LOADERS & ARQUIVOS EXECUTÁVEIS

202200560061 Filipe Viotto Rodrigues
202200560356 Isadora Vanço

O QUE É UM ARQUIVO EXECUTÁVEL?

O QUE É?

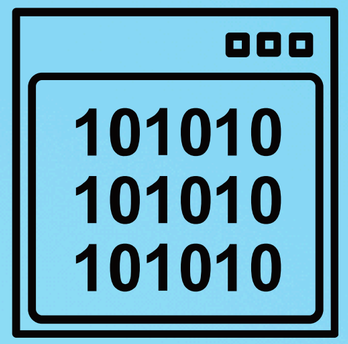
EXTENSÕES MAIS UTILIZADAS

- exe
- bat
- com
- cmd

- out
- run
- bin
- appimage

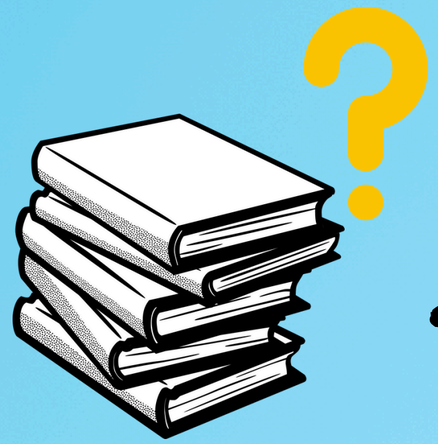
O QUE TEM DENTRO DE UM ARQUIVO EXECUTÁVEL?

O QUE TEM DENTRO?

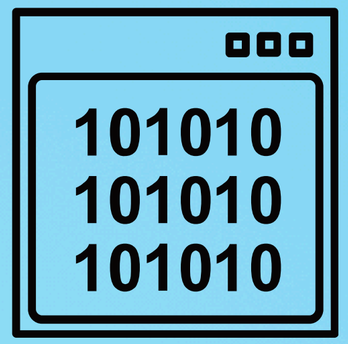


Instruções de máquina

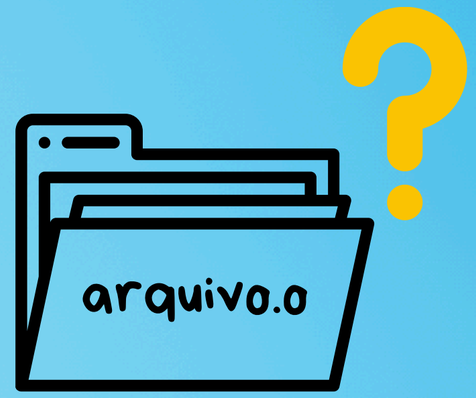
O QUE TEM DENTRO?



Biblioteca dinâmica

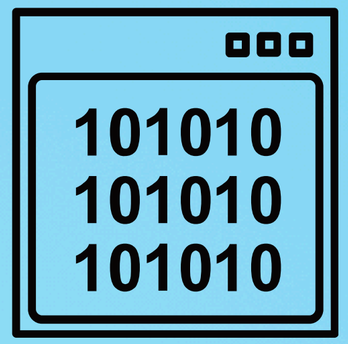


Instruções de máquina



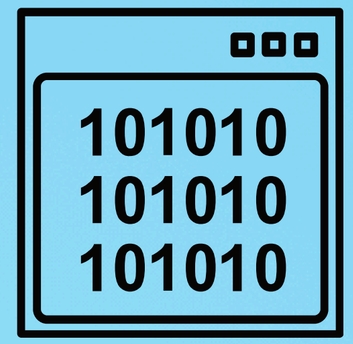
Arquivo objeto

O QUE TEM DENTRO?



Instruções de máquina

O QUE TEM DENTRO?



Instruções de máquina

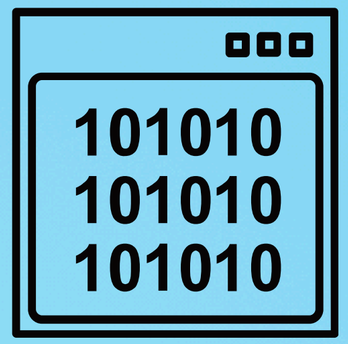
Chamadas de sistema

API do Win32

Posix

O QUE TEM DENTRO?

Metadados do programa



Instruções de máquina

Chamadas de sistema

API do Win32

Posix

QUAIS SÃO OS FORMATOS DE ARQUIVOS EXECUTÁVEIS?

FORMATOS DE ARQUIVOS EXECUTÁVEIS

- PE
- ELF
- MACH-O
- A.OUT
- WASM
- PEF
- MZ/DOS EXE
- COFF
- NE

OUTROS...

Portable Executable

Executable and Linkable Format

Common Object File Format

⚠ Nem todos os executáveis precisam de um formato → Bootloader

FORMATOS DE ARQUIVOS EXECUTÁVEIS

Portable Executable

- Introduzido no Windows NT (1993)
- Substituir o NE
- Derivado do COFF
- Flexível e extensível
- Ambiente UEFI

Common Object File Format

- Introduzido no UNIX System V (1983)
- Substituir o a.out
- Ainda utilizado em sistemas embarcados
- Arquivo de objeto portátil
- Estruturas sobre seções do programa

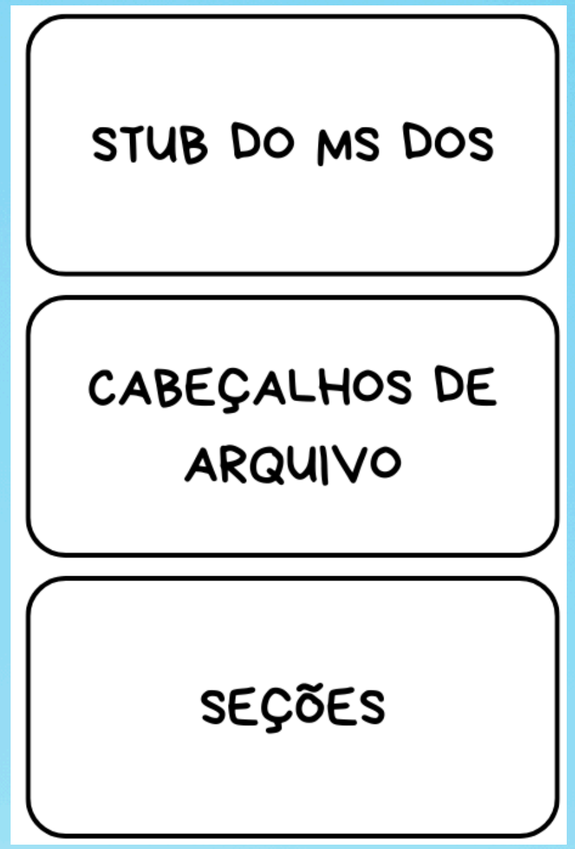
Executable and Linkable Format

- Introduzido no UNIX System V (1989)
- substituir o COFF
- Flexível, extensível e multiplataforma
- Padrão para arquivos binários em Unix

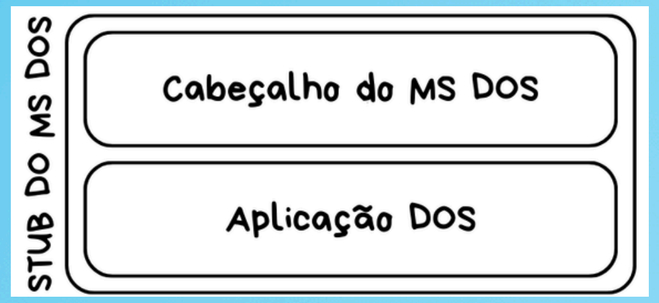
A ESTRUTURA DO FORMATO PE

PORTABLE EXECUTABLE (PE)

.exe, .dll, .sys, ...



