Reverse Engineering #0

TwinkleStar03

2023/10/20

TwinkleStar03

whoami

- TwinkleStar03 / 星星
  - RE/Pwn @ XxTSJxX
  - Member of UNDEFINED Conclave
- ~~大權~~ 比賽經驗
  - FLARE-On Challenge 9 Finisher
  - DC30 CTF Finalist, DC31 CTF Final 3rd
- https://blog.star03.me/about/

這堂課

- 這堂課只會講到 Linux 下 x86-64 ELF Binary
  - 架構太多太複雜，運用技巧也不盡相同
- 透過此堂課大致了解程式從編譯到運行，然後反其道而行

RE 課程安排

- 第一週
  - Linux ELF SRE
- 第二週
  - Windows PE Malware SRE
- 第三週
  - 奇技淫巧：Anti-debug, Advance Tool, 更多的實作

Outline

- Introduction
- Tools
- ELF Executable to Process
- x86-64 assembly
- Calling Convention
- Stack Frame
- Memory Layout
- Endianness
- Compiler Optimization
- Getting Foothold

逆向工程是什麼?可以吃嗎?

What? When? Where? How?

- 逆向工程 From Wiki:

> **逆向工程**（英語：Reverse Engineering），又稱**反向工程**，是一種技術仿造過程，即對一專案標產品進行逆向分析及研究，從而演繹並得出該產品的處理流程、組織結構、功能效能規格等設計要素，以製作出功能相近，但又不完全一樣的產品。逆向工程源於商業及軍事領域中的硬體分析。其主要目的是，在無法輕易獲得必要的生產資訊下，直接從成品的分析，推導產品的設計原理。

What? When? Where? How?

- 正常寫程式的流程
  - 有一個想法
  - 寫成程式碼
  - 編譯成執行檔並執行

What? When? Where? How?

- 一個逆向工程師出生了

What? When? Where? How?

- 逆向的過程
  - 取得執行檔
  - 使用工具看懂內部邏輯
  - 根據過往經驗猜測並推斷作者想法

What? When? Where? How?

- 何時需要逆向工程？
  - 合法途徑
    - 漏洞研究
    - 惡意程式分析
    - 軟體行為分析
    - 開發軟體應用

What? When? Where? How?

- 何時需要逆向工程？
  - 非法途徑
    - 破解軟體序號機
    - 仿製別家的產品

What? When? Where? How?

- 在哪裡逆向工程？
  - 一個使自己心靈平靜的地方
  - 需要一台可以用的電腦

What? When? Where? How?

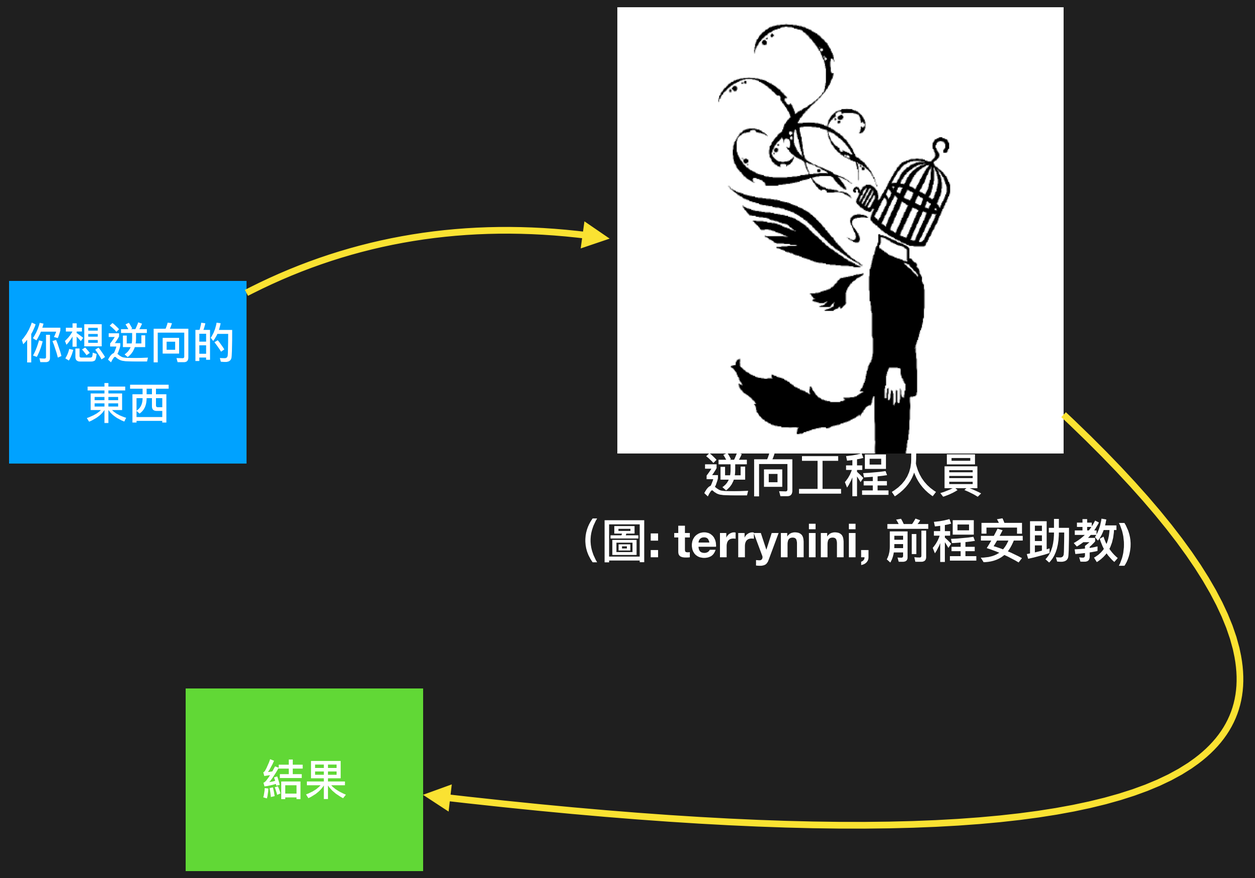
- 在哪裡逆向工程？
  - 一個使自己心靈平靜的地方
  - 需要一台可以用的電腦

What? When? Where? How?

- 那要怎麼開始逆向工程？

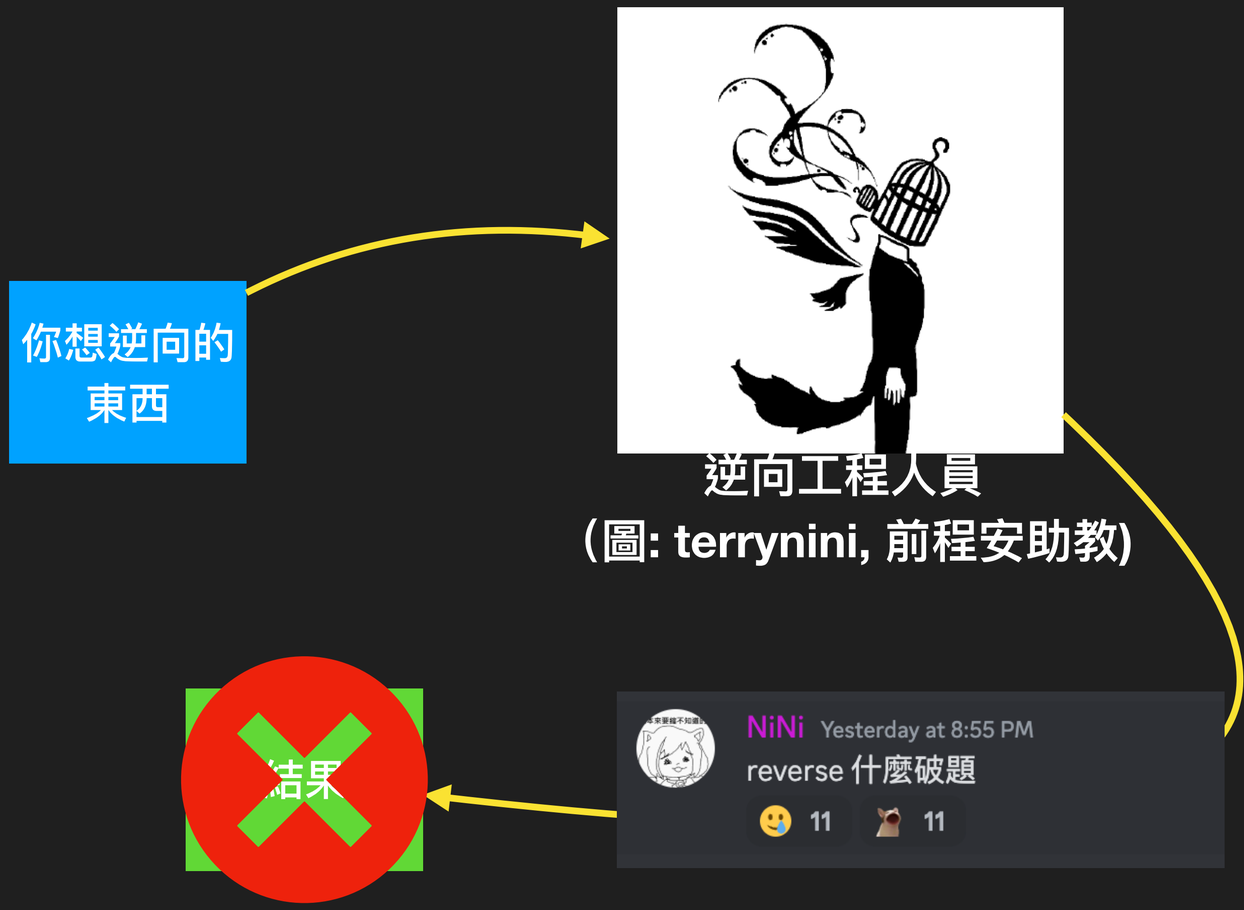
What? When? Where? How?

- 那要怎麼開始逆向工程？
  - 請朋友幫忙



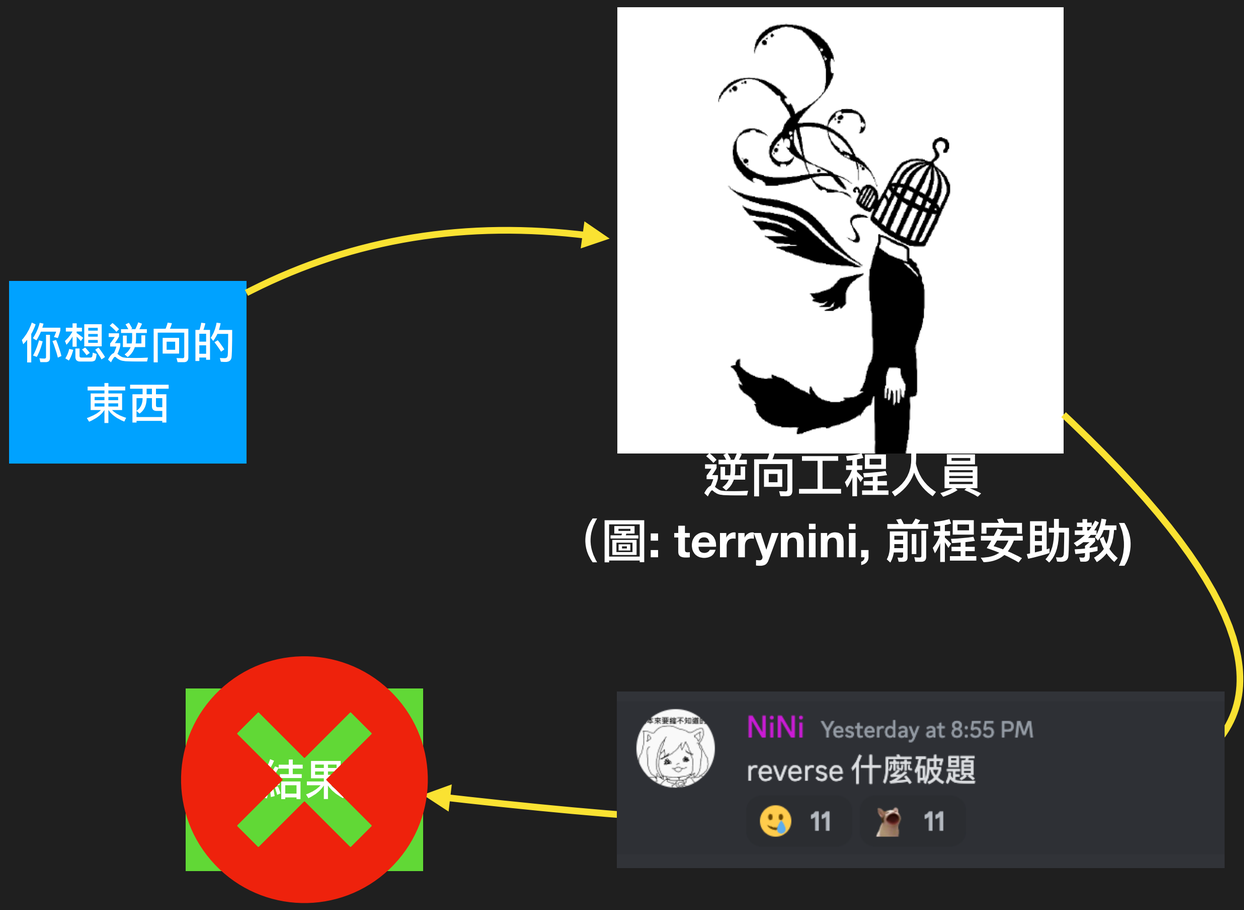
What? When? Where? How?

- 那要怎麼開始逆向工程？
  - ~~請朋友幫忙~~



What? When? Where? How?

- 那要怎麼開始逆向工程？
  - ~~請朋友幫忙~~
  - 好好聽課寫作業



Executable & Linkable Format

Program / Process

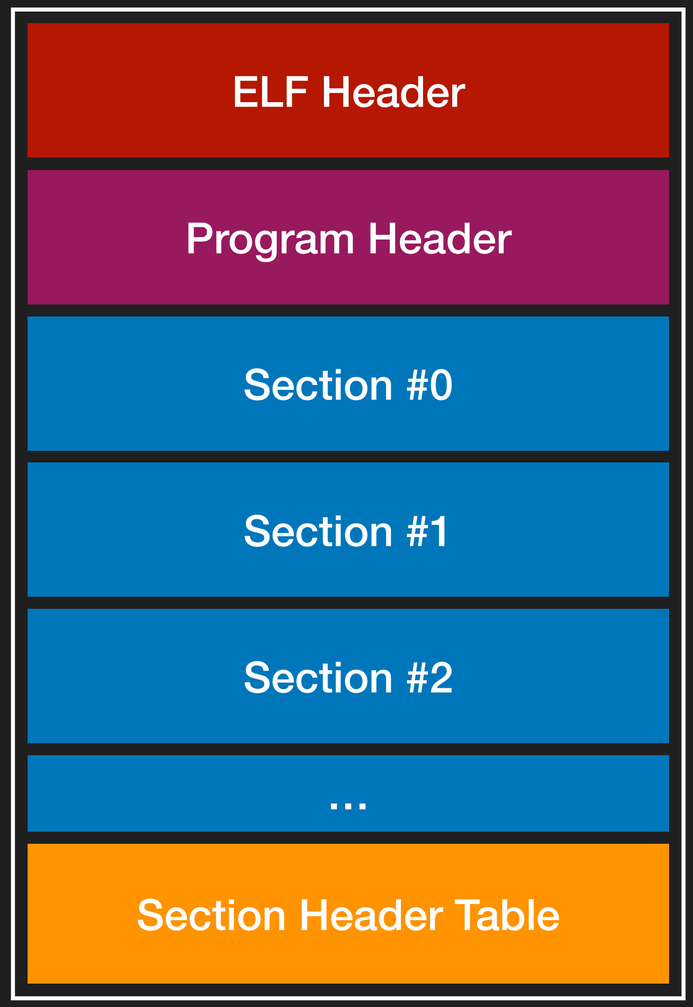
- Program / Image
  - 程式在硬碟上的樣子
  - 記載 Loader 所需的資訊
- Process
  - 程式映射到記憶體上運行，由 OS 進行管理的單位

Program / Process

- Linux 上的 Program 格式就是 Executable & Linkable Format
- Windows 上面的是 Portable Executable
- Mac 上面的是 Mach-O

ELF Overview

- Major Components
  - ELF Header
  - Program Header
  - Sections
  - Section Header Table
- 我們來看看 Raw Data 怎麼 Map
  - 結構都可以在 elf.h 上面找到



ELF Header (Ehdr)

```
typedef struct {
    unsigned char e_ident[EI_NIDENT];
    uint16_t      e_type;
    uint16_t      e_machine;
    uint32_t      e_version;
    ElfN_Addr     e_entry;
    ElfN_Off      e_phoff;
    ElfN_Off      e_shoff;
    uint32_t      e_flags;
    uint16_t      e_ehsize;
    uint16_t      e_phentsize;
    uint16_t      e_phnum;
    uint16_t      e_shentsize;
    uint16_t      e_shnum;
    uint16_t      e_shstrndx;
} ElfN_Ehdr;
```

ELF Header (Ehdr)

```
00000000: 7f45 4c46 0201 0100 0000 0000 0000 0000  .ELF............
00000010: 0300 3e00 0100 0000 6010 0000 0000 0000  ..>.....`.......
00000020: 4000 0000 0000 0000 a036 0000 0000 0000  @........6......
00000030: 0000 0000 4000 3800 0d00 4000 1f00 1e00  ....@.8...@.....
00000040: 0600 0000 0400 0000 4000 0000 0000 0000  ........@.......
00000050: 4000 0000 0000 0000 4000 0000 0000 0000  @.......@.......
00000060: d802 0000 0000 0000 d802 0000 0000 0000  ................
00000070: 0800 0000 0000 0000 0300 000
00000080: 1803 0000 0000 0000 1803 000
00000090: 1803 0000 0000 0000 1c00 000
000000a0: 1c00 0000 0000 0000 0100 000
000000b0: 0100 0000 0400 0000 0000 000
000000c0: 0000 0000 0000 0000 0000 000
000000d0: 2806 0000 0000 0000 2806 000
000000e0: 0010 0000 0000 0000 0100 000
000000f0: 0010 0000 0000 0000 0010 000
00000100: 0010 0000 0000 0000 8101 000
```

```
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:                              DYN (Position-Independent Executable file)
  Machine:                           Advanced Micro Devices X86-64
  Version:                           0x1
  Entry point address:               0x1060
  Start of program headers:          64 (bytes into file)
  Start of section headers:          13984 (bytes into file)
  Flags:                             0x0
  Size of this header:               64 (bytes)
  Size of program headers:           56 (bytes)
  Number of program headers:         13
  Size of section headers:           64 (bytes)
  Number of section headers:         31
  Section header string table index: 30
```

ELF Header (Ehdr)

```
00000000: 7f45 4c46 0201 0100 0000 0000 0000 0000  .ELF............
00000010: 0300 3e00 0100 0000 6010 0000 0000 0000  ..>.....`.......
00000020: 4000 0000 0000 0000 a036 0000 0000 0000  @........6......
00000030: 0000 0000 40
00000040: 0600 0000 04
00000050: 4000 0000 00
00000060: d802 0000 00
00000070: 0800 0000 00
00000080: 1803 0000 00
00000090: 1803 0000 00
000000a0: 1c00 0000 00
000000b0: 0100 0000 04
000000c0: 0000 0000 00
000000d0: 2806 0000 00
000000e0: 0010 0000 00
000000f0: 0010 0000 00
00000100: 0010 0000 00
```


```
00 00 00

t, little endian

V

-Independent Executable file)
o Devices X86-64

o file)
into file)

Size of this header:                64 (bytes)
Size of program headers:            56 (bytes)
Number of program headers:          13
Size of section headers:            64 (bytes)
Number of section headers:          31
Section header string table index: 30
```

ELF Header (Ehdr)

```
00000000: 7f45 4c46 0201 0100 0000 0000 0000 0000  .ELF............
00000010: 0300 3e00 0100 0000 6010 0000 0000 0000  ..>.....`.......
00000020: 4000 0000 0000 0000 a036 0000 0000 0000  @........6......
00000030: 0000 0000 0000 0000 0d00 4000 1f00 1e00  ....@.8...@.....
00000040: 0600 0000 0000 0000 4000 0000 0000 0000  ........@.......
00000050: 4000 0000 0000 0000 4000 0000 0000 0000  @.......@.......
00000060: d802 0000 0000 0000 d802 0000 0000 0000  ................
00000070: 0800 0000 0000 0000 0300 0000 0000 0000  ................
00000080: 1803 0000 0000 0000 1803 0000 0000 0000  ................
00000090: 1803 0000 0000 0000 1c00 0000 0000 0000  ................
000000a0: 1c00 0000 0000 0000 0100 0000 0000 0000  ................
000000b0: 0100 0000 0400 0000 0000 0000 0000 0000  ................
000000c0: 0000 0000 0000 0000 0000 0000 0000 0000  ................
000000d0: 2806 0000 0000 0000 2806 0000 0000 0000  (.......(.......
000000e0: 0010 0000 0000 0000 0100 0000 0000 0000  ................
000000f0: 0010 0000 0000 0000 0010 0000 0000 0000  ................
00000100: 0010 0000 0000 0000 8101 0000 0000 0000  ................
```

E_IDENT

This array of bytes specifies how to interpret the file

```
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:                              DYN (Position-Independent Executable file)
  Machine:                           Advanced Micro Devices X86-64
  Version:                           0x1
  Entry point address:               0x1060
  Start of program headers:          64 (bytes into file)
  Start of section headers:          13984 (bytes into file)
  Flags:                             0x0
  Size of this header:               64 (bytes)
  Size of program headers:           56 (bytes)
  Number of program headers:         13
  Size of section headers:           64 (bytes)
  Number of section headers:         31
  Section header string table index: 30
```

ELF Header (Ehdr)

```
00000000: 7f45 4c46 0201 0100 0000 0000 0000 0000  .ELF............
00000010: 0300 3e00 0100 0000 6010 0000 0000 0000  ..>.....`.......
00000020: 4000 0000 0000 0000 a036 0000 0000 0000  @........6......
00000030: 0000 0(                                  @.
00000040: 0600 0(                                  ..
00000050: 4000 0000 0000 0000 4000 0000 0000 0000  @.......@.......
00000060: d802 0000 0000 0000 d802 0000 0000 0000  ................
00000070: 0800 0000 0000 0000 0300 000
00000080: 1803 0000 0000 0000 1803 000
00000090: 1803 0000 0000 0000 1c00 000
000000a0: 1c00 0000 0000 0000 0100 000
000000b0: 0100 0000 0400 0000 0000 000
000000c0: 0000 0000 0000 0000 0000 000
000000d0: 2806 0000 0000 0000 2806 000
000000e0: 0010 0000 0000 0000 0100 000
000000f0: 0010 0000 0000 0000 0010 000
00000100: 0010 0000 0000 0000 8101 000
```

EI_MAG0 EI_MAG1 EI_MAG2 EI_MAG3

```
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:                              DYN (Position-Independent Executable file)
  Machine:                           Advanced Micro Devices X86-64
  Version:                           0x1
  Entry point address:               0x1060
  Start of program headers:          64 (bytes into file)
  Start of section headers:          13984 (bytes into file)
  Flags:                             0x0
  Size of this header:               64 (bytes)
  Size of program headers:           56 (bytes)
  Number of program headers:         13
  Size of section headers:           64 (bytes)
  Number of section headers:         31
  Section header string table index: 30
```

ELF Header (Ehdr)

```
00000000: 7f45 4c46 0201 0100 0000 0000 0000 0000  .ELF............
00000010: 0300 3e00 0100 0000 6010 0000 0000 0000  ..>.....`.......
00000020: 4000 0000 0000 0000 a036 0000 0000 0000  @........6......
00000030: 0000 0000 4000 0000 0000 1f00 1e00       ....@.8...@.....
00000040: 0600 0000 0400
00000050: 4000 0000 0000
00000060: d802 0000 0000
00000070: 0800 0000 0000 0000 0300 000
00000080: 1803 0000 0000 0000 1803 000
00000090: 1803 0000 0000 0000 1c00 000
000000a0: 1c00 0000 0000 0000 0100 000
000000b0: 0100 0000 0400 0000 0000 000
000000c0: 0000 0000 0000 0000 0000 000
000000d0: 2806 0000 0000 0000 2806 000
000000e0: 0010 0000 0000 0000 0100 000
000000f0: 0010 0000 0000 0000 0010 000
00000100: 0010 0000 0000 0000 8101 000
```

EI_CLASS

For 32-bit or 64-bit or invalid

```
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:                              DYN (Position-Independent Executable file)
  Machine:                           Advanced Micro Devices X86-64
  Version:                           0x1
  Entry point address:               0x1060
  Start of program headers:          64 (bytes into file)
  Start of section headers:          13984 (bytes into file)
  Flags:                             0x0
  Size of this header:               64 (bytes)
  Size of program headers:           56 (bytes)
  Number of program headers:         13
  Size of section headers:           64 (bytes)
  Number of section headers:         31
  Section header string table index: 30
```

ELF Header (Ehdr)

```
00000000: 7f45 4c46 0201 0100 0000 0000 0000 0000  .ELF............
00000010: 0300 3e00 0100 0000 6010 0000 0000 0000  ..>.....`.......
00000020: 4000 0000 0000 0000 a036 0000 0000 0000  @........6......
00000030: 0000 0000 3800 0d00 4000 1f00 1e00       ....@.8...@.....
00000040: 06
00000050: 40
00000060: d802
00000070: 0800 0000 0000 0000 0300 000
00000080: 1803 0000 0000 0000 1803 000
00000090: 1803 0000 0000 0000 1c00 000
000000a0: 1c00 0000 0000 0000 0100 000
000000b0: 0100 0000 0400 0000 0000 000
000000c0: 0000 0000 0000 0000 0000 000
000000d0: 2806 0000 0000 0000 2806 000
000000e0: 0010 0000 0000 0000 0100 000
000000f0: 0010 0000 0000 0000 0010 000
00000100: 0010 0000 0000 0000 8101 000
```

EI_DATA

ELFDATALSB (little-endian), ELFDATAMSB(big-endian)

```
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:                              DYN (Position-Independent Executable file)
  Machine:                           Advanced Micro Devices X86-64
  Version:                           0x1
  Entry point address:               0x1060
  Start of program headers:          64 (bytes into file)
  Start of section headers:          13984 (bytes into file)
  Flags:                             0x0
  Size of this header:               64 (bytes)
  Size of program headers:           56 (bytes)
  Number of program headers:         13
  Size of section headers:           64 (bytes)
  Number of section headers:         31
  Section header string table index: 30
```

