EC 440 – Introduction to Operating Systems

Manuel Egele

Department of Electrical &
Computer Engineering

Boston University

Slides from Christopher Kruegel, UCSB

Segmentation

- One-dimensional address space is cumbersome to deal with if different portions of the program have to grow/shrink

- Provide the virtual machine with several independent address spaces, called *segments*

- Addressing is done by specifying
  - Segment
  - Address within the segment

- Advantages
  - Easy to share code and data segments (shared libraries)
  - Different segments can have different types of protection

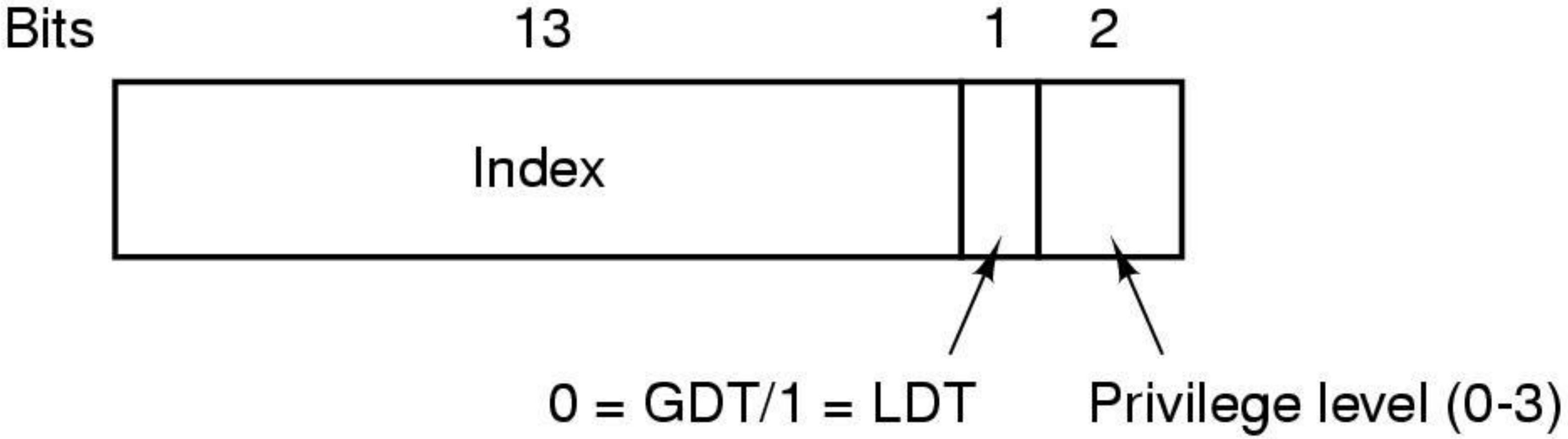
- Segmentation is usually composed with paging

Segmentation with Paging: Pentium

- Virtual memory with 16K segments
- Local Descriptor Table (LDT) for each program
- Global Descriptor Table (GDT) for the whole system
- Interrupt Descriptor Table (IDT) for the whole system
- To access a segment a selector for the segment is loaded into one of the segment registers (six in total)
  - CS holds code segment
  - DS holds data segment

