

## 3. Foliensatz

# Betriebssysteme und Rechnernetze

Prof. Dr. Christian Baun

Frankfurt University of Applied Sciences  
(1971–2014: Fachhochschule Frankfurt am Main)  
Fachbereich Informatik und Ingenieurwissenschaften  
[christianbaun@fb2.fra-uas.de](mailto:christianbaun@fb2.fra-uas.de)

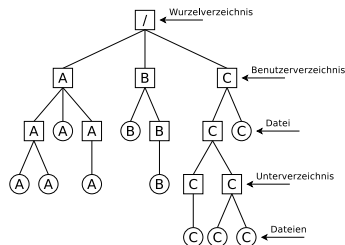
# Lernziele dieses Foliensatzes

- Am Ende dieses Foliensatzes kennen/verstehen Sie...
  - die **Aufgaben und Grundbegriffe von Dateisystemen**
  - was **Inodes** und **Cluster** sind
  - wie **Blockadressierung** funktioniert
  - den **Aufbau** ausgewählter Dateisysteme
  - eine Übersicht über die **Windows-Dateisysteme** und deren Eckdaten
  - was **Journaling** ist und warum es viele Dateisysteme heute implementieren
  - wie **Extent-basierte Adressierung** funktioniert und warum zahlreiche moderne Betriebssysteme diese verwenden
  - was **Copy-On-Write** ist
  - ~~wie **Defragmentierung** funktioniert und wann es sinnvoll ist zu defragmentieren~~

Übungsblatt 3 wiederholt die für die Lernziele relevanten Inhalte dieses Foliensatzes

# Dateisysteme...

- organisieren die Ablage von Dateien auf Datenspeichern
  - Dateien sind beliebig lange Folgen von Bytes und enthalten inhaltlich zusammengehörende Daten
- verwalten Dateinamen und Attribute (Metadaten) der Dateien
- bilden einen Namensraum
  - Hierarchie von Verzeichnissen und Dateien



- Absolute Pfadnamen:** Beschreiben den kompletten Pfad **von der Wurzel bis zur Datei**
- Relative Pfadnamen:** Alle Pfade, die **nicht mit der Wurzel** beginnen

- sind eine Schicht/Funktionalität des Betriebssystems
  - Prozesse und Benutzer greifen auf Dateien abstrakt über deren Dateinamen und nicht direkt auf Speicheradressen zu
- sollen wenig Overhead für Verwaltungsinformationen verursachen

# Technische Grundlagen der Dateisysteme

- Dateisysteme adressieren **Cluster** und nicht Blöcke des Datenträgers
  - Jede Datei belegt eine ganzzahlige Menge an Clustern
  - In der Literatur heißen die Cluster häufig **Zonen** oder **Blöcke**
    - Das führt zu Verwechslungen mit den Sektoren der Laufwerke, die in der Literatur auch manchmal Blöcke heißen
- Die Größe der Cluster ist wichtig für die Effizienz des Dateisystems
  - Je kleiner die Cluster...
    - Steigender Verwaltungsaufwand für große Dateien
    - Abnehmender Kapazitätsverlust durch interne Fragmentierung
  - Je größer die Cluster...
    - Abnehmender Verwaltungsaufwand für große Dateien
    - Steigender Kapazitätsverlust durch interne Fragmentierung

Je größer die Cluster, desto mehr Speicher geht durch interne Fragmentierung verloren

- Dateigröße: 1 kB. Clustergröße: 2 kB  $\Rightarrow$  1 kB geht verloren
- Dateigröße: 1 kB. Clustergröße: 64 kB  $\Rightarrow$  63 kB gehen verloren!

- Die Clustergröße kann man beim Anlegen des Dateisystems festlegen

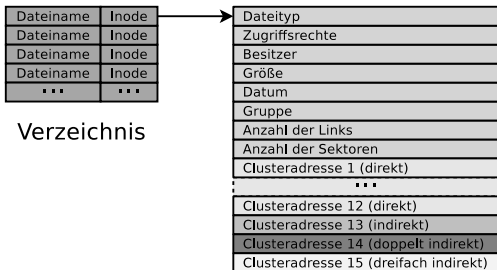
# Grundbegriffe von Linux-Dateisystemen

Unter Linux gilt: Clustergröße  $\leq$  Größe der Speicherseiten (Pagesize)

- Die Pagesize hängt von der Architektur ab
- x86 = 4 kB, Alpha und UltraSPARC = 8 kB, Apple Silicon = 16 kB, IA-64 = 4/8/16/64 kB
- Wird eine **Datei** angelegt, wird auch ein **Inode** (*Index Node*) angelegt
  - Er speichert die Verwaltungsdaten (*Metadaten*) einer Datei, außer dem Dateinamen
    - Metadaten sind u.a. Dateigröße, UID/GID, Zugriffsrechte und Datum
  - Jeder Inode hat eine im Dateisystem eindeutige Inode-Nummer
  - Im Inode wird auf die Cluster der Datei verwiesen
  - Alle Linux-Dateisysteme basieren auf dem Funktionsprinzip der Inodes
- Auch ein **Verzeichnis** ist eine Datei
  - Inhalt: Dateiname und Inode-Nummer für jede Datei des Verzeichnisses
- Arbeitsweise der Linux-Dateisysteme traditionell: **Blockadressierung**
  - Eigentlich ist der Begriff irreführend, weil Dateisysteme immer Cluster adressieren und nicht Blöcke (des Datenträgers)
    - Der Begriff ist aber seit Jahrzehnten in der Literatur etabliert

# Blockadressierung am Beispiel ext2/3/4

- Jeder Inode speichert die Nummern von bis zu 12 Clustern direkt



Inode

- Benötigt eine Datei mehr Cluster, wird indirekt adressiert mit Clustern, deren Inhalt Clusternummern sind
- Blockadressierung verwenden Minix, ext2/3/4, ReiserFS und Reiser4

## Gute Erklärung

<http://lwn.net/Articles/187321/>

- Szenario: Im Dateisystem können keine Dateien mehr erstellt werden, obwohl noch ausreichend freie Kapazität vorhanden ist
- Mögliche Erklärung: Es sind keine Inodes mehr verfügbar
- Das Kommando `df -i` zeigt, wie viele Inodes existieren wie viele noch verfügbar sind

# Direkte und indirekte Adressierung am Beispiel ext2/3/4

|           |       |
|-----------|-------|
| Dateiname | Inode |
| Dateiname | Inode |
| Dateiname | Inode |
| Dateiname | Inode |
| ...       | ...   |

Verzeichnis

|                                       |
|---------------------------------------|
| Dateityp                              |
| Zugriffsrechte                        |
| Besitzer                              |
| Größe                                 |
| Datum                                 |
| Gruppe                                |
| Anzahl der Links                      |
| Anzahl der Sektoren                   |
| Clusteradresse 1 (direkt)             |
| ...                                   |
| Clusteradresse 12 (direkt)            |
| Clusteradresse 13 (indirekt)          |
| Clusteradresse 14 (doppelt indirekt)  |
| Clusteradresse 15 (dreifach indirekt) |

Inode

Standardgröße bei ext2: 128 Bytes  
Standardgröße bei ext3/4: 256 Bytes

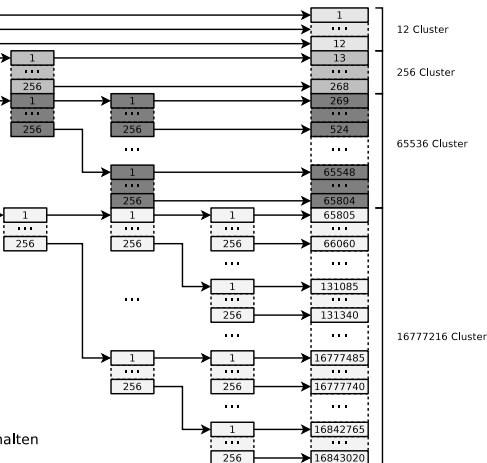
Clusternummern  
(Daten der Datei)

ext2/3 verwenden 32-Bit Cluster-Nummern  
ext4 verwendet 48-Bit Cluster-Nummern

Clustergröße: 1 kB

Ein Cluster kann 256 Adressen der Länge 4 Byte (32 Bit) enthalten

Maximale Dateigröße: 16 GB



# Minix

## Das Betriebssystem Minix

- Unix-ähnliches Betriebssystem
  - Wird seit 1987 von Andrew S. Tanenbaum als Lehrsystem entwickelt
  - Aktuelle Version: 3.3.0 aus dem Jahr 2014
- 
- Standard-Dateisystem unter Linux bis 1992
    - Naheliegend, denn Minix war die Grundlage der Entwicklung von Linux
  - Verwaltungsaufwand des Minix-Dateisystems ist gering
    - Sinnvolle Einsatzbereiche „heute“: Boot-Disketten und RAM-Disks
  - Ein Minix-Dateisystem enthält nur 6 Bereiche
    - Die einfache Struktur macht es für die Lehre optimal

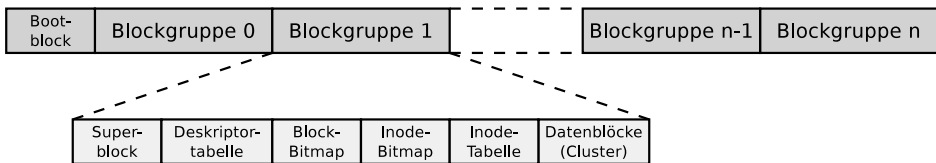


# Bereiche in einem Minix-Dateisystem

| Bereich 1                | Bereich 2                 | Bereich 3                    | Bereich 4                     | Bereich 5              | Bereich 6                    |
|--------------------------|---------------------------|------------------------------|-------------------------------|------------------------|------------------------------|
| Bootblock<br>(1 Cluster) | Superblock<br>(1 Cluster) | Inodes-Bitmap<br>(1 Cluster) | Cluster-Bitmap<br>(1 Cluster) | Inodes<br>(15 Cluster) | Daten<br>(Restliche Cluster) |

- **Bootblock.** Enthält den Boot-Loader, der das Betriebssystem startet
- **Superblock.** Enthält Informationen über das Dateisystem,
  - z.B. Anzahl der Inodes und Cluster
- **Inodes-Bitmap.** Enthält eine Liste aller Inodes mit der Information, ob der Inode belegt (Wert: 1) oder frei (Wert: 0) ist
- **Cluster-Bitmap.** Enthält eine Liste aller Cluster mit der Information, ob der Cluster belegt (Wert: 1) oder frei (Wert: 0) ist
- **Inodes.** Enthält Inodes mit den Metadaten
  - Jede Datei und jedes Verzeichnis ist von einem Inode repräsentiert, der Metadaten enthält
    - Metadaten sind u.a. Dateityp, UID/GID, Zugriffsrechte, Größe
- **Daten.** Hier ist der Inhalt der Dateien und Verzeichnisse
  - Das ist der größte Bereich im Dateisystem

## ext2/3



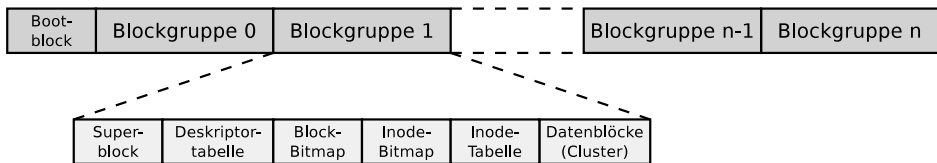
- Die Cluster des Dateisystems werden in **Blockgruppen** gleicher Größe zusammengefasst
  - Die Informationen über die Metadaten und freien Cluster jeder Blockgruppe werden in der jeweiligen Blockgruppe verwaltet

Maximale Größe einer Blockgruppe: 8x Clustergröße in Bytes

Beispiel: Ist die Clustergröße 1024 Bytes, kann jede Blockgruppe maximal 8192 Cluster umfassen

- Vorteil der Blockgruppen (bei Festplatten!): Die Inodes (Metadaten) liegen physisch nahe bei den Clustern, die sie adressieren
  - Das reduziert die Suchzeiten und den Grad der Fragmentierung
  - Bei Flash-Speicher ist die Position der Daten in den einzelnen Speicherzellen für die Zugriffsgeschwindigkeit irrelevant

# Schema der Blockgruppen bei ext2/3



- Der erste Cluster enthält den **Bootblock** (Größe: 1 kB)
  - Er enthält den Bootmanager, der das Betriebssystem startet
- Jede Blockgruppe enthält eine **Kopie des Superblocks**
  - Das verbessert die Datensicherheit
- Die **Deskriptor-Tabelle** enthält u.a.
  - Die Clusternummern des Block-Bitmaps und des Inode-Bitmaps
  - Die Anzahl der freien Cluster und Inodes in der Blockgruppe
- **Block-** und **Inode-Bitmap** sind jeweils einen Cluster groß
  - Sie enthalten die Information, welche Cluster und welche Inodes in der Blockgruppe belegt sind
- Die **Inode-Tabelle** enthält die Inodes der Blockgruppe
- Die restlichen Cluster der Blockgruppe sind für die **Daten** nutzbar

# Dateisysteme mit Dateizuordnungstabellen

Das Dateisystem FAT wurde 1980 mit QDOS, später umbenannt in MS-DOS, veröffentlicht

QDOS = Quick and Dirty Operating System

- Das Dateisystem File Allocation Table (FAT) basiert auf der gleichnamigen Datenstruktur, deren deutsche Bezeichnung Dateizuordnungstabelle ist
- Die FAT (**Dateizuordnungstabelle**) ist eine Tabelle fester Größe
- Für jeden Cluster des Dateisystems existiert ein Eintrag in der FAT mit folgenden Informationen über den Cluster:
  - Cluster ist frei oder das Medium an dieser Stelle beschädigt
  - Cluster ist von einer Datei belegt
    - In diesem Fall speichert er die Adresse des nächsten Clusters, der zu dieser Datei gehört oder er ist der letzte Cluster der Datei
- Die Cluster einer Datei bilden eine verkettete Liste (**Clusterkette**)  
⇒ siehe Folien 15 und 17

# Bereiche in einem FAT-Dateisystem (1/2)

| Bereich 1  | Bereich 2            | Bereich 3 | Bereich 4 | Bereich 5        | Bereich 6 |
|------------|----------------------|-----------|-----------|------------------|-----------|
| Bootsektor | Reservierte Sektoren | FAT1      | FAT2      | Stammverzeichnis | Daten     |

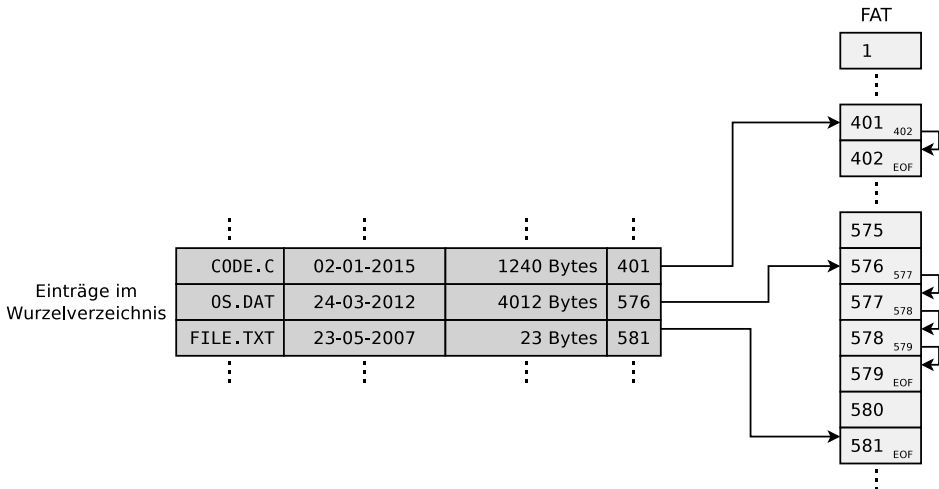
- Im **Bootsektor** liegen ausführbarer x86-Maschinencode, der das Betriebssystem starten soll, und Informationen über das Dateisystem:
  - Blockgröße des Speichermediums (512, 1.024, 2.048 oder 4.096 Bytes)
  - Anzahl der Blöcke pro Cluster
  - Anzahl der Blöcke (Sektoren) auf dem Speichermedium
  - Beschreibung des Speichermediums
  - Beschreibung der FAT-Version
- Zwischen Bootsektor und (erster) FAT können sich optionale **reservierte Sektoren**, z.B. für den Bootmanager, befinden
  - Diese Cluster können nicht vom Dateisystem benutzt werden

## Bereiche in einem FAT-Dateisystem (2/2)

| Bereich 1  | Bereich 2            | Bereich 3 | Bereich 4 | Bereich 5        | Bereich 6 |
|------------|----------------------|-----------|-----------|------------------|-----------|
| Bootsektor | Reservierte Sektoren | FAT1      | FAT2      | Stammverzeichnis | Daten     |

- In der **Dateizuordnungstabelle** (FAT) sind die belegten und freien Cluster im Dateisystem erfasst
  - Konsistenz der FAT ist für die Funktionalität des Dateisystems elementar
    - Darum existiert üblicherweise eine Kopie der FAT, um bei Datenverlust noch eine vollständige FAT als Backup zu haben
- Im **Stammverzeichnis** (Wurzelverzeichnis) ist jede Datei und jedes Verzeichnis durch einen Eintrag repräsentiert:
  - Bei FAT12 und FAT16 befindet sich das Stammverzeichnis direkt hinter der FAT und hat eine feste Größe
    - Die maximale Anzahl an Verzeichniseinträgen ist somit begrenzt
  - Bei FAT32 kann sich das Stammverzeichnis an beliebiger Position im Datenbereich befinden und hat eine variable Größe
- Der letzte Bereich enthält die eigentlichen **Daten**

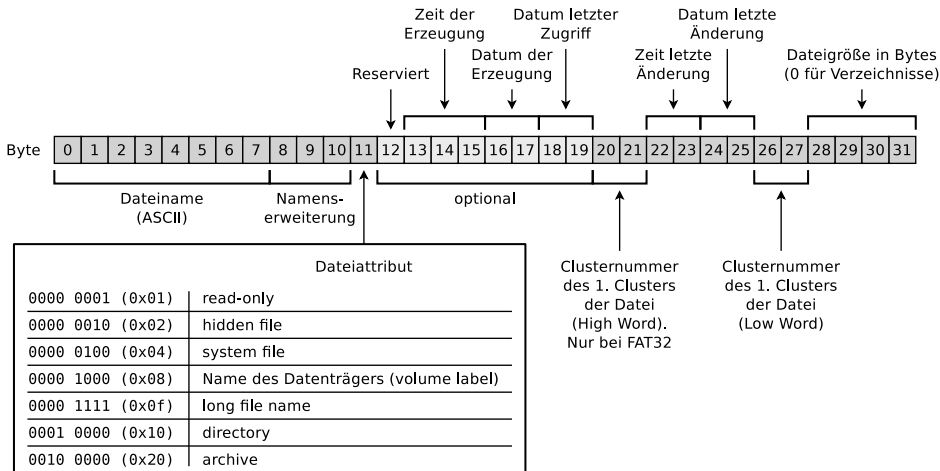
# Stammverzeichnis (Wurzelverzeichnis) und FAT



Das Thema FAT ist anschaulich erklärt bei...

- **Betriebssysteme**, Carsten Vogt, 1. Auflage, Spektrum Akademischer Verlag (2001), S. 178-179

# Struktur der Einträge im Stammverzeichnis



Warum ist 4 GB die maximale Dateigröße unter FAT32?

Es stehen nur 4 Bytes für die Angabe der Dateigröße zur Verfügung.

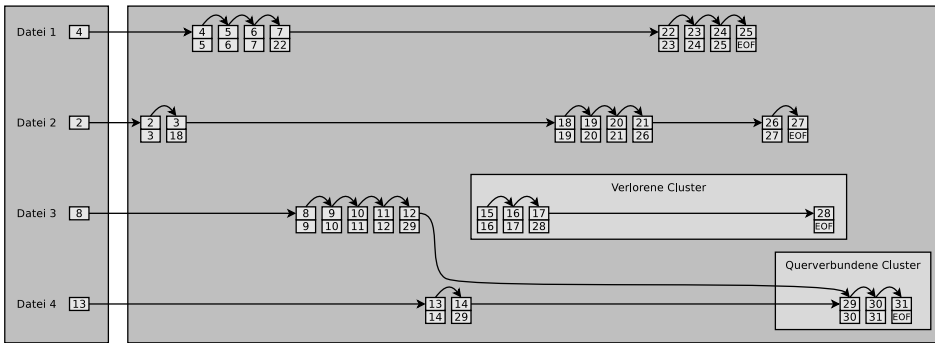


# Gefahr von Inkonsistenzen im Dateisystem

- Typische Probleme von Dateisystemen, die auf einer FAT basieren:
  - verlorene Cluster
  - querverbundene Cluster

Stammverzeichnis

Dateizuordnungstabelle (File Allocation Table)



Quelle: [http://www.sal.ksu.edu/faculty/tim/ossg/File\\_sys/file\\_system\\_errors.html](http://www.sal.ksu.edu/faculty/tim/ossg/File_sys/file_system_errors.html)

# FAT12

Erschien 1980 mit der ersten QDOS-Version

- Die Clusternummern sind 12 Bits lang
  - Maximal  $2^{12} = 4.096$  Cluster können adressiert werden
- Clustergröße: 512 Bytes bis 4 kB
- Unterstützt nur Speichermedien (Partitionen) bis 16 MB

$2^{12} * 4 \text{ kB Clustergröße} = 16.384 \text{ kB} = 16 \text{ MB maximale Dateisystemgröße}$

- Dateinamen werden nur im Schema 8.3 unterstützt
  - 8 Zeichen stehen für den Dateinamen und 3 Zeichen für die Dateinamenserweiterung zur Verfügung

Wird „heute“ nur für DOS- und Windows-Disketten eingesetzt

# FAT16

- Erschien 1983, da absehbar war, dass 16 MB Adressraum nicht ausreicht
- Maximal  $2^{16} = 65.536$  Cluster können adressiert werden
  - 12 Cluster sind reserviert
- Clustergröße: 512 Bytes bis 256 kB
- Dateinamen werden nur im Schema 8.3 unterstützt
- Haupteinsatzgebiet heute: mobile Datenträger  $\leq 2$  GB

| Partitionsgröße | Clustergröße |
|-----------------|--------------|
| bis 31 MB       | 512 Bytes    |
| 32 MB - 63 MB   | 1 kB         |
| 64 MB - 127 MB  | 2 kB         |
| 128 MB - 255 MB | 4 kB         |
| 256 MB - 511 MB | 8 kB         |
| 512 MB - 1 GB   | 16 kB        |
| 1 GB - 2 GB     | 32 kB        |
| 2 GB - 4 GB     | 64 kB        |
| 4 GB - 8 GB     | 128 kB       |
| 8 GB - 16 GB    | 256 kB       |

Die Tabelle enthält die Standard-Clustergrößen unter Windows 2000/XP/Vista/7. Die Clustergröße kann beim Erzeugen des Dateisystems festgelegt werden

Einige Betriebssysteme (z.B. MS-DOS und Windows 95/98/Me) unterstützen keine 64 kB Cluster

Einige Betriebssysteme (z.B. MS-DOS und Windows 2000/XP/7) unterstützen keine 128 kB und 256 kB Cluster

Quelle: <http://support.microsoft.com/kb/140365/de>

# FAT32

- Erschien 1997 als Reaktion auf die höhere Festplattenkapazität und weil Cluster  $> 32$  kB sehr viel Speicher verschwenden
- 32 Bits pro Eintrag in der FAT stehen für Clusternummern zur Verfügung
  - 4 Bits sind reserviert
  - Darum können nur  $2^{28} = 268.435.456$  Cluster adressiert werden
- Clustergröße: 512 Bytes bis 32 kB
- Maximale Dateigröße: 4 GB
  - Grund: Es stehen nur 4 Bytes für die Angabe der Dateigröße zur Verfügung
- Haupteinsatzgebiet heute: mobile Datenträger  $> 2$  GB

| Partitionsgröße | Clustergröße |
|-----------------|--------------|
| bis 63 MB       | 512 Bytes    |
| 64 MB - 127 MB  | 1 kB         |
| 128 MB - 255 MB | 2 kB         |
| 256 MB - 511 MB | 4 kB         |
| 512 MB - 1 GB   | 4 kB         |
| 1 GB - 2 GB     | 4 kB         |
| 2 GB - 4 GB     | 4 kB         |
| 4 GB - 8 GB     | 4 kB         |
| 8 GB - 16 GB    | 8 kB         |
| 16 GB - 32 GB   | 16 kB        |
| 32 GB - 2 TB    | 32 kB        |

Die Tabelle enthält die Standard-Clustergrößen unter Windows 2000/XP/Vista/7. Die Clustergröße kann beim Erzeugen des Dateisystems festgelegt werden

# Längere Dateinamen durch VFAT

- VFAT (Virtual File Allocation Table) erschien 1997
  - Erweiterung für FAT12/16/32, die längere Dateinamen ermöglicht
- Durch VFAT wurden unter Windows erstmals...
  - Dateinamen unterstützt, die nicht dem Schema 8.3 folgen
  - Dateinamen bis zu einer Länge von 255 Zeichen unterstützt
- Verwendet die Zeichenkodierung Unicode

## Lange Dateinamen – Long File Name Support (LFN)

- VFAT ist ein interessantes Beispiel für die Realisierung einer neuen Funktionalität unter Beibehaltung der Abwärtskompatibilität
- Lange Dateinamen (max. 255 Zeichen) werden auf max. 20 Pseudo-Verzeichniseinträge verteilt (siehe Folie 22)
- Dateisysteme ohne Long File Name Support ignorieren die Pseudo-Verzeichniseinträge und zeigen nur den verkürzten Namen an
- Bei einem VFAT-Eintrag in der FAT, haben die ersten 4 Bits im Feld **Dateiattribute** den Wert 1 (siehe Folie 15)
- Besonderheit: Groß/Kleinschreibung wird angezeigt, aber ignoriert

# Kompatibilität zu MS-DOS

- VFAT und NTFS (siehe Folie 34) speichern für jede Datei einen eindeutigen Dateinamen im Format 8.3
  - Betriebssysteme ohne die VFAT-Erweiterung ignorieren die Pseudo-Verzeichniseinträge und zeigen nur den verkürzten Dateinamen
    - So können Microsoft-Betriebssysteme ohne VFAT-Unterstützung auf Dateien mit langen Dateinamen zugreifen
- Problem: Die kurzen Dateinamen müssen eindeutig sein
- Lösung:
  - Alle Sonderzeichen und Punkte innerhalb des Namens werden gelöscht
  - Alle Kleinbuchstaben werden in Großbuchstaben umgewandelt
  - Es werden nur die ersten 6 Zeichen beibehalten
    - Danach folgt ein ~1 vor dem Punkt
  - Die ersten 3 Zeichen hinter dem Punkt werden beibehalten und der Rest gelöscht
  - Existiert schon eine Datei gleichen Namens, wird ~1 zu ~2, usw.
- Beispiel: Die Datei Ein ganz langer Dateiname.test.pdf wird unter MS-DOS so dargestellt: EINGAN~1.pdf

# FAT-Dateisysteme analysieren (1/3)

```
# dd if=/dev/zero of=./fat32.dd bs=1024000 count=34
34+0 Datensätze ein
34+0 Datensätze aus
34816000 Bytes (35 MB) kopiert, 0,0213804 s, 1,6 GB/s
# mkfs.vfat -F 32 fat32.dd
mkfs.vfat 3.0.16 (01 Mar 2013)

# mkdir /mnt/fat32
# mount -o loop -t vfat fat32.dd /mnt/fat32/

# mount | grep fat32
/tmp/fat32.dd on /mnt/fat32 type vfat (rw,relatime,fmask=0022,dmask=0022,codepage=437,ioccharset=utf8,shortname
=mixed,errors=remount-ro)
# df -h | grep fat32
/dev/loop0      33M    512   33M    1% /mnt/fat32

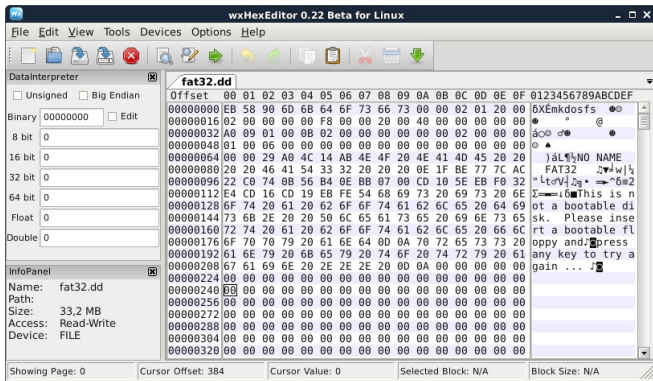
# ls -l /mnt/fat32
insgesamt 0

# echo "Betriebssysteme" > /mnt/fat32/liesmich.txt
# cat /mnt/fat32/liesmich.txt
Betriebssysteme
# ls -l /mnt/fat32/liesmich.txt
-rwxr-xr-x 1 root root 16 Feb 28 10:45 /mnt/fat32/liesmich.txt

# umount /mnt/fat32/
# mount | grep fat32
# df -h | grep fat32

# wxHexEditor fat32.dd
```

# FAT-Dateisysteme analysieren (2/3)



Hilfreiche Informationen:

<http://dorumugs.blogspot.de/2013/01/file-system-geography-fat32.html>

<http://www.win.tue.nl/~aeb/linux/fs/fat/fat-1.html>



# FAT-Dateisysteme analysieren (3/3)

wxHexEditor 0.22 Beta for Linux

File Edit View Tools Devices Options Help

DataInterpreter

☐ Unsigned ☐ Big Endian

Binary 00000000 ☐ Edit

8 bit 0

16 bit 0

32 bit 0

64 bit 0

Float 0

Double 0

InfoPanel

Name: fat32.dd

Path:

Size: 33,2 MB

Access: Read-Write

Device: FILE

| Offset   | 00 | 01 | 02 | 03 | 04 | 05 | 06 | 07 | 08 | 09 | 0A | 0B | 0C | 0D | 0E | 0F | 0123456789ABCDEF              |
|----------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|-------------------------------|
| 00551920 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 |                               |
| 00551936 | 41 | 6C | 00 | 69 | 00 | 65 | 00 | 73 | 00 | 6D | 00 | 0F | 00 | 61 | 69 | 00 | A l i e s m * a i             |
| 00551952 | 63 | 00 | 68 | 00 | 2E | 00 | 74 | 00 | 78 | 00 | 00 | 00 | 74 | 00 | 00 | 00 | c h , t x t                   |
| 00551968 | 4C | 49 | 45 | 53 | 4D | 49 | 43 | 48 | 54 | 58 | 54 | 20 | 00 | 64 | B4 | 55 | L I E S M I C H T X T d \ U   |
| 00551984 | 5C | 44 | 5C | 44 | 00 | 00 | B4 | 55 | 5C | 44 | 03 | 00 | 10 | 00 | 00 | 00 | \ D \ D { \ U \ D ▶           |
| 00552000 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 |                               |
| 00552016 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 |                               |
| 00552032 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 |                               |
| 00552048 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 |                               |
| 00552064 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 |                               |
| 00552080 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 |                               |
| 00552096 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 |                               |
| 00552112 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 |                               |
| 00552128 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 |                               |
| 00552144 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 |                               |
| 00552160 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 |                               |
| 00552176 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 |                               |
| 00552192 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 |                               |
| 00552208 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 |                               |
| 00552224 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 |                               |
| 00552240 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 |                               |
| 00552256 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 |                               |
| 00552272 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 |                               |
| 00552288 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 |                               |
| 00552304 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 |                               |
| 00552320 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 |                               |
| 00552336 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 |                               |
| 00552352 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 |                               |
| 00552368 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 |                               |
| 00552384 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 |                               |
| 00552400 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 |                               |
| 00552416 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 |                               |
| 00552432 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 |                               |
| 00552448 | 42 | 65 | 74 | 72 | 69 | 65 | 62 | 73 | 73 | 79 | 73 | 74 | 65 | 6D | 65 | 0A | B e t r i e b s s y s t e m e |
| 00552464 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 |                               |

Showing Page: 985    Cursor Offset: 551920    Cursor Value: 0    Selected Block: N/A    Block Size: N/A

# Problematik von Schreibzugriffen

- Sollen Dateien oder Verzeichnisse erstellt, verschoben, umbenannt, gelöscht oder einfach verändert werden, sind Schreibzugriffe im Dateisystem nötig
  - Schreiboperationen sollen Daten von einem konsistenten Zustand in einen neuen konsistenten Zustand überführen
- Kommt es während eines Schreibzugriffs zum Ausfall, muss die Konsistenz des Dateisystems überprüft werden
  - Ist ein Dateisystem mehrere GB groß, kann die Konsistenzprüfung mehrere Stunden oder Tage dauern
  - Die Konsistenzprüfung zu überspringen, kann zum Datenverlust führen
- Ziel: Die bei der Konsistenzprüfung zu überprüfenden Daten eingrenzen
- Lösung: Über Schreibzugriffe Buch führen  $\implies$  Journaling-Dateisysteme

# Journaling-Dateisysteme

- Diese Dateisysteme führen ein Journal, in dem die Schreibzugriffe gesammelt werden, bevor sie durchgeführt werden
  - In festen Zeitabständen werden das Journal geschlossen und die Schreiboperationen durchgeführt
- Vorteil: Nach einem Absturz müssen nur diejenigen Dateien (Cluster) und Metadaten überprüft werden, die im Journal stehen
- Nachteil: Journaling erhöht die Anzahl der Schreiboperation, weil Änderungen erst ins Journal geschrieben und danach durchgeführt werden
- 2 Varianten des Journaling:
  - Metadaten-Journaling
  - Vollständiges Journaling

Gute Beschreibungen der unterschiedlichen Journaling-Konzepte...

- **Analysis and Evolution of Journaling File Systems**, Vijayan Prabhakaran, Andrea C. Arpaci-Dusseau, Remzi H. Arpaci-Dusseau, 2005 USENIX Annual Technical Conference,  
[https://www.usenix.org/legacy/events/usenix05/tech/general/full\\_papers/prabhakaran/prabhakaran.pdf](https://www.usenix.org/legacy/events/usenix05/tech/general/full_papers/prabhakaran/prabhakaran.pdf)
- <http://www.ibm.com/developerworks/library/l-journaling-filesystems/index.html>

# Metadaten-Journaling und vollständiges Journaling

## • Metadaten-Journaling (*Write-Back*)

- Das Journal enthält nur Änderungen an den Metadaten (Inodes)
  - Nur die Konsistenz der Metadaten ist nach einem Absturz garantiert
- Änderungen an Clustern führt erst das `sync()` durch ( $\implies$  Write-Back)
  - Der Systemaufruf `sync()` überträgt die Änderungen im Page Cache, auch = Buffer Cache genannt (siehe Folie 38), auf die HDD/SDD
- Vorteil: Konsistenzprüfungen dauern nur wenige Sekunden
- Nachteil: Datenverlust durch einen Systemabsturz ist weiterhin möglich
- Optional bei ext3/4 und ReiserFS
- NTFS und XFS bieten ausschließlich Metadaten-Journaling

## • Vollständiges Journaling

- Änderungen an den Metadaten und alle Änderungen an Clustern der Dateien werden ins Journal aufgenommen
- Vorteil: Auch die Konsistenz der Dateien ist garantiert
- Nachteil: Alle Schreiboperation müssen doppelt ausgeführt werden
- Optional bei ext3/4 und ReiserFS

Die Alternative ist also hohe Datensicherheit und hohe Schreibgeschwindigkeit

# Kompromiss aus beiden Varianten: Ordered-Journaling

- Die meisten Linux-Distributionen verwenden standardmäßig einen Kompromiss aus beiden Varianten
- **Ordered-Journaling**
  - Das Journal enthält nur Änderungen an den Metadaten
  - **Dateiänderungen werden erst im Dateisystem durchgeführt und danach die Änderungen an den betreffenden Metadaten ins Journal geschrieben**
  - Vorteil: Konsistenzprüfungen dauern nur wenige Sekunden und ähnliche hohe Schreibgeschwindigkeit wie beim Metadaten-Journaling
  - Nachteil: Nur die Konsistenz der Metadaten ist garantiert
    - Beim Absturz mit nicht abgeschlossenen Transaktionen im Journal sind neue Dateien und Dateianhänge verloren, da die Cluster noch nicht den Inodes zugeordnet sind
    - Überschriebene Dateien haben nach einem Absturz möglicherweise inkonsistenten Inhalt und können nicht mehr repariert werden, da die alte Version nicht gesichert wurde
  - Beispiele: Einzige Alternative bei JFS, Standard bei ext3/4 und ReiserFS

Interessant: <https://www.heise.de/newsticker/meldung/Kernel-Entwickler-streiten-ueber-Ext3-und-Ext4-209350.html>

# Problem des Overheads für Verwaltungsinformationen

- Jeder Inode bei Blockadressierung adressiert eine bestimmte Anzahl Clusternummern direkt
- Benötigt eine Datei mehr Cluster, wird indirekt adressiert

Inode bei ext3/4 (Blockadressierung)

|                                       |
|---------------------------------------|
| Clusteradresse 1 (direkt)             |
| Clusteradresse 2 (direkt)             |
| Clusteradresse 3 (direkt)             |
| Clusteradresse 4 (direkt)             |
| Clusteradresse 5 (direkt)             |
| Clusteradresse 6 (direkt)             |
| Clusteradresse 7 (direkt)             |
| Clusteradresse 8 (direkt)             |
| Clusteradresse 9 (direkt)             |
| Clusteradresse 10 (direkt)            |
| Clusteradresse 11 (direkt)            |
| Clusteradresse 12 (direkt)            |
| Clusteradresse 13 (indirekt)          |
| Clusteradresse 14 (doppelt indirekt)  |
| Clusteradresse 15 (dreifach indirekt) |

32 Bits (4 Bytes)

Clusternummern (Daten der Datei)

|                  |
|------------------|
| 1                |
| 2                |
| 3                |
| 4                |
| 5                |
| 6                |
| 7                |
| 8                |
| 9                |
| 10               |
| 11               |
| 12               |
| 13 - 268         |
| 269 - 65804      |
| 65805 - 16843020 |

maximal  
48 kB direkt  
adressierbar  
bei 4 kB  
Clustergröße

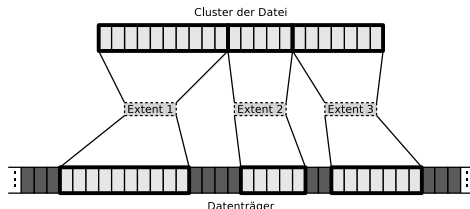
12 Cluster

256 Cluster  
65536 Cluster  
16777216 Cluster

- Dieses Adressierungsschema führt bei steigender Dateigröße zu zunehmendem Overhead für Verwaltungsinformationen
- Lösung: Extents

# Extent-basierte Adressierung

- Inodes adressieren nicht einzelne Cluster, sondern bilden möglichst große Dateibereiche auf Bereiche zusammenhängender Blöcke (**Extents**) auf dem Speichergerät ab
- Statt vieler einzelner Clusternummern sind nur 3 Werte nötig:
  - Start (Clusternummer) des Bereichs (Extents) in der Datei
  - Größe des Bereichs in der Datei (in Clustern)
  - Nummer des ersten Clusters auf dem Speichergerät
- Ergebnis: Weniger Verwaltungsaufwand
- Beispiele: JFS, XFS, btrfs, NTFS, ext4



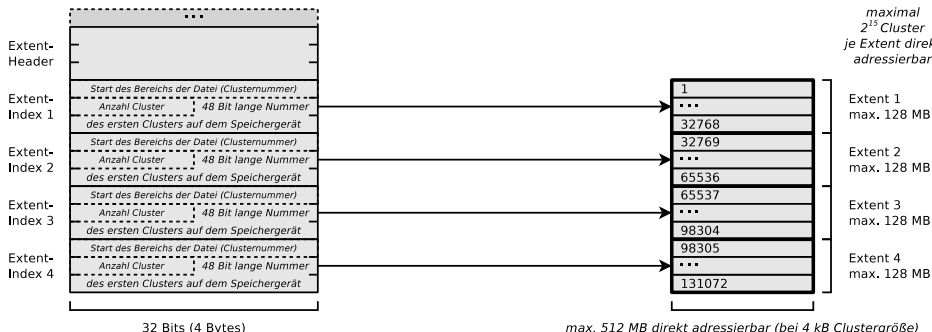
# Extents am Beispiel von ext4

- Bei Blockadressierung mit ext3/4 sind in jedem Inode 15 je 4 Bytes große Felder, also 60 Bytes, zur Adressierung von Clustern verfügbar
- ext4 verwendet diese 60 Bytes für einen Extent-Header (12 Bytes) und zur Adressierung von 4 Extents (jeweils 12 Bytes)

Inode bei ext4 (Extent-basierte Adressierung)

Clusternummern (Daten der Datei)

maximal  
 $2^{15}$  Cluster  
je Extent direkt  
adressierbar



32 Bits (4 Bytes)

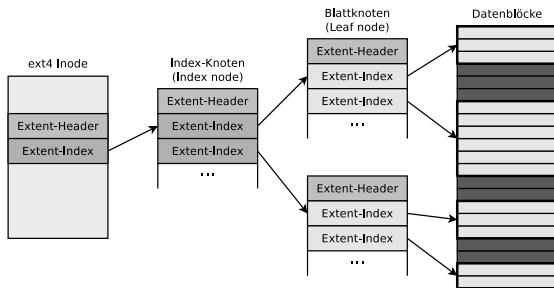
max. 512 MB direkt adressierbar (bei 4 kB Clustergröße)

Maximale Partitionsgröße bei ext4:  $2^{48}$  Clusternummern  $\times$  4096 Byte Clustergröße = 1 Exabyte



# Vorteil von Extents am Beispiel von ext4

- Mit maximal 12 Clustern kann ein ext3/4-Inode 48 kB (bei 4 kB Clustergröße) direkt adressieren
- Mit 4 Extents kann ein ext4-Inode 512 MB direkt adressieren
- Ist eine Datei > 512 MB, baut ext4 einen Baum aus Extents auf
  - Das Prinzip ist analog zur indirekten Blockadressierung



# NTFS – New Technology File System

## Verschiedene Versionen des NTFS-Dateisystems existieren

- NTFS 1.0: Windows NT 3.1
- NTFS 1.1: Windows NT 3.5/3.51
- NTFS 2.x: Windows NT 4.0 bis SP3
- NTFS 3.0: Windows NT 4.0 ab SP3/2000
- NTFS 3.1: Windows XP/2003/Vista/7/8/10

## Aktuelle Versionen von NTFS bieten zusätzlich...

- Unterstützung für Kontingente (Quota) ab Version 3.x
- transparente Kompression
- transparente Verschlüsselung (Triple-DES und AES) ab Version 2.x

- Clustergröße: 512 Bytes bis 64 kB
- NTFS bietet im Vergleich zu seinem Vorgänger FAT u.a.:
  - Maximale Dateigröße: 16 TB ( $\implies$  Extents)
  - Maximale Partitionsgröße: 256 TB ( $\implies$  Extents)
  - Sicherheitsfunktionen auf Datei- und Verzeichnisebene
- Genau wie VFAT...
  - speichert NTFS Dateinamen bis zu einer Länge von 255 Unicode-Zeichen
  - realisiert NTFS eine Kompatibilität zur Betriebssystemfamilie MS-DOS, indem es für jede Datei einen eindeutigen Dateinamen im Format 8.3 speichert

# Struktur von NTFS (1/2)

- Das Dateisystem enthält eine Hauptdatei – **Master File Table (MFT)**
  - Enthält die Referenzen, welche Cluster zu welcher Datei gehören
  - Enthält auch die Metadaten der Dateien (Dateigröße, Dateityp, Datum der Erstellung, Datum der letzten Änderung und evtl. den Dateiinhalt)
    - Der Inhalt kleiner Dateien  $\leq 900$  Bytes wird direkt in der MFT gespeichert

Quelle: **How NTFS Works**. Microsoft. 2003. [https://technet.microsoft.com/en-us/library/cc781134\(v=ws.10\).aspx](https://technet.microsoft.com/en-us/library/cc781134(v=ws.10).aspx)

- Beim Formatieren einer Partition wird für die MFT ein fester Bereich reserviert
  - Standardmäßig werden für die MFT 12,5% der Partitionsgröße reserviert
  - Ist der Bereich voll, verwendet das Dateisystem freien Speicher der Partition für die MFT
    - Dabei kann es zu einer Fragmentierung der MFT kommen

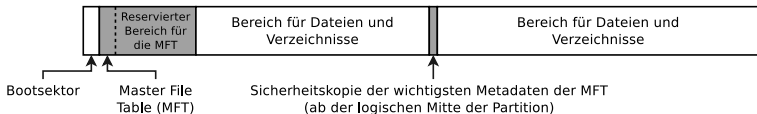
| Partitionsgröße | Clustergröße |
|-----------------|--------------|
| < 16 TB         | 4 kB         |
| 16 TB - 32 TB   | 8 kB         |
| 32 TB - 64 TB   | 16 kB        |
| 64 TB - 128 TB  | 32 kB        |
| 128 TB - 256 TB | 64 kB        |

Die Tabelle enthält die Standard-Clustergrößen unter Windows 2000/XP/Vista/7. Die Clustergröße kann beim Erzeugen des Dateisystems festgelegt werden. Die Standardgröße ist 4 kB.

Quelle: <http://support.microsoft.com/kb/140365/de>

# Struktur von NTFS (2/2)

## Struktur des Dateisystems



## MFT-Eintrag einer Datei ( $\leq 900$ Bytes)

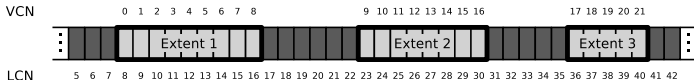
| Datei-Attribute | Datei-Name | Zugriffsrechte<br>(Security Descriptor) | Dateiinhalte |
|-----------------|------------|---|--------------|
|-----------------|------------|---|--------------|

## MFT-Eintrag einer Datei mit Extents

| Datei-Attribute | Datei-Name | Zugriffsrechte<br>(Security Descriptor) | Verweise auf Extents ("Data Runs") |     |                |     |
|-----------------|------------|---|------------------------------------|-----|----------------|-----|
|                 |            |   | Extent                             | VCN | Anzahl Cluster | LCN |
|                 |            |   | 1                                  | 0   | 9              | 8   |
|                 |            |   | 2                                  | 9   | 8              | 23  |
|                 |            |   | 3                                  | 17  | 5              | 36  |

(Länges jedes MFT-Eintrags: 1 kB)

## Datenträger



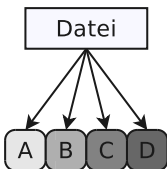
Verweist ein MFT-Eintrag auf Extents (sog. „Data Runs“), speichert er pro Extent 3 Werte

- Start (Clusternummer) des Bereichs (Extents) in der Datei  $\Rightarrow$  **Virtual Cluster Number (VCN)**
- Größe des Bereichs in der Datei (in Clustern)  $\Rightarrow$  **Anzahl Cluster**
- Nummer des ersten Clusters auf dem Speichergerät  $\Rightarrow$  **Logical Cluster Number (LCN)**

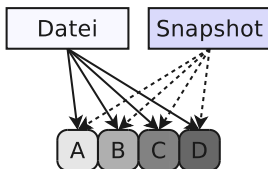
Auch ein Verzeichnis ist eine Datei (MFT-Eintrag), deren Dateiinhalte die Nummern der MFT-Einträge (Dateien) sind, die dem Verzeichnis zugeordnet sind.

# Modernstes Konzept: Copy-On-Write

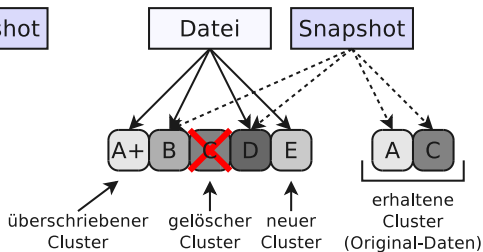
Vor dem Snapshot



Nach dem Snapshot



Nach Änderungen



- Schreibzugriffe im Dateisystem ändern/löschen keine Dateiinhalte
  - Veränderte Inhalte werden in freie Cluster geschrieben
  - Anschließend werden die Metadaten auf die neue Datei angepasst
- Ältere Dateiversionen bleiben erhalten und können wiederhergestellt werden
  - Die Datensicherheit ist besser als bei Dateisystemen mit Journal
  - Snapshots können sehr schnell erzeugt werden (nur Metadaten-Änderung)
- Beispiele: ZFS, btrfs und ReFS (Resilient File System)

# Datenzugriffe mit einem Cache beschleunigen (1/2)

- Moderne Betriebssysteme beschleunigen Datenzugriffe mit einem **Page Cache** (auch *Buffer Cache* genannt) im Hauptspeicher
  - Wird eine Datei lesend angefragt, schaut der Kernel zuerst im Page Cache nach, ob die Datei dort vorliegt
    - Liegt die Datei nicht im Page Cache vor, wird sie in diesen geladen
- Der Page Cache ist nie so groß, wie die Menge der Daten auf dem System
  - Darum müssen selten nachgefragte Daten verdrängt werden
    - Wurden Daten im Cache verändert, müssen die Änderungen spätestens beim Verdrängen nach unten durchgereicht (zurückgeschrieben) werden
    - Optimales Verwenden des Cache ist nicht möglich, da Datenzugriffe nicht deterministisch (nicht vorhersagbar) sind
- Die meisten Betriebssystemen geben Schreibzugriffe nicht direkt weiter (⇒ **Write-Back**)
  - Vorteil: Höhere System-Geschwindigkeit
  - Nachteil: Stürzt das System ab, kann es zu Inkonsistenzen kommen

## Datenzugriffe mit einem Cache beschleunigen (2/2)

- DOS und Windows bis Version 3.11 verwenden das Programm *Smartdrive* um einen Page Cache zu realisieren
  - Auch alle späteren Versionen von Windows enthalten einen *Cache Manager*, der einen Page Cache verwaltet
- Linux puffert automatisch so viele Daten wie Platz im Hauptspeicher ist
  - Das Kommando `free -m` gibt unter Linux eine Übersicht der Speicherbelegung aus
    - Es informiert auch in den Spalten `buffers` und `cached` darüber, wie viel Speicherkapazität des Hauptspeichers gegenwärtig für den Page Cache verwendet wird

```
$ free -m
```

|                    | total | used | free  | shared | buffers | cached |
|--------------------|-------|------|-------|--------|---------|--------|
| Mem:               | 7713  | 6922 | 790   | 361    | 133     | 1353   |
| -/+ buffers/cache: |       | 5436 | 2277  |        |         |        |
| Swap:              | 11548 | 247  | 11301 |        |         |        |

Gute Quellen zum Thema Page Cache unter Linux und wie man ihn manuell leeren kann

[http://www.thomas-krenn.com/de/wiki/Linux\\_Page\\_Cache](http://www.thomas-krenn.com/de/wiki/Linux_Page_Cache)

<http://unix.stackexchange.com/questions/87908/how-do-you-empty-the-buffers-and-cache-on-a-linux-system>

<http://serverfault.com/questions/85470/meaning-of-the-buffers-cache-line-in-the-output-of-free>