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**
  - eine Übersicht über die **Linux-Dateisysteme** und deren Eckdaten
  - 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 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
- sind eine Schicht des Betriebssystems ($\Longrightarrow$ Systemsoftware)
  - Prozesse und Benutzer greifen auf Dateien abstrakt über deren Dateinamen und nicht direkt auf Speicheradressen zu
- sollen wenig Overhead für Verwaltungsinformationen benötigen

Linux-Dateisysteme

Vergangenheit	Gegenwart	Zukunft
klassische Dateisysteme	**Dateisysteme mit Journal**	**Copy-on-Write**
Minix (1991)		**Btrfs** (nicht stabil)
ext2 (1993)	**ext3** (2001)	**ext4** (2008)
	ReiserFS (2001)	**Reiser4** (2009)
	JFS (2002) **XFS** (2002)	

In Klammer ist jeweils das Jahr der Integration in den Linux-Kernel angegeben

Zudem existieren noch:

- Shared Storage-Dateisysteme: OCFS2, GPFS, GFS2, VMFS
- Verteilte Dateisysteme: Lustre, AFS, Ceph, HDFS, PVFS2, GlusterFS
- Dateisysteme für Flash-Speicher: JFFS, JFFS2, YAFFS
- ...

Grundbegriffe von Linux-Dateisystemen

- Wird eine **Datei** angelegt, wird auch ein Inode angelegt
- **Inode** (*Index Node*)
  - 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
- 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 genannt werden

Clustergröße

- 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
- Unter Linux gilt: Clustergröße $\leq$ Größe der Speicherseiten (Pagesize)
  - Die Pagesize hängt von der Architektur ab
  - x86 = 4 kB, Alpha und Sparc = 8 kB, IA-64 = 4/8/16/64 kB

Benennung von Dateien und Verzeichnissen

- Wichtige Eigenschaft von Dateisystemen aus Benutzersicht:
  - Möglichkeiten der Benennung von Dateien und Verzeichnisse
- Dateisysteme haben diesbezüglich unterschiedlichste Eckdaten
  - Länge der Dateinamen:
    - Alle Dateisysteme akzeptieren Zeichenketten von 1 bis 8 Buchstaben als Dateinamen
    - Aktuelle Dateisysteme akzeptieren deutlich längere Dateinamen und auch Zahlen und Sonderzeichen im Dateinamen
  - Groß- und Kleinschreibung:
    - DOS/Windows-Dateisysteme unterscheiden nicht zwischen Groß- und Kleinschreibung
    - UNIX-Dateisysteme unterscheiden zwischen Groß- und Kleinschreibung

Bedeutung von Dateinamen

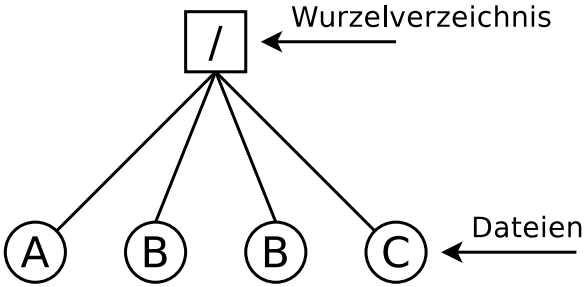
- Aktuelle Dateisysteme unterstützen zweigeteilte Dateinamen
- Der zweite Teil, die **Endung** (*Extension*), wird verwendet, um auf den Dateiinhalt hinzuweisen
  - Die Endung ist eine kurze Folge von Buchstaben und Zahlen nach dem letztem Punkt im Dateinamen
- Manche Dateien haben 2 oder mehr Endungen. z.B. `programm.c.Z`
  - Die Endung `.c` zeigt an, dass die Datei C-Quelltext enthält
  - Die Endung `.Z` steht für den Ziv-Lempel-Kompressionsalgorithmus
- Unter UNIX haben Dateiendungen ursprünglich keine Bedeutung
  - Die Dateiendung soll den Eigentümer nur daran erinnern, was für Daten sich in der Datei befinden
- Unter Windows spielen Dateiendungen von je her eine große Rolle und werden Anwendungen zugewiesen

Datenzugriffe mit einem Cache beschleunigen

- Moderne Betriebssysteme beschleunigen Zugriffe auf gespeicherte Daten mit einem **Cache** (genannt *Buffer Cache* oder *Page Cache*) im Hauptspeicher
- Wird eine Datei lesend angefragt, schaut der Kernel zuerst im Cache nach, ob die Datei dort vorliegt
  - Liegt die Datei nicht im Cache vor, wird sie in den Cache geladen
- Der 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**)
  - DOS und Windows verwenden den Puffer *Smartdrive*
  - Linux puffert automatisch so viele Daten wie Platz im Hauptspeicher ist
    - Vorteil: Höhere System-Geschwindigkeit
    - Nachteil: Stürzt das System ab, kann es zu Inkonsistenzen kommen

Verzeichnisstruktur mit einer Ebene

- Dateisysteme stellen **Verzeichnisse** (Ordner) zur Verfügung, um die Daten zu organisieren
  - Verzeichnisse sind nur Text-Dateien, die die Namen und Pfade von Dateien enthalten
- Einfachste Verzeichnisstruktur:
  - Das **Wurzelverzeichnis** enthält alle Dateien



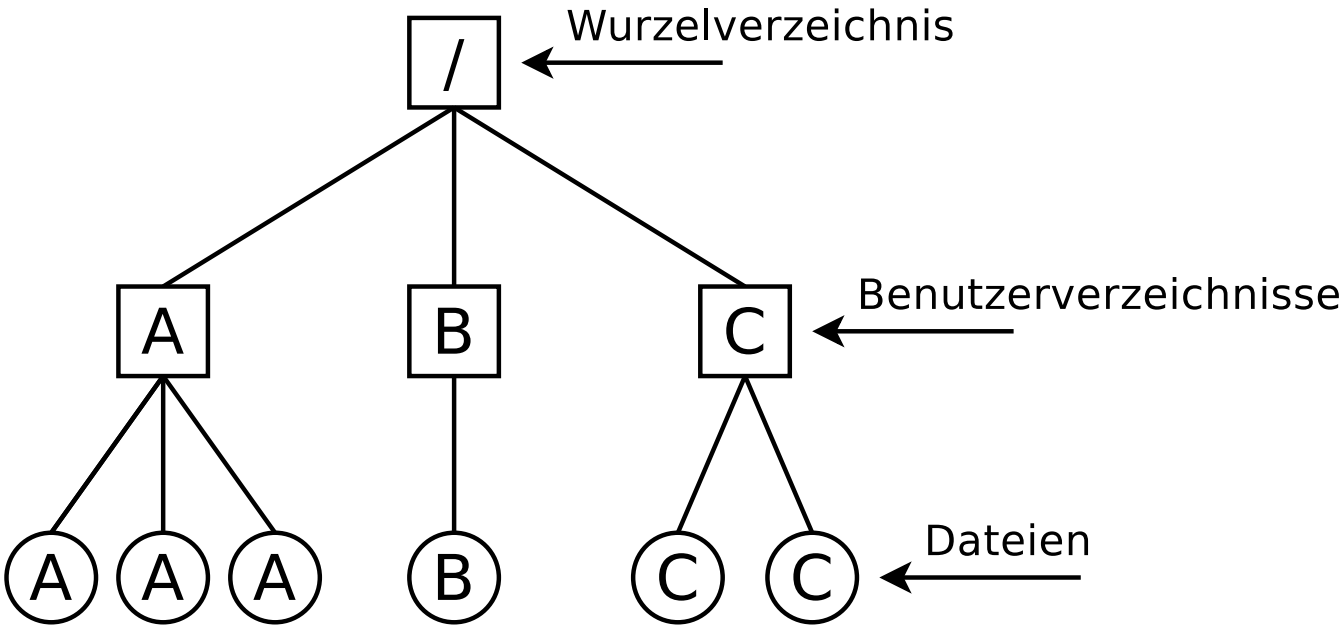
- Situation auf frühen Computern:
  - nur ein Benutzer
  - geringe Speicherkapazität