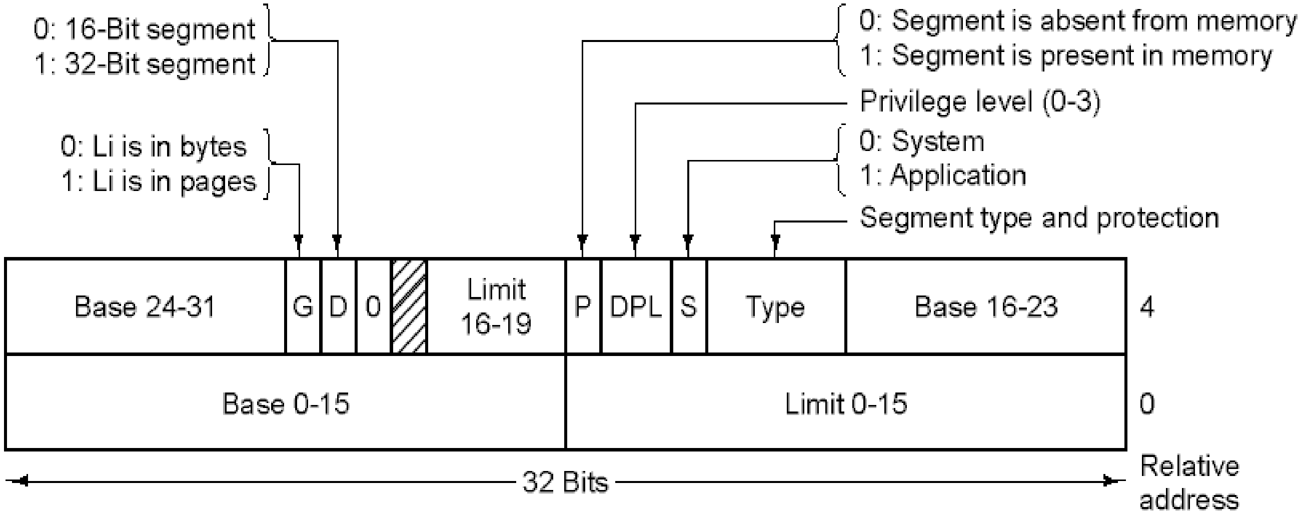
Segment Selector

- A Pentium selector contains a bit to specify if the selector is part of the GDT or the LDT (8K segments each)
- A set of bits determines the privilege level
- Segment selector determines which segment descriptor to use