PORTABLE EXECUTABLE (PE)



"This program cannot be run in DOS mode"

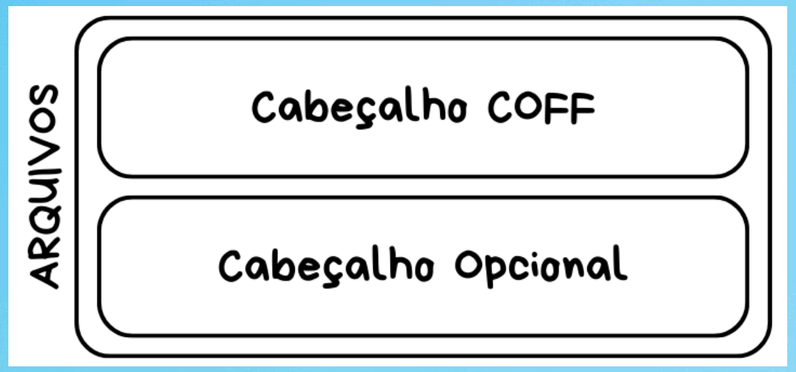
```
typedef struct {
    uint16_t e_magic;
    uint16_t e_cblp;
    uint16_t e_cp;
    uint16_t e_crlc;
    uint16_t e_cparhdr;
    uint16_t e_minalloc;
    uint16_t e_maxalloc;
    uint16_t e_ss;
    uint16_t e_sp;
    uint16_t e_csum;
    uint16_t e_ip;
    uint16_t e_cs;
    uint16_t e_lfarlc;
    uint16_t e_ovno;
    uint16_t e_res[4];
    uint16_t e_oemid;
    uint16_t e_oeminfo;
    uint16_t e_res2[10];
    uint32_t e_lfanew;
} IMAGE_DOS_HEADER;
```

e_magic: MZ
(Mark Zbikowski)

e_lfanew: Assinatura PE
"PE\0\0"

winnt.h

PORTABLE EXECUTABLE (PE)




```
typedef struct {
    uint16_t Machine;
    uint16_t NumberOfSections;
    uint32_t TimeDateStamp;
    uint32_t PointerToSymbolTable;
    uint32_t NumberOfSymbols;
    uint16_t SizeOfOptionalHeader;
    uint16_t Characteristics;
} IMAGE_FILE_HEADER,
IMAGE_COFF_HEADER;
```

SizeOfOptionalHeader: tamanho do Cabeçalho Opcional

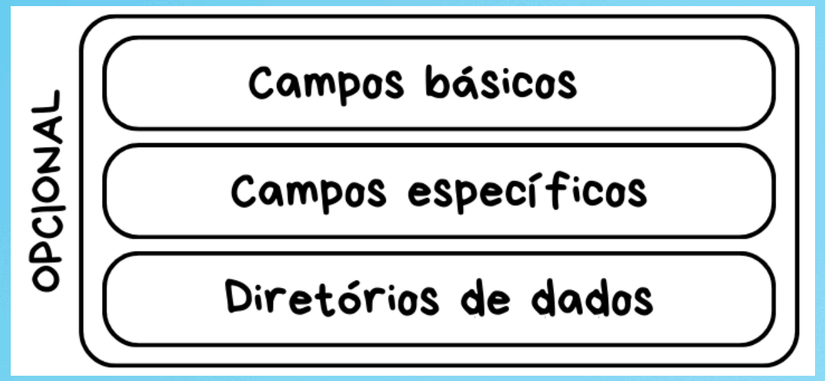
Characteristics: atributos usando máscara de bits

TimeDataStamp: data de criação

NumberOfSections: número de seções

Machine: arquitetura

PORTABLE EXECUTABLE (PE)

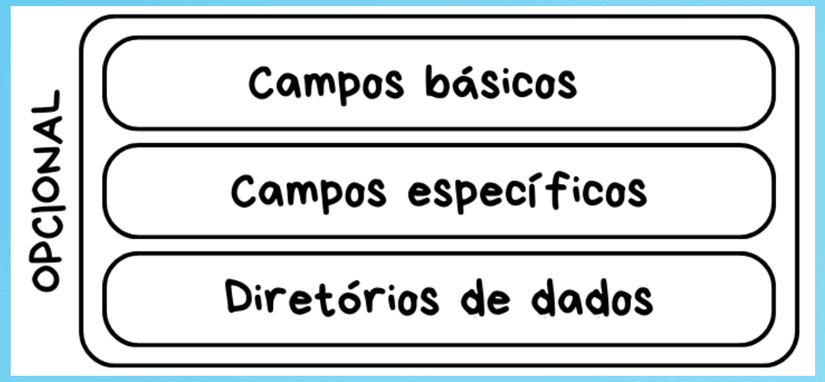


AddressOfEntryPoint: endereço de inicio do programa

Magic: número mágico que representa 32/64bits

```
typedef struct {
    uint16_t Magic;
    uint8_t MajorLinkerVersion;
    uint8_t MinorLinkerVersion;
    uint32_t SizeOfCode;
    uint32_t SizeOfInitializedData;
    uint32_t SizeOfUninitializedData;
    uint32_t AddressOfEntryPoint;
    uint32_t BaseOfCode;
    uint32_t BaseOfData;
    uint32_t ImageBase;
    uint32_t SectionAlignment;
    uint32_t FileAlignment;
    uint16_t MajorOperatingSystemVersion;
    uint16_t MinorOperatingSystemVersion;
    uint16_t MajorImageVersion;
    uint16_t MinorImageVersion;
    uint16_t MajorSubsystemVersion;
    uint16_t MinorSubsystemVersion;
    uint32_t Reserved1;
    uint32_t SizeOfImage;
    uint32_t SizeOfHeaders;
    uint32_t CheckSum;
    uint16_t Subsystem;
    uint16_t DllCharacteristics;
    uint32_t SizeOfStackReserve;
    uint32_t SizeOfStackCommit;
    uint32_t SizeOfHeapReserve;
    uint32_t SizeOfHeapCommit;
    uint32_t LoaderFlags;
    uint32_t NumberOfRvaAndSizes;
    IMAGE_DATA_DIRECTORY DataDirectory[16];
} IMAGE_OPTIONAL_HEADER_32;
```

PORTABLE EXECUTABLE (PE)



ImageBase: endereço base do arquivo na memória

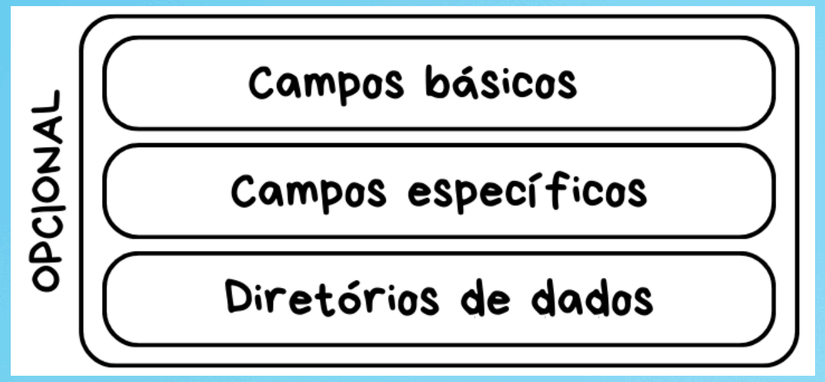
Por padrão:

