Windows Process Environment

Guo Yu

2014.01

School of Software Engineering
USTC-Suzhou

Outline

Address Space

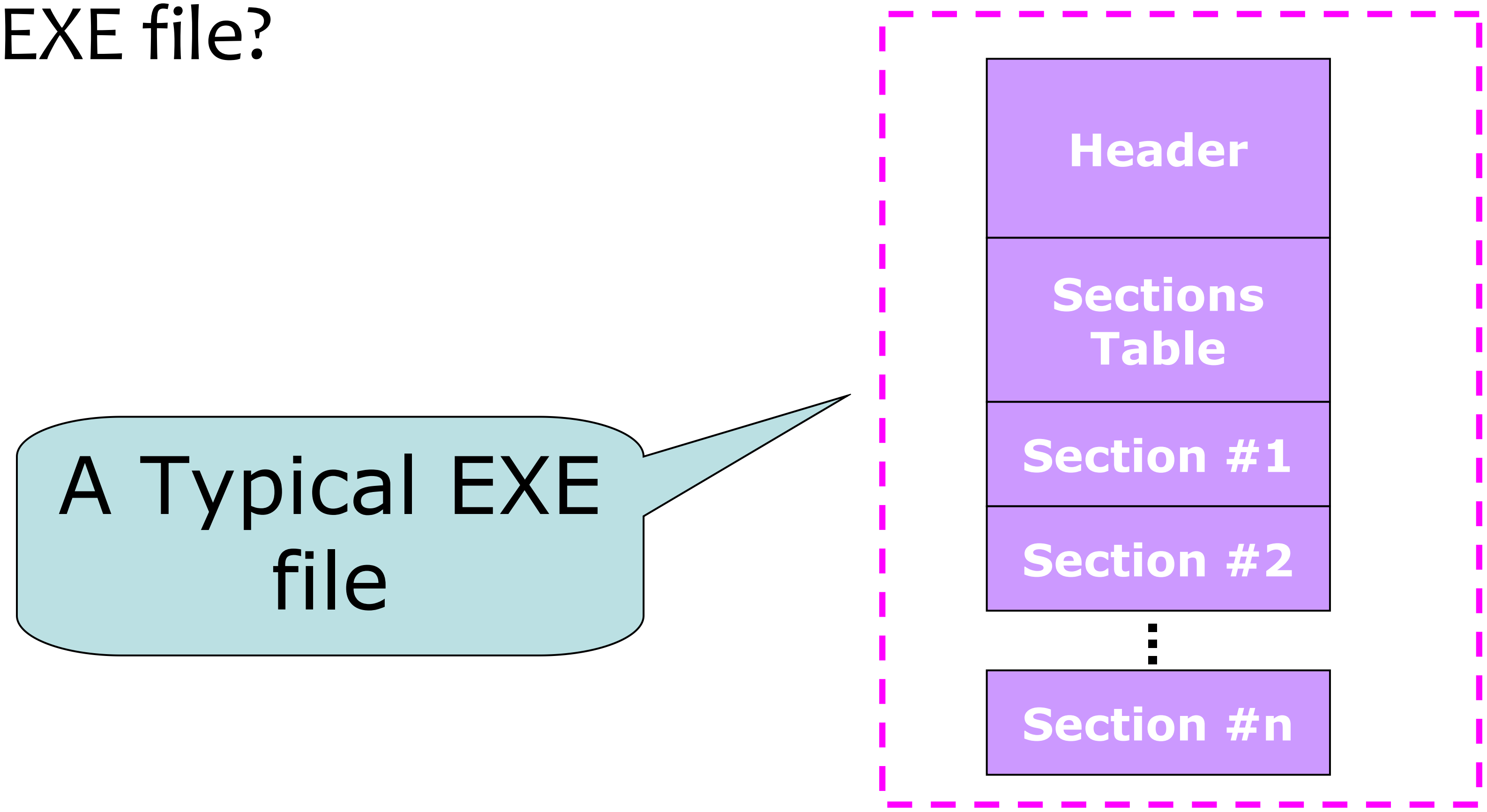
Process Environment Block

SEH & VEH

Window Messages

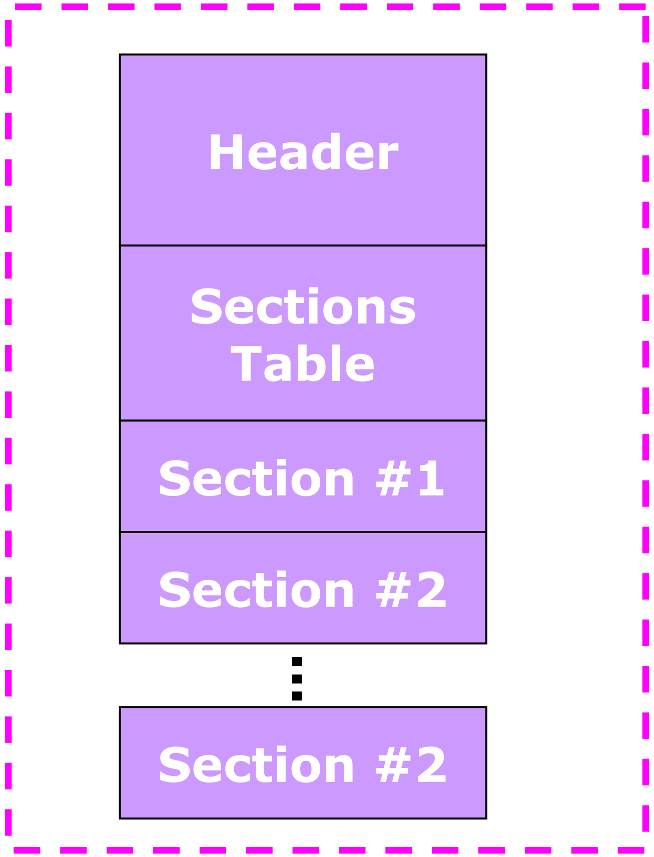
Motivation

- What will happen when we double-click an EXE file?



EXE file

- Include
  - Header
  - Section table
  - Code section
    - Assembly
  - Data section
    - Global variables
  - Resource section
    - Icon, button, window, …
  - Others



Answer

- Windows System Kernel “load” the EXE file into Memory, and jump to the entry of code in EXE file
- “Loading” ?
  - Create a process object
  - Map the EXE file into process space
    - Include “EXE Header” “.data” “.text” “.rdata” …
    - Drop some sections, e.g. “.debug” “.relc”
  - Relocation
  - Dynamic Linking it with DLLs
    - E.g. Kernel32.dll, user32.dll

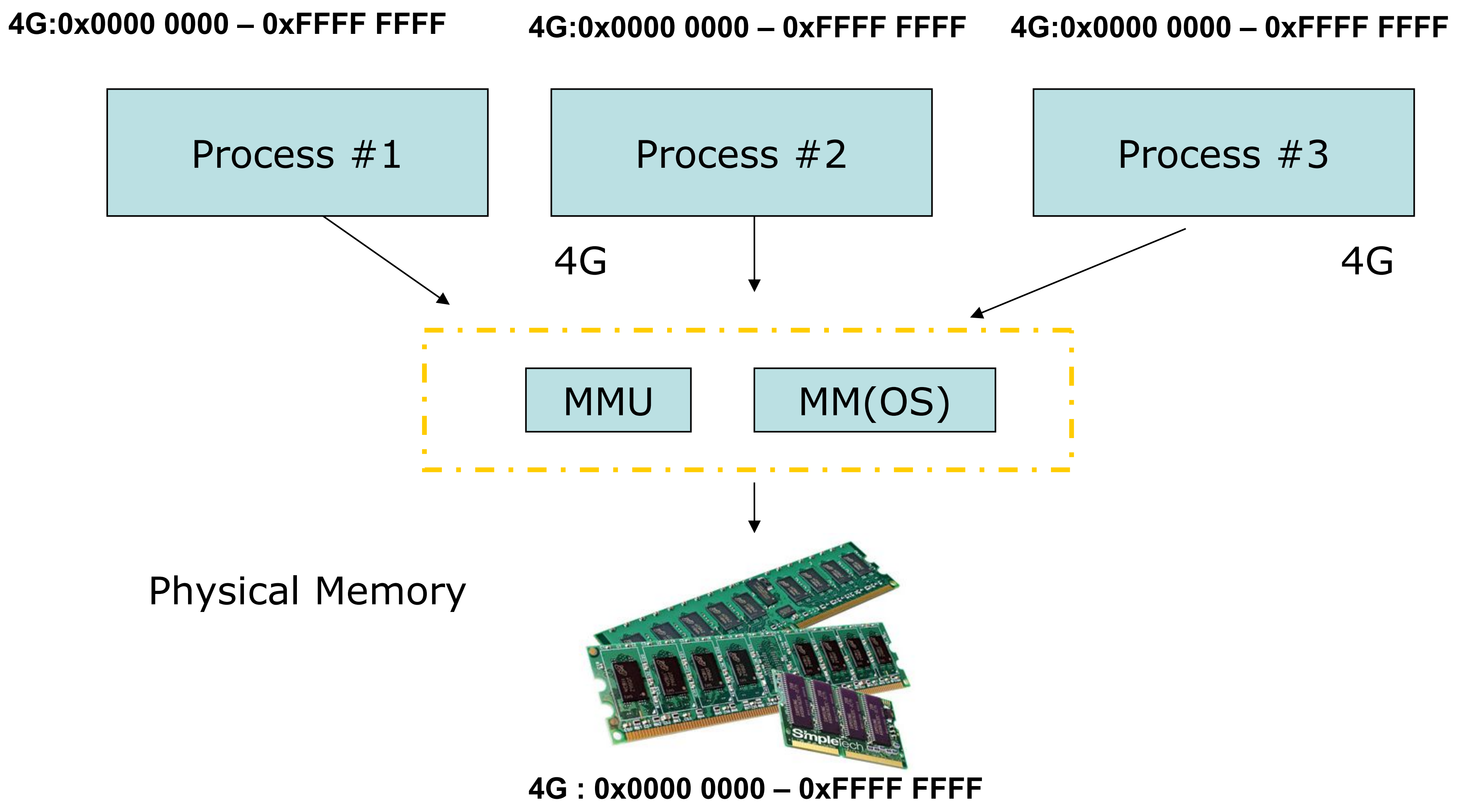
Example

- Let’s use ollydbg to watch the loading

Process Space

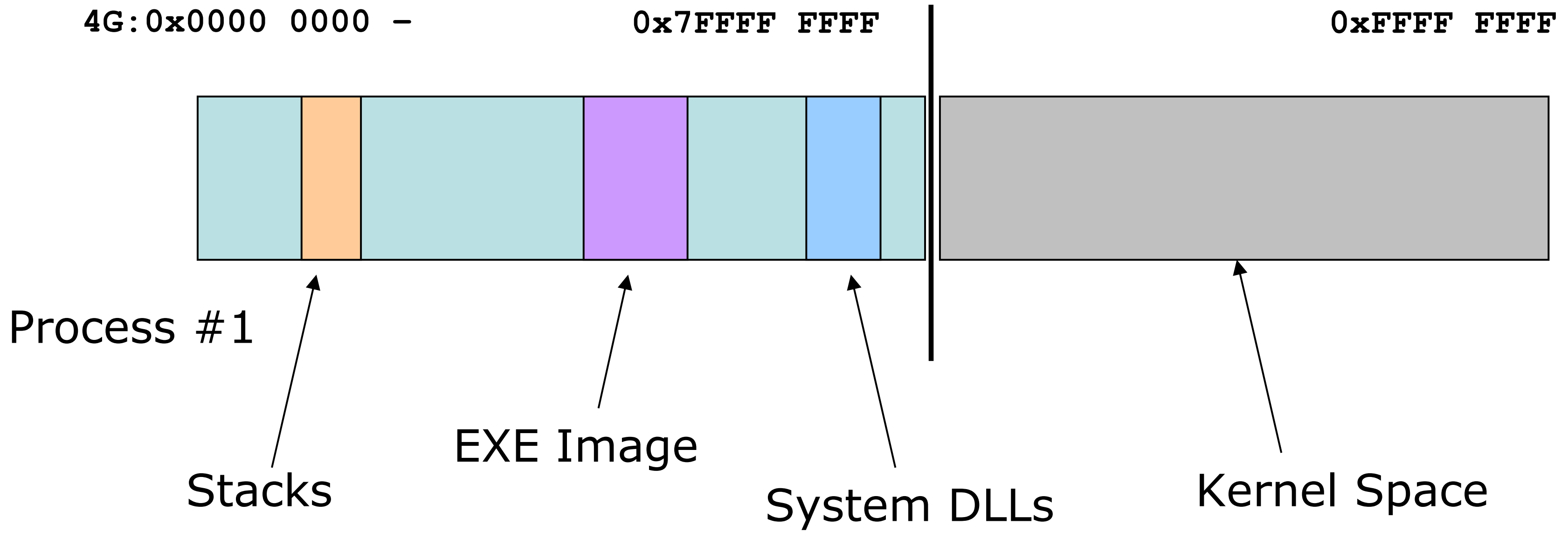
- Under Windows NT family Oss
  - Process Space
    - Addressing, 32bit Space
    - Size, 4G
  - We see many processes on my system
    - e.g. QQ.exe, svchost.exe, cmd.exe, a lot
  - Different processes are isolated from each other
    - Why ?

Virtual Memory


4G:0x0000 0000 – 0xFFFF FFFF
4G:0x0000 0000 – 0xFFFF FFFF
4G:0x0000 0000 – 0xFFFF FFFF
Process #1
Process #2
Process #3
4G
4G
MMU
MM(OS)
Physical Memory
4G : 0x0000 0000 – 0xFFFF FFFF

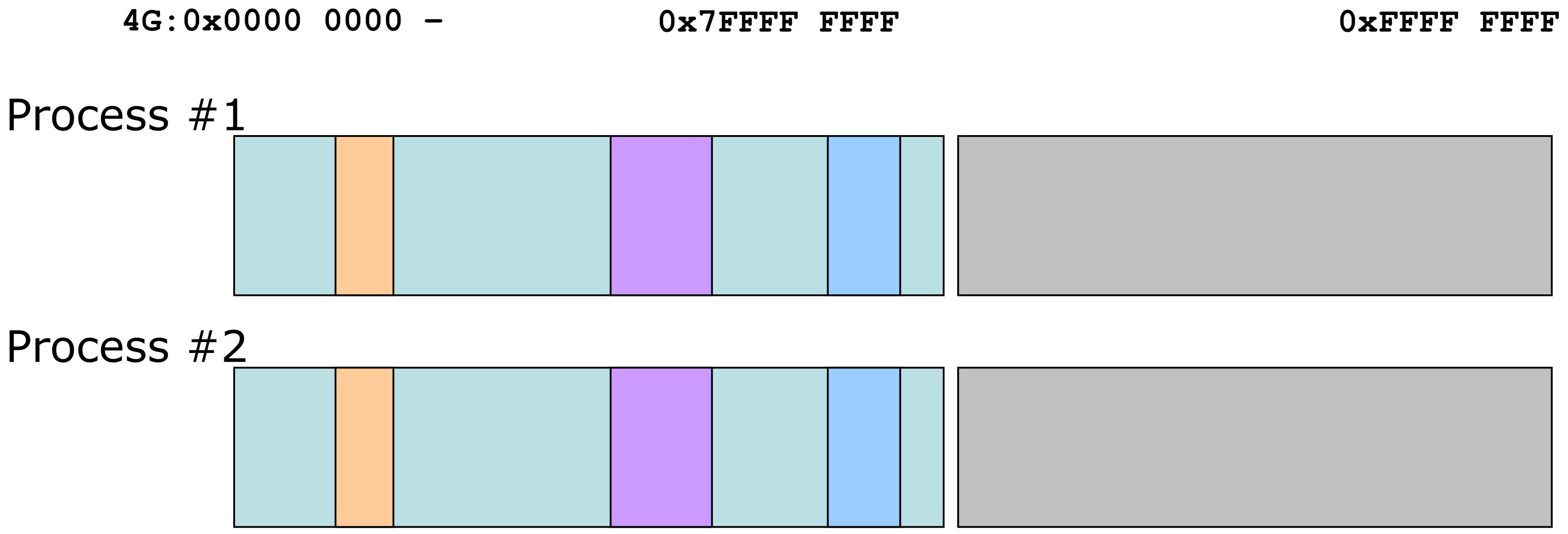
Process Memory

- 4G size space is divided into pages
  - Page size, 4k usually



Process Memory (cont.)

- Processes are separated by virtual memory

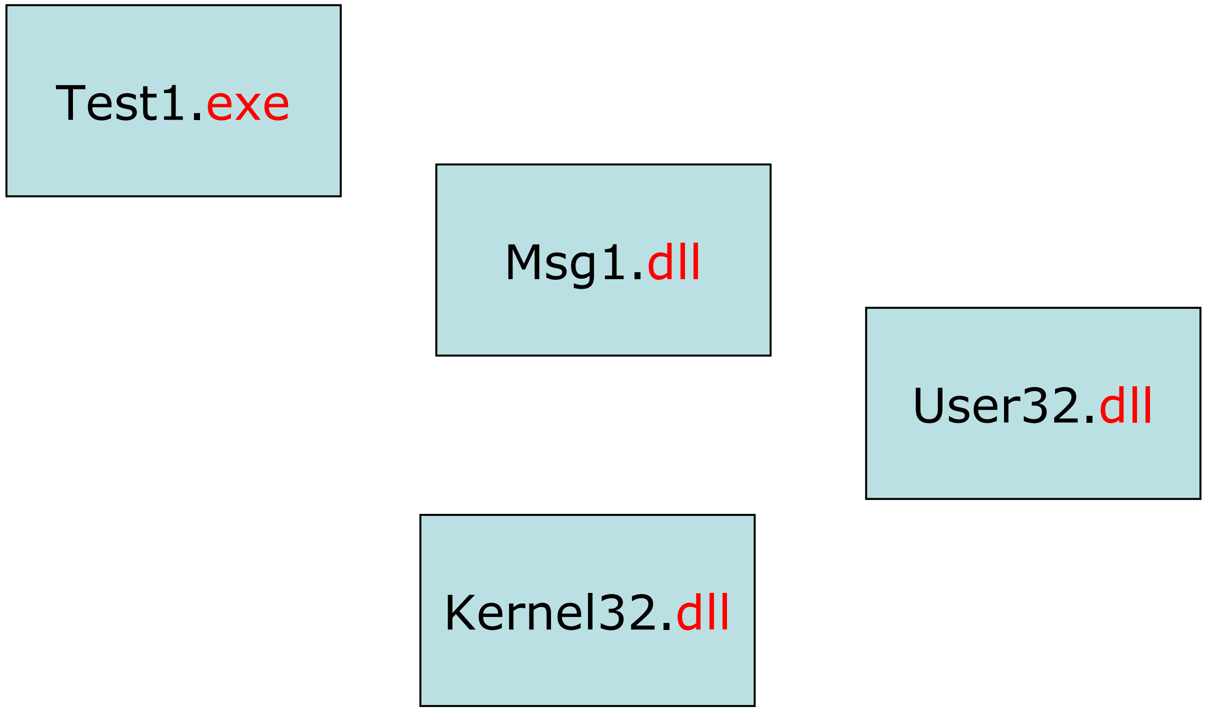


Module

- When an EXE file is loaded into memory, we call it as **module**
  - Also , someone call it as Image
- DLLs as well
  - DLL Module
- When an EXE file is loaded, many related DLLs are loaded into same process space, too

Then

- How these modules are arranged in the process space ?



Memory Layout

- Now, we take a look at memory layout of a process space
  - Note : It's important to understand memory layout before you learn virus infection mechanism

Memory Layout

- Ollydbg

Memory Layout

- 0x0000 0000 -> 0x7FFF FFFF
  - User space
  - User .exe file

- 0x8000 0000 -> 0xFFFF FFFF
  - Kernel space
  - Kernel core data structures
  - Can’t access it as users directly

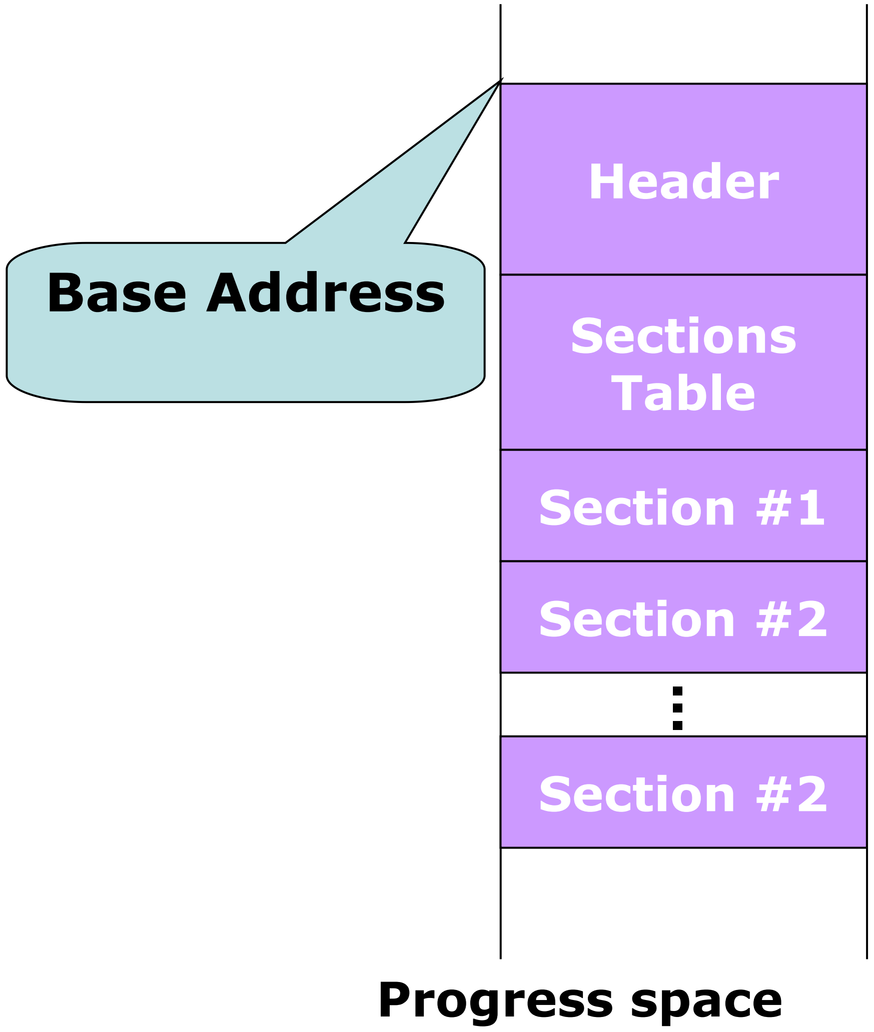
User Space

- Modules
  - Test1, test1.exe
  - Msg1, msg1.dll

- Stack of main thread
  - Every thread need one stack
  - Stack is created when EXE loading
    - Actually, when thread creating
  - Stack doesn’t appear in EXE file

Base Address

- The base address of an EXE image
  - Lowest address
- Base Address is important
  - If we know base address of some module, we can analyze the whole module and get all knowledge



Outline

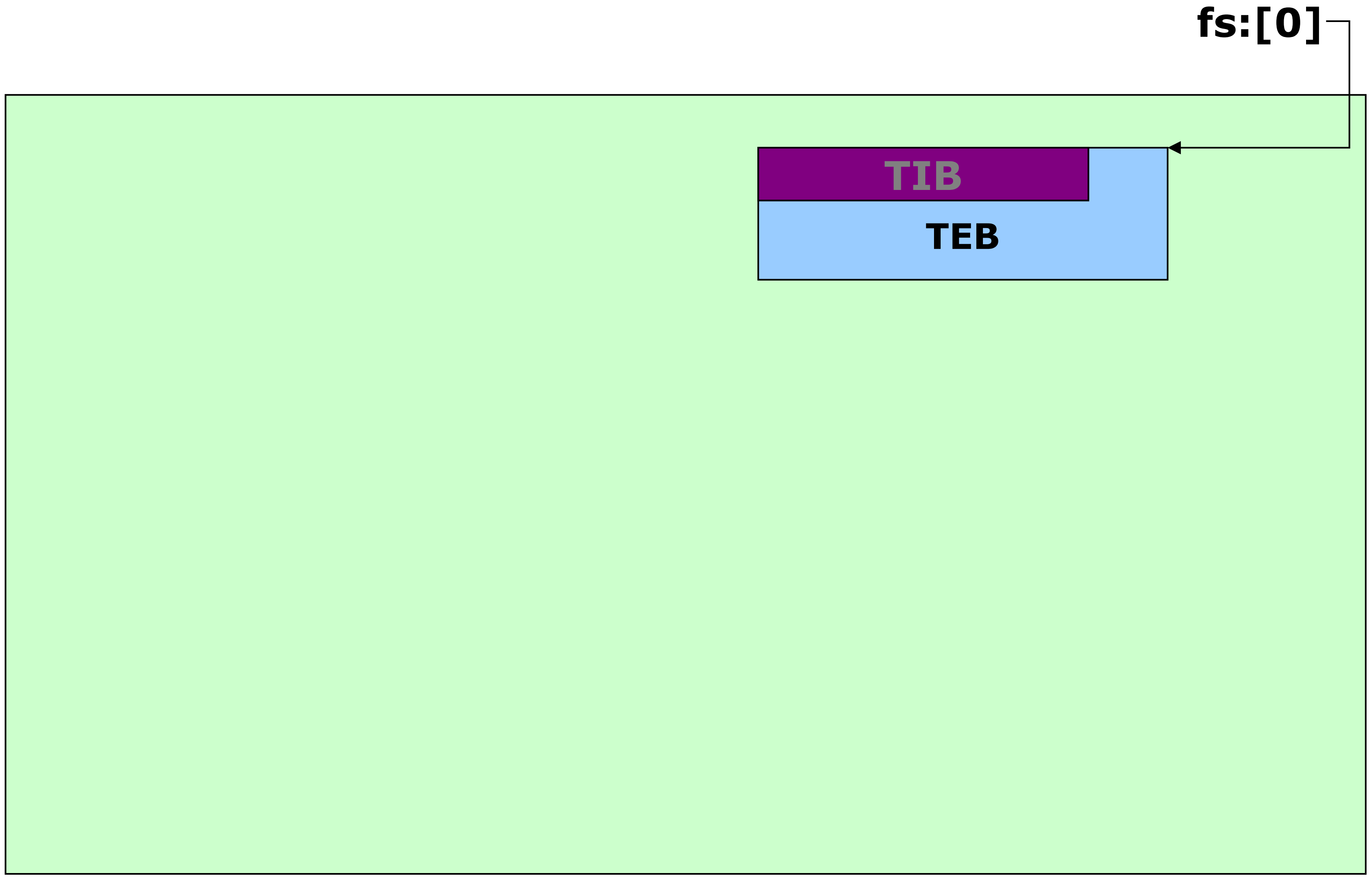
Address Space

Process Environment Block

SEH & VEH

Window Messages

Thread Environment Block (TEB)



Thread Environment Block (TEB)

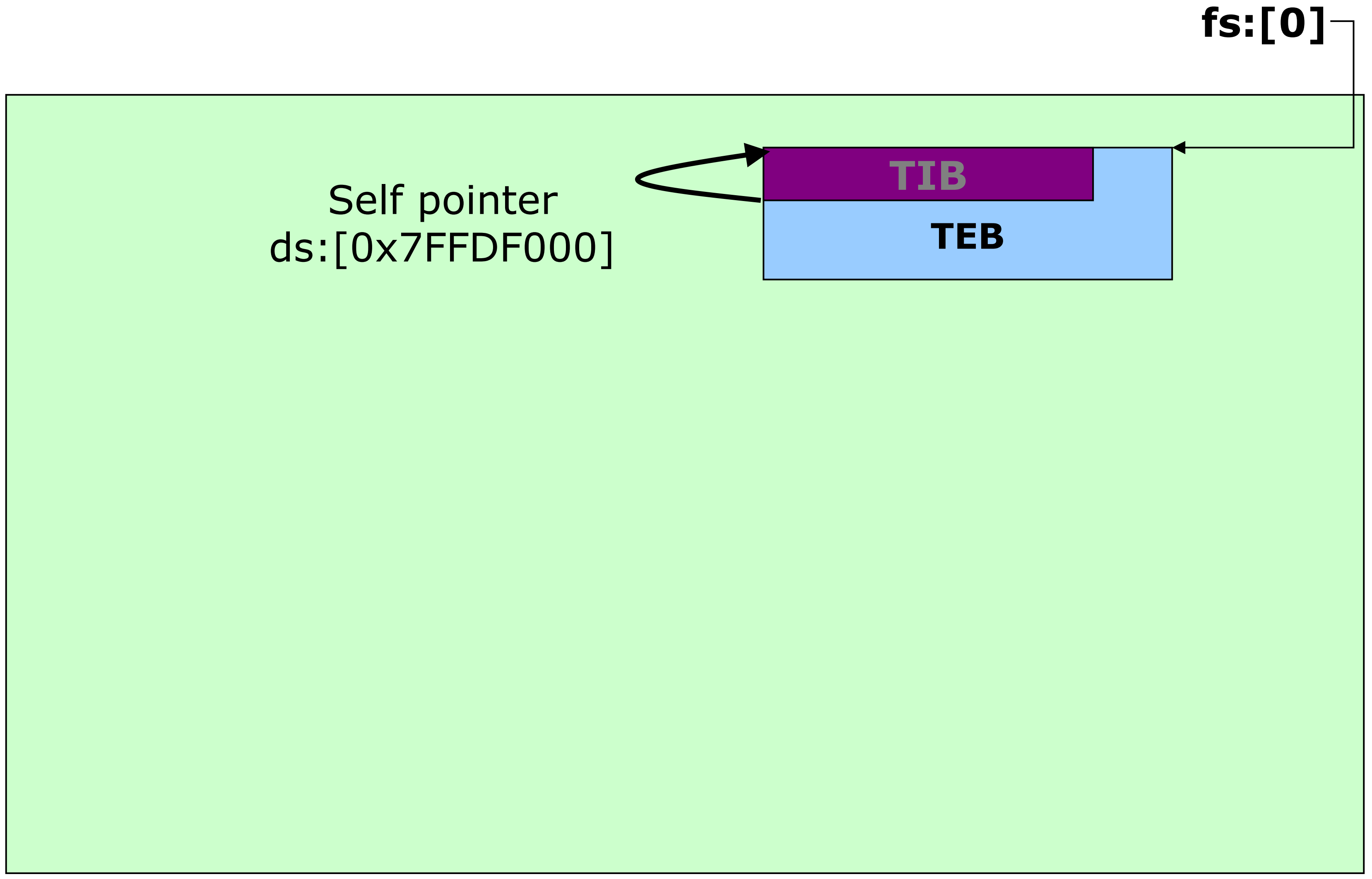
- TEB, including
  - Thread Info Block (TIB)
  - TLS Storage
  - PEB Address
    - (Process Environment Block)
  - . . .
- Undocumented structure definition
  - <winternl.h>

Thread Info Block (TIB)

- TIB, including
  - Exception List
  - Stack base
  - Stack limit
  - Self Address
- Defined in
  - <windows.h>

```
typedef struct _NT_TIB {
    struct _EXCEPTION_REGISTRATION_RECORD
*ExceptionList;
    PVOID StackBase;
    PVOID StackLimit;
    PVOID SubSystemTib;
    union {
        PVOID FiberData;
        DWORD Version;
    };
    PVOID ArbitraryUserPointer;
    struct _NT_TIB *Self;
} NT_TIB;
typedef NT_TIB *PNT_TIB;
```

Thread Environment Block (TEB)



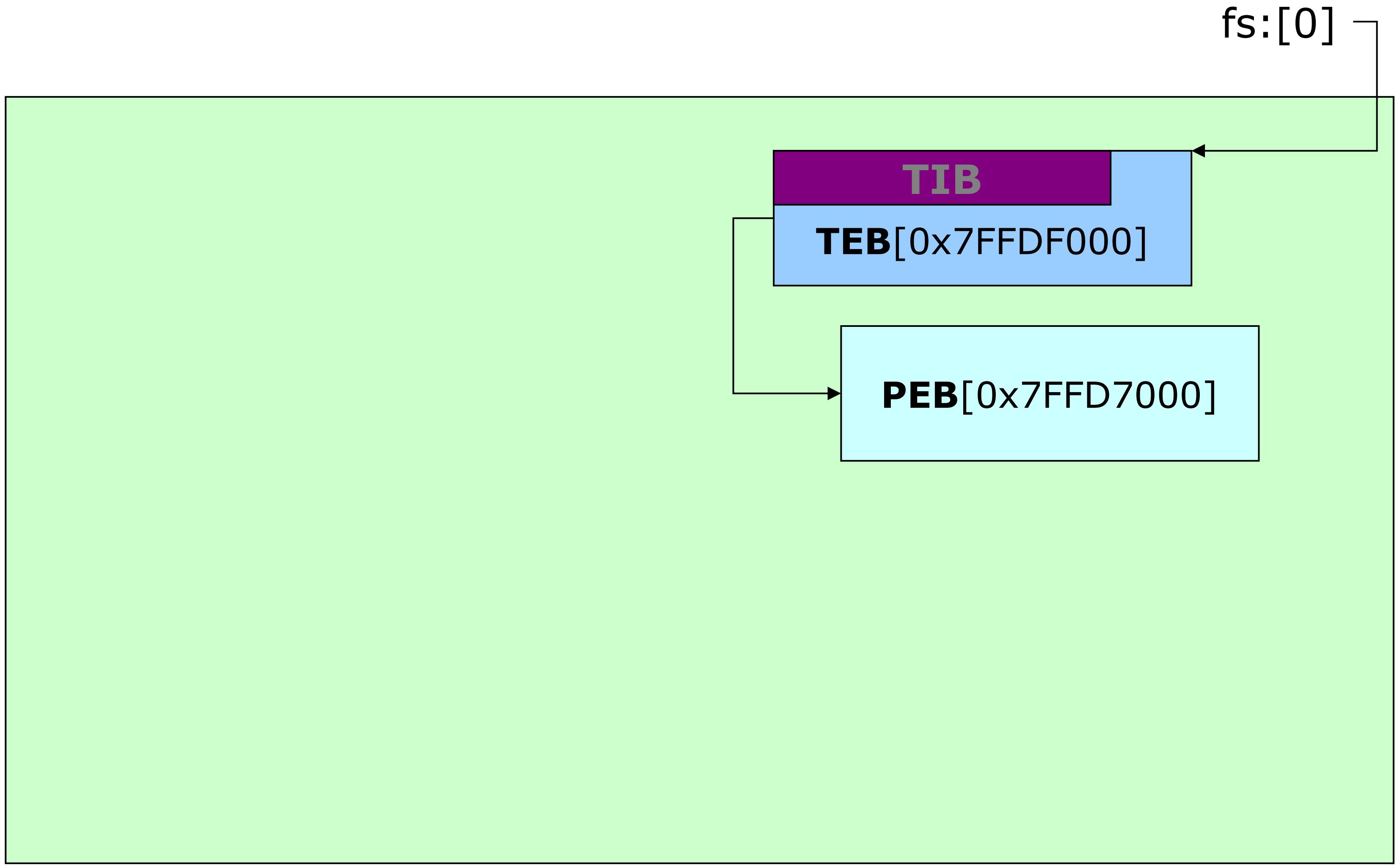
Demo

```
c:\demo> cl /c tebbase.c
 ...

c:\demo> link /dynamicbase:no tebbase.obj
 ...

c:\demo> tebbase
stack base: 0x00130000
teb base: 0x7ffdf000
```

Process Environment Block (PEB)



Process Environment Block (PEB)

- PEB, including
  - Image Base Address
  - Ldr
  - All User-mode parameters of the process
- Undocumented structure definition
  - <winternl.h>

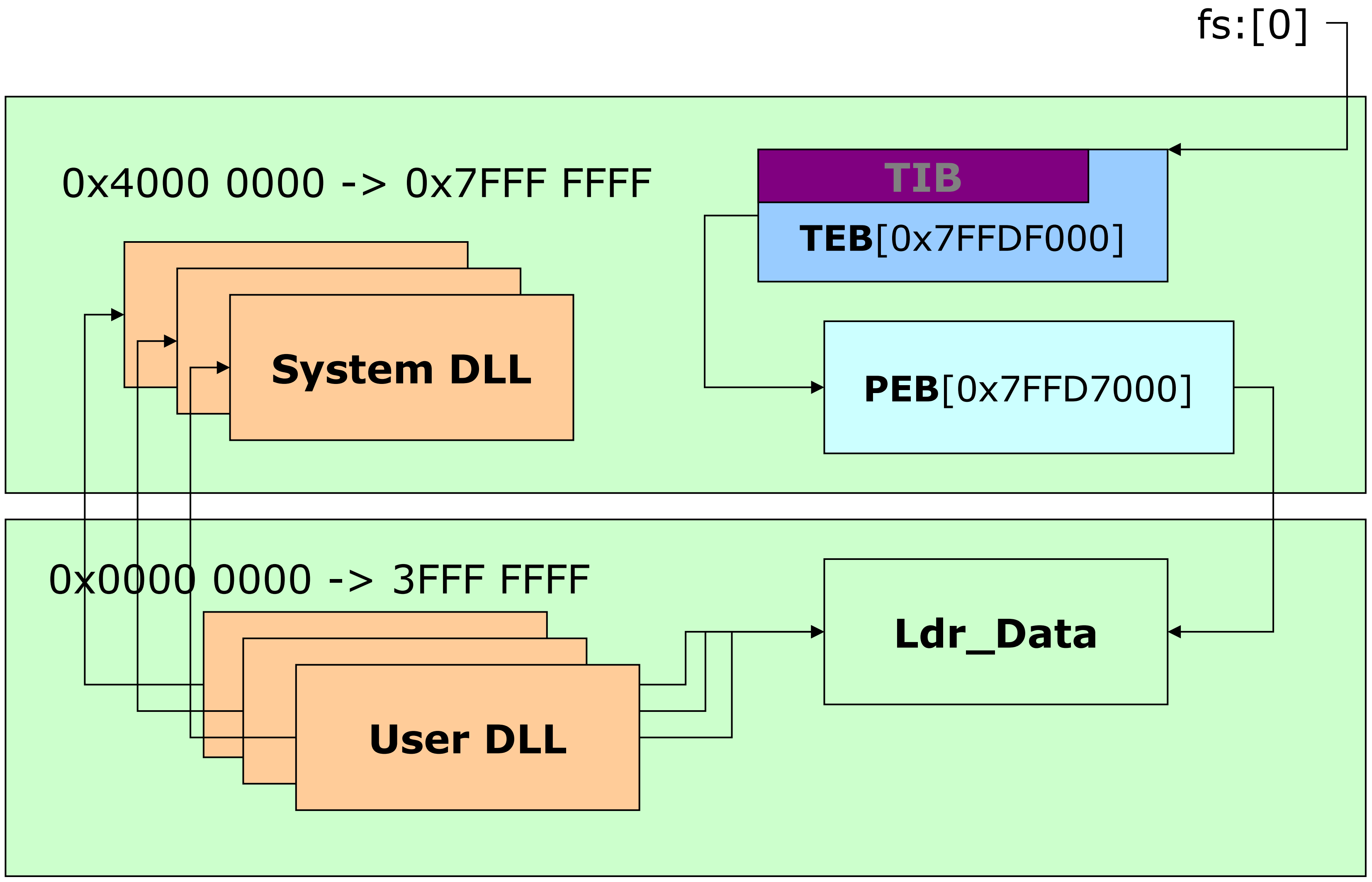
Demo

```
c:\demo> cl /c pebbase.c
 ...

c:\demo> link /dynamicbase:no pebbase.obj
 ...

c:\demo> pebbase
```

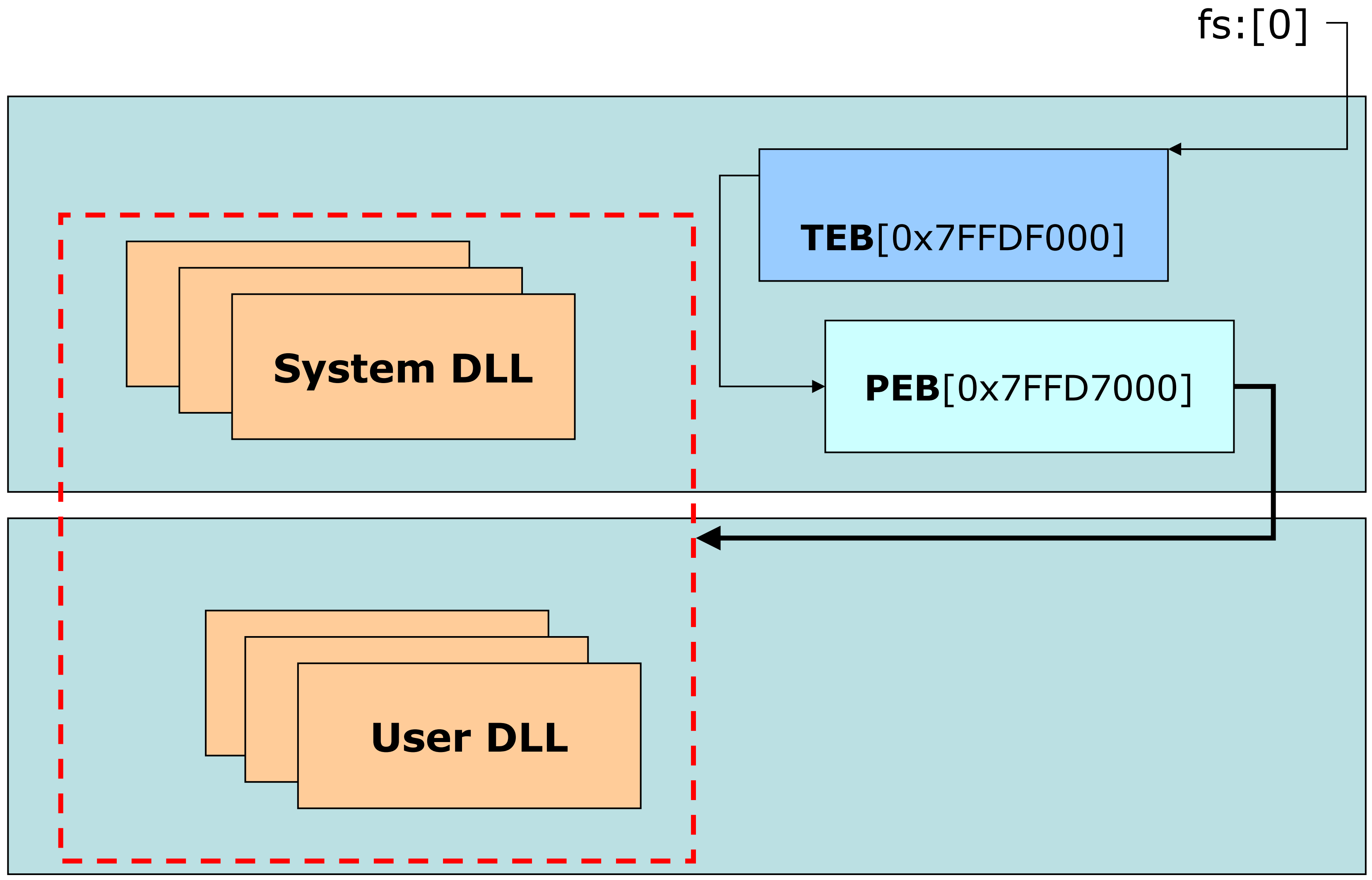
Ldr_DATA


fs:[0]
0x4000 0000 -> 0x7FFF FFFF
TIB
TEB[0x7FFDF000]
PEB[0x7FFD7000]
System DLL
0x0000 0000 -> 3FFF FFFF
Ldr_Data
User DLL

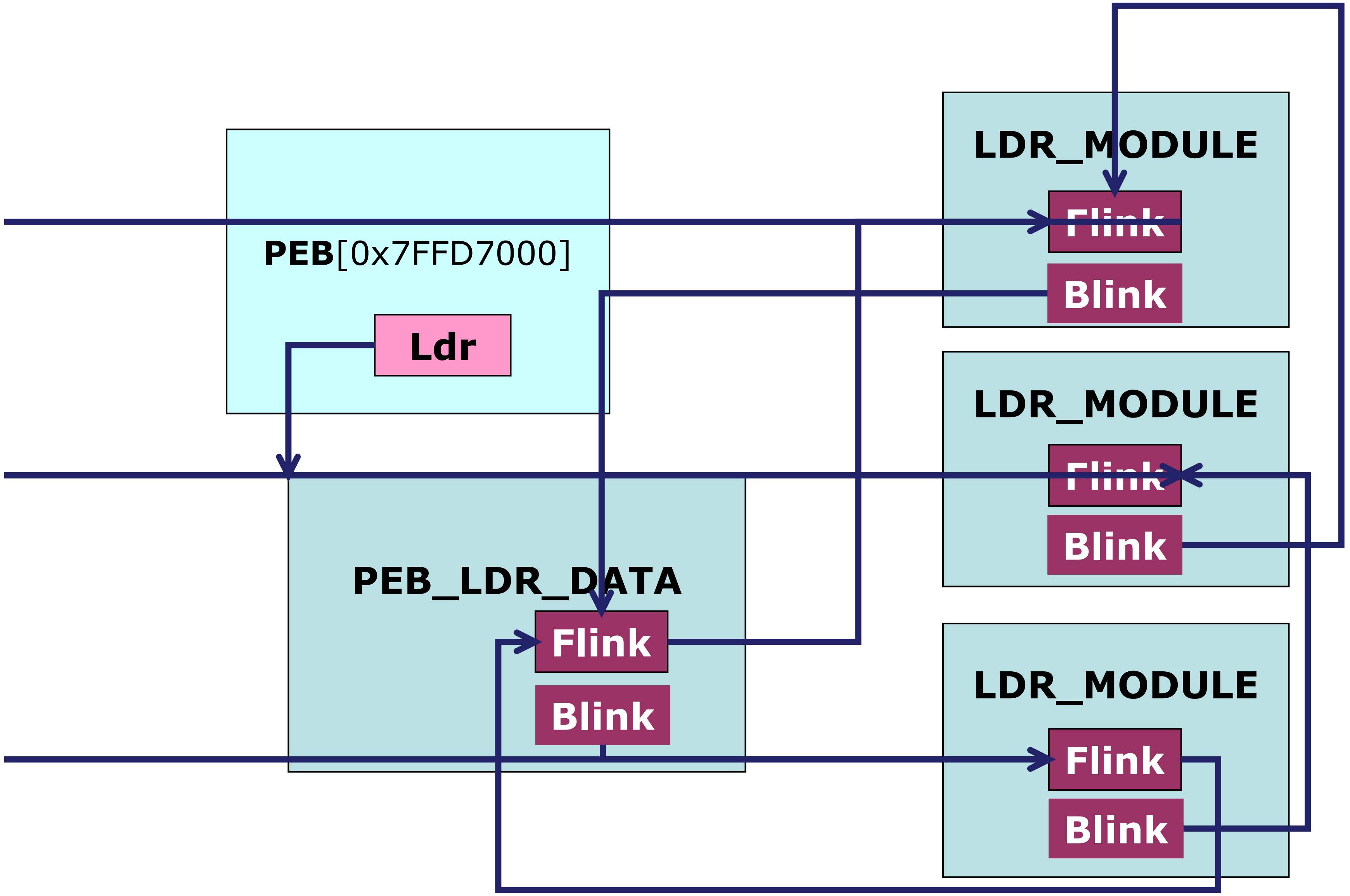
Cool!

- From PEB, we can get interesting data associated with the current process
  - Base Address
  - DLL modules loaded
  - EXE file name

Enumerate DLL Modules



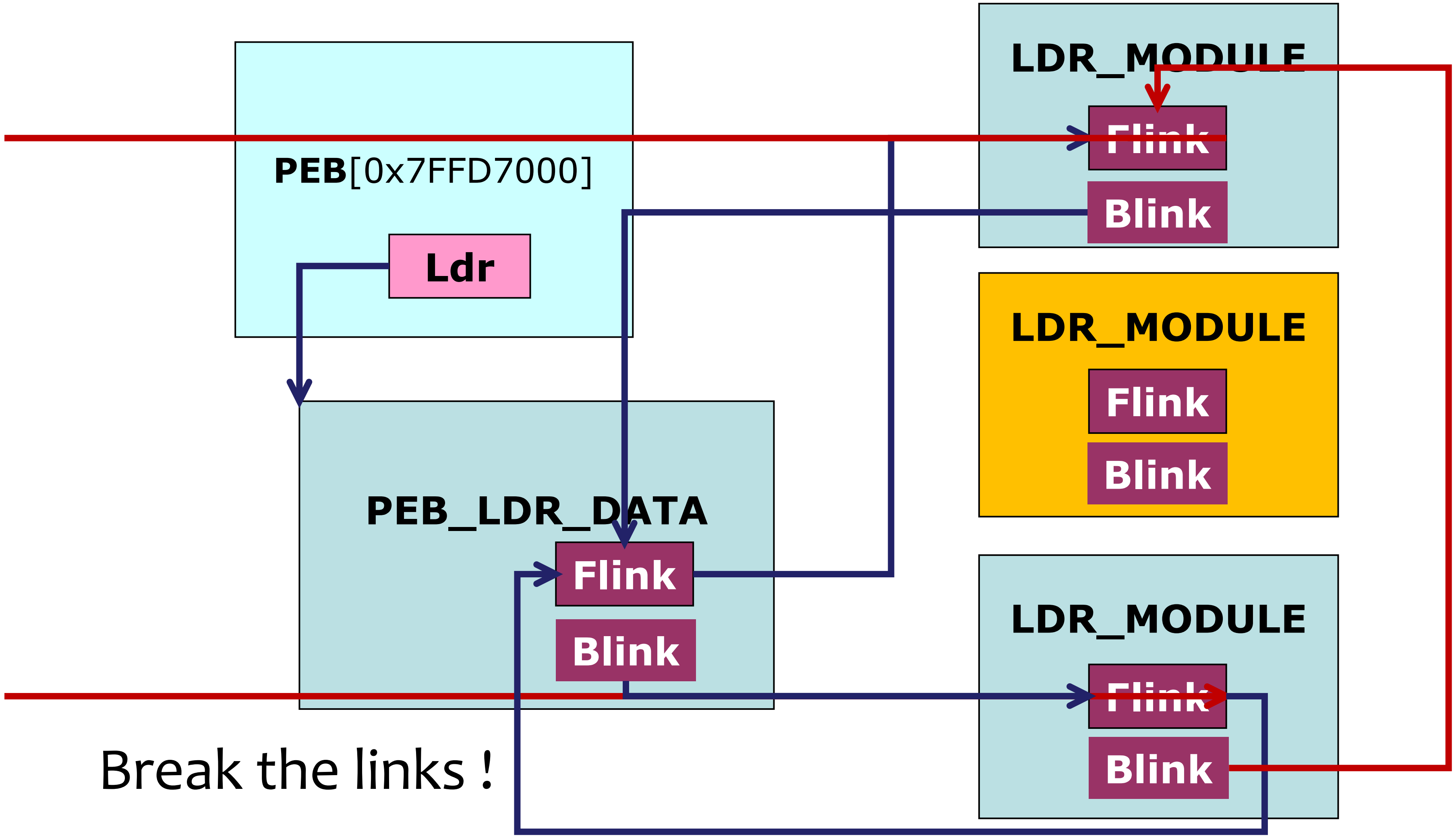
Ldr Data Table



Demo

- ldr.c

Hide Module



Break the links !

