intro2rev

You had to be there for the attendance flag :)

~RISC 27/8/25

Acknowledgment of Country

RISC acknowledges the people of the Woi Wurrung and Boon Wurrung language groups of the eastern Kulin Nation on whose unceded lands we conduct the business of the University and the club. RISC acknowledges their Ancestors and Elders, past, present, and emerging

Today’s Sponsor…

BINARY NINJA

Binary Ninja is an interactive decompiler, disassembler, debugger, and binary analysis platform built by reverse engineers, for reverse engineers.

Developed with a focus on delivering a high-quality API for automation and a clean and usable GUI, Binary Ninja is in active use by malware analysts, vulnerability researchers, and software developers worldwide.

Decompile software built for many common architectures on Windows, macOS, and Linux.

Hello, world!

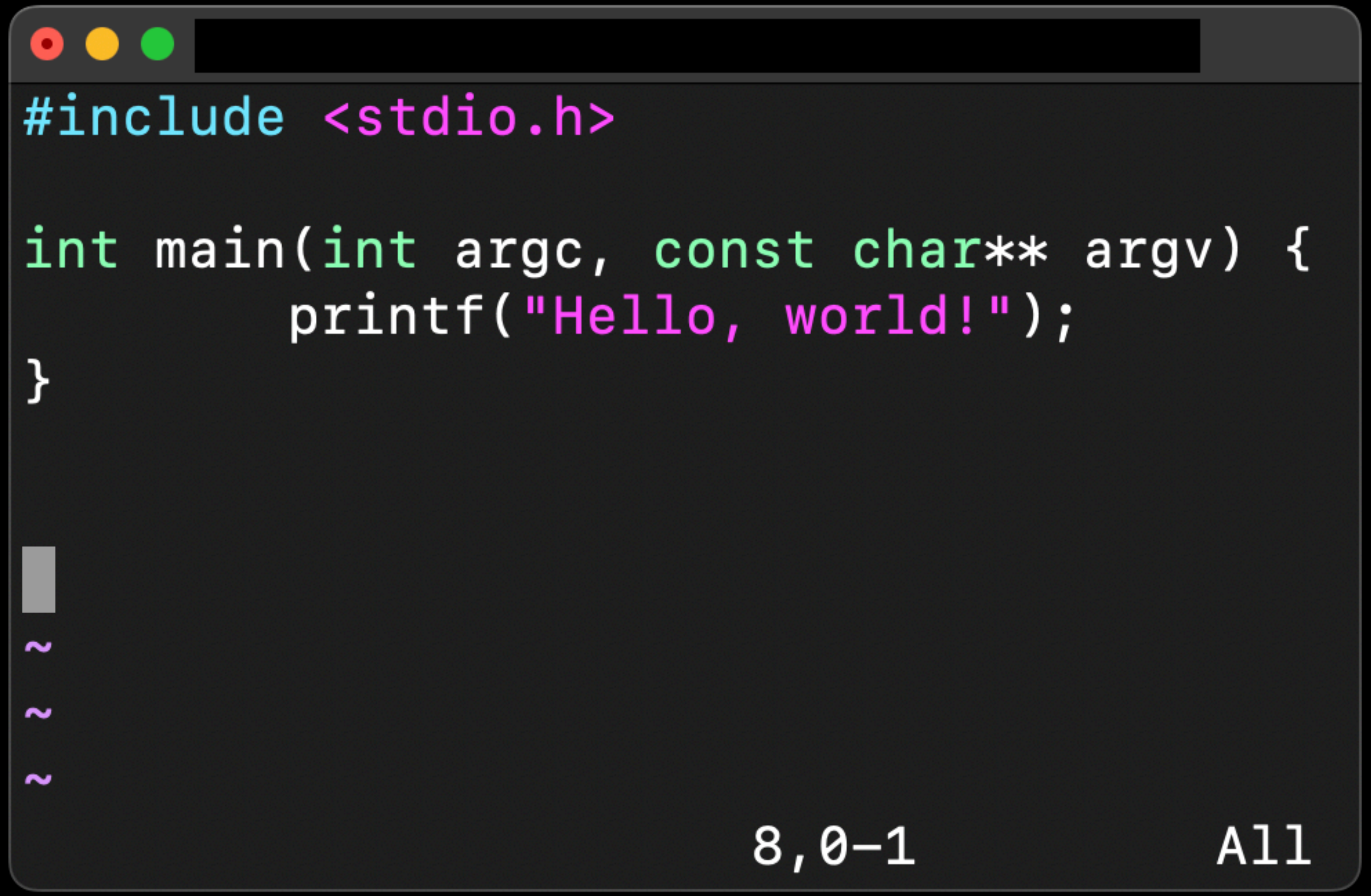
baby steps

- We’ve all (hopefully) written hello world before

Hello, world!

baby steps

- We’ve all (hopefully) written hello world before
- Maybe in C?

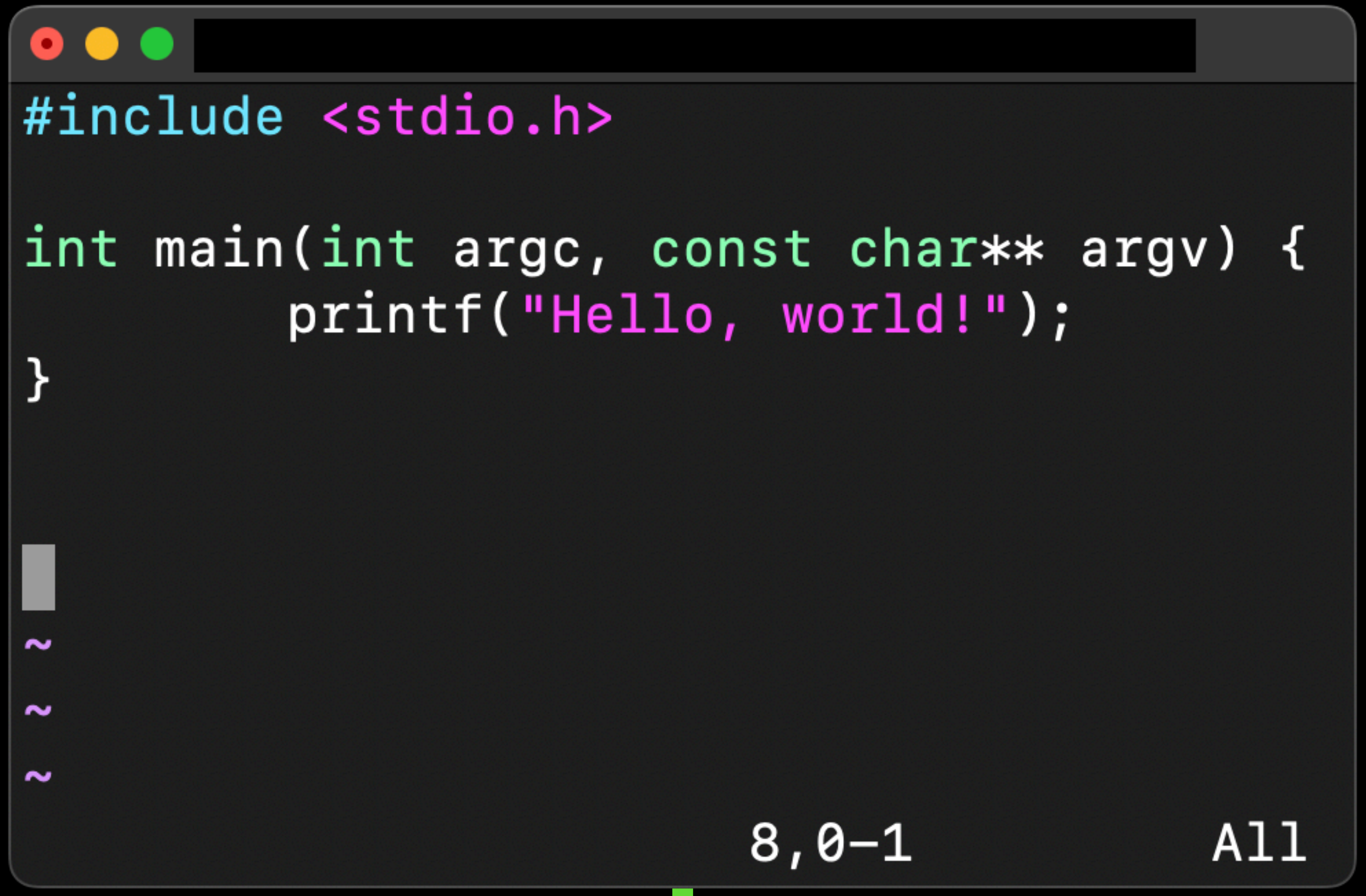


```
#include <stdio.h>

int main(int argc, const char** argv) {
        printf("Hello, world!");
}


~
~
~
                                   8,0-1         All
```

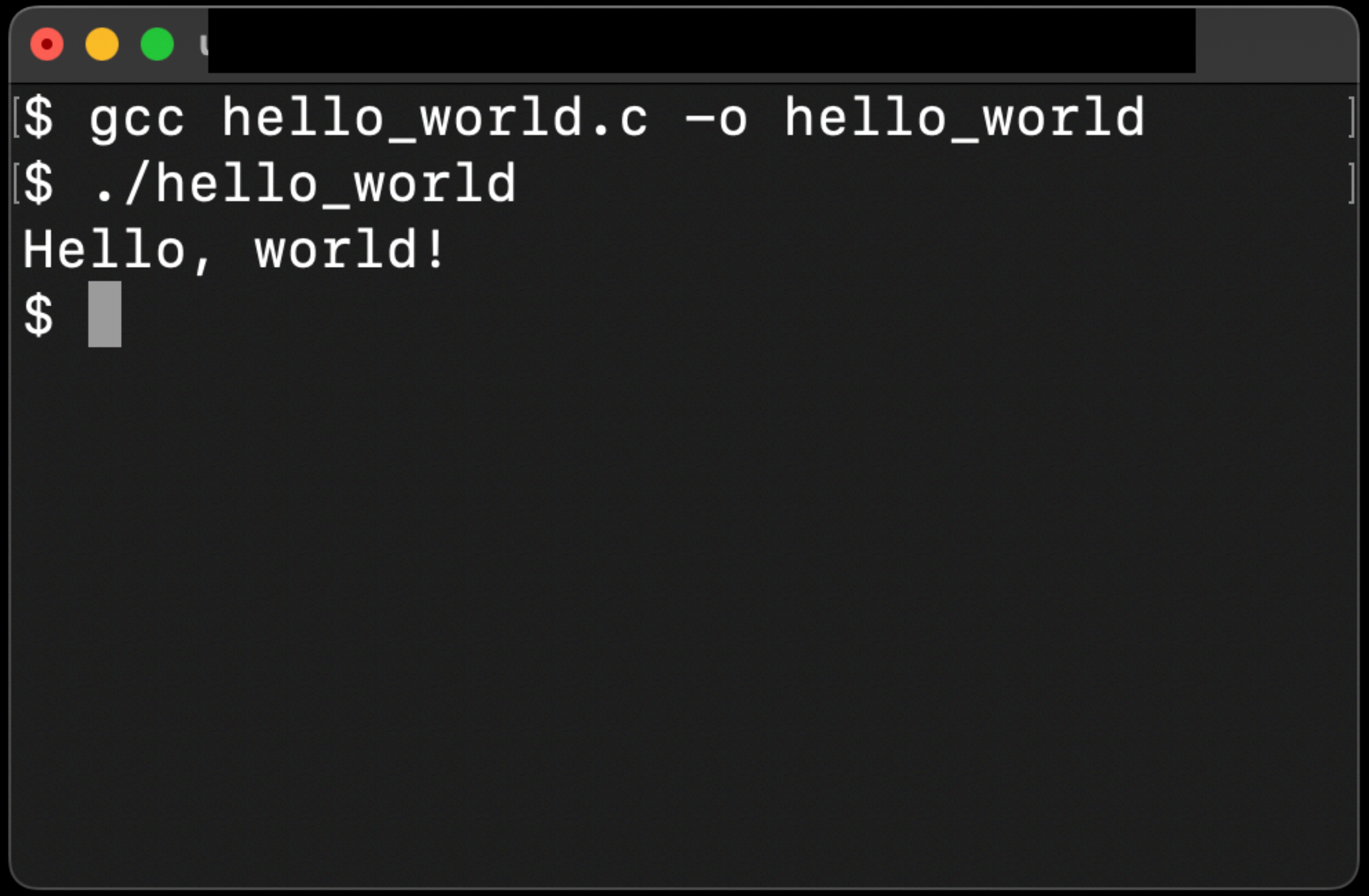
Hello, world!

baby steps

- We’ve all (hopefully) written hello world before
- Maybe in C?
- “Compile” using gcc and then run



```
#include <stdio.h>

int main(int argc, const char** argv) {
        printf("Hello, world!");
}


~
~
~
                      8,0-1          All
```

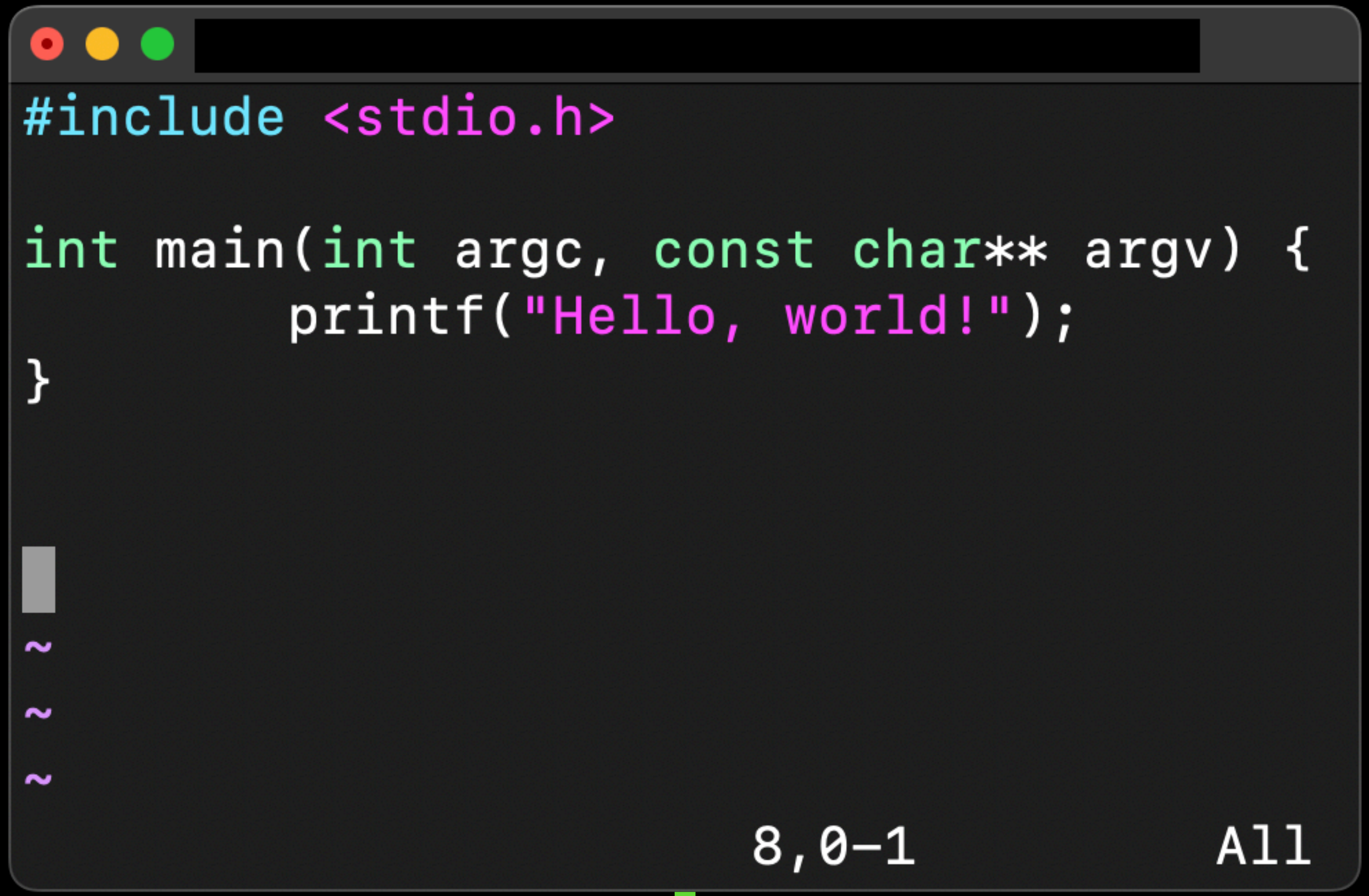


```
[$ gcc hello_world.c -o hello_world
[$ ./hello_world
Hello, world!
$
```

Hello, world!

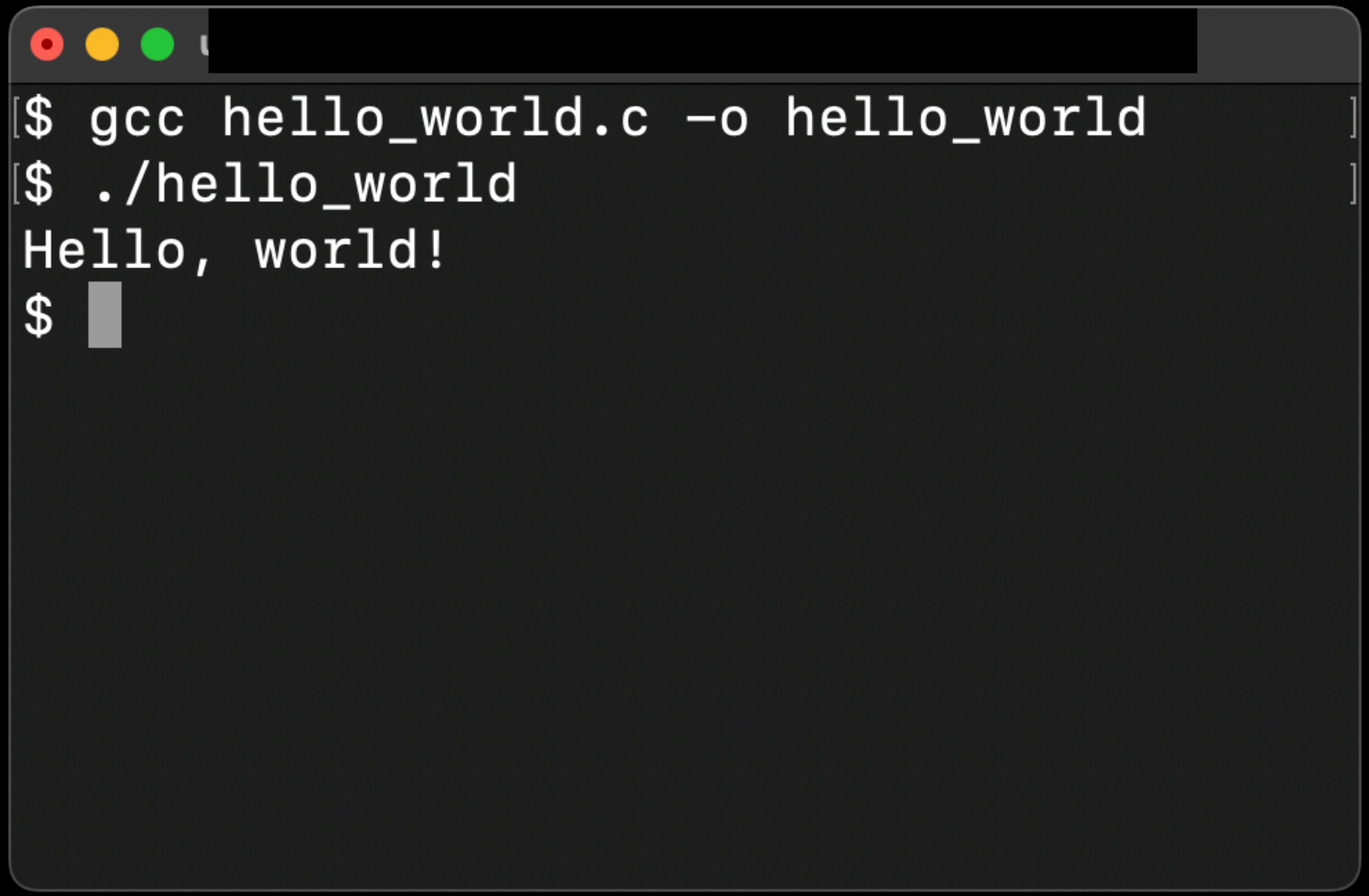
baby steps

- We’ve all (hopefully) written hello world before
- Maybe in C?
- “Compile” using gcc and then run
  - What actually happens here?



[redacted]

```
#include <stdio.h>

int main(int argc, const char** argv) {
        printf("Hello, world!");
}


█
~
~
~
                      8,0-1         All
```



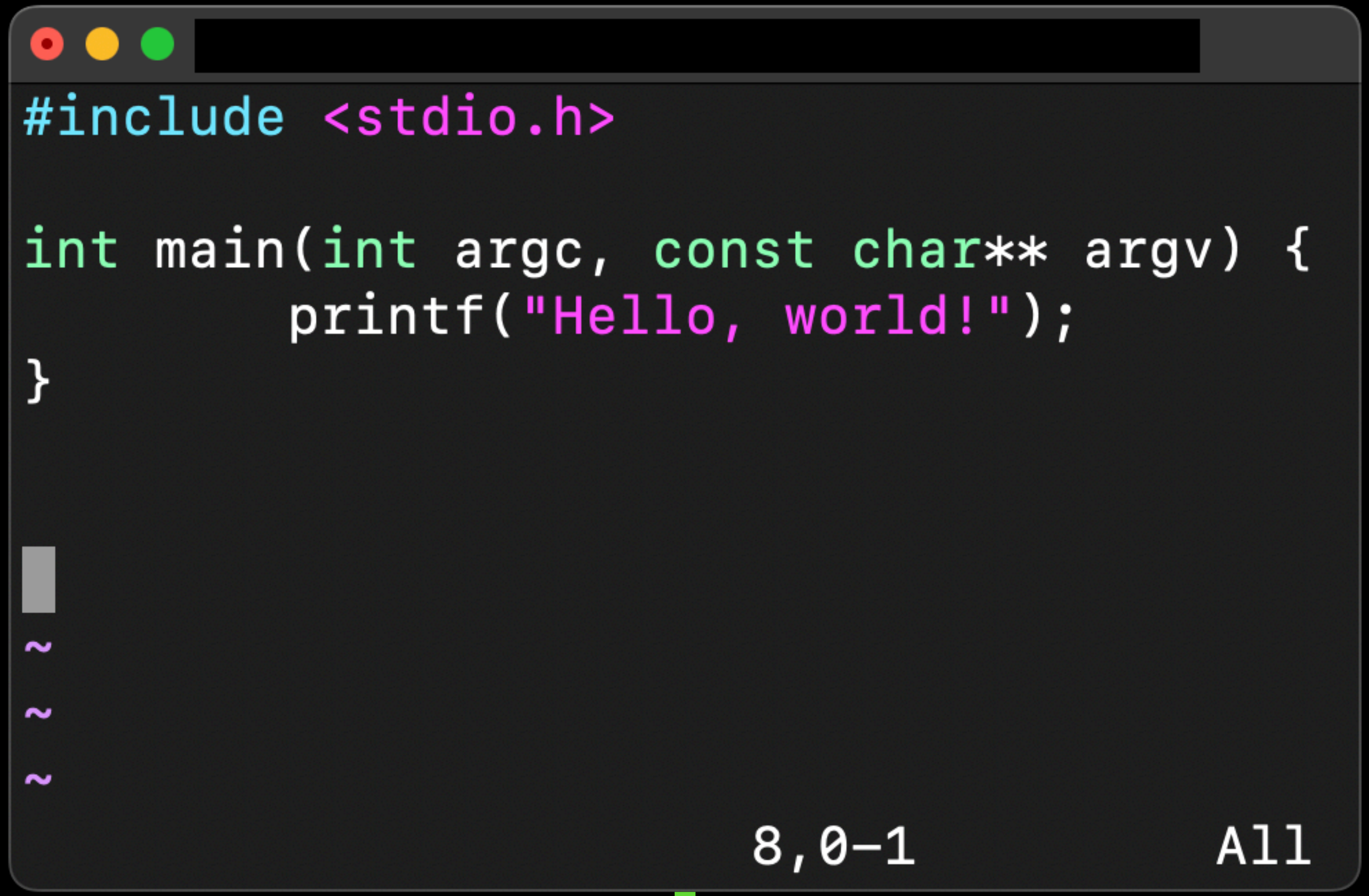
[redacted]

```
[$ gcc hello_world.c -o hello_world    ]
[$ ./hello_world                        ]
Hello, world!
$ █
```

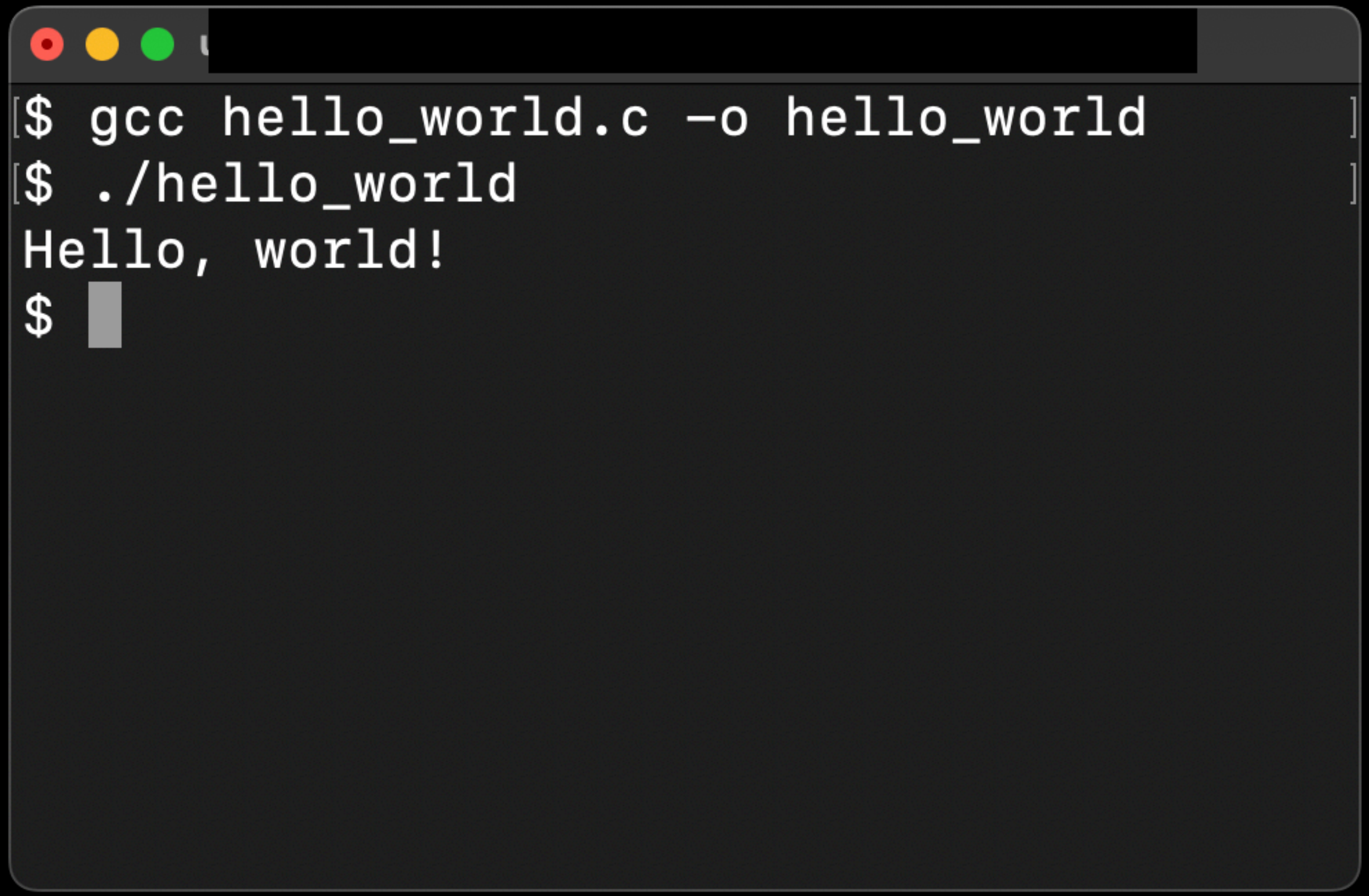
Hello, world!

baby steps

- We’ve all (hopefully) written hello world before
- Maybe in C?
- “Compile” using gcc and then run
  - What actually happens here?
  - Why can’t I just run `hello_world.c`?



```
#include <stdio.h>

int main(int argc, const char** argv) {
        printf("Hello, world!");
}



~
~
~
                      8,0-1         All
```

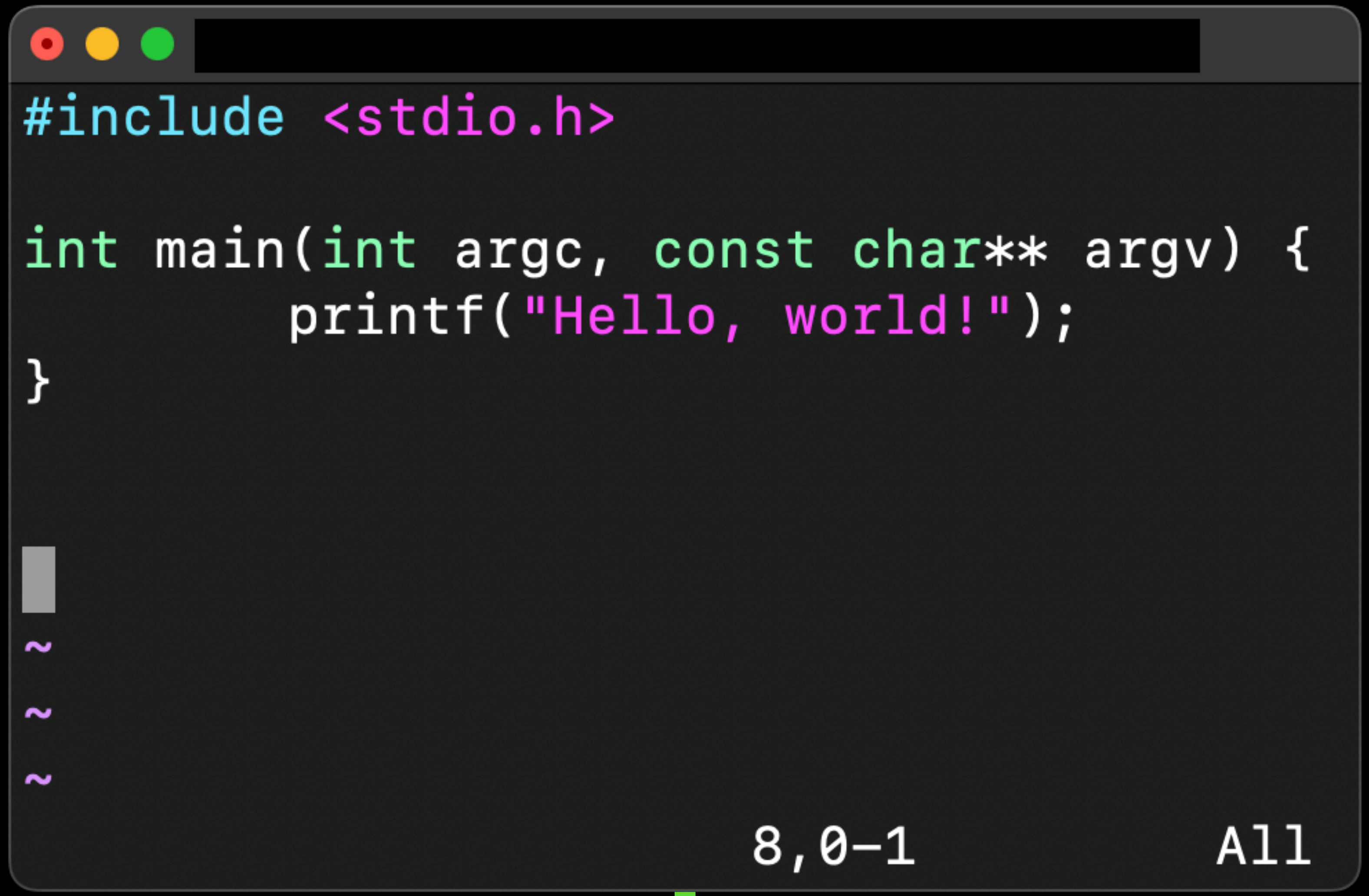


```
[$ gcc hello_world.c -o hello_world                  ]
[$ ./hello_world                                     ]
Hello, world!
$ 
```

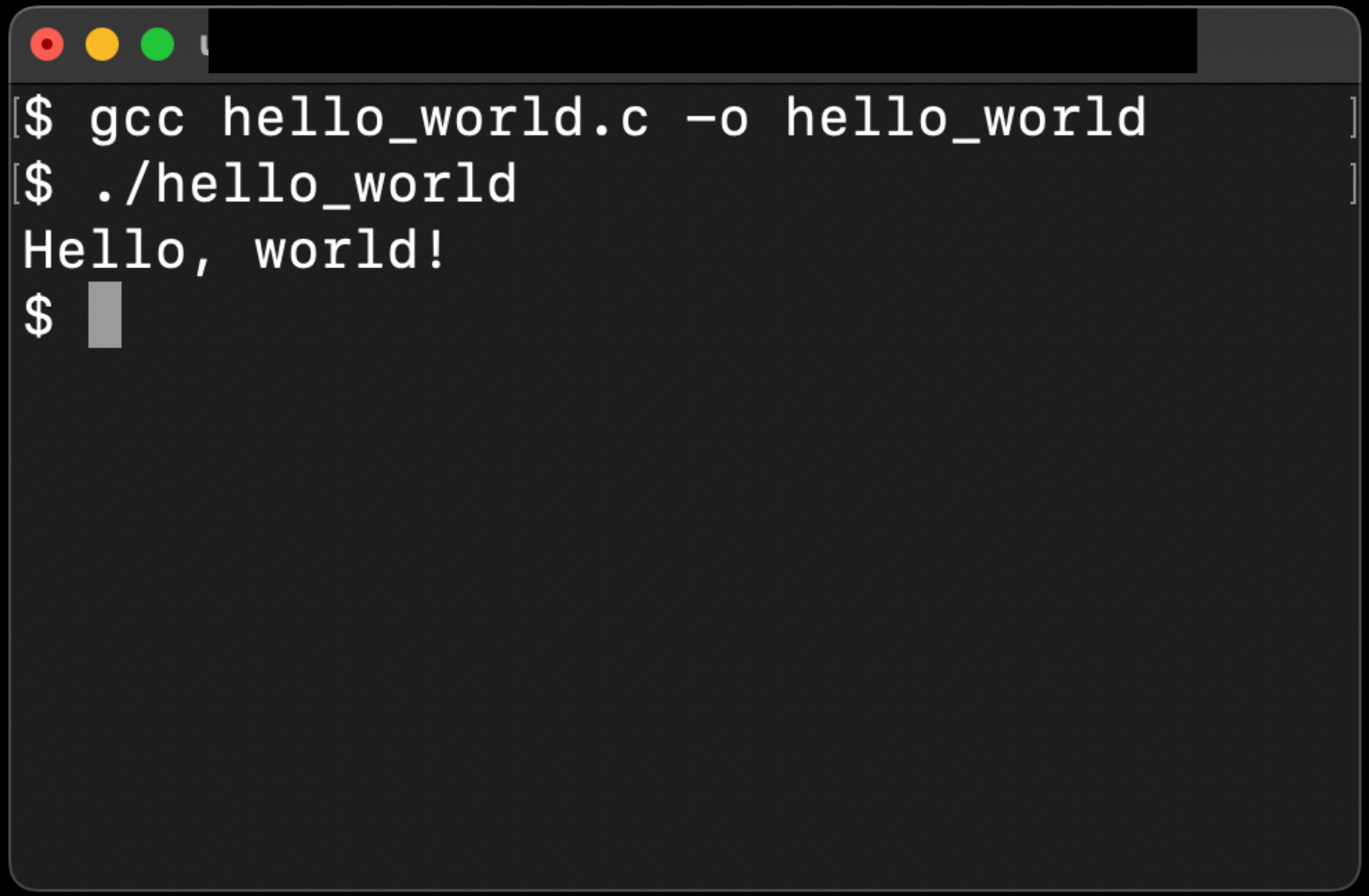
Hello, world!

baby steps

- We’ve all (hopefully) written hello world before
- Maybe in C?
- “Compile” using gcc and then run
    - What actually happens here?
    - Why can’t I just run `hello_world.c`?
    - What even is the `hello_world` file?



```
#include <stdio.h>

int main(int argc, const char** argv) {
        printf("Hello, world!");
}


~
~
~
                        8,0-1         All
```

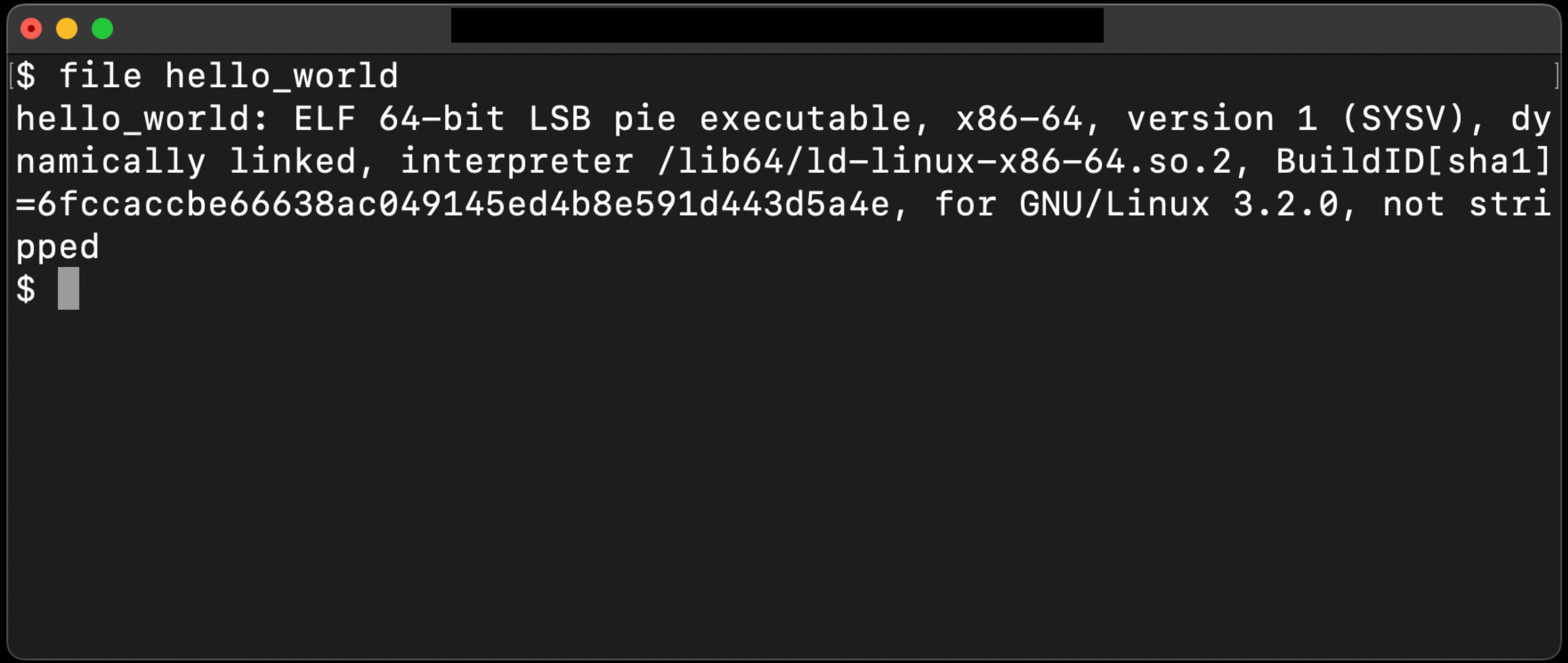


```
[$ gcc hello_world.c -o hello_world
[$ ./hello_world
Hello, world!
$
```

Hello, world!

baby steps

- `file` command tells us a lot



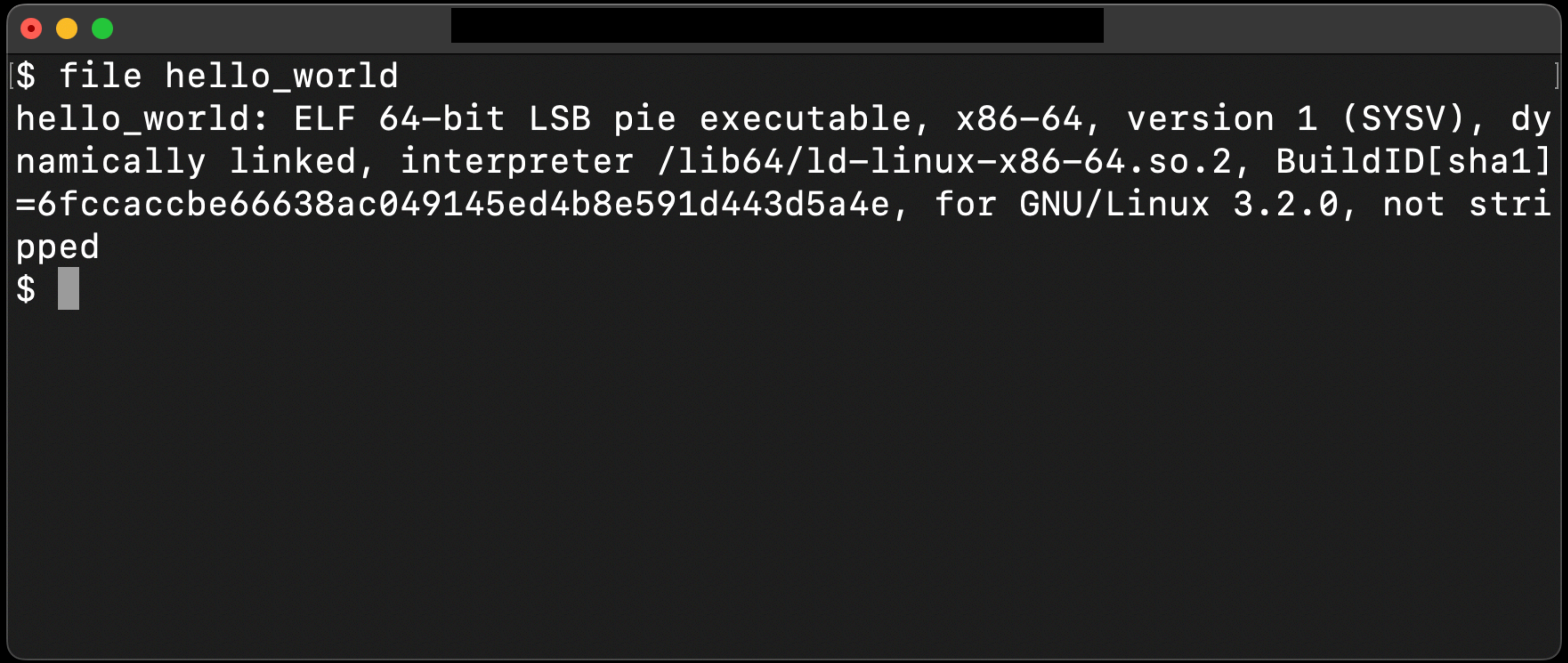
[redacted]

```
[$ file hello_world
hello_world: ELF 64-bit LSB pie executable, x86-64, version 1 (SYSV), dy
namically linked, interpreter /lib64/ld-linux-x86-64.so.2, BuildID[sha1]
=6fccaccbe66638ac049145ed4b8e591d443d5a4e, for GNU/Linux 3.2.0, not stri
pped
$ 
```

Hello, world!

baby steps

- `file` command tells us a lot
  - ELF executable



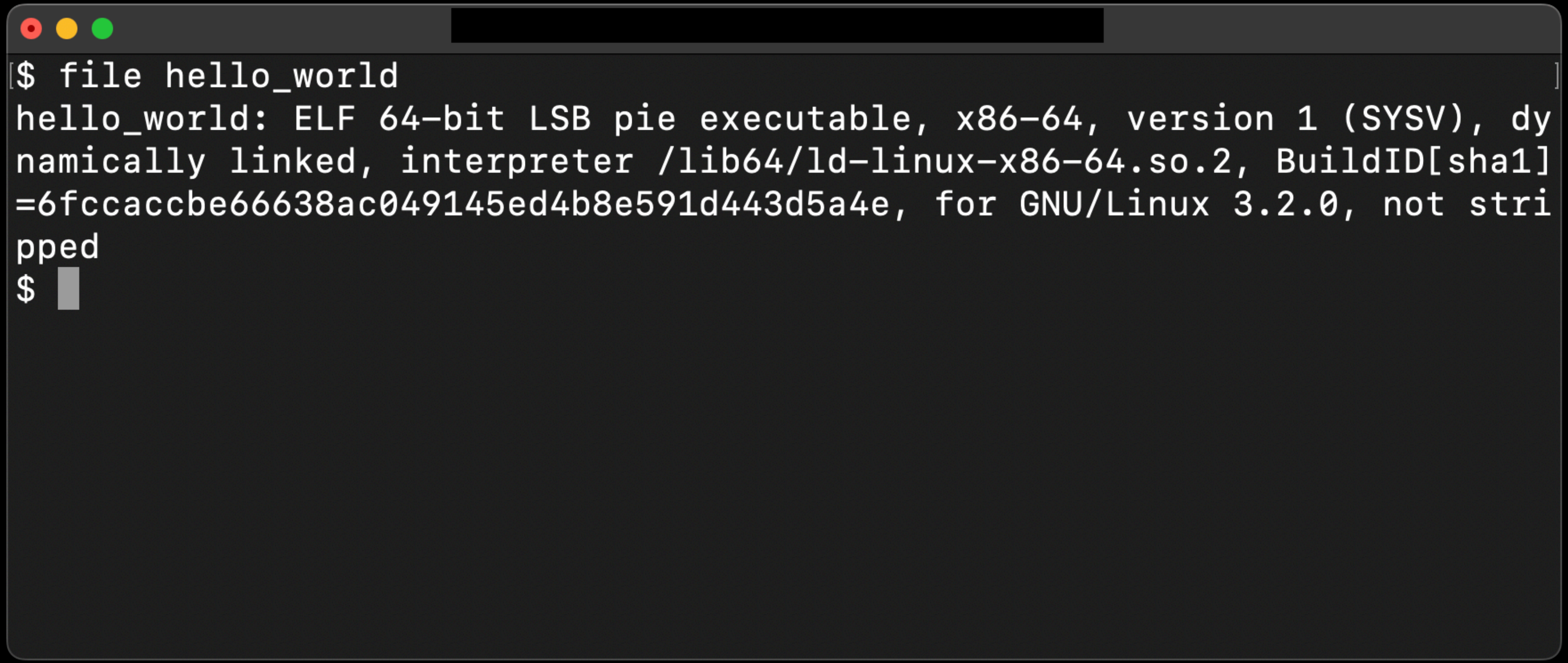
[redacted]

```
[$ file hello_world                                                             ]
hello_world: ELF 64-bit LSB pie executable, x86-64, version 1 (SYSV), dy
namically linked, interpreter /lib64/ld-linux-x86-64.so.2, BuildID[sha1]
=6fccaccbe66638ac049145ed4b8e591d443d5a4e, for GNU/Linux 3.2.0, not stri
pped
$ 
```

Hello, world!

baby steps

- `file` command tells us a lot
  - ELF executable
  - Compiled for x86-64 CPUs



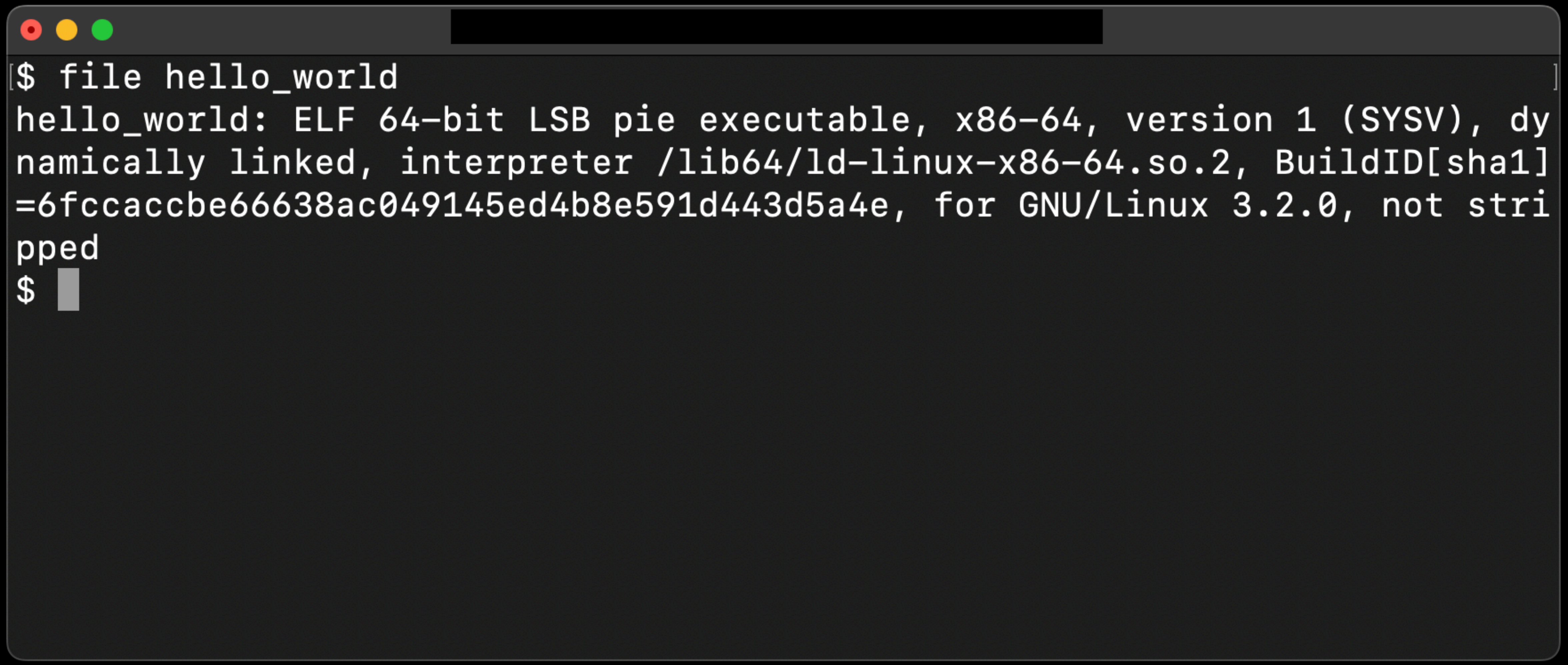
[redacted]

```
$ file hello_world
hello_world: ELF 64-bit LSB pie executable, x86-64, version 1 (SYSV), dy
namically linked, interpreter /lib64/ld-linux-x86-64.so.2, BuildID[sha1]
=6fccaccbe66638ac049145ed4b8e591d443d5a4e, for GNU/Linux 3.2.0, not stri
pped
$ 
```

Hello, world!

baby steps

- `file` command tells us a lot
  - ELF executable
  - Compiled for x86-64 CPUs
  - SYSV ABI



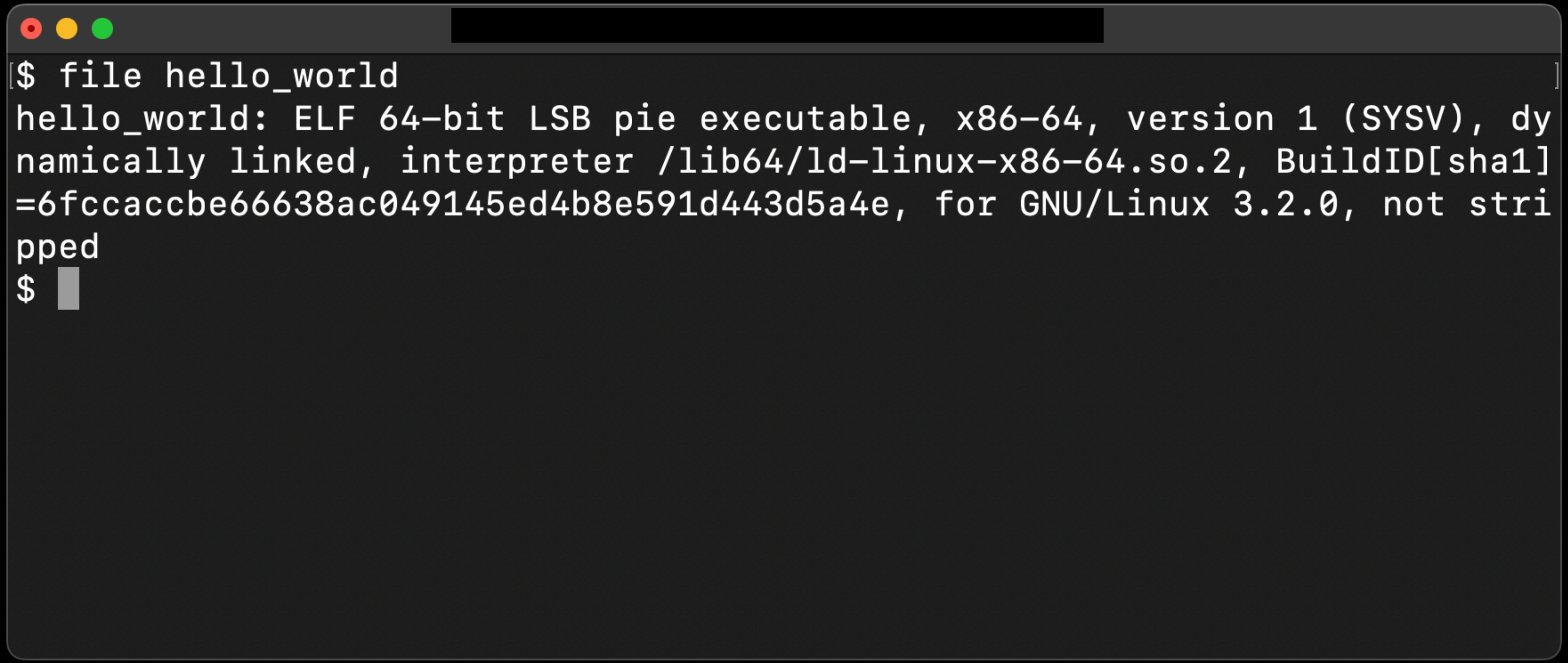
[redacted]

```
$ file hello_world
hello_world: ELF 64-bit LSB pie executable, x86-64, version 1 (SYSV), dy
namically linked, interpreter /lib64/ld-linux-x86-64.so.2, BuildID[sha1]
=6fccaccbe66638ac049145ed4b8e591d443d5a4e, for GNU/Linux 3.2.0, not stri
pped
$ 
```

Hello, world!

baby steps

- `file` command tells us a lot
  - ELF executable
  - Compiled for x86-64 CPUs
  - SYSV ABI
  - Dynamically linked



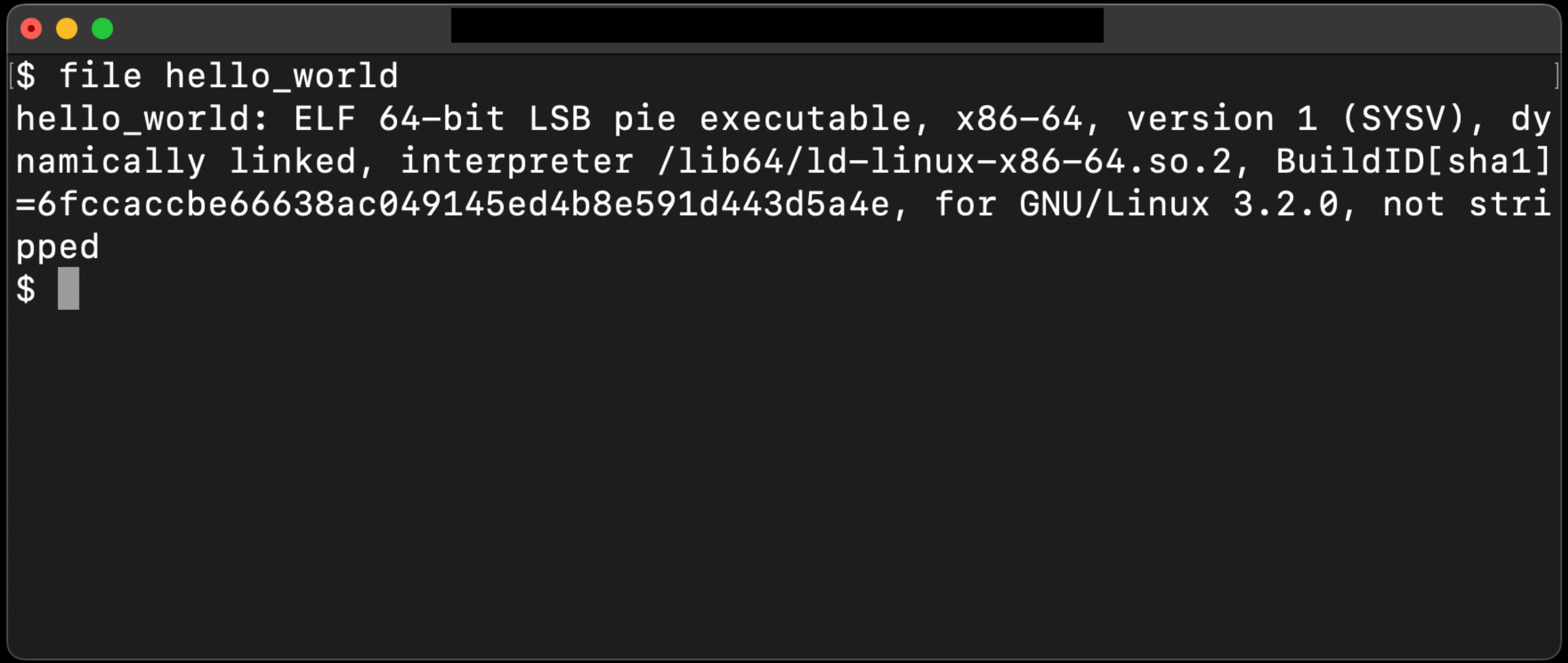
[redacted]

```
$ file hello_world
hello_world: ELF 64-bit LSB pie executable, x86-64, version 1 (SYSV), dy
namically linked, interpreter /lib64/ld-linux-x86-64.so.2, BuildID[sha1]
=6fccaccbe66638ac049145ed4b8e591d443d5a4e, for GNU/Linux 3.2.0, not stri
pped
$ 
```

Hello, world!

baby steps

- `file` command tells us a lot
  - ELF executable
  - Compiled for x86-64 CPUs
  - SYSV ABI
  - Dynamically linked
  - Not stripped



[redacted]

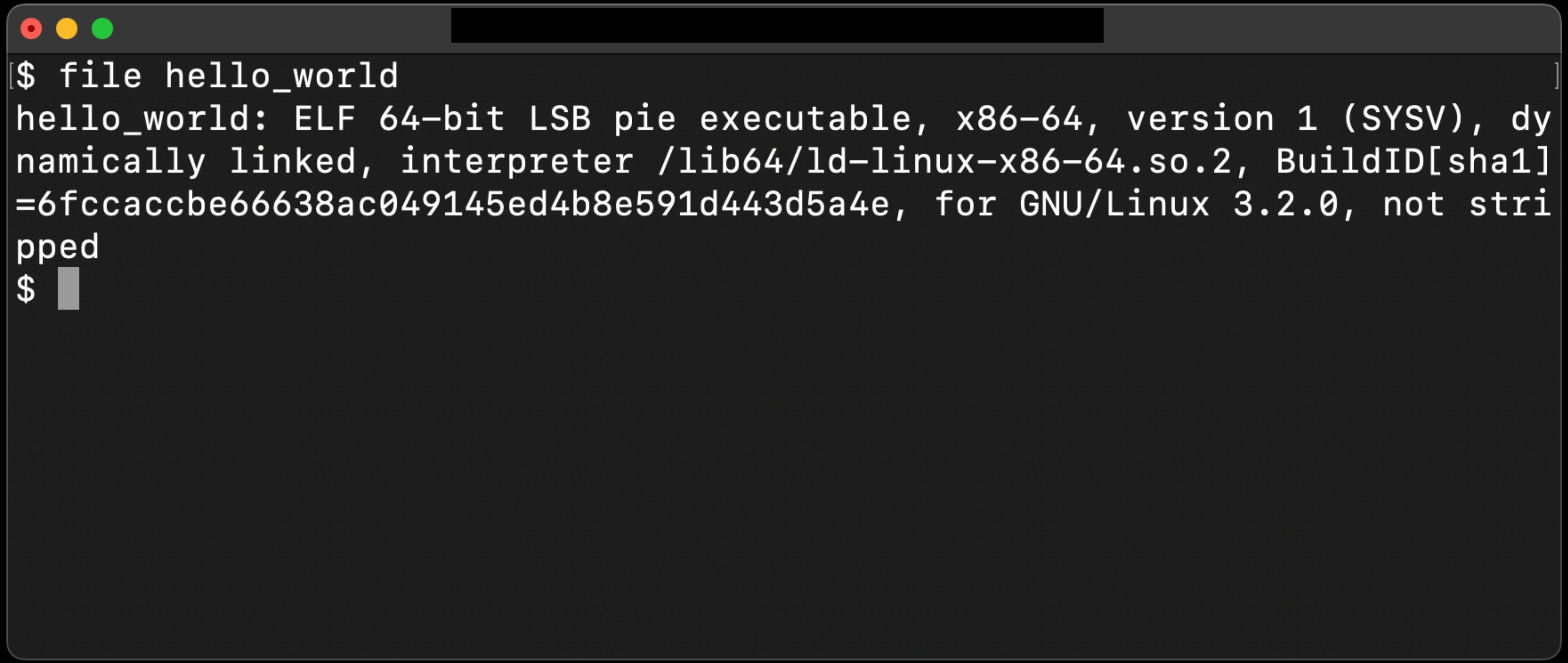
```
$ file hello_world
hello_world: ELF 64-bit LSB pie executable, x86-64, version 1 (SYSV), dy
namically linked, interpreter /lib64/ld-linux-x86-64.so.2, BuildID[sha1]
=6fccaccbe66638ac049145ed4b8e591d443d5a4e, for GNU/Linux 3.2.0, not stri
pped
$ 
```

Hello, world!

baby steps

- `file` command tells us a lot
  - ELF executable
  - Compiled for x86-64 CPUs
  - SYSV ABI
  - Dynamically linked
  - Not stripped

???



