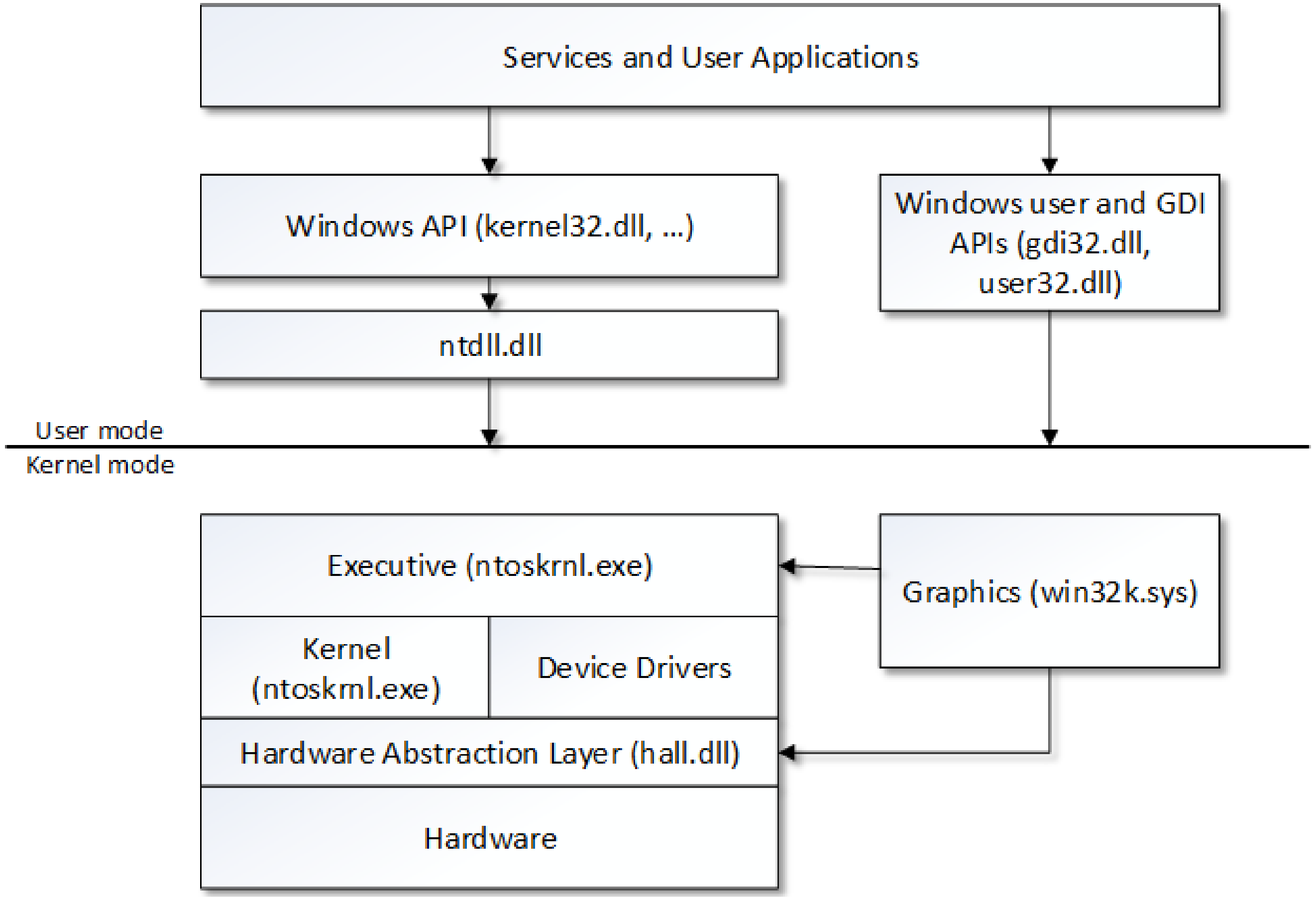
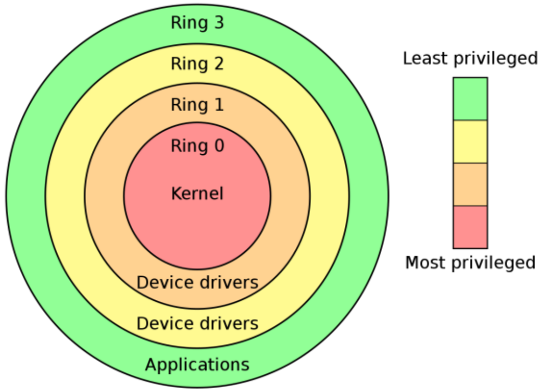
Lab 01

Architecture Overview


Services and User Applications
Windows API (kernel32.dll, ...)
Windows user and GDI APIs (gdi32.dll, user32.dll)
ntdll.dll
User mode
Kernel mode
Executive (ntoskrnl.exe)
Graphics (win32k.sys)
Kernel (ntoskml.exe)
Device Drivers
Hardware Abstraction Layer (hall.dll)
Hardware

User and Kernel Mode

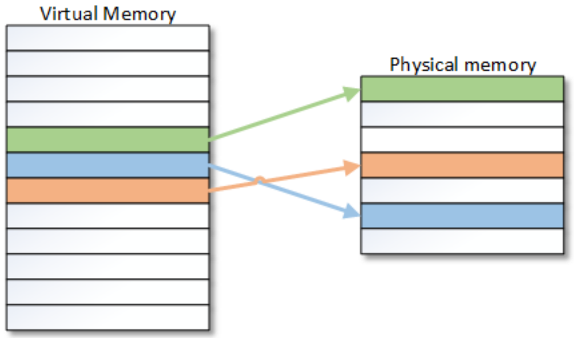
- Windows uses a two-ring model for protection:
  - ring 0 – kernel mode
  - ring 3 – user mode



- Processes have separated memory spaces in user mode
- All drivers and Windows modules in kernel mode share the same virtual address space. In kernel space you have complete access to all drivers and windows objects

Virtual Memory

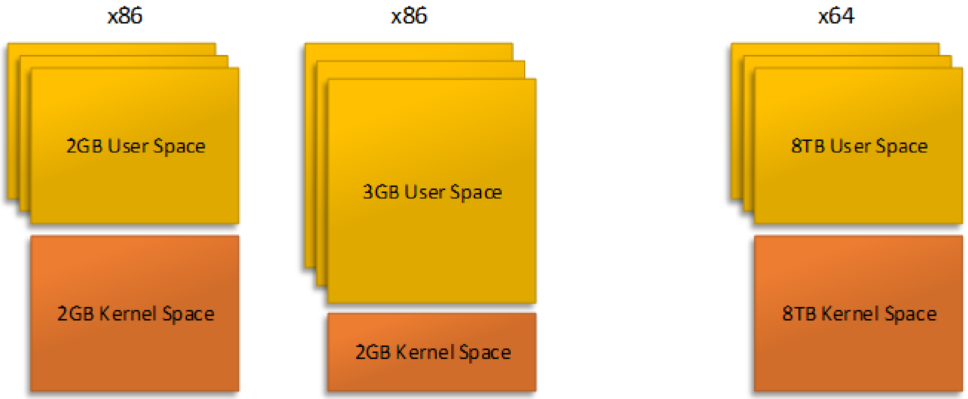
- Each process has its own large, private address space
- Windows uses 32-bit protected mode or 64-bit mode
- The memory manager maps virtual memory to physical memory at run time



- If physical memory is low, memory can be saved to disk (in page file) and loaded back when needed, transparent to applications (you can play with SysInternals' testlimit.exe)

Virtual Memory

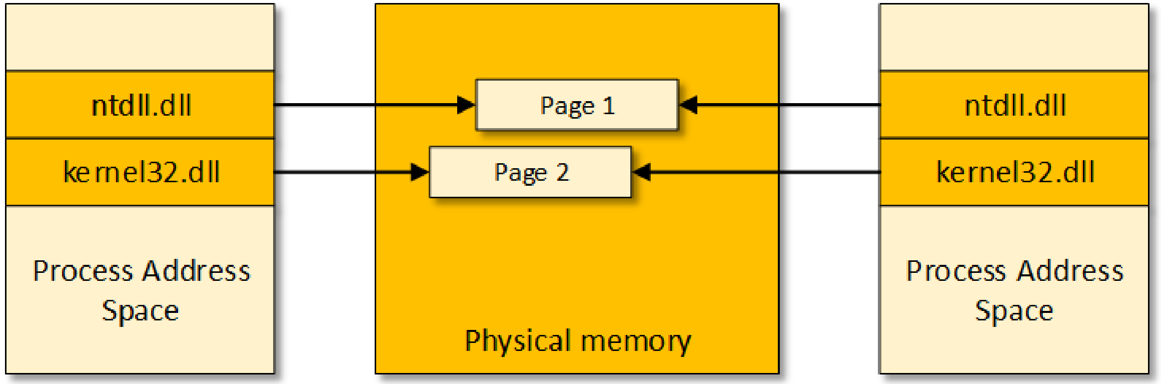
- Virtual memory layout



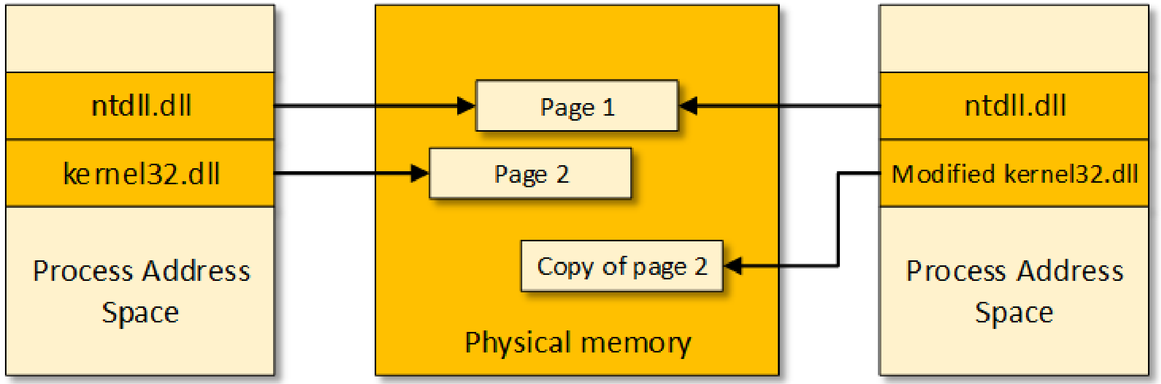
General memory limits	32-bit	64-bit
Total VA space	4GB	8TB 128TB*
VA space per 32-bit process	2GB 3GB with /3GB switch and flag**	2GB 4GB with flag**
VA space per 64-bit process		8TB 128TB*

* Windows 8.1, Windows Server 2012 R2 or later
** IMAGE_FILE_LARGE_ADDRESS_AWARE flag in FileHeader

Copy on Write

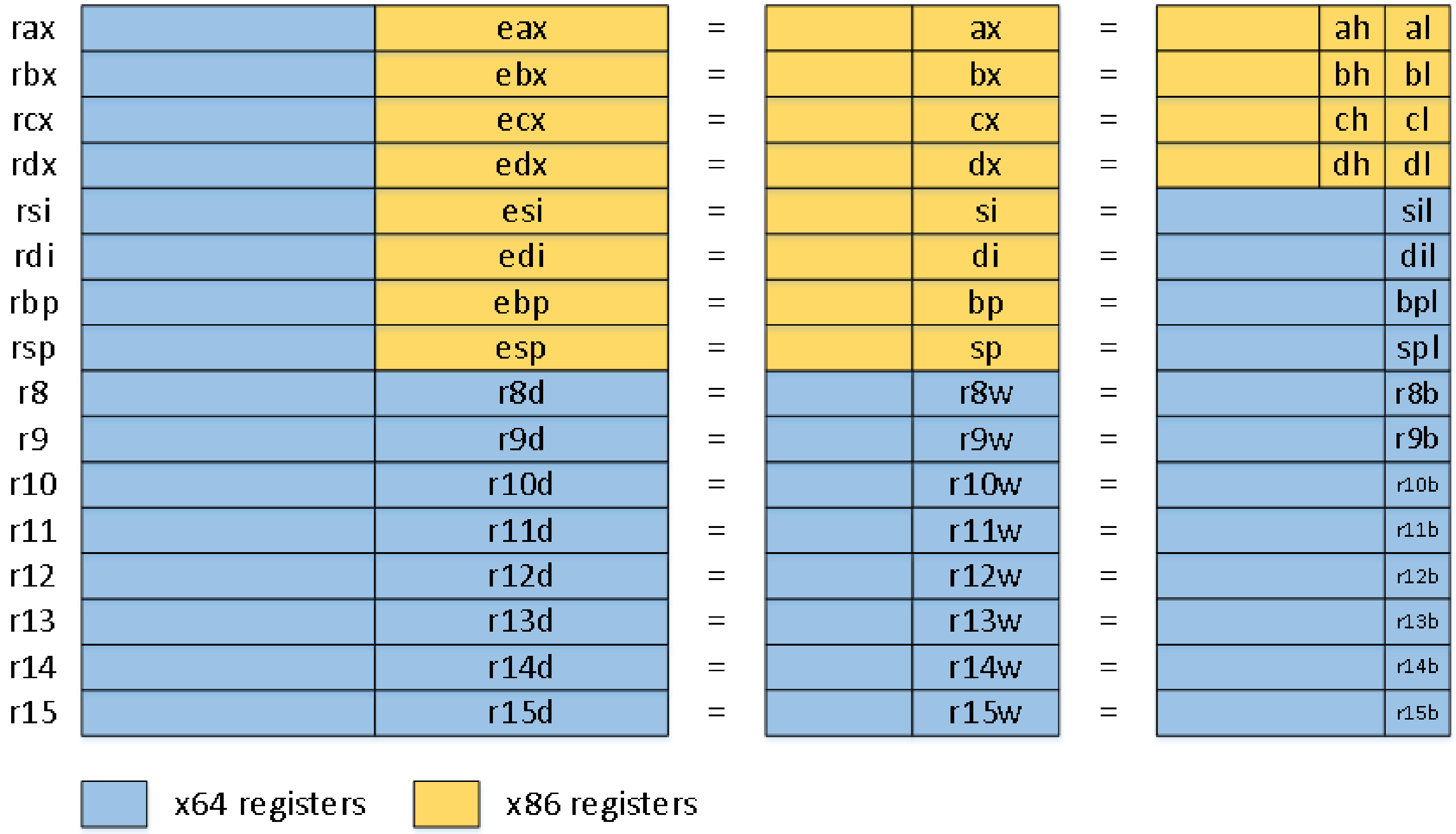


Before, kernel32.dll not modified, same memory mapped in processes



After, one process modifies kernel32.dll, another copy is made in physical memory

