Betriebssysteme

Vorlesung im Herbstsemester 2008
Universität Mannheim

Kapitel 3: Laufzeitunterstützung aus Anwendersicht

Prof. Dr. Felix C. Freiling
Lehrstuhl für Praktische Informatik 1
Universität Mannheim

Ankündigungen 18.9.2008

- Voraussichtliche Prüfungstermine:
  - Erster Termin: 19. Dezember 2008
  - Zweiter Termin: 12. Februar 2008 9
- Bei aktuellen Teilnehmerzahlen ist mit einer schriftlichen Prüfung zu rechnen
  - Aktuelle Arbeitshypothese: für „großen" und „kleinen" Kurs zwei unterschiedliche Klausuren

Motivation

- Direkter Umgang mit Rechnerhardware ist wie in Kapitel 2 beschrieben problematisch
  - Schnittstellen zu den Geräten sind zwar standardisiert, aber trotzdem unhandlich
  - Direkte Programmierung der Hardware (z.B. den Unterbrechungsmechanismus) führt zu schwer durchschaubaren Softwarestrukturen
  - Direkter Zugriff auf die Hardware führt zu Schutzproblemen im Mehrbenutzerprinzip (gegenseitige Beeinflussung, Diebstahl von Ressourcen)
  - Koordination unabhängiger Benutzeraktivitäten erfordert Koordinationsaufwand, der ohne Systemunterstützung schwer zu handhaben ist
- Aber was wollen wir eigentlich an Laufzeitunterstützung haben?

Einordnung in die Vorlesung

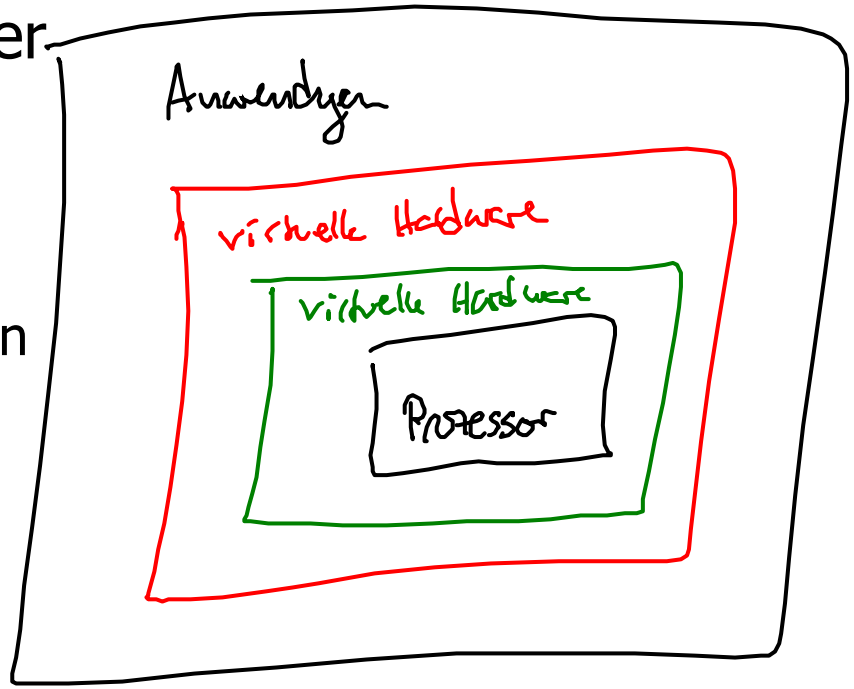
- Gliederung der Vorlesung:
  1. Einführung und Formalia
  2. Auf was baut die Systemsoftware auf? Hardware-Grundlagen
  3. **Was wollen wir eigentlich haben? Laufzeitunterstützung aus Anwendersicht**
  4. Verwaltung von Speicher: Virtueller Speicher
  5. Verwaltung von Rechenzeit: Virtuelle Prozessoren (Threads)
  6. Synchronisation paralleler Aktivitäten auf dem Rechner
  7. Implementierungsaspekte

Übersicht

- Mögliche Dienste im Laufzeitmodell (Übersicht)
- Unverzichtbare Dienste
- Elementare Laufzeitmodelle
- Erweiterung der elementaren Laufzeitmodelle

Laufzeitmodell

- Wir möchten den direkten Zugang zur Rechnerhardware verbergen
  - Nur indirekten Zugang zulassen
  - Dienstschicht zwischen Anwendung und Hardware realisieren
  - Dienstschicht realisiert eine virtuelle Maschine
- Inkrementelle Erweiterung der Maschinenhierarchie
  - Bereitgestellte Abstraktionen bilden das Laufzeitmodell (den Rahmen für Anwendungen zur Laufzeit)



Häufig verwendete Dienste

- Systembedienung
  - (Ggf. grafische) Kommandoschnittstelle
- Prozessmanagement
  - Verwaltung von Rechenaufträgen
  - (Ggf. zeitgesteuerter) Mechanismus für Ausnahmebehandung
- Prozessinteraktion
  - Informationsaustausch/Synchronisation zwischen Prozessen
- Datenhaltung
  - Langlebige Aufbewahrung von Daten und Programmen
- Gerätemanagement
  - Komfortabler Zugriff auf Geräte, z.B. in Form von "Dateien"
  - Schwieriger sind Abstraktionen für interaktive Geräte (wie Tastatur, Maus)