$\Longrightarrow$ nur wenige Dateien

- Nachteil: Verursacht Probleme in Mehrbenutzersystemen
  - Will ein Benutzer eine Datei anlegen, und die Datei eines anderen Benutzers hat bereits den gleichen Namen, wird diese überschrieben

Verzeichnisstruktur mit zwei Ebenen

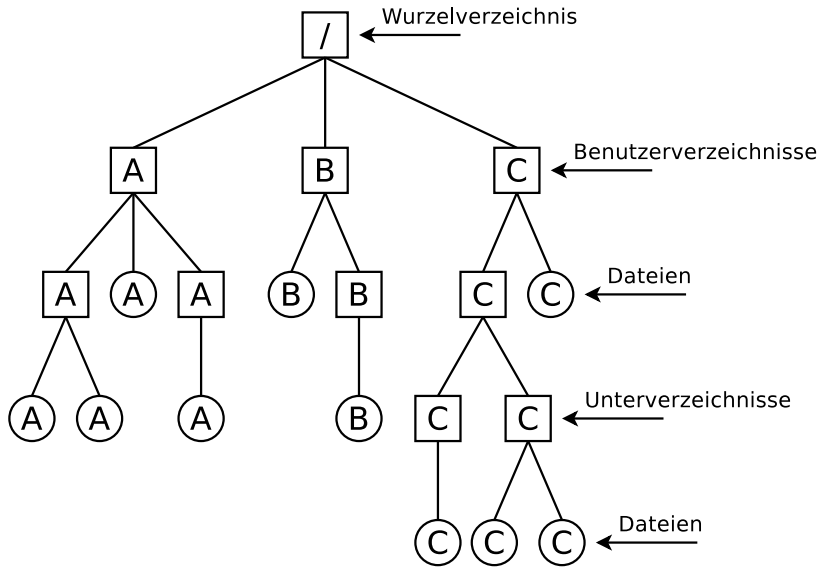
- Herausforderung: Verschiedene Benutzer verwenden die gleichen Dateinamen
- Lösung: Jeder Benutzer bekommt sein eigenes, privates Verzeichnis



- So ist es kein Problem, wenn mehrere Benutzer Dateien mit gleichen Dateinamen anlegen

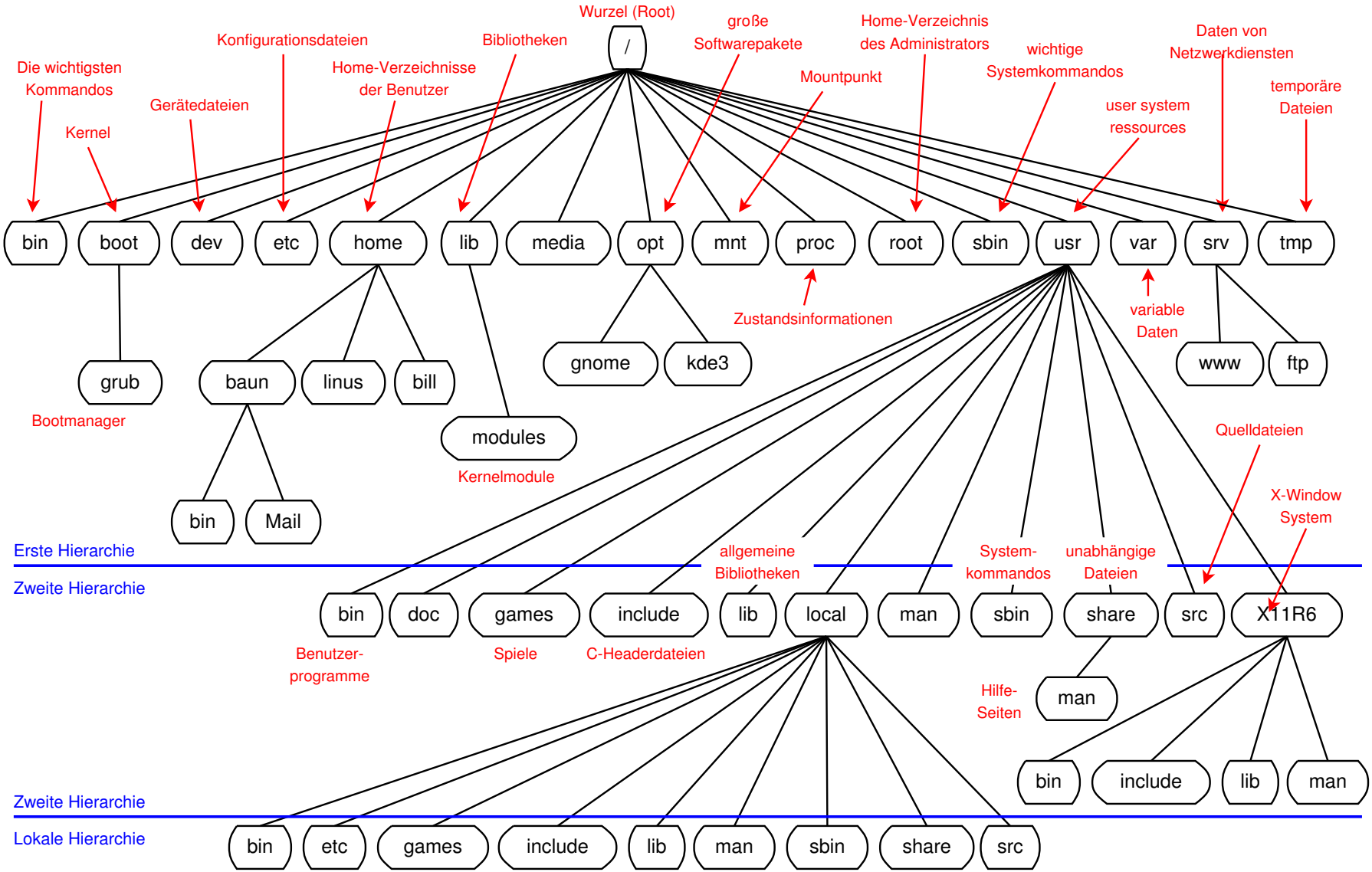
Hierarchische Verzeichnisstruktur

- Verzeichnisstrukturen mit 2 Ebenen sind nicht immer ausreichend
  - Bei vielen Dateien genügt es nicht, die Dateien nach Benutzern separieren zu können
- Es ist hilfreich, Dateien nach ihrem Inhalt und/oder der Zugehörigkeit zu Projekten bzw. Anwendungen zu sortieren



- In einem Baum von Verzeichnissen können die Benutzer ihre Dateien einsortieren und beliebig viele Verzeichnisse anlegen
- Die Verzeichnisstrukturen nahezu aller heutigen Betriebssysteme arbeiten nach dem hierarchischen Prinzip
  - Gelegentliche Ausnahmen sind sehr kleine, eingebettete Systeme

Linux/UNIX-Verzeichnisstruktur



2 verschiedene Arten von Pfadnamen existieren

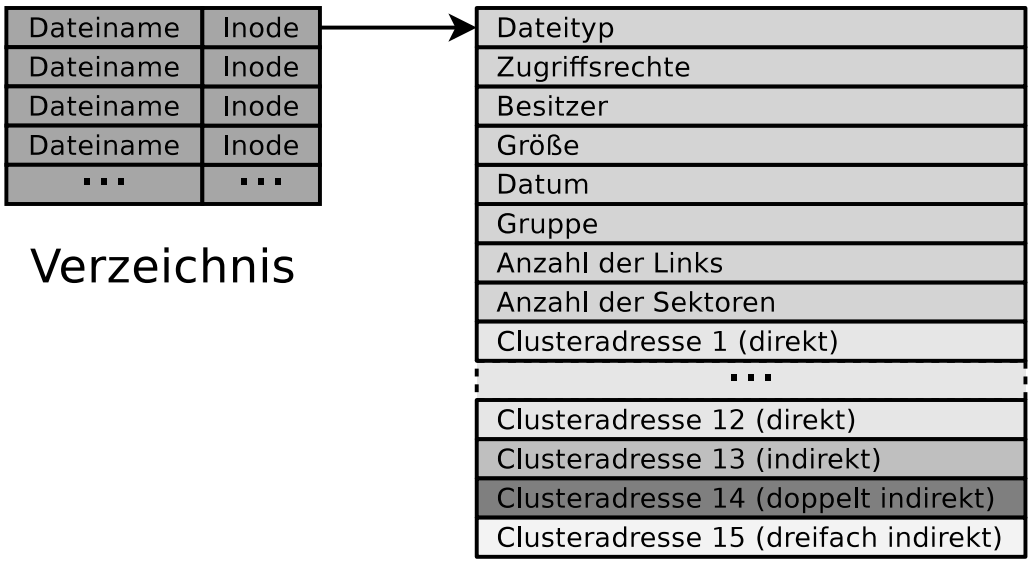
1. **Absolute Pfadnamen**
   - Beschreiben dieser den kompletten Pfad **von der Wurzel bis zur Datei**
   - Absolute Pfade funktionieren immer
     - Das aktuelle Verzeichnis ist egal
   - Beispiel: `/usr/src/linux/arch/i386/boot/bzImage`
2. **Relative Pfadnamen**
   - Wird immer in Verbindung mit dem aktuellen Verzeichnis gesehen
   - Alle Pfade, die **nicht mit der Wurzel** beginnen
   - Beispiel: `Vorlesung_06/bts_SS2016_vorlesung_06.tex`

Der Separator trennt die Komponenten des Pfads voneinander und steht auch immer für das Wurzelverzeichnis

System	Separator	Beispiel
Linux/UNIX	/	/var/log/messages
DOS/Windows	\	\var\log\messages
MacOS (früher)	:	:var:log:messages
MULTICS	>	>var>log>messages

Blockadressierung am Beispiel ext2/3/4

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



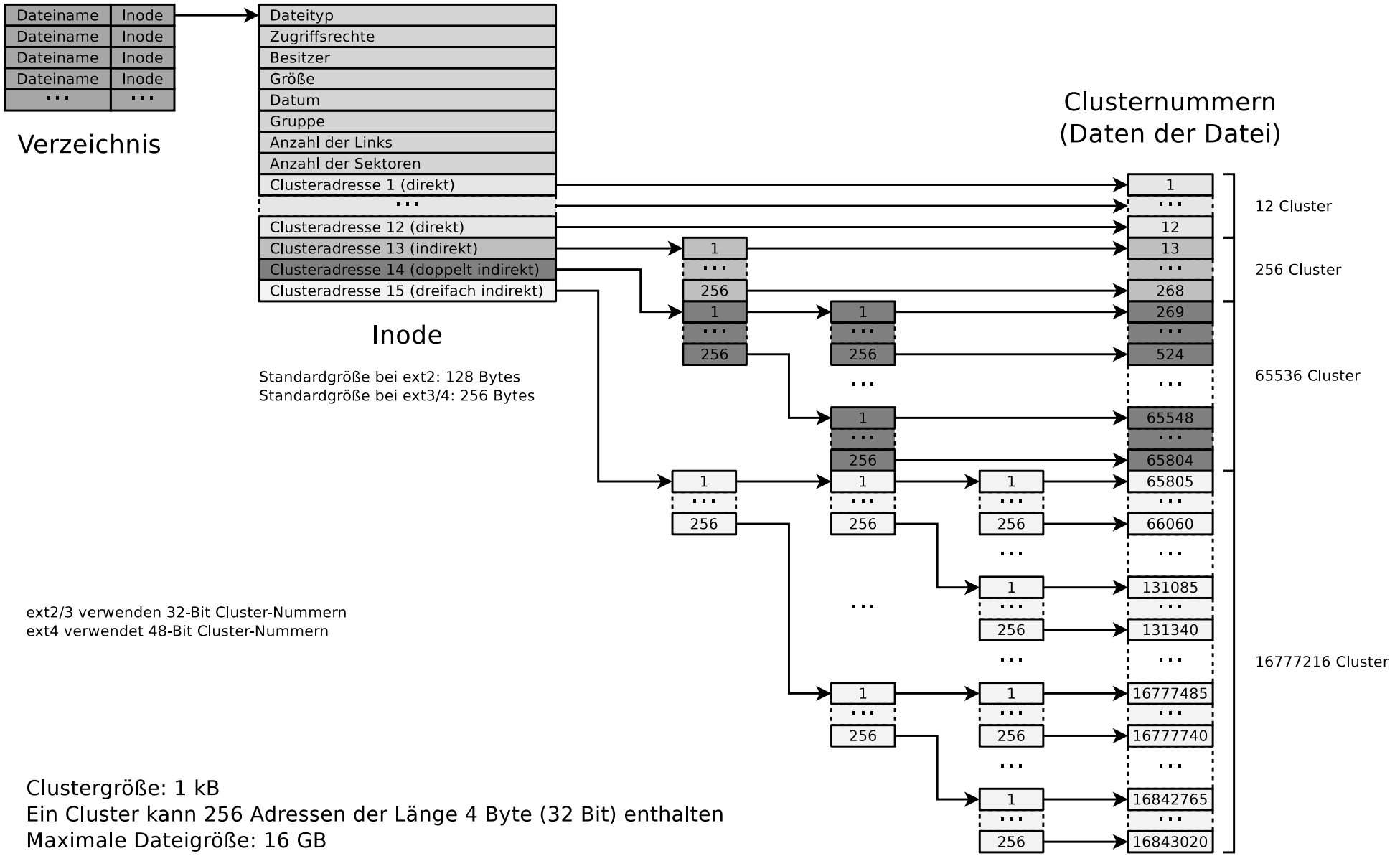
- Benötigt eine Datei mehr Cluster, wird indirekt adressiert mit Clustern, deren Inhalt Clusternummern sind
- Blockadressierung verwenden Minix, ext2/3/4, ReiserFS und Reiser4

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



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
- Speicher wird als lineare Kette gleichgroßer Blöcke (1-8 kB) dargestellt
- Ein Minix-Dateisystem enthält nur 6 Bereiche
  - Die einfache Struktur macht es für die Lehre optimal

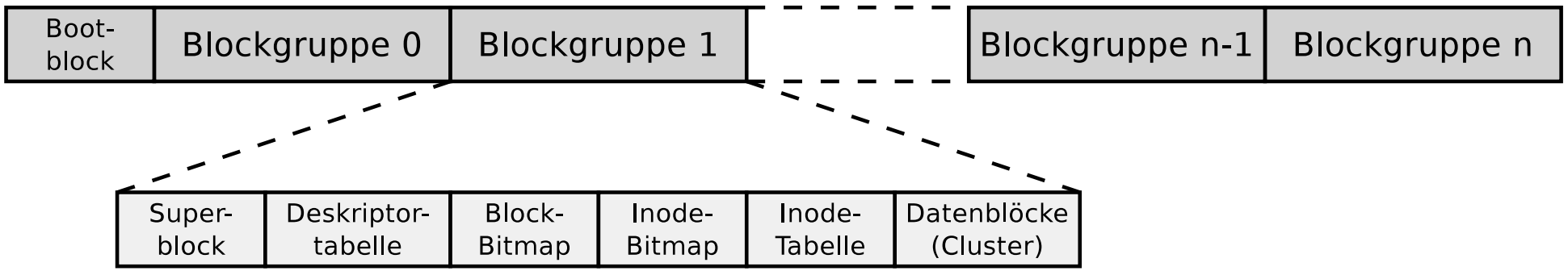
Bereiche in einem Minix-Dateisystem

Bereich 1	Bereich 2	Bereich 3	Bereich 4	Bereich 5	Bereich 6
Bootblock *1 Block*	Superblock *1 Block*	Inodes-Bitmap *1 Block*	Cluster-Bitmap *1 Block*	Inodes *15 Blöcke*	Daten *Rest*

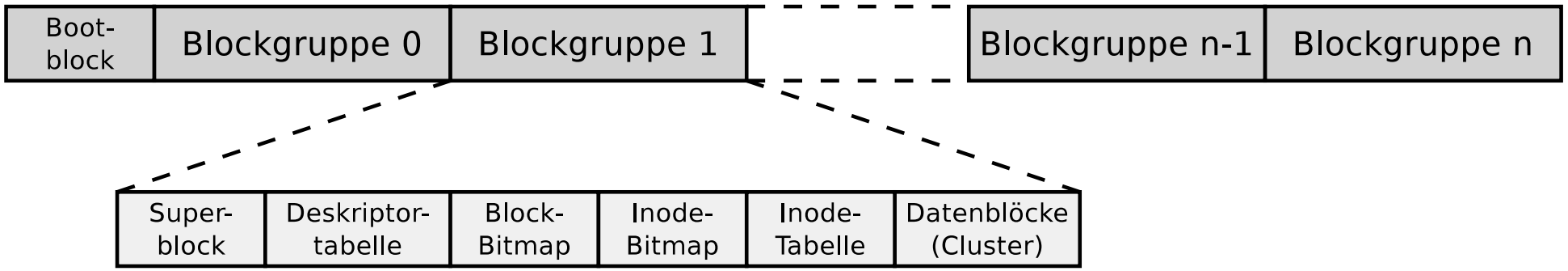
- **Bootblock**. Enthält den Boot-Loader, der das Betriebssystem startet
- **Superblock**. Enthält Informationen über das Dateisystem,
  - z.B. Anzahl der Inodes und Cluster
- **Inodes-Bitmap**. Enthält eine Liste aller Inodes mit der Information, ob der Inode belegt (Wert: 1) oder frei (Wert: 0) ist
- **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

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 1.024 Bytes, kann jede Blockgruppe maximal 8.192 Cluster umfassen
  - Vorteil der Blockgruppen: Die Inodes (Metadaten) liegen physisch nahe bei den Clustern, die sie adressieren



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

File Allocation Table (FAT)

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

QDOS = Quick and Dirty Operating System

- 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**)

Bereiche in einem FAT-Dateisystem (1/2)

Bereich 1	Bereich 2	Bereich 3	Bereich 4	Bereich 5	Bereich 6
Bootsektor	Reservierte Sektoren	FAT 1	FAT 2	Stamm-verzeichnis	Datenbereich

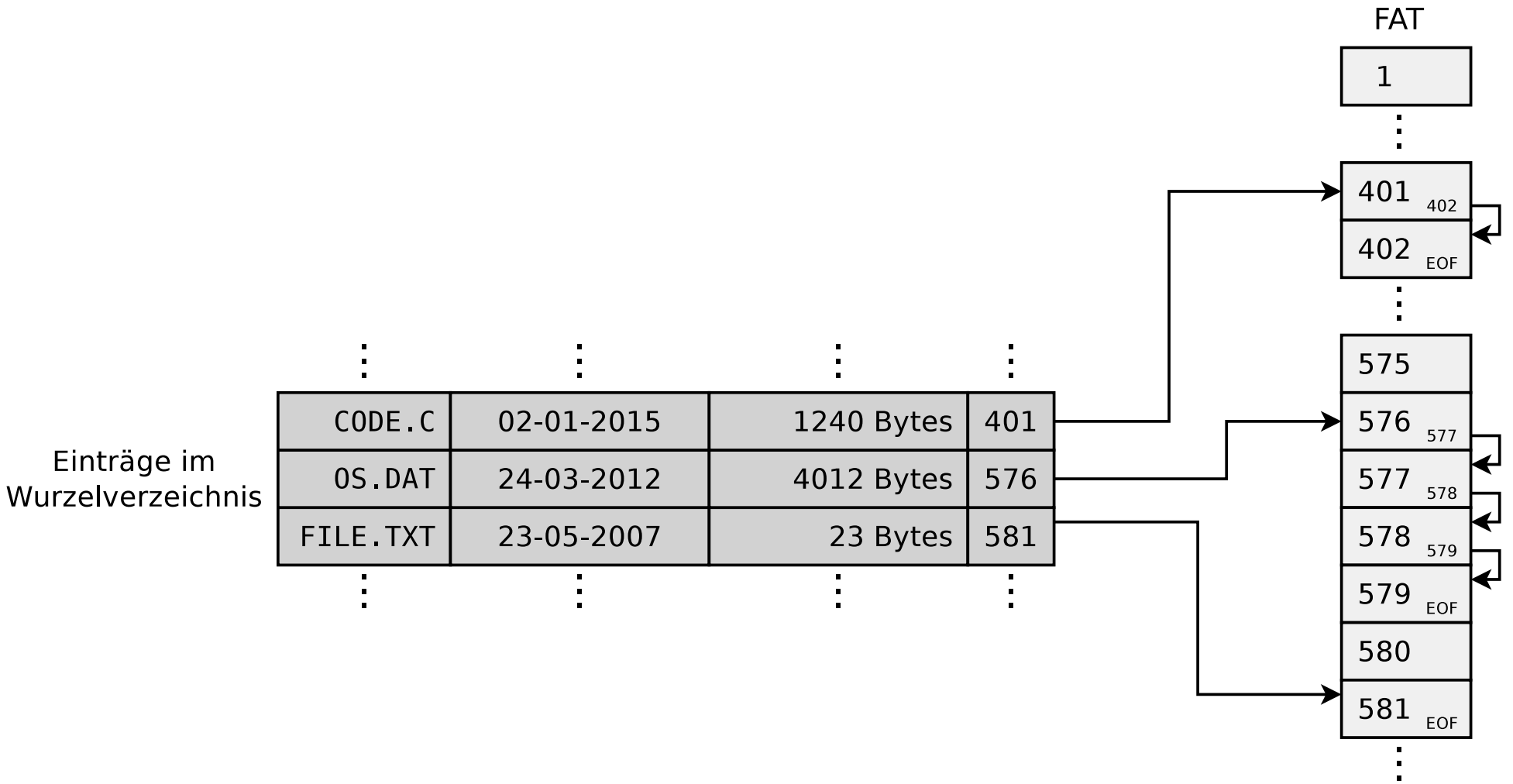
- Im **Bootsektor** liegen ausführbarer x86-Maschinencode, der das Betriebssystem starten soll, und Informationen über das Dateisystem:
  - Blockgröße des Speichermediums (512, 1.024, 2.048 oder 4.096 Bytes)
  - Anzahl der Blöcke pro Cluster
  - Anzahl der Blöcke (Sektoren) auf dem Speichermedium
  - 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	FAT 1	FAT 2	Stamm-verzeichnis	Datenbereich

- 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**

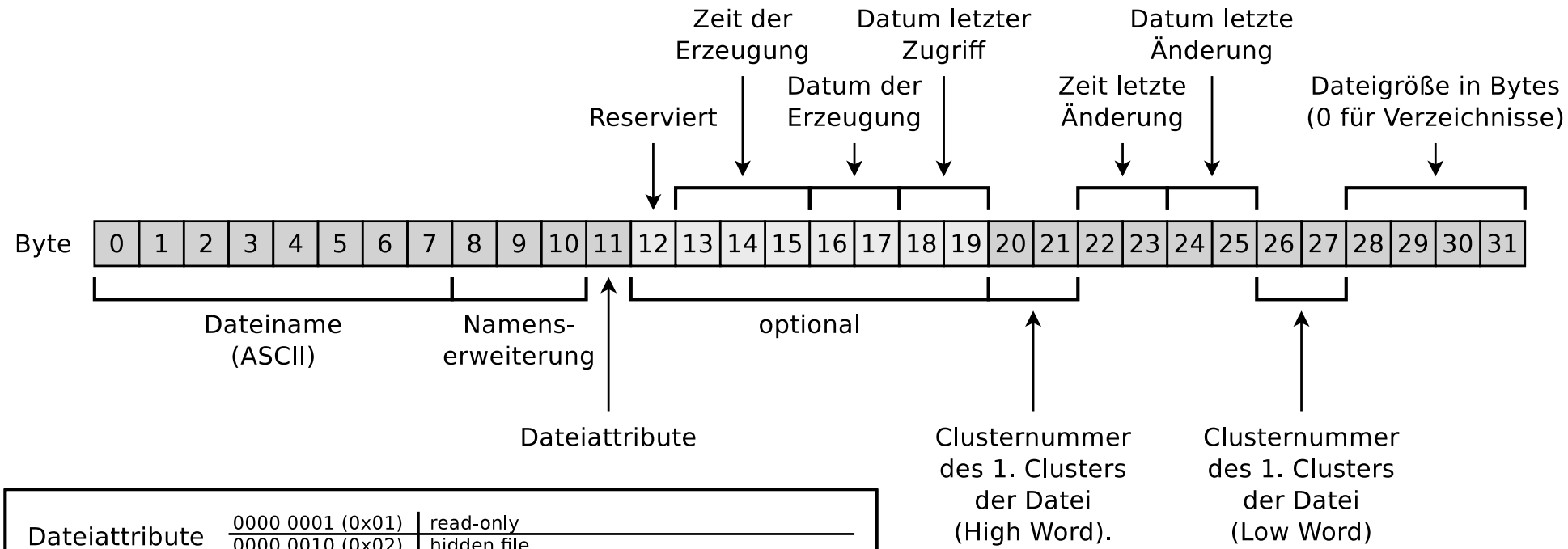
Stammverzeichnis (Wurzelverzeichnis) und FAT



Das Thema FAT ist anschaulich erklärt bei...

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

Struktur der Einträge im Stammverzeichnis



Dateiattribute	
0000 0001 (0x01)	read-only
0000 0010 (0x02)	hidden file
0000 0100 (0x04)	system file
0000 1000 (0x08)	Name des Datenträgers (volume label)
0000 1111 (0x0f)	long file name
0001 0000 (0x10)	directory
0010 0000 (0x20)	archive

Warum ist 4 GB die maximale Dateigröße unter FAT32?

Es stehen nur 4 Bytes für die Angabe der Dateigröße zur Verfügung.
Eine Dateigröße von 0 Bits macht keinen Sinn, weil es unmöglich ist.
Darum ist die maxmimale Dateigröße sogar nur $2^{32} - 1 = 4.294.967.295$ Bit

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 $2^{16} = 65.524$ 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

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

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

FAT32

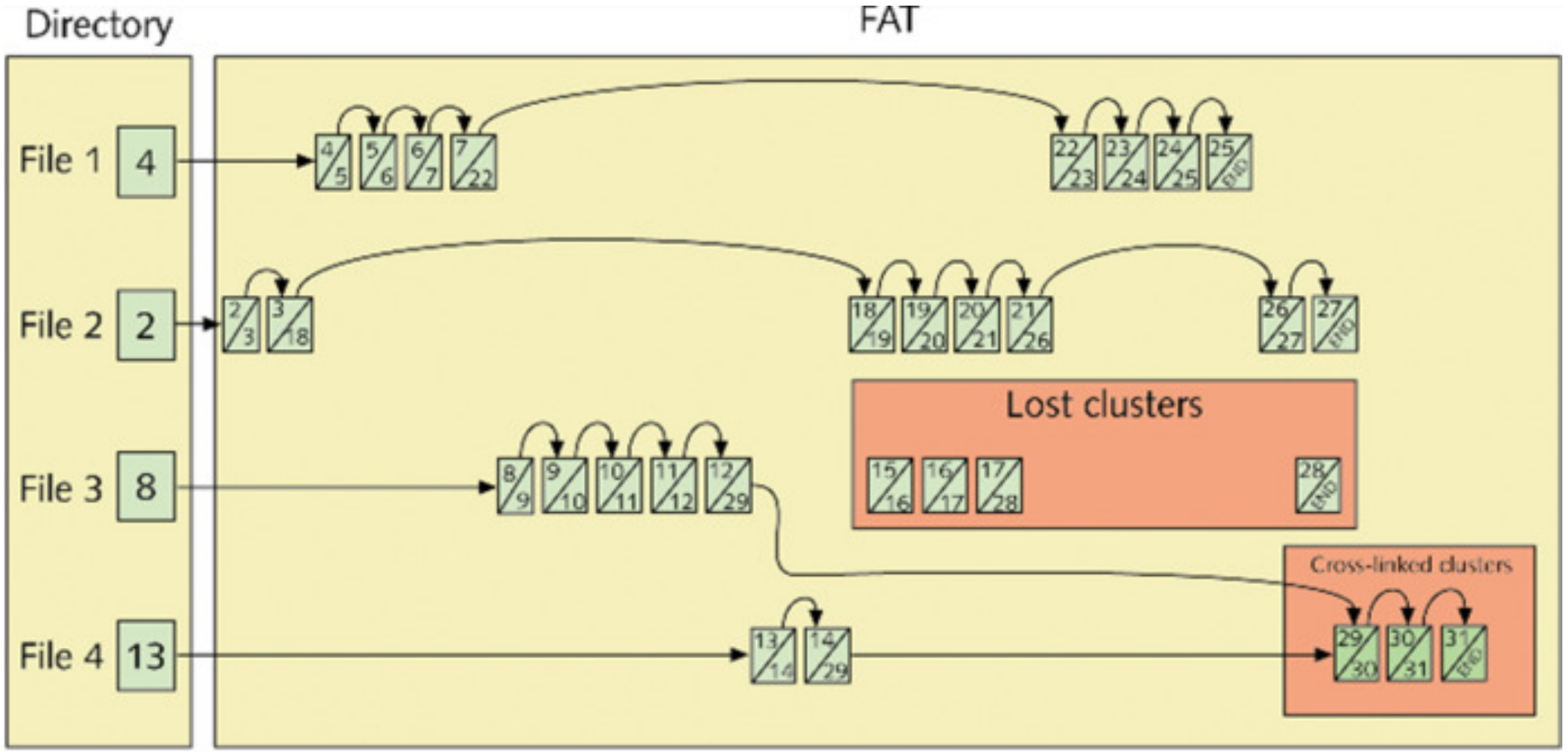
- Erschien 1997 als Reaktion auf die höhere Festplattenkapazität und weil Cluster $>$ 32 kB sehr viel Speicher verschwenden
- 32 Bits pro Eintrag in der FAT stehen für Clusternummern zur Verfügung
  - 4 Bits sind reserviert
  - Darum können nur $2^{28} = 268.435.456$ Cluster adressiert werden
- Clustergröße: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

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

Gefahr von Inkonsistenzen im Dateisystem



Lost and cross-linked clusters

Quelle: `http://www.sal.ksu.edu/faculty/tim/ossg/File_sys/file_system_errors.html`

