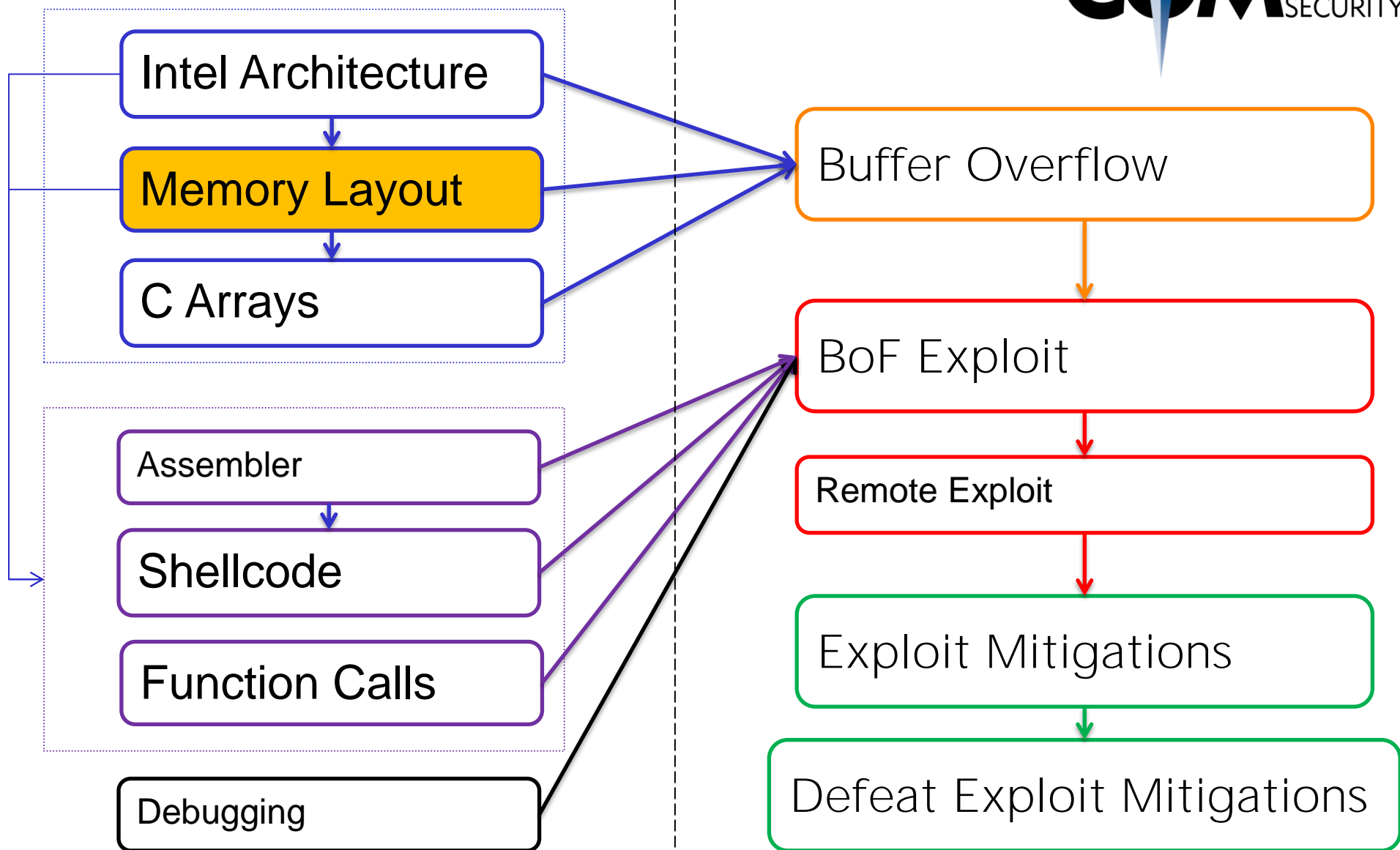


Memory Layout

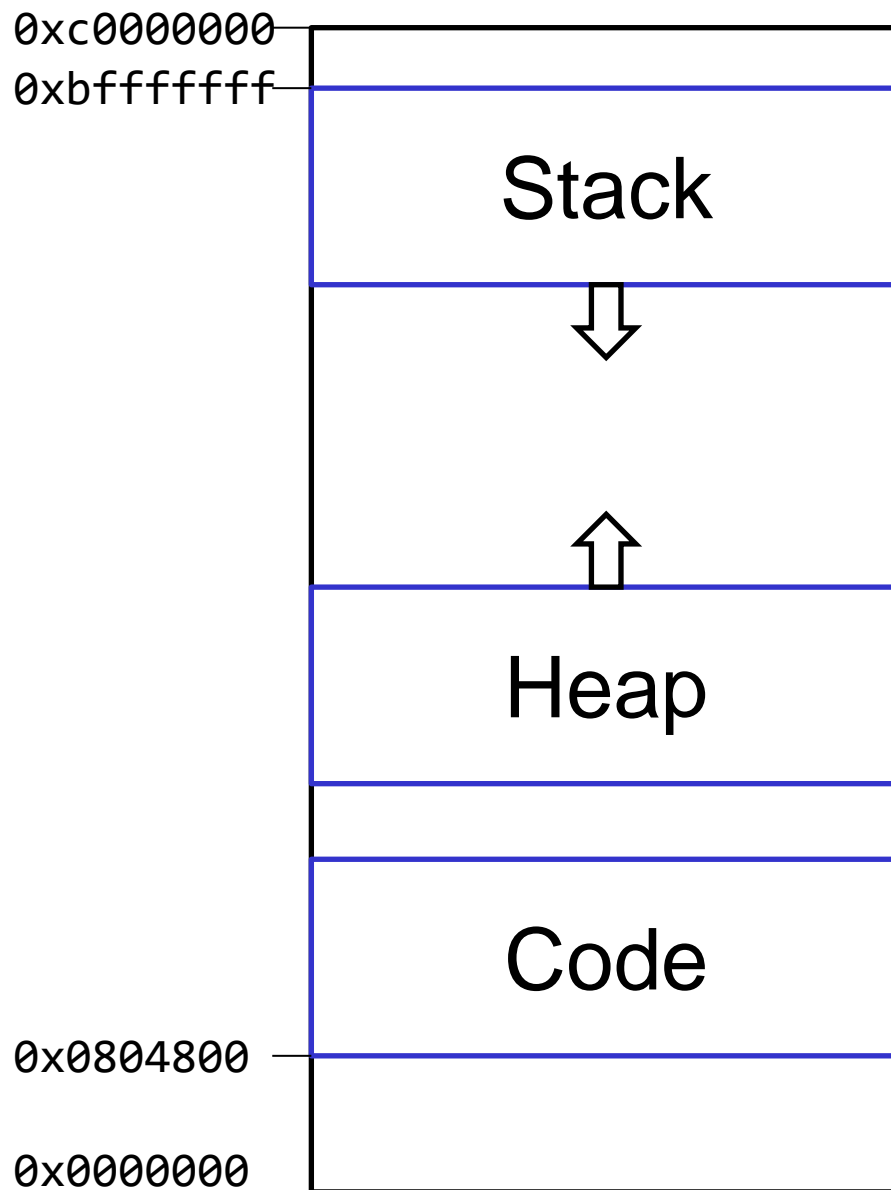
Linux Userspace Process Memory Layout



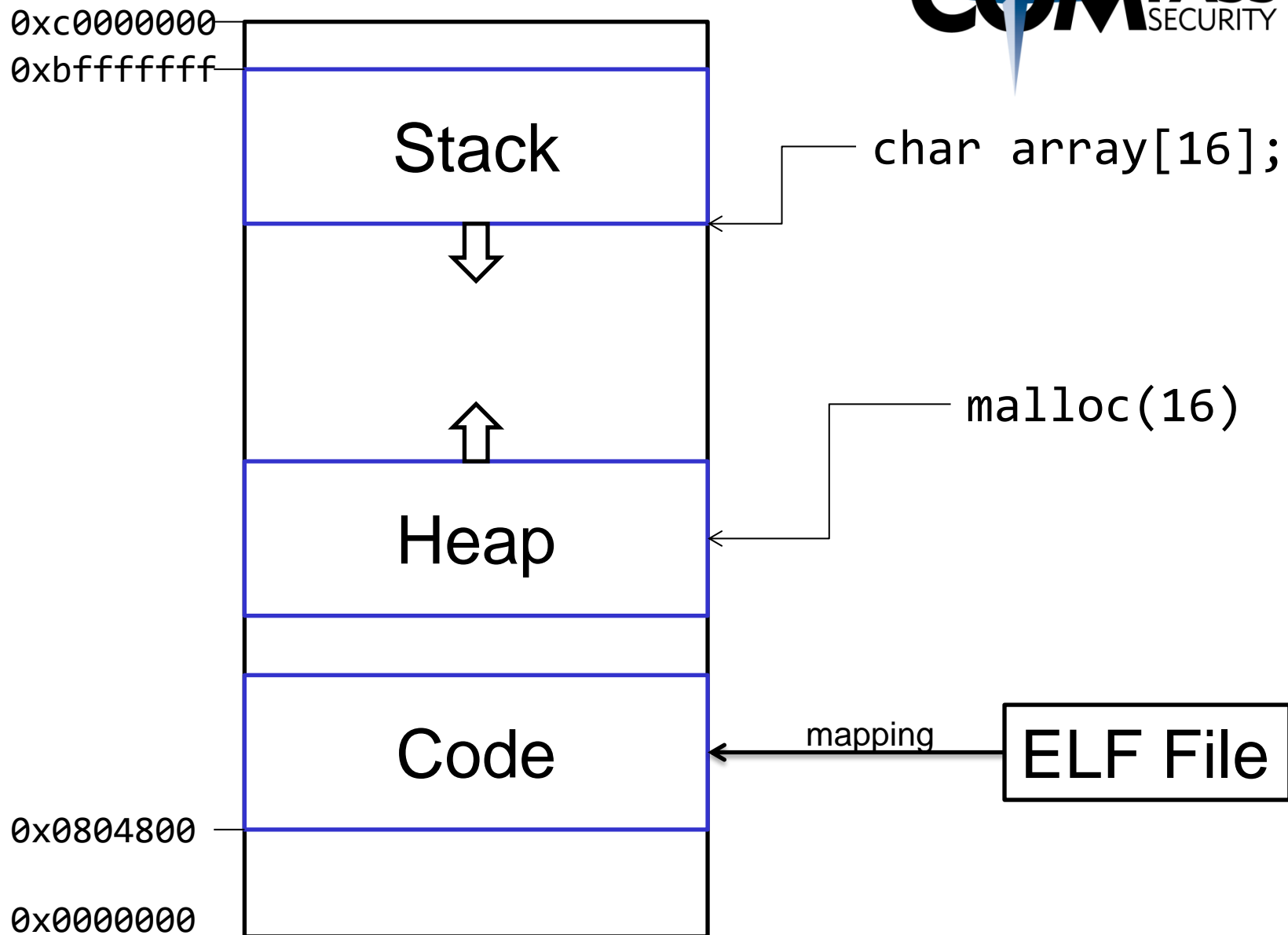
Userspace Memory Layout

In x32

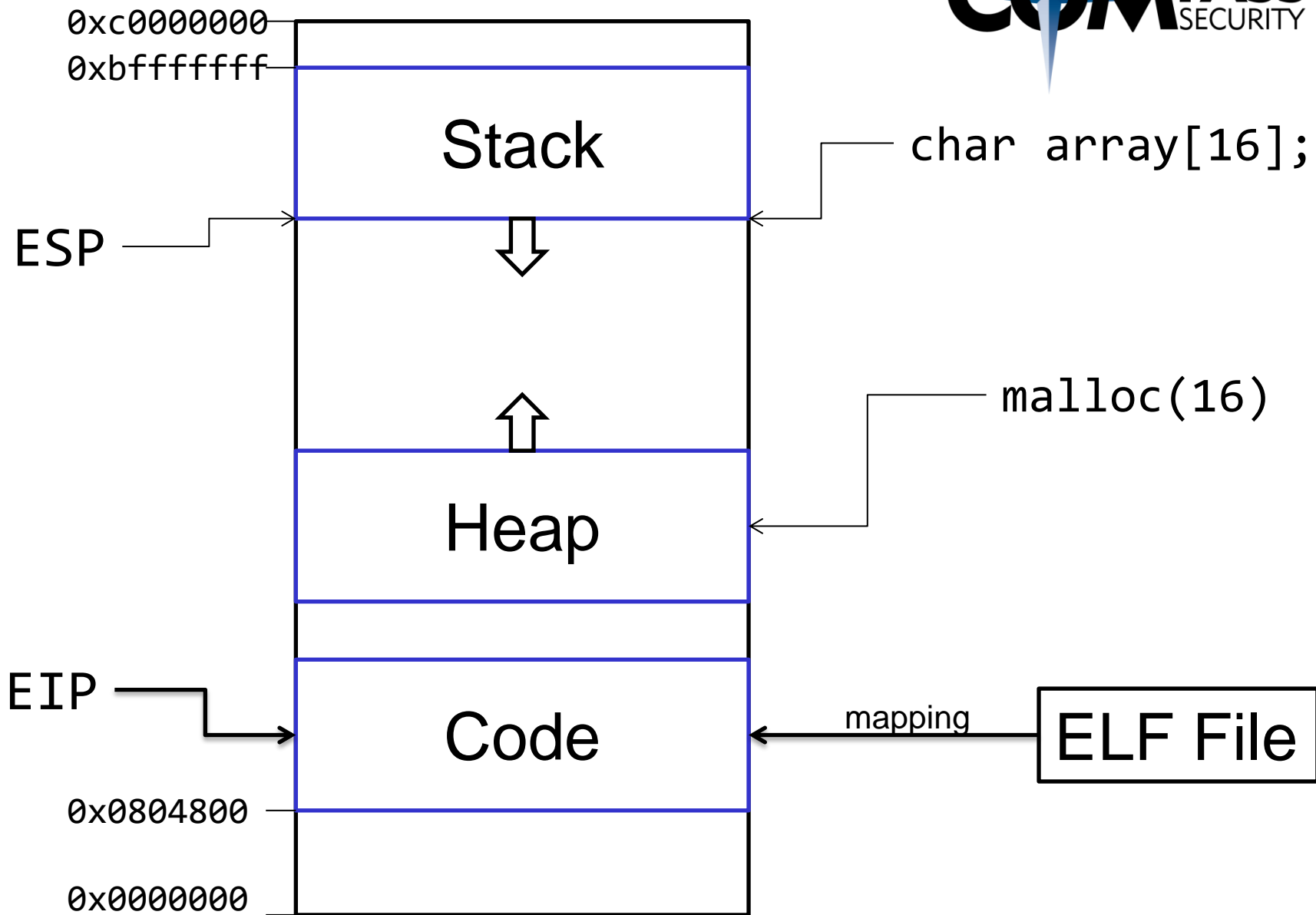
x32 Memory Layout



x32 Memory Layout



x32 Memory Layout



Memory regions:

Stack

- ✦ There's one contiguous memory region containing the stack for the process
- ✦ LIFO – Last in, First Out
- ✦ Contains function **local variables**
- ✦ Also contains: **Saved Instruction Pointer (SIP)**
- ✦ Current function adds data to the top (bottom) of the stack

Heap

- ✦ There's one contiguous memory region containing the heap
- ✦ Memory allocator returns specific pieces of the memory region
- ✦ For **malloc()**
- ✦ Also contains: heap management data

x32 Memory Layout



Memory regions:

Code

- ✦ Compiled program code

ELF Format

How do programs on disk look like

Programs are stored in ELF files

ELF: Executable and Linkable Format

- ✦ Previously: "a.out" (Linux 1.2)
- ✦ Like COFF, PE (EXE), COM, ...

