Microkernel Construction

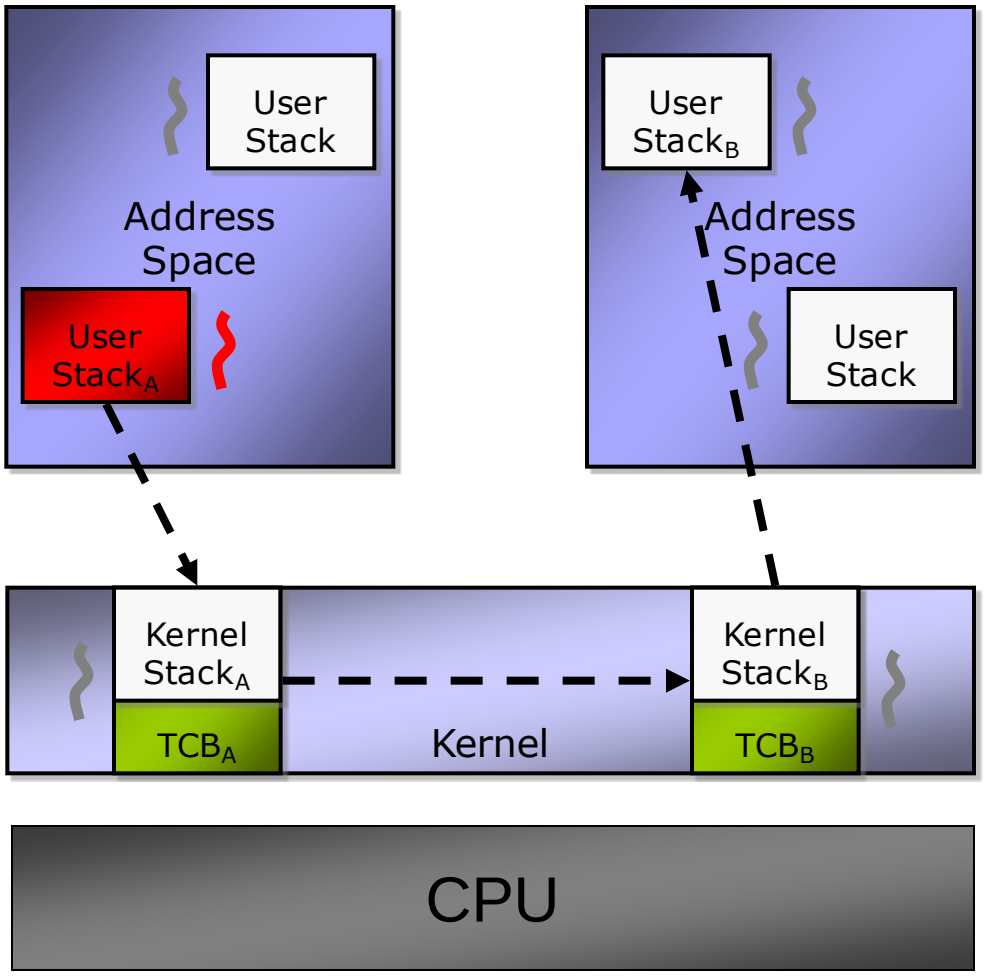
Kernel Entry / Exit

SS2013

Benjamin Engel

TU Dresden
Operating Systems Group

Control Transfer

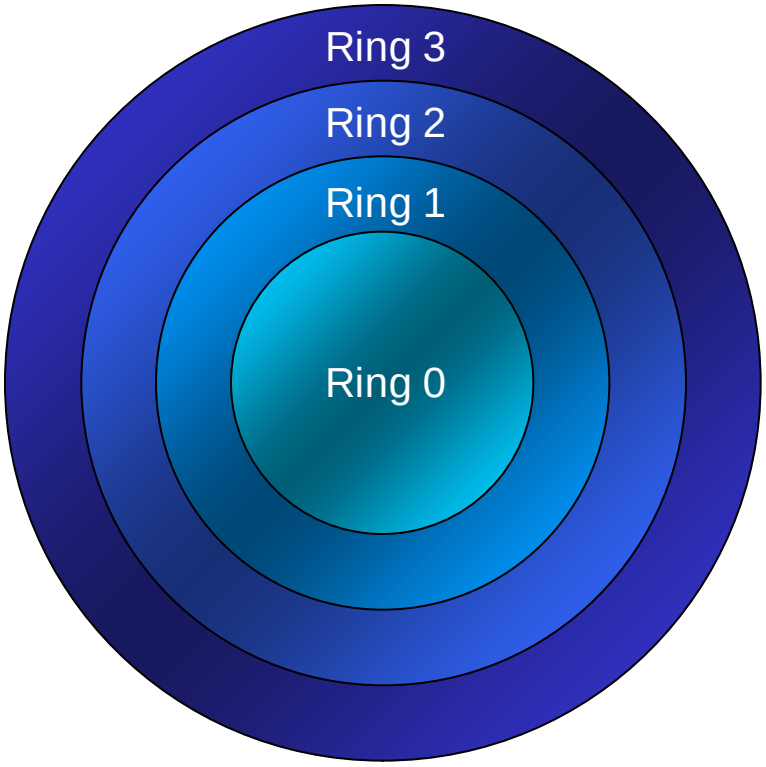


1. Kernel Entry (A)
2. Thread Switch (A -> B)
3. Kernel Exit (B)

Udo Steinberg

TU Dresden
Operating
Systems Group

Privilege Levels



- Ring 0
  - Operating System Kernel
  - privileged
- Rings 1 & 2
  - Operating System Services
  - unprivileged
- Ring 3
  - User-Level Applications
  - unprivileged

- Rings 1, 2 rarely used

Udo Steinberg

TU Dresden
Operating
Systems Group

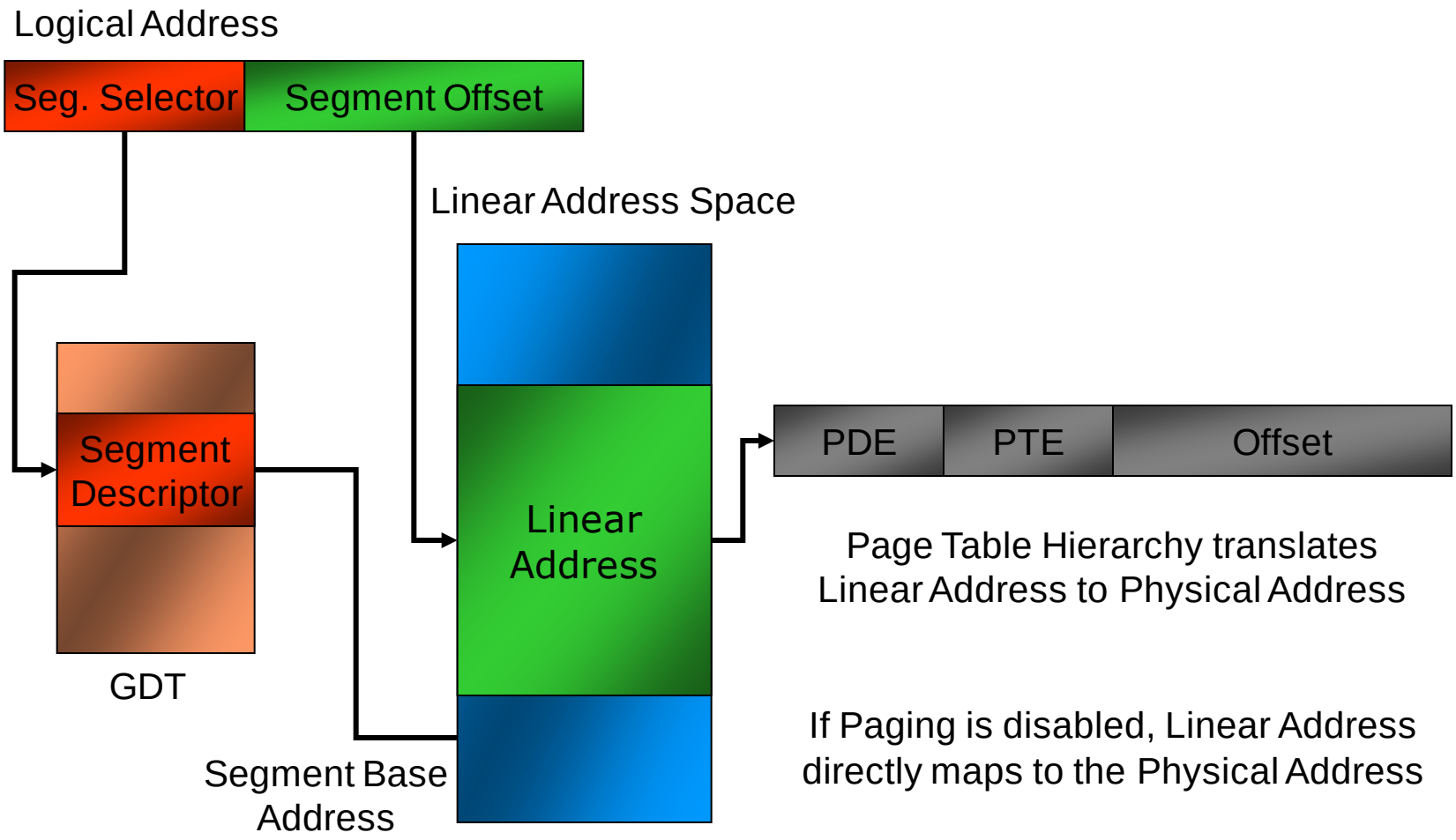
Protection Facilities

- Segmentation
  - Mechanism for dividing linear address space into segments
  - Different Segment Types: Code, Data, Stack, ...
  - Segment Base Address, Segment Limit
  - Mandatory mechanism, cannot be disabled
- Paging
  - Mechanism for translating linear to physical addresses
  - Per-page access rights (present, writable, user/superuser)
  - Implements virtual memory
  - Optional, can be turned off

Udo Steinberg

TU Dresden
Operating
Systems Group

Segmentation and Paging



Udo Steinberg

TU Dresden
Operating
Systems Group

Logical Address Translation

- Logical address consists of
  - segment selector
  - segment offset
- CPU uses table indicator (TI) in segment selector to access segment descriptor in GDT or LDT
- Segment descriptor provides segment base address, segment limit and access rights (checked by CPU)
- CPU adds segment offset to segment base address to form linear address

Udo Steinberg

TU Dresden
Operating
Systems Group

Segment Registers

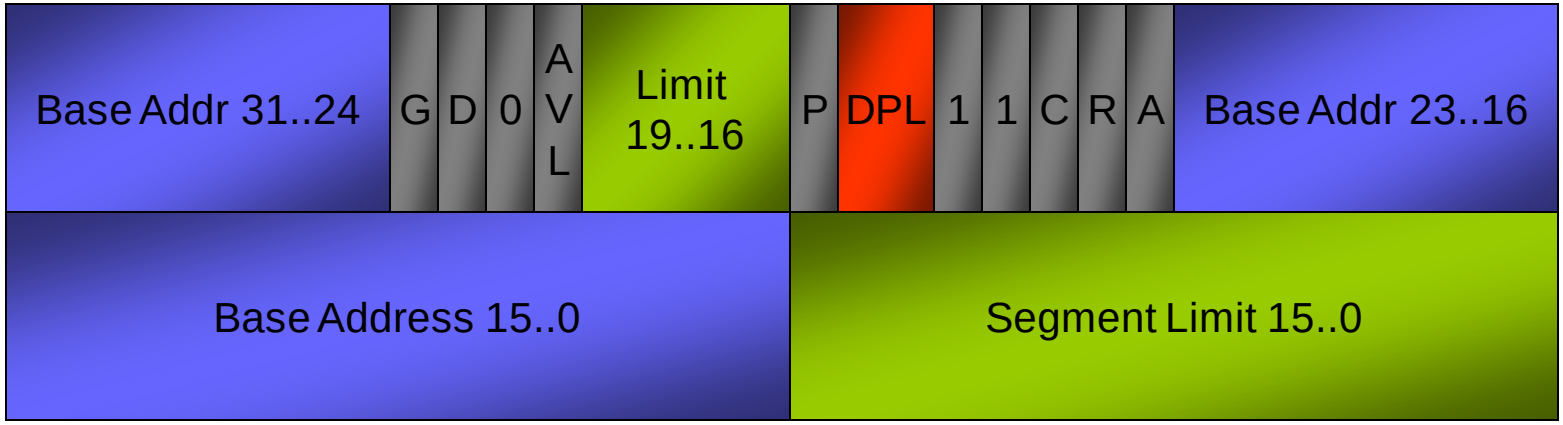
- Processor provides 6 segment registers for holding segment selectors
  - CS (code segment)
  - DS (data segment)
  - SS (stack segment)
  - ES, FS, GS (extra data segments)
- Selector forms visible part; segment base address, limit and access rights form shadow part of the segment register (invisible)

