barkhausen institut

Microkernel Construction

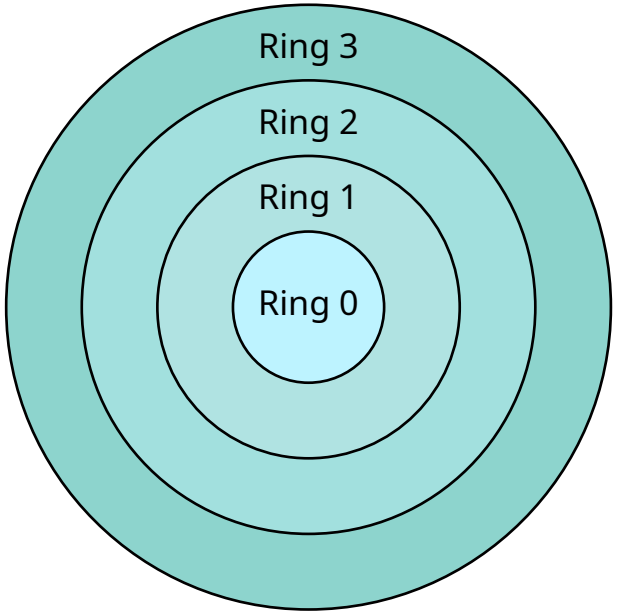
Kernel Entry/Exit

Nils Asmussen, Viktor Reusch

04/30/2026

- **x86-64 Details**
  - **Protection Facilities**
  - Interrupts and Exceptions
- Entering NOVA
- Leaving NOVA
- Syscall Instructions

Privilege Levels



- Ring 0
  - Operating-System Kernel
  - Privileged
- Ring 1 & 2
  - Operating-System Services
  - Unprivileged
- Ring 3
  - User-level Applications
  - Unprivileged

Protection Facilities

- Segmentation
  - Mechanism for dividing linear address space into segments
  - Different segment types: code, data, stack, …
  - Segment has base address and limit
  - Largely disabled/flat in `x86-64` (exceptions: `GS` and `FS`)

Protection Facilities

- Segmentation
  - Mechanism for dividing linear address space into segments
  - Different segment types: code, data, stack, …
  - Segment has base address and limit
  - Largely disabled/flat in `x86-64` (exceptions: `GS` and `FS`)
- Paging
  - Mechanism for translating virtual to physical addresses
  - Splits virt. & phys. address space into same-sized pages/frames
  - Per-page access rights (present, writable, user/kernel)
  - Mandatory on `x86-64`

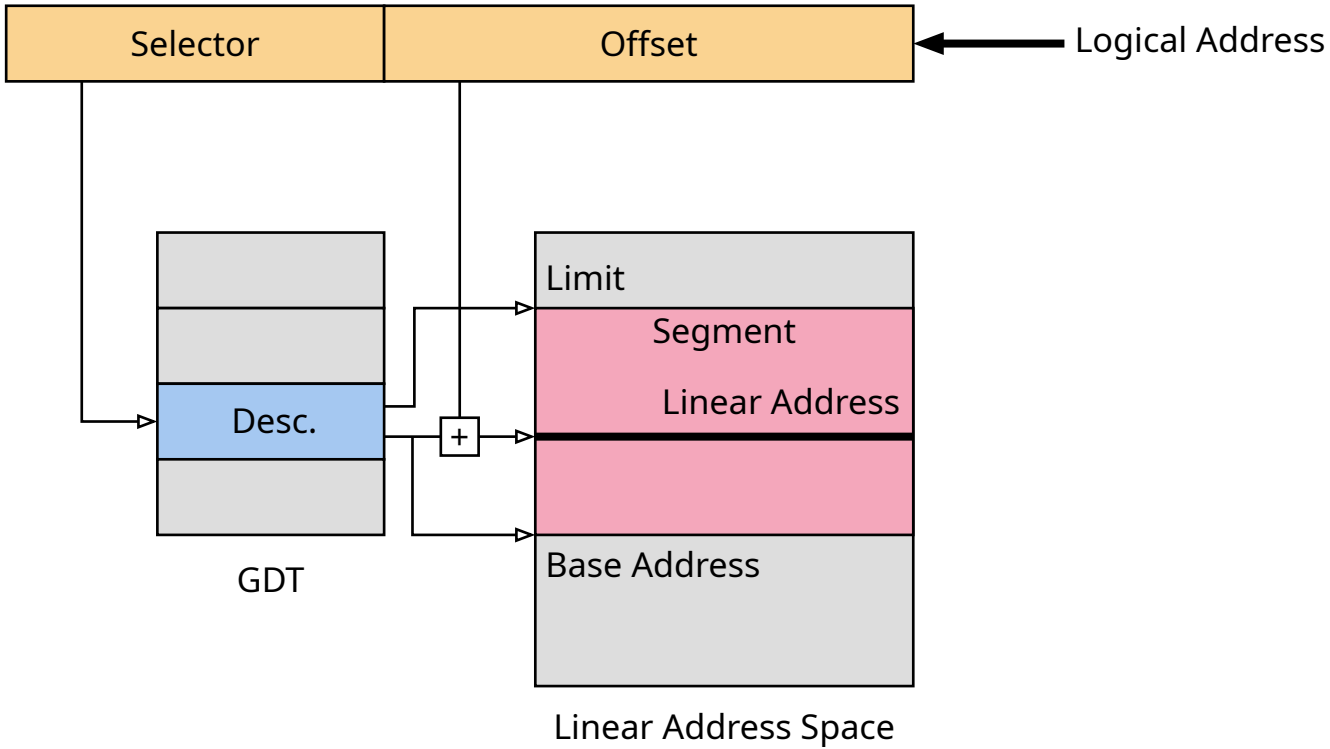
Logical Address Translation

- Logical address consists of
  - Segment *selector*
  - Segment *offset*
- Selector references segment with:
  - Base address and limit
  - Access rights
- CPU adds offset to base address to form linear address

