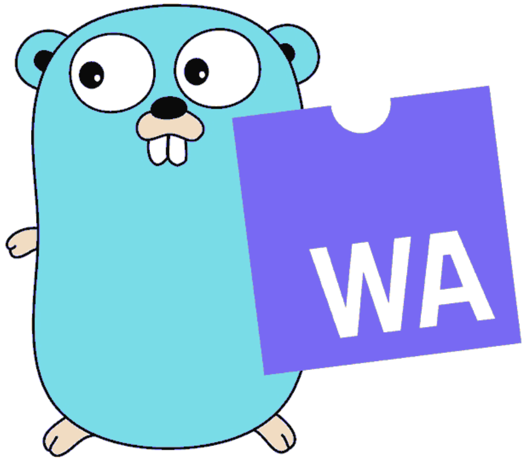
The state of WebAssembly in Go

Matias Insaurralde <matias@insaurral.de>

Golang Estonia Meetup - Feb 5, 2020

Tallinn, Estonia



Intro

- 25 years old, living in Asunción, Paraguay 🇵🇾
- Proud Estonian e-resident 🇪🇪
- Engineer at Tyk (we're building an open source API gateway!).
- I love FOSS and security topics.
- Hobbies: meet random people, improvise music, eat pasta.

Agenda

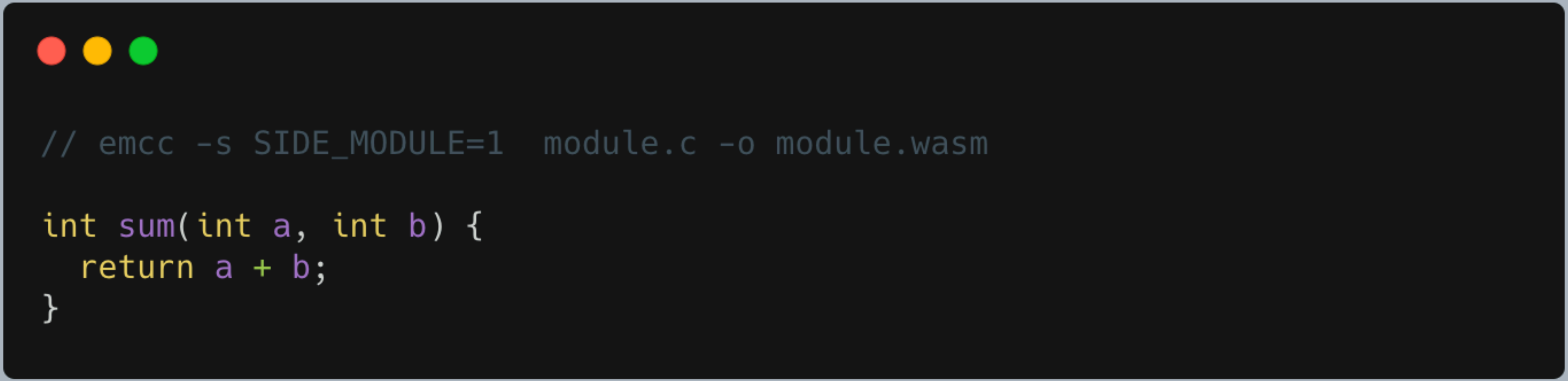
- What's WebAssembly?
- GOARCH=wasm
- WebAssembly VMs in Go
- Performance and interop potential
- What's next?

What’s WebAssembly?

- Portable.
- Linear memory.
- Sandboxed (more on this later…)
- Browser support.

What’s WebAssembly?

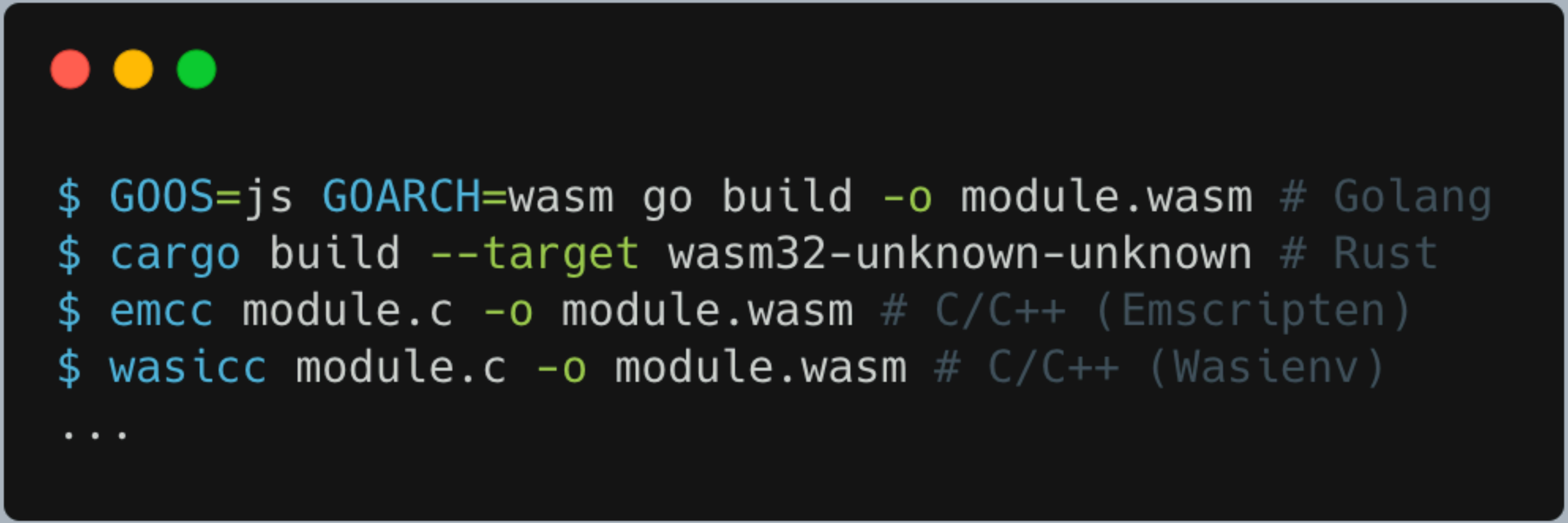
(i) How to write a WASM module?



```c
// emcc -s SIDE_MODULE=1  module.c -o module.wasm

int sum(int a, int b) {
  return a + b;
}
```

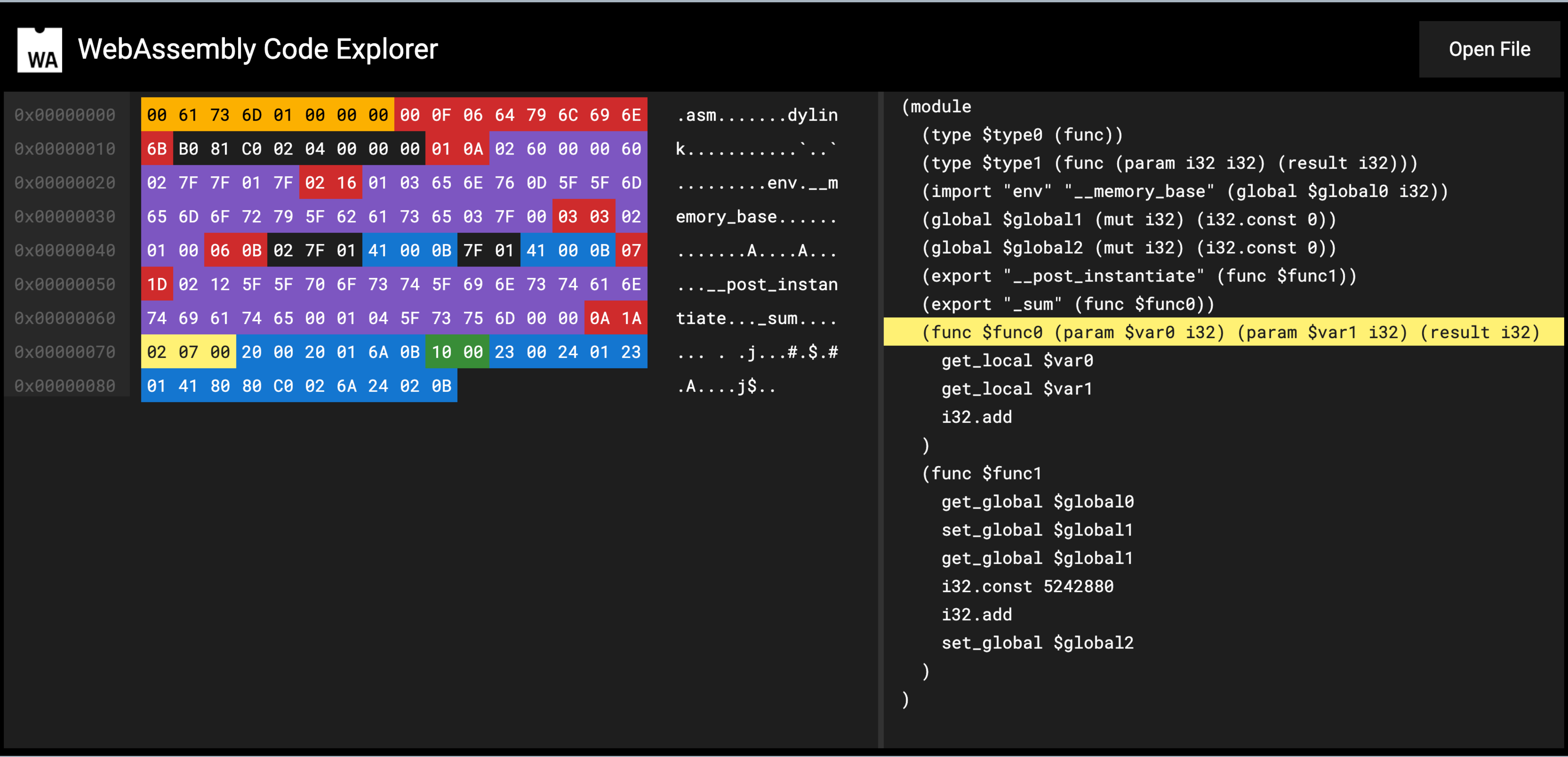
What’s WebAssembly?

(ii) How to build WASM modules?



```
$ GOOS=js GOARCH=wasm go build -o module.wasm # Golang
$ cargo build --target wasm32-unknown-unknown # Rust
$ emcc module.c -o module.wasm # C/C++ (Emscripten)
$ wasicc module.c -o module.wasm # C/C++ (Wasienv)
...
```

(iii) How does a WebAssembly look like?



```
0x00000000  00 61 73 6D 01 00 00 00 00 0F 06 64 79 6C 69 6E   .asm.......dylin
0x00000010  6B B0 81 C0 02 04 00 00 00 01 0A 02 60 00 00 60   k...........`..`
0x00000020  02 7F 7F 01 7F 02 16 01 03 65 6E 76 0D 5F 5F 6D   .........env.__m
0x00000030  65 6D 6F 72 79 5F 62 61 73 65 03 7F 00 03 03 02   emory_base......
0x00000040  01 00 06 0B 02 7F 01 41 00 0B 7F 01 41 00 0B 07   .......A....A...
0x00000050  1D 02 12 5F 5F 70 6F 73 74 5F 69 6E 73 74 61 6E   ....__post_instan
0x00000060  74 69 61 74 65 00 01 04 5F 73 75 6D 00 00 0A 1A   tiate...._sum....
0x00000070  02 07 00 20 00 20 01 6A 0B 10 00 23 00 24 01 23   ... . .j...#.$.#
0x00000080  01 41 80 80 C0 02 6A 24 02 0B                     .A....j$..
```

```
(module
  (type $type0 (func))
  (type $type1 (func (param i32 i32) (result i32)))
  (import "env" "__memory_base" (global $global0 i32))
  (global $global1 (mut i32) (i32.const 0))
  (global $global2 (mut i32) (i32.const 0))
  (export "__post_instantiate" (func $func1))
  (export "_sum" (func $func0))
  (func $func0 (param $var0 i32) (param $var1 i32) (result i32)
    get_local $var0
    get_local $var1
    i32.add
  )
  (func $func1
    get_global $global0
    set_global $global1
    get_global $global1
    i32.const 5242880
    i32.add
    set_global $global2
  )
)
```

What’s WebAssembly?

(iv) How to run WASM modules?

- Modern browsers support WASM.

- Many software communities are creating VMs for different languages.

What’s WebAssembly?

(v) Other interesting projects:

- TeaVM: Java Bytecode to WASM.

- Pyiodide: Python scientific stack in WASM.

- Lumen: compiler/runtime, Erlang -> WASM

- Blazor: .NET -> WASM, in the browser.

