Linux Kernel Programmierung

graphic design is my passion.

Warum?

Themenfelder Kernelentwicklung

- Prozesse
  - Scheduling, Preemption, Schlafen
  - Syscalls
- VFS, Dateisysteme und Block IO
- Memory Managment
- Network
- Interrupts, Treiber, etc.
- Locking

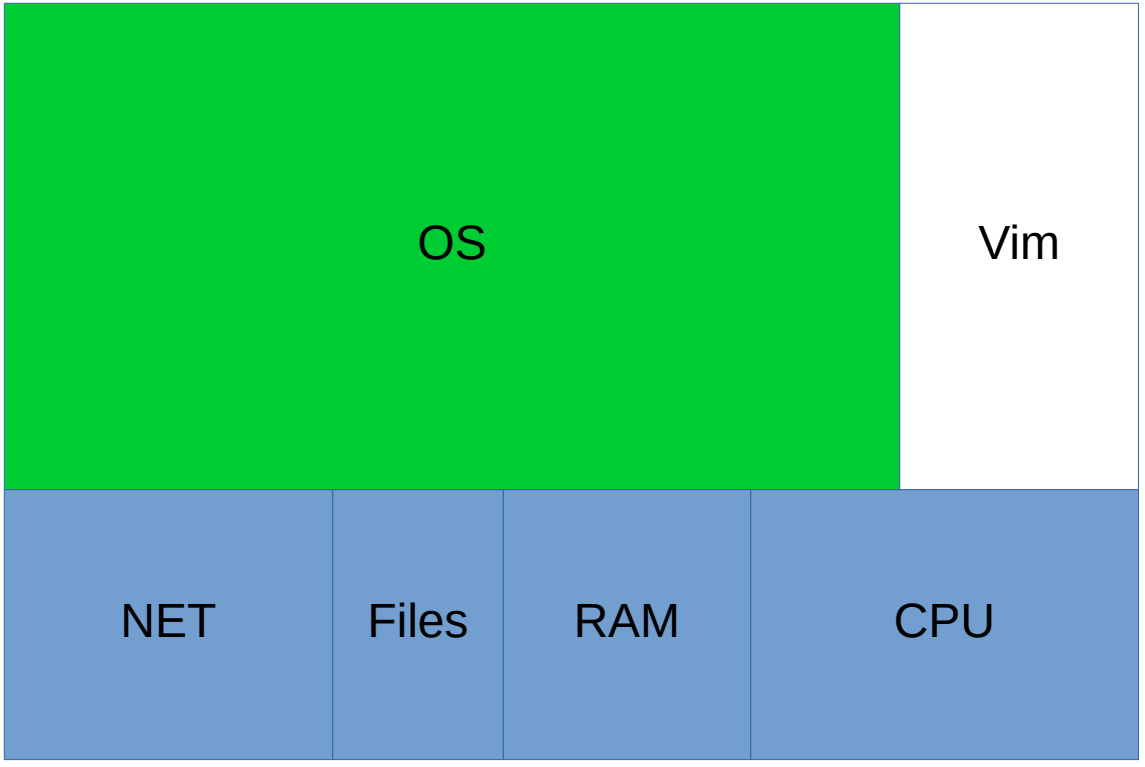
Überblick (Was)

1) Was macht der Kernel überhaupt?
2) System Calls und Scheduling
3) Interrupts
4) Memory?

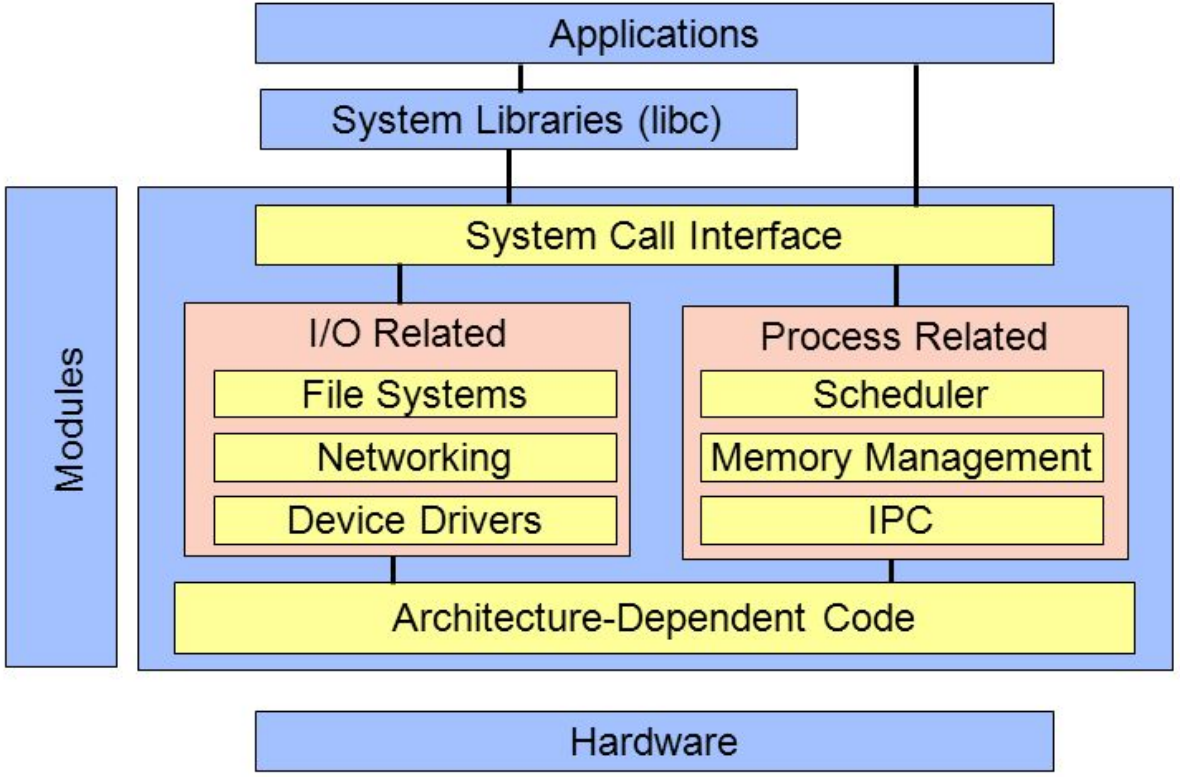
Gute Quellen

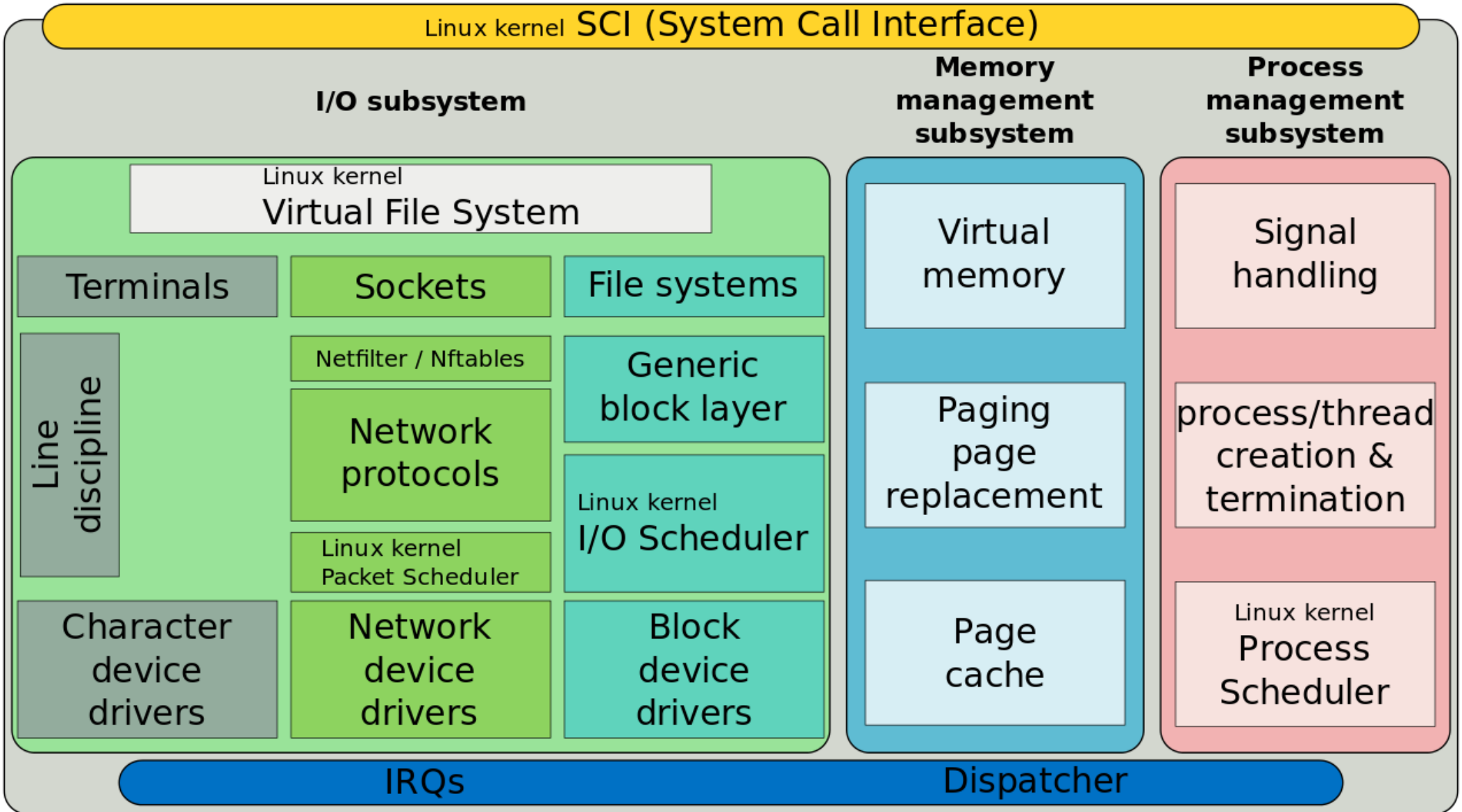
- Linux Kernel Development von Robert Love

- Unreliable Guide To Hacking The Linux Kernel (einfach googlen, ist im linux kernel source repo dabei)

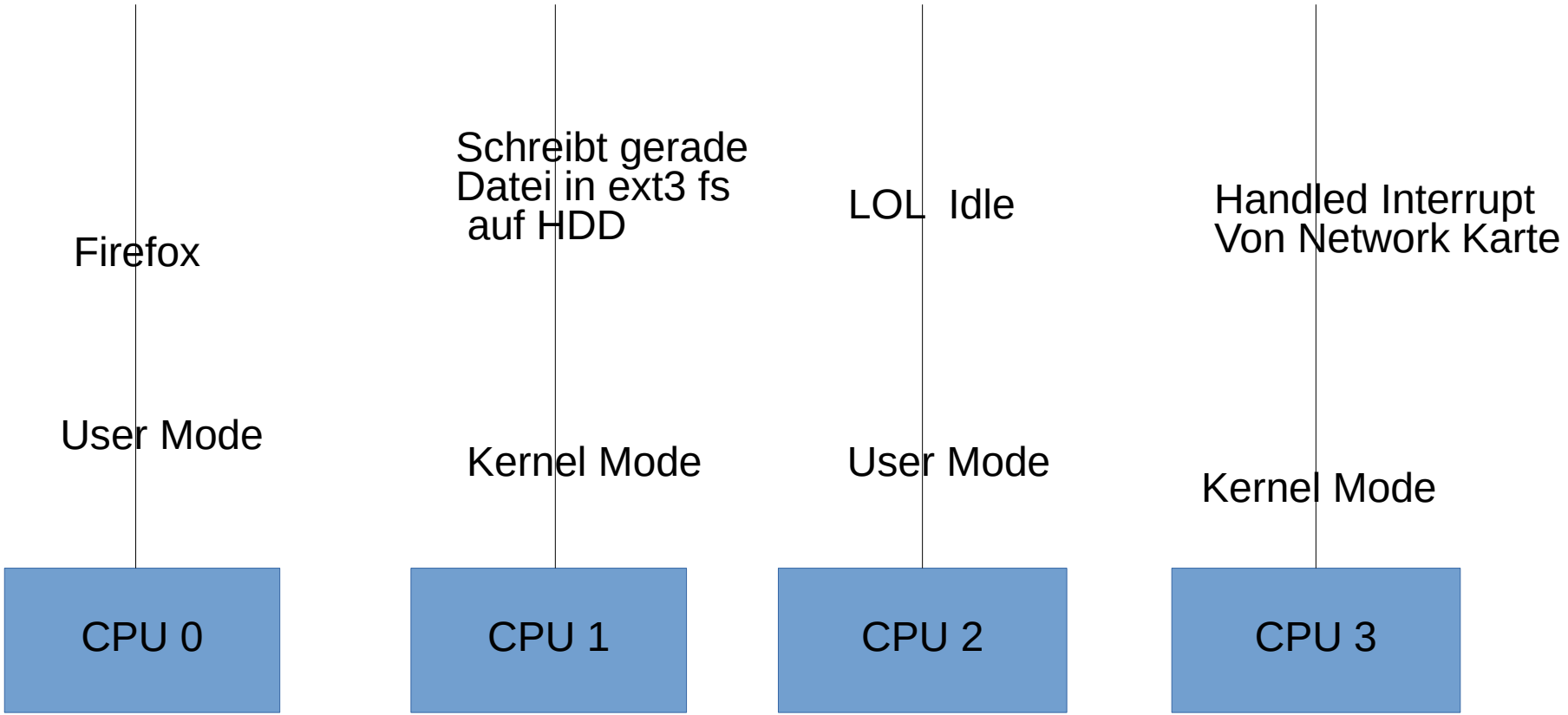

OS
Vim
NET
Files
RAM
CPU

Linux Architecture




Linux kernel SCI (System Call Interface)
I/O subsystem
Memory management subsystem
Process management subsystem
Linux kernel
Virtual File System
Terminals
Sockets
File systems
Line discipline
Netfilter / Nftables
Network protocols
Linux kernel
Packet Scheduler
Generic block layer
Linux kernel
I/O Scheduler
Character device drivers
Network device drivers
Block device drivers
Virtual memory
Paging page replacement
Page cache
Signal handling
process/thread creation & termination
Linux kernel
Process Scheduler
IRQs
Dispatcher

Und wie ist das bei mehreren CPU-Cores?


Firefox
User Mode
CPU 0
Schreibt gerade
Datei in ext3 fs
auf HDD
Kernel Mode
CPU 1
LOL Idle
User Mode
CPU 2
Handled Interrupt
Von Network Karte
Kernel Mode
CPU 3

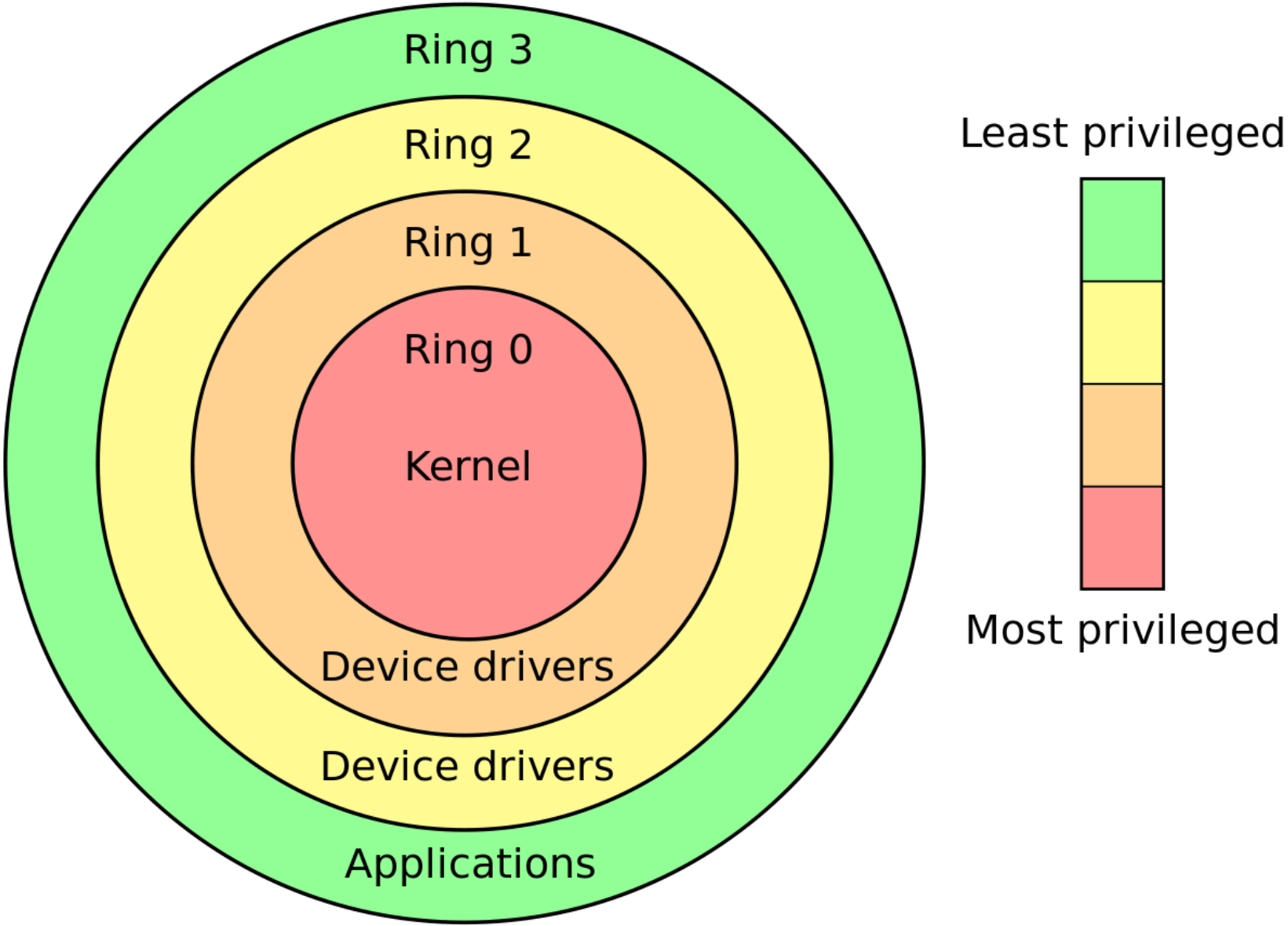
User Mode vs. Kernel Mode

Code in User Mode:

- Darf nichts

Code in Kernel Mode:

- Darf alles


Ring 3
Ring 2
Ring 1
Ring 0
Kernel
Device drivers
Device drivers
Applications
Least privileged
Most privileged

Mode Transitions

- Sind billig

- User Mode → Kernel Mode
  - (Syscall)
- Kernel Mode → User Mode
  - Return von syscall
  - Prozess der vom scheduler unterbrochen wurde wrd wieder gescheduled

Process Context vs. Interrupt Context

Code im Interrupt Kontext

- Darf nicht schlafen
- Man will da nur kurz sein
- Eigener Stack
- Unabhängig von einem Prozess

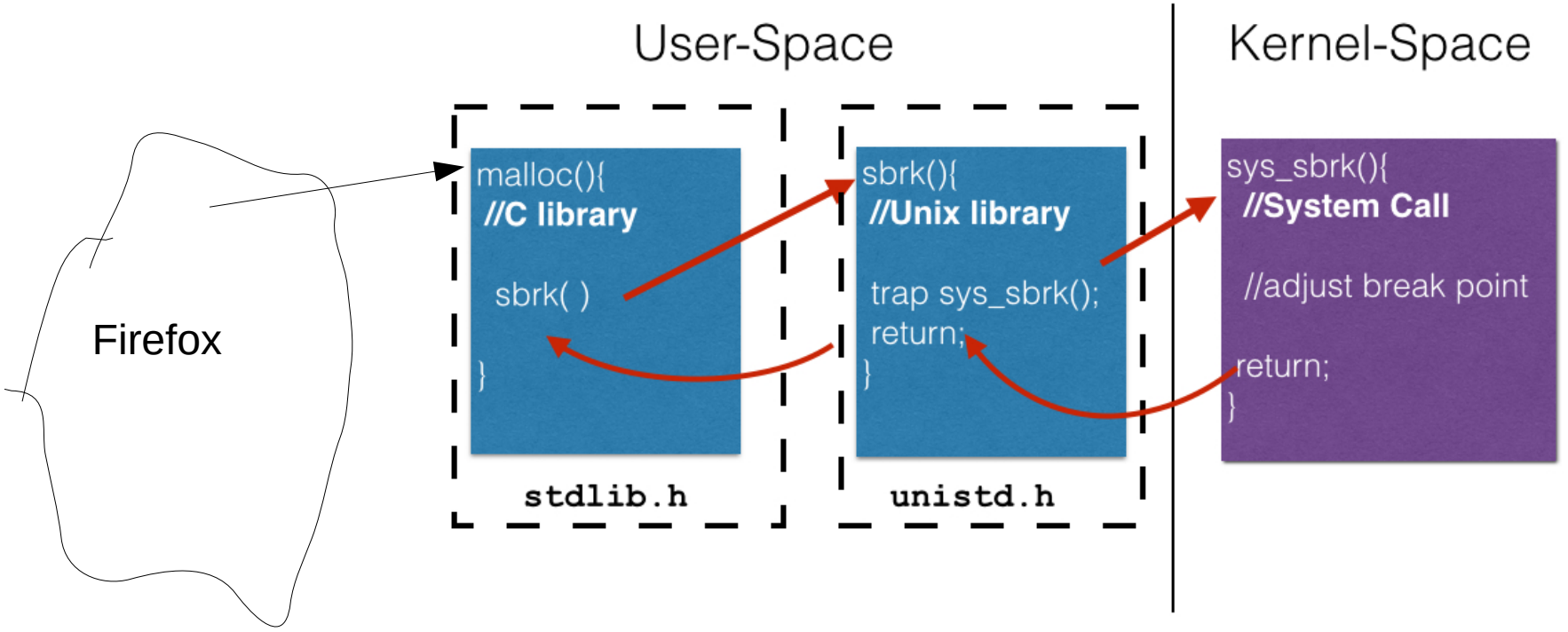
Prozess Kontext

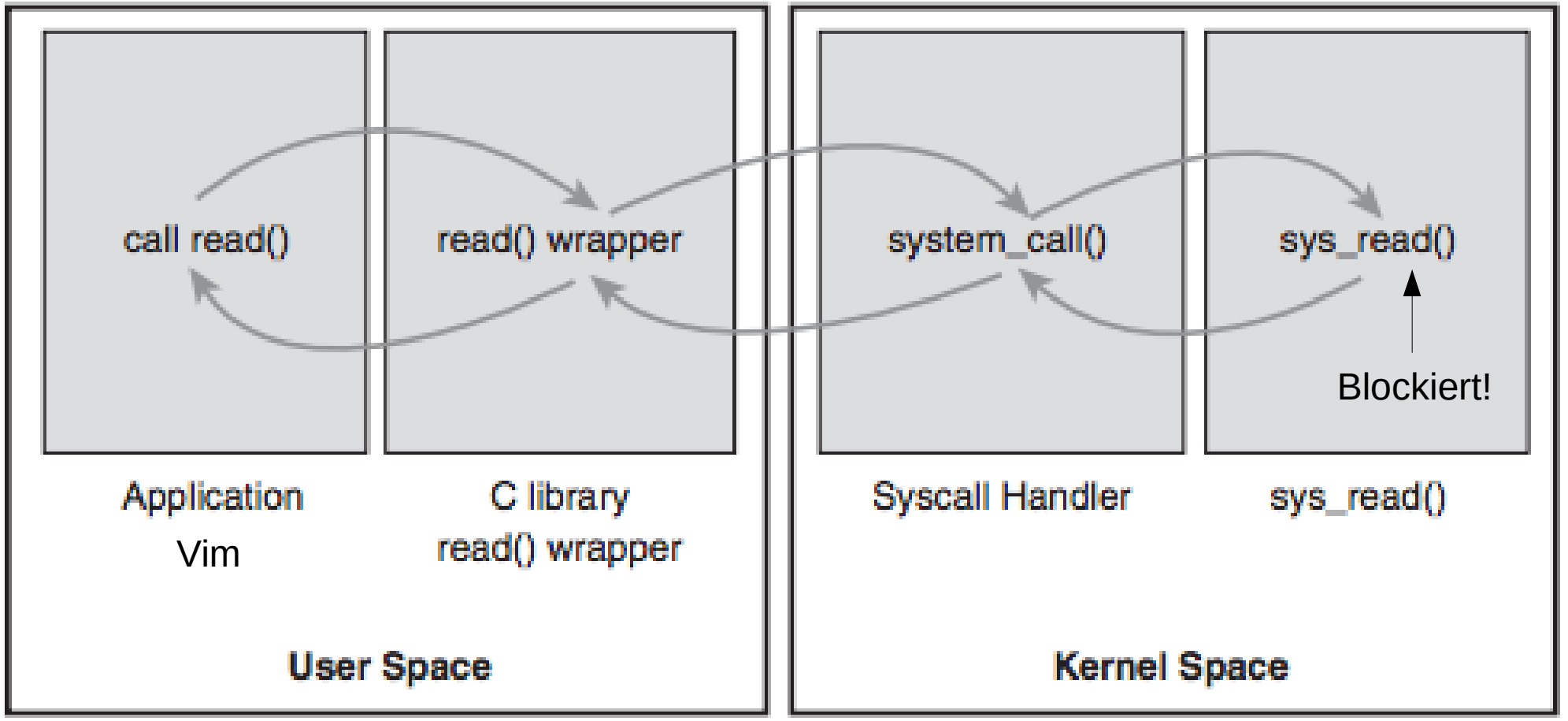
- Man kann und darf schlafen
- Relatiert zu einem Prozess
- Man muss nicht bresieren

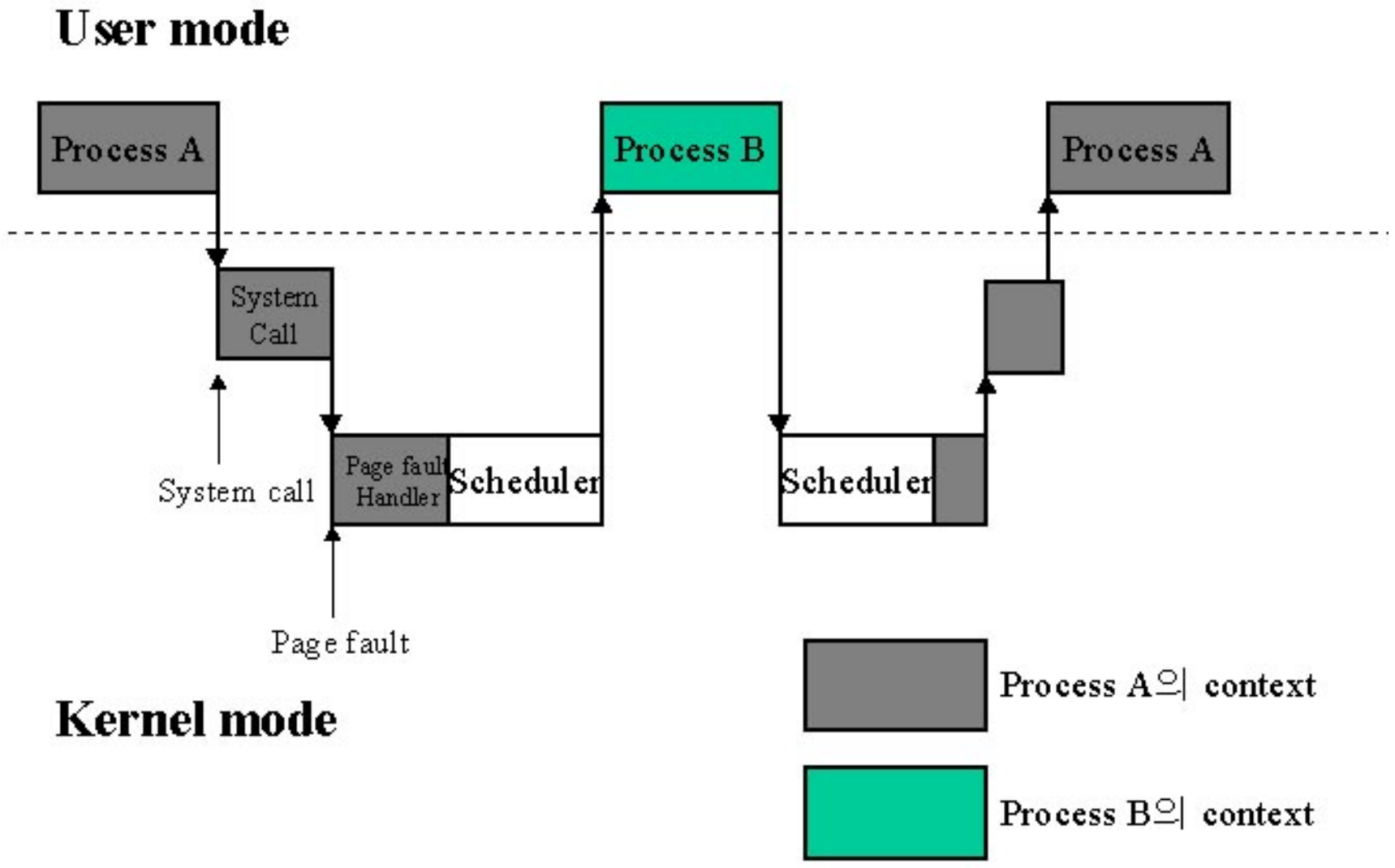
Wie hängt das zusammen?

	User Mode	Kernel Mode
Process Kontext	Normaler Prozess	Syscalls, Kernel Threads, großteil der “Kernel Arbeit”
Interrupt Kontext	Gibts net	Interrupt Handler

Syscalls


User-Space
Kernel-Space
Firefox
malloc(){
//C library
sbrk( )
}
stdlib.h
sbrk(){
//Unix library
trap sys_sbrk();
return;
}
unistd.h
sys_sbrk(){
//System Call
//adjust break point
return;
}


call read()
read() wrapper
system_call()
sys_read()
Blockiert!
Application
Vim
C library
read() wrapper
Syscall Handler
sys_read()
User Space
Kernel Space


User mode
Process A
Process B
Process A
System Call
System call
Page fault Handler
Scheduler
Scheduler
Page fault
Kernel mode
Process A의 context
Process B의 context

Context Switch (Process Switch)

Enthält:

- 2 Mode Transitions
- Register und Prozess State Saven
- Memory Mapping Switch

- Also:
- Cashes kühlen ab
- TLB (Translation Lookaside Buffer) kühlt ab

Thread Switch

System Timer
(fires)

