Exploring the x64

Fourteenforty Research Institute, Inc.

DISCLAIMER and Goal

- ~~IA64~~
- ~~Linux, *BSD, Mac, etc.~~

✓ Make clear the various weird techniques which are used under x86 on x64 environment

Environment

- Windows 7 x64 Edition
- Visual Studio 2008
- Windbg
- IDA Pro Advanced
  - *STD doesn't support x64, an offering is needed!*

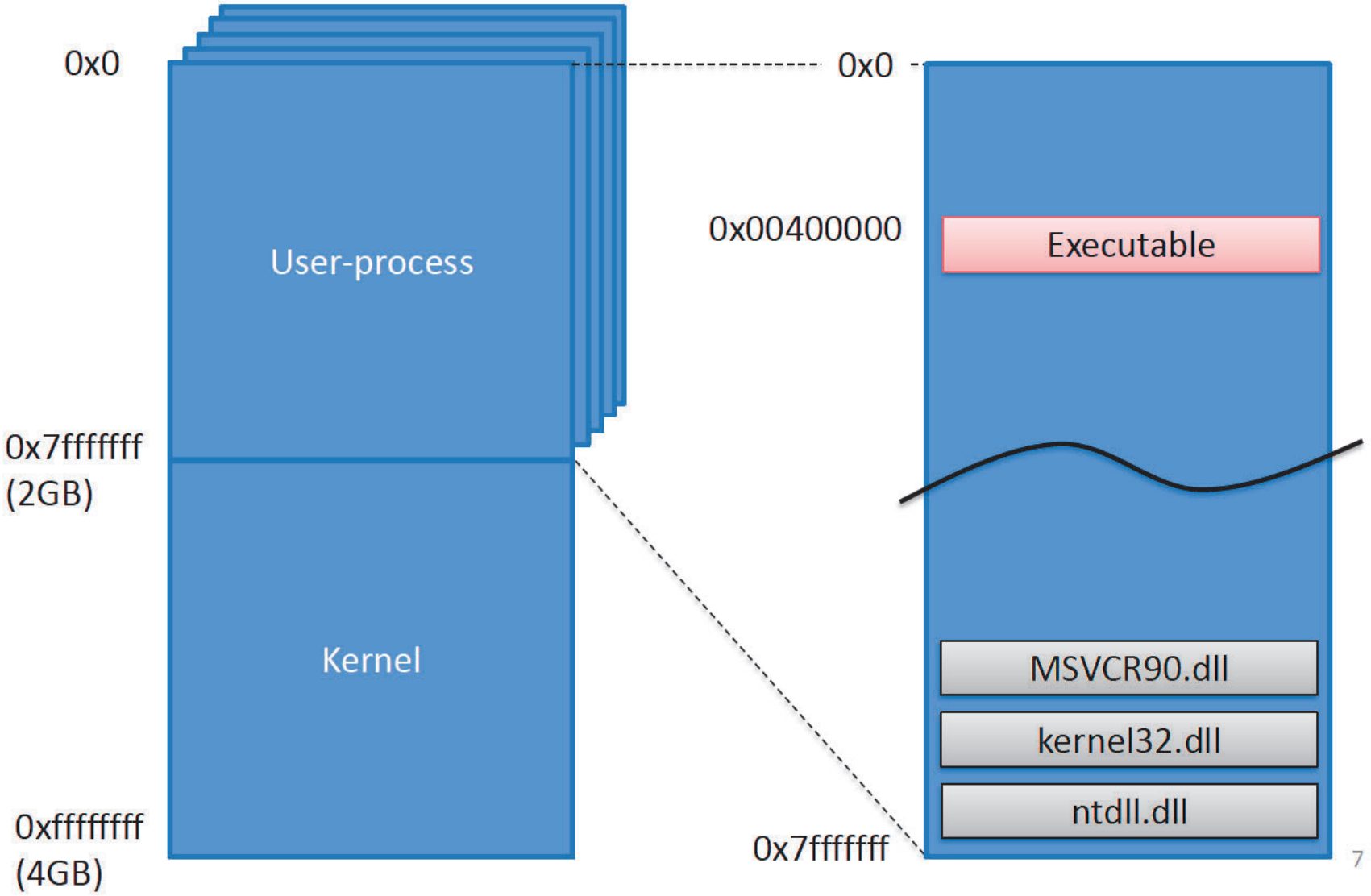
Agenda

- Windows x64
- ABI(Application Binary Interface)
- API Hooking
- Code Injection

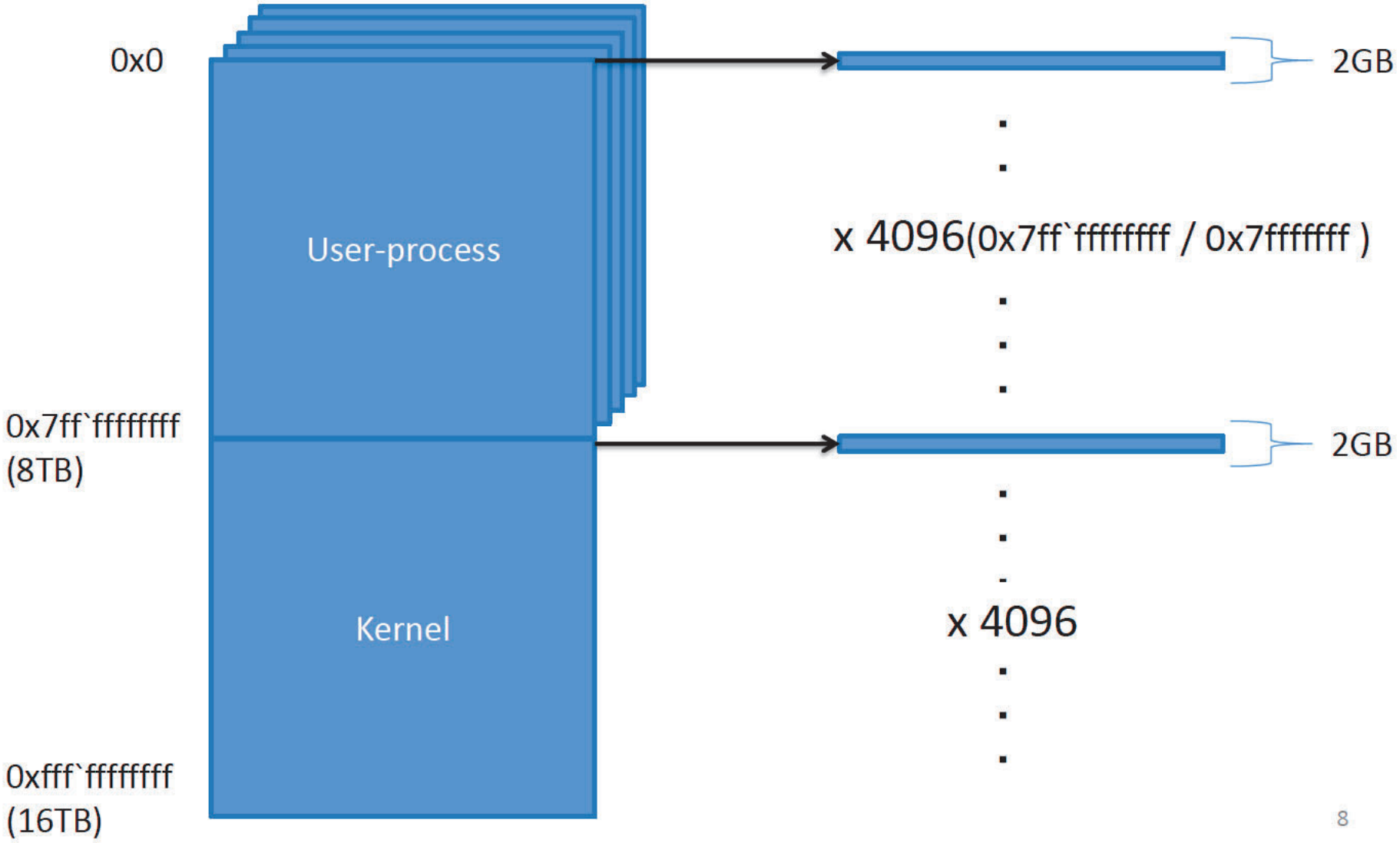
Windows x64

- Native x64 and WoW64
- Virtual Address Space
  - 2^64 = 16 Exa Byte (Exa: 10^18)
  - but, limited to 16TB by Microsoft
- File/Registry reflection
- New 64-bit APIs
  - IsWow64Process, GetNativeSystemInfo, etc.

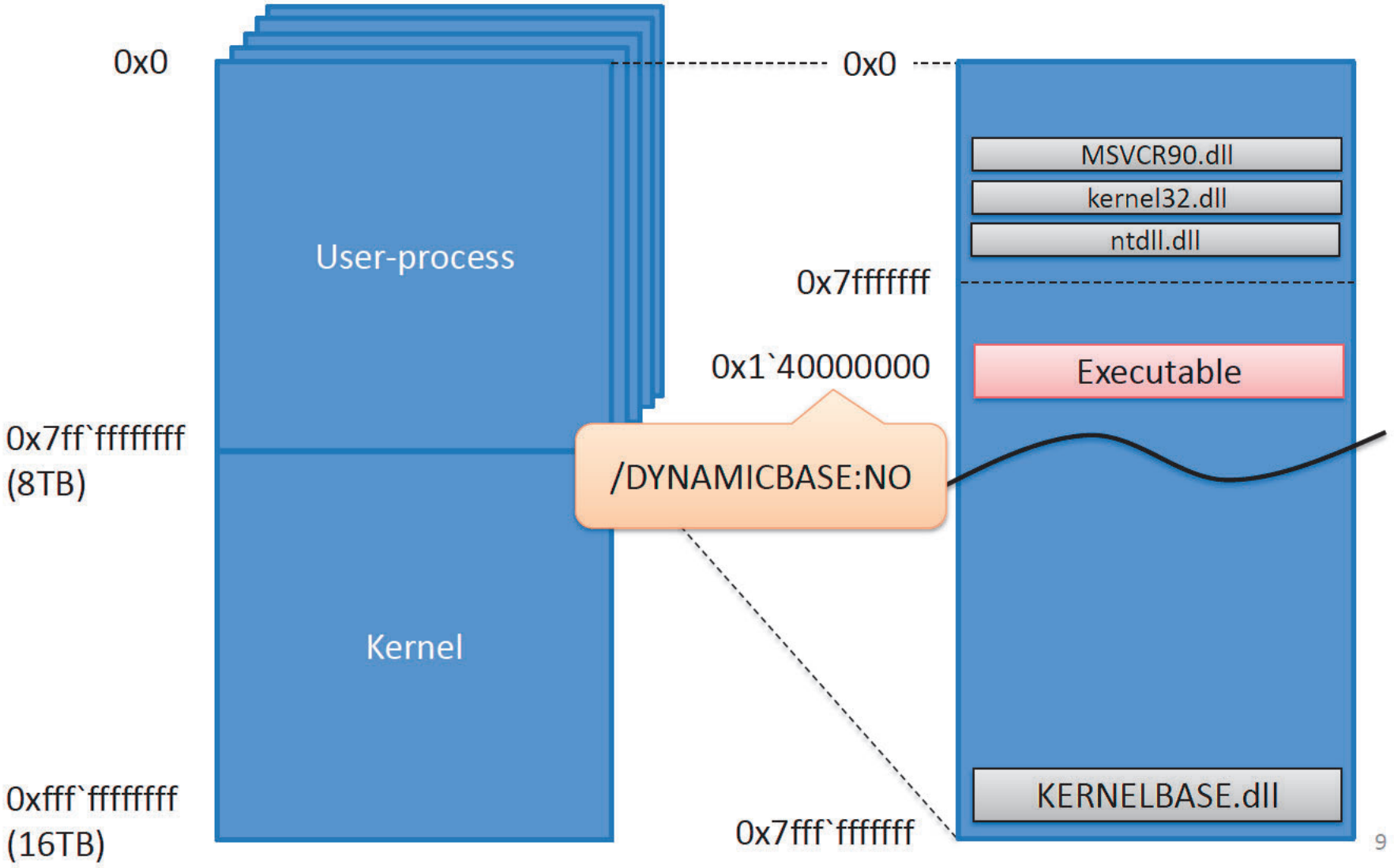
x86 – Process Memory Layout



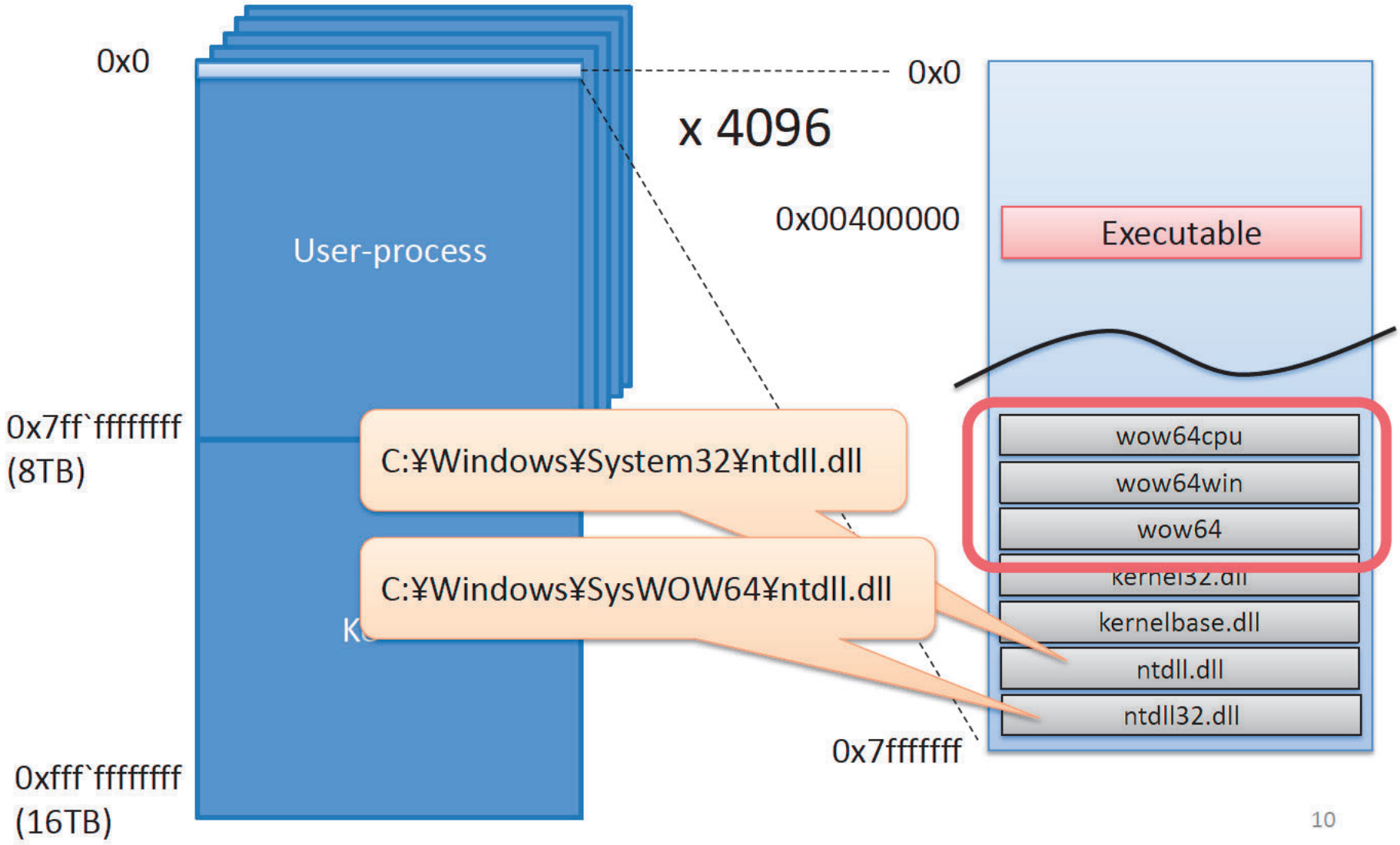
x64 – Process Memory Layout



x64 – Process Memory Layout(Cont.)



