THOMAS KRENN®

So funktioniert der Linux Storage Stack

Werner Fischer (Thomas-Krenn.AG)
Chemnitzer Linux-Tage 2025 – 22. und 23. März

Chemnitzer Linux-Tage
22. und 23. März 2025

Ein paar Fragen zum Einstieg …

Ein paar Fragen zum Einstieg …

1. Hast du schon mal **tune2fs -l** ausgeführt?

Ein paar Fragen zum Einstieg …

1. Hast du schon mal **tune2fs -l** ausgeführt?
2. Hast du **md-raid** konfiguriert?

Ein paar Fragen zum Einstieg …

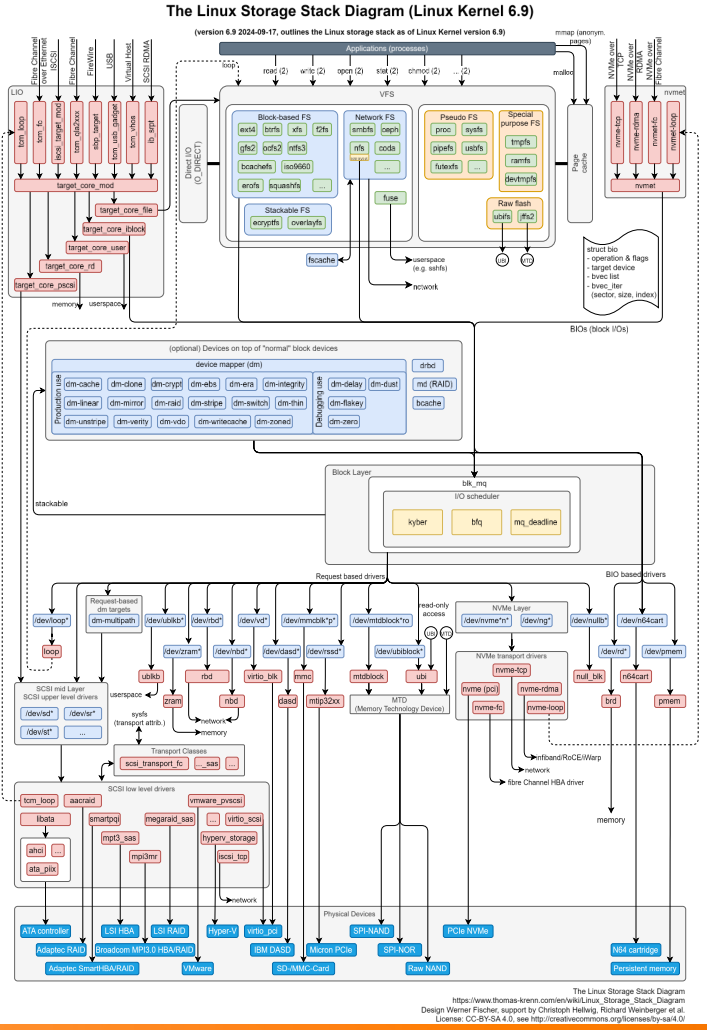
1. Hast du schon mal **tune2fs -l** ausgeführt?
2. Hast du **md-raid** konfiguriert?
3. Hast du **I/O-Scheduler** eingestellt?

Ein paar Fragen zum Einstieg …

1. Hast du schon mal **tune2fs -l** ausgeführt?
2. Hast du **md-raid** konfiguriert?
3. Hast du **I/O-Scheduler** eingestellt?
4. Hast du schon einmal **nvme-cli** benutzt?

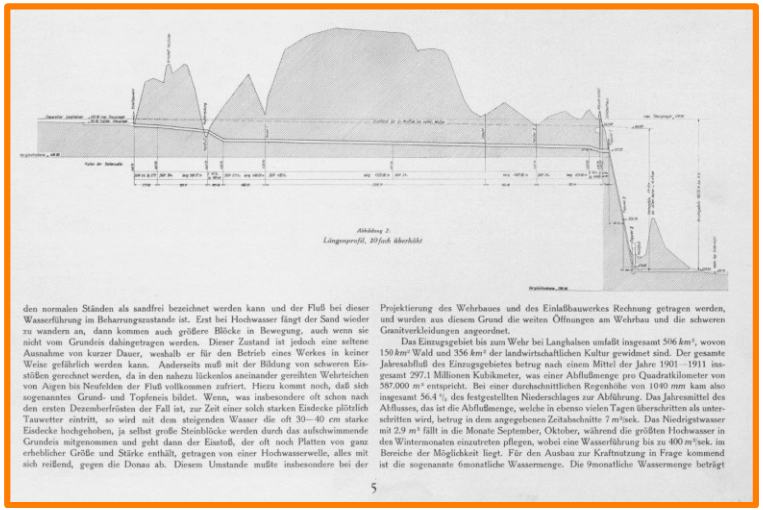
Agenda

- Wozu (secondary) Storage?
- Die Sicht des Nutzers
- VFS (Virtual FileSystem)
- Beispiel ext3/ext4
- Beispiel btrfs
- Block Layer: remapper
- Block Layer: Virtual Data Optimizer
- Block Layer: blk-mq & I/O scheduler
- Gerätetreiber
- The “big picture”



Wozu (secondary) Storage?

Pic: © Werner Fischer / CC-BY-SA-4.0

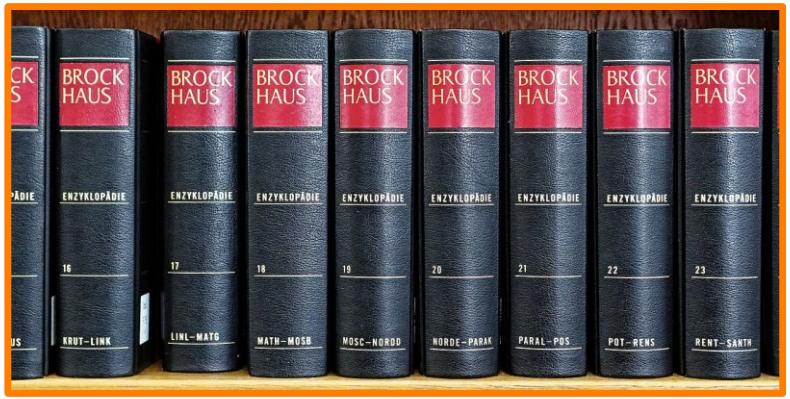


den normalen Ständen als sandfrei bezeichnet werden kann und der Fluß bei dieser Wasserführung im Beharrungszustande ist. Erst bei Hochwasser fängt der Sand wieder zu wandern an, dann kommen auch größere Blöcke in Bewegung, auch wenn sie nicht vom Grundeis dahingetragen werden. Dieser Zustand ist jedoch eine seltene Ausnahme von kurzer Dauer, weshalb er für den Betrieb eines Werkes in keiner Weise gefährlich werden kann. Andererseits muß mit der Bildung von schweren Eisstößen gerechnet werden, da in den nahezu lückenlos aneinander gereihten Wehrteichen von Aigen bis Neufelden der Fluß vollkommen zufriert. Hiezu kommt noch, daß sich sogenanntes Grund- und Topfeneis bildet. Wenn, was insbesondere oft schon nach den ersten Dezemberfrösten der Fall ist, zur Zeit einer solch starken Eisdecke plötzlich Tauwetter eintritt, so wird mit dem steigenden Wasser die oft 30—40 *cm* starke Eisdecke hochgehoben, ja selbst große Steinblöcke werden durch das aufschwimmende Grundeis mitgenommen und geht dann der Eisstoß, der oft noch Platten von ganz erheblicher Größe und Stärke enthält, getragen von einer Hochwasserwelle, alles mit sich reißend, gegen die Donau ab. Diesem Umstande mußte insbesondere bei der Projektierung des Wehrbaues und des Einlaßbauwerkes Rechnung getragen werden, und wurden aus diesem Grund die weiten Öffnungen am Wehrbau und die schweren Granitverkleidungen angeordnet.

Das Einzugsgebiet bis zum Wehr bei Langhalsen umfaßt insgesamt 506 km^2, wovon 150 km^2 Wald und 356 km^2 der landwirtschaftlichen Kultur gewidmet sind. Der gesamte Jahresabfluß des Einzugsgebietes betrug nach einem Mittel der Jahre 1901—1911 insgesamt 297.1 Millionen Kubikmeter, was einer Abflußmenge pro Quadratkilometer von 587.000 m^3 entspricht. Bei einer durchschnittlichen Regenhöhe von 1040 *mm* kam also insgesamt 56.4 % des festgestellten Niederschlages zur Abführung. Das Jahresmittel des Abflusses, das ist die Abflußmenge, welche in ebenso vielen Tagen überschritten als unterschritten wird, betrug in dem angegebenen Zeitabschnitte 7 m^3/sek. Das Niedrigstwasser mit 2.9 m^3 fällt in die Monate September, Oktober, während die größten Hochwasser in den Wintermonaten einzutreten pflegen, wobei eine Wasserführung bis zu 400 m^3/sek. im Bereiche der Möglichkeit liegt. Für den Ausbau zur Kraftnutzung in Frage kommend ist die sogenannte 6monatliche Wassermenge. Die 9monatliche Wassermenge beträgt

5

Pic: © OÖ Landesbibliothek / CC-BY-SA-3.0

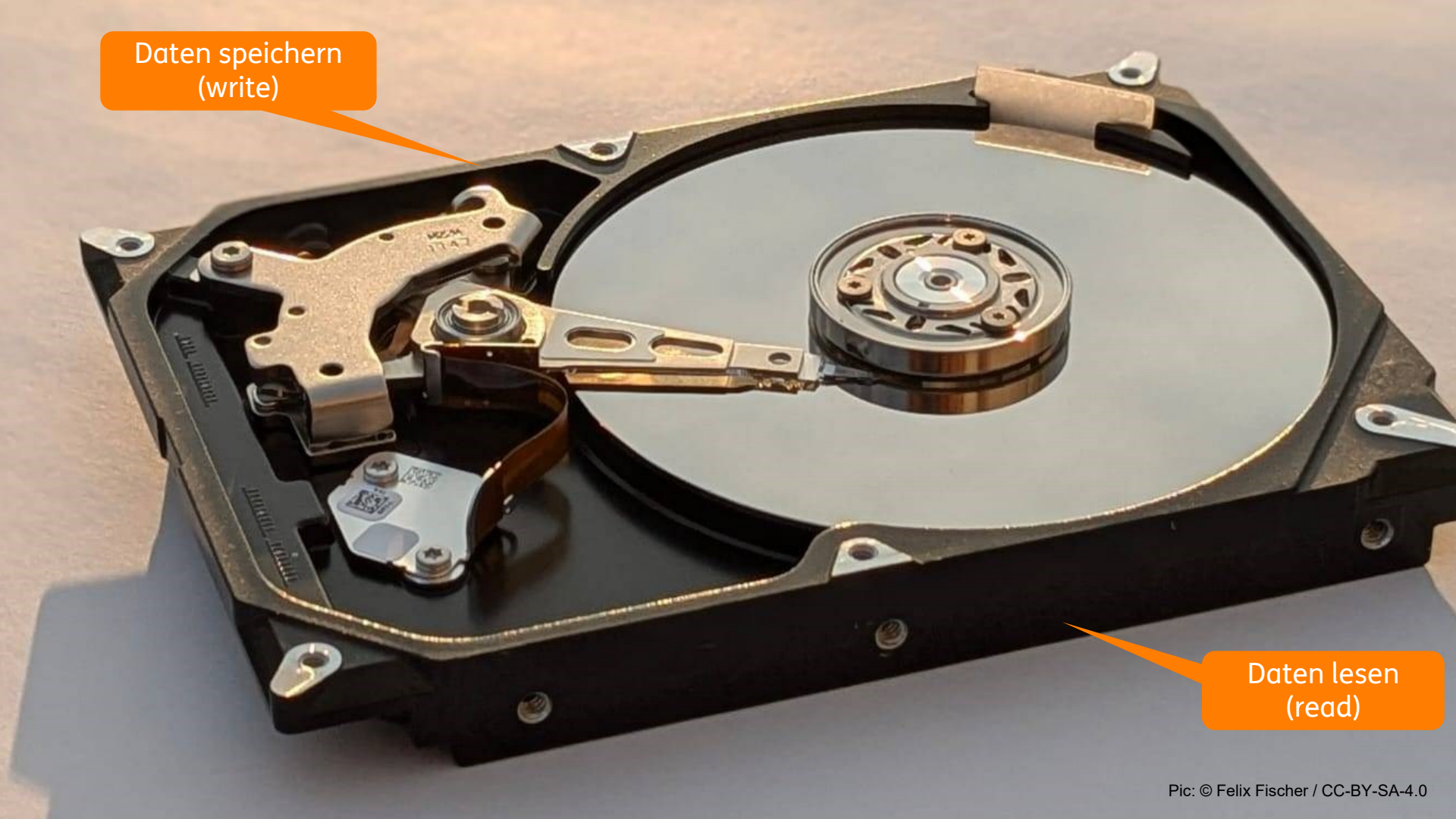


Pic: © Burkhard Mücke / CC-BY-SA-4.0

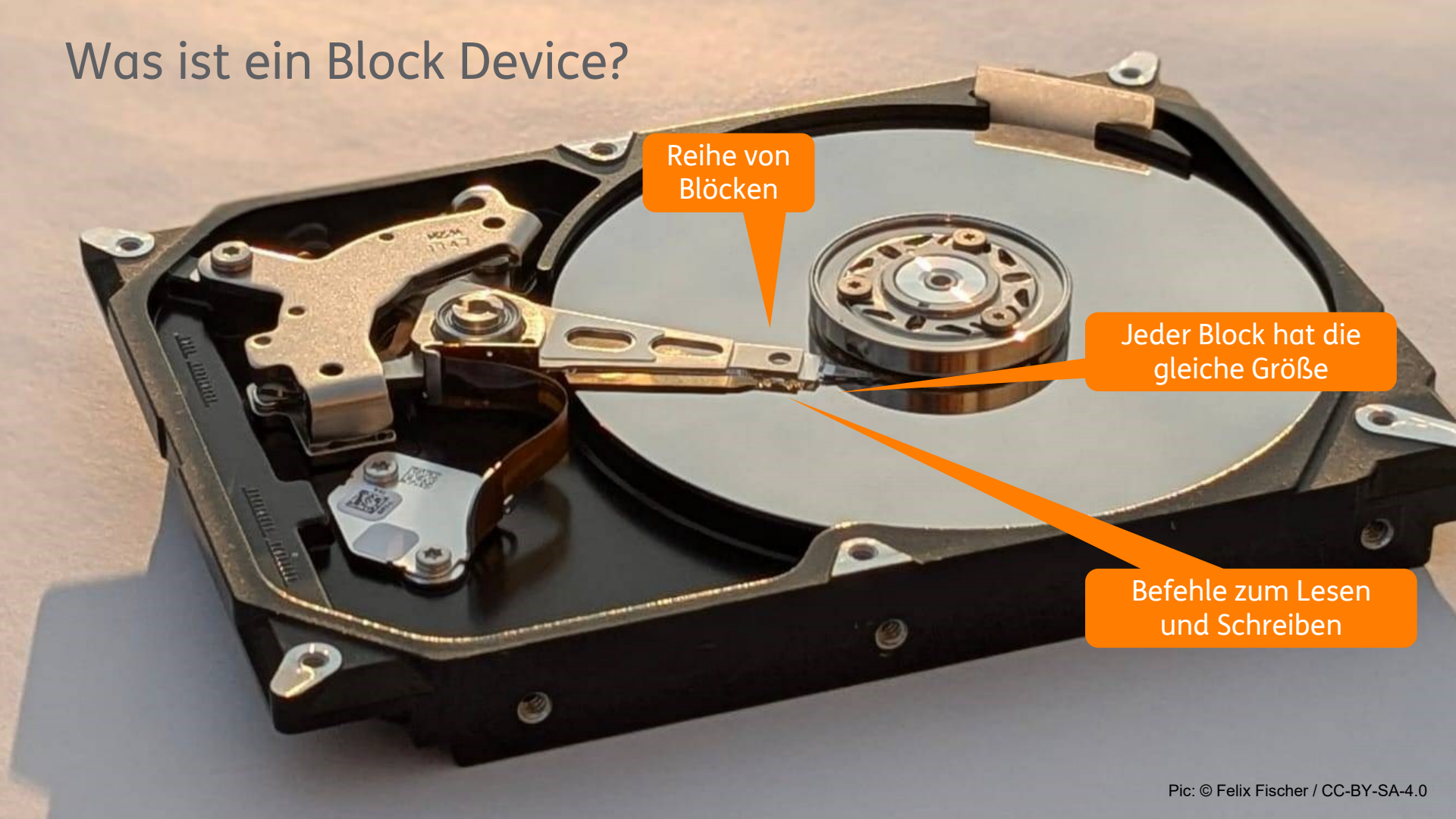
Chemnitzer Linux-Tage 22. und 23. März 2025

THOMAS KRENN

Die Sicht des Nutzers

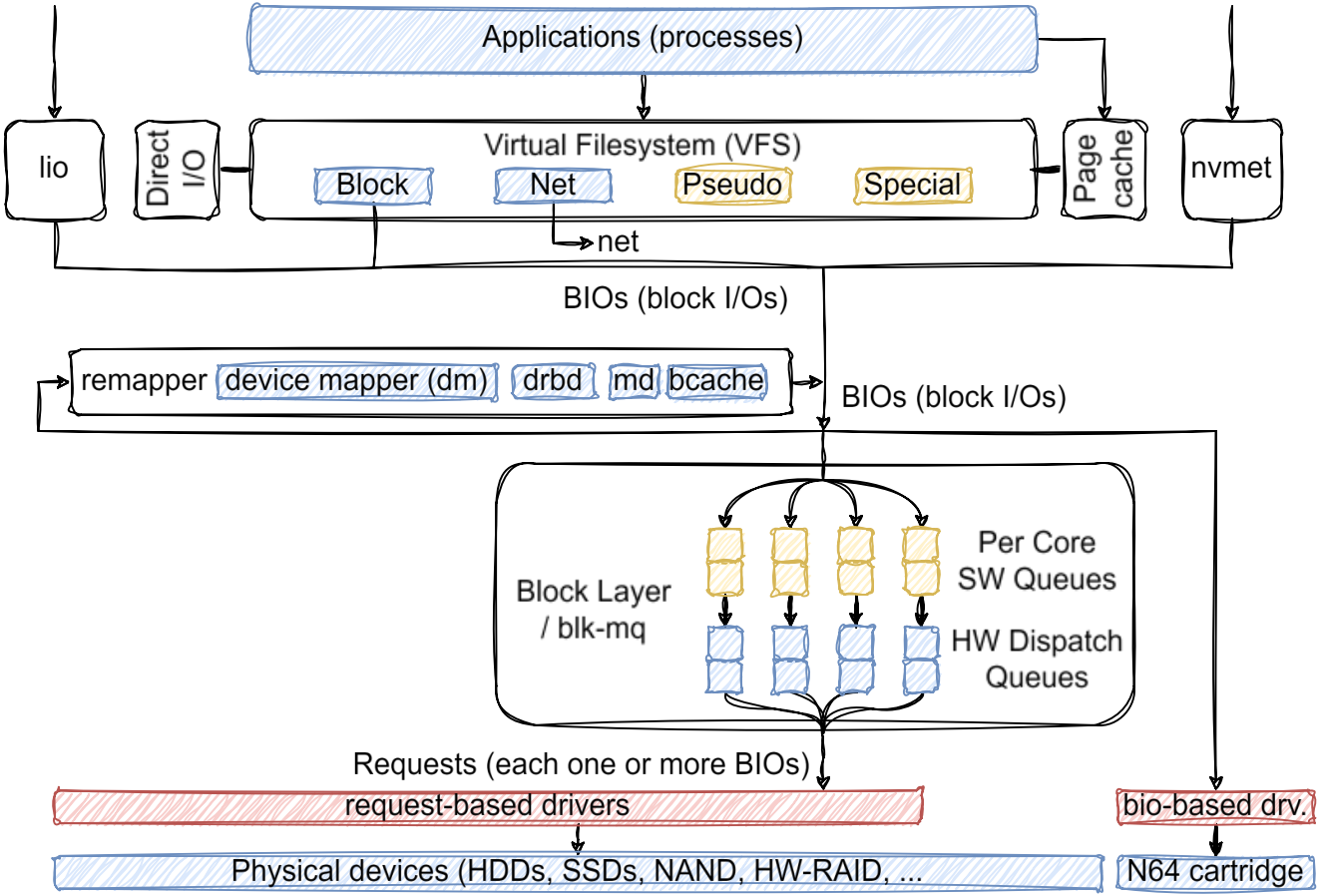


Was ist ein Block Device?



Synchroner versus asynchronem I/O

- Synchroner Kernel I/O
  - Thread startet I/O-Operation
  - Geht in den Wartezustand über, bis die E/A-Anforderung abgeschlossen ist
  - Beispiele: read(2), write(2), …
- Asynchroner Kernel I/O
  - Thread sendet I/O-Anfrage an den Kernel
  - Fortlaufende Bearbeitung eines anderen Auftrags, bis der Kernel signalisiert, dass die E/A-Anforderung abgeschlossen ist
  - Beispiele: aio_read, aio_write, aio, io_uring
- Asynchronous User-space I/O
  - Storage Performance Development Kit (SPDK)
  - Tools und Bibliotheken zum Schreiben von Hochleistungsspeicheranwendungen im Benutzermodus
  - Die Grundlage von SPDK ist ein asynchroner, „lockless“, „polled-mode“ NVMe-Treiber.
- Weitere Details: Improved Storage Performance Using the New Linux Kernel I/O Interface (John Kariuki & Vishal Verma, SDC 2019)

