Teil 3

Dateisysteme

Datenverwaltung auf externen Datenträgern

Ein Filesystem ist eine Organisationsform für Daten auf einem (Random-Access-) Datenträger [Festplatte, DVD, Flash, Memorycard/-stick…]. Damit werden Ablageorganisation und Zugriffsmöglichkeiten auf Dateneinheiten (Dateien) festlegt.
Üblicherweise handelt es sich um Software, die im Betriebssystem implementiert ist.

Dateisysteme

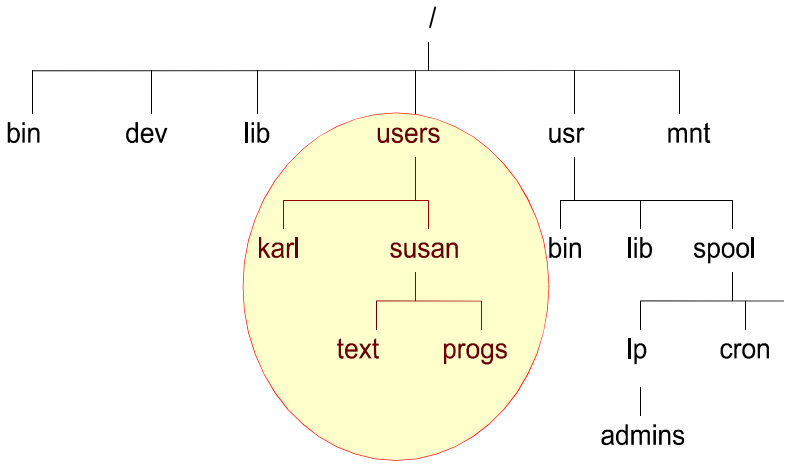
Dateien in modernen Dateisystemen:

- Verwalten Bytestrom (keine Struktur aus Sicht des OS)
- Erlauben wahlfreien Zugriff (Random Access)
- Haben neben Namen und Daten viele weitere Attribute, die zB der Sicherheit dienen können

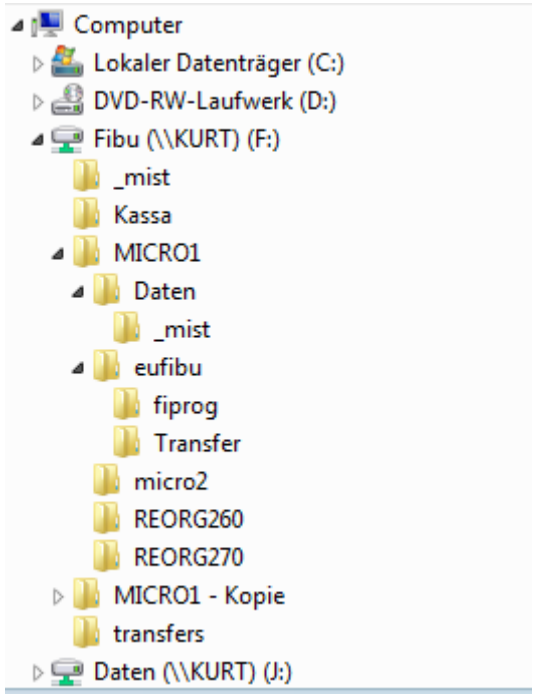
Organisation der Dateien

- derzeit übliche User-Sicht: Hierarchischer Verzeichnisbaum

1 Baum in Unix-Systemen



Mehrere “Laufwerke” mit je 1 Baum in Microsoft-Systemen



- früher auch nur 1 Verzeichnis von Dateien
- in Zukunft ev. andere Organisation (Datenbank-basiedende FS organisieren intern relational und können viele VIEWs anbieten)

Das für NT6 angekündigte WinFS wurde zurückgezogen!
ZFS wurde 2006 für Solaris 10 nachgereicht, Integration in Linux scheitert bislang an Lizenzstreitereien

Typische Zugriffsmethoden die das FS/OS der Anwendung bieten kann:

Klassischer Dateizugriff mit Systemcalls

- open
- close
- read
- write
- seek
- delete
- append
- getattributes
- setattributes
- rename

Die Applikation muss eine Datei öffnen und kann dann Daten zw. Memeory und FS übertragen, wobei für jede Übertragung ein Systemcall nötig ist.

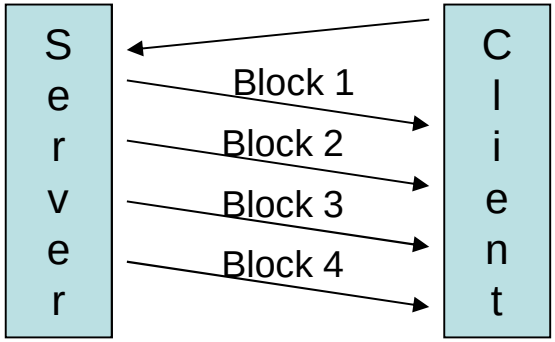
Memory Mapping:

- mmap(…)
- munmap(…)

Die Datei wird zum Auslagerungsspeicher für einen Teil der virtuellen Adressen des Prozesses!
→ Seitenzugriffe können über Pointer erfolgen
→ Das Lesen passiert mit Seitenfehlern (Page Fault)
→ Geschrieben wird mit dem Seitenersetzungsalgorithmus

Streaming:

Nach Öffnen und Startanwesung sendet der FS (der Server) unaufgefordert in definierter Periode die Daten



Dateisystem Implementierungen

Dateien bestehen aus Datenblöcken, die gefunden werden müssen!

Verwaltung der Blöcke einer Datei:

- Zusammenhängend (DVD)
- Verkettete Listen (wahlfreier Zugriff zu langsam!)
- Verkettete Listen im Memory (FAT)
- Inode
  - 64 Byte Inode von ufs
  - 128 Byte Inode von vxfs
- NTFS (ähnlich Inodes)
- Entwicklungen in Richtung Datenbank-basierend (Gnome-Storage [eingestellt], WinFS[schon eingestellt])

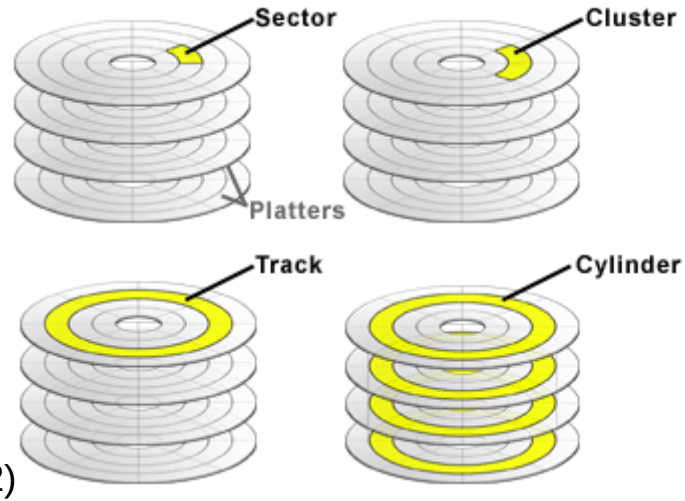
Dateisysteme auf Festplatten

Zusammenhang zwischen Sector, Cluster und (Daten-)Block

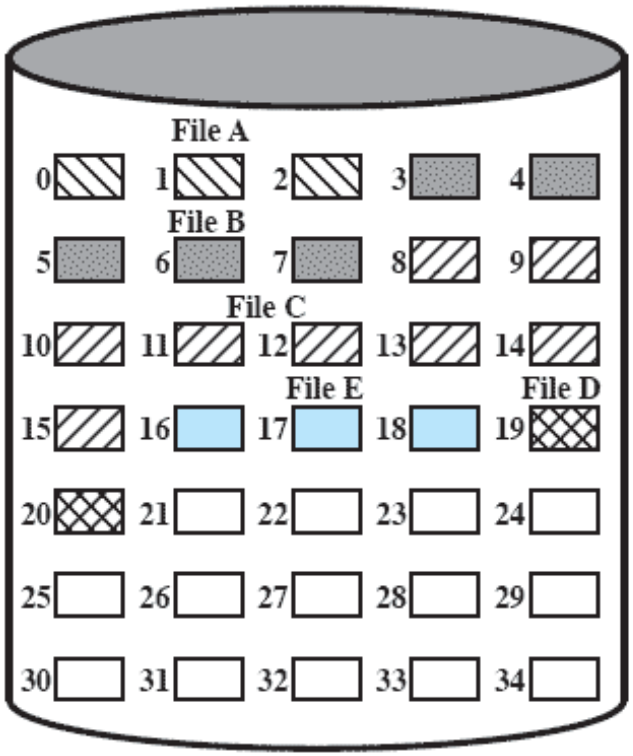
- Sector = Kleinste Lese-/Schreib-Einheit der HDD
  - 512 Byte, Neue Festplatten evt. 4KB
- Datenblock bezogen auf die HDD = Sector
- Cluster = kleinste Lese-/Schreib-Einheit des Dateisystems
- Datenblock bezogen aufs Filesystem = Cluster
  - Cluster (unter Windows üblich)
  - (Daten-)Block (unter Linux üblich)
- Früher CHS Adressierung
- Jetzt LBA Adressierung

$$LBA = (c \cdot H + h) \cdot S + s - 1$$

LBA	Adresse des Blocks
c	Zylindernummer
H	Anzahl der Leseköpfe (= Anzahl der Platten x 2)
h	Lesekopfnummer
S	Anzahl der Sektoren pro Track
s	Sektornummer



