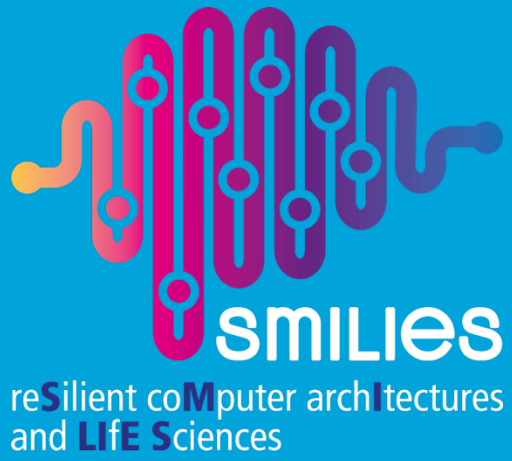


Politecnico di Torino

Department of Control and Computer Engineering

THE ELF FILE FORMAT

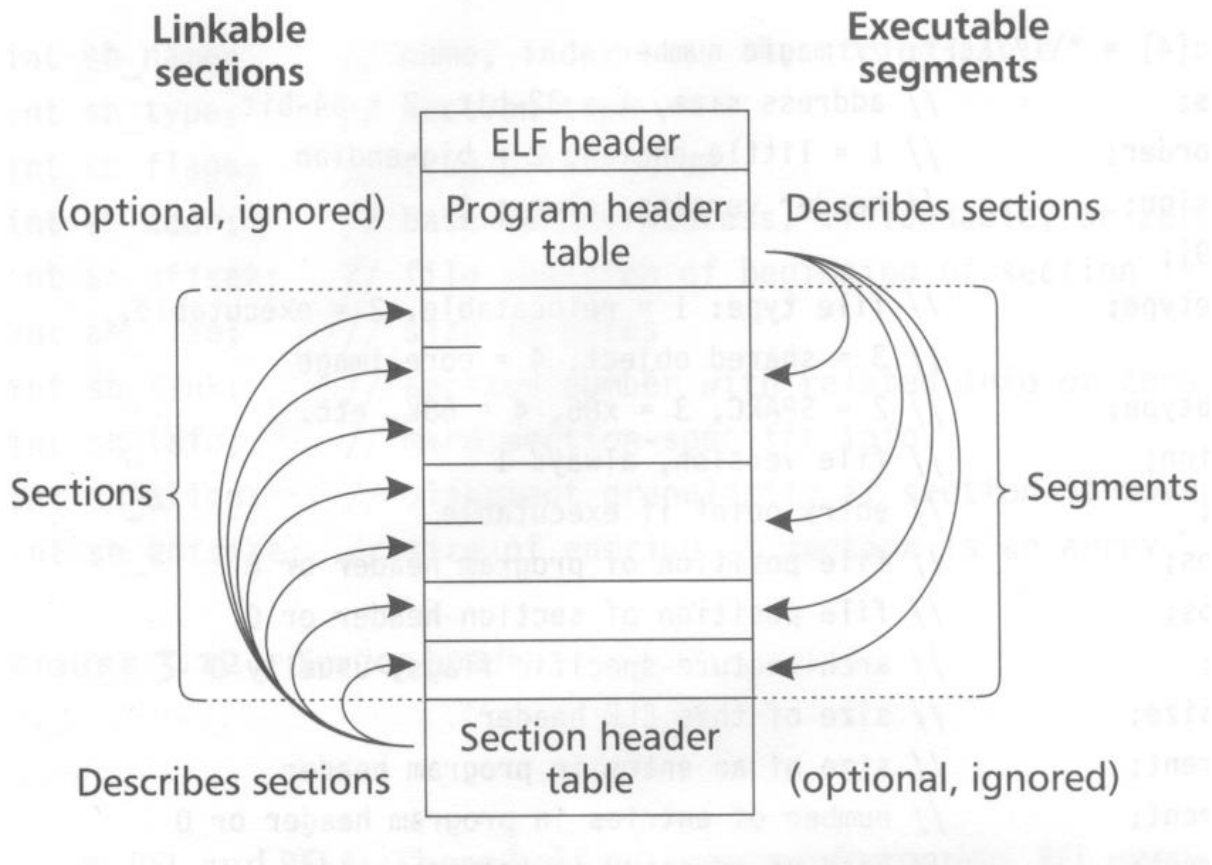
STEFANO DI CARLO

ELF FILE FORMAT

- The a.out (https://en.wikipedia.org/wiki/A.out) format served the Unix community well for over 10 years.
- However, to **better support cross-compilation**, dynamic linking, initializer/finalizer (e.g., the constructor and destructor in C++) and other advanced system features, a.out has been replaced by the **elf** file format.
- Elf stands for "**Executable and Linking Format**."
- Elf has been adopted by **FreeBSD and Linux** as the current standard.

ELF STRUCTURE DUAL NATURE

- Compilers, assemblers, and linkers treat the file as a set of **logical sections** described by a section header table.
- The system loader treats the file as a set of **segments** described by a program header table.
  - A single segment usually consist of several sections. E.g., a loadable read-only segment could contain sections for executable code, read-only data, and symbols for the dynamic linker.
- Relocatable files have section header tables. Executable files have program header tables. Shared object files have both.
- Sections are intended for further processing by a linker, while the segments are intended to be mapped into memory.



ELF FILE TYPES

- Elf defines the format of executable binary files.
- There are four different types:
  - **Relocatable:** created by compilers or assemblers. Need to be processed by the linker before running.
  - **Shared object:** shared library containing both symbol information for the linker and directly runnable code for run time.
  - **Executable:** have all relocation done and all symbol resolved except perhaps shared library symbols that must be resolved at run time.
  - **Core file:** a core dump file.

