



Bild von Mike by Pexels

(Grundlagen der) Betriebssysteme | G.1



Franz J. Hauck | Institut für Verteilte Systeme, Univ. Ulm



Bild von Mike by Pexels

G | Speicherverwaltung (Grundlagen der) Betriebssysteme

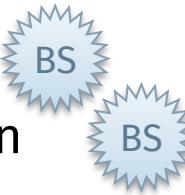


Franz J. Hauck | Institut für Verteilte Systeme, Univ. Ulm

Überblick

Überblick der Themenabschnitte

- A – Organatorisches
- B – Zahlendarstellung und Rechnerarithmetik
- C – Aufbau eines Rechnersystems
- D – Einführung in Betriebssysteme
- E – Prozessverwaltung und Nebenläufigkeit
- F – Dateiverwaltung
- G – Speicherverwaltung
- H – Ein-, Ausgabe und Geräteverwaltung
- I – Virtualisierung
- J – Verklemmungen
- K – Rechteverwaltung



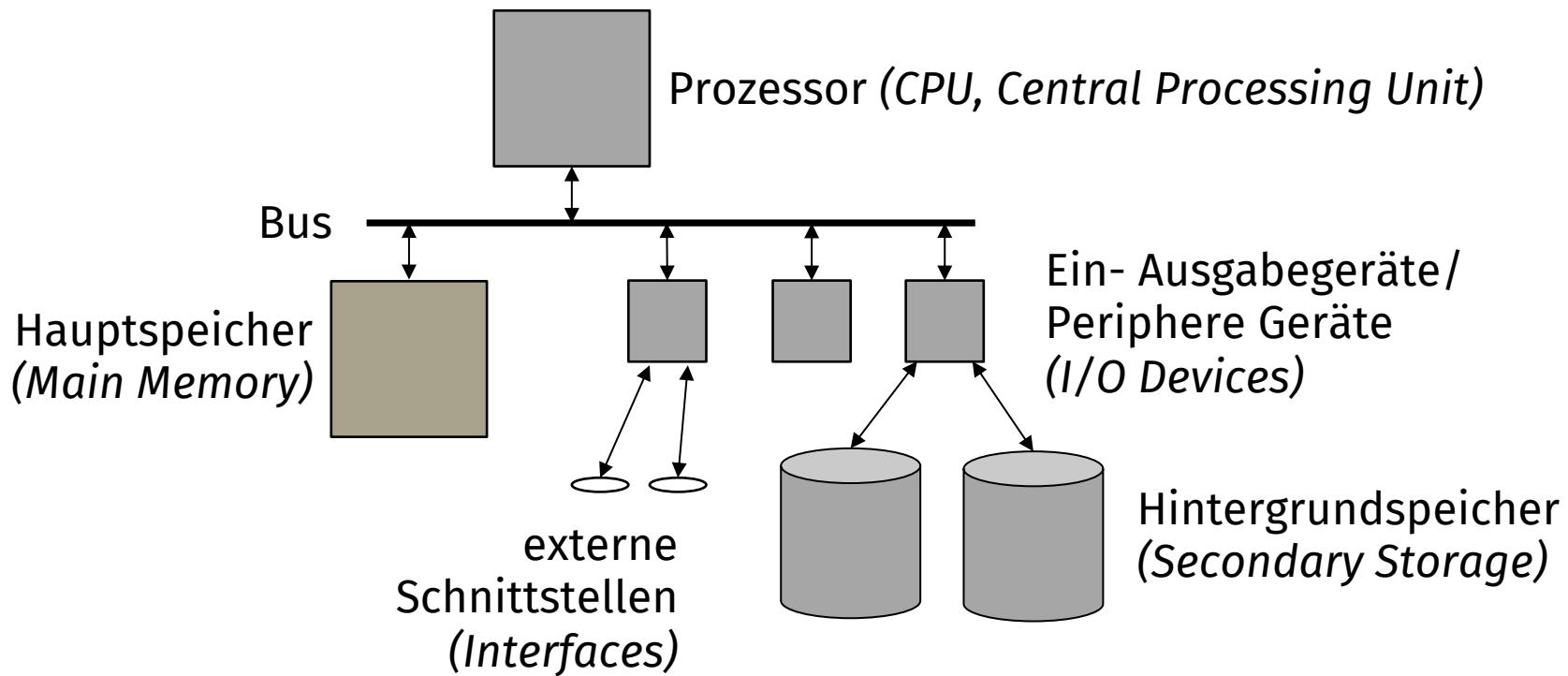
Inhaltsüberblick

Speicherverwaltung

- Phänomene
- Speichervergabe
 - statische und dynamische Zuteilung, Vergabestrategien
- Mehrprozessbetrieb
 - Segmentierung, Kompaktieren
 - Seitenadressierung
 - TLB
- Virtueller Speicher
 - Seitenersetzungsstrategien, FIFO, B₀, LRU, 2nd-Chance
 - Mehrprozessbetrieb
- Systemaufrufe

Einordnung

Betroffene physikalische Ressourcen



Phänomene der Speicherverwaltung

Beispiel: Initialisierung großer Matrizen (geschrieben in C)

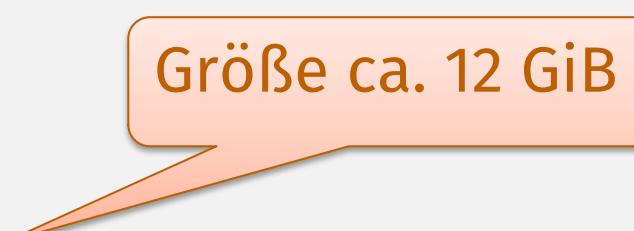
■ Variante 1

```
#define DIM 40000

int main()
{
    register long i, j;
    static long matrix[DIM][DIM];

    for ( i = 0; i < DIM; i++ )
        for ( j = 0; j < DIM; j++ )
            matrix[i][j] = 1;

    return 0;
}
```



Größe ca. 12 GiB

Phänomene der Speicherverwaltung (2)

Beispiel: Initialisierung großer Matrizen (geschrieben in C)

■ Variante 2

```
#define DIM 40000

int main()
{
    register long i, j;
    static long matrix[DIM][DIM];

    for ( j = