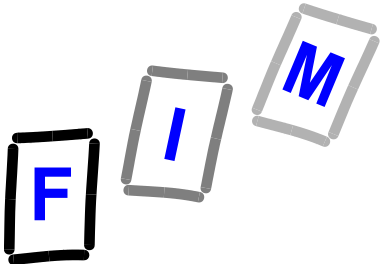
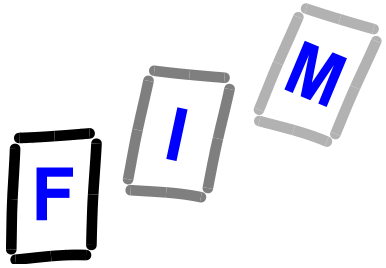


Betriebssysteme

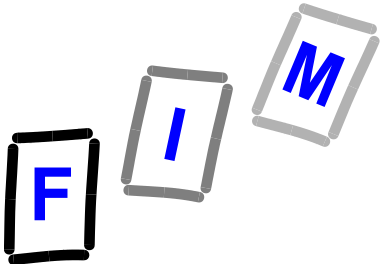
Kapitel D
Klassifikation



Klassifikations-merkmale

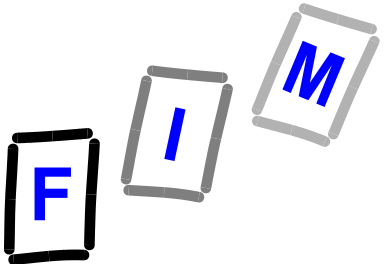
1. **Betriebsarten**
2. **Strategie der CPU-Steuerung**

   (engl.: scheduling strategy)
3. **Technischer Aufbau**
4. **Hauptspeicherverwaltung**
5. **Benutzersicht: Ein- Mehrbenutzerbetrieb**
6. **Gesonderter Gesichtspunkt: Virtuelle Maschinen**



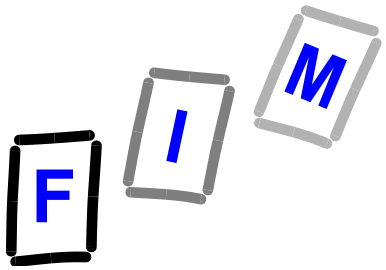
2. Scheduling

- **Scheduling betrifft ganz allgemein Strategien für die Zuteilung von Betriebsmitteln**
  - ➔ **Z.B. Disk-Scheduling**
    - » **Aufgabe, das Betriebsmittel “externer Speicher” möglichst effizient jenen Prozessen zuzuteilen, die darauf schreiben oder davon lesen wollen.**
  - ➔ **CPU Scheduling:**
    - » **Zuteilung der CPU an einen Prozess**
- **Wir konzentrieren uns im Folgenden auf CPU Scheduling.**



Scheduling (Planung)

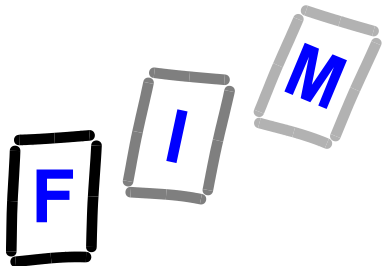
- **Zunächst Unterscheidung in**
  - **➔Langfristiges Scheduling ("Long-Term - Scheduling")**
    - » **Grundsätzliche Reihung von Aufgaben**
    - » **Sie sind hier noch nicht als Prozesse gestartet, sondern sind sozusagen erst in der Phase der Arbeitsvorbereitung).**
  - **➔Kurzfristiges Scheduling ("Short-Term - Scheduling")**
    - » **Betrifft primär die CPU-Zuteilung**



Three Level Scheduling

- Zwischen *Long Term Scheduling* und dem *Short Term Scheduling*
- allenfalls noch eine Zwischenstufe (*Mid Term Scheduling*)
- 3-stufiges Konzept:
  Three Level Scheduling (TLS)
- WICHTIG:
  Scheduling soll (muss?) immer fair sein
  ➔Starvation-Problem !

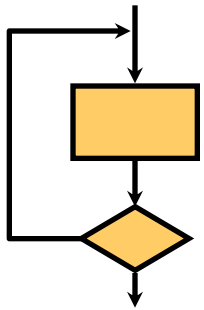
Schematisch



LongTerm

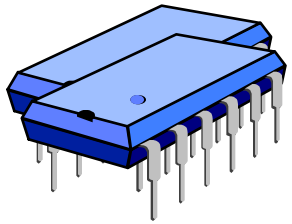
Mid-Term Scheduling

Short Term Scheduling

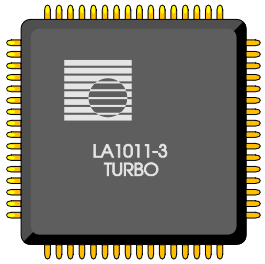


Jobs

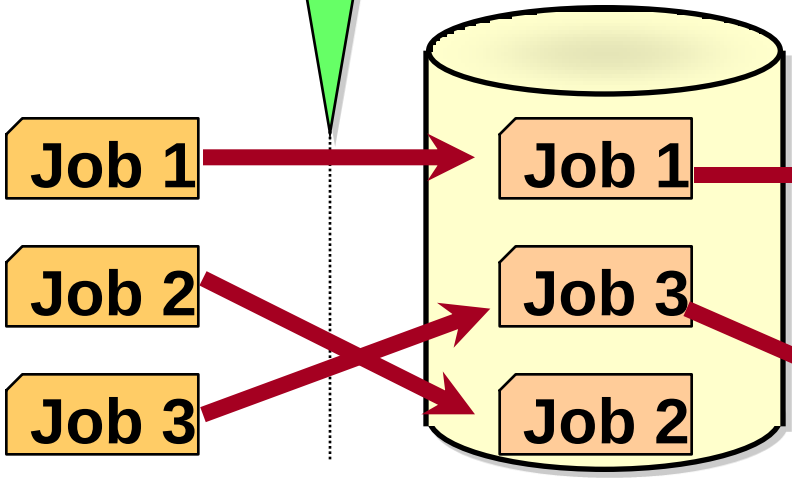
Queues im Massen-speicher

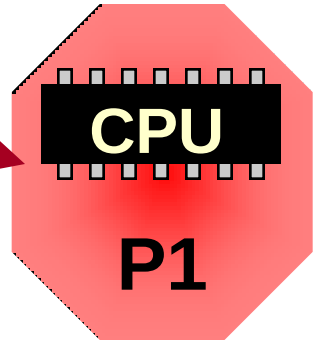


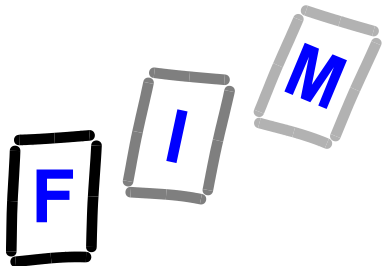
Haupt-speicher



CPU

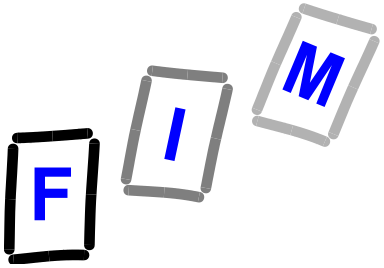






Short Term CPU Scheduling

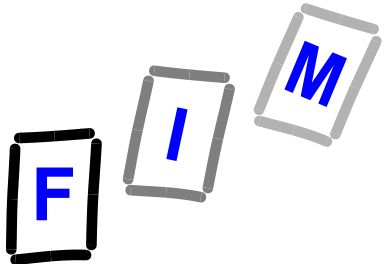
- **Klassifikation von BS richtet sich primär nach dem Short Term Scheduling**
  - ➔ **Darauf beziehen wir uns von nun ab**
- **Zwei grosse Klassen:**
  - ➔ **Non-preemptive Scheduling (nicht unterbrechende Steuerung)**
    - » **Bezeichnung leicht irreführend, wegen “Interrupt”**
  - ➔ **Preemptive Scheduling (unterbrechende Steuerung)**



CPU-Scheduling

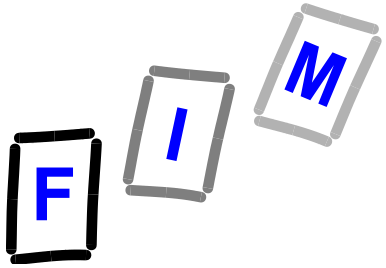
- **Non-preemptive scheduling**
  - ➔ **Erlaubt im Prinzip keine Prozesswechsel**
  - ➔ **Ausnahmen:**
    - » **Interrupts bei I/O**
    - » **Kooperatives Multitasking**
- **Preemptive scheduling**
  - ➔ **Erlaubt preemptive multitasking**

Note: preemption = „Vorkaufsrecht“
Hier verwendet für „Verdrängen“



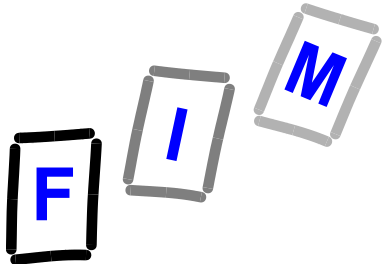
Nicht-unterbrechende CPU-Steuerung

- **Sobald einem Prozess die CPU zugeteilt wird, gibt er sie (im Prinzip) nicht mehr ab, selbst dann nicht, wenn er sie nicht braucht (z.B. bei I/O)**
  - **Kleine Ausnahme: Interrupt-getriebene I/O**
  - **Siehe dazu: Spooling (zB Drucker!)**
- **Beispiele:**
  - **➔Stapelverarbeitungssysteme, MS-DOS**



Kooperative CPU-Steuerung

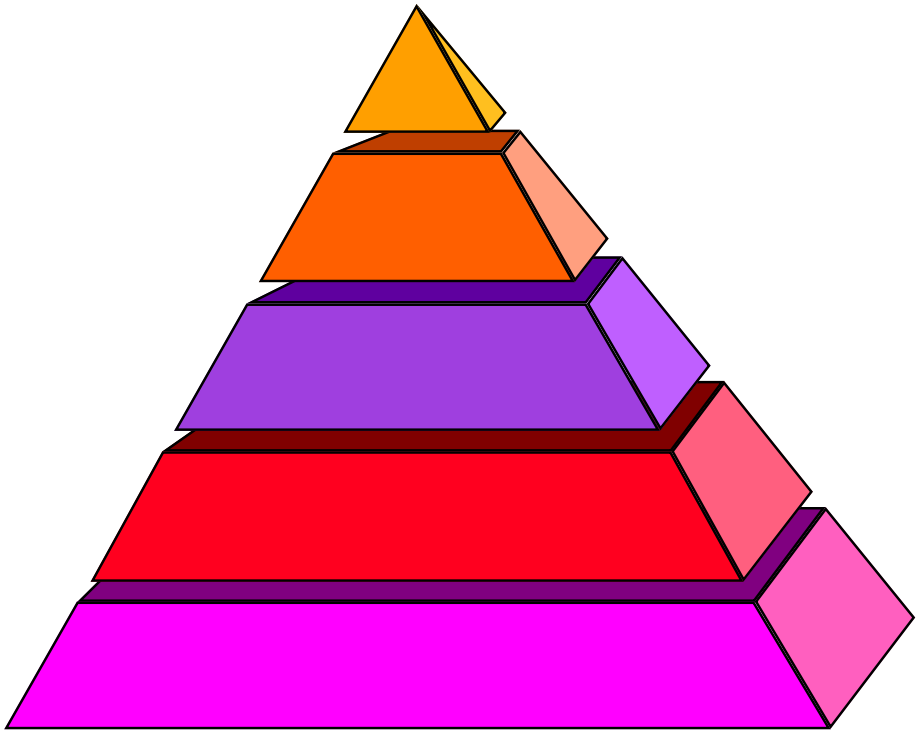
- **Prozess gibt freiwillig unter bestimmten Bedingungen bei definierten Haltepunkten die CPU für einen anderen Prozess frei**
  - ➔ **Zweck: Kooperation mit anderen Prozessen, aber mit minimalem Aufwand bei bestimmten Prozess-Zuständen, auf die man leicht wieder aufsetzen kann**
- **Früher bei**
  - ➔ **Apple-Macintosh OS 9.x**
  - ➔ **Windows 3.x, Win95**
    - » **Mit unterbrechender Steuerung zwischen DOS-Boxes und Windows-Programm-Pool**



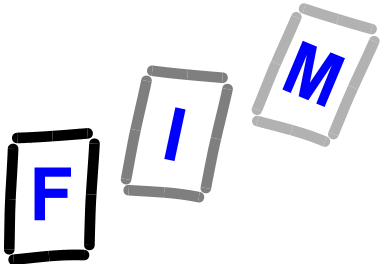
Unterbrechende CPU-Steuerung

- CPU kann einem Prozess „jederzeit“ auf Grund höherer Strategien zu Gunsten eines anderen Prozesses (temporär) entzogen werden
- Beispiele:
  - ➔UNIX, AIX, …
  - ➔Linux, MacOS
  - ➔Windows ab XP

3.Technischer Aufbau



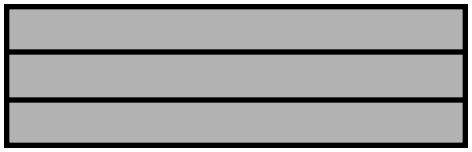
D.3



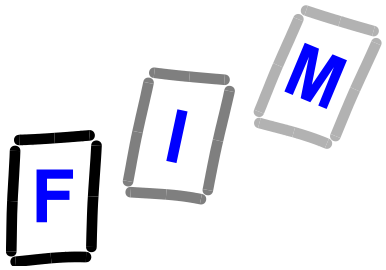
Interne Struktur

Monolithisches BS

Schichten-BS



Client Server Modelle



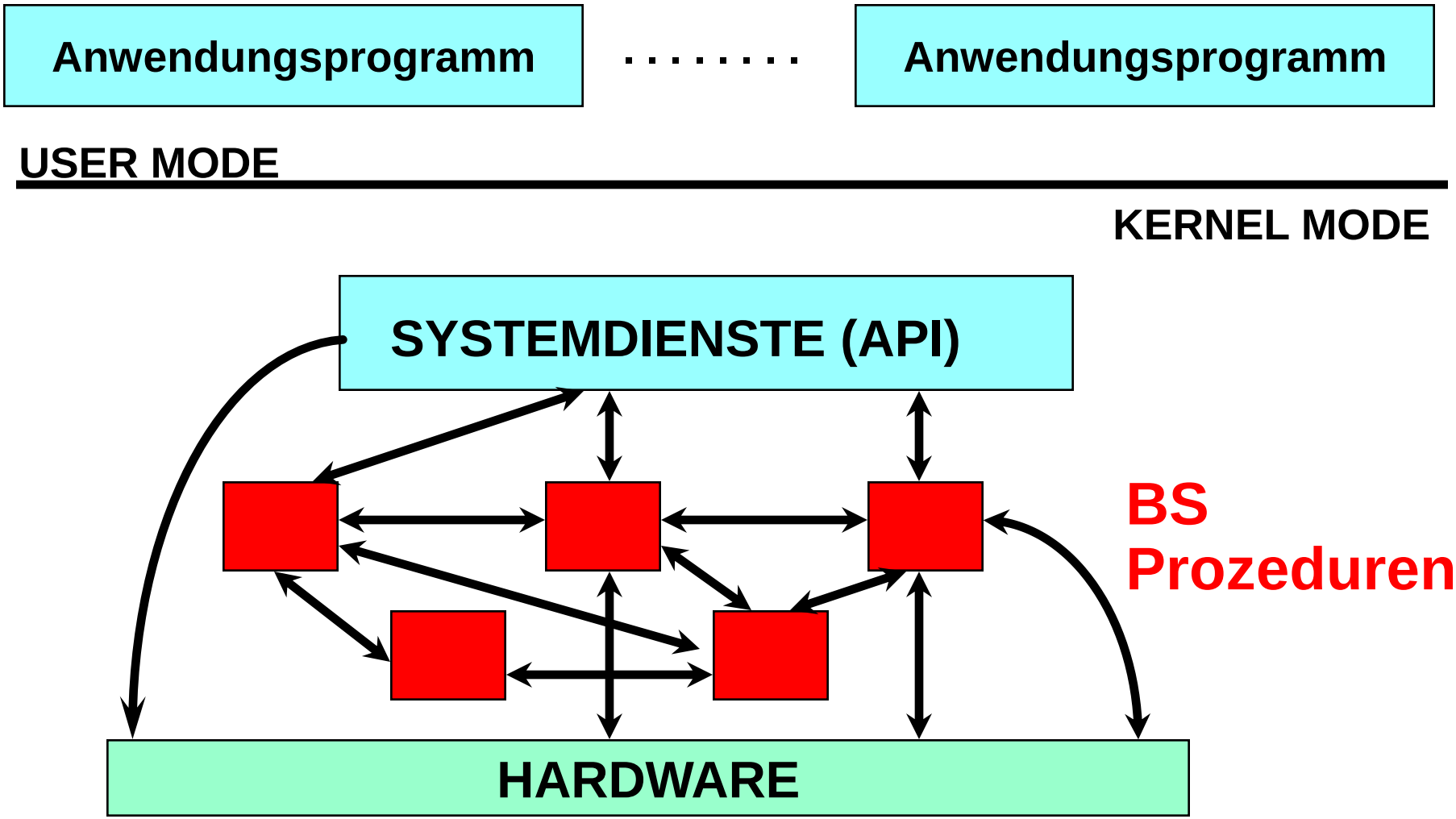
Monolithische Systeme

- **Erinnert an „Spaghetti-Code“**

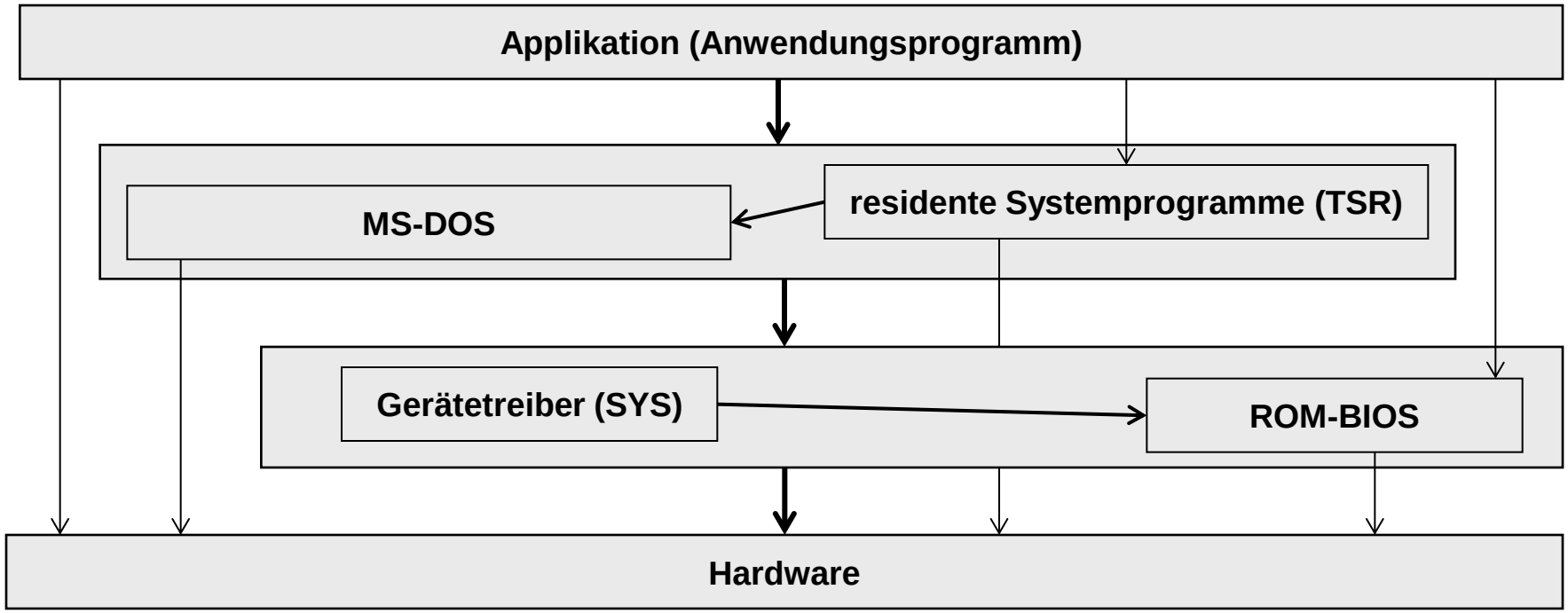
  Genauso wie früher unstrukturiert programmiert worden ist, hat man OS implementiert

- **Trennung zwischen Anwendungsprogrammen und BS-Komponenten vorhanden**
- **Historische Entwicklung: Allmählicher Übergang zu strukturierteren Systemen**

Monolithische Systeme



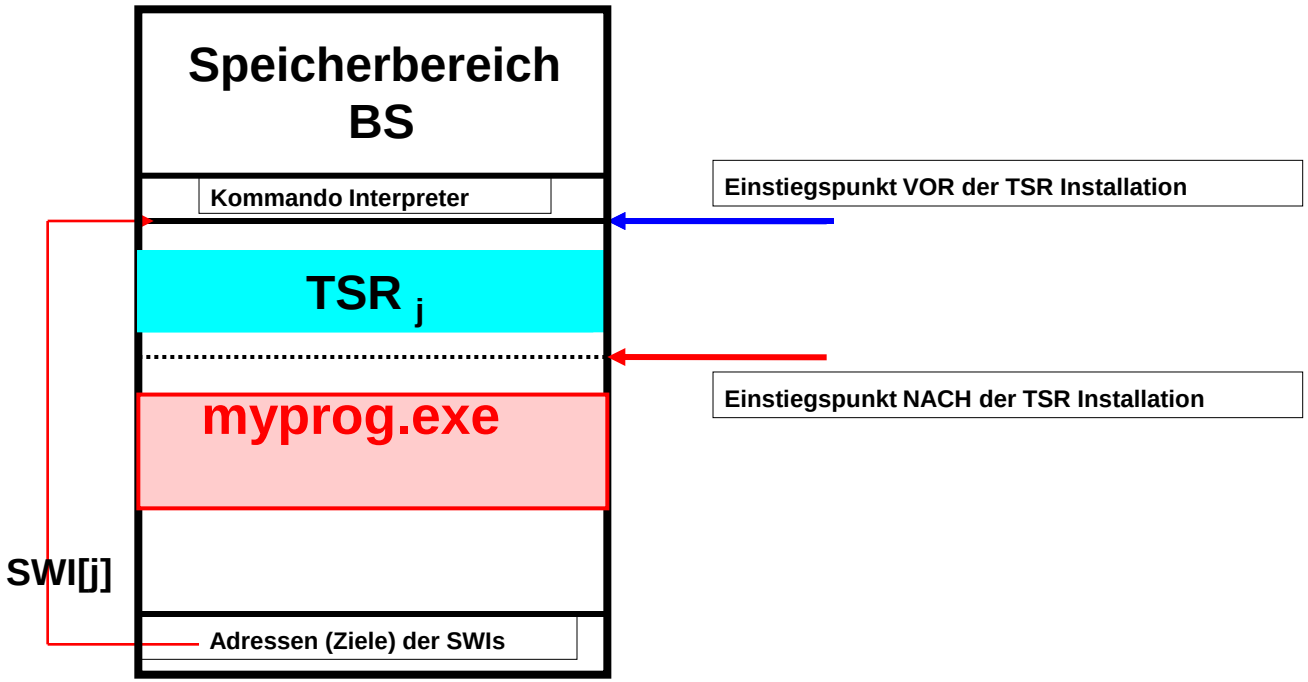
Monolithische Systeme
Übergang zu Schichten

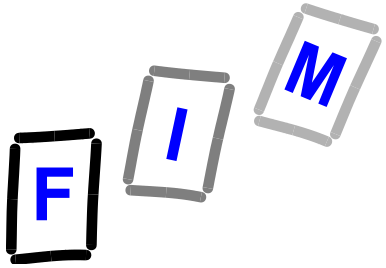


Bemerkung zu TSR (Terminate but stay resident)

- Dienten zur dynamischen Erweiterung von BS
- Heute noch, um Treiber im Hauptspeicher zu installieren
- Werden (meist) über Software-Interruptus aufgerufen:
  SWI[j] zeigt zu TSR „j“

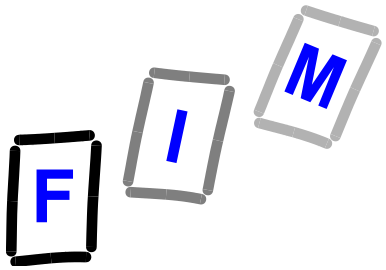
Technik an simplem Beispiel erläutert



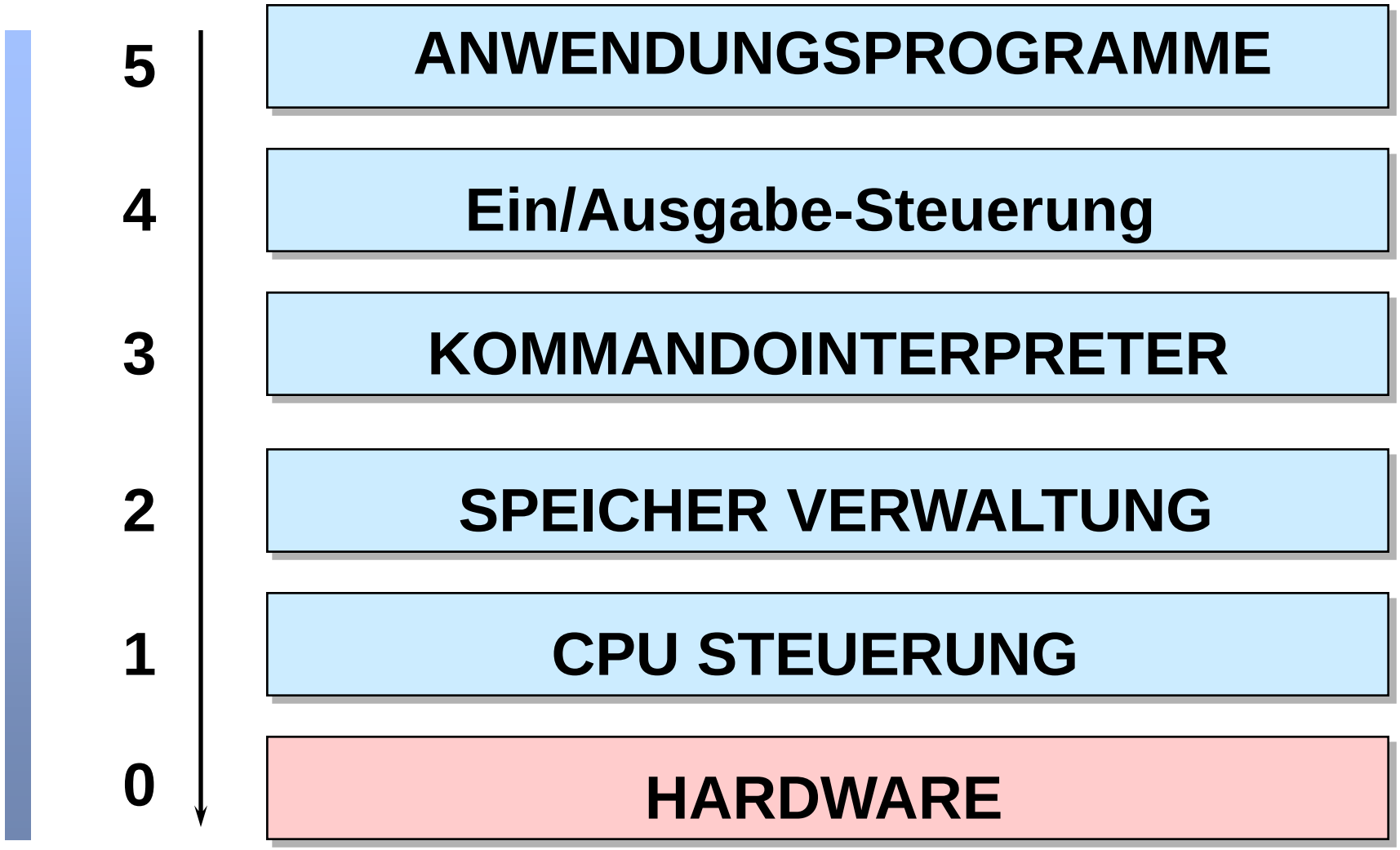


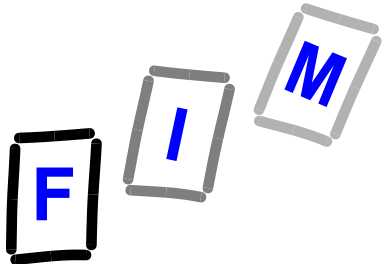
Schichten-Betriebssystem
(engl.: layered OS)

- **Stark beeinflusst durch „*Strukturiertes Programmieren*“**
- ***THE* Operating System (TH Eindhoven,1968)**
- **Strenge Schichten-Struktur („layer“)**
  - ➔ **Schwierig, die Schichten durchzuhalten**
    - » **Gerätetreiber für Massenspeicher sollten über dem Speicher-Management liegen (benötigen Buffer!), aber**
    - » **„Auslagern“ (Virtuelles Speicher-Management) erfordert Zugriff auf Festplatte und benötigt ein File-System**



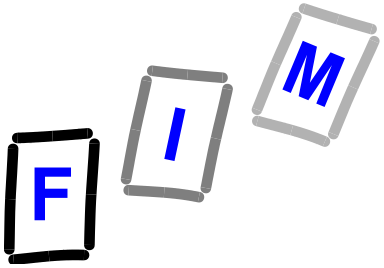
THE Schichtenstruktur





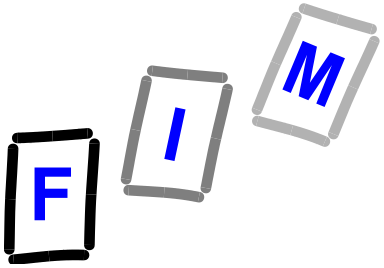
Schichten - BS: Vorteile

- Jeder Ebene im Code wird nur der Zugriff auf Schnittstellen einer niedrigeren Ebene erlaubt
- Fehlersuche / Fehlerbehebung ist einfacher: Beginnt auf der untersten Ebene
- Hinzufügen / Unterteilen von Schichten möglich
- Austausch / Ändern einer Schicht ist ohne Beeinflussung anderer Systemkomponenten möglich
  - ➔ Praxis: Hardware-Treiber, zusätzliche „Treiber“ (zB Verschlüsselung für ein Dateisystem, Tastaturtreiber zum Ersetzen von Kürzeln, Grafik-Filter für Replikation auf andere Displays) einschieben



Schichten des UNIX 4.3.BSD

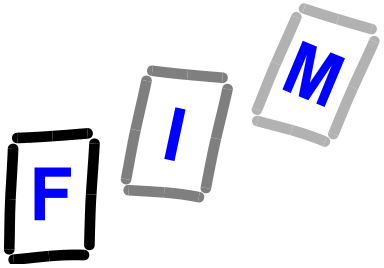
(Benutzer)		
Shell und Befehle Compiler und Interpreter Systembibliotheken		
System-Aufruf Schnittstelle zum Kernel		
Signale Terminalhandhabung Ein/Ausgabesystem Terminaltreiber	Dateisystem Swapping Block I/O system Platten/Bandtreiber	CPU Steuerung Seitenauswechslung Demand Paging Virtueller Speicher
Kernel-Schnittstelle zur Hardware		
Terminalkontroller Terminal	Gerätekontroller Massenspeicher	Speicherkontroller Physischer Speicher



Kernel

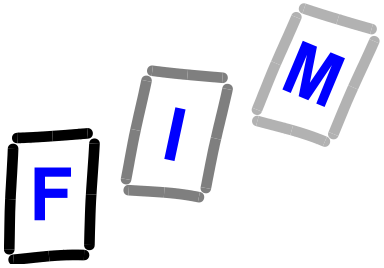
Vereinfachte Beschreibung:

- **Der Teil des BS, der sich ständig im Hauptspeicher befindet**
- **Falls minimal: Micro-Kernel**
- **Zuständig für grundlegend notwendige Tätigkeiten**
  - ➔ **CPU Scheduling, Process Handling**
  - ➔ **IPC (Inter-Process Communication)**
- **Zugrundeliegende Basisstruktur für**
  - ➔ **Client Server Architektur**
  - ➔ **Distributed Operating Systems**



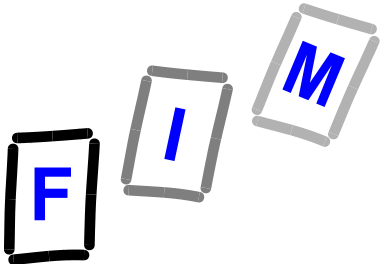
User/Kernel - Mode

- **Aus Sicherheitsgründen wird zwischen zwei Betriebsarten unterschieden:**
  - ➔ **User-Mode**
  - ➔ **Kernel-Mode (auch: System-Mode)**
  - ➔ **Praxis: Auch Hardwaremäßig (CPU) unterschieden**
- **Prozesse die im Kernel-Mode ablaufen, besitzen höhere Rechte und Privilegien**
  - ➔ **Sollten nur wenigen BS-Funktionen vorbehalten sein**
- **Zusätzliche Komplexität bei Virtualisierung**
  - ➔ **Dritter Mode oder müssen sich VM + OS den Kernel-Mode „teilen“?**
  - ➔ **HW unterstützt oft 4 „Ringe“ (=Modi)**



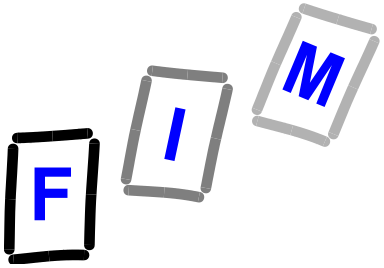
Client Server Modell + (Micro)Kernel

- Teile BS in mehrere Prozesse
  - ➔ Jeder verwirklicht eine bestimmte Menge von Diensten
    - » Jeder gilt als „Server“
    - » Dienste für Speicherverwaltung,
      Dienste zur I/O-Steuerung bei Massenspeicher (Disk)
  - ➔ Jeder Server läuft im Benutzer-Modus
    - » Keine Gefahr, da eigener Prozess + Speicherschutz!
- Clients fordern einen Dienst durch Senden einer Nachricht an den betreffenden Server
  - ➔ Clients: BS-Komponenten oder Benutzerprogramme
- (Micro)Kernel läuft im Kernel-Modus



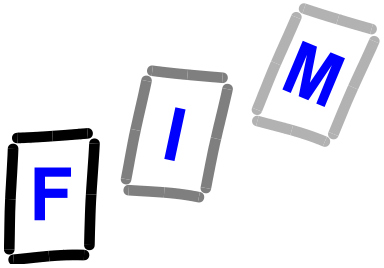
Microkernel

- Erhält Nachrichten vom Client Ci
- Überprüft Rechte etc.
- Übermittelt Nachrichten an den Server Sj
  - ➔ Sj führt den entsprechenden Dienst aus, leitet die Ergebnisse an den Kernel
  - ➔ Kernel gibt die Ergebnisse an Ci weiter, indem er eine weitere Nachricht an Ci sendet
    - » Mitteilung: Dienst ausgeführt, Pufferadresse, etc.
- *Jeder Server läuft als eigener Prozess im Benutzer-Modus (im Idealfall)*
  - ➔ *Zusatzvorteil: Bei Absturz kann er einfach neu gestartet werden*



Microkernel

- ***Jeder Server läuft als eigener Benutzer-Modus-Prozess***
  - ➔**Das ist der Idealfall**
  - ➔**Aus Effizienz-Gründen laufen andere Prozesse ebenfalls im Kernel-Modus**
  - ➔**Daher: Ideale, reine Microkernel-Struktur wird aufgeweicht:**
    - » **Windows XP, W2K … Win7: Gerätetreiber und GDI (graphical device interface) laufen im Kernel-Modus**

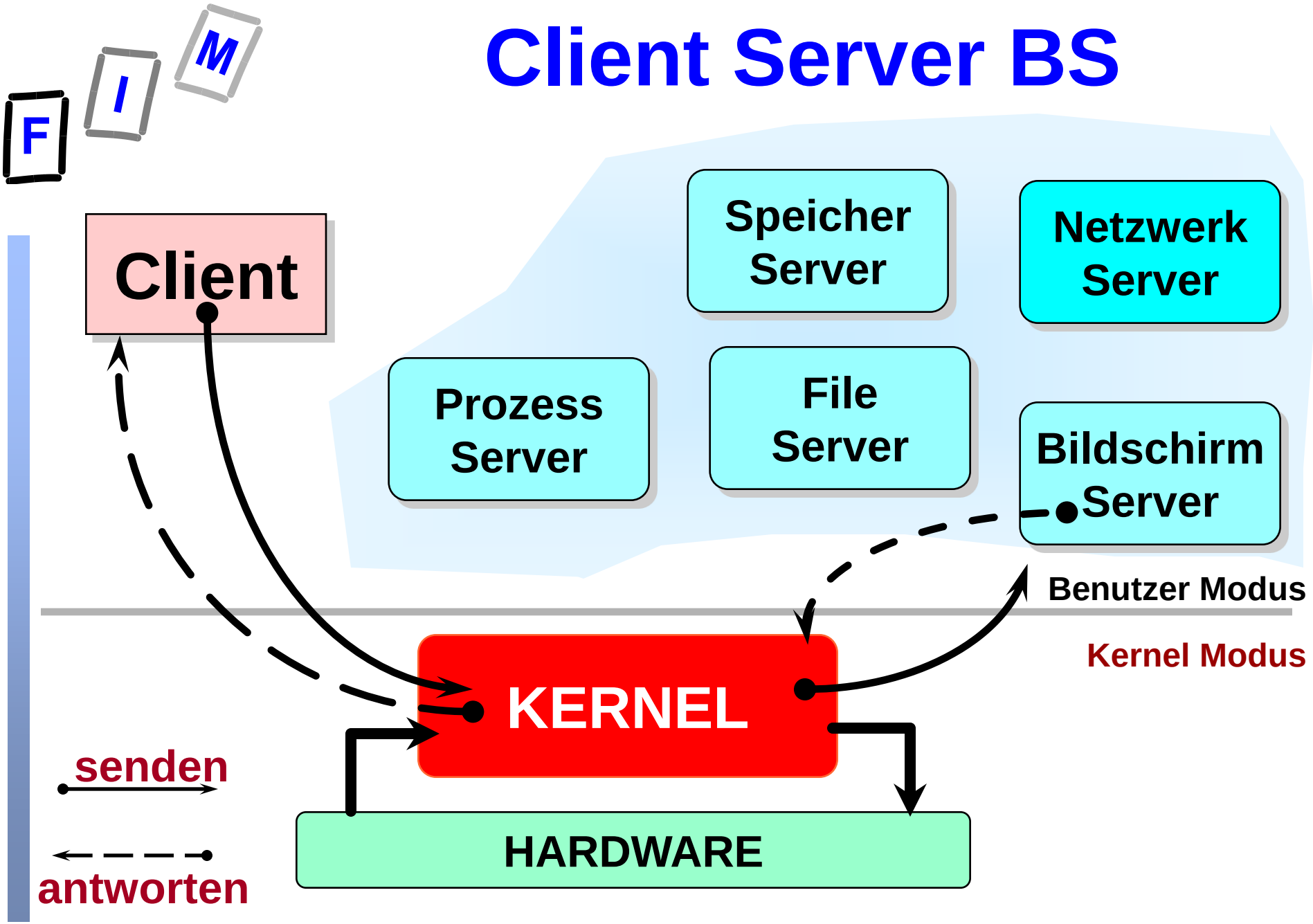


Gefahr!

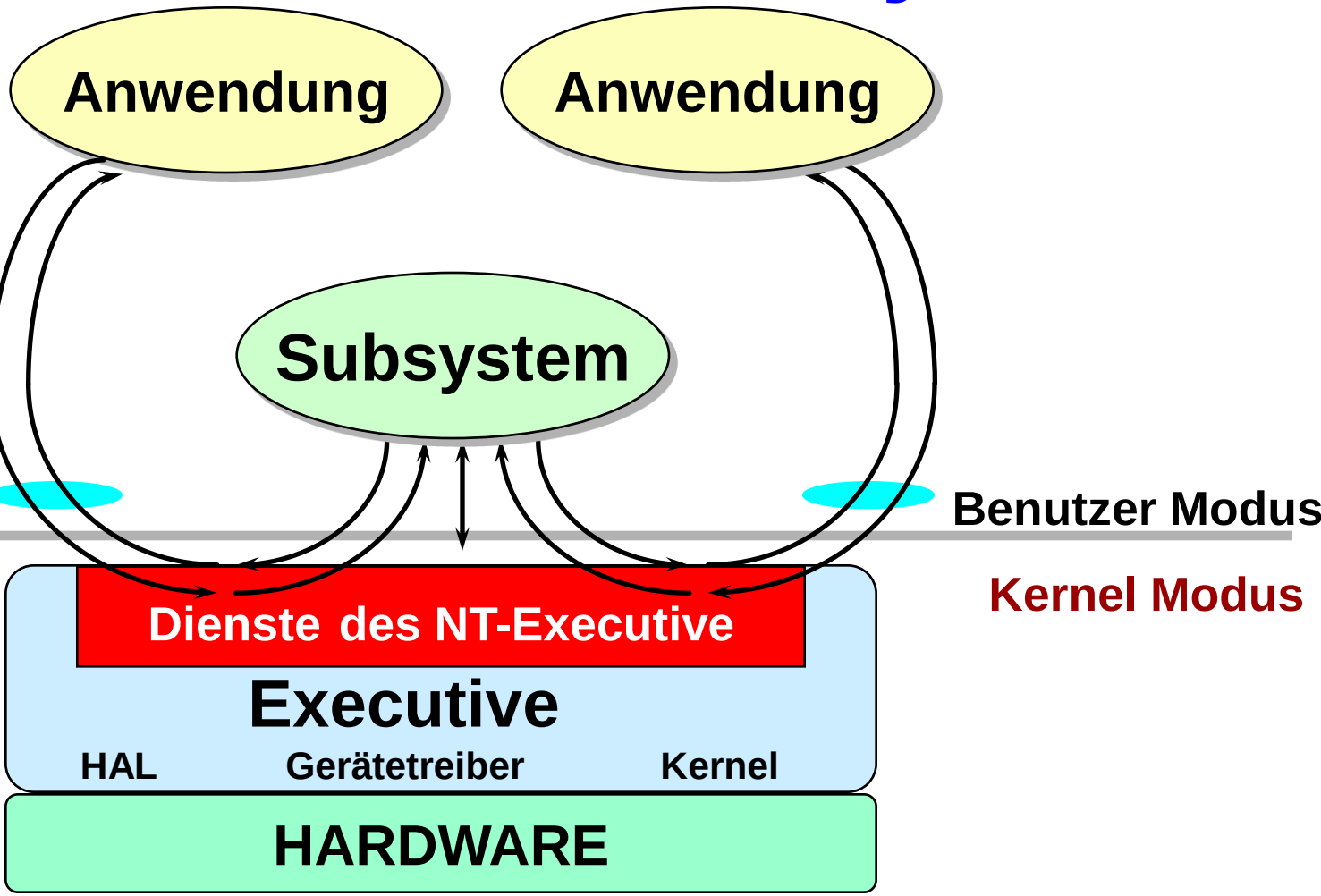
- **Graphiktreiber**
  - ➔**Computerspiele etc**
- **Gerätetreiber allgemein**

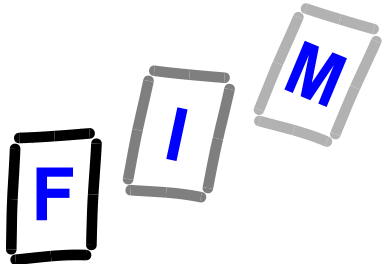
Aus Sicherheitsgründen:
Treiberzertifizierung

Client Server BS



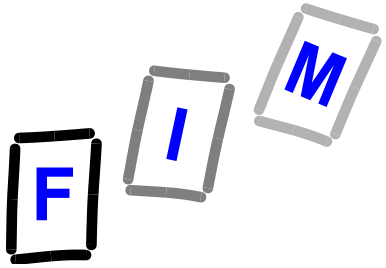
Client Server BS
Autonome Subsysteme





Beispiele (Client/Server)

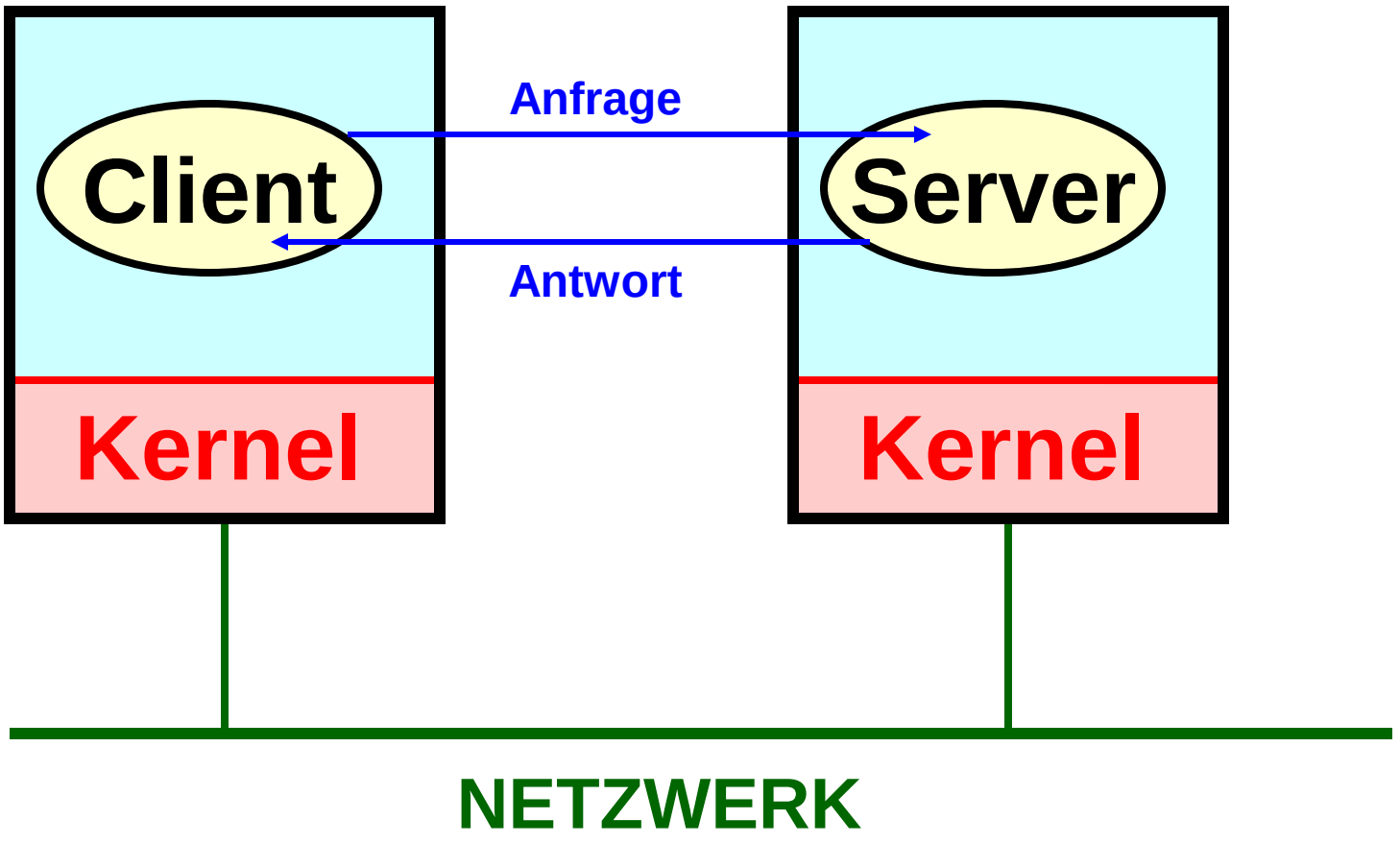
- **MACH Betriebssystem**
  - ➔ **Carnegie Mellon University (CMU), 1986**
  - ➔ **basierend auf: OSF/1 von der Open Software Foundation (OSF), 1989**
- **MINIX von Andrew S. Tanenbaum 1987 (teilweise)**
- **MS-Windows NT, 1992**
- **MS-Windows NT 4 bis Win8 & Server 2012**
  - ➔ **CS Architektur**
  - ➔ **Kein Microkernel**
    z.B. Treiber und Grafik (GDI) bleiben beim Kernel



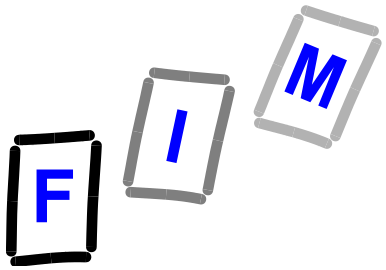
Client-Server-BS: Vorteile

- **Komponenten des BS sind klein und in sich abgeschlossen**
- **Ein einzelner Server kann ausfallen (und neu gestartet werden), ohne den Rest des BS abstürzen zu lassen**
- **Verschiedene Server können auf verschiedenen Prozessoren / Computer laufen**
  - ➔ **Wichtiger Schritt in Richtung „Verteilte Systeme“**

Vereinfachtes CS-Model über Netzwerk



Nachrichtenübertragung erfolgt letztlich durch die Kernels !!

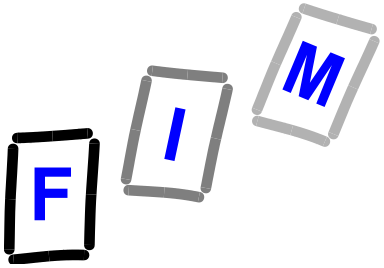


Verteilte Betriebssysteme (distributed OS)

- **Client Server Konzept führt zu VERTEILTEn SYSTEMEn**
- **A.S. Tanenbaum:**

  „ein Verteiltes System ist eine Sammlung von unabhängigen Rechnern, die aus der Sicht der Anwender aber wie ein einziges Computersystem arbeiten“
- **Vergleiche dazu: Cloud!**

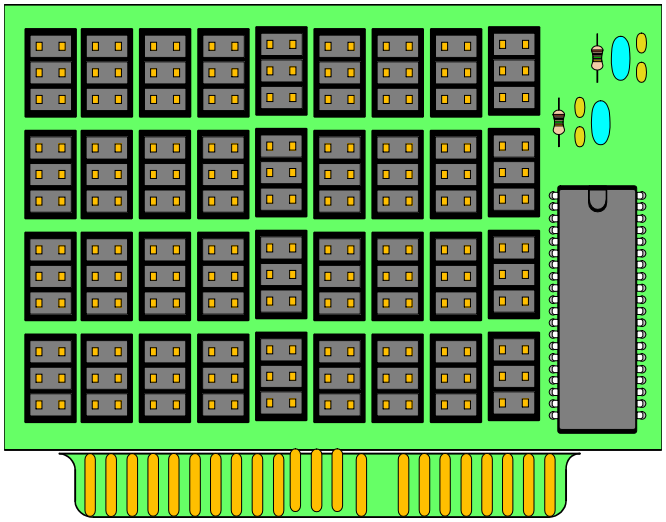
  ➔ **Sieht eine solche (Wann? Für wen?) wie ein einziges Computersystem aus?**



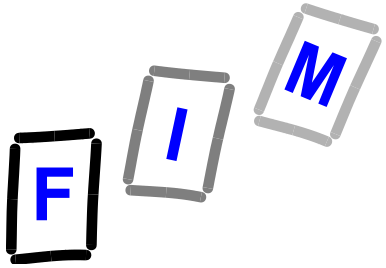
Verteiltes BS

- **BS ist unterteilt in autonome, in sich abgeschlossene Subsysteme**
- **Diese Subsysteme kommunizieren miteinander mittels Nachrichten**
  - ➔ **Message passing**
- **Die physische Position ist nicht mehr so wichtig, die Subsysteme können auch auf Netzwerkknoten liegen**
  - ➔ **Voraussetzung: Breitbandverbindung verfügbar**

4.Hauptspeicher-verwaltung



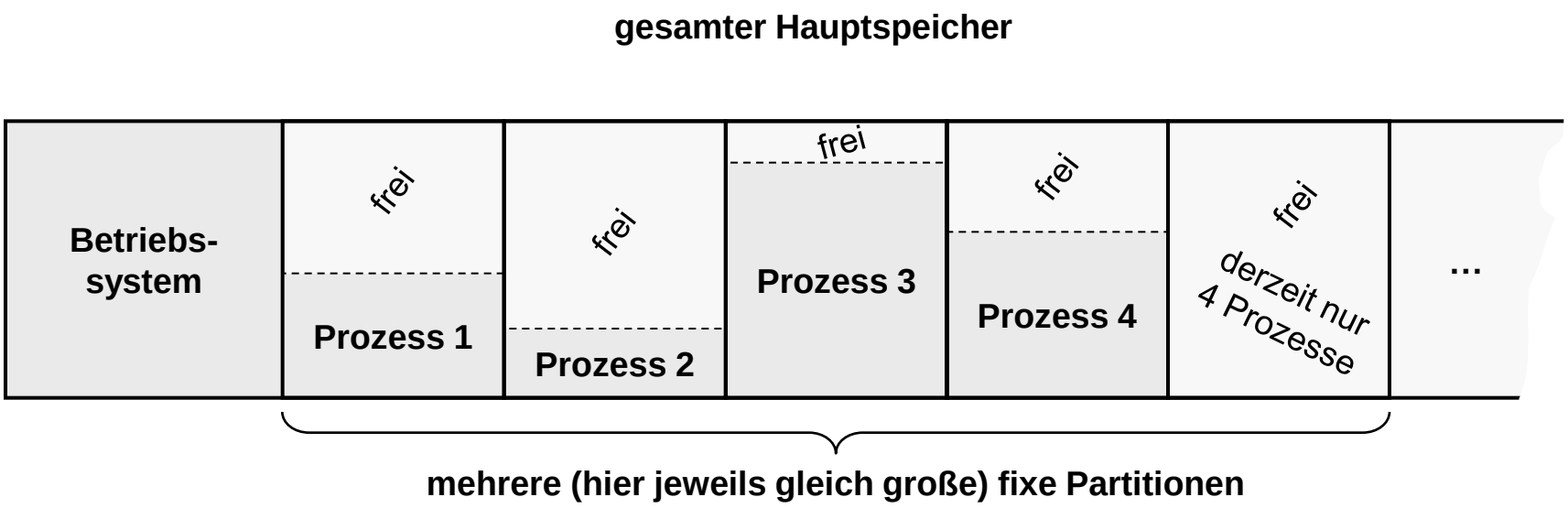
D 4



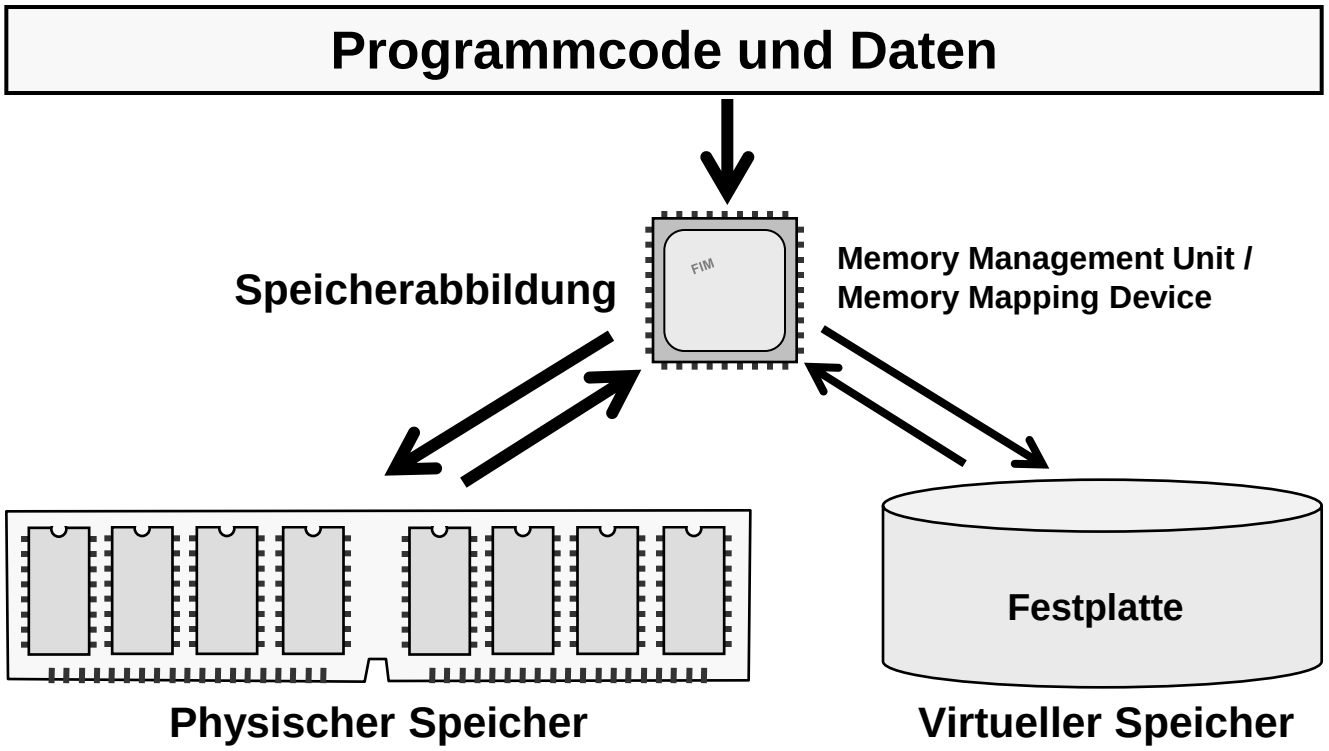
Hauptspeicherverwaltung

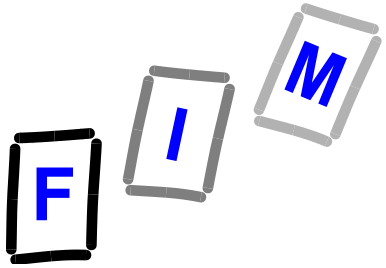
- Wird ein gesondertes Thema sein
  - ➔Realspeicherverwaltung
  - ➔Virtueller Speicher
    - » Abbildung Virtueller Speicher -> Realspeicher
    - » z.B. durch Paging
  - ➔Schutzmechanismen
  - ➔Gemeinsamer Speicher für parallele Prozesse

Einfache Realspeicheraufteilung



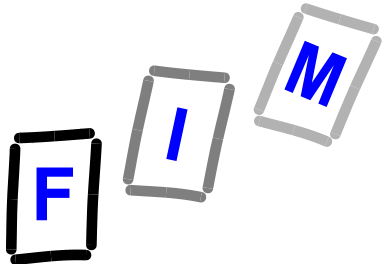
Trennung
Realer/Virtueller Speicher





5. Benutzersicht

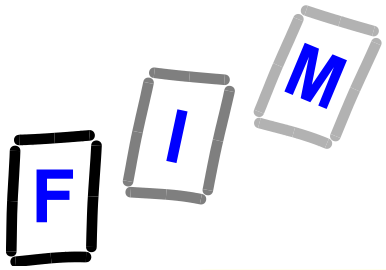
- Vgl.: Ausführungen im Kapitel A (1)
- Art des User-Interface
  - ➔ Zeilenorientiert
  - ➔ GUI
- Zielgruppe und Zweck:
  - ➔ Einbenutzer: Workstation
  - ➔ Mehrbenutzer:
    - » Großrechner
    - » Server allgemein



6. Virtuelle Maschinen

- ***Aus der Sicht eines Anwenders ist eine virtuelle Maschine ein Computer, der (teilweise) durch ein Programm simuliert wird***

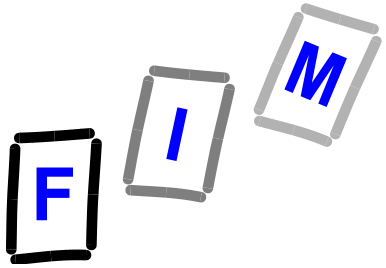
- **Konzept aus Software Engineering**
  - **➔Top down-Design**
  - **➔Schichtenweiser Entwurf**



Virtuelle Maschinen

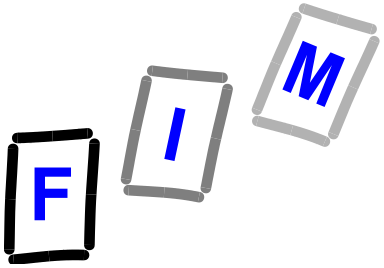
```
public class Keyboard
 {. . . . . .
 public boolean keyPressed();
 public char readKey();
 . . . . . .
}
```

- **Angesprochene Tastatur ist *virtuell*,**
- **Entstehendes Programm wird hardwareunabhängig.**
- **Bei Bedarf sind bloß die Implementierungen der Methoden an eine geänderte Tastatur anzupassen.**



Virtuelle Maschinen
API

- **BS verbirgt die darunter liegende Hardware, und**
- **bietet den Anwenderprogrammen in seiner Schnittstelle zu diesen eine Sammlung von Prozeduraufrufen.**
- **API: Application Programming Interface**
  - ➔ **Systemaufrufe (system calls) über Unterprogramme (Prozeduren)**
  - ➔ **Systemaufruf mittels Software Interrupt**



API
Beispiele

- **Ausgabe von Text / Bitmaps auf den Bildschirm / Cursor setzen**

  VioSetCursPos(row, column, 0)

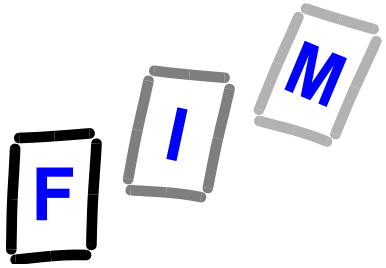
- **Verschiedene File-Operationen**

  (file open, file close, file read, file delete, ...)

  rc= DosRead(Handle, Buffer, bufflen, BytesRead)

- **Starten eines parallelen Prozesses / Threads**

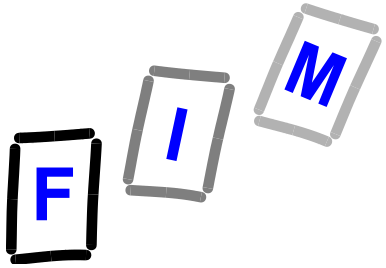
  rc= DosCreateThread
  (procname, &threadID, stackptr)



API
Windows, GetSystemInfo

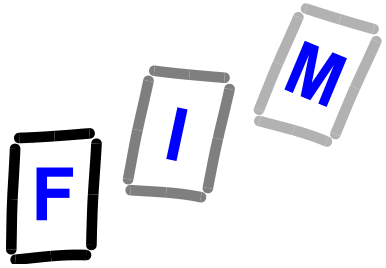
Windows, API Proc GetSystemInfo

```
#include <windows.h>
#include <stdio.h>
void main()
{
   SYSTEM_INFO siSysInfo;
  GetSystemInfo(&siSysInfo);
  printf("  Number of processors: %u\n",
        siSysInfo.dwNumberOfProcessors);
}
```



API
Linux mit SWI

- **Beispiel Linux:**
  - ➔ **SWI(128) = SWI(80H)**
  - ➔ **Gestattet Zugriff auf Linux-Funktionen**
    - » **ZB Datei öffnen/schließen/lesen/schreiben…**
- **Linux selbst verwendet Softwareinterrupts**
  - ➔ **z.B. für Zugriff auf BIOS (selten nötig)**



API
Virtual Machine

Die API kann man sehen als

„set of commands“
***einer* Virtual Machine**

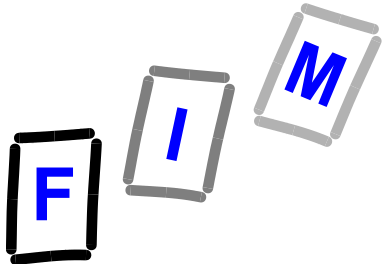
und das BS implementiert / unterstützt diese Befehle, sodass sie von

➔ Anwendungsprogrammen

➔ Betriebssystemkomponenten (hier: System Calls)

genutzt werden können.

●Allgemein: Damit Clients das API nutzen können

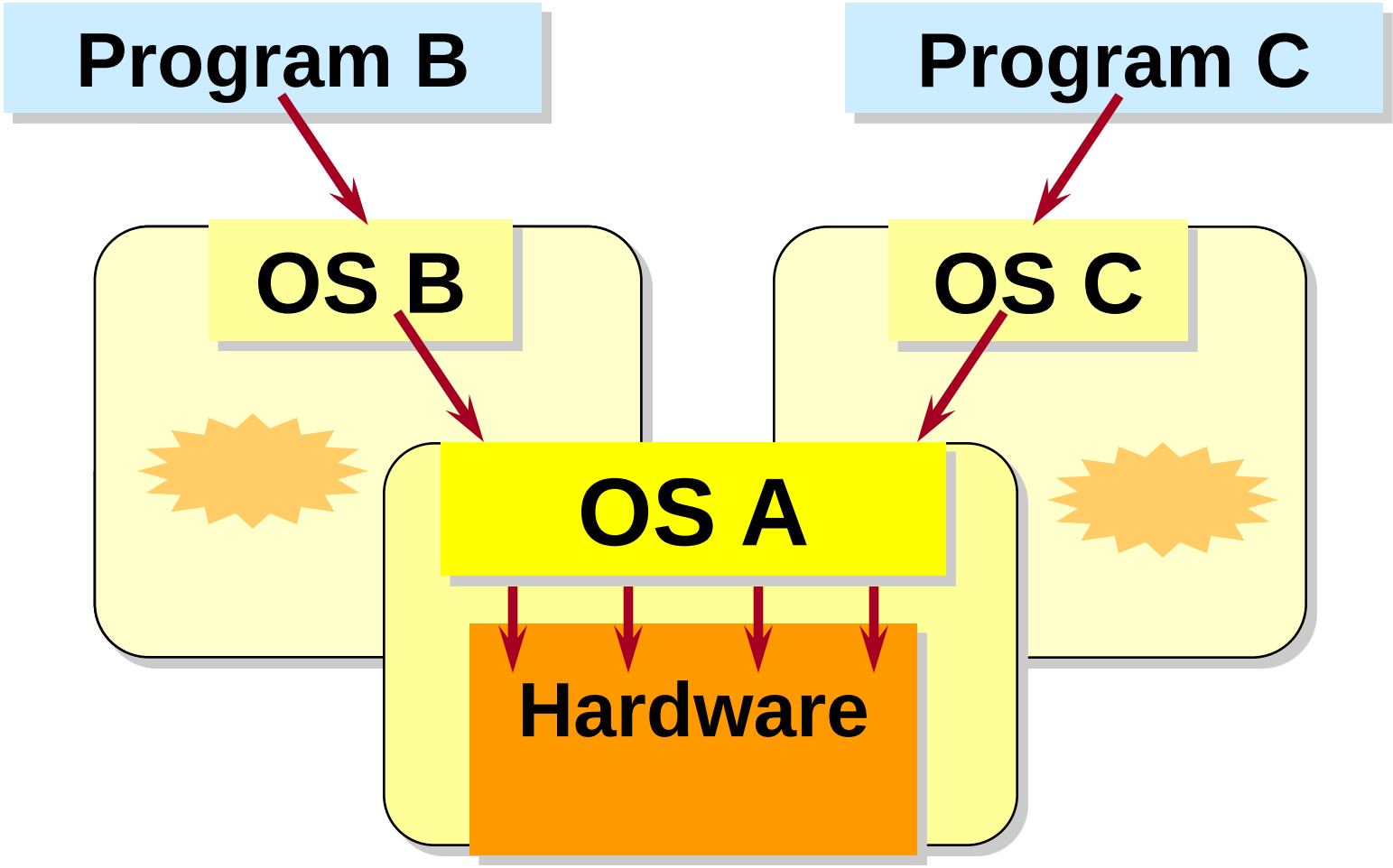


BS ist selbst ein Programm

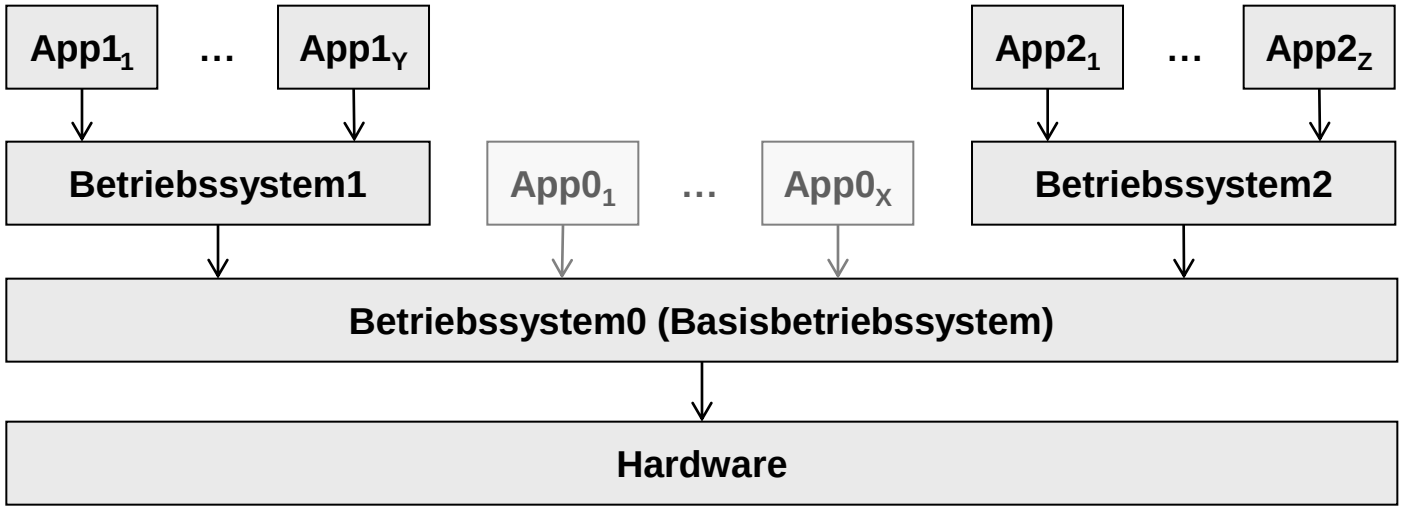
Mehr als ein Betriebsmodus/-system gleichzeitig

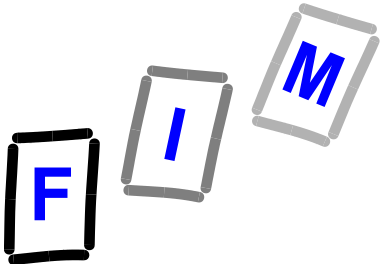
- **Systemnahes Basissystem für gemeinsame Aufgaben *(siehe: kernel)***
- **Eine Schicht darüber: Für jeden Modus ein geeignetes BS (oder ein BS eines Drittherstellers, ...)**
- **Aktuell: z.B. Software von *vmware.com* (Linux + Windows)**
- **!! Multiboot Feature ist etwas völlig anderes**

Mehrere BS gleichzeitig aufgesetzt auf einem Basis-BS



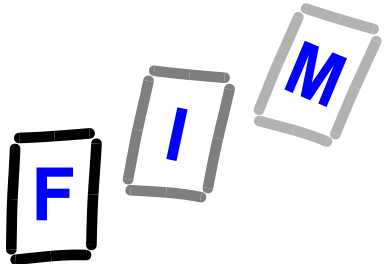
Virtualisierung Idee





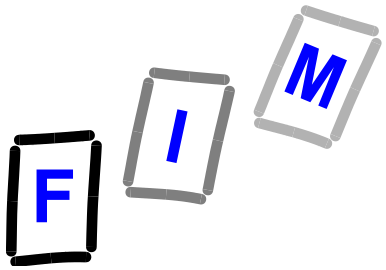
Virtualisierung

- **Vorige Folie zeigt die Idee nur grundsätzlich**
  - ➔ **Details: Siehe VO über System Administration**
    - » **Kapitel über Virtualisierung, zB bei Servern**
- **Hypervisor / Virtual Maschine Monitor (VMM)**
  - ➔ **Zur Umsetzung der Virtualisierung**
  - ➔ **Damit verbundener Steuerungsaufgaben**
  - ➔ **Kann verstanden werden als spezielles BS mit dem Zweck, virtuelle Maschine auszuführen/zu steuern**



Hypervisor
Virtual Machine Monitor

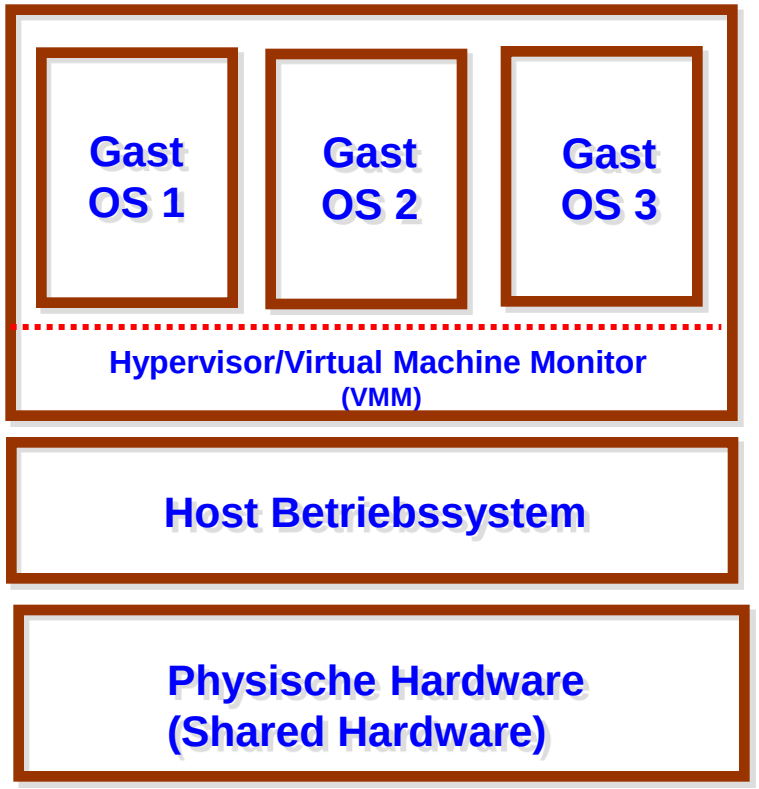
- **Erlaubt es mehreren Betriebssystemen auf einer Hardware Plattform zu laufen**
- **„Verwaltungszentrale“ der virtualisierten Maschine**
  - ➔ **Verwaltet alle Ressourcen die unter den Gästen geteilt werden müssen**
- **Unterscheidung nach Installationstyp**
  - ➔ **Bare-Metal (Typ-1)**
    - » **Hypervisor setzt direkt über physischer Hardware auf und läuft im Kernel, bzw. privileged Mode**
    - » **Gäste laufen auf einem höheren Level über dem Hypervisor**
  - ➔ **Hosted (Typ-2)**
    - » **Hypervisor läuft als User-Level Programm im Host OS**
    - » **Interaktion mit der Hardware über Host OS mittels Kernel-Module und Device-Driver**

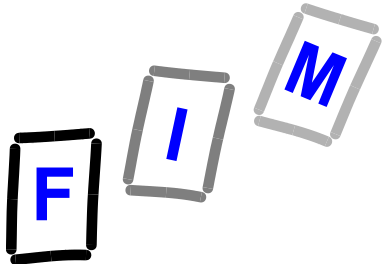


Typ-2 „Hosted“

- Hypervisor läuft als „normale“ Applikation in einem Host OS
- Host OS stellt Geräte Treiber zur Verfügung und verwaltet physische Ressourcen
  - ➔ Zugriff auf Hardware über Device Driver

✓ Vorteil: Einfache Integration in bestehendes, unvirtualisiertes Betriebssystem



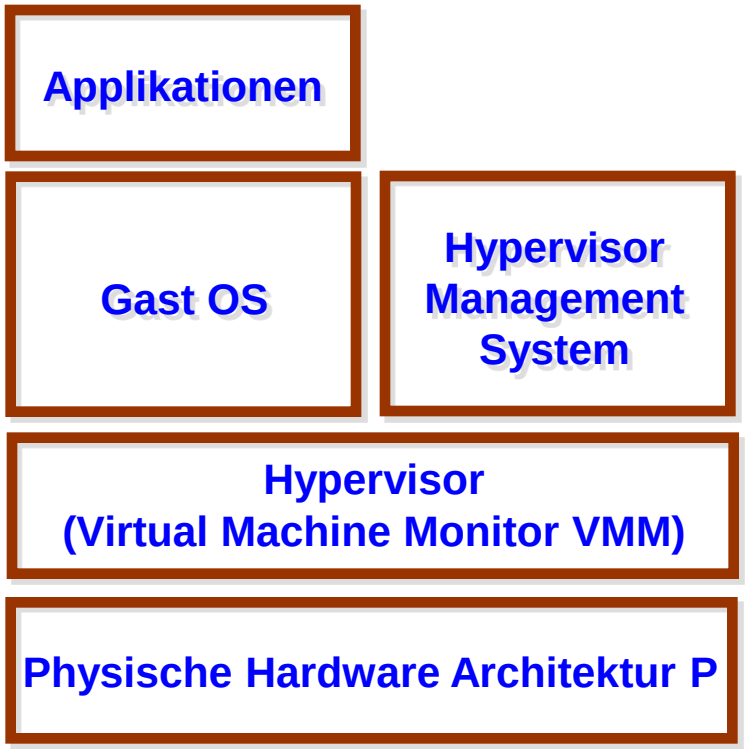


Typ-2 „Hosted“

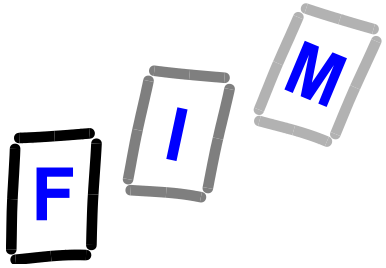
- **Vertreter:**
  - ➔ **VMware Workstation**
  - ➔ **Microsoft Virtual PC**
  - ➔ **VMware Server**
  - ➔ **Microsoft Virtual Server**
  - ➔ **Oracle Virtual Box**

Typ-1 / Bare-Metal Hypervisor

- Spezieller, schlanker Kernel/Betriebssystem mit Virtualisierungsfunktion
- Läuft direkt auf der Hardware
- Hypervisor Management Interface zur Verwaltung der virtuellen Hardware und der Gäste

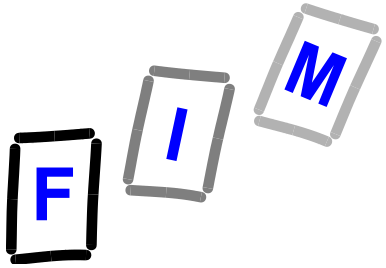


in der Abb nur 1 Gast OS



Typ-1 / Bare-Metal Hypervisor

- **Vertreter**
  - ➔ **z/VM von IBM**
  - ➔ **VMware ESX(i) Server**
  - ➔ **Xen Server**
  - ➔ **Microsoft Hyper-V**



Typ-1 / Bare-Metal Hypervisor

- **Besondere Vorteile**
  - ➔ **Speziell für gegebene Hardware entwickelt**
  - ➔ **Nützt Architekturerweiterungen aus**
    - » **Intel–VT („Vanderpool Technology“)**
    - » **AMD-V („Pacifica“)**
  - ➔ **Bare Metal Hypervisor kann auf im Gast-OS verwendetes CPU-Scheduling eingehen**
    - » **Daher auch für Real Time Lösungen (mit Einschränkungen) geeignet**

Weitere Varianten

- **Paravirtualisierung**
  - ➔ **Performance-Überlegungen**
    - » **Teile des virtualisierten OS werden speziell adaptiert/verändert**

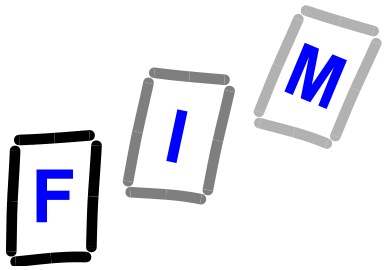
      ZB um ohne (physikalischen!) Kernel-Mode auszukommen oder um direkter auf HW zugreifen zu können (Netzwerkschnittstellen, Grafikkarte etc.)
  - ➔ **Spezialhardware**
    - » **Bestimmte HW wird an eine virtuelle Maschine „durchgereicht“, sodass sie nicht virtualisiert werden muss (bzw weil sie es nicht kann – kein virtueller Treiber!)**
- **Kernel-basierte Virtuelle Maschinen (KVM)**
  - ➔ **Wird in Kernel des Linux integriert**
  - ➔ **Firma Qumranet, (->QEMU)**

Emulation

vers.Virtualisierung

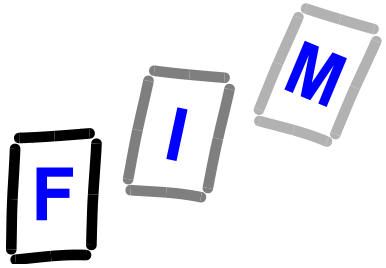
- **Unterschied zur Virtualisierung**
  - ➔ **Emulation ist softwaremäßiger Nachbau der Hardware**

    vgl.: HAL = Hardware Abstraction Layer
  - ➔ **im Beispiel**
    - » **2 unterschiedliche Hardware Systeme simuliert**
    - » **Darauf läuft jeweils das „ursprüngliche“ OS**



Emulation Beispiel

- DOS-Box für 16 Bit Generation
  - ➔Verhalten eines „16 –Bit Rechners“
    mit aufgesetztem MS-DOS wird funktionsgleich nachgebaut
  - ➔Unabhängig von der Hardware
  - ➔Exakt ident im Verhalten
- Spielkonsolen: Super Nintendo, Arcade-Automaten, PlayStation2, …



Emulation
vs. Simulation

- **Unterschied zur (Computer-) Simulation**
  - ➔ **Ein komplexes System wird durch ein Modell nachgebildet**
    - » **Dabei „nur“ das „I/O“ Verhalten**
  - ➔ **Modell durch Abstraktion**
    - » **Nicht relevante Details werden ausgelassen oder können nicht berücksichtigt werden, weil reales System z.B. nicht genau bekannt**