Ein tragbares Prüfsystem für Beatmungsgeräte: Ein Fall für Embedded Linux

Hans Heinrich Heitmann
Hochschule für Angewandte Wissenschaften Hamburg
Fachbereich Elektrotechnik und Informatik

Übersicht

- Gerätevorstellung
- Softwareentwicklung
  - Entwicklungsumgebung
  - Softwarearchitektur
  - Zeitverhalten
- Fazit

Zweck

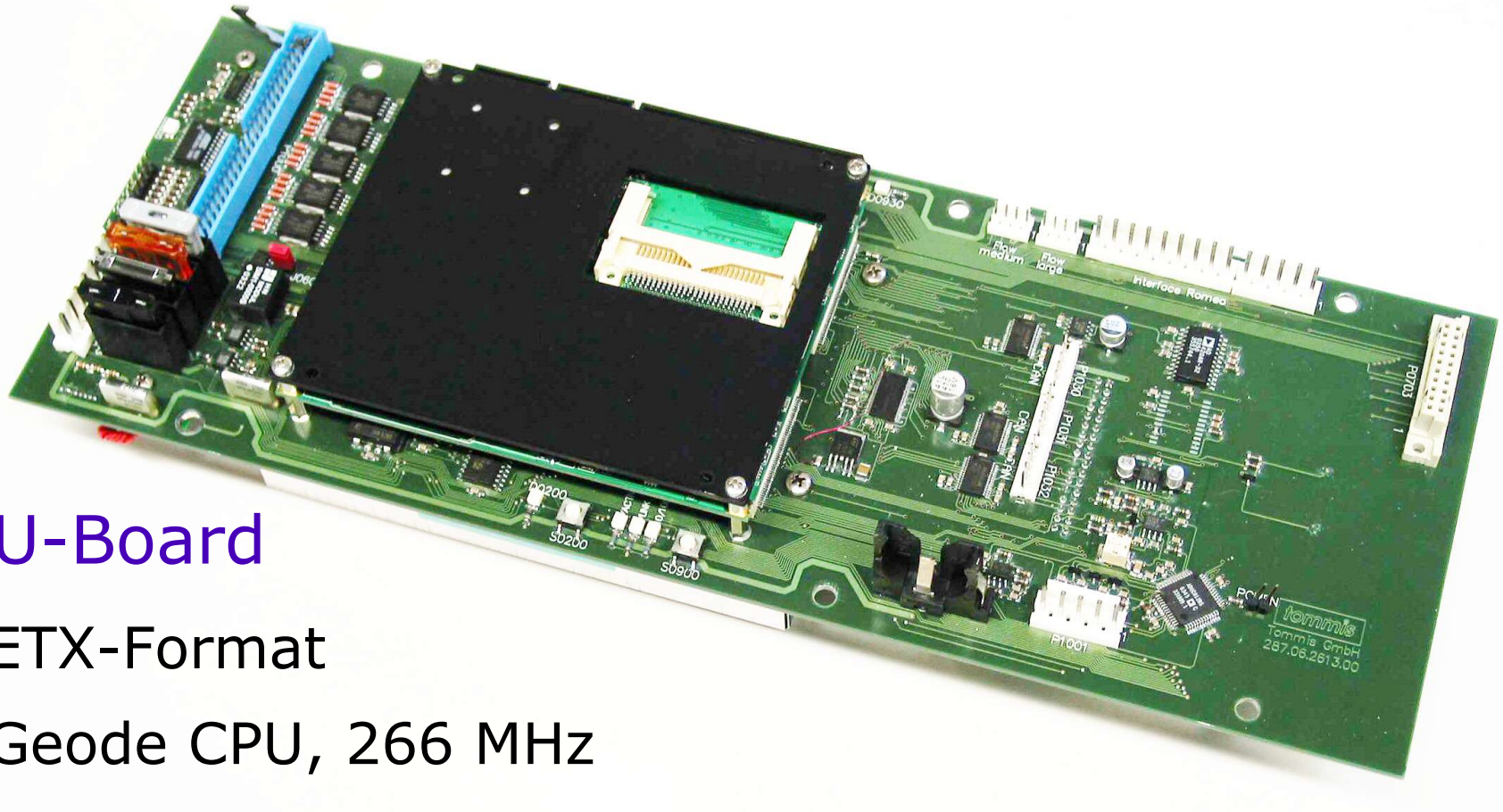
- Durchführung der Wartung lt. Prüfkarte
  - Beatmungsgeräte
  - Narkosesysteme
- Check der Gerätefunktionen
  - Überprüfung der Sensoren
    - Flow, Druck
    - Gasanalyse: O_2, CO_2, Narkosegas
  - Überprüfung der Aktoren
    - Dichtigkeit, Strömungswiderstand
    - Gasmischung
    - Ventilsteuerungen
    - ...

Das Gerät

Gerätebestandteile

- Sensoren
  - 5 Drucksensoren (CanOpen)
  - 5 Flowsensoren (RS232)
  - O_2-, CO_2- und Anästhesiegasmessung
- Aktoren
  - 20 Ventile
  - 1 Proportionalventil
- Prüfabläufe
  - zur Zeit ca. 25 unterschiedliche Prüfungen
- Bildschirmseiten
  - zur Zeit ca. 80 verschiedene Seiten

Elektronik



- CPU-Board
  - ETX-Format
  - Geode CPU, 266 MHz
  - 24 MByte RAM
  - 64 MByte Flash Card
- Display
  - LCD mit 320 x 240 Pixel

Übersicht

- Gerätevorstellung
- Softwareentwicklung
  - Entwicklungsumgebung
  - Softwarearchitektur
  - Zeitverhalten
- Fazit

