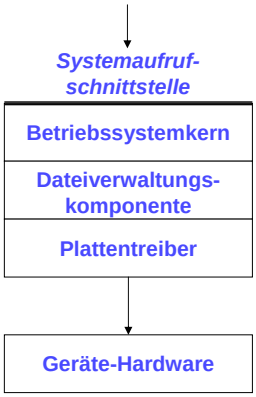
Kapitel 6

Dateiverwaltung

- FAT-Dateisystem (MS-DOS, Win95, Win98)
- NTFS-Dateisystem (NT, W2K, WinXP)
- Journaling Dateisysteme
- RAID
- Unix-Dateisystem
- CD-ROM Dateisystem
- Plattenzugriffsverwaltung

Dateiverwaltung



- **Begriffe**
  - Datei oder File - benannte logischer Abschnitt der Platte
  - Dateisystem - Gesamtheit der Dateien
  - Directories, Subdirectories, root, working directory
  - flache vs. hierarchische Dateisysteme
- **Plattenbuchhaltung**
  - Merken der physikalischen Lage einer benannten Datei
  - Umsetzen von Schreib- und Leseaufträge in physikalisch adressierte Transferaufträge
- **Von der Plattenbuchhaltung benötigte Daten**
  - freie Allokationseinheiten (z. B. Allokationsvektor)
  - Verzeichnis(se) der auf der Platte vorhandenen Dateien
  - für jede Datei die belegten Allokationseinheiten
  - ggf. Informationen über die Plattenpartitionierung
  - ggf. Dateiattribute und Eigentums- und Zugriffsrechte

Der interne Aufbau des FAT-Datei-Systems

- **Plattenaufteilung**
  - Partitionssektor
  - Urlade-Bereich (Boot sector)
  - FAT 1 (File Allocation Table), FAT 2 (Sicherheitskopie von FAT 1)
  - Stamm-Katalog (\)
  - Nutzdatenbereich (inklusive aller anderen Kataloge)
- **Aufbau der FAT**
  - 1½ Byte=12 Bit (~FAT12) bzw. 2 Byte (~FAT16)
  - FAT(i) = 0 --> frei, FAT(i) = FFF7H --> defekt, FAT(i) > FFF7H --> Dateiende, sonst durch Datei belegt
- **Directory-Einträge**
  - jeweils 32 Byte pro Eintrag

Aufbau der FAT-Directory Struktur

0								8			11				15
Name								Extension			Attr	reserviert ...			
16						22		24		26		28			31
... reserviert						Uhrzeit		Datum		FAT-Eintr.		Dateigröße			

1	R	*read only file*	schreibgeschützte Datei
2	H	*hidden file*	versteckte Datei
4	S	*system file*	System-Datei
8	V	*volume label*	Partitions- oder Datenträgernamen
0X10	D	*directory*	Katalogdatei
0X20	A	*archive bit*	Archiv-Bit (Datei wurde geändert)

Attribut-Flags

Auswirkungen von Dateioperationen auf die FAT

- **Belegungsreihenfolge: first-fit, rotating-first-fit, next-fit**

Clusterung / Clustergrößen

Größe der Partition	Sektoren pro Cluster	Clustergröße
0-31 MB	1	0.5K
-63 MB	2	1K
-127 MB	4	2K
-255 MB	8	4K
-511 MB	16	8K
-1023 MB	32	16K
-2047 MB	64	32K

- Durch den Overhead empfiehlt sich das Dateisystem FAT16 nicht für Datenträger größer als 511MB (sagt Microsoft)

Lange Dateinamen bei Win95 und NT

0x42	w	n	.	f	o	0x0F	0x00	Prüf.	x
0x0000	0xFFFF	0xFFFF	0xFFFF	0xFFFF	0x0000	0xFFFF	0xFFFF		
0x01	T	h	e		q	0x0F	0x00	Prüf.	u
i	c	k		b	0x0000	r	o		
T	H	E	Q	U	I	~	1	F	O	X	0x20	NT	Uhrzeit
Datum	Letzter Zu-griff	0x0000	Veränderung: Uhrzeit	Veränderung: Datum	erster Cluster	Dateigröße			

- **Verwendung von sekundären Ordnereinträgen**
- **Gekennzeichnet durch das Attribut 0x0F =**
  "Datenträgerbezeichnung", "Schreibgeschützt", "System" und "versteckt"
- Nummerierung: 01, 02, ... Ende+0x40

NTFS-Dateisystem

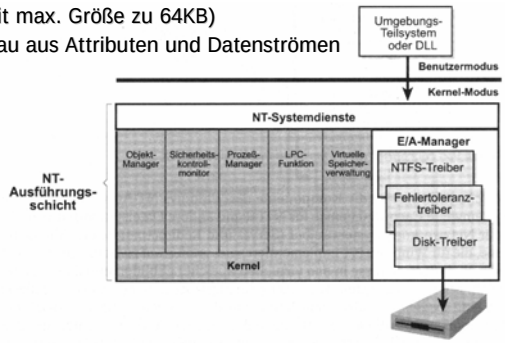
Dateisysteme FAT und NTFS im Vergleich

- **Vorteile des Dateisystems FAT**
  - Bessere Eignung bei kleinen Datenträgern (~500MB), da nur geringer Overhead

- **Vorteile des Dateisystems NTFS**
  - Zugriffsberechtigungen
  - Effiziente Verwaltung großer Datenträger bis 2 TBytes (derzeit), FAT16 nur bis 2 GB
  - Kleinere Clustergrößen durch 64 Bit-Adressen möglich --> geringerer Verschnitt
  - Wiederherstellbarkeit des Dateisystems
  - Einbindung eines Fehlertoleranztreibers möglich
  - Schnellerer Zugriff auf Dateien in großen Ordnern (B-Tree)
  - Kleine Dateien und Ordner können direkt im MFT-Eintrag abgelegt sein
    --> weniger Plattenzugriffe

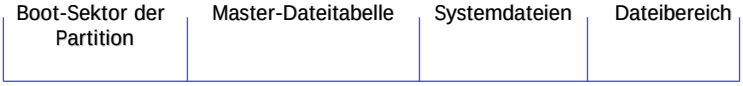
Windows NT Dateisystem (NTFS)

- **NTFS-Entwurfsziele:**
  - Wiederherstellbarkeit des Dateisystems
  - Sicherheit (Zugriffsschutz)
  - Datenredundanz und Fehlertoleranz
  - Große Festplatten (2^{64} Cluster mit max. Größe zu 64KB)
  - Multiple Datenströme: Dateiaufbau aus Attributen und Datenströmen
  - max. Dateilänge: 2^{64} Byte



Windows NT Dateisystem (NTFS)