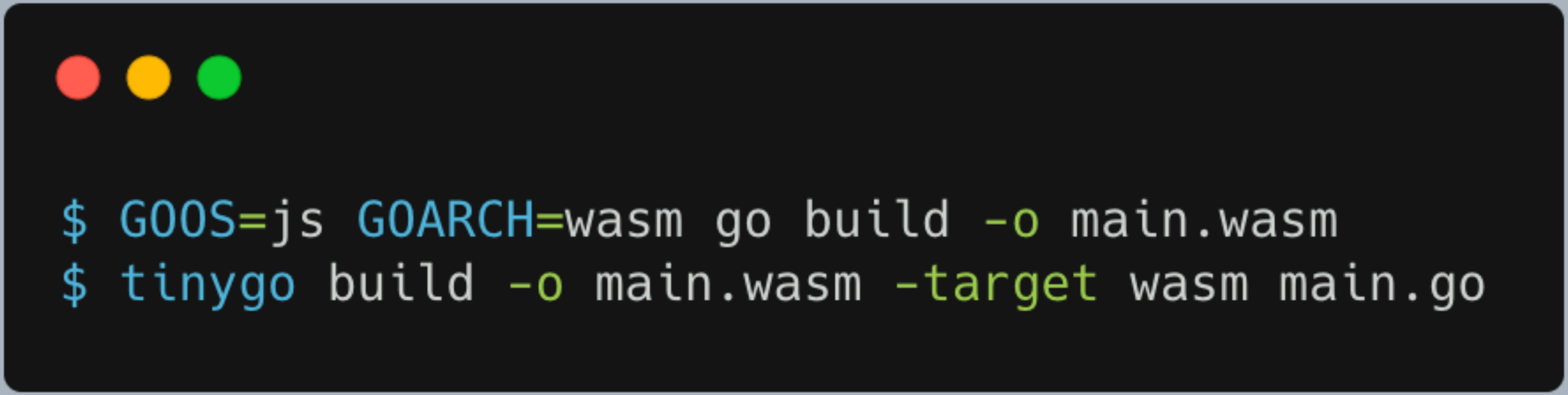
GOARCH=wasm

GOARCH=wasm

WebAssembly **target** support in Go:

- Go 1.11 and newer versions.

- TinyGo (minimal Go compiler focused on microcontrollers).



```
$ GOOS=js GOARCH=wasm go build -o main.wasm
$ tinygo build -o main.wasm -target wasm main.go
```

WebAssembly VMs in Go

WebAssembly VMs in Go

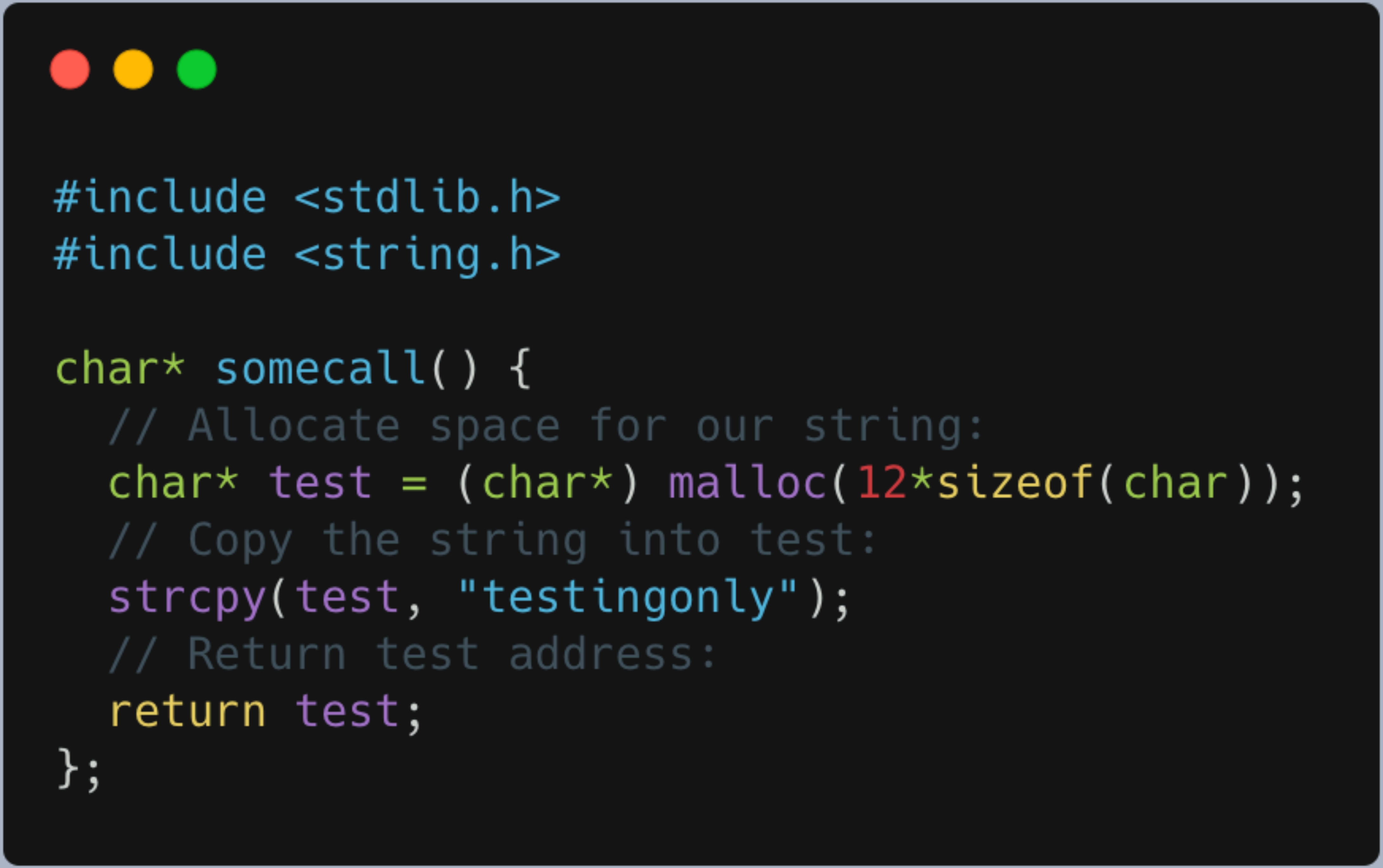
- wasmer (written in Rust, there's a Go package for it)
  https://github.com/wasmerio/wasmer

- life (written in Go, targeting decentralized applications)
  https://github.com/perlin-network/life

- go-interpreter/wagon (written in Go)
  https://github.com/go-interpreter/wagon

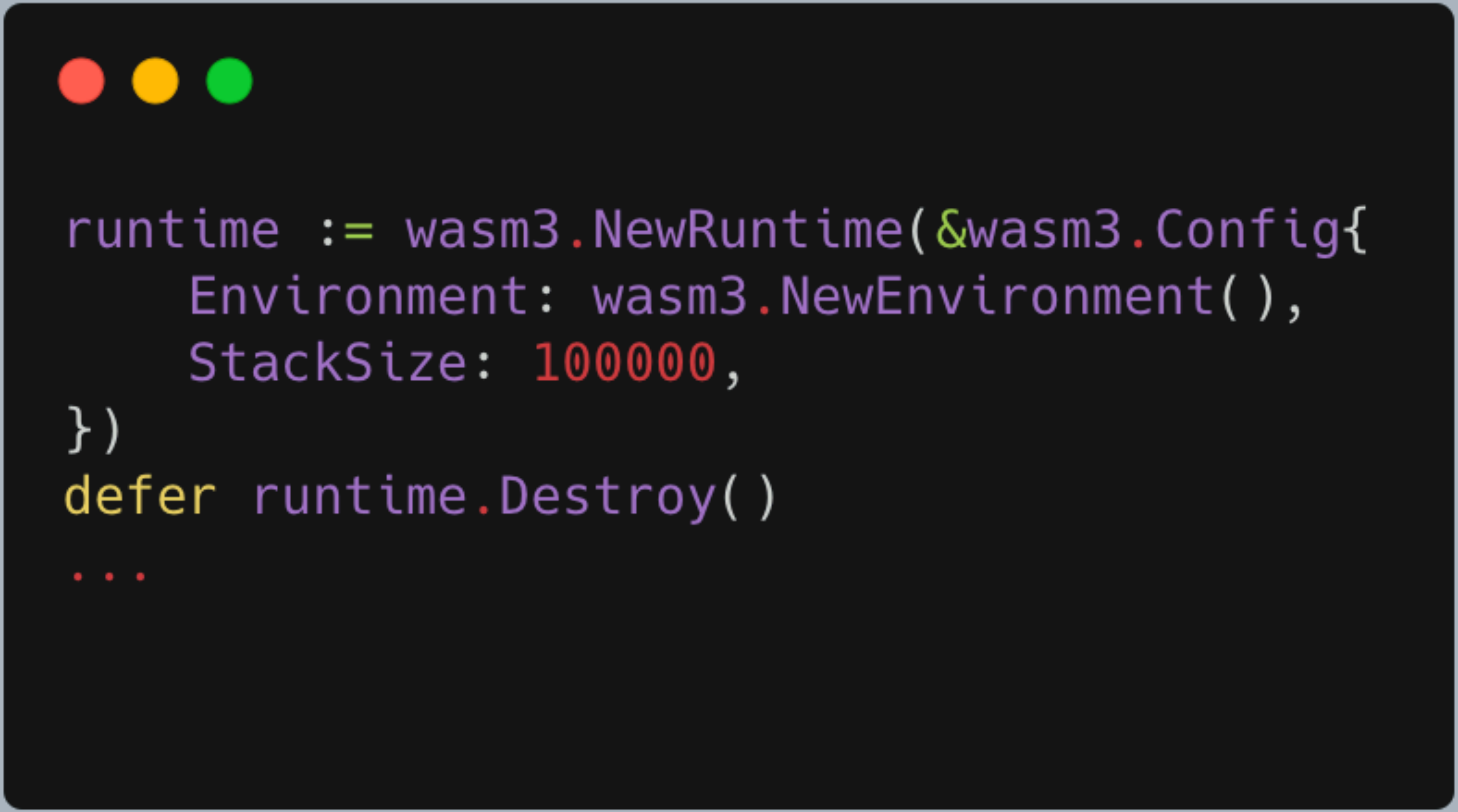
- go-wasm3 (written in C, I've started working on this Go
package last month)
  https://github.com/matiasinsaurralde/go-wasm3

walk-through



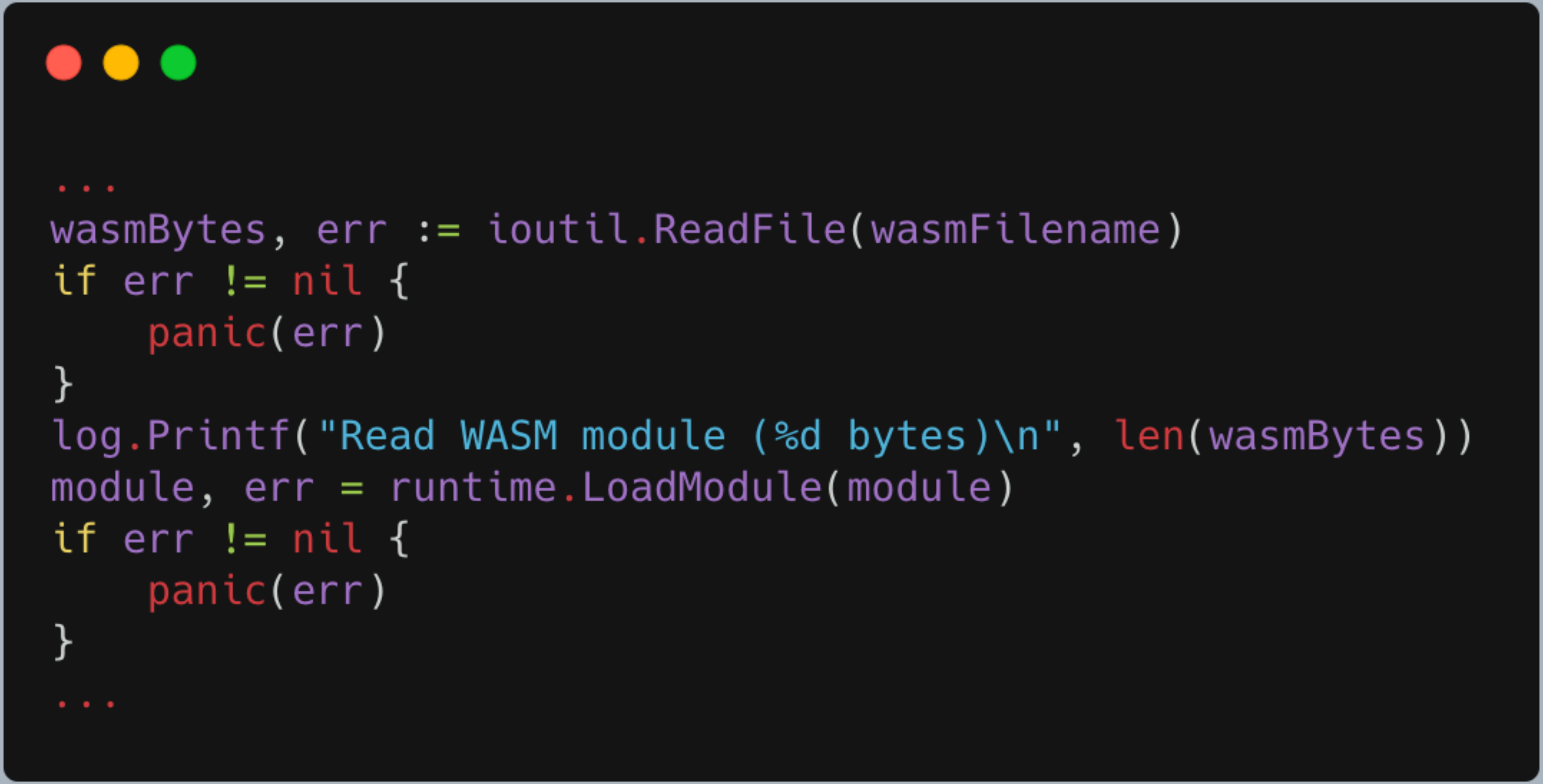
```c
#include <stdlib.h>
#include <string.h>

char* somecall() {
  // Allocate space for our string:
  char* test = (char*) malloc(12*sizeof(char));
  // Copy the string into test:
  strcpy(test, "testingonly");
  // Return test address:
  return test;
};
```

