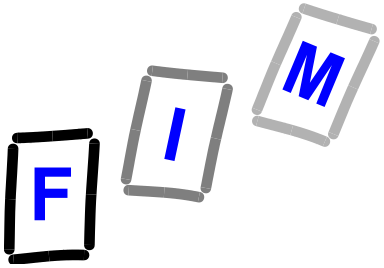
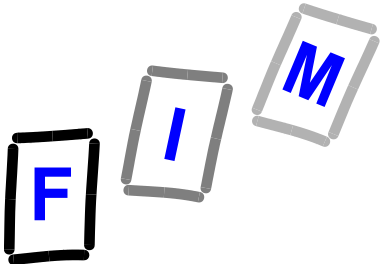


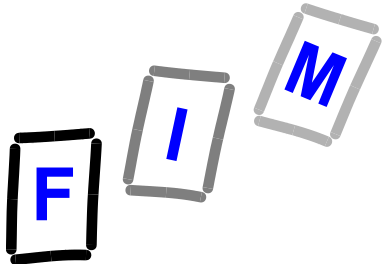
Betriebssysteme

Teil 11: Files, Filesysteme Allgemeines



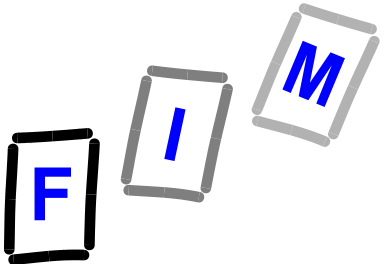
Allgemeines

- **Dateiverwaltung**
  - ➔**Aufgabe des BS, Dateien zu verwalten**
  - ➔***Dateisystem*** **(**File System**)**
  - ➔**Zugehörige logische Speichereinheit: Datei; (der/das) File**



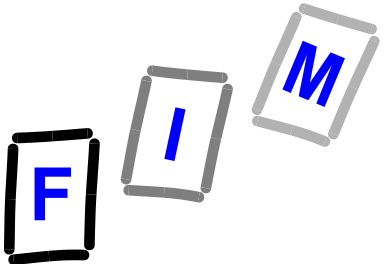
Verbergen von Eigenschaften der Datenträger

- **Das Betriebssystem schafft eine einheitliche logische Sicht von Daten**
  - ➔ **Abstraktion von physikalischen Eigenschaften des (nichtflüchtigen) Speichermediums/ Speichergerät**
  - ➔ **Logische Einheit: File**



Datei, File

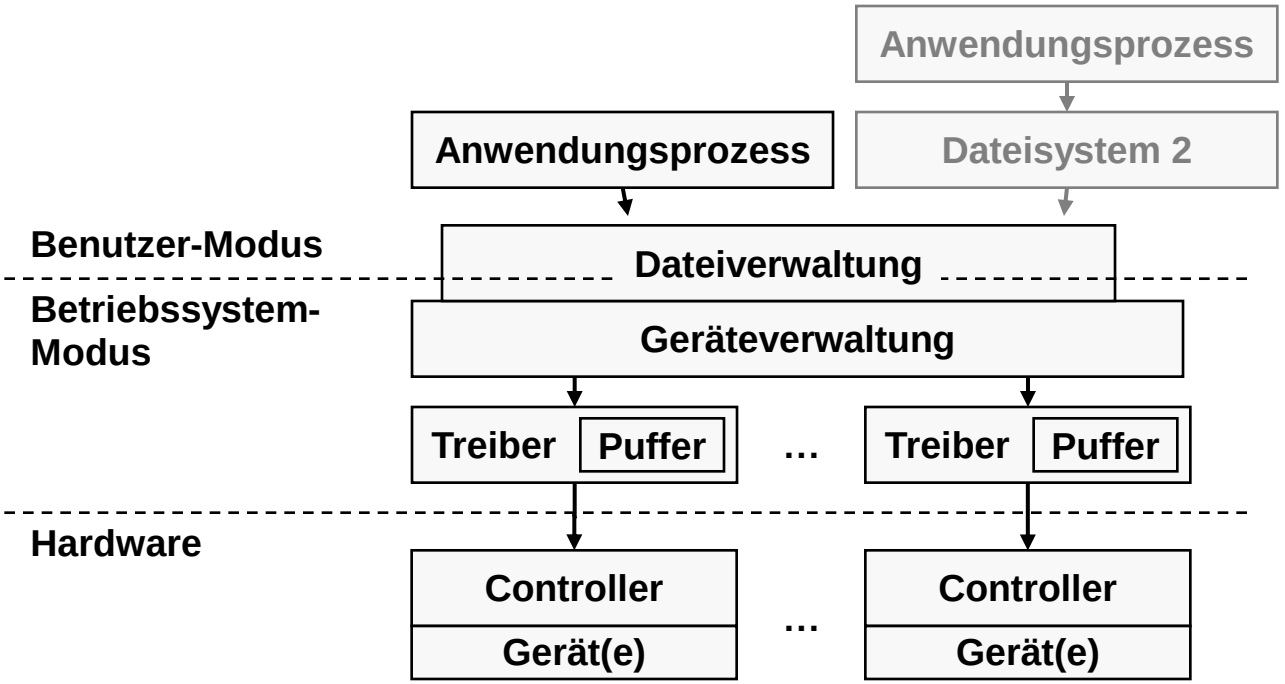
- ➔**Folge von Bytes**
  - » **Zusammenfassung in Blöcke**
  - » **Werden gruppenweise als Cluster übertragen**
- ➔**Residiert normalerweise auf einem nichtflüchtigen Medium**
  - » **Inhalt bleibt *persistent***
  - » **Vgl. jedoch: RAM – Disk**
- ➔**Enthält zusätzlich *Metadaten***