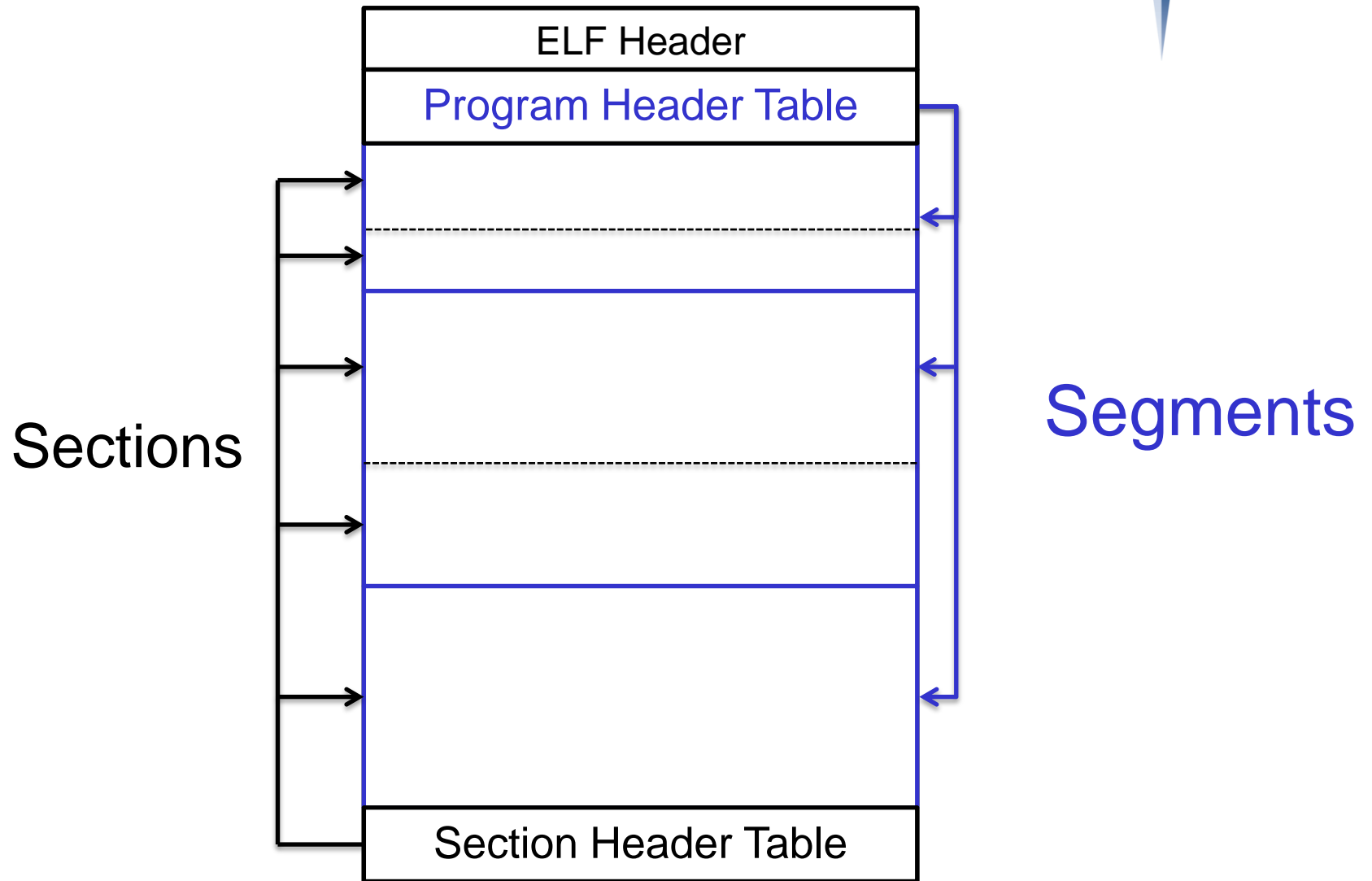
ELF types:

- ✦ ET_EXEC: Executable File
- ✦ ET_REL: Relocatable File
- ✦ ET_DYN: Shared Object File

ELF "views":

- ✦ Sections
- ✦ Segments

\$ readelf -I <binary>



Program Headers:

Type	Offset FileSiz	VirtAddr MemSiz	PhysAddr Flags Align
PHDR	0x00000040 0x000001c0	0x0000400040 0x00000001c0	0x0000000000400040 R E 8
INTERP	0x00000200 0x0000001c	0x0000400200 0x000000001c	0x0000000000400200 R 1
LOAD	0x00000000 0x00000b24	0x0000400000 0x0000000b24	0x0000000000400000 R E 200000
LOAD	0x00000b28 0x00000270	0x0000600b28 0x0000000278	0x0000000000600b28 RW 200000
DYNAMIC	0x00000b40 0x000001e0	0x0000600b40 0x00000001e0	0x0000000000600b40 RW 8
NOTE	0x0000021c 0x00000044	0x000040021c 0x0000000044	0x000000000040021c R 4
GNU_EH_FRAME	0x000009ac 0x00000044	0x00004009ac 0x0000000044	0x00000000004009ac R 4
GNU_STACK	0x00000000 0x00000000	0x0000000000 0x0000000000	0x0000000000000000 RW 10

```
$ readelf -l challenge0
```