Udo Steinberg

TU Dresden
Operating
Systems Group

Segment Descriptor

Code Segment Descriptor



G	Granularity	DPL	Descriptor Privilege Level
D	Default Size (16/32 Bit)	C	Conforming
AVL	Available to Programmer	R	Readable
P	Present	A	Accessed

Udo Steinberg

TU Dresden
Operating
Systems Group

Flat Memory Model

- Hide segmentation mechanism by
  - setting segment base address to 0
  - setting segment size (limit) to 4 GB
- We need at least 2 segments
  - code segment
  - data/stack segment
- If kernel/user use different segment base addresses or segment limits then we need 2 segments for each
  - e.g., Small Address Spaces

Udo Steinberg

TU Dresden
Operating
Systems Group

Protection Checks

- CPU checks on instructions and memory references that certain protection conditions hold:
  - Segment Limit Check
  - Segment Type Check
  - Privilege Level Check
  - Restricted addressable space
  - Restricted procedure entry points
  - Restricted instruction set
- CPU raises exception if protection check fails
- Protection checks in parallel with address translation
- No big performance penalty

Udo Steinberg

TU Dresden
Operating
Systems Group

Privileged Instructions

- LGDT
- LLDT
- LTR
- LIDT
- MOV to %CR
- LMSW
- CLTS
- MOV to %DR
- INVD
- WBINVD
- INVLPG
- HLT
- RDMSR / WRMSR
- RDPMC / RDTSC
- CLI / STI (depending on IOPL)

Udo Steinberg

TU Dresden
Operating
Systems Group

Interrupts & Exceptions

- Forced transfer of execution from currently executing code to interrupt- or exception handler
- Hardware-Interrupts usually occur asynchronously in response to hardware events
- Exceptions (incl. Software Interrupts) occur synchronously while executing an instruction

Udo Steinberg

TU Dresden
Operating
Systems Group

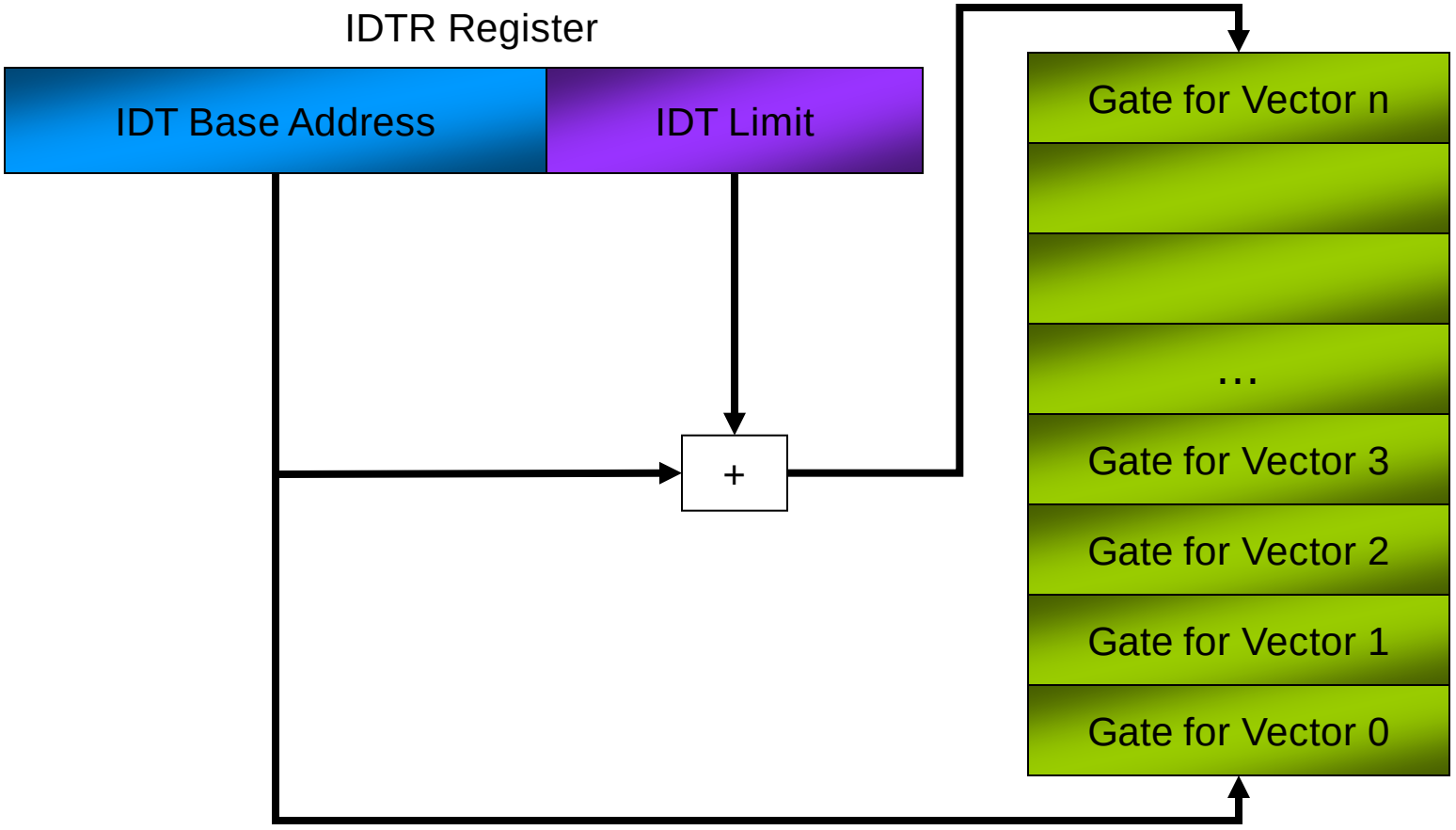
Interrupt / Exception Vectors

- Vector uniquely identifies the source of the interrupt or exception (0 .. 255)
  - vectors 0 .. 31 reserved (exceptions)
  - vectors 32 .. 255 designated user vectors (interrupts)
- Interrupt Descriptor Table (IDT) is table of handler functions for all interrupts and exceptions
  - vector = offset into table

