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.

Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

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

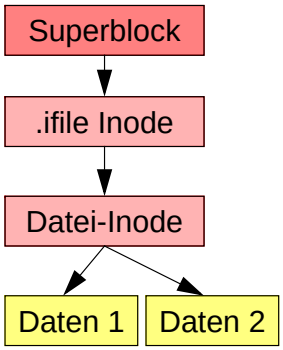
▲ Nachteile
- ineffizienter, da zusätzliches Log-File geschrieben wird

- Beispiele: NTFS, EXT3, EXT4, ReiserFS

Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

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

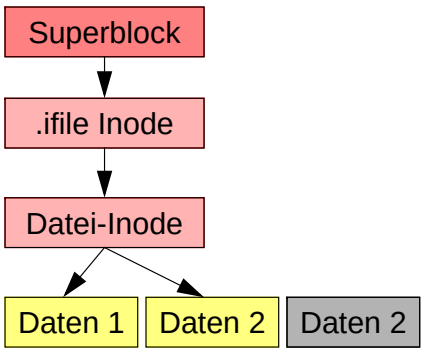


  - Beispiel LinLogFS: Superblock einziger nicht ersetzter Block

Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

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

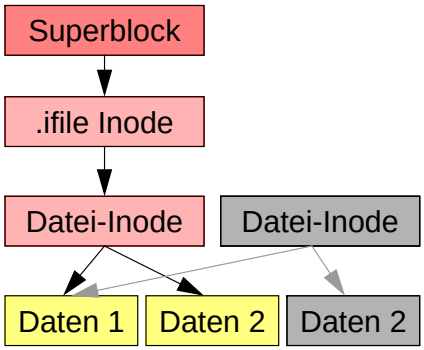


  - Beispiel LinLogFS: Superblock einziger nicht ersetzter Block

Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

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

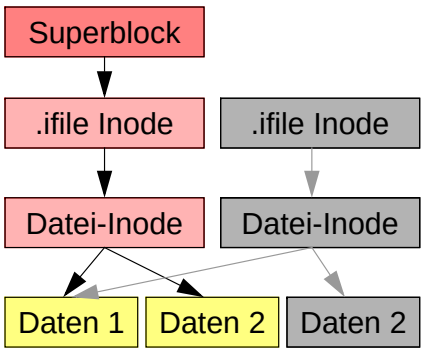


  - Beispiel LinLogFS: Superblock einziger nicht ersetzter Block

Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

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

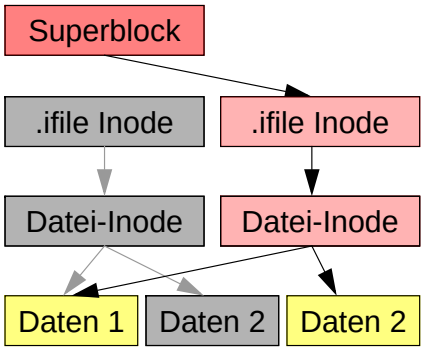


  - Beispiel LinLogFS: Superblock einziger nicht ersetzter Block

Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

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

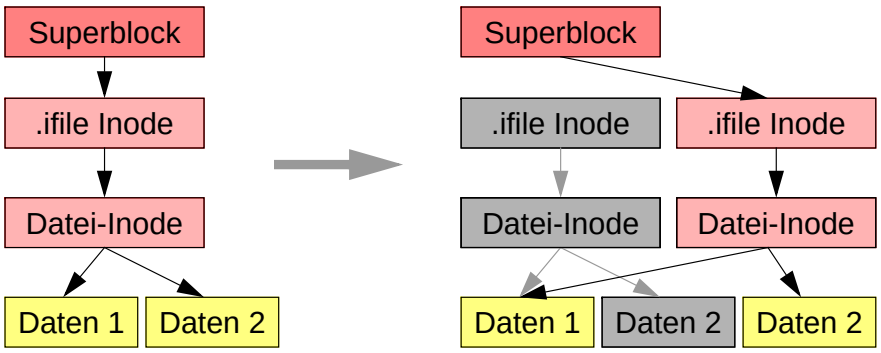


  - Beispiel LinLogFS: Superblock einziger nicht ersetzter Block

Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

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)

Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

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 dasEnde des belegten Plattenbereichs und geben vorne die Blöcke wieder frei (kontinuierlicher Log)
- Beispiele: Log-Structured: LinLogFS, BSD LFS
  Copy-on-Write: ZFS, Btrfs (Oracle)

Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

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

Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

Datensicherung

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

Sichern der Daten auf Tertiärspeicher

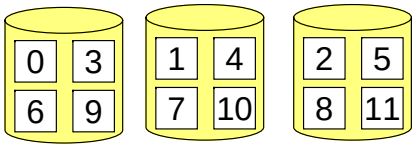
- ➤ Bänder, Bandroboter mit vorgelagertem Platten-Cache
- ➤ WORM-Speicherplatten *(Write Once Read Many)*

- Sichern großer Datenbestände
  - Total-Backups benötigen lange Zeit
  - Inkrementelle Backups sichern nur Änderungen ab einem bestimmten Zeitpunkt
  - Mischen von Total-Backups mit inkrementellen Backups

Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

Einsatz mehrerer (redundanter) Platten

- Gestreifte Platten (*Striping*; RAID 0)
  - Daten werden über mehrere Platten gespeichert



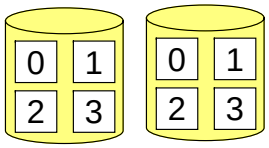
  - Datentransfers sind nun schneller, da mehrere Platten gleichzeitig angesprochen werden können

- ▲ Nachteil
  - keinerlei Datensicherung: Ausfall einer Platte lässt Gesamtsystem ausfallen

Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

Einsatz mehrerer redundanter Platten (2)

- Gespiegelte Platten (*Mirroring*; RAID 1)
  - Daten werden auf zwei Platten gleichzeitig gespeichert

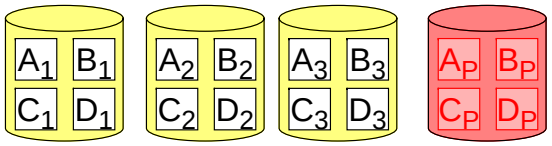


  - Implementierung durch Software (File-System, Plattentreiber) oder Hardware (spez. Controller)
  - eine Platte kann ausfallen
  - schnelleres Lesen (da zwei Platten unabhängig voneinander beauftragt werden können)
- Nachteil
  - doppelter Speicherbedarf
- wenig langsameres Schreiben durch Warten auf zwei Plattentransfers
- Verknüpfung von RAID 0 und 1 möglich (RAID 0+1)

Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

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)

Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

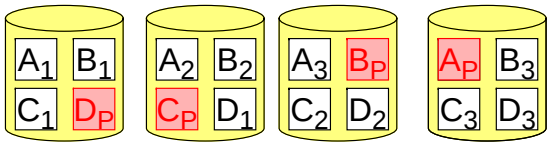
Einsatz mehrerer redundanter Platten (4)

- Nachteil von RAID 4
  - jeder Schreibvorgang erfordert auch das Schreiben des Paritätsblocks
  - Erzeugung des Paritätsblocks durch Speichern des vorherigen Blockinhalts möglich: $P_{neu} = P_{alt} \oplus B_{alt} \oplus B_{neu}$ (P=Parity, B=Block)
  - Schreiben eines kompletten Streifens benötigt nur einmaliges Schreiben des Paritätsblocks
  - Paritätsplatte ist hoch belastet

Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

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 (verkraftet 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

Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.