Systemprogrammierung

Grundlagen von Betriebssystemen

Teil B – VII.1 Betriebsarten: Stapelverarbeitung

Wolfgang Schröder-Preikschat

21. Juli 2022

Agenda

Einführung

Einprogrammbetrieb
- Manueller Rechnerbetrieb
- Automatisierter Rechnerbetrieb
- Schutzvorkehrungen
- Aufgabenverteilung

Mehrprogrammbetrieb
- Multiplexverfahren
- Schutzvorkehrungen
- Dynamisches Laden
- Simultanverarbeitung

Zusammenfassung

Gliederung

Lehrstoff

- Ziel ist es, „zwei Fliegen mit einer Klappe zu schlagen“, nämlich:
  i einen Einblick in **Betriebssystemgeschichte** zu geben und
  ii damit gleichfalls **Betriebsarten** von Rechensystemen zu erklären
- im Vordergrund stehen die Entwicklungsstufen im **Stapelbetrieb**
  - Adressraumschutz durch Abteilung, Eingrenzung und Segmentierung
  - Durchsatz-/Leistungssteigerung durch abgesetzten Betrieb, überlappte und abgesetzte Ein-/Ausgabe und Simultanbetrieb
  - Speicherminimierung durch Programmzerlegung und Überlagerungen

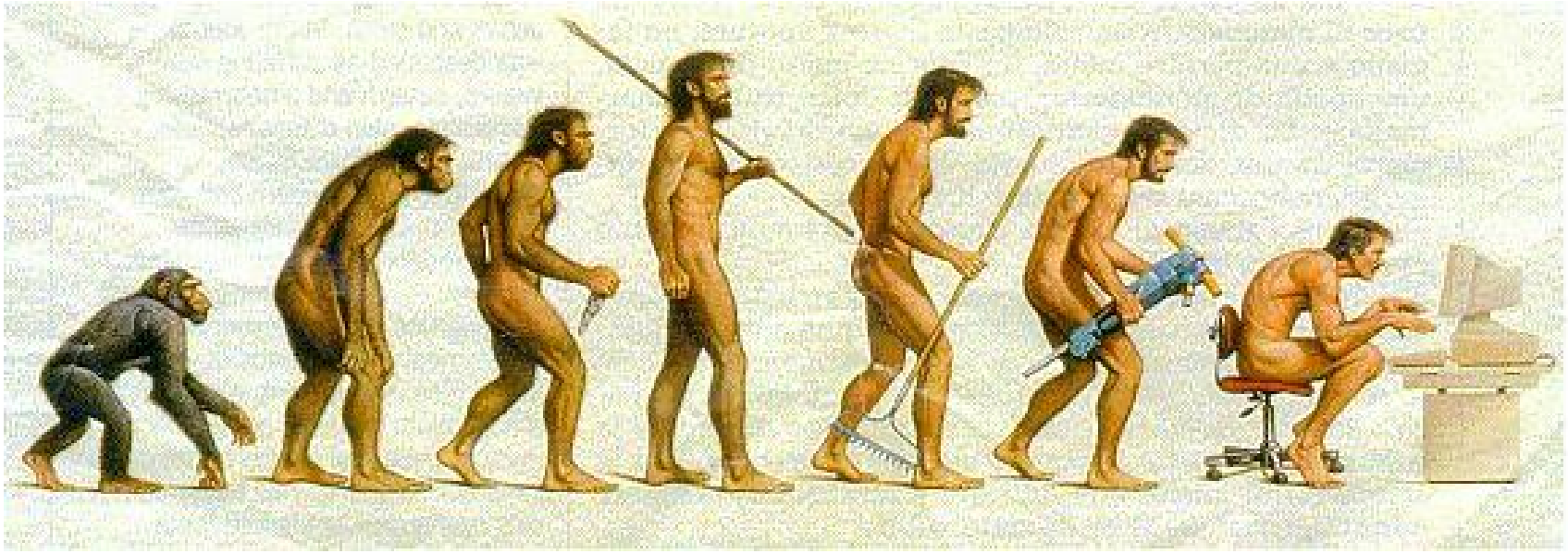
Hinweis

*Viele dieser Techniken — wenn nicht sogar alle — sind auch heute noch in einem **Universalbetriebssystem** auffindbar.*

- kennzeichnend ist, die Programme **interaktionslos** auszuführen
  - hierzu ist eine vollständige Auftragsbeschreibung erforderlich
  - die in einer speziellen „Skriptsprache“ ausformuliert wird
  - um das fertige „Skript“ von einem Interpretierer verarbeiten zu lassen
  - der anfangs als Monitor und später als Betriebssystem in Erscheinung tritt

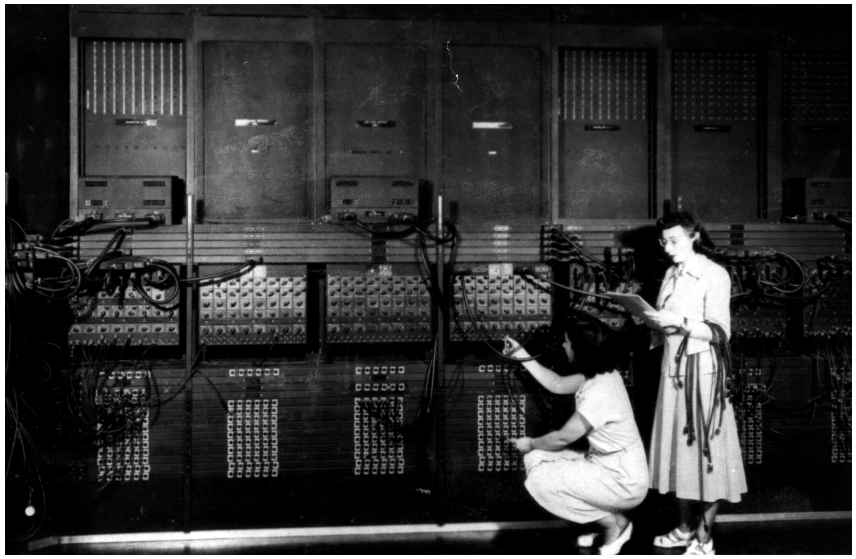
Betriebssystemevolution

Wenn wir nicht absichtlich unsere Augen verschließen, so können wir nach unseren jetzigen Kenntnissen annähernd unsere Abstammung erkennen, und dürfen uns derselben nicht schämen. (Charles Darwin)



Am Anfang war das Feuer. . .

- ENIAC (*electronic numerical integrator and computer*), 1945 [18]

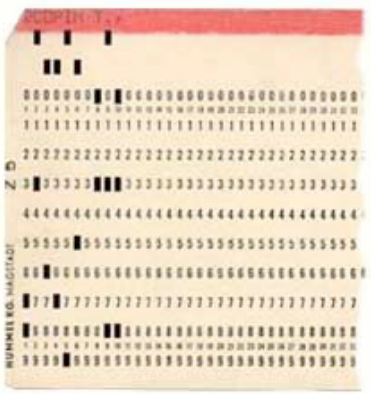


A small amount of time is required in preparing the ENIAC for a problem by such steps as setting program switches, putting numbers into the function table memory by setting its switches, and establishing connections between units of the ENIAC for the communication of programming and numerical information. (Pressemitteilung, DoD, 1946)

- elektronischer Allzweckrechner von 30 Tonnen Gewicht
- $15\,m \times 3\,m \times 5\,m$ große Zentraleinheit, $30\,m$ lange Frontplatte
- für seinen Betrieb waren 18 000 Röhren zuständig
- die elektrische Anschlussleistung belief sich auf etwa 174 000 Watt

Gliederung

Lochkartenverarbeitung



Hermann Holerith (1860–1929), Begründer der maschinellen Datenverarbeitung.

IBM ließ sich 1928 das Format patentieren: 80 Spalten, 12 Zeilen und rechteckige Löcher an den Schnittpunkten.

- **Ziffernlochkarte** (*numeric punch card*)
  - Datenaufzeichnung durch Lochung (Loch $\mapsto$ 1, kein Loch $\mapsto$ 0)
    - Dezimalzahlen werden mit einer Lochung dargestellt
    - Buchstaben und Sonderzeichen mit zwei oder drei
    - negative Vorzeichen ggf. durch eine Lochung in Zeile 11
  - manuell ca. 15 000 Zeichen/h, maschinell 4 000–10 000 Karten/h

Offene Stelle, an der die Substanz nicht mehr vorhanden ist — aber Information!

Alternativ kamen auch **Lochstreifen** (*punched [paper] tape*) zum Einsatz, mit ähnlichen Leistungsmerkmalen.

Manueller Betrieb des Rechners

- vollständige Kontrolle beim Programmier- oder Bedienpersonal[1]
  1. Programme mit **Lochkartenstanzer** (*card puncher*) ablochen
  2. Übersetzerkarten in den **Lochkartenleser** (*card reader*) geben
  3. Lochkartenleser durch Knopfdruck starten
  4. Programmkarten (aus Pt. 1 zur Übersetzung) in den Kartenleser einlegen
  5. Eingabekarten in den Kartenleser einlegen
  6. Leere Lochkarten für die Ausgabe in den Kartenstanzer einlegen
  7. Ausgabekarten in den Kartenleser des Druckers einlegen
  8. Ergebnisse der Programmausführung vom Drucker abholen
- **Problem**:
  - Urlader (z.B. Lochkartenleseprogramm) in den Rechner einspeisen

Programmieren bedeutet nicht Ausprobieren!

Wenigstens einen großen Vorteil hätte diese Betriebsart auch heute noch: man überlegt sich Programme sorgfältig und interpretiert (d.h., deutet) sie selbst, bevor ihre Eingabe in den Rechner erfolgt.

[1] „*Operator*“: ein Berufsbild, dass es in der Form heute nicht mehr gibt.

Lochkartenleseprogramm

- Rechnersystem durch **Ureingabe** (*bootstrap*) laden
  1. Bitmuster einer Speicherwortadresse über Schalter einstellen
     - die Adresse der nächsten zu beschreibenden Stelle im Hauptspeicher
  2. den eingestellten Adresswert in den PC der CPU laden
  3. Bitmuster für den Speicherwortinhalt über Schalter einstellen
     - ein Befehl, ein Direktwert oder eine Operandenadresse des Programms
  4. den eingestellten Datenwert an die adressierte Stelle laden
- **Problem**:
  - Bedienung, Mensch, Permanenz

Programmiertes Urladen als Firmware

Heute liegen Urlader nicht-flüchtig im Speicher (ROM/EPROM bzw. *flash*). Sie starten automatisch, wenn der Rechner angeschaltet und hochgefahren oder, während des Betriebs, manuell oder automatisch zum Neustart (*restart*) bzw. Zurücksetzen (*reset*) gezwungen wird.

Automatisierter Betrieb des Rechners

- **Dienstprogramme** sind abrufbereit im Rechnersystem gespeichert:
  - Systembibliothek, „Datenbank“ (Dateiverwaltung)
- **Anwendungsprogramme** fordern Dienstprogramme explizit an:
  - spezielle **Steuerkarten** sind dem Lochkartenstapel hinzugefügt
    - Systemkommandos, die auf eigenen Lochkarten kodiert sind
    - Anforderungen betreffen auch Betriebsmittel (z.B. Speicher, Drucker)
  - der erweiterte Lochkartenstapel bildet einen **Auftrag** (*job*)
    - an einen **Kommandointerpretierer** (*command interpreter*)
    - formuliert als **Auftragssteuersprache** (*job control language*, JCL)
  - die Programmausführung erfolgt ohne weitere Interaktion
- **Problem**:
  - vollständige Auftragsbeschreibung (inkl. Betriebsmittelbedarf)

Bestimmte Tätigkeiten sehr sicher und schnell durchführen

Eignet sich (nach wie vor) zur Bewältigung von „Routineaufgaben“.

Auftragssteuersprache

- FORTRAN Monitoring System, FMS [9], um 1957
  - wird zuweilen auch als „erstes Betriebssystem“ bezeichnet

```
                *JOB, 42, ARTHUR DENT
                *XEQ
                *FORTRAN
Programmkarten  {
                *DATA
Datenkarten     {
                *END
```

Residentes Steuerprogramm

resident monitor

- **hauptspeicherresidente Systemsoftware**, zur Auftragssteuerung:
  - Kommandointerpreter, Lochkartenleseprogramm, E/A-Prozeduren
  - Systemprogramme, die ein „embryonales Betriebssystem“ bilden
- **Solitär**: einzelstehende und getrennt übersetzte Programmeinheit, die verschiedenartig vom Anwendungsprogramm entkoppelt ist
  - Einsprungtabelle (*jump table*)
  - partielle Interpretation von Monitoraufrufen
  - getrennte/partitionierte reale Adressräume (Schutzgatter)
  - Arbeitsmodi (System-/Benutzermodus, privilegiert/unprivilegiert)
- **Problem**:
  - Arbeitsgeschwindigkeit der Peripherie, sequentielle Ein-/Ausgabe

Systemsoftware als Firmware

Heute ist diese Funktionseinheit noch in dem „*basic input/output system*“ (BIOS) präsent, mit dem nahezu jeder Rechner ausgestattet ist und das nach wie vor grundlegende Aufgaben, die nicht nur zum Urladezeitpunkt anfallen, übernimmt.

Adressraumschutz durch Abgrenzung

fencing

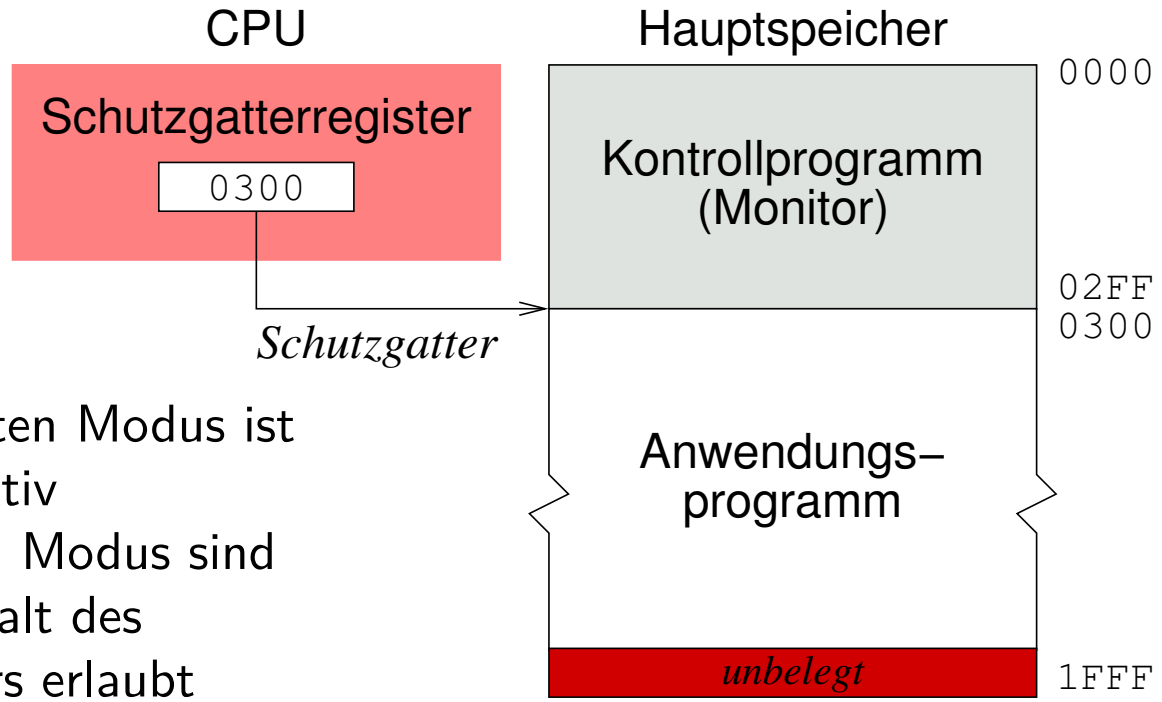
- ein **Schutzgatter** (*fence*) trennt den vom residenten Steuerprogramm und vom transienten Programm belegten Hauptspeicherbereich
  - Anwendungsprogramme werden komplett, d.h. statisch gebunden
    - das Schutzgatter entspricht einer unveränderlichen, realen Speicheradresse
    - Verlagerungskonstante beim Binden, Ladeadresse für die Programme
  - ggf. ist gewisse Flexibilität durch ein **Schutzgatterregister** gegeben
    - veränderliche, reale Speicheradresse; programmierbarer Wert
    - Verlagerungsvariable beim Binden, (zur Basis 0) relative Programmadressen
    - ein **verschiebender Lader** platziert das Programm im Hauptspeicher
- innerhalb der (Anwendungs-) **Partition**, die am Schutzgatter startet oder endet, ist dynamischer Speicher begrenzt zuteilbar
  - d.h., in der oberen oder unteren Hälfte des Hauptspeichers
  - der Monitor selbst betreibt jedoch nur **statische Speicherverwaltung**
- **Problem**:
  - übergroße und statisch gebundene (vollständige) Anwendungsprogramme

Isolation permanent benötigter Programme

Geschützt wird der Monitor, nicht jedoch das Anwendungsprogramm.

