4 Dateisysteme (Stand 05.06.2024)

- 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 Schicht des Betriebssystems (gehören zur Systemsoftware)
  - Prozesse und Benutzer greifen auf Dateien abstrakt über deren Dateinamen zu
  - Dateisystem greift auf (physische) Speicheradressen zu
- sollen wenig Overhead für Verwaltungsinformationen benötigen

Linux-Dateisysteme

Vergangenheit	Gegenwart	„Zukunft“
klassische Dateisysteme	Dateisysteme mit Journal	Copy-on-Write
Minix (1991)		**Btrfs** (2007/2016 WinBtrfs)
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

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 auch **Blöcke**
    - das führt zu Verwechslungen mit den Blöcken (Sektoren) auf Datenspeicherebene

Clustergröße

- die Größe der Cluster ist wichtig für die Effizienz des Dateisystems
  - kleine Cluster
    - steigender Verwaltungsaufwand für große Dateien
    - abnehmender Kapazitätsverlust durch interne Fragmentierung
  - große 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 ⇒ 1 kB geht verloren
- Dateigröße: 1 kB. Clustergröße: 64 kB ⇒ 63 kB gehen verloren!

- die Clustergröße kann man beim Anlegen des Dateisystems festlegen
- unter Linux gilt: Clustergröße ≤ Größe der Speicherseiten (Pagesize)
  - die Pagesize hängt von der Architektur ab
  - i386 = 4 kB, Alpha und Sparc = 8 kB, ia64 = maximal 64 kB

Benennung von Dateien und Verzeichnisssen

- wichtigste Eigenschaft von Dateisystemen aus Benutzersicht
  - Möglichkeiten der Benennung von Dateien und Verzeichnissen

- Dateisysteme haben diesbezüglich unterschiedlichste Eckdaten
  - Länge der Namen
    - 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*), weist auf den Dateiinhalt hin
  - die Endung ist eine kurze Folge von Buchstaben und Zahlen nach dem letzten Punkt im Dateinamen
- es existieren unterschiedlichste Arten von Dateiinhalten
- es hat sich durchgesetzt, dass der Inhalt von Dateien mit ihrer Endung deutlich angezeigt wird
- manche Dateien haben zwei 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 bestimmten Anwendungen zugewiesen

Datenzugriffe mit einem Cache beschleunigen

- moderne BS beschleunigen Zugriffe auf gespeicherte Daten mit einem **Cache** im HSP
- wird eine Datei lesend angefragt, schaut der Kernel zuerst im Cache nach

  liegt die Datei nicht im Cache vor, wird sie in den Cache geladen und von dort an die Anwendung durchgereicht
- der Cache ist nie so groß, wie die Menge der Daten auf dem System

  => selten nachgefrage Daten müssen verdrängt werden

  wurden Daten im Cache verändert, müssen die Änderungen spätestens beim Verdrägen zurückgeschrieben werden

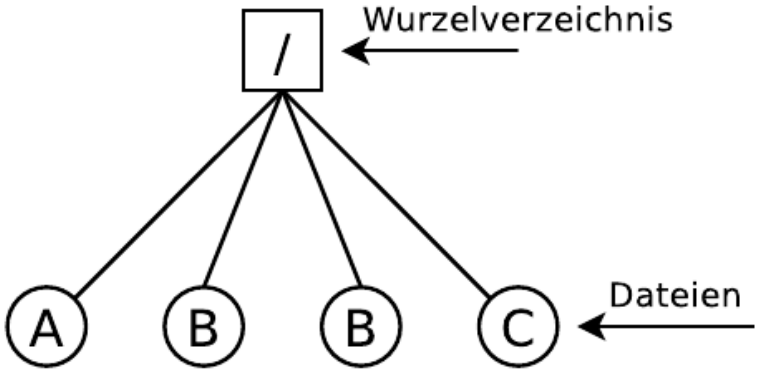
  optimales Verwenden des Cache ist nicht möglich, da Datenzugriffe nicht deterministisch (nicht vorhersagbar) sind
- die meisten Betriebssystemen geben Schreibzugriffe nicht direkt weiter (-> **Write-Back**)
- DOS und Windows verwenden den Puffer *Smartdrive*
- Linux puffert automatisch so viele Daten wie Platz im Hauptspeicher ist

  – – – stürzt das System ab, kann es zu Inkonsistenzen kommen

  + + + es wird eine höhere System-Geschwindigkeit erreicht

Verzeichnisse mit einer Ebene

- Dateisysteme stellen **Verzeichnisse** zum besseren Überblick zur Verfügung
  - Verzeichnisse sind häufig nur Text-Dateien, die die Namen und Pfade von Dateien enthalten
- Die einfachste Form eines Verzeichnissystems ist ein einzelnes Verzeichnis, das **Wurzelverzeichnis**, in dem alle Dateien abgelegt sind