[redacted]

```
$ file hello_world
hello_world: ELF 64-bit LSB pie executable, x86-64, version 1 (SYSV), dy
namically linked, interpreter /lib64/ld-linux-x86-64.so.2, BuildID[sha1]
=6fccaccbe66638ac049145ed4b8e591d443d5a4e, for GNU/Linux 3.2.0, not stri
pped
$ 
```

CPU…

is

- *Compiled for x86-64 CPUs*

CPU…

is

- *Compiled for x86-64 CPUs*
- CPU → **C**entral **P**rocessing **U**nit

CPU…

is

- *Compiled for x86-64 CPUs*
- CPU → **C**entral **P**rocessing **U**nit
- Runs programs by executing **instructions**

CPU…

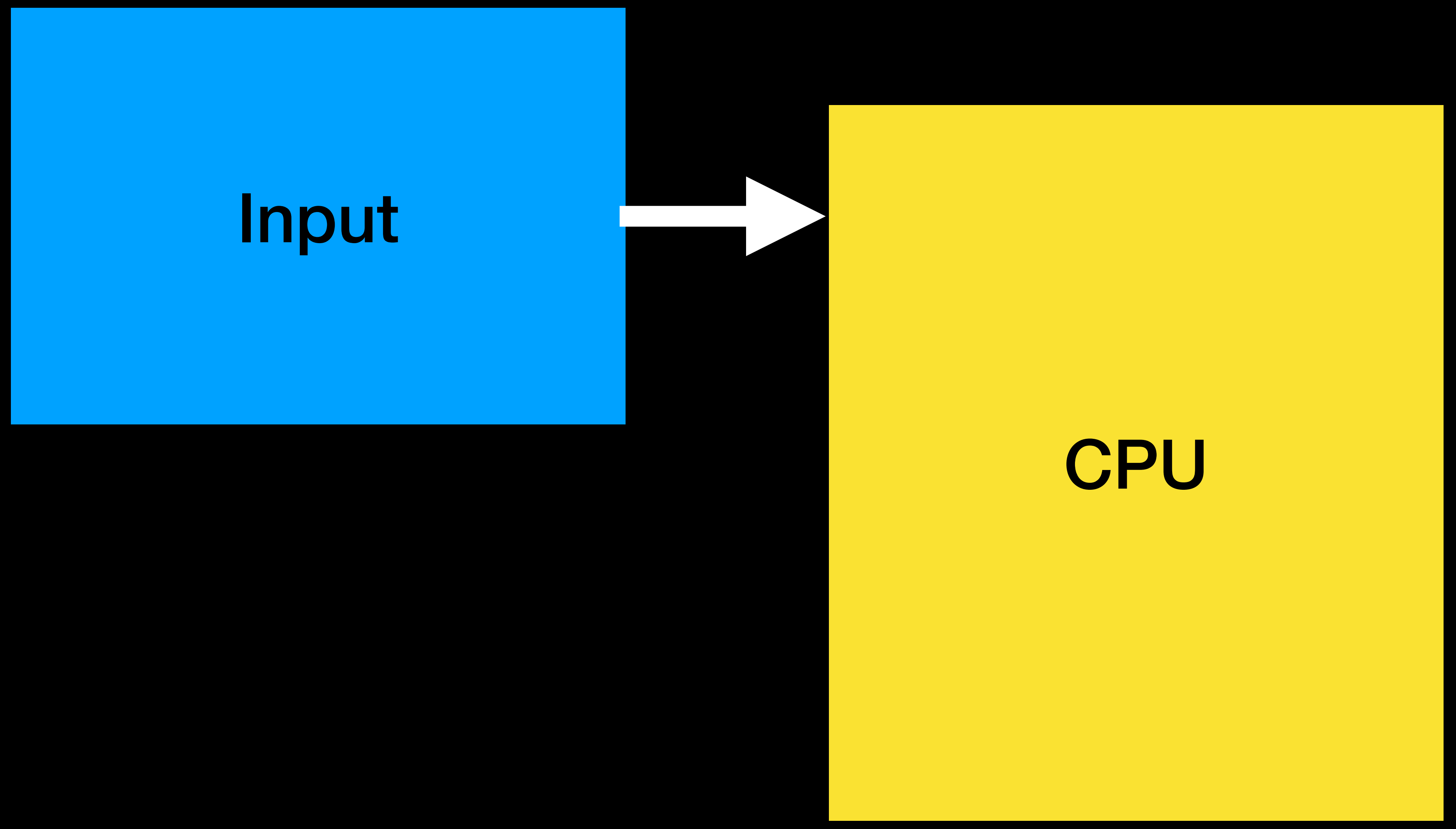
is

- *Compiled for x86-64 CPUs*
- CPU → **C**entral **P**rocessing **U**nit
- Runs programs by executing **instructions**
- One instruction at a time, **billions** per second (GHz)

CPU…

as a machine

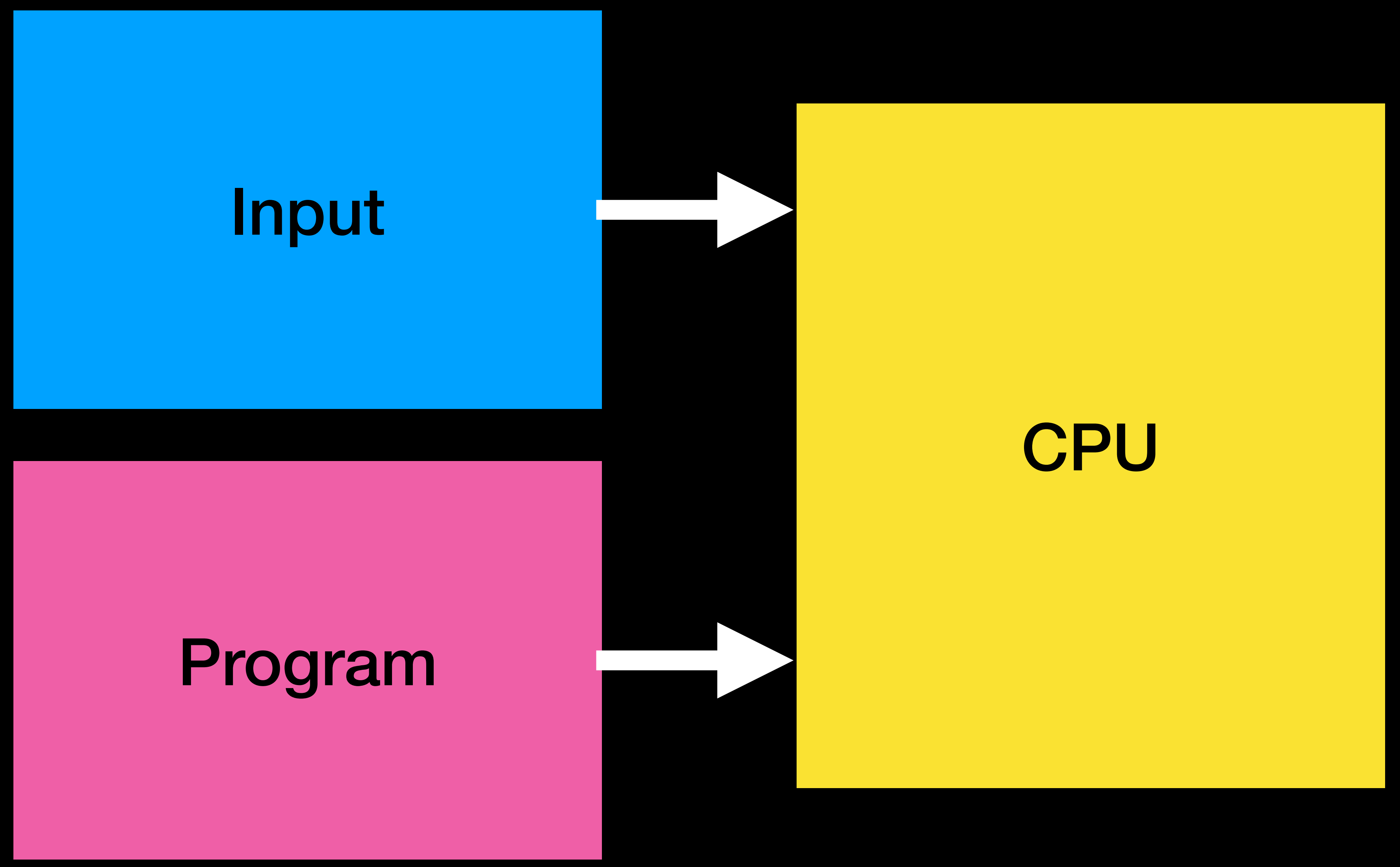
- CPU takes **input (data)**



CPU…

as a machine

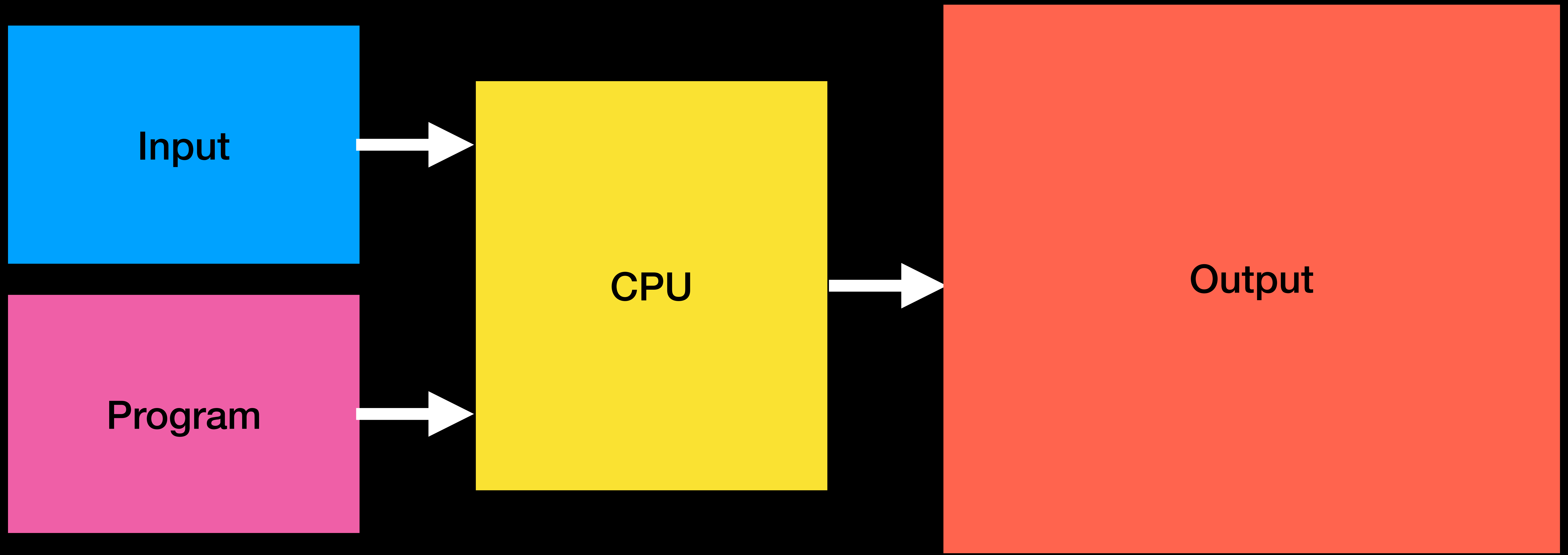
- CPU takes **input (data)**
- Follows a **program (instructions)**



CPU…

as a machine

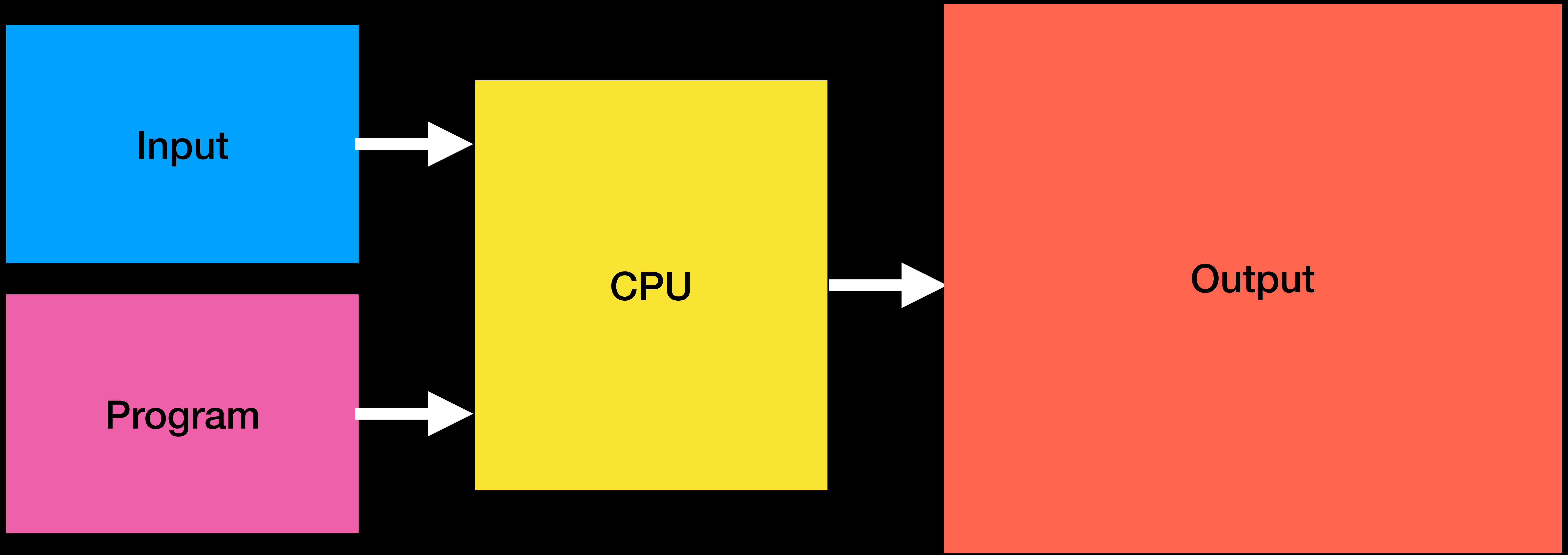
- CPU takes **input (data)**
- Follows a **program (instructions)**
- Produces **output (results)**



CPU…

as a machine

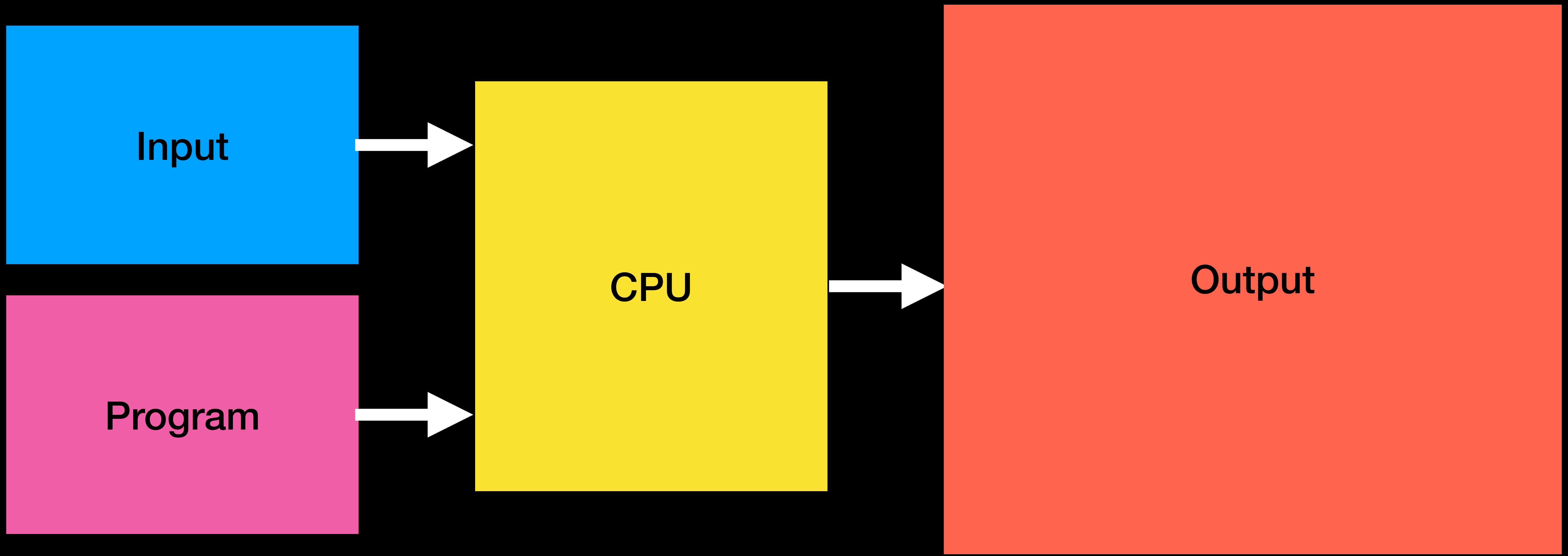
- CPU takes **input (data)**
- Follows a **program (instructions)**
- Produces **output (results)**
- That’s all!



CPU…

in reality

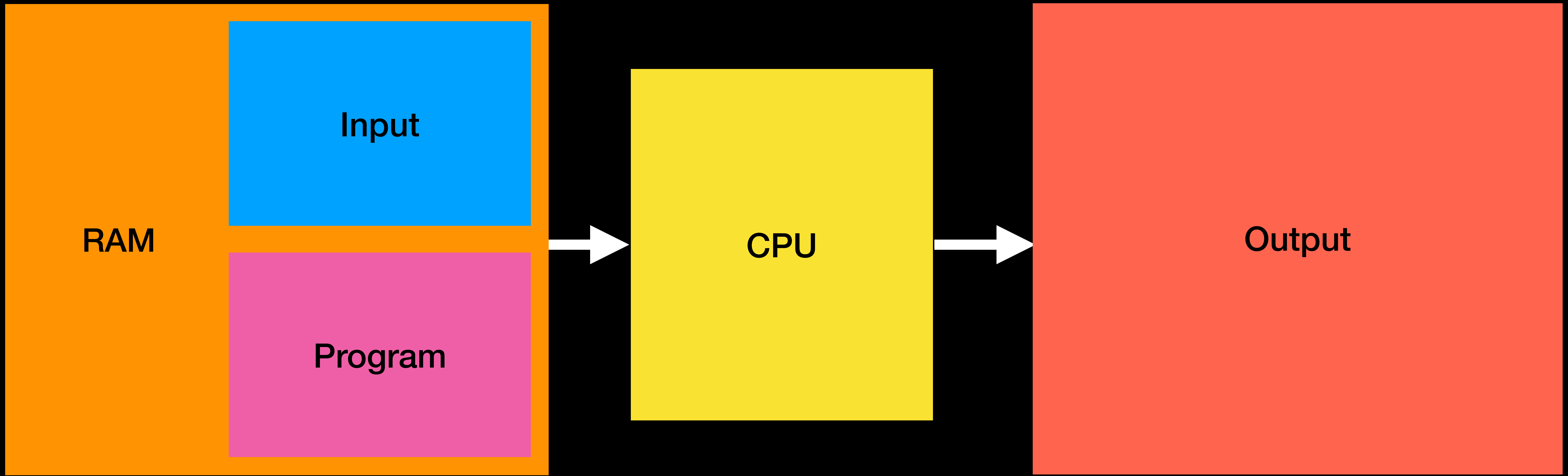
- Incredibly complex internal machinery



CPU…

in reality

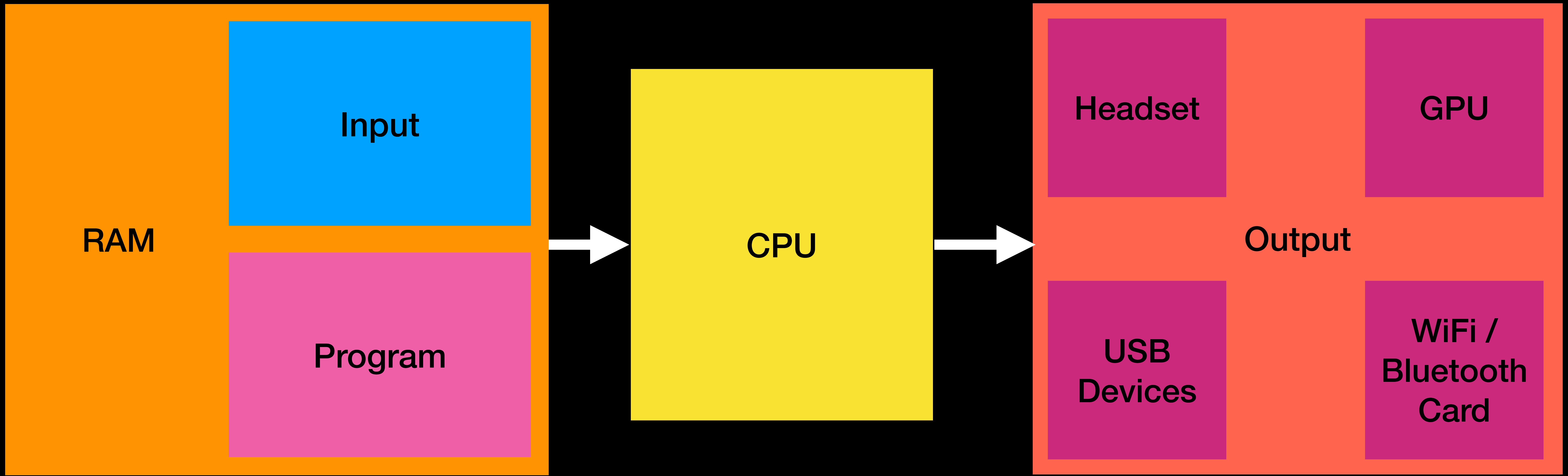
- Incredibly complex internal machinery
- Typically program and input refer to the same physical hardware (RAM)



CPU…

in reality

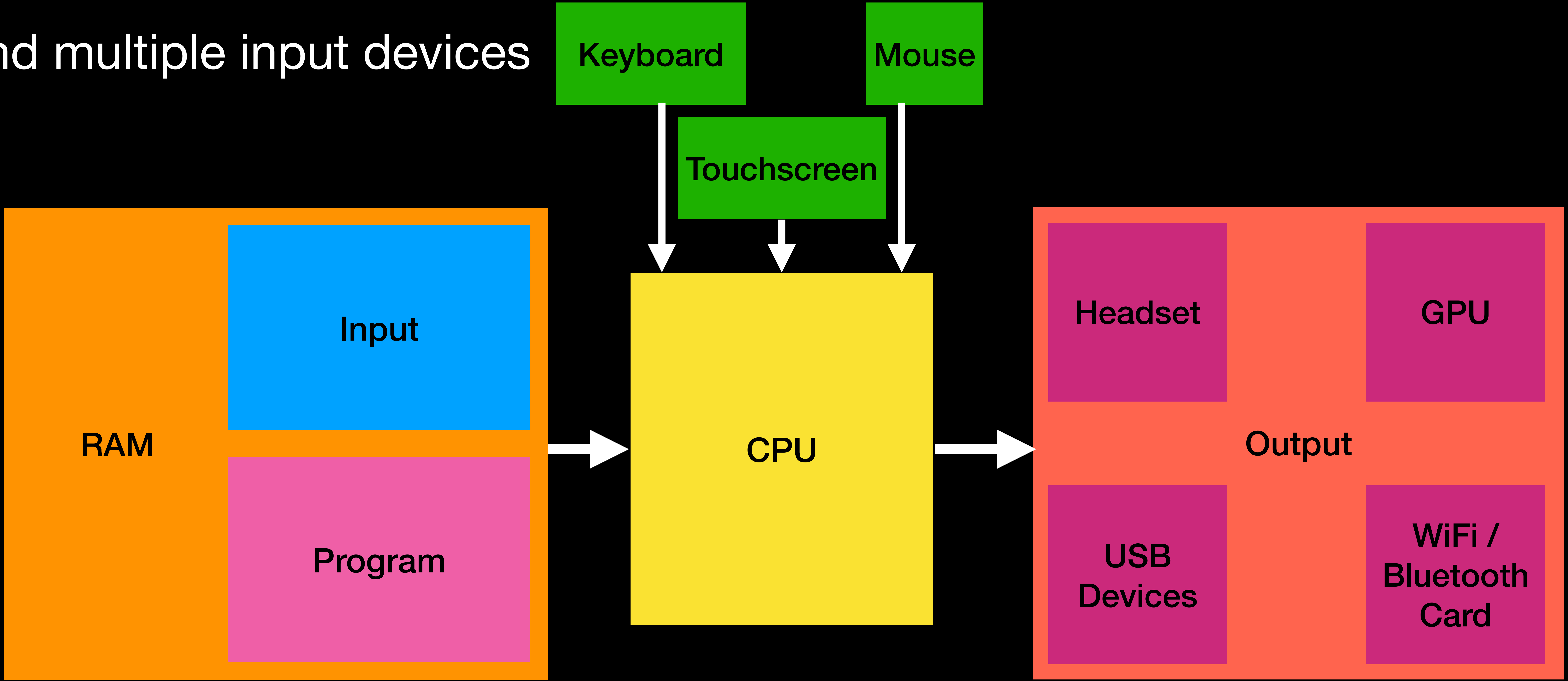
- Connect to multiple output devices (graphics, I/O devices, peripherals…)



CPU…

in reality

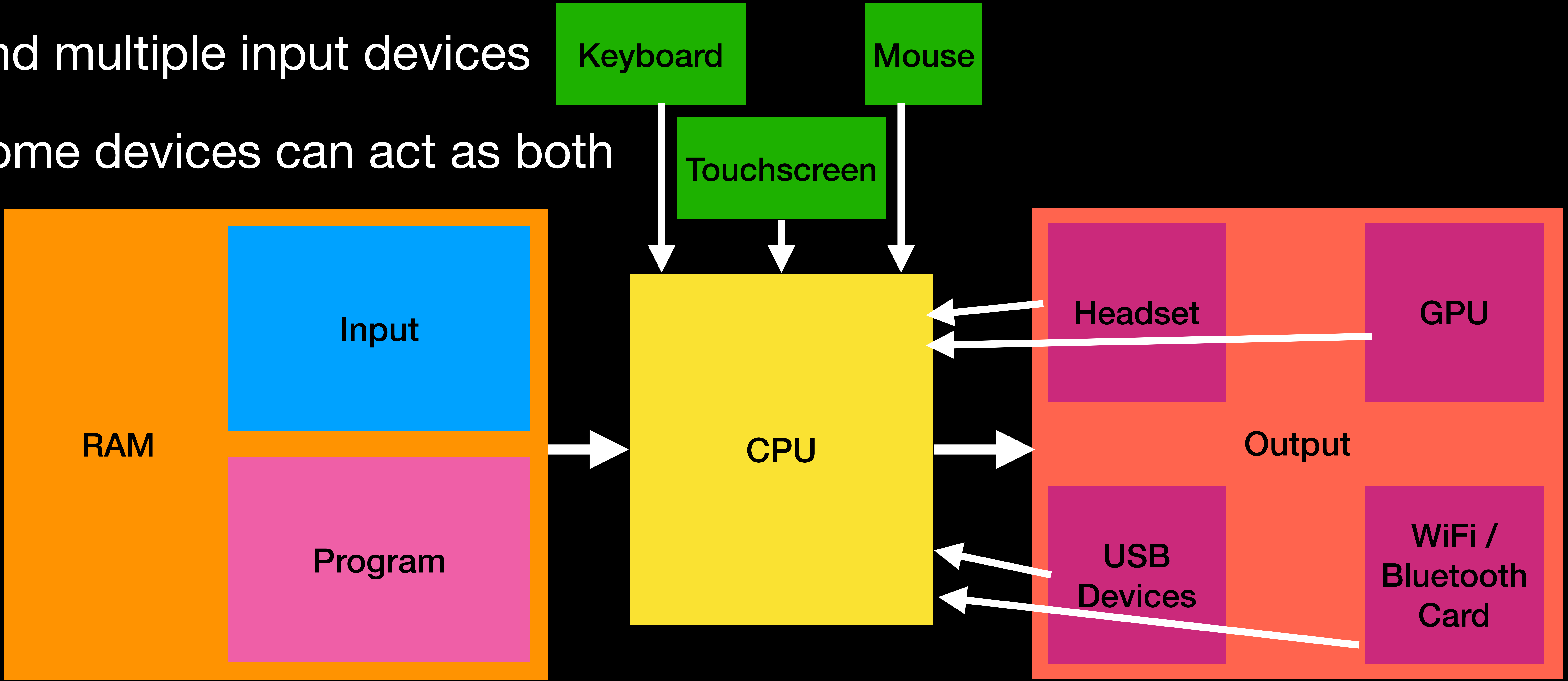
- Connect to multiple output devices (graphics, I/O devices, peripherals…)
- And multiple input devices



CPU…

in reality

- Connect to multiple output devices (graphics, I/O devices, peripherals…)
- And multiple input devices
- Some devices can act as both

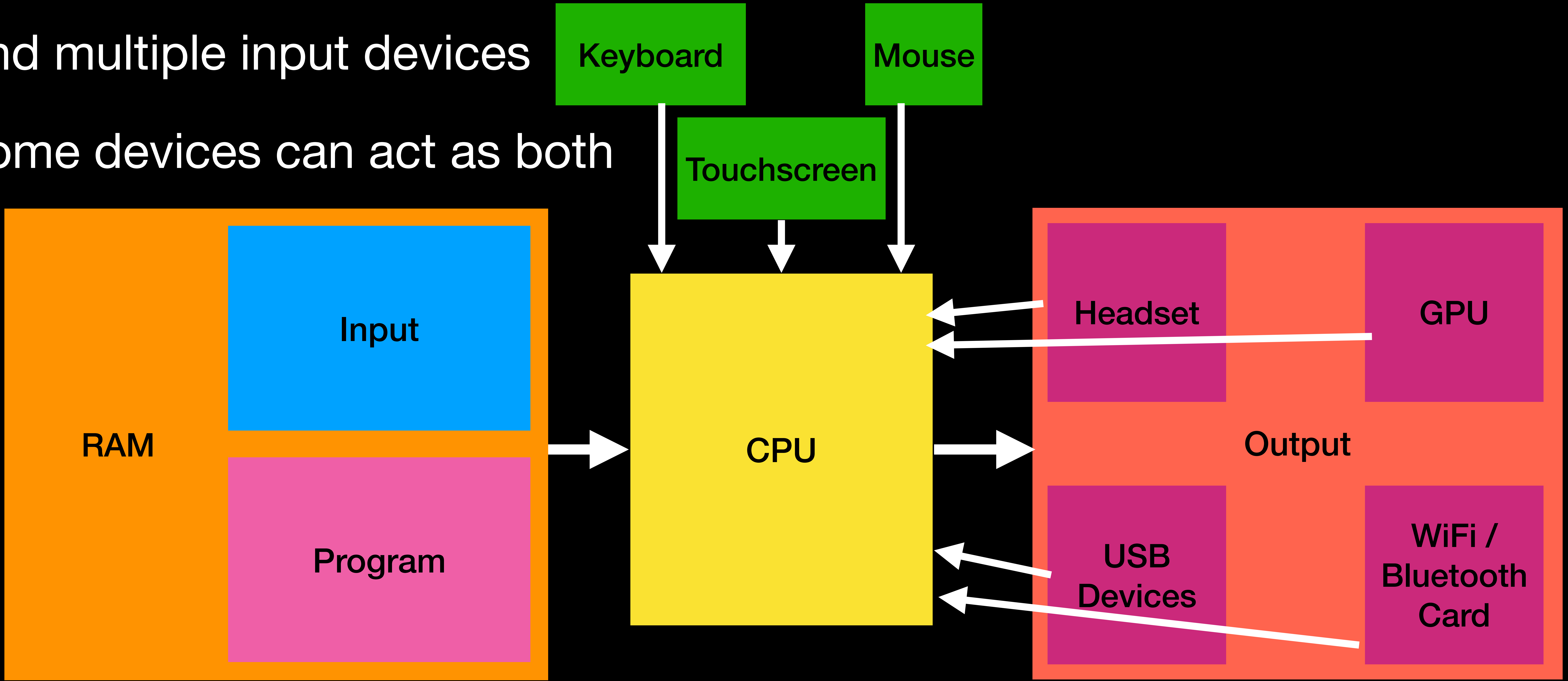


CPU…

in reality

This is somewhat of an oversimplification, but I can only fit so many arrows on screen

- Connect to multiple output devices (graphics, I/O devices, peripherals…)
- And multiple input devices
- Some devices can act as both

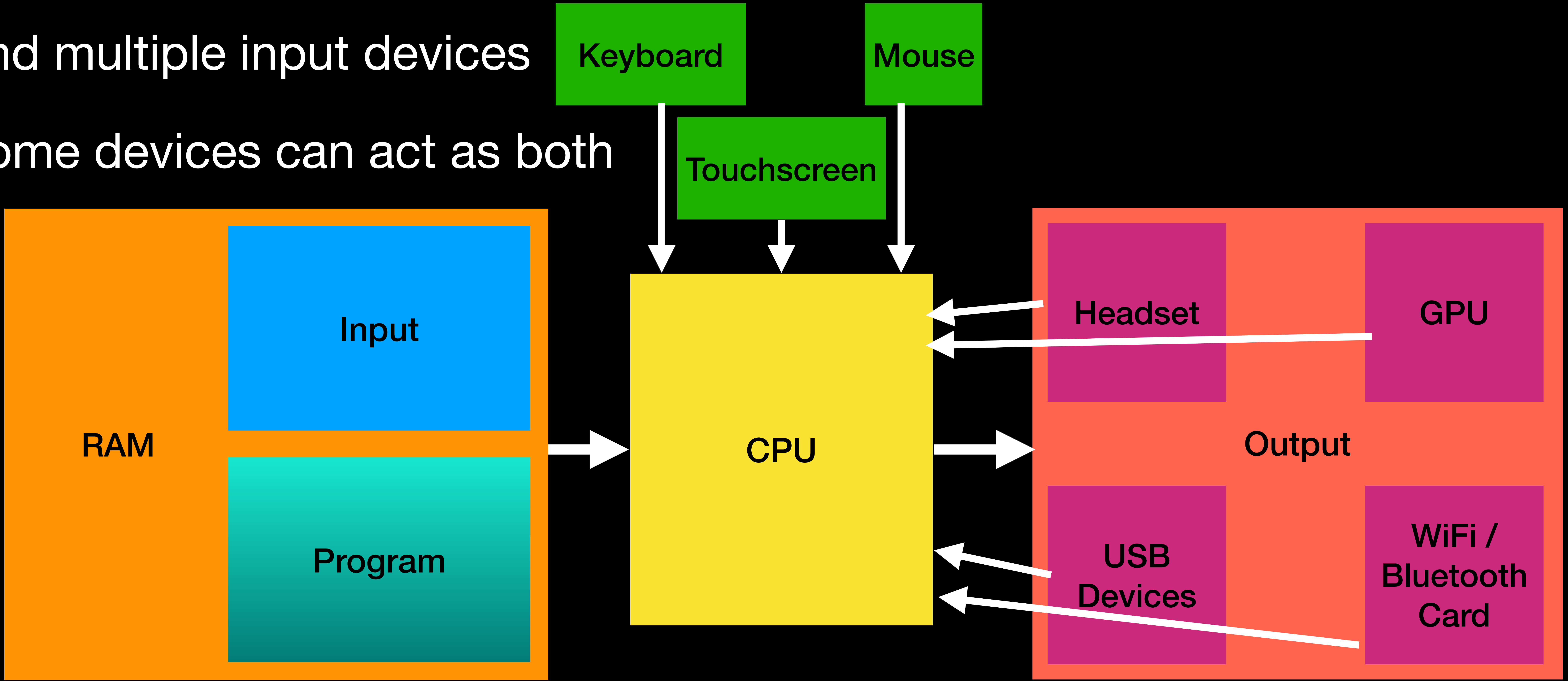


CPU…

in reality

This is somewhat of an oversimplification, but I can only fit so many arrows on screen

- Connect to multiple output devices (graphics, I/O devices, peripherals…)
- And multiple input devices
- Some devices can act as both



CPU…

does as instructed

- Programs are sequences of simple[1] instructions

CPU…

does as instructed

- Programs are sequences of simple[1] instructions

[1] instructions like `VGF2P8AFFINEINVQB` are the exception

CPU…

does as instructed

- Programs are sequences of simple[1] instructions
- Move data around (`mov, xchg,` …)

[1] instructions like `VGF2P8AFFINEINVQB` are the exception

CPU…

does as instructed

- Programs are sequences of simple[1] instructions
- Move data around (`mov, xchg,` …)
- Perform arithmetic (`add, sub, xor,` …)

[1] instructions like `VGF2P8AFFINEINVQB` are the exception

CPU…

does as instructed

- Programs are sequences of simple[1] instructions
- Move data around (`mov, xchg, …`)
- Perform arithmetic (`add, sub, xor, …`)
- Compare values (`cmp`)

[1] instructions like `VGF2P8AFFINEINVQB` are the exception

CPU…

does as instructed

- Programs are sequences of simple[1] instructions
- Move data around (`mov, xchg,` …)
- Perform arithmetic (`add, sub, xor,` …)
- Compare values (`cmp`)
- Modify control flow (`jmp, call, ret, jg, jle,` …)

[1] instructions like `VGF2P8AFFINEINVQB` are the exception

CPU…

does as instructed

- Programs are sequences of simple[1] instructions
- Move data around (`mov, xchg,` …)
- Perform arithmetic (`add, sub, xor,` …)
- Compare values (`cmp`)
- Modify control flow (`jmp, call, ret, jg, jle,` …)
- Different CPUs have different instruction “sets” (ISA)

[1] instructions like `VGF2P8AFFINEINVQB` are the exception

CPU…

does as instructed

- Programs are sequences of simple[1] instructions
- Move data around (`mov, xchg, …`)
- Perform arithmetic (`add, sub, xor, …`)
- Compare values (`cmp`)
- Modify control flow (`jmp, call, ret, jg, jle, …`)
- Different CPUs have different instruction “sets” (ISA)
  - We will focus on x86/x64 in this talk, but challenges may involve MIPS, ARM, …

[1] instructions like `VGF2P8AFFINEINVQB` are the exception

CPU…

has “registers”

- CPU keeps a set of internal “registers” that hold N-bit values

CPU…

has “registers”

- CPU keeps a set of internal “registers” that hold N-bit values
- RAX, RBX, RCX, RDX, RSI, RDI, R8→R15, RIP, CS, DS, ES, FS, GS, EFLAGS, …

CPU…

has “registers”

- CPU keeps a set of internal “registers” that hold N-bit values
- RAX, RBX, RCX, RDX, RSI, RDI, R8→R15, RIP, CS, DS, ES, FS, GS, EFLAGS, …
- Some registers have special purposes (RIP, RSP, CS/DS/ES/…, EFLAGS, …)

CPU…

has “registers”

- CPU keeps a set of internal “registers” that hold N-bit values
- RAX, RBX, RCX, RDX, RSI, RDI, R8→R15, RIP, CS, DS, ES, FS, GS, EFLAGS, …
- Some registers have special purposes (RIP, RSP, CS/DS/ES/…, EFLAGS, …)
- Registers are much faster than RAM

CPU…

has “registers”

- CPU keeps a set of internal “registers” that hold N-bit values
- RAX, RBX, RCX, RDX, RSI, RDI, R8→R15, RIP, CS, DS, ES, FS, GS, EFLAGS, …
- Some registers have special purposes (RIP, RSP, CS/DS/ES/…, EFLAGS, …)
- Registers are much faster than RAM
    - Say we want to compute (x + 7) * 2 + 14

CPU…

has “registers”

- CPU keeps a set of internal “registers” that hold N-bit values
- RAX, RBX, RCX, RDX, RSI, RDI, R8→R15, RIP, CS, DS, ES, FS, GS, EFLAGS, …
- Some registers have special purposes (RIP, RSP, CS/DS/ES/…, EFLAGS, …)
- Registers are much faster than RAM
  - Say we want to compute (x + 7) * 2 + 14
  - Much faster to load X to a register and operate on that register

CPU…

instruction syntax

- Two main conventions:

CPU…

instruction syntax

- Two main conventions:
  - Intel Syntax - `<op> <dst> <src>`

CPU…

instruction syntax

- Two main conventions:
  - Intel Syntax - `<op> <dst> <src>`
  - AT&T Syntax - `<op> <src> <dst>`

CPU…

instruction syntax

- Two main conventions:
  - Intel Syntax - `<op> <dst> <src>`
  - AT&T Syntax - `<op> <src> <dst>`
    - Also prefix registers with `%`, constants with `$`

CPU…

instruction syntax

- Two main conventions:
  - Intel Syntax - `<op> <dst> <src>`
  - AT&T Syntax - `<op> <src> <dst>`
    - Also prefix registers with `%`, constants with `$`

Intel

```
mov  rax, 2
add  rax, 3
imul rax, 5
```

CPU…

instruction syntax

- Two main conventions:
  - Intel Syntax - `<op> <dst> <src>`
  - AT&T Syntax - `<op> <src> <dst>`
    - Also prefix registers with `%`, constants with `$`

Intel

```
mov  rax, 2
add  rax, 3
imul rax, 5
```

AT&T

```
movq  $2, %rax
addq  $3, %rax
imulq $5, %rax
```

CPU…

instruction syntax

- Two main conventions:
    - Intel Syntax - `<op> <dst> <src>`
    - AT&T Syntax - `<op> <src> <dst>`
        - Also prefix registers with `%`, constants with `$`

We will use Intel syntax for our workshops, but it is entirely personal preference

Intel

```
mov  rax, 2
add  rax, 3
imul rax, 5
```

AT&T

```
movq  $2, %rax
addq  $3, %rax
imulq $5, %rax
```

CPU…

instruction syntax

- Two main conventions:
  - Intel Syntax - `<op> <dst> <src>`
  - AT&T Syntax - `<op> <src> <dst>`
    - Also prefix registers with `%`, constants with `$`

We will use Intel syntax for our workshops, but it is entirely personal preference

Intel

```
mov   rax, 2
add   rax, 3
imul  rax, 5
```

What is the value of rax after this program?

AT&T

```
movq  $2, %rax
addq  $3, %rax
imulq $5, %rax
```

CPU…

instruction syntax

```
mov  rax, 2
add  rax, 3
imul rax, 5
```

CPU…

instruction syntax

Move the value 2 into `rax`

```
mov  rax, 2
add  rax, 3
imul rax, 5
```

```
rax = 2
```

CPU…

instruction syntax

Move the value 2 into `rax`	`mov  rax, 2`	`rax = 2`
Add the value 3 to `rax`	`add  rax, 3`	`rax = 5`
	`imul rax, 5`	

CPU…

instruction syntax

Move the value 2 into `rax`	`mov rax, 2`	`rax = 2`
Add the value 3 to `rax`	`add rax, 3`	`rax = 5`
Multiply `rax` by 5	`imul rax, 5`	`rax = 25`

CPU…

control flow

- Sometimes we want to do **X** if **Y**

CPU…

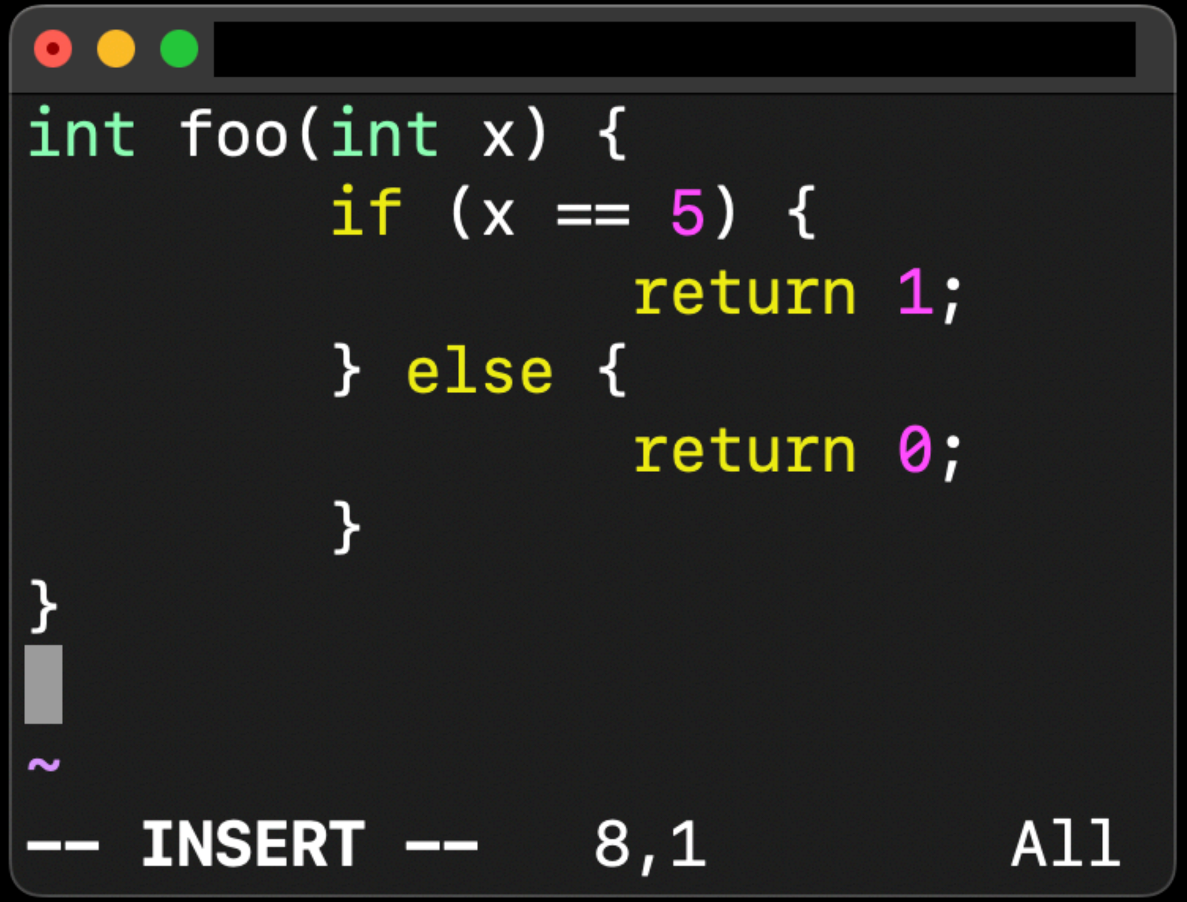
control flow

- Sometimes we want to do **X** if **Y**
  - Make cake if have ingredients

CPU…

control flow

- Sometimes we want to do **X** if **Y**
    - Make cake if have ingredients

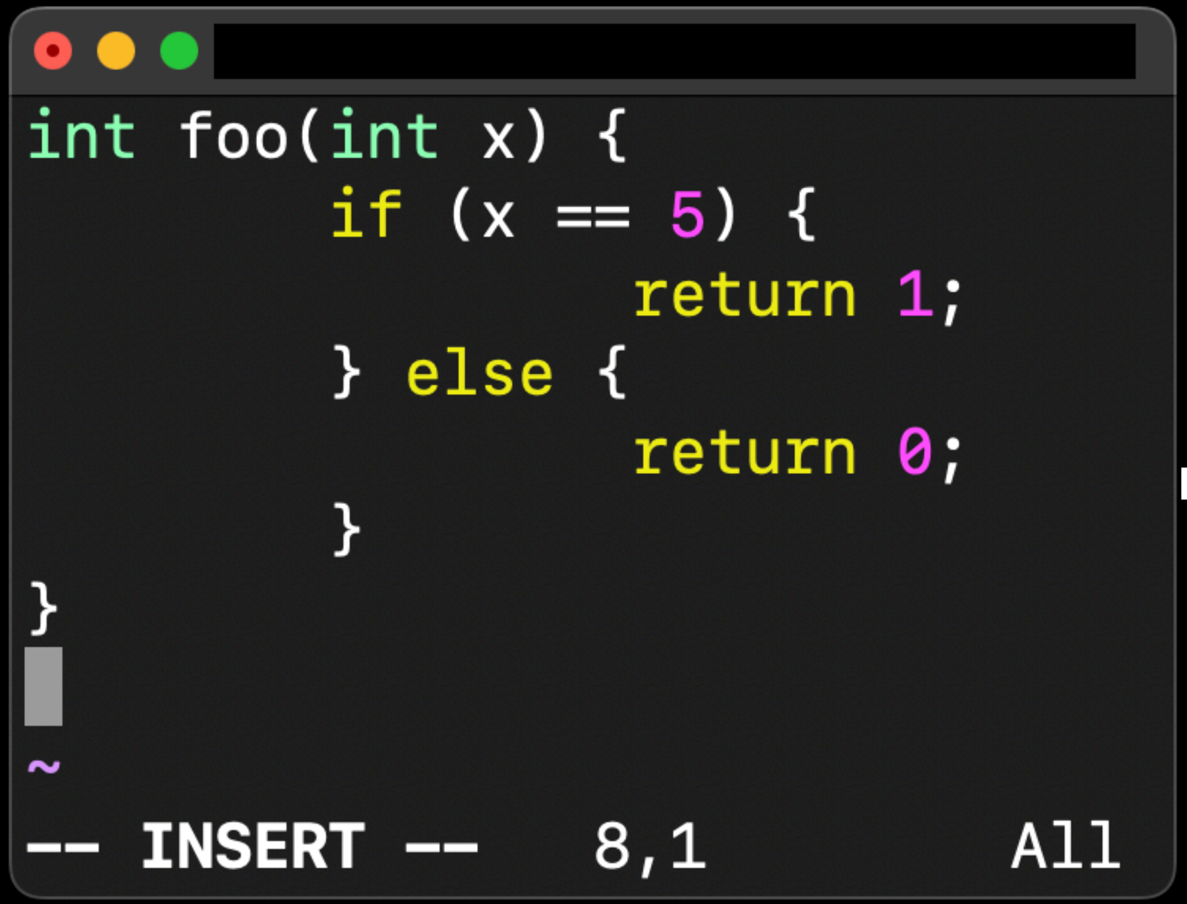


```
int foo(int x) {
        if (x == 5) {
                return 1;
        } else {
                return 0;
        }
}

~
-- INSERT --    8,1           All
```

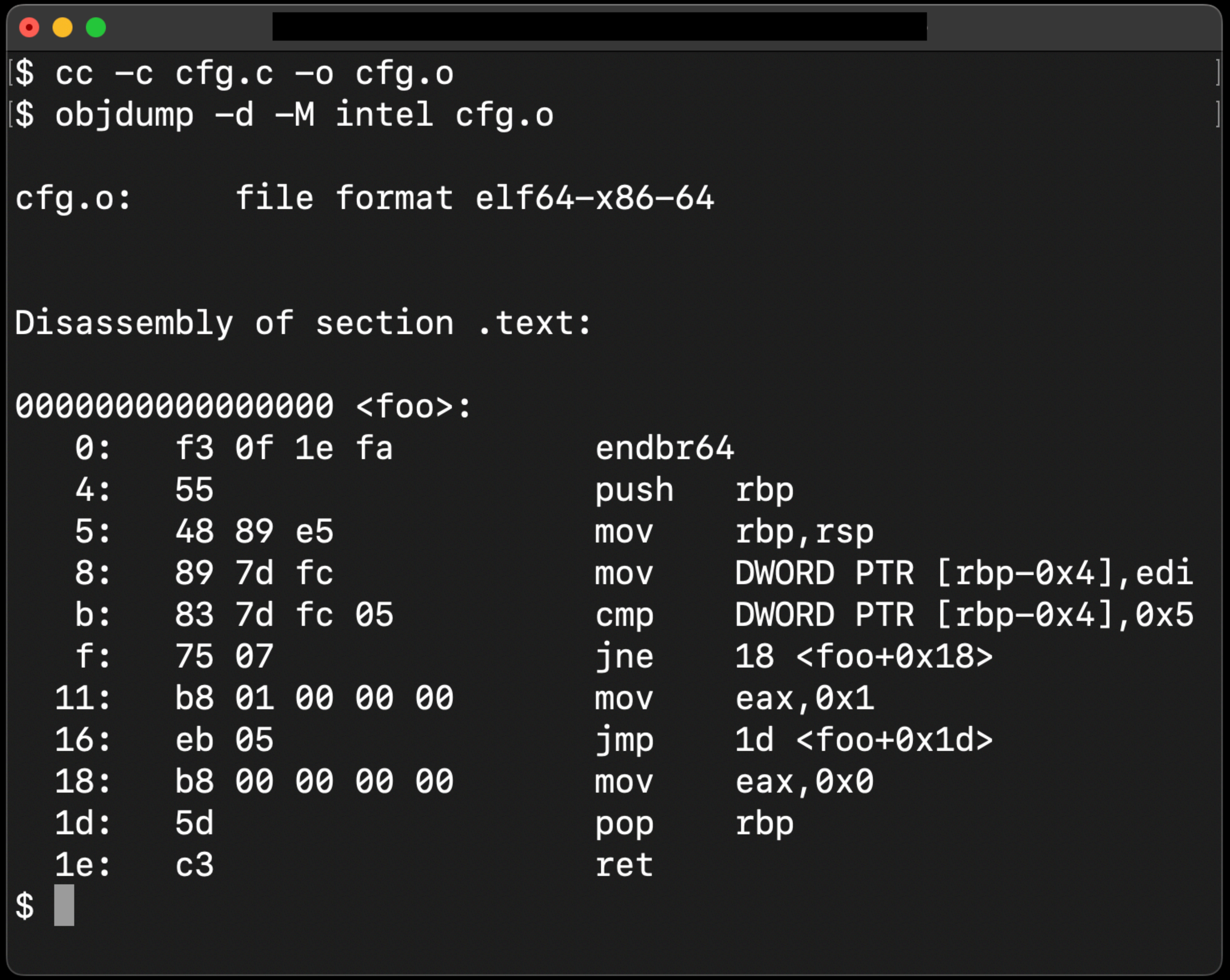
CPU…

control flow

- Sometimes we want to do X if Y
  - Make cake if have ingredients



```
int foo(int x) {
        if (x == 5) {
                return 1;
        } else {
                return 0;
        }
}

~
-- INSERT --      8,1           All
```



```
[$ cc -c cfg.c -o cfg.o
[$ objdump -d -M intel cfg.o

cfg.o:     file format elf64-x86-64


Disassembly of section .text:

0000000000000000 <foo>:
   0:	f3 0f 1e fa          	endbr64
   4:	55                   	push   rbp
   5:	48 89 e5             	mov    rbp,rsp
   8:	89 7d fc             	mov    DWORD PTR [rbp-0x4],edi
   b:	83 7d fc 05          	cmp    DWORD PTR [rbp-0x4],0x5
   f:	75 07                	jne    18 <foo+0x18>
  11:	b8 01 00 00 00       	mov    eax,0x1
  16:	eb 05                	jmp    1d <foo+0x1d>
  18:	b8 00 00 00 00       	mov    eax,0x0
  1d:	5d                   	pop    rbp
  1e:	c3                   	ret
$
```

CPU…

control flow

```
 0:   endbr64
 4:   push   rbp
 5:   mov    rbp, rsp
 8:   mov    DWORD PTR [rbp-0x4], edi
 b:   cmp    DWORD PTR [rbp-0x4], 0x5
 f:   jne    18 <foo+0x18>
11:   mov    eax, 0x1
16:   jmp    1d <foo+0x1d>
18:   mov    eax, 0x0
1d:   pop    rbp
1e:   ret
```

CPU…

control flow

Function entry →

```
 0:  endbr64
 4:  push    rbp
 5:  mov     rbp, rsp
 8:  mov     DWORD PTR [rbp-0x4], edi
 b:  cmp     DWORD PTR [rbp-0x4], 0x5
 f:  jne     18 <foo+0x18>
11:  mov     eax, 0x1
16:  jmp     1d <foo+0x1d>
18:  mov     eax, 0x0
1d:  pop     rbp
1e:  ret
```

CPU…

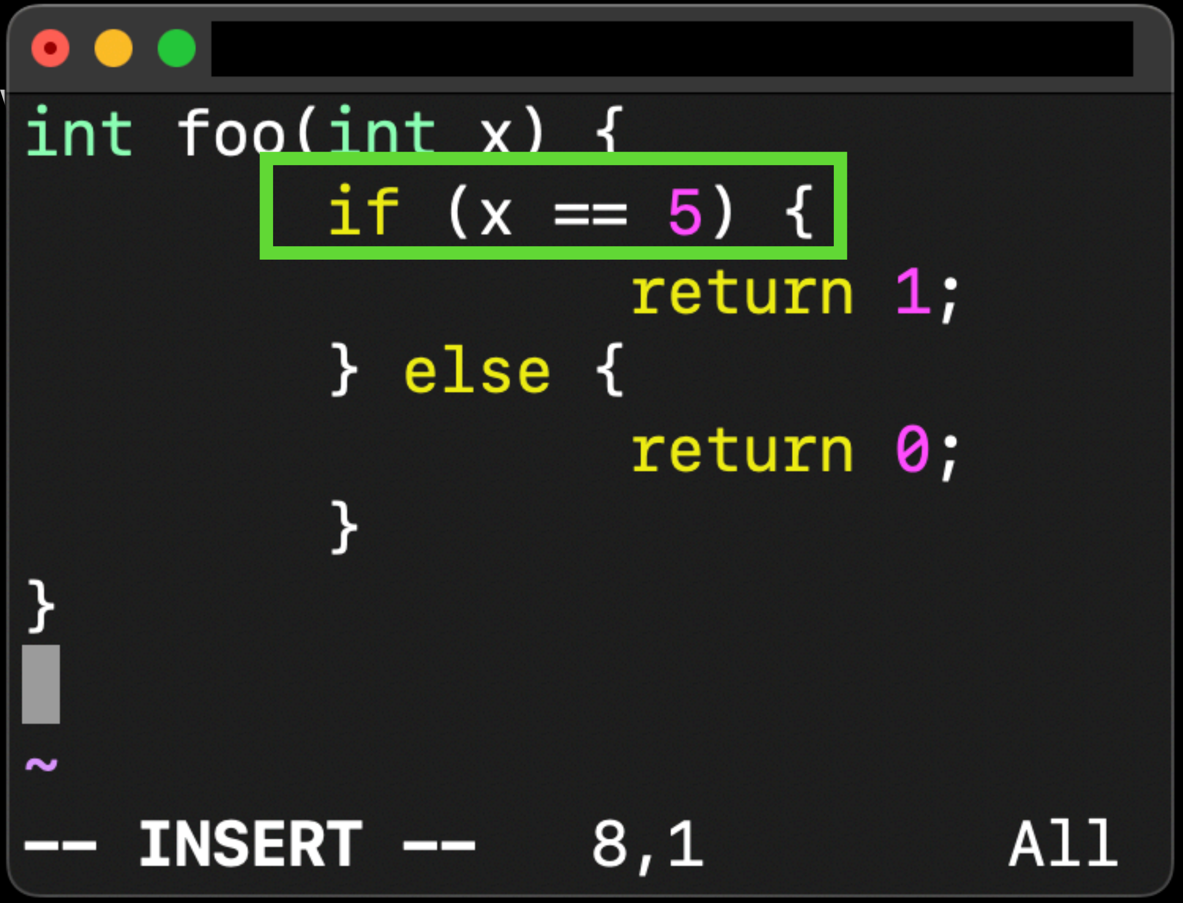
control flow

Set up stack
(more on this later!)

```
 0:   endbr64
 4:   push    rbp
 5:   mov     rbp, rsp
 8:   mov     DWORD PTR [rbp-0x4], edi
 b:   cmp     DWORD PTR [rbp-0x4], 0x5
 f:   jne     18 <foo+0x18>
11:   mov     eax, 0x1
16:   jmp     1d <foo+0x1d>
18:   mov     eax, 0x0
1d:   pop     rbp
1e:   ret
```

CPU…

control flo



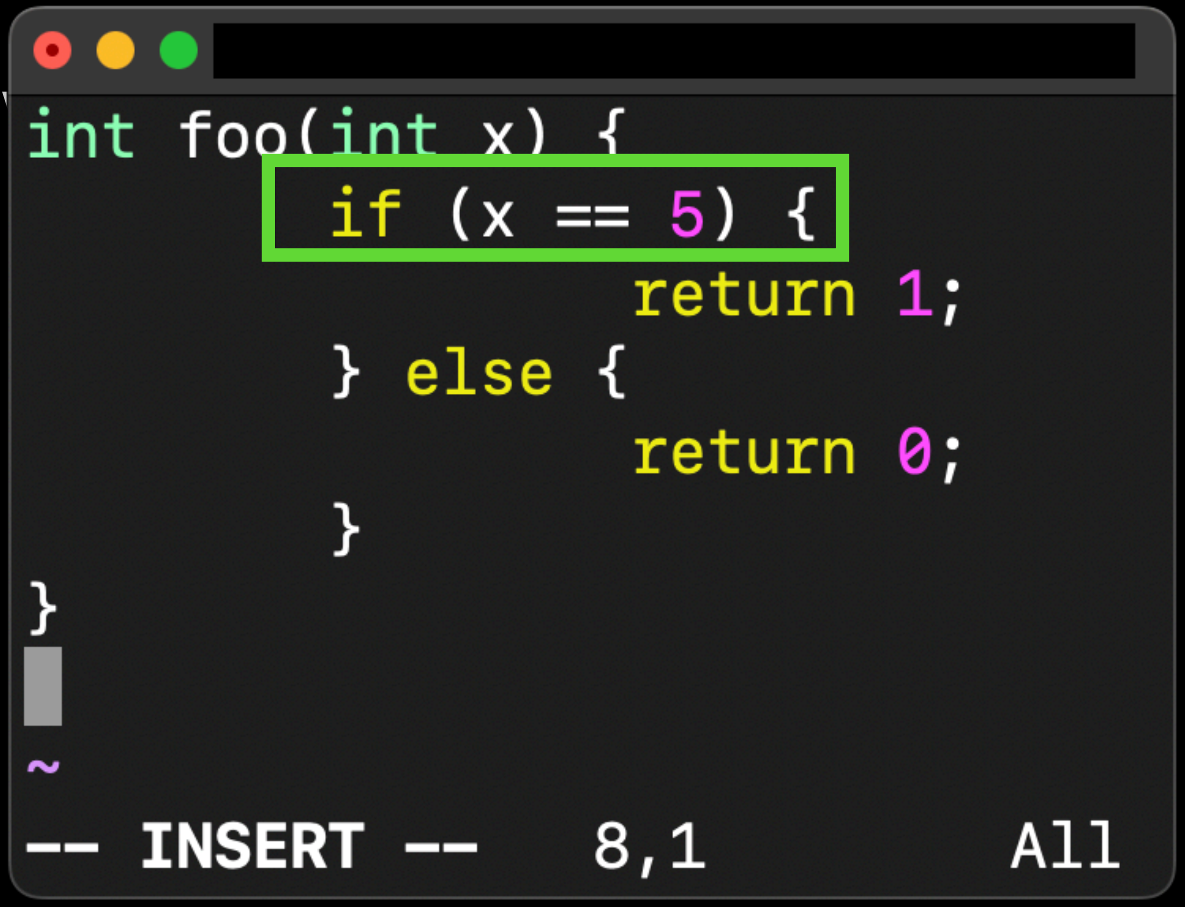
```
int foo(int x) {
        if (x == 5) {
                return 1;
        } else {
                return 0;
        }
}

~
-- INSERT --      8,1           All
```

Compare x to 5 →

```
 0:  endbr64
 4:  push   rbp
 5:  mov    rbp, rsp
 8:  mov    DWORD PTR [rbp-0x4], edi
 b:  cmp    DWORD PTR [rbp-0x4], 0x5
 f:  jne    18 <foo+0x18>
11:  mov    eax, 0x1
16:  jmp    1d <foo+0x1d>
18:  mov    eax, 0x0
1d:  pop    rbp
1e:  ret
```

CPU…

control flo



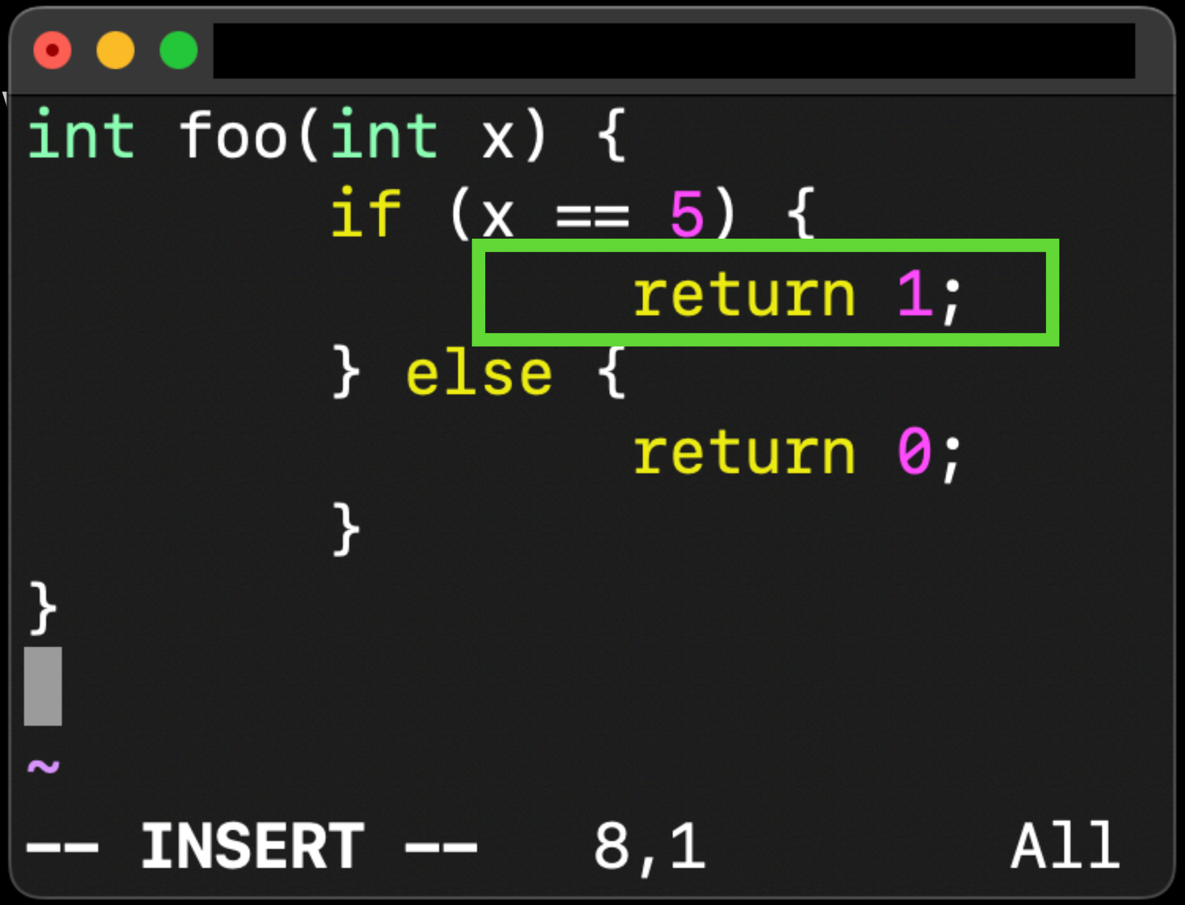
```
int foo(int x) {
        if (x == 5) {
                return 1;
        } else {
                return 0;
        }
}

~
-- INSERT --      8,1           All
```

If not equal (ne), jump (j) to instruction at 0x18

```
 0:   endbr64
 4:   push    rbp
 5:   mov     rbp, rsp
 8:   mov     DWORD PTR [rbp-0x4], edi
 b:   cmp     DWORD PTR [rbp-0x4], 0x5
 f:   jne     18 <foo+0x18>
11:   mov     eax, 0x1
16:   jmp     1d <foo+0x1d>
18:   mov     eax, 0x0
1d:   pop     rbp
1e:   ret
```

CPU…

control flo



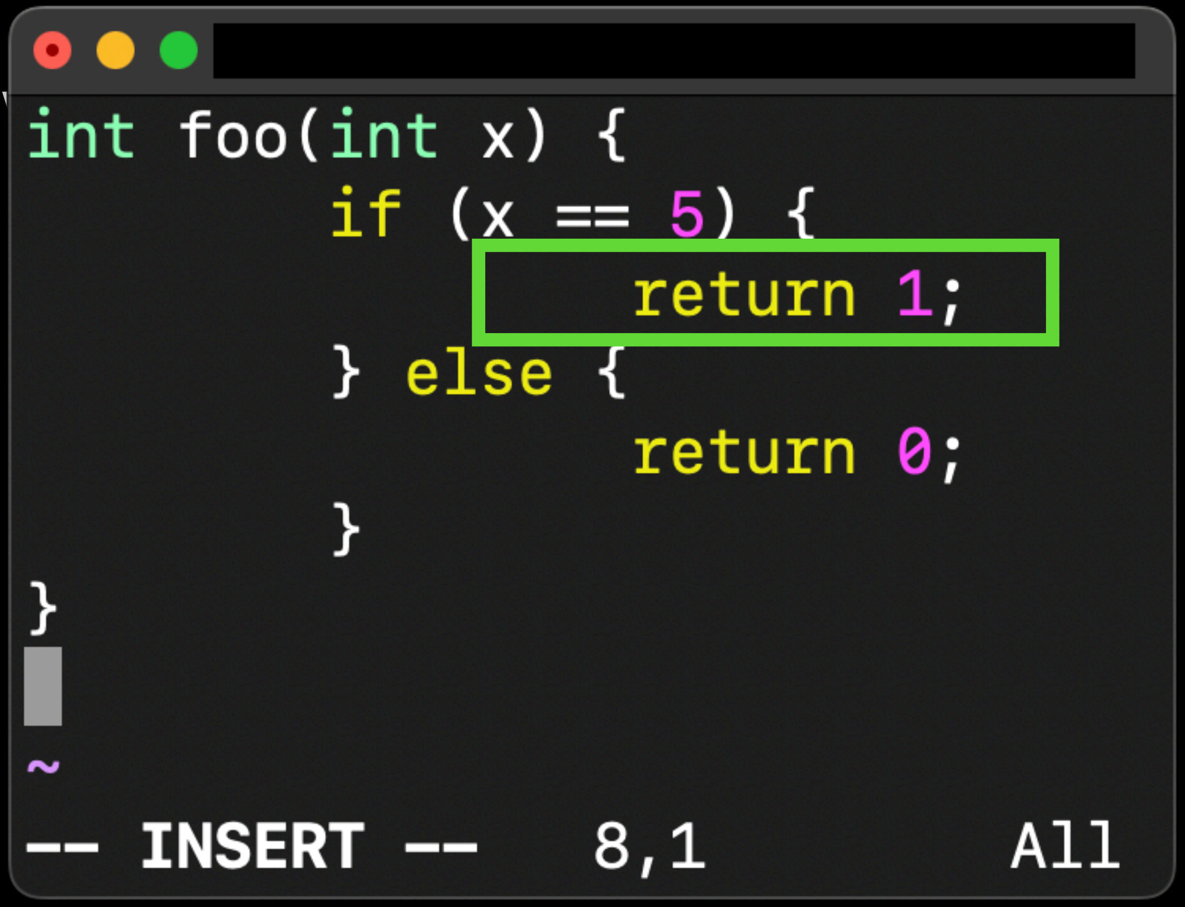
```
int foo(int x) {
        if (x == 5) {
                return 1;
        } else {
                return 0;
        }
}

~
-- INSERT --    8,1           All
```

```
 0:   endbr64
 4:   push    rbp
 5:   mov     rbp, rsp
 8:   mov     DWORD PTR [rbp-0x4], edi
 b:   cmp     DWORD PTR [rbp-0x4], 0x5
 f:   jne     18 <foo+0x18>
11:   mov     eax, 0x1
16:   jmp     1d <foo+0x1d>
18:   mov     eax, 0x0
1d:   pop     rbp
1e:   ret
```

True branch → 11: mov eax, 0x1

CPU…

control flo



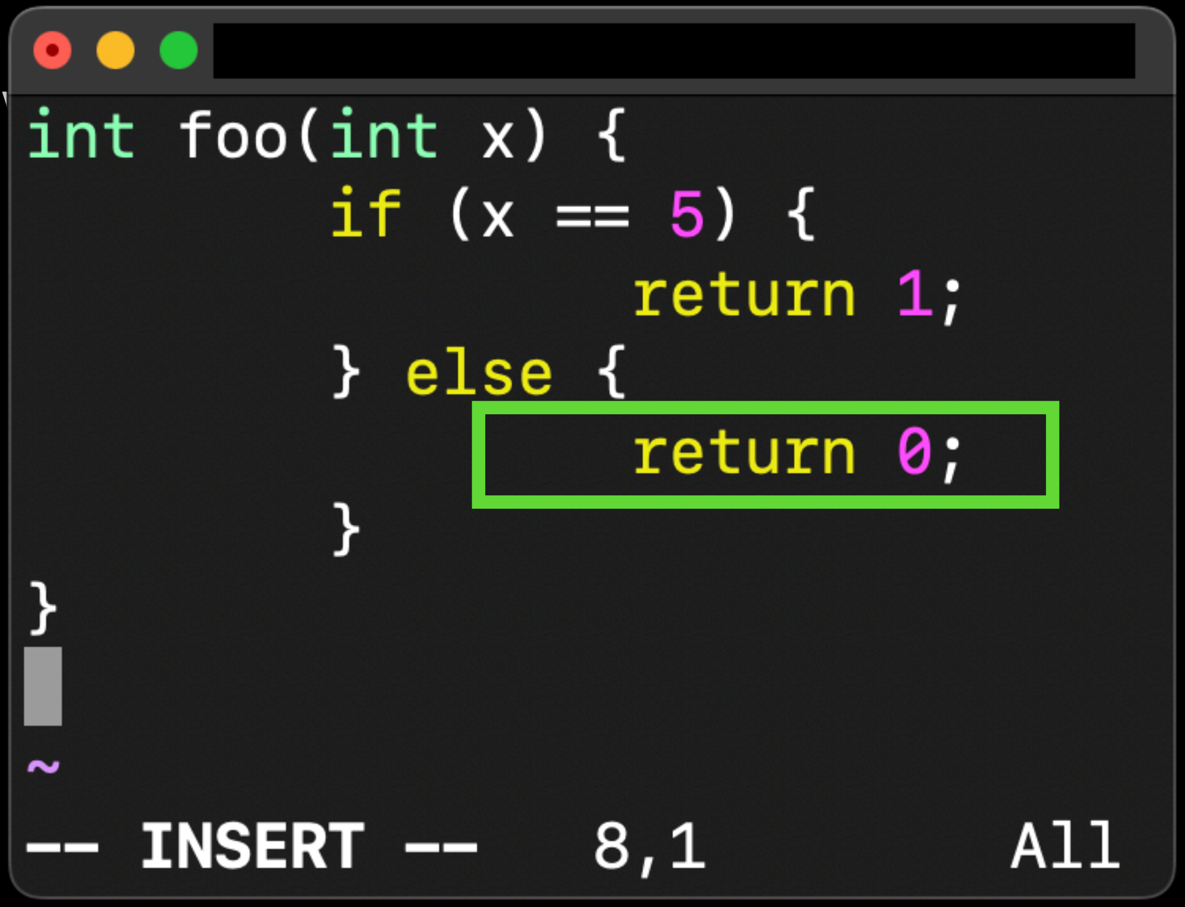
```
int foo(int x) {
        if (x == 5) {
                return 1;
        } else {
                return 0;
        }
}

~
-- INSERT --      8,1           All
```

```
 0:   endbr64
 4:   push    rbp
 5:   mov     rbp, rsp
 8:   mov     DWORD PTR [rbp-0x4], edi
 b:   cmp     DWORD PTR [rbp-0x4], 0x5
 f:   jne     18 <foo+0x18>
11:   mov     eax, 0x1
16:   jmp     1d <foo+0x1d>
18:   mov     eax, 0x0
1d:   pop     rbp
1e:   ret
```

Jump to instruction at address `0x1d`

CPU…

control flo



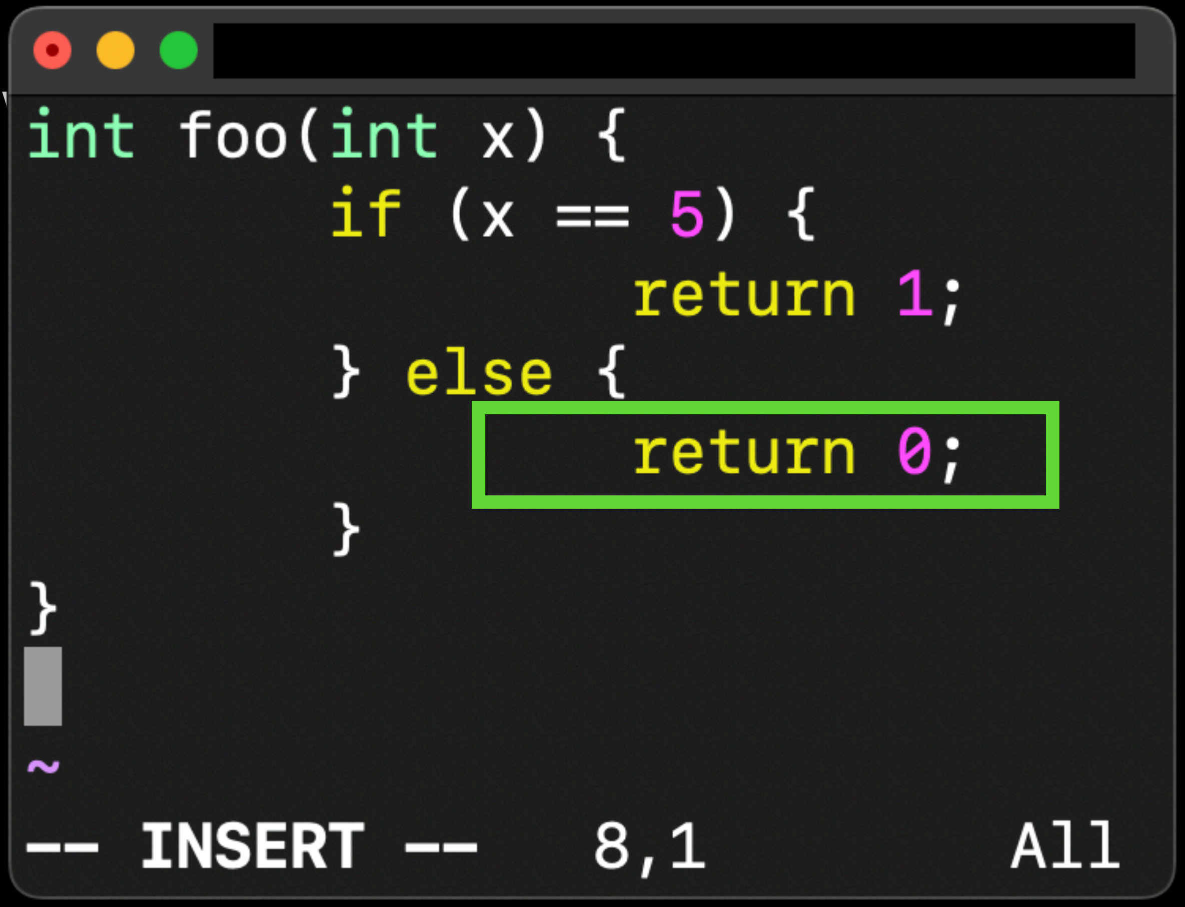
```
int foo(int x) {
        if (x == 5) {
                return 1;
        } else {
                return 0;
        }
}

~
-- INSERT --     8,1          All
```

```
 0:   endbr64
 4:   push    rbp
 5:   mov     rbp, rsp
 8:   mov     DWORD PTR [rbp-0x4], edi
 b:   cmp     DWORD PTR [rbp-0x4], 0x5
 f:   jne     18 <foo+0x18>
11:   mov     eax, 0x1
16:   jmp     1d <foo+0x1d>
18:   mov     eax, 0x0
1d:   pop     rbp
1e:   ret
```

False branch →

CPU…

control flo



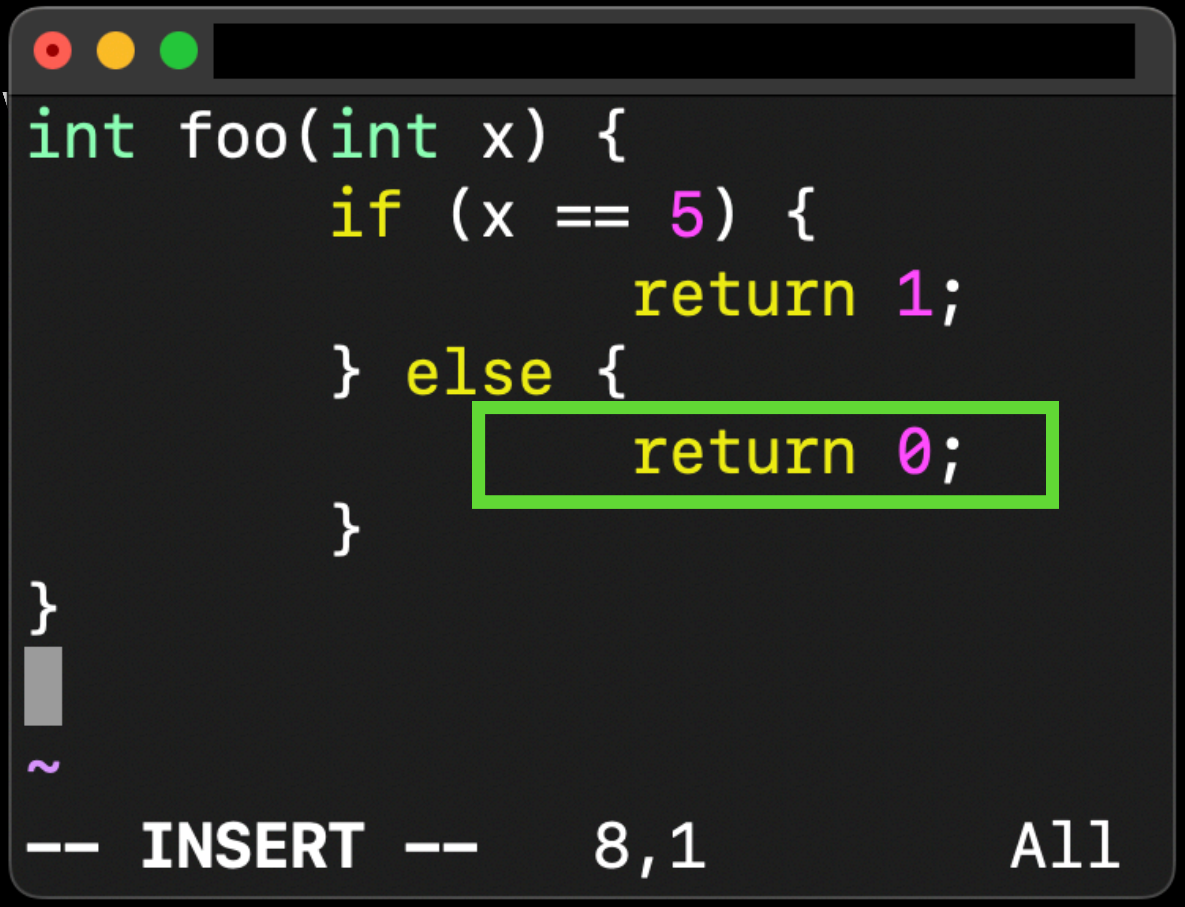
```
int foo(int x) {
        if (x == 5) {
                return 1;
        } else {
                return 0;
        }
}
~
-- INSERT --      8,1           All
```

```
 0:   endbr64
 4:   push    rbp
 5:   mov     rbp, rsp
 8:   mov     DWORD PTR [rbp-0x4], edi
 b:   cmp     DWORD PTR [rbp-0x4], 0x5
 f:   jne     18 <foo+0x18>
11:   mov     eax, 0x1
16:   jmp     1d <foo+0x1d>
18:   mov     eax, 0x0
1d:   pop     rbp
1e:   ret
```

Leave function (return value is stored in `rax`)

CPU…

control flo



```
int foo(int x) {
        if (x == 5) {
                return 1;
        } else {
                return 0;
        }
}

~
-- INSERT --      8,1          All
```

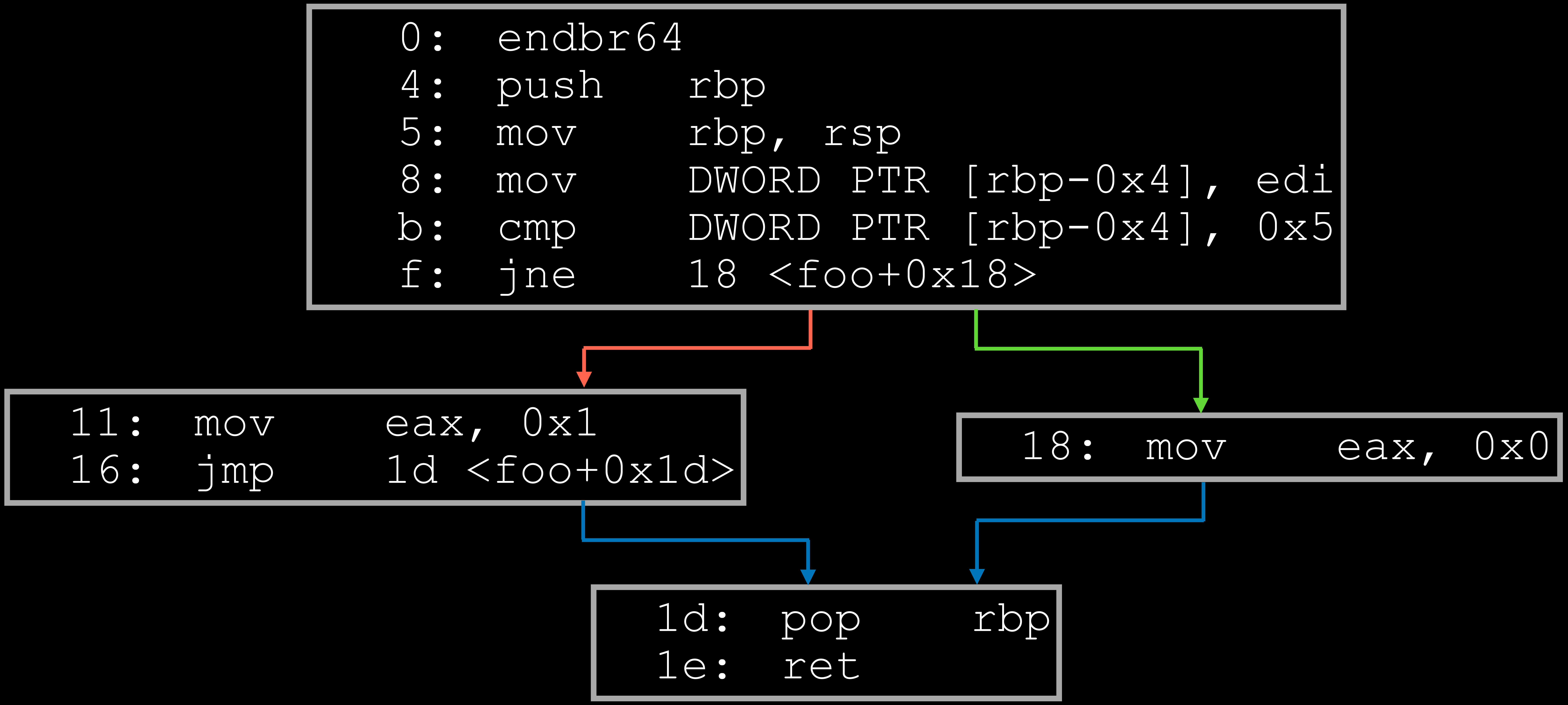
```
 0:   endbr64
 4:   push    rbp
 5:   mov     rbp, rsp
 8:   mov     DWORD PTR [rbp-0x4], edi
 b:   cmp     DWORD PTR [rbp-0x4], 0x5
 f:   jne     18 <foo+0x18>
11:   mov     eax, 0x1
16:   jmp     1d <foo+0x1d>
18:   mov     eax, 0x0
1d:   pop     rbp
1e:   ret
```

This kinda sucks to read, doesn't it?

