6. Foliensatz Betriebssysteme

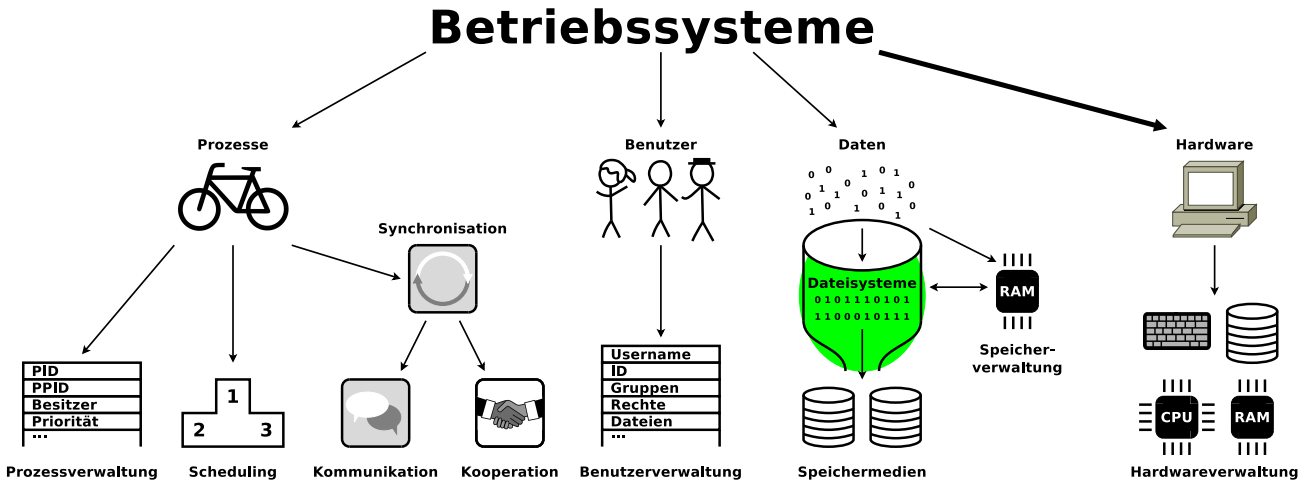
Prof. Dr. Christian Baun

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

Lernziele dieses Foliensatzes

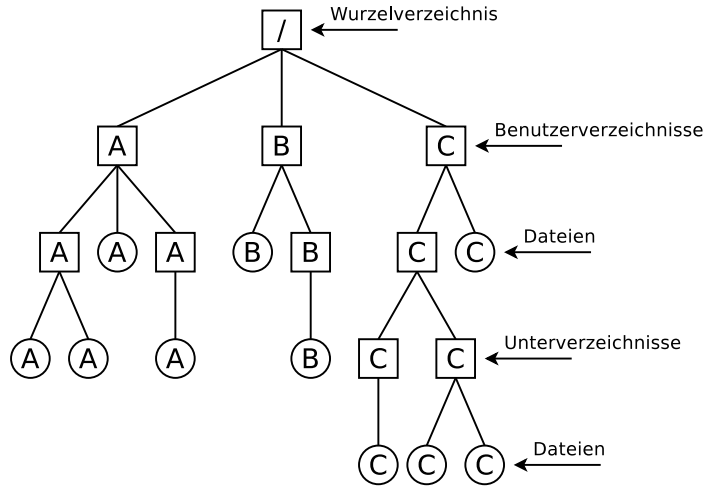
- Am Ende dieses Foliensatzes kennen/verstehen Sie...
  - die **Aufgaben und Grundbegriffe von Dateisystemen**
  - was **Inodes** und **Cluster** sind
  - wie **Blockadressierung** funktioniert
  - den **Aufbau** ausgewählter Dateisysteme
  - eine Übersicht über die **Windows-Dateisysteme** und deren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

Übungsblatt 6 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 Litaratur 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 $\Longrightarrow$ 1 kB geht verloren
- Dateigröße: 1 kB. Clustergröße: 64 kB $\Longrightarrow$ 63 kB gehen verloren!

- Die Clustergröße kann man beim Anlegen des Dateisystems festlegen

Grundbegriffe von Linux-Dateisystemen

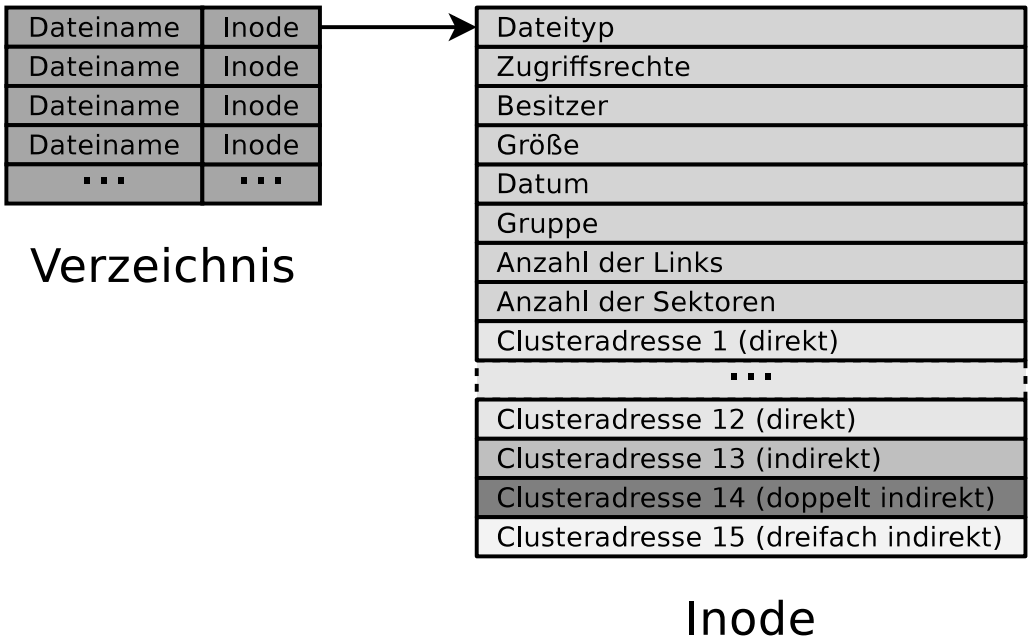
Unter Linux gilt: Clustergröße $\leq$ Größe der Speicherseiten (Pagesize)

- 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 (siehe Folie 12)
  - Inhalt: Dateiname und Inode-Nummer für jede Datei des Verzeichnisses
- Arbeitssweise 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

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



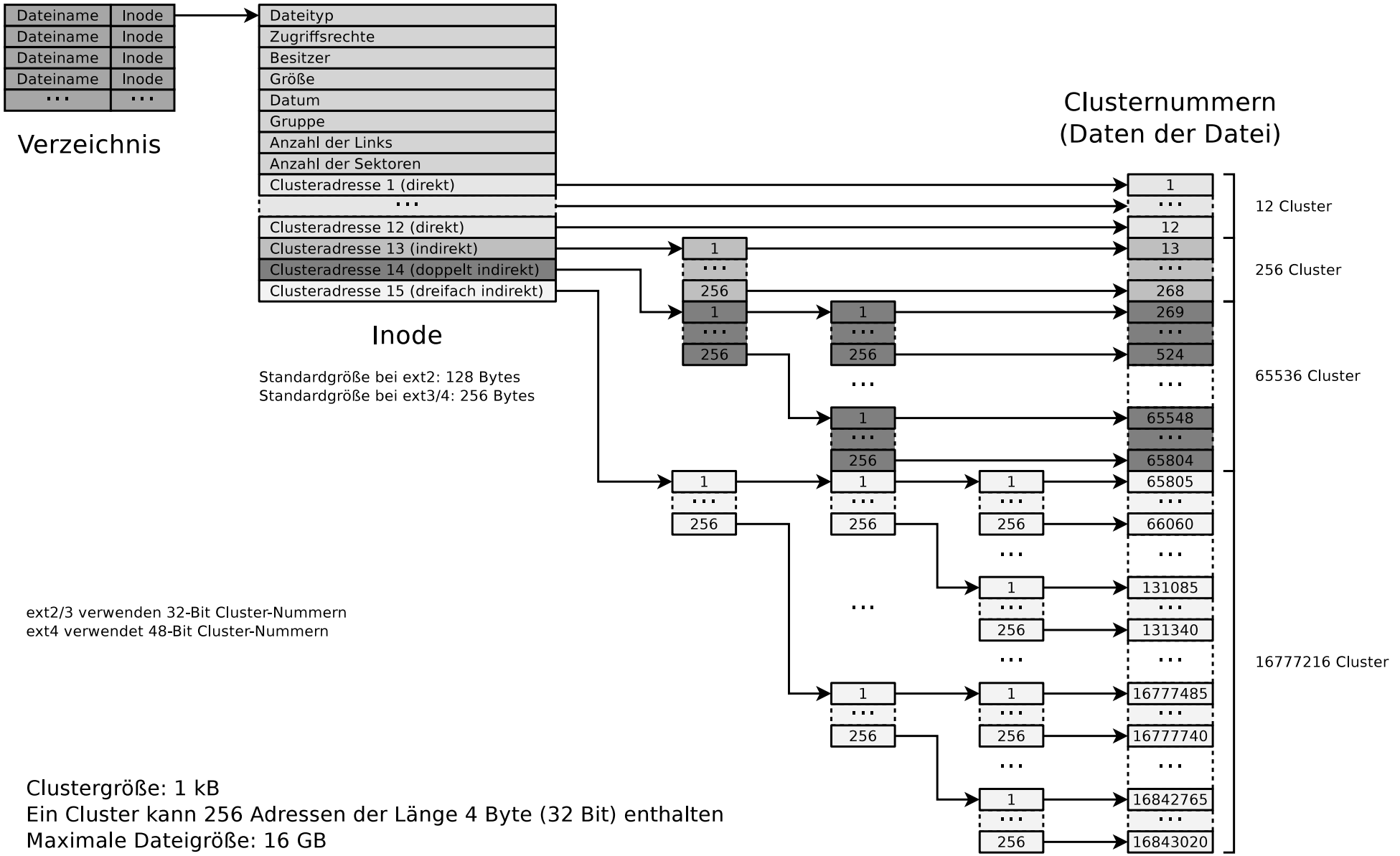
- Benötigt eine Datei mehr Cluster, wird indirekt adressiert mit Clustern, deren Inhalt Clusternummern sind
- Blockadressierung verwenden Minix, ext2/3, ReiserFS und Reiser4

