# Microkernel Construction

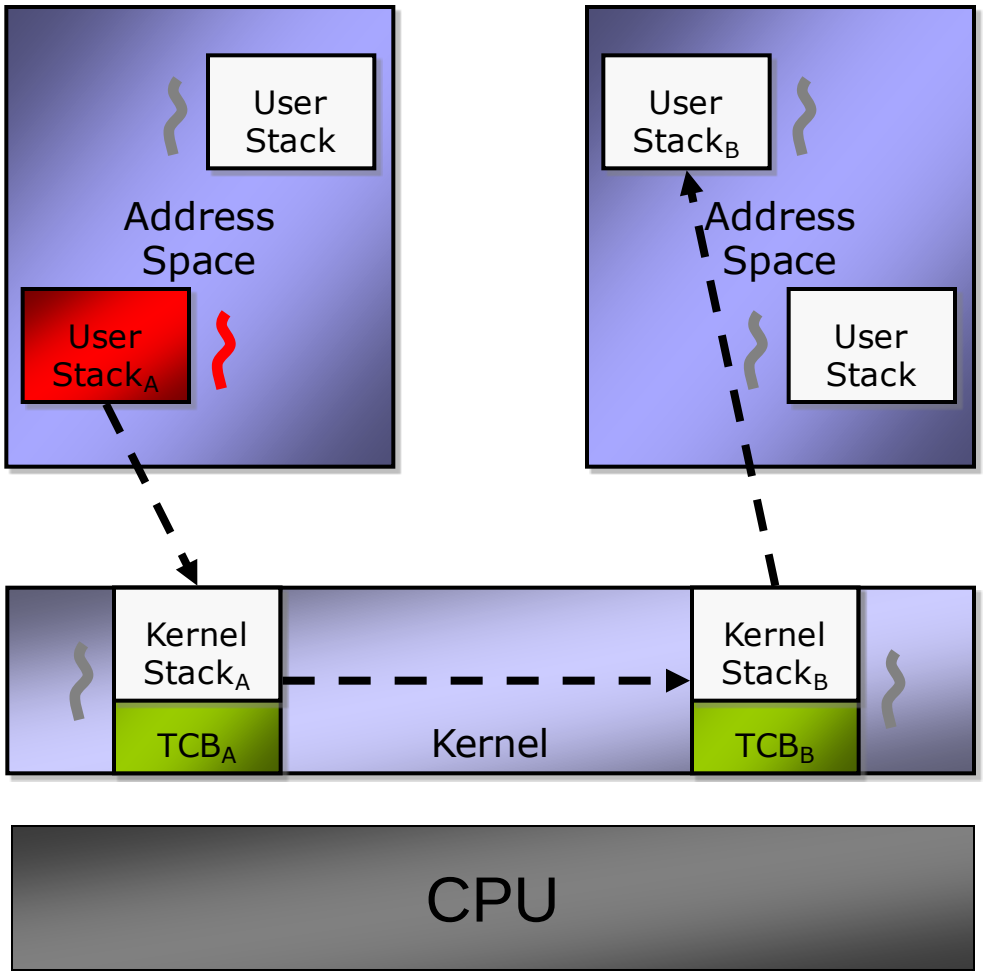
## Kernel Entry and Kernel Exit

SS2011

Udo Steinberg

TU Dresden
Operating
Systems Group

# Control Transfer

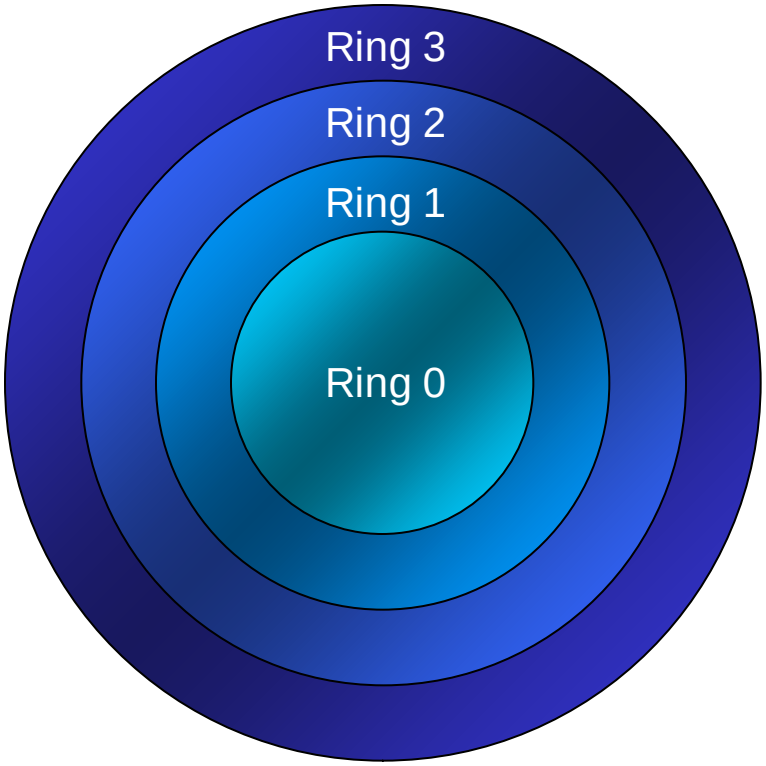



1. Kernel Entry (A)
2. Thread Switch (A -> B)
3. Kernel Exit (B)

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

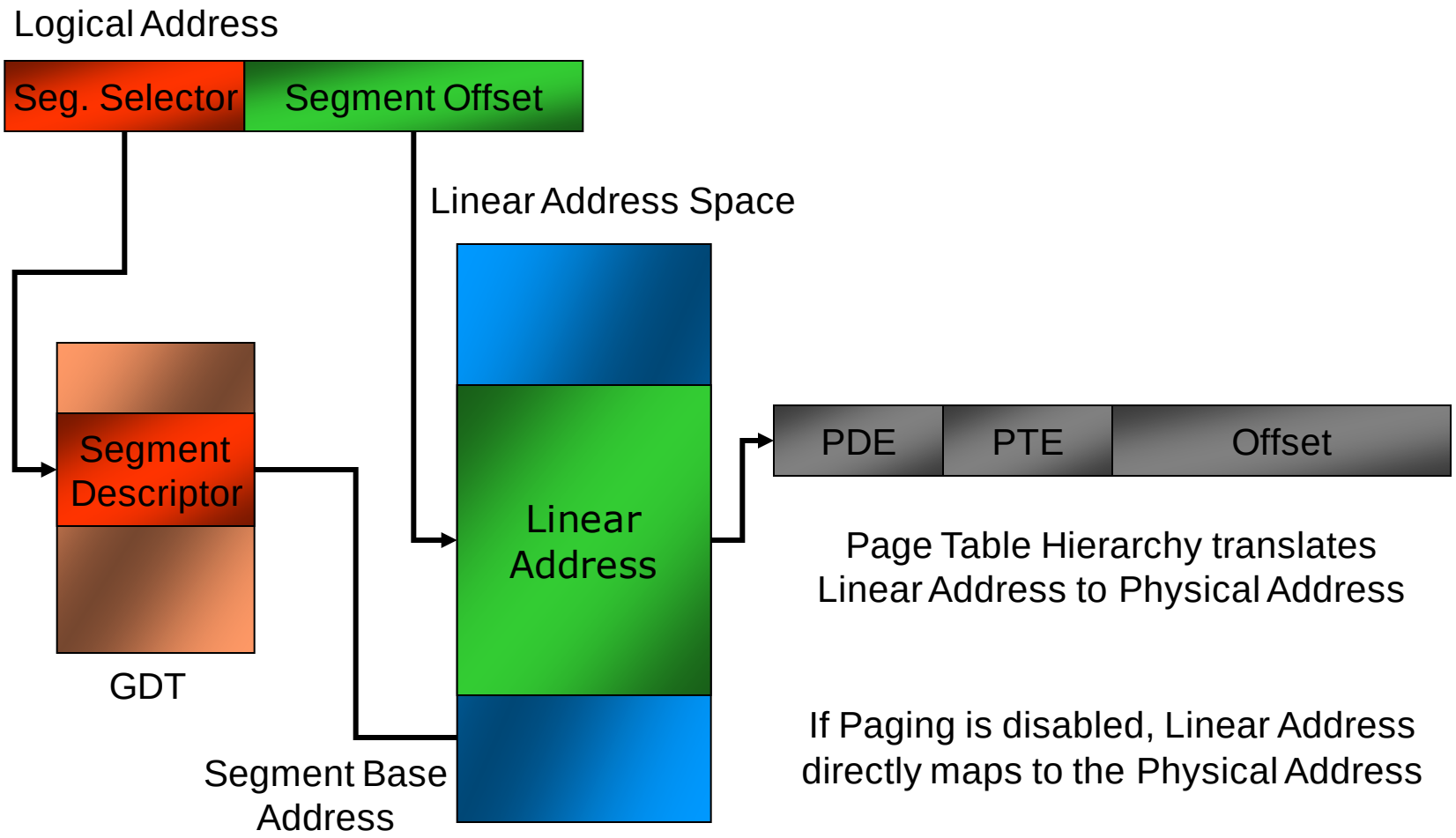
