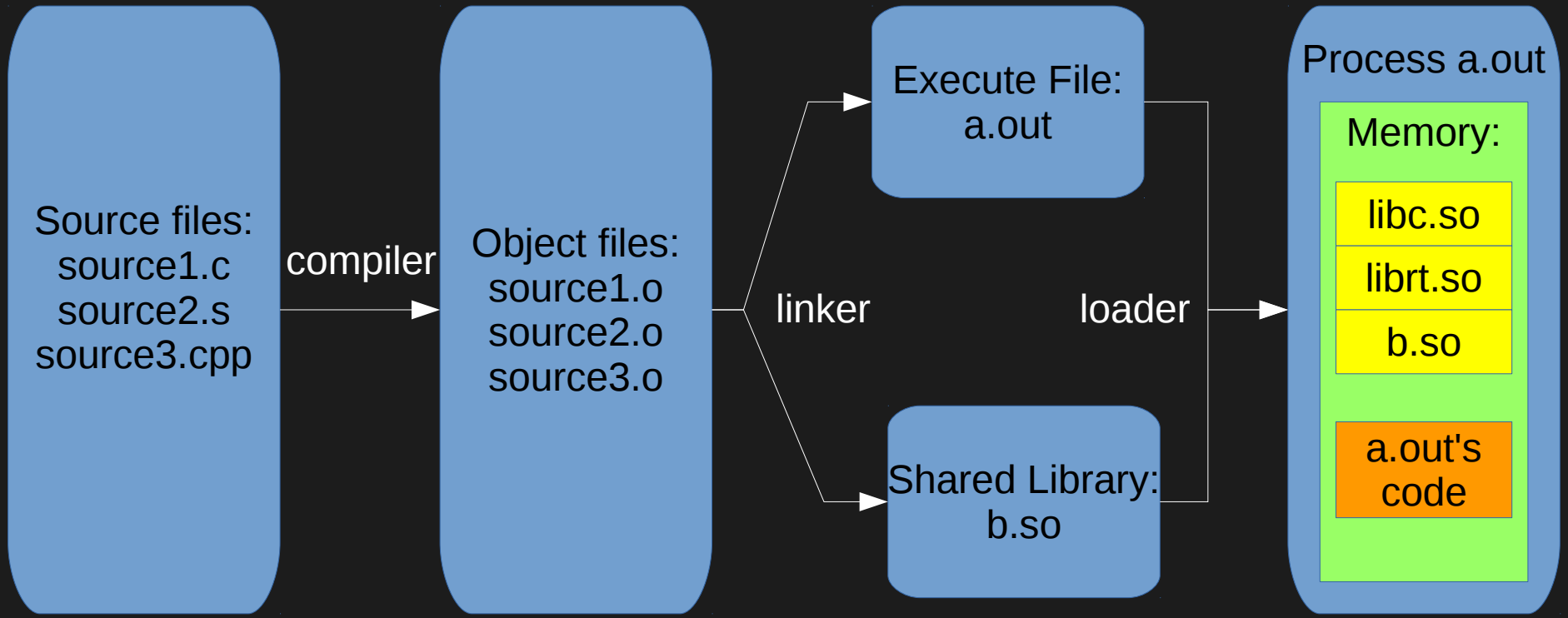
Executable File Format

Wei Wang

Viruses

- File infection
- Types of viruses
- Detection of viruses
- Anti-anti-virus

Compiler, Linker and Loader



Compiler, Linker and Loader cont'd

- Compiler transforms source code into binary machine code (object code)
  - Example: gcc, Clang, vc_compilerCTP.exe
- Linker takes object files and libraries files, and combies them into a single executable file or library file
  - Example: GNU ld, lld, LINK.exe
- Loader load an executable file and libraries into memory to start a new process (part of OS)
  - Executable loader: load executable files
    - Example: execve (system call)
  - Dynamic linking load: load dynamic libraries
    - Example: ld-linux.so

