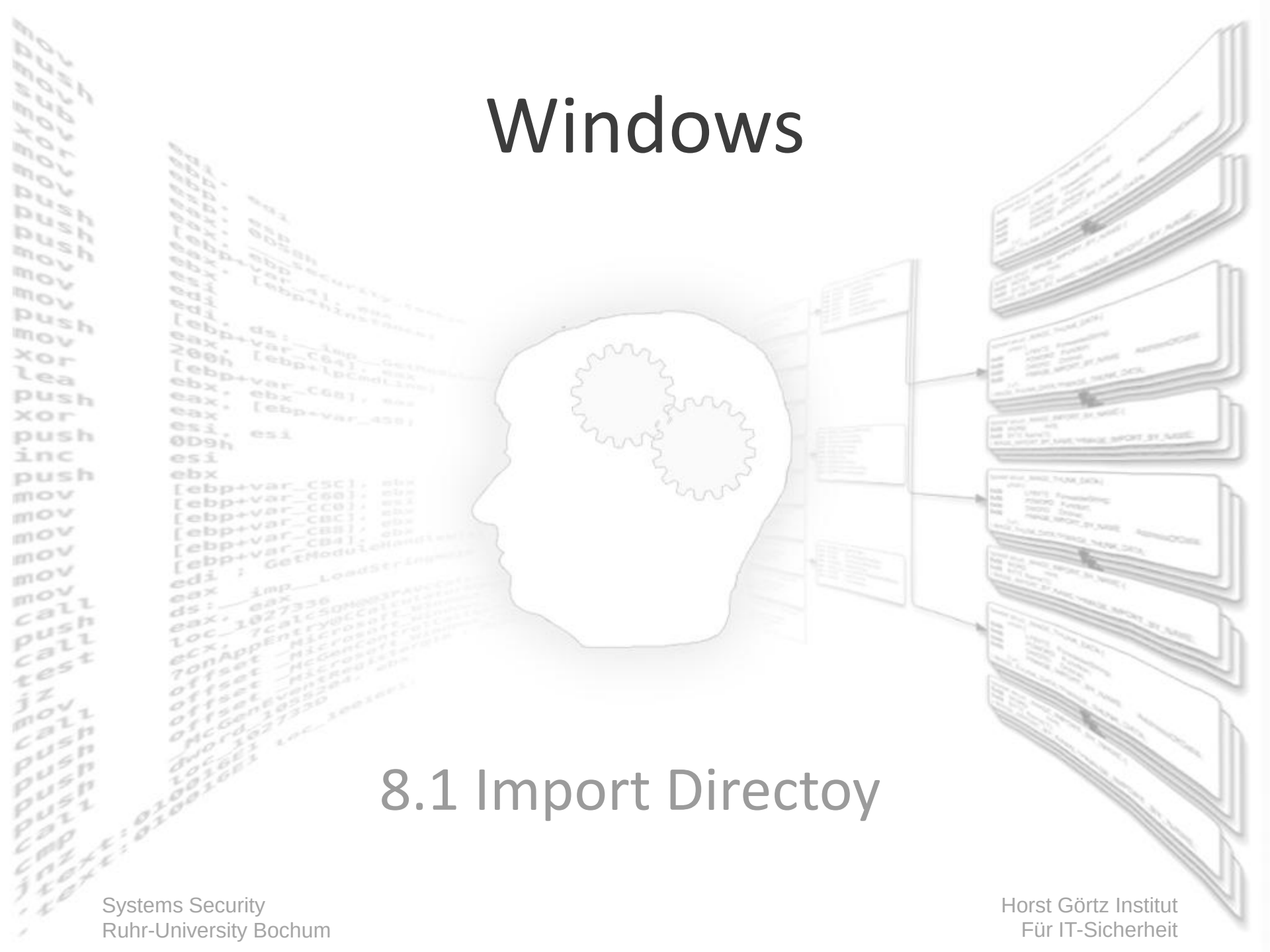
Program Analysis

Lecture 08: Windows II
Winter term 2011/2012

Dipl.-Inform. Carsten Willems

Windows

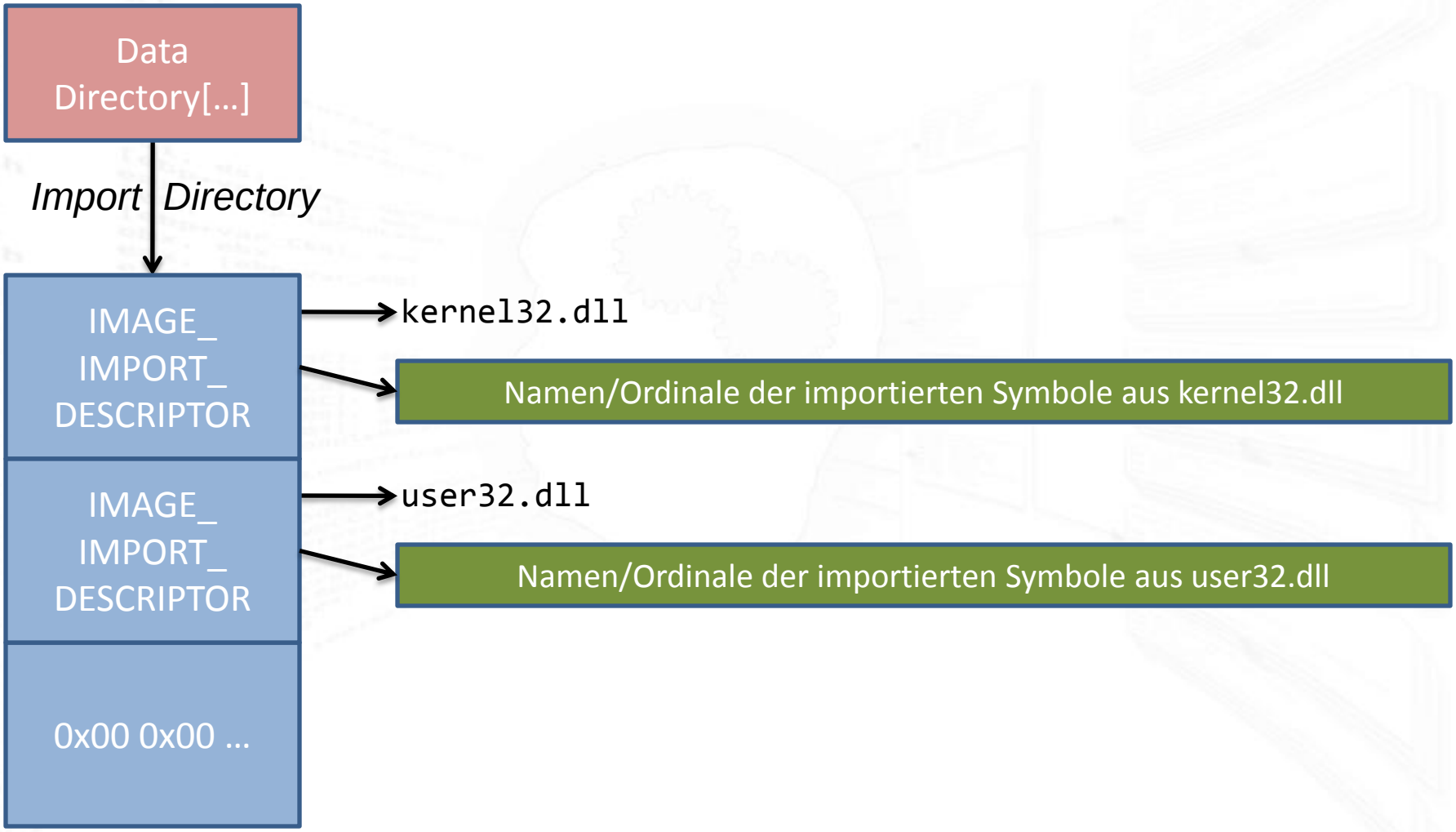


8.1 Import Directoy

Import Directory

- Anwendungen und Bibliotheken *importieren* Symbole aus (anderen) Bibliotheken
- *Import Directory* enthält dafür
  - Liste der zu importierenden DLLs / Funktionen / Daten
  - Einträge mit Namen/Ordinale der Symbole
- Windows-Loader
  - analysiert beim Laden eines Prozesses / Bibliothek rekursiv das Import Directory
    - Lädt alle referenzierten Bibliotheken
    - Ermittelt Adressen aller referenzierten Symbole

