11. Dateisysteme

Michael Schöttner

Betriebssysteme und Systemprogrammierung

11.1 Vorschau

- FAT16
- UNIX Inodes
- UNIX System V
- EXT2, 3 und 4
- Zugriffsrechte
- NTFS
- Journaling

11.2 Einführung

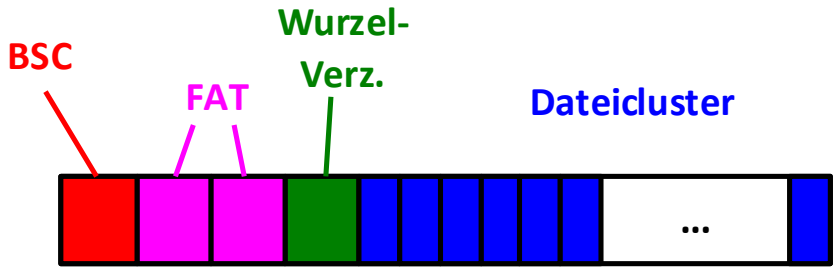
- Anforderungen: Persistenz, Zugang per Name, Schutz durch Zugriffsrechte.
- Datei/File (Behälter zur dauerhaften Speicherung beliebiger Daten):
  - Programme und Daten (Dateistruktur abhängig vom Dateityp),
  - Attribute: Name, Typ, Größe, letzte Änderung, Zugriffsschutz, ...
- Namenskonventionen:
  - UNIX, NT: 256 Zeichen a.b.c.d
  - UNIX: Groß- & Kleinschreibung beachten!
- Dateien können mehrere Streams (Unterbereiche) haben:
  - DOS: nur ein Stream
  - MacOS: Data Fork, Resource Fork
  - NT: mehrere Streams möglich → datei:stream
    (default stream ohne Suffix = :0).

11.2 Einführung

- Verweis (Link, Shortcut, Alias):
  - direkter Zugriff, ohne navigieren zu müssen.
  - **UNIX hard link**: Verweis auf Inode: ln datei link
    (Datei erst löschen, wenn letzter Hardlink entfernt wurde)
  - **UNIX symbolic link**: Verweis auf Dateinamen: ln –s datei link
    (Link ungültig, falls Datei gelöscht)
  - **Hard links bei NTFS**: für mit `mklink` über Terminal

11.3 File Allocation Table - FAT

- Belegungseinheiten heißen Cluster (z.B. 512, 1024, 4096 Bytes).
- Partitionsstruktur:
  - BSC = Bootsector → optionaler Bootcode
  - FAT = Tabelle mit fester Größe zur Verwaltung freier und belegter Blöcke (repliziert, aus Zuverlässigkeitsgründen)
  - Wurzelverzeichnis: Länge im BSC
  - Dateicluster: eigentliche Datenblöcke



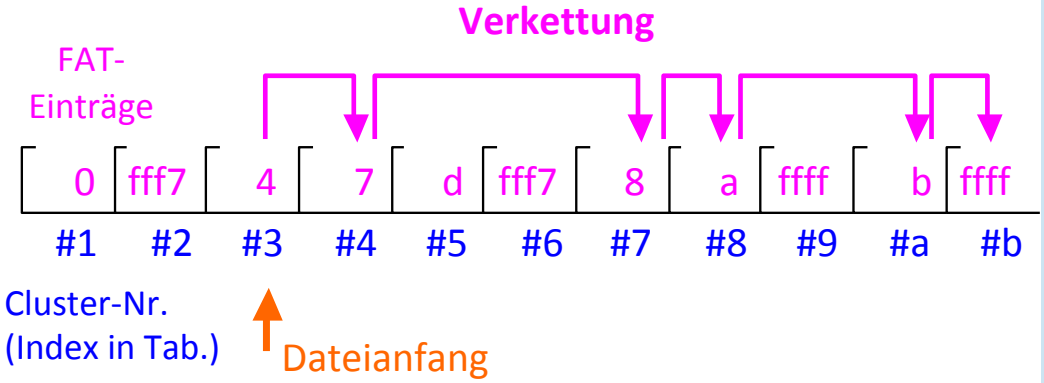
11.3 File Allocation Table - FAT

- FAT16:
  - Adressierung mit 16-Bit → max. 65.536 Cluster
  - Max. 32 KB Cluster → max. 2 GB pro Partition

- FAT32:
  - Adressierung mit 28-Bit → >2 GB und weniger Verschnitt, da kleinere Cluster verwendbar sind

FAT-Tabelle

- Separate Verkettung der Blöcke in Dateizuordnungstabelle (FAT):
  - für jeden Cluster einen Eintrag (16 Bit)
  - Kopie im Hauptspeicher (schneller Zugriff)
- Cluster/Blöcke einer Datei sind über die FAT verkettet:
  - 0 = leerer Block
  - 0xFFF7 = schadhafter Block
  - 0xFFFF = letzter Block einer Datei
- Der Dateianfang steht im Verzeichnis



Aufbau eines FAT16 Verzeichnisses

- Verzeichnisse sind ebenfalls Dateien (bei fast allen Dateisystemen)
- Aufbau eines Datei-Eintrags:

Offset	Größe	Inhalt
0	**8**	**Dateiname**
8	**3**	**Dateinamen-Erweiterung**
11	1	Attribute: Archiv-Bit, read-only, Verzeichnis/Datei, versteckt
12	10	Reserviert
22	2	Uhrzeit der Erstellung
24	2	Datum der Erstellung
26	**2**	**Startcluster der Datei**
28	**4**	**Dateilänge in Bytes**

Aufbau eines FAT16 Verzeichnisses

- Datei-Eintrag zeigt auf ersten Block einer Datei, bzw. ersten Eintrag in der FAT-Tabelle
- Länge der Verzeichnisdatei steht:
  - im übergeordneten Verzeichnis.
  - Das Wurzelverzeichnis ist eine Ausnahme → Länge steht im BSC
- Sonderfälle für das 1. Zeichen im Namensfeld:
  - 00h: letzter Eintrag im Verzeichnis (ungültig)
  - 2E: aktuelles Verzeichnis („.’)
  - E5h: Eintrag wurde gelöscht
- Sonderfall für die ersten 2 Zeichen:
  - 2E2E = übergeordnetes Verzeichnis („..“)

Bemerkungen

- Keine Schutzmechanismen
- Aber FAT-Systeme unter fast allen Betriebssystemen zugänglich

- Ursprünglich nur 8+3 Dateinamen (ohne VFAT)
- Beschränkung der Dateigröße auf 4 GB

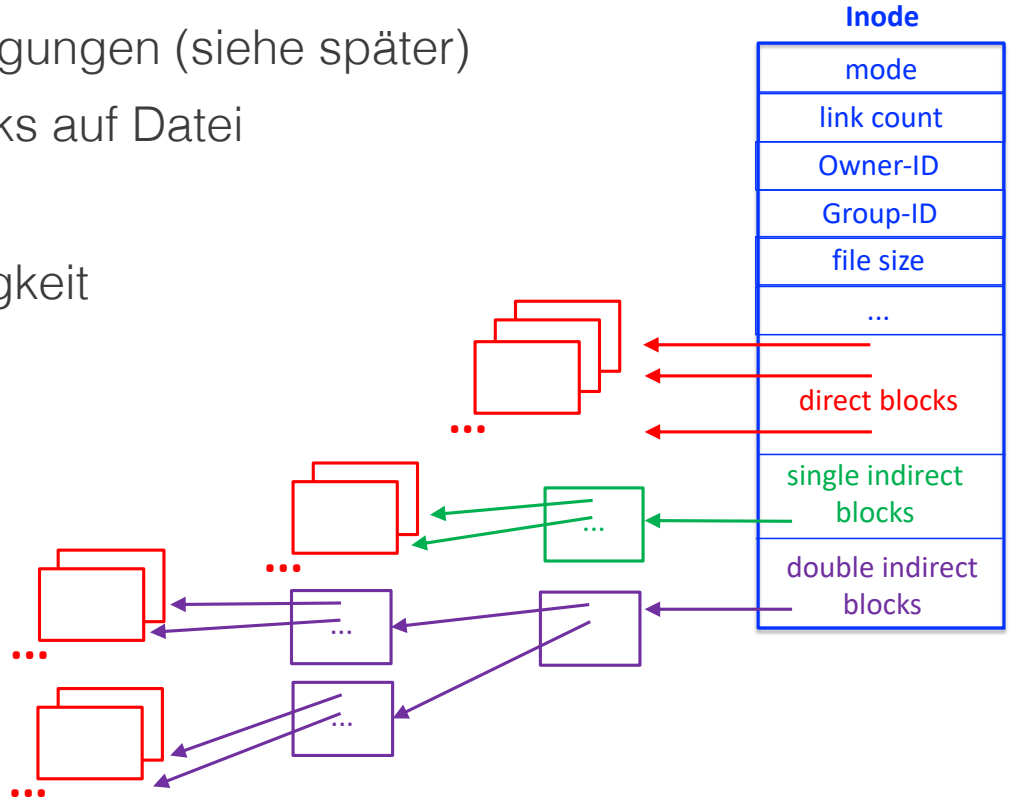
- Achtung: FAT enthält keine Namen, sondern nur Clusterverkettung

11.4 UNIX System V Dateisystem

- **Inodes** sind die zentrale Struktur in vielen UNIX-Dateisystemen
  - Inodes speichern Meta-Daten
  - Auch Verwaltungsblock genannt
- **Dateitypen**:
  - Verzeichnis
  - Reguläre Datei
  - Spezialdateien in `/proc`
    - Inodes nur im Speicher ohne Datenblöcke
    - Schnittstelle zum Kern → Daten werden beim Auslesen erzeugt
  - Spezialdateien in `/dev` für Geräte (block, character, sockets, pipes)
    - Inodes, aber keine Datenblöcke
    - Inode beinhaltet Referenz (ID) auf Device-Driver

Inhalt eines Inode

- Mode: Dateityp und Berechtigungen (siehe später)
- Link count: Anzahl harter Links auf Datei
- Owner-ID: Eigentümer-ID
- Group-ID: Gruppenzugehörigkeit
- File size: Dateigröße in Bytes
- Datumseinträge, ...
- Zeiger auf Daten- und weitere Zeigerblöcke



Zugriffsrechte (gespeichert im Inode)

- Eigentümer (engl. owner): durch eindeutige Nummer (UID) repräsentiert
- Gruppe (engl. group): GID ordnet eine Datei einer Gruppe zu
  - Ein Benutzer kann einer oder mehreren Benutzergruppen angehören, die durch eine eindeutige Gruppen-Nummer (GID) repräsentiert werden.
- Pro Datei werden Zugriffsbits (r = read, w = write, x = execute) gespeichert, jeweils für den Eigentümer, die Gruppe und Andere (engl. other).
- Bedeutung der Zugriffsbits bei Verzeichnissen:
  - `r`: Inhalt darf aufgelistet werden, x: mit cd darf in das Verz. gewechselt werden,
  - `x+w`: neue Dateien dürfen im Verzeichnis angelegt werden.

Spezialbit: SUID

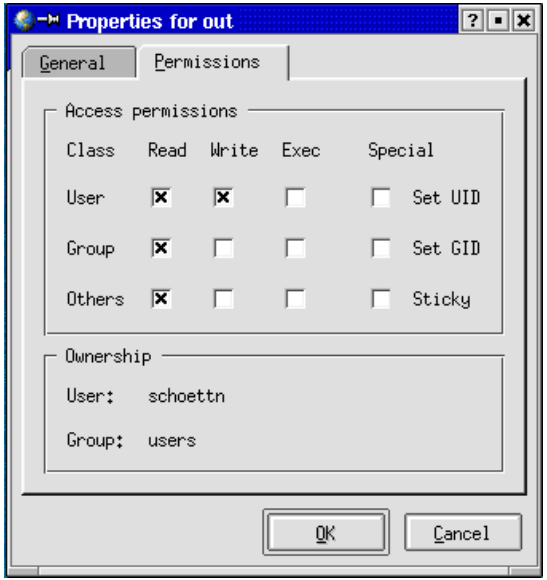
- Problem: User sollten gelegentlich ihr Passwort ändern, dürfen aber nicht auf die `/etc/passwd` Datei zugreifen (nur root ist berechtigt).
- Lösungsansatz:
  - Passwortprogramm `/bin/passwd` gehört root und hat **Setuid-Bit** gesetzt
  - **Wird es vom User gerufen, so läuft es nicht etwa mit der Zugriffsberechtigung des Users, sondern mit derjenigen des Eigentümers (z.B. root)**
  - Nur der Eigentümer kann das Setuid-Bit setzen (und immer auch root)
  - Bem.: Es geht nicht um die Ausführungsberechtigung von `/bin/passwd` sondern um den Zugriff auf `/etc/passwd`
  - (Verfahren war einmal patentiert durch Dennis Ritchie)
- `ls –l` zeigt bei den **Besitzer-Zugriffsbits ein `s` (statt eines `x`) an.**

Spezialbit: SGID

- Bei der Ausführung des Programms wird immer die GID der Datei verwendet (und nicht die GID des aktuellen Benutzers).
- `ls –l` zeigt bei den Gruppen-Zugriffsbits ein **`s`** (statt eines **`x`**) an.
- Bei Verzeichnissen bewirkt dieses Bit, dass neu angelegte Dateien der Gruppe des Verzeichnisses angehören (nicht der Gruppe des Benutzers).

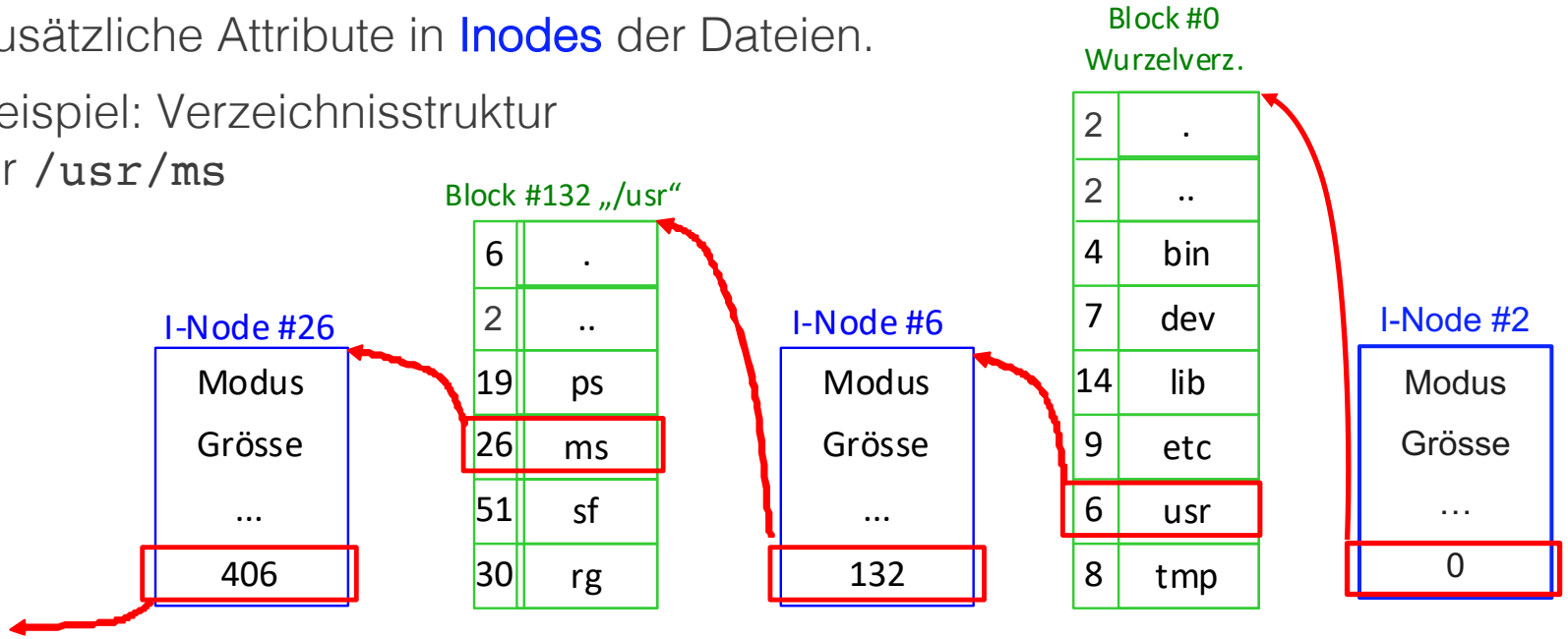
Spezialbit: Sticky

- Für gemeinsame öffentliche Verzeichnisse.
  - Zum Beispiel für `/tmp`.
  - `ls –l` zeigt bei allen gültigen Zugriffsbits ein `t` (statt eines `x`) an.
- Sind alle Berechtigungen gesetzt (rwxrwxrwx), so kann jeder alle Dateien ändern/löschen.
- Durch setzen des Sticky Bits kann nur der Eigentümer oder root Dateien ändern/löschen.