# Protection Facilities

- 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

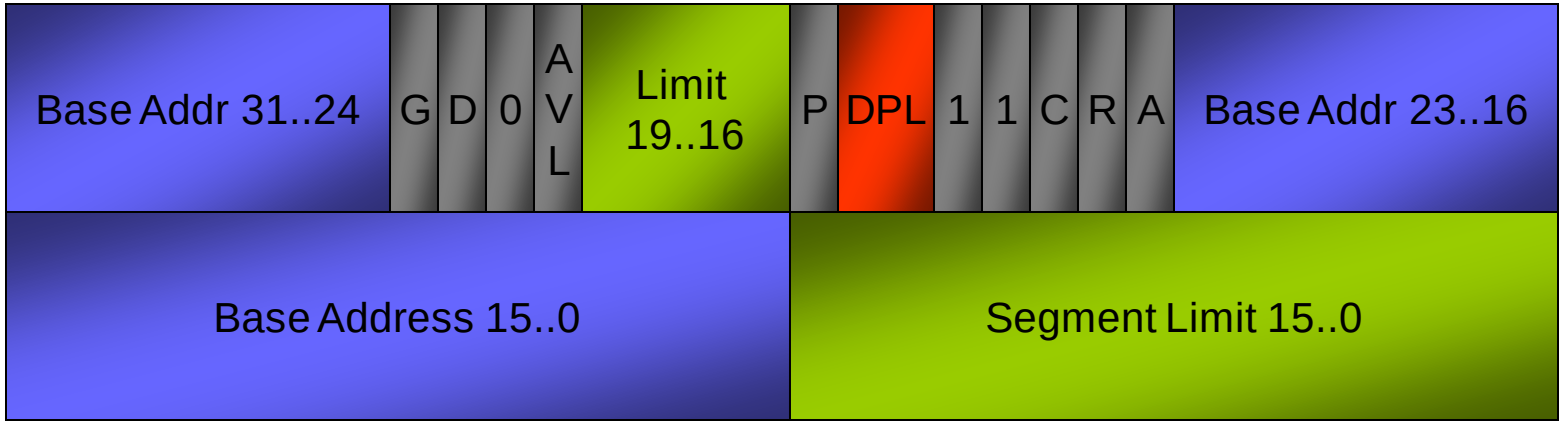
- Processor provides 6 segment registers for holding segment selectors
  - CS (code segment)
  - DS (data segment)
  - SS (stack segment)
  - ES, FS, GS (extra data segments)
- Selector forms visible part; segment base address, limit and access rights form shadow part of the segment register (invisible)

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

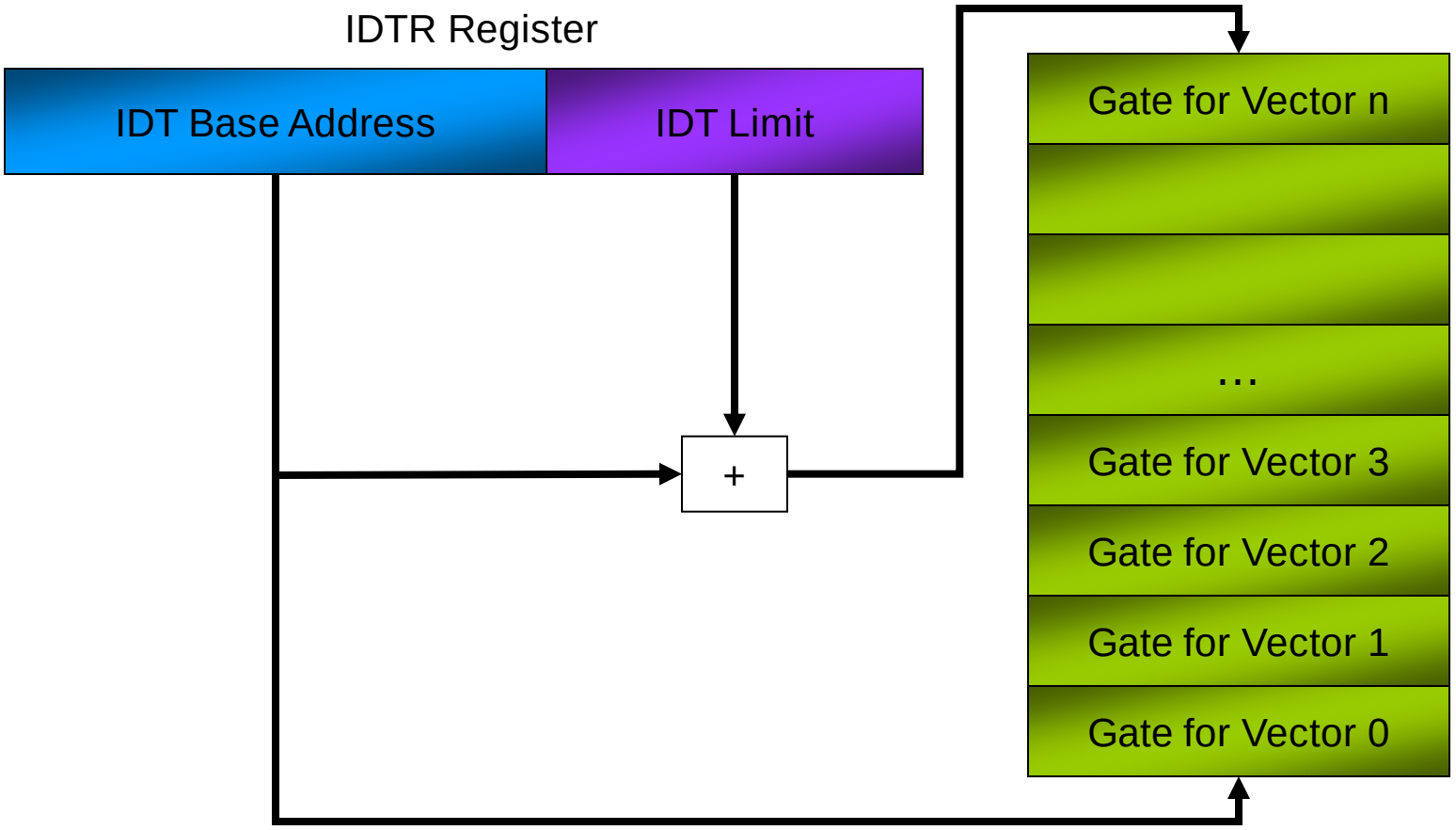
# Interrupt / Exception Vectors

- 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

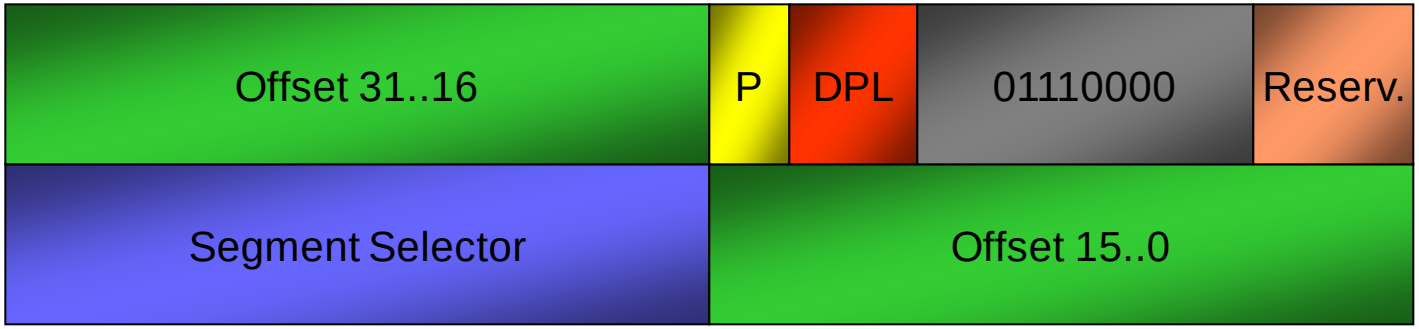
# Restricted Procedure Entry Points: Gates