- executáveis: 0x400000
- dlls: 0x10000000

SectionAlignment: fator de alinhamento para as seções mapeadas

```
typedef struct {
    uint16_t Magic;
    uint8_t MajorLinkerVersion;
    uint8_t MinorLinkerVersion;
    uint32_t SizeOfCode;
    uint32_t SizeOfInitializedData;
    uint32_t SizeOfUninitializedData;
    uint32_t AddressOfEntryPoint;
    uint32_t BaseOfCode;
    uint32_t BaseOfData;
    uint32_t ImageBase;
    uint32_t SectionAlignment;
    uint32_t FileAlignment;
    uint16_t MajorOperatingSystemVersion;
    uint16_t MinorOperatingSystemVersion;
    uint16_t MajorImageVersion;
    uint16_t MinorImageVersion;
    uint16_t MajorSubsystemVersion;
    uint16_t MinorSubsystemVersion;
    uint32_t Reserved1;
    uint32_t SizeOfImage;
    uint32_t SizeOfHeaders;
    uint32_t CheckSum;
    uint16_t Subsystem;
    uint16_t DllCharacteristics;
    uint32_t SizeOfStackReserve;
    uint32_t SizeOfStackCommit;
    uint32_t SizeOfHeapReserve;
    uint32_t SizeOfHeapCommit;
    uint32_t LoaderFlags;
    uint32_t NumberOfRvaAndSizes;
    IMAGE_DATA_DIRECTORY DataDirectory[16];
} IMAGE_OPTIONAL_HEADER_32;
```

PORTABLE EXECUTABLE (PE)



SubSystem: tipo de subsistema necessário para rodar o app

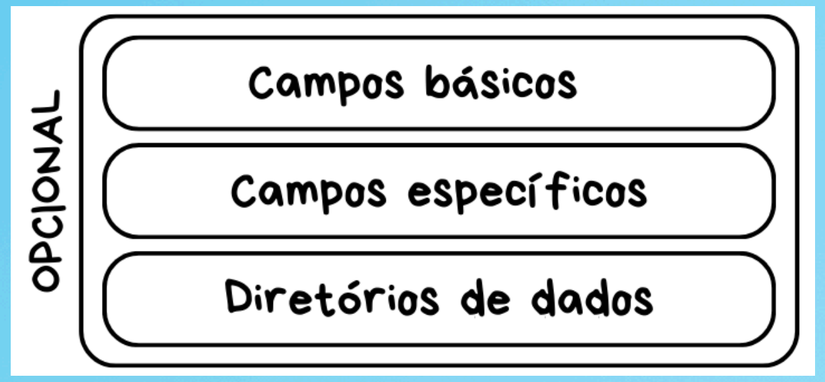
Exemplo:

- GUI: 0x002
- CUI: 0x003

DllCharacteristics: descreve caracterísitcas do arquivo usando máscara de bits

```
typedef struct {
    uint16_t Magic;
    uint8_t MajorLinkerVersion;
    uint8_t MinorLinkerVersion;
    uint32_t SizeOfCode;
    uint32_t SizeOfInitializedData;
    uint32_t SizeOfUninitializedData;
    uint32_t AddressOfEntryPoint;
    uint32_t BaseOfCode;
    uint32_t BaseOfData;
    uint32_t ImageBase;
    uint32_t SectionAlignment;
    uint32_t FileAlignment;
    uint16_t MajorOperatingSystemVersion;
    uint16_t MinorOperatingSystemVersion;
    uint16_t MajorImageVersion;
    uint16_t MinorImageVersion;
    uint16_t MajorSubsystemVersion;
    uint16_t MinorSubsystemVersion;
    uint32_t Reserved1;
    uint32_t SizeOfImage;
    uint32_t SizeOfHeaders;
    uint32_t CheckSum;
    uint16_t Subsystem;
    uint16_t DllCharacteristics;
    uint32_t SizeOfStackReserve;
    uint32_t SizeOfStackCommit;
    uint32_t SizeOfHeapReserve;
    uint32_t SizeOfHeapCommit;
    uint32_t LoaderFlags;
    uint32_t NumberOfRvaAndSizes;
    IMAGE_DATA_DIRECTORY DataDirectory[16];
} IMAGE_OPTIONAL_HEADER_32;
```

PORTABLE EXECUTABLE (PE)



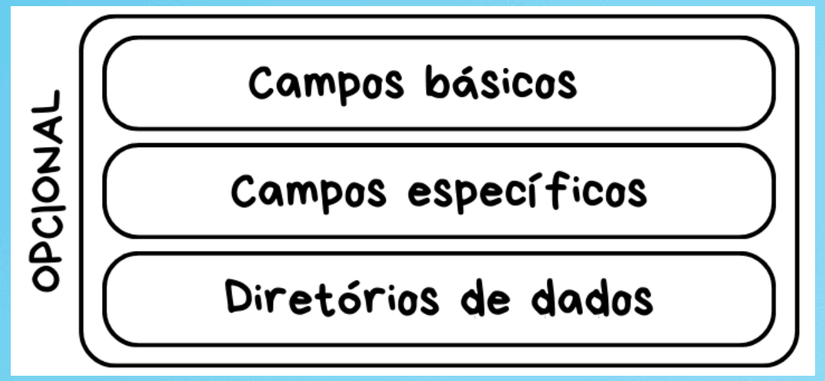
DataDirectory:
diretórios de dados

16 no total

```
typedef struct {
    uint32_t   VirtualAddress;
    uint32_t   Size;
} IMAGE_DATA_DIRECTORY;
```

```
typedef struct {
    uint16_t Magic;
    uint8_t MajorLinkerVersion;
    uint8_t MinorLinkerVersion;
    uint32_t SizeOfCode;
    uint32_t SizeOfInitializedData;
    uint32_t SizeOfUninitializedData;
    uint32_t AddressOfEntryPoint;
    uint32_t BaseOfCode;
    uint32_t BaseOfData;
    uint32_t ImageBase;
    uint32_t SectionAlignment;
    uint32_t FileAlignment;
    uint16_t MajorOperatingSystemVersion;
    uint16_t MinorOperatingSystemVersion;
    uint16_t MajorImageVersion;
    uint16_t MinorImageVersion;
    uint16_t MajorSubsystemVersion;
    uint16_t MinorSubsystemVersion;
    uint32_t Reserved1;
    uint32_t SizeOfImage;
    uint32_t SizeOfHeaders;
    uint32_t CheckSum;
    uint16_t Subsystem;
    uint16_t DllCharacteristics;
    uint32_t SizeOfStackReserve;
    uint32_t SizeOfStackCommit;
    uint32_t SizeOfHeapReserve;
    uint32_t SizeOfHeapCommit;
    uint32_t LoaderFlags;
    uint32_t NumberOfRvaAndSizes;
    IMAGE_DATA_DIRECTORY DataDirectory[16];
} IMAGE_OPTIONAL_HEADER_32;
```

PORTABLE EXECUTABLE (PE)

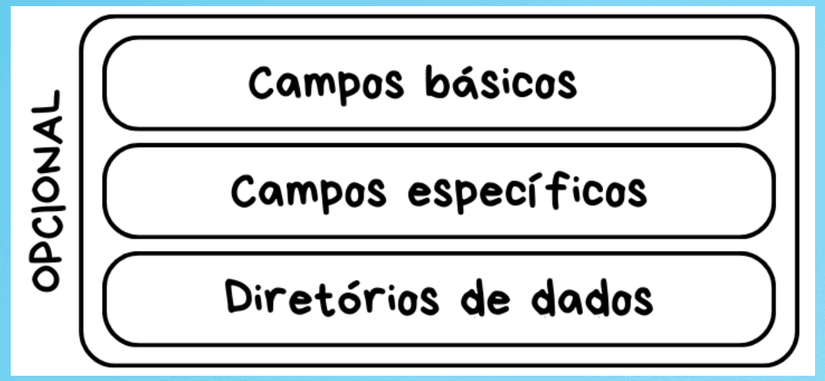



```
typedef struct {
    uint32_t   VirtualAddress;
    uint32_t   Size;
} IMAGE_DATA_DIRECTORY;
```

VirtualAddress: endereço virtual relativo da tabela

Size: tamanho da tabela em bytes

PORTABLE EXECUTABLE (PE)



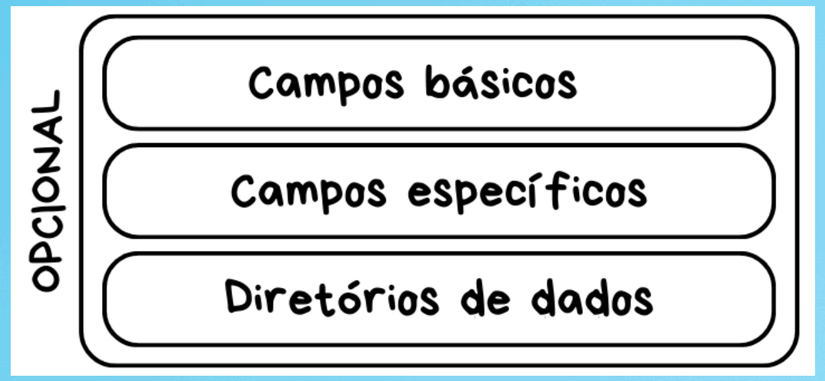

```
typedef struct {
    uint32_t   VirtualAddress;
    uint32_t   Size;
} IMAGE_DATA_DIRECTORY;
```

Export Table:
tabela de funções exportadas

Import Table:
tabela de funções importadas

Resource Table:
árvore binária com todos resources

PORTABLE EXECUTABLE (PE)




```
typedef struct {
    uint32_t   VirtualAddress;
    uint32_t   Size;
} IMAGE_DATA_DIRECTORY;
```

Export Table:
tabela de funções exportadas

Import Table:
tabela de funções importadas

Resource Table:
árvore binária com todos resources

PORTABLE EXECUTABLE (PE)

IMPORT TABLE

VA

Import Directory Table (IDT): array contendo todas as dlls, termina com valores null

```
typedef struct {
    union {
        uint32_t Characteristics;
        uint32_t OriginalFirstThunk;
    } u1;
    uint32_t TimeDateStamp;
    uint32_t ForwarderChain;
    uint32_t Name;
    uint32_t FirstThunk;
} IMAGE_IMPORT_DESCRIPTOR;
```

Import Address Table (IAT): array contendo todos os endereços reais das funções importadas

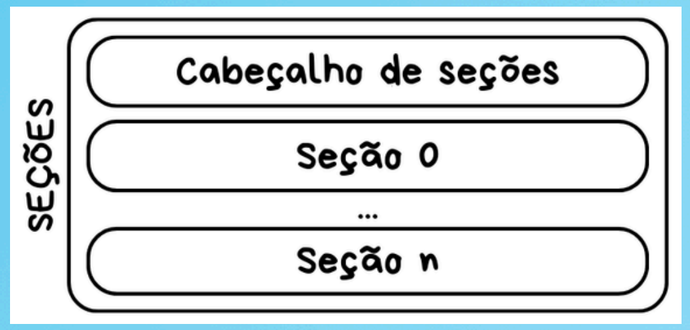
FirstThunk: endereço virtual relativo da IAT

Import Lookup Table (ILT): array contendo todos os nomes das funções importadas

OriginalFirstThunk: endereço virtual relativo da ILT

Name: nome da dll importada

PORTABLE EXECUTABLE (PE)




```
typedef struct {
    uint8_t Name[SECTION_NAME_SIZE];
    uint32_t VirtualSize;
    uint32_t VirtualAddress;
    uint32_t SizeOfRawData;
    uint32_t PointerToRawData;
    uint32_t PointerToRelocations;
    uint32_t PointerToLinenumbers;
    uint16_t NumberOfRelocations;
    uint16_t NumberOfLinenumbers;
    uint32_t Characteristics;
} IMAGE_SECTION_HEADER;
```

Characteristics: flags da seção usando máscara de bits

Exemplo:

- bit 5: código executável
- bit 29: execução
- bit 30: leitura

PointerToRawData: primeiro byte da seção

VirtualAddress: endereço relativo da base da imagem

VirtualSize: tamanho em bytes da seção depois de mapeada

Name: nome da seção

ESTRUTURA NA PRÁTICA

Vamos ver se é verdade mesmo!

APLICATIVOS DE ENGENHARIA REVERSA

Verificar o código em hexadecimal:

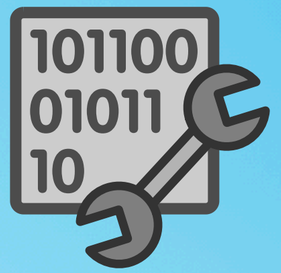
- xxd
- super hex editor
- hex dump

Verificar as seções do formato:

- readelf
- dumpbin
- die (Detect it Easy)

Disassemblers:

- IDA Freeware

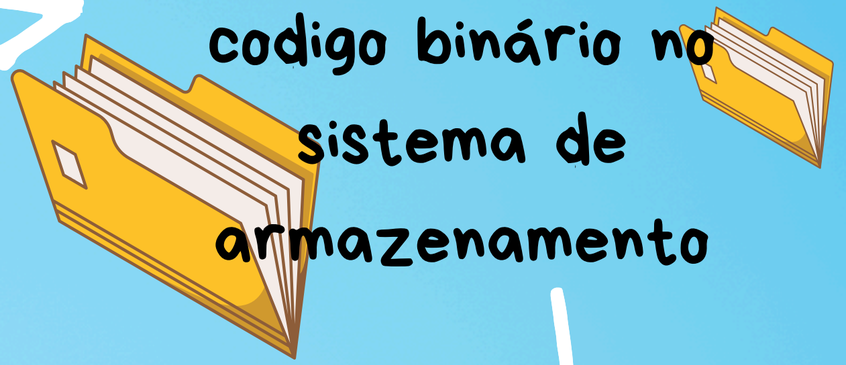


O QUE SÃO LOADERS?

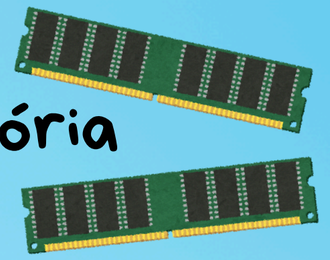
RESUMO

Evento exige a execução de um programa

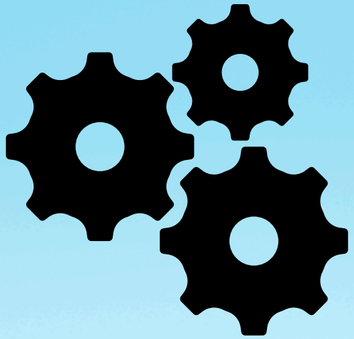
loader localiza o codigo binário no sistema de armazenamento