File Allocation “Zusammenhängend”



File Allocation Table

File Name	Start Block	Length
File A	0	3
File B	3	5
File C	8	8
File D	19	2
File E	16	3

ISO 9660

Merkmale von ISO 9660:

- Zusammenhängende Dateien
- Sektor = 2352 Byte, von denen 2048 für Daten benutzt werden. Rest für Sync (12), Mode(4) und Fehlerkor.(288)
  - CD-ROM: CD-ROM Mode 1 bzw. CD-ROM XA Mode 2 Form 1
- ersten 16 Sektoren frei (nicht in Norm definiert, verwendbar zB für Bootprogramm)
- Primärer Volume Deskriptor mit diversen ID´s und Verweis auf Root Directory
- Verzeichniseinträge können auch auf andere CD´s verweisen (1 großes VZ auf 1. CD möglich)
- 8.3 Namen mit sehr eingeschränktem Zeichensatz
- Maximal 8 Ebenen Verschachtelungstiefe der VZ

ISO 9660

Erweiterungen:

- ISO 9660-Level 2
  - Dateinamen bis 31 Zeichen
- ISO 9660-Level 3
  - Erlaubt fragmentierte Dateien. Multi-Extent-Dateien
- Rockridge
  - Erweiterung für Unix
- Joliet
  - Keine standardkonforme Erweiterung
  - Aber abwärtskompatibel wegen zwei Dateisystem(-baume)

Rock-Ridge

Merkmale von Rock Ridge:

- Erweiterung von ISO 9660
- Kann die zusätzlichen Eigenschaften der UNIX Dateien speichern (Perms, UID, GID, 3 x Timestamp
- DeviceNummern (für Gerätedateien)
- Symbolische Links
- beliebige Verzeichnistiefe
- Zeichensatz und Längenbeschränkungen wie in UNIX

Joliet

Merkmale von Joliet (MS):

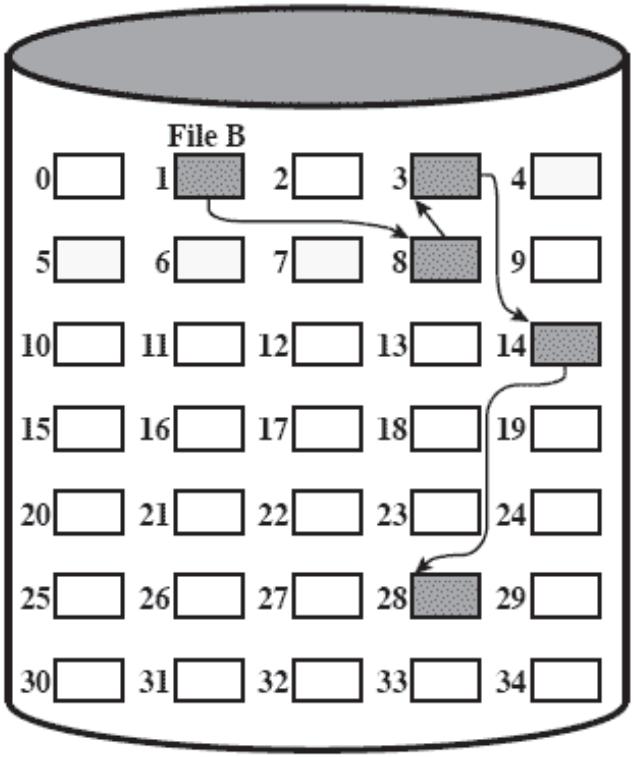
- Erweiterung von ISO 9660
  - nicht standardkonform
- Aber abwärtskompatibel wegen zwei Dateisystem(-baume)
  - ISO-9660 Dateisystem
  - Joliet-System
  - Verweisen auf die selben Dateiinhalte
- Lange Dateinamen (64 Byte)
- Unicode Zeichensatz
- keine Verzeichnistiefenbeschränkung
- Verzeichnisnamen mit Erweiterung

UDF (ISO 13346)

Universal Disk Format

- Offizieller Nachfolgr von ISO9660
- 256 Zeichen Dateinamen
- keine Ebenenbeschränkung
- 1023 Zeichen Pfadlänge
- 16-bit Zeichensatz-Unterstützung
- Case-Sensitiv-Unterstützung
- Attribute der gängigen Betriebssysteme enthalten
- Auch für andere RO- und RW-Medien gedacht (ev Ersatz für FAT!)

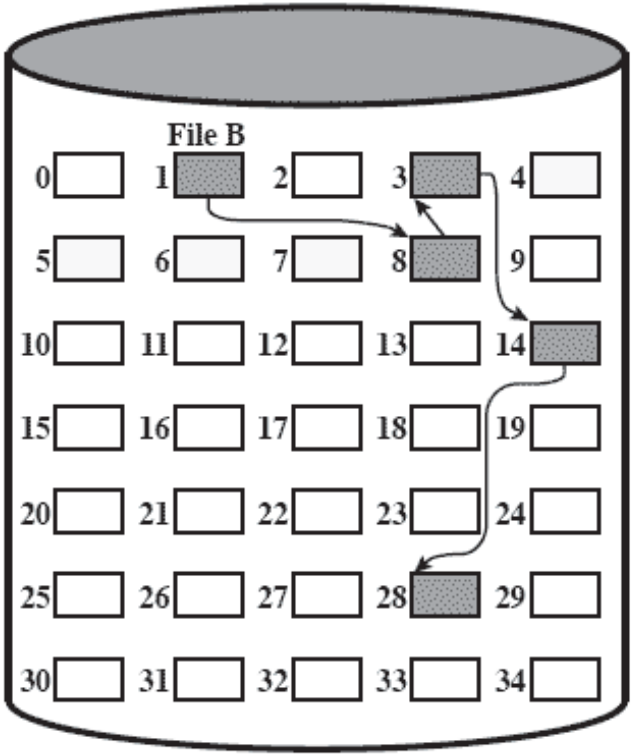
File Allocation “Verkettet”



File Allocation Table

File Name	Start Block	Length
...	...	...
File B	1	5
...	...	...

File Allocation “Verkettet mit FAT im Memory”



Verzeichnis:

Name	Startblock
….	
FileB	1
FileC	15
……	

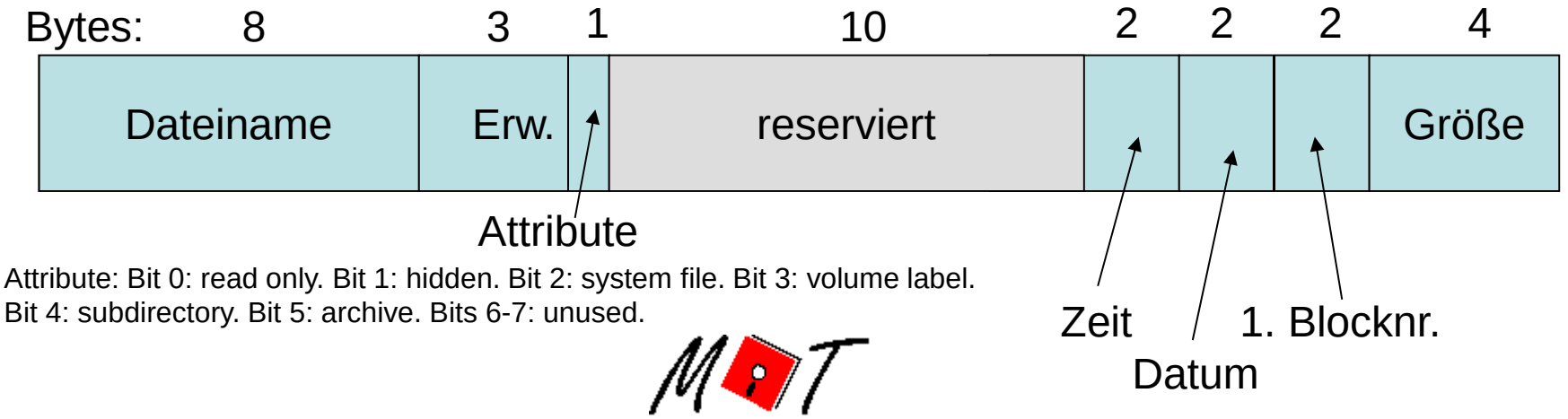
File Allocation Table

FAT:

0	x
1	8
2	x
3	14
4	x
5	x
6	x
7	x
8	3
…….	
14	28
…….	
27	x
28	e

(spätes) MS-DOS (FAT 16)

- Maximal File- und FS Size: 2 GB (bei maximaler Block[Cluster-]Größe von 32 KB)
- FAT benötigt 128 KB im Speicher (für 64K Einträge. Eintrag verweist auf Block mit zB 32KB Block-Größe)
  - Ein FAT Eintrag benötigt **16 Bit (2 Byte)**
- Verzeichniseintrag: 32 Byte



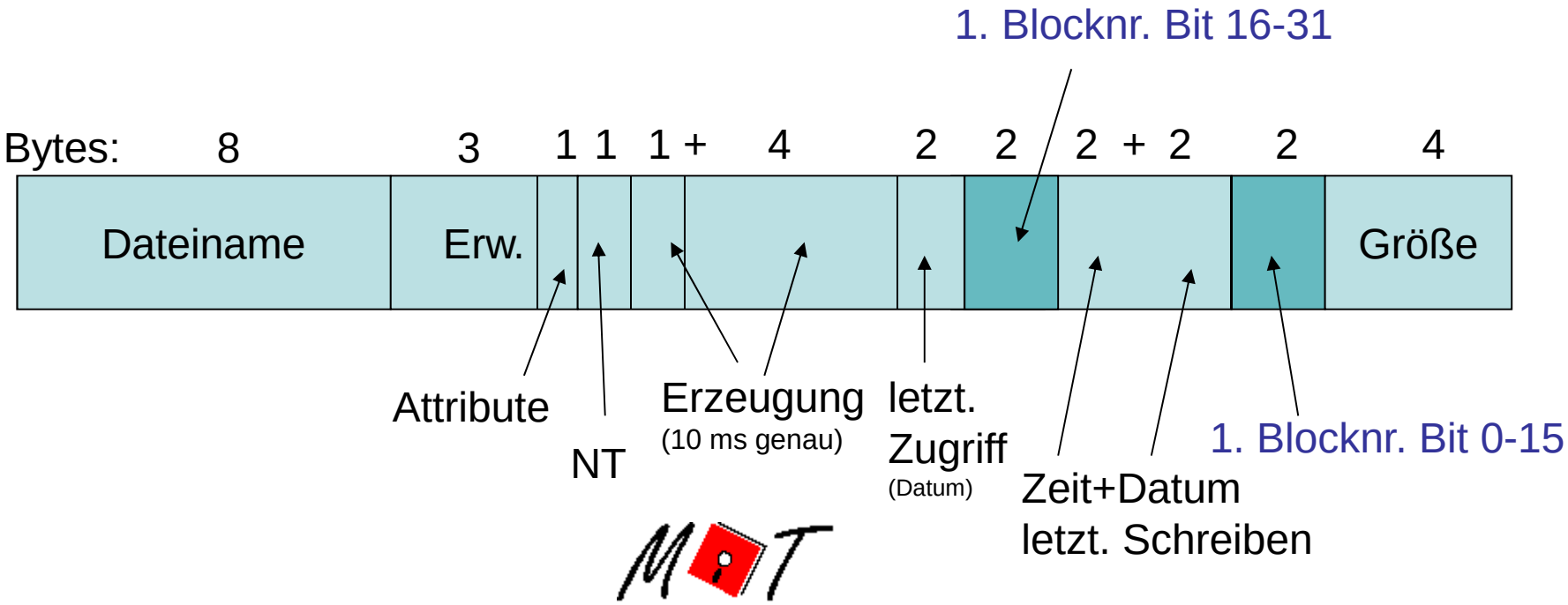
Attribute: Bit 0: read only. Bit 1: hidden. Bit 2: system file. Bit 3: volume label. Bit 4: subdirectory. Bit 5: archive. Bits 6-7: unused.

(spätes) MS-DOS (FAT 16)

- Verzeichniseintrag: Attribute
  - Bit 0: read only
  - Bit 1: hidden
  - Bit 2: system file
  - Bit 3: volume label
  - Bit 4: subdirectory
  - Bit 5: archive
  - Bits 6-7: unused
- Löschen einer Datei:
  - Erster Buchstabe wird durch 0xE5 ersetzt

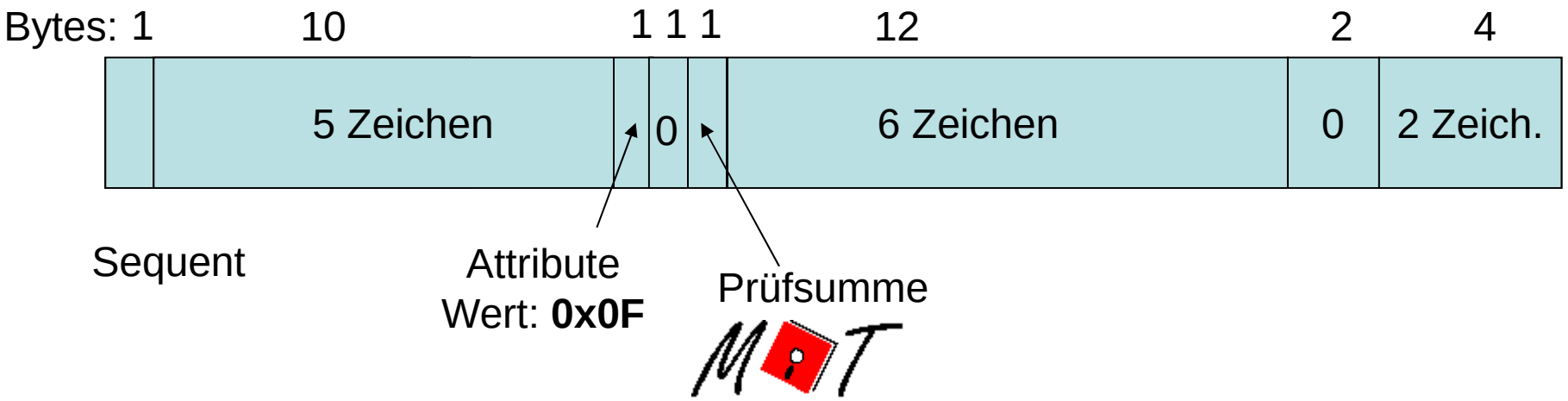
Windows 98 (FAT 32)

- Maximal FS Size 8 TB (bei Clustergröße von 32 KB)
- Dateilimit: 4GB - 1 wegen Größeneintrag (Feld 4 Byte)
- FAT benötigt zB: bei 4KB Blöcken auf 2 GB Platte 524288 Einträge = 2 MB RAM!
  - Ein FAT Eintrag benötigt **32 Bit (28 verwendet, 4 reserviert)**
  - 2^28 Clusters → max. Dateianzahl = 2^28
- Verzeichniseintrag: 32 Byte