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

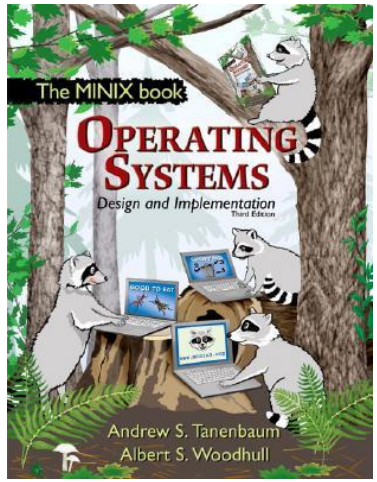
Direkte und indirekte Adressierung am Beispiel ext2/3



Minix

Das Betriebssystem Minix `http://www.minix3.org`

- Unix-ähnliches Betriebssystem
- Wird seit 1987 von Andrew S. Tanenbaum als Lehrsystem entwickelt
  `https://www.youtube.com/watch?v=bx3KuE7UjGA`
- Aktuelle Version: 3.3.0 aus dem Jahr 2014
- Auf Intel-Chipsätzen nach 2015 läuft MINIX 3 intern als Software-Komponente der Intel Management Engine
  `https://www.zdnet.com/article/minix-intels-hidden-in-chip-operating-system/`
  `https://linuxnews.de/2017/11/minix-in-der-intel-management-engine/`
  `https://itsfoss.com/fact-intel-minix-case/`



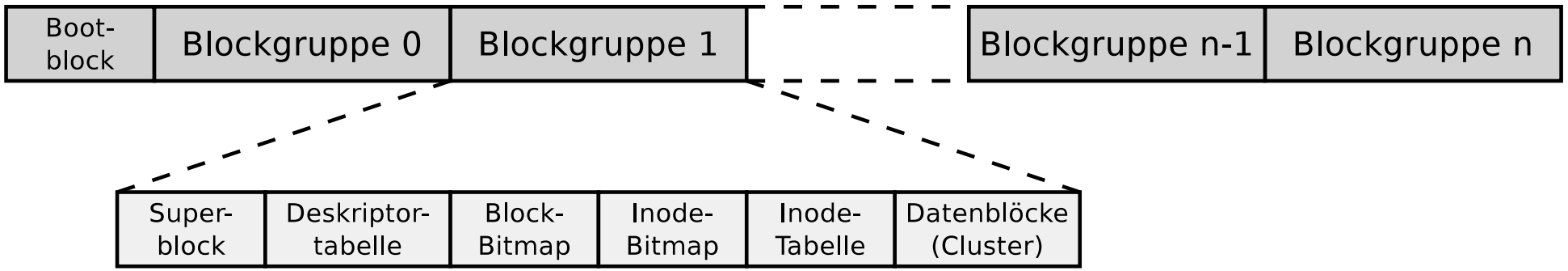
- Standard-Dateisystem unter Linux bis 1992
  - Naheliegend, denn Minix war die Grundlage der Entwicklung von Linux
- Verwaltungsaufwand des Minix-Dateisystems ist gering
  - Sinnvolle Einsatzbereiche „heute“: Boot-Disketten und RAM-Disks
- Speicher wird als lineare Kette gleichgroßer Cluster (1-8 kB) dargestellt
- 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 (1 Cluster)	Superblock (1 Cluster)	Inodes-Bitmap (1 Cluster)	Cluster-Bitmap (1 Cluster)	Inodes (15 Cluster)	Daten (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 wird von mindestens 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

Bereiche in einem ext2/3/4-Dateisystem



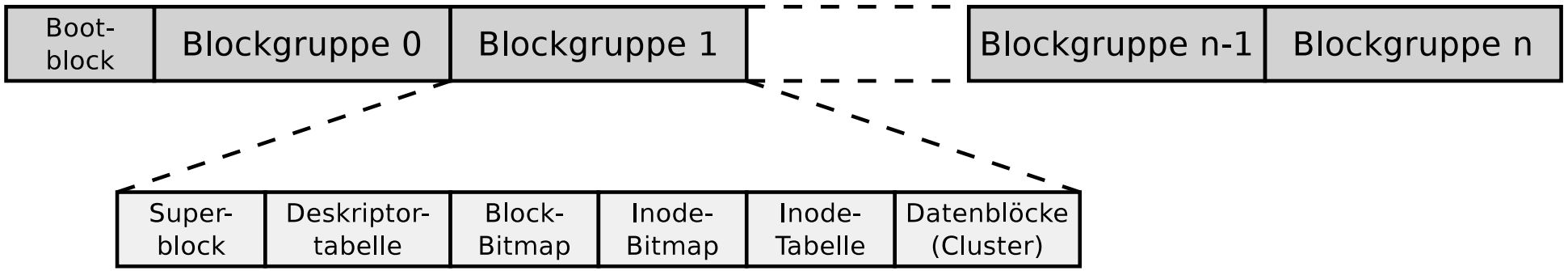
- Die Cluster des Dateisystems sind in **Blockgruppen** gleicher Größe zusammengefasst
  - Die Informationen über die Metadaten und freien Cluster jeder Blockgruppe werden in der jeweiligen Blockgruppe verwaltet

Maximale Größe einer Blockgruppe: 8x Clustergröße in Bytes

Beispiel: Ist die Clustergröße 4096 Bytes, kann jede Blockgruppe maximal 32768 Cluster umfassen.
$\Longrightarrow$ Die maximale Blockgröße ist somit 32768 Cluster * 4096 Bytes Clustergröße = 134.217.728 Bytes = 131.072 kB = 128 MB

- 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 ext2/3/4



- 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 analysieren (1/3)

Wiederholung

Verzeichnisse sind Dateien mit Dateinamen und Inode-Nummern

Da kann man doch mal reinschauen...

Mit der Spezifikation des Dateisystems und geeigneten Werkzeugen können u.a. die einzelnen Felder der Verzeichniseinträge untersucht werden. Zum Beispiel der Eintrag von `liesmich.txt`

```
# lsblk | grep sda1
sda             8:0    1  29,3G  0 disk
  sda1          8:1    1  29,3G  0 part

# mkfs.ext4 /dev/sda1

# mkdir /mnt/test
# mount -t ext4 /dev/sda1 /mnt/test/

# df -h | grep test
/dev/sda1         29G     45M   28G    1% /mnt/test

# ls -l /mnt/test
insgesamt 16
drwx------ 2 root root 16384 Sep 14 09:38 lost+found

# mkdir /mnt/test/testfolder
# echo "Betriebssysteme" > /mnt/test/testfolder/liesmich.txt
# echo "OpSys" > /mnt/test/testfolder/file2.txt
# echo "12345" > /mnt/test/testfolder/anotherfile.dat
# touch /mnt/test/testfolder/empty_file

# ls -lai /mnt/test/testfolder/
insgesamt 20
392449 drwxr-xr-x 2 root root 4096 Sep 14 09:59 .
     2 drwxr-xr-x 4 root root 4096 Sep 14 09:46 ..
392452 -rw-r--r-- 1 root root    6 Sep 14 09:58 anotherfile.dat
392453 -rw-r--r-- 1 root root    0 Sep 14 09:59 empty_file
392451 -rw-r--r-- 1 root root    6 Sep 14 09:47 file2.txt
392450 -rw-r--r-- 1 root root   16 Sep 14 09:47 liesmich.txt
```

Dateisysteme analysieren (2/3)

Wir analysieren den Inhalt von `testfolder.out` mit Kommandozeilenwerkzeugen wie `hexdump` oder `od` (octal dump). Alternativ ginge auch ein grafischer grafisches Werkzeug wie z.B. `wxHexEditor` (siehe Folie 27).

```
# debugfs /dev/sda1
debugfs 1.44.5 (15-Dec-2018)
debugfs: imap <392449>
Inode 392449 is part of block group 48
  located at block 1572896, offset 0x0000
debugfs: dump testfolder testfolder.out
debugfs: quit

# ls -l testfolder.out
-rw-r--r-- 1 root root 4096 Sep 14 10:00 testfolder.out

# hexdump -C testfolder.out
00000000 01 fd 05 00 0c 00 01 02 2e 00 00 00 02 00 00 00 |................|
00000010 0c 00 02 02 2e 2e 00 00 02 fd 05 00 14 00 0c 01 |................|
00000020 6c 69 65 73 6d 69 63 68 2e 74 78 74 03 fd 05 00 |liesmich.txt....|
00000030 14 00 09 01 66 69 6c 65 32 2e 74 78 74 00 00 00 |....file2.txt...|
00000040 04 fd 05 00 18 00 0f 01 61 6e 6f 74 68 65 72 66 |........anotherf|
00000050 69 6c 65 2e 64 61 74 00 05 fd 05 00 9c 0f 0a 01 |ile.dat.........|
00000060 65 6d 70 74 79 5f 66 69 6c 65 00 00 00 00 00 00 |empty_file......|
00000070 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 |................|
...
```

Ein Hex-Editor stellt die Daten auf verschiedene Arten dar

- 1.Spalte: Anzahl der vorausgegangenen Bytes $\Longrightarrow$ Offset bzw. Positionsangabe (Adresse) in der Datei in Hex-Schreibweise (Wertebereich 0-F)
- 2.Spalte: Bytes der Zeile in hexadezimaler Darstellung
- 3.Spalte: Bytes der Zeile in ASCII-Darstellung

Einige Grundlagen...

- Hexadezimalsystem $\Longrightarrow$ Basis 16
- 1 Hex-Zeichen stellt 4 Bits dar
- 2 Hex-Zeichen stellen 1 Byte dar

Dateisysteme analysieren (3/3)

https://ext4.wiki.kernel.org/index.php/Ext4_Disk_Layout

Offset	Größe	Name	Bedeutung
0x0	4 Bytes	inode	Nummer des Inodes auf den der Verzeichniseintrag verweist
0x4	2 Bytes	record length	Länge dieses Verzeichniseintrags
0x6	1 Byte	name length	Länge des Dateinamens
0x7	1 Byte	file type	0x0 = Unbekannt, 0x1 = Reguläre Datei, 0x2 = Verzeichnis, 0x3 = Zeichenorientierte Gerätedatei, 0x4 = Blockorientierte Gerätedatei, 0x5 = FIFO (Benannte Pipe), 0x6 = Socket, 0x7 = symbolischer Link
0x8	Zeichenkette	name	Dateiname

Hinweis zur Byte-Reihenfolge Big-Endian vs. Little-Endian

x86-Prozessoren verwenden die **Little-Endian Byte-Reihenfolge**. Besteht ein Datenfeld aus mehreren Bytes, liegt das Least Significant Byte (LSB) an erster Stelle, also auf der niedrigsten Speicheradresse. Die höherwertigen Bytes liegen auf den nachfolgenden Speicheradressen

```
00000010 -- -- -- -- -- -- -- -- 02 fd 05 00 14 00 0c 01 |................|
00000020 6c 69 65 73 6d 69 63 68 2e 74 78 74 -- -- -- -- |liesmich.txt....|
```

Inode-Nummer: 02 fd 05 00 $\Longrightarrow$ 00 05 fd 02

```
$ printf "%d\n" 0x0005fd02
392450
```

Länge des Verzeichniseintrags: 14 00 $\Longrightarrow$ 00 14 $\Longrightarrow$ 20 Bytes

```
$ printf "%d\n" 0x0014
20
```

Länge des Dateinamens: 0c $\Longrightarrow$ 12 Bytes
Dateityp: 01 $\Longrightarrow$ Reguläre Datei

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**)
  $\Longrightarrow$ siehe Folien 18 und 20

Bereiche in einem FAT-Dateisystem (1/2)

Bereich 1	Bereich 2	Bereich 3	Bereich 4	Bereich 5	Bereich 6
Bootsektor	Reservierte Sektoren	FAT1	FAT2	Stamm-verzeichnis	Daten

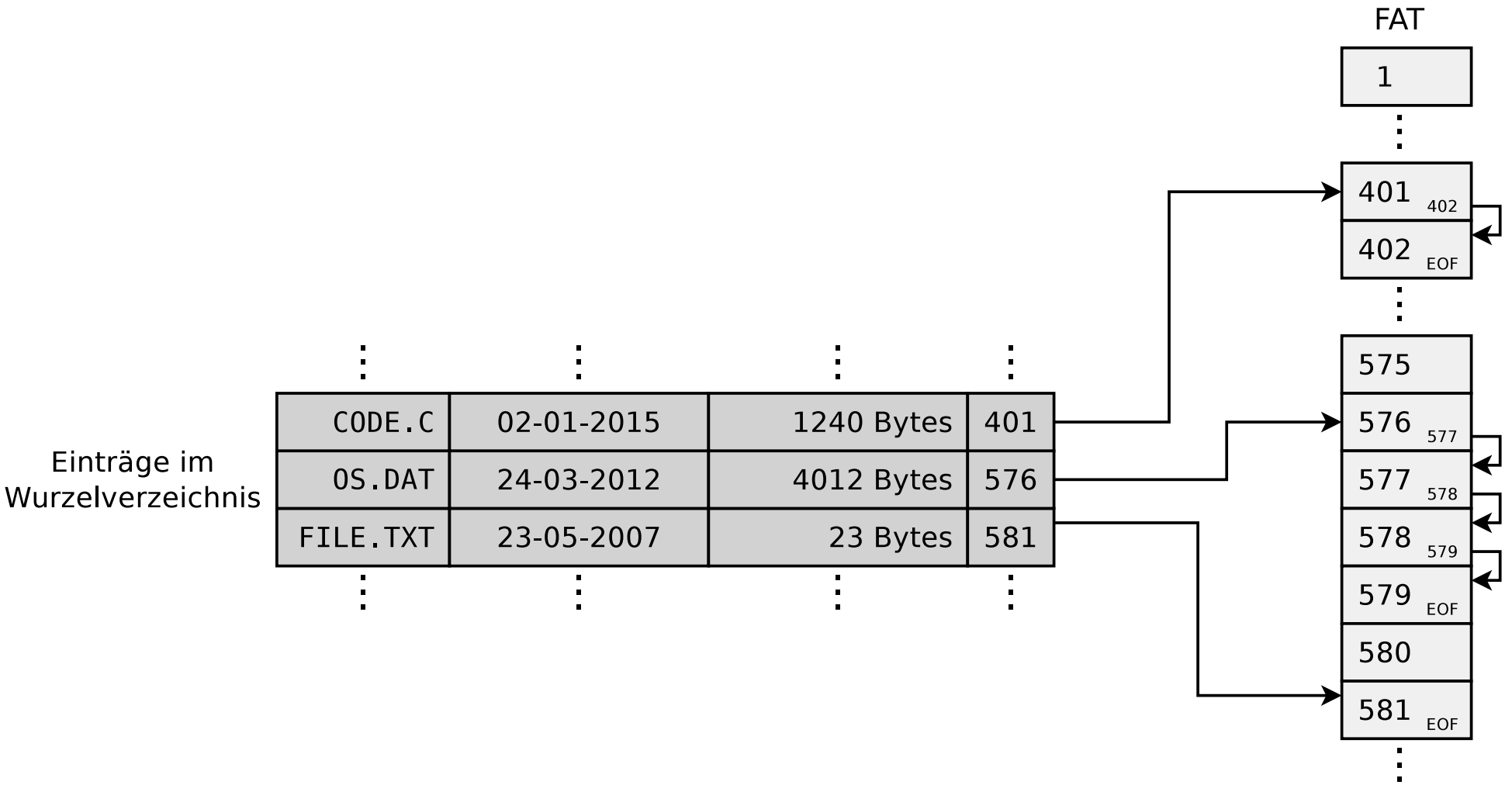
- Im **Bootsektor** liegen ausführbarer x86-Maschinencode, der das Betriebssystem starten soll, und Informationen über das Dateisystem:
  - Blockgröße des Speichermediums (512, 1024, 2048 oder 4096 Bytes)
  - Anzahl der Blöcke pro Cluster
  - Anzahl der Blöcke (Sektoren) auf dem Speichermedium
  - Beschreibung 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	Stamm-verzeichnis	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**

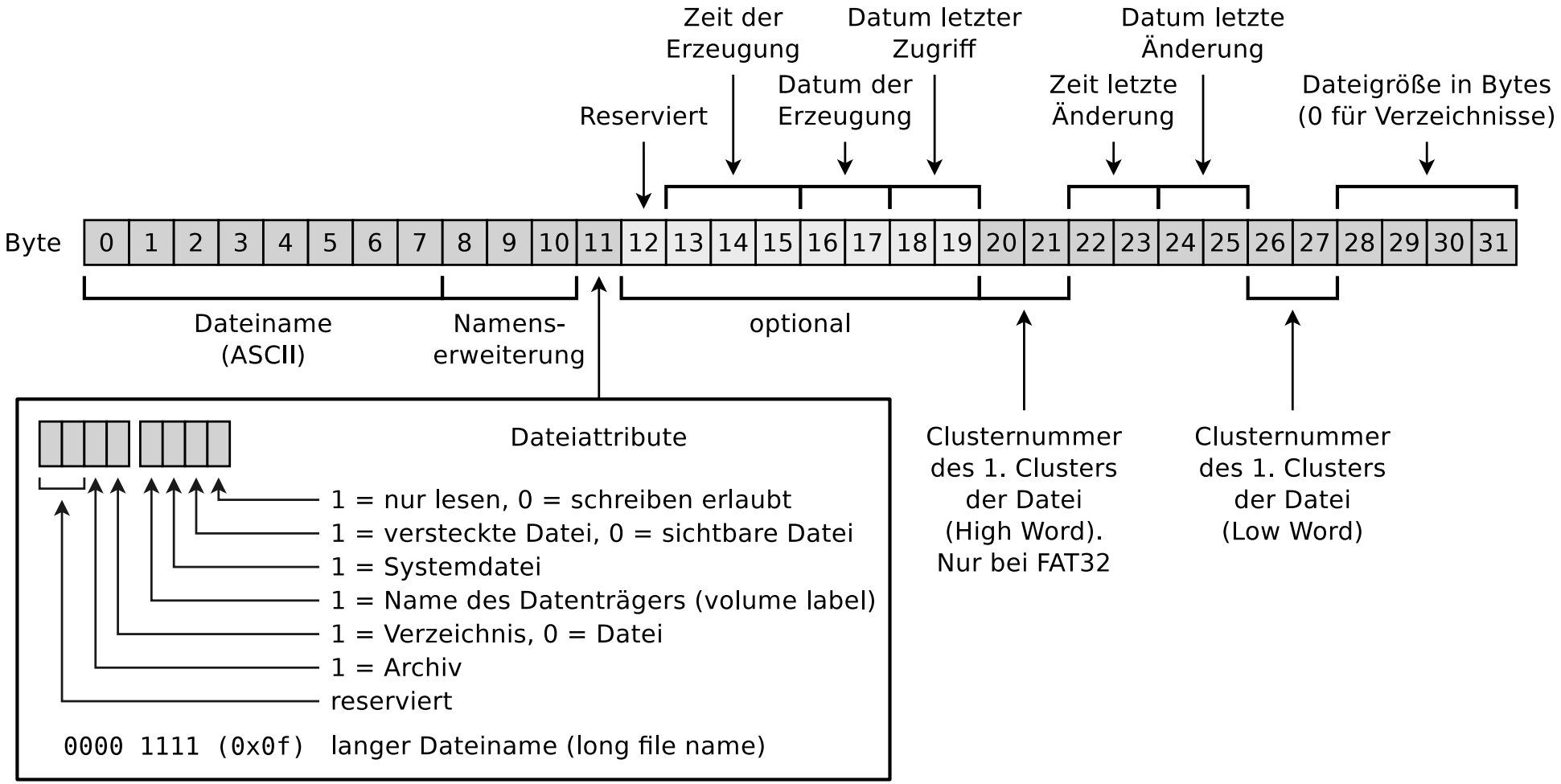
Stammverzeichnis (Wurzelverzeichnis) und FAT



Das Thema FAT ist anschaulich erklärt bei...

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

Struktur der Einträge im Stammverzeichnis

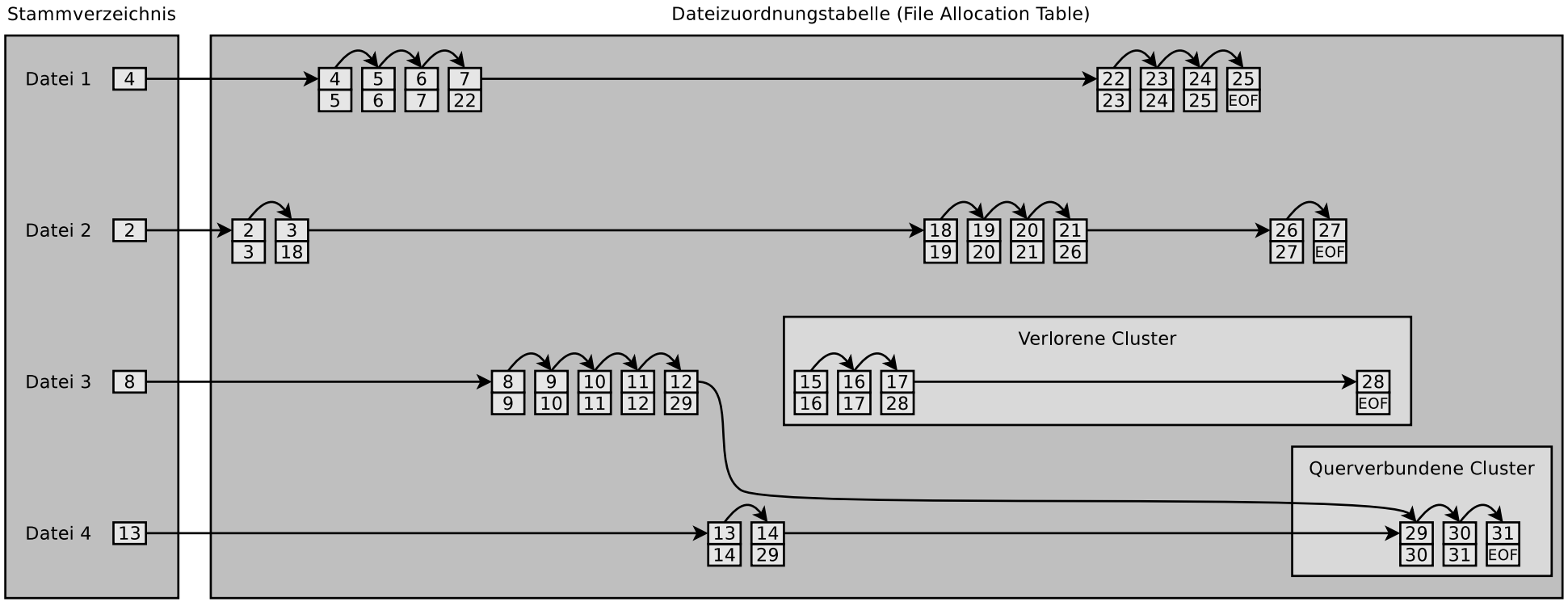


Warum ist 4 GB die maximale Dateigröße unter FAT32?

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



Quelle: `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 $2^{12} = 4096$ Cluster können adressiert werden
- Clustergröße: 512 Bytes bis 4 kB
- Unterstützt nur Speichermedien (Partitionen) bis 16 MB

$2^{12} * 4\,\text{kB Clustergröße} = 16384\,\text{kB} = 16\,\text{MB maximale Dateisystemgröße}$

- 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 $2^{16} = 65536$ 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 ≤ 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/7/8/10. Die Clustergröße kann beim Erzeugen des Dateisystems festgelegt werden

Einige Betriebssysteme (z.B. MS-DOS und Windows 95/98/Me) unterstützen keine Cluster $>$ 32 kB

Einige Betriebssysteme (z.B. Windows 2000/XP/7/8/10) unterstützen keine Cluster $>$ 64 kB

Quellen:
`http://support.microsoft.com/kb/140365/de`
`http://secrets.mysfyts.com/index.asp?Page=Fat`
`http://web.allensmith.net/Storage/HDDlimit/FAT16.htm`

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/7/8/10. Die Clustergröße kann beim Erzeugen des Dateisystems festgelegt werden

Quelle: `http://support.microsoft.com/kb/140365/de`

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 Funktion unter Beibehaltung der Abwärtskompatibilität
- Lange Dateinamen (max. 255 Zeichen) werden auf max. 20 Pseudo-Verzeichniseinträge verteilt (siehe Folie 25)
- 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 18)
- Besonderheit: Groß/Kleinschreibung wird angezeigt, aber ignoriert

Kompatibilität zu MS-DOS und Windows bis v3.11

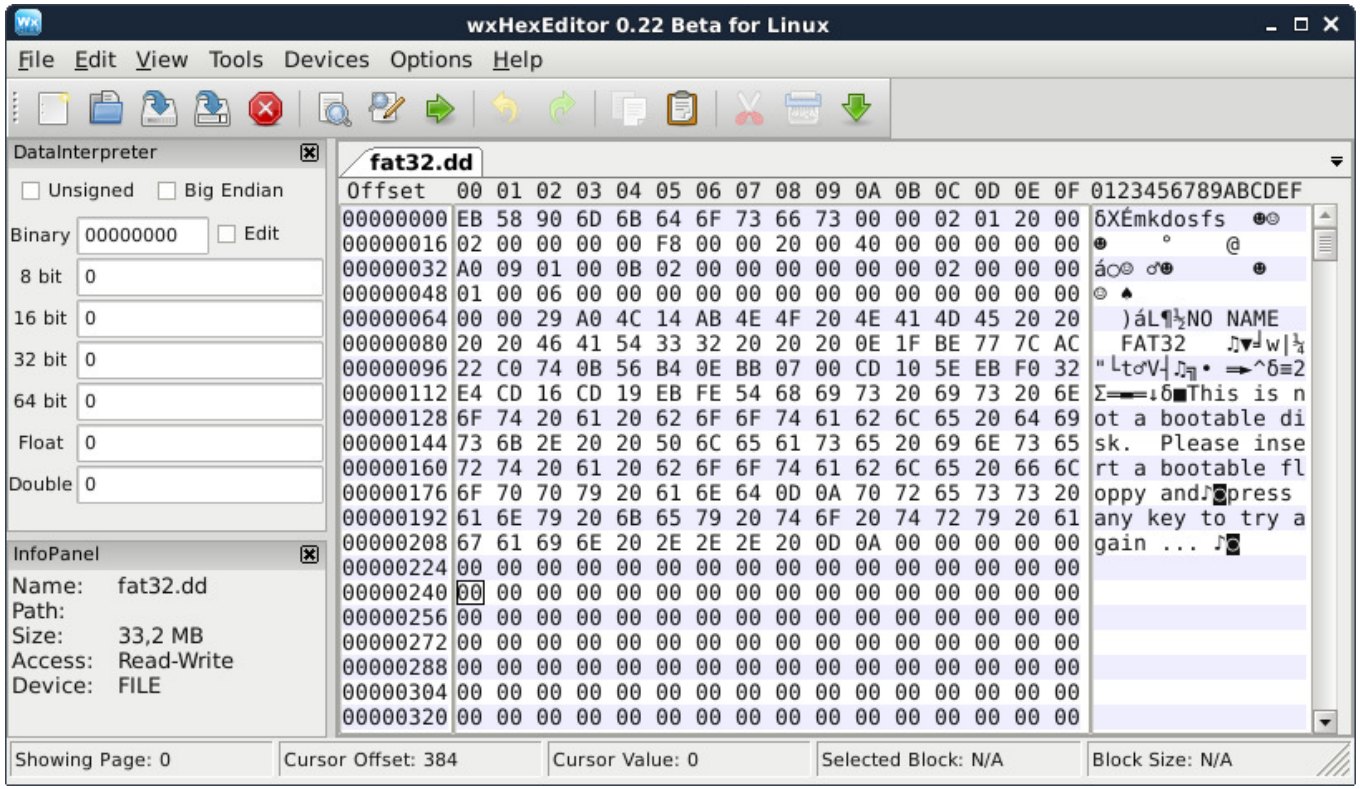
- VFAT und NTFS (siehe Folie 38) speichern für jede Datei einen eindeutigen Dateinamen im Format 8.3
  - Betriebssysteme ohne die VFAT-Erweiterung ignorieren die Pseudo-Verzeichniseinträge und zeigen nur den verkürzten Dateinamen
    - So können Microsoft-Betriebssysteme ohne VFAT-Unterstützung auf Dateien 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:

  `Ein ganz langer Dateiname.test.pdf` $\Longrightarrow$ `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,iocharset=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)



Ein Hex-Editor stellt die Daten auf verschiedene Arten dar

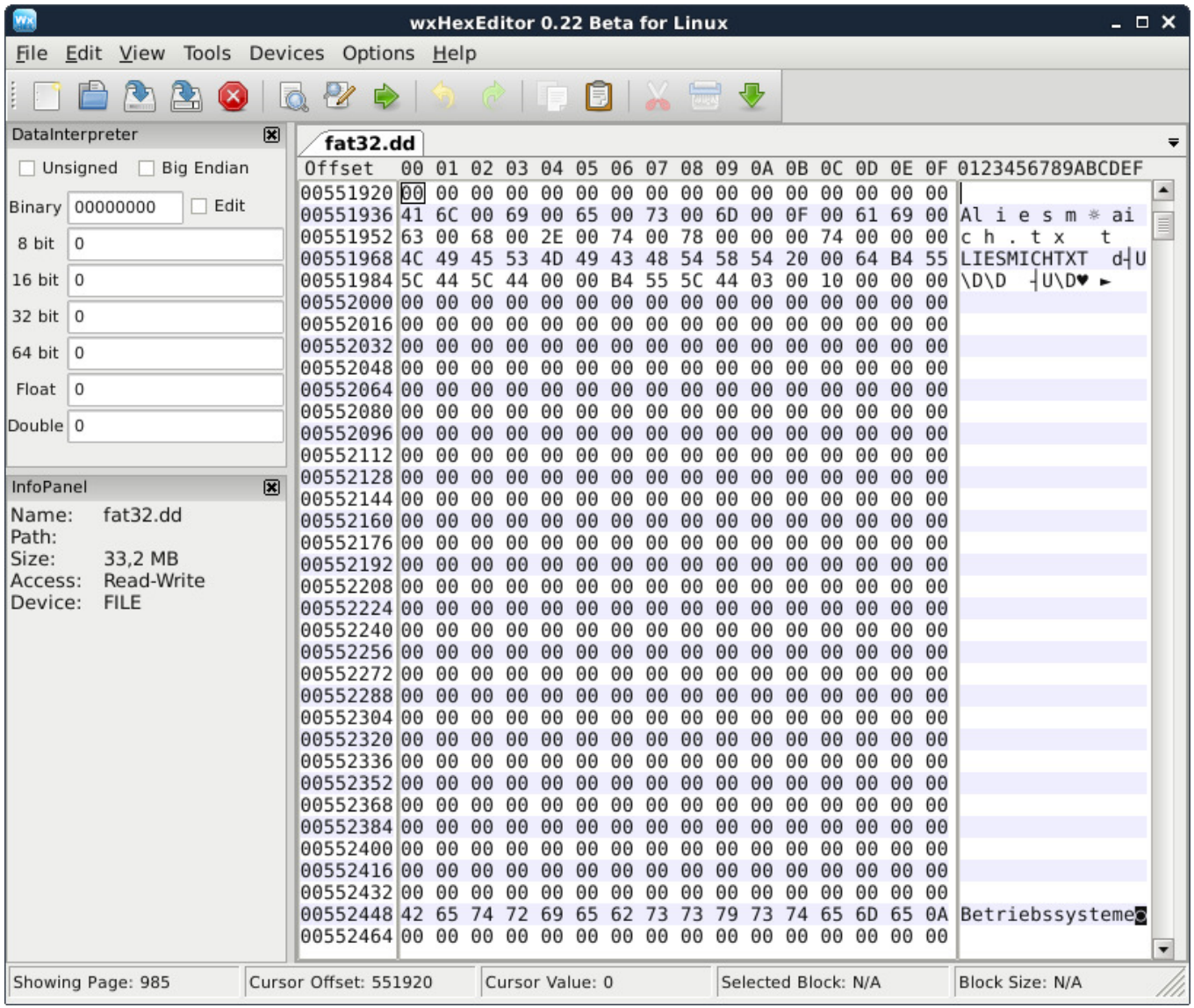
- 1.Spalte: Anzahl der vorausgegangenen Bytes $\Longrightarrow$ Offset
- 2.Spalte: Bytes der Zeile in hexadezimaler Darstellung
- 3.Spalte: Bytes der Zeile in ASCII-Darstellung

Einige Grundlagen...

- Hexadezimalsystem $\Longrightarrow$ Basis 16
- 1 Hex-Zeichen stellt 4 Bits dar
- 2 Hex-Zeichen stellen 1 Byte dar

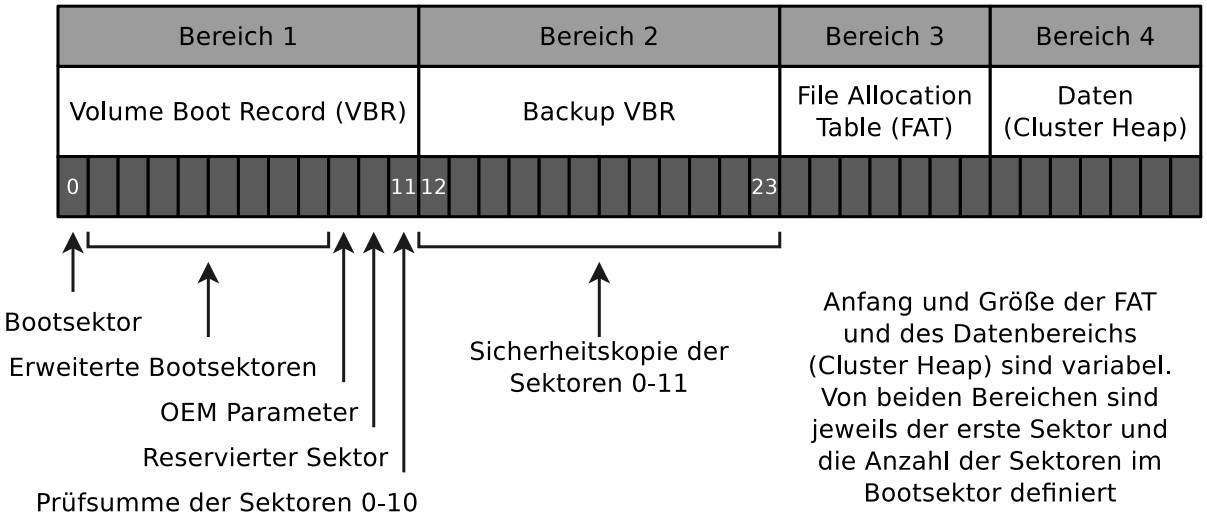
`http://dorumugs.blogspot.de/2013/01/file-system-geography-fat32.html`
`http://www.win.tue.nl/~aeb/linux/fs/fat/fat-1.html`

FAT-Dateisysteme analysieren (3/3)



exFAT

- Veröffentlicht 2006 (patentrechtlich unbedenklich seit 2019)
- Bis zu $2^{32} = 4.294.967.296$ Cluster können adressiert werden
- Clustergröße: 512 Bytes bis 32 MB
- Maximale Dateigröße: 16 EB (2^{64} Bytes)
- Einsatzgebiet: mobile Flash-Speicher (> 32 GB)
  - Weniger Schreibzugriffe als Dateisysteme mit Journal (z.B. NTFS $\Longrightarrow$ Folie 38)



Das Stammverzeichnis (Wurzelverzeichnis) hat im Gegensatz zu den übrigen FAT-Dateisystemversionen kein feste Position. Es befindet sich innerhalb des Datenbereichs und liegt dort üblicherweise nicht am Stück vor, sondern fragmentiert.

Partitionsgröße	Clustergröße
bis 256 MB	4 kB
256 MB - 32 GB	32 kB
32 GB - 256 TB	128 kB

Die Tabelle enthält die Standard-Clustergrößen unter Windows XP/Vista/7/8/10. Die Clustergröße kann beim Erzeugen des Dateisystems festgelegt werden
`https://support.microsoft.com/de-de/kb/140365`

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: Mit einem Journal über die Schreibzugriffe Buch führen
  $\Longrightarrow$ **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`
- `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 ($\Longrightarrow$ Write-Back)
    - Der Systemaufruf `sync()` übertragt die Änderungen im Page Cache, auch = Buffer Cache genannt (siehe Folie 42), 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

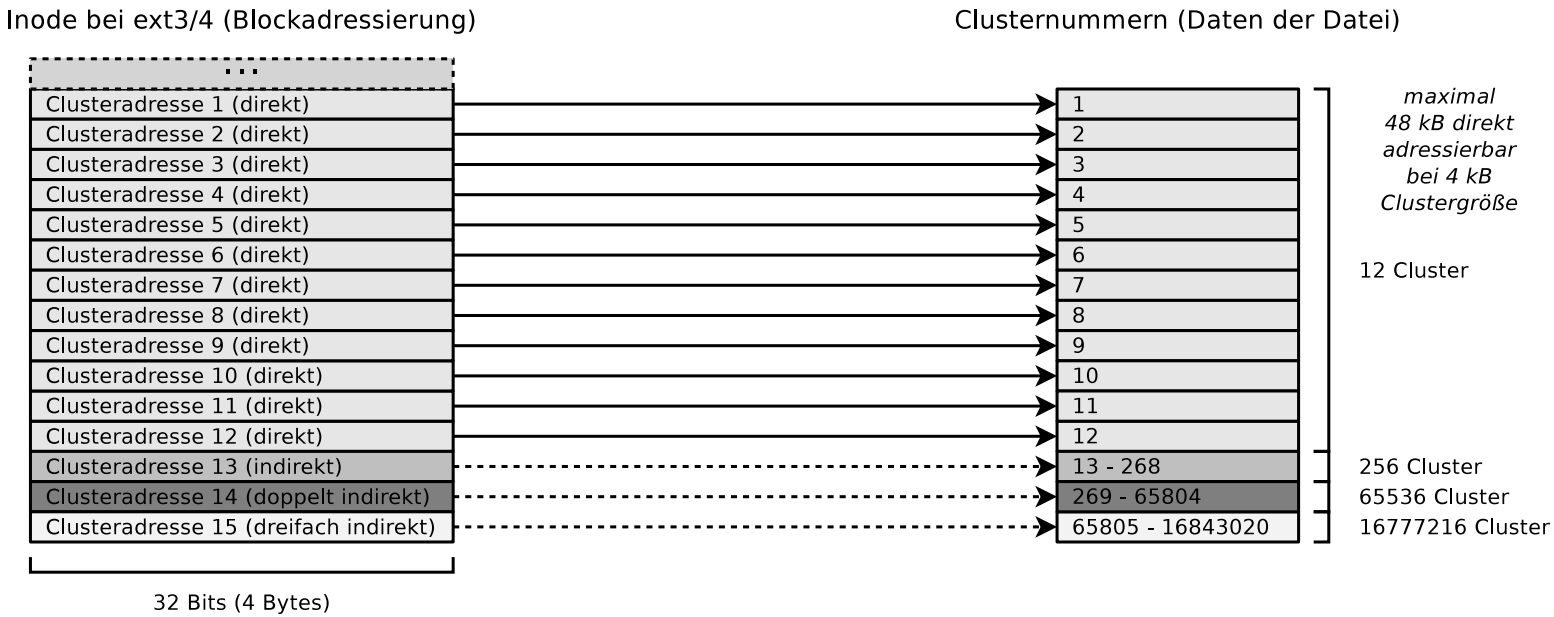
Kompromiss aus beiden Varianten: Ordered-Journaling

- 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

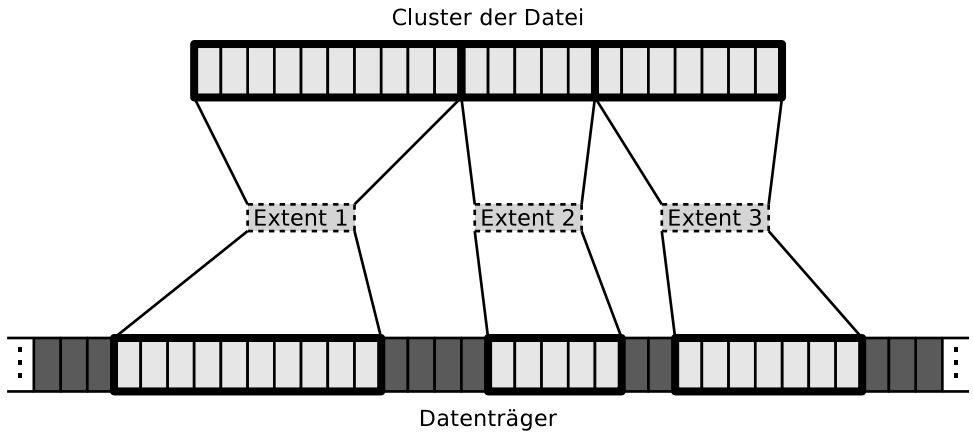


- Dieses Adressierungsschema führt bei steigender Dateigröße zu zunehmendem Overhead für Verwaltungsinformationen
- Lösung: Extents

Extent-basierte Adressierung

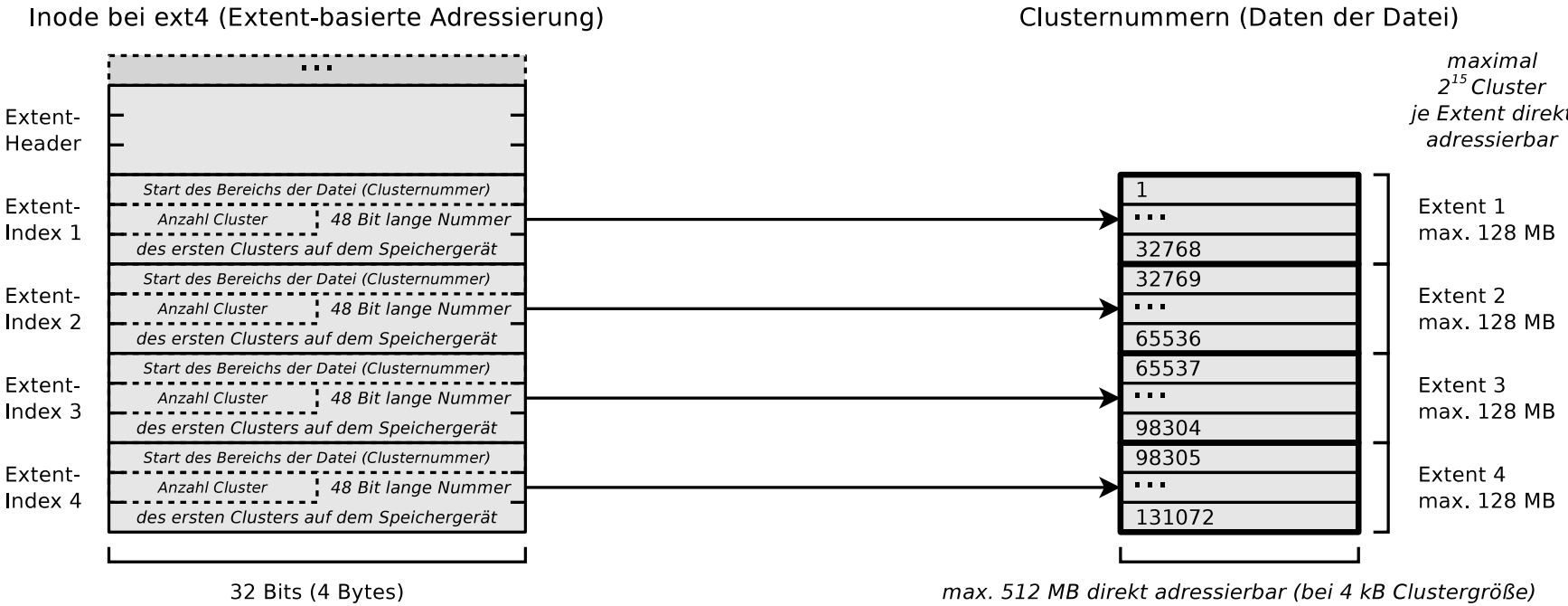
- Inodes adressieren nicht einzelne Cluster, sondern bilden möglichst große Dateibereiche auf Bereiche zusammenhängender Blöcke (**Extents**) auf dem Speichergerät ab
- Statt vieler einzelner Clusternummern sind nur 3 Werte nötig:
  - Start (Clusternummer) des Bereichs (Extents) in der Datei
  - Größe 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 ext2/3 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)

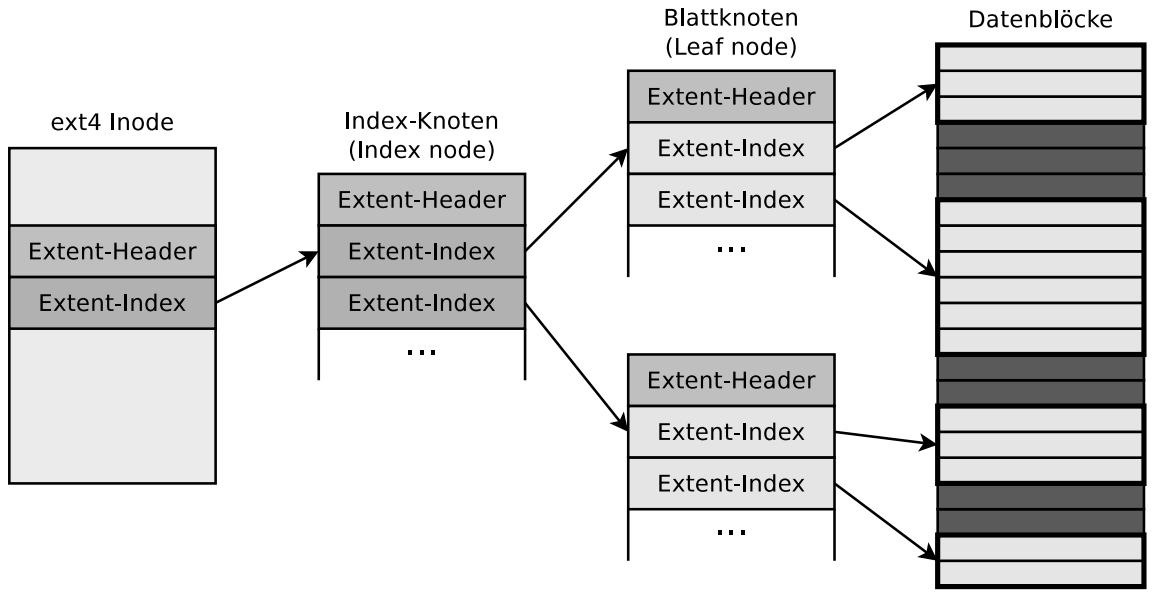


Extents können nicht größer als 128 MB (2^{15} Bits) sein, weil ext4, genau wie seine Vorgänger ext2 und ext3, die Cluster des Dateisystems in sog. Blockgruppen (siehe Folie 10) mit einer maximalen Größe von 128 MB organisiert.

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



Gute Beschreibungen zu Extents in ext4...

`https://ext4.wiki.kernel.org/index.php/Ext4_Disk_Layout#Extent_Tree`
`https://www.sans.org/blog/understanding-ext4-part-3-extent-trees/`
`https://metebalci.com/blog/a-minimum-complete-tutorial-of-linux-ext4-file-system/`
An analysis of Ext4 for digital forensics: `https://www.sciencedirect.com/science/article/pii/S1742287612000357`

NTFS – **N**ew **T**echnology **F**ile **S**ystem

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 ($\Longrightarrow$ Extents)
  - Maximale Partitionsgröße: 256 TB ($\Longrightarrow$ 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 ≤ 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`

- 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 (was bei Flash-Speicher keine negativen Auswirkungen hat)

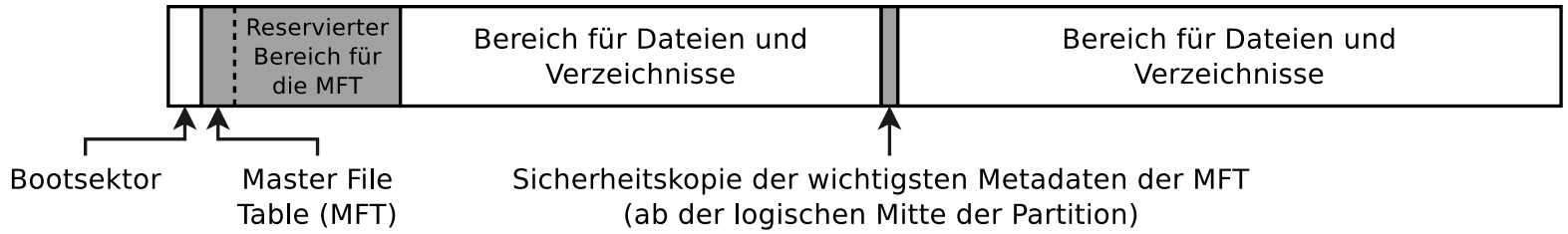
Partitionsgröße	Clustergröße
< 16 TB	4 kB
16 TB - 32 TB	8 kB
32 TB - 64 TB	16 kB
64 TB - 128 TB	32 kB
128 TB - 256 TB	64 kB

Die Tabelle enthält die Standard-Clustergrößen unter Windows 2000/XP/Vista/7/8/10. Die Clustergröße kann beim Erzeugen des Dateisystems festgelegt werden. Die Standardgröße ist 4 kB.

Quelle: `http://support.microsoft.com/kb/140365/de`

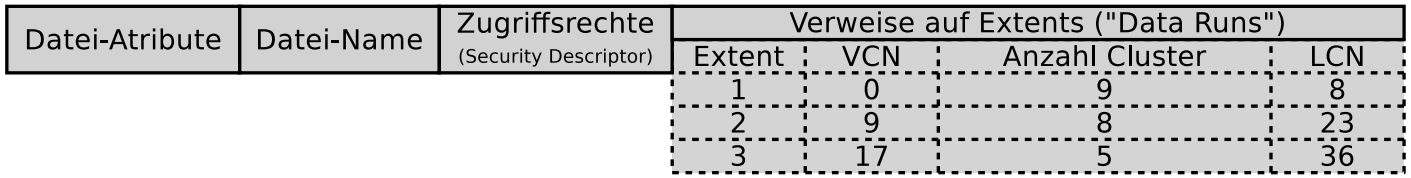
Struktur von NTFS (2/2)

Struktur des Dateisystems



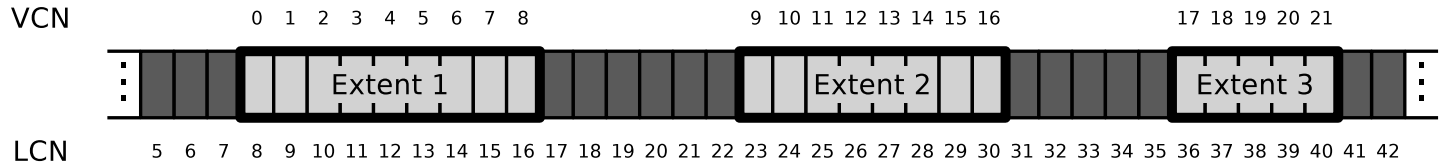
MFT-Eintrag einer Datei (≤ 900 Bytes)

MFT-Eintrag einer Datei mit Extents



(Länges jedes MFT-Eintrags: 1 kB)

Datenträger

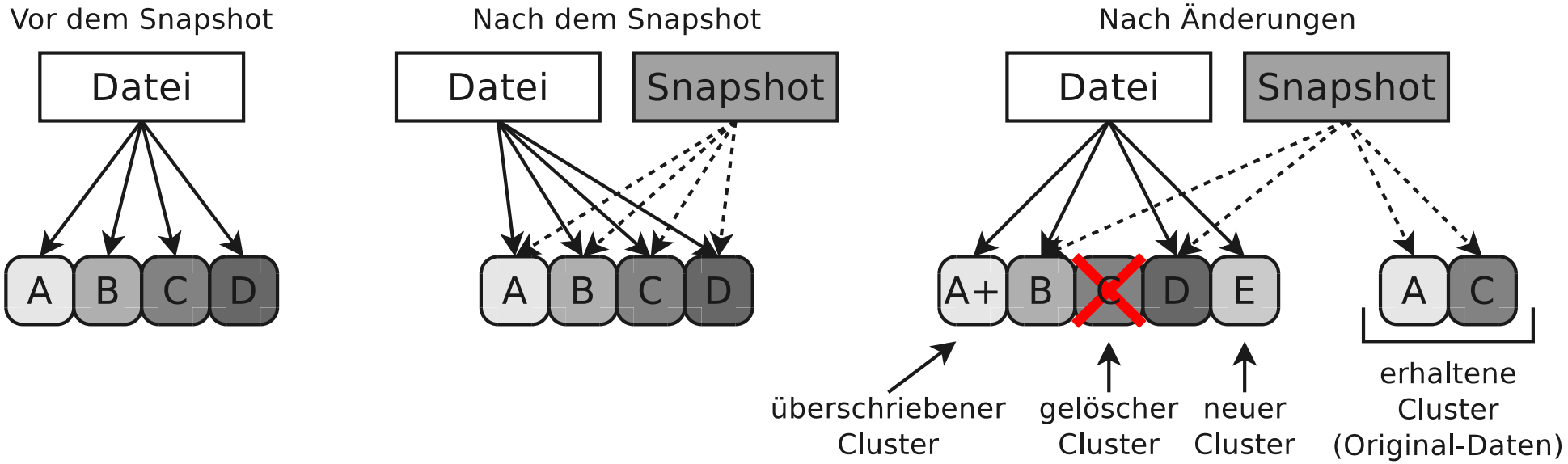


Verweist ein MFT-Eintrag auf Extents (sog. „Data Runs“), speichert er pro Extent 3 Werte

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

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

Modernstes Konzept: Copy-On-Write



- 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 ($\Longrightarrow$ **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 eine Übersicht der Speicherbelegung aus
    - Es informiert auch in der Spalte `buff/cache`, wie viel Speicherkapazität des Hauptspeichers gegenwärtig für den Page Cache verwendet wird

```
$ free -m
              total        used        free      shared  buff/cache   available
Mem:          15540        5774        1609         785        8156        8650
Swap:         10689           1       10688
```

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

https://askubuntu.com/questions/770108/what-do-the-changes-in-free-output-from-14-04-to-16-04-mean
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

Wie kann man zu Write-Through wechseln? (1/2)

- Die Schreibstrategie **Write-back** ist in vielen Hardware-und Betriebssystemkomponenten Standard (siehe Foliensätze 3, 5 und 6)
  - Wie ist es möglich, auf die Schreibstrategie **Write-through** umzuschalten?
- Der Funktionsaufruf **Windows** `CreateFileA` (`fileapi.h`) zum Erstellen oder Öffnen einer Datei bietet die Flags `FILE_FLAG_WRITE_THROUGH` und `FILE_FLAG_WRITE_THROUGH`
  - Sind diese Flags gesetzt, umgehen Schreibzugriffe den System-Cache umgehen und werden sofort an das Laufwerk weitergegeben

```
https://devblogs.microsoft.com/oldnewthing/20210729-00/?p=105494
https://learn.microsoft.com/en-us/windows/win32/api/fileapi/nf-fileapi-createfilea
```

- In **Linux** ist die globale Deaktivierung des Seitencache unmöglich
  - Es ist aber auf Anwendungs- und teilweise auf Dateisystemebene möglich
    - Es ist aber nur selten sinnvoll, wie z.B. bei Nutzung einer Anwendung mit eigenem Cache im Userspace (z.B. ein DBMS), bei speicherähnlichen Datenträgern, bei Ramdisks oder bei Performance-Tests von Geräten

Wie kann man zu Write-Through wechseln? (2/2)

- Der **Linux-Funktionsaufruf** `open` (`fcntl.h`) zum Erstellen oder Öffnen einer Datei bietet die Flags `O_DIRECT` und `O_SYNC`
  - Das Setzen dieser Flags definiert, dass Schreibzugriffe auf die Datei den Page Cache umgehen und sofort an das Laufwerk weitergegeben werden

```
https://man7.org/linux/man-pages/man2/open.2.html
https://pubs.opengroup.org/onlinepubs/9699919799.2018edition/functions/open.html
```

- Die **Option** `dax` beim Einhängen („*mounten*“) eines Dateisystems umgeht den Page Cache (bei ext2, ext4 und XFS)
  - Benötigt speicherähnliche Blockgeräte wie Persistent Memory (pmem)

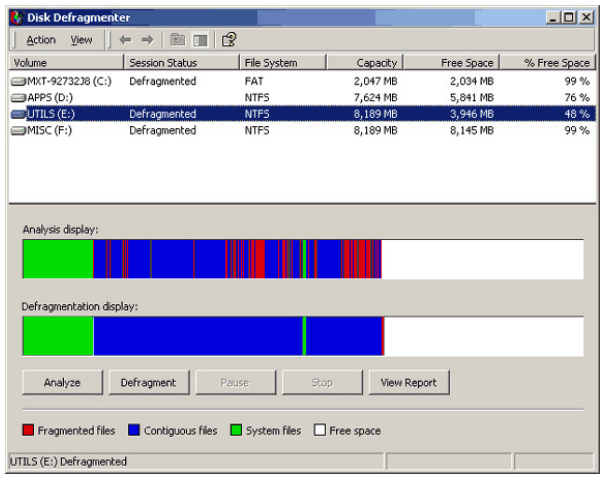
```
https://www.kernel.org/doc/Documentation/filesystems/dax.txt
https://lwn.net/Articles/787233/
```

- Dateisysteme, die das Modul **FUSE** (Filesystem in Userspace) verwenden, implementieren `direct-io` zur Umgehung des Page Cache

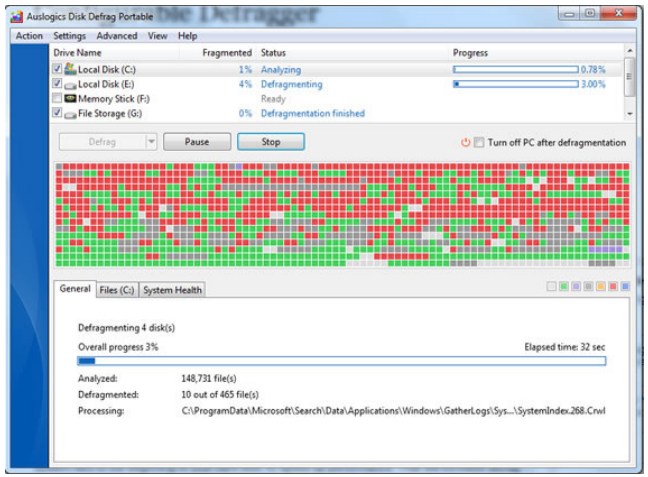
```
https://www.kernel.org/doc/Documentation/filesystems/fuse-io.txt
```

Fragmentierung

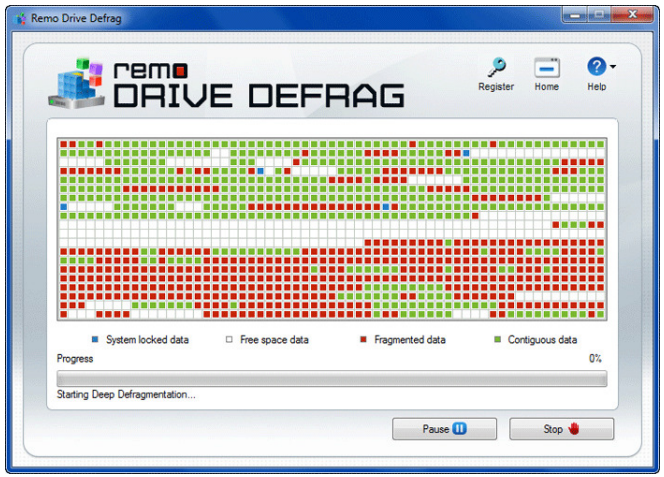
- In einem Cluster darf nur eine Datei gespeichert werden
  - Ist eine Datei größer als ein Cluster, wird sie auf mehrere verteilt
    - **Fragmentierung** beißt, dass logisch zusammengehörenden Cluster nicht räumlich beieinander sind
  - Ziel: Häufige Bewegungen des Schwungarme vermeiden
    - Liegen die Cluster einer Datei über die Festplatte verteilt, müssen die Festplattenköpfe bei Zugriffen auf die Datei mehr zeitaufwendige Positionswechsel durchführen
    - Bei SSDs spielt die Position der Cluster keine Rolle für die Zugriffsgeschwindigkeit



Bildquelle: http://windowsitpro.com



Bildquelle: http://www.teknobites.com



Bildquelle: http://www.remosoftware.com

Defragmentierung (1/3)

- Es kommen immer wieder folgende Fragen auf:
  - Warum ist es unter Linux/UNIX nicht üblich ist zu defragmentieren?
  - Kommt es unter Linux/UNIX überhaupt zu Fragmentierung?
  - Gibt es unter Linux/UNIX Möglichkeiten zu defragmentieren?
- Zu allererst ist zu klären: Was will man mit **Defragmentieren** erreichen?
  - Durch das Schreiben von Daten auf einen Datenträger kommt es zwangsläufig zu Fragmentierung
    - Die Daten sind nicht mehr zusammenhängend angeordnet
  - Eine zusammenhängende Anordnung würde das **fortlaufende Vorwärtslesen** der Daten maximal beschleunigen, da keine Suchzeiten mehr vorkommen
  - Nur wenn die Suchzeiten sehr groß sind, macht Defragmentierung Sinn
    - Bei Betriebssystemen, die kaum Hauptspeicher zum Cachen der Festplattenzugriffe verwenden, sind hohe Suchzeiten sehr negativ

Erkenntnis 1: Defragmentieren beschleunigt primär das fortlaufende Vorwärtslesen

Defragmentierung (2/3)

- Singletasking-Betriebssysteme (z.B. MS-DOS)
  - Es kann immer nur eine Anwendung laufen
    - Wenn diese Anwendung hängt, weil sie auf die Ergebnisse von Lese- und Schreibanforderungen wartet, verringert das die Systemgeschwindigkeit

Erkenntnis 2: Defragmentieren kann bei Singletasking-Betriebssystemen sinnvoll sein.
In der Praxis werden Singletasking-Betriebssysteme allerdings sehr selten verwendet

- Multitasking-Betriebssysteme
  - Es laufen immer mehrere Programme
    - **Anwendungen können fast nie große Datenmengen am Stück lesen, ohne dass andere Anwendungen ihre Lese- und Schreibanforderungen dazwischenschieben**
- Damit sich gleichzeitig laufende Programme nicht zu sehr gegenseitig behindern, lesen Betriebssysteme mehr Daten ein als angefordert
  - Das System liest einen Vorrat an Daten in den **Cache** ein, auch wenn dafür noch keine Anfragen vorliegen

Erkenntnis 3: In Multitasking-Betriebssysteme können Anwendungen fast nie große Datenmengen am Stück lesen

Defragmentierung (3/3)

- Linux-Systeme halten Daten, auf die Prozesse häufig zugreifen, automatisch im Cache
  - **Die Wirkung des Cache überwiegt bei weitem die kurzzeitigen Vorteile, die eine Defragmentierung hätte**

Erkenntnis 4: Defragmentieren hat primär einen Benchmark-Effekt
Erkenntnis 5: Dateisystemcache vergrößern bringt bessere Resultate als Defragmentieren

- Defragmentieren hat primär einen **Benchmark-Effekt**
  - In der Realität bringt Defragmentierung (unter Linux!) fast nichts
  - Es gibt Werkzeuge (z.B: `defragfs`) zur Defragmentierung unter Linux
    - Deren Einsatz ist häufig nicht empfehlenswert und sinnvoll

Gute Quelle zum Thema: `http://www.thomas-krenn.com/de/wiki/Linux_Page_Cache`