Logical Address Translation

- Logical address consists of
  - Segment *selector*
  - Segment *offset*
- Selector references segment with:
  - Base address and limit
  - Access rights
- CPU adds offset to base address to form linear address
- Two segment tables:
  1. global descriptor table (`GDT`)
  2. local descriptor table (`LDT`)
- *Table indicator* in segment selector to distinguish `GDT`/`LDT`

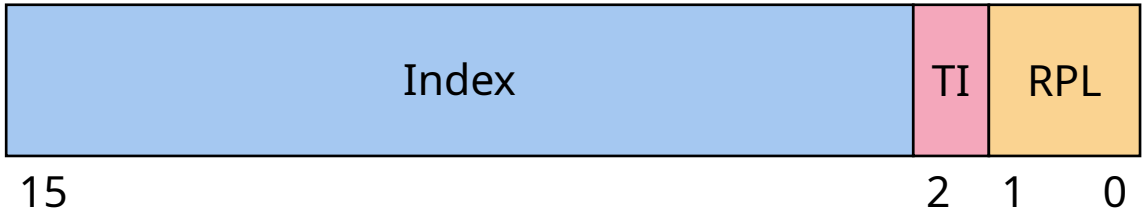
Logical Address Translation



Segment Registers

- CPU provides 6 segment registers for holding selectors
  - `CS` (code segment)
  - `DS` (data segment)
  - `SS` (stack segment)
  - `ES`, `FS`, `GS` (extra data segments)
- Selector forms visible part
- Base address, limit and access rights form shadow part

Segment Selector

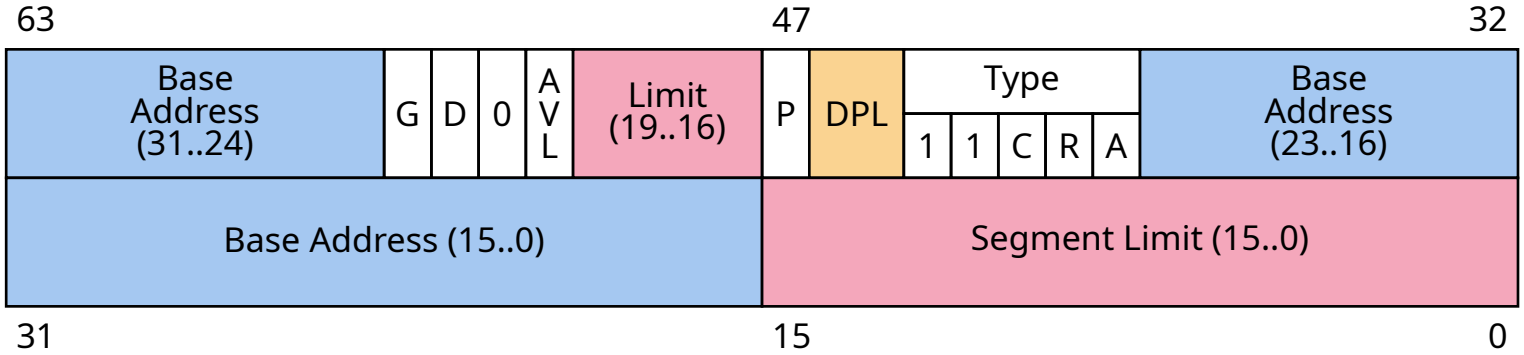


Index Index into GDT/LDT

TI Table Indicator

RPL Requested Privilege Level

Segment Descriptor



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

Flat Memory Model

- Hide segmentation mechanism by
  - Setting segment base address to 0
  - Setting segment limit to **max-address**
- Mandatory for `x86_64` (except `FS`/`GS`)
- We need at least 2 segments
  - Code segment
  - Data/Stack segment
- If kernel/user use different segment settings, we need 2 additional segments

Protection Checks

- CPU ensures that certain protection conditions hold:
  - Segment limit check
  - Segment type check
  - Privilege level check
  - Restricted procedure entry points
  - Restricted instruction set
  - ...
- CPU raises exception if protection check fails
- Protection checks performed in parallel with address translation

Privileged Instructions

Group	Instructions
Descriptor Table Management	`LGDT`, `LLDT`, `LTR`, `LIDT`
Control Register Access	`MOV` to/from `CR`, `LMSW`, `CLTS`
Debug Register Access	`MOV` to/from `DR`
Cache and TLB Management	`INVD`, `WBINVD`, `INVLPG`
Processor State Management	`HLT`, `CLI` / `STI`
Model-Specific Register Access	`RDMSR` / `WRMSR`

- **x86-64 Details**
  - Protection Facilities
  - **Interrupts and Exceptions**
- Entering NOVA
- Leaving NOVA
- Syscall Instructions

Interrupts & Exceptions

- Forced control transfer to interrupt/exception handler

Interrupts & Exceptions

- Forced control transfer to interrupt/exception handler

Interrupts

- Asynchronous to current control flow
- Response to hardware event (timer, storage device, ...)

Interrupts & Exceptions

- Forced control transfer to interrupt/exception handler

Interrupts

- Asynchronous to current control flow
- Response to hardware event (timer, storage device, ...)

Exceptions

- Synchronous to current control flow
- Caused by a specific instruction (page fault, division by zero, ...)