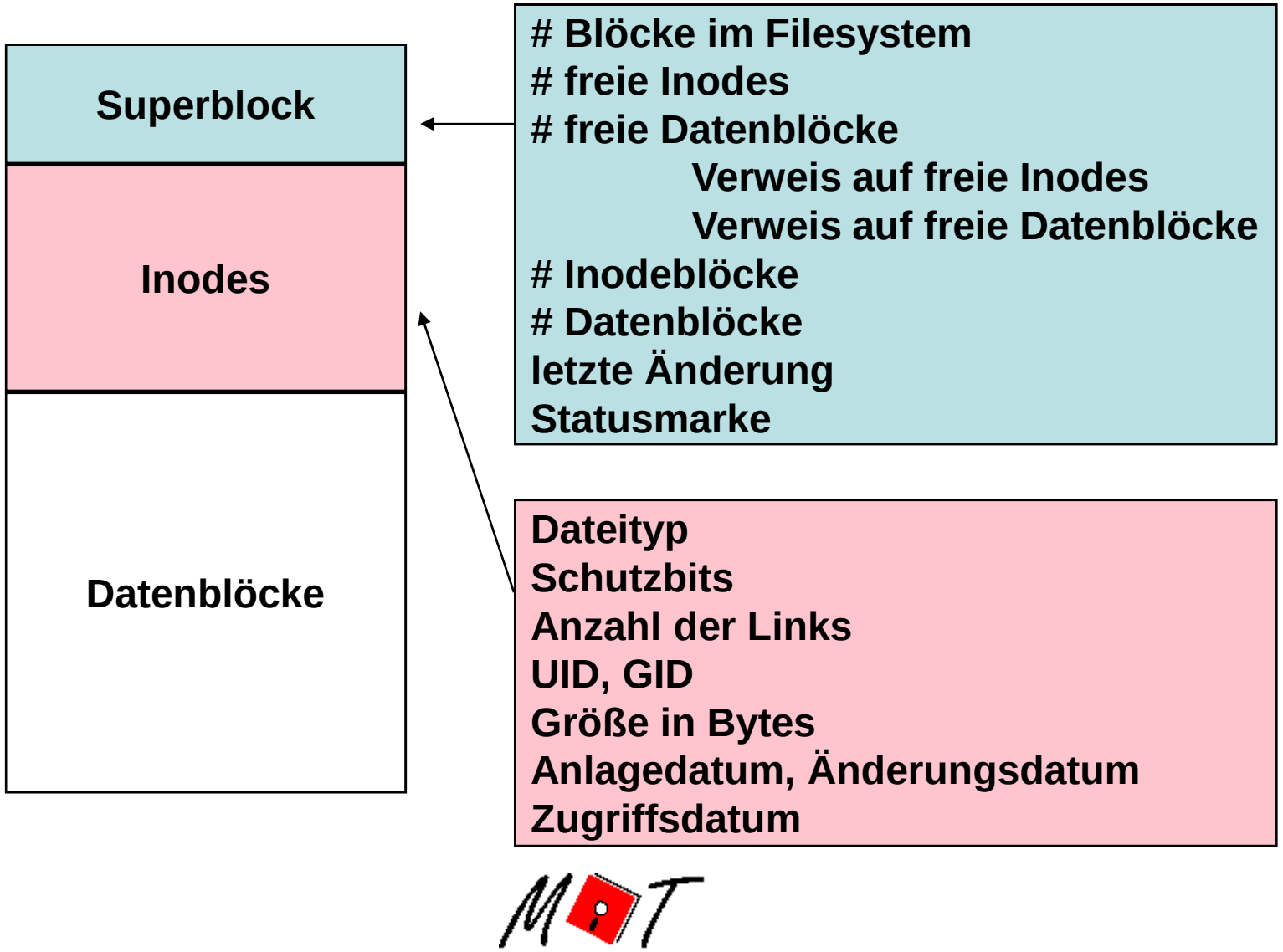


FAT – Umgang mit langen Dateinamen

- VFAT Virtual File Allocation Table
  - Erweiterung des FAT-Formats
  - Ermöglicht lange Dateinamen
- Vor 8.3 Namen können weitere Einträge stehen, die mit einer für DOS ungültigen Attributangabe (0x0F)markiert sind.
- Verzeichniseintrag, der Teil eines langen Namen speichert:



Logischer Aufbau von Unix Filesystemen (File Allocation: Inodes)



Das Unix Directory

- Inode Number
- Name

Hard-Link

directory

123	Name_1
123	Name_5
123	Name_6

Symbolic-Link

directory

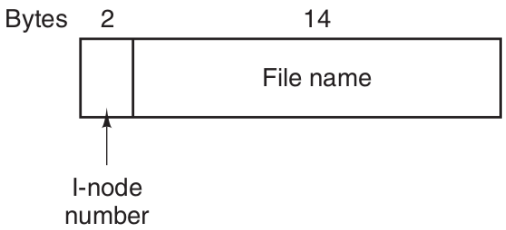
123	Name_1
234	Name_2
177	Name_3

datafile

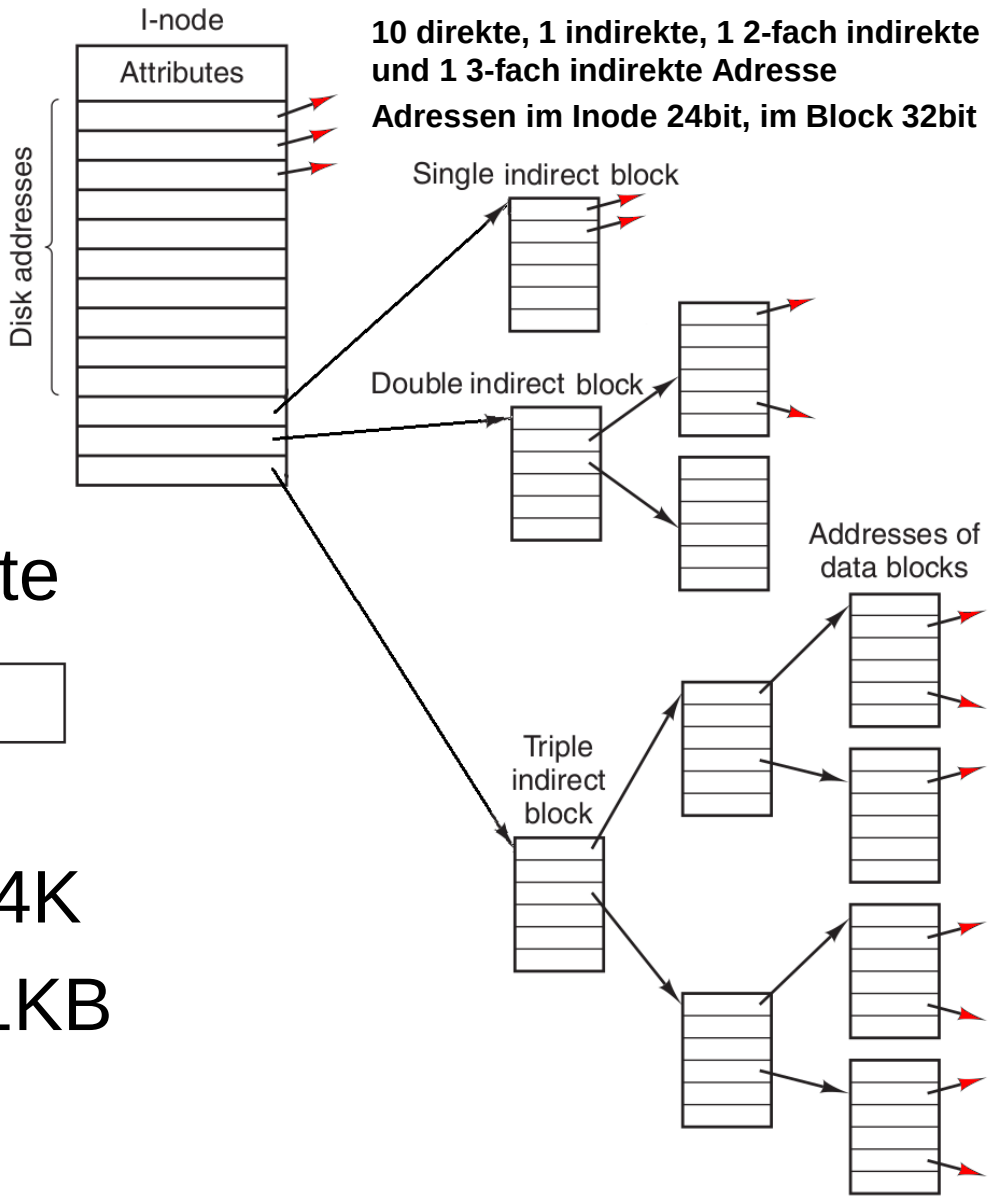
/Pfad.../Name_2

Unix System V File System (s5fs)

- Vorgänger des UFS (Unix File System)
- Superblock 512 Byte
- Inode 64 Byte
- Directory Entry 16 Byte

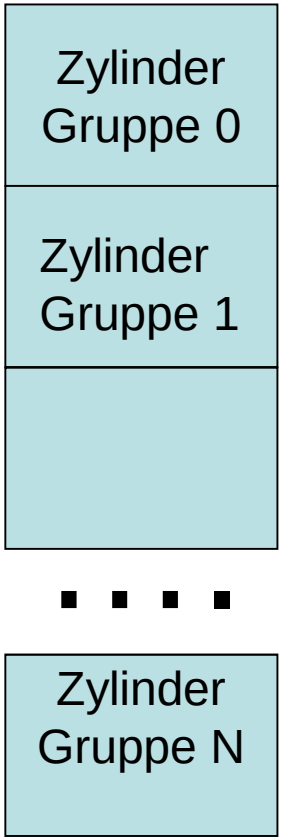


- Max Anzahl der Files: 64K
- Blocksize: 512B oder 1KB

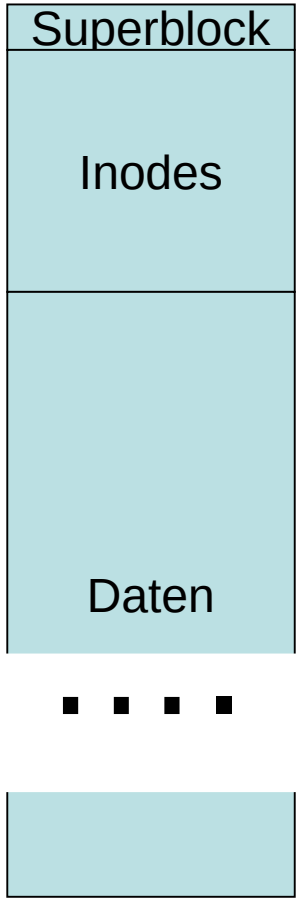


UFS – Unix File System

Filesystem



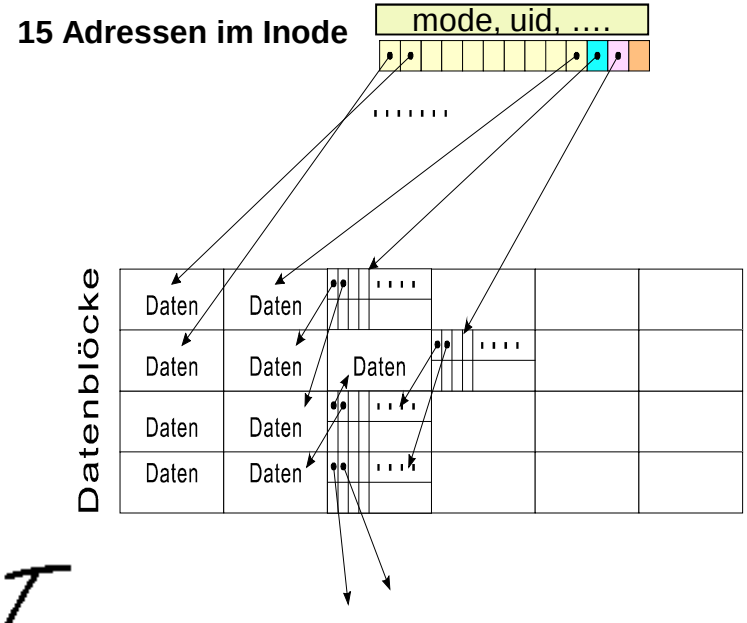
Zylindergruppe



- Inode 32 bit (statt 16)
- Unterschiedliche Erweiterungen
  - Solaris UFS, UFS2, FFS

FFS: Inode 128 Byte, Adressen: 12 direkte, 1 indirekte, 1 2-fach indirekte und 1 3-fach indirekte Adresse

Adressen im Inode 24bit, im Block 32bit



Konsistenz des Dateisystem

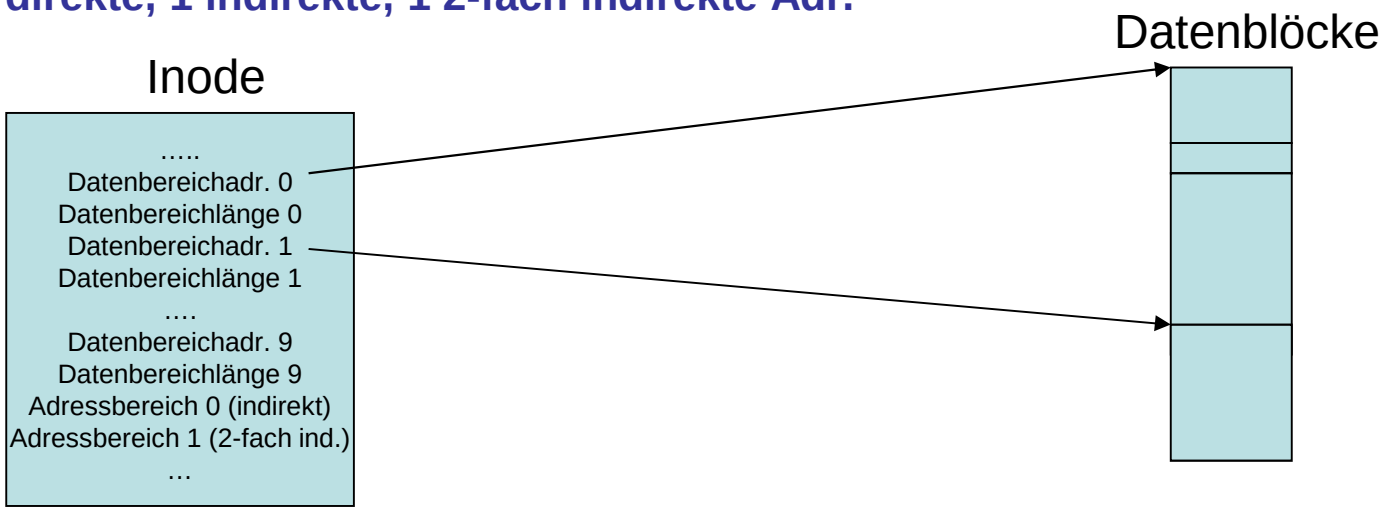
- Konsistenz: Korrektheit der gespeicherten Informationen (Metadaten und/oder Dateiinhalt)
- Ziele: Konsistenz auch bei plötzlichen Stromausfall
  - Konsistenz des Dateisystems (Metadaten, nicht Dateiinhalt)
  - Konsistenz der Dateiinhalte
- Realisierung
  - **Journaling**: Zeichnet vor dem Schreiben im Journal auf
    - Metadaten-Journaling
    - Full-Journaling
  - Softupdates: Konsistenz der Metadaten ohne Journaling (in UFS)
    - Blöcke werden in einer bestimmten Reihenfolge geschrieben
    - Stromausfall: Blöcke können als belegt markiert sein, obwohl schon frei (fsck)
  - Copy-On-Write (ZFS, btrfs, Beschreibung folgt in den hinteren Folien)

Adressierung im Veritas Filesystem (VxFS)

(bereits in den 90er Jahren in Unix basierenden Serversytem weit verbreitet!)

- Extent-basiertes Dateisystem (Extent = Mehrere zusammenhängende Blöcke)
- 128 Byte Inodes
- 12 Extentadressen (96 Bytes)
- die direkten Adr. bestehen aus Startblocknummer + Größe [=Anzahl Blöcke]

10 direkte, 1 indirekte, 1 2-fach indirekte Adr.



sollte Dateiinhalt < 96 Byte → Daten in den Inode statt den Adressen!

Performance-Features im Veritas Filesystem (vxfs)

- Datenspeicherung:
  - <=96 Bytes im Inode
  - Adressierung von Extents variabler Größe
  - Fixe Extent Größe von 64 Blöcken bei indirekter Adressierung
- Integrierte Defragmentierungs und Directory-Optimierungs Kommandos
- Filegröße (-> zusammenhängende große Extents) kann reserviert werden
- Optimierter fsck durch Transaction-Bitmaps in jeder Allocation Unit (AU entspricht etwa den ZylinderGruppen des UFS)
- Optimierte Verwaltung des freien Speichers durch Freispeicherbitmaps (statt Verkettung)

Linux Filesysteme

- ext2 (extended FS)
- ext3
- ext4 (aktueller Standard ab Kernel 2.6.19 [2008])
- btrfs (möglicherweise die Zukunft, Ähnlichkeiten mit zfs, zB.: snapshots, ACL, compression, subvolumes, RAID-integration, transaction based, …)
- xfs (SGI Entwicklung, ACLs, Quotas, optimiert für große Dateien, wie Videos, …)

ext2, ext3, ext4

ext2,3: Klassiche Adressierung (Vgl. ufs)