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

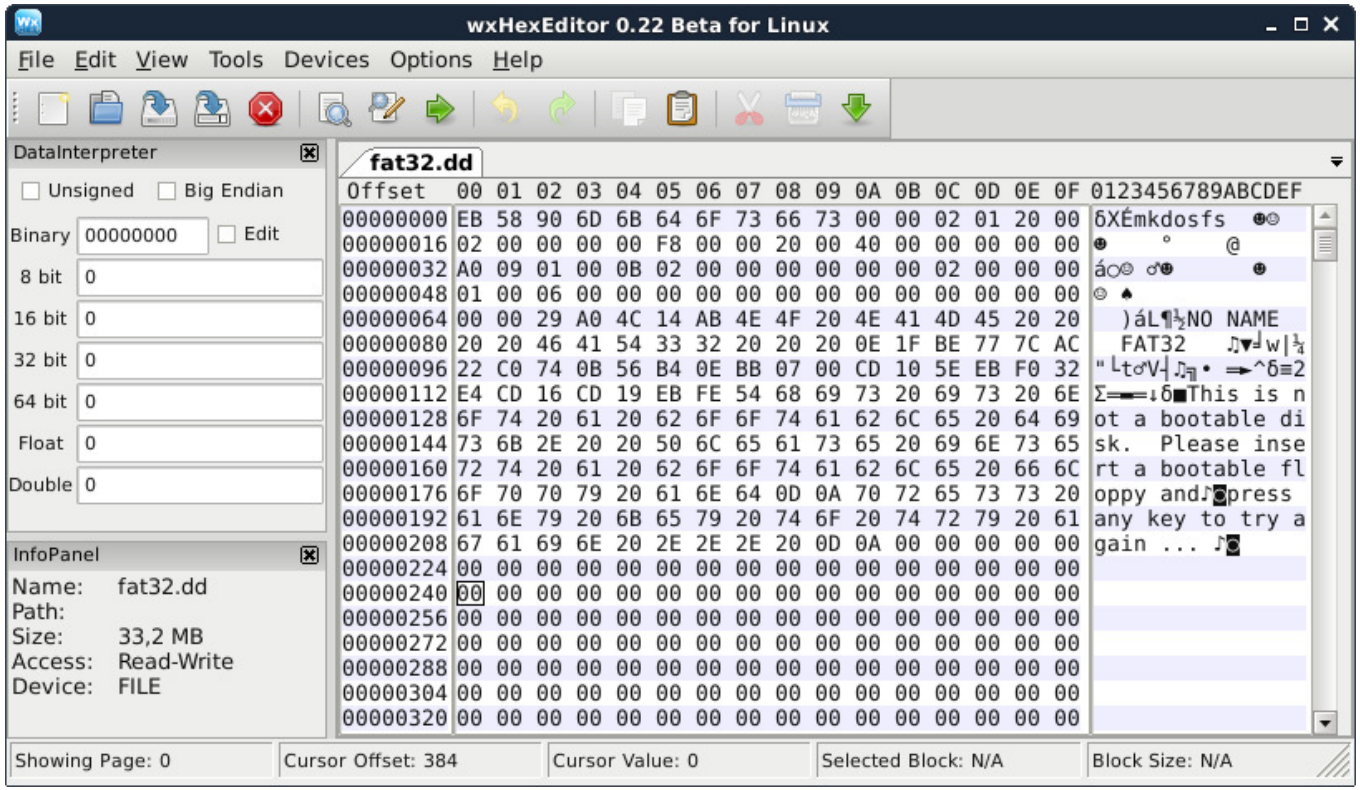
Lange Dateinamen – Long File Name Support (LFN)

- 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 35)
- 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 24)
- Besonderheit: Groß/Kleinschreibung wird angezeigt, aber ignoriert

FAT-Dateisysteme analysieren (1/3)

