# The K Project

## Memory Management

LSE Team

EPITA

May 06, 2019

# x86 modes



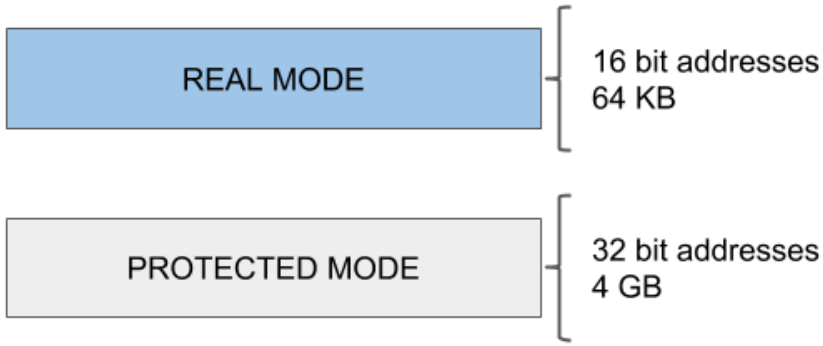



Figure: x86 modes

# Address translation



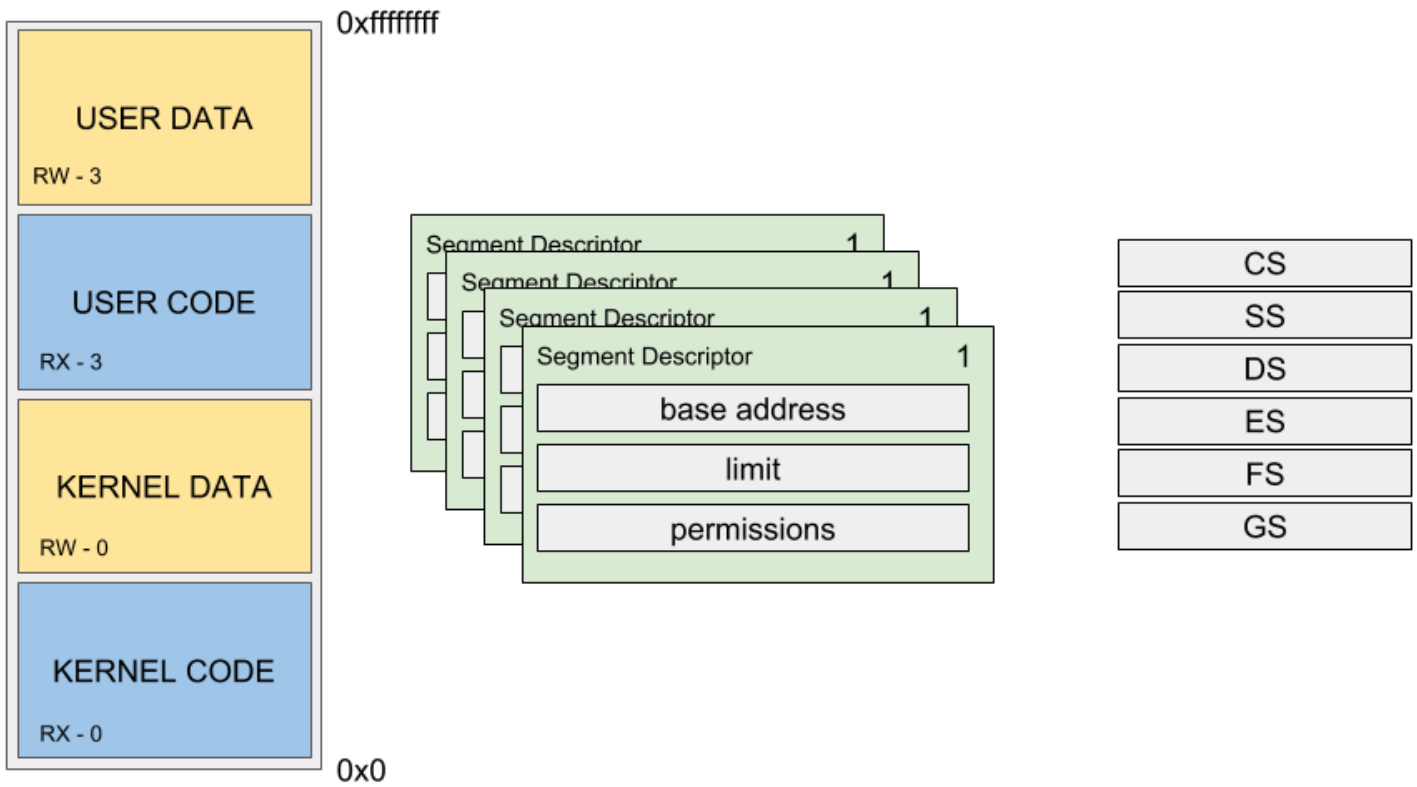



Figure: x86_segmentation

# Segmentation

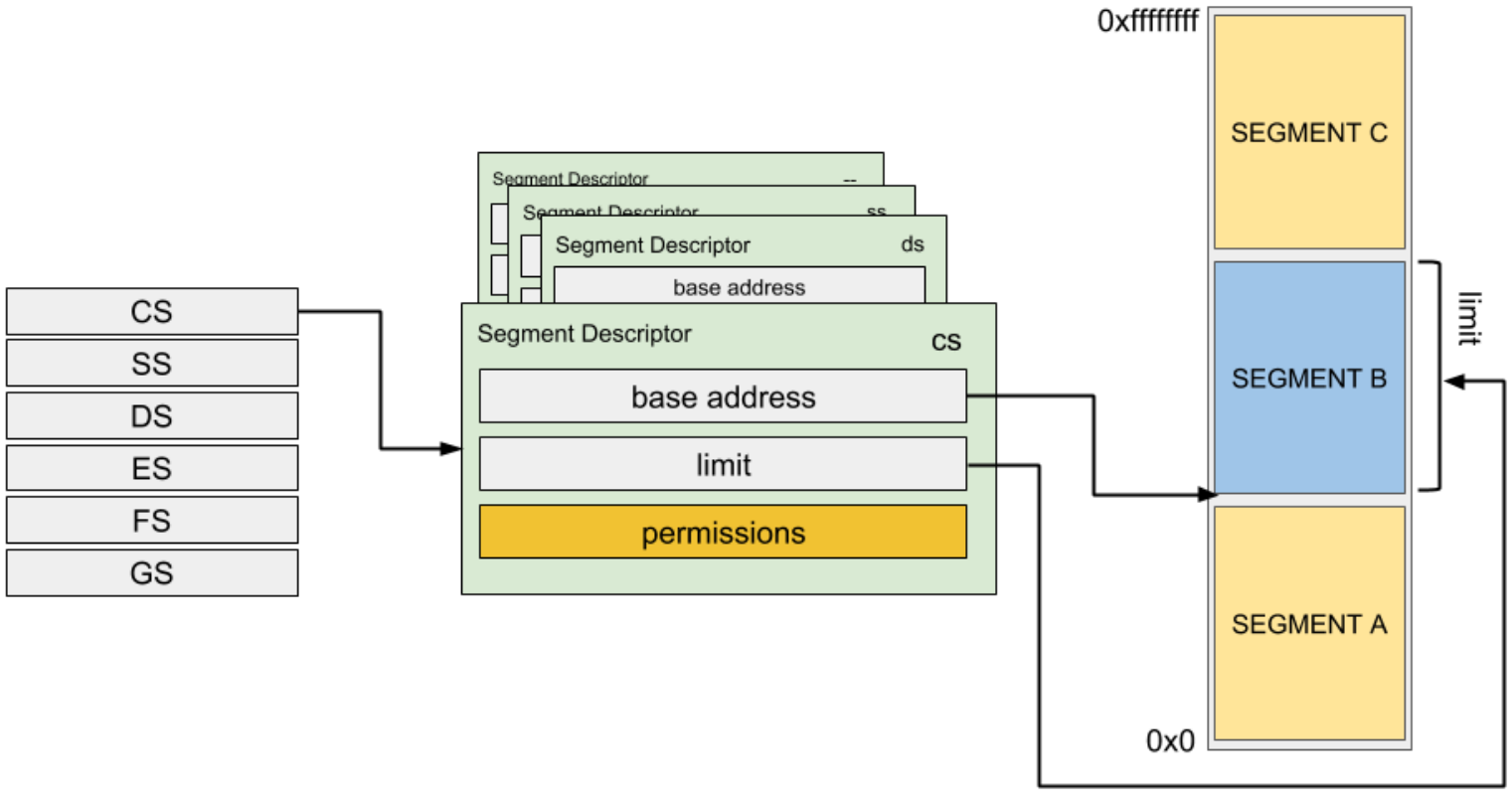



Figure: x86 segmentation

# Segment selectors



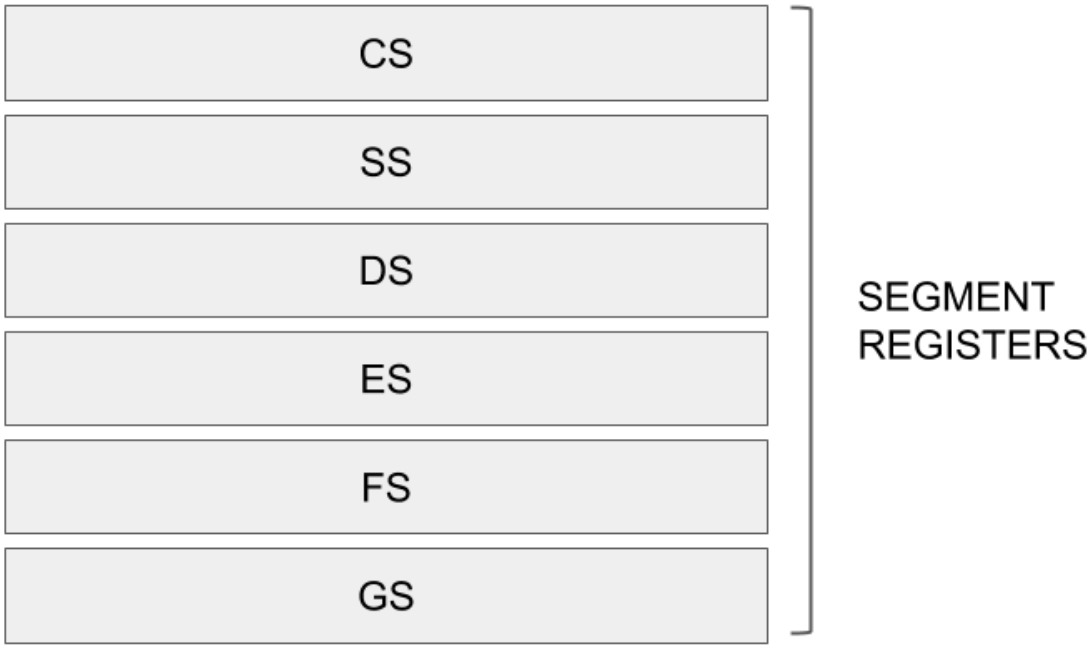



Figure: Segment selectors list

# GDT: Structure overview



- The GDT (Global Descriptor Table) is an array that contains information about every segment.
- Segment Descriptor:
  - base address
  - limit (segment size)
  - segment type (code or data)
  - access rights

First entry must be null.

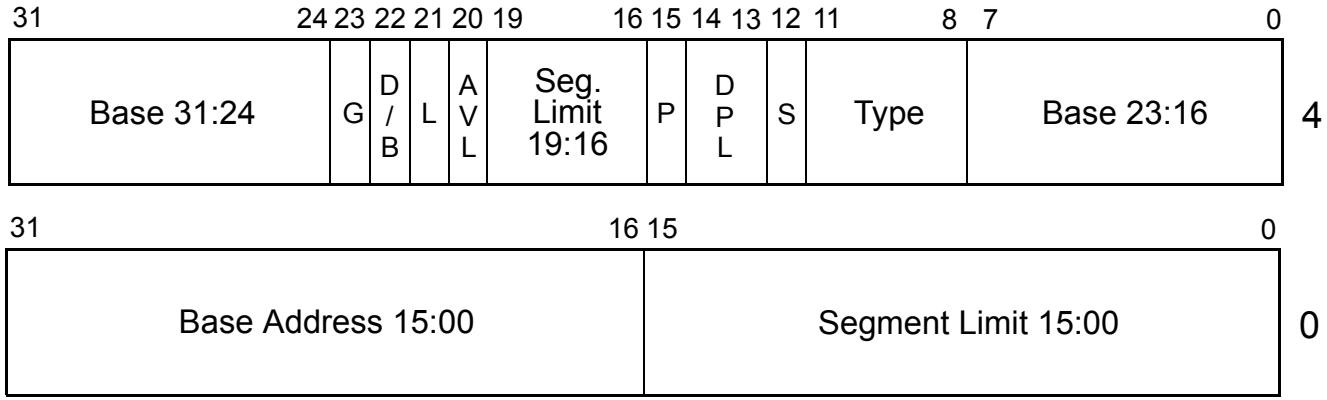
# GDT Entries



- Null segment
- Kernel Code segment
- Kernel Data segment
- Userland Code segment
- Userland Data segment
- Task State Segment (don't worry about that right now)

# GDT Entry






L — 64-bit code segment (IA-32e mode only)
AVL — Available for use by system software
BASE — Segment base address
D/B — Default operation size (0 = 16-bit segment; 1 = 32-bit segment)
DPL — Descriptor privilege level
G — Granularity
LIMIT — Segment Limit
P — Segment present
S — Descriptor type (0 = system; 1 = code or data)
TYPE — Segment type

Figure: GDT Entry description

# Entry Types and Permissions



| Type Field | | | | | Descriptor Type | Description |
|---|---|---|---|---|---|---|
| Decimal | 11 | 10<br>E | 9<br>W | 8<br>A | | |
| 0 | 0 | 0 | 0 | 0 | Data | Read-Only |
| 1 | 0 | 0 | 0 | 1 | Data | Read-Only, accessed |
| 2 | 0 | 0 | 1 | 0 | Data | Read/Write |
| 3 | 0 | 0 | 1 | 1 | Data | Read/Write, accessed |
| 4 | 0 | 1 | 0 | 0 | Data | Read-Only, expand-down |
| 5 | 0 | 1 | 0 | 1 | Data | Read-Only, expand-down, accessed |
| 6 | 0 | 1 | 1 | 0 | Data | Read/Write, expand-down |
| 7 | 0 | 1 | 1 | 1 | Data | Read/Write, expand-down, accessed |
| | | C | R | A | | |
| 8 | 1 | 0 | 0 | 0 | Code | Execute-Only |
| 9 | 1 | 0 | 0 | 1 | Code | Execute-Only, accessed |
| 10 | 1 | 0 | 1 | 0 | Code | Execute/Read |
| 11 | 1 | 0 | 1 | 1 | Code | Execute/Read, accessed |
| 12 | 1 | 1 | 0 | 0 | Code | Execute-Only, conforming |
| 13 | 1 | 1 | 0 | 1 | Code | Execute-Only, conforming, accessed |
| 14 | 1 | 1 | 1 | 0 | Code | Execute/Read, conforming |
| 15 | 1 | 1 | 1 | 1 | Code | Execute/Read, conforming, accessed |

Figure: GDT Entry permissions

# GDTR register




Figure: GDTR layout

# Loading GDT



```
gdt_r gdtr;

gdtr.base = gdt;              /* gdt base address */
gdtr.limit = sizeof(gdt) - 1    /* gdt size - 1 */

asm volatile("lgdt %0\n"
             : /* no output */
             : "m" (gdtr)
             : "memory");
```

