

Vertiefung Dateisysteme

Was haben wir bisher zum Thema Dateisysteme behandelt?

- Allgemeine Grundlagen Dateisysteme
- Grobe Unterschiede MBR und GPT
- Manuelles und automatisches Carven von Dateien
- Rudimentärer Aufbau von FAT- und NTFS-Dateisysteme

Aufbau MBR (Master-Boot-Record)

- Für diese Aufgabe dürfen Sie Tabellen verwenden → wenn diese Aufgabe in der Prüfung gestellt wird erhalten Sie diese
- Grundsätzlicher Aufbau nach festen Schema
- Von besonderem forensischen Interesse ist die Partitionstabelle und ggf. der Bootloader

Adresse		Funktion / Inhalt		Größe (Bytes)
hex	dez			
0x0000	0	Startprogramm (englisch <i>Bootloader</i>) (Programmcode)		440
0x01B8	440	Datenträgersignatur (seit Windows 2000)		4
0x01BC	444	Null (0x0000)		2
0x01BE	446	Partitionstabelle		64
0x01FE	510	55 _{hex}	Bootsektor-Signatur (wird vom BIOS für den ersten Bootloader geprüft)	2
0x01FF	511	AA _{hex}		
Gesamt:				512

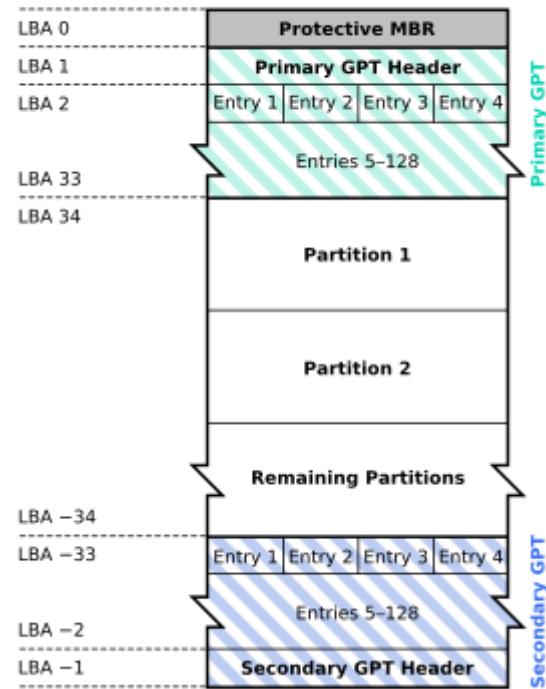
Bildquelle Wikipedia

MBR-Partitionseintrag

Speicherplatzadresse (Hexadezimal)	Größe (Byte)	Inhalt
0x00	1	bestimmt ob Partition gestartet werden kann oder nicht (80 _{hex} =bootfähig, 00 _{hex} =nicht bootfähig)
0x01	3	CHS-Eintrag des ersten Sektors
0x04	1	Typ der Partition (Partitionstyp)
0x05	3	CHS-Eintrag des letzten Sektors
0x08	4	Startsektor (relativ zum Anfang der Festplatte, oder zur erweiterten Partitionstabelle) per LBA-Methode
0x0C	4	Anzahl der Sektoren in der Partition per LBA-Methode

GPT(GUID Partition Table) Partitionsschema

GUID Partition Table Scheme



Bildquelle Wikipedia

GPT-Schema

Offset	Länge	Inhalt
0	8 bytes	Signatur („EFI PART“, 45h 46h 49h 20h 50h 41h 52h 54h)
8	4 bytes	Revision (00h 00h 01h 00h)
12	4 bytes	Header-Größe – Little Endian (5Ch 00h 00h 00h entspricht 92 bytes)
16	4 bytes	Header-CRC32-Prüfsumme (von Offset 0 bis Header-Größe, dieses Feld selbst wird bei der Berechnung auf 0 gesetzt)
20	4 bytes	Reservierter Bereich – muss Null (0) sein
24	8 bytes	Position des eigenen LBA (dieses Headers)
32	8 bytes	Position des Backup-LBA (des anderen Headers)
40	8 bytes	Erster benutzbare LBA für Partitionen (Letzter LBA der primären Partitionstabelle + 1, normalerweise 34)
48	8 bytes	Letzter benutzbare LBA (Erster LBA der sekundären Partitionstabelle – 1, normalerweise Datenträgergröße – 34)
56	16 bytes	Datenträger-GUID (als Referenz siehe auch UUID bei UNIXe)
72	8 bytes	Start-LBA der Partitionstabelle
80	4 bytes	Anzahl der Partitionseinträge (Partitionen)
84	4 bytes	Größe eines Partitionseintrags (normalerweise 128)
88	4 bytes	Partitionstabellen-CRC32-Prüfsumme
92	*	Reservierter Bereich; muss mit Nullen, für den Rest des Blocks, belegt sein (420 Bytes bei einem 512-byte LBA)

Bildquelle Wikipedia

GPT-Partitionseintrag

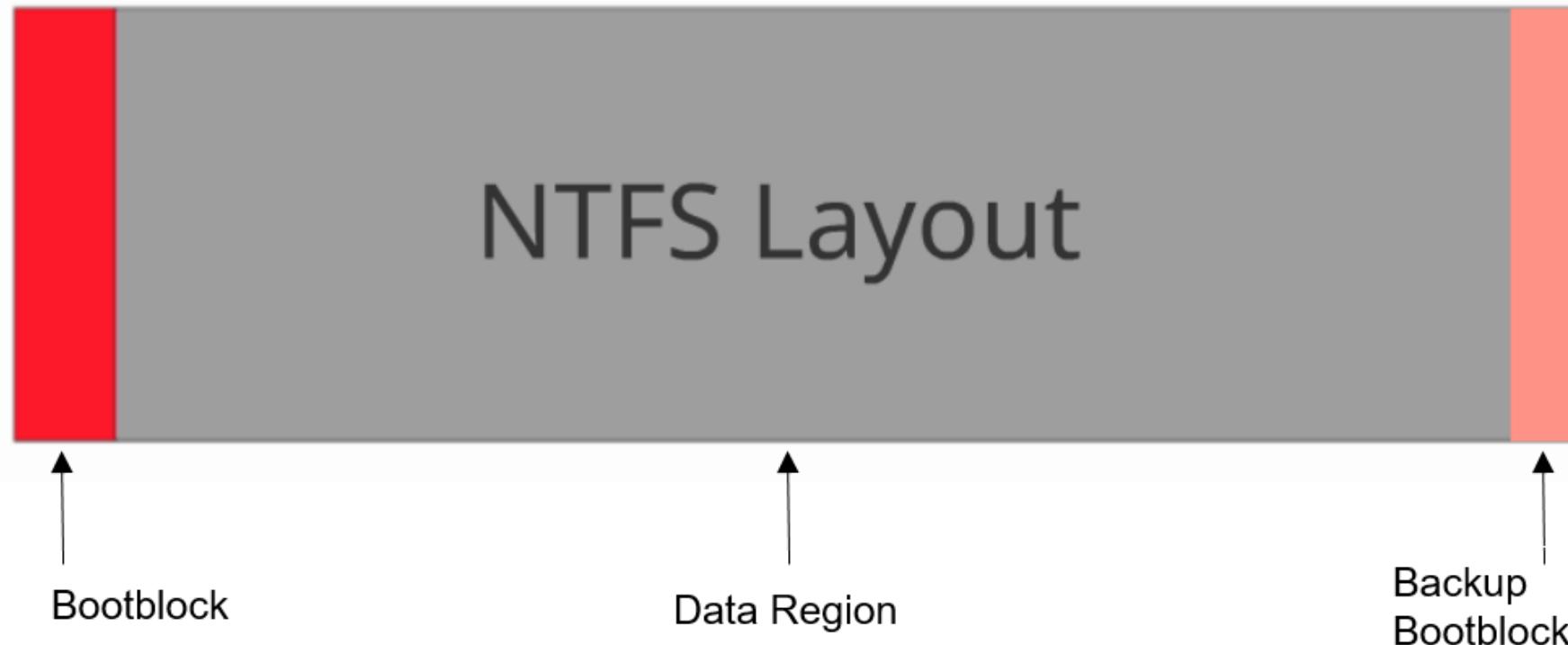
Offset	Länge	Inhalt
0	16 Bytes	Partitionstyp-GUID
16	16 Bytes	Eindeutige Partitions-GUID
32	8 Bytes	Beginn der Partition (erster LBA – Little-Endian)
40	8 Bytes	Ende der Partition (letzter LBA – inklusive)
48	8 Bytes	Attribute (siehe folgende Tabelle)
56	72 Bytes	Partitionsname (36 UTF-16LE -Zeichen)
insg.	128 Bytes	

Bildquelle Wikipedia

NTFS

- NTFS ist ein proprietäres Dateisystem von Microsoft, d.h. es existiert keine öffentliche Dokumentation, die den genauen Aufbau des Dateisystems beschreibt.
- Aus diesem Grund tun sich andere sehr schwer, Treiber für schreibenden Zugriff auf NTFS bereitzustellen.

NTFS



NTFS Bootblock

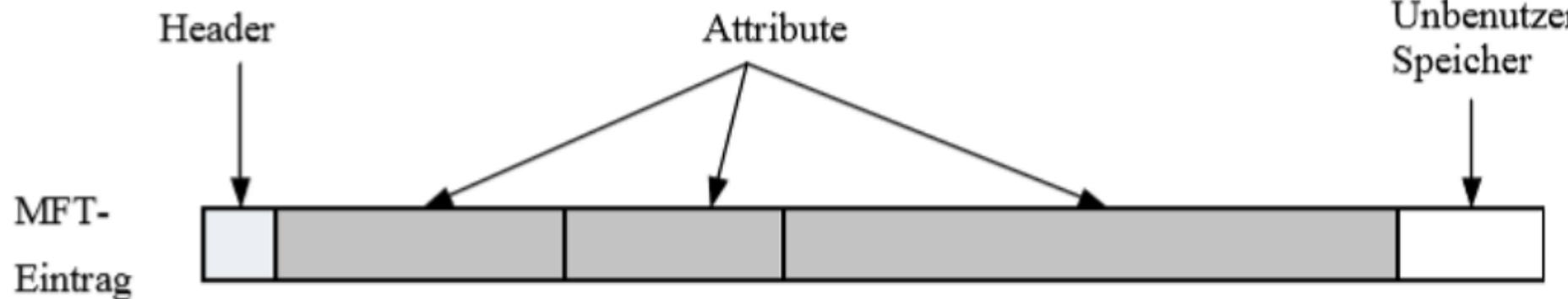
Offset	Länge	Beschreibung
0x000	3	EB 52 90 (Sprung zum Start des Bootprogramms + NOP)
0x003	8	System-ID: "NTFS"
0x00B	2	Anzahl Bytes pro Sektor
0x00D	1	Anzahl Sektoren pro Cluster
0x00E	7	reserviert (00 00 00 00 00 00 00)
0x015	1	Media Descriptor Byte (0xF8 für Hard Disk)
0x016	2	reserviert (00 00)
0x018	2	Anzahl Sektoren pro Spur
0x01A	2	Anzahl der Schreib-/Leseköpfe (Oberflächen)
0x01C	8	reserviert (00 00 00 00 00 00 00 00)
0x024	4	80 00 80 00 (immer?)
0x028	8	Gesamtzahl Sektoren im logischen Laufwerk
0x030	8	Logische Cluster-Nr. des ersten Clusters der MFT (Datenattribut von \$MFT)
0x038	8	Logische Cluster-Nr des ersten Clusters der Backup-MFT (\$MFTMirr)
0x040	4	Anzahl Cluster pro MFT-Record (0xF6 bedeutet 1/4) neg: $2^{\lfloor n \rfloor}$ Bytes
0x044	4	Anzahl Cluster pro Index-Record
0x048	8	Seriennummer des logischen Laufwerks
0x050	4	reserviert (00 00 00 00)
0x054	426	Boot-Code (boot loader routine)
0x1FE	2	Kennung für Bootsektoren (55 AA)

NTFS

Record-Nr

0	MFT (\$MFT)
1	Teilkopie der MFT (\$MFTMirr)
2	Log File (\$LogFile)
3	Volume File (\$Volume)
4	Attribute Definition File (\$AttrDef)
5	Root Directory (.)
6	Bitmap File (\$Bitmap)
7	Boot File (\$Boot)
8	BadCluster File (\$BadClus)
9	weitere NTFS Metadata Files
...	...
16	reserviert für <i>extension file records</i> der MFT
...	...
24	Anwender-Dateien und Directories

NTFS MFT



- **Jede Datei und jedes Verzeichnis hat mindestes einen Eintrag in der MFT, Default 1024 Bytes**
- **Die ersten 42 Bytes eines MFT-Eintrages bestehen aus 12 definierten Feldern, die restlichen 982 Bytes besitzen keine Struktur und können mit sog. Attributen aufgefüllt werden**
- **Attribute: Dateiname, Dateigröße, MAC, Freigabe, Dateityp, Dateiinhalt**
- **Bei sehr kleinen Dateien wird auch der Dateiinhalt in der MFT abgelegt (residente Attribute)**

Hummert 2017

NTFS Einträge

Offset	Länge	Beschreibung
0x00	4	“FILE”
0x04	2	Offset zur Update Sequenz
0x06	2	Größe der Update Sequenz
0x08	8	\$LogFile Sequenznummer
0x10	2	Sequenznummer
0x12	2	HardLink Counter
0x14	2	Offset des Ersten Attributs
0x16	2	Flags
0x18	4	Verwendete Größe des Dateieintrags
0x1C	4	Allozierte Größe des Dateieintrags
0x20	8	Referenz zum BaseFile Record
0x28	2	Next Attribt ID

NTFS Einträge

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	F	I	L	E	Update Seq array offset	Update Seq array size										\$LogFile Sequence Number
1	Seq no	Hard Link Count		1 st attrib offset	Flags		Used size of file record		Allocated size of file record							
2		File reference to base file record			Next attrib ID				MFT Record No							
3	default location of update seq array (size determined by seq size)				Reserved for update sequence array?											
		Reserved for sequence array?			Common location of 1 st attrib											

NTFS Attribut-Header Teil 1

OFFSET	Größe Bytes	Beschreibung
00	4	Attribut Identifizierungscode
04	4	Absolute Größe der Attribute
08	1	Kennzeichnung "Non-Resident" 00 – Resident 01 – Nichtresident
09	1	Länge des Namens
10	2	OFFSET zum Namen
12	2	Kennzeichnungen Flags 01 00 – komprimiert 00 40 – verschlüsselt 00 80 – sparse (wenig)
14	2	ID des Attributes

Attribut-Header Teil Resident

Residente Attribute (wird benutzt, wenn „Non-Resident“ Kennzeichen auf „00“ gesetzt ist)		
16	4	Länge des Attributes
20	2	Offset zum Start des "Attribut Stream"
22	1	Indiziertes Kennzeichen
23	1	Auffüllen
24	--	Name des Attributes
24	--	Start des Attributes

Attribut-Header, Nonresident

Nicht residente Attribute (wird benutzt, wenn „Non-Resident“ Kennzelchen auf „01“ gesetzt ist)		
16	8	Start VCN (Virtuelle Cluster Number)
24	8	Ende VCN
32	2	Offset zum Datarun
34	2	Größe der Kompressionseinheit
36	4	Auffüllung
40	8	Physikalische Größe des Attributes
48	8	Logische Größe des Attributes
56	8	Initialisierte Größe des Stromes (Stream)
64	--	Datarun (unbenannter Strom)
64	--	Name des Attributes (benannter Strom)

Attribut-Typen

ID	Attribute Typ	Beschreibung
0x10	Standard Information	Enthält unter anderem Zeitstempel und den Link-Count
0x20	Attribut List	Liste der Positionen aller Attribut Records, die nicht in den MFT Record passen
0x30	File Name	Ein Attribut für kurze und lange Dateinamen (oft ist das Attribut mehrfach vorhanden). Die langen Dateinamen sind bis zu 255 UnicodeZeichen lang. Die kurzen Dateinamen haben das 8+3 MSDOS Schema. Es können noch weitere File Name Attribute für zusätzliche Dateinamen oder Hard Links vorhanden sein.
0x40	Object ID	Ein partitionsweit eindeutiger Identifier. Wird vom Link Tracking Service verwendet. Nicht alle Dateien haben Object Identifier.
0x50	Security Descriptor	Rechteinformationen zur Datei
0x60	Volume Name	Wird nur im \$Volume System File verwendet. Enthält das Volume Label.
0x70	Volume Information	Wird nur im \$Volume System File verwendet. Enthält die Volume Version.
0x80	Data	Enthält die Daten der Datei. NTFS erlaubt mehrere Data Attribute pro Datei. Typischerweise enthält jede Datei ein unbenanntes Data Attribut – Weitere Data Attribute sind typischerweise benannt.
0x90	Index Root	Wird verwendet um Ordner und andere Indizes zu implementieren.
0xA0	Index Allocation	Wird verwendet um Ordner und andere Indizes zu implementieren.
0xB0	Bitmap	Wird verwendet um Ordner und andere Indizes zu implementieren.
0xC0	Reparse Point	Wird für Ordner Verzweigungen und Volume Mount Points verwendet. Werden darüberhinaus von manchen File System Filter Treibern verwendet.
0x100	Logged Tool Stream	Wird von EFS verwendet. Ähnlich zum Data Attribut, aber alle Änderungen werden in das NTFS Log aufgenommen.

Relevante NTFS Attribute

- **\$FILE_NAME (Identifier 48): Dateiname und Zeitstempel**
- **\$STANDARD_INFORMATION (16): Besitzer, weitere Zeitstempel, Zugriffsinformationen**
- **\$DATA (128): eigentlicher Inhalt (für Dateien)**
- **\$INDEX_ROOT (144): Basisknoten eines Indexbaums von Einträgen (für Verzeichnisse)**

Zeitstempel in NTFS

- **\$FILE_NAME, \$STANDARD_INFORMATION: MAC und MFT Modified Time**
- **64 Bit Anzahl von Hundert Nanosekunden seit dem 1. Januar 1601 UTC.**
- **1. Januar, 1970 UTC ist 116.444.736.000.000**

\$FILE_NAME

Offset	Size	Description
~	~	Standard Attribute Header
0x00	8	File reference to the parent directory.
0x08	8	C Time - File Creation
0x10	8	A Time - File Altered
0x18	8	M Time - MFT Changed
0x20	8	R Time - File Read
0x28	8	Allocated size of the file
0x30	8	Real size of the file
0x38	4	Flags, e.g. Directory, compressed, hidden
0x3c	4	Used by EAs and Reparse
0x40	1	Filename length in characters (L)
0x41	1	Filename namespace
0x42	2L	File name in Unicode (not null terminated)

\$STANDARD_INFORMATION

Offset	Size	OS	Description
~	~		Standard Attribute Header
0x00	8		C Time - File Creation
0x08	8		A Time - File Altered
0x10	8		M Time - MFT Changed
0x18	8		R Time - File Read
0x20	4		DOS File Permissions
0x24	4		Maximum Number of Versions
0x28	4		Version Number
0x2C	4		Class Id
0x30	4	2K	Owner Id
0x34	4	2K	Security Id
0x38	8	2K	Quota Charged
0x40	8	2K	Update Sequence Number (USN)

\$STANDARD_INFORMATION

Flag	Description
0x0001	Read-Only
0x0002	Hidden
0x0004	System
0x0020	Archive
0x0040	Device
0x0080	Normal
0x0100	Temporary
0x0200	Sparse File
0x0400	Reparse Point
0x0800	Compressed
0x1000	Offline
0x2000	Not Content Indexed
0x4000	Encrypted

\$ATTRIBUTE_LIST

Offset	Size	Description
~	~	Standard Attribute Header
0x00	4	Type
0x04	2	Record length
0x06	1	Name length (N)
0x07	1	Offset to Name (a)
0x08	8	Starting VCN (b)
0x10	8	Base File Reference of the attribute
0x18	2	Attribute Id (c)
0x1A	2N	Name in Unicode (if N > 0)

Reservierte Dateien in NTFS

\$MFT Record Nummer	Dateiname	Name	Beschreibung
0	\$Mft	Master File Table	Enthält einen Inode für jede Datei und jeden Ordner auf einem NTFS Volume.
1	\$MftMirr	MFT Mirror	1:1 Kopie der MFT
2	\$LogFile	Log File	Enthält Informationen zur schnellen Dateiwiederherstellung. Die Größe variiert.
3	\$Volume	Volume	Informationen über das Volume, wie Volumelabel und Volume-Größe
4	\$AttrDef	Attribute Definitions	Liste aller Attributnamen, Typnummern und Beschreibungen
5	.	Root File Name Index	Das Root Directory
6	\$Bitmap	Cluster Bitmap	Zeigt die freien und unbenutzten Cluster des Volumes
7	\$Boot	Boot Sector	Enthält den Bootblock
8	\$BadClus	Bad Cluster File	Liste der beschädigten Cluster des Volumes
9	\$Secure	Security File	Enthält eindeutige Security Descriptoren für alle Dateien im Volume
10	\$Upcase	Upcase Table	Zuordnung Groß- Kleinbuchstaben in Unicode
11	\$Extend	NTFS Extension File	Enthält optionale Erweiterungen wie Quotas, Reparse Points und Object Identifier
12-15			Reserviert für zukünftige Anwendungen

Reservierte Dateien in NTFS

Merke:

Auch reservierte Dateien haben das Attribut \$STANDARD_INFORMATION. Daraus lässt sich möglicherweise ableiten, wann das Dateisystem erstellt wurde.

