Intro: Portable Executable (PE)

File Format

Agenda

- Portable Executable File Format
  - PE Headers
  - Data Directories
    - Import Address Table
    - Relocation Table
  - Misc Structures
- Building a PE Loader
  - Segment Mapping
  - IAT Processing
  - Relocation

Portal Executable

The Portable Executable (PE) format is a file format for executables, object code and DLLs, used in 32-bit and 64-bit versions of Windows operating systems. The term "portable" refers to the format's versatility in numerous environments of operating system software architecture.

The PE format is a data structure that encapsulates the information necessary for the Windows OS loader to manage the wrapped executable code. This includes dynamic library references for linking, API export and import tables, resource management data and thread-local storage (TLS) data.

http://en.wikipedia.org/wiki/Portable_Executable

PE Loader: Questions?

- How to **validate** Executables against Platform (x86, x64?)
- **Where** to **Load** an Executable
  - Where to load data
    - Initialized/Uninitialized
  - Where to load code
- How to **resolve** Library Dependencies
- What to do if required address is **not available**
- How to ensure **Position Independence**
- How to associate **symbols** and **debug information**
- …….

PE File Format

PE Headers

Field Name	Data Value	Description
Machine	014Ch	i386®
Number of Sections	0003h	
Time Date Stamp	3B7D8410h	17/08/2001 20:52:32
Pointer to Symbol Table	00000000h	
Number of Symbols	00000000h	
Size of Optional Header	00E0h	
Characteristics	010Fh	
Magic	010Bh	PE32
Linker Version	0007h	7.0
Size of Code	00012800h	
Size of Initialized Data	00009C00h	
Size of Uninitialized Data	00000000h	
Address of Entry Point	01012475h	
Base of Code	00001000h	
Base of Data	00014000h	
Image Base	01000000h	

Field Name	Data Value	Description
Section Alignment	00001000h	
File Alignment	00000200h	
Operating System Version	00010005h	5.1
Image Version	00010005h	5.1
Subsystem Version	00000004h	4.0
Win32 Version Value	00000000h	Reserved
Size of Image	0001F000h	126976 bytes
Size of Headers	00000400h	
Checksum	0001D7FCh	
Subsystem	0002h	Win32 GUI
Dll Characteristics	8000h	Terminal Server aware
Size of Stack Reserve	00040000h	
Size of Stack Commit	00001000h	
Size of Heap Reserve	00100000h	
Size of Heap Commit	00001000h	
Loader Flags	00000000h	Obsolete
Number of Data Directories	00000010h	

