Operating Systems

Auseinandergenommen

Andreas Galauner
SigInt 2010

Was ist ein Betriebssystem?

- DIN 44300:
  "Die Programme eines digitalen Rechensystems, die zusammen mit den Eigenschaften dieser Rechenanlage die Basis der möglichen Betriebsarten des digitalen Rechensystems bilden und die insbesondere die Abwicklung von Programmen steuern und überwachen."

Was ist ein Betriebssystem?

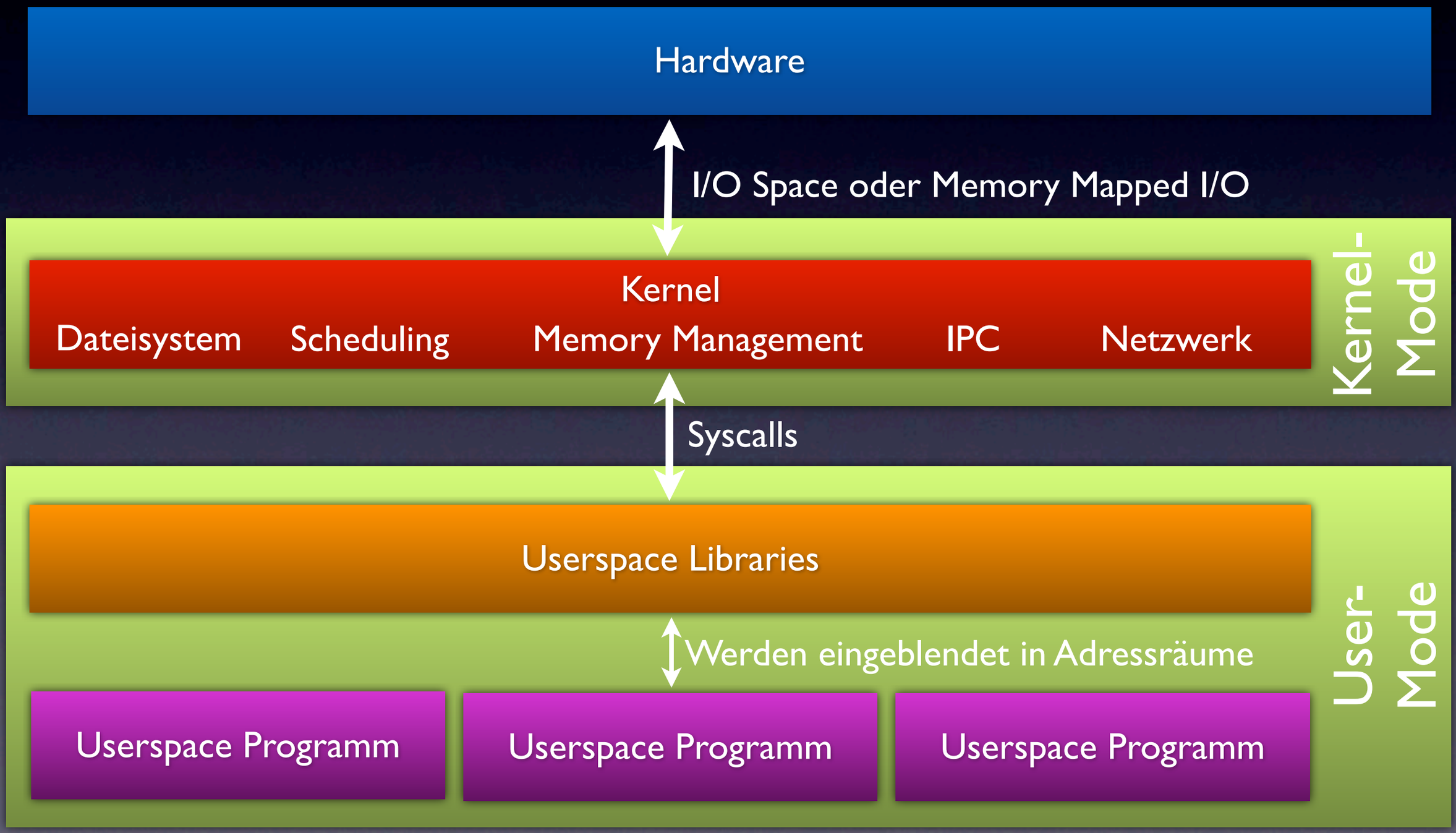
- Etwas geekiger:
  - Abstraktion der Hardware
  - Schaffung einheitlicher Schnittstellen
  - Gerechte Verteilung zur Verfügung stehender Resourcen unter den „Anwärtern“

Woraus besteht ein Betriebssystem?

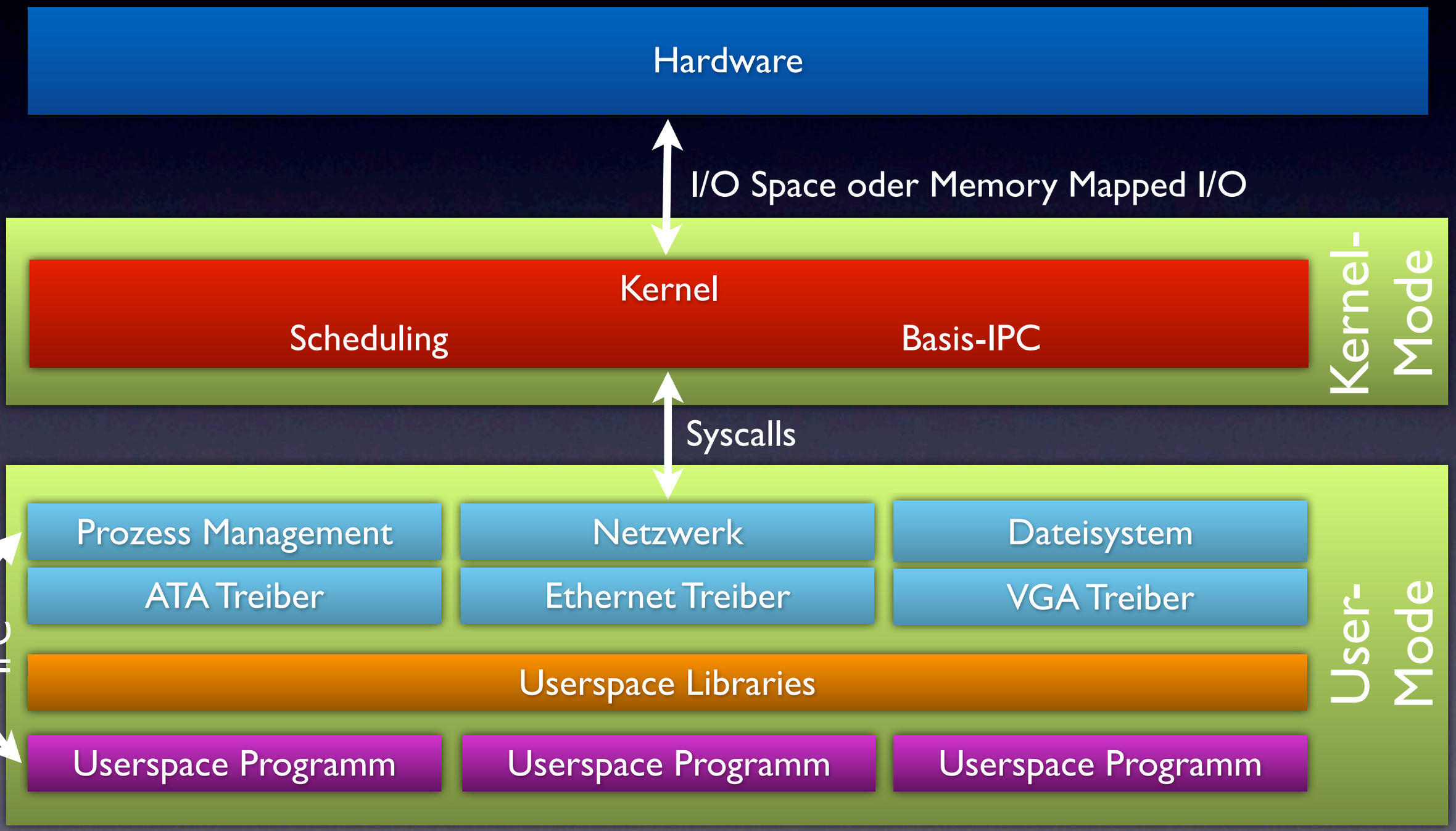
- Kernel, evtl. mit abgespaltenem HAL
- Userspace Libraries
- Userspace Services
- Userspace Applikationen

Architekturen

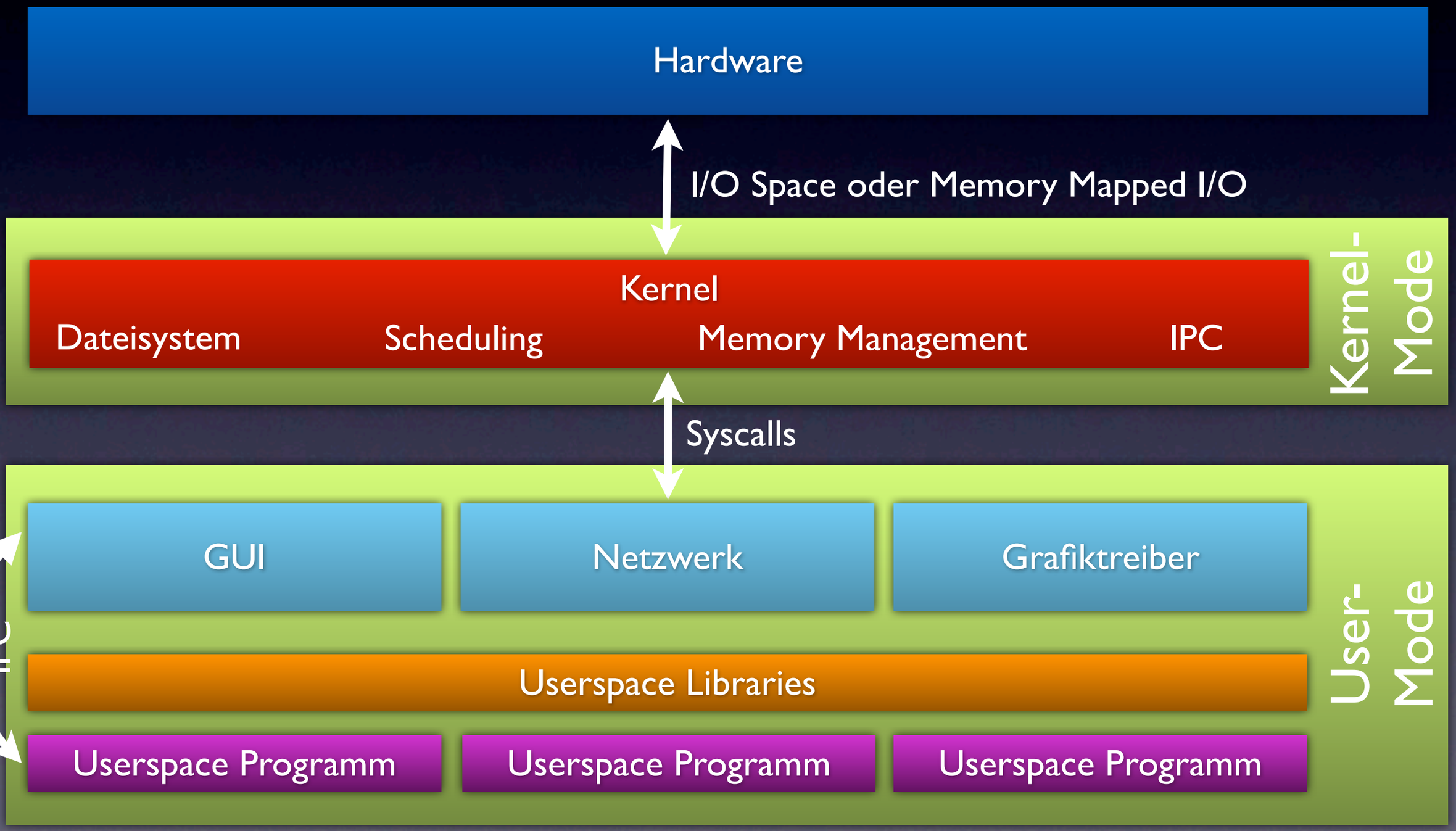
Monolithischer Kernel



Microkernel



Hybridkernel



Prozess- und Speichermanagement

Prozesse

- Eine Ansammlung von Code und Daten
- Ein Betriebssystem sollte Prozesse ausführen und verwalten können
- Sollten aus Sicherheitsgründen nicht ohne Kontrolle andere Prozesse beeinflussen können

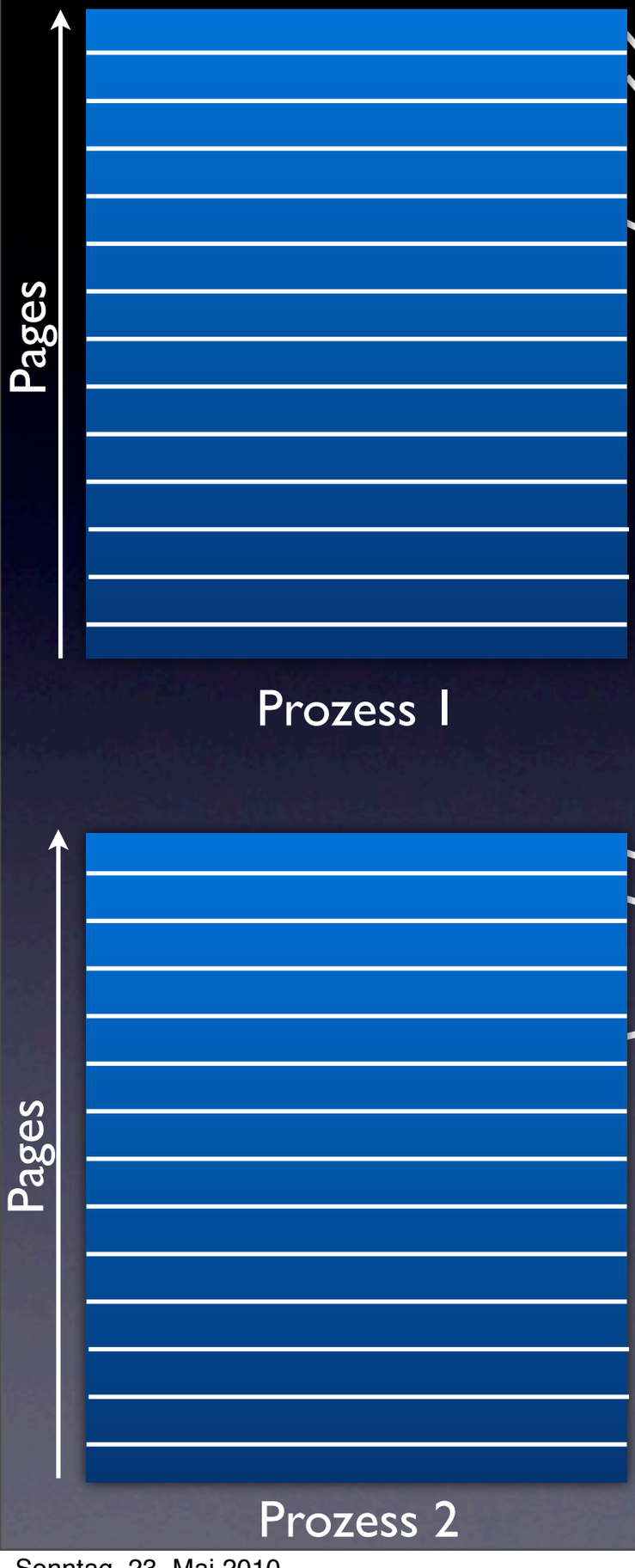
Virtueller Speicher

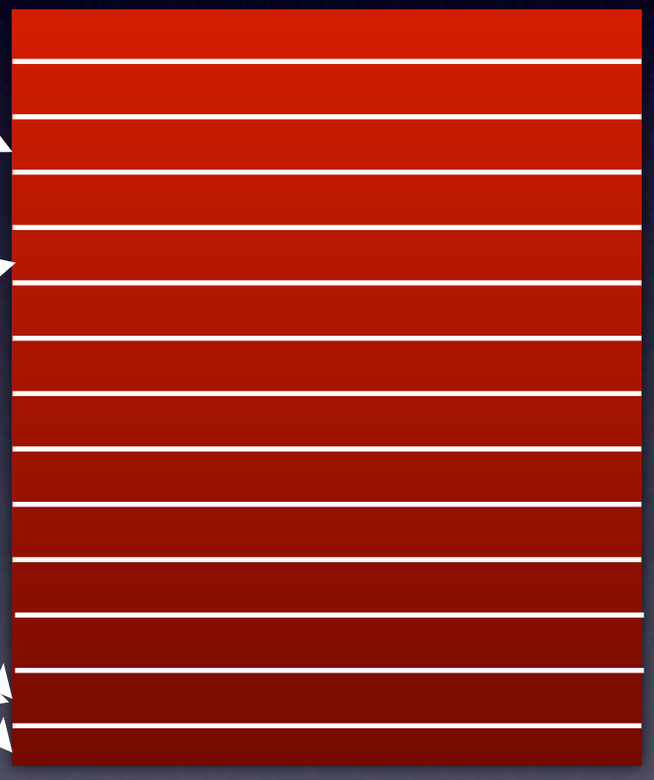
- Einer der Grundpfeiler zur Abschottung einzelner Prozesse
- Jeder Prozess erhält einen eigenen Adressraum
- Der Adressraum kann dann via Paging realer Speicher zugewiesen werden
- Virtueller Speicher ermöglicht setzen einiger Kontrollbits (Privilege Level, NX-Bit, Read/Write etc.)
- Bei Missachtung dieser Kontrollbits, wird eine Exception geworfen: Page Fault
- Adressumsetzung passiert transparent in der CPU durch die Memory Management Unit (MMU)
- Auch andere Inhalte im Speicher möglich, wie z.B. Dateien
- Optimierung des Speicherverbrauchs durch gleichzeitiges Mappen einzelner Frames in mehrere Adressräumen und evtl. Copy-On-Write
- Swapping