Udo Steinberg

TU Dresden
Operating
Systems Group

Interrupt Descriptor Table (IDT)



Udo Steinberg

TU Dresden
Operating
Systems Group

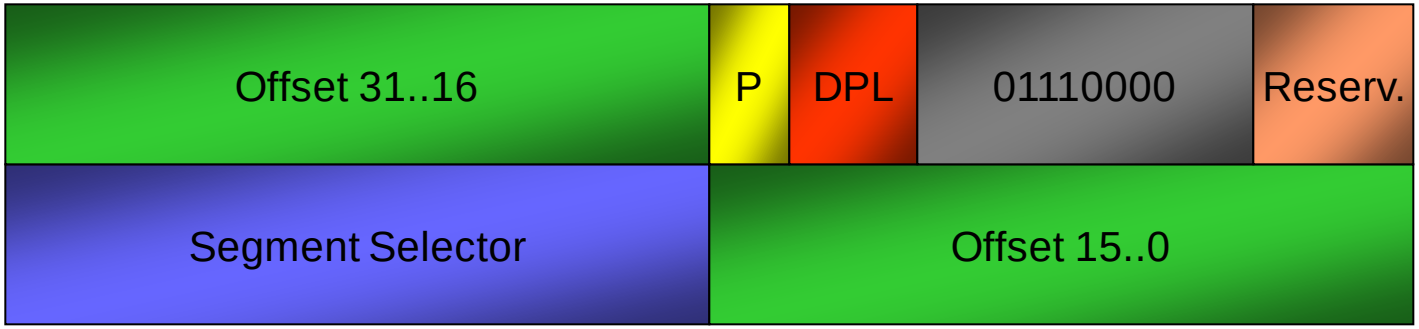
Restricted Procedure Entry Points: Gates

- Special descriptors for protected gateways to procedures at different privilege levels
  - Subject to CPL/RPL vs. DPL checking
  - CS and EIP loaded from descriptor
  - Interrupt Gate disables interrupts, Trap Gate doesn‘t
- When switching privilege level, new ESP loaded from Task State Segment
  - TSS contains stack pointers for privilege levels 0,1,2
  - Kernel updates ESP0 in TSS on every thread switch to point to the current thread‘s kernel stack
  - Interrupt and exception handlers runs in the context and on the stack of the currently executing thread

Udo Steinberg

TU Dresden
Operating
Systems Group

Interrupt Gate



Selector	Segment Selector for Destination Code Segment
Offset	Offset to Procedure Entry Point in Segment
DPL	Descriptor Privilege Level (ignored for HW ints)
P	Segment Present Flag

Udo Steinberg

TU Dresden
Operating
Systems Group

Exception Types

- Faults
  - Can be corrected and allow the program to be restarted
  - Processor restores machine state prior to the execution of faulting instruction (CS and EIP point to the faulting instruction)
- Traps
  - Are reported immediately after execution of trapping instruction
  - Allow the program to be restarted (CS and EIP point to the instruction following the faulting instruction)
- Aborts
  - Are not always reported at the precise location of the exception
  - Do not allow the restart of the program
  - Usually report severe hardware errors or inconsistent state

Udo Steinberg

TU Dresden
Operating
Systems Group

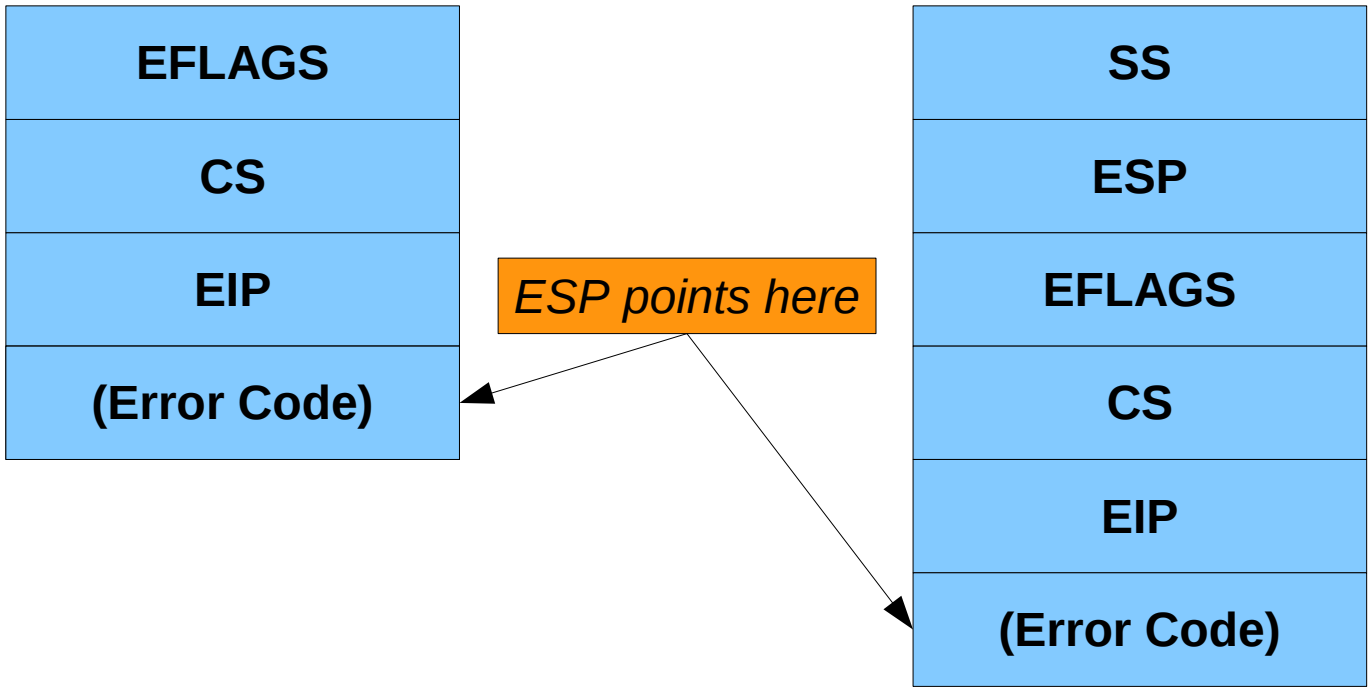
x86 Exceptions

- 0 Divide Error (F)
- 1 Debug Exception (F/T)
- 2 NMI
- 3 Breakpoint (T)
- 4 Overflow (T)
- 5 Bound Range Exceeded (F)
- 6 Invalid Opcode (F)
- 7 Device Not Available (F)
- 8 Double Fault (A)
- 9 Coprocessor Seg. Overrun (F)
- 10 Invalid TSS (F)
- 11 Segment Not Present (F)
- 12 Stack Segment Fault (F)
- 13 General Protection Fault (F)
- 14 Page Fault (F)
- 15 reserved
- 16 FPU Math Fault (F)
- 17 Alignment Check (F)
- 18 Machine Check (A)
- 19 SIMD FP Exception (F)