ELF Header (Ehdr)

```
00000000: 7f45 4c46 0201 0100 0000 0000 0000 0000  .ELF............
00000010: 0300 3e00 0100 0000 6010 0000 0000 0000  ..>.....`.......
00000020: 4000 0000 0000 0000 a036 0000 0000 0000  @........6......
00000030: 0000 0000 4000 3800 1f00 1e00 0600 0000  ....@.8...@.....
00000040: 0600 0000 0000 0000 0000 0000 0000 0000  ................
00000050: 4000 0000 0000 0000 4000 0000 0000 0000  @.......@.......
00000060: d802 0000 0000 0000 d802 0000 0000 0000  ................
00000070: 0800 0000 0000 0000 0300 0000 0400 0000  ................
00000080: 1803 0000 0000 0000 1803 0000 0000 0000  ................
00000090: 1803 0000 0000 0000 1c00 0000 0000 0000  ................
000000a0: 1c00 0000 0000 0000 0100 0000 0500 0000  ................
000000b0: 0100 0000 0400 0000 0000 0000 0000 0000  ................
000000c0: 0000 0000 0000 0000 0000 0000 0000 0000  ................
000000d0: 2806 0000 0000 0000 2806 0000 0000 0000  (.......(.......
000000e0: 0010 0000 0000 0000 0100 0000 0600 0000  ................
000000f0: 0010 0000 0000 0000 0010 0000 0000 0000  ................
00000100: 0010 0000 0000 0000 8101 0000 0600 0000  ................
```

EI_VERSION

VERSION OF ELF SPECIFICATION

```
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:                              DYN (Position-Independent Executable file)
  Machine:                           Advanced Micro Devices X86-64
  Version:                           0x1
  Entry point address:               0x1060
  Start of program headers:          64 (bytes into file)
  Start of section headers:          13984 (bytes into file)
  Flags:                             0x0
  Size of this header:               64 (bytes)
  Size of program headers:           56 (bytes)
  Number of program headers:         13
  Size of section headers:           64 (bytes)
  Number of section headers:         31
  Section header string table index: 30
```

ELF Header (Ehdr)

```
00000000: 7f45 4c46 0201 0100 0000 0000 0000 0000  .ELF............
00000010: 0300 3e00 0100 0000 6010 0000 0000 0000  ..>.....`.......
00000020: 4000 0000 0000 0000 a036 0000 0000 0000  @........6......
00000030: 0000 0000 4[illegible]4000 1f00 1e00  ....@.8...@.....
00000040: 0600 0000 0[illegible]0000 0000 0000  @...............
00000050: 4000 0000 0[illegible]0000 0000 0000  @...............
00000060: d802 0000 0[illegible]d802 0000 0000  ................
00000070: 0800 0000 0000 0000 0300 000[illegible]
00000080: 1803 0000 0000 0000 1803 000[illegible]
00000090: 1803 0000 0000 0000 1c00 000[illegible]
000000a0: 1c00 0000 0000 0000 0100 000[illegible]
000000b0: 0100 0000 0400 0000 0000 000[illegible]
000000c0: 0000 0000 0000 0000 0000 000[illegible]
000000d0: 2806 0000 0000 0000 2806 000[illegible]
000000e0: 0010 0000 0000 0000 0100 000[illegible]
000000f0: 0010 0000 0000 0000 0010 000[illegible]
00000100: 0010 0000 0000 0000 8101 000[illegible]
```

EI_OSABI

OS AND ABI which object is targeted

```
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:                              DYN (Position-Independent Executable file)
  Machine:                           Advanced Micro Devices X86-64
  Version:                           0x1
  Entry point address:               0x1060
  Start of program headers:          64 (bytes into file)
  Start of section headers:          13984 (bytes into file)
  Flags:                             0x0
  Size of this header:               64 (bytes)
  Size of program headers:           56 (bytes)
  Number of program headers:         13
  Size of section headers:           64 (bytes)
  Number of section headers:         31
  Section header string table index: 30
```

ELF Header (Ehdr)

```
00000000: 7f45 4c46 0201 0100 0000 0000 0000 0000  .ELF............
00000010: 0300 3e00 0100 0000 6010 0000 0000 0000  ..>.....`.......
00000020: 4000 0000 0000 0000 a036 0000 0000 0000  @........6......
00000030: 0000 0000 4[illegible]                   ....@.8...@.....
00000040: 0600 0000 0[illegible]                   ................
00000050: 4000 0000 0[illegible]                   ................
00000060: d802 0000 0[illegible]                   ................
00000070: 0800 0000 0000 0000 0300 000[illegible]  ................
00000080: 1803 0000 0000 0000 1803 000[illegible]  ................
00000090: 1803 0000 0000 0000 1c00 000[illegible]  ................
000000a0: 1c00 0000 0000 0000 0100 000[illegible]  ................
000000b0: 0100 0000 0400 0000 0000 000[illegible]  ................
000000c0: 0000 0000 0000 0000 0000 000[illegible]  ................
000000d0: 2806 0000 0000 0000 2806 000[illegible]  ....(.......(...
000000e0: 0010 0000 0000 0000 0100 000[illegible]  ................
000000f0: 0010 0000 0000 0000 0010 000[illegible]  ................
00000100: 0010 0000 0000 0000 8101 000[illegible]  ................
```

EI_ABIVERSION

ABI Version, 0 for most of the time

```
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:                              DYN (Position-Independent Executable file)
  Machine:                           Advanced Micro Devices X86-64
  Version:                           0x1
  Entry point address:               0x1060
  Start of program headers:          64 (bytes into file)
  Start of section headers:          13984 (bytes into file)
  Flags:                             0x0
  Size of this header:               64 (bytes)
  Size of program headers:           56 (bytes)
  Number of program headers:         13
  Size of section headers:           64 (bytes)
  Number of section headers:         31
  Section header string table index: 30
```

ELF Header (Ehdr)

```
00000000: 7f45 4c46 0201 0100 0000 0000 0000 0000  .ELF............
00000010: 0300 3e00 0100 0000 6010 0000 0000 0000  ..>.....`.......
00000020: 4000 0000 0000 0000 a036 0000 0000 0000  @........6......
00000030: 0000 0000 4[illegible]00 4000 1f00 1e00  ....@.8...@.....
00000040: 0600 0000 0[illegible]                   .....@.......
00000050: 4000 0000 0[illegible]                   .....@.......
00000060: d802 0000 0[illegible]                   ................
00000070: 0800 0000 0000 0000 0300 000[illegible]
00000080: 1803 0000 0000 0000 1803 000[illegible]
00000090: 1803 0000 0000 0000 1c00 000[illegible]
000000a0: 1c00 0000 0000 0000 0100 000[illegible]
000000b0: 0100 0000 0400 0000 0000 000[illegible]
000000c0: 0000 0000 0000 0000 0000 000[illegible]
000000d0: 2806 0000 0000 0000 2806 000[illegible]
000000e0: 0010 0000 0000 0000 0100 000[illegible]
000000f0: 0010 0000 0000 0000 0010 000[illegible]
00000100: 0010 0000 0000 0000 8101 000[illegible]
```

EI_PAD

For padding, leave 0's

```
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:                              DYN (Position-Independent Executable file)
  Machine:                           Advanced Micro Devices X86-64
  Version:                           0x1
  Entry point address:               0x1060
  Start of program headers:          64 (bytes into file)
  Start of section headers:          13984 (bytes into file)
  Flags:                             0x0
  Size of this header:               64 (bytes)
  Size of program headers:           56 (bytes)
  Number of program headers:         13
  Size of section headers:           64 (bytes)
  Number of section headers:         31
  Section header string table index: 30
```

ELF Header (Ehdr)

```
00000000: 7f45 4c46 0201 0100 0000 0000 0000 0000  .ELF............
00000010: 0300 3e00 0100 0000 6010 0000 0000 0000  ..>.....`.......
00000020: 4000 0000 0000 0000 a036 0000 0000 0000  @........6......
00000030: 0000 0000 4[illegible]00 4000 1f00 1e00  ....@.8...@.....
00000040: 0600 0000 0[illegible]                   ....@...........
00000050: 4000 0000 0[illegible]                   @...............
00000060: d802 0000 0[illegible]                   ................
00000070: 0800 0000 0000 0000 0300 000[illegible]  ................
00000080: 1803 0000 0000 0000 1803 000[illegible]  ................
00000090: 1803 0000 0000 0000 1c00 000[illegible]  ................
000000a0: 1c00 0000 0000 0000 0100 000[illegible]  ................
000000b0: 0100 0000 0400 0000 0000 000[illegible]  ................
000000c0: 0000 0000 0000 0000 0000 000[illegible]  ................
000000d0: 2806 0000 0000 0000 2806 000[illegible]  (.......(.....
000000e0: 0010 0000 0000 0000 0100 000[illegible]  ................
000000f0: 0010 0000 0000 0000 0010 000[illegible]  ................
00000100: 0010 0000 0000 0000 8101 000[illegible]  ................
```

E_TYPE

ET_NONE, ET_REL, ET_DYN, ET_CORE

```
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:                              DYN (Position-Independent Executable file)
  Machine:                           Advanced Micro Devices X86-64
  Version:                           0x1
  Entry point address:               0x1060
  Start of program headers:          64 (bytes into file)
  Start of section headers:          13984 (bytes into file)
  Flags:                             0x0
  Size of this header:               64 (bytes)
  Size of program headers:           56 (bytes)
  Number of program headers:         13
  Size of section headers:           64 (bytes)
  Number of section headers:         31
  Section header string table index: 30
```

ELF Header (Ehdr)

```
00000000: 7f45 4c46 0201 0100 0000 0000 0000 0000  .ELF............
00000010: 0300 3e00 0100 0000 6010 0000 0000 0000  ..>.....`.......
00000020: 4000 0000 0000 0000 a036 0000 0000 0000  @........6......
00000030: 0000 0000 4000 3800 0d00 4000 1f00 1e00  ....@.8...@.....
00000040: 0600 0000 0400 0000 4000 0000 0000 0000  ........@.......
00000050: 4000 0000 0000 0000 4000 0000 0000 0000  @.......@.......
00000060: d802 0000 0000 0000 d802 0000 0000 0000  ................
00000070: 0800 0000 0000 0000 0300 000
00000080: 1803 0000 0000 0000 1803 000
00000090: 1803 0000 0000 0000 1c00 000
000000a0: 1c00 0000 0000 0000 0100 000
000000b0: 0100 0000 0400 0000 0000 000
000000c0: 0000 0000 0000 0000 0000 000
000000d0: 2806 0000 0000 0000 2806 000
000000e0: 0010 0000 0000 0000 0100 000
000000f0: 0010 0000 0000 0000 0010 000
00000100: 0010 0000 0000 0000 8101 000
```

E_ENTRY

Entrypoint of the program

```
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:                              DYN (Position-Independent Executable file)
  Machine:                           Advanced Micro Devices X86-64
  Version:                           0x1
  Entry point address:               0x1060
  Start of program headers:          64 (bytes into file)
  Start of section headers:          13984 (bytes into file)
  Flags:                             0x0
  Size of this header:               64 (bytes)
  Size of program headers:           56 (bytes)
  Number of program headers:         13
  Size of section headers:           64 (bytes)
  Number of section headers:         31
  Section header string table index: 30
```

ELF Header (Ehdr)

```
00000000: 7f45 4c46 0201 0100 0000 0000 0000 0000  .ELF............
00000010: 0300 3e00 0100 0000 6010 0000 0000 0000  ..>.....`.......
00000020: 4000 0000 0000 0000 a036 0000 0000 0000  @........6......
00000030: 0000 0000 4000 3800 0d00 4000 1f00 1e00  ....@.8...@.....
00000040: 0600 0000 0400 0000 4000 0000 0000 0000  ........@.......
00000050: 0000 0000 4000 0000 0000 0000 0000 0000  @.......@.......
00000060: 0000 0000 d802 0000 0000 0000 0000 0000  ................
00000070:
00000080:
00000090: 1803 0000 0000 0000 1c00 000
000000a0: 1c00 0000 0000 0000 0100 000
000000b0: 0100 0000 0400 0000 0000 000
000000c0: 0000 0000 0000 0000 0000 000
000000d0: 2806 0000 0000 0000 2806 000
000000e0: 0010 0000 0000 0000 0100 000
000000f0: 0010 0000 0000 0000 0010 000
00000100: 0010 0000 0000 0000 8101 000
```

E_PHOFF

Offset of Program Header

```
45 4c 46 02 01 01 00 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:                              DYN (Position-Independent Executable file)
Machine:                           Advanced Micro Devices X86-64
Version:                           0x1
Entry point address:               0x1060
Start of program headers:          64 (bytes into file)
Start of section headers:          13984 (bytes into file)
Flags:                             0x0
Size of this header:               64 (bytes)
Size of program headers:           56 (bytes)
Number of program headers:         13
Size of section headers:           64 (bytes)
Number of section headers:         31
Section header string table index: 30
```

ELF Header (Ehdr)

```
00000000: 7f45 4c46 0201 0100 0000 0000 0000 0000  .ELF............
00000010: 0300 3e00 0100 0000 6010 0000 0000 0000  ..>.....`.......
00000020: 4000 0000 0000 0000 a036 0000 0000 0000  @........6......
00000030: 0000 0000 4000 3800 0d00 4000 1f00 1e00  ....@.8...@.....
00000040: 0600 0000 0400 0000 4000 0000 0000 0000  ........@.......
00000050: 4000 0000 0000 0000 4000 0000 0000 0000  @.......@.......
00000060: 0802 0000 0000 0000 0802 0000 0000 0000  ................
00000070:
00000080:
00000090: 1803 0000 0000 0000 1c00 000
000000a0: 1c00 0000 0000 0000 0100 000
000000b0: 0100 0000 0400 0000 0000 000
000000c0: 0000 0000 0000 0000 0000 000
000000d0: 2806 0000 0000 0000 2806 000
000000e0: 0010 0000 0000 0000 0100 000
000000f0: 0010 0000 0000 0000 0010 000
00000100: 0010 0000 0000 0000 8101 000
```

E_PHENTSIZE

Size of Program Header

```
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:                              DYN (Position-Independent Executable file)
Machine:                           Advanced Micro Devices X86-64
Version:                           0x1
Entry point address:               0x1060
Start of program headers:          64 (bytes into file)
Start of section headers:          13984 (bytes into file)
Flags:                             0x0
Size of this header:               64 (bytes)
Size of program headers:           56 (bytes)
Number of program headers:         13
Size of section headers:           64 (bytes)
Number of section headers:         31
Section header string table index: 30
```

ELF Header (Ehdr)

```
00000000: 7f45 4c46 0201 0100 0000 0000 0000 0000  .ELF............
00000010: 0300 3e00 0100 0000 6010 0000 0000 0000  ..>.....`.......
00000020: 4000 0000 0000 0000 a036 0000 0000 0000  @........6......
00000030: 0000 0000 4000 3800 0d00 4000 1f00 1e00  ....@.8...@.....
00000040: 0600 0000 0400 0000 4000 0000 0000 0000  ........@.......
00000050: 0000 0000 0000 0000 4000 0000 0000 0000  @.......@.......
00000060: 0000 0000 0000 0000 d802 0000 0000 0000  ................
00000070: 0000 0000 0000 0000 0000 0000 0000 0000  ................
00000080: 0000 0000 0000 0000 0000 0000 0000 0000  ................
00000090: 1803 0000 0000 0000 1c00 0000 0000 0000  ................
000000a0: 1c00 0000 0000 0000 0100 0000 0000 0000  ................
000000b0: 0100 0000 0400 0000 0000 0000 0000 0000  ................
000000c0: 0000 0000 0000 0000 0000 0000 0000 0000  ................
000000d0: 2806 0000 0000 0000 2806 0000 0000 0000  (.......(.......
000000e0: 0010 0000 0000 0000 0100 0000 0000 0000  ................
000000f0: 0010 0000 0000 0000 0010 0000 0000 0000  ................
00000100: 0010 0000 0000 0000 8101 0000 0000 0000  ................
```

E_PHNUM

Number of Program Header Entries

```
0 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:                              DYN (Position-Independent Executable file)
Machine:                           Advanced Micro Devices X86-64
Version:                           0x1
Entry point address:               0x1060
Start of program headers:          64 (bytes into file)
Start of section headers:          13984 (bytes into file)
Flags:                             0x0
Size of this header:               64 (bytes)
Size of program headers:           56 (bytes)
Number of program headers:         13
Size of section headers:           64 (bytes)
Number of section headers:         31
Section header string table index: 30
```

ELF Header (Ehdr)

```
00000000: 7f45 4c46 0201 0100 0000 0000 0000 0000  .ELF............
00000010: 0300 3e00 0100 0000 6010 0000 0000 0000  ..>.....`.......
00000020: 4000 0000 0000 0000 a036 0000 0000 0000  @........6......
00000030: 0000 0000 4000 3800 0d00 4000 1f00 1e00  ....@.8...@.....
00000040: 0600 0000 0400 0000 4000 0000 0000 0000  ........@.......
00000050: 0000 0000 0000 0000 4000 0000 0000 0000  @.......@.......
00000060: 0000 0000 0000 0000 d802 0000 0000 0000  ................
00000070:
00000080:
00000090: 1803 0000 0000 0000 1c00 000
000000a0: 1c00 0000 0000 0000 0100 000
000000b0: 0100 0000 0400 0000 0000 000
000000c0: 0000 0000 0000 0000 0000 000
000000d0: 2806 0000 0000 0000 2806 000
000000e0: 0010 0000 0000 0000 0100 000
000000f0: 0010 0000 0000 0000 0010 000
00000100: 0010 0000 0000 0000 8101 000
```

E_SHOFF

Offset of Section Header

```
f 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:                              DYN (Position-Independent Executable file)
Machine:                           Advanced Micro Devices X86-64
Version:                           0x1
Entry point address:               0x1060
Start of program headers:          64 (bytes into file)
Start of section headers:          13984 (bytes into file)
Flags:                             0x0
Size of this header:               64 (bytes)
Size of program headers:           56 (bytes)
Number of program headers:         13
Size of section headers:           64 (bytes)
Number of section headers:         31
Section header string table index: 30
```

ELF Header (Ehdr)

```
00000000: 7f45 4c46 0201 0100 0000 0000 0000 0000  .ELF............
00000010: 0300 3e00 0100 0000 6010 0000 0000 0000  ..>.....`.......
00000020: 4000 0000 0000 0000 a036 0000 0000 0000  @........6......
00000030: 0000 0000 4000 3800 0d00 4000 1f00 1e00  ....@.8...@.....
00000040: 0600 0000 0400 0000 4000 0000 0000 0000  ........@.......
00000050: 0000 0000 0000 0000 0000 0000 0000 0000  @.......@.......
00000060: 0000 0000 0000 0000 f802 0000 0000 0000  ................
00000070: 0000 0000 0000 0000 0000 0000 0000 0000
00000080: 0000 0000 0000 0000 0000 0000 0000 0000
00000090: 1803 0000 0000 0000 1c00 0000 0000 0000
000000a0: 1c00 0000 0000 0000 0100 0000 0000 0000
000000b0: 0100 0000 0400 0000 0000 0000 0000 0000
000000c0: 0000 0000 0000 0000 0000 0000 0000 0000
000000d0: 2806 0000 0000 0000 2806 0000 0000 0000
000000e0: 0010 0000 0000 0000 0100 0000 0000 0000
000000f0: 0010 0000 0000 0000 0010 0000 0000 0000
00000100: 0010 0000 0000 0000 8101 0000 0000 0000
```

E_SHENTSIZE

Size of Section Header

```
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:                              DYN (Position-Independent Executable file)
  Machine:                           Advanced Micro Devices X86-64
  Version:                           0x1
  Entry point address:               0x1060
  Start of program headers:          64 (bytes into file)
  Start of section headers:          13984 (bytes into file)
  Flags:                             0x0
  Size of this header:               64 (bytes)
  Size of program headers:           56 (bytes)
  Number of program headers:         13
  Size of section headers:           64 (bytes)
  Number of section headers:         31
  Section header string table index: 30
```

ELF Header (Ehdr)

```
00000000: 7f45 4c46 0201 0100 0000 0000 0000 0000  .ELF............
00000010: 0300 3e00 0100 0000 6010 0000 0000 0000  ..>.....`.......
00000020: 4000 0000 0000 0000 a036 0000 0000 0000  @........6......
00000030: 0000 0000 4000 3800 0d00 4000 1f00 1e00  ....@.8...@.....
00000040: 0600 0000 0400 0000 4000 0000 0000 0000  ........@.......
00000050: 0000 0000 0000 0000 4000 0000 0000 0000  @.......@.......
00000060: 0000 0000 0000 0000 d802 0000 0000 0000  ................
00000070:                                          ........
00000080:                                          00 00 00 00 00 00 00 00
00000090: 1803 0000 0000 0000 1c00 000
000000a0: 1c00 0000 0000 0000 0100 000
000000b0: 0100 0000 0400 0000 0000 000
000000c0: 0000 0000 0000 0000 0000 000
000000d0: 2806 0000 0000 0000 2806 000
000000e0: 0010 0000 0000 0000 0100 000
000000f0: 0010 0000 0000 0000 0010 000
00000100: 0010 0000 0000 0000 8101 000
```

E_SHNUM

Number of Section Header Entries

```
Class:                             ELF64
Data:                              2's complement, little endian
Version:                           1 (current)
OS/ABI:                            UNIX - System V
ABI Version:                       0
Type:                              DYN (Position-Independent Executable file)
Machine:                           Advanced Micro Devices X86-64
Version:                           0x1
Entry point address:               0x1060
Start of program headers:          64 (bytes into file)
Start of section headers:          13984 (bytes into file)
Flags:                             0x0
Size of this header:               64 (bytes)
Size of program headers:           56 (bytes)
Number of program headers:         13
Size of section headers:           64 (bytes)
Number of section headers:         31
Section header string table index: 30
```

ELF Header (Ehdr)

```
00000000: 7f45 4c46 0201 0100 0000 0000 0000 0000  .ELF............
00000010: 0300 3e00 0100 0000 6010 0000 0000 0000  ..>.....`.......
00000020: 4000 0000 0000 0000 a036 0000 0000 0000  @........6......
00000030: 0000 0000 4000 3800 0d00 4000 1f00 1e00  ....@.8...@.....
00000040: 0600 0000 0400 0000 4000 0000 0000 0000  ........@.......
00000050: 0000 0000 0000 0000 4000 0000 0000 0000  @.......@.......
00000060:                     d802 0000 0000 0000
00000070:                                         00 00 00 00 00 00
00000080:
00000090: 1803 0000 0000 0000 1c00 000
000000a0: 1c00 0000 0000 0000 0100 000
000000b0: 0100 0000 0400 0000 0000 000
000000c0: 0000 0000 0000 0000 0000 000
000000d0: 2806 0000 0000 0000 2806 000
000000e0: 0010 0000 0000 0000 0100 000
000000f0: 0010 0000 0000 0000 0010 000
00000100: 0010 0000 0000 0000 8101 000
```

E_SHSTRNDX

Index of String Table at Section Header

```
Class:                             ELF64
Data:                              2's complement, little endian
Version:                           1 (current)
OS/ABI:                            UNIX - System V
ABI Version:                       0
Type:                              DYN (Position-Independent Executable file)
Machine:                           Advanced Micro Devices X86-64
Version:                           0x1
Entry point address:               0x1060
Start of program headers:          64 (bytes into file)
Start of section headers:          13984 (bytes into file)
Flags:                             0x0
Size of this header:               64 (bytes)
Size of program headers:           56 (bytes)
Number of program headers:         13
Size of section headers:           64 (bytes)
Number of section headers:         31
Section header string table index: 30
```

Program Header (Phdr)

Each Program Header is 0x38.
Program Header tells OS how to load the memory

```
00000000: 7f
00000010: 03
00000020: 40
00000030: 00
00000040: 0600 0000 0400 0000 4000 0000 0000 0000  ........@.......
00000050: 4000 0000 0000 0000 4000 0000 0000 0000  @.......@.......
00000060: d802 0000 0000 0000 d802 0000 0000 0000  ................
00000070: 0800 0000 0000 0000 0300 0000 0400 0000  ................
00000080: 1803 0000 0000 0000 1803 0000 0000 0000  ................
00000090: 1803 0000 0000 0000 1c00 0000 0000 0000  ................
000000a0: 1c00 0000 0000 0000 0100 0000 0000 0000  ................
000000b0: 0100 0000 0400 0000 0000 00
000000c0: 0000 0000 0000 0000 0000 00
000000d0: 2806 0000 0000 0000 2806 00
000000e0: 0010 0000 0000 0000 0100 00
000000f0: 0010 0000 0000 0000 0010 00
00000100: 0010 0000 0000 0000 8101 00
```

```
Program Headers:
  Type           Offset             VirtAddr           PhysAddr
                 FileSiz            MemSiz              Flags  Align
  PHDR           0x0000000000000040 0x0000000000000040 0x0000000000000040
                 0x00000000000002d8 0x00000000000002d8  R      0x8
  INTERP         0x0000000000000318 0x0000000000000318 0x0000000000000318
                 0x000000000000001c 0x000000000000001c  R      0x1
      [Requesting program interpreter: /lib64/ld-linux-x86-64.so.2]
  LOAD           0x0000000000000000 0x0000000000000000 0x0000000000000000
                 0x0000000000000628 0x0000000000000628  R      0x1000
  LOAD           0x0000000000001000 0x0000000000001000 0x0000000000001000
                 0x0000000000000181 0x0000000000000181  R E    0x1000
  LOAD           0x0000000000002000 0x0000000000002000 0x0000000000002000
```

Program Header (Phdr)

Each Program Header is 0x38.
Program Header tells OS how to load the memory

```
00000000: 7f
00000010: 03
00000020: 40
00000030: 00
00000040: 0600 0000 0400 0000 4000 0000 0000 0000  ........@.......
00000050: 4000 0000 0000 0000 4000 0000 0000 0000  @.......@.......
00000060: d802 0000 0000 0000 d802 0000 0000 0000  ................
00000070: 0800 0000 0000 0000 0300 0000 0400 0000  ................
00000080: 1803 0000 0000 0000 1803 0000 0000 0000  ................
00000090: 1803 0000 0000 0000 1c00 0000 0000 0000  ................
000000a0: 1c00 0000 0000 0000 0100 0000 0000 0000  ................
000000b0: 0100 0000 0400 0000 0000 00
000000c0: 0000 0000 0000 0000 0000 00
000000d0: 2806 0000 0000 0000 2806 00
000000e0: 0010 0000 0000 0000 0100 00
000000f0: 0010 0000 0000 0000 0010 00
00000100: 0010 0000 0000 0000 8101 00
```

```
Program Headers:
  Type           Offset             VirtAddr           PhysAddr
                 FileSiz            MemSiz              Flags  Align
  PHDR           0x0000000000000040 0x0000000000000040 0x0000000000000040
                 0x00000000000002d8 0x00000000000002d8  R      0x8
  INTERP         0x0000000000000318 0x0000000000000318 0x0000000000000318
                 0x000000000000001c 0x000000000000001c  R      0x1
      [Requesting program interpreter: /lib64/ld-linux-x86-64.so.2]
  LOAD           0x0000000000000000 0x0000000000000000 0x0000000000000000
                 0x0000000000000628 0x0000000000000628  R      0x1000
  LOAD           0x0000000000001000 0x0000000000001000 0x0000000000001000
                 0x0000000000000181 0x0000000000000181  R E    0x1000
  LOAD           0x0000000000002000 0x0000000000002000 0x0000000000002000
```