Paging

- Wir unterteilen den physikalischen und virtuellen Speicher in gleich große Teile
- Physikalische Teile nennen wir „Frames“
- Virtuelle Teile werden die „Pages“
- Jeder Teil ist bei x86 (IA-32) 4KB oder 4MB groß
- Bei 4KB haben wir also 4GB / 4KB = 1048576 Pages, die wir nun mit Frames befüllen können
- Ein Page Directory (Mapping von Pages zu Frames) ist also (bei 4B langen Pointern) 1048576 * 4B = 4MB groß → zu groß!

Paging

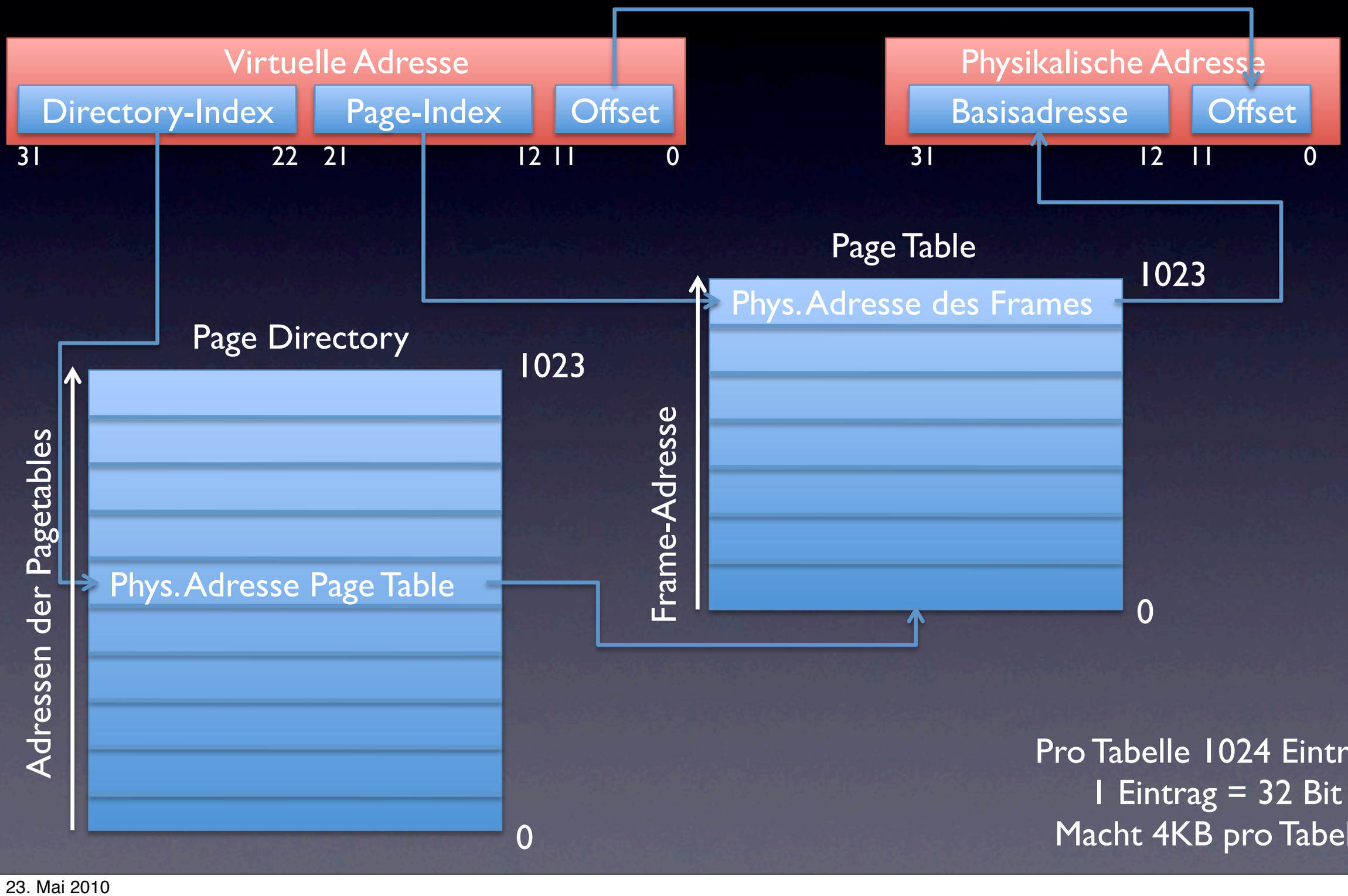

Pages
Prozess 1
Pages
Prozess 2


Frames
Physikalischer Speicher

Paging

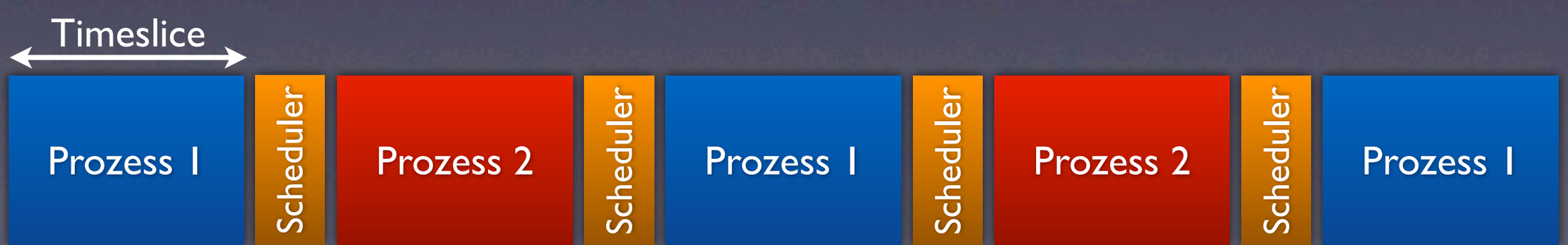
- Lösung für unser Größenproblem: 2-stufiges Paging
- Obere 10 Bit der virtuellen Adresse ergeben Index im Page Directory, der auf eine Page Table zeigt
- Folgende 10 Bit ergeben den Index der Page in der Page Table
- Letzte 12 Bit sind dann Offset im Frame selbst

Paging


Virtuelle Adresse
Directory-Index
Page-Index
Offset
31
22
21
12
11
0
Physikalische Adresse
Basisadresse
Offset
31
12
11
0
Page Table
1023
Phys. Adresse des Frames
0
Page Directory
1023
Phys. Adresse Page Table
0
Adressen der Pagetables
Frame-Adresse
Pro Tabelle 1024 Einträge
1 Eintrag = 32 Bit
Macht 4KB pro Tabelle

Scheduling