- 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

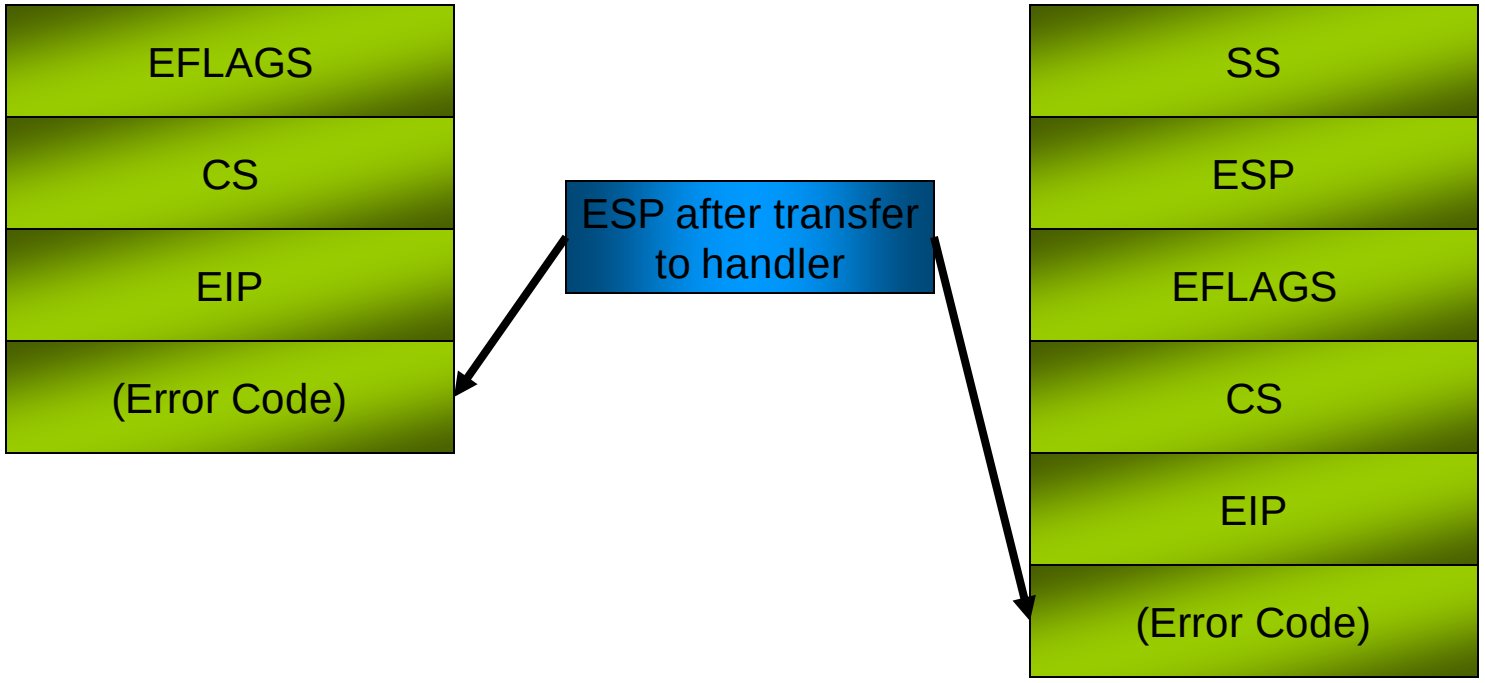
# x86 Exceptions

- 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 Privilege-Level Change

Privilege-Level Change

Udo Steinberg

TU Dresden
Operating
Systems Group

# Kernel Entry: Page Fault (from User Mode)

```
entry_vec0e_page_fault:
    pushl %eax
    pushl %ecx
    pushl %edx
    RESET_KERNEL_SEGMENTS
    movl  12(%esp),%edx     /* ARG2: error code */
    movl  %cr2,%eax         /* ARG1: fault address */
    movl  20(%esp),%ecx     /* save CS */
    andb  $3,%cl
    jz    1f
    ESP_TO_TCB_AT %ecx
    RESET_THREAD_CANCEL_AT %ecx
1:  movl  %ebp,12(%esp)     /* save frame pointer */
    leal  12(%esp),%ebp     /* new frame pointer */
    pushl %eax              /* save pf address */
    pushl %edx              /* save error code */
    leal  24(%esp),%ecx
    pushl %ecx              /* ARG5: ptr to frame */
    pushl 36(%esp)          /* ARG4: eflags */
    movl  32(%esp),%ecx     /* ARG3: eip */
    call  thread_page_fault
```

Udo Steinberg

TU Dresden
Operating
Systems Group

# Kernel Entry: Page Fault (from User Mode)


```
entry_vec0e_page_fault:
    pushl %eax
    pushl %ecx
    pushl %edx
    RESET_KERNEL_SEGMENTS
    movl  12(%esp),%edx     /* ARG2: error code */
    movl  %cr2,%eax         /* ARG1: fault address */
    movl  20(%esp),%ecx     /* save CS */
    andb  $3,%cl
    jz    1f
    ESP_TO_TCB_AT %ecx
    RESET_THREAD_CANCEL_AT %ecx
1:  movl  %ebp,12(%esp)     /* save frame pointer */
    leal  12(%esp),%ebp     /* new frame pointer */
    pushl %eax              /* save pf address */
    pushl %edx              /* save error code */
    leal  24(%esp),%ecx
    pushl %ecx              /* ARG5: ptr to frame */
    pushl 36(%esp)          /* ARG4: eflags */
    movl  32(%esp),%ecx     /* ARG3: eip */
    call  thread_page_fault
```