Effizienzsteigerungen

- Komfortable Entwicklungsumgebung
  - Kurze Debugzyklen
  - Umfangreiche Testmöglichkeiten
- Moderne Anwendungsarchitektur
  - Objekttechnologie
  - Zustandsautomaten
  - Skriptsprachen
- Geringe Komplexität
  - Keine aufwändigen Nebenläufigkeiten
  - Keine komplizierten Ereignissteuerungen

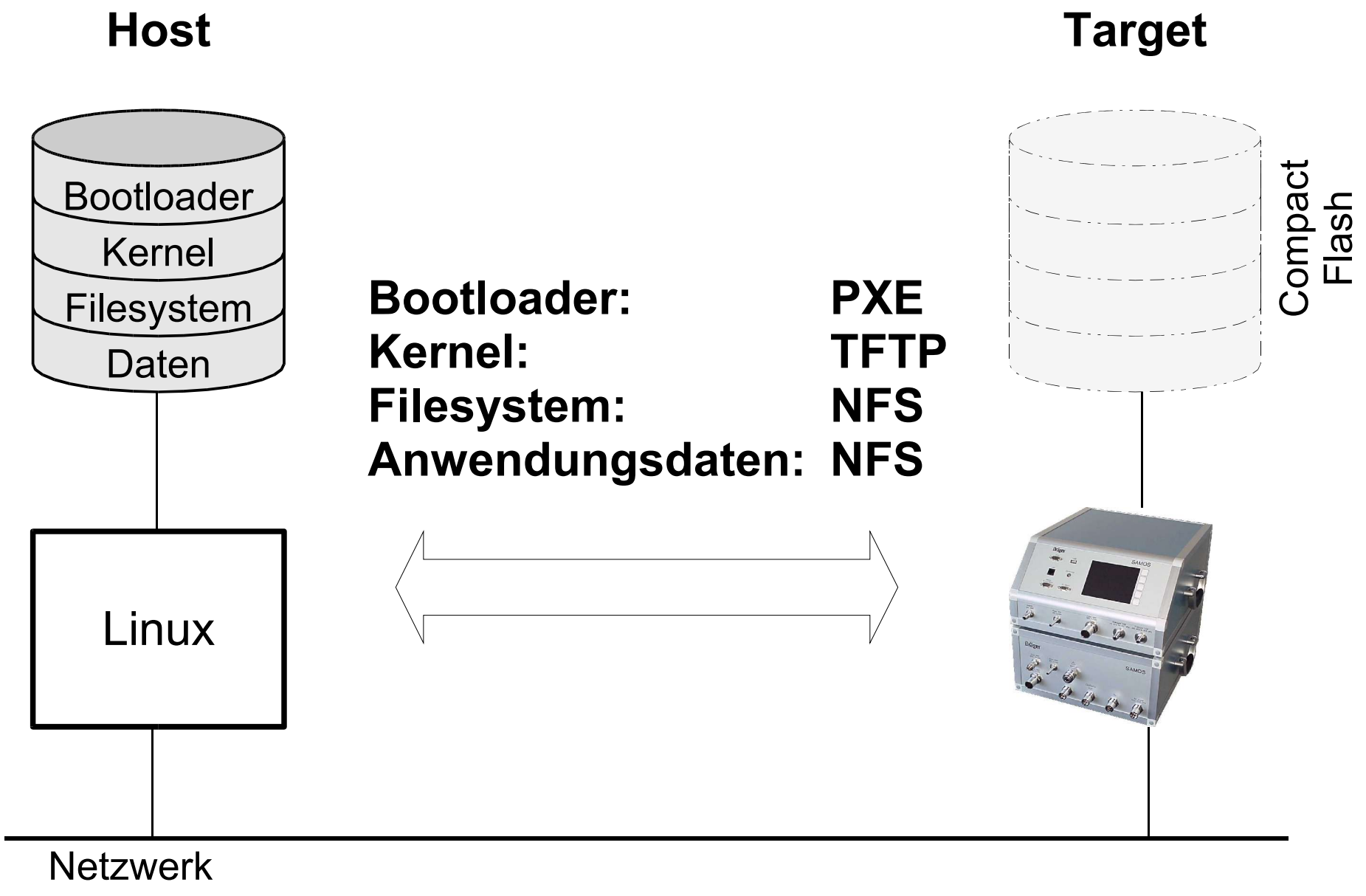
Übersicht

- Gerätevorstellung
- Softwareentwicklung
  - Entwicklungsumgebung
  - Softwarearchitektur
  - Zeitverhalten
- Fazit