Udo Steinberg

TU Dresden
Operating
Systems Group

Kernel Stack after Kernel Entry



No Priviledge-Level Change

Priviledge-Level Change

Benjamin Engel

TU Dresden
Operating Systems Group

Kernel Stack after Kernel Entry


```
IDT_vector          push vector
entry_exc:          push $0
entry_exc_err:      SAVE_STATE
                    mov %cr2, %eax
                    mov %eax, 0xc(%ebx)
                    mov %ebx, %eax
                    call exc_handler
                    jmp ret_from_interupt

SAVE_STATE          push %ds
                    push %es
                    push %fs
                    push %gs
                    pusha
                    mov %esp, %ebx
                    mov $KSTCK_BEGIN, %esp
```

Benjamin Engel

TU Dresden
Operating Systems Group

Kernel Stack after Kernel Entry


```
IDT_vector          push vector
entry_exc:          push $0
entry_exc_err:      SAVE_STATE
                    mov %cr2, %eax
                    mov %eax, 0xc(%ebx)
                    mov %ebx, %eax
                    call exc_handler
                    jmp ret_from_interupt

SAVE_STATE          push %ds
                    push %es
                    push %fs
                    push %gs
                    pusha
                    mov %esp, %ebx
                    mov $KSTCK_BEGIN, %esp
```

Benjamin Engel

TU Dresden Operating Systems Group

Kernel Stack after Kernel Entry

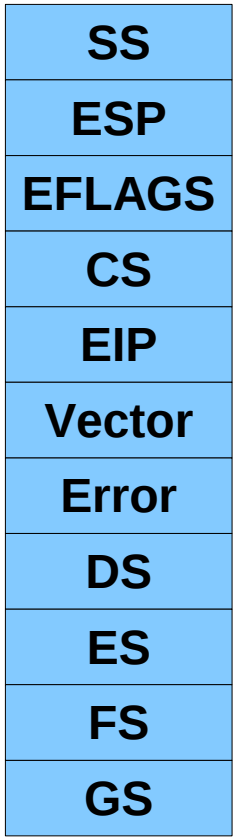

```
IDT_vector          push vector
entry_exc:          push $0
entry_exc_err:      SAVE_STATE
                    mov %cr2, %eax
                    mov %eax, 0xc(%ebx)
                    mov %ebx, %eax
                    call exc_handler
                    jmp ret_from_interupt

SAVE_STATE          push %ds
                    push %es
                    push %fs
                    push %gs
                    pusha
                    mov %esp, %ebx
                    mov $KSTCK_BEGIN, %esp
```

Benjamin Engel

TU Dresden
Operating Systems Group

Kernel Stack after Kernel Entry



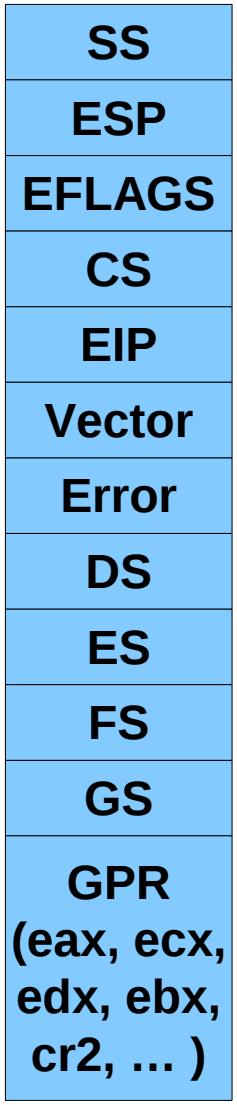

```
IDT_vector          push vector
entry_exc:          push $0
entry_exc_err:      SAVE_STATE
                    mov %cr2, %eax
                    mov %eax, 0xc(%ebx)
                    mov %ebx, %eax
                    call exc_handler
                    jmp ret_from_interupt

SAVE_STATE          push %ds
                    push %es
                    push %fs
                    push %gs
                    pusha
                    mov %esp, %ebx
                    mov $KSTCK_BEGIN, %esp
```

Benjamin Engel

TU Dresden Operating Systems Group

Kernel Stack after Kernel Entry



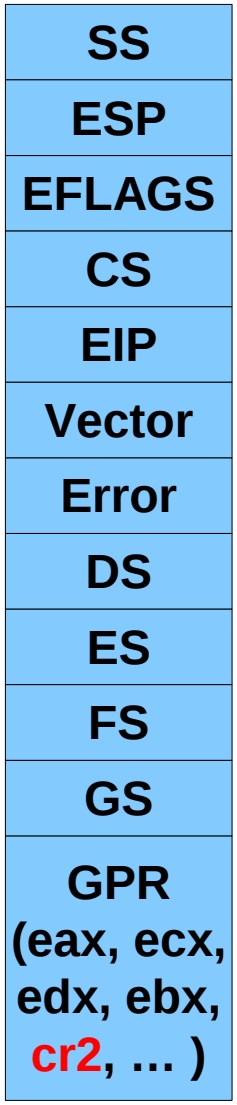

```
IDT_vector          push vector
entry_exc:          push $0
entry_exc_err:      SAVE_STATE
                    mov %cr2, %eax
                    mov %eax, 0xc(%ebx)
                    mov %ebx, %eax
                    call exc_handler
                    jmp ret_from_interupt

SAVE_STATE          push %ds
                    push %es
                    push %fs
                    push %gs
                    pusha
                    mov %esp, %ebx
                    mov $KSTCK_BEGIN, %esp
```