Leave function (return value is stored in `rax`) →

CPU…

control flow graphs



```
   0:   endbr64
   4:   push   rbp
   5:   mov    rbp, rsp
   8:   mov    DWORD PTR [rbp-0x4], edi
   b:   cmp    DWORD PTR [rbp-0x4], 0x5
   f:   jne    18 <foo+0x18>
```

```
  11:   mov    eax, 0x1
  16:   jmp    1d <foo+0x1d>
```

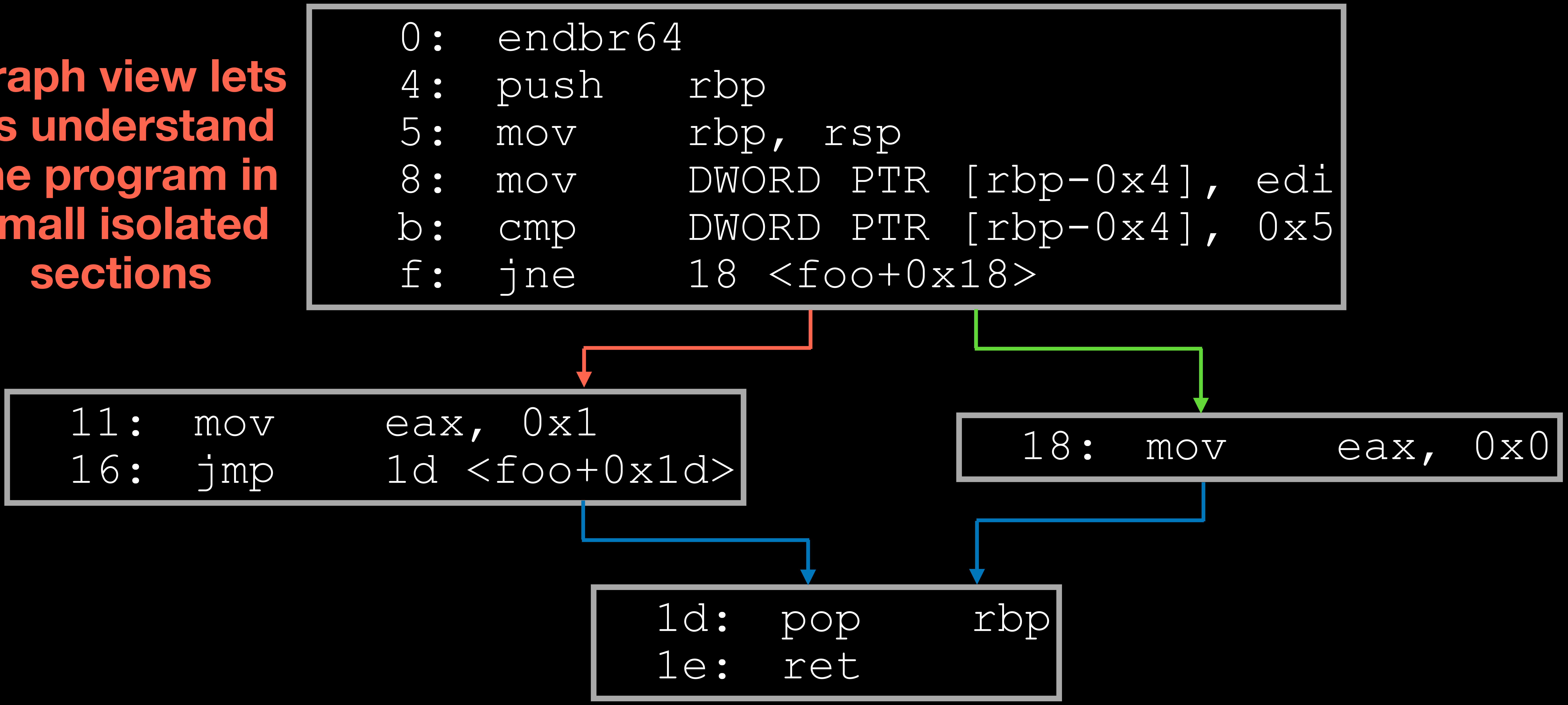
```
  18:   mov    eax, 0x0
```

```
  1d:   pop    rbp
  1e:   ret
```

CPU…

control flow graphs

Graph view lets us understand the program in small isolated sections



```
 0:   endbr64
 4:   push    rbp
 5:   mov     rbp, rsp
 8:   mov     DWORD PTR [rbp-0x4], edi
 b:   cmp     DWORD PTR [rbp-0x4], 0x5
 f:   jne     18 <foo+0x18>
```

```
11:   mov     eax, 0x1
16:   jmp     1d <foo+0x1d>
```

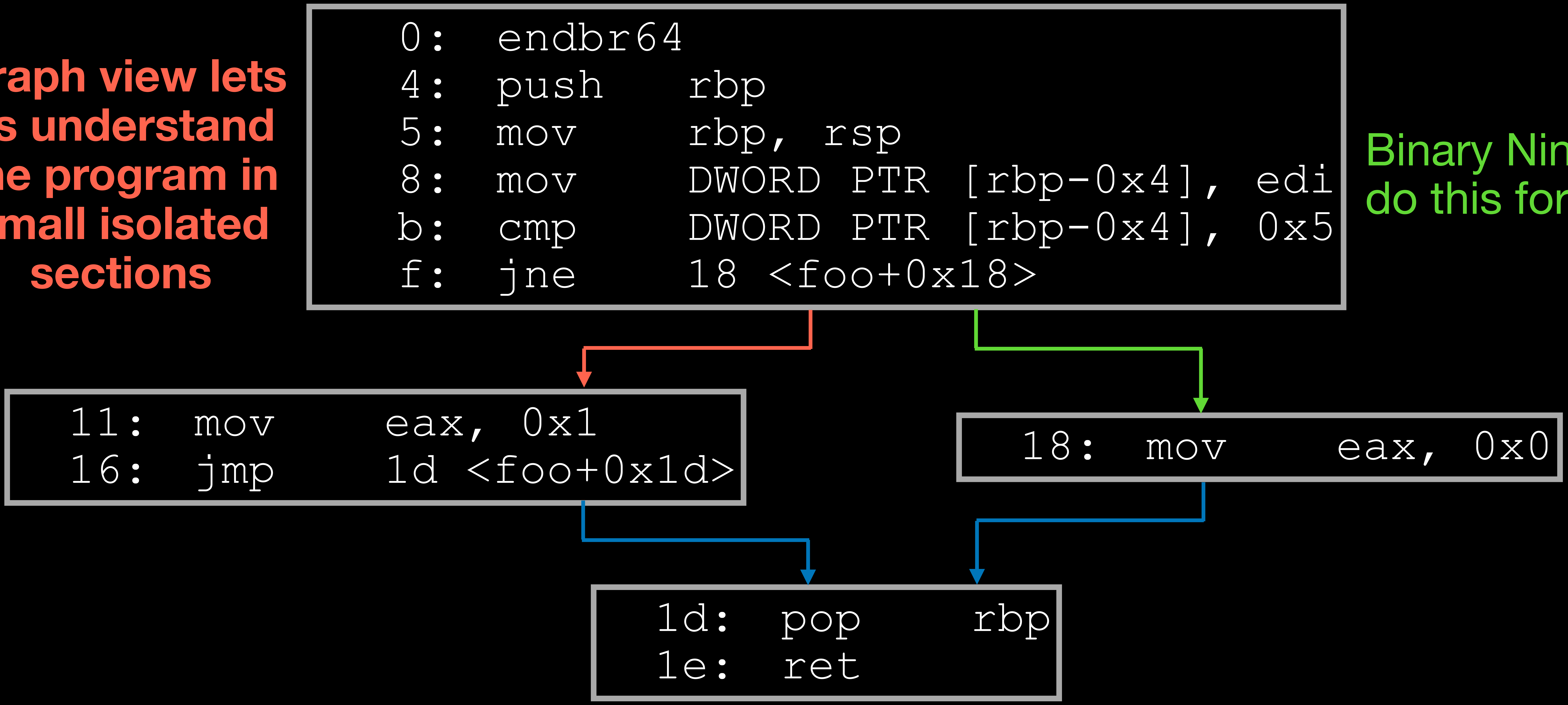
```
18:   mov     eax, 0x0
```

```
1d:   pop     rbp
1e:   ret
```

CPU…

control flow graphs

Graph view lets us understand the program in small isolated sections



```
   0:   endbr64
   4:   push    rbp
   5:   mov     rbp, rsp
   8:   mov     DWORD PTR [rbp-0x4], edi
   b:   cmp     DWORD PTR [rbp-0x4], 0x5
   f:   jne     18 <foo+0x18>
```

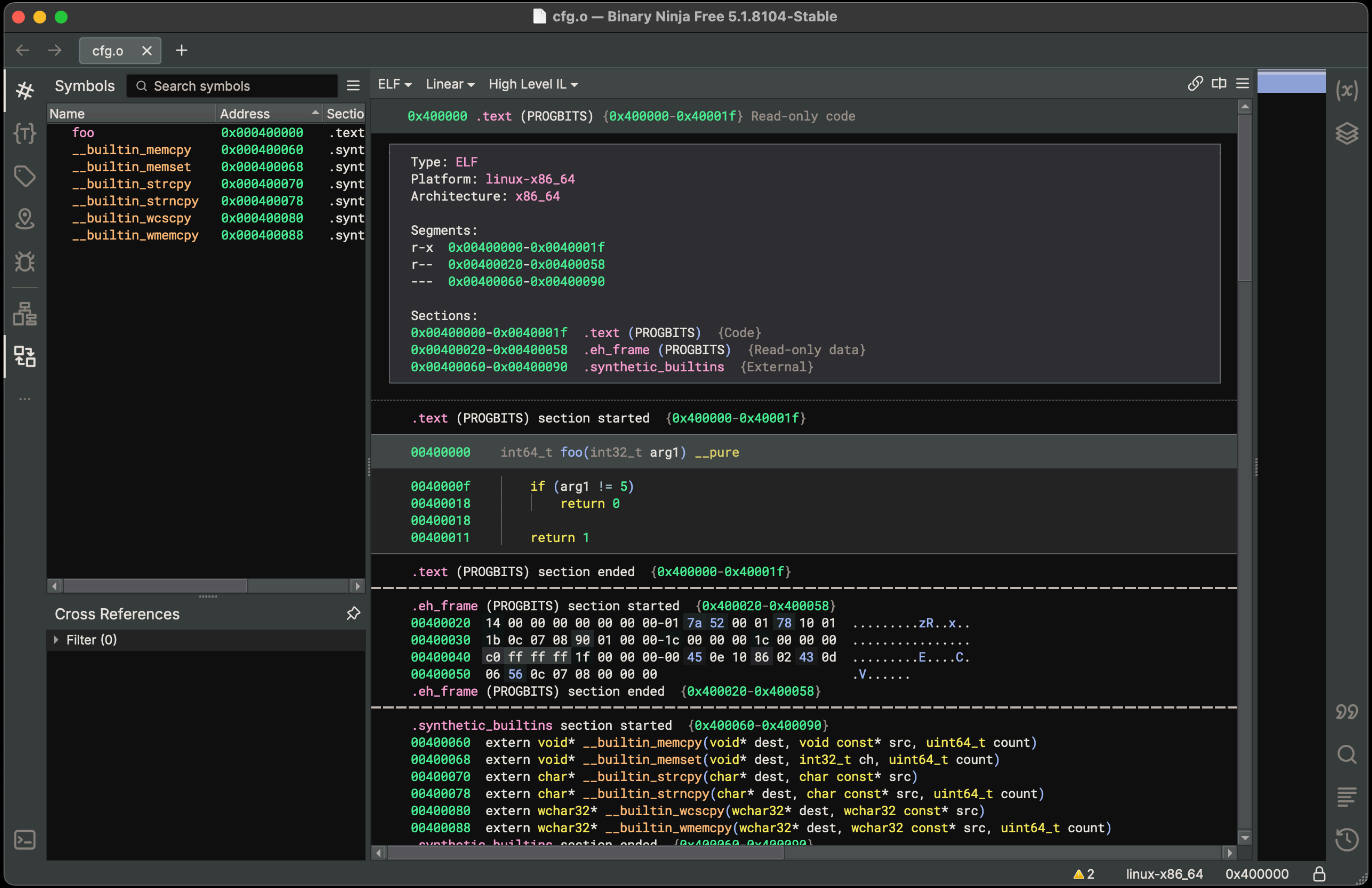
```
  11:   mov     eax, 0x1
  16:   jmp     1d <foo+0x1d>
```

```
  18:   mov     eax, 0x0
```

```
  1d:   pop     rbp
  1e:   ret
```

Binary Ninja can do this for us :)

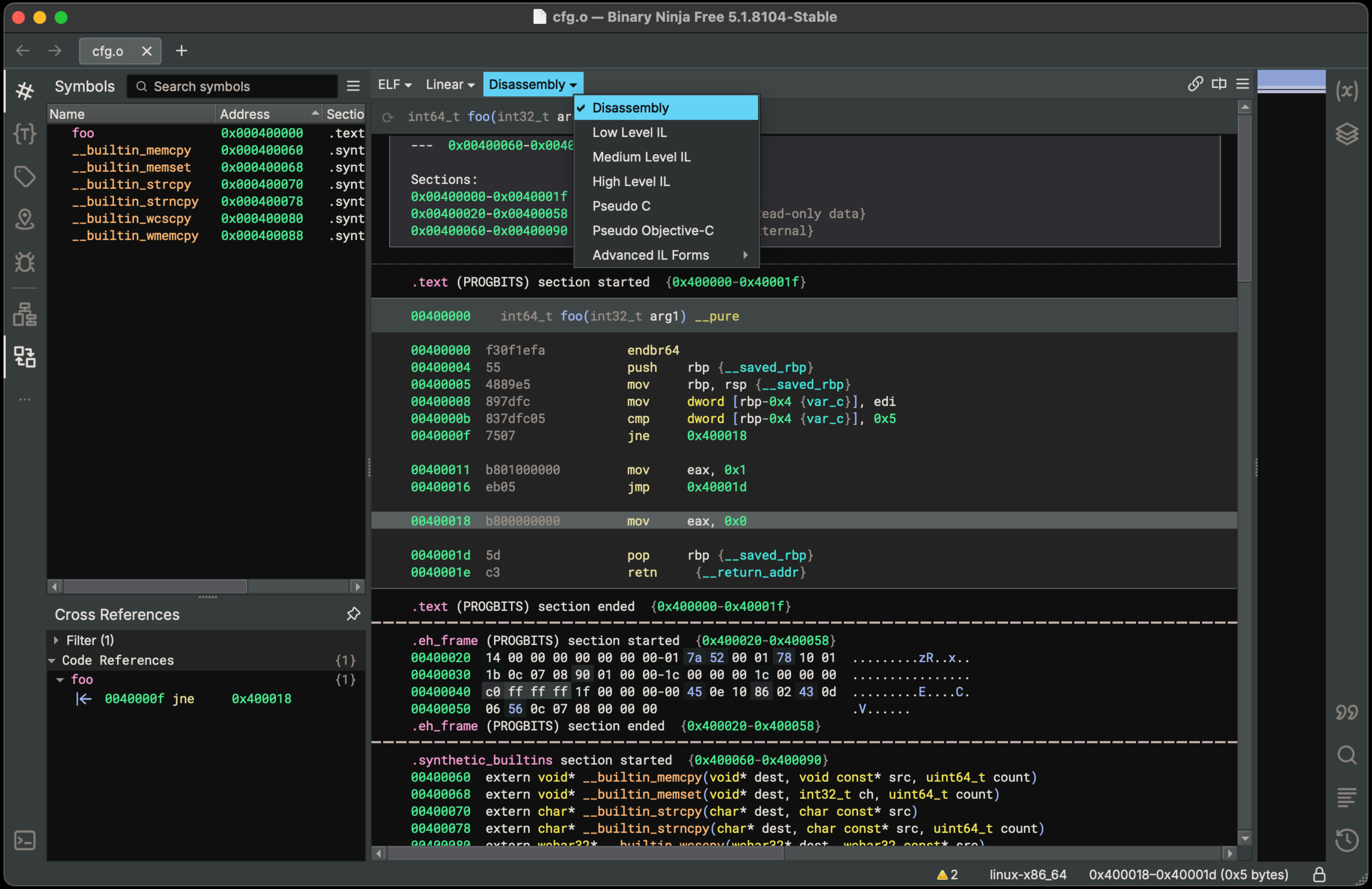
Binary Ninja



Name	Address	Sectio
foo	0x000400000	.text
__builtin_memcpy	0x000400060	.synt
__builtin_memset	0x000400068	.synt
__builtin_strcpy	0x000400070	.synt
__builtin_strncpy	0x000400078	.synt
__builtin_wcscpy	0x000400080	.synt
__builtin_wmemcpy	0x000400088	.synt

```
0x400000 .text (PROGBITS) {0x400000-0x40001f} Read-only code

  Type: ELF
  Platform: linux-x86_64
  Architecture: x86_64

  Segments:
  r-x  0x00400000-0x0040001f
  r--  0x00400020-0x00400058
  ---  0x00400060-0x00400090

  Sections:
  0x00400000-0x0040001f  .text (PROGBITS)  {Code}
  0x00400020-0x00400058  .eh_frame (PROGBITS)  {Read-only data}
  0x00400060-0x00400090  .synthetic_builtins  {External}

 .text (PROGBITS) section started  {0x400000-0x40001f}

 00400000    int64_t foo(int32_t arg1) __pure

 0040000f        if (arg1 != 5)
 00400018            return 0
 00400018
 00400011        return 1

 .text (PROGBITS) section ended  {0x400000-0x40001f}

 .eh_frame (PROGBITS) section started  {0x400020-0x400058}
 00400020  14 00 00 00 00 00 00 00-01 7a 52 00 01 78 10 01  .........zR..x..
 00400030  1b 0c 07 08 90 01 00 00-1c 00 00 00 1c 00 00 00  ................
 00400040  c0 ff ff ff 1f 00 00 00-00 45 0e 10 86 02 43 0d  .........E....C.
 00400050  06 56 0c 07 08 00 00 00                          .V......
 .eh_frame (PROGBITS) section ended  {0x400020-0x400058}

 .synthetic_builtins section started  {0x400060-0x400090}
 00400060  extern void* __builtin_memcpy(void* dest, void const* src, uint64_t count)
 00400068  extern void* __builtin_memset(void* dest, int32_t ch, uint64_t count)
 00400070  extern char* __builtin_strcpy(char* dest, char const* src)
 00400078  extern char* __builtin_strncpy(char* dest, char const* src, uint64_t count)
 00400080  extern wchar32* __builtin_wcscpy(wchar32* dest, wchar32 const* src)
 00400088  extern wchar32* __builtin_wmemcpy(wchar32* dest, wchar32 const* src, uint64_t count)
 .synthetic_builtins section ended  {0x400060-0x400090}
```

Binary Ninja

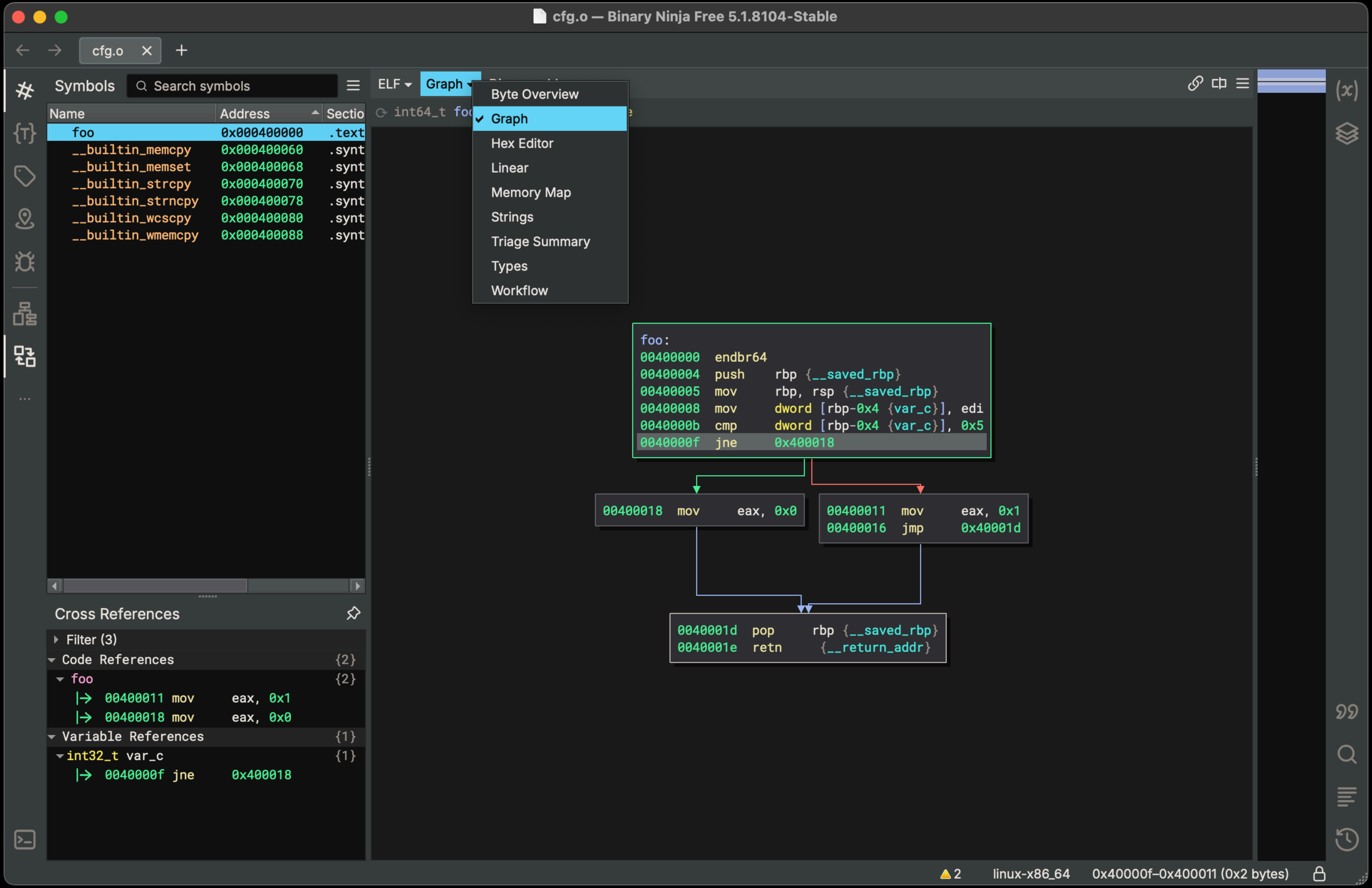


Name	Address	Sectio
foo	0x000400000	.text
__builtin_memcpy	0x000400060	.synt
__builtin_memset	0x000400068	.synt
__builtin_strcpy	0x000400070	.synt
__builtin_strncpy	0x000400078	.synt
__builtin_wcscpy	0x000400080	.synt
__builtin_wmemcpy	0x000400088	.synt

```
Code References                    {1}
  foo                              {1}
    |← 0040000f jne    0x400018
```

```
int64_t foo(int32_t ar
 ---  0x00400060-0x0040

 Sections:
 0x00400000-0x0040001f
 0x00400020-0x00400058                    ead-only data}
 0x00400060-0x00400090                    ternal}

.text (PROGBITS) section started  {0x400000-0x40001f}

00400000    int64_t foo(int32_t arg1) __pure

00400000  f30f1efa          endbr64
00400004  55                push    rbp {__saved_rbp}
00400005  4889e5            mov     rbp, rsp {__saved_rbp}
00400008  897dfc            mov     dword [rbp-0x4 {var_c}], edi
0040000b  837dfc05          cmp     dword [rbp-0x4 {var_c}], 0x5
0040000f  7507              jne     0x400018

00400011  b801000000        mov     eax, 0x1
00400016  eb05              jmp     0x40001d

00400018  b800000000        mov     eax, 0x0

0040001d  5d                pop     rbp {__saved_rbp}
0040001e  c3                retn     {__return_addr}

.text (PROGBITS) section ended  {0x400000-0x40001f}
------------------------------------------------------------------------
.eh_frame (PROGBITS) section started  {0x400020-0x400058}
00400020  14 00 00 00 00 00 00 00-01 7a 52 00 01 78 10 01  .........zR..x..
00400030  1b 0c 07 08 90 01 00 00-1c 00 00 00 1c 00 00 00  ................
00400040  c0 ff ff ff 1f 00 00 00-00 45 0e 10 86 02 43 0d  .........E....C.
00400050  06 56 0c 07 08 00 00 00                          .V......
.eh_frame (PROGBITS) section ended  {0x400020-0x400058}
------------------------------------------------------------------------
.synthetic_builtins section started  {0x400060-0x400090}
00400060  extern void* __builtin_memcpy(void* dest, void const* src, uint64_t count)
00400068  extern void* __builtin_memset(void* dest, int32_t ch, uint64_t count)
00400070  extern char* __builtin_strcpy(char* dest, char const* src)
00400078  extern char* __builtin_strncpy(char* dest, char const* src, uint64_t count)
```

Binary Ninja



Name	Address	Sectio
foo	0x000400000	.text
__builtin_memcpy	0x000400060	.synt
__builtin_memset	0x000400068	.synt
__builtin_strcpy	0x000400070	.synt
__builtin_strncpy	0x000400078	.synt
__builtin_wcscpy	0x000400080	.synt
__builtin_wmemcpy	0x000400088	.synt

```
foo:
00400000  endbr64
00400004  push    rbp {__saved_rbp}
00400005  mov     rbp, rsp {__saved_rbp}
00400008  mov     dword [rbp-0x4 {var_c}], edi
0040000b  cmp     dword [rbp-0x4 {var_c}], 0x5
0040000f  jne     0x400018
```

```
00400018  mov     eax, 0x0
```

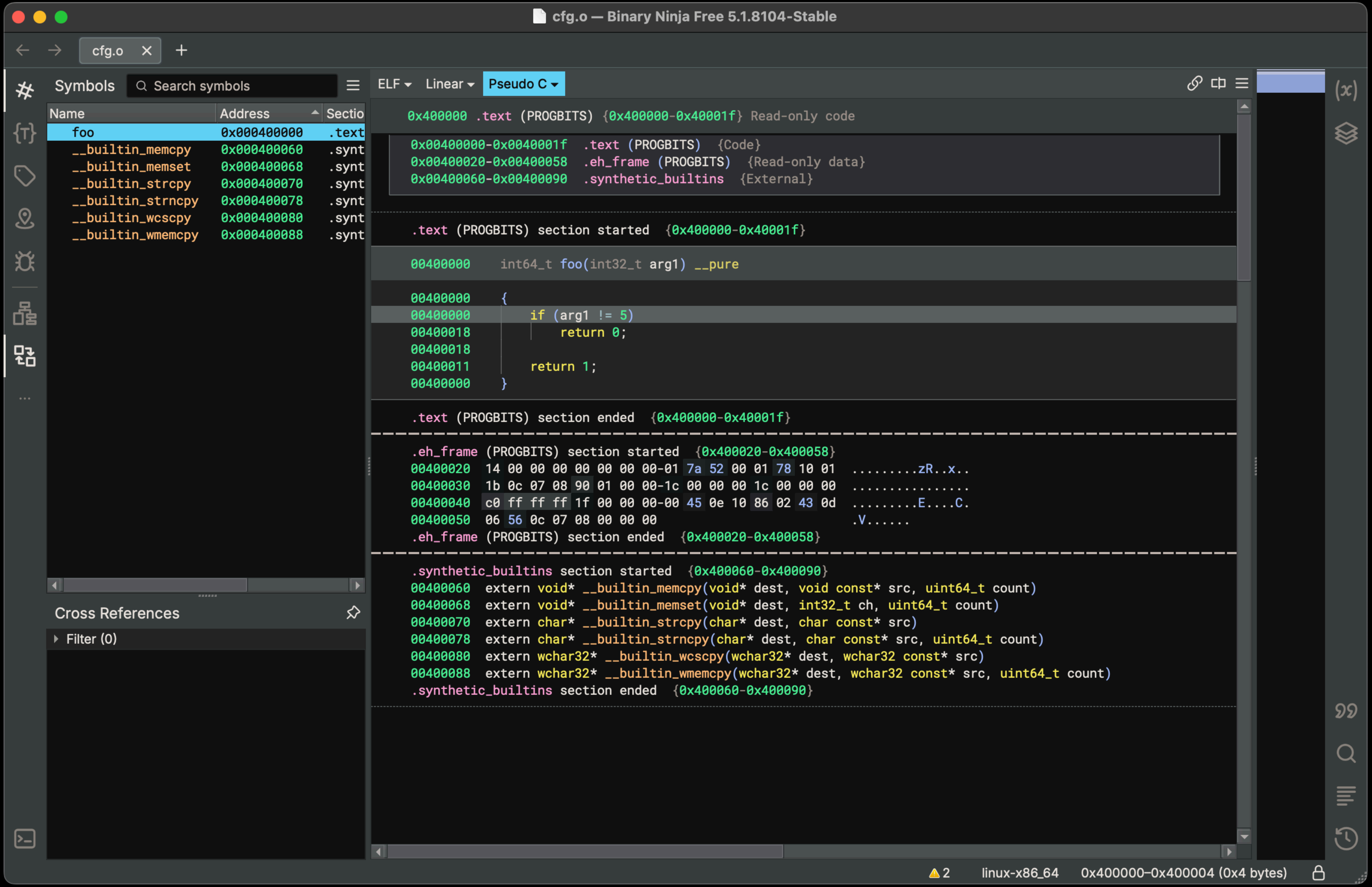
```
00400011  mov     eax, 0x1
00400016  jmp     0x40001d
```

```
0040001d  pop     rbp {__saved_rbp}
0040001e  retn     {__return_addr}
```

```
▸ Filter (3)
▾ Code References                    {2}
  ▾ foo                              {2}
     |→ 00400011 mov     eax, 0x1
     |→ 00400018 mov     eax, 0x0
▾ Variable References                {1}
  ▾int32_t var_c                     {1}
     |→ 0040000f jne     0x400018
```

What if we could turn assembly → C style code?

Binary Ninja



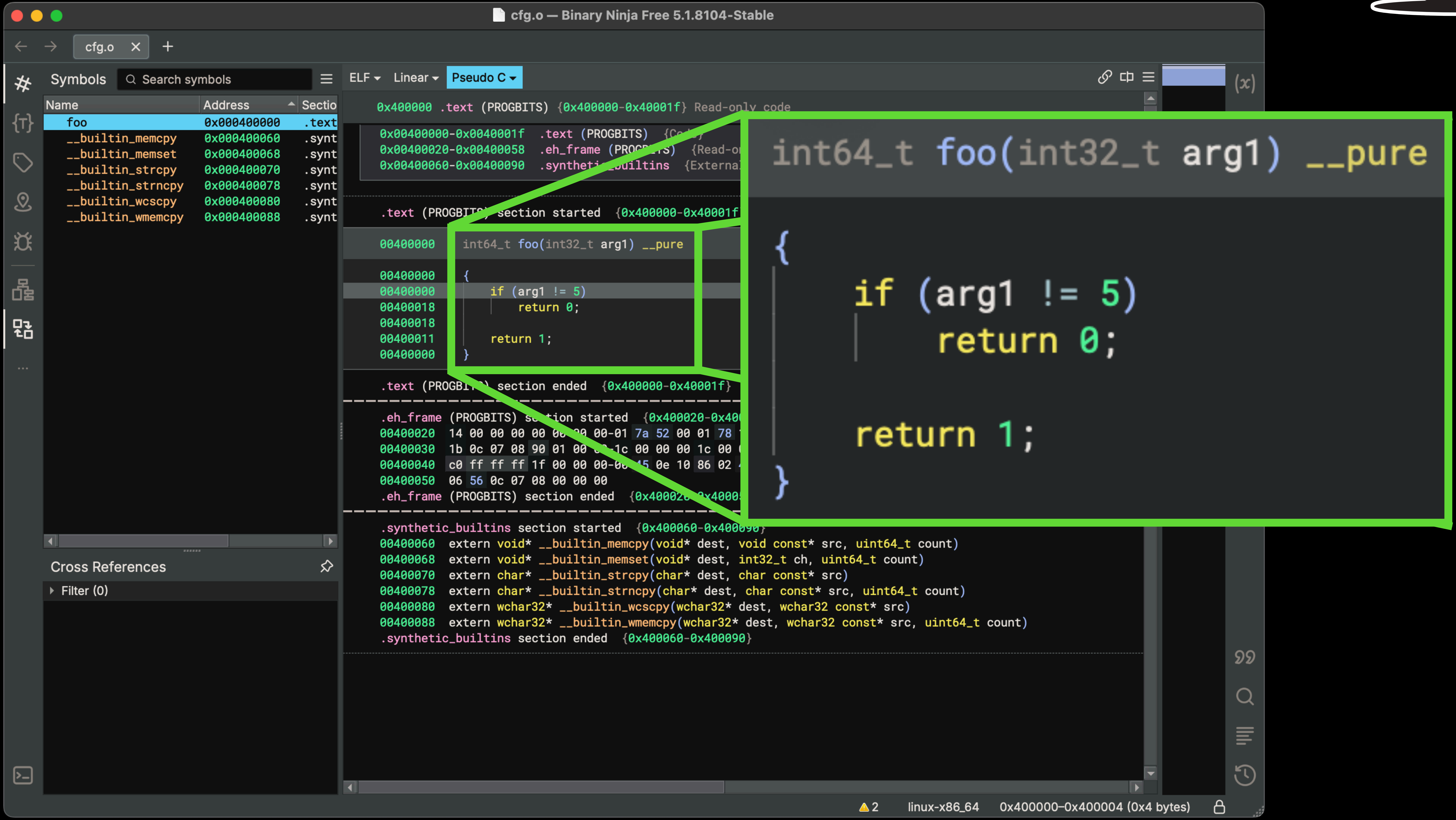
cfg.o — Binary Ninja Free 5.1.8104-Stable

Name	Address	Sectio
foo	0x000400000	.text
__builtin_memcpy	0x000400060	.synt
__builtin_memset	0x000400068	.synt
__builtin_strcpy	0x000400070	.synt
__builtin_strncpy	0x000400078	.synt
__builtin_wcscpy	0x000400080	.synt
__builtin_wmemcpy	0x000400088	.synt

```
0x400000 .text (PROGBITS) {0x400000-0x40001f} Read-only code

 0x00400000-0x0040001f  .text (PROGBITS)  {Code}
 0x00400020-0x00400058  .eh_frame (PROGBITS)  {Read-only data}
 0x00400060-0x00400090  .synthetic_builtins  {External}

 .text (PROGBITS) section started  {0x400000-0x40001f}

 00400000    int64_t foo(int32_t arg1) __pure

 00400000    {
 00400000        if (arg1 != 5)
 00400018            return 0;
 00400018
 00400011        return 1;
 00400000    }

 .text (PROGBITS) section ended  {0x400000-0x40001f}

 .eh_frame (PROGBITS) section started  {0x400020-0x400058}
 00400020  14 00 00 00 00 00 00 00-01 7a 52 00 01 78 10 01  .........zR..x..
 00400030  1b 0c 07 08 90 01 00 00-1c 00 00 00 1c 00 00 00  ................
 00400040  c0 ff ff ff 1f 00 00 00-00 45 0e 10 86 02 43 0d  .........E....C.
 00400050  06 56 0c 07 08 00 00 00                          .V......
 .eh_frame (PROGBITS) section ended  {0x400020-0x400058}

 .synthetic_builtins section started  {0x400060-0x400090}
 00400060  extern void* __builtin_memcpy(void* dest, void const* src, uint64_t count)
 00400068  extern void* __builtin_memset(void* dest, int32_t ch, uint64_t count)
 00400070  extern char* __builtin_strcpy(char* dest, char const* src)
 00400078  extern char* __builtin_strncpy(char* dest, char const* src, uint64_t count)
 00400080  extern wchar32* __builtin_wcscpy(wchar32* dest, wchar32 const* src)
 00400088  extern wchar32* __builtin_wmemcpy(wchar32* dest, wchar32 const* src, uint64_t count)
 .synthetic_builtins section ended  {0x400060-0x400090}
```

⚠2 linux-x86_64 0x400000–0x400004 (0x4 bytes)

Binary Ninja



Name	Address	Sectio
foo	0x000400000	.text
__builtin_memcpy	0x000400060	.synt
__builtin_memset	0x000400068	.synt
__builtin_strcpy	0x000400070	.synt
__builtin_strncpy	0x000400078	.synt
__builtin_wcscpy	0x000400080	.synt
__builtin_wmemcpy	0x000400088	.synt

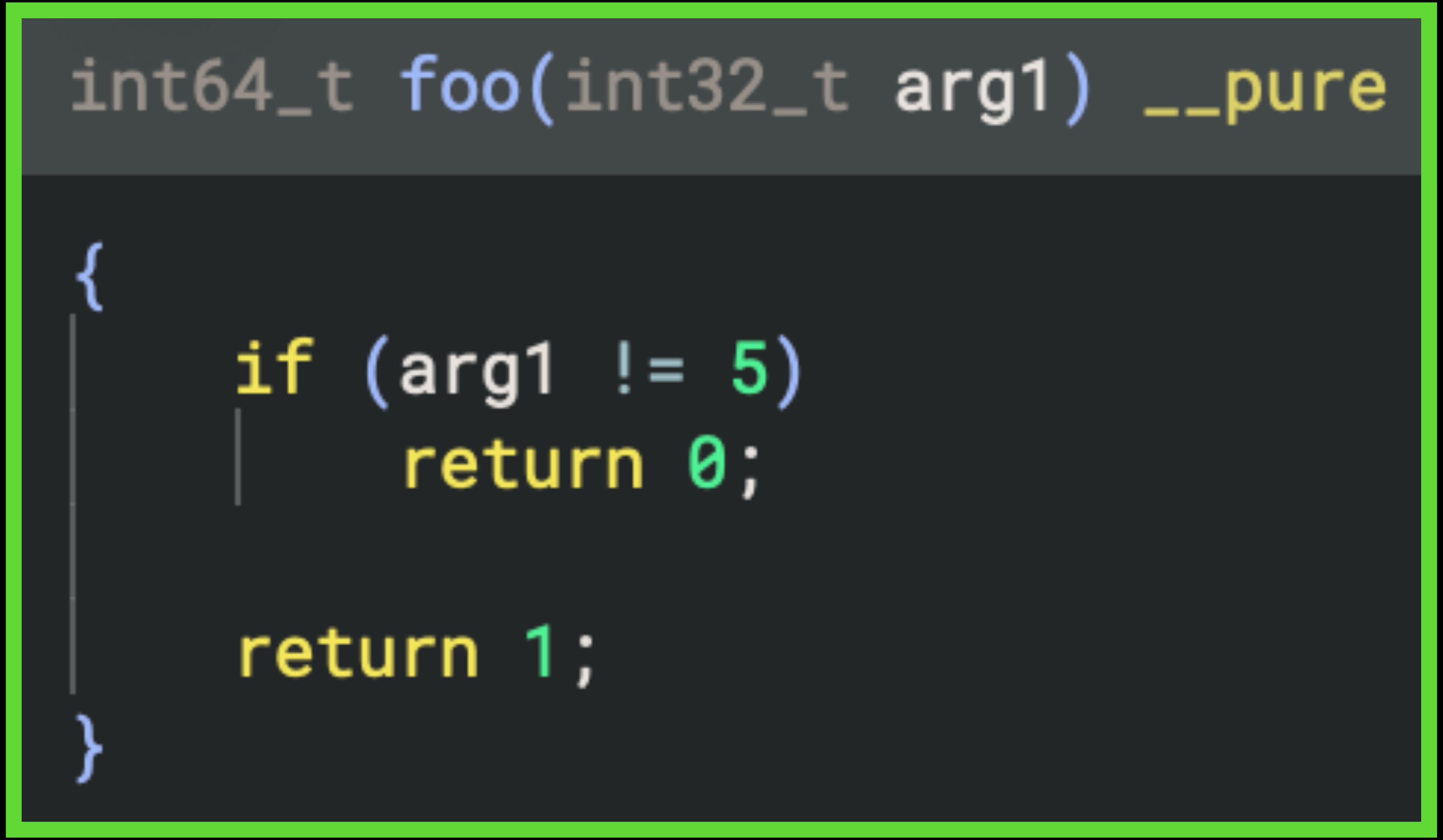
```
0x400000 .text (PROGBITS) {0x400000-0x40001f} Read-only code

  0x00400000-0x0040001f  .text (PROGBITS)  {Co
  0x00400020-0x00400058  .eh_frame (PROG    S)  {Read-o
  0x00400060-0x00400090  .syntheti  builtins  {Externa

 .text (PROGBIT  section started  {0x400000-0x40001f

00400000  int64_t foo(int32_t arg1) __pure

00400000  {
00400000      if (arg1 != 5)
00400018          return 0;
00400018
00400011      return 1;
00400000  }

 .text (PROGBI   section ended  {0x400000-0x40001f}

 .eh_frame (PROGBITS) se  ion started  {0x400020-0x400
00400020  14 00 00 00 00 0   0 00-01 7a 52 00 01 78
00400030  1b 0c 07 08 90 01 00   -1c 00 00 00 1c 00
00400040  c0 ff ff ff 1f 00 00 00-0   45 0e 10 86 02
00400050  06 56 0c 07 08 00 00 00
 .eh_frame (PROGBITS) section ended  {0x40002   x4000

 .synthetic_builtins section started  {0x400060-0x400090}
00400060  extern void* __builtin_memcpy(void* dest, void const* src, uint64_t count)
00400068  extern void* __builtin_memset(void* dest, int32_t ch, uint64_t count)
00400070  extern char* __builtin_strcpy(char* dest, char const* src)
00400078  extern char* __builtin_strncpy(char* dest, char const* src, uint64_t count)
00400080  extern wchar32* __builtin_wcscpy(wchar32* dest, wchar32 const* src)
00400088  extern wchar32* __builtin_wmemcpy(wchar32* dest, wchar32 const* src, uint64_t count)
 .synthetic_builtins section ended  {0x400060-0x400090}
```

```
int64_t foo(int32_t arg1) __pure

{
    if (arg1 != 5)
        return 0;

    return 1;
}
```

Binary Ninja

Decompilation

- Key things to note:

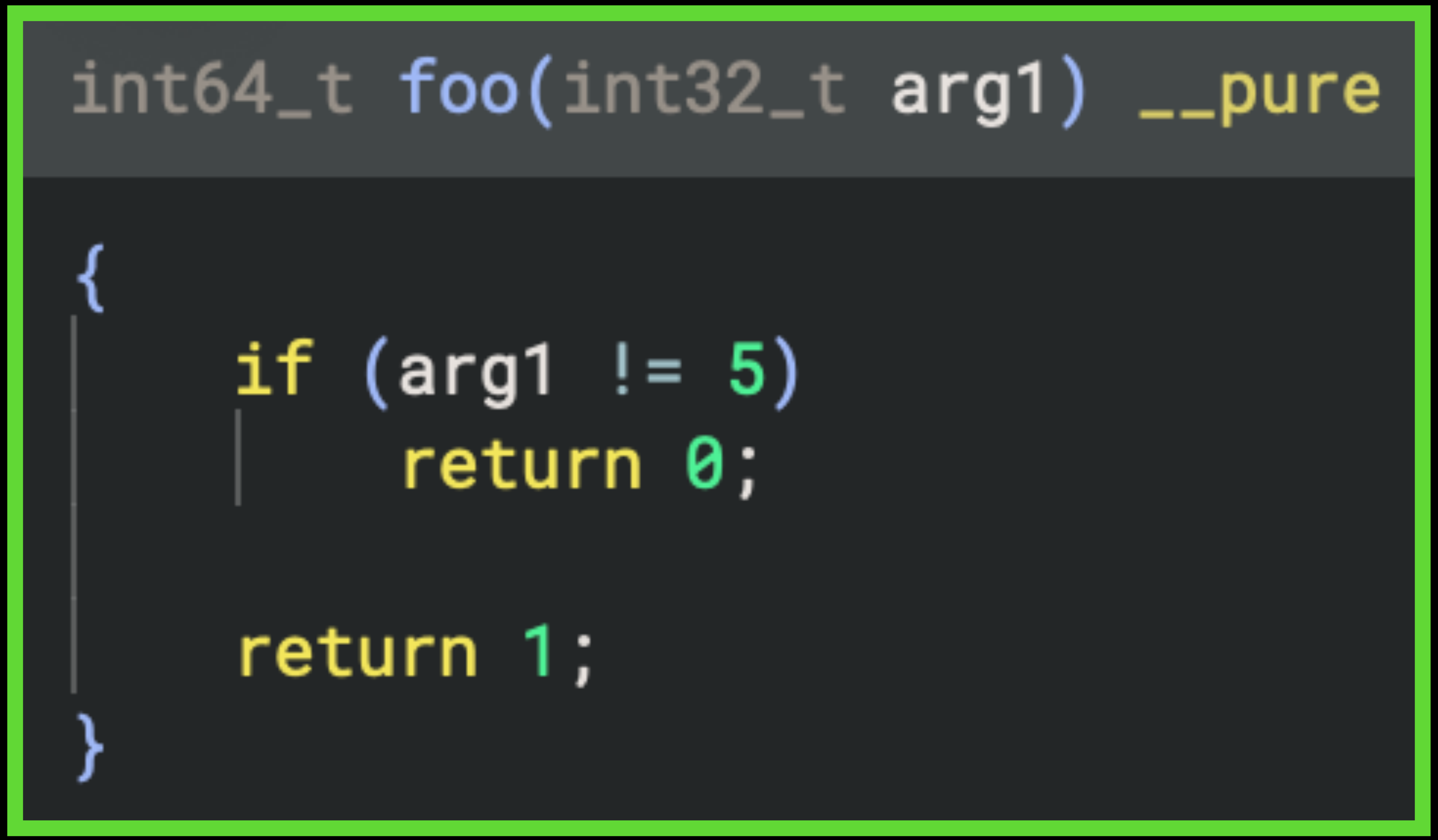


```
int64_t foo(int32_t arg1) __pure

{
    if (arg1 != 5)
        return 0;

    return 1;
}
```

Binary Ninja

Decompilation

- Key things to note:
    - Variable names are lost

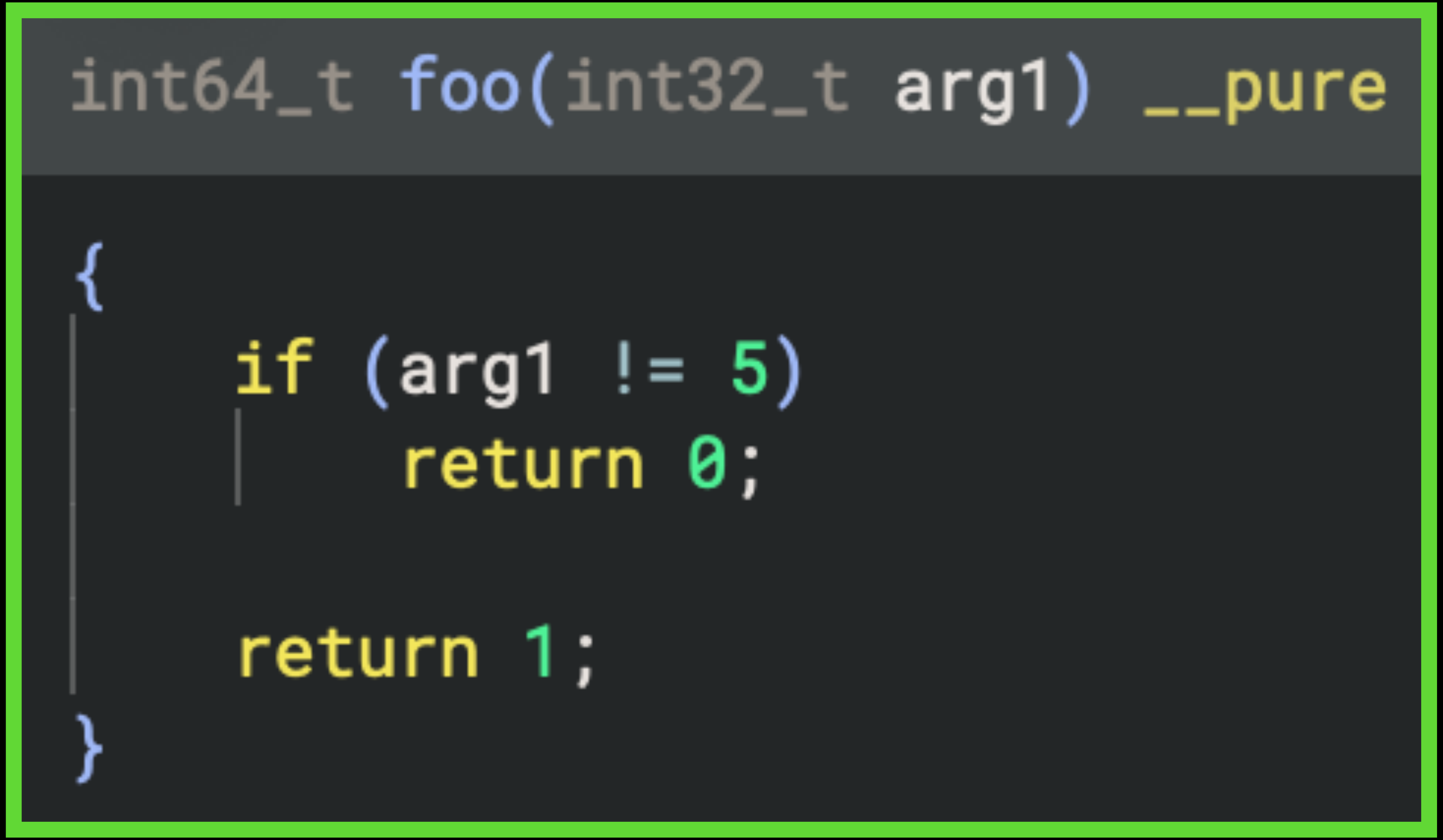


```
int64_t foo(int32_t arg1) __pure

{
    if (arg1 != 5)
        return 0;

    return 1;
}
```

Binary Ninja

Decompilation

- Key things to note:
  - Variable names are lost
  - Exact source structure is lost

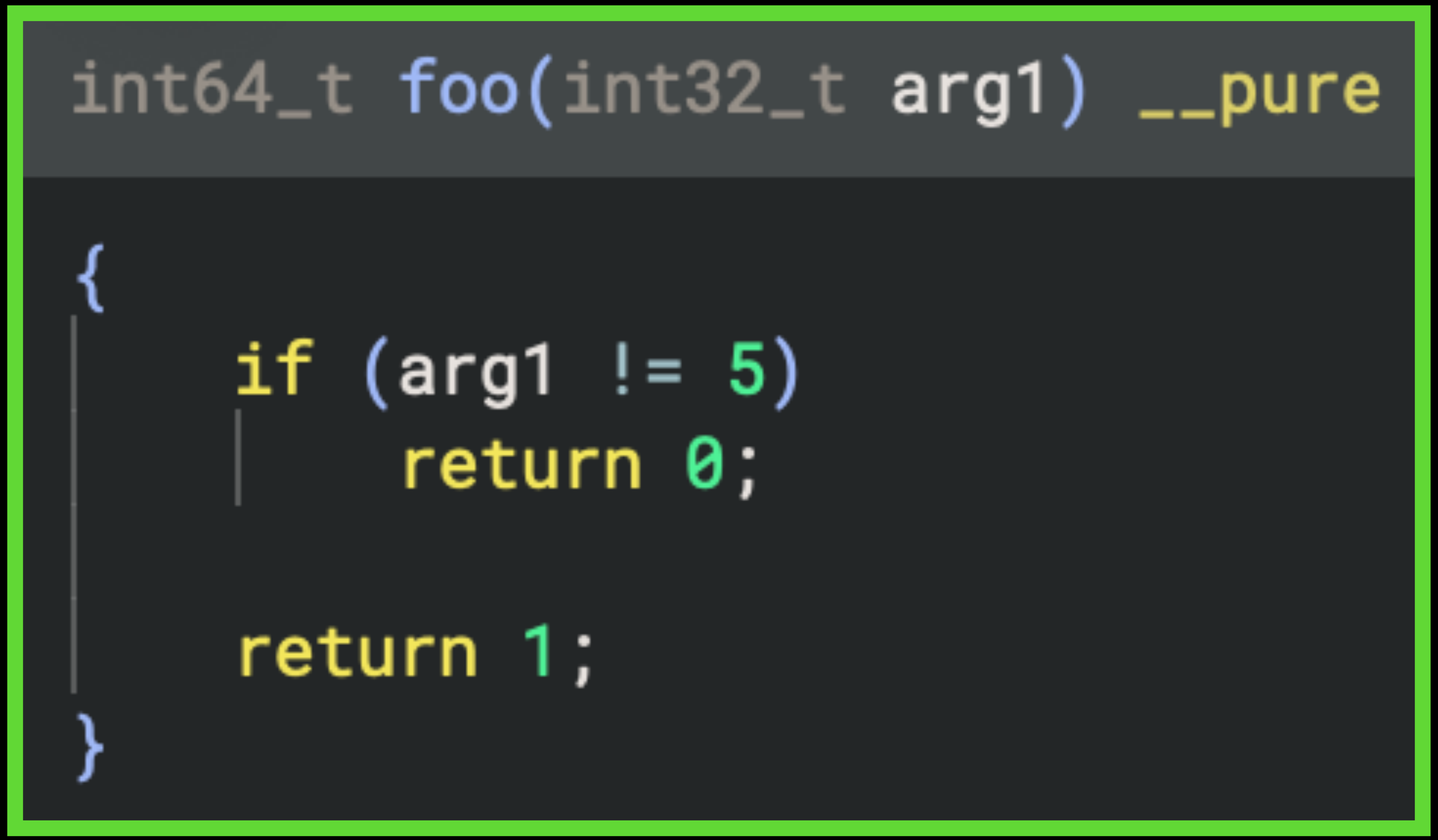


```
int64_t foo(int32_t arg1) __pure

{
    if (arg1 != 5)
        return 0;

    return 1;
}
```

Binary Ninja

Decompilation

- Key things to note:
  - Variable names are lost
  - Exact source structure is lost
  - Comments are lost

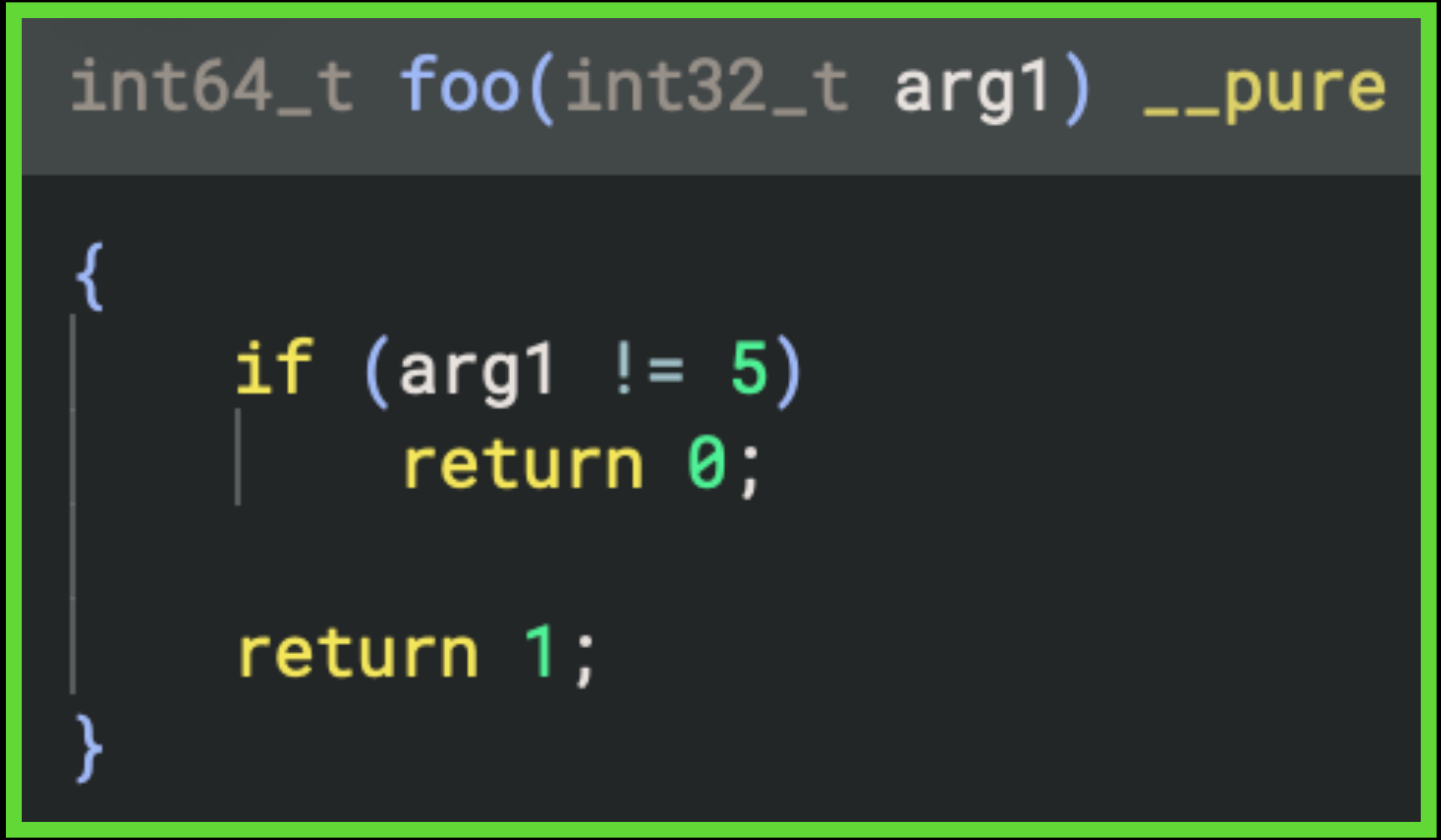


```
int64_t foo(int32_t arg1) __pure

{
    if (arg1 != 5)
        return 0;

    return 1;
}
```

Binary Ninja

Decompilation

- Key things to note:
  - Variable names are lost
  - Exact source structure is lost
  - Comments are lost
  - Function names *may* be lost

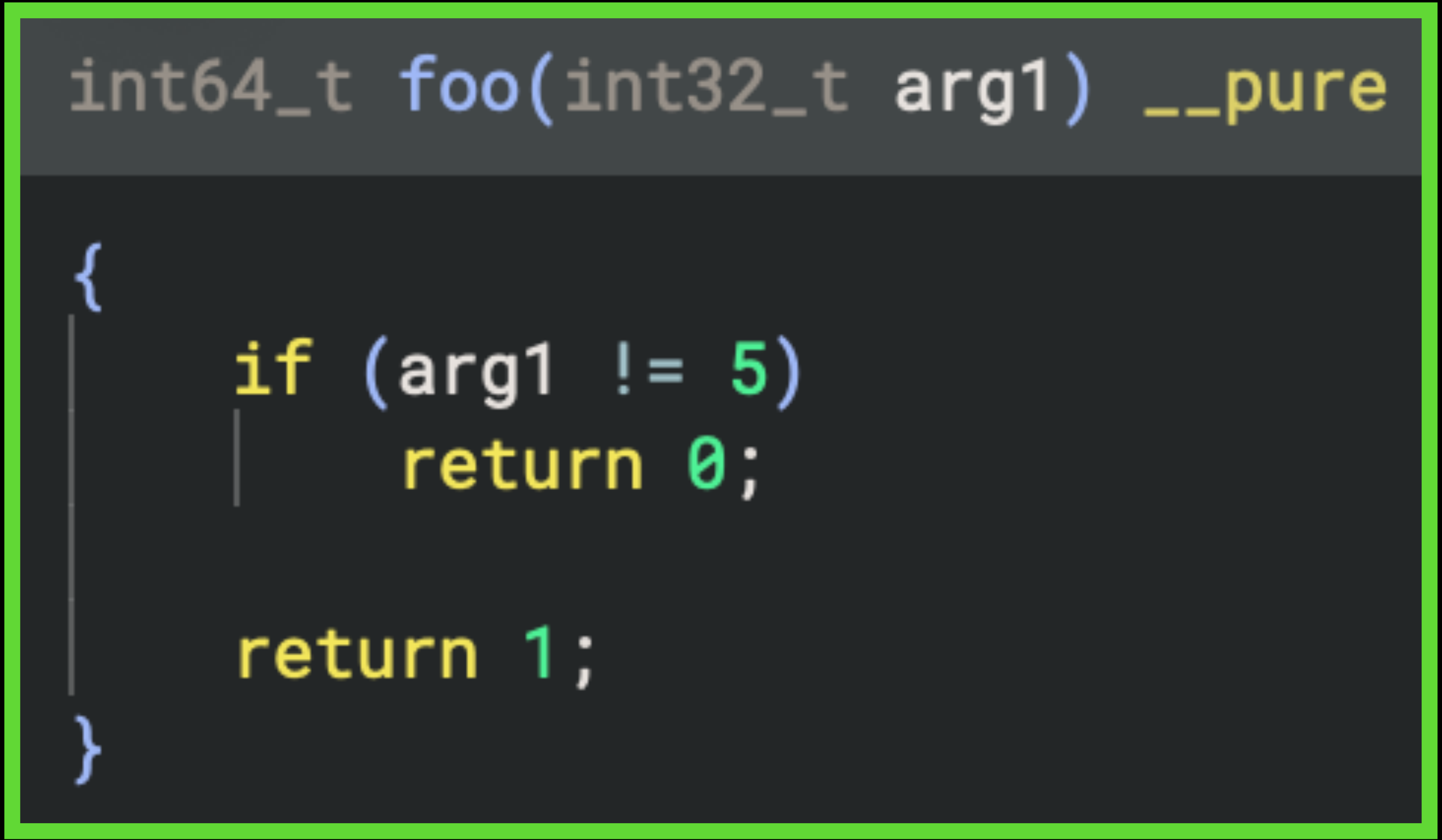


```
int64_t foo(int32_t arg1) __pure

{
    if (arg1 != 5)
        return 0;

    return 1;
}
```

Binary Ninja

Decompilation

- Key things to note:
  - Variable names are lost
  - Exact source structure is lost
  - Comments are lost
  - Function names *may* be lost
  - ***Decompilation is not an exact science***

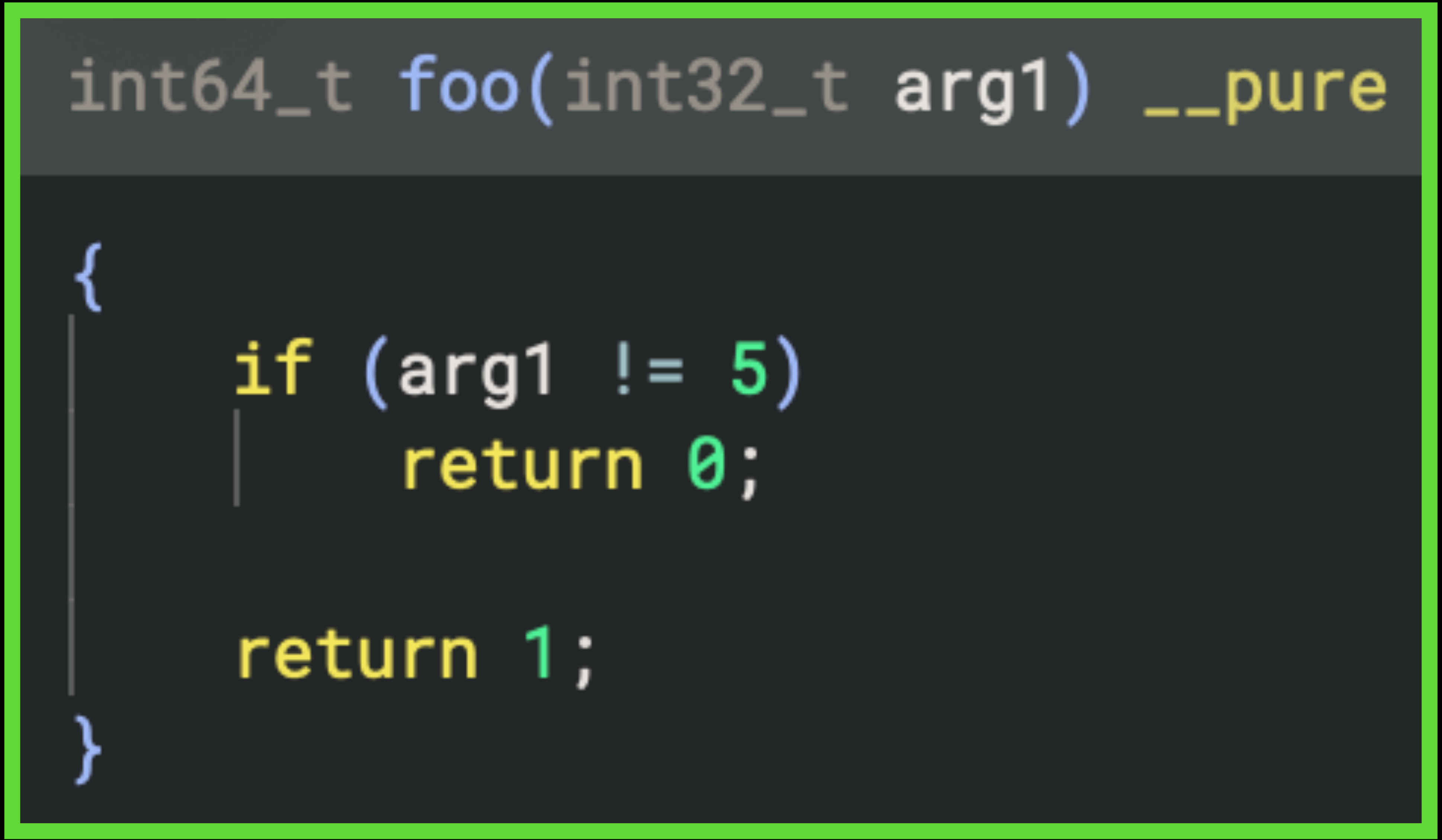


```
int64_t foo(int32_t arg1) __pure

{
    if (arg1 != 5)
        return 0;

    return 1;
}
```

Binary Ninja

Decompilation

- Key things to note:
    - Variable names are lost
    - Exact source structure is lost
    - Comments are lost
    - Function names *may* be lost
    - ***Decompilation is not an exact science***
    - But it is still immensely helpful to understand how a program works :)



```
int64_t foo(int32_t arg1) __pure

{
    if (arg1 != 5)
        return 0;

    return 1;
}
```

What if function names are missing?