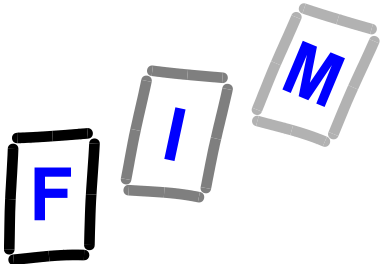


Layered file system
Schichten-Konzept

- File System besteht aus mehreren Schichten
  - ➔Schichtenkonzept extrem umgesetzt
- Oberste Schicht
  - ➔Dateiverwaltung
- Unterste Schicht:
  - ➔ I/O Kontrolle
    - » Treiber (device drivers)
    - » Interrupt handlers

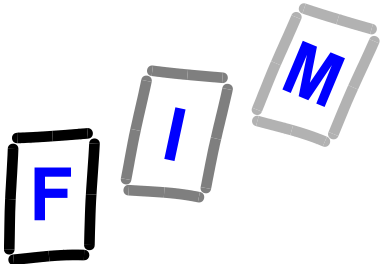
Einordnung in Schichten





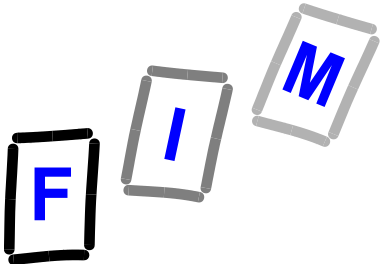
Besondere Aufgaben

- Persistenz
- Konsistenz
- Mutual Exclusion beim Zugriff
- Journaling File Systeme
  - ➔ Mitschreiben der Transaktionen
- Fragmentierung klein halten
  - ➔ Reorganisations-Utilities bzw. Strategien beim Schreiben
- Spezialaufgaben: Verschlüsselung, Komprimierung, Verstecken, …



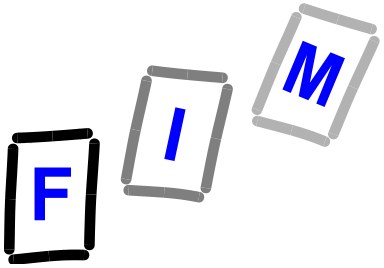
Operationen auf/mit Files

- **Einrichten:** creating a file
- **Schreiben:** writing a file
- **Anhängen:** appending to a file
- **Lesen:** reading a file
- **Löschen:** deleting a file
- **Satzzeiger positionieren:**
  repositioning within a file
  - ➔ **Wenn vom System unterstützt:**
    Random Access:
    Satzzeiger auf Position „i“ stellen, i-ten Satz lesen/schreiben
    - » **Heute eher: i-tes Byte (Satz: Record-orientierte Dateien)**
- **Abschneiden:** truncating a file
  - ➔ **Inhalt löschen, Länge auf 0,**
    aber File mit allen anderen Attributen behalten



Metadaten

- **NAME und TYPE: Siehe später**
- **LOCATION: Zeiger zu Device und logischer bzw. physischer Adresse, wo File gespeichert ist**
- **SIZE: Größe**
- **PROTECTION: Information, *wer was* mit dem File machen darf**
  - ➔ **Eigentümer, read only,…… .**
- **Datum- und Zeitangaben:**
  - ➔ **Wann erzeugt, letzter Zugriff ,…..**



FILE Typen

- **Frage:**

  Soll das BS an Hand bestimmter Merkmale den Typ eines Files erkennen?

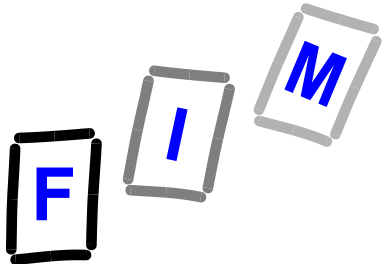
  - ➔ **Text file (*.txt): Ausführung (execute) zwecklos**
  - ➔ **Typ *.bin oder *.exe: Drucken macht (normalerweise) keinen Sinn**

- **Hinweis:**

  Bei ZIP (compress) auf *.zip, ist Typinformation

  - ➔ **entweder verloren**
  - ➔ **oder “unzip” vorher erforderlich**

- **Zusatz: Geht dies das OS etwas an? Oder nur die “Shell”?**
  - ➔ **Was ist bei Windows die “Shell”?**



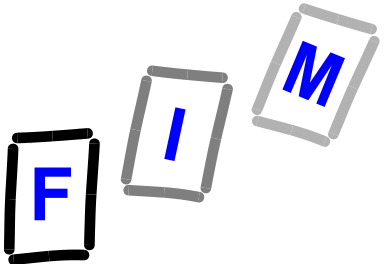
Lösungsansätze (1)

- Unterteile Filenamen in <filename>.<extension>
  - ➔<extension> ist Acronym für den Typ
  - ➔typisch: *.txt; *.exe; *.mod; *.doc; *.lib;
- Assoziiere mit <extension> Prozesse

  clicking *.txt Aufruf eines Editors

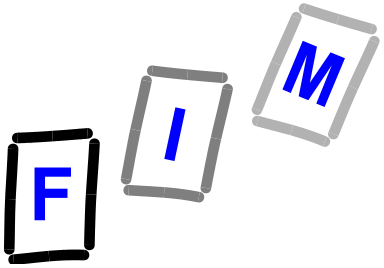
  clicking *.exe File Laden & ausführen

  . . . .
- Was ist mit „*.txt.exe“ ???