Tick

Jiffy

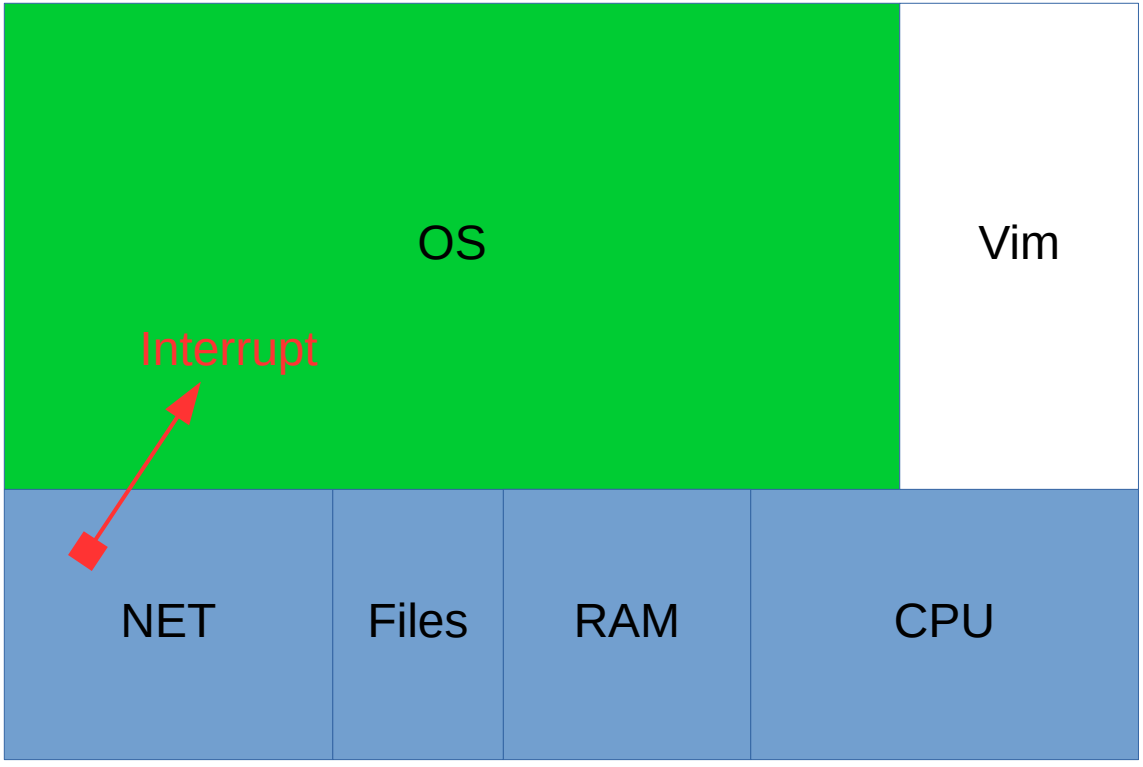
250 Hz

Interrupts (IRQs)

Dinge die Interrupts senden:

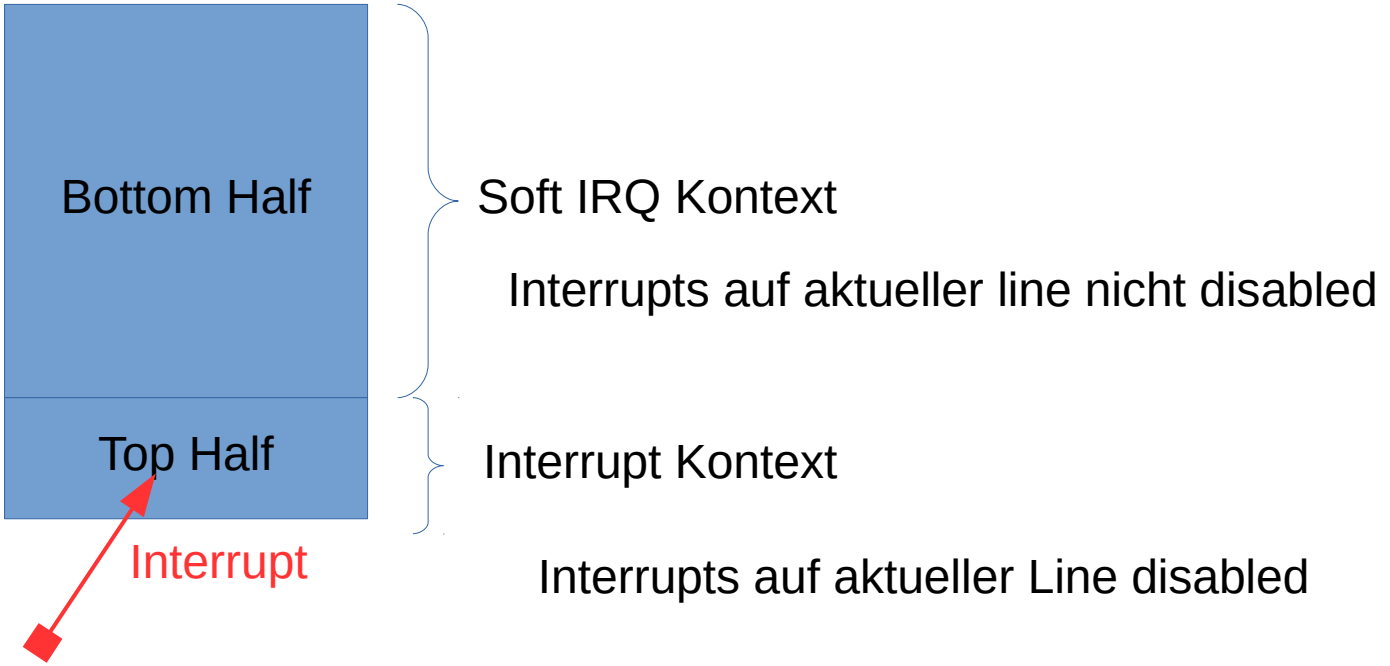
- Maus
- Tastatur
- Netzwerkkarte
- HDD?
- USB
- Hardware halt