Reverse Engineering

Making sense of things

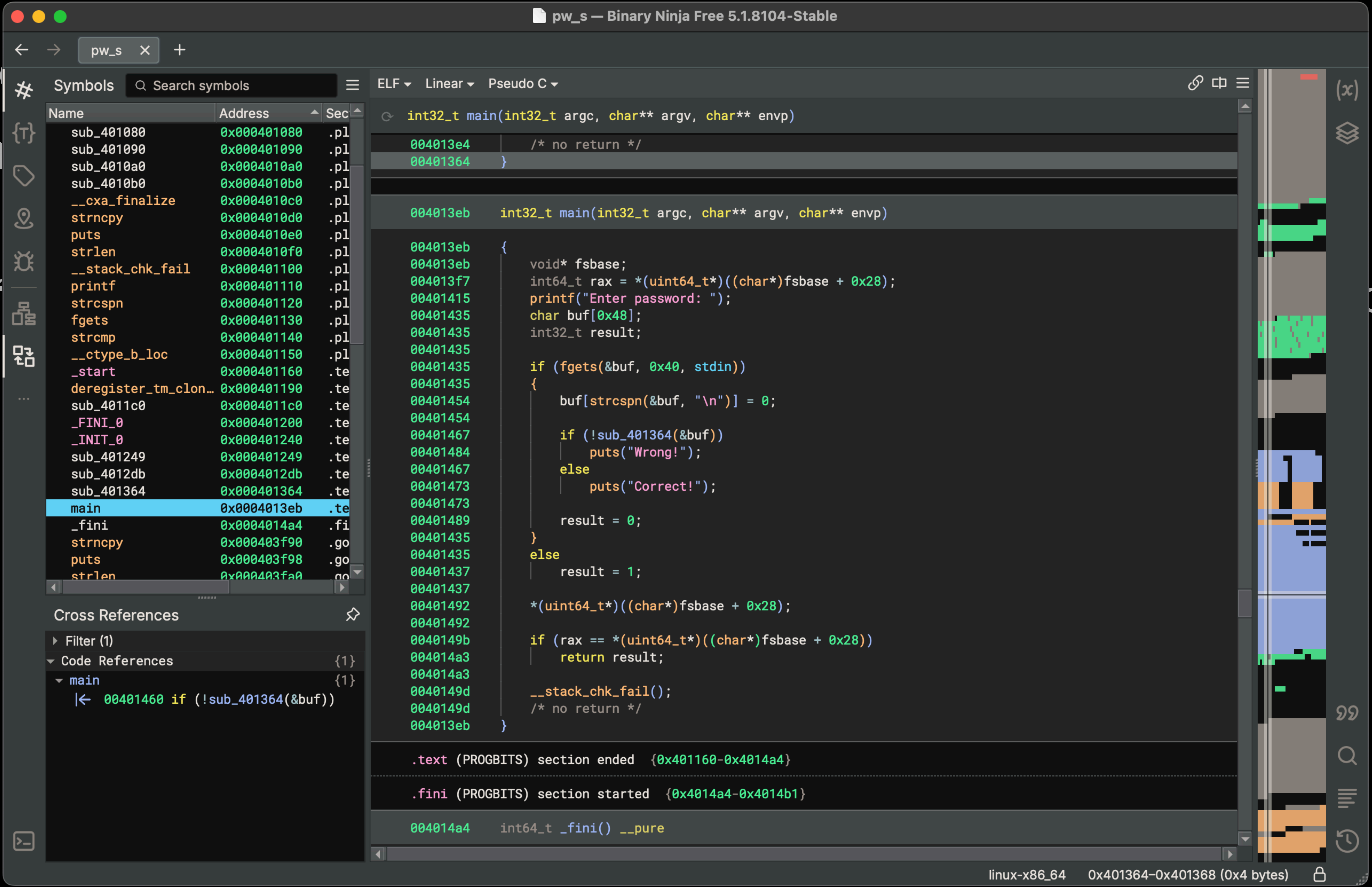
- I’ve “stripped” a binary of it’s symbols, removing any function name info

Rev

Makin

- I've "

o



Name	Address	Sec
sub_401080	0x000401080	.pl
sub_401090	0x000401090	.pl
sub_4010a0	0x0004010a0	.pl
sub_4010b0	0x0004010b0	.pl
__cxa_finalize	0x0004010c0	.pl
strncpy	0x0004010d0	.pl
puts	0x0004010e0	.pl
strlen	0x0004010f0	.pl
__stack_chk_fail	0x000401100	.pl
printf	0x000401110	.pl
strcspn	0x000401120	.pl
fgets	0x000401130	.pl
strcmp	0x000401140	.pl
__ctype_b_loc	0x000401150	.pl
_start	0x000401160	.te
deregister_tm_clon…	0x000401190	.te
sub_4011c0	0x0004011c0	.te
_FINI_0	0x000401200	.te
_INIT_0	0x000401240	.te
sub_401249	0x000401249	.te
sub_4012db	0x0004012db	.te
sub_401364	0x000401364	.te
main	0x0004013eb	.te
_fini	0x0004014a4	.fi
strncpy	0x000403f90	.go
puts	0x000403f98	.go
strlen	0x000403fa0	.go

```
int32_t main(int32_t argc, char** argv, char** envp)

004013e4        /* no return */
00401364    }

004013eb    int32_t main(int32_t argc, char** argv, char** envp)

004013eb    {
004013eb        void* fsbase;
004013f7        int64_t rax = *(uint64_t*)((char*)fsbase + 0x28);
00401415        printf("Enter password: ");
00401435        char buf[0x48];
00401435        int32_t result;
00401435
00401435        if (fgets(&buf, 0x40, stdin))
00401435        {
00401454            buf[strcspn(&buf, "\n")] = 0;
00401454
00401467            if (!sub_401364(&buf))
00401484                puts("Wrong!");
00401467            else
00401473                puts("Correct!");
00401473
00401489            result = 0;
00401435        }
00401435        else
00401437            result = 1;
00401437
00401492        *(uint64_t*)((char*)fsbase + 0x28);
00401492
0040149b        if (rax == *(uint64_t*)((char*)fsbase + 0x28))
004014a3            return result;
004014a3
0040149d        __stack_chk_fail();
0040149d        /* no return */
004013eb    }

.text (PROGBITS) section ended  {0x401160-0x4014a4}

.fini (PROGBITS) section started  {0x4014a4-0x4014b1}

004014a4    int64_t _fini() __pure
```

```
Filter (1)
Code References                         {1}
  main                                  {1}
    |← 00401460 if (!sub_401364(&buf))
```

linux-x86_64   0x401364–0x401368 (0x4 bytes)

```
           0x0004010b0   .pl
ze         0x0004010c0   .pl
           0x0004010d0   .pl
           0x0004010e0   .pl
           0x0004010f0   .pl
fail       0x000401100   .pl
           0x000401110   .pl
           0x000401120   .pl
           0x000401130   .pl
           0x000401140   .pl
c          0x000401150   .pl
           0x000401160   .te
m_clon…    0x000401190   .te
           0x0004011c0   .te
           0x000401200   .te
           0x000401240   .te
           0x000401249   .te
           0x0004012db   .te
           0x000401364   .te
           0x0004013eb   .te
           0x0004014a4   .fi
           0x000403f90   .go
           0x000403f98   .go
           0x000403fa0   .go
```

```
es
es                                  {1}
                                    {1}
) if (!sub_401364(&buf))
```

```
004013eb    int32_t main(int32_t argc, char** argv, char** envp)

004013eb    {
004013eb        void* fsbase;
004013f7        int64_t rax = *(uint64_t*)((char*)fsbase + 0x28);
00401415        printf("Enter password: ");
00401435        char buf[0x48];
00401435        int32_t result;
00401435        
00401435        if (fgets(&buf, 0x40, stdin))
00401435        {
00401454            buf[strcspn(&buf, "\n")] = 0;
00401454            
00401467            if (!sub_401364(&buf))
00401484                puts("Wrong!");
00401467            else
00401473                puts("Correct!");
00401473            
00401489            result = 0;
00401435        }
00401435        else
00401437            result = 1;
00401437        
00401492        *(uint64_t*)((char*)fsbase + 0x28);
00401492        
0040149b        if (rax == *(uint64_t*)((char*)fsbase + 0x28))
004014a3            return result;
004014a3        
0040149d        __stack_chk_fail();
0040149d        /* no return */
004013eb    }

    .text (PROGBITS) section ended  {0x401160-0x4014a4}
```

```
      0x0004010b0  .pl
ze    0x0004010c0  .pl
      0x0004010d0  .pl
      0x0004010e0  .pl
      0x0004010f0  .pl
fail  0x000401100  .pl
      0x000401110  .pl
      0x000401120  .pl
      0x000401130  .pl
      0x000401140  .pl
c     0x000401150  .pl
      0x000401160  .te
m_clon… 0x000401190  .te
      0x0004011c0  .te
      0x000401200  .te
      0x000401240  .te
      0x000401249  .te
      0x0004012db  .te
      0x000401364  .te
      0x0004013eb  .te
      0x0004014a4  .fi
      0x000403f90  .go
      0x000403f98  .go
```

es

es {1}

{1}

```
) if (!sub_401364(&buf))
```

```
004013eb    int32_t main(int32_t argc, char** argv, char** envp)

004013eb    {
004013eb        void* fsbase;
004013f7        int64_t rax = *(uint64_t*)((char*)fsbase + 0x28);
00401415        printf("Enter password: ");
00401435        char buf[0x48];
00401435        int32_t result;
00401435        
00401435        if (fgets(&buf, 0x40, stdin))
00401435        {
00401454            buf[strcspn(&buf, "\n")] = 0;
00401454            
00401467            if (!sub_401364(&buf))
00401484                puts("Wrong!");
00401467            else
00401473                puts("Correct!");
00401473            
00401489            result = 0;
00401435        }
00401435        else
00401437            result = 1;
00401437        
00401492        *(uint64_t*)((char*)fsbase + 0x28);
00401492        
0040149b        if (rax == *(uint64_t*)((char*)fsbase + 0x28))
004014a3            return result;
004014a3        
0040149d        __stack_chk_fail();
0040149d        /* no return */
004013eb    }

.text (PROGBITS) section ended {0x401160-0x4014a4}
```

Some functions are “imported” from other places

We will still have names for these

```
           0x004010a0    .pl
           0x004010b0    .pl
ze         0x004010c0    .pl
           0x004010d0    .pl
           0x004010e0    .pl
           0x004010f0    .pl
fail       0x00401100    .pl
           0x00401110    .pl
           0x00401120    .pl
           0x00401130    .pl
           0x00401140    .pl
c          0x00401150    .pl
           0x00401160    .te
m_clon…    0x00401190    .te
           0x004011c0    .te
           0x00401200    .te
           0x00401240    .te
           0x00401249    .te
           0x004012db    .te
           0x00401364    .te
           0x004013eb    .te
           0x004014a4    .fi
           0x00403f90    .go
           0x00403f98    .go
           0x00403fa0    .go
```

```
es
es                                {1}
                                  {1}
if (!sub_401364(&buf))
```

```
004013eb    int32_t main(int32_t argc, char** argv, char** envp)

004013eb    {
004013eb        void* fsbase;
004013f7        int64_t rax = *(uint64_t*)((char*)fsbase + 0x28);
00401415        printf("Enter password: ");
00401435        char buf[0x48];
00401435        int32_t result;
00401435
00401435        if (fgets(&buf, 0x40, stdin))
00401435        {
00401454            buf[strcspn(&buf, "\n")] = 0;
00401454
00401467            if (!sub_401364(&buf))
00401484                puts("Wrong!");
00401467            else
00401473                puts("Correct!");
00401473
00401489            result = 0;
00401435        }
00401435        else
00401437            result = 1;
00401437
00401492        *(uint64_t*)((char*)fsbase + 0x28);
00401492
0040149b        if (rax == *(uint64_t*)((char*)fsbase + 0x28))
004014a3            return result;
004014a3
0040149d        __stack_chk_fail();
0040149d        /* no return */
004013eb    }
```

.text (PROGBITS) section ended {0x401160-0x4014a4}

But functions that are part of this program won’t have a name associated with them

```
        0x0004010b0   .pl
ze      0x0004010c0   .pl
        0x0004010d0   .pl
        0x0004010e0   .pl
        0x0004010f0   .pl
fail    0x000401100   .pl
        0x000401110   .pl
        0x000401120   .pl
        0x000401130   .pl
        0x000401140   .pl
c       0x000401150   .pl
        0x000401160   .te
m_clon… 0x000401190   .te
        0x0004011c0   .te
        0x000401200   .te
        0x000401240   .te
        0x000401249   .te
        0x0004012db   .te
        0x000401364   .te
        0x0004013eb   .te
        0x0004014a4   .fi
        0x000403f90   .go
        0x000403f98   .go
```

```
es
es                      {1}
                        {1}
) if (!sub_401364(&buf))
```

```
004013eb    int32_t main(int32_t argc, char** argv, char** envp)

004013eb    {
004013eb        void* fsbase;
004013f7        int64_t rax = *(uint64_t*)((char*)fsbase + 0x28);
00401415        printf("Enter password: ");
00401435        char buf[0x48];
00401435        int32_t result;
00401435        
00401435        if (fgets(&buf, 0x40, stdin))
00401435        {
00401454            buf[strcspn(&buf, "\n")] = 0;
00401454            
00401467            if (!sub_401364(&buf))
00401484                puts("Wrong!");
00401467            else
00401473                puts("Correct!");
00401473            
00401489            result = 0;
00401435        }
00401435        else
00401437            result = 1;
00401437        
00401492        *(uint64_t*)((char*)fsbase + 0x28);
00401492        
0040149b        if (rax == *(uint64_t*)((char*)fsbase + 0x28))
004014a3            return result;
004014a3        
0040149d        __stack_chk_fail();
0040149d        /* no return */
004013eb    }
```

We need to figure out what this function does, and give it a name ourselves :)

```
00401364    uint64_t sub_401364(char* arg1)

00401364    {
00401364        void* fsbase;
00401374        int64_t rax = *(uint64_t*)((char*)fsbase + 0x28);
00401396        char var_58[0x3f];
00401396        strncpy(&var_58, arg1, 0x40);
0040139b        char var_19 = 0;
004013a6        sub_401249(&var_58);
004013b2        sub_4012db(&var_58);
004013cf        int32_t rax_1;
004013cf        (uint8_t)rax_1 = !strcmp(&var_58, "tfds{zpv_hpu_ju}");
004013d9        *(uint64_t*)((char*)fsbase + 0x28);
004013d9
004013e2        if (rax == *(uint64_t*)((char*)fsbase + 0x28))
004013ea            return (uint64_t)(uint8_t)rax_1;
004013ea
004013e4        __stack_chk_fail();
004013e4        /* no return */
00401364    }
```

```
00401364    uint64_t sub_401364(char* arg1)

00401364    {
00401364        void* fsbase;
00401374        int64_t rax = *(uint64_t*)((char*)fsbase + 0x28);
00401396        char var_58[0x3f];
00401396        strncpy(&var_58, arg1, 0x40);
0040139b        char var_19 = 0;
004013a6        sub_401249(&var_58);
004013b2        sub_4012db(&var_58);
004013cf        int32_t rax_1;
004013cf        (uint8_t)rax_1 = !strcmp(&var_58, "tfds{zpv_hpu_ju}");
004013d9        *(uint64_t*)((char*)fsbase + 0x28);
004013d9
004013e2        if (rax == *(uint64_t*)((char*)fsbase + 0x28))
004013ea            return (uint64_t)(uint8_t)rax_1;
004013ea
004013e4        __stack_chk_fail();
004013e4        /* no return */
```

```
00401364    uint64_t sub_401364(char* arg1)

00401364    {
00401364        void* fsbase;
00401374        int64_t rax = *(uint64_t*)((char*)fsbase + 0x28);
00401396        char var_58[0x3f];
00401396        strncpy(&var_58, arg1, 0x40);
0040139b        char var_19 = 0;
004013a6        sub_401249(&var_58);
004013b2        sub_4012db(&var_58);
004013cf        int32_t rax_1;
004013cf        (uint8_t)rax_1 = !strcmp(&var_58, "tfds{zpv_hpu_ju}");
004013d9        *(uint64_t*)((char*)fsbase + 0x28);
004013d9
004013e2        if (rax == *(uint64_t*)((char*)fsbase + 0x28))
004013ea            return (uint64_t)(uint8_t)rax_1;
004013ea
004013e4        __stack_chk_fail();
004013e4        /* no return */
```

```
00401364    uint64_t sub_401364(char* arg1)

00401364    {
00401364        void* fsbase;
00401374        int64_t rax = *(uint64_t*)((char*)fsbase + 0x28);
00401396        char var_58[0x3f];
00401396        strncpy(&var_58, arg1, 0x40);
0040139b        char var_19 = 0;
004013a6        sub_401249(&var_58);
004013b2        sub_4012db(&var_58);
004013cf        int32_t rax_1;
004013cf        (uint8_t)rax_1 = !strcmp(&var_58, "tfds{zpv_hpu_ju}");
004013d9        *(uint64_t*)((char*)fsbase + 0x28);
004013d9
004013e2        if (rax == *(uint64_t*)((char*)fsbase + 0x28))
004013ea            return (uint64_t)(uint8_t)rax_1;
004013ea
004013e4        __stack_chk_fail();
004013e4        /* no return */
```

```
00401364    uint64_t sub_401364(char* arg1)

00401364    {
00401364        void* fsbase;
00401374        int64_t rax = *(uint64_t*)((char*)fsbase + 0x28);
00401396        char var_58[0x3f];
00401396        strncpy(&var_58, arg1, 0x40);
0040139b        char var_19 = 0;
004013a6        sub_401249(&var_58);
004013b2        sub_4012db(&var_58);
004013cf        int32_t rax_1;
004013cf        (uint8_t)rax_1 = !strcmp(&var_58, "tfds{zpv_hpu_ju}");
004013d9        *(uint64_t*)((char*)fsbase + 0x28);
004013d9
004013e2        if (rax == *(uint64_t*)((char*)fsbase + 0x28))
004013ea            return (uint64_t)(uint8_t)rax_1;
004013ea
004013e4        __stack_chk_fail();
004013e4        /* no return */
```

Copy arg1 to some buffer

Do some stuff on that buffer

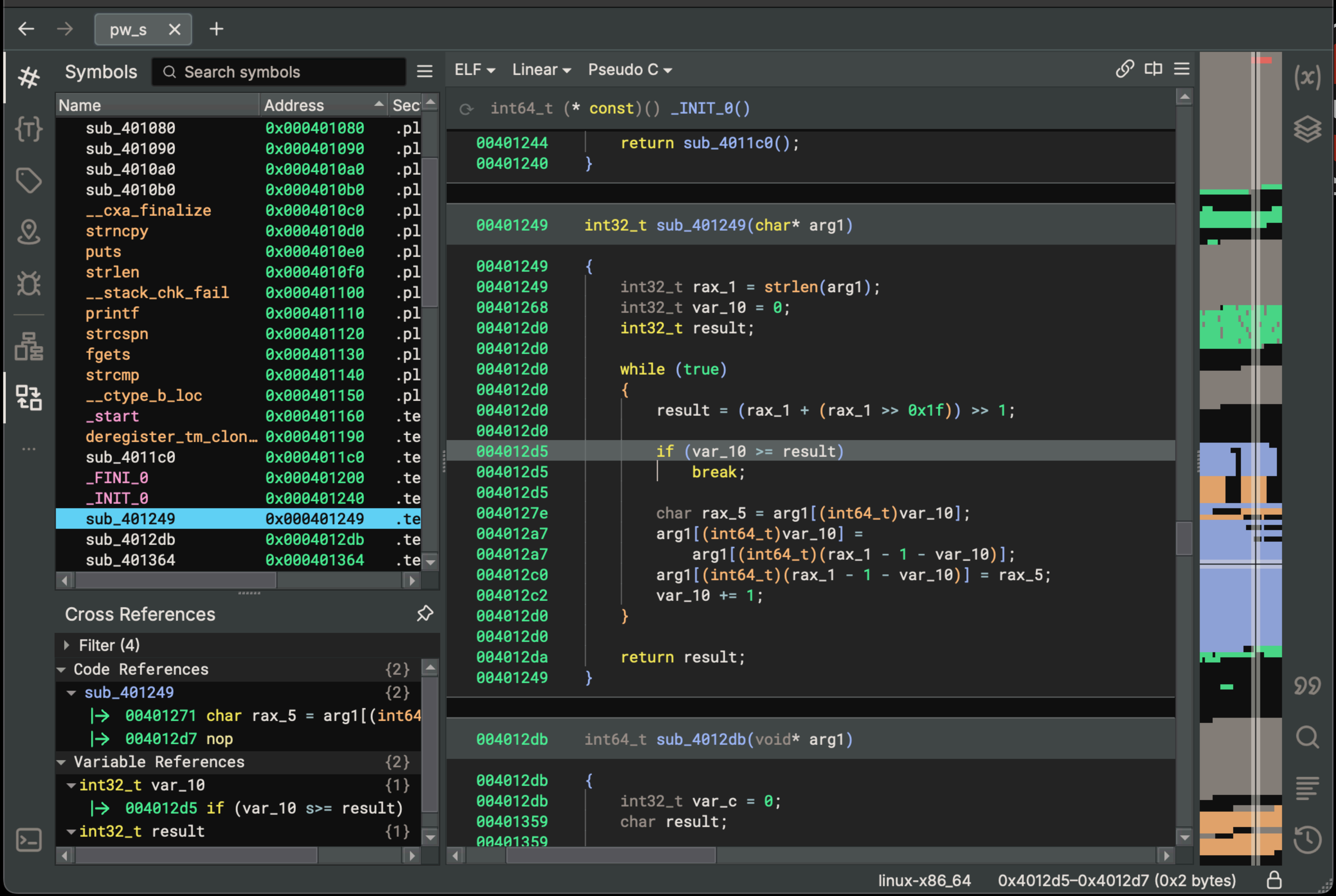
See if it matches this constant

```
00401364    uint64_t sub_401364(char* arg1)

00401364    {
00401364        void* fsbase;
00401374        int64_t rax = *(uint64_t*)((char*)fsbase + 0x28);
00401396        char var_58[0x3f];
00401396        strncpy(&var_58, arg1, 0x40);
0040139b        char var_19 = 0;
004013a6        sub_401249(&var_58);
004013b2        sub_4012db(&var_58);
004013cf        int32_t rax_1;
004013cf        (uint8_t)rax_1 = !strcmp(&var_58, "tfds{zpv_hpu_ju}");
004013d9        *(uint64_t*)((char*)fsbase + 0x28);
004013d9
004013e2        if (rax == *(uint64_t*)((char*)fsbase + 0x28))
004013ea            return (uint64_t)(uint8_t)rax_1;
004013ea
004013e4        __stack_chk_fail();
004013e4        /* no return */
00401364    }
```

Copy arg1 to some buffer

Do some stuff on that buffer

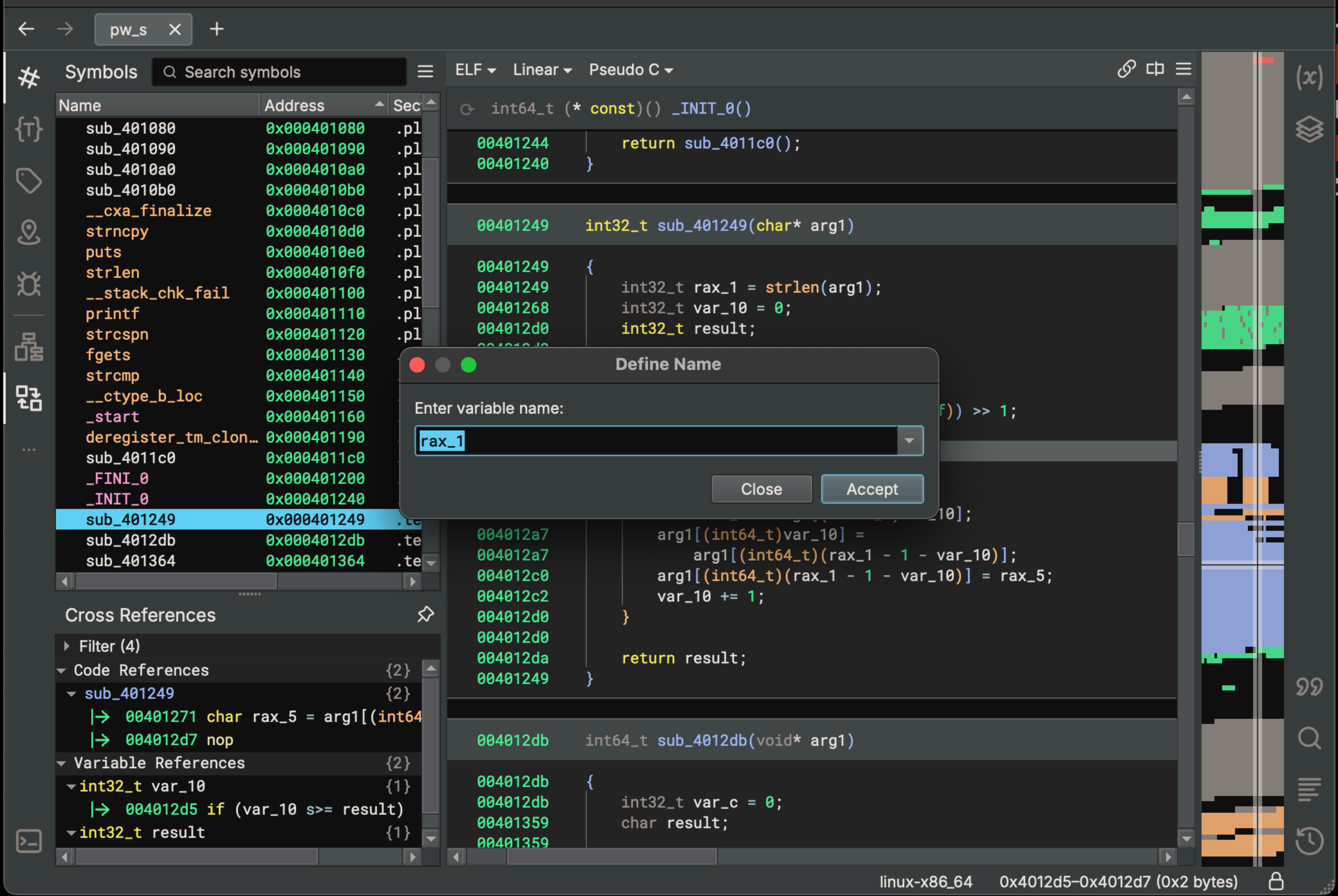
See if it matches this constant



Name	Address	Sec
sub_401080	0x000401080	.pl
sub_401090	0x000401090	.pl
sub_4010a0	0x0004010a0	.pl
sub_4010b0	0x0004010b0	.pl
__cxa_finalize	0x0004010c0	.pl
strncpy	0x0004010d0	.pl
puts	0x0004010e0	.pl
strlen	0x0004010f0	.pl
__stack_chk_fail	0x000401100	.pl
printf	0x000401110	.pl
strcspn	0x000401120	.pl
fgets	0x000401130	.pl
strcmp	0x000401140	.pl
__ctype_b_loc	0x000401150	.pl
_start	0x000401160	.te
deregister_tm_clon…	0x000401190	.te
sub_4011c0	0x0004011c0	.te
_FINI_0	0x000401200	.te
_INIT_0	0x000401240	.te
sub_401249	0x000401249	.te
sub_4012db	0x0004012db	.te
sub_401364	0x000401364	.te

```
int64_t (* const)() _INIT_0()
00401244        return sub_4011c0();
00401240    }

00401249    int32_t sub_401249(char* arg1)

00401249    {
00401249        int32_t rax_1 = strlen(arg1);
00401268        int32_t var_10 = 0;
004012d0        int32_t result;
004012d0
004012d0        while (true)
004012d0        {
004012d0            result = (rax_1 + (rax_1 >> 0x1f)) >> 1;
004012d0
004012d5            if (var_10 >= result)
004012d5                break;
004012d5
0040127e            char rax_5 = arg1[(int64_t)var_10];
004012a7            arg1[(int64_t)var_10] =
004012a7                arg1[(int64_t)(rax_1 - 1 - var_10)];
004012c0            arg1[(int64_t)(rax_1 - 1 - var_10)] = rax_5;
004012c2            var_10 += 1;
004012d0        }
004012d0
004012da        return result;
00401249    }

004012db    int64_t sub_4012db(void* arg1)

004012db    {
004012db        int32_t var_c = 0;
00401359        char result;
00401359
```

```
Code References                                   {2}
  sub_401249                                      {2}
    |→ 00401271 char rax_5 = arg1[(int64
    |→ 004012d7 nop
Variable References                               {2}
  int32_t var_10                                  {1}
    |→ 004012d5 if (var_10 s>= result)
  int32_t result                                  {1}
```



Name	Address	Sec
sub_401080	0x000401080	.pl
sub_401090	0x000401090	.pl
sub_4010a0	0x0004010a0	.pl
sub_4010b0	0x0004010b0	.pl
__cxa_finalize	0x0004010c0	.pl
strncpy	0x0004010d0	.pl
puts	0x0004010e0	.pl
strlen	0x0004010f0	.pl
__stack_chk_fail	0x000401100	.pl
printf	0x000401110	.pl
strcspn	0x000401120	.pl
fgets	0x000401130	
strcmp	0x000401140	
__ctype_b_loc	0x000401150	
_start	0x000401160	
deregister_tm_clon…	0x000401190	
sub_4011c0	0x0004011c0	
_FINI_0	0x000401200	
_INIT_0	0x000401240	
sub_401249	0x000401249	.te
sub_4012db	0x0004012db	.te
sub_401364	0x000401364	.te

```
int64_t (* const)() _INIT_0()
00401244        return sub_4011c0();
00401240    }

00401249    int32_t sub_401249(char* arg1)

00401249    {
00401249        int32_t rax_1 = strlen(arg1);
00401268        int32_t var_10 = 0;
004012d0        int32_t result;
                                          f)) >> 1;

                                          10];
004012a7            arg1[(int64_t)var_10] =
004012a7                arg1[(int64_t)(rax_1 - 1 - var_10)];
004012c0            arg1[(int64_t)(rax_1 - 1 - var_10)] = rax_5;
004012c2            var_10 += 1;
004012d0        }
004012d0
004012da        return result;
00401249    }

004012db    int64_t sub_4012db(void* arg1)

004012db    {
004012db        int32_t var_c = 0;
00401359        char result;
00401359
```

Define Name

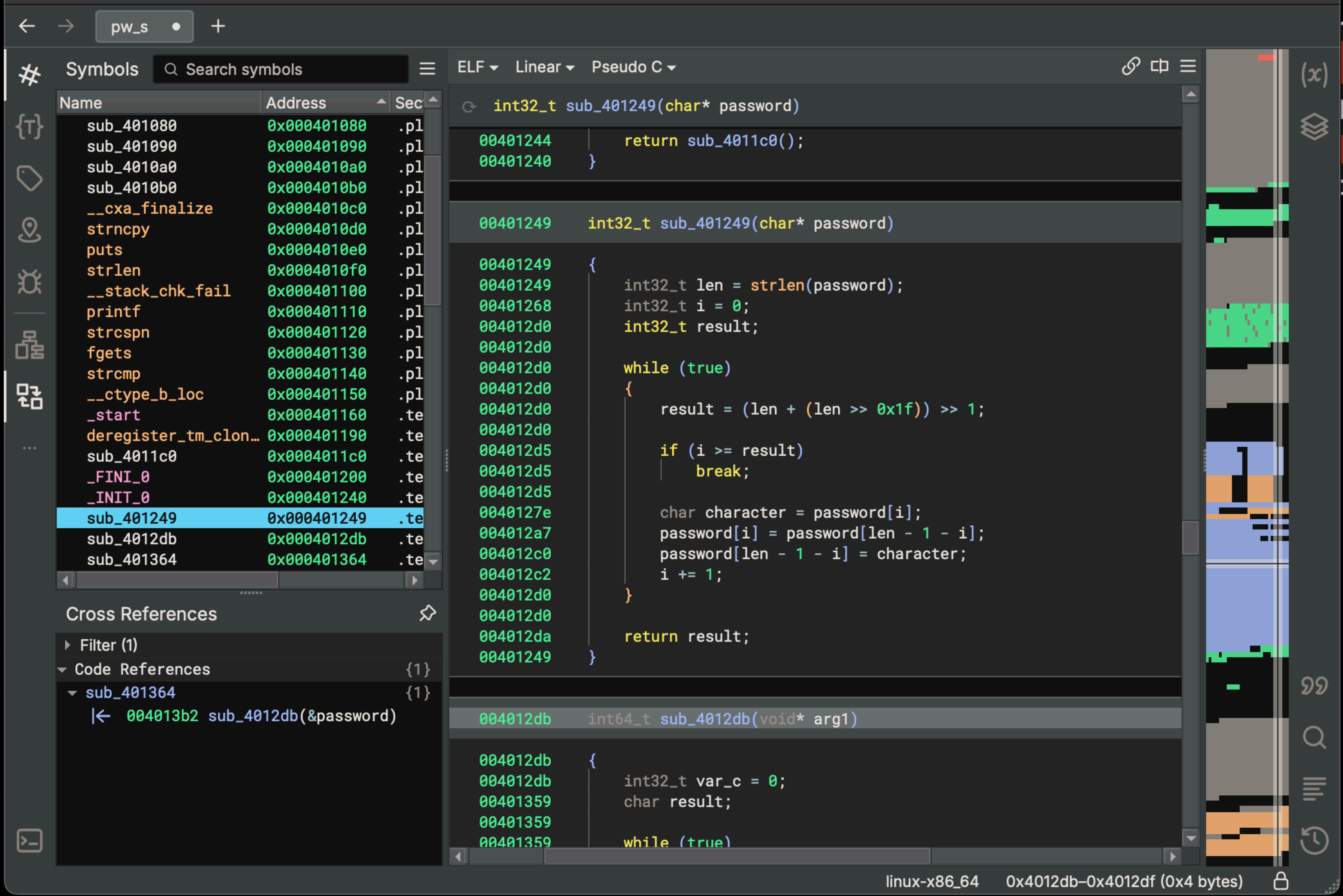
Enter variable name:

rax_1

Close Accept

```
Code References                                {2}
  sub_401249                                   {2}
    |-> 00401271 char rax_5 = arg1[(int64
    |-> 004012d7 nop
Variable References                            {2}
  int32_t var_10                               {1}
    |-> 004012d5 if (var_10 s>= result)
  int32_t result                               {1}
```

linux-x86_64 0x4012d5–0x4012d7 (0x2 bytes)

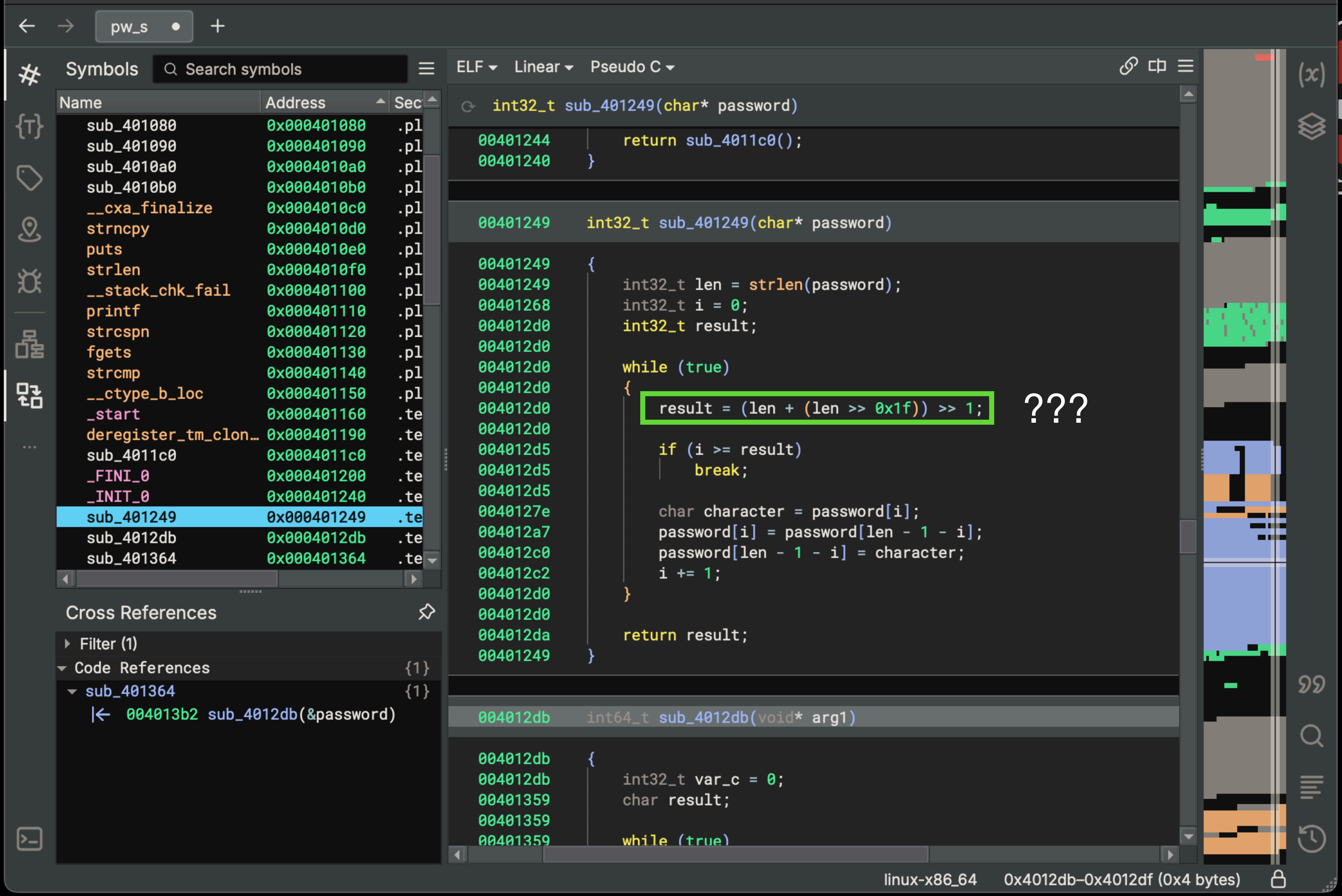


Name	Address	Sec
sub_401080	0x000401080	.pl
sub_401090	0x000401090	.pl
sub_4010a0	0x0004010a0	.pl
sub_4010b0	0x0004010b0	.pl
__cxa_finalize	0x0004010c0	.pl
strncpy	0x0004010d0	.pl
puts	0x0004010e0	.pl
strlen	0x0004010f0	.pl
__stack_chk_fail	0x000401100	.pl
printf	0x000401110	.pl
strcspn	0x000401120	.pl
fgets	0x000401130	.pl
strcmp	0x000401140	.pl
__ctype_b_loc	0x000401150	.pl
_start	0x000401160	.te
deregister_tm_clon…	0x000401190	.te
sub_4011c0	0x0004011c0	.te
_FINI_0	0x000401200	.te
_INIT_0	0x000401240	.te
sub_401249	0x000401249	.te
sub_4012db	0x0004012db	.te
sub_401364	0x000401364	.te

```
int32_t sub_401249(char* password)

00401244        return sub_4011c0();
00401240    }

00401249    int32_t sub_401249(char* password)

00401249    {
00401249        int32_t len = strlen(password);
00401268        int32_t i = 0;
004012d0        int32_t result;
004012d0
004012d0        while (true)
004012d0        {
004012d0            result = (len + (len >> 0x1f)) >> 1;
004012d0
004012d5            if (i >= result)
004012d5                break;
004012d5
0040127e            char character = password[i];
004012a7            password[i] = password[len - 1 - i];
004012c0            password[len - 1 - i] = character;
004012c2            i += 1;
004012d0        }
004012d0
004012da        return result;
00401249    }

004012db    int64_t sub_4012db(void* arg1)

004012db    {
004012db        int32_t var_c = 0;
00401359        char result;
00401359
00401359        while (true)
```

Filter (1)
Code References {1}
sub_401364 {1}
004013b2 sub_4012db(&password)

linux-x86_64   0x4012db–0x4012df (0x4 bytes)

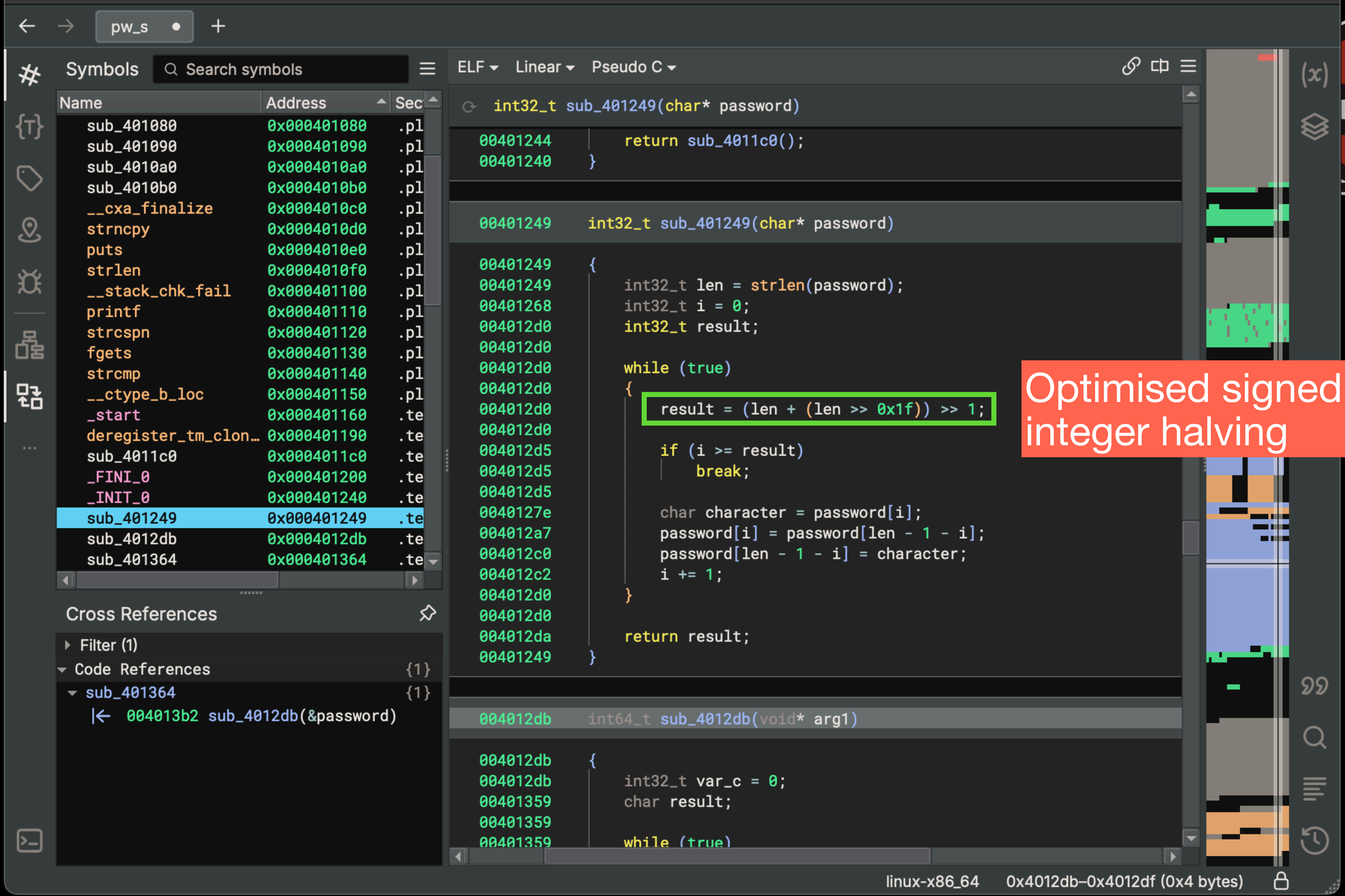


Name	Address	Sec
sub_401080	0x000401080	.pl
sub_401090	0x000401090	.pl
sub_4010a0	0x0004010a0	.pl
sub_4010b0	0x0004010b0	.pl
__cxa_finalize	0x0004010c0	.pl
strncpy	0x0004010d0	.pl
puts	0x0004010e0	.pl
strlen	0x0004010f0	.pl
__stack_chk_fail	0x000401100	.pl
printf	0x000401110	.pl
strcspn	0x000401120	.pl
fgets	0x000401130	.pl
strcmp	0x000401140	.pl
__ctype_b_loc	0x000401150	.pl
_start	0x000401160	.te
deregister_tm_clon…	0x000401190	.te
sub_4011c0	0x0004011c0	.te
_FINI_0	0x000401200	.te
_INIT_0	0x000401240	.te
sub_401249	0x000401249	.te
sub_4012db	0x0004012db	.te
sub_401364	0x000401364	.te

```
Code References                {1}
  sub_401364                   {1}
    004013b2 sub_4012db(&password)
```

```
int32_t sub_401249(char* password)

00401244        return sub_4011c0();
00401240    }

00401249    int32_t sub_401249(char* password)

00401249    {
00401249        int32_t len = strlen(password);
00401268        int32_t i = 0;
004012d0        int32_t result;
004012d0
004012d0        while (true)
004012d0        {
004012d0            result = (len + (len >> 0x1f)) >> 1;
004012d0
004012d5            if (i >= result)
004012d5                break;
004012d5
0040127e            char character = password[i];
004012a7            password[i] = password[len - 1 - i];
004012c0            password[len - 1 - i] = character;
004012c2            i += 1;
004012d0        }
004012d0
004012da        return result;
00401249    }

004012db    int64_t sub_4012db(void* arg1)

004012db    {
004012db        int32_t var_c = 0;
00401359        char result;
00401359
00401359        while (true)
```

???

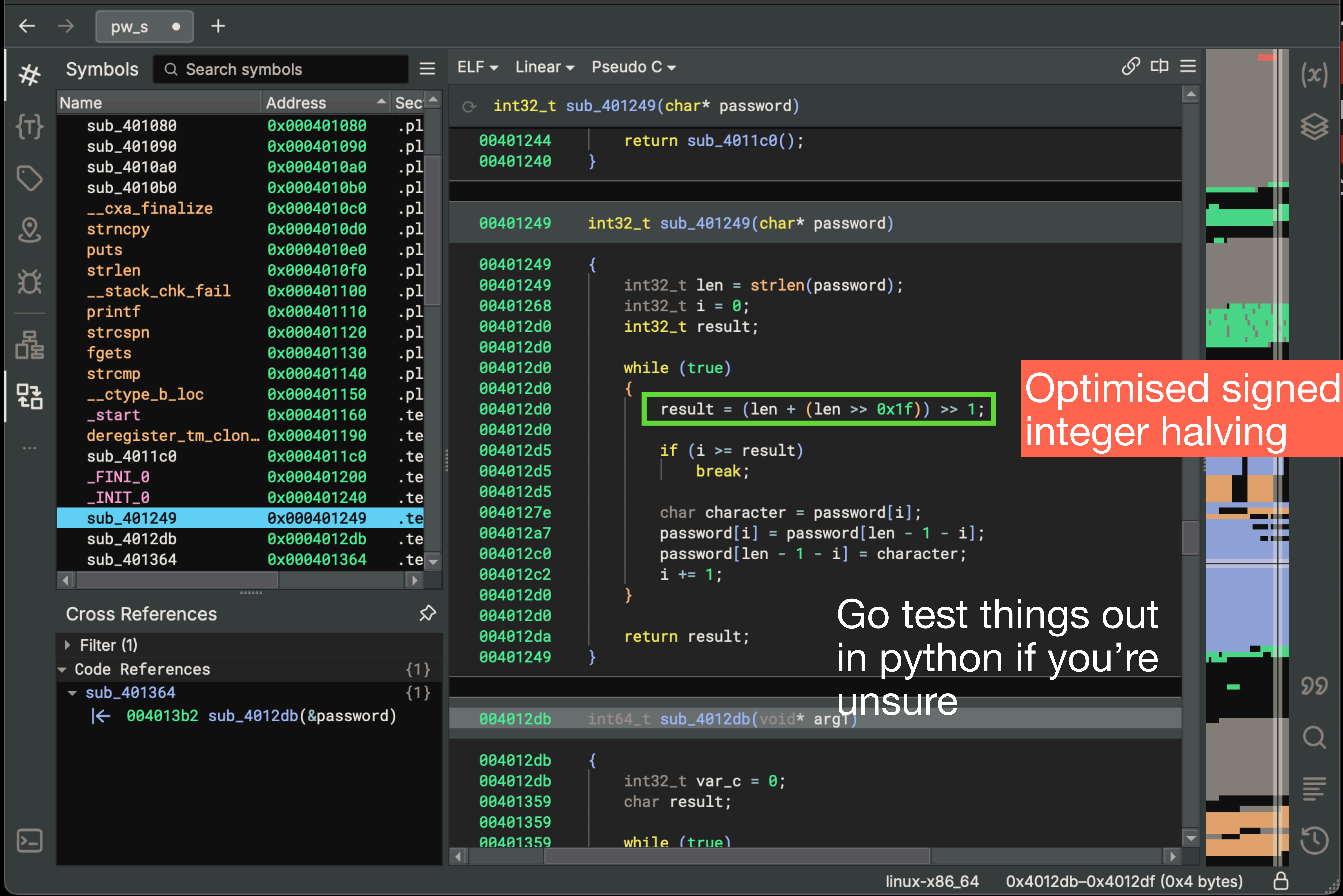


Name	Address	Sec
sub_401080	0x000401080	.pl
sub_401090	0x000401090	.pl
sub_4010a0	0x0004010a0	.pl
sub_4010b0	0x0004010b0	.pl
__cxa_finalize	0x0004010c0	.pl
strncpy	0x0004010d0	.pl
puts	0x0004010e0	.pl
strlen	0x0004010f0	.pl
__stack_chk_fail	0x000401100	.pl
printf	0x000401110	.pl
strcspn	0x000401120	.pl
fgets	0x000401130	.pl
strcmp	0x000401140	.pl
__ctype_b_loc	0x000401150	.pl
_start	0x000401160	.te
deregister_tm_clon…	0x000401190	.te
sub_4011c0	0x0004011c0	.te
_FINI_0	0x000401200	.te
_INIT_0	0x000401240	.te
sub_401249	0x000401249	.te
sub_4012db	0x0004012db	.te
sub_401364	0x000401364	.te

```
Code References                    {1}
  sub_401364                       {1}
    004013b2 sub_4012db(&password)
```

```
int32_t sub_401249(char* password)

00401244        return sub_4011c0();
00401240    }

00401249    int32_t sub_401249(char* password)

00401249    {
00401249        int32_t len = strlen(password);
00401268        int32_t i = 0;
004012d0        int32_t result;
004012d0
004012d0        while (true)
004012d0        {
004012d0            result = (len + (len >> 0x1f)) >> 1;
004012d0
004012d5            if (i >= result)
004012d5                break;
004012d5
0040127e            char character = password[i];
004012a7            password[i] = password[len - 1 - i];
004012c0            password[len - 1 - i] = character;
004012c2            i += 1;
004012d0        }
004012d0
004012da        return result;
00401249    }

004012db    int64_t sub_4012db(void* arg1)

004012db    {
004012db        int32_t var_c = 0;
00401359        char result;
00401359
00401359        while (true)
```

Optimised signed integer halving



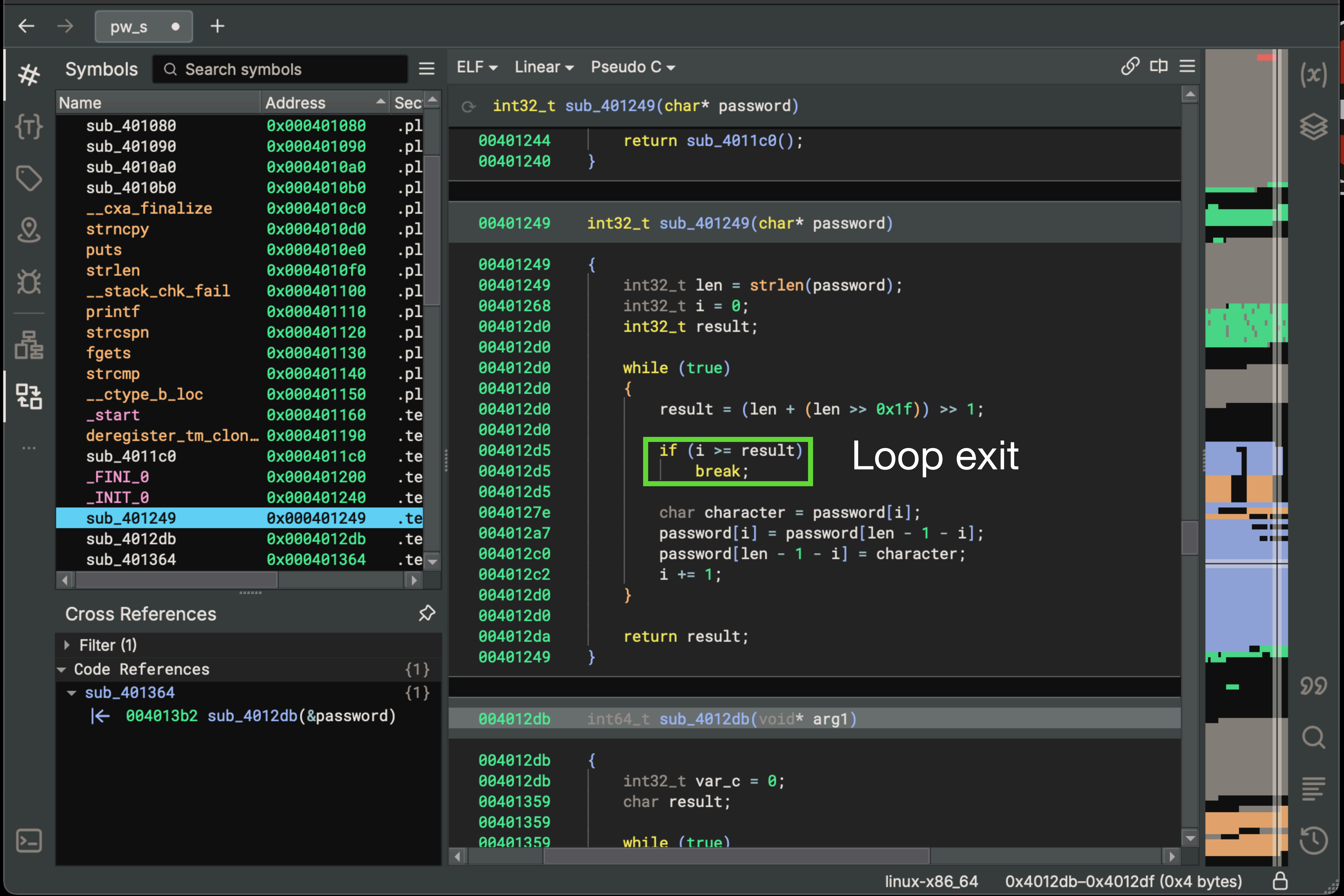
Name	Address	Sec
sub_401080	0x000401080	.pl
sub_401090	0x000401090	.pl
sub_4010a0	0x0004010a0	.pl
sub_4010b0	0x0004010b0	.pl
__cxa_finalize	0x0004010c0	.pl
strncpy	0x0004010d0	.pl
puts	0x0004010e0	.pl
strlen	0x0004010f0	.pl
__stack_chk_fail	0x000401100	.pl
printf	0x000401110	.pl
strcspn	0x000401120	.pl
fgets	0x000401130	.pl
strcmp	0x000401140	.pl
__ctype_b_loc	0x000401150	.pl
_start	0x000401160	.te
deregister_tm_clon…	0x000401190	.te
sub_4011c0	0x0004011c0	.te
_FINI_0	0x000401200	.te
_INIT_0	0x000401240	.te
sub_401249	0x000401249	.te
sub_4012db	0x0004012db	.te
sub_401364	0x000401364	.te

```
Code References                              {1}
  sub_401364                                 {1}
    |← 004013b2 sub_4012db(&password)
```

```
int32_t sub_401249(char* password)

00401244        return sub_4011c0();
00401240    }

00401249    int32_t sub_401249(char* password)

00401249    {
00401249        int32_t len = strlen(password);
00401268        int32_t i = 0;
004012d0        int32_t result;
004012d0
004012d0        while (true)
004012d0        {
004012d0            result = (len + (len >> 0x1f)) >> 1;
004012d0
004012d5            if (i >= result)
004012d5                break;
004012d5
0040127e            char character = password[i];
004012a7            password[i] = password[len - 1 - i];
004012c0            password[len - 1 - i] = character;
004012c2            i += 1;
004012d0        }
004012d0
004012da        return result;
00401249    }

004012db    int64_t sub_4012db(void* arg1)

004012db    {
004012db        int32_t var_c = 0;
00401359        char result;
00401359
00401359        while (true)
```

Optimised signed integer halving

Go test things out in python if you’re unsure

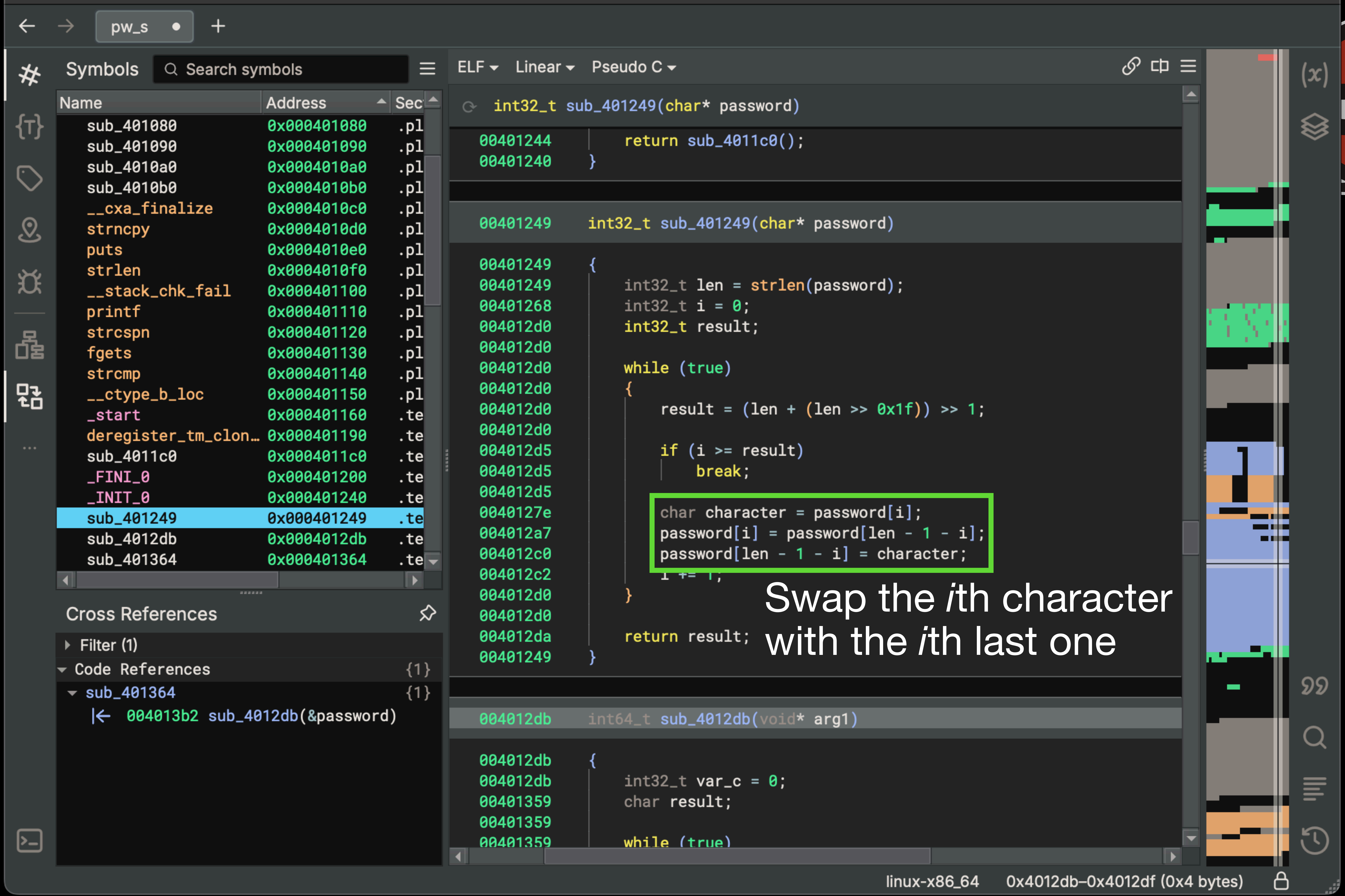


Name	Address	Sec
sub_401080	0x000401080	.pl
sub_401090	0x000401090	.pl
sub_4010a0	0x0004010a0	.pl
sub_4010b0	0x0004010b0	.pl
__cxa_finalize	0x0004010c0	.pl
strncpy	0x0004010d0	.pl
puts	0x0004010e0	.pl
strlen	0x0004010f0	.pl
__stack_chk_fail	0x000401100	.pl
printf	0x000401110	.pl
strcspn	0x000401120	.pl
fgets	0x000401130	.pl
strcmp	0x000401140	.pl
__ctype_b_loc	0x000401150	.pl
_start	0x000401160	.te
deregister_tm_clon…	0x000401190	.te
sub_4011c0	0x0004011c0	.te
_FINI_0	0x000401200	.te
_INIT_0	0x000401240	.te
sub_401249	0x000401249	.te
sub_4012db	0x0004012db	.te
sub_401364	0x000401364	.te

```
Code References                      {1}
  sub_401364                         {1}
    |← 004013b2 sub_4012db(&password)
```

```
int32_t sub_401249(char* password)

00401244        return sub_4011c0();
00401240    }

00401249    int32_t sub_401249(char* password)

00401249    {
00401249        int32_t len = strlen(password);
00401268        int32_t i = 0;
004012d0        int32_t result;
004012d0
004012d0        while (true)
004012d0        {
004012d0            result = (len + (len >> 0x1f)) >> 1;
004012d0
004012d5            if (i >= result)
004012d5                break;
004012d5
0040127e            char character = password[i];
004012a7            password[i] = password[len - 1 - i];
004012c0            password[len - 1 - i] = character;
004012c2            i += 1;
004012d0        }
004012d0
004012da        return result;
00401249    }

004012db    int64_t sub_4012db(void* arg1)

004012db    {
004012db        int32_t var_c = 0;
00401359        char result;
00401359
00401359        while (true)
```

Loop exit

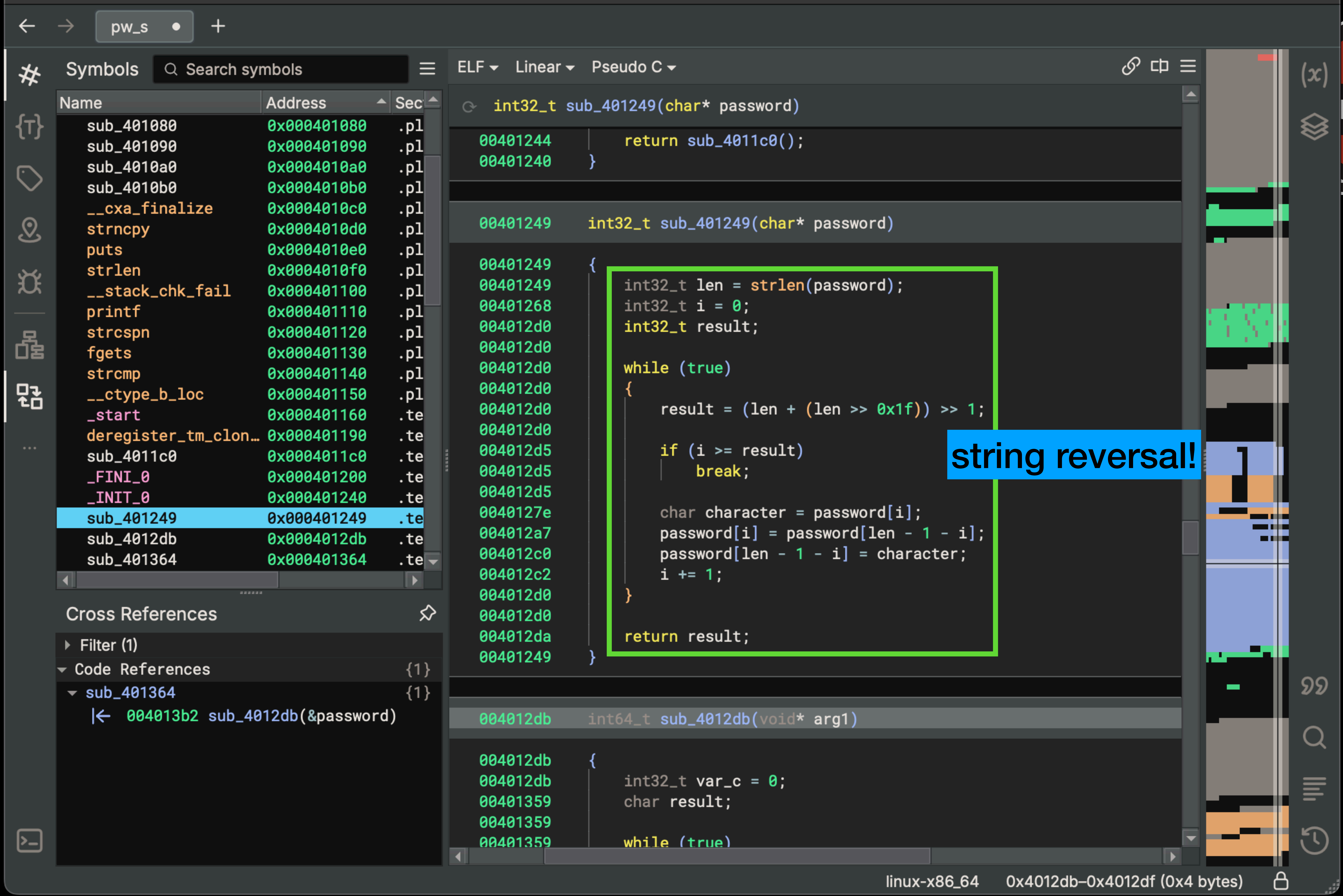


Name	Address	Sec
sub_401080	0x000401080	.pl
sub_401090	0x000401090	.pl
sub_4010a0	0x0004010a0	.pl
sub_4010b0	0x0004010b0	.pl
__cxa_finalize	0x0004010c0	.pl
strncpy	0x0004010d0	.pl
puts	0x0004010e0	.pl
strlen	0x0004010f0	.pl
__stack_chk_fail	0x000401100	.pl
printf	0x000401110	.pl
strcspn	0x000401120	.pl
fgets	0x000401130	.pl
strcmp	0x000401140	.pl
__ctype_b_loc	0x000401150	.pl
_start	0x000401160	.te
deregister_tm_clon…	0x000401190	.te
sub_4011c0	0x0004011c0	.te
_FINI_0	0x000401200	.te
_INIT_0	0x000401240	.te
sub_401249	0x000401249	.te
sub_4012db	0x0004012db	.te
sub_401364	0x000401364	.te

```
Code References                {1}
  sub_401364                   {1}
    |← 004013b2 sub_4012db(&password)
```

```
int32_t sub_401249(char* password)

00401244        return sub_4011c0();
00401240    }

00401249    int32_t sub_401249(char* password)

00401249    {
00401249        int32_t len = strlen(password);
00401268        int32_t i = 0;
004012d0        int32_t result;
004012d0
004012d0        while (true)
004012d0        {
004012d0            result = (len + (len >> 0x1f)) >> 1;
004012d0
004012d5            if (i >= result)
004012d5                break;
004012d5
0040127e            char character = password[i];
004012a7            password[i] = password[len - 1 - i];
004012c0            password[len - 1 - i] = character;
004012c2            i += 1;
004012d0        }
004012d0
004012da        return result;
00401249    }

004012db    int64_t sub_4012db(void* arg1)

004012db    {
004012db        int32_t var_c = 0;
00401359        char result;
00401359
00401359        while (true)
```

Swap the *i*th character with the *i*th last one

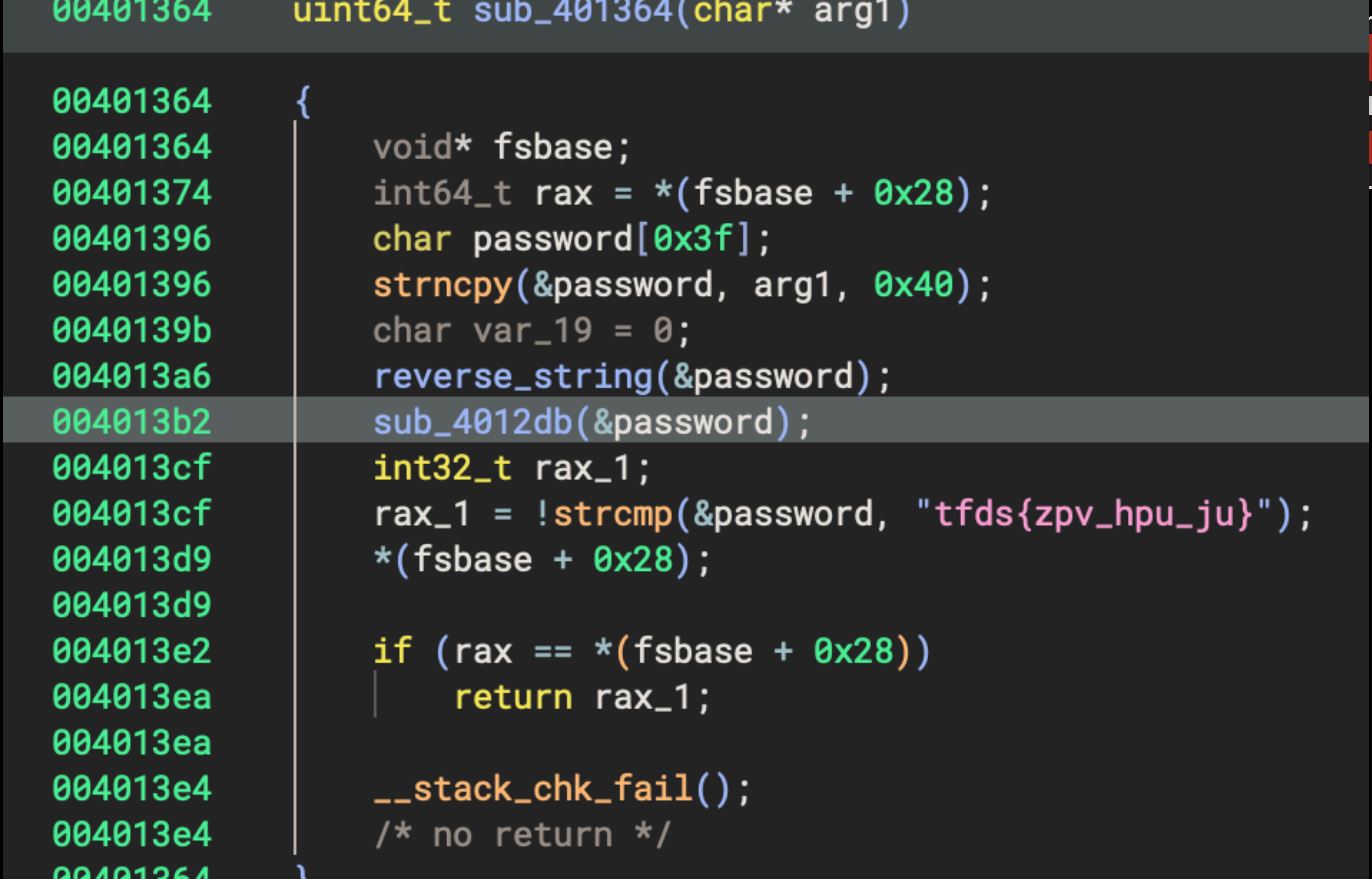


Name	Address	Sec
sub_401080	0x000401080	.pl
sub_401090	0x000401090	.pl
sub_4010a0	0x0004010a0	.pl
sub_4010b0	0x0004010b0	.pl
__cxa_finalize	0x0004010c0	.pl
strncpy	0x0004010d0	.pl
puts	0x0004010e0	.pl
strlen	0x0004010f0	.pl
__stack_chk_fail	0x000401100	.pl
printf	0x000401110	.pl
strcspn	0x000401120	.pl
fgets	0x000401130	.pl
strcmp	0x000401140	.pl
__ctype_b_loc	0x000401150	.pl
_start	0x000401160	.te
deregister_tm_clon…	0x000401190	.te
sub_4011c0	0x0004011c0	.te
_FINI_0	0x000401200	.te
_INIT_0	0x000401240	.te
sub_401249	0x000401249	.te
sub_4012db	0x0004012db	.te
sub_401364	0x000401364	.te

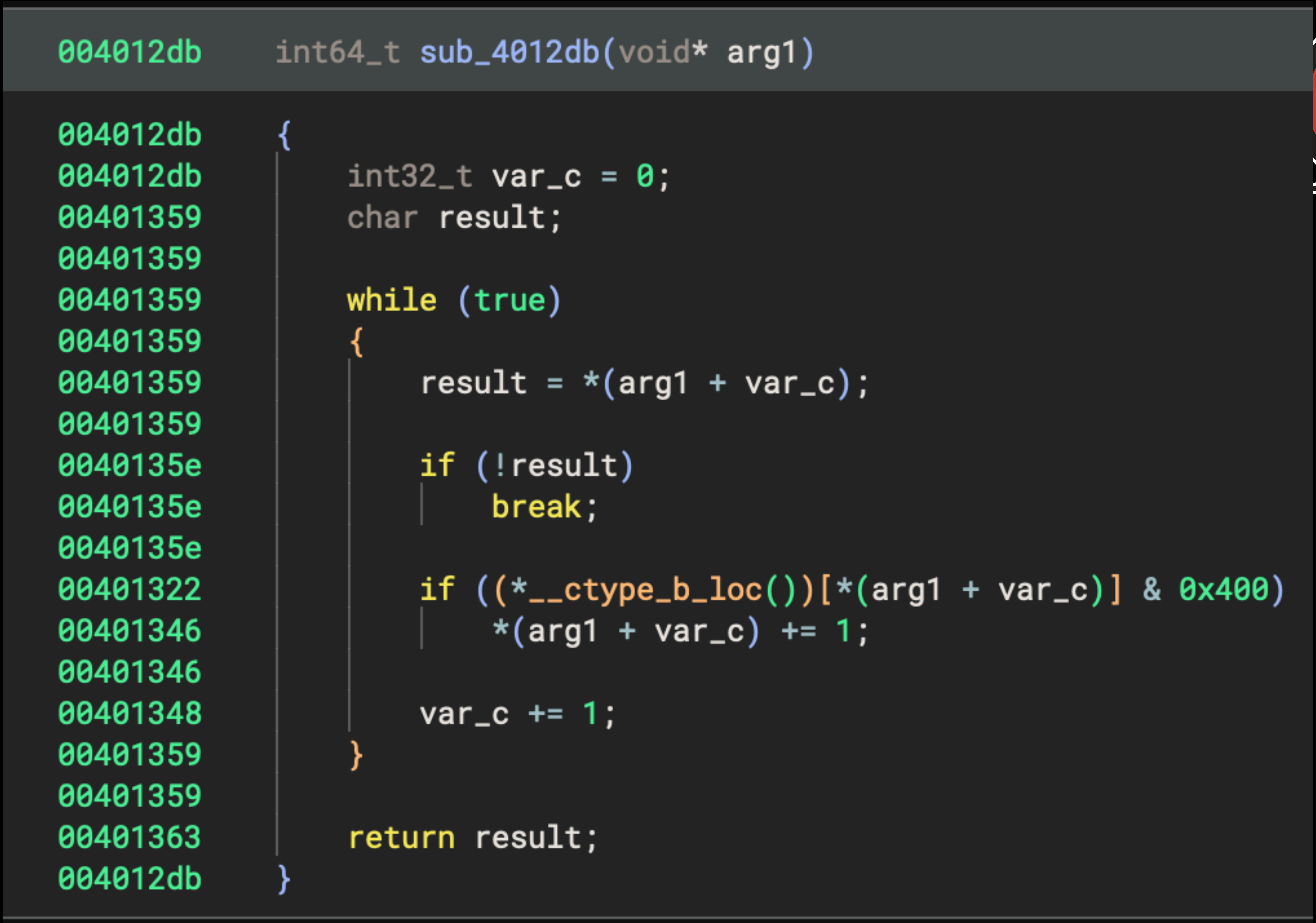
```
Code References                     {1}
  sub_401364                        {1}
    |← 004013b2 sub_4012db(&password)
```

```
int32_t sub_401249(char* password)
00401244        return sub_4011c0();
00401240    }

00401249    int32_t sub_401249(char* password)

00401249    {
00401249        int32_t len = strlen(password);
00401268        int32_t i = 0;
004012d0        int32_t result;
004012d0
004012d0        while (true)
004012d0        {
004012d0            result = (len + (len >> 0x1f)) >> 1;
004012d0
004012d5            if (i >= result)
004012d5                break;
004012d5
0040127e            char character = password[i];
004012a7            password[i] = password[len - 1 - i];
004012c0            password[len - 1 - i] = character;
004012c2            i += 1;
004012d0        }
004012d0
004012da        return result;
00401249    }

004012db    int64_t sub_4012db(void* arg1)

004012db    {
004012db        int32_t var_c = 0;
00401359        char result;
00401359
00401359        while (true)
```

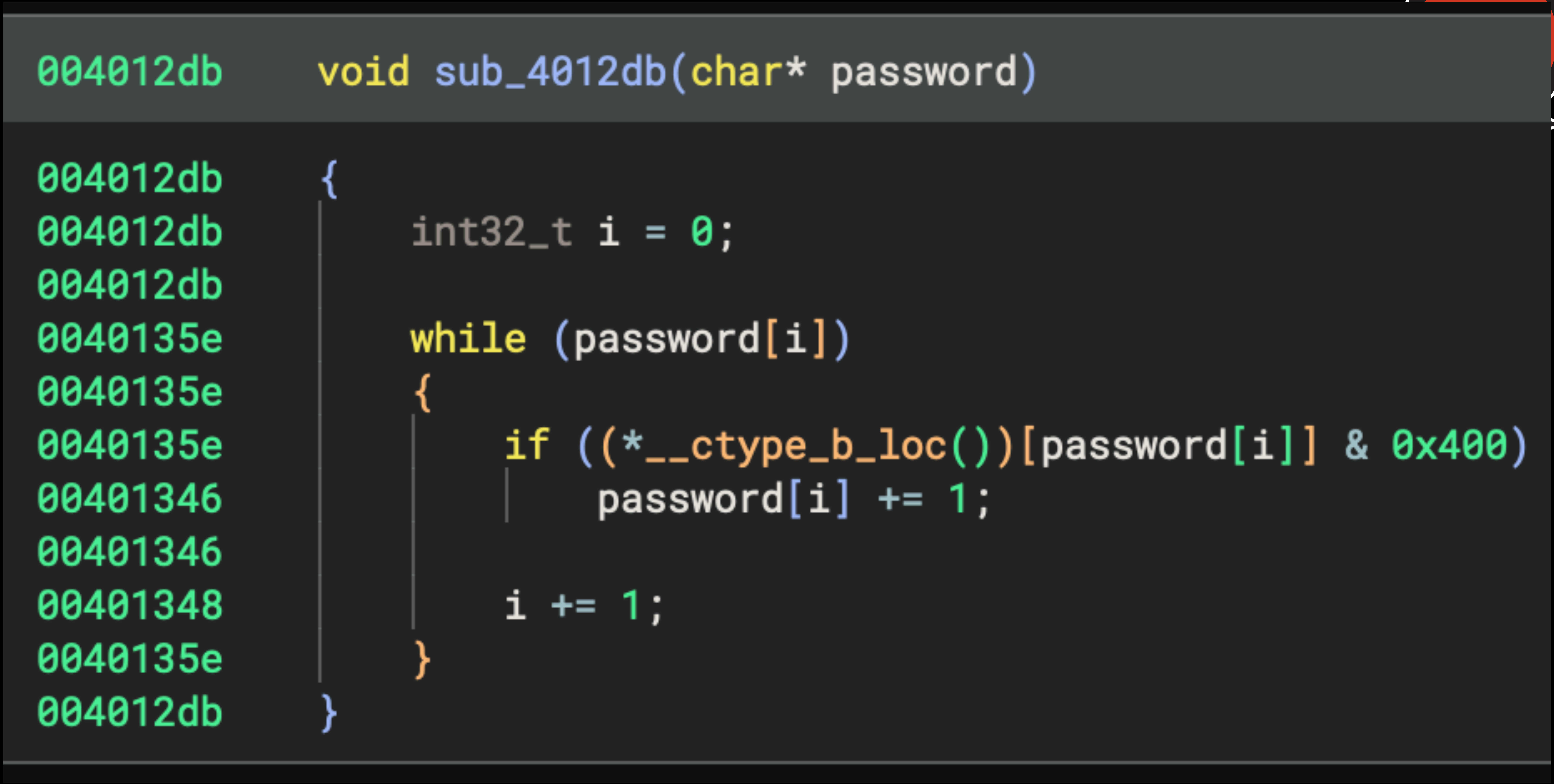
string reversal!



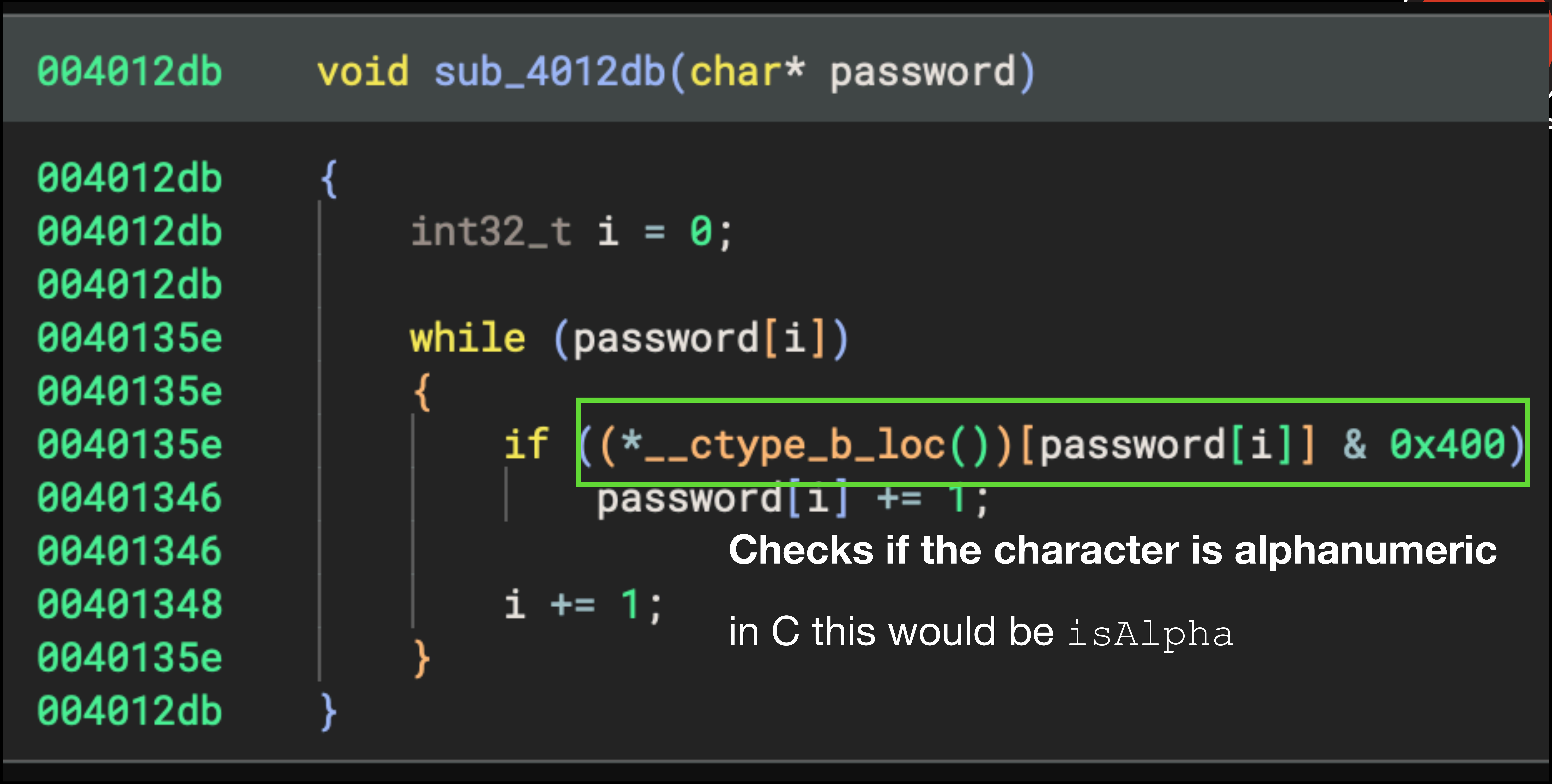
```
00401364    uint64_t sub_401364(char* arg1)

00401364    {
00401364        void* fsbase;
00401374        int64_t rax = *(fsbase + 0x28);
00401396        char password[0x3f];
00401396        strncpy(&password, arg1, 0x40);
0040139b        char var_19 = 0;
004013a6        reverse_string(&password);
004013b2        sub_4012db(&password);
004013cf        int32_t rax_1;
004013cf        rax_1 = !strcmp(&password, "tfds{zpv_hpu_ju}");
004013d9        *(fsbase + 0x28);
004013d9
004013e2        if (rax == *(fsbase + 0x28))
004013ea            return rax_1;
004013ea
004013e4        __stack_chk_fail();
004013e4        /* no return */
00401364    }
```



```
004012db    int64_t sub_4012db(void* arg1)

004012db    {
004012db        int32_t var_c = 0;
00401359        char result;
00401359
00401359        while (true)
00401359        {
00401359            result = *(arg1 + var_c);
00401359
0040135e            if (!result)
0040135e                break;
0040135e
00401322            if ((*__ctype_b_loc())[*(arg1 + var_c)] & 0x400)
00401346                *(arg1 + var_c) += 1;
00401346
00401348            var_c += 1;
00401359        }
00401359
00401363        return result;
004012db    }
```



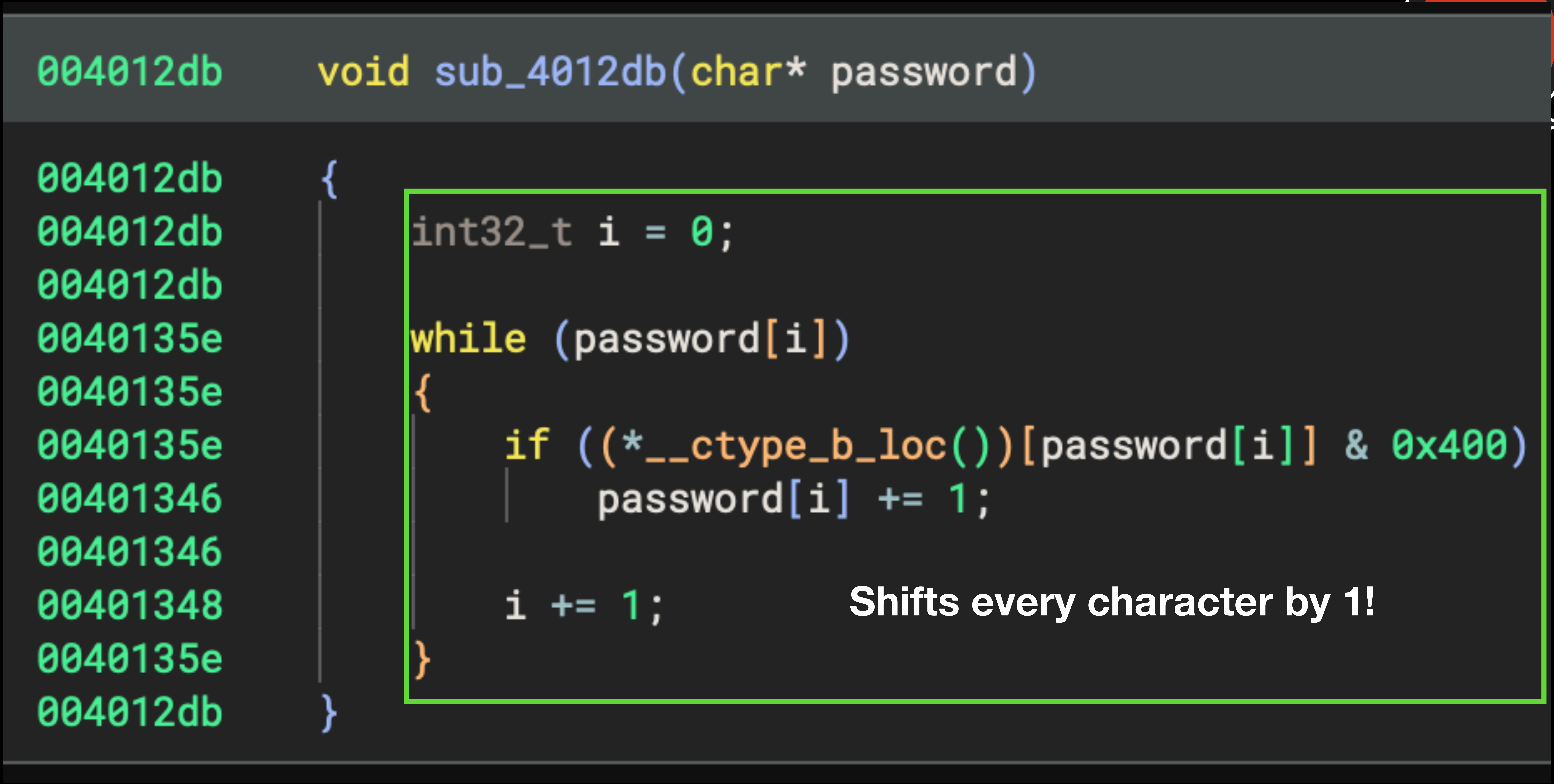
```
004012db    void sub_4012db(char* password)

004012db    {
004012db        int32_t i = 0;
004012db    
0040135e        while (password[i])
0040135e        {
0040135e            if ((*__ctype_b_loc())[password[i]] & 0x400)
00401346                password[i] += 1;
00401346    
00401348            i += 1;
0040135e        }
004012db    }
```



```
004012db    void sub_4012db(char* password)

004012db    {
004012db        int32_t i = 0;
004012db
0040135e        while (password[i])
0040135e        {
0040135e            if ((*__ctype_b_loc())[password[i]] & 0x400)
00401346                password[i] += 1;
00401346
00401348            i += 1;
0040135e        }
004012db    }
```

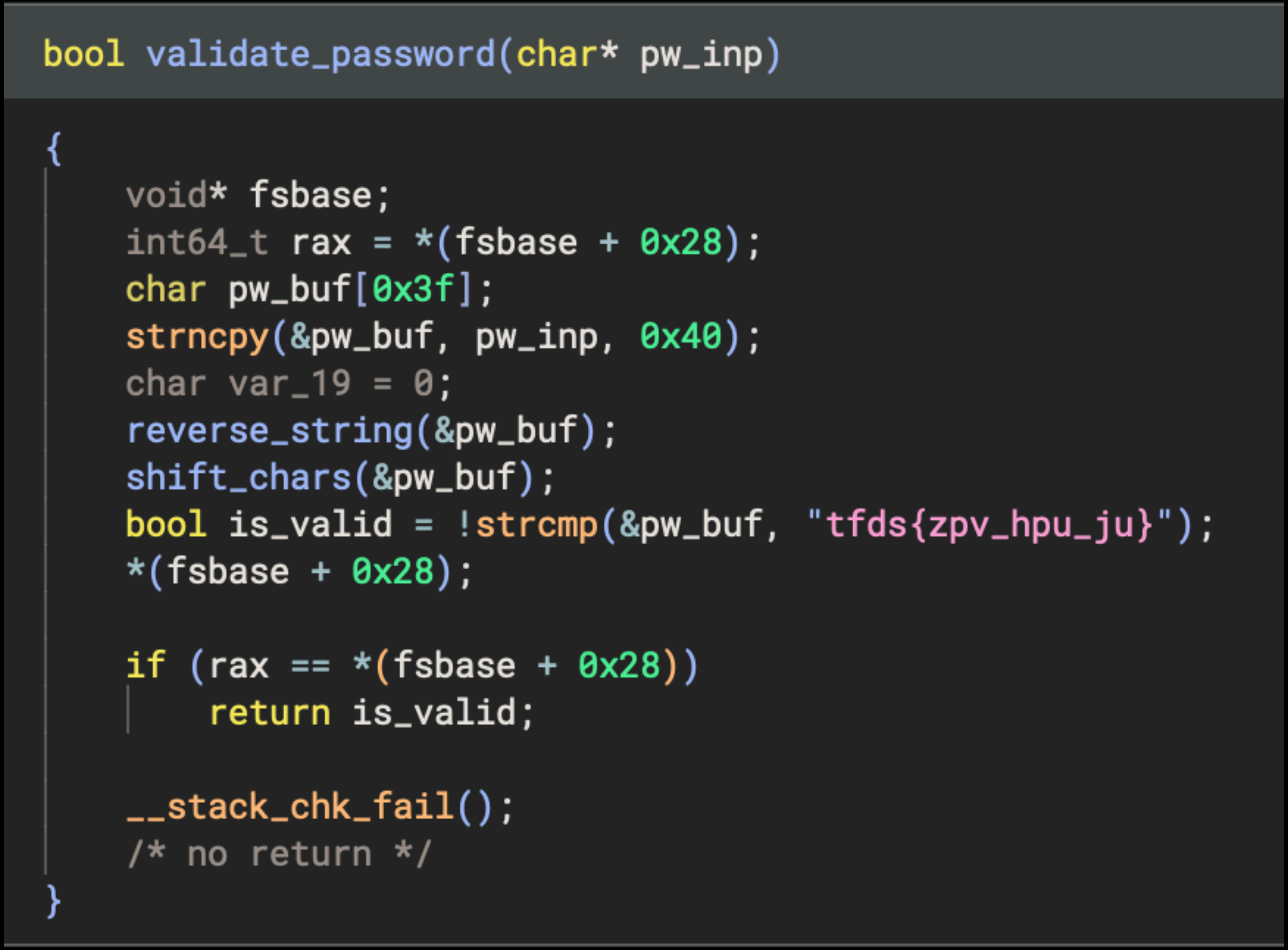
Checks if the character is alphanumeric

in C this would be `isAlpha`

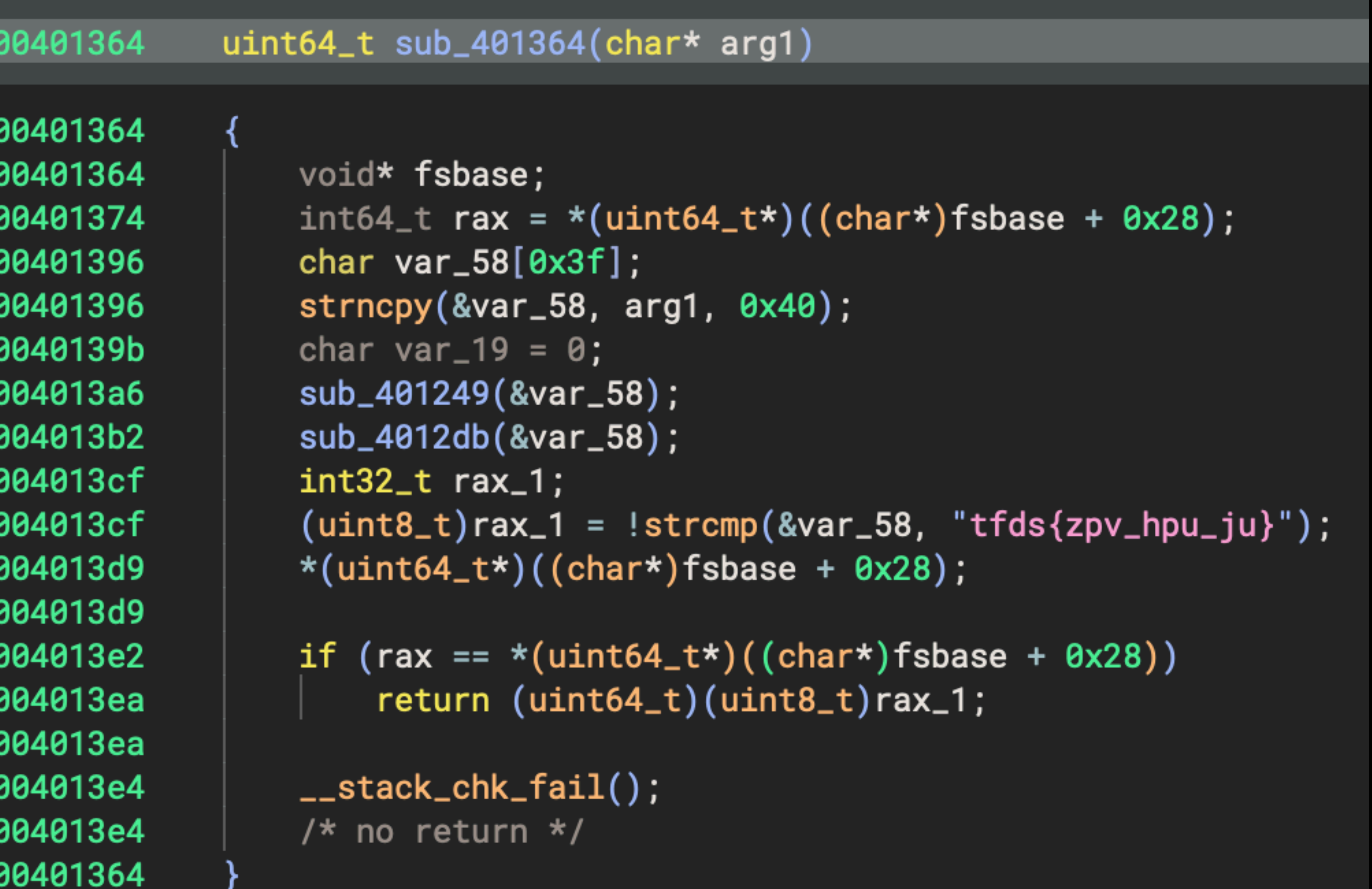


```
004012db    void sub_4012db(char* password)

004012db    {
004012db        int32_t i = 0;
004012db
0040135e        while (password[i])
0040135e        {
0040135e            if ((*__ctype_b_loc())[password[i]] & 0x400)
00401346                password[i] += 1;
00401346
00401348            i += 1;
0040135e        }
004012db    }
```

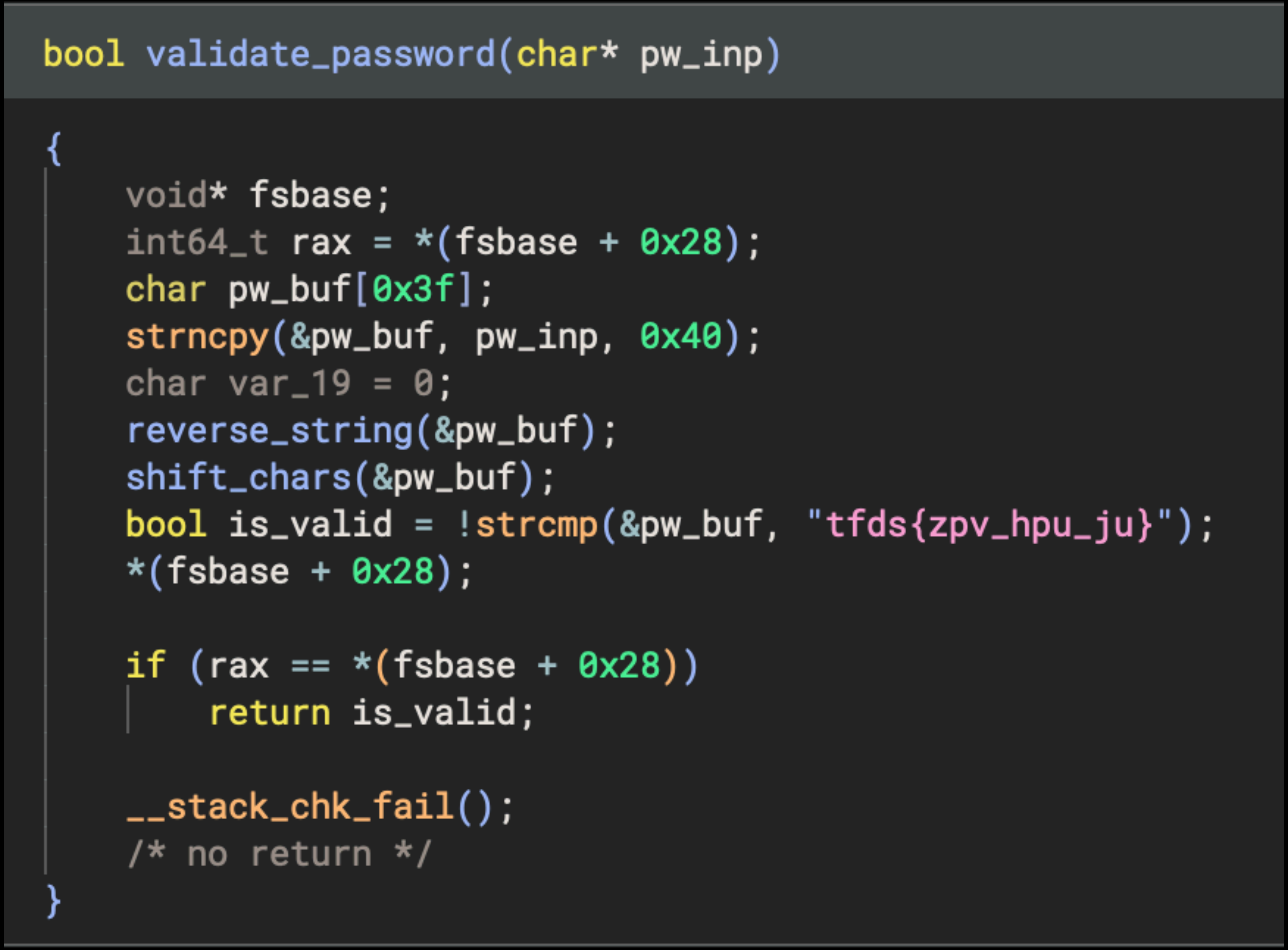
Shifts every character by 1!



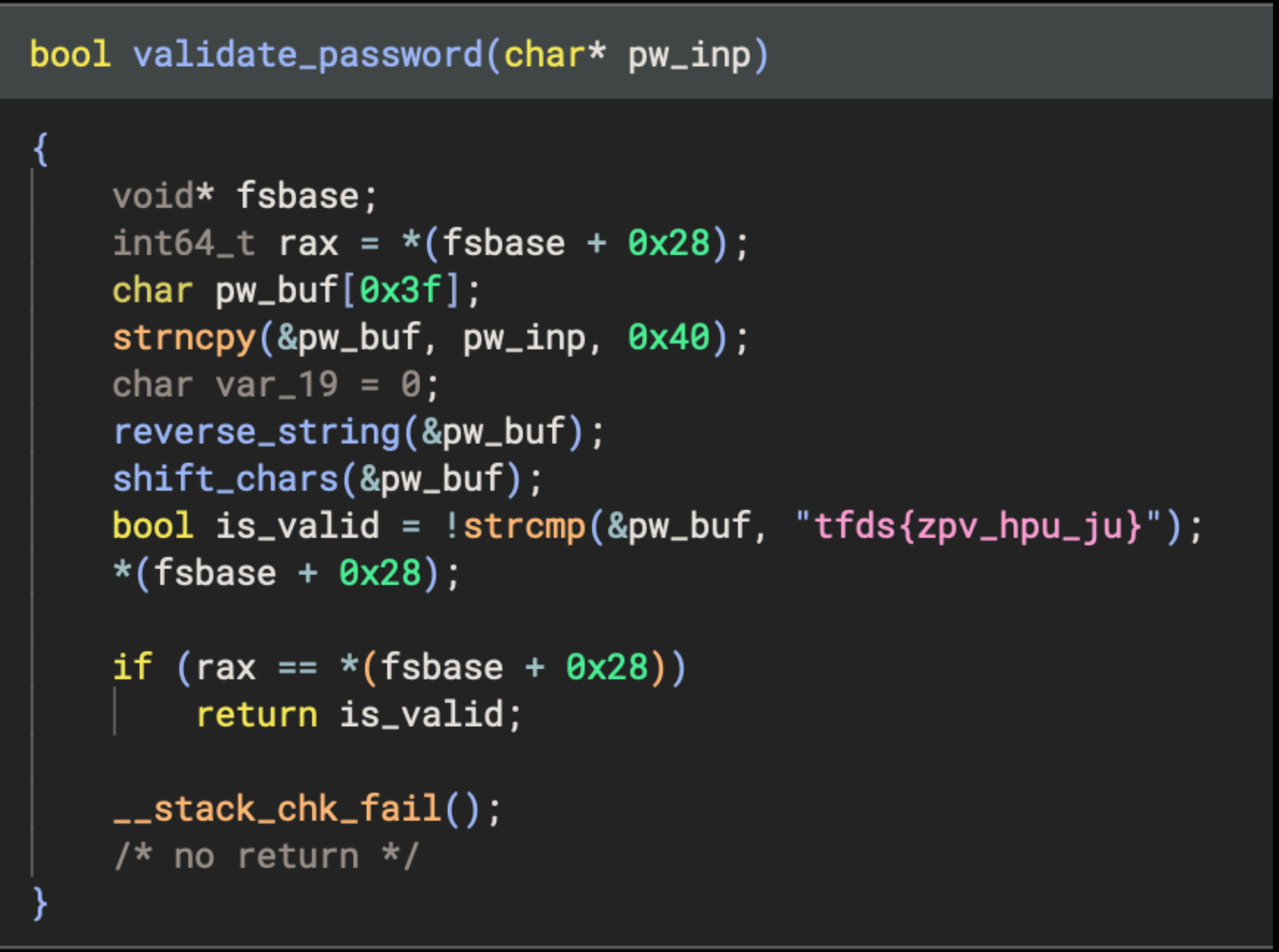
```
bool validate_password(char* pw_inp)

{
    void* fsbase;
    int64_t rax = *(fsbase + 0x28);
    char pw_buf[0x3f];
    strncpy(&pw_buf, pw_inp, 0x40);
    char var_19 = 0;
    reverse_string(&pw_buf);
    shift_chars(&pw_buf);
    bool is_valid = !strcmp(&pw_buf, "tfds{zpv_hpu_ju}");
    *(fsbase + 0x28);

    if (rax == *(fsbase + 0x28))
        return is_valid;

    __stack_chk_fail();
    /* no return */
}
```



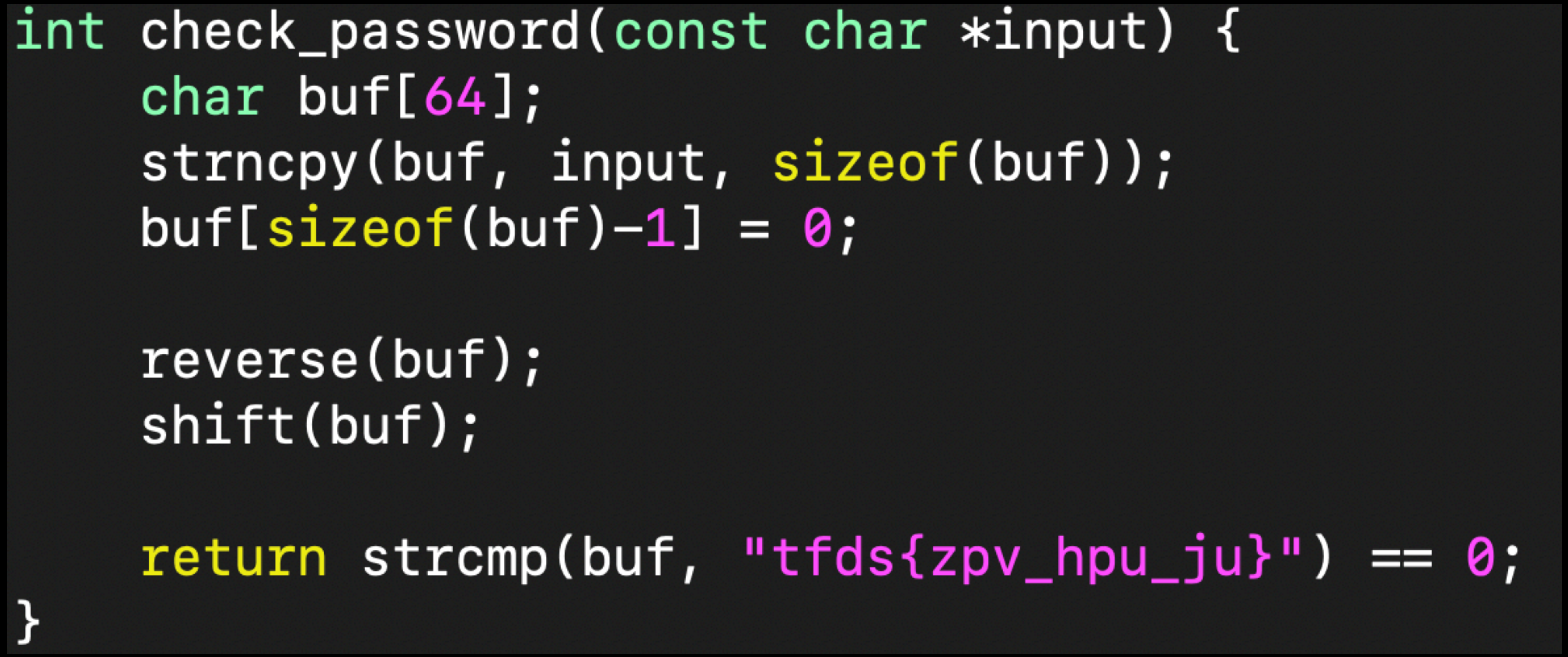
```
00401364    uint64_t sub_401364(char* arg1)

00401364    {
00401364        void* fsbase;
00401374        int64_t rax = *(uint64_t*)((char*)fsbase + 0x28);
00401396        char var_58[0x3f];
00401396        strncpy(&var_58, arg1, 0x40);
0040139b        char var_19 = 0;
004013a6        sub_401249(&var_58);
004013b2        sub_4012db(&var_58);
004013cf        int32_t rax_1;
004013cf        (uint8_t)rax_1 = !strcmp(&var_58, "tfds{zpv_hpu_ju}");
004013d9        *(uint64_t*)((char*)fsbase + 0x28);
004013d9
004013e2        if (rax == *(uint64_t*)((char*)fsbase + 0x28))
004013ea            return (uint64_t)(uint8_t)rax_1;
004013ea
004013e4        __stack_chk_fail();
004013e4        /* no return */
00401364    }
```



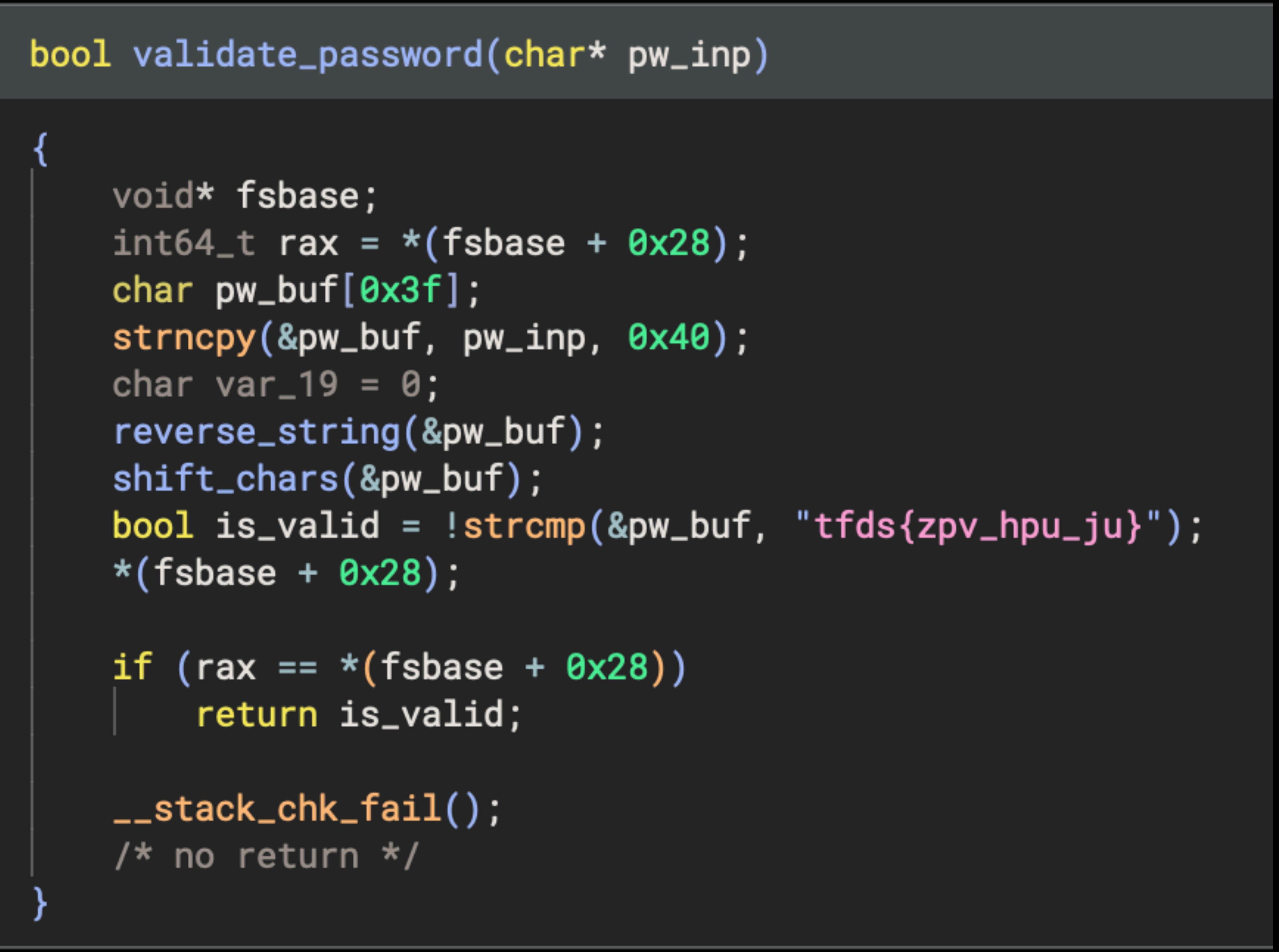
```
bool validate_password(char* pw_inp)

{
    void* fsbase;
    int64_t rax = *(fsbase + 0x28);
    char pw_buf[0x3f];
    strncpy(&pw_buf, pw_inp, 0x40);
    char var_19 = 0;
    reverse_string(&pw_buf);
    shift_chars(&pw_buf);
    bool is_valid = !strcmp(&pw_buf, "tfds{zpv_hpu_ju}");
    *(fsbase + 0x28);

    if (rax == *(fsbase + 0x28))
        return is_valid;

    __stack_chk_fail();
    /* no return */
}
```



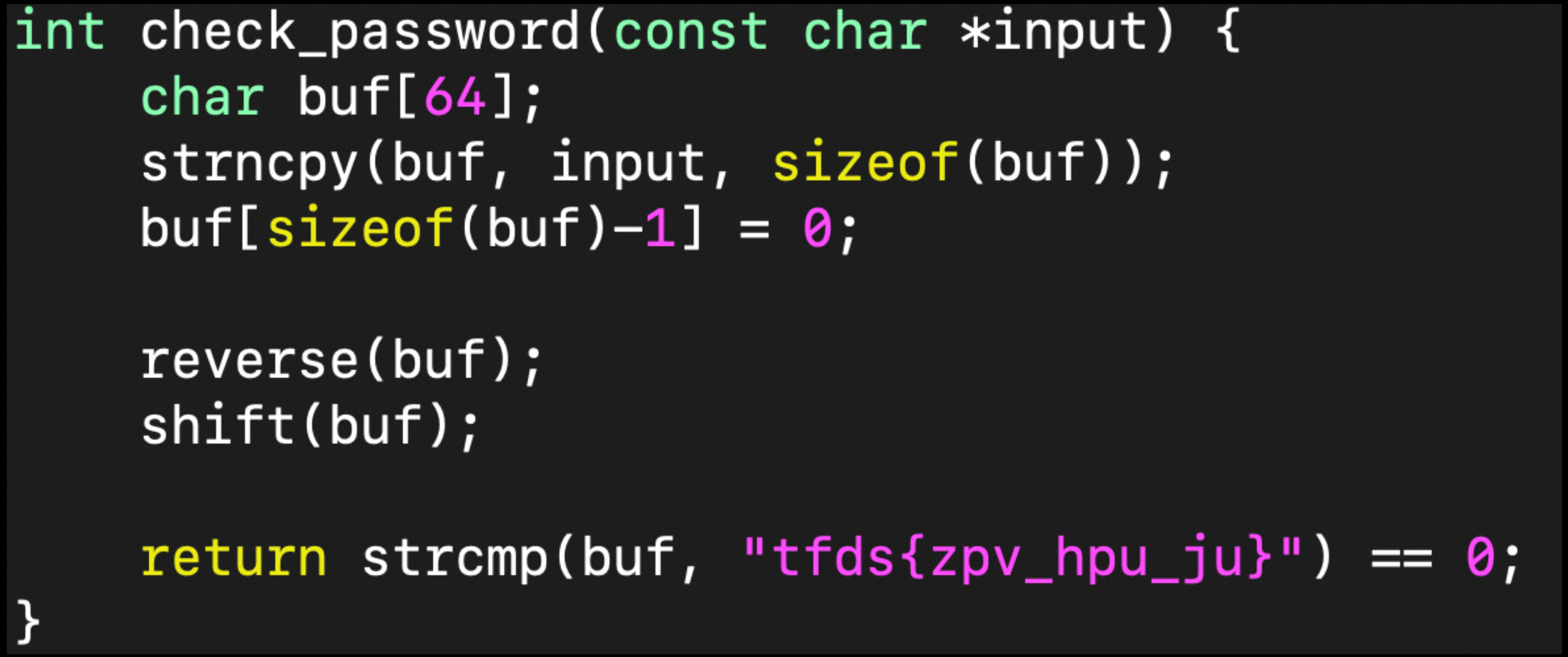
```
bool validate_password(char* pw_inp)

{
    void* fsbase;
    int64_t rax = *(fsbase + 0x28);
    char pw_buf[0x3f];
    strncpy(&pw_buf, pw_inp, 0x40);
    char var_19 = 0;
    reverse_string(&pw_buf);
    shift_chars(&pw_buf);
    bool is_valid = !strcmp(&pw_buf, "tfds{zpv_hpu_ju}");
    *(fsbase + 0x28);

    if (rax == *(fsbase + 0x28))
        return is_valid;

    __stack_chk_fail();
    /* no return */
}
```



```
int check_password(const char *input) {
    char buf[64];
    strncpy(buf, input, sizeof(buf));
    buf[sizeof(buf)-1] = 0;

    reverse(buf);
    shift(buf);

    return strcmp(buf, "tfds{zpv_hpu_ju}") == 0;
}
```



```
bool validate_password(char* pw_inp)

{
    void* fsbase;
    int64_t rax = *(fsbase + 0x28);
    char pw_buf[0x3f];
    strncpy(&pw_buf, pw_inp, 0x40);
    char var_19 = 0;
    reverse_string(&pw_buf);
    shift_chars(&pw_buf);
    bool is_valid = !strcmp(&pw_buf, "tfds{zpv_hpu_ju}");
    *(fsbase + 0x28);

    if (rax == *(fsbase + 0x28))
        return is_valid;

    __stack_chk_fail();
    /* no return */
}
```



```
int check_password(const char *input) {
    char buf[64];
    strncpy(buf, input, sizeof(buf));
    buf[sizeof(buf)-1] = 0;

    reverse(buf);
    shift(buf);

    return strcmp(buf, "tfds{zpv_hpu_ju}") == 0;
}
```

Decompilation is not 1:1

Memory...

... can be used as a stack

- From before: `mov     DWORD PTR [rbp-0x4], edi`

Memory...

... can be used as a stack

- From before: `mov     DWORD PTR [rbp-0x4], edi`
- What's going on here?

Memory...

... can be used as a stack

- From before: `mov      DWORD PTR [rbp-0x4], edi`
- What's going on here?
- This is putting an item (`edi`) into some location in the stack (`rbp-0x4`)

Memory...

... can be used as a stack

- From before: `mov      DWORD PTR [rbp-0x4], edi`
- What's going on here?
- This is putting an item (`edi`) into some location in the stack (`rbp-0x4`)
- What is a stack?

Memory...

... can be used as a stack

- From before: `mov     DWORD PTR [rbp-0x4], edi`
- What's going on here?
- This is putting an item (`edi`) into some location in the stack (`rbp-0x4`)
- What is a stack?
- What is `rbp`?

Memory...

... can be used as a stack

- From before: `mov     DWORD PTR [rbp-0x4], edi`
- What's going on here?
- This is putting an item (`edi`) into some location in the stack (`rbp-0x4`)
- What is a stack?
- What is `rbp`?
- Why haven't I mentioned `rsp`?

Memory...

... can be used as a stack

- The "stack" is a section of memory used during execution

Memory...

... can be used as a stack

- The "stack" is a section of memory used during execution
- Every function allocates it's own stack space on entry

Memory...

... can be used as a stack

- The "stack" is a section of memory used during execution
- Every function allocates it's own stack space on entry
    - and thus returns it back to it's previous state on exit

Memory...

... can be used as a stack

- The "stack" is a section of memory used during execution
- Every function allocates it's own stack space on entry
    - and thus returns it back to it's previous state on exit
- Allows for push/pop from the "top" of the stack (much like the ADT)

Memory...

... can be used as a stack

```
push 0x41
push 0x42
push 0x43
pop eax
push 0x44
push 0x45
pop ebx
pop ecx
push 0x46
push 0x47
push 0x48
pop edx
pop esi
pop edi
pop ebp
```

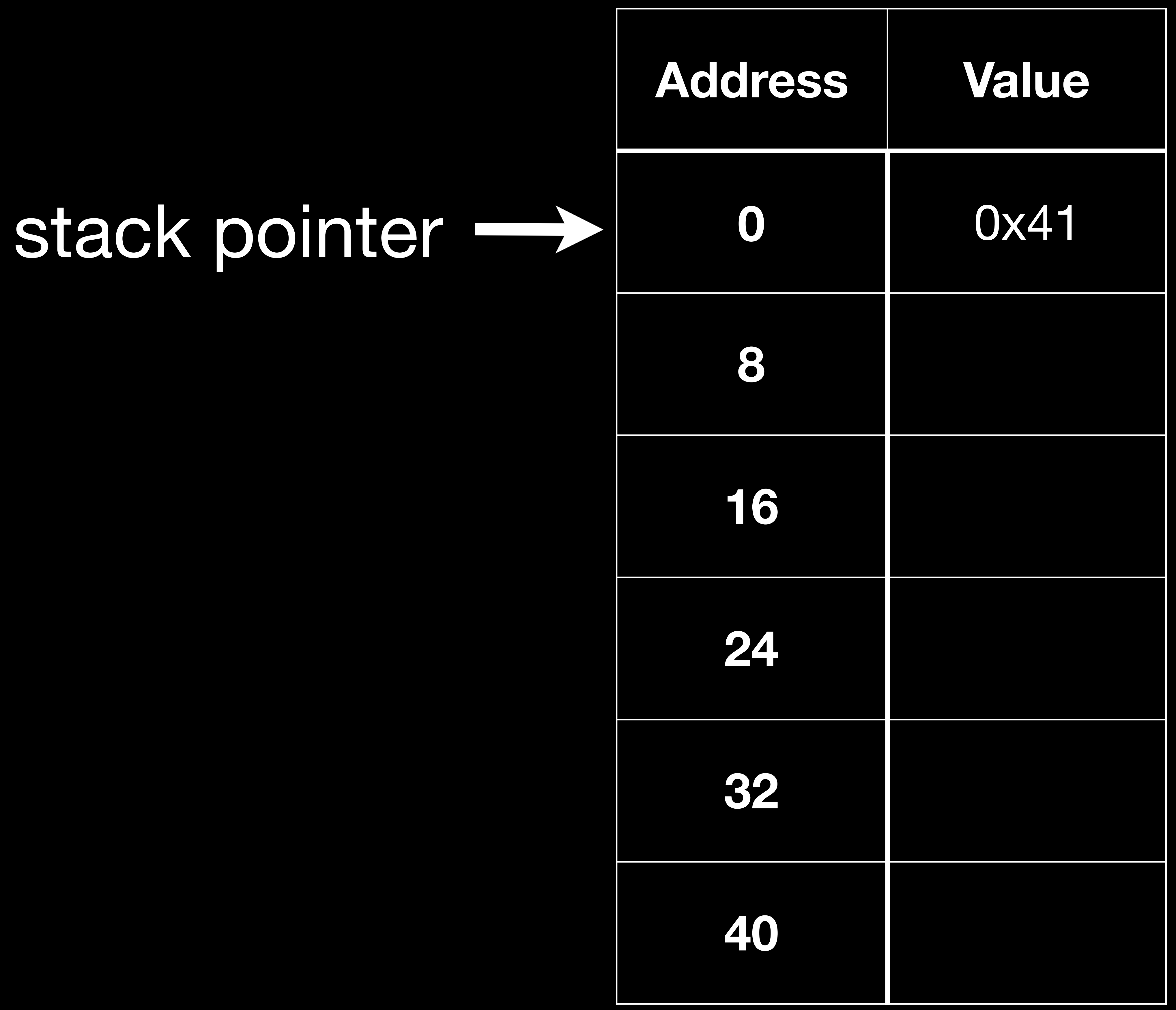
stack pointer → 0

Address	Value
0	
8	
16	
24	
32	
40	

Memory...

... can be used as a stack

```
push 0x41
push 0x42
push 0x43
pop eax
push 0x44
push 0x45
pop ebx
pop ecx
push 0x46
push 0x47
push 0x48
pop edx
pop esi
pop edi
pop ebp
```



stack pointer →

Address	Value
0	0x41
8	
16	
24	
32	
40	

Memory...

... can be used as a stack

```
push 0x41
push 0x42
push 0x43
pop eax
push 0x44
push 0x45
pop ebx
pop ecx
push 0x46
push 0x47
push 0x48
pop edx
pop esi
pop edi
pop ebp
```

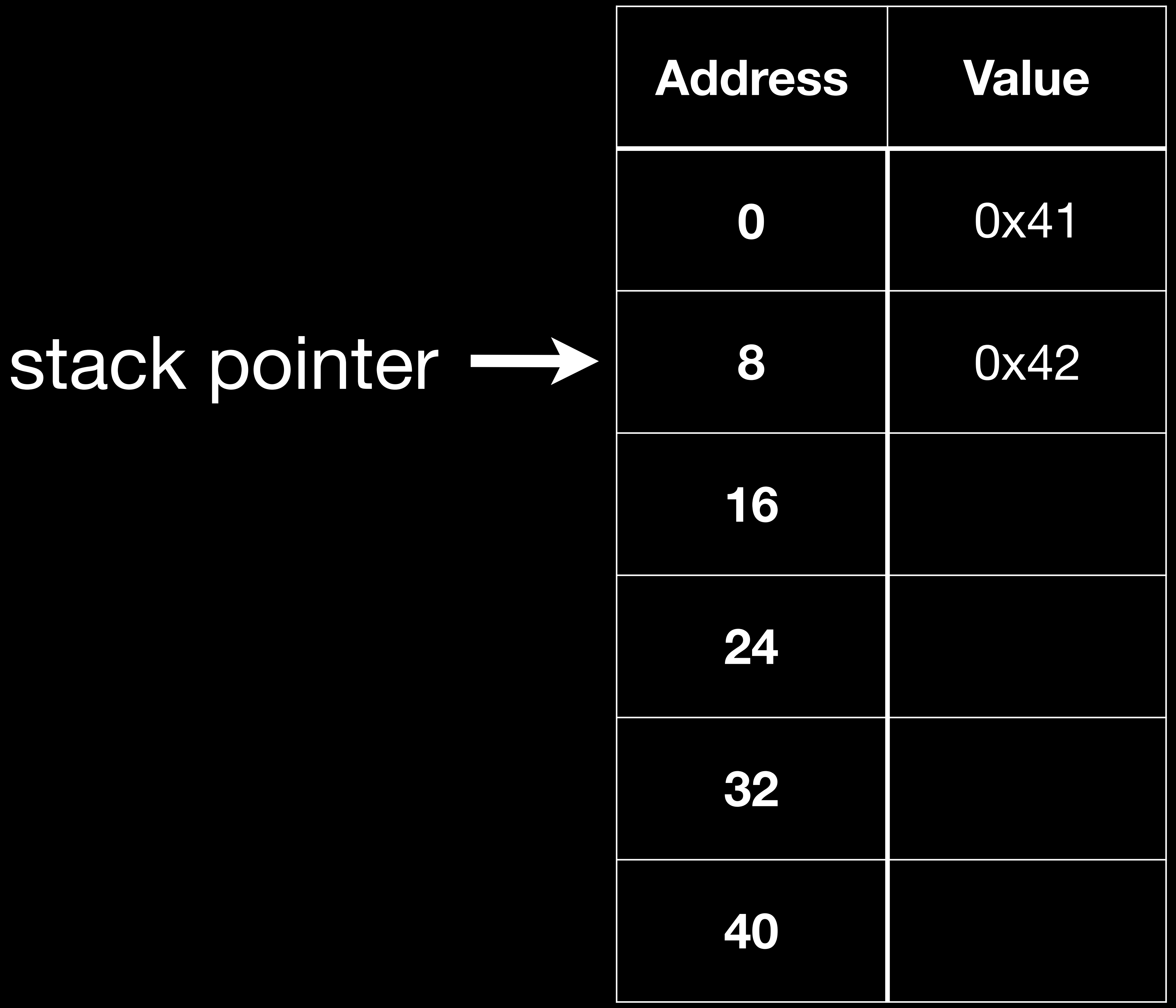
stack pointer →

Address	Value
0	0x41
8	
16	
24	
32	
40	

Memory...

... can be used as a stack

```
push 0x41
push 0x42
push 0x43
pop eax
push 0x44
push 0x45
pop ebx
pop ecx
push 0x46
push 0x47
push 0x48
pop edx
pop esi
pop edi
pop ebp
```



stack pointer →

Address	Value
0	0x41
8	0x42
16	
24	
32	
40	

Memory...

... can be used as a stack

```
push 0x41
push 0x42
push 0x43
pop eax
push 0x44
push 0x45
pop ebx
pop ecx
push 0x46
push 0x47
push 0x48
pop edx
pop esi
pop edi
pop ebp
```

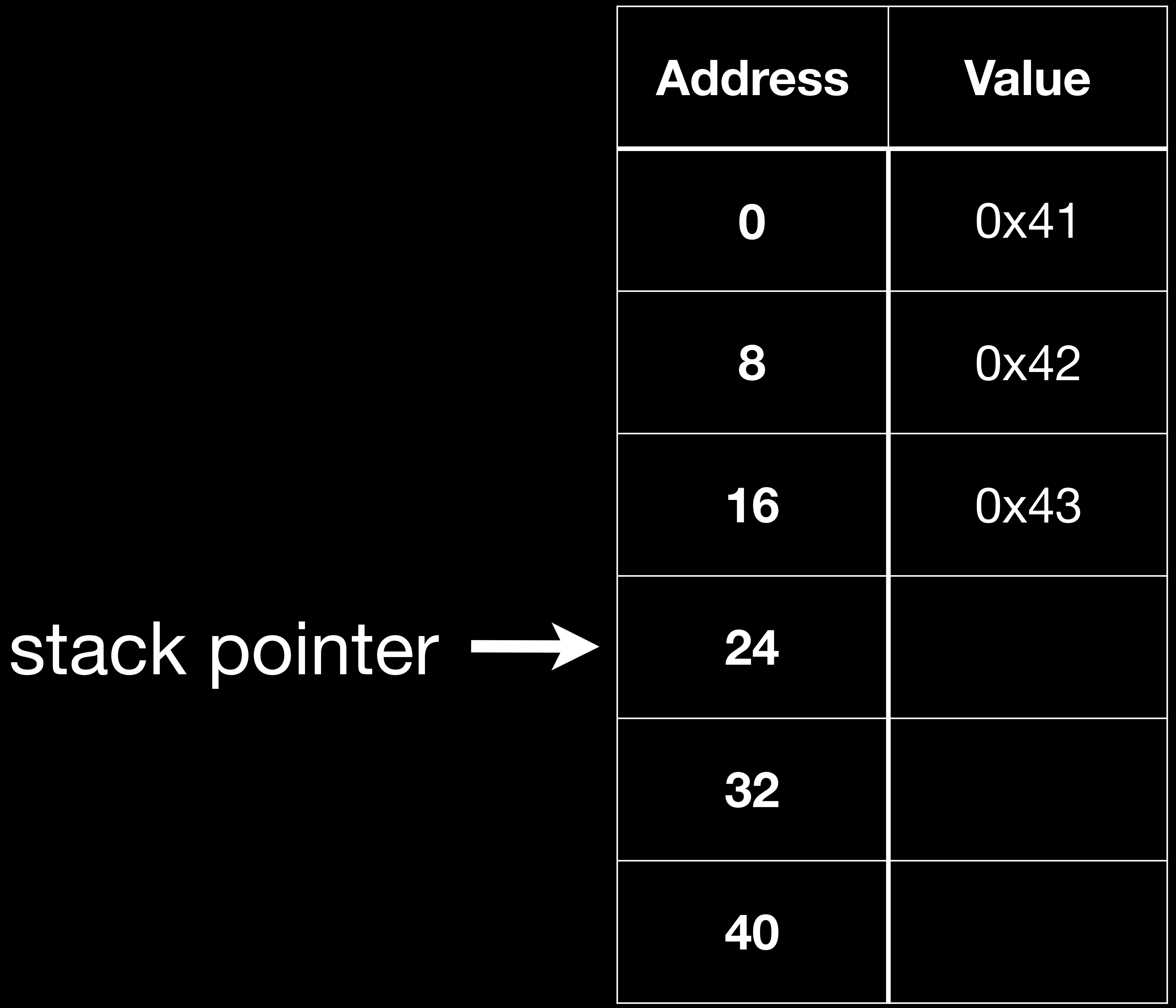
stack pointer → 16

Address	Value
0	0x41
8	0x42
16	
24	
32	
40	

Memory...

... can be used as a stack

```
push 0x41
push 0x42
push 0x43
pop eax
push 0x44
push 0x45
pop ebx
pop ecx
push 0x46
push 0x47
push 0x48
pop edx
pop esi
pop edi
pop ebp
```



stack pointer → 24

Address	Value
0	0x41
8	0x42
16	0x43
24	
32	
40	

Memory...

... can be used as a stack

```
push 0x41
push 0x42
push 0x43
pop eax
push 0x44
push 0x45
pop ebx
pop ecx
push 0x46
push 0x47
push 0x48
pop edx
pop esi
pop edi
pop ebp
```

```
eax: 0x43
```

stack pointer → 16

Address	Value
0	0x41
8	0x42
16	0x43
24	
32	
40	

Memory...