Program Header (Phdr)

p_type

Type of Program Header

```
00000000: 7f45 4c46 02                                    .ELF.....
00000010: 0300 3e00 01                                    ..>.....
00000020: 4000 0000 00                                    @.....6.
00000030: 0000 0000 40                                    ....@...
00000040: 0600 0000 0400 0000 4000 0000 0000 0000  ........@.......
00000050: 4000 0000 0000 0000 4000 0000 0000 0000  @.......@.......
00000060: d802 0000 0000 0000 d802 0000 0000 0000  ................
00000070: 0800 0000 0000 0000 0300 0000 0400 0000  ................
00000080: 1803 0000 0000 0000 1803 0000 0000 0000  ................
00000090: 1803 0000 0000 0000 1c00 0000 0000 0000  ................
000000a0: 1c00 0000 0000 0000 0100 0000 0000 0000  ................
000000b0: 0100 0000 0400 0000 0000 00              ................
000000c0: 0000 0000 0000 0000 0000 00              ................
000000d0: 2806 0000 0000 0000 2806 00              ....(.....(.....
000000e0: 0010 0000 0000 0000 0100 00              ................
000000f0: 0010 0000 0000 0000 0010 00              ................
00000100: 0010 0000 0000 0000 8101 00              ................
```

```
Program Headers:
  Type           Offset             VirtAddr           PhysAddr
                 FileSiz            MemSiz              Flags  Align
  PHDR           0x0000000000000040 0x0000000000000040 0x0000000000000040
                 0x00000000000002d8 0x00000000000002d8  R      0x8
  INTERP         0x0000000000000318 0x0000000000000318 0x0000000000000318
                 0x000000000000001c 0x000000000000001c  R      0x1
      [Requesting program interpreter: /lib64/ld-linux-x86-64.so.2]
  LOAD           0x0000000000000000 0x0000000000000000 0x0000000000000000
                 0x0000000000000628 0x0000000000000628  R      0x1000
  LOAD           0x0000000000001000 0x0000000000001000 0x0000000000001000
                 0x0000000000000181 0x0000000000000181  R E    0x1000
  LOAD           0x0000000000002000 0x0000000000002000 0x0000000000002000
```

Program Header (Phdr)

p_flag

R/W/X Permission

```
00000000: 7f45 4c46 02                    000 0000 0000 0000   .ELF............
00000010: 0300 3e00 01                    010 0000 0000 0000   ..>.....`.......
00000020: 4000 0000 00                                         @........6......
00000030: 0000 0000 40                                         ....@.8...@.....
00000040: 0600 0000 0400 0000 4000 0000 0000 0000   ........@.......
00000050: 4000 0000 0000 0000 4000 0000 0000 0000   @.......@.......
00000060: d802 0000 0000 0000 d802 0000 0000 0000   ................
00000070: 0800 0000 0000 0000 0300 0000 0400 0000   ................
00000080: 1803 0000 0000 0000 1803 0000 0000 0000   ................
00000090: 1803 0000 0000 0000 1c00 0000 0000 0000   ................
000000a0: 1c00 0000 0000 0000 0100 0000 0000 0000   ................
000000b0: 0100 0000 0400 0000 0000 00
000000c0: 0000 0000 0000 0000 0000 00
000000d0: 2806 0000 0000 0000 2806 00
000000e0: 0010 0000 0000 0000 0100 00
000000f0: 0010 0000 0000 0000 0010 00
00000100: 0010 0000 0000 0000 8101 00
```

```
Program Headers:
  Type           Offset             VirtAddr           PhysAddr
                 FileSiz            MemSiz              Flags  Align
  PHDR           0x0000000000000040 0x0000000000000040 0x0000000000000040
                 0x00000000000002d8 0x00000000000002d8  R      0x8
  INTERP         0x0000000000000318 0x0000000000000318 0x0000000000000318
                 0x000000000000001c 0x000000000000001c  R      0x1
      [Requesting program interpreter: /lib64/ld-linux-x86-64.so.2]
  LOAD           0x0000000000000000 0x0000000000000000 0x0000000000000000
                 0x0000000000000628 0x0000000000000628  R      0x1000
  LOAD           0x0000000000001000 0x0000000000001000 0x0000000000001000
                 0x0000000000000181 0x0000000000000181  R E    0x1000
  LOAD           0x0000000000002000 0x0000000000002000 0x0000000000002000
```

Program Header (Phdr)

p_offset

File Offset

```
00000000: 7f45 4c46 02                    0000 0000 0000   .ELF............
00000010: 0300 3e00 01                    0000 0000 0000   ..>.....`.......
00000020: 4000 0000 00                    0000 0000 0000   @........6......
00000030: 0000 0000 40                    0000 1f00 1e00   ....@.8...@.....
00000040: 0600 0000 0400 0000 4000 0000 0000 0000   ........@.......
00000050: 4000 0000 0000 0000 4000 0000 0000 0000   @.......@.......
00000060: d802 0000 0000 0000 d802 0000 0000 0000   ................
00000070: 0800 0000 0000 0000 0300 0000 0400 0000   ................
00000080: 1803 0000 0000 0000 1803 0000 0000 0000   ................
00000090: 1803 0000 0000 0000 1c00 0000 0000 0000   ................
000000a0: 1c00 0000 0000 0000 0100 0000 0000 0000   ................
000000b0: 0100 0000 0400 0000 0000 00
000000c0: 0000 0000 0000 0000 0000 00
000000d0: 2806 0000 0000 0000 2806 00
000000e0: 0010 0000 0000 0000 0100 00
000000f0: 0010 0000 0000 0000 0010 00
00000100: 0010 0000 0000 0000 8101 00
```

```
Program Headers:
  Type           Offset             VirtAddr           PhysAddr
                 FileSiz            MemSiz              Flags  Align
  PHDR           0x0000000000000040 0x0000000000000040 0x0000000000000040
                 0x00000000000002d8 0x00000000000002d8  R      0x8
  INTERP         0x0000000000000318 0x0000000000000318 0x0000000000000318
                 0x000000000000001c 0x000000000000001c  R      0x1
      [Requesting program interpreter: /lib64/ld-linux-x86-64.so.2]
  LOAD           0x0000000000000000 0x0000000000000000 0x0000000000000000
                 0x0000000000000628 0x0000000000000628  R      0x1000
  LOAD           0x0000000000001000 0x0000000000001000 0x0000000000001000
                 0x0000000000000181 0x0000000000000181  R E    0x1000
  LOAD           0x0000000000002000 0x0000000000002000 0x0000000000002000
```

Program Header (Phdr)

p_vaaddr

Virtual Address

```
00000000: 7f45 4c46 02                    0000 0000 0000   .ELF............
00000010: 0300 3e00 01                    0000 0000 0000   ..>.....`.......
00000020: 4000 0000 00                    0000 0000 0000   @........6......
00000030: 0000 0000 40                              1e00   ....@.8...@.....
00000040: 0600 0000 0400 0000 4000 0000 0000 0000   ........@.......
00000050: 4000 0000 0000 0000 4000 0000 0000 0000   @.......@.......
00000060: d802 0000 0000 0000 d802 0000 0000 0000   ................
00000070: 0800 0000 0000 0000 0300 0000 0400 0000   ................
00000080: 1803 0000 0000 0000 1803 0000 0000 0000   ................
00000090: 1803 0000 0000 0000 1c00 0000 0000 0000   ................
000000a0: 1c00 0000 0000 0000 0100 0000 0000 0000   ................
000000b0: 0100 0000 0400 0000 0000 00
000000c0: 0000 0000 0000 0000 0000 00
000000d0: 2806 0000 0000 0000 2806 00
000000e0: 0010 0000 0000 0000 0100 00
000000f0: 0010 0000 0000 0000 0010 00
00000100: 0010 0000 0000 0000 8101 00
```

```
Program Headers:
  Type           Offset             VirtAddr           PhysAddr
                 FileSiz            MemSiz              Flags  Align
  PHDR           0x0000000000000040 0x0000000000000040 0x0000000000000040
                 0x00000000000002d8 0x00000000000002d8  R      0x8
  INTERP         0x0000000000000318 0x0000000000000318 0x0000000000000318
                 0x000000000000001c 0x000000000000001c  R      0x1
      [Requesting program interpreter: /lib64/ld-linux-x86-64.so.2]
  LOAD           0x0000000000000000 0x0000000000000000 0x0000000000000000
                 0x0000000000000628 0x0000000000000628  R      0x1000
  LOAD           0x0000000000001000 0x0000000000001000 0x0000000000001000
                 0x0000000000000181 0x0000000000000181  R E    0x1000
  LOAD           0x0000000000002000 0x0000000000002000 0x0000000000002000
```

Program Header (Phdr)

p_paaddr

Physical Address, used when PA is relevant

```
00000000: 7f45 4c46 02                    0000 0000 0000   .ELF.....
00000010: 0300 3e00 01                    0000 0000 0000   ..>.....
00000020: 4000 0000 00                                     @.....
00000030: 0000 0000 40                                     ....@...
00000040: 0600 0000 0400 0000 4000 0000 0000 0000  ........@.......
00000050: 4000 0000 0000 0000 4000 0000 0000 0000  @.......@.......
00000060: d802 0000 0000 0000 d802 0000 0000 0000  ................
00000070: 0800 0000 0000 0000 0300 0000 0400 0000  ................
00000080: 1803 0000 0000 0000 1803 0000 0000 0000  ................
00000090: 1803 0000 0000 0000 1c00 0000 0000 0000  ................
000000a0: 1c00 0000 0000 0000 0100 0000 0000 0000  ................
000000b0: 0100 0000 0400 0000 0000 00              ................
000000c0: 0000 0000 0000 0000 0000 00              ................
000000d0: 2806 0000 0000 0000 2806 00              ....(.....(.....
000000e0: 0010 0000 0000 0000 0100 00              ................
000000f0: 0010 0000 0000 0000 0010 00              ................
00000100: 0010 0000 0000 0000 8101 00              ................
```

```
Program Headers:
  Type           Offset             VirtAddr           PhysAddr
                 FileSiz            MemSiz              Flags  Align
  PHDR           0x0000000000000040 0x0000000000000040 0x0000000000000040
                 0x00000000000002d8 0x00000000000002d8  R      0x8
  INTERP         0x0000000000000318 0x0000000000000318 0x0000000000000318
                 0x000000000000001c 0x000000000000001c  R      0x1
      [Requesting program interpreter: /lib64/ld-linux-x86-64.so.2]
  LOAD           0x0000000000000000 0x0000000000000000 0x0000000000000000
                 0x0000000000000628 0x0000000000000628  R      0x1000
  LOAD           0x0000000000001000 0x0000000000001000 0x0000000000001000
                 0x0000000000000181 0x0000000000000181  R E    0x1000
  LOAD           0x0000000000002000 0x0000000000002000 0x0000000000002000
```

Program Header (Phdr)

p_filesz

File Size, size of this segment in image

```
00000000: 7f45 4c46 02                    0 0000 0000 0000   .ELF.....
00000010: 0300 3e00 01                    0 0000 0000 0000   ..>.....
00000020: 4000 0000 00                                       @.....
00000030: 0000 0000 40                                       ....@...
00000040: 0600 0000 0400 0000 4000 0000 0000 0000  ........@.......
00000050: 4000 0000 0000 0000 4000 0000 0000 0000  @.......@.......
00000060: d802 0000 0000 0000 d802 0000 0000 0000  ................
00000070: 0800 0000 0000 0000 0300 0000 0400 0000  ................
00000080: 1803 0000 0000 0000 1803 0000 0000 0000  ................
00000090: 1803 0000 0000 0000 1c00 0000 0000 0000  ................
000000a0: 1c00 0000 0000 0000 0100 0000 0000 0000  ................
000000b0: 0100 0000 0400 0000 0000 00              ..........
000000c0: 0000 0000 0000 0000 0000 00              ..........
000000d0: 2806 0000 0000 0000 2806 00              (.......(..
000000e0: 0010 0000 0000 0000 0100 00              ..........
000000f0: 0010 0000 0000 0000 0010 00              ..........
00000100: 0010 0000 0000 0000 8101 00              ..........
```

```
Program Headers:
  Type           Offset             VirtAddr           PhysAddr
                 FileSiz            MemSiz              Flags  Align
  PHDR           0x0000000000000040 0x0000000000000040 0x0000000000000040
                 0x00000000000002d8 0x00000000000002d8  R      0x8
  INTERP         0x0000000000000318 0x0000000000000318 0x0000000000000318
                 0x000000000000001c 0x000000000000001c  R      0x1
      [Requesting program interpreter: /lib64/ld-linux-x86-64.so.2]
  LOAD           0x0000000000000000 0x0000000000000000 0x0000000000000000
                 0x0000000000000628 0x0000000000000628  R      0x1000
  LOAD           0x0000000000001000 0x0000000000001000 0x0000000000001000
                 0x0000000000000181 0x0000000000000181  R E    0x1000
  LOAD           0x0000000000002000 0x0000000000002000 0x0000000000002000
```

Program Header (Phdr)

p_memsz

Memory Image Size

```
00000000: 7f45 4c46 02                                    .ELF.....
00000010: 0300 3e00 01                                    ..>.....
00000020: 4000 0000 00                                    @.....
00000030: 0000 0000 40                                    ....@.8...@.....
00000040: 0600 0000 0400 0000 4000 0000 0000 0000  ........@.......
00000050: 4000 0000 0000 0000 4000 0000 0000 0000  @.......@.......
00000060: d802 0000 0000 0000 d802 0000 0000 0000  ................
00000070: 0800 0000 0000 0000 0300 0000 0400 0000  ................
00000080: 1803 0000 0000 0000 1803 0000 0000 0000  ................
00000090: 1803 0000 0000 0000 1c00 0000 0000 0000  ................
000000a0: 1c00 0000 0000 0000 0100 0000 0000 0000  ................
000000b0: 0100 0000 0400 0000 0000 00
000000c0: 0000 0000 0000 0000 0000 00
000000d0: 2806 0000 0000 0000 2806 00
000000e0: 0010 0000 0000 0000 0100 00
000000f0: 0010 0000 0000 0000 0010 00
00000100: 0010 0000 0000 0000 8101 00
```

```
Program Headers:
  Type           Offset             VirtAddr           PhysAddr
                 FileSiz            MemSiz              Flags  Align
  PHDR           0x0000000000000040 0x0000000000000040 0x0000000000000040
                 0x00000000000002d8 0x00000000000002d8  R      0x8
  INTERP         0x0000000000000318 0x0000000000000318 0x0000000000000318
                 0x000000000000001c 0x000000000000001c  R      0x1
      [Requesting program interpreter: /lib64/ld-linux-x86-64.so.2]
  LOAD           0x0000000000000000 0x0000000000000000 0x0000000000000000
                 0x0000000000000628 0x0000000000000628  R      0x1000
  LOAD           0x0000000000001000 0x0000000000001000 0x0000000000001000
                 0x0000000000000181 0x0000000000000181  R E    0x1000
  LOAD           0x0000000000002000 0x0000000000002000 0x0000000000002000
```

Program Header (Phdr)

p_align

Segment Alignment.

```
00000000: 7f45 4c46 02                                    .ELF.....
00000010: 0300 3e00 01                                    ..>.....
00000020: 4000 0000 00                                    @.......
00000030: 0000 0000 40                                    ....@.8.
00000040: 0600 0000 0400 0000 4000 0000 0000 0000         ........@.......
00000050: 4000 0000 0000 0000 4000 0000 0000 0000         @.......@.......
00000060: d802 0000 0000 0000 d802 0000 0000 0000         ................
00000070: 0800 0000 0000 0000 0300 0000 0400 0000         ................
00000080: 1803 0000 0000 0000 1803 0000 0000 0000         ................
00000090: 1803 0000 0000 0000 1c00 0000 0000 0000         ................
000000a0: 1c00 0000 0000 0000 0100 0000 0000 0000         ................
000000b0: 0100 0000 0400 0000 0000 00
000000c0: 0000 0000 0000 0000 0000 00
000000d0: 2806 0000 0000 0000 2806 00
000000e0: 0010 0000 0000 0000 0100 00
000000f0: 0010 0000 0000 0000 0010 00
00000100: 0010 0000 0000 0000 8101 00
```

```
Program Headers:
  Type           Offset             VirtAddr           PhysAddr
                 FileSiz            MemSiz              Flags  Align
  PHDR           0x0000000000000040 0x0000000000000040 0x0000000000000040
                 0x00000000000002d8 0x00000000000002d8  R      0x8
  INTERP         0x0000000000000318 0x0000000000000318 0x0000000000000318
                 0x000000000000001c 0x000000000000001c  R      0x1
      [Requesting program interpreter: /lib64/ld-linux-x86-64.so.2]
  LOAD           0x0000000000000000 0x0000000000000000 0x0000000000000000
                 0x0000000000000628 0x0000000000000628  R      0x1000
  LOAD           0x0000000000001000 0x0000000000001000 0x0000000000001000
                 0x0000000000000181 0x0000000000000181  R E    0x1000
  LOAD           0x0000000000002000 0x0000000000002000 0x0000000000002000
```

Program Header (Phdr)

```
LOAD           0x0000000000002db8 0x0000000000003db8 0x0000000000003db8
               0x0000000000000258 0x0000000000000260  RW     0x1000
```

Loadable process segment.
p_vaddr % page_size == p_offset % page_size
p_vaddr % p_align == p_offset % p_align
p_align must be power of 2

Program Header (Phdr)

```
LOAD           0x0000000000002db8 0x0000000000003db8 0x0000000000003db8
               0x0000000000000258 0x0000000000000260  RW     0x1000
```

OS load memory by segment,
which contains one or many sections

```
Section to Segment mapping:
 Segment Sections...
  00
  01     .interp
  02     .interp .note.gnu.property .note.gnu.build-id .note.ABI-tag .gnu.hash .dynsym .dynstr .gnu.version .gnu.version_r .rela.dyn .rela.plt
  03     .init .plt .plt.got .plt.sec .text .fini
  04     .rodata .eh_frame_hdr .eh_frame
  05     .init_array .fini_array .dynamic .got .data .bss
  06     .dynamic
  07     .note.gnu.property
  08     .note.gnu.build-id .note.ABI-tag
  09     .note.gnu.property
  10     .eh_frame_hdr
  11
  12     .init_array .fini_array .dynamic .got
```

Program Header (Phdr)

```
LOAD           0x0000000000002db8 0x0000000000003db8 0x0000000000003db8
               0x0000000000000258 0x0000000000000260  RW     0x1000
```

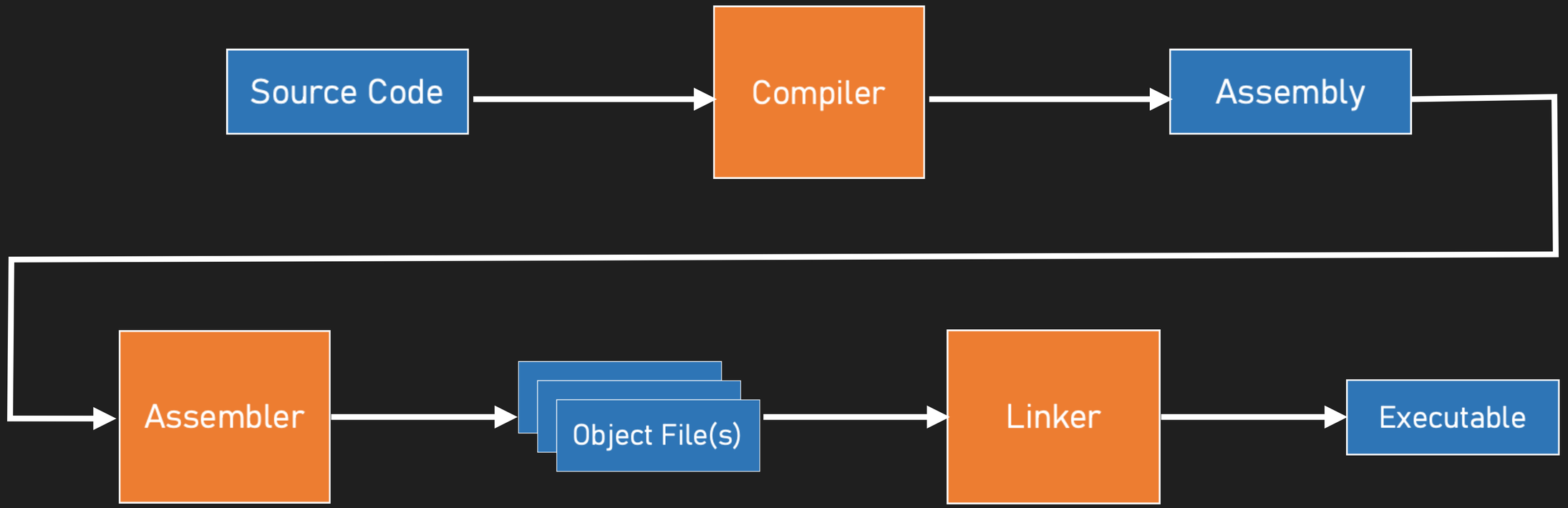
```
In [6]: 0x2db8 % 0x1000 == 0x3db8 % 0x1000
Out[6]: True

In [7]: 0x3db8 % 0x8 == 0x2db8 % 0x8
Out[7]: True
```

Common Section

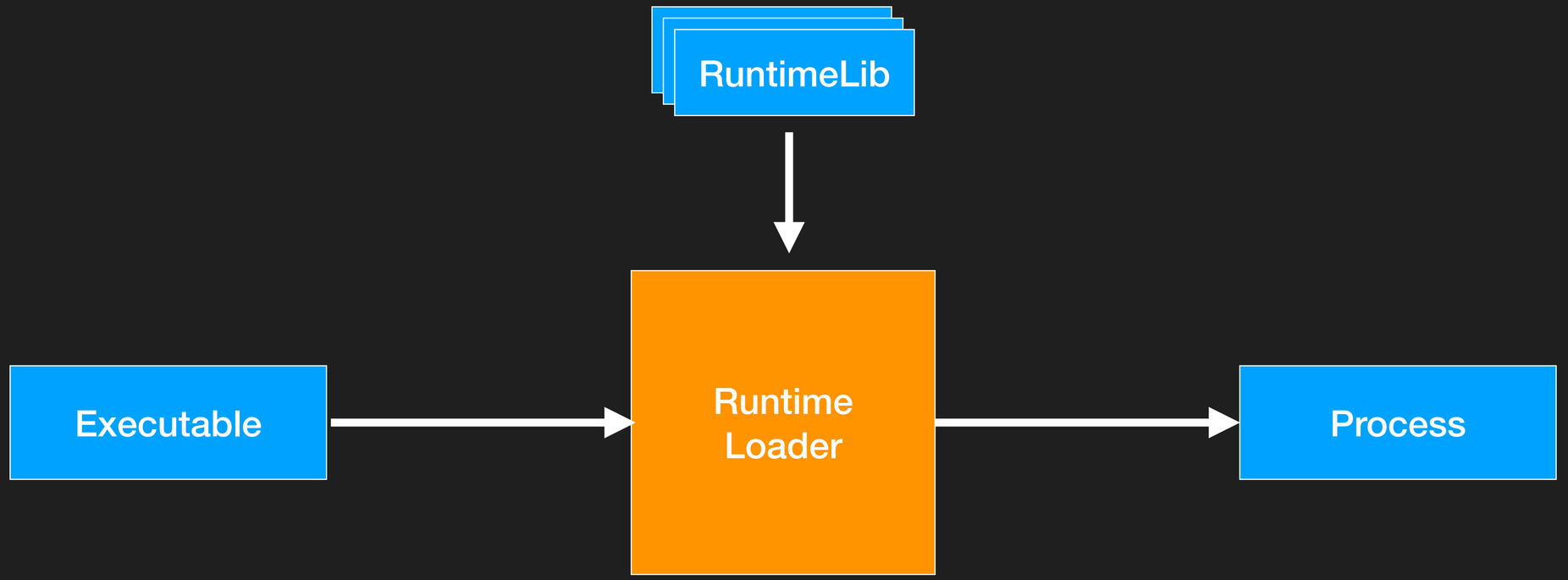
Section	Perm	Description
.text	R-X	Executable Code (Instructions)
.data	RW-	Global with initial data
.rodata	R—	ReadOnly Data
.bss	RW-	Global without initial data

