CSE251: System Programming

4. Assembly (2)

Seongil Wi

Department of Computer Science and Engineering

X86-64 Instructions Summary

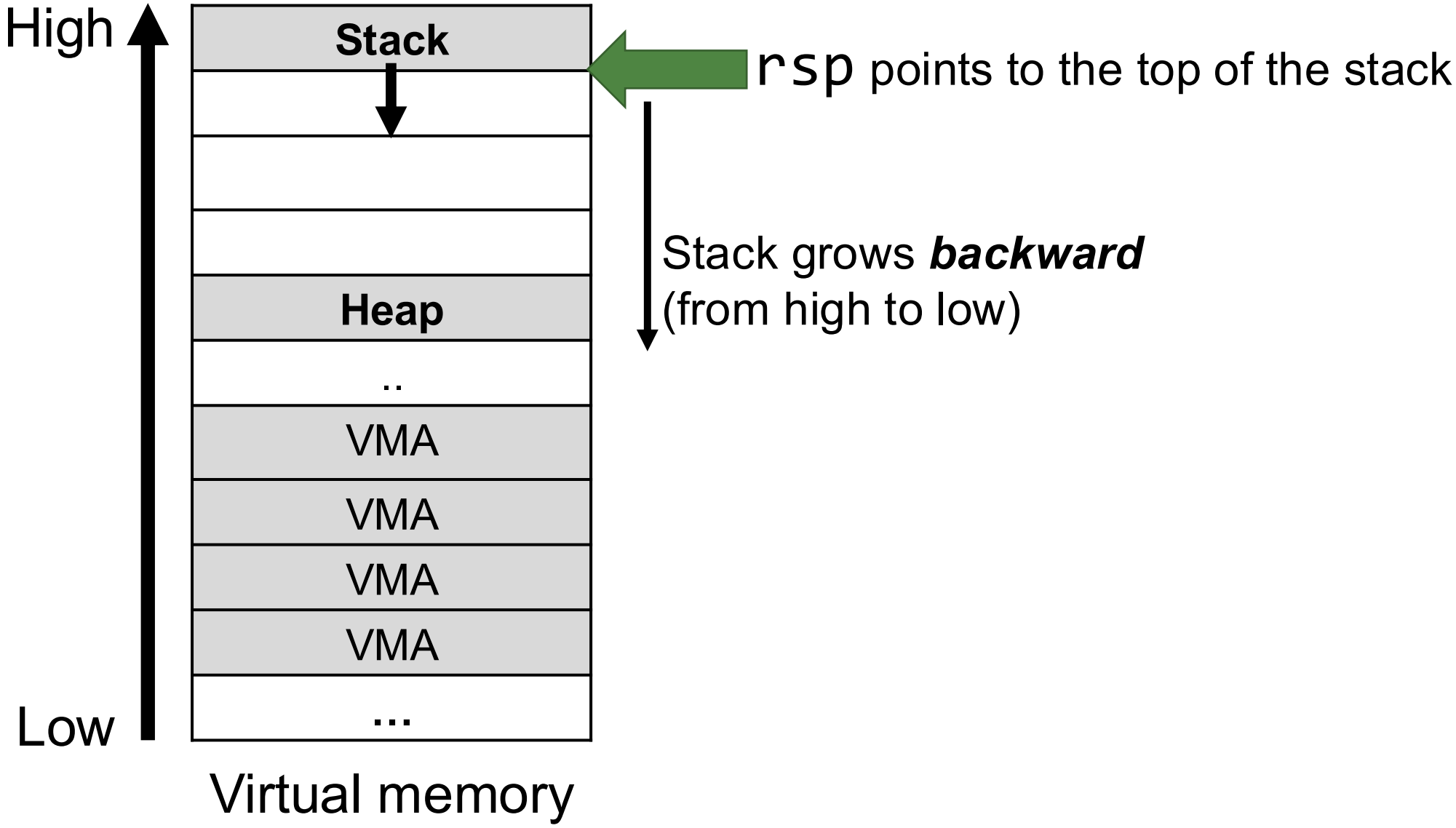
- We learned how to move around data
  - `mov, lea, push, pop, etc.`
- We learned how to perform arithmetic and logical operations
  - `add, sub, and, or, etc.`
- We also learned how to control program flows
  - `cmp, jmp, ja, jz, etc.`

Stack Operations

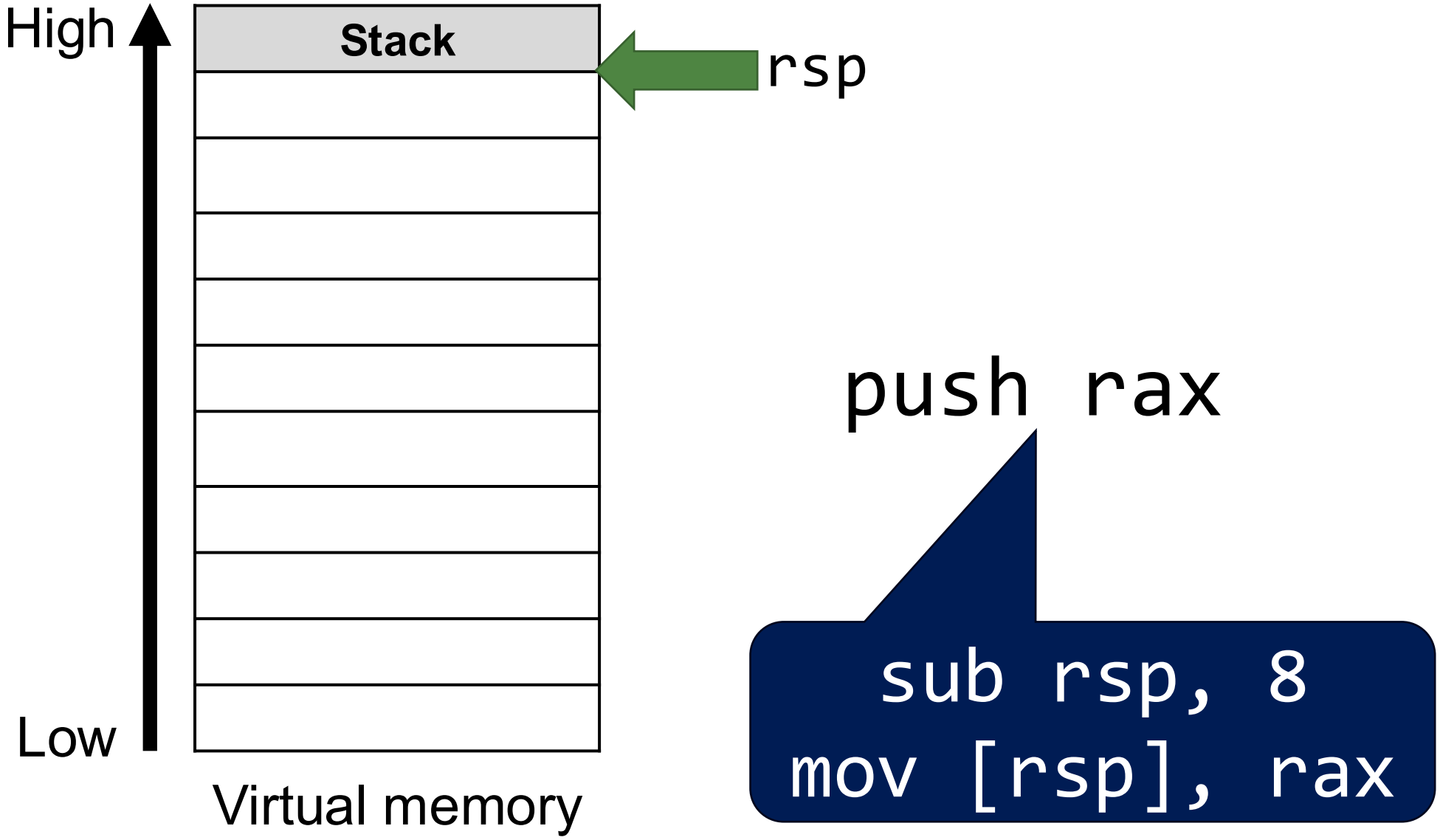
Stack Memory

- Stack stores data in a LIFO (Last-In-First-Out) fashion. When a function is invoked, a new ***stack frame*** is allocated at the top of the stack memory

Stack Operations



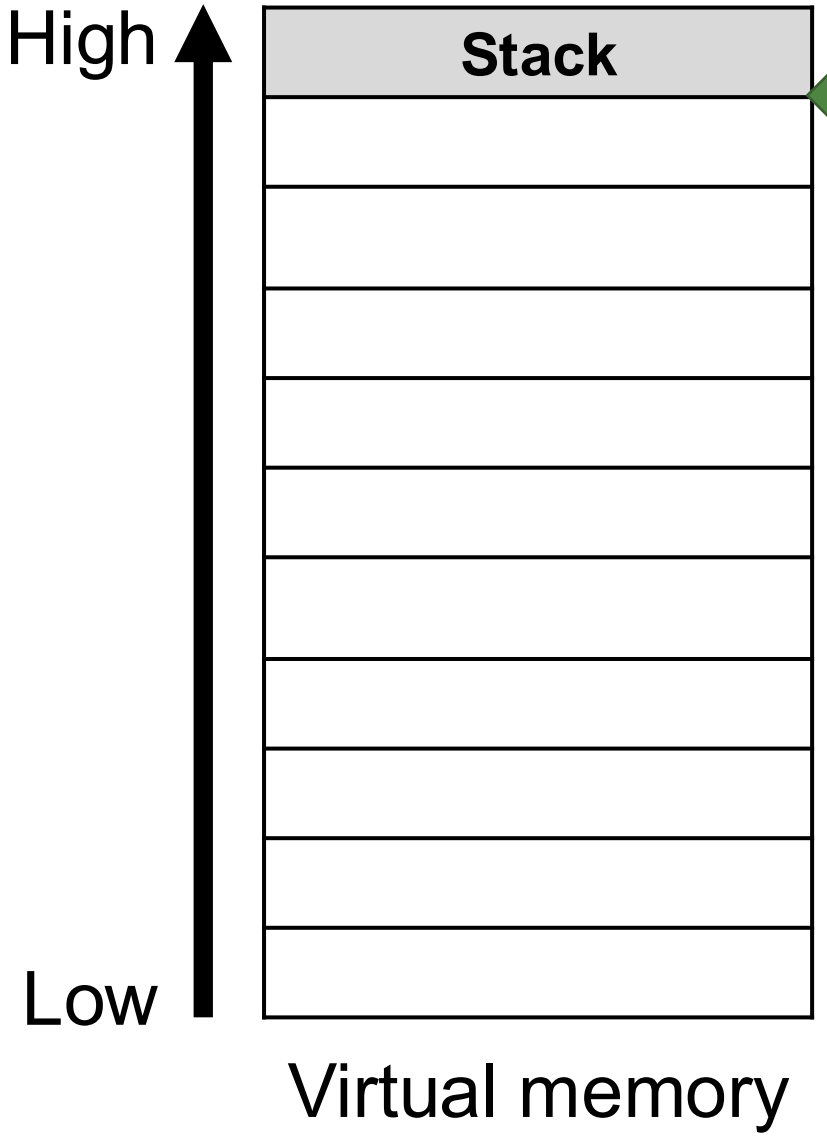
Stack Operations – Push and Pop




```
push rax
```

```
sub rsp, 8
mov [rsp], rax
```

Stack Operations – Push and Pop



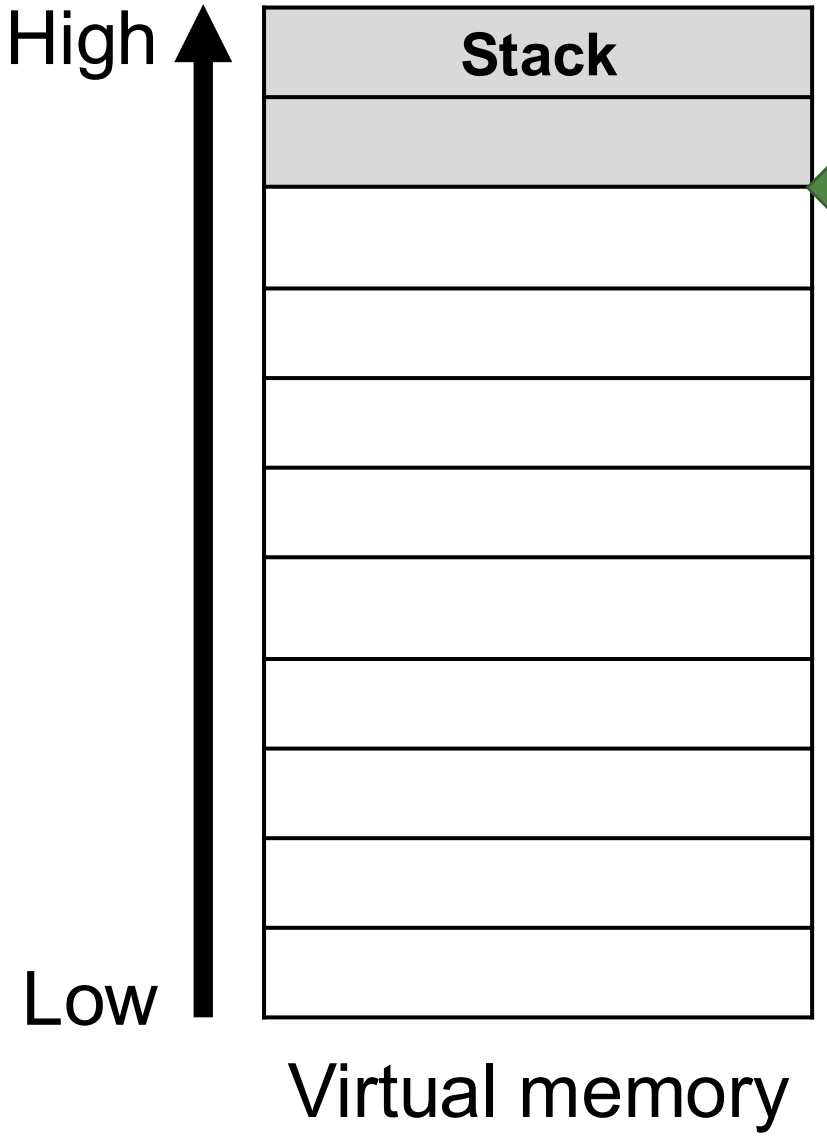
Registers

rax 0x9

```
push rax
```

```
sub rsp, 8
mov [rsp], rax
```

Stack Operations – Push and Pop



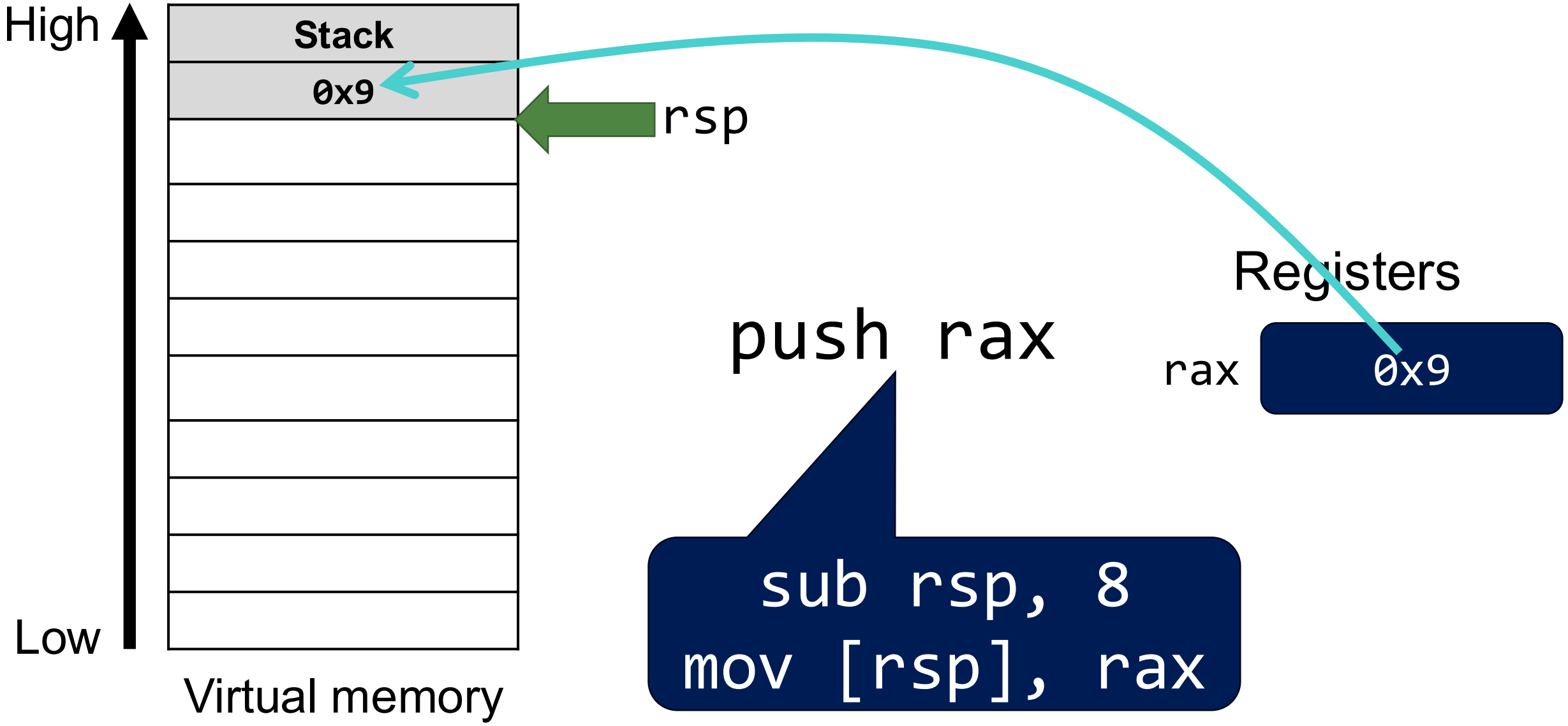

```
push rax
```

```
sub rsp, 8
mov [rsp], rax
```

Registers

rax 0x9

Stack Operations – Push and Pop



Stack Operations (push)

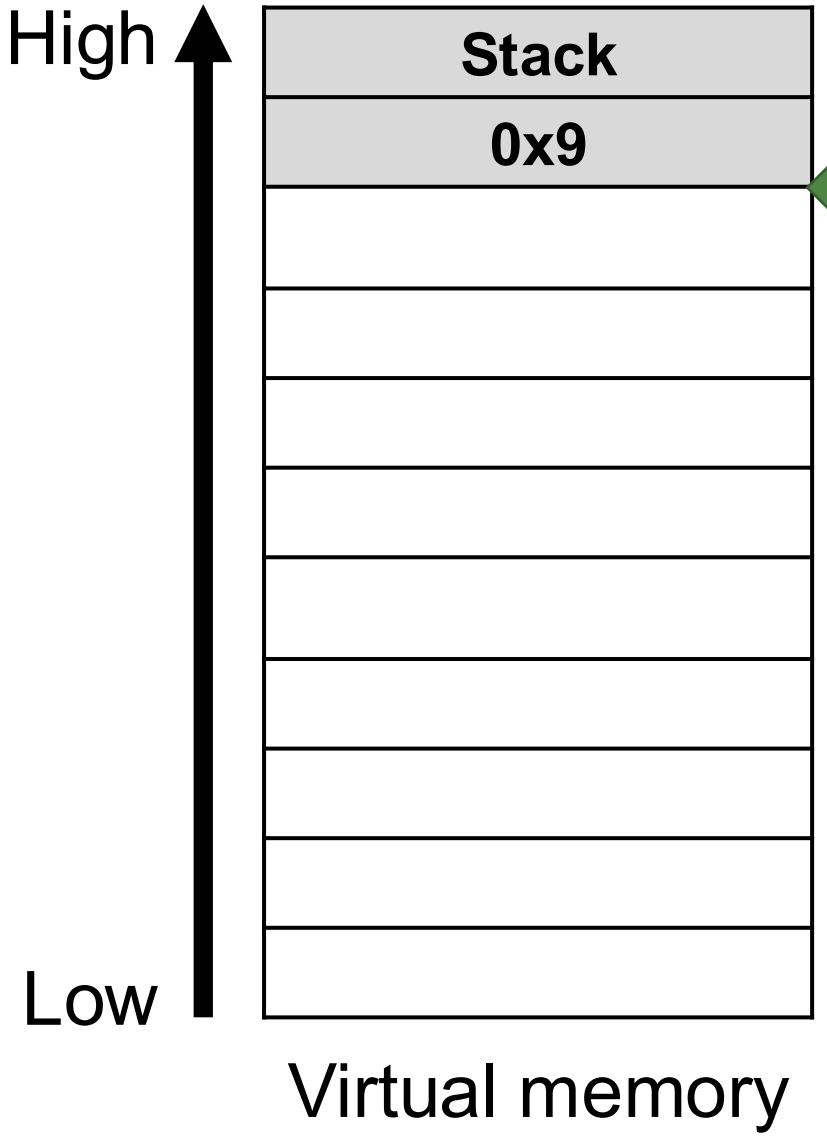
`push rax`	Push register on the stack
`push 0x42`	Push constant on the stack
`push [rax]`	Push a value at the memory address on the stack

```
push x
```

=

```
sub rsp, 8
mov [rsp], x
```

Stack Operations – Push and Pop



pop rcx

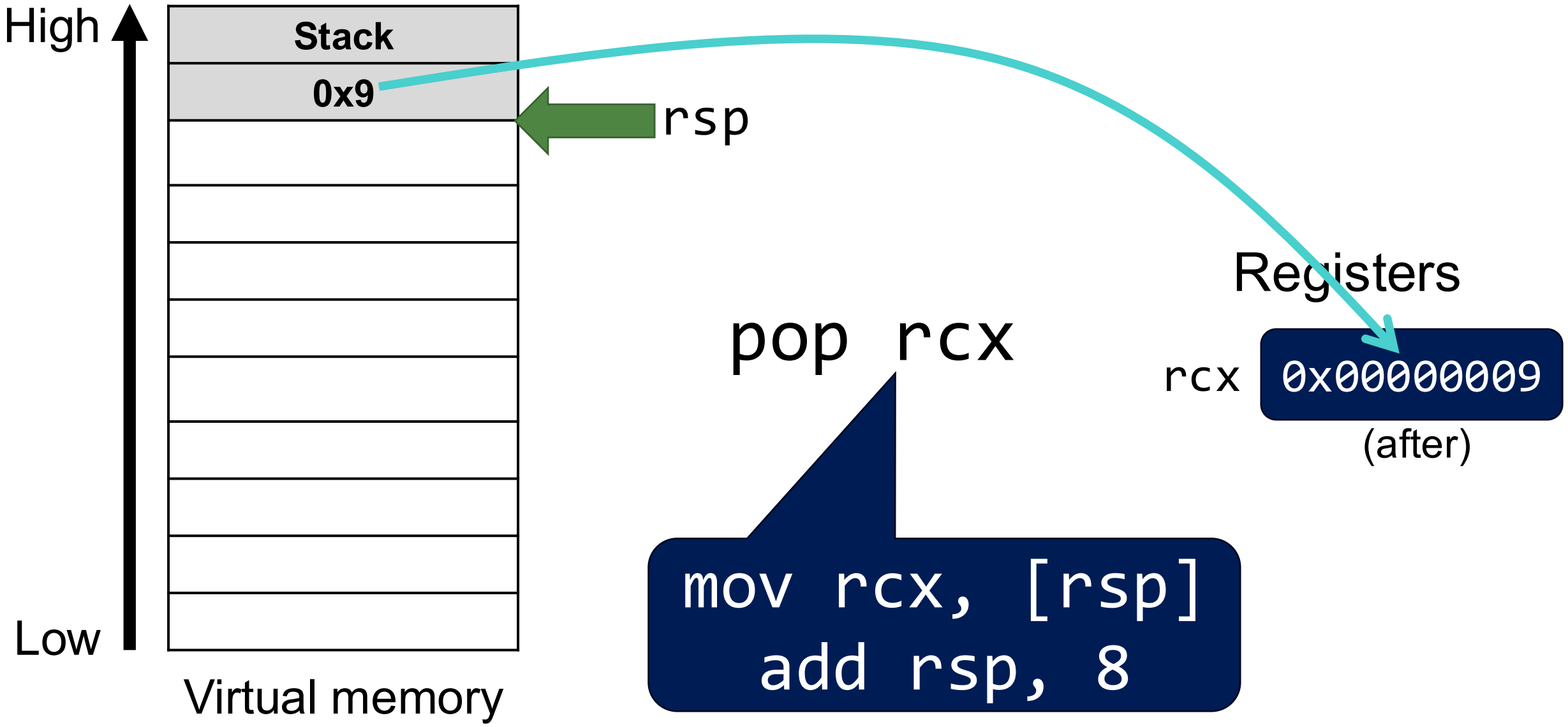
```
mov rcx, [rsp]
add rsp, 8
```

Registers

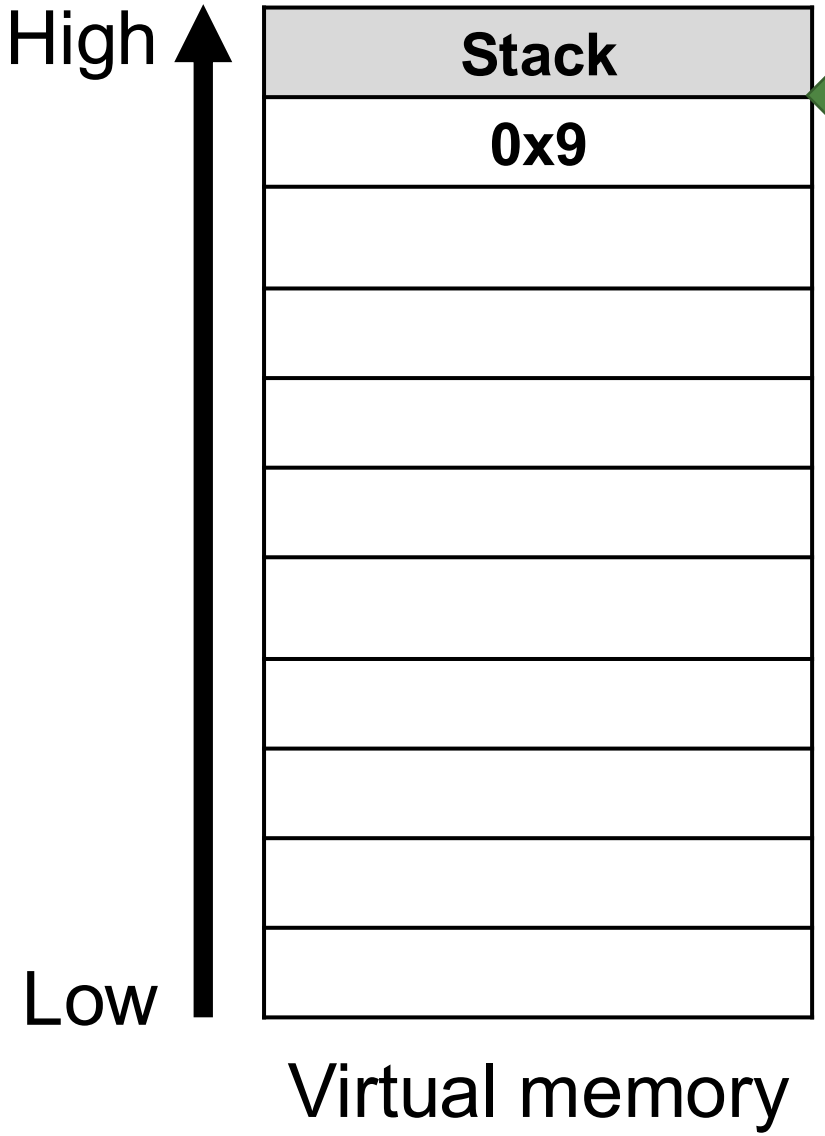
rcx 0x13ef0ca1

(before)

Stack Operations – Push and Pop



Stack Operations – Push and Pop



pop rcx

```
mov rcx, [rsp]
add rsp, 8
```

Registers

rcx 0x00000009

(after)

Stack Operations (pop)

`pop rax`	Pop the top element of the stack into register
`pop [rax]`	Pop the top element of the stack into the memory address

```
pop x
```

=

```
mov x, [rsp]
add rsp, 8
```

Stack Push/Pop (Summary)

- (On x86): `push x = sub esp, 4; mov [esp], x`
- (On x64): `push x = sub rsp, 8; mov [rsp], x`

- (On x86): `pop x = mov x, [esp]; add esp, 4`
- (On x64): `pop x = mov x, [rsp]; add rsp, 8`

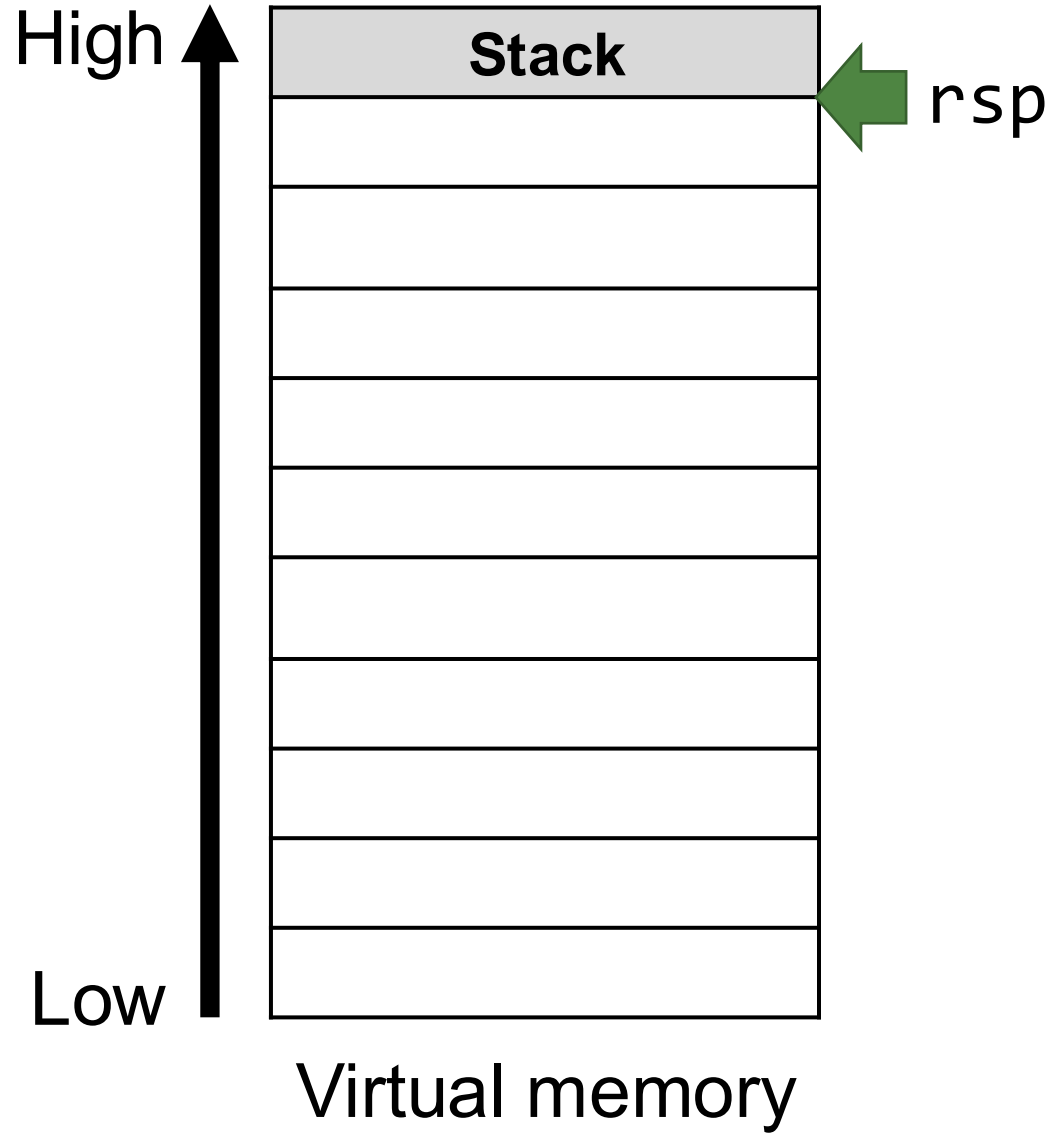
Stack Operations (leave)


```
leave = mov rsp, rbp
        pop rbp
```

Function Call (call)

```
    …
    call foo
nextret:
    nop
    nop
    …
foo:
    nop
    nop
```

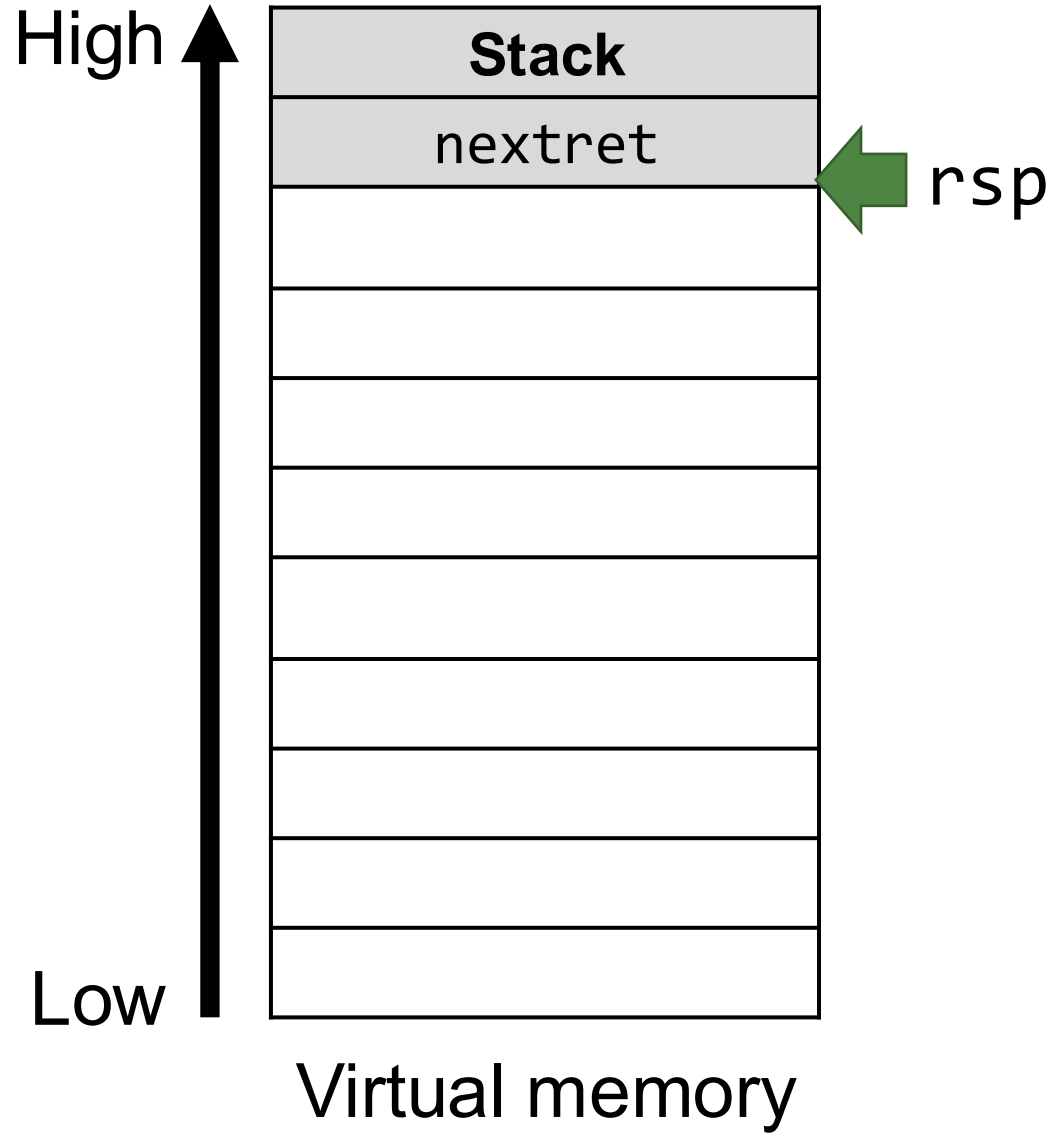
push nextret
jmp foo



Function Call (call)


```
      …
      call foo
nextret:
      nop
      nop
      …
foo:
      nop
      nop
```

```
push nextret
jmp foo
```

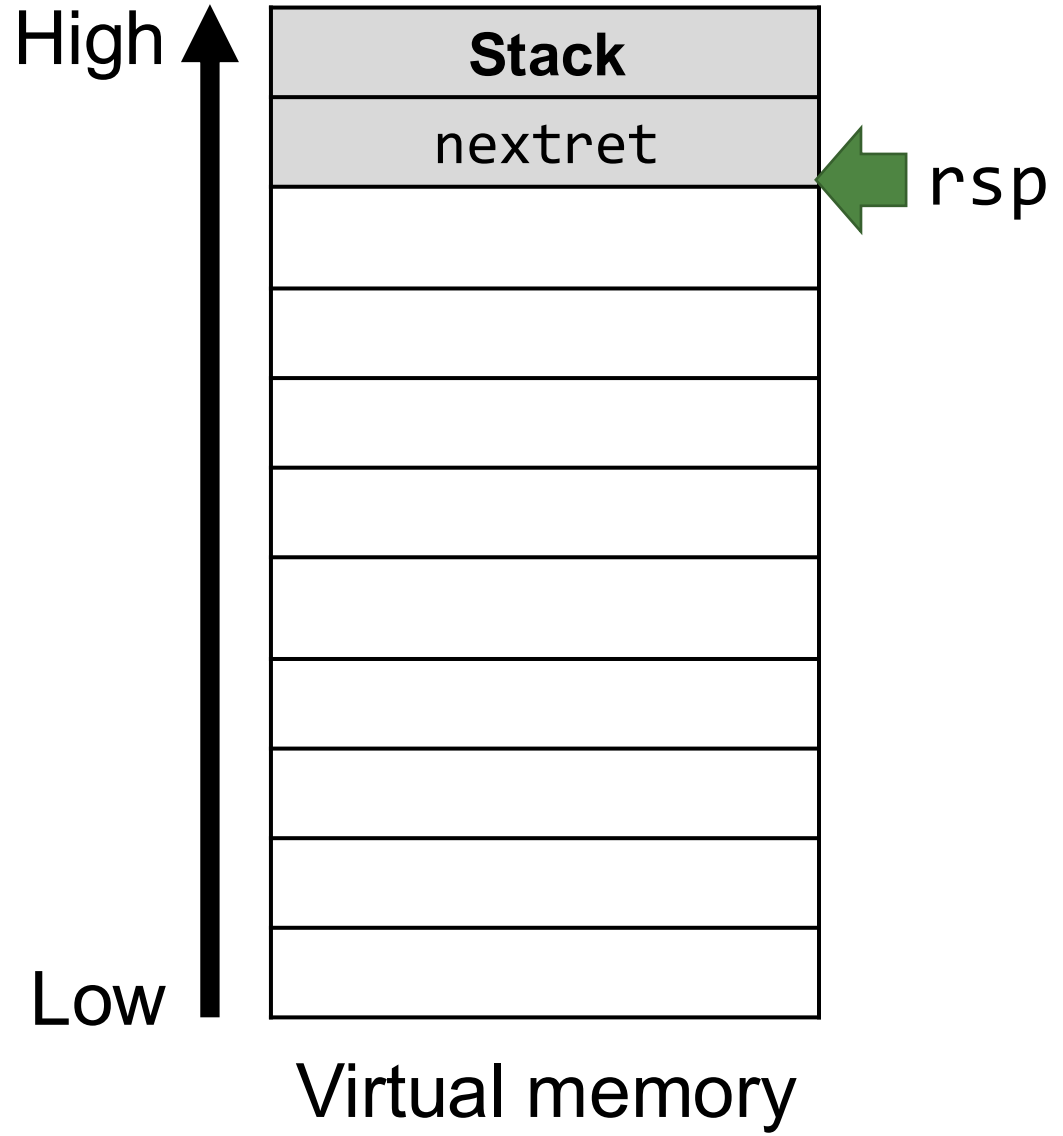


Function Call (call)


```
    …
    call foo
nextret:
    nop
    nop
    …
foo:
    nop
    nop
```

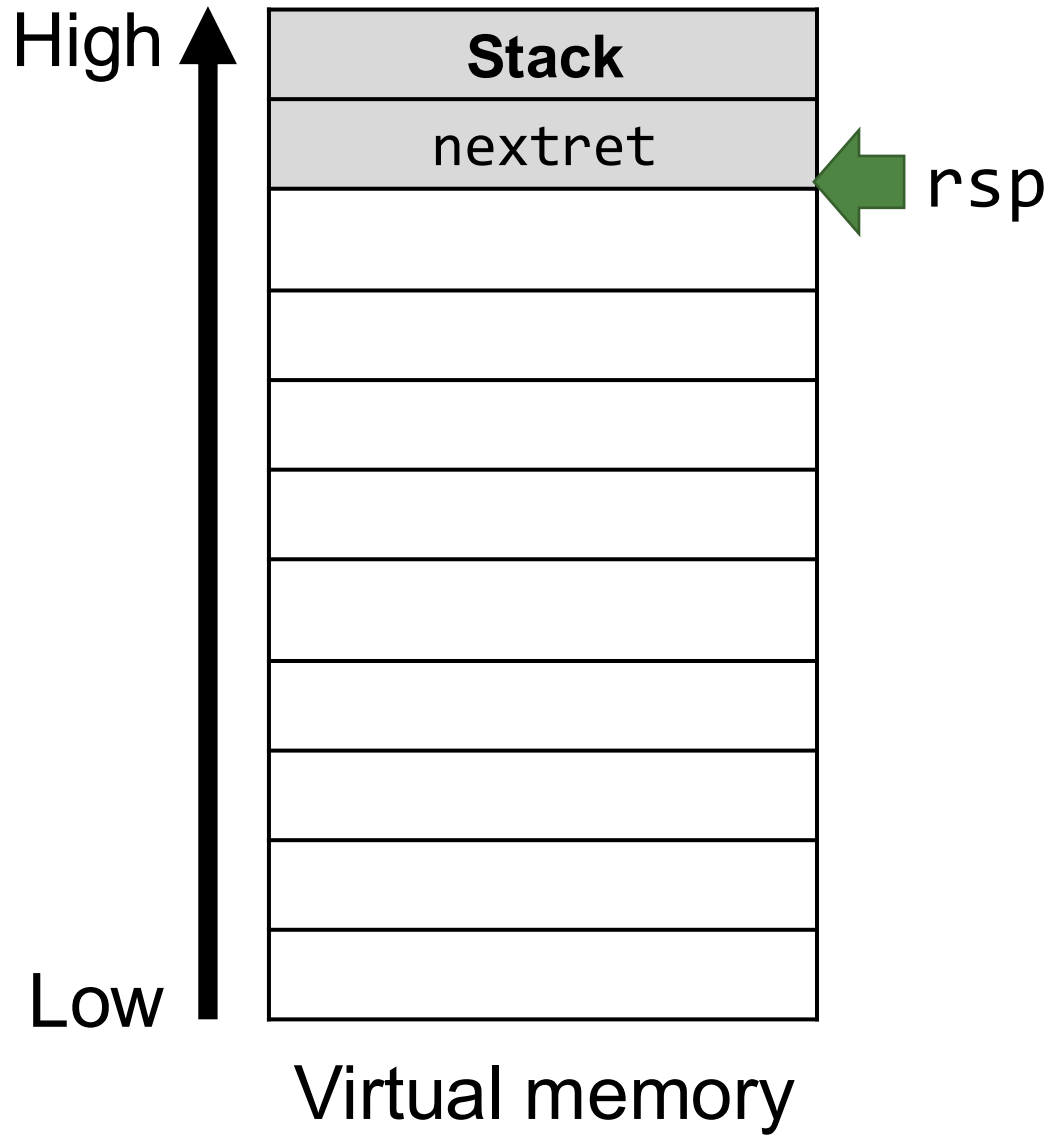
```
push nextret
jmp foo
```

Control flow



Function Return (ret)

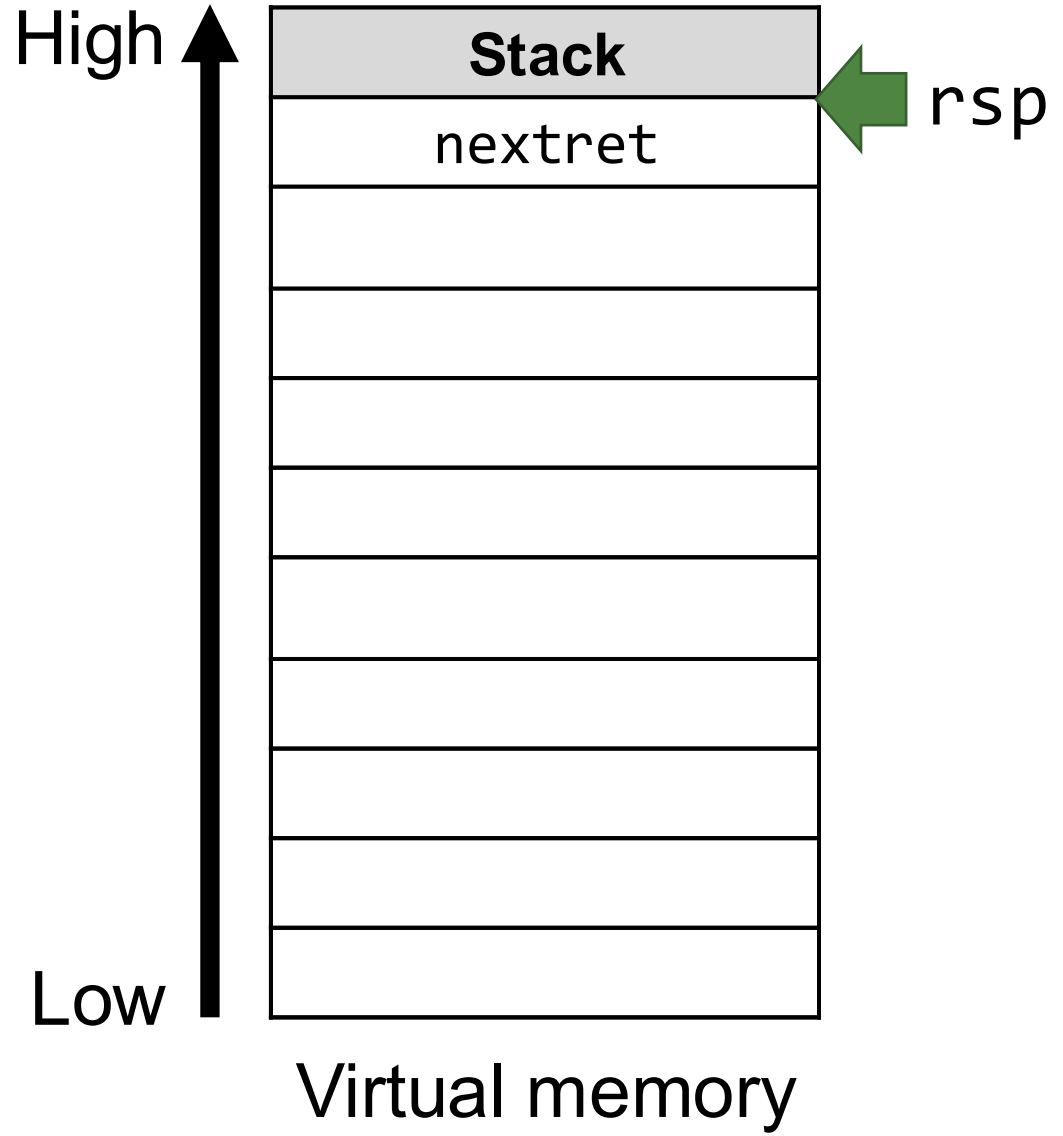
```
ret = pop rip
```



Function Return (ret)

```
ret  =  pop rip
```

rip → nextret



x86 Execution Model

Recap: Stack Frame

- When a function is invoked, a new ***stack frame*** is allocated at the top of the stack memory

- Also, called as procedure frame or activation record

Our Example

```
long purple(long a1, long a2) {
    return 5 + a1 - a2;
}

long blue(long a1) {
    return purple(a1, 4);
}

long red(long a1) {
    return blue(a1 - 42) + a1;
}
```

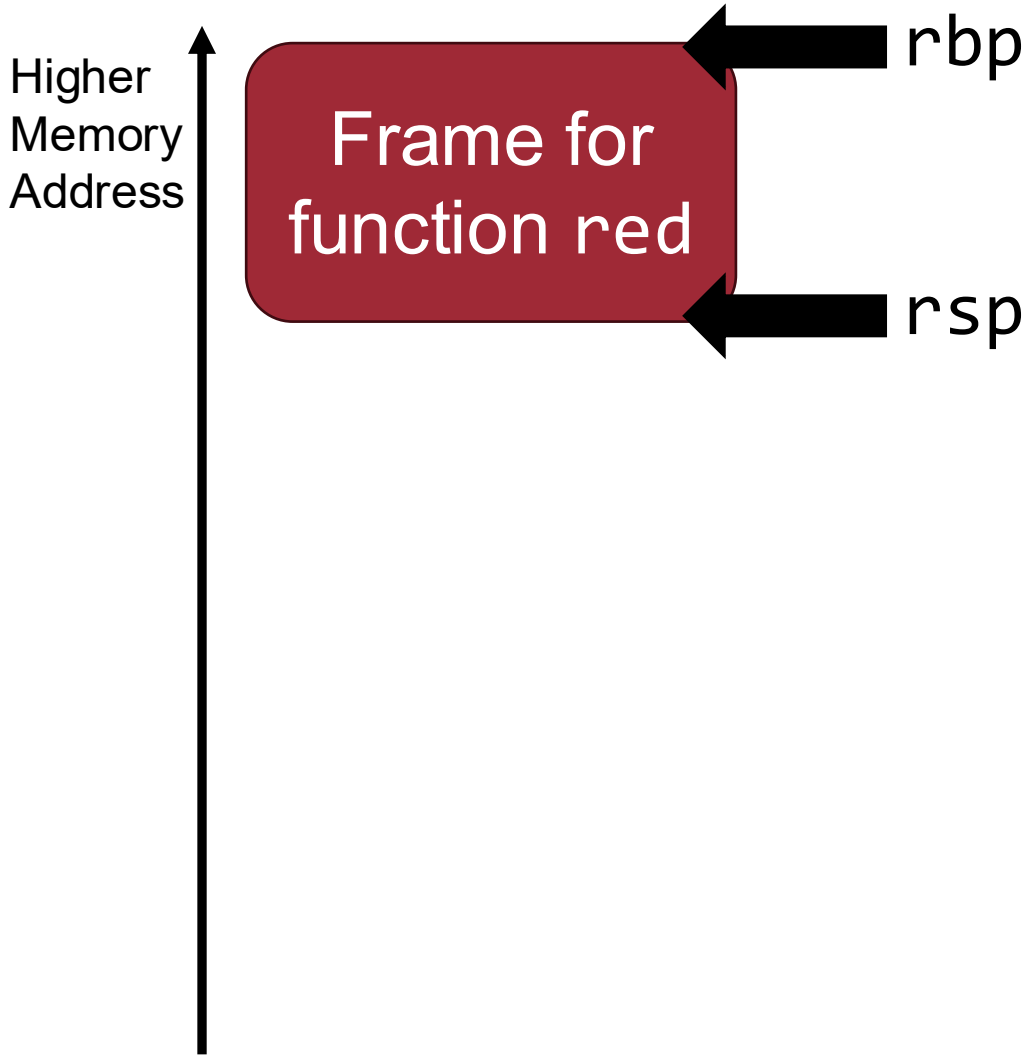
Our Example – Stack

```
long purple(long a1, long a2) {
    return 5 + a1 - a2;
}

long blue(long a1) {
    return purple(a1, 4);
}

long red(long a1) {
    return blue(a1 - 42) + a1;
}
```

Start



Recap: Registers in x86-64

- Program counter (instruction pointer)
  - **`RIP`**: points to the instruction to execute
- Stack pointers
  - **`RSP`**: points to the top of the stack
  - **`RBP`**: points to the base of the current stack frame
- Status register (FLAGS register)
  - **`RFLAGS`**: contains the current condition flags
- Other general purpose registers
  - **`RAX`**, **`RBX`**, **`RCX`**, **`RDX`**, **`RSI`**, **`RDI`**
  - **`R8`**, **`R9`**, **`R10`**, **`R11`**, **`R12`**, **`R13`**, **`R14`**, **`R15`**

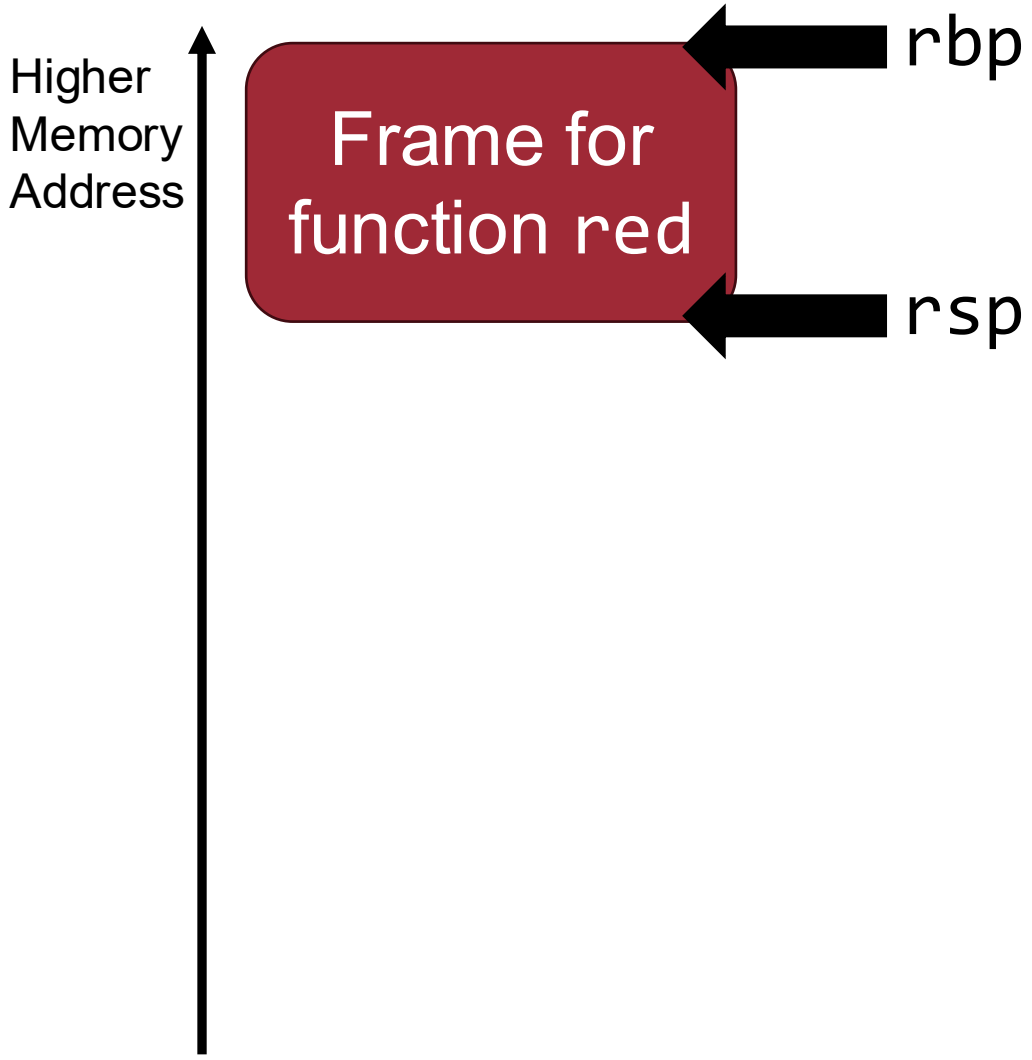
Our Example – Stack

```
long purple(long a1, long a2) {
    return 5 + a1 - a2;
}

long blue(long a1) {
    return purple(a1, 4);
}

long red(long a1) {
    return blue(a1 - 42) + a1;
}
```

Start

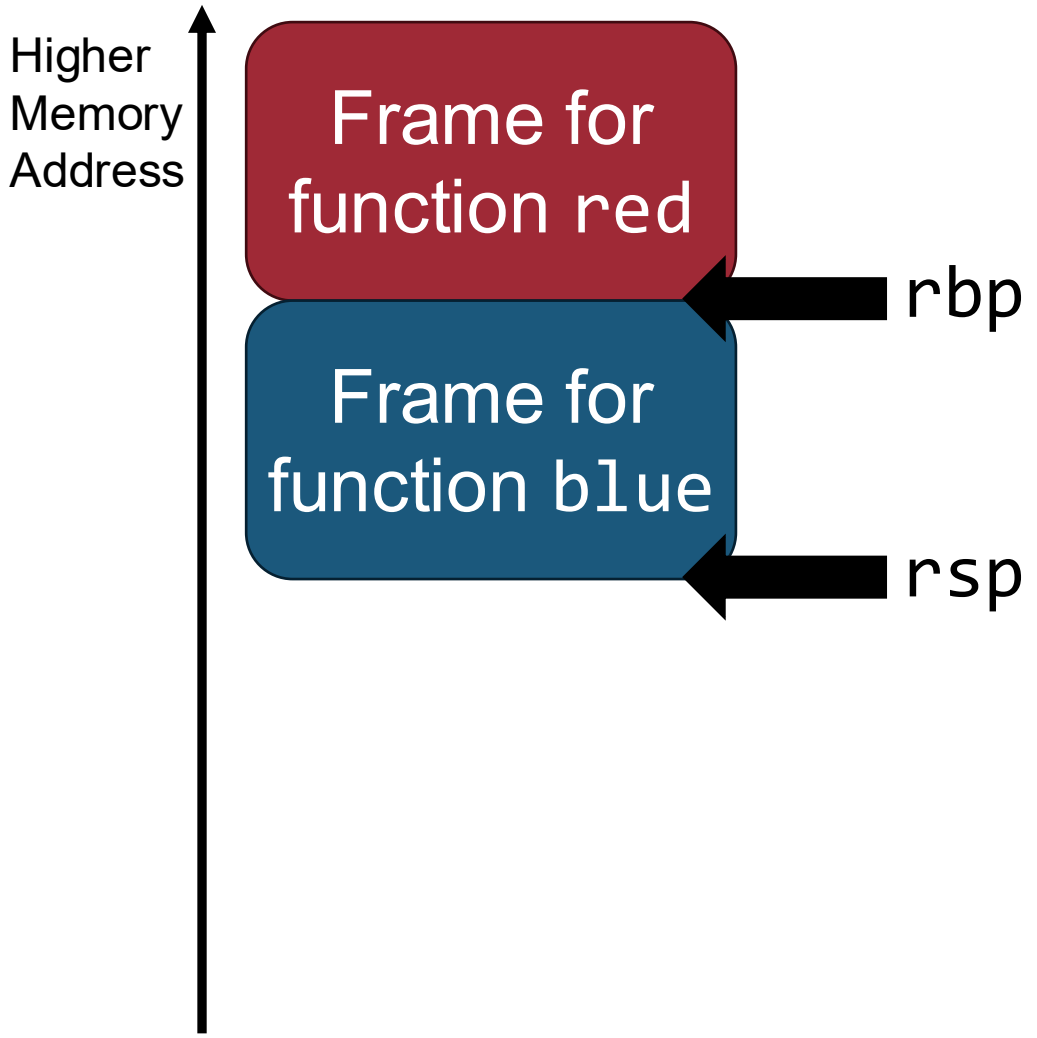


Our Example – Stack

```
long purple(long a1, long a2) {
    return 5 + a1 - a2;
}

long blue(long a1) {
    return purple(a1, 4);
}

long red(long a1) {
    return blue(a1 - 42) + a1;
}
```

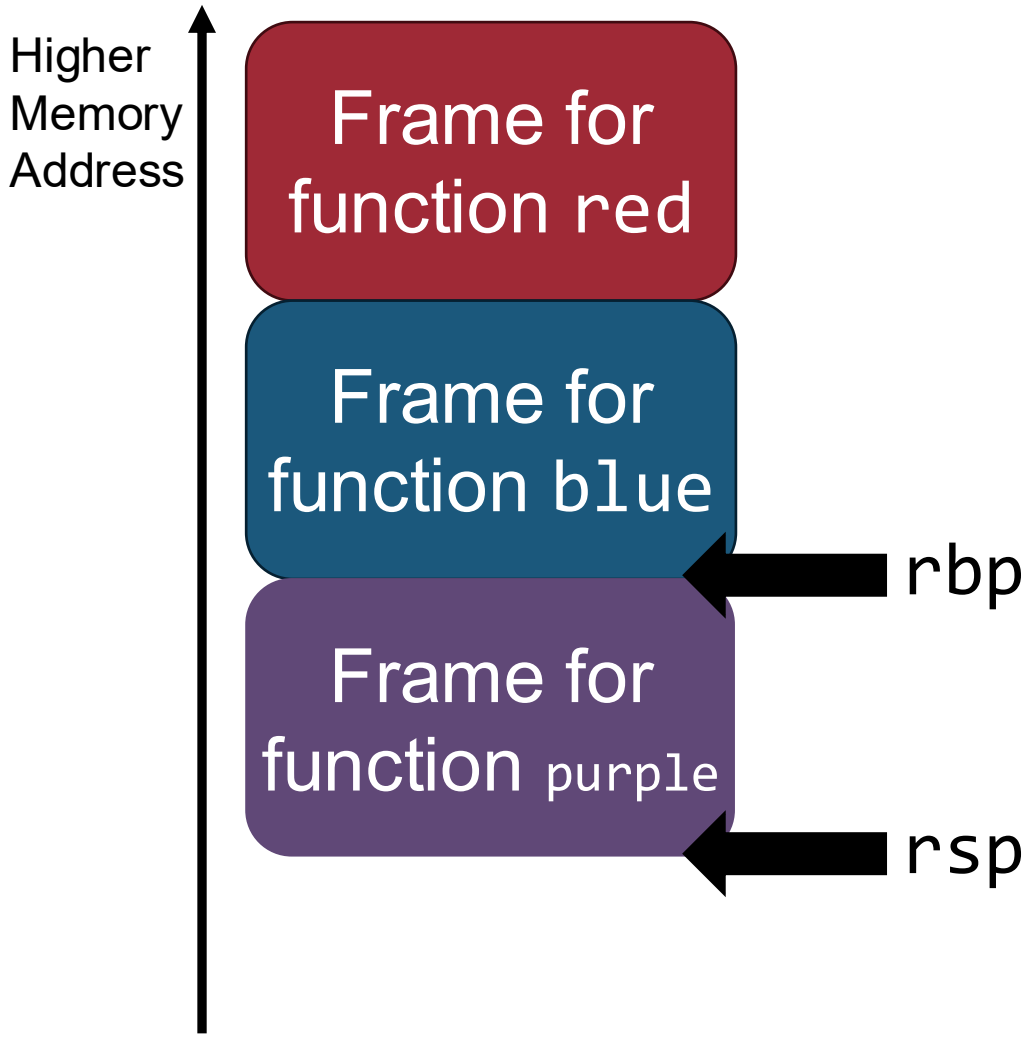


Our Example – Stack

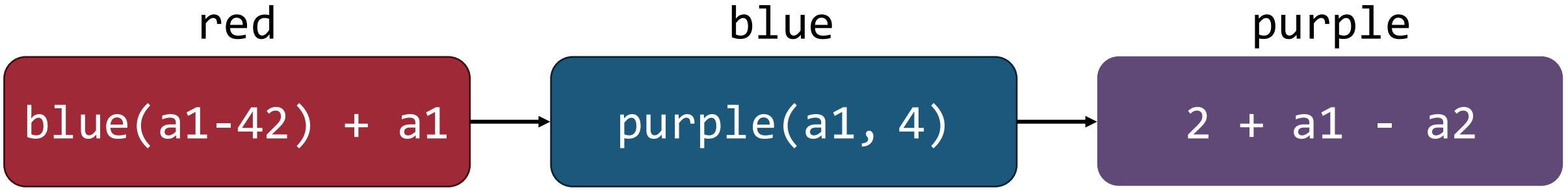
```
long purple(long a1, long a2) {
    return 5 + a1 - a2;
}

long blue(long a1) {
    return purple(a1, 4);
}

long red(long a1) {
    return blue(a1 - 42) + a1;
}
```



Questions



- How do we pass function parameters?
- When a function returns, how do we restore the register values of the caller
- Where do we store local variables?

We can easily get the answer by compiling the example program and disassembling the resulting binary

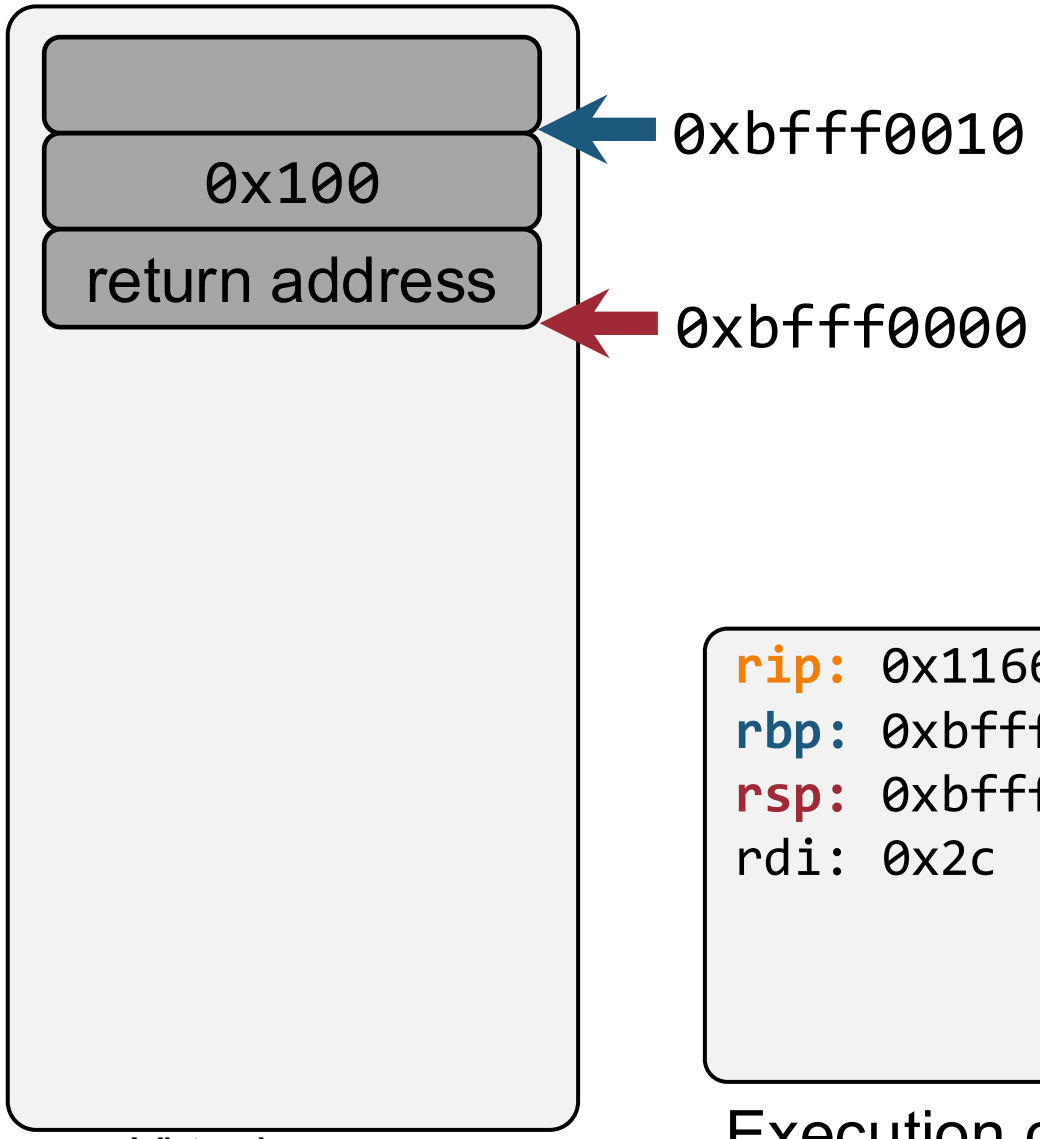
Disassembled Code (x86-64)


```
<red>:
    1166: push  rbp
    1167: mov   rbp, rsp
    116a: sub   rsp, 0x8
    116e: mov   QWORD PTR [rbp-0x8], rdi
    1172: mov   rax, QWORD PTR [rbp-0x8]
    1176: sub   rax, 0x2a
    117a: mov   rdi, rax
    117d: call  blue
    117e: mov   rdx, QWORD PTR [rbp-0x8]
    1182: add   rax, rdx
    1185: leave
    1186: ret

<blue>:
    1143: push  rbp
    1144: mov   rbp, rsp
    1147: sub   rsp, 0x8
    114b: mov   QWORD PTR [rbp-0x8], rdi
    114f: mov   rax, QWORD PTR [rbp-0x8]
    1153: mov    esi, 0x4
    1158: mov    rdi, rax
    115b: call   purple
    1160: leave
    1161: ret

<purple>:
    1129: push  rbp
    112a: mov   rbp, rsp
    112d: mov   QWORD PTR [rbp-0x8], rdi
    1131: mov   QWORD PTR [rbp-0x10], rsi
    1135: mov   rax, QWORD PTR [rbp-0x8]
    1139: add   rax, 0x5
    113d: sub   rax, QWORD PTR [rbp-0x10]
    1141: pop   rbp
    1142: ret
```

Execution Example




```
rip: 0x1166
rbp: 0xbfff0010
rsp: 0xbfff0000
rdi: 0x2c
```

Execution context

```
<red>:
  1166: push rbp
  1167: mov  rbp, rsp
  116a: sub  rsp, 0x8
  116e: mov  QWORD PTR [rbp-0x8], rdi
  1172: mov  rax, QWORD PTR [rbp-0x8]
  1176: sub  rax, 0x2a
  117a: mov  rdi, rax
  117d: call blue
  117e: mov  rdx, QWORD PTR [rbp-0x8]
  1182: add  rax, rdx
  1185: leave
  1186: ret

<blue>:
  1143: push   rbp
  1144: mov    rbp, rsp
  ...
  1160: leave
  1161: ret
```

Execution Example

Points to instruction to be executed

```
rip: 0x1167
rbp: 0xbfff0010
rsp: 0xbfff0000
rdi: 0x2c
```

Execution context

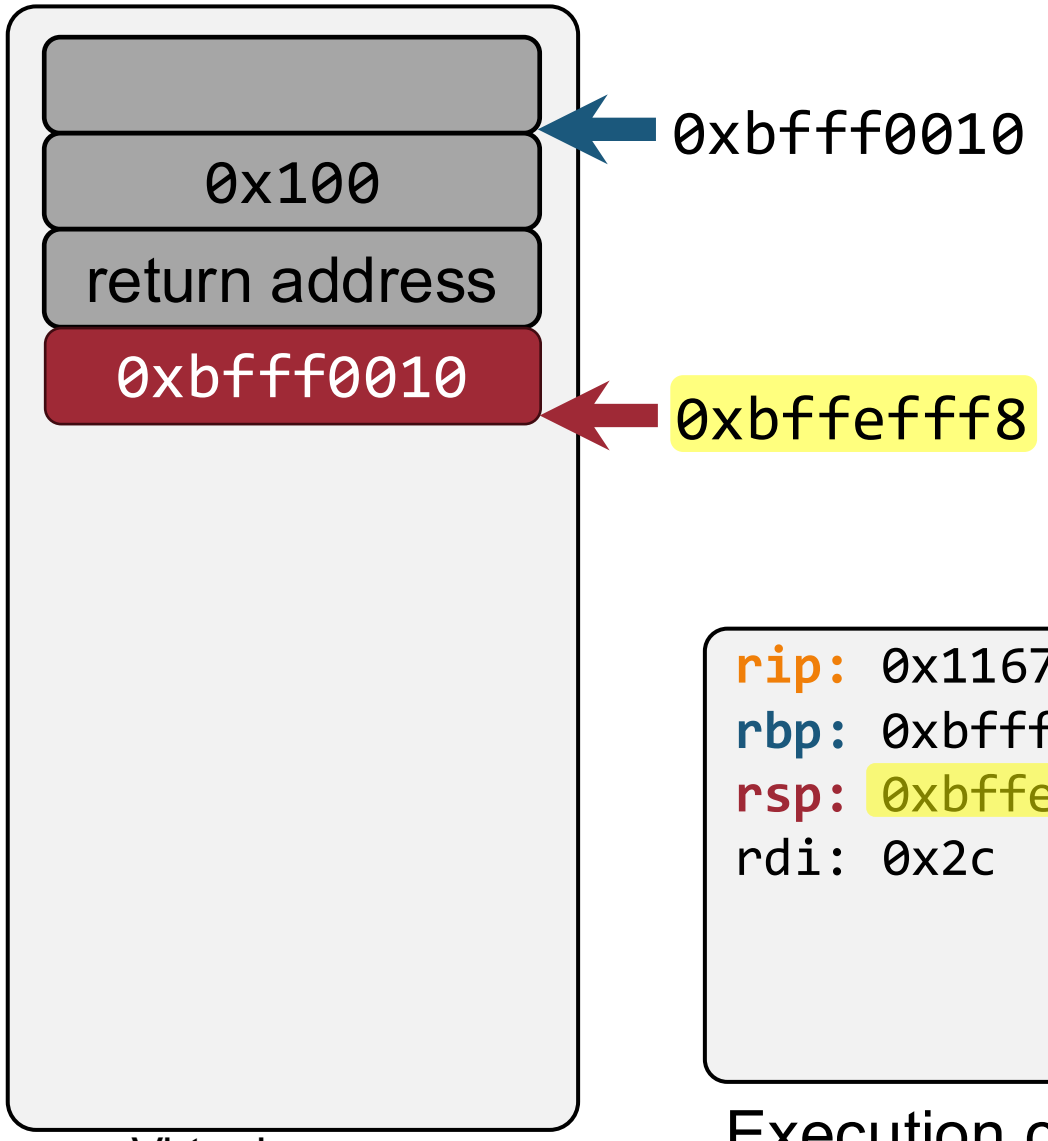
Currently executed instruction

```
<red>:
  1166: push rbp
  1167: mov  rbp, rsp
  116a: sub  rsp, 0x8
  116e: mov  QWORD PTR [rbp-0x8], rdi
  1172: mov  rax, QWORD PTR [rbp-0x8]
  1176: sub  rax, 0x2a
  117a: mov  rdi, rax
  117d: call blue
  117e: mov  rdx, QWORD PTR [rbp-0x8]
  1182: add  rax, rdx
  1185: leave
  1186: ret

<blue>:
  1143: push   rbp
  1144: mov    rbp, rsp
  ...
  1160: leave
  1161: ret
```

Execution Example

Save the base address of the previous function's stack frame



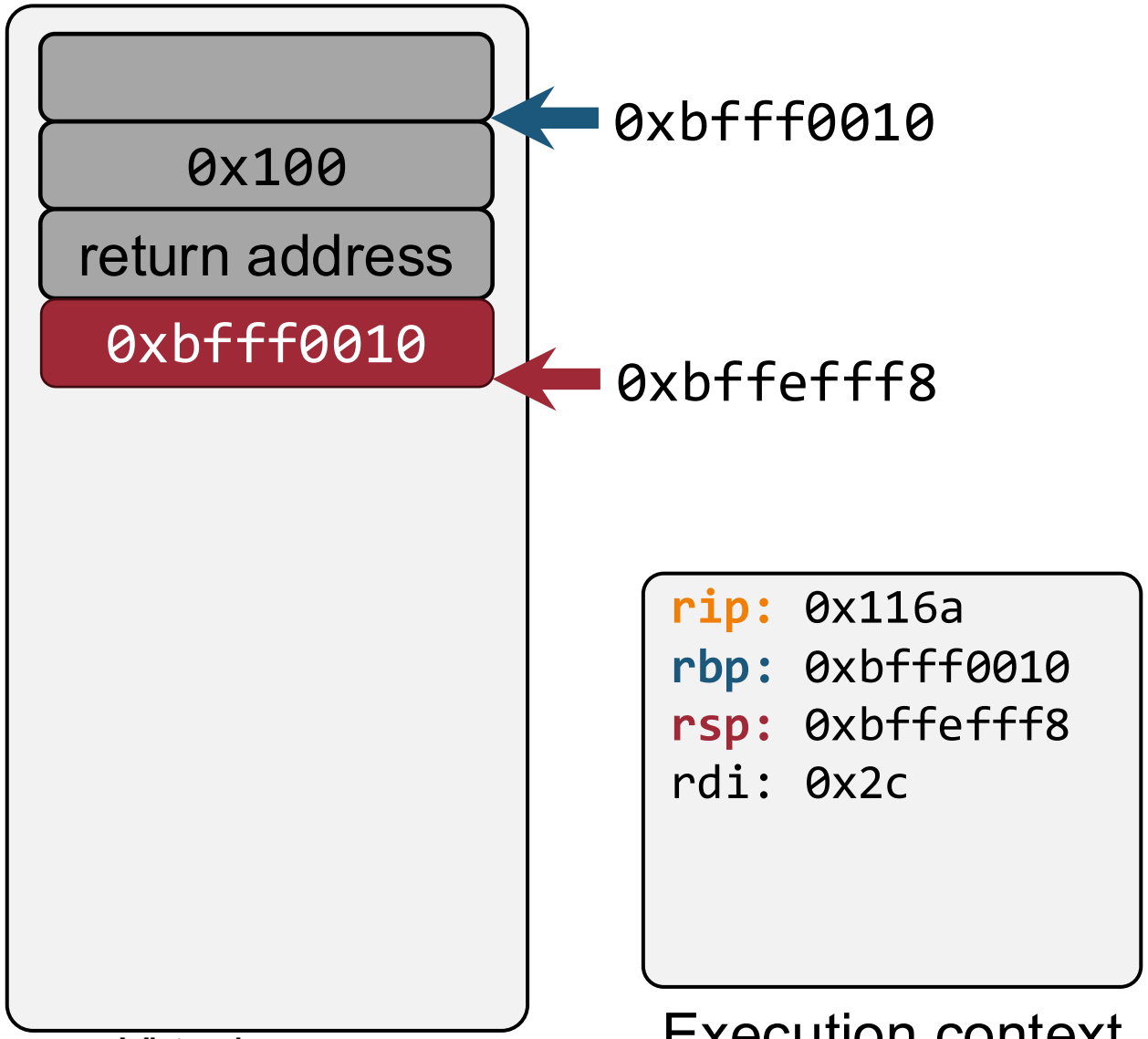

```
rip: 0x1167
rbp: 0xbfff0010
rsp: 0xbffefff8
rdi: 0x2c
```

Execution context

```
<red>:
  1166: push rbp
  1167: mov  rbp, rsp
  116a: sub  rsp, 0x8
  116e: mov  QWORD PTR [rbp-0x8], rdi
  1172: mov  rax, QWORD PTR [rbp-0x8]
  1176: sub  rax, 0x2a
  117a: mov  rdi, rax
  117d: call blue
  117e: mov  rdx, QWORD PTR [rbp-0x8]
  1182: add  rax, rdx
  1185: leave
  1186: ret

<blue>:
  1143: push   rbp
  1144: mov    rbp, rsp
  ...
  1160: leave
  1161: ret
```

Execution Example




```
rip: 0x116a
rbp: 0xbfff0010
rsp: 0xbffefff8
rdi: 0x2c
```

Execution context

```
<red>:
  1166: push rbp
  1167: mov  rbp, rsp
  116a: sub  rsp, 0x8
  116e: mov  QWORD PTR [rbp-0x8], rdi
  1172: mov  rax, QWORD PTR [rbp-0x8]
  1176: sub  rax, 0x2a
  117a: mov  rdi, rax
  117d: call blue
  117e: mov  rdx, QWORD PTR [rbp-0x8]
  1182: add  rax, rdx
  1185: leave
  1186: ret

<blue>:
  1143: push   rbp
  1144: mov    rbp, rsp
  ...
  1160: leave
  1161: ret
```

Execution Example

Now, ebp points to the base of the current stack frame

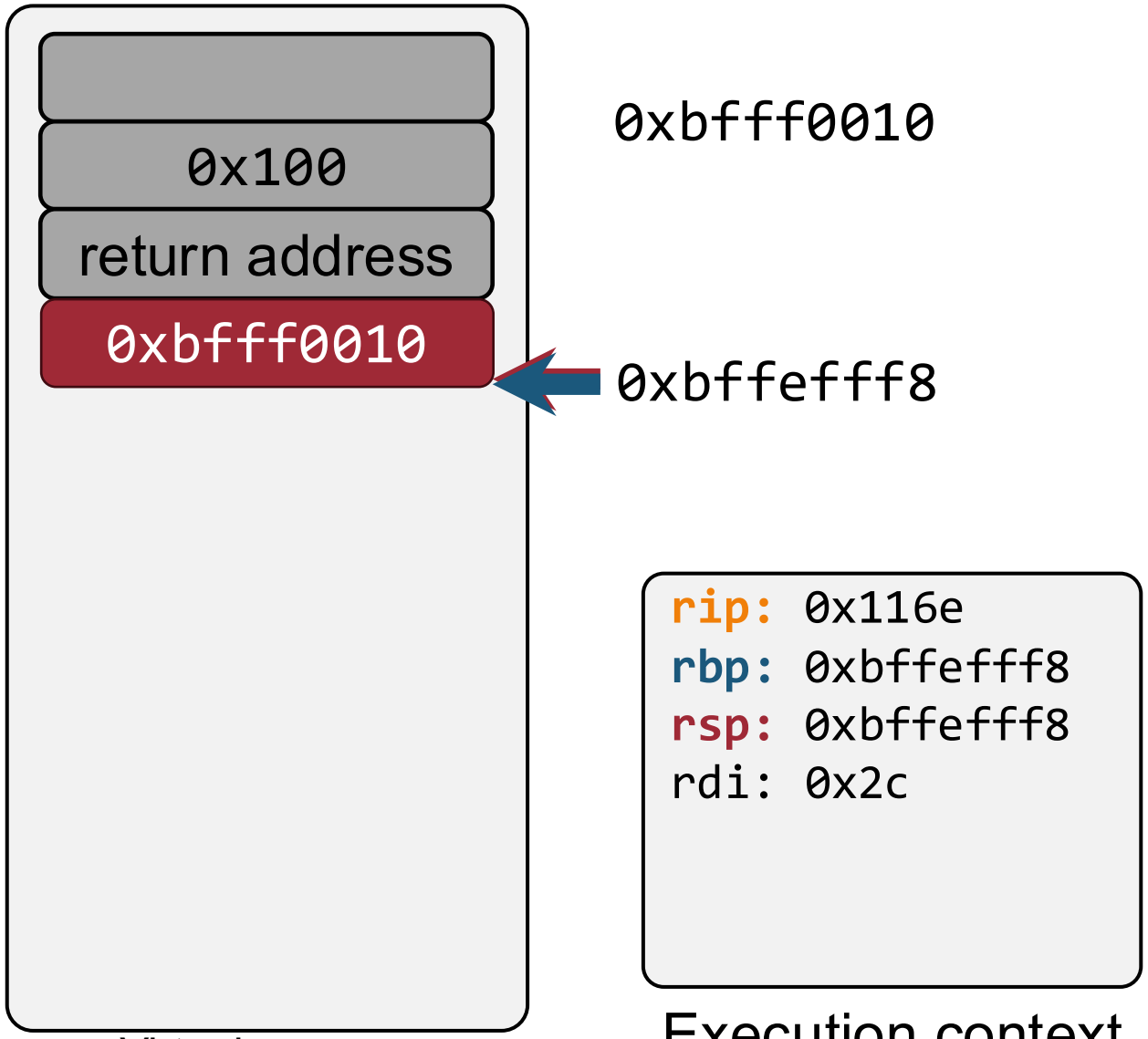

```
rip: 0x116a
rbp: 0xbffefff8
rsp: 0xbffefff8
rdi: 0x2c
```

Execution context

```
<red>:
  1166: push rbp
  1167: mov  rbp, rsp
  116a: sub  rsp, 0x8
  116e: mov  QWORD PTR [rbp-0x8], rdi
  1172: mov  rax, QWORD PTR [rbp-0x8]
  1176: sub  rax, 0x2a
  117a: mov  rdi, rax
  117d: call blue
  117e: mov  rdx, QWORD PTR [rbp-0x8]
  1182: add  rax, rdx
  1185: leave
  1186: ret

<blue>:
  1143: push   rbp
  1144: mov    rbp, rsp
  ...
  1160: leave
  1161: ret
```

Execution Example



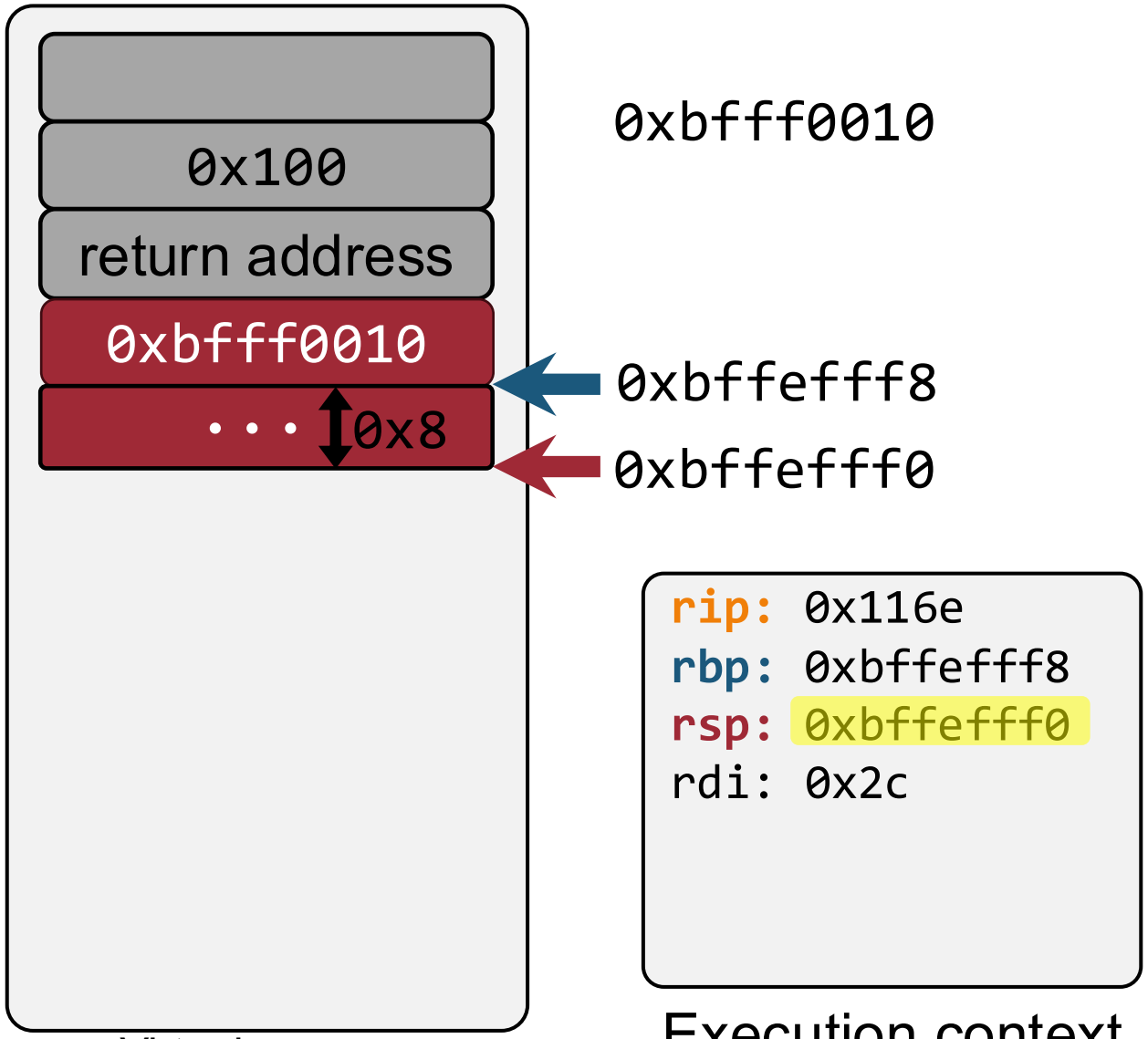

```
rip: 0x116e
rbp: 0xbffefff8
rsp: 0xbffefff8
rdi: 0x2c
```

Execution context

```
<red>:
  1166: push rbp
  1167: mov  rbp, rsp
  116a: sub  rsp, 0x8
  116e: mov  QWORD PTR [rbp-0x8], rdi
  1172: mov  rax, QWORD PTR [rbp-0x8]
  1176: sub  rax, 0x2a
  117a: mov  rdi, rax
  117d: call blue
  117e: mov  rdx, QWORD PTR [rbp-0x8]
  1182: add  rax, rdx
  1185: leave
  1186: ret

<blue>:
  1143: push   rbp
  1144: mov    rbp, rsp
  ...
  1160: leave
  1161: ret
```

Execution Example



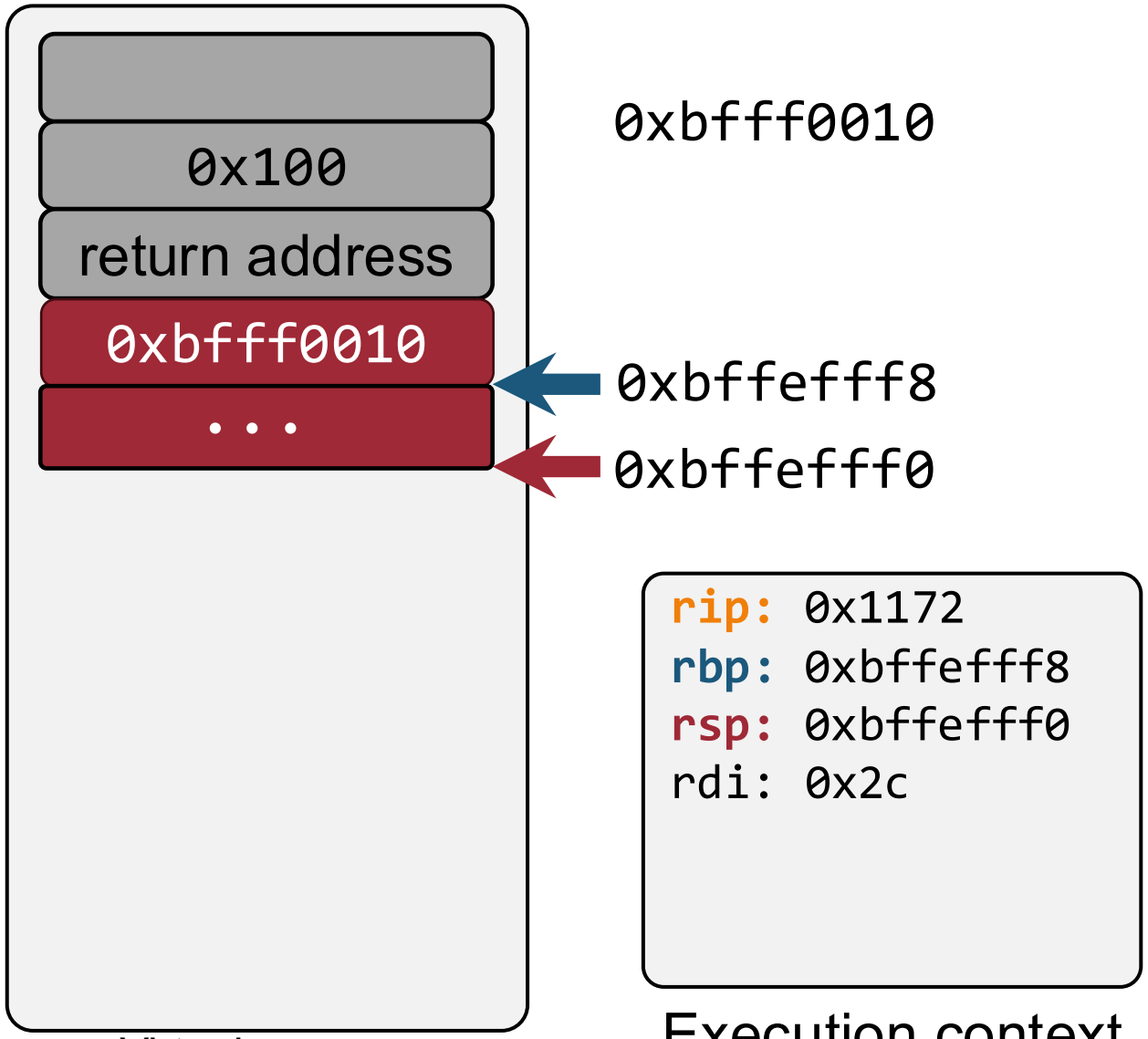

```
rip: 0x116e
rbp: 0xbffefff8
rsp: 0xbffefff0
rdi: 0x2c
```

Execution context

```
<red>:
  1166: push rbp
  1167: mov  rbp, rsp
  116a: sub  rsp, 0x8
  116e: mov  QWORD PTR [rbp-0x8], rdi
  1172: mov  rax, QWORD PTR [rbp-0x8]
  1176: sub  rax, 0x2a
  117a: mov  rdi, rax
  117d: call blue
  117e: mov  rdx, QWORD PTR [rbp-0x8]
  1182: add  rax, rdx
  1185: leave
  1186: ret

<blue>:
  1143: push   rbp
  1144: mov    rbp, rsp
  ...
  1160: leave
  1161: ret
```

Execution Example

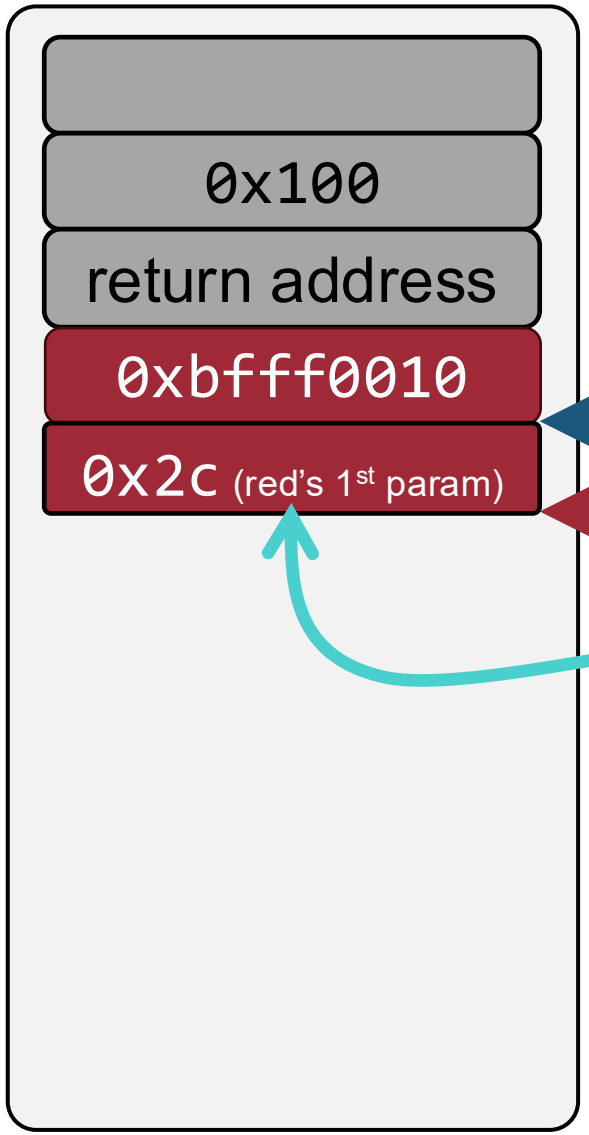



```
<red>:
  1166: push rbp
  1167: mov  rbp, rsp
  116a: sub  rsp, 0x8
  116e: mov  QWORD PTR [rbp-0x8], rdi
  1172: mov  rax, QWORD PTR [rbp-0x8]
  1176: sub  rax, 0x2a
  117a: mov  rdi, rax
  117d: call blue
  117e: mov  rdx, QWORD PTR [rbp-0x8]
  1182: add  rax, rdx
  1185: leave
  1186: ret

<blue>:
  1143: push   rbp
  1144: mov    rbp, rsp
  ...
  1160: leave
  1161: ret
```

Execution Example

rdi stores the value of the first parameter



0xbfff0010

```
long red(long a1) {
    return blue(a1 - 42) + a;
}
```

0xbffefff8

0xbffefff0

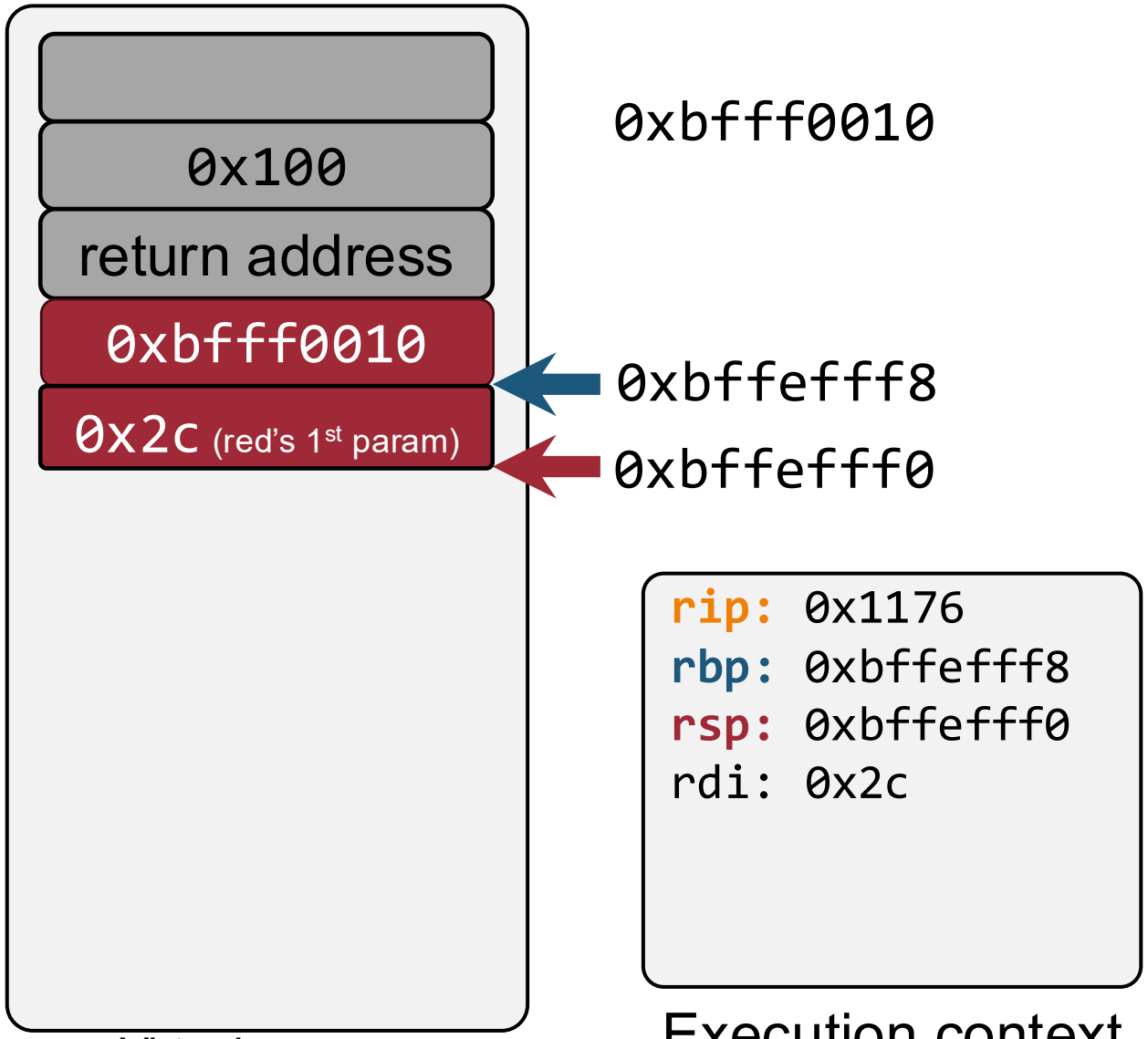
```
rip: 0x1172
rbp: 0xbffefff8
rsp: 0xbffefff0
rdi: 0x2c
```

Execution context

```
                                    rsp
                                    0x8
  116e: mov   QWORD PTR [rbp-0x8], rdi
  1172: mov   rax, QWORD PTR [rbp-0x8]
  1176: sub   rax, 0x2a
  117a: mov   rdi, rax
  117d: call  blue
  117e: mov   rdx, QWORD PTR [rbp-0x8]
  1182: add   rax, rdx
  1185: leave
  1186: ret

<blue>:
  1143: push   rbp
  1144: mov    rbp, rsp
  ...
  1160: leave
  1161: ret
```

Execution Example



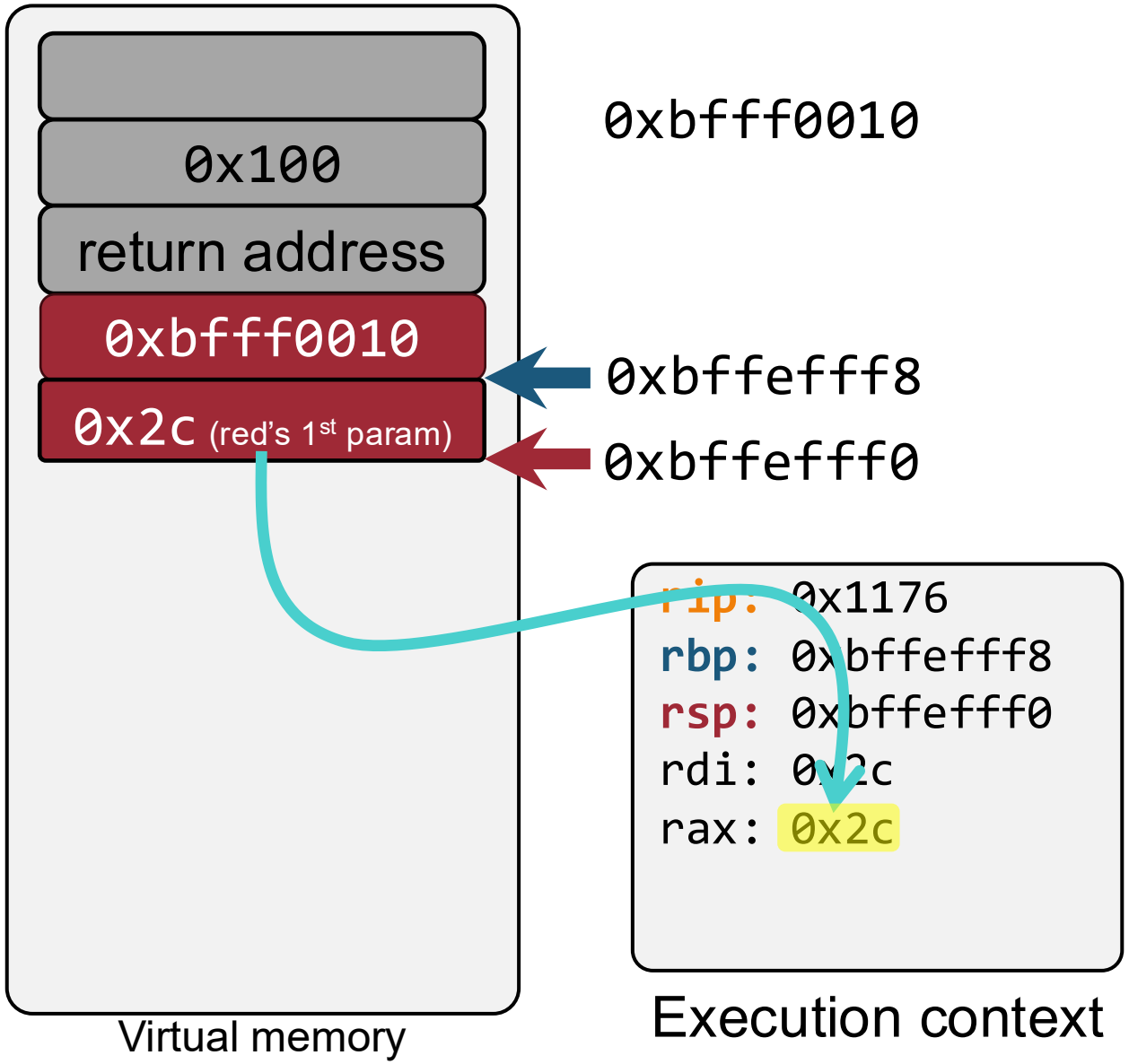

```
rip: 0x1176
rbp: 0xbffefff8
rsp: 0xbffefff0
rdi: 0x2c
```

Execution context

```
<red>:
  1166: push rbp
  1167: mov  rbp, rsp
  116a: sub  rsp, 0x8
  116e: mov  QWORD PTR [rbp-0x8], rdi
  1172: mov  rax, QWORD PTR [rbp-0x8]
  1176: sub  rax, 0x2a
  117a: mov  rdi, rax
  117d: call blue
  117e: mov  rdx, QWORD PTR [rbp-0x8]
  1182: add  rax, rdx
  1185: leave
  1186: ret

<blue>:
  1143: push   rbp
  1144: mov    rbp, rsp
  ...
  1160: leave
  1161: ret
```

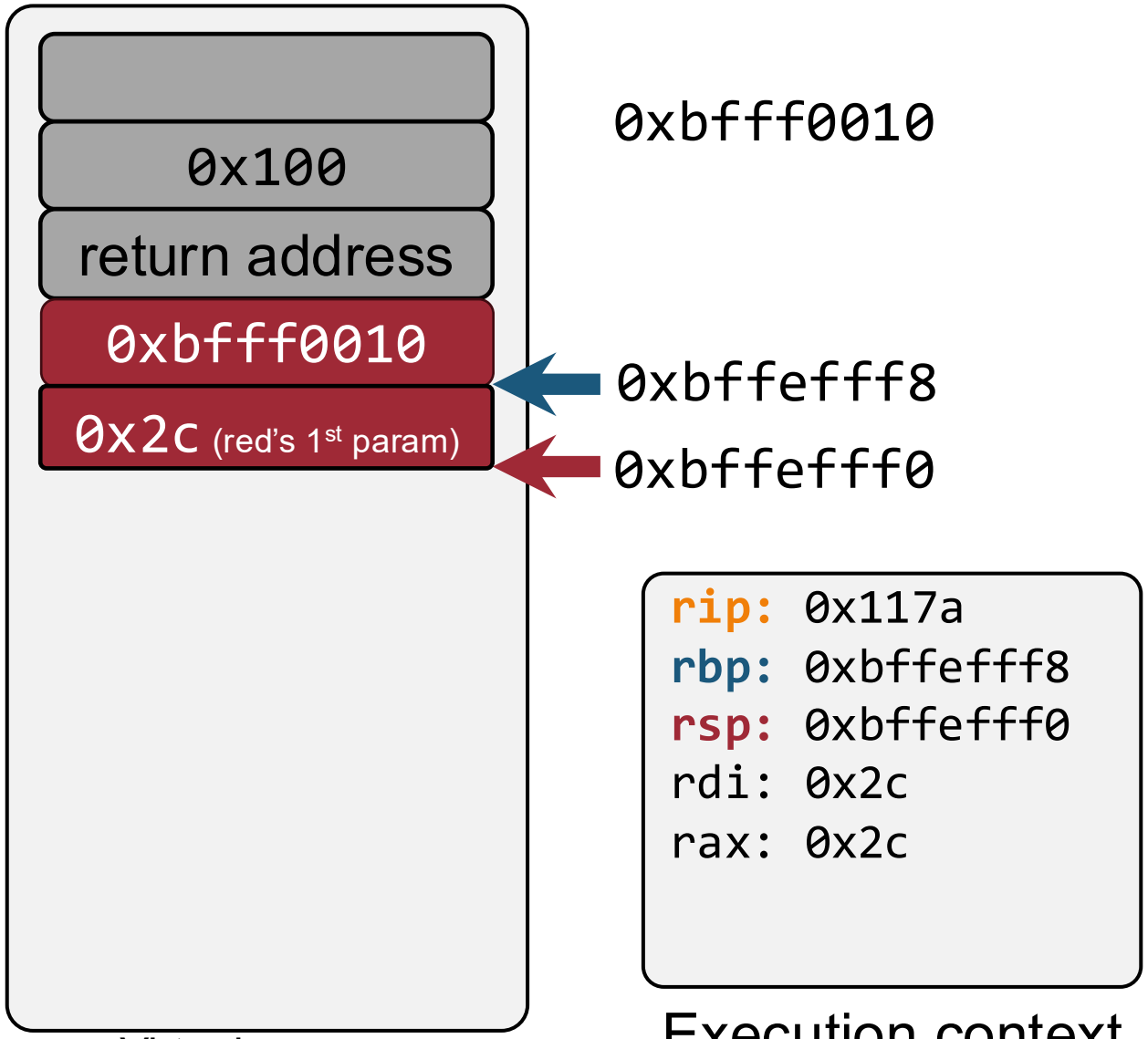
Execution Example




```
<red>:
  1166: push rbp
  1167: mov  rbp, rsp
  116a: sub  rsp, 0x8
  116e: mov  QWORD PTR [rbp-0x8], rdi
  1172: mov  rax, QWORD PTR [rbp-0x8]
  1176: sub  rax, 0x2a
  117a: mov  rdi, rax
  117d: call blue
  117e: mov  rdx, QWORD PTR [rbp-0x8]
  1182: add  rax, rdx
  1185: leave
  1186: ret

<blue>:
  1143: push   rbp
  1144: mov    rbp, rsp
  ...
  1160: leave
  1161: ret
```

Execution Example



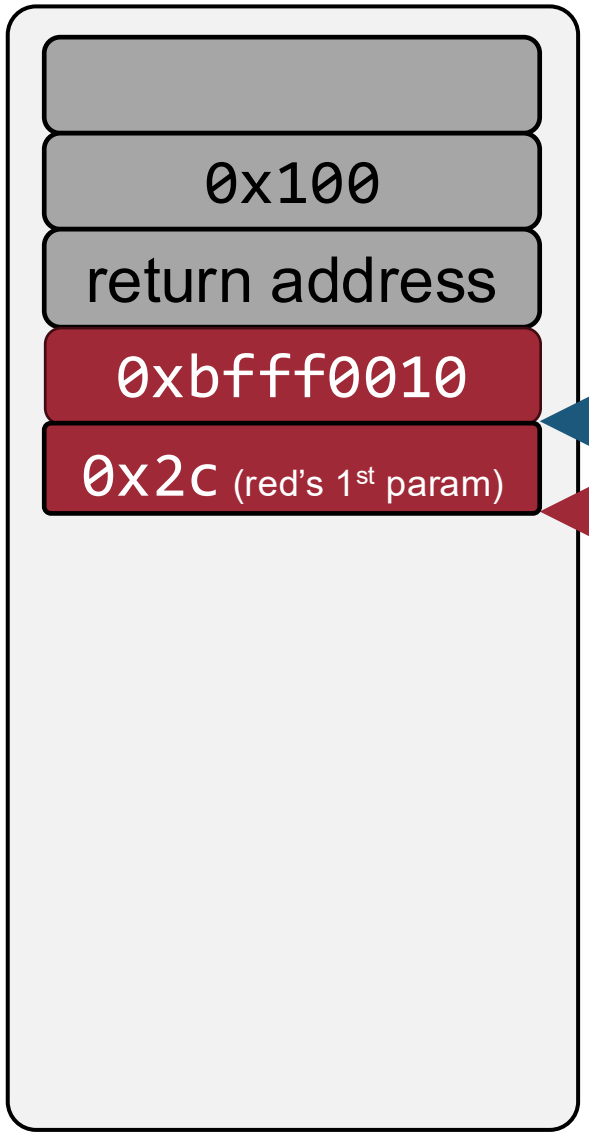

```
rip: 0x117a
rbp: 0xbffefff8
rsp: 0xbffefff0
rdi: 0x2c
rax: 0x2c
```

Execution context

```
<red>:
  1166: push rbp
  1167: mov  rbp, rsp
  116a: sub  rsp, 0x8
  116e: mov  QWORD PTR [rbp-0x8], rdi
  1172: mov  rax, QWORD PTR [rbp-0x8]
  1176: sub  rax, 0x2a
  117a: mov  rdi, rax
  117d: call blue
  117e: mov  rdx, QWORD PTR [rbp-0x8]
  1182: add  rax, rdx
  1185: leave
  1186: ret

<blue>:
  1143: push   rbp
  1144: mov    rbp, rsp
  ...
  1160: leave
  1161: ret
```

Execution Example



0xbfff0010

0xbffefff8

0xbffefff0

```
long red(long a1) {
    return blue(a1 - 42) + a1;
}
```

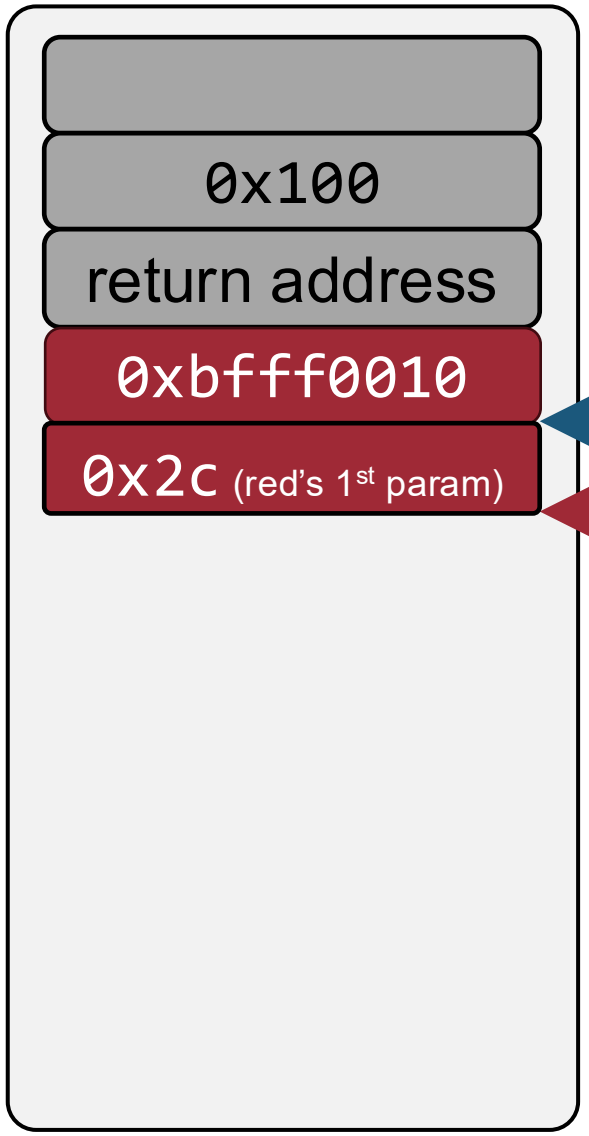
```
rip: 0x117a
rbp: 0xbffefff8
rsp: 0xbffefff0
rdi: 0x2c
rax: 0x2
```

Execution context

```
<red>:
  1166: push rbp
  1167: mov  rbp, rsp
  116a: sub  rsp, 0x8
  116e: mov  QWORD PTR [rbp-0x8], rdi
  1172: mov  rax, QWORD PTR [rbp-0x8]
  1176: sub  rax, 0x2a
  117a: mov  rdi, rax
  117d: call blue
  117e: mov  rdx, QWORD PTR [rbp-0x8]
  1182: add  rax, rdx
  1185: leave
  1186: ret

<blue>:
  1143: push   rbp
  1144: mov    rbp, rsp
  ...
  1160: leave
  1161: ret
```

Execution Example



0xbfff0010

0xbffefff8

0xbffefff0

```
long red(long a1) {
    return blue(a1 - 42) + a1;
}
```

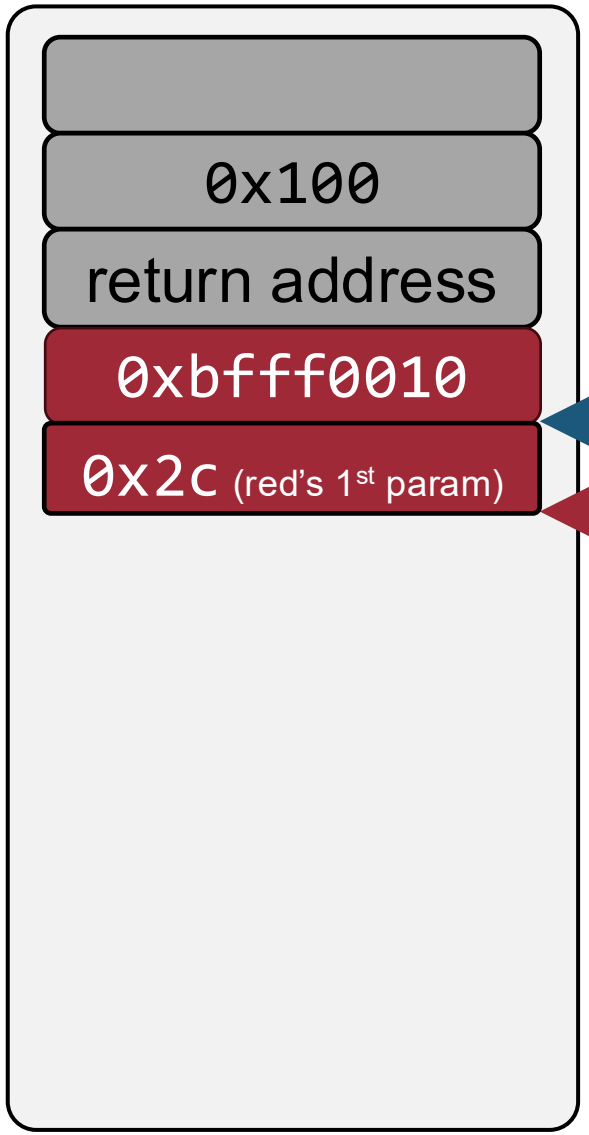
```
rip: 0x117d
rbp: 0xbffefff8
rsp: 0xbffefff0
rdi: 0x2c
rax: 0x2
```

Execution context

```
<red>:
  1166: push rbp
  1167: mov  rbp, rsp
  116a: sub  rsp, 0x8
  116e: mov  QWORD PTR [rbp-0x8], rdi
  1172: mov  rax, QWORD PTR [rbp-0x8]
  1176: sub  rax, 0x2a
  117a: mov  rdi, rax
  117d: call blue
  117e: mov  rdx, QWORD PTR [rbp-0x8]
  1182: add  rax, rdx
  1185: leave
  1186: ret

<blue>:
  1143: push   rbp
  1144: mov    rbp, rsp
  ...
  1160: leave
  1161: ret
```

Execution Example




```
long red(long a1) {
    return blue(a1 - 42) + a1;
}
```

Prepare rdi (the first parameter) for blue( )

```
rip: 0x117d
rbp: 0xbffefff8
rsp: 0xbffefff0
rdi: 0x2
rax: 0x2
```

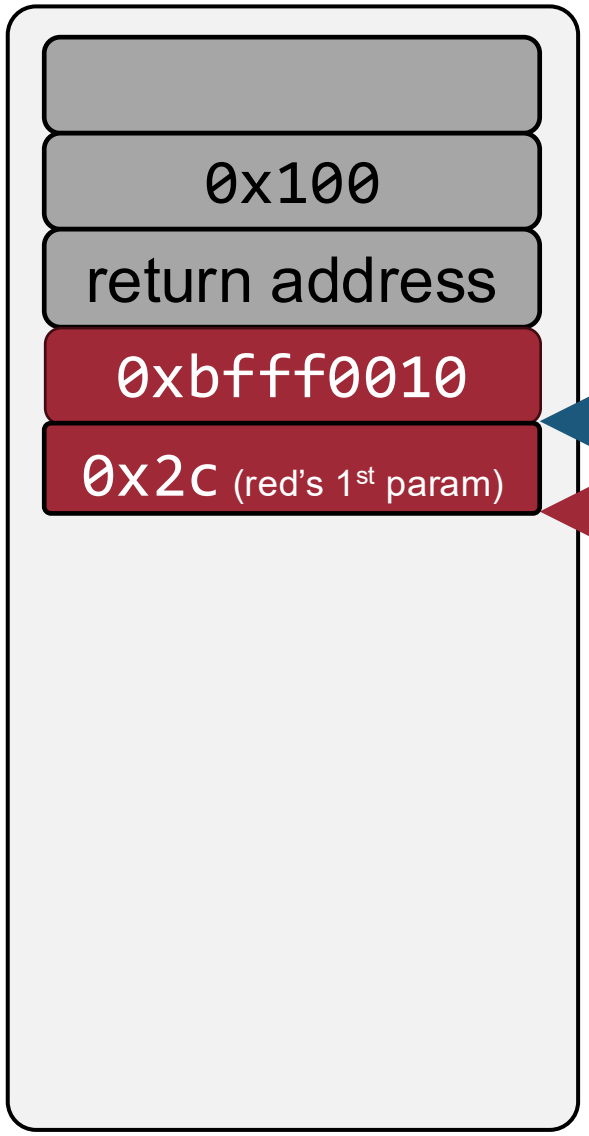
Execution context

```
<red>:
  116
  116
  1172: mov   rax, QWORD PTR [rbp-0x8]
  1176: sub   rax, 0x2a
  117a: mov   rdi, rax
  117d: call  blue
  117e: mov   rdx, QWORD PTR [rbp-0x8]
  1182: add   rax, rdx
  1185: leave
  1186: ret

<blue>:
  1143: push  rbp
  1144: mov   rbp, rsp
  ...
  1160: leave
  1161: ret
```

```
0x8
RD PTR [rbp-0x8], rdi
```

Execution Example



0xbfffefff8

0xbfffefff0

push retaddress
jmp blue

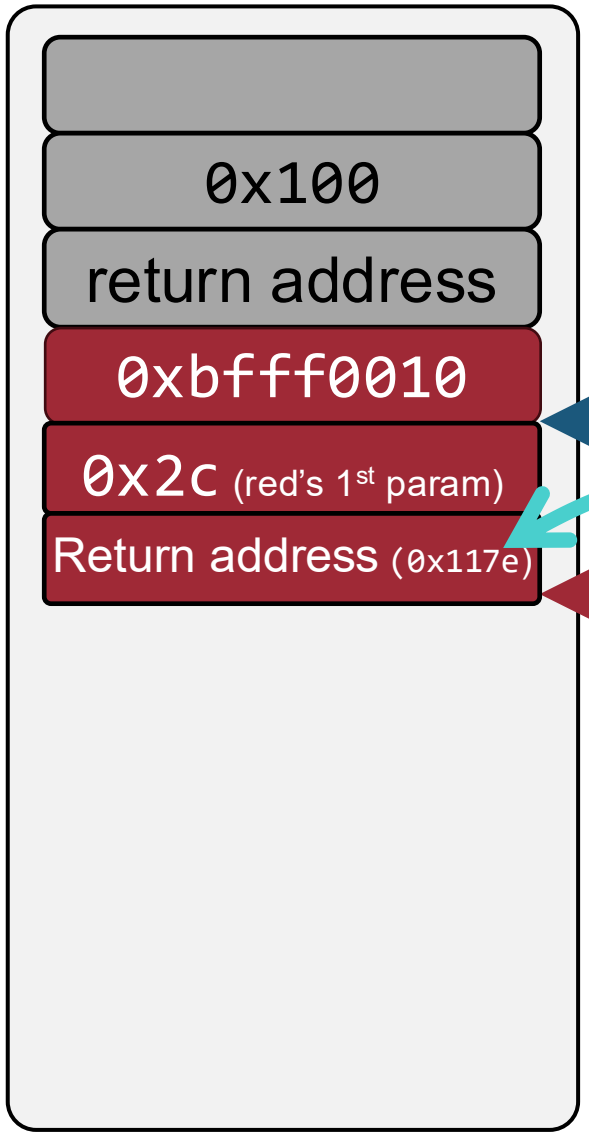
```
rip: 0x117e
rbp: 0xbfffefff8
rsp: 0xbfffefff0
rdi: 0x2
rax: 0x2
```

Execution context

```
<red>:
  1166: push rbp
  1167: mov  rbp, rsp
  116a: sub  rsp, 0x8
  116e: mov  QWORD PTR [rbp-0x8], rdi
  1172: mov  rax, QWORD PTR [rbp-0x8]
  1176: sub  rax, 0x2a
  117a: mov  rdi, rax
  117d: call blue
  117e: mov  rdx, QWORD PTR [rbp-0x8]
  1182: add  rax, rdx
  1185: leave
  1186: ret

<blue>:
  1143: push   rbp
  1144: mov    rbp, rsp
  ...
  1160: leave
  1161: ret
```

Execution Example



0xbfff0010

push retaddress
jmp blue

0xbffefff8

0xbffefff0

0xbffeffe8

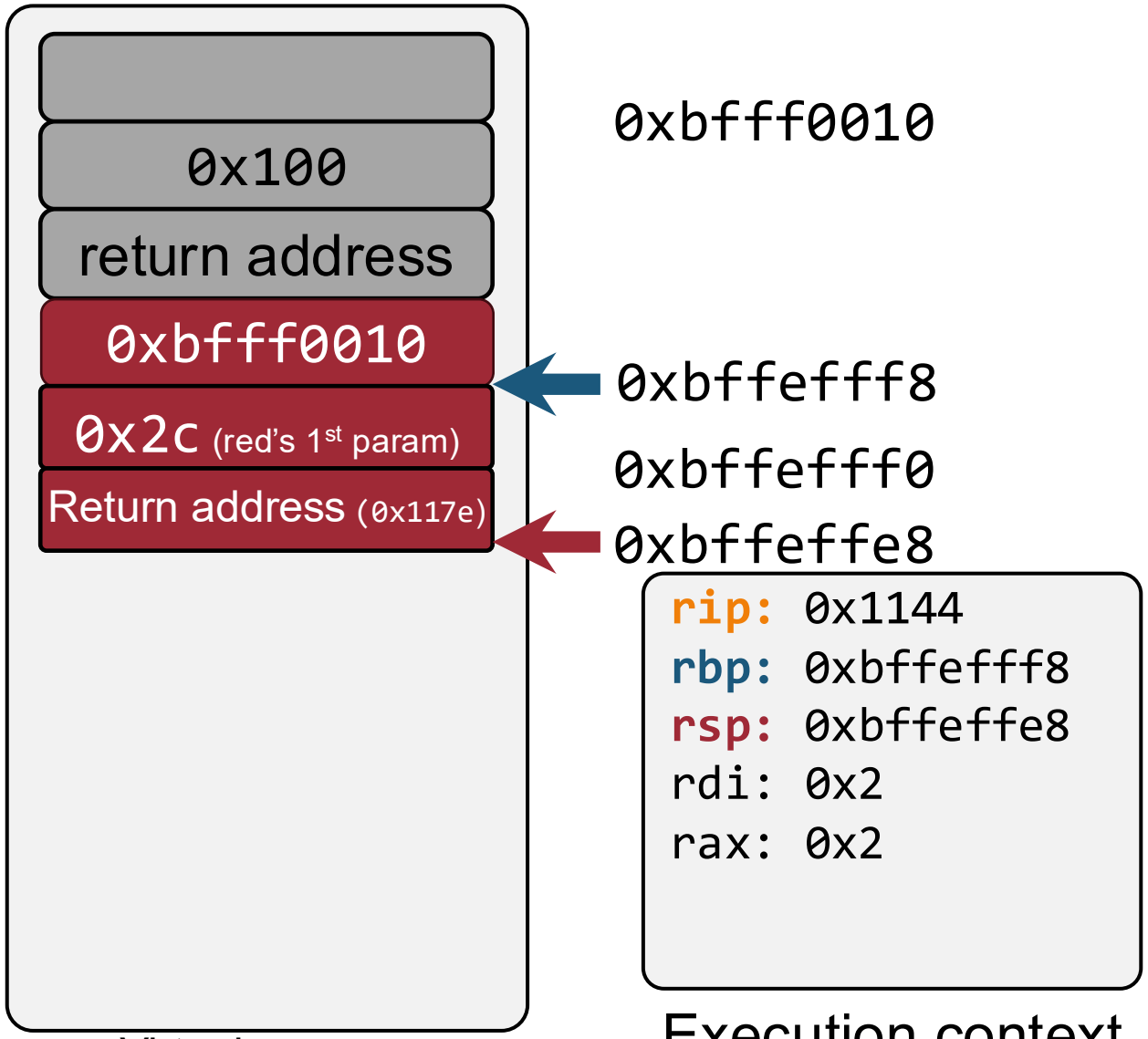
```
rip: 0x1143
rbp: 0xbffefff8
rsp: 0xbffeffe8
rdi: 0x2
rax: 0x2
```

Execution context

```
<red>:
   1166: push rbp
   1167: mov  rbp, rsp
   116a: sub  rsp, 0x8
   116e: mov  QWORD PTR [rbp-0x8], rdi
   1172: mov  rax, QWORD PTR [rbp-0x8]
   1176: sub  rax, 0x2a
   117a: mov  rdi, rax
   117d: call blue
   117e: mov  rdx, QWORD PTR [rbp-0x8]
   1182: add  rax, rdx
   1185: leave
   1186: ret

<blue>:
   1143: push   rbp
   1144: mov    rbp, rsp
   ...
   1160: leave
   1161: ret
```

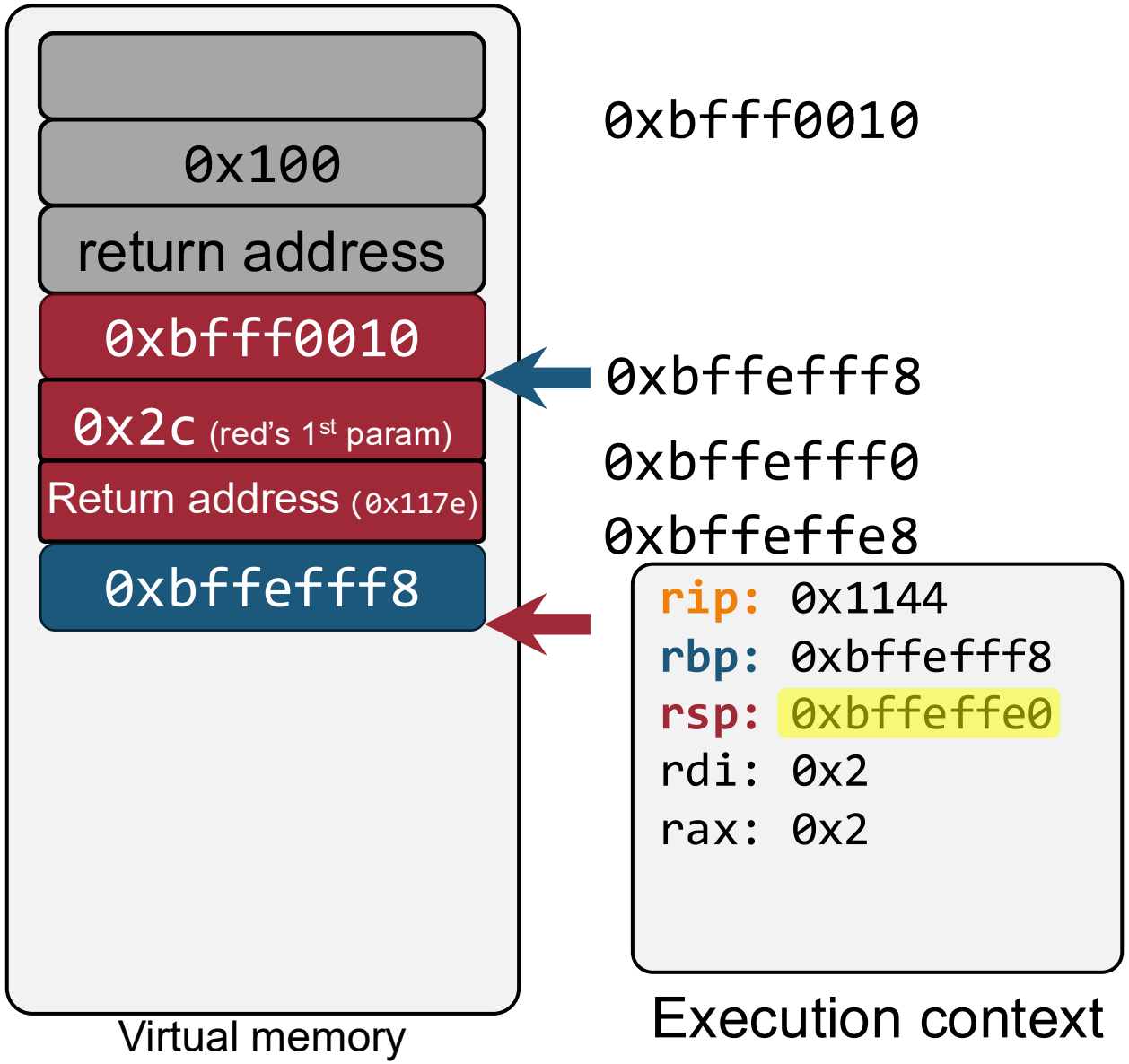
Execution Example




```
<red>:
   1166: push rbp
   1167: mov  rbp, rsp
   116a: sub  rsp, 0x8
   116e: mov  QWORD PTR [rbp-0x8], rdi
   1172: mov  rax, QWORD PTR [rbp-0x8]
   1176: sub  rax, 0x2a
   117a: mov  rdi, rax
   117d: call blue
   117e: mov  rdx, QWORD PTR [rbp-0x8]
   1182: add  rax, rdx
   1185: leave
   1186: ret

<blue>:
   1143: push   rbp
   1144: mov    rbp, rsp
   ...
   1160: leave
   1161: ret
```

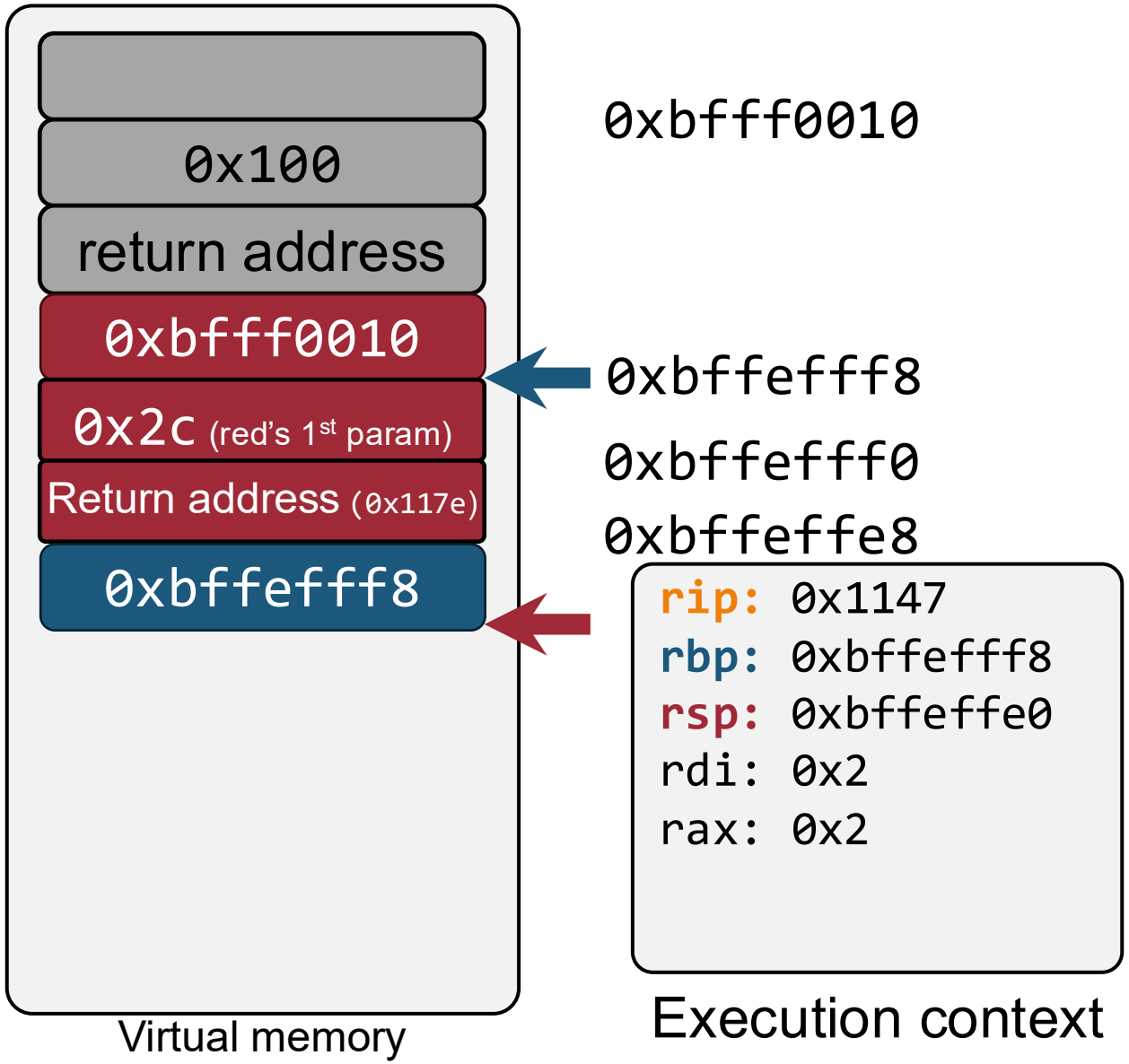
Execution Example




```
<red>:
   1166: push rbp
   1167: mov  rbp, rsp
   116a: sub  rsp, 0x8
   116e: mov  QWORD PTR [rbp-0x8], rdi
   1172: mov  rax, QWORD PTR [rbp-0x8]
   1176: sub  rax, 0x2a
   117a: mov  rdi, rax
   117d: call blue
   117e: mov  rdx, QWORD PTR [rbp-0x8]
   1182: add  rax, rdx
   1185: leave
   1186: ret

<blue>:
   1143: push   rbp
   1144: mov    rbp, rsp
   ...
   1160: leave
   1161: ret
```

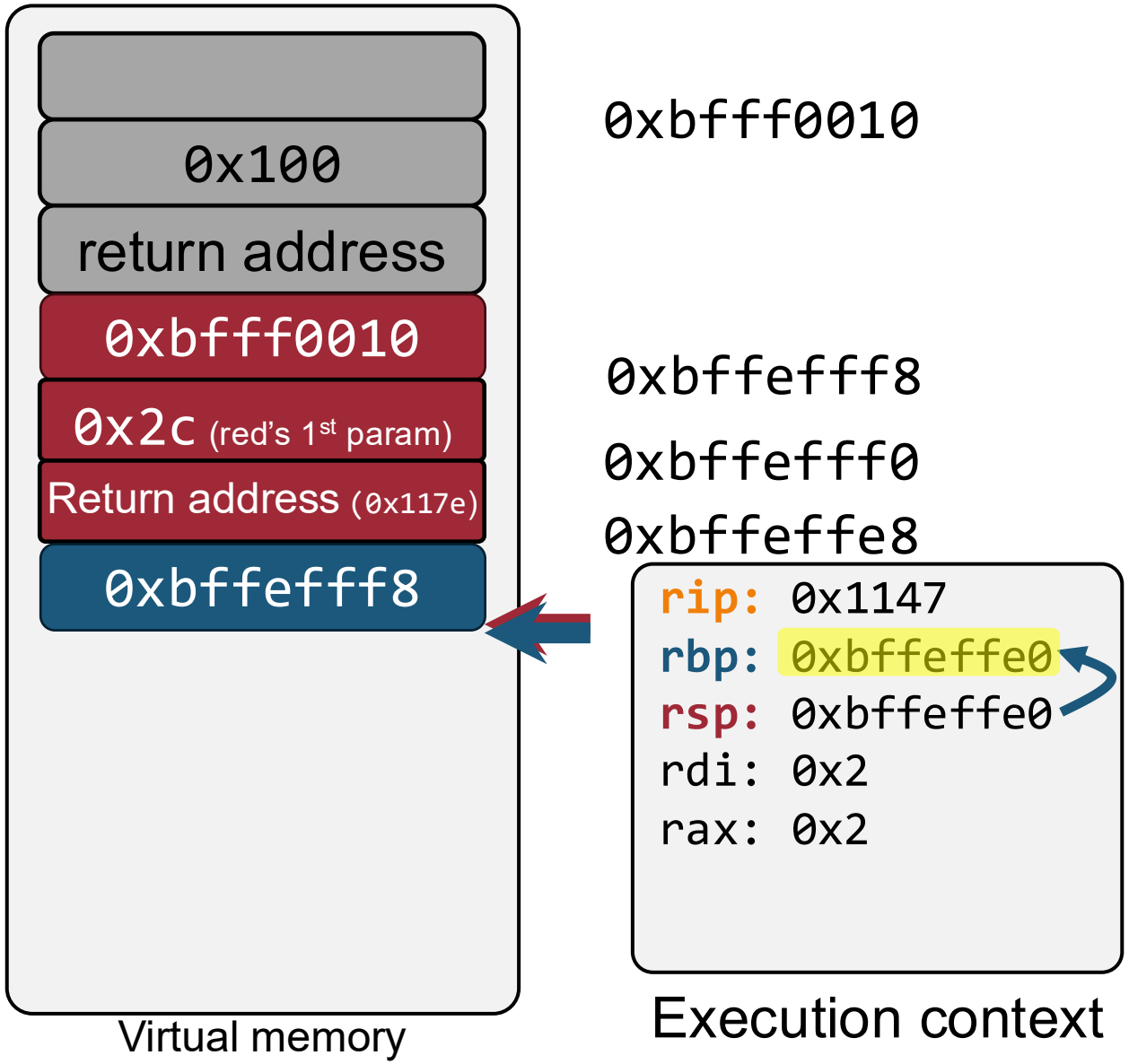
Execution Example




```
<red>:
   1166: push rbp
   1167: mov  rbp, rsp
   116a: sub  rsp, 0x8
   116e: mov  QWORD PTR [rbp-0x8], rdi
   1172: mov  rax, QWORD PTR [rbp-0x8]
   1176: sub  rax, 0x2a
   117a: mov  rdi, rax
   117d: call blue
   117e: mov  rdx, QWORD PTR [rbp-0x8]
   1182: add  rax, rdx
   1185: leave
   1186: ret

<blue>:
   1143: push   rbp
   1144: mov    rbp, rsp
   ...
   1160: leave
   1161: ret
```

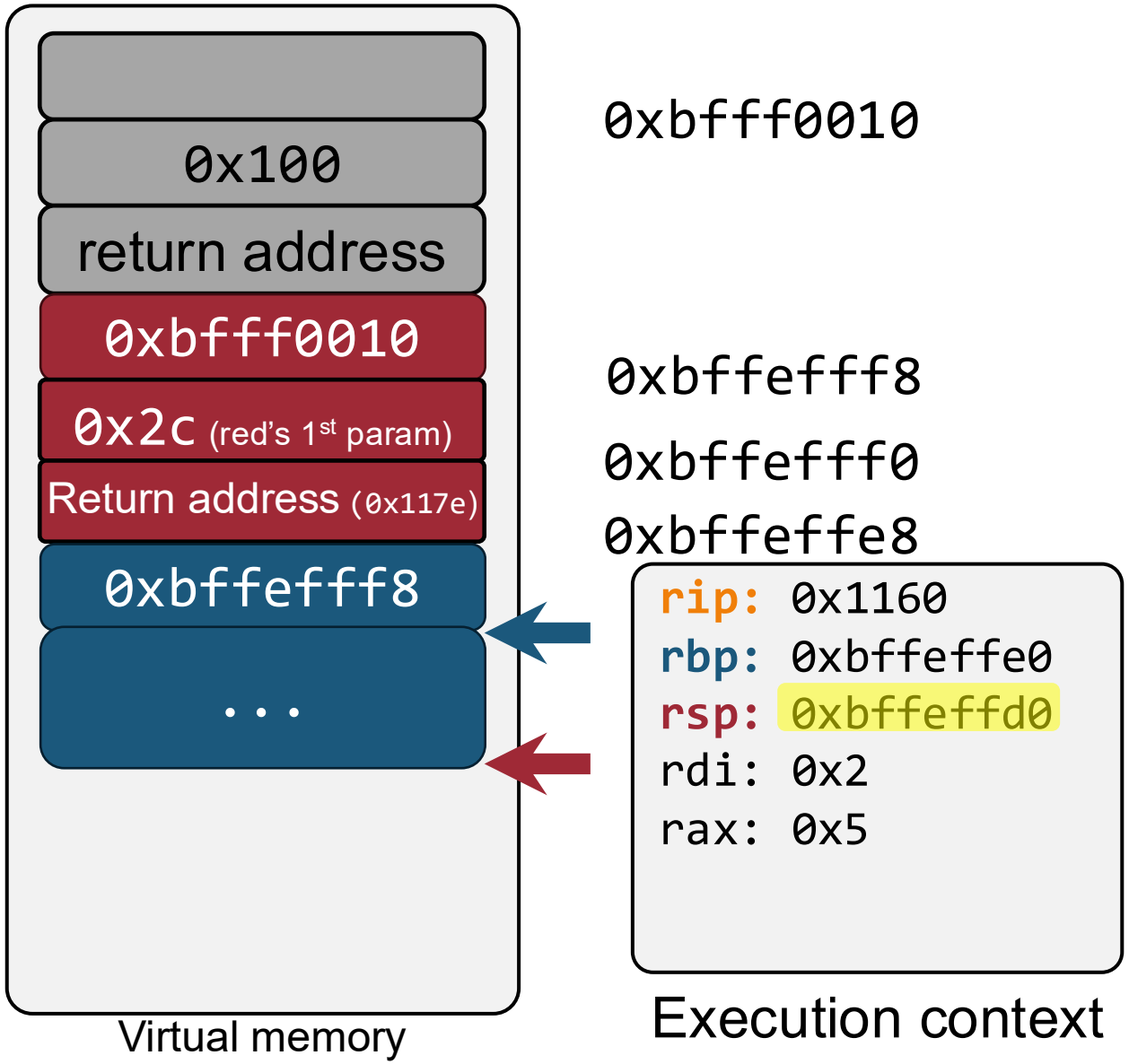
Execution Example




```
<red>:
   1166: push rbp
   1167: mov  rbp, rsp
   116a: sub  rsp, 0x8
   116e: mov  QWORD PTR [rbp-0x8], rdi
   1172: mov  rax, QWORD PTR [rbp-0x8]
   1176: sub  rax, 0x2a
   117a: mov  rdi, rax
   117d: call blue
   117e: mov  rdx, QWORD PTR [rbp-0x8]
   1182: add  rax, rdx
   1185: leave
   1186: ret

<blue>:
   1143: push   rbp
   1144: mov    rbp, rsp
   ...
   1160: leave
   1161: ret
```

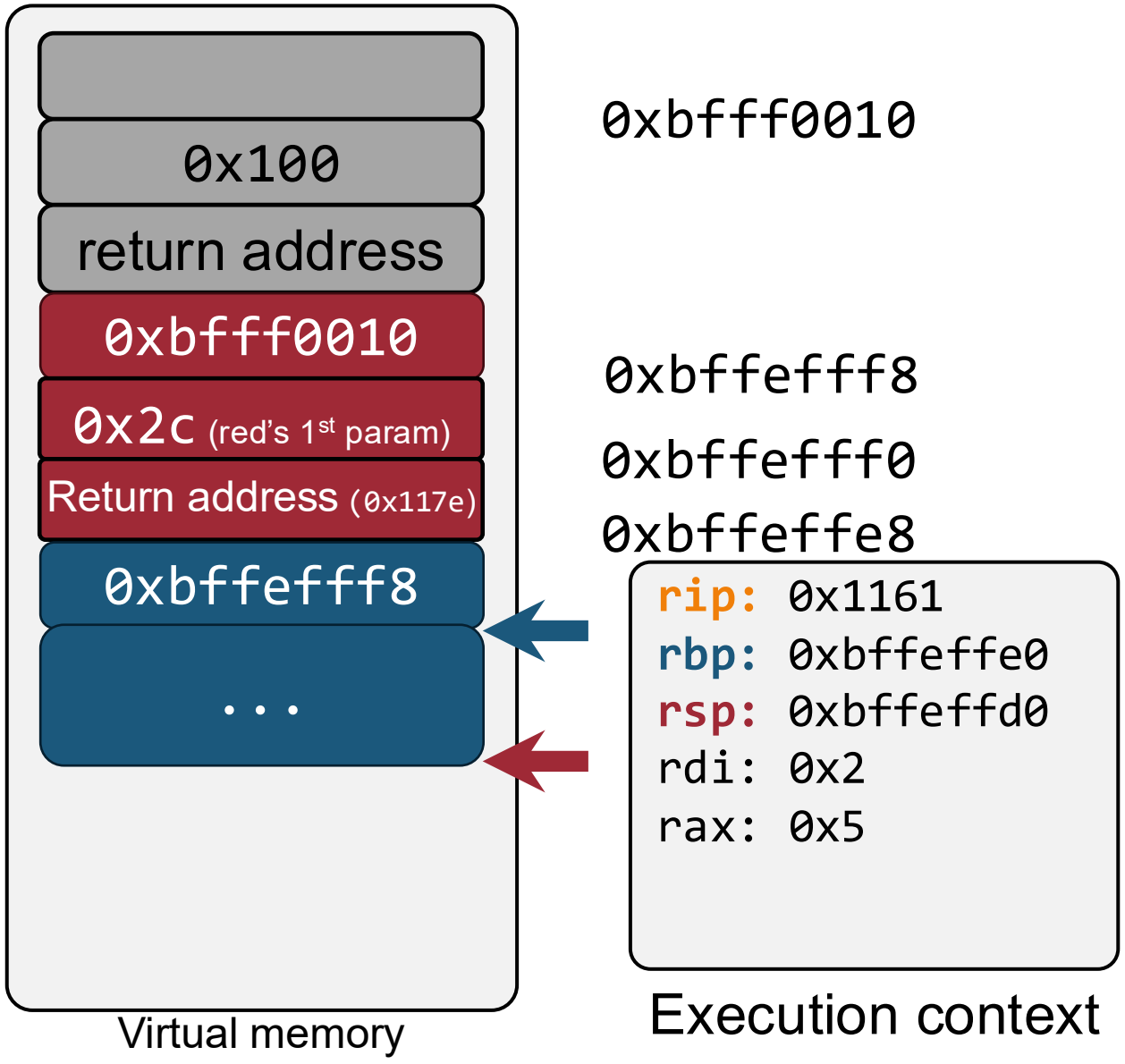
Execution Example




```
<red>:
  1166: push rbp
  1167: mov  rbp, rsp
  116a: sub  rsp, 0x8
  116e: mov  QWORD PTR [rbp-0x8], rdi
  1172: mov  rax, QWORD PTR [rbp-0x8]
  1176: sub  rax, 0x2a
  117a: mov  rdi, rax
  117d: call blue
  117e: mov  rdx, QWORD PTR [rbp-0x8]
  1182: add  rax, rdx
  1185: leave
  1186: ret

<blue>:
  1143: push   rbp
  1144: mov    rbp, rsp
  ...                    Skip!
  1160: leave
  1161: ret
```

Execution Example

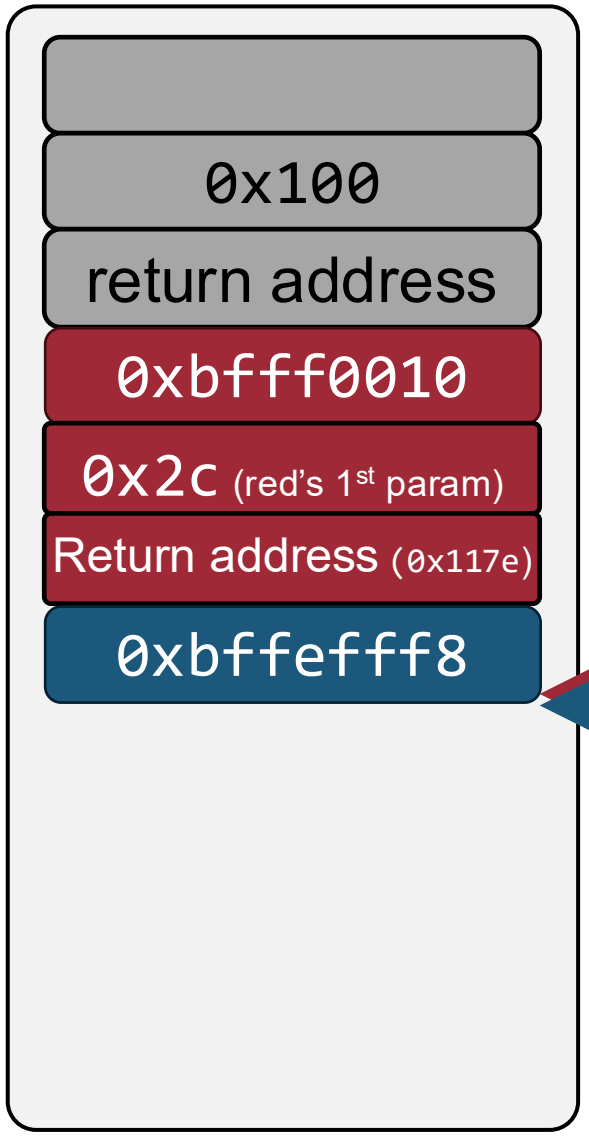



```
<red>:
   1166: push rbp
   1167: mov  rbp, rsp
   116a: sub  rsp, 0x8
   116e: mov  QWORD PTR [rbp-0x8], rdi
   1172: mov  rax, QWORD PTR [rbp-0x8]
   1176: sub  rax, 0x2a
   117a: mov  rdi, rax
   117d: call blue
   117e: mov  rdx, QWORD PTR [rbp-0x8]
   1182: add  rax, rdx
   1185: leave
   1186: ret

<blue>:
   1143: push   rbp
   1144: mov    rbp, rsp
   ...
   1160: leave
   1161: ret
```

mov rsp, rbp
pop rbp

Execution Example



0xbfff0010

0xbffefff8

0xbffefff0

0xbffeffe8

```
rip: 0x1161
rbp: 0xbffeffe0
rsp: 0xbffeffe0
rdi: 0x2
rax: 0x5
```

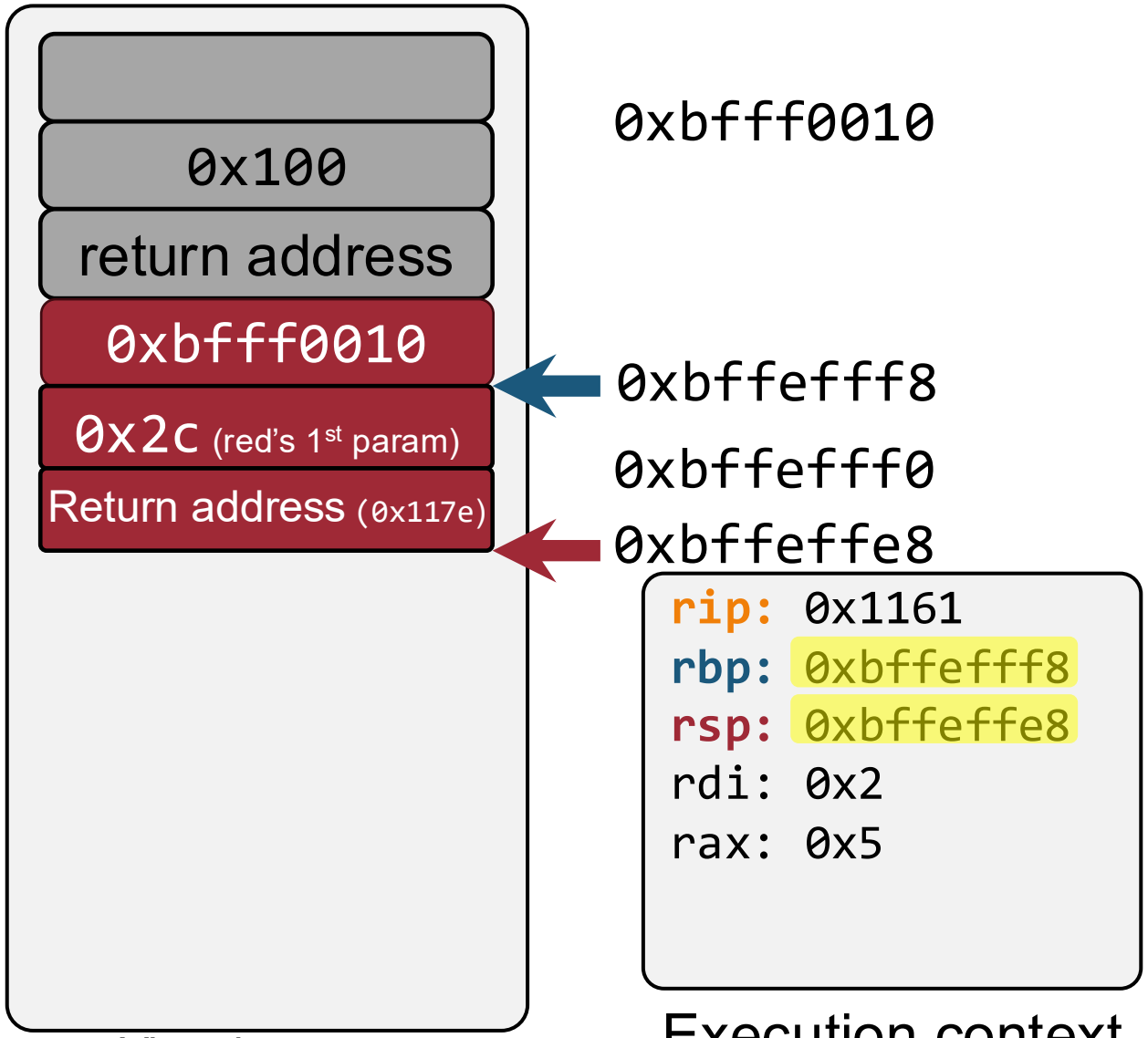
Execution context

```
<red>:
   1166: push rbp
   1167: mov  rbp, rsp
   116a: sub  rsp, 0x8
   116e: mov  QWORD PTR [rbp-0x8], rdi
   1172: mov  rax, QWORD PTR [rbp-0x8]
   1176: sub  rax, 0x2a
   117a: mov  rdi, rax
   117d: call blue
   117e: mov  rdx, QWORD PTR [rbp-0x8]
   1182: add  rax, rdx
   1185: leave
   1186: ret

<blue>:
   1143: push   rbp
   1144: mov    rbp, rsp
   ...
   1160: leave
   1161: ret
```