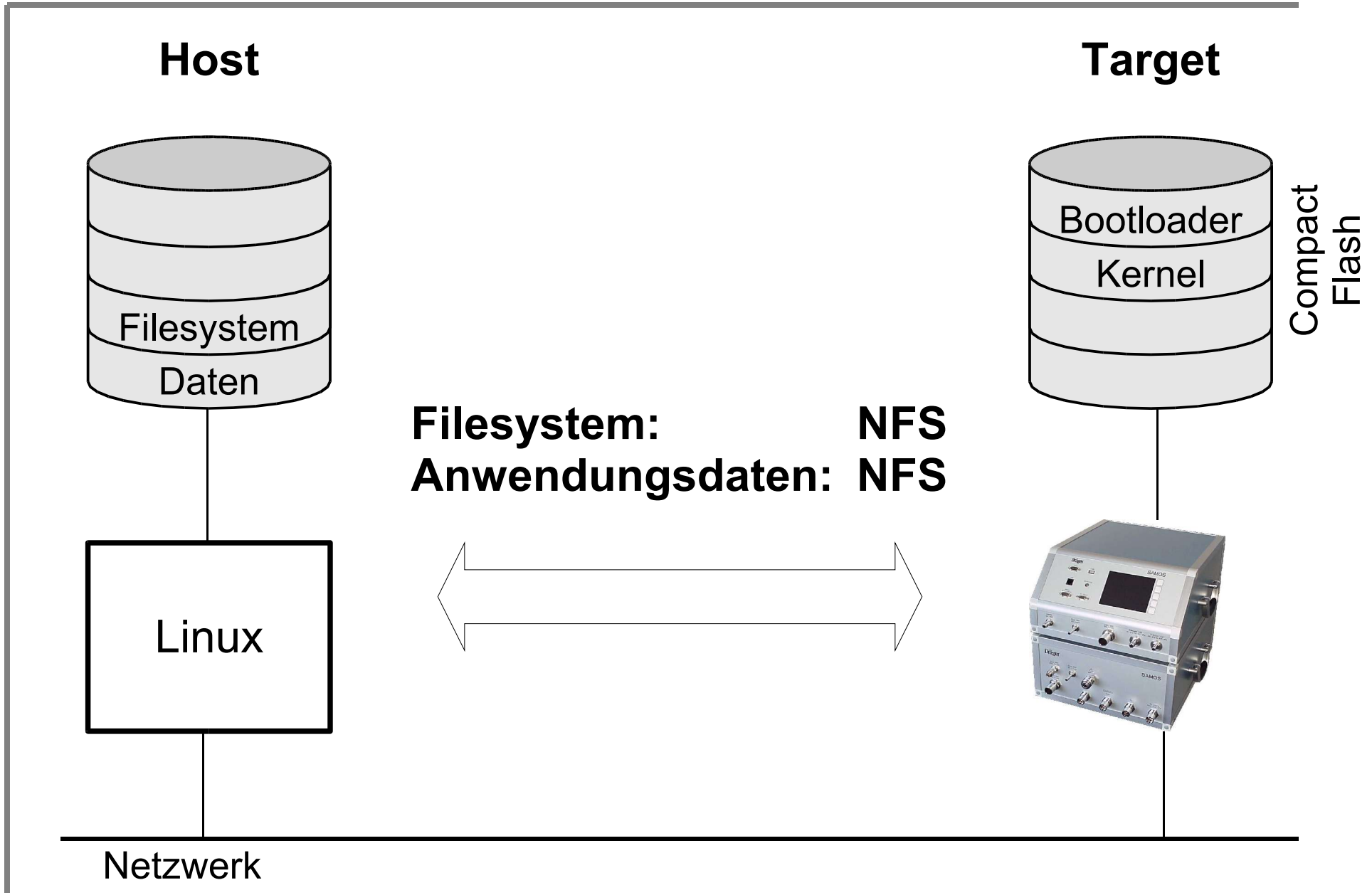
Entwicklungsumgebung

- Teilaspekte
  - Filesystem
    - Beschleunigung des Debugzyklusses
    - Für Auslieferung: Readonly Filesystem
  - Grafische Benutzeroberfläche
    - Kompromiss zwischen
      Funktionenvielfalt und
      Ressourcenverbrauch
  - Integrierter Webserver
    - Hilfreich für
      Fehlersuche und
      automatisches Testen

Inbetriebnahme

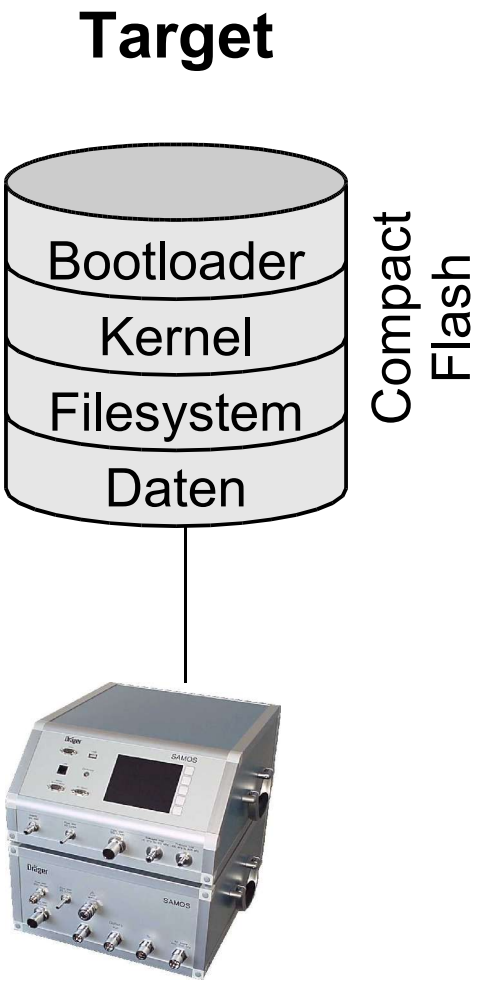


Entwicklungsphase



Auslieferung

- Aufteilung des Filesystems:
  - Readonly
    - Bootloader
    - Kernel
    - Filesystem
  - R/W
    - Persistente Anwendungsdaten
      z.B. Konfiguration
  - RAM-Disk
    - Flüchtige Anwendungsdaten
      z.B. Logdateien



Embedded Filesysteme

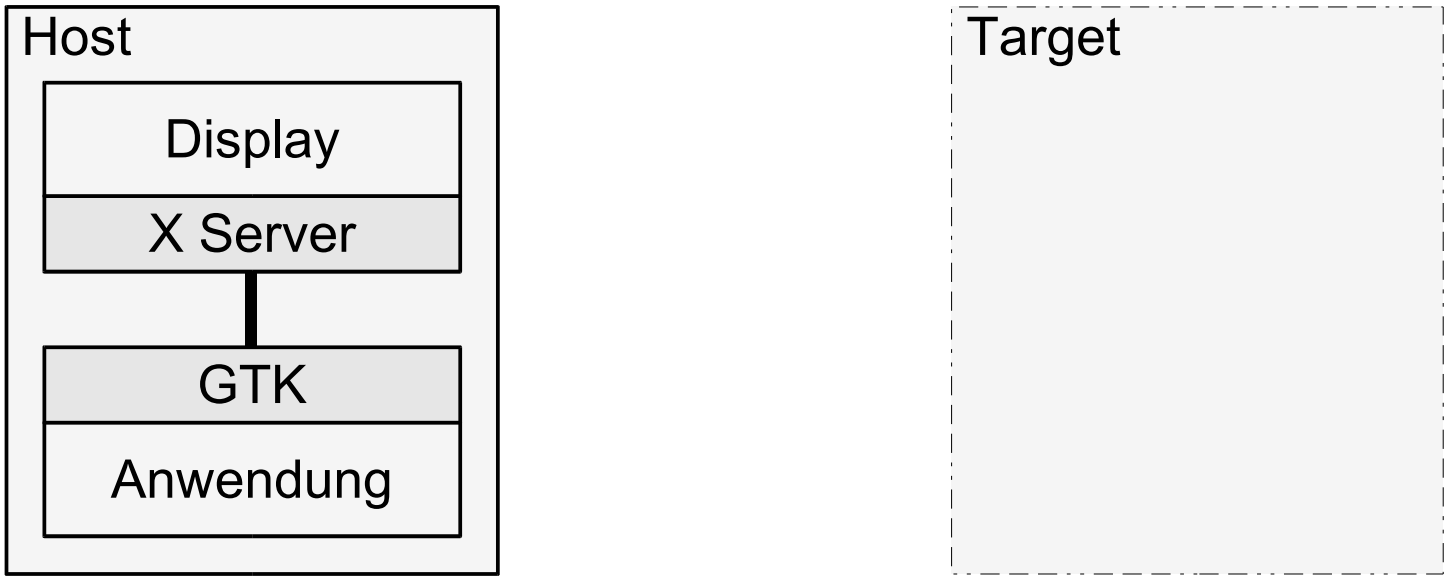
Filesystem	Read-only	Platzbedarf für 27 MByte	Lesen von 27 MByte	
			Dauer	CPU-Belastung
Minix	Nein	27 MByte	33 s	51 %
Romfs	Ja	27 MByte	73 s	28 %
Cramfs	Ja	16 MByte	36 s	70 %
Cloop	Ja	14 MByte	33 s	88 %

- Cloop
  - Verwendung: Knoppix Live CD
  - Filesystem ist in einer großen Datei abgelegt
  - Kann für Softwareupdate einfach ausgetauscht werden

Grafische Benutzeroberfläche

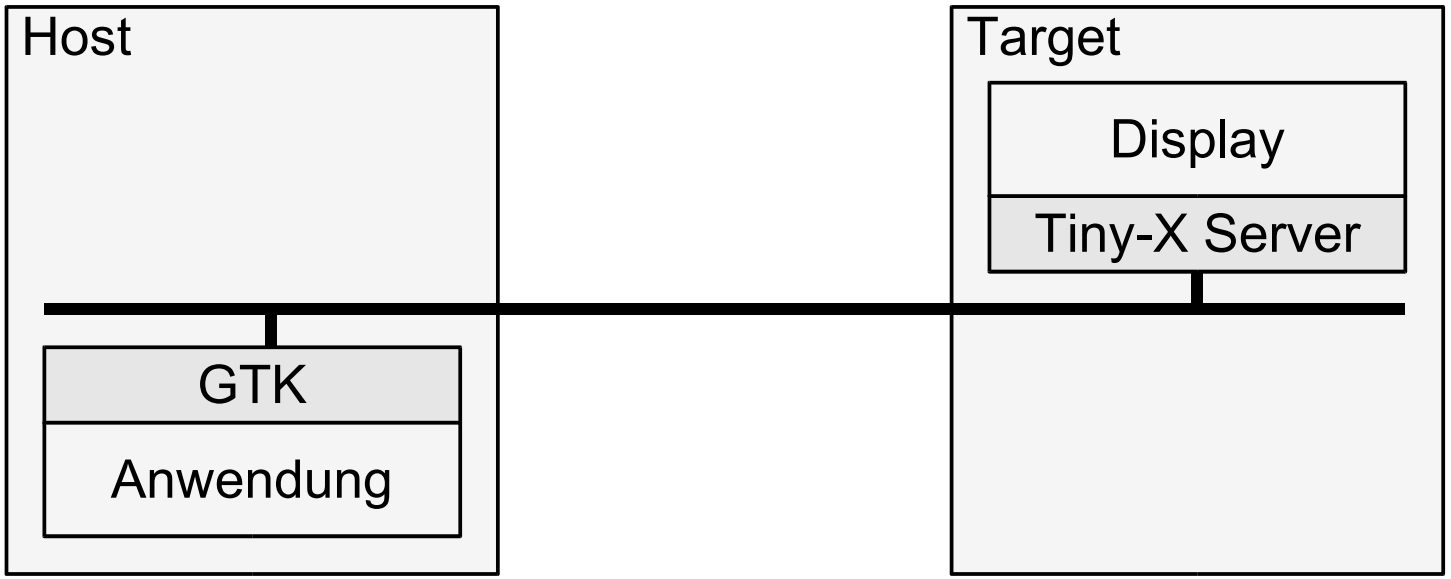
- Anforderungen
  - geringer Ressourcenbedarf
  - Umfangreiche Bibliothek
  - Unterstützung für spezielle Bedienelemente
    - Hardkeys
    - Touchpanels
- Linux bietet
  - Vielzahl von Grafiksystemen,
    auch speziell für eingebette Systeme

X Window und GTK



- X Window und GTK, weil
  - bewährte Software
  - große Entwicklergemeinde
  - hohe Flexibilität während der Entwicklung

Tiny-X Server



- Tiny-X Server
  - Bestandteil von XFree86 4.0
  - Speziell entwickelt für Low-Memory-Systeme
    - Speicherbedarf < 1 MByte