WoW64 – Process Memory Layout

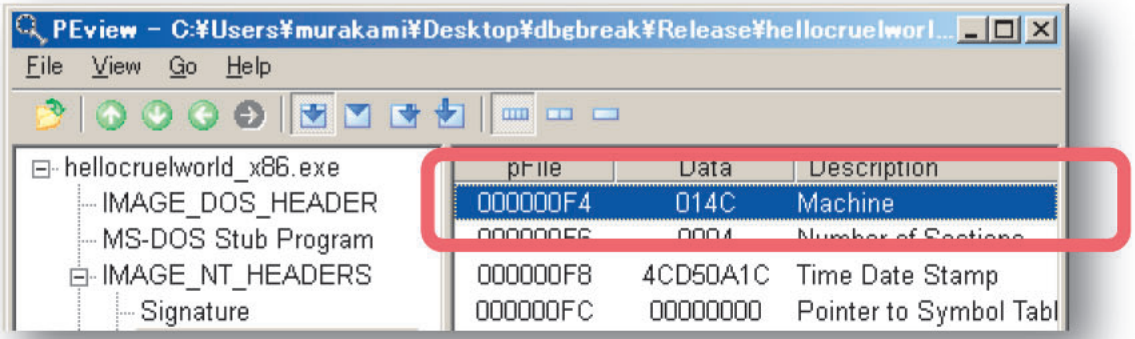


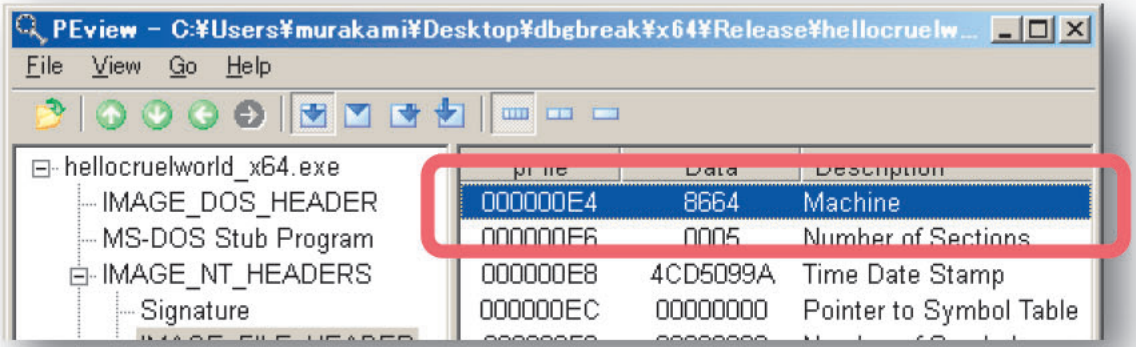
ABI

- Binary Format
- Register
- Calling Convention
- Exception Handling
- Systemcall(x64, WoW64)

Binary Format = PE32+

- Mostly the same as PE32
- IMAGE_NT_HEADERS.FileHeader.Machine
  - 0x014c => x86
  - 0x8664 => x64





Binary Format(Cont.)

- Some fields were extended to 64-bits
  - IMAGE_NT_HEADERS.IMAGE_OPTIONAL_HEADER
    - ImageBase
    - SizeOfStackReserve
    - SizeOfStackCommit
    - SizeOfHeapReserve
    - SizeOfHeapCommit

Register

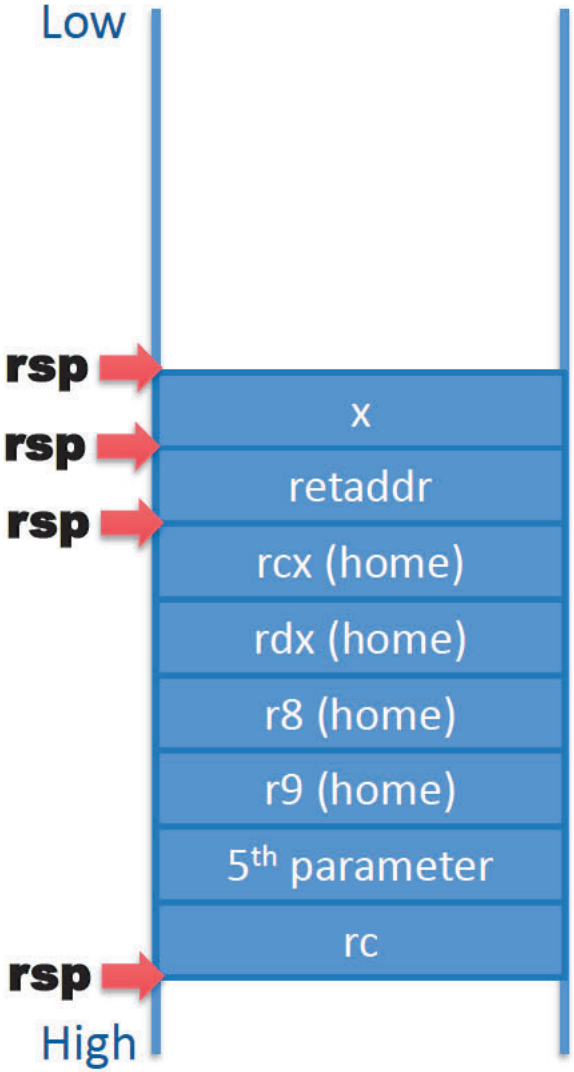
x86(32-bits)	x64(64-bits)	
EAX	RAX	R8
ECX	RCX	R9
EDX	RDX	R10
EBX	RBX	R11
ESI	RSI	R12
EDI	RDI	R13
ESP	RSP	R14
EBP	RBP	R15
EIP	RIP	

Just a GPR, not *BasePointer*

Calling Convention

- first 4 parameters are passed by RCX, RDX, R8, R9
  - 5th and later are passed on the stack
- caller allocates register home space on the stack
- RAX is used for return values
- leaf / non-leaf function
  - leaf function: never use stack
  - PE32+ contains non-leaf function's information in its EXCEPTION DIRECTORY
- Register's volatility
  - volatile: RAX, RCX, RDX, R8-R11

Calling Convention




```
int foo(int a, int b, int c, int d, int e)
{
        int x = 0;

}

int
{
        call foo

        rc = foo(1, 2, 3, 4, 5);
        printf("%d¥n", rc);

        return rc;
}
```