Demo

- ldr_hide.c

Outline

Process Space

Process Environment Block

SHE & VEH

Window Messages

Break point

- int 3
- 0xCC

Demo

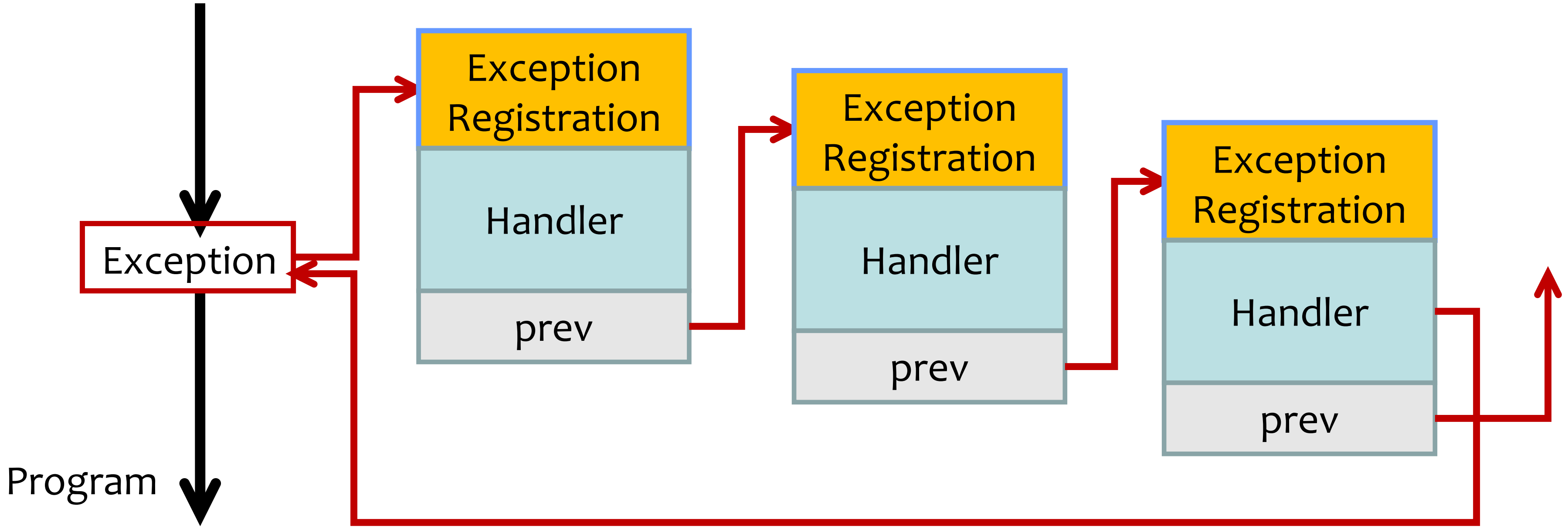
- seho.c

Exception

- Software caused
  - Breakpoints
  - Exceptions raised
  - …
- Hardware caused
  - Memory access
  - Divide zero
  - …

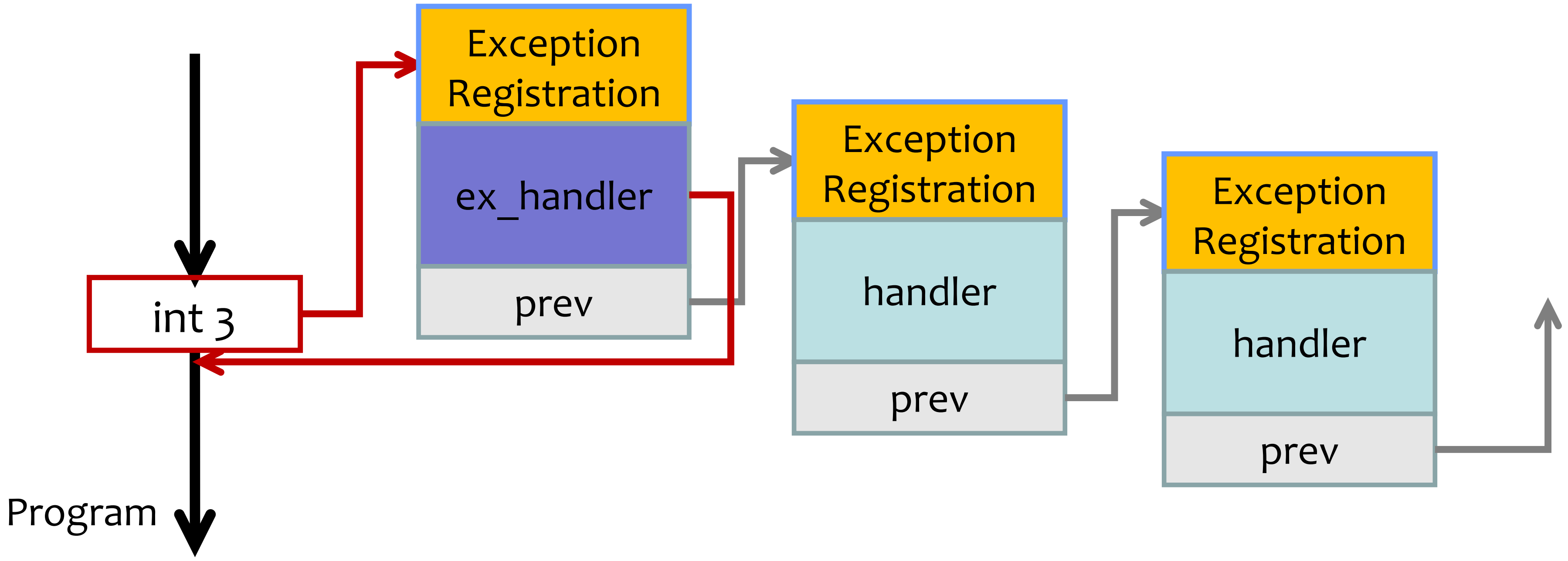
Structured Exception Handling

- SEH



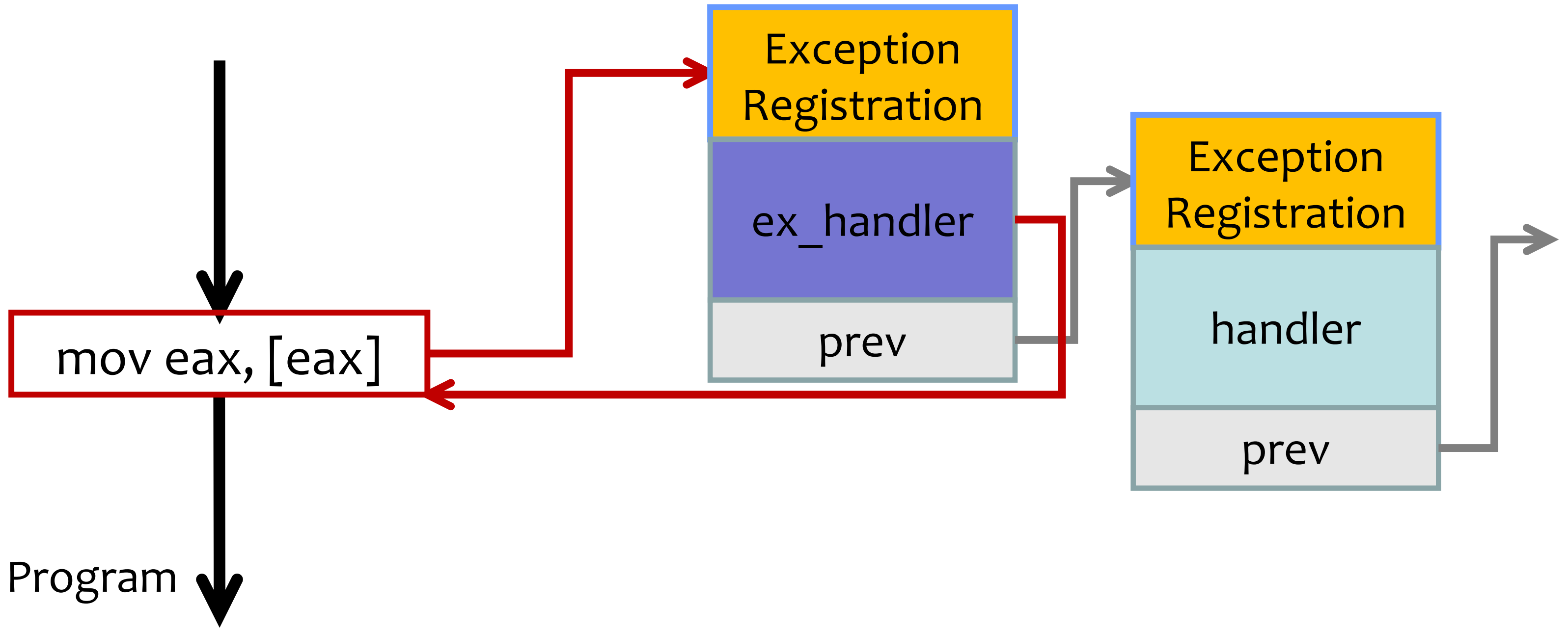
Demo

- seh1.c


Exception Registration
ex_handler
prev
Exception Registration
handler
prev
Exception Registration
handler
prev
int 3
Program

Demo

- seh2.c


mov eax, [eax]
Program
Exception Registration
ex_handler
prev
Exception Registration
handler
prev

Exception Handler

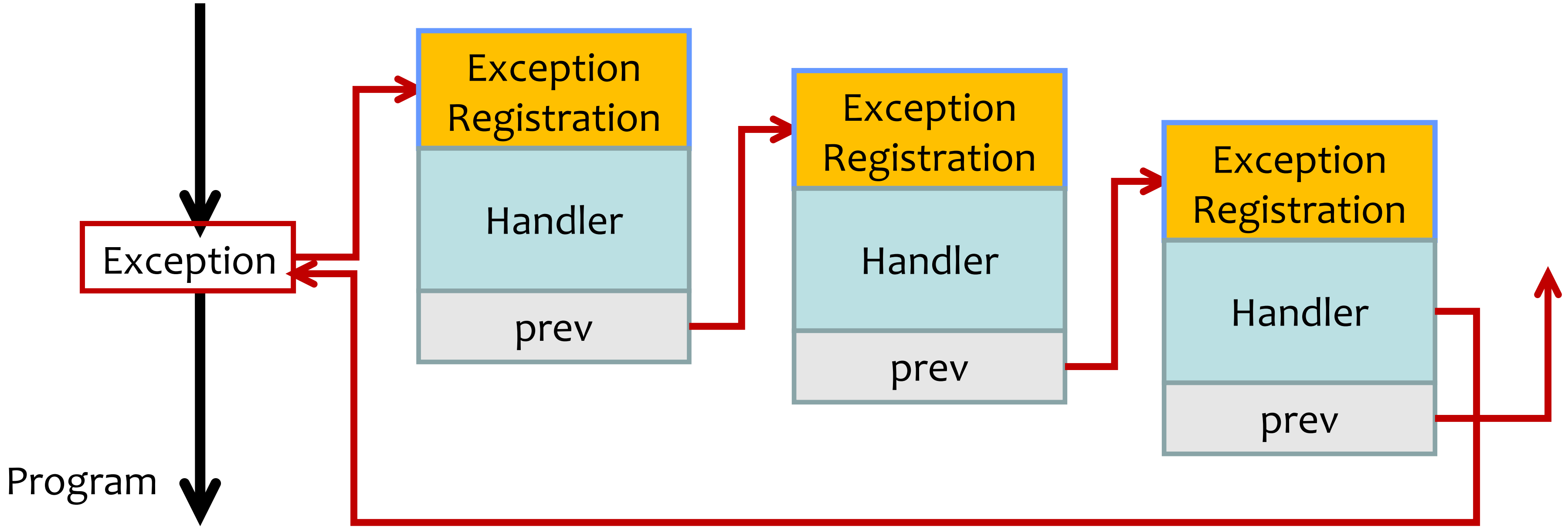
```
EXCEPTION_DISPOSITION __cdecl exception_handler(
    struct _EXCEPTION_RECORD *ExceptionRecord,
    void * EstablisherFrame,
    struct _CONTEXT *ContextRecord,
    void * DispatcherContext)
{
   ...
   return ExceptionContinueExecution;
}
```