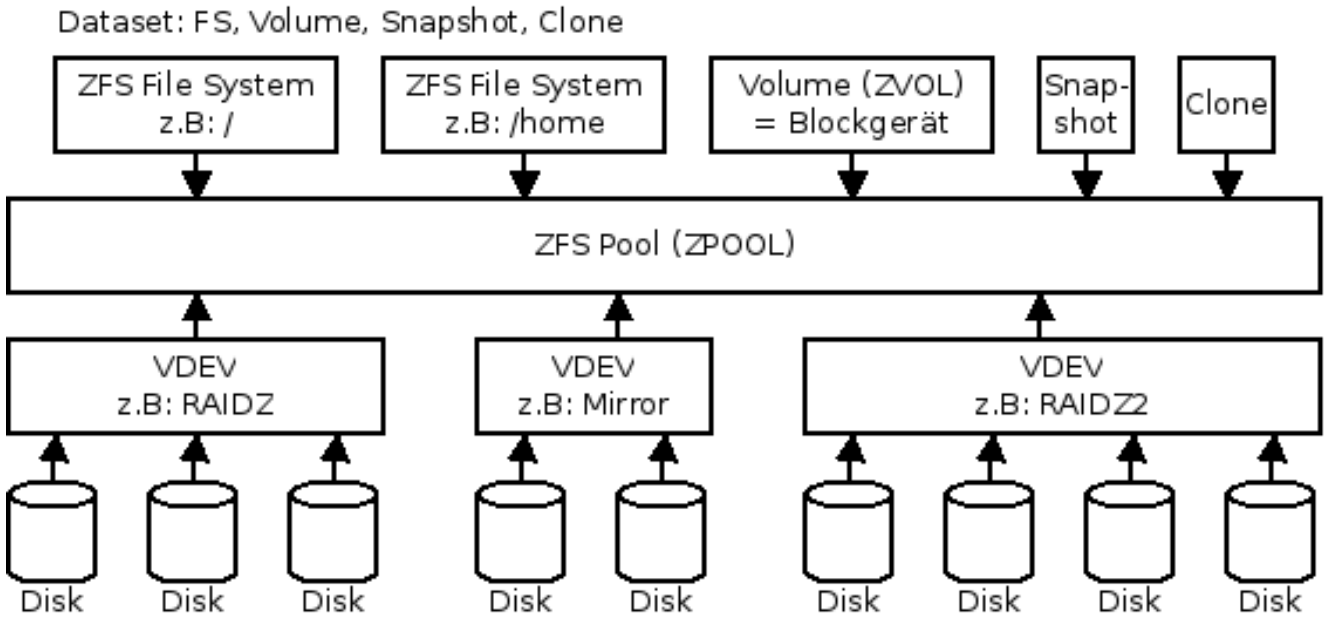
ext3: Journaling (→schneller fsck), ACL

ext4:

- FS size: 1 EB, File size: 16 TB
- Extent Adressierung (Vgl. Veritas):
  - 4 Extent-Adressen im Inode, sonst Htree (hashtable tree)
  - erweitererte Timestamps (kein year 2036 problem)

ZFS

- Kombiniert
  - **Dateisystem** und
  - **Logical Volume Manager** (LVM)



ZFS

Datenintegrität erkennt „stille Verfälschung“ durch Checksummen. Behebung durch Redundanz

Storage pools & RAID Implementation Raid-1, Raid-Z, Raid-Z2 and Raid-Z3 (2009)

Capacity ZFS ist ein 128-bit Dateisystem. Dadruch kann es 18 quintillion (1.84×10^{19}) mal mehr Daten adressieren als aktuelle 64-bit Systeme.

Copy-on-write Transactions alles wird in neue Blöcke geschrieben, danach werden die Referenzen an diese neuen Blöcke angepasst.

Snapshots & Clones ohne Overhead!!! (copy on write), Klone veränderbar, Snapshot nicht

Dynamic Striping in Zpool Verteilt Blöcke auf VDevs. Beste Performance, ermöglicht dyn. Wachsen.

Variable Blocksize keine fixe Blockgröße, 1776 Bytes brauchen 1776 Bytes! (+Checksumme)

Adaptive Replacement Cache (ARC) statt klassischem **page cache**: Kombination aus Recently Used, Frequently Used und Verdrängungs-History (IBM Entwicklung)

Data Deduplication ab 2009 (doppelte gleiche Blöcke werden eliminiert)

Access Optimization Priorities, Deadline-Scheduling, Global IO Request Sorting, Prefetching

Quotas, ACLs, Compression, Encryption per User & per Group Quotas, NFSv4 ACLs

NFS & CIFS Integration CIFS beinhaltet richtige SID Verwendung (anstatt UID Mapping)

NTFS

- Master File Table (MFT)
- Ist eine Datei, deren 1. Blockadresse im Bootblock steht
- Besteht aus 1 KB großen Einträgen (File Record), die jeweils 1 Datei beschreiben
- Die ersten 26 Einträge (sind auch Dateien) enthalten Metadaten von NTFS (MFT selbst + Kopie davon, Logdatei (für Recovery), Volumeinfo, Bitmap mit belegten/freien Blöcken, Wurzel-VZ, Quotas, Security*[Win200x]*…)
- Benötigt eine komplexe Datei mehrere MFT-Einträge, verweist der 1. Eintrag (Base File Record) auf die weiteren.
- MFT beinhaltet einen Eintrag für **jede Datei** auf dem NTFS Volume.
- Jedes File besitzt eine **File Record Number**. Diese entspricht dem **Index für den MFT-Eintrag** (File Record) in der MFT.

Jeder MFT-Eintrag (File Record) besteht aus:
Eintragskopf, Attribut-Kopf+Attribut-Wert, Attribut-Kopf+Attribut-Wert, ….

Benutzte Bytes im Eintrag
ID des Basiseintrags (bei mehreren Eintr. pro Datei)
GültigkeitsKZ
Sequenznummer (Verwendungszähler)
…..

variable Anzahl
variable Größe
manche können mehrfach vorkommen
Wert klein → steht im MFT-Eintrag (resident attr.),
Wert groß → Block-Adressen im MFT-Eintrag
(= ausgelagertes Attribut,
nonresident attribute)

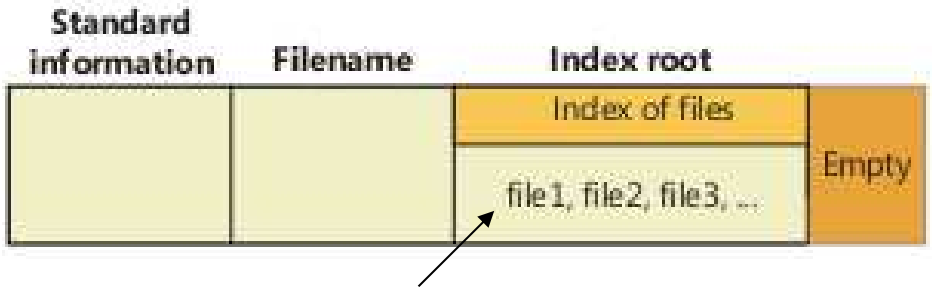
NTFS – MFT Attribute

- Standardinformation (Zeiten, Posixrechte,read/archiv Bit)
- Dateiname (ev. mehrere für DOS 8.3 Namen oder Hardlinks)
- Security Deskriptor (NT4: Rechte, stehen ab Win2000 in eig. Datei)
- Attributliste (Ort der weiteren MTF Einträge zu dieser Datei)
- Objekt-ID (64 Bit ID)
- Reparse (Mount-Points und Symlinks)
- Daten (enthält für große Dateien Blockadressen oder die Daten selbst für kleine Dateien, kann ebenfalls mehrfach auftreten [mehrere Datenstreams!])
- ….

MFT Verzeichnisse

- kleine VZ sind Tabellen
- große VZ sind B+-Bäume

MTF Eintrag für kleines VZ:



Verzeichniseintrag besteht aus:

- MTF Index der Datei (File Record Number)
- Länge des Namens
- .....
- Name

} *Struktur fixer Länge*

→ *VZ- od. Dateiname variabler Länge*

Adressierung in NTFS
(= Attribut Daten in der MFT)

VCN: Virtual Cluster Number … Clusters bezogen auf eine Datei von 0 bis m
LCN: Logical Cluster Number … Nummerierung aller Cluster von Begin zu Ende des Volume
VCN-to-LCN Mappings … Zuordnung von VCN zu LCN

MFT-Eintrag für Datei: Eintrag für Datei

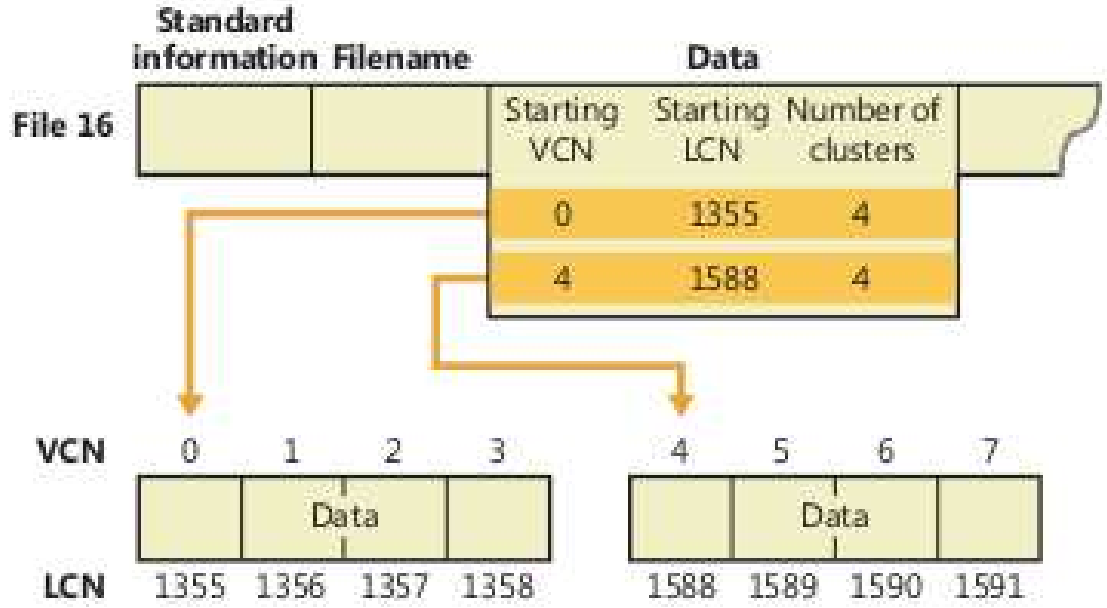


FIGURE 12-37 VCN-to-LCN mappings for a nonresident data attribute

Adressierung in NTFS
(= Attribut Daten in der MFT)

MFT-Eintrag für Datei: Eintrag für Datei(teil)

mehrere Einträge, falls Datei mit Löchern (Spare File) erzeugt wurde

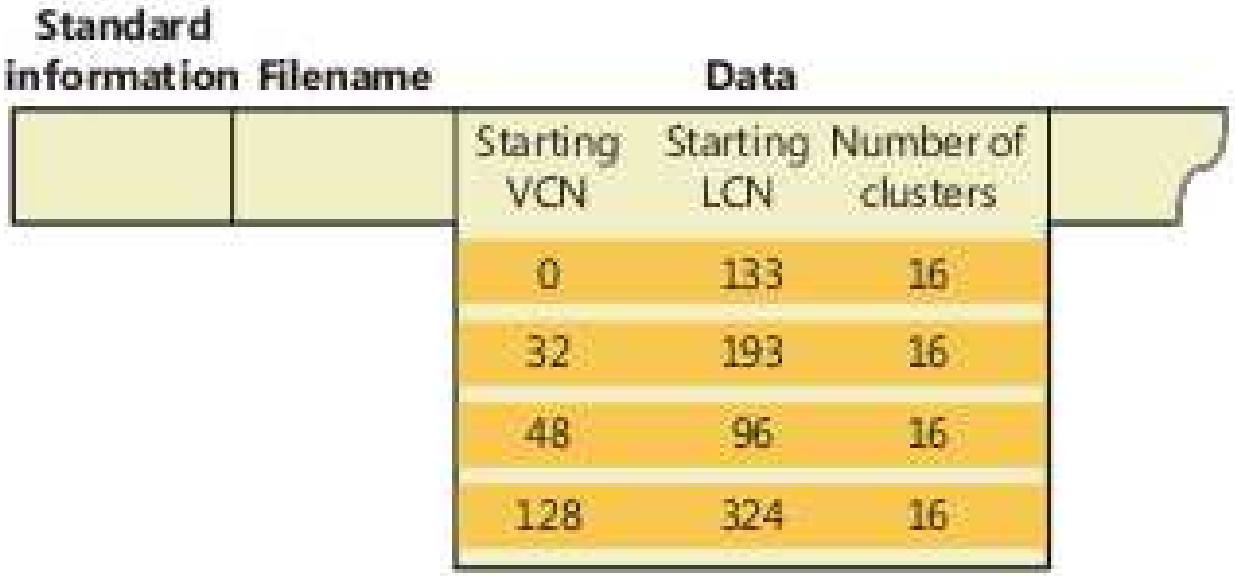


FIGURE 12-41 MFT record for a compressed file containing sparse data

NTFS Komprimierung

- Transparent für Applikationen direkt beim Lesen/Schreiben
- Datei wird in 16 Blöcke Einheiten zerlegt und für jede Einheit entschieden, ob komprimierbar
- Verzeichnis Komprimierung → Jede Datei im Verzeichnis wird komp.

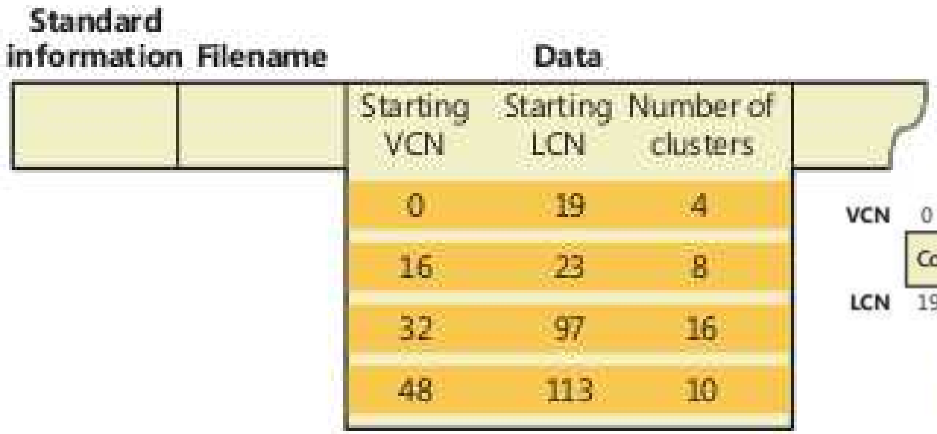
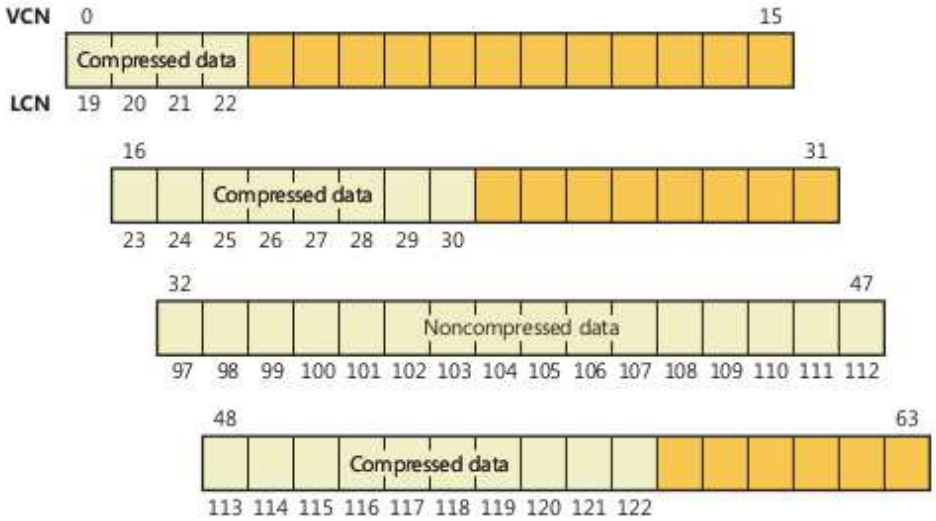


FIGURE 12-43 MFT record for a compressed file



NTFS Versionen

- NT 6 (NTFS v3.2, Vista, 2008)

  Transactional NTFS (series of file operations done as a transaction), NTFS symbolic links, Partition shrinking and self-healing (automatisch zur Laufzeit) functionality

- NT 5 .1 (NTFS v3.1, XP, 2003)

  Redundant MFT Entrys, better Recovery

- NT 5.0 (NTFS v3.0, 2000)