Compiler, Linker and Loader cont'd
More about Loader

- Brings an executable file and required libraries on disk into memory to start a new process
- Tasks:
  - Copy executable file code (text section) and global variables (data section) into memory
  - Copy arguments and environment variables into memory
  - Initialize registers
  - Jump to start of program to execute (_start function)
  - Load dynamic libraries (map dynamic libraries code into memory)

Compiler, Linker and Loader cont'd

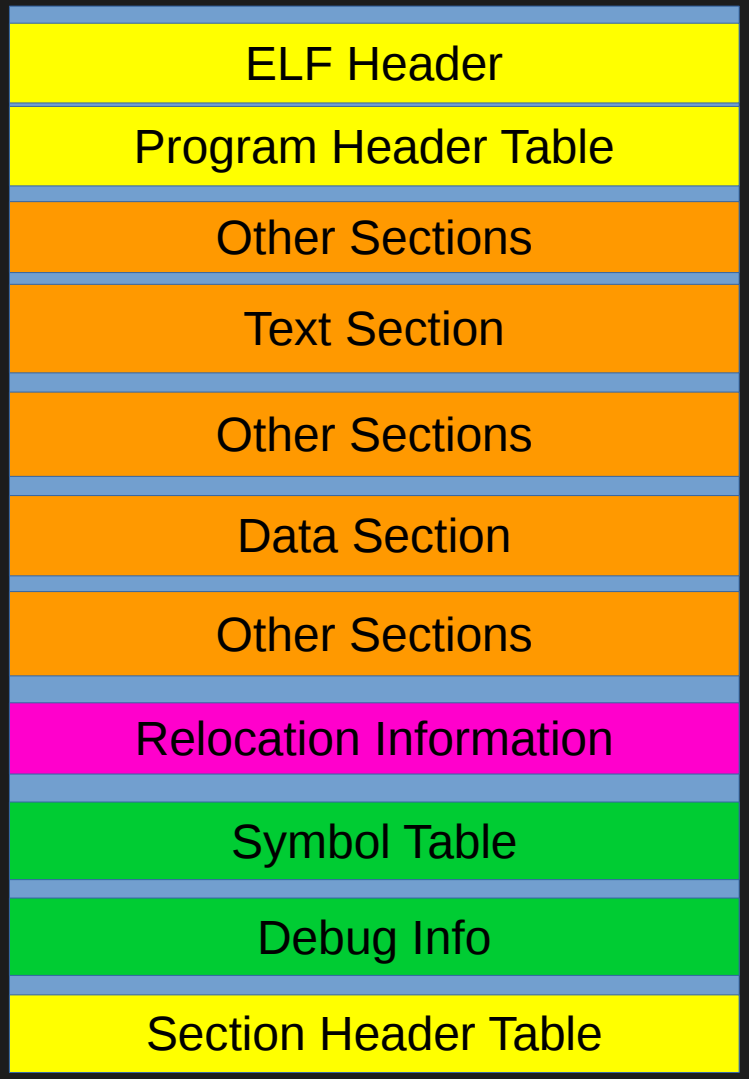
- For compiler, linker and loaders to work properly, they have to agree on the format of object files, executable files and library files
- The most common formats are:
  - ELF on *nix: Executable and Linkable Format
  - PE on Windows: Portable Executable
  - Mach-O on OS X

The ELF Format

- Executable and Linkable Format
- Defines format for:
  - Executables
  - Object files
  - Dynamic libraries (shared libraries)
  - Core dumps

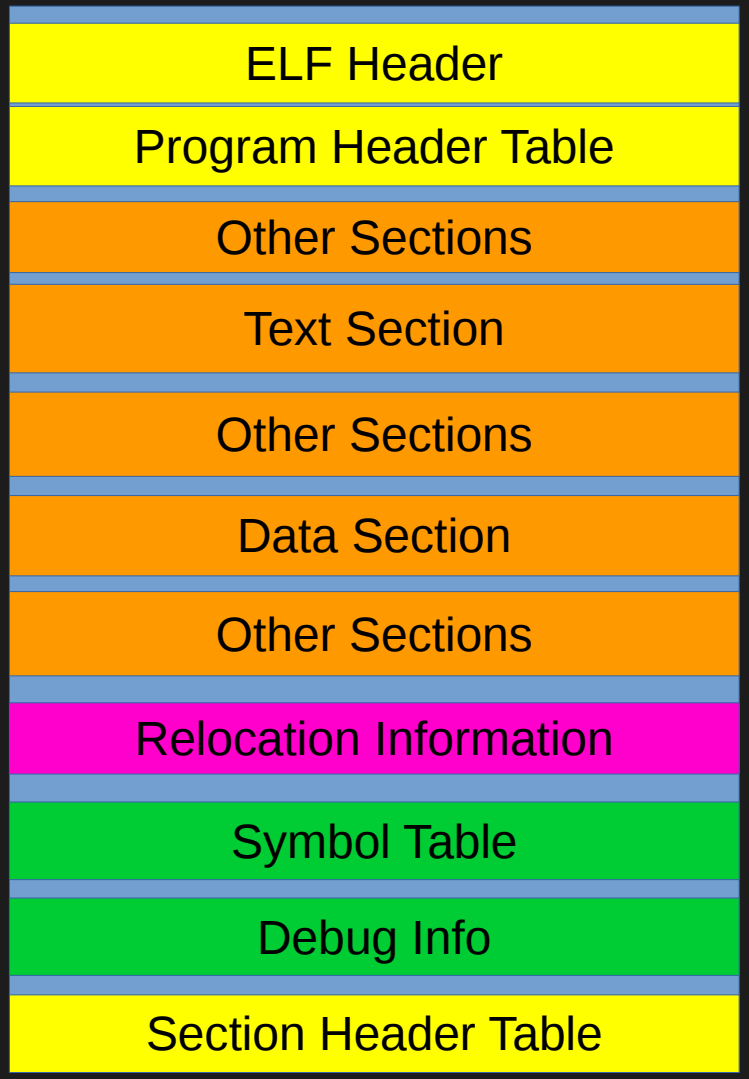
ELF Format Examples

- ELF Header: basic identification information of this file
- Program header table: location of text and data sections
- Text section: the code
- Relocation information: for relocatable text and data sections



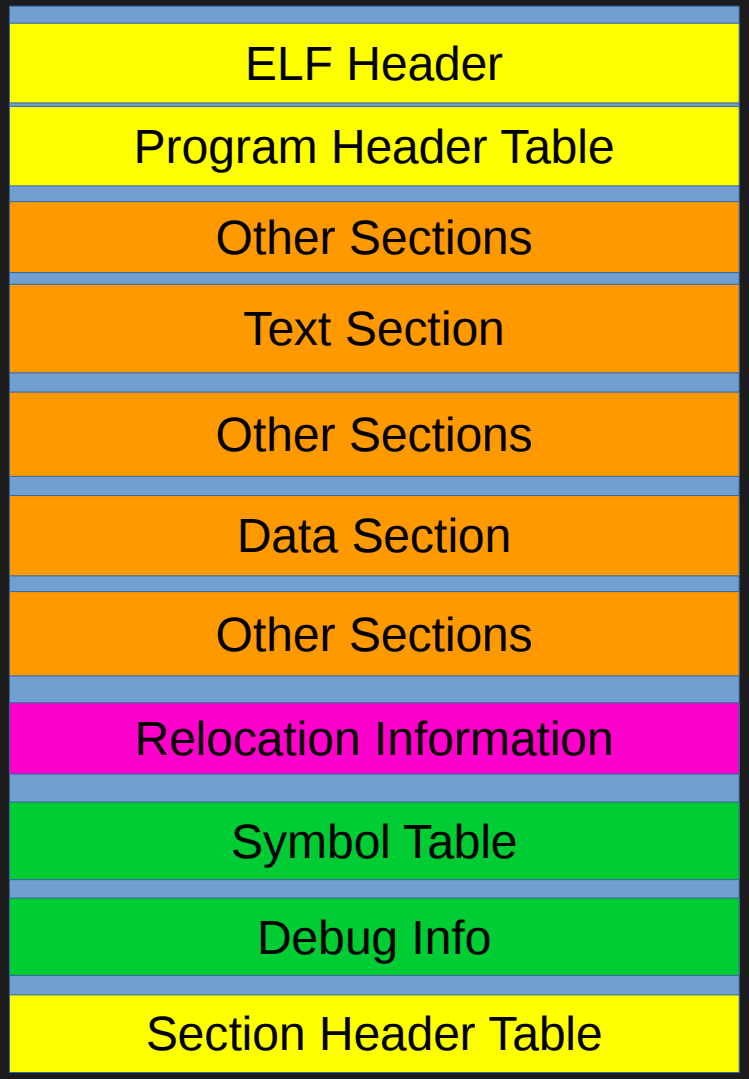
ELF Format Examples (cont'd)

- Data sections:
  - .rodata: read-only
  - .bss: uninitialized global variables
  - .data: initialized global variables
- Example of other sections:
  - .dynamic: dynamic linking information
  - .got: global offset table
  - .init; process initialization code

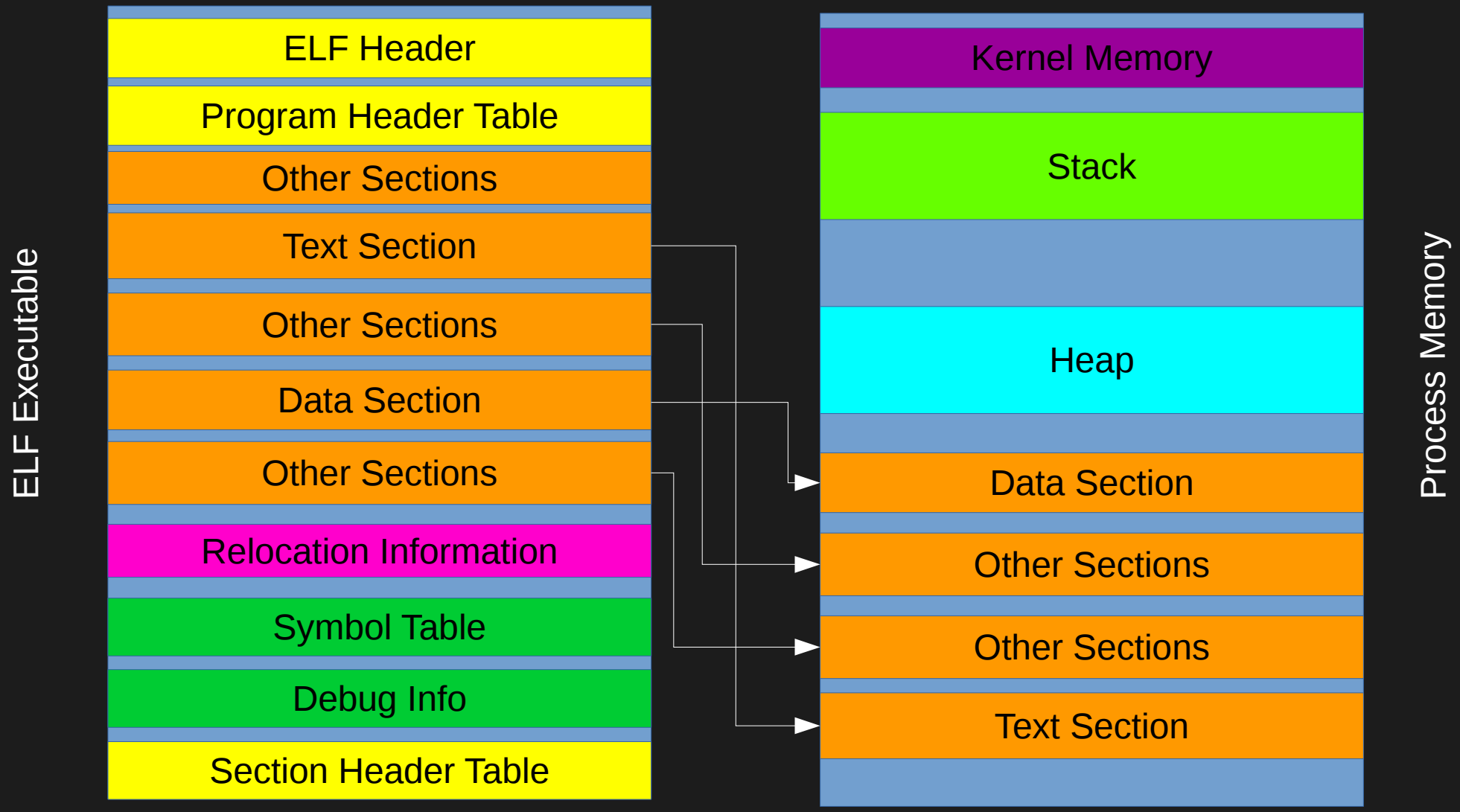


ELF Format Examples (cont'd)

- Symbol table: locate program symbolic definition (e.g., exported function name)
- Section header table: location and information of each section



ELF File to Process Memory



ELF File to Process Memory cont'd

- Some sections will be directly copied to memory:
  - Example: .text, .data, .init, .dynamic
  - The location (memory addresses) of these sections are defined in the ELF (if not PIC/PIE and ASLR)
- Some sections will not be copied to memory
  - Example: symbol table, debug info

Analyzing ELF Files

- readelf: Display information about ELF files
  - readelf -h *executable*
    - Show ELF header

```
ELF Header:
  Magic:   7f 45 4c 46 01 01 01 00 00 00 00 00 00 00 00 00
  Class:                             ELF32
  Data:                              2's complement, little endian
  Version:                           1 (current)
  OS/ABI:                            UNIX - System V
  ABI Version:                       0