Schutzgatterregister I

fence register

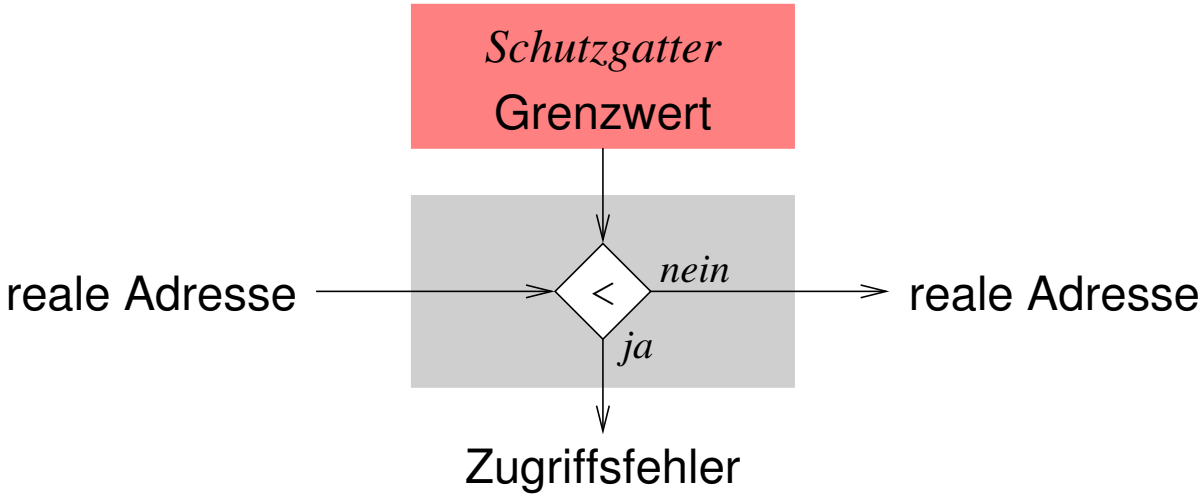


- **Arbeitsmodi**
  - nur im unprivilegierten Modus ist das Schutzgatter aktiv
  - nur im privilegierten Modus sind Änderungen am Inhalt des Schutzgatterregisters erlaubt

Physisch voneinenander unabhängige Bereiche

Es erfolgt die **Partitionierung** des realen Adressraums. Läuft jedoch die CPU im privilegierten Modus, umfasst der Monditoradressraum den Adressraum des Anwendungsprogramms. Ein ähnliches Modell des virtuellen Adressraums verfolgen heute Linux, macOS und Windows.

Schutzgatterregister II



- ein **Zugriffsfehler** führt zum Abbruch der Programmausführung
  - Schutzfehler (*segmentation fault*), *Trap*
- **Problem**:
  - „interne Fragmentierung“: Zugriff auf unbelegten Bereich (*false positive*)

Etwas fälschlich als gültigen Speicherzugriff akzeptieren müssen

Dieses Zugriffsproblem ist Merkmal einer jeden seitennummerierten Technik in Bezug auf den vom Programm **unbelegten Bereich einer Seite**, der jedoch im Adressraum gültig ist. ⇝ **interne Fragmentierung**

Abgesetzter Betrieb I

- Berechnung erfolgt getrennt von Ein-/Ausgabe (*off-line*)
  - **Satellitenrechner** zur Ein-/Ausgabe mit „langsamer Peripherie“
    - Kartenleser, Kartenstanzer, Drucker
    - Ein-/Ausgabedaten werden über **Magnetbänder** transferiert
  - **Hauptrechner** zur Berechnung mit „schneller Peripherie“
    - Be-/Entsorgung des Hauptspeichers auf Basis von **Bandmaschinen**
    - dadurch erheblich verkürzte Wartezeiten bei der Ein-/Ausgabe
- **Problem**:
  - sequentieller Bandzugriff, feste Auftragsreihenfolge

Altbewährte Technik von hoher Aktualität

Heute finden zusätzlich **Wechselplatten** Verwendung. Auch die Ausführung von ausgewählten Systemfunktionen auf eigens dafür bereitgestellte Recheneinheiten auszulagern, hat durch **Multiprozessoren** und insb. auch **mehrkernige Prozessoren** eine Erneuerung erfahren [2, 19].

Abgesetzter Betrieb II

- dedizierte Rechner/Geräte für verschiedene Arbeitsphasen:

Phase	Medium			Rechner
Eingabe	Text/Daten Lochkarten	⇒ ⇒	Lochkarten Magnetband	Satellitenrechner
↓				↓
Verarbeitung	Magnetband Hauptspeicher	⇒ ⇒	Hauptspeicher Magnetband	Hauptrechner
↓				↓
Ausgabe	Magnetband Daten	⇒ ⇒	Lochkarten Drucker	Satellitenrechner

- **Problem**:
  - programmierte Ein-/Ausgabe: bei Satelliten-und Hauptrechner

Programmierte E/A

Ein-/Ausgabe, die ausschließlich durch **prozessorseitige Aktionen** vorangetrieben wird und damit eine CPU voll in Anspruch nimmt.

Überlappte Ein-/Ausgabe

- durch **asynchrone Programmunterbrechungen** [8, S. 225–227] die technische Voraussetzung für **gleichzeitige Pozesse** schaffen [1, 17]
  - die E/A-Geräte fordern der CPU weitere E/A-Aufträge ab, plötzlich
  - E/A und Berechnung desselben Programms überlappen sich
- **Speicherdirektzugriff** (*direct memory access*, DMA, [10]) ersetzt die bislang gebräuchliche **programmierte Ein-/Ausgabe**
  - **E/A-Kanäle**, die unabhängig von der CPU arbeiten
    - d.h., Zugriffskanäle zwischen einem E/A-Gerät und dem Hauptspeicher
  - Datentransfer erfolgt durch ein **Kanalprogramm** (*cycle stealing*, [7, 5])
- ermöglicht **nebenläufige Ausführung** von Gerätetreiberprogrammen und dem (einen) Hauptprogramm
  - Kooperation und Konkurrenz gleichzeitiger Prozesse sind zu koordinieren
  - d.h., Erfordernis zur **Synchronisation** [4, S. 28–33]
- **Problem**:
  - Leerlauf beim Auftragswechsel

Überlappte Auftragsverarbeitung *single-stream batch monitor*

- Grundlage ist ein als **Verarbeitungsstrom** ausgelegter Stapel (*batch*) sequentiell auszuführender Programme/Aufträge
  - während Ausführung eines Auftrags wird der nachfolgende Auftrag bereits in den Hauptspeicher eingelesen (Zwischenpuffern)
  - Programme werden im **Vorgriffsverfahren** (*prefetching*) abgearbeitet
- **Auftragseinplanung** (*job scheduling*) geschieht im Hintergrund, aber mitlaufend (*on-line*) zur gegenwärtigen Programmausführung
  - statische (*off-line*) Einplanung nach unterschiedlichsten Kriterien
    - Ankunftszeit, erwartete Laufzeit, erwarteter Betriebsmittelbedarf
  - die Aufträge werden dazu im Hintergrundspeicher zusammengestellt
    - *wahlfreier Zugriff* (z.B. Plattenspeicher [6]) beschleunigt die Vorgriffe
- **Problem**:
  - Hauptspeicher, Monopolisierung der CPU, Leerlauf bei Ein-/Ausgabe

Monopolisierung

Jedes Programm bekommt die CPU **exklusiv** zugewiesen und gibt sie erst bei vollendeter Ausführung **freiwillig** ab („*run to completion*“).

Abgesetzte Ein-/Ausgabe I

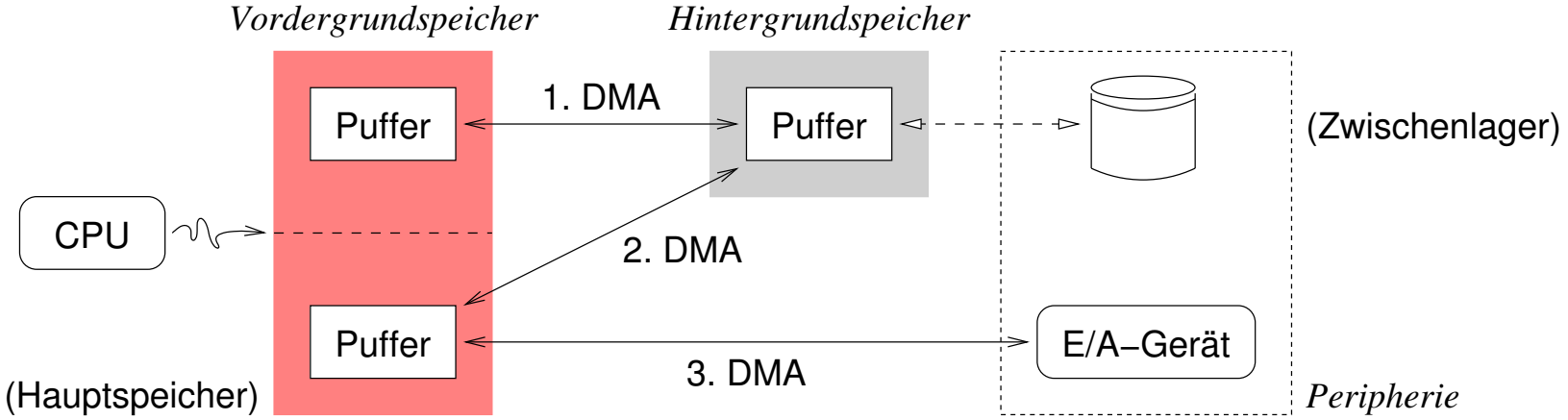
- Programmausführung ist eine periodische Abfolge von zwei Phasen:
  - CPU-Stoß
    - Aktionen mit ausschließlich Berechnungen („*CPU burst*“)
    - jede einzelne Aktion verläuft vergleichsweise schnell
  - E/A-Stoß
    - Aktionen mit ausschließlich Ein-/Ausgabe („*I/O burst*“)
    - jede einzelne Aktion verläuft vergleichsweise langsam
- CPU- und E/A-Stoß lassen sich durch **Puffer** zeitlich entkoppeln [14]
  - ein CPU-Stoß endet mit der Pufferung das Auftrags für einen E/A-Stoß, er setzt nur einen E/A-Auftrag ab und dann seine Ausführung fort
  - ein E/A-Stoß läuft im Hintergrund der Ausführung des CPU-Stoßes ab, indem ein gepufferter E/A-Auftrag zur Ausführung gebracht wird
- **Problem**:
  - Leerlauf im Wartezustand: konsumier-/wiederverwendbare Betriebsmittel

Spooling (*simultaneous peripheral operations online*)

Bewerkstelligt durch Systemprogramme einerseits zum Absetzen und andererseits zur Überwachung/Steuerung der Verarbeitung von Ein- oder Ausgabeaufträgen (in UNIX: `lpr(1)` bzw. `lpd(8)`).