Section to Segment mapping:

```
Segment Sections...
```

```
00
```

```
01      .interp
```

```
02
```

```
.interp .note.ABI-tag .note.gnu.build-id .gnu.hash  
.dynsym .dynstr .gnu.version .gnu.version_r  
.rela.dyn .rela.plt .init .plt .text .fini .rodata  
.eh_frame_hdr .eh_frame
```

```
03
```

```
.init_array .fini_array .jcr .dynamic .got .got.plt  
.data .bss
```

```
04      .dynamic
```

```
05      .note.ABI-tag .note.gnu.build-id
```

```
06      .eh_frame_hdr
```

```
07
```

Sections:

- ✦ .text: Executable instructions
- ✦ .bss: Uninitialized data (usually the heap)
- ✦ .data: initialized data
- ✦ .rodata: Read-Only data
- ✦ .got: Global Offset Table
- ✦ .plt: Procedure Linkage Table
- ✦ .init/.fini: Initialization instructions ("glibc")

Program Headers:

Type	Offset	PhysAddr
	FileSiz	Flags Align
(02) LOAD	0x0000000000000000	0x0000000000400000
	0x00000000000000b24	R E 200000
(03) LOAD	0x00000000000000b28	0x0000000000600b28
	0x00000000000000270	RW 200000
(07) GNU_STACK	0x0000000000000000	0x0000000000000000
	0x0000000000000000	RW 10

02 .init .plt .text .fini .rodata

03 .got .got.plt .data .bss

07

A vertical strip on the left side of the slide shows a close-up of a computer keyboard with a yellow padlock resting on one of the keys.

ELF Loader

Compass Security Schweiz AG
Werkstrasse 20
Postfach 2038
CH-8645 Jona

Tel +41 55 214 41 60
Fax +41 55 214 41 61
team@csnc.ch
www.csnc.ch

ELF Header
Program Header Table
.plt
.text
.init
.got
.data
.bss
Section Header Table