Benjamin Engel

TU Dresden Operating Systems Group

Kernel Stack after Kernel Entry



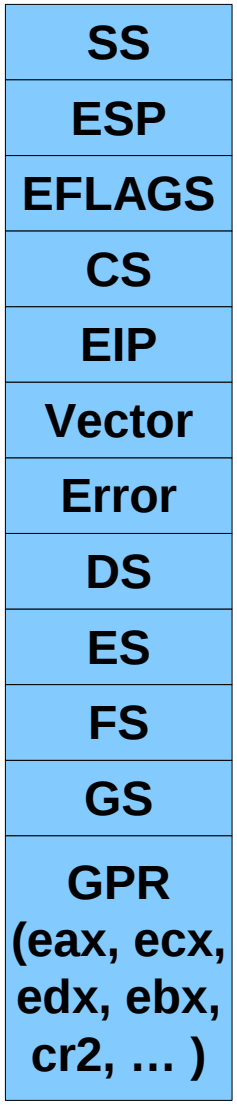

```
IDT_vector          push vector
entry_exc:          push $0
entry_exc_err:      SAVE_STATE
                    mov %cr2, %eax
                    mov %eax, 0xc(%ebx)
                    mov %ebx, %eax
                    call exc_handler
                    jmp ret_from_interupt

SAVE_STATE          push %ds
                    push %es
                    push %fs
                    push %gs
                    pusha
                    mov %esp, %ebx
                    mov $KSTCK_BEGIN, %esp
```

Benjamin Engel

TU Dresden Operating Systems Group

Kernel Stack after Kernel Entry




```
IDT_vector          push vector
entry_exc:          push $0
entry_exc_err:      SAVE_STATE
                    mov %cr2, %eax
                    mov %eax, 0xc(%ebx)
                    mov %ebx, %eax
                    call exc_handler
                    jmp ret_from_interupt

SAVE_STATE          push %ds
                    push %es
                    push %fs
                    push %gs
                    pusha
                    mov %esp, %ebx
                    mov $KSTCK_BEGIN, %esp
```

Benjamin Engel

TU Dresden Operating Systems Group

Exception Handler

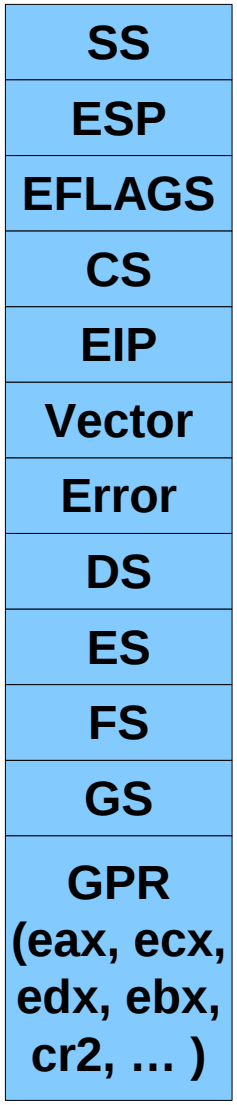
```
REGPARM(1)
static void exc_handler (Exc_regs * r) {
    switch (r->vec) {
        case PAGE_FAULT:
            handle_pf (r);
            return;
        case PROTECTION_FAULT:
            handle_gp (r);
            return;
    }
}

handle_pf (Exc_regs * r) {
    mword addr = r->cr2;
    // ...
}
```

Benjamin Engel

TU Dresden
Operating Systems Group

Kernel Exit (to Kernel Mode)



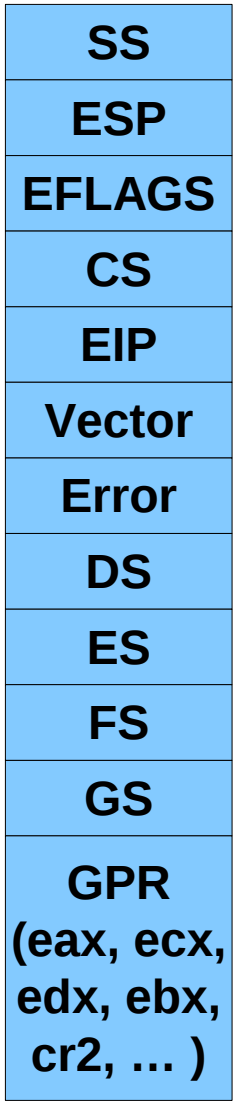

```
IDT_vector          push vector
entry_exc:          push $0
entry_exc_err:      SAVE_STATE
                    mov %cr2, %eax
                    mov %eax, 0xc(%ebx)
                    mov %ebx, %eax
                    call exc_handler
                    jmp ret_from_interupt

ret_from_interrupt:
                    testb $3, 0x3c(%ebx)
                    jnz ret_user_iret
                    popa
                    add $24, %esp
                    iret
```

Benjamin Engel

TU Dresden Operating Systems Group

Kernel Exit (to Kernel Mode)



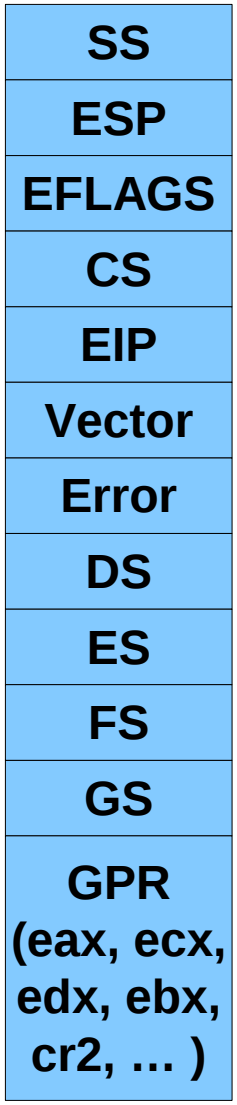

```
IDT_vector          push vector
entry_exc:          push $0
entry_exc_err:      SAVE_STATE
                    mov %cr2, %eax
                    mov %eax, 0xc(%ebx)
                    mov %ebx, %eax
                    call exc_handler
                    jmp ret_from_interupt

ret_from_interrupt:
                    testb $3, 0x3c(%ebx)
                    jnz ret_user_iret
                    popa
                    add $24, %esp
                    iret
```

Benjamin Engel

TU Dresden Operating Systems Group

Kernel Exit (to Kernel Mode)