Design-Fragestellungen

- Welche Dienste soll man bereitstellen und welche nicht?
  - Zielt auf die Offenheit des Laufzeitmodells, d.h. kann man aus den bestehenden Diensten neue (höherwertige) Dienste bauen?
  - Sind die Dienste orthogonal (frei kombinierbar)?
- Welche Dienstmenge ist minimal?
  - Gibt es einen inneren Zusammenhang zwischen den Diensten?
  - Gibt es elementare Dienste, auf die andere Dienste aufsetzen können?
    - Erhöht die Wirtschaftlichkeit der Implementierung
    - Maximiert die Offenheit
- Wo sollen einzelne Dienste realisiert werden?
  - Frage nach der Architektur des Laufzeitsystems
  - Laufzeitsystem = Menge an Systemsoftwarekomponenten
    - Eine konkrete Instantiierung des Laufzeitmodells

Übersicht

- Mögliche Dienste im Laufzeitmodell
- Unverzichtbare Dienste
- Elementare Laufzeitmodelle
- Erweiterung der elementaren Laufzeitmodelle

Vorüberlegungen

- Die beiden grundlegenden Ressourcen des Rechners sind Prozessorzeit und Speicher
  - Ein Laufzeitmodell muss den Zugriff auf diese Ressourcen ermöglichen
- Randbedingung 1: Keine Monopolisierung der Ressourcen
  - Keine Anwendung darf die Ressourcen des Rechners dauerhaft und vollständig für sich beanspruchen
  - Andernfalls hätten andere Anwendungen keine Chance mehr, selbst in den Genuß der Ressourcen zu kommen
- Randbedingung 2: Isolation der Anwendungen untereinander
  - Unerwünschte wechselseitige Beeinflussung muss ausgeschlossen sein
  - Gewollte Interaktionen müssen in kontrollierter Weise erfolgen

Prozesse

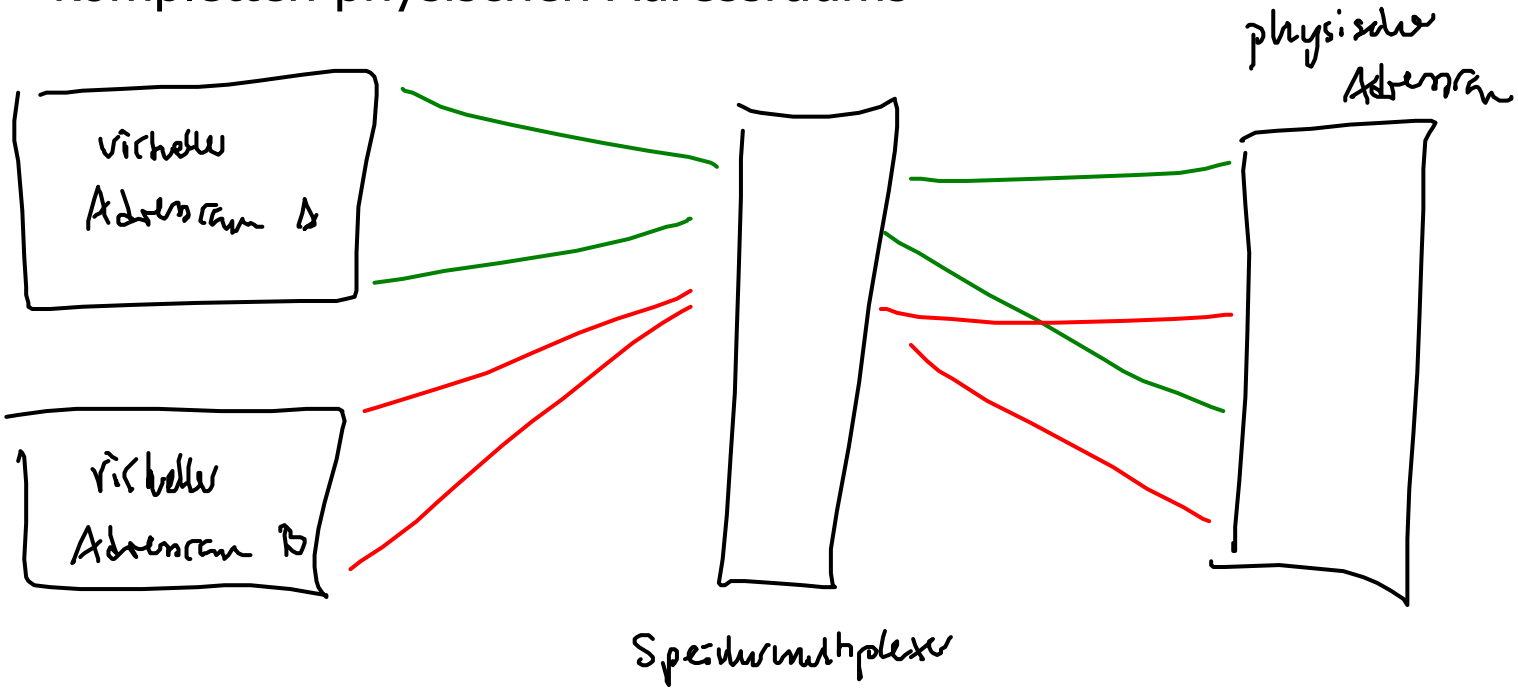
- Die Dienste Prozessmanagement und Prozessinteraktion realisieren den nicht-monopolisierenden, geschützten Zugriff auf **Prozessor und Speicher**
  - Sollten in keinem elementaren Laufzeitmodell fehlen
- Zentrale Abstraktion: Der Prozess
  - Siehe: E.W. Dijkstra: Cooperating sequential processes.
  - Prozesse sind dynamische Objekte, die sequentielle Aktivitäten in einem System repräsentieren
- Ein Prozess besteht aus:
  - Adressraum
  - Im Adressraum gespeicherte Handungsvorschrift (Programm)
    - Abstraktion der Ressource Speicher
  - Aktivitätsträger, der mit der Handlungsvorschift verknüpft ist und sie ausführt (Thread, engl. Faden, sequentieller Kontrollfluß)
    - Abstraktion der Ressource Prozessorzeit