... can be used as a stack

```
push 0x41
push 0x42
push 0x43
pop eax
push 0x44
push 0x45
pop ebx
pop ecx
push 0x46
push 0x47
push 0x48
pop edx
pop esi
pop edi
pop ebp
```

```
eax: 0x43
```

Address	Value
0	0x41
8	0x42
16	0x44
24	
32	
40	

stack pointer → 24

Memory...

... can be used as a stack

```
push 0x41
push 0x42
push 0x43
pop eax
push 0x44
push 0x45
pop ebx
pop ecx
push 0x46
push 0x47
push 0x48
pop edx
pop esi
pop edi
pop ebp
```

```
eax: 0x43
```

Address	Value
0	0x41
8	0x42
16	0x44
24	0x45
32	
40	

stack pointer → 32

Memory...

... can be used as a stack

```
push 0x41
push 0x42
push 0x43
pop eax
push 0x44
push 0x45
pop ebx
pop ecx
push 0x46
push 0x47
push 0x48
pop edx
pop esi
pop edi
pop ebp
```

```
eax: 0x43
ebx: 0x45
```

	Address	Value
	0	0x41
	8	0x42
	16	0x44
stack pointer →	24	0x45
	32	
	40	

Memory...

... can be used as a stack

```
push 0x41
push 0x42
push 0x43
pop eax
push 0x44
push 0x45
pop ebx
pop ecx
push 0x46
push 0x47
push 0x48
pop edx
pop esi
pop edi
pop ebp
```

```
eax: 0x43
ebx: 0x45
ecx: 0x44
```

stack pointer → 16

Address	Value
0	0x41
8	0x42
16	0x44
24	0x45
32	
40	

Memory...

... can be used as a stack

```
push 0x41
push 0x42
push 0x43
pop eax
push 0x44
push 0x45
pop ebx
pop ecx
push 0x46
push 0x47
push 0x48
pop edx
pop esi
pop edi
pop ebp
```

```
eax: 0x43
ebx: 0x45
ecx: 0x44
```

stack pointer → 24

Address	Value
0	0x41
8	0x42
16	0x46
24	0x45
32	
40	

Memory...

... can be used as a stack

```
push 0x41
push 0x42
push 0x43
pop eax
push 0x44
push 0x45
pop ebx
pop ecx
push 0x46
push 0x47
push 0x48
pop edx
pop esi
pop edi
pop ebp
```

```
eax: 0x43
ebx: 0x45
ecx: 0x44
```

Address	Value
0	0x41
8	0x42
16	0x46
24	0x47
32	
40	

stack pointer → 32

Memory...

... can be used as a stack

```
push 0x41
push 0x42
push 0x43
pop eax
push 0x44
push 0x45
pop ebx
pop ecx
push 0x46
push 0x47
push 0x48
pop edx
pop esi
pop edi
pop ebp
```

```
eax: 0x43
ebx: 0x45
ecx: 0x44
```

Address	Value
0	0x41
8	0x42
16	0x46
24	0x47
32	0x48
40	

stack pointer → 40

Memory...

... can be used as a stack

```
push 0x41
push 0x42
push 0x43
pop eax
push 0x44
push 0x45
pop ebx
pop ecx
push 0x46
push 0x47
push 0x48
pop edx
pop esi
pop edi
pop ebp
```

```
eax: 0x43
ebx: 0x45
ecx: 0x44
edx: 0x48
```

stack pointer → 32

Address	Value
0	0x41
8	0x42
16	0x46
24	0x47
32	0x48
40	

Memory...

... can be used as a stack

```
push 0x41
push 0x42
push 0x43
pop eax
push 0x44
push 0x45
pop ebx
pop ecx
push 0x46
push 0x47
push 0x48
pop edx
pop esi
pop edi
pop ebp
```

```
eax: 0x43
ebx: 0x45
ecx: 0x44
edx: 0x48
esi: 0x47
```

Address	Value
0	0x41
8	0x42
16	0x46
24	0x47
32	0x48
40	

stack pointer → 24

Memory...

... can be used as a stack

```
push 0x41
push 0x42
push 0x43
pop eax
push 0x44
push 0x45
pop ebx
pop ecx
push 0x46
push 0x47
push 0x48
pop edx
pop esi
pop edi
pop ebp
```

```
eax: 0x43
ebx: 0x45
ecx: 0x44
edx: 0x48
esi: 0x47
edi: 0x48
```

stack pointer → 16

Address	Value
0	0x41
8	0x42
16	0x46
24	0x47
32	0x48
40	

Memory...

... can be used as a stack

```
push 0x41
push 0x42
push 0x43
pop eax
push 0x44
push 0x45
pop ebx
pop ecx
push 0x46
push 0x47
push 0x48
pop edx
pop esi
pop edi
pop ebp
```

```
eax: 0x43
ebx: 0x45
ecx: 0x44
edx: 0x48
esi: 0x47
edi: 0x48
ebp: 0x42
```

stack pointer

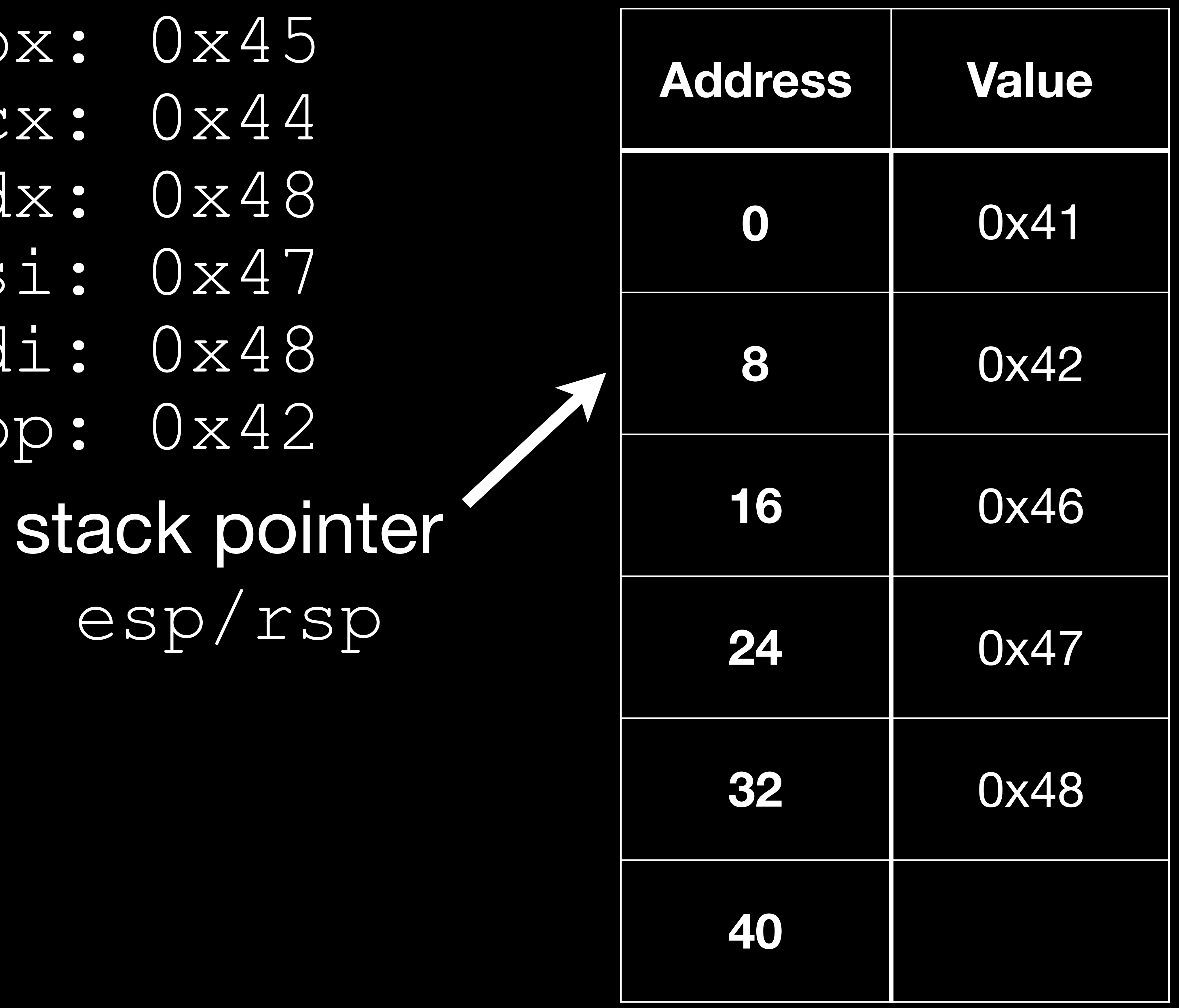
Address	Value
0	0x41
8	0x42
16	0x46
24	0x47
32	0x48
40	

Memory...

... can be used as a stack

```
push 0x41
push 0x42
push 0x43
pop eax
push 0x44
push 0x45
pop ebx
pop ecx
push 0x46
push 0x47
push 0x48
pop edx
pop esi
pop edi
pop ebp
```

```
eax: 0x43
ebx: 0x45
ecx: 0x44
edx: 0x48
esi: 0x47
edi: 0x48
ebp: 0x42
```



stack pointer
esp/rsp

Address	Value
0	0x41
8	0x42
16	0x46
24	0x47
32	0x48
40	

Memory...

...needs to be function local

- *"Every function allocates it's own stack space on entry"*

Memory...

...needs to be function local

- "*Every function allocates it's own stack space on entry*"
- How is this accomplished?

Memory...

...needs to be function local

- "*Every function allocates it's own stack space on entry*"
- How is this accomplished?
- Start of function (prologue) sets up a "frame"

prologue

```
push    rbp
mov     rbp, rsp
sub     rsp, N
```

Memory...

...needs to be function local

- "*Every function allocates it's own stack space on entry*"
- How is this accomplished?
- Start of function (prologue) sets up a "frame"
- End of function (epilogue) goes back to the previous (caller's) frame

prologue

```
push   rbp
mov    rbp,  rsp
sub    rsp,  N
```

epilogue

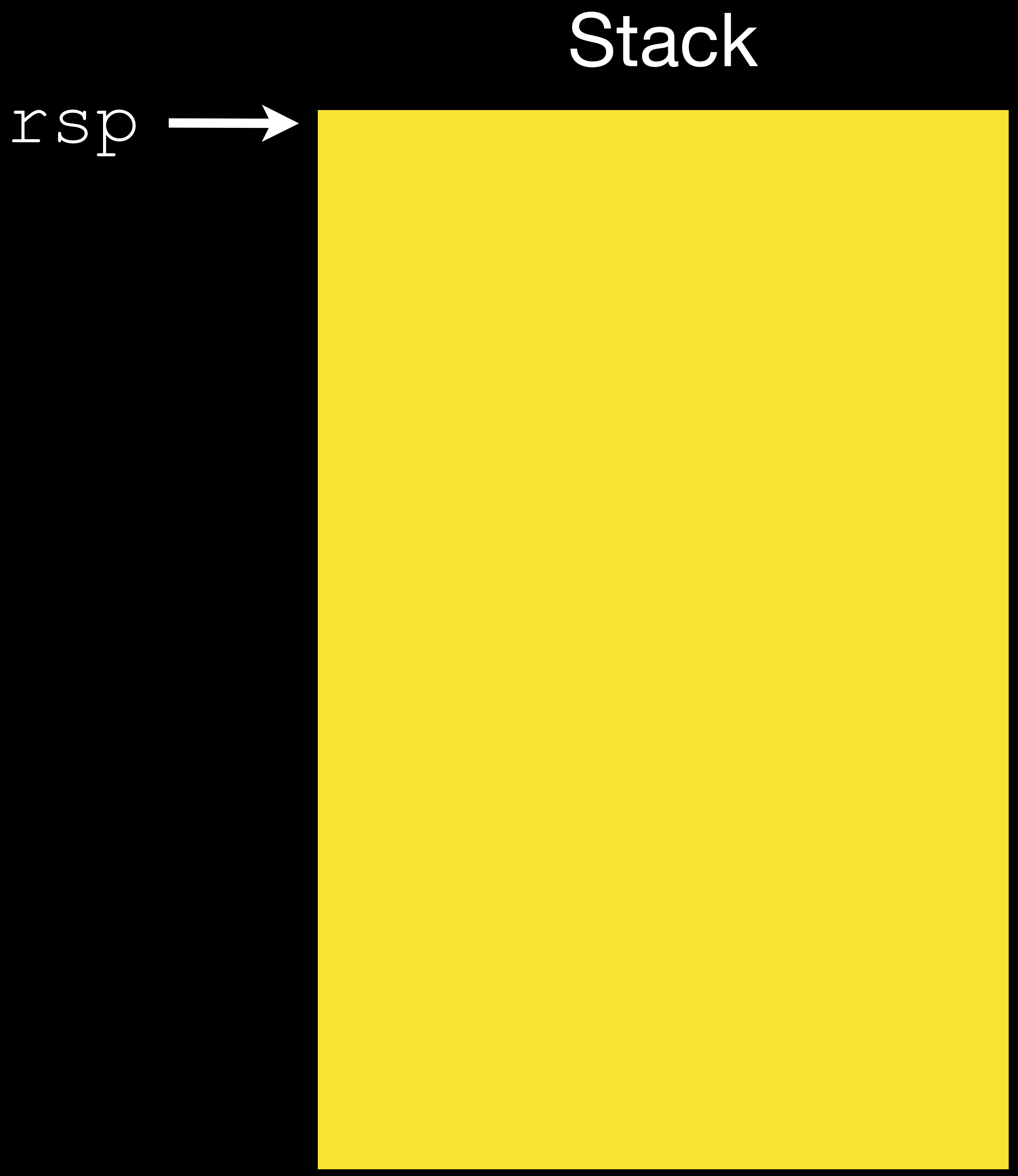
```
mov    rsp,  rbp
pop    rbp
ret
```

Memory...

...needs to be function local

prologue

```
push    rbp
mov     rbp, rsp
sub     rsp, N
```

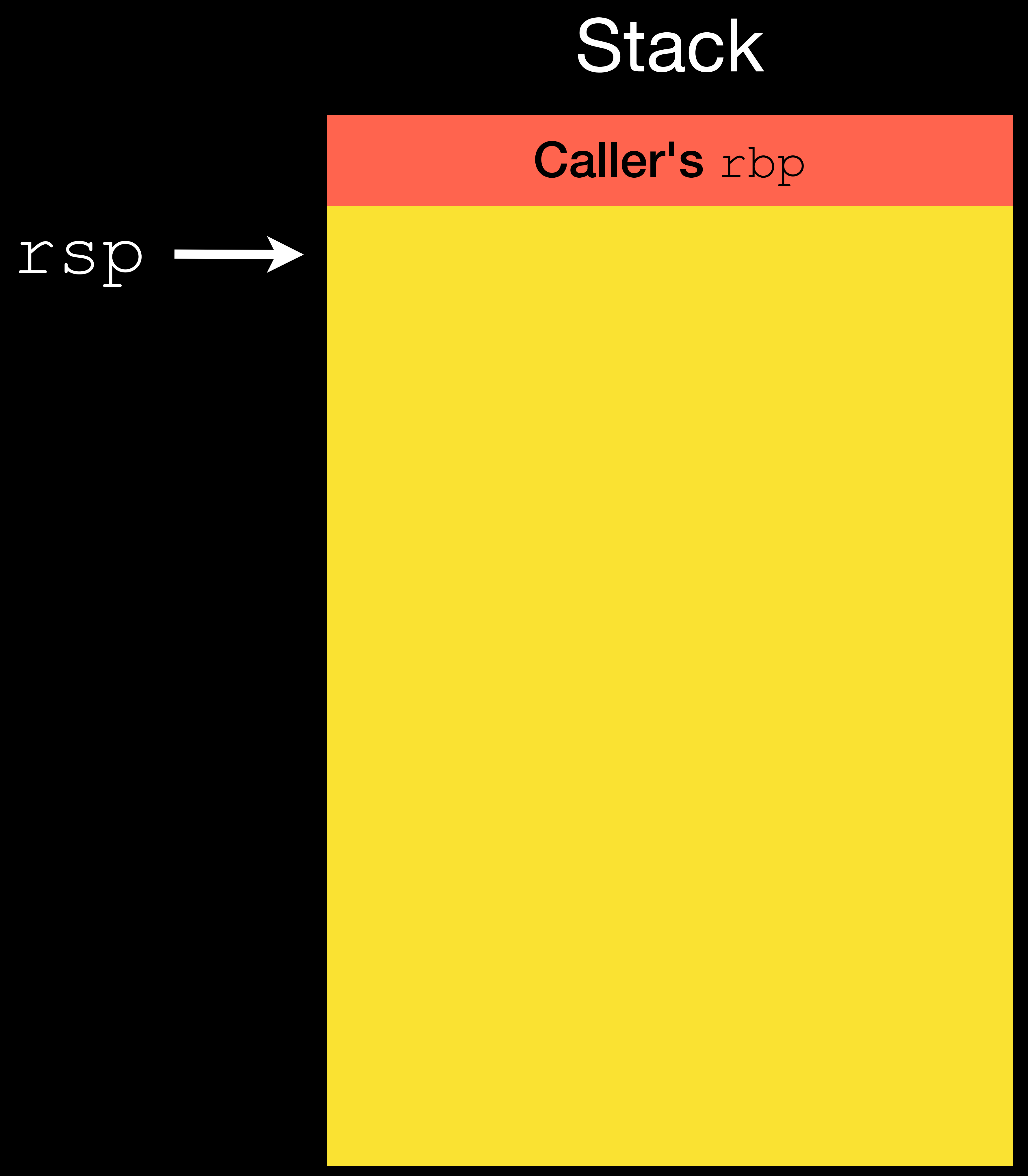


Memory...

...needs to be function local

prologue

```
push    rbp
mov     rbp, rsp
sub     rsp, N
```

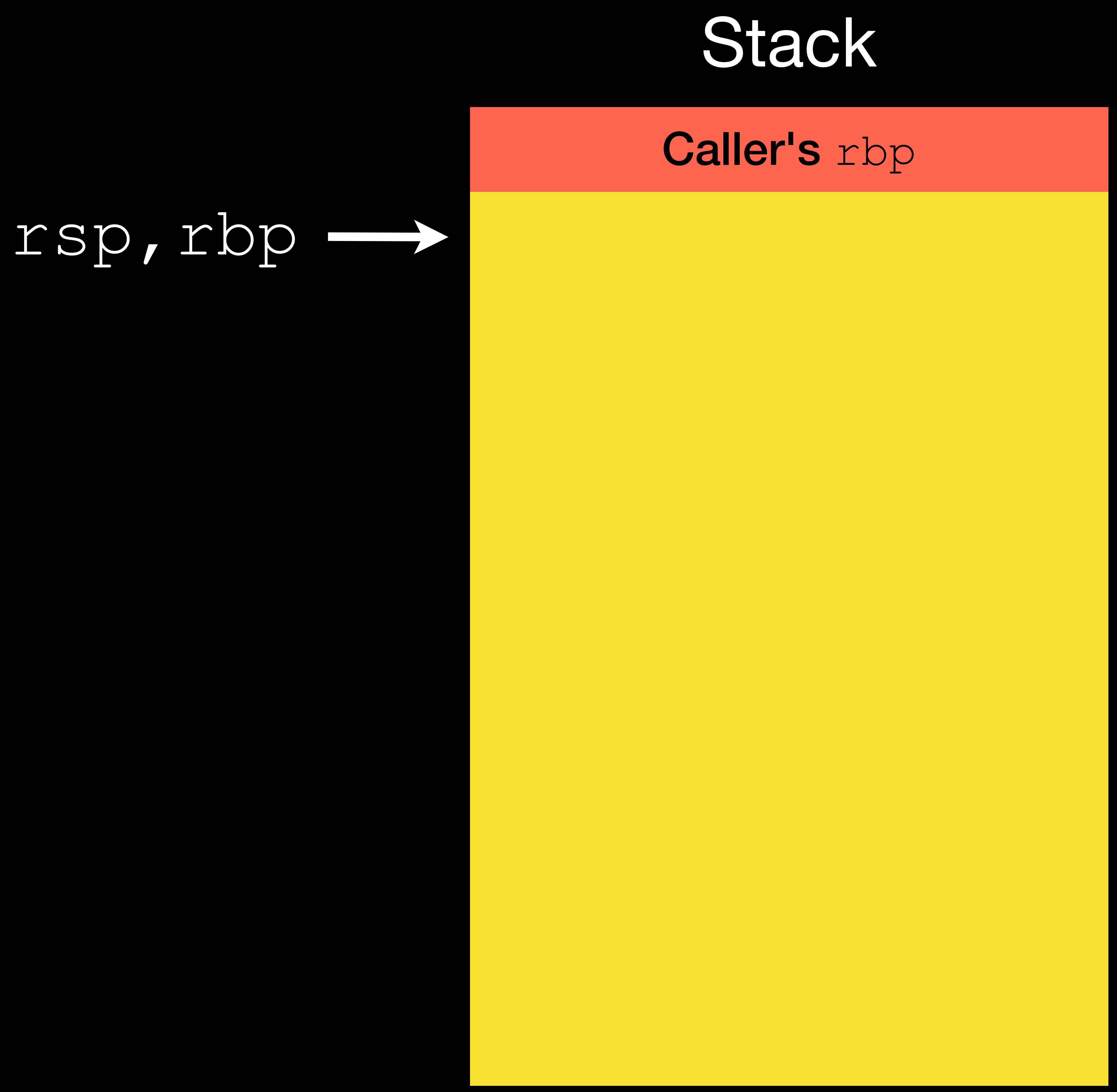


Memory...

...needs to be function local

prologue

```
push    rbp
mov     rbp,  rsp
sub     rsp,  N
```

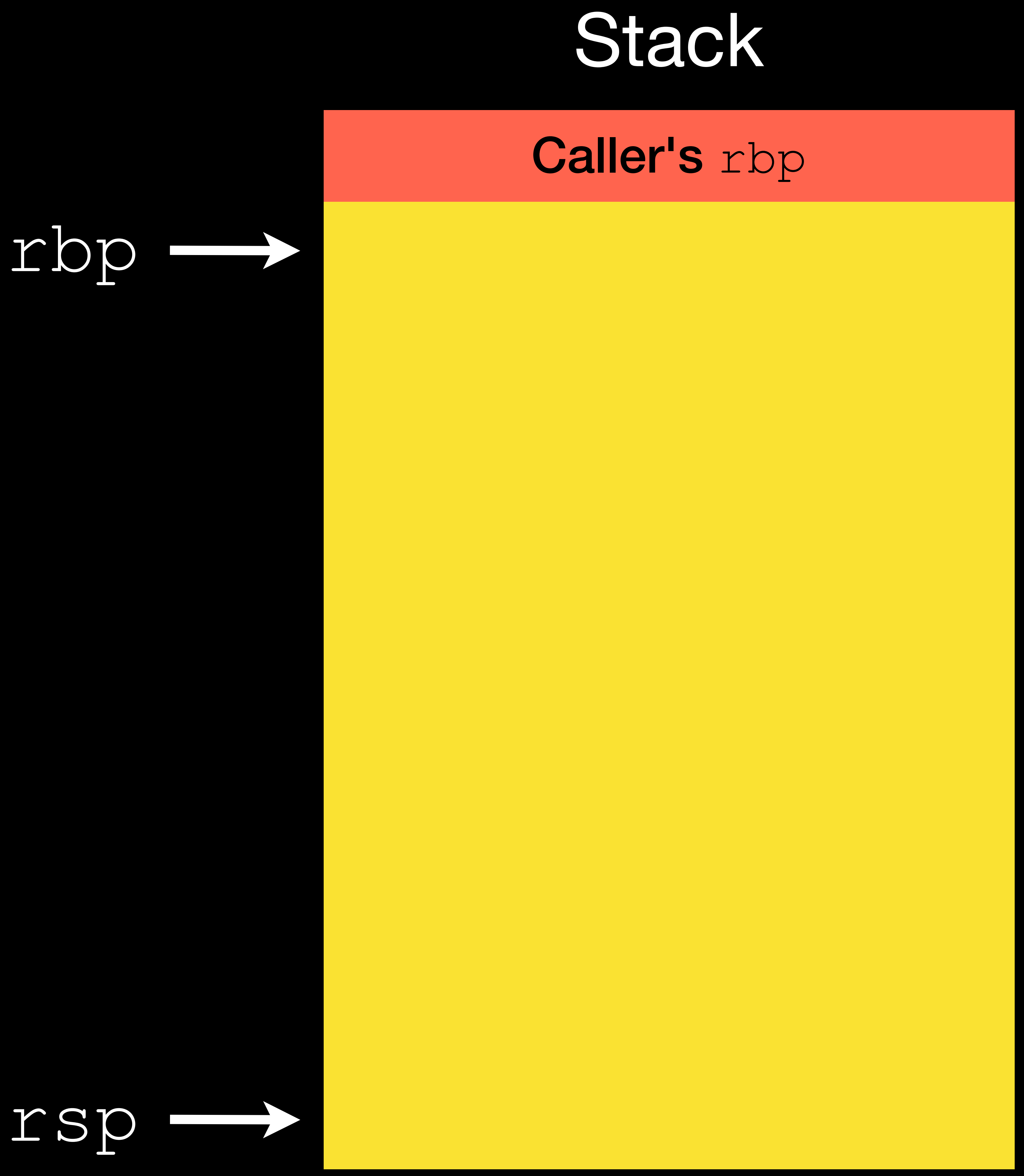


Memory...

...needs to be function local

prologue

```
push    rbp
mov     rbp, rsp
sub     rsp, N
```

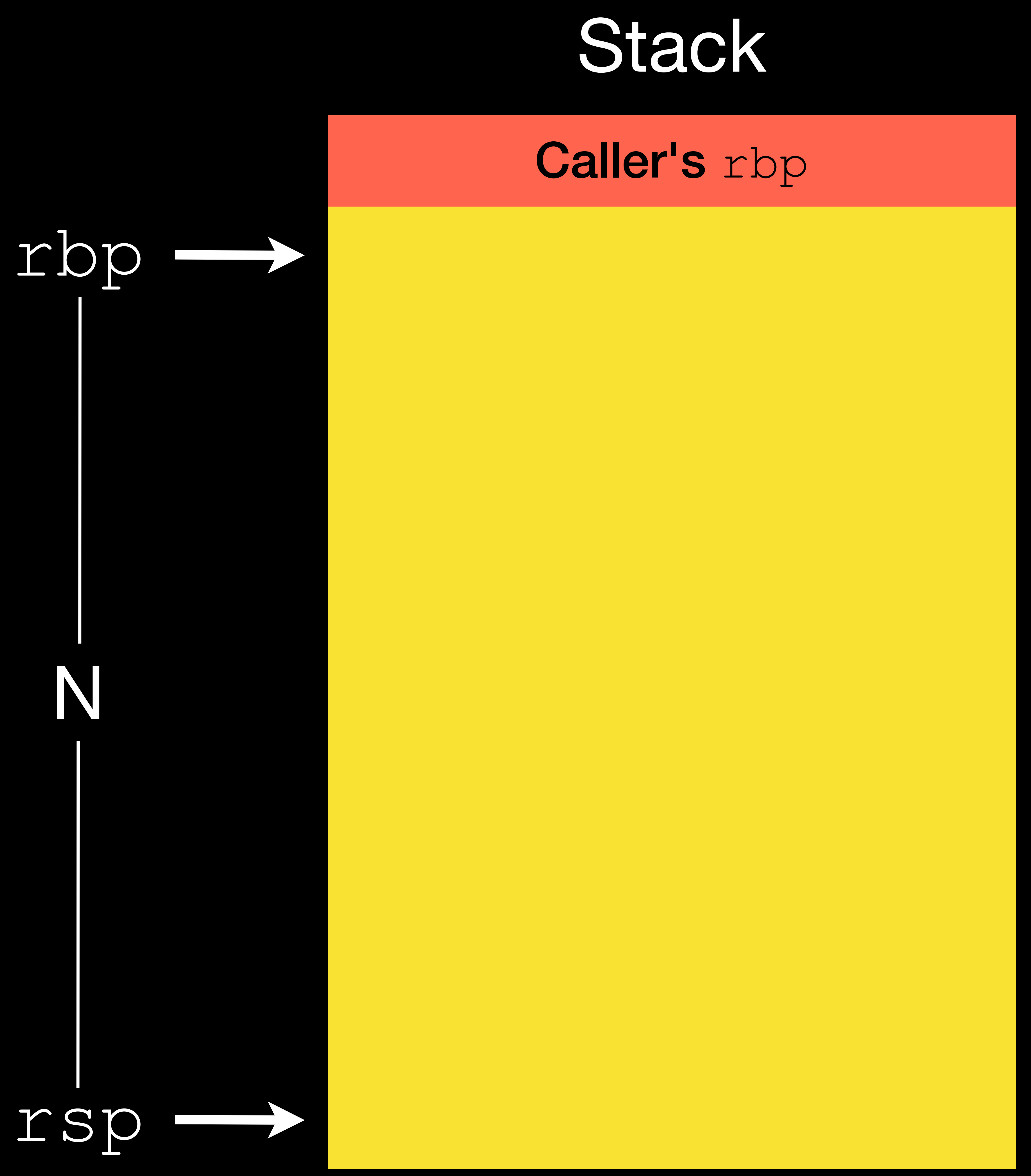


Memory...

...needs to be function local

prologue

```
push    rbp
mov     rbp, rsp
sub     rsp, N
```

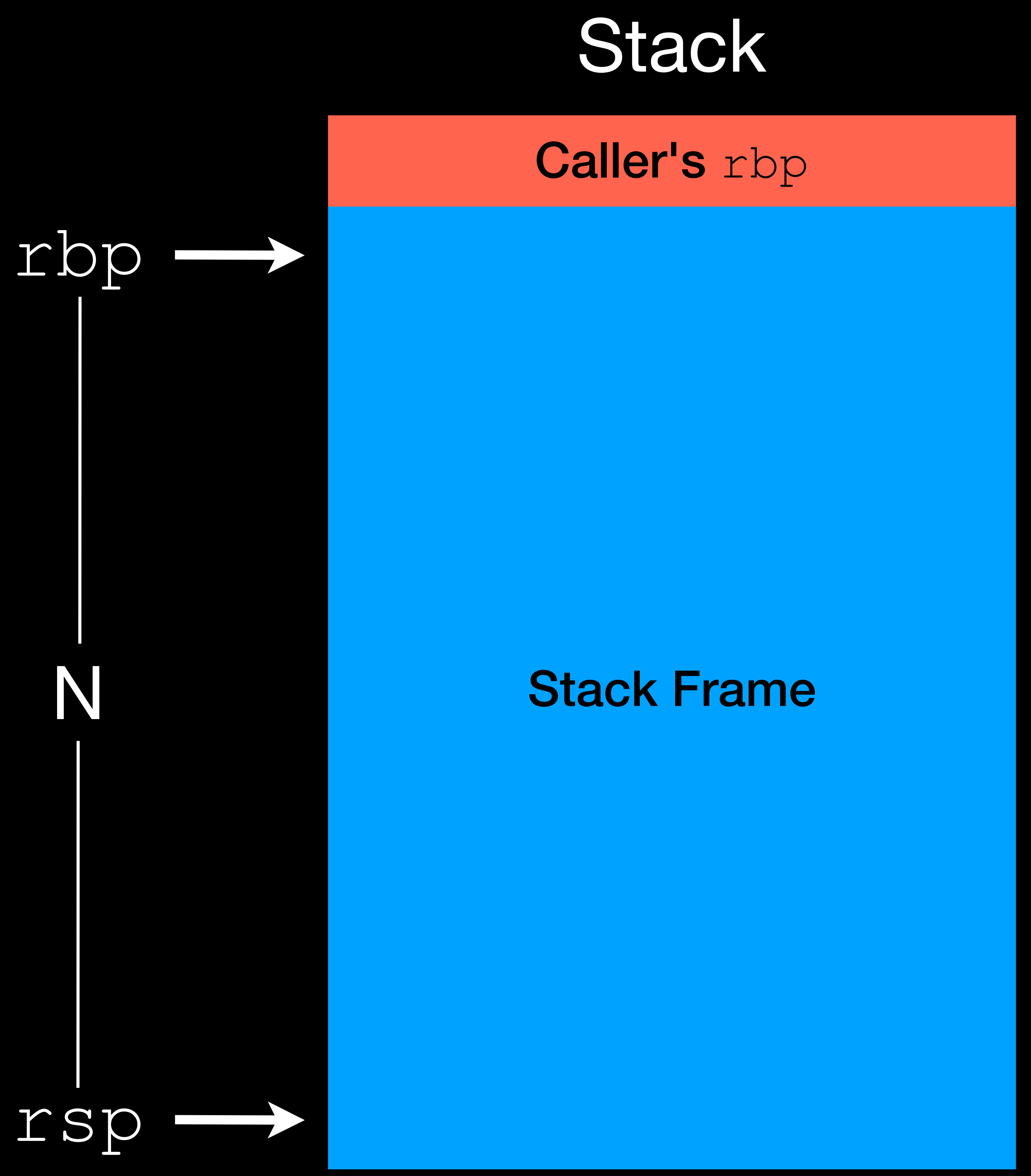


Memory...

...needs to be function local

prologue

```
push    rbp
mov     rbp, rsp
sub     rsp, N
```



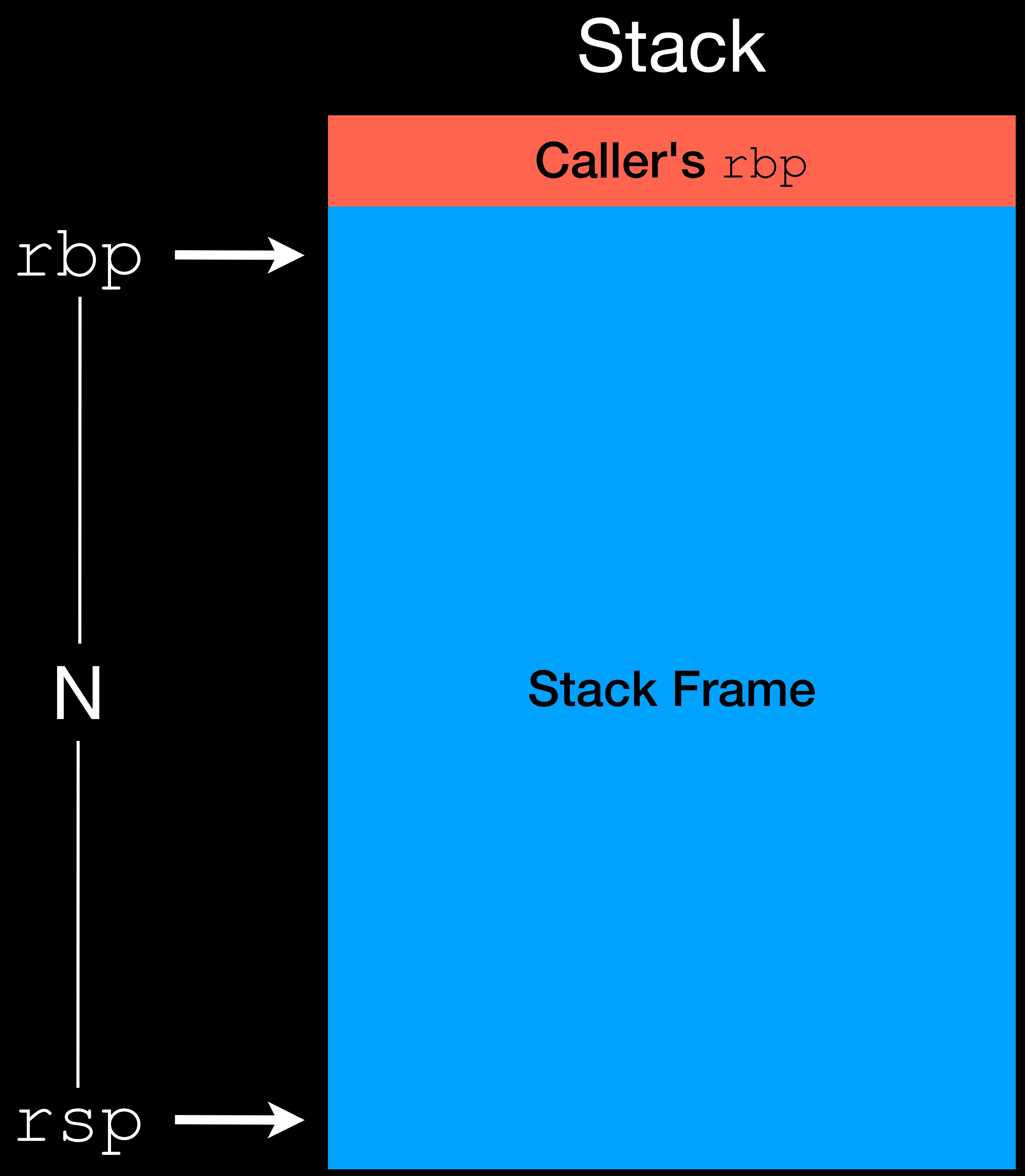
Memory...

...needs to be function local

prologue

```
push    rbp
mov     rbp, rsp
sub     rsp, N
```

```
mov     DWORD PTR [rbp-0x4], edi
```



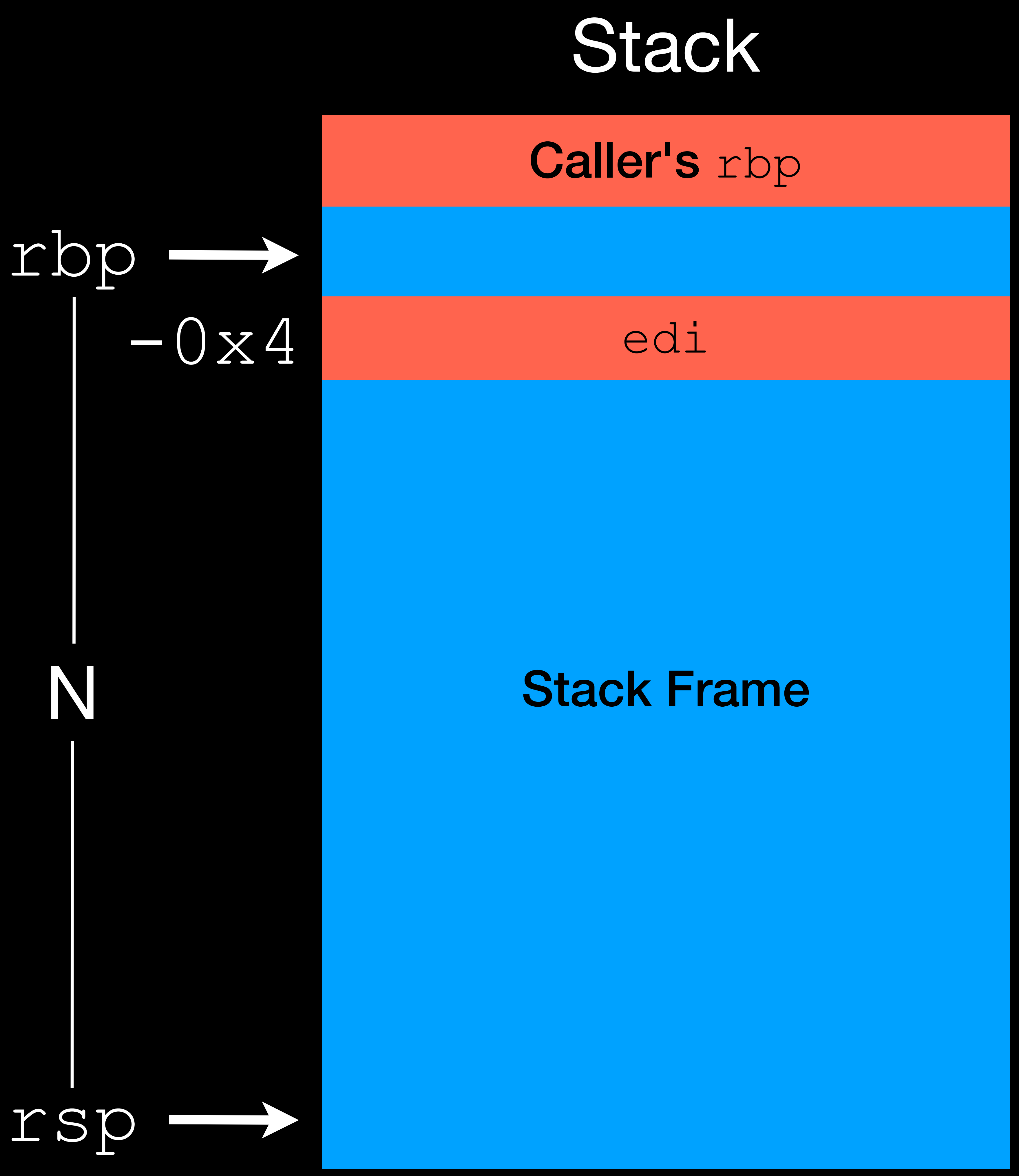
Memory...

...needs to be function local

prologue

```
    push    rbp
    mov     rbp, rsp
    sub     rsp, N
```

```
mov     DWORD PTR [rbp-0x4], edi
```

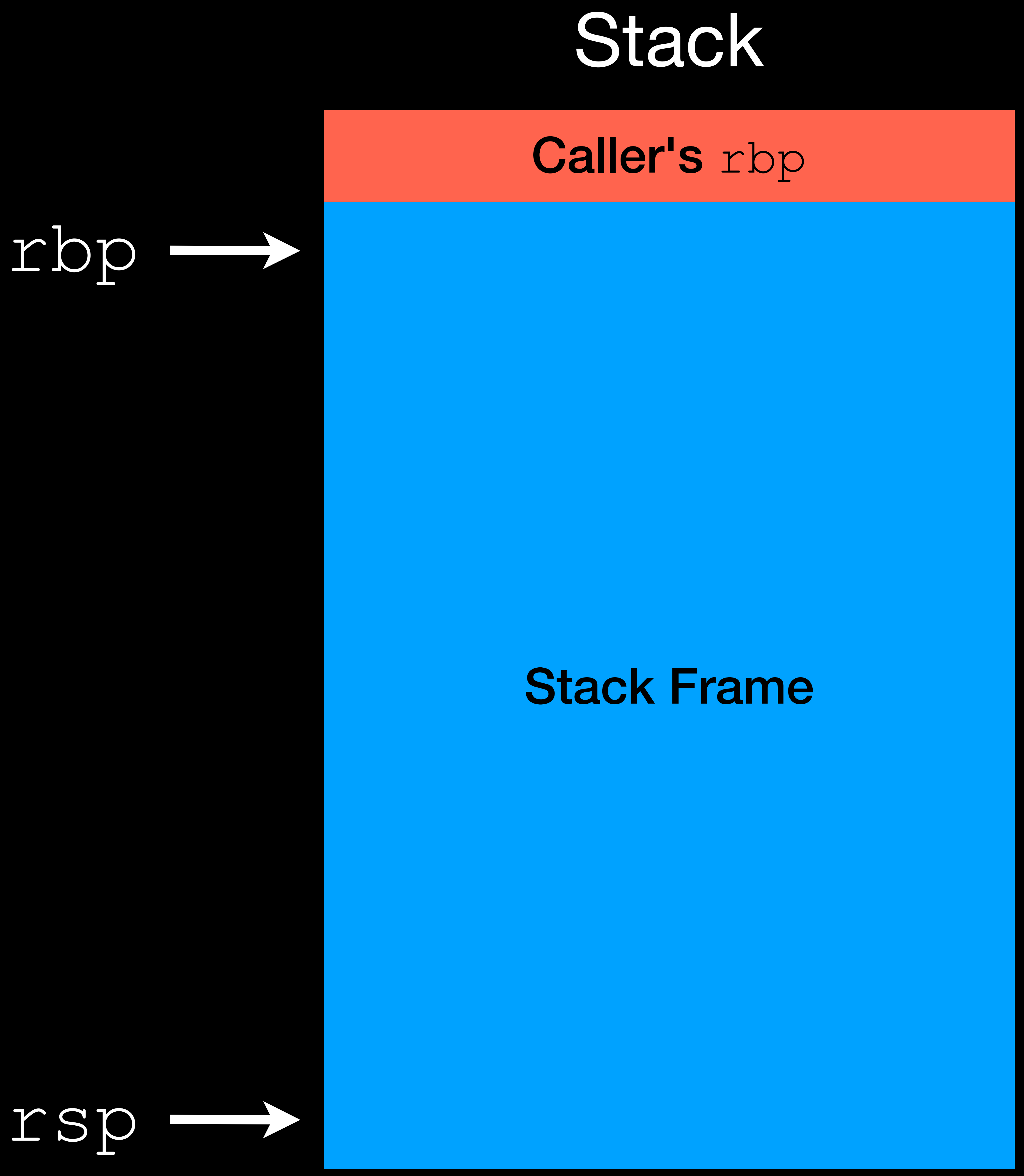


Memory...

...needs to be function local

epilogue

```
mov     rsp, rbp
pop     rbp
ret
```

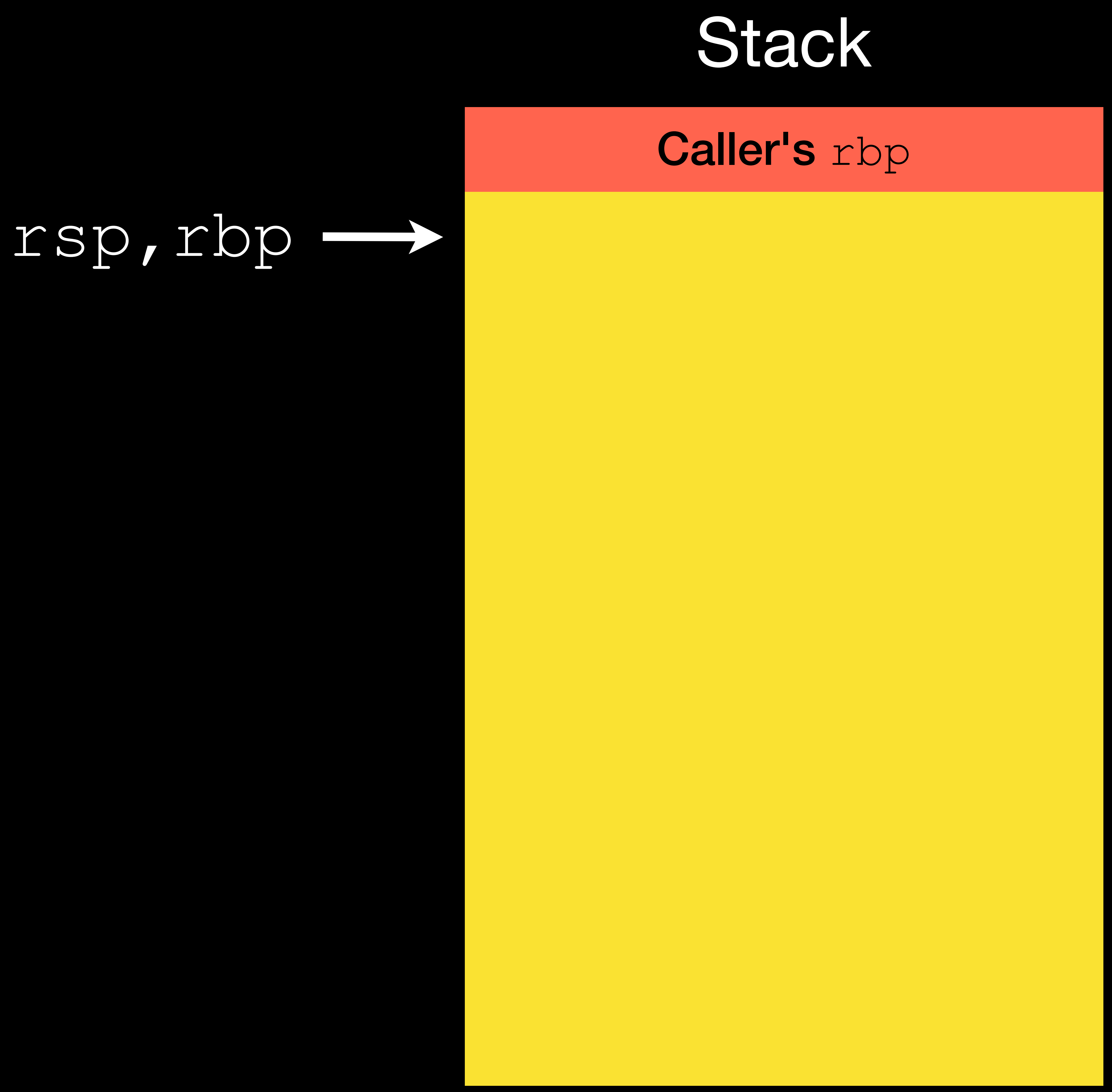


Memory...

...needs to be function local

epilogue

```
mov     rsp, rbp
pop     rbp
ret
```

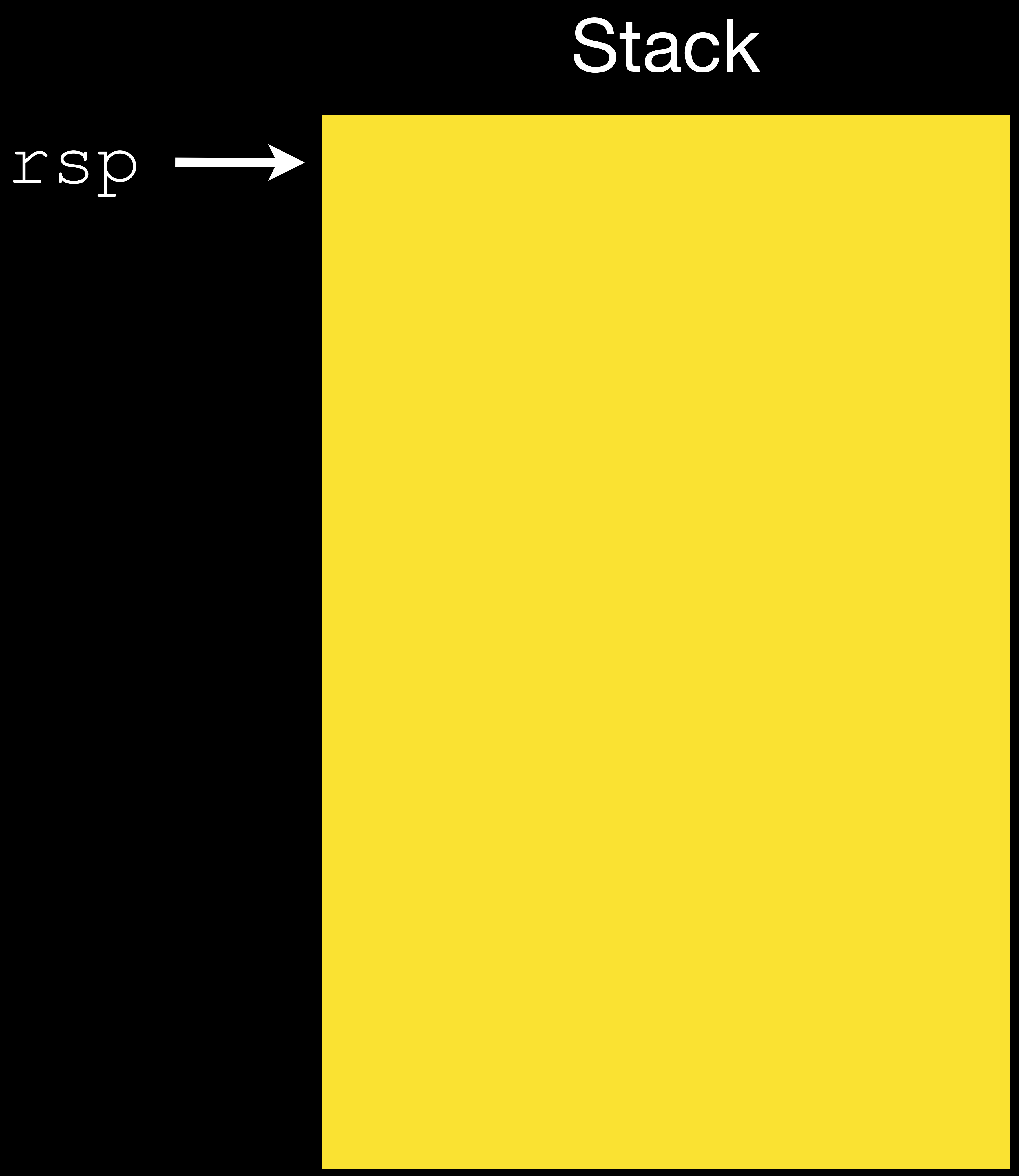


Memory...

...needs to be function local

epilogue

```
mov     rsp, rbp
pop     rbp
ret
```



Stack

rsp →

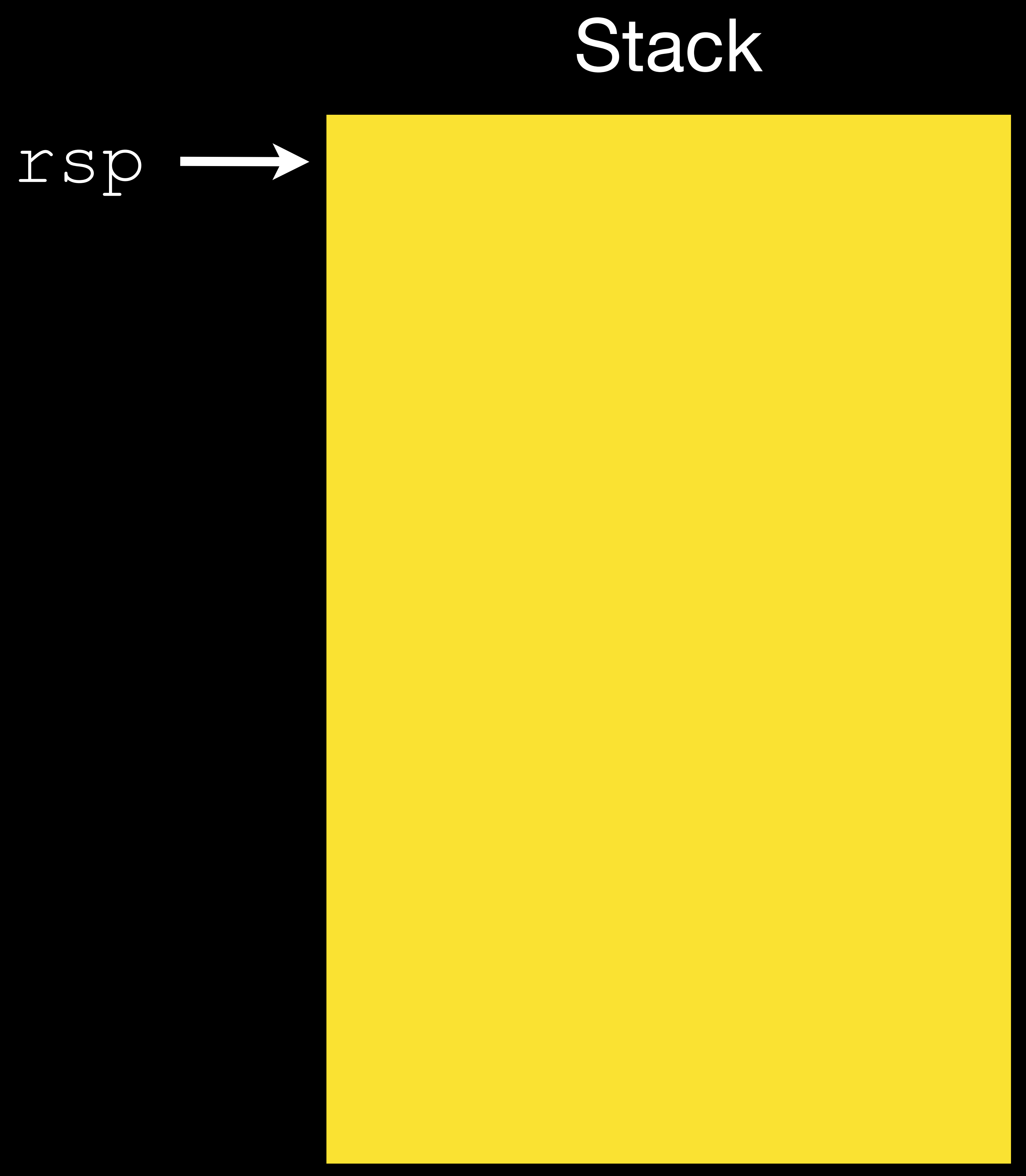
Memory...

...needs to be function local

epilogue

```
mov     rsp, rbp
pop     rbp
ret
```

return to caller



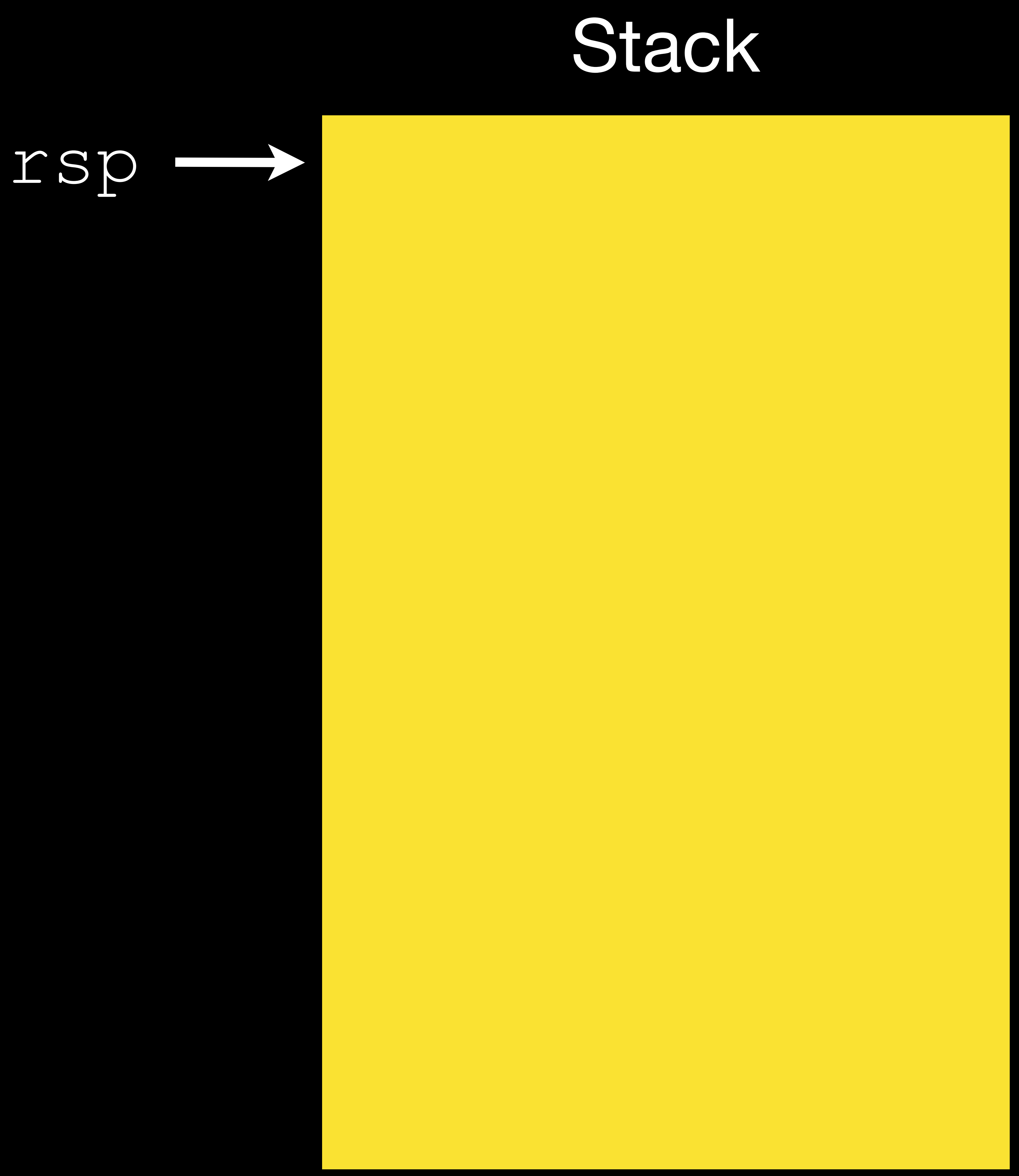
Memory...

...needs to be function local

epilogue

```
mov     rsp, rbp
pop     rbp
ret
```

After a function call returns, the stack of the previous function remains unchanged



Memory...

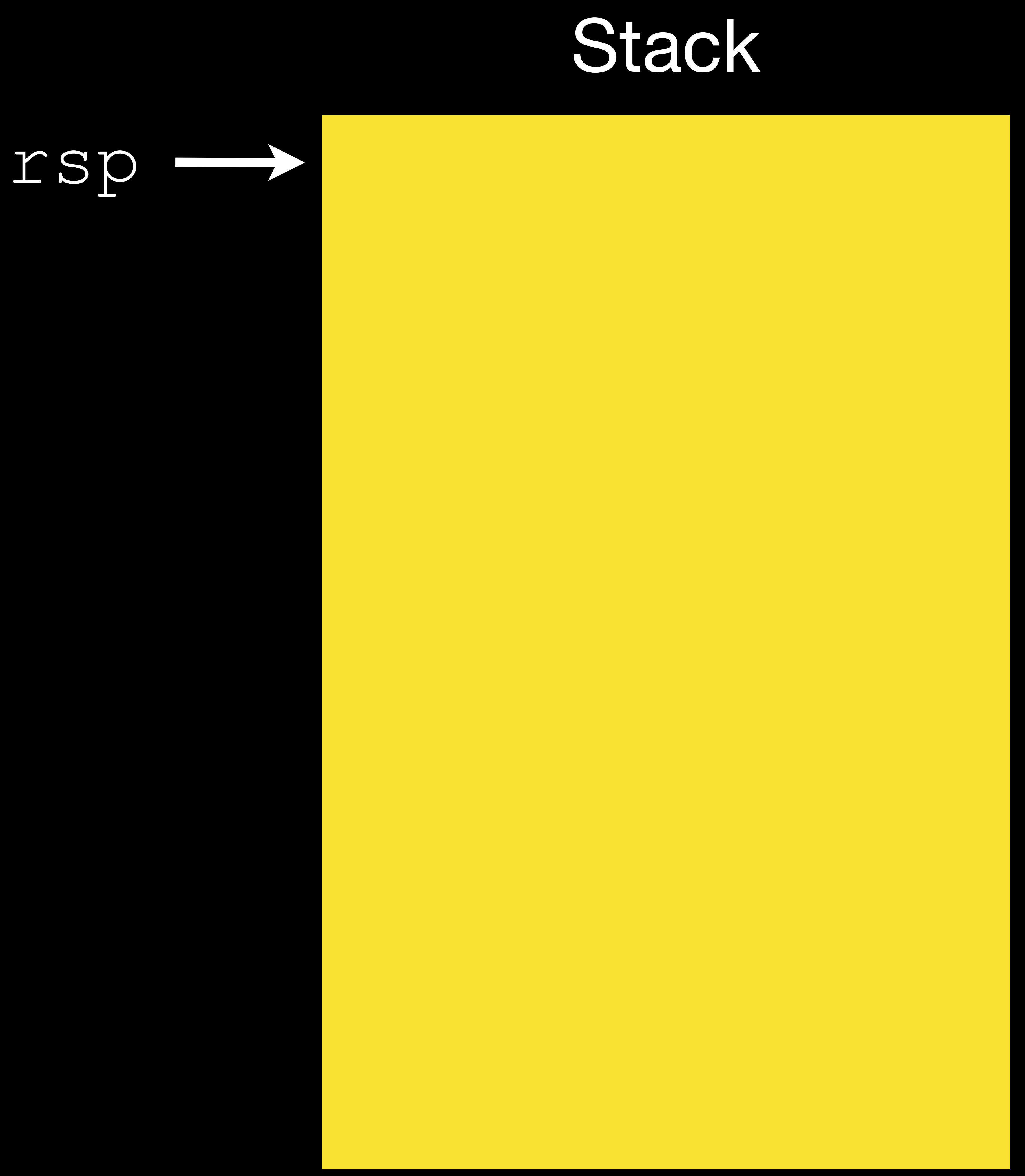
...needs to be function local

epilogue

```
mov     rsp, rbp
pop     rbp
ret
```

After a function call returns, the stack of the previous function remains unchanged

rsp/rbp should remain exactly as they were before and after the function call



Decompilation...

... can fail

- Certain tricks can be used to trip up a decompiler

Decompilation…

… can fail

- Certain tricks can be used to trip up a decompiler
- Some decompilers will fail if the previous invariant cannot be statically proven

Decompilation...

... can fail

- Certain tricks can be used to trip up a decompiler
- Some decompilers will fail if the previous invariant cannot be statically proven
    - *rsp/rbp should remain exactly as they were before and after the function call*

Decompilation…

… can fail

- Certain tricks can be used to trip up a decompiler
- Some decompilers will fail if the previous invariant cannot be statically proven
    - *rsp/rbp should remain exactly as they were before and after the function call*
- Can we prevent this invariant from being proven?

Decompilation...

... can fail

```
...prologue

       call   MyFunc
       cmp    rax, 0
       jz     CONTINUE
       add    rsp, 4
CONTINUE:
       pop    rax

...rest of function
```

Decompilation...

... can fail

```
...prologue

        call    MyFunc
        cmp     rax, 0
        jz      CONTINUE
        add     rsp, 4
CONTINUE:
        pop     rax

...rest of function
```

- Let's say that `MyFunc` always returns 0

Decompilation...

... can fail

```
...prologue

        call    MyFunc
        cmp     rax, 0
        jz      CONTINUE
        add     rsp, 4
CONTINUE:
        pop     rax

...rest of function
```

- Let's say that `MyFunc` always returns 0
    - `rax=0` after call

Decompilation...

... can fail

```
 ...prologue

       call   MyFunc
       cmp    rax, 0
       jz     CONTINUE
       add    rsp, 4
CONTINUE:
       pop    rax

 ...rest of function
```

- Let's say that `MyFunc` always returns 0
  - `rax=0` after call
- Does the decompiler know that `MyFunc` always returns 0?

Decompilation...

... can fail

```
...prologue

       call   MyFunc
       cmp    rax, 0
       jz     CONTINUE
       add    rsp, 4
CONTINUE:
       pop    rax

...rest of function
```

- Let's say that `MyFunc` always returns 0
  - `rax=0` after call
- Does the decompiler know that `MyFunc` always returns 0?
  - hint: depending on the decompiler, maybe/maybe not

Decompilation...

... can fail

```
 ...prologue

        call    MyFunc
        cmp     rax, 0
        jz      CONTINUE
        add     rsp, 4
CONTINUE:
        pop     rax

 ...rest of function
```

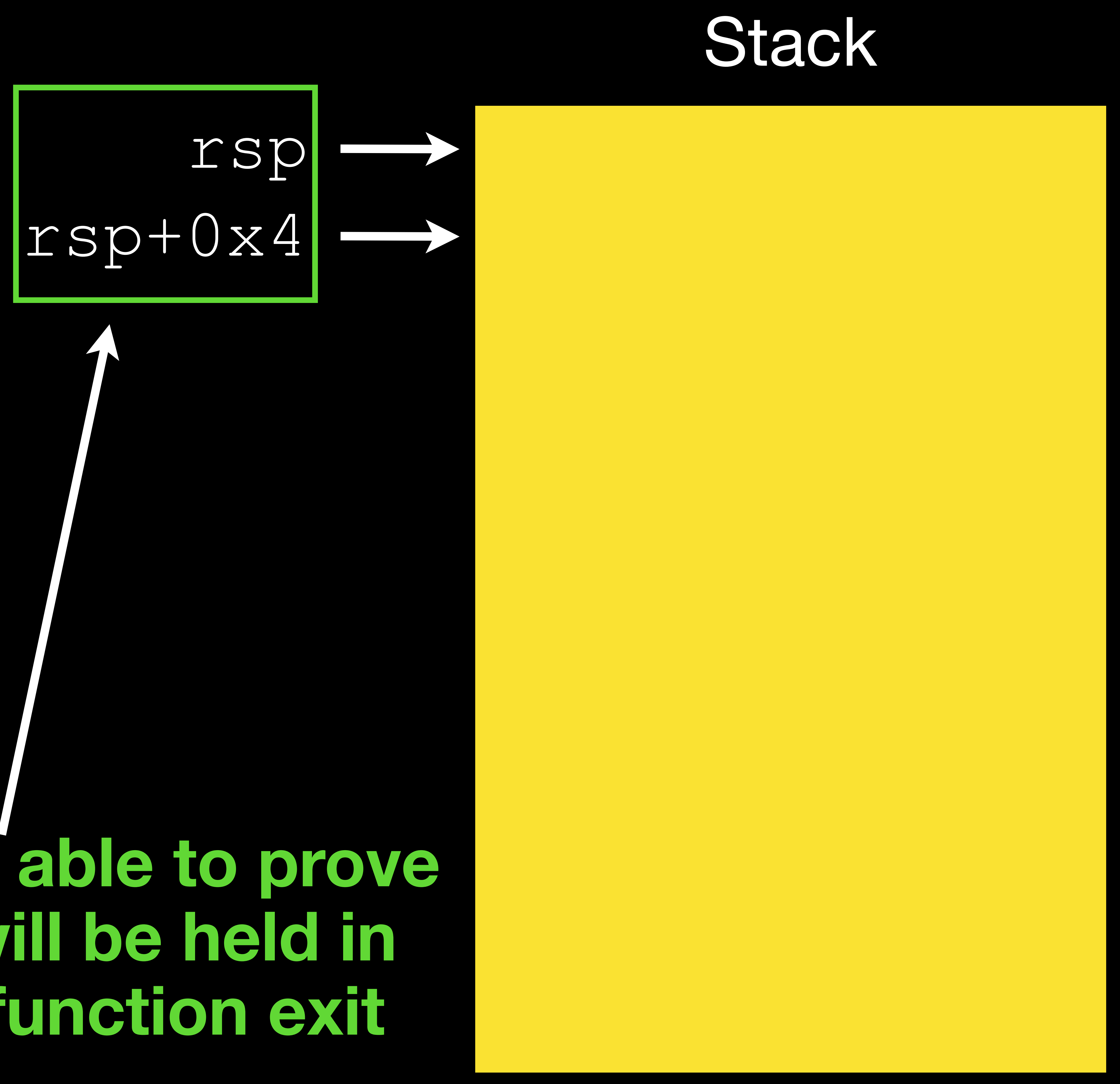
- Let's say that `MyFunc` always returns 0
  - `rax=0` after call
- Does the decompiler know that `MyFunc` always returns 0?
  - hint: depending on the decompiler, maybe/maybe not
- Decompiler cannot statically prove that `add rsp, 4` will not get executed

Decompilation...