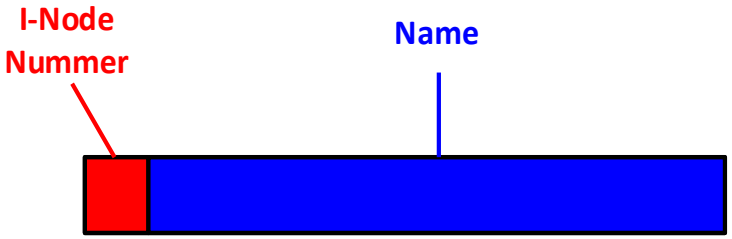
Verzeichnisse

- Implementiert als Dateien, Kennzeichnung über Typfeld im Inode
- Datenblöcke: Dateinnamen und Inode-Nummer (z.B. 16 Bit).
- Zusätzliche Attribute in Inodes der Dateien.
- Beispiel: Verzeichnisstruktur für `/usr/ms`

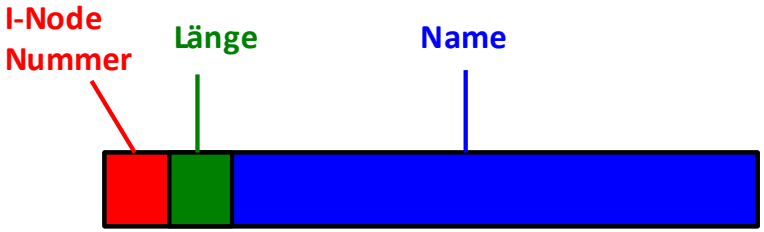


Dateinamen

- Feste Länge: z.B. 14 Zeichen bei UNIX System V Release 3:



- Variable Länge: z.B. 255 Zeichen bei BSD 4.2



Harte Links

- Dateien
  - Jede benannte Datei ist ein Hard-Link (verweist auf einen Inode)
  - Wird ein Hard-Link auf eine Datei angelegt, so entsteht eine neue Datei die auf den gleichen Inode verweist. Im Prinzip hat eine Datei dann zwei Namen
  - Die Daten einer Datei werden erst gelöscht, wenn kein Hard-Link auf den Inode mehr verweist.
    - Dann ist link-count=0 im Inode der Datei
    - Deswegen heißt der System-Aufruf zum Löschen von Dateien `unlink`
- Beispiel:

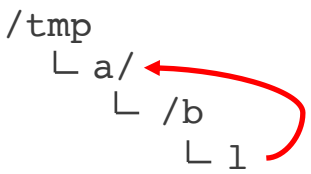
Gleicher Inode

```
$ls -li
8647678067 -rw-r--r--  1  mschoett1  staff    0 Dec  4 19:23 helper.c
$ ln helper.c mylink
$ls -li
8647678067 -rw-r--r--  2  mschoett1  staff    0 Dec  4 19:23 helper.c
8647678067 -rw-r--r--  2  mschoett1  staff    0 Dec  4 19:23 mylink
```

Harte Links (2)

- Verzeichnisse
  - Ein leeres Verzeichnis hat einen Hart-Link-Count von 2
    - Erster Hardlink: Verweis auf sich selbst → `.`
    - Zweiter Hardlink: Verweis vom Elternverzeichnis auf das neu erzeugte Verzeichnis
  - Jedes Unterverzeichnis im Verzeichnis erhöht den Hardlink-Count um 1, da das Unterverzeichnis einen Verweis zum Parent-Verzeichnis hat → `..`
  - Ein Hard-Link darf nicht auf ein Verzeichnis zeigen
    - Es könnten dadurch Zyklen im Verzeichnis-Baum entstehen (im Prinzip ein Verzeichnis mit unendlicher Tiefe)
    - Beispiel (nicht erlaubt):

```
mkdir -p /tmp/a/b
cd /tmp/a/b
ln -d /tmp/a l
```



Beispiel: Link-Count bei Verzeichnissen

- Beispiel: `ls -l`

```
Type+rights #L Owner   Group Size Date          Name
drwxr-xr-x   3 ms      staff  170 Jun  5 21:41 test
```

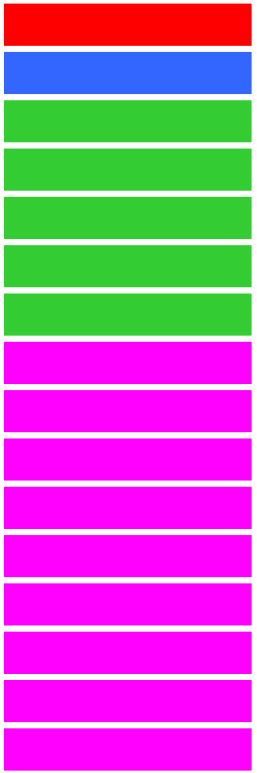
- Beispiel: `ls –la test`

```
Type+rights #L Owner    Group   Size Date          Name
drwxr-xr-x   3 mschoett staff    170 Jun  5 21:41 .
drwxr-xr-x+ 39 mschoett staff   1326 Jun  5 21:14 ..
drwxr-xr-x   2 mschoett staff    102 Jun  5 21:20 obj
-rw-r--r--   1 mschoett staff      9 Jun  5 21:14 hello.c
-rw-r--r--   1 mschoett staff      9 Jun  5 21:41 helper.c
```

Enthält kein Unterverzeichnis, aber evt. keine, eine oder mehrere Dateien

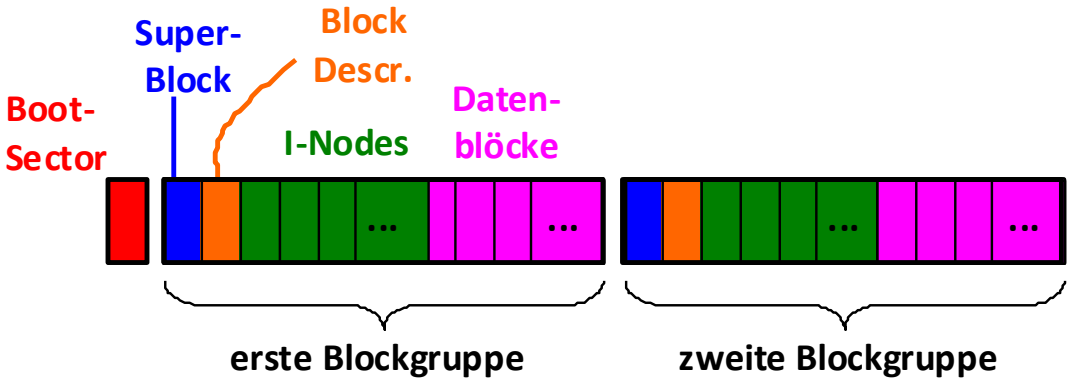
Partitionsstruktur

- Boot-Sector: Bootprogramm.
- Super-Block: Verwaltungsinformation für Dateisystem
  - Anzahl der Blöcke, Anzahl der Inodes
  - Anzahl freier Blöcke und freier Inodes
  - Anker der Listen freier Blöcke und freier Inodes ...
  - Super Block wird im Hauptspeicher gehalten
- Inodes:
  - Werden beim Formatieren fortlaufen angelegt
  - 16-Bit Adressierung → max. 65536 Inodes
- Datenblöcke: folgen im Anschluss an die Inodes



11.5 Linux EXT2-Dateisystem

- 32-Bit Adressierung; Lange Dateinamen mit max. 255 Zeichen.
- Super-Block: wichtige globale Informationen: Anzahl Blöcke pro Gruppe, Blockgröße, …
- Blockgruppen: Ziel ist es den Abstand zwischen Datenblöcke und Inodes klein halten → Pfad-Auflösung schneller



11.5 Linux EXT2-Dateisystem

- Block-Descriptor: Metadaten für eine Block-Gruppe
  - Anzahl freier Blöcke und freier Inodes
  - Bitmaps für frei Datenblöcke & Inodes
    - Bitmaps belegen jeweils einen Block
  - Beispiel-Konfiguration: 1 KB Blockgröße;
    1024 Blöcke pro Gruppe verwaltbar → ergibt 1 MB pro Gruppe

- Fehlertoleranz durch Replikation:
  - Super-Block in jeder Blockgruppe repliziert
  - Blockdeskriptoren ebenfalls in jeder Blockgruppe repliziert,
    jedoch ohne Bitmaps

11.6 Linux EXT3-Dateisystem

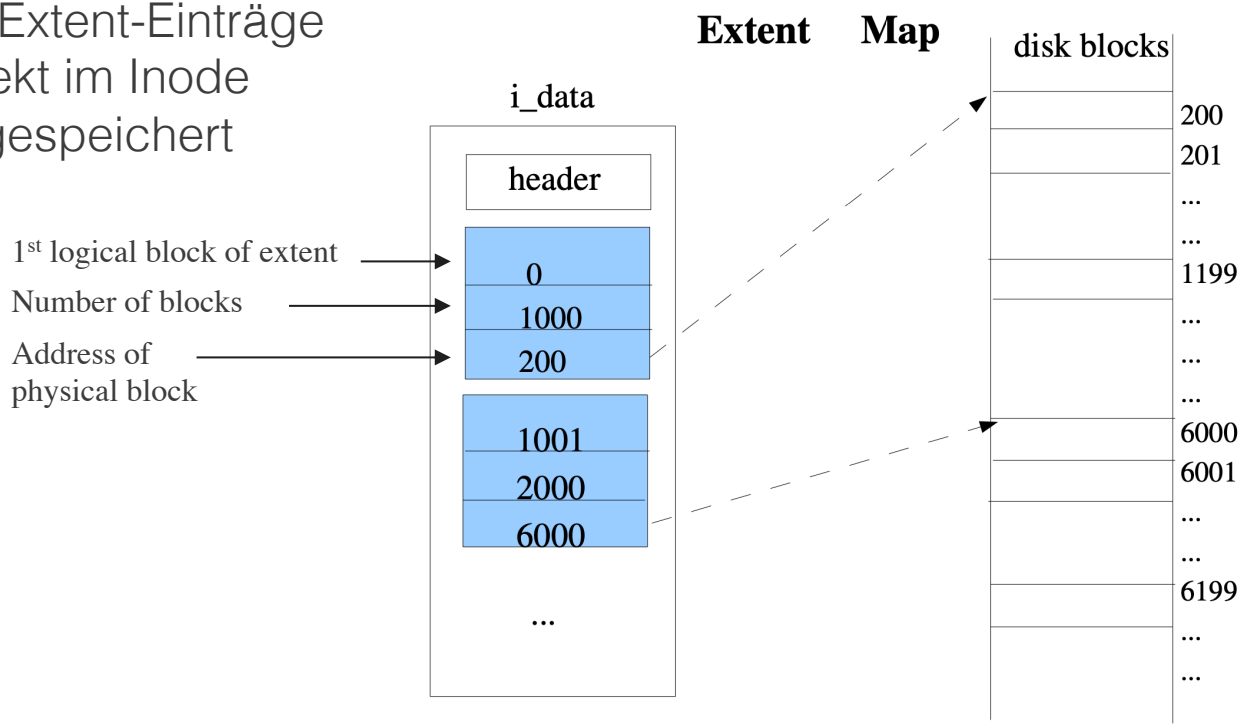
- EXT3 verwendet intern die gleichen Datenstrukturen wie EXT2
  - 32-Bit Blockadressierung (wie EXT2)
  - 100% abwärtskompatibel
- Wesentlicher Unterschied ist, dass EXT3 ein Journaling-Dateisystem ist
  - Veränderungen an Metadaten und werden protokolliert
  - **writeback-Mode**: nur Veränderungen an den Metadaten werden protokolliert
  - **ordered-Mode**: Dateiänderungen werden direkt im Zuge der Protokollierung der Änderungen an den Metadaten geschrieben → sehr langsam

11.7 Linux EXT4-Dateisystem

- Kompatibel zu EXT3
- 48-Bit für Blockadressierung
- Größere Anzahl von Unterverzeichnisse möglich
- Zusätzlich gibt es Extents, ergänzend zu mehrfach indirekten Zeigern
  - Nützlich bei großen Dateien
  - Extent definiert durch Startadresse + Länge → Menge fortlaufender Blöcke

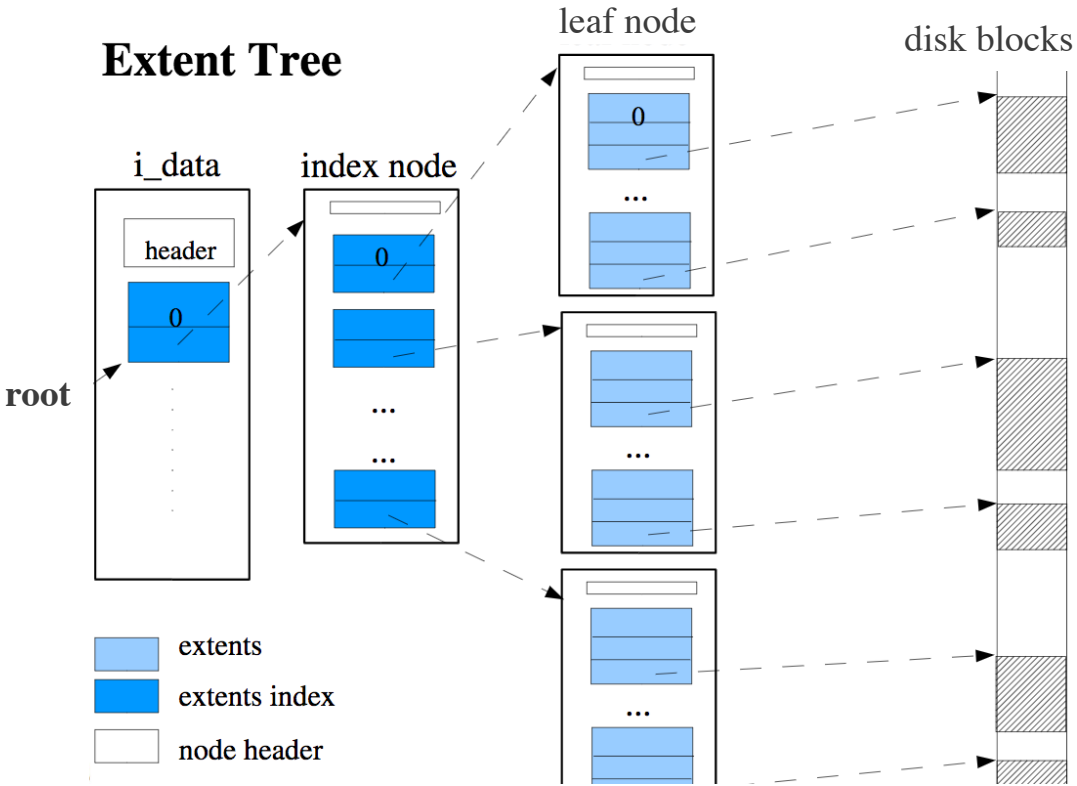
11.7 Linux EXT4-Dateisystem

- Bis zu drei Extent-Einträge werden direkt im Inode (= i_data) gespeichert



11.7 Linux EXT4-Dateisystem

- Bei mehr als drei Extents erfolgt die Auslagerung der Extent-Deskriptoren in Datenblöcke. Diese werden als B+ Baum organisiert.



https://www.usenix.org/legacy/event/lsf07/tech/cao_m.pdf

Ext4: The Next Generation of Ext2/3 Filesystem, USENIX, 2007.

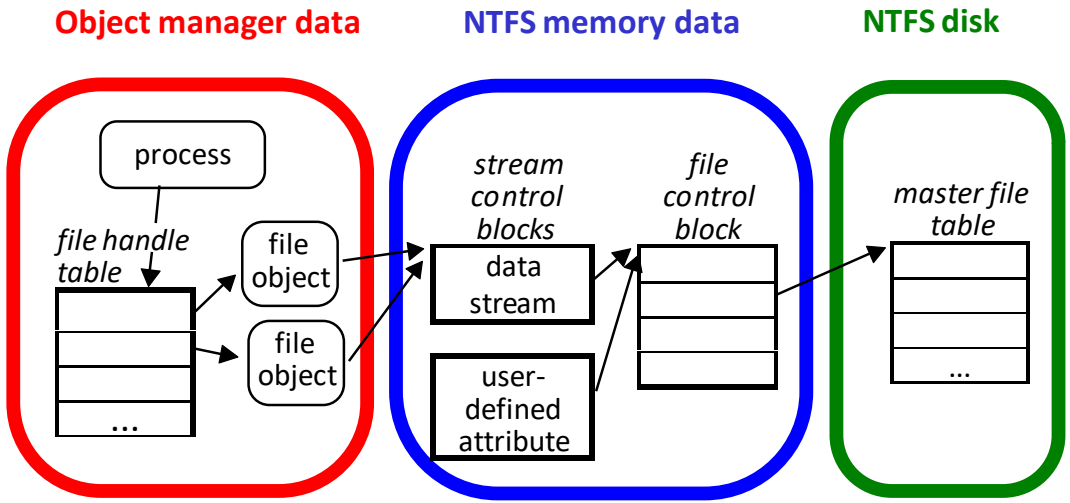
Ende Teile 1

11.8 NTFS = New Technology File System

- Weiterentwicklung des HPFS (OS/2), seit XP NTFS Version 3.1.
- Alle Dateiinformationen werden als Datei gespeichert, auch die Metadaten.
- 64-Bit Adressierung für große Dateien.
  - Kleine Blockgröße für große Partitionen möglich
- Optional: Kompression & mehrere Datenströme pro Datei.
- Sicherheit:
  - Zugriffskontrolle pro Benutzer oder Gruppe, optionale Verschlüsselung
- Fehlertoleranz:
  - NTFS ist ein Journaling File-System (wie auch EXT3, ReiserFS …)
  - Transaktionen mit Logging für Meta-Daten,
  - RAID-Unterstützung

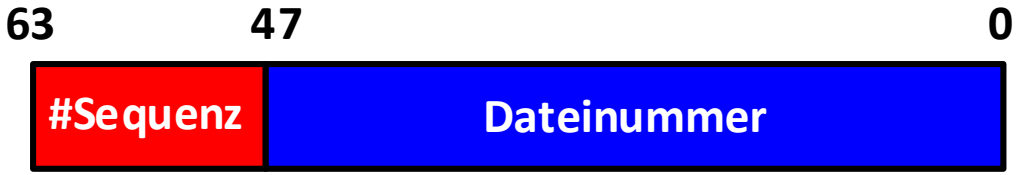
Datenstrukturen für eine Datei

- Eine Datei ist eventuell mehrfach geöffnet.
- Möglicherweise mehrere Ströme pro Datei.
- File control block hat Datei-Referenz in MFT



Dateiverwaltung

- Dateinamen werden in Verzeichnisdateien gehalten und sind von Datei-Referenzen zu unterscheiden.
- Datei-Referenz (engl. file reference):
  - bezeichnet eindeutig eine Datei bzw. ein Verzeichnis innerhalb eines Volumes
  - Sequenznummer:
    - wird hochgezählt, für jede neue Datei mit gleicher Dateinummer
    - dient zur Erkennung veralteter Dateireferenzen (Inkonsistenz)
  - Dateinummer = Index in Master File Table (MFT)



Aufbau eines NTFS Volumes (Partition)

- Unterteilung in Clusters (ein oder mehrere fortlaufenden Sektoren).
- BSC (Boot Sector):
  - Boot-Programm (optional)
  - #Sektoren pro Cluster, „Plattengeometrie“, ...
- MFT (Master File Table):
  - pro Datei/Verzeichnis ein Eintrag und ein Cluster
  - MFT ist selbst auch eine Datei
    - 12,5 % der Partition werden dafür reserviert
    - ggf. später erweitern → Fragmentierung der MFT
  - Entspricht ungefähr der Inode-Tabelle in EXT

Aufbau eines NTFS Volumes (Partition)

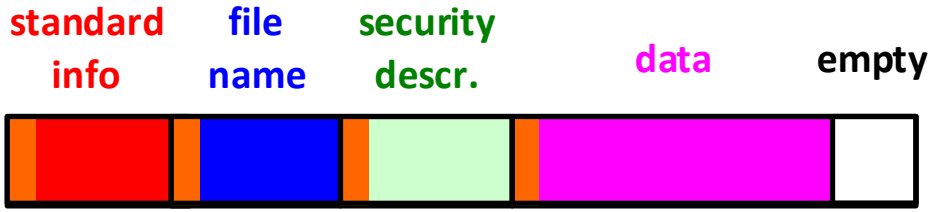
- System files:
  - Log-Datei: Änderungen an MFT protokollieren
  - Volume-Datei: Größe & Name des Volumes, Versionsnummer der Partition
  - Bitmap-Datei: freie & belegte Cluster
  - Boot-Datei:
    - Größe eines Clusters
    - und eines MFT-Eintrags (1-4 KB)
    - sowie Boot-Code (Kopie von Partitionsboot-Sektor)
  - Quota-Tabelle: Speicherplatz-Kontingent pro Benutzer
  - Bad-Cluster-Datei: vermerkt defekte Cluster

Attribute

- Jede Datei bzw. Verzeichnis ist Satz von Attributen
- Alle Elemente einer Datei, wie der Name, SicherheitsID,
  aber auch Daten sind Attribute
- Jedes Attribut wird durch einen Code für den Attributtyp identifiziert

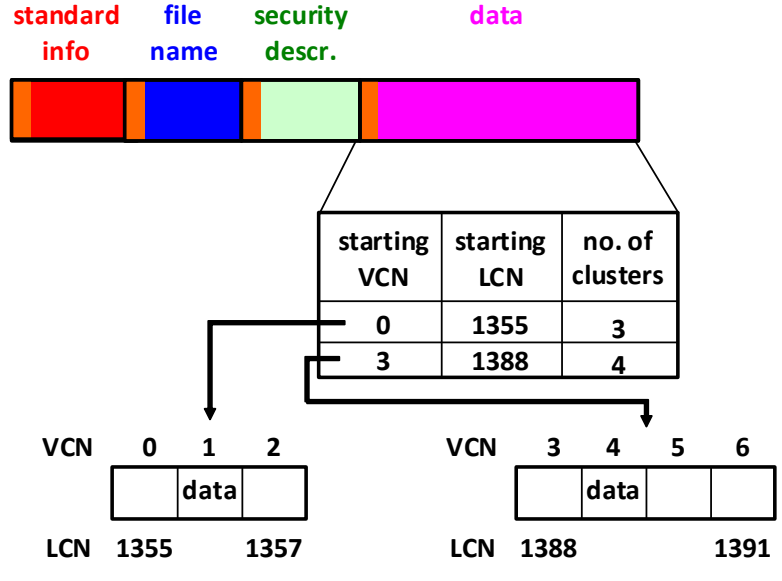
MFT-Eintrag für eine kurze Datei

- attribut code
- standard info:
  - Dateilänge
  - MS-DOS Attribute
  - Anzahl der Hard-Links
  - Zeitstempel für Zugriffe
  - Sequenznr. der gültigen File-Reference
- security descriptor: Zugriffskontrolle.
- data der Datei direkt in der MFT abgelegt.
- file name: in Datei-Eintrag & Verzeichnis (im Gegensatz zu UNIX).