Adressräume

- Virtueller Adressraum ist die Abstraktion des physischen Speichers: virtueller Speicher
  - Unabhängig von Speichertechnologie
  - Unabhängig von den beschränkten Ausbaumöglichkeiten des physischen Speichers
- Adressraum ist eine Folge von Speicherzellen (gewöhnlich Bytes), die von Adresse 0 an aufwärts durchnumeriert sind
  - Adressräume können bei Bedarf erzeugt und gelöscht werden
  - Adressräume sind Container für Programme und Daten
  - Adressräume sind gegeneinander abgeschottet
    - Thread in Adressraum A kann (ohne zusätzliche Vereinbarungen) nicht auf Adressraum B zugreifen
- Zuordnung der Adressräume zu den physischen Ressourcen geschieht über einen **Speichermultiplexer**

Speichermultiplexer

- Physischer Speicher wird reihum verschiedenen virtuellen Adressräumen zur Verfügung gestellt
- Für Prozesse ist der Mechanismus transparent:
  - Jeder Prozess denkt, er habe jederzeit exklusive Verwendung des kompletten physischen Adressraums

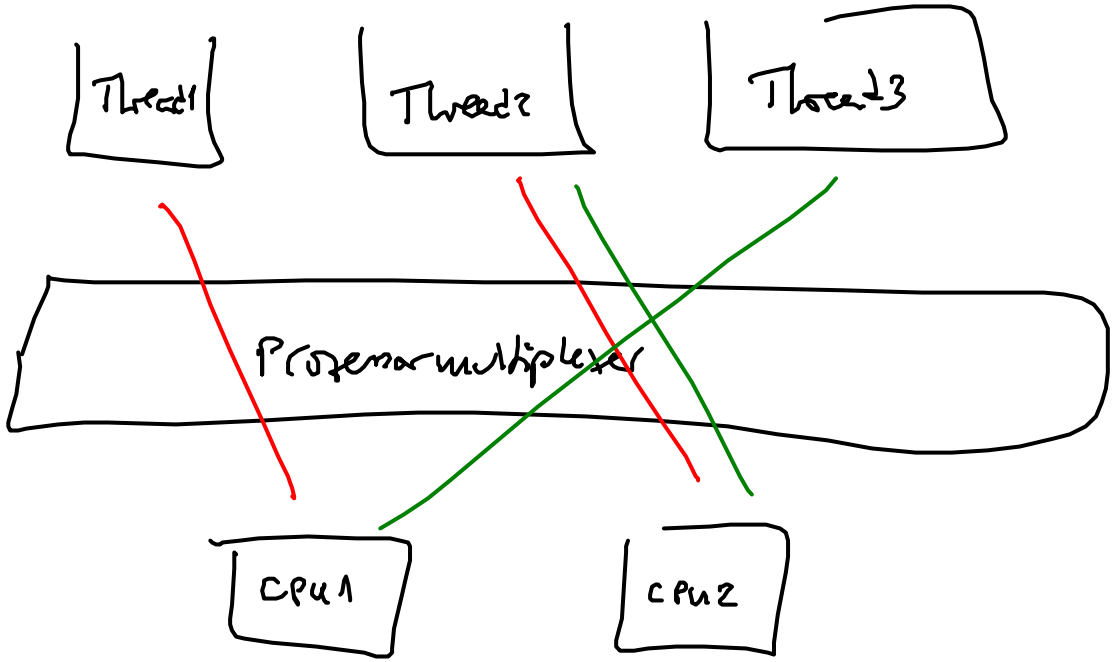


Threads

- Thread ist die Abstraktion eines physischen Prozessors
  - Träger sequentieller Aktivität
  - Aktivität definiert durch das dem Thread zugeordnete Programm
- Ein Thread ist ein **virtueller Prozessor**
  - Threads können bei Bedarf erzeugt und gelöscht werden
  - Selbstterminierung eines Threads, wenn das zugeordnete Programm bis zum Ende abgewickelt wurde
  - In einem Adressraum können mehrere Threads existieren
    - Adressraum mit einem Thread ist der "klassische Prozess"
- Zuordnung von Threads zu Prozessoren geschieht durch einen **Prozessormultiplexer**

Prozessormultiplexer

- Physische Prozessoren werden reihum verschiedenen Threads (virtuellen Prozessoren) zur Verfügung gestellt
- Für Threads ist der Mechanismus transparent:
  - Jeder Thread denkt, er habe jederzeit exklusiven Zugriff auf den physischen Prozessor



Prozessinteraktion

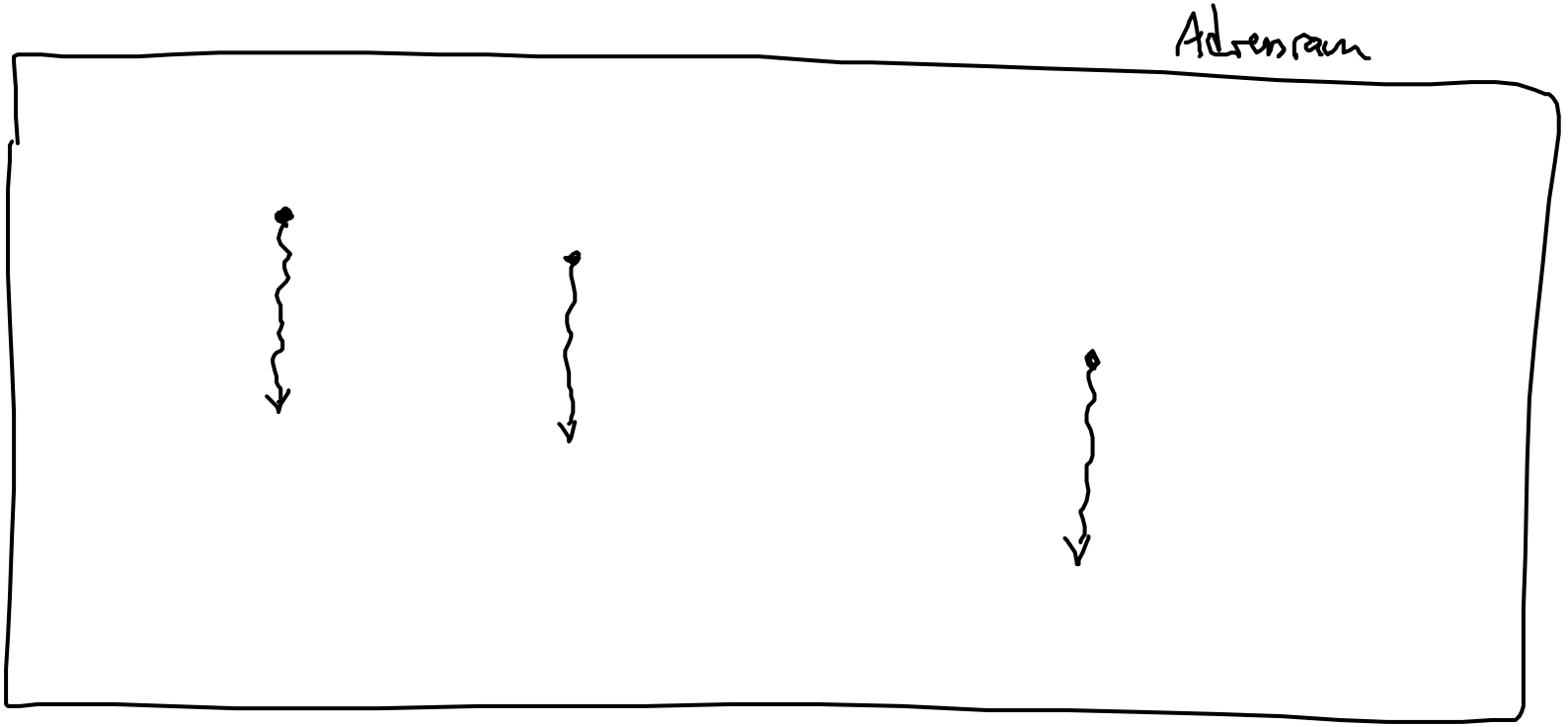
- Anwendungen möchten/müssen miteinander interagieren
  - Interaktion wird durch Prozessinteraktionsdienst ermöglicht
  - Kommunikation über gemeinsamen Speicher oder Nachrichten
- Zwei grundlegende Interaktionsmuster:
  - Konkurrenz: mehrere Prozesse möchten gleichzeitig auf ein exklusiv benutzbares Betriebsmittel zugreifen (z.B. einen Drucker)
    - Es darf immer maximal ein Prozess gleichzeitig zugreifen
    - Problem des wechselseitigen Ausschlusses (mutual exclusion)
    - Problem der Prozesssynchronisation
  - Kooperation: Prozesse möchten gezielt Informationen austauschen
    - Hier müssen Prozesse sich gegenseitig kennen
    - Häufiges Muster: Auftragsbeziehung
      - Producer/Consumer: Producer stellt Daten her, die der Consumer verarbeitet
      - Client/Server: Client stellt Anfrage und wartet auf Resultat vom Server