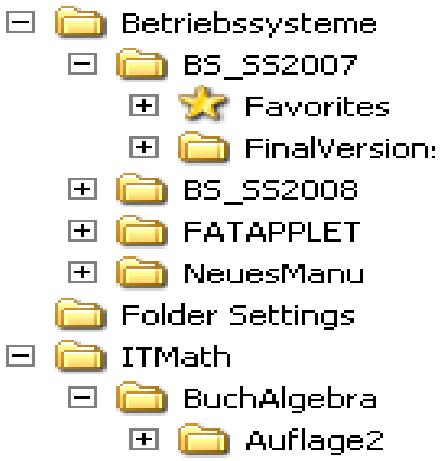
Lösungsansätze (2)

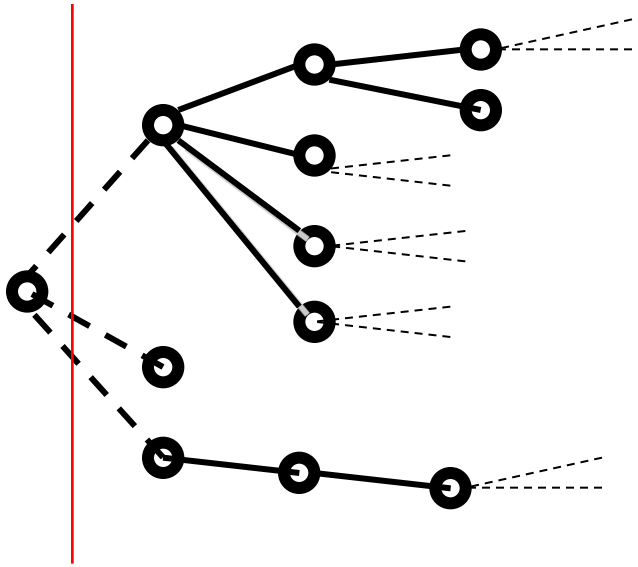
- **Einige BS wie APPLE OS <=9.x, MacOS X**
  - ➔ **unterstützen *.<extension> nicht, aber**
  - ➔ **der "Erzeuger" (creator) des Files wird zu den Metadaten hinzugefügt**
- **Oder *.asm; *.com; *.exe; . . .**
  sind nur empfohlene extensions,
  aber *nicht verpflichtend*
  - ➔ **Z.B.: MS-DOS**
- **Windows Familie**
  - ➔ **In der Registry wird eingetragen, welches Programm aufzurufen ist**
  - ➔ **Kann jederzeit geändert werden**

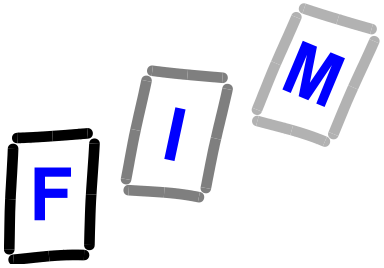


Verzeichnis Directory

- Für Suchfunktionen im Filesystem bedarf es eines Verzeichnisses: Directory
- Üblich: Baumstruktur







Baumstruktur

- Knoten können sein
  - ➔Files
  - ➔Ordner (Folder, Container)
    - » Enthalten wieder: Files und/oder Ordner
- Ordner dienen zur übersichtlichen Strukturierung
  - ➔Zugeordnete Zugriffsrechte werden vererbt
- Wege im Baum erklären Namensraum