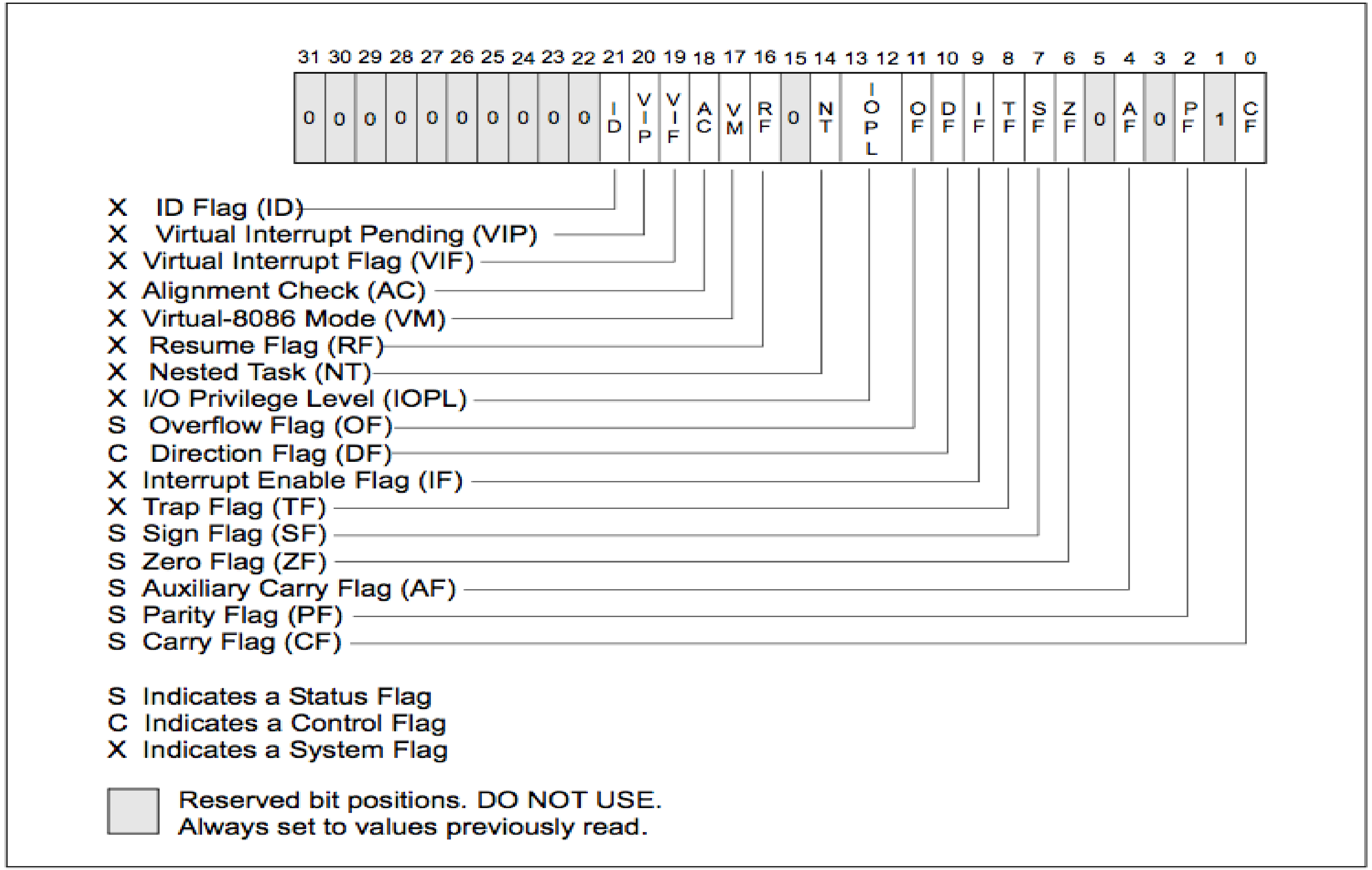
x86-x64 processors - Register set (general purpose)



rax	🟦	🟨 eax	=	🟨	🟨 ax	=	🟨	🟨 ah	🟨 al
rbx	🟦	🟨 ebx	=	🟨	🟨 bx	=	🟨	🟨 bh	🟨 bl
rcx	🟦	🟨 ecx	=	🟨	🟨 cx	=	🟨	🟨 ch	🟨 cl
rdx	🟦	🟨 edx	=	🟨	🟨 dx	=	🟨	🟨 dh	🟨 dl
rsi	🟦	🟨 esi	=	🟨	🟨 si	=	🟦		🟦 sil
rdi	🟦	🟨 edi	=	🟨	🟨 di	=	🟦		🟦 dil
rbp	🟦	🟨 ebp	=	🟨	🟨 bp	=	🟦		🟦 bpl
rsp	🟦	🟨 esp	=	🟨	🟨 sp	=	🟦		🟦 spl
r8	🟦	🟦 r8d	=	🟦	🟦 r8w	=	🟦		🟦 r8b
r9	🟦	🟦 r9d	=	🟦	🟦 r9w	=	🟦		🟦 r9b
r10	🟦	🟦 r10d	=	🟦	🟦 r10w	=	🟦		🟦 r10b
r11	🟦	🟦 r11d	=	🟦	🟦 r11w	=	🟦		🟦 r11b
r12	🟦	🟦 r12d	=	🟦	🟦 r12w	=	🟦		🟦 r12b
r13	🟦	🟦 r13d	=	🟦	🟦 r13w	=	🟦		🟦 r13b
r14	🟦	🟦 r14d	=	🟦	🟦 r14w	=	🟦		🟦 r14b
r15	🟦	🟦 r15d	=	🟦	🟦 r15w	=	🟦		🟦 r15b