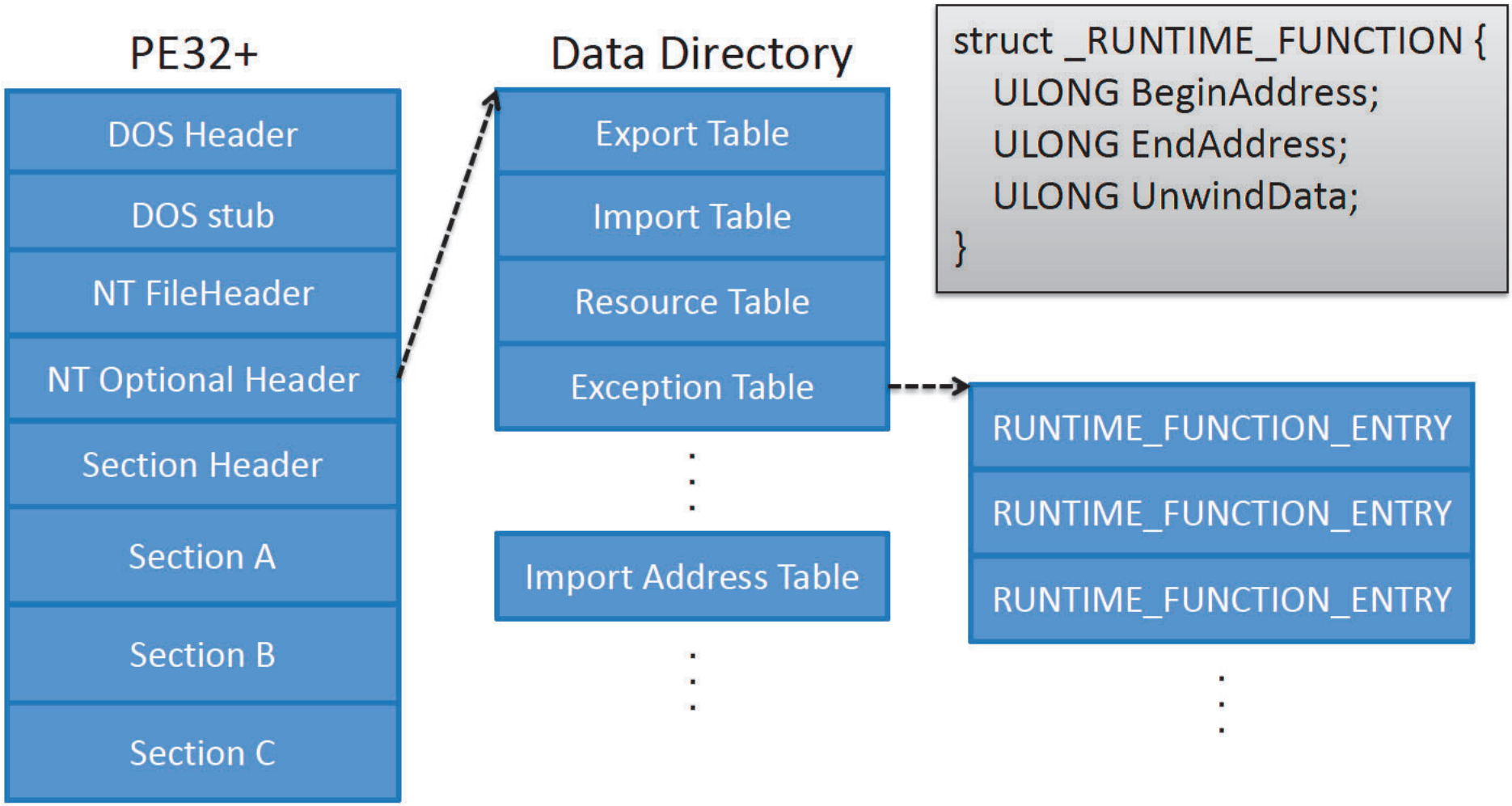
```
mov dword ptr [rsp+20h],r9d
mov dword ptr [rsp+18h],r8d
mov dword ptr [rsp+10h],edx
mov dword ptr [rsp+8h],ecx
sub  rsp,8h
```

Exception Handling

- Table-base
  - linked-list is no longer used

if you don't know the classic SEH mechanism, you should check Shuichiro Suzuki's works!

Exception Directory and RUNTIME_FUNCTION



dumpbin /unwindinfo

```
Begin    End      Info     Function Name

 00000000 00001000 00001041 000022D4  foo
   Unwind version: 1
   Unwind flags: None
   Size of prologue: 0x16
   Count of codes: 1
   Unwind codes:
     16: ALLOC_SMALL, size=0x18

 0000000C 00001050 00001095 000022DC  main
   Unwind version: 1
   Unwind flags: None
   Size of prologue: 0x04
   Count of codes: 1
   Unwind codes:
     04: ALLOC_SMALL, size=0x48
```

RUNTIME_FUNCTION.UnwindData

```
typedef struct _UNWIND_INFO {
    UBYTE Version         : 3;
    UBYTE Flags           : 5;
    UBYTE SizeOfProlog;
    UBYTE CountOfCodes;
    UBYTE FrameRegister   : 4;
    UBYTE FrameOffset     : 4;
    UNWIND_CODE UnwindCode[1];
    union {
        // If (Flags & UNW_FLAG_EHANDLER)
        OPTIONAL ULONG ExceptionHandler;
        // Else if (Flags & UNW_FLAG_CHAININFO)
        OPTIONAL ULONG FunctionEntry;
    };
    // If (Flags & UNW_FLAG_EHANDLER)
    OPTIONAL ULONG ExceptionData[];
} UNWIND_INFO, *PUNWIND_INFO;
```

```
#define UNW_FLAG_NHANDLER 0x0
#define UNW_FLAG_EHANDLER 0x1
#define UNW_FLAG_UHANDLER 0x2
#define UNW_FLAG_CHAININFO 0x4
```

cf. http://www.osronline.com/article.cfm?article=469

ExceptionData

```
typedef struct _SCOPE_TABLE {
    ULONG Count;
    struct
    {
        ULONG BeginAddress;
        ULONG EndAddress;
        ULONG HandlerAddress;
        ULONG JumpTarget;
    } ScopeRecord[1];
} SCOPE_TABLE, *PSCOPE_TABLE;
```

cf. http://www.osronline.com/article.cfm?article=469

try/except

```
int main(void)
{
  int x = 0;

  __try {
    printf("%d¥n", 100/x);
    printf("foo¥n");
    printf("bar¥n");
    printf("baz¥n");
  } __except(EXCEPTION_EXECUTE_HANDLER) {
    printf("catch!¥n");
  }

  return 0;
}
```

try/except

```
Name  main
Unwind version: 1
Unwind flags: EHANDLER
Size of prologue: 0x04
Count of codes: 1

Unwind codes
04: ALLOC_SMALL, size=0x28
Handler:0000165C __C_specific_handler

Count of scope table entries: 1
Begin   00001004
End     00001046
Handler 00000001
Target  00001046
```