(i) Initialize the WASM runtime ✅



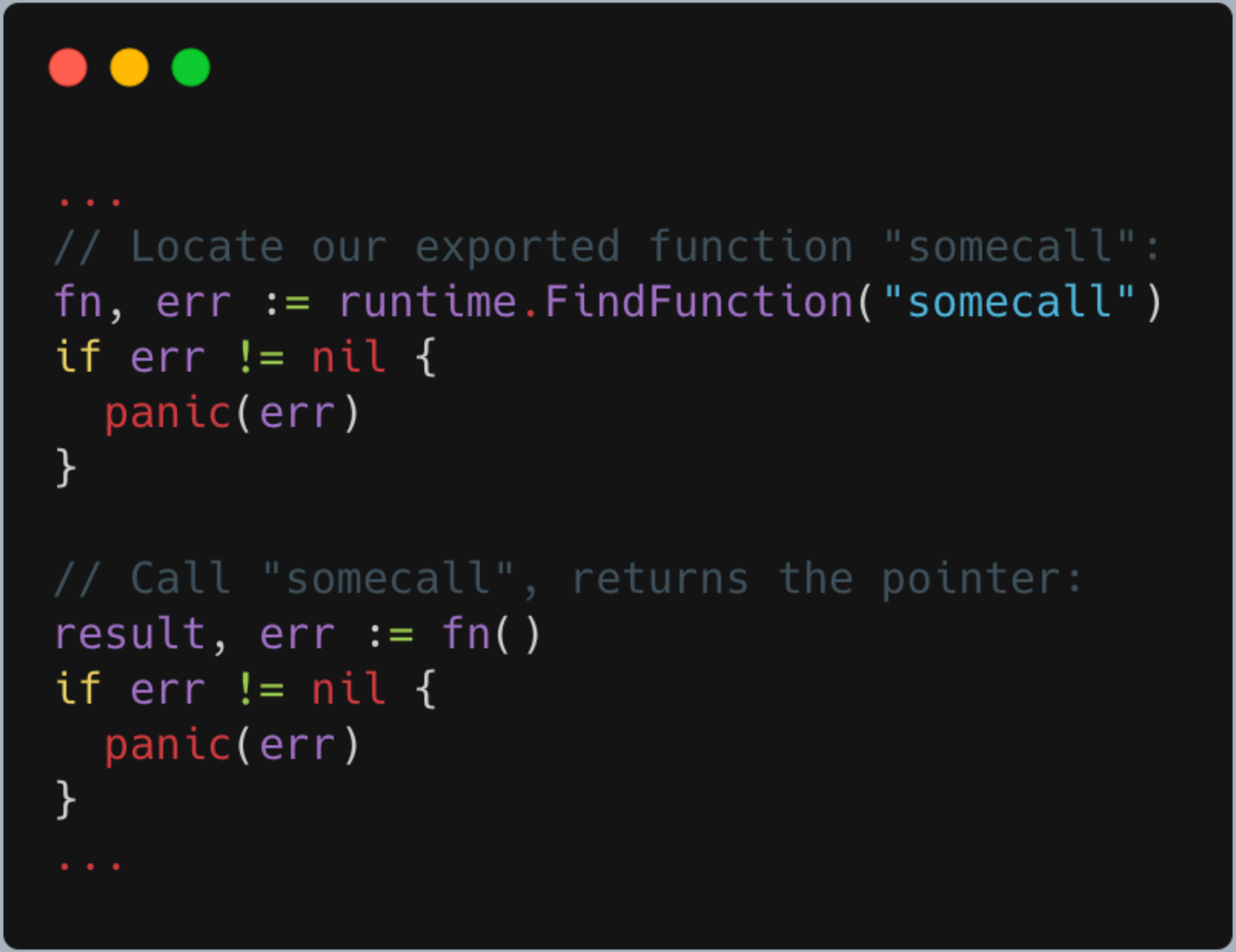
```
runtime := wasm3.NewRuntime(&wasm3.Config{
    Environment: wasm3.NewEnvironment(),
    StackSize: 100000,
})
defer runtime.Destroy()
...
```

(ii) Load the WASM module ✅



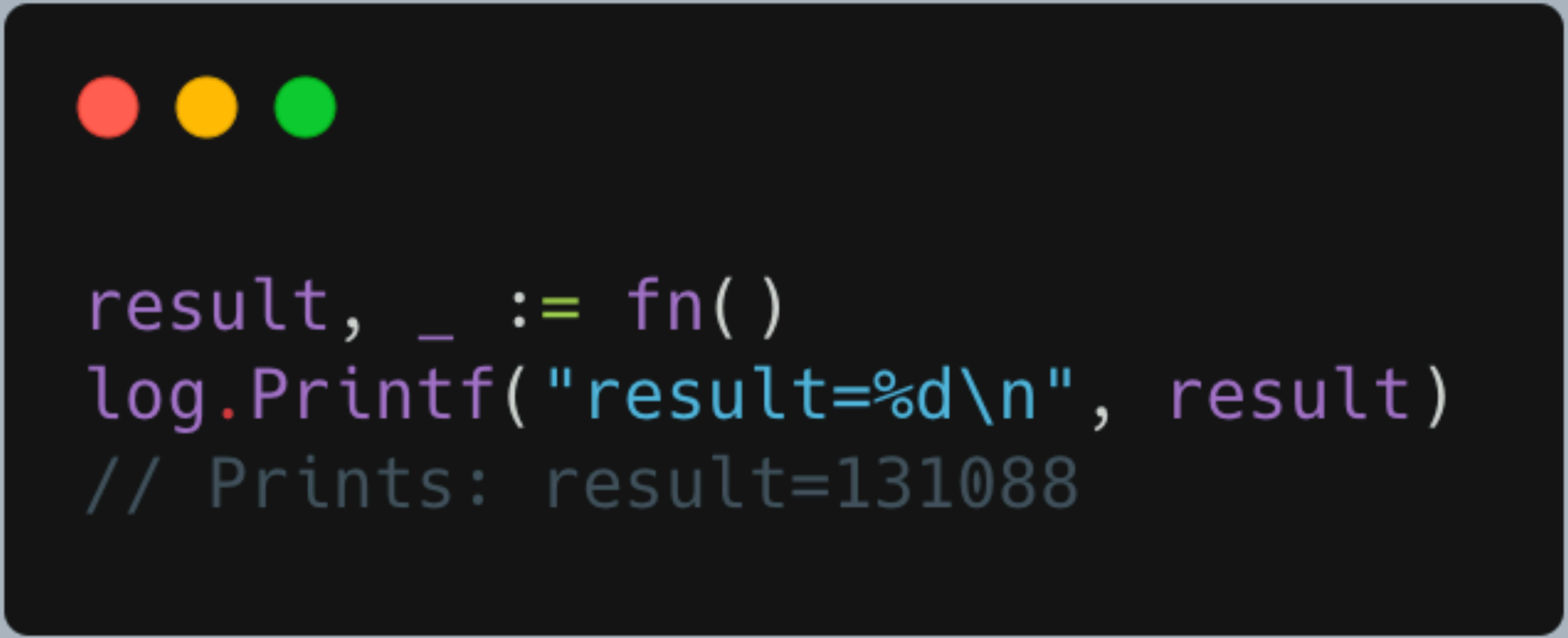
```go
...
wasmBytes, err := ioutil.ReadFile(wasmFilename)
if err != nil {
    panic(err)
}
log.Printf("Read WASM module (%d bytes)\n", len(wasmBytes))
module, err = runtime.LoadModule(module)
if err != nil {
    panic(err)
}
...
```

(iii) Locate and call our function ✅



```
...
// Locate our exported function "somecall":
fn, err := runtime.FindFunction("somecall")
if err != nil {
  panic(err)
}

// Call "somecall", returns the pointer:
result, err := fn()
if err != nil {
  panic(err)
}
...
```

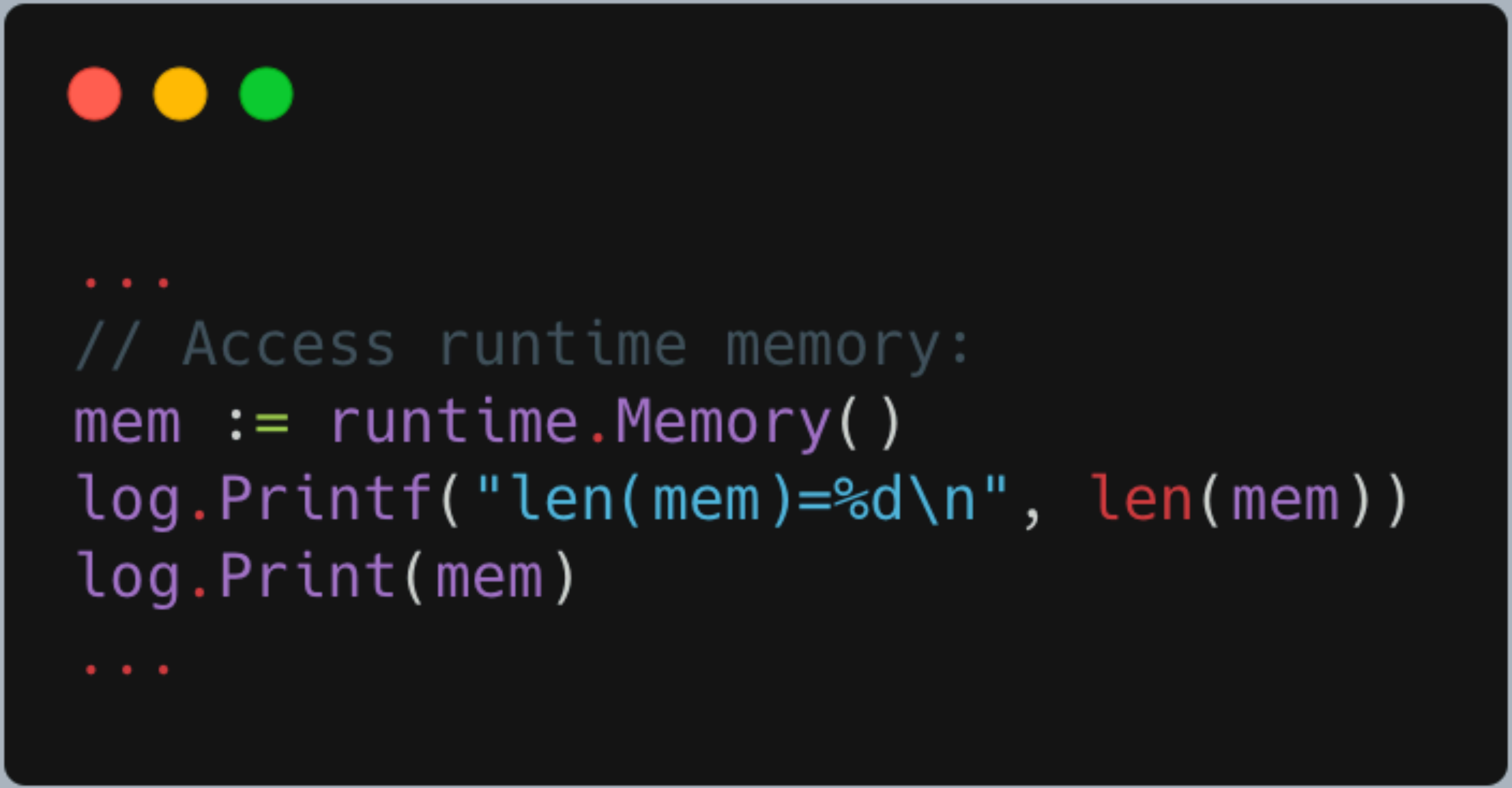
(iv) What’s “result”? 🤔



```
result, _ := fn()
log.Printf("result=%d\n", result)
// Prints: result=131088
```

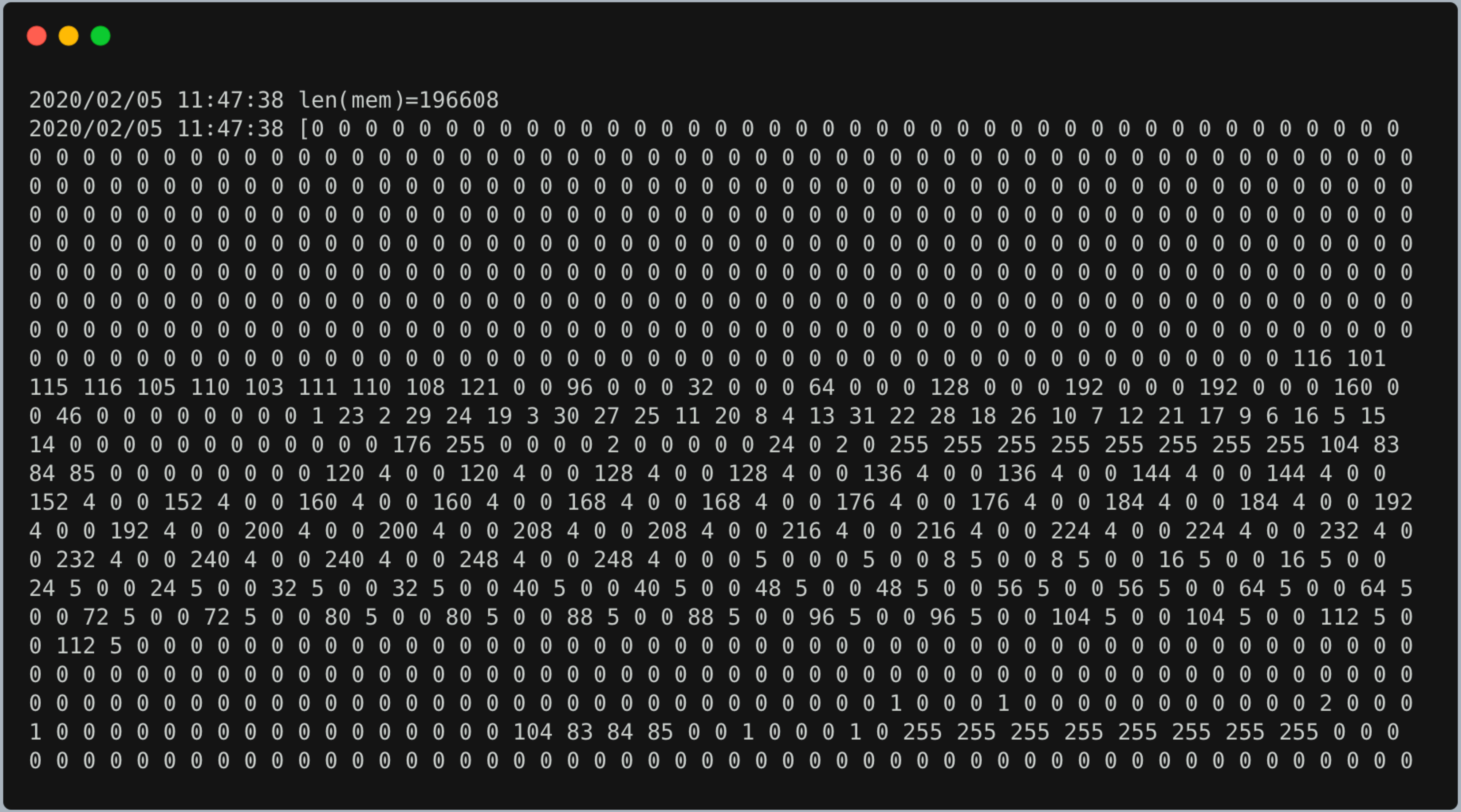
(v) So how do we access the data? 🤔

(vi) Enter linear memory (“runtime.Memory()”)

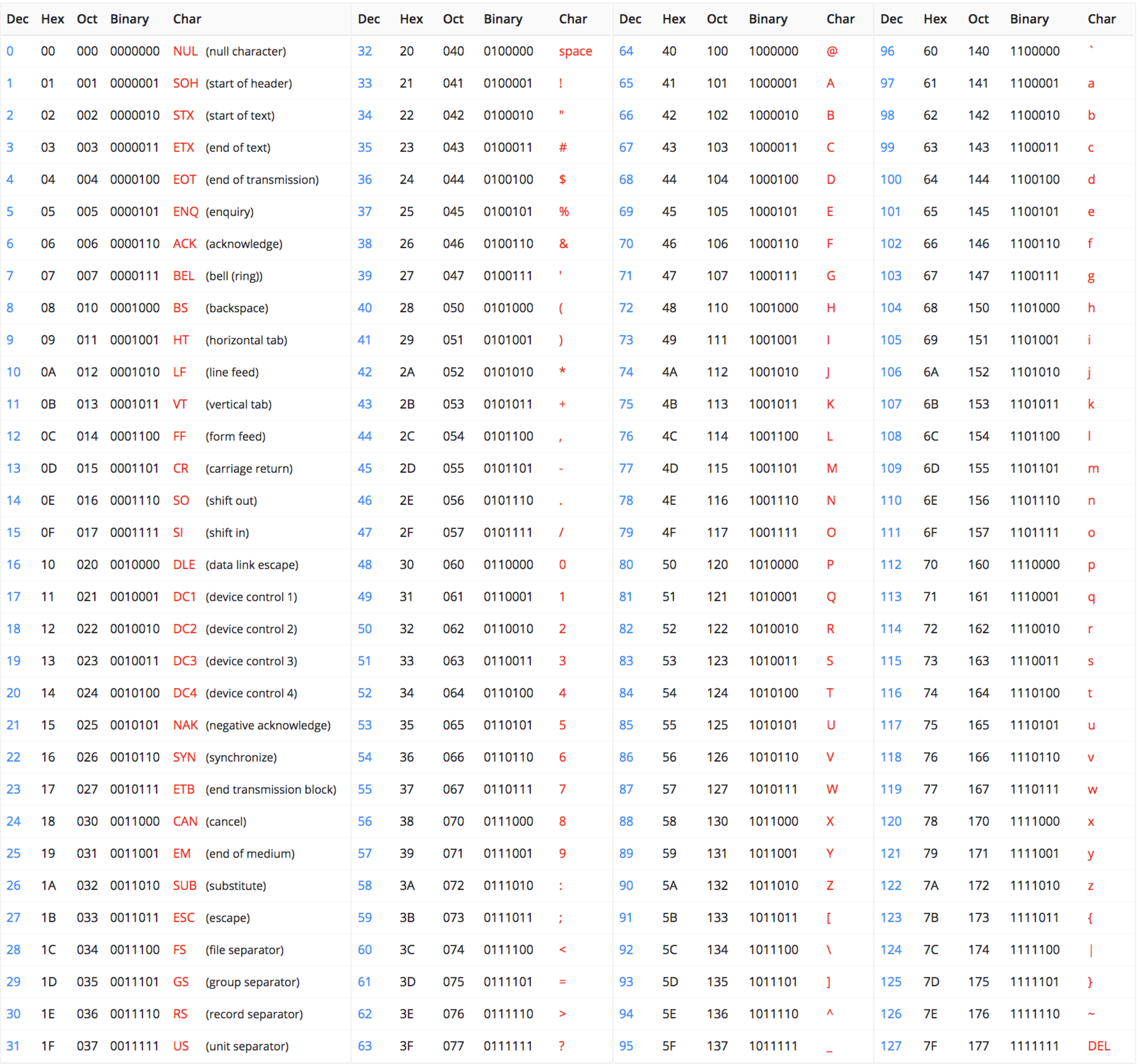


```
...
// Access runtime memory:
mem := runtime.Memory()
log.Printf("len(mem)=%d\n", len(mem))
log.Print(mem)
...
```

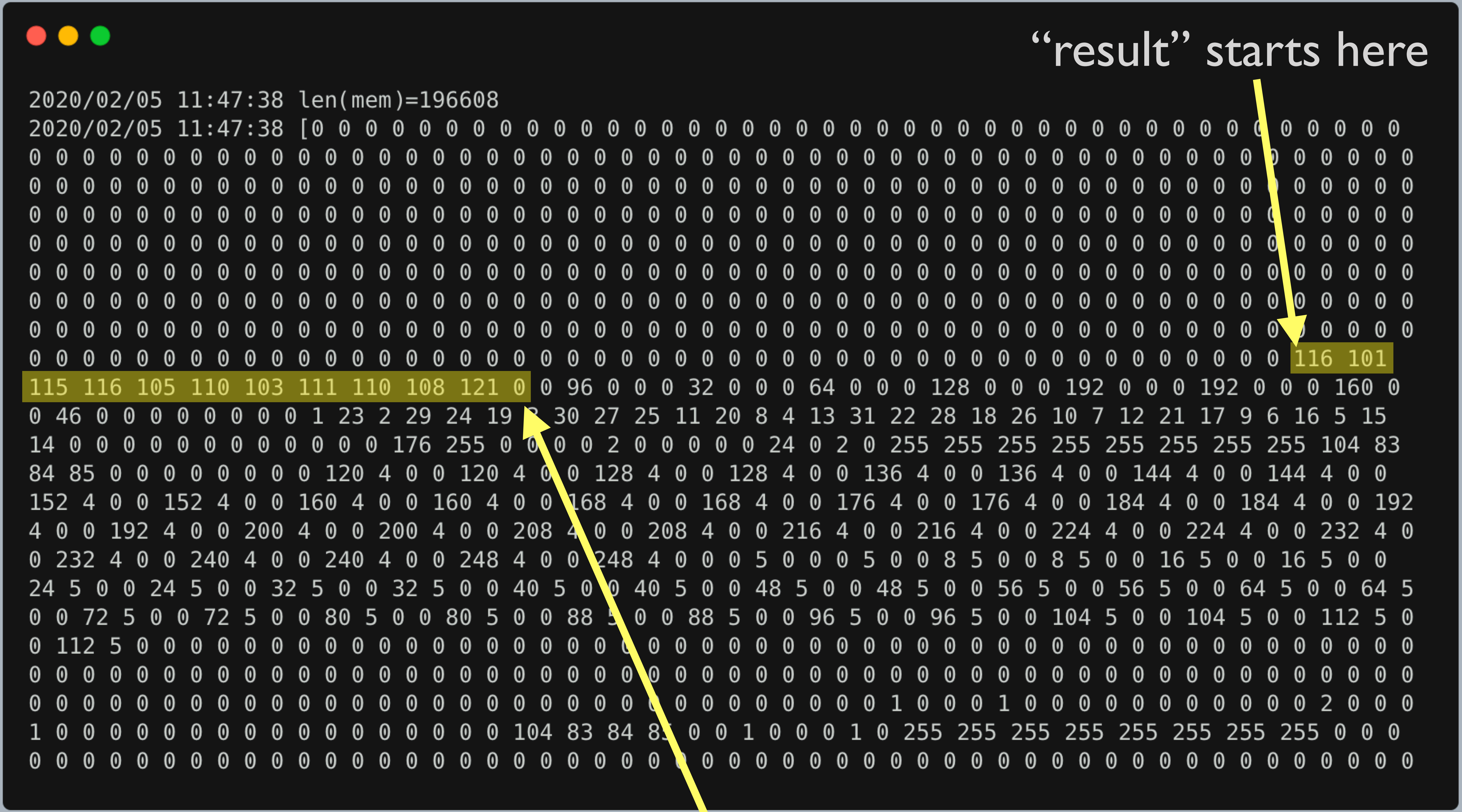
(vi) Enter linear memory (“runtime.Memory()”)



```
2020/02/05 11:47:38 len(mem)=196608
2020/02/05 11:47:38 [0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 116 101
115 116 105 110 103 111 110 108 121 0 0 96 0 0 0 32 0 0 0 64 0 0 0 128 0 0 0 192 0 0 0 192 0 0 0 160 0
0 46 0 0 0 0 0 0 0 0 1 23 2 29 24 19 3 30 27 25 11 20 8 4 13 31 22 28 18 26 10 7 12 21 17 9 6 16 5 15
14 0 0 0 0 0 0 0 0 0 0 0 0 176 255 0 0 0 0 2 0 0 0 0 0 24 0 2 0 255 255 255 255 255 255 255 255 104 83
84 85 0 0 0 0 0 0 0 0 120 4 0 0 120 4 0 0 128 4 0 0 128 4 0 0 136 4 0 0 136 4 0 0 144 4 0 0 144 4 0 0
152 4 0 0 152 4 0 0 160 4 0 0 160 4 0 0 168 4 0 0 168 4 0 0 176 4 0 0 176 4 0 0 184 4 0 0 184 4 0 0 192
4 0 0 192 4 0 0 200 4 0 0 200 4 0 0 208 4 0 0 208 4 0 0 216 4 0 0 216 4 0 0 224 4 0 0 224 4 0 0 232 4 0
0 232 4 0 0 240 4 0 0 240 4 0 0 248 4 0 0 248 4 0 0 0 5 0 0 0 5 0 0 8 5 0 0 8 5 0 0 16 5 0 0 16 5 0 0
24 5 0 0 24 5 0 0 32 5 0 0 32 5 0 0 40 5 0 0 40 5 0 0 48 5 0 0 48 5 0 0 56 5 0 0 56 5 0 0 64 5 0 0 64 5
0 0 72 5 0 0 72 5 0 0 80 5 0 0 80 5 0 0 88 5 0 0 88 5 0 0 96 5 0 0 96 5 0 0 104 5 0 0 104 5 0 0 112 5 0
0 112 5 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 2 0 0 0
1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 104 83 84 85 0 0 1 0 0 0 1 0 255 255 255 255 255 255 255 255 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
```