Übersicht

- Mögliche Dienste im Laufzeitmodell
- Unverzichtbare Dienste
- Elementare Laufzeitmodelle
  - Betrachten jetzt eine Reihe von vollständig elementaren Laufzeitmodellen
  - Erhält man durch geeignete Verknüpfung der Abstraktionen Adressraum, Thread und Prozessinteraktion
- Erweiterung der elementaren Laufzeitmodelle

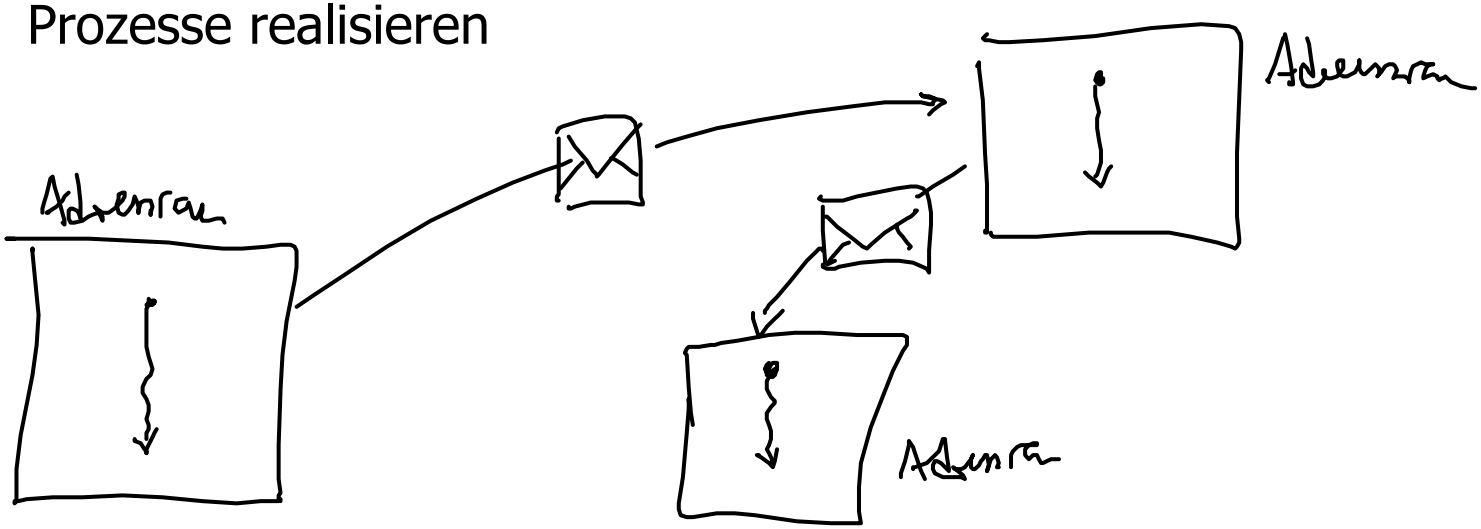
Speichergekoppelte Prozesse

- Mehrere Threads in einem Adressraum (Team)
  - Kommunikation effizient über gemeinsamen Speicher
  - Ein Team kann sogar verteilt arbeiten (mit Hilfe der DSM-Abstraktion, Distributed Shared Memory)
  - Interaktion mit anderen Teams undefiniert