IMAGE_IMPORT_DIRECTORY

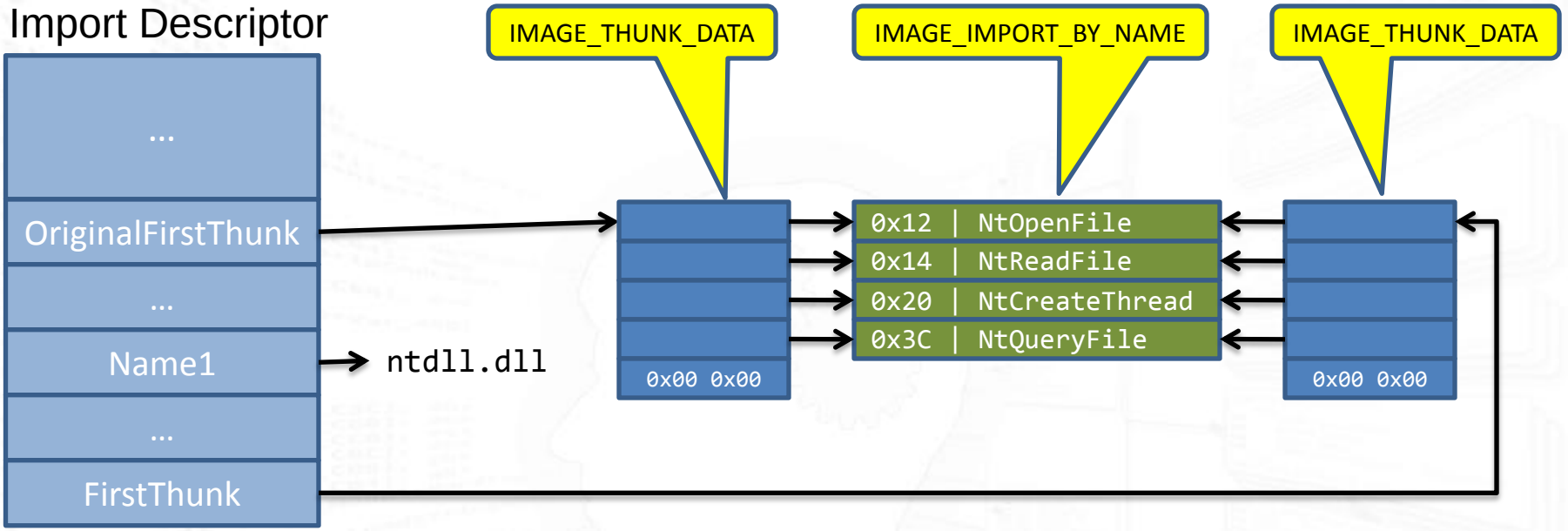


IMAGE_IMPORT_DESCRIPTOR

- *Import Directory* ist ein Array aus *Import Descriptors*
- Letzter Eintrag: 0x00, 0x00, 0x00, …

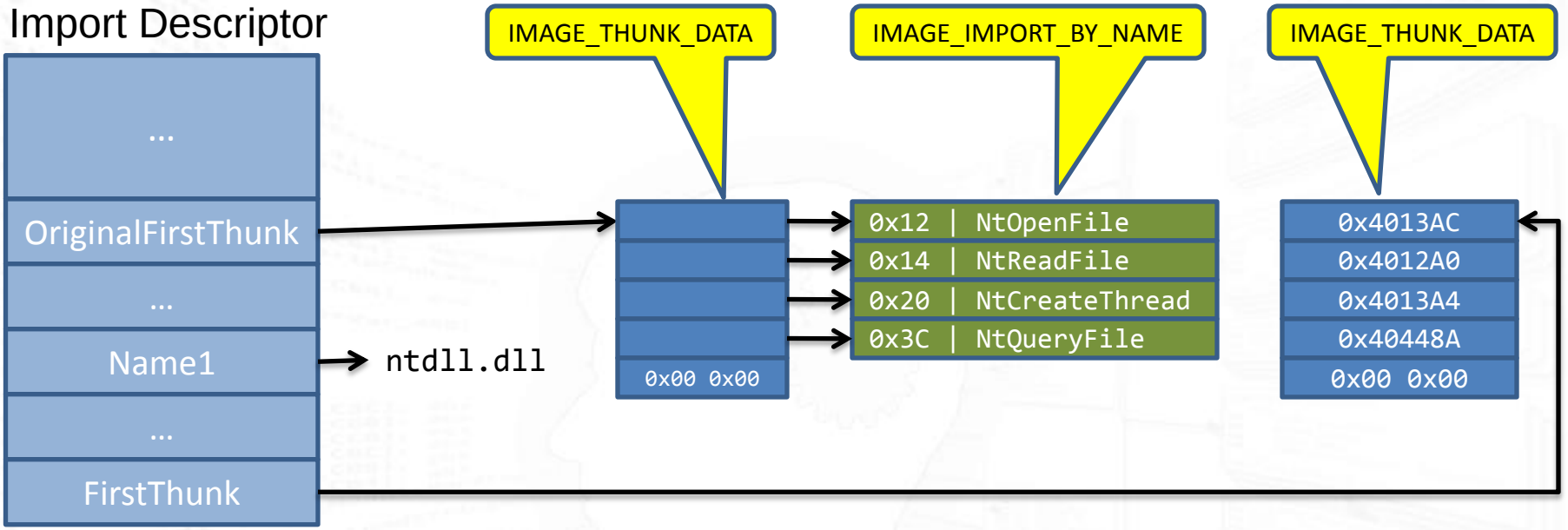
```
typedef struct _IMAGE_IMPORT_DESCRIPTOR
{
  union
  {
      DWORD   Characteristics;          // alt; 0 für Listenende
      DWORD   OriginalFirstThunk;       // RVA zur INT/ILT
  };
  DWORD   TimeDateStamp;                // für „bound imports“
  DWORD   ForwarderChain;               // alt
  DWORD   Name;                         // RVA zum Namen der DLL
  DWORD   FirstThunk;                   // RVA zur IAT
}
IMAGE_IMPORT_DESCRIPTOR, *PIMAGE_IMPORT_DESCRIPTOR;
```

Thunks in Datei



- OriginalFirstThunk und FirstThunk zeigen auf Array
- Vor dem Laden (=in Datei):
  - Zwei identische Arrays
  - Array-Elemente zeigen auf dieselben Namen / Ordinale

Thunks im Speicher



- Nach dem Laden (=im Speicher):
  - OriginalFirstThunk: unverändert
  - FirstThunk: Array enthält VA der jeweiligen Importe

(Original)FirstThunk

- Für jede importierte DLL gibt es
  - *OriginalFirstThunk*
    - Import Name/Locator Table (INT/ILT)
  - *FirstThunk*
    - Import Address Table (IAT)
- Vor dem Auflösen durch Loader:
  - Beiden zeigen auf Array aus *IMAGE_THUNK_DATA*
  - Jeder Array-Eintrag zeigt auf *Namen* oder *Ordinal*
- Nach dem Auflösen:
  - *FirstThunk*-Array enthält nun VA der importierten Symbole

Symbol-Informationen

```
typedef struct _IMAGE_THUNK_DATA32
{
   union
   {
     DWORD ForwarderString;
     DWORD Function;
     DWORD Ordinal;            // enthält entweder Ordinal oder
     DWORD AddressOfData;      // zeigt auf PIMAGE_IMPORT_BY_NAME
   }
}
IMAGE_THUNK_DATA32, *PIMAGE_THUNK_DATA32;

typedef struct _IMAGE_IMPORT_BY_NAME
{
   WORD      Hint;             // Um Laden zu beschleunigen
   BYTE      Name1;            // null-terminierter ASCII-String
}
IMAGE_IMPORT_BY_NAME, *PIMAGE_IMPORT_BY_NAME;
```