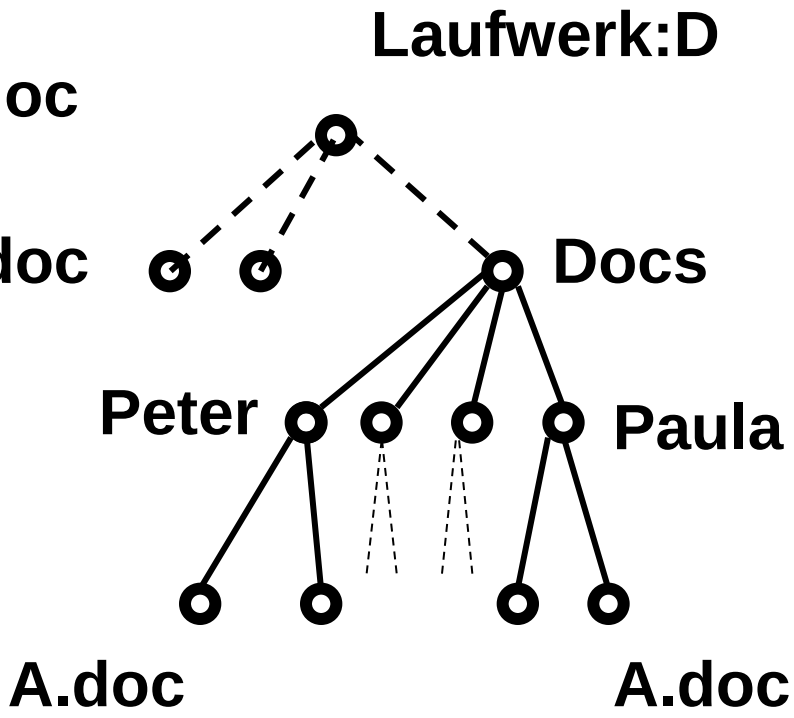
Namensraum

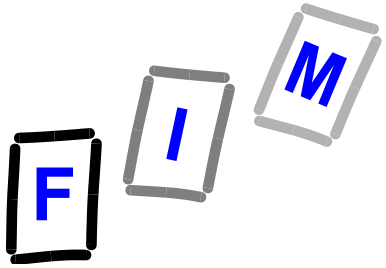
Delimiter „\“ oder „/“

D:\Docs\Peter\A.doc

≠

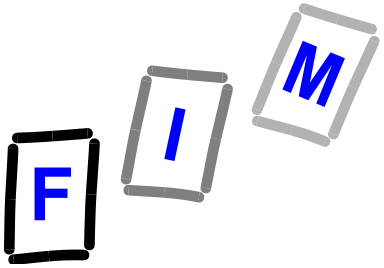
D:\Docs\Paula\A.doc





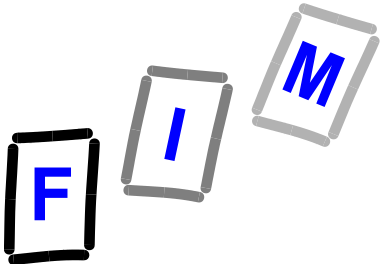
Directory Implementierung
≠
Physische Anordnung der Files

- **Verzeichnis bietet “nur” eine Sicht**
  - ➔ **Allgemein (für alle Anwender)**
  - ➔ **Für jeden einzelnen User individuell**
- **Tatsächliche Anordnung ist anders**
  - ➔ **Hängt vom Typ des Massenspeichers ab**
- **Probleme:**
  - ➔ **Fragmentierung der Blöcke**
  - ➔ **Effektiver Baumdurchlauf zum Suchen**



File-Allokation

- Blockgröße üblich
  - ➔512 Bytes bei Disks (Neu: 4096 Bytes)
  - ➔2048 bei optischen Medien (CD, DVD)
- Formatierung notwendig
  - ➔Bei Platten (disks)
    - » Einteilung in konzentrische Kreise (Spuren)
    - » Unterteilung in Partitionen möglich
  - ➔Formatierung legt auch Clustergröße fest
    - » Abhängig vom Filessystem
    - » Typischer Wert: 4096 Bytes = 8 Blöcke

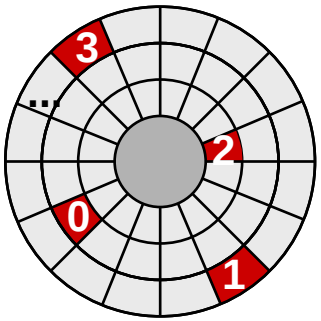


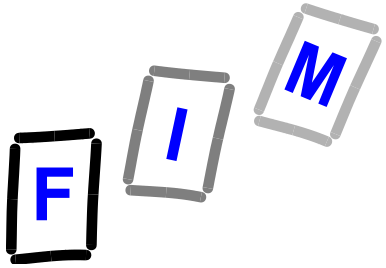
Interne Fragmentierung

- File als Folge von Bytes
- Unterteilt in Blöcke
- Letzter Block im Mittel nur halb voll
  - ➔Ähnlich wie bei Seitengröße
  - ➔Große Blöcke versus kleine Blöcke

Externe Fragmentierung

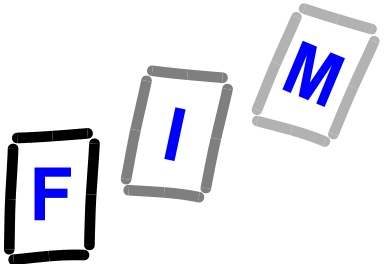
- **Übergeordnetes Ziel:**
  - ➔ **Folge der Blöcke sollte nicht am Speicher verstreut sein**
    - » **Beschleunigung beim Zugriff**
    - » **Mehrere Blöcke als Cluster in einem lesen/schreiben**
  - ➔ **Bei Platten: Lesekopfbewegung minimieren**





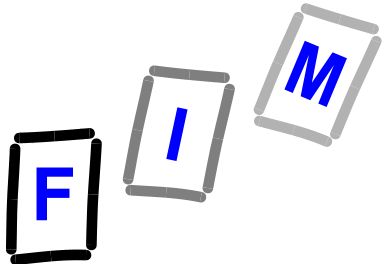
Externe Fragmentierung

- File zusammenhängend abspeichern
  - \+ leicht machbar, wenn Filegröße bekannt
  - ? Files werden angelegt, gelöscht ,… es entstehen Lücken
  - ? Files wachsen und schrumpfen
- Bedarf an Disk - Defragmentierungs- Programmen



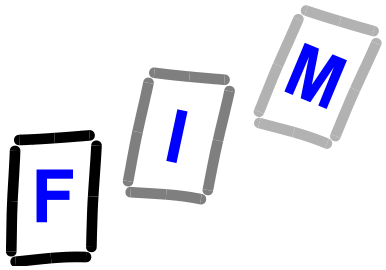
Flash-Speichermedien

- **Defragmentierung nötig?**
  - ➔ **Zugriffszeit für Cluster hängt von dessen Position ab, und wo der Lesekopf jetzt ist**
  - ➔ **Bei Flash-Speicher fällt dies weg!**
- **Dafür gibt es neue Probleme!**
  - ➔ **Wie teile ich dem Gerät mit, dass ein Cluster nicht mehr benötigt wird?**
    - » **„TRIM“-Kommando**
  - ➔ **Wie stelle ich sicher, dass Daten überschrieben werden?**
    - » **„SECURE ERASE“-Kommando**
    - » **Verschlüsselt speichern + „Entsorgung“ (Wie?) der Keys**



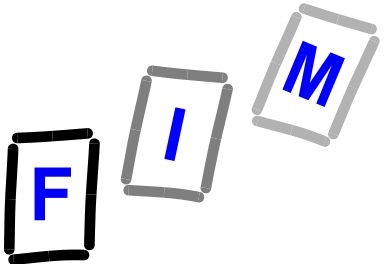
Welche Dateisysteme gibt es?

- FAT (File Allocation Table), 16 Bit DOS-System, FAT16
- FAT32, Windows 95B (OSR2)
- NTFS, Windows >=NT, 32-Bit-Dateisystem
- ext, ext2, ext3, ext4, ReiserFS, XFS, JFS, GPFS, Linux
- HPFS (High Performance File System), OS/2, 32-Bit-Dateisystem
- NetWare, eigenes 32-Bit-Dateisystem von Novell
- ISO 9660 für CD-ROM und ISO 13346 für DVD
- UDF (Universal Disk Format) ist für Speichermedien mit einer großen Kapazität gedacht, wie z.B. DVD-RAM
  - ➔ ISO 13346; Nachfolger von ISO 9660
- etc.



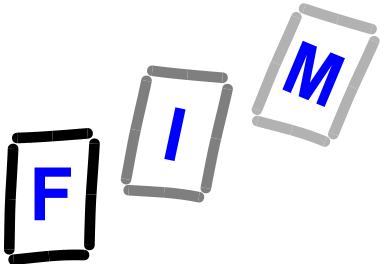
Welche Dateisysteme gibt es?

- **BtrFS: Basiert auf B-Bäumen (wie NTFS); ähnlich ZFS**
  - ➔ **Copy-on-Write: Geänderte Blöcke werden nicht überschrieben sondern neu angelegt ➔ Transaktionen**
  - ➔ **Snapshots: „Einfrieren“ ➔ Spätere Änderungen werden separat gespeichert (Versionierung; oder zB für Backups!)**
  - ➔ **Integrierte Spiegelung/RAID, Prüfsummen, …**
- **CarvFS: „Virtuelles“ Dateisystem für Forensik**
  - ➔ **Dateiname bestimmt, welche Cluster zu einer Datei gehören**
  - ➔ **Read-Only**
- **F2FS: Flash-Friendly File System**
  - ➔ **Basiert auf „logs“: Copy-on-Write; sequentielles Schreiben; bei Bedarf zusammenfassen**
  - ➔ **Großer inode (4kB!) mit vielen direkten Block-Pointern**
- **FUSE: Filesystem in Userspace**
  - ➔ **Für eigene Dateisysteme (Viele bereits vorhanden!)**



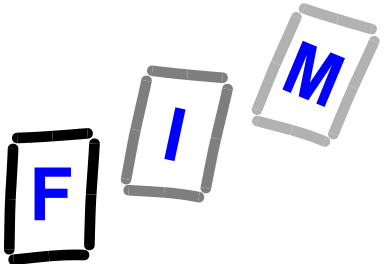
Journaling (1)

- **Eine „Datei“ ist nur ein Abstraktionselement. Sie besteht einerseits aus Meta-Information (Dateiname, in welchem Verzeichnis, Größe, etc.) und den eigentlichen Daten, die eine logische Einheit darstellen, physisch aber meist in mehreren Blöcken (Cluster, etc.) auf der Festplatte verteilt liegen.**
- **Das bedeutet, dass das Erstellen einer Datei aus mehreren Schritten besteht:**
  - ➔ **Anlegen der Metadaten im Directory-Teil**
  - ➔ **Reservieren der Cluster**
  - ➔ **Schreiben der eigentlichen Daten**
  - ➔ **Aktualisieren der Verwaltung über verfügbaren Speicher**
  - ➔ **Etc.**



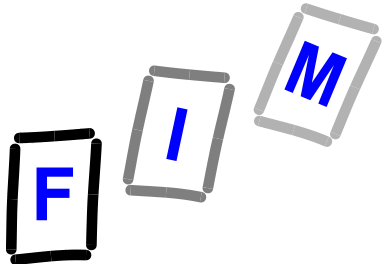
Journaling (2)

- **In vielen Filesystemen wird dies in unabhängigen sequentiellen Operationen durchgeführt**
  - ➔ **Problem der Möglichkeit von Inkonsistenzen**
    - » **Vgl. Datenbanken!**
    - » **ScanDisk/fsck zum Beheben solcher Probleme [oft mit Datenverlust verbunden!] wird wahrscheinlich jeden schon die eine oder andere halbe Stunde genervt haben!**
  - ➔ **Gründe: Absturz, Stromausfall, …**
- ***Lösung*: Eine Operation auf das Dateisystem wird in einer „Transaktion“ ausgeführt. D.h., ein Zugriff wird entweder vollständig durchgeführt oder gar nicht.**
  - ➔ **Vorteile:**
    - » **Keine Inkonsistenzen**
    - » **Schnelles Wiederherstellen eines konsistenten Zustands beim Neustart des Systems nach vorhergehendem Absturz**



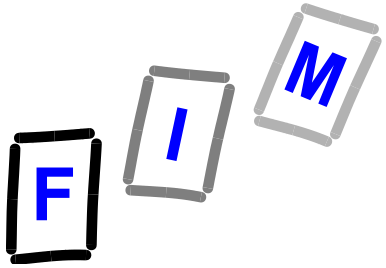
Journaling (3)

- **Windows**
  - ➔ **FAT unterstützt kein Journaling**
  - ➔ **NTFS (alle Versionen) unterstützen Journaling**
- **UNIX(e)**
  - ➔ **ext, ext2 haben kein Journaling**
  - ➔ **Bei ext3 wird eine Journaling Schicht über ext2 gelegt, die das Journaling durchführt**
    - » **Großer Vorteil der Kompatibilität zu ext2!**
  - ➔ **JFS, ReiserFS, XFS, BtrFS unterstützen Journaling schon seit ihrem Beginn direkt**
    - » **Die meisten anderen neueren Dateisysteme ebenso**



