

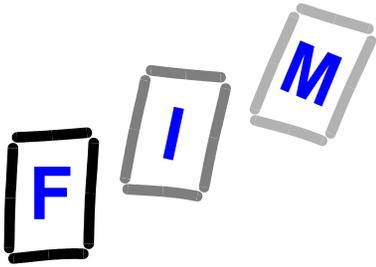
**SS 2004**

# **KV Betriebssysteme**

**(Rudolf Hörmanseder, Michael Sonntag)**

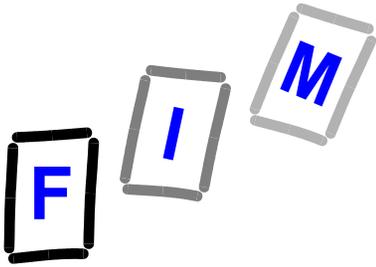
## **Wiederholung Dateisysteme**

**© P. R. Dietmüller, A. Putzinger, FIM 2004**



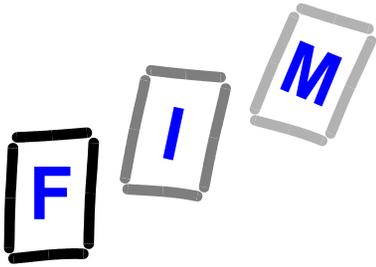
# Was ist ein Dateisystem?

- **Permanente Speicherung**
- **Schnittstelle zwischen Betriebssystem und Laufwerken**
- **Organisation der Informationen**
  - **Dateien**
  - **Verzeichnisse**



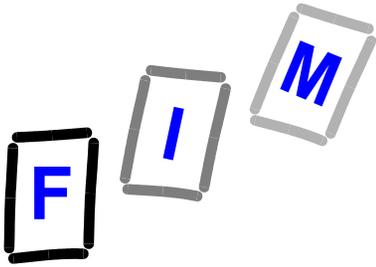
# Welche Dateisysteme gibt es?

- **FAT (File Allocation Table), 16 Bit DOS-System, FAT16**
- **FAT32, Windows 95B OSR2**
- **HPFS (High Performance File System), OS/2, 32-Bit-Dateisystem**
- **NTFS, Windows NT, 32-Bit-Dateisystem**
- **NetWare, eigenes 32-Bit-Dateisystem von Novell**
- **ISO 9660 für CD-ROM und ISO 13346 für DVD**
- **UDF (Universal Disk Format) ist für Speichermedien mit einer großen Kapazität gedacht, wie z.B. DVD-RAM.**
- **ReiserFS, ext, ext2, ext3, xfs, jfs (Linux)**



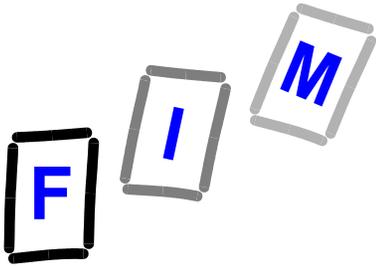
# Betriebssysteme und Dateisysteme

<b>Betriebssystem</b>	<b>Dateisystem(e)</b>
▪ <b>DOS</b>	<b>FAT16</b>
▪ <b>Windows 95</b>	<b>FAT16</b>
▪ <b>Windows 95 OSR2</b>	<b>FAT16, FAT32</b>
▪ <b>Windows 98, ME</b>	<b>FAT16, FAT32</b>
▪ <b>Windows NT 4</b>	<b>FAT16, NTFS4</b>
▪ <b>Windows 2000</b>	<b>FAT16, FAT32, NTFS4</b>
▪ <b>Windows XP</b>	<b>FAT16, FAT32, NTFS4, NTFS5</b>
▪ <i>Windows Longhorn</i>	<i>WinFS</i>
▪ <b>OS/2</b>	<b>FAT16, HPFS</b>
▪ <b>Novell NetWare</b>	<b>eigenes Dateisystem</b>
▪ <b>Linux</b>	<b>ReiserFS, ext, ext2, ext3, xfs, jfs, FAT16, FAT32, NTFS4, NTFS5</b>

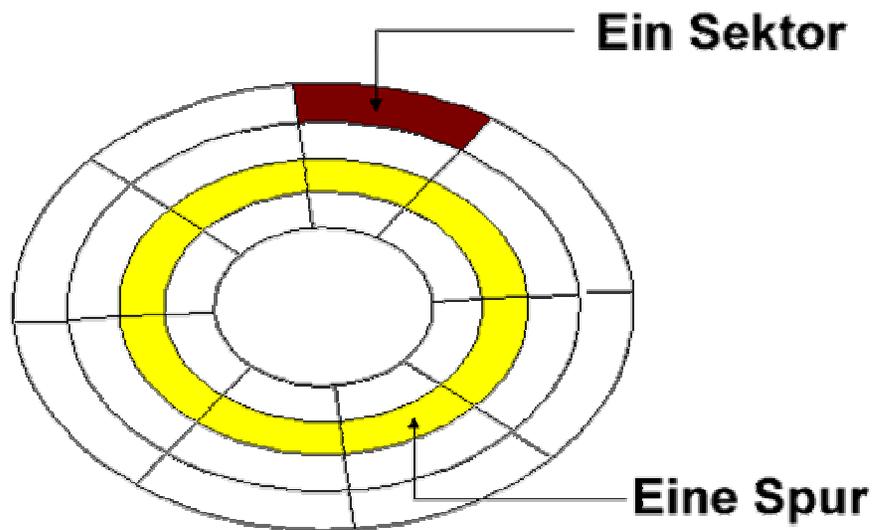


# Betriebssysteme und Größen?

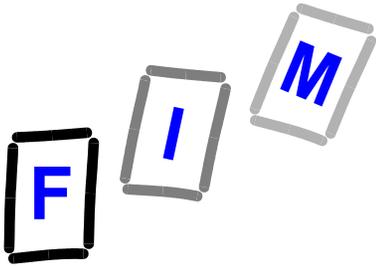
- | <b>Betriebssystem / Dateisystem</b>   | <b>Größe</b>                 |
|---|------------------------------|
| ▪ DOS-Versionen vor 3.0   | bis 16 MB                    |
| ▪ DOS-Version 3.0 und 3.32  | bis 32 MB                    |
| ▪ DOS 4.0   | bis 128 MB                   |
| ▪ DOS 5.0   | bis 528 MB                   |
| ➤ <b>DOS 5.0 und das BIOS, welches für IDE-Laufwerke zuständig ist, akzeptierten nur 1024 Zylinder und Festplatten bis zu 528 MB. Dieses Limit wurde durch den EIDE-Standard gebrochen.</b> |                              |
| ▪ FAT16   | bis 2 GB<br>(16-Bit-Cluster) |
| ▪ FAT32   | bis 2048 GB                  |



# Wozu Spuren und Sektoren?

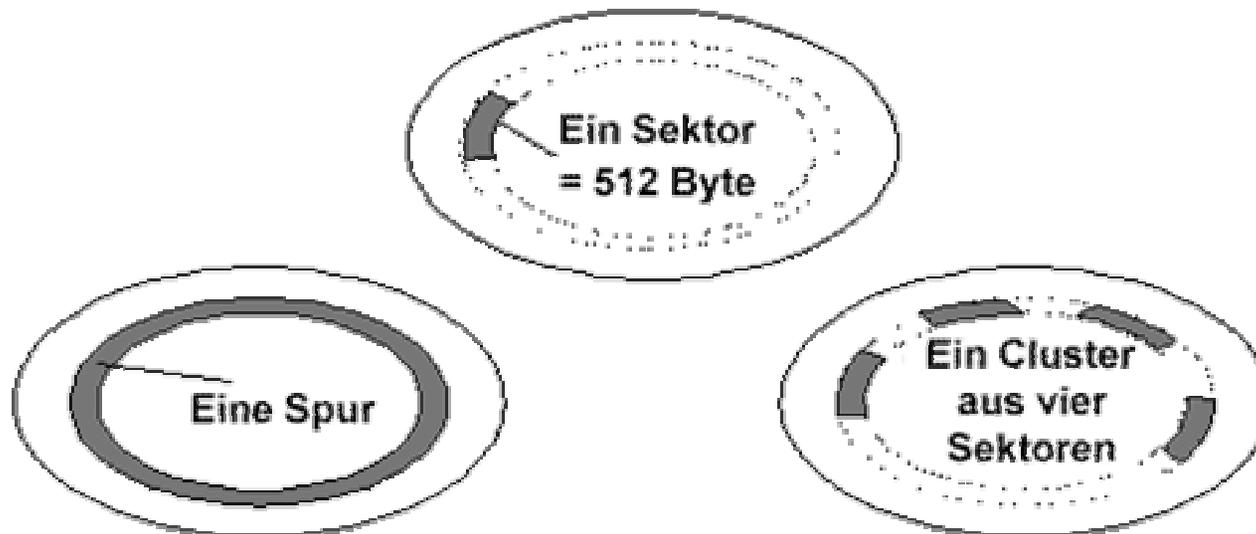


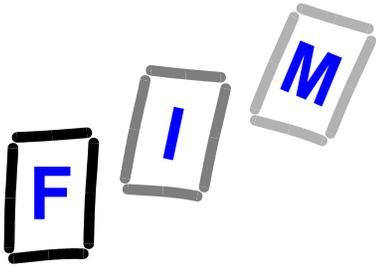
- Das Formatieren erzeugt ein Dateisystem auf einem Speichermedium.
- Sektoren zu je 512 Byte
- Ein Sektor entsteht, wenn das runde Medium in konzentrische Spuren eingeteilt wird. Jede Spur ist in Sektoren unterteilt.



# Wozu Cluster?

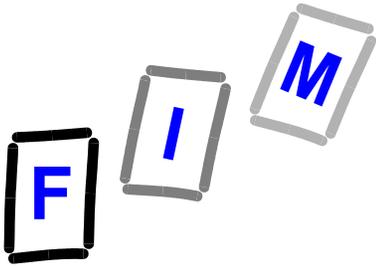
- Zusammenfassung mehrerer Sektoren zu einem Cluster
- Verwaltungstechnische Erfindung, damit Betriebssysteme mit unterschiedlich großen Festplatten umgehen können.
- Zahl der Sektoren hängt von der Größe des Mediums ab.
- Fragmentierung?





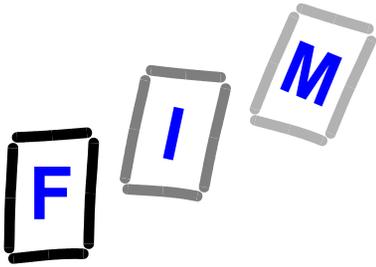
# Clustergrößen?

Partitionsgröße	FAT16	FAT32	NTFS
< 16 MB	2 KB	-	512 Byte
< 32 MB	512 Byte	-	512 Byte
< 64 MB	1 KB	512 Byte	512 Byte
< 128 MB	2 KB	1 KB	512 Byte
< 256 MB	4 KB	2 KB	512 Byte
< 512 MB	8 KB	4 KB	512 Byte
< 1 GB	16 KB	4 KB	1 KB
< 2 GB	32 KB	4 KB	2 KB
< 4 GB	64 KB	4 KB	4 KB
< 8 GB	-	4 KB	4 KB
< 16 GB	-	8 KB	4 KB
< 32 GB	-	16 KB	4 KB
< 2 TB	-	-	4 KB



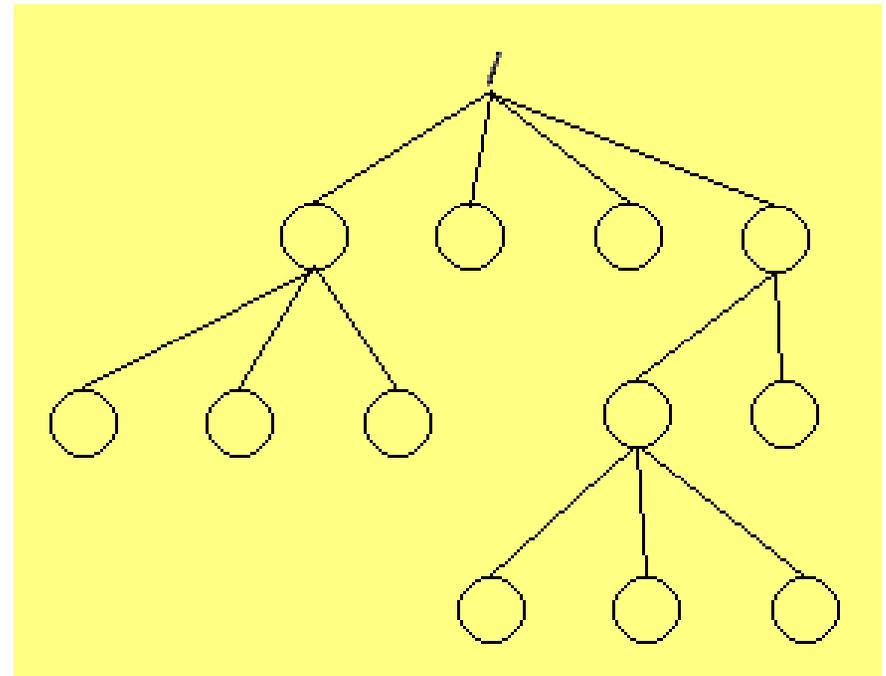
# Was ist eine Datei?

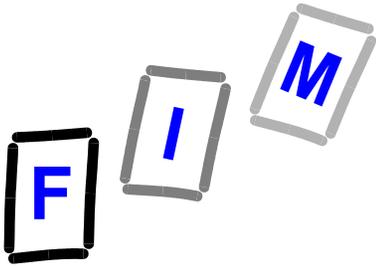
- **Aus der Sicht des Betriebssystems ist eine Datei eine Folge von Bytes, deren Bedeutung dem Betriebssystem nicht bekannt ist. Die Bedeutung definiert die Anwendung.**
- **Zur eindeutigen Identifikation hat eine Datei einen Namen.**
- **Darüber hinaus kann sie noch weitere Attribute haben: Typ, Größe, Zugriffsberechtigungen, Datum und Uhrzeit der Dateianlage, der letzten Änderung, des letzten Zugriffs**



# Was ist ein Verzeichnis?

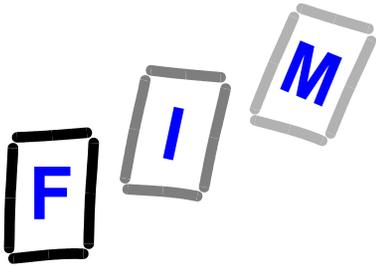
- **Strukturierung der Dateien**
- **Ein Verzeichnis kann Dateien und Unterverzeichnisse beinhalten.**
- **Jedes Verzeichnis hat einen Namen.**
- **Es entsteht ein Baum.**
- **Hierarchisches Dateisystem**



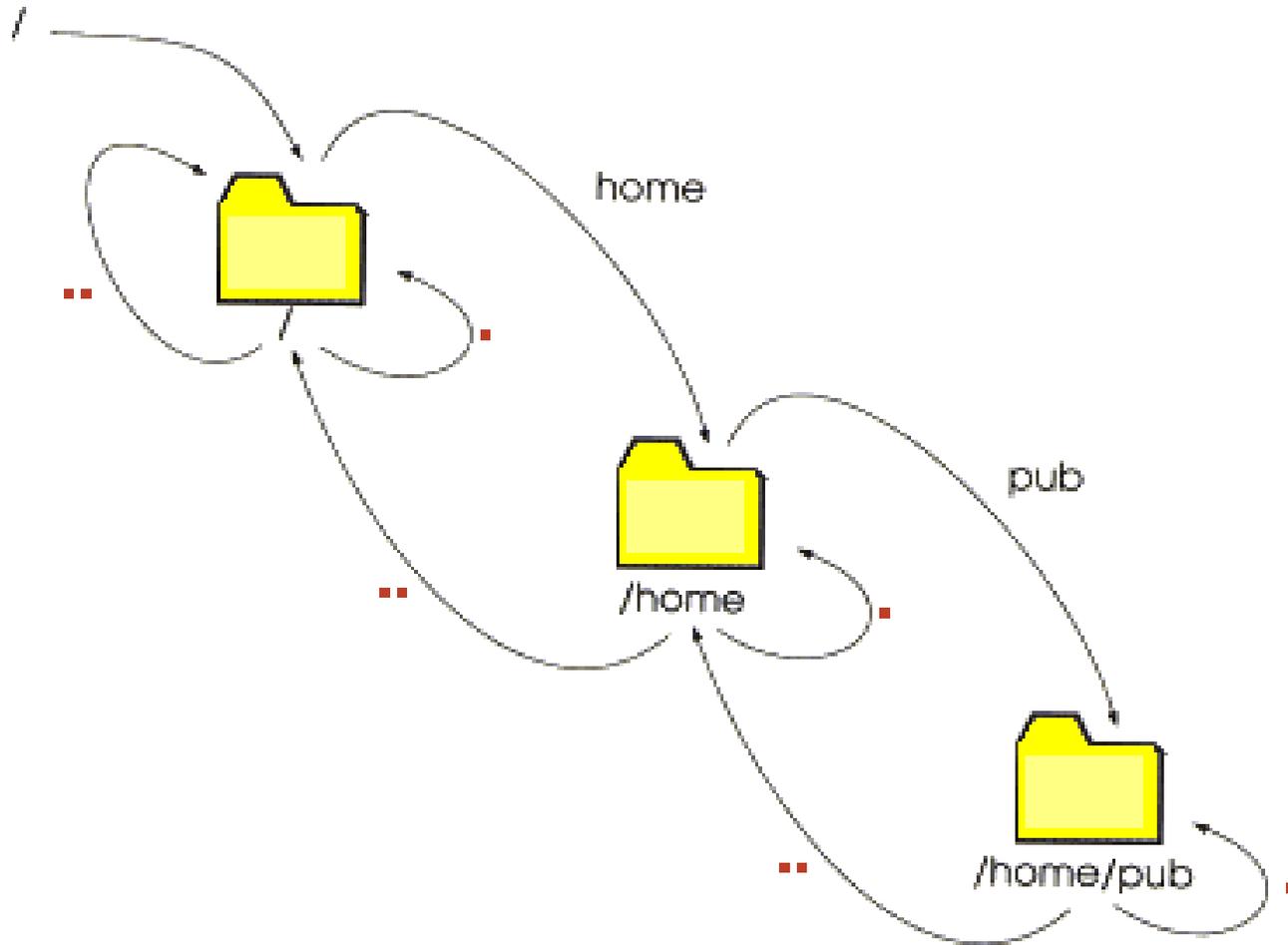


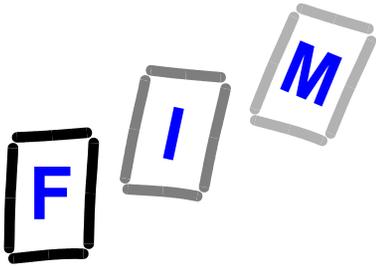
# Datei- und Pfadnamen (1)

- **Verzeichnisse und Dateinamen durch \ oder / trennen,**
  - `/Home/Hoe/Lva/Betriebssystem/2004/Lektion3/Test.doc`
- **Absolute Pfadangaben (beginnen beim Wurzelverzeichnis)**
  - `C:\Hoe\Lva\Betriebssystem\2004\Lektion3\Test.doc`
- **Relative Pfadangaben (beginnen beim akt. Verzeichnis)**
  - `Test.doc`, `.\Test.doc`
  - `..\Test.doc`
- **Spezielle Verzeichnisse**
  - `.`     **Aktuelles Verzeichnis**
  - `..`    **Übergeordnetes Verzeichnis**



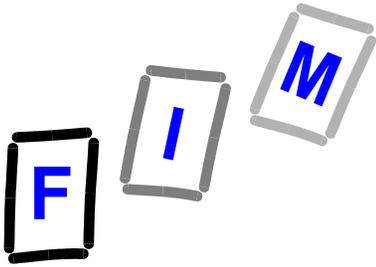
# Datei- und Pfadnamen (2)





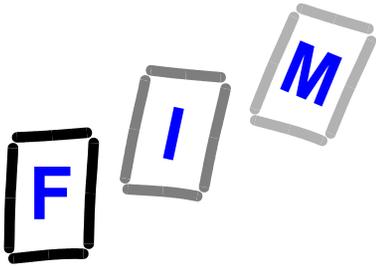
# Typische Dateioperationen

- Datei öffnen / anlegen, z.B.: `new File`
- Datei lesen, z.B.: `FileInputStream.read`
- Datei schreiben, z.B.: `FileOutputStream.write`
- Datei positionieren, z.B.:  
`RandomAccessFile.seek`
- Datei schließen, z.B.: `FileInputStream.close`
- Datei löschen, z.B.: `File.delete`
- (Datei leeren)



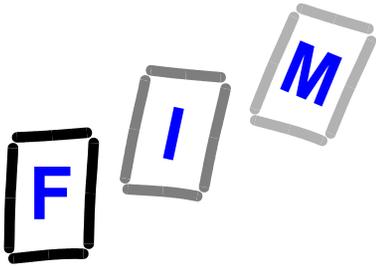
# Datei sequentiell lesen

```
public static void main(String[] args) throws ... {  
  
    int data;  
    FileInputStream fis;  
  
    fis = new FileInputStream(  
        "c:\\winnt\\taskman.exe");  
    while ((data = fis.read()) != -1) {  
        ....  
    }  
    fis.close();  
  
}
```



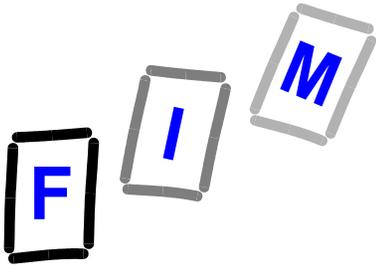
# Datei sequentiell schreiben

```
public static void main(String[] args) throws ... {  
  
    byte d[];  
    FileOutputStream fos;  
  
    fos = new FileOutputStream("c:\\test.dat");  
    for (int i = 0; i < 1024; i++) {  
        fos.write(d);  
    }  
    fos.close();  
  
}
```



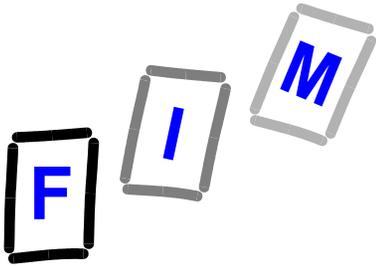
# Wahlfreier Zugriff

```
public static void main(String[] args) throws ... {  
  
    RandomAccessFile rf;  
  
    rf = new RandomAccessFile(  
        "c:\\winnt\\taskman.exe", "r");  
    rf.seek(200); /* Auf Byte 200 positionieren */  
    System.out.println("Byte 200: " + rf.read());  
    System.out.println("Position: " +  
        rf.getFilePointer());  
    rf.close();  
  
}
```



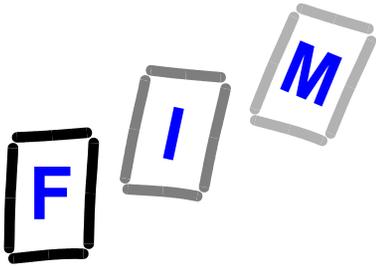
# Verzeichnis durchgehen

```
public static void main(String[] args) throws ... {  
  
    File f;  
    File fs[];  
  
    f = new File("c:\\");  
    fs = f.listFiles();  
    for (int i = 0; i < fs.length; i++) {  
        ...    // fs[i].length() für die Dateigröße  
               // verwenden ...  
    }  
  
}
```



# Links in Dateisystemen (1)

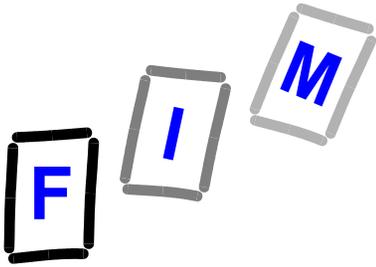
- **Grundsatzüberlegung:** Datei / Verzeichnis / Partition existiert physisch 1x, soll aber mehrmals an verschiedenen Stellen in der logischen Sicht enthalten sein. Es wird nur eine Referenz („Link“) auf die physische Struktur bzw. auf die Beschreibungsstruktur der physischen Daten gespeichert.
- **Windows**
  - **Konzept der „Verknüpfungen“** 
    - » Findet nicht auf Dateisystem-Ebene statt, sondern jede Windows-Verknüpfung ist eine separate Datei mit der Endung „.lnk“, die vom Explorer beim Doppelklick interpretiert wird. Also KEIN richtiger Link.
    - » Daher unabhängig vom Filesystem!
  - **NTFS beherrscht Links auf Dateisystem-Ebene. Wird aber kaum verwendet. Ganze Laufwerke könnten beispielsweise in ein Unterverzeichnis „gemappt“ werden.**



# Links in Dateisystemen (1)

## ▪ UNIX(e)

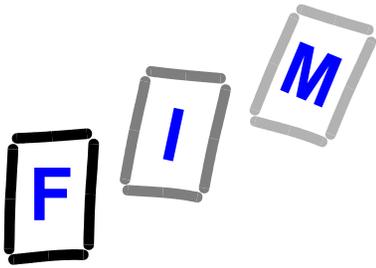
- Es existiert nur ein globaler Verzeichnisbaum („/“ - root). Mittels dem Vorgang des „Mountens“ werden Partitionen und Laufwerke in diesen Verzeichnisbaum eingebunden.
- Das „Temp“ Verzeichnis kann sich z. B. entweder in „/tmp“ oder auch in „/var/tmp“ befinden. Einige Distributionen definieren „/tmp“ beispielsweise als Link auf „/var/tmp“. Somit sind sie zu fast allen Programmen in dieser Hinsicht kompatibel.
- Eine Datei gilt erst dann als gelöscht, wenn alle Links auf die Datei gelöscht sind (Zähler wird mitgeführt)
- Ähnlich wie „Verknüpfungen“ unter Windows sind unter UNIX die SymLinks vorhanden.
  - » Unterschied zu Windows-Verknüpfungen: SymLink wird von Betriebssystem aufgelöst (anders als in Windows, dort muss bspw. Die Verknüpfung vom Explorer aufgelöst werden)
  - » Wird die eigentliche Datei gelöscht, ist sie weg, auch wenn SymLinks existieren. Die SymLinks bleiben bestehen und sind ungültig („Dangling Pointer“ Problem)



# Berechtigungen (1)

## FAT und NTFS

- Unter DOS wurde das Standard-FAT Dateisystem (s. Skript Betriebssysteme VL) verwendet, welches keinerlei Berechtigungsvergabe auf Dateien ermöglicht. Es gibt nur „globale“ Schalter („Schreibgeschützt“, „Archiv“, „Versteckt“, „System“), welche von jedem Benutzer eigenständig geändert werden können.
- Seit Windows NT wird NTFS (New Technology File System) eingesetzt:
  - Sehr feingranulare Berechtigungsvergabe mittels Access-Control-Listen (ACL) möglich. Zu jeder Datei wird Information darüber gespeichert, welcher Benutzer bzw. welche Benutzergruppe welches Recht auf die Datei hat (grant) / explizit nicht hat (deny). Rechtevererbung (Ordner auf enthaltene Objekte bzw. Unterordner) spielt hierbei eine wichtige Rolle.
  - *Problem:* Man sieht stdm. nur, welche Rechte für welchen Benutzer auf welches Objekt gesetzt sind. Den sicherheitsbewußten Administrator interessiert auch, welcher Benutzer überhaupt welche Rechte am System / Dateisystem hat (nicht Objekt, sondern Benutzer steht im Mittelpunkt der Betrachtung) → SAT (<http://www.fim.uni-linz.ac.at/Research/sat/>)



# Berechtigungen (2) NTFS contd.

## NTFS Rechte-Dialog

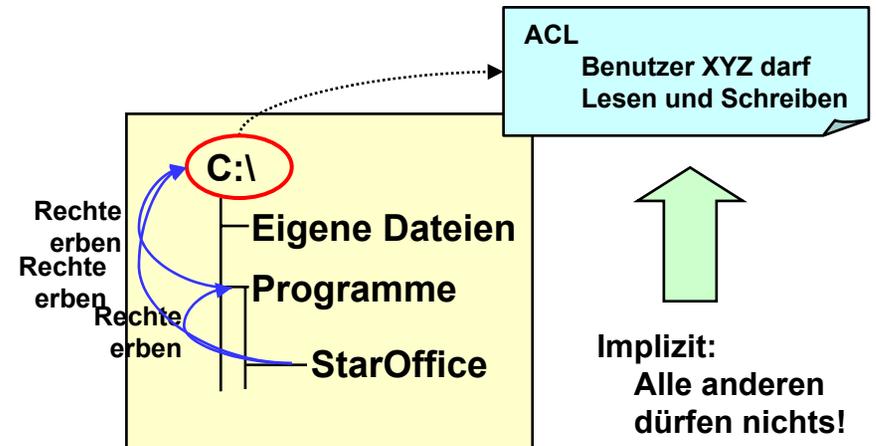
**WER**  
darf

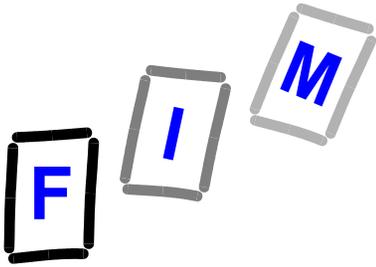
**WAS bzw. WAS NICHT**  
auf

**WELCHES OBJEKT**

Permissions for Administrators	Allow	Deny
Full Control	<input checked="" type="checkbox"/>	<input type="checkbox"/>
Modify	<input checked="" type="checkbox"/>	<input type="checkbox"/>
Read & Execute	<input checked="" type="checkbox"/>	<input type="checkbox"/>
Read	<input checked="" type="checkbox"/>	<input type="checkbox"/>
Write	<input checked="" type="checkbox"/>	<input type="checkbox"/>
Special Permissions	<input type="checkbox"/>	<input type="checkbox"/>

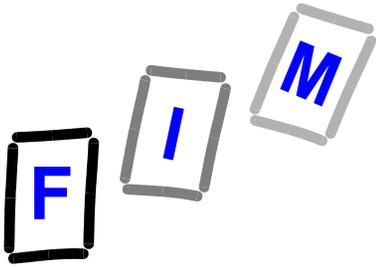
## NTFS Rechte-Vererbung





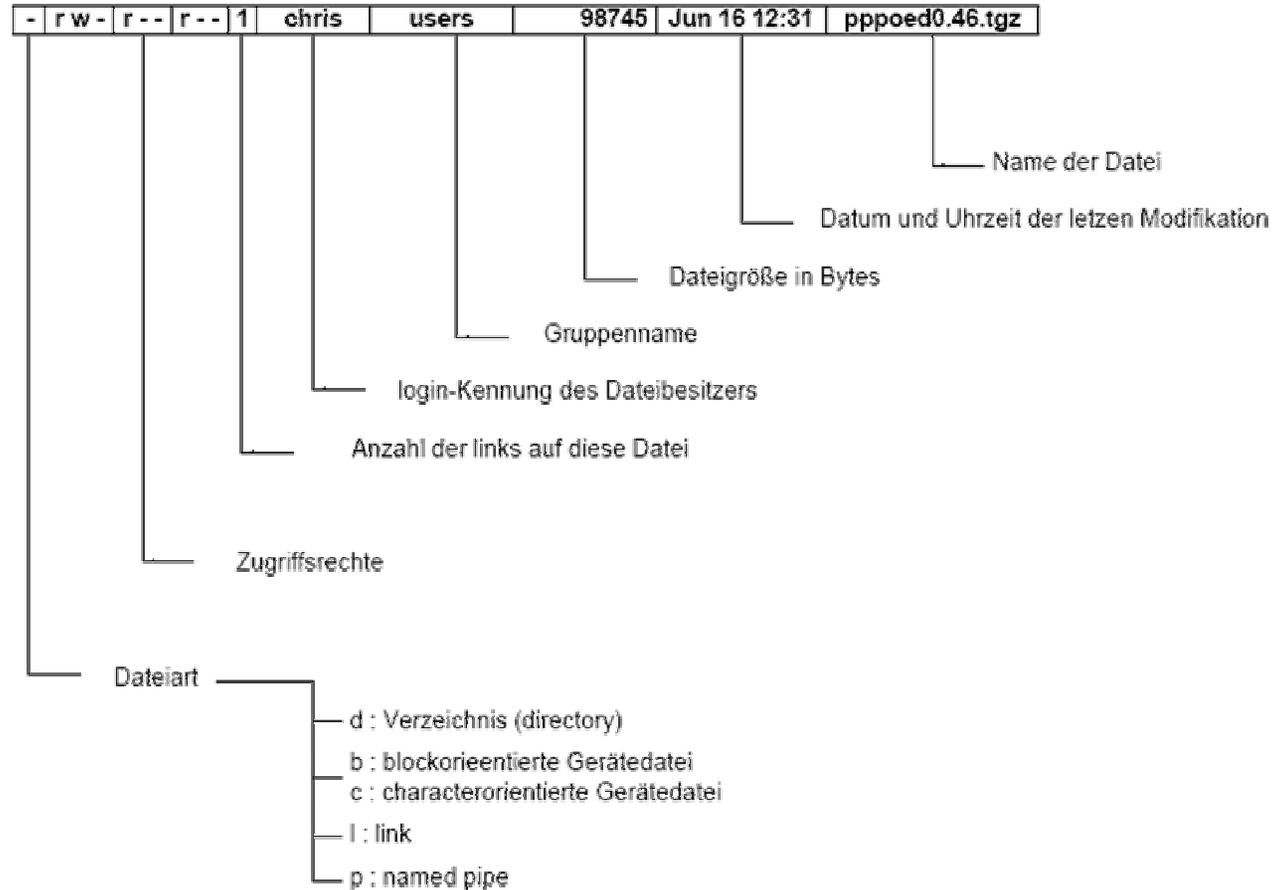
# Berechtigungen (3) ext

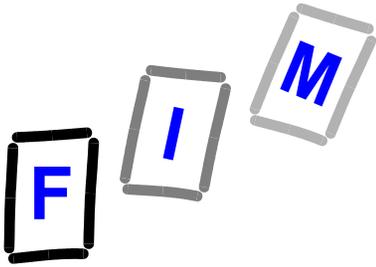
- **Unix bzw. Linux bietet traditionellerweise ein sehr einfaches Rechtekonzept:**
  - **Es gibt Benutzer und Gruppen**
  - **Jeder Benutzer ist einer Primärgruppe und beliebig vielen Sekundärgruppen zugeordnet.**
  - **Es gibt einen speziellen Benutzer („root“), der alle Rechte auf alle (normalen) Dateien hat bzw. sich die Rechte verschaffen kann.**
  - **Jede Datei hat einen Besitzer und eine „besitzende Gruppe“.**
  - **Es gibt lediglich 3 Rechte: read, write und execute**
  - **Eine Kombination dieser 3 Rechte kann getrennt für 3 Personengruppen definiert werden: für Besitzer, für besitzende Gruppe und für alle anderen.**
  - **Weiters existieren einige Spezialbits (Ausführen als Besitzer, als besitzende Gruppe, etc.)**
- **Mittlerweile gibt es auch die feingranulare ACL Unterstützung!**



# Berechtigungen (4) ext contd.

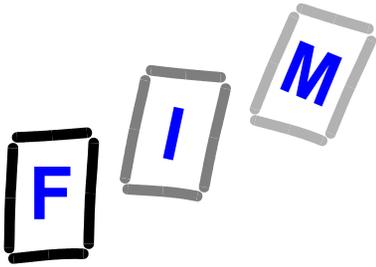
Befehl: `ls -la`





# Journaling (1)

- Eine „Datei“ ist eigentlich nur eine Abstraktionselement. Sie besteht einerseits aus Meta-Information (Dateiname, in welchem Verzeichnis, Größe, etc.) und den eigentlichen Daten, die eine logische Einheit darstellen, physisch aber meist in mehreren Blöcken (Cluster, etc.) auf der Festplatte verteilt liegen.
- D. h. z. B., dass das Erstellen einer Datei aus mehreren Schritten besteht (Anlegen der Metadaten im Directory-Teil, Reservieren der Cluster, etc.). In vielen Filesystemen wird dies in unabhängigen, sequentiellen Operationen durchgeführt → Problem der Inkonsistenz, vgl. Datenbanken! („ScanDisk“ zum Beheben solcher Probleme [meist mit Datenverlust verbunden!] wird wahrscheinlich jeden schon einmal die eine oder andere halbe Stunde genervt haben!)
- *Lösung*: Eine Operation auf das Dateisystem wird in einer „Transaktion“ ausgeführt. D. h., ein Zugriff wird vollständig durchgeführt, oder gar nicht. → Vorteil: Keine Inkonsistenzen, schnelles Wiederherstellen eines konsistenten Zustands beim Neustart des Systems nach vorhergehendem Absturz.



## Journaling (2)

### ▪ Windows

- FAT bis FAT32 unterstützen kein Journaling.
- NTFS (alle Versionen) unterstützen Journaling. WinFS sowieso, da es auf Datenbank basiert.

### ▪ UNIX(e)

- ext, ext2 haben kein Journaling
- Bei ext3 wird eine Journaling Schicht über ext2 gelegt, die das Journaling durchführt (Großer Vorteil der Kompatibilität zu ext2!!)
- JFS, ReiserFS und XFS unterstützen Journaling seit Anfang an.