Udo Steinberg

TU Dresden
Operating
Systems Group

# Kernel Entry: Page Fault (from User Mode)


```
entry_vec0e_page_fault:
    pushl %eax
    pushl %ecx
    pushl %edx
    RESET_KERNEL_SEGMENTS
    movl  12(%esp),%edx     /* ARG2: error code */
    movl  %cr2,%eax         /* ARG1: fault address */
    movl  20(%esp),%ecx     /* save CS */
    andb  $3,%cl
    jz    1f
    ESP_TO_TCB_AT %ecx
    RESET_THREAD_CANCEL_AT %ecx
1:  movl  %ebp,12(%esp)     /* save frame pointer */
    leal  12(%esp),%ebp     /* new frame pointer */
    pushl %eax              /* save pf address */
    pushl %edx              /* save error code */
    leal  24(%esp),%ecx
    pushl %ecx              /* ARG5: ptr to frame */
    pushl 36(%esp)          /* ARG4: eflags */
    movl  32(%esp),%ecx     /* ARG3: eip */
    call  thread_page_fault
```



Udo Steinberg

TU Dresden
Operating
Systems Group

# Kernel Entry: Page Fault (from User Mode)

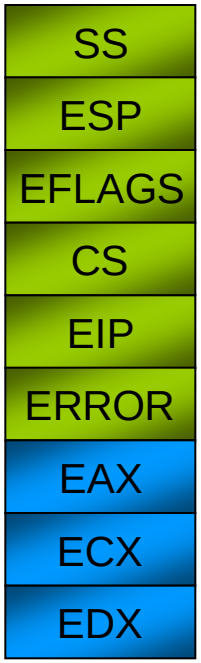

```
entry_vec0e_page_fault:
    pushl %eax
    pushl %ecx
    pushl %edx
    RESET_KERNEL_SEGMENTS
    movl  12(%esp),%edx     /* ARG2: error code */
    movl  %cr2,%eax         /* ARG1: fault address */
    movl  20(%esp),%ecx     /* save CS */
    andb  $3,%cl
    jz    1f
    ESP_TO_TCB_AT %ecx
    RESET_THREAD_CANCEL_AT %ecx
1:  movl  %ebp,12(%esp)     /* save frame pointer */
    leal  12(%esp),%ebp     /* new frame pointer */
    pushl %eax              /* save pf address */
    pushl %edx              /* save error code */
    leal  24(%esp),%ecx
    pushl %ecx              /* ARG5: ptr to frame */
    pushl 36(%esp)          /* ARG4: eflags */
    movl  32(%esp),%ecx     /* ARG3: eip */
    call  thread_page_fault
```



Udo Steinberg

TU Dresden
Operating
Systems Group

# Kernel Entry: Page Fault (from User Mode)



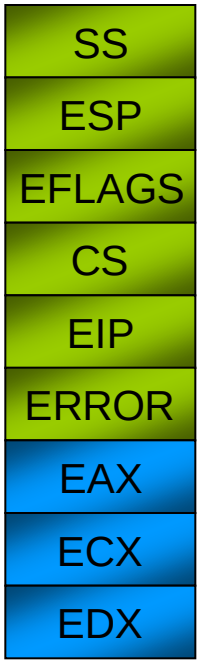

```
entry_vec0e_page_fault:
    pushl %eax
    pushl %ecx
    pushl %edx
    RESET_KERNEL_SEGMENTS
    movl  12(%esp),%edx     /* ARG2: error code */
    movl  %cr2,%eax         /* ARG1: fault address */
    movl  20(%esp),%ecx     /* save CS */
    andb  $3,%cl
    jz    1f
    ESP_TO_TCB_AT %ecx
    RESET_THREAD_CANCEL_AT %ecx
1:  movl  %ebp,12(%esp)     /* save frame pointer */
    leal  12(%esp),%ebp     /* new frame pointer */
    pushl %eax              /* save pf address */
    pushl %edx              /* save error code */
    leal  24(%esp),%ecx
    pushl %ecx              /* ARG5: ptr to frame */
    pushl 36(%esp)          /* ARG4: eflags */
    movl  32(%esp),%ecx     /* ARG3: eip */
    call  thread_page_fault
```



Udo Steinberg

TU Dresden
Operating
Systems Group

# Kernel Entry: Page Fault (from User Mode)

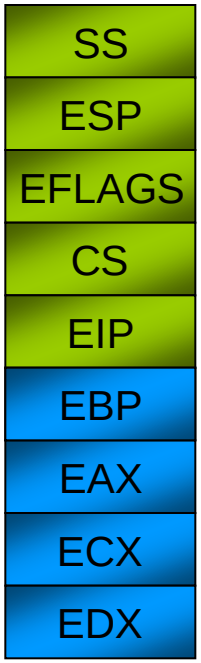



```
entry_vec0e_page_fault:
    pushl %eax
    pushl %ecx
    pushl %edx
    RESET_KERNEL_SEGMENTS
    movl  12(%esp),%edx     /* ARG2: error code */
    movl  %cr2,%eax         /* ARG1: fault address */
    movl  20(%esp),%ecx     /* save CS */
    andb  $3,%cl
    jz    1f
    ESP_TO_TCB_AT %ecx
    RESET_THREAD_CANCEL_AT %ecx
1:  movl  %ebp,12(%esp)     /* save frame pointer */
    leal  12(%esp),%ebp     /* new frame pointer */
    pushl %eax              /* save pf address */
    pushl %edx              /* save error code */
    leal  24(%esp),%ecx
    pushl %ecx              /* ARG5: ptr to frame */
    pushl 36(%esp)          /* ARG4: eflags */
    movl  32(%esp),%ecx     /* ARG3: eip */
    call  thread_page_fault
```