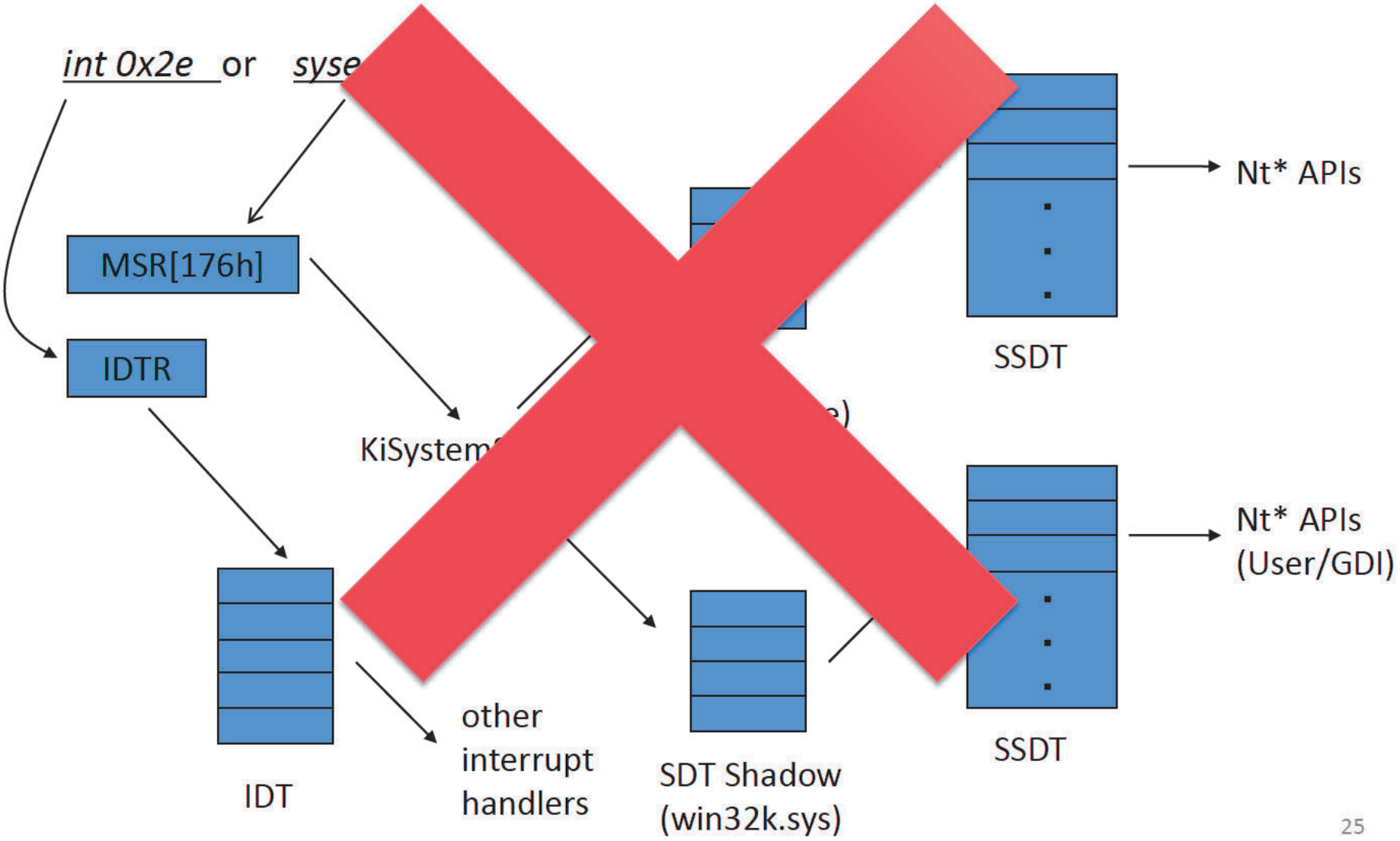
```
140001000  sub  rsp,28h
140001004  mov  eax,64h
140001009  cdq
14000100A  xor  ecx,ecx
14000100C  idiv eax,ecx
14000100E  mov  edx,eax
140001010  lea  rcx,[400021B0h]
140001017  call qword ptr [40002130h]
14000101D  lea  rcx,[400021B4h]
140001024  call qword ptr [40002130h]
14000102A  lea  rcx,[400021BCh]
140001031  call qword ptr [40002130h]
140001037  lea  rcx,[400021C4h]
14000103E  call qword ptr [40002130h]
140001044  jmp  0000000140001054

140001046  lea  rcx,[400021D0h]
14000104D  call qword ptr [40002130h]
140001053  nop
140001054  xor  eax,eax
140001056  add  rsp,28h
14000105A  ret
```

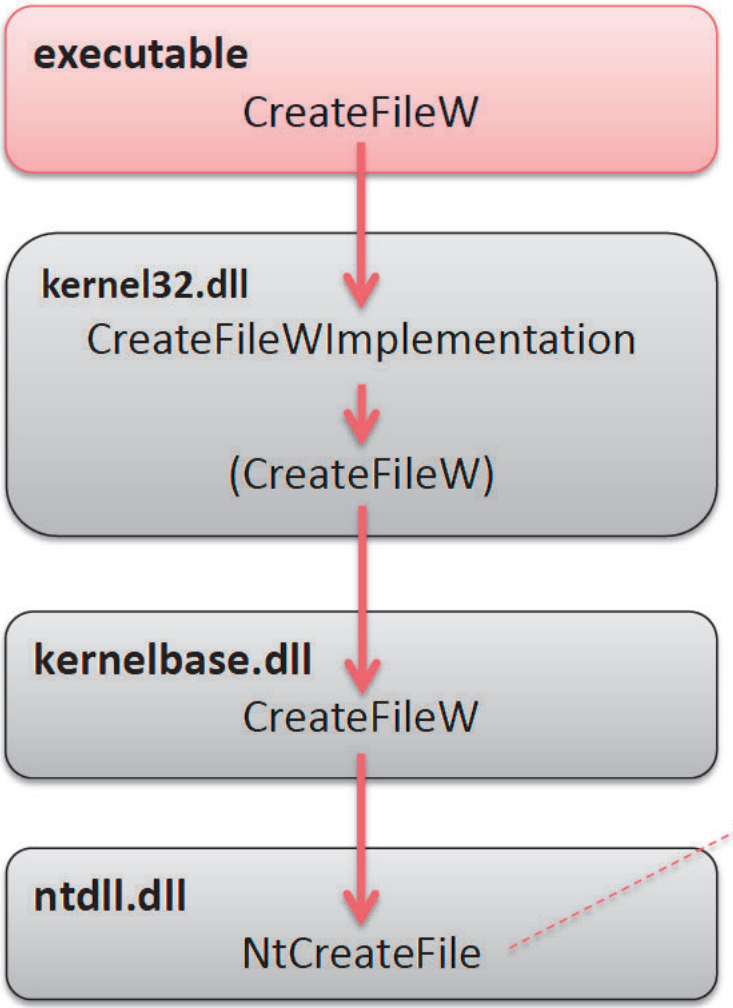
advantages of the exception directory (RF structure)

- possible to enumerate all non-leaf functions
- possible to understand
  - each function's exception information
  - each function's usage of stack and volatile registers

Systemcall x86

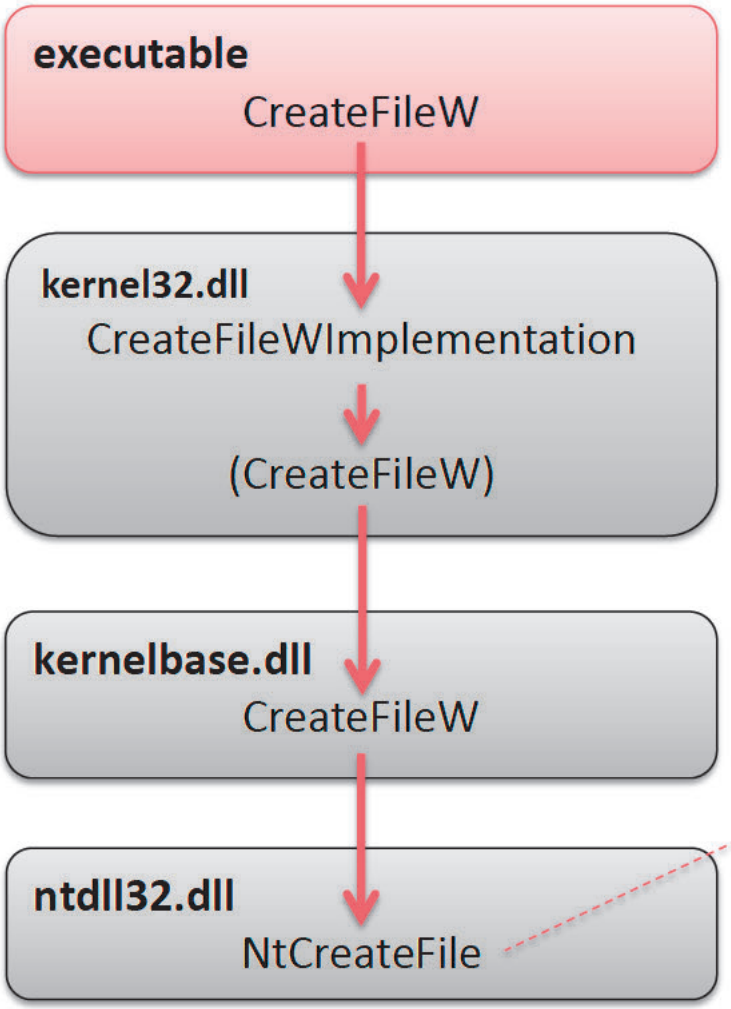


Systemcall x64




```
mov     r10,rcx
mov     eax,52h
syscall
ret
nop     dword ptr [rax+rax]
```

Systemcall WoW64




```
mov     eax,52h
xor     ecx,ecx
lea     edx,[esp+4]
call    dword ptr fs:[0C0h]
add     esp,4
ret     2Ch
```

fs:[0C0h]

- *FS* register points to TEB(Thread Environment Block)

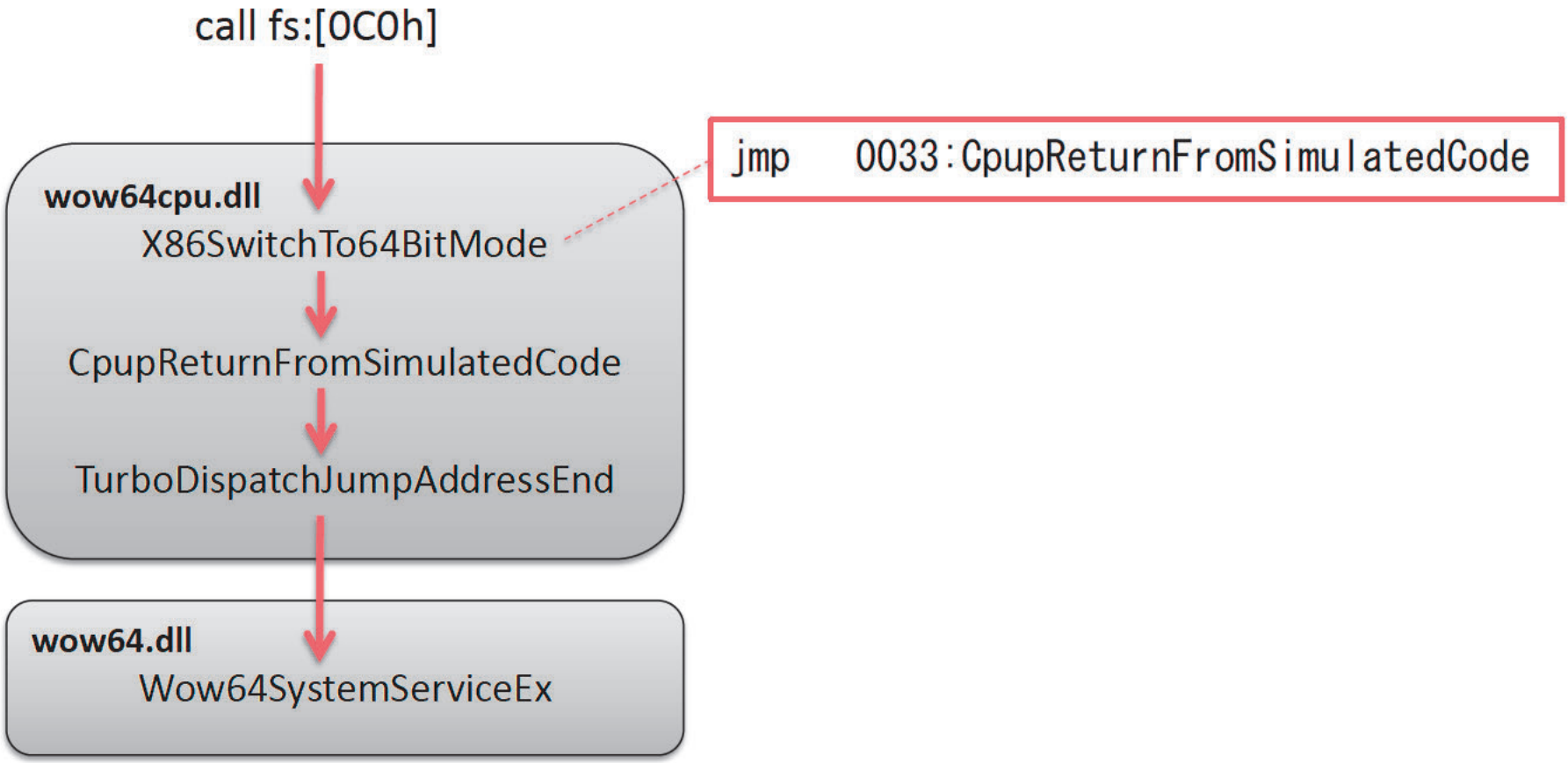
```
0:000:x86> dt _TEB
dbgbreak!_TEB
   +0x000 NtTib            : _NT_TIB
   ( ... )
```

```
0:000:x86> dd fs:[0C0h]
0053:000000c0  738c2320 00000411 00000000 00000000
```

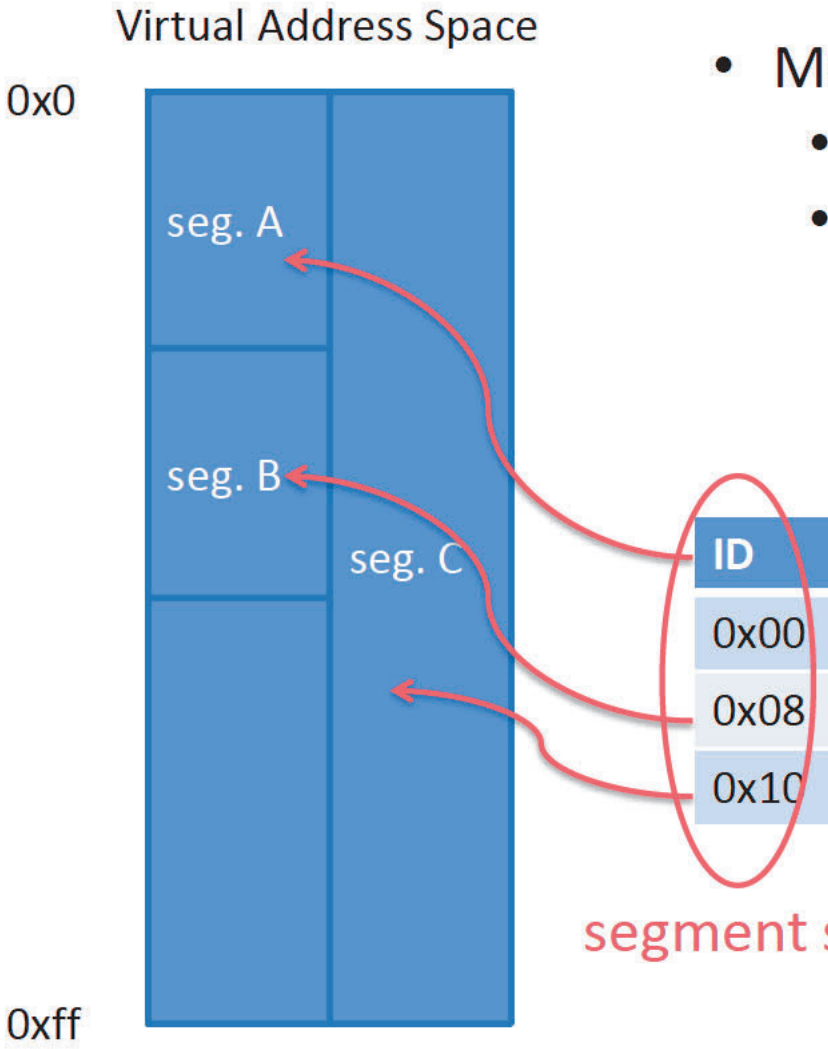
↑ X86SwitchTo64BitMode

```
   +0x040 Win32ThreadInfo  : Ptr32 Void
   +0x044 User32Reserved   : [26] Uint4B
   +0x0ac UserReserved     : [5] Uint4B
   +0x0c0 WOW32Reserved    : Ptr32 Void
```

Systemcall WoW64



GDT *(super quick interpretation)*



- Manage memory as *segment*
  - Kernel code/data
  - User code/data, etc.

GDT

ID	seg.	base	limit	type
0x00	A	0x0	0x3f	RW
0x08	B	0x40	0x7f	RE
0x10	C	0x0	0xff	RE

segment selector

Dumping GDT entries

```
kd> dg 0x00 0x60
                                                     P Si Gr Pr Lo
Sel        Base              Limit             Type    l ze an es ng Flags
---- ----------------- ----------------- ---------- - -- -- -- -- --------
0000 00000000`00000000 00000000`00000000 <Reserved> 0 Nb By Np Nl 00000000
0008 00000000`00000000 00000000`00000000 <Reserved> 0 Nb By Np Nl 00000000
0010 00000000`00000000 00000000`00000000 Code RE Ac 0 Nb By P  Lo 0000029b
0018 00000000`00000000 00000000`ffffffff Data RW Ac 0 Bg Pg P  Nl 00000c93
0020 00000000`00000000 00000000`ffffffff Code RE    3 Bg Pg P  Nl 00000cfa
0028 00000000`00000000 00000000`ffffffff Data RW Ac 3 Bg Pg P  Nl 00000cf3
0030 00000000`00000000 00000000`00000000 Code RE Ac 3 Nb By P  Lo 000002fb
0038 00000000`00000000 00000000`00000000 <Reserved> 0 Nb By Np Nl 00000000
0040 00000000`00b9b080 00000000`00000067 Code RE Ac 0 Nb By P  Nl 0000008b
0048 00000000`0000ffff 00000000`0000f800 <Reserved> 0 Nb By Np Nl 00000000
0050 ffffffff`fffe0000 00000000`00003c00 Data RW Ac 3 Bg By P  Nl 000004f3
0058 00000000`00000000 00000000`00000000 <Reserved> 0 Nb By Np Nl 00000000
0060 00000000`00000000 00000000`ffffffff Code RE    0 Bg Pg P  Nl 00000c9a
```

Kernel CS(x64), Kernel DS

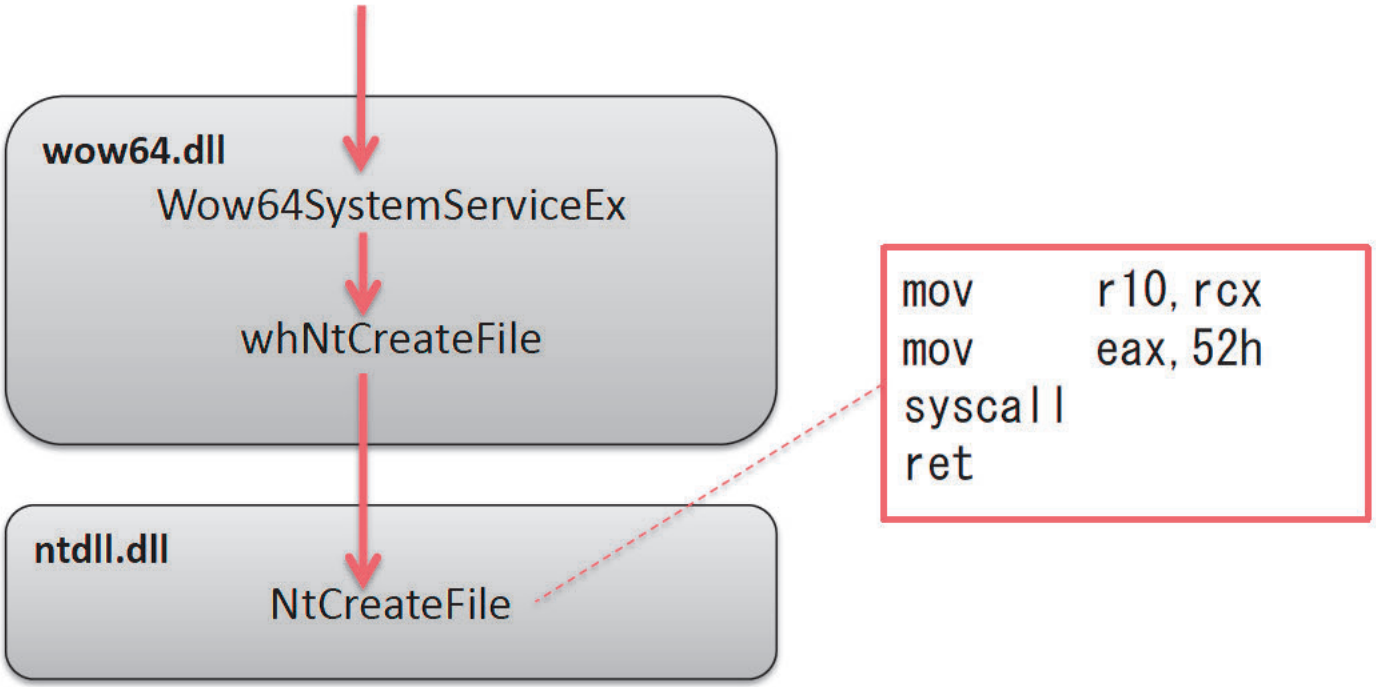
User CS(x86), User DS

User CS(x64)

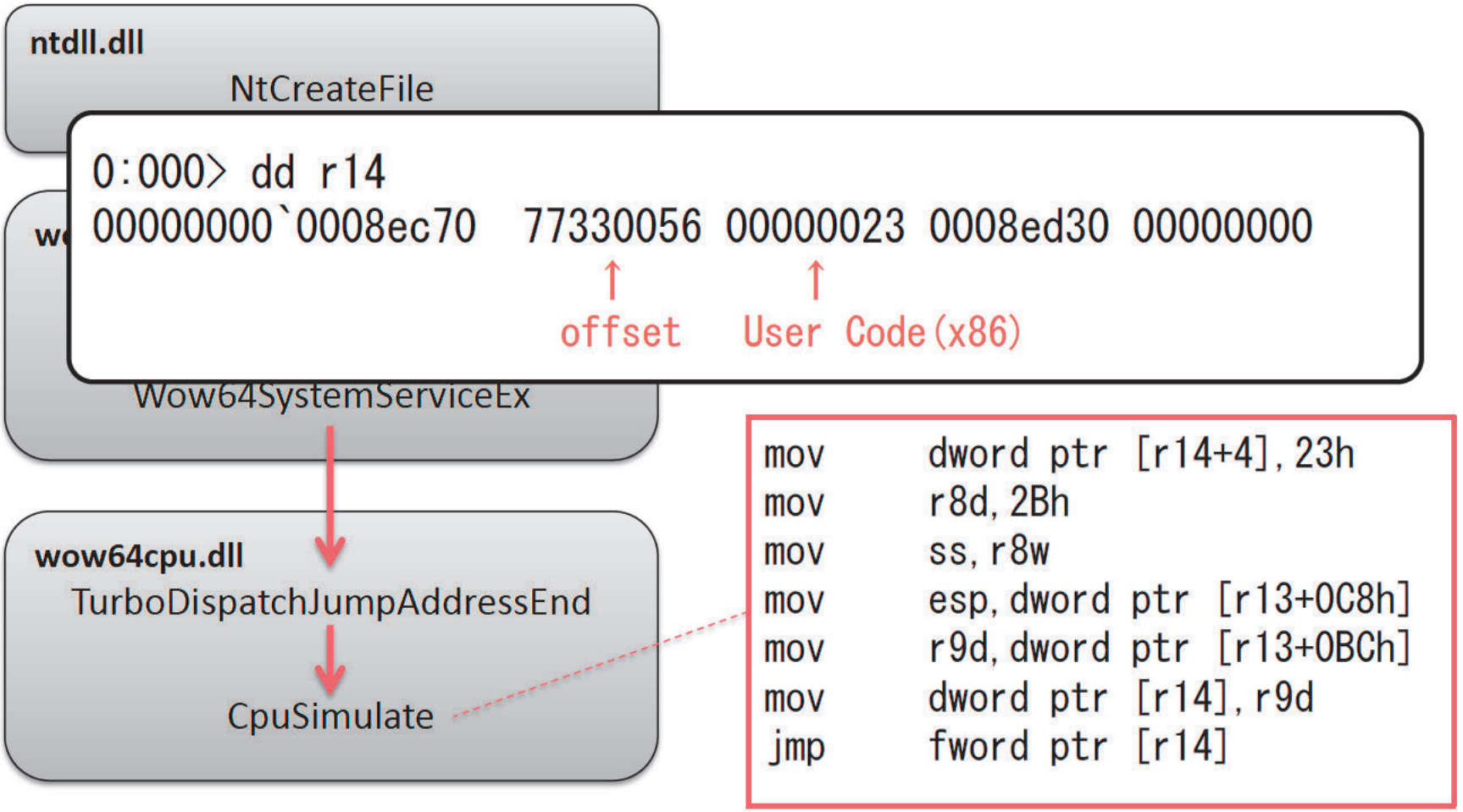
Content of GDT[0x30]

- base: 0x000`00000000
- limite: 0x000`00000000
- type: CODE, Read, Execute and Accessed
- Privilege Level: 3(User-mode)
- L (64-bit code segment) flag: set

Systemcall WoW64



Systemcall WoW64 (return to x86)



Demo: Direct x64 API call from WoW64

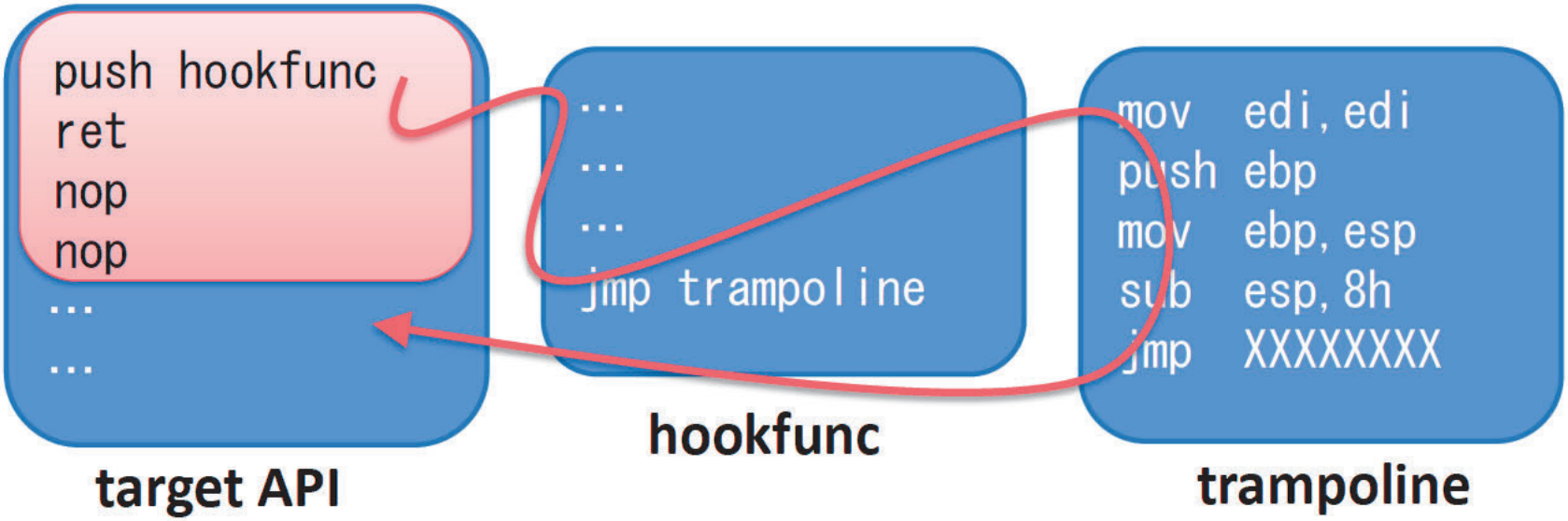
- x86 to x64
  - jmp 0033:XXXXXXXX
- API call
  - rax: syscall number
  - rdx: pointer to parameter list
  - syscall
- x64 to x86
  - call 0023:XXXXXXXX

API Hooking

- IAT Hooking
  - possible to hook IAT on x64 in the same manner as x86
- Code Hooking

Code Hooking

- basic idea is same as x86
- implementation detail is a little different



REX prefix

- 0x40～0x4E
  - x86: INC and DEC inst.
  - x64: REX prefix (register extension)
- ex) 0x48,0xB8,0x11,0x22,0x33,0x44,0x55,0x66,0x77,0x88

x86:

```
48                  dec     eax
B8 11 22 33 44      mov     eax, 44332211h
55                  push    ebp
66 77 88            ja      00004E9F
```

x64:

```
48 B8 11 22 33 44 55 66 77 88  mov     rax, 8877665544332211h
```

Code Hooking

```
00000000779811E4  mov     rax, 7FFFFFA0028h
00000000779811EE  push    rax
00000000779811EF  ret
00000000779811F2  …
```

hookfunc

```
000007FFFFFA0034  sub     rsp, 38h
000007FFFFFA0038  xor     r11d, r11d
000007FFFFFA003B  cmp     dword ptr [7FFFFFC0F8Ch], r11d
000007FFFFFA0042  push    rax
000007FFFFFA0043  mov     rax, 779811F2h
000007FFFFFA004D  xchg    rax, qword ptr [rsp]
000007FFFFFA0051  ret
```

original inst.

jump-back address

Code Injection

- WoW64 to WoW64
- x64 to x64
- ~~WoW64 to x64~~
- x64 to WoW64

(Fail CreateRemoteThead API)

x64 is the 魔除け(talisman) against x86 malware ?

Conclusions

- Windows x64
- ABI(Application Binary Interface)
- API Hooking
- Code Injection