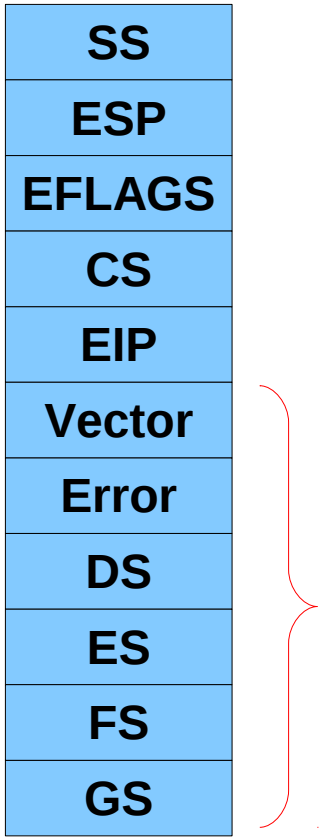

```
IDT_vector          push vector
entry_exc:          push $0
entry_exc_err:      SAVE_STATE
                    mov %cr2, %eax
                    mov %eax, 0xc(%ebx)
                    mov %ebx, %eax
                    call exc_handler
                    jmp ret_from_interupt

ret_from_interrupt:
                    testb $3, 0x3c(%ebx)
                    jnz ret_user_iret
                    popa
                    add $24, %esp
                    iret
```

Benjamin Engel

TU Dresden Operating Systems Group

Kernel Exit (to Kernel Mode)




```
IDT_vector          push vector
entry_exc:          push $0
entry_exc_err:      SAVE_STATE
                    mov %cr2, %eax
                    mov %eax, 0xc(%ebx)
                    mov %ebx, %eax
                    call exc_handler
                    jmp ret_from_interupt

ret_from_interrupt:
                    testb $3, 0x3c(%ebx)
                    jnz ret_user_iret
                    popa
                    add $24, %esp
                    iret
```

Benjamin Engel

TU Dresden Operating Systems Group

Kernel Exit (to Kernel Mode)


```
IDT_vector          push vector
entry_exc:          push $0
entry_exc_err:      SAVE_STATE
                    mov %cr2, %eax
                    mov %eax, 0xc(%ebx)
                    mov %ebx, %eax
                    call exc_handler
                    jmp ret_from_interupt

ret_from_interrupt:
                    testb $3, 0x3c(%ebx)
                    jnz ret_user_iret
                    popa
                    add $24, %esp
                    iret
```

Benjamin Engel

TU Dresden
Operating Systems Group

Kernel Exit (to Kernel Mode)

```
IDT_vector          push vector
entry_exc:          push $0
entry_exc_err:      SAVE_STATE
                    mov %cr2, %eax
                    mov %eax, 0xc(%ebx)
                    mov %ebx, %eax
                    call exc_handler
                    jmp ret_from_interupt

ret_from_interrupt:
                    testb $3, 0x3c(%ebx)
                    jnz ret_user_iret
                    popa
                    add $24, %esp
                    iret
```

Benjamin Engel

TU Dresden Operating Systems Group

Kernel Exit (to User Mode)

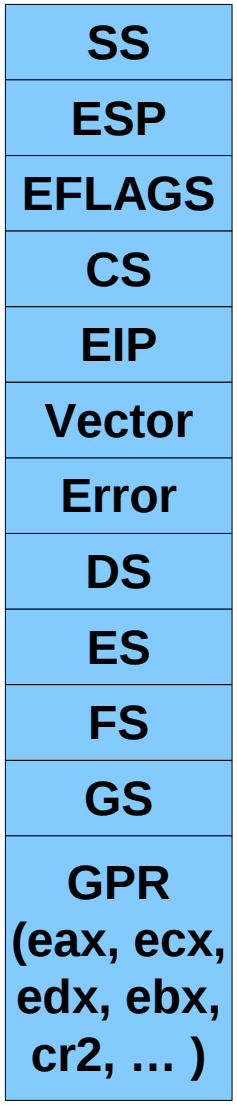
```
IDT_vector          push vector
entry_exc:          push $0
entry_exc_err:      SAVE_STATE
                    mov %cr2, %eax
                    mov %eax, 0xc(%ebx)
                    mov %ebx, %eax
                    call exc_handler
                    jmp ret_from_interupt

ret_from_interrupt:
                    testb $3, 0x3c(%ebx)
                    jnz ret_user_iret
                    popa
                    add $24, %esp
                    iret
```

Benjamin Engel

TU Dresden Operating Systems Group

Kernel Exit (to User Mode)

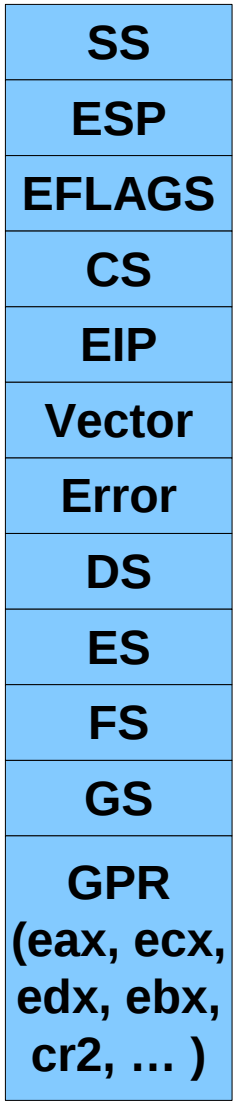



```
void ret_user_iret()
{
    asm volatile (
            "lea %0, %%esp      ;"
            "popa               ;"
            "pop %%gs           ;"
            "pop %%fs           ;"
            "pop %%es           ;"
            "pop %%ds           ;"
            "add $8, %%esp      ;"
            "iret               ;"
        : : "m" (register_file) : "memory");
}
```

Benjamin Engel

TU Dresden Operating Systems Group

Kernel Exit (to User Mode)

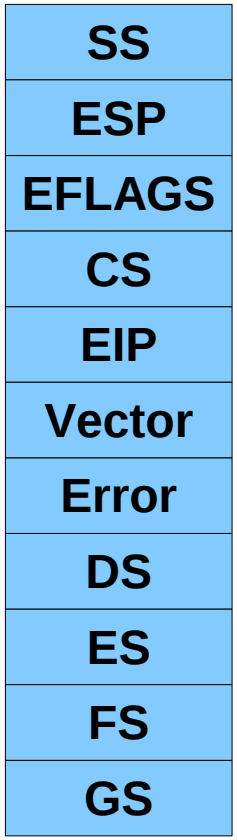



```
void ret_user_iret()
{
    asm volatile (
            "lea %0, %%esp      ;"
            "popa               ;"
            "pop %%gs           ;"
            "pop %%fs           ;"
            "pop %%es           ;"
            "pop %%ds           ;"
            "add $8, %%esp      ;"
            "iret               ;"
        : : "m" (register_file) : "memory");
}
```