🟦 x64 registers 🟨 x86 registers

Flags register



31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	0	0	0	0	0	0	ID	VIP	VIF	AC	VM	RF	0	NT	IOPL		OF	DF	IF	TF	SF	ZF	0	AF	0	PF	1	CF

X ID Flag (ID)
X Virtual Interrupt Pending (VIP)
X Virtual Interrupt Flag (VIF)
X Alignment Check (AC)
X Virtual-8086 Mode (VM)
X Resume Flag (RF)
X Nested Task (NT)
X I/O Privilege Level (IOPL)
S Overflow Flag (OF)
C Direction Flag (DF)
X Interrupt Enable Flag (IF)
X Trap Flag (TF)
S Sign Flag (SF)
S Zero Flag (ZF)
S Auxiliary Carry Flag (AF)
S Parity Flag (PF)
S Carry Flag (CF)

S Indicates a Status Flag
C Indicates a Control Flag
X Indicates a System Flag

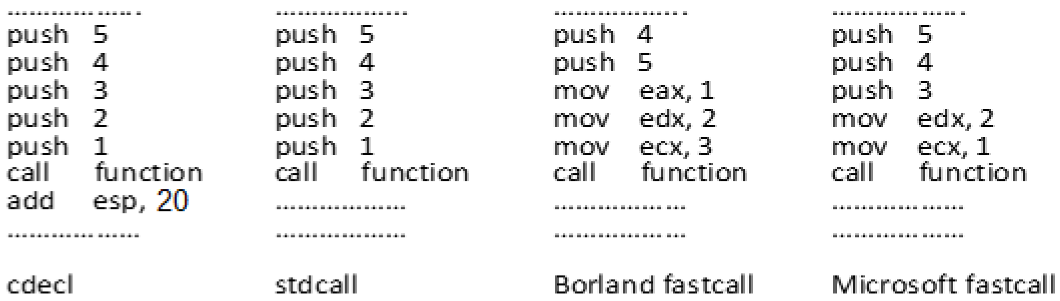
▨ Reserved bit positions. DO NOT USE.
Always set to values previously read.

Assembly Language

- Transfer instructions
  - MOV, PUSH, POP, XCHG, LEA, MOVS
  - Do not change the flags
- Arithmetic and logical instructions
  - ADD, XOR, CMP, TEST, SHL
  - Change the flags
- Flow control instructions
  - Conditional jumps, based on the flags
  - JMP, CALL, RET, LOOP
- ESP contains pointer to the top of the stack
  - PUSH val ⇔ ESP=ESP-4 [ESP]=val
  - POP val ⇔ val=[ESP] ESP=ESP+4
  - CALL addr ⇔ PUSH retaddr EIP=addr
  - RET ⇔ POP EIP

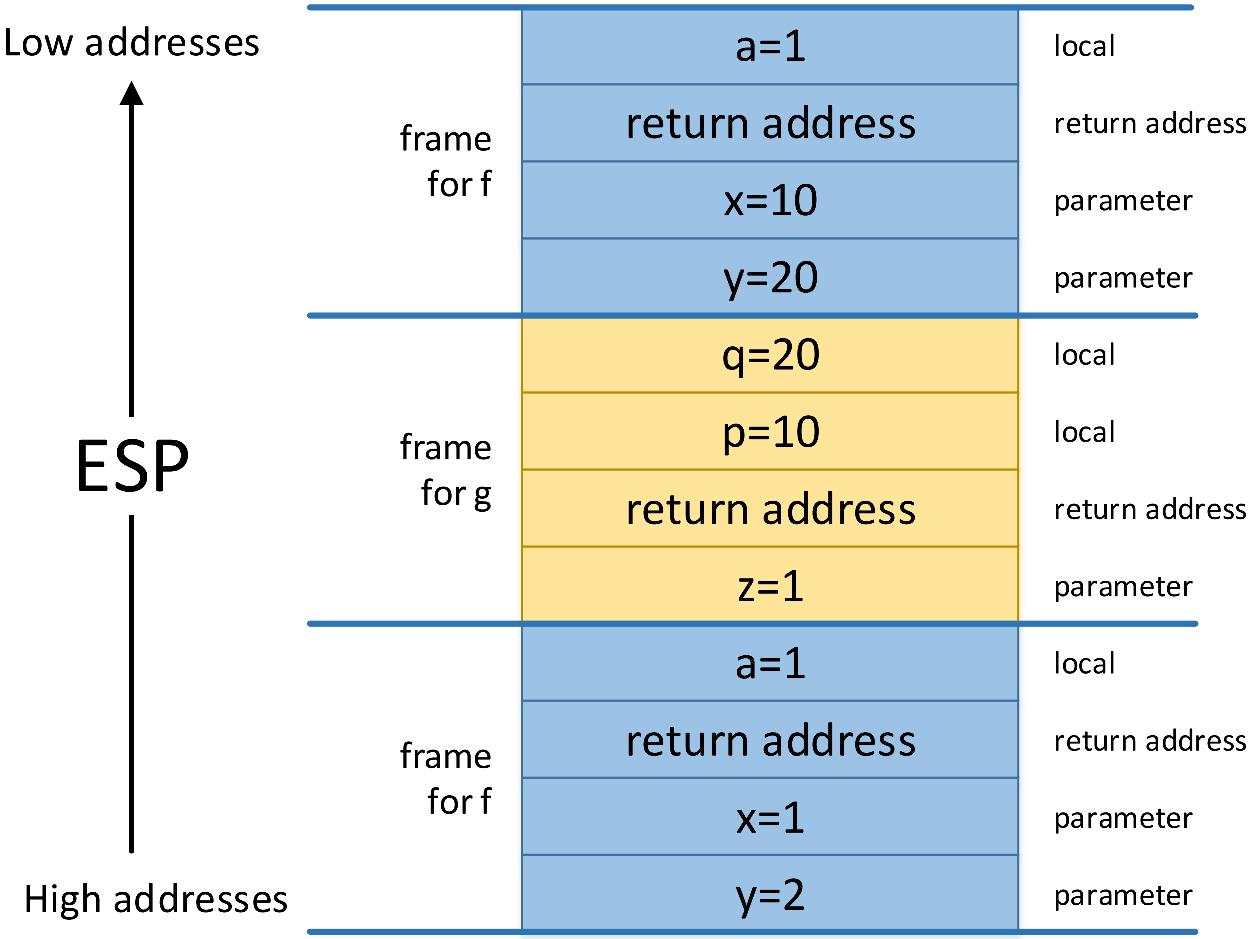
Calling Conventions (x86)