... can fail

After a function call returns, the stack of the previous function remains unchanged

rsp/rbp should remain exactly as they were before and after the function call



Decompiler may not be able to prove statically which value will be held in the stack pointer after function exit

Decompilation...

... can fail in multiple ways

- There are an endless list of anti-decompiler tricks

Decompilation…

… can fail in multiple ways

- There are an endless list of anti-decompiler tricks
- No time to cover all of them

Decompilation...

... can fail in multiple ways

- There are an endless list of anti-decompiler tricks
- No time to cover all of them
- Expect some on the challenges :)

Have a break :)

Debuggers…

… let you have full introspection

- A tool that lets you control/observe a program while it runs

Debuggers…

… let you have full introspection

- A tool that lets you control/observe a program while it runs
- Inspect what’s happening “under the hood”

Debuggers…

… let you have full introspection

- A tool that lets you control/observe a program while it runs
- Inspect what's happening "under the hood"
- Step through instructions (or source lines) one at a time

Debuggers…

… let you have full introspection

- A tool that lets you control/observe a program while it runs
- Inspect what’s happening “under the hood”
- Step through instructions (or source lines) one at a time
- Observe memory, registers, local variables, …

Debuggers…

… are really useful

- Static reversing only gets you so far

Debuggers…

… are really useful

- Static reversing only gets you so far
- Debuggers let you examine runtime behaviour

Debuggers…

… are really useful

- Static reversing only gets you so far
- Debuggers let you examine runtime behaviour
  - See real values of variables

Debuggers…

… are really useful

- Static reversing only gets you so far
- Debuggers let you examine runtime behaviour
    - See real values of variables
    - See what functions return

Debuggers…

… are really useful

- Static reversing only gets you so far
- Debuggers let you examine runtime behaviour
    - See real values of variables
    - See what functions return
    - Understand which branches your input causes the program to take

GDB Demo

GDB

Cheat sheet

Start	`gdb ./my_prog`	**Show backtrace/frame**	`(gdb) bt`
			`(gdb) frame`

Set breakpoint	`(gdb) b my_func`
	`(gdb) b *0x400284`
		Inspect registers	`(gdb) info reg`

Step into	`(gdb) step`
	`(gdb) s`
		Inspect memory	`(gdb) x/s` ***address***
			`(gdb) x/20gx $rsp`

Step over	`(gdb) next`
	`(gdb) n`
		Disassemble	`(gdb) disas`
			`(gdb) x/20i $rip`

Step until function exit	`(gdb) finish`
	`(gdb) fin`
		Watch variable/memory	`(gdb) watch` ***varname***

Continue until program stops (i.e., breakpoint)	`(gdb) continue`
	`(gdb) c`
		Quit	`(gdb) q`

Anti debug tricks

cat and mouse

- There are multiple ways for a program to detect if it is being run under a debugger

Anti debug tricks

cat and mouse

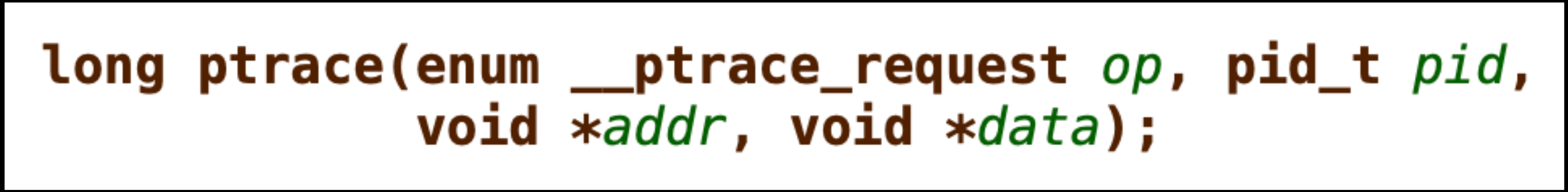
- There are multiple ways for a program to detect if it is being run under a debugger
- On Linux, one such technique is using `ptrace`

Anti debug tricks

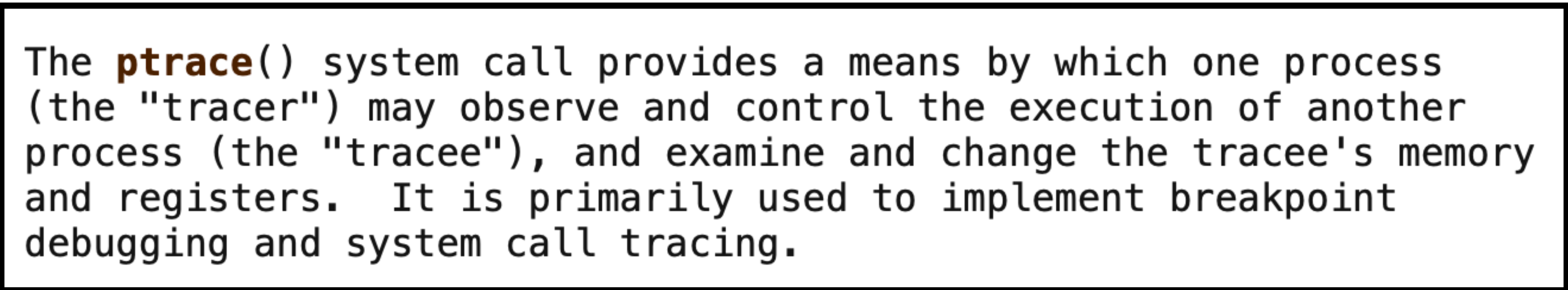
cat and mouse

- There are multiple ways for a program to detect if it is being run under a debugger

- On Linux, one such technique is using `ptrace`



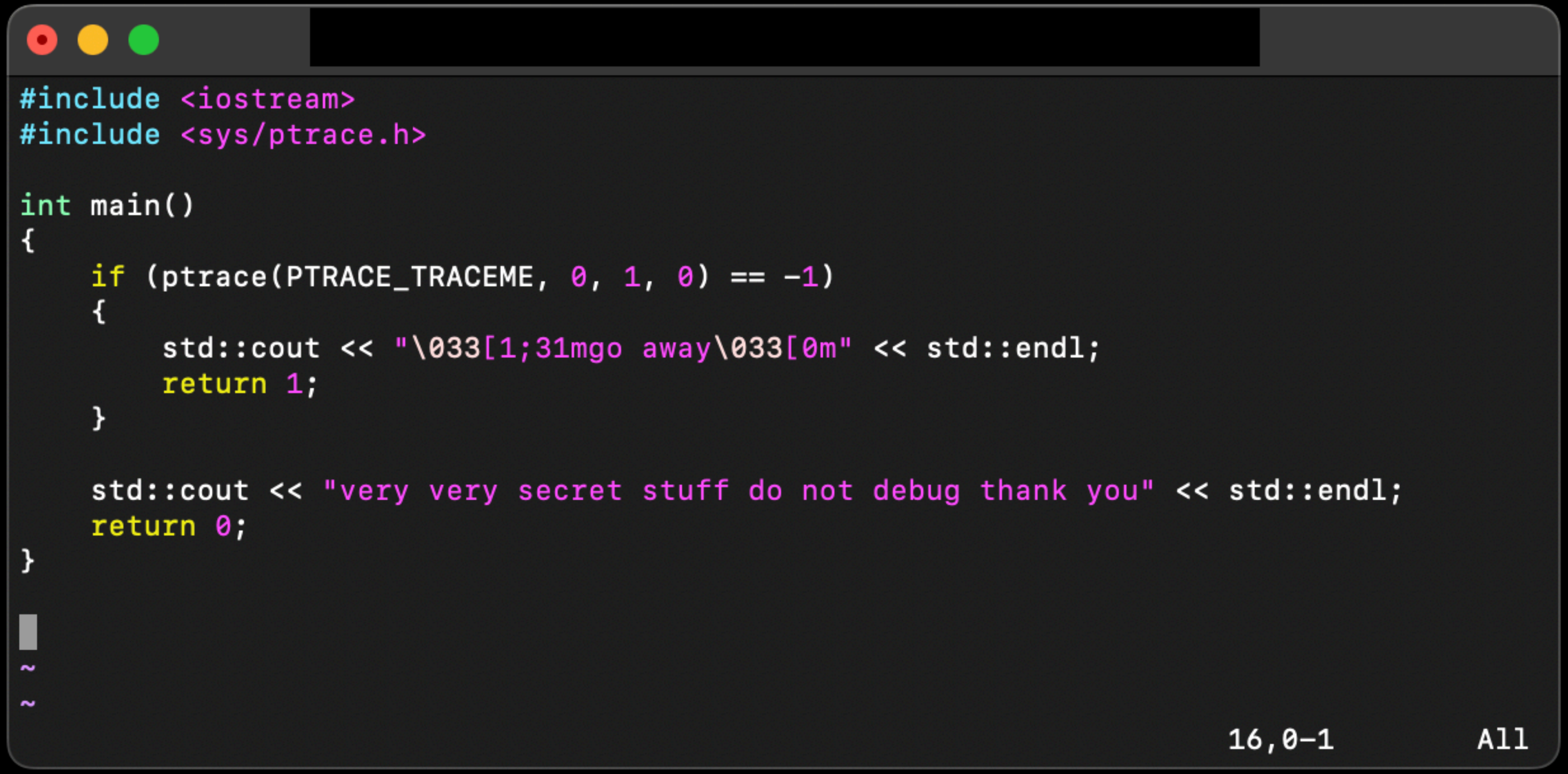
```
long ptrace(enum __ptrace_request op, pid_t pid,
            void *addr, void *data);
```



```
The ptrace() system call provides a means by which one process
(the "tracer") may observe and control the execution of another
process (the "tracee"), and examine and change the tracee's memory
and registers.  It is primarily used to implement breakpoint
debugging and system call tracing.
```

Anti debug tricks

cat and mouse

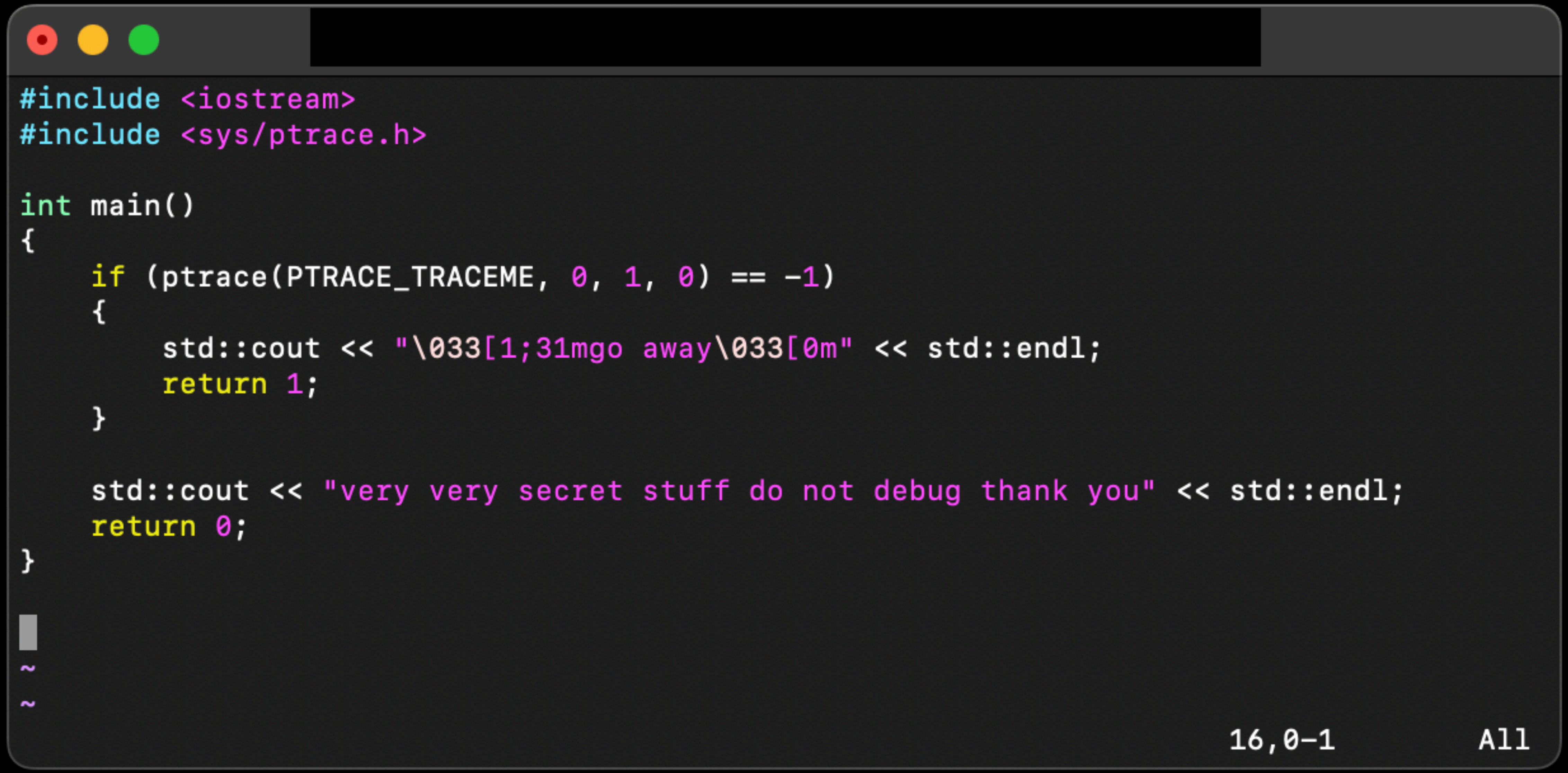


[redacted]

```
#include <iostream>
#include <sys/ptrace.h>

int main()
{
    if (ptrace(PTRACE_TRACEME, 0, 1, 0) == -1)
    {
        std::cout << "\033[1;31mgo away\033[0m" << std::endl;
        return 1;
    }

    std::cout << "very very secret stuff do not debug thank you" << std::endl;
    return 0;
}

~
~
                                                                            16,0-1        All
```

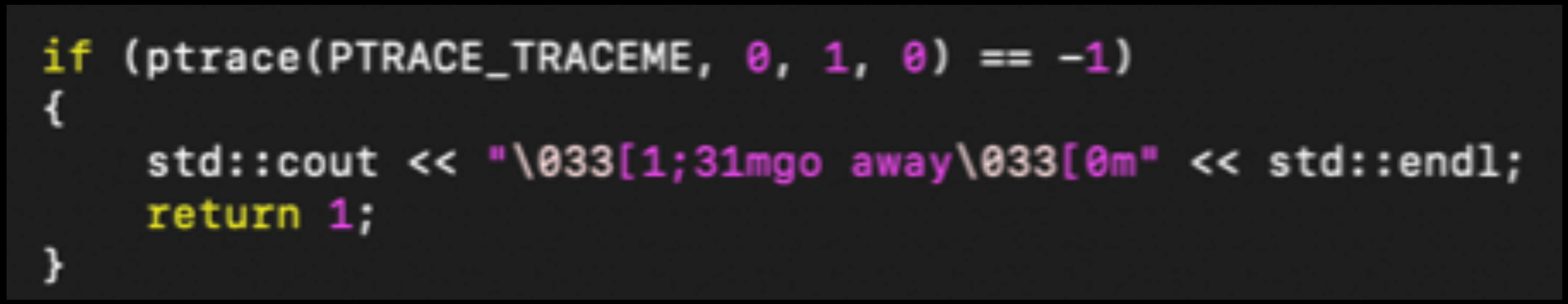
Anti debug tricks

cat and mouse



[redacted]

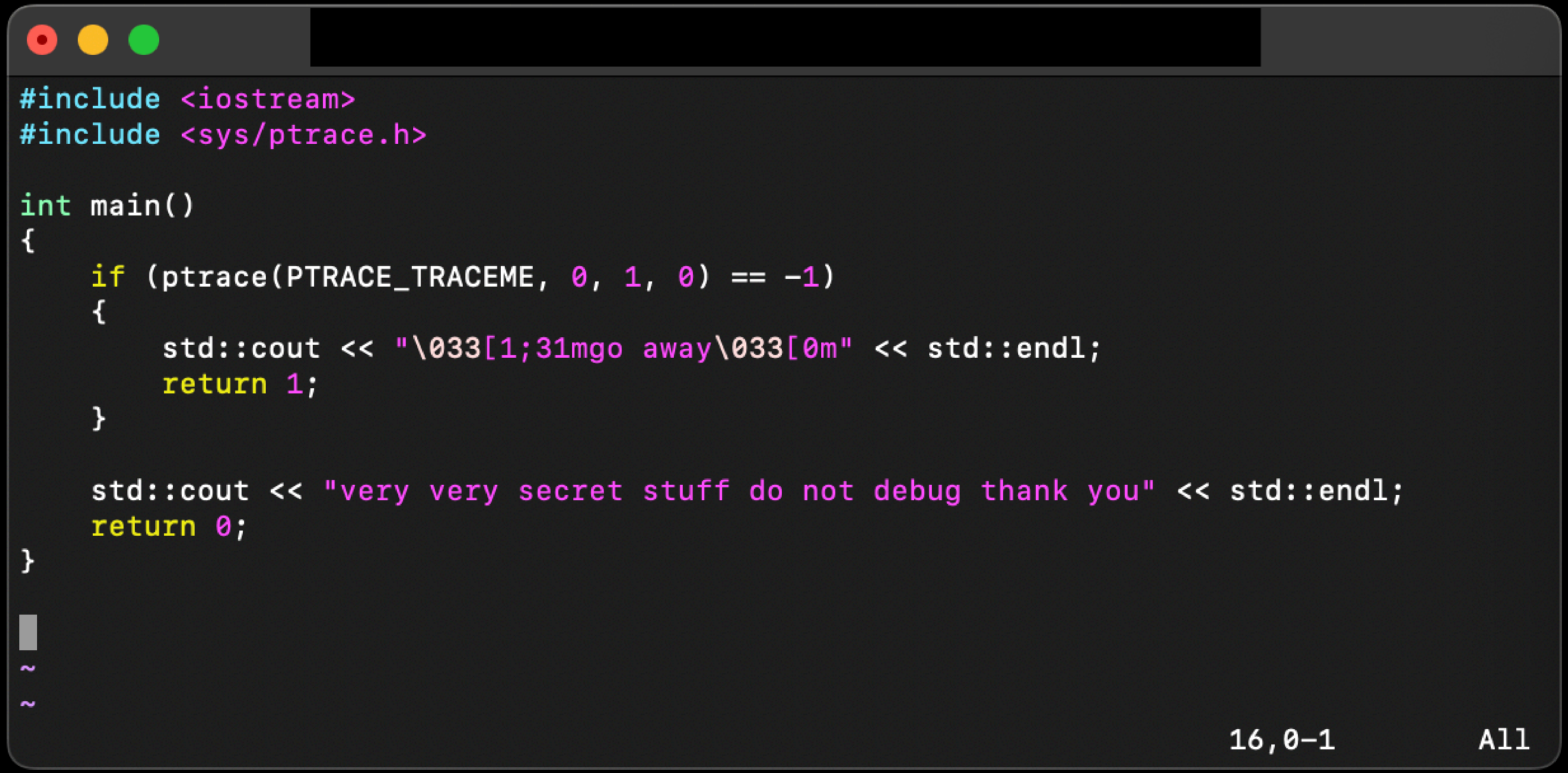
```
#include <iostream>
#include <sys/ptrace.h>

int main()
{
    if (ptrace(PTRACE_TRACEME, 0, 1, 0) == -1)
    {
        std::cout << "\033[1;31mgo away\033[0m" << std::endl;
        return 1;
    }

    std::cout << "very very secret stuff do not debug thank you" << std::endl;
    return 0;
}

~
~
                                                                    16,0-1        All
```



```
if (ptrace(PTRACE_TRACEME, 0, 1, 0) == -1)
{
    std::cout << "\033[1;31mgo away\033[0m" << std::endl;
    return 1;
}
```

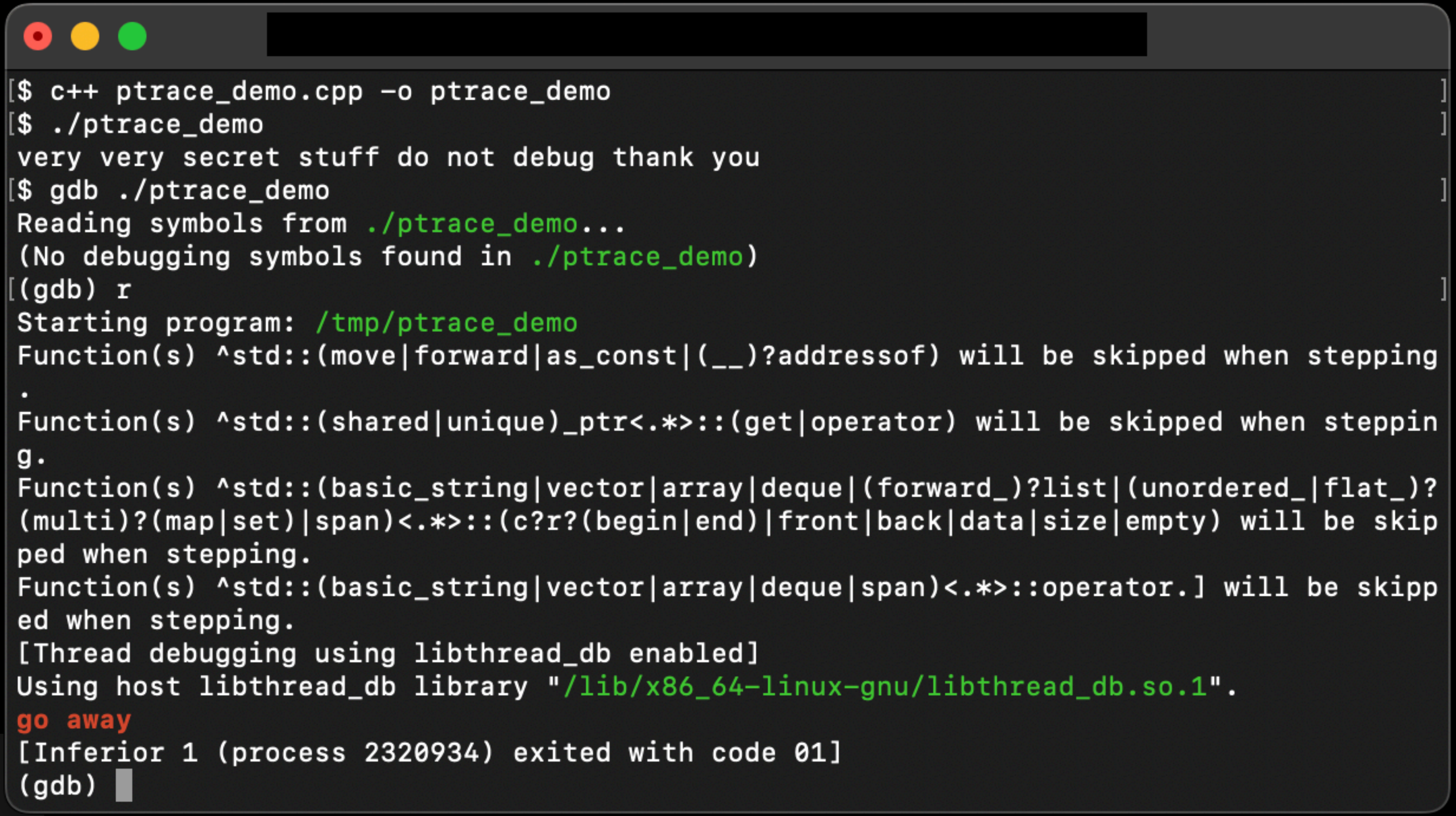
Anti debug tricks

cat and mouse



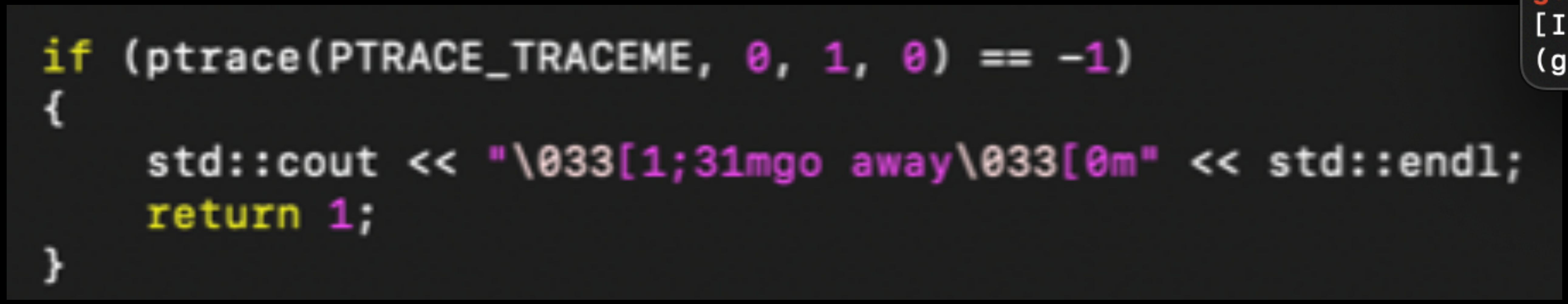
[redacted]

```
#include <iostream>
#include <sys/ptrace.h>

int main()
{
    if (ptrace(PTRACE_TRACEME, 0, 1, 0) == -1)
    {
        std::cout << "\033[1;31mgo away\033[0m" << std::endl;
        return 1;
    }

    std::cout << "very very secret stuff do not debug thank you" << std::endl;
    return 0;
}

~
~
                                                                16,0-1        All
```



[redacted]

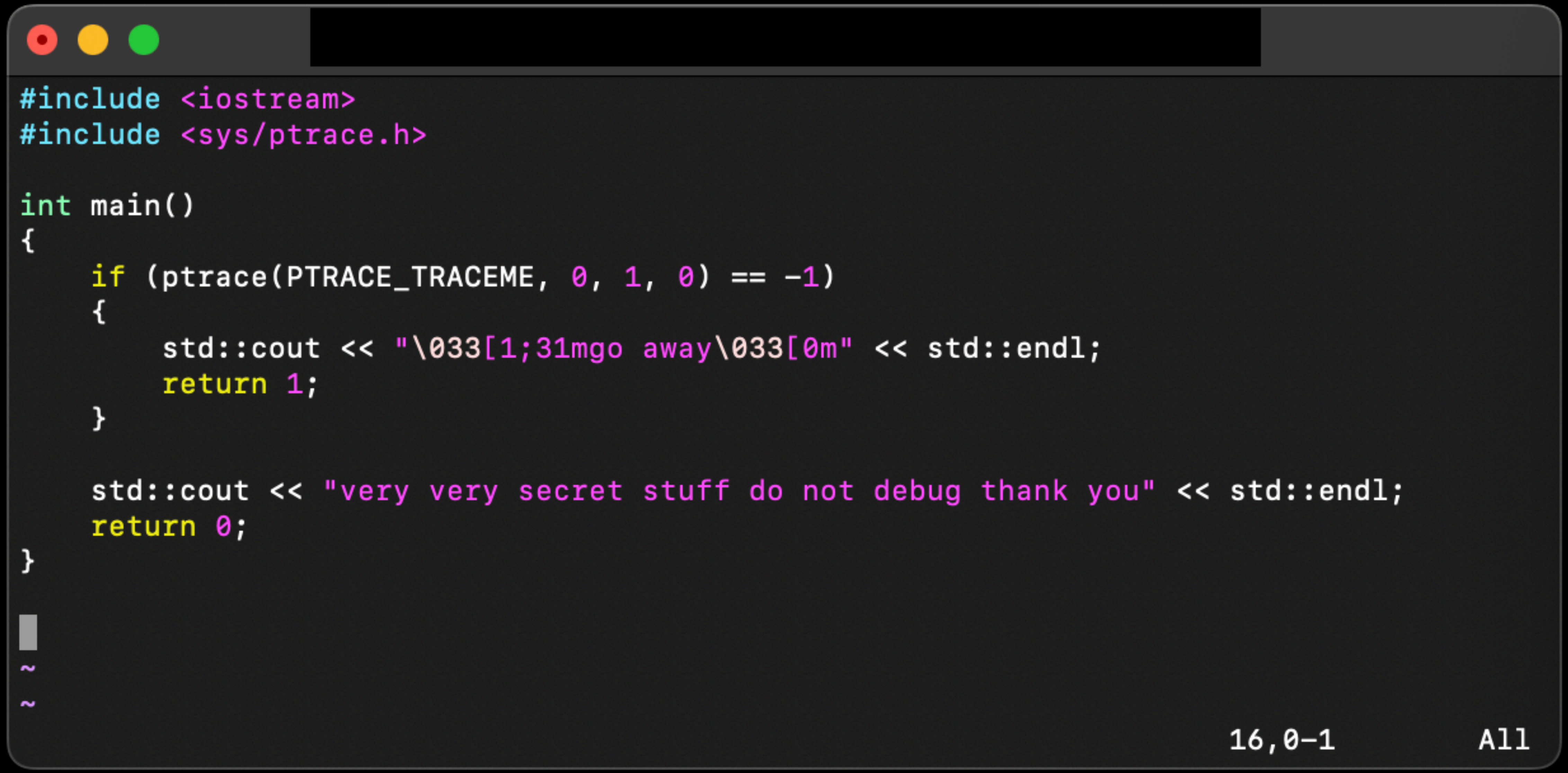
```
[$ c++ ptrace_demo.cpp -o ptrace_demo
[$ ./ptrace_demo
very very secret stuff do not debug thank you
[$ gdb ./ptrace_demo
Reading symbols from ./ptrace_demo...
(No debugging symbols found in ./ptrace_demo)
[(gdb) r
Starting program: /tmp/ptrace_demo
Function(s) ^std::(move|forward|as_const|(__)?addressof) will be skipped when stepping
.
Function(s) ^std::(shared|unique)_ptr<.*>::(get|operator) will be skipped when steppin
g.
Function(s) ^std::(basic_string|vector|array|deque|(forward_)?list|(unordered_|flat_)?
(multi)?(map|set)|span)<.*>::(c?r?(begin|end)|front|back|data|size|empty) will be skip
ped when stepping.
Function(s) ^std::(basic_string|vector|array|deque|span)<.*>::operator.] will be skipp
ed when stepping.
[Thread debugging using libthread_db enabled]
Using host libthread_db library "/lib/x86_64-linux-gnu/libthread_db.so.1".
go away
[Inferior 1 (process 2320934) exited with code 01]
(gdb) 
```



```
if (ptrace(PTRACE_TRACEME, 0, 1, 0) == -1)
{
    std::cout << "\033[1;31mgo away\033[0m" << std::endl;
    return 1;
}
```

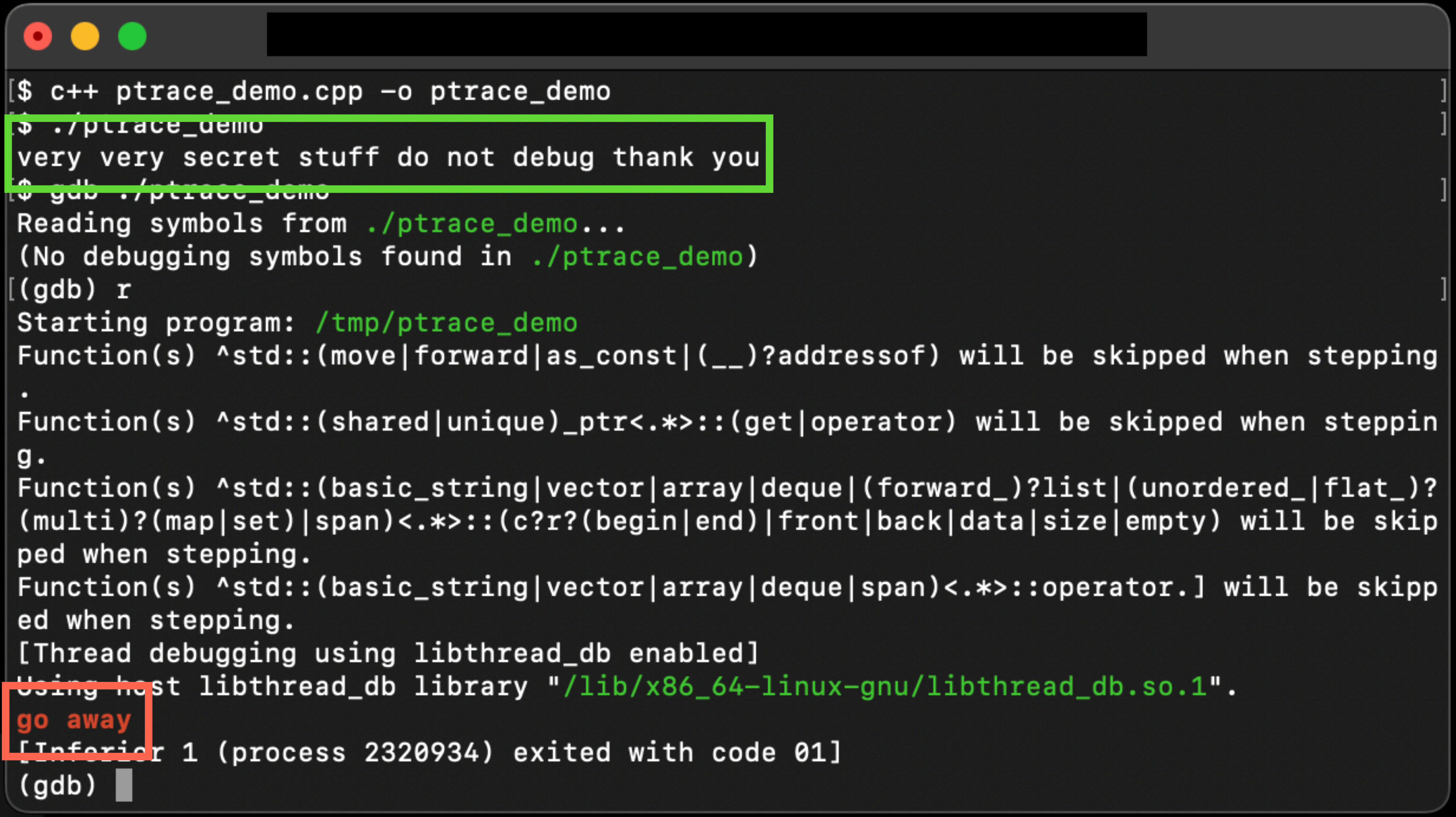
Anti debug tricks

cat and mouse



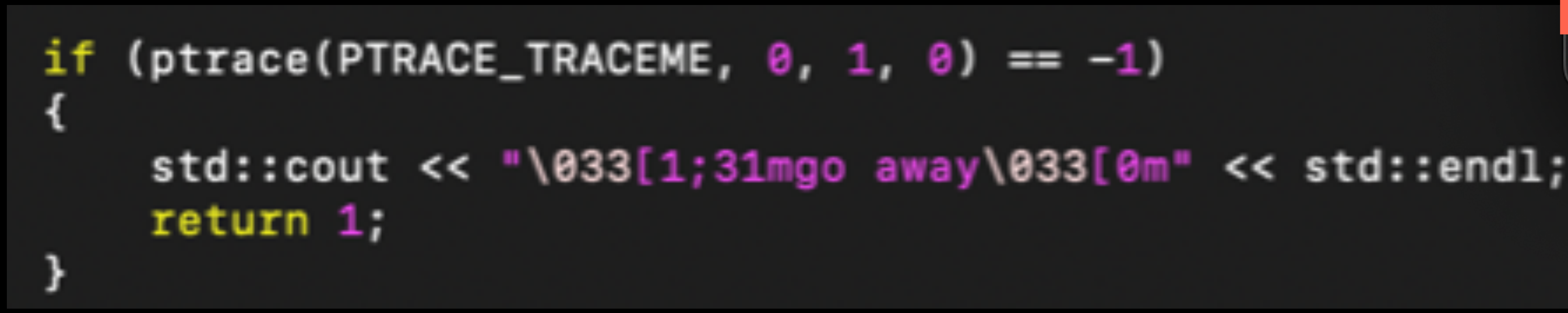
[redacted]

```
#include <iostream>
#include <sys/ptrace.h>

int main()
{
    if (ptrace(PTRACE_TRACEME, 0, 1, 0) == -1)
    {
        std::cout << "\033[1;31mgo away\033[0m" << std::endl;
        return 1;
    }

    std::cout << "very very secret stuff do not debug thank you" << std::endl;
    return 0;
}

~
~
                                                          16,0-1        All
```



[redacted]

```
$ c++ ptrace_demo.cpp -o ptrace_demo
$ ./ptrace_demo
very very secret stuff do not debug thank you
$ gdb ./ptrace_demo
Reading symbols from ./ptrace_demo...
(No debugging symbols found in ./ptrace_demo)
(gdb) r
Starting program: /tmp/ptrace_demo
Function(s) ^std::(move|forward|as_const|(__)?addressof) will be skipped when stepping
.
Function(s) ^std::(shared|unique)_ptr<.*>::(get|operator) will be skipped when steppin
g.
Function(s) ^std::(basic_string|vector|array|deque|(forward_)?list|(unordered_|flat_)?
(multi)?(map|set)|span)<.*>::(c?r?(begin|end)|front|back|data|size|empty) will be skip
ped when stepping.
Function(s) ^std::(basic_string|vector|array|deque|span)<.*>::operator.] will be skipp
ed when stepping.
[Thread debugging using libthread_db enabled]
Using host libthread_db library "/lib/x86_64-linux-gnu/libthread_db.so.1".
go away
[Inferior 1 (process 2320934) exited with code 01]
(gdb)
```



```
if (ptrace(PTRACE_TRACEME, 0, 1, 0) == -1)
{
    std::cout << "\033[1;31mgo away\033[0m" << std::endl;
    return 1;
}
```

Anti debug tricks

cat and mouse

- Many such techniques

Anti debug tricks

cat and mouse

- Many such techniques
    - Runtime `.text` CRC computation

Anti debug tricks

cat and mouse

- Many such techniques
  - Runtime `.text` CRC computation
  - `int3` (debugger trap)

Anti debug tricks

cat and mouse

- Many such techniques
    - Runtime `.text` CRC computation
    - `int3` (debugger trap)
    - `/proc/self/status (TracerPid)`

Anti debug tricks

cat and mouse

- Many such techniques
  - Runtime `.text` CRC computation
  - `int3` (debugger trap)
  - `/proc/self/status (TracerPid)`
- This talk is already very long, so I’ll leave learning these as an exercise to you

Anti debug tricks

cat and mouse

- Many such techniques
  - Runtime `.text` CRC computation
  - `int3` (debugger trap)
  - `/proc/self/status (TracerPid)`
- This talk is already very long, so I’ll leave learning these as an exercise to you
- You can always find a way around these (i.e., patch the program)

Packers...

... pack stuff

- Executables can get pretty big

Packers...

... pack stuff

- Executables can get pretty big
- **Packers** exist to compress an executable into a smaller binary

Packers...

... pack stuff

- Executables can get pretty big
- **Packers** exist to compress an executable into a smaller binary
- Resulting binary is compressed payload + decompressor

Packers...

... pack stuff

- Executables can get pretty big
- **Packers** exist to compress an executable into a smaller binary
- Resulting binary is compressed payload + decompressor
- At runtime, decompresses payload and jumps to it

Packers...

... pack stuff

- Executables can get pretty big
- **Packers** exist to compress an executable into a smaller binary
- Resulting binary is compressed payload + decompressor
- At runtime, decompresses payload and jumps to it
- Common in malware and some “commercial software protection solutions”

Packers...

... pack stuff

- Executables can get pretty big
- **Packers** exist to compress an executable into a smaller binary
- Resulting binary is compressed payload + decompressor
- At runtime, decompresses payload and jumps to it
- Common in malware and some “commercial software protection solutions”
- You can typically recognise the use of a packer from `strings`

Packers...

... pack stuff

- Executables can get pretty big
- **Packers** exist to compress an executable into a smaller binary
- Resulting binary is compressed payload + decompressor
- At runtime, decompresses payload and jumps to it
- Common in malware and some “commercial software protection solutions”
- You can typically recognise the use of a packer from `strings`
  - `“This file is packed with the UPX executable packer”`

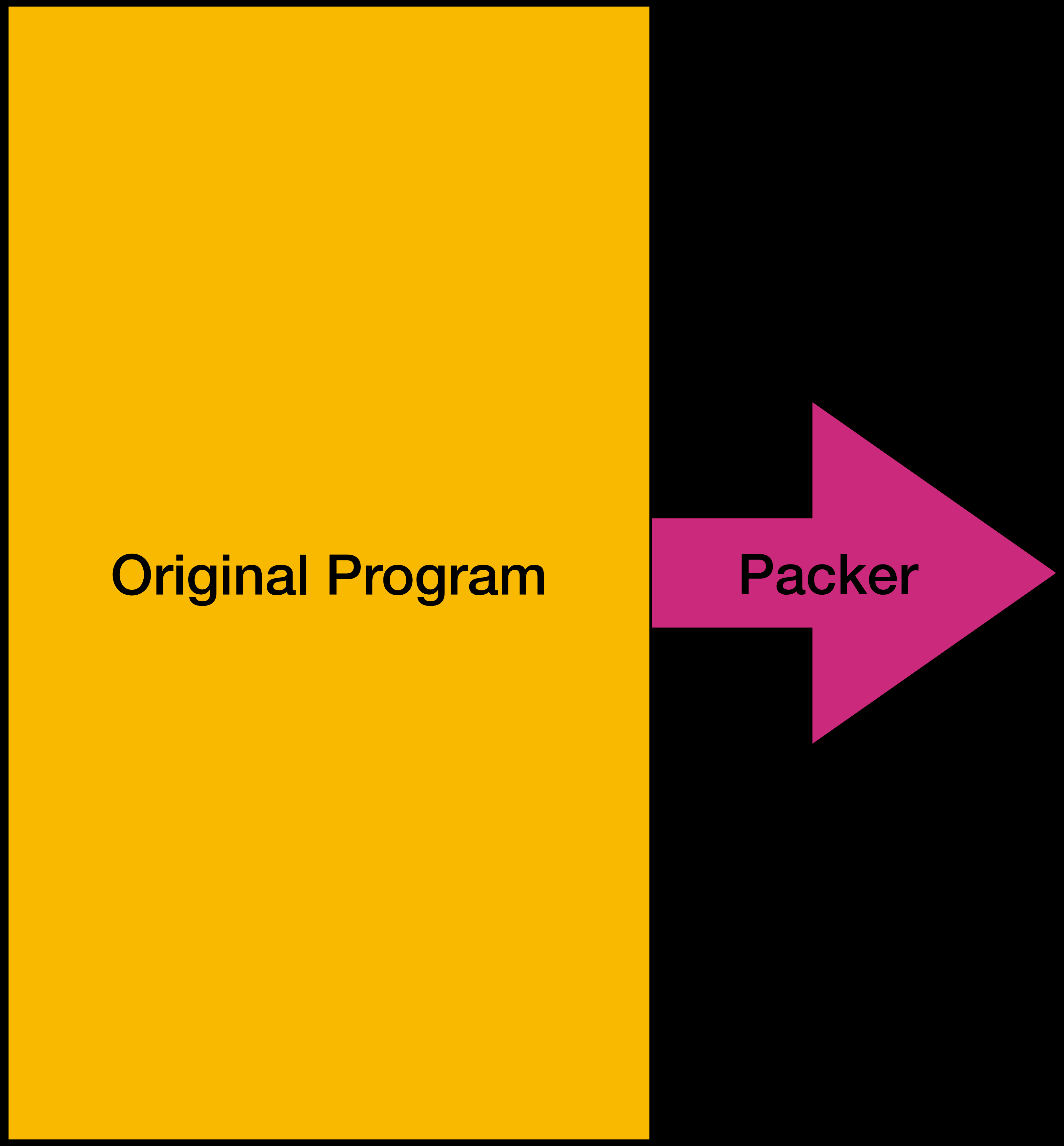
Packers...

... pack stuff

Original Program

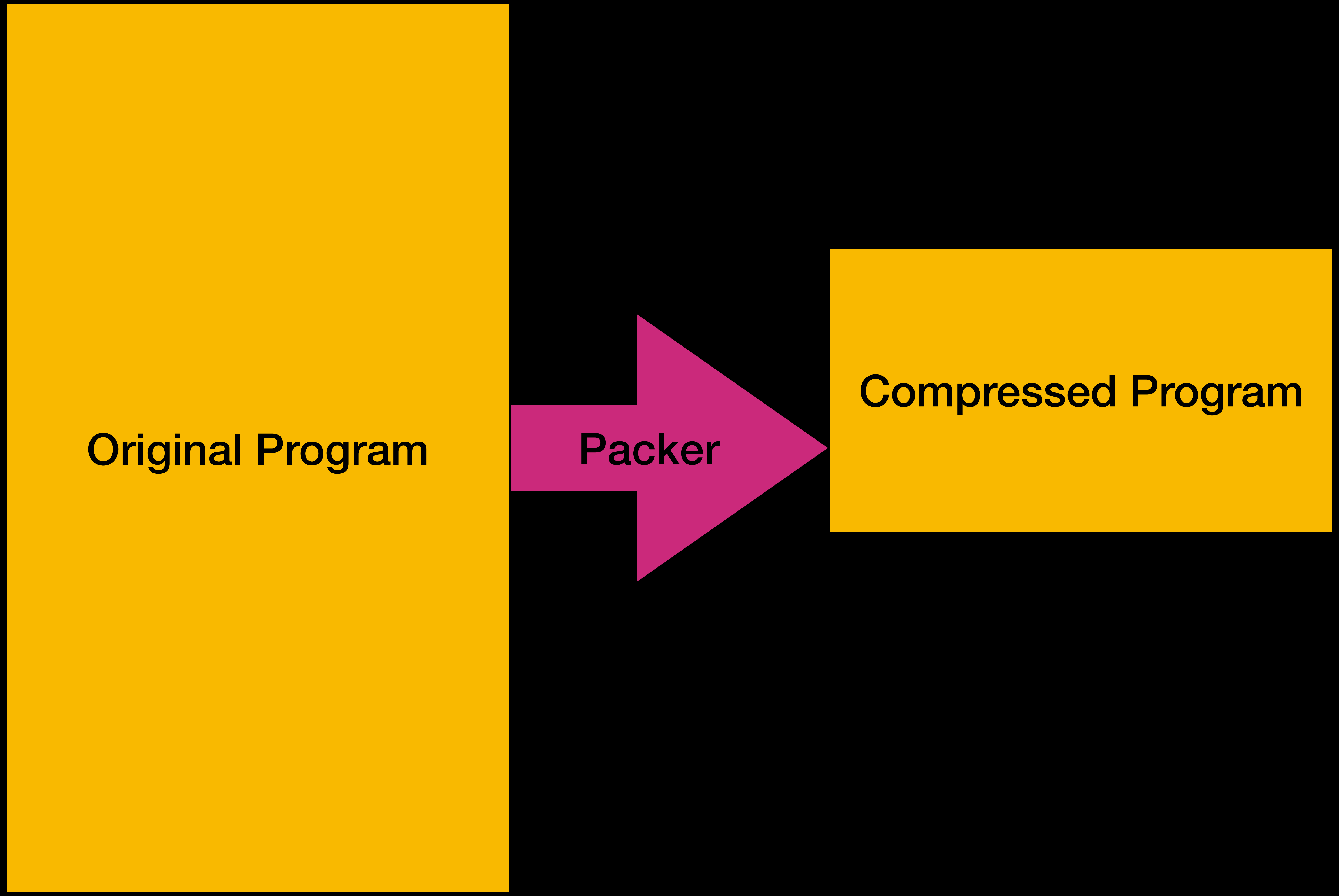
Packers...

... pack stuff



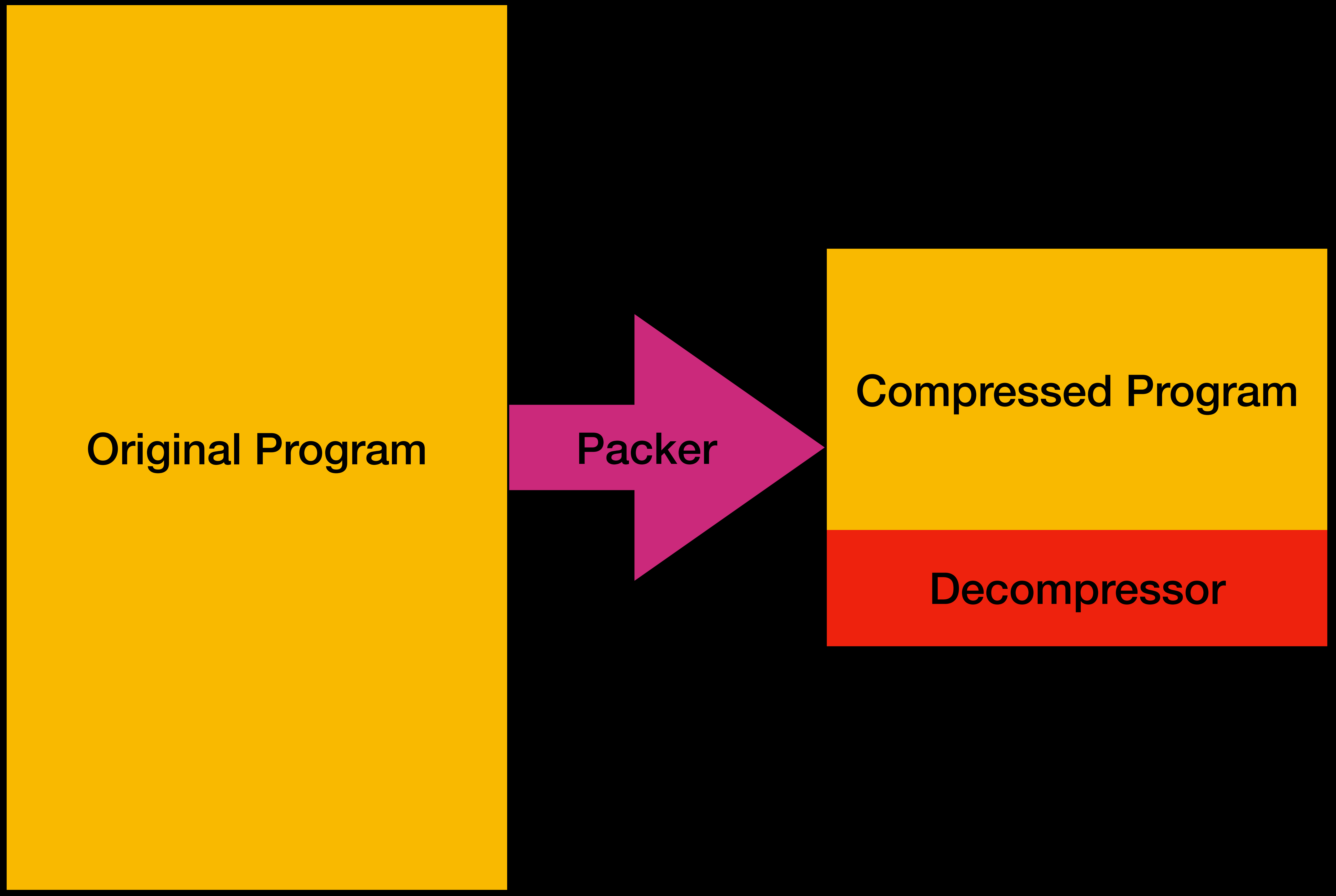
Packers...

... pack stuff



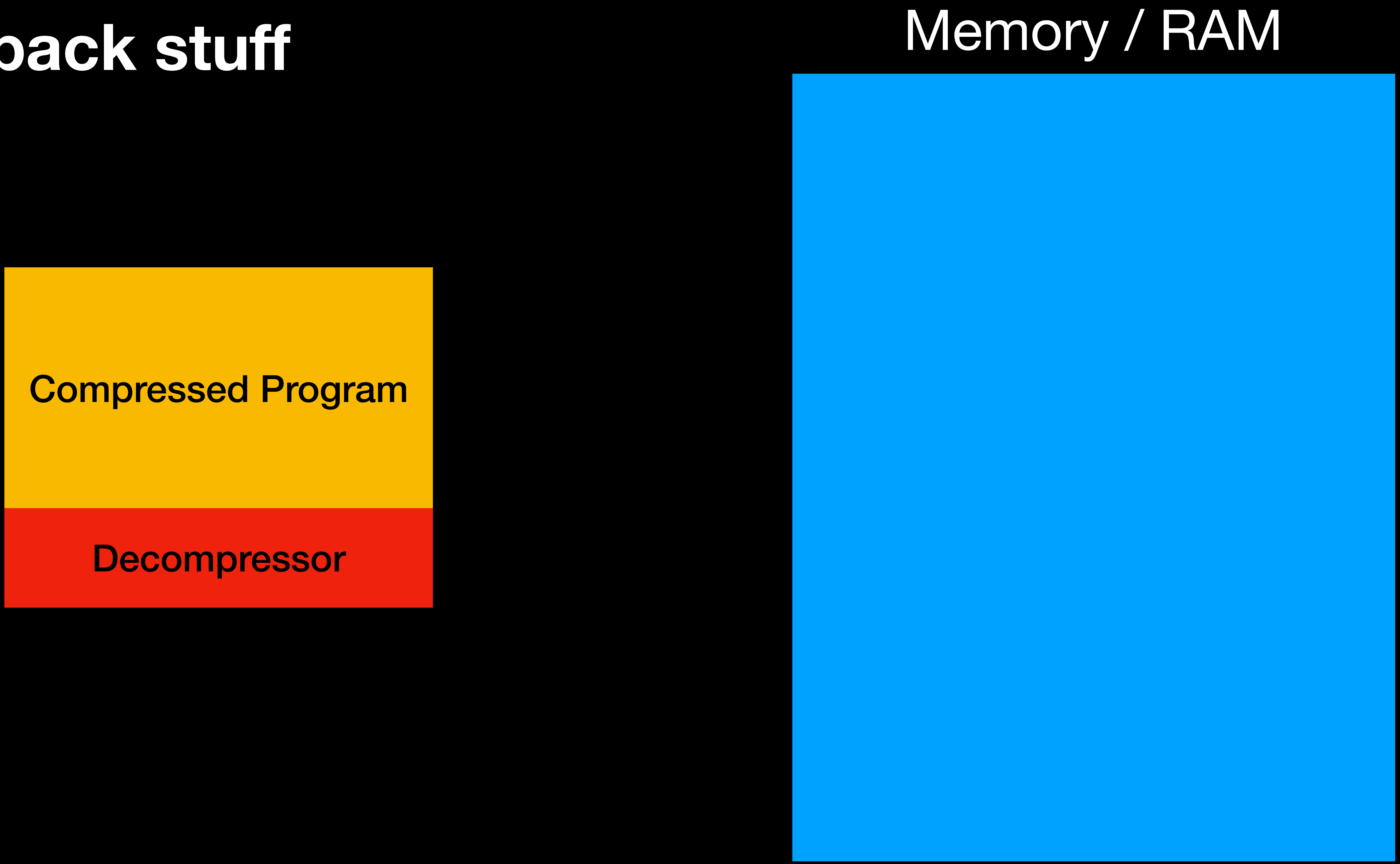
Packers...

... pack stuff



Packers...

... also unpack stuff



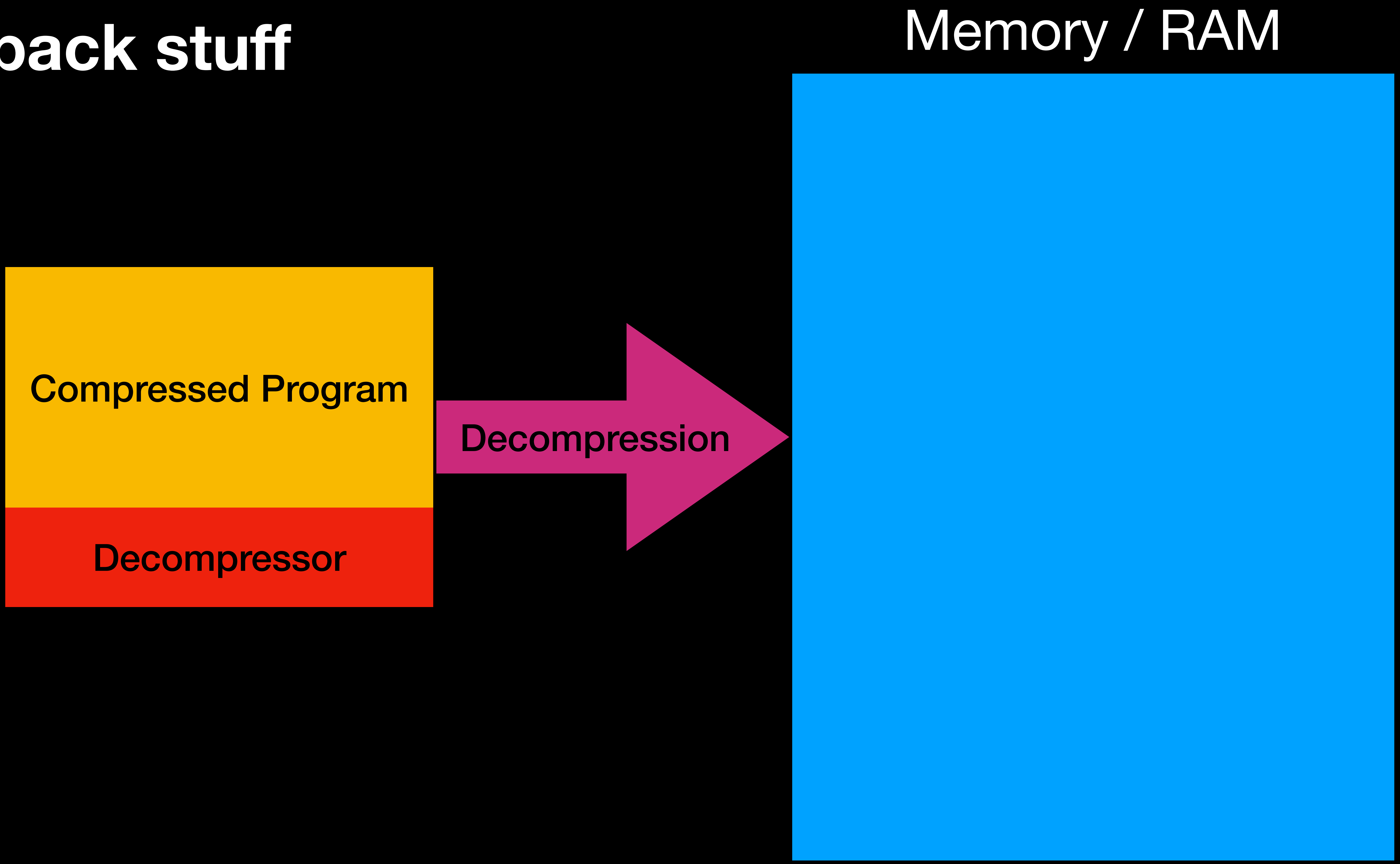
Compressed Program

Decompressor

Memory / RAM

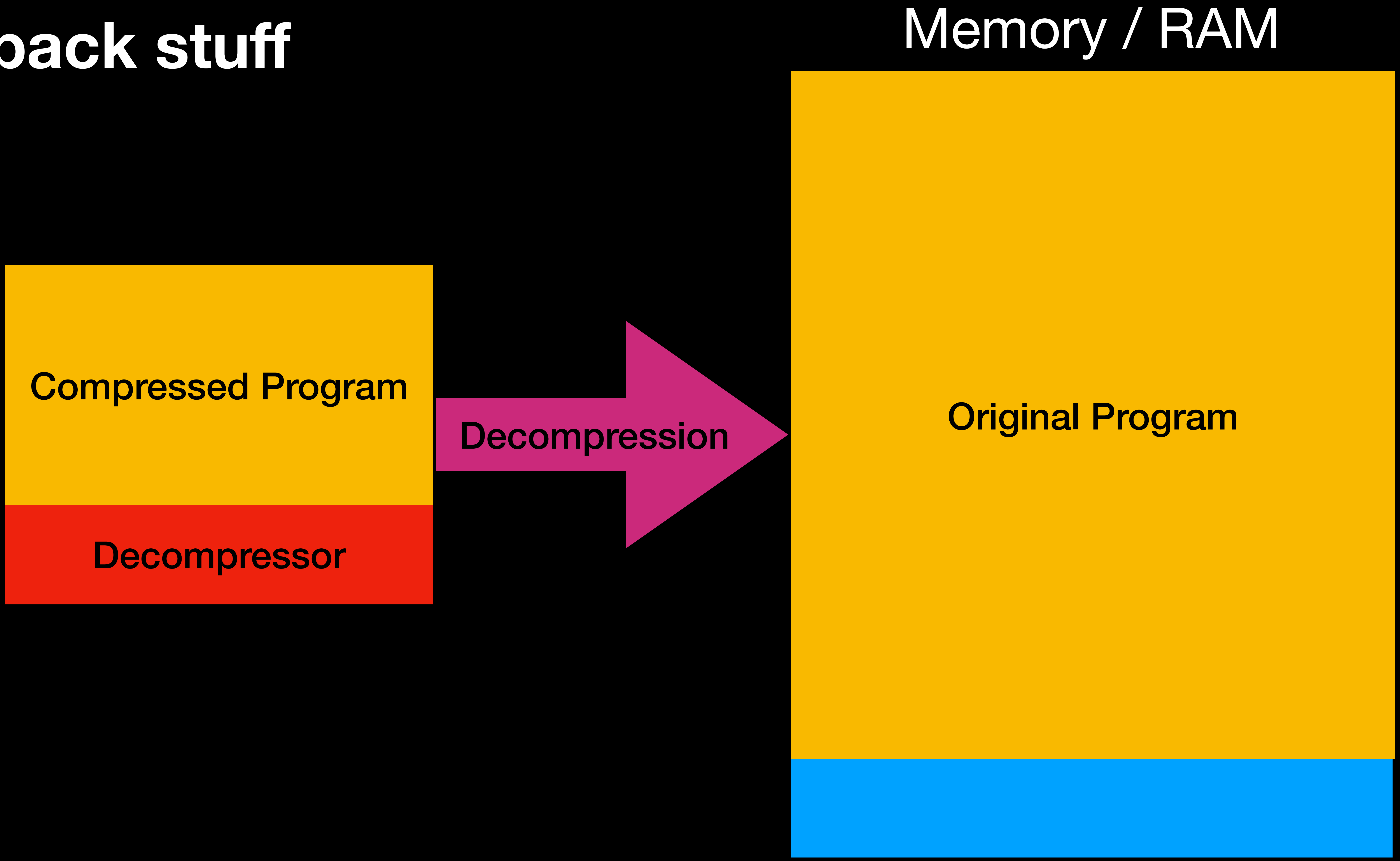
Packers...

... also unpack stuff



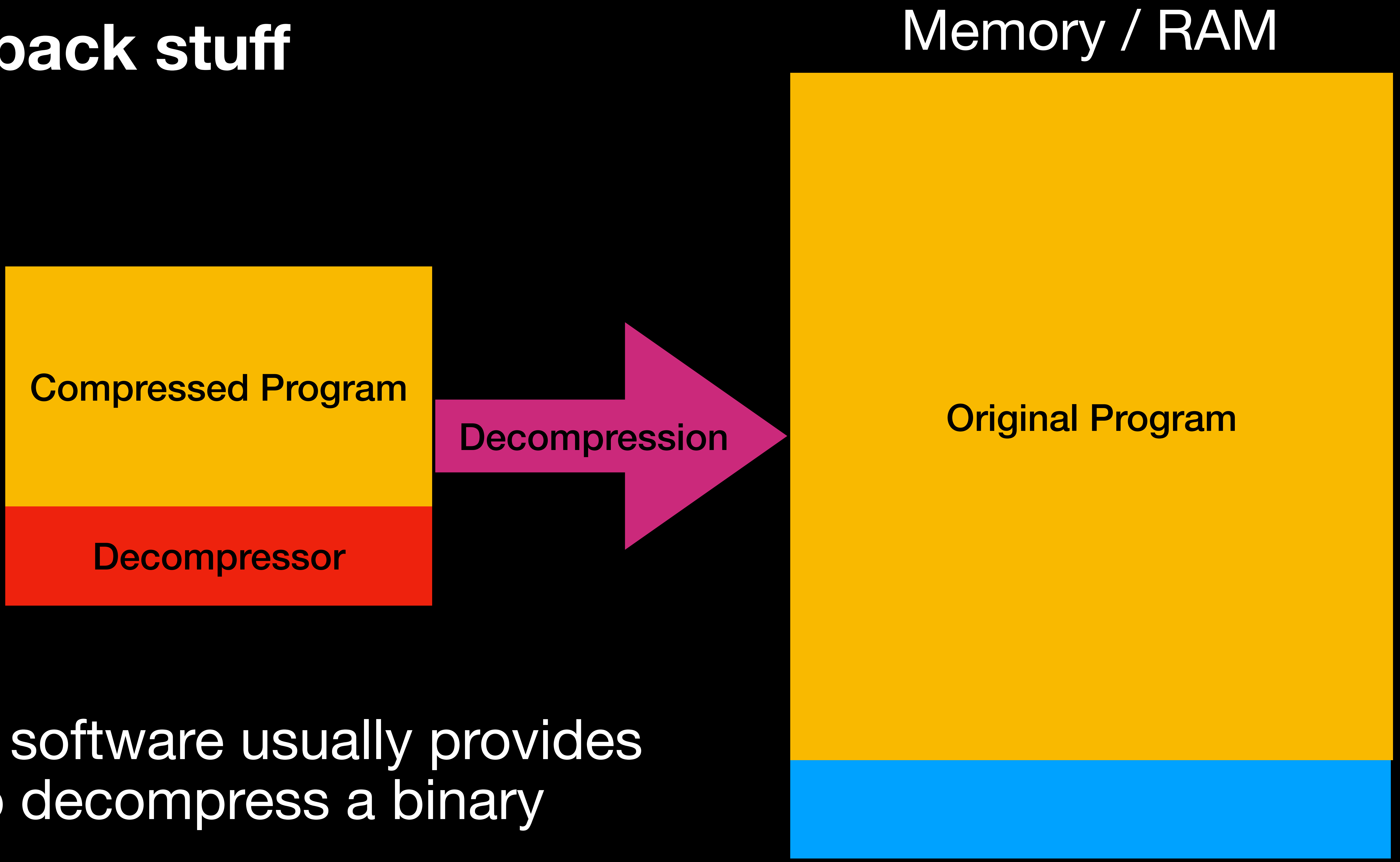
Packers...

... also unpack stuff



Packers...

... also unpack stuff



Packing software usually provides a way to decompress a binary

i.e. `upx -d`

Packers...

... pack and unpack things

- A lot of packers exist

Packers...

... pack and unpack things

- A lot of packers exist
    - Yoda

Packers...

... pack and unpack things

- A lot of packers exist
  - Yoda
  - UPX

Packers...

... pack and unpack things

- A lot of packers exist
  - Yoda
  - UPX
  - 20to4

Packers...

... pack and unpack things

- A lot of packers exist
    - Yoda
    - UPX
    - 20to4
    - eXPressor

Packers...

... pack and unpack things

- A lot of packers exist
  - Yoda
  - UPX
  - 20to4
  - eXPressor
- Some may offer password protection, encryption, etc

Packers...

... pack and unpack things

- A lot of packers exist
  - Yoda
  - UPX
  - 20to4
  - eXPressor
- Some may offer password protection, encryption, etc
- Another “exercise to the reader” to go and learn more

Obfuscaters...

... make your life harder

- Obfuscation is the process of transforming code so it’s harder to read but still does the same thing

Obfuscaters...

... make your life harder

- Obfuscation is the process of transforming code so it’s harder to read but still does the same thing
- Goal: frustrate reverse engineers (you!)

Obfuscaters…

… make your life harder

- Obfuscation is the process of transforming code so it’s harder to read but still does the same thing
- Goal: frustrate reverse engineers (you!)
- Commonly used in malware, DRM,

Obfuscaters...

... make your life harder

- Obfuscation is the process of transforming code so it’s harder to read but still does the same thing
- Goal: frustrate reverse engineers (you!)
- Commonly used in malware, DRM, and CTFs :^)

Obfuscaters...

... why?

- Hide algorithms, secret constants (decryption keys, flags)

Obfuscaters...

... why?

- Hide algorithms, secret constants (decryption keys, flags)
- Slow down reverse engineers

Obfuscaters…

… why?

- Hide algorithms, secret constants (decryption keys, flags)
- Slow down reverse engineers
- Avoid threat detection (less common)

Obfuscaters...

... why?

- Hide algorithms, secret constants (decryption keys, flags)
- Slow down reverse engineers
- Avoid threat detection (less common)
- Increase cost of defence

Obfuscaters...

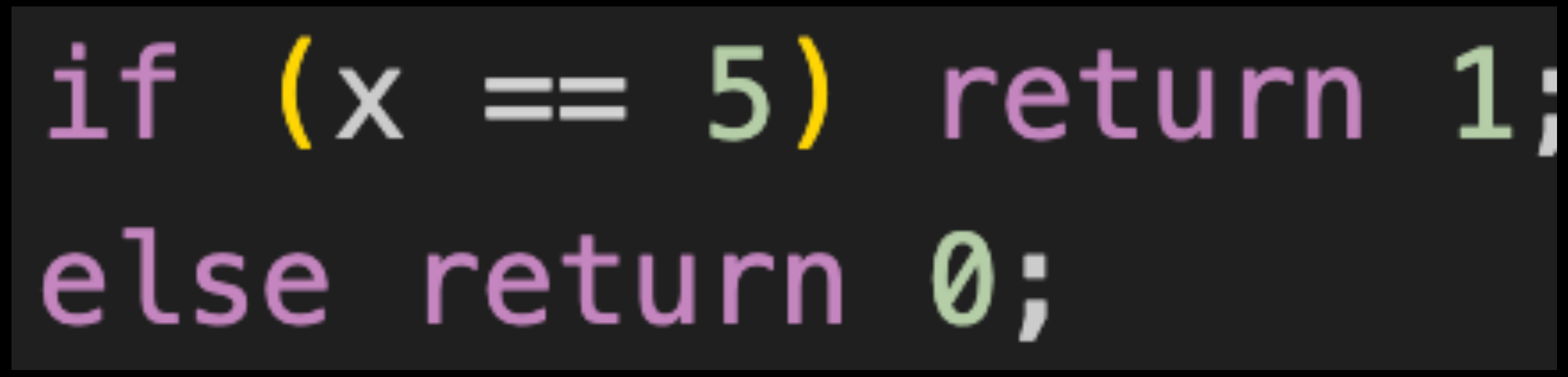
... how? - control flow flattening

- Turn neat control flow structures into spaghetti mess

Obfuscaters...