Exception Record

```
typedef struct _EXCEPTION_RECORD {
    DWORD ExceptionCode;
    DWORD ExceptionFlags;
    struct _EXCEPTION_RECORD *ExceptionRecord;
    PVOID ExceptionAddress;
    DWORD NumberParameters;
    DWORD ExceptionInformation
                     [EXCEPTION_MAXIMUM_PARAMETERS];
} EXCEPTION_RECORD;
```

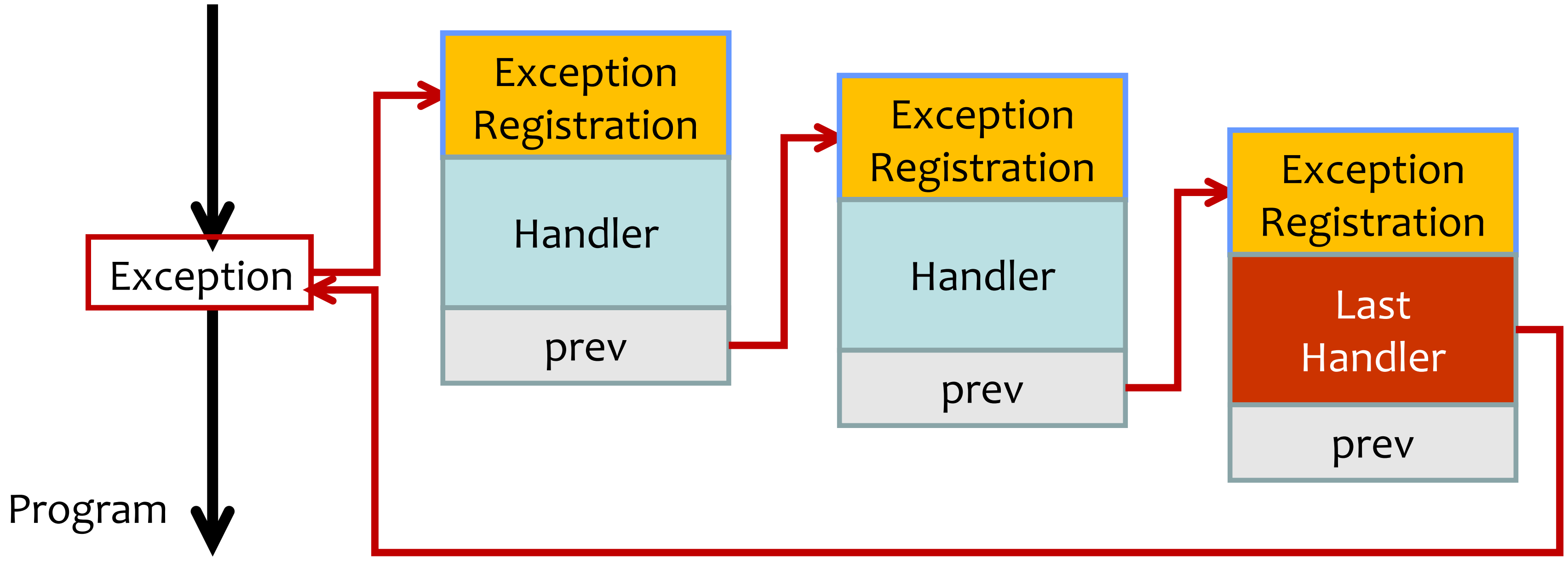
Structured Exception Handling

- SEH



SEH Hooking

- Install exception handler
  - SetUnhandledExceptionFilter() API



Vectored Exception Handling

- VEH
  - Process scope
- SEH
  - Thread scope

- VEH Installation (Hooking)
  - AddVectoredExceptionHandler()
  - RemoveVectoredExceptionHandler()

Outline

Process Space	Process Environment Block
SEH & VEH	Window Messages

Demo

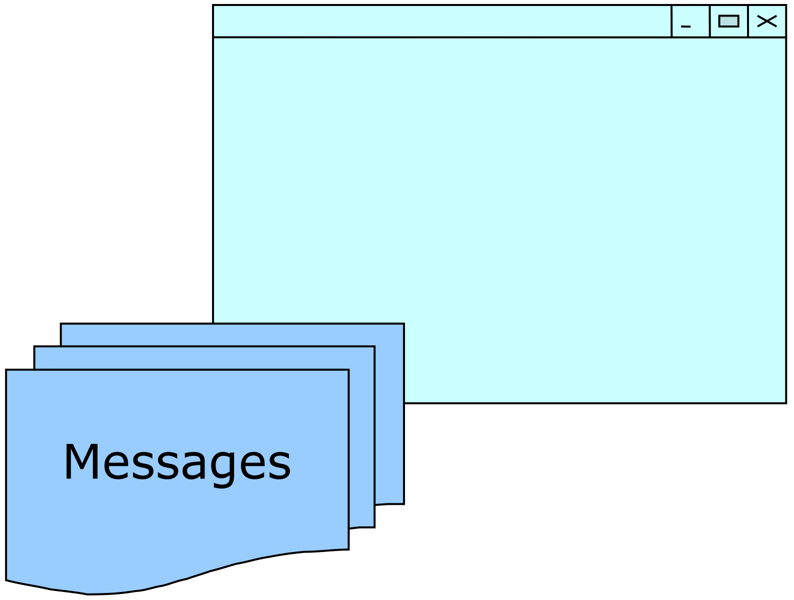
- wnd.c

Window Messages

- Window programs are event-based
  - Message
  - Message Queue

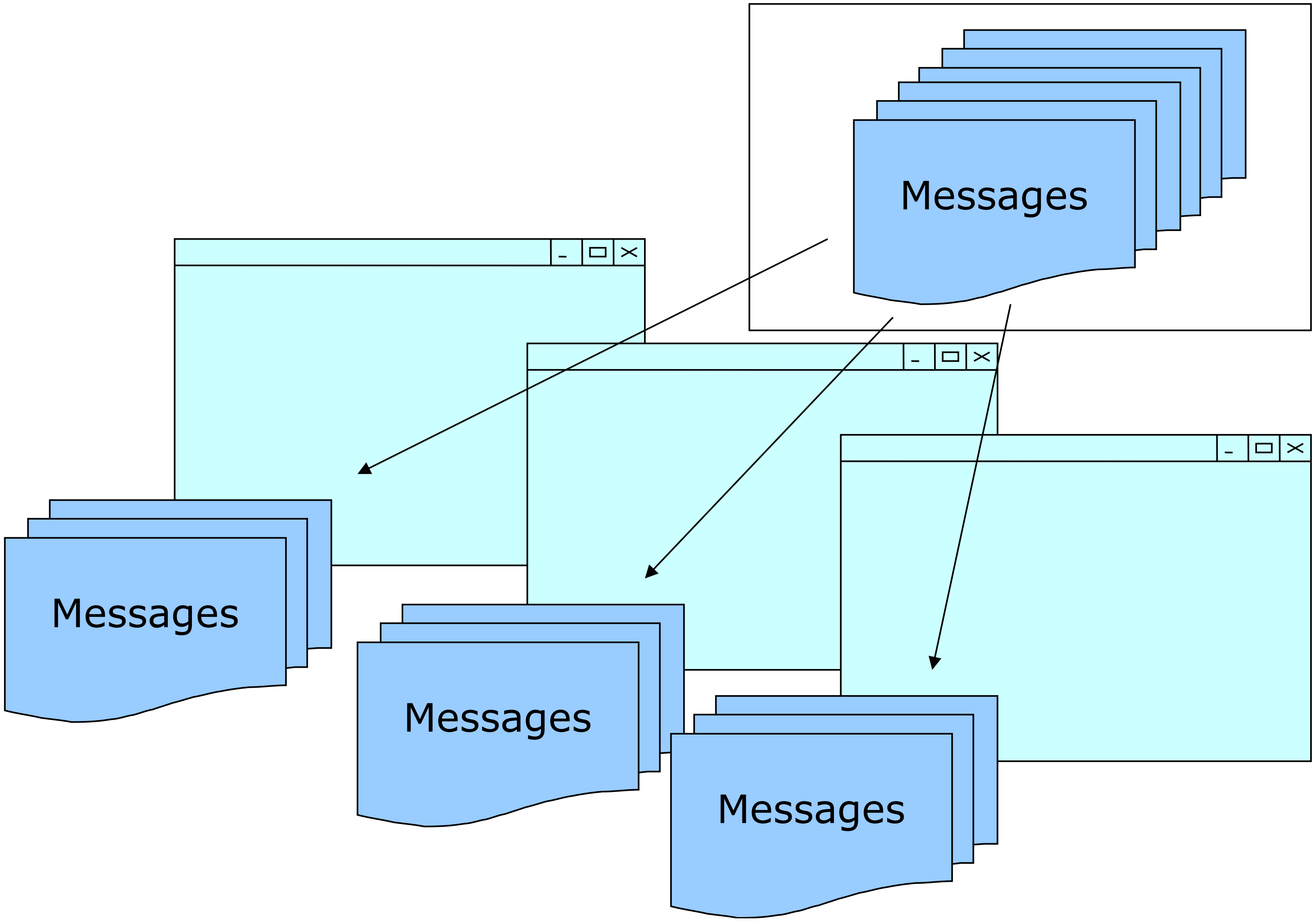
- Win32 GUI Framework
  - Windows Forms, MFC, WPF, VCL, GTK+, Qt, wxWidgets

Message Queue



One Message Queue per Thread

Message Queue



Window Messages Viewer

- Spy++
  - Visual Studio
- Winspector

Message Definition

- Messages
  - WM_CHAR
  - WM_KEYDOWN
  - WM_QUIT
  - WM_MOUSEMOVE
  - etc.

```
typedef struct tagMSG {
    HWND          hwnd;
    UINT          message;
    WPARAM        wParam;
    LPARAM        lParam;
    DWORD         time;
    POINT         pt;
} MSG
```