Udo Steinberg

TU Dresden
Operating
Systems Group

# Kernel Entry: Page Fault (from User Mode)



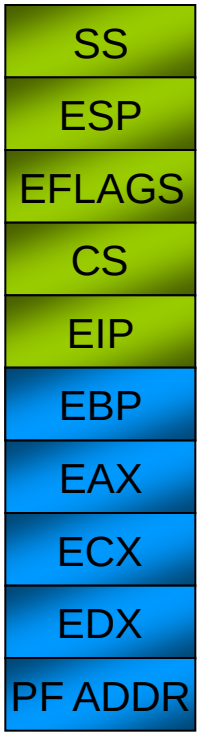

```
entry_vec0e_page_fault:
    pushl %eax
    pushl %ecx
    pushl %edx
    RESET_KERNEL_SEGMENTS
    movl  12(%esp),%edx     /* ARG2: error code */
    movl  %cr2,%eax         /* ARG1: fault address */
    movl  20(%esp),%ecx     /* save CS */
    andb  $3,%cl
    jz    1f
    ESP_TO_TCB_AT %ecx
    RESET_THREAD_CANCEL_AT %ecx
1:  movl  %ebp,12(%esp)     /* save frame pointer */
    leal  12(%esp),%ebp     /* new frame pointer */
    pushl %eax              /* save pf address */
    pushl %edx              /* save error code */
    leal  24(%esp),%ecx
    pushl %ecx              /* ARG5: ptr to frame */
    pushl 36(%esp)          /* ARG4: eflags */
    movl  32(%esp),%ecx     /* ARG3: eip */
    call  thread_page_fault
```



Udo Steinberg

TU Dresden
Operating
Systems Group

# Kernel Entry: Page Fault (from User Mode)

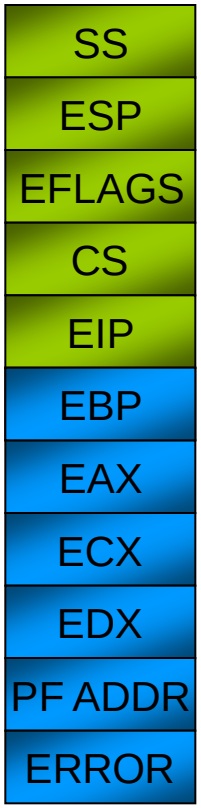



```
entry_vec0e_page_fault:
    pushl %eax
    pushl %ecx
    pushl %edx
    RESET_KERNEL_SEGMENTS
    movl  12(%esp),%edx     /* ARG2: error code */
    movl  %cr2,%eax         /* ARG1: fault address */
    movl  20(%esp),%ecx     /* save CS */
    andb  $3,%cl
    jz    1f
    ESP_TO_TCB_AT %ecx
    RESET_THREAD_CANCEL_AT %ecx
1:  movl  %ebp,12(%esp)     /* save frame pointer */
    leal  12(%esp),%ebp     /* new frame pointer */
    pushl %eax              /* save pf address */
    pushl %edx              /* save error code */
    leal  24(%esp),%ecx
    pushl %ecx              /* ARG5: ptr to frame */
    pushl 36(%esp)          /* ARG4: eflags */
    movl  32(%esp),%ecx     /* ARG3: eip */
    call  thread_page_fault
```

Udo Steinberg

TU Dresden
Operating
Systems Group

# Kernel Entry: Page Fault (from User Mode)

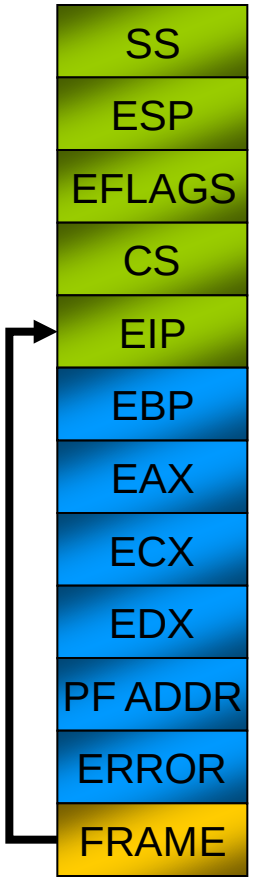



```
entry_vec0e_page_fault:
    pushl %eax
    pushl %ecx
    pushl %edx
    RESET_KERNEL_SEGMENTS
    movl  12(%esp),%edx     /* ARG2: error code */
    movl  %cr2,%eax         /* ARG1: fault address */
    movl  20(%esp),%ecx     /* save CS */
    andb  $3,%cl
    jz    1f
    ESP_TO_TCB_AT %ecx
    RESET_THREAD_CANCEL_AT %ecx
1:  movl  %ebp,12(%esp)     /* save frame pointer */
    leal  12(%esp),%ebp     /* new frame pointer */
    pushl %eax              /* save pf address */
    pushl %edx              /* save error code */
    leal  24(%esp),%ecx
    pushl %ecx              /* ARG5: ptr to frame */
    pushl 36(%esp)          /* ARG4: eflags */
    movl  32(%esp),%ecx     /* ARG3: eip */
    call  thread_page_fault
```