- Situation auf frühen Computern

  nur ein Benutzer

  geringe Speicherkapazität

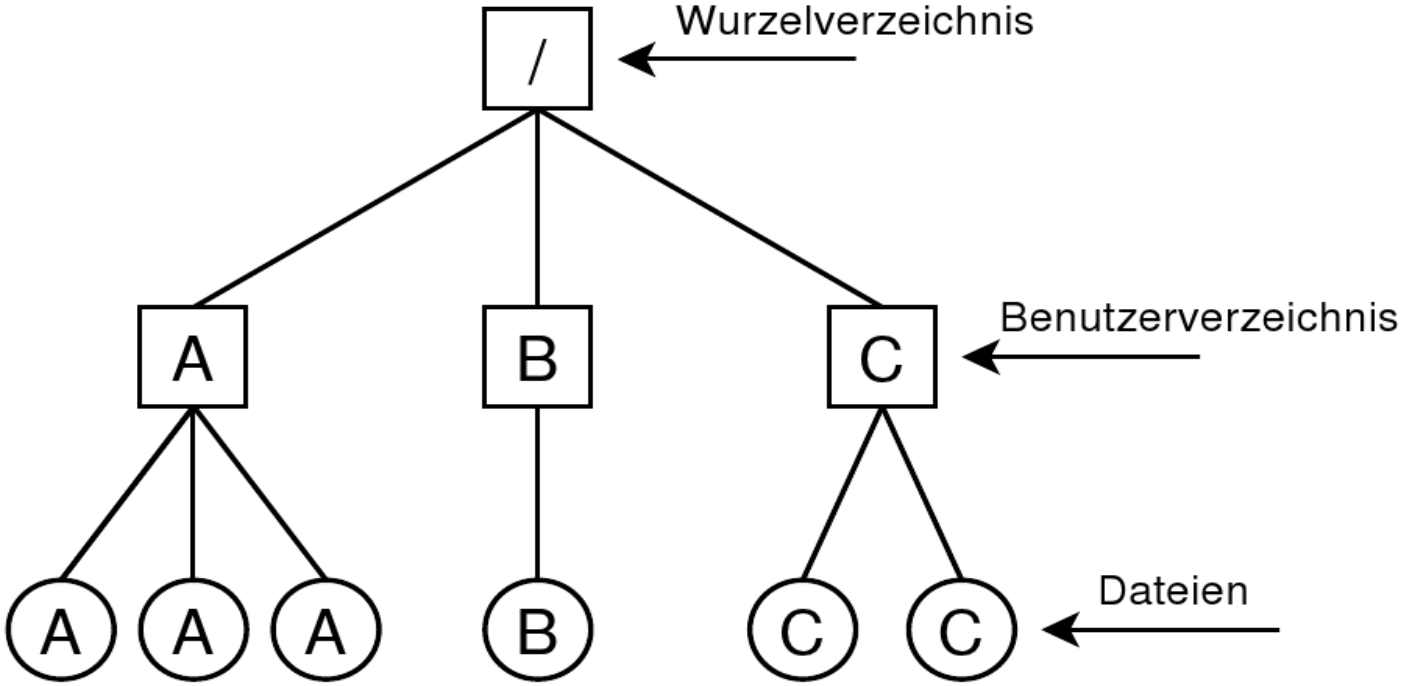
  => nur wenige Dateien

- liegen alle Dateien in einem Verzeichnis, kann es bei Mehrbenutzersystemen Konflikte mit den Dateinamen geben, da ein Nutzer die Daten eines anderen Nutzers überschreiben kann

  => dieser Ansatz ist heute nicht mehr praktikabel

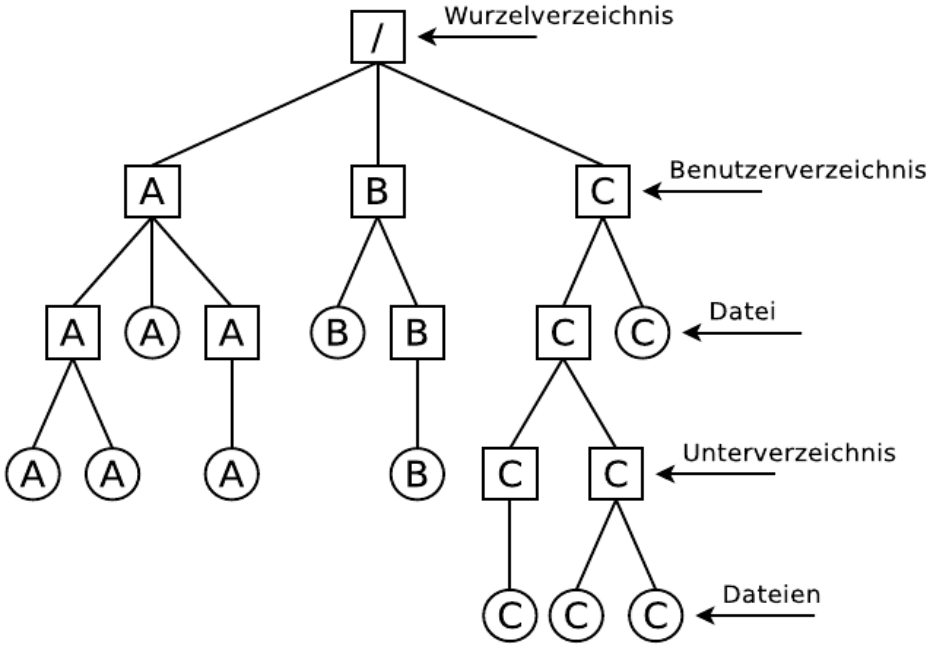
Verzeichnisse mit zwei Ebenen

- Problem: verschiedene Benutzer wollen für ihre Dateien die gleichen Dateinamen verwenden
- Lösung: jeder Benutzer bekommt sein eigenes, privates Verzeichnis



Hierarchische Verzeichnissysteme

- Verzeichnissysteme mit zwei Ebenen sind nicht immer ausreichend. Bei vielen Dateien genügt es nicht, die Dateien nach Benutzern separieren zu können
- es ist hilfreich, Dateien in logischen Gruppen zu arrangieren
- sinnvolle Unterteilung: Sortierung der Dateien nach ihrem Inhalt und/ oder der Zugehörigkeit zu Projekten bzw. Anwendungen



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

Pfadangaben

- ist das Verzeichnissystem als Baum organisiert, müssen die Pfadnamen zu den Dateien spezifiziert werden
- zwei Methoden existieren:

1. absolute Pfadnamen

beschreiben den kompletten Pfad **von der Wurzel bis zur Datei**

Absolute Pfade funktionieren immer

Beispiel: /home/xyz/.bashrc

2. relative Pfadnamen

alle Pfade, die nicht mit der Wurzel beginnen

wird immer in Verbindung mit dem aktuellen Verzeichnis gesehen

Beispiel: xyz/.bashrc

Pfadangaben - Trennzeichen

um in Pfadangaben Verzeichnishierarchien zu verdeutlichen, ist ein Separator (Trennzeichen) nötig

der Separator trennt die Komponenten des Pfads voneinander

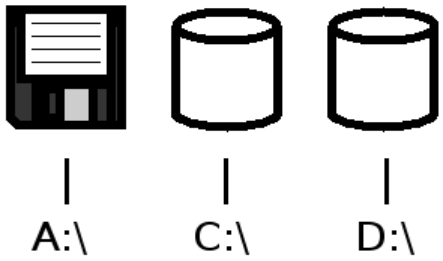
der Separator steht auch immer für das Wurzelverzeichnis

die meisten Betriebssystemfamilien verwenden ein eigenes Trennzeichen

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

Einbinden von Dateisystemen (1/3)

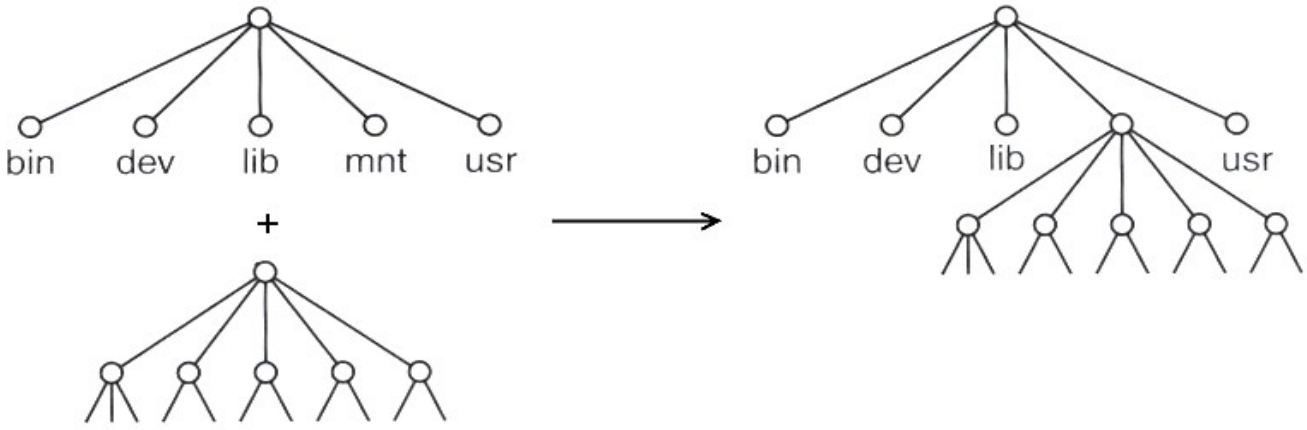
- Partitionen, USB-Sticks, CDs... besitzen jeweils eigenständige Dateisysteme.
- unter DOS/Windows werden diese als unterschiedliche "Laufwerke" betrachtet und erhalten jeweils einen **Laufwerkbuchstaben,** wodurch die eigenständigen Verzeichnishierarchien erhalten bleiben (entsprechend gibt es auch mehr als ein Wurzelverzeichnis)
- traditionell ist die Systempartition unter C:\ eingebunden, A:\ und B:\ stehen für Diskettenlaufwerke, weiteren Partitionen und Speichermedien werden D:\, E:\ usw. bis Z:\ zugewiesen
  - dieses Schema ist heute prinzipiell änderbar, jedoch setzen insbesondere ältere Programme die traditionellen Pfade voraus