Dec	Hex	Oct	Binary	Char	Dec	Hex	Oct	Binary	Char	Dec	Hex	Oct	Binary	Char	Dec	Hex	Oct	Binary	Char
0	00	000	0000000	NUL (null character)	32	20	040	0100000	space	64	40	100	1000000	@	96	60	140	1100000	`
1	01	001	0000001	SOH (start of header)	33	21	041	0100001	!	65	41	101	1000001	A	97	61	141	1100001	a
2	02	002	0000010	STX (start of text)	34	22	042	0100010	"	66	42	102	1000010	B	98	62	142	1100010	b
3	03	003	0000011	ETX (end of text)	35	23	043	0100011	#	67	43	103	1000011	C	99	63	143	1100011	c
4	04	004	0000100	EOT (end of transmission)	36	24	044	0100100	$	68	44	104	1000100	D	100	64	144	1100100	d
5	05	005	0000101	ENQ (enquiry)	37	25	045	0100101	%	69	45	105	1000101	E	101	65	145	1100101	e
6	06	006	0000110	ACK (acknowledge)	38	26	046	0100110	&	70	46	106	1000110	F	102	66	146	1100110	f
7	07	007	0000111	BEL (bell (ring))	39	27	047	0100111	'	71	47	107	1000111	G	103	67	147	1100111	g
8	08	010	0001000	BS (backspace)	40	28	050	0101000	(	72	48	110	1001000	H	104	68	150	1101000	h
9	09	011	0001001	HT (horizontal tab)	41	29	051	0101001	)	73	49	111	1001001	I	105	69	151	1101001	i
10	0A	012	0001010	LF (line feed)	42	2A	052	0101010	*	74	4A	112	1001010	J	106	6A	152	1101010	j
11	0B	013	0001011	VT (vertical tab)	43	2B	053	0101011	+	75	4B	113	1001011	K	107	6B	153	1101011	k
12	0C	014	0001100	FF (form feed)	44	2C	054	0101100	,	76	4C	114	1001100	L	108	6C	154	1101100	l
13	0D	015	0001101	CR (carriage return)	45	2D	055	0101101	-	77	4D	115	1001101	M	109	6D	155	1101101	m
14	0E	016	0001110	SO (shift out)	46	2E	056	0101110	.	78	4E	116	1001110	N	110	6E	156	1101110	n
15	0F	017	0001111	SI (shift in)	47	2F	057	0101111	/	79	4F	117	1001111	O	111	6F	157	1101111	o
16	10	020	0010000	DLE (data link escape)	48	30	060	0110000	0	80	50	120	1010000	P	112	70	160	1110000	p
17	11	021	0010001	DC1 (device control 1)	49	31	061	0110001	1	81	51	121	1010001	Q	113	71	161	1110001	q
18	12	022	0010010	DC2 (device control 2)	50	32	062	0110010	2	82	52	122	1010010	R	114	72	162	1110010	r
19	13	023	0010011	DC3 (device control 3)	51	33	063	0110011	3	83	53	123	1010011	S	115	73	163	1110011	s
20	14	024	0010100	DC4 (device control 4)	52	34	064	0110100	4	84	54	124	1010100	T	116	74	164	1110100	t
21	15	025	0010101	NAK (negative acknowledge)	53	35	065	0110101	5	85	55	125	1010101	U	117	75	165	1110101	u
22	16	026	0010110	SYN (synchronize)	54	36	066	0110110	6	86	56	126	1010110	V	118	76	166	1110110	v
23	17	027	0010111	ETB (end transmission block)	55	37	067	0110111	7	87	57	127	1010111	W	119	77	167	1110111	w
24	18	030	0011000	CAN (cancel)	56	38	070	0111000	8	88	58	130	1011000	X	120	78	170	1111000	x
25	19	031	0011001	EM (end of medium)	57	39	071	0111001	9	89	59	131	1011001	Y	121	79	171	1111001	y
26	1A	032	0011010	SUB (substitute)	58	3A	072	0111010	:	90	5A	132	1011010	Z	122	7A	172	1111010	z
27	1B	033	0011011	ESC (escape)	59	3B	073	0111011	;	91	5B	133	1011011	[	123	7B	173	1111011	{
28	1C	034	0011100	FS (file separator)	60	3C	074	0111100	<	92	5C	134	1011100	\	124	7C	174	1111100	|
29	1D	035	0011101	GS (group separator)	61	3D	075	0111101	=	93	5D	135	1011101	]	125	7D	175	1111101	}
30	1E	036	0011110	RS (record separator)	62	3E	076	0111110	>	94	5E	136	1011110	^	126	7E	176	1111110	~
31	1F	037	0011111	US (unit separator)	63	3F	077	0111111	?	95	5F	137	1011111	_	127	7F	177	1111111	DEL



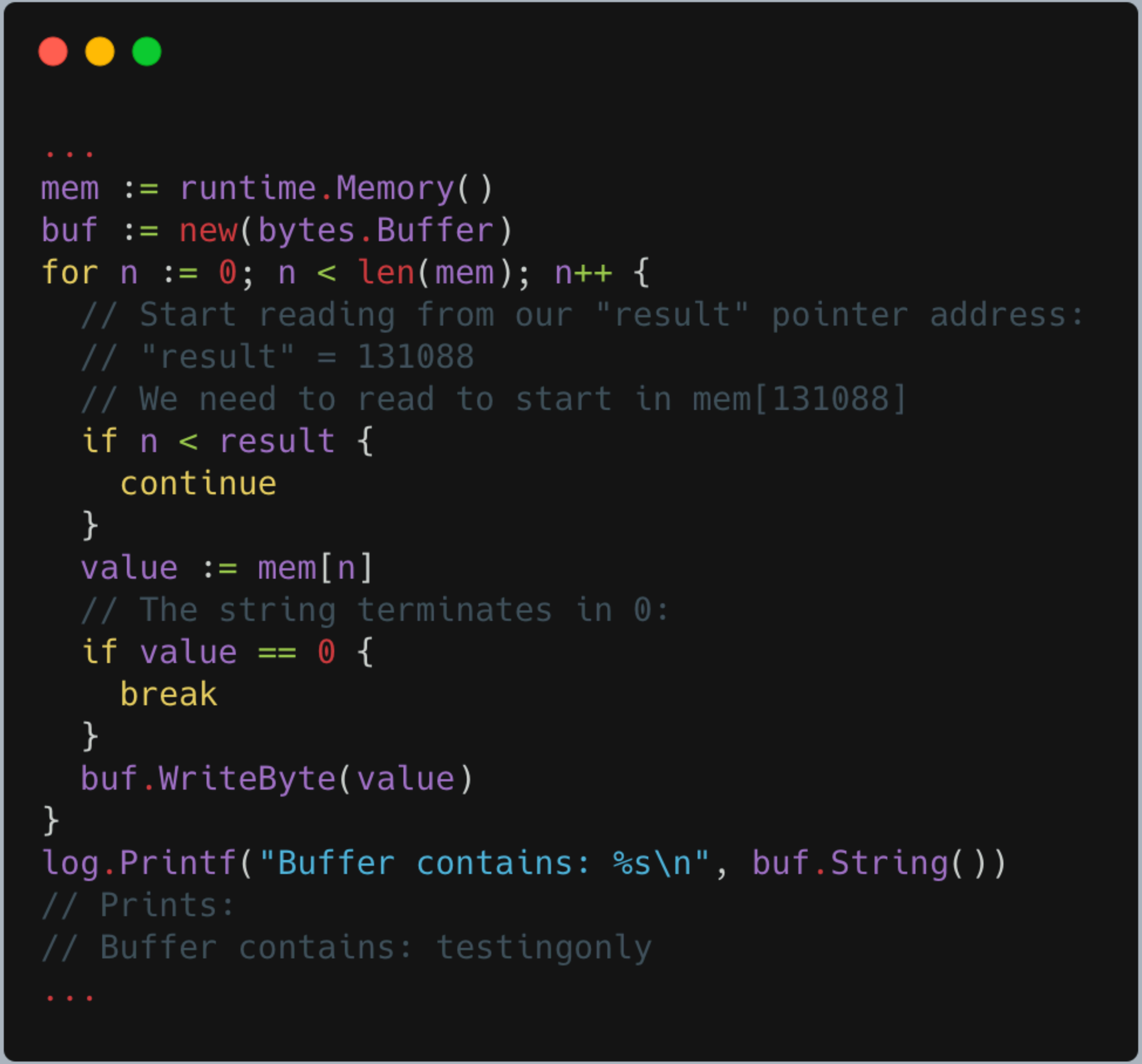
“result” starts here

```
2020/02/05 11:47:38 len(mem)=196608
2020/02/05 11:47:38 [0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 116 101
115 116 105 110 103 111 110 108 121 0 0 96 0 0 0 32 0 0 0 64 0 0 0 128 0 0 0 192 0 0 0 192 0 0 0 160 0
0 46 0 0 0 0 0 0 0 0 1 23 2 29 24 19 3 30 27 25 11 20 8 4 13 31 22 28 18 26 10 7 12 21 17 9 6 16 5 15
14 0 0 0 0 0 0 0 0 0 0 0 0 176 255 0 0 0 0 2 0 0 0 0 0 24 0 2 0 255 255 255 255 255 255 255 255 104 83
84 85 0 0 0 0 0 0 0 0 120 4 0 0 120 4 0 0 128 4 0 0 128 4 0 0 136 4 0 0 136 4 0 0 144 4 0 0 144 4 0 0
152 4 0 0 152 4 0 0 160 4 0 0 160 4 0 0 168 4 0 0 168 4 0 0 176 4 0 0 176 4 0 0 184 4 0 0 184 4 0 0 192
4 0 0 192 4 0 0 200 4 0 0 200 4 0 0 208 4 0 0 208 4 0 0 216 4 0 0 216 4 0 0 224 4 0 0 224 4 0 0 232 4 0
0 232 4 0 0 240 4 0 0 240 4 0 0 248 4 0 0 248 4 0 0 0 5 0 0 0 5 0 0 8 5 0 0 8 5 0 0 16 5 0 0 16 5 0 0
24 5 0 0 24 5 0 0 32 5 0 0 32 5 0 0 40 5 0 0 40 5 0 0 48 5 0 0 48 5 0 0 56 5 0 0 56 5 0 0 64 5 0 0 64 5
0 0 72 5 0 0 72 5 0 0 80 5 0 0 80 5 0 0 88 5 0 0 88 5 0 0 96 5 0 0 96 5 0 0 104 5 0 0 104 5 0 0 112 5 0
0 112 5 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 2 0 0 0
1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 104 83 84 85 0 0 1 0 0 0 1 0 255 255 255 255 255 255 255 255 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
```