  Type:                              EXEC (Executable file)
  Machine:                           Intel 80386
  Version:                           0x1
  Entry point address:               0x8048330
```

Analyzing ELF Files cont'd

- readelf -S executable
  - Show section information


```
[Nr] Name              Type            Addr     Off    Size   ES Flg Lk Inf Al
[ 0]                   NULL            00000000 000000 000000 00      0   0  0
[ 1] .interp           PROGBITS        08048134 000134 000013 00   A  0   0  1
  . . .
[ 5] .dynsym           DYNSYM          080481ac 0001ac 000060 10   A  6   1  4
[ 6] .dynstr           STRTAB          0804820c 00020c 000053 00   A  0   0  1
  . . .
[ 9] .rel.dyn          REL             0804828c 00028c 000008 08   A  5   0  4
  . . .
[13] .text             PROGBITS        08048330 000330 0001d2 00  AX  0   0 16
[24] .data             PROGBITS        08049750 000750 000008 00  WA  0   0  4
[28] .debug_info PROGBITS 00000000 00079e 0000d6 00      0   0  1
```

Analyzing ELF Files cont'd

- Each section also has a flg
- In the end of readelf -S output, the flags explained
- The flag bits determine whether a section can be read, written, executed, etc., NOT the section name; viruses might modify the flag bits so that a .text section becomes writable!