Reservierte Dateien in NTFS

\$MFTMirr

- Kopie der \$MFT und steht in der MFT am Index 1
- allokiert das \$DATA Attribut grundsätzlich immer in der Mitte
- des Dateisystems
- Sollten die ersten Sektoren nicht lesbar sein, so kann nach der MFTMirr gesucht werden

Reservierte Dateien in NTFS

\$Boot

- enthält die Daten des Bootsektors
- Diese Datei befindet sich in der MFT am Index 7
- einzige Datei mit einem statischen Layout
- erster Sektor des \$DATA Attributs ist gleichzeitig immer der erste Sektor auf dem Laufwerk
- Kopie befindet sich am Ende des Dateisystems

Reservierte Dateien in NTFS

\$Volume

- MFT Index 3
- enthält den Laufwerknamen im VOLUME_NAME Attribut
- NTFS Version befindet sich im \$VOLUME_INFORMATION Attribut
- Beide Attribute tauchen nur im \$Volume-Eintrag auf und sollten nirgendwo anders zu finden sein

Reservierte Dateien in NTFS

\$AttrDef

- MFT Index 4
- Namen und Type Identifier aller Attribute

Reservierte Dateien in NTFS

\$Bitmap

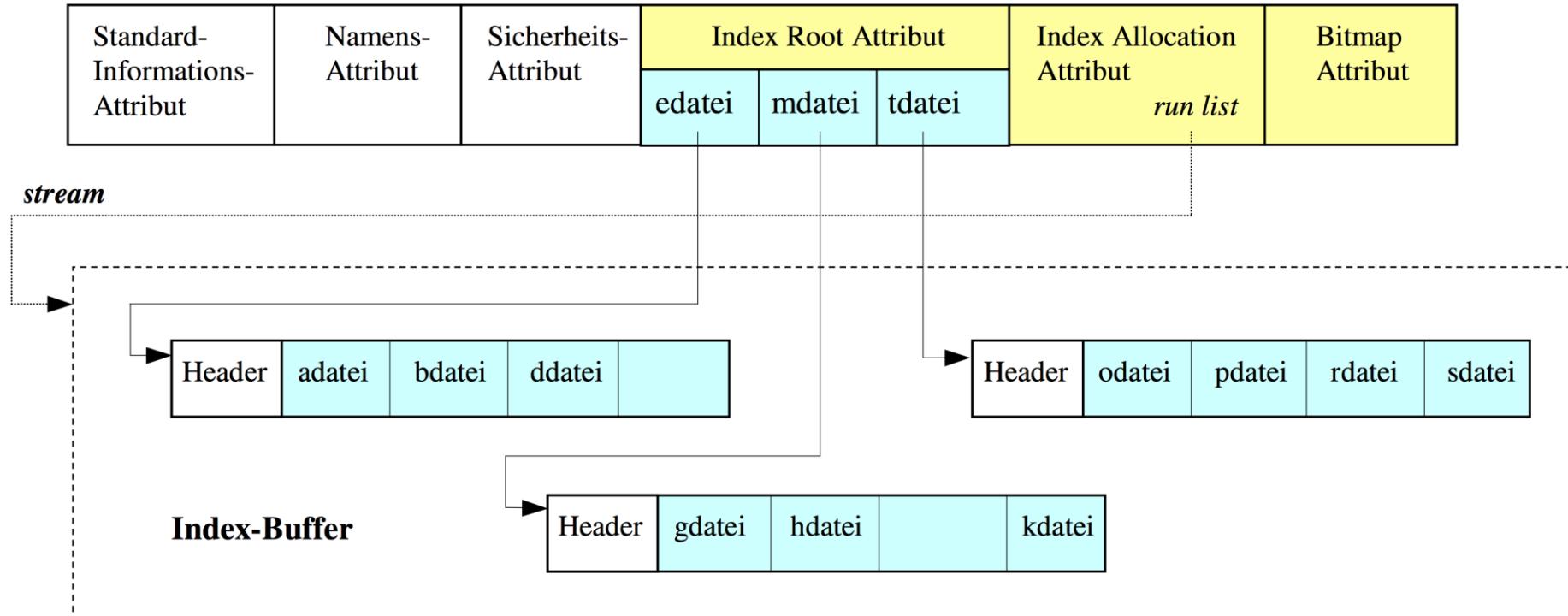
- MFT Index 6
- Belegstatus der Cluster
- Größe des \$DATA Attributs, ohne Header, ist gleich die Anzahl der Cluster in Bit

Reservierte Dateien in NTFS

\$BadClust

- MFT Index 8
- beschädigte Cluster auf dem Laufwerk
- Attribut dieser Datei hat den Namen “Bad”
- gesetztes Bit bedeutet, der Cluster kann nicht allokiert werden

Verzeichnisse in NTFS



Löschen eines MFT-Eintrags

- Header und \$Bitmap werden verändert
- Zeitstempel ggf. vorhanden
- Die Clusterketten lassen sich ggf. wiederfinden
- Kleine Dateien (\$DATA resident) findet man komplett in MFT-Eintrag

Erstellen einer Datei

1. Boot-Sektor lesen, Beginn der MFT finden, Größe eines MFT-Eintrages lesen
2. \$MFT nach unbelegtem Eintrag durchsuchen (dafür \$BITMAP anschauen)
3. Eintrag in \$LOGFILE vornehmen
4. Eintrag belegen in \$BITMAP, Eintrag in MFT ausnullen
5. Attribute \$STANDARD_INFORMATION, \$FILE_NAME erzeugen und Zeitstempel setzen
6. Neue Cluster in \$Bitmap finden und belegen
7. Dateinhalt in die frisch belegten Cluster schreiben
8. Indexstruktur in \$Root absuchen nach Eintrag für VERZEICHNIS
9. Dort neuen Eintrag für DATEINAME anlegen, Baum ggf. reorganisieren
10. Zeitstempel des Verzeichnisses aktualisieren

Löschen einer Datei

1. Boot-Sektor lesen, MFT finden, Größe eines MFT-Eintrags lesen
2. \$MFT auslesen, um das Layout des Dateisystems zu bestimmen
3. \$Root analysieren, Eintrag für VERZEICHNIS im Index finden, Zeitstempel (Zugriff) aktualisieren
4. Eintrag in \$LOGFILE vornehmen
5. In VERZEICHNIS den Eintrag für DATEI im Index finden
6. Eintrag aus Index entfernen, Baum ggf. reorganisieren
7. MFT-Eintrag von DATEI deallokieren, d.h. jetzt freie Cluster in \$Bitmap als unbelegt markieren
8. Nicht-residente Attribute von DATEI deallokieren

Journaling

- **\$LogFile ist in der MFT am Index 2**
- **Schreibt alle Veränderungen mit**
- **Aktionen werden protokolliert, bevor sie ausgeführt werden**
- **Sicherstellen, dass unterbrochene oder nur teilweise abgeschlossene Transaktionen entweder abgeschlossen oder zurück abgewickelt werden können**
- **Bei NTFS als Logging bezeichnet, sonst Journaling**
- **Jede Modifikation am Dateisystem durch einen I/O-Prozess wird protokolliert**
- **Die Logdaten werden ganz normal im \$DATA-Attribut gespeichert.**

\$LogFile

- \$LogFile teilt sich in Pages, die 4096 Bytes groß sind
- Es gibt zwei generelle Bereiche, Restart und Logging

Restart	Logging	
2 Pages	Buffer Area	General Area
	2 Pages	→ EOF

\$LogFile: Restart

- **Restart-Bereich enthält zwei Kopien einer Datenstruktur, die dem Betriebssystem dabei zu unterscheiden hilft, welche Transaktion durchgeführt werden soll**
- **Pointer, welcher auf den Logging Bereich zu der Stelle zeigt, an dem die letzte erfolgreiche Transaktion zu einer Datei durchgeführt wurde**

\$LogFile: Logging

- Der Logging-Bereich enthält eine Serie an Einträgen, mit einer logischen Sequenz Nummer, kurz LSN. Diese ist ein eindeutiger 64 Bit Wert.
- Die Einträge im Logging-Bereich sind in aufsteigender Reihenfolge angeordnet
- Es ist ein Ringspeicher, d.h. ist der Speicher voll, wird der älteste Eintrag überschrieben
- Es kann also sein, dass der Eintrag am Anfang des Loggingbereichs eine größere LSN hat, als der letzte Eintrag
- Entscheidend ist also WANN ein Logeintrag entstanden ist und nicht WO.

Page Header

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F							
0x00	"RCRD" (signature)			Update Sequence Offset	Update Sequence Count	Last LSN or File Offset																	
0x10	Flags			Page Count	Page Position	Next Record Offset	Word Align	DWord Align															
0x20	Last End LSN																						
0x30	Update Sequence Array																						

Page Count:

Zahl der Pages die für diese Transaktion belegt werden

Page Position:

Aktuelle Pagenummer der Transaktion

Next Record Offset:

Offset der letzten LSN auf der Page

Last LSN: Letzte LSN auf der Page

Last End LSN: Letzte komplette LSN auf der Page.

