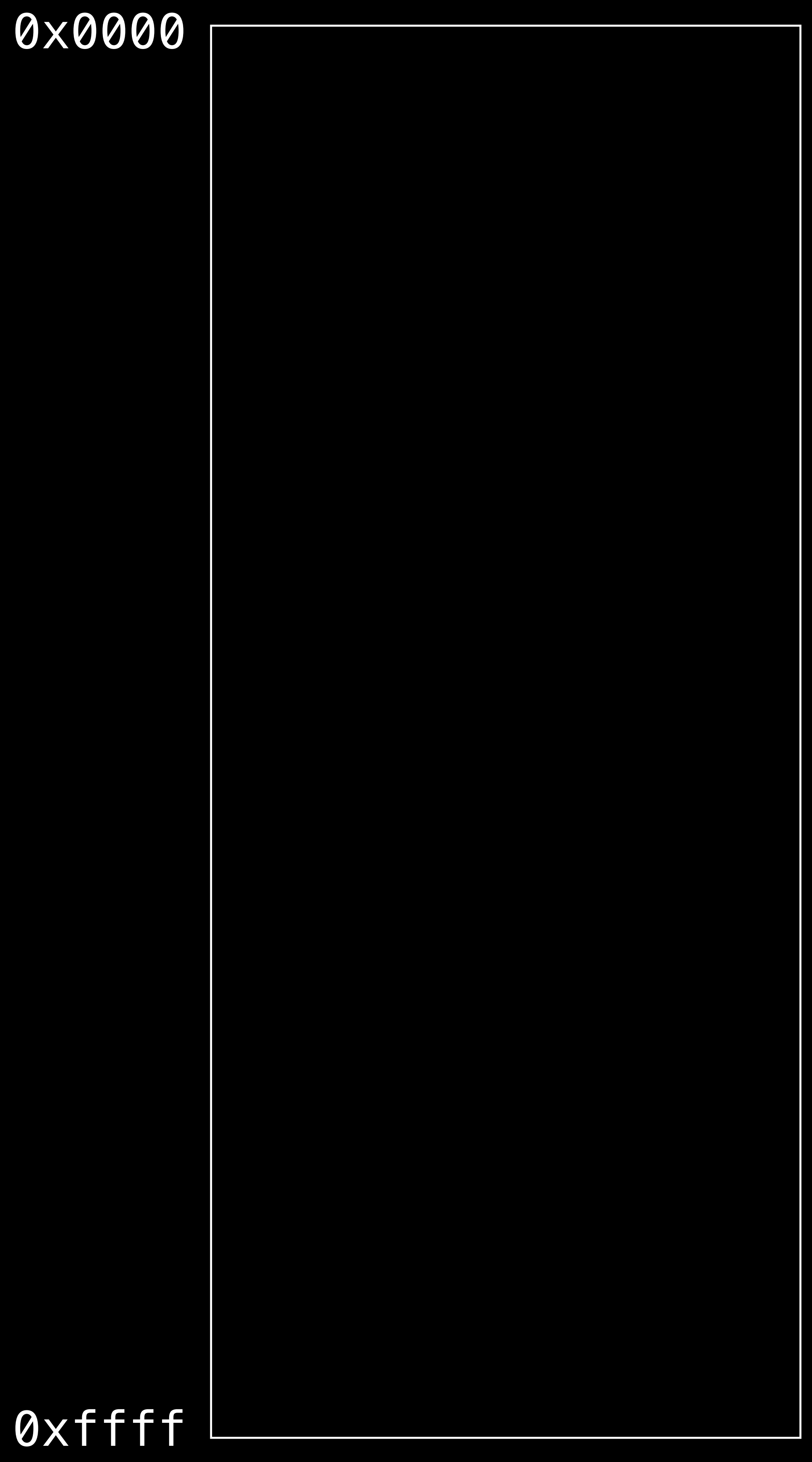
Vom buffer overflow zur shell

hanemile - GPN21 - 2023-06-09

Memory

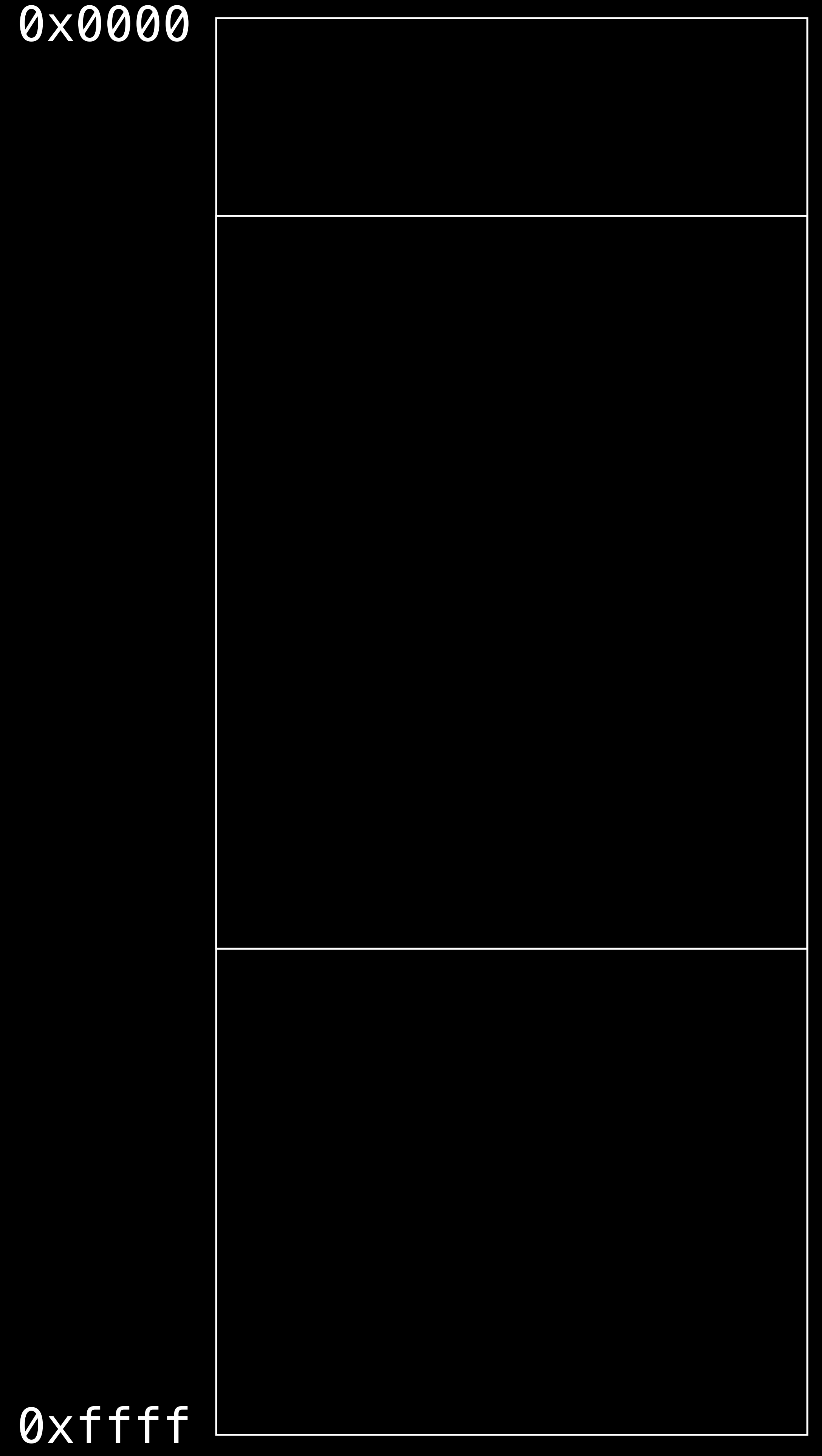
Memory - Binary layout

Where’s everything located?



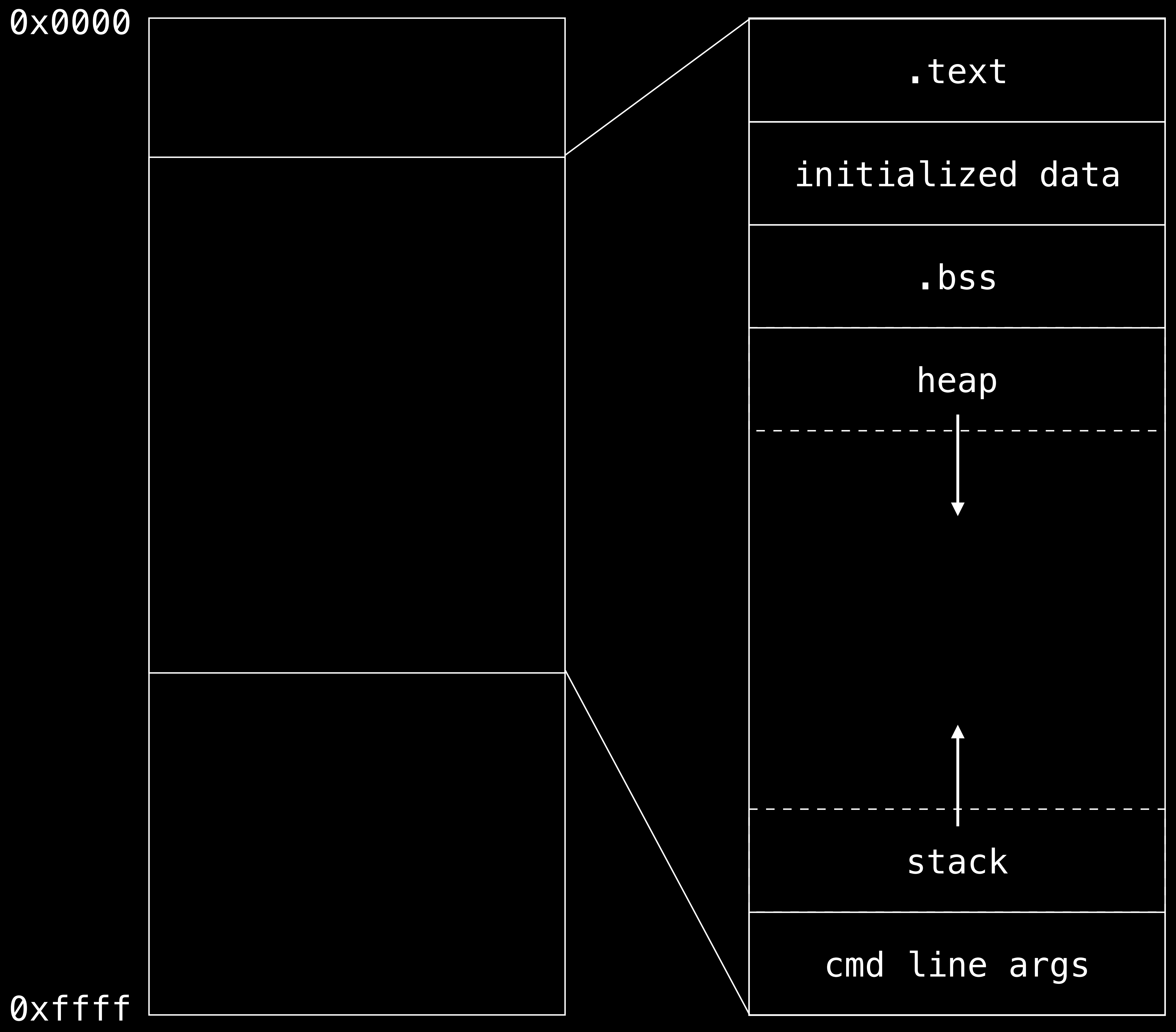
Memory - Binary layout

Where’s everything located?



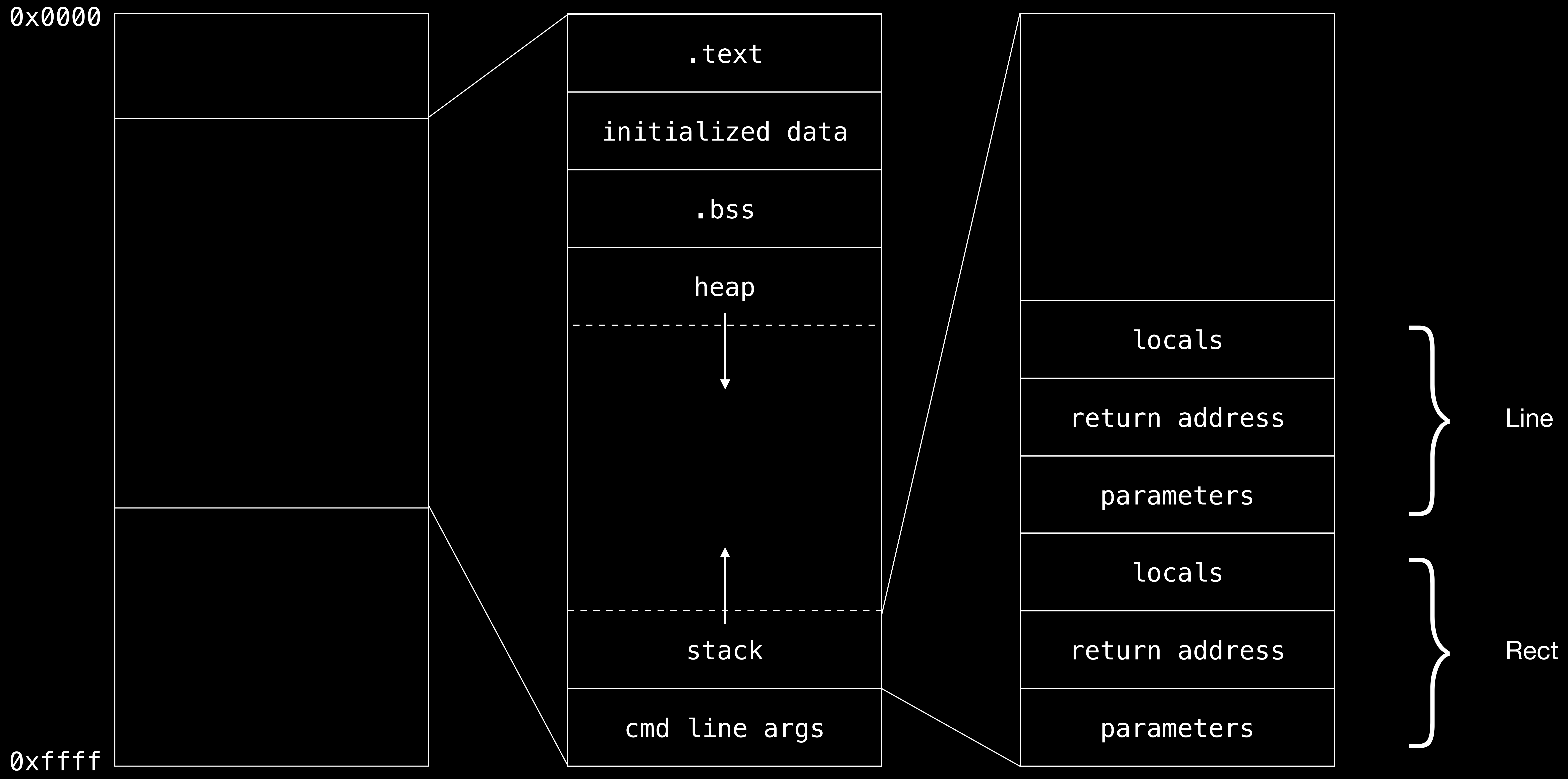
Memory - Binary layout

Where's everything located?



Memory - Binary layout

Where’s everything located?

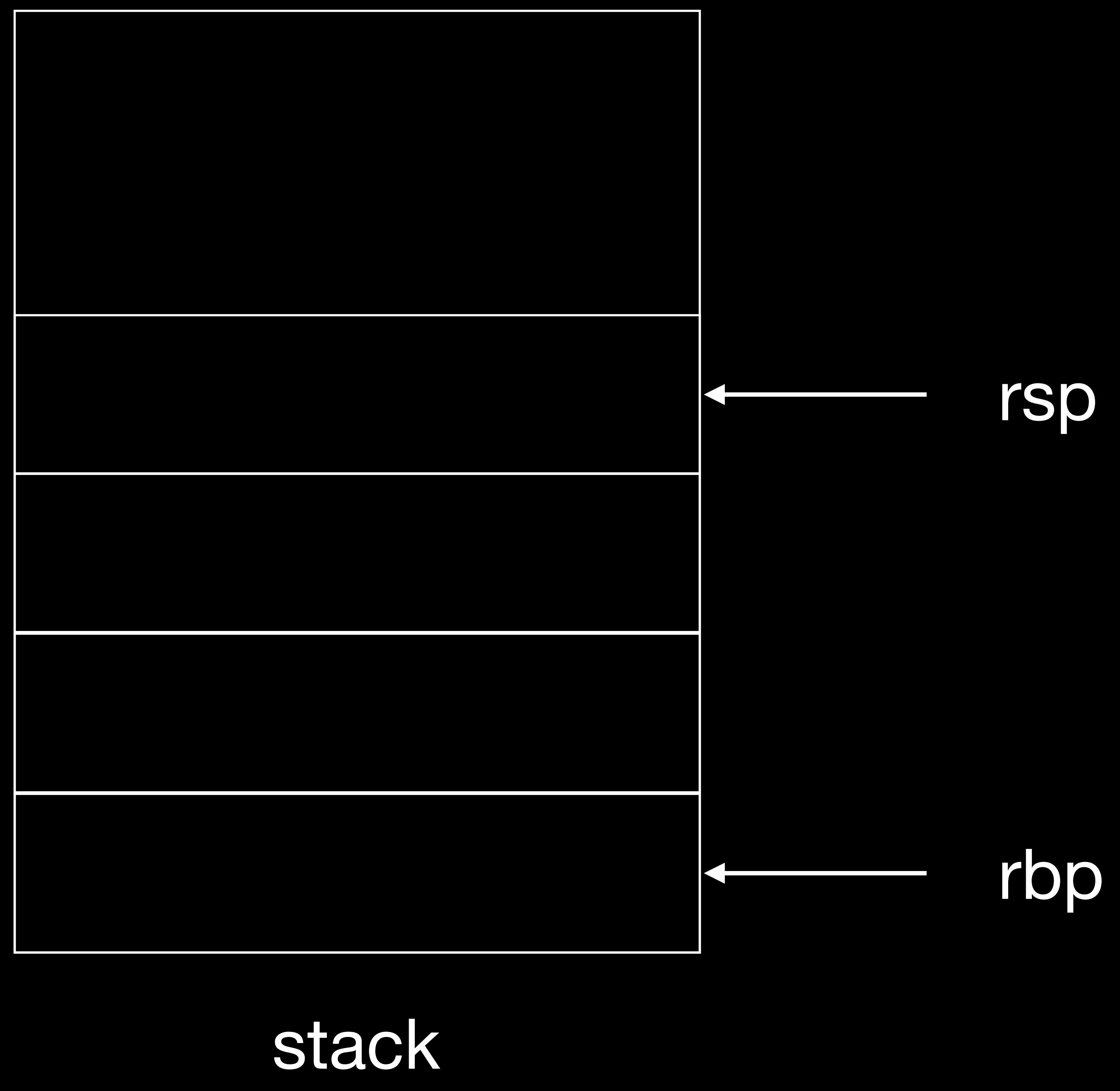


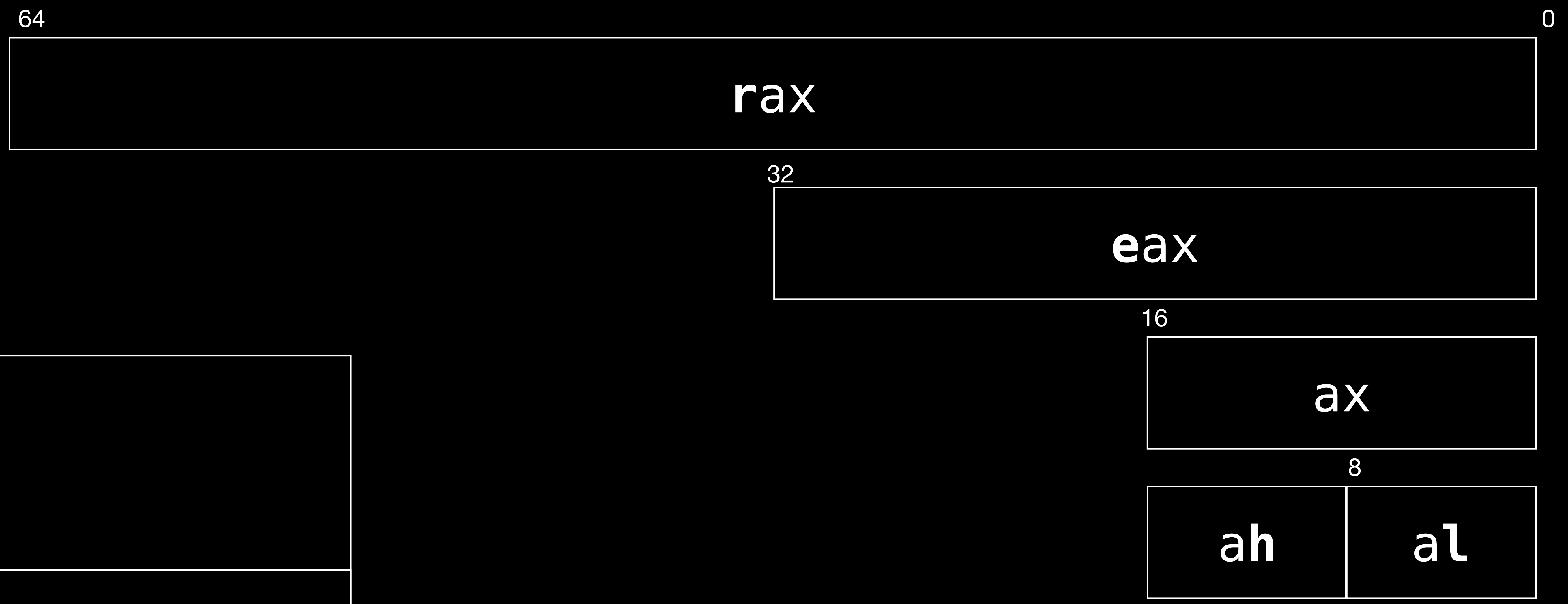
Registers

Register - The basics

[…] a quickly accessible location available to a computer’s processors”

`rdi`	1st arg
`rsi`	2nd arg
`rdx`	3rd arg
`rcx`	4th arg
`rax`	return value
`rip`	next instr.

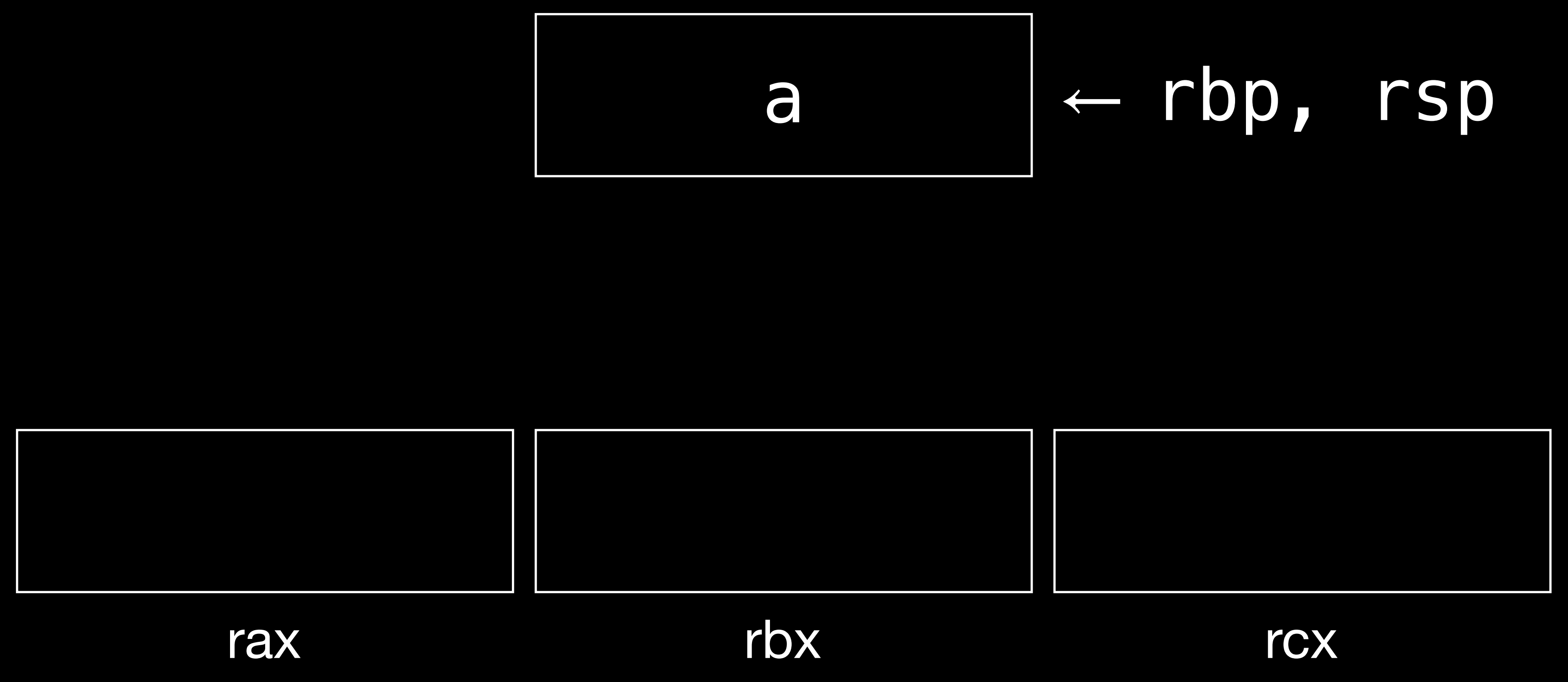




Stack

Stack

push

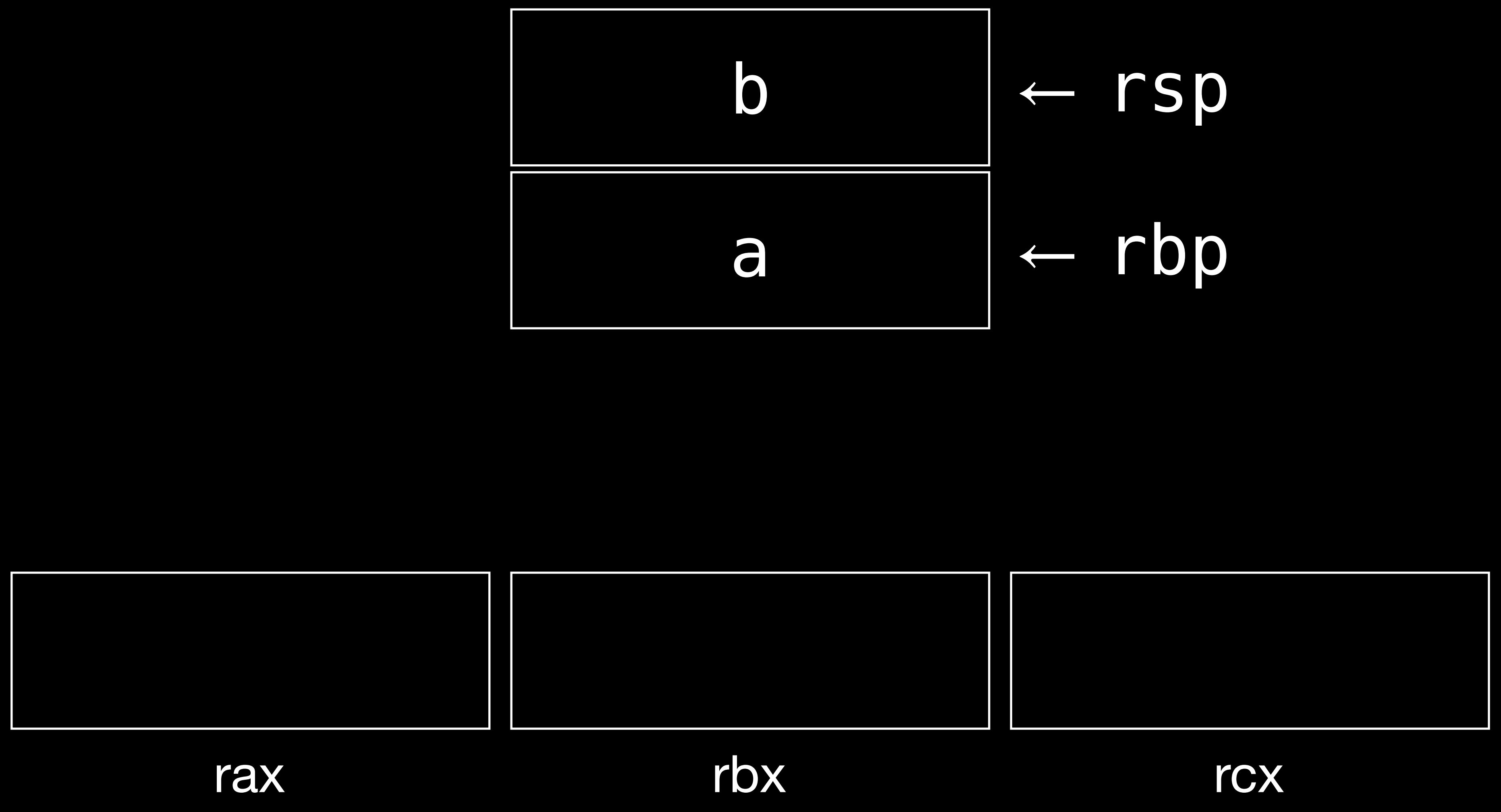


```
push a <—
push b
push c
pop rax
pop rbx
pop rcx
```

(rbp = Base Pointer, rsp = Stack Pointer)

Stack

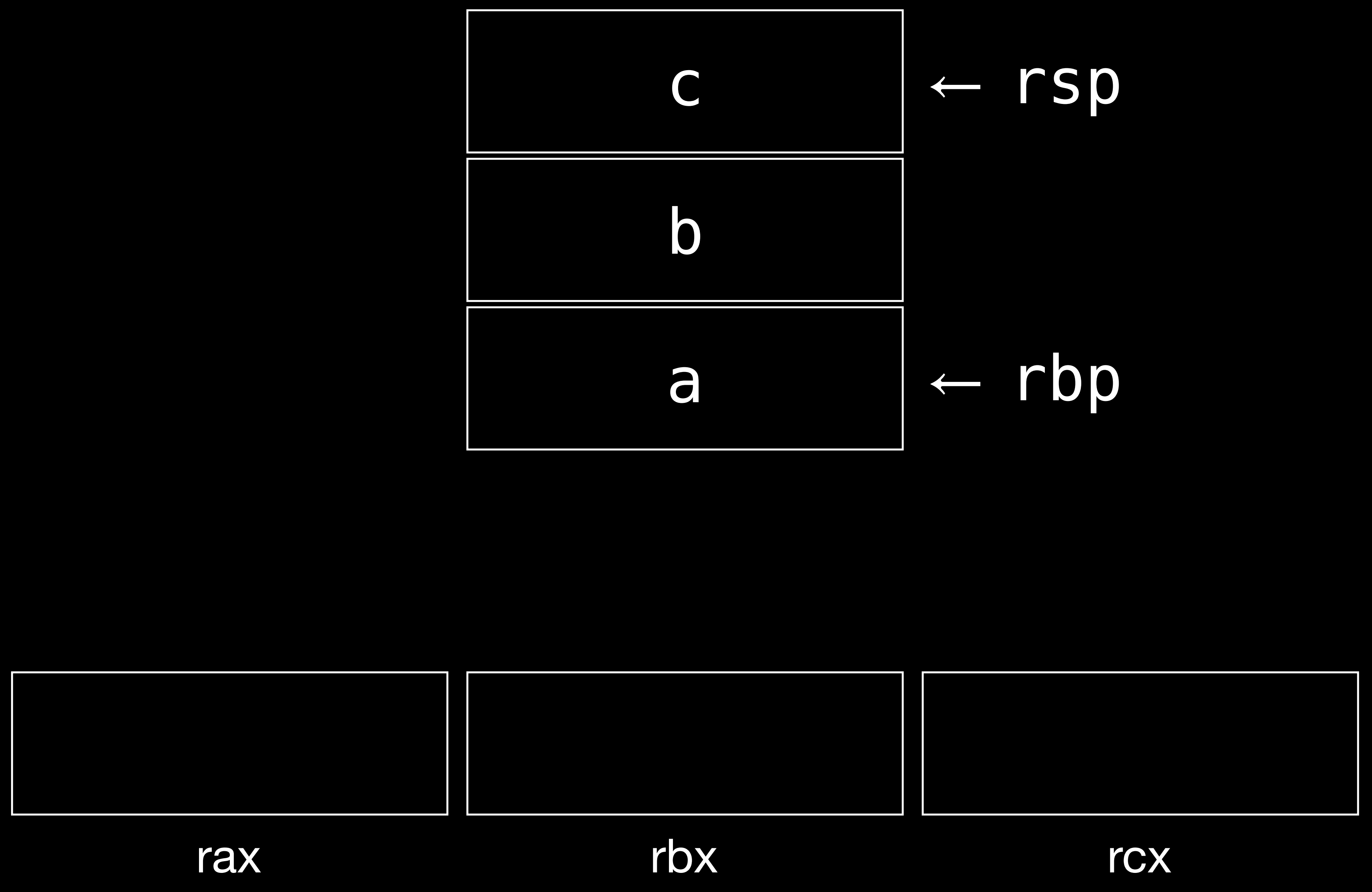
push



```
push a
push b <—
push c
pop rax
pop rbx
pop rcx
```

Stack

push



c ← rsp

b

a ← rbp

rax

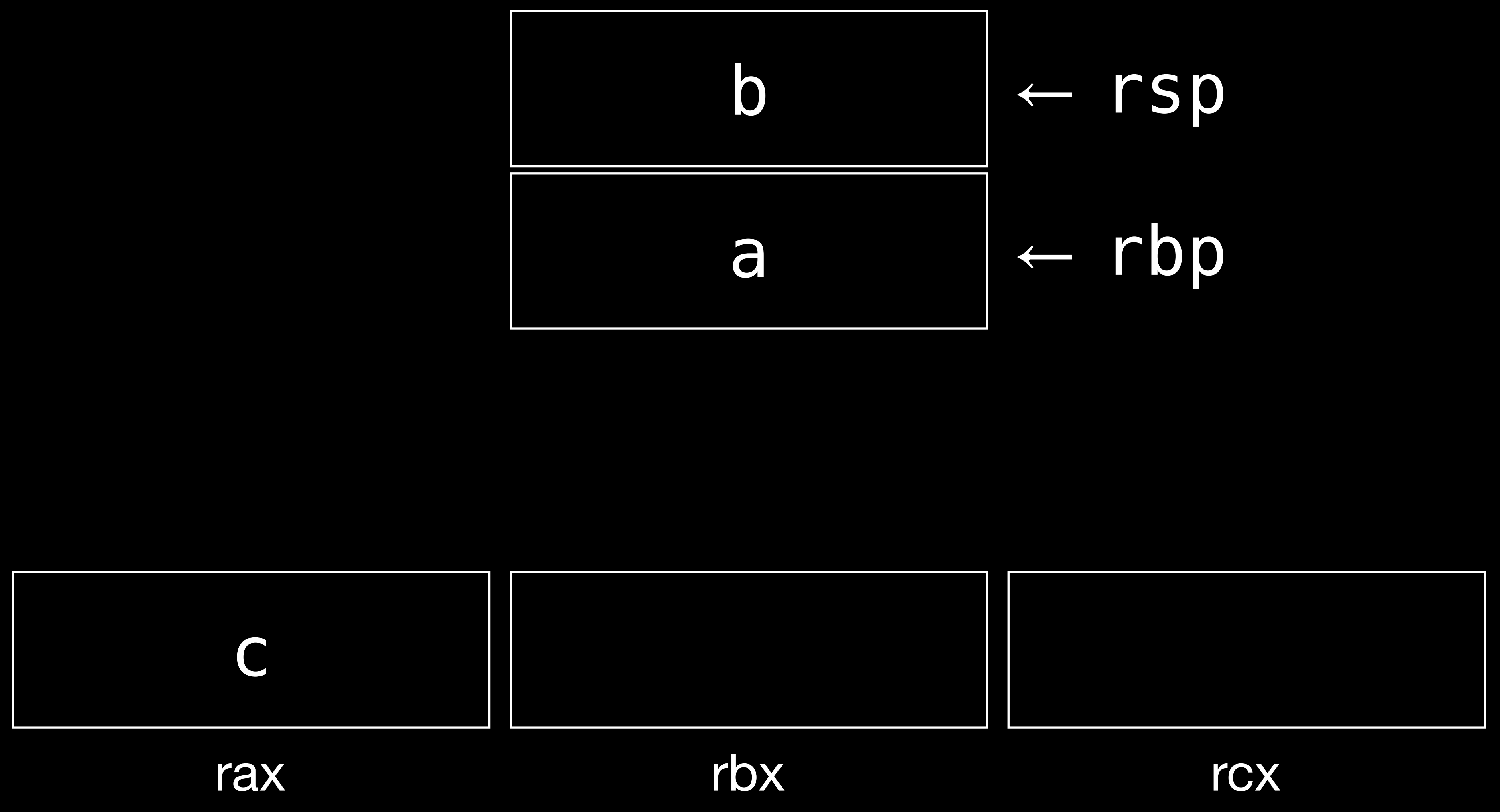
rbx

rcx

```
push a
push b
push c <–
pop rax
pop rbx
pop rcx
```

Stack

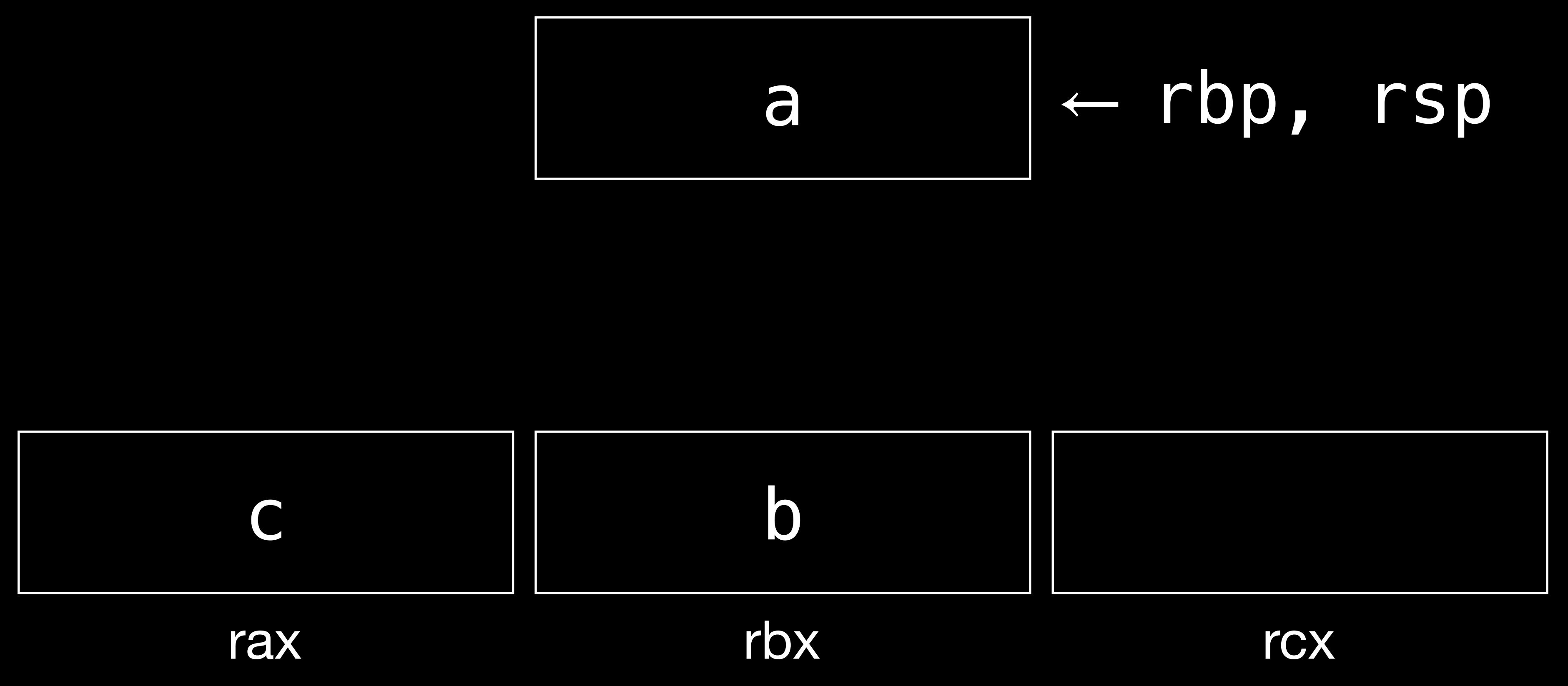
pop



```
push a
push b
push c
pop rax <-
pop rbx
pop rcx
```

Stack

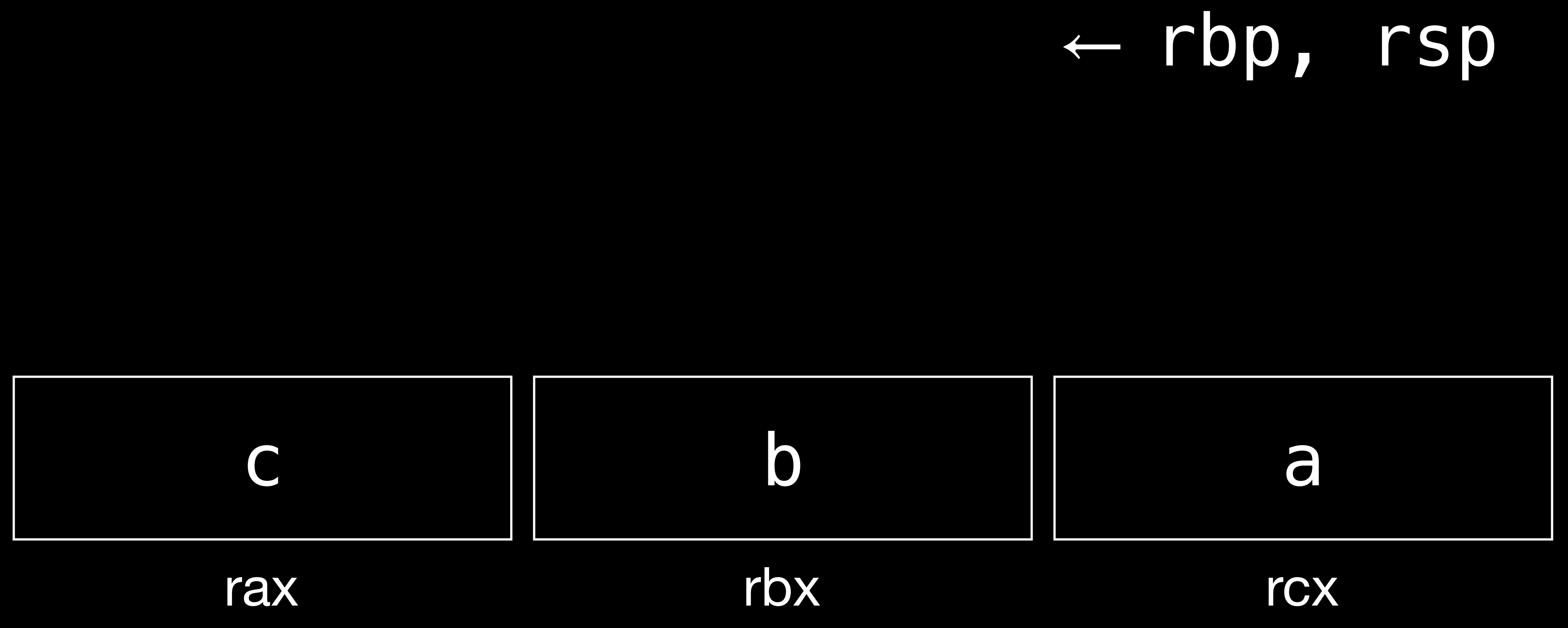
pop



```
push a
push b
push c
pop rax
pop rbx <–
pop rcx
```

Stack

pop



← rbp, rsp

c	b	a
rax	rbx	rcx

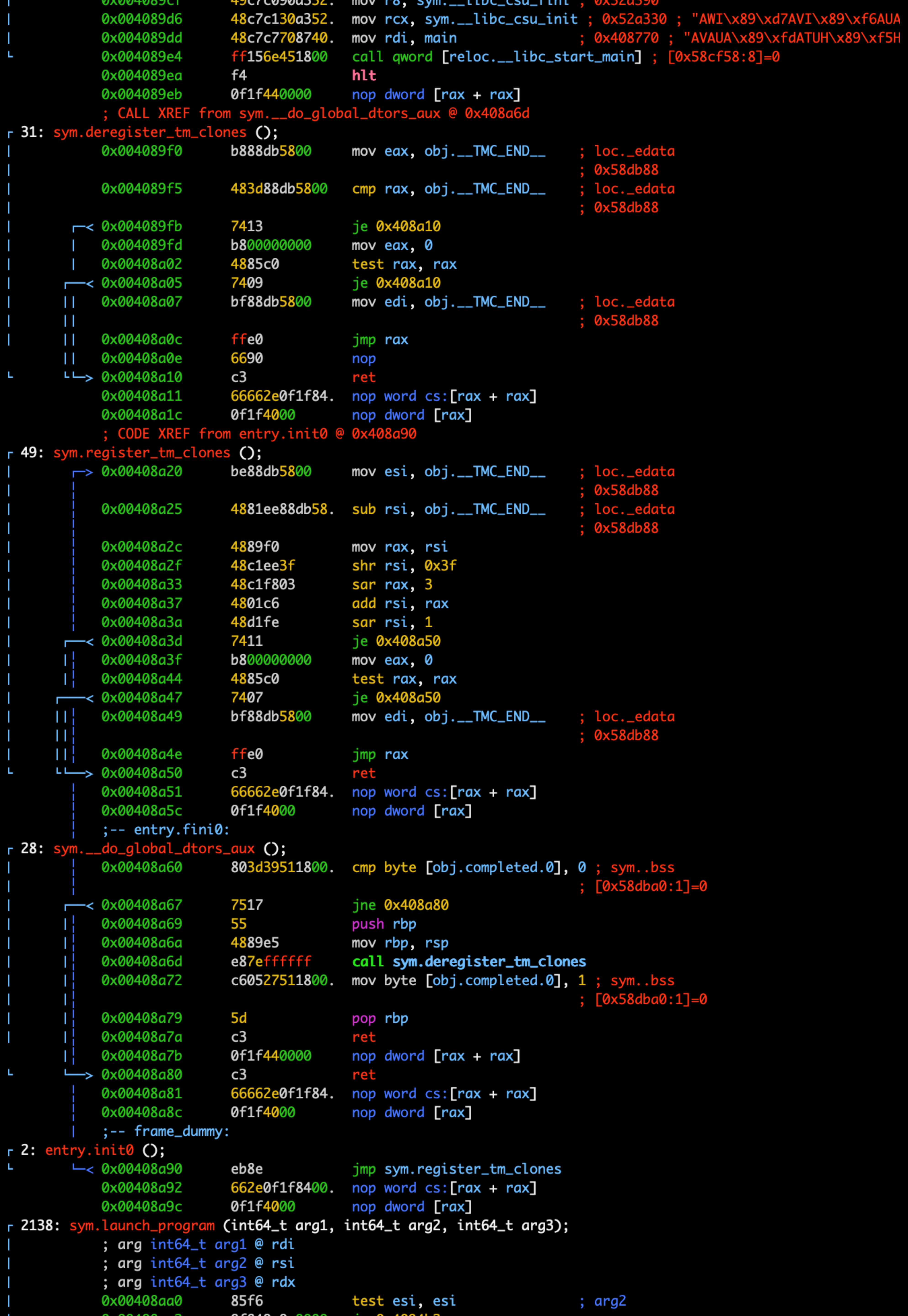
```
push a
push b
push c
pop rax
pop rbx
pop rcx <—
```

Assembly

Assembly

Low level programming

- opcodes
- mnemonics
- common mnemonics
    - push/pop
    - sub, add, mul, div
    - mov, lea
    - ret, leave, call
    - jmp, jne, jz
- syntax (Intel vs AT&T)

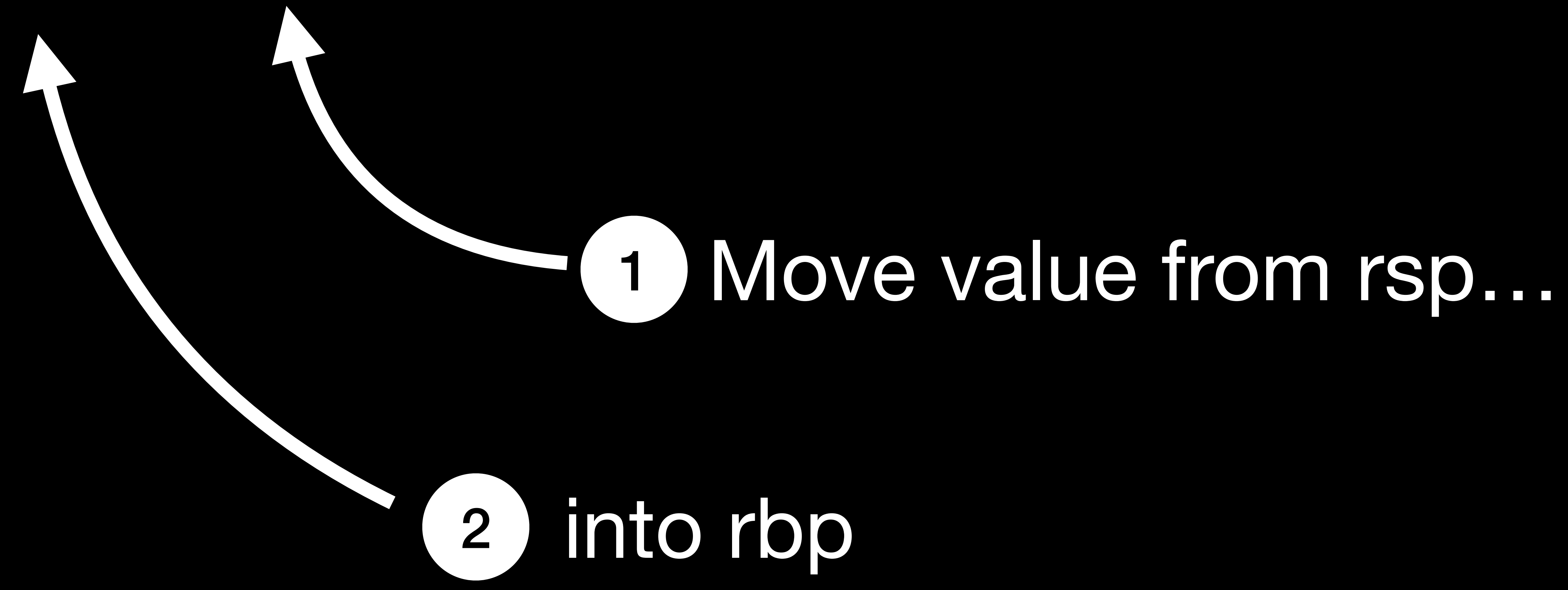


```
|           0x004089d6      48c7c130a352.  mov rcx, sym.__libc_csu_init ; 0x52a330 ; "AWI\x89\xd7AVI\x89\xf6AUA
|           0x004089dd      48c7c7708740.  mov rdi, main               ; 0x408770 ; "AVAUA\x89\xfdATUH\x89\xf5H
└           0x004089e4      ff156e451800   call qword [reloc.__libc_start_main] ; [0x58cf58:8]=0
            0x004089ea      f4             hlt
            0x004089eb      0f1f440000     nop dword [rax + rax]
            ; CALL XREF from sym.__do_global_dtors_aux @ 0x408a6d
┌ 31: sym.deregister_tm_clones ();
|           0x004089f0      b888db5800     mov eax, obj.__TMC_END__    ; loc._edata
|                                                                      ; 0x58db88
|           0x004089f5      483d88db5800   cmp rax, obj.__TMC_END__    ; loc._edata
|                                                                      ; 0x58db88
|       ┌─< 0x004089fb      7413           je 0x408a10
|       |   0x004089fd      b800000000     mov eax, 0
|       |   0x00408a02      4885c0         test rax, rax
|      ┌──< 0x00408a05      7409           je 0x408a10
|      ||   0x00408a07      bf88db5800     mov edi, obj.__TMC_END__    ; loc._edata
|      ||                                                              ; 0x58db88
|      ||   0x00408a0c      ffe0           jmp rax
|      ||   0x00408a0e      6690           nop
└      └└─> 0x00408a10      c3             ret
            0x00408a11      66662e0f1f84.  nop word cs:[rax + rax]
            0x00408a1c      0f1f4000       nop dword [rax]
            ; CODE XREF from entry.init0 @ 0x408a90
┌ 49: sym.register_tm_clones ();
|       ┌─> 0x00408a20      be88db5800     mov esi, obj.__TMC_END__    ; loc._edata
|       |                                                              ; 0x58db88
|       |   0x00408a25      4881ee88db58.  sub rsi, obj.__TMC_END__    ; loc._edata
|       |                                                              ; 0x58db88
|       |   0x00408a2c      4889f0         mov rax, rsi
|       |   0x00408a2f      48c1ee3f       shr rsi, 0x3f
|       |   0x00408a33      48c1f803       sar rax, 3
|       |   0x00408a37      4801c6         add rsi, rax
|       |   0x00408a3a      48d1fe         sar rsi, 1
|      ┌──< 0x00408a3d      7411           je 0x408a50
|      ||   0x00408a3f      b800000000     mov eax, 0
|      ||   0x00408a44      4885c0         test rax, rax
|     ┌───< 0x00408a47      7407           je 0x408a50
|     |||   0x00408a49      bf88db5800     mov edi, obj.__TMC_END__    ; loc._edata
|     |||                                                              ; 0x58db88
|     |||   0x00408a4e      ffe0           jmp rax
└     └└──> 0x00408a50      c3             ret
        |   0x00408a51      66662e0f1f84.  nop word cs:[rax + rax]
        |   0x00408a5c      0f1f4000       nop dword [rax]
        |   ;-- entry.fini0:
┌ 28: sym.__do_global_dtors_aux ();
|       |   0x00408a60      803d39511800.  cmp byte [obj.completed.0], 0 ; sym..bss
|       |                                                              ; [0x58dba0:1]=0
|      ┌──< 0x00408a67      7517           jne 0x408a80
|      ||   0x00408a69      55             push rbp
|      ||   0x00408a6a      4889e5         mov rbp, rsp
|      ||   0x00408a6d      e87effffff     call sym.deregister_tm_clones
|      ||   0x00408a72      c60527511800.  mov byte [obj.completed.0], 1 ; sym..bss
|      ||                                                              ; [0x58dba0:1]=0
|      ||   0x00408a79      5d             pop rbp
|      ||   0x00408a7a      c3             ret
|      ||   0x00408a7b      0f1f440000     nop dword [rax + rax]
└      └──> 0x00408a80      c3             ret
        |   0x00408a81      66662e0f1f84.  nop word cs:[rax + rax]
        |   0x00408a8c      0f1f4000       nop dword [rax]
        |   ;-- frame_dummy:
┌ 2: entry.init0 ();
└       └─< 0x00408a90      eb8e           jmp sym.register_tm_clones
            0x00408a92      662e0f1f8400.  nop word cs:[rax + rax]
            0x00408a9c      0f1f4000       nop dword [rax]
┌ 2138: sym.launch_program (int64_t arg1, int64_t arg2, int64_t arg3);
|           ; arg int64_t arg1 @ rdi
|           ; arg int64_t arg2 @ rsi
|           ; arg int64_t arg3 @ rdx
|           0x00408aa0      85f6           test esi, esi               ; arg2
```

Functions

Prolog

```
push rip
push rbp
mov rbp, rsp
```



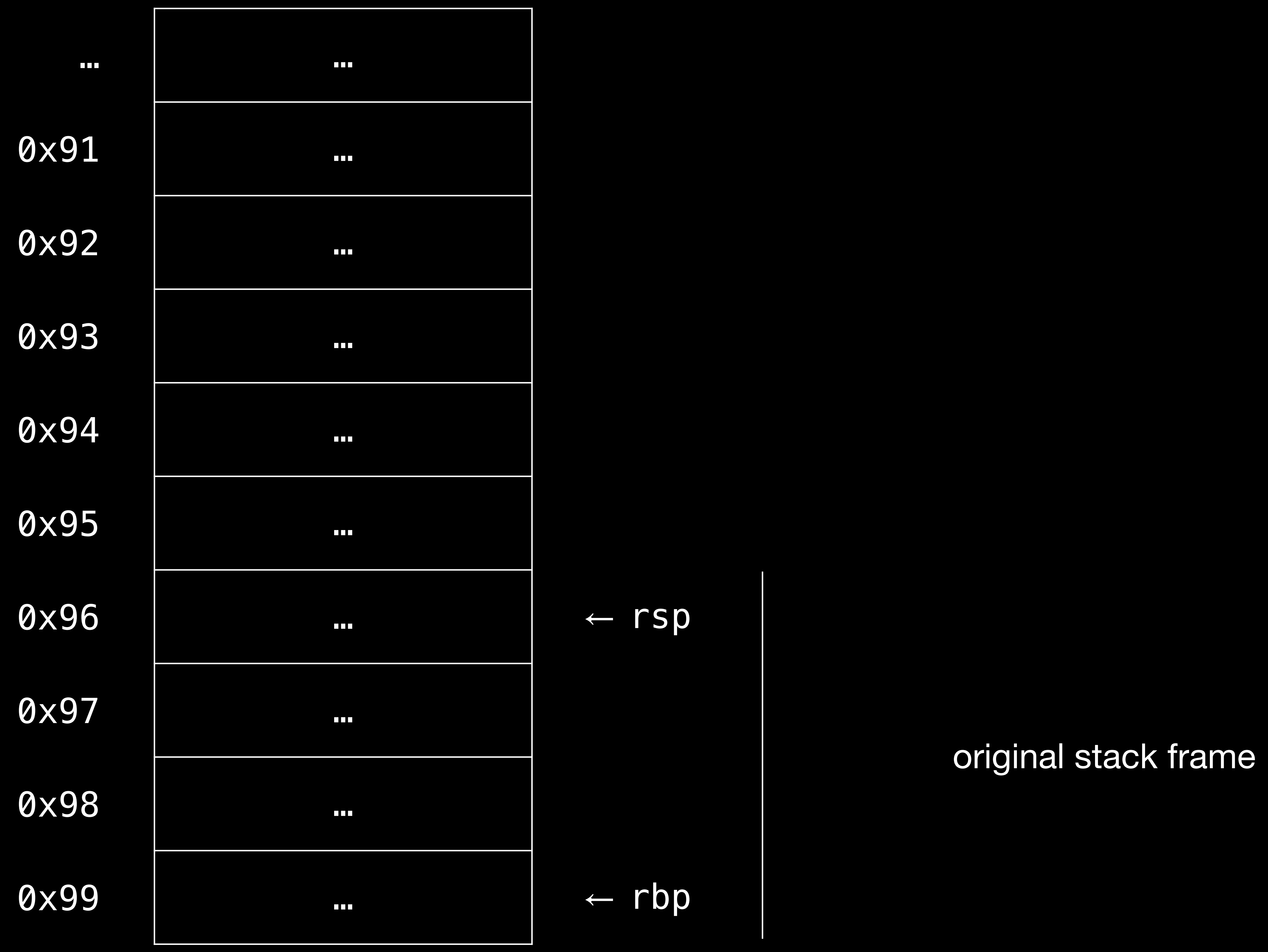
1 Move value from rsp…

2 into rbp

prolog

original state

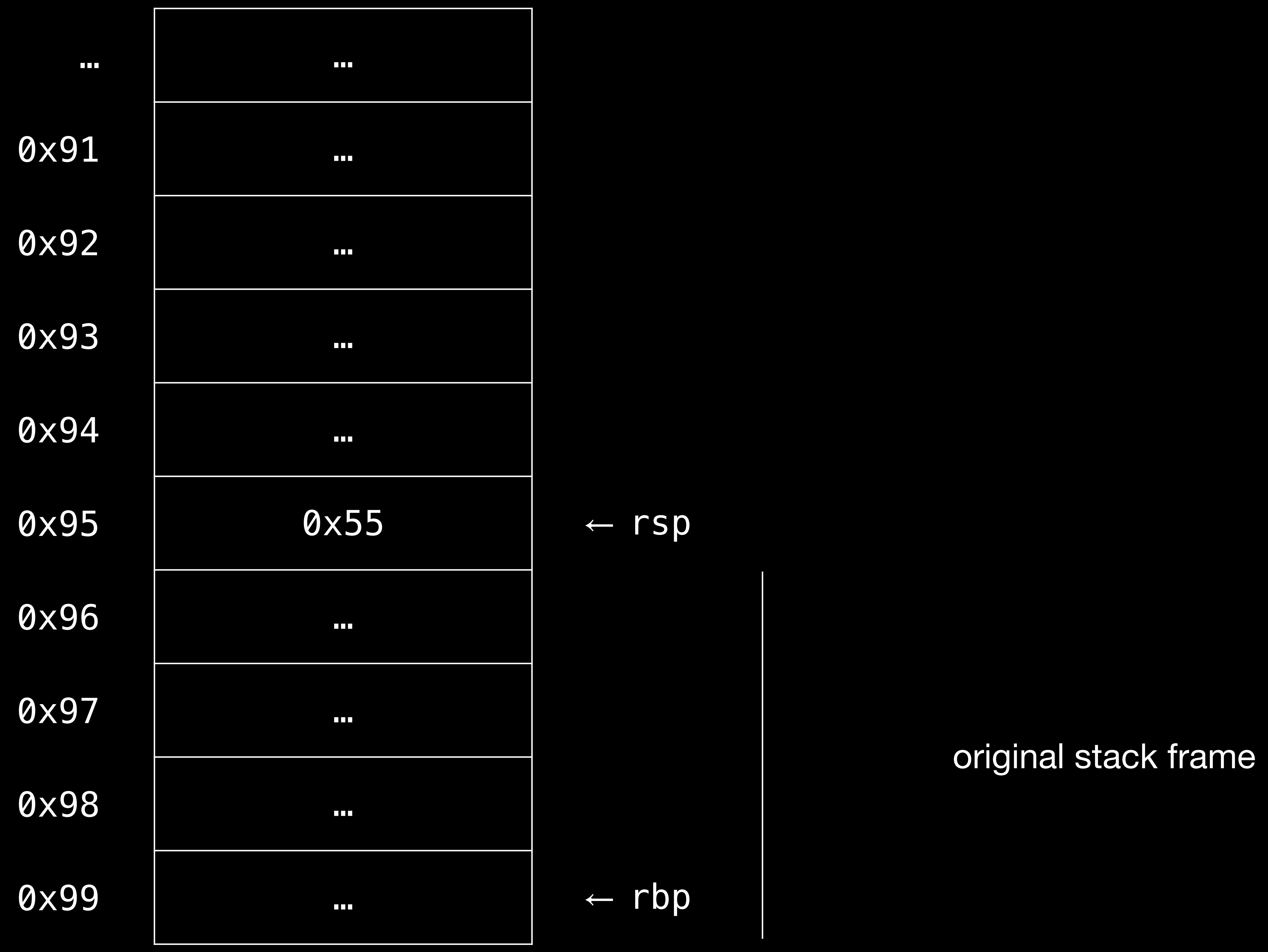
```
push rip
push rbp
mov rbp, rsp
```



prolog

store the instruction pointer

```
push rip
push rbp
mov rbp, rsp
```

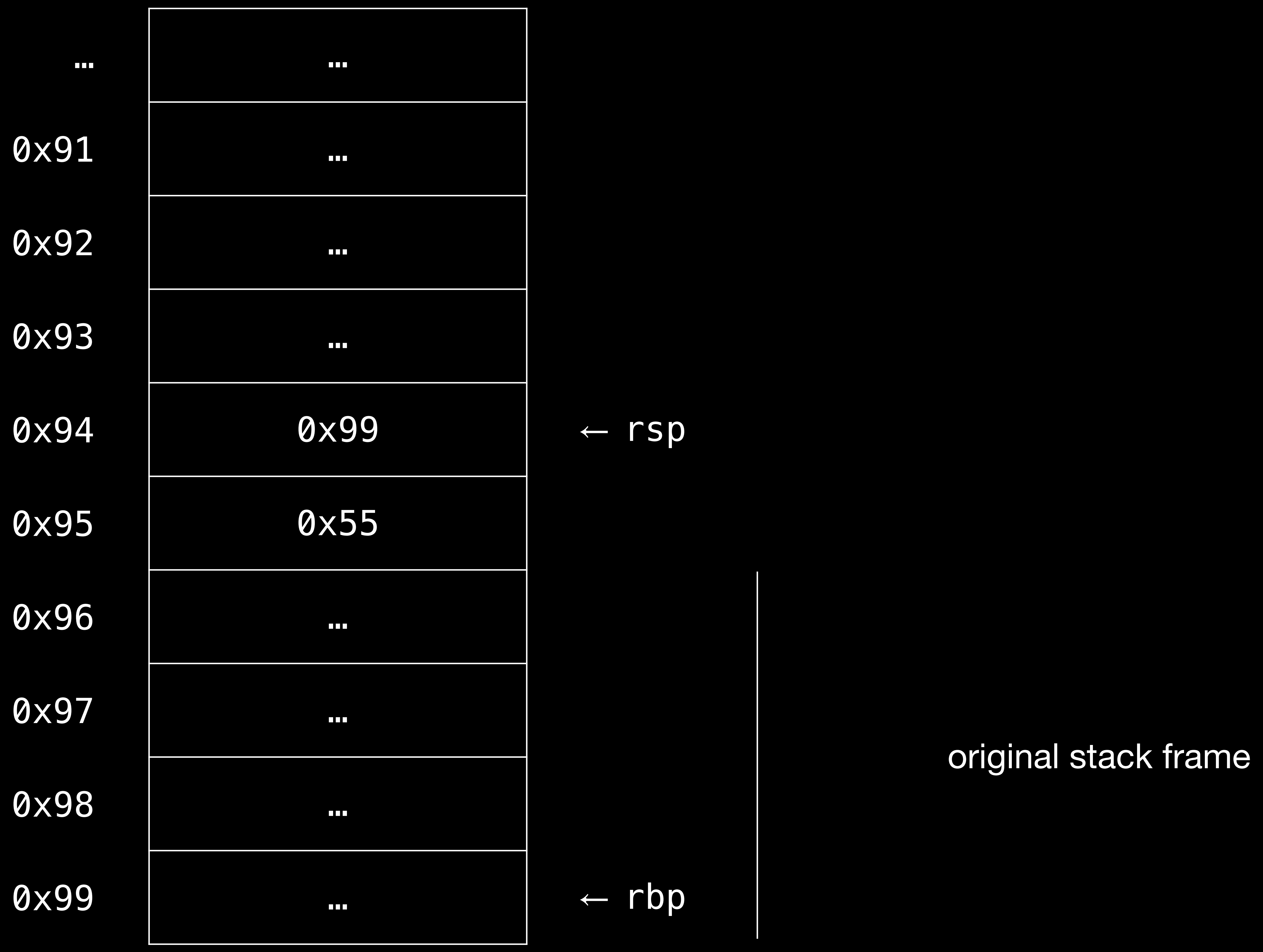


…	…	
0x91	…	
0x92	…	
0x93	…	
0x94	…	
0x95	0x55	← rsp
0x96	…	
0x97	…	
0x98	…	
0x99	…	← rbp

prolog

store the base pointer

```
push rip
push rbp
mov rbp, rsp
```



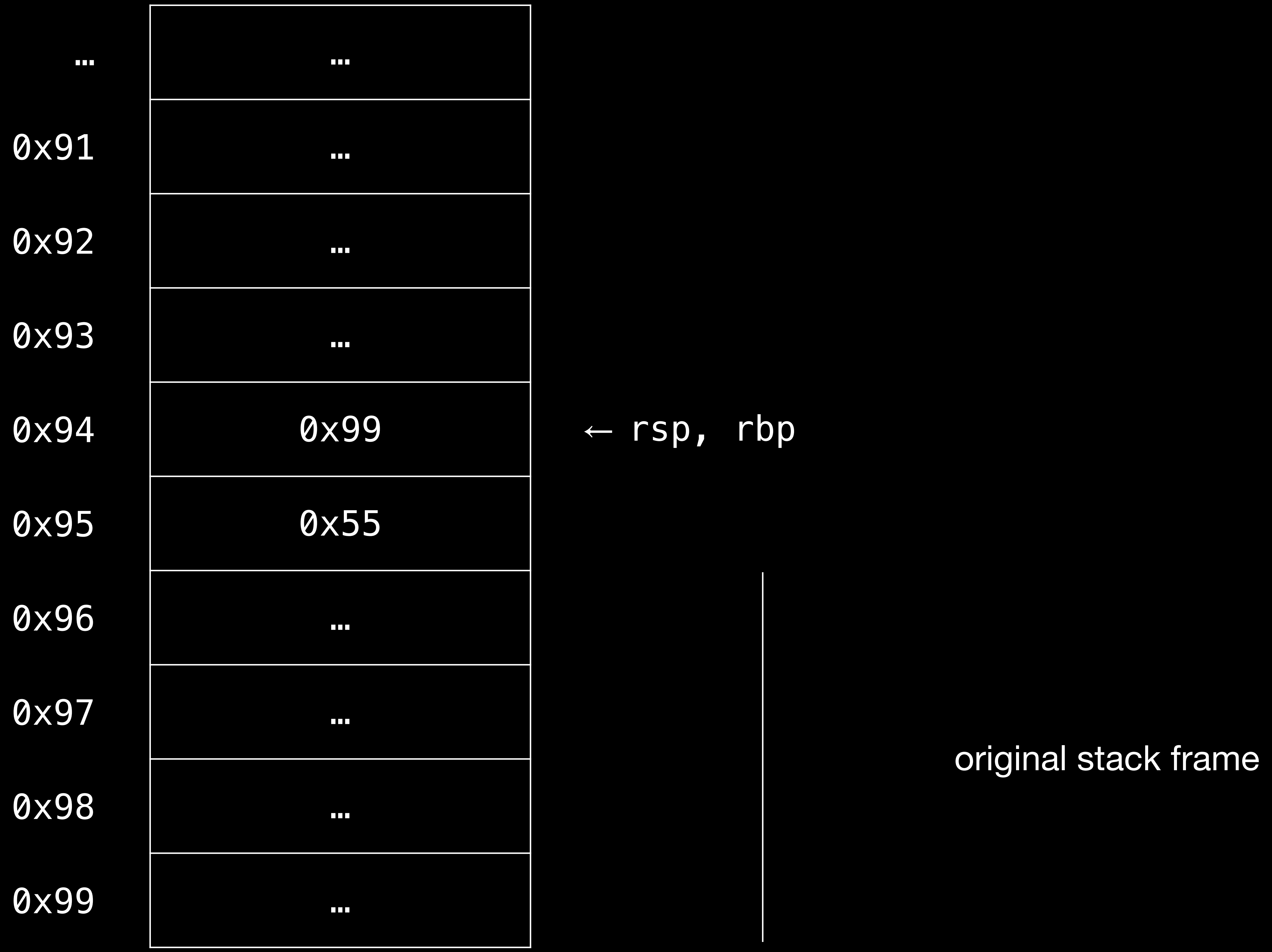
…	…	
0x91	…	
0x92	…	
0x93	…	
0x94	0x99	← rsp
0x95	0x55	
0x96	…	