“result” ends here

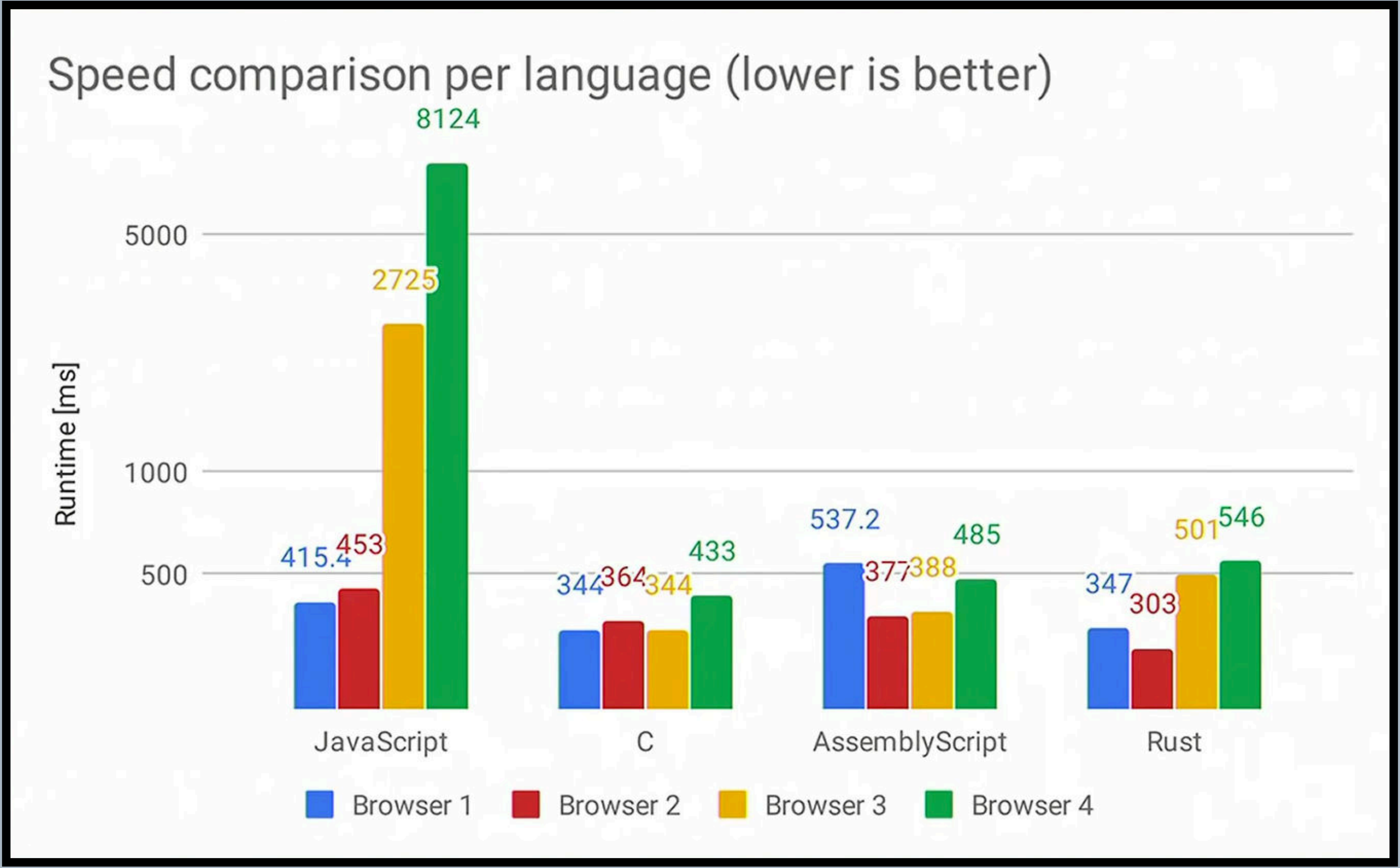
```
 t   e   s   t   i   n   g   o   n   l   y
116 101 115 116 105 110 103 111 110 108 121 0
```

(vii) Now we can reconstruct the string 😃



```go
...
mem := runtime.Memory()
buf := new(bytes.Buffer)
for n := 0; n < len(mem); n++ {
  // Start reading from our "result" pointer address:
  // "result" = 131088
  // We need to read to start in mem[131088]
  if n < result {
    continue
  }
  value := mem[n]
  // The string terminates in 0:
  if value == 0 {
    break
  }
  buf.WriteByte(value)
}
log.Printf("Buffer contains: %s\n", buf.String())
// Prints:
// Buffer contains: testingonly
...
```

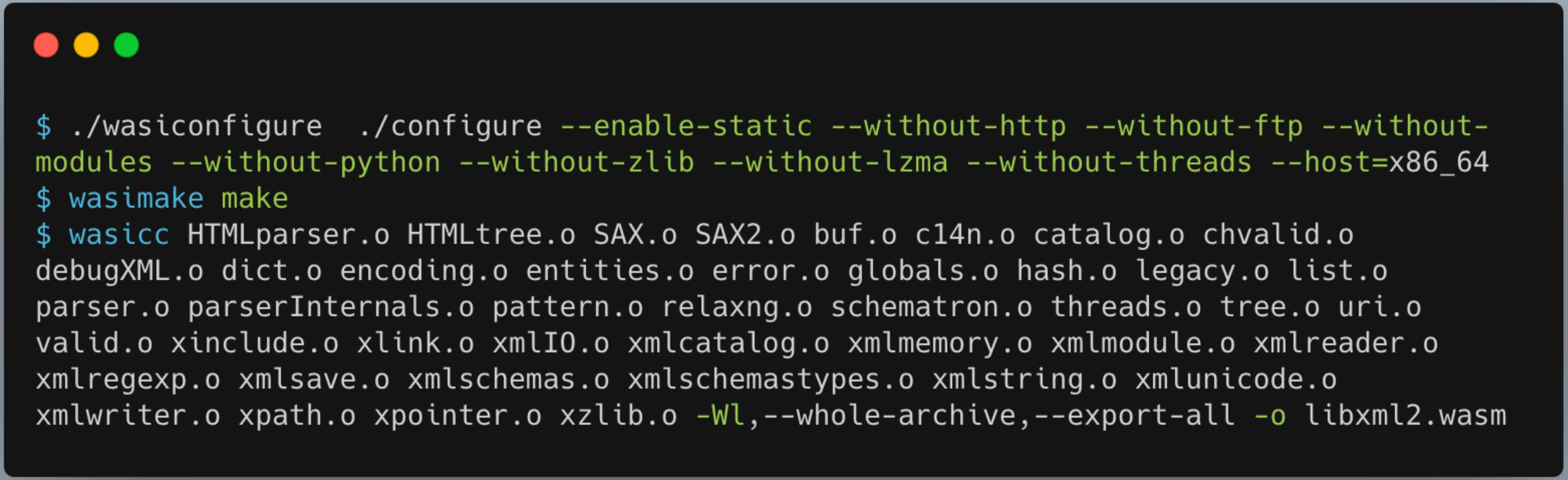
Performance and interop potential



From Google I/O 19: "WebAssembly for Web Developers"

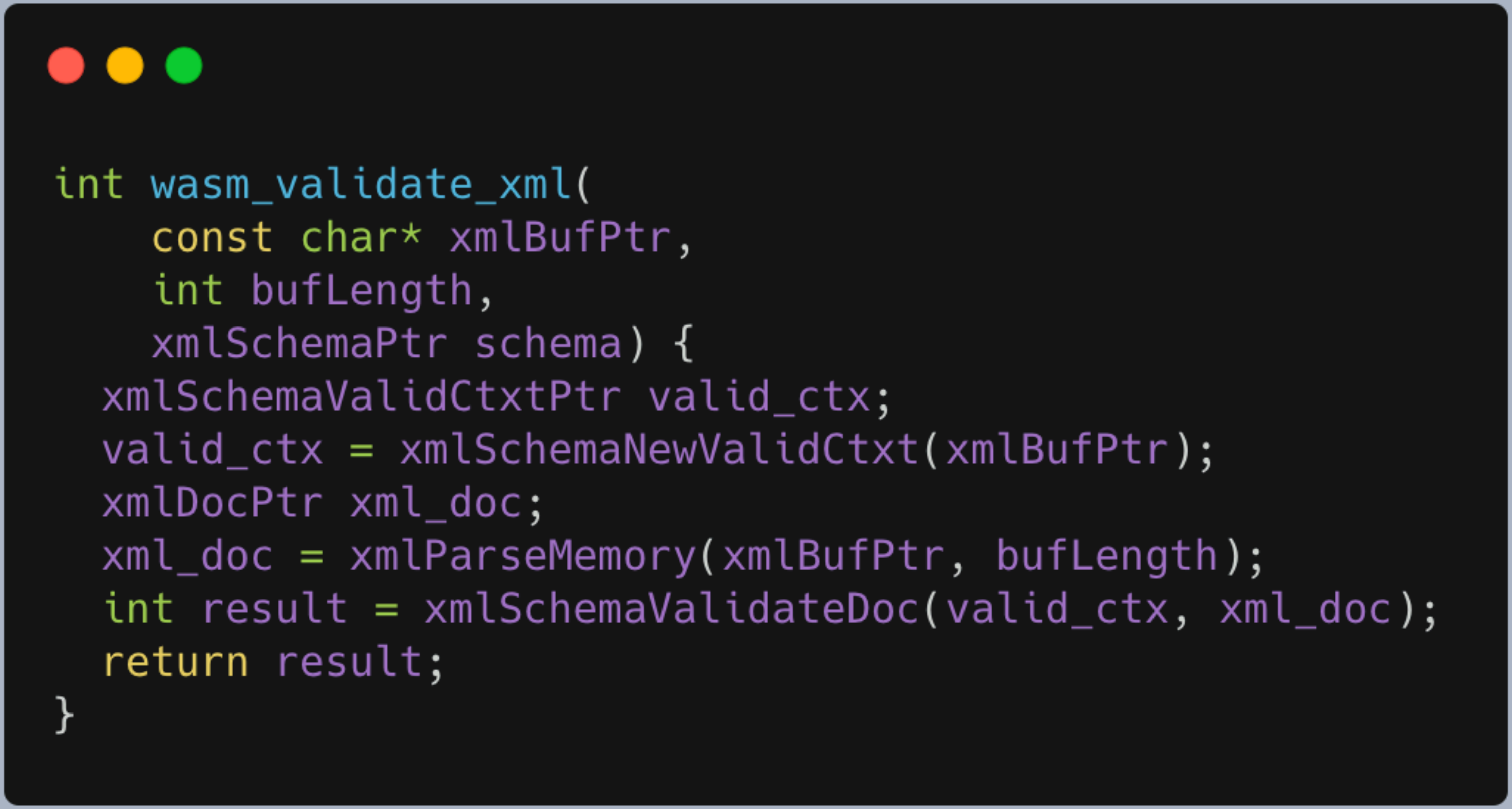
Performance and interop potential

(i) Build libxml2 as a WASM module.
The output (libxml2.wasm) is a 4 MB file, containing everything we need:



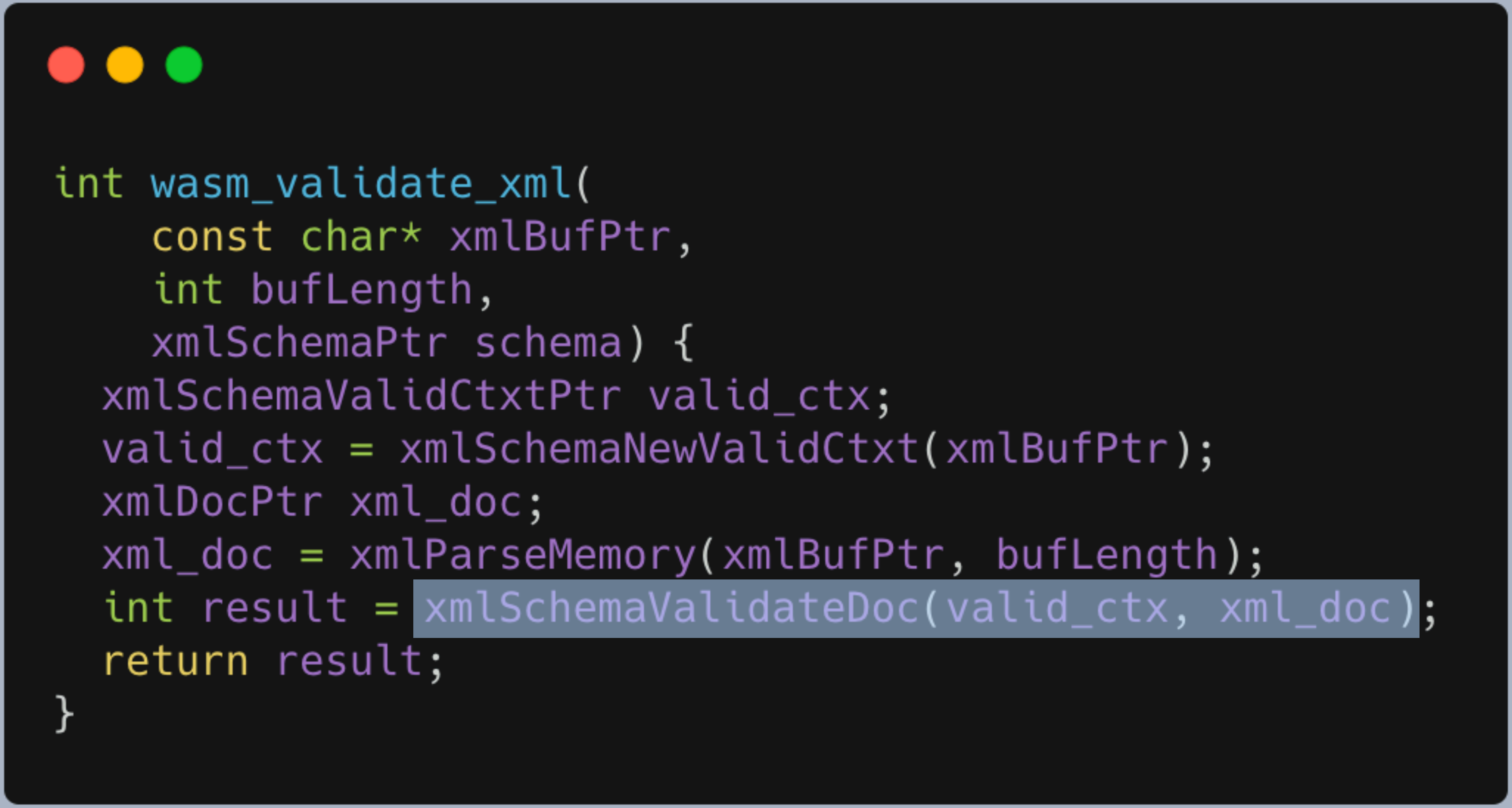
```
$ ./wasiconfigure  ./configure --enable-static --without-http --without-ftp --without-
modules --without-python --without-zlib --without-lzma --without-threads --host=x86_64
$ wasimake make
$ wasicc HTMLparser.o HTMLtree.o SAX.o SAX2.o buf.o c14n.o catalog.o chvalid.o
debugXML.o dict.o encoding.o entities.o error.o globals.o hash.o legacy.o list.o
parser.o parserInternals.o pattern.o relaxng.o schematron.o threads.o tree.o uri.o
valid.o xinclude.o xlink.o xmlIO.o xmlcatalog.o xmlmemory.o xmlmodule.o xmlreader.o
xmlregexp.o xmlsave.o xmlschemas.o xmlschemastypes.o xmlstring.o xmlunicode.o
xmlwriter.o xpath.o xpointer.o xzlib.o -Wl,--whole-archive,--export-all -o libxml2.wasm
```

(ii) When building libxml2, the following wrapper function was included:

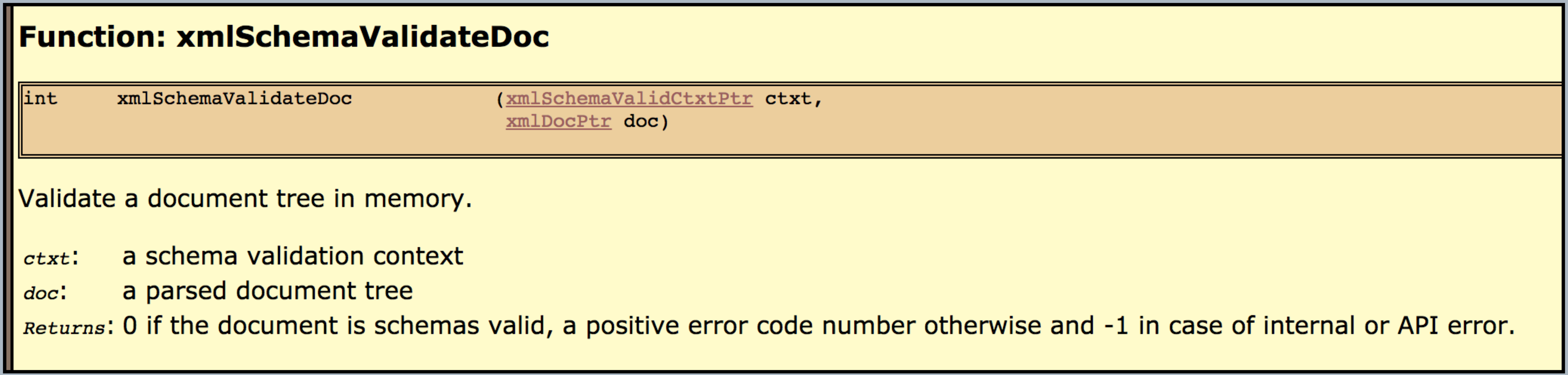


```
int wasm_validate_xml(
    const char* xmlBufPtr,
    int bufLength,
    xmlSchemaPtr schema) {
  xmlSchemaValidCtxtPtr valid_ctx;
  valid_ctx = xmlSchemaNewValidCtxt(xmlBufPtr);
  xmlDocPtr xml_doc;
  xml_doc = xmlParseMemory(xmlBufPtr, bufLength);
  int result = xmlSchemaValidateDoc(valid_ctx, xml_doc);
  return result;
}
```

(ii) When building libxml2, the following wrapper function was included:



```
int wasm_validate_xml(
    const char* xmlBufPtr,
    int bufLength,
    xmlSchemaPtr schema) {
  xmlSchemaValidCtxtPtr valid_ctx;
  valid_ctx = xmlSchemaNewValidCtxt(xmlBufPtr);
  xmlDocPtr xml_doc;
  xml_doc = xmlParseMemory(xmlBufPtr, bufLength);
  int result = xmlSchemaValidateDoc(valid_ctx, xml_doc);
  return result;
}
```



Function: xmlSchemaValidateDoc

```
int	xmlSchemaValidateDoc		(xmlSchemaValidCtxtPtr ctxt, 
					 xmlDocPtr doc)
```

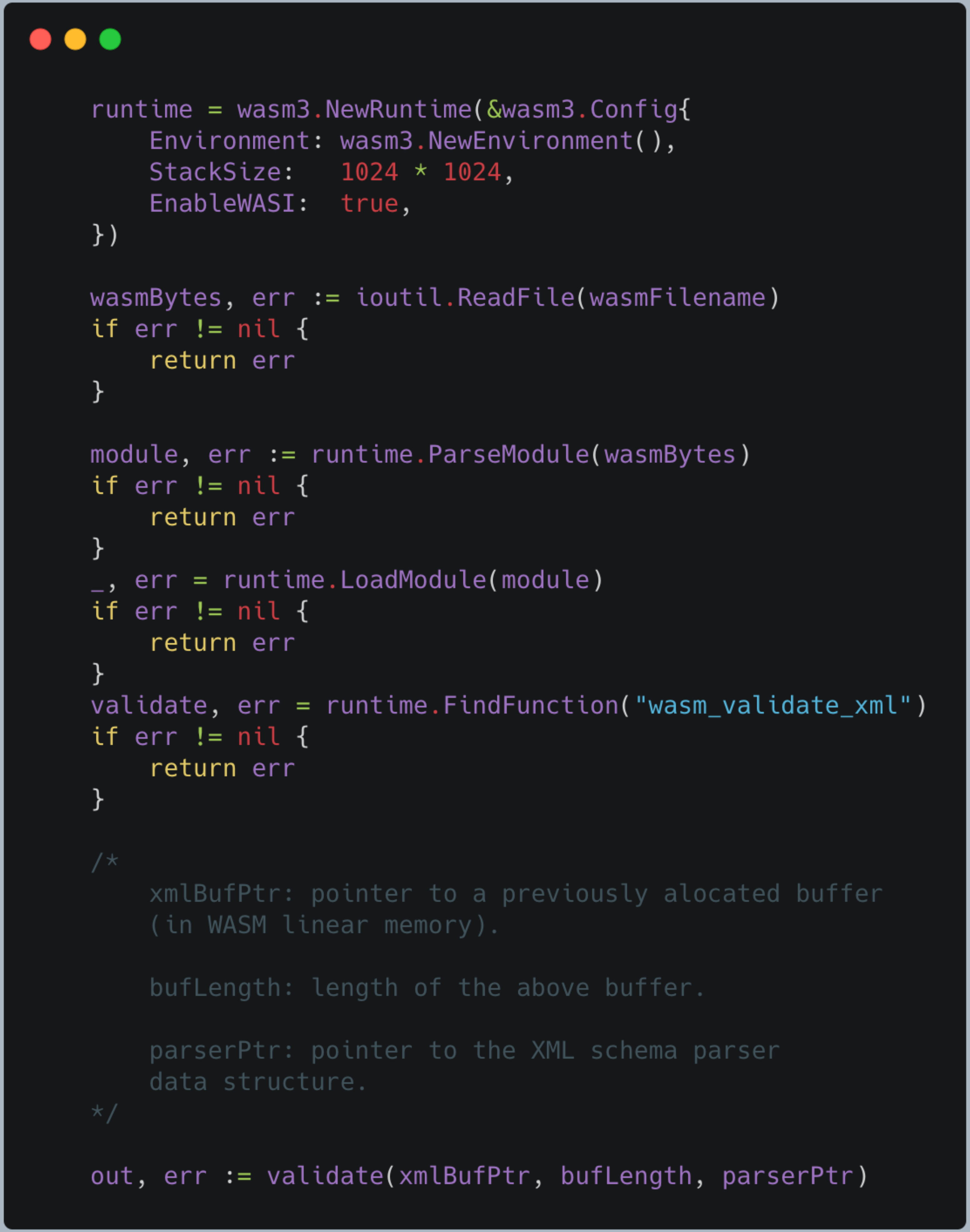
Validate a document tree in memory.

ctxt: a schema validation context

doc: a parsed document tree

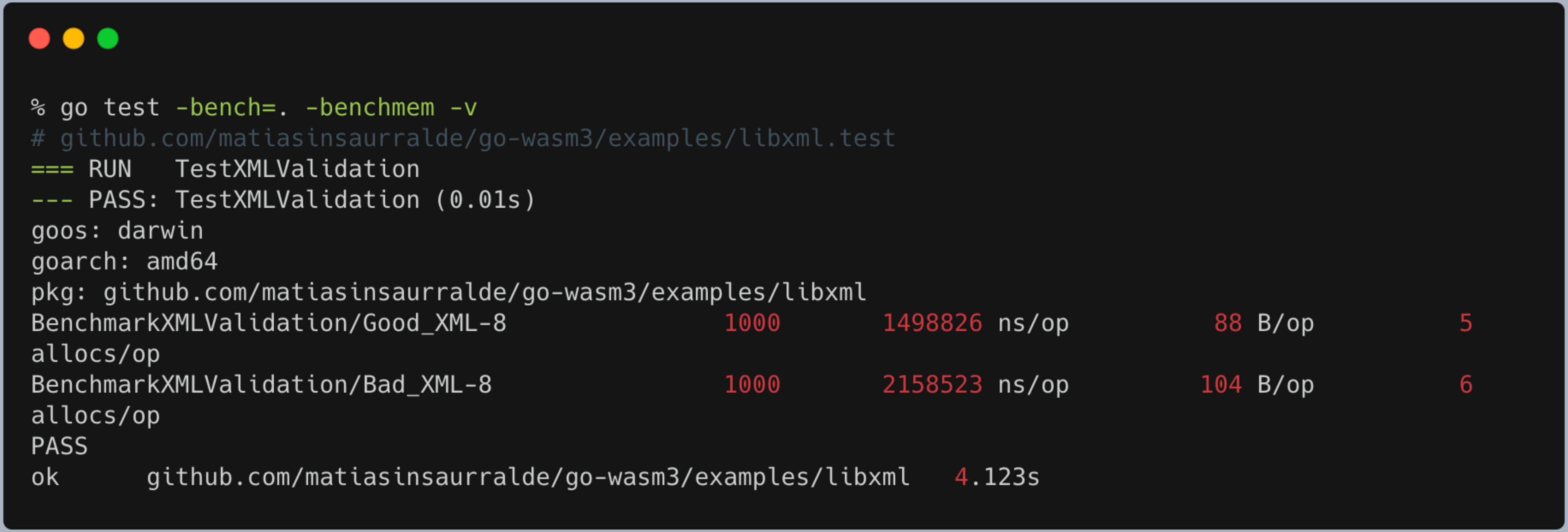
Returns: 0 if the document is schemas valid, a positive error code number otherwise and -1 in case of internal or API error.

(iii) Write a Go program that loads the WASM module and calls “wasm_validate_xml”:



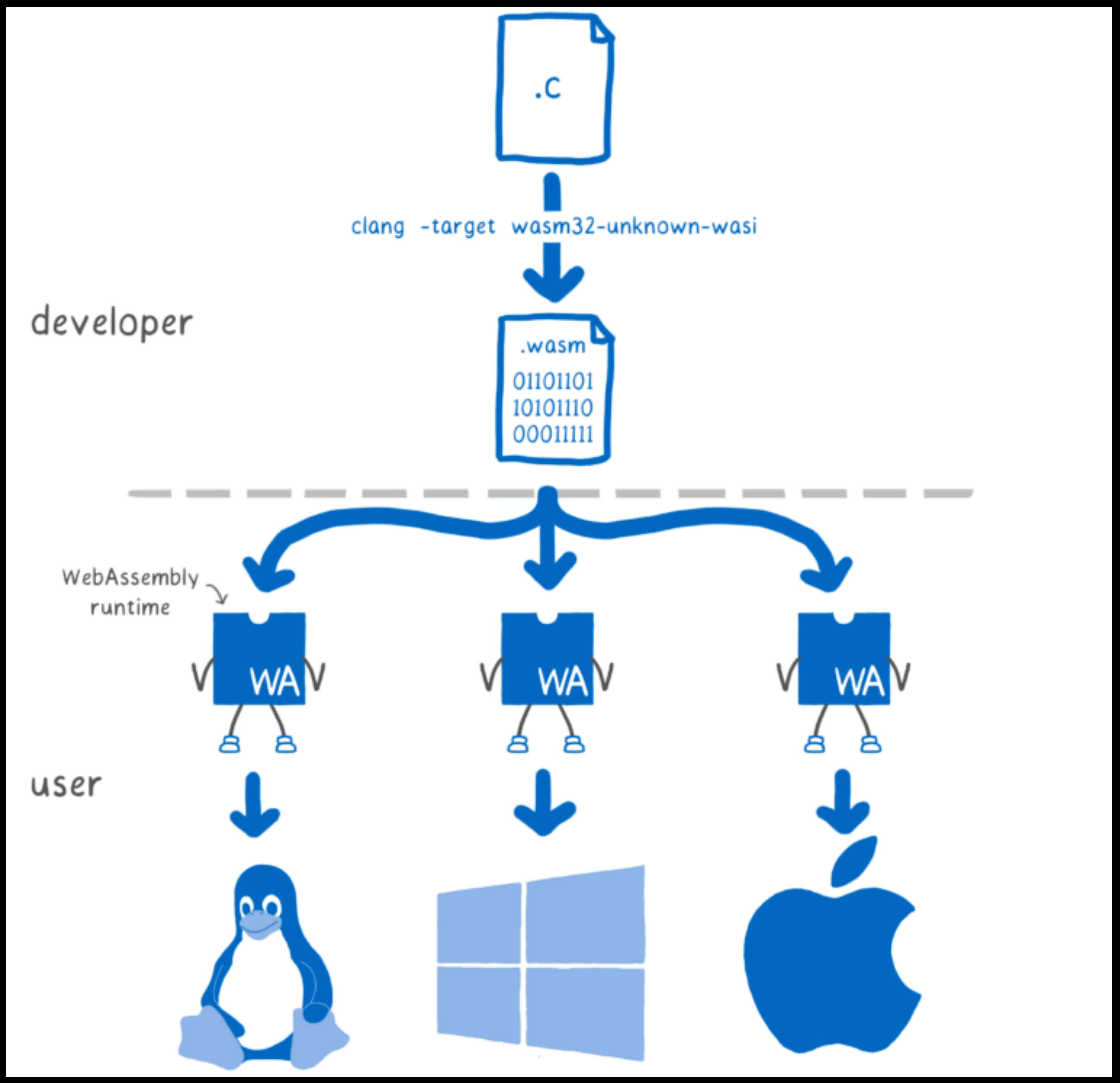
```
runtime = wasm3.NewRuntime(&wasm3.Config{
    Environment: wasm3.NewEnvironment(),
    StackSize:   1024 * 1024,
    EnableWASI:  true,
})

wasmBytes, err := ioutil.ReadFile(wasmFilename)
if err != nil {
    return err
}

module, err := runtime.ParseModule(wasmBytes)
if err != nil {
    return err
}
_, err = runtime.LoadModule(module)
if err != nil {
    return err
}
validate, err = runtime.FindFunction("wasm_validate_xml")
if err != nil {
    return err
}

/*
    xmlBufPtr: pointer to a previously alocated buffer
    (in WASM linear memory).

    bufLength: length of the above buffer.

    parserPtr: pointer to the XML schema parser
    data structure.
*/

out, err := validate(xmlBufPtr, bufLength, parserPtr)
```

(iv) Benchmark:

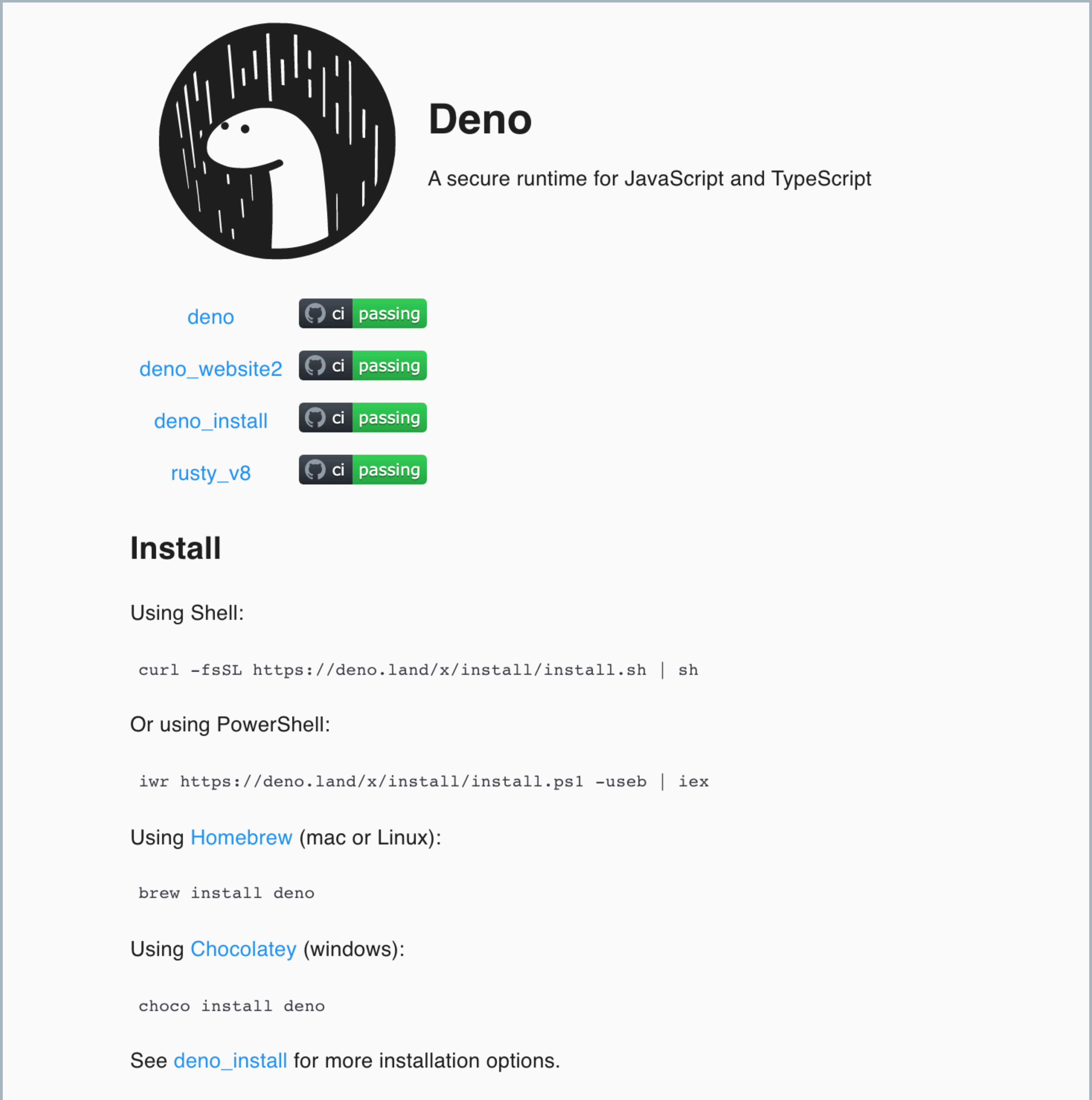


```
% go test -bench=. -benchmem -v
# github.com/matiasinsaurralde/go-wasm3/examples/libxml.test
=== RUN   TestXMLValidation
--- PASS: TestXMLValidation (0.01s)
goos: darwin
goarch: amd64
pkg: github.com/matiasinsaurralde/go-wasm3/examples/libxml
BenchmarkXMLValidation/Good_XML-8               1000       1498826 ns/op          88 B/op          5
allocs/op
BenchmarkXMLValidation/Bad_XML-8                1000       2158523 ns/op         104 B/op          6
allocs/op
PASS
ok      github.com/matiasinsaurralde/go-wasm3/examples/libxml   4.123s
```

What’s next?
WASI and beyond



```
clang -target wasm32-unknown-wasi
```



Deno

A secure runtime for JavaScript and TypeScript

deno	ci passing
deno_website2	ci passing
deno_install	ci passing
rusty_v8	ci passing

Install

Using Shell:

```
curl -fsSL https://deno.land/x/install/install.sh | sh
```

Or using PowerShell:

```
iwr https://deno.land/x/install/install.ps1 -useb | iex
```

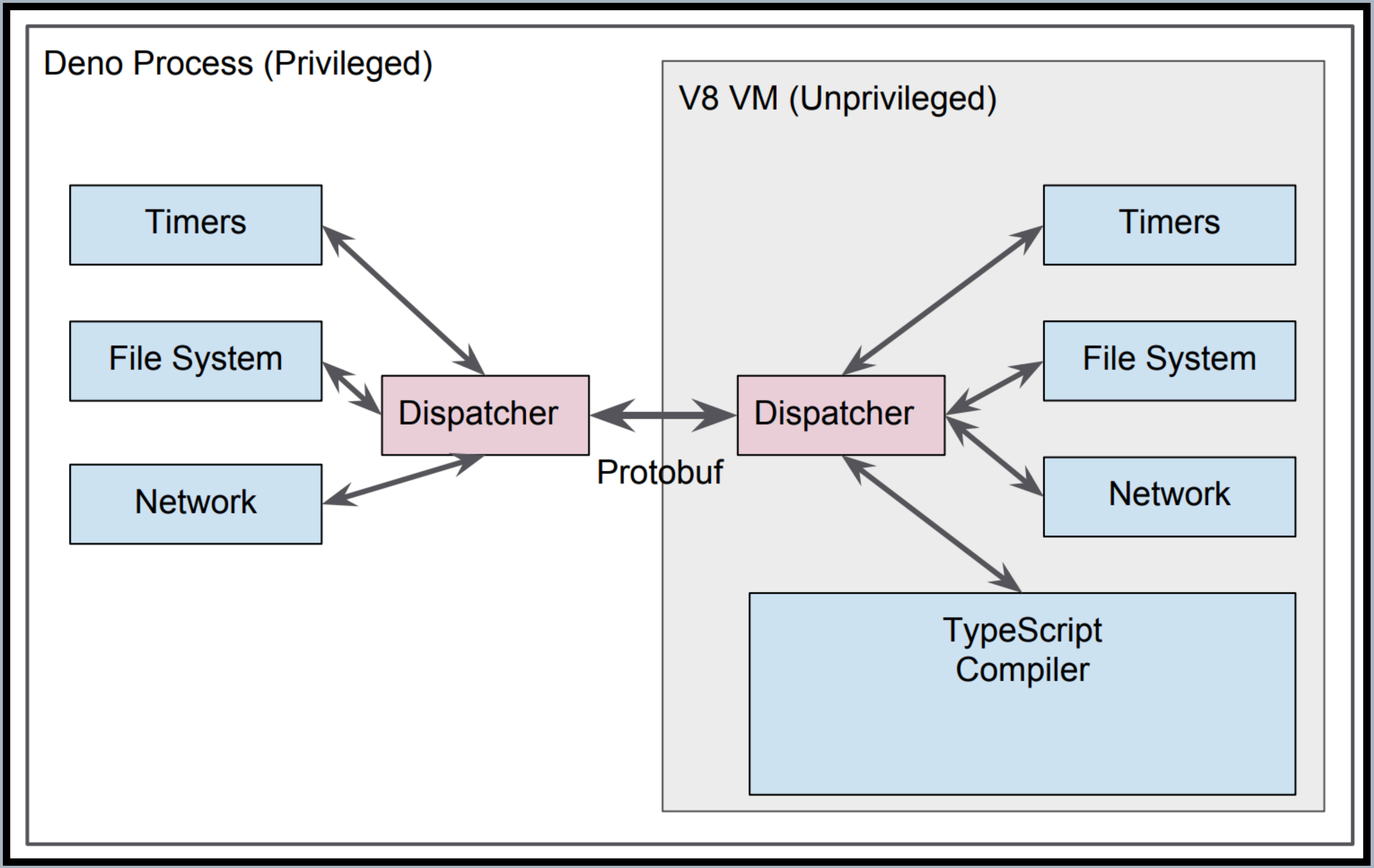
Using Homebrew (mac or Linux):

```
brew install deno
```

Using Chocolatey (windows):

```
choco install deno
```

See deno_install for more installation options.



deno architecture: http://tinyclouds.org/jsconf2018.pdf

What’s next for Go VMs?

- Support reentrancy.

- Enhance performance in concurrent scenarios (e.g. when invoking WASM functions from a Go HTTP server).

- Is cgo overhead ok? Should we focus on pure Go VMs?

- Implement WASI in Go VMs.

Thanks!

go-wasm3 repo:

https://github.com/matiasinsaurralde/go-wasm3

Twitter: @matias_baruch
Github: matiasinsaurralde
E-mail: matias@insaurral.de