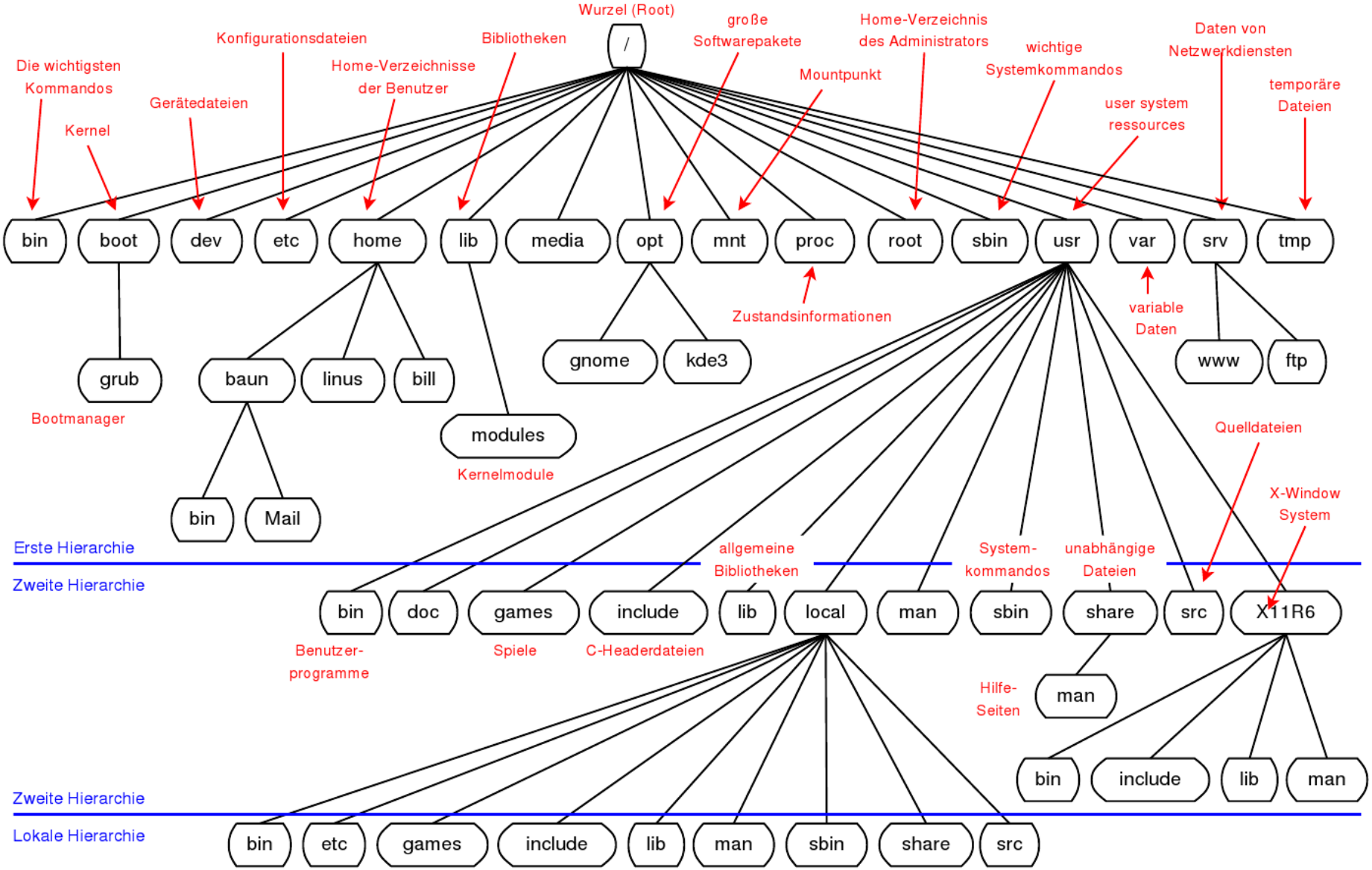
Einbinden von Dateisystemen (2/3)

- unter Unix/Linux werden alle Dateisysteme in eine Verzeichnishierarchie eingebunden (**gemountet**), es gibt nur ein Wurzelverzeichnis
- jeweils ein Dateisystem wird in ein Verzeichnis, den **Mountpunkt** gemountet. Unter Linux braucht dieses Verzeichnis nicht leer zu sein, nach dem Mounten sind seine Inhalte jedoch verborgen und damit nicht zugreifbar
- / ist der Mountpunkt für das Wurzelverzeichnis
- es ist nicht direkt ersichtlich, ob sich Verzeichnisse unterhalb von / auf dem selben oder auf unterschiedlichen Dateisystemen befinden. Hierdurch ist es möglich, ein Verzeichnis durch eine gemountete Partition zu ersetzen, ohne dass sich die Pfade ändern.
- Aufschluss über die gemounteten Systeme gibt der Befehl *mount*

Einbinden von Dateisystemen (3/3)