Abgesetzte Ein-/Ausgabe II

- überlappte E/A befüllt/entleert Datenpuffer 1. und 2. DMA
  - Puffer im **Vordergrundspeicher** (*primary/main store*) und
  - Puffer im **Hintergrundspeicher** (*secondary/backing store*)
- **effektive E/A** in Bezug auf die Endgeräte operiert dann mit den im Hintergrundspeicher gepufferten Daten 2. und 3. DMA
  - beansprucht dazu jedoch die CPU und nutzt wiederum überlappte E/A



Abgesetzter Betrieb

Hier sind die im Hauptspeicher liegenden Puffer dem Haupt- (oben) und Satellitenrechner (unten) zugeordnet.

Gliederung

Maßnahmen zur Leistungssteigerung

- in Raum vermöge **Mehrprogrammbetrieb** (*multi-programming*) und Zeit mittels **Simultanverarbeitung** (*multiprocessing*)
  - Multiplexen der CPU zwischen mehreren Programmen[2]
    - **Nebenläufigkeit** (*concurrency*)
  - Abschottung der sich in Ausführung befindlichen Programme
    - **Adressraumschutz** (*address space protection*)
  - Überlagerung unabhängiger Programmteile
    - **Segmentierung** (*segmentation*, [15]) — im Sinne von „Aufteilung“
- beides auch Maßnahmen eines Betriebssystems zur Unterstützung der besseren Strukturierung von Programmen

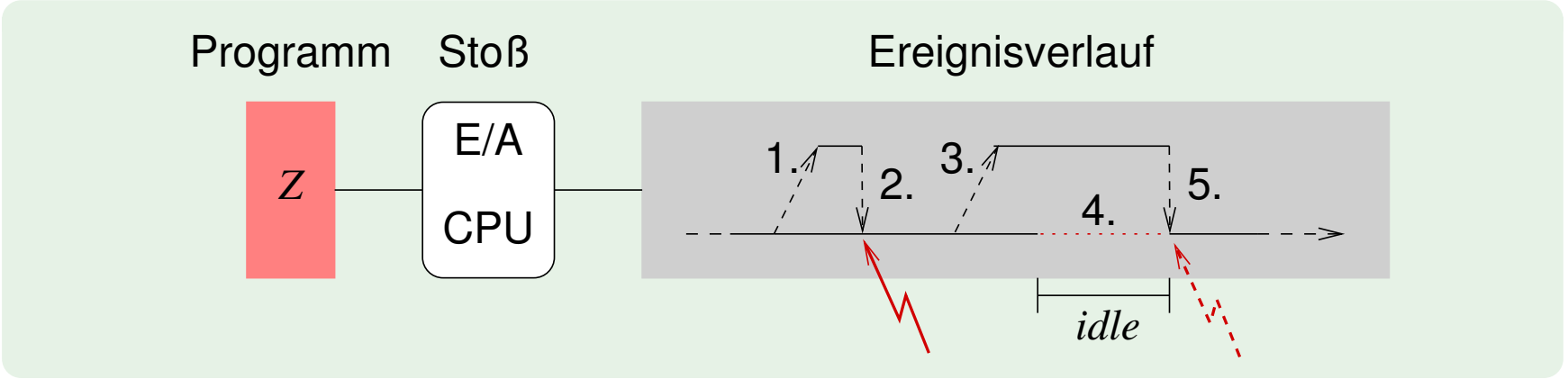
Nichtsequentielle (System-) Programmierung

> *If we believe in data structures, we must believe in independent (hence simultaneous) processing. For why else would we collect items within a structure? Why do we tolerate **languages** that give us the one without the other? (Alan Perlis [16, Epigram 68])*

[2] Zunächst als „befremdliches Ergebnis“ von *Interrupts* angesehen [12, S. 43].

Aktives Warten

- überlappte Ein-/Ausgabe bedeutet die parallele Ausführung von CPU- und E/A-Stößen — im **Einprogrammbetrieb** jedoch nicht immer



1. Programm Z löst einen zum CPU-Stoß nebenläufigen E/A-Stoß aus
2. die Beendigung des E/A-Stoßes wird durch eine Unterbrechung angezeigt
3. der Unterbrechungshandhaber[3] löst einen weiteren E/A-Stoß aus
4. Z muss auf E/A warten, dem logischen Verlauf nach endet der CPU-Stoß
5. die CPU ist untätig (*idle*) bis der E/A-Stoß endet, ein CPU-Stoß beginnt

- **Problem**:
  - Durchsatz, Auslastung (CPU, E/A-Geräte)

[3] *interrupt handler*

Multiplexen des Prozessors

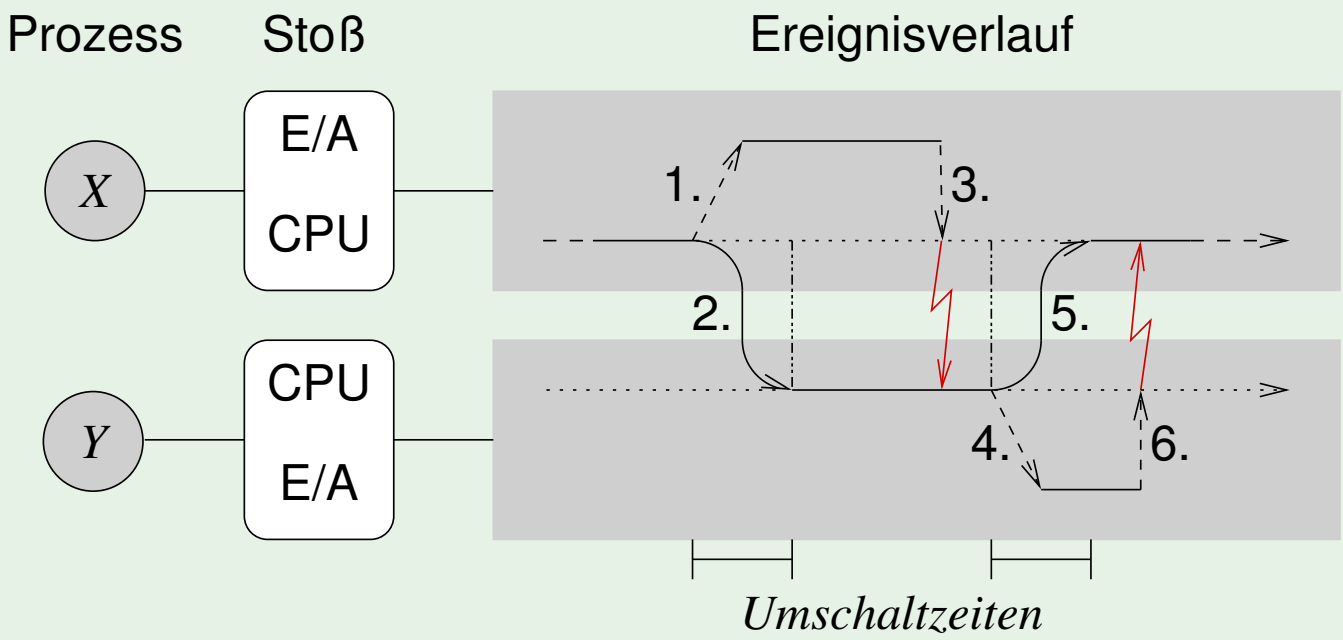
- die Auslastung der CPU steigt, wenn die Wartezeit eines Programms als Laufzeit eines anderen Programms nutzbar ist
  - aktives Warten (*busy waiting*) ist grundsätzlich unproduktiv und beschert der CPU nur kostbare **Leerlaufzeit** (*idle time*)
  - im Falle eines alternativen Ausführungsstrangs ist eine solche Wartephase allerdings als **Produktivzeit** nutzbar
  - dazu ist die CPU zur Aufnahme eines solchen Ausführungsstrangs (eines anderen Programms) umzuschalten $\rightsquigarrow$ „passives Warten“
- der **Umschaltmechanismus** selbst ist unabhängig von den konkret auszuführenden Programmen, er ist generisch auslegbar
  - dazu wird von dem „Programm in Ausführung“ abstrahiert, der **Prozess** eingeführt, dessen Inkarnation einen eigenen **Prozessorzustand** besitzt
  - Umschalten der CPU bedeutet dann lediglich, den Prozessorzustand eines anderen Prozesses zu aktivieren

Leerlaufzeit

Es gibt jedoch Fälle, wo aktives Warten auf einen Ereigniseintritt die effiziente Alternative darstellt: **Umschaltzeit** $>$ erwartete Wartezeit.