# Segment selectors

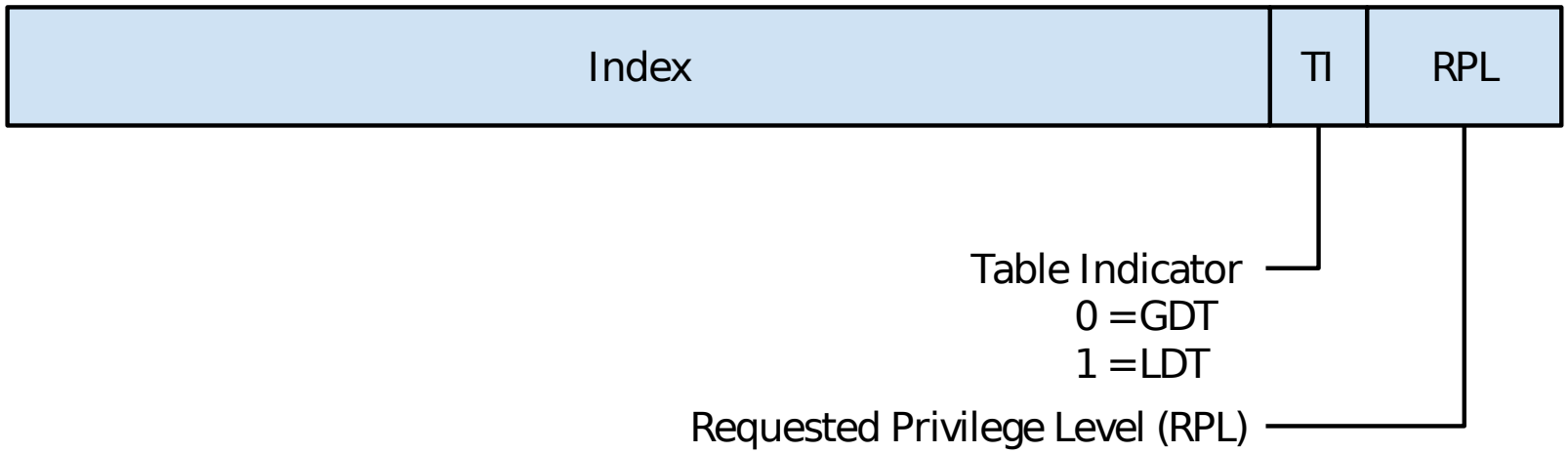



Figure: Segment selector layout

# Segment selectors



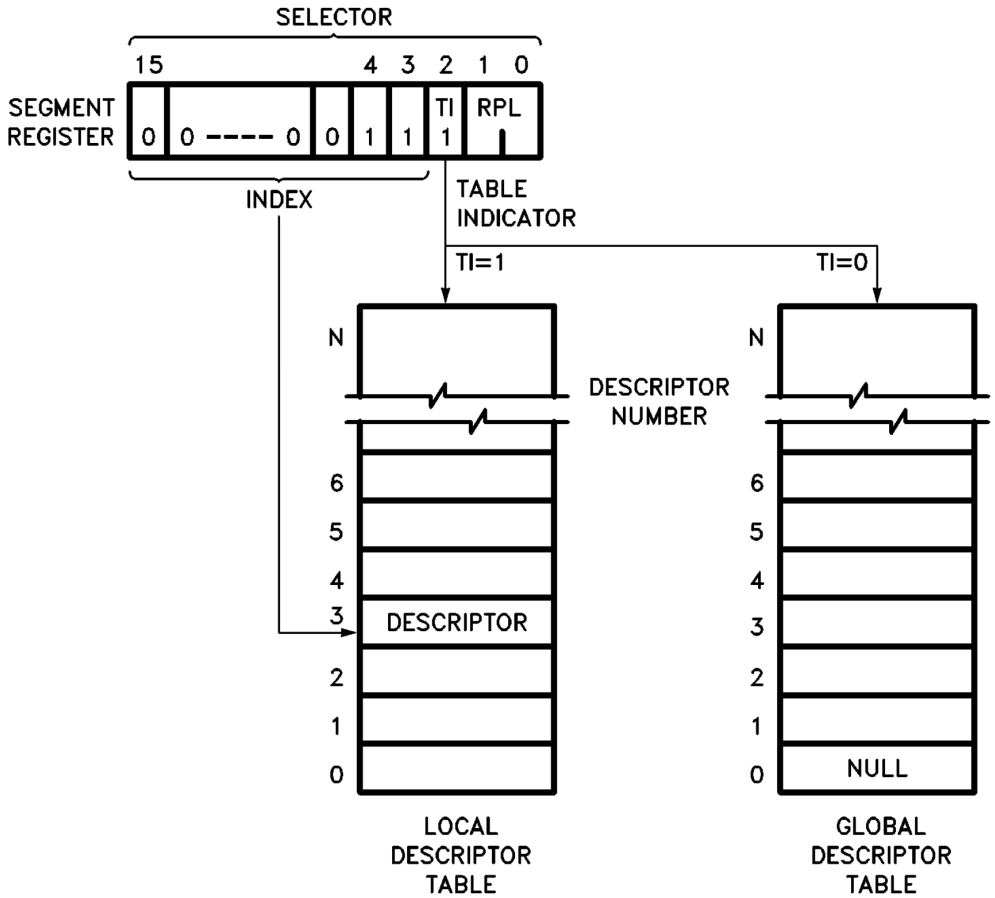



Figure: Segment selector usage

# Switching to Protected Mode



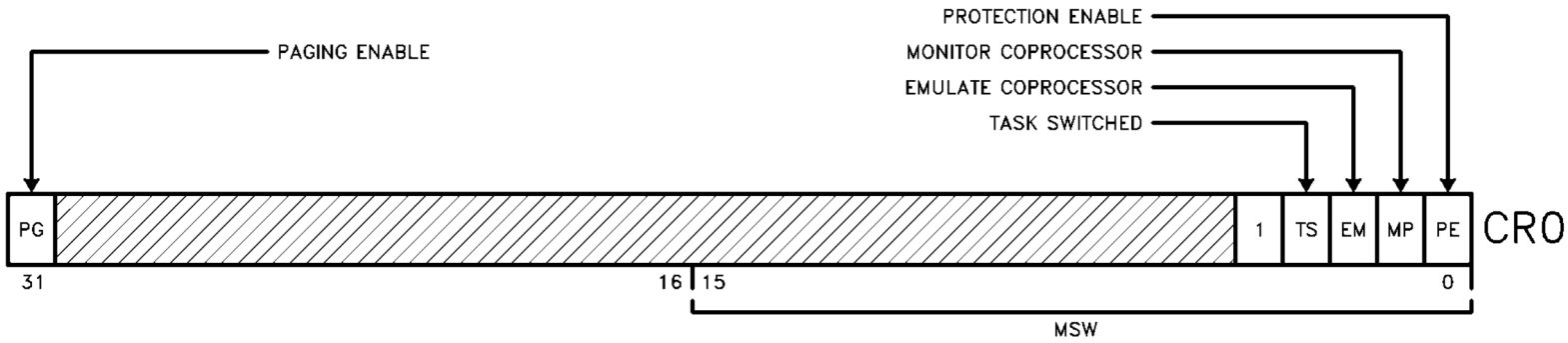



Figure: CR0 layout

# Switching to Protected Mode



You have to use an auxiliary register to set cr0:

```
movl $0x42, %cr0   ; It won't assemble

movl $0x42, %eax
movl %eax, %cr0    ; OK
```

# Switching to Protected Mode



Likewise, setting data in segment register:

```
movw $0x42, %ax
movw %ax, %ds
movw %ax, %fs
movw %ax, %gs
movw %ax, %ss
```

# Switching to Protected Mode



## First method

```
pushl $0x42          ; push %cs on the stack
pushl $1f            ; push %eip on the stack
lret                 ; far return
1:                   ; After the lret you will get
                     ; here, with cs set to 0x42
```

## Second method

```
ljmp $0x42, $1f      ; long jump
1:
```

# Switching



Again, this is not something you *have* to do in k!

# Summary



- Build GDT
- Load GDT
- Reload segment selectors

# Advices



- Use packed struct and bitfields
- Write a GDT Pretty Printer

# Contact



- k[at]lse.epita.fr
- labos.lse with [K] tag
- #k (irc.rezosup.org)
- guillaume.pagnoux[at]lse.epita.fr
- tom.decrette[at]lse.epita.fr