- **Datenträger enthält nur Dateien**
  Selbst die Metadaten des Dateisystems sind Teil einer Datei
- **Dateien bestehen nur aus Attributen, z.B.**
  - Datenattribut
  - Sicherheitsattribut
  - Dateinamenattribut
- **Aufbau eines NTFS-Datenträgers**



NTFS-Systemdateien

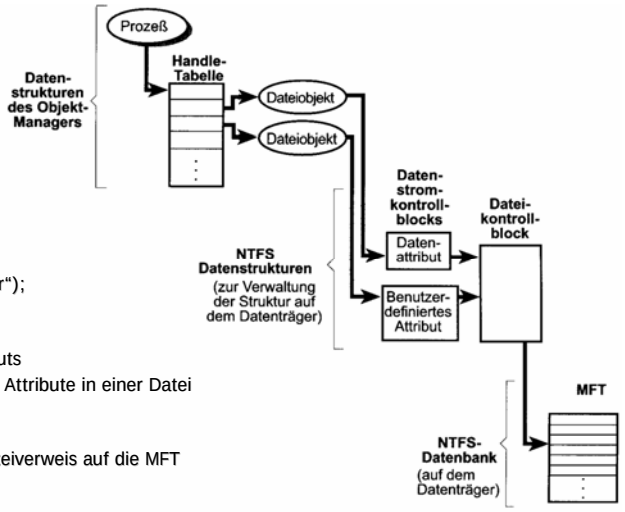
Systemdatei	D-Name	Nr	Zweck der Datei
Masterdateitabelle	$Mft	0	Auflistung des Inhalts des NTFS-Datenträgers
Master-Dateitabelle2	$MftMirr	1	Spiegelung der ersten drei Datensätze der MFT
Logdatei	$LogFile	2	Schritte für die Wiederherstellung von Dateien
Datenträger	$Volume	3	Name, NTFS-Version und andere Informationen
Attribut-Definitionen	$AttrDef	4	Tabelle mit Attributnamen, -ziffern und –beschreibungen
Stammindex	$.	5	Stammverzeichnis
Cluster-Bitmuster	$Bitmap	6	Zeigt, welche Zuordnungseinheiten belegt sind
Boot-Datei	$Boot	7	Enthält den Bootstrap des Boot-Datenträgers
Beschädigtr Cluster	$BadClus	8	Enthält die Orte beschädigter Cluster
Kontingent-Tabelle	$Quota	9	Auslastung der Festplatte für jeden Benutzer
Großbuchstaben-Tab.	$Upcase	10	Konvertierung der Kleinbuchstaben in die entsprechenden Großbuchstaben im Unicode
		11-15	Reserviert für zukünftige Verwendung

NTFS-Systemdateien

NTFS File Sector Information Utility (nfi)

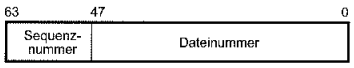
```
File 0
Master File Table ($Mft)
    $STANDARD_INFORMATION (resident)
    $FILE_NAME (resident)
    $DATA (nonresident)
        logical sectors 32-102207 (0x20-0x18f3f)
    $BITMAP (nonresident)
        logical sectors 16-23 (0x10-0x17)
        logical sectors 2048272-2048279 (0x1f4110-0x1f4117)

File 1
Master File Table Mirror ($MftMirr)
    $STANDARD_INFORMATION (resident)
    $FILE_NAME (resident)
    $DATA (nonresident)
        logical sectors 12289688-12289695 (0xbb8698-0xbb869f)

File 2
Log File ($LogFile)
    $STANDARD_INFORMATION (resident)
    $FILE_NAME (resident)
    $DATA (nonresident)
        logical sectors 12289696-12416703 (0xbb86a0-0xbd76bf)
```

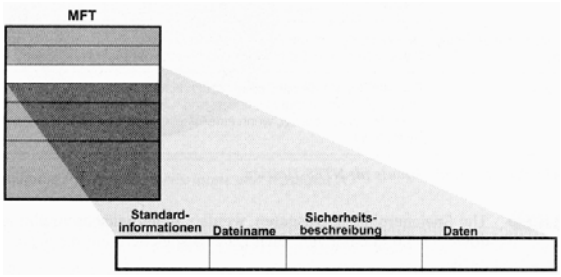

NTFS-Dateiobjekte



- **Dateiobjekte**
  - z.B. fp=fopen("test.dat", "r");
- **Datenstromkontrollblock**
  - Selektion eines Dateiattributs
  - Wie findet man bestimmte Attribute in einer Datei
- **Dateikontrollblock**
  - Datenstruktur, die den Dateiverweis auf die MFT enthält

Zuordnung MFT zur Datei





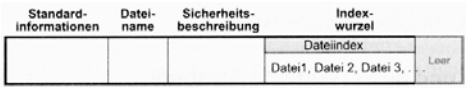
- **Eine Datei wird mit einem 64-Bit-Wert, dem sogenannten Dateiverweis identifiziert**
  - Dateinummer = Position der Datei in der MFT-1
  - Sequenznummer = fortlaufende Nummer bei Wiederverwendung von Dateinummern

Dateiattribute

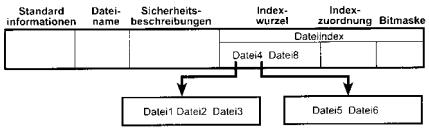
Attributtyp	Beschreibung
Standardinformationen	Zeitpunkt der letzten Speicherung, Bindungszähler, usw.
Attributliste	Orte aller Attributeinträge, wenn diese nicht alle in den MFT-Datensatz passen.
Dateiname	Wiederholbares Attribut sowohl für lange / kurze Dateinamen
Sicherheitsdeskriptor	Eigentums und Zugriffsrechte
Daten	Daten der Datei. NTFS erlaubt mehrere Datenattribute
Index-Wurzel	Wird für die Implementierung von Ordnern benötigt (B-Tree)
Index-Zuordnung	Wird für die Implementierung von Ordnern benötigt
Datenträgerinformation	Wird nur bei der Systemdatei des Datenträgers verwendet und enthält u.a. die Version und den Namen des Datenträgers
Bitmuster	Eine Karte der in der MFT oder im Ordner belegten Einträge
Erweiterte Attributinformation	Wird von Datei-Servern verwendet, die mit OS/2-Systemen verbunden sind
Erweiterte Attribute	dto.

Darstellung von Dateien und Directories

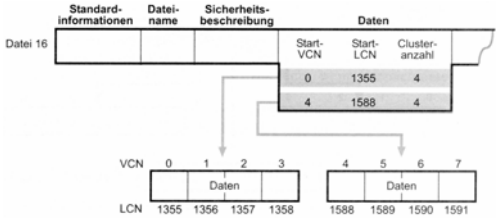
Aufbau eines kleinen Directories:



Aufbau eines großen Directories:

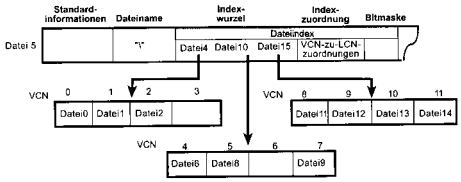


Aufbau einer großen Datei:



VCN zu LCN-Zuordnung

Beispiel: Root-Verzeichnis



Dateisystemwiederherstellung

Wiederherstellung des Dateisystems nach

- Stromausfall (vorzeitigem Abschalten)
- Systemfehlern (Absturz)

Dateisystemarten:

- **Careful-Write-Dateisysteme**
  - Schreiboperationen auf das Dateisystem werden so angeordnet, daß ein Systemabsturz im schlimmsten Fall eine vorhersagbare, unkritische Inkonsistenz erzeugt
  - Konsequenz: strenge Sequenzialisierung der Schreiboperationen -> eher langsam
- **Lazy-Write-Dateisysteme**
  - Veränderungen von Dateien finden zunächst nur im Cache statt und werden dann in optimierter Weise (im Hintergrund) zurückgeschrieben
  - Verringerung der physikalischen Plattenzugriffe -> schnell, aber eher unsicher

Dateisystemwiederherstellung (2)

- **Wiederherstellbares Dateisystem**
  - Erreichen der Sicherheit von Careful-Write-Systemen
  - Erzielen der Geschwindigkeit von Lazy-Write-Systemen
  - Konsistenzgarantie durch Protokollierungstechniken
- **Transaktionsbasiertes Protokollierungsschema**
  - Protokolldatei (Log File)
  - Vermerk: Aktion abgeschlossen oder ...
  - aufgezeichnete Teilaktionen wieder rückgängig machen

  → Journaling File Systems

Journaling-Dateisysteme am Beispiel von NTFS

- **Ursprünglich bei DEC für das Betriebssystem VMS entwickelt, später Übernahme nach Unix und auch NTFS**
- **Maßnahmen von NTFS sichern nur die Konsistenz des Dateisystems, verhindern jedoch keinen Nutzdatenverlust**
  - Transaktionskonzept: Alle Veränderungen an Dateien werden durch einen **Transaktionsbeginn** und ein **Transaktionsende** (Zusicherung, committment) geklammert, die beide zusammen mit den geplanten Datei-Operationen in der Protokolldatei $LogFil (sog. Journal) festgehalten
  - Die angegebenen Datei-Operationen können in einem beliebigen Zeitraum bis zum Transaktionsende abgewickelt und dabei insbesondere mit Hilfe eines Platten-Caches im Hauptspeicher zwecks Durchsatzerhöhung aufgeschoben werden.
  - Der **Cache-Manager** muss nur dafür sorgen, dass der Transaktionsbeginn als erste Teiloperation physikalisch auf die Platte geschrieben wird und das Transaktionsende als letzte
  - Durch einen Ausfall des Rechners unvollständig gebliebene Transaktionen werden später mit Hilfe der Protokolldatei entweder vervollständigt oder ungeschehen gemacht

Journaling-Dateisysteme
Protokolldatei $LogFil

- **Protokolldatei $LogFil ist von ihrer Lage und Größe her konstant.**
  - Veränderungen an dieser Datei betreffen daher nur ihren Nutzdatenbereich
  - Schreibaufträge, die die Buchhaltungsdaten von $LogFil betreffen, sind somit überflüssig und können, sofern sie überhaupt ausgeführt werden, keine Konsistenzprobleme verursachen
  - Die Protokolldatei wird als (scheinbar endloser) Ringpuffer geführt
    Sinnvoll, weil die älteren Einträge in der Protokolldatei nach dem Abschluss der zugehörigen Transaktion ohnehin veraltet sind und überflüssig werden.
- **Verwaltung der Protokolldatei:**
  Protokolldateidienst (LFS, log file service)
  - Nummerierung der **Protokolleinträge mit Folgenummern** (LSNs, logical sequence numbers) und
  - führt vor dem eigentlichen Protokollbereich einen **Zeiger auf den Beginn des aktuellen Bereichs**, ab dem die Protokolldatei zur Wiederherstellung des Dateisystems nach einem Ausfall des Rechners gelesen werden muss
  - Auch alle Änderungen in der Protokolldatei werden vom LFS veranlasst.

Journaling-Dateisysteme
Einträge in die Protokolldatei

- **Einträge in die Protokolldatei**
  - **Aktualisierungseintrag (update entry)**
    Beschreibt eine Teiloperation und enthält zweierlei Informationen (s. Beispiel unten):
    - Informationen für die **Wiederholung** geben an, wie eine Teiloperation erneut durchgeführt werden soll
    - Informationen für die **Rückgängigmachung** geben an, wie eine Teiloperation rückgängig gemacht wird
  - **Aufsetzpunkteintrag (checkpoint entry)**
    Definiert einen Wiederaufsetzpunkt (checkpoint) und wird alle 5 Sekunden in die Protokolldatei geschrieben
    Mit den Informationen, die in diesem Eintrag gespeichert sind, kann NTFS entscheiden, wie weit es in der Protokolldatei bei einer Wiederherstellungsaktion zurückgehen muss. Der obengenannte Zeiger auf den Beginn des aktuellen Bereichs zielt auf den letzten Aufsetzpunkteintrag.
  - **Zusicherungseintrag (committment entry)**
    Ein solcher Eintrag schließt eine Transaktion ab. Die zu einer Transaktion gehörigen Einträge sind in der Protokolldatei in einer verzeigerten Liste miteinander verbunden, zu deren letztem Element der Zusicherungseintrag wird
- **Transaktionsanfang**
  Markierung durch den ersten Aktualisierungseintrag der Transaktion

Journaling-Dateisysteme
Übertragungsaspekte

- **Schreiben auf die Festplatte**
  - Nicht sofort, wenn ein Zusicherungseintrag in die Protokolldatei geschrieben wird, wird diese auch auf die Platte hinausgeschrieben, sondern mehrere zugesicherte Transaktionen werden erst einmal gesammelt und dann in einem Rutsch physikalisch übertragen
  - Jedoch spätestens dann, wenn ein Aufsetzpunkteintrag in die Protokolldatei geschrieben wird; dies löst immer auch ein Hinausschreiben der Protokolldatei aus.

