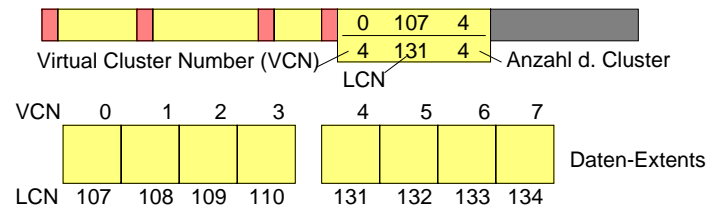


Master-File-Table (3)

Eintrag für eine längere Datei



- **Extents** werden außerhalb der MFT in aufeinanderfolgenden Clustern gespeichert
- Lokalisierungsinformationen werden in einem eigenen Stream gespeichert



Master-File-Table (4)

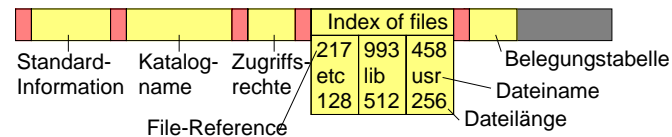
Mögliche weitere Streams (*Attributes*)

- Index
 - Index über einen Attributsschlüssel (z.B. Dateinamen) implementiert Katalog
- Indexbelegungstabelle
 - Belegung der Struktur eines Index
- Attributliste (immer in der MFT)
 - wird benötigt, falls nicht alle Streams in einen MFT Eintrag passen
 - referenzieren weitere MFT Einträge und deren Inhalt
- Streams mit beliebigen Daten
 - wird gerne zum Verstecken von Viren genutzt, da viele Standard-Werkzeuge von Windows nicht auf die Bearbeitung mehrerer Streams eingestellt sind (arbeiten nur mit dem unbenannten Stream)



Master File Table (5)

Eintrag für einen kurzen Katalog

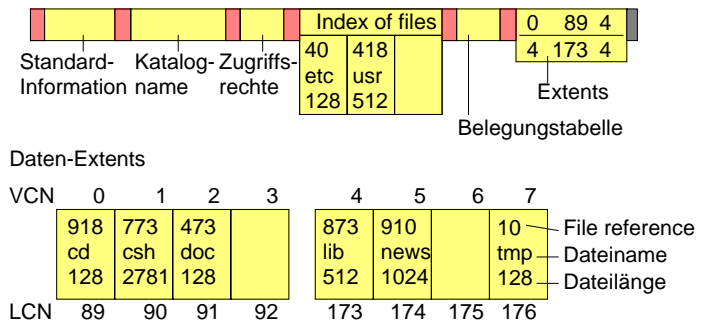


- Dateien des Katalogs werden mit File-References benannt
- Name und Standard-Attribute (z.B. Länge) der im Katalog enthaltenen Dateien und Kataloge werden auch im Index gespeichert (doppelter Aufwand beim Update; schnellerer Zugriff beim Kataloglisten)



Master File Table (6)

Eintrag für einen längeren Katalog



- Speicherung als B⁺-Baum (sortiert, schneller Zugriff)
- in einen Cluster passen zwischen 3 und 15 Dateien (im Bild nur eine)



Metadaten

- Alle Metadaten werden in Dateien gehalten

Indexnummer	0	MFT	Feste Dateien in der MFT
	1	MFT Kopie (teilweise)	
	2	Log File	
	3	Volume Information	
	4	Attributtabelle	
	5	Wurzelkatalog	
	6	Clusterbelegungstabelle	
	7	Boot File	
	8	Bad Cluster File	
	...		
	16	Benutzerdateien u. -kataloge	
	17		
	...		



Metadaten (2)

- Bedeutung der Metadateien
 - MFT und MFT Kopie: MFT wird selbst als Datei gehalten (d.h. Cluster der MFT stehen im Eintrag 0)
MFT Kopie enthält die ersten 16 Einträge der MFT (Fehlertoleranz)
 - Log File: enthält protokollierte Änderungen am Dateisystem
 - Volume Information: Name, Größe und ähnliche Attribute des Volumes
 - Attributtabelle: definiert mögliche Ströme in den Einträgen
 - Wurzelkatalog
 - Clusterbelegungstabelle: Bitmap für jeden Cluster des Volumes
 - Boot File: enthält initiales Programm zum Laden, sowie ersten Cluster der MFT
 - Bad Cluster File: enthält alle nicht lesbaren Cluster der Platte
NTFS markiert automatisch alle schlechten Cluster und versucht die Daten in einen anderen Cluster zu retten



Fehlererholung

- NTFS ist ein Journaling-File-System
 - Änderungen an der MFT und an Dateien werden protokolliert.
 - Konsistenz der Daten und Metadaten kann nach einem Systemausfall durch Abgleich des Protokolls mit den Daten wieder hergestellt werden.
- ▲ Nachteile
 - etwas ineffizienter
 - nur für Volumes >400 MB geeignet



Dateisysteme mit Fehlererholung

- Metadaten und aktuell genutzte Datenblöcke geöffneter Dateien werden im Hauptspeicher gehalten (Dateisystem-Cache)
 - effizienter Zugriff
 - Konsistenz zwischen Cache und Platte muss regelmäßig hergestellt werden
 - synchrone Änderungen: Operation kehrt erst zurück, wenn Änderungen auf der Platte gespeichert wurden
 - asynchrone Änderungen: Änderungen erfolgen nur im Cache, Operation kehrt danach sofort zurück, Synchronisation mit der Platte erfolgt später
- Mögliche Fehlerursachen
 - Stromausfall (dummer Benutzer schaltet einfach Rechner aus)
 - Systemabsturz



Konsistenzprobleme

- Fehlerursachen & Auswirkungen auf das Dateisystem
 - Cache-Inhalte und aktuelle E/A-Operationen gehen verloren
 - inkonsistente Metadaten
 - z. B. Katalogeintrag fehlt zur Datei oder umgekehrt
 - z. B. Block ist benutzt aber nicht als belegt markiert
- ★ Reparaturprogramme
 - Programme wie **chkdsk**, **scandisk** oder **fsck** können inkonsistente Metadaten reparieren
- ▲ Datenverluste bei Reparatur möglich
- ▲ Große Platten bedeuten lange Laufzeiten der Reparaturprogramme



Journaling-File-Systems

- Zusätzlich zum Schreiben der Daten und Meta-Daten (z. B. Inodes) wird ein Protokoll der Änderungen geführt
 - Grundidee: Log-based Recovery bei Datenbanken
 - alle Änderungen treten als Teil von Transaktionen auf.
 - Beispiele für Transaktionen:
 - Erzeugen, Löschen, Erweitern, Verkürzen von Dateien
 - Dateiattribute verändern
 - Datei umbenennen
 - Protokollieren aller Änderungen am Dateisystem zusätzlich in einer Protokolldatei (*Log File*)
 - beim Bootvorgang wird Protokolldatei mit den aktuellen Änderungen abgeglichen und damit werden Inkonsistenzen vermieden.



Journaling-File-Systems (2)

- Protokollierung
 - für jeden Einzelvorgang einer Transaktion wird zunächst ein Logeintrag erzeugt und
 - danach die Änderung am Dateisystem vorgenommen
 - dabei gilt:
 - der Logeintrag wird immer **vor** der eigentlichen Änderung auf Platte geschrieben
 - wurde etwas auf Platte geändert, steht auch der Protokolleintrag dazu auf der Platte
- Fehlererholung
 - Beim Bootvorgang wird überprüft, ob die protokollierten Änderungen vorhanden sind:
 - Transaktion kann wiederholt bzw. abgeschlossen werden (*Redo*) falls alle Logeinträge vorhanden
 - angefangene, aber nicht beendete Transaktionen werden rückgängig gemacht (*Undo*).



Journaling-File-Systems (3)

- Beispiel: Löschen einer Datei im NTFS
 - Vorgänge der Transaktion
 - Beginn der Transaktion
 - Freigeben der Extents durch Löschen der entsprechenden Bits in der Belegungstabelle (gesetzte Bits kennzeichnen belegten Cluster)
 - Freigeben des MFT-Eintrags der Datei
 - Löschen des Katalogeintrags der Datei (evtl. Freigeben eines Extents aus dem Index)
 - Ende der Transaktion
 - Alle Vorgänge werden unter der File-Reference im Log-File protokolliert, danach jeweils durchgeführt.
 - Protokolleinträge enthalten Informationen zum *Redo* und zum *Undo*



Journaling-File-Systems (4)

- Log vollständig (Ende der Transaktion wurde protokolliert und steht auf Platte):
 - *Redo* der Transaktion:
alle Operationen werden wiederholt, falls nötig
- Log unvollständig (Ende der Transaktion steht nicht auf Platte):
 - *Undo* der Transaktion:
in umgekehrter Reihenfolge werden alle Operation rückgängig gemacht
- Checkpoints
 - Log-File kann nicht beliebig groß werden
 - gelegentlich wird für einen konsistenten Zustand auf Platte gesorgt (*Checkpoint*) und dieser Zustand protokolliert (alle Protokolleinträge von vorher können gelöscht werden)
 - ähnlich verfährt NTFS, wenn Ende des Log-Files erreicht wird.