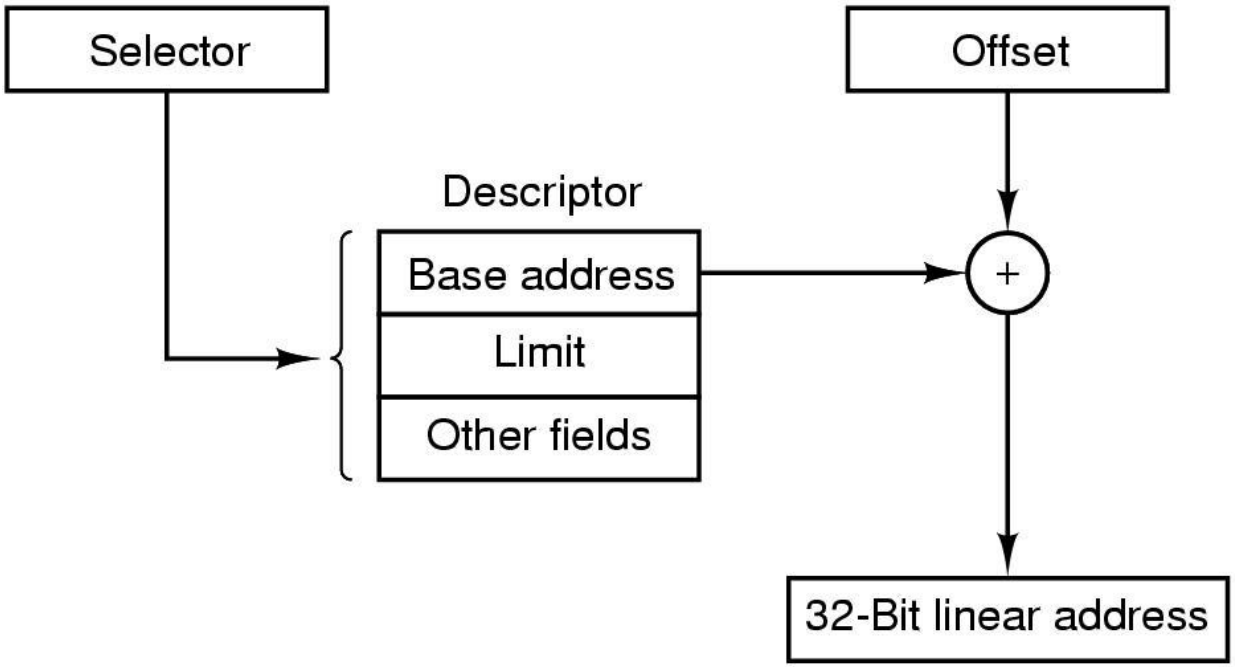
Segment Descriptor

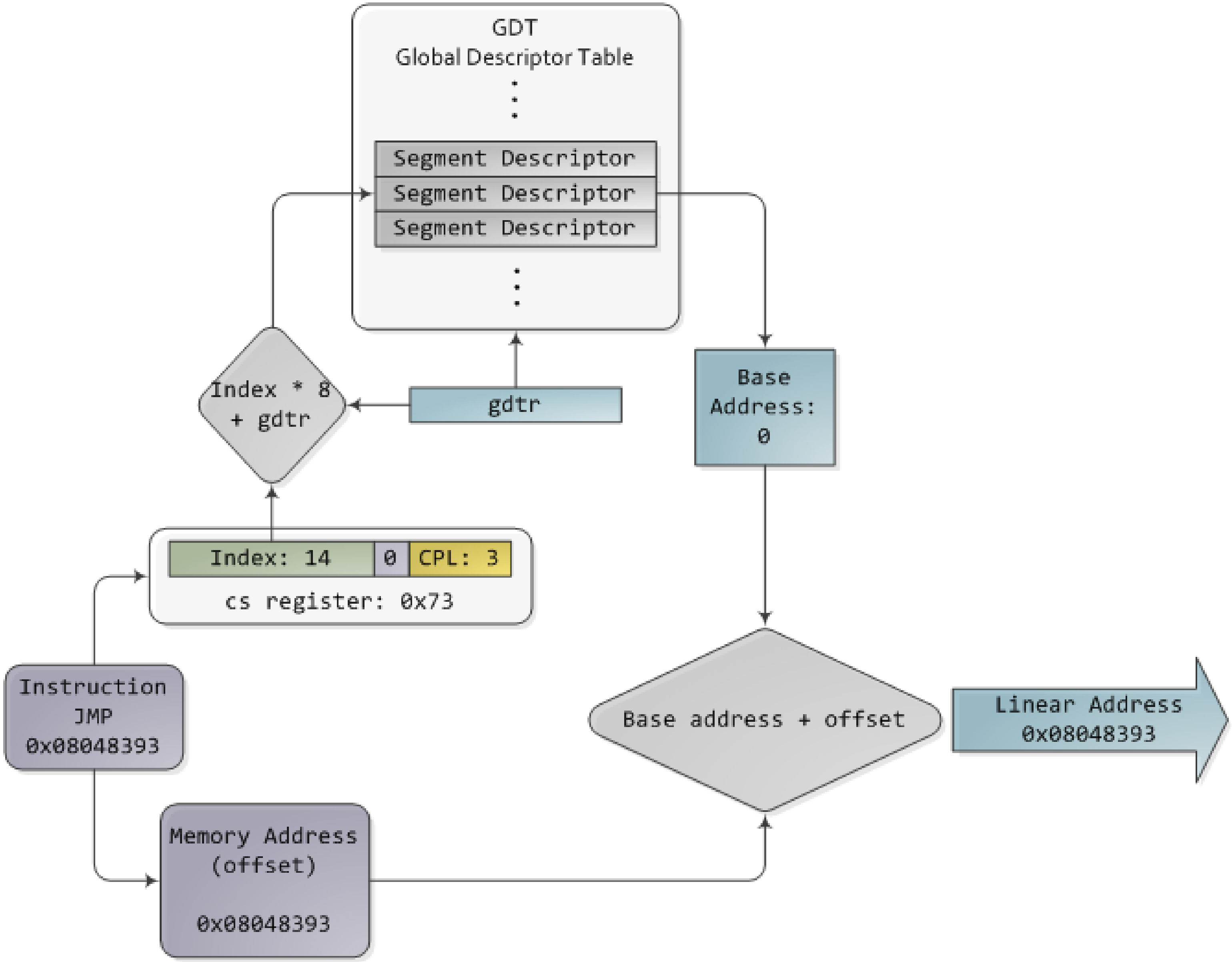
- The segment descriptor is 64 bit long
- The “limit” is expressed with 20 bits: if Granularity bit is 0, then max limit is 1MB; if the G-bit is 1, then limit is in pages of 4K (the missing 12 bits!)
- 3 Segment Types (code, data, system)



Mapping An Address

- Conversion of (selector, offset) pair to a linear address





GDT
Global Descriptor Table

Segment Descriptor
Segment Descriptor
Segment Descriptor

Index * 8 + gdtr

gdtr

Base Address: 0

Index: 14 | 0 | CPL: 3

cs register: 0x73

Instruction JMP 0x08048393

Base address + offset

Linear Address 0x08048393

Memory Address (offset)