- **Gestaltung der Teiloperationen**
  - Mehrfache Ausführung bewirkt dasselbe wie nur eine einfache Ausführung (Idempotenz)
    - Beispiel: bei der Allokation neuer Sektorgruppen müssen die zugehörigen Bits im Allokationsvektor (Sektorgruppen-Bitmuster) in der Datei $Bitmap gesetzt werden. Eine Wiederholung dieser Teiloperation setzt die bereits gesetzten Bits eben ein zweites Mal

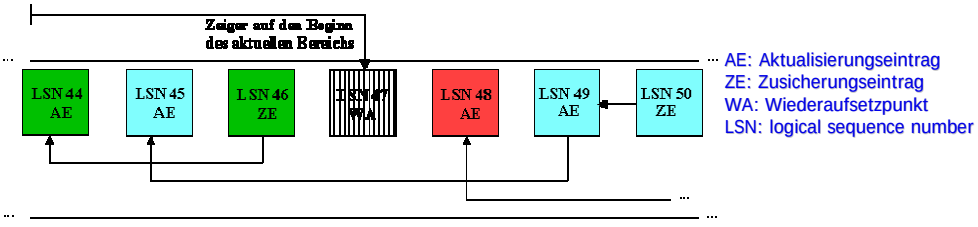
Journaling-Dateisysteme
Ablauf einer Transaktion

- **Ablauf einer Transaktion**
  - Mit dem Einfügen eines Aktualisierungseintrags in die Protokolldatei, der keiner anderen Transaktion angehört (also mit deren Einträgen verkettet ist), beginnt eine neue Transaktion
  - Es werden danach verschiedene Teiloperationen ausgeführt
    - jeweils vorher mit einem Eintrag in der Protokolldatei vermerkt und
    - untereinander verkettet werden
  - Wegen der Parallelität der Dateizugriffe verschiedener Prozesse sind übrigens die verketteten Listen mehrerer Transaktionen ineinander verschachtelt
  - Das alles findet zunächst im Hauptspeicher, also im Platten-Cache, statt. Wenn zu diesem Zeitpunkt der Rechner ausfällt, geht diese Transaktion zwar komplett verloren, aber das Dateisystem verbleibt in einem konsistenten (dem alten) Zustand.
- **Hinausschreiben auf die Platte**
  - z. B., wenn der Platten-Cache voll ist und Platz geschaffen werden muss
  - spätestens aber an einem Wiederaufsetzpunkt

Journaling-Dateisysteme
Ablauf einer Transaktion, Log File Service

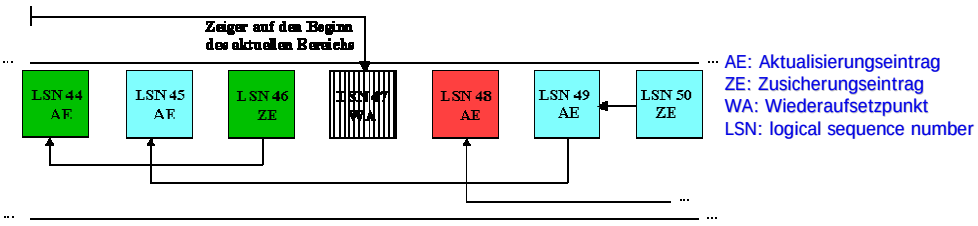
- **LFS (log file service)**
  - LFS & Cache-Manager: Immer erst die Änderungen in der Protokolldatei vor den Auswirkungen der zugehörigen Teiloperationen hinausschreiben
  - Wenn nach der Änderung der Protokolldatei auf der Platte der Rechner ausfällt, können anhand der Einträge in der Protokolldatei die betroffenen Teiloperationen entweder wiederholt oder, falls nötig, rückgängig gemacht werden
- **Abschluß einer Transaktion:**
  Schreiben eines Zusicherungseintrags in die Protokolldatei geschrieben (nicht sofort auch auf die Platte)
- **Wiederaufsetzpunkt**
  - Alle 5 Sekunden wird ein Wiederaufsetzpunkt definiert
    - Zunächst Schreiben aller Änderungen in der Protokolldatei
    - Dann alle übrigen Änderungen im Platten-Cache auf die Platte schreiben
    - Schließlich einen Aufsetzpunkteintrag in die Protokolldatei (und auch auf die Platte) schreiben
  - Außerdem wird in der Protokolldatei der Zeiger auf den Beginn des aktuellen Bereichs auf den neuen Aufsetzpunkteintrag gesetzt.
- **Abgeschlossene Transaktionen brauchen später bei einer Wiederherstellung nicht mehr berücksichtigt werden**

Journaling-Dateisysteme
Beispiel für den Ablauf einer Transaktion – 1



- **Neustart bzw. Wiederanlauf von Windows NT**
  - Veranlassung einer Wiederherstellung (die im Normalfall natürlich überflüssig ist). Dazu wird in der Protokolldatei der letzte Aufsetzpunkteintrag bestimmt, und es werden alle Transaktionen, die am Wiederaufsetzpunkt noch nicht zugesichert waren oder die später begonnen (und möglicherweise schon zugesichert) wurden, untersucht.
- **Nicht abgeschlossene Transaktionen (kein Zusicherungseintrag vorhanden)**
  - werden mit Hilfe der Informationen für die Rückgängigmachung ungeschehen gemacht (man weiß ja nicht, welche Teiloperationen noch bis zur Zusicherung gefehlt haben). In der obigen Abbildung betrifft dies die mit **rot** gekennzeichnete Transaktion.

Journaling-Dateisysteme
Beispiel für den Ablauf einer Transaktion – 2



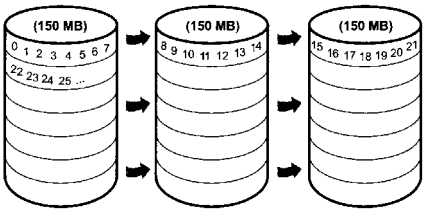
- **Zugesicherte Transaktionen**
  - werden hingegen mit Hilfe der Informationen für die Wiederholung in den Aktualisierungseinträgen wiederholt (denn es können ja zum Zeitpunkt des Ausfalls noch Teilergebnisse im Cache gewesen sein) und damit komplett wiederhergestellt. In der obigen Abbildung betrifft dies die mit **blau** gekennzeichnete Transaktion, an der man auch erkennt, dass die Wiederherstellungsaktionen teilweise auch noch Einträge vor dem Wiederaufsetzpunkt betreffen.
- **Anschließend ist das Dateisystem wieder konsistent, auch wenn die nicht zugesicherten Transaktionen dabei verloren gegangen sind!**

Journaling-Dateisysteme vs. Standard-Dateisysteme

- **Geschwindigkeit**
  - Im Gegensatz zu Unix läuft eine Wiederherstellung des Dateisystems in NT wesentlich schneller ab, da im wesentlichen ja nur die Transaktionen seit dem letzten Wiederaufsetzpunkt berücksichtigt werden müssen.
  - In Unix wird, sofern das Dateisystem nicht sauber heruntergefahren wurde (erkenntlich an einem Bit im Superblock), das Programm fsck gestartet, das den ganzen Dateisystembaum nach Inkonsistenzen absucht. Das kann bei großen Festplatten einige Minuten dauern.
- **Sicherheit**
  - Es werden tatsächlich alle Operationen auf dem Dateisystem erfasst. Es bleiben keine etwaigen Inkonsistenzen – wie sie trotz fsck oder scandisk auftreten können – zurück (s. Übungsaufgabe)