Tiny-X Server

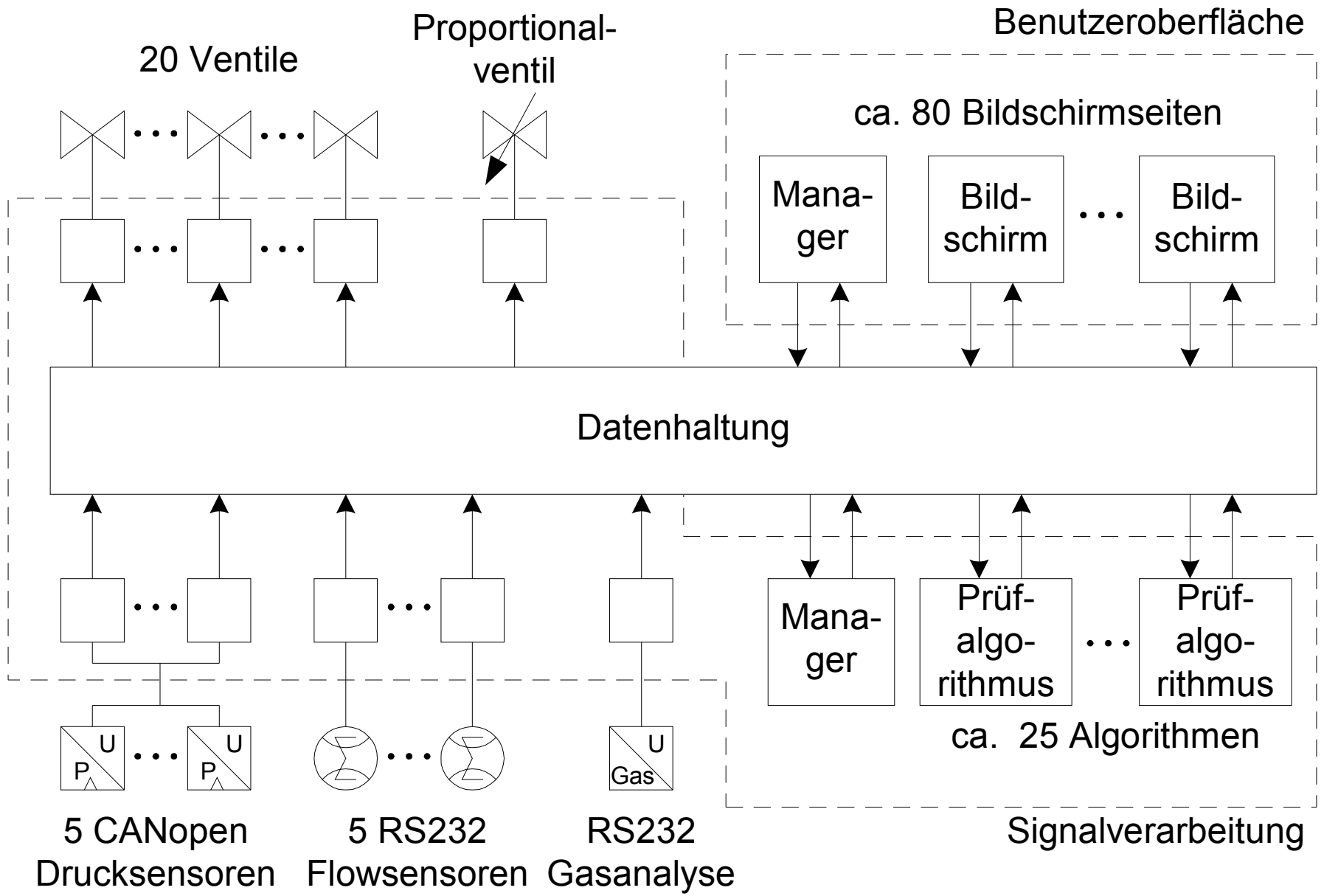
Integration Webserver

- Eingebetteter Webserver ermöglicht:
  - Fernabfrage aller Mess- und Steuerwerte
  - Setzen von Mess- und Steuerwerten
  - Fernsteuerung des Gerätes
- Zum Testen:
  - per
    http und
    einfachen Skripten
    können automatisierte Testläufe programmiert werden