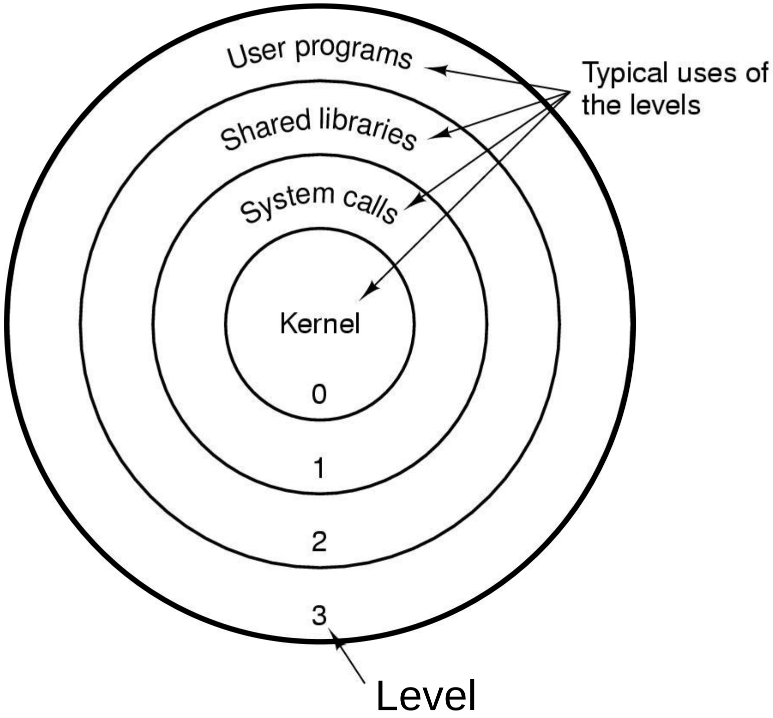
0x08048393

Protection on the Pentium

- Calls to procedures between protection levels must be performed by specifying a selector

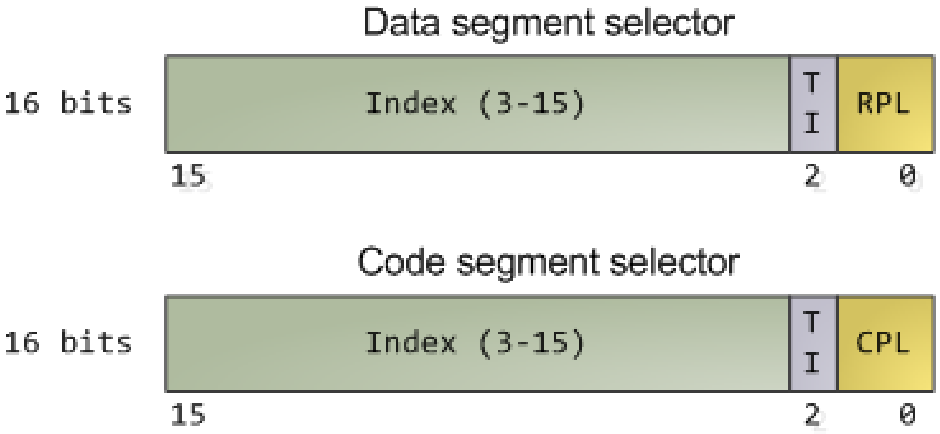
- The selector is used to locate a *call gate* that gives the address of the required procedure