Import Address Table

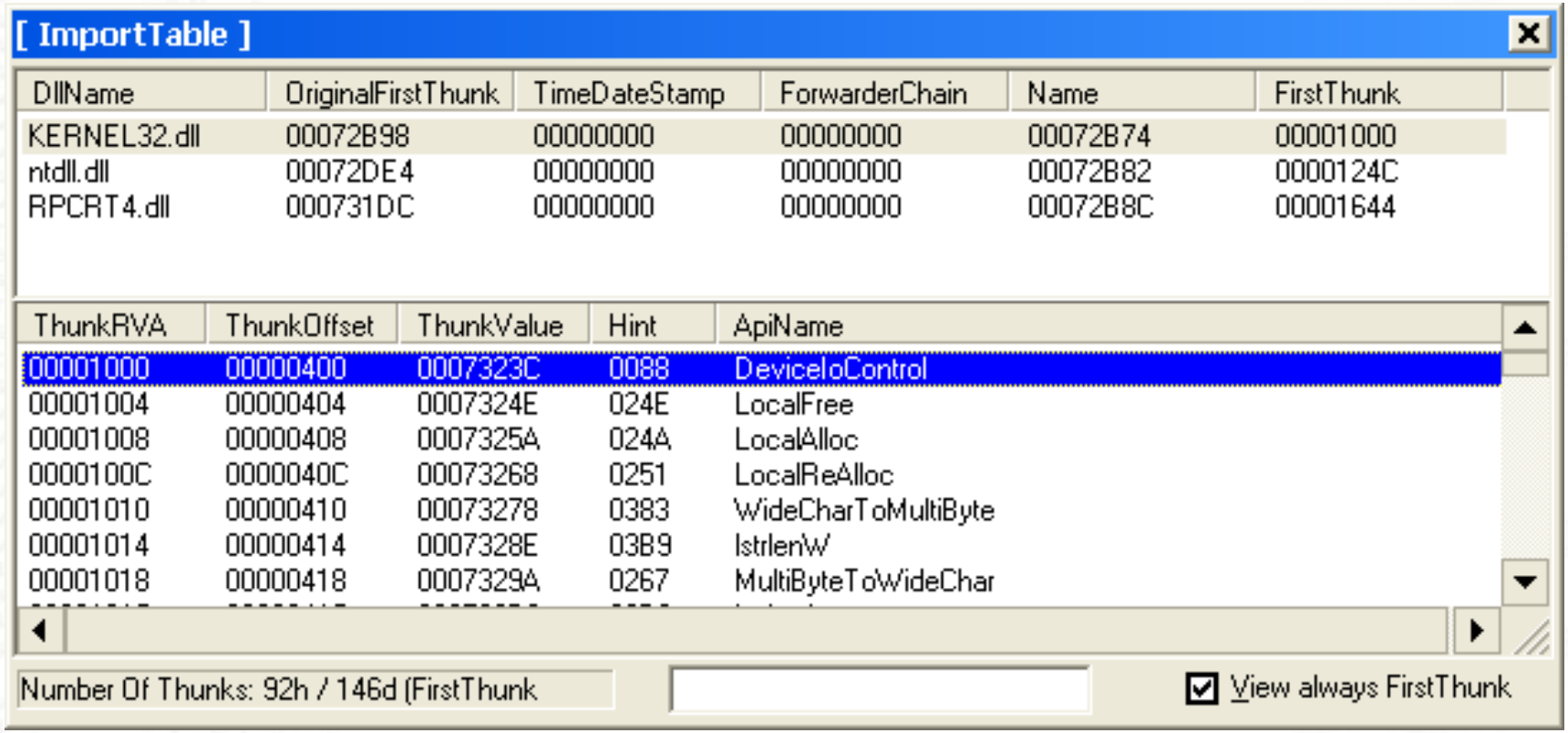
- Windows-Loader
  - Füllt *Import Address Table* mit VA der Importe
  - Ermittelt diese aus der *Export Address Table* der importierten DLL
- Verwendung der importierten Symbole
  - Direkt über IAT-Eintrag

```
call [IAT_Entry_CreateFileA]
```

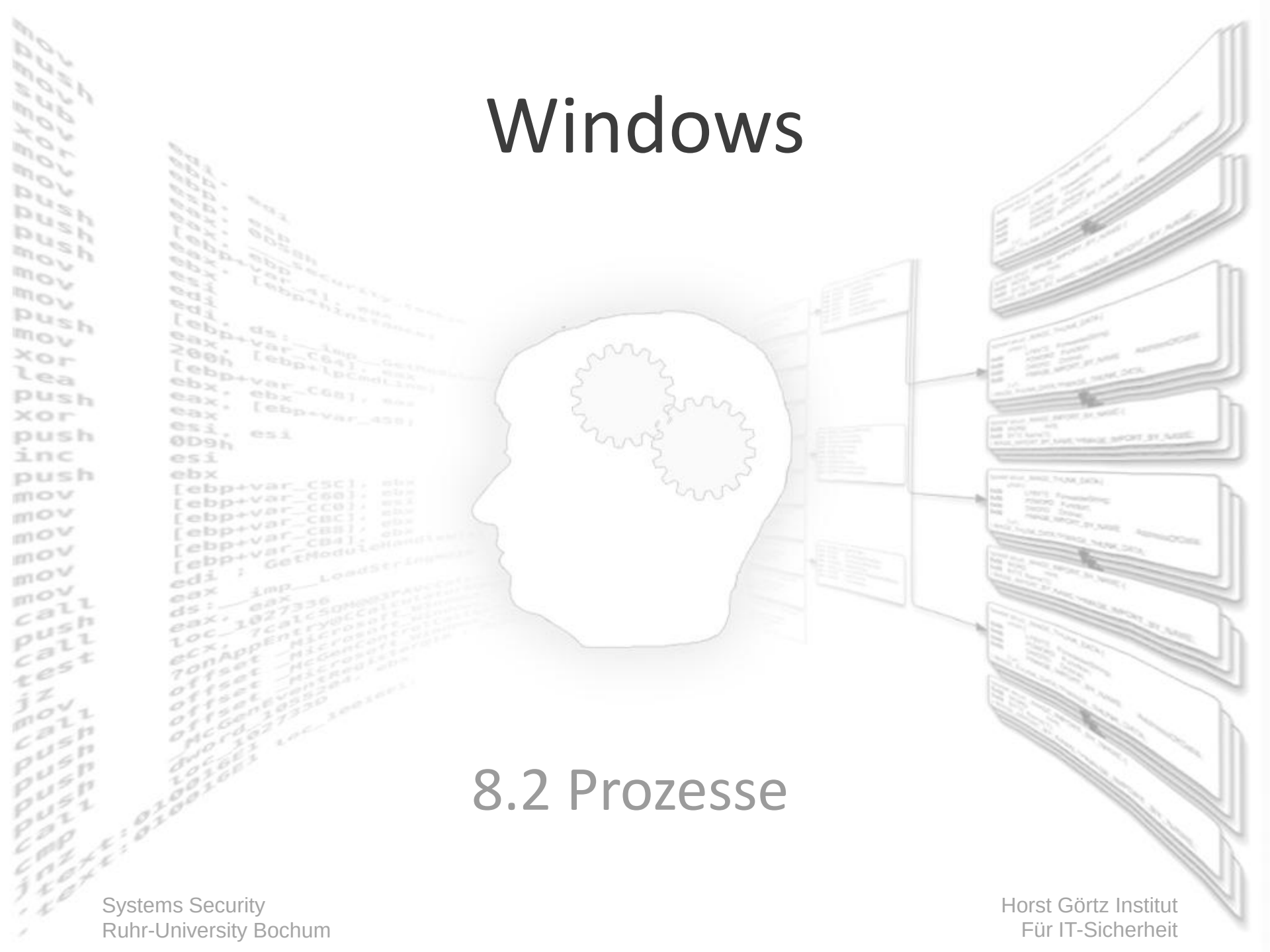
  - Oder über Jmp-Table

```
        call JmpTable_CreateFileA
        …
JmpTable_CreateFileA:
        jmp [IAT_Entry_CreateFileA]
```

