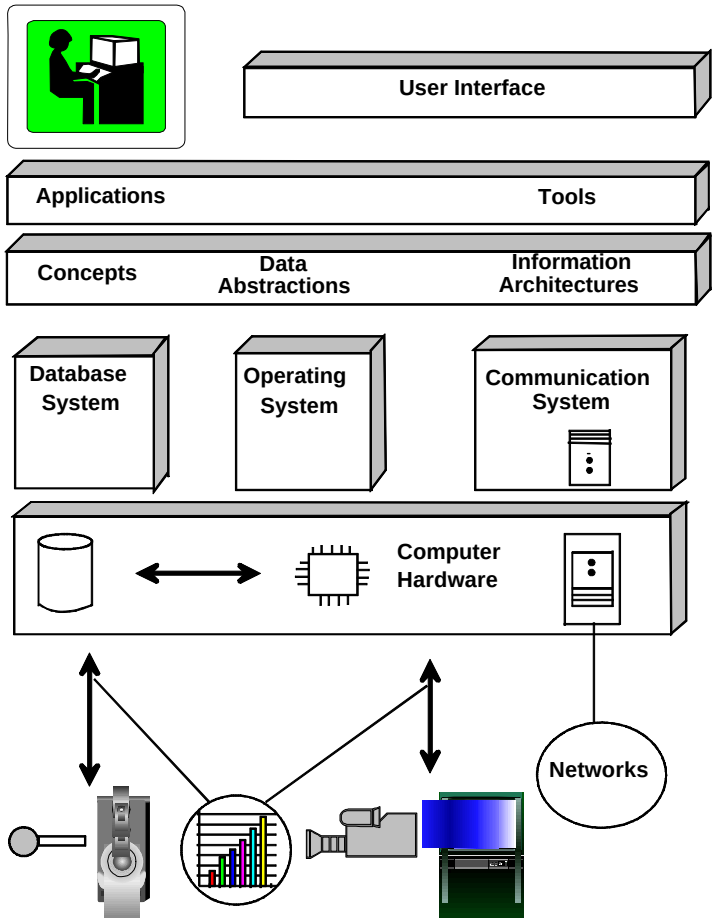
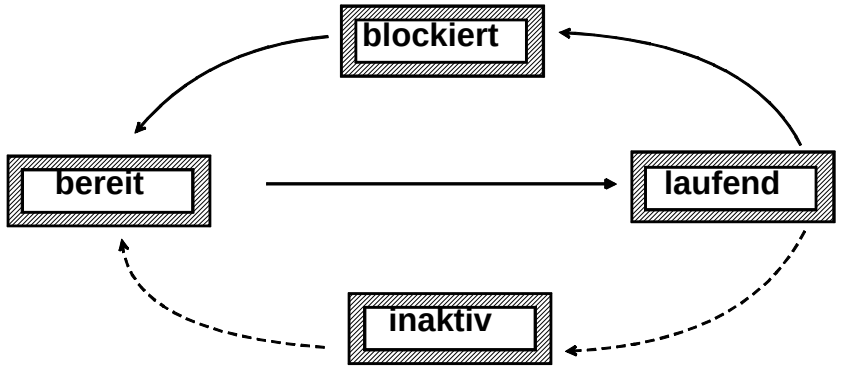
5 Betriebssystemunterstützung für Multimedia



5.1 Echtzeitfähigkeit

Prozess-Verwaltung: Scheduler und Dispatcher

- Logische Ressourcen (Prozesse) werden auf physikalische Ressourcen (z.B. CPU(s)) abgebildet.
- Ein Prozess kann verschiedene Zustände einnehmen:
  - laufend: Ein realer Prozessor ist ihm zugeordnet.
  - bereit: Er besitzt alle Betriebsmittel bis auf den Pro-zessor.
  - blockiert: Er wartet auf das Eintreten eines Ereignisses (z.B. das Ende einer Plattenausgabe).
  - inaktiv: Dem Prozess ist kein Programm zur Ausführung zugeordnet.



Scheduler und Dispatcher

- Der Scheduler entscheidet, welche der wartenden Prozesse vom Zustand inaktiv in den Zustand bereit übernommen werden.
- Der Dispatcher verwaltet den Übergang von bereit nach laufend. Er arbeitet meist mit Prioritäten.

Scheduler und Dispatcher in herkömmlichen Betriebssystemen sind in der Regel **nicht echtzeitfähig**.

Anforderungen für Multimedia

Verarbeitung von kontinuierlichen Datenströmen

- Die zu verarbeitenden Daten treten in **periodischen Zeitabständen** auf.
- Die Operationen auf diesen Daten **wiederholen sich** immer wieder:
  - Erfassung/Ausgabe von Audio- und Videopaketen
  - Weiterleitung von Audio- und Videopaketen
  - Kompression/Dekompression von Audio- und Videopaketen
- **Echtzeit-Anforderung**
  - Die Verarbeitung muss **bis zu einem gewissen Zeitpunkt** abgeschlossen sein (Deadline).
  - Die Verarbeitung benötigt pro Periode in etwa dieselben Ressourcen.

Verfahren zur Reservierung von Betriebsmitteln

- **Pessimistisch/deterministisch**
  - berücksichtigt den ungünstigsten Fall
  - führt in der Regel zu einer Überreservierung
  - schlechte Ausnutzung der Ressourcen

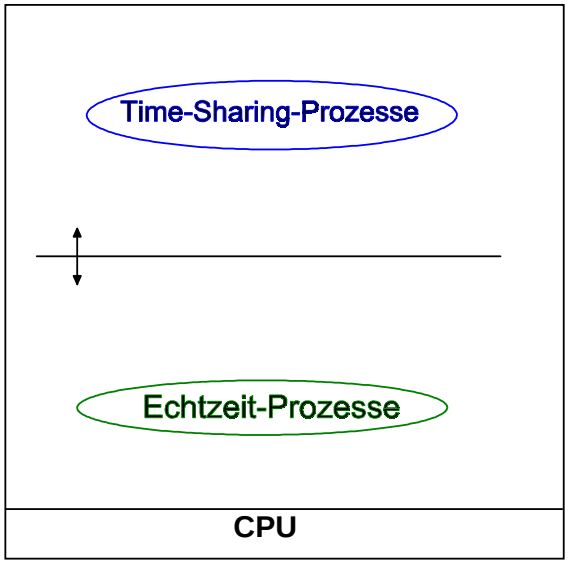
- **Optimistisch/stochastisch**
  - Belegung richtet sich nach einem Erwartungswert/ Mittelwert
  - gute Ausnutzung der Ressourcen
  - kann zu Überlastfällen führen
  - Monitor: bemerkt diese Überlast und veranlasst entsprechende Aktionen: Entzug der Ressource oder Benachrichtigung des Prozesses. Der Prozess muss dann eine entsprechende Ausnahmebehandlung ausführen.

Echtzeit-Prozessverwaltung für Multimedia

Bisher gibt es in der Regel in Betriebssystemen

- entweder Scheduling für Time-Sharing
- oder Scheduling für Echtzeitanwendungen

Erforderlich für Multimedia ist ein Scheduler für beides



- **Echtzeit heißt**
  - Daten sind nur brauchbar, wenn sie vor der Deadline bereitstehen bzw. verarbeitet sind

Unterschiede zu traditionellen Echtzeitsystemen

Geringere Zuverlässigkeitsanforderungen

- Daten traditioneller Echtzeitsysteme steuern/regeln Prozesse
- Multimedia-Daten dienen in der Regel der Präsentation für den Menschen =>
  - Tolerierbare Verletzung der Zeitschranken: Fehler im Video (z.B. Pixel noch aus dem alten Block) kann evtl. toleriert werden
  - Auswirkungen abhängig von der Kodierungstechnik

Daten treten meist periodisch auf

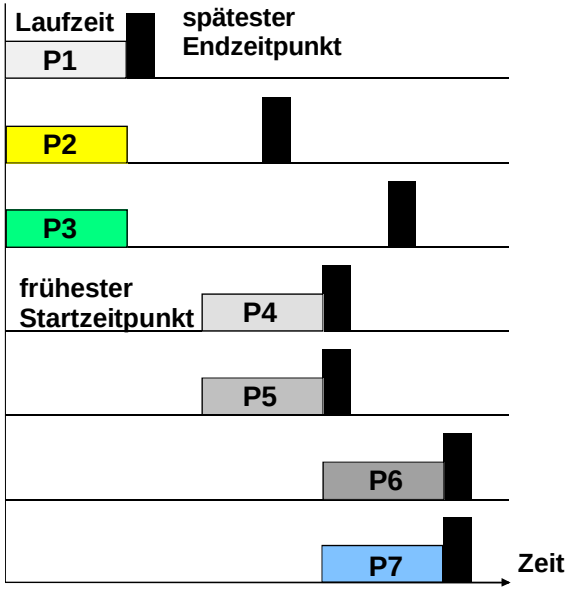
- wegen periodischer Abtastung
- **periodische Prozesse sind einfacher zu verwalten als allgemeine Realzeit-Prozesse**

Ziele der Prozessverwaltung

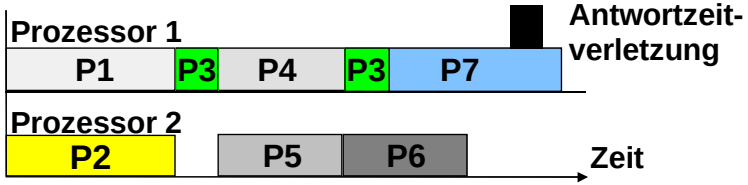
- **Verwaltung** der Ressource (z.B. CPU) so, dass alle zeitlichen Randbedingungen (Deadlines) **aller Prozesse** befriedigt werden
  - bei hoher Auslastung der Betriebsmittel
  - schneller Berechnung der Zuteilung.

Die Zuteilung muss nicht optimal sein. Wie man zeigen kann, wäre das Problem sonst NP-vollständig.

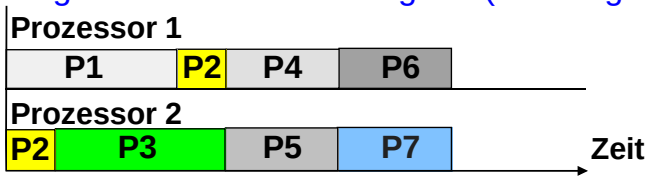
Scheduling-Problem: Ein Beispiel



Prozessorvergabe nach Spielraum



Zeitgerechte Prozessorvergabe (nicht algorithmisch)



5.2 Scheduling-Verfahren

Anforderungen an den Scheduler

Muss Dienstgütegarantien unterstützen!

- Die Berechnung von Scheduling-Entscheidungen muss Dienstgüteparameter einbeziehen
- Alle gegebenen Zusagen müssen zur Laufzeit eingehalten werden

Der Scheduler sollte die Eigenschaften von kontinuierlichen Strömen einbeziehen:

- Periodische Anforderungen sollten gut unterstützt werden
- Aperiodische Anforderungen (z.B. durch Kontrollprozesse) sollten nicht "verhungern".

Entzug der Ressource

Preemptive scheduling

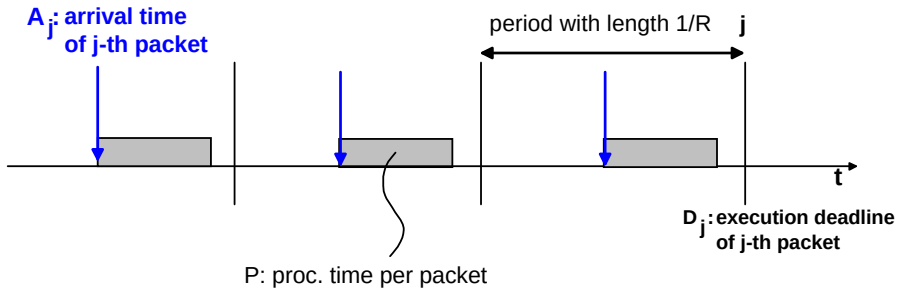
- Wenn ein Prozess mit höherer Priorität eine Anforderung stellt, wird dem gerade aktiven Prozess die Ressource entzogen
- Es entsteht ein erhöhter Aufwand für die Prozessverwaltung, und es gibt mehr Prozesswechsel

Non-preemptive scheduling

- Aktive Prozesse werden nicht unterbrochen
- In manchen Fällen durch äußere Zwänge vorgegeben, z.B. für einen Prozess, der vom Netz liest
- Weniger Aufwand für Prozesswechsel

Non-preemptive scheduling ist in der Regel vorzuziehen, wenn die Ausführungszeiten kurz sind.

Modell des periodischen Datenstroms

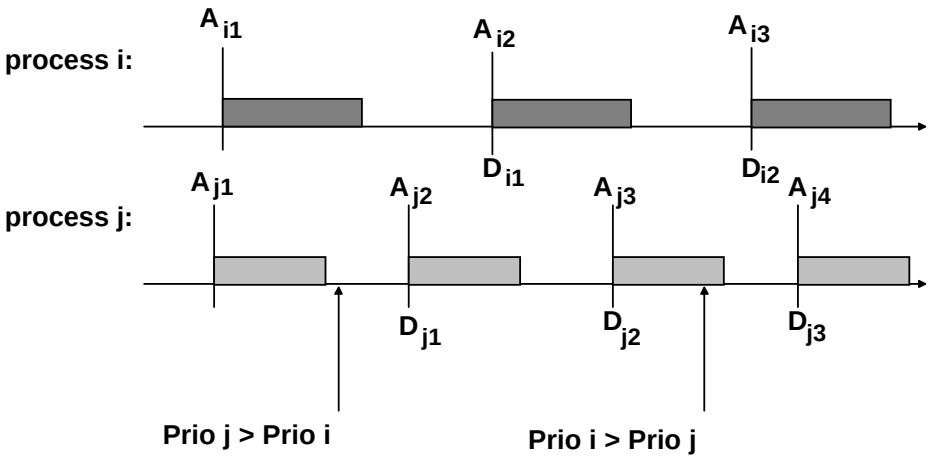


- R_j: Ankunftsrate
- P_j: Verarbeitungszeit (processing time)
- D_j: Deadline (Verarbeitung muss beendet sein)

=> Diese Parameter steuern den Scheduling-Algorithmus

Algorithmus 1: Earliest Deadline First (EDF)

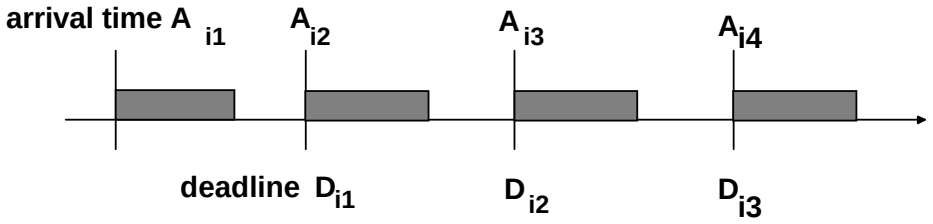
Der Prozess mit der frühesten Deadline erhält jeweils die höchste Priorität.



Die Prioritäten ändern sich über der Zeit.
Man kann zeigen, dass EDF stets einen möglichen Ablaufplan (eine "Schedule") erzeugt, wenn es überhaupt einen gibt. Die Auslastung der Ressource kann bis zu 100% betragen.

EDF Scheduling

In den meisten Fällen des periodischen Schedulings ist Deadline = Periodenende, weil beispielsweise der Datenpuffer wieder gebraucht wird.



QoS-Berechnung

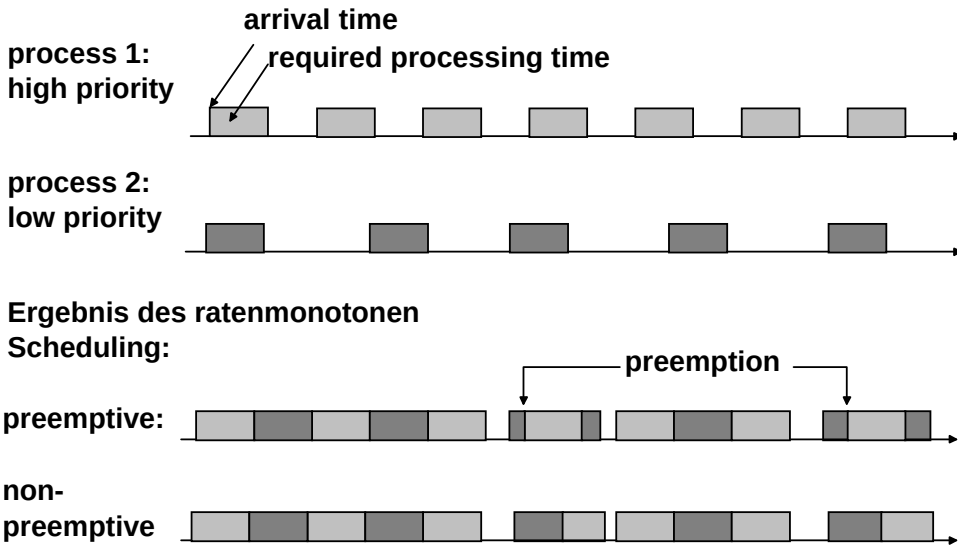
- Preemptive scheduling (Liu /Layland, 1973):
  - maximal möglicher Durchsatz:

$$\sum_{alle\,Ströme\ i} R_i P_i \le 1$$

  - Paketverzögerung ≤ 1/ R_i
- Non-preemptive scheduling (Nagarajan/Vogt, 1992)
  - gleicher Durchsatz wie oben
  - Paketverzögerung ≤ 1/ R_i + P, P = Verarbeitungszeit

Algorithmus 2: Ratenmonotones Scheduling

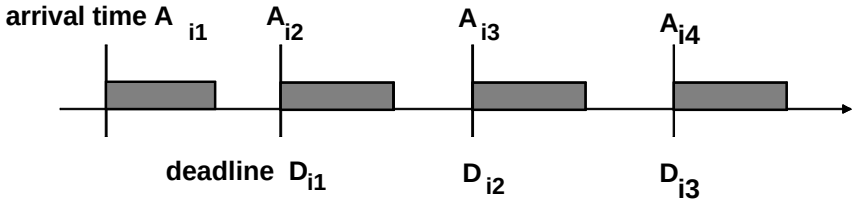
Der Prozess mit der höchsten Paketrate erhält die höchste Priorität.



Die Priorität ist konstant; nur zu Beginn und Ende eines Stromes müssen Prioritäten neu berechnet werden.

Ratenmonotones Scheduling (RM)

Wir setzen wieder Deadline = Periodenende:



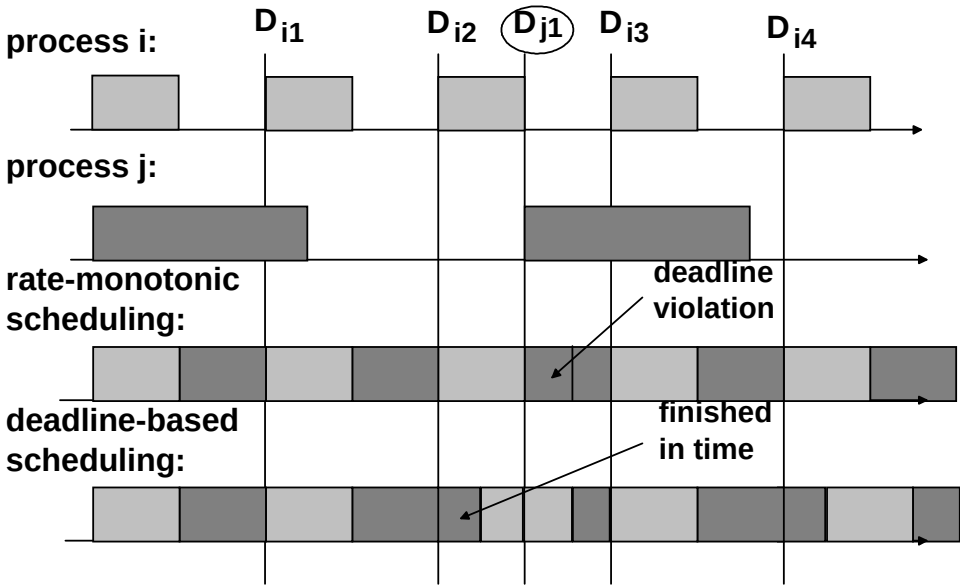
QoS-Berechnung:

- Preemptive scheduling (Liu/ Layland, 1973):
  - Maximal möglicher Durchsatz:

$$\sum_{alle\,Ströme\ i} R_i P_i \leq \ln 2$$

  - Paketverzögerung ≤ 1/ R_i
- Non-preemptive scheduling (Nagarajan/ Vogt, 1992):
  - sehr komplizierte Berechnung
  - garantierter Durchsatz deutlich niedriger

Beispiel

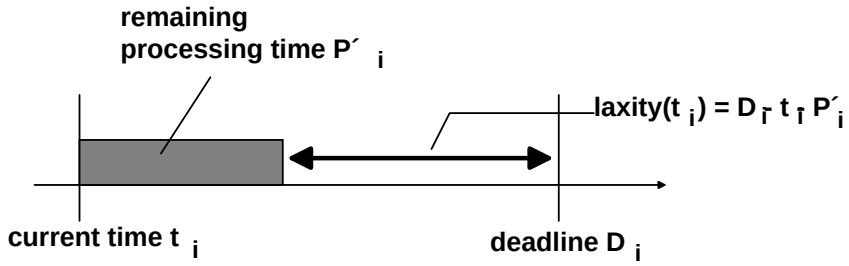


Andere Scheduling-Algorithmen

Scheduling mit festen Prioritäten:

- Jeder Strom erhält eine feste Priorität.
- Ratenmonotones Scheduling ist ein Spezialfall.
- Die Scheduling-Berechnungen erfolgen auf der Basis von "worst-case"-Annahmen für alle Ströme höherer Priorität.

Scheduling nach "laxity":



- "Laxity" = maximale Wartezeit bis zum Beginn der Verarbeitung
- Der Strom mit der kürzesten "Laxity" erhält die höchste Priorität.
- Die Prioritäten müssen ständig neu berechnet werden (hoher Overhead).

Implementierungsprobleme

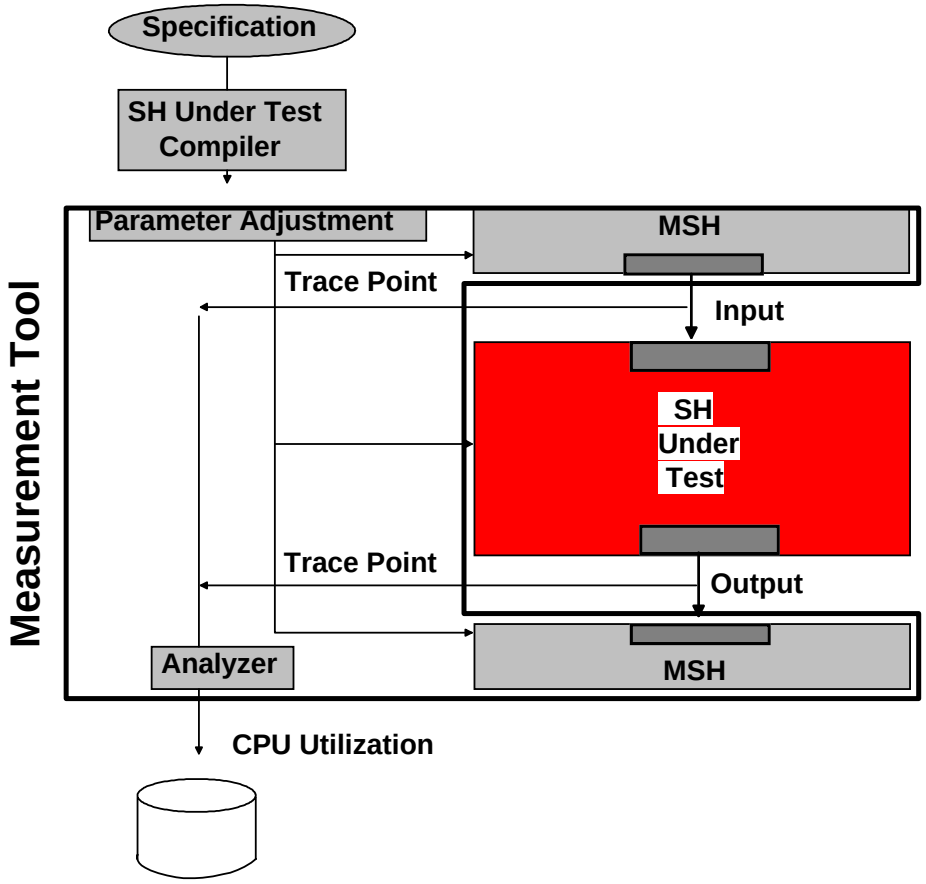
Problem

Die tatsächliche Verarbeitungszeit pro Paket auf der aktuellen CPU muss bekannt sein!

Eine analytische Berechnung ist kaum möglich.

Lösungsansatz: Messung und Normierung

Messung von Bearbeitungszeiten



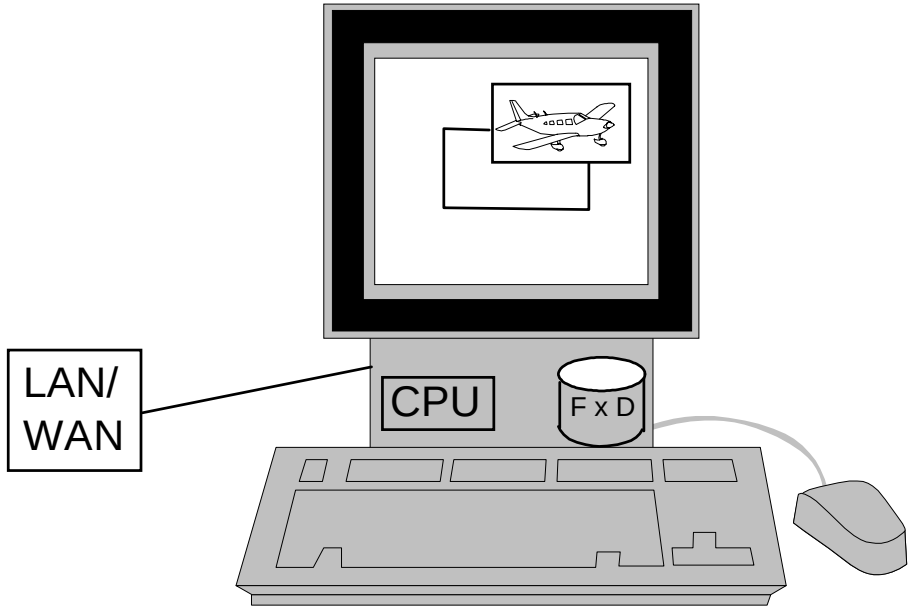
SH = Stream Handler

5.3 Geräteverwaltung

Allgemein

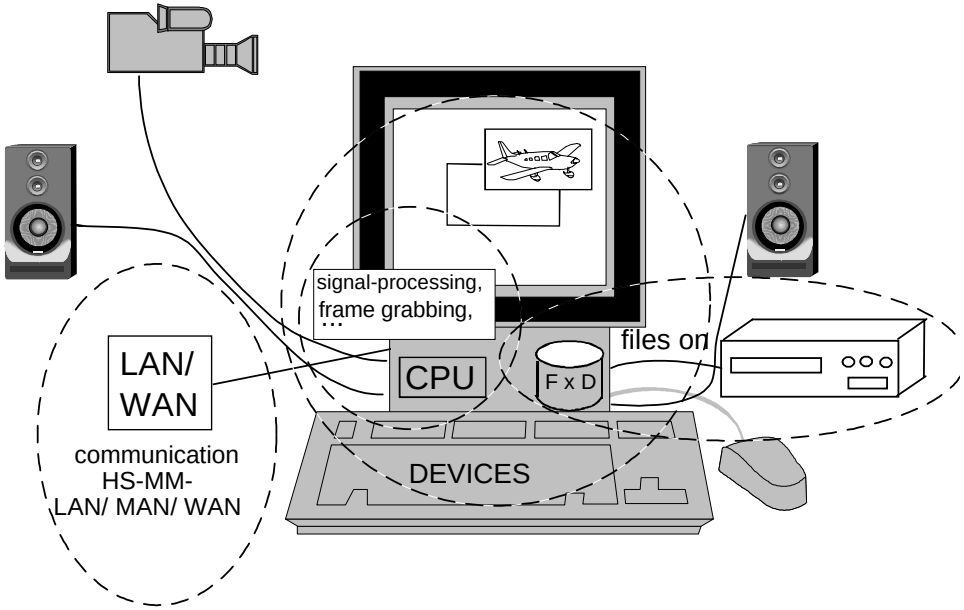
- Gerät = "Device"
- Sekundärspeichercontroller (z.B. Festplatten-Controller) ist auch ein Device
- Es wird eine Abstraktion von realen Geräten vorgenommen. Bestimmte Hardware-Eigenschaften des physikalischen Geräts bleiben verborgen.
- meist durch Device Driver/ Server-Ebene bereitgestellt

Konventionelles System



- Devices
- Monitor, Grafikkarte
- LAN Controller
- Tastatur, Maus
- FxD = Festplatte, Floppy Disk

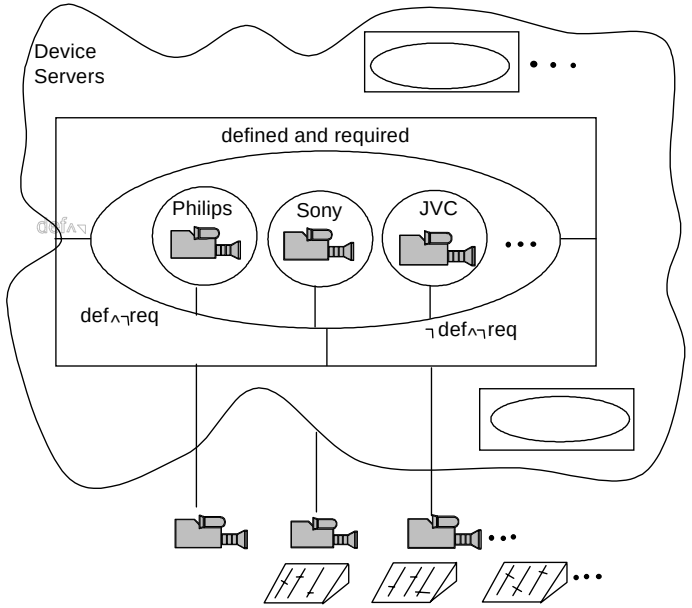
Multimedia-System mit "Devices"



- zusätzliche Devices wie bisherige behandeln
- **"neue" Devices**
  - Kamera, Monitor/ Fenster für Bewegtbild
  - Mikrofon, Lautsprecher, ...
  - Speichermedien für Audio, Video
    - intern: CD-ROM
    - extern: VCR, optische Platten, ...

Lösungsansätze

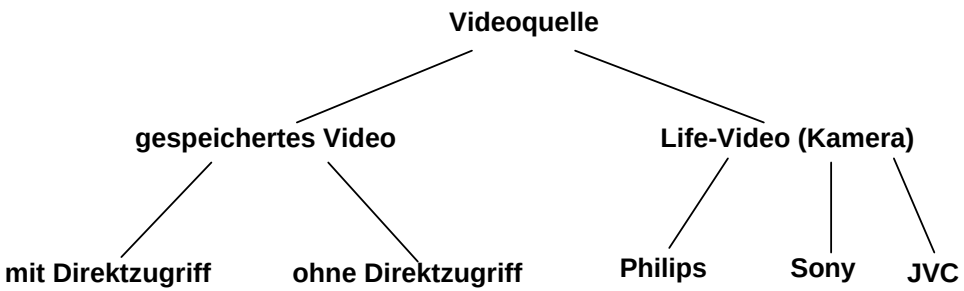
Device Server/ Abstraktion am Beispiel Kamera



Abstraktion von der speziellen Kamera

- für Geräte vom Typ Kamera gibt es
  - set focus
  - zoom (ggf. nur mit Parameter "1")
- Abbildung auf entsprechende Steuerkommandos innerhalb des Treibers

Abstraktionshierarchie (Beispiel)



Eine objektorientierte Implementierung der Gerä-
te"treiber" mit Vererbung bietet sich an!