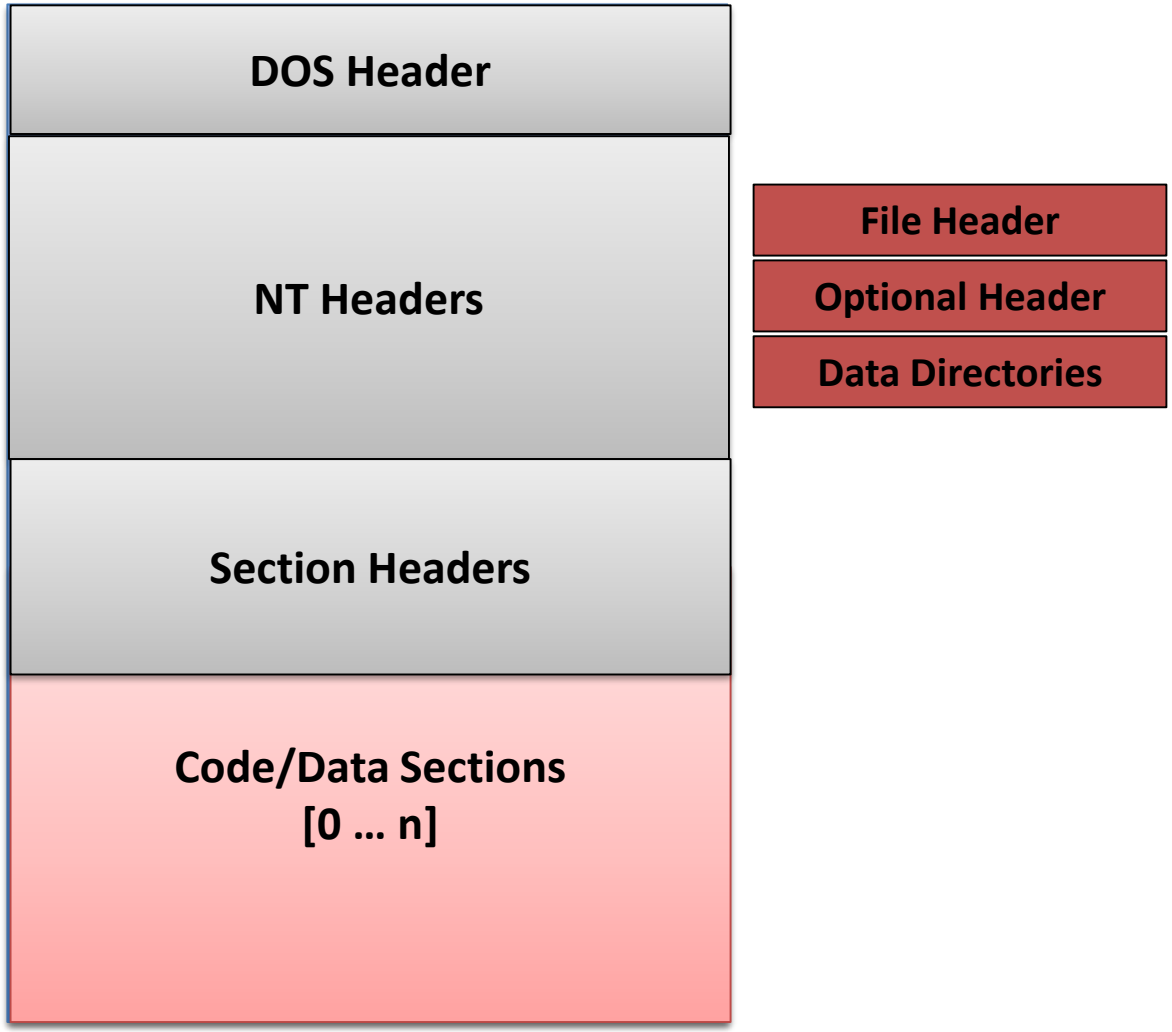
Data Directories

Directory Name	Virtual Address	Size
Export Table		
Import Table	01012B80h	0000008Ch
Resource Table	01016000h	00008960h
Exception Table		
Certificate Table		
Relocation Table		
Debug Data	01001240h	0000001Ch
Architecture-specific data		
Machine Value (MIPS GP)		
TLS Table		
Load Configuration Table		
Bound Import Table	01000260h	00000080h
Import Address Table	01001000h	00000228h
Delay Import Descriptor		
COM+ Runtime Header		
(15) Reserved		

