4.1 CD-ROM (3)

- Varianten der CD-ROM (Philips, *Orange Book*, 1988)
  - **CD-R** (*Recordable*)
    - reflektierende Farbschicht auf Substrat
    - leistungsstarker Laserstrahl kann Reflexivität der Farbschicht irreversibel verändern und somit nicht reflektierende *Pits* in der Spur erzeugen
    - i.a. mit normalen CD-ROM Laufwerken lesbar
    - Schreibmodi: *Disk-at-Once* oder *Multi-Session* (iterativ beschreibbar)
  - **CD-RW** (*ReWritable*)
    - Schicht mit zwei stabilen Zuständen (polykristallin mit hoher Reflexivität und amorph mit geringer Reflexivität) auf Substrat
    - Zustandswechsel durch Ersitzen auf verschiedene Temperaturen (200ºC für polykristallin und 500 ºC für amorph) mittels Laserstrahl
    - vielfach beschreibbar!
    - aufgrund geringerer Reflexivität nicht kompatibel zu CD-ROMs
    - Rohlinge sind geringfügig teurer als bei CD-R

4.1 CD-ROM (4)

- **DVD** (*Digital Versatile Disk,* 1996)
  - Weiterentwicklung der CD-ROM:
    - schmalere Spur (Breite 0.74 µm statt 1.6 µm)
    - kleinere *Pits* (Länge nur 0.4 µm statt 0.8 µm), möglich durch bessere Optik und Einsatz eines Laserstrahls kürzerer Wellenlänge (635 nm statt 780 nm)
    - Lesegeschwindigkeit 3.5 m/s statt 1.2 m/s (1x)
    - jedoch höhere Empfindlichkeit gegenüber Fingerabdrücken und Kratzer
  - Kapazität: **4.7 GByte** (einseitig), 9.4 GByte (zweiseitig)
- wiederbeschreibbare Varianten der DVD:
  - **DVD-R** (Pioneer, 1997): nur einmal beschreibbar
  - **DVD-RAM** (Toshiba, Hitachi, 1998): ca. 100000-fach beschreibbar
  - **DVD-RW** (Pioneer, 2001): ca. 1000-fach beschreibbar
  - **DVD+RW** (Sony, Philips, ... , 2001): ca. 1000-fach beschreibbar
  - kaum Kompatibilität!

4.2 Floppy

- eine Diskette hat einen ähnlichen Aufbau wie eine Festplatte
  - flexible magnetisierbare Scheibe in einer Plastikschutzhülle
  - maximal zwei Köpfe (Ober- und Unterseite)
  - Köpfe berühren die Diskettenoberfläche (nicht bei ZIP)
- typische Daten einiger Diskettenformate:

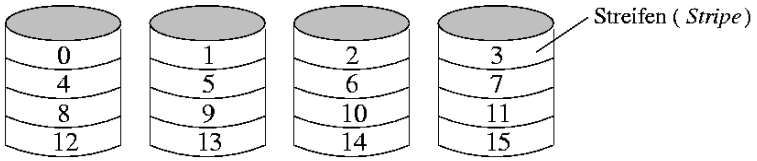
Bezeichnung	**3 ½ Zoll, HD**	**3 ½ Zoll, ED**	**ZIP**
Kapazität (formatiert)	1.44 MByte	2.88 MByte	100 MByte
Köpfe	2	2	2
Spurbreite	187.5 µm	187.5 µm	12 µm
Spuren je Seite	80	80	1817
Sektoren je Spur	18	36	ZBR
Transferrate	62.5 kByte/s	125 kByte/s	1 MByte/s
Umdrehungen / Minute	360	360	3000

5 RAID-Systeme

- Idee von Patterson et al. (University of Berkeley, 1988):
  Einsatz mehrerer unabhängiger Platten (**RAID** = *Redundant Array of Independent Disks*) zur
  - Erhöhung der Transferleistung durch **verteilte** Speicherung
  - Erhöhung der Ausfallsicherheit durch **redundante** Speicherung
- Steuerung eines RAID entweder in Software (Betriebssystem) oder in Hardware (eigener Controller) implementierbar
- heute (2003) existieren 8 verschiedene RAID-Varianten
  - als RAID 0 bis RAID 7 bezeichnet
  - jedoch werden nur **RAID 0**, **RAID 1**, **RAID 4** und **RAID 5** in der Praxis eingesetzt und hier behandelt
- Voraussetzung: mehrere Platten gleicher Größe, schneller Festplattenbus

5.1 RAID 0

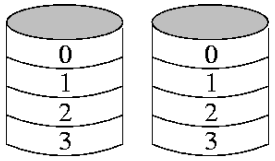
- auch als **gestreifte Platten** (*Striping*) bezeichnet
- eine logische Platte wird in kleine Streifen (*Stripes*) zerschnitten, die zyklisch über mehrere physikalische Platten verteilt werden



- Vorteile:
  - schnellere Datentransfers, da bei Zugriff auf eine große Datei mehrere Platten gleichzeitig angesprochen werden
- Nachteile:
  - keine Redundanz, d.h. bei Ausfall einer Platte fällt RAID-System aus

5.1 RAID 1

- auch als **gespiegelte Platten** (*Mirroring*) bezeichnet
- Dateien werden gleichzeitig auf zwei Platten gespeichert

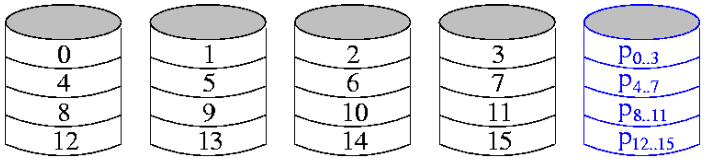


(auch mit RAID 0 kombinierbar)

- Vorteile:
  - schnelleres Lesen großer Dateien, da Zugriffe über zwei Platten verteilt werden können
  - auch bei Ausfall einer Platte noch betriebsbereit
- Nachteile:
  - geringfügig langsameres Schreiben durch Warten auf zwei Plattenaufträge
  - doppelter Speicherbedarf und somit doppelte Kosten für Festplatten

5.1 RAID 4

- Dateien werden wie bei RAID 0 über mehrere Platten verteilt; eine zusätzliche **Paritätsplatte** speichert Parität



- jeder Paritätsblock enthält die bitweise Parität p von allen zugehörigen Datenblöcken

  z.B. für Bit *i* in den Streifen 0 bis 3 gilt: $p_{0..3}(i) = x_0(i) \oplus x_1(i) \oplus x_2(i) \oplus x_3(i)$
- Vorteile:
  - schnelleres Lesen, da Zugriffe über mehrere Platten verteilt werden
  - auch bei Ausfall einer Platte noch betriebsbereit, z.B. bei Ausfall der Platte 1 kann $x_1(i) = x_0(i) \oplus p_{0..3}(i) \oplus x_2(i) \oplus x_3(i)$ rekonstruiert werden

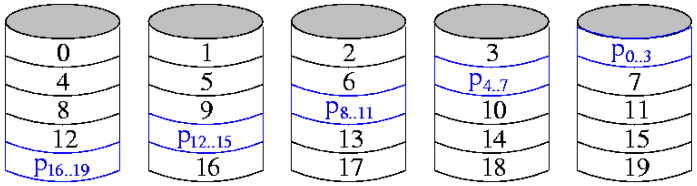
5.1 RAID 4 (2)

- Nachteile:
  - jeder Schreibvorgang eines einzelnen Datenblocks j erfordert die Aktualisierung des zugehörigen Paritätsblocks:

    $p^{neu}(i) = p^{alt}(i) \oplus x_j^{neu}(i) \oplus x_j^{alt}(i)$
  - erfordert Lesen der alten Inhalte von Paritätsblock und Datenblock j, d.h. vier Zugriffe je Schreibvorgang!
  - Paritätsplatte ist hoch belastet

5.1 RAID 5

- wie RAID 4, jedoch mit **verteilten Paritätsblöcken**



- Vorteile und Nachteile:
  - wie RAID 4, jedoch wird die zusätzliche Belastung durch Schreiben des Paritätsblocks auf alle Platten verteilt
- RAID 5 ist die heute gängigste RAID-Variante!

6 Dateisysteme

- ein Dateisystem ist eine Abstraktion des Betriebssystems zur **geräteunabhängigen** Verwaltung von Dateien
  - einheitliche Sicht auf verschiedene Arten von Sekundärspeicher, z.B. Festplatten, Floppys, CD-ROMs, DVDs und Bandlaufwerke
  - Benutzer muss sich nicht um physikalische Datenformate auf den verschiedenen Arten der Sekundärspeicher kümmern
  - jede Datei wird als eine Menge von **Blöcken** fester Größe repräsentiert (wobei ein Block einem oder mehrerer Sektoren entsprechen kann)
- Dateisysteme bestehen aus
  - **Dateien** (*Files*) zur Speicherung von ausführbaren Programmen und Daten (Quelltexte, Dokumente, Bilder, ...) und
  - **Verzeichnissen** (*Directories*) zur hierarchischen Strukturierung von Dateien
  - **Verknüpfungen** (*Links*)
  - **Spezialdateien** (*Special Files*)

6 Dateisysteme (2)

- eine **Partition** ist eine Menge von Verzeichnissen und deren Dateien auf einem Teil eines Datenträgers
  - dient zum Trennen von Teilbereichen einer Festplatte oder CD (⇒ die Zylinder einer Festplatte können in mehrere unabhängige Partitionen eingeteilt werden, z.B. für Betriebssystem und Benutzerdateien)
  - jede Partition erhält ihr eigenes Dateisystem
  - einige Dateisysteme gestatten das **Montieren** (Einhängen) weiterer Partitionen (z.B. von anderen Geräten)
- **Anforderungen** an ein Dateisystem:
  - persistente Speicherung von Daten in Dateien
  - schneller Zugriff auf Dateien
  - gleichzeitiger Zugriff durch mehrere Prozesse
  - Implementierung von Schutzrechten

6 Dateisysteme (3)

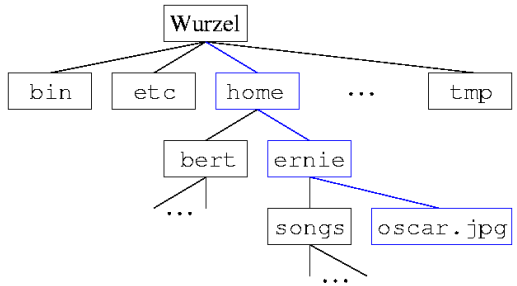
- typische **Attribute** von Dateien:
  - **Symbolischer Name**, vom Benutzer les- und interpretierbar (i.a. bestehend aus Bezeichnung + Erweiterung, z.B. `loesung1.pdf`)
  - **Typ**, z.B. zeichenorientierte Datei (für sequentiellen Zugriff), satzorientierte Datei (für wahlfreien Zugriff), ausführbare Binärdatei, Spezialdatei (für E/A-Gerät), ...
  - **Größe** (in Bytes oder Dateiblöcken)
  - Identifikation des **Eigentümers** (*User ID*)
  - **Zugriffsrechte**
  - **Zeitstempel**, z.B. der Erstellung und der letzten Modifikation (wichtig für inkrementelles Backup und Entwicklungswerkzeuge)
  - **Ortsinformation**, d.h. Ort der physikalischen Speicherung (z.B. Sektoren auf einer Festplatte)

6 Dateisysteme (4)

- typische **Operationen** auf Dateien:
  - **Erzeugen** (*create*): Eintrag für leere Datei im Verzeichnis erstellen und Attribute setzen
  - **Löschen** (*delete*): Eintrag für Datei im Verzeichnis löschen und Freigeben des Speicherplatzes auf Datenträger
  - **Öffnen** (*open*): Zugriff durch Prozess und Laden der Attribute
  - **Schließen** (*close*): Freigabe durch Prozess
  - **Lesen** (*read*): Lesen an aktueller Position eines Dateizeigers
  - **Schreiben** (*write*): dito, für Schreiben
  - **Anfügen** (*append*): Schreiben neuer Daten am Ende einer Datei
  - **Positionieren** (*seek*): Dateizeiger auf bestimmte Stelle setzen

6 Dateisysteme (5)

- ein **Verzeichnis** enthält Namen von Dateien und ggf. Namen von Unterverzeichnissen
  - auch als Ordner (*Folder*) bezeichnet
  - Benutzer kann Dateien in logischen Gruppen arrangieren
  - Aufbau einer hierarchischen Verzeichnis-Struktur unter Berücksichtigung mehrerer Benutzer:



  - ermöglicht einfaches Auffinden von Dateien durch Benutzer

6 Dateisysteme (6)

- Zugriff i.a. über **Pfadnamen**
  - Namen aller Verzeichnisse auf dem Pfad von der Wurzel bis zur Datei, durch spezielles Trennzeichen separiert
  - Unix/Linux: `/home/ernie/oscar.jpg`
  - Windows: `C:\home\ernie\oscar.jpg`
- typische **Operationen** auf Verzeichnissen:
  - **Erzeugen** (*mkdir*)
  - **Löschen** (*rmdir*), i.a. nur von leeren Verzeichnissen möglich
  - **Öffnen** (*opendir*) und **Schließen** (*closedir*)
  - **Lesen** eines **Verzeichniseintrags** (*readdir*)
  - Schreiben eines neuen Verzeichniseintrags erfolgt i.a. implizit beim Erzeugen einer neuen Datei
  - Löschen eines Verzeichniseintrags erfolgt i.a. implizit beim Löschen einer Datei bzw. eines Unterverzeichnisses

6.1 Linux-Dateisystem EXT2

- EXT2 ist seit 1992 das Standard-Dateisystem für Linux
  (heute i.a. ersetzt durch Nachfolger EXT3 mit Journaling)
- arbeitet auf Blöcken der Größe 1 kByte, 2 kByte oder 4 kByte
  (beim Anlegen des Dateisystems durch `mke2fs` als Parameter wählbar)
- **Datei** in EXT2:
  - unstrukturierte Bytefolge mit beliebigem Inhalt
  - Zugriffsrechte: lesbar (`r`), schreibbar (`w`), ausführbar (`x`), separat für Eigentümer, Gruppe und alle anderen Nutzer setzbar
- **Verzeichnis** in EXT2:
  - jedem Prozess ist ein aktuelles Verzeichnis zugeordnet (*cwd* = *current working directory*) zugeordnet
  - Zugriffsrechte: lesbar (`r`), schreibbar (`w`), durchsuchbar (`x`), separat für Eigentümer, Gruppe und alle anderen Nutzer setzbar

6.1 Linux-Dateisystem EXT2 (2)

- **Eigentümer** und **Zugriffsrechte**:
  - jeder Benutzer wird durch eine eindeutige *User ID* (UID) repräsentiert
  - UID 0 ist reserviert für Supervisor („`root`“)
  - jeder Benutzer gehört einer oder mehrerer Gruppen an, die durch eine eindeutige Nummer (GID, *Group ID*) repräsentiert werden
  - jede Datei und jedes Verzeichnis ist genau einer UID und einer GID zugeordnet
  - nur Eigentümer darf Rechte zum Lesen (`r`), Schreiben (`w`) und Ausführen/Durchsuchen (`x`) von Dateien/Verzeichnissen ändern
  - Rechte für Eigentümer, Gruppenangehörige, und alle anderen Nutzer können separat gewählt werden
  - Ändern der Zugriffsrechte erfolgt mit Unix-Kommando `chmod`
    Beispiel: `chmod g+w,o+r myfile`

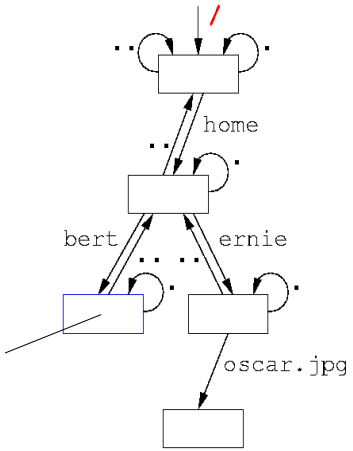
6.1 Linux-Dateisystem EXT2 (3)

- alle Attribute einer **Datei** werden in einem **Inode** (*Index node*) gespeichert:
  - jeder Inode hat eine eindeutige **Inode-Nummer** und belegt 128 Byte
  - Inodes sind für jede Partition getrennt durchnumeriert
  - Inhalt eines Inodes:
    - Dateityp (*mode*), z.B. einfache Datei, Verzeichnis, Spezialdatei
    - Eigentümer (UID) und Gruppenzugehörigkeit (GID)
    - Zugriffsrechte
    - Zugriffszeiten: letzte Änderung (*mtime*), letzter Zugriff (*atime*), letzte Änderung des Inodes (*ctime*)
    - Anzahl der *Hard Links* auf diesen Inode
    - Dateigröße in Bytes
    - Adressen der zugehörigen Dateiblöcke:
      12 direkte Adressen und 3 indirekte Adressen (einfach, doppelt und dreifach) mit jeweils 32 Bit pro Blockadresse

  (vgl. `struct ext3_inode.h` in `/usr/include/linux/ext3_fs.h`)

6.1 Linux-Dateisystem EXT2 (4)

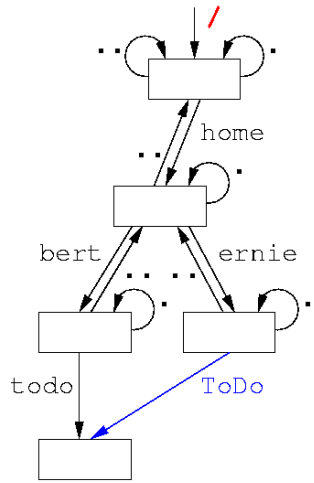
- **Dateibaum** in EXT2:
  - Wurzelverzeichnis ist „**/**“
  - nicht Dateien und Verzeichnisse, sondern die Verbindungen zwischen ihnen sind benannt
  - jedes Verzeichnis hat einen Verweis auf sich selbst („`.`“) und auf das übergeordnete Verzeichnis („`..`“)
  - Pfadnamen mit Trennzeichen „/“ (*Slash*)
    - **absoluter** Pfad von **Wurzel** aus, z.B. `/home/ernie/oscar.jpg`
    - **relativer** Pfad vom aktuellen Verzeichnis aus, z.B. `../ernie/oscar.jpg`
  - Dateien und Verzeichnisse sind somit über mehrere Pfadnamen ansprechbar



6.1 Linux-Dateisystem EXT2 (5)

- zwei Arten von **Verknüpfungen** (*Links*):

  1) Feste Verknüpfung (*Hard Link*)
  - Dateien können mehrere auf sich zeigende Verweise haben
    (auch aus verschiedenen Verzeichnissen)
  - erzeugbar durch Unix-Kommando `ln`
    Beispiel (*cwd* sei `/home/ernie`):
    `ln ../bert/todo ToDo`
  - Datei wird erst gelöscht, wenn die letzte feste Verknüpfung gelöscht wird
  - nur bei Dateien erlaubt!
  - nur innerhalb einer Partition möglich!



6.1 Linux-Dateisystem EXT2 (6)

2) Symbolische Verknüpfung (*Symbolic Link*)

- statt eines festen Verweises auf den Inode der Zieldatei wird hier dessen **symbolischer Pfadname** als Verknüpfung gespeichert
- benötigt jeweils einen Inode
- erzeugbar durch Unix-Kommando `ln -s`

  Beispiel: `ln -s /home/bert/todo /home/ernie/ToDo`
- bei kurzen Pfadnamen (bis zu 60 Zeichen) kann dieser direkt im Inode des gespeichert werden (*Fast Symbolic Link*), ansonsten ist zusätzlicher Plattenblock erforderlich
- geringe Geschwindigkeit, da Pfadnamen erst interpretiert werden müssen
- höhere Transparenz als bei festen Verknüpfungen
- auch zwischen Partitionen möglich!
- auch bei Verzeichnissen erlaubt!

6.1 Linux-Dateisystem EXT2 (7)

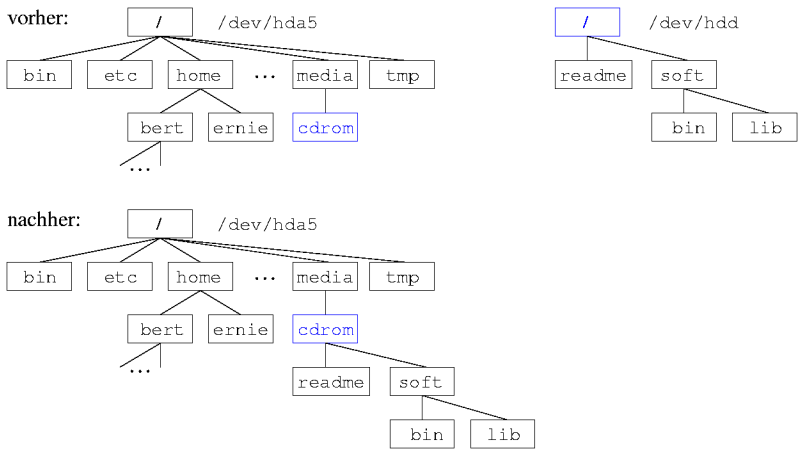
- Linux-Dateibaum kann aus mehreren **Partitionen** zusammen-montiert werden
  - jede Partition hat ihr eigenes Dateisystem
  - eine Partition wird durch blockorientierte Spezialdatei repräsentiert

    Beispiel: `/dev/hda7` oder `/dev/fd0`
  - Einhängen (Montieren) einer neuen Partition erfolgt durch privilegierten Linux-Befehl `mount`

    Beispiel: `mount /dev/hda7 /data`
  - Entfernen erfolgt durch privilegierten Linux-Befehl `umount`
  - das *Root File System* (mit Wurzel „/“) stellt das Wurzelverzeichnis des Gesamtsystems dar

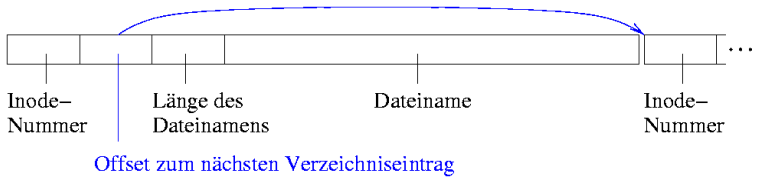
6.1 Linux-Dateisystem EXT2 (8)

- Beispiel: Montieren einer CD-ROM in ein Dateisystem mittels des Unix-Kommandos `mount /dev/hdd /media/cdrom`



6.1 Linux-Dateisystem EXT2 (9)

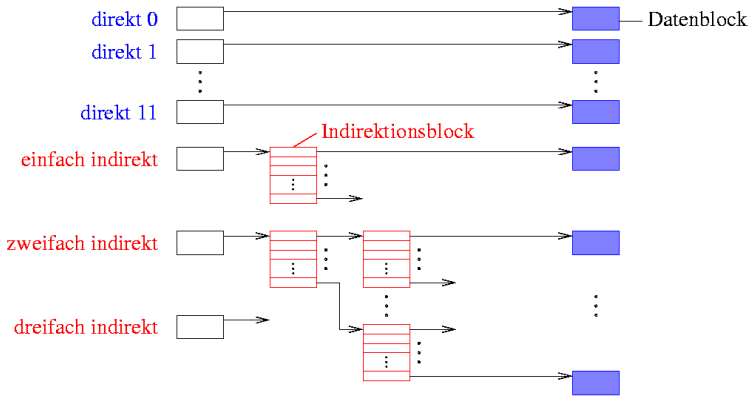
- Aufbau eines **Verzeichnisses** in EXT2:
  - Speicherung von (unsortierten) verketteten Einträgen in einer speziellen Datei, die aus einem oder mehreren Plattenblöcken bestehen kann
  - jeder Eintrag variabler Länge enthält:
    - Inode-Nummer
    - Offset zum nächsten Verzeichniseintrag (in Byte)
    - Länge des Dateinamen (in Byte)
    - Dateinamen (bis zu 255 Zeichen, abgeschlossen mit einem Nullbyte)



  - Wurzelverzeichnis hat i.a. die Inode-Nummer 2

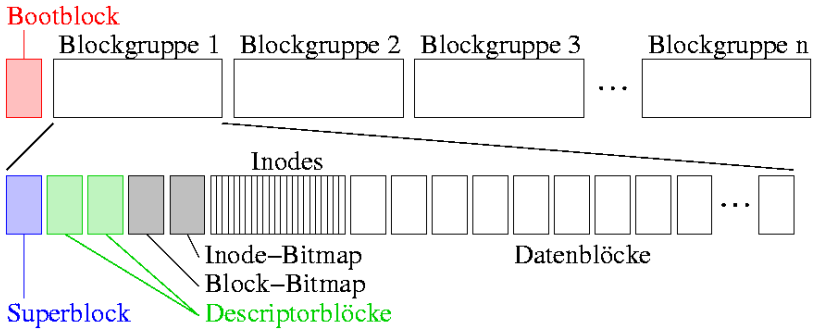
6.1 Linux-Dateisystem EXT2 (10)

- Möglichkeiten zur **Adressierung** der Dateiblöcke in EXT2:
  - 12 **direkte** Blockadressen in jedem Inode
  - je 1 einfach, zweifach und dreifach **indirekte** Blockadresse in jedem Inode



6.1 Linux-Dateisystem EXT2 (11)

- Blockorganisation in einer Partition:



  - Aufteilung in mehrere gleich große **Blockgruppen**
    (zur Reduktion der Distanz zwischen Inodes und zugehörigen Datenblöcken
    ⇒ schnellerer Plattenzugriff)

6.1 Linux-Dateisystem EXT2 (12)

  - **Bootblock** enthält den *Bootloader* (zum Start des Betriebssystems)
  - **Superblock** enthält wichtige Angaben über Layout des Dateisystems, z.B. Anzahl der Inodes, Anzahl der Plattenblöcke, Blockgröße
    (ist zur Sicherheit in jeder Blockgruppe als Kopie vorhanden!)
  - **Gruppendeskriptoren** enthalten u.a. Informationen über Position der Bitmaps für Blöcke und Inodes sowie über Anzahl freier Blöcke und freier Inodes in jeder Blockgruppe
    (Anzahl der Inodes wird beim Erzeugen des Dateisystems festgelegt)
  - das **Block-Bitmap** belegt einen Block und kennzeichnet die freien Blöcke in einer Blockgruppe (max. 8192 Einträge bei 1 kByte Blöcken)
  - das **Inode-Bitmap** belegt einen Block und kennzeichnet die freien Inodes in einer Blockgruppe (max. 8192 Einträge bei 1 kByte Blöcken)
  - sehr große Dateien sind über mehrere Blockgruppen verteilt!

6.1 Linux-Dateisystem EXT2 (13)

- einige weitere Eigenschaften:
  - eine wählbare Anzahl von Dateiblöcken (i.a. 5 %) bleibt für Superuser reserviert
    (⇒ zur Erhaltung der Systemstabilität bei einer voll geschriebenen Platte)
  - **Konsistenz** des Dateisystems kann mittels `fsck` überprüft werden; in vielen Fällen ist bei Inkonsistenz eine automatische Reparatur möglich
  - Flag im Superblock gibt **Status** (*clean* / *not clean*) des Dateisystems an
    (⇒ erspart extrem zeitaufwendigen Konsistenzcheck beim Booten)
  - EXT3 besitzt zusätzlich ein **Journaling**:
    - in einer *Journal*-Datei werden sämtliche Änderungen am Dateisystem **vor** ihrer Durchführung protokolliert (und nach erfolgreicher Durchführung wieder entfernt)
    - nach einem Systemabsturz müssen nur die in der *Journal*-Datei enthaltenen Änderungen nachvollzogen und ggf. die Konsistenz der betroffenen Dateien überprüft werden