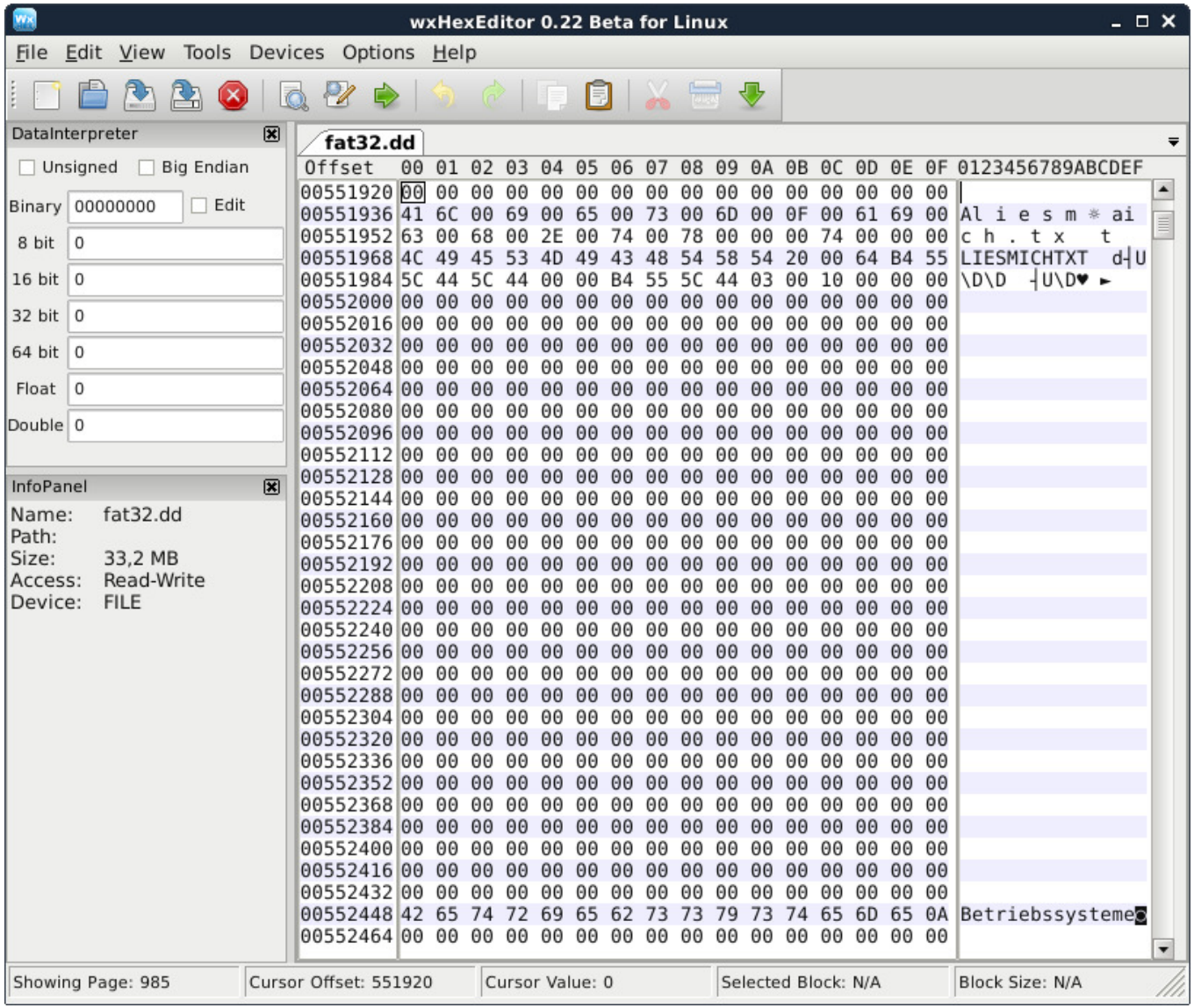
```
# dd if=/dev/zero of=./fat32.dd bs=1024000 count=34
34+0 Datensätze ein
34+0 Datensätze aus
34816000 Bytes (35 MB) kopiert, 0,0213804 s, 1,6 GB/s
# mkfs.vfat -F 32 fat32.dd
mkfs.vfat 3.0.16 (01 Mar 2013)

# 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)



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)



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

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

- 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
  - Maximale Länge der Dateinamen: 255 Zeichen
  - Dateinamen können fast beliebige Unicode-Zeichen enthalten
    - Ausnahmen: `\0` und `/`
- Clustergröße: 512 Bytes bis 64 kB

Kompatibilität zu MS-DOS

- NTFS und VFAT speichert für jede Datei einen eindeutigen Dateinamen im Format 8.3
  - So können Microsoft-Betriebssysteme ohne NTFS- und VFAT-Unterstützung auf Dateien auf NTFS-Partitionen 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 Buchstaben 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`

Struktur von NTFS

- 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`

- 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

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

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 $\Longrightarrow$ 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, `http://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 (= Buffer Cache) auf die HDD/SDD
  - Vorteil: Konsistenzprüfungen dauern nur wenige Sekunden
  - Nachteil: Datenverlust durch einen Systemabsturz ist weiterhin möglich
  - Einzige Alternative bei XFS, optional bei ext3/4 und ReiserFS
  - NTFS bietet 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

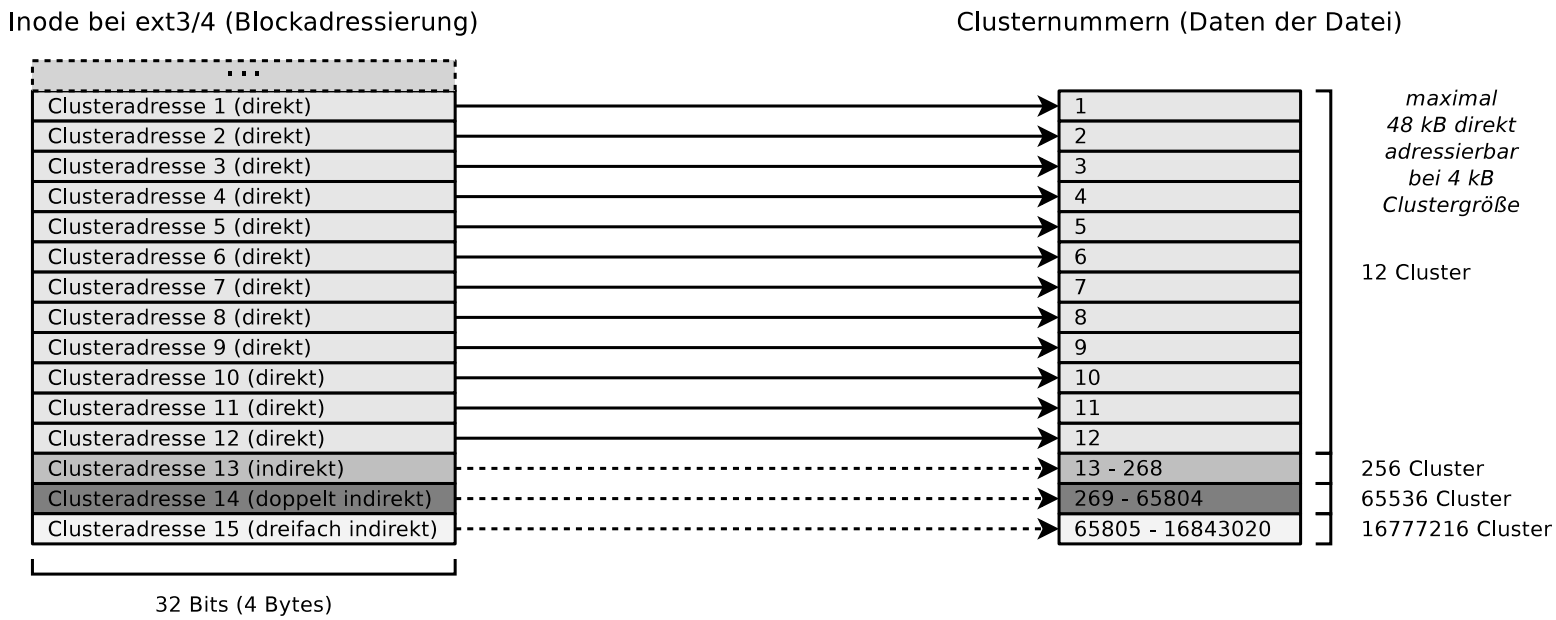
Die Alternative ist also hohe Datensicherheit und hohe Schreibgeschwindigkeit

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

Problem des Overheads für Verwaltungsinformationen

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

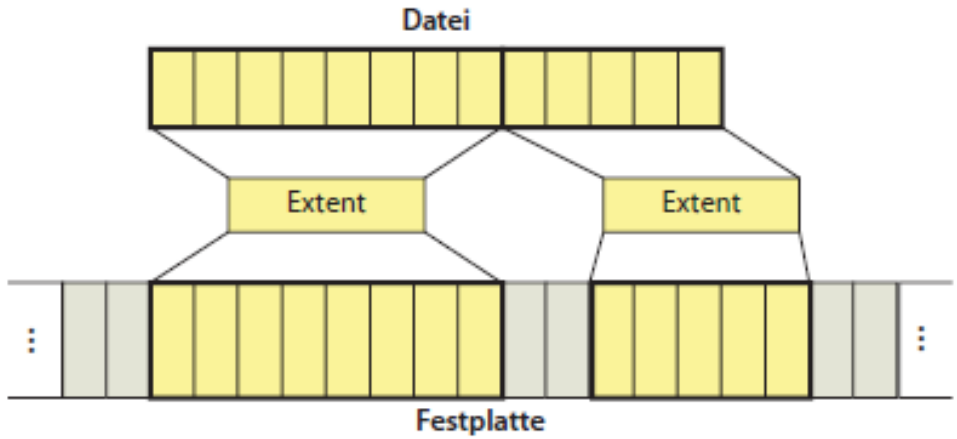


- Dieses Adressierungsschema führt bei steigender Dateigröße zu zunehmendem Overhead für Verwaltungsinformationen
- Lösung: Extents

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 des Bereichs der Datei (Clusternummer)
  - Größe des Bereichs in der Datei (in Clustern)
  - Nummer des ersten Blocks auf dem Speichergerät

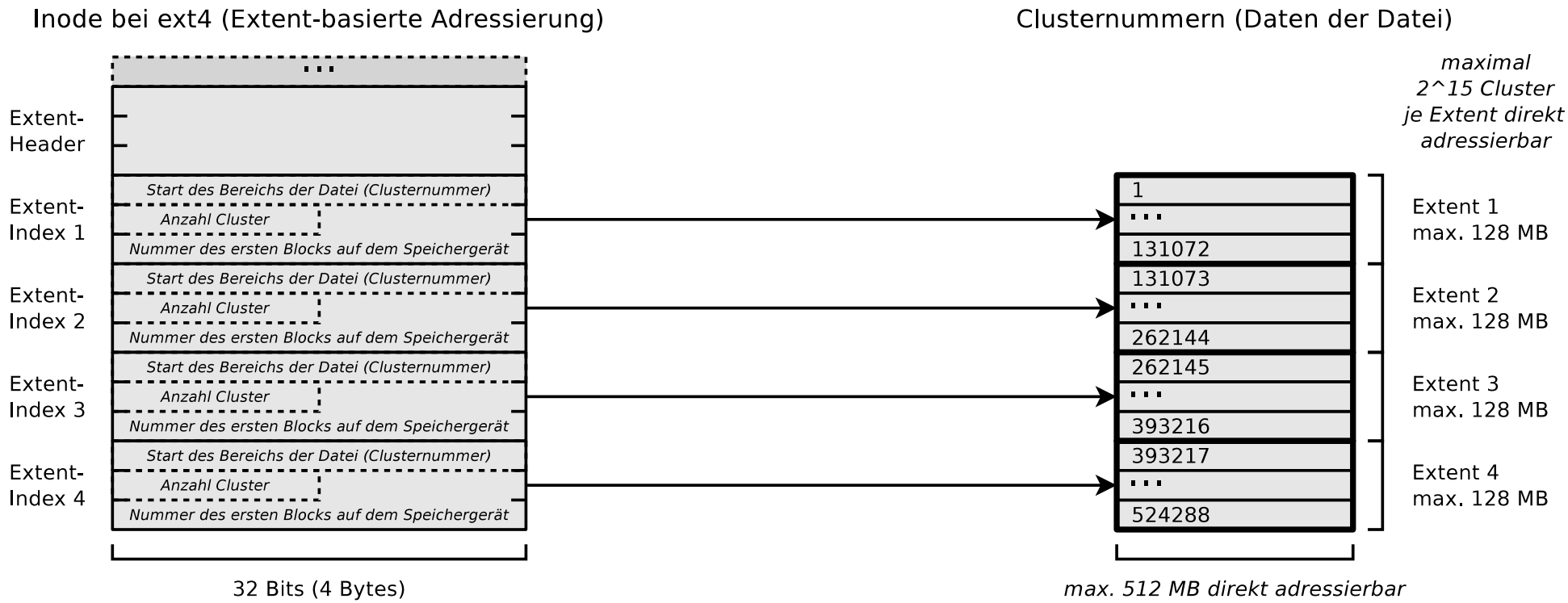
- Ergebnis: Weniger Verwaltungsaufwand
- Beispiele: JFS, XFS, btrfs, NTFS, ext4



Bildquelle: `http://www.heise.de/open/artikel/Extents-221268.html`

Extents am Beispiel von ext4

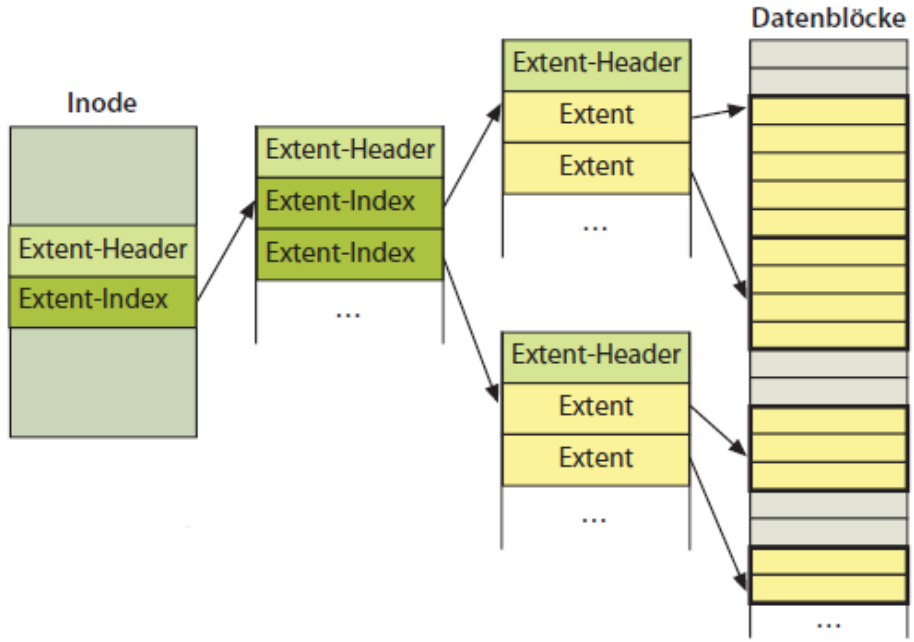
- Bei Blockadressierung mit ext3/4 sind in jedem Inode 15 je 4 Bytes große Felder, also 60 Bytes, zur Adressierung von Clustern verfügbar
- ext4 verwendet diese 60 Bytes für einen Extent-Header (12 Bytes) und zur Adressierung von 4 Extents (jeweils 12 Bytes)



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

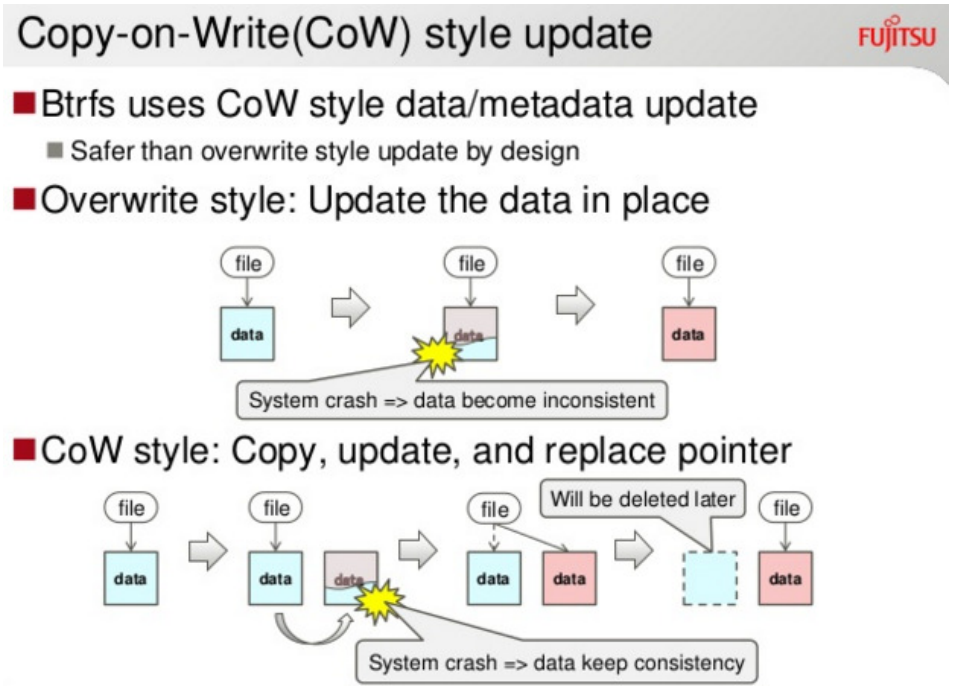


Bildquelle: `http://www.heise.de/open/artikel/Extents-221268.html`

Modernstes Konzept: Copy-On-Write

Bildquelle: Satoru Takeuchi (Fujitsu)

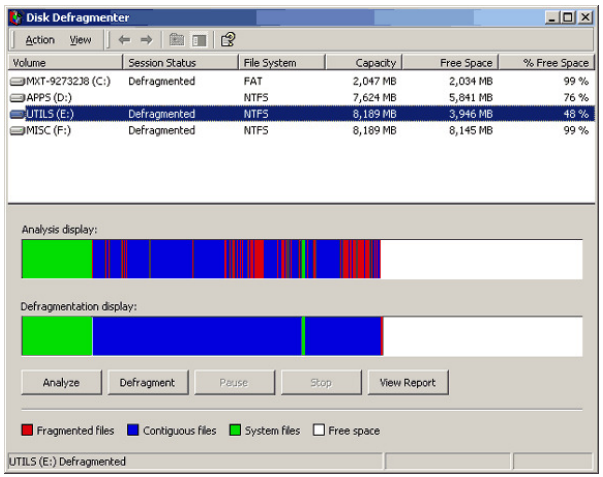
- Beim Schreibzugriff im Dateisystem wird nicht der Inhalt der Originaldatei geändert, sondern der veränderte Inhalt als neue Datei in freie Cluster geschrieben
- Anschließend werden die Metadaten auf die neue Datei angepasst



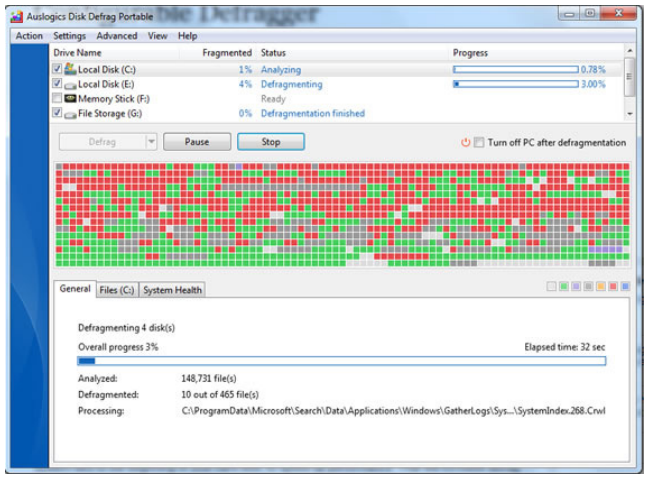
- Bis die Metadaten angepasst sind, bleibt die Originaldatei erhalten und kann nach einem Absturz weiter verwendet werden
- Vorteile:
  - Die Datensicherheit ist besser als bei Dateisystemen mit Journal
  - Snapshots können ohne Verzögerung erzeugt werden
- Beispiele: ZFS und btrfs

Fragmentierung

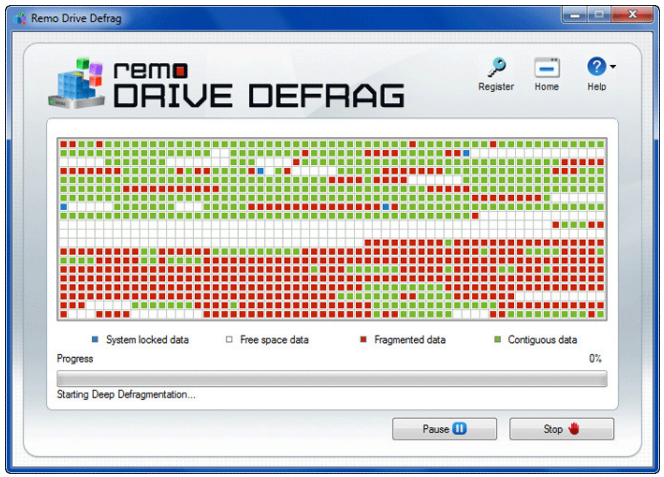
- In einem Cluster darf nur eine Datei gespeichert werden
  - Ist eine Datei größer als ein Cluster, wird sie auf mehrere verteilt
    - **Fragmentierung** beißt, dass logisch zusammengehörenden Cluster nicht räumlich beieinander sind
  - Ziel: Häufige Bewegungen des Schwungarme vermeiden
    - Liegen 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

- 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**
- 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

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

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