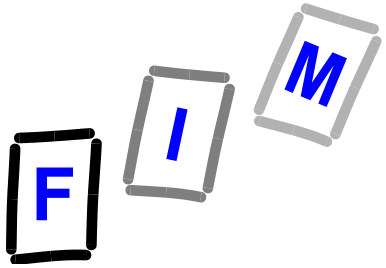
Log-structured Dateisysteme

- **Idee: Wenn wir schon Änderungen in ein Log schreiben müssen, warum noch zusätzl. an die "Originalstelle"?**
  - ➔ **Könnten wir nicht NUR mit dem Log auskommen?**
- **Das sind „log-structured file systems"!**
- **Grundprinzipien:**
  - ➔ **Daten werden nie überschrieben/verändert/gelöscht**
  - ➔ **Änderungen werden als Logeinträge am Ende angehängt**
  - ➔ **Dies betrifft auch die Verwaltungsinformationen, die ebenfalls nicht überschrieben werden. → „Rück"verkettung um ältere Teile zu finden**
  - ➔ **Große Speicher-Caches (für Verwaltungs- und Inhaltsdaten)**
    - » **Bzw SSD: Lesen ist sehr schnell und unabhängig von der Position**
  - ➔ **Periodische „Garbage-Collection" erforderlich, um unbenutzte „Logeinträge" (=gelöscht/überschrieben) wieder freizugeben**



Log-structured Dateisysteme

- **Vorteile:**
  - ➔ **Ermöglicht Versionierung: Alte Versionen der Datei existieren immer noch; sie müssen nur bei der GC ausgelassen werden**
  - ➔ **Crash-Recovery ist einfacher: Nur das Ende des Logs ist zu untersuchen und rückgängig zu machen/neu durchzuführen**
  - ➔ **Schnelleres Schreiben, da ausschließlich große sequentielle Blöcke geschrieben werden (Write-Caching!)**
    - » **Besonderes Problem bei RAID (siehe später): Einen Datenblock zu schreiben erfordert sonst sehr Schreibvorgänge: Index-Block, Indirekter-Index-Block, Inode/Block-Bitmap, Datenblock (RAID: auf vielen phys. Disks); Auf Harddisks wegen Rotations-Verzögerung problematisch!**
  - ➔ **Nie veränderte Dateien können sehr schnell gelesen werden, da sie unfragmentiert sind**
    - » **Oder nur äußerst gering: Interne Organisation in „Großbereiche“**

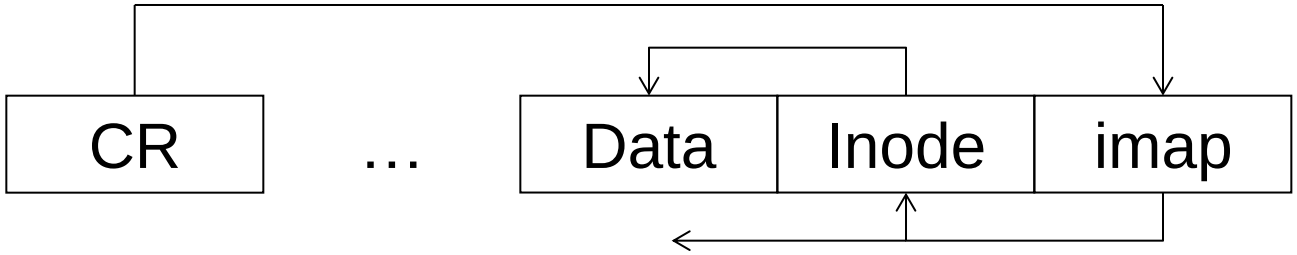


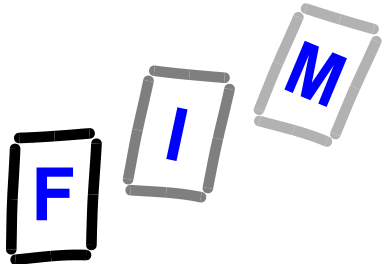
Log-structured Dateisysteme

- **Nachteile:**
  - ➔ **Lesen oft veränderter Dateien ist langsam, da eine veränderte Datei niemals am Stück gespeichert ist**
    - » **Aber: „Große Caches“ sowie „SSD“; ebenso Garbage Collection!**
  - ➔ **Garbage Collection ist langwierig oder kompliziert**
    - » **Große unveränderte Dateien jedes mal komplett umkopieren?**
    - » **Optimiert durch Einführung von „Segments“: GC nimmt M Segmente und macht daraus N neue Segmente, die an einen anderen Platz geschrieben werden, dann werden die M Segmente freigegeben**
    - » **Die Auswahl von „M“ ist die „Magie“ hier! Selten geänderte Daten sollten nicht bearbeitet werden, sondern eher die, die oft verändert wurden**
  - ➔ **Funktioniert umso schlechter, je voller das Dateisystem ist**
    - » **Maximum-Empfehlung: Ca. 90% voll. Darüber wird es meist „schlecht“!**
- **Beispiele: UDF (Optical disks), JFFS(2), YAFFS, …**