GetMessage() API

- User32.dll
- Get a message from the calling thread's message queue
- Return value
  - If it gets the WM_QUIT message, the return value is zero
  - If it fails, the return valus is -1
  - Otherwise, non-zero

```
BOOL WINAPI GetMessage( _Out_     LPMSG lpMsg,
                       _In_opt_  HWND hWnd,
                       _In_      UINT wMsgFilterMin,
                       _In_      UINT wMsgFilterMax );
```

DispatchMessage() API

- User32.dll
- Dispatches a message to a window procedure
- Return value
  - The value returned by the window procedure

```
LRESULT WINAPI DispatchMessage(
                _In_  const MSG *lpmsg
);
```

Window Procedure

- Callback function for each window
- Receive and processe all messages sent to the window

```
LRESULT CALLBACK WndProc(HWND hWnd,
                         UINT uMsg,
                         WPARAM wParam,
                         LPARAM lParam)
{
   switch(uMsg)
   {
     case ...

     Case ...
   }

   return DefWindowProc(hWnd, uMsg, wParam, lParam);
}
```

Default Window Procedure

- If a window procedure does not process a message, it must send the message back to the system for default processing
- Ensure that every message is processed

```
LRESULT WINAPI DefWindowProc(
                  _In_  HWND hWnd,
                  _In_  UINT Msg,
                  _In_  WPARAM wParam,
                  _In_  LPARAM lParam
);
```

TranslateMessage() API

- User32.dll
- BOOL TranslateMessage(CONST MSG*lpMsg)
  - Translates virtual-key messages into character messages
  - IME Translation
- Return:
  - If the message is not translated, return zero
  - If the message is WM_KEYDOWN, WM_KEYUP, return non-zero
- WM_KEYDOWN + WM_KEYUP ==>
  - WM_KEYDOWN, WM_CHAR, WM_KEYUP

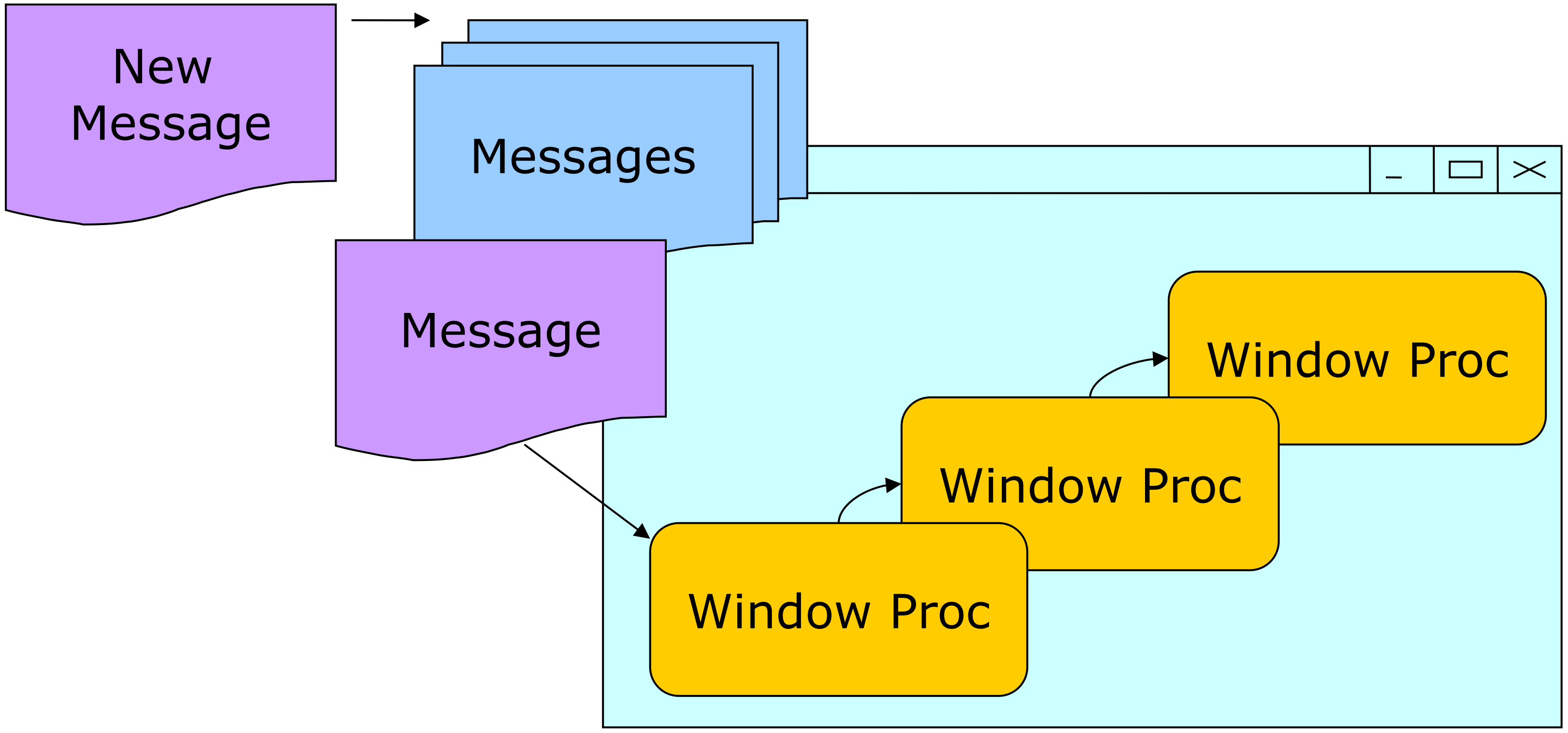
```
BOOL WINAPI TranslateMessage(
                    _In_  const MSG *lpMsg
);
```

Window Main Loop

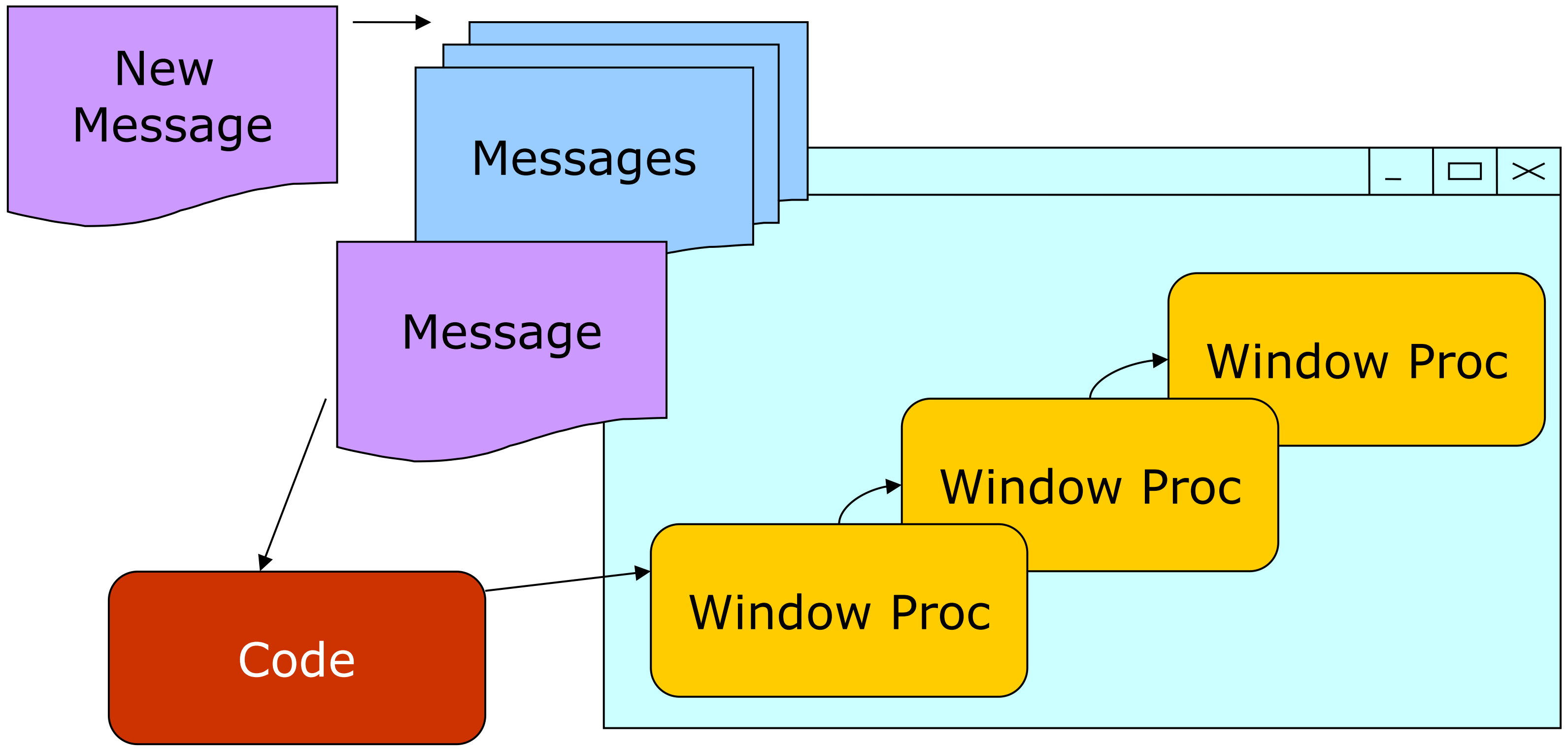
```
int WINAPI WinMain(HINSTANCE hinstance,
                  HINSTANCE hprevinstance,
                  LPSTR lpcmdline,
                  int ncmdshow)
{
    HWND hWnd;
    MSG msg;
    ...
    WNDCLASS  wndcls.lpfnWndProc = winproc;
    ...
    RegisterClass(&wndclass)
    hWnd=CreateWindowEx(...);
    ShowWindow(hWnd, ncmdshow);
    while(GetMessage(&msg, NULL, 0, 0)>0)
    {
      TranslateMessage(&msg);
      DispatchMessage(&msg);
    }
    return 0;
}
```

The Loop

WinProc Chain



Window Procedure Hooking



SetWindowLongA() API

- User32.dll
- Changes an attribute of the specified window
  - Window procedure (nIndex = **GWL_WNDPROC** (-4))
- Return:
  - return the previous value of the specified 32-bit integer
  - return zero if it fails

```
LONG WINAPI SetWindowLongA(
              _In_  HWND hWnd,
              _In_  int nIndex,
              _In_  LONG dwNewLong
);
```

CallWindowProc() API

- User32.dll
- Passes message information to the specified window procedure.
- Return:
  - the result of the window procedure

```
LRESULT WINAPI CallWindowProc(
  _In_  WNDPROC lpPrevWndFunc,
  _In_  HWND hWnd,
  _In_  UINT Msg,
  _In_  WPARAM wParam,
  _In_  LPARAM lParam
);
```

Any Question?