0x97	…	
0x98	…	
0x99	…	← rbp

original stack frame

prolog

define the new base

```
push rip
push rbp
mov rbp, rsp
```



…	…	
0x91	…	
0x92	…	
0x93	…	
0x94	0x99	← rsp, rbp
0x95	0x55	
0x96	…	
0x97	…	
0x98	…	
0x99	…	

original stack frame

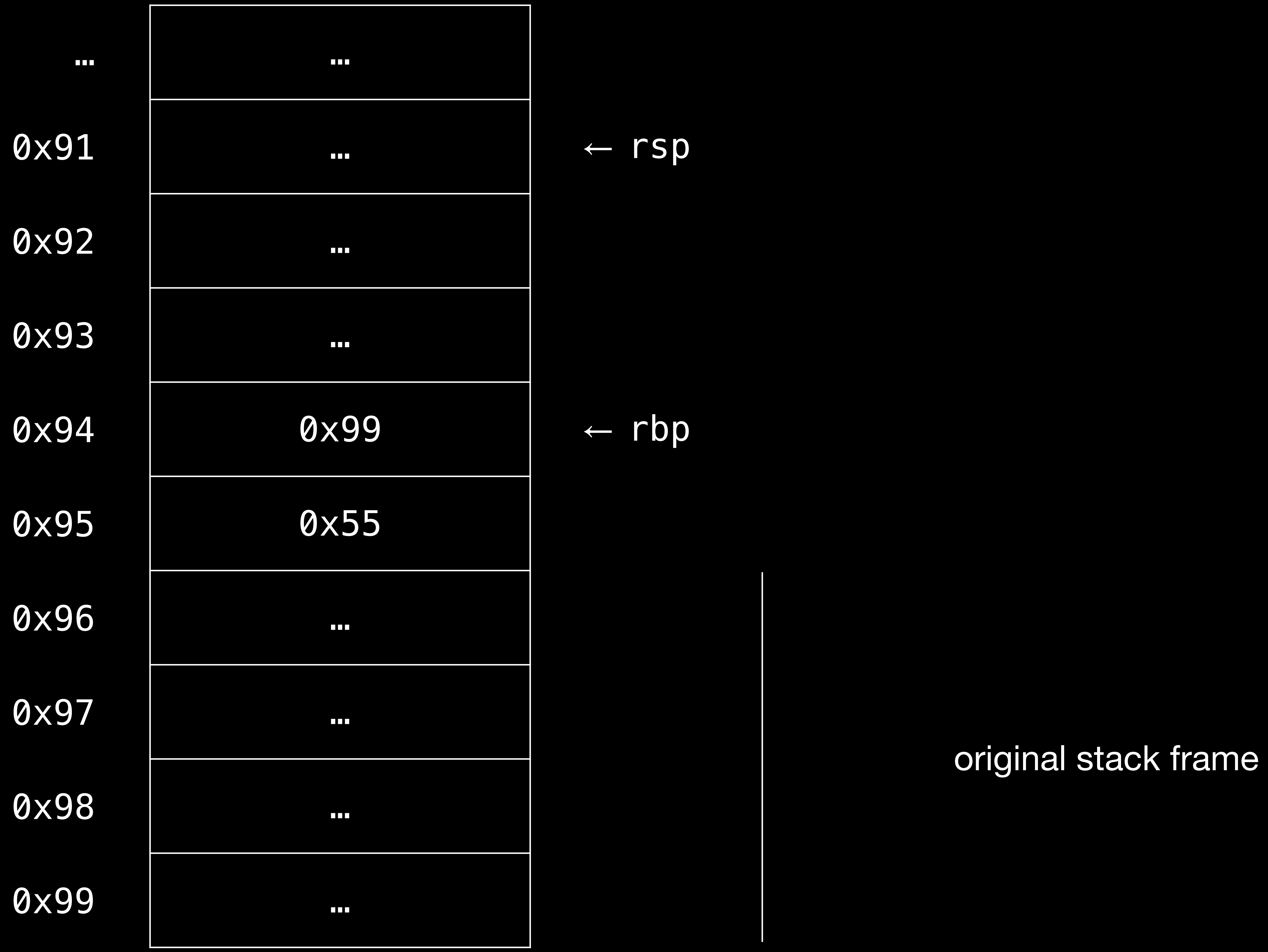
Epilog

```
mov rsp, rbp
pop rbp
pop rip
```

epilogue

start state

```
mov rsp, rbp
pop rbp
pop rip
```



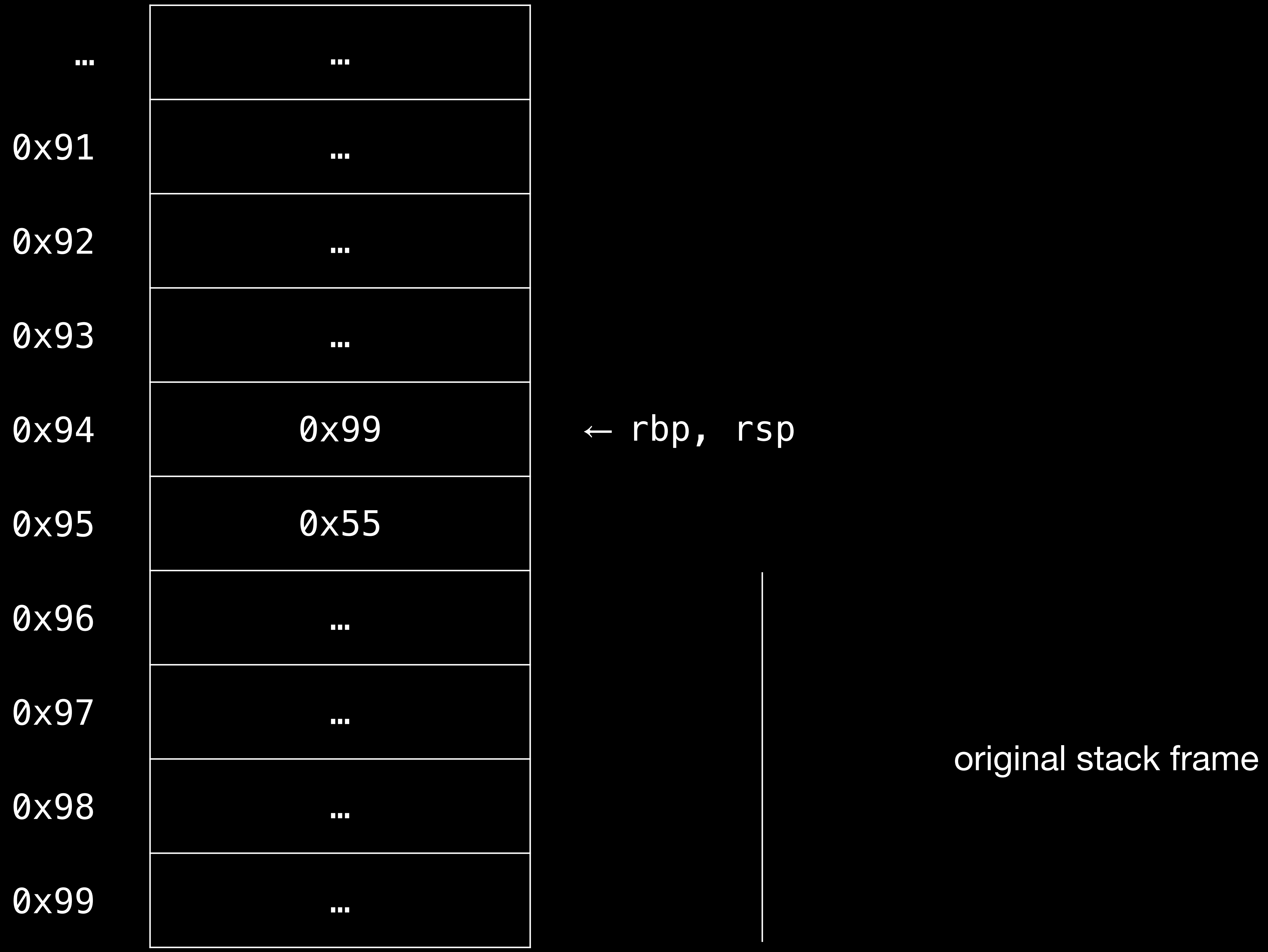
…	…	
0x91	…	← rsp
0x92	…	
0x93	…	
0x94	0x99	← rbp
0x95	0x55	
0x96	…	
0x97	…	
0x98	…	
0x99	…	

original stack frame

epilogue

move the stack pointer to the old base pointer

```
mov rsp, rbp
pop rbp
pop rip
```