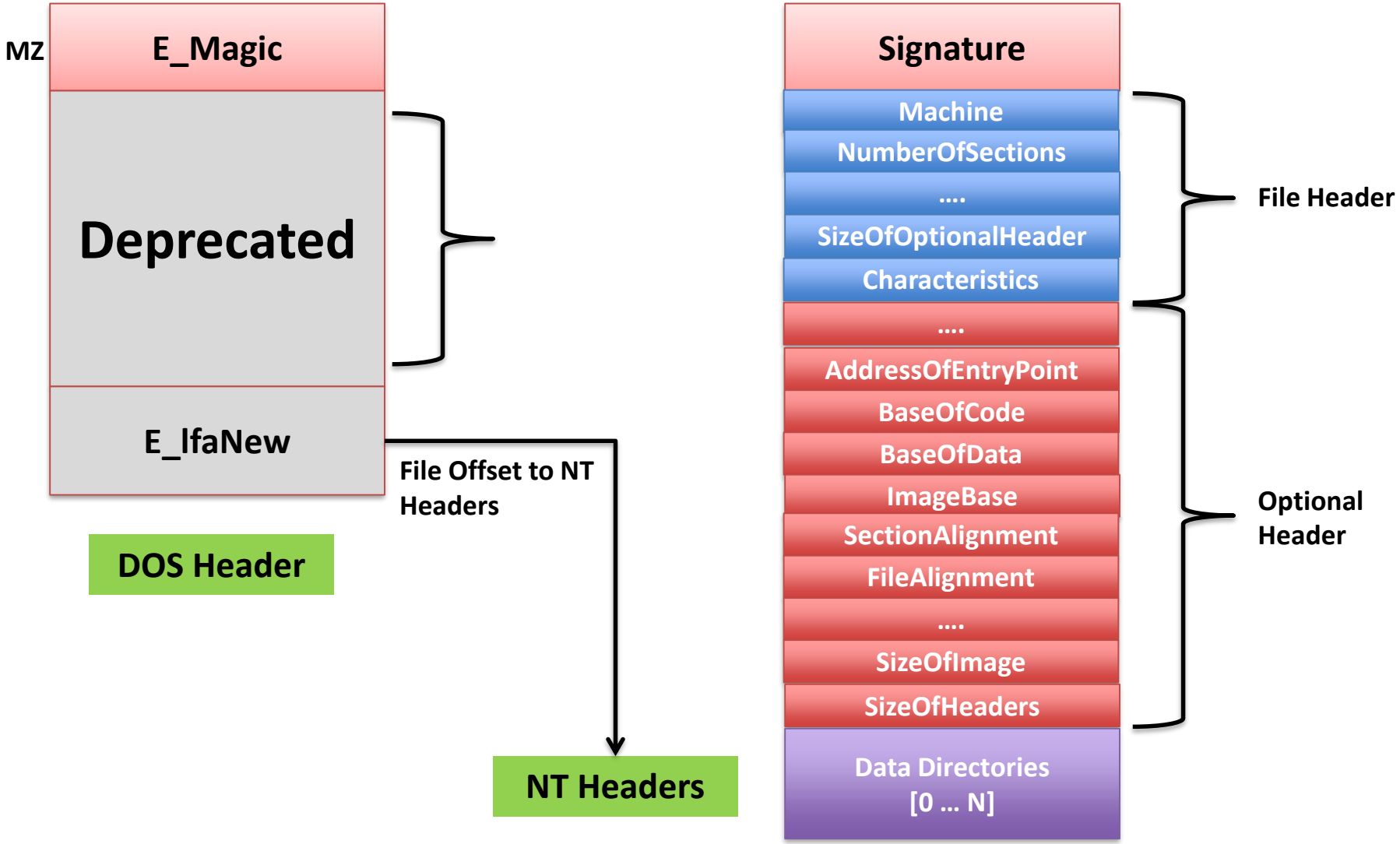
Section Headers

Name	Virtual Size	Virtual Address	Size of Raw Data	Pointer to Raw Data	Characteristics	Pointing Directories
.text	000126B0h	01001000h	00012800h	00000400h	60000020h	Import Table; Debug Data; Import ...
.data	0000101Ch	01014000h	00000A00h	00012C00h	C0000040h	
.rsrc	00008960h	01016000h	00008A00h	00013600h	40000040h	Resource Table

