

# Grundlagen der Informatik für Ingenieure I

## Background:

### 4. Dateisystem/Betriebssystemschnittstelle

- 4.1 Überblick
- 4.2 Dateien
  - 4.2.1 Dateiattribute
  - 4.2.2 Operationen auf Dateien
- 4.3 Kataloge
  - 4.3.1 Katalogattribute
  - 4.3.2 Operationen auf Katalogen
- 4.4 Implementierung von Dateisystemen
  - 4.4.1 Festplatten
  - 4.4.2 Speicherung von Daten
    - 4.4.2.1 Kontinuierliche Speicherung
    - 4.4.2.2 Verkettete Speicherung
    - 4.4.2.3 Indiziertes Speichern
    - 4.4.2.4 Baumsequentielles Speichern
  - 4.4.3 Implementierung von Katalogen

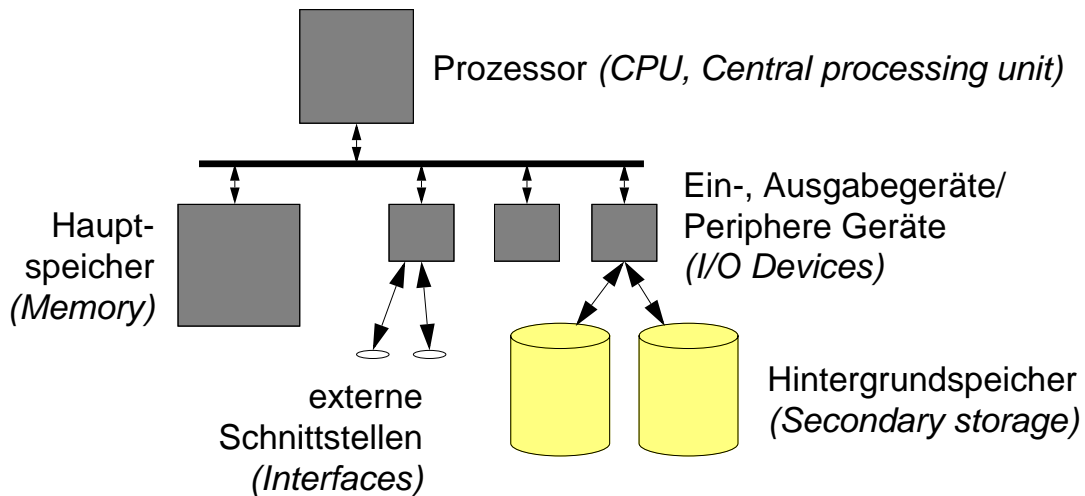
## Background:

### 4. Dateisystem/Betriebssystemschnittstelle

- 4.5 Fallstudie: UNIX - Solaris
  - 4.5.1 Pfadnamen
  - 4.5.2 Eigentümer und Rechte
  - 4.5.3 Inodes
  - 4.5.4 Spezialdateien
  - 4.5.5 Montieren eines Dateibaums (Verbindungen v. Partitionen)
  - 4.5.6 Limitierung der Plattennutzung
  - 4.5.7 Fehlerhafte Plattenblöcke
  - 4.5.8 Datensicherung
    - 4.5.8.1 Backup Scheduling (Beispiele)
    - 4.5.8.2 Redundante Datenhaltung

## 4.1 Dateisysteme - Überblick

### ■ Einordnung:

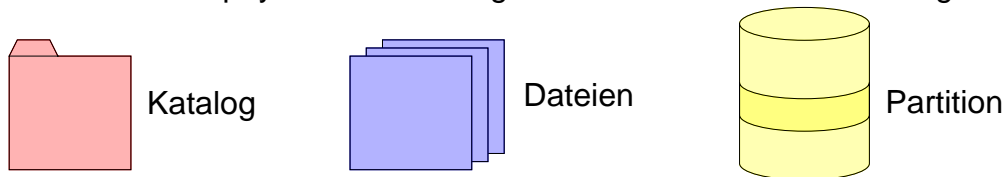


## 4.1 Dateisysteme - Überblick

- Dateisysteme speichern Daten und Programme in Dateien
  - ◆ Betriebssystemabstraktion zur Nutzung von Hintergrundspeichern (z.B. Platten, CD-ROM, Floppy Disk, Bandlaufwerke)
    - Benutzer muss sich nicht um die Ansteuerungen verschiedener Speichermedien kümmern
    - einheitliche Sicht auf den Sekundärspeicher
  
- Dateisysteme bestehen aus
  - ◆ Dateien (*Files*)
  - ◆ Katalogen / Verzeichnissen (*Directories*)
  - ◆ Partitionen (*Partitions*)

## 4.1 Dateisysteme - Überblick

- Datei
  - ◆ speichert Daten oder Programme
- Katalog
  - ◆ enthält Namen der Dateien
  - ◆ enthält Zusatzinformationen zu Dateien
- Partitionen (logische Plattensysteme)
  - ◆ eine Menge von Katalogen und deren Dateien
  - ◆ Sie dienen zum physischen oder logischen Trennen von Dateimengen.



n.

## 4.2 Dateien

- Kleinste Einheit, in der etwas auf den Hintergrundspeicher geschrieben werden kann.

### 1 Dateiattribute

- *Name* — Symbolischer Name, vom Benutzer les- und interpretierbar
  - ◆ z.B. `GiroKonto.java`; `GiroKonto.class`; `GiroKonto.html`
- *Typ* — Für Dateisysteme, die verschiedene Dateitypen unterscheiden
  - ◆ z.B. sequentielle Datei, satzorientierte Datei
- *Ortsinformation* — Wo werden die Daten physisch gespeichert?
  - ◆ Gerätenummer, Nummern der Plattenblocks

# 1 Dateiattribute

- **Größe** — Länge der Datei in Größeneinheiten (z.B. Bytes, Blocks, Sätze)
  - ◆ steht in engem Zusammenhang mit der Ortsinformation
  - ◆ wird zum Prüfen der Dateigrenzen z.B. beim Lesen benötigt
  
- **Zeitstempel** — Zeit und Datum der Erstellung, letzten Modifikation etc.
  - ◆ unterstützt Backup, automatische Dateierzeugung, Benutzerüberwachung etc.
  
- **Rechte** — Zugriffsrechte bestimmen, wer lesen, schreiben etc. kann
  - ◆ z.B. nur für den Eigentümer schreibbar für alle anderen nur lesbar
  
- **Eigentümer** — Identifikation des Eigentümers
  - ◆ Eventuell eng mit den Rechten verknüpft
  - ◆ Zuordnung beim Accounting (Abrechnung des Plattenplatzes)

# 2 Operationen auf Dateien

- **Öffnen (Open):**
  - ◆ Dem Betriebssystem wird bekannt gegeben, dass E/A-Operationen auf eine bestimmte Datei ausgeführt werden sollen.
  - ◆ Daraufhin werden sämtliche, für die Verwaltung und Kontrolle von E/A-Operationen notwendigen, Daten in Kontrolldatenbereiche des Betriebssystemkerns im Hauptspeicher transferiert.
  - ◆ Diese Maßnahme beschleunigt die späteren E/A-Vorgänge erheblich.
  
- **Schließen (Close):**
  - ◆ Dem Betriebssystem wird bekannt gegeben, dass keine weiteren E/A-Operationen auf eine bestimmte Datei ausgeführt werden sollen.
  - ◆ Daraufhin werden sämtliche für die Verwaltung und Kontrolle von E/A-Operationen notwendigen Daten aus Kontrolldatenbereichen des Betriebssystemkerns auf die Platte transferiert.

## 2 Operationen auf Dateien

---

- Erzeugen (*Create*)
  - ◆ Nötiger Speicherplatz wird angefordert
  - ◆ Katalogeintrag wird erstellt
  - ◆ Initiale Attribute werden gespeichert
  
- Schreiben (*Write*)
  - ◆ Daten werden auf Platte geschrieben
  - ◆ Eventuelle Anpassung der Attribute, z.B. Länge
  
- Lesen (*Read*)
  - ◆ Daten werden von Platte gelesen

## 2 Operationen auf Dateien

---

- Positionieren des Schreib-/Lesezeigers (*Seek*)
  - ◆ In manchen Systemen wird dieser Zeiger implizit bei Schreib- und Leseoperationen positioniert
  - ◆ Ermöglicht explizites Positionieren
  
- Löschen (*Delete*)
  - ◆ Entfernen der Datei aus dem Katalog und Freigabe der Plattenblocks

## 4.3 Kataloge

---

- Ein Katalog enthält Dateien und evtl. andere Kataloge
  - ◆ Katalog enthält Namen und Verweise auf Dateien und andere Kataloge  
z.B. *UNIX*, *MS-DOS*
  - ◆ Zusätzliche Bedingung
    - Katalog enthält Namen und Verweise auf Dateien, die einer bestimmten Zusatzbedingung gehorchen  
z.B. gleiche Gruppennummer in *CP/M*  
z.B. eigenschaftsorientierte und dynamische Gruppierung in *BeOS-BFS*

### 1 Katalogattribute

---

- ◆ Die meisten Dateiattribute treffen auch für Kataloge zu
  - Name, Ortsinformationen, Größe, Zeitstempel, Rechte, Eigentümer

### 2 Operationen auf Katalogen

---

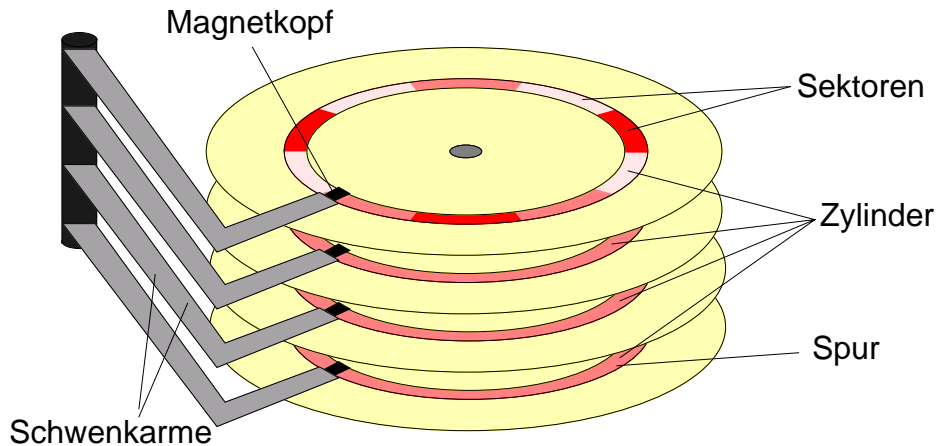
- ◆ Erzeugen und Löschen von Katalogen (*Create and Delete Directory*)
- ◆ Auslesen der Einträge (*Read, Read directory*)
  - Daten des Kataloginhalts werden gelesen
- ◆ Erzeugen und Löschen der Einträge erfolgt implizit mit der zugehörigen Dateioperation

## 4.4 Implementierung von Dateisystemen

### 1 Festplatten

- Häufigstes Medium zum Speichern von Dateien
  - ◆ Floppy Disks sind in der Handhabung ähnlich

- Aufbau einer Festplatte



### 1 Festplatten (cont.)

- Zugriffsmerkmale
  - ◆ blockorientierter Zugriff
  - ◆ Blockgröße zwischen 32 und 4096 Bytes (typisch 512 Bytes)
  - ◆ Zugriff erfordert Positionierung des Schwenkarms auf den richtigen Zylinder und Warten auf den entsprechenden Sektor
- Blöcke sind üblicherweise nummeriert
  - ◆ getrennte Numerierung: Zylindernummer, Sektornummer
  - ◆ kombinierte Numerierung: durchgehende Nummern über alle Sektoren (Reihenfolge: aufsteigend innerhalb eines Zylinders, dann folgender Zylinder, etc.)

## 2 Speicherung von Dateien

- ◆ Dateien benötigen meist mehr als einen Block auf der Festplatte
  - Welche Blöcke werden für die Speicherung einer Datei verwendet?

### 4.4.2.1 Kontinuierliche Speicherung

- ◆ Datei wird in Blöcken mit aufsteigenden Blocknummern gespeichert
  - Zugriff auf alle Blöcke mit minimaler Positionierzeit des Schwenkarms
  - Einsatz z.B. bei Systemen mit Echtzeitanforderungen
- ◆ Probleme
  - Finden des freien Platzes auf der Festplatte (Menge aufeinanderfolgender und freier Plattenblöcke)
  - Fragmentierungsproblem (Verschnitt: nicht nutzbare Plattenblöcke; siehe auch Speicherverwaltung)

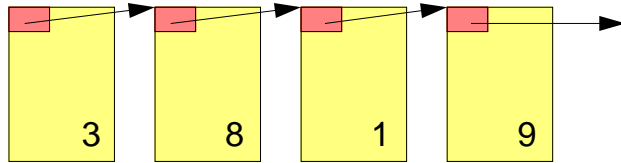
### 4.4.2.1 Kontinuierliche Speicherung

- ◆ Weiteres Problem:
  - Größe bei neuen Dateien oft nicht im voraus bekannt
  - Erweitern ist problematisch
    - Umkopieren, falls kein freier angrenzender Block mehr verfügbar
- ◆ Variation:
  - Unterteilen einer Datei in Folgen von Blocks (*Chunks, Extents*)
  - Blockfolgen werden kontinuierlich gespeichert



## 4.4.2.2 Verkettete Speicherung

- ◆ Blöcke einer Datei sind verkettet

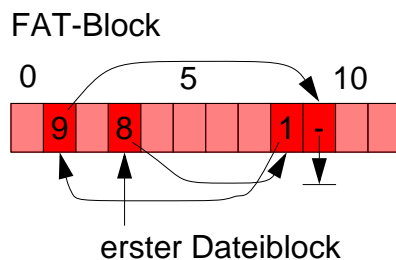


- z.B. Commodore Systeme (CBM 64 etc.)
  - Blockgröße 256 Bytes
  - die ersten zwei Bytes bezeichnen Zylinder- und Sektornummer des nächsten Blocks
  - wenn Zylindernummer gleich Null: letzter Block
  - 254 Bytes Nutzdaten

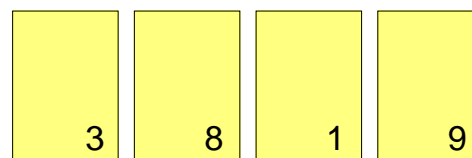
- ◆ File kann wachsen und verlängert werden

## 4.4.2.2 Verkettete Speicherung

- ◆ Probleme:
  - Speicher für Verzeigerung geht von den Nutzdaten im Block ab
  - Fehleranfälligkeit: Datei ist nicht restaurierbar, falls einmal Verzeigerung fehlerhaft
- ◆ Verkettung wird in speziellen Plattenblocks gespeichert
  - FAT-Ansatz (*FAT: File allocation table*), z.B. MS-DOS, Windows 95



Blöcke der Datei: 3, 8, 1, 9



## 4.4.2.2 Verkettete Speicherung

### ◆ Vorteile:

- kompletter Inhalt des Datenblocks ist nutzbar
- mehrfache Speicherung der FAT möglich: Einschränkung der Fehleranfälligkeit

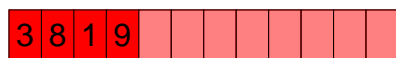
### ◆ Probleme:

- mindestens ein zusätzlicher Block muss geladen werden
- FAT enthält Verkettungen für alle Dateien: das Laden der FAT-Blöcke lädt auch nicht benötigte Informationen

## 4.4.2.3 Indiziertes Speichern

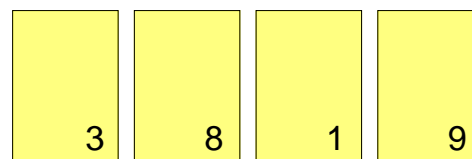
### ◆ Spezieller Plattenblock enthält Blocknummern der Datenblocks einer Datei:

Indexblock



↑  
erster Dateiblock

Blöcke der Datei: 3, 8, 1, 9

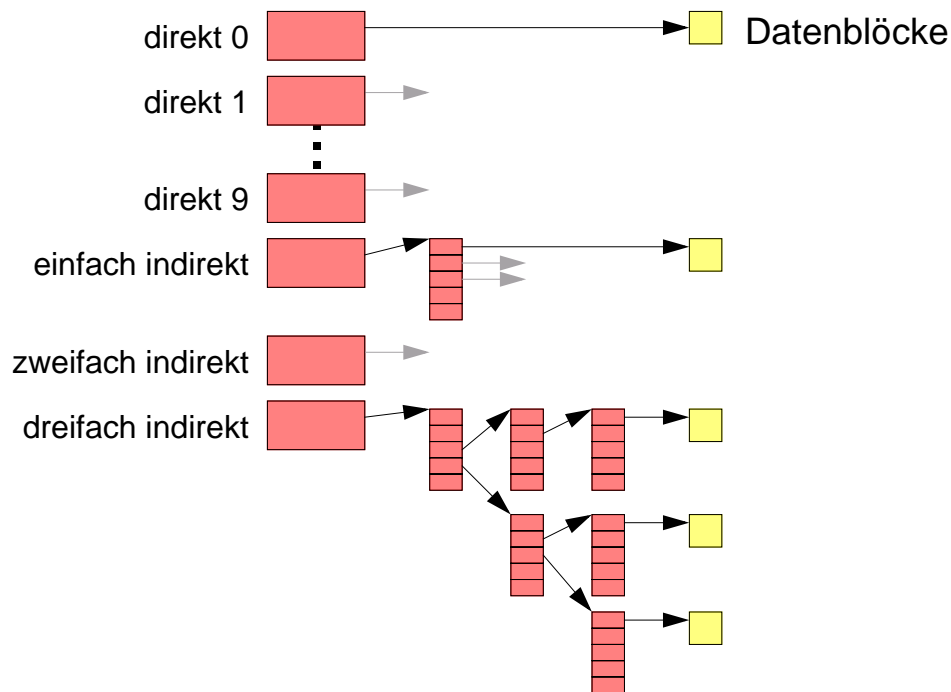


### ◆ Problem:

- feste Anzahl von Blöcken
  - Verschnitt bei kleinen Dateien
  - Erweiterung nötig bei großen Dateien

### 4.4.2.3 Indiziertes Speichern

◆ Beispiel UNIX Inode:



### 4.4.2.3 Indiziertes Speichern

- ◆ Einsatz von mehreren Stufen der Indizierung:
  - durch mehrere Stufen der Indizierung auch große Dateien adressierbar
- ◆ Nachteil:
  - mehrere Blöcke müssen geladen werden (nur bei langen Dateien)

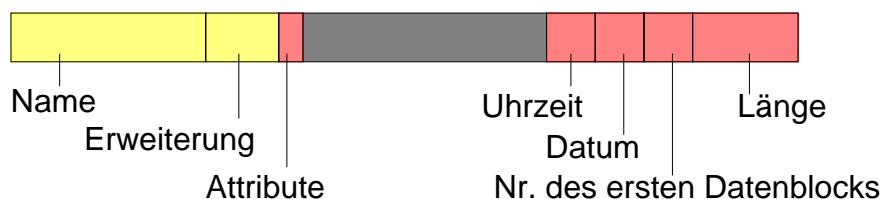
## 4.4.2.4 Baumsequentielle Speicherung

Nur zur Kenntnisnahme - nähere Erklärungen im Kapitel "Datenstrukturen"!

- ◆ Satzorientierte Dateien:
  - Schlüssel + Datensatz
  - effizientes Auffinden des Datensatz mit einem bekannten Schlüssel
  - Schlüsselmenge spärlich besetzt
  - häufiges Einfügungen und Löschen von Datensätzen
  
- ◆ Einsatz von B-Bäumen zur Satzspeicherung:
  - innerhalb von Datenbanksystemen
  - als Implementierung spezieller Dateitypen kommerzieller Betriebssysteme:
    - z.B. VSAM-Dateien in MVS (*Virtual storage access method*)
    - z.B. NTFS Katalogimplementierung

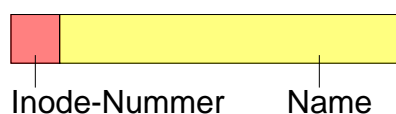
## 3 Implementierung von Katalogen

- Einträge werden hintereinander in eine Liste gespeichert  
z.B. *FAT File systems*



- Eintrag kann feste oder variable Länge haben

- in UNIX



- Inode-Nummer verweist auf einen Eintrag mit den restlichen Daten

## 4.5 Fallstudie: UNIX - Solaris

### ◆ Datei:

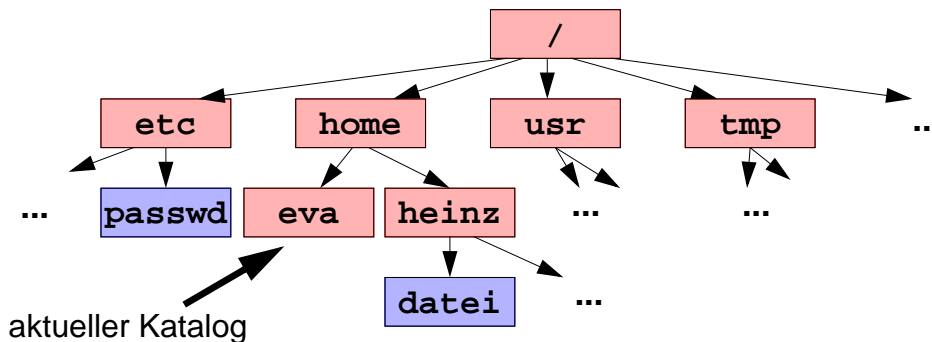
- einfache, unstrukturierte Folge von Bytes
- beliebiger Inhalt; für das Betriebssystem ist der Inhalt transparent
- dynamisch erweiterbar
- Zugriffsrechte: lesbar, schreibbar, ausführbar

### ◆ Katalog:

- baumförmig strukturiert
  - Knoten des Baums sind Kataloge
  - Blätter des Baums sind Verweise auf Dateien (*Links*)
- jedem UNIX Prozess ist zu jeder Zeit ein aktueller Katalog (*Current working directory*) zugeordnet
- Zugriffsrechte: lesbar, schreibbar, durchsuchbar, „nur“ erweiterbar

## 1 Pfadnamen

### ◆ Baumstruktur:

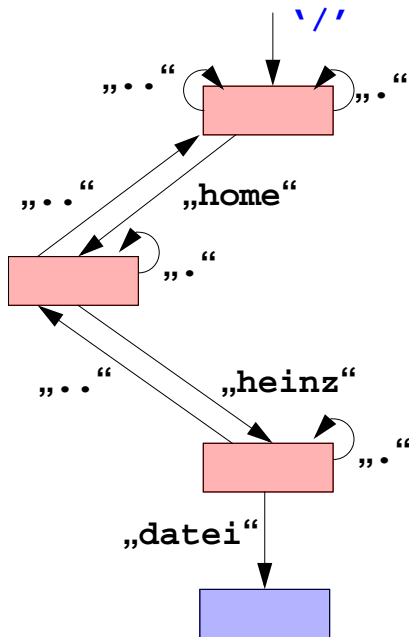


### ◆ Pfade:

- z.B. „/home/heinz/datei“, „/tmp“, „../heinz/datei“
- „/“ ist Trennsymbol (*Slash*); beginnender „/“ bezeichnet Wurzelkatalog; sonst Beginn implizit mit dem aktuellem Katalog

# 1 Pfadnamen

## ◆ Eigentliche Baumstruktur:



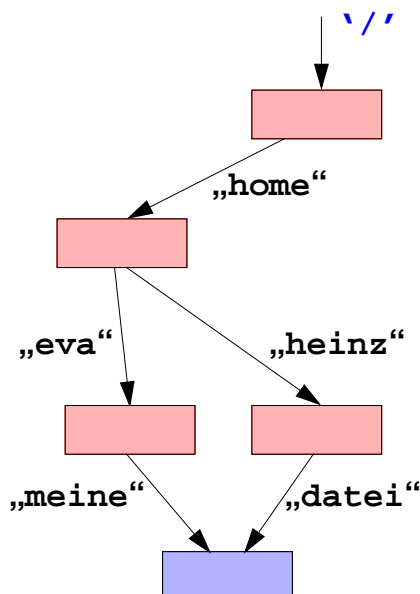
## ◆ benannt sind nicht Dateien und Kataloge, sondern die Verbindungen zwischen ihnen:

- Kataloge und Dateien können auf verschiedenen Pfaden erreichbar sein  
z.B. `../heinz/datei` und `/home/heinz/datei`
- Jeder Katalog enthält einen Verweis auf sich selbst („.“) und einen Verweis auf den darüberliegenden Katalog im Baum („..“)

# 1 Pfadnamen

## ◆ Links (*Hard links*):

- Dateien können mehrere auf sich zeigende Verweise besitzen, sogenannte *Hard links*

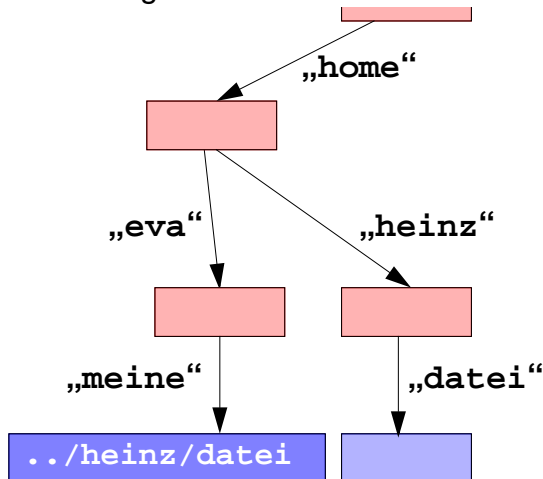


- Die Datei hat zwei Einträge in verschiedenen Katalogen, die völlig gleichwertig sind:  
`/home/eva/meine`  
`/home/heinz/datei`
- Datei wird erst gelöscht, wenn letzter Link gekappt wird.

## 1 Pfadnamen

### ◆ Symbolische Namen (*Symbolic links*):

- Verweise auf einen anderen Pfadnamen (sowohl auf Dateien als auch Kataloge)
- Symbolischer Name bleibt auch bestehen, wenn Datei oder Katalog nicht mehr existiert



- Symbolischer Name enthält einen neuen Pfadnamen, der vom FS interpretiert wird.

## 2 Eigentümer und Rechte

### ◆ Eigentümer:

- Jeder Benutzer wird durch eindeutige Nummer (UID) repräsentiert
- Ein Benutzer kann einer oder mehreren Benutzergruppen angehören, die durch eine eindeutige Nummer (GID) repräsentiert werden
- Eine Datei oder ein Katalog ist genau einem Benutzer und einer Gruppe zugeordnet

### ◆ Rechte auf Dateien:

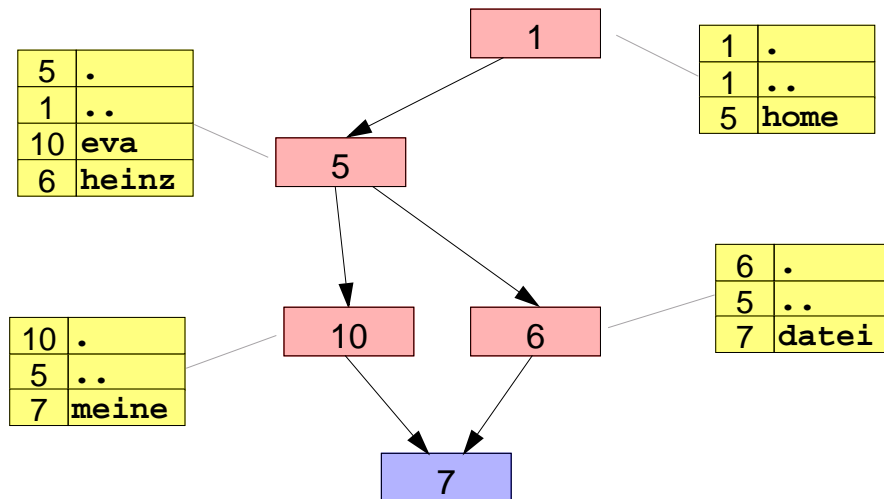
- Lesen, Schreiben, Ausführen (nur vom Eigentümer veränderbar)
- einzeln für den Eigentümer, für Angehörige der Gruppe und für alle anderen einstellbar

### ◆ Rechte auf Kataloge:

- Lesen, Schreiben (Löschen und Anlegen von Dateien etc.), Durchsuchen
- Schreibrecht ist einschränkbar auf eigene Dateien

### 3 Inodes

- ◆ Attribute einer Datei und Ortsinformationen über ihren Inhalt werden in sogenannten Inodes gehalten
  - Inodes werden pro Partition numeriert (*Inode number*)
- ◆ Kataloge enthalten lediglich Paare von Namen und Inode-Nummern



### 3 Inodes

- ◆ Inhalt eines Inodes:
  - Inodenummer
  - Dateityp: Katalog, normale Datei, Spezialdatei (z.B. Gerät)
  - Eigentümer und Gruppe
  - Zugriffsrechte
  - Zugriffszeiten: letzte Änderung (*mtime*), letzter Zugriff (*atime*), letzte Änderung des Inodes (*ctime*)
  - Anzahl der Hard links auf den Inode
  - Dateigröße (in Bytes)
  - Adressen der Datenblöcke des Datei- oder Kataloginhalts (zehn direkt Adressen und drei indirekte)



## 4 Spezialdateien

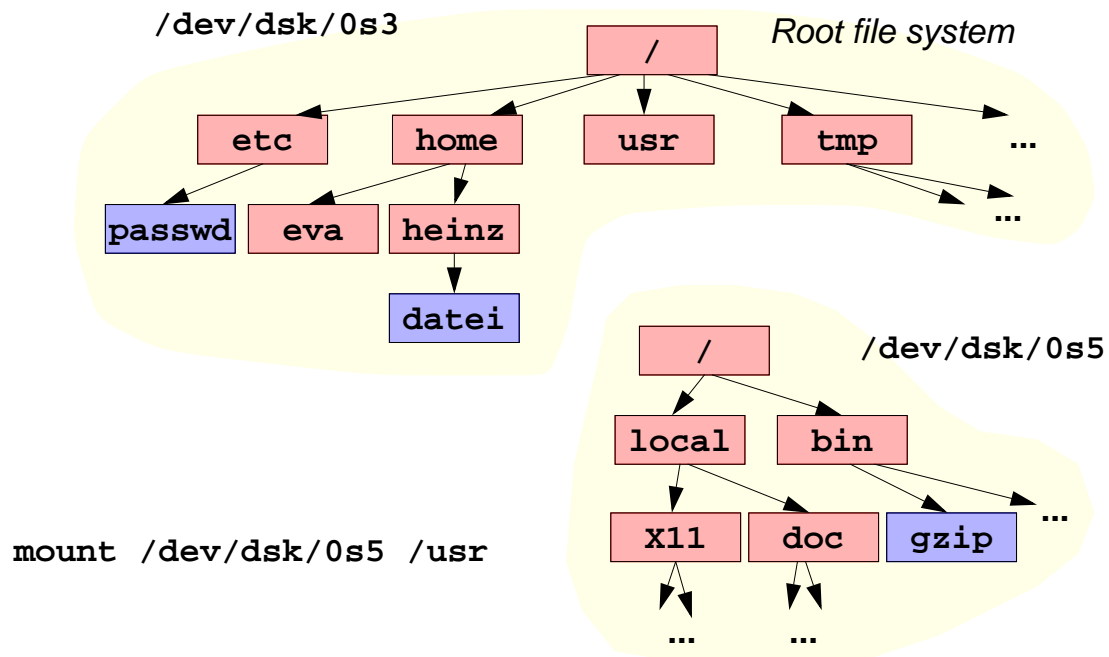
- ◆ Periphere Geräte werden als Spezialdateien repräsentiert:
  - Geräte können wie Dateien mit Lese- und Schreiboperationen angesprochen werden
  - Öffnen der Spezialdateien schafft eine (evtl. exklusive) Verbindung zum Gerät, die durch einen Treiber hergestellt wird
- ◆ Blockorientierte Spezialdateien:
  - Plattenlaufwerke, Bandlaufwerke, Floppy Disks, CD-ROMs
- ◆ Zeichenorientierte Spezialdateien:
  - Serielle Schnittstellen, Drucker, Audiokanäle etc.
  - blockorientierte Geräte haben meist auch eine zusätzliche zeichenorientierte Repräsentation

## 5 Montieren des Dateibaums

- ◆ Der UNIX-Dateibaum kann aus mehreren Partitionen zusammenmontiert werden
  - Partition wird Dateisystem genannt (*File system*)
  - wird durch blockorientierte Spezialdatei repräsentiert (z.B. `/dev/dsk/0s3`)
  - Das Montieren wird *Mounten* genannt
  - Ausgezeichnetes Dateisystem ist das *Root file system*, dessen Wurzelkatalog gleichzeitig Wurzelkatalog des Gesamtsystems ist
  - Andere Dateisysteme können mit dem Befehl `mount` in das bestehende System hineinmontiert werden

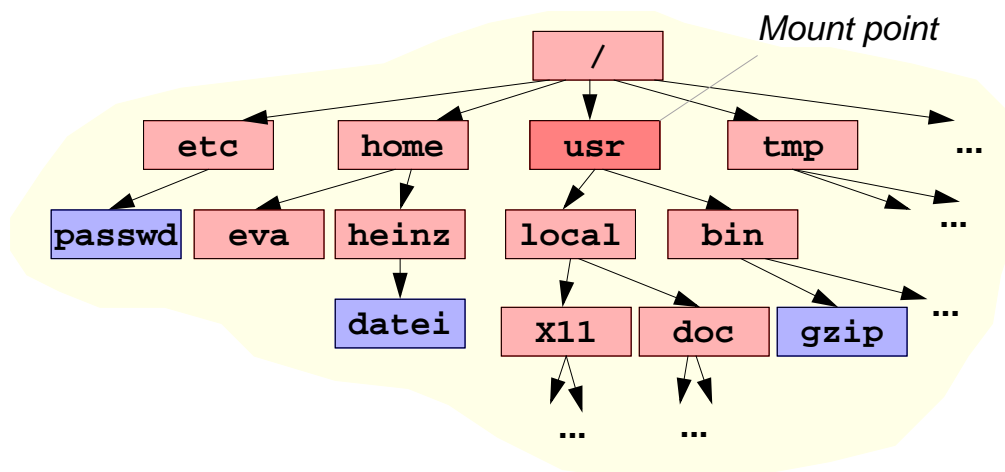
## 5 Montieren des Dateibaums

◆ Beispiel:



## 5 Montieren des Dateibaums

◆ Beispiel nach Ausführung des Montierbefehls:



## 6 Limitierung der Plattennutzung

- ◆ Mehrbenutzersysteme:
  - einzelnen Benutzern sollen verschieden große Kontingente zur Verfügung stehen
  - gegenseitige Beeinflussung soll vermieden werden (*disk full* Fehlermeldung)
- ◆ Quota Systeme (Quantensysteme):
  - Tabelle enthält maximale und augenblickliche Anzahl von Blöcken für die Dateien und Kataloge eines Benutzers
  - Tabelle steht auf Platte und wird vom *file system* fortgeschrieben
  - Benutzer erhält *disk full* Meldung, wenn sein Quota verbraucht ist
  - üblicherweise gibt es eine weiche und eine harte Grenze (weiche Grenze kann für eine bestimmte Zeit überschritten werden)

## 7 Fehlerhafte Plattenblöcke

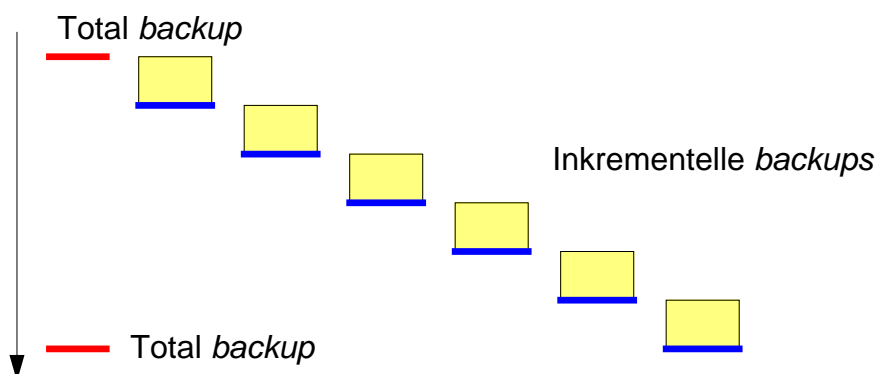
- ◆ Blöcke, die beim Lesen Fehlermeldungen erzeugen:
  - z.B. Checksummenfehler
- ◆ Hardwarelösung:
  - Platte und Plattencontroller bemerken selbst fehlerhafte Blöcke und markieren diese als fehlerhaft
  - Zugriff auf den Block wird vom Controller automatisch auf einen „gesunden“ Block umgeleitet
- ◆ Softwarelösung:
  - *file system* bemerkt fehlerhafte Blöcke und markiert diese auch als belegt

## 8 Datensicherung

- ◆ Schutz vor dem Totalausfall von Platten:
  - z.B. durch *head crash* oder andere Fehler
- ◆ Sichern der Daten auf Tertiärspeicher:
  - Bänder
  - WORM Speicherplatten (*Write once read many*)
- ◆ Sichern großer Datenbestände:
  - Total *backups* benötigen lange Zeit
  - Inkrementelle *backups* sichern nur Änderungen ab einem bestimmten Zeitpunkt
  - Mischen von Total *backups* mit inkrementellen *backups*

### 4.5.8.1 Beispiele für *Backup Scheduling*

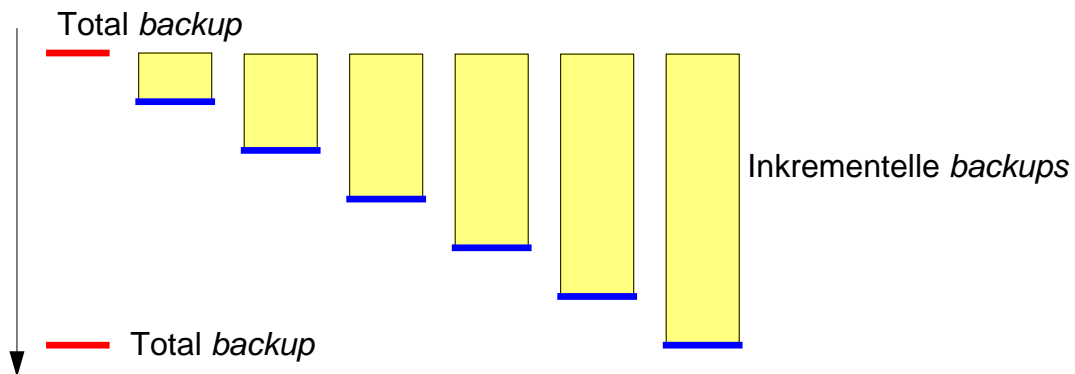
- ◆ Gestaffelte inkrementelle *backups*:



- z.B. alle Woche ein Total *backup* und jeden Tag ein inkrementelles *backup* zum Vortag: maximal 7 *backups* müssen eingespielt werden

## 4.5.8.1 Beispiele für *Backup Scheduling* (2)

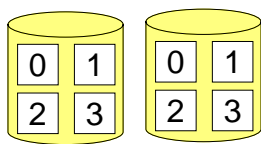
- ◆ Gestaffelte inkrementelle *backups* zum letzten Total *backup*:



- z.B. alle Woche ein Total *backup* und jeden Tag ein inkrementelles *backup* zum letzten Total *backup*: maximal 2 *backups* müssen eingespielt werden
- ◆ Hierarchie von Backupläufen:
  - mehrstufige inkrementelle *backups* zum *backup* der nächst höheren Stufe
  - optimiert Archivmaterial und Restaurierungszeit

## 4.5.8.2 Einsatz mehrerer redundanter Platten

- ◆ Gespiegelte Platten - *Mirroring*; *RAID 0*  
(*RAID*: *redundant array of independent disks*):
  - Daten werden auf zwei Platten gleichzeitig gespeichert

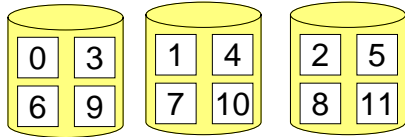


- Implementierung durch Software (Filesystem, Plattentreiber) oder Hardware (spez. Controller)
- eine Platte kann ausfallen
- schnelleres Lesen (da zwei Platten unabhängig beauftragt werden können)
- ◆ Nachteil:
  - doppelter Speicherbedarf
  - wenig langsames Schreiben durch Warten auf zwei Plattentransfers

## 4.5.8.2 Einsatz mehrerer redundanter Platten

### ◆ Gestreifte Platten - *Striping*; *RAID 1*:

- Daten werden über mehrere Platten gespeichert



- Datentransfers sind nun schneller, da mehrere Platten gleichzeitig angesprochen werden können

### ◆ Nachteil:

- keinerlei Datensicherung: Ausfall einer Platte lässt Gesamtsystem ausfallen

### ◆ Verknüpfung von *RAID 0* und *1* möglich (*RAID 0+1*)