LSN Record Header

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F						
0x00	Current LSN										Previous LSN											
0x10	Client Undo LSN										Client Data Length		Client ID									
0x20	Record Type			Transaction ID				Flags	Alignment or Reserved													
0x30	Redo OP	Undo OP	Redo Offset	Redo Length	Undo Offset	Undo Length	Target Attribute	LCNs to Follow														
0x40	Record Offset	Attribute Offset	MFT Cluster Index	Alignment or Reserved	Target VCN				Alignment or Reserved													
0x50	Target LCN			Alignment or Reserved																		

LSN Record Header

Current LSN:	LSN des aktuellen Records
Previous LSN:	LSN des vorhergehenden Records
Client Undo LSN:	Üblicherweise die gleiche wie die Previous LSN
Client Data Length:	Länge des LSN Records
Record Type:	0x01 ist ein normaler Record, 0x02 ist ein Check Point Record
Flags:	0x00 Record reicht in die nächste Page, 0x01 Record hat keine Überlappung
Redo Op:	Redo Operation Code
Undo Op:	Undo Operation Code
Redo Offset:	Offset an dem die Redo Daten beginnen (ab Redo Op Offset)
Redo Length:	Länge der Redo Daten

LSN Record Header

Undo Offset: Offset an dem die Undo Daten beginnen
(ab Undo Op Offset)

Undo Length: Länge der Undo Daten

LCNs to Follow: 0x01 LCNs folgen dem LSN Header,
0x00 keine LCNs nach dem

LSN Header

Record Offset: Der MFT Record Offset, falls Transaktion
MFT berührt, sonst 0x00

Attribute Offset: Der Offset der Attribute die im MFT Record
berührt werden

Target LCN: Cluster Nummer der Redo/Undo Daten
auf dem Datenträger

Redo Undo Opcodes

Noop	0x00	DeleteIndexEntryAllocation	0x0F
CompensationLogRecord	0x01	SetIndexEntryVcnAllocation	0x12
InitializeFileRecordsSegment	0x02	UpdateFileNameRoot	0x13
DeallocateFileRecordSegment	0x03	UpdateFileNameAllocation	0x14
WriteEndOfFileRecordSegment	0x04	SetBitsInNonresidentBitMap	0x15
CreateAttribute	0x05	ClearBitsInNonresidentBitMap	0x16
DeleteAttribute	0x06	PrepareTransaction	0x19
UpdateResidentValue	0x07	CommitTransaction	0x1A
UpdateNonresidentValue	0x08	ForgetTransaction	0x1B
UpdateMappingParis	0x09	OpenNonresidentAttribute	0x1C
DeleteDirtyClusters	0x0A	DirtyPageTableDump	0x1F
SetNewAttributeSizes	0x0B	TransactionTable Dump	0x20
AddIndexEntryRoot	0x0C	UpdateRecordDataRoot	0x21
DeleteIndexEntryRoot	0x0D		
AddIndexEntryAllocation	0x0E		

Anlegen einer Datei

record_lsn	record_lsn_previous	redo_op	redo_op_s	undo_op	undo_op_s	event_number	lsn_number	logfile_offset
2110961	0	1500	SetBitsInNonresidentBitMap	1600	ClearBitsInNonresidentBitMap	26	508	110472
2110984	2110961	0000	Noop	0300	DeallocateFileRecordSegment	26	509	110656
2110995	2110984	0e00	AddIndexEntryAllocation	0f00	DeleteIndexEntryAllocation	26	510	110744
2111020	2110995	0e00	AddIndexEntryAllocation	0f00	DeleteIndexEntryAllocation	26	511	110944
2111045	2111020	0200	InitializeFileRecordSegment	0000	Noop	26	512	111144
2111109	2111045	1b00	ForgetTransaction	0100	CompensationLogRecord	26	513	111656

InitializeFileRecordSegment Transaction at 0x1B228

```
0001b220 54 00 00 00 00 00 00 00 45 36 20 00 00 00 00 00 T.....E6 .....
0001b230 2c 36 20 00 00 00 00 00 2c 36 20 00 00 00 00 00 ,6 .....6 .....
0001b240 d0 01 00 00 00 00 00 00 01 00 00 00 18 00 00 00 D.....
0001b250 00 00 00 00 00 00 00 00 02 00 00 00 29 00 a8 01 .....(,.
0001b260 d0 01 00 00 18 00 01 00 00 00 00 06 00 02 00 D.....
0001b270 08 00 00 00 00 00 00 00 5d 54 01 00 00 00 00 00 .....]T.....
0001b280 46 49 4c 45 | 30 00 03 00 08 36 20 00 00 00 00 FILE0....6 .....
0001b290 01 00 02 00 38 00 01 00 a8 01 00 00 00 04 00 00 ...8...."...
0001b2a0 00 00 00 00 00 00 00 00 04 00 00 00 23 00 00 00 .....#...
0001b2b0 01 00 00 00 00 00 00 00 10 00 00 00 60 00 00 00 .....`...
0001b2c0 00 00 00 00 00 00 00 00 48 00 00 00 18 00 00 00 .....H.....
0001b2d0 f9 08 95 1f be 79 ce 01 f9 08 95 1f be 79 ce 01 ü..äyí.ü..äyí.
0001b2e0 f9 08 95 1f be 79 ce 01 f9 08 95 1f be 79 ce 01 ü..äyí.ü..äyí.
```

AddIndexEntryAllocation Transaction at 0x1B098

```
0001b090 5d 54 01 00 00 00 00 00 | 13 36 20 00 00 00 00 00 ]T.....6 .....
0001b0a0 08 36 20 00 00 00 00 00 08 36 20 00 00 00 00 00 .6 .....6 .....
0001b0b0 98 00 00 00 00 00 00 00 01 00 00 00 18 00 00 00 .....
0001b0c0 00 00 00 00 00 00 00 00 00 00 0e 00 0f 00 28 00 70 00 .....(,.
0001b0d0 98 00 00 00 44 00 01 00 00 00 f0 04 00 00 08 00 .....D.....8....
0001b0e0 00 00 00 00 00 00 00 00 2c 00 00 00 00 00 00 00 00 .....,
0001b0f0 23 00 00 00 00 00 01 00 70 00 5e 00 00 00 00 00 #.....p.^.....
0001b100 05 00 00 00 00 00 05 00 f9 08 95 1f be 79 ce 01 .....ü..äyí.
0001b110 f9 08 95 1f be 79 ce 01 f9 08 95 1f be 79 ce 01 ü..äyí.ü..äyí.
0001b120 f9 08 95 1f be 79 ce 01 00 00 00 00 00 00 00 00 ü..äyí.....
0001b130 00 00 00 00 00 00 00 00 20 00 00 00 00 00 00 00 .....
0001b140 0e 01 74 00 65 00 73 00 74 00 66 00 69 00 6c 00 ...t.e.s.t.f.i.l.
0001b150 65 00 30 00 31 00 2e 00 74 00 78 00 74 00 00 00 e.0.1...tx.t...
```

Umbenennen einer Datei

record_lsn	record_lsn_previous	redo_op	redo_op_s	undo_op	undo_op_s	event_number	lsn_number	logfile_offset
2112777	0	0f00	DeleteIndexEntryAllocation	0000	AddIndexEntryAllocation	44	565	125000
2112802	2112777	0600	DeleteAttribute	0500	CreateAttribute	44	566	125200
2112828	2112802	0600	DeleteAttribute	0500	CreateAttribute	44	567	125408
2112854	2112828	0f00	DeleteIndexEntryAllocation	0000	AddIndexEntryAllocation	44	568	125616
2112879	2112854	0500	CreateAttribute	0600	DeleteAttribute	44	569	125816
2112906	2112879	0000	AddIndexEntryAllocation	0f00	DeleteIndexEntryAllocation	44	570	126032
2112932	2112906	0500	CreateAttribute	0600	DeleteAttribute	44	571	126240
2112958	2112932	0000	AddIndexEntryAllocation	0f00	DeleteIndexEntryAllocation	44	572	126448
2112983	2112958	1b00	ForgetTransaction	0100	CompensationLogRecord	44	573	126648

0x1E910

0x1EB78

DeleteAttribute Operation at 0x1E910

```
0001e910 22 3d 20 00 00 00 00 00 09 3d 20 00 00 00 00 00 00 "= .....= .....  
0001e920 09 3d 20 00 00 00 00 00 a0 00 00 00 00 00 00 00 ..= .....  
0001e930 01 00 00 00 18 00 00 00 00 00 00 00 00 00 00 00 .....  
0001e940 06 00 05 00 28 00 00 00 28 00 78 00 18 00 01 00 ....(...(x...  
0001e950 10 01 00 00 06 00 02 00 08 00 00 00 00 00 00 00 .....  
0001e960 5d 54 01 00 00 00 00 00 30 00 00 00 78 00 00 00 ]T.....0...x...  
0001e970 00 00 00 00 00 00 02 00 5e 00 00 00 18 00 01 00 .....^.....  
0001e980 05 00 00 00 00 00 05 00 f9 08 95 1f be 79 ce 01 .....ù..³yí.  
0001e990 f9 08 95 1f be 79 ce 01 f9 08 95 1f be 79 ce 01 ù..³yí.ù..³yí.  
0001e9a0 f9 08 95 1f be 79 ce 01 00 00 00 00 00 00 00 00 .....ù..³yí.....  
0001e9b0 00 00 00 00 00 00 00 20 00 00 00 00 00 00 00 00 .....  
0001e9c0 0e 01 74 00 65 00 73 00 74 00 66 00 69 00 6c 00 ..t.e.s.t.f.i.l.  
0001e9d0 65 00 30 00 31 00 2e 00 74 00 78 00 74 00 00 00 e.0.l...t.x.t....
```

Create Attribute Operation at 0x1EB78

```
0001eb70 54 00 00 00 00 00 00 00 00 6f 3d 20 00 00 00 00 00 T.....o= .....  
0001eb80 56 3d 20 00 00 00 00 00 00 56 3d 20 00 00 00 00 00 V= .....V= .....  
0001eb90 a8 00 00 00 00 00 00 00 00 01 00 00 00 18 00 00 00 .....  
0001eba0 00 00 00 00 00 00 00 00 00 05 00 06 00 28 00 80 00 .....(.e.  
0001ebb0 a8 00 00 00 18 00 01 00 98 00 00 00 06 00 02 00 .....  
0001ebc0 08 00 00 00 00 00 00 00 00 5d 54 01 00 00 00 00 00 .....]T.....  
0001ebd0 30 00 00 00 80 00 00 00 00 00 00 00 00 00 05 00 00 0...e.....  
0001ebf0 62 00 00 00 18 00 01 00 05 00 00 00 00 00 00 05 00 00 b.....  
0001ebf0 f9 08 95 1f be 79 ce 01 55 44 4e 2e be 79 f4 06 ù..³yí.UDN.³yí.  
0001ec00 55 44 4e 2e be 79 ce 01 f9 08 95 1f be 79 ce 01 UDN.³yí.ù..³yí.  
0001ec10 40 00 00 00 00 00 00 00 00 3d 00 00 00 00 00 00 00 @.....=.....  
0001ec20 20 00 00 00 00 00 00 00 10 01 72 00 65 00 6e 00 .....r.e.n.  
0001ec30 61 00 6d 00 65 00 66 00 69 00 6c 00 65 00 30 00 a.m.e.f.i.l.e.0.  
0001ec40 31 00 2e 00 74 00 78 00 74 00 2e 2e 2e 00 00 00 1...t.x.t.....
```

Löschen einer Datei

record_lsn	record_lsn_previous	redo_op	redo_op_s	undo_op	undo_op_s	event_number	lsn_number	logfile_offset
2114014	0	0f00	DeleteIndexEntryAllocation	0e00	AddIndexEntryAllocation	50	593	134896
2114039	2114014	0f00	DeleteIndexEntryAllocation	0e00	AddIndexEntryAllocation	50	594	135096
2114073	2114039	0d00	DeleteIndexEntryRoot	0c00	AddIndexEntryRoot	50	595	135368
2114095	2114073	0300	DeallocateFileRecordSegment	0200	InitializeFileRecordSegment	50	596	135544
2114109	2114095	1600	ClearBitsInNonresidentBitMap	1500	SetBitsInNonresidentBitMap	50	597	135656
2114121	2114109	1b00	ForgetTransaction	0100	CompensationLogRecord	50	598	135752