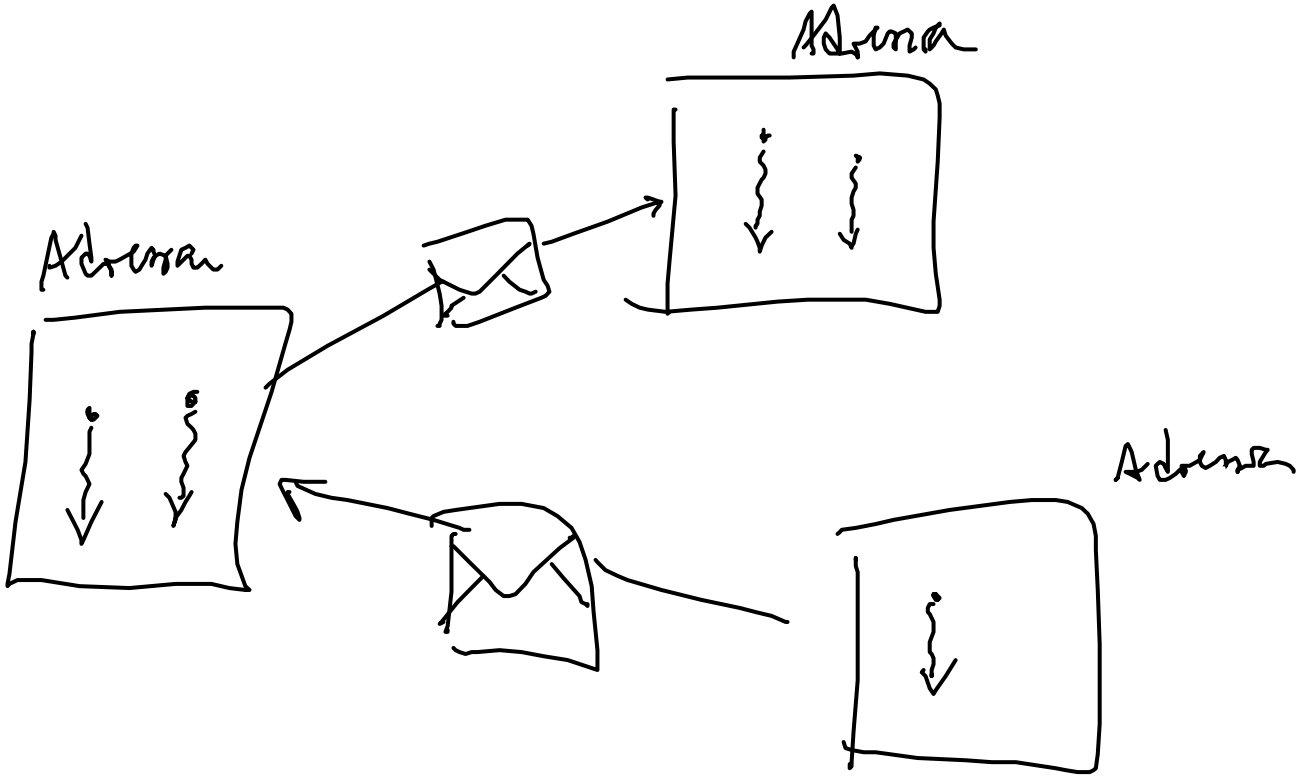
Nachrichtengekoppelte Prozesse

- Mehrere Prozesse mit jeweils einem Adressraum und einem Thread
  - Da Adressräume disjunkt, erfolgt Kommunikation über Nachrichten
  - Prozesse werden auf einzelnen Rechnern allokiert (naheliegende Abbildung auf Rechnernetze)
    - Für Prozesse auf demselben Rechner gibt es effiziente Implementierungen Nachrichtenbasierter Kommunikation
  - Offenes Laufzeitmodell: Zusätzliche Dienste durch zusätzliche Prozesse realisieren



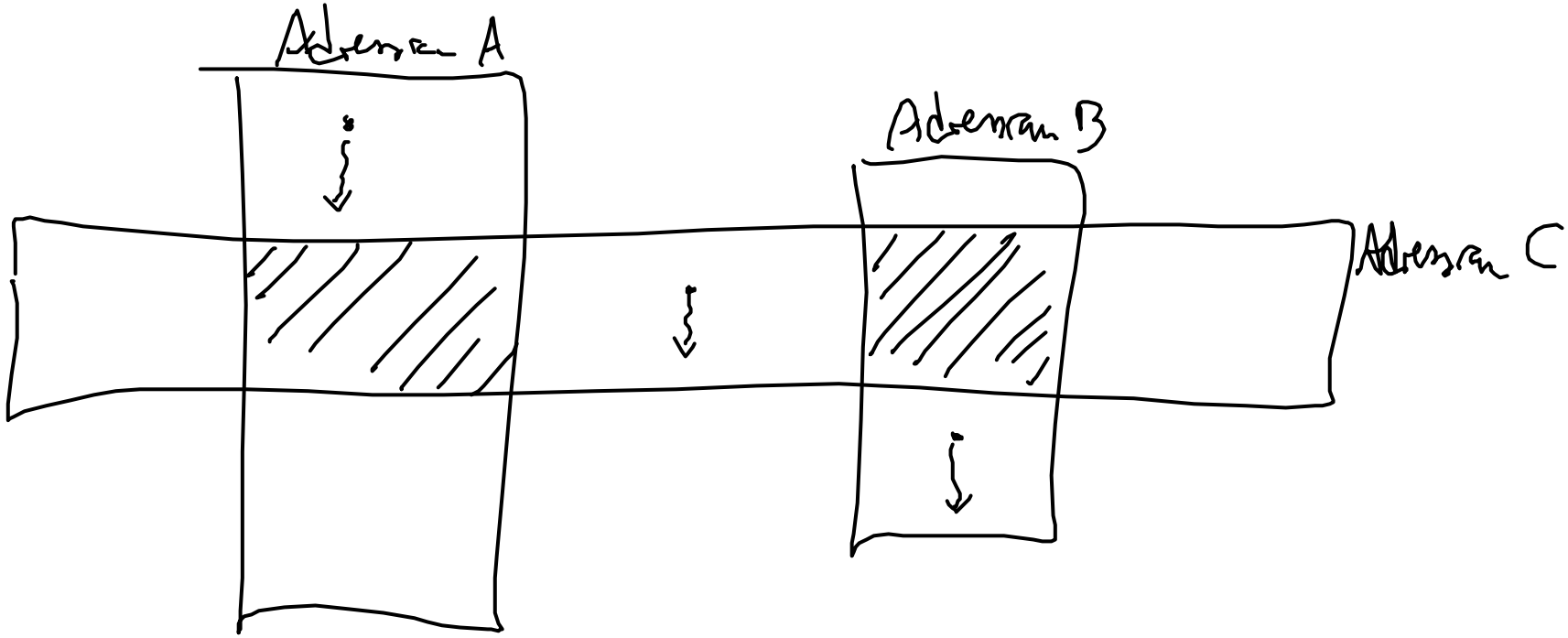
Nachrichtengekoppelte Teams

- Kombination der beiden vorherigen Modelle
  - Innerhalb der Teams schnelle Interaktion durch gemeinsamen Speicher (enge Kopplung)
  - Nachrichtenbasierte Interaktion (lose Kopplung) und damit Offenheit zwischen Prozessen