Udo Steinberg

TU Dresden
Operating
Systems Group

# Kernel Entry: Page Fault (from User Mode)

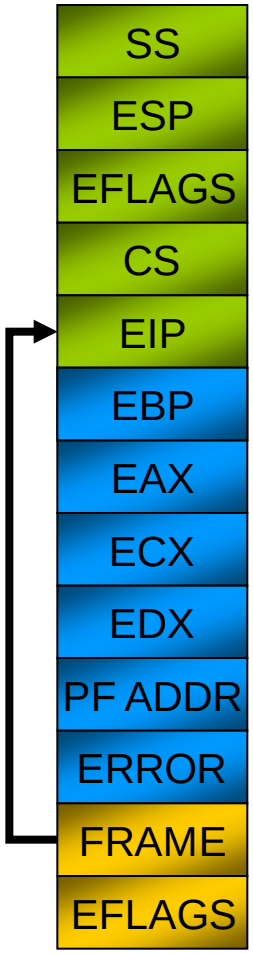



```
entry_vec0e_page_fault:
    pushl %eax
    pushl %ecx
    pushl %edx
    RESET_KERNEL_SEGMENTS
    movl  12(%esp),%edx     /* ARG2: error code */
    movl  %cr2,%eax         /* ARG1: fault address */
    movl  20(%esp),%ecx     /* save CS */
    andb  $3,%cl
    jz    1f
    ESP_TO_TCB_AT %ecx
    RESET_THREAD_CANCEL_AT %ecx
1:  movl  %ebp,12(%esp)     /* save frame pointer */
    leal  12(%esp),%ebp     /* new frame pointer */
    pushl %eax              /* save pf address */
    pushl %edx              /* save error code */
    leal  24(%esp),%ecx
    pushl %ecx              /* ARG5: ptr to frame */
    pushl 36(%esp)          /* ARG4: eflags */
    movl  32(%esp),%ecx     /* ARG3: eip */
    call  thread_page_fault
```

Udo Steinberg

TU Dresden
Operating
Systems Group

# Kernel Entry: Page Fault (from User Mode)

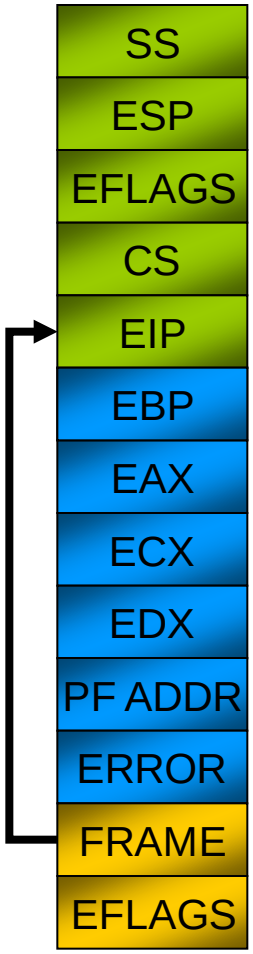



```
entry_vec0e_page_fault:
    pushl %eax
    pushl %ecx
    pushl %edx
    RESET_KERNEL_SEGMENTS
    movl  12(%esp),%edx     /* ARG2: error code */
    movl  %cr2,%eax         /* ARG1: fault address */
    movl  20(%esp),%ecx     /* save CS */
    andb  $3,%cl
    jz    1f
    ESP_TO_TCB_AT %ecx
    RESET_THREAD_CANCEL_AT %ecx
1:  movl  %ebp,12(%esp)     /* save frame pointer */
    leal  12(%esp),%ebp     /* new frame pointer */
    pushl %eax              /* save pf address */
    pushl %edx              /* save error code */
    leal  24(%esp),%ecx
    pushl %ecx              /* ARG5: ptr to frame */
    pushl 36(%esp)          /* ARG4: eflags */
    movl  32(%esp),%ecx     /* ARG3: eip */
    call  thread_page_fault
```

Udo Steinberg

TU Dresden
Operating
Systems Group

# Kernel Entry: Page Fault (from User Mode)




```
entry_vec0e_page_fault:
    pushl %eax
    pushl %ecx
    pushl %edx
    RESET_KERNEL_SEGMENTS
    movl  12(%esp),%edx     /* ARG2: error code */
    movl  %cr2,%eax         /* ARG1: fault address */
    movl  20(%esp),%ecx     /* save CS */
    andb  $3,%cl
    jz    1f
    ESP_TO_TCB_AT %ecx
    RESET_THREAD_CANCEL_AT %ecx
1:  movl  %ebp,12(%esp)     /* save frame pointer */
    leal  12(%esp),%ebp     /* new frame pointer */
    pushl %eax              /* save pf address */
    pushl %edx              /* save error code */
    leal  24(%esp),%ecx
    pushl %ecx              /* ARG5: ptr to frame */
    pushl 36(%esp)          /* ARG4: eflags */
    movl  32(%esp),%ecx     /* ARG3: eip */
    call  thread_page_fault
```

Udo Steinberg

TU Dresden
Operating
Systems Group

# Page-Fault Handler

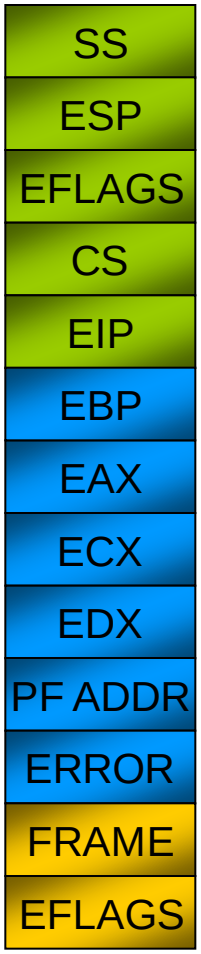
```
extern "C" FIASCO_FASTCALL
int
thread_page_fault (    Address pfa,
                       Mword error_code,
                       Address ip,
                       Mword flags,
                       Return_frame *regs){…}
```

- FIASCO_FASTCALL selects „regparm“ calling convention for passing function parameters
  - parameter1 (EAX)
  - parameter2 (EDX)
  - parameter3 (ECX)
  - all other parameters (stack)

Udo Steinberg

TU Dresden
Operating
Systems Group

# Kernel Exit: After Page Fault (to User Mode)



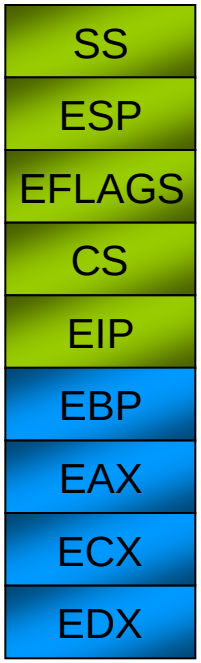

```
before_iret_page_fault:
   orl     %eax,%eax
   jz      bad_page_fault
   lea     16(%esp),%esp
   RESET_USER_SEGMENTS 20(%esp),no_cli
   popl    %edx
   popl    %ecx
   popl    %eax
   popl    %ebp
   iret
```



