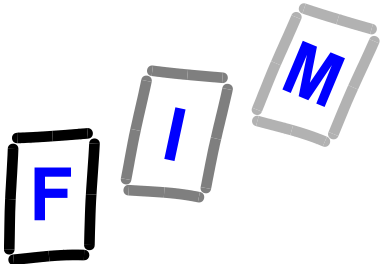


Betriebssysteme

Teil C: Betriebsarten

Zielgruppe

- **Ein Betriebssystem (bzw. das von ihm verwaltete Computersystem)**

  kann konzipiert sein

  ➔ **dezidiert für nur einen einzigen Benutzer (vgl.: Workstation, Arbeitsplatzrechner)**

  » **Einbenutzerbetrieb (single user mode),**

  Das heißt nicht unbedingt, dass auch tatsächlich jemand davor sitzt; zB bei Meß- oder Steuerungsrechnern!

  ➔ **für mehrere Benutzer**

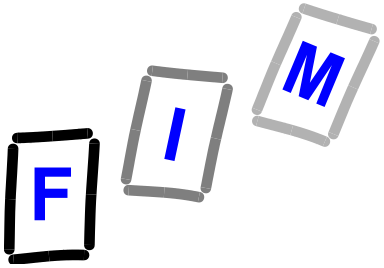
  » **Mehrbenutzerbetrieb (multi user mode)**

  » **Server:**

  File-, Mail-, Drucker-, …-Server: Dienste für mehrere Benutzer

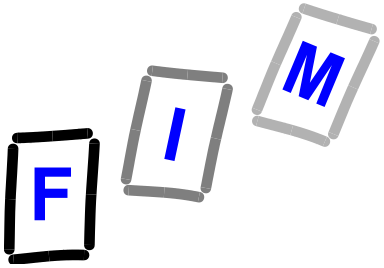
  Remote Desktop Services-Server: Direkt mehrere Benutzer

  .....



Betriebsarten

- ***Betriebsart:***
  die Art und Weise, wie ein Betriebssystem Aufgaben übernehmen und durchführen kann



Sequentielle u. parallele Prozesse

- **Das System kann ausgelegt sein, um**
  - **➔ mehrere als Prozesse ablaufende Aufgaben (Tasks) abzuarbeiten:**
    - **» einfach der Reihe nach (FIFO)**
    - **» oder nach Prioritäten, der Reihe nach**
    - **» oder parallel bzw. ineinander verzahnt**

      Parallelität wird ein gesondertes Thema sein

Betriebsarten

- **Stapelverarbeitung**

  (Batchprocessing): Heute geringe Bedeutung

  ➔ **Als Konzept weiterhin wichtig und praktisch eingesetzt!**

- **Mehrprogrammbetrieb**

  (Multitasking, Multiprogramming): Standard für „normale“ Betriebssysteme

  ➔ **Ein Benutzer, mehrere Programme**

- **Teilnehmerbetrieb**

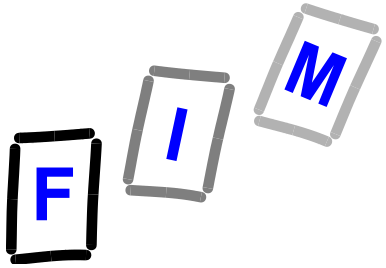
  (Timesharing): Remote Desktop Services etc.

  ➔ **Mehrere Benutzer, jeder davon ein/mehrere Programme**

- **Echtzeitverarbeitung**

  (Real Time): Standard für Spezialaufgaben, zB Steuerungsrechner

- **Cloud-Computing?**



Betriebsarten

- **In der Realität beherrschen moderne Betriebssysteme meist mehrere Betriebsarten gleichzeitig, sind also in diesem Sinne *hybrid*.**

Stapelverarbeitung
(Batchprocessing)

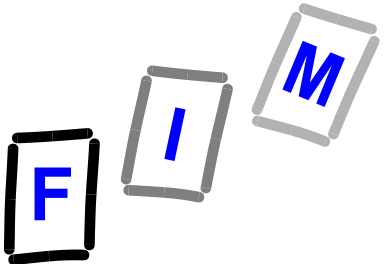
- ➔ **JCL: Job Control Language**
  - » **Zunächst auf Lochkarten,**
  - » **dann auf Magnetbänder oder Disks**
- ➔ **Batch-Files (*.bat) für Kommandos**
  - » **z.B. DOS Betriebssystem (DOS-Box)**
    - – Kommandos interpretiert durch command.com
  - » **Shell scripts unter Linux (div. Varianten: bash, ksh, …)**
- ➔ **Konzept: Abarbeitung von großen Aufgaben in Teilschritten oder vielen gleichartigen Aufgaben**
  - » **Beispiele: Überweisungen in Banken, Parallelisierung durch Aufteilung in Kleinaufgaben**

Lochkarte
Hermann Hollerith

- **Lockarten wurden gestapelt**
  - **➔für JCL**
  - **➔für die Daten**
    - **» Quelltext (FORTRAN, COBOL, Assembler)**
    - **» „Eigentliche“ Input-Daten**

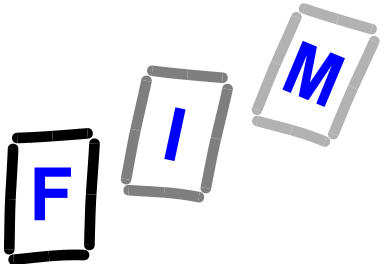
Hermann Hollerith

1860-1929, USA



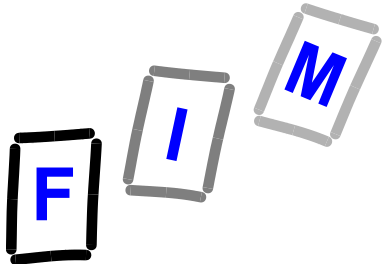
Batchprocessing
JCL Beispiel (Schablone)

- **//COMPILE JOB job card information //***
  //* EXAMPLE JCL FOR COMPILATION. //* REPLACE GENERIC NAMES AS APPROPRIATE. //*----------
- **//* SYNTAX ANALYSIS PHASE**
  //STEP1 EXEC PGM=LC1370,REGION=1024K,
  // PARM='-R P: -M !X:'
  //STEPLIB DD DISP=SHR,DSN=*compiler.library*
  // DD DISP=SHR,DSN=*runtime.library*
  //SYSLIB DD DISP=SHR,DSN=*standard.macro.library*
  //H DD DISP=SHR,DSN=*your.macro.library*
  //SYSPRINT DD SYSOUT=*class*
  //SYSTERM DD SYSOUT=*class*
  //SYSUT1 DD UNIT=SYSDA,SPACE=(CYL,(1,1)),
  // DISP=(NEW,PASS),
  // DSN=&&QUADS
  //SYSIN DD DISP=SHR,DSN=*your.source.library(member)*
  //*
- **//* CODE GENERATION PHASE //STEP2 EXEC PGM=LC2370,REGION=1024K,COND=(4,LT)**
  //STEPLIB DD DISP=SHR,DSN=*compiler.library*
  // DD DISP=SHR,DSN=*runtime.library*
  //SYSPRINT DD SYSOUT=*class*
  //SYSTERM DD SYSOUT=*class*
  //SYSUT1 DD DISP=(OLD,PASS),DSN=&&QUADS
  //SYSLIN DD DISP=OLD,DSN=*your.object.library(member)*
  //



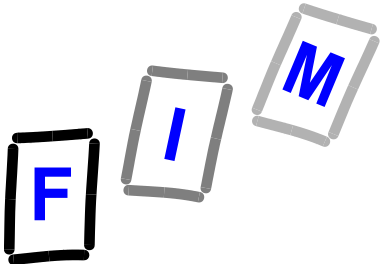
Batchprocessing
JCL Beispiel (Schablone)

- **//COMPILE JOB job card information //***
  //* EXAMPLE JCL FOR COMPILATION. //* REPLACE GENERIC NAMES AS APPROPRIATE. //*----------
- **//* SYNTAX ANALYSIS PHASE**
  //STEP1 EXEC PGM=LC1370,REGION=1024K,
  - ➔ Ausführen von Programm „LC1370“
- **// PARM='-R P: -M !X:'**
  - ➔ Parameter für das Programm, kann von diesem ausgelesen werden
- **//STEPLIB DD DISP=SHR,DSN=*compiler.library***
  - ➔ Bibliothek, wo das Programm liegt (Dateiname: „compiler.library“)
- **// DD DISP=SHR,DSN=*runtime.library***
  - ➔ Zweite Bibliothek
- **//SYSLIB DD DISP=SHR,DSN=*standard.macro.library***
  - ➔ Weitere Bibliothek
- **//H DD DISP=SHR,DSN=*your.macro.library***
  - ➔ Programm liest „your.macro.library“ ein unter Namen „H“
- **//SYSPRINT DD SYSOUT=*class*** **// Standardausgabe → Konsole**
  //SYSTERM DD SYSOUT=*class* **// Standardfehlerausgabe → Konsole**
- **//SYSUT1 DD UNIT=SYSDA,SPACE=(CYL,(1,1)),**
  // DISP=(NEW,PASS),
  // DSN=&&QUADS
  - ➔ Ausgabedatei auf virt. Speicherplatz „SYSDA“ mit Größe 1 Zylinder wird neu angelegt; Dateiname aus Variable „QUADS“
- **//SYSIN DD DISP=SHR,DSN=*your.source.library(member)***
  - ➔ Quellcode des Programms wird angegeben



Batchprocessing
JCL Beispiel (Schablone)

- **//STEP2 EXEC PGM=LC2370,REGION=1024K,COND=(4,LT)**
  - ➔ **Ausführen von Programm „LC2370“**
- **//STEPLIB DD DISP=SHR,DSN=compiler.library**
  // DD DISP=SHR,DSN=runtime.library
  - ➔ **Bibliotheken**
- **//SYSPRINT DD SYSOUT=class**
  //SYSTERM DD SYSOUT=class
  - ➔ **Standard(Fehler)ausgabe**
- **//SYSUT1 DD DISP=(OLD,PASS),DSN=&&QUADS**
  //SYSLIN DD DISP=OLD,DSN=your.object.library(member)
  - ➔ **Fertig compilierte Datei wird auf die Objekt-Bibliothek „member“ ausgegeben, wobei nur dieser Job die Datei verändern darf (andere Jobs haben keinen Zugriff)**

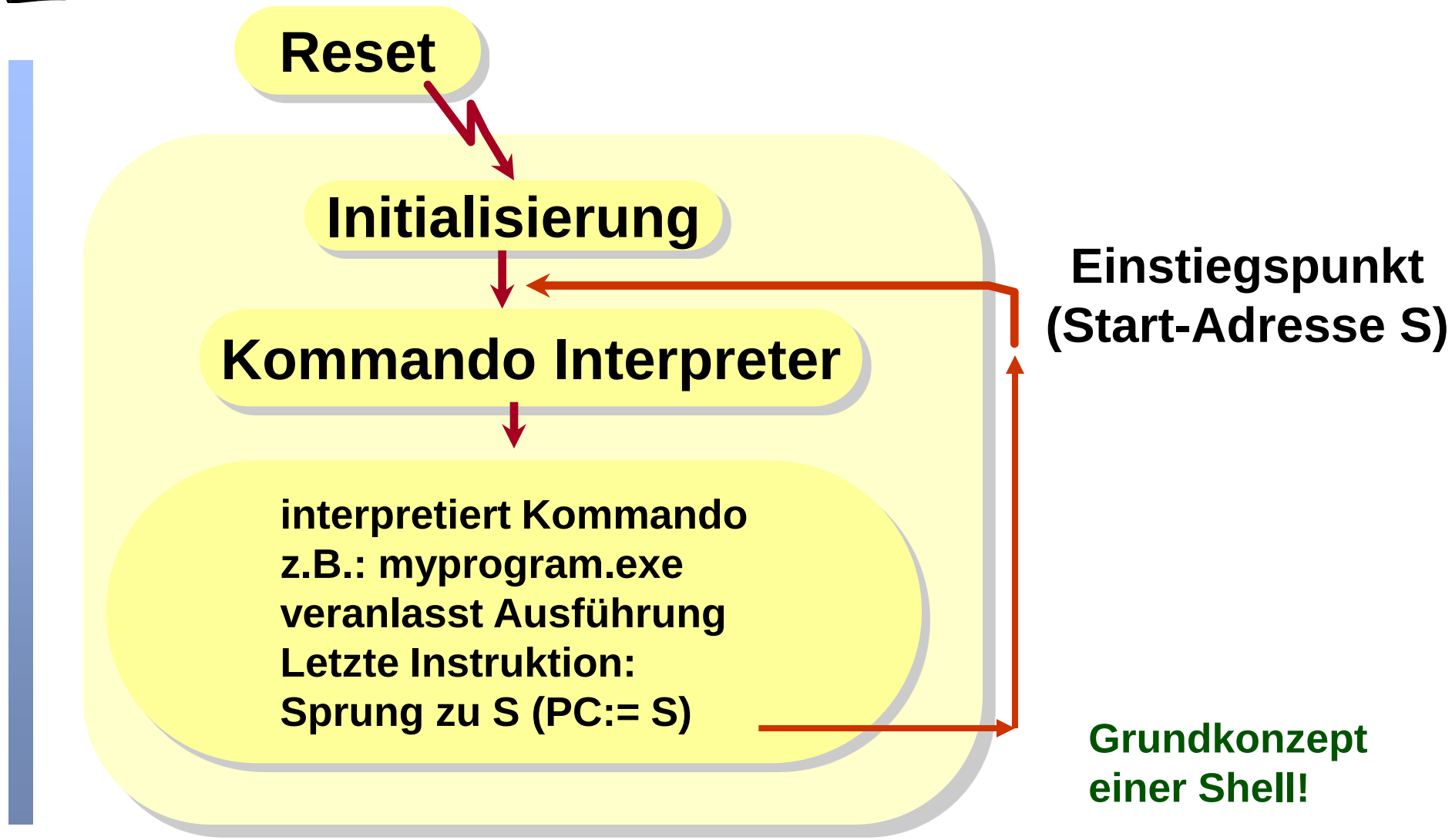


Batchprocessing
batch-file in DOS

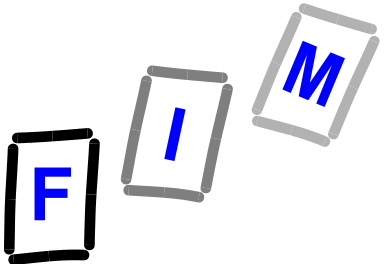
```
@ECHO OFF

:START
COPY file.txt file2.txt

IF errorlevel 1 GOTO MKFILE
GOTO :END

:MKFILE
ECHO file text > file.txt
GOTO START

:END
ECHO beendet

        @echo on

PAUSE
```

Groß-/Kleinschreibung, Einrückung → Egal!

„Monitor“
Ablauf bei einfachem BS



Grundkonzept einer Shell!



Spooling Konzept

➔ **Überlappende I/O Operationen (*SPOOL*)**

Simultaneous **P**eripheral **O**peration **O**n**L**ine

» **„Spooler“ ist ein Betriebssystemprozess zusammen mit ein oder mehreren Puffern**

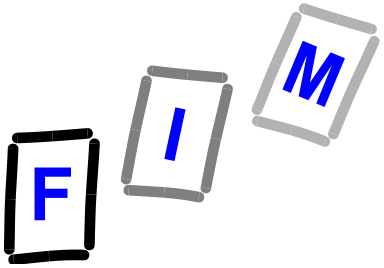
» **Verwaltet I/O für ein oder mehrere Geräte**

» **Kommuniziert mit der CPU über Interrupts**

➔ **Spooling-Konzept:**
Erstmals 1961 auf der in Manchester (UK) entwickelten „Atlas-Maschine“ eingesetzt.

- **Praktisch: Drucker-Spooler**

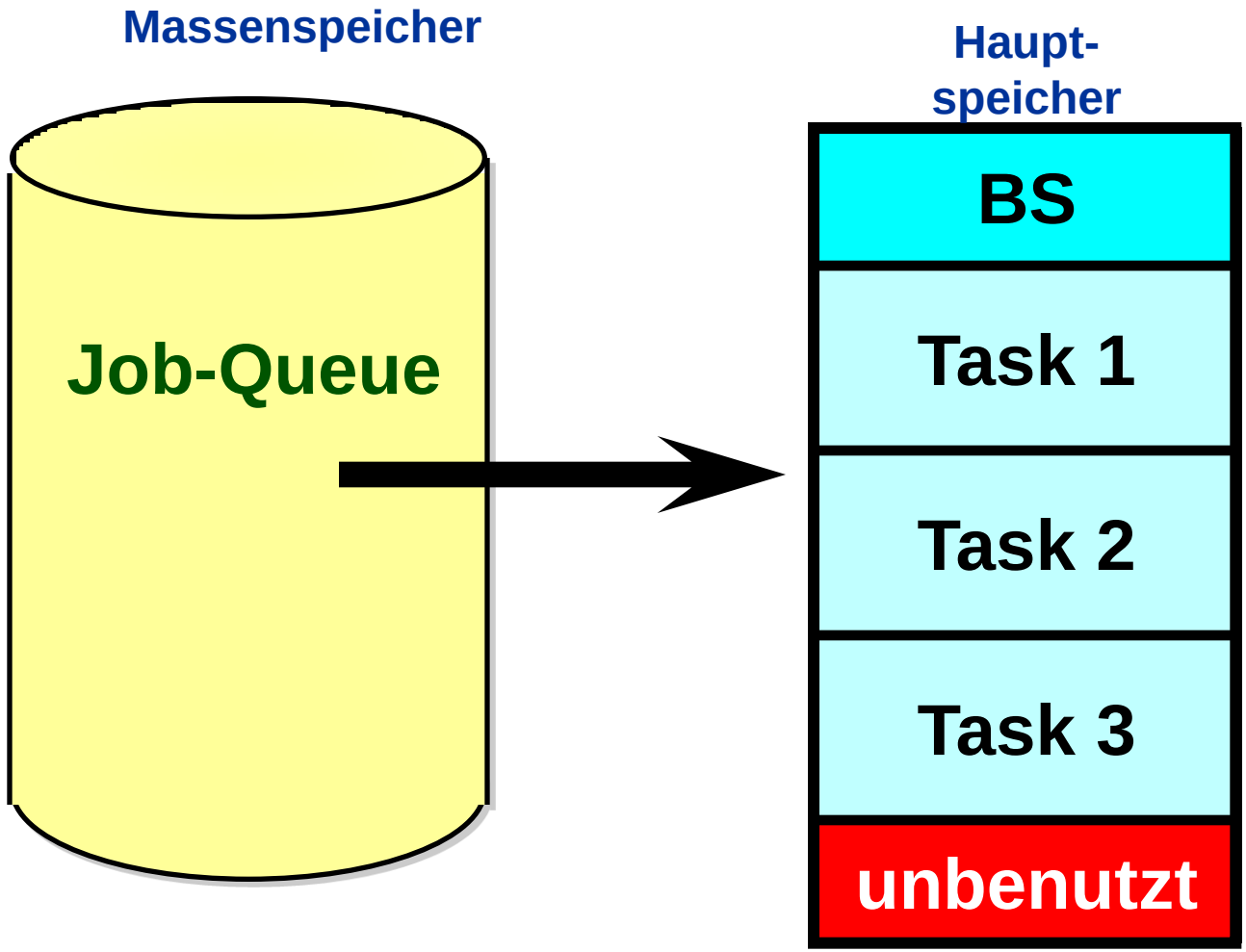
➔ **Ein Drucker, gleichzeitiges Drucken möglich, Zwischenspeicherung bis zur tatsächlichen Verarbeitung, Organisation der Reihenfolge etc.**

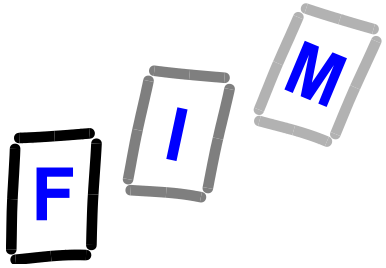


Mehrprogrammbetrieb

- **Multitasking, Multiprogramming**
  - ➔ **Wenn ein Prozess aus irgendeinem Grund warten muss, schaltet das BS zu einem anderen Prozess (task switch)**
    - » **D.h.: die CPU wird einem anderen Prozess zugeteilt**
  - ➔ **Insbesondere beim Warten auf I/O Prozesse**
  - ➔ **Multiprogramming-Grad**
- **Ursprünglich:**
  Prozesse werden nach Prioritäten von der CPU bedient
- **Jetzt: Weitere und feinere Methoden**
  - ➔ **Generalthema dazu: CPU- Scheduling**

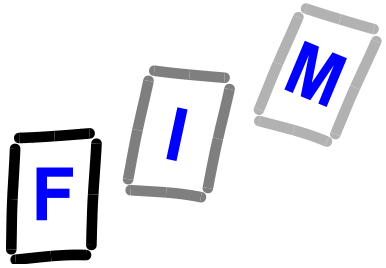
Multitasking





Teilnehmerbetrieb Timesharing

- I/O kann interaktiv sein
  - ➔ Z.B.: Ist langsam, läuft nur mit Geschwindigkeit entsprechend der Eingabe durch den Anwender
  - ➔ Daher sollte die CPU zwischenzeitlich zu einem anderen Benutzer-Prozess wechseln
- TS erlaubt Benutzern, einen Computer gemeinsam zu benutzen (share)
  - ➔ Timesharing bei Terminalservern



Teilnehmerbetrieb Timesharing

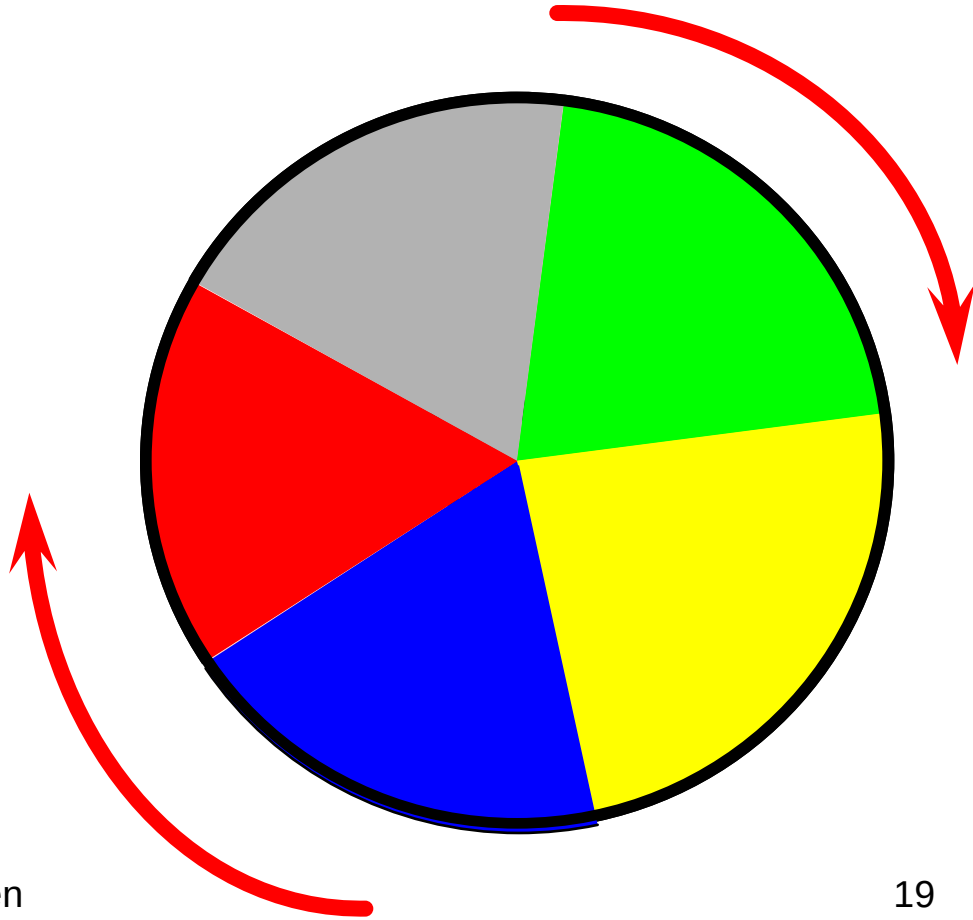
- ***Jedem anstehenden Prozess wird (der Reihe nach zyklisch) eine gewisse Zeit die CPU zugeteilt.***
- **Nach Ablauf dieser Zeit**

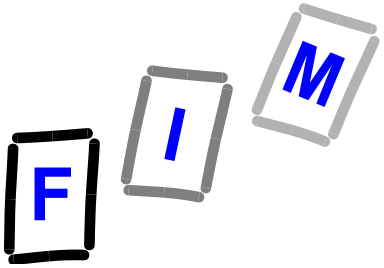
  ➔Rechenzeitperiode (time slice, time-quantum)

  gibt der Prozess die CPU wieder ab, der nächste Prozess kommt zum Zug.
- **Dies wird zyklisch wiederholt.**

Timesharing

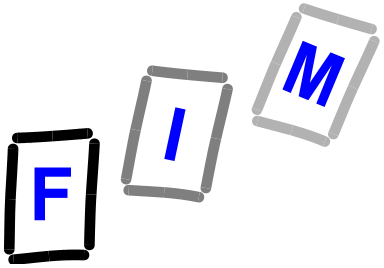
Beachte: Ein „Job“ löst im System einen Prozess aus





Timesharing <> Round Robin

- **Realisierung?**
  - » **Spezielles CPU Scheduling**
  - » **Im Beispiel:** ***Round Robin***
  - » **Wird später noch besprochen**



Probleme bei Timesharing/ Multiprogramming

- ***Speicherverwaltung*:**

  Wenn Speicher für die Aufnahme aller n > 1 „gleichzeitig“ laufenden Prozesse zu klein ist ?

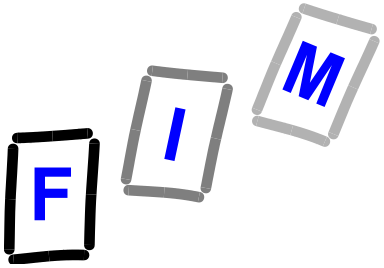
- ***Speicherschutz***

  Prozess x darf nicht (ungewollt) in den/vom Bereich von Prozess y schreiben / lesen

- **Ressourcen-Zuweisung:**

  Welcher Prozess erhält wann welches Gerät (allgemein: Ressource) ?

  - **Ressourcensteuerung**
  - **CPU-Scheduling**

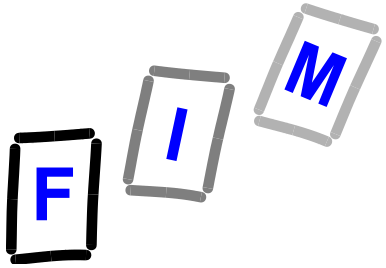


Echtzeitsysteme

- ***Benutzerauftrag muss innerhalb einer vordefinierten Zeitspanne ausgeführt sein, sofern dies möglich ist***

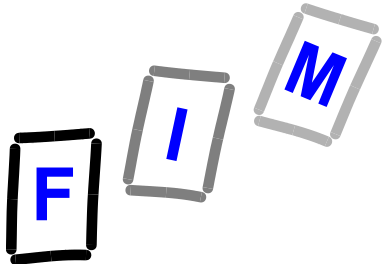
bzw.

- ***System muss innerhalb einer vordefinierten Zeitspanne reagieren***



“Harte” Echtzeitsysteme Hard Real-Time Systems

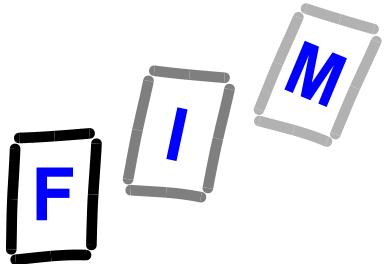
- Garantiert, dass kritische Aufgaben rechtzeitig abgeschlossen werden
- Massive Nutzung von ROM als nicht-flüchtiger Speicher anstatt Massenspeicher (Festplatten,..)
- Konflikt mit den Zielen eines Timesharing- Systems
- Im Allgemeinen anders konzipiert als „normale Betriebssysteme“
- Oft enorm überdimensioniert oder sehr spezielle Hardware



“Hard” Real-Time Systems

Typische Anwendungen

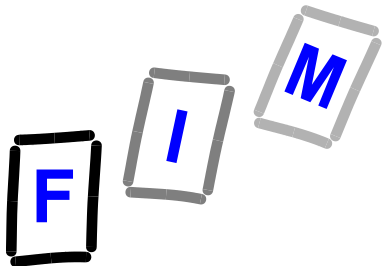
- **Intensivstation (Medizin)**
- **Prozesskontrolle**
  - ➔ **Flugzeuge**
  - ➔ **Roboter**
  - ➔ **Jegliche industrielle Mess- und Regelungsaufgaben**
  - ➔ **…**



“Sanfte” Echtzeitsysteme
Soft Real-Time Systems

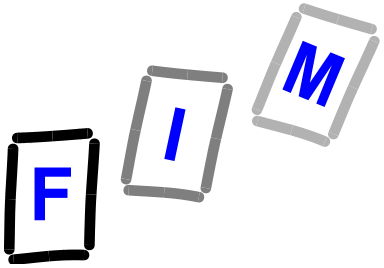
- Ein kritischer Prozess erhält Priorität vor anderen Prozessen
- Hält diese Priorität, bis er fertig ist

- Konzept in den meisten "neuen" Betriebssystemen enthalten
  - ➔UNIX
  - ➔Linux
  - ➔ab Windows NT 3.5



Hard-versus Soft-Realtime Systems

- **Unterschied in „Echtzeitanforderungen“:**
  - ➔ **Bei Hard Real Time Systemen ist das Überschreiten einer Zeitschranke (*deadline*) ein schwerwiegender Systemfehler**
    - » **Antwortzeit wird garantiert**
  - ➔ **Bei Soft Real Time Systemen ist der Mittelwert der Antwortzeiten ausschlaggebend.**
    - » **Wird ausnahmsweise die Reaktionszeit überschritten, so kann dies – abhängig von der konkreten Anwendung**
      ➔ hingenommen werden
      oder
      ➔ das Ergebnis wird schlicht und einfach verworfen.
    - » **Antwortzeit wird meist/durchschnittlich eingehalten**

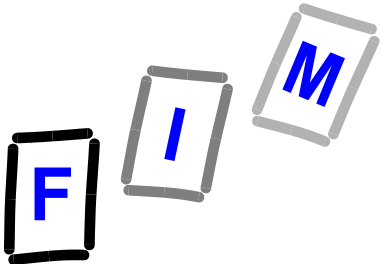


Betriebsarten

- **Die bislang erwähnten Betriebsarten dienen zur Charakterisierung und Klassifizierung früherer Betriebssysteme**
- **Moderne Systeme vereinigen diese Betriebsarten und verwenden sie miteinander kombiniert**

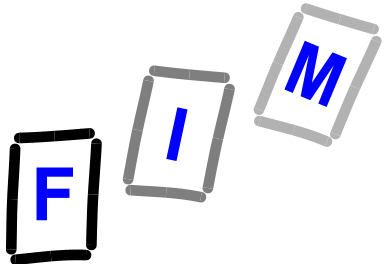
  ➔ **Ein Server für ein LAN könnte …**

  - » **vorbereitete Aufgaben im Batch-Mode verarbeiten,**
  - » **mit dem Administrator in einer Time-Sharing ähnlichen Art und Weise interagieren, oder**
  - » **auf Notfälle reagieren wie ein Real Time System**



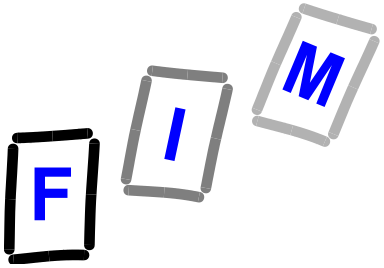
Grid-Computing

- **Ausgangssituation: Rechnerverbund**
  - ➔ **„Computer Cluster“: Menge von Computern, die bzw. deren Ressourcen über ein Computernetzwerk miteinander verbunden sind**
  - ➔ **Besondere Ziele:**
    - » **Durch Zusammenschalten einen besonders leistungsfähigen Computer zu erhalten**
      - – Details: Nächste Folie



Grid-Computing Ziele

- **Betriebsmittel, die nicht lokal verfügbar sind**
- **Erhöhung der Computerleistung**
  - ➔ **Spezialrechner**
  - ➔ **Parallele Verarbeitung von Teilaufgaben**
- **Failover-Funktion**
  - ➔ **(Erhöhung der Verfügbarkeit)**
- **Lastverteilung (Load Balancing)**
  - ➔ **Wenn mehrere „Knoten“ im Verbund für dieselben/ähnliche Aufgaben konzipiert sind**



Cloud-Betriebssystem

- Ähnlich zu Grid-Computing
- Minimales Betriebssystem lokal
  - ➔ Die meisten Ressourcen sind jedoch verteilt und transparent für den Benutzer
- Trivialform: Rechner startet zu Remote-Desktop/Webbrowser und Anwendung liegt irgendwo im Internet
  - ➔ Irgendwo: Fixe „Adresse“ die Loadbalancing betreibt und zu passender physischer Ressource weiterleitet
- Oft aber nur: Rechnernetz/Verteiltes System
  - ➔ Beliebig (Kosten!) viele virt. Server können gemietet werden
    → Ganz normales Betriebssystem + spezielle Anwendung

Middleware Schicht