Passives Warten I

- Programmverarbeitung im **Mehrprozessbetrieb**:



- Prozesse X und Y als Produkte desselben nichtsequentiellen Programms oder verschiedener sequentieller Programme
- der CPU-Stoß von Prozess Y wird durch Beendigung des E/A-Stoßes von Prozess X unterbrochen und umgekehrt
- solche Unterbrechungsszenarien rufen unvorhersehbare **Interferenz** hevor, die für zeitabhängige Prozesse kritisch sein kann

Passives Warten II

. . . zum Programmablauf auf S. 27

- ein Prozess, der warten muss, gibt die CPU zugunsten eines anderen Prozesses ab, der nicht warten muss: **Prozessormultiplexverfahren**
  1. Prozess X löst einen E/A-Stoß und beendet seinen CPU-Stoß
  2. Prozess Y wird eingelastet und beginnt seinen CPU-Stoß
  3. Beendigung des E/A-Stoßes von Prozess X unterbricht Prozess Y
     - die Unterbrechungsbehandlung überlappt sich mit Prozess Y
  4. Prozess Y löst einen E/A-Stoß und beendet seinen CPU-Stoß
  5. Prozess X wird eingelastet und beginnt seinen CPU-Stoß
  6. Beendigung des E/A-Stoßes von Prozess Y unterbricht Prozess X
     - die Unterbrechungsbehandlung überlappt sich mit Prozess X
- **Einlastung** (*dispatching*) eines Prozesses ist der Moment, in dem die CPU umgeschaltet wird
  - den Prozessorzustand des die CPU abgebenden Prozesses sichern und des die CPU erhaltenden Prozesses (wieder) herstellen
  - das Betriebssystem vollzieht genau damit den **Prozesswechsel**
- **Problem**:
  - Interferenz, Adressraumschutz, Koordination

Adressraumschutz

- bei mehr als einem Programm, das neben dem residenten Monitor im Hauptspeicher liegen muss, reichen Schutzgatter nicht
  - jedes einzelne Programm ist von anderen Programmen zu isolieren
- der **Bindungszeitpunkt**, zu dem Namen mit Speicherorten verknüpft werden, begründet verschiedene Schutztechniken:

  vor Laufzeit $\rightsquigarrow$ Schutz durch **Einfriedung**
  - Programme laufen (weiterhin) im realen Adressraum
  - ein **verschiebender Lader** besorgt die Bindung zur *Ladezeit*
  - die CPU (bzw. MPU) überprüft die realen Adressen zur Ausführungszeit

  zur Laufzeit $\rightsquigarrow$ Schutz durch **Segmentierung** [3]
  - jedes Programm läuft in seinem eigenen **logischen Adressraum**
  - der Lader bestimmt die Bindungsparameter (reale Basisadresse)
  - die CPU (bzw. MMU) besorgt die Bindung zur *Ausführungszeit*
    - dynamische Programmumlagerung (*program relocation*, [13])

Positionsrelative Ausrichtung des Adressraums

In beiden Fällen ist der Namensraum eines Programms relativ zu der logischen Anfangsadresse 0 ausgerichtet.

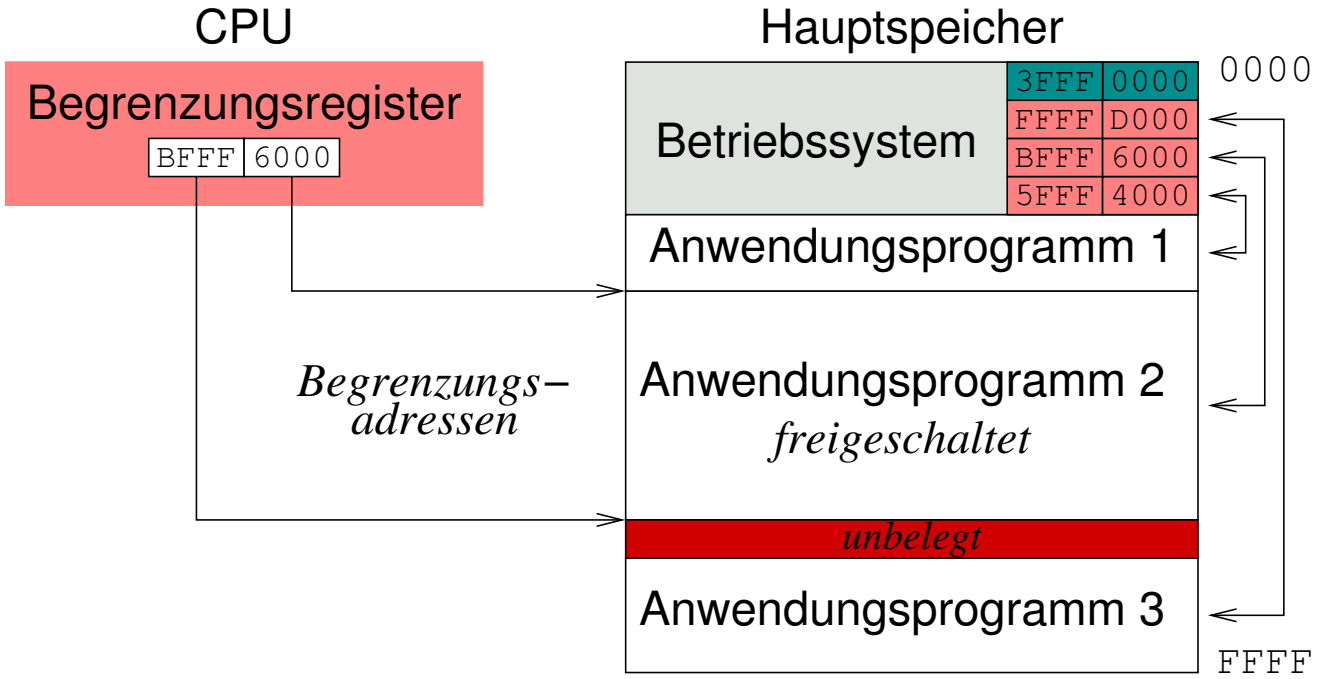
Adressraumschutz durch Einfriedung

boundary fence

- **Begrenzungsregister** (*bounds register*) legen die Unter-/Obergrenze eines Programms im realen Adressraum fest
  - jedem Programm wird ein solches Wertepaar spätestens zur Ladezeit fest zugeordnet und seiner Prozessinkarnation zur „Einzäunung“ mitgegeben
    - die *Softwareprototypen* dieser Register der **Adressüberprüfungshardware**
    - verwaltet als Attribute des Prozesskontrollblocks
  - bei Prozesseinlastung werden die *Hardwareprototypen* dieser Register mit den in den Softwareprototypen vermerkten Werten definiert
    - wodurch der Hauptspeicherbereich des Prozesses „freigeschaltet“ wird
    - zu einem Zeitpunkt ist damit immer nur ein solcher Bereich freigegeben
- innerhalb des durch die Begrenzungsregister festgelegten Bereichs ist dynamischer Speicher bedingt zuteilbar
  - der Bedarf dafür muss jedoch spätestens zum Ladezeitpunkt bekannt sein
  - das Betriebssystem kann nur **statische Speicherverwaltung** betreiben
- **Problem**:
  - Fragmentierung, Verdichtung

Begrenzungsregister I

bounds register

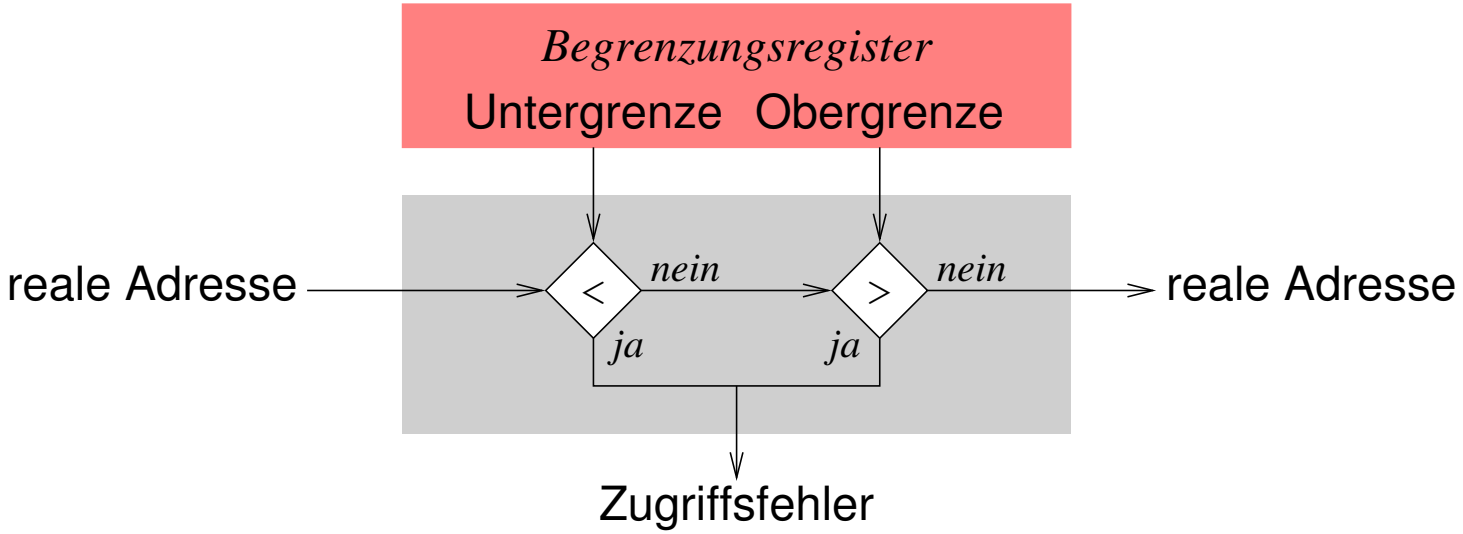


- konsequente Umsetzung des Modells ist es, Anwendungsprogramme auch vor direkten Zugriffen durch das Betriebssystem zu schützen
  - in dem Fall verfügt jeder Arbeitsmodus über eigene Begrenzungsregister
  - ein **Arbeitsmoduswechsel** aktiviert das jeweilige Registerpaar implizit

Hinweis

Heute in Form einer „memory protection unit“ (MPU) gebräuchlich.