IAT – advapi32.dll



Windows

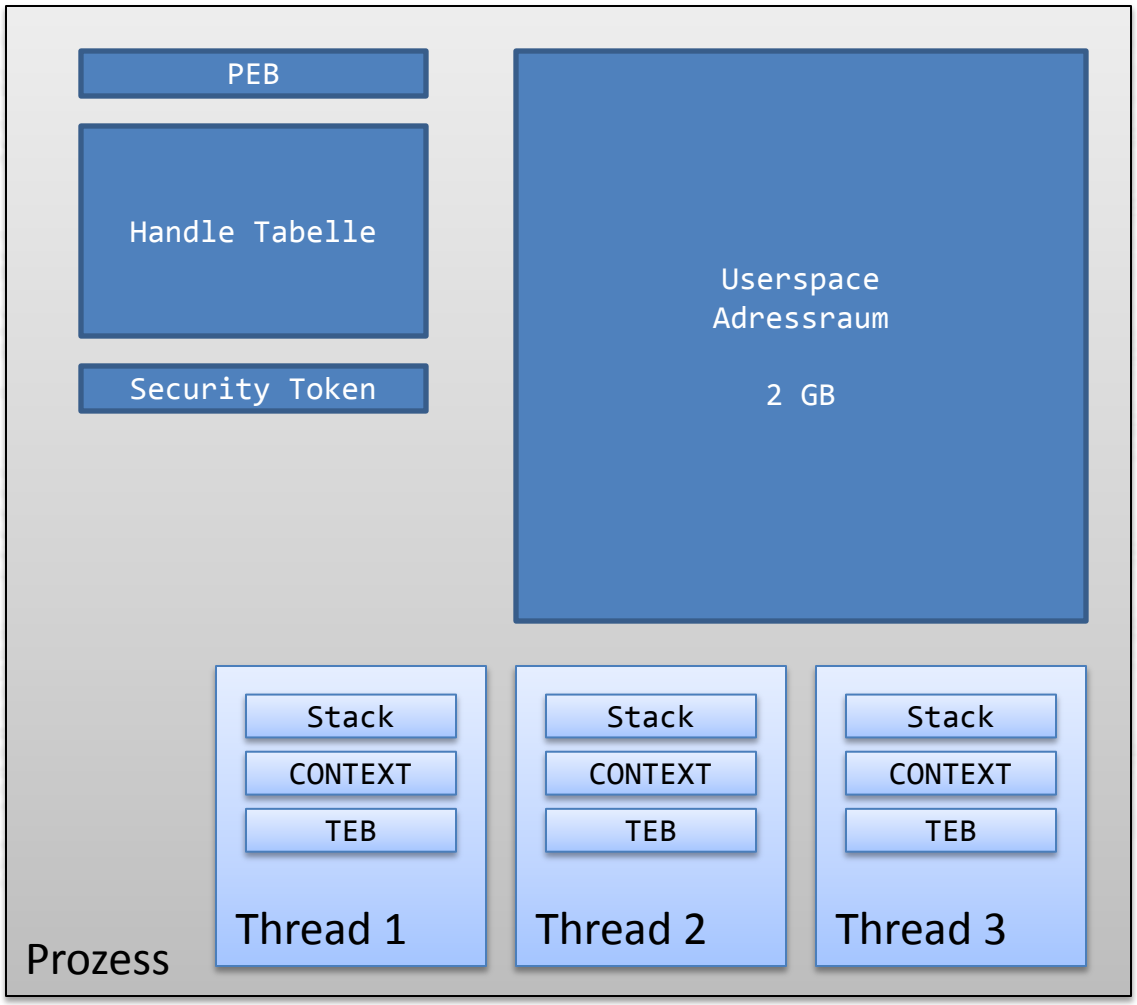


8.2 Prozesse

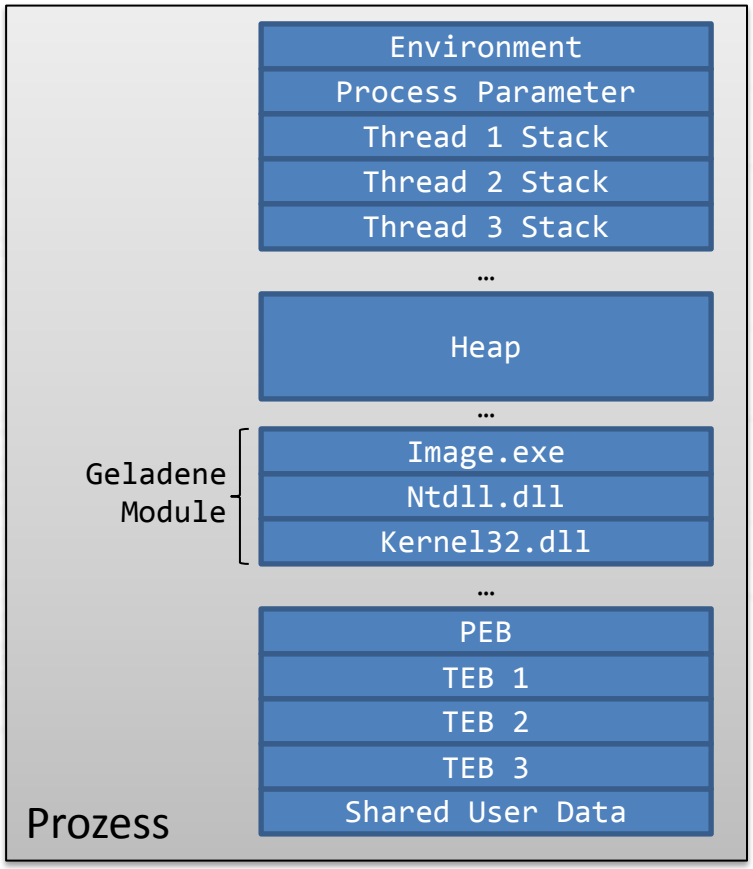
Prozess

- *Prozess* = Ablaufendes Programm
  - Eigener Userspace-Adressraum (2 GB)
  - Eigene Handle-Tabelle (kann von Elternprozess erben)
  - Process Environment Block (PEB)
  - Security Token
- Ein Prozess hat einen oder mehrere *Threads* mit je
  - Thread Environment Block (TEB)
  - Aktueller CONTEXT
    - Registerinhalte *(Instruktionspointer, EAX, …)*
  - Eigener Stack

Prozess Komponenten



Userspace Adressraum



API-Funktionen zur Prozesserstellung

- Zahlreiche API-Funktionen zur Prozesserstellung
  - *ShellExecuteA/W()*
  - *WinExec ()*
  - *CreateProcessA/W ()*
  - *CreateProcessUserA/W ()*
  - *CreateProcessWithTokenA/W ()*
- Im Endeffekt führen alle zu *CreateProcessInternalW()*

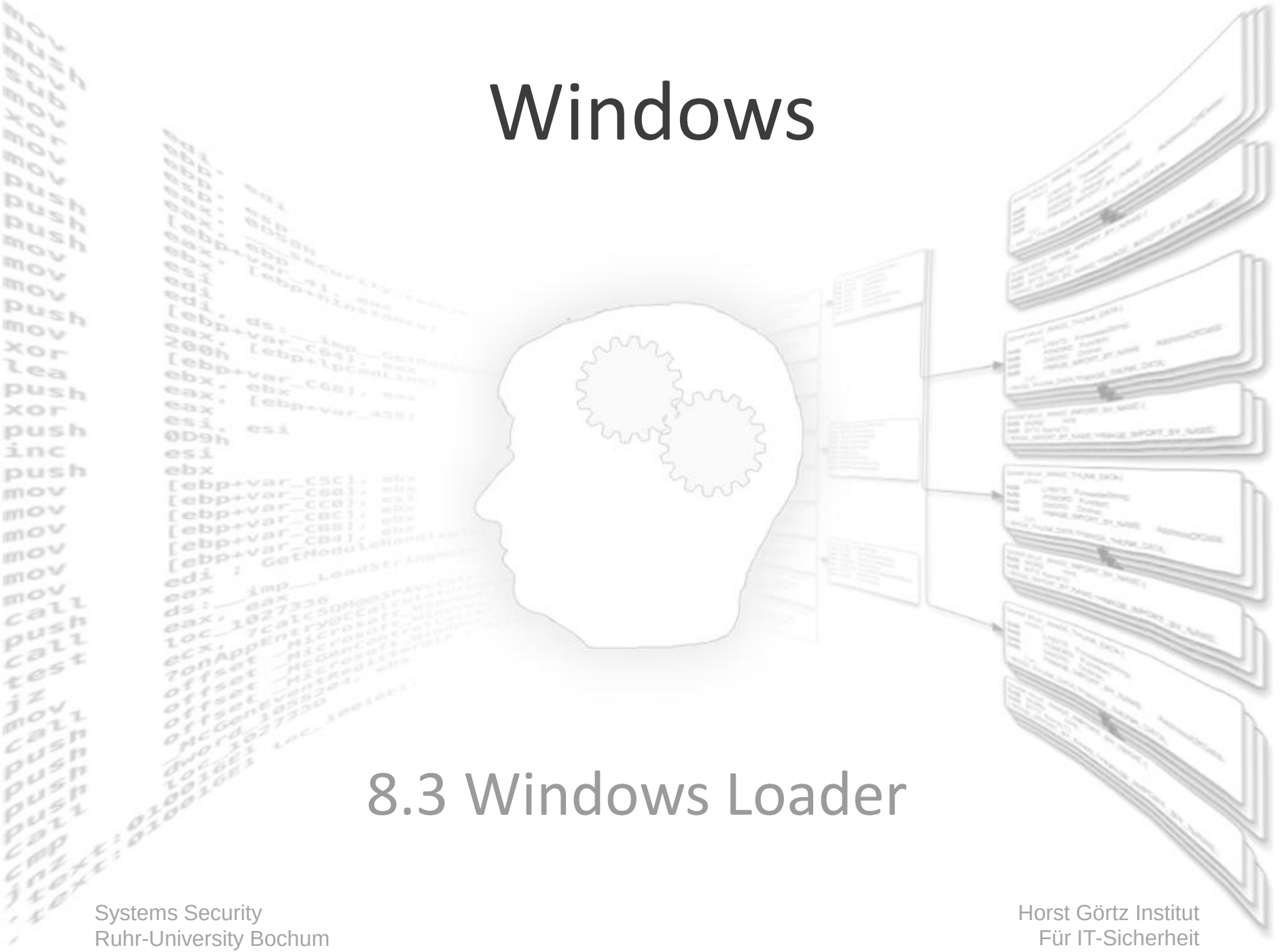
Prozesserstellung 1

- Im erstellenden Prozess:
  - Neuen Prozess-Adreßraum erstellen
  - Programm-Image in den Speicher laden
  - System DLL mappen (=ntdll.dll)
  - PEB erstellen
  - Hauptthread erstellen
    - Stack initialisieren
    - TEB erstellen
    - CONTEXT initialisieren
  - Windows-Subsystem informieren
  - Hauptthread starten