Überlappende Adressräume

- Vereinbarung gezielter Überlappung einzelner Adressräume
  - Gut für Anwendungen, die gemeinsam große Datenmengen verarbeiten
  - Probleme: Verteilung auf Prozessoren nicht mehr so einfach
    - Lokalisierung der Adressraumteile nichttrivial
    - Erhaltung der Konsistenz des verteilten Speichers nichttrivial

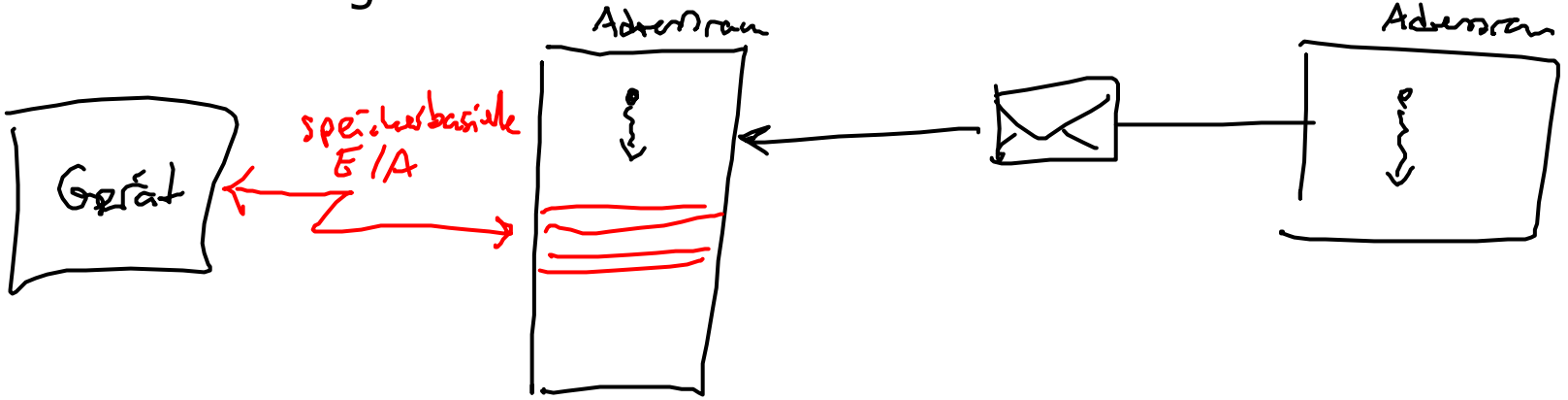


Übersicht

- Mögliche Dienste im Laufzeitmodell
- Unverzichtbare Dienste
- Elementare Laufzeitmodelle
- Erweiterung der elementaren Laufzeitmodelle
  - Elementare Laufzeitmodelle können in vielfältiger Weise ergänzt werden
  - Zusätzliche Dienste können abgeleitet oder hinzugefügt werden
  - Es folgen einige Beispiele für solche Dienste

Gerätemanagement

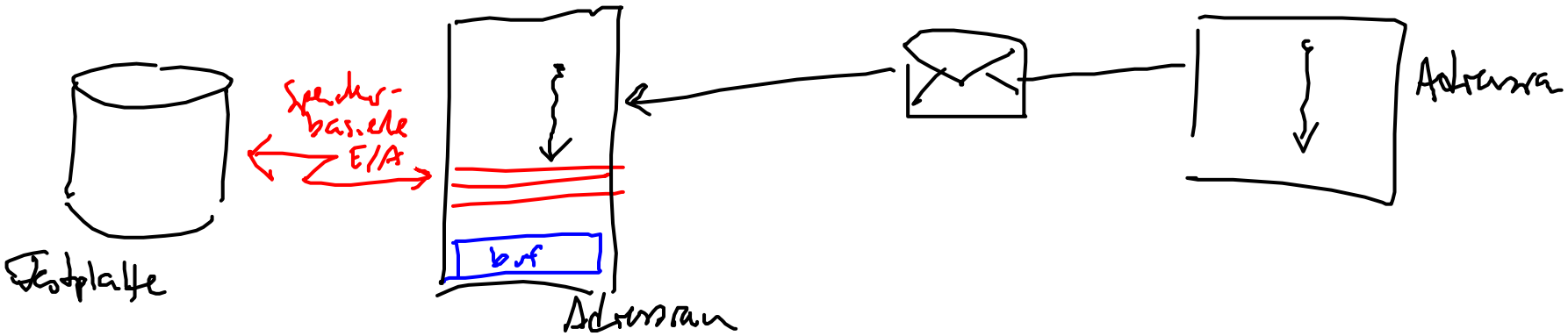
- Laufzeitmodell muß den kontrollierten Zugriff auf Geräte erlauben
  - Ansatz bei einer speicherbasierten Ein/Ausgabe:
    - Einblenden der Geräteregister in den Adressraum eines Prozesses/Teams
    - Gerätedienst wird durch das Team erbracht
    - Zugriff auf den Dienst durch Nachrichten