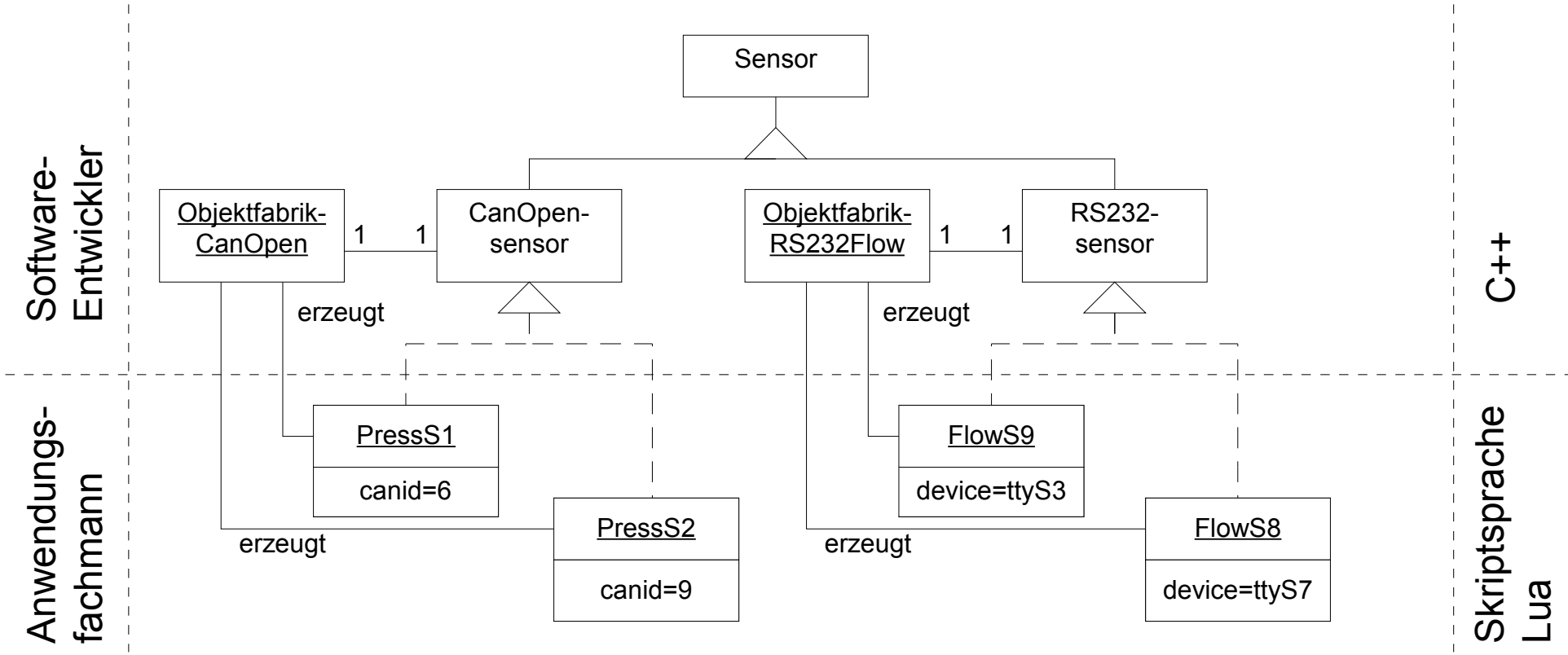
Übersicht

- Gerätevorstellung
- Softwareentwicklung
  - Entwicklungsumgebung
  - Softwarearchitektur
  - Zeitverhalten
- Fazit

Blockdiagramm



Softwarearchitektur



Beispiel Skript

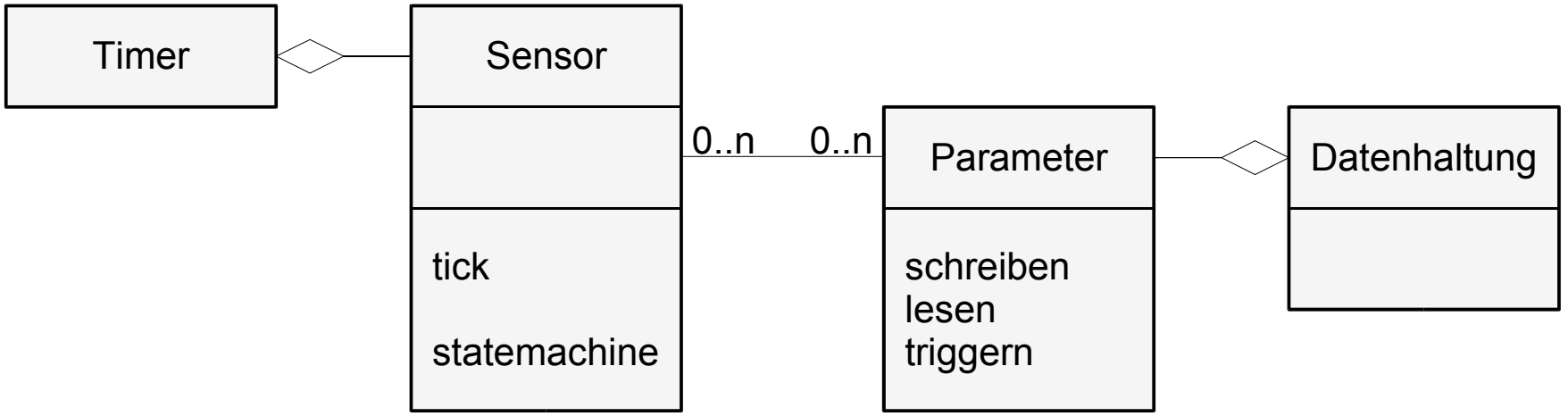
- Konfiguration
  - mittels einfacher Skriptsprache „LUA“ direkt durch den Anwendungsfachmann
  - Lua:
    - Entwickelt von der Tecgraf in Rio
    - machtvolle Konstrukte zur Datenbeschreibung
    - gern verwendet von Spieleprogrammierer

```
PRESS_SDOS_100 ="0x1a00,1,0x61300120,4/" .. -- pdo mapping
                "0x210A,0,2,1/"          .. -- switch to 100 Hz sampling
                "0x210B,0,200,1"

devices.PressSensor1 =
    {'DeviceCanopen',
      {node=6,sdos=PRESS_SDOS_100}}

params.PressS1 =
    {'PressSensor1',
      {index=0,format='float',factor=1.0}}
```

Software-Architektur



- Periodische Aktivierung der Objekte
- Herzstück: Statemachine
- Messwerte und Ereignisse über Datenhaltung
- Ereignisse werden gepollt

„Tickende“ Objekte

- Ermöglichen
  - einfache Überwachung des Zeitverhaltens
  - Vermeidung von Synchronisations-Problemen
  - Zustandsautomat ermöglicht robusten Aufbau
- Erfordern
  - kurze Rechenalgorithmen
  - nicht blockierende Systemaufrufe
  - genauen Timer