Log-structured FS: Funktionsweise

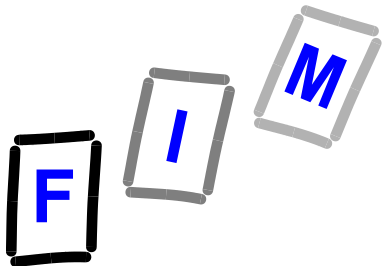
- **Schreiben von Daten:**
  - ➔ **Anhängen der Daten am Ende**
  - ➔ **Anhängen des (neuen/veränderten) Inodes am Ende**
  - ➔ **Anhängen der veränderten imap am Ende**
    - » **„Imap“ = Verzeichnis der Inodes (um diese finden zu können!)**
  - ➔ **Checkpoint Region (CR): Fixer Platz auf dem Datenträger mit Zeiger zur (aktuellen) imap**
    - » **Wird nur selten aktualisiert (zB alle 30 Sekunden); gecacht**
    - » **Absturz: Rekonstruktion mittels Durchgehen aller Logeinträge seit der letzten Aktualisierung der CR**





Berechtigungen (1) FAT

- Unter DOS wird das Standard-FAT Dateisystem verwendet, welches keinerlei Berechtigungsvergabe für einzelne Dateien oder Verzeichnisse ermöglicht
  - ➔ Es gibt nur „globale“ Schalter
    - » „Schreibgeschützt“: Kein Schreibzugriff möglich
    - » „Archiv“: Nur als Hinweis für Backup-Programme
    - » „Versteckt“: Normalerweise nicht angezeigt
    - » „System“: Besonders geschützt
  - ➔ Die jedem Benutzer betreffen, und
  - ➔ von jedem Benutzer geändert werden können!

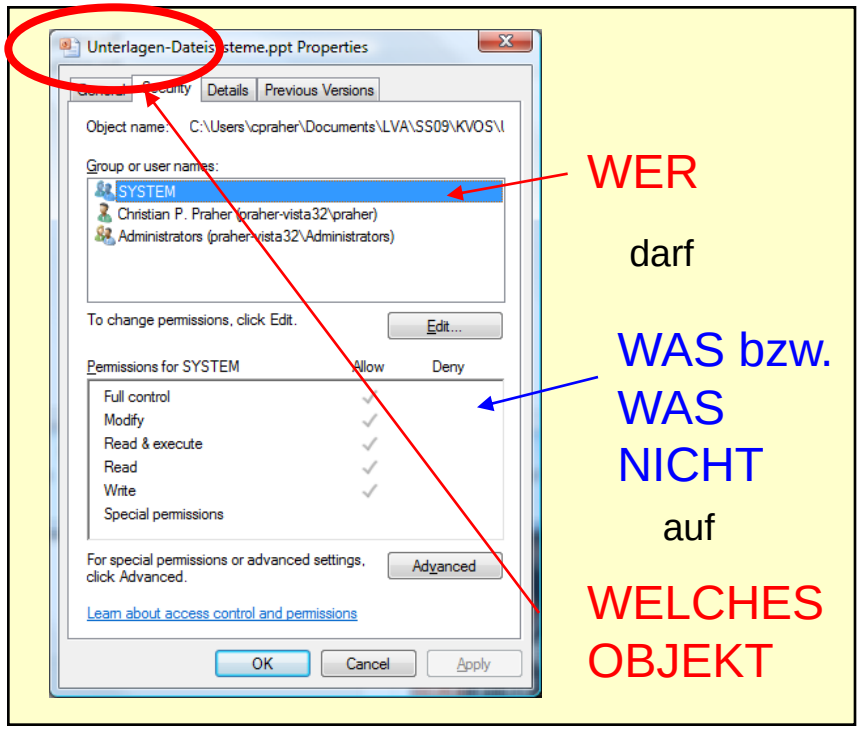


Berechtigungen (2) NTFS

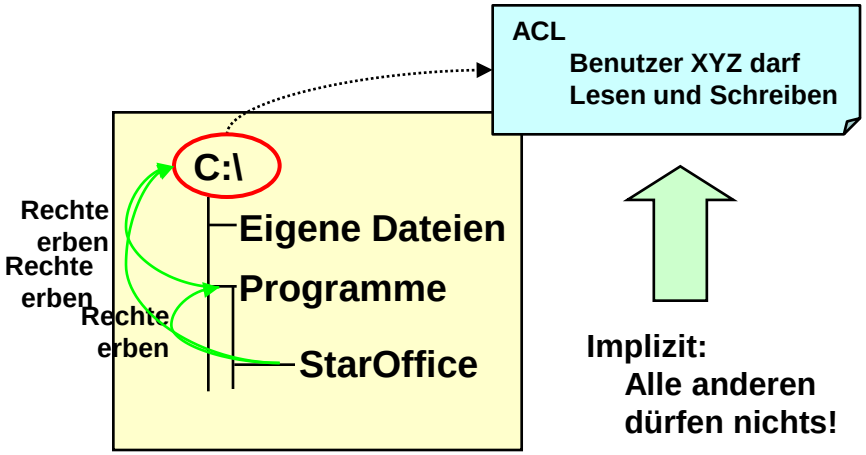
- Seit WinNT wird NTFS (New Technology File System) eingesetzt:
  - ➔ Feingranulare Berechtigungsvergabe durch Access-Control-Lists (ACL)
  - ➔ Zu jeder Datei werden Information gespeichert:
    - » Welcher Benutzer bzw. welche Gruppe welches Recht
      - auf die Datei besitzt (grant)
      - explizit verweigert wird (deny)
  - ➔ Rechtevererbung (Ordner auf enthaltene Objekte bzw. Unterordner) spielt hierbei eine wichtige Rolle zur Verwaltungsvereinfachung

Berechtigungen (3) NTFS contd.