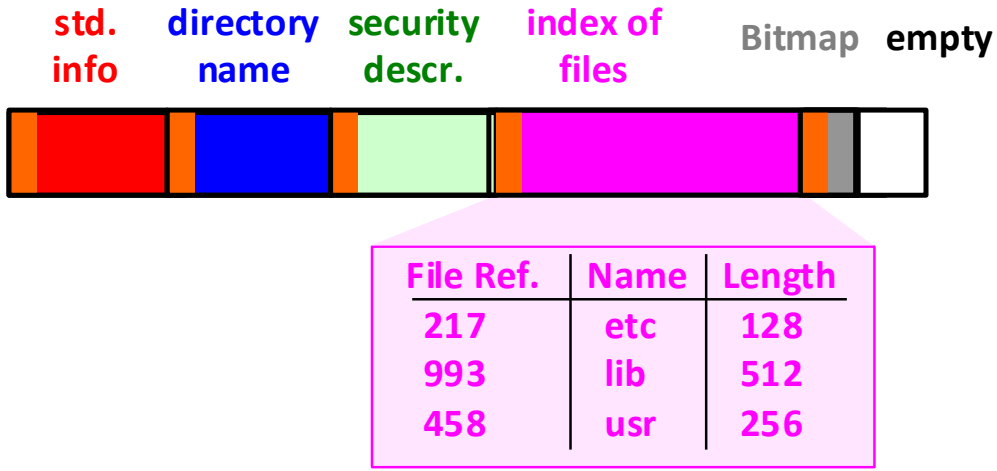
MFT-Eintrag für eine lange Datei

- Große Dateien außerhalb der MFT in sogenannten **runs** / **extents** abgelegt.
- Datenbereich des MFT-Eintrags enthält nur Zeiger.
- Falls ein MFT-Eintrag nicht genügt, werden weitere alloziert.
- Adressierung:
  - virtuelle Clusternummer (VCN): Adressierung innerhalb einer Datei
  - logische Clusternummer (LCN): Adressierung innerhalb der Partition



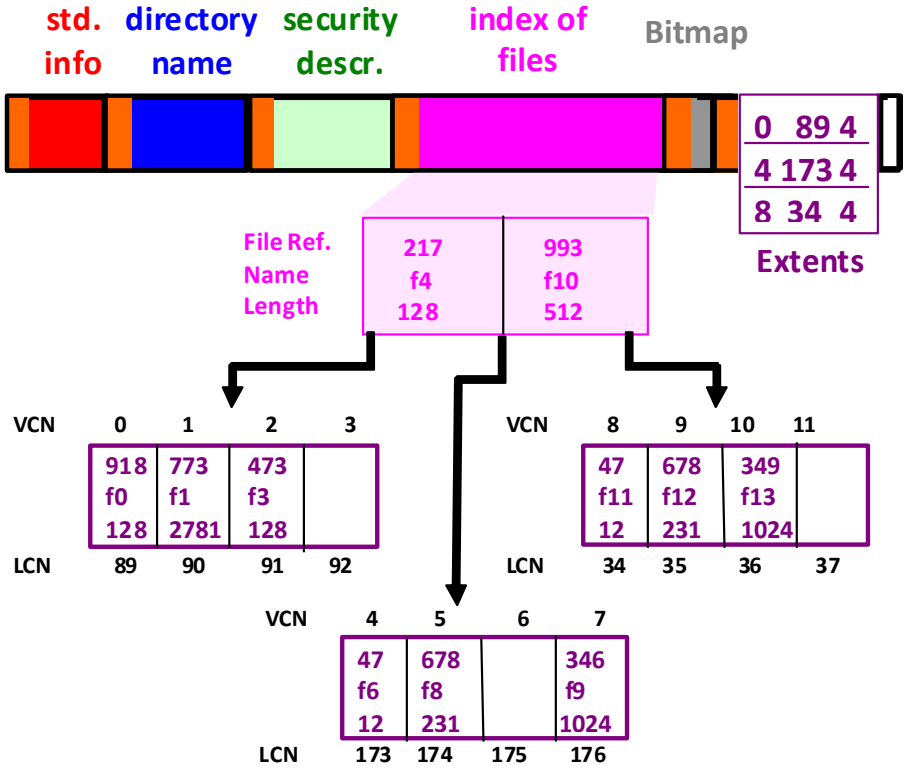
Aufbau eines kurzen Verzeichnisses

- Index of Files: Sammlung von Referenzen auf Dateien
  - File-Reference, Datei-Name und Länge der Datei
- Sequentielle Suche in kleinen Verz. (die in MFT-Eintrag Platz finden).
- Achtung: Dateinamen im Verzeichnis repliziert.
- Bitmap: Belegung von index of files.



Aufbau eines langen Verzeichnisses

- Pro Verzeichnis ein B-Baum indiziert nach Namen der Dateien.
  - Aber kein B-Baum über alle Dateien eines Volumes.
  - Extents: zeigen, wo sich weitere index of files
    - Inhalt eines Eintrags: VCNstart | LCNstart | #Clusters
      - VCN: Adressierung in der Datei
      - LCN: Adressierung in d. Partition
  - Bitmap: Speicherverwaltung für index of files



11.9 Dateisysteme mit Fehlererholung

- Inkonsistente Metadaten durch z. B. Absturz:
  - Verzeichniseintrag fehlt zur Datei oder umgekehrt
  - Block ist benutzt aber nicht als belegt markiert
- Reparaturprogramme:
  - Programme wie **`chkdsk`**, **`scandisk`** oder **`fsck`** können u. U. inkonsistente Metadaten reparieren
  - Datenverluste bei Reparatur sind jedoch wahrscheinlich & u.U. lange Laufzeit
- Lösung: Journaling-Dateisysteme

Journal / Protokoll / Log

- Schreibzugriffe auf Metadaten werden in Transaktionen (TAs) gekapselt
  - Keine Datenbank-Transaktion
  - Aber es gilt Atomarität → alle Operationen einer TA durchführen oder keine
- Buchführung über Modifikationen erfolgen in einer Protokolldatei (log file)
- Protokollierung erfolgt immer vor der Durchführung
  - Redo-Log: TAs wiederholen/abschließen
  - Undo-Log: TAs rückgängig machen
- Im Recovery-Fall wird Protokolldatei mit den aktuellen Änderungen abgeglichen

Journal / Protokoll / Log

- Checkpoints:
  - Log-File kann nicht beliebig groß werden
  - Gelegentlich wird für einen konsistenten Zustand (Checkpoint) auf Platte gesorgt
  - Alle Protokolleinträge vor diesem Checkpoint können gelöscht/überschrieben werden
  - Außerdem wird hierdurch Recovery schneller
- Bewertung:
  - Metadaten immer konsistent
  - Datenverlust aber weiter möglich
  - Transaktionen und Logging verursachen Overhead
  - Logging erfolgt deshalb auch über den Disk-Cache
- Beispiele: NTFS, EXT3, EXT4

Fallstudie: Journaling in NTFS

- Logdatei:
  - Restart-Bereich: zeigt an, wo im Fehlerfall Recovery beginnt (repliziert)
  - Logging-Bereich: für Protokollierung (feste Dateigröße → zirkuläres Schreiben)

- Einträge im Logging-Bereich:
  - nummeriert mit Logical Sequence Number (LSN)
  - Alle Einträge einer Transaktion (TA) sind rückwärts-verkettet
  - TA-Beispiele: Create/DeleteAttribute, Checkpoint-Record, PrepareTA, Commit, ...
  - Update-Eintrag: besteht aus idempotenter Undo- und Redo-Operation
    - Wiederholtes Ausführen einer Operation führt immer zum gleichem Ergebnis
    - Beispiele: undo: Cluster freigeben, redo: Cluster belegen

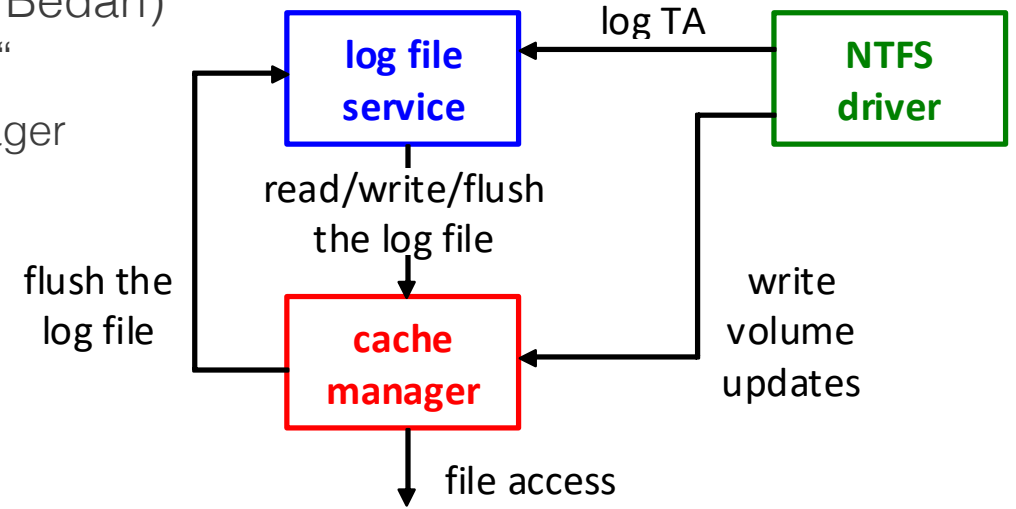
Fallstudie: Journaling in NTFS

- Im Fehlerfall verwendet NTFS die Logdatei:
  - Abgeschlossen TAs werden wiederholt → redo
  - Nicht vollständige TAs werden zurückgesetzt → undo

- Logfile-Service (LFS) schreibt in die Logdatei
  - NTFS greift immer über LFS auf die Logdatei zu
  - Zugriffe auf Logdatei und Dateien erfolgen immer über Cache-Manager → Arbeiten im Cache schneller und Logging wird gebündelt

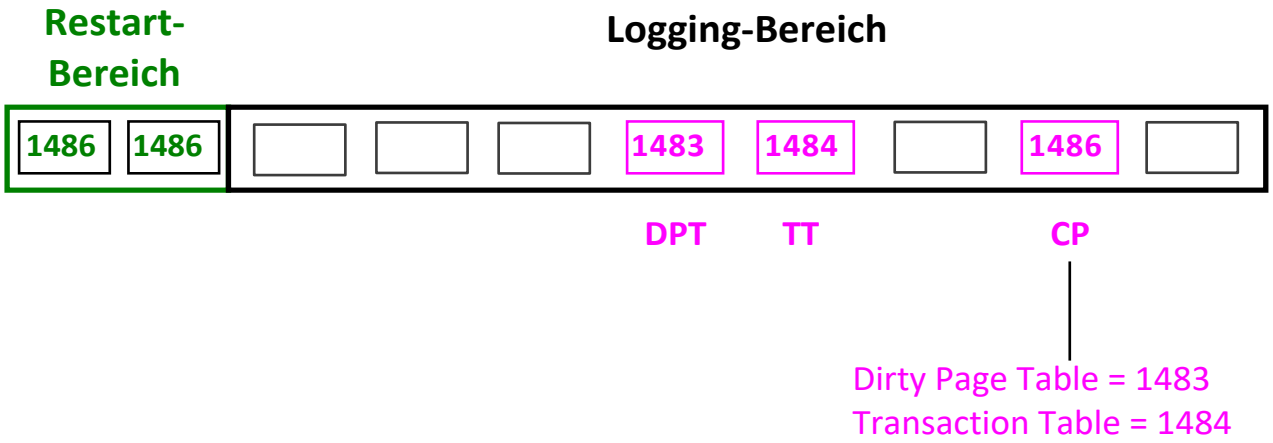
Ablauf der Protokollierung von Metadaten-Änderungen

- NTFS schreibt über LFS alle Transaktionen die Metadaten modifizieren in das gecachte Log-File
- NTFS modifiziert das Volume ebenfalls im Cache
- Cache-Manager fordert (bei Bedarf) LSF auf das Log zu „flushen“
  - LSF teilt dem Cache-Manager mit, welche Seiten er durchschreiben soll
- Erst danach schreibt der Cache-Manager die Volume-Änderungen auch auf Disk



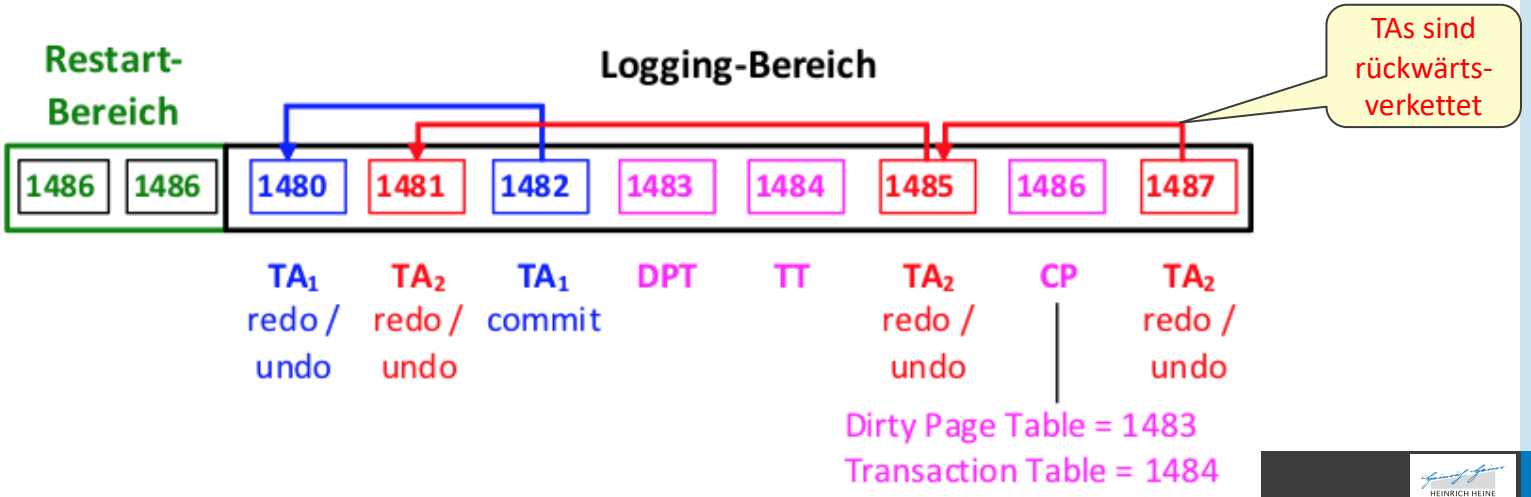
Schreiben eines Checkpoints

- NTFS schreibt alle 5 Sek. einen Checkpoint:
  - LSN des letzten Checkpoints wird im Restartbereich abgespeichert.
- Ferner werden zwei Tabellen gesichert, die LFS im RAM verwaltet:
  - dirty page table (DPT): LSNs zu Metadaten im Cache (noch nicht gespeichert)
  - transaction table (TT): nicht abgeschlossene Transaktionen (letzte LSN einer TA)

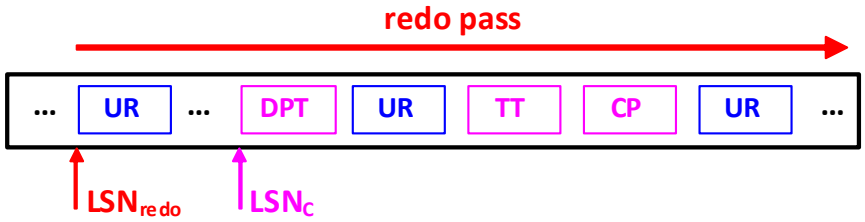


Schreiben eines Checkpoints

- Checkpoint speichert LSNs beider Tabellen (DPT und TT)
  - Alle Einträge mit LSN < LSN_{min} = min (LSN_{min}(DPT), LSN_{min}(TT)) sind nun frei
  - Checkpoint erzwingt weder Abschluss ausstehender TAs noch das Durchschreiben von Log-Daten
  - Zwischen den Einträgen eines Checkpoints können weitere Transaktionen eingestreut sein, wie im Beispiel

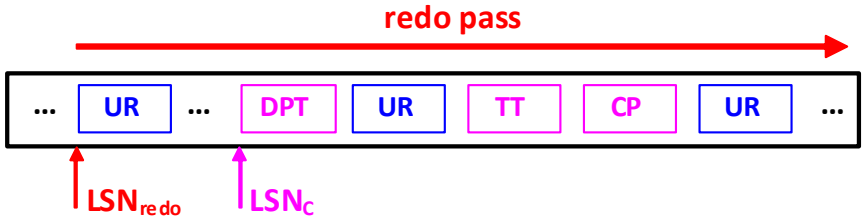


Recovery



- Initialisierung (0):
  - Letzter Checkpoint aus Restart-Bereich
  - Laden der zugehörigen transaction table (TT) und die dirty page table (DPT)
- Analyse Phase (1):
  - ab erstem Checkpoint-Record (LSN_C), also beim DPT-Record aufsteigend arbeiten
  - Update-Records mit LSN > LSN_C verwenden, um gesicherte Tabellen zu aktualisieren
  - Wird ein Commit gefunden, so muss TA aus der TT entfernt werden
  - Werden Metadaten modifiziert, so muss die LSN in der DPT vermerkt werden
  - Wenn beide Tab. aktuell sind, so sucht NTFS in beiden Tabellen die kleinste LSN = LSN_{min}
  - Redo-Phase beginnt bei LSN_{redo} = LSN_{min}., außer wenn LSN_C < LSN_{min},, dann gilt LSN_{redo} = LSN_C

Recovery

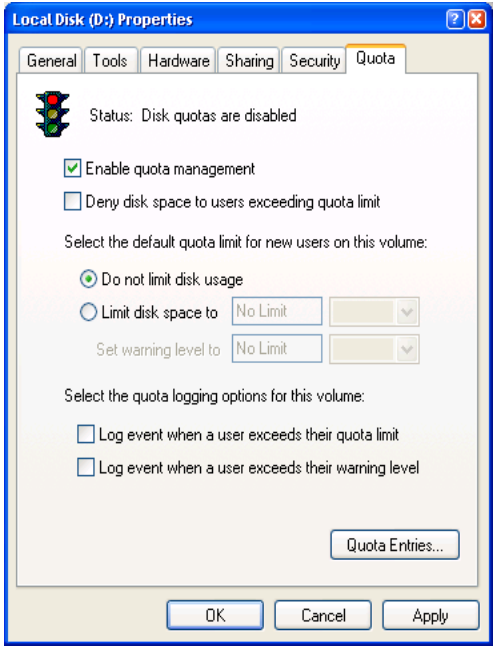


- Redo Phase (2):
  - Operationen ab LSN_{redo} wiederholen, Zugriffe wieder über Cache leiten

- Undo Phase (3):
  - nicht abgeschlossene TAs rückgängig machen
  - Einstieg ist jeweils Eintrag aus der TT (jeweils letzte LSN einer TA), dann rückwärts arbeiten
  - Abschließend werden Cache-Daten durchgeschrieben

11.10 Limitierung der Plattennutzung

- Mehrbenutzersysteme:
  - Einzelnen Benutzern sollen verschieden große Kontingente zur Verfügung stehen
  - Ziel: gegenseitige Beeinflussung vermeiden.
- Quota-Systeme:
  - Tabelle enthält maximale und augenblickliche Anzahl von Blöcken für Dateien und Verzeichnisse eines Benutzers
  - Dateisystem verwaltet Tabelle auf Disk
  - In der Regel weiche (können kurzfristig Überschritten werden) und harte Grenzen
  - „Disk-full“ Meldung, wenn Quota verbraucht
- Beispiele: NTFS, EXT3/4



11.11 Memory-Mapped Files

- (Teil einer) Datei wird in den virtuellen Adreßraum eingeblendet.
- Speicherzugriffe auf diesen Bereich werden wie Dateizugriffe behandelt.
- Implementierung:
  - Logischen Adressraum reservieren; Seitentabellendeskr. zeigen auf Disk-Blöcke
  - Bei einem Seitenfehler wird von Disk geladen
  - Zurückschreiben erfolgt beim Schließen des Mappings

11.11 Memory-Mapped Files

- Beispiel:

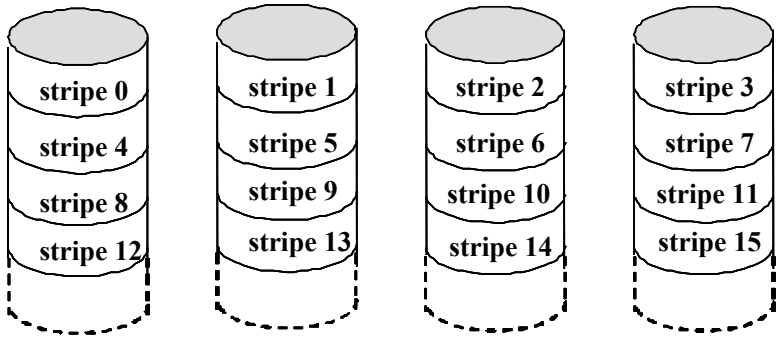
```
#include <unistd.h>
#include <stdio.h>
#include <fcntl.h>
#include <sys/mman.h>

int main() {
   char  *p;
   int   fd;

   fd = open("txt", O_RDWR);
   p = mmap(0, getpagesize(), PROT_READ|PROT_WRITE, MAP_PRIVATE, fd, 0);
   printf("%08X\n", p);
   munmap(p, getpagesize());
   close ( fd );
   return 0;
}
```

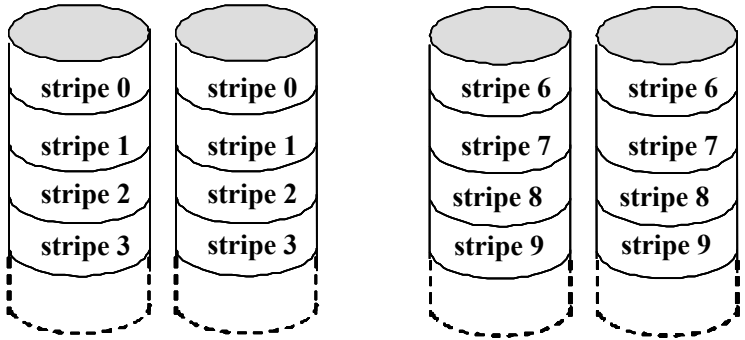
11.12 Redundanz

- RAID = Redundant Array of Inexpensive Disks:
  - Daten auf mehrere Disks verteilen und replizieren
  - mehrere Levels definiert von Universität Berkeley
  - Software- und Hardware-Lösungen vorhanden
- RAID Level 0:
  - keine Redundanz der Daten
  - Aber hohe Lese- und Schreibleistung
  - Datenblöcke (Stripes) auf mehrere Disks verteilt



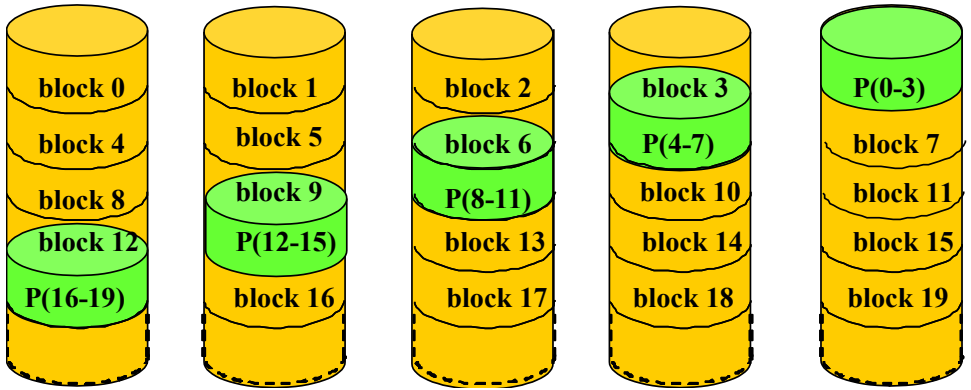
11.12 Redundanz

- RAID Level 1:
  - benötigt zwei Festplatten
  - wird auch als „Spiegeln“ bezeichnet
  - Gesamtkapazität entspricht einer Platte
  - Schreiboperationen immer auf beiden Disks



11.12 Redundanz

- RAID Level 5:
  - Platten werden nicht mehr komplett gespiegelt
  - → Festplattenkapazität wird besser ausgenutzt
  - Redundanz durch Parity/FEC (XOR)
  - Paritätsinfo verteilt über alle Disks



11.13 Hörsaal-Aufgabe

- Schreiben Sie ein C-Programm, welches ein Pfad und Datei-Namen als Argument übergeben bekommt
- Das Programm soll dann nach der Datei suchen, auch in allen Unterverzeichnissen im angegebenen Pfad und ausgeben ob, und wo die Datei gefunden wurde.
- Hinweis: folgende System-Aufrufe sind für die Aufgabe wichtig
  - `opendir`: opens a directory
  - `readdir`: returns a pointer to the next directory entry
  - `closedir`: closes the named directory stream