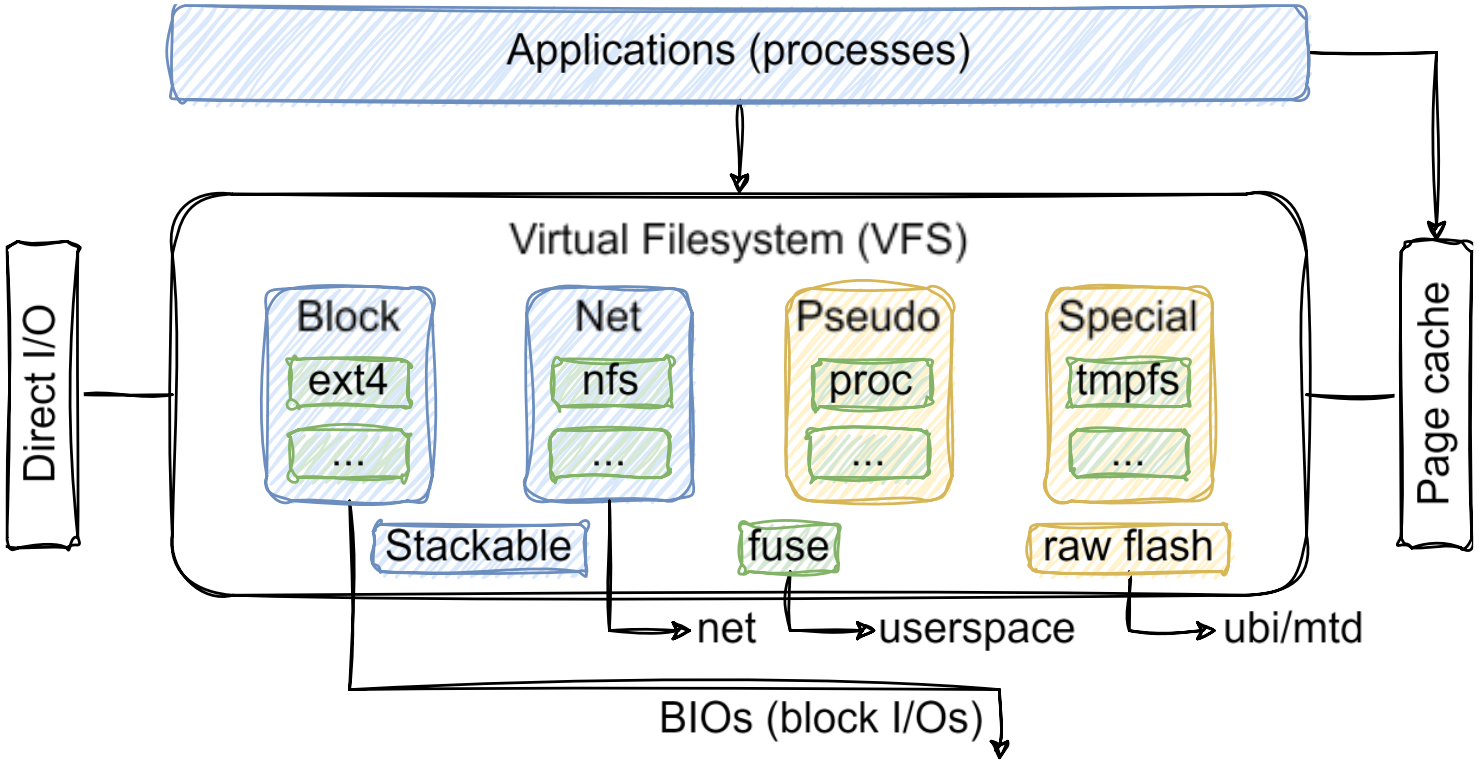

Applications (processes)
lio
Direct I/O
Virtual Filesystem (VFS)
Block
Net
Pseudo
Special
Page cache
nvmet
net
BIOs (block I/Os)
remapper
device mapper (dm)
drbd
md
bcache
BIOs (block I/Os)
Block Layer / blk-mq
Per Core SW Queues
HW Dispatch Queues
Requests (each one or more BIOs)
request-based drivers
bio-based drv.
Physical devices (HDDs, SSDs, NAND, HW-RAID, ...
N64 cartridge

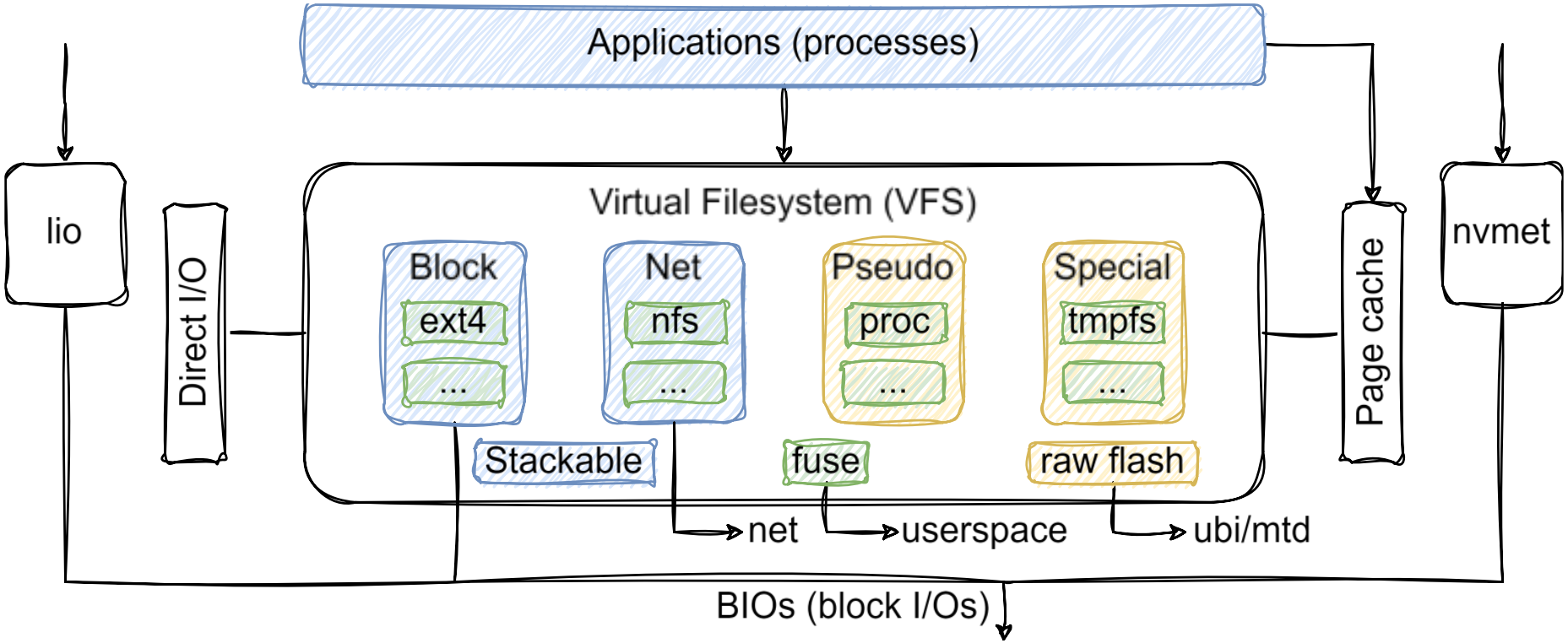
VFS (Virtual FileSystem)

VFS

- Eingeführt mit System V ~Mitte der 80er Jahre
- Bietet eine generische Schnittstelle für Anwendungen
- Operationen wie
  - read(2)
  - write(2)
  - open(2)
  - statx(2)
  - chmod(2), …

  funktionieren immer auf die gleiche Weise, egal welches Dateisystem verwendet wird.


Applications (processes)
Virtual Filesystem (VFS)
Direct I/O
Page cache
Block
ext4
...
Net
nfs
...
Pseudo
proc
...
Special
tmpfs
...
Stackable
fuse
raw flash
net
userspace
ubi/mtd
BIOs (block I/Os)

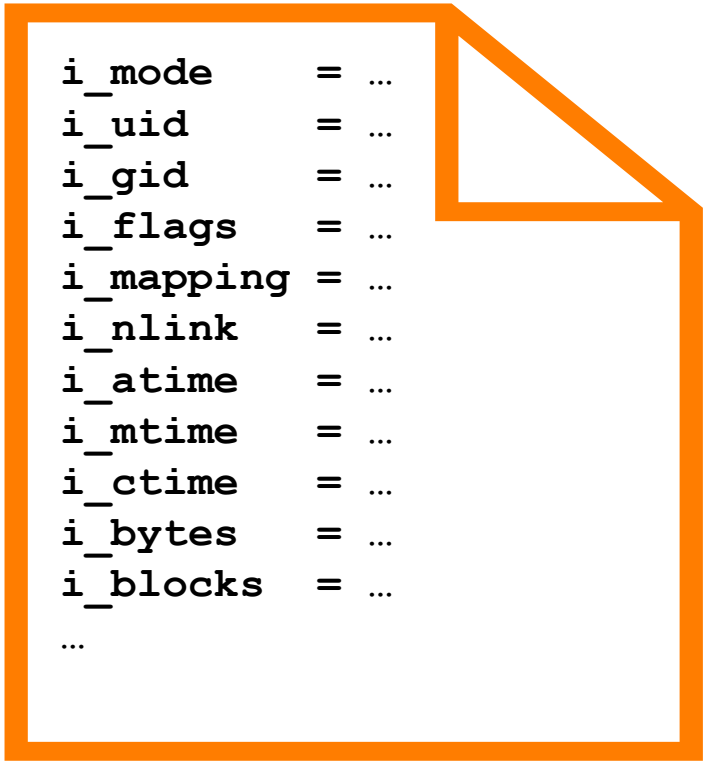

Applications (processes)
lio
Direct I/O
Virtual Filesystem (VFS)
Block
ext4
...
Net
nfs
...
Pseudo
proc
...
Special
tmpfs
...
Page cache
nvmet
Stackable
fuse
raw flash
net
userspace
ubi/mtd
BIOs (block I/Os)

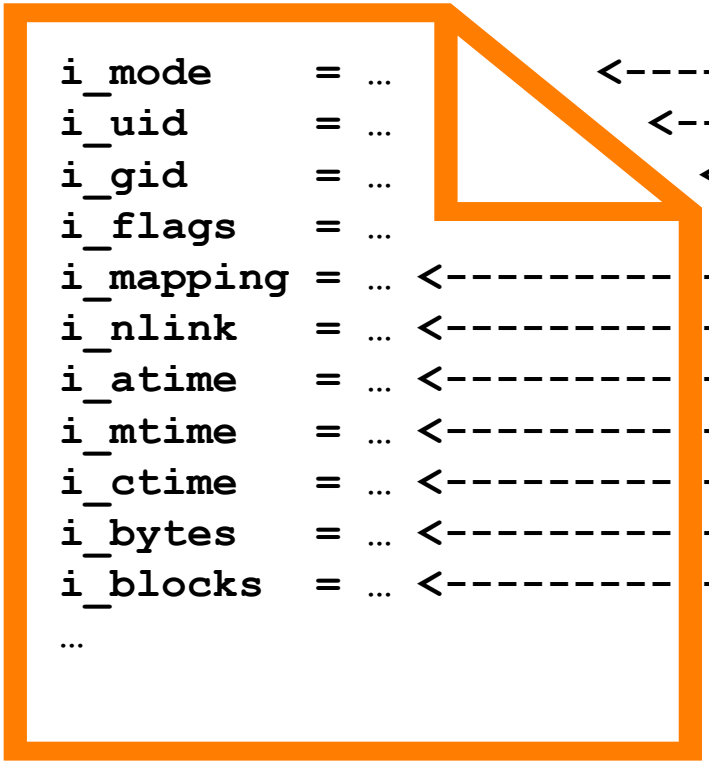
VFS Datenstrukturen / Objekte

- Inodes
- Directory entries
- File objects
- Superblocks

Inodes

- Index node
- Enthält Metadaten, welche die eigentlichen Daten _beschreiben_
- Enthält keine “echten” Daten




i_mode = …
i_uid = …
i_gid = …
i_flags = …
i_mapping = …
i_nlink = …
i_atime = …
i_mtime = …
i_ctime = …
i_bytes = …
i_blocks = …
…

<------- owner/group/others, eg 0777=rwxrwxrwx
<----- user
<--- group
<- flags
<--------- ---- mapping
<--------- ---- number of hard links to this file
<--------- ---- last time file was open()-ed
<--------- ---- last time file was modified write()-d
<--------- ---- last time inode was changed (not creation!)
<--------- ---- file size, beginning to end
<--------- ---- number of used blocks

```
werner@x390:~/test$ echo "example line 1" > examplefile.txt
werner@x390:~/test$ stat examplefile.txt
  File: examplefile.txt
  Size: 15              Blocks: 8          IO Block: 4096   regular file
Device: 254,1   Inode: 97422132    Links: 1
Access: (0644/-rw-r--r--)  Uid: ( 1000/  werner)   Gid: ( 1000/  werner)
Access: 2024-09-10 16:46:31.220485190 +0200
Modify: 2024-09-10 16:46:31.220485190 +0200
Change: 2024-09-10 16:46:31.220485190 +0200
 Birth: 2024-09-10 16:46:31.220485190 +0200
```

```
werner@x390:~/test$ date && cat examplefile.txt
Di 10 Sep 2024 16:47:10 CEST
example line 1
werner@x390:~/test$ stat examplefile.txt
  File: examplefile.txt
  Size: 15              Blocks: 8          IO Block: 4096   regular file
Device: 254,1   Inode: 97422132    Links: 1
Access: (0644/-rw-r--r--)  Uid: ( 1000/  werner)   Gid: ( 1000/  werner)
Access: 2024-09-10 16:47:10.212575439 +0200
Modify: 2024-09-10 16:46:31.220485190 +0200
Change: 2024-09-10 16:46:31.220485190 +0200
 Birth: 2024-09-10 16:46:31.220485190 +0200
```

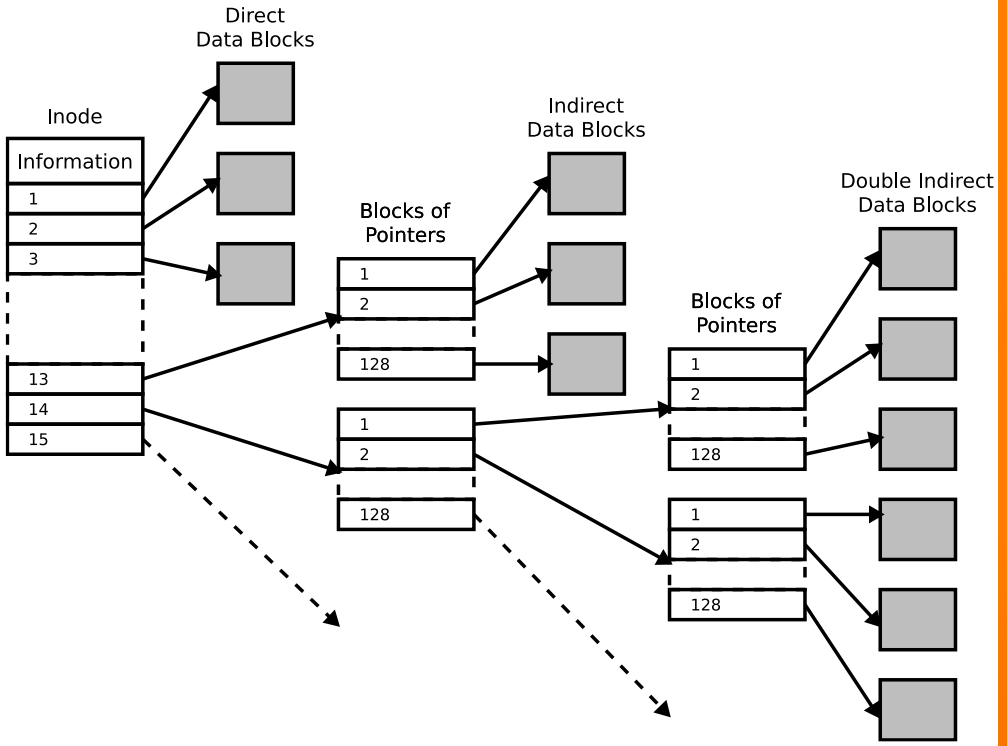
```
werner@x390:~/test$ date && echo "example line 2" >> examplefile.txt
Di 10 Sep 2024 16:48:08 CEST
werner@x390:~/test$ stat examplefile.txt
  File: examplefile.txt
  Size: 30              Blocks: 8          IO Block: 4096   regular file
Device: 254,1   Inode: 97422132    Links: 1
Access: (0644/-rw-r--r--)  Uid: ( 1000/  werner)   Gid: ( 1000/  werner)
Access: 2024-09-10 16:47:10.212575439 +0200
Modify: 2024-09-10 16:48:08.600709043 +0200
Change: 2024-09-10 16:48:08.600709043 +0200
 Birth: 2024-09-10 16:46:31.220485190 +0200
```

```
werner@x390:~/test$ date && chmod g+w examplefile.txt
Di 10 Sep 2024 16:49:11 CEST
werner@x390:~/test$ stat examplefile.txt
  File: examplefile.txt
  Size: 30              Blocks: 8          IO Block: 4096   regular file
Device: 254,1   Inode: 97422132    Links: 1
Access: (0664/-rw-rw-r--)  Uid: ( 1000/  werner)   Gid: ( 1000/  werner)
Access: 2024-09-10 16:47:10.212575439 +0200
Modify: 2024-09-10 16:48:08.600709043 +0200
Change: 2024-09-10 16:49:11.904851911 +0200
 Birth: 2024-09-10 16:46:31.220485190 +0200
```

Beispiel: Ext3 / Ext4

Ext3

- Block-basiert
  - 12 direkte Zeiger zu Datenblöcken
  - 1 einfach indirekter Zeiger (der auf einen Block von direkten Zeigern zeigt)
  - 1 zweifach indirekter Zeiger
  - 1 dreifach indirekter Zeiger
  - (Weitere Details: Ext3 for large filesystems, lwn.net, 2006)
- Journal



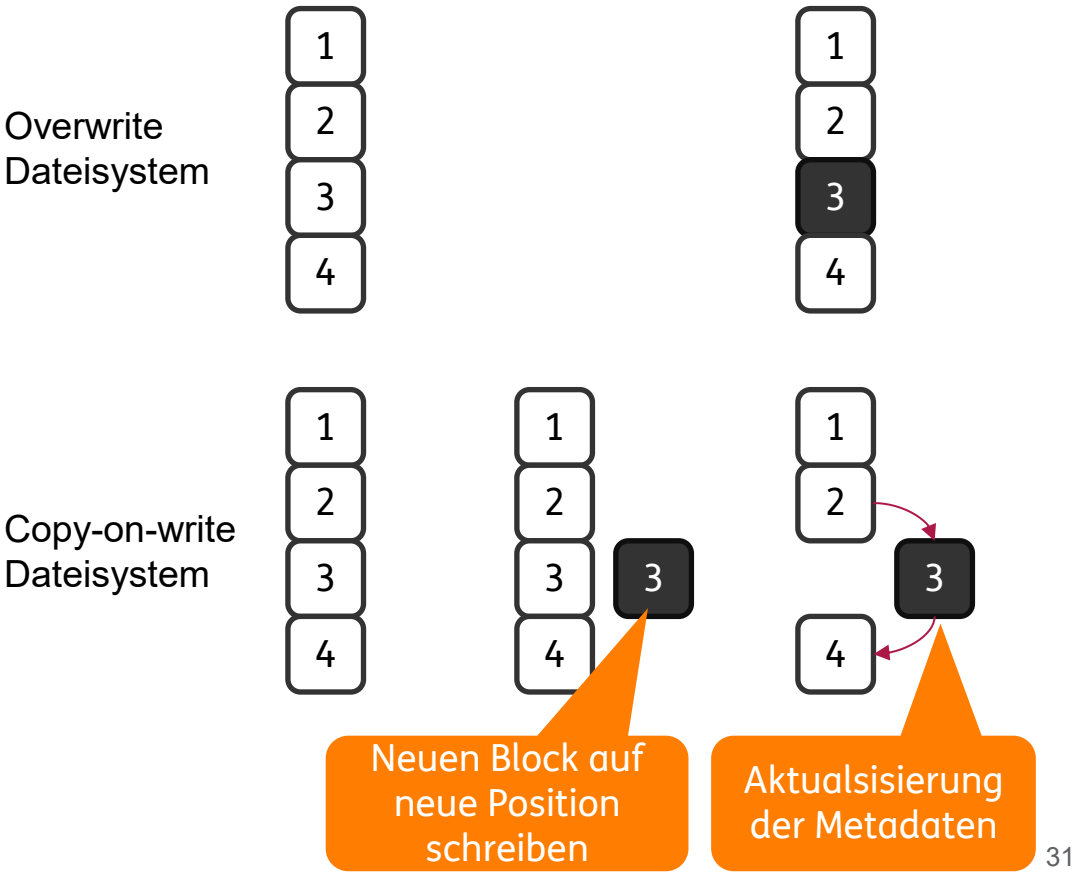
Ext4

- Extent-basierend
  - Extents adressieren keine einzelnen Blöcke, sondern bilden einen (möglichst großen) Bereich einer Datei auf einen Bereich zusammenhängender Blöcke auf der Festplatte ab.
  - Statt vieler einzelner Blocknummern werden nur noch drei Werte benötigt:
    - Start und
    - die Größe des Bereichs in der Datei (jeweils in Dateisystemblöcken) und
    - die Nummer des ersten Datenblocks
- Preallocation
- Delayed allocation
- Journal
- (Weitere Details Das Linux-Dateisystem Ext4, heise.de, 2009)

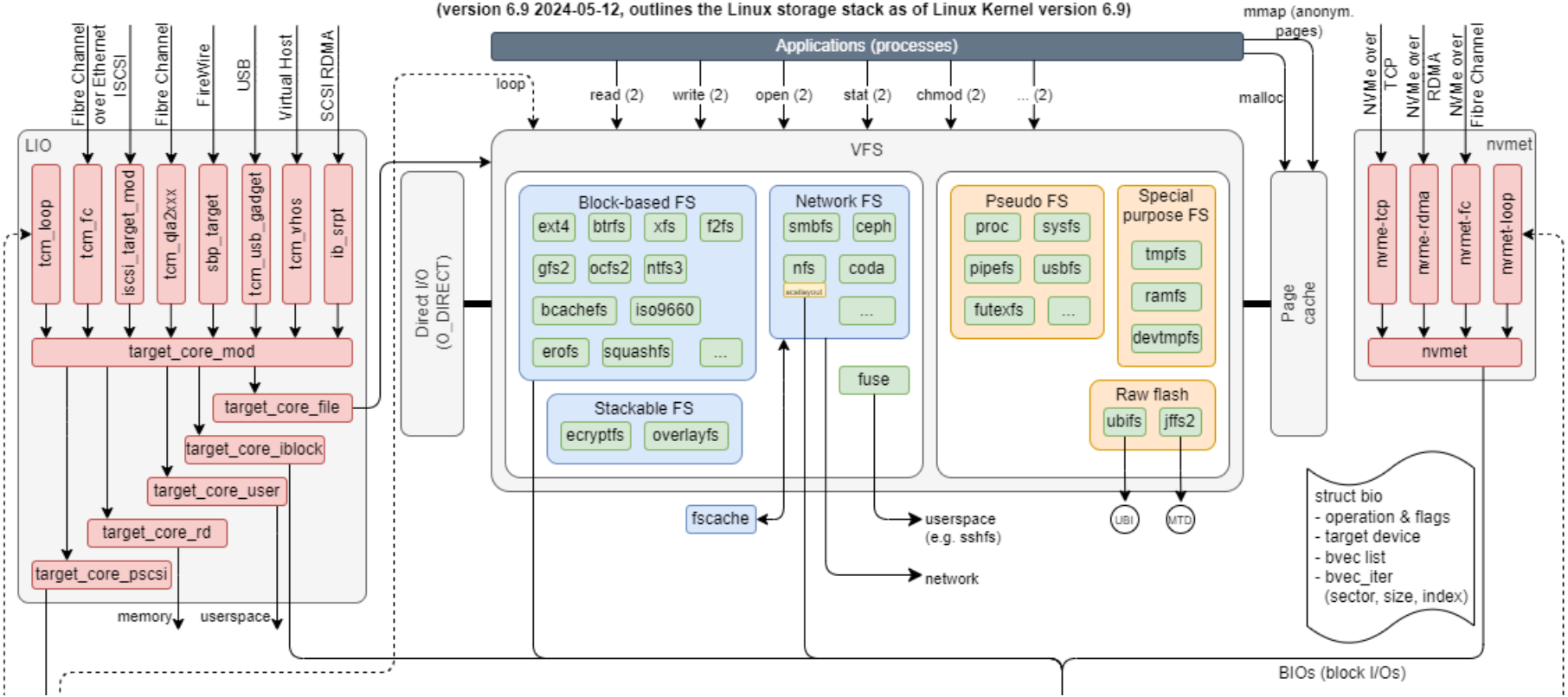
Beispiel: btrfs

btrfs

- Copy-On-Write
- Komprimierung
- Subvolumes
- Integriertes RAID
- Resize



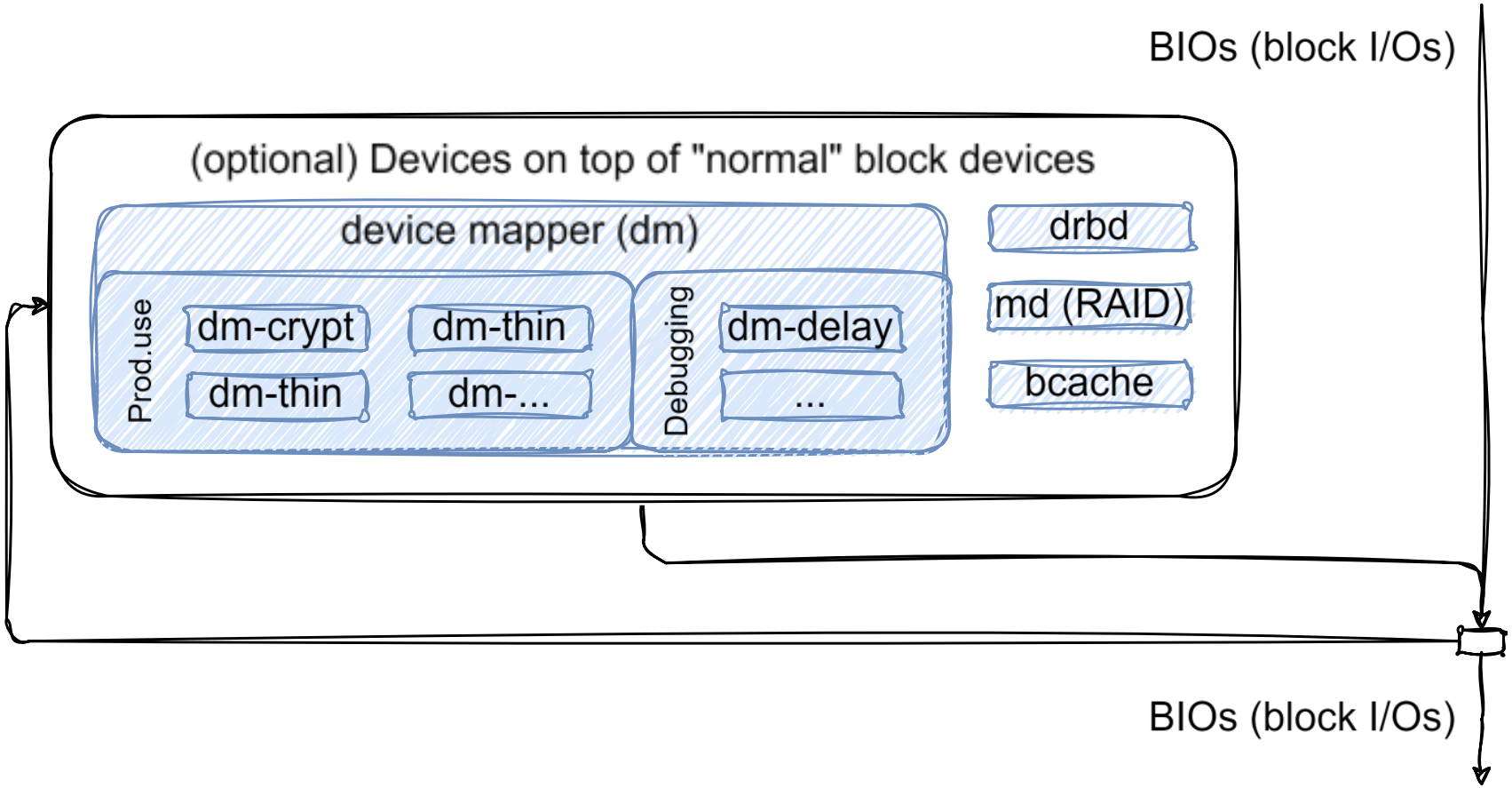
The Linux Storage Stack Diagram (Linux Kernel 6.9)



Block Layer: remapper

Stacked Block Devices (DM, MD, DRBD, …)

- Device Mapper
  - Device Mapper wird am häufigsten via LVM verwendet
  - Device Mapper Targets für Produktivbetrieb: dm-cache, dm-clone, dm-crypt, …, dm-vdo (virtual data optimizer), …, dm-zoned
  - Targets für Debugging-Zwecke: dm-delay, dm-dust, dm-flakey, dm-zero
- MD (RAID)
- DRBD
- bcache


BIOs (block I/Os)
(optional) Devices on top of "normal" block devices
device mapper (dm)
Prod.use
dm-crypt
dm-thin
dm-thin
dm-...
Debugging
dm-delay
...
drbd
md (RAID)
bcache
BIOs (block I/Os)

Stacked Block Devices (DM, MD, DRBD, …)

- Bei gestapelten Blockgeräten („stacked block devices“) gibt es kein BLK-MQ und kein I/O Scheduling (Elevators)!

```
werner@x390:~$ lsblk -s /dev/dm-0
NAME        MAJ:MIN RM  SIZE RO TYPE  MOUNTPOINTS
lvmlmde     254:0    0  1,9T  0 crypt
└─nvme0n1p3 259:3    0  1,9T  0 part
  └─nvme0n1 259:0    0  1,9T  0 disk
werner@x390:~$ ls /sys/block/dm-0/
alignment_offset   dm                 holders     removable   trace
bdi                events             inflight    ro          uevent
capability         events_async       integrity   size
dev                events_poll_msecs  power       slaves
discard_alignment  ext_range          queue       stat
diskseq            hidden             range       subsyste
werner@x390:~$ ls /sys/block/dm-0/mq/
ls: cannot access '/sys/block/dm-0/mq/': No such file or directory
werner@x390:~$ cat /sys/block/dm-0/queue/scheduler
none
```

verschlüsselte LVM volume group

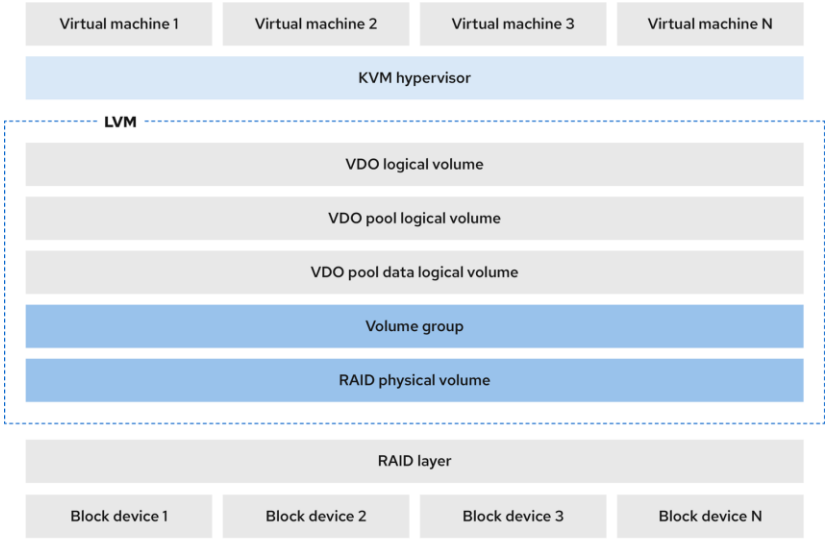
kein mq (multi queue) Unterverzeichnis

kein I/O Scheduler zur Auswahl...

Block Layer:
Virtual Data Optimizer (VDO)

Virtual Data Optimizer (VDO)

- VDO bietet auf Block-Ebene:
  - Deduplizierung
  - Komprimierung
  - Thin-Provisioning
- Ursprünglich proprietär von Permabit entwickelt, 2017 von Red Hat gekauft, seit Kernel 6.9 nun mainline
- Einsatz z.B. in KVM-Umgebungen (Bildquelle: Red Hat)
  - Weitere Infos: VDO (Thomas-Krenn Wiki)



```
tk@lmde:~$ sudo lvcreate --type vdo --name vdo-test --size 40G --virtualsize 100G vg-sata
    The VDO volume can address 36.00 GB in 18 data slabs, each 2.00 GB.
    It can grow to address at most 16.00 TB of physical storage in 8192 slabs.
    If a larger maximum size might be needed, use bigger slabs.
  Logical volume "vdo-test" created.
tk@lmde:~$ lsblk
NAME                          MAJ:MIN RM   SIZE RO TYPE MOUNTPOINTS
sda                             8:0    0 223,6G  0 disk
└─sda1                          8:1    0 223,6G  0 part
  └─vg--sata-vpool0_vdata     254:0    0    40G  0 lvm
    └─vg--sata-vpool0-vpool   254:1    0   100G  0 lvm
      └─vg--sata-vdo--test    254:2    0   100G  0 lvm
nvme0n1                       259:0    0 111,8G  0 disk
├─nvme0n1p1                   259:1    0   286M  0 part /boot/efi
├─nvme0n1p2                   259:2    0   8,2G  0 part [SWAP]
└─nvme0n1p3                   259:3    0 103,3G  0 part /
```

```
tk@lmde:~$ sudo mkfs.ext4 -E nodiscard /dev/vg-sata/vdo-test
[...]
tk@lmde:~$ sudo mount /dev/mapper/vg--sata-vdo--test /mnt
tk@lmde:~$ sudo dd if=/dev/urandom of=/mnt/testfile-1.bin bs=1M count=1024
1024+0 records in
1024+0 records out
1073741824 bytes (1,1 GB, 1,0 GiB) copied, 2,03686 s, 527 MB/s
tk@lmde:~$ sudo cp /mnt/testfile-1.bin /mnt/testfile-2.bin
[…]
tk@lmde:~$ sudo cp /mnt/testfile-1.bin /mnt/testfile-[…].bin
tk@lmde:~$ sync
tk@lmde:~$ sudo vdostats
Device                   1k-blocks      Used Available Use% Space saving%
vg--sata-vpool0-vpool     41943040   5279324  36663716  13%            85%
```

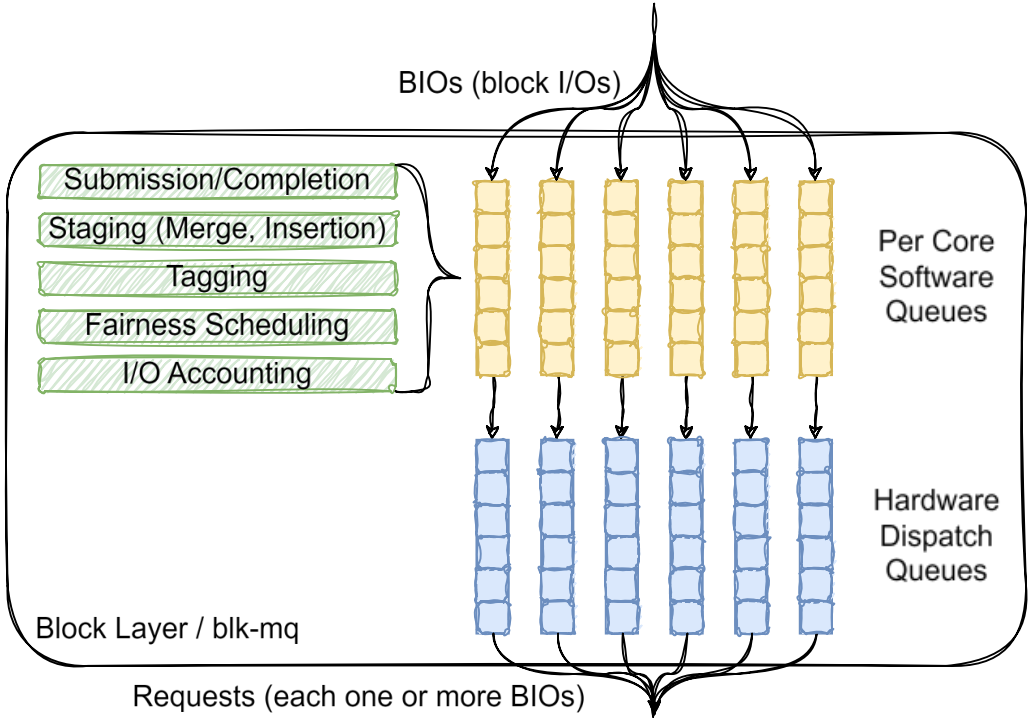
Block Layer:
BLK-MQ & I/O Scheduler

Chemnitzer Linux-Tage
22. und 23. März 2025

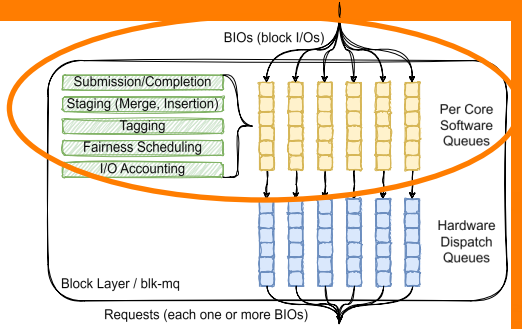
THOMAS KRENN®

BLK-MQ Architektur

- Per-CPU Software Queues
  - CPU-lokale Listen
  - Kein Locking
  - Keine Cache Lines von anderen CPUs
- Darunter: Hardware Queues
  - Optimum:
    N(hardware queues) = N(CPUs)

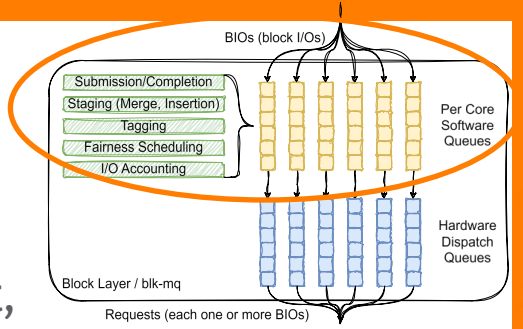


BLK-MQ: SW Staging Queues



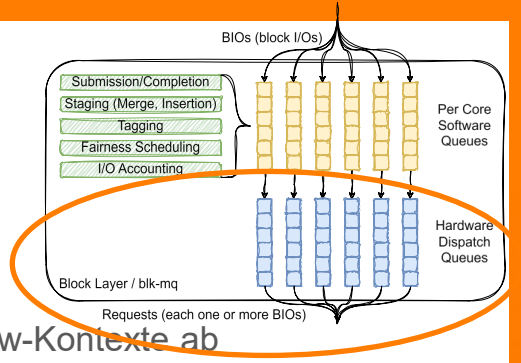
- Ein „Request“ besteht aus einem oder mehreren BIOs
- Jede Queue hat ihren eigenen Lock
- Anzahl der Warteschlangen wird pro CPU oder pro Knoten definiert
- Anforderungen können zusammengeführt werden („plugging“)
- Anforderungen können neu geordnet werden (I/O-Scheduler)

BLK-MQ: SW Queues – I/O Scheduler



- Mehrere Scheduler, von denen jeder einer Heuristik folgt, um die I/O-Leistung zu verbessern:
  - none (technisch kein Scheduler)
  - mq_deadline
  - bfq
  - kyber
- „pluggable“ - kann während der Laufzeit über sysfs ausgewählt werden
- Das Scheduling erfolgt nur zwischen Anfragen in derselben Warteschlange

BLK-MQ: HW Dispatch Queues

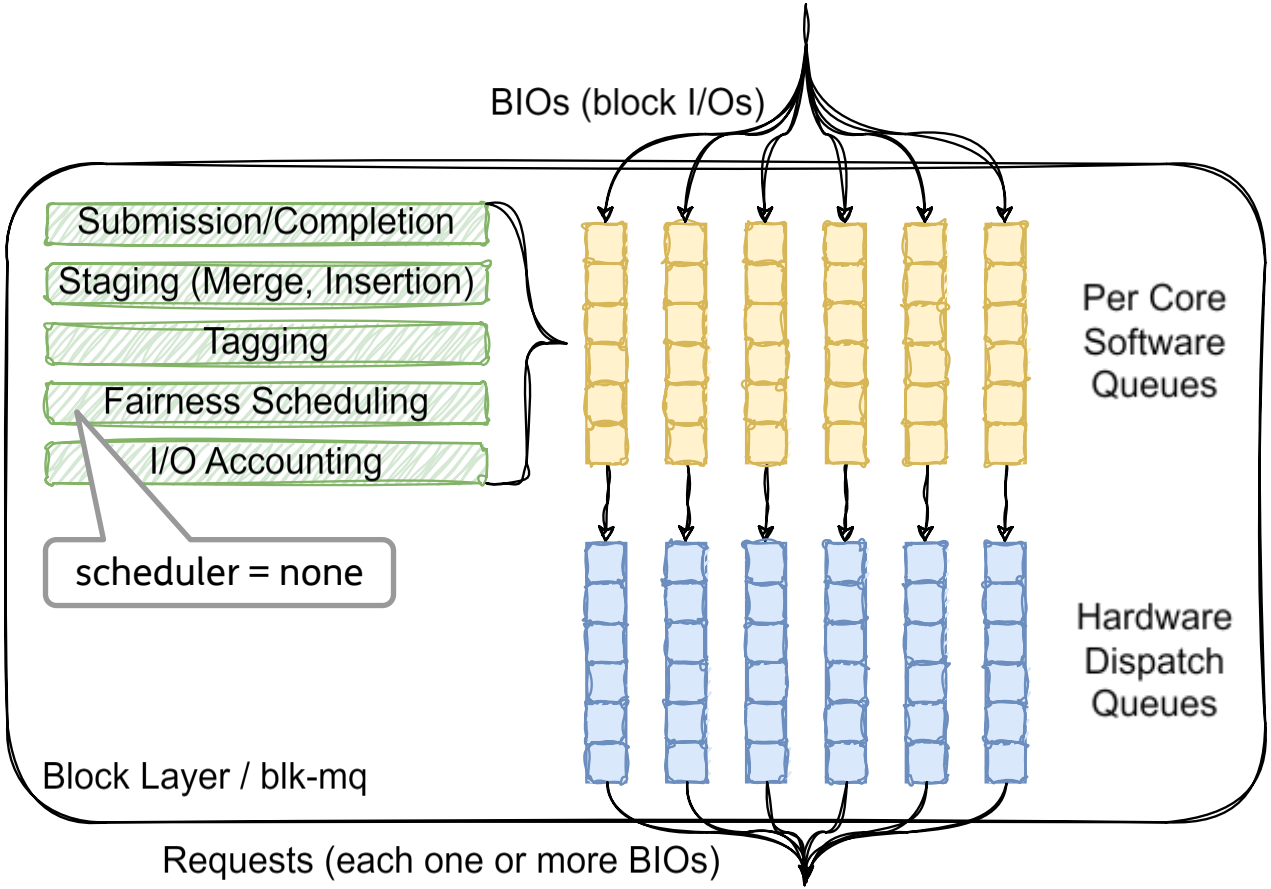


- Anzahl an HW Queues:
  - hängt von der Anzahl der von der Hardware/dem Treiber unterstützten hw-Kontexte ab
  - Treiber können zwischen 1 und 2.048 hw-Warteschlangen unterstützen (MSI-X-Limit)
  - ist nicht größer als die Anzahl der Kerne
  - SATA- und SAS-Geräte unterstützen eine einzige hw-Warteschlange
- Hardware-Warteschlangen sind unabhängig voneinander
  - jede Software-Warteschlange kann in ihre Hardware-Warteschlange befüllen, ohne dass eine globale Reihenfolge eingehalten werden muss (innerhalb des Block Layers wird kein Ordering unterstützt)
  - keine Locks zwischen Hardware-Warteschlangen -> höhere Performance
  - HW kann eine oder mehrere Queues implementieren, die direkt auf NUMA-Knoten / CPU-Kerne abgebildet werden
    - Dies bietet einen schnellen IO-Pfad von Anwendungen zur Hardware ohne Zugriff auf den entfernten Speicher/Caches eines anderen NUMA-Knotens.

6 Cores, no HT

Completion Queues Allocated (NCQA): 16

Submission Queues Allocated (NSQA): 16



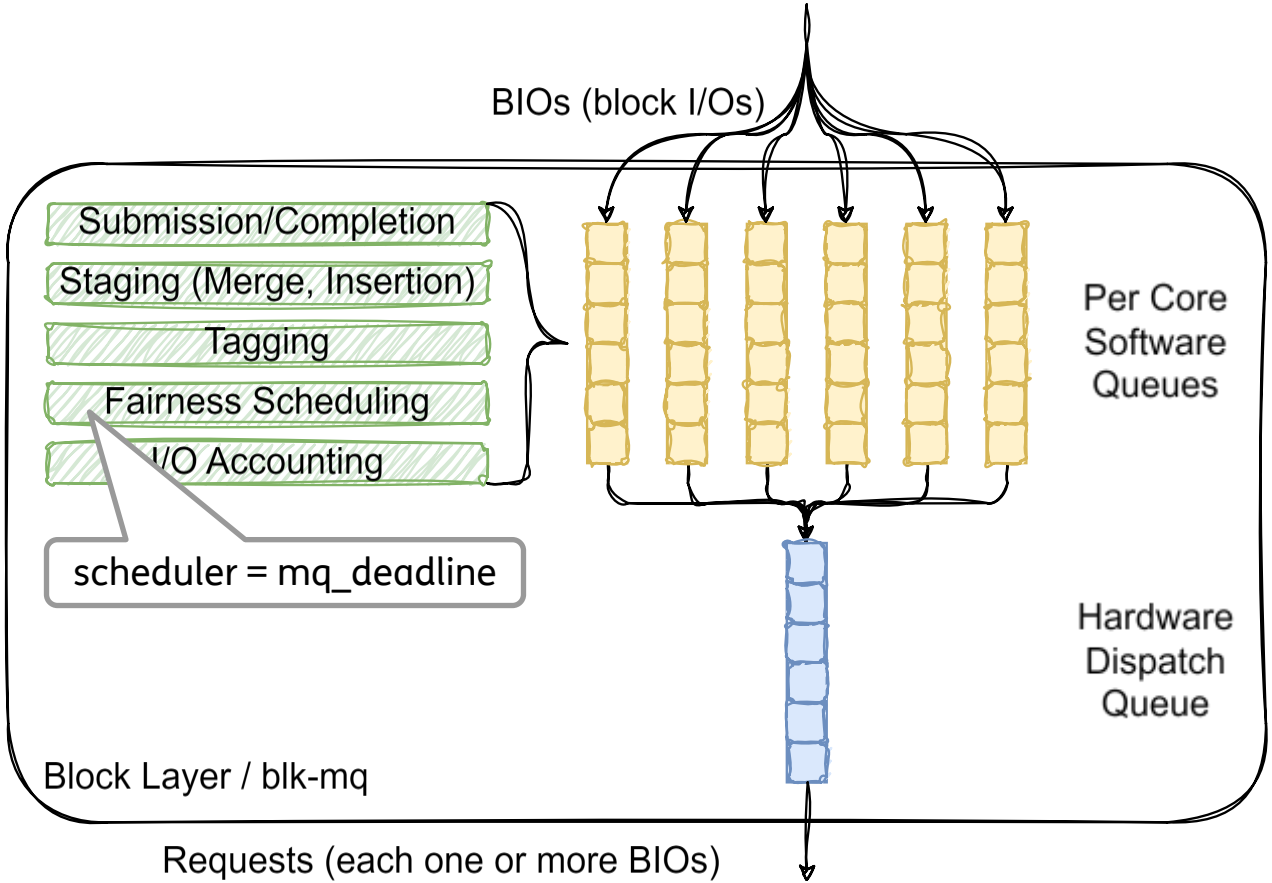
```
werner@pc:~$ lscpu
CPU(s):                  6
  Model name:            Intel(R) Core(TM) i5-8400 CPU @ 2.80GHz
werner@pc:~$ ls /sys/block/nvme0n1/mq/
0  1  2  3  4  5
werner@pc:~$ cat /sys/block/nvme0n1/mq/0/cpu_list
0
werner@pc:~$ cat /sys/block/nvme0n1/queue/scheduler
[none] mq-deadline
werner@pc:~$ sudo nvme get-feature /dev/nvme0n1 -f 7 -H
get-feature:0x07 (Number of Queues), Current value:0x000f000f
Number of IO Completion Queues Allocated (NCQA): 16
Number of IO Submission Queues Allocated (NSQA): 16
```

```
werner@pc:~$ lscpu
CPU(s):                  6
  Model name:            Intel(R) Core(TM) i5-8400 CPU @ 2.80GHz
werner@pc:~$ ls /sys/block/nvme0n1/mq/
0  1  2  3  4  5
werner@pc:~$ cat /sys/block/nvme0n1/mq/0/cpu_list
0
werner@pc:~$ cat /sys/block/nvme0n1/queue/scheduler
[none] mq-deadline
werner@pc:~$ sudo nvme get-feature /dev/nvme0n1 -f 7 -H
get-feature:0x07 (Number of Queues), Current value:0x000f000f
Number of IO Completion Queues Allocated (NCQA): 16
Number of IO Submission Queues Allocated (NSQA): 16
```

Obwohl die NVMe SSD 16 Queues unterstützt, richtet blk-mq nur 6 Queues ein (da die CPU nur 6 Cores hat)

6 Cores, no HT

SATA HDD (Single Queue)



```
werner@pc:~$ lscpu
CPU(s):                  6
  Model name:            Intel(R) Core(TM) i5-8400 CPU @ 2.80GHz
    Thread(s) per core:  1
werner@pc:~$ uname -a
Linux pc 6.10.6+bpo-amd64 #1 SMP PREEMPT_DYNAMIC […] x86_64
werner@pc:~$ ls /sys/block/sda/mq/
0
werner@pc:~$ cat /sys/block/sda/mq/0/cpu_list
0, 1, 2, 3, 4, 5
werner@pc:~$ cat /sys/block/sda/queue/scheduler
none [mq-deadline]
```

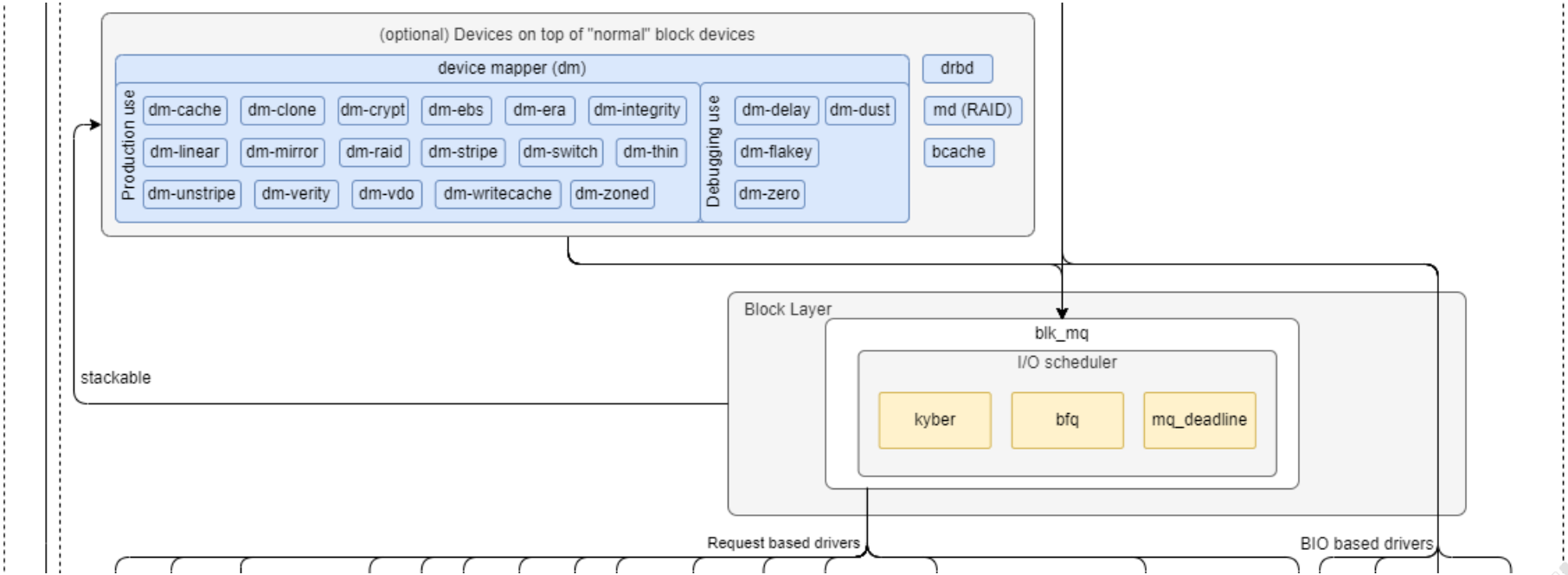
```
root@PMX5:~# nvme list -v
Device   SN                   MN
-------- -------------------- ----------------------------------------
nvme0    9240A029TLW9         KIOXIA KCD81RUG960G
root@PMX5:~# nvme get-feature /dev/nvme0n1 -f 7 -H
get-feature:0x07 (Number of Queues), Current value:0x007f007f
        Number of IO Completion Queues Allocated (NCQA): 128
        Number of IO Submission Queues Allocated (NSQA): 128
root@PMX5:~# cat /sys/block/nvme0n1/queue/scheduler
[none] mq-deadline
```

```
root@PMX5:~# lscpu
CPU(s):                  32
  On-line CPU(s) list:   0-31
  Model name:            AMD EPYC 9124 16-Core Processor
    Thread(s) per core:  2
    Core(s) per socket:  16
    Socket(s):           1
root@PMX5:~# ls /sys/block/nvme0n1/mq/
0  1  10  11  12  13  14  15  16  17  18  19  2  20  21  22  23  24  25
26  27  28  29  3  30  31  4  5  6  7  8  9
root@PMX5:~# cat /sys/block/nvme0n1/mq/0/cpu_list
0
root@PMX5:~# cat /sys/block/nvme0n1/mq/1/cpu_list
1
[…]
```