From ELF to Process

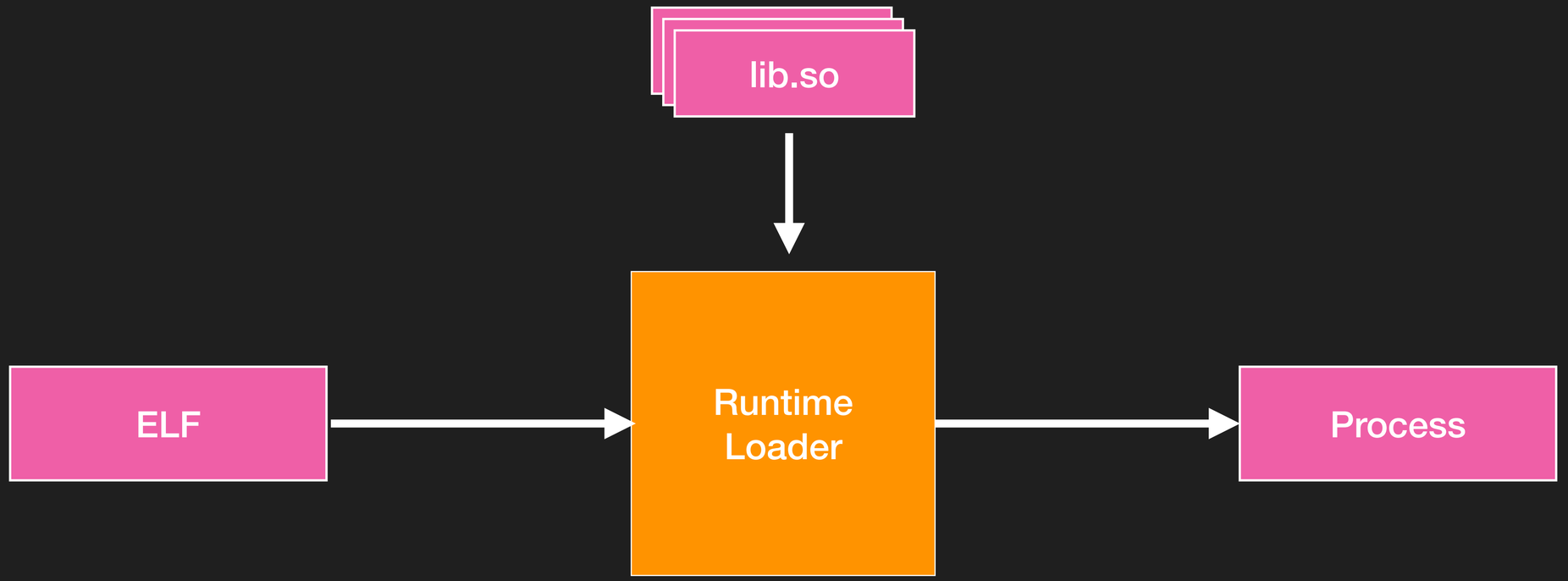
Source to ELF



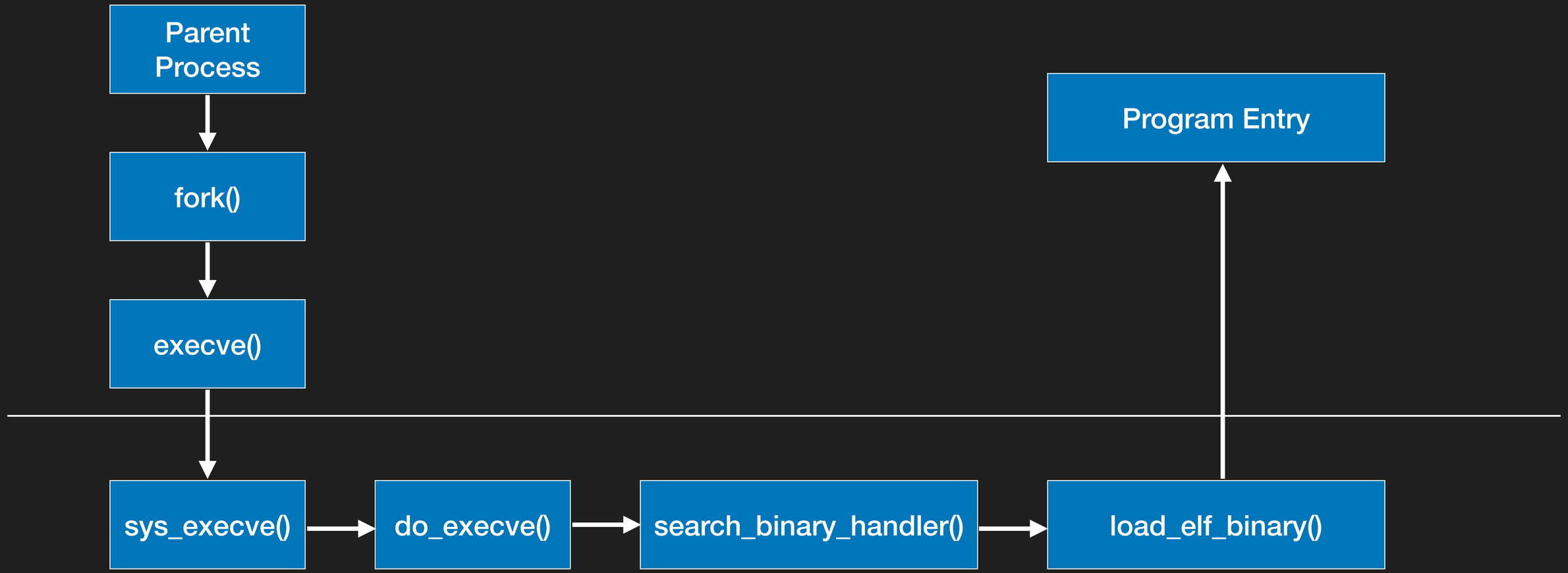
ELF to Process



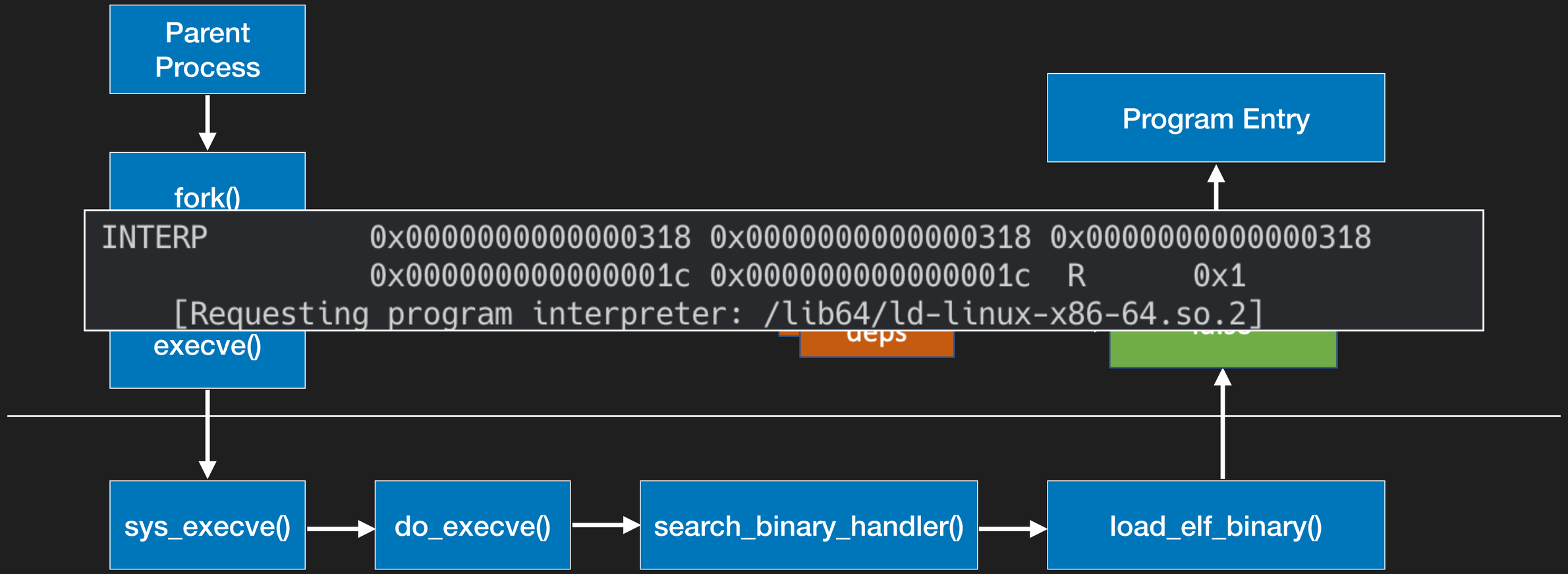
ELF to Process



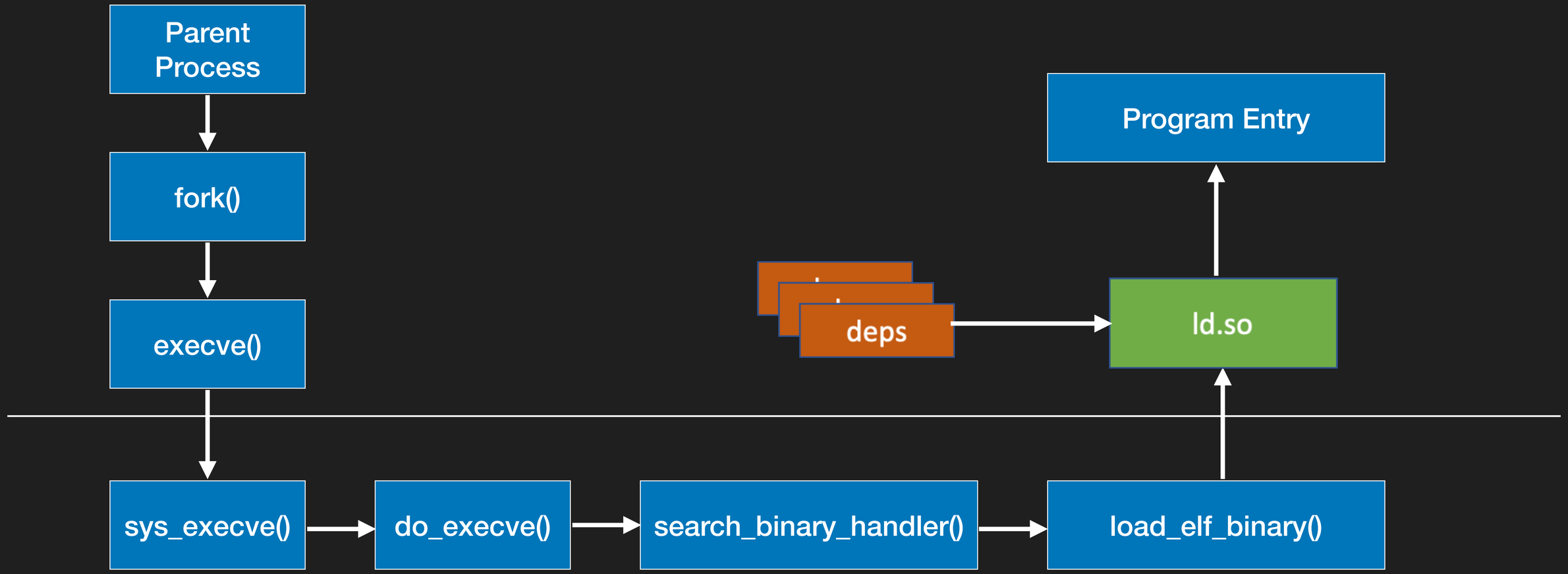
ELF to Process



ELF to Process (DYN)



ELF to Process (DYN)



工欲善其事，必先利其器

逆向工程師的吃飯工具

- 取決於你想要剖析的方向
  - 動態分析
  - 靜態分析

逆向工程師的吃飯工具

- 取決於你想要剖析的方向
  - 動態分析
    - 通過觀察執行流程與結果加以猜測程式本身的行為
    - 網路流量、記憶體的變化、某些 API Call
  - 靜態分析

逆向工程師的吃飯工具

- 取決於你想要剖析的方向
  - 動態分析
    - gdb (GNU Debugger)
      - Plugins (PEDA, pwndbg, gef)
    - Unicorn (CPU emulation, no OS context)
    - Qling Framework (Emulation with OS Context)
  - 靜態分析

逆向工程師的吃飯工具

- 取決於你想要剖析的方向
  - 動態分析
  - 靜態分析
    - 根據執行檔內容直接進行分析
    - Assembly, Decompile, Tables

逆向工程師的吃飯工具

- 取決於你想要剖析的方向
  - 動態分析
  - 靜態分析
    - IDA (Hex-ray, GUI)
    - Radare2 (Opensource, CLI)
      - Cutter for R2 GUI
    - Ghidra (NSA, Java, GUI)

逆向工程師的吃飯工具

- Symbolic Execution
  - 通過收集 Path Constraints 然後丟去 z3 解出答案
  - 將路徑/記憶體內容做成 Symbol
  - Angr
    - Week3 再來聊聊這些酷東西

Lab

HelloWorld

x86-64 Assembly

General-Purpose Registers (AMD-64)

- Accumulator Register
- Counter Register
- Data Register
- Base Register
- Stack Pointer Register
- Source Index Register
- Destination Index Register

General-Purpose Registers (AMD-64)

16-Bit	32-Bit	64-Bit
AX	EAX	RAX
BX	EBX	RBX
CX	ECX	RCX
DX	EDX	RDX
BP	EBP	RBP
SP	ESP	RSP
SI	ESI	RSI
DI	EDI	RDI

General-Purpose Registers (AMD-64)

64	32	16	0
RAX	EAX	AX	
RBX	EBX	BX	
RCX	ECX	CX	
RDX	EDX	DX	
RBP	EBP	BP	
RSP	ESP	SP	
RSI	ESI	SI	
RDI	EDI	DI	

16-Bit	32-Bit	64-Bit
AX	EAX	RAX
BX	EBX	RBX
CX	ECX	RCX
DX	EDX	RDX
BP	EBP	RBP
SP	ESP	RSP
SI	ESI	RSI
DI	EDI	RDI

General-Purpose Registers (AMD-64)

64	32	16	0
RAX	EAX	AH	AL
RBX	EBX	BH	BL
RCX	ECX	CH	CL
RDX	EDX	DH	DL
RBP	EBP	BP	
RSP	ESP	SP	
RSI	ESI	SI	
RDI	EDI	DI	

16-Bit	32-Bit	64-Bit
AX	**EAX**	**RAX**
BX	**EBX**	**RBX**
CX	**ECX**	**RCX**
DX	**EDX**	**RDX**
BP	**EBP**	**RBP**
SP	**ESP**	**RSP**
SI	**ESI**	**RSI**
DI	**EDI**	**RDI**

General-Purpose Registers (AMD-64)

64	32	16	0
RAX	EAX	AH	AL
RBX	EBX	BH	BL
RCX	ECX	CH	CL
RDX	EDX	DH	DL
RBP	EBP	BP	
RSP	ESP	SP	
RSI	ESI	SI	
RDI	EDI	DI	

16-Bit	32-Bit	64-Bit
AX	**EAX**	**RAX**
BX	**EBX**	**RBX**
CX	**ECX**	**RCX**
DX	**EDX**	**RDX**
BP	**EBP**	**RBP**
SP	**ESP**	**RSP**
SI	**ESI**	**RSI**
DI	**EDI**	**RDI**
		R8
		R9
		… R15

We got more registers in x64

Segment Registers

- Stack Segment (SS)
- Code Segment (CS)
- Data Segment (DS)
- Extra Segment (ES)
- F Segment (FS)
- G Segment (GS)
- OS use Paging nowadays, most segment registers are no longer used

Segment Registers

- Stack Segment (SS)
-

3.4.2.1 Segment Registers in 64-Bit Mode

In 64-bit mode: CS, DS, ES, SS are treated as if each segment base is 0, regardless of the value of the associated segment descriptor base. This creates a flat address space for code, data, and stack. FS and GS are exceptions. Both segment registers may be used as additional base registers in linear address calculations (in the addressing of local data and certain operating system data structures).

Even though segmentation is generally disabled, segment register loads may cause the processor to perform segment access assists. During these activities, enabled processors will still perform most of the legacy checks on loaded values (even if the checks are not applicable in 64-bit mode). Such checks are needed because a segment register loaded in 64-bit mode may be used by an application running in compatibility mode.

Limit checks for CS, DS, ES, SS, FS, and GS are disabled in 64-bit mode.

- G Segment (GS)
- OS use Paging nowadays, most segment registers are no longer used

FLAGS

- Actually a Register!
- Something good to know
  - EFLAGS = 32bit FLAGS
  - RFLAGS = 64bit FLAGS

EFLAGS

Reference: Intel® 64 and IA-32 Architectures Software Developer Manuals

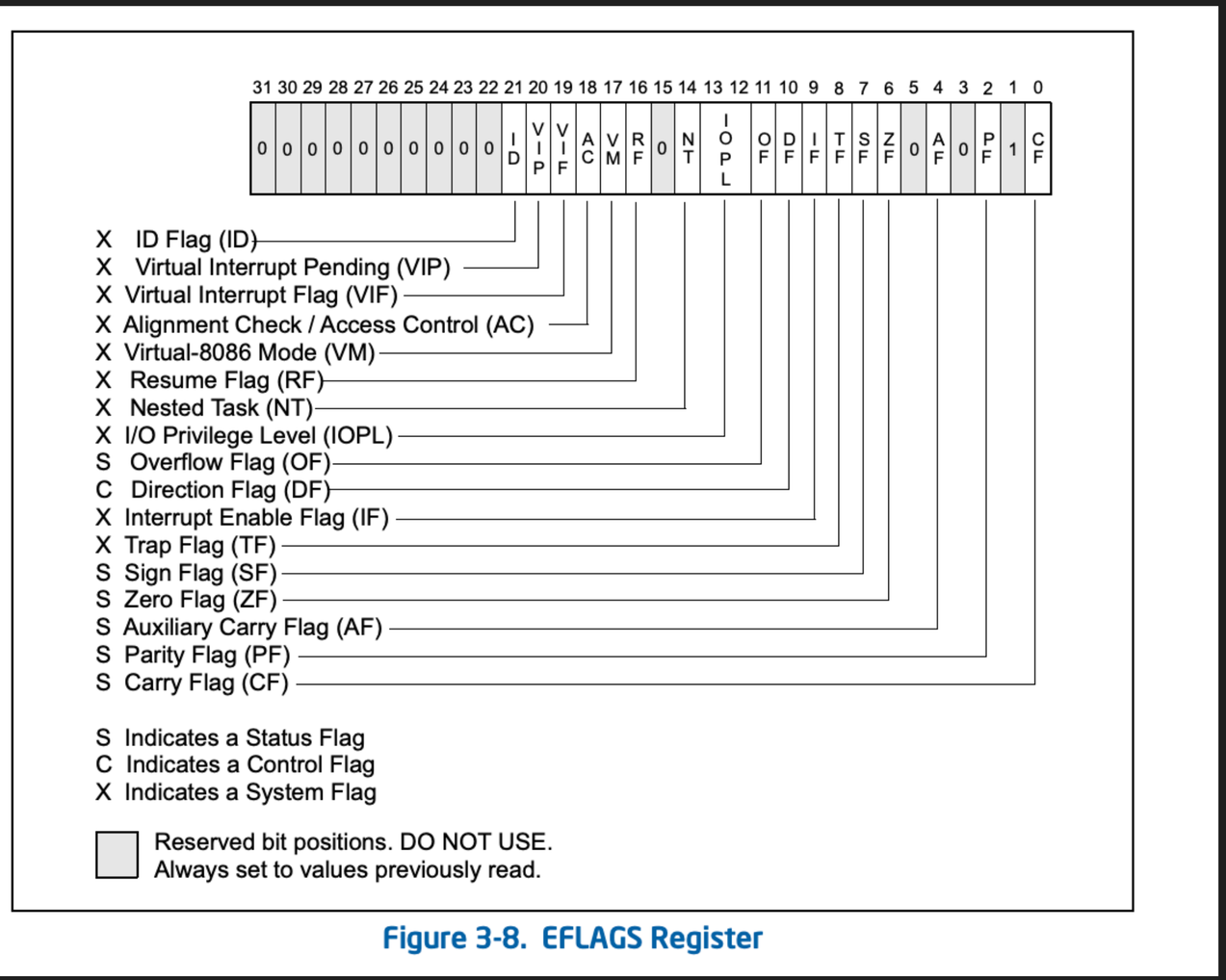


Figure 3-8. EFLAGS Register

RFLAGS

- Nothing… All of them are reserved

Instruction Pointer Register

- Pointed to Instructions
- RIP / EIP / IP

Assembly Syntax

- Intel Syntax
  - `mov rax, rbx // rax = rbx`
- AT&T Syntax
  - `mov %rbx, %rax // rbx->rax`

Moving Data

```
// mov data/register
mov rax, rbx                // rax = rbx
mov rax, 0xdeadbeef         // rax = 0xdeadbeef
mov al, byte ptr [rbx]      // char al = *(char*)
mov eax, dword ptr [rbx]    // uint32_t eax ...

// Load effective address
lea rax, [0xdeadbeef]       // rax = 0xdeadbeef
lea rax, [rax*2+32]         // rax = 0xdeadbeef*2+32
```

Operations

```
add rax, rbx  // rax = rax+rbx
sub rax, rbx  // rax = rax-rbx
inc rax       // rax++
dec rax       // rax--
and rax, rbx  // rax = rax&rbx
or rax, rbx   // rax = rax|rbx
// logical shift
shl rax, 3
shr rax, 3
// arithmetic shift
sal rax, 3
sar rax, 3
```

```
xor rax, rbx  // rax = rax^rbx
not rax       // rax = ~rax

neg rax       // rax = -rax

dec rax       // rax--

shl rax, 3
cmp rax, rbx
// rax - rbx (set flags)
test rax, rbx
// rax & rbx (set flags)
```

Comparing

cmp A, B

ZF	CF	Result
F	T	A < B
F	F	A > B
T	F	A = B

Condition	Result
Sign != Overflow	A < B
Sign = Overflow	A > B
Zero = True	A = B

Condition Jump

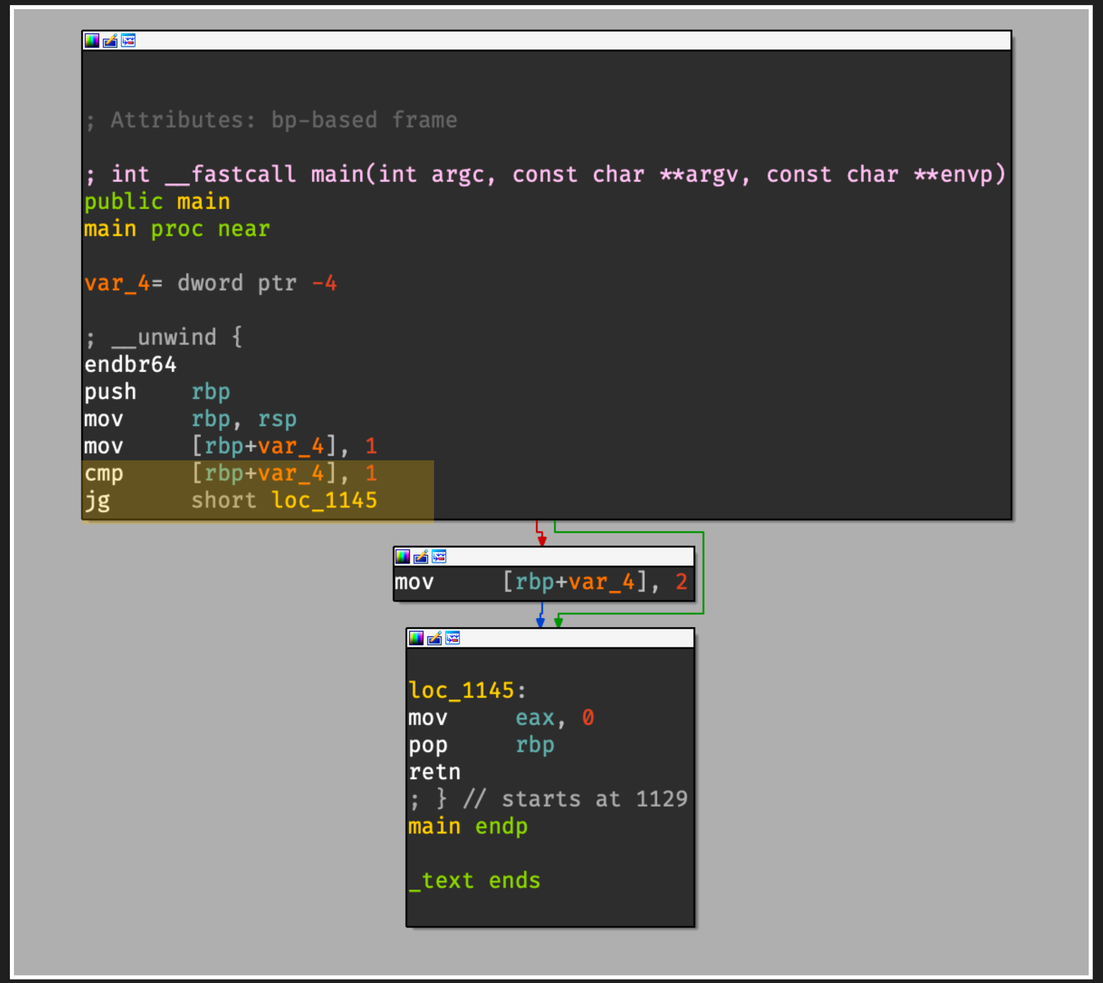
JA	If above
JNBE	If not below / not equal
JAE	If above / equal
JB	If below
JNB	If not below
JNAE	If not above or not equal
JBE	If below or equal
JNA	If not above

JG	If greater
JNLE	If not less or equal
JGE	If greater / equal
JNL	If not less
JL	If less
JLE	If less or equal
JNG	If not greater
JNGE	If not greater or equal

JE	If equal
JNE	If not equal
JMP	JUST JUMP

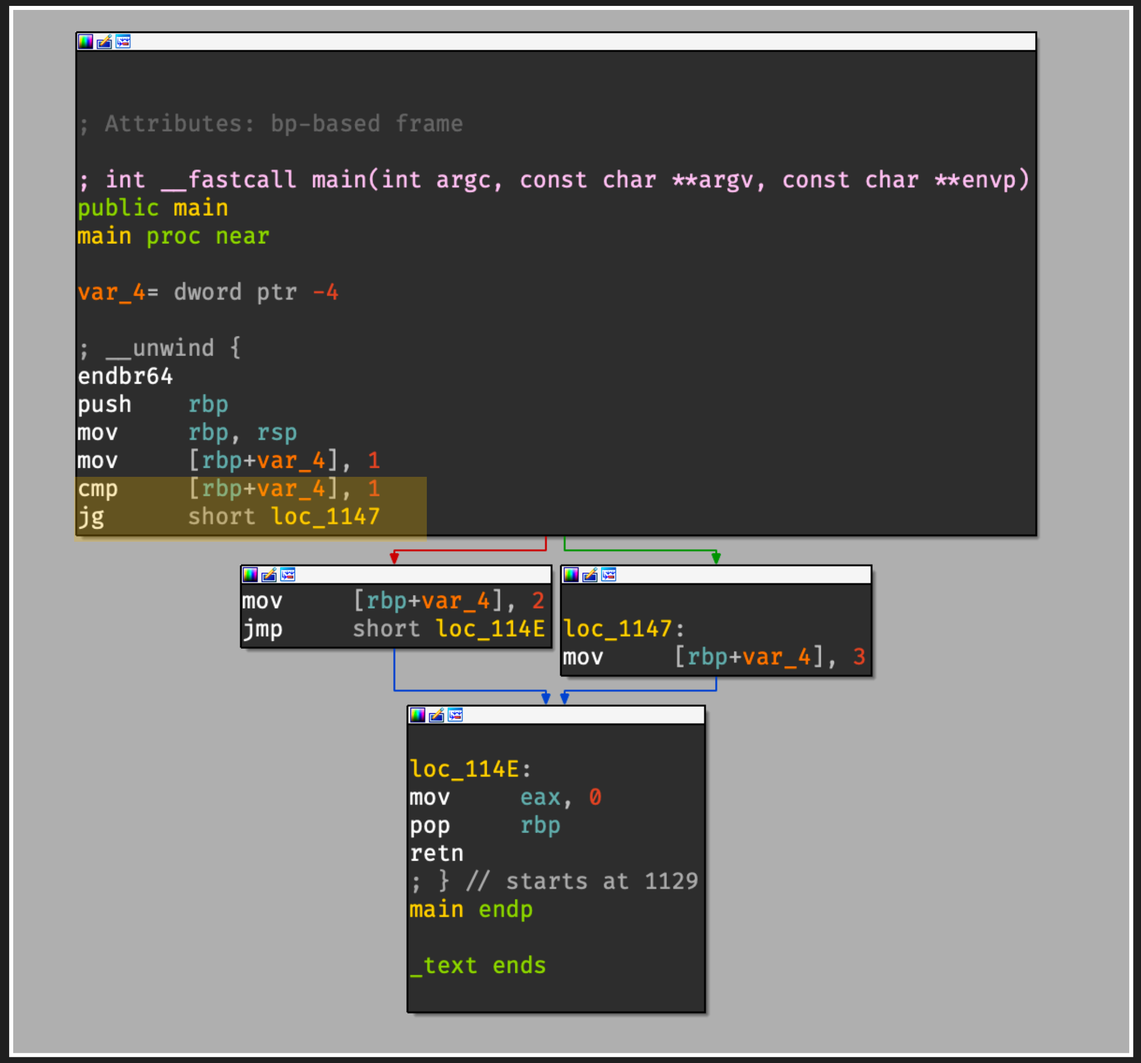
IF

```c
int main() {
    int a = 1;
    if (a < 2) {
        a = 2;
    }

    return 0;
}
```