DeallocateFileRecordSegment Operation at 0x21178

```
00021170 00 00 00 00 00 00 00 00 00 2f 42 20 00 00 00 00 00 ...../B .....
00021180 19 42 20 00 00 00 00 00 19 42 20 00 00 00 00 00 .B .....B .....
00021190 40 00 00 00 00 00 00 00 01 00 00 00 18 00 00 00 @.....
000211a0 00 00 00 00 00 00 00 00 03 00 02 00 28 00 00 00 .....(...
000211b0 28 00 18 00 18 00 01 00 00 00 00 00 06 00 02 00 (.....
000211c0 08 00 00 00 00 00 00 00 5d 54 01 00 00 00 00 00 ]T.....
000211d0 45 49 4c 45 30 00 03 00 23 3e 20 00 00 00 00 00 FILE0...#> .....
000211e0 01 00 02 00 38 00 01 00 3d 42 20 00 00 00 00 00 ....8...=B .....
```

DeleteIndexEntryAllocation Operation at 0x20FB8

```
00020fb0 54 00 00 00 00 00 00 00 f7 41 20 00 00 00 00 00 T.....+A .....
00020fc0 de 41 20 00 00 00 00 00 de 41 20 00 00 00 00 00 PA .....PA .....
00020fd0 a0 00 00 00 00 00 00 00 01 00 00 00 18 00 00 00 .....
00020fe0 01 00 00 00 00 00 00 00 0f 00 0e 00 28 00 00 00 .....(...
00020ff0 28 00 78 00 44 00 01 00 00 00 f0 04 00 00 f6 06 (.x.D....6...8.
00021000 52 43 52 44 28 00 09 00 e7 43 20 00 00 00 00 00 RCRD(...çC .....
00021010 01 00 00 00 08 00 01 00 f8 0f 00 00 00 00 00 00 .....ç.....
00021020 e7 43 20 00 00 00 00 00 f7 06 00 00 00 00 00 00 çC .....+.....
00021030 00 00 00 00 00 00 01 00 00 00 00 00 00 00 00 00 .....ç.....
00021040 00 00 00 00 00 00 00 00 2c 00 00 00 00 00 00 00 .....ç.....
00021050 23 00 00 00 00 00 01 00 78 00 62 00 00 00 00 00 #.....x.b....
00021060 05 00 00 00 00 00 05 00 f9 08 95 1f be 79 ce 01 .....ü..%yí.
00021070 55 44 4e 2e be 79 ce 01 5b 12 aa 3b be 79 ce 01 UDN.%yí.[.*;%yí.
00021080 f9 08 95 1f be 79 ce 01 40 00 00 00 00 00 00 00 ü..%yí.Ø.....
00021090 3d 00 00 00 00 00 00 00 20 00 00 00 00 00 00 00 =.....ç.....
000210a0 10 01 72 00 65 00 6e 00 61 00 6d 00 65 00 66 00 ..r.e.n.a.m.e.f.
000210b0 69 00 6c 00 65 00 30 00 31 00 2e 00 74 00 78 00 i.l.e.0.1...tx.
000210c0 74 00 00 00 00 00 00 00 19 42 20 00 00 00 00 00 .....B .....
```

NTFS

- **NTFS verfügt über zentrale Datenstrukturen, die in Dateien organisiert sind**
- **Fehlende offizielle Dokumentation macht eine genaue Analyse schwierig**
- **Vorteile gegenüber FAT, vor allem in der Dateiwiederherstellung**
- **Metadaten geben Aufschluss über eventuelle Tathergänge**
- **Journaling bei NTFS nur schwer zu fälschen**
- **Ist der Forensiker in der Lage dieses richtig auszuwerten, können damit alle Tätigkeiten im Dateisystem rekonstruiert werden**
- **Auch die MFT stellt eine wertvolle Quelle für Metainformationen dar**

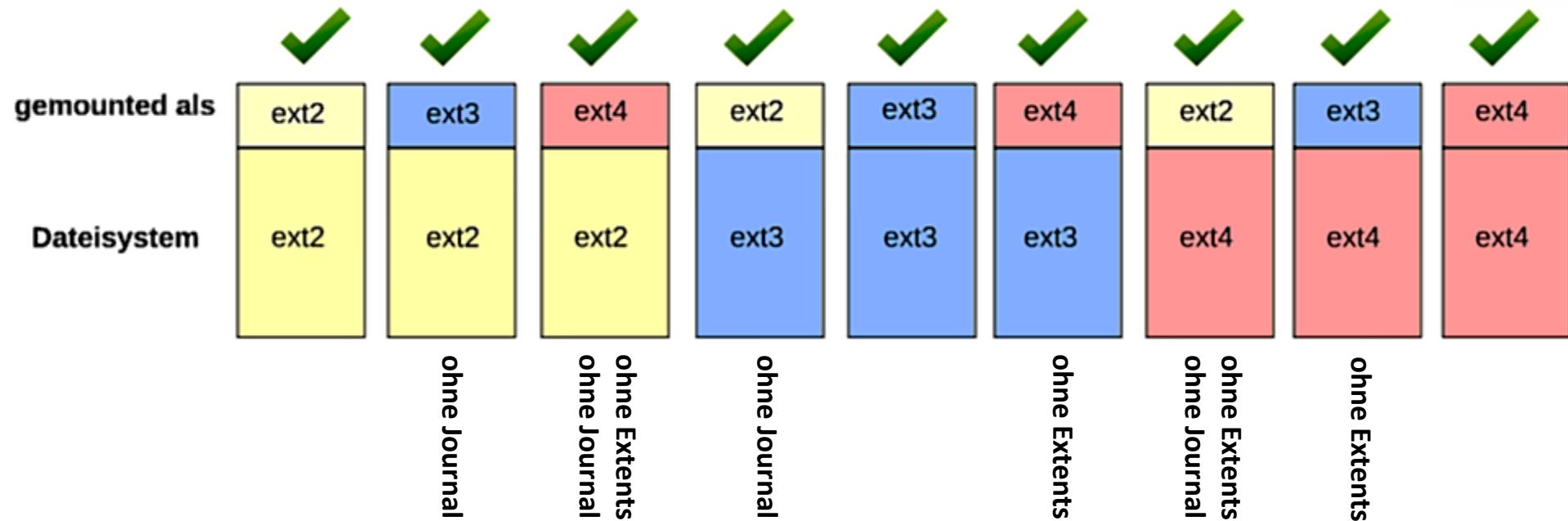
- Nachfolger des Unix File Systems, UFS
- Standard Dateisystem in vielen Linux Distributionen
- ext2, ext3 und ext4, haben die gleiche Grundstruktur
 - Little Endian
 - Die uns bekannten Cluster (kleinstmögliche Zuordnungseinheit des Dateisystems) heißen hier Blocks oder Blöcke

- **Geschwindigkeit und Zuverlässigkeit**
- **Kopien zentraler Datenstrukturen mehrfach auf dem Datenträger**
- **Datenblöcke einer Datei werden nah beieinander gehalten, um so die Wege des Lesekopfes zu minimieren**

ext

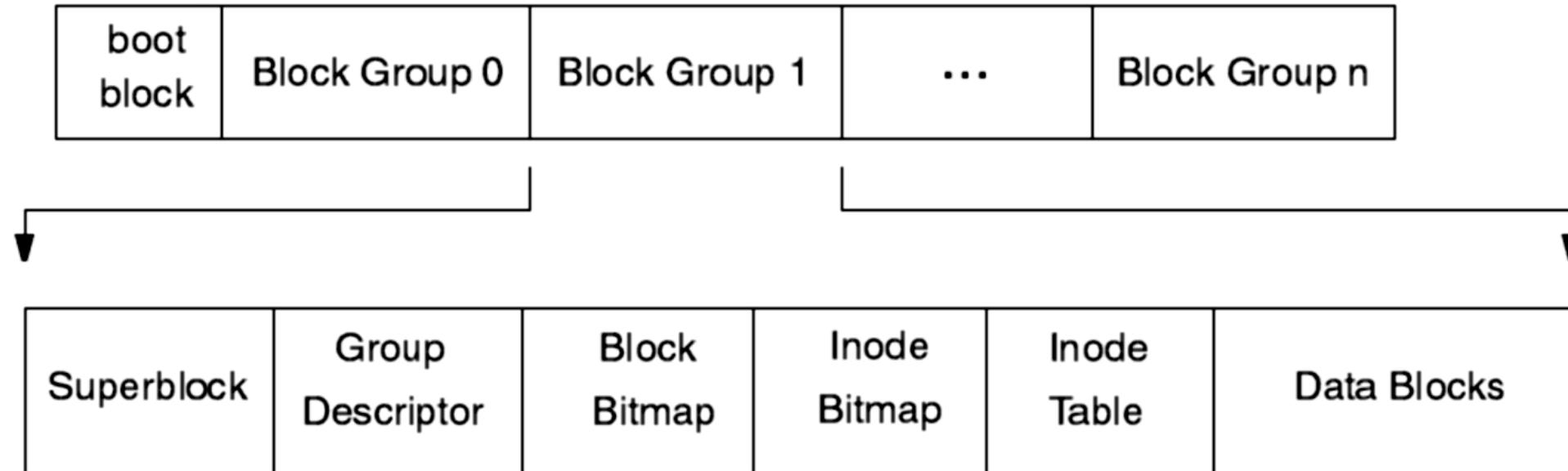
- **ext (1992)** Nachfolger von UFS und quasi sofort von ext2 abgelöst
- **ext3 (2001)** Journaling. Das Journal ist eine Dateistruktur, in die Metadaten (optional die Nutzdaten) geschrieben werden, bevor sie auf das tatsächliche Dateisystem geschrieben werden.
- **ext4 (2008)** Extents. Extents bringen Geschwindigkeitsvorteile bei der Verwaltung großer Dateien und beugen Fragmentierung vor
- **Btrfs (B-tree File System)** soll ext ablösen

ext





Bootblock ist immer 1024 Bytes groß und ohne Wert für die Analyse



Der Superblock enthält Angaben über Größen und Konfigurationen.

Der Superblock befindet sich immer am Anfang und hat eine Kopie am Anfang jeder Blockgruppe.