Prozesserstellung 2

- Im erstellten Prozess (im Hauptthread):
  - Loader-Funktion *LdrInitializeThunk()* über *Asynchronous Procedure Call* (APC) aufrufen
    - Rekursiv DLLs importieren
    - Symbole auflösen
    - IAT füllen
  - Danach *BaseProcessStartThunk()* aufrufen
    - ruft *Entrypoint (EP)* des Images auf (=> main)
- Prozess läuft, bis alle Threads beendet sind
- Vorzeitige Beendigung möglich
  - *TerminateProcess()* oder *ExitProcess()*

Windows



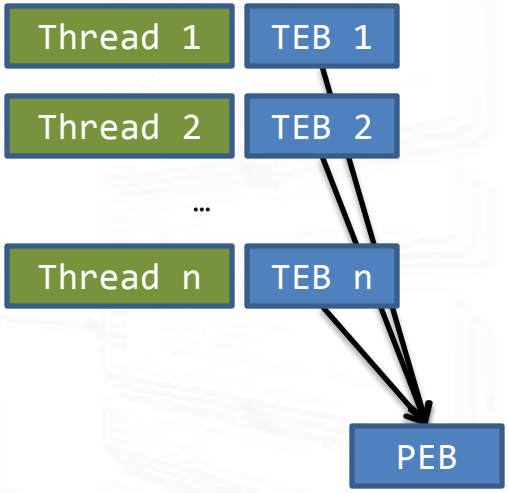
8.3 Windows Loader

Windows Loader

- Implementiert in *ntdll.dll*
- Aufgaben:
  - Initialisierung eines neuen Prozesses
  - (Rekursives) Nachladen von DLLs
  - Auffinden von importierten Funktionen
  - Füllen der *Import Address Table* (IAT)
- Eng verknüpft mit *Process Environment Block* (PEB)

TEB / PEB

- *Process Environment Block* (PEB)
  - Informationen über laufenden Prozess
    - Loader Informationen
    - Prozess Informationen
- Thread Environment Block (TEB)
  - Informationen über aktuellen Thread
    - Stackadresse, -größe, TLS-Infos
    - Exception Handler
- PEB / TEB selten direkt von Anwendungen benutzt
  - FS:[0] zeigt auf TEB, FS:[0x30] zeigt auf PEB



TEB

```
+0x000 NtTib                          : _NT_TIB
+0x01c EnvironmentPointer             : Ptr32 Void
+0x020 ClientId                       : _CLIENT_ID
+0x028 ActiveRpcHandle                : Ptr32 Void
+0x02c ThreadLocalStoragePointer      : Ptr32 Void
+0x030 ProcessEnvironmentBlock        : Ptr32 _PEB
+0x034 LastErrorValue                 : Uint4B
+0x038 CountOfOwnedCriticalSections   : Uint4B
+0x03c CsrClientThread                : Ptr32 Void
+0x040 Win32ThreadInfo                : Ptr32 Void
+0x044 User32Reserved                 : [26] Uint4B
+0x0ac UserReserved                   : [5] Uint4B
+0x0c0 WOW32Reserved                  : Ptr32 Void
+0x0c4 CurrentLocale                  : Uint4B
+0x0c8 FpSoftwareStatusRegister       : Uint4B
+0x0cc SystemReserved1                : [54] Ptr32 Void
+0x1a4 ExceptionCode                  : Int4B
+0x1a8 ActivationContextStack         : _ACTIVATION_CONTEXT_STACK
+0x1bc SpareBytes1                    : [24] UChar
+0x1d4 GdiTebBatch                    : _GDI_TEB_BATCH
+0x6b4 RealClientId                   : _CLIENT_ID
+0x6bc GdiCachedProcessHandle         : Ptr32 Void
+0x6c0 GdiClientPID                   : Uint4B
+0x6c4 GdiClientTID                   : Uint4B
+0x6c8 GdiThreadLocalInfo             : Ptr32 Void
+0x6cc Win32ClientInfo                : [62] Uint4B
+0x7c4 glDispatchTable                : [233] Ptr32 Void
+0xb68 glReserved1                    : [29] Uint4B
+0xbdc glReserved2                    : Ptr32 Void
+0xbe0 glSectionInfo                  : Ptr32 Void
+0xbe4 glSection                      : Ptr32 Void
+0xbe8 glTable                        : Ptr32 Void
+0xbec glCurrentRC                    : Ptr32 Void
+0xbf0 glContext                      : Ptr32 Void
+0xbf4 LastStatusValue                : Uint4B
+0xbf8 StaticUnicodeString            : _UNICODE_STRING
+0xc00 StaticUnicodeBuffer            : [261] Uint2B
+0xe0c DeallocationStack              : Ptr32 Void
+0xe10 TlsSlots                       : [64] Ptr32 Void
+0xf10 TlsLinks                       : _LIST_ENTRY
+0xf18 Vdm                            : Ptr32 Void
+0xf1c ReservedForNtRpc               : Ptr32 Void
+0xf20 DbgSsReserved                  : [2] Ptr32 Void
+0xf28 HardErrorsAreDisabled          : Uint4B
+0xf2c Instrumentation                : [16] Ptr32 Void
+0xf6c WinSockData                    : Ptr32 Void
+0xf70 GdiBatchCount                  : Uint4B
+0xf74 InDbgPrint                     : UChar
+0xf75 FreeStackOnTermination         : UChar
+0xf76 HasFiberData                   : UChar
+0xf77 IdealProcessor                 : UChar
+0xf78 Spare3                         : Uint4B
+0xf7c ReservedForPerf                : Ptr32 Void
+0xf80 ReservedForOle                 : Ptr32 Void
+0xf84 WaitingOnLoaderLock            : Uint4B
+0xf88 Wx86Thread                     : _Wx86ThreadState
+0xf94 TlsExpansionSlots              : Ptr32 Ptr32 Void
+0xf98 ImpersonationLocale            : Uint4B
+0xf9c IsImpersonating                : Uint4B
+0xfa0 NlsCache                       : Ptr32 Void
+0xfa4 pShimData                      : Ptr32 Void
+0xfa8 HeapVirtualAffinity            : Uint4B
+0xfac CurrentTransactionHandle       : Ptr32 Void
+0xfb0 ActiveFrame                    : Ptr32 _TEB_ACTIVE_FRAME
+0xfb4 SafeThunkCall                  : UChar
+0xfb5 BooleanSpare                   : [3] UChar
```