PE File Format

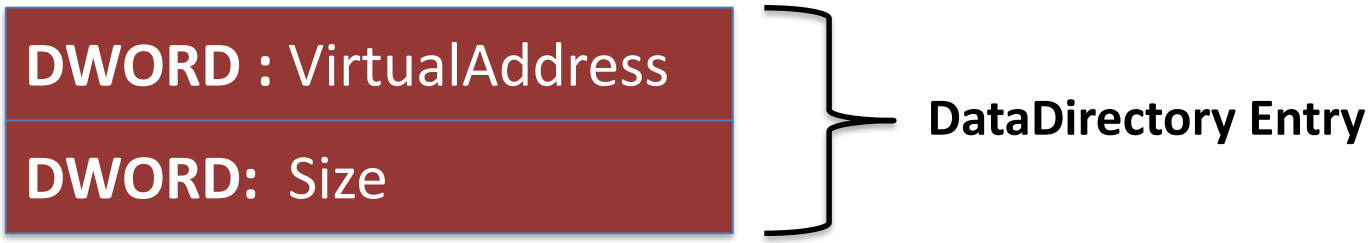


PE Header: DOS & NT Header



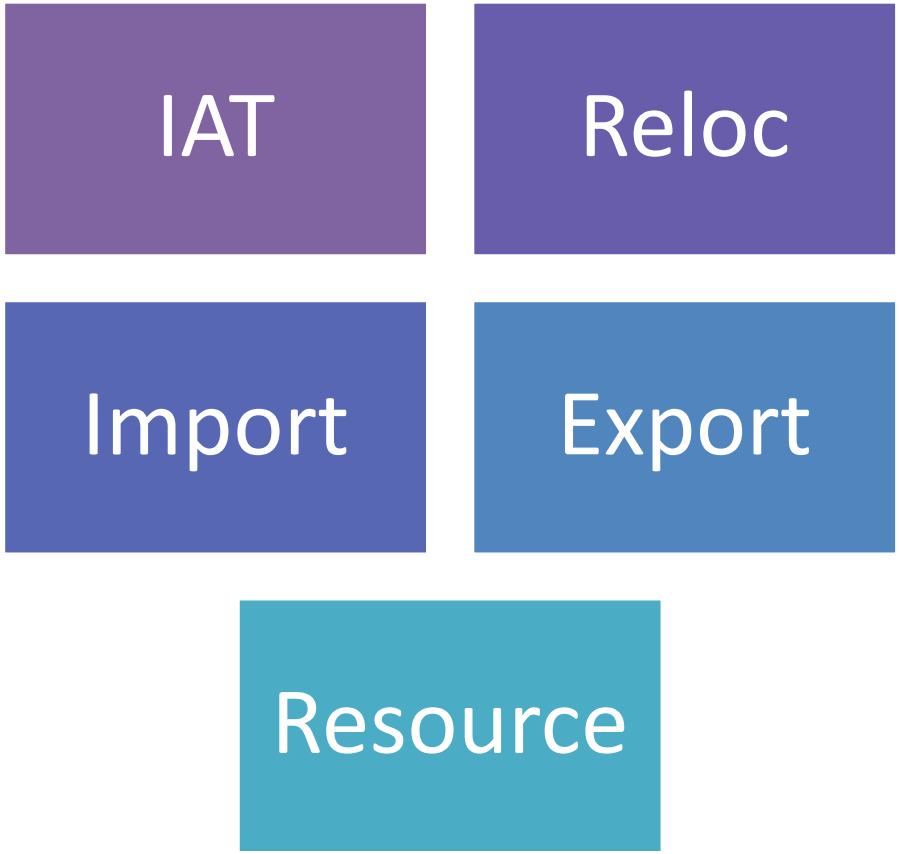
PE: Addressing

PE Header: **OptionalHeader->DataDirectory[]**

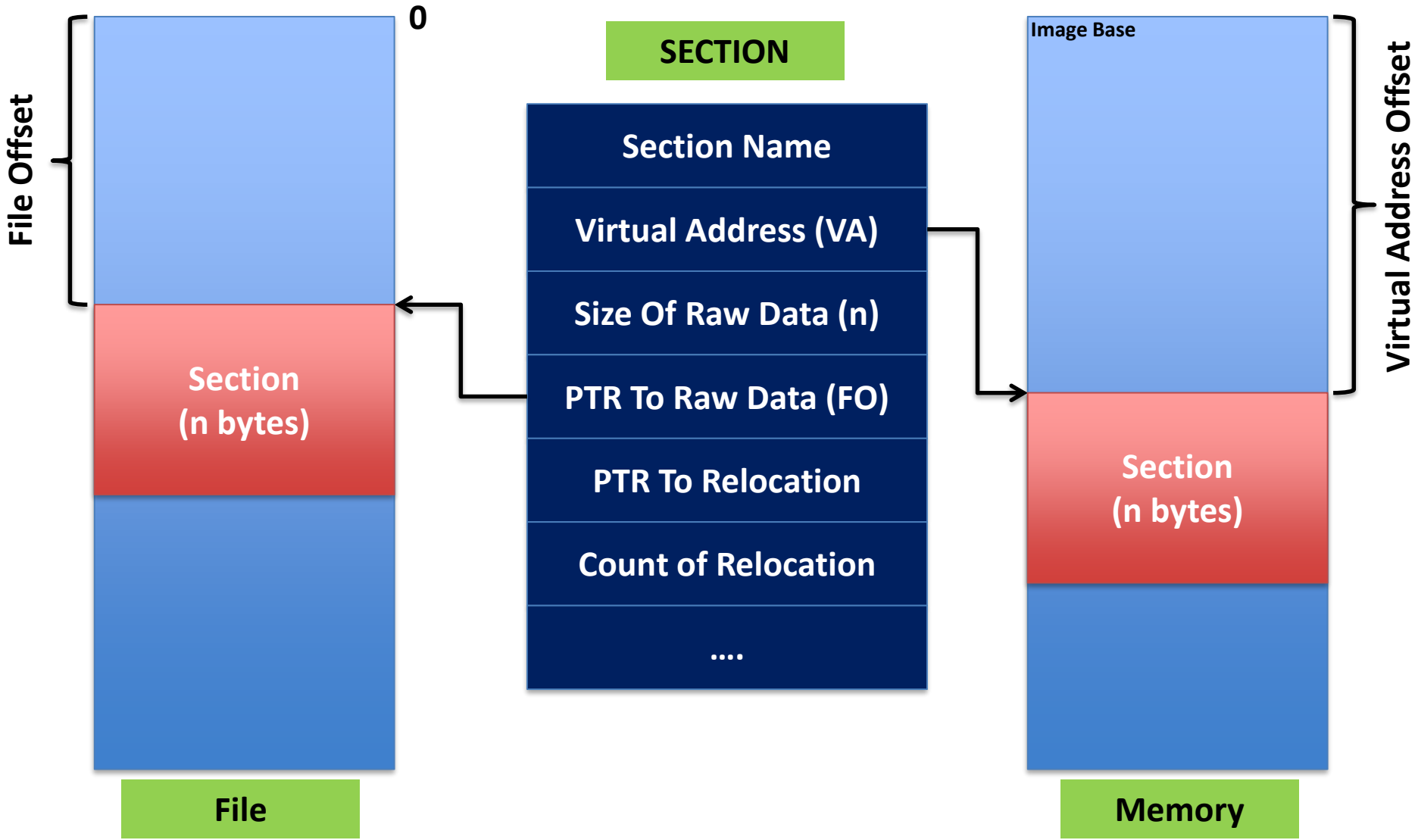


Directory Type	Array Index
IMAGE_DIRECTORY_ENTRY_EXPORT	0
IMAGE_DIRECTORY_ENTRY_IMPORT	1
IMAGE_DIRECTORY_ENTRY_RESOURCE	2
IMAGE_DIRECTORY_ENTRY_EXCEPTION	3
IMAGE_DIRECTORY_ENTRY_SECURITY	4
IMAGE_DIRECTORY_ENTRY_BASERELOC	5
…	
IMAGE_DIRECTORY_ENTRY_IAT	12
….	