Begrenzungsregister II



- ein **Zugriffsfehler** führt zum Abbruch der Programmausführung
  - Schutzfehler (*segmentation fault*), *Trap*
- **Problem** wie beim Schutzgatterkonzept (vgl. S. 16), zusätzlich:
  - „externe Fragmentierung“, d.h., unbelegte Hauptspeicherbereiche

Externe Fragmentierung

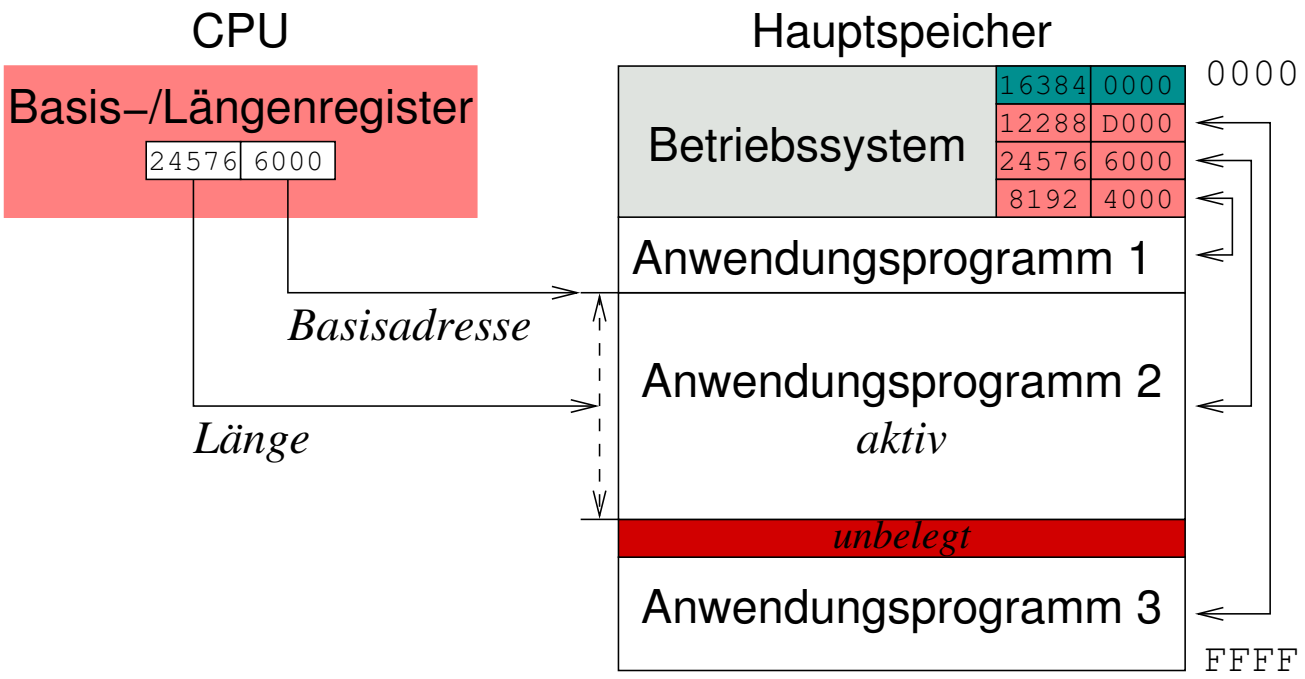
Freie Speicherbereiche sind jeder für sich zu klein, insgesamt jedoch groß genug für die Speicheranforderung eines Prozesses. Sie bleiben dennoch nutzlos, da sie im realen Adressraum nicht linear zusammenhängend angeordnet sind.

Adressraumschutz durch Segmentierung

- **Basis-/Längenregister** (*base/limit register*) legen Segmentort sowie -größe eines Programms im realen Adressraum fest
  - jedem Programm wird ein solches Wertepaar zur Ladezeit zugeordnet und seiner Prozessinkarnation mitgegeben
    - die *Softwareprototpypen* dieser Register der **Adressverschiebungshardware**
    - verwaltet als Attribute des Prozesskontrollblocks
  - bei Prozesseinlastung werden die *Hardwareprototypen* dieser Register mit den in den Softwareprototypen vermerkten Werten definiert
    - wodurch der Hauptspeicherbereich des Prozesses „aktiviert“ wird
    - zu einem Zeitpunkt ist damit immer nur ein solches Segment freigegeben
- innerhalb des durch die Basis-/Längenregister definierten Bereichs ist dynamischer Speicher bedingt zuteilbar
  - der Bedarf dafür braucht erst zur Ausführungszeitpunkt bekannt zu sein
    - ist die **maximale Größe** nicht erreicht, kann das Segment expandiert werden
    - verhindert dies ein angrenzendes Segment, wird das Programm umgelagert
    - ggf. wird der Hauptspeicher für einen passenden Bereich verdichtet
  - das Betriebssystem kann **dynamische Speicherverwaltung** betreiben
- **Problem**:
  - externe Fragmentierung, Hauptspeichergröße

Basis-/Längenregister I

base/limit register

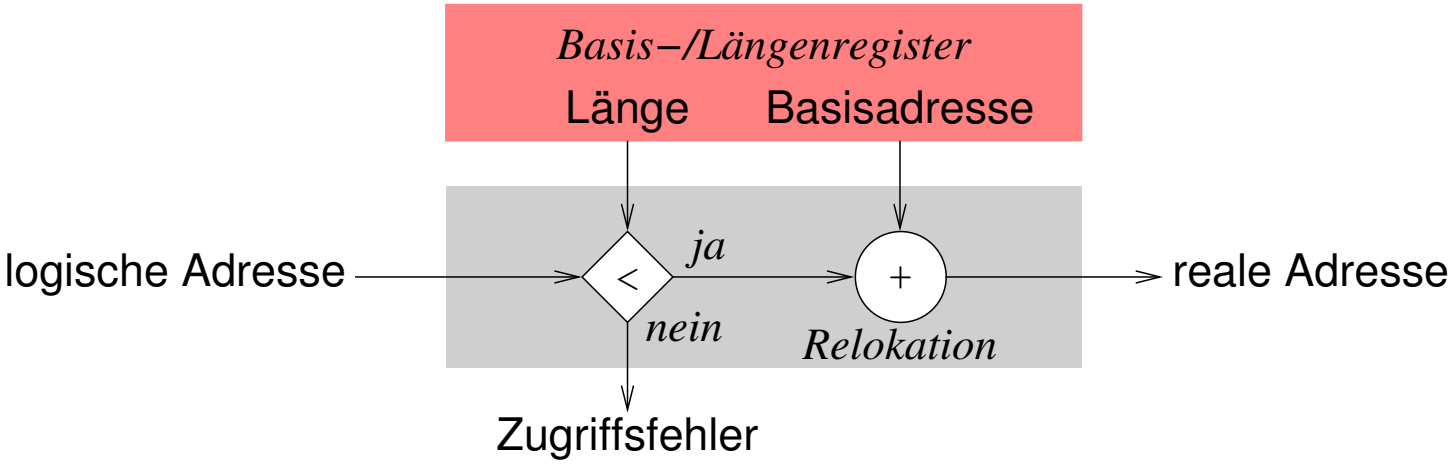


- das Betriebssystem ist in einem eigenen Segment gekapselt, durch Basis-/Längenregister für den privilegierten Arbeitsmodus

Segmentierende MMU

Kennzeichnend für eine „*memory management unit*“ (MMU), allerdings wie hier gezeigt nur zur eher unüblichen Isolation eines ganzen ausführbaren Programms durch ein einzelnes Segment.

Basis-/Längenregister II



- ein **Zugriffsfehler** führt zum Abbruch der Programmausführung
  - Schutzfehler (*segmentation fault*), *Trap*
- **Problem**:
  - Unisegmentansatz, d.h., pro Programm nur ein Segment

Verlagerung

Die MMU verschiebt die bei Programmausführung erzeugte effektive logische Adresse um die Segmentbasisadresse (Verlagerungskonstante/-variable). Der Basisregisterinhalt ist zwischen den einzelnen Ausführungsphasen veränderlich, das Programm kann somit zur Laufzeit im Hauptspeicher umgelagert werden.