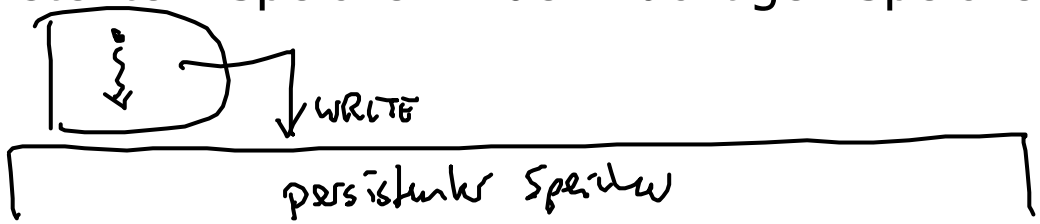
  - Gerätemanagement wird deshalb oft als (von Adressraum, Thread, und Prozessinteraktion) abgeleiteter Dienst betrachtet

Datenhaltung

- Realisierung als abgeleiteter Dienst
  - Prozess/Team, welches den Zugriff auf die Festplatte kapselt
  - Zugriff über bestehende Mechanismen der Prozessinteraktion

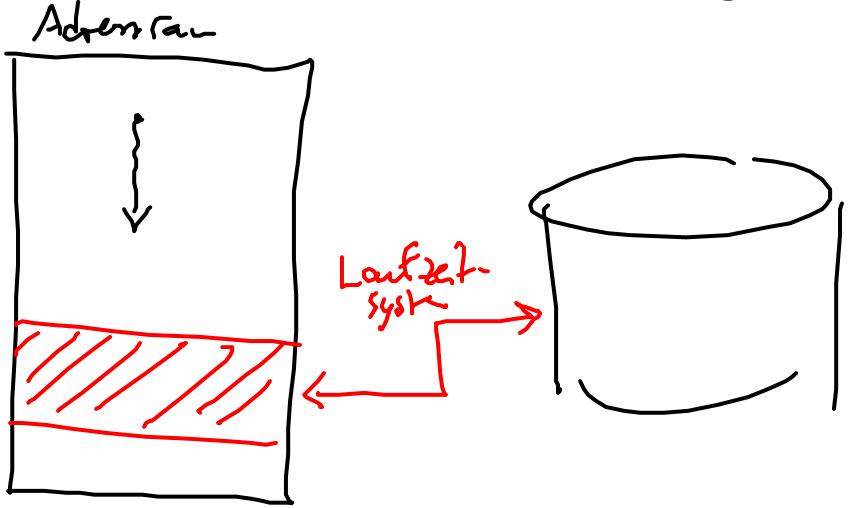


- Erweiterung des Laufzeitmodells: persistenter Speicher
  - Üblich: Konzept einer Datei
  - Operationen WRITE (persistentes Speichern) und READ (Lesen von persistentem Speicher in den flüchtigen Speicher)



Persistente Segmente

- Trennung von flüchtigem und nicht-flüchtigem (persistentem) Speicher ist etwas unnatürlich
- Überwindung der Trennung durch persistente Segmente
  - Bestimmte Speicherbereiche im Adressraum werden als persistent vereinbart
  - Transport zwischen flüchtigem und nicht-flüchtigem Speicher wir durch das Laufzeitsystem durchgeführt
    - Kann bis zu transaktionsorientierten Zugriffssemantiken gehen



Echtzeitanwendungen

- Entzug des direkten Zugriffs auf die Hardware ist Grundlage für faire Zuteilung der Ressourcen
- Harte Echtzeitanwendungen: Anwendungen mit fest vorgegebenen Zeitschranken für die Ausführung bestimmter Operationen
  - Beispiel: Avionik-Systeme
  - Nichtdeterministisches Verdrängen von Threads nur bis zu einem gewissen Grad tolerierbar
- Laufzeitmodelle müssen um Möglichkeiten der Einflußnahme auf die Prozessor- und Speicherzuteilung ergänzt werden
  - Beispiele: festgelegte Zuteilungsstrategien basierend auf Prioritäten

Zusammenfassung

- Mögliche Dienste im Laufzeitmodell
- Unverzichtbare Dienste
  - Adressraum, Thread, Prozessinteraktion
- Elementare Laufzeitmodelle
  - Speicherbasierte und nachrichtenbasierte Interaktion
- Erweiterung der elementaren Laufzeitmodelle
  - insbesondere persistente Datenhaltung

Ausblick

- In der Freitagsvorlesung:
  - Überblick über den Entwurf von ULIX
- In der Übung:
  - Hardware, Interrupts und Konsorten
- Nächste Woche:
  - Implementierung von virtuellem Speicher