- cdecl
  - Arguments passed on stack, right to left. The caller cleans the arguments from the stack, allowing variable argument lists (e.g. printf)
- stdcall
  - Arguments passed on stack, right to left. The callee cleans the stack
- Borland fastcall
  - First three arguments in EAX, EDX, ECX, remaining arguments pushed on stack. All arguments in left to right order. The callee cleans the stack
- Microsoft fastcall
  - First two arguments (left to right) in ECX, EDX. Remaining arguments pushed on stack right to left. The callee cleans the stack



```
..................   ..................   ..................   ..................
push   5             push   5             push   4             push   5
push   4             push   4             push   5             push   4
push   3             push   3             mov    eax, 1        push   3
push   2             push   2             mov    edx, 2        mov    edx, 2
push   1             push   1             mov    ecx, 3        mov    ecx, 1
call   function      call   function      call   function      call   function
add    esp, 20       ..................   ..................   ..................
..................   ..................   ..................   ..................

cdecl                stdcall              Borland fastcall     Microsoft fastcall
```

Function Stack Frames (x86)



Low addresses

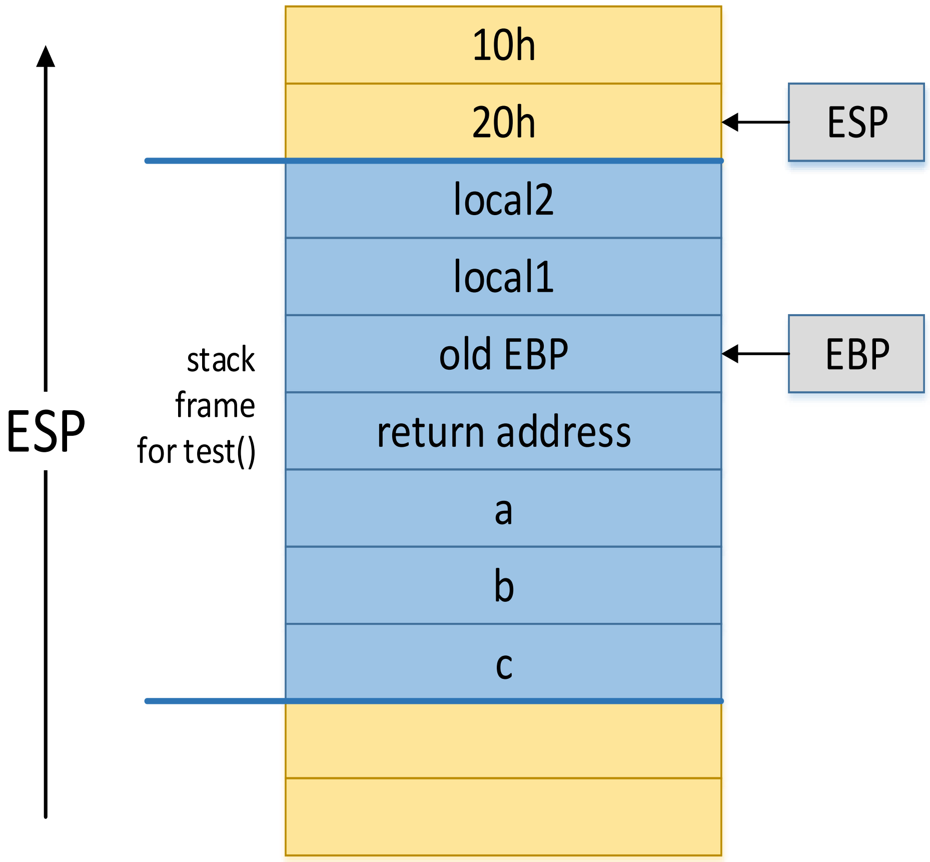
ESP

High addresses

frame for f	a=1	local
	return address	return address
	x=10	parameter
	y=20	parameter
frame for g	q=20	local
	p=10	local
	return address	return address
	z=1	parameter
frame for f	a=1	local
	return address	return address
	x=1	parameter
	y=2	parameter

```
function f(int x, int y)
{
        int a = 1;
        return g(a);
}

function g(int z)
{
        int p=10, q=20;
        return f(p, q);
}
……………………………
……………………………

f(1, 2);
```

Function Stack Frames (x86)



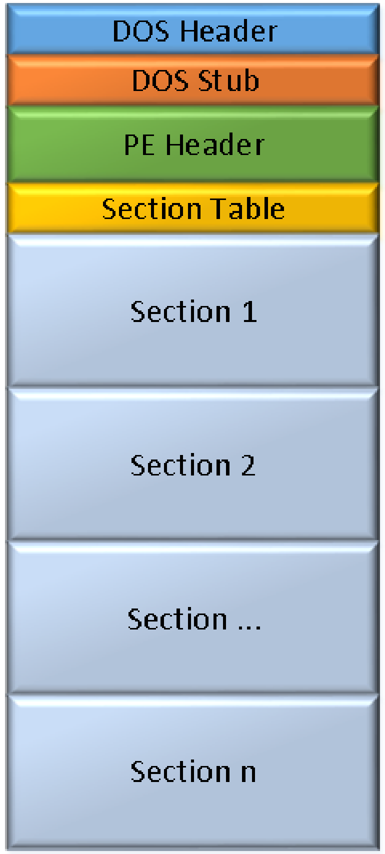
```
function test(int a, int b, int c)
{
        int local1, local2;
        ..................
        ..................
        return local1;
}

push    c
push    b
push    a
call    test
```

```
test proc
        push    ebp
        mov     ebp,esp
        sub     esp, 8
        ................
        mov     eax, [ebp+8]
        mov     ebx, [ebp+0Ch]
        mov     ecx, [ebp+10h]
        ................
        mov     eax, [ebp-4]
        mov     ebx, [ebp-8]
        ................
        push    20h
        push    10h
        call    addFunc
        ................
        mov     eax, [ebp-4]
        mov     esp, ebp
        pop     ebp
        ret     0Ch
test endp
```

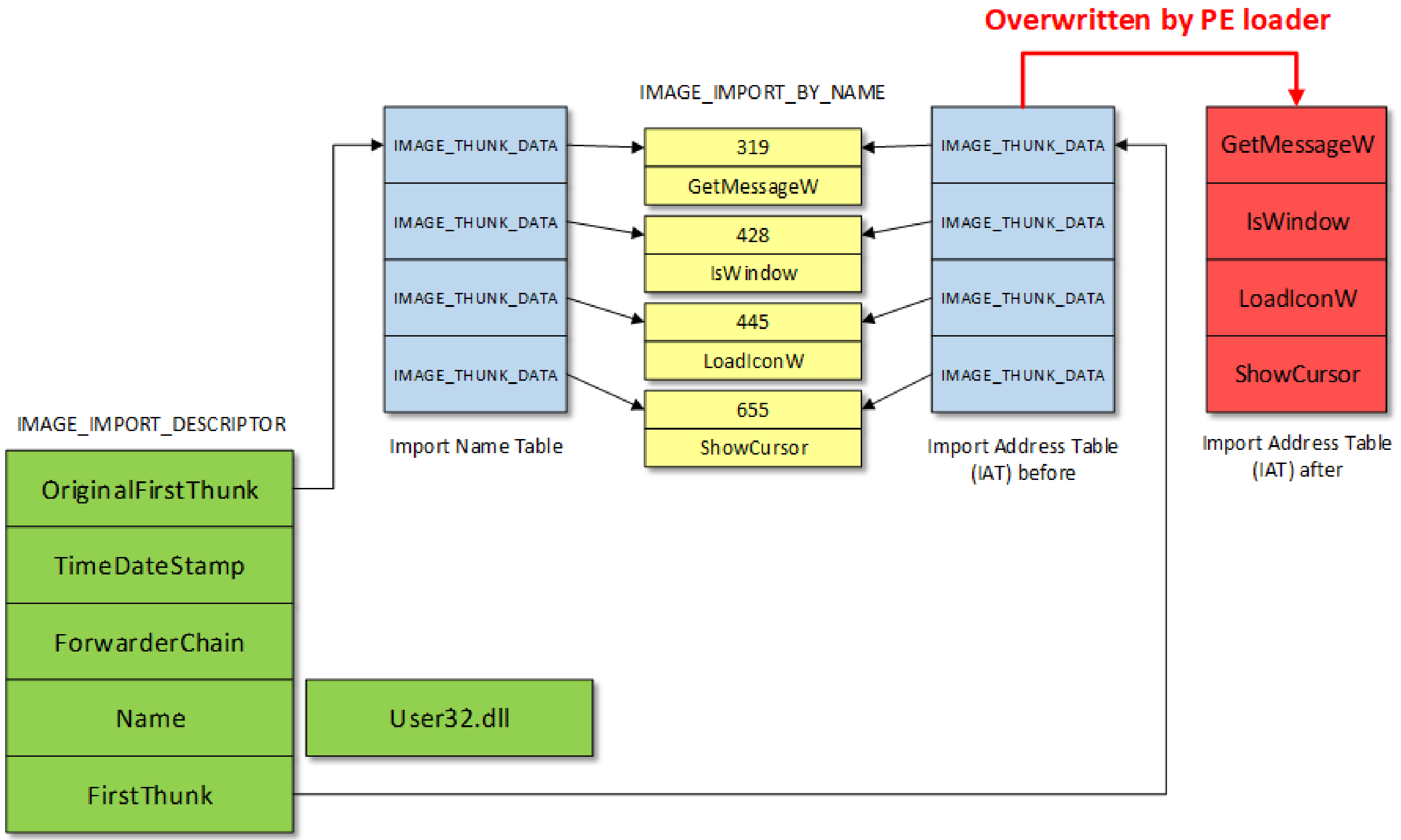
MZ/PE - Basic Structure/Concepts

- File Address
  - Address in file (as saved on disk)
- ImageBase
  - Address of the beginning of the file loaded in memory

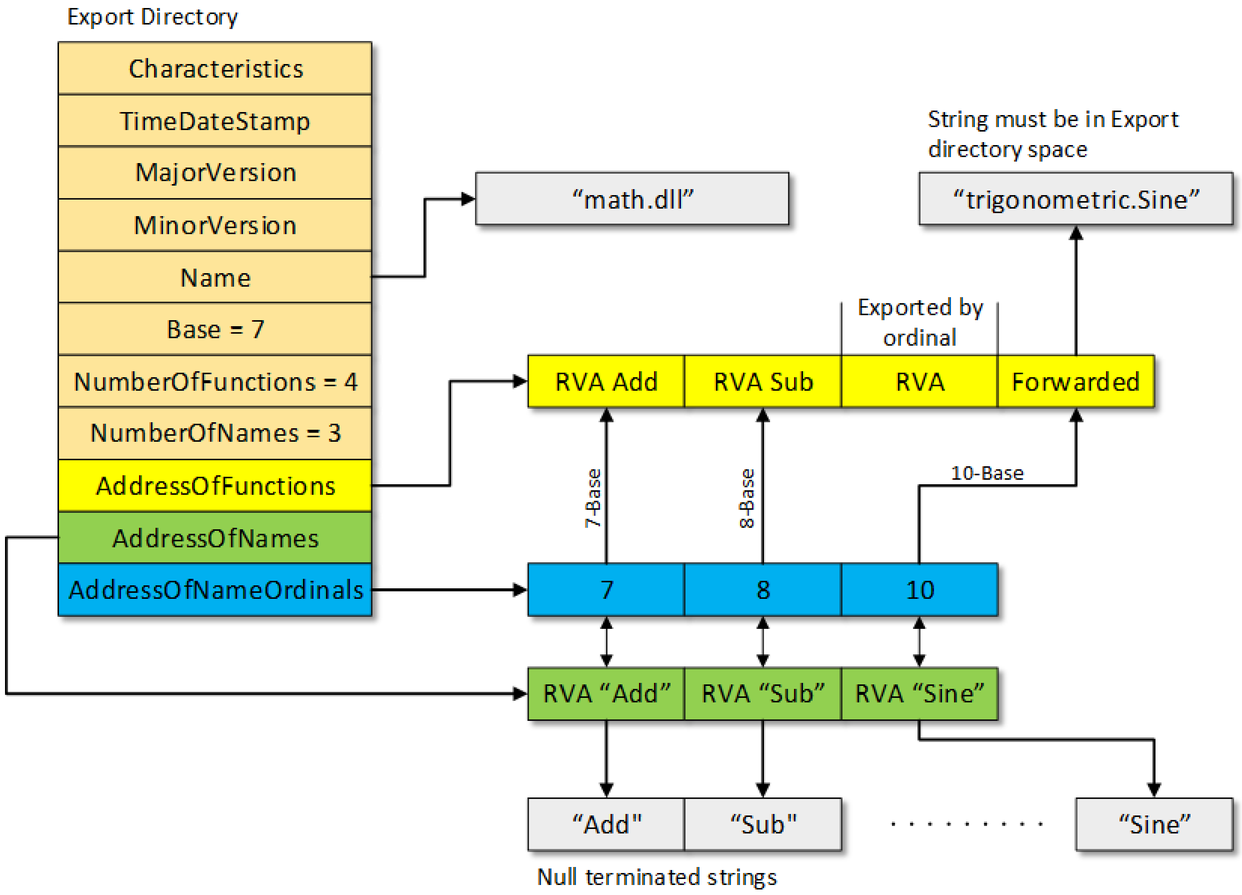
  (this is a Virtual Address)
- Relative Virtual Address (RVA)
  - Offset from ImageBase
- Virtual Address (VA)
  - Full memory address
- File in memory differs from file on disk
  - File alignment/Memory alignment
  - Sections aligned to page size (4KB) in memory
  - On disk aligned to smaller size (usually 512B) to save space



Import Directory



Export Directory



Export Directory

Characteristics

TimeDateStamp

MajorVersion

MinorVersion

Name

Base = 7

NumberOfFunctions = 4

NumberOfNames = 3

AddressOfFunctions

AddressOfNames

AddressOfNameOrdinals