Superblock

Offset	Länge	Beschreibung
0x00	4	Gesamtgröße des Dateisystems in Inodes
0x04	4	Gesamtgröße des Dateisystems in Blöcken
0x08	4	Anzahl Reservierter Blöcke
0x0C	4	Anzahl freier Blöcke
0x10	4	Anzahl freier Inodes
0x14	4	Adresse des ersten Daten Blocks
0x18	4	Blockgröße (0=1k, 1=2k, 2=4k)
0x1C	4	Fragmentgröße (0=1k, 1=2k, 2=4k) (Teildatenblöcke)
0x20	4	Blocks pro Blockgruppe
0x24	4	Fragmente pro Blockgruppe
0x28	4	Inodes pro Blockgruppe
0x2C	4	Zeitpunkt der letzten Einbindung (Last Mount)
0x30	4	Zeitpunkt des letzten Schreibzugriffs (Last Write)
0x34	2	Anzahl der Einbindung (Mount Count)
0x36	2	Maximale Anzahl Mounts vor Dateisystemcheck
0x38	2	Magic Byte 0x 53 EF

Superblock

Offset	Länge	Beschreibung
0x3A	2	File System Status (i.O. oder mit Fehlern)
0x3C	2	Fehlerbehandlungsmethode
0x3E	2	Subversionnummer
0x40	4	Zeitstempel des letzten Filesystemchecks
0x44	4	Max. Zeit zwischen zwei Checks (in Sec)
0x48	4	Betriebssystem (0 = Linux)
0x4C	4	Versionsnummer
0x50	2	User ID für reservierte Blöcke
0x52	2	Group ID für reservierte Blöcke
Ab hier Extended Section		
0x54	4	Erster nicht reservierter Inode
0x58	4	Inode Größe
0x5A	2	ID der Blockgruppe dieses Superblocks

Group Descriptor

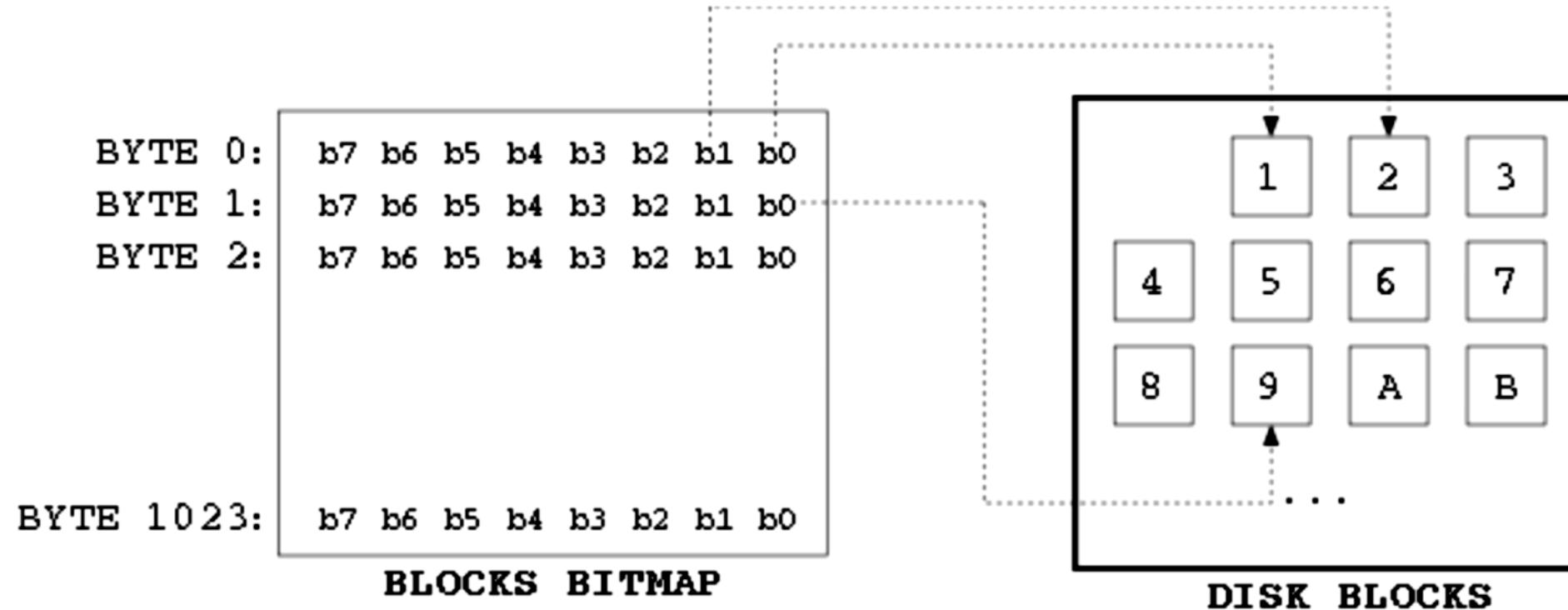
Der Group Descriptor enthält Informationen über das Layout der Blockgruppe. Der Group Descriptor ist immer im Block direkt hinter dem Superblock. Kopie in jeder Blockgruppe direkt im Block hinter der Kopie des Superblocks

Offset	Länge	Beschreibung
0x00	4	Blocknummer für Block Bitmap
0x04	4	Blocknummer für Inode Bitmap
0x08	4	Blocknummer für Inode Tabelle
0x0A	2	Anzahl der freien Blöcke
0x0C	2	Anzahl der freien Inodes
0x0E	2	Anzahl der Verzeichniseinträge
0x10	2	Padding
0x14	4	Reserviert

Block Bitmap

- Jede Blockgruppe hat ihre eigene Block Bitmap
- verwaltet den Allokationsstatus der Blöcke innerhalb der Blockgruppe
- Der Startblock der Bitmap ist im Gruppendeskriptor angegeben
- Beim Formatieren wird die Anzahl der Blöcke für jede Blockgruppe gleich mit der Anzahl der Bits in jedem Block definiert
- Block Bitmap ist genau einen Block groß ist
- Ist ein Block Beispielsweise 1 KiB groß, so ist die Blockgruppe auf $1024 \times 8 = 8192$ Blöcke beschränkt, da für jeden Block ein Bit in der Block Bitmap zu Verfügung steht muss

Block Bitmap



Inode Bitmap

- Inodes sind vergleichbar mit Dateieinträgen mit Metadaten
- Die Inode Bitmap verwaltet den Belegstatus von Inodes in der Blockgruppe
- Auch die Inode Bitmap ist immer genau einen Block groß
- Für jede Inode wird ebenfalls immer genau ein Bit benötigt.

Inode Table

- Die Inode Table besteht aus vielen aneinander gereihten Inodes, die alle exakt die gleiche Größe haben
- Diese ist im Superblock definiert
- Im Standard besitzt jede Blockgruppe 2048 Inodes
- Die Inode ist der Dreh- und Angelpunkt des Dateizugriffs, denn diese speichert die Metadaten einer Datei oder eines Verzeichnisses
- Inodes haben eine eindeutige Nummer beginnend bei 1.

Inode Table

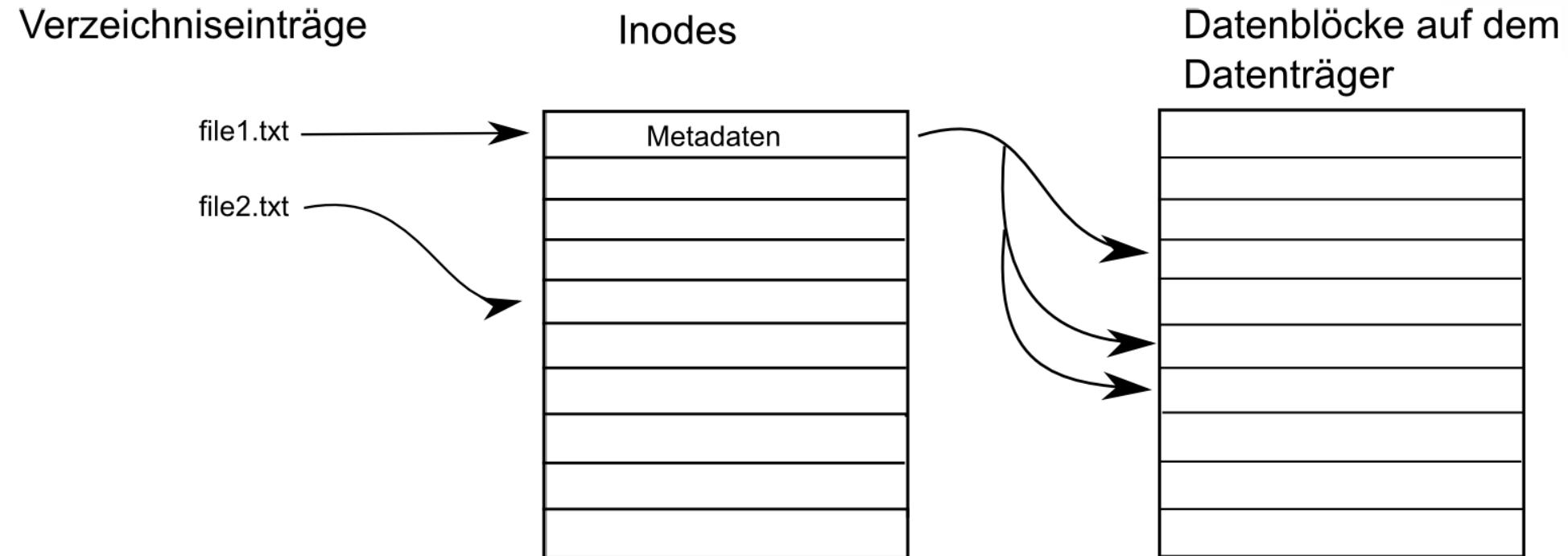
Inodes 1-10 sind reserviert

- **Inode 1: beschädigte Blöcke**
- **Inode 2: Wurzelverzeichnis**
- **Inode 8: Journal**
- **Inode 11: lost+found**

Inode Table

- Ein Inode speichert Metadaten zu einer Datei oder einem Verzeichnis und den oder die Blöcke in denen die Daten gespeichert sind.
- Die Inode Tabelle ist also so etwas wie ein Reiseführer für die Blockgruppe.
- Die Inodes sind die Sehenswürdigkeiten und geben zusätzliche Informationen sowie den genauen Standort an.
- Verzeichniseinträge enthalten Dateinamen und einen Verweis auf einen Inode
- Inodes selbst enthalten Verweise auf Datenblöcken auf dem Datenträger

Inodes



Zeitstempel in Inodes

- **letzter Zugriff**
- **letzte Veränderung**
- **Löschen einer Datei oder eines Verzeichnisses**
- **Epoch Time:**
 - **32 Bit**
 - **Sekunden seit dem 01.01.1970, 00:00 Uhr UTC**
 - **ohne Schaltsekunden**
 - **Wertebereich bis 19.01.2038 um 3:14:08 Uhr UTC**