PEB

```
+0x000 InheritedAddressSpace     : UChar
+0x001 ReadImageFileExecOptions  : UChar
+0x002 BeingDebugged             : UChar
+0x003 SpareBool                 : UChar
+0x004 Mutant                    : Ptr32 Void
+0x008 ImageBaseAddress          : Ptr32 Void
+0x00c Ldr                       : Ptr32 _PEB_LDR_DATA
+0x010 ProcessParameters         : PRTL_USER_PROCESS_PARAMETERS
+0x014 SubSystemData             : Ptr32 Void
+0x018 ProcessHeap               : Ptr32 Void
+0x01c FastPebLock               : Ptr32 _RTL_CRITICAL_SECTION
+0x020 FastPebLockRoutine        : Ptr32 Void
+0x024 FastPebUnlockRoutine      : Ptr32 Void
+0x028 EnvironmentUpdateCount    : Uint4B
+0x02c KernelCallbackTable       : Ptr32 Void
+0x030 SystemReserved            : [1] Uint4B
+0x034 AtlThunkSListPtr32        : Uint4B
+0x038 FreeList                  : Ptr32 _PEB_FREE_BLOCK
+0x03c TlsExpansionCounter       : Uint4B
+0x040 TlsBitmap                 : Ptr32 Void
+0x044 TlsBitmapBits             : [2] Uint4B
+0x04c ReadOnlySharedMemoryBase  : Ptr32 Void
+0x050 ReadOnlySharedMemoryHeap  : Ptr32 Void
+0x054 ReadOnlyStaticServerData  : Ptr32 Ptr32 Void
+0x058 AnsiCodePageData          : Ptr32 Void
+0x05c OemCodePageData           : Ptr32 Void
+0x060 UnicodeCaseTableData      : Ptr32 Void
+0x064 NumberOfProcessors        : Uint4B
+0x068 NtGlobalFlag              : Uint4B
+0x070 CriticalSectionTimeout    : _LARGE_INTEGER
+0x078 HeapSegmentReserve        : Uint4B
+0x07c HeapSegmentCommit         : Uint4B
+0x080 HeapDeCommitTotalFreeThreshold : Uint4B
+0x084 HeapDeCommitFreeBlockThreshold : Uint4B
+0x088 NumberOfHeaps             : Uint4B
+0x08c MaximumNumberOfHeaps      : Uint4B
+0x090 ProcessHeaps              : Ptr32 Ptr32 Void
+0x094 GdiSharedHandleTable      : Ptr32 Void
+0x098 ProcessStarterHelper      : Ptr32 Void
+0x09c GdiDCAttributeList        : Uint4B
+0x0a0 LoaderLock                : Ptr32 Void
+0x0a4 OSMajorVersion            : Uint4B
+0x0a8 OSMinorVersion            : Uint4B
+0x0ac OSBuildNumber             : Uint2B
+0x0ae OSCSDVersion              : Uint2B
+0x0b0 OSPlatformId              : Uint4B
+0x0b4 ImageSubsystem            : Uint4B
+0x0b8 ImageSubsystemMajorVersion : Uint4B
+0x0bc ImageSubsystemMinorVersion : Uint4B
+0x0c0 ImageProcessAffinityMask : Uint4B
+0x0c4 GdiHandleBuffer           : [34] Uint4B
+0x14c PostProcessInitRoutine    : Ptr32     void
+0x150 TlsExpansionBitmap        : Ptr32 Void
+0x154 TlsExpansionBitmapBits    : [32] Uint4B
+0x1d4 SessionId                 : Uint4B
+0x1d8 AppCompatFlags            : _ULARGE_INTEGER
+0x1e0 AppCompatFlagsUser        : _ULARGE_INTEGER
+0x1e8 pShimData                 : Ptr32 Void
+0x1ec AppCompatInfo             : Ptr32 Void
+0x1f0 CSDVersion                : _UNICODE_STRING
+0x1f8 ActivationContextData     : Ptr32 Void
+0x1fc ProcessAssemblyStorageMap : Ptr32 Void
+0x200 SystemDefaultActivationContextData : Ptr32 Void
+0x204 SystemAssemblyStorageMap  : Ptr32 Void
+0x208 MinimumStackCommit        : Uint4B
```

RTL_USER_PROCESS_PARAMETERS

- Prozess-Laufzeit-Informationen
  - Pfad und Dateiname der Anwendung
  - Kommandozeilenparameter
  - Dll-Suchpfad
  - Hauptfenster-Informationen (maximiert, …)

RTL_USER_PROCESS_PARAMETERS

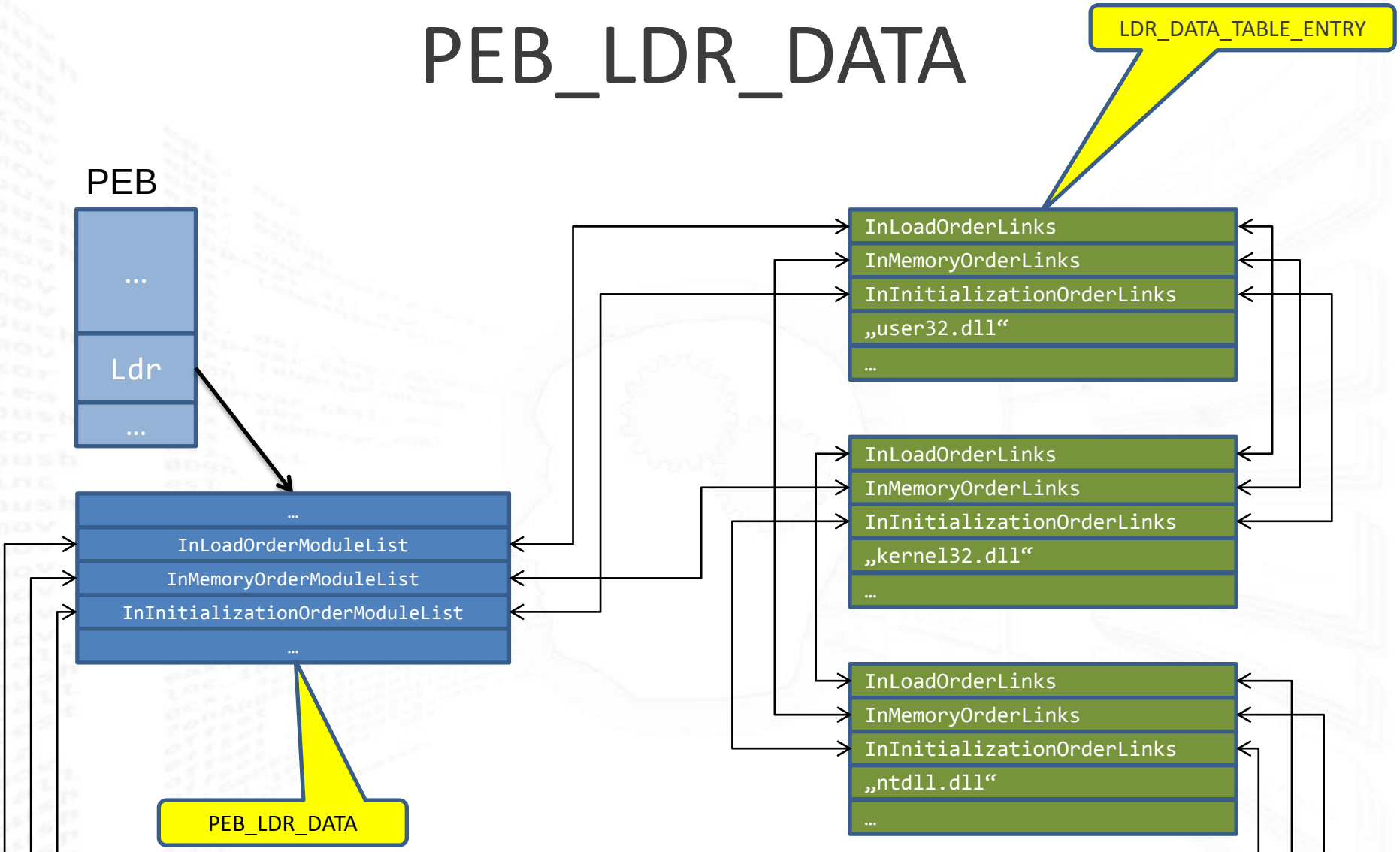
```
+0x000 MaximumLength          : Uint4B
+0x004 Length                 : Uint4B
+0x008 Flags                  : Uint4B
+0x00c DebugFlags             : Uint4B
+0x010 ConsoleHandle          : Ptr32 Void
+0x014 ConsoleFlags           : Uint4B
+0x018 StandardInput          : Ptr32 Void
+0x01c StandardOutput         : Ptr32 Void
+0x020 StandardError          : Ptr32 Void
+0x024 CurrentDirectory       : _CURDIR
+0x030 DllPath                : _UNICODE_STRING
+0x038 ImagePathName          : _UNICODE_STRING
+0x040 CommandLine            : _UNICODE_STRING
+0x048 Environment            : Ptr32 Void
+0x04c StartingX              : Uint4B
+0x050 StartingY              : Uint4B
+0x054 CountX                 : Uint4B
+0x058 CountY                 : Uint4B
+0x05c CountCharsX            : Uint4B
+0x060 CountCharsY            : Uint4B
+0x064 FillAttribute          : Uint4B
+0x068 WindowFlags            : Uint4B
+0x06c ShowWindowFlags        : Uint4B
+0x070 WindowTitle            : _UNICODE_STRING
+0x078 DesktopInfo            : _UNICODE_STRING
+0x080 ShellInfo              : _UNICODE_STRING
+0x088 RuntimeData            : _UNICODE_STRING
+0x090 CurrentDirectores      : [32] _RTL_DRIVE_LETTER_CURDIR
```