Überlagerungstechnik I

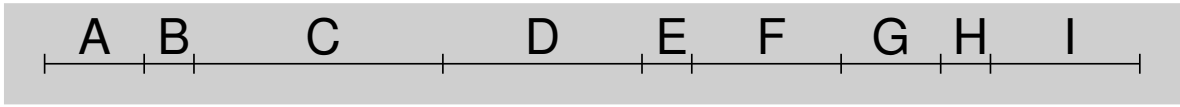
overlay [15]

- ein einzelnes Programm ausführen zu können, obwohl es in seiner Gesamtheit nicht in den Hauptspeicher passt:
  - i es ist überhaupt zu groß für den Hauptspeicher, auch falls es als einziges Anwendungsprogramm zur Ausführung kommen soll, oder
  - ii es ist größer als die einem Anwendungsprogramm statisch zur Verfügung stehende **Hauptspeicherpartition**
- dazu wird das Programm in hinreichend kleine Teile zergliedert, die nicht ständig im Hauptspeicher vorhanden sein müssen
  - dies betrifft sowohl den Text- als auch den Datenbereich
  - nicht benötigte Teile liegen abrufbereit im Hintergrundspeicher
  - sie werden bei Bedarf nachgeladen, überlagern nicht mehr benötigte Teile
- das Nachladen ist programmiert, d.h., in dem Programm festgelegt, die Entscheidung dazu fällt zur Programmlaufzeit
  - ein Programm löst **dynamisches Laden** von Überlagerungen aus
- **Problem**:
  - „optimale“ Überlagerungsstruktur, manueller Ansatz

Überlagerungstechnik II

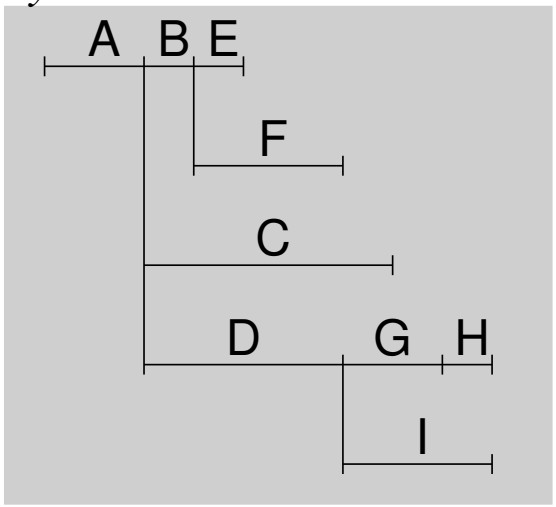
- Hauptspeicherbedarf für ein unzerteiltes Programm:

statisch



- Hauptspeicherbedarf für ein zerteiltes Programm:

dynamisch



eingelagert

ABE

ABF

AC

ADGH

ADI

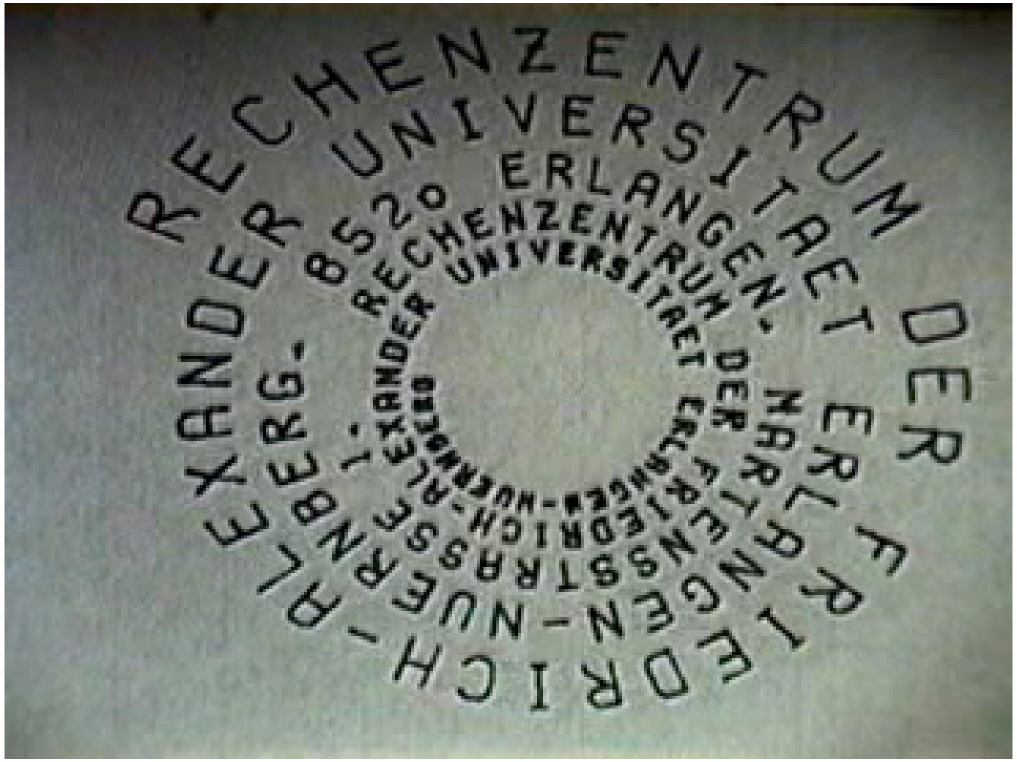
  - nicht alle Bestandteile (A – I) eines Programms müssen gleichzeitig im Hauptspeicher vorliegen, damit das Programm auch ausführbar ist
  - wann welches Programmteil zur Ausführung gelangt, legt der dynamische Ablauf des Programms selbst fest

Multistapelbetrieb

- **Simultanverarbeitung** (*multiprocessing*) mehrerer Auftragsströme
  - das Betriebssytem als „*multi-stream batch monitor*“
- echt/quasi parallele Verarbeitung von Auftragsströmen sequentieller Programme, gleichzeitig, im **Multiplexverfahren**
  - sequentielle Ausführung der Programme desselben Auftragstapels
    - „wer [innerhalb des Stapels] zuerst kommt, mahlt zuerst“...
  - nichtsequentielle Ausführung der Programme verschiedener Auftragstapel
- die Stapelwechsel kommen **kooperativ** (*cooperative*), **verdrängend** (*preemptive*) oder als Kombination von beiden zustande
  - kooperativ bei (programmierter) Ein-/Ausgabe, also bei Beendigung des CPU-Stoßes eines Prozesses, und am Programmende
  - verdrängend bei Ablauf einer Frist (*time limit*), innerhalb welcher die Ausführung eines jeweiligen Programms abgeschlossen sein muss
- **Problem**:
  - interaktionsloser Betrieb, Mensch/Maschine-Schnittstelle

Gliederung

Rechnerbetrieb am RRZE um 1973 [4]



[4] https://www.video.uni-erlangen.de/clip/id/4251.html

Resümee

. . . immer nach dem gleichen Schema zu verrichtende Arbeit

- **Stapelbetrieb** meint die sequentielle Abwicklung von Aufgaben
  - anfangs manuelle Bestückung des Rechners durch Programmierpersonal
  - später automatisierte Bestückung mittels **Kommandointerpreter**
    - Programmierer organisier(t)en ihren Auftragsstapel mittels Steuerkarten
    - Operateure übern{a,e}hmen die manuelle Bestückung des Rechners
  - **Aufgabenverteilung** dehnt(e) sich auch auf systeminterne Abläufe aus
    - abgesetzter Betrieb, überlappte Ein-/Ausgabe
    - überlappte Auftragsverarbeitung, abgesetzte Ein-/Ausgabe
- **Mehrprogrammbetrieb** hält mehrere Programme im Hauptspeicher
  - Hauptspeicherknappheit wird durch dynamisches Laden begegnet
  - der Prozessor wird im Multiplexverfahren betrieben: **Prozesse**
  - Schutzvorkehrungen schotten Prozessadressräume voneinander ab
  - um das Rechensystem damit sicher in **Simultanbetrieb** fahren zu können
- nach wie vor bestens für **Routinearbeit** geeignet, wobei:
  - nur Operateure interaktiven Zugang zum Rechensystem bek{a,om}men
  - die Anwender mit dem Rechensystem jedoch interaktionslos arbeit(et)en

Literaturverzeichnis I

[1] Bell, C. G. ; Newell, A. :
Computer Structures: Readings and Examples.
New York, NY, USA : McGraw-Hill Inc., 1971. –
668 S.