mov rsp, rbp
pop rbp

Execution Example

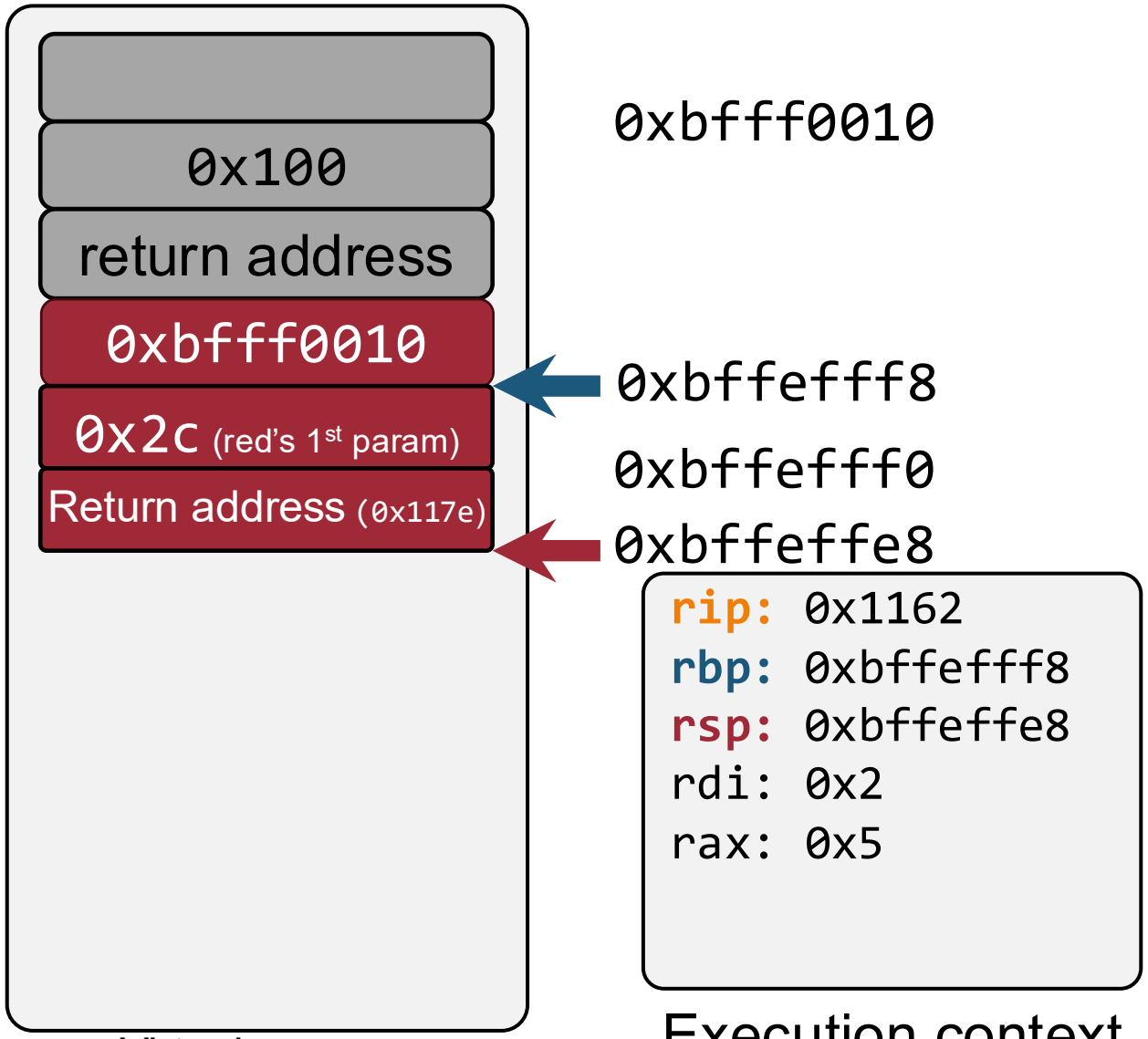



```
<red>:
   1166: push rbp
   1167: mov  rbp, rsp
   116a: sub  rsp, 0x8
   116e: mov  QWORD PTR [rbp-0x8], rdi
   1172: mov  rax, QWORD PTR [rbp-0x8]
   1176: sub  rax, 0x2a
   117a: mov  rdi, rax
   117d: call blue
   117e: mov  rdx, QWORD PTR [rbp-0x8]
   1182: add  rax, rdx
   1185: leave
   1186: ret

<blue>:
   1143: push   rbp
   1144: mov    rbp, rsp
   ...
   1160: leave
   1161: ret
```

mov rsp, rbp
pop rbp

Execution Example

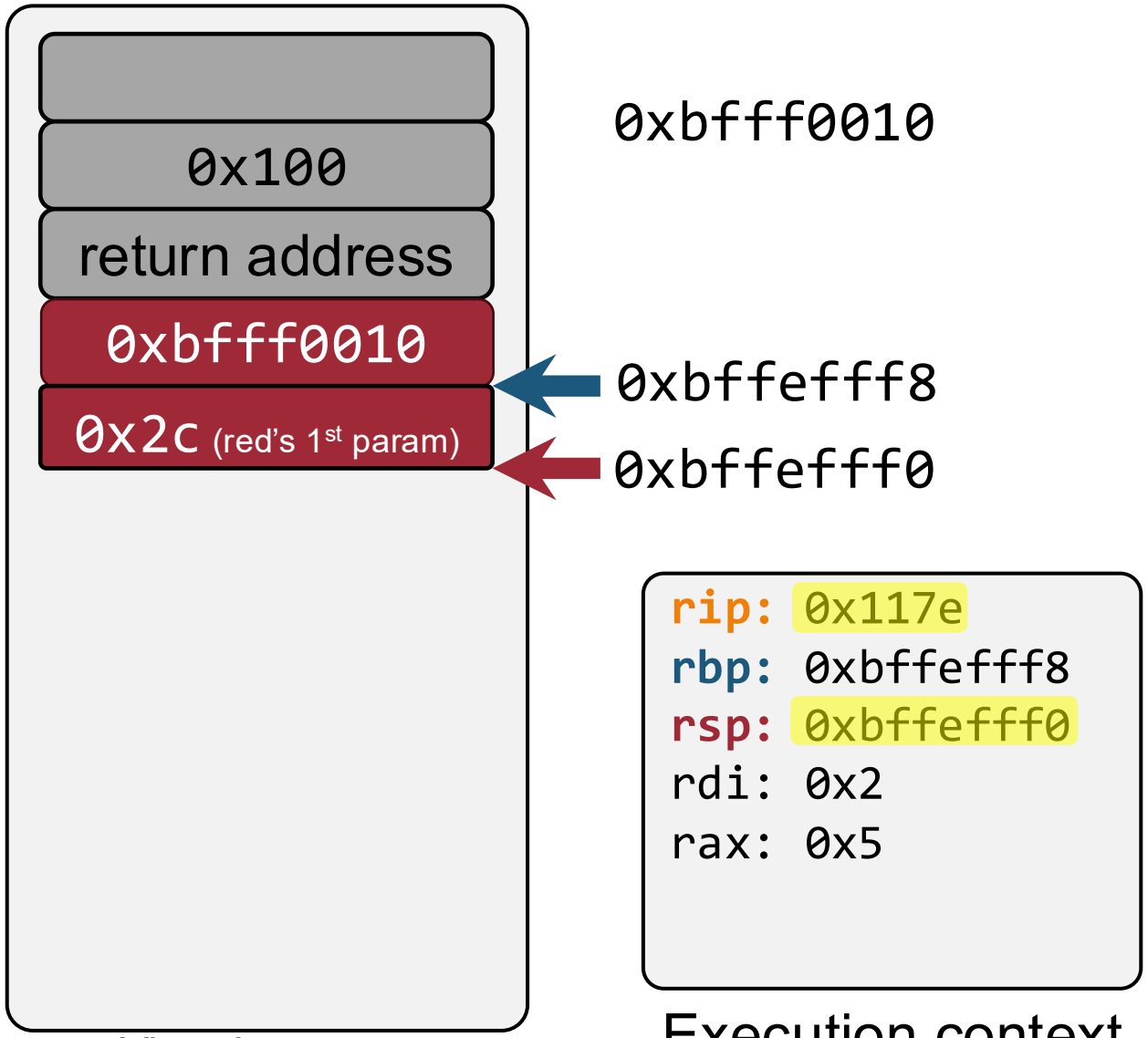



```
<red>:
  1166: push rbp
  1167: mov  rbp, rsp
  116a: sub  rsp, 0x8
  116e: mov  QWORD PTR [rbp-0x8], rdi
  1172: mov  rax, QWORD PTR [rbp-0x8]
  1176: sub  rax, 0x2a
  117a: mov  rdi, rax
  117d: call blue
  117e: mov  rdx, QWORD PTR [rbp-0x8]
  1182: add  rax, rdx
  1185: leave
  1186: ret

<blue>:
  1143: push   rbp
  1144: mov    rbp, rsp
  ...
  1160: leave
  1161: ret
```

pop rip

Execution Example




```
rip: 0x117e
rbp: 0xbffefff8
rsp: 0xbffefff0
rdi: 0x2
rax: 0x5
```

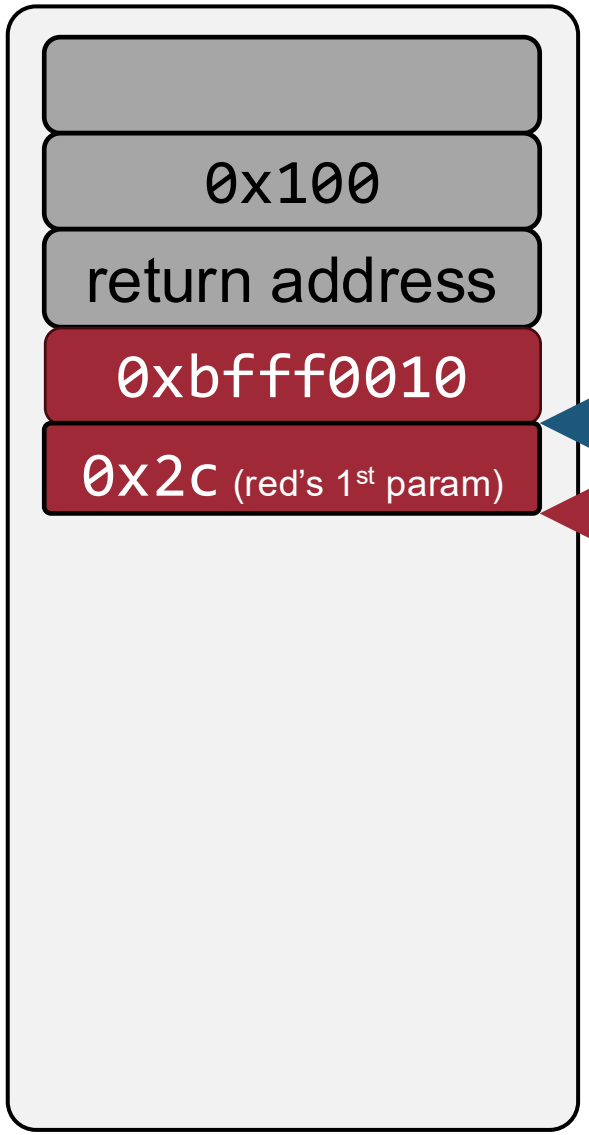
Execution context

```
<red>:
  1166: push rbp
  1167: mov  rbp, rsp
  116a: sub  rsp, 0x8
  116e: mov  QWORD PTR [rbp-0x8], rdi
  1172: mov  rax, QWORD PTR [rbp-0x8]
  1176: sub  rax, 0x2a
  117a: mov  rdi, rax
  117d: call blue
  117e: mov  rdx, QWORD PTR [rbp-0x8]
  1182: add  rax, rdx
  1185: leave
  1186: ret

<blue>:
  1143: push   rbp
  1144: mov    rbp, rsp
  ...
  1160: leave
  1161: ret
```

pop rip

Execution Example



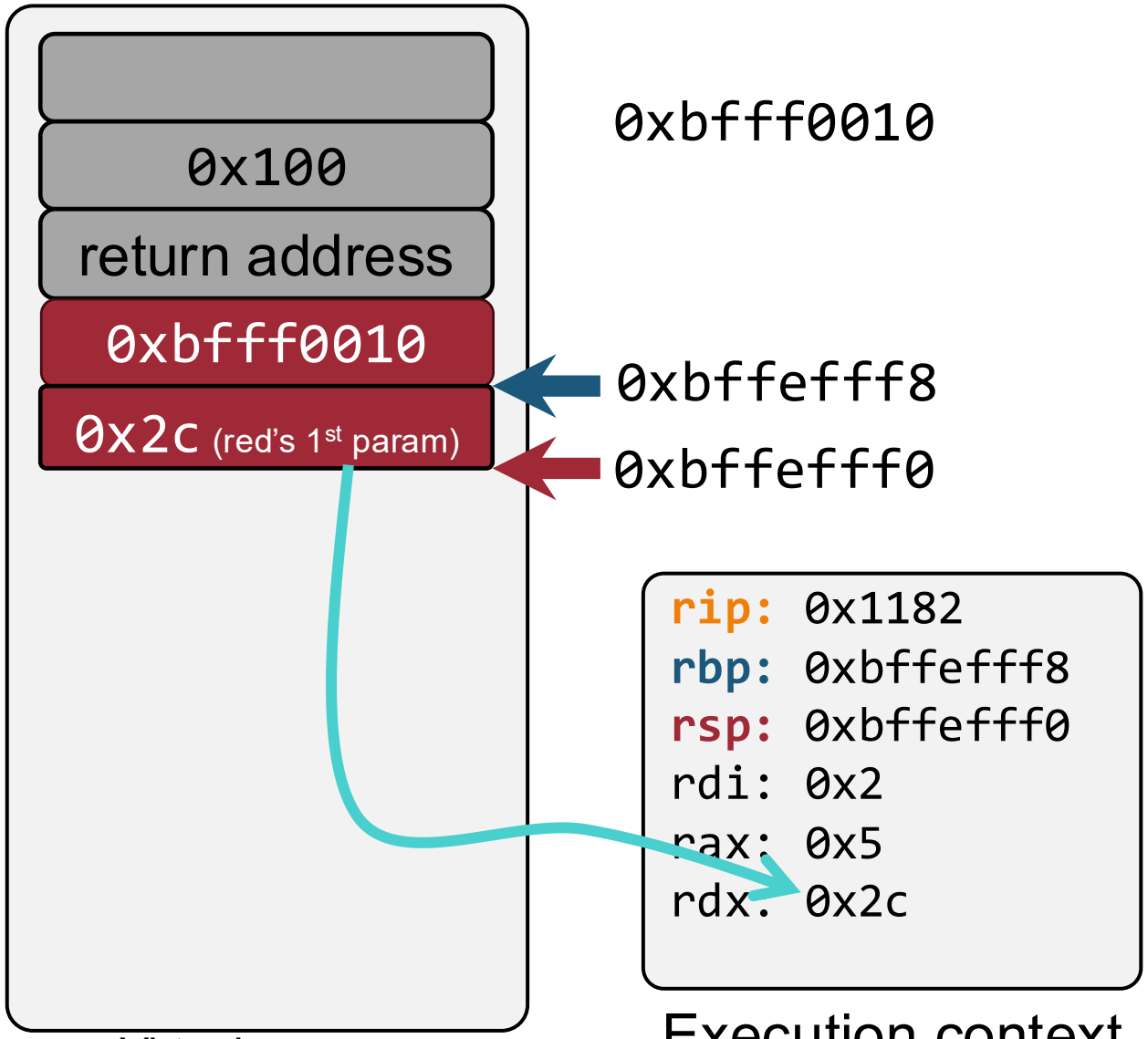

```
rip: 0x1182
rbp: 0xbffefff8
rsp: 0xbffefff0
rdi: 0x2
rax: 0x5
```

Execution context

```
<red>:
  1166: push rbp
  1167: mov  rbp, rsp
  116a: sub  rsp, 0x8
  116e: mov  QWORD PTR [rbp-0x8], rdi
  1172: mov  rax, QWORD PTR [rbp-0x8]
  1176: sub  rax, 0x2a
  117a: mov  rdi, rax
  117d: call blue
  117e: mov  rdx, QWORD PTR [rbp-0x8]
  1182: add  rax, rdx
  1185: leave
  1186: ret

<blue>:
  1143: push   rbp
  1144: mov    rbp, rsp
  ...
  1160: leave
  1161: ret
```

Execution Example



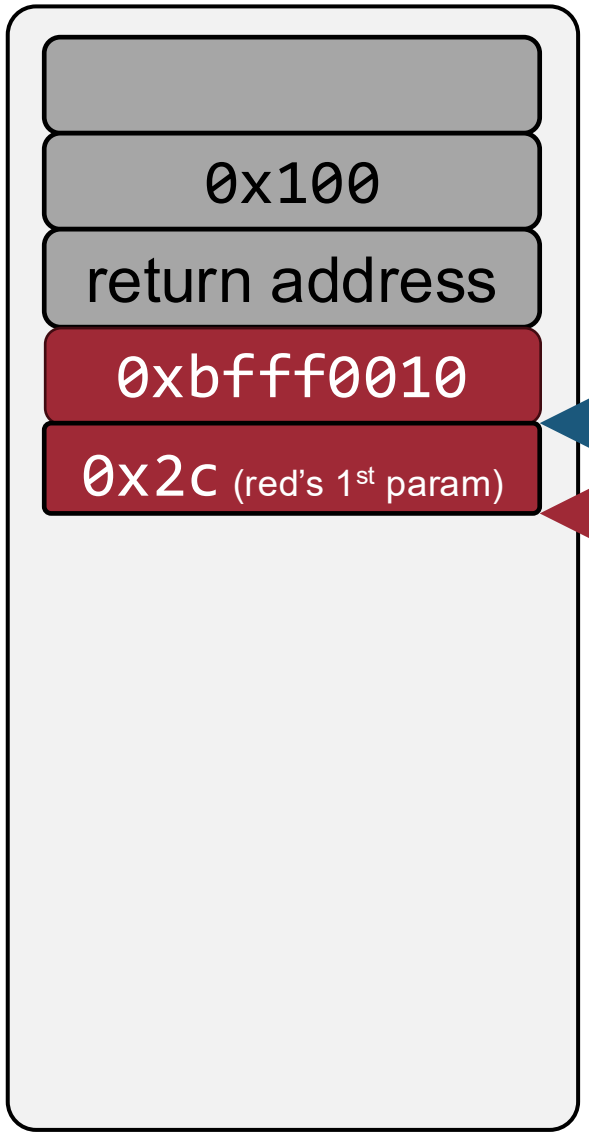

```
rip: 0x1182
rbp: 0xbffefff8
rsp: 0xbffefff0
rdi: 0x2
rax: 0x5
rdx: 0x2c
```

Execution context

```
<red>:
  1166: push rbp
  1167: mov  rbp, rsp
  116a: sub  rsp, 0x8
  116e: mov  QWORD PTR [rbp-0x8], rdi
  1172: mov  rax, QWORD PTR [rbp-0x8]
  1176: sub  rax, 0x2a
  117a: mov  rdi, rax
  117d: call blue
  117e: mov  rdx, QWORD PTR [rbp-0x8]
  1182: add  rax, rdx
  1185: leave
  1186: ret

<blue>:
  1143: push   rbp
  1144: mov    rbp, rsp
  ...
  1160: leave
  1161: ret
```

Execution Example



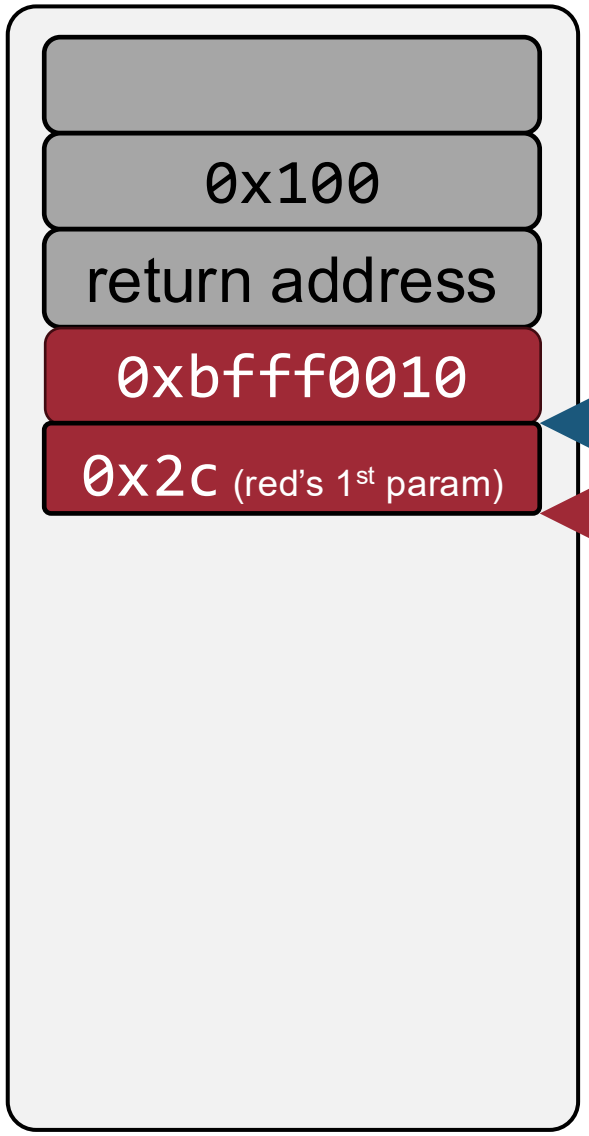

```
rip: 0x1185
rbp: 0xbffefff8
rsp: 0xbffefff0
rdi: 0x2
rax: 0x5
rdx: 0x2c
```

Execution context

```
<red>:
  1166: push rbp
  1167: mov  rbp, rsp
  116a: sub  rsp, 0x8
  116e: mov  QWORD PTR [rbp-0x8], rdi
  1172: mov  rax, QWORD PTR [rbp-0x8]
  1176: sub  rax, 0x2a
  117a: mov  rdi, rax
  117d: call blue
  117e: mov  rdx, QWORD PTR [rbp-0x8]
  1182: add  rax, rdx
  1185: leave
  1186: ret

<blue>:
  1143: push   rbp
  1144: mov    rbp, rsp
  ...
  1160: leave
  1161: ret
```

Execution Example




```
rip: 0x1185
rbp: 0xbffefff8
rsp: 0xbffefff0
rdi: 0x2
rax: 0x31
rdx: 0x2c
```

Execution context

rax stores the return value

```
long red(long a1) {
    return blue(a1 - 42) + a1;
}
```

```
<red>:
  1166: push rbp
  116e:      QWORD PTR [rbp-0x8], rdi
  1172:      QWORD PTR [rbp-0x8]
             x2a
             ax
  117e: mov  rdx, QWORD PTR [rbp-0x8]
  1182: add  rax, rdx
  1185: leave
  1186: ret

<blue>:
  1143: push   rbp
  1144: mov    rbp, rsp
  ...
  1160: leave
  1161: ret
```

Question?