02 Executable Segment

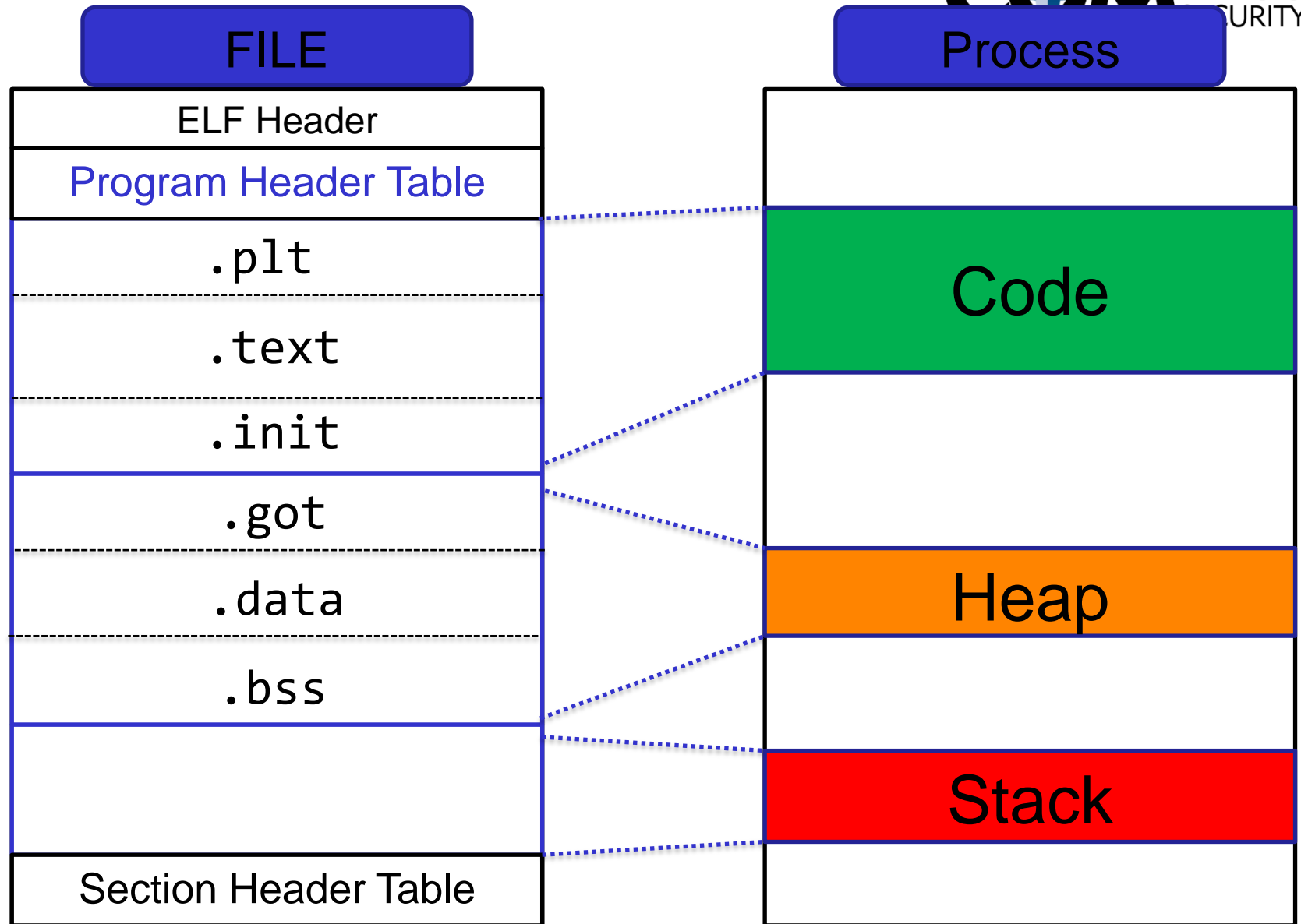
r-X

03 Data Segment

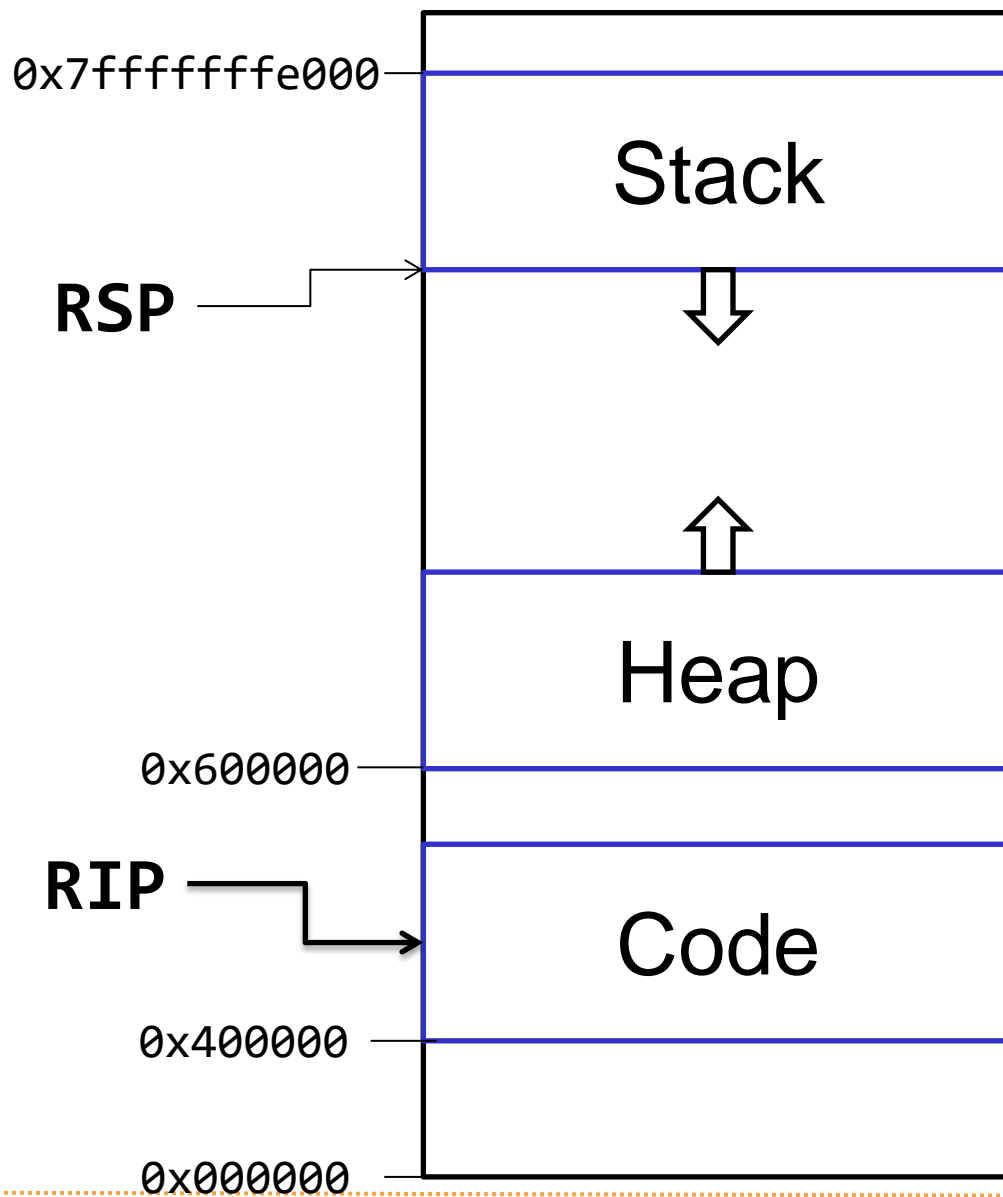
rw-

07 Stack

rw-



x64 Memory Layout



A vertical strip on the left side of the slide shows a close-up of a computer keyboard with a magnifying glass resting on it, highlighting the keys. The image is in a light blue and white color scheme.

Lets do an example
some static and dynamic binary
analysis

```
char *globalVar = "Global";

void main(void) {
    char stackVar[16];
    char *heapVar = (char *) malloc(4);

    printf("Global var: %p\n", globalVar);
    printf("Heap var: %p\n", heapVar);
    printf("Stack var: %p\n", stackVar);
}
```

Global var: **0x400654**

Heap var: **0x601010**

Stack var: **0x7fffffffefe990**

(2) **LOAD** **0x000000000000400000**

R E 200000

(3) **LOAD** **0x000000000000600b28**

RW 200000

(7) **GNU_STACK** **0x000000000000000000**

RW 10

See it at runtime

```
# cat /proc/self/maps
```

```
00400000-0040c000 r-xp 00000000 08:01 391694 /bin/cat
```

```
0060b000-0060c000 r--p 0000b000 08:01 391694 /bin/cat
```

```