Inodes

Offset	Länge	Beschreibung
0x00	2	Dateityp (Datei, Verzeichnis, Device, ...)
0x02	2	Niederwertige 16Bit der Owner UID
0x04	4	Dateigröße in Byte
0x08	4	Access Time
0x0C	4	Create Time
0x10	4	Modified Time
0x14	4	Deletion Time
0x18	2	Niederwertige 16Bit der Owner GID
0x1A	2	Link Counter
0x1C	4	Block Counter
0x20	4	Flags

Inodes

0x24	4	Betriebssystemabhängig
0x28	15x4	Zeiger auf die Blocks
0x7C	4	Dateiversion
0x80	4	Datei Rechte
0x84	4	Verzeichnis Rechte
0x88	4	Addresse des Fragments

betriebssystemabhängig Bytes

0x8C	1	Fragment ID
0x8D	1	Fragment Größe
0x8E	2	Padding
0x90	2	Höherwertige 16Bit der Owner UID
0x92	2	Höherwertige 16Bit der Owner G ID
0x96	4	Reserviert

Dateitypen

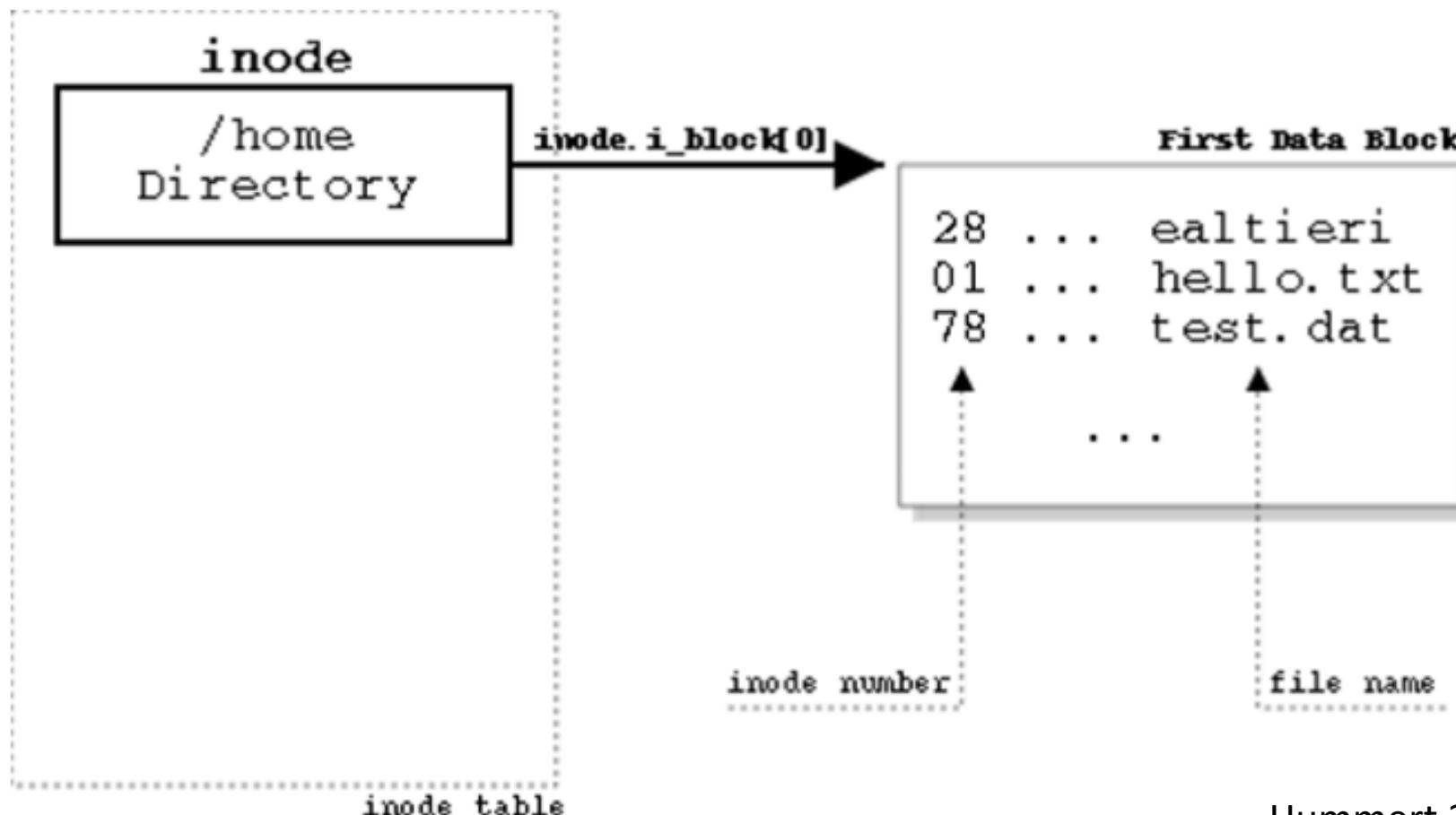
0xC000	socket
0xA000	symbolic link
0x8000	regular file
0x6000	block device
0x4000	directory
0x2000	character device
0x1000	fifo

Dateirechte

0x0100	user read
0x0080	user write
0x0040	user execute
0x0020	group read
0x0010	group write
0x0008	group execute
0x0004	others read
0x0002	others write
0x0001	others execute
0x0800	Set process User ID
0x0400	Set process Group ID
0x0200	sticky bit

Verzeichnisse

- Blockpointer zeigen auf einen Datenblock der eine Liste von Inode-Nummern enthält



Verzeichniseinträge

Offset	Länge	Beschreibung
0x00	4	Inode-Nummer
0x04	2	Länge des Verzeichniseintrags
0x06	1	Länge des Eintrag-Namens
0x07	1	Typ des Eintrags
0x08	var.	Name des Eintrags

Verzeichniseinträge: Typ

Type	Beschreibung
0	Unknown
1	Regular File
2	Directory
3	Character Device
4	Block Device
5	Named pipe
6	Socket
7	Symbolic Link

Verzeichnisse

- Die ersten beiden Einträge zeigen auf „.“ und „..“

	inode	rec_len	file_type	name_len	name		
0	13	12	1	2	.	\0	\0
12	2	12	2	2	.	\0	\0
24	18	16	5	2	m	u	s
40	15	16	8	1	t	e	s
56	19	12	3	2	b	i	n

Padding



..

music/

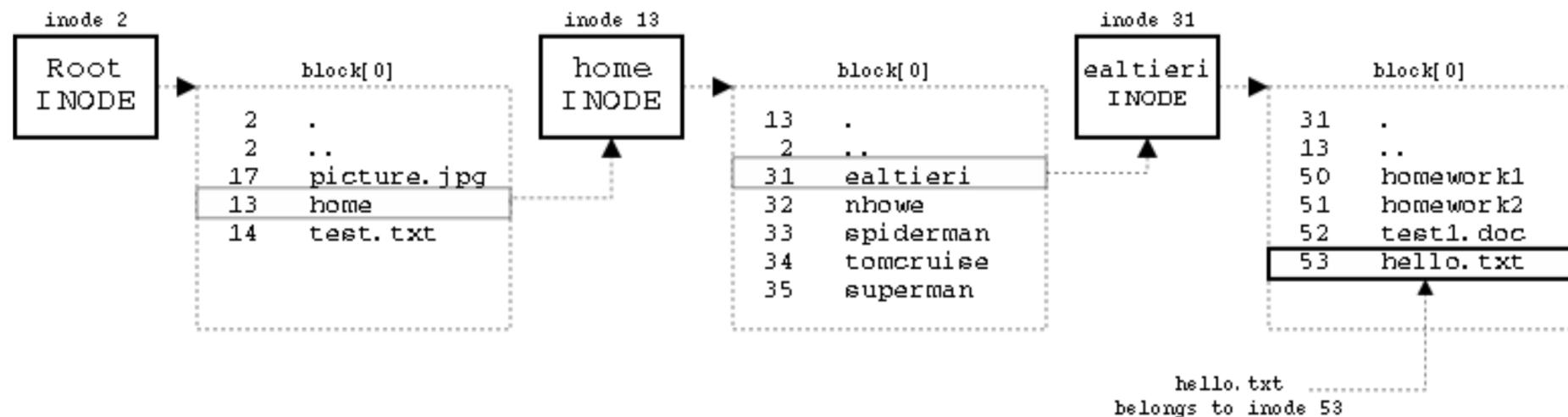
test.txt

bin/

Dateiaufruf



- Es wird immer im Root Directory begonnen und dann dem Pfad gefolgt, Inode für Inode



Mount Points

Verzeichnisse können

- **sowohl Dateien und Verzeichnisse beinhalten**
- **und gleichzeitig als Mount Point dienen**

Allerdings kann auf Dateien und Verzeichnisse in einem Verzeichnis, dass als Mount Point dient, solange nicht zugegriffen werden, wie in das Verzeichnis ein Dateisystem gemountet ist