PEB_LDR_DATA

- Loader-Informationen
  - Geladene DLLs mit
    - Startadresse, Grösse, Adresse der Initialisierungsroutine
  - Speicher-/Lade-/Initialisierungsreihenfolge

```
+0x000 Length                          : Uint4B
+0x004 Initialized                     : UChar
+0x008 SsHandle                        : Ptr32 Void
+0x00c InLoadOrderModuleList           : _LIST_ENTRY
+0x014 InMemoryOrderModuleList         : _LIST_ENTRY
+0x01c InInitializationOrderModuleList : _LIST_ENTRY
+0x024 EntryInProgress                 : Ptr32 Void
```

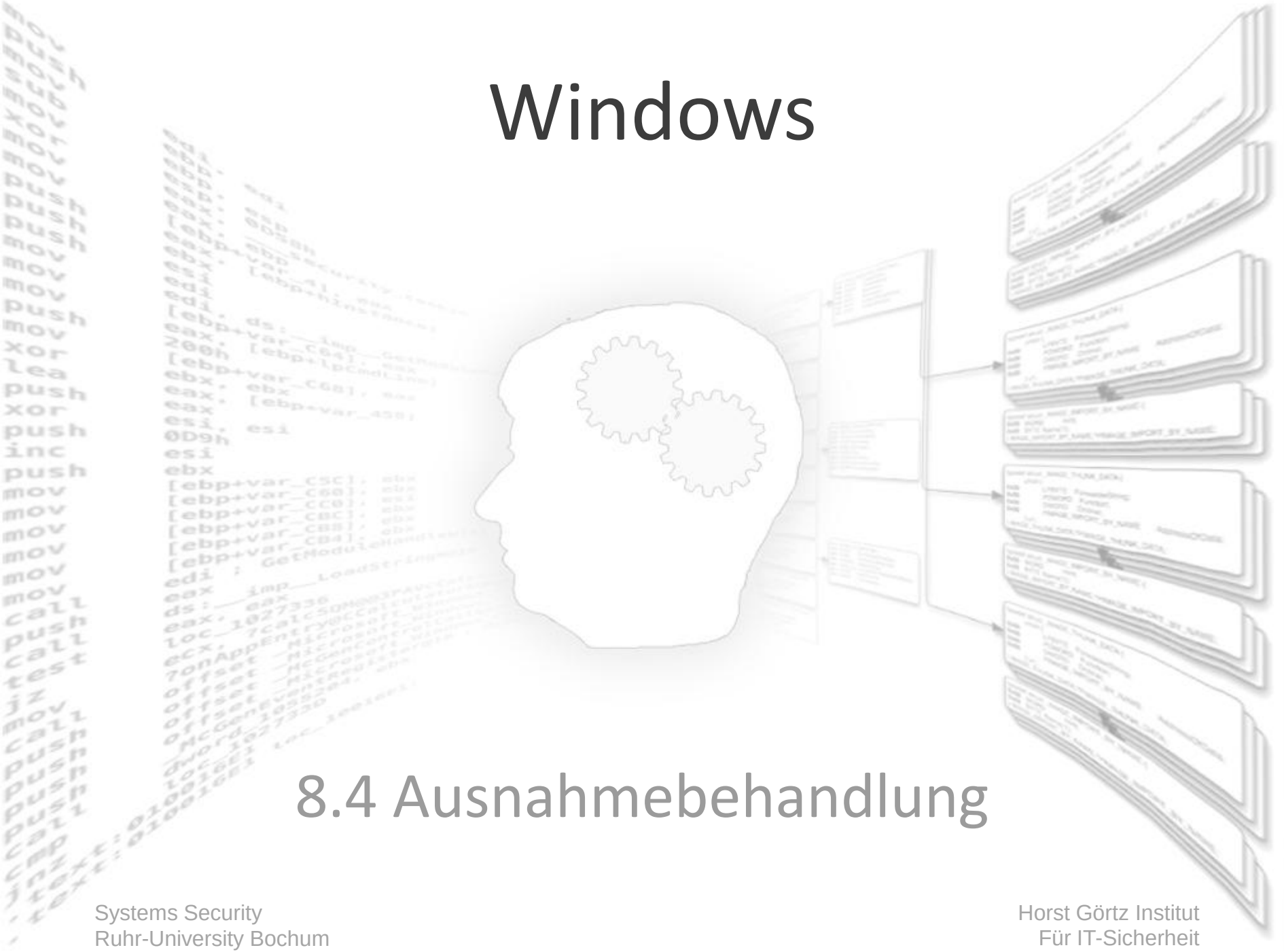
PEB_LDR_DATA


LDR_DATA_TABLE_ENTRY
PEB
...
Ldr
...
...
InLoadOrderModuleList
InMemoryOrderModuleList
InInitializationOrderModuleList
...
PEB_LDR_DATA
InLoadOrderLinks
InMemoryOrderLinks
InInitializationOrderLinks
„user32.dll“
...
InLoadOrderLinks
InMemoryOrderLinks
InInitializationOrderLinks
„kernel32.dll“
...
InLoadOrderLinks
InMemoryOrderLinks
InInitializationOrderLinks
„ntdll.dll“
...

LDR_DATA_TABLE_ENTRY

```
+0x000 InLoadOrderLinks                 : _LIST_ENTRY
+0x008 InMemoryOrderLinks               : _LIST_ENTRY
+0x010 InInitializationOrderLinks       : _LIST_ENTRY
+0x018 DllBase                          : Ptr32 Void
+0x01c EntryPoint                       : Ptr32 Void
+0x020 SizeOfImage                      : Uint4B
+0x024 FullDllName                      : _UNICODE_STRING
+0x02c BaseDllName                      : _UNICODE_STRING
+0x034 Flags                            : Uint4B
+0x038 LoadCount                        : Uint2B
+0x03a TlsIndex                         : Uint2B
+0x03c HashLinks                        : _LIST_ENTRY
+0x03c SectionPointer                   : Ptr32 Void
+0x040 CheckSum                         : Uint4B
+0x044 TimeDateStamp                    : Uint4B
+0x044 LoadedImports                    : Ptr32 Void
+0x048 EntryPointActivationContext      : Ptr32 Void
+0x04c PatchInformation                 : Ptr32 Void
```

Windows



8.4 Ausnahmebehandlung

Ausnahmen / Exceptions

- Exceptions zeigen Fehler-/Ausnahmesituation an
- Hardware Exceptions
  - Von CPU ausgelöst
    - Division durch 0
    - Ungültige Speicheradresse
- Software Exceptions
  - Explizit vom OS oder Anwendung ausgelöst
    - Software Breakpoint
    - Ungültige Parameter in Funktionsaufruf

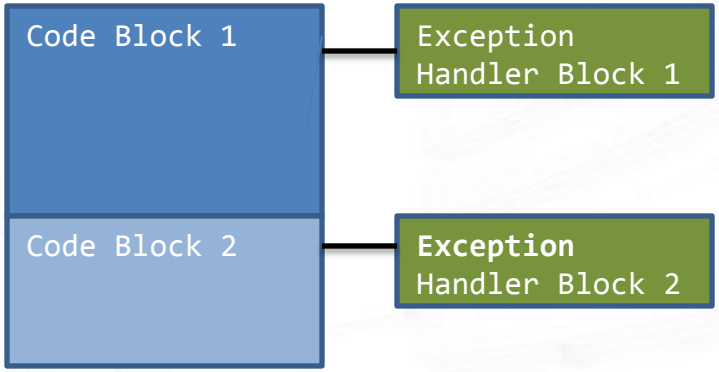
Structured Exception Handling (SEH)

- *Structured* == Ausnahmebehandlung **getrennt** vom eigentlichen Anwendungscode
- Exception beschrieben durch
  - ExceptionCode, ExceptionFlags, …
  - ExceptionRecords (abhängig vom Exceptioncode)
  - Context-Informationen (EIP, CS, …)
- Exception Handler
  - Funktion, die auf Exception reagiert

Frame Based Exception Handling

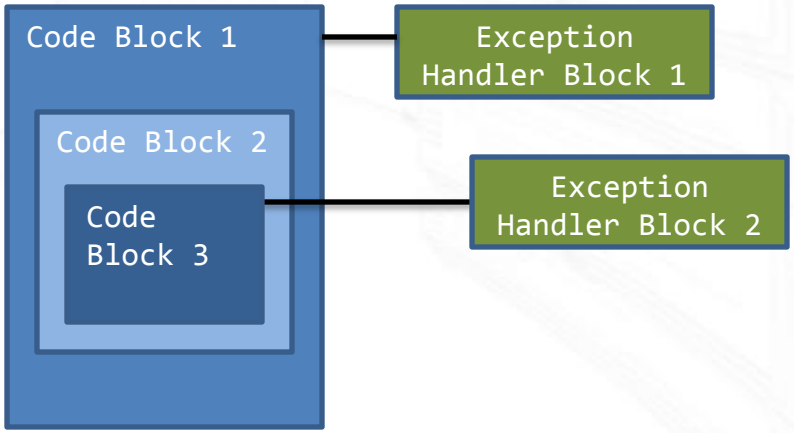
- Codeblock mit Exception Handler geschützt
  - Bei Exception in Codeblock