- Unterteilung der Zeit in Zeitschlitze
- Zuweisung einzelner Zeitschlitze an Prozesse = CPU-Zeit
- Zeitschlitz wird periodisch durch Timer Interrupt unterbrochen
- Auswahl des nächsten Prozesses kann anhand von Prioritäten, Wartezuständen etc. geschehen
- Scheduler übernimmt, je nach Verfahren, auch die unterste Schicht des IPC (z.B. Signal Handling unter UNIX)



Privilege Levels

- Einführung mehrerer Ebenen in Hardware
- Je niedriger die Ebene, umso mehr Rechte hat sie
- IA-32 hat 4 Ringe
- Instruktionen werden abhängig vom Ring, in dem ein Prozess läuft, ausgeführt oder blockiert
- Kernel darf alles
- Ring 0 ist Kernel, Ring 3 Userspace
- Userspace ist stark limitiert: Kein direkter Hardwarezugriff, etc.
- Betriebssysteme nutzen meist nur Ring 0 und 3 aus Gründen der Kompatiblität zu anderen Architekturen

Interrupts

- Können von Hardware genutzt werden, um der Software asynchrone Events mitzuteilen
- Können von Software genutzt werden, um Privilege Level zu ändern
- Im Kernelmodus deaktivierbar
- Exceptions sind ebenfalls Interrupts
- Beispiel: Alle $x ms ruft der Timer Interrupt den Scheduler auf
- Beispiel: General Protection Fault bei Aufruf einer privilegierten Instruktion in einem oberen Ring