Analyzing ELF Files cont'd

- readelf has many other useful options
  - Read the man page for more information
- objdump: the disassemblr
- hexdump: raw hexadecimal dump
- file: determine file type
  - file *executable*
- For more information, Google “ELF format specification”

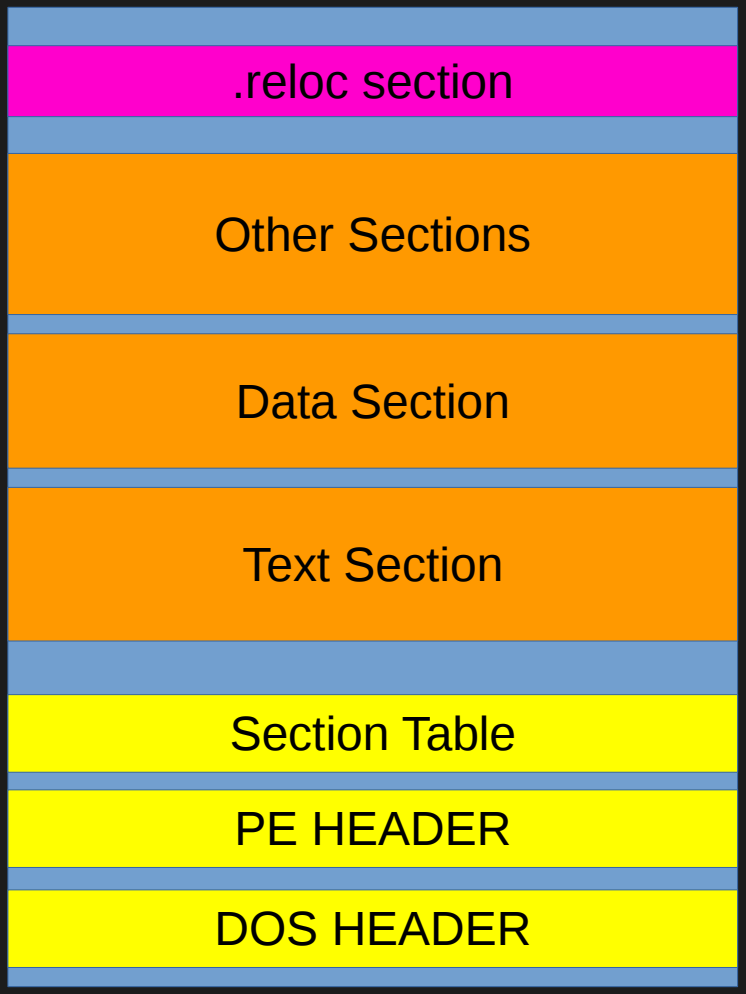
The PE Format

- Portable Executable
- Also called PE32 (because it is 32-bit code); PE32+ is for 64-bit code
- Older formats exist for 16-bit DOS and Windows 3.1

The PE Format cont'd

- Similar to ELF format
  - PE header and DOS header
  - Text and data sections
  - Relocation informations
  - Symbol table
  - Debug information
  - And other sections
- Common sections are .text (for code), .data (read/write data), .rdata (read-only data, .reloc (relocation data used to build IATs)

PE Format Example



DOS Header

- If a program is invoked within a DOS command prompt window, it starts executing here
- For most PE32 executables, the DOS header contains a tiny executable that prints: “This application must be run from Windows”, then exits

Dead Space in Executable File Formats

- There are empty spaces in executable files
  - The beginning of ELF files
  - Empty spaces between functions
  - Empty spaces between sections
  - Nops in functions
  - Some linkers make executable file align to page boundaries
    - Simpilies the loader's job

Executable File Format and Viruses

- Question: Why do we care about the details of the PE file format?
- Answer: Because a virus writer will try to infect the PE file in such a way as to make the virus code execute, while making the PE file look as it would normally look. The job of anti-virus software is to find well-disguised viruses.
- Dead spaces are perfect locations to hide viruses
  - CIH virus break itself into parts and hide in the dead spaces between PE sections