Aloca na memória



Inicia execução



Processo de carregamento e inicialização em forma de história

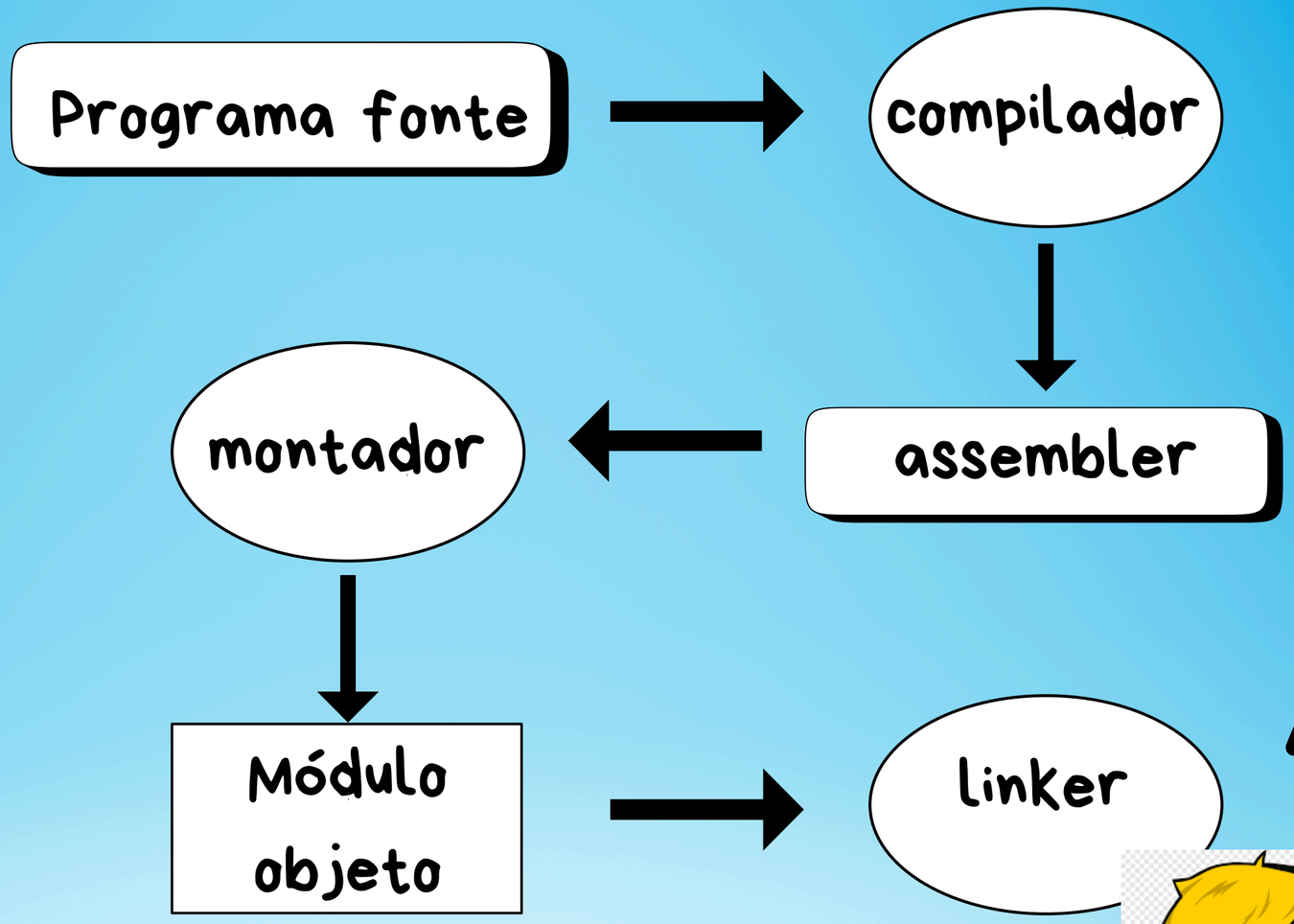
DISCO RÍGIDO

No sistema de armazenamento secundário, como um disco rígido...

SISTEMAS DE ARQUIVO

Há vário arquivos binários esperando ansiosamente para serem alocados e executados.

COMO OS EXECUTÁVEIS NASCEM?



O ARQUIVO EXECUTÁVEL

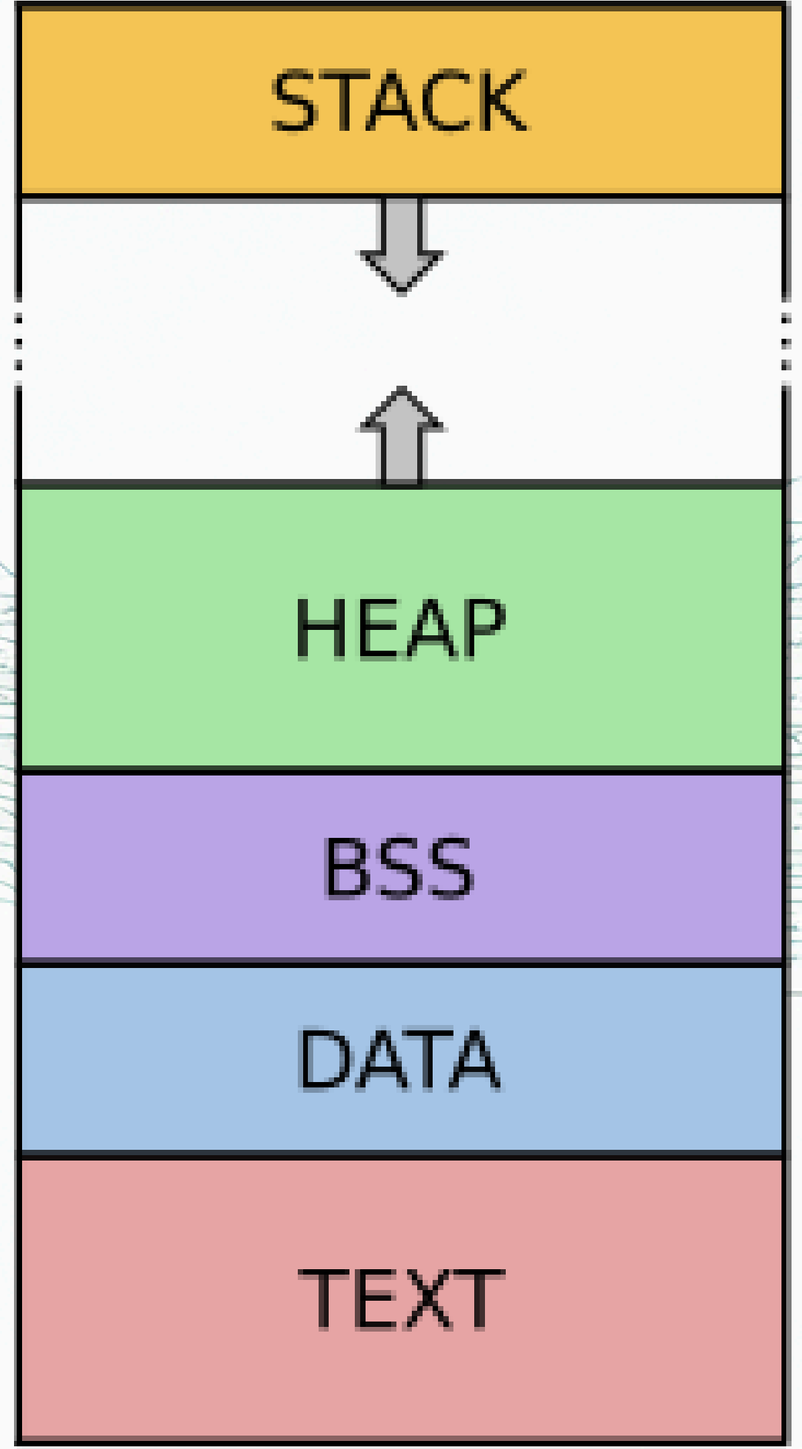
Contém mais do que código binário. O header possui informações de alocação, bibliotecas compartilhadas, etc. que são usadas pelo loader.

INICIO DA EXECUÇÃO

Quando o usuário executa um programa, o loader busca o codigo binário no sistema de armazenamento e aloca um espaço na memoria principal para carregá-lo.

ALOCAÇÃO EM MEMÓRIA

É reservado uma área para o código, área de dados para variáveis e área de pilhas de execução.