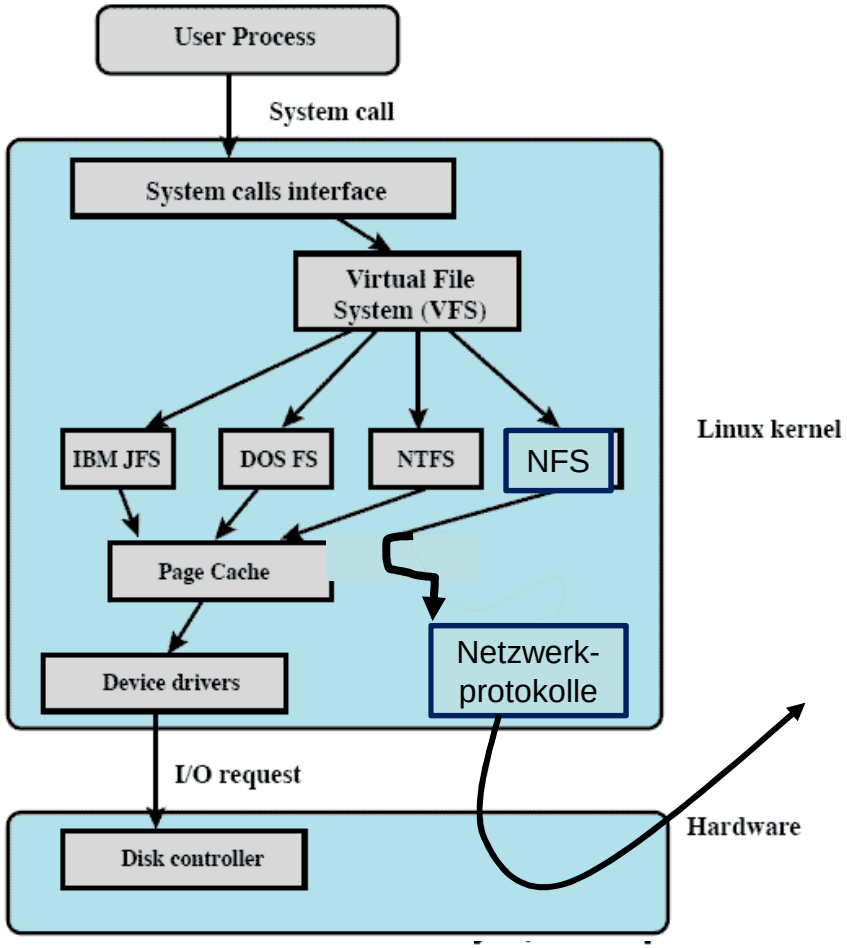
  Quotas, EFS (Encrypting File System), sparse Files, reparse points (Mountpoints), USN Journaling (Registiert Dateiänderungen), All Security Descriptors together in separate File (runtime ACL inheritance), Volume-Shadow-Copy (= Snapshot) (per copy on write),

- NT x (NTFS v1.x, <2000)

  compressed files, multiple named data-streams (mehere Datenzweige), ACL-based security, NTFS Metadata Journaling, MS & Posix Attributes

Netzwerkfilesysteme

Daten die über Netzwerk angeboten werden, sind ebenfalls über Filesystemschnittstellen ansprechbar, auch wenn dahinter nicht ein Gerätetreiber, sondern ein Netzwerkprotokoll liegt.



NFS
CIFS/SMB/Samba
NCP

Snapshot-Implementierungen (in Filesystemen)

- dienten ursprünglich nur dazu während des Sicherungsvorgangs eine konsistente FS Sicht zu haben

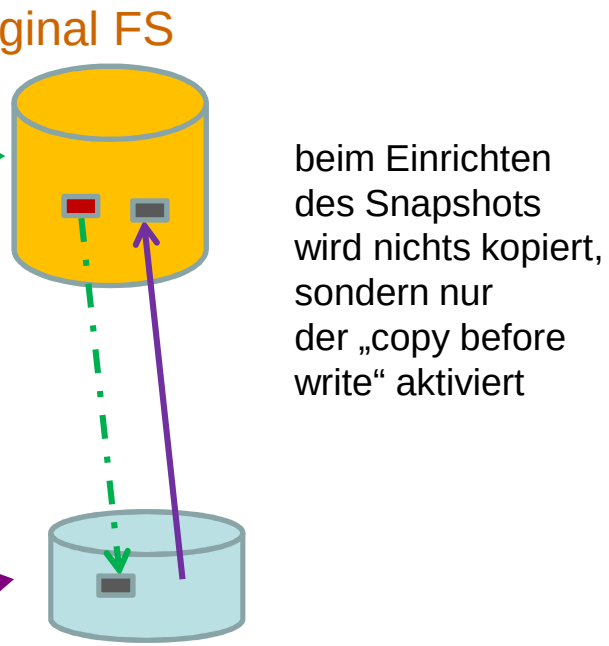
  → existieren nur temporär und arbeiten mit einem „copy-before-write“ Mechanismus auf extra Speicherplatz

- werden heute auch für „online“-Sicherungen (zB Wiederherstellungspunkte“) benutzt

  → bleiben erhalten und arbeiten mit einem „copy-on-write“-Mechanismus direkt im (File-)System

Herkömmliche Snapshot Implementierungen

Vertreter: ufs, vxfs, Linux-LVM



beim Einrichten des Snapshots wird nichts kopiert, sondern nur der „copy before write“ aktiviert

<u>Lesezugriff auf Life-FS mit aktivem Snapshot:</u>
Zugriff auf das Original

<u>Schreibzugriff auf Life-FS mit aktivem Snapshot:</u>
Wenn die Seite im Snapshot-Speicher noch nicht vorhanden ist,
wird die Seite zuerst dorthin kopiert;
verändert wird die Seite im Original FS

Wenn die Seite im Snapshot-Speicher vorhanden ist,
wird diese geliefert,
wenn nicht, wird auf das Original FS „durchgegriffen“

Moderne Snapshot Implementierungen

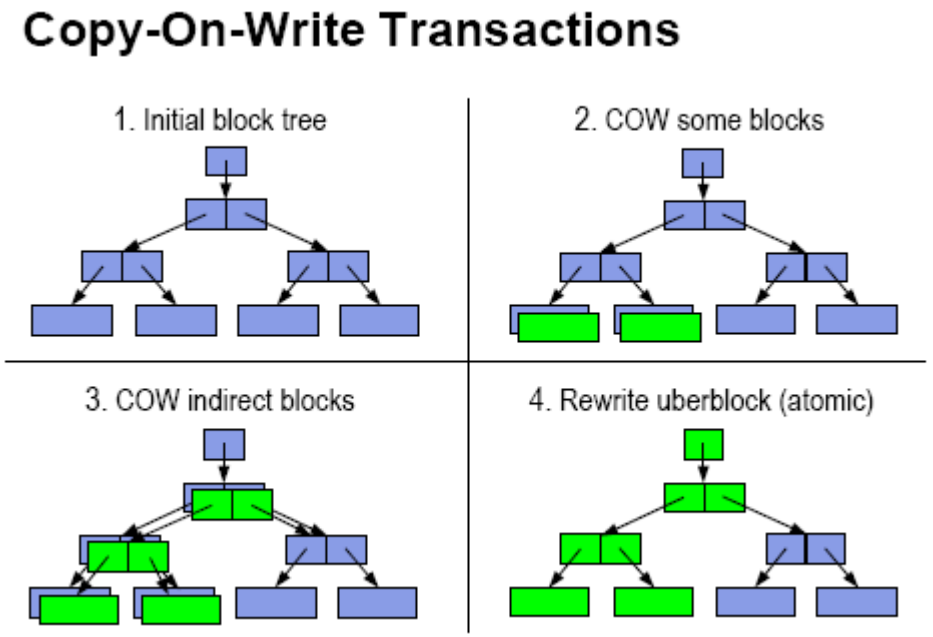
Vertreter: zfs, btrfs, ntfs für shadow copy

Alles passiert im (File-)System (in der Regel auf Blockebene)

wobei copy-on-write immer verwendet wird → Transactional FS

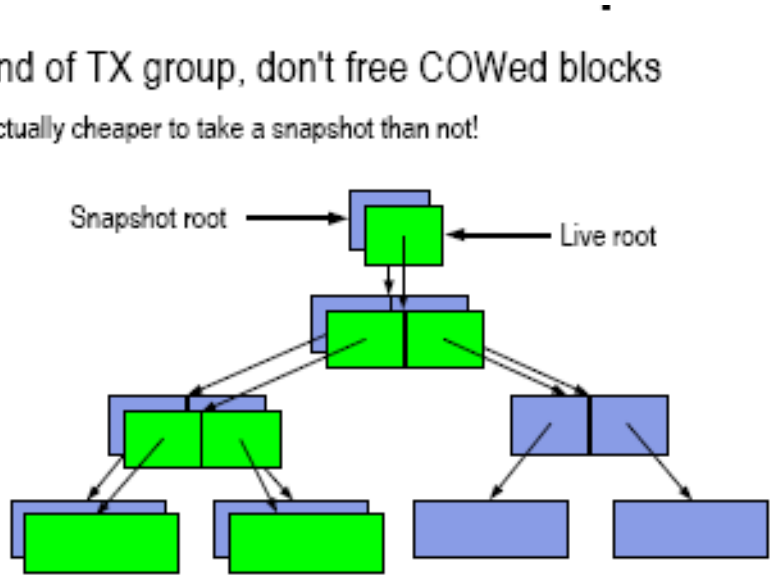
Beipsiel zfs:

Transaction ohne Snapshot:



Moderne Snapshot Implementierungen

Transaction mit Snapshot (Bsp. zfs):



Zugriff auf beliebigen Snapshot: