

1. Filesysteme

- Linux-Dateisystem EXT2
- FAT32 Dateisystem
- NTFS Dateisystem

2. Datenträger

- ~~– Floppy, CD, DVD~~
- USB
- ~~– SSD~~
- RAID

3. PC Bussysteme

- EXT2 ist seit 1992 das Standard-Dateisystem für Linux (heute i.a. ersetzt durch Nachfolger EXT3, EXT4 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

- **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`

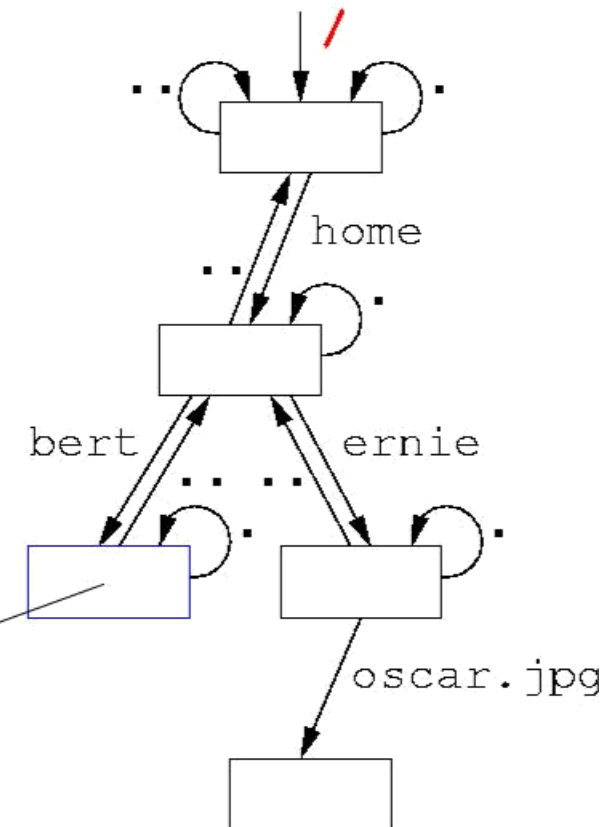
Linux-Dateisystem EXT2 (3)

- 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 durchnummeriert
 - 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`)

Linux-Dateisystem EXT2 (4)

- **Dateibaum** in EXT2:

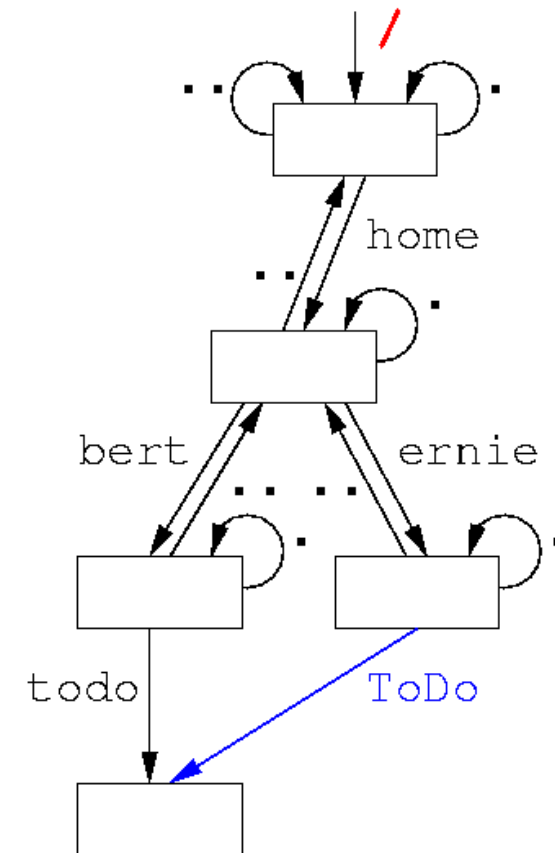
- Wurzelverzeichnis ist „/“
- nicht Dateien und Verzeichnisse, sondern die Verbindungen zwischen ihnen sind benannt
- jedes Verzeichnis hat einen Verweis auf sich selbst („.“) und auf das übergeordnete Verzeichnis („..“)
- Pfadnamen mit Trennzeichen „/“ (*Slash*)
 - **absoluter** Pfad von **Wurzel** 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



- 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!



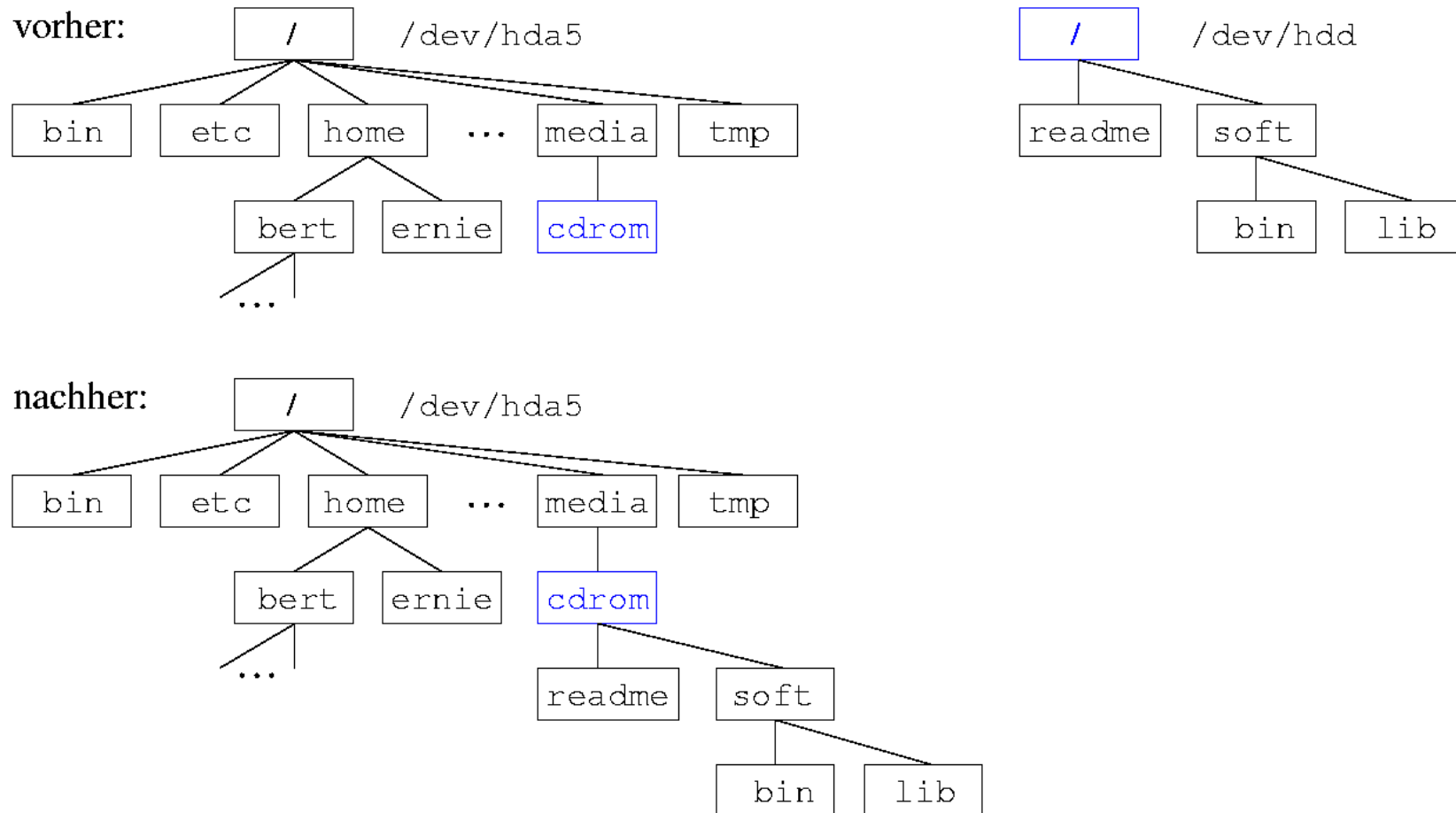
2) Symbolische Verknüpfung (*Symbolic Link*)

- 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!

- Linux-Dateibaum kann aus mehreren **Partitionen** zusammenmontiert 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

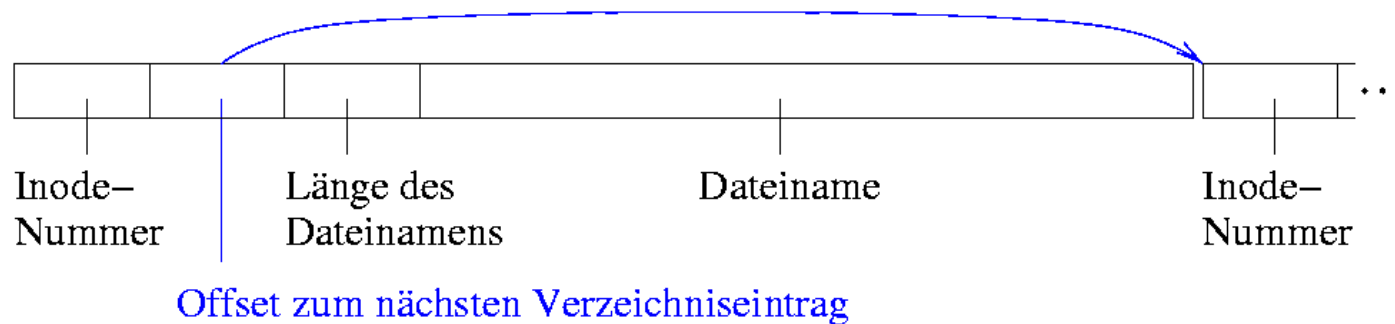
Linux-Dateisystem EXT2 (8)

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



Linux-Dateisystem EXT2 (9)

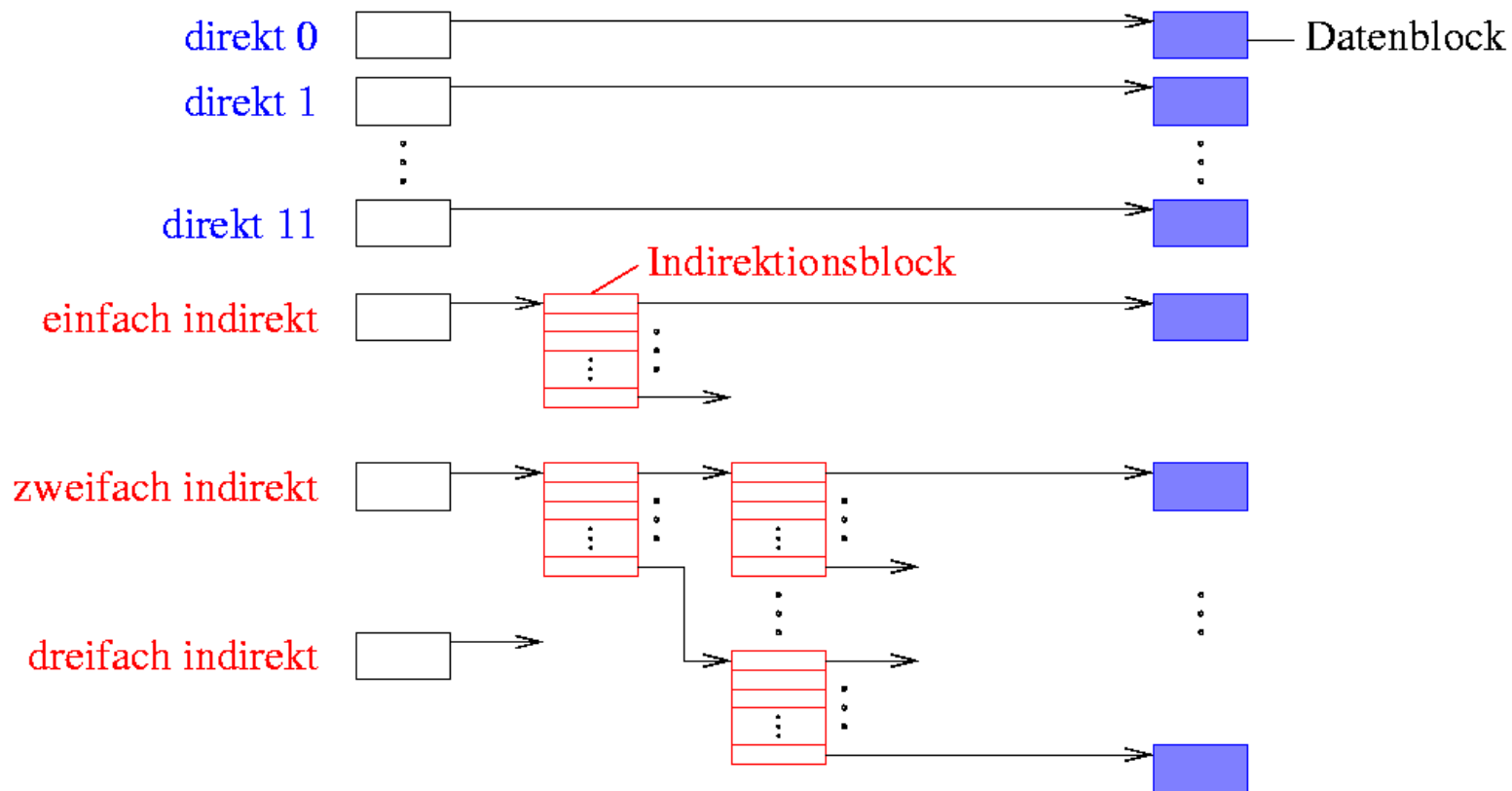
- Aufbau eines **Verzeichnisses** in EXT2:
 - Speicherung von (unsortierten) verketteten Einträgen in einer speziellen Datei, die aus einem oder mehreren Plattenblöcken bestehen kann
 - jeder Eintrag variabler Länge enthält:
 - Inode-Nummer
 - Offset zum nächsten Verzeichniseintrag (in Byte)
 - Länge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



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

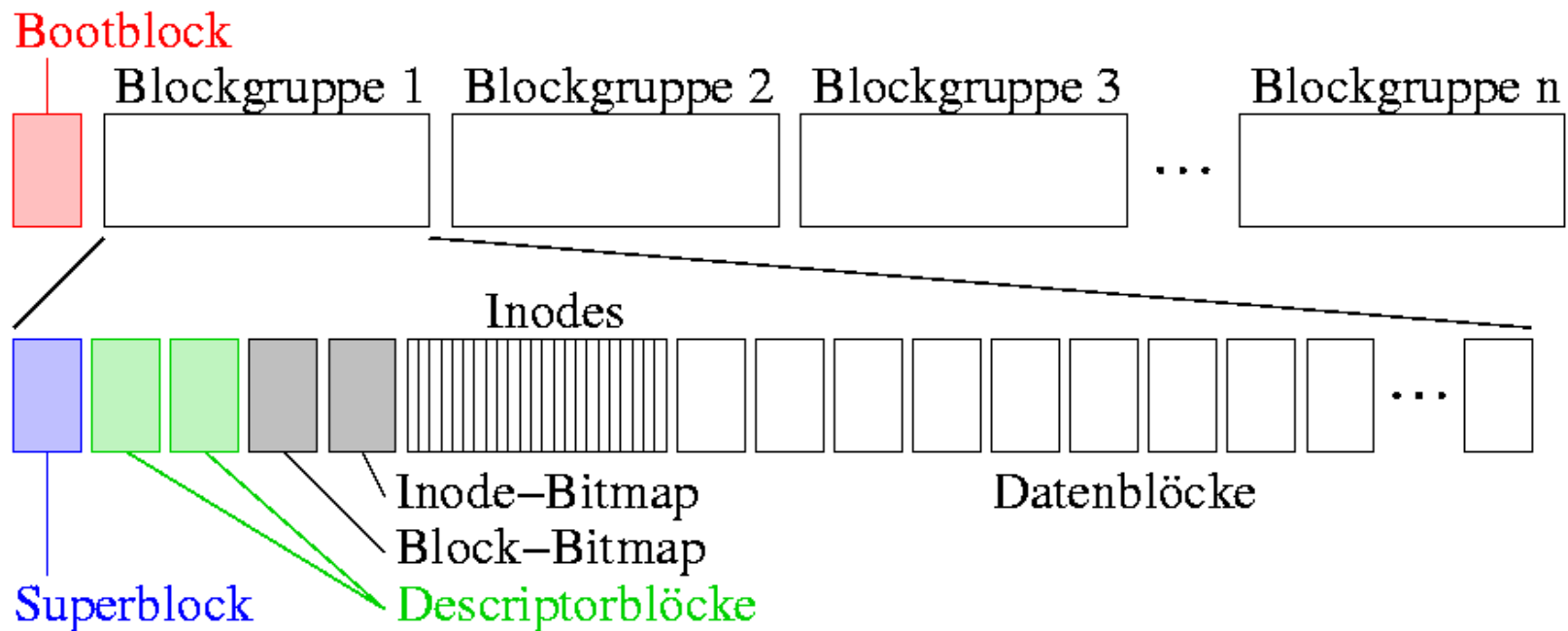
Linux-Dateisystem EXT2 (10)

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



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)

- **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!

- 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

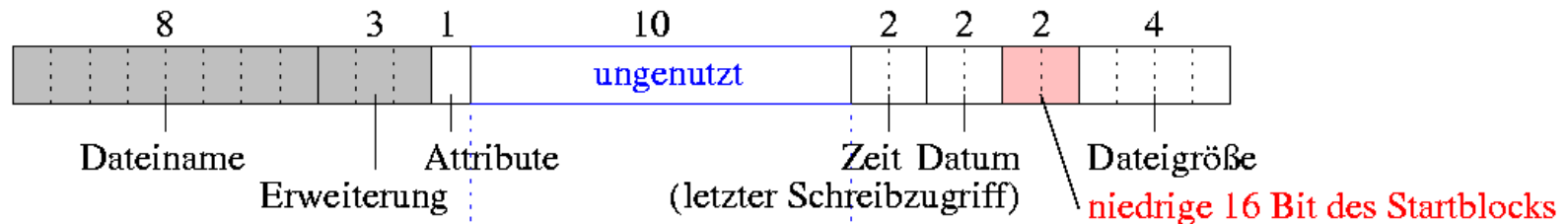
- Dateisystem für **Windows 98**
- einige Unterschiede zum Linux-Dateisystem EXT2:
 - **keine Benutzeridentifikation** für Dateien und Verzeichnisse!
 - Partitionen werden durch Laufwerke repräsentiert, die durch Buchstaben dargestellt werden, z.B.: A: (Floppy), C: (Platte), D: (DVD)
 - jedem Windows-Programm ist ein **aktuelles Laufwerk** und ein **aktuelles Verzeichnis** aus Dateibaum zugeordnet
 - es gibt keine Inodes: die Speicherung aller Attribute einer Datei erfolgt im Verzeichnis
 - es gibt keine *Hard Links*
 - die kleinste adressierbare Einheit heißt **Cluster** und ist ein Block mit einer Zweierpotenz von 1 bis 128 Sektoren (bei Formatierung wählbar)
 - die Blockadressierung erfolgt über eine Tabelle (**FAT** = *File Allocation Table*), in der die Verkettung der Cluster aller Dateien gespeichert ist

FAT32 Dateisystem (2)

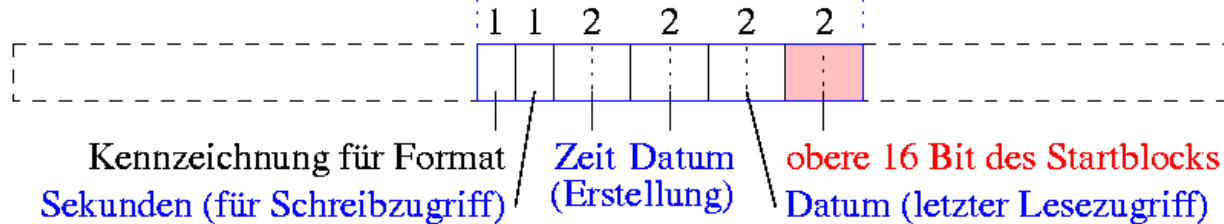
- Attribute einer **FAT32-Datei**:
 - **Name**
 - im MS-DOS Modus: 8 Zeichen Name + 3 Zeichen Erweiterung (z.B.: „AUTOEXEC.BAT“)
 - im Windows 98 Modus: 255 Zeichen inklusive Sonderzeichen (z.B.: „Eigene Dateien“)
 - **Dateilänge**
 - **Typ**: Verzeichnis, versteckte Datei (*hidden*), Systemdatei (*system*), zu archivierende Datei (*archive*)
 - nur **zwei Zugriffsrechte**: „nur lesbar“ und „schreib- und lesbar“
 - **Ortsinformation**: Nummer des ersten Clusters einer Datei
 - **Zeitstempel**:
 - zunächst nur Datum und Uhrzeit des letzten Schreibzugriffs
 - bei Windows 98 zusätzlich Datum und Uhrzeit der Erstellung sowie Datum des letzten Lesezugriffs

FAT32 Dateisystem (3)

- Aufbau eines **FAT32-Verzeichnis**:
 - unsortierte 32-Byte Einträge werden hintereinander in Liste gespeichert
 - ursprünglicher Eintrag (MS-DOS mit FAT16):



erweiterter Eintrag (Windows 98 mit FAT32):



- aus langen Dateinamen (bis zu 255 Zeichen) wird ein neuer eindeutiger Name aus 8+3 Zeichen generiert und eingetragen; der vollständige Name wird in zusätzlichen vorangestellten 32-Byte Feldern gespeichert

FAT32 Dateisystem (4)

- die **Verkettung** der Cluster wird in der **FAT** festgehalten:
 - enthält Eintrag für jedes Cluster auf der Festplatte
 - für jedes Cluster einer Datei ist die Nummer des nachfolgenden Clusters als 32-Bit Zahl eingetragen
 - die Nummer des Startblocks kann dem Verzeichnis entnommen werden
 - Dateiende wird durch Eintrag -1 markiert
 - freie Cluster werden durch Eintrag 0 markiert

Auszug aus einer FAT:

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	...
-	-	0	8	5	6	20	0	9	15	11	17	0	0	0	16	18	4	-1	0	-1	0	...

Plattenblöcke (Cluster) für Datei A:

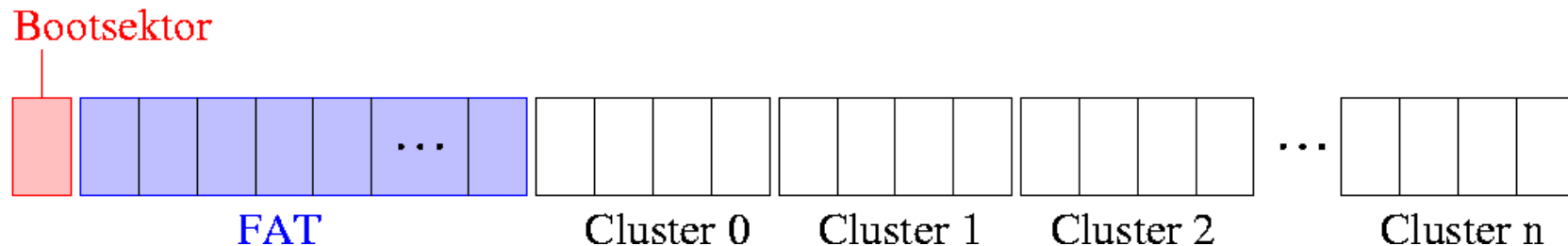
10	11	17	4	5	6	20
----	----	----	---	---	---	----

Plattenblöcke (Cluster) für Datei B:

3	8	9	15	16	18
---	---	---	----	----	----

FAT32 Dateisystem (5)

- Blockorganisation einer Partition mit FAT32:



- **Bootsektor** enthält neben dem *Bootloader* noch einige Angaben über das Dateisystem, z.B:
 - Gesamtanzahl der Sektoren (4 Byte)
 - Bytes je Sektor (2 Byte)
 - Sektoren je Cluster (1 Byte, nur Zweierpotenzen von 1 bis 128 erlaubt)
 - Startposition des Hauptverzeichnisses (4 Byte)
 - Label (10 Byte) und Serien-Nummer (4 Byte)
 - Anzahl FATs (1 Byte) und Sektoren je FAT (4 Byte)
- **FAT** kann zur Erhöhung der Sicherheit auch mehrfach auf Festplatte gespeichert sein

- einige **Nachteile** von FAT32:
 - **umständliche** Datenstrukturen (wegen Kompatibilität zu MS-DOS)
 - **sehr große FAT** bei modernen Festplatten hoher Kapazität
 - Positionieren eines Dateizeigers bei großen Dateien sehr zeitaufwendig
 - für jeden Dateizugriff muss mindestens ein Plattenblock mit einem Teil der FAT von der Festplatte geladen werden
 - FAT enthält Verkettungen für alle Dateien \Rightarrow es werden stets auch viele nicht benötigte Verkettungsinformationen geladen
 - langsame Suche nach freien Clustern
 - **sehr viele Kopfbewegungen**, wenn Cluster einer Datei verstreut sind
(\Rightarrow regelmäßiger Aufruf eines Defragmentierungsprogramms sinnvoll; es versucht die Cluster jeder Datei zusammenhängend anzuordnen)
- FAT32 wird nicht mehr weiterentwickelt!

- Dateisystem für **Windows NT**, optional auch für **Windows XP** und **Nachfolger**
- einige Unterschiede zum FAT32 Dateisystem:
 - Unterstützung mehrerer **Benutzer** und **Gruppen** mit umfangreichen Zugriffsrechten
 - jede Partition wird als **Volume** bezeichnet und besteht aus einer linearen Sequenz von **Clustern** (mit z. Zt. 512, 1024, 2048 oder 4096 Byte)
 - Adressierung eines Clusters erfolgt über **64-Bit Cluster-Nummern** (⇒ sehr große Dateien möglich)
 - zentrales Element der Dateiorganisation ist die *Master File Table* (**MFT**), die für jede Datei einen Eintrag enthält
 - Unterstützung von *Hard Links*
 - Dateien können automatisch komprimiert abgespeichert werden
 - Konsistenzüberprüfung mittels Journal-Datei

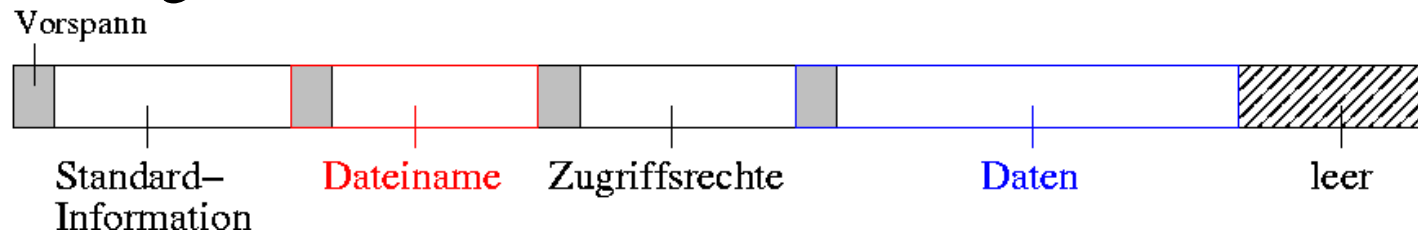
NTFS Dateisystem (2)

- Zugriffsrechte einer **NTFS-Datei**:
 - *no access* : kein Zugriff (---)
 - *list* : Anzeige von Verzeichnisinhalt erlaubt (r--)
 - *read* : Lesen und Ausführen von Dateien erlaubt (rw-)
 - *add* : Hinzufügen von Einträgen in einem Verzeichnis erlaubt (-wx)
 - *change* : Ändern und Löschen von Dateien erlaubt (rwx)
 - *full* : zusätzlich Ändern von Eigentümer und Zugriffsrechten erlaubt
- jede Datei wird eindeutig durch eine **64-Bit Dateireferenz** (*File reference*) bezeichnet; sie besteht aus:
 - **48-Bit Dateinummer** (*File ID*), die einen eindeutigen Index in der MFT darstellt
 - **16-Bit Folgenummer** (*Sequence ID*), die bei jeder Wiederverwendung der Dateinummer hochgezählt wird

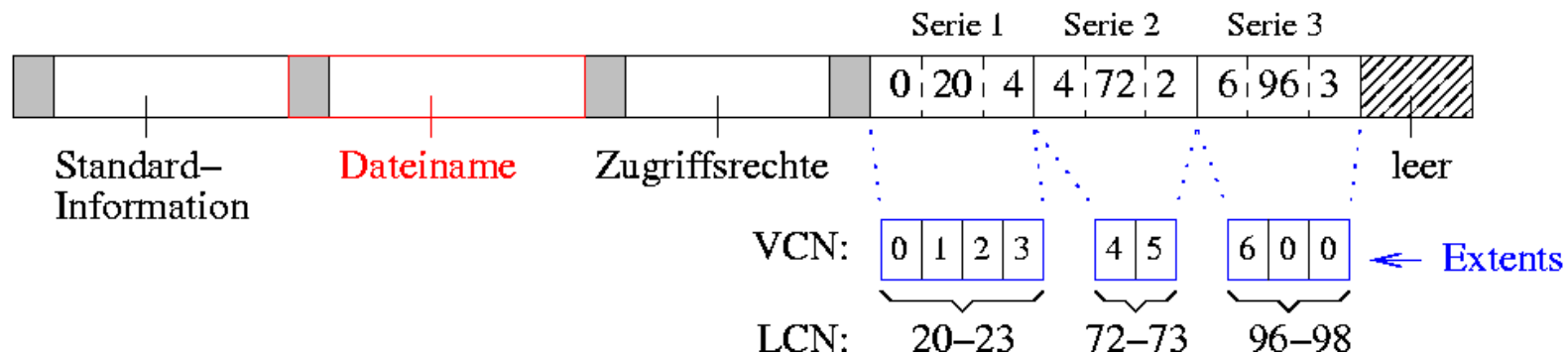
- jede Datei besteht aus mehreren **Strömen**, z.B.:
 - **Standard-Information** (zu MS-DOS kompatibler Dateiname sowie klassische MS-DOS Attribute wie Dateilänge, Zeitstempel, Typ, ...)
 - **Dateiname** (in Unicode mit 16-Bit Zeichen)
 - **Dateireferenz** (64-Bit Wert)
 - **Sicherheits-Beschreibung** (enthält **Eigentümer** und **Zugriffsrechte**)
 - **eigentliche Daten**
- Dateiorganisation erfolgt mittels *Master File Table* (**MFT**):
 - enthält **für jede Datei genau einen Eintrag**
 - Größe jedes Eintrags entspricht der Cluster-Größe
 - **Index** in Tabelle wird durch die **Datei-Nummer** festgelegt
 - Eintrag in Bootsektor verweist auf Beginn der MFT
 - ein Eintrag besteht aus **Hintereinanderreihung mehrerer Ströme**, die jeweils durch einen kurzen Vorspann (mit Länge, ...) eingeleitet werden

NTFS Dateisystem (4)

- Eintrag in MFT für eine **kurze Datei**:



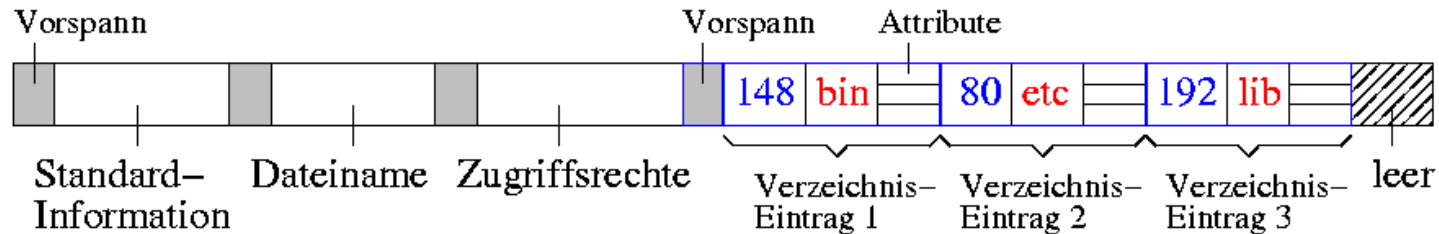
- Eintrag in MFT für eine lange Datei (Beispiel):



- Daten befinden sich in Serien aus zusammengehörigen Clustern (**Extents**)
- Zuordnung von virtuellen Cluster-Nummern (VCN) zu logischen Cluster-Nummern (LCN) wird als weiterer Strom gespeichert

NTFS Dateisystem (5)

- Eintrag in MFT für ein **kurzes Verzeichnis** (Beispiel):



- Inhalt eines Verzeichnisses wird als eigener Strom gespeichert
- jeder Verzeichniseintrag enthält **Dateireferenz**, **Dateiname** und einige ausgewählte **Attribute** (z.B. Dateilänge, Datum der letzten Modifikation)
- Sortierung in lexikographischer Reihenfolge
- Eintrag in MFT für ein **langes Verzeichnis**:
 - Verzeichniseinträge werden nicht in MFT, sondern in separaten **Extents** gespeichert
 - Organisation als **B⁺-Baum** ermöglicht eine schnelle Suche in großen Verzeichnissen

NTFS Dateisystem (6)

- die ersten 16 Dateien in der MFT sind **Metadateien**, die für das System reserviert sind:

Index	Bedeutung
0	MFT
1	Kopie der MFT
2	Journal-Datei (protokolliert die Änderungen am Dateisystem)
3	Volume-Informationen (z.B. Name, Größe des Volumes)
4	Attribut-Tabelle (definiert erlaubte Ströme in den Einträgen)
5	Wurzelverzeichnis
6	Cluster-Bitmap (kennzeichnet alle freien und belegten Cluster)
7	<i>Bootloader</i>
8	<i>Bad Cluster List</i> (enthält die Indizes aller fehlerhaften Cluster)
9 bis 15	... (reserviert für weitere Systemdateien)
16	erste Benutzerdatei

Fallstudie: USB

- **Universal Serial Bus**, spezifiziert vom USB Implementers Forum (www.usb.org)
- Ziel: preiswerter, einheitlicher und einfacher Anschluß diverser E/A-Geräte
- **serieller, asynchroner Peripherie-Bus**
 - 4-adriges Kabel: Vcc (Stromversorgung 5V, max. 0.5A), GND, D+, D- (Pegel 3.3V)
 - unterschiedliche Stecker für Host (USB-A) und E/A-Gerät (USB-B)
- **USB1.1** (1995)
 - Transferraten: 1.5 MBit/s (*low speed*)
oder 12 MBit/s (*full speed*)
- **USB2.0** (2001)
 - weitere Transferrate: 480 MBit/s (*high speed*)
- **USB3.0** (2008))
 - höhere Transferrate: 5 GBit/s (*SuperSpeed-Modus*)



Beschriftung:



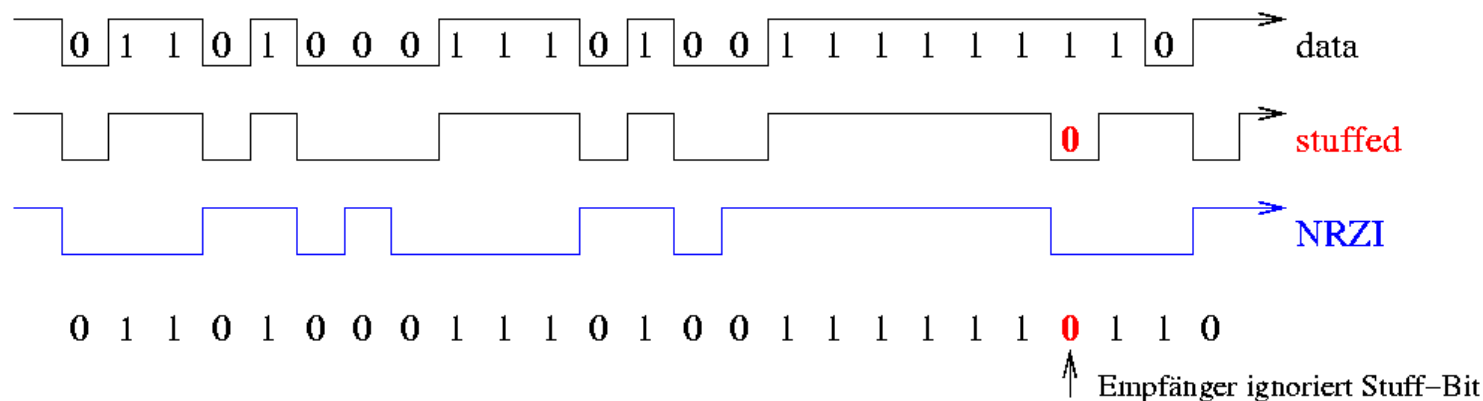
- **hierarchischer Aufbau** eines USB-Bussystems:
 - ausschließlich Punkt-zu-Punkt Verbindungen
 - **USB Host** (auch **Root Hub**, mit 2 bis 4 USB Anschlüssen) ist einziger Master, fragt alle USB Geräte durch *Polling* ab
 - **USB Hub** (Verstärker, ggf. mit Anpassung der Transferrate) verteilt Signale auf mehrere USB-Anschlüsse und ermöglicht den Aufbau eines pyramidenartigen Bussystems
 - maximal **7 physikalische Ebenen** in Pyramide, logisch jedoch eine Ebene
 - insgesamt maximal **127 Buskomponenten**
 - Länge eines Kabels max. **5m** \Rightarrow insgesamt max. **35m** bei 7 Ebenen (USB-A Stecker stets zum Host, USB-B Stecker stets zum E/A-Gerät gerichtet)
 - **Autokonfiguration**: E/A-Geräte identifizieren sich selbst beim Host und erhalten eine Adresse zwischen 1 und 127 (Host hat Adresse 0)
 - Geräteanschluss im laufendem Betrieb möglich (*Hot Plugging*)
- Kopplung zweier USB Hosts ist nicht möglich!

Fallstudie: USB (3)

- Busprotokoll gestattet vier verschiedene Übertragungsarten:
 - 1) **Kontroll-Transfer**: Initialisierung und Konfiguration eines Gerätes durch USB Host
 - 2) **Interrupt-Transfer**: USB Host fragt alle E/A-Geräte ab, ob Interrupts angefordert wurden
 - 3) **Bulk-Transfer**: Senden langer Datenströme (nur bei *full / high speed*, falls ausreichende Bandbreite verfügbar)
 - 4) **Isochroner Transfer**: Übertragung von Daten mit einer garantierten Bandbreite (d.h. in Echtzeit), z.B. für Sprach- oder Videodaten (nur bei *full / high speed*)
- jeder Transfer wird vom USB Host initiiert!
- periodische Transfers (Interrupt-/Isochroner Transfer) dürfen nicht mehr als 80-90% der Busbandbreite verwenden!
- Hin- und Rückrichtung über die gleichen Leitungen!

Fallstudie: USB (4)

- **NRZI-Kodierung** (Non Return to Zero Inverted):
 - Wechsel des Leitungspiegels **nur bei Übertragung eines Null-Bit**
 - **Bit-Stuffing**: Einfügen eines Null-Bits nach jeweils 6 Eins-Bits
 - Beispiel: NRZI-Kodierung der drei Bytes 68_{16} , $E9_{16}$ und FE_{16}



- **Differentielle Signale**
 - zur Übertragung des NRZI-Signals über verdrehtes Kabelpaar D+, D-:
 - Sender: $(D+) - (D-) > 1V$ (Eins-Bit) bzw. $< -1V$ (Null-Bit)
 - Empfänger: $(D+) - (D-) > 0.2V$ (Eins-Bit) bzw. $< -0.2V$ (Null-Bit)

Fallstudie: USB (5)

- **paketorientierte Übertragung:**
 - Einteilung in Zeitabschnitte von **1 ms** Dauer, auch als *Frame* bezeichnet (⇒ 12000 Bit/Frame im *High Speed* Modus)
 - Adressierung der Endgeräte durch **7 Adress-Bits** (für 127 Geräte) und **4 EP-Bits** (für 16 verschiedene Endpunkte je Gerät, z.B. EP0 = *Control*, EP1 = *Bulk*, EP3 = *Interrupt*)
 - Kommunikation zwischen Host und Endpunkten von E/A-Geräten über logische Kanäle (*Pipes*), die einen Teil der Busbandbreite belegen
 - jede Datenübertragung innerhalb eines Frames besteht aus **drei Paketen**, wobei jedes Paket mit einer 8-Bit Typkennung beginnt:
 - 1) **Paket** mit **Richtung** und **Zieladresse** (11 Bit + 5 Bit Prüfsumme)
 - 2) **Datenpaket** (variable Länge + 16 Bit Prüfsumme)
 - 3) **Bestätigungspaket** (Handshaking) des Empfängers



Fallstudie: USB (6)

- Vergleich von USB1.1 und **USB 2.0**:

	<i>low speed</i>	<i>full speed</i>	<i>high speed</i>
Bit-Transferate	1.5 MBit/s	12 MBit/s	480 MBit/s
max. Bulk-Datenpaketgröße	—	64 Byte	512 Byte
max. Transferate	16 KByte/s	1.2 MByte/s	54 MByte/s

- USB stellt heute die Standardschnittstelle für weit über 50% aller (mittelschnellen) Peripheriegeräte dar
- **Firewire** (IEEE1394) ist sehr ähnlich zu USB:
 - entwickelt und lizenziert von Apple
 - unterstützt mehrere Busmaster (d.h. auch die direkte Kommunikation zwischen zwei E/A-Geräten ist möglich)
 - bis zu 400 MBit/s
 - erfordert aufwendigere Logik auf Host- und Peripherieseite

Fallstudie: USB (7)

- noch höhere Datenraten mit **USB 3.0**
(5 Gbit **Symbolrate**, 500 MB/s **Bruttodatenrate**)
 - höhere Frequenzen (ca. achtfach)
 - verbessertes USB-Protokoll
 - Vollduplex-Übertragung
 - weitere Signal-Adernpaare im Kabel und Stecker

- USB-3.0-Übertragungen finden nur statt, wenn *alle* drei Komponenten (Host, Kabel, Endgerät) USB-3.0-tauglich sind.
(Rückfall auf USB 2.0)

zusätzliche Pins bei USB 3.0

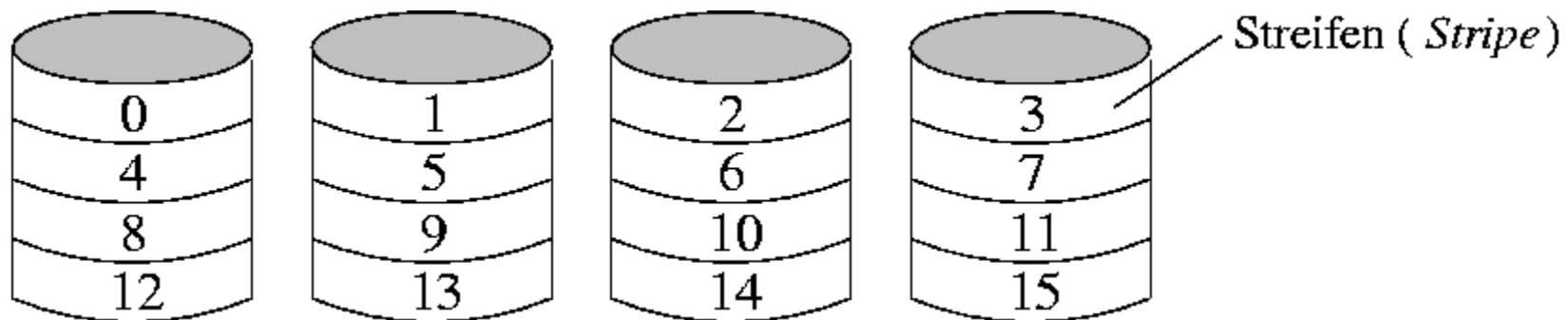
Name	Beschreibung
SSTX+	Datenübertragung vom Host zum Gerät
SSTX-	mit SSTX+ verdrillt
GND	Masse
SSRX+	Datenübertragung vom Gerät zum Host
SSRX-	mit SSRX+ verdrillt

- weitere Geschwindigkeitsverdopplung durch **USB 3.1**

- 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 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

RAID 0

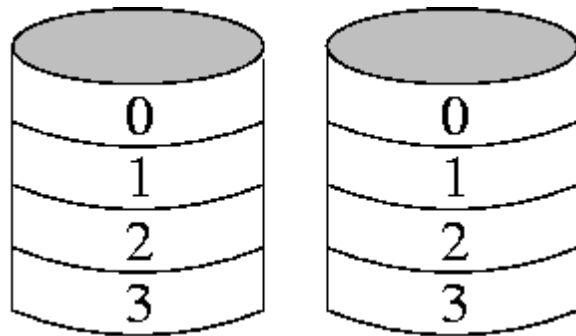
- auch als **gestreifte Platten** (*Striping*) bezeichnet
- eine logische Platte wird in kleine Streifen (*Stripes*) zerschnitten, die zyklisch über mehrere physikalische Platten verteilt werden



- **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

RAID 1

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

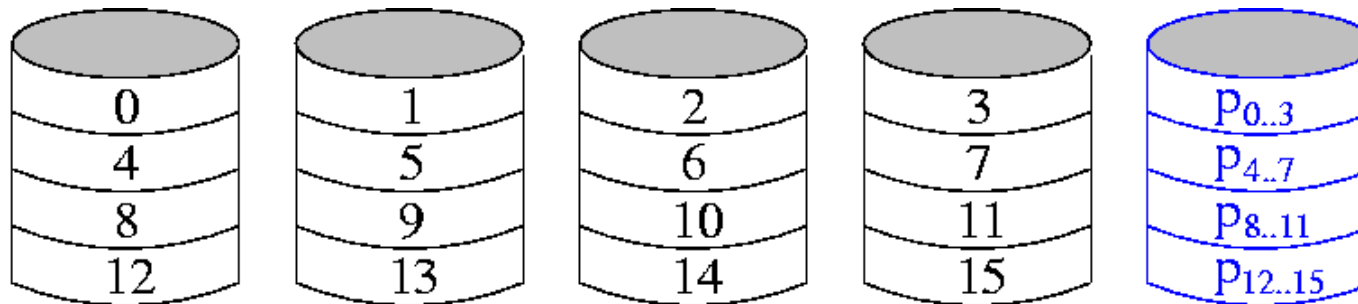


(auch mit RAID 0 kombinierbar)

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

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

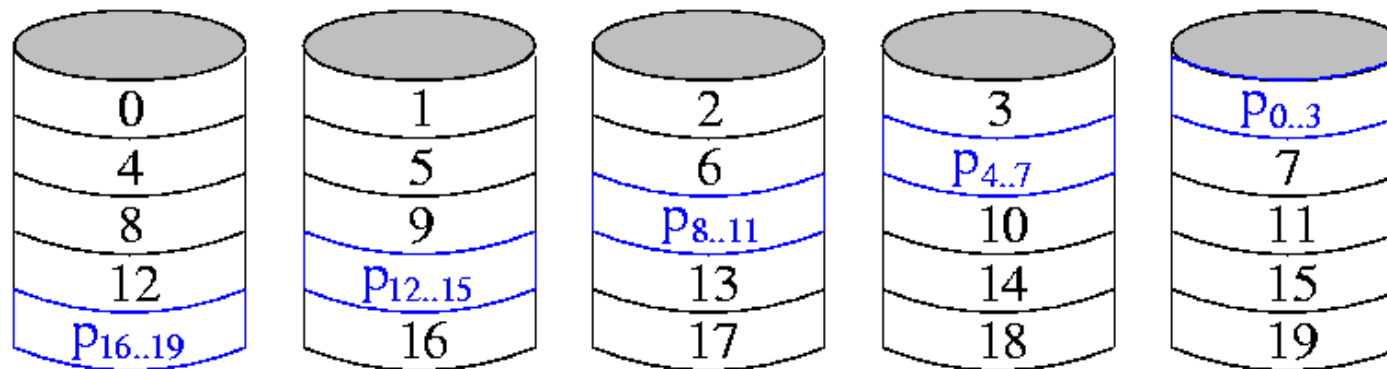
RAID 4 (2)

- **Nachteile:**

- jeder Schreibvorgang eines einzelnen Datenblocks j erfordert die Aktualisierung des zugehörigen Paritätsblocks:
$$p^{\text{neu}}(i) = p^{\text{alt}}(i) \oplus x_j^{\text{neu}}(i) \oplus x_j^{\text{alt}}(i)$$
- erfordert Lesen der alten Inhalte von Paritätsblock und Datenblock j , d.h. vier Zugriffe je Schreibvorgang!
- Paritätsplatte ist hoch belastet

RAID 5

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

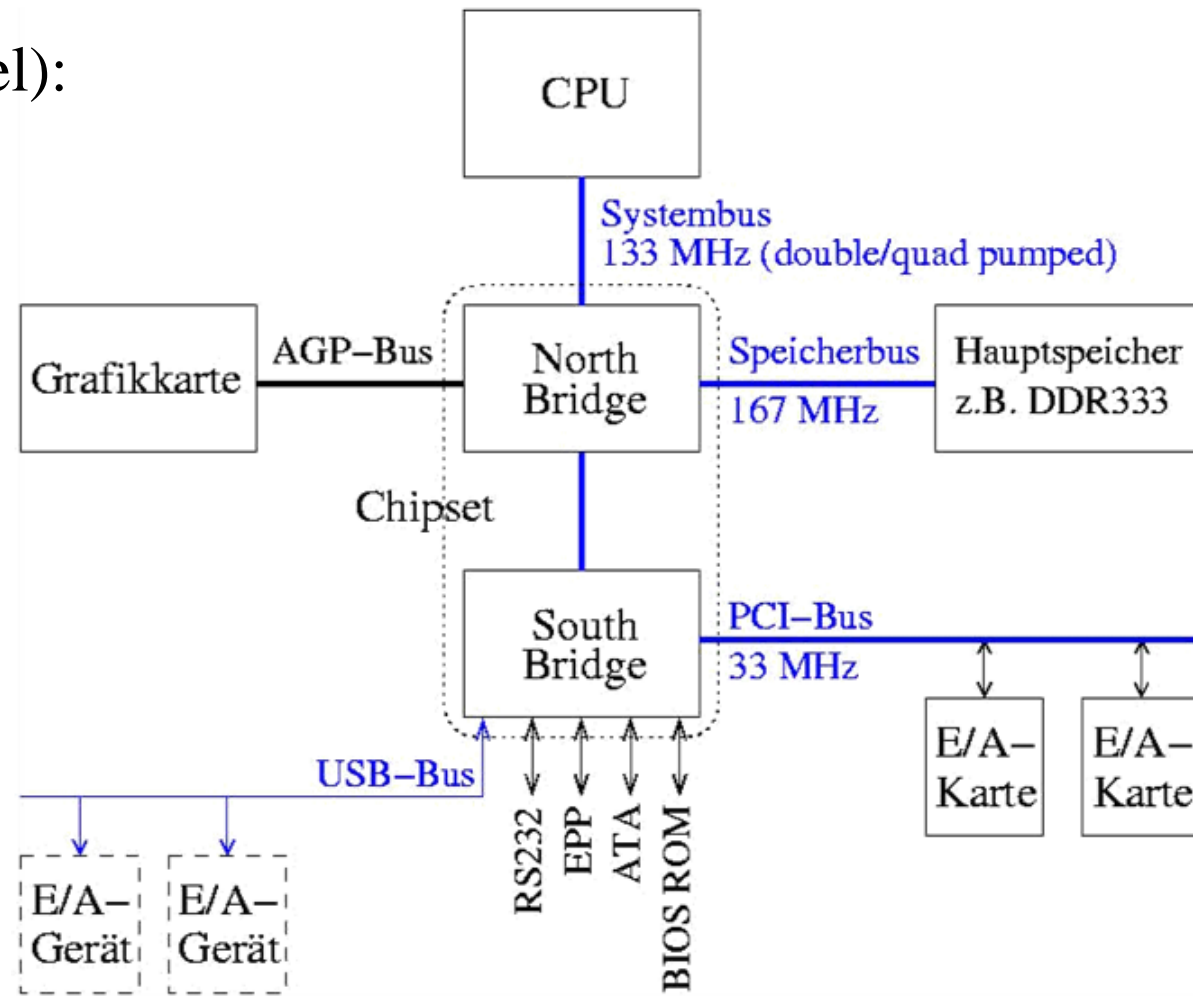


- **Vorteile** und **Nachteile:**
 - wie RAID 4, jedoch wird die zusätzliche Belastung durch Schreiben des Paritätsblocks auf alle Platten verteilt
- RAID 5 ist die heute gängigste RAID-Variante!

- Bussysteme im PC:
 - Hauptplatine enthält x86 CPU und Chipset, gekoppelt über **Systembus** (64 Bit Daten, 32 Bit Adressen, synchron, typisch 100 bis 200 MHz Taktfrequenz)
 - Datenübertragungsrate höher, da je Taktzyklus oft mehrere Datenworte übertragen werden („*double pumped*“ oder „*quad pumped*“)
 - Chipset steuert **Speicherbus** (i.a mit einer vom Systembus abweichenden Taktfrequenz)
 - Chipset enthält serielle und parallele Schnittstelle, DMA-Baustein, Plattenkontroller, Bridge für PCI-Bus, Bridge für USB, ...
 - Chipset kann mehrere CPU-Zugriffe puffern und ggf. zusammenfassen
 - schneller **AGP-Bus** (*Accelerated Graphics Port*) für Grafikkarte
 - **PCI-Bus** mit mehreren PCI-Steckplätze (*PCI-Slots*) für Soundkarte, 10/100 MBit-Netzwerkkarte, ...

Fallstudie: PC-Bussysteme (2)


- Bus-Architektur eines PC (Beispiel):



Fallstudie: PC-Bussysteme (3)

- früher: **ISA-Bus** (*Industry Standard Architecture*, 1984)
 - zuerst 8-Bit Daten, später **16-Bit Daten** und **24 Bit Adressen**
 - synchroner/asynchroner Bus mit **8 MHz** Taktfrequenz, max. 8 MByte/s
 - konzipiert als prozessornaher Systembus für 286-basierte AT PCs, Bussignale überwiegend identisch zum Prozessorbus
 - ISA-Steckverbinder:
(62+36 Pins) 
 - E/A-Adressen und Interrupts auf ISA-Buskarte über Jumper einzustellen
 - bis vor kurzem als weiterer Ein-/Ausgabebus für langsame, preiswerte E/A-Karten in PCs eingesetzt
 - die Übertragungsrate war Ende der 80er Jahre für PCs nicht mehr ausreichend; viele Alternativen wurden entwickelt:
 - **MCA** (*Microchannel Architecture*, IBM, 1987): 32-Bit Daten, 10 MHz
 - **EISA** (*Enhanced ISA*, 1989): 32-Bit Daten und Adressen, 8 MHz
 - **VLB** (*VESA Local Bus*, 1992): 32-Bit Daten, 40 MHz

Fallstudie: PC-Bussysteme (4)

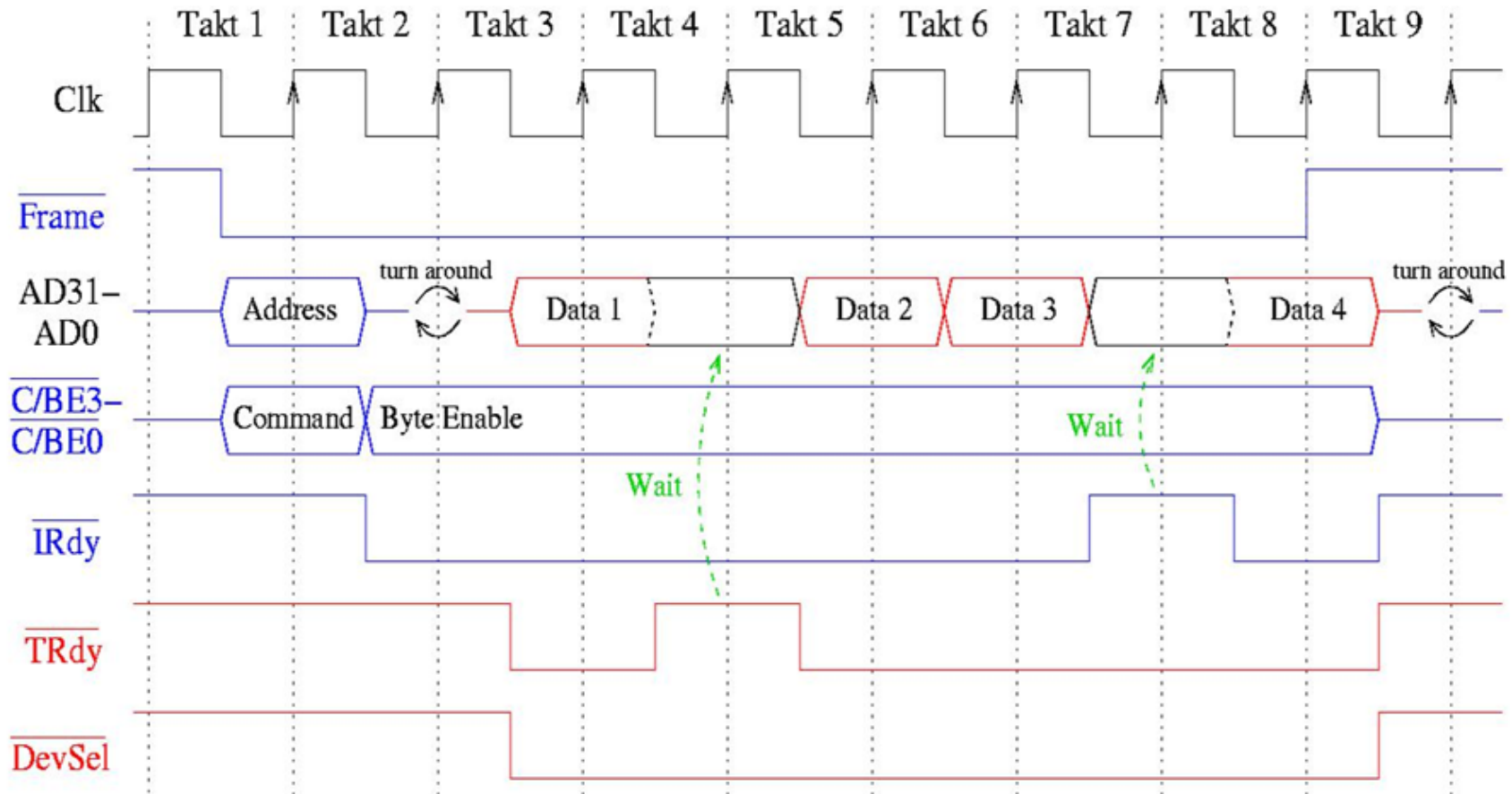
- **PCI-Bus** (*Peripheral Component Interconnect*, 1993)
 - synchroner Bus, von Intel entwickelt
 - 12 Arten von Buszyklen, u.a. auch Einzelwort- und Burst-Transfer mit beliebiger Blocklänge
 - PCI 1.0 (lange Zeit typisch): **32-Bit Daten**, bis zu **33 MHz** Bustaktfrequenz
 - PCI 2.1: auch **64-Bit Daten**, bis zu **66 MHz** Bustaktfrequenz möglich
 - theor. max. Übertragungsraten von **133 MByte/s** (32-Bit Bus, 33 MHz) bis zu **533 MByte/s** (64-Bit Bus, 66 MHz) bei Burst-Transfer
 - **Multiplexing** von Daten und **32-Bit Adressen**
 - bis zu **4 masterfähige Slots** mit zentraler Bus-Arbitrierung
 - PCI-Steckverbinder:
(124 Pins bei 32-Bit Daten) 
 - prozessorunabhängiger Bus (nicht als Systembus einsetzbar!), auch in anderen Architekturen (z.B. Ultra-Sparc, PowerPC, ...) verbreitet
 - Weiterentwicklung: **PCI-X** mit bis zu 133 MHz Taktfrequenz

Fallstudie: PC-Bussysteme (5)

- **PCI-Busleitungen** (Auswahl) für **Master** und **Slave**:
 - **Clk**: Bustakt, auf dem alle Signale synchronisiert sind
 - **AD31** bis **AD0**: gemultiplexer 32-Bit Adress-/Datenbus
 - **C/BE3** bis **C/BE0** (*Command / Byte Enable*): enthält entweder ein Bus-Kommando (zur Auswahl einer Buszyklusart) oder eine Byteauswahl (aus 32-Bit Datenwort)
 - **/REQ_i**: Busanforderung von Karte in Slot i
 - **/GNT_i**: Buszuteilung an Karte in Slot i
 - **/INTA** bis **/INTD**: vier Unterbrechungs-Leitungen
 - **/Frame**: signalisiert Beginn und Ende eines Buszyklus
 - **/IRdy** (*Initiator Ready*): Master ist bereit zum Datentransfer
 - **/TRdy** (*Target Ready*): Slave ist bereit zum Datentransfer
 - **/DevSel**: Target bestätigt die Dekodierung seiner Adresse
 - **/IdSel_i**: Auswahl von Karte in Slot i zur Konfiguration

Fallstudie: PC-Bussysteme (6)

- Beispiel eines PCI-Buszyklus (Lesen eines Blocks aus 4 Worten):



- **bidirektionale Flußkontrolle:**
 - sowohl **Master** (*Initiator*) als auch **Slave** (*Target*) können den Transfer eines Datenworts durch Aktivierung von **/IRDY** bzw. **/TRDY** um einen oder mehrere Bustakte verzögern
- max. **Transferrate:** ein 32-Bit Wort je Bustakt
 - kann nur bei einem sehr langen Burst-Transfer erreicht werden
- **Richtungsumschaltung** der Bustreiber (*turn around*)
 - für gemultiplexte Adress-/Datenleitungen
 - benötigt einen zusätzlichen Bustakt
 - nur beim Lesen erforderlich (langsamer als Schreiben!)
- über PCI-PCI-Bridges **hierarchisch** erweiterbares Bus-System mit maximal 255 PCI-Bussen

- **3 Adressräume:**

- 32 Bit **Speicher-Adressraum** (max. 4GByte)
- 32 Bit **E/A-Adressraum** (I/O-Ports)
- **Konfigurations-Adressraum** (256 Byte je PCI-Karte)
- Buskommando legt für jeden Buszyklus den Adressraum fest, z.B.:

0010	I/O Read	0110	Memory Read	1010	Configuration Read
0011	I/O Write	0111	Memory Write	1011	Configuration Write

- bei Zugriff auf den Speicher-Adressraum sind **Burst-Transfers** möglich, z.B. mit den Buskommandos

1110 *Memory Read Line* (Lesen einer kompletten Cachezeile)

1100 *Memory Read Multiple* (Lesen mehrerer Cachezeilen in einem Buszyklus)