RELOCATABLE FILES

- A relocatable or **object file** is a collection of sections.
- Each section contains a single type of information, such as:
  - program code
  - read-only data,
  - read/write data
  - relocation entries: records used to adjust addresses and symbols in the program during linking
  - Symbols, i.e., a description of variables or functions stored in the ELF file including simple information such as size, value, name, etc.
- Every address is defined relative to a section
  - Therefore, a procedure's entry point is relative to the program code section that contains that procedure's code.

SHARED OBJECTS

- A shared object, typically with a .so extension (e.g., libxyz.so), is a file that contains code and data intended to be shared by multiple programs at runtime.
- It is commonly used for dynamic linking, where a program can load and link to these shared libraries either at load time or during execution.
- The key characteristics are:
  - It contains relocatable code, allowing it to be loaded at different memory addresses.
  - It includes exported symbols (functions or variables) that can be referenced by other programs.
  - It reduces memory usage and binary size by sharing common code across multiple programs.

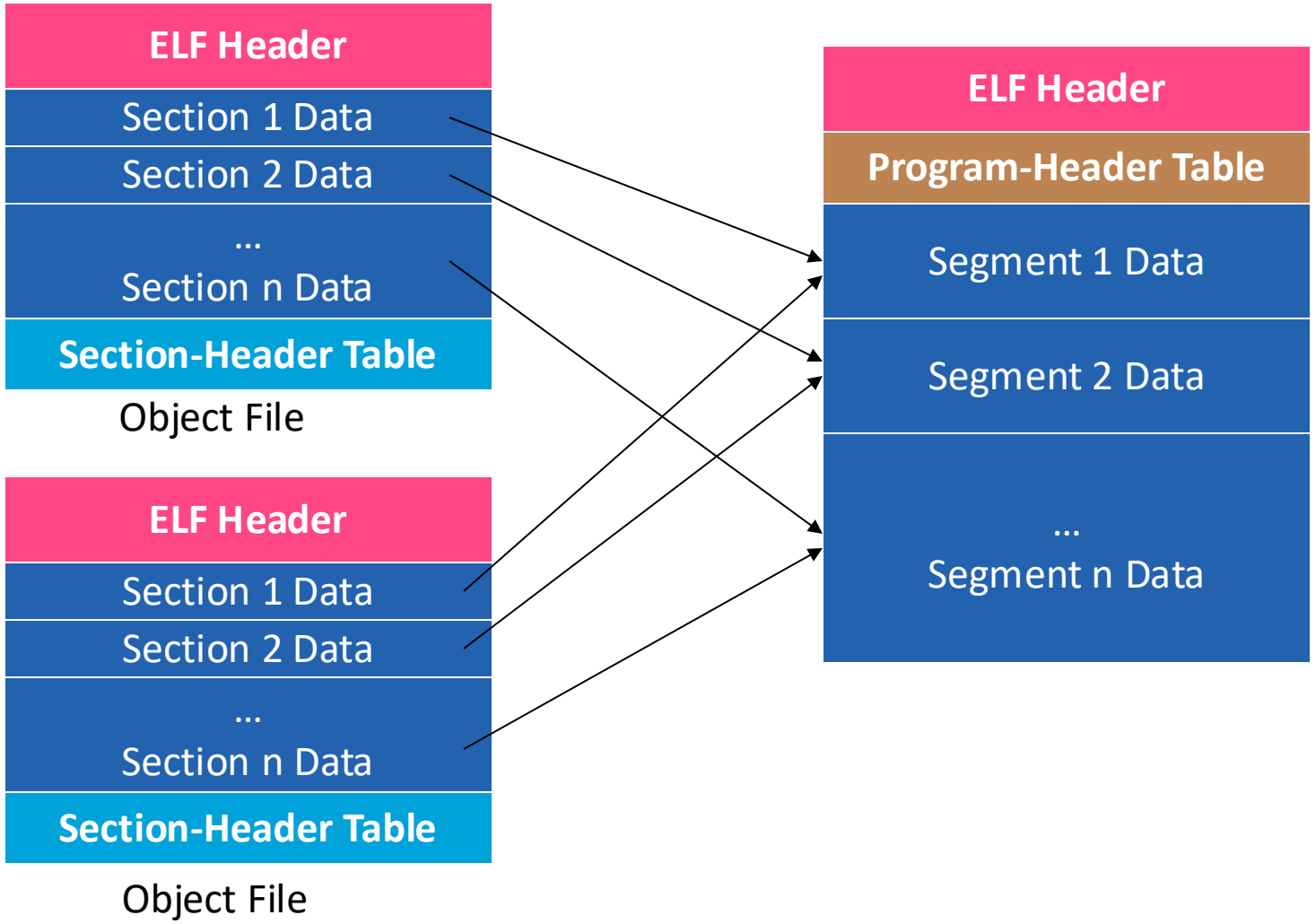
EXECUTABLES

- An executable is very similar to a shared object
  - It can be loaded at a specific address in memory.
  - It has a function that is called when a program starts.
  - _start() run first in an executable.

CORE FILES

- An ELF core file is a special type of file that contains a snapshot of a program's memory and execution state at the moment it crashes or is terminated unexpectedly.
- It is typically generated by the operating system when a program encounters a serious error, such as a segmentation fault.
- Key features of an ELF core file:
  - **Process State:** It includes the program's memory (stack, heap, data segments) and processor registers at the time of the crash.
  - **Debugging Use:** Core files are used for post-mortem debugging, allowing developers to analyze the state of the program when it crashed to determine the cause of the error.
  - Core files are often analyzed using debugging tools like gdb to trace the cause of a crash and diagnose software bugs.

ELF LINKING PROCESS



ELF HEADER

- The Elf header is always at offset zero of the file.
- The program header table and the section header table's offset in the file are defined in the ELF header.
- The elf format can support two different address sizes:
  - 32 bits
  - 64 bits

```
typedef struct
{
  unsigned char e_ident[EI_NIDENT]; /* Magic number and other info */
  Elf64_Half    e_type;             /* Object file type */
  Elf64_Half    e_machine;          /* Architecture */
  Elf64_Word    e_version;          /* Object file version */
  Elf64_Addr    e_entry;            /* Entry point virtual address */
  Elf64_Off e_phoff;                /* Program header table file offset */
  Elf64_Off e_shoff;                /* Section header table file offset */
  Elf64_Word    e_flags;            /* Processor-specific flags */
  Elf64_Half    e_ehsize;           /* ELF header size in bytes */
  Elf64_Half    e_phentsize;        /* Program header table entry size */
  Elf64_Half    e_phnum;            /* Program header table entry count */
  Elf64_Half    e_shentsize;        /* Section header table entry size */
  Elf64_Half    e_shnum;            /* Section header table entry count */
  Elf64_Half    e_shstrndx;         /* Section header string table index */
} Elf64_Ehdr;
```

ELF HEADER

Name	Value	Purpose
EI_MAG0	0	File identification
EI_MAG1	1	File identification
EI_MAG2	2	File identification
EI_MAG3	3	File identification
EI_CLASS	4	File class
EI_DATA	5	Data encoding
EI_VERSION	6	File version
EI_PAD	7	Start of padding bytes
EI_NIDENT	16	Size of e_ident[]

e_ident

```
linuxias@ubuntu:~/ELF/exam$ hexdump -C test | head -4
00000000  7f 45 4c 46 02 01 01 00  00 00 00 00 00 00 00 00  |.ELF............|
00000010  02 00 3e 00 01 00 00 00  40 04 40 00 00 00 00 00  |..>.....@.@.....|
00000020  40 00 00 00 00 00 00 00  a0 11 00 00 00 00 00 00  |@...............|
00000030  00 00 00 00 40 00 38 00  09 00 40 00 1e 00 1b 00  |....@.8...@.....|
```

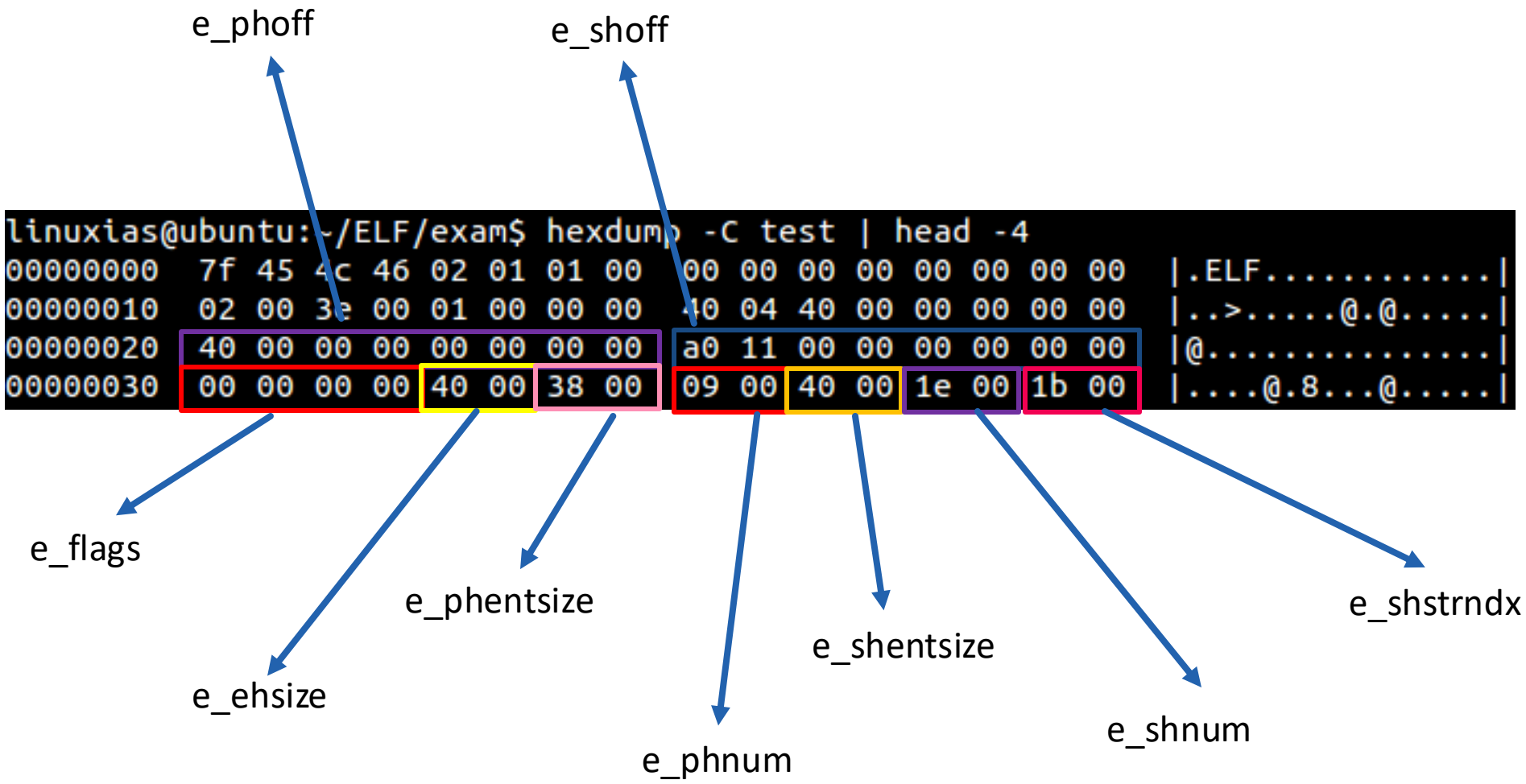
e_type

e_machine

e_version

e_entry

ELF HEADER


e_phoff
e_shoff
linuxias@ubuntu:~/ELF/exam$ hexdump -C test | head -4
00000000 7f 45 4c 46 02 01 01 00 00 00 00 00 00 00 00 00 |.ELF............|
00000010 02 00 3e 00 01 00 00 00 40 04 40 00 00 00 00 00 |..>.....@.@.....|
00000020 40 00 00 00 00 00 00 00 a0 11 00 00 00 00 00 00 |@...............|
00000030 00 00 00 00 40 00 38 00 09 00 40 00 1e 00 1b 00 |....@.8...@.....|
e_flags
e_ehsize
e_phentsize
e_phnum
e_shentsize
e_shnum
e_shstrndx

ELF HEADER

variable	Size(byte)	Data(Hex [Dex])
e_ident	16	7f 45 4c 46 02 01 01 00
e_type	2	2 [2]
e_machine	2	3e [64]
e_version	4	1 [1]
e_entry	8	40 04 40 [4,195,392]
e_phoff	8	40 [64]
e_shoff	8	11 a0 [4512]
e_flags	4	0 [0]
e_ehsize	2	40 [64]
e_phentsize	2	38 [56]
e_phnum	2	9 [9]
e_shentsize	2	40 [64]
e_shnum	2	1e [30]
e_shstrndx	2	1b [27]

```
linuxias@ubuntu:~/ELF/exam$ readelf -h test
ELF Header:
  Magic:   7f 45 4c 46 02 01 01 00 00 00 00 00 00 00 00 00
  Class:                             ELF64
  Data:                              2's complement, little endian
  Version:                           1 (current)
  OS/ABI:                            UNIX - System V
  ABI Version:                       0
  Type:                              EXEC (Executable file)
  Machine:                           Advanced Micro Devices X86-64
  Version:                           0x1
  Entry point address:               0x400440
  Start of program headers:          64 (bytes into file)
  Start of section headers:          4512 (bytes into file)
  Flags:                             0x0
  Size of this header:               64 (bytes)
  Size of program headers:           56 (bytes)
  Number of program headers:         9
  Size of section headers:           64 (bytes)
  Number of section headers:         30
  Section header string table index: 27
```

SECTION HEADER TABLE

- Contains information about every part of an ELF file (except the ELF Header, Program Header Table , Section Header Table itself).
- List of section header structures, each defining a different section in the ELF file.

```
typedef struct
{
  Elf64_Word    sh_name;        /* Section name (string tbl index) */
  Elf64_Word    sh_type;        /* Section type */
  Elf64_Xword   sh_flags;       /* Section flags */
  Elf64_Addr    sh_addr;        /* Section virtual addr at execution */
  Elf64_Off sh_offset;          /* Section file offset */
  Elf64_Xword   sh_size;        /* Section size in bytes */
  Elf64_Word    sh_link;        /* Link to another section */
  Elf64_Word    sh_info;        /* Additional section information */
  Elf64_Xword   sh_addralign;       /* Section alignment */
  Elf64_Xword   sh_entsize;     /* Entry size if section holds table */
} Elf64_Shdr;
```

SECTION HEADER TABLE

- ELF header contains the file offset of the section header table

```
linuxias@ubuntu:~/ELF/exam$ readelf -h test.o
ELF Header:
  Magic:   7f 45 4c 46 02 01 01 00 00 00 00 00 00 00 00 00
  Class:                             ELF64
  Data:                              2's complement, little endian
  Version:                           1 (current)
  OS/ABI:                            UNIX - System V
  ABI Version:                       0
  Type:                              REL (Relocatable file)
  Machine:                           Advanced Micro Devices X86-64
  Version:                           0x1
  Entry point address:               0x0
  Start of program headers:          0 (bytes into file)
  Start of section headers:          384 (bytes into file)
  Flags:                             0x0
  Size of this header:               64 (bytes)
  Size of program headers:           0 (bytes)
  Number of program headers:         0
  Size of section headers:           64 (bytes)
  Number of section headers:         13
  Section header string table index: 10
```

SECTIONS AND THE SECTION HEADER TABLE

```
int list[10] = { 0, 1, 2, 3, 4, 5, 6, 7, 8, 9 };
```

```
linuxias@ubuntu:~/ELF$ nm -v -f s foo.o | grep list
list                |0000000000000020|   D  |            OBJECT|0000000000000028|     |.data
```

```
linuxias@ubuntu:~/ELF$ readelf -S foo.o | grep "\.data "
  [ 5] .data             PROGBITS         0000000000000000  00000120
```

- File offset 0x140
  (.data section offset 0x120 + list value offset 0x20)

```
linuxias@ubuntu:~/ELF$ hexdump -C -s 0x140 foo.o | head -4
00000140  00 00 00 00 01 00 00 00  02 00 00 00 03 00 00 00  |................|
00000150  04 00 00 00 05 00 00 00  06 00 00 00 07 00 00 00  |................|
00000160  08 00 00 00 09 00 00 00  55 48 89 e5 48 89 7d f8  |........UH..H.}.|
```

FLAGS IN SECTION HEADER

- WRITE: This section contains data that is writable during process execution.
- ALLOC: This section occupies memory during process execution.
- EXECINSTR: This section contains executable machine instructions.

VARIOUS SECTIONS

- .text:
  - This section holds executable instructions of a program.
  - Type: PROGBITS
  - Flags: ALLOC + EXECINSTR
- .data:
  - This section holds initialized data that contributes to the program’s image.
  - Type: PROGBITS
  - Flags: ALLOC + WRITE

VARIOUS SECTIONS

- .rodata:
  - This section holds read-only data.
  - Type: PROGBITS
  - Flags: ALLOC
- .bss :
  - This section holds uninitialized data that contributed to the program’s image. By definition, the system will initialize the data with zero when the program begins to run.
  - Type: NOBITS
  - Flags: ALLOC + WRITE

VARIOUS SECTIONS

- .rel.text, .rel.data, and .rel.rodata:
  - These contain the relocation information for the corresponding text or data sections.
  - Type: REL
  - Flags: ALLOC is turned on if the file has a loadable segment that includes relocation.
- .symtab:
  - This section hold a symbol table.
- .strtab:
  - This section holds strings.

VARIOUS SECTIONS

- .init:
  - This section holds executable instructions that contribute to the process initialization code.
  - Type: PROGBITS
  - Flags: ALLOC + EXECINSTR
- .fini:
  - This section hold executable instructions that contribute to the process termination code.
  - Type: PROGBITS
  - Flags: ALLOC + EXECINSTR
- C does not need these two sections. However, C++ needs them.

VARIOUS SECTIONS

- .interp:
  - This section holds the pathname of a program interpreter.
  - Type: ALLOC
  - Flags: PROGBITS
  - If this section is present, rather than running the program directly, the system runs the interpreter and passes it the elf file as an argument.
  - For many years (used in a.out), UNIX has had self-running interpreted text files, using
  - #! /bin/csh as the first line of the file.
  - Elf extends this facility to interpreters that run nontext programs.
  - In practice, this is used to run the run-time dynamic linker to load the program and to link in any required shared libraries.

VARIOUS SECTIONS

- .debug:
  - This section holds symbolic debugging information.
  - Type: PROGBIT
- .line:
  - This section holds line number information for symbolic debugging, which describes the correspondence between the program source and the machine code (ever used gdb?)
  - Type: PROGBIT
- .comment
  - This section may store extra information.

VARIOUS SECTIONS

- .got:
  - This section holds the global offset table.
    - We will explain got when we present shared library.
  - Type: PROGBIT
- .plt:
  - This section holds the procedure linkage table.
  - Type: PROGBIT
- .note:
  - This section contains some extra information.

A TYPICAL RELOCATABLE FILE.

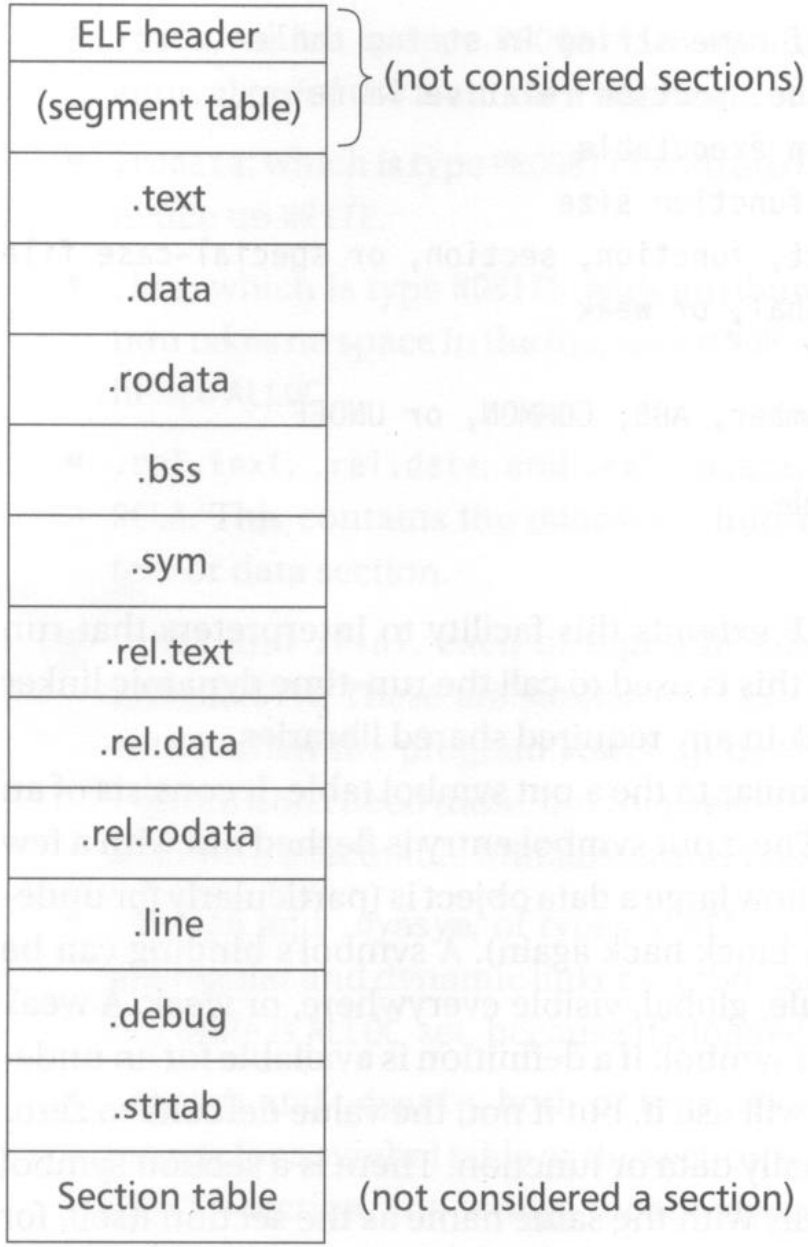


FIGURE 3.14 • Sample relocatable ELF file.

STRING TABLE

- String table sections hold null-terminated character sequences, commonly called strings.
- The object file uses these strings to represent symbol and section names.
- We use an index into the string table section to reference a string.
- The reason why we separate symbol names from symbol tables is that in C or C++, there is no limitation on the length of a symbol.

SYMBOL TABLE

- An object file's symbol table holds information needed to locate and relocate a program's symbolic definition and references.
- A symbol table index is a subscript into this array.

```
int name;     // position of name string in string table
int value;    // symbol value, section relative in reloc,
              // absolute in executable
int size;     // object or function size
char type:4;  // data object, function, section, or special-case file
char bind:4;  // local, global, or weak
char other;   // spare
short sect;   // section number, ABS, COMMON, or UNDEF
```

The section relative to which the symbol is defined. (e.g., the function entry points are defined relative to .text)

If a definition is available for an undefined weak symbol, the linker will use it. Otherwise, the value defaults to 0.

RELOCATION TABLE

- Relocation is the process of connecting symbolic references with symbolic definitions.
- Relocatable files must have information that describes how to modify their section contents.
- A relocation table consists of many relocation structures.

RELOCATION STRUCTURE

```
Struct {
    R_offset;
```

- This field gives the location at which to apply the relocation.
- For a relocatable file, the value is the byte offset from the beginning of the section to the storage unit affect by the relocation.
- For an executable file and shared object, the value is the virtual address of the storage unit affected by the relocation.

RELOCATION STRUCTURE

```
    R_info;
```

- This field gives both the symbol table index with respect to which the relocation must be made and the type of relocation to apply.

```
    R_addend;
```

- This field specifies a constant addend used to compute the value to be stored into the relocation field.

```
}
```

EXECUTABLE FILES

- An executable file usually has only a few segments. E.g.,
  - A read-only one for the code.
  - A read-only one for read-only data.
  - A read/write one for read/write data.
- All of the loadable sections are packed into the appropriate segments so that the system can map the file with just one or two operations.
  - E.g., If there is a .init and .fini sections, those sections will be put into the read-only text segment.

PROGRAM HEADER

- The **Program Header Table** contains information about the segments in an ELF file and how to load them into memory (segments : contiguous ranges of an ELF file that have the same memory attribution.)

```
typedef struct
{
  Elf64_Word    p_type;         /* Segment type */
  Elf64_Word    p_flags;        /* Segment flags */
  Elf64_Off p_offset;         /* Segment file offset */
  Elf64_Addr    p_vaddr;        /* Segment virtual address */
  Elf64_Addr    p_paddr;        /* Segment physical address */
  Elf64_Xword   p_filesz;       /* Segment size in file */
  Elf64_Xword   p_memsz;        /* Segment size in memory */
  Elf64_Xword   p_align;        /* Segment alignment */
} Elf64_Phdr;
```

THE TYPES IN PROGRAM HEADER

- This field tells what kind of segment this array element describes:
  - PT_LOAD: This segment is a loadable segment.
  - PT_DYNAMIC: This array element specifies dynamic linking information.
  - PT_INTERP: This element specified the location and size of a null-terminated path name to invoke as an interpreter.

PROGRAM HEADER

```
linuxias@ubuntu:~/ELF/exam$ readelf -h test
ELF Header:
  Magic:   7f 45 4c 46 02 01 01 00 00 00 00 00 00 00 00 00
  Class:                             ELF64
  Data:                              2's complement, little endian
  Version:                           1 (current)
  OS/ABI:                            UNIX - System V
  ABI Version:                       0
  Type:                              EXEC (Executable file)
  Machine:                           Advanced Micro Devices X86-64
  Version:                           0x1
  Entry point address:               0x400440
  Start of program headers:          64 (bytes into file)
  Start of section headers:          4512 (bytes into file)
  Flags:                             0x0
  Size of this header:               64 (bytes)
  Size of program headers:           56 (bytes)
  Number of program headers:         9
  Size of section headers:           64 (bytes)
  Number of section headers:         30
  Section header string table index: 27
```

SEGMENTS AND THE PROGRAM HEADER TABLE

- Only used for executable, shared libraries and core files

```
linuxias@ubuntu:~/ELF/exam$ readelf -h test.o
ELF Header:
  Magic:   7f 45 4c 46 02 01 01 00 00 00 00 00 00 00 00 00
  Class:                             ELF64
  Data:                              2's complement, little endian
  Version:                           1 (current)
  OS/ABI:                            UNIX - System V
  ABI Version:                       0
  Type:                              REL (Relocatable file)
  Machine:                           Advanced Micro Devices X86-64
  Version:                           0x1
  Entry point address:               0x0
  Start of program headers:          0 (bytes into file)
  Start of section headers:          384 (bytes into file)
  Flags:                             0x0
  Size of this header:               64 (bytes)
  Size of program headers:           0 (bytes)
  Number of program headers:         0
  Size of section headers:           64 (bytes)
  Number of section headers:         13
  Section header string table index: 10
```

SEGMENTS AND THE PROGRAM HEADER TABLE

```
linuxias@ubuntu:~/ELF/exam$ readelf -l test

Elf file type is EXEC (Executable file)
Entry point 0x400440
There are 9 program headers, starting at offset 64

Program Headers:
  Type           Offset             VirtAddr           PhysAddr
                 FileSiz            MemSiz              Flags  Align
  PHDR           0x0000000000000040 0x0000000000400040 0x0000000000400040
                 0x00000000000001f8 0x00000000000001f8  R E    8
  INTERP         0x0000000000000238 0x0000000000400238 0x0000000000400238
                 0x000000000000001c 0x000000000000001c  R      1
      [Requesting program interpreter: /lib64/ld-linux-x86-64.so.2]
  LOAD           0x0000000000000000 0x0000000000400000 0x0000000000400000
                 0x000000000000075c 0x000000000000075c  R E    200000
  LOAD           0x0000000000000e10 0x0000000000600e10 0x0000000000600e10
                 0x0000000000000238 0x0000000000000240  RW     200000
  DYNAMIC        0x0000000000000e28 0x0000000000600e28 0x0000000000600e28
                 0x00000000000001d0 0x00000000000001d0  RW     8
  NOTE           0x0000000000000254 0x0000000000400254 0x0000000000400254
                 0x0000000000000044 0x0000000000000044  R      4
  GNU_EH_FRAME   0x0000000000000608 0x0000000000400608 0x0000000000400608
                 0x000000000000003c 0x000000000000003c  R      4
  GNU_STACK      0x0000000000000000 0x0000000000000000 0x0000000000000000
                 0x0000000000000000 0x0000000000000000  RW     10
  GNU_RELRO      0x0000000000000e10 0x0000000000600e10 0x0000000000600e10
                 0x00000000000001f0 0x00000000000001f0  R      1

 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
   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
   08     .init_array .fini_array .jcr .dynamic .got
```

SEGMENTS AND THE PROGRAM HEADER TABLE

program header itself.

“INTERP” segment. That only includes the name of the program interpreter

```
linuxias@ubuntu:~/ELF/exam$ readelf -l test

Elf file type is EXEC (Executable file)
Entry point 0x400440
There are 9 program headers, starting at offset 64

Program Headers:
  Type           Offset             VirtAddr           PhysAddr
                 FileSiz            MemSiz              Flags  Align
  PHDR           0x0000000000000040 0x0000000000400040 0x0000000000400040
                 0x00000000000001f8 0x00000000000001f8  R E    8
  INTERP         0x0000000000000238 0x0000000000400238 0x0000000000400238
                 0x000000000000001c 0x000000000000001c  R      1
      [Requesting program interpreter: /lib64/ld-linux-x86-64.so.2]
  LOAD           0x0000000000000000 0x0000000000400000 0x0000000000400000
                 0x000000000000075c 0x000000000000075c  R E    200000
  LOAD           0x0000000000000e10 0x0000000000600e10 0x0000000000600e10
                 0x0000000000000238 0x0000000000000240  RW     200000
  DYNAMIC        0x0000000000000e28 0x0000000000600e28 0x0000000000600e28
                 0x00000000000001d0 0x00000000000001d0  RW     8
  NOTE           0x0000000000000254 0x0000000000400254 0x0000000000400254
                 0x0000000000000044 0x0000000000000044  R      4
  GNU_EH_FRAME   0x0000000000000608 0x0000000000400608 0x0000000000400608
                 0x000000000000003c 0x000000000000003c  R      4
  GNU_STACK      0x0000000000000000 0x0000000000000000 0x0000000000000000
                 0x0000000000000000 0x0000000000000000  RW     10
  GNU_RELRO      0x0000000000000e10 0x0000000000600e10 0x0000000000600e10
                 0x00000000000001f0 0x00000000000001f0  R      1

 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
   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
   08     .init_array .fini_array .jcr .dynamic .got
```

SEGMENTS AND THE PROGRAM HEADER TABLE

program header itself.

“INTERP” segment. That only includes the name of the program interpreter

The executable contains the name of the program interpreter

```
linuxias@ubuntu:~/ELF/exam$ readelf -l test

Elf file type is EXEC (Executable file)
Entry point 0x400440
There are 9 program headers, starting at offset 64

Program Headers:
  Type           Offset             VirtAddr           PhysAddr
                 FileSiz            MemSiz              Flags  Align
  PHDR           0x0000000000000040 0x0000000000400040 0x0000000000400040
                 0x00000000000001f8 0x00000000000001f8  R E    8
  INTERP         0x0000000000000238 0x0000000000400238 0x0000000000400238
                 0x000000000000001c 0x000000000000001c  R      1
      [Requesting program interpreter: /lib64/ld-linux-x86-64.so.2]
  LOAD           0x0000000000000000 0x0000000000400000 0x0000000000400000
                 0x000000000000075c 0x000000000000075c  R E    200000
  LOAD           0x0000000000000e10 0x0000000000600e10 0x0000000000600e10
                 0x0000000000000238 0x0000000000000240  RW     200000
  DYNAMIC        0x0000000000000e28 0x0000000000600e28 0x0000000000600e28
                 0x00000000000001d0 0x00000000000001d0  RW     8
  NOTE           0x0000000000000254 0x0000000000400254 0x0000000000400254
                 0x0000000000000044 0x0000000000000044  R      4
  GNU_EH_FRAME   0x0000000000000608 0x0000000000400608 0x0000000000400608
                 0x000000000000003c 0x000000000000003c  R      4
  GNU_STACK      0x0000000000000000 0x0000000000000000 0x0000000000000000
                 0x0000000000000000 0x0000000000000000  RW     10
                                                       00000000600e10 0x0000000000600e10
                                                       000000000001f0  R      1

   04     .dynamic
   05     .note.ABI-tag .note.gnu.build-id
   06     .eh_frame_hdr
   07
   08     .init_array .fini_array .jcr .dynamic .got
```

```
00000220  f0 01 00 00 00 00 00 00  f0 01 00 00 00 00 00 00  |................|
00000230  01 00 00 00 00 00 00 00  2f 6c 69 62 36 34 2f 6c  |......../lib64/l|
00000240  64 2d 6c 69 6e 75 78 2d  78 38 36 2d 36 34 2e 73  |d-linux-x86-64.s|
00000250  6f 2e 32 00 04 00 00 00  10 00 00 00 01 00 00 00  |o.2.............|
00000260  47 4e 55 00 00 00 00 00  02 00 00 00 06 00 00 00  |GNU.............|
00000270  18 00 00 00 04 00 00 00  14 00 00 00 03 00 00 00  |................|
```

```
u.build-id .gnu.hash .dynsym .dynstr .gnu.version .gnu.version
dynamic .got .got.plt .data .bss
```

SEGMENTS AND THE PROGRAM HEADER TABLE

DYNAMIC segment used for dynamic linking.

Special segments for Vendor-specific information

```
linuxias@ubuntu:~/ELF/exam$ readelf -l test

Elf file type is EXEC (Executable file)
Entry point 0x400440
There are 9 program headers, starting at offset 64

Program Headers:
  Type           Offset             VirtAddr           PhysAddr
                 FileSiz            MemSiz              Flags  Align
  PHDR           0x0000000000000040 0x0000000000400040 0x0000000000400040
                 0x00000000000001f8 0x00000000000001f8  R E    8
  INTERP         0x0000000000000238 0x0000000000400238 0x0000000000400238
                 0x000000000000001c 0x000000000000001c  R      1
      [Requesting program interpreter: /lib64/ld-linux-x86-64.so.2]
  LOAD           0x0000000000000000 0x0000000000400000 0x0000000000400000
                 0x000000000000075c 0x000000000000075c  R E    200000
  LOAD           0x0000000000000e10 0x0000000000600e10 0x0000000000600e10
                 0x0000000000000238 0x0000000000000240  RW     200000
  DYNAMIC        0x0000000000000e28 0x0000000000600e28 0x0000000000600e28
                 0x00000000000001d0 0x00000000000001d0  RW     8
  NOTE           0x0000000000000254 0x0000000000400254 0x0000000000400254
                 0x0000000000000044 0x0000000000000044  R      4
  GNU_EH_FRAME   0x0000000000000608 0x0000000000400608 0x0000000000400608
                 0x000000000000003c 0x000000000000003c  R      4
  GNU_STACK      0x0000000000000000 0x0000000000000000 0x0000000000000000
                 0x0000000000000000 0x0000000000000000  RW     10
  GNU_RELRO      0x0000000000000e10 0x0000000000600e10 0x0000000000600e10
                 0x00000000000001f0 0x00000000000001f0  R      1

 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
   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
   08     .init_array .fini_array .jcr .dynamic .got
```

SEGMENTS AND THE PROGRAM HEADER TABLE

LOAD segment
: Loadable program segment

Note Segment

: The array element specifies the location and size of auxiliary information

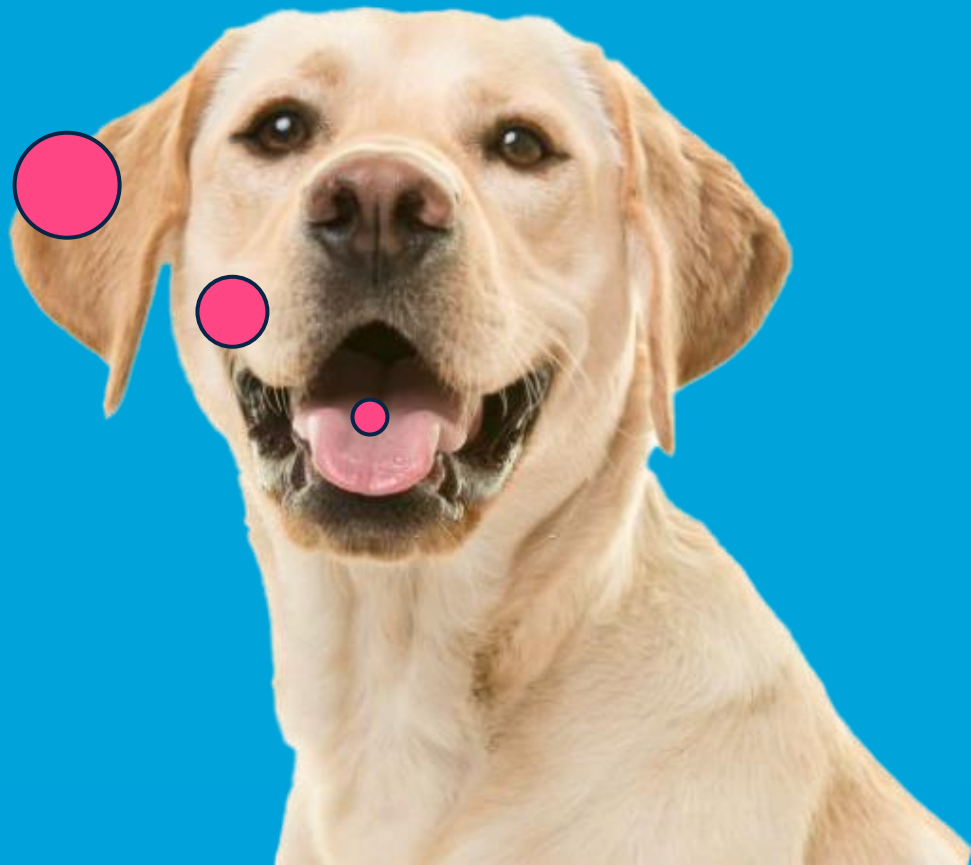
```
linuxias@ubuntu:~/ELF/exam$ readelf -l test

Elf file type is EXEC (Executable file)
Entry point 0x400440
There are 9 program headers, starting at offset 64

Program Headers:
  Type           Offset             VirtAddr           PhysAddr
                 FileSiz            MemSiz              Flags  Align
  PHDR           0x0000000000000040 0x0000000000400040 0x0000000000400040
                 0x00000000000001f8 0x00000000000001f8  R E    8
  INTERP         0x0000000000000238 0x0000000000400238 0x0000000000400238
                 0x000000000000001c 0x000000000000001c  R      1
      [Requesting program interpreter: /lib64/ld-linux-x86-64.so.2]
  LOAD           0x0000000000000000 0x0000000000400000 0x0000000000400000
                 0x000000000000075c 0x000000000000075c  R E    200000
  LOAD           0x0000000000000e10 0x0000000000600e10 0x0000000000600e10
                 0x0000000000000238 0x0000000000000240  RW     200000
  DYNAMIC        0x0000000000000e28 0x0000000000600e28 0x0000000000600e28
                 0x00000000000001d0 0x00000000000001d0  RW     8
  NOTE           0x0000000000000254 0x0000000000400254 0x0000000000400254
                 0x0000000000000044 0x0000000000000044  R      4
  GNU_EH_FRAME   0x0000000000000608 0x0000000000400608 0x0000000000400608
                 0x000000000000003c 0x000000000000003c  R      4
  GNU_STACK      0x0000000000000000 0x0000000000000000 0x0000000000000000
                 0x0000000000000000 0x0000000000000000  RW     10
  GNU_RELRO      0x0000000000000e10 0x0000000000600e10 0x0000000000600e10
                 0x00000000000001f0 0x00000000000001f0  R      1

 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
   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
   08     .init_array .fini_array .jcr .dynamic .got
```

SMILIES
reSilient coMputer archItectures and LIfE Sciences

Politecnico di Torino
Department of Control and Computer Engineering

DAUIN

QUESTIONS?



THANK YOU!