... how? - control flow flattening

- Turn neat control flow structures into spaghetti mess

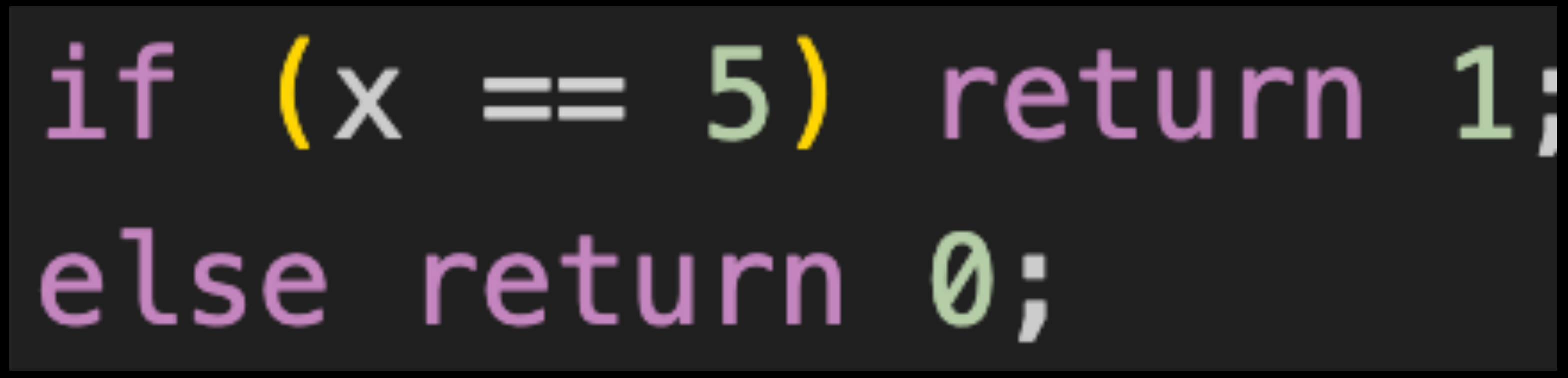


```
if (x == 5) return 1;
else return 0;
```

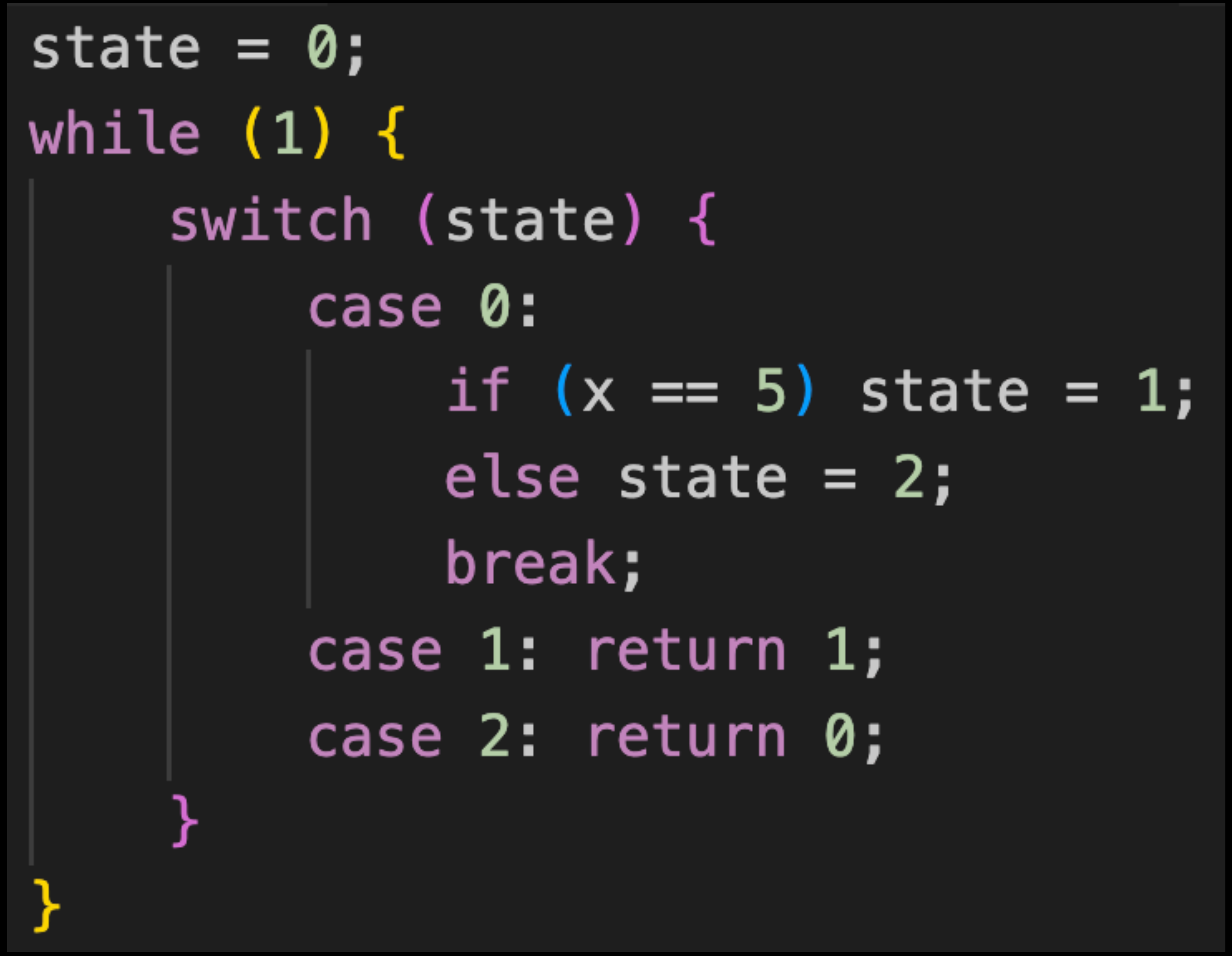
Obfuscaters...

... how? - control flow flattening

- Turn neat control flow structures into spaghetti mess



```
if (x == 5) return 1;
else return 0;
```

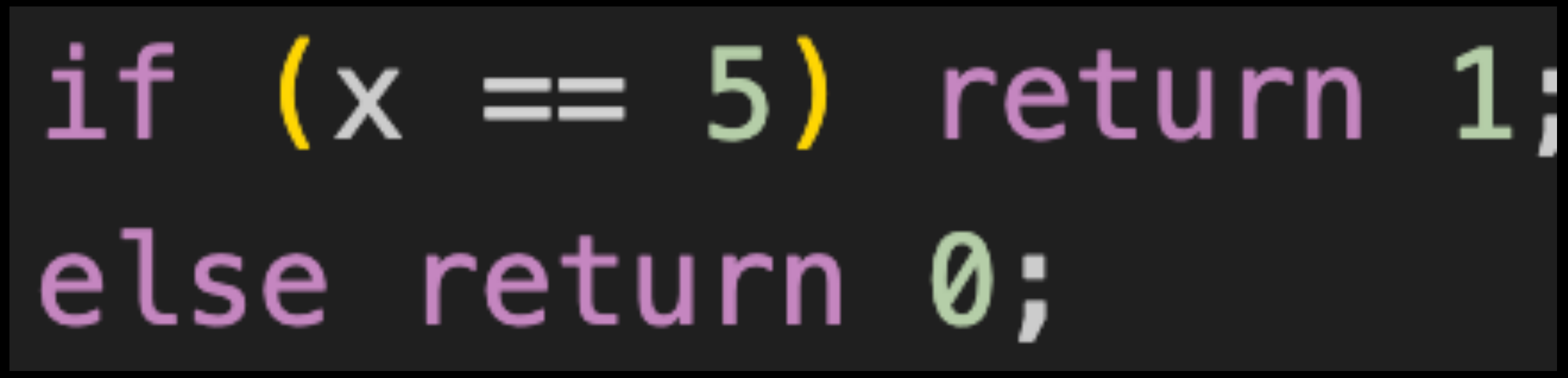


```
state = 0;
while (1) {
    switch (state) {
        case 0:
            if (x == 5) state = 1;
            else state = 2;
            break;
        case 1: return 1;
        case 2: return 0;
    }
}
```

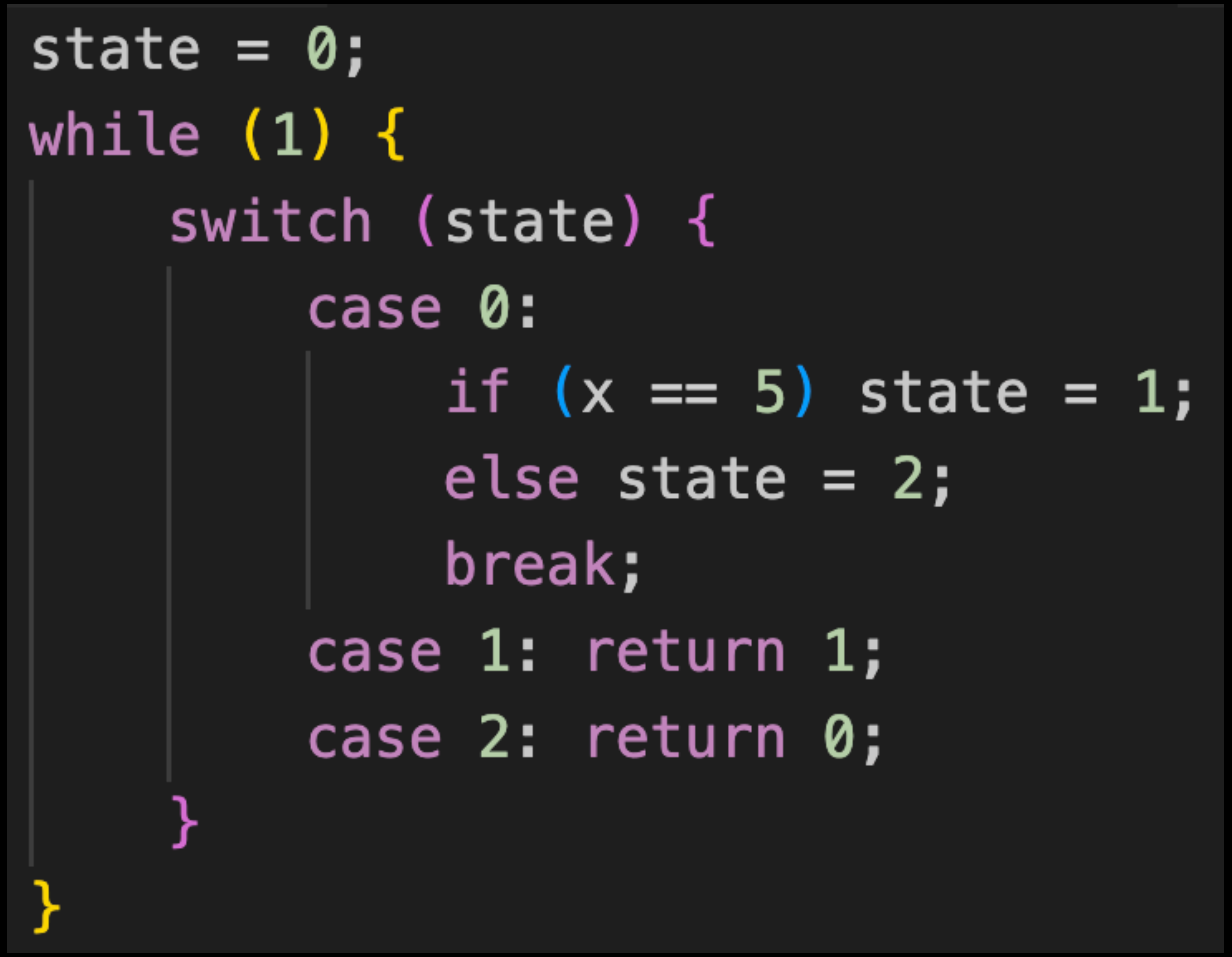
Obfuscaters...

... how? - control flow flattening

- Turn neat control flow structures into spaghetti mess



```
if (x == 5) return 1;
else return 0;
```



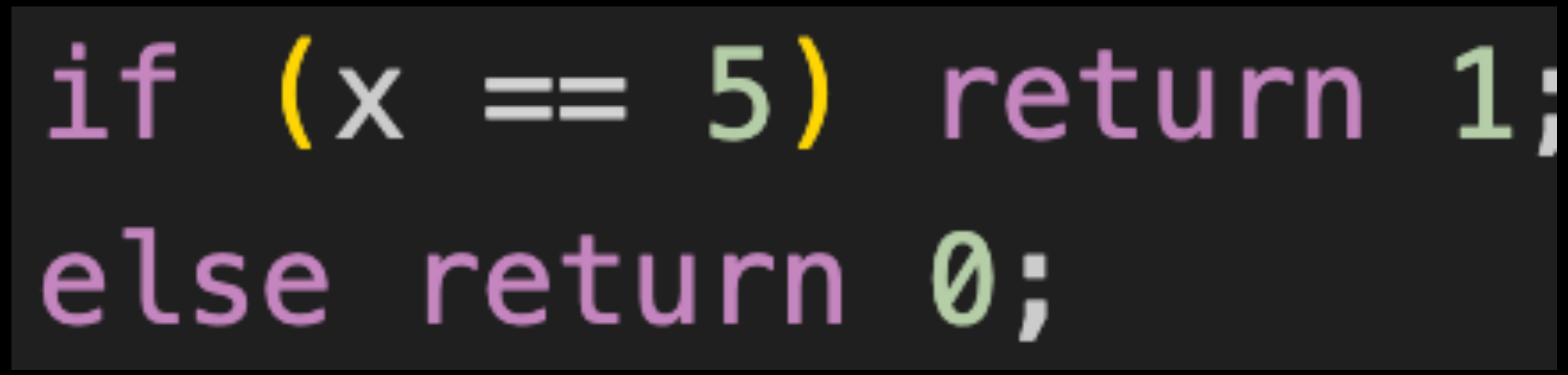
```
state = 0;
while (1) {
    switch (state) {
        case 0:
            if (x == 5) state = 1;
            else state = 2;
            break;
        case 1: return 1;
        case 2: return 0;
    }
}
```

Both of these return 1 if x == 5

Obfuscaters...

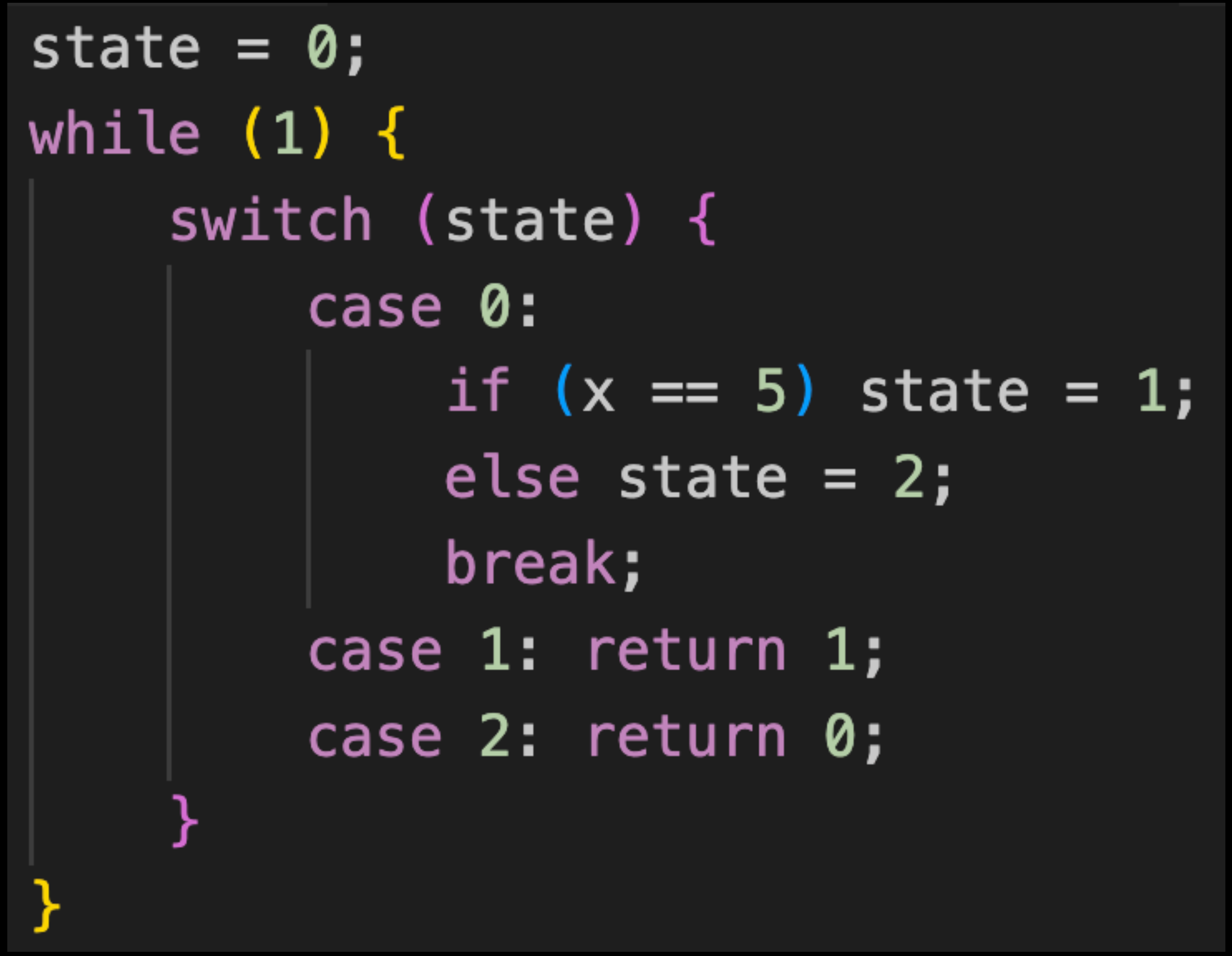
... how? - control flow flattening

- Turn neat control flow structures into spaghetti mess



```
if (x == 5) return 1;
else return 0;
```

Both of these return 1 if x == 5



```
state = 0;
while (1) {
    switch (state) {
        case 0:
            if (x == 5) state = 1;
            else state = 2;
            break;
        case 1: return 1;
        case 2: return 0;
    }
}
```

But this is much harder
to reason about

Obfuscaters...

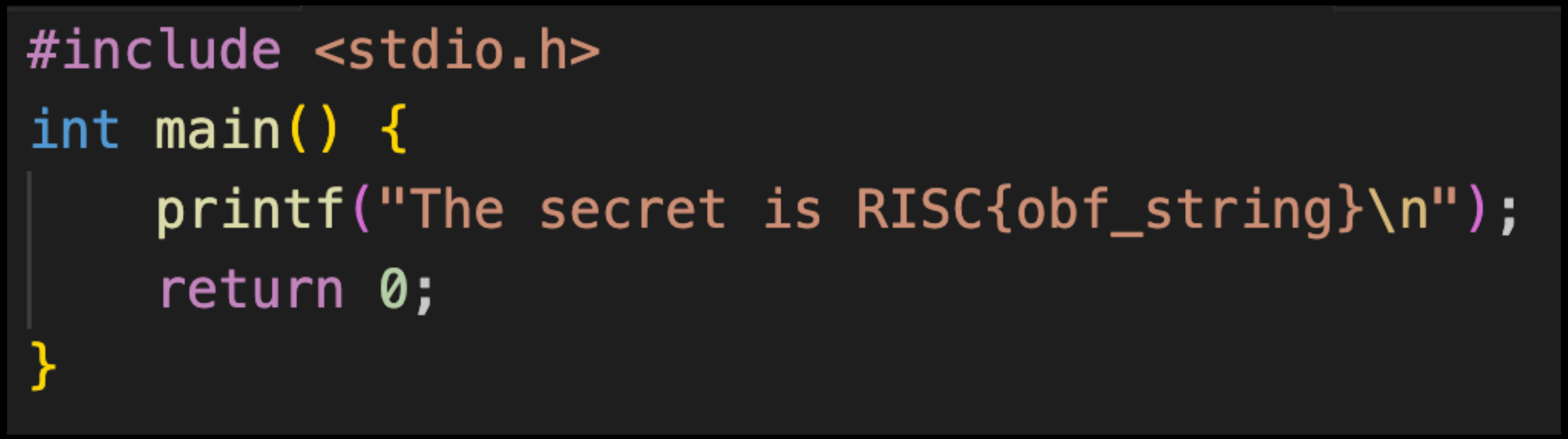
... how? - string encryption

- Encrypt all strings, and only decrypt when that string is used

Obfuscaters...

... how? - string encryption

- Encrypt all strings, and only decrypt when that string is used

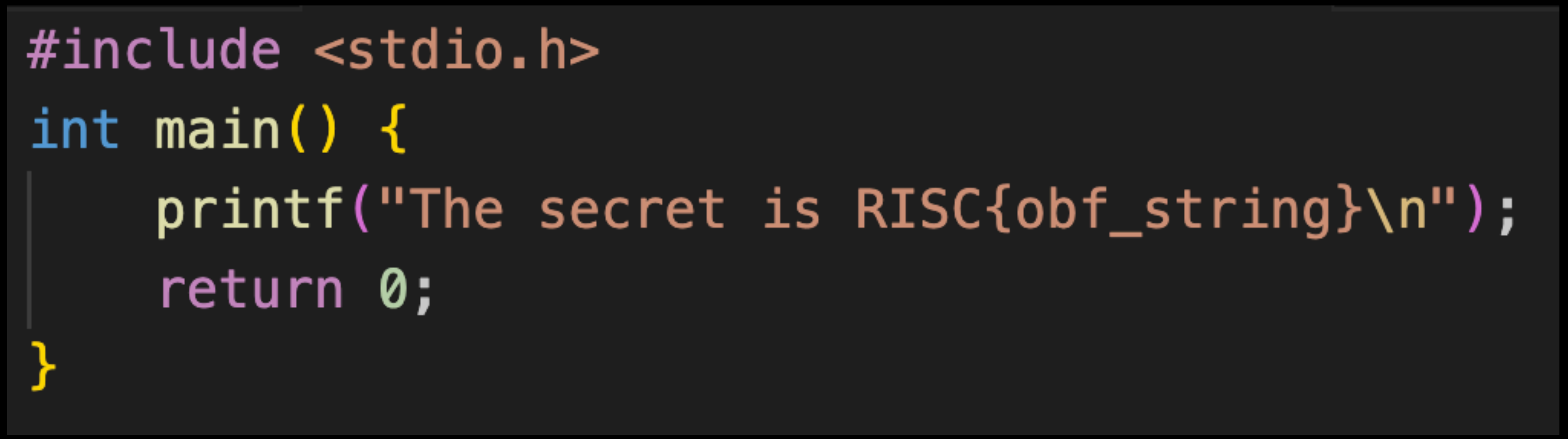


```c
#include <stdio.h>
int main() {
    printf("The secret is RISC{obf_string}\n");
    return 0;
}
```

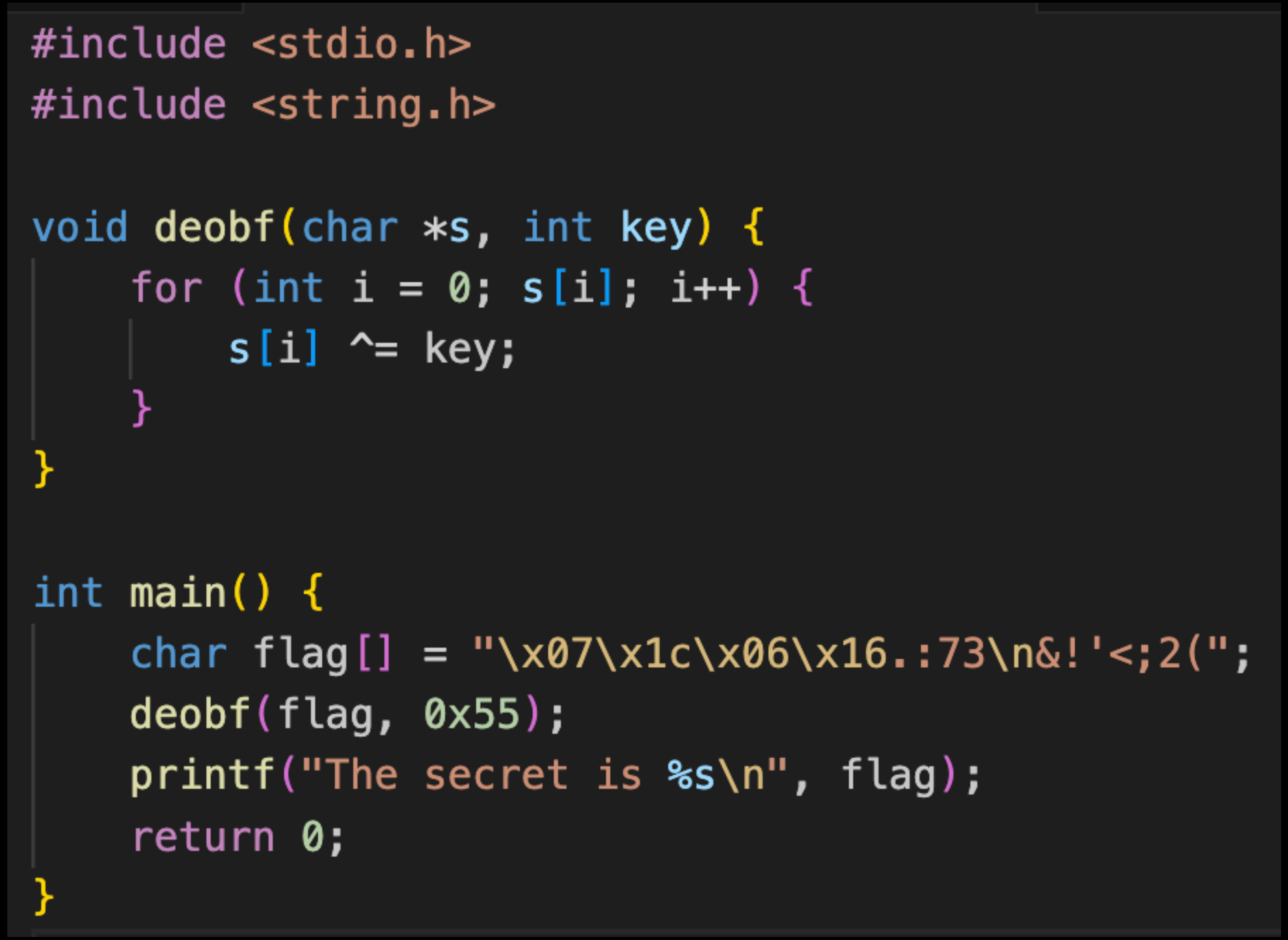
Obfuscaters...

... how? - string encryption

- Encrypt all strings, and only decrypt when that string is used



```c
#include <stdio.h>
int main() {
    printf("The secret is RISC{obf_string}\n");
    return 0;
}
```

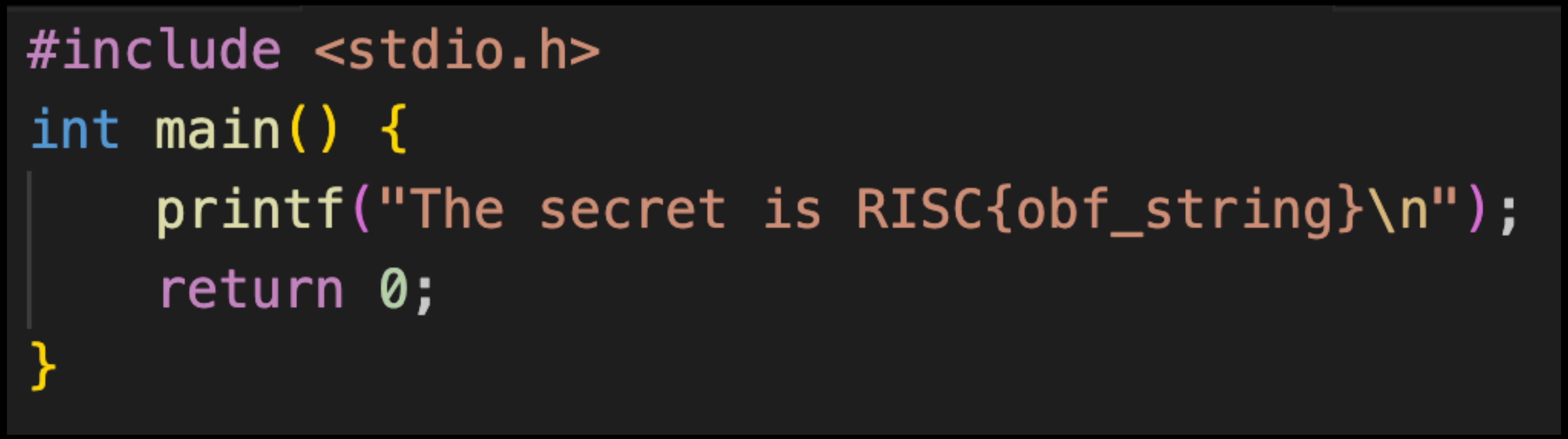


```c
#include <stdio.h>
#include <string.h>

void deobf(char *s, int key) {
    for (int i = 0; s[i]; i++) {
        s[i] ^= key;
    }
}

int main() {
    char flag[] = "\x07\x1c\x06\x16.:73\n&!'<;2(";
    deobf(flag, 0x55);
    printf("The secret is %s\n", flag);
    return 0;
}
```

Obfuscaters...

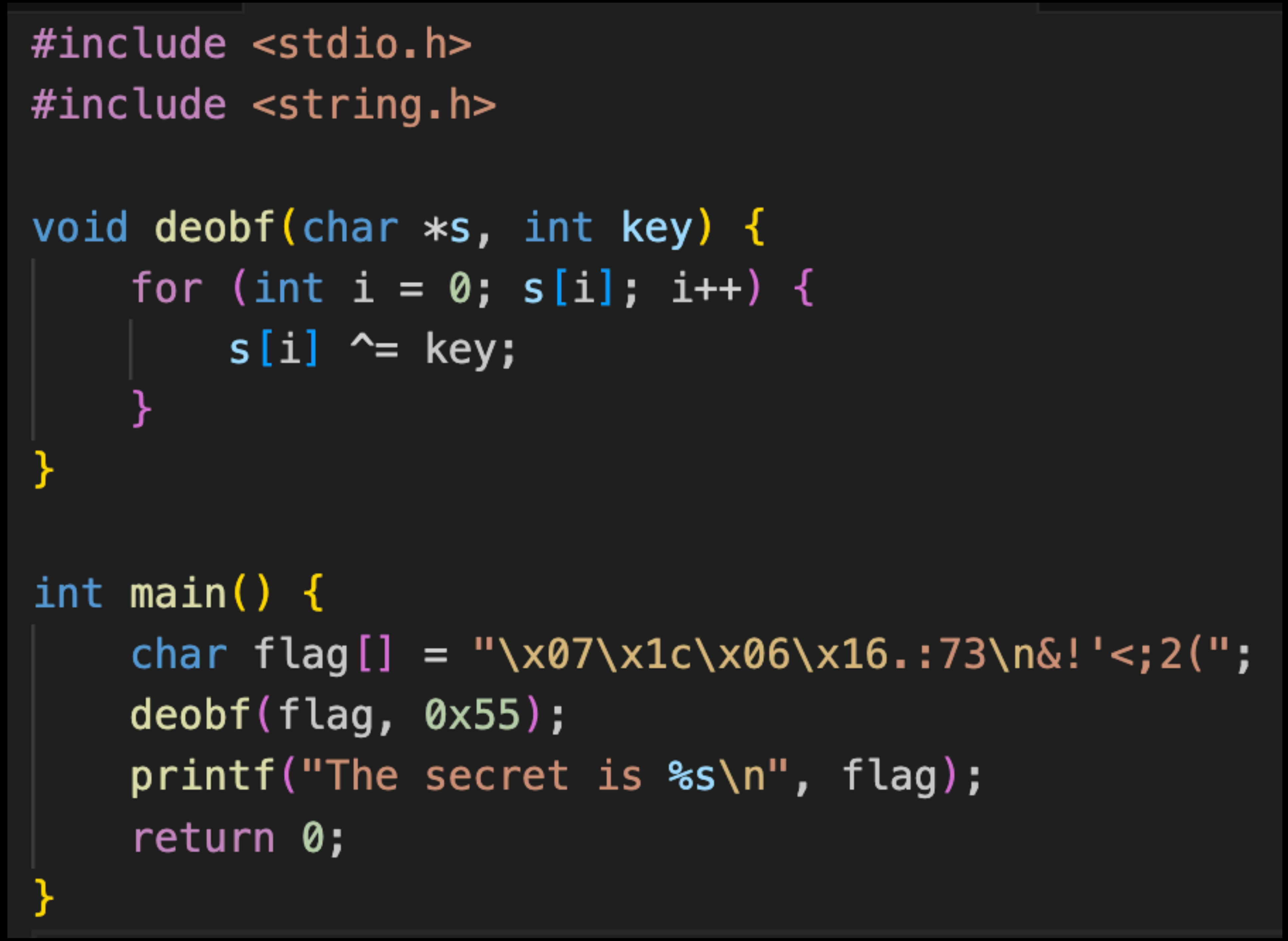
... how? - string encryption

- Encrypt all strings, and only decrypt when that string is used



```c
#include <stdio.h>
int main() {
    printf("The secret is RISC{obf_string}\n");
    return 0;
}
```

Both will print the same thing

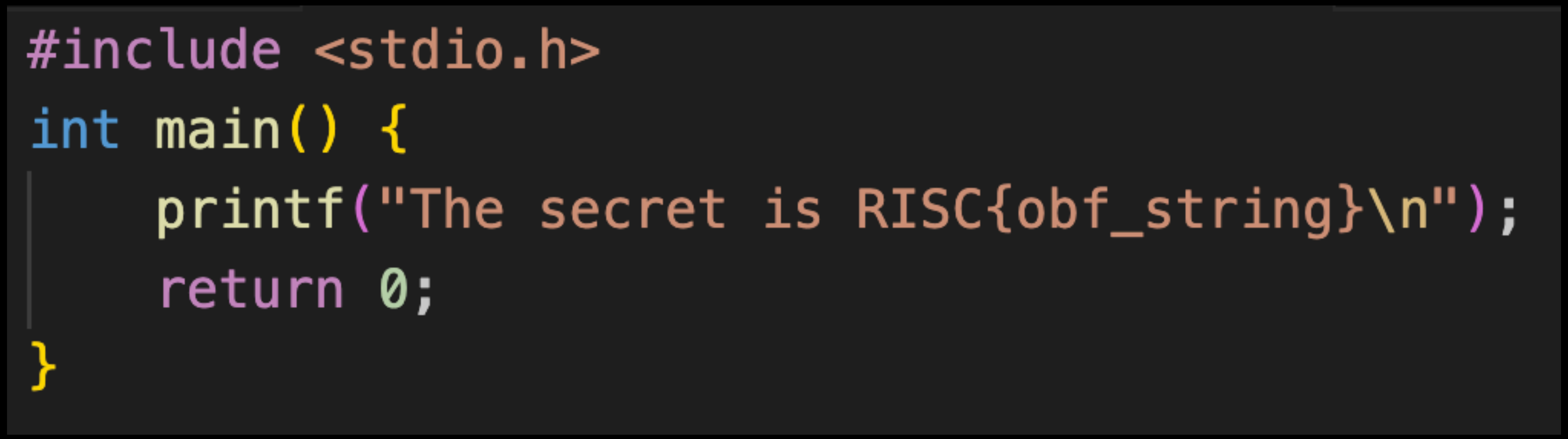


```c
#include <stdio.h>
#include <string.h>

void deobf(char *s, int key) {
    for (int i = 0; s[i]; i++) {
        s[i] ^= key;
    }
}

int main() {
    char flag[] = "\x07\x1c\x06\x16.:73\n&!'<;2(";
    deobf(flag, 0x55);
    printf("The secret is %s\n", flag);
    return 0;
}
```

Obfuscaters...

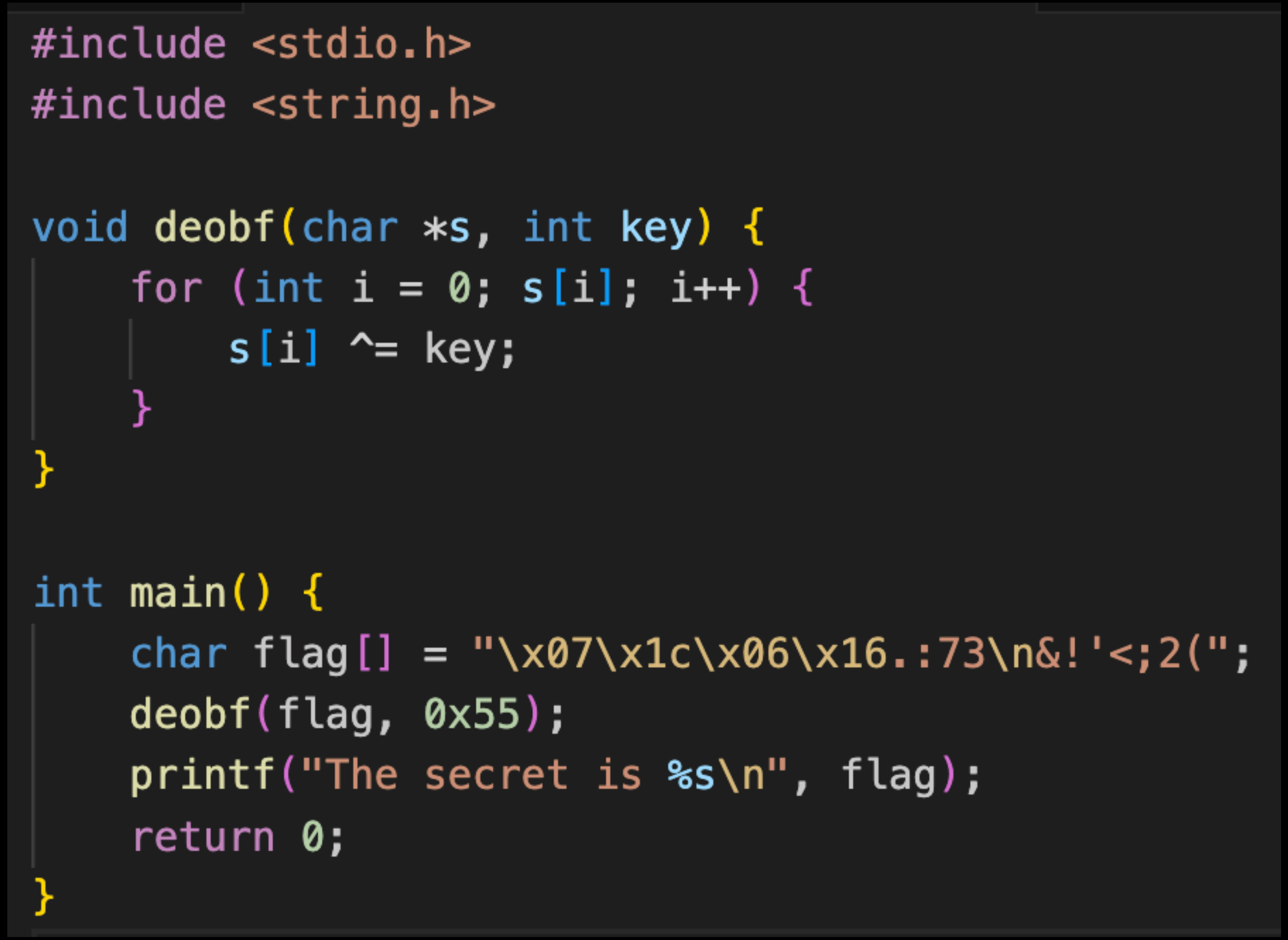
... how? - string encryption

- Encrypt all strings, and only decrypt when that string is used



```c
#include <stdio.h>
int main() {
    printf("The secret is RISC{obf_string}\n");
    return 0;
}
```

Both will print the same thing



```c
#include <stdio.h>
#include <string.h>

void deobf(char *s, int key) {
    for (int i = 0; s[i]; i++) {
        s[i] ^= key;
    }
}

int main() {
    char flag[] = "\x07\x1c\x06\x16.:73\n&!'<;2(";
    deobf(flag, 0x55);
    printf("The secret is %s\n", flag);
    return 0;
}
```

But this is harder
to reason about

Obfuscaters...

... how?

- Non exhaustive list, of course

Obfuscaters...

... how?

- Non exhaustive list, of course
- Can range from simple string encryption…

Obfuscaters...

... how?

- Non exhaustive list, of course
- Can range from simple string encryption…
- … to reducing the entire program down to a single instruction (MOV)

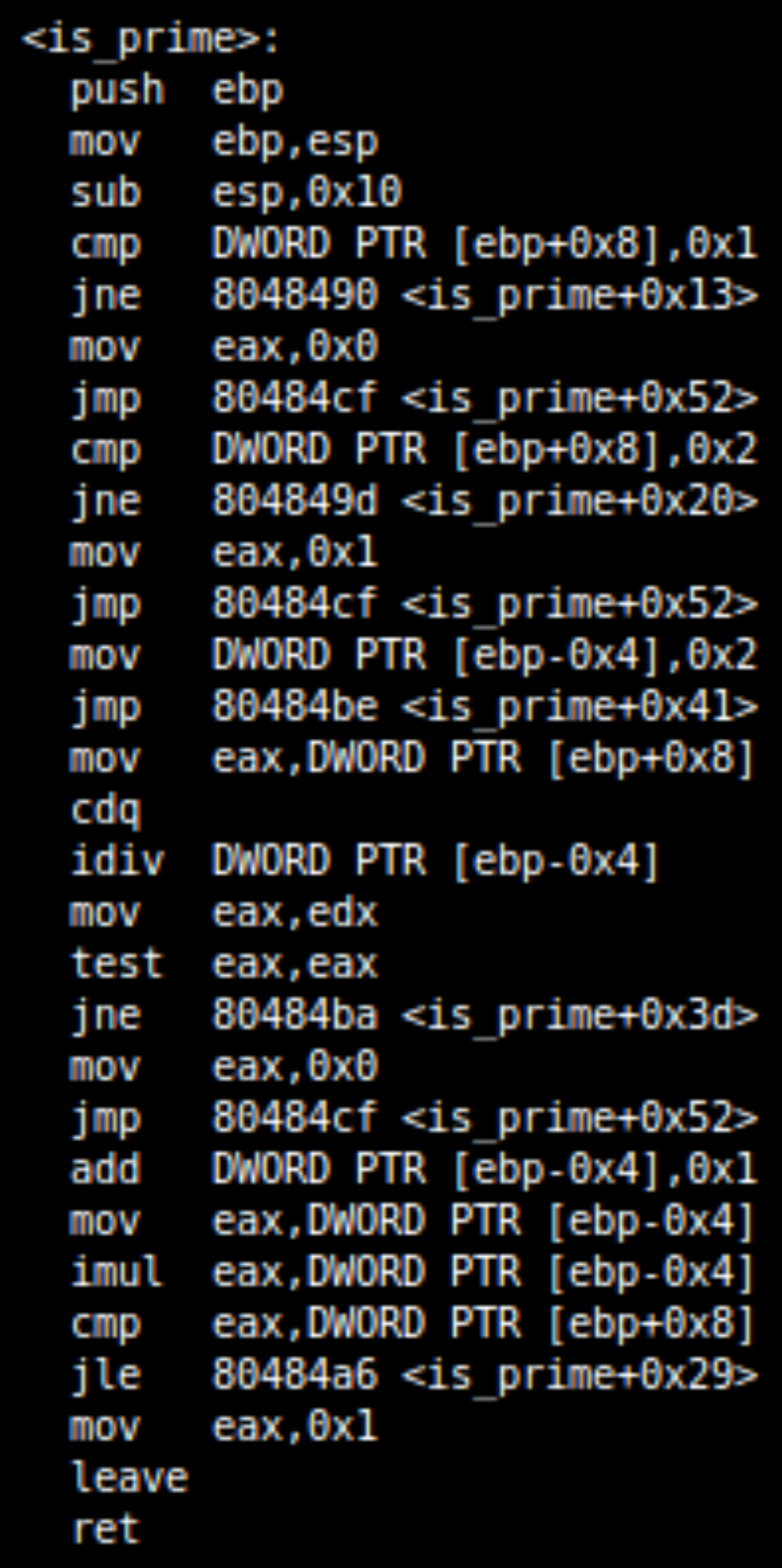
Obfuscaters...

... how?

- N

- C encryption…

- … am down to a single instruction (MOV)

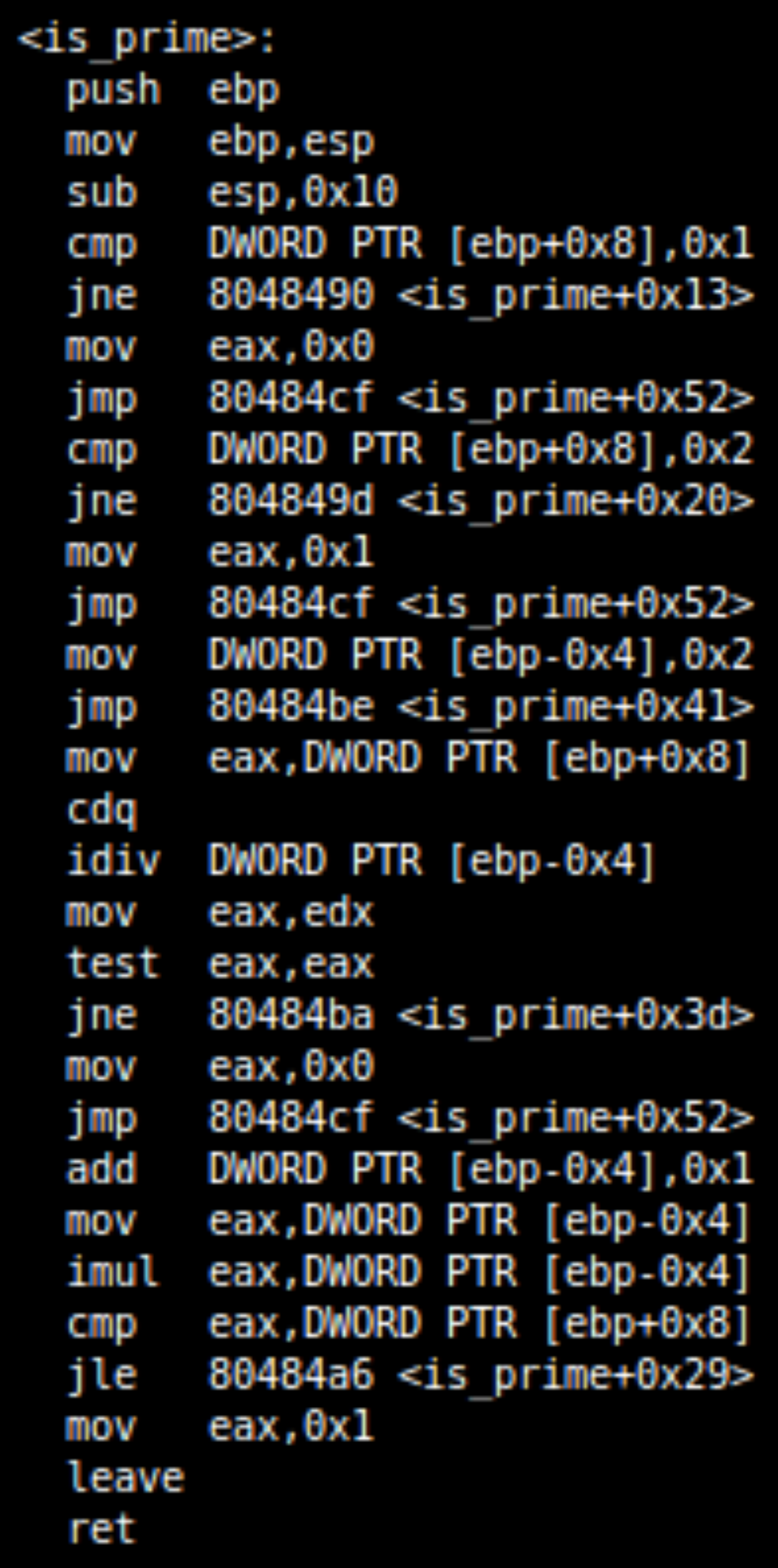


```
<is_prime>:
  push  ebp
  mov   ebp,esp
  sub   esp,0x10
  cmp   DWORD PTR [ebp+0x8],0x1
  jne   8048490 <is_prime+0x13>
  mov   eax,0x0
  jmp   80484cf <is_prime+0x52>
  cmp   DWORD PTR [ebp+0x8],0x2
  jne   804849d <is_prime+0x20>
  mov   eax,0x1
  jmp   80484cf <is_prime+0x52>
  mov   DWORD PTR [ebp-0x4],0x2
  jmp   80484be <is_prime+0x41>
  mov   eax,DWORD PTR [ebp+0x8]
  cdq
  idiv  DWORD PTR [ebp-0x4]
  mov   eax,edx
  test  eax,eax
  jne   80484ba <is_prime+0x3d>
  mov   eax,0x0
  jmp   80484cf <is_prime+0x52>
  add   DWORD PTR [ebp-0x4],0x1
  mov   eax,DWORD PTR [ebp-0x4]
  imul  eax,DWORD PTR [ebp-0x4]
  cmp   eax,DWORD PTR [ebp+0x8]
  jle   80484a6 <is_prime+0x29>
  mov   eax,0x1
  leave
  ret
```

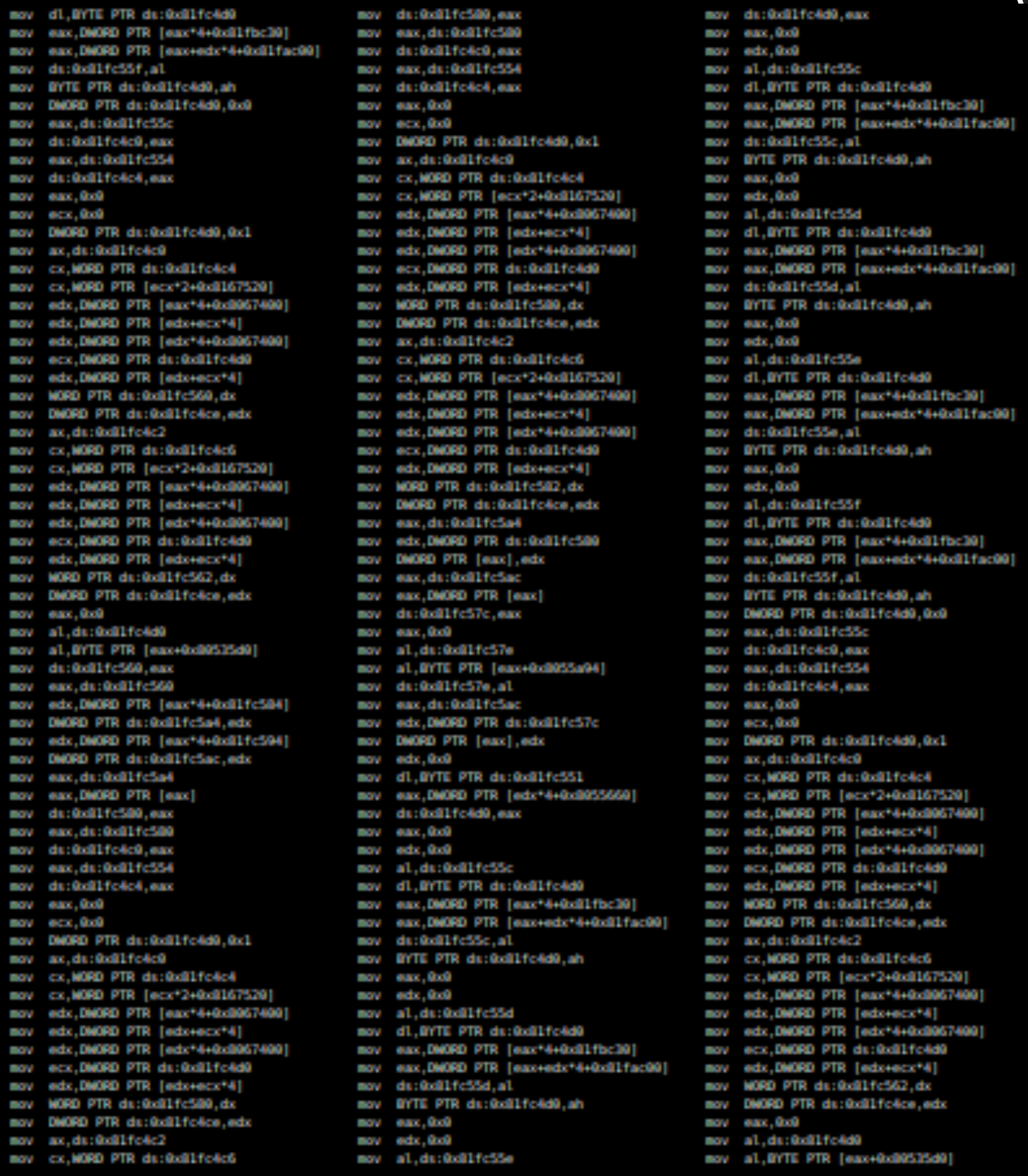
Obfuscaters...

... how?

- N
- C encrypti
- … am dov



```
<is_prime>:
  push  ebp
  mov   ebp,esp
  sub   esp,0x10
  cmp   DWORD PTR [ebp+0x8],0x1
  jne   8048490 <is_prime+0x13>
  mov   eax,0x0
  jmp   80484cf <is_prime+0x52>
  cmp   DWORD PTR [ebp+0x8],0x2
  jne   804849d <is_prime+0x20>
  mov   eax,0x1
  jmp   80484cf <is_prime+0x52>
  mov   DWORD PTR [ebp-0x4],0x2
  jmp   80484be <is_prime+0x41>
  mov   eax,DWORD PTR [ebp+0x8]
  cdq
  idiv  DWORD PTR [ebp-0x4]
  mov   eax,edx
  test  eax,eax
  jne   80484ba <is_prime+0x3d>
  mov   eax,0x0
  jmp   80484cf <is_prime+0x52>
  add   DWORD PTR [ebp-0x4],0x1
  mov   eax,DWORD PTR [ebp-0x4]
  imul  eax,DWORD PTR [ebp-0x4]
  cmp   eax,DWORD PTR [ebp+0x8]
  jle   80484a6 <is_prime+0x29>
  mov   eax,0x1
  leave
  ret
```



```
mov  dl,BYTE PTR ds:0x81fc4d0
mov  eax,DWORD PTR [eax*4+0x81fbc30]
mov  eax,DWORD PTR [eax+edx*4+0x81fac00]
mov  ds:0x81fc55f,al
mov  BYTE PTR ds:0x81fc4d0,ah
mov  DWORD PTR ds:0x81fc4d0,0x0
mov  eax,ds:0x81fc55c
mov  ds:0x81fc4c0,eax
mov  eax,ds:0x81fc554
mov  ds:0x81fc4c4,eax
mov  eax,0x0
mov  ecx,0x0
mov  DWORD PTR ds:0x81fc4d0,0x1
mov  ax,ds:0x81fc4c0
mov  cx,WORD PTR ds:0x81fc4c4
mov  cx,WORD PTR [ecx*2+0x8167520]
mov  edx,DWORD PTR [eax*4+0x8067400]
mov  edx,DWORD PTR [edx+ecx*4]
mov  edx,DWORD PTR [edx*4+0x8067400]
mov  ecx,DWORD PTR ds:0x81fc4d0
mov  edx,DWORD PTR [edx+ecx*4]
mov  WORD PTR ds:0x81fc560,dx
mov  DWORD PTR ds:0x81fc4ce,edx
mov  ax,ds:0x81fc4c2
mov  cx,WORD PTR ds:0x81fc4c6
mov  cx,WORD PTR [ecx*2+0x8167520]
mov  edx,DWORD PTR [eax*4+0x8067400]
mov  edx,DWORD PTR [edx+ecx*4]
mov  edx,DWORD PTR [edx*4+0x8067400]
mov  ecx,DWORD PTR ds:0x81fc4d0
mov  edx,DWORD PTR [edx+ecx*4]
mov  WORD PTR ds:0x81fc562,dx
mov  DWORD PTR ds:0x81fc4ce,edx
mov  eax,0x0
mov  al,ds:0x81fc4d0
mov  al,BYTE PTR [eax+0x80535d0]
mov  ds:0x81fc560,eax
mov  eax,ds:0x81fc560
mov  edx,DWORD PTR [eax*4+0x81fc584]
mov  DWORD PTR ds:0x81fc5a4,edx
mov  edx,DWORD PTR [eax*4+0x81fc594]
mov  DWORD PTR ds:0x81fc5ac,edx
mov  eax,ds:0x81fc5a4
mov  eax,DWORD PTR [eax]
mov  ds:0x81fc580,eax
mov  eax,ds:0x81fc580
mov  ds:0x81fc4c0,eax
mov  eax,ds:0x81fc554
mov  ds:0x81fc4c4,eax
mov  eax,0x0
mov  ecx,0x0
mov  DWORD PTR ds:0x81fc4d0,0x1
mov  ax,ds:0x81fc4c0
mov  cx,WORD PTR ds:0x81fc4c4
mov  cx,WORD PTR [ecx*2+0x8167520]
mov  edx,DWORD PTR [eax*4+0x8067400]
mov  edx,DWORD PTR [edx+ecx*4]
mov  edx,DWORD PTR [edx*4+0x8067400]
mov  ecx,DWORD PTR ds:0x81fc4d0
mov  edx,DWORD PTR [edx+ecx*4]
mov  WORD PTR ds:0x81fc580,dx
mov  DWORD PTR ds:0x81fc4ce,edx
mov  ax,ds:0x81fc4c2
mov  cx,WORD PTR ds:0x81fc4c6
```

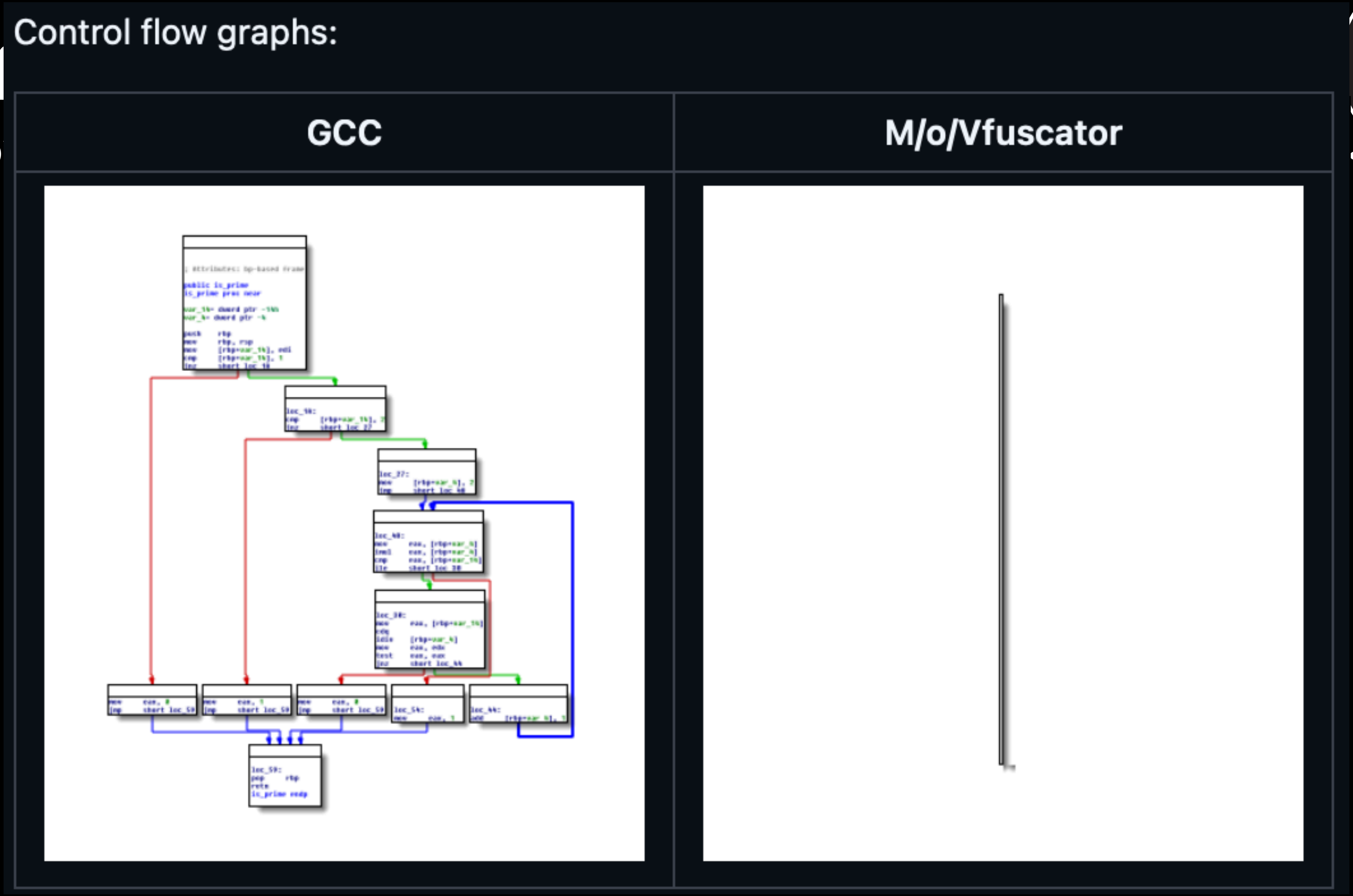
```
mov  ds:0x81fc580,eax
mov  eax,ds:0x81fc580
mov  ds:0x81fc4c0,eax
mov  eax,ds:0x81fc554
mov  ds:0x81fc4c4,eax
mov  eax,0x0
mov  ecx,0x0
mov  DWORD PTR ds:0x81fc4d0,0x1
mov  ax,ds:0x81fc4c0
mov  cx,WORD PTR ds:0x81fc4c4
mov  cx,WORD PTR [ecx*2+0x8167520]
mov  edx,DWORD PTR [eax*4+0x8067400]
mov  edx,DWORD PTR [edx+ecx*4]
mov  edx,DWORD PTR [edx*4+0x8067400]
mov  ecx,DWORD PTR ds:0x81fc4d0
mov  edx,DWORD PTR [edx+ecx*4]
mov  WORD PTR ds:0x81fc580,dx
mov  DWORD PTR ds:0x81fc4ce,edx
mov  ax,ds:0x81fc4c2
mov  cx,WORD PTR ds:0x81fc4c6
mov  cx,WORD PTR [ecx*2+0x8167520]
mov  edx,DWORD PTR [eax*4+0x8067400]
mov  edx,DWORD PTR [edx+ecx*4]
mov  edx,DWORD PTR [edx*4+0x8067400]
mov  ecx,DWORD PTR ds:0x81fc4d0
mov  edx,DWORD PTR [edx+ecx*4]
mov  WORD PTR ds:0x81fc582,dx
mov  DWORD PTR ds:0x81fc4ce,edx
mov  eax,ds:0x81fc5a4
mov  edx,DWORD PTR ds:0x81fc580
mov  DWORD PTR [eax],edx
mov  eax,ds:0x81fc5ac
mov  eax,DWORD PTR [eax]
mov  ds:0x81fc57c,eax
mov  eax,0x0
mov  al,ds:0x81fc57e
mov  al,BYTE PTR [eax+0x8055a94]
mov  ds:0x81fc57e,al
mov  eax,ds:0x81fc5ac
mov  edx,DWORD PTR ds:0x81fc57c
mov  DWORD PTR [eax],edx
mov  edx,0x0
mov  dl,BYTE PTR ds:0x81fc551
mov  eax,DWORD PTR [edx*4+0x8055660]
mov  ds:0x81fc4d0,eax
mov  eax,0x0
mov  edx,0x0
mov  al,ds:0x81fc55c
mov  dl,BYTE PTR ds:0x81fc4d0
mov  eax,DWORD PTR [eax*4+0x81fbc30]
mov  eax,DWORD PTR [eax+edx*4+0x81fac00]
mov  ds:0x81fc55c,al
mov  BYTE PTR ds:0x81fc4d0,ah
mov  eax,0x0
mov  edx,0x0
mov  al,ds:0x81fc55d
mov  dl,BYTE PTR ds:0x81fc4d0
mov  eax,DWORD PTR [eax*4+0x81fbc30]
mov  eax,DWORD PTR [eax+edx*4+0x81fac00]
mov  ds:0x81fc55d,al
mov  BYTE PTR ds:0x81fc4d0,ah
mov  eax,0x0
mov  edx,0x0
mov  al,ds:0x81fc55e
```

```
mov  ds:0x81fc4d0,eax
mov  eax,0x0
mov  edx,0x0
mov  al,ds:0x81fc55c
mov  dl,BYTE PTR ds:0x81fc4d0
mov  eax,DWORD PTR [eax*4+0x81fbc30]
mov  eax,DWORD PTR [eax+edx*4+0x81fac00]
mov  ds:0x81fc55c,al
mov  BYTE PTR ds:0x81fc4d0,ah
mov  eax,0x0
mov  edx,0x0
mov  al,ds:0x81fc55d
mov  dl,BYTE PTR ds:0x81fc4d0
mov  eax,DWORD PTR [eax*4+0x81fbc30]
mov  eax,DWORD PTR [eax+edx*4+0x81fac00]
mov  ds:0x81fc55d,al
mov  BYTE PTR ds:0x81fc4d0,ah
mov  eax,0x0
mov  edx,0x0
mov  al,ds:0x81fc55e
mov  dl,BYTE PTR ds:0x81fc4d0
mov  eax,DWORD PTR [eax*4+0x81fbc30]
mov  eax,DWORD PTR [eax+edx*4+0x81fac00]
mov  ds:0x81fc55e,al
mov  BYTE PTR ds:0x81fc4d0,ah
mov  eax,0x0
mov  edx,0x0
mov  al,ds:0x81fc55f
mov  dl,BYTE PTR ds:0x81fc4d0
mov  eax,DWORD PTR [eax*4+0x81fbc30]
mov  eax,DWORD PTR [eax+edx*4+0x81fac00]
mov  ds:0x81fc55f,al
mov  BYTE PTR ds:0x81fc4d0,ah
mov  DWORD PTR ds:0x81fc4d0,0x0
mov  eax,ds:0x81fc55c
mov  ds:0x81fc4c0,eax
mov  eax,ds:0x81fc554
mov  ds:0x81fc4c4,eax
mov  eax,0x0
mov  ecx,0x0
mov  DWORD PTR ds:0x81fc4d0,0x1
mov  ax,ds:0x81fc4c0
mov  cx,WORD PTR ds:0x81fc4c4
mov  cx,WORD PTR [ecx*2+0x8167520]
mov  edx,DWORD PTR [eax*4+0x8067400]
mov  edx,DWORD PTR [edx+ecx*4]
mov  edx,DWORD PTR [edx*4+0x8067400]
mov  ecx,DWORD PTR ds:0x81fc4d0
mov  edx,DWORD PTR [edx+ecx*4]
mov  WORD PTR ds:0x81fc560,dx
mov  DWORD PTR ds:0x81fc4ce,edx
mov  ax,ds:0x81fc4c2
mov  cx,WORD PTR ds:0x81fc4c6
mov  cx,WORD PTR [ecx*2+0x8167520]
mov  edx,DWORD PTR [eax*4+0x8067400]
mov  edx,DWORD PTR [edx+ecx*4]
mov  edx,DWORD PTR [edx*4+0x8067400]
mov  ecx,DWORD PTR ds:0x81fc4d0
mov  edx,DWORD PTR [edx+ecx*4]
mov  WORD PTR ds:0x81fc562,dx
mov  DWORD PTR ds:0x81fc4ce,edx
mov  eax,0x0
mov  al,ds:0x81fc4d0
mov  al,BYTE PTR [eax+0x80535d0]
```

Obf

... ho

- N
- C
- ...



Control flow graphs:

GCC	M/o/Vfuscator

Obfuscaters...

... how?

- Non exhaustive list, of course

Obfuscaters...

... how?

- Non exhaustive list, of course
- Can range from simple string encryption…

Obfuscaters...

... how?

- Non exhaustive list, of course
- Can range from simple string encryption…
- … to reducing the entire program down to a single instruction (MOV)

Obfuscaters...

... how?

- Non exhaustive list, of course
- Can range from simple string encryption…
- … to reducing the entire program down to a single instruction (MOV)
    - M/o/Vfuscator - xoreaxeax

Bonus: SMT solvers

- Many reversing problems boil down to “Find input X that satisfies constraints Y”

Bonus: SMT solvers

- Many reversing problems boil down to “Find input X that satisfies constraints Y”
- Constraints can be reduced down into boolean algebra

Bonus: SMT solvers

- Many reversing problems boil down to “Find input X that satisfies constraints Y”
- Constraints can be reduced down into boolean algebra
- SMT solvers find solutions to these equations

Bonus: SMT solvers

- Many reversing problems boil down to “Find input X that satisfies constraints Y”
- Constraints can be reduced down into boolean algebra
- SMT solvers find solutions to these equations
    - i.e., finding X such that Y

Bonus: SMT solvers

- Many reversing problems boil down to “Find input X that satisfies constraints Y”
- Constraints can be reduced down into boolean algebra
- SMT solvers find solutions to these equations
    - i.e., finding X such that Y
- Simple example:

Bonus: SMT solvers

- Many reversing problems boil down to “Find input X that satisfies constraints Y”
- Constraints can be reduced down into boolean algebra
- SMT solvers find solutions to these equations
  - i.e., finding X such that Y
- Simple example:

```
If (a * 3 + b == 42 && (a ^ b) == 7) {
    win();
}
```

Bonus: SMT solvers

- Many reversing problems boil down to “Find input X that satisfies constraints Y”
- Constraints can be reduced down into boolean algebra
- SMT solvers find solutions to these equations
  - i.e., finding X such that Y
- Simple example:

```
If (a * 3 + b == 42 && (a ^ b) == 7) {
    win();
}
```

Exercise for the reader :)

Questions?

Tools that you should go and learn

- `strings`
- `Binary Ninja`
- `objdump`
- `binwalk`
- `file`
- `gdb`
- `nc`

Feedback

https://forms.office.com/r/3L9rMBd2iQ

ctf.urisc.club