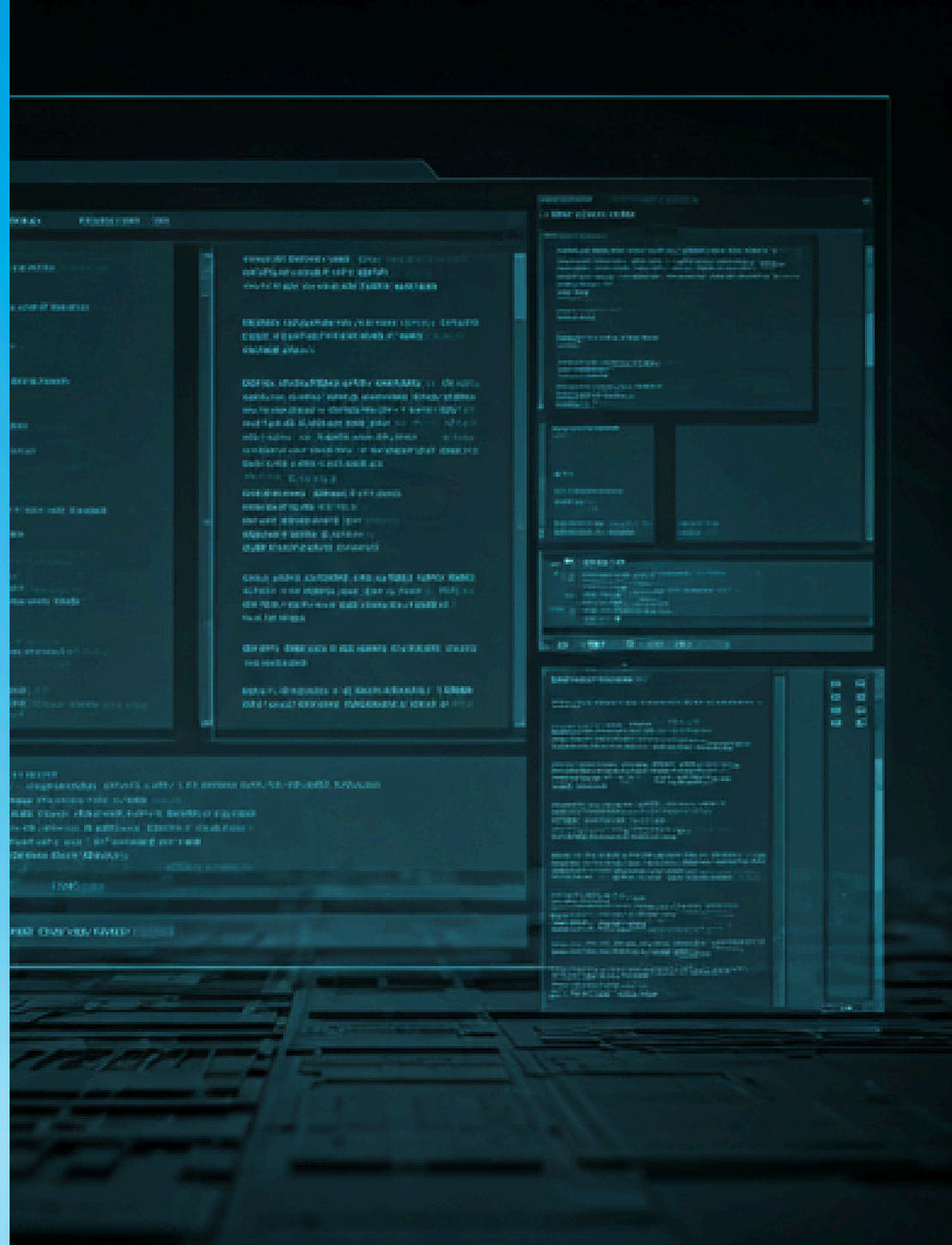


COMANDOS DO LOADER PARA O S.O.

linux: mmap

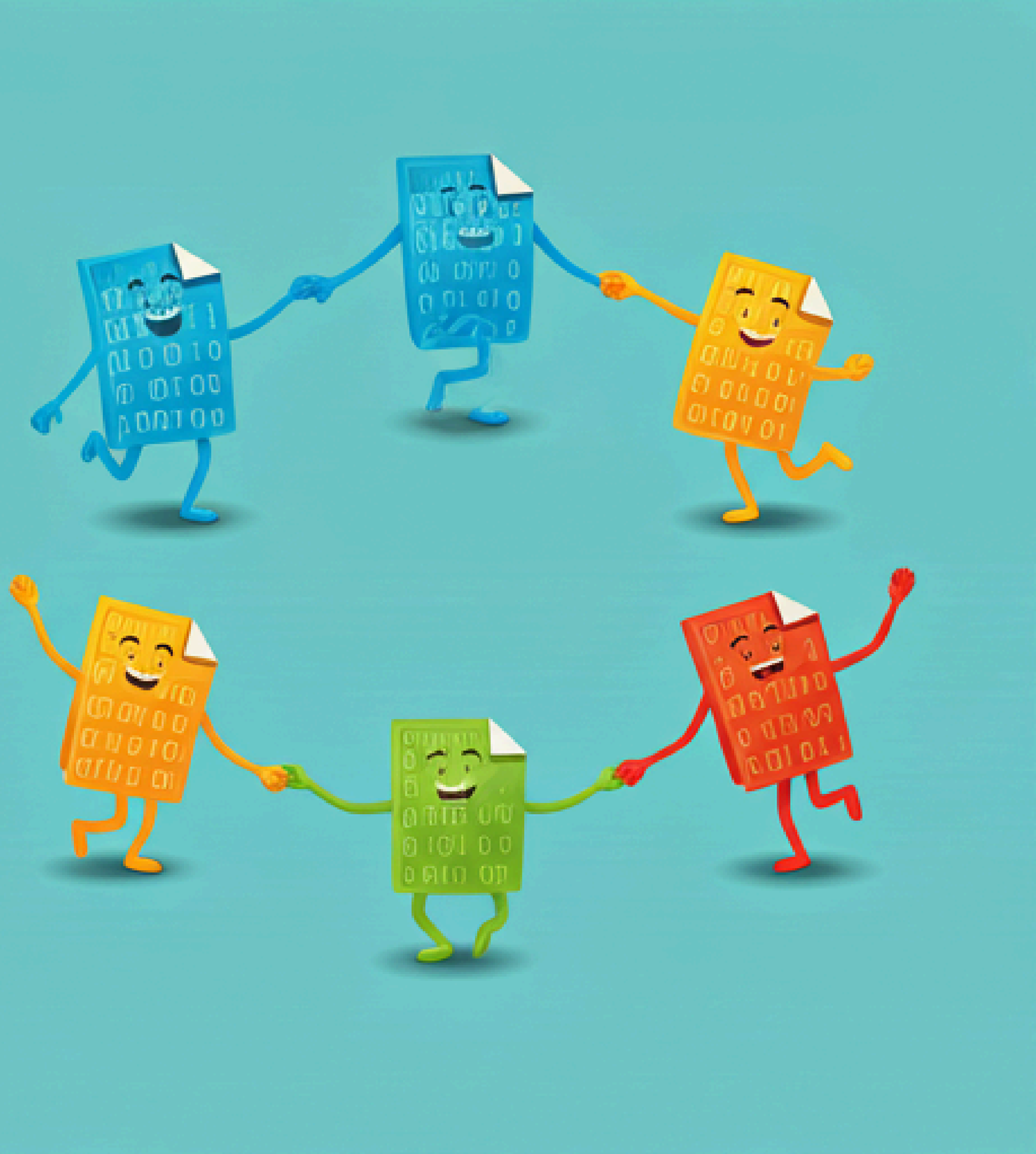
Windows: VirtualAlloc

```
LPVOID VirtualAlloc(
  [in, optional] LPVOID lpAddress,
  [in]           SIZE_T dwSize,
  [in]           DWORD  flAllocationType,
  [in]           DWORD  flProtect
);
```



RESOLUÇÃO DE DEPENDÊNCIAS

Os programas podem não gostar de ficar sozinhos e precisar de bibliotecas compartilhadas, como .dll e .so, então essas também precisam ser levadas para memória em algum momento.