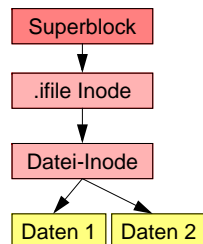
Journaling-File-Systems (5)

- ★ Ergebnis
 - eine Transaktion ist entweder vollständig durchgeführt oder gar nicht
 - Benutzer kann ebenfalls Transaktionen über mehrere Dateizugriffe definieren, wenn diese ebenfalls im Log erfasst werden
 - keine inkonsistenten Metadaten möglich
 - Hochfahren eines abgestürzten Systems benötigt nur den relativ kurzen Durchgang durch das Log-File.
 - Alternative **chkdsk** benötigt viel Zeit bei großen Platten
- ▲ Nachteile
 - ineffizienter, da zusätzliches Log-File geschrieben wird
- Beispiele: NTFS, EXT3, EXT4, ReiserFS



Copy-on-Write- / Log-Structured-File-Systems

- Alternatives Konzept zur Realisierung von atomaren Änderungen
- Alle Änderungen im Dateisystem erfolgen auf Kopien
 - Der Inhalt veränderter Blöcke wird in einen neuen Block geschrieben

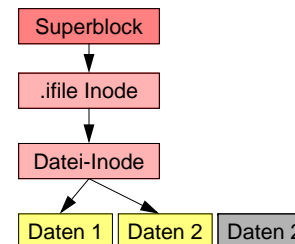


- Beispiel LinLogFS: Superblock einziger nicht ersetzter Block



Copy-on-Write- / Log-Structured-File-Systems

- Alternatives Konzept zur Realisierung von atomaren Änderungen
- Alle Änderungen im Dateisystem erfolgen auf Kopien
 - Der Inhalt veränderter Blöcke wird in einen neuen Block geschrieben

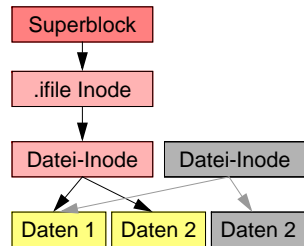


- Beispiel LinLogFS: Superblock einziger nicht ersetzter Block



Copy-on-Write- / Log-Structured-File-Systems

- Alternatives Konzept zur Realisierung von atomaren Änderungen
- Alle Änderungen im Dateisystem erfolgen auf Kopien
 - Der Inhalt veränderter Blöcke wird in einen neuen Block geschrieben

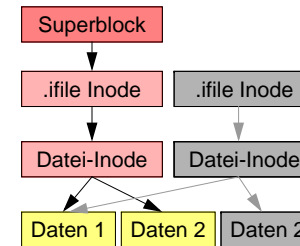


- Beispiel LinLogFS: Superblock einziger nicht ersetzter Block



Copy-on-Write- / Log-Structured-File-Systems

- Alternatives Konzept zur Realisierung von atomaren Änderungen
- Alle Änderungen im Dateisystem erfolgen auf Kopien
 - Der Inhalt veränderter Blöcke wird in einen neuen Block geschrieben

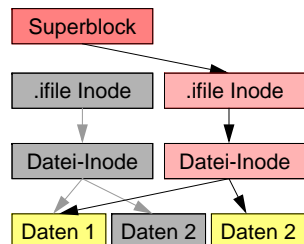


- Beispiel LinLogFS: Superblock einziger nicht ersetzter Block



Copy-on-Write- / Log-Structured-File-Systems

- Alternatives Konzept zur Realisierung von atomaren Änderungen
- Alle Änderungen im Dateisystem erfolgen auf Kopien
 - Der Inhalt veränderter Blöcke wird in einen neuen Block geschrieben

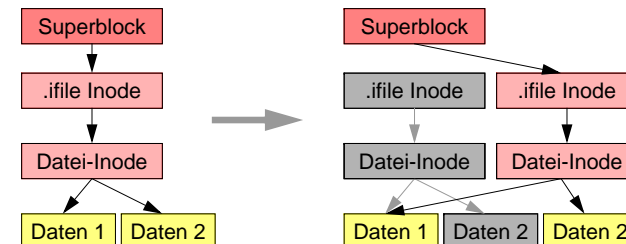


- Beispiel LinLogFS: Superblock einziger nicht ersetzter Block



Copy-on-Write- / Log-Structured-File-Systems

- Alternatives Konzept zur Realisierung von atomaren Änderungen
- Alle Änderungen im Dateisystem erfolgen auf Kopien
 - Der Inhalt veränderter Blöcke wird in einen neuen Block geschrieben



- Beispiel LinLogFS: Superblock einziger statischer Block (Anker im System)



Copy-on-Write- / Log-Structured-File-Systems (2)

- ★ Vorteile
 - Gute Schreibeffizienz - vor allem bei Log-Structured-File-Systems
 - Datenkonsistenz bei Systemausfällen
 - ein atomare Änderung macht alle zusammengehörigen Änderungen sichtbar
 - Schnappschüsse / Checkpoints einfach realisierbar
- ▲ Nachteile
 - Erzeugt starke Fragmentierung, die sich beim Lesen auswirken kann
 - Performanz nur akzeptabel, wenn Lesen primär aus Cache erfolgen kann oder Positionierzeiten keine Rolle spielen (SSD)
- Unterschied zwischen Copy-on-Write- und Log-Structured-File-Systems
 - Log-Structured-File-Systems schreiben kontinuierlich an das Ende des belegten Plattenbereichs und geben vorne die Blöcke wieder frei (kontinuierlicher Log)
 - Beispiele: Log-Structured: LinLogFS, BSD LFS
Copy-on-Write: ZFS, Btrfs (Oracle)



Fehlerhafte Plattenblöcke

- Blöcke, die beim Lesen Fehlermeldungen erzeugen
 - z.B. Prüfsummenfehler
- Hardwarelösung
 - Platte und Plattencontroller bemerken selbst fehlerhafte Blöcke und maskieren diese aus
 - Zugriff auf den Block wird vom Controller automatisch auf einen „gesunden“ Block umgeleitet
- Softwarelösung
 - File-System bemerkt fehlerhafte Blöcke und markiert diese auch als belegt



Datensicherung

- Schutz vor dem Totalausfall von Platten
 - z. B. durch Head-Crash oder andere Fehler

Sichern der Daten auf Tertiärspeicher

- Bänder, Bandroboter mit vorgelagertem Platten-Cache
- WORM-Speicherplatten (*Write Once Read Many*)
- Sichern großer Datenbestände
 - Total-Backups benötigen lange Zeit
 - Inkrementelle Backups sichern nur Änderungen ab einem bestimmten Zeitpunkt
 - Mischen von Total-Backups mit inkrementellen Backups



Einsatz mehrerer (redundanter) Platten

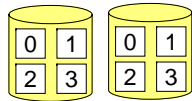
- Gestreifte Platten (*Striping*; RAID 0)
 - Daten werden über mehrere Platten gespeichert
-
- Datentransfers sind nun schneller, da mehrere Platten gleichzeitig angesprochen werden können
- ▲ Nachteil
 - keinerlei Datensicherung: Ausfall einer Platte lässt Gesamtsystem ausfallen



Einsatz mehrerer redundanter Platten (2)

■ Gespiegelte Platten (*Mirroring*; RAID 1)

- Daten werden auf zwei Platten gleichzeitig gespeichert



- Implementierung durch Software (File-System, Plattentreiber) oder Hardware (spez. Controller)
- eine Platte kann ausfallen
- schnelleres Lesen (da zwei Platten unabhängig voneinander beauftragt werden können)

▲ Nachteil

- doppelter Speicherbedarf

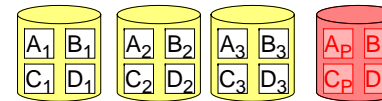
■ wenig langsames Schreiben durch Warten auf zwei Plattentransfers

■ Verknüpfung von RAID 0 und 1 möglich (RAID 0+1)

Einsatz mehrerer redundanter Platten (3)

■ Paritätsplatte (RAID 4)

- Daten werden über mehrere Platten gespeichert, eine Platte enthält Parität



- Paritätsblock enthält byteweise XOR-Verknüpfungen von den zugehörigen Blöcken aus den anderen Streifen
- eine Platte kann ausfallen
- schnelles Lesen
- prinzipiell beliebige Plattenanzahl (ab drei)

Einsatz mehrerer redundanter Platten (4)

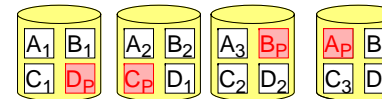
▲ Nachteil von RAID 4

- jeder Schreibvorgang erfordert auch das Schreiben des Paritätsblocks
- Erzeugung des Paritätsblocks durch Speichern des vorherigen Blockinhalts möglich: $P_{\text{neu}} = P_{\text{alt}} \oplus B_{\text{alt}} \oplus B_{\text{neu}}$ (P=Parity, B=Block)
- Schreiben eines kompletten Streifens benötigt nur einmaliges Schreiben des Paritätsblocks
- Paritätsplatte ist hoch belastet

Einsatz mehrerer redundanter Platten (5)

■ Verstreuter Paritätsblock (RAID 5)

- Paritätsblock wird über alle Platten verstreut



- zusätzliche Belastung durch Schreiben des Paritätsblocks wird auf alle Platten verteilt
- heute gängigstes Verfahren redundanter Platten
- Vor- und Nachteile wie RAID 4

■ Doppelte Paritätsblöcke (RAID 6)

- ähnlich zu RAID 5, aber zwei Paritätsblöcke (verkräftet damit den Ausfall von bis zu zwei Festplatten)
- wichtig bei sehr großen, intensiv genutzten RAID-Systemen, wenn die Wiederherstellung der Paritätsinformation nach einem Plattenausfall lange dauern kann