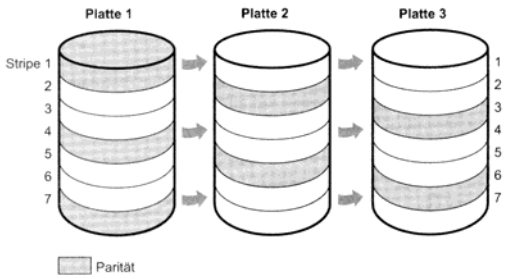
Weitere NTFS-Eigenschaften

- **Fehlertoleranztreiber**
  - Funktionen zur redundanten Datenspeicherung und zur dynamischen Wiederherstellung von Daten
- **Datenträgersätze**
  - Mehrere Partitionen können zu einem Dateisystem konsolidiert werden
- **Stripe Sets**
  - Folge von Partitionen, von denen jede auf einer anderen Platte liegt
  - Zusammenfassung zu einem logischen Dateisystem
  - Dateien werden über mehrere Platten hinweg verteilt -> Geschwindigkeitssteigerung



NTFS-Fehlertoleranz-Eigenschaften

- **Spiegelsätze**
  - Inhalt einer Partition wird dupliziert und in eine Partition der gleichen Größe auf einer anderen Platte geschrieben
  - Im Fehlerfall: Zugreifen auf die andere Platte
  - Möglichkeit: Lastverteilung durch parallelen Lesezugriff auf beide Platten
- **Stripe-Sets mit Parität**
  - Parität: Byte-für-Byte gebildetes log. XOR



RAID

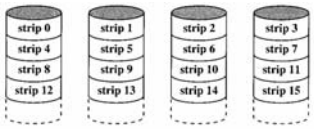
Redundant Array of Independent Disks

RAID – Redundant Array of Independent Disks
(früher auch: Redundant Array of Inexpensive Disks)

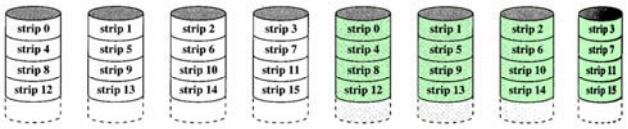
- **Charakteristika**
  - RAID: Menge von physikalischen Laufwerken, die vom BS als ein einzelnes (logisches) Laufwerk gesehen werden
  - Die Daten sind über die physikalischen Laufwerke verteilt
  - Redundante Datenkapazität wird verwendet, um Paritätsinformationen abzuspeichern, die beim Ausfall einer Platte ermöglicht, die Daten wieder herzustellen
  - RAID-Ebenen: 0 ... 6
- **Ausfallsicherheit**
  - Verwendung mehrerer unabhängiger Platten erhöht die Fehlerwahrscheinlichkeit
    n Geräte haben i.a. 1/n der Ausfallsicherheit eines Laufwerks!
  - Redundante Informationen können die Ausfallsicherheit erhöhen
  - MTTF (mean time to fail) von Platten [10 Jahre], MTTR (mean time to repair) [Stunden], d.h. kritischer Zeitraum lediglich: Während eine Platte repariert wird, fällt noch eine zweite aus.
- **Hot-spares:** Platten, die nur als Ersatz für den Ausfall anderer Platten bereitstehen
- **Hot-swapping:** Platten können ausgetauscht werden, ohne dass das System heruntergefahren werden muss

RAID – Übersicht

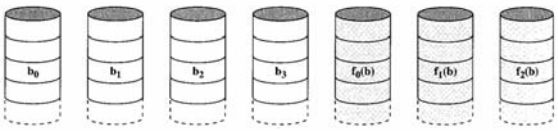
- **RAID-0**
  - Verteilung der Daten auf mehrere Platten
  - Erhöhung der I/O-Bandbreite
- **RAID-1**
  - RAID-0 + Spiegelung
  - Erhöhung der I/O-Bandbreite
  - Kostenintensiv
- **RAID-2**
  - sehr kleine 'strips' [Byte/word]
  - Beim Schreiben muss auf alle Platten zugegriffen werden
  - Kostenintensiv, overkill



(a) RAID 0 (nonredundant)



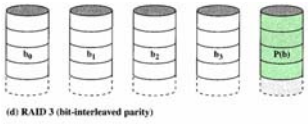
(b) RAID 1 (mirrored)



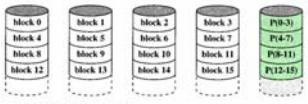
(c) RAID 2 (redundancy through Hamming code)

RAID – Übersicht

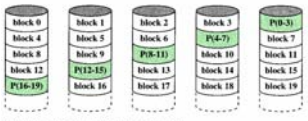
- **RAID – 3**
  - ebenfalls sehr kleine Streifen
  - dedizierte Paritätsplatte (Bottleneck)
  - sehr hohe Bandbreite, niedrige I/O-Auftragsrate
  - Allerdings nur ein Zugriff zu einer Zeit mögl.
- **RAID – 4**
  - relativ große Streifen
  - niedrigere I/O-Bandbreite, hohe I/O-Auftragsrate
  - Write-penalty:
    Pro Schreibauftrag: 2xLesen u. 2xSchreiben!
  - Paritätsplatte: Bottleneck
- **RAID – 5**
  - analog zu RAID-4, aber Verteilung der Paritätsinformationen über alle Platten
- **RAID – 6**
  - Zwei verschiedene Sicherungsalgorithmen
  - Zwei gleichzeitige Plattenausfälle können kompensiert werden



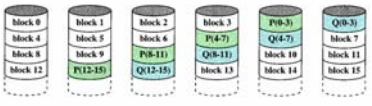
(d) RAID 3 (bit-interleaved parity)



(e) RAID 4 (block-level parity)



(f) RAID 5 (block-level distributed parity)



(g) RAID 6 (dual redundancy)