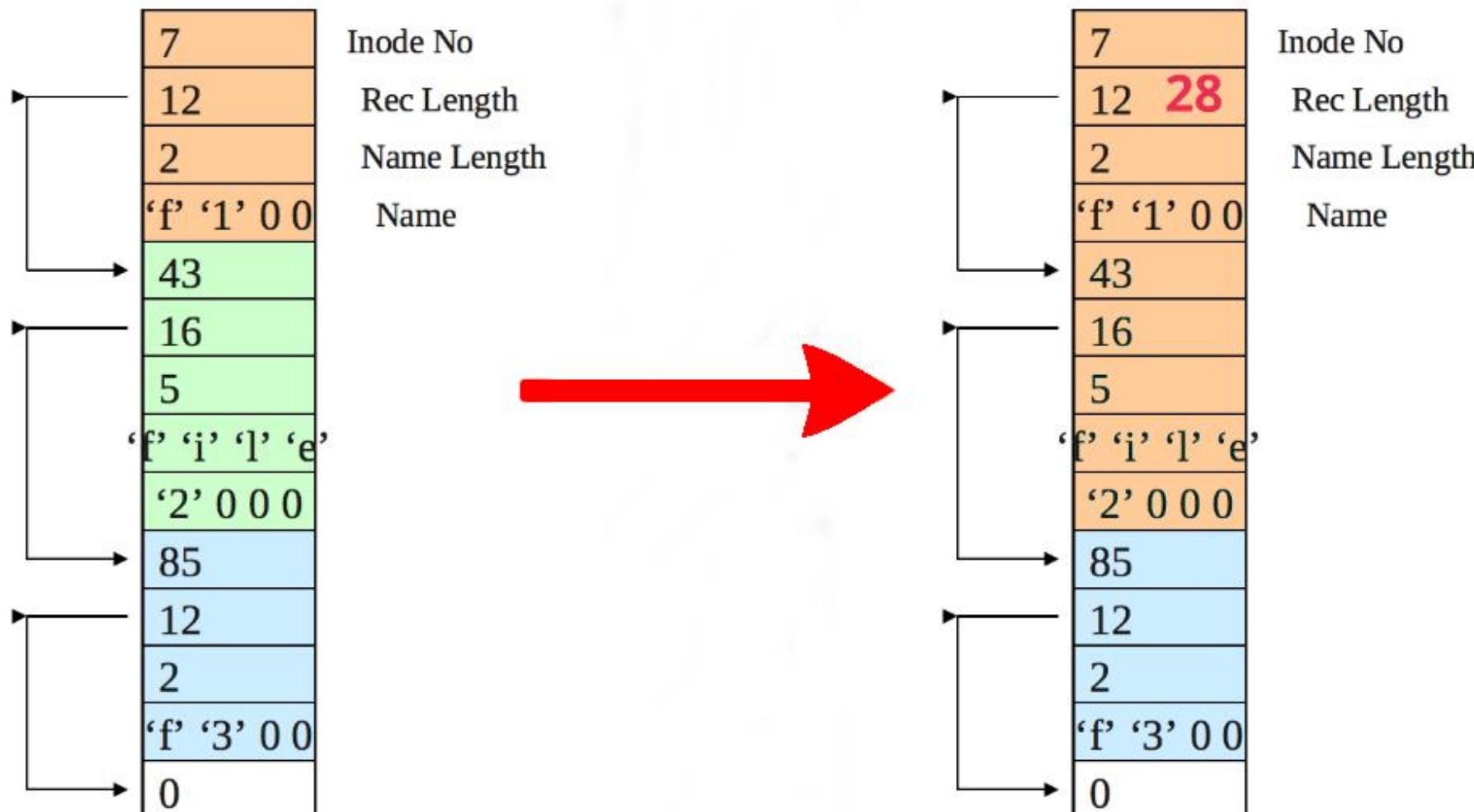
Datei erstellen

- Beginne in der Blockgruppe des Elternverzeichnisses und suche in der Inode Bitmap nach freiem Inode
- Sonst: suche eine andere Blockgruppe mit freiem Inode
- Trage Inode in Inode Bitmap und Inode Table ein
- Metadaten in Inode eintragen
- Schreibe Inhalte in die Data Blöcke

Datei löschen

- Inode in Inode Bitmap frei setzen
- Blöcke in Block Bitmap als frei markieren
- Vorhergehenden Verzeichniseintrag vergrößern

Datei löschen



Datei löschen ext3

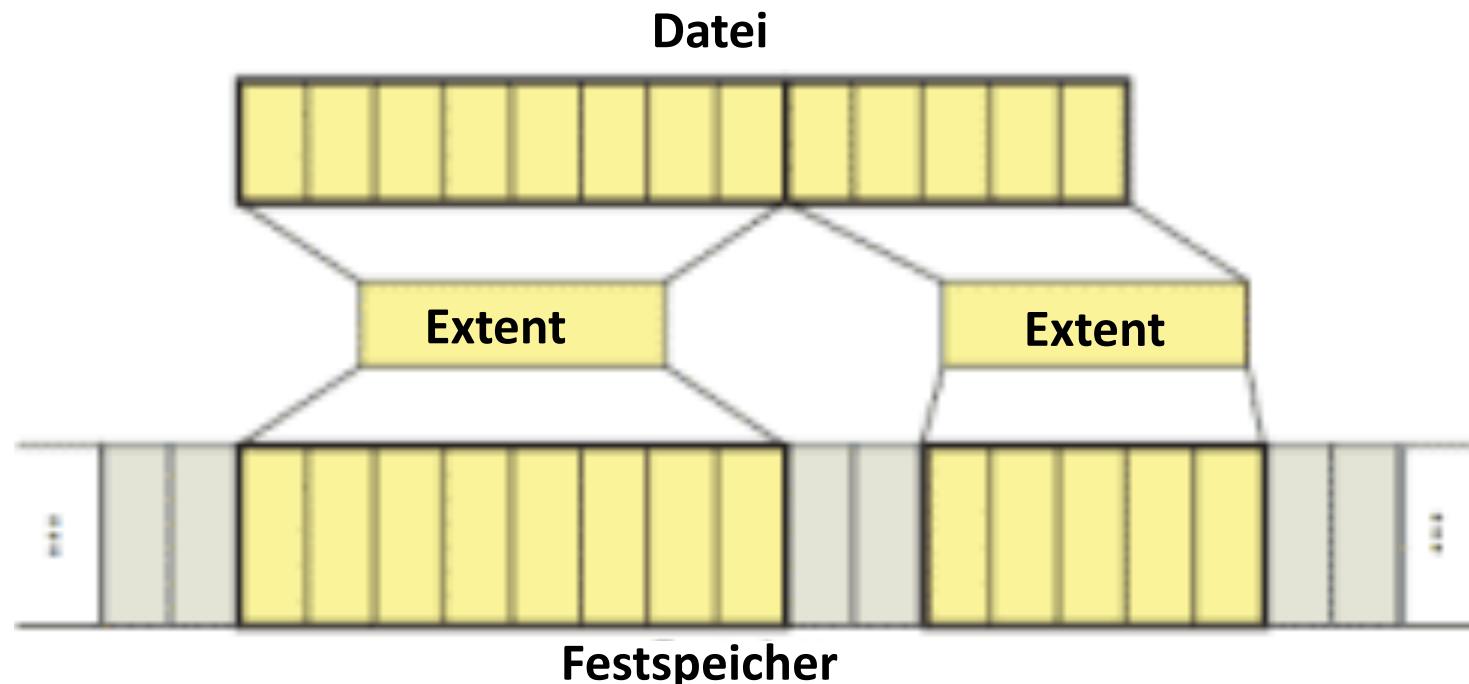
- Dateigröße auf 0
- Löschen aller Blockverweise

- **Journaling der I/O-Operationen**
 - Schreibe Kopie aller zu verändernden Blöcke ins Journal
 - Führe I/O-Operation durch
 - Lösche Blöcke aus dem Journal

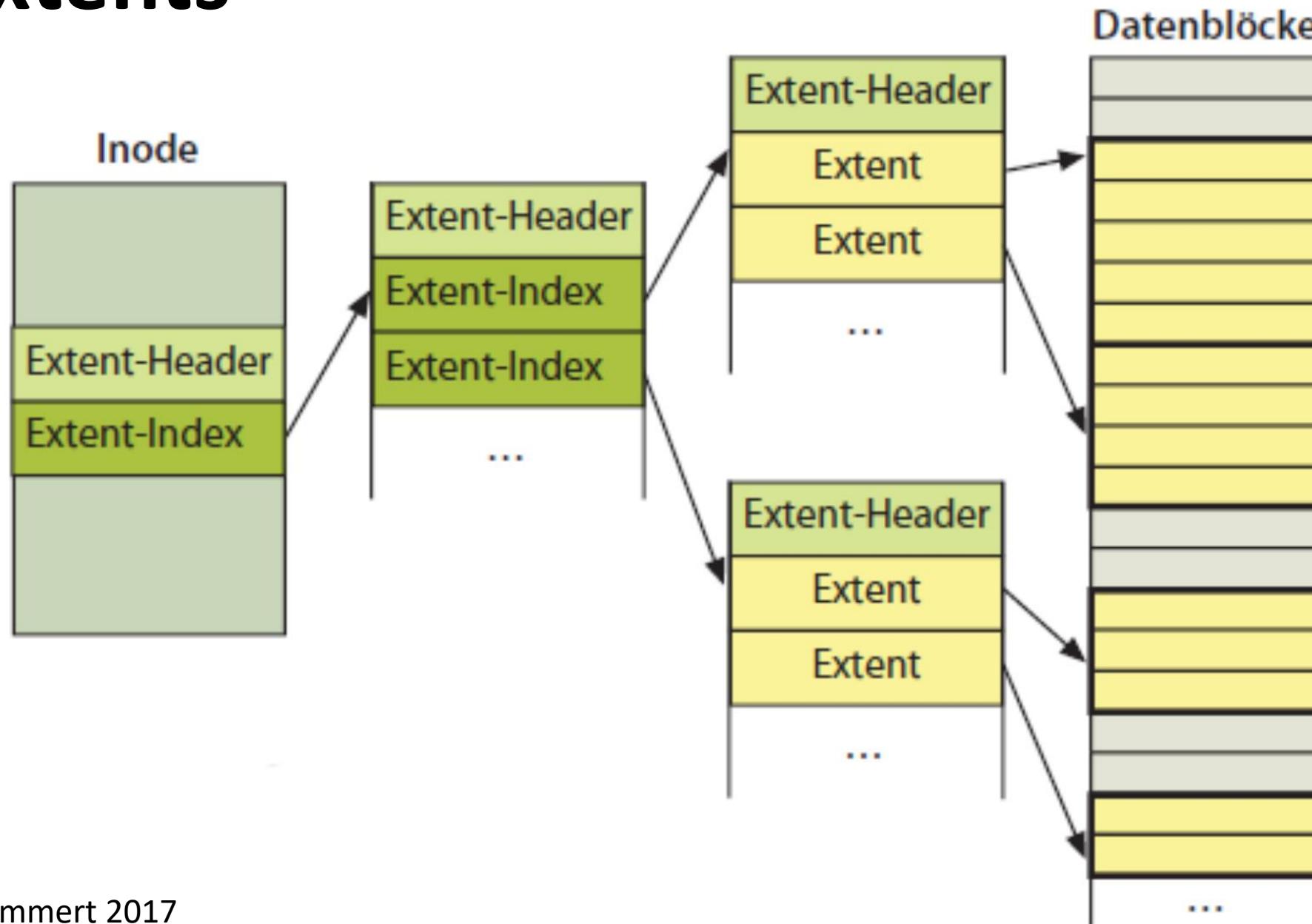
ext4

- **48 Bit Blocknummern (anstatt 32 Bit)**
- **Zeitstempel auf Nanosekundenbasis**
- **Journal bekommt Prüfsumme**
- **TRIM (für SSD)**
- **Automatische Defragmentierung im Betrieb**
- **Extents**

Extents



Extents



Achtung!

- Open Souce Dateisysteme
- frei anpassbar
- Superblock wird in struct `ext2_super_block` in Zeile 339 von `include/linux/ext2_fs.h` definiert
- Gruppendeskriptoren werden in `ext2_group_descr` structure in Zeile 148 von `ext2_fs.h` definiert
- Inode wird in struct `ext2_inode` in `ext2_fs.h` definiert
- Unter Linux kein File Slack, da zu bescheinige Blöcke immer erst mit Nullen überschrieben werden