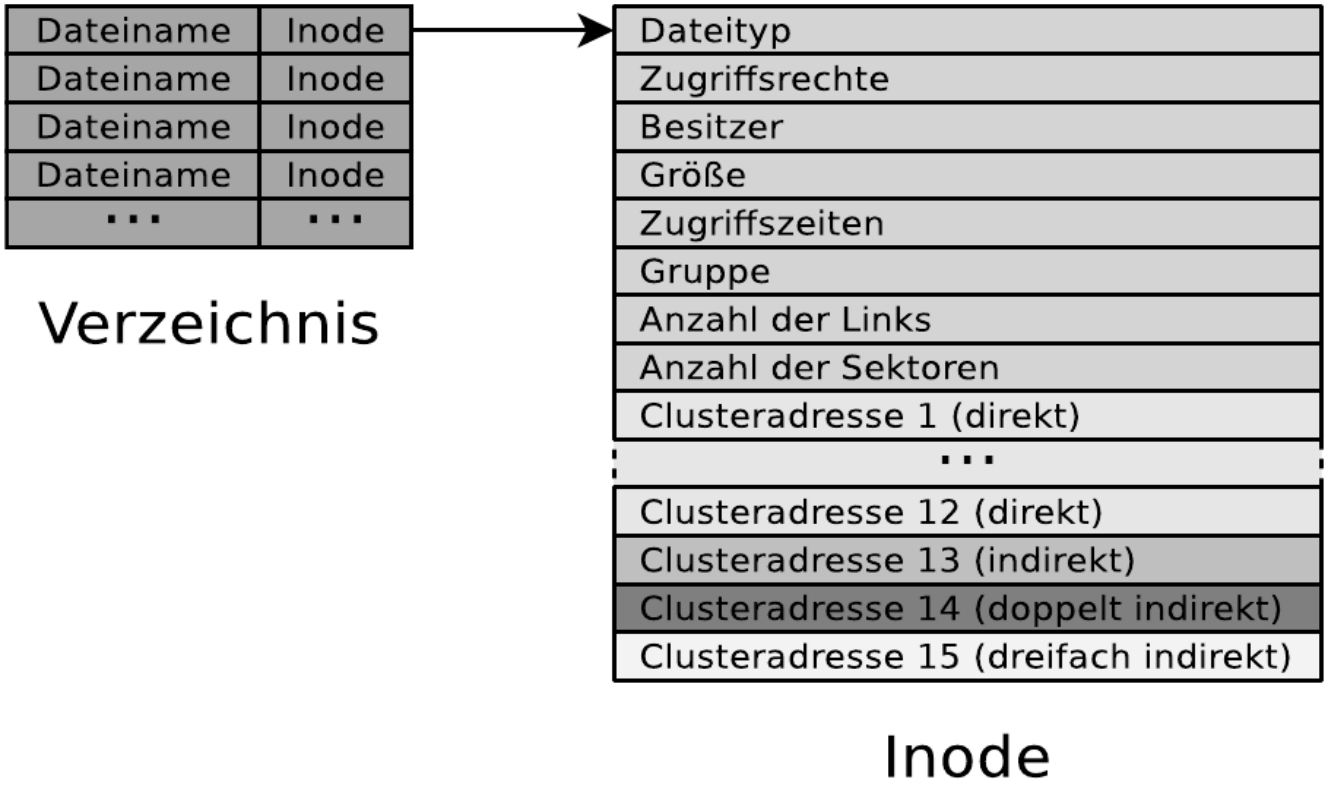


Linux/Unix-Verzeichnisstruktur

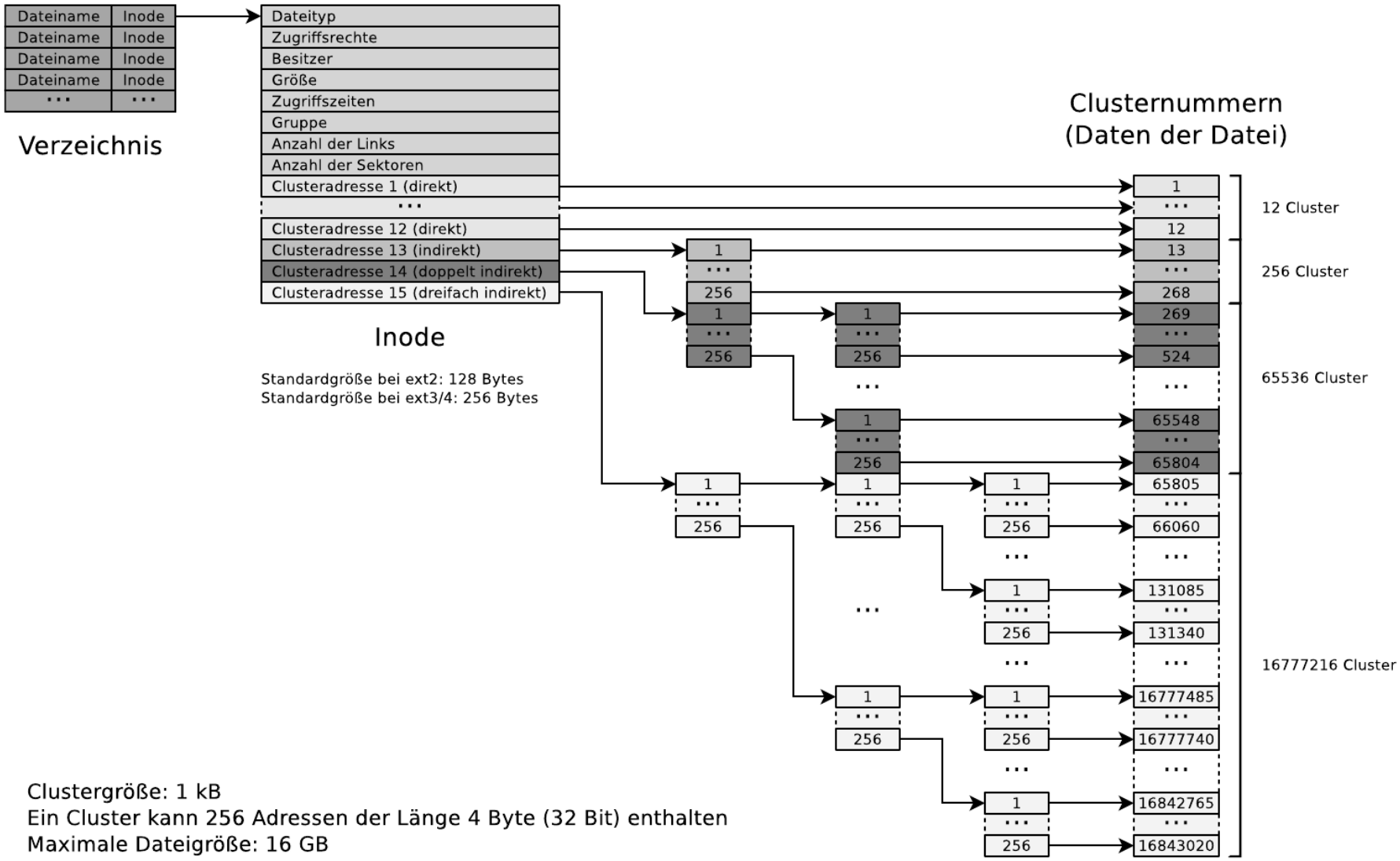


Blockadressierung am Beispiel ext2/3/4

- jeder Inode speichert die Nummern von 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



Direkte und indirekte Adressierung bei ext2, ext3, ext4



Clustergröße: 1 kB
Ein Cluster kann 256 Adressen der Länge 4 Byte (32 Bit) enthalten
Maximale Dateigröße: 16 GB

Minix

- ist ein Unix-ähnliches Betriebssystem, das seit 1987 von Andrew S. Tanenbaum als **Lehrsystem** entwickelt wird
- Minix war das Standard-Dateisystem unter Linux bis 1992; naheliegend, denn Minix war die Grundlage der Entwicklung von Linux. Die Entwicklung von Minix ist nicht eingeschlafen: **aktuell ist Version 3.3.0** aus dem Jahr 2015
- der Verwaltungsaufwand des Minix-Dateisystems ist gering. Es wird heute noch für Boot-Disketten und RAM-Disks verwendet. Speicher wird als **verkettete Liste gleichgroßer Blöcke** (1-8 kB) dargestellt
- ein Minix-Dateisystem enthält 6 Bereiche. **Die einfache Struktur macht es für die Lehre optimal**. Die Abbildung zeigt die Struktur einer Diskette mit 1440 Datenblöcken

Bereich 1	Bereich 2	Bereich 3	Bereich 4	Bereich 5	Bereich 6
Bootsektor *1 Block*	Superblock *1 Block*	Inode-Bitmap *1 Block*	Cluster-Bitmap *1 Block*	Inodes-Bereich *15 Blöcke*	Datenbereich *Rest*

Bereiche in einem Minix-Dateisystem

Bereich 1: Bootsektor. enthält den Boot-Loader, der das Betriebssystem startet

Bereich 2: Superblock. enthält Informationen über das Dateisystem,

z.B. Anzahl der Inodes und Cluster

Bereich 3: Inode-Bitmap. enthält eine Liste aller Inodes mit der Information, ob der Inode belegt (Wert: 1) oder frei (Wert: 0) ist

Bereich 4: Cluster-Bitmap. enthält eine Liste aller Cluster mit der Information, ob der Cluster belegt (Wert: 1) oder frei (Wert: 0) ist.

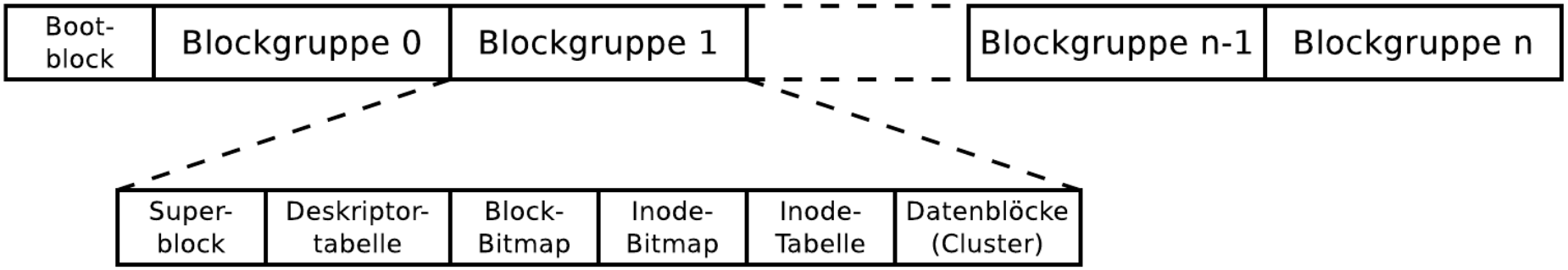
Bereich 5: Inodes-Bereich. enthält die Metadaten der Inodes.

- jede Datei und jedes Verzeichnis wird von mindestens einem Inode repräsentiert, der Metadaten enthält
- Metadaten sind u.a. Dateityp, UID/GID, Zugriffsrechte, Größe

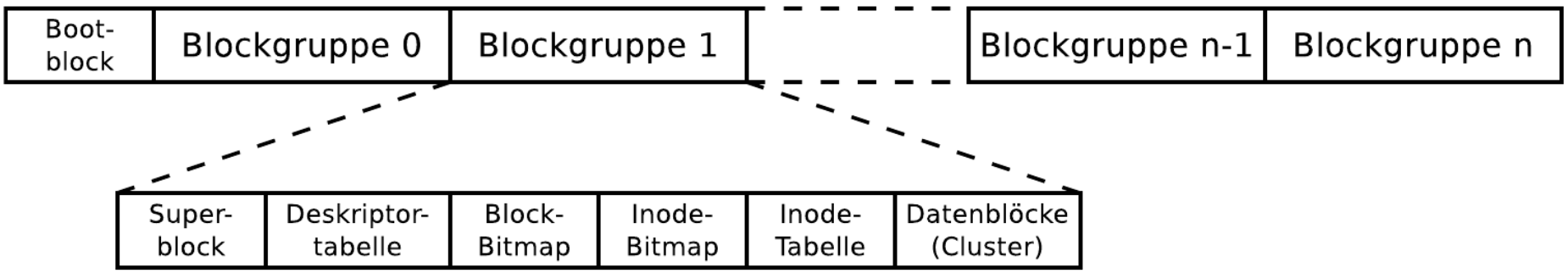
Bereich 6: Datenbereich. Hier ist der Inhalt der Dateien und Verzeichnisse. Dieser Bereich nimmt den größten Platz im Dateisystem ein

ext2, ext3

- 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 jeder 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

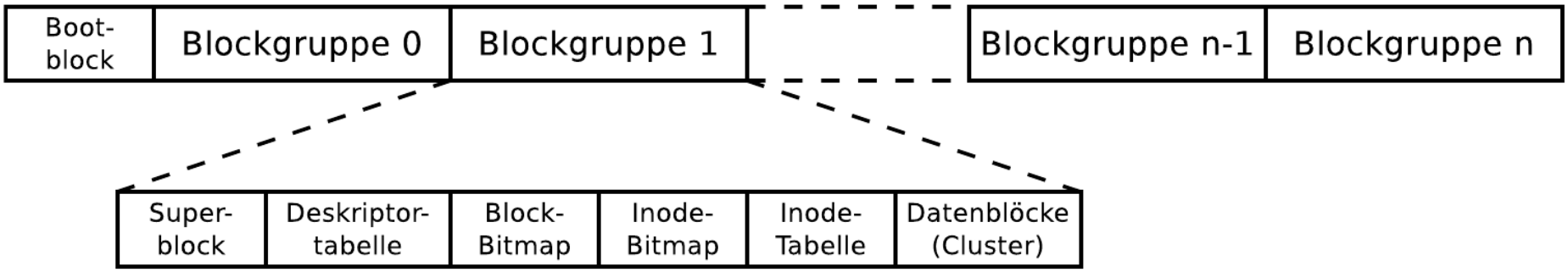


Schema der Blockgruppen bei ext2, ext3 (1/2)



- der erste Cluster des Dateisystems enthält den **Bootblock**
  - der Bootblock ist 1 kB groß
  - er enthält den Bootmanager, der das Betriebssystem startet
- in jeder Blockgruppe befindet sich zur Verbesserung der Datensicherheit eine **Kopie des Superblocks**
- 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

Schema der Blockgruppen bei ext2, ext3 (2/2)



- Block- und Inode-Bitmap sind jeweils einen Cluster groß
  - sie enthalten Informationen über die Cluster- und Inode-Belegung in der Blockgruppe
  - diese Bitmaps sind ein Abbild der freien und belegten Cluster und Inodes
- die Inode-Tabelle enthält die Inodes der Blockgruppe
- die restlichen Cluster der Blockgruppe sind für die Daten nutzbar

File Allocation Table (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, in der die belegten und freien Cluster im Dateisystem erfasst sind

- für jeden Cluster des Dateisystems existiert ein Eintrag in der Dateizuordnungstabelle mit folgenden Informationen über den Cluster:
  - Cluster ist frei oder das Medium an dieser Stelle beschädigt
  - Cluster ist von einer Datei belegt und enthält die Adresse des nächsten Clusters, der zu dieser Datei gehört bzw. ist der letzte Cluster der Datei

- die Cluster einer Datei bilden eine verkettete Liste (**Clusterkette**)

Bereiche in einem FAT-Dateisystem (1/3)

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 (soll das Betriebssystem laden) 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

Bereiche in einem FAT-Dateisystem (2/3)

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

- zwischen dem Bootsektor und erster Dateizuordnungstabelle können sich **reservierte Sektoren**, eventuell für den Bootmanager, befinden

  diese Cluster können nicht vom Dateisystem benutzt werden

- in der **Dateizuordnungstabelle** (FAT) sind die belegten und freien Cluster im Dateisystem erfasst

  die Konsistenz der FAT ist für die Funktionalität des Dateisystems elementar darum existiert üblicherweise eine Kopie, um bei Datenverlust noch eine vollständige Dateizuordnungstabelle zu haben

Bereiche in einem FAT-Dateisystem (3/3)

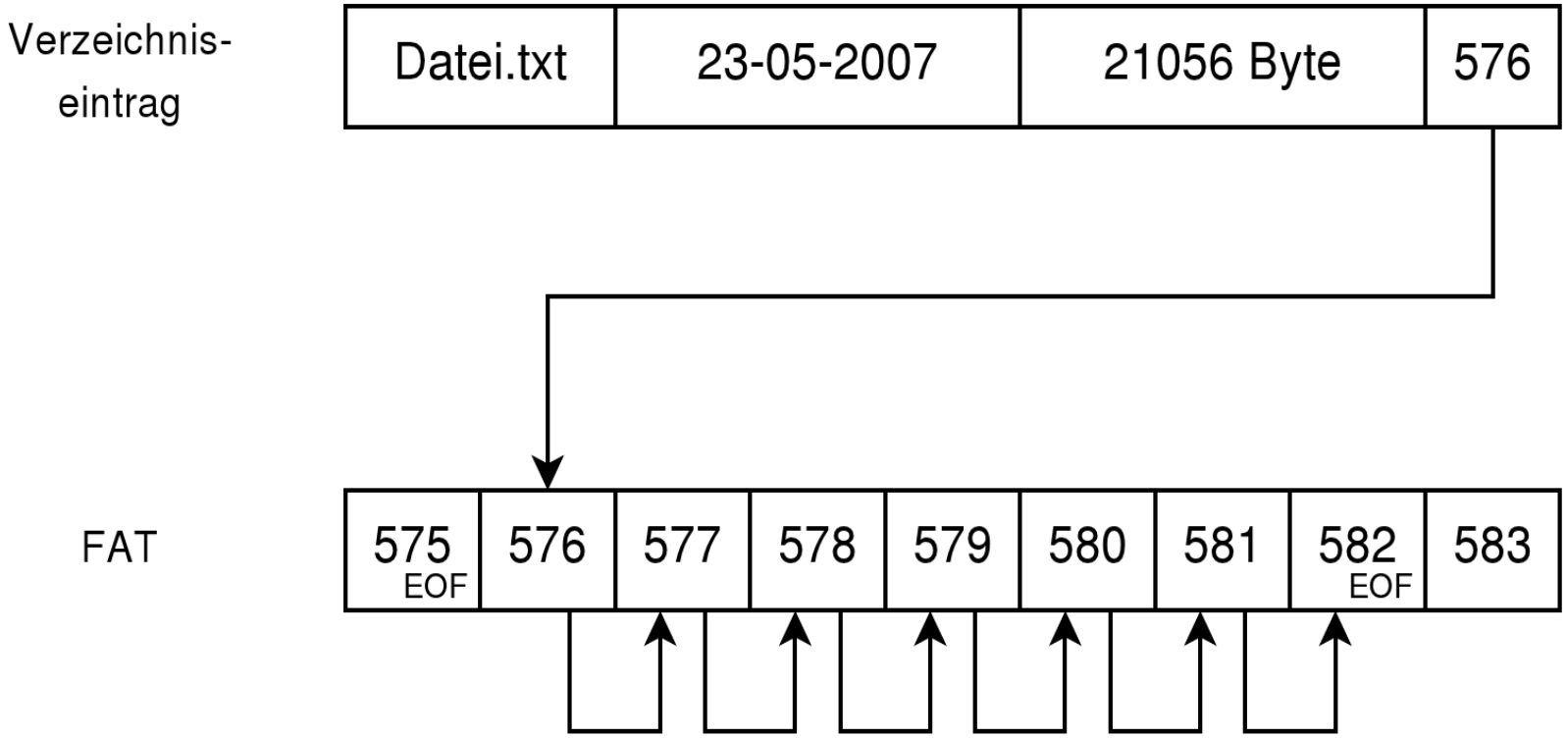
Bereich 1	Bereich 2	Bereich 3	Bereich 4	Bereich 5	Bereich 6
Bootsektor	Reservierte Sektoren	FAT 1	FAT 2	Stamm-verzeichnis	Datenbereich

- im **Stammverzeichnis** (Wurzelverzeichnis) ist jede Datei und jedes Verzeichnis durch einen Eintrag repräsentiert:
  - Bei FAT12 und FAT16 befindet sich das Stammverzeichnis direkt hinter den Kopien der FAT und hat eine feste Größe

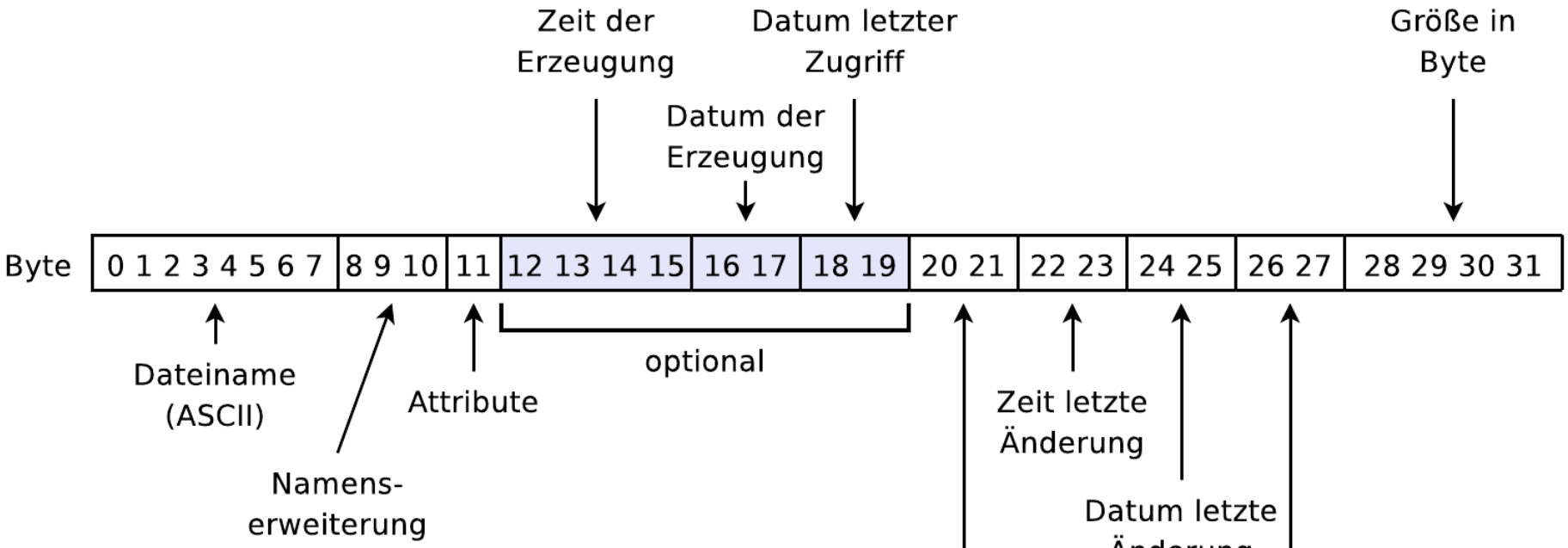
    Die maximale Anzahl an Verzeichniseinträgen ist somit begrenzt
  - Bei FAT32 kann sich das Stammverzeichnis an beliebiger Position im Datenbereich befinden und hat eine variable Größe

- Der letzte Bereich enthält die eigentlichen **Daten**

Stammverzeichis (Wurzelverzeichnis) und FAT



Struktur eines Eintrags im Stammverzeichnis der FAT



Attribute	
0	read-only
1	hidden
2	system
3	Name des Datenträgers
4	directory
5	archiv
6	x
7	x

FAT12

- erschien 1980 mit der ersten QDOS-Version
- die Clusternummern sind 12 Bits lang
  - maximal 2^12 = 4.096 Cluster können angesprochen werden
- Clustergröße: 512 Bytes bis 4.096 kB
  - eine 3,5 Zoll Diskette mit 1,44 MB Speicherkapazität hat 2.880 Cluster
- Dateinamen werden nur im Schema 8.3 unterstützt
  - es stehen 8 Zeichen für den Dateinamen und 3 Zeichen für die Dateinamenserweiterung zur Verfügung
- FAT12 wird heute noch auf DOS- und Windows-Disketten eingesetzt

FAT16

- erschien 1983: absehbar, dass ein Adressraum von 16 MB zukünftig zu wenig ist
- maximal 2^16 = 65.524 Cluster können angesprochen werden
  - 12 Cluster sind reserviert
- Clustergröße: 512 Bytes bis 16 kB, eine Clustergröße von 64 kB ist nur unter Windows NT verfügbar
- die maximale Datei- und Partitionsgröße ist unter MS-DOS und Windows 9x bis zu 2 GB und unter Windows NT bis zu 4 GB groß
- Dateinamen werden nur im Schema 8.3 unterstützt
- Haupteinsatzgebiet heute mobile Datenträger mit Speicherkapazität < 2 GB

FAT32

- erschien 1997 als Reaktion auf die höhere Festplattenkapazität und weil 64 kB große Cluster sehr viel Speicher verschwenden
- Einträge in der Dateizuordnungstabelle sind 32 Bytes groß
  - es sind theoretisch 2^32 = 4.294.967.295 Cluster adressierbar
  - 4 Bytes sind reserviert
  - deshalb können nur 2^28 = 268.435.456 Cluster adressiert werden (korrekter Name wäre also FAT28)
- Clustergröße: 512 Bytes bis 32 kB
- maximale Länge der Dateinamen: 255 Zeichen
- maximale Dateigröße: 4 GB
- das Haupteinsatzgebiet sind heute mobile Datenträger > 2 GB

VFAT

- VFAT (Virtual Fat) erschien 1997
  - Erweiterung für FAT12, FAT16 und FAT32
- 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

NTFS – New Technology File System

- eingeführt 1993 mit Windows NT 3.1

- NTFS bietet im Vergleich zu seinem Vorgänger FAT u.a.:
  - maximale Dateigröße: 16 TB (--> Extents)
  - maximale Partitionsgröße: 256 TB (--> Extents)
  - Sicherheitsfunktionen auf Datei- und Verzeichnisebene
  - maximale Länge der Dateinamen: 255 Zeichen
  - Dateinamen können aus fast beliebigen Unicode-Zeichen bestehen
    Ausnahmen: \0 und /
  - Clustergröße: 512 Bytes bis 64 kB

Entwicklung von NTFS (1/2)

- es existieren verschiedene Versionen des NTFS-Dateisystems
  alle Versionen sind zu früheren Versionen abwärtskompatibel

  - NTFS 1.x: 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

Entwicklung von NTFS (2/2)

- NTFS-Unterstützung für nicht-Windows Betriebssysteme ist schwierig
  - es existieren viele unterschiedliche Versionen
  - Microsoft behandelt die genaue Funktionsweise als Betriebsgeheimnis

- aktuelle Versionen von NTFS bieten zusätzliche Features:
  - Unterstützung für Kontingente (Quota) ab Version 3.x
  - Transparente Kompression
  - Transparente Verschlüsselung (Triple-DES und AES) ab Version 2.x

Kompatibilität zu MS-DOS (1/2)

- NTFS speichert für jede Datei einen eindeutigen 8+3 Zeichen langen Dateinamen. So können Microsoft-Betriebssysteme ohne NTFS-Unterstützung auf Dateien auf NTFS-Partitionen zugreifen

- Problem: Die kurzen Dateinamen müssen eindeutig sein

- Lösung:
  - Alle Sonderzeichen und Punkte innerhalb des Namens werden bis auf den letzten Namen gelöscht

  - Alle Kleinbuchstaben werden in Großbuchstaben umgewandelt

Kompatibilität zu MS-DOS (2/2)

- Lösung (weiter):
  - 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 dargestellt als „EINGAN~1.pdf“

Struktur von NTFS

- das Dateisystem enthält eine Hauptdatei, die Master File Table (MFT)
  - diese enthält die Referenzen, welche Blöcke zu welcher Datei gehören
  - zudem befinden sich die Metadaten der Dateien (Dateigröße, Datum der Erstellung, Datum der letzten Änderung, Freigabe, Dateityp und eventuell auch der Dateiinhalt in der MFT

    der Inhalt kleiner Dateien wird in der MFT direkt gespeichert

- beim Formatieren einer Partition wird für die MFT ein fester Platz reserviert
  - standardmäßig werden für die MFT 12,5% der Partitionsgröße reserviert
  - dieser Platz kann nicht von anderen Dateien belegt werden
  - ist der Platz voll, verwendet das Dateisystem freien Speicher der Partition für die MFT

    dabei kann es zu einer Fragmentierung der MFT kommen

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
  - Daten sind meist nicht zusammenhängend angeordnet und darum sind Schreibzugriffe an mehreren Stellen im Dateisystem nötig
- kommt es während Schreibzugriffen zum Ausfall, muss das Dateisystem nach inkonsistenten Daten untersucht 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 => Journaling-Dateisysteme

Journaling-Dateisysteme

- diese Dateisysteme führen ein Journal über die Daten, auf die Schreibzugriffe durchgeführt werden
  - jede Schreiboperation besteht aus mehreren Teiloperationen (Schritten)
  - die Schritte mehrerer Schreibzugriffe werden im Journal gesammelt
- in festen Zeitabständen wird das Journal geschlossen und die Schreiboperationen durchgeführt
- Vorteil: nach einem Absturz müssen nur diejenigen Dateiinhalte (Cluster) und Metadaten überprüft werden, die im Journal stehen
- Nachteil: Journaling erhöht die Anzahl der Schreiboperation, weil Änderungen erst im Journal angemeldet und dann durchgeführt werden
- zwei Varianten des Journaling
  - Metadaten-Journaling
  - vollständiges Journaling

Metadaten-Journaling und vollständiges Journaling

- **Metadaten-Journaling**
  - das Journal enthält nur Änderungen an den Metadaten
    - Änderungen werden direkt in das Dateisystem geschrieben
    - nur die Konsistenz der Metadaten ist nach einem Absturz garantiert
  - Vorteil: Konsistenzprüfungen dauern nur wenige Sekunden
  - Nachteil: Datenverlust durch einen Systemabsturz ist weiterhin möglich
  - Optional bei ext3, ext4 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 Dateiinhalte ist garantiert
  - Nachteil: alle Schreiboperation müssen doppelt ausgeführt werden
  - Optional bei ext3, ext4 und ReiserFS

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 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
  - Beispiel: XFS, JFS, Standard bei ext3, ext4 und ReiserFS

Modernstes Konzept: Copy-On-Write

- 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 wurden, bleibt die Originaldatei erhalten und kann nach einem Absturz weiter verwendet werden

  die Datensicherheit ist besser als bei Dateisystemen mit Journal
- Beispiel: btrfs

Fragmentierung

- in einem Cluster darf nur eine Datei gespeichert werden

  ist eine Datei größer als ein Cluster, wird sie auf mehrere verteilt

- im besten Fall liegen die Cluster einer Datei direkt nebeneinander
  - Ziel: häufige Bewegungen des Schwungarms der Festplatte vermeiden
    - liegen die Cluster einer Datei über die Festplatte verteilt, müssen die Festplattenköpfe bei Zugriffen auf die Datei viele zeitaufwendige Positionswechsel durchführen
    - bei Solid State Drives (SSD) spielt die Position der Cluster keine Rolle für die Zugriffsgeschwindigkeit
  - sind die logisch zusammengehörenden Cluster einer Datei nicht räumlich nebeneinander, spricht man von **Fragmentierung**

Defragmentierung (1/3)

- es kommen immer wieder folgende Fragen auf:
  - warum ist es unter Linux/UNIX nicht üblich, zu defragmentieren?
  - kommt es unter Linux/UNIX überhaupt zu Fragmentierung?
  - gibt es unter UNIX überhaupt Möglichkeiten, zu defragmentieren?
- zu allererst ist zu klären, was man mit defragmentieren erreichen will
- 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 Warte- und Suchzeiten mehr vorkommen können
- nur wenn die Suchzeiten sehr groß sind, macht Defragmentierung Sinn

Defragmentierung (2/3)

- bei Betriebssystemen, die kaum Hauptspeicher zum Cachen der Festplattenzugriffe verwenden, sind die Suchzeiten dominierend
- bei einem Betriebssystem, das nur Singletasking unterstützt, kann immer nur eine Anwendung laufen
  - wenn diese Anwendung ständig blockiert, weil sie auf die Ergebnisse von Lese- und Schreibanforderungen auf einen Datenträger wartet, ist das negativ für die Leistung des Systems, vgl. MS-DOS
- bei Betriebssystemen mit Multitasking laufen immer mehrere Programme und es steht ausreichend Cache im RAM zur Verfügung
- Anwendungen wollen oder können fast nie große Datenmengen am Stück lesen, ohne dass andere Anwendungen weitere Zugriffe an anderen Stellen des Mediums dazwischenschieben

Defragmentierung (3/3)

- damit sich gleichzeitig laufende Programme nicht zu sehr gegenseitig behindern, wird davon ausgegangen, dass nach einer Anzahl vorwärts laufender Leseanfragen weitere in der gleichen Sequenz erfolgen
  - das System liest dann einen Vorrat an Daten in den Cache ein, auch wenn dafür noch keine Anfragen vorliegen
- in Betriebssystemen mit Singletasking können konkurrierende Zugriffe auf Speichermedien nicht auftreten
- Daten, auf die Prozesse häufig zugreifen, werden von Linux-Systemen automatisch im Cache gehalten
  - die Wirkung des Cache überwiegt bei weitem die kurzzeitigen Vorteile, die eine Defragmentierung hätte
- Defragmentieren hat mehr 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 aber häufig nicht empfehlenswert und sinnvoll

NFS - Network File System (1/2)

- NFS ist kein Dateisystem im eigentlichen Sinne, sondern ein Protokoll, das es ermöglicht, auf Dateisysteme entfernter Rechner in einer transparenten Weise zuzugreifen.
- hierzu wird auf dem NFS-Server eine Freigabe eingerichtet (unter Linux in der Datei /etc/exports)
  - Beispiel (Freigabe nur für bestimmte IP)

```
# /pfad/zur/freigabe  hostname1(Optionen) hostname2(Optionen)
/etc/ 86.59.118.148(rw,sync)
```

- durch den Client kann diese Freigabe mittels *mount servername:/etc /mountpunkt* gemountet werden
  - während für den Mountvorgang der Server und der genaue Pfad bekannt sein muss, kann die Freigabe anschließend genauso verwendet werden, wie ein Verzeichnis im lokalen Dateisystem
- den Mountvorgang kann man durch einen Eintrag in die /etc/fstab automatisieren

```
# <file system>  <mount point>   <type>  <options>  <dump>  <pass>
  servername:/etc /mountpunkt     nfs        rw        0       0
```

NFS - Network File System (2/2)

Optionen für /etc/exports (siehe auch *man exports*):

- **ro**,**rw** - read only, read write
- **no_root_squash** - Superuser des Clients erhält vollen root-Zugriff
  - default ist **root_squash** - Anfragen des Superusers werden in solche des Users nobody umgewandelt
- **subtree_check** - für alle Anfragen wird überprüft, ob sie sich auf eine Datei unterhalb des exportierten Verzeichnisses beziehen - dies ist der Default
- **no_subtree_check** - es wird lediglich überprüft, ob sich die Anfrage auf eine Datei im gleichen Dateisystem bezieht.
- **sync** - für Schreiboperationen wird erst Erfolg gemeldet, wenn sie tatsächlich durch den Server durchgeführt wurden - Standard in neueren Versionen von NFS
- **async** - es wird sofort Erfolg gemeldet - bessere Performance, aber möglicher Datenverlust

Erwägungen zur Sicherheit:

- ohne zusätzliche Authentifizierung (z.B. Kerberos) besteht keine Möglichkeit, die Quelle einer Anfrage sicher zu erkennen
- NFS ist ein unverschlüsseltes Protokoll und daher ungeeignet für sensible Daten