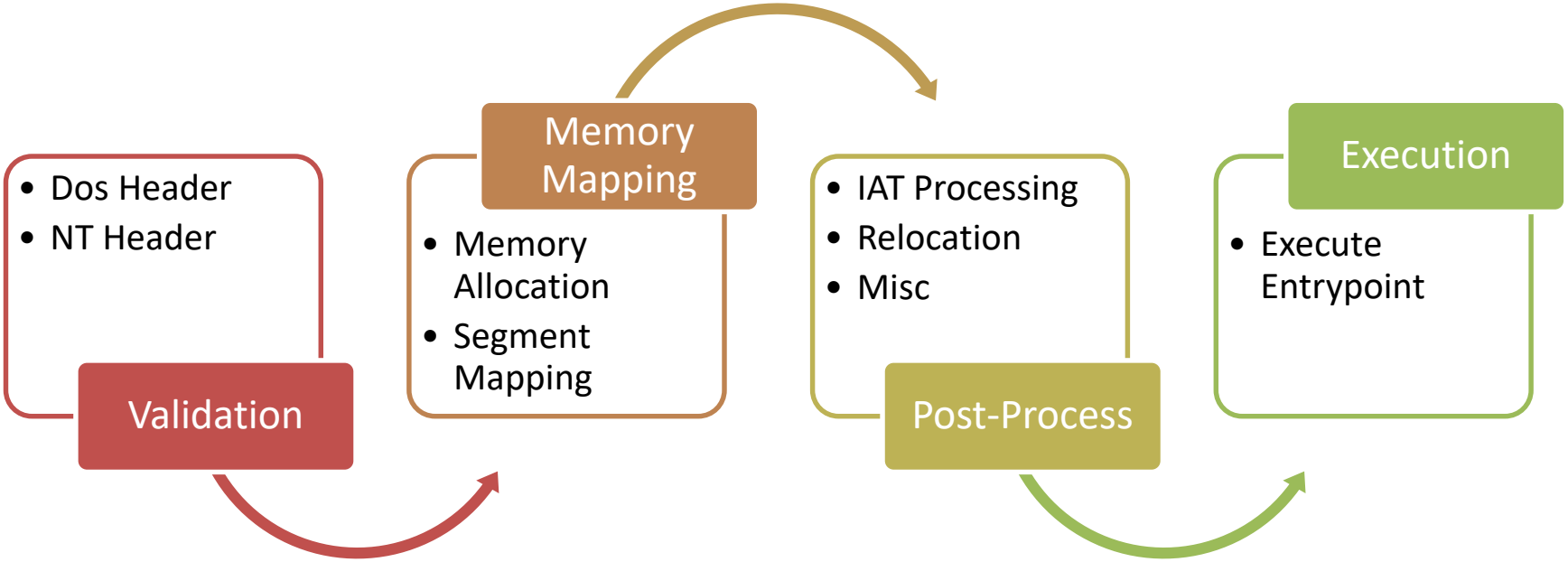
Important Data Directories



PE Header: Section



PE: Loader Process



PE Loader: Section Mapping

- For-Each-Section
  - Allocate memory for Image
  - Copy Section Data from File (PTR To Raw Data) to Memory (Image Base + Virtual Address)

PE Loader: IAT Processing

- For-Each-IAT-Entry
  - Load Library
  - For-Each-Function in IAT-Entry
    - Resolve Function
    - Patch IAT with Address of Resolved Function

PE Loader: Relocation

- If **ImageBase != PeHeader.OptHeader.ImageBase**
  - **D = PeHeader.OptHeader.ImageBase – ImageBase**
  - For-Each-Relocation
    - Apply Delta

PE Exports

- A special Data Directory in PE File that maps **Function Name** or **Function Ordinal** to **Function Offset**.
  - Use by **GetProcAddress(..)** to resolve Function Address in Loaded Library.

Explore PE Files with **Metasm**

```
require 'metasm'
```

```
pe = Metasm::PE.decode_file("1.exe")
```

```
pe.header          # File Header
```

```
pe.optheader   # Optional File Header
```

```
pe.sections       # PE Sections
```

```
pe.imports       # PE Import Entries
```

Thank You !

http://www.3SLabs.com