RAID
Redundanz durch Parität, Zusatzaufwendungen

- **RAID – 3**
  - X4(i) = X3(i) ⊕ X2(i) ⊕ X1(i) ⊕ X0(i)
    X0 .. X3 - Datenplatten
    X4 – Paritätsplatte
  - Annahme X1 fällt aus. Wir addieren X4(i) ⊕ X1(i) auf beide Seiten der Gleichung
  - X1(i) = X4(i) ⊕ X3(i) ⊕ X2(i) ⊕ X0(i)
  - Fehlende Daten können somit aus den verbleibenden Platten rekonstruiert werden
- **RAID – 4/5**
  - Schreiben von Daten in den Streifen auf Platte X1; d.h. X1 → X1'
  - Initial: X4(i) = X3(i) ⊕ X2(i) ⊕ X1(i) ⊕ X0(i) — wie oben
  - X4'(i) = X3(i) ⊕ X2(i) ⊕ X1'(i) ⊕ X0(i) — Daten von allen Platten lesen?
    = X3(i) ⊕ X2(i) ⊕ X1'(i) ⊕ X0(i) ⊕ X1(i) ⊕ X1(i) — Einsetzen von X1(i) s.o.
    = ~~X3(i)~~ ⊕ ~~X2(i)~~ ⊕ X1'(i) ⊕ ~~X0(i)~~ ⊕ X1(i) ⊕ X4(i) ⊕ ~~X3(i)~~ ⊕ ~~X2(i)~~ ⊕ ~~X0(i)~~
    = X4(i) ⊕ X1(i) ⊕ X1'(i)
  - Write-penalty: Zwei Lesezugriffe (X1-alt, X4-Parität) und zwei Schreibzugriffe (X1'-neu, X4-neue Parität) erforderlich!
  - Im Falle größerer Schreiboperationen, die alle Platten betreffen, ist ein einfaches Neuerrechnen der Parität ausreichend

RAID – Ebenen im Vergleich

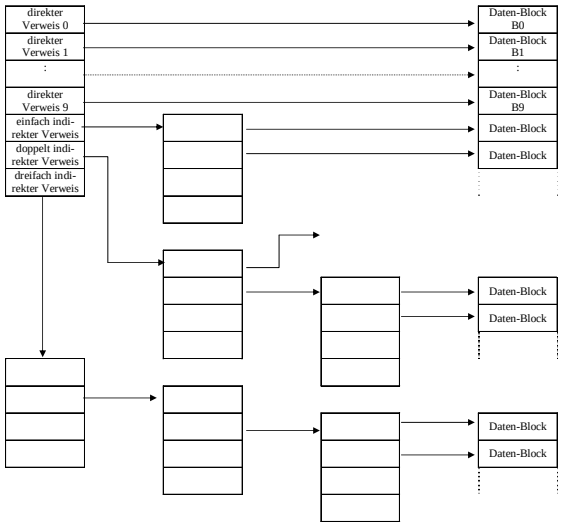
Category	Level	Description	I/O Request Rate (Read/Write)	Data Transfer Rate (Read/Write)	Typical Application
Striping	0	Non-redundant	Large strips: Excellent	Small strips: Excellent	Applications requiring high performance for noncritical data
Mirroring	1	Mirrored	Good/fair	Fair/fair	System drives; critical files
Parallel access	2	Redundant via Hamming code	Poor	Excellent	
	3	Bit-interleaved parity	Poor	Excellent	Large I/O request size applications such as imaging, CAD
Independent access	4	Block-interleaved parity	Excellent/fair	Fair/poor	
	5	Block-interleaved distributed parity	Excellent/fair	Fair/poor	High request rate, read-intensive, data lookup
	6	Block-interleaved dual distributed parity	Excellent/poor	Fair/poor	Applications requiring extremely high availability

Unix-Dateisystem

Unix - Dateisystem

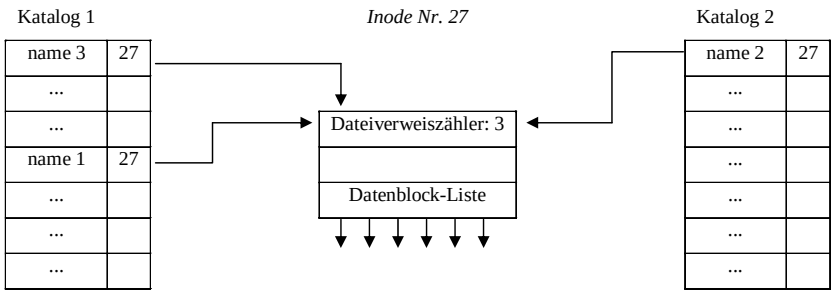
- **Urladeblock**
  - enthält Code für den Urladevorgang
- **Superblock**
  - Größe des Dateisystems (# Blöcke)
  - # freie Blöcke des Dateisystems
  - Liste der freien Blöcke
  - Zeiger auf nächsten Block der Freiliste
  - Größe der Inode-Tabelle
  - # freie Inodes
  - Liste der freien Inodes
  - Zeiger auf nächsten Inode der Freiliste
  - Semaphorfelder für die zwei Freilisten
  - Update-Flag
- **Inode (index node)**
  - Dateityp (regulär, Katalog, Zeichen- oder Block-Spezial, FIFO)
  - Zugriffsrechte (rwxrwxrwx)
  - Dateiverweiszähler
  - Datei-Eigentümer und dessen Benutzer-Gruppe
  - Dateigröße in Bytes
  - Zeitstempel für die letzte Datei-Änderung
  - den letzten Datei-Zugriff
  - die letzte Inode-Änderung
  - Liste der Datenblöcke der Datei

Aufbau der Liste der Datenblöcke in einem Inode



Kataloge und Dateireferenzierung

- **Katalog ist eine spezielle Datei mit jeweils folgenden Einträgen**
  - Inode-Nummer
  - Zeichenkette mit maximaler Länge 14



CD-ROM Dateisystem

CD ROM - Dateisystem

- **Unterteilung in Sektoren à 3234 Byte**
  - 2048 Byte Nutzdaten
  - Byte wird durch 14 Bits + 3 Koppelbits dargestellt
  - Synchronisations-, Typ-, Steuer- und Fehlerkorrektur-Informationen
  - Sektor ist in 1,2 oder 4 Blöcke unterteilbar
- **Adressierung**
  - in Form einer Zeitangabe MM:SS:DD (DD ~ 1/75)
  - fortlaufende Sektornumerierung
  - fortlaufende Blocknumerierung
- **Audio-CDs**
  - Verzicht auf eine der Fehlerkorrektur-Ebenen
  - 2352 Byte Nutzdaten pro Sektor
  - Fehlerwahrscheinlichkeit 10^{-8} statt 10^{-12}

Grundstruktur einer CD-ROM

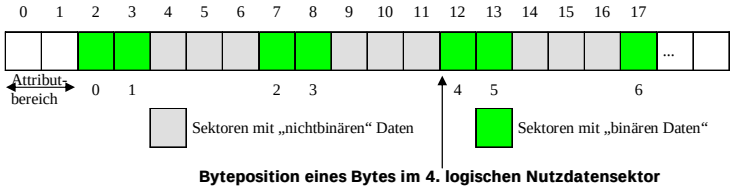
- **Die ersten 16 Sektoren (0 bis 15) bleiben bei CD-ROMs frei**
- **Sektor 16: primärer CD-Deskriptor**
  - die Größe der logischen Blöcke (512, 1024 oder 2048 byte)
  - der Name der CD-ROM
  - belegter Platz in Blöcken
  - Zeiger auf ersten Sektor und Größe der Pfadtabellen
  - Katalogeintrag für den Stammkatalog (mit Zeiger auf dessen ersten Sektor)
  - Zeitpunkte für früheste und letztmalige Nutzung der CD-ROM (!)
  - verwendeter Standard (normal oder XA, ISO-9660-Version)
  - Hersteller- und urheberrechtliche Informationen
  - Datum und Zeitpunkt der Herstellung
  - Ende durch Terminatorsektor
- **Anschließend Pfadtabellen und der Datenbereich des Stammkatalogs**
- **Dann Datenbereiche der Unterkataloge und Dateien des Stammkatalogs**

Kataloge

- **Kataloge ~ spezielle Dateien, Einträge lexikographisch geordnet**
- **Zusätzlich: gesamte Katalog-Hierarchie in eigenem Datenbereich (Pfadtabelle) am Anfang des Systembereichs der CD ROM**
- **Datei-/Katalogeintrag unterschiedlicher Länge**
  - 1 — Länge des vorliegenden Eintrags in byte (maximal 256)
  - 2 — Länge des optionalen Attributbereichs in byte (oder Blöcken?), ≤ 256
  - *3 – 10* — des Blocks, mit dem der Datenbereich der Datei beginnt
  - *11 – 18* — Länge des Datenbereichs in byte
  - 19 – 25 — Erstellungsdatum und -zeit
  - 26 — Dateiflaggen
  - 27 — Bei Verschachtelung: Größe einer Dateneinheit (Sektorgruppe)
  - 28 — Lücke zw. 2 Dateneinheiten in Anzahl Sektoren bei Verschachtelung
  - *29 - 32* — Folgenummer der CD-ROM in einem Satz von CD-ROMs
  - 33 — Länge des Dateinamens
  - 34 - x — ISO-konformer Dateiname der Länge x-33 byte (max. 31 byte)
  - (x+1) — falls x ungerade: Füllbyte zur Ausrichtung auf Wortadressen.
  - ab x+1|2 — Systemdaten: frei für Systemnutzung, z. B. bei XA und RRIP belegt.

Dateidarstellung

- **Die Dateien selbst sind zusammenhängend abgelegt**
  - Adressierung durch Angabe des Anfangssektors und der Dateilänge in Byte
- **Bei XA Möglichkeit zur Verschachtelung (Interleaving) von Binärdaten mit "nichtbinären" Daten**
  - Datei durch Dateinummer von 1 bis 255 (= 0, wenn nicht verschachtelt) identifiziert
  - Abstand der Dateisektoren als Anzahl von Sektoren im Katalogeintrag
- **Beispiel einer verschachtelten Datei**
  - zwei Sektor langer Attributbereich
  - Sektorgruppengröße (Byte Nr. 27) von zwei Sektoren
  - Lücke (Byte Nr. 28) von drei Sektoren zwischen zwei Sektorgruppen



Auflistung von Dateisystemen

- **affs** - Amiga fast file system
- **ext / ext2** - extended Filesystem (Linux)
- **hpfs** - high performance file system (OS/2)
- **iso9660** - CDROMs
- **minix** - Lehrdateisystem (A. Tanenbaum)
- **ncpfs** - Einbinden von Novell-Volumes
- **nfs** - Network file system
- **smbfs** - Server Message Block (Windows Protokoll für das Anbinden von Netzlaufwerken)
- **sysv** - SCO-Unix, Xenix und Coherent (Unix-Derivate)
- **ufs** - BSD, SunOS und NeXTstep
- **umsdos** - Unix on MSDOS
- **vfat** - virtual fat (FAT32 und lange Dateinamen von FAT16)

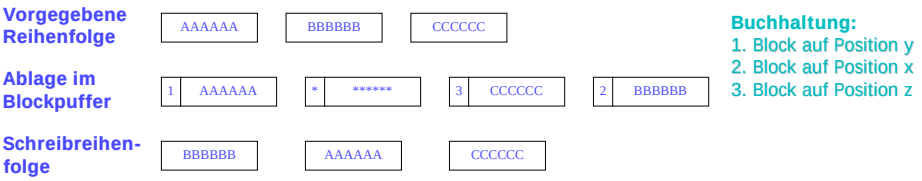
Plattenzugriffsverwaltung

Ein-/Ausgabe-Behandlung
Wiederholung: Zweck der Geräteverwaltung durch das BS

- **BS bietet über Systemaufrufe beim HW-Zugriff mehr Komfort und vor allem Geräteunabhängigkeit**
  - Programmierer braucht sich nicht mit der Programmierung einzelner Geräte zu befassen, Geräte-Adressen und die Länge von Transferabschnitten (z. B. Sektoren) kennen, Zeitabhängigkeiten beachten, die Daten in eine gerätespezifische Form bringen
- **reale physikalische Geräte werden zu virtuellen Geräten mit idealisierten Eigenschaften**
  - virtueller Drucker
  - Plattendateien (Abstraktion von Festplatten, Disketten, CD-ROMs...)
- **Betriebssystem beansprucht das Monopol auf alle Gerätezugriffe**
  - notwendig für die Koordination solcher Zugriffe zwischen verschiedenen Prozessen
  - Verhinderung des Verstellens von Geräteparametern durch einzelne Prozesse
  - Alle Ein-/Ausgabebefehle des Prozessors müssen privilegiert sein
    - Abbruch als Privilegverletzung
    - Betriebssystem behandelt den Befehl als E/A-Auftrag und führt die Ein-/Ausgabe nach eigenen Gesichtspunkten durch.
  - Betriebssystem kann besser als einzelne Prozesse **Optimierungsmaßnahmen** zur Effizienzsteigerung durchführen

Block- und Zeichengeräte

- **Datenübertragung an Zeichengerät in streng sequentieller Reihenfolge**
  - Bsp.: Drucker, Terminals oder serielle Schnittstellen
  - Datenpufferung nur insoweit, wie die vorgegebene Reihenfolge nicht verletzt wird
- **Blockgeräte**
  - typischerweise Massenspeichergeräte mit wahlfreier Zugriffspositionierung
  - Buchhaltung (Dateisystem) zum Merken der räumlichen Verteilung.
  - Zur Vereinfachung der Buchhaltung Datenablage mit einheitlicher Länge
  - Optimierungsmöglichkeiten:
    - Weiterleiten in anderer Reihenfolge als in der Transfer-Warteliste
    - Wichtig: Blockpufferung mit entsprechendem Vorspann



Optimierungsstrategien Kriterien

- **Kriterien zur Beurteilung verschiedener Strategien:**
  - Mittlerer Durchsatz (abgearbeitete Aufträge pro Zeiteinheit)
  - mittlere Zugriffszeit T (wird durch lange Kopfbewegungen vergrößert)
  - (Theoretisch) maximale Zugriffszeit T_{max} einer Auftragsfolge
  - Varianz v der Wartezeit (vom Eintreffen bis zur Abarbeitung eines Auftrags)
  - Fairness (werden bestimmte Aufträge bzw. Datenbereiche im Gerät benachteiligt oder gar ausgehungert?)
- **Aspekte:**
  - hoher Durchsatz --> evtl. Benachteiligung einzelner Aufträge (hohes T_{max}).
  - Niedriges v --> Aufträge werden mit einer Zugriffszeit in der Nähe von T abgearbeitet
  - Hohes v --> viele Aufträge weichen stark vom Mittelwert T ab
- **Wahl der günstigsten Strategie abhängig von:**
  - Typ des bedienten Geräts
  - Zugriffsverhalten der Anwendungsprozesse (sequentiell/wahlfrei positionierend)
  - Verteilung der Zugriffspositionen (Fragmentierungsgrad)
  - Systemlast

Disk-Scheduling Strategien

Beispielauftragsfolge: 23 111 44 66 105 11 66

- **Ankunftsreihenfolge (FCFS)**
  - wenig effizient, sehr gute Fairness, T besonders schlecht, insbes. bei hoher Systemlast
    Ausführung: 23 111 44 66 105 11 66 Start: 100: Σ = 442
- **Einfache Fahrstuhlstrategie (Pickup)**
  - ungefähr genauso fair wie FCFS, deutliche Durchsatzverbesserung
    Ausführung: 66 66 44 23 105 111 11 Start: 100: Σ = 265
- **Volle Fahrstuhlstrategie (Scan, Look)**
  - noch fair, Tendenz zu hohen Wartezeiten in den äußeren Spuren (hohes v)
    Ausführung: 66 66 44 23 11 105 111 Start: 100↓: Σ = 189
    Ausführung: 105 111 66 66 44 23 11 Start: 100↑: Σ = 111
- **Zirkuläre Fahrstuhlstrategie (Circular Scan / Look)**
  - Niedrigeres v (als bei der vollen Fahrstuhlstrategie), etwas höheres T.
    Praxis: niedrige Systemlast: volle Fahrstuhlstrategie verwenden, sonst: zirkuläre
    Ausführung: 66 66 44 23 11 111 105 Start: 100↓ : Σ = 195
    Ausführung: 105 111 11 23 44 66 66 Start: 100↑ : Σ = 166

Disk-Scheduling Strategien (2)

- **Beispielauftragsfolge: 23 111 44 66 105 11 66**
- **Kürzeste Positionierungszeit (SSTF)**
  - Minimierung der Positionierungsvorgänge, hoher Durchsatz, Gefahr des Aushungerns (besonders hohes v, für Dialogbetrieb unzumutbar)
    Ausführung: 105 111 66 66 44 23 11 Start: 100: Σ = 111
- **Prioritätsgesteuerte Strategie**
  - Blöcken erhalten Prioritäten (z.B. Prozeßpriorität)
  - Gesamtsystemdurchsatz???, Gefahr des Aushungerns für Blöcke schlechter Priorität
  - Meist: Kombination mit anderer Strategie zur Durchsatzoptimierung

Vorauslesen und aufgeschobenes Schreiben

- **Vorauslesen**
  - nicht nur den gewünschten Block, sondern alle Blöcke auf derselben Spur lesen
  - Lohnend, wenn Spurwechselzeit deutl. höher als die Zeit für eine Umdrehung ist
  - Variante: Mit dem Lesen der Spur sofort nach Erreichen beginnen, OHNE erst auf den gewünschten Sektor zu warten.
    - Die zu lesende Datei sollte nicht zu klein sein (Umfang von mehr als einer Spur)
    - weitgehend zusammenhängende Plattenablage (Fragmentierungsgrad ?)
    - überwiegend sequentielles Lesen (Programmladen vs. Datenbankzugriff)
- **Aufgeschobenes (verzögertes) Schreiben (deferred / delayed write, write behind)**
  - Schreibauftrag nicht sofort, sondern zu einem geeigneteren Zeitpunkt ausführen
    - System (Prozessor bzw. Gerät) wird nicht anderweitig beansprucht
    - Blockpuffer voll
    - nach Ablauf einer Zeitschranke, so mit Sicherheit irgendwann tatsächlich geschrieben wird
  - Nebeneffekt: schreibender Prozeß kann sofort nach Auftragsabgabe fortgesetzt werden
  - Problem, wenn der Schreibauftrag aus irgendwelchen Gründen später nicht ausgeführt werden kann (Benachrichtigung des Auftraggebers? Evtl. bereits beendet!)

Datenmodifikation

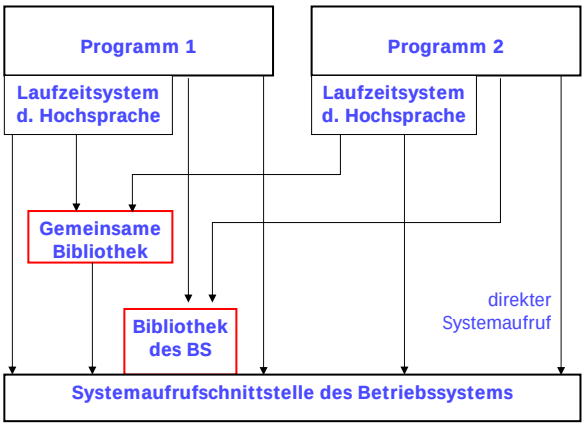
- **Für den Anwendungsprozeß transparentes Modifizieren von Daten:**
  - Kompression:
    Zur Reduzierung der Datenmenge werden die Daten vor dem Transfer komprimiert.
  - Verschlüsselung:
    Zur Wahrung des Datenschutzes werden die Daten vor dem Transfer verschlüsselt (chiffriert).
  - Fehlererkennung und -korrektur:
    Die Daten werden mit Redundanz (Paritätsbits, Prüfsummen usw.) versehen.
  - speziell bei Tastaturen:
    einfache Editiermöglichkeiten, bei denen keine Rücktransformation erforderlich ist (Einige Beispiele: das eingegebene Zeichen ^H wird zusammen mit dem vorangehenden nicht transferiert, Tabulatorzeichen werden expandiert, ein Zeilenende dient als Abschlußzeichen (Begrenzer) für die aktuelle Eingabe, wird selbst aber nicht übertragen).

Systemaufrufschnittstelle zur Dateiverwaltung

Datei einrichten	create
Datei (Gerät) eröffnen	open
Datei (Gerät) schließen	close
Datei löschen	unlink
Zugriffsposition bestimmen / positionieren	seek
Lesezugriff	read
Schreibzugriff	write
Parametrieraufruf	control

- + Erweiterungen

Bibliotheksfunktionen



Beispiele:

- gepufferte Datenzugriffe
- Datenablage in formatierter, lesbarer Form
- Eigentümer- oder weitere Einzelparameteränderungen