Syscalls

- Wechsel von höheren Ringen in tiefere nicht ohne weiteres möglich, ist Sinn der Sache
- Ist aber nötig, um Kernelcode vom Usermode aus aufzurufen
- Wird über Syscalls geregelt
- Verschiedene Möglichkeiten: Trap, Software Interrupt, Sysenter/Sysexit
- Beispiel: open, mmap, fork, Berkley-Sockets
- Rootkits würden sich in die Syscalls klemmen und Ergebnisse „passend“ modifizieren

Real World Beispiel

Beispiel: Linux

- Was passiert, wenn wir ein Programm ausführen?
- Beispielcode:

```
#include <stdio.h>
#include <unistd.h>

int main(void)
{
        printf("%u\n", getpid());
        printf("Hello, SigInt!\n");
        for(;;);
        return 0;
}
```

- Kompiliert mit

gcc -o dynhello hello.c

und

gcc -static -o stathello hello.c

Linken

- Compiler erstellt Objectfiles aus jedem *.c File mit Binärcode und Listen von internen, exportierten und importierten Symbolen
- Linker durchsucht alle Listen, verknüpft gleiche Einträge und platziert das Symbol an eine Adresse im virtuellen Adressraum des Prozesses
- Orte für verschiedene Arten von Daten (Code, Readonly Daten, initialisierte und uninitialisierte Daten etc.) sind im Linkerscript der entsprechenden Plattform (OS) festgelegt
- Erstelltes „Gesamtwerk“ wird, wieder mit etlichen Metadaten, in weiteres File gepackt und kann ausgeführt werden
- Nirgendwo importierte, an anderer Stelle aber exportierte Symbole, werden weggeworfen und kommen im Zielbinary nicht mehr vor

Linken

- Unauflösbare Symbole sind blöd
- Zum Beispiel importiert ein Objectfile die Funktion xyz
- Aber kein anderes exportiert ein Symbol mit dem Namen xyz
- Resultiert in Linker-Error:

```
andy@alufolie ~ $ gcc -o dynhello hello.c
/tmp/cc8D2DPg.o: In function `main':
hello.c:(.text+0x33): undefined reference to `xyz'
collect2: ld returned 1 exit status
```

- Externe Libraries ausser der libc müssen in der Commandline extra angegeben werden

Linken

- Aber: Symbole können teilweise unaufgelöst bleiben
- Falls bestimmte Funktionen nicht im Zielbinary vorkommen sollen, können sie beim Laden des Programms importiert werden
- Nennt sich dynamisches Linken
- Linker ist bekannt, welche Funktionen aus einer Library kommen und vermerkt das in Tabellen im Zielbinary
- Zur Laufzeit muss dann ein Runtime-Linker den Adressraum des zu ladenden Prozesses „zusammenflicken“ und die fehlenden Referenzen ausfüllen
- Funktioniert begrenzt auch ohne Runtime-Linker: Linux Kernel-Module

ELF

readelf zeigen
Dateigrößen zwischen static und dynamic vergleichen

- Was Matroska für euren (unseren) p0rn ist, ist ELF für Programmcode und -daten → ein Containerformat
- Enthält globalen Header und jeweils einen Header für einzelne Sektionen im File
- Benötigte Sektionen: .text, .data/.rodata, .bss
- Sektionen haben verschiedene Flags: R/W, Executable
- Sektionen können auch Metadaten enthalten, wie Debugging-Infos

Laden eines Programms

- Mutter-/Vaterprozess ruft fork() auf und klont sich selbst
- Kindprozess macht dann ein execve und ersetzt somit seine Sektionen mit denen aus ELF-File
- Kernel muss also das ELF parsen
- Evtl. vorhergehender Interpreter wird ausgeführt
- Hier: Dynamischer Linker /lib/ld-linux.so.2

strace

strace zeigen

- Was genau macht ein Prozess?
- Welche Syscalls werden aufgerufen?
- strace spuckt zur Laufzeit aufgerufene Syscalls an den Kernel inkl. Argumente und Ergebnis aus

strace

- Statisch gelinkt:

```
andy@alufolie ~ $ strace ./stathello
execve("./stathello", ["./stathello"], [/* 45 vars */]) = 0
uname({sys="Linux", node="alufolie", ...}) = 0
brk(0)                                  = 0x80cb000
brk(0x80cbcd0)                          = 0x80cbcd0
set_thread_area({entry_number:-1 -> 6, base_addr:0x80cb830, limit:1048575,
seg_32bit:1, contents:0, read_exec_only:0, limit_in_pages:1, seg_not_present:0,
useable:1}) = 0
brk(0x80eccd0)                          = 0x80eccd0
brk(0x80ed000)                          = 0x80ed000
getpid()                                = 14288
fstat64(1, {st_mode=S_IFCHR|0620, st_rdev=makedev(136, 0), ...}) = 0
mmap2(NULL, 4096, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1,
0) = 0xb7839000
write(1, "14288\n", 614288
)                       = 6
write(1, "Hello, SigInt!\n", 15Hello, SigInt!
)       = 15
```

strace

- Dynamisch gelinkt:

```
andy@alufolie ~ $ strace ./dynhello
execve("./dynhello", ["./dynhello"], [/* 45 vars */]) = 0
brk(0)                                  = 0x804b000
access("/etc/ld.so.preload", R_OK)      = -1 ENOENT (No such file or directory)
open("/etc/ld.so.cache", O_RDONLY)      = 3
fstat64(3, {st_mode=S_IFREG|0644, st_size=67671, ...}) = 0
mmap2(NULL, 67671, PROT_READ, MAP_PRIVATE, 3, 0) = 0xb76e6000
close(3)                                = 0
open("/lib/libc.so.6", O_RDONLY)        = 3
read(3, "\177ELF\1\1\1\0\0\0\0\0\0\0\0\3\0\3\0\1\0\0\0300k\1\0004\0\0\0"..., 512) = 512
fstat64(3, {st_mode=S_IFREG|0755, st_size=1343772, ...}) = 0
mmap2(NULL, 4096, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) = 0xb76e5000
mmap2(NULL, 1349928, PROT_READ|PROT_EXEC, MAP_PRIVATE|MAP_DENYWRITE, 3, 0) = 0xb759b000
mmap2(0xb76df000, 12288, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x144) = 0xb76df000
mmap2(0xb76e2000, 10536, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_FIXED|MAP_ANONYMOUS, -1, 0) = 0xb76e2000
close(3)                                = 0
mmap2(NULL, 4096, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) = 0xb759a000
set_thread_area({entry_number:-1 -> 6, base_addr:0xb759a6c0, limit:1048575, seg_32bit:1, contents:0, read_exec_only:0, limit_in_pages:1,
seg_not_present:0, useable:1}) = 0
mprotect(0xb76df000, 8192, PROT_READ)   = 0
mprotect(0x8049000, 4096, PROT_READ)    = 0
mprotect(0xb7713000, 4096, PROT_READ)   = 0
munmap(0xb76e6000, 67671)               = 0
getpid()                                = 14296
fstat64(1, {st_mode=S_IFCHR|0620, st_rdev=makedev(136, 0), ...}) = 0
mmap2(NULL, 4096, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) = 0xb76f6000
write(1, "14296\n", 614296
)               = 6
write(1, "Hello, SigInt!\n", 15Hello, SigInt!
)       = 15
```

Virtueller Speiche

auf proc rumrutschen:
Sektionen
private-/shared-memory

mit dem GDB vdso
erklären

- Kernel exportiert im proc-FS viele, nützliche Sachen
- So auch /proc/<pid>/maps:

```
andy@alufolie /usr/include $ cat /proc/14297/maps
08048000-08049000 r-xp 00000000 08:03 1384688    /home/andy/dynhello
08049000-0804a000 r--p 00000000 08:03 1384688    /home/andy/dynhello
0804a000-0804b000 rw-p 00001000 08:03 1384688    /home/andy/dynhello
b76af000-b76b0000 rw-p 00000000 00:00 0
b76b0000-b77f4000 r-xp 00000000 08:03 1878203    /lib/libc-2.10.1.so
b77f4000-b77f6000 r--p 00144000 08:03 1878203    /lib/libc-2.10.1.so
b77f6000-b77f7000 rw-p 00146000 08:03 1878203    /lib/libc-2.10.1.so
b77f7000-b77fb000 rw-p 00000000 00:00 0
b780b000-b780c000 rw-p 00000000 00:00 0
b780c000-b7828000 r-xp 00000000 08:03 1878185    /lib/ld-2.10.1.so
b7828000-b7829000 r--p 0001c000 08:03 1878185    /lib/ld-2.10.1.so
b7829000-b782a000 rw-p 0001d000 08:03 1878185    /lib/ld-2.10.1.so
bff33000-bff48000 rw-p 00000000 00:00 0          [stack]
ffffe000-fffff000 r-xp 00000000 00:00 0          [vdso]
```

Fragen?
Antworten!

- Es gibt noch einen Workshop zum Thema OS-Development: x86 Bootstrapping im Workshopraum, Gebäude MP7, 23.05.2010 15:00 Uhr