- This way, it is not possible to jump to arbitrary locations

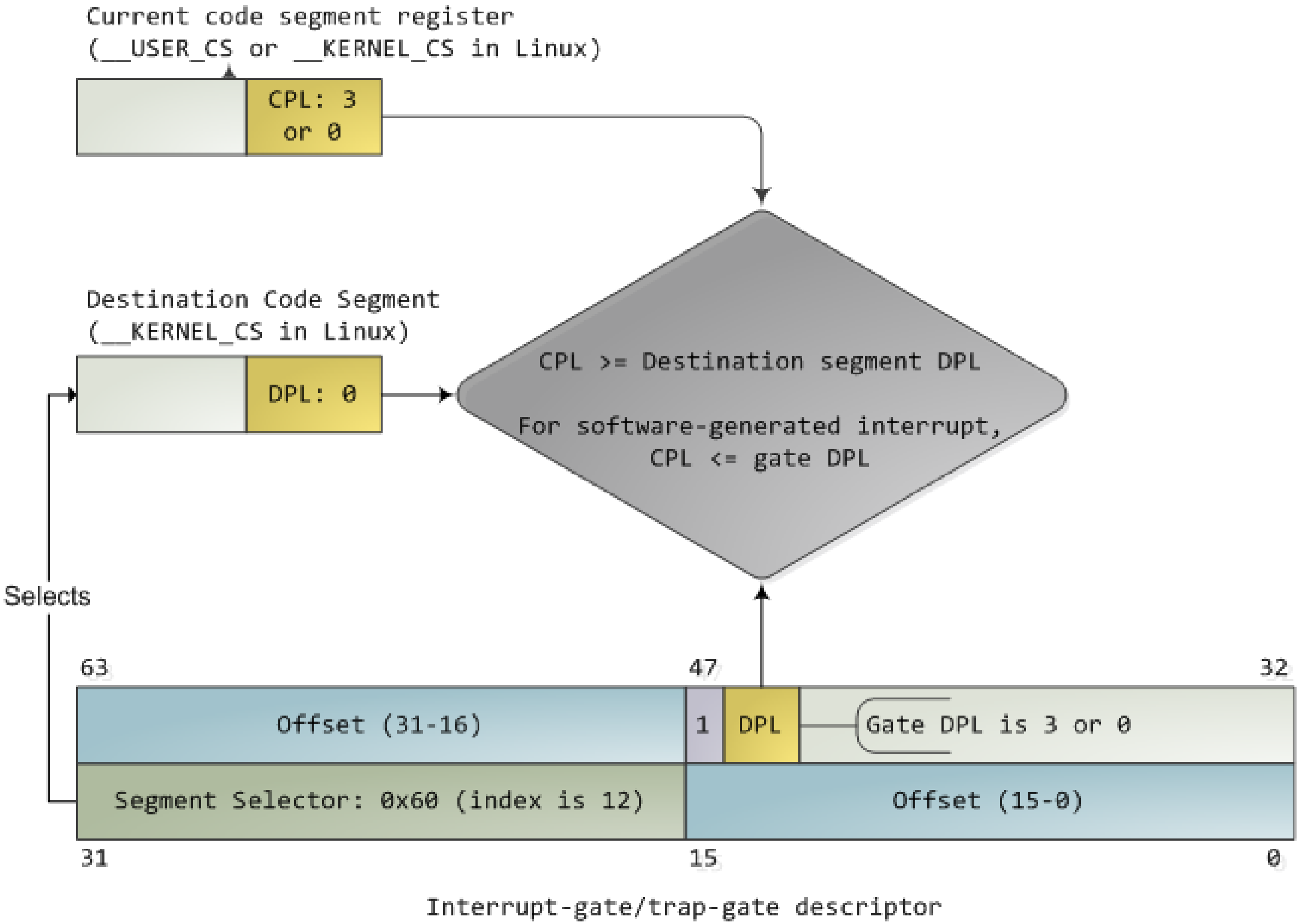


Change Rings

- Jumping from user-space to kernel-space requires that the privilege level (i.e., the ring) is changed
- How does this work?
  - Segments!



Segments & Interrupts / Syscalls


Current code segment register
(__USER_CS or __KERNEL_CS in Linux)
CPL: 3 or 0
Destination Code Segment
(__KERNEL_CS in Linux)
DPL: 0
CPL >= Destination segment DPL
For software-generated interrupt,
CPL <= gate DPL
Selects
63
47
32
Offset (31-16)
1
DPL
Gate DPL is 3 or 0
Segment Selector: 0x60 (index is 12)
Offset (15-0)
31
15
0
Interrupt-gate/trap-gate descriptor

INT 0x80 (system call) in Linux

```
/*
 * The default IDT entries which are set up in trap_init() before
 * cpu_init() is invoked. Interrupt stacks cannot be used at that point and
 * the traps which use them are reinitialized with IST after cpu_init() has
 * set up TSS.
 */
static const __initconst struct idt_data def_idts[] = {
    INTG(X86_TRAP_DE,     divide_error),
…

#if defined(CONFIG_IA32_EMULATION)
    SYSG(IA32_SYSCALL_VECTOR, entry_INT80_compat),
#elif defined(CONFIG_X86_32)
    SYSG(IA32_SYSCALL_VECTOR, entry_INT80_32),
#endif
};
```

INT 0x80 in Linux

```
struct idt_data {
     unsigned int      vector;
     unsigned int      segment;
     struct idt_bits   bits;
     const void        *addr;
};

#define DPL0          0x0
#define DPL3          0x3

#define DEFAULT_STACK     0
```

```
#define G(_vector, _addr, _ist, _type, _dpl, _segment) \
     {                              \
          .vector          = _vector,          \
          .bits.ist        = _ist,             \
          .bits.type       = _type,       \
          .bits.dpl        = _dpl,             \
          .bits.p          = 1,           \
          .addr            = _addr,       \
          .segment         = _segment,         \
     }

/* System interrupt gate */
#define SYSG(_vector, _addr)                   \
     G(_vector, _addr, DEFAULT_STACK, GATE_INTERRUPT,
DPL3, __KERNEL_CS)
```

Can be called from ring 3

Loads a (kernel) code-segment with CPL = 0 -> transitions to ring 0

Questions?