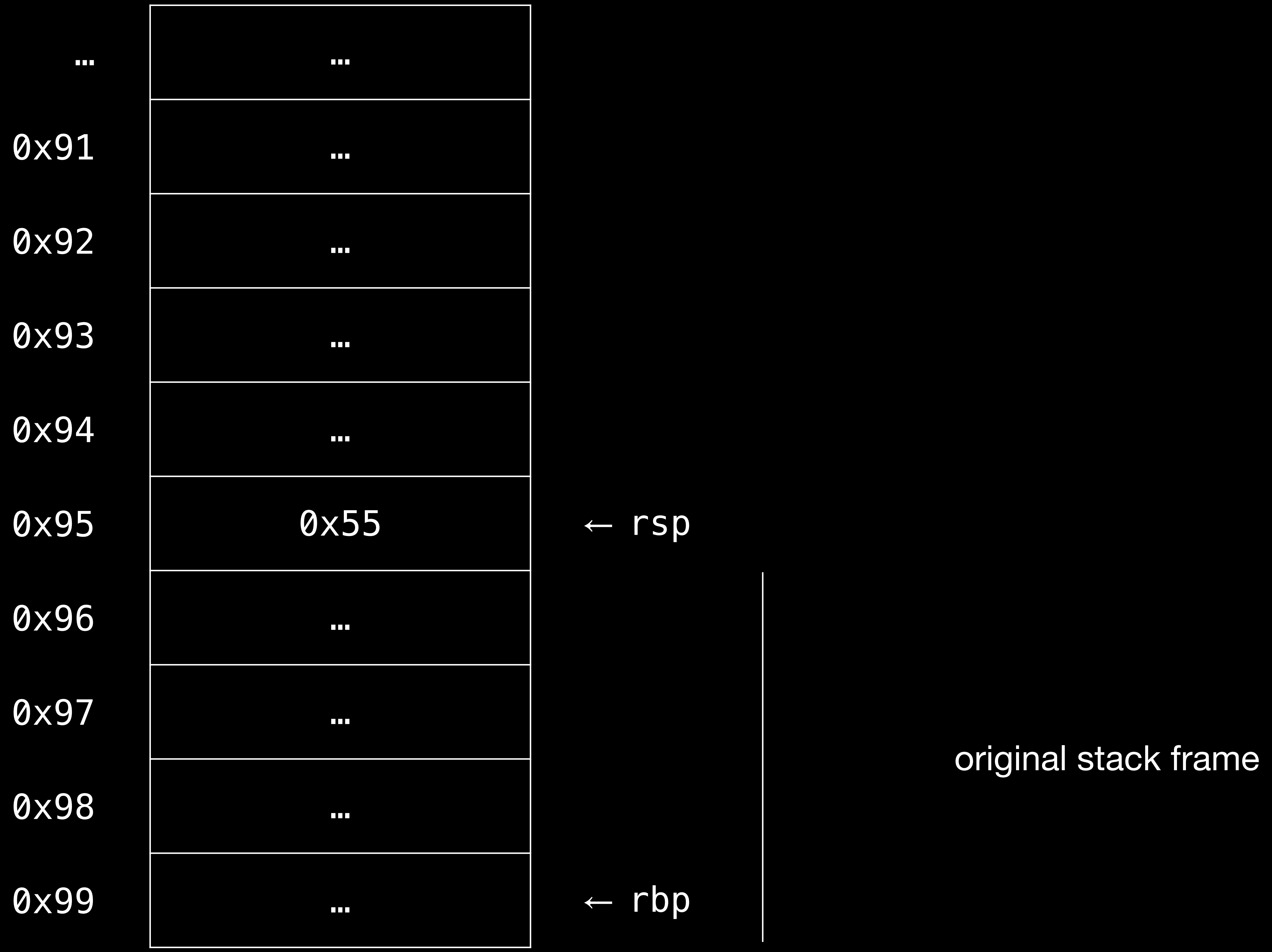


…	…	
0x91	…	
0x92	…	
0x93	…	
0x94	0x99	← rbp, rsp
0x95	0x55	
0x96	…	
0x97	…	
0x98	…	
0x99	…	

epilogue

reset the base pointer

```
mov rsp, rbp
pop rbp
pop rip
```



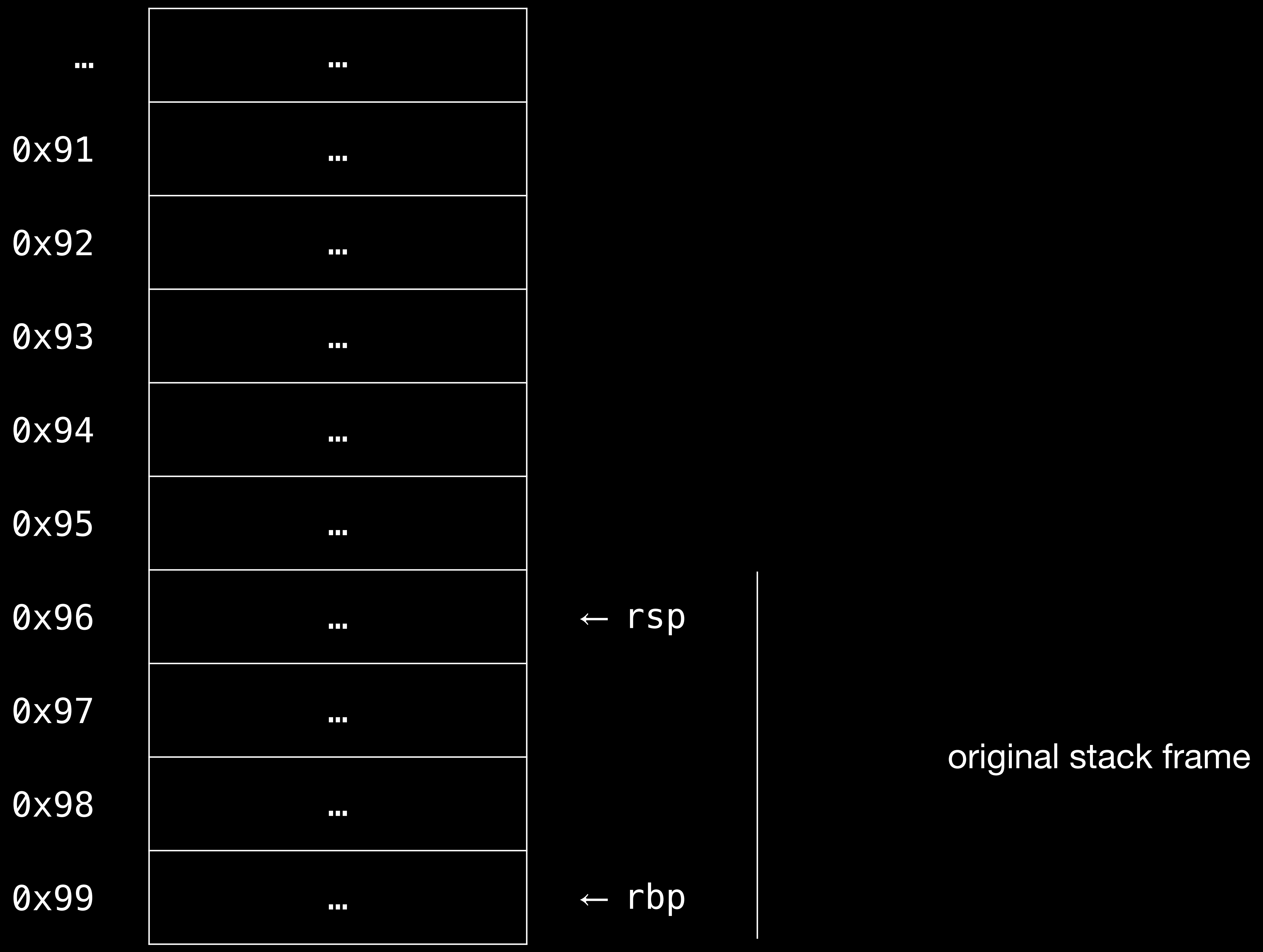
…	…	
0x91	…	
0x92	…	
0x93	…	
0x94	…	
0x95	0x55	← rsp
0x96	…	
0x97	…	
0x98	…	
0x99	…	← rbp

original stack frame

epilogue

reset the instruction pointer

```
mov rsp, rbp
pop rbp
pop rip
```



…	…	
0x91	…	
0x92	…	
0x93	…	
0x94	…	
0x95	…	
0x96	…	← rsp
0x97	…	
0x98	…	
0x99	…	← rbp