[2] Brüning, U. ; Giloi, W. K. ; Schröder-Preikschat, W. :
Latency Hiding in Message-Passing Architectures.
In: Siegel, H. J. (Hrsg.): *Proceedings of the 8th International Symposium on Parallel Processing, April 26–29, 1994, Cancún, Mexico*.
Washington, DC, USA : IEEE Computer Society, 1994. –
ISBN 0–8186–5602–6, S. 704–709

[3] Dennis, J. B.:
Segmentation and the Design of Multiprogrammed Computer Systems.
In: *Journal of the ACM* 12 (1965), Okt., Nr. 4, S. 589–602

[4] Dijkstra, E. W.:
Communication with an Automatic Computer, Universiteit van Amsterdam, Diss., Okt. 1959

[5] IBM:
Data Processing Machine Including Program Interrupt Feature.
U.S. Pat. No. 3,319,230 (1967), 1956

Literaturverzeichnis II

[6] IBM:
IBM 503 RAMAC.
http://www-03.ibm.com/ibm/history/exhibits/storage/storage_PH0305.html, 1956

[7] IBM:
709 Data Processing System.
http://www-03.ibm.com/ibm/history/exhibits/mainframe/mainframe_PP709.html, Jan. 1957

[8] Knuth, D. E.:
The Art of Computer Programming. Bd. 1: *Fundamental Algorithms*.
Reading, MA, USA : Addison-Wesley, 1973

[9] Larner, R. A.:
FMS: The FORTRAN Monitor System.
In: Butler, M. K. (Hrsg.) ; Brown, J. M. (Hrsg.) ; American Federation of Information Processing Societies, Inc. (Veranst.): *1987 Proceedings of the National Computer Conference, June 15–18, 1987, Chicago, Illinois, USA* Bd. 56 American Federation of Information Processing Societies, Inc., AFIPS Press, 1987, S. 815–820

Literaturverzeichnis III

[10] Leiner, A. L.:
System Specifications for the DYSEAC.
In: *Journal of the ACM* 1 (1954), Apr., Nr. 2, S. 57–81

[11] Leiner, A. L. ; Alexander, S. N.:
System Organization of DYSEAC.
In: *IRE Transactions on Electronic Computers* EC-3 (1954), März, Nr. 1, S. 1–10

[12] Loopstra, B. J.:
The X-1 Computer.
In: *The Computer Journal* 2 (1959), Nr. 1, S. 39–43

[13] McGee, W. C.:
On Dynamic Program Relocation.
In: *IBM Systems Journal* 4 (1965), Sept., Nr. 3, S. 184–199

[14] Mock, O. ; Swift, C. J.:
The Share 709 System: Programmed Input-Output Buffering.
In: *Journal of the ACM* 6 (1959), Apr., Nr. 2, S. 145–151

[15] Pankhurst, R. J.:
Operating Systems: Program Overlay Techniques.
In: *Communications of the ACM* 11 (1968), Febr., Nr. 2, S. 119–125

Literaturverzeichnis IV

[16] Perlis, A. J.:
Epigrams on Programming.
In: *SIGPLAN Notices* 17 (1982), Nr. 9, S. 7–13

[17] Smotherman, M. :
Interrupts.
http://www.cs.clemson.edu/~mark/interrupts.html, Jul. 2008

[18] US War Department, Bureau of Public Relations:
For Radio Broadcast: Physical Aspect, Operation of ENIAC are Described.
http://americanhistory.si.edu/collections/comphist/pr4.pdf, Febr. 1946

[19] Wentzlaff, D. ; Agarwal, A. :
Factored Operating Systems (fos): The Case for a Scalable Operating System for Multicores.
In: *ACM SIGOPS Operating Systems Review* 43 (2009), Apr., Nr. 2, S. 76–85

Generationen von Hardware und Betriebssoftware

- zeitliche Einordnung von Technologiemerkmalen
  - „unscharf“, Übergänge fließend

Generation	Epoche	Hardware	Rechnerbetriebsart
1	1945	Röhre	Stapelbetrieb
2	1955	Halbleiter, Gatter	Echtzeitbetrieb
3	1965	MSI/LSI, WAN	Mehrprogrammbetrieb
4	1975	VLSI, LAN	Mehrzugangsbetrieb
5	1985	ULSI, RISC	Massenparallelbetrieb
?	1995	WLAN, RFID, SoC	?
?	?	?	Quantenrechenbetrieb

Legende: MSI, LSI, VLSI, ULSI — *medium*, *large*, *very large*, *ultra large scale integration*
LAN — *local area network* (Ethernet)
WAN — *wide area network* (Arpanet)
RISC — *reduced instruction set computer*
WLAN — *wireless* LAN
RFID — *radio frequency identification* (Funkerkennung)
SoC — *system on chip*

Überlagerungsstruktur eines Programms

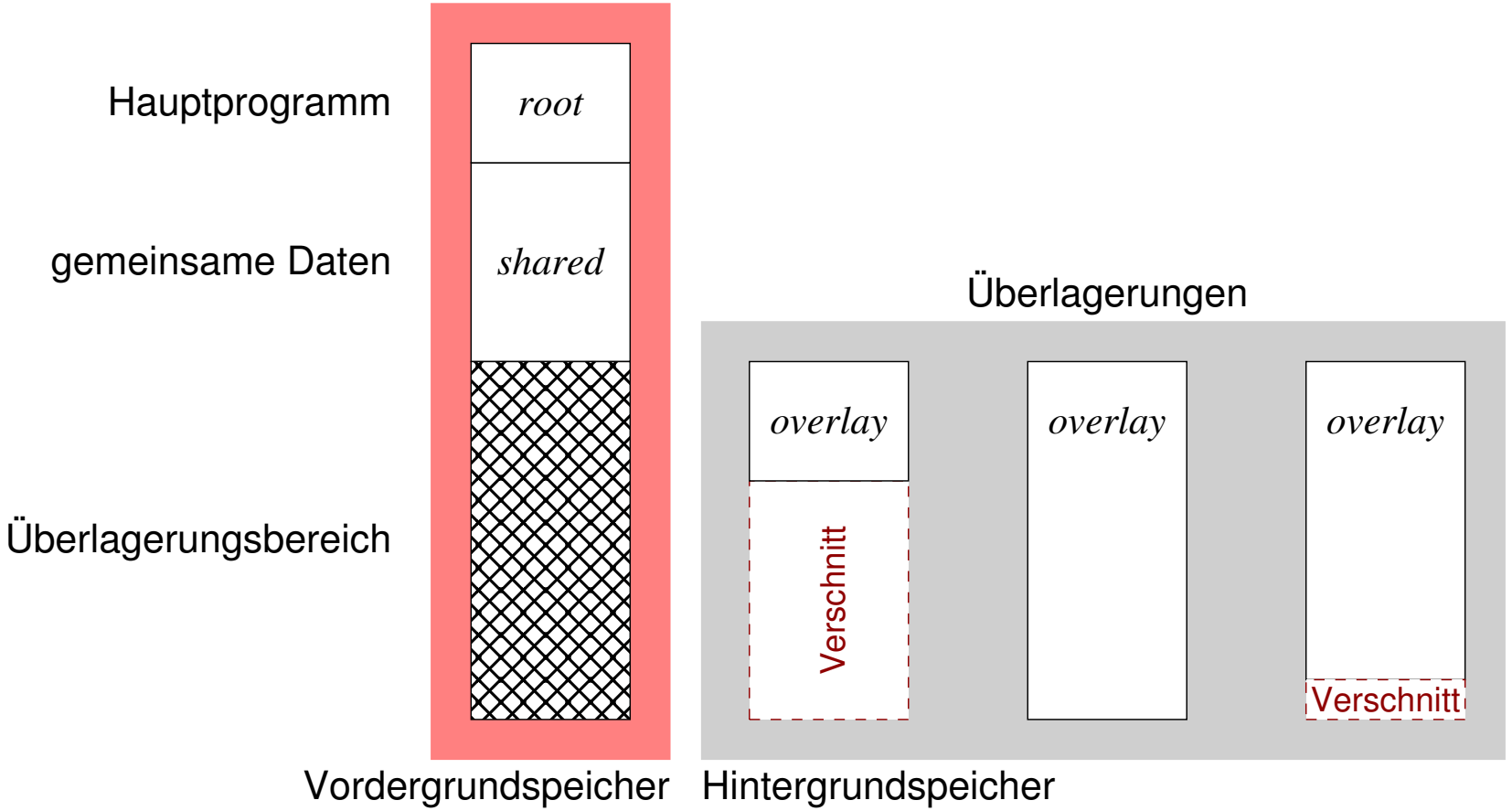
- Programme, die Überlagerungen enthalten, sind (grob) dreigeteilt:
  i. ein hauptspeicherresidenter Programmteil (*root program*), repräsentiert das Überlagerungen aufrufende Hauptprogramm
  ii. mehrere in Überlagerungen zusammengefasste Unterprogramme, die ihrerseits Überlagerungen aufrufen können
  iii. ein gemeinsamer Datenbereich (*common section*), zur Speicherung überlagerungsübergreifender Information
- Überlagerungen sind das Ergebnis einer **Programmzerlegung** zur Programmier-, Übersetzungs- und/oder Bindezeit
  - manuelle bzw. automatisierte **Abhängigkeitsanalyse** des Programms
  - Anzahl und Größe von Überlagerungen werden **statisch** festgelegt
  - lediglich aktivieren (d.h. laden) von Überlagerungen ist dynamisch

Hinweis

Überlagerungen sind Teile eines Programms und keine Elemente einer mehreren Programmen gemeinsamen Bibliothek (shared library) oder DLL (dynamic link library).

Programm-/Adressraumstruktur

- Organisation des Prozessadressraums im Hauptspeicher:



- **Problem**:
  - Verkettungstechnik, kein Zurückschreiben von Überlagerungen