Interrupt & Exception Vectors

Vector

- Uniquely identifies source of the interrupt or exception
- Vectors 0 .. 31 are used for exceptions
- Vectors 32 .. 255 are designated user vectors (e.g., interrupts)

Interrupt & Exception Vectors

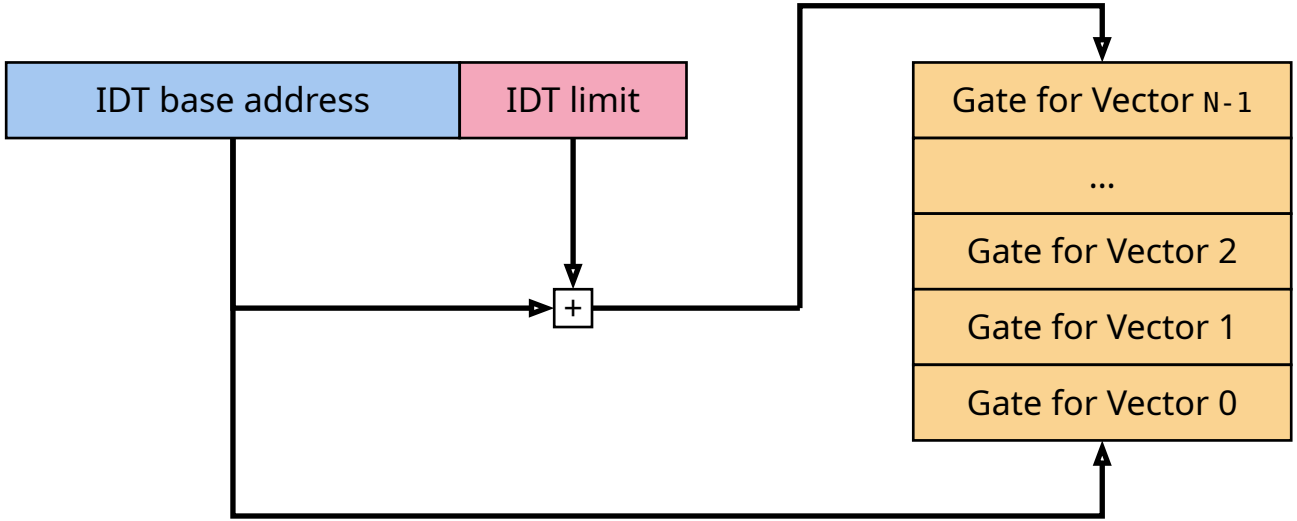
Vector

- Uniquely identifies source of the interrupt or exception
- Vectors 0 .. 31 are used for exceptions
- Vectors 32 .. 255 are designated user vectors (e.g., interrupts)

Interrupt Descriptor Table (IDT)

- Table of handler functions for all interrupts and exceptions
- Setup by OS
- Hardware is informed about it via `LIDT` instruction
- Vector = offset into IDT

Interrupt Descriptor Table (IDT)



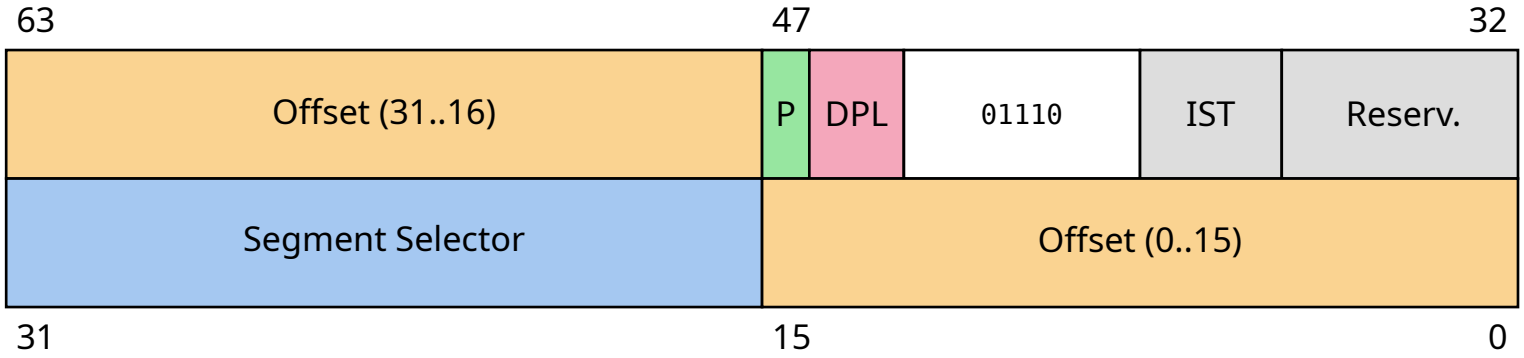
Restricted Procedure Entry Points: Gates

- Descriptors to call procedures at different privilege levels
  - Subject to CPL vs. DPL checking
  - `CS` and `RIP` loaded from descriptor
  - Interrupt Gate disables interrupts, Trap Gate doesn’t

Restricted Procedure Entry Points: Gates

- Descriptors to call procedures at different privilege levels
  - Subject to CPL vs. DPL checking
  - `CS` and `RIP` loaded from descriptor
  - Interrupt Gate disables interrupts, Trap Gate doesn’t
- When switching privilege levels, new `RSP` loaded from TSS
  - TSS contains stack pointers for privilege levels 0, 1 and 2
  - Kernel updates `RSP0` in TSS on every thread switch
  - Interrupt and exception handlers run current thread’s context

Interrupt Gate



Selector Segment Selector for destination code segment

Offset Offset to procedure entry point in segment

P Segment present flag

DPL Descriptor Privilege Level (ignored for HW interrupts)

01110 Type of gate (14 = interrupt gate)

IST Interrupt Stack Table, for specific interrupts/exceptions

Exception Types

1. Faults
   - Can be corrected and allow the program to be resumed
   - Repeats same instruction on resume (`RIP` points to faulting instruction)

Exception Types

1. Faults
   - Can be corrected and allow the program to be resumed
   - Repeats same instruction on resume (`RIP` points to faulting instruction)
2. Traps
   - Reported immediately after execution of trapping instruction
   - Allow the program to be resumed (`RIP` points to following instruction)

Exception Types

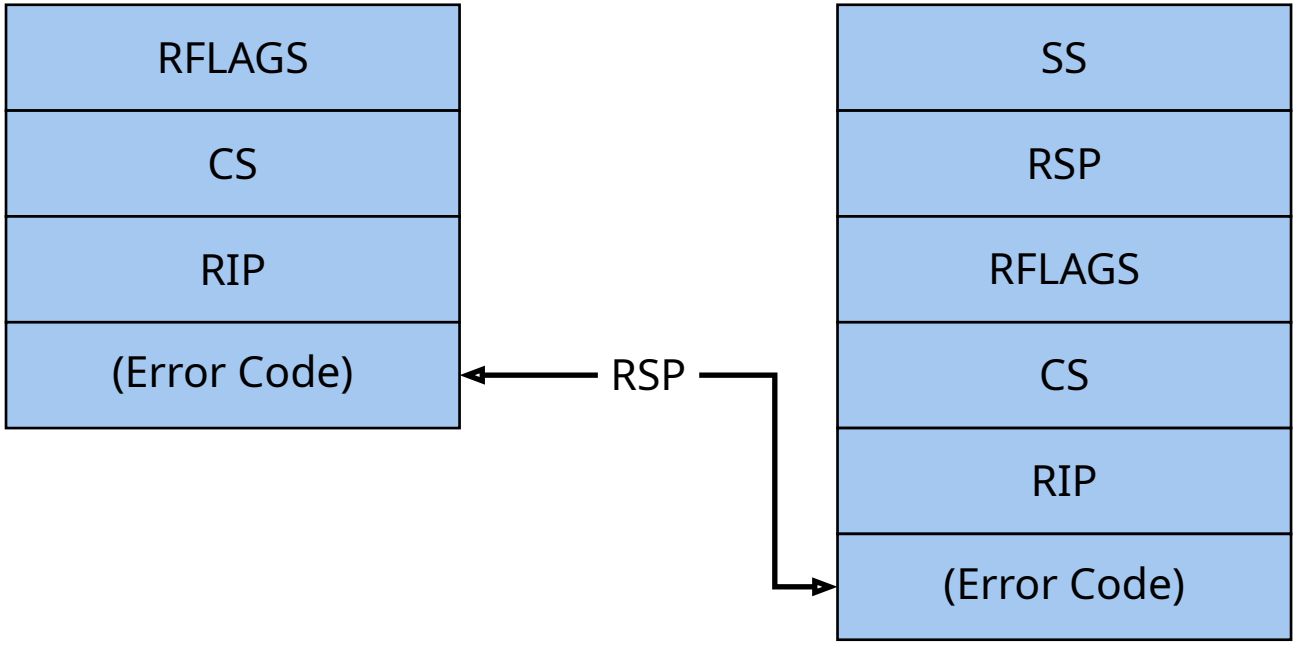
1. Faults
   - Can be corrected and allow the program to be resumed
   - Repeats same instruction on resume (`RIP` points to faulting instruction)
2. Traps
   - Reported immediately after execution of trapping instruction
   - Allow the program to be resumed (`RIP` points to following instruction)
3. Aborts
   - Are not always reported at the precise location of the exception
   - Do not allow to resume the program
   - Usually report severe hardware errors or inconsistent state

Exceptions

0 Divide Error (F)
1 Debug Exception (F/T)
2 Non-maskable Interrupt
3 Breakpoint (T)
4 Overflow (T)
5 Bound Range Exc. (F)
6 Invalid Opcode (F)
7 Device Not Available (F)
8 Double Fault (A)
9 *reserved*
10 Invalid TSS (F)
11 Segm. Not Present (F)
12 Stack Segment Fault (F)
13 General Prot. Fault (F)
14 Pagefault (F)
15 *reserved*
16 FPU Math Fault (F)
17 Alignment Check (F)
18 Machine Check (A)
19 SIMD FP Exception (F)

- x86-64 Details
- **Entering NOVA**
- Leaving NOVA
- Syscall Instructions

Kernel Stack after Kernel Entry



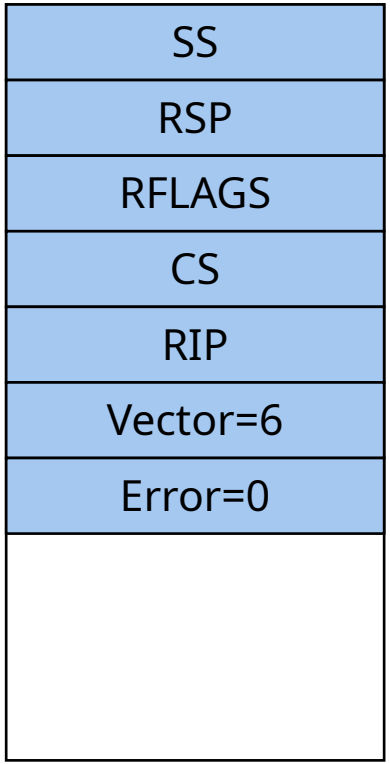
Kernel Entry From User Mode


```asm
1   entry_6:          push    $6
2                     jmp     entry_exc
3
4   entry_exc:        push    $0
5   entry_exc_err:    push    %rax ; %rcx, %rdx, %rbx, ...
6                     mov     %rsp, %rbx
7                     cmp     $KSTCK_ADDR, %rsp
8                     jae     entry_k_stack
9                     cld
10                    mov     $(KSTCK_ADDR + PAGE_SIZE), %rsp
11  entry_k_stack:    mov     %cr2, %rax
12                    mov     %rax, OFS_CR2(%rbx)
13                    mov     %rbx, %rdi
14                    call    exc_handler
15                    jmp     ret_from_interrupt
```

Kernel Entry From User Mode


```asm
1   entry_6:        push    $6
2                   jmp     entry_exc
3
4   entry_exc:      push    $0
5   entry_exc_err:  push    %rax ; %rcx, %rdx, %rbx, ...
6                   mov     %rsp, %rbx
7                   cmp     $KSTCK_ADDR, %rsp
8                   jae     entry_k_stack
9                   cld
10                  mov     $(KSTCK_ADDR + PAGE_SIZE), %rsp
11  entry_k_stack:  mov     %cr2, %rax
12                  mov     %rax, OFS_CR2(%rbx)
13                  mov     %rbx, %rdi
14                  call    exc_handler
15                  jmp     ret_from_interrupt
```

Kernel Entry From User Mode




```asm
1   entry_6:        push    $6
2                   jmp     entry_exc
3
4   entry_exc:      push    $0
5   entry_exc_err:  push    %rax ; %rcx, %rdx, %rbx, ...
6                   mov     %rsp, %rbx
7                   cmp     $KSTCK_ADDR, %rsp
8                   jae     entry_k_stack
9                   cld
10                  mov     $(KSTCK_ADDR + PAGE_SIZE), %rsp
11  entry_k_stack:  mov     %cr2, %rax
12                  mov     %rax, OFS_CR2(%rbx)
13                  mov     %rbx, %rdi
14                  call    exc_handler
15                  jmp     ret_from_interrupt
```

Kernel Entry From User Mode


```asm
1   entry_6:        push    $6
2                   jmp     entry_exc
3
4   entry_exc:      push    $0
5   entry_exc_err:  push    %rax ; %rcx, %rdx, %rbx, ...
6                   mov     %rsp, %rbx
7                   cmp     $KSTCK_ADDR, %rsp
8                   jae     entry_k_stack
9                   cld
10                  mov     $(KSTCK_ADDR + PAGE_SIZE), %rsp
11  entry_k_stack:  mov     %cr2, %rax
12                  mov     %rax, OFS_CR2(%rbx)
13                  mov     %rbx, %rdi
14                  call    exc_handler
15                  jmp     ret_from_interrupt
```

Kernel Entry From User Mode


```asm
1   entry_6:        push    $6
2                   jmp     entry_exc
3
4   entry_exc:      push    $0
5   entry_exc_err:  push    %rax ; %rcx, %rdx, %rbx, ...
6                   mov     %rsp, %rbx
7                   cmp     $KSTCK_ADDR, %rsp
8                   jae     entry_k_stack
9                   cld
10                  mov     $(KSTCK_ADDR + PAGE_SIZE), %rsp
11  entry_k_stack:  mov     %cr2, %rax
12                  mov     %rax, OFS_CR2(%rbx)
13                  mov     %rbx, %rdi
14                  call    exc_handler
15                  jmp     ret_from_interrupt
```

Kernel Entry From User Mode


```asm
1   entry_6:        push    $6
2                   jmp     entry_exc
3
4   entry_exc:      push    $0
5   entry_exc_err:  push    %rax ; %rcx, %rdx, %rbx, ...
6                   mov     %rsp, %rbx
7                   cmp     $KSTCK_ADDR, %rsp
8                   jae     entry_k_stack
9                   cld
10                  mov     $(KSTCK_ADDR + PAGE_SIZE), %rsp
11  entry_k_stack:  mov     %cr2, %rax
12                  mov     %rax, OFS_CR2(%rbx)
13                  mov     %rbx, %rdi
14                  call    exc_handler
15                  jmp     ret_from_interrupt
```

Kernel Entry From User Mode


```asm
1   entry_6:        push    $6
2                   jmp     entry_exc
3
4   entry_exc:      push    $0
5   entry_exc_err:  push    %rax ; %rcx, %rdx, %rbx, ...
6                   mov     %rsp, %rbx
7                   cmp     $KSTCK_ADDR, %rsp
8                   jae     entry_k_stack
9                   cld
10                  mov     $(KSTCK_ADDR + PAGE_SIZE), %rsp
11  entry_k_stack:  mov     %cr2, %rax
12                  mov     %rax, OFS_CR2(%rbx)
13                  mov     %rbx, %rdi
14                  call    exc_handler
15                  jmp     ret_from_interrupt
```

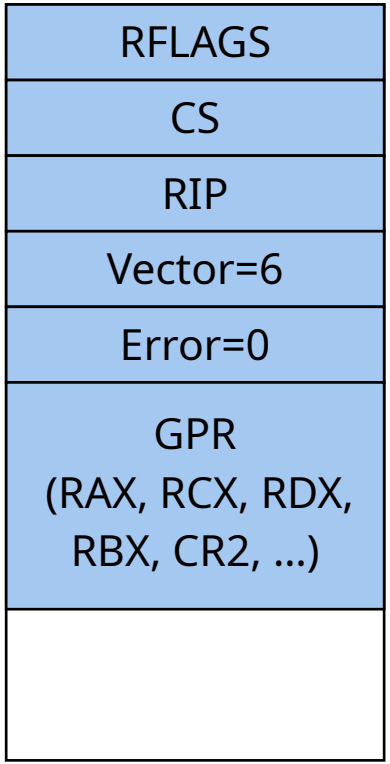
Exception Handler

C++

```
1  REGPARM(1) void exc_handler(Exc_regs *r) {
2      switch (r->vec) {
3          case Cpu::EXC_TS: handle_exc_ts(r); break;
4          case Cpu::EXC_PF: handle_exc_pf(r); break;
5          ...
6      }
7  }
8
9  void handle_exc_pf(Exc_regs *r) {
10     mword addr = r->cr2;
11     if (Pd::current->Space_mem::loc[Cpu::id].sync_from(
12         Pd::current->Space_mem::hpt, addr))
13         return;
14     ...
15 }
```

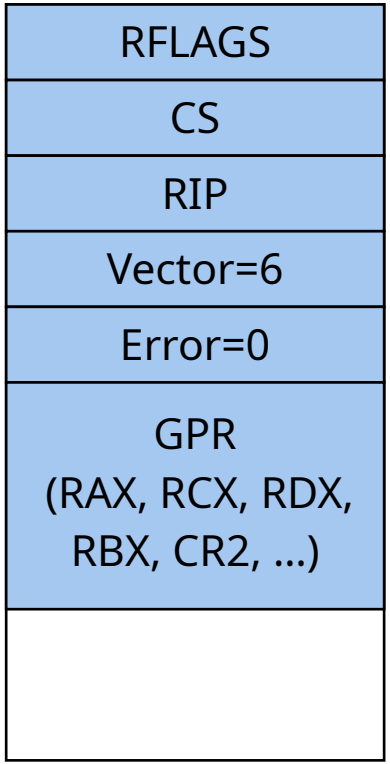
- x86-64 Details
- Entering NOVA
- **Leaving NOVA**
- Syscall Instructions

Kernel Exit to Kernel Mode



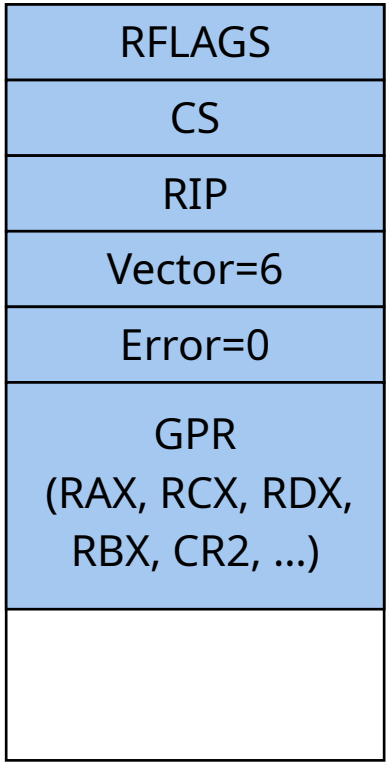

```asm
1                   ...
2                   cld
3                   mov     $(KSTCK_ADDR + PAGE_SIZE), %rsp
4   entry_k_stack:  mov     %cr2, %rax
5                   mov     %rax, OFS_CR2(%rbx)
6                   mov     %rbx, %rdi
7                   call    exc_handler
8                   jmp     ret_from_interrupt
9
10  ret_from_interrupt:
11                  testb   $3, OFS_CS(%rbx)
12                  jnz     ret_user_iret
13                  pop     %r15 ; ... %rdx, %rcx, %rax
14                  add     $16, %rsp
15                  iretq
```

Kernel Exit to Kernel Mode



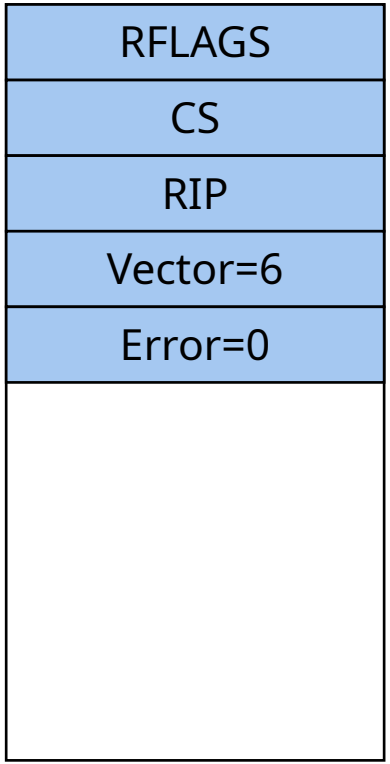

```asm
1                   ...
2                   cld
3                   mov     $(KSTCK_ADDR + PAGE_SIZE), %rsp
4   entry_k_stack:  mov     %cr2, %rax
5                   mov     %rax, OFS_CR2(%rbx)
6                   mov     %rbx, %rdi
7                   call    exc_handler
8                   jmp     ret_from_interrupt
9
10  ret_from_interrupt:
11                  testb   $3, OFS_CS(%rbx)
12                  jnz     ret_user_iret
13                  pop     %r15 ; ... %rdx, %rcx, %rax
14                  add     $16, %rsp
15                  iretq
```

Kernel Exit to Kernel Mode



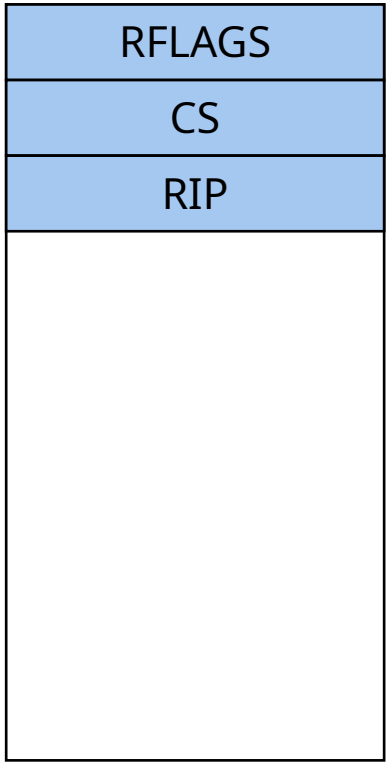

```asm
1                   ...
2                   cld
3                   mov     $(KSTCK_ADDR + PAGE_SIZE), %rsp
4   entry_k_stack:  mov     %cr2, %rax
5                   mov     %rax, OFS_CR2(%rbx)
6                   mov     %rbx, %rdi
7                   call    exc_handler
8                   jmp     ret_from_interrupt
9
10  ret_from_interrupt:
11                  testb   $3, OFS_CS(%rbx)
12                  jnz     ret_user_iret
13                  pop     %r15 ; ... %rdx, %rcx, %rax
14                  add     $16, %rsp
15                  iretq
```

Kernel Exit to Kernel Mode



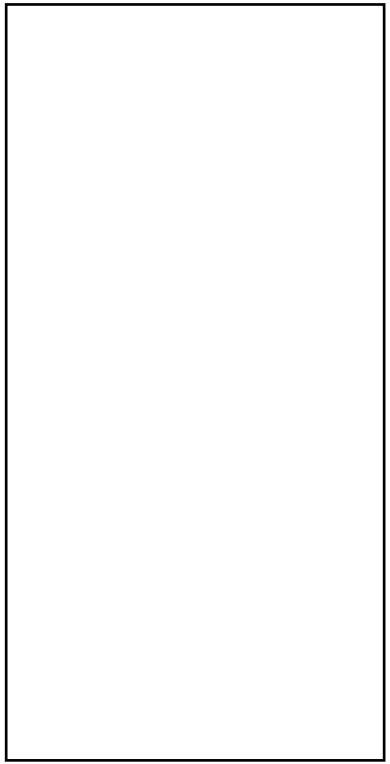

```asm
1                   ...
2                   cld
3                   mov     $(KSTCK_ADDR + PAGE_SIZE), %rsp
4   entry_k_stack:  mov     %cr2, %rax
5                   mov     %rax, OFS_CR2(%rbx)
6                   mov     %rbx, %rdi
7                   call    exc_handler
8                   jmp     ret_from_interrupt
9
10  ret_from_interrupt:
11                  testb   $3, OFS_CS(%rbx)
12                  jnz     ret_user_iret
13                  pop     %r15 ; ... %rdx, %rcx, %rax
14                  add     $16, %rsp
15                  iretq
```

Kernel Exit to Kernel Mode




```asm
1                   ...
2                   cld
3                   mov     $(KSTCK_ADDR + PAGE_SIZE), %rsp
4   entry_k_stack:  mov     %cr2, %rax
5                   mov     %rax, OFS_CR2(%rbx)
6                   mov     %rbx, %rdi
7                   call    exc_handler
8                   jmp     ret_from_interrupt
9
10  ret_from_interrupt:
11                  testb   $3, OFS_CS(%rbx)
12                  jnz     ret_user_iret
13                  pop     %r15 ; ... %rdx, %rcx, %rax
14                  add     $16, %rsp
15                  iretq
```

Kernel Exit to Kernel Mode



```asm
1                   ...
2                   cld
3                   mov     $(KSTCK_ADDR + PAGE_SIZE), %rsp
4   entry_k_stack:  mov     %cr2, %rax
5                   mov     %rax, OFS_CR2(%rbx)
6                   mov     %rbx, %rdi
7                   call    exc_handler
8                   jmp     ret_from_interrupt
9
10  ret_from_interrupt:
11                  testb   $3, OFS_CS(%rbx)
12                  jnz     ret_user_iret
13                  pop     %r15 ; ... %rdx, %rcx, %rax
14                  add     $16, %rsp
15                  iretq
```

Kernel Exit to User Mode


```asm
1                   ...
2                   cld
3                   mov     $(KSTCK_ADDR + PAGE_SIZE), %rsp
4   entry_k_stack:  mov     %cr2, %rax
5                   mov     %rax, OFS_CR2(%rbx)
6                   mov     %rbx, %rdi
7                   call    exc_handler
8                   jmp     ret_from_interrupt
9
10  ret_from_interrupt:
11                  testb   $3, OFS_CS(%rbx)
12                  jnz     ret_user_iret
13                  pop     %r15 ; ... %rdx, %rcx, %rax
14                  add     $16, %rsp
15                  iretq
```

Kernel Exit to User Mode


```asm
1                   ...
2                   cld
3                   mov     $(KSTCK_ADDR + PAGE_SIZE), %rsp
4   entry_k_stack:  mov     %cr2, %rax
5                   mov     %rax, OFS_CR2(%rbx)
6                   mov     %rbx, %rdi
7                   call    exc_handler
8                   jmp     ret_from_interrupt
9
10  ret_from_interrupt:
11                  testb   $3, OFS_CS(%rbx)
12                  jnz     ret_user_iret
13                  pop     %r15 ; ... %rdx, %rcx, %rax
14                  add     $16, %rsp
15                  iretq
```

Kernel Exit to User Mode


```C++
void Ec::ret_user_iret() {
    handle_hazards();

    asm volatile(
        "lea %0, %rsp;"
        "pop %r15;" // ... %rdx, %rcx, %rax
        "add $16, %rsp;"
        "iretq"
        :
        : "m"(current->regs)
        : "memory");
    );

    UNREACHED;
}
```

Kernel Exit to User Mode

C++

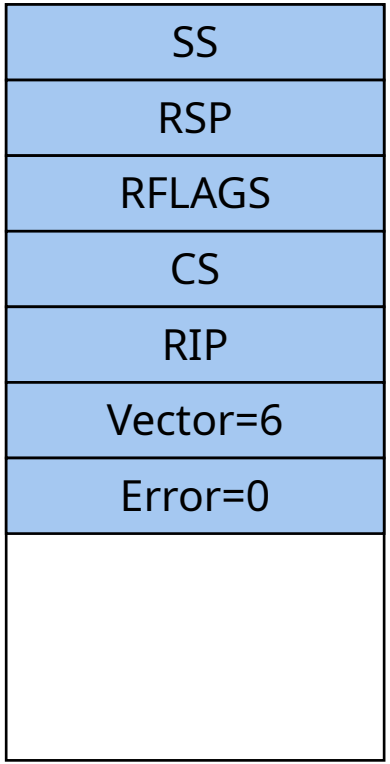
```
1   void Ec::ret_user_iret() {
2       handle_hazards();
3
4       asm volatile(
5           "lea %0, %rsp;"
6           "pop %r15;" // ... %rdx, %rcx, %rax
7           "add $16, %rsp;"
8           "iretq"
9           :
10          : "m"(current->regs)
11          : "memory");
12      );
13
14      UNREACHED;
15  }
```

Kernel Exit to User Mode

C++

```
1  void Ec::ret_user_iret() {
2      handle_hazards();
3
4      asm volatile(
5          "lea %0, %rsp;"
6          "pop %r15;" // ... %rdx, %rcx, %rax
7          "add $16, %rsp;"
8          "iretq"
9          :
10         : "m"(current->regs)
11         : "memory");
12     );
13
14     UNREACHED;
15 }
```

Kernel Exit to User Mode



C++

```
1   void Ec::ret_user_iret() {
2       handle_hazards();
3
4       asm volatile(
5           "lea %0, %rsp;"
6           "pop %r15;" // ... %rdx, %rcx, %rax
7           "add $16, %rsp;"
8           "iretq"
9           :
10          : "m"(current->regs)
11          : "memory");
12      );
13
14      UNREACHED;
15  }
```

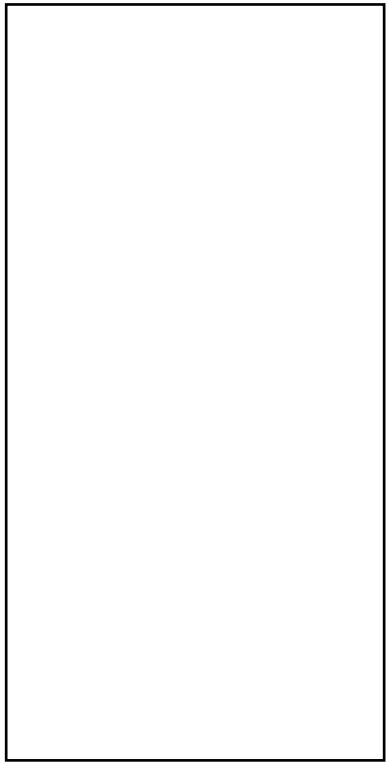
Kernel Exit to User Mode


```C++
void Ec::ret_user_iret() {
    handle_hazards();

    asm volatile(
        "lea %0, %rsp;"
        "pop %r15;" // ... %rdx, %rcx, %rax
        "add $16, %rsp;"
        "iretq"
        :
        : "m"(current->regs)
        : "memory");
    );

    UNREACHED;
}
```

Kernel Exit to User Mode



```C++
void Ec::ret_user_iret() {
    handle_hazards();

    asm volatile(
        "lea %0, %rsp;"
        "pop %r15;" // ... %rdx, %rcx, %rax
        "add $16, %rsp;"
        "iretq"
        :
        : "m"(current->regs)
        : "memory");
    );

    UNREACHED;
}
```

- x86-64 Details
- Entering NOVA
- Leaving NOVA
- **Syscall Instructions**

Kernel Exit: IRET

- Load code segment selector, `RIP` and flags from stack
- Evaluate RPL of `CS` from stack
  - `RPL > CPL`: return to other privilege level Change CPL to value of RPL
  - otherwise: return to same privilege level
- If privilege level changes
  - Load stack segment selector and `RSP` from stack
  - Adjust CPL

Fast Kernel Entry/Exit via `syscall`/`sysret`

Instructions

- `syscall`: fast ring transition from ring 3 to ring 0
- `sysret`: fast ring transition from ring 0 to ring 3

Fast Kernel Entry/Exit via `syscall`/`sysret`

Instructions

- `syscall`: fast ring transition from ring 3 to ring 0
- `sysret`: fast ring transition from ring 0 to ring 3

Setup by Kernel

- `STAR` MSR: segment selectors for kernel/user code
- `LSTAR` MSR: kernel entry point (RIP)
- `SFMASK` MSR: bits to clear/set in `RFLAGS`

Fast Kernel Entry/Exit via `syscall`/`sysret`

Executing `syscall`

- Loads kernel `CS` and `RIP`
- Clears/sets bits in `RFLAGS`
- Does *not* save user `RIP` and `RSP`

Fast Kernel Entry/Exit via `syscall`/`sysret`

Executing `syscall`

- Loads kernel `CS` and `RIP`
- Clears/sets bits in `RFLAGS`
- Does *not* save user `RIP` and `RSP`

Executing `sysret`

- Loads user `CS` and `SS`
- Loads user `RIP` from `RCX`
- Loads user `RSP` from the current stack pointer
- Restores `RFLAGS` with bits cleared/set