NTFS Rechte-Dialog



NTFS Rechte-Vererbung

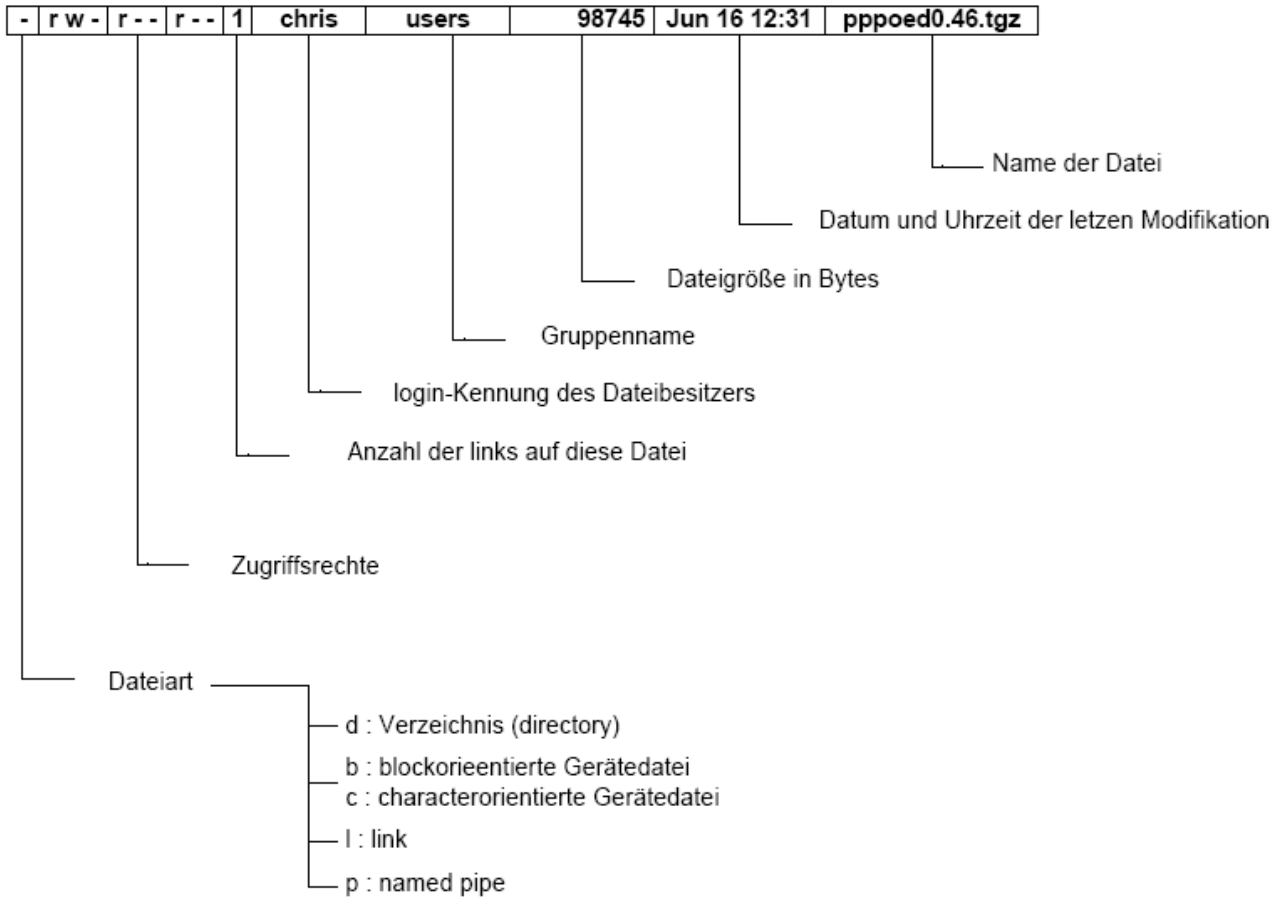


Berechtigungen (4) ext*

- **Unix bzw. Linux bietet traditionellerweise ein sehr einfaches Rechtekonzept:**
  - ➔ **Es gibt Benutzer und Gruppen**
  - ➔ **Jeder Benutzer ist einer Primärgruppe und beliebig vielen Sekundärgruppen zugeordnet**
  - ➔ **Ein spezieller Benutzer („root“) hat alle Rechte auf alle (normalen) Dateien bzw. kann sich diese verschaffen**
  - ➔ **Jede Datei/Verzeichnis hat genau einen Besitzer und eine „besitzende Gruppe“**
  - ➔ **Es gibt lediglich 3 Rechte: "read", "write" und "execute"**
  - ➔ **Eine Kombination dieser 3 Rechte kann getrennt für 3 Personengruppen definiert werden: für Besitzer, für besitzende Gruppe und für alle anderen**
  - ➔ **Weiters existieren einige Spezialbits (Ausführen als Besitzer, als besitzende Gruppe, etc.)**

Berechtigungen (5) ext*

Befehl: **`ls -al`**



Berechtigungen (6)
ACL für Linux

- ACLs gibt es inzwischen auch für Linux
  - ➔ Basierend auf POSIX 1003.1e
  - ➔ ext2, ext3, XFS, JFS (inkludiert im Kernel 2.6), ext4, …
- Berechtigungen (rwx) können Dateien zusätzlich für beliebige weitere Benutzer/Gruppen zugeordnet werden
  - ➔ Kommandos: getfacl, setfacl
- Beispiel:
  - ➔ "getfacl index.html"
  - ➔ # file: index.html
    # owner: root
    # group: apache
    user::rw-
    user:sonntag:rwx
    group::r--
    other::---

Achtung: Dateisystem muss uU erst auf Benutzung von ACLs umgestellt werden (/etc/fstab !)