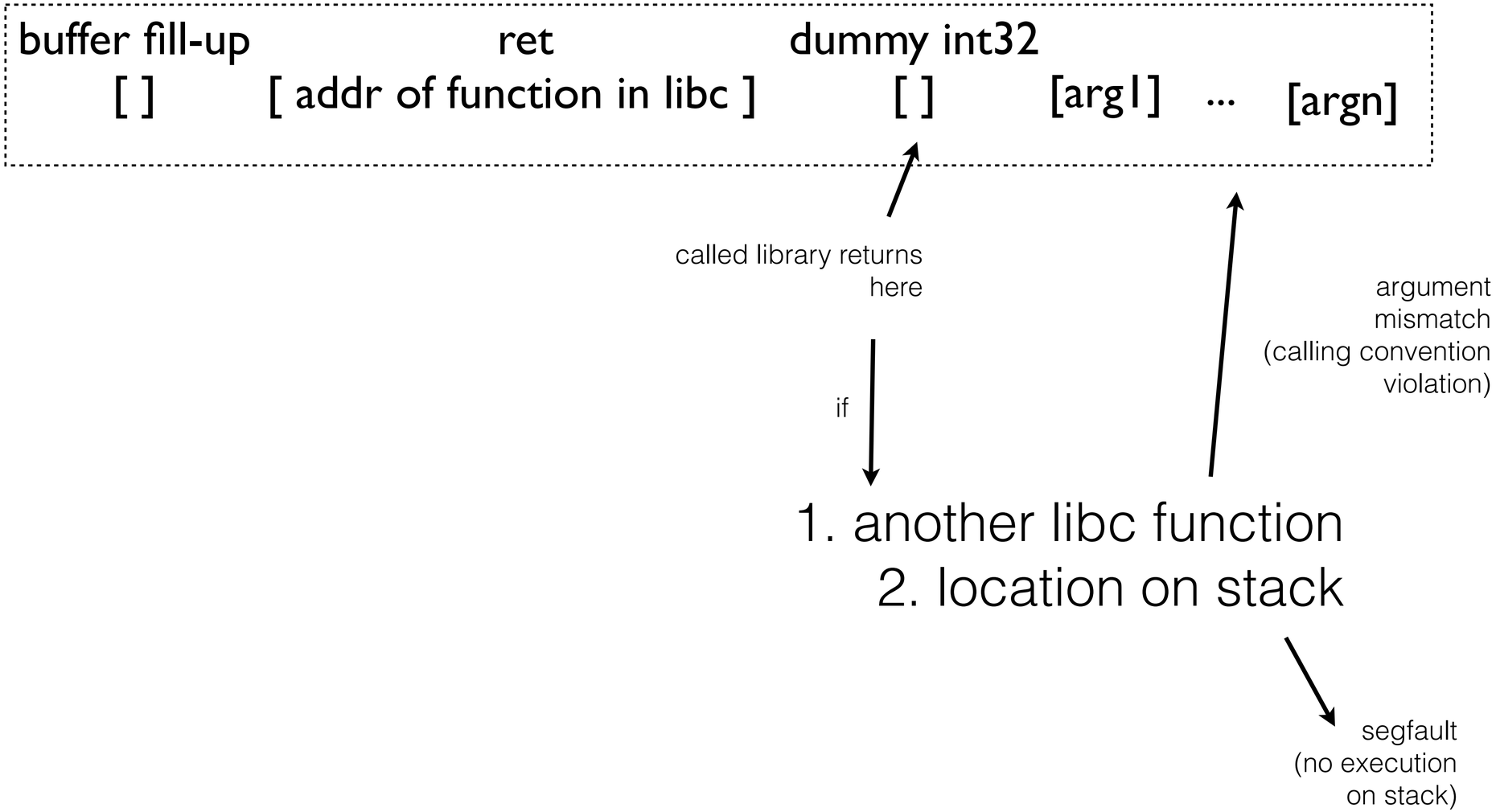
return-into-libc: one function call



return-into-libc: one function call



return-into-libc: one function call



stack is not prepared for another call

classic return-into-libc prevention:

```
seteuid(getuid())
```

classic return-into-libc prevention:

```
seteuid(getuid())
```

program drops privileges

1. `getuid`: get uid of calling process
2. `seteuid`: set uid of current process

classic return-into-libc prevention:

`seteuid(getuid())`

program drops privileges

1. `getuid`: get uid of calling process
2. `seteuid`: set uid of current process

for exploit need following:

1. `setuid()` (restore privileges)
2. libc function

classic return-into-libc prevention:

`seteuid(getuid())`

program drops privileges

1. `getuid`: get uid of calling process
2. `seteuid`: set uid of current process

for exploit need following:

1. `setuid()` (restore privileges)
2. libc function

two calls....

`seteuid(getuid())` example

`seteuid(getuid())` example

`passwd`:

`seteuid(getuid())` example

`passwd`:

1. `seteuid(getuid())`

`seteuid(getuid())` example

`passwd`:

1. `seteuid(getuid())`
2. take user input

`seteuid(getuid())` example

`passwd`:

1. `seteuid(getuid())`
2. take user input
3. hash

`seteuid(getuid())` example

`passwd`:

1. `seteuid(getuid())`
2. take user input
3. hash
4. `setuid()`

`seteuid(getuid())` example

`passwd`:

1. `seteuid(getuid())`
2. take user input
3. hash
4. `setuid()`
5. write hash

`seteuid(getuid())` example

`passwd`:

1. `seteuid(getuid())`
2. take user input
3. hash
4. `setuid()`
5. write hash
6. exit

`seteuid(getuid())` example

`passwd`:

1. `seteuid(getuid())`
2. take user input
3. hash
4. `setuid()`
5. write hash
6. exit

exploit here:

1. `setuid()` (restore privileges)
2. libc function

advanced return-into-libc: `esp` lifting (1)

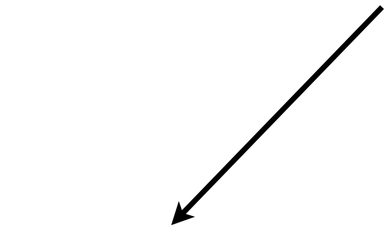
if program compiled with:

`-fomit-frame-pointer`

advanced return-into-libc: `esp` lifting (1)

if program compiled with:

`-fomit-frame-pointer`



epilogue:

code at end of function

advanced return-into-libc: `esp` lifting (1)

if program compiled with:

`-fomit-frame-pointer`

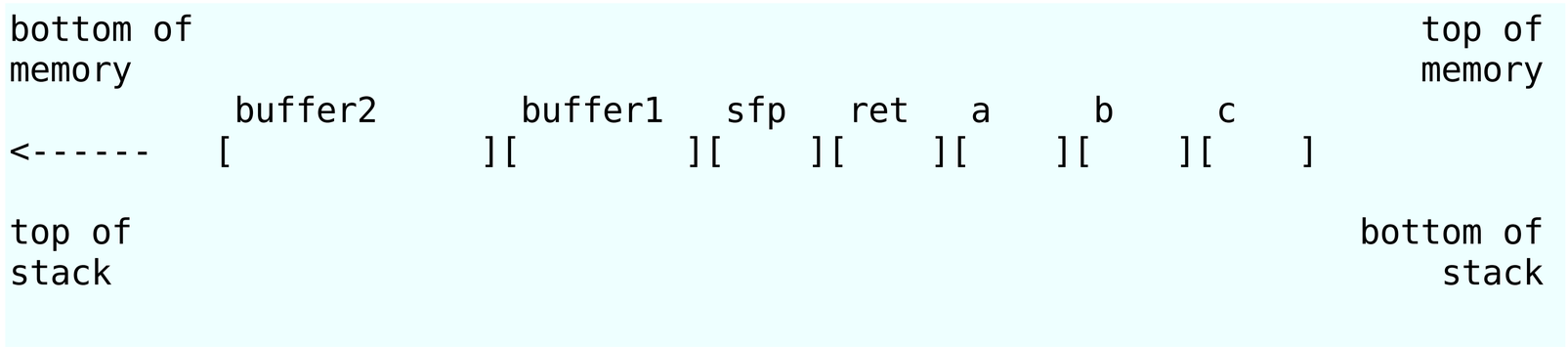
epilogue:

```
eplg:
        addl    $LOCAL_VARS_SIZE,%esp
        ret
```

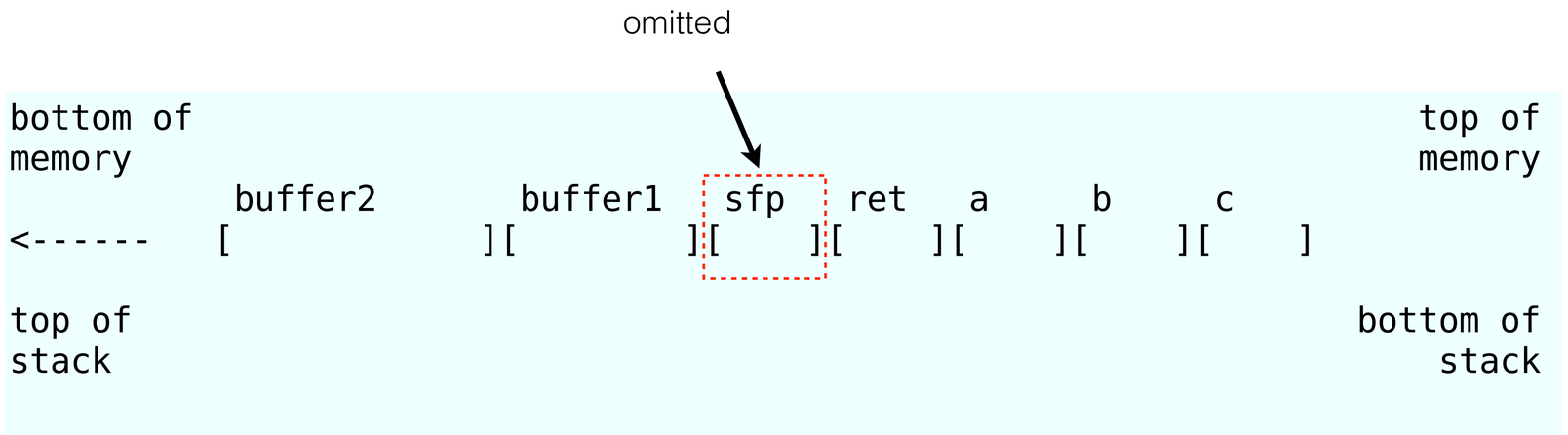
resetting stack pointer

code at end of function

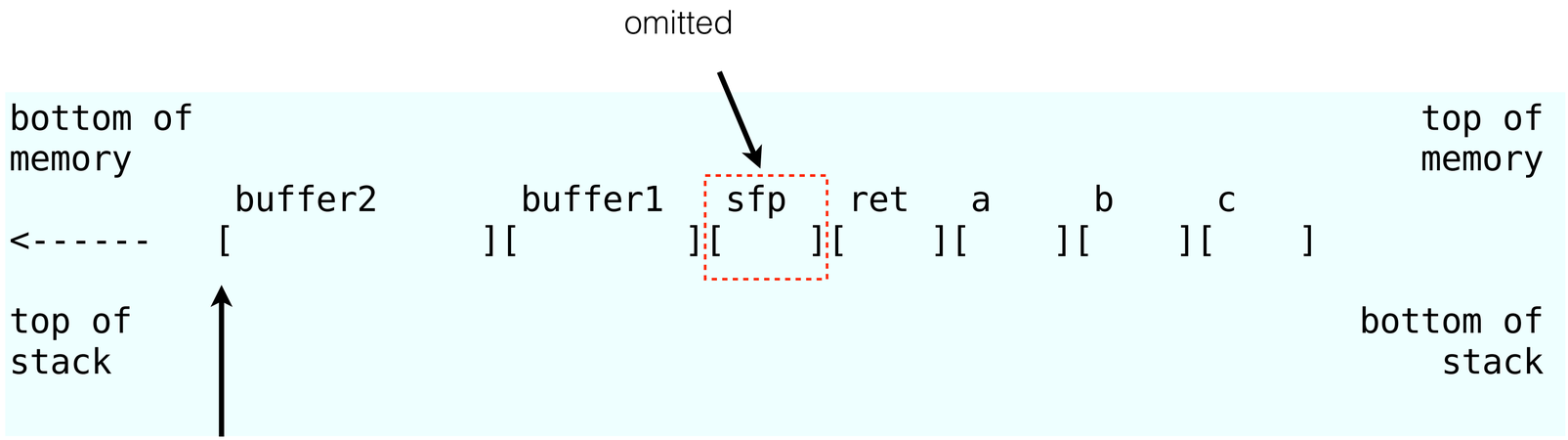
-fomit-frame-pointer



-fomit-frame-pointer

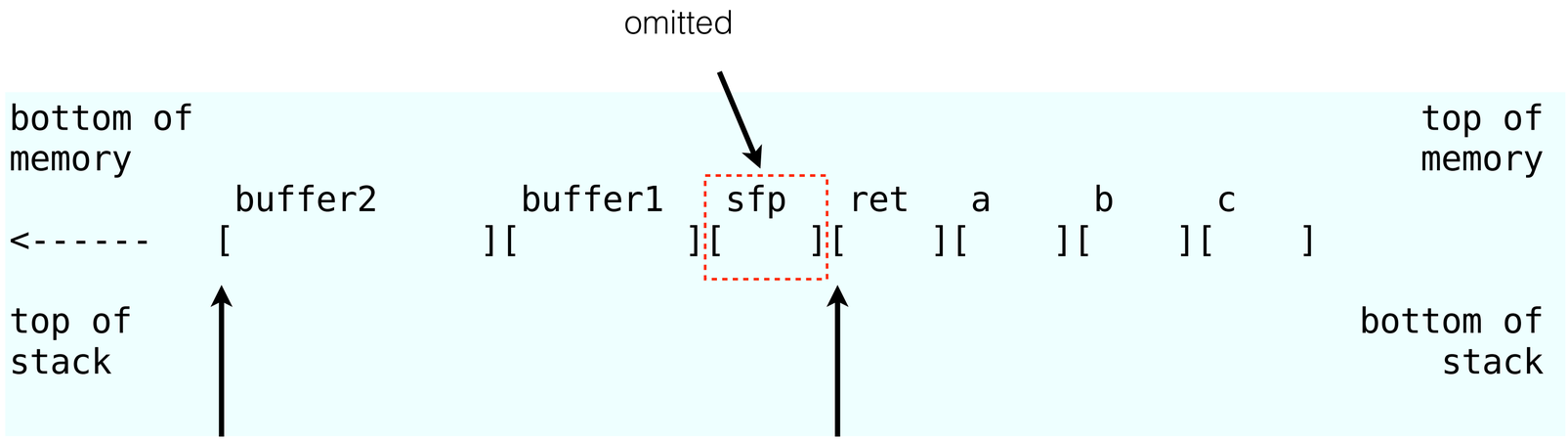


-fomit-frame-pointer



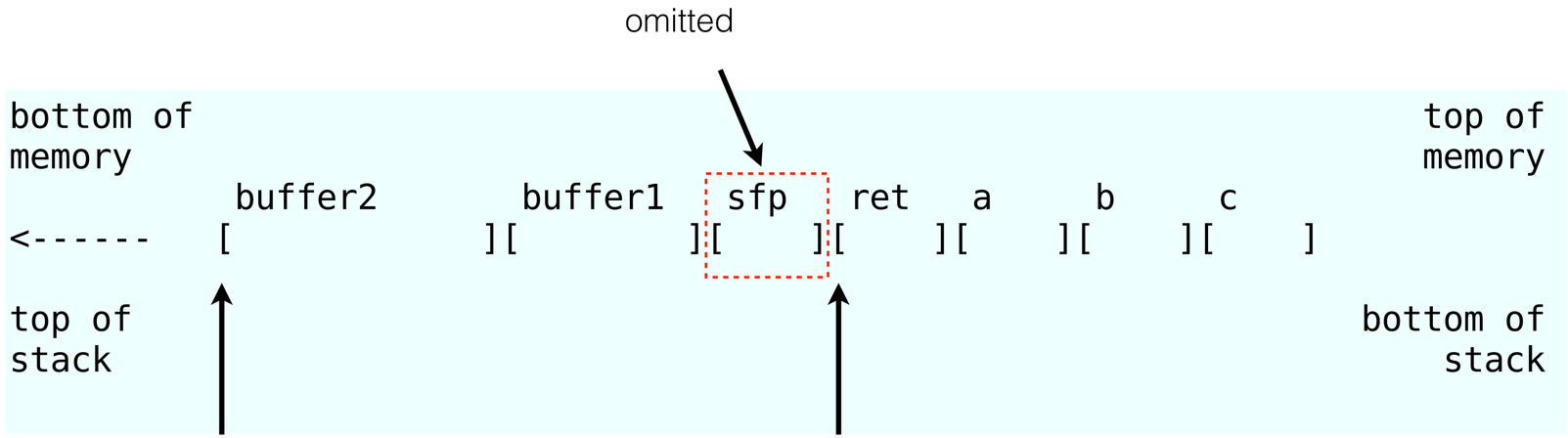
before **`eplg`**, **`%esp`**

-fomit-frame-pointer



before **eplg**, **%esp** after **eplg**, **%esp**

-fomit-frame-pointer



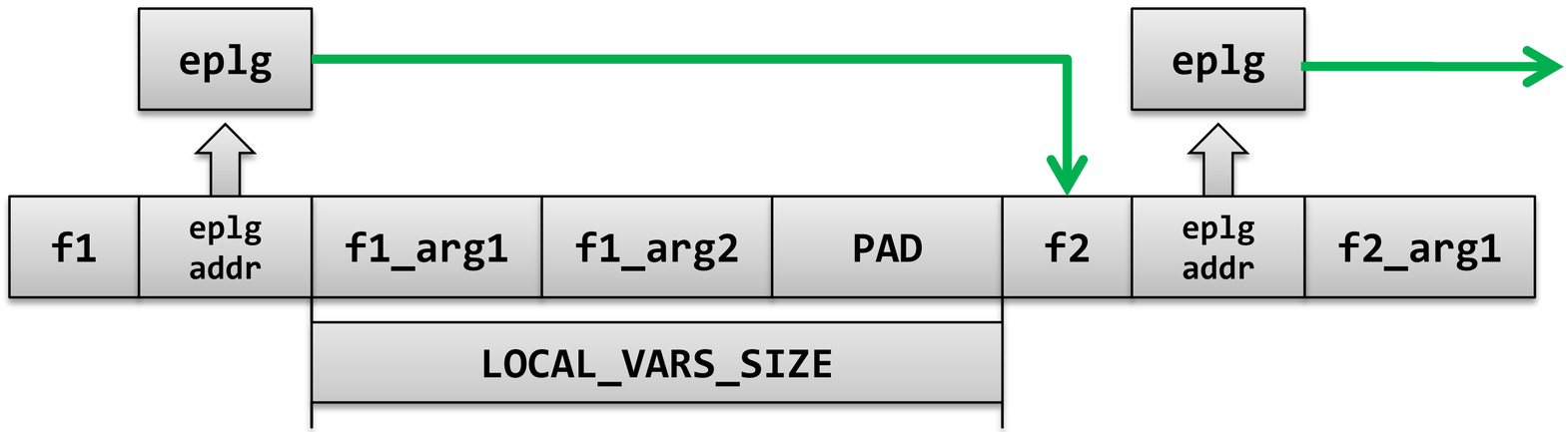
before **eplg**, **%esp** after **eplg**, **%esp**

```
eplg:
        addl    $LOCAL_VARS_SIZE,%esp
        ret
```

pops **ret** addr. off stack and branches to it

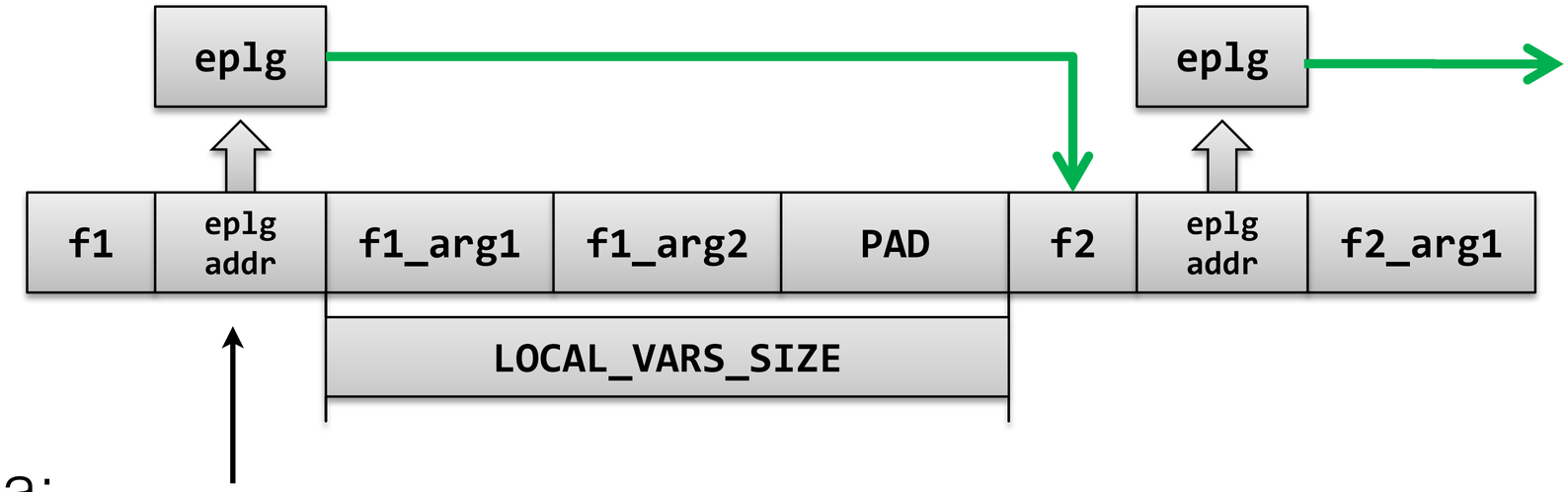
advanced return-into-libc: `esp` lifting (1)

(libc [or other function] should return to epilog of exploited [or other] function)



advanced return-into-libc: `esp` lifting (1)

(libc [or other function] should return to epilog of exploited [or other] function)



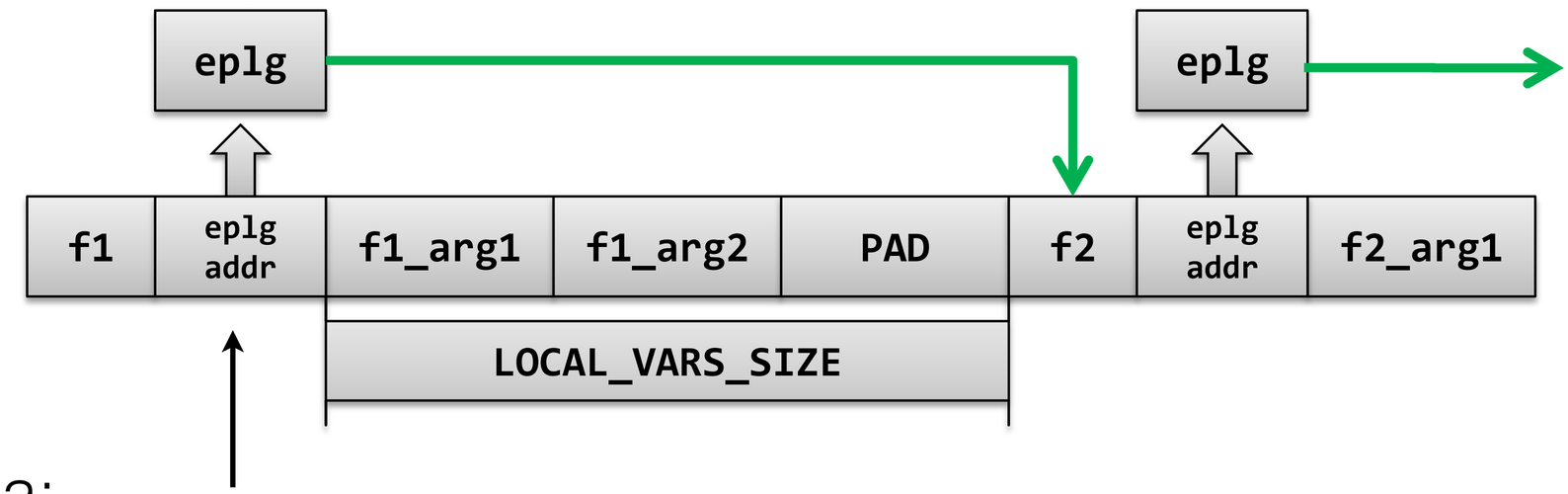
idea:

1. function `f1` (libc) returns to epilogue

advanced return-into-libc: `esp` lifting (1)

(libc [or other function] should return to epilog of exploited [or other] function)

2. epilogue moves stack pointer to so that `ret` calls function `f2` (libc)



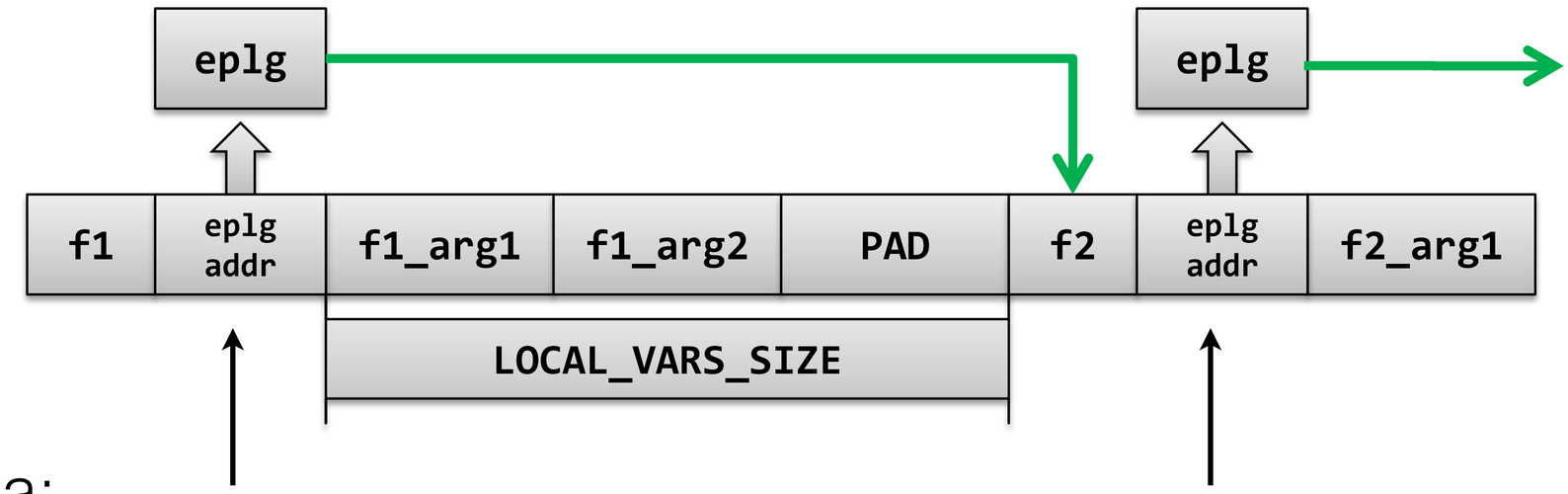
idea:

1. function `f1` (libc) returns to epilogue

advanced return-into-libc: `esp` lifting (1)

(libc [or other function] should return to epilog of exploited [or other] function)

2. epilogue moves stack pointer to so that `ret` calls function `f2` (libc)



idea:

1. function `f1` (libc) returns to epilogue

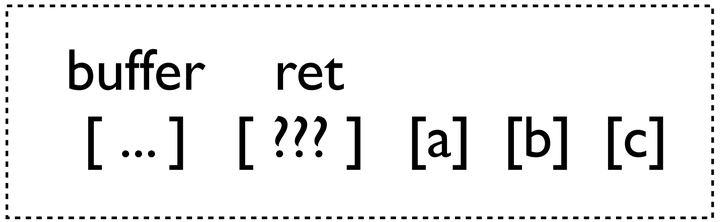
3. function `f2` (libc) returns to epilogue

advanced return-into-libc: `esp` lifting (1)

(stack we build: part one)

stack after call:

```
[function(int a, int b, int c)]
```

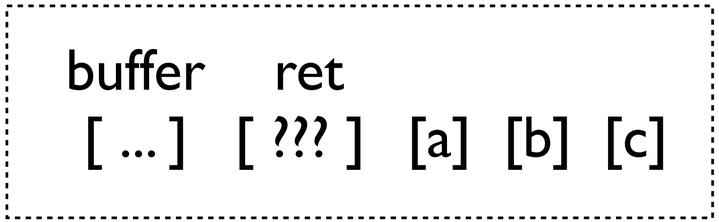


advanced return-into-libc: `esp` lifting (1)

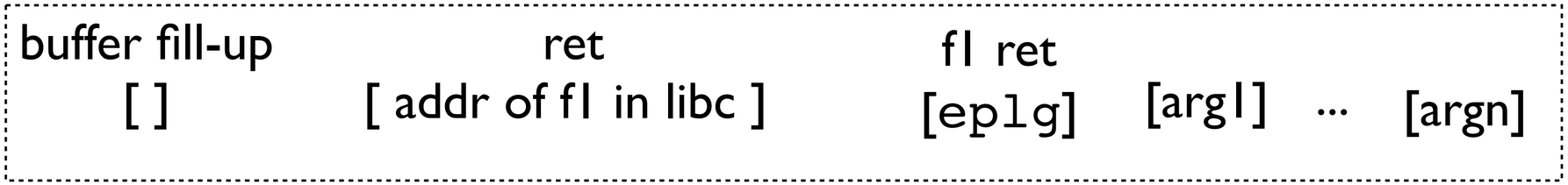
(stack we build: part one)

stack after call:

```
[function(int a, int b, int c)]
```



after overflowing buffer:

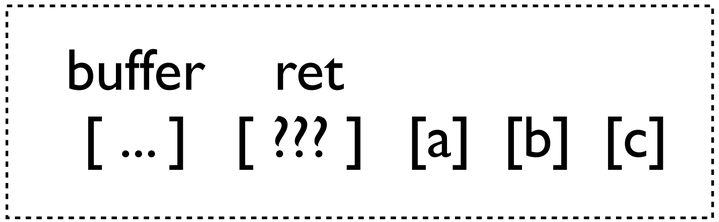


advanced return-into-libc: `esp` lifting (1)

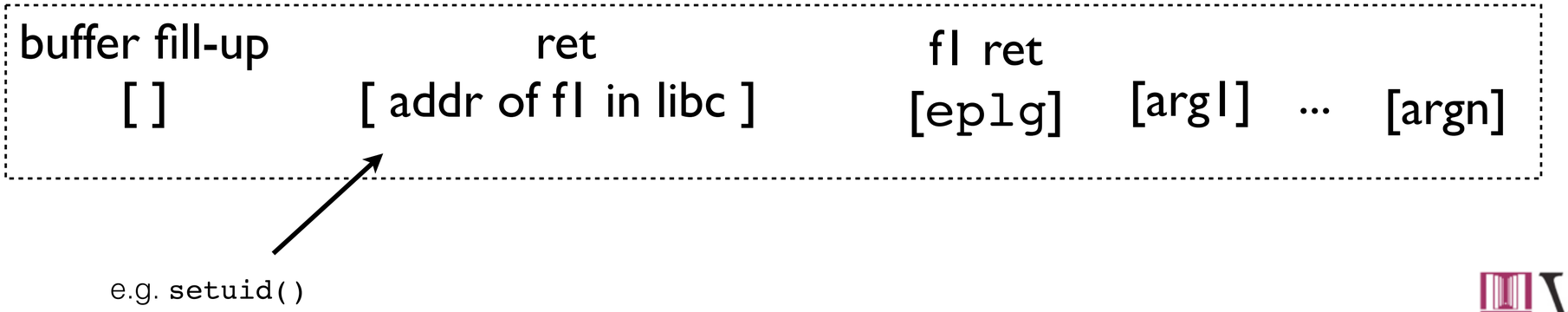
(stack we build: part one)

stack after call:

```
[function(int a, int b, int c)]
```



after overflowing buffer:

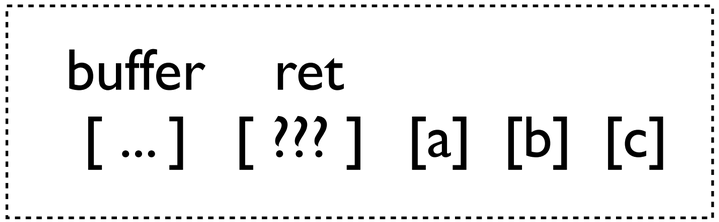


advanced return-into-libc: `esp` lifting (1)

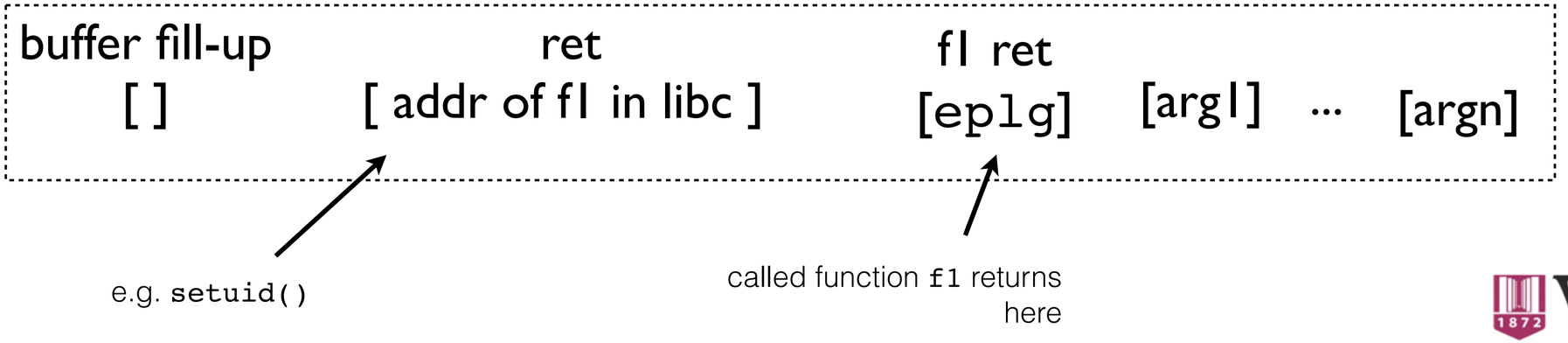
(stack we build: part one)

stack after call:

`[function(int a, int b, int c)]`



after overflowing buffer:

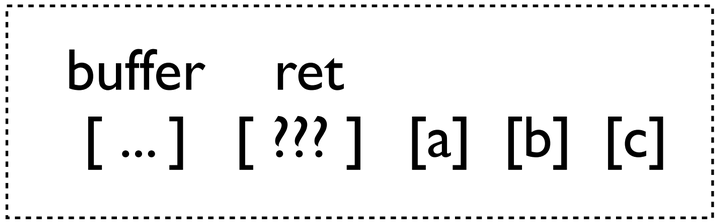


advanced return-into-libc: `esp` lifting (1)

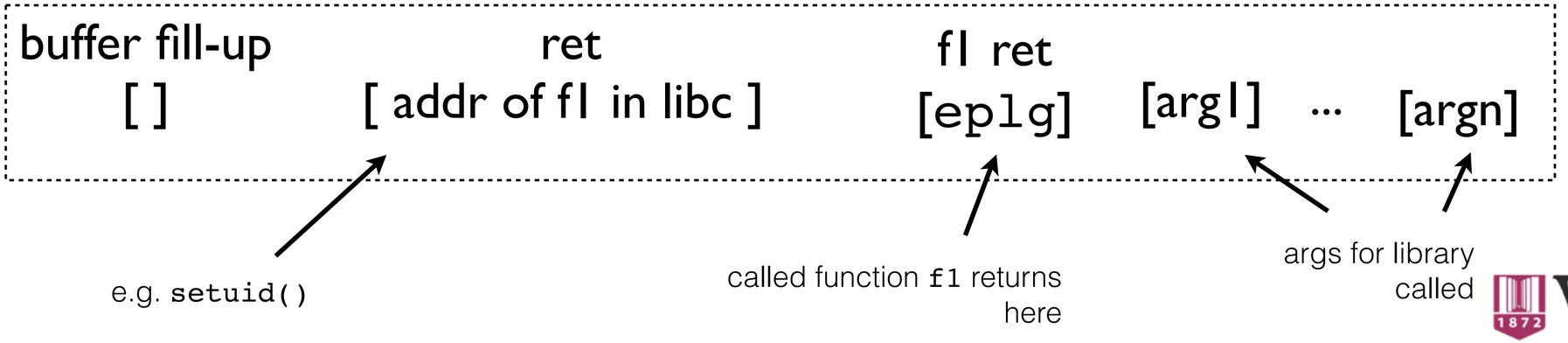
(stack we build: part one)

stack after call:

`[function(int a, int b, int c)]`



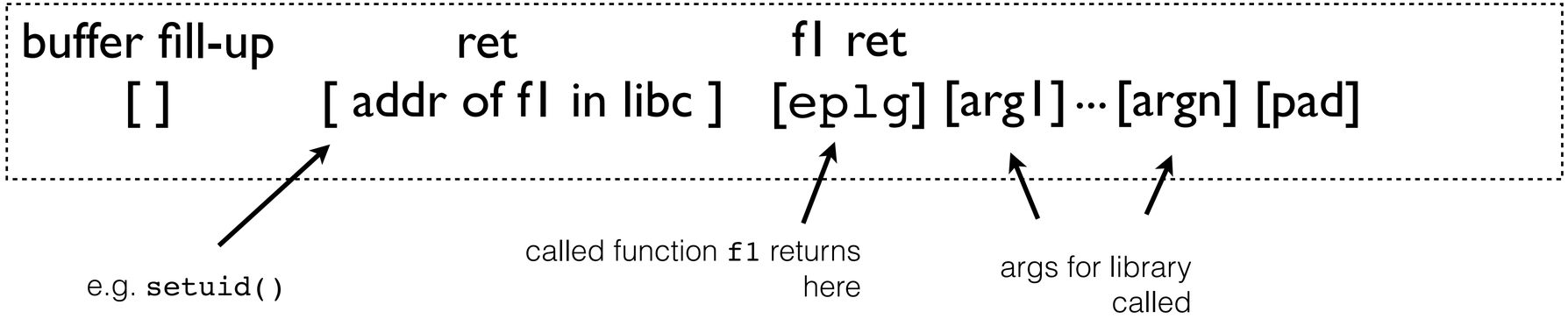
after overflowing buffer:



advanced return-into-libc: `esp` lifting (1)

(stack we build: part two)

after overflowing buffer:

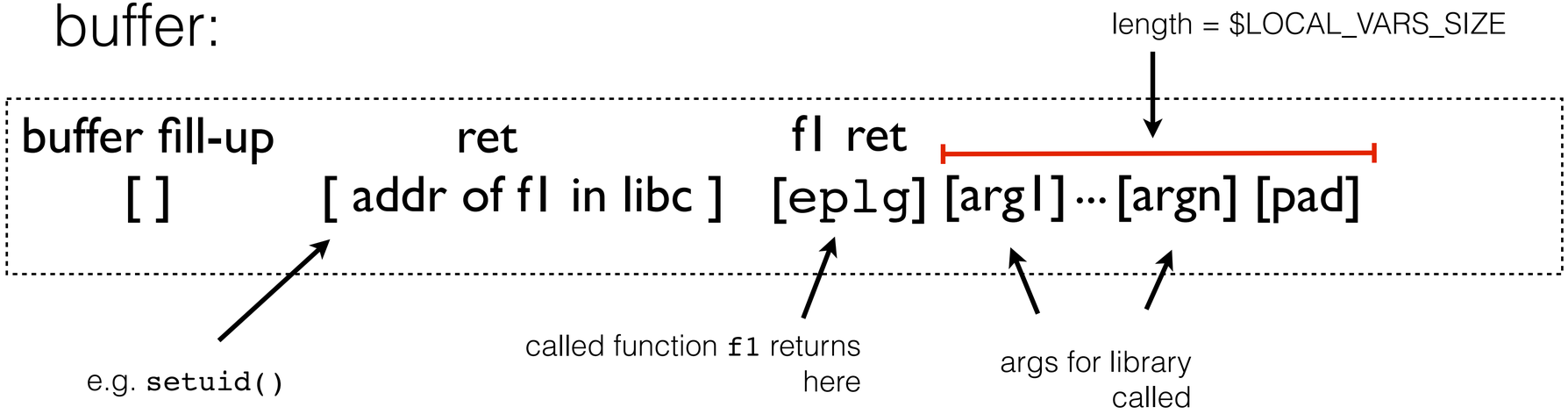



```
eplg:
        addl    $LOCAL_VARS_SIZE,%esp
        ret
```

advanced return-into-libc: `esp` lifting (1)

(stack we build: part two)

after overflowing buffer:

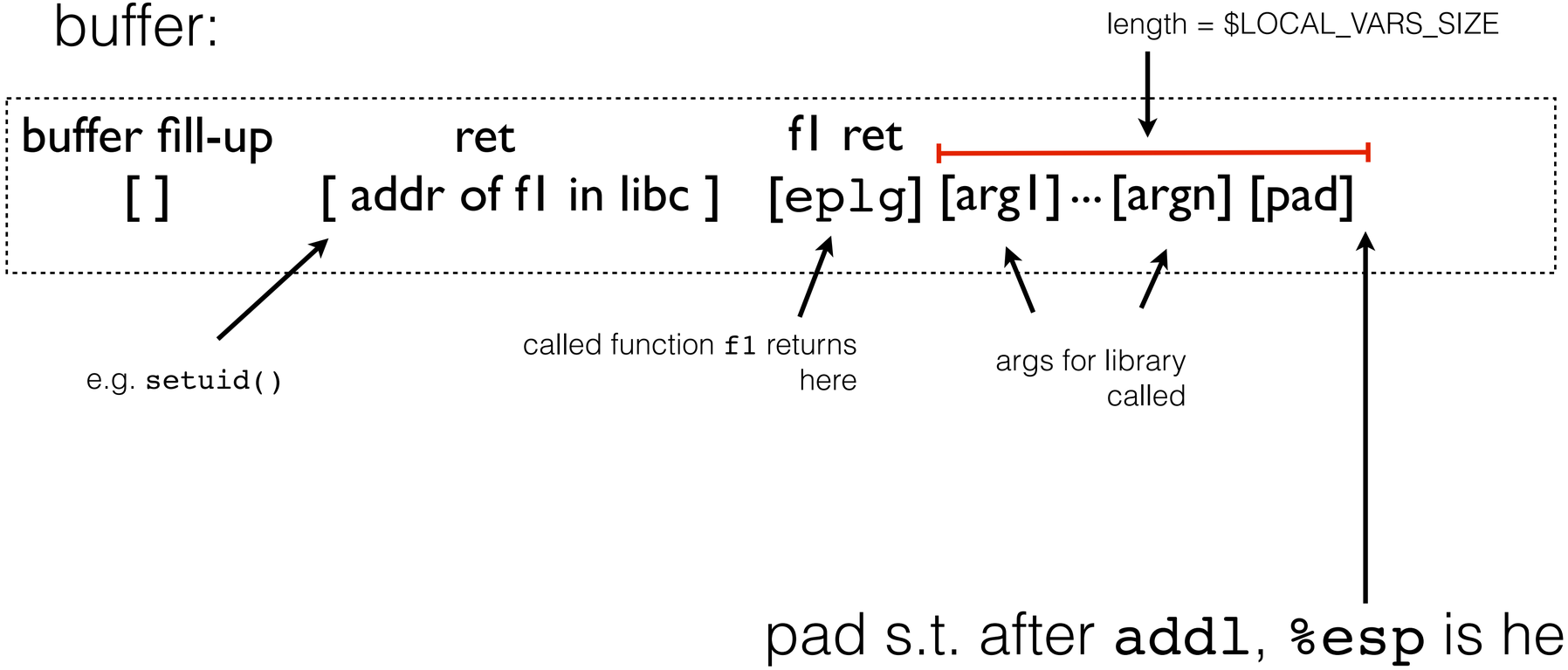



```
eplg:
        addl    $LOCAL_VARS_SIZE,%esp
        ret
```

advanced return-into-libc: `esp` lifting (1)

(stack we build: part two)

after overflowing buffer:

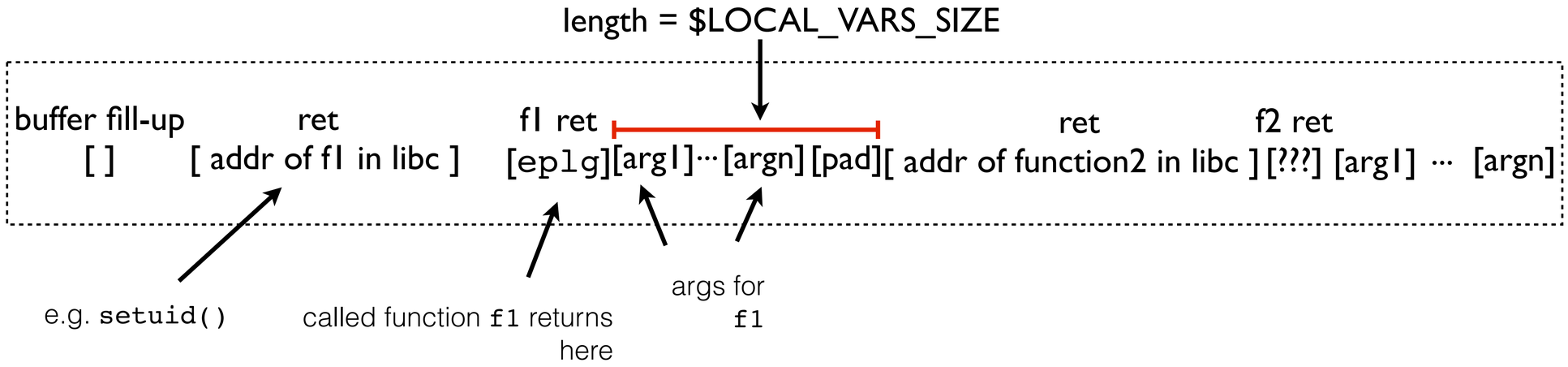



```
eplg:
        addl    $LOCAL_VARS_SIZE,%esp
        ret
```

advanced return-into-libc: `esp` lifting (1)

(stack we build: part three)

after overflowing buffer:

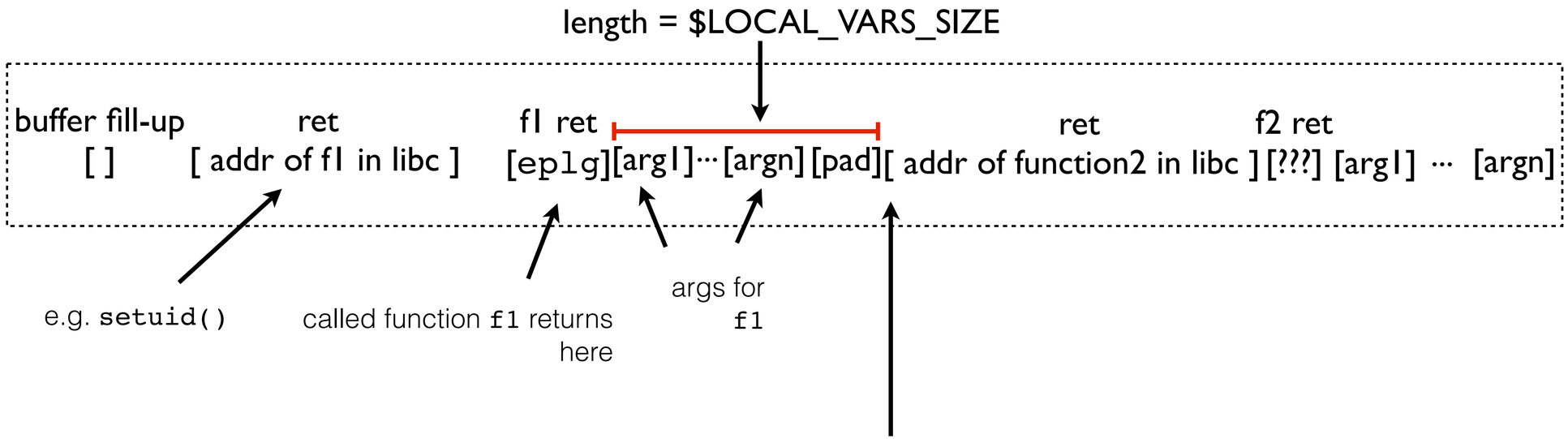



```
eplg:
        addl    $LOCAL_VARS_SIZE,%esp
        ret
```

advanced return-into-libc: `esp` lifting (1)

(stack we build: part three)

after overflowing buffer:



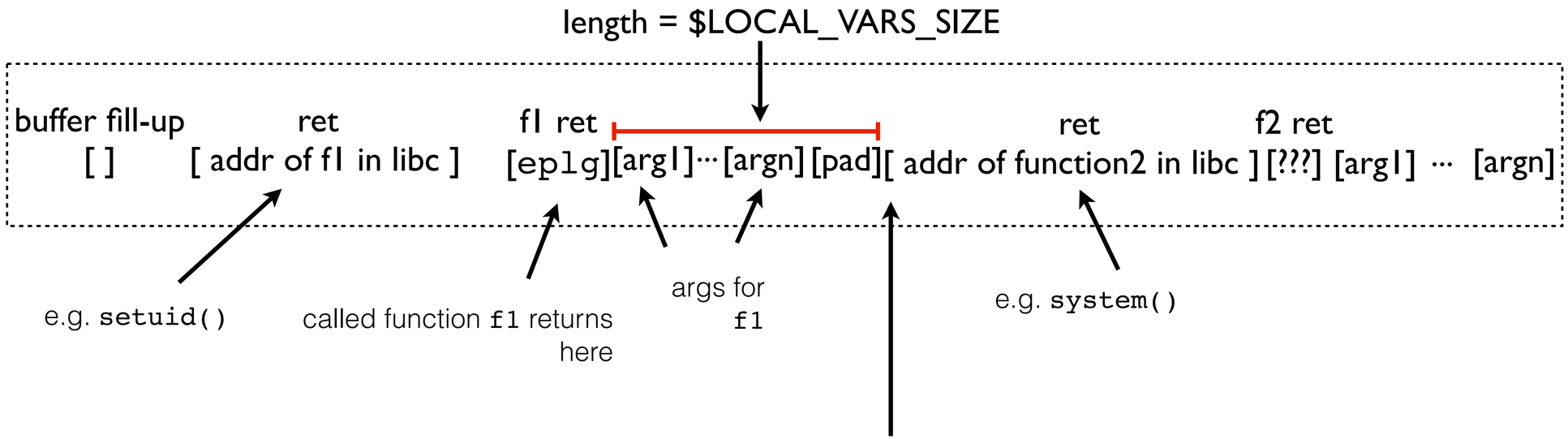
`ret` of `eplg` pops and uses this addr

```
eplg:
        addl    $LOCAL_VARS_SIZE,%esp
        ret
```

advanced return-into-libc: `esp` lifting (1)

(stack we build: part three)

after overflowing buffer:



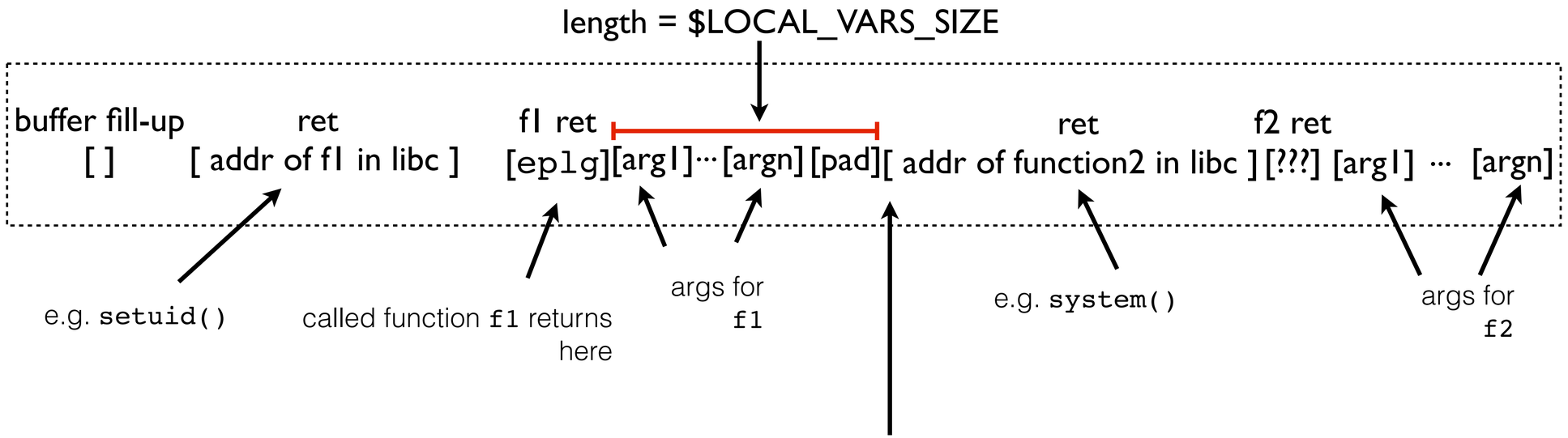
`ret` of `eplg` pops and uses this addr

```
eplg:
        addl    $LOCAL_VARS_SIZE,%esp
        ret
```

advanced return-into-libc: `esp` lifting (1)

(stack we build: part three)

after overflowing buffer:



`ret` of **`eplg`** pops and uses this addr

```
eplg:
        addl    $LOCAL_VARS_SIZE,%esp
        ret
```

advanced return-into-libc: `esp` lifting (1)

(execution)

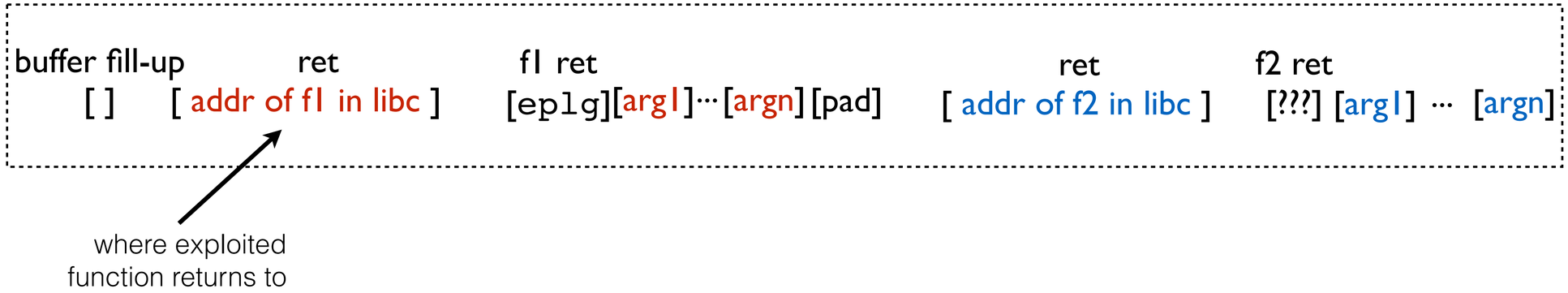
after overflowing buffer:


```
eplg:
        addl    $LOCAL_VARS_SIZE,%esp
        ret
```

advanced return-into-libc: `esp` lifting (1)

(execution)

after overflowing buffer:

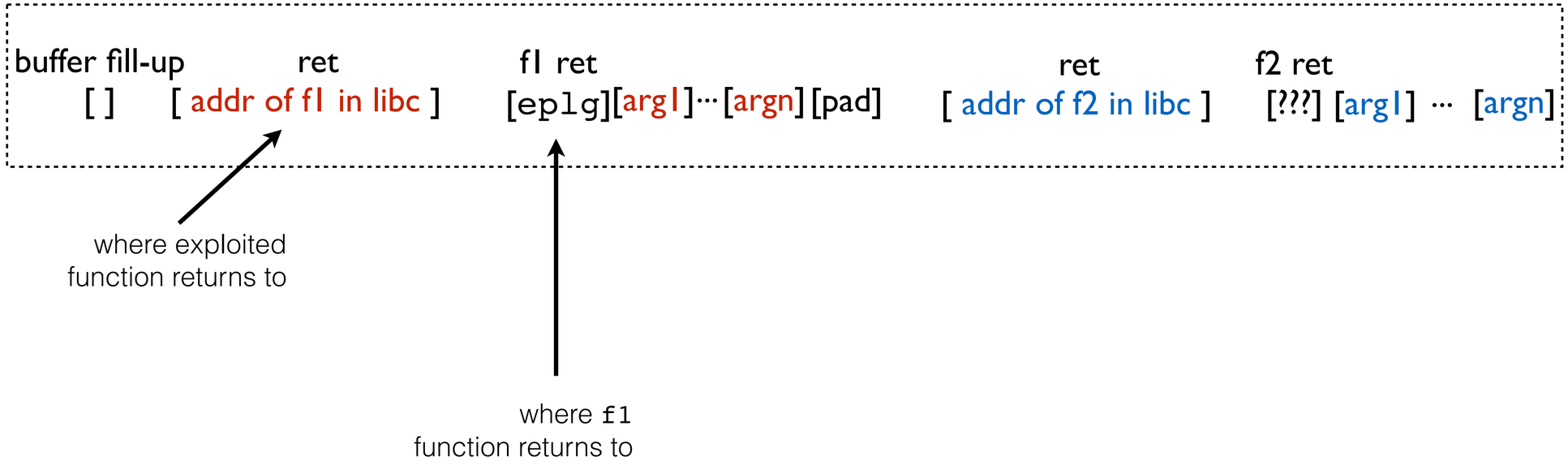



```
eplg:
        addl    $LOCAL_VARS_SIZE,%esp
        ret
```

advanced return-into-libc: `esp` lifting (1)

(execution)

after overflowing buffer:

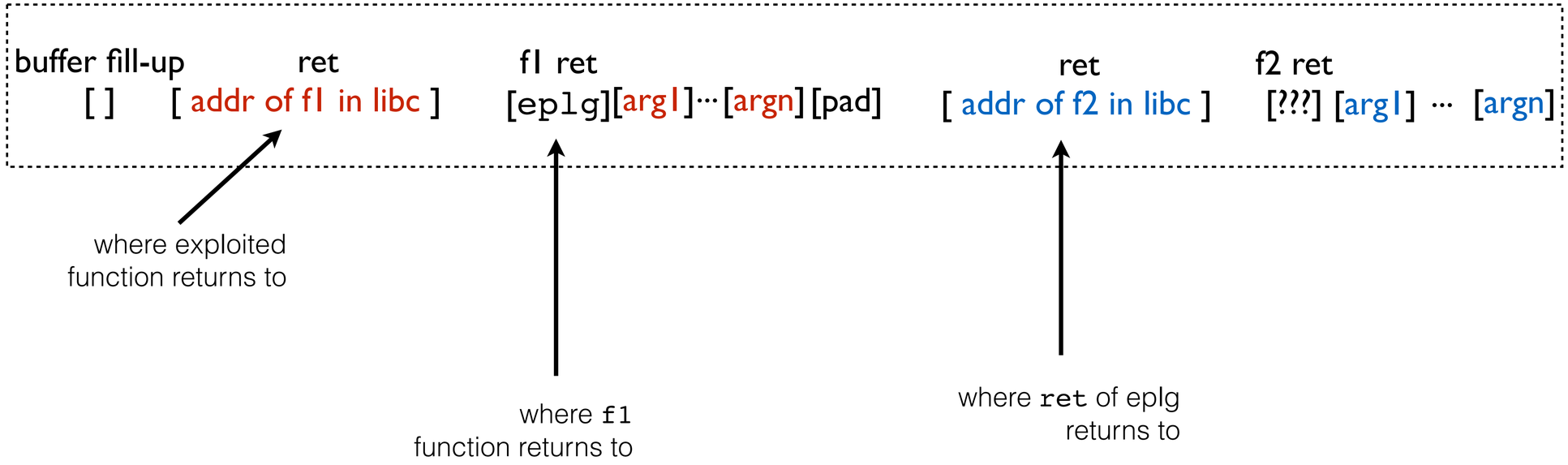



```
eplg:
        addl    $LOCAL_VARS_SIZE,%esp
        ret
```

advanced return-into-libc: `esp` lifting (1)

(execution)

after overflowing buffer:

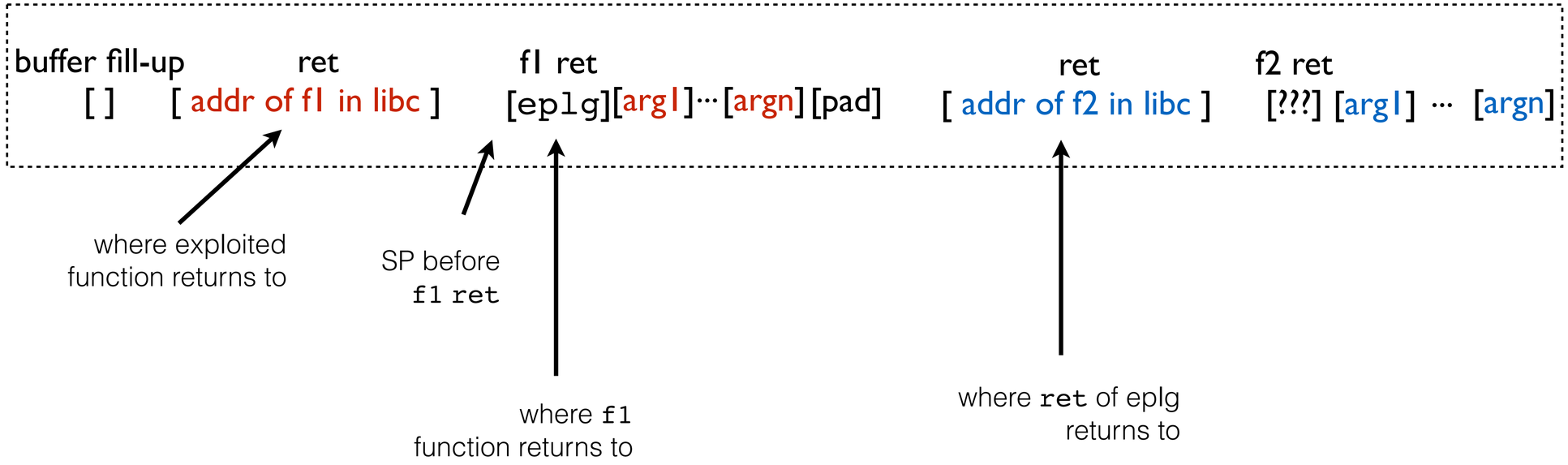



```
eplg:
        addl    $LOCAL_VARS_SIZE,%esp
        ret
```

advanced return-into-libc: `esp` lifting (1)

(execution)

after overflowing buffer:

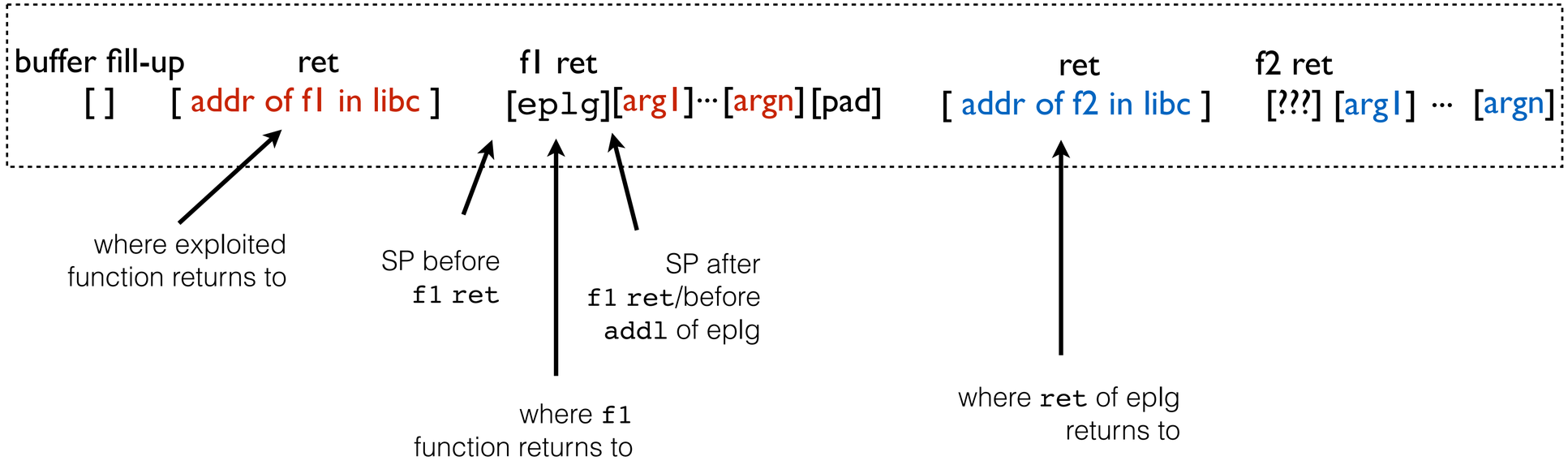



```
eplg:
        addl    $LOCAL_VARS_SIZE,%esp
        ret
```

advanced return-into-libc: `esp` lifting (1)

(execution)

after overflowing buffer:

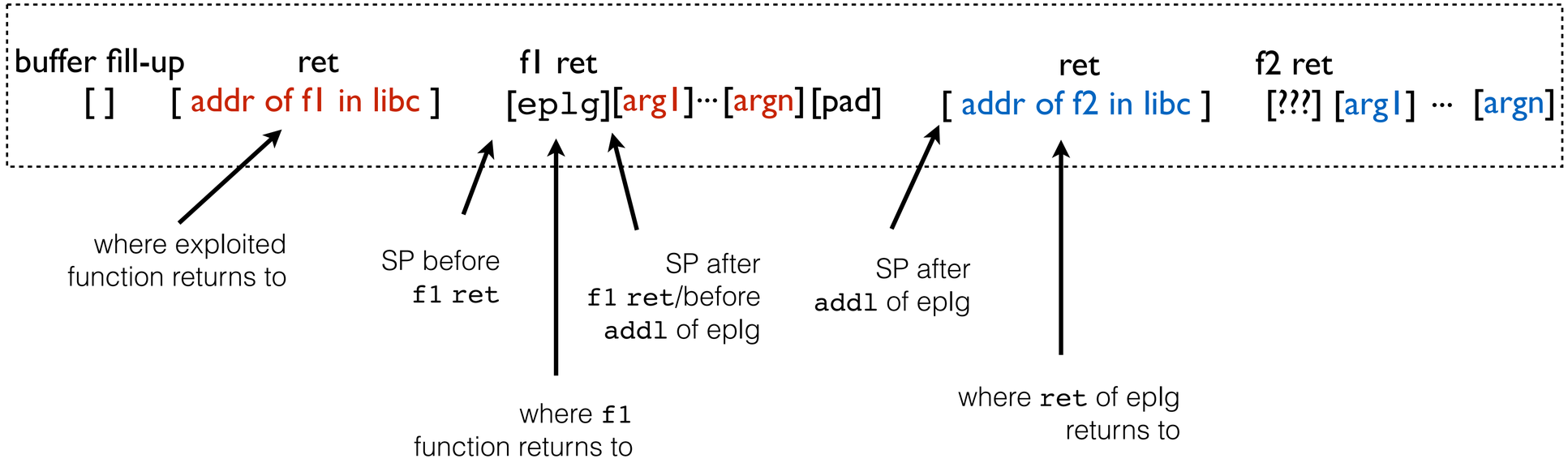



```
eplg:
        addl    $LOCAL_VARS_SIZE,%esp
        ret
```

advanced return-into-libc: `esp` lifting (1)

(execution)

after overflowing buffer:

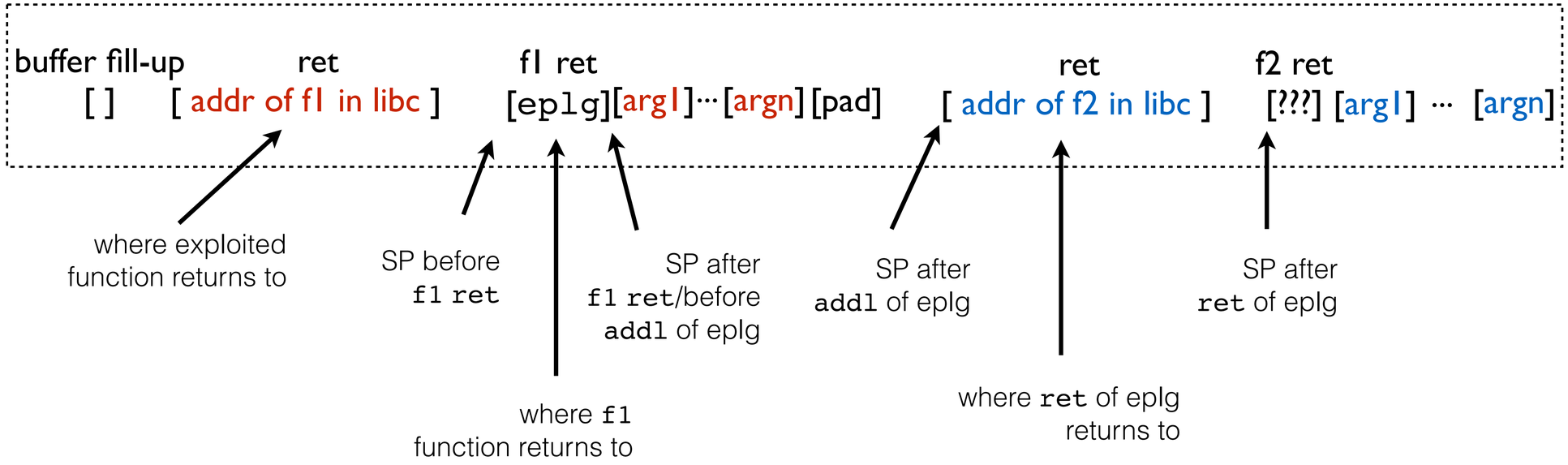



```
eplg:
        addl    $LOCAL_VARS_SIZE,%esp
        ret
```

advanced return-into-libc: `esp` lifting (1)

(execution)

after overflowing buffer:




```
eplg:
        addl    $LOCAL_VARS_SIZE,%esp
        ret
```

advanced return-into-libc: `esp` lifting (2)

(move stack pointer to functions via `pop`)

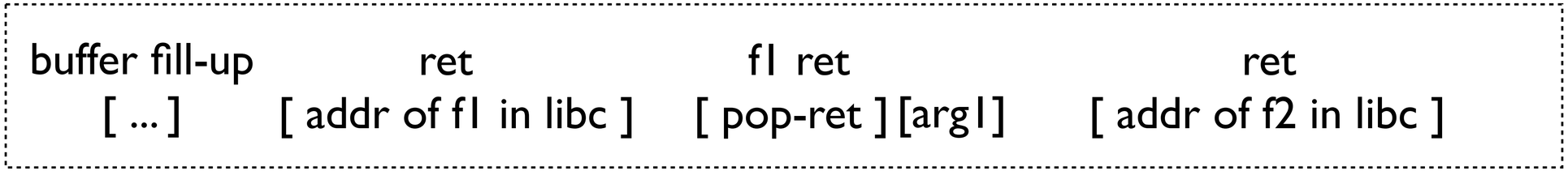
find addr. of following instructions in linked library(ies):

```
pop-ret:
        popl any_register
        ret
```

advanced return-into-libc: `esp` lifting (2)

(move stack pointer to functions via `pop`)

after overflowing buffer:

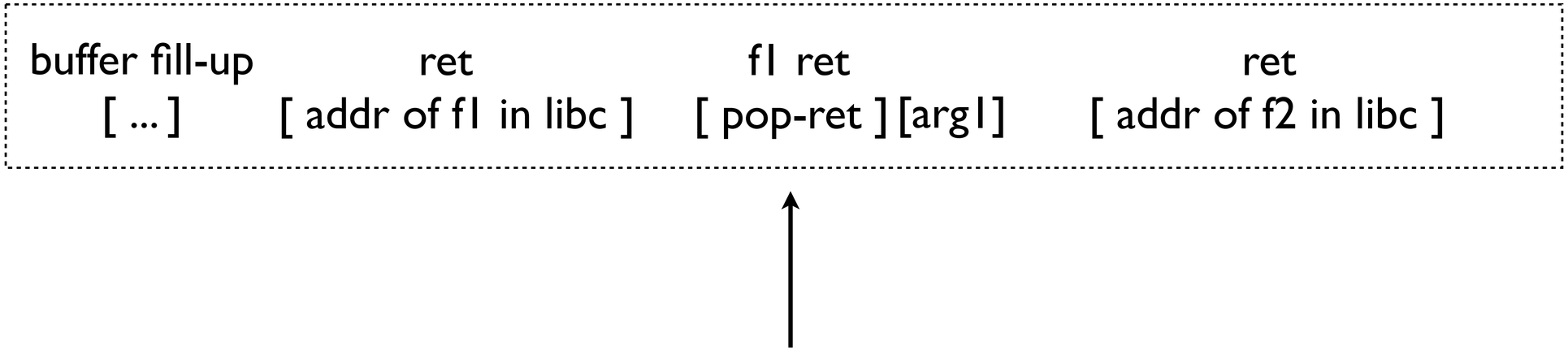



```
pop-ret:
        popl any_register
        ret
```

advanced return-into-libc: `esp` lifting (2)

(move stack pointer to functions via `pop`)

after overflowing buffer:



1. takes four bytes off of stack

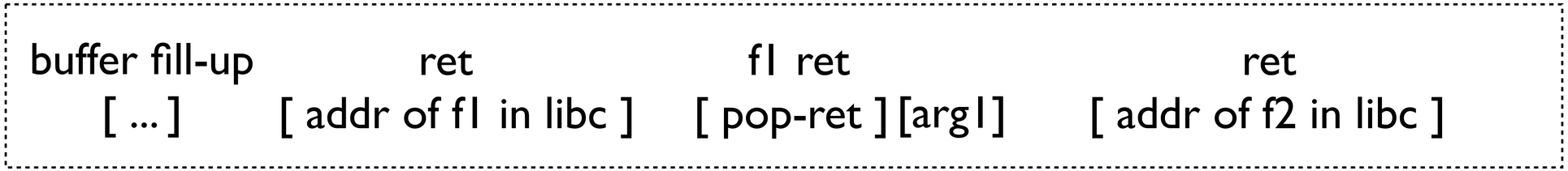
(SP moved four bytes back/up)

```
pop-ret:
        popl any_register
        ret
```

advanced return-into-libc: `esp` lifting (2)

(move stack pointer to functions via `pop`)

after overflowing buffer:



1. takes four bytes off of stack

(SP moved four bytes back/up)

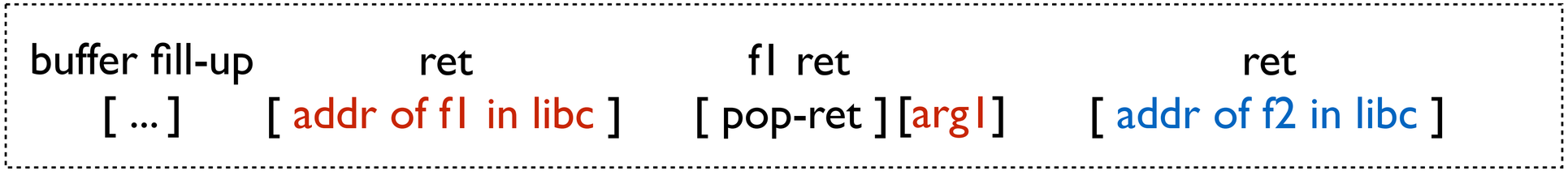
2. `ret` uses this addr after `popl`

```
pop-ret:
        popl any_register
        ret
```

advanced return-into-libc: `esp` lifting (2)

(execution)

after overflowing buffer:

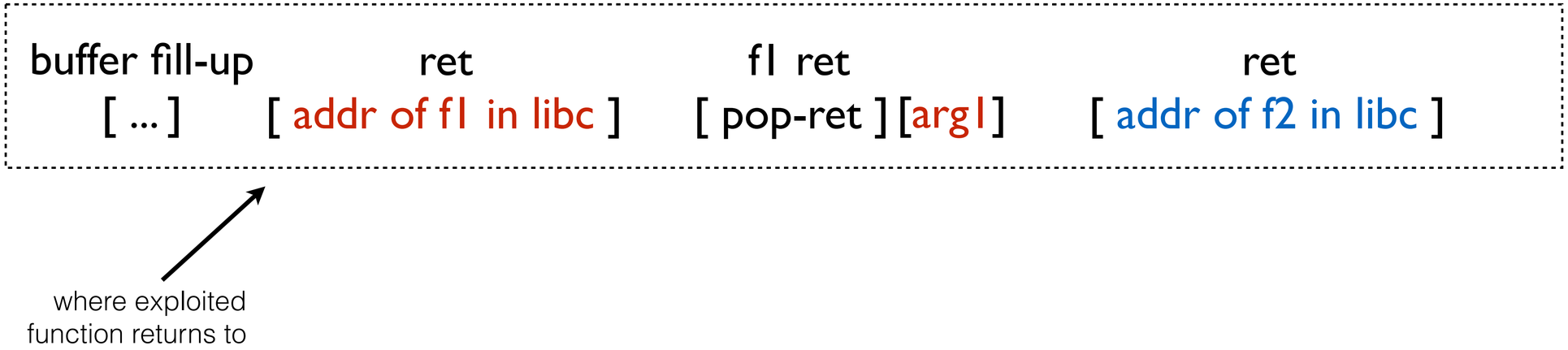



```
pop-ret:
        popl any_register
        ret
```

advanced return-into-libc: `esp` lifting (2)

(execution)

after overflowing buffer:

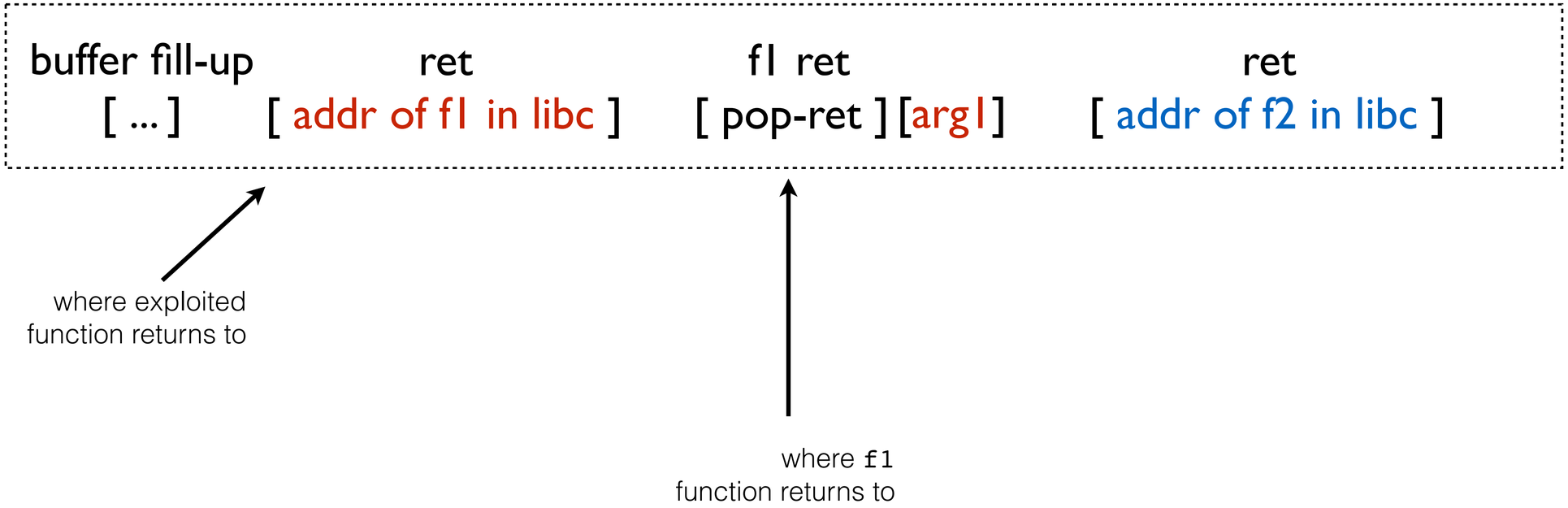



```
pop-ret:
        popl any_register
        ret
```

advanced return-into-libc: `esp` lifting (2)

(execution)

after overflowing buffer:

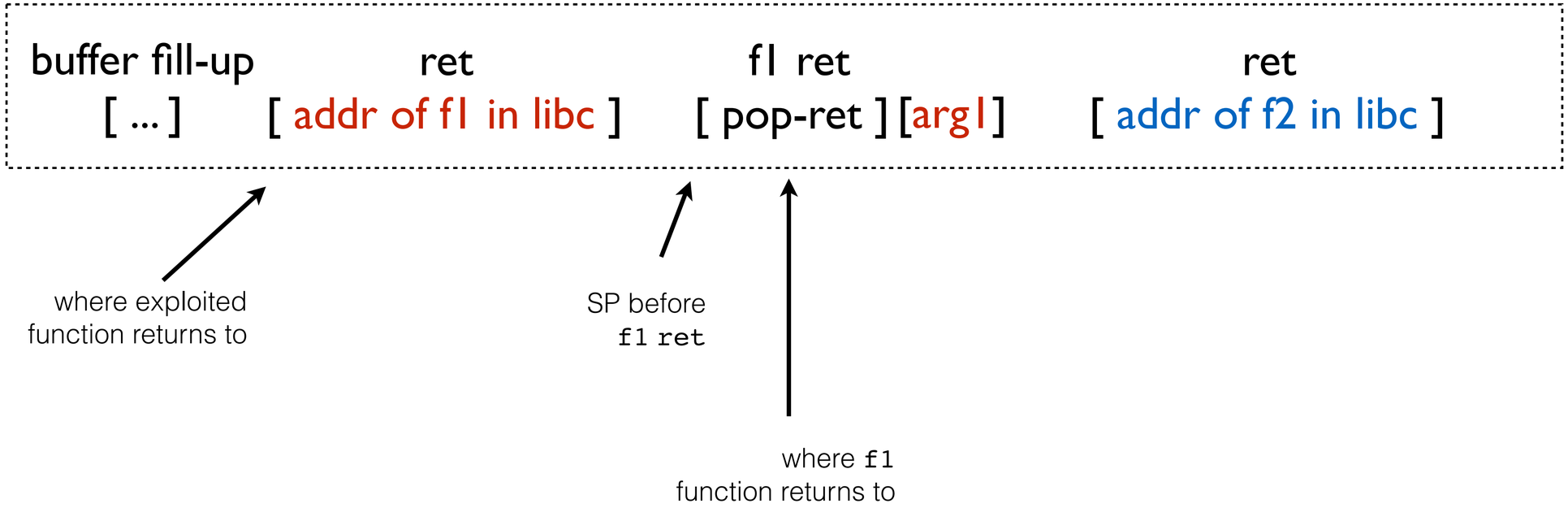



```
pop-ret:
        popl any_register
        ret
```

advanced return-into-libc: `esp` lifting (2)

(execution)

after overflowing buffer:

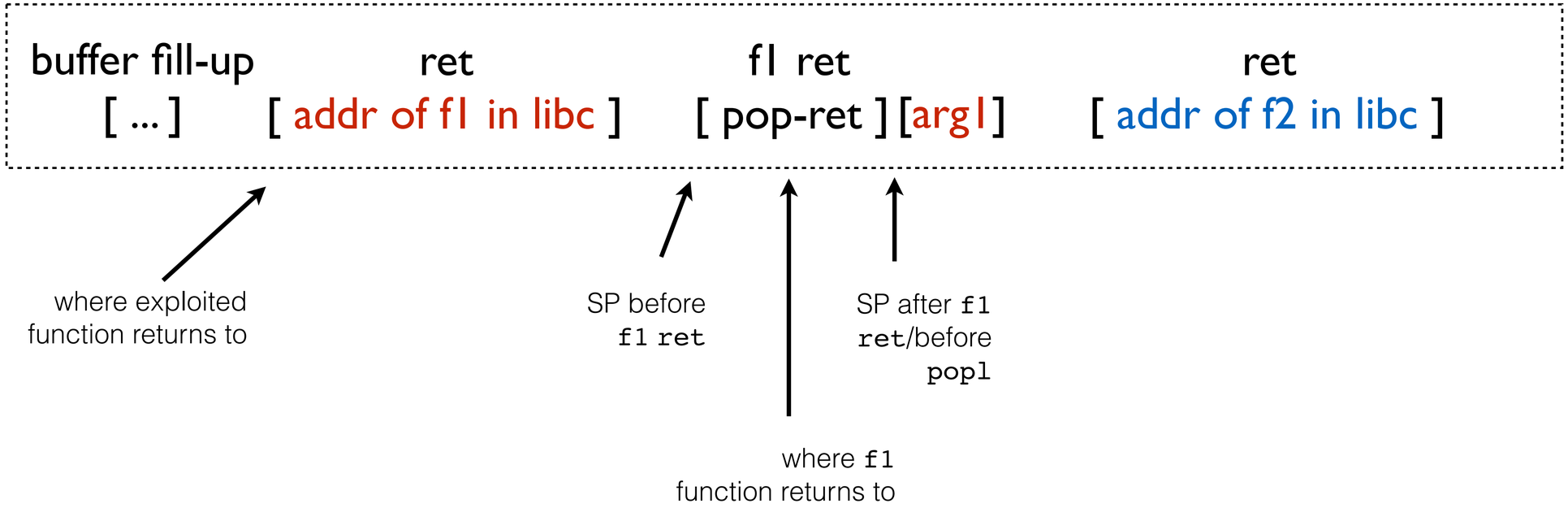



```
pop-ret:
        popl any_register
        ret
```

advanced return-into-libc: `esp` lifting (2)

(execution)

after overflowing buffer:

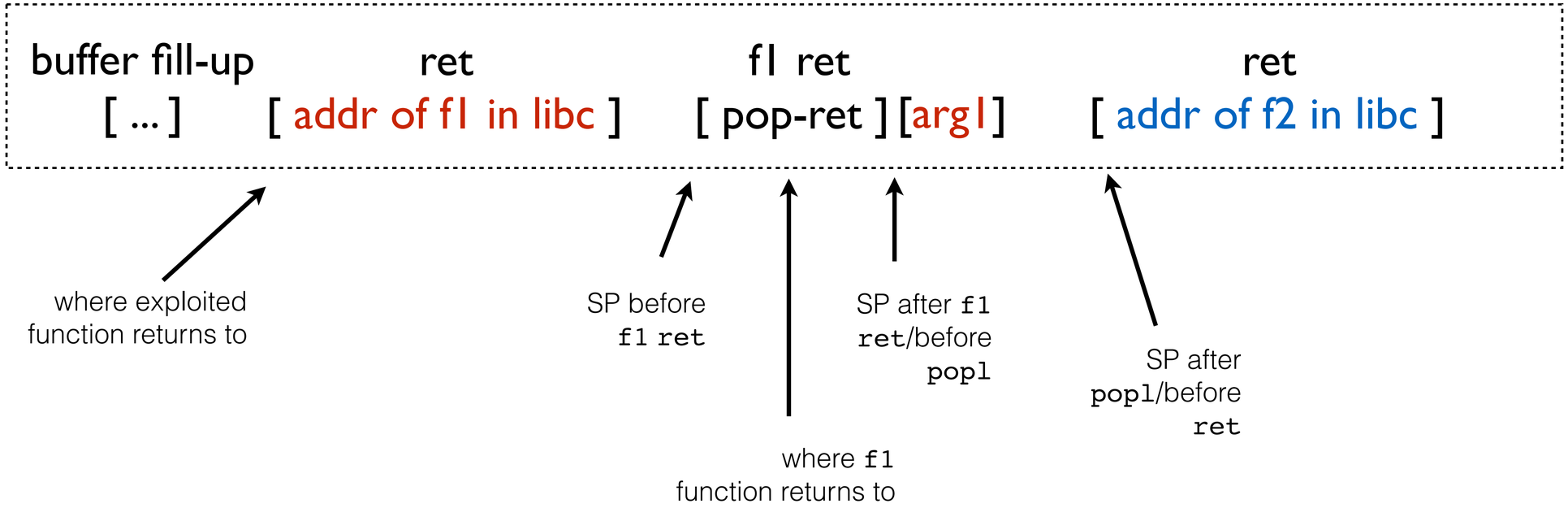



```
pop-ret:
        popl any_register
        ret
```

advanced return-into-libc: `esp` lifting (2)

(execution)

after overflowing buffer:

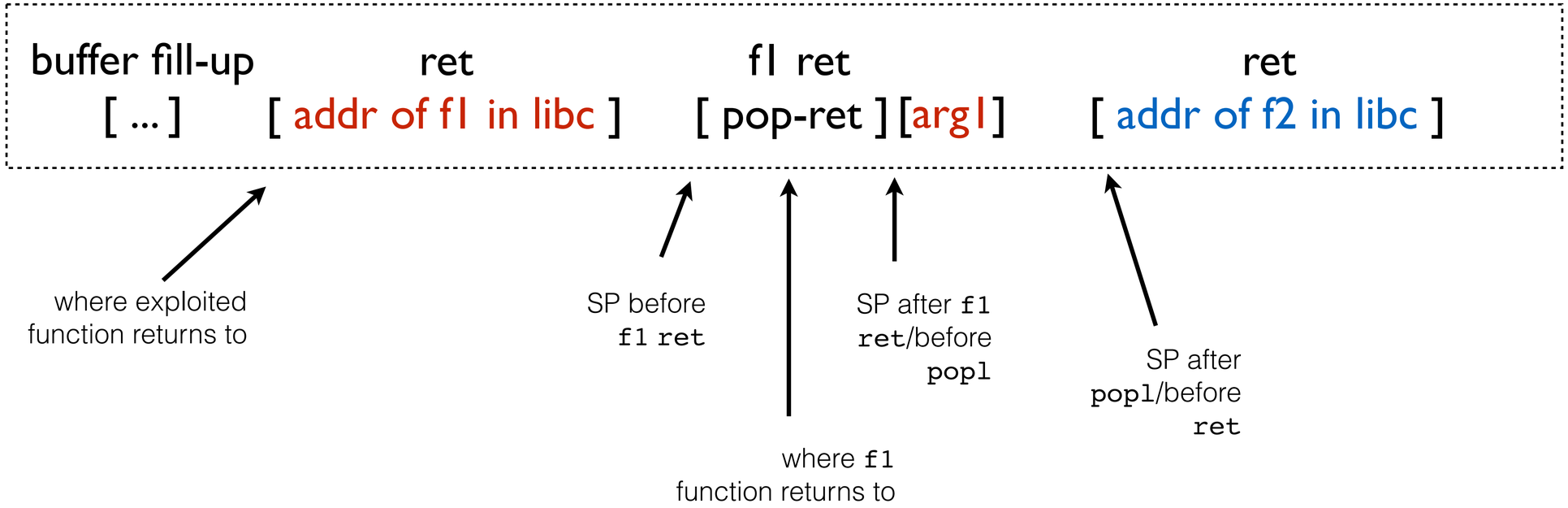



```
pop-ret:
        popl any_register
        ret
```

advanced return-into-libc: `esp` lifting (2)

(execution)

after overflowing buffer:



limitation: argument can only be four bytes

```
pop-ret:
        popl any_register
        ret
```

advanced return-into-libc: `esp` lifting (2)

(arguments more than four bytes)

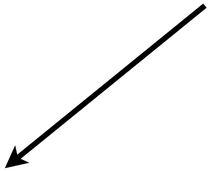
after overflowing buffer:

find sequence:

```
pop-ret:
        popl any_register
        popl any_register
        ret
```

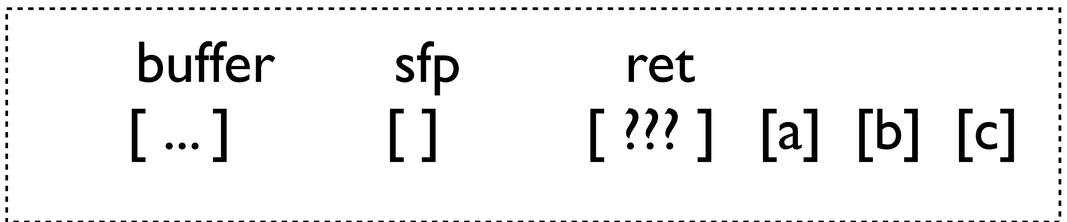
→ eight bytes for arguments

advanced return-into-libc: frame stacking

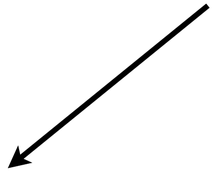


if program compiled *without*:

`-fomit-frame-pointer`



advanced return-into-libc: frame stacking

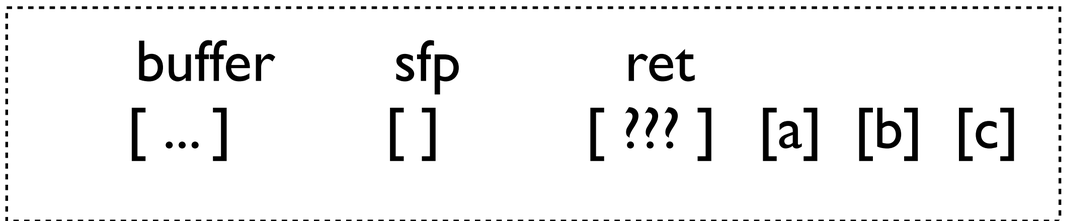


if program compiled *without*:

```
-fomit-frame-pointer
```

epilogue:

```
leaveret:
        leave
        ret
```



advanced return-into-libc: frame stacking

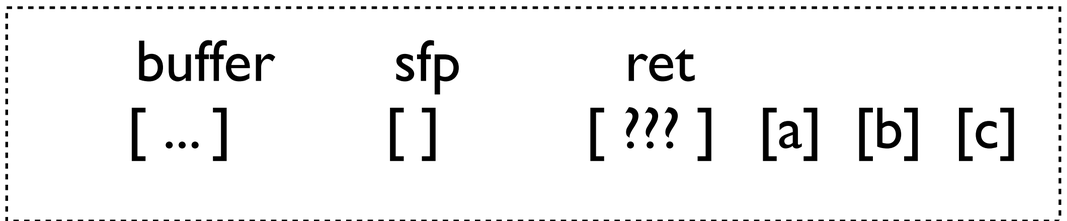
if program compiled *without*:

```
-fomit-frame-pointer
```

epilogue:

```
leaveret:
        leave
        ret
```

1. EBP -> ESP
2. then old frame pointer popped from stack onto EBP

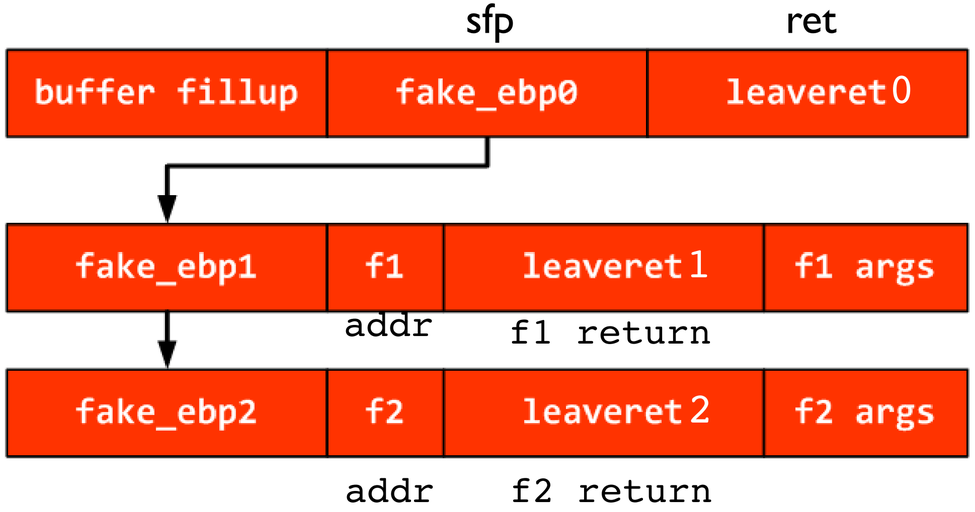


advanced return-into-libc: frame stacking

(stack after buffer overflow)

vulnerable function epilogue (function executes leaveret normally):

`fake_ebp0` into **`%ebp`** and returns into **`leaveret0`**



note:

1. exploited function executes **`leaveret`** eplg as normal (actual ebp->esp; fake_ebp0->ebp; esp at leaveret0)
2. then returns to **`leaveret`** eplg again (**`leaveret0`**) (fake_ebp0->esp; fake_ebp1->ebp; esp at f1 addr)

```
leaveret:
 1.%ebp->%esp,popl  %ebp
                    2.ret
```

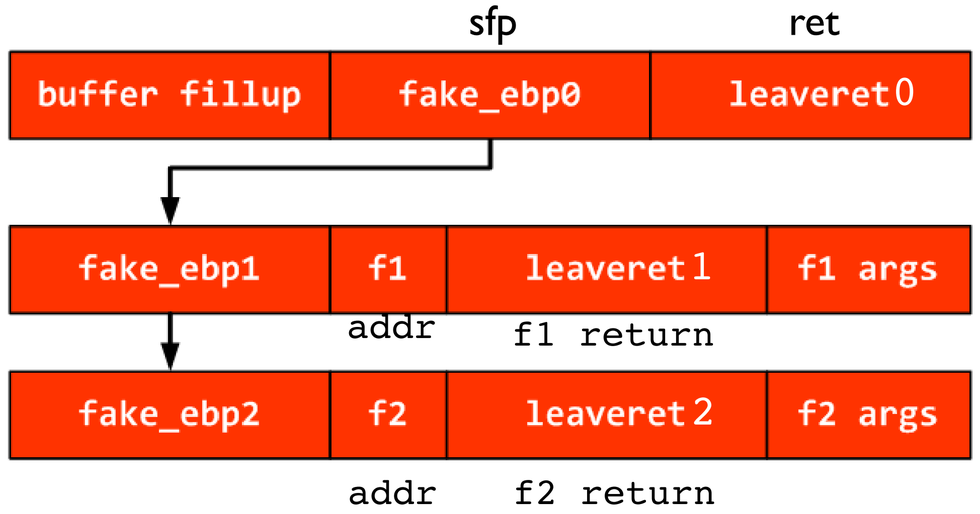
advanced return-into-libc: frame stacking

(stack after buffer overflow)

vulnerable function epilogue (function executes leaveret normally):

`fake_ebp0` into **`%ebp`** and returns into **`leaveret0`**

`leaveret0:`



note:

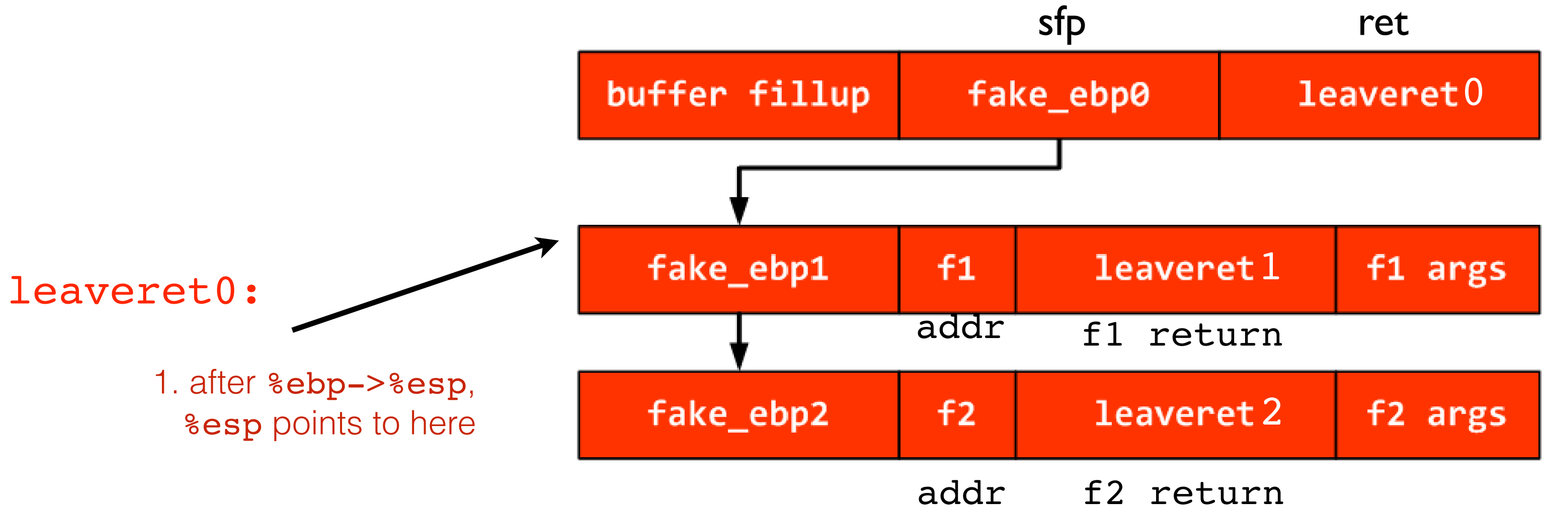
1. exploited function executes **`leaveret`** eplg as normal
   (actual ebp->esp; fake_ebp0->ebp; esp at leaveret0)
2. then returns to **`leaveret`** eplg again (**`leaveret0`**)
   (fake_ebp0->esp; fake_ebp1->ebp; esp at f1 addr)

```
leaveret:
 1.%ebp->%esp,popl  %ebp
                    2.ret
```

advanced return-into-libc: frame stacking

(stack after buffer overflow)

vulnerable function epilogue (function executes leaveret normally):
`fake_ebp0` into `%ebp` and returns into `leaveret0`



note:

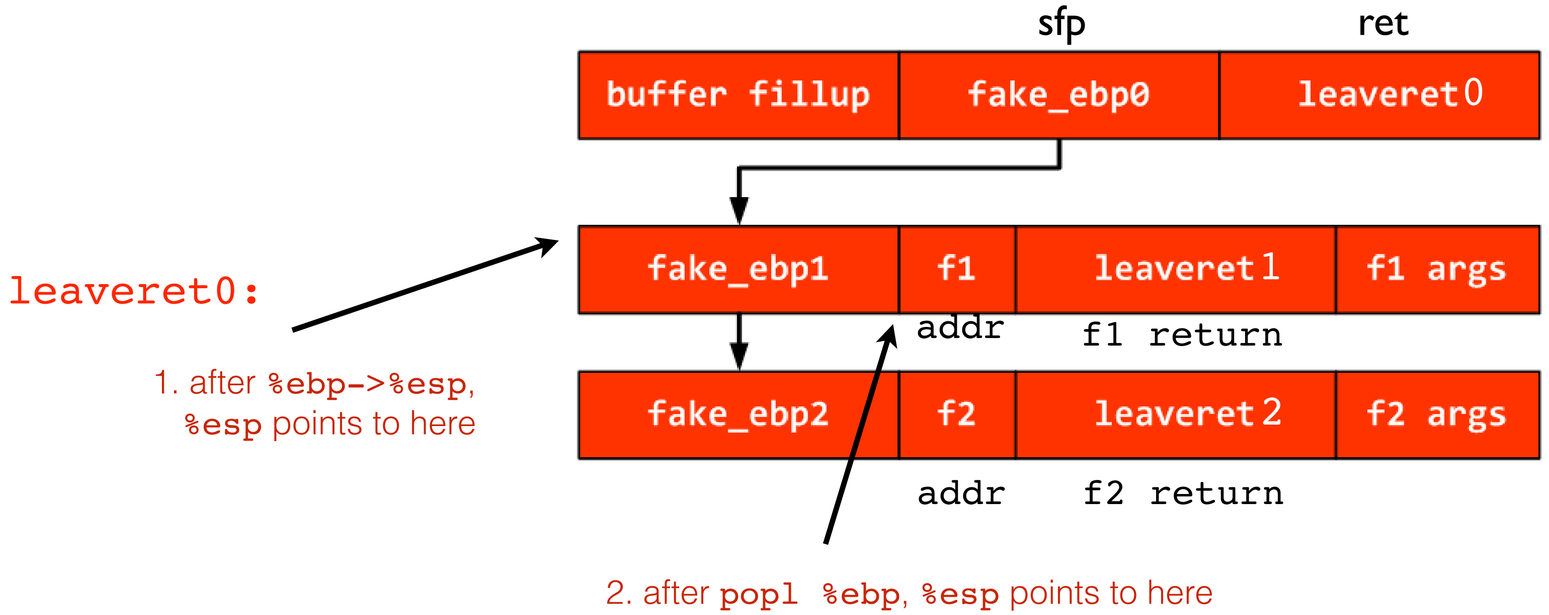
1. exploited function executes `leaveret` eplg as normal
   (actual ebp->esp; fake_ebp0->ebp; esp at leaveret0)
2. then returns to `leaveret` eplg again (`leaveret0`)
   (fake_ebp0->esp; fake_ebp1->ebp; esp at f1 addr)

```
leaveret:
 1.%ebp->%esp,popl  %ebp
                2.ret
```

advanced return-into-libc: frame stacking

(stack after buffer overflow)

vulnerable function epilogue (function executes leaveret normally):
`fake_ebp0` into `%ebp` and returns into `leaveret0`



note:

1. exploited function executes **`leaveret`** eplg as normal (actual ebp->esp; fake_ebp0->ebp; esp at leaveret0)
2. then returns to **`leaveret`** eplg again (**`leaveret0`**) (fake_ebp0->esp; fake_ebp1->ebp; esp at f1 addr)

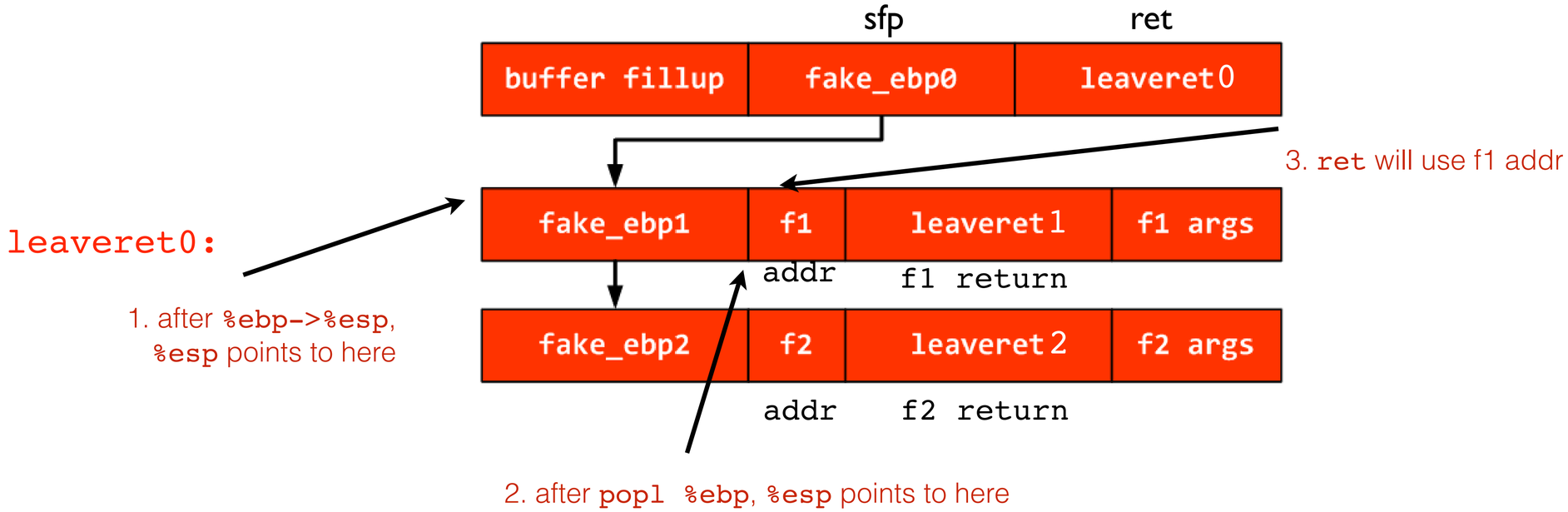
```
leaveret:
 1.%ebp->%esp,popl  %ebp
                2.ret
```

advanced return-into-libc: frame stacking

(stack after buffer overflow)

vulnerable function epilogue (function executes leaveret normally):

`fake_ebp0` into `%ebp` and returns into `leaveret0`



note:

1. exploited function executes `leaveret` eplg as normal (actual ebp->esp; fake_ebp0->ebp; esp at leaveret0)
2. then returns to `leaveret` eplg again (`leaveret0`) (fake_ebp0->esp; fake_ebp1->ebp; esp at f1 addr)

```
leaveret:
 1.%ebp->%esp,popl %ebp
                2.ret
```

advanced return-into-libc: frame stacking

(execution; remember: exploited function uses `leaveret` as usual)

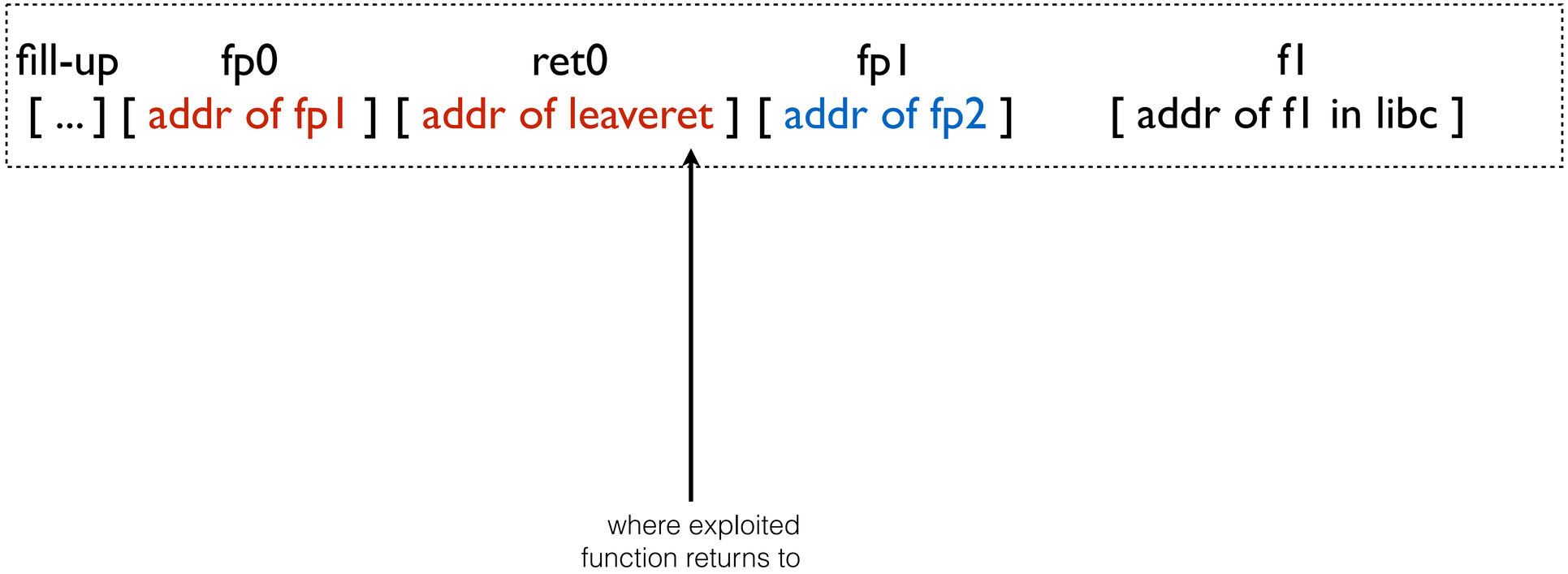
after overflowing buffer:


```
leaveret:
 1.%ebp->%esp,popl %ebp
                2.ret
```

advanced return-into-libc: frame stacking

(execution; remember: exploited function uses `leaveret` as usual)

after overflowing buffer:

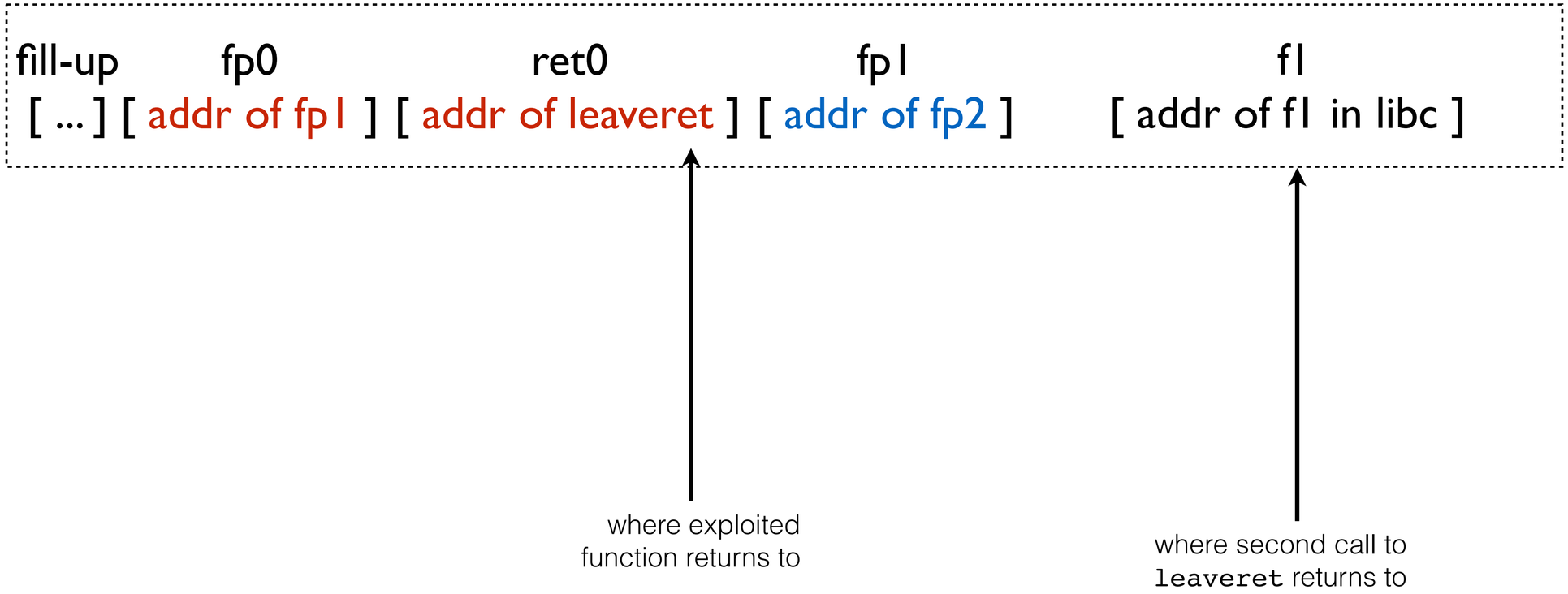



```
leaveret:
 1.%ebp->%esp,popl %ebp
                2.ret
```

advanced return-into-libc: frame stacking

(execution; remember: exploited function uses `leaveret` as usual)

after overflowing buffer:

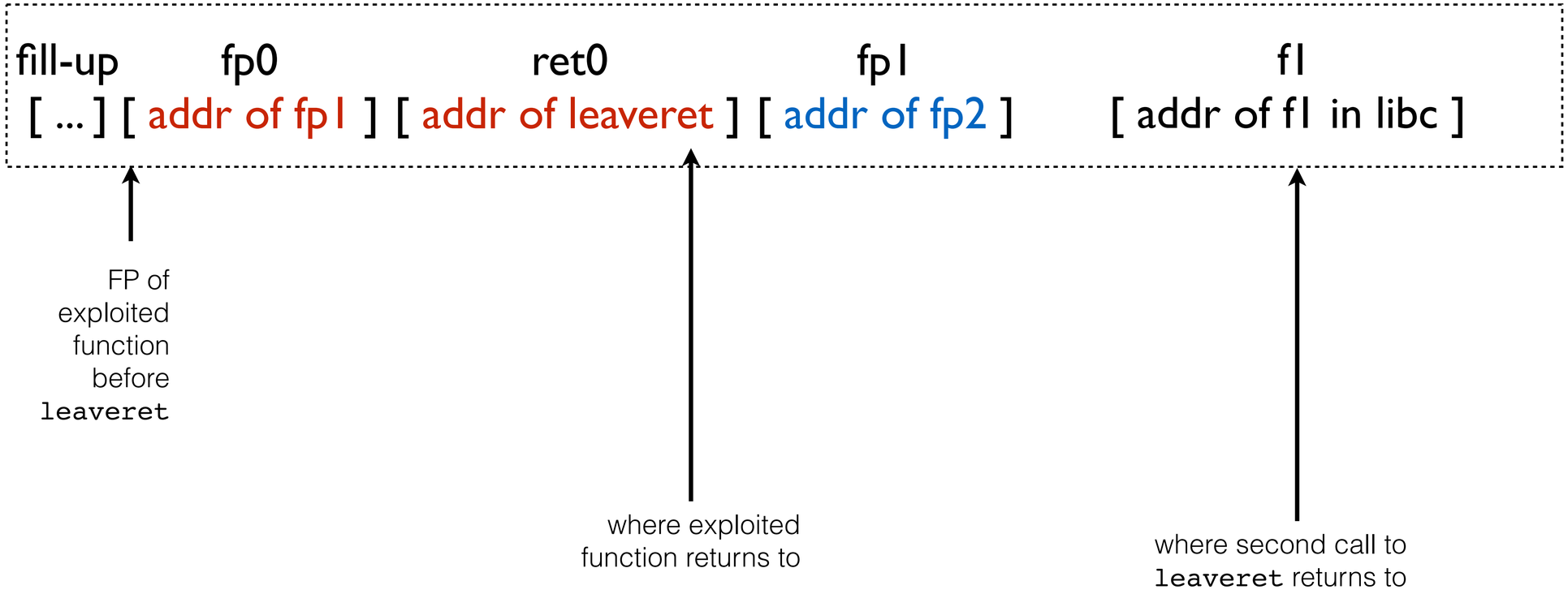



```
leaveret:
 1.%ebp->%esp,popl %ebp
                2.ret
```

advanced return-into-libc: frame stacking

(execution; remember: exploited function uses `leaveret` as usual)

after overflowing buffer:

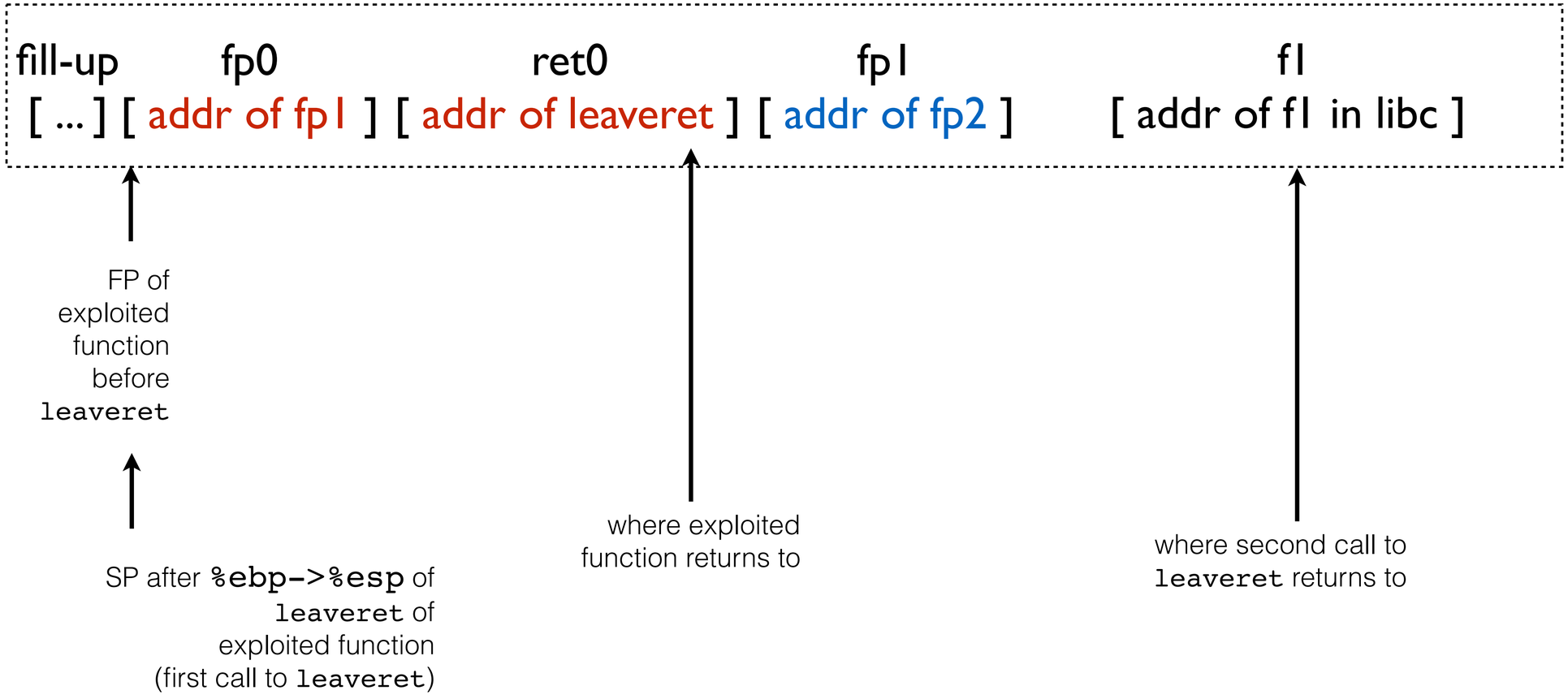



```
leaveret:
 1.%ebp->%esp,popl %ebp
                  2.ret
```

advanced return-into-libc: frame stacking

(execution; remember: exploited function uses `leaveret` as usual)

after overflowing buffer:

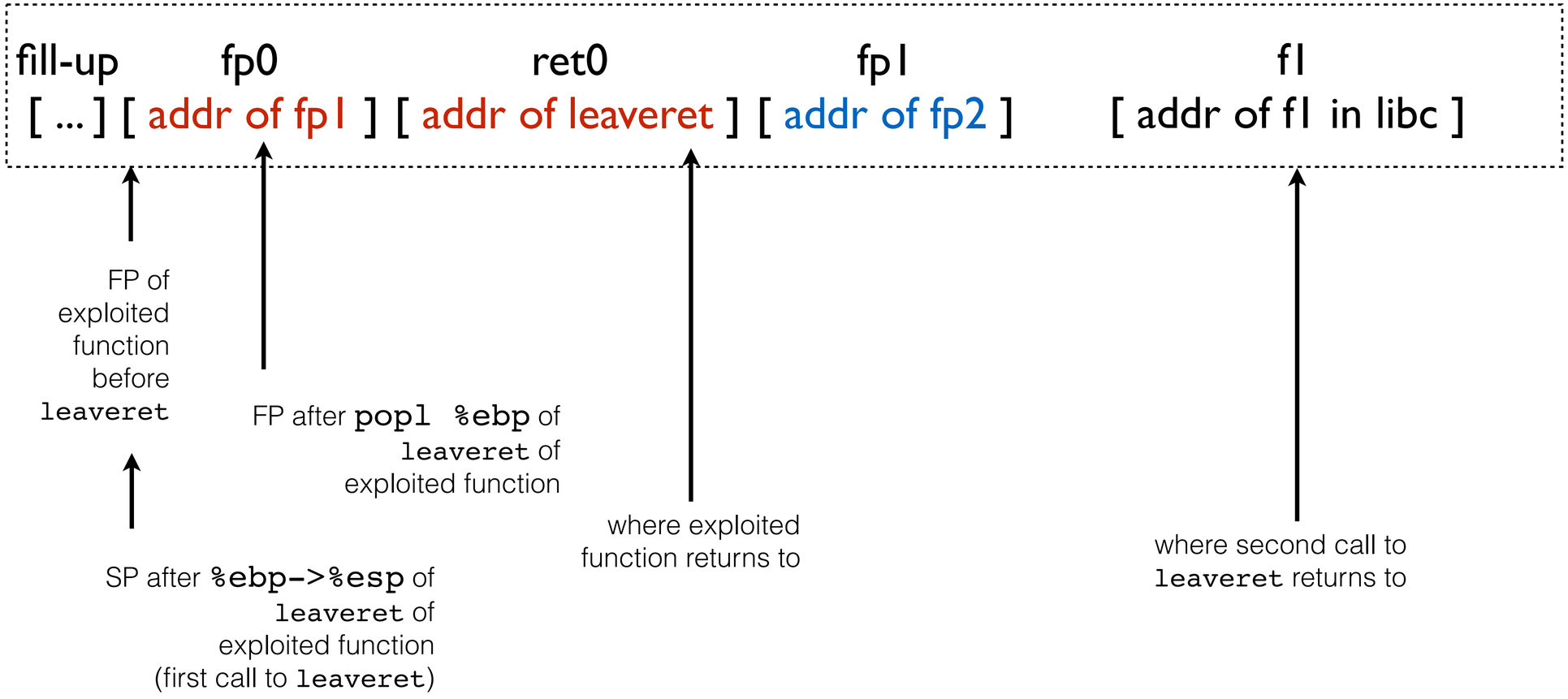



```
leaveret:
 1.%ebp->%esp,popl %ebp
                2.ret
```

advanced return-into-libc: frame stacking

(execution; remember: exploited function uses `leaveret` as usual)

after overflowing buffer:

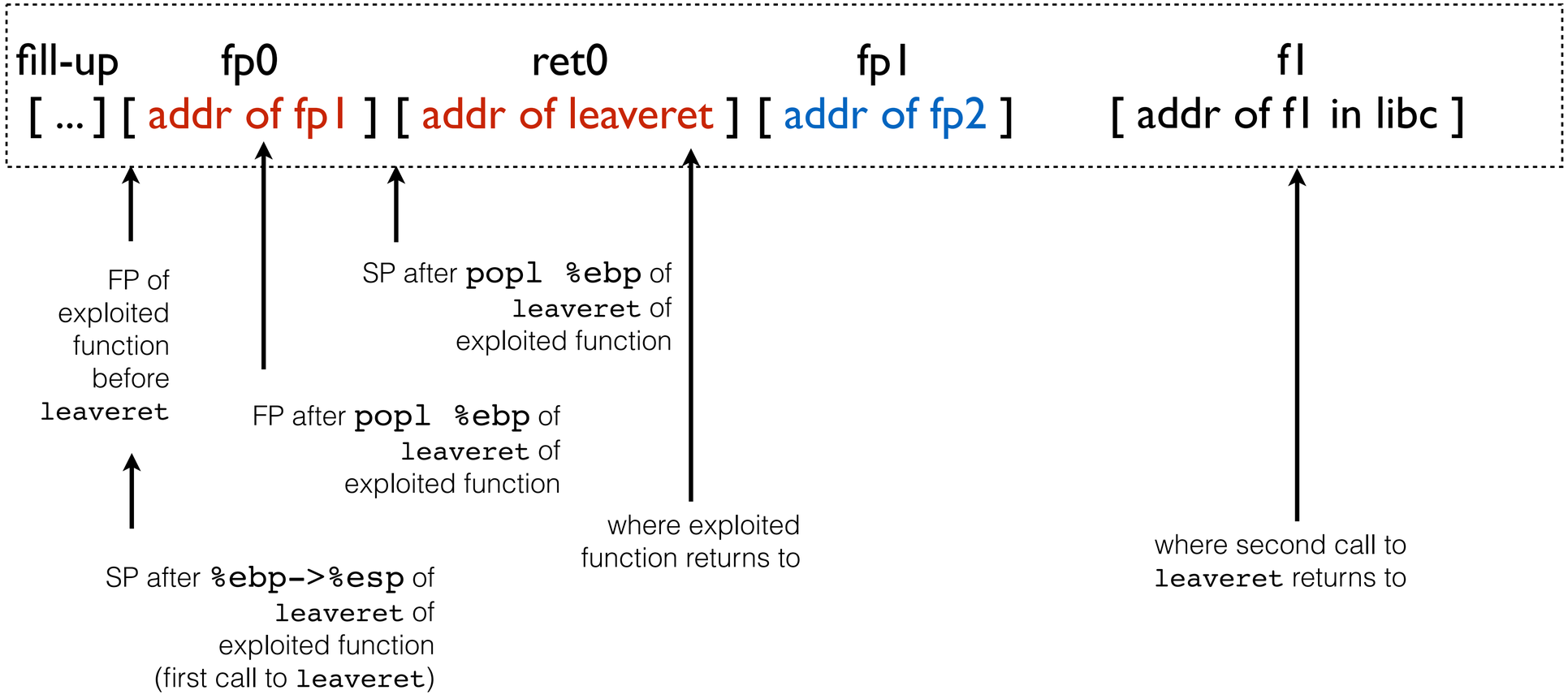



```
leaveret:
 1.%ebp->%esp,popl %ebp
                2.ret
```

advanced return-into-libc: frame stacking

(execution; remember: exploited function uses `leaveret` as usual)

after overflowing buffer:

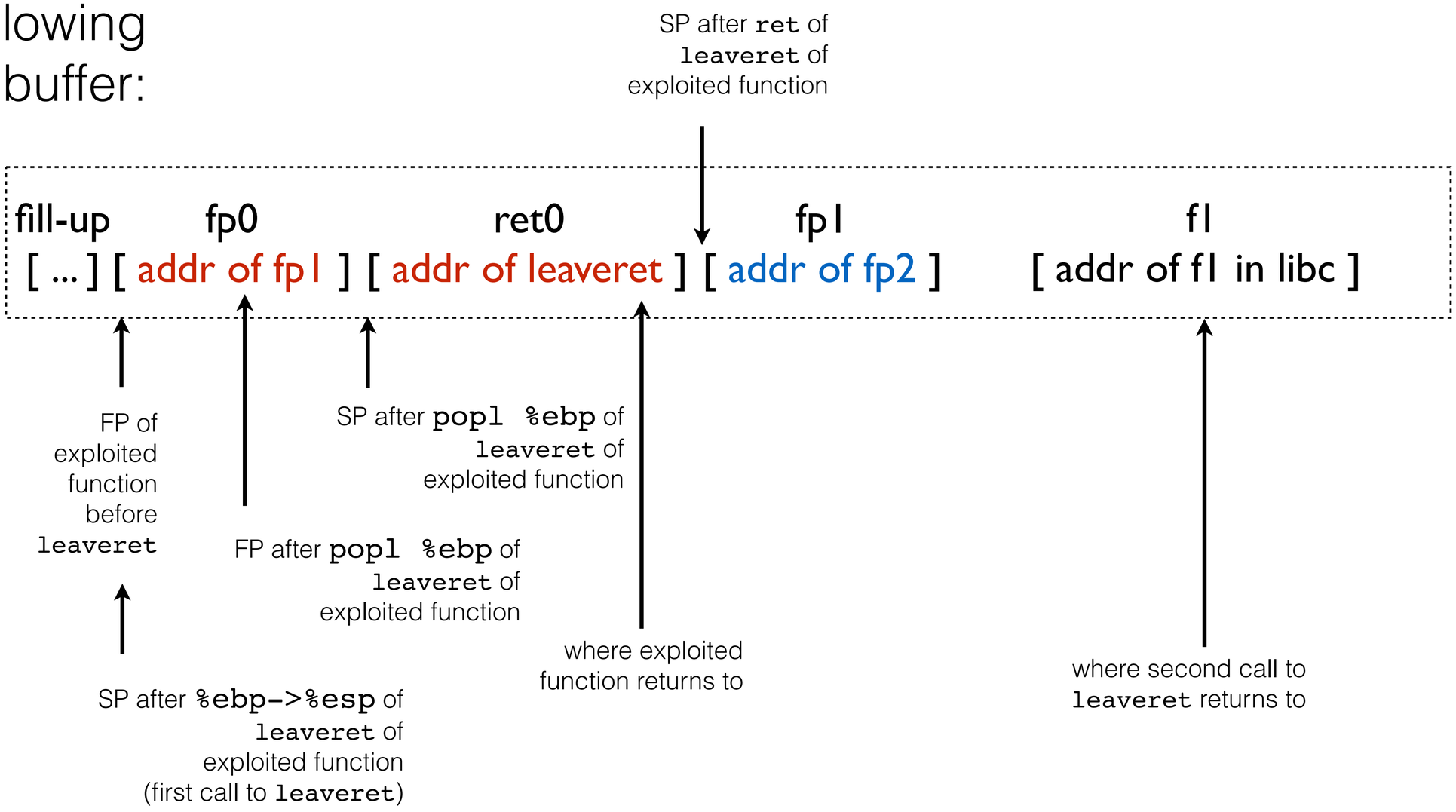



```
leaveret:
 1.%ebp->%esp,popl %ebp
               2.ret
```

advanced return-into-libc: frame stacking

(execution; remember: exploited function uses `leaveret` as usual)

after overflowing buffer:

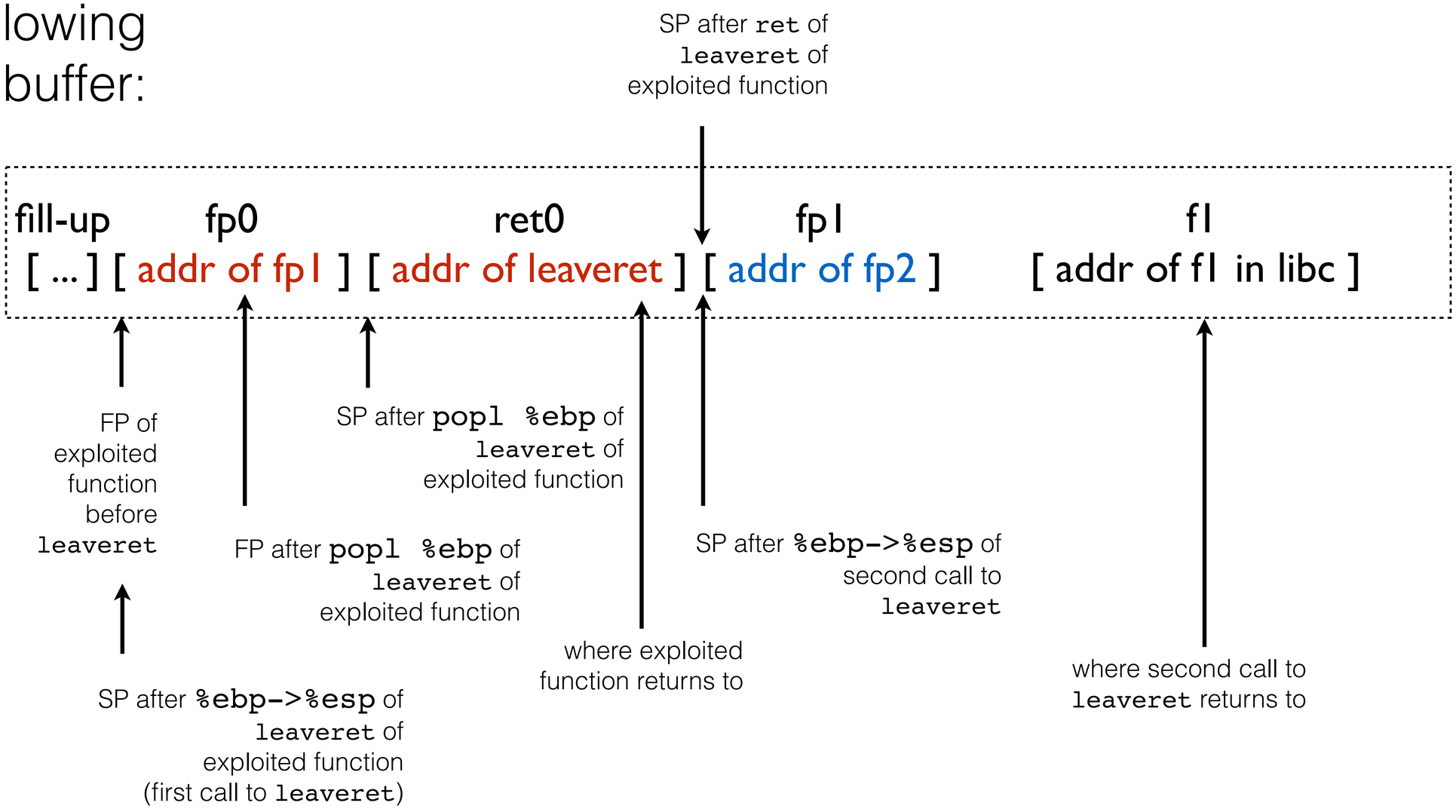



```
leaveret:
 1.%ebp->%esp,popl %ebp
               2.ret
```

advanced return-into-libc: frame stacking

(execution; remember: exploited function uses `leaveret` as usual)

after overflowing buffer:

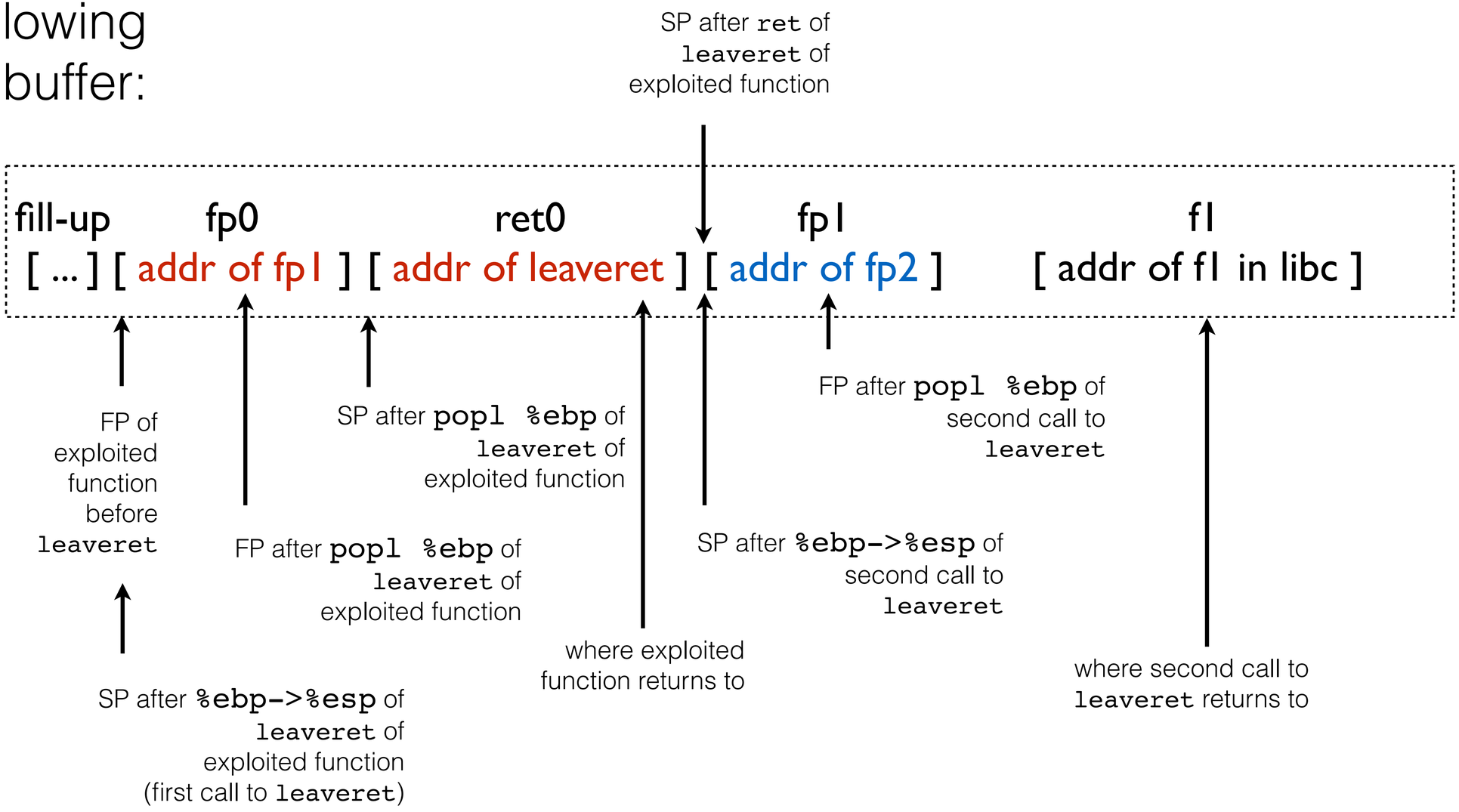



```
leaveret:
 1.%ebp->%esp,popl %ebp
               2.ret
```

advanced return-into-libc: frame stacking

(execution; remember: exploited function uses `leaveret` as usual)

after overflowing buffer:

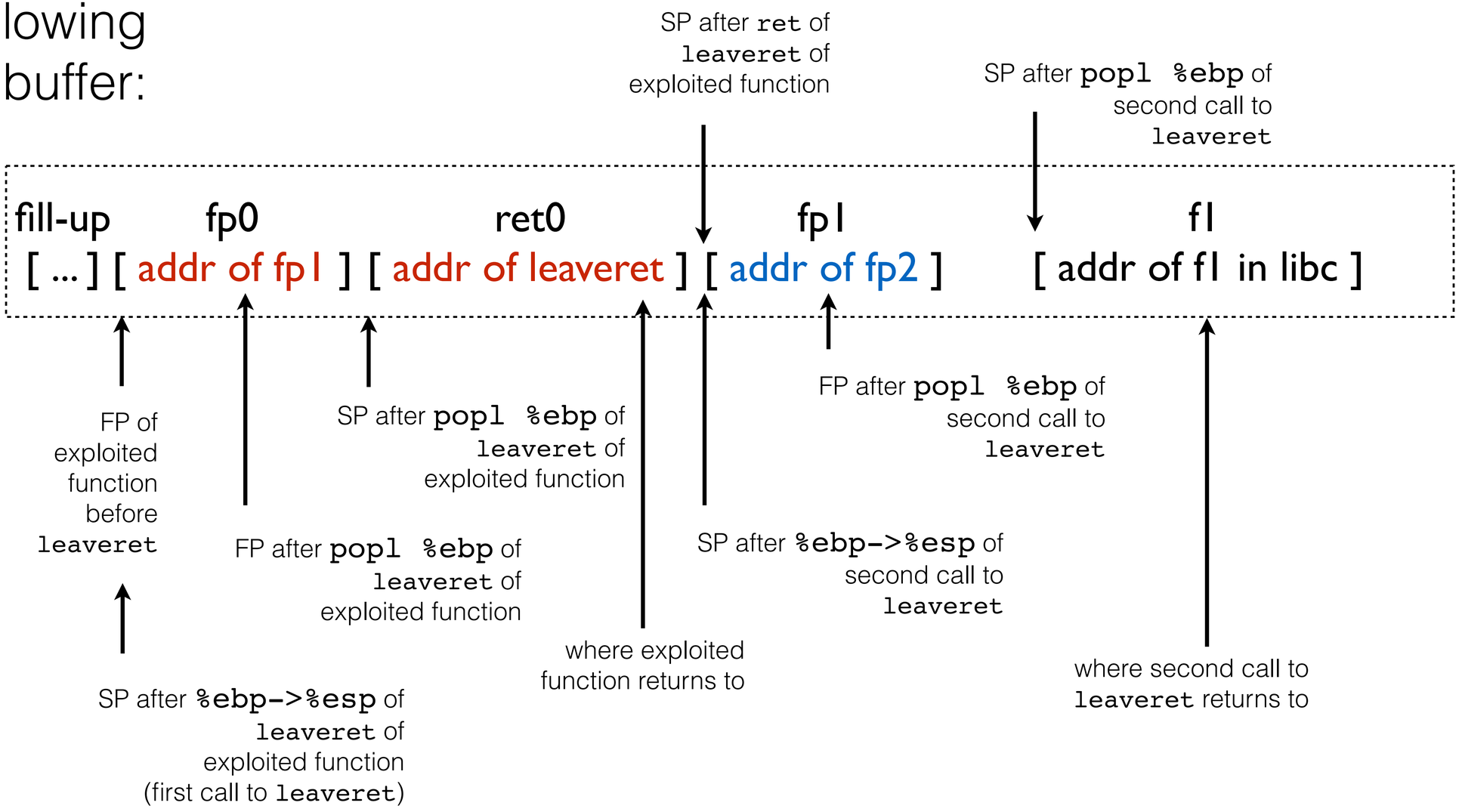



```
leaveret:
 1.%ebp->%esp,popl %ebp
               2.ret
```

advanced return-into-libc: frame stacking

(execution; remember: exploited function uses `leaveret` as usual)

after overflowing buffer:




```
leaveret:
 1.%ebp->%esp,popl %ebp
               2.ret
```