REALOCAÇÃO DE ENDEREÇO

Quando o processo é alocado na memroia ele pode conter endereços relativos que precisam ser ajustados aos endereços reais da memória.

CONFIGURAÇÃO INICIAL

Quando o programa já está carregado na memória, o loader configura o ambiente do programa, inicializa variáveis e configurações de segurança.

TRANSFERÊNCIA DE CONTROLE

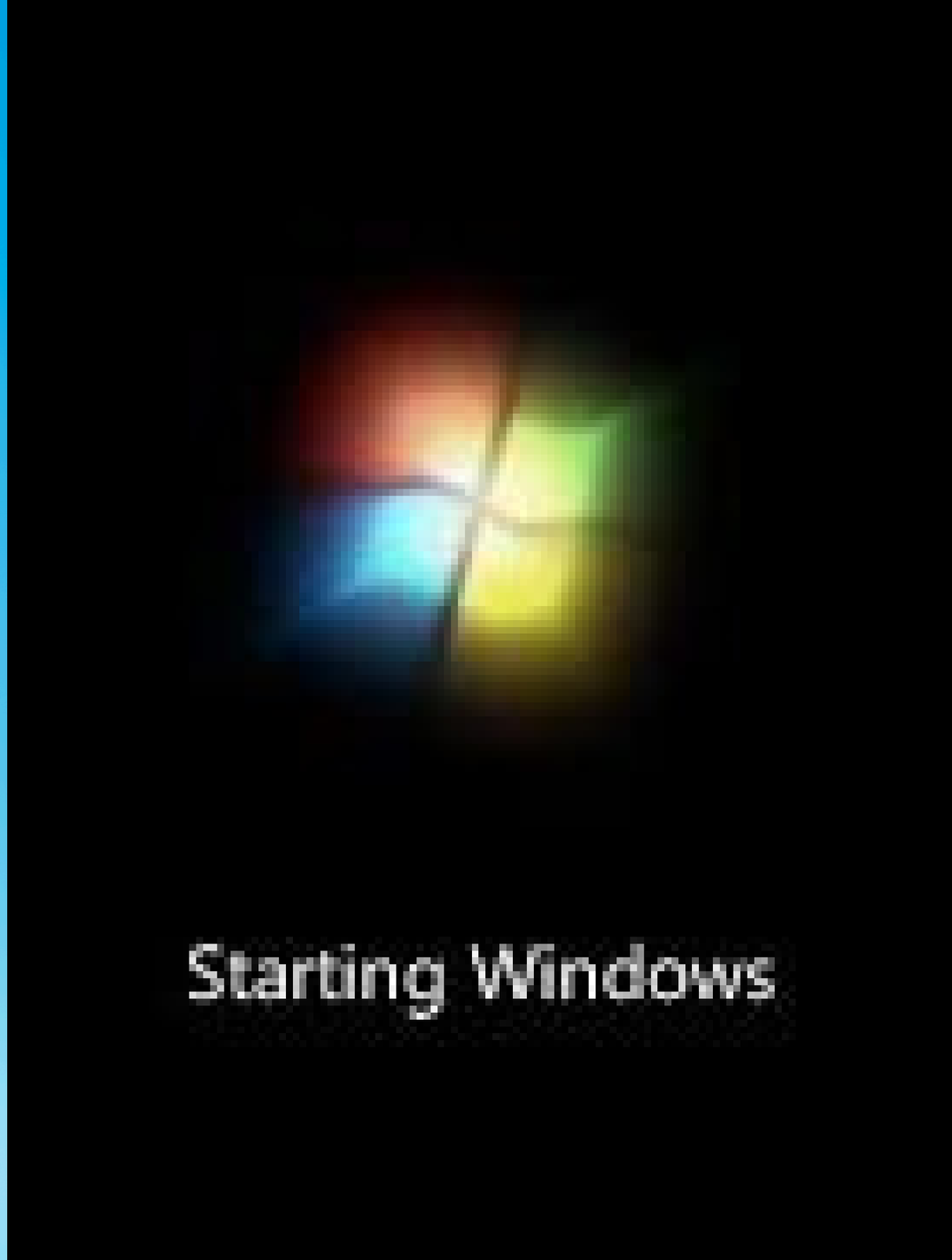
Quando tudo já está configurado e ajustado. O loader transfere o controle para o ponto de entrada e a execução do programa segue com as próprias pernas "sem o loader".

fim.

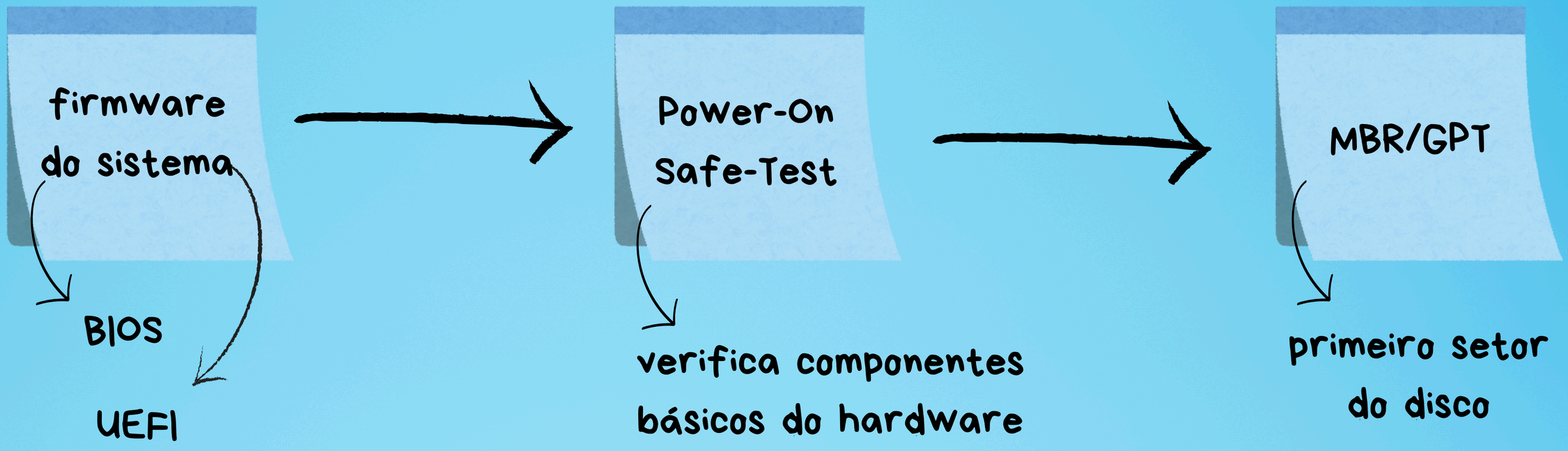
TIPOS DE LOADER

Boot Loader: é usado quando o computador é inicializado pois vai carrregar o sitema operacional na memória principal.

Se o computador possui mais de um sistema operacional instalado, o bootloader gerecia qual sistema operacional deve iniciar. Além de realizar verificações de segurança.



QUEM CARREGA O BOOTLOADER?



TIPOS DE LOADER

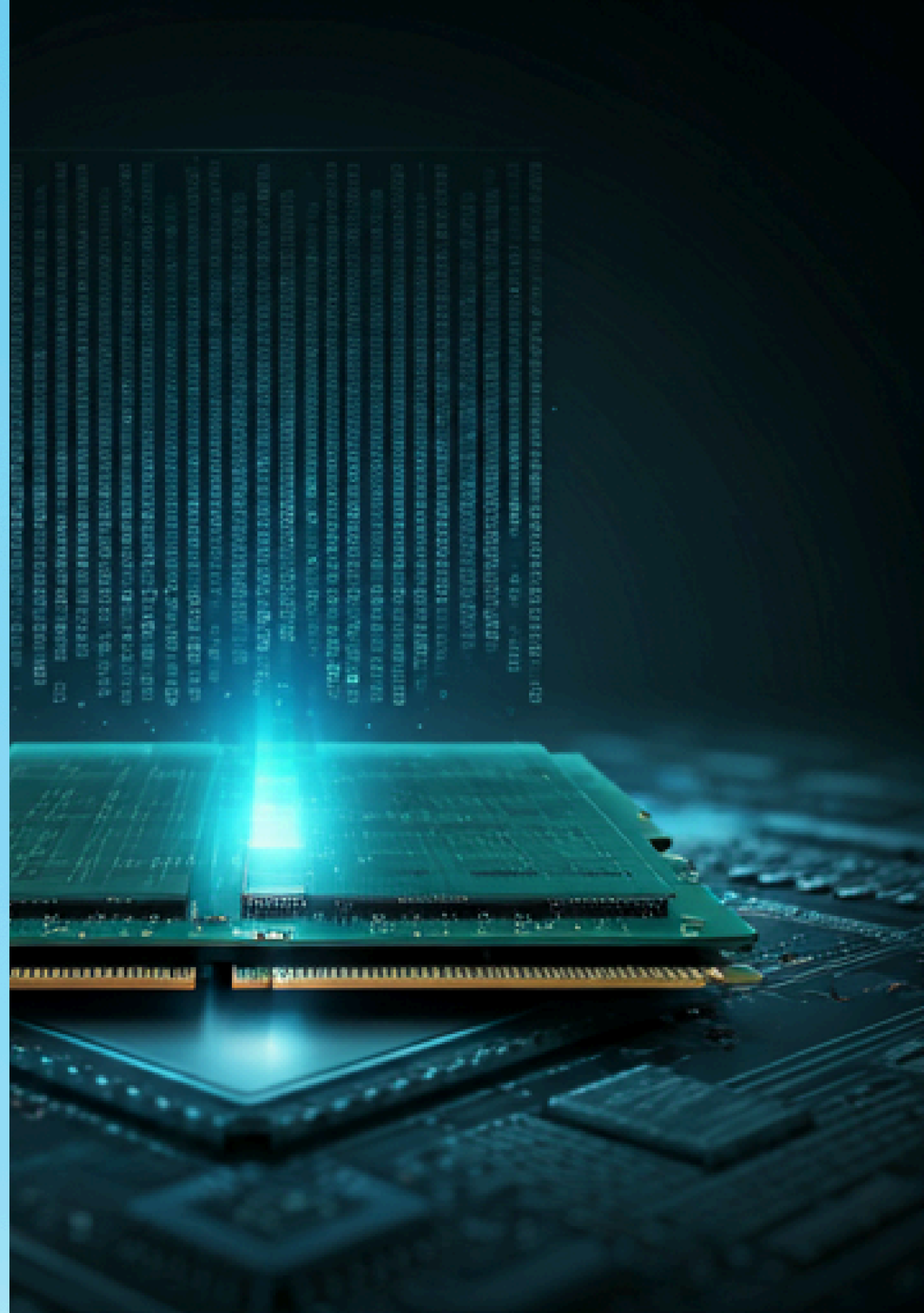
Kernel Loader: Carrega o núcleo do sistema operacional

Application Loader: Carrega programas de aplicativos na memória, permitindo sua execução.

TIPOS DE LOADER

Loader Absoluto: O loader só necessita conhecer o endereço de memória inicial e o tamanho do módulo para realizar o carregamento.

Loader realocável: Faz a realocação dos endereços na memória. Permite o carregamento em qualquer local da memória.



MODO DE AGIR

ESTÁTICO: carrega todos os dados necessários para a execução do programa, como todos os dados estarão disponíveis, a execução é mais estável.

DINÂMICO: carrega parte do programa na memória e carrega outros módulos conforme a necessidade. Garante um uso mais eficiente da memória.

OBRIGADO!

REFERÊNCIAS

- https://www.geeksforgeeks.org/static-and-dynamic-loader-in-operating-system/
- https://www.ufsm.br/app/uploads/sites/413/2018/12/08_sistemas_operacionais.pdf
- https://learn.microsoft.com/pt-br/windows/win32/api/memoryapi/nf-memoryapi-virtualalloc
- https://www.dic.app.br/2003/02/loader-utilitario-de-carga.html
- https://www.geeksforgeeks.org/basic-functions-of-loader/
- https://www.lenovo.com/us/en/glossary/loader/?orgRef=https%253A%252F%252Fwww.google.com%252F&srsltid=AfmBOorKEt895D15jz7o5E80SV30pZ6C_sWjK9yCEuDP117N9kyqHhb5
- https://cs.uok.edu.in/Files/79755f07-9550-4aeb-bd6f-5d802d56b46d/Custom/introduction%20to%20loaders.pdf
- https://www.scaler.com/topics/what-is-loader/
- https://www.ibm.com/docs/en/i/7.4?topic=ssw_ibm_i_74/apis/mmap.html
- https://www.fortinet.com/resources/cyberglossary/what-is-firmware#:~:text=Firmware%20Security-,Firmware%20Definition,their%20memory%20to%20function%20smoothly.

REFERÊNCIAS

- https://www-techtarget-com.translate.goog/whatis/definition/executable-file-exe-file?_x_tr_sl=en&_x_tr_tl=pt&_x_tr_hl=pt&_x_tr_pto=sge#:~:text=Hera%20Wigmore-,O%20que%20%C3%A9%20um%20arquivo%20execut%C3%A1vel%20(arquivo%20EXE)?,%E2%80%8B%E2%80%8Bs%C3%A3o%20arquivos%20EXE.
- https://stackoverflow.com/questions/1616772/how-exactly-do-executables-work
- https://stackoverflow.com/questions/2048052/what-does-executable-file-actually-contain?rq=3
- https://stackoverflow.com/questions/1495638/whats-in-a-exe-file?rq=3
- https://learn.microsoft.com/pt-br/windows/win32/debug/pe-format
- https://www.youtube.com/watch?v=-ojciptvVtY
- https://mentebinaria.gitbook.io/engenharia-reversa/o-formato-pe
- https://www.youtube.com/watch?v=cX5tQJhuNeY
- https://alacerda.github.io/posts/o-formato-elf/
- https://www.cs.cmu.edu/afs/cs/academic/class/15213-f00/docs/elf.pdf

REFERÊNCIAS

- https://man7.org/linux/man-pages/man5/elf.5.html
- https://learn.microsoft.com/en-us/windows/win32/dlls/dynamic-link-libraries
- https://www.spiceworks.com/tech/tech-general/articles/what-is-dll/#:~:text=Advantages%20of%20DLL-,What%20Is%20a%20Dynamic%20Link%20Library%20(DLL)?,applications%2C%20to%20access%20UI%20components.
- https://pt.wikipedia.org/wiki/Arquivo_objeto
- https://www.linuxjournal.com/article/1059
- https://maskray.me/blog/2024-01-14-exploring-object-file-formats
- https://www.youtube.com/watch?v=1VnnbpHDBBA&t=1s