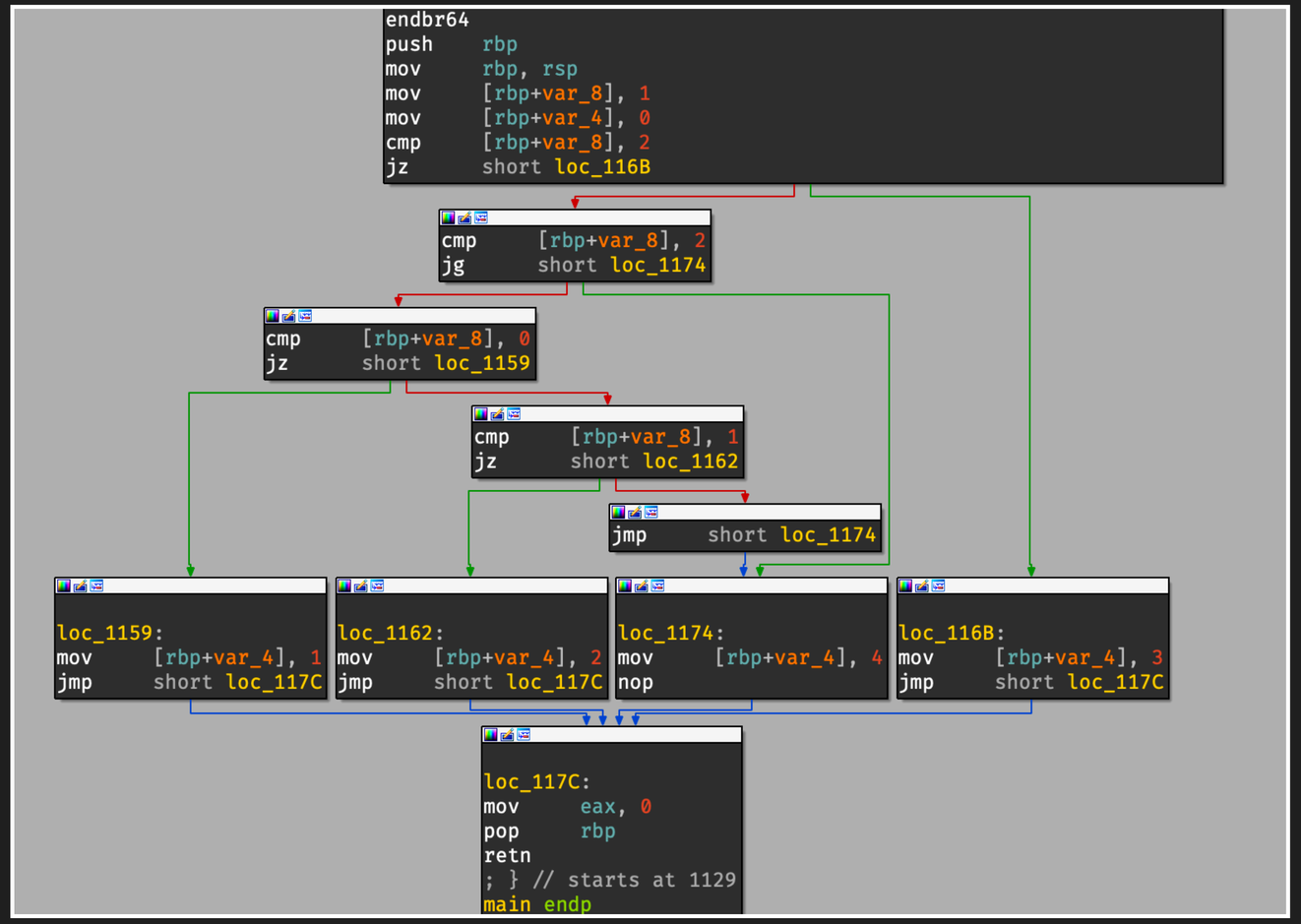
IF-ELSE

```c
int main() {
    int a = 1;
    if (a < 2)
{       a = 2;
    } else {
        a = 3;
    }

    return 0;
}
```



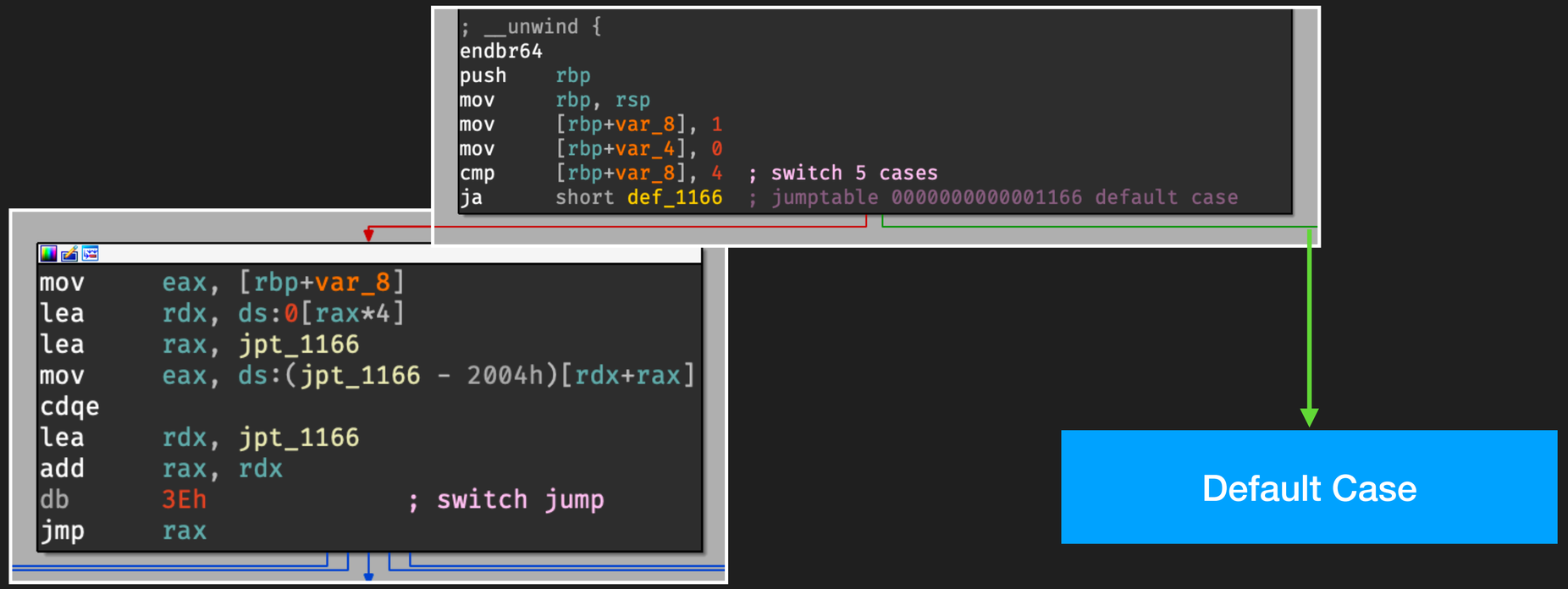
Switch (3 Cases)

```c
int main() {
    int a = 1;
    int b = 0;

    switch (a) {
        case 0:
            b = 1;
            break;
        case 1:
            b = 2;
            break;
        case 2:
            b = 3;
            break;
        default:
            b = 4;
            break;
    }

    return 0;
}
```



Switch (5 Cases)

Case夠多的時候，用 JumpTable 來做 Switch
因為直接做會太多 instruction 用來串巢狀結構



Lab

AssemblyDev

Calling Convention

呼叫慣例

Calling Convention

- 程式呼叫子邏輯的時候用來傳遞參數的慣例
- x86 呼叫慣例大部分會將 Parameters 往 Stack 上面堆
  - 比較多奇怪的 Calling Convention
  - 回傳值放到 edx:eax (超過 32bit 的話)
- x64 則是依賴 Register 傳參
  - 就一種，~~我們先不管 Windows :D~~
  - 回傳值放到 rax 上

x86 fastcall

- 前兩個參數分別放於 ecx, edx
- 其他都堆到 Stack 上

```
public main

endbr32
lea     ecx, [esp+4]
and     esp, 0FFFFFFF0h
push    dword ptr [ecx-4]
push    ebp
mov     ebp, esp
push    ecx
sub     esp, 4
call    __x86_get_pc_thunk_ax
add     eax, (offset
_GLOBAL_OFFSET_TABLE_ - $)
sub     esp, 0Ch
push    3
mov     edx, 2
mov     ecx, 1
call    addIntegers
add     esp, 0Ch
mov     eax, 0
mov     ecx, [ebp+var_4]
leave
lea     esp, [ecx-4]
retn
```

```
public addIntegers
addIntegers proc near

endbr32
push    ebp
mov     ebp, esp
push    ebx
sub     esp, 14h
call    __x86_get_pc_thunk_ax
add     eax, (offset _GLOBAL_OFFSET_TABLE_ - $)
mov     [ebp+var_C], ecx
mov     [ebp+var_10], edx
mov     ecx, [ebp+var_C]
mov     edx, [ebp+var_10]
add     ecx, edx
mov     edx, [ebp+arg_0]
add     edx, ecx
sub     esp, 8
push    edx
lea     edx, (unk_2008 - 3FD8h)[eax]
push    edx             ; format
mov     ebx, eax
call    _printf
add     esp, 10h
nop
mov     ebx, [ebp+var_4]
leave
retn    4
```

x86 thiscall

- ecx 用來存放 this pointer
- 其他全都堆到 Stack 上面

x86 stdcall

- 在 Win32 上面用的
- 參數全都堆到 Stack 上面
- 由 Callee 做 Stack Cleanup

```
public main

endbr32
lea     ecx, [esp+4]
and     esp, 0FFFFFFF0h
push    dword ptr [ecx-4]
push    ebp
mov     ebp, esp
push    ecx
sub     esp, 4
call    __x86_get_pc_thunk_ax
add     eax, (offset
_GLOBAL_OFFSET_TABLE_ - $)
sub     esp, 4
push    3
push    2
push    1
call    addIntegers
add     esp, 4
mov     eax, 0
mov     ecx, [ebp+var_4]
leave
lea     esp, [ecx-4]
retn
```

Stack Alignment

```
public addIntegers
addIntegers proc near

endbr32
push    ebp
mov     ebp, esp
push    ebx
sub     esp, 4
call    __x86_get_pc_thunk_ax
add     eax, (offset _GLOBAL_OFFSET_TABLE_ - $)
mov     ecx, [ebp+arg_0]
mov     edx, [ebp+arg_4]
add     ecx, edx
mov     edx, [ebp+arg_8]
add     edx, ecx
sub     esp, 8
push    edx
lea     edx, (unk_2008 - 3FD8h)[eax]
push    edx             ; format
mov     ebx, eax
call    _printf
add     esp, 10h
nop
mov     ebx, [ebp+var_4]
leave
retn    0Ch
```

Stack Cleanup

x86 cdecl

- 參數全都堆到 Stack 上面
- 由 Caller 做 Stack Cleanup

```
public main

endbr32
lea     ecx, [esp+4]
and     esp, 0FFFFFFF0h
push    dword ptr [ecx-4]
push    ebp
mov     ebp, esp
push    ecx
sub     esp, 4
call    __x86_get_pc_thunk_ax
add     eax, (offset
_GLOBAL_OFFSET_TABL
sub     esp, 4
push    3
push    2
push    1
call    addIntegers
add     esp, 10h
mov     eax, 0
mov     ecx, [ebp+var_4]
leave
lea     esp, [ecx-4]
retn
```

Stack Alignment

Stack Cleanup

```
public addIntegers

endbr32
push    ebp
mov     ebp, esp
push    ebx
sub     esp, 4
call    __x86_get_pc_thunk_ax
add     eax, (offset _GLOBAL_OFFSET_TABLE_ - $)
mov     ecx, [ebp+arg_0]
mov     edx, [ebp+arg_4]
add     ecx, edx
mov     edx, [ebp+arg_8]
add     edx, ecx
sub     esp, 8
push    edx
lea     edx, (unk_2008 - 3FD8h)[eax]
push    edx             ; format
mov     ebx, eax
call    _printf
add     esp, 10h
nop
mov     ebx, [ebp+var_4]
leave
retn
```

x64 Calling Convention

- Syscall 跟 C ABI 有一點點差異
- Syscall Calling Convention
  - Syscall Number: RAX
  - Parameters: RDI, RSI, RDX, r10, r8, r9, Stack
  - Return Value: RAX

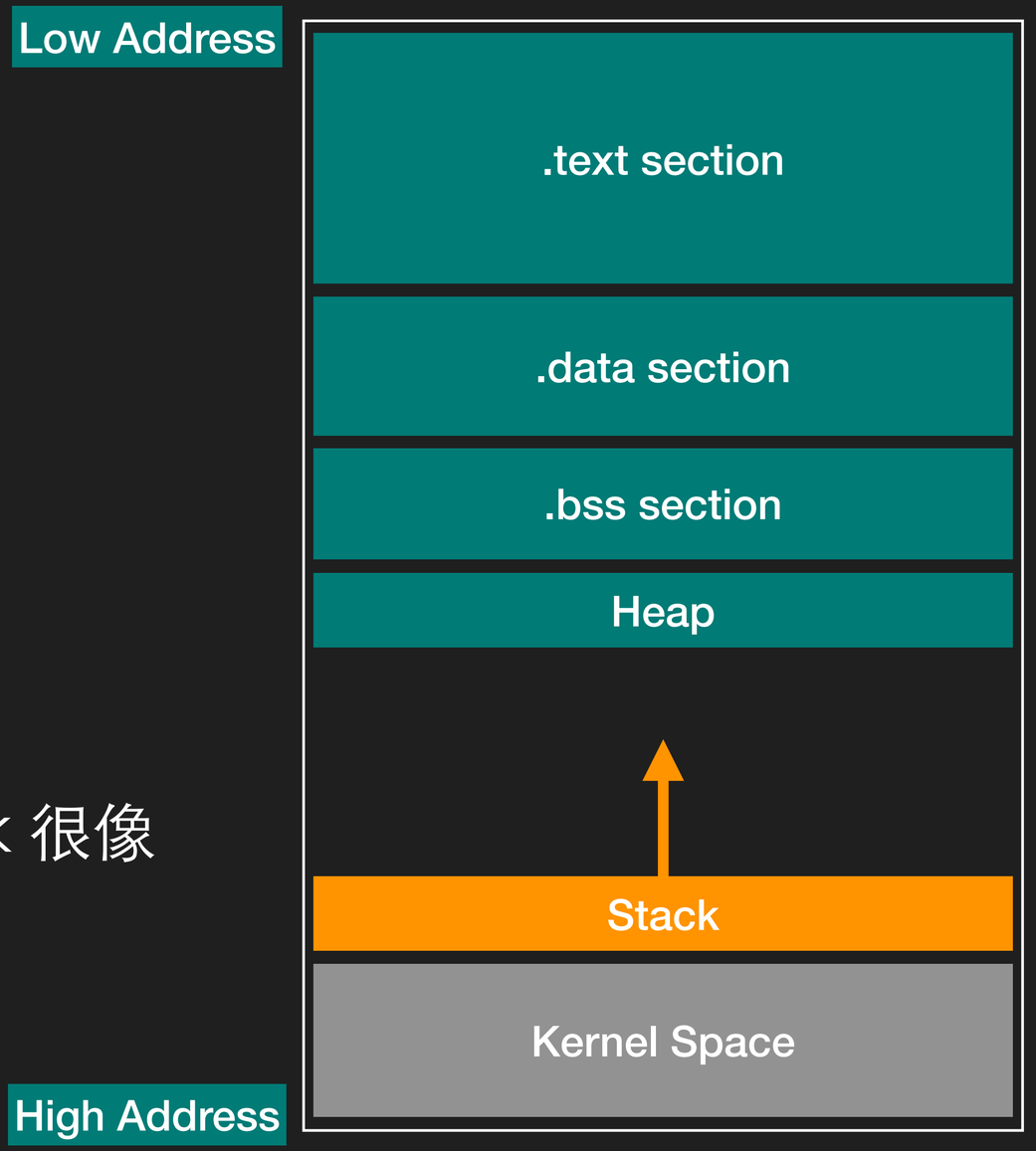
x64 Calling Convention

- C ABI
  - Return Value: RDX:RAX
  - Parameters: RDI, RSI, RDX, RCX, r8, r9, Stack

Stack Frame

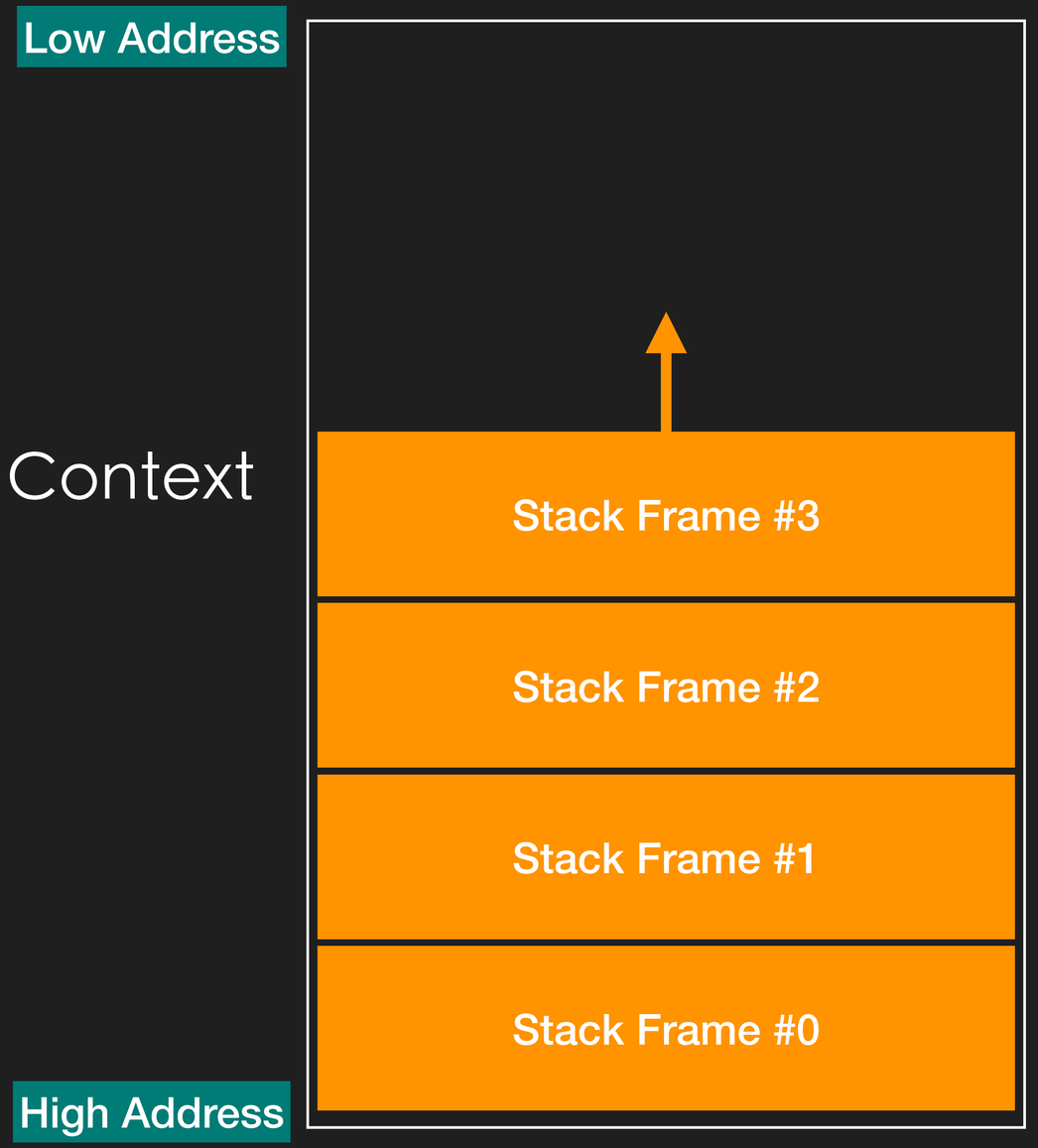
Stack

- Stack 從高位址往低位址成長
  - 倒過來成長
  - 支援 push, pop 跟資料結構的 Stack 很像



Stack Frame

- Stack Frame 乘載了當前 Function 的 Context
  - Return Address
  - Local Datas



Stack Register

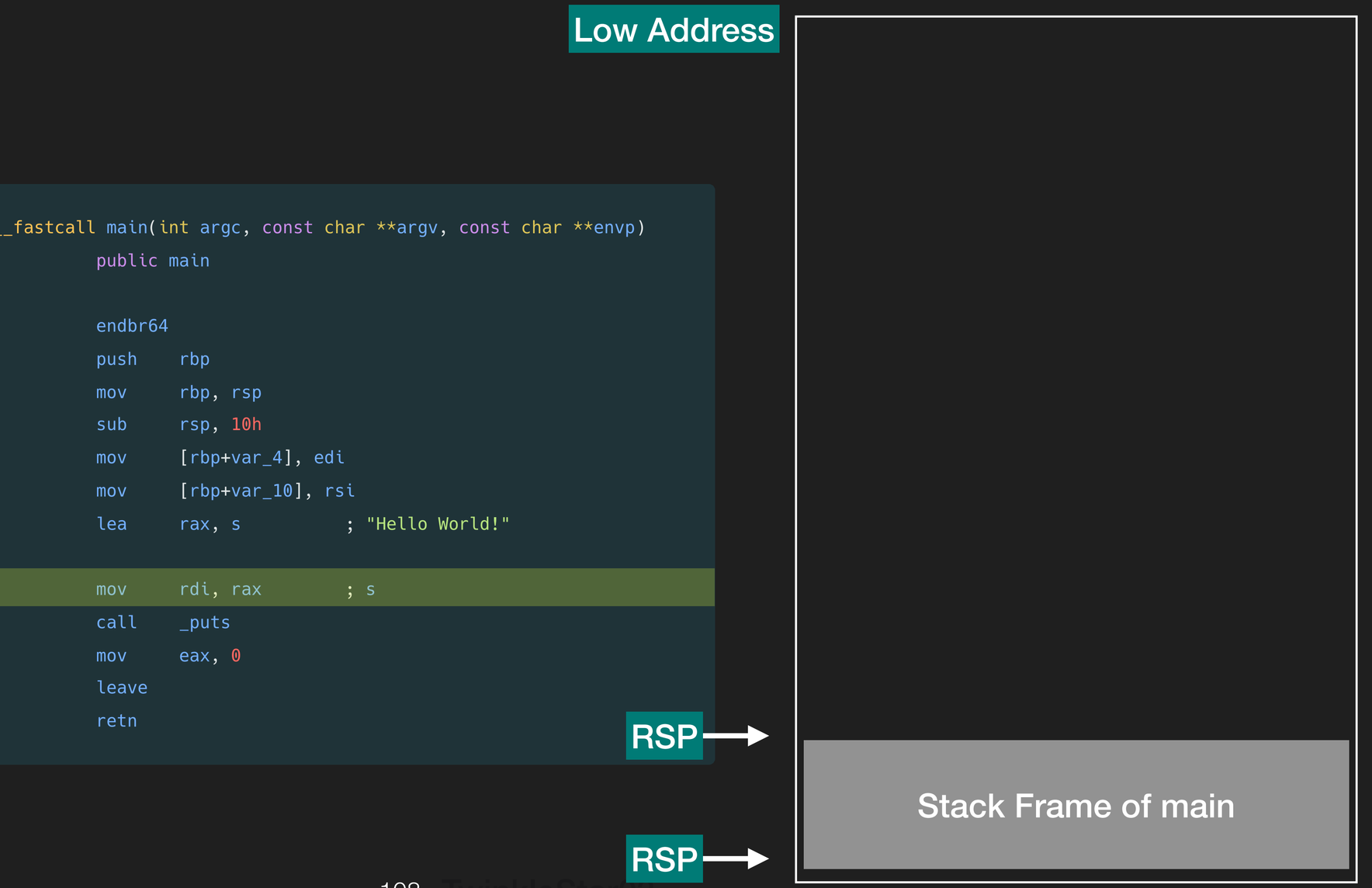
- SP, BP
- Stack Pointer Register
  - 指向 Stack Frame 的頂端
- Stack Base Pointer Register
  - 指向 Stack Frame 的底部

Call

```
.text:0000000000001149                  ; int __fastcall main(int argc, const char **argv, const char **envp)
.text:0000000000001149                  public main

.text:0000000000001149 F3 0F 1E FA      endbr64
.text:000000000000114D 55               push    rbp
.text:000000000000114E 48 89 E5         mov     rbp, rsp
.text:0000000000001151 48 83 EC 10      sub     rsp, 10h
.text:0000000000001155 89 7D FC         mov     [rbp+var_4], edi
.text:0000000000001158 48 89 75 F0      mov     [rbp+var_10], rsi
.text:000000000000115C 48 8D 05 A1 0E 00 lea    rax, s          ; "Hello World!"
.text:000000000000115C 00
.text:0000000000001163 48 89 C7         mov     rdi, rax        ; s
.text:0000000000001166 E8 E5 FE FF FF   call    _puts
.text:000000000000116B B8 00 00 00 00   mov     eax, 0
.text:0000000000001170 C9               leave
.text:0000000000001171 C3               retn
```

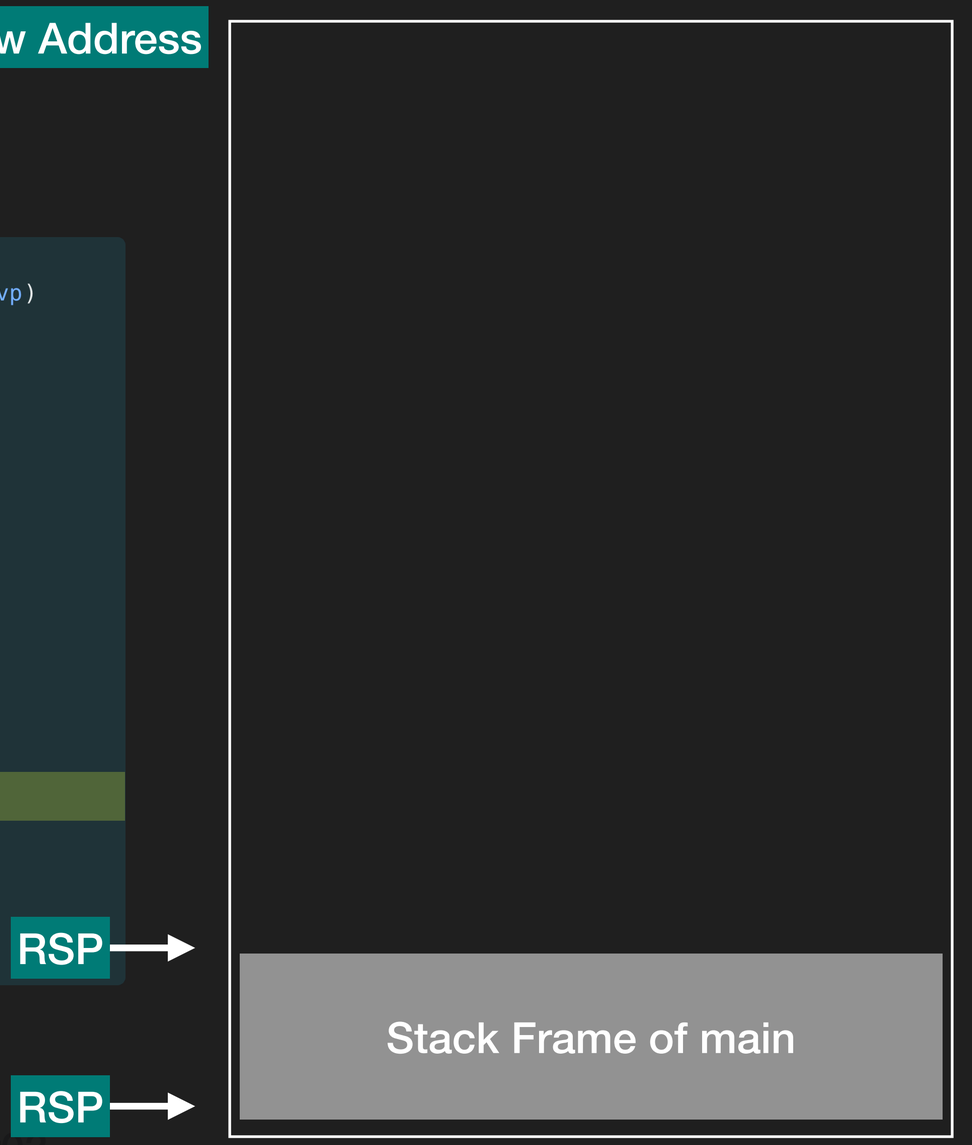
Call

```
.text:0000000000001149                  ; int __fastcall main(int argc, const char **argv, const char **envp)
.text:0000000000001149                  public main

.text:0000000000001149 F3 0F 1E FA      endbr64
.text:000000000000114D 55               push    rbp
.text:000000000000114E 48 89 E5         mov     rbp, rsp
.text:0000000000001151 48 83 EC 10      sub     rsp, 10h
.text:0000000000001155 89 7D FC         mov     [rbp+var_4], edi
.text:0000000000001158 48 89 75 F0      mov     [rbp+var_10], rsi
.text:000000000000115C 48 8D 05 A1 0E 00 lea    rax, s          ; "Hello World!"
.text:000000000000115C 00
.text:0000000000001163 48 89 C7         mov     rdi, rax        ; s
.text:0000000000001166 E8 E5 FE FF FF   call    _puts
.text:000000000000116B B8 00 00 00 00   mov     eax, 0
.text:0000000000001170 C9               leave
.text:0000000000001171 C3               retn
```



Call

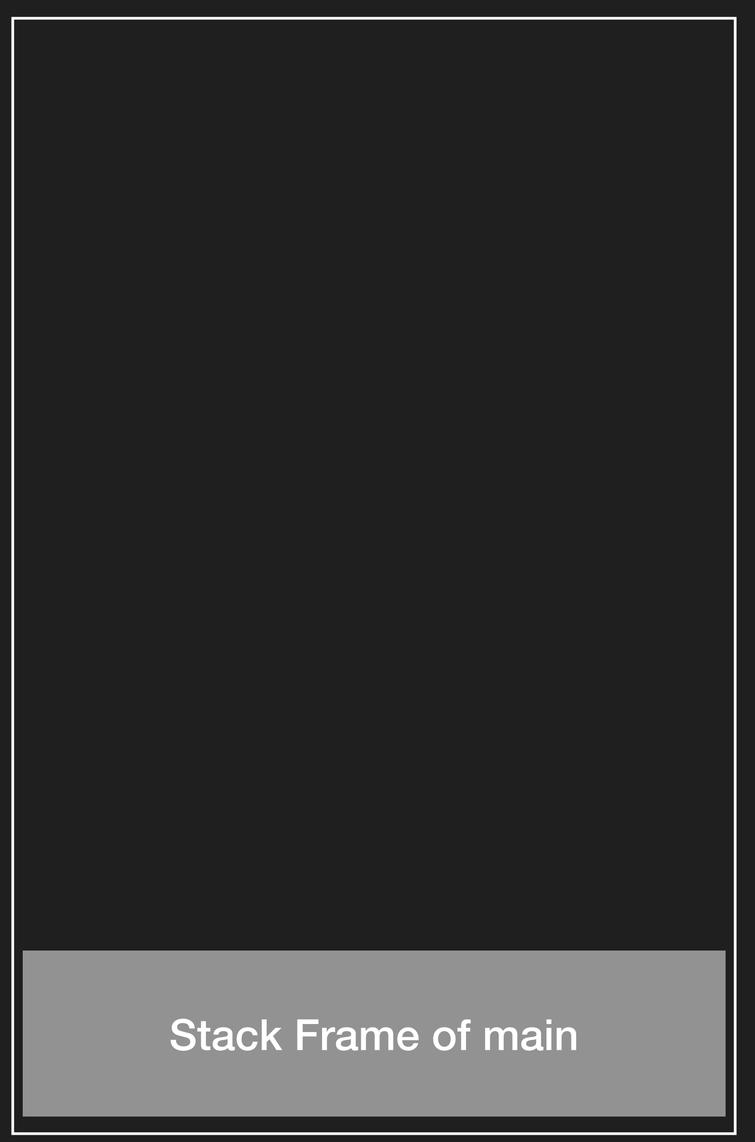
```
.text:0000000000001149                  ; int __fastcall main(int argc, const char **argv, const char **envp)
.text:0000000000001149                  public main

.text:0000000000001149 F3 0F 1E FA      endbr64
.text:000000000000114D 55               push    rbp
.text:000000000000114E 48 89 E5         mov     rbp, rsp
.text:0000000000001151 48 83 EC 10      sub     rsp, 10h
.text:0000000000001155 89 7D FC         mov     [rbp+var_4], edi
.text:0000000000001158 48 89 75 F0      mov     [rbp+var_10], rsi
.text:000000000000115C 48 8D 05 A1 0E 00 lea    rax, s          ; "Hello World!"
.text:000000000000115C 00
.text:0000000000001163 48 89 C7         mov     rdi, rax        ; s
.text:0000000000001166 E8 E5 FE FF FF   call    _puts
.text:000000000000116B B8 00 00 00 00   mov     eax, 0
.text:0000000000001170 C9               leave
.text:0000000000001171 C3               retn
```



Call

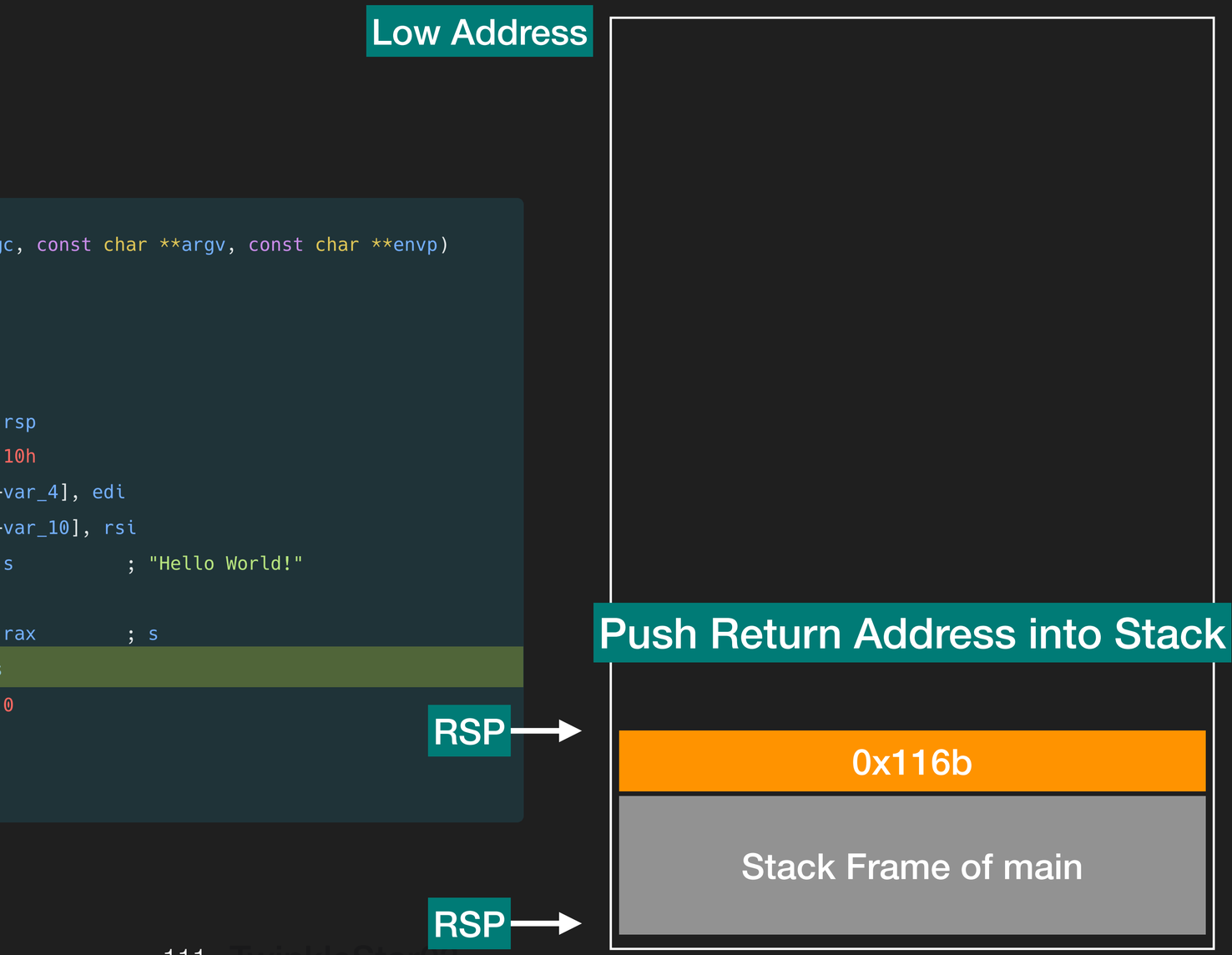
```
.text:0000000000001149                  ; int __fastcall main(int argc, const char **argv, const char **envp)
.text:0000000000001149                  public main

.text:0000000000001149 F3 0F 1E FA      endbr64
.text:000000000000114D 55               push    rbp
.text:000000000000114E 48 89 E5         mov     rbp, rsp
.text:0000000000001151 48 83 EC 10      sub     rsp, 10h
.text:0000000000001155 89 7D FC         mov     [rbp+var_4], edi
.text:0000000000001158 48 89 75 F0      mov     [rbp+var_10], rsi
.text:000000000000115C 48 8D 05 A1 0E 00 lea    rax, s          ; "Hello World!"
.text:000000000000115C 00
.text:0000000000001163 48 89 C7         mov     rdi, rax        ; s
.text:0000000000001166 E8 E5 FE FF FF   call    _puts
.text:000000000000116B B8 00 00 00 00   mov     eax, 0
.text:0000000000001170 C9               leave
.text:0000000000001171 C3               retn
```

call = push Return Address (0x116b) + jmp



Call

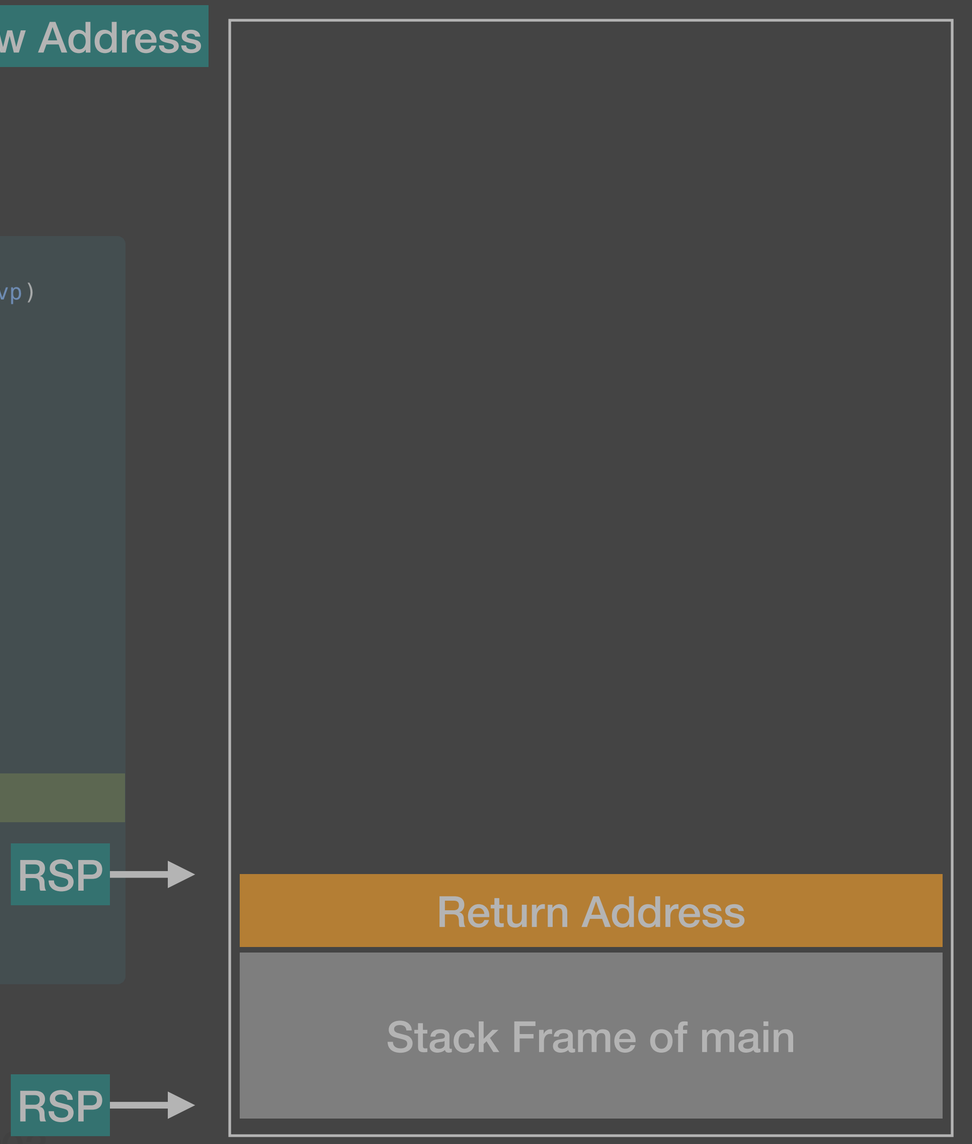
```
.text:0000000000001149                  ; int __fastcall main(int argc, const char **argv, const char **envp)
.text:0000000000001149                  public main

.text:0000000000001149 F3 0F 1E FA      endbr64
.text:000000000000114D 55               push    rbp
.text:000000000000114E 48 89 E5         mov     rbp, rsp
.text:0000000000001151 48 83 EC 10      sub     rsp, 10h
.text:0000000000001155 89 7D FC         mov     [rbp+var_4], edi
.text:0000000000001158 48 89 75 F0      mov     [rbp+var_10], rsi
.text:000000000000115C 48 8D 05 A1 0E 00 lea    rax, s          ; "Hello World!"
.text:000000000000115C 00
.text:0000000000001163 48 89 C7         mov     rdi, rax        ; s
.text:0000000000001166 E8 E5 FE FF FF   call    _puts
.text:000000000000116B B8 00 00 00 00   mov     eax, 0
.text:0000000000001170 C9               leave
.text:0000000000001171 C3               retn
```



Call

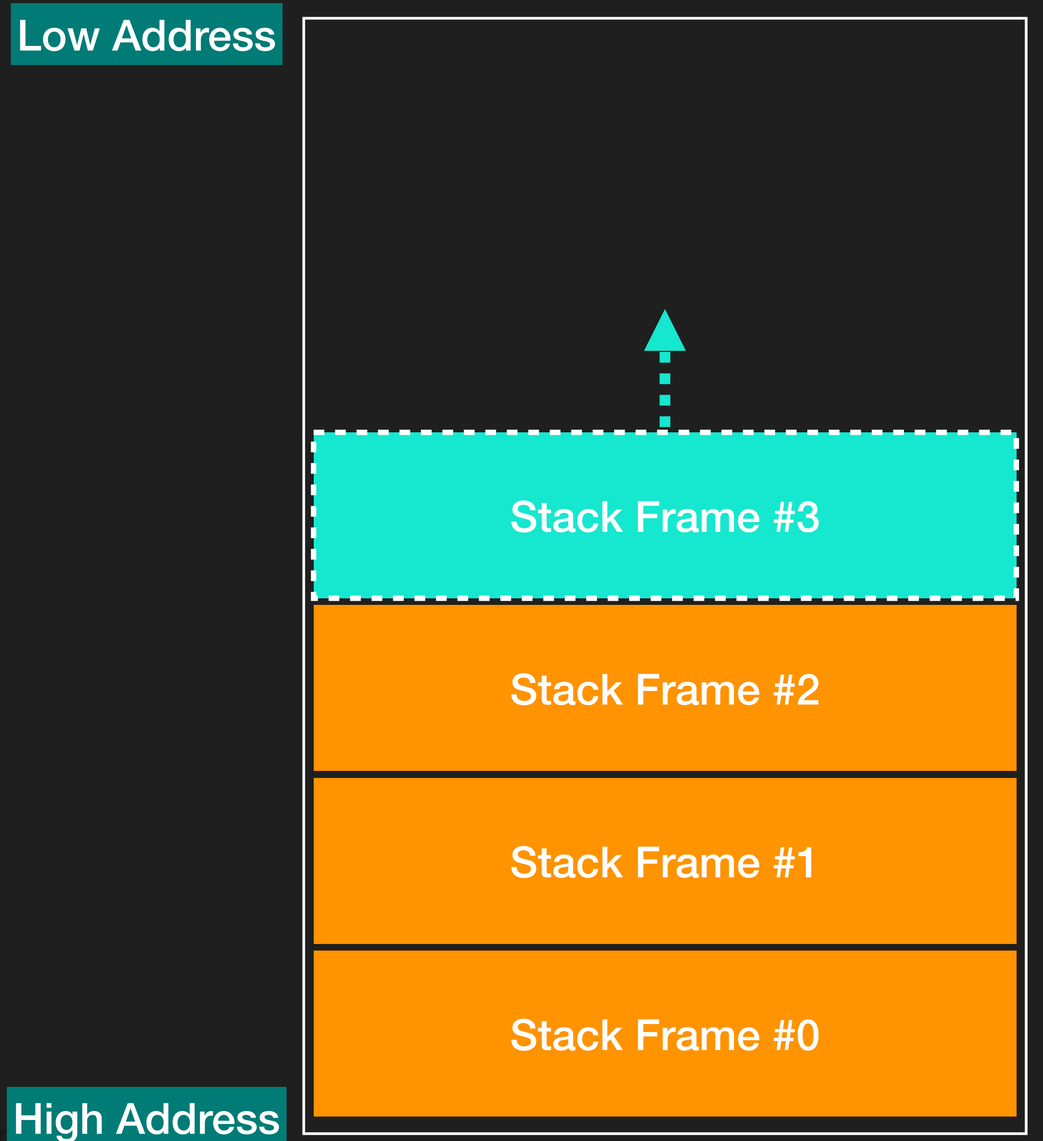
```
.text:0000000000001149                  ; int __fastcall main(int argc, const char **argv, const char **envp)
.text:0000000000001149                  public main

.text:0000000000001149 F3 0F 1E FA      endbr64
.text:000000000000114D 55               push    rbp
.text:000000000000114E 48 89 E5         mov     rbp, rsp
.text:0000000000001151 48 83 EC 10      sub     rsp, 10h
.text:0000000000001155 89 7D FC         mov     [rb
.text:0000000000001158 48 89 75 F0      mov     [rb
.text:000000000000115C 48 8D 05 A1 0E 00  lea     rax, s          ; "Hello World!"
.text:000000000000115C 00
.text:0000000000001163 48 89 C7         mov     rdi, rax        ; s
.text:0000000000001166 E8 E5 FE FF FF   call    _puts
.text:000000000000116B B8 00 00 00 00   mov     eax, 0
.text:0000000000001170 C9               leave
.text:0000000000001171 C3               retn
```

jmp to _puts…



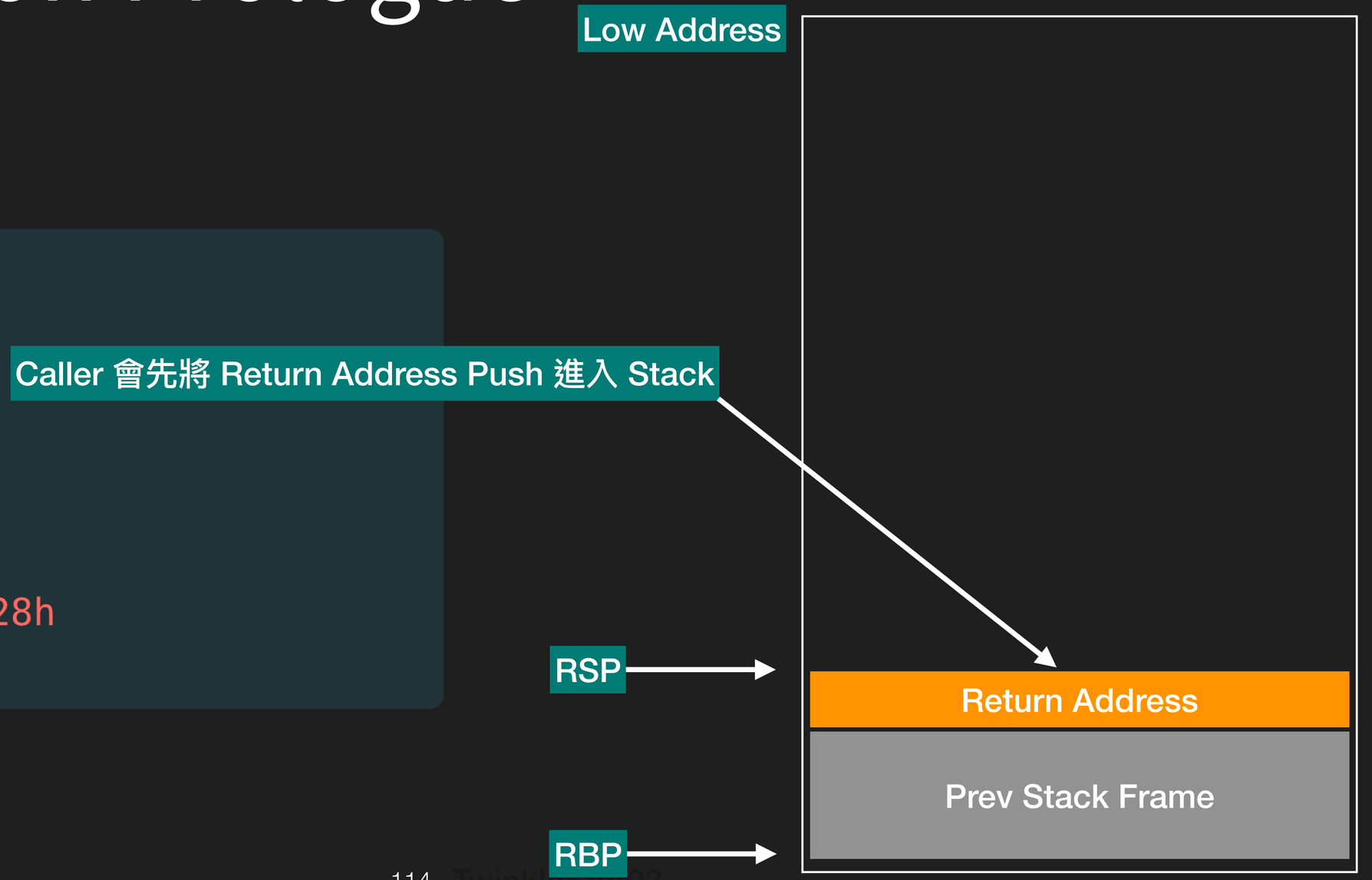
Function Prologue

- Function 開始執行前的前置作業
  - 規劃出所需要的 Stack 空間
  - 建立新的 Function Context



Function Prologue

```
endbr64
push    rbp
mov     rbp, rsp
sub     rsp, 30h
mov     rax, fs:28h
```

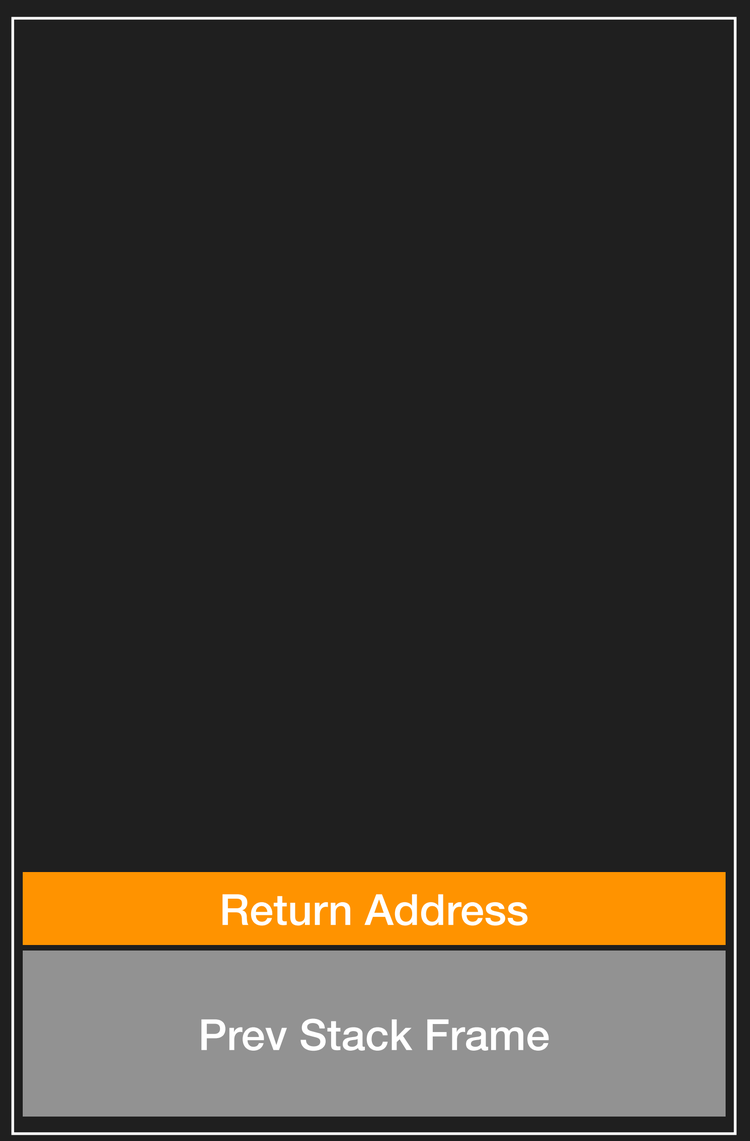


Function Prologue

CET Instruction

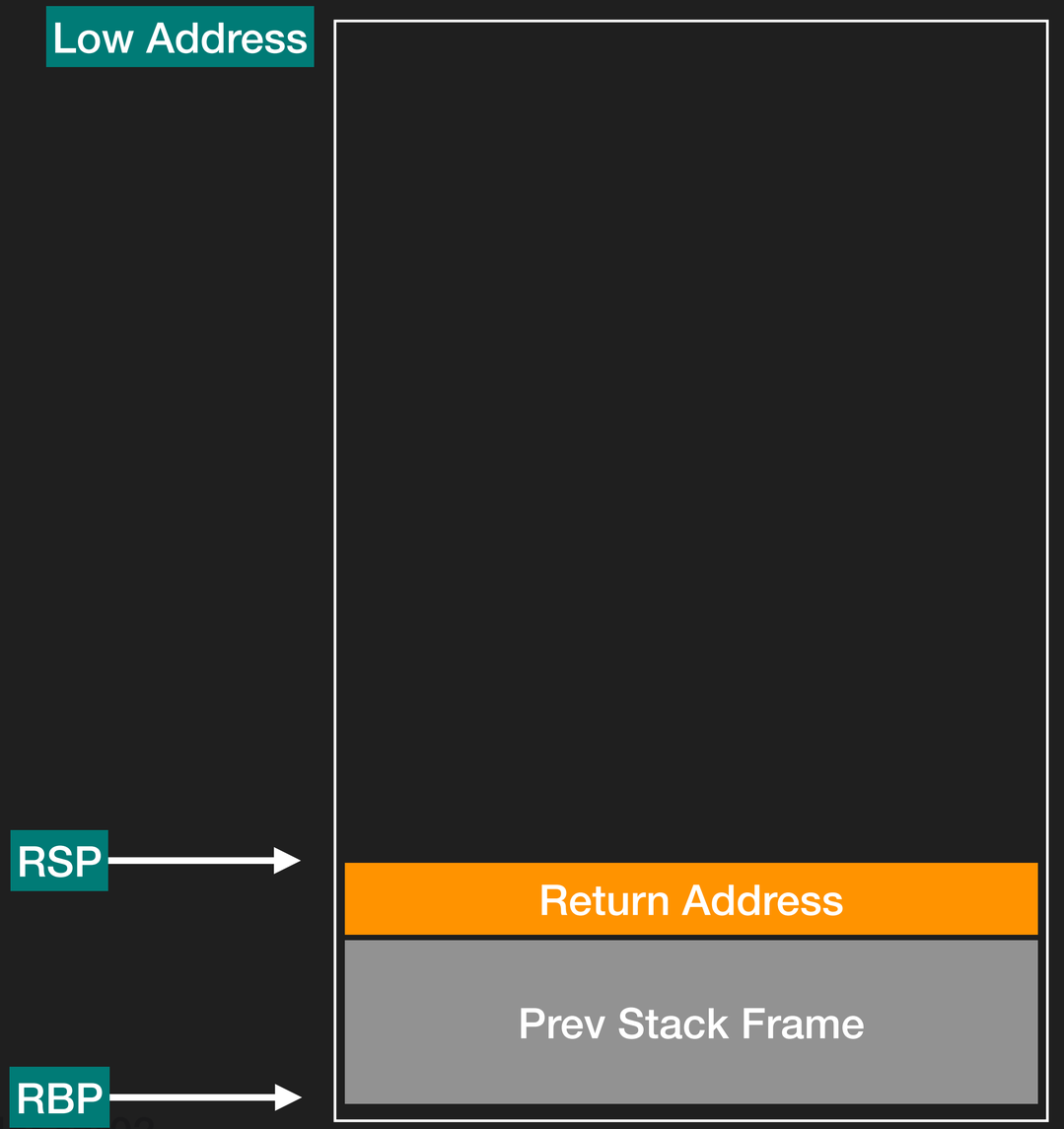
```
endbr64
push    rbp
mov     rbp, rsp
sub     rsp, 30h
mov     rax, fs:28h
```

Stack Canary



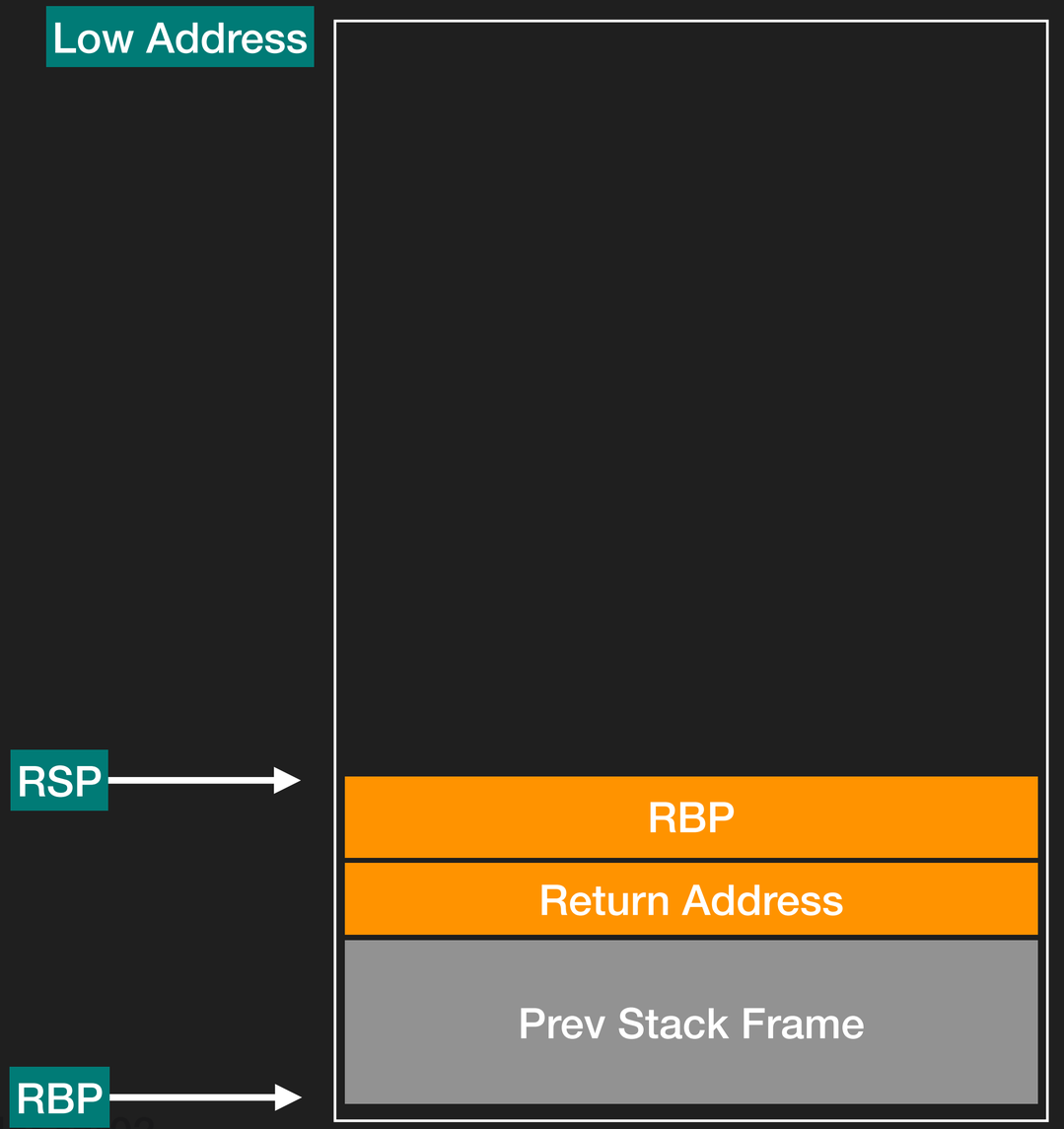
Function Prologue

```
endbr64
push    rbp
mov     rbp, rsp
sub     rsp, 30h
mov     rax, fs:28h
```



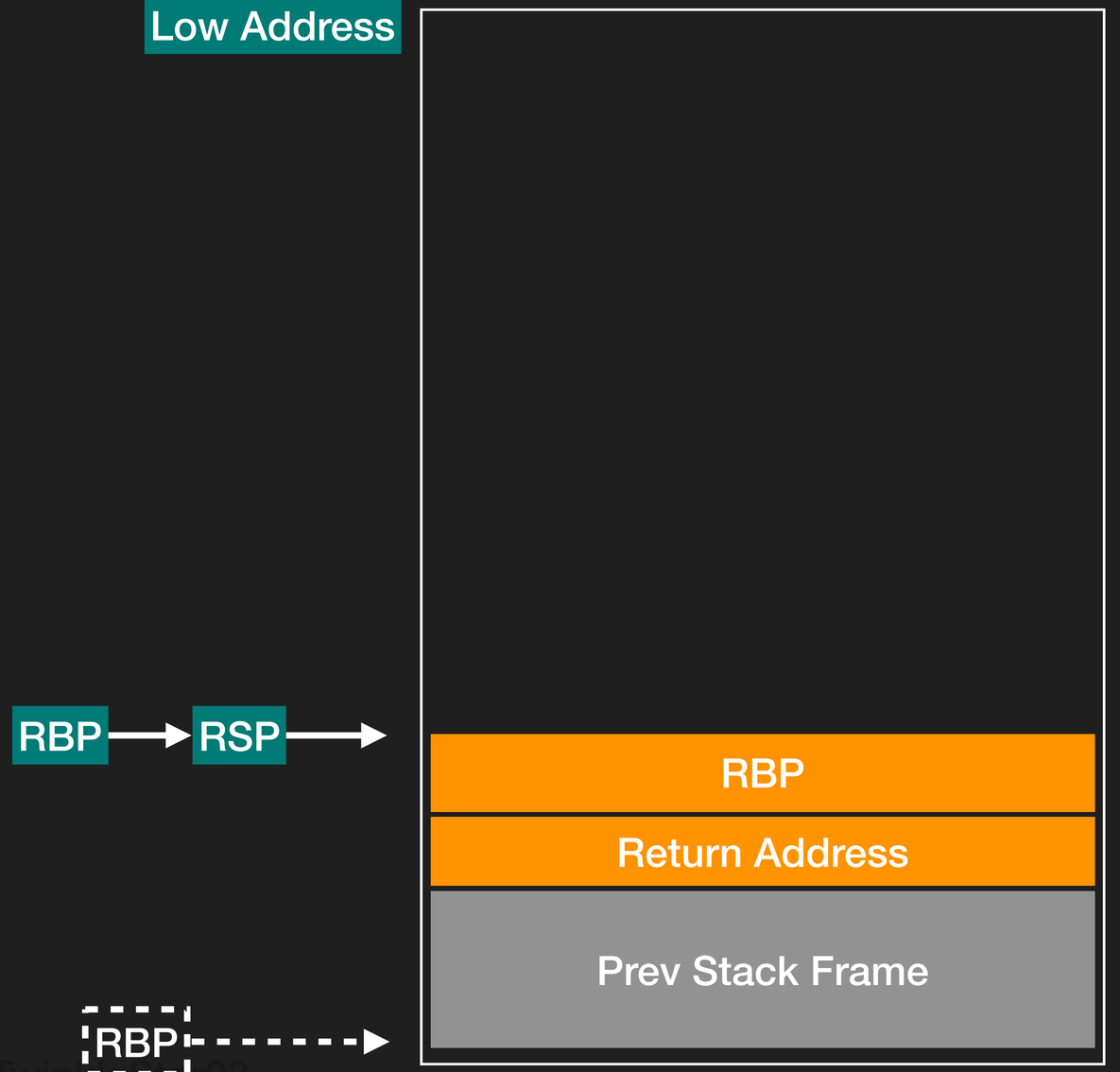
Function Prologue

```
endbr64
push    rbp
mov     rbp, rsp
sub     rsp, 30h
mov     rax, fs:28h
```



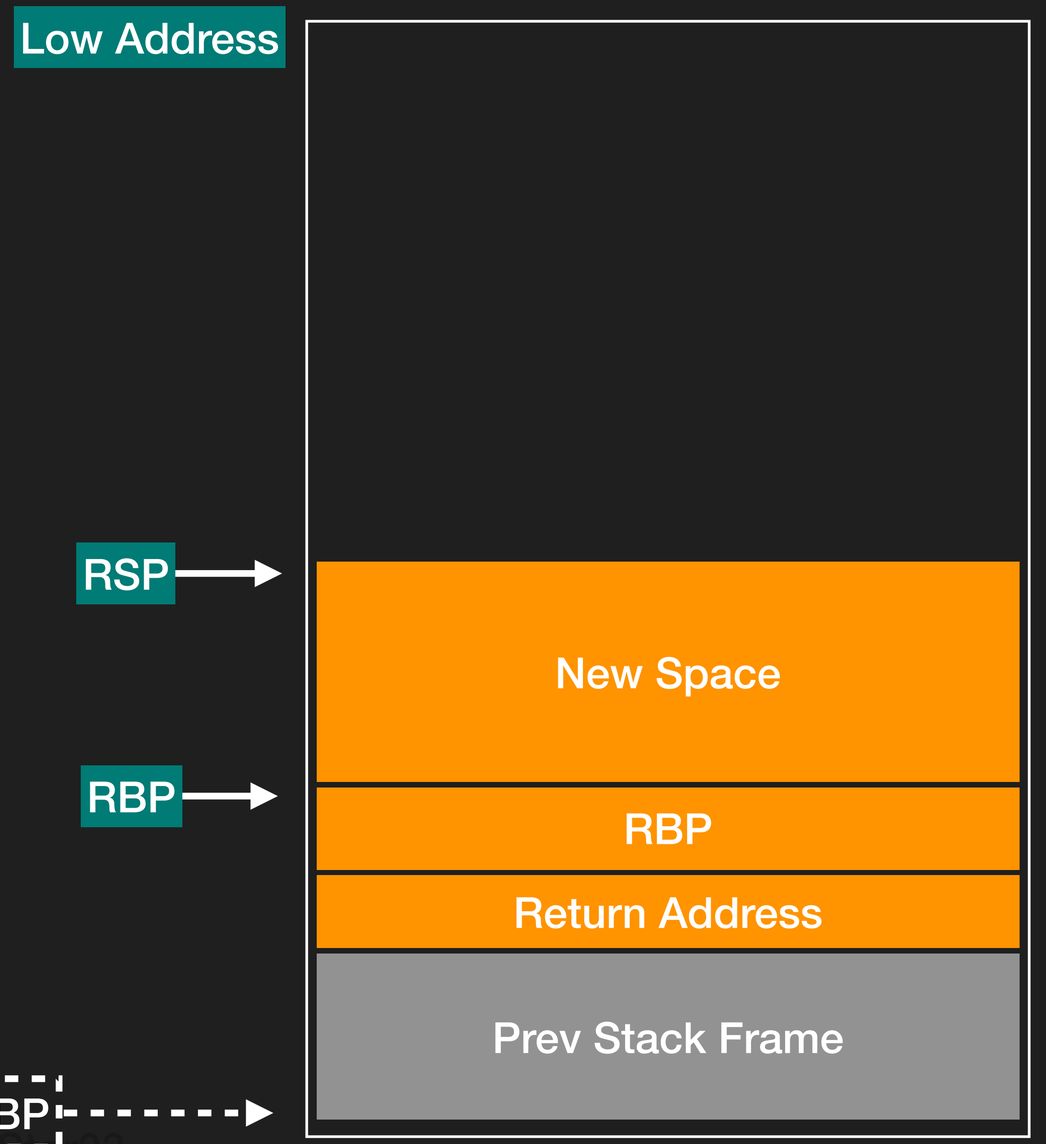
Function Prologue

```
endbr64
push    rbp
mov     rbp, rsp
sub     rsp, 30h
mov     rax, fs:28h
```



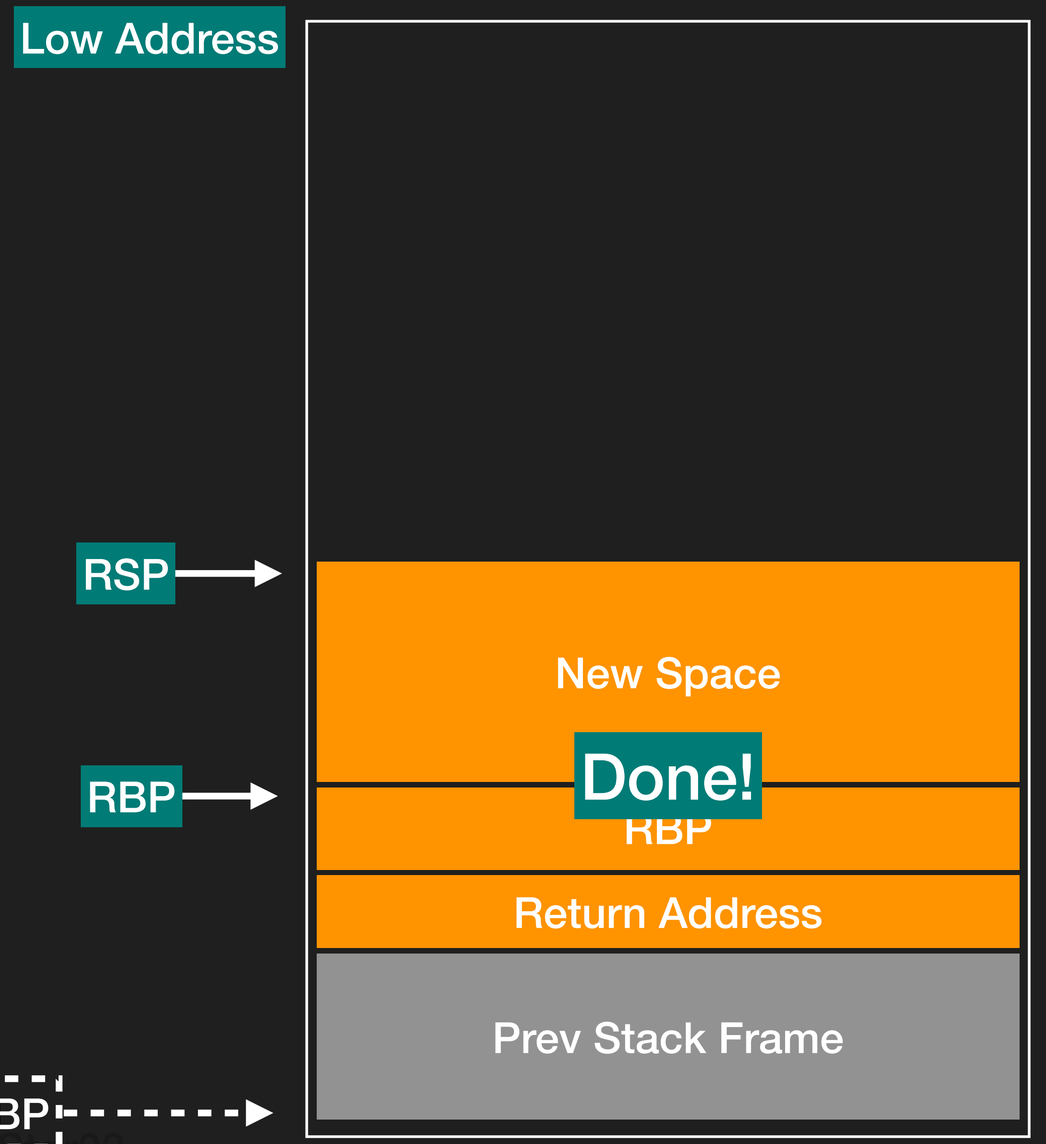
Function Prologue

```
endbr64
push    rbp
mov     rbp, rsp
sub     rsp, 30h
mov     rax, fs:28h
```



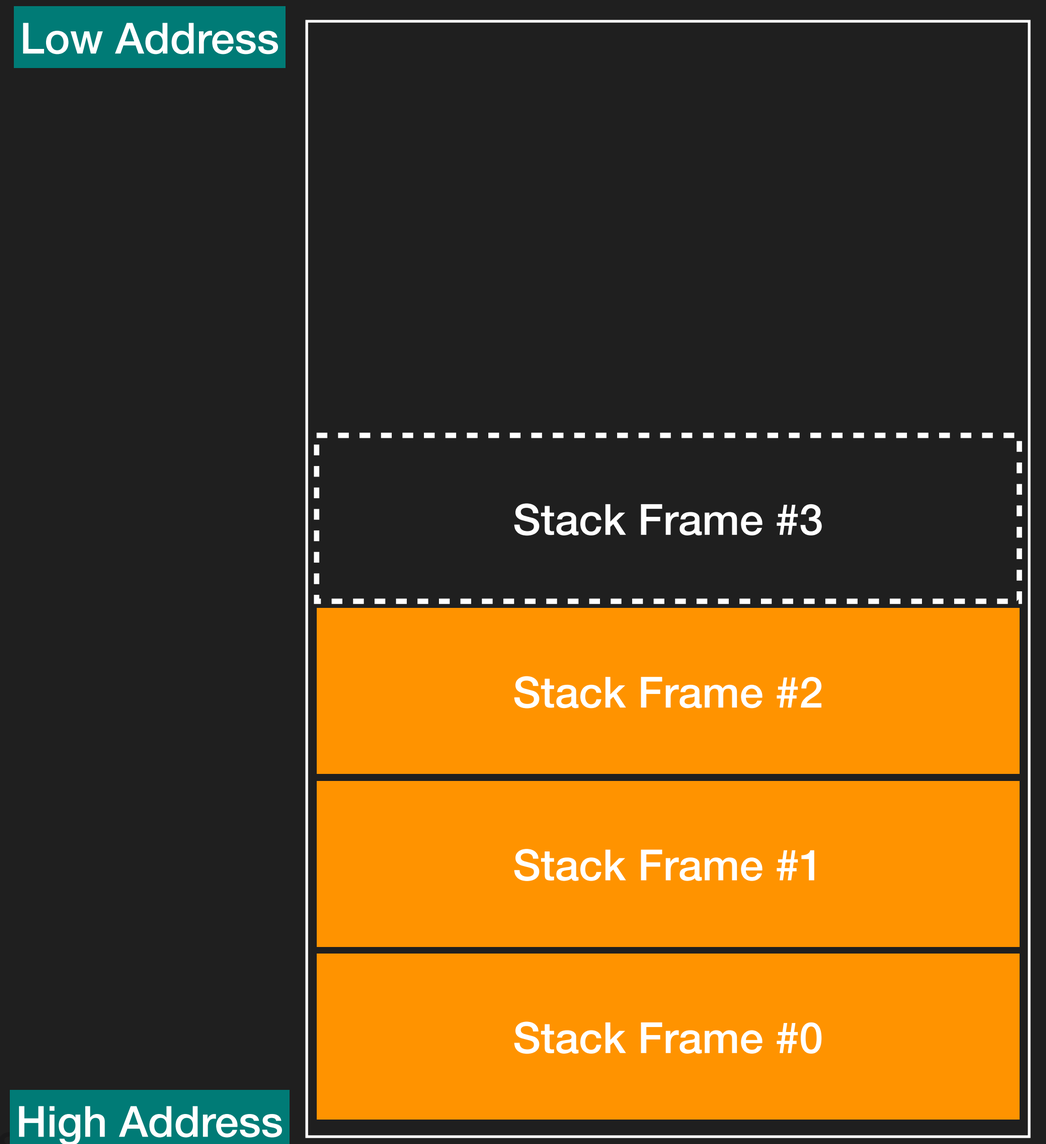
Function Prologue

```
endbr64
push    rbp
mov     rbp, rsp
sub     rsp, 30h
mov     rax, fs:28h
```



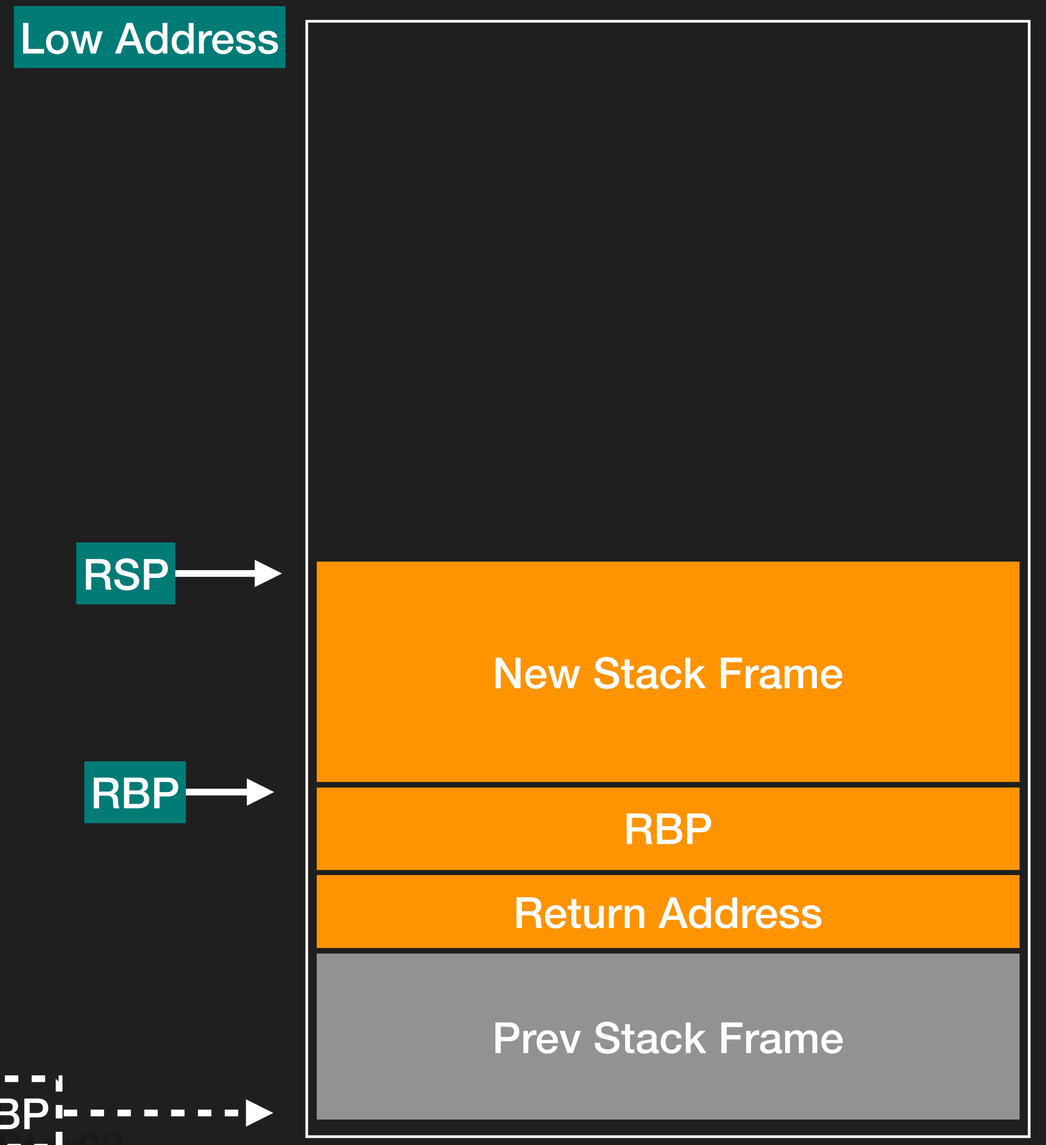
Function Epilogue

- Function 執行結束後的行為
  - 回收 Stack 空間
  - 還原前一個 Function Context



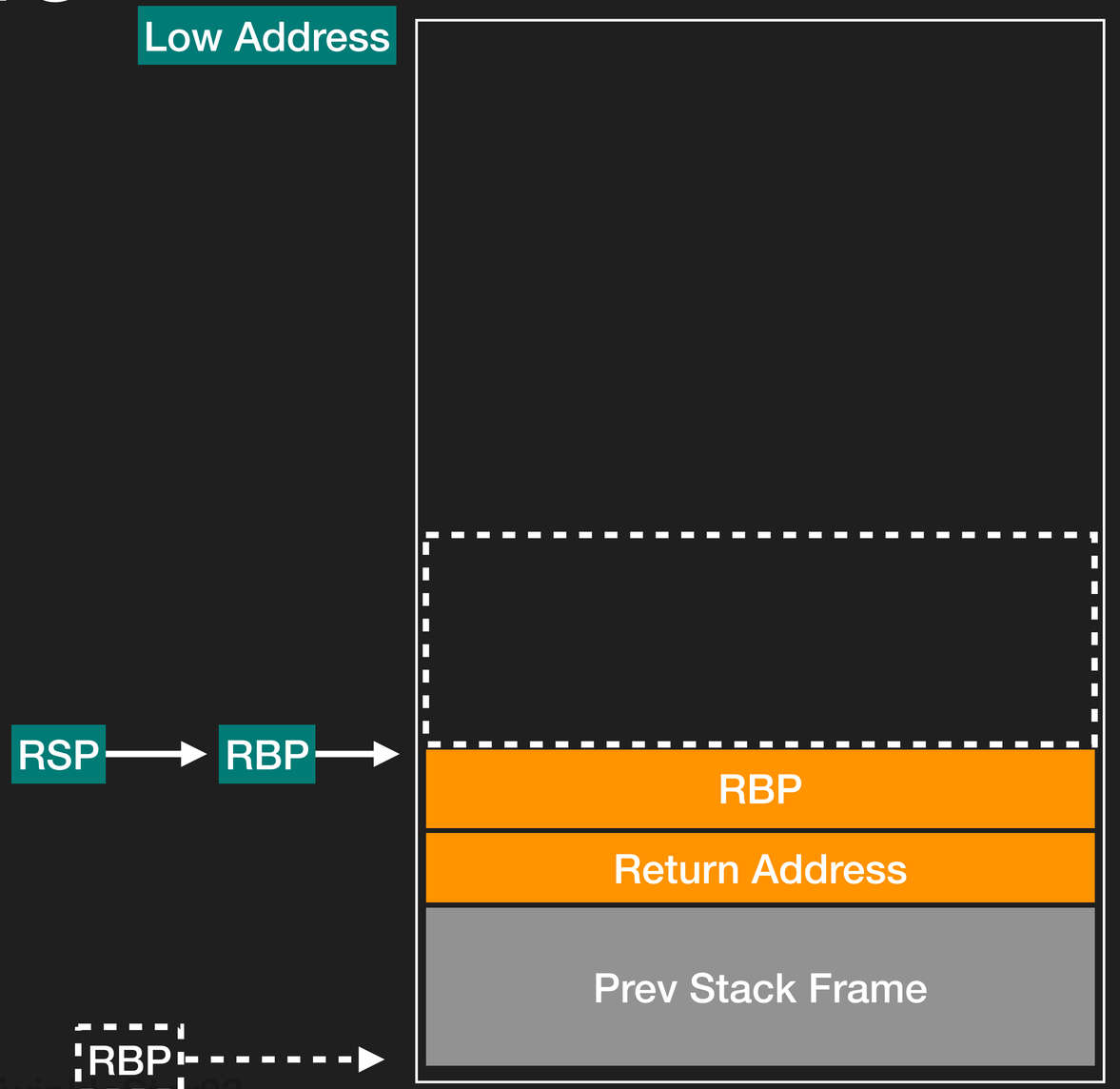
Function Prologue

```
leave    // mov $rsp, $rbp
         // pop $rbp
retn     // pop $rip
```



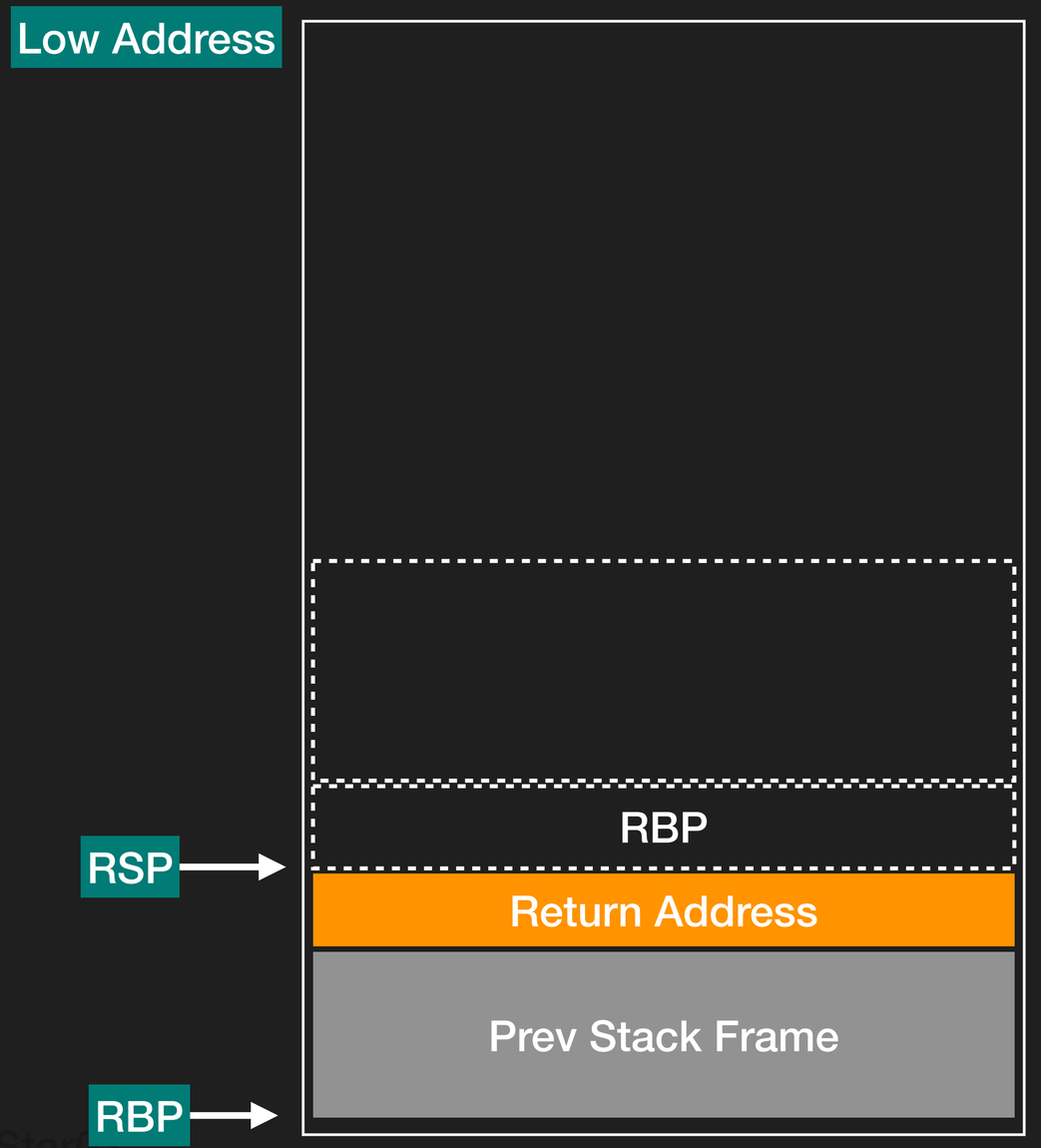
Function Prologue

```
leave    // mov $rsp, $rbp
         // pop $rbp
retn     // pop $rip
```



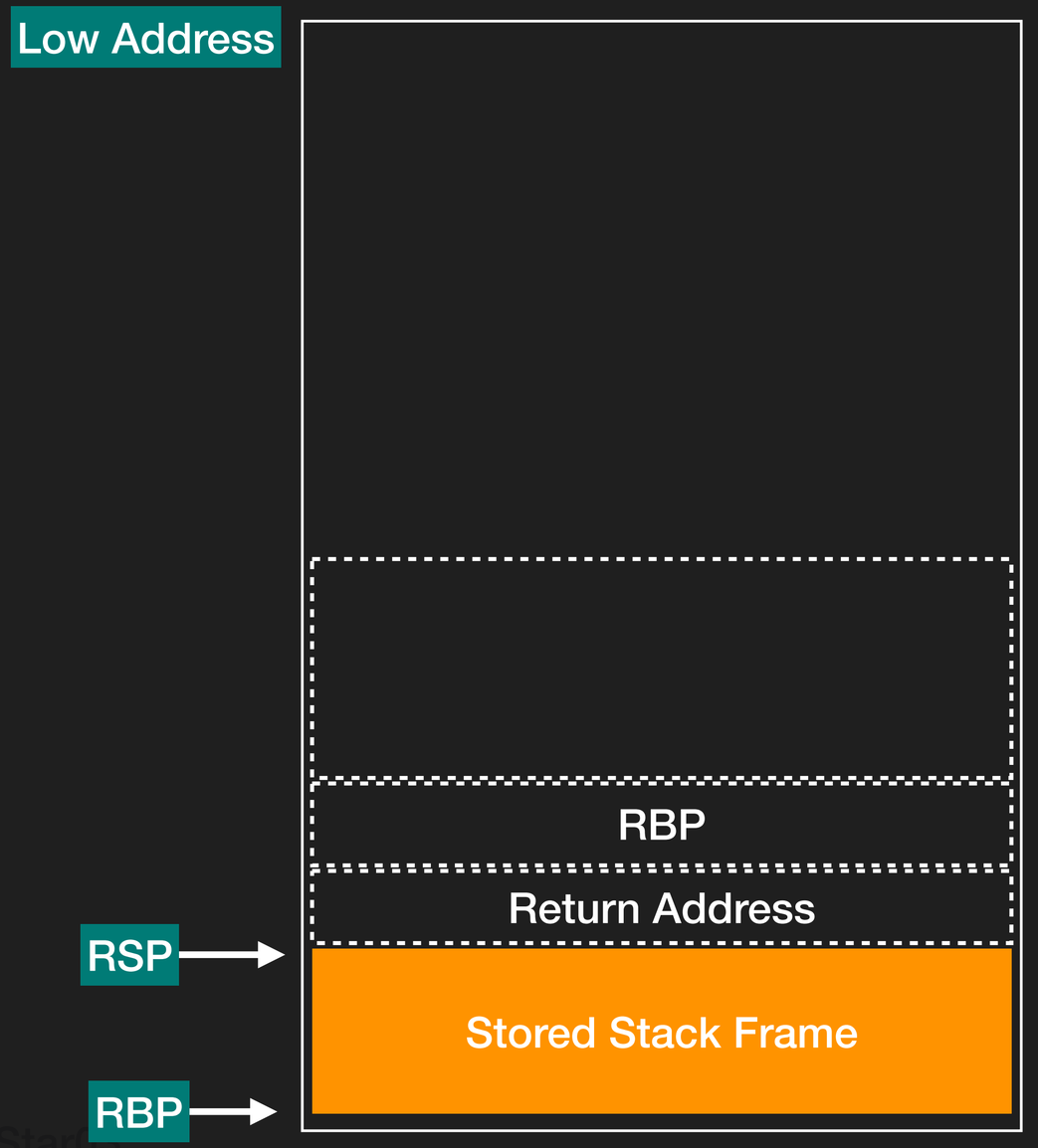
Function Prologue

```
leave    // mov $rsp, $rbp
         // pop $rbp
retn     // pop $rip
```



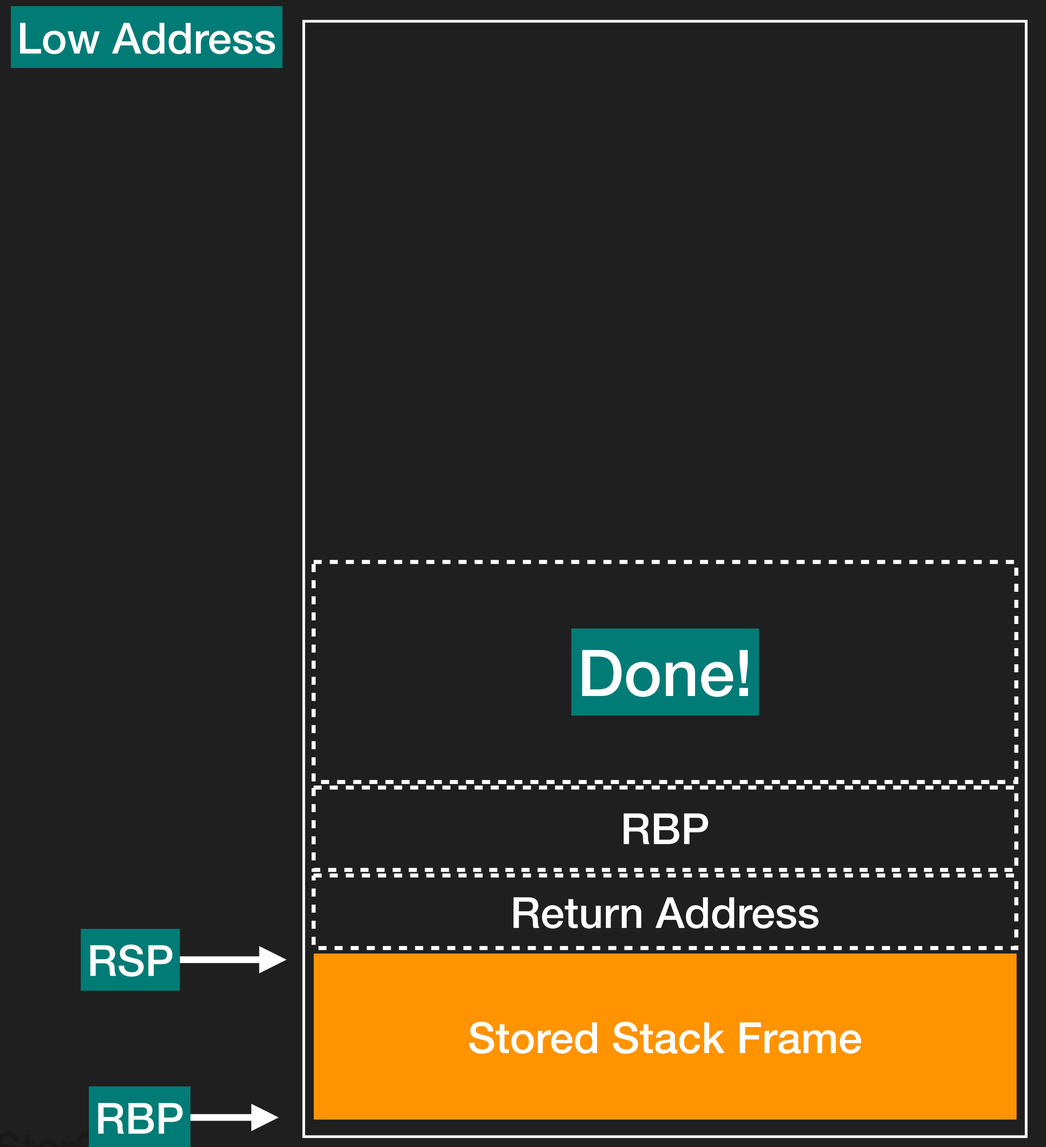
Function Prologue

```
leave   // mov $rsp, $rbp
        // pop $rbp
retn    // pop $rip
```



Function Prologue

```
leave   // mov $rsp, $rbp
        // pop $rbp
retn    // pop $rip
```



為什麼我要現在知道這個？

- 你可能會想說，現在不是 Reverse 課程嗎？為啥要講這個？
- 因為這樣你才知道 Debugger 可以調整 RIP 到哪
  - Stack 歪掉可能就不是正常運作的行為
  - Local Variable 歪掉、Stack Alignment 跑掉

Memory Layout

Memory Layout

- 程式執行的時候，使用的資料是如何長出來的?
- int, string, pointer, struct 各自尺寸大不同，長在記憶體裡面的規則呢?

Struct Alignment

- Struct 可以用 Attribute(Pack) 來設定整體對齊，預設是 8
- Struct 中的 Member 的 Alignment 取 min(sizeof(member), pack)
- Struct 最後的 Padding 取 min(sizeof(最大的成員), pack)
- Nested Struct 以該 struct 的 Pack 當 Alignment
  - 不是用整體結構大小

Struct Recovery

- 如何從 Offset 回推回去 Source Code 勒
  - IDA Auto Create Struct
  - 人工自己看

Struct Recovery

- Local Type
  - 用寫 C-Struct 的方式定義出結構
- Structure
  - 用 IDA 的功能來定義出結構

DEMO

Struct Alignment

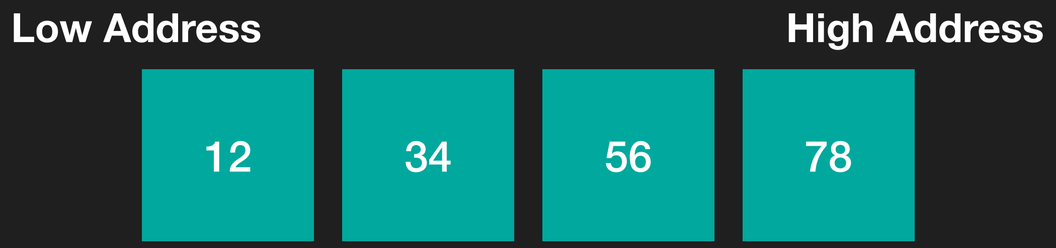
Endian

Endian

- Big-Endian
  - 大端序
- Little-Endian
  - 小端序
- 從左讀到右邊，是 MSB 還是 LSB 決定了端序

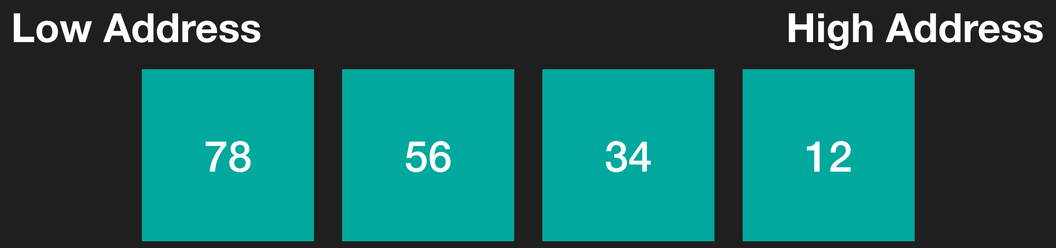
Endian

- Big Endian 第一個是 MSB
  - Human Readable (不用當人體 Parser)
- 0x12345678



Endian

- Little Endian 第一個是 LSB
- 轉型很方便 (e.g. int to short)
- 0x12345678



Compiler Optimization

Tail Call

- 當一個函數最後的 Return 會是來自另一個函數
- Example from Wikipedia
-

```
function foo(data1, data2)
    B(data1)
    return A(data2)
```

Tail Call

```
foo:
   mov  reg,[sp+data1] ; fetch data1 from stack (sp) parameter into a scratch register.
   push reg            ; put data1 on stack where B expects it
   call B              ; B uses data1
   pop                 ; remove data1 from stack
   mov  reg,[sp+data2] ; fetch data2 from stack (sp) parameter into a scratch register.
   push reg            ; put data2 on stack where A expects it
   call A              ; A uses data2
   pop                 ; remove data2 from stack.
   ret
```

Tail Call

```
foo:
   mov  reg,[sp+data1]
   push reg
   call B
   pop
   mov  reg,[sp+data2]
   push reg
   call A
   pop
   ret
```

```
foo:
   mov  reg,[sp+data1] ; fetch data1 from stack (sp) parameter into a scratch register.
   push reg            ; put data1 on stack where B expects it
   call B              ; B uses data1
   pop                 ; remove data1 from stack
   mov  reg,[sp+data2] ; fetch data2 from stack (sp) parameter into a scratch register.
   mov  [sp+data1],reg ; put data2 where A expects it
   jmp  A              ; A uses data2 and returns immediately to caller.
```

Loop Unroll

- 把 Loop 展開來
  - 利用 CPU 來加速整體執行流程

Loop Unroll

```
void blah() {
    int a = 0;
    for (int i = 0; i < 100000; ++i) {
        add(i);
    }
}
```

```
void blah() {
  v0 = 0;
  do
  {
    add(v0);
    v1 = v0 + 1;
    add(v0 + 1);
    add(v0 + 2);
    add(v0 + 3);
    add(v0 + 4);
    add(v0 + 5);
    add(v0 + 6);
    result = add(v0 + 7);
    v0 += 8;
  }
  while ( v1 != 99993 );
  return result;
}
```

Fast Division

- Division Instruction 很貴
- 通過一些簡單的數學可以把整數除法變成 Bitwise 相關的操作
  - 雖然會多一點 Instruction 但整體速度快很多

Fast Division

Division by a constant [edit]

The division by a constant *D* is equivalent to the multiplication by its reciprocal. Since the denominator is constant, so is its reciprocal (1/*D*). Thus it is possible to compute the value of (1/*D*) once at compile time, and at run time perform the multiplication *N*·(1/*D*) rather than the division *N*/*D*. In floating-point arithmetic the use of (1/*D*) presents little problem,[a] but in integer arithmetic the reciprocal will always evaluate to zero (assuming |*D*| > 1).

It is not necessary to use specifically (1/*D*); any value (*X*/*Y*) that reduces to (1/*D*) may be used. For example, for division by 3, the factors 1/3, 2/6, 3/9, or 194/582 could be used. Consequently, if *Y* were a power of two the division step would reduce to a fast right bit shift. The effect of calculating *N*/*D* as (*N*·*X*)/*Y* replaces a division with a multiply and a shift. Note that the parentheses are important, as *N*·(*X*/*Y*) will evaluate to zero.

However, unless *D* itself is a power of two, there is no *X* and *Y* that satisfies the conditions above. Fortunately, (*N*·*X*)/*Y* gives exactly the same result as *N*/*D* in integer arithmetic even when (*X*/*Y*) is not exactly equal to 1/*D*, but "close enough" that the error introduced by the approximation is in the bits that are discarded by the shift operation.[21][22][23] Barrett reduction uses powers of 2 for the value of *Y* to make division by *Y* a simple right shift.[b]

As a concrete fixed-point arithmetic example, for 32-bit unsigned integers, division by 3 can be replaced with a multiply by $\frac{2863311531}{2^{33}}$, a multiplication by 2863311531 (hexadecimal 0xAAAAAAAB) followed by a 33 right bit shift. The value of 2863311531 is calculated as $\frac{2^{33}}{3}$, then rounded up. Likewise, division by 10 can be expressed as a multiplication by 3435973837 (0xCCCCCCCD) followed by division by 2^{35} (or 35 right bit shift).[25]:p230-234 OEIS provides sequences of the constants for multiplication as A346495 and for the right shift as A346496.

For general x-bit unsigned integer division where the divisor D is not a power of 2, the following identity converts the division into two x-bit addition/subtraction, one x-bit by x-bit multiplication (where only the upper half of the result is used) and several shifts, after precomputing $k = x + \lceil \log_2 D \rceil$ and $a = \left\lceil \frac{2^k}{D} \right\rceil - 2^x$:

$$\left\lfloor \frac{N}{D} \right\rfloor = \left\lfloor \frac{\left\lfloor \frac{N-b}{2} \right\rfloor + b}{2^{k-x-1}} \right\rfloor \quad \text{where } b = \left\lfloor \frac{Na}{2^x} \right\rfloor$$

In some cases, division by a constant can be accomplished in even less time by converting the "multiply by a constant" into a series of shifts and adds or subtracts.[26] Of particular interest is division by 10, for which the exact quotient is obtained, with remainder if required.[27]

Fast Division

- 舉個例子，在 32-bit 底下的整數除法，除以五

```
# x / 5 = (x / 2**33) * (2**33 / 5)
magic = 0x66666667 # math.ceil(2**33 / 5)
x = 90

result = (x * magic) >> 33
# 90
```

Vectorization

- Advanced Vector Extensions (AVX)
  - AVX2, AVX512, AVX10 (Super New)
- 前身是 Streaming SIMD Extensions (SSE)
- SIMD
  - Single Instruction Multiple Data

Vectorization

- 提供了更大的 Register
  - XMM, YMM, ZMM
- 提供了更多的 Instruction
  - Broadcast
  - Shuffle

Vectorization

- TL;DR
  - 可以一次移動、運算大量的資料
  - 整個 Control-Flow 會變得很雜
- 你說了這麼多，對大家有啥影響嗎？
  - 變醜
  - 就連 IDAPro 最新版都還沒辦法解各種 Intrinsic

Getting Foothold

Getting Foothold

- 下斷點在 API Call
  - 猜程式行為然後下斷點在可能會走到的地方
- 找重要參數
  - Magic Bytes
  - 固定的邏輯 (加解密演算法們)

Getting Foothold

- 如果 Library 或是部分邏輯是從開源軟體 Link 進來的
  - 去找 Source Code 對著看
- 尋找 Debug Message
  - 有好好寫 Debug Message 的話可以從這邊推敲出邏輯

Getting Foothold

- 在 Function List 亂滑
  - 有時候就剛好被你看到重要邏輯了
  - 有工具可以幫你做到這件事情

Gettin

- 在 Function Lis
  - 有時候就剛好
  - 有工具可以幫

Q & A

關於下週 (Week2)

- Windows Malware RE
  - 不是我教，是另一個助教 ID: Ice1187
- 請自行準備以下事務
  - Windows 10/11 VM
  - x64dbg
  - Wireshark
  - Text Editor
  - IDA Freeware
  - PE-Bear
  - Python3
  - 逆向工程的耐心


啊，今天跟人講太多話有點累
我先回家了

Thanks for attention 🥰