Benjamin Engel

TU Dresden Operating Systems Group

Kernel Exit (to User Mode)

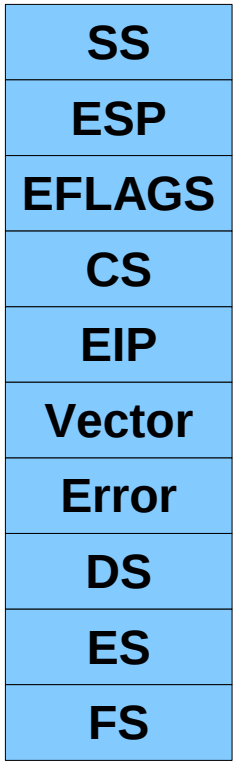



```
void ret_user_iret()
{
    asm volatile (
            "lea %0, %%esp      ;"
            "popa               ;"
            "pop %%gs           ;"
            "pop %%fs           ;"
            "pop %%es           ;"
            "pop %%ds           ;"
            "add $8, %%esp      ;"
            "iret               ;"
        : : "m" (register_file) : "memory");
}
```

Benjamin Engel

TU Dresden
Operating Systems Group

Kernel Exit (to User Mode)

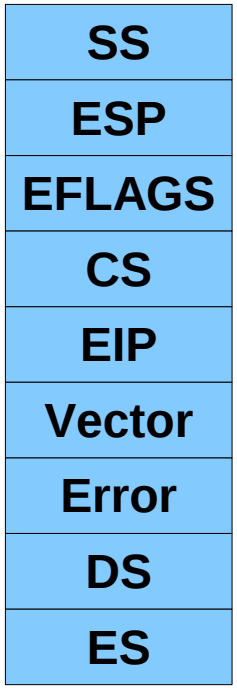



```
void ret_user_iret()
{
    asm volatile (
            "lea %0, %%esp      ;"
            "popa               ;"
            "pop %%gs           ;"
            "pop %%fs           ;"
            "pop %%es           ;"
            "pop %%ds           ;"
            "add $8, %%esp      ;"
            "iret               ;"
        : : "m" (register_file) : "memory");
}
```

Benjamin Engel

TU Dresden
Operating Systems Group

Kernel Exit (to User Mode)

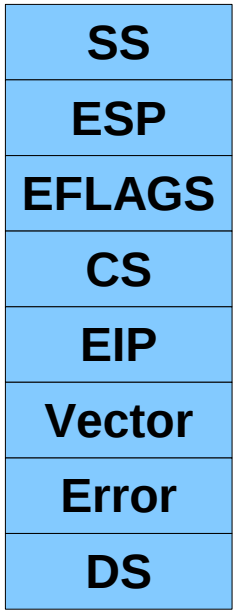



```
void ret_user_iret()
{
    asm volatile (
            "lea %0, %%esp      ;"
            "popa               ;"
            "pop %%gs           ;"
            "pop %%fs           ;"
            "pop %%es           ;"
            "pop %%ds           ;"
            "add $8, %%esp      ;"
            "iret               ;"
        : : "m" (register_file) : "memory");
}
```

Benjamin Engel

TU Dresden
Operating Systems Group

Kernel Exit (to User Mode)




```
void ret_user_iret()
{
    asm volatile (
            "lea %0, %%esp      ;"
            "popa               ;"
            "pop %%gs           ;"
            "pop %%fs           ;"
            "pop %%es           ;"
            "pop %%ds           ;"
            "add $8, %%esp      ;"
            "iret               ;"
        : : "m" (register_file) : "memory");
}
```

Benjamin Engel

TU Dresden
Operating Systems Group

Kernel Exit (to User Mode)


```
void ret_user_iret()
{
    asm volatile (
            "lea %0, %%esp      ;"
            "popa               ;"
            "pop %%gs           ;"
            "pop %%fs           ;"
            "pop %%es           ;"
            "pop %%ds           ;"
            "add $8, %%esp      ;"
            "iret               ;"
        : : "m" (register_file) : "memory");
}
```

Benjamin Engel

TU Dresden
Operating Systems Group

Kernel Exit (to User Mode)


```
void ret_user_iret()
{
    asm volatile (
            "lea %0, %%esp      ;"
            "popa               ;"
            "pop %%gs           ;"
            "pop %%fs           ;"
            "pop %%es           ;"
            "pop %%ds           ;"
            "add $8, %%esp      ;"
            "iret               ;"
        : : "m" (register_file) : "memory");
}
```

Kernel Exit (to User Mode)

```
void ret_user_iret()
{
    asm volatile (
            "lea %0, %%esp      ;"
            "popa               ;"
            "pop %%gs           ;"
            "pop %%fs           ;"
            "pop %%es           ;"
            "pop %%ds           ;"
            "add $8, %%esp      ;"
            "iret               ;"
        : : "m" (register_file) : "memory");
}
```

Kernel Exit : IRET

- Load Code Segment Selector, Instruction Pointer and Flags from Stack
- Evaluate RPL of CS from the Stack
  - RPL > CPL return to other priviledge level
  - otherwise return to same priviledge level
- If priviledge Level changes
  - Load Stack Segment Selector and Stack Pointer from Stack
  - Adjust CPL (current priviledge level)

Fast Kernel Entry: SYSENTER

- Fast ring transition from any ring to ring 0
- Kernel code entry point specified via
  - SYSENTER_CS_MSR (code segment)
  - SYSENTER_EIP_MSR (entry function)
  - SYSENTER_ESP_MSR (kernel stack ptr)
- Processor…
  - disables interrupts
  - loads kernel CS (from SYSENTER_CS_MSR)
  - loads kernel SS (from SYSENTER_CS_MSR + 8)
  - loads kernel ESP (from SYSENTER_ESP_MSR)
  - loads kernel EIP (from SYSENTER_EIP_MSR)
  - does NOT save user return EIP or other user state!

Udo Steinberg

TU Dresden
Operating
Systems Group

Fast Kernel Exit: SYSEXIT

- Fast ring transition from any ring to ring 3

- Processor…
  - loads user CS (from SYSENTER_CS_MSR + 16)
  - loads user SS (from SYSENTER_CS_MSR + 24)
  - loads user ESP (from ECX)
  - loads user EIP (from EDX)
  - enables interrupts

Udo Steinberg

TU Dresden
Operating
Systems Group