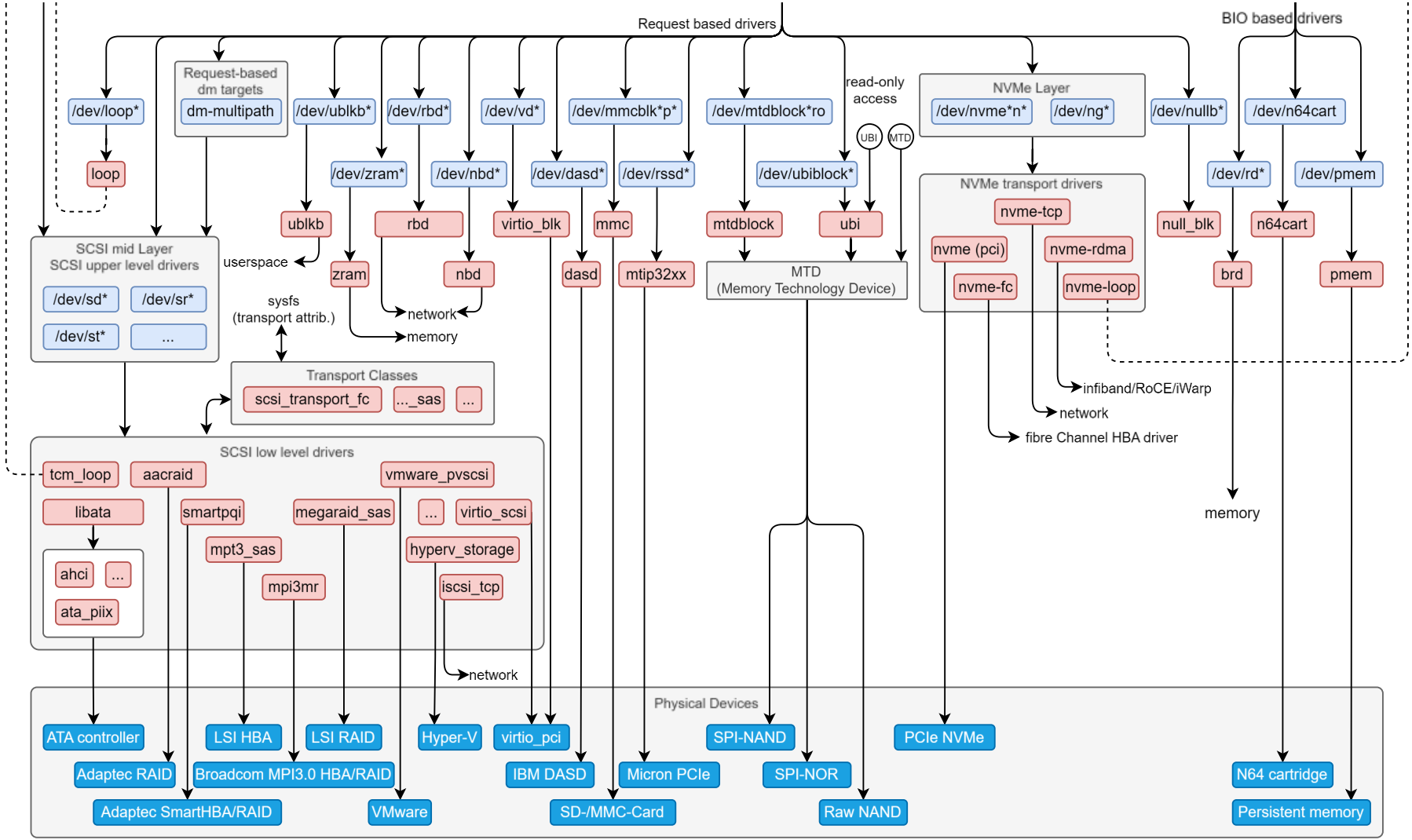
NVMe device

32 Hardware Dispatch Queues (die NVMe SSD würde 128 unterstützen)

Genau eine Software Staging Queue ist jeder HDQ zugeordnet

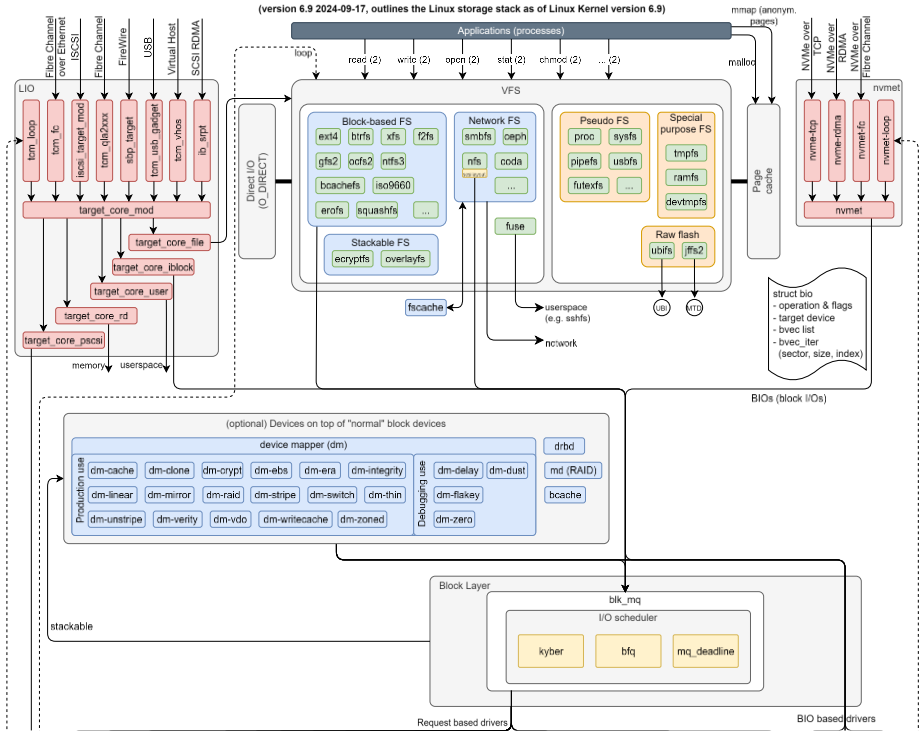

(optional) Devices on top of "normal" block devices
device mapper (dm)
Production use
dm-cache
dm-clone
dm-crypt
dm-ebs
dm-era
dm-integrity
dm-linear
dm-mirror
dm-raid
dm-stripe
dm-switch
dm-thin
dm-unstripe
dm-verity
dm-vdo
dm-writecache
dm-zoned
Debugging use
dm-delay
dm-dust
dm-flakey
dm-zero
drbd
md (RAID)
bcache
stackable
Block Layer
blk_mq
I/O scheduler
kyber
bfq
mq_deadline
Request based drivers
BIO based drivers

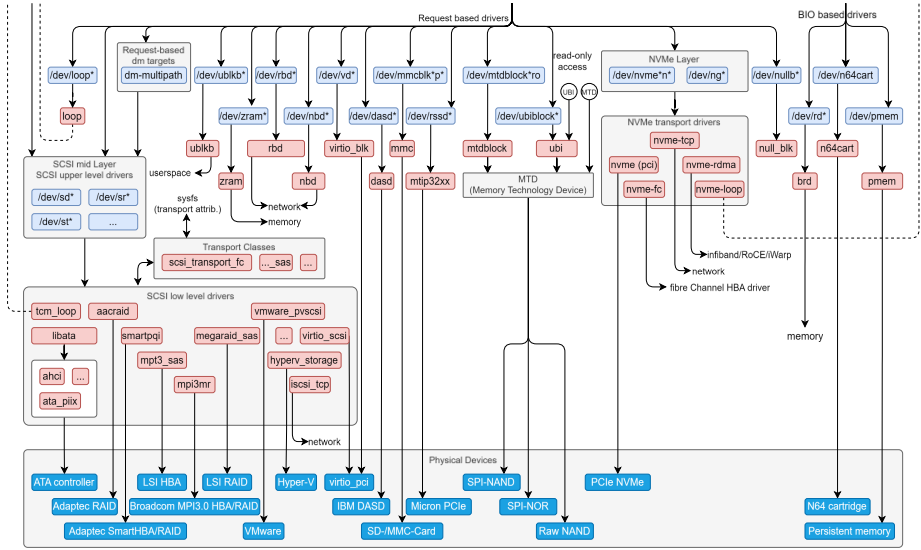
Gerätetreiber


Request based drivers
BIO based drivers
/dev/loop*
loop
Request-based dm targets
dm-multipath
/dev/ublkb*
/dev/rbd*
/dev/vd*
/dev/mmcblk*p*
/dev/mtdblock*ro
read-only access
NVMe Layer
/dev/nvme*n*
/dev/ng*
/dev/nullb*
/dev/n64cart
/dev/zram*
/dev/nbd*
/dev/dasd*
/dev/rssd*
/dev/ubiblock*
UBI
MTD
/dev/rd*
/dev/pmem
NVMe transport drivers
nvme-tcp
nvme (pci)
nvme-rdma
nvme-fc
nvme-loop
ublkb
rbd
virtio_blk
mmc
mtdblock
ubi
null_blk
n64cart
SCSI mid Layer
SCSI upper level drivers
/dev/sd*
/dev/sr*
/dev/st*
...
userspace
zram
nbd
dasd
mtip32xx
MTD
(Memory Technology Device)
brd
pmem
network
memory
sysfs
(transport attrib.)
Transport Classes
scsi_transport_fc
..._sas
...
infiband/RoCE/iWarp
network
fibre Channel HBA driver
memory
SCSI low level drivers
tcm_loop
aacraid
vmware_pvscsi
libata
smartpqi
megaraid_sas
...
virtio_scsi
mpt3_sas
hyperv_storage
ahci
...
mpi3mr
iscsi_tcp
ata_piix
network
Physical Devices
ATA controller
LSI HBA
LSI RAID
Hyper-V
virtio_pci
SPI-NAND
PCIe NVMe
Adaptec RAID
Broadcom MPI3.0 HBA/RAID
IBM DASD
Micron PCIe
SPI-NOR
N64 cartridge
Adaptec SmartHBA/RAID
VMware
SD-/MMC-Card
Raw NAND
Persistent memory

The “big picture”

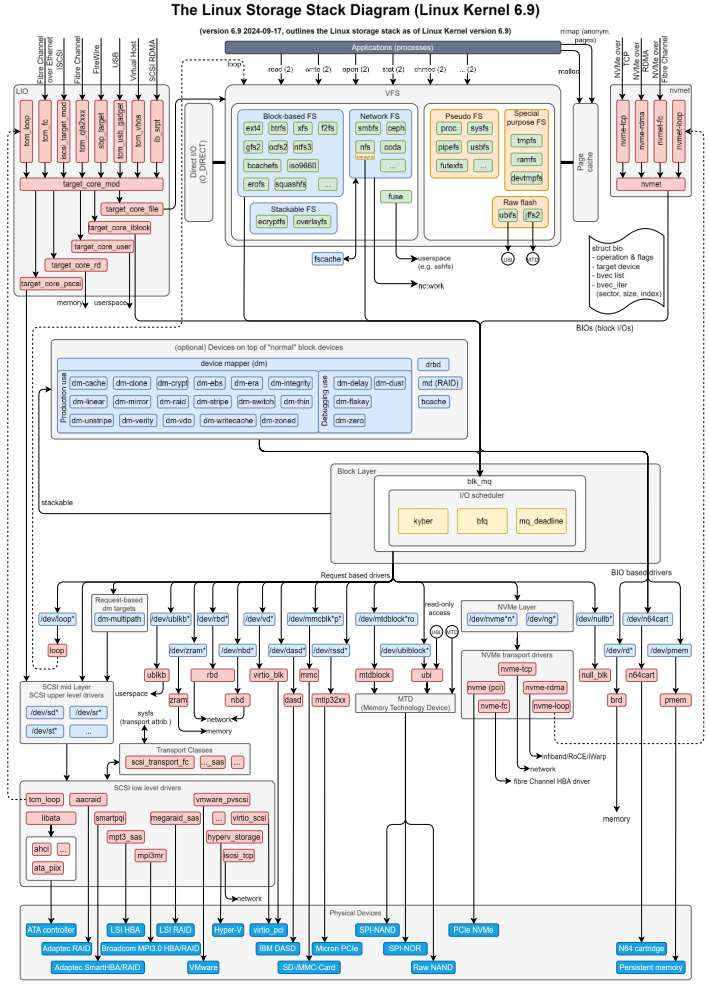
The Linux Storage Stack Diagram (Linux Kernel 6.9)





The Linux Storage Stack Diagram
https://www.thomas-krenn.com/en/wiki/Linux_Storage_Stack_Diagram
Design Werner Fischer, support by Christoph Hellwig, Richard Weinberger et al.

Danke an Christoph Hellwig, Richard Weinberger, Hannes Reinecke für zahlreiche Ideen und Beiträge zum Diagramm.



The Linux Storage Stack Diagram
https://www.thomas-krenn.com/en/wiki/Linux_Storage_Stack_Diagram
Design Werner Fischer, support by Christoph Hellwig, Richard Weinberger et al.
License: CC-BY-SA 4.0, see http://creativecommons.org/licenses/by-sa/4.0/

THOMAS KRENN®