



# Isolation und Schutz in Betriebssystemen

## 2. Segmentierung & Tasks bei x86-64

Michael Schöttner

- x86-64: ursprünglicher Name für 64-Bit Erweiterung von AMD
  - Wurde manchmal auch abgekürzt als x64
  - Später ersetzt durch Begriff AMD64
- x86-64 hat Intel von AMD lizenziert
- Intel 64 ist Intels Gegenbegriff (für die gleiche Architektur)
  - minimale Unterschiede im Befehlssatz zu AMD64
  - aber komplett anders als die eigene IA-64 (Architektur des Itanium)

- Es gibt 16 allgemeine 64 Bit Register
- Paging ist zwingend notwendig
- Segmentierung stark eingeschränkt

- Real mode = 16 Bit Adressierung
- Protected mode = 32 Bit Adressierung, auch IA-32
- **Long mode = 64 Bit Adressierung, auch IA-32e**
  - e steht für extension
  - Verwendet in den Manuals von Intel
  - Zur Abgrenzung gegenüber dem IA-64
- (compatibility mode)
  - Unterstützung von Protected Mode Anwendungen im Long Mode

# x86-64: Registersatz

## General-purpose registers

	63	16	15	0
RAX				AX
RBX				BX
RCX				CX
RDX				DX
RSI				SI
RDI				DI
RBP				BP
R8				
⋮	⋮			
R15				

## Instruction and stack pointer

	63	16	15	0
RIP				IP
RSP				SP

## Status register

	63	16	15	0
RFLAGS				FLAGS

## Segment registers

	15	0		15	0
Code	CS		Extra	FS	
Stack	SS		Extra	GS	
Data	DS				
Extra	ES				

# x86-64: Registersatz (2)

## Memory-management registers

	15	0 63	0 31	0
TR	TSS sel.	TSS Base Address	TSS Limit	
LDTR	LDT sel.	LDT Base Address	LDT Limit	
IDTR		IDT Base Address	IDT Limit	
GDTR		GDT Base Address	GDT Limit	
			15	0

Details follow ...

## Control registers

	63	32 31	0
CR8			
CR4			
CR3			
CR2			
CR0			

## Debug registers

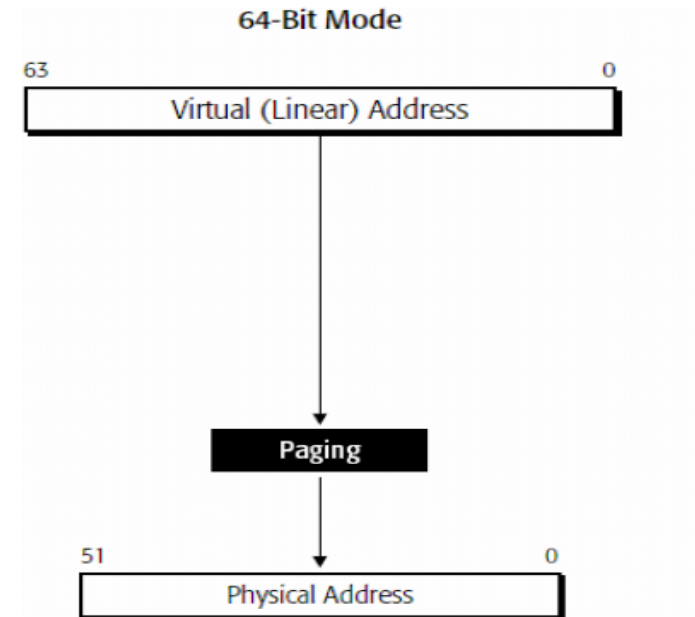
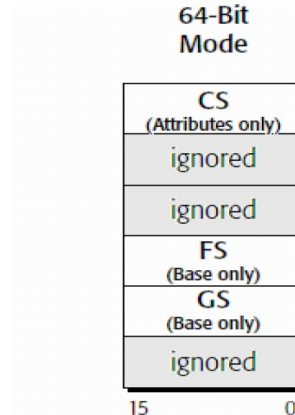
	63	32 31	0
DR0			
DR1			
DR2			
DR3			
DR6			
DR7			

## Model-specific registers (MSRs)

- Ein Segment hat eine Startadresse und ein Limit
- Segment-Register speichern Segment-Selektoren = Index in die GDT/LDT
  - LDT wurde von den meisten Betriebssystemen nicht genutzt
- Ein Programm (in Ausführung) besteht aus mehreren Speichersegmenten
  - mindestens Code, Daten und Stack
- „Lineare Adresse“ = Segmentstartadresse + Effektive Adresse
  - Effektive Adresse = Wert in einem Vielzweckregister

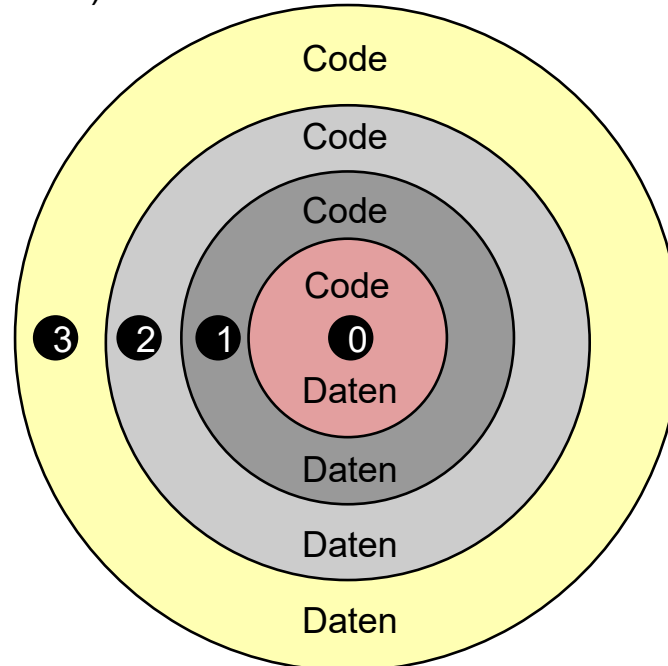
# Long Mode: Segmentierung

- 16-Bit Selektoren
- Startadressen werden bei fast allen Segmenten ignoriert, außer FS und GS
  - Werden für schnelle Systemaufrufe genutzt
- Generell keine Limit-Prüfung auf Segmentebene

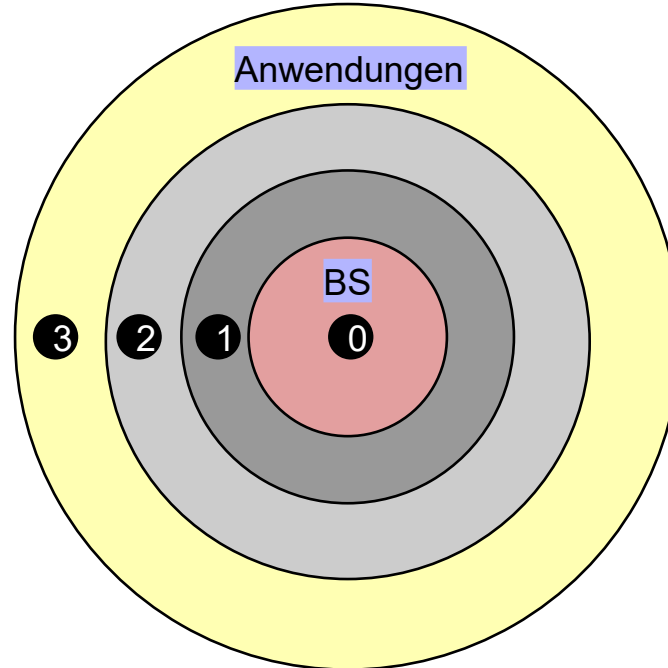


- Ziele:
  - Schutz vor unberechtigten Datenzugriffen oder Funktionsaufrufen
  - Schutz vor Absturz des Kernels respektive anderer Prozesse
  
- Voraussetzungen: Code und Daten ...
  - Werden hinsichtlich der Vertrauenswürdigkeit kategorisiert
  - Und diese Einteilung wird vom Betriebssystem mithilfe des Prozessors durchgesetzt

- Jeder Segmentdeskriptor (Code / Daten) hat 2 Bit, welche anzeigen welcher Privilegstufe das Segment zugeordnet ist
  - Ring und Privilegstufe werden gleichbedeutend verwendet
  - Hierdurch sind max. vier Privilegstufen / Ringe mögl.
  - Genutzt werden in der Praxis nur zwei



- Je kleiner die Nummer der Privilegstufe ist, desto höher die Berechtigung
- Stufe 3 ist für Anwendungen
  - = User-Mode
- Stufe 0 ist für das Betriebssystem
  - = Kernel-/Supervisor-Mode



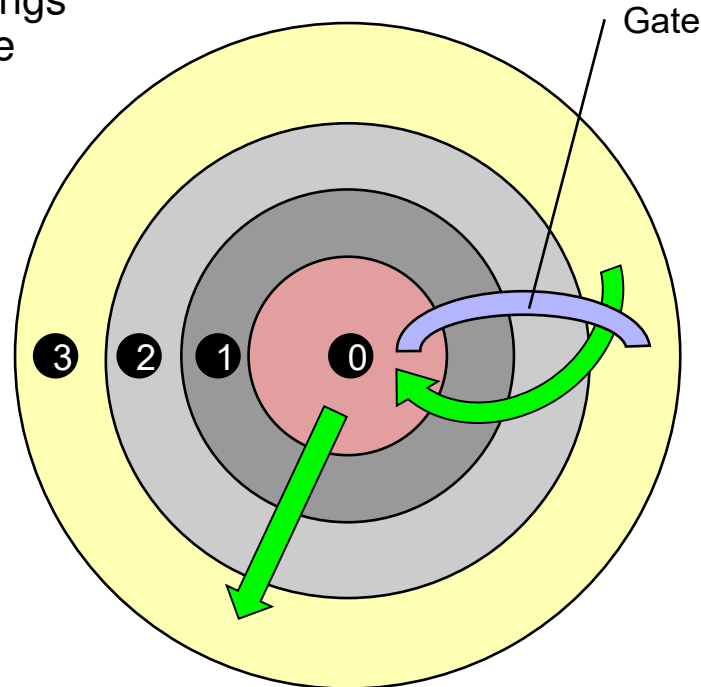
- Zugriffe auf Daten eines inneren Rings sind verboten und werden durch die MMU verhindert

- General Protection Fault

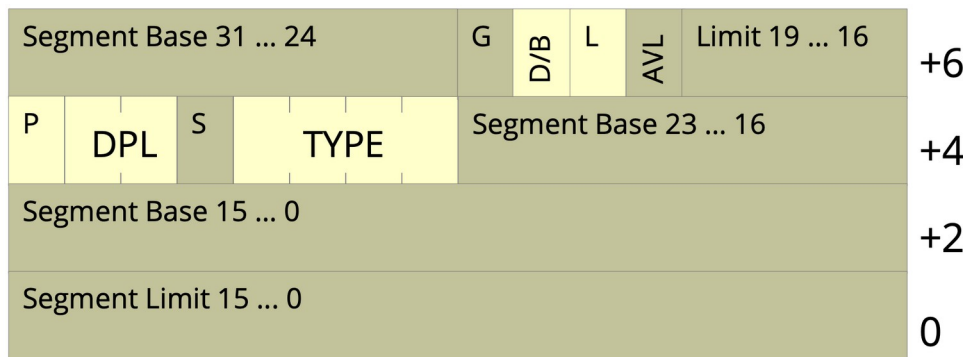
- Code eines inneren Rings kann nur mithilfe eines **Gates** aufgerufen werden

- Hiermit sind direkte Aufrufe von Ring 3 nach Ring 0 möglich
  - Call Gate oder Interrupt Gate

- Zugriffe / Aufrufe auf äußere Segmente sind erlaubt



- Stehen in der Global Descriptor Table (GDT) und erlauben Schutz von Code
  - Pro Eintrag 8 Byte (Base & Limit werden im Long Mode ignoriert)
  - Ausnahmen: Task State Segments und Gate Deskriptoren haben 16 Byte



**TYPE**  
Code/Data  
Read/Write  
Non-/Conforming

System-Segmente  
(Call-Gate, Task  
State Segment)

P Present Bit  
**DPL Descriptor Privilege Level**  
S System Segment

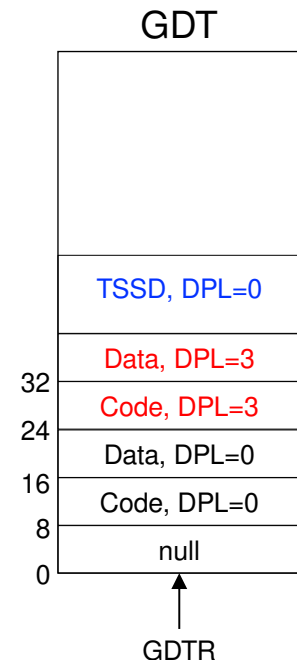
G - Granularity  
D/B - 16/32 Bit Seg.  
L - Long Mode aktiv

- Stehen in einem 16 Bit Segmentregister
- Der Inhalt umfasst den Index (13-Bit) in die GDT sowie den Requested Privilege Level (RPL)
- Privilegierungscodes:
  - Requested Privilege Level (RPL) im Segment-Register
  - Descriptor Privilege Level (DPL) im jeweiligen Deskriptor
  - Current Privilege Level (CPL) steht im aktuellen Codesegment-Register (Bit 0 und 1)
  - IOPL (2-Bits) in Flags-Register: legen fest welche Privilegstufe zum Ausführen von I/O-Befehlen notwendig ist
- Falls  $RPL > CPL$  wird der RPL verwendet, hiermit kann also die Berechtigung nur freiwillig abgeschwächt werden



# Global Descriptor Table (GDT)

- Gibt es nur ein Mal im Betriebssystem, unabhängig von der Anzahl Cores
- Global Deskriptor Table Register (GDTR) zeigt auf GDT
- Typische GDT (auch für unser BS) →
  - DPL = 0: Ring 0 (kernel mode)
  - DPL = 3: Ring 3 (user mode)
- Beispiel: `mov ds, 16` würde eine General Protection Fault auslösen, wenn dies im User-Mode gemacht würde
- TSSD folgt gleich



- Da im Long Mode bei der Segmentierung die Basis und das Limit nicht mehr beachtet wird, kann der Zugriff auf bestimmte Adressbereiche auf Segmentebene nicht eingeschränkt werden
- Dies erfolgt daher auf der Paging-Ebene wo wir für jede 4 KB Seite in den Seitentabellen festlegen können, ob diese Seite im User-Mode zugreifbar ist, oder nur im Kernel-Mode
- Zudem gibt es auf Paging-Ebene auch die Möglichkeit die Art des Zugriffs zu beschränken: read only, no execute etc.
- Paging besprechen wir in der Vorlesung später, wenn wir es programmieren

- Damit User- und Kerne-Mode genutzt werden kann wird ein TSS pro Core benötigt
  - Wo sich das TSS befindet wird durch einen TSS Deskriptor in der GDT beschrieben
  - Der TSS-Deskriptor wird wiederum vom TSS-Register referenziert
- Bei 32 Bit konnte in einem TSS der Zustand eines Cores (alle Register etc.) mit HW-Unterstützung gesichert werden, wurde aber nie wirklich genutzt und wurde daher im Long Mode abgeschafft
- Nun werden im TSS „nur“ noch Stacks gespeichert und optional gibt es noch die I/O-Permission Bitmap (siehe gleich)

## ■ IST = Interrupt Stack Table

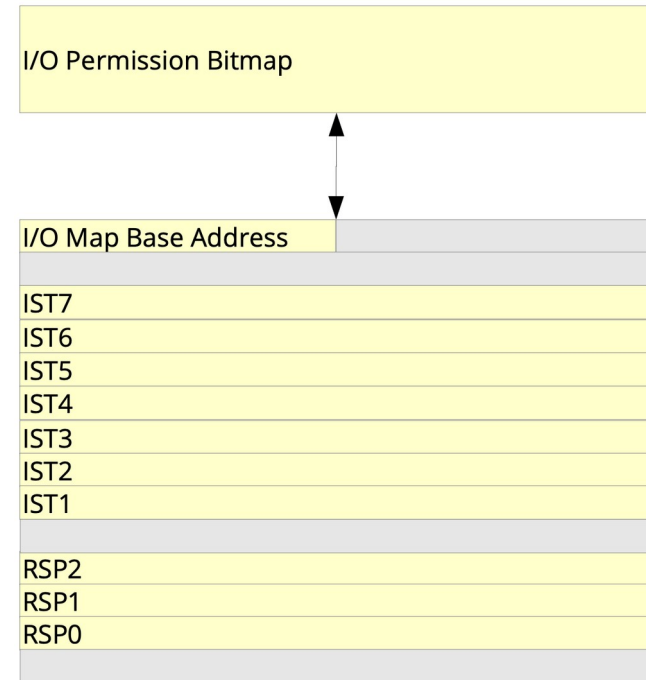
- Optional separate Stacks für Interrupt-Handler
- IST-Nummer kann in einem Eintrag in der Interrupt-Descriptor Table (IDT) angegeben werden

## ■ RSP0, RSP1, RSP2

- Stack-Zeiger für Ring 0 – 2
- Bei einem Ring Wechsel wird der Stack umgeschaltet und der Core findet den Stack-Zeiger über das TSS
- Die meisten Betriebssysteme verwenden nur zwei Ringe, wie wir, sodass wir nur RSP0 verwenden

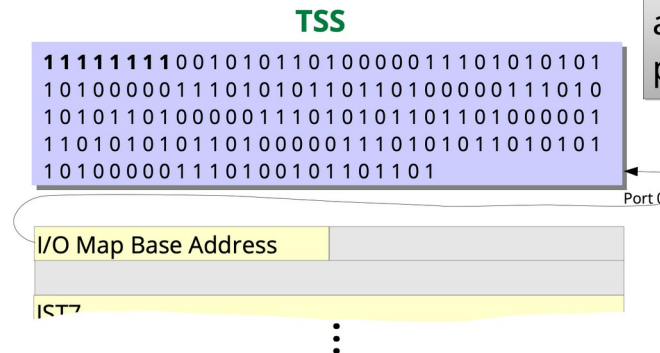
## ■ RSP3?

- Fehlt? → Nein, wird auf dem Stack gesichert bei einem Ringwechsel per Call-Gate oder Interrupt-Gate (siehe später)



- Zugriffsschutz für Port-Zugriffe (weniger wichtig heute)
  - **I/O Privilege Level** (IOPL) der aktuellen Task steht in zwei Bits im RFLAGS-Register
  - Portzugriff ist erlaubt, falls  $CPL \leq IOPL$
  - Andernfalls wird die **I/O-Permission-Bitmap** verwendet
    - Je ein Bit pro Port; falls Bit gesetzt ist, so ist der Zugriff nicht erlaubt
- Zugriffsschutz für Memory-Mapped I/O über die Seitentabelle

Bitmap ends with the TSS segment's end, ports with higher numbers must not be accessed.



an 1 prevents port access

- Der Taskwechsel findet im Kernel statt, ausgelöst durch den Timer-Interrupt
- Alle Register sichern wir auf dem Stack des Threads der verdrängt wird.
  - Das ist der Kernel-Stack, da durch den Interrupt in Ring 0 gewechselt wurde
  - Zusätzlich wird auch alles aus dem TSS auf dem Stack gesichert, außer RSP0
    - (wir sichern nichts aus dem TSS in unserem BS, da wir nur RSP0 verwenden)
  - Und zum Schluss sichern wir RSP0 im Heap in der Thread-Verwaltungsstruktur
- An den gesicherten Zustand des Threads, auf den umgeschaltet wird, gelangen wir über das zuvor gesicherte RSP0 aus dessen Verwaltungsstruktur

- x86-64 Architektur ist sehr komplex
- Selten genutzte Eigenschaften wurden im Long Mode entfernt
  - Segmentierung
  - Task-Umschaltung in Hardware
- Aber immer noch rückwärtskompatibel
  - Prozessor respektive Cores startet/n immer noch im Real Mode
  - Wir verwenden für das Booten grub, sodass grub für uns schon in den 32 Bit Protected Mode schaltet
- Intel X86S wollte dies vereinfachen, indem viel Rückwärtskompatibilität verworfen wird; Intel hat jedoch die Arbeit an X86S eingestellt...