0060c000-0060d000 rw-p 0000c000 08:01 391694 /bin/cat
```

```
...
```

```
7fffffffde000-7fffffffff000 rw-p 00000000 00:00 0 [stack]
```

Show Code section, and disassemble:

```
$ objdump -d ./challenge1
```

```
./challenge1:      file format elf64-x86-64
```

```
Disassembly of section .init:
```

```
0000000000400588 <_init>:
```

```
...
```

```
000000000040077f <handleData>:
```

40077f:	55	push	%rbp
400780:	48 89 e5	mov	%rsp,%rbp
400783:	48 83 ec 30	sub	\$0x30,%rsp
400787:	48 89 7d d8	mov	%rdi,-0x28(%rbp)
40078b:	48 89 75 d0	mov	%rsi,-0x30(%rbp)

The process of creating a process from an ELF file is called:

- ✦ "Linking and Loading"

Sections:

- ✦ Are for compiler (gcc), to link several object files together (.o)

Segments:

- ✦ Are for the loader, to create the process
- ✦ Consists of one or more sections

Recap:

- ✦ Program Code is stored in ELF Files
- ✦ ELF Files contain segments
- ✦ Segments are copied 1:1 in the memory

Challenges:

<https://exploit.courses>

- ✦ Challenge 0: Introduction to memory layout – basic
- ✦ Challenge 1: Introduction to memory layout - advanced