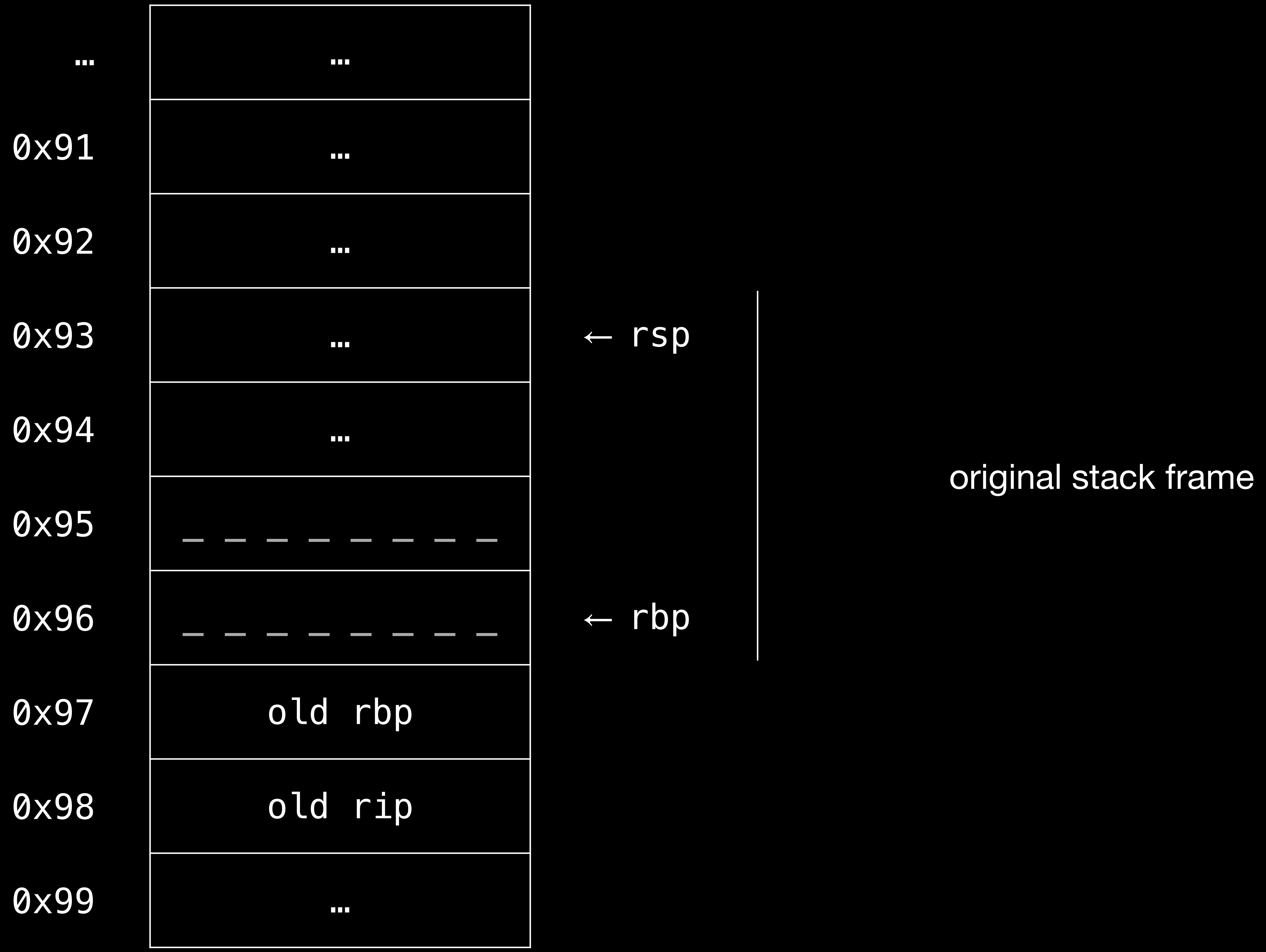
original stack frame

Buffers

buffers

(this is the point where s*** starts to break)

```
char str[16];
gets(str);
```



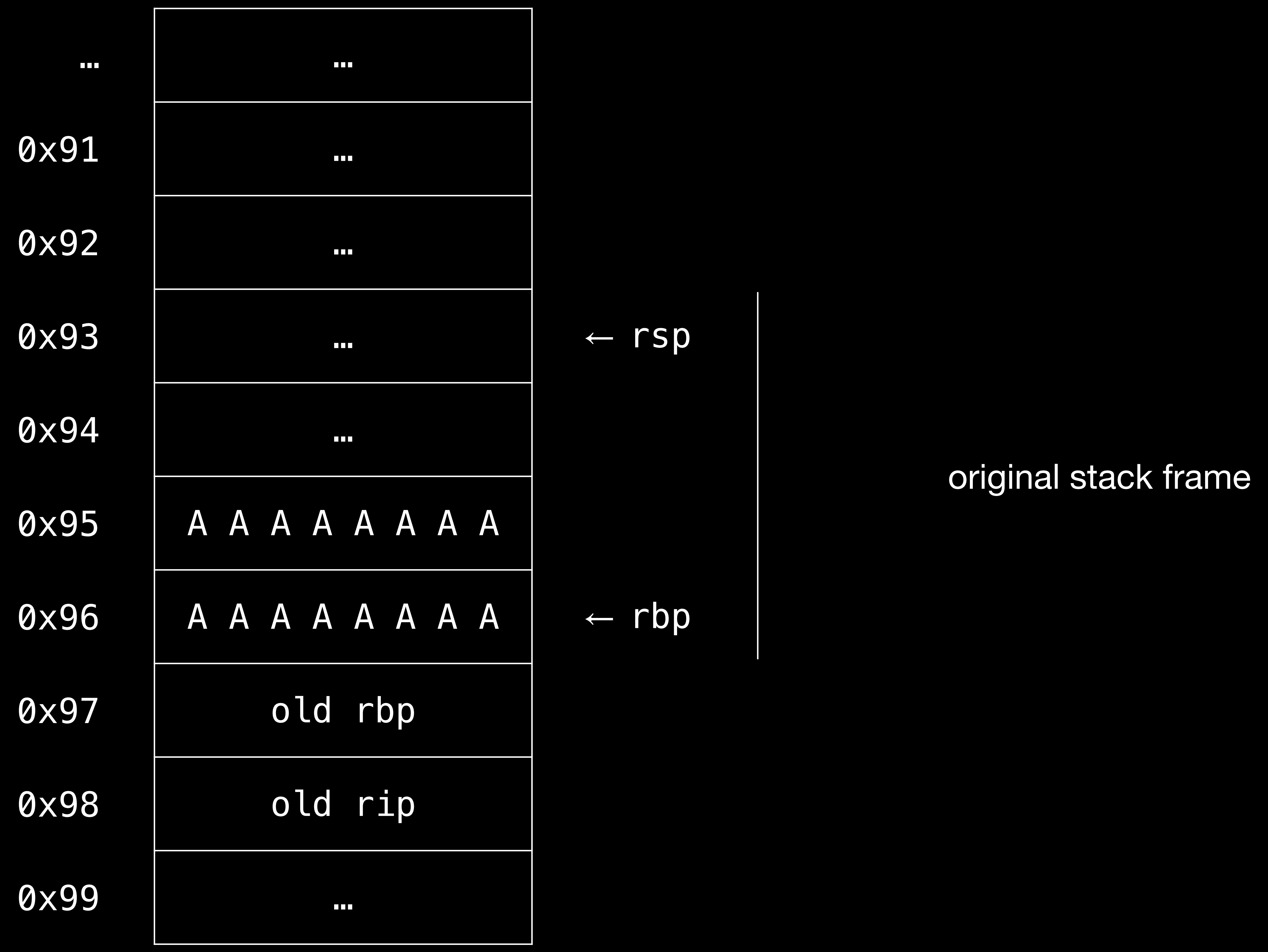
…	…		
0x91	…		
0x92	…		
0x93	…	← rsp	
0x94	…		original stack frame
0x95	_ _ _ _ _ _ _ _		
0x96	_ _ _ _ _ _ _ _	← rbp	
0x97	old rbp		
0x98	old rip		
0x99	…		

Buffer Overflow

buffers

filling the buffer with “A”s

```
char str[16];
gets(str);
```

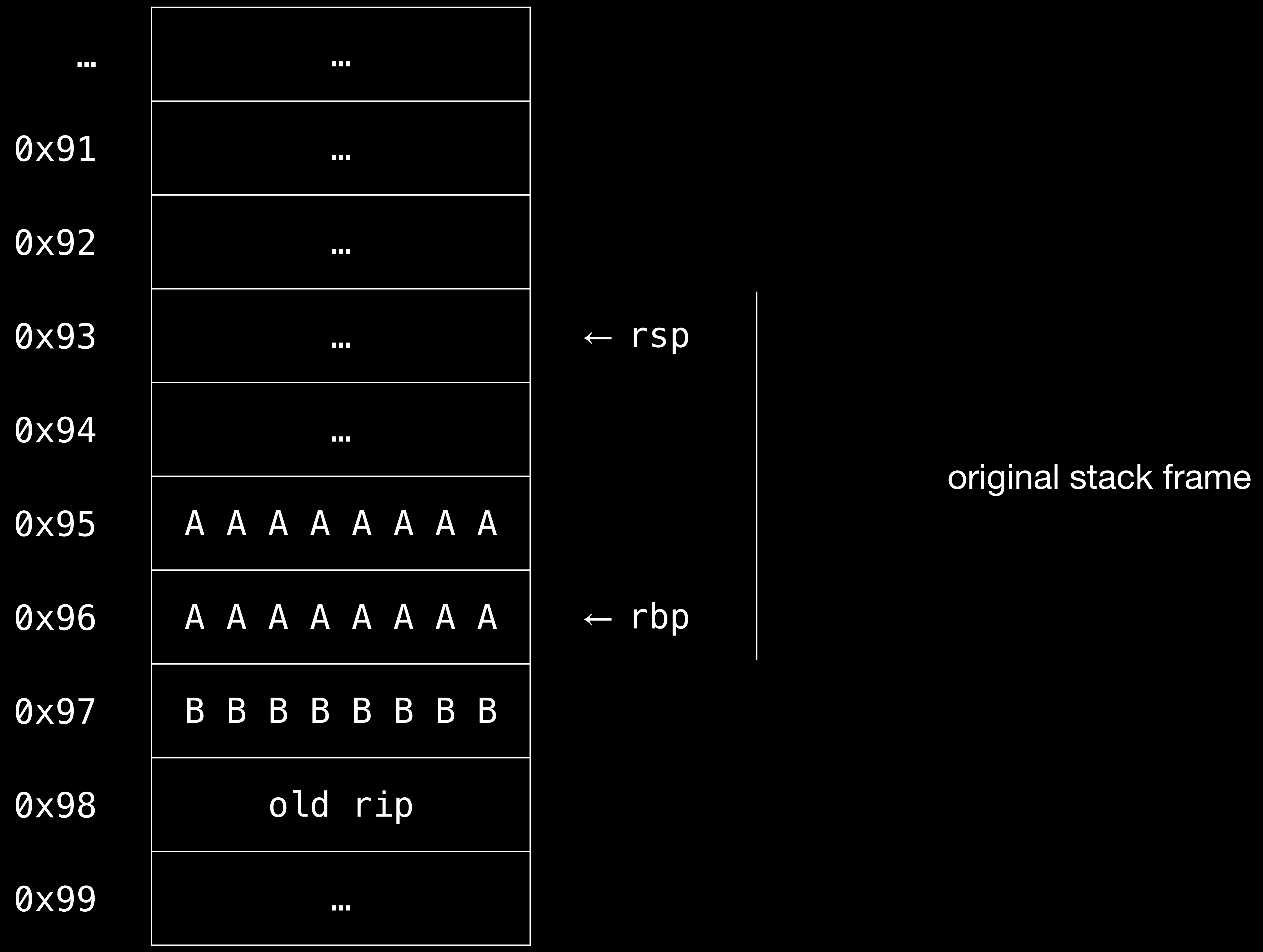


…	…	
0x91	…	
0x92	…	
0x93	…	← rsp
0x94	…	
0x95	A A A A A A A A	
0x96	A A A A A A A A	← rbp
0x97	old rbp	
0x98	old rip	
0x99	…	

buffers

overwriting the base pointer of the caller with “B”

```
char str[16];
gets(str);
```



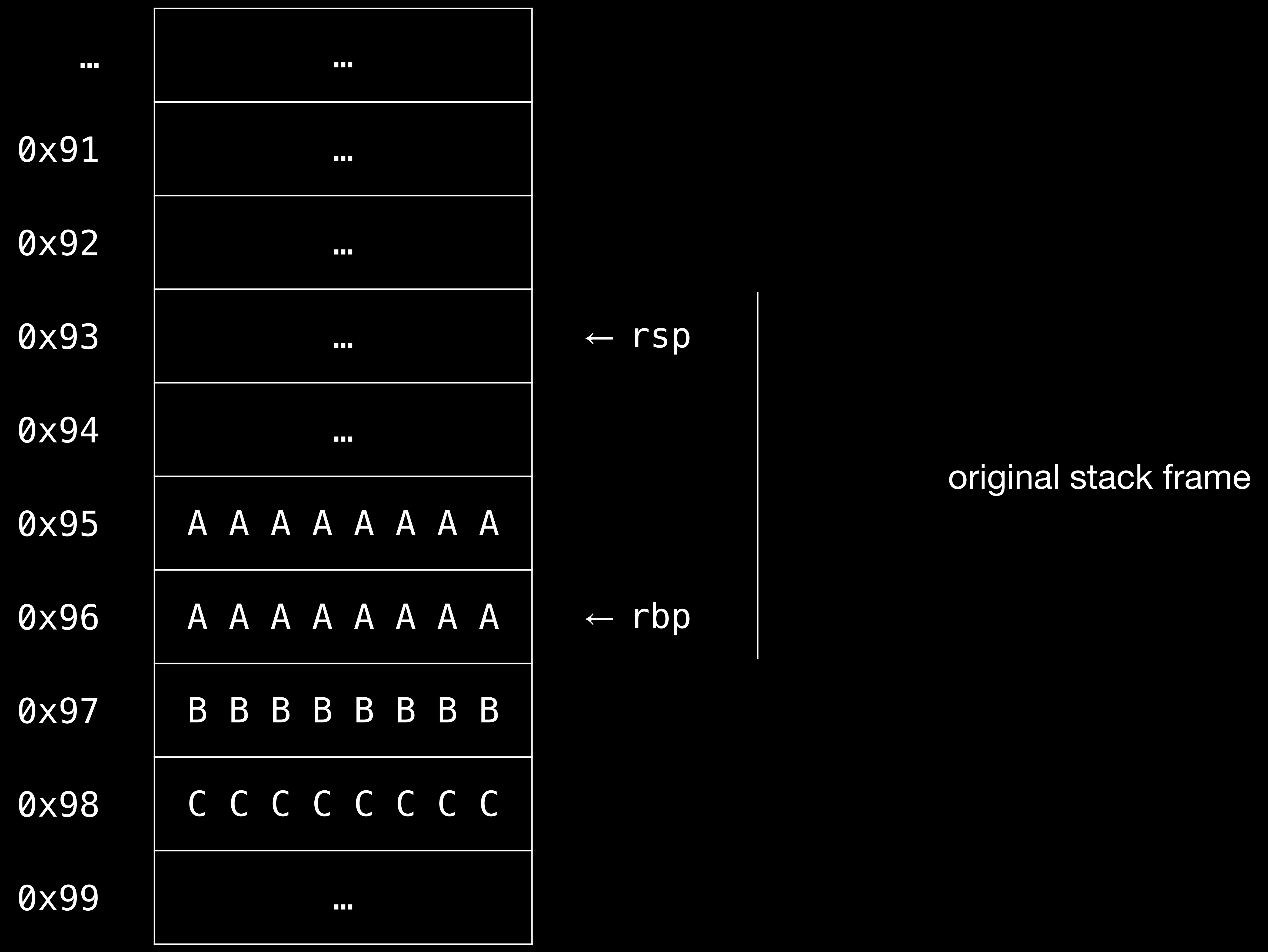
…	…	
0x91	…	
0x92	…	
0x93	…	← rsp
0x94	…	
0x95	A A A A A A A A	
0x96	A A A A A A A A	← rbp
0x97	B B B B B B B B	
0x98	old rip	
0x99	…	

original stack frame

buffers

overwriting the return address of the function with “C”

```
char str[16];
gets(str);
```



…	…	
0x91	…	
0x92	…	
0x93	…	← rsp
0x94	…	
0x95	A A A A A A A A	
0x96	A A A A A A A A	← rbp
0x97	B B B B B B B B	
0x98	C C C C C C C C	
0x99	…	

original stack frame