Übersicht

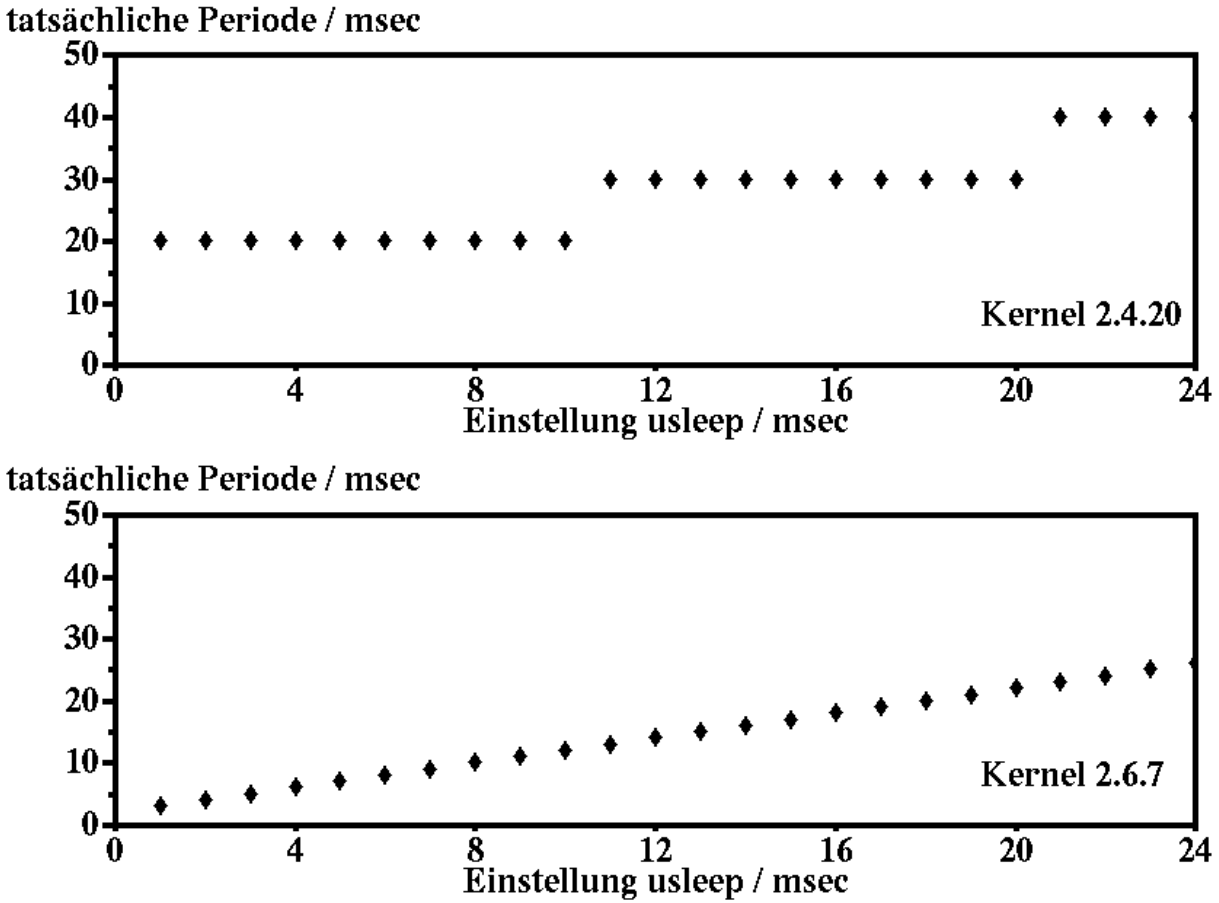
- Gerätevorstellung
- Softwareentwicklung
  - Entwicklungsumgebung
  - Softwarearchitektur
  - Zeitverhalten
- Fazit

Zeitverhalten

- Anforderungen
  - Anwendung vollständig im User-Mode
  - Keine Echtzeiterweiterung
  - Aufteilung in
    - Signalverarbeitung:
      Takt ≤ 10 ms
      unbedingt einzuhalten
    - Benutzeroberfläche:
      Takt ≤ 20 ms