/proc/interrupts


OS
Vim
Interrupt
NET
Files
RAM
CPU

Interrupt Handler (ISRs)

Top Half vs. Bottom Half

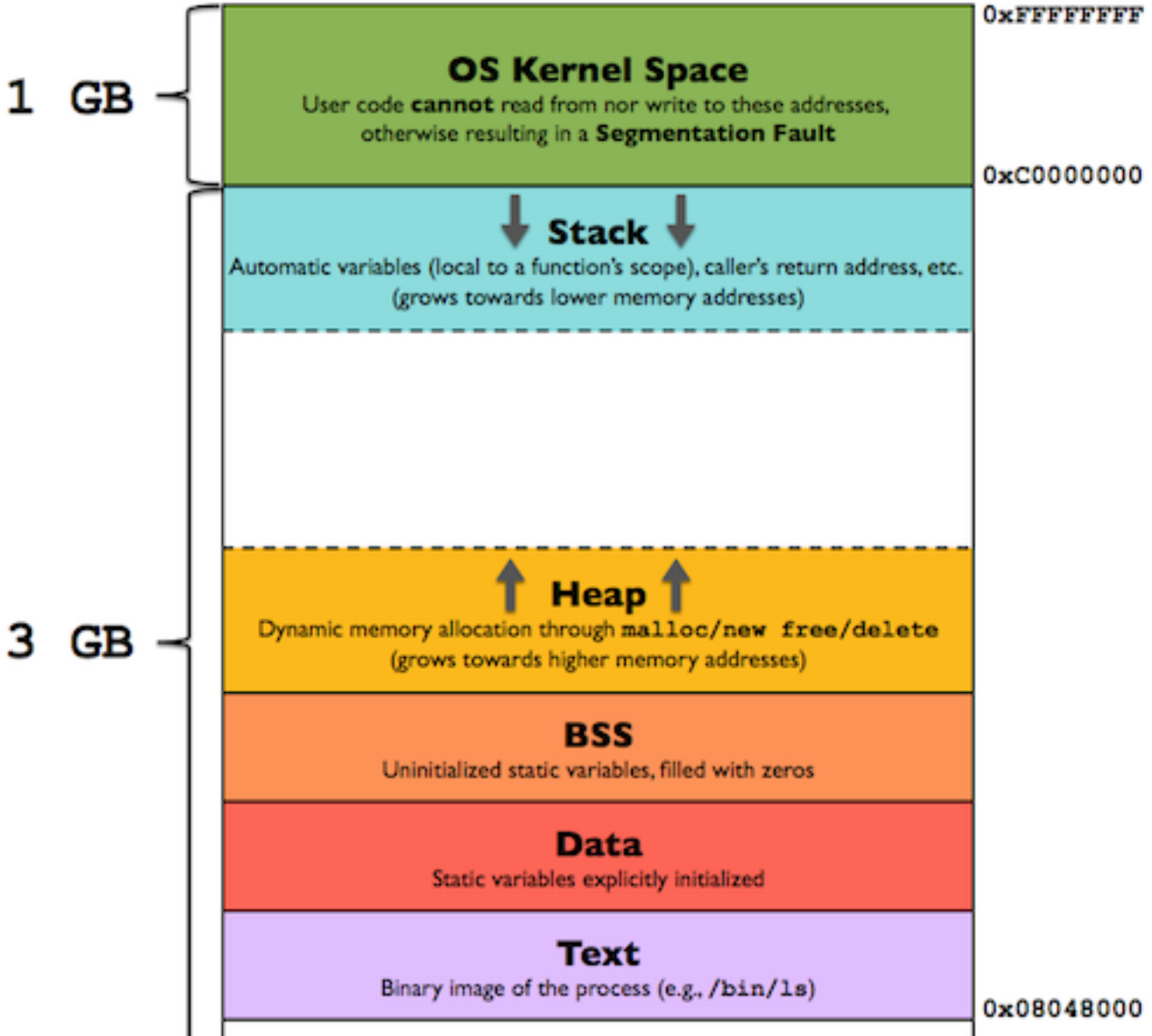

Bottom Half
Top Half
Interrupt
Soft IRQ Kontext
Interrupts auf aktueller line nicht disabled
Interrupt Kontext
Interrupts auf aktueller Line disabled

Bottom Halves

- Soft IRQs
- Tasklets

- Work Queues

Linux Memory Managment


1 GB
3 GB
0xFFFFFFFF
OS Kernel Space
User code cannot read from nor write to these addresses, otherwise resulting in a Segmentation Fault
0xC0000000
Stack
Automatic variables (local to a function's scope), caller's return address, etc.
(grows towards lower memory addresses)
Heap
Dynamic memory allocation through malloc/new free/delete
(grows towards higher memory addresses)
BSS
Uninitialized static variables, filled with zeros
Data
Static variables explicitly initialized
Text
Binary image of the process (e.g., /bin/ls)
0x08048000

Locking

Sleeping Locks
Mutex,
Semaphore

“Awake Locks”
Spinlocks

Trace-cmd und kernelshark Beispiel

drivers/char/rtc.c