Udo Steinberg

TU Dresden
Operating
Systems Group

# Kernel Exit: After Page Fault (to User Mode)




```
before_iret_page_fault:
   orl     %eax,%eax
   jz      bad_page_fault
   lea     16(%esp),%esp
   RESET_USER_SEGMENTS 20(%esp),no_cli
   popl    %edx
   popl    %ecx
   popl    %eax
   popl    %ebp
   iret
```



Udo Steinberg

TU Dresden
Operating
Systems Group

# Kernel Exit: After Page Fault (to User Mode)


```
before_iret_page_fault:
   orl     %eax,%eax
   jz      bad_page_fault
   lea     16(%esp),%esp
   RESET_USER_SEGMENTS 20(%esp),no_cli
   popl    %edx
   popl    %ecx
   popl    %eax
   popl    %ebp
   iret
```



Udo Steinberg

TU Dresden
Operating
Systems Group

# Kernel Exit: After Page Fault (to User Mode)


```
before_iret_page_fault:
   orl     %eax,%eax
   jz      bad_page_fault
   lea     16(%esp),%esp
   RESET_USER_SEGMENTS 20(%esp),no_cli
   popl    %edx
   popl    %ecx
   popl    %eax
   popl    %ebp
   iret
```



Udo Steinberg

TU Dresden
Operating
Systems Group

# Kernel Exit: After Page Fault (to User Mode)


```
before_iret_page_fault:
   orl     %eax,%eax
   jz      bad_page_fault
   lea     16(%esp),%esp
   RESET_USER_SEGMENTS 20(%esp),no_cli
   popl    %edx
   popl    %ecx
   popl    %eax
   popl    %ebp
   iret
```



Udo Steinberg

TU Dresden
Operating
Systems Group

# Kernel Exit: After Page Fault (to User Mode)


```
before_iret_page_fault:
   orl     %eax,%eax
   jz      bad_page_fault
   lea     16(%esp),%esp
   RESET_USER_SEGMENTS 20(%esp),no_cli
   popl    %edx
   popl    %ecx
   popl    %eax
   popl    %ebp
   iret
```



Udo Steinberg

TU Dresden
Operating
Systems Group

# Kernel Exit: After Page Fault (to User Mode)

```
before_iret_page_fault:
   orl     %eax,%eax
   jz      bad_page_fault
   lea     16(%esp),%esp
   RESET_USER_SEGMENTS 20(%esp),no_cli
   popl    %edx
   popl    %ecx
   popl    %eax
   popl    %ebp
   iret
```



Udo Steinberg

TU Dresden
Operating
Systems Group

# Kernel Exit: IRET

- Evaluate CS's RPL on stack
  - RPL > CPL - return to other privilege level
  - otherwise - return to same privilege level
- Reload EIP, CS, EFLAGS from stack
- On other-privilege-level return
  - reload ESP and SS from stack
  - adjust CPL to new privilege level

Udo Steinberg

TU Dresden
Operating
Systems Group

# System Call

- System calls are technically exceptions
- L4 system calls use vectors 0x30 through 0x36
- Invocation
  - User code calls L4 binding, e.g. l4_myself
  - Binding transfers parameters into registers
  - Binding invokes kernel via interrupt gate or SYSENTER instruction
  - Binding transfers return registers into return parameters of L4 function call

Udo Steinberg

TU Dresden
Operating
Systems Group

# System Call: User Mode Binding

```
L4_INLINE
l4_threadid_t
l4_myself (void)
{
  l4_threadid_t temp_id;
  asm ("push %%ebp  \n\t"
       "int $0x31   \n\t"          // id_nearest
       "popl %%ebp  \n\t"
       : "=S" (temp_id.lh_low),
         "=D" (temp_id.lh_high)
       : "S" (0)
       : "ebx, "eax", "ecx", "edx");
  return temp_id;
}
```

Udo Steinberg

TU Dresden
Operating
Systems Group

# System Call Entry

```
#define SYSTEM_CALL(int,name)                                          \
GATE_ENTRY(int,entry_##name,ACC_PL_U | ACC_INTR_GATE)                 ;\
   entry_##name:                                                      ;\
        pushl   %eax                                                  ;\
        movl    $(syscall_table+4*(int-0x30)), %eax                   ;\
        jmp     all_syscalls

all_syscalls:
        SAVE_STATE
        ESP_TO_TCB_AT %ebx
        RESET_KERNEL_SEGMENTS
        RESET_THREAD_CANCEL_AT %ebx
        call    *(%eax)

ret_from_syscall:
        RESET_USER_SEGMENTS $3,no_cli
        RESTORE_STATE
        popl    %eax
        iret
```

Udo Steinberg

TU Dresden
Operating
Systems Group

# System-Call Handling

```
void *(syscall_table[])() =
{
   sys_ipc_wrapper,
   sys_id_nearest_wrapper,
   sys_fpage_unmap_wrapper,
   sys_thread_switch_wrapper,
   sys_thread_schedule_wrapper,
   sys_thread_ex_regs_wrapper,
   sys_task_new_wrapper
};

extern "C" void sys_id_nearest_wrapper()
{ Proc::sti(); current_thread()->sys_id_nearest(); }

/** L4 system call id_nearest */
IMPLEMENT inline NOEXPORT ALWAYS_INLINE
void
Thread::sys_id_nearest()
{ … }
```

Udo Steinberg

TU Dresden
Operating
Systems Group

# 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