Linux: Periodische Aktivierung

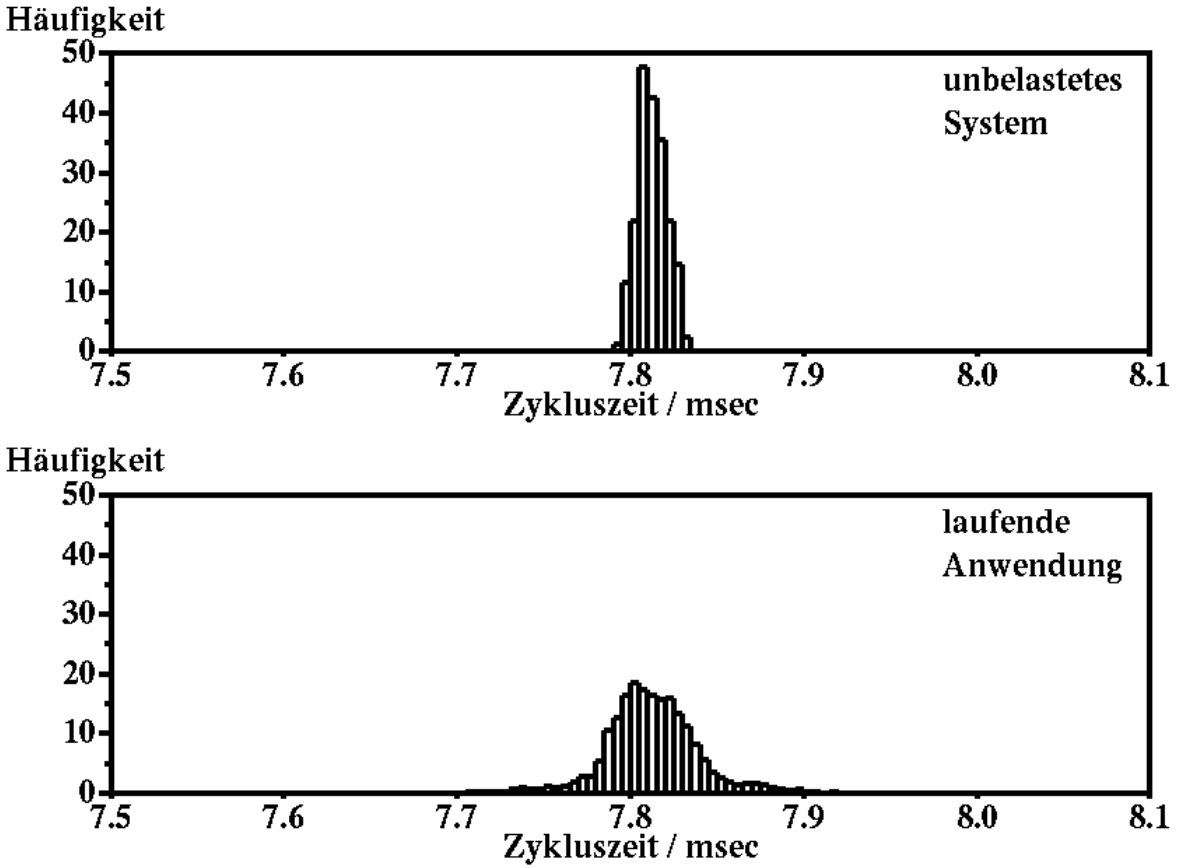
- Realisierung mittels Warteschleife



  - Problem:
    Takte können verpasst werden

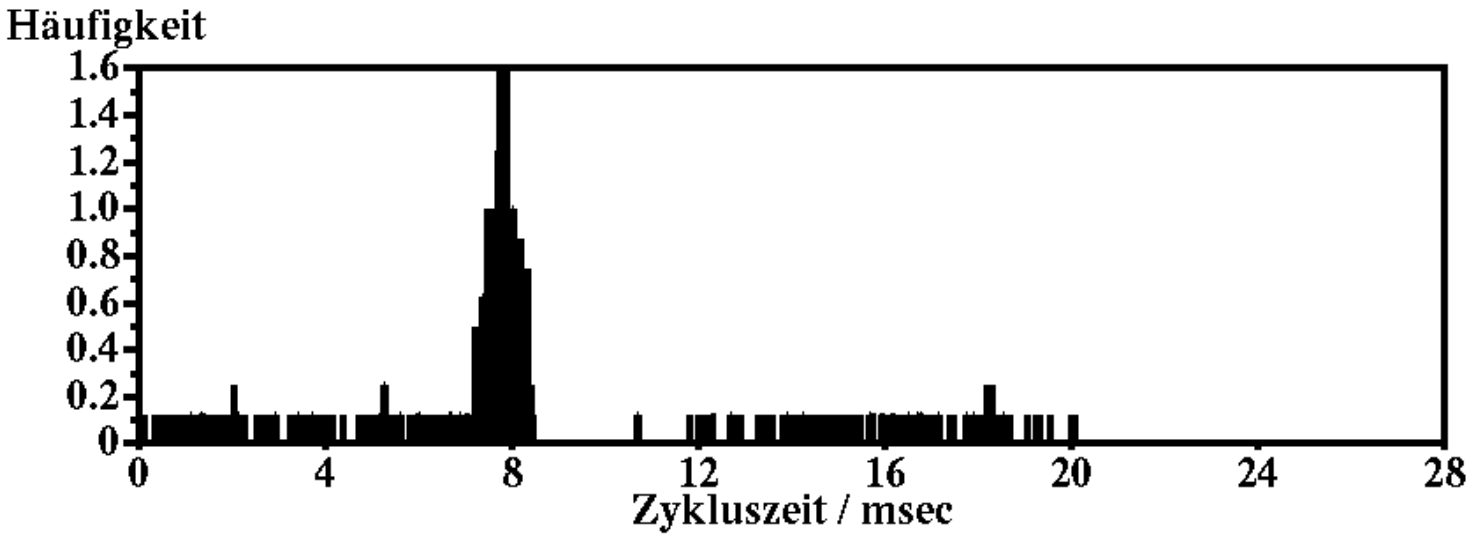
Linux: Periodische Aktivierung

- Realisierung mittels Echtzeituhr
  - Spezielles Device mit eigenem Interrupt
  - Frequenz von 2 Hz bis 8192 Hz
  - Anzahl verpaßter Interrupts



Linux: Periodische Aktivierung

- Aktivierungsperiode bei zusätzlicher Belastung durch das Netzwerk
  - keine wesentlichen Unterschiede zwischen Kernel 2.4 und 2.6



Übersicht

- Gerätevorstellung
- Softwareentwicklung
  - Entwicklungsumgebung
  - Softwarearchitektur
  - Zeitverhalten
- Fazit

Fazit

- Schnelle und flexible Entwicklung durch
  - Komfortable Entwicklungsumgebung
    - vielfältige Programmierwerkzeuge
    - umfangreiche Bibliotheken
    - kurze Debugzyklen
    - umfangreiche Testmöglichkeiten
  - Verwendung einer simplen und robusten Software-Architektur
    - Objektorientierung
    - Zentrale Datenhaltung
    - Vermeidung aufwändiger Nebenläufigkeiten und komplizierter Ereignissteuerungen
  - Einbindung der Anwendungsfachleute in die Softwareentwicklung

Fazit

- Aber ...
  höhere CPU-Anforderungen und
  höherer Speicherbedarf durch
  - Funktionsbibliotheken
    - Grafik
    - Skriptsprache
  - und einfache und robuste SW-Architektur
    - Periodische Aktivierung, Polling

Fazit

- Linux ermöglicht
  optimale Anpassung an
  Projekterfordernissen

- Allerdings:
  - Große Flexibilität kostet hohen Einarbeitungsaufwand
  - Gilt auch bei Verwendung spezieller Linux-Distributionen