  ➔ zugeordneter Handler



- Bei Funktionsaufruf/ rekursivem Aufruf
  - Exception Handler Stack
  - Wenn Exception nicht bearbeitet werden kann

  ➔ nächsten Handler aufrufen



SEH Implementierung

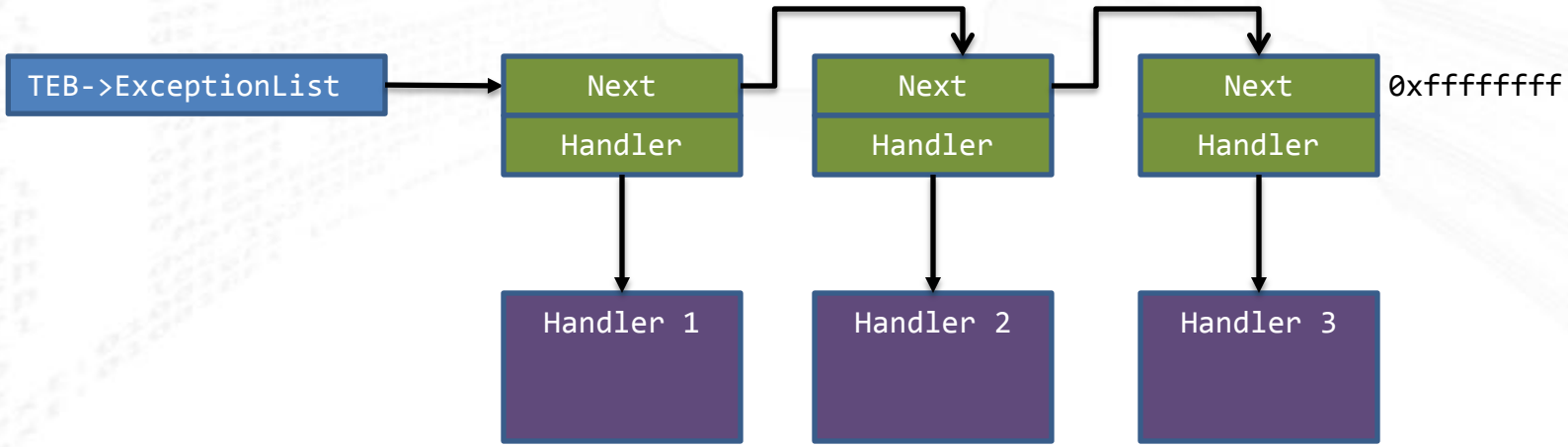
- Erstes Feld in TEB

```
ExceptionList : Ptr32 EXCEPTION_REGISTRATION_RECORD
```

- zeigt auf Liste aus Exception Handlern:

```
EXCEPTION_REGISTRATION_RECORD =
   +0x000 Next    : Ptr32 EXCEPTION_REGISTRATION_RECORD
   +0x004 Handler : Ptr32 EXCEPTION_DISPOSITION
```

- Letztes Listenelement: *default exception handler*

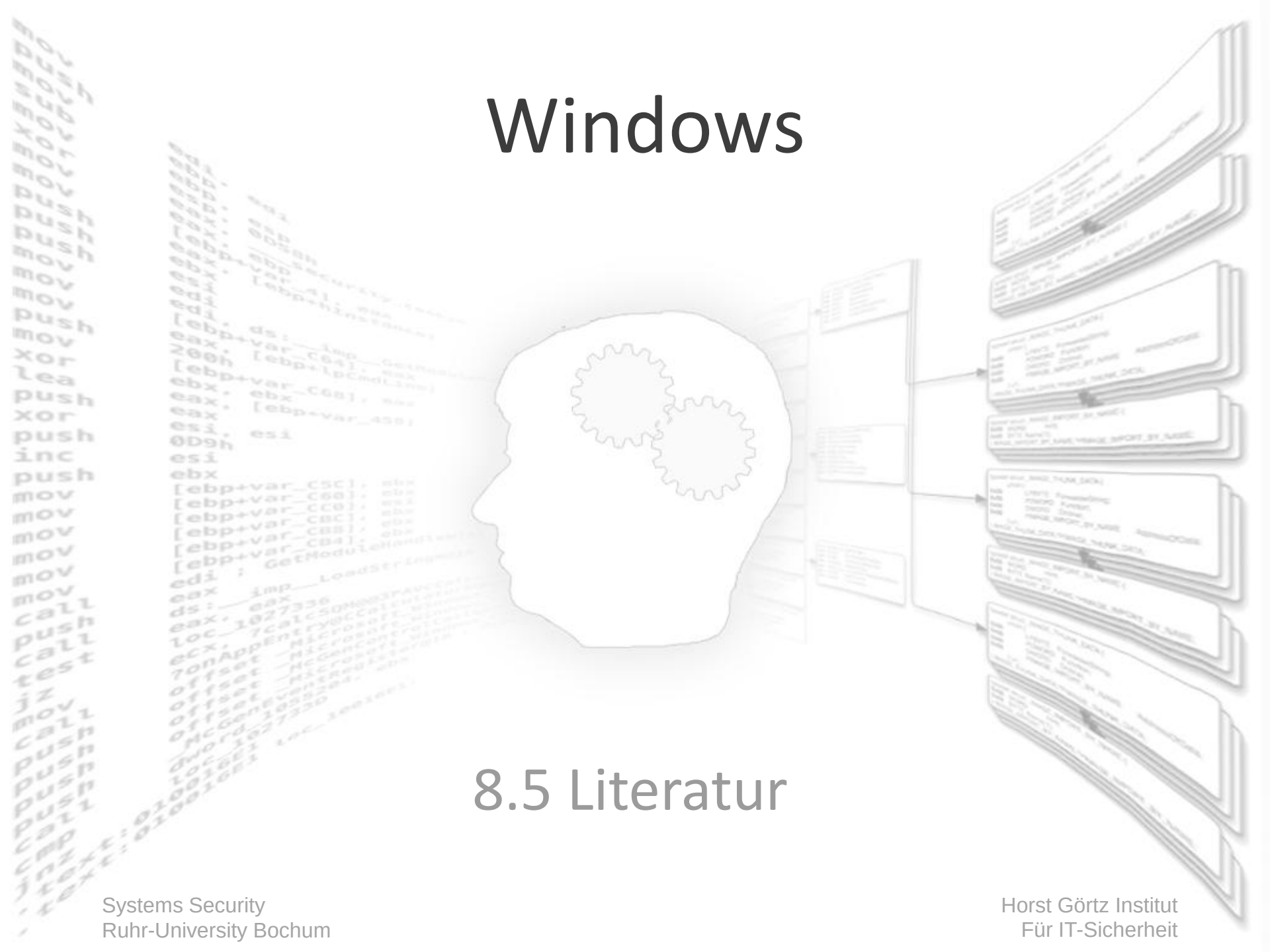


Vectored Exception Handling

- Erweiterung von SEH
- Handler sind *nicht* an Codeblock gebunden
- Beliebig lange Liste von Handler möglich
- Handler werden der Reihe nach aufgerufen
  - Erst VEH-Handler, dann SEH-Handler
- Beispiel:

```
PVOID WINAPI AddVectoredExceptionHandler(
  __in  ULONG FirstHandler,
  __in  PVECTORED_EXCEPTION_HANDLER VectoredHandler);
```

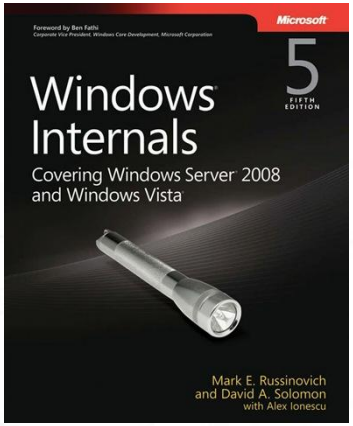
Windows



8.5 Literatur

Literatur

- *Windows Internals*
  von Mark E. Russinovich
  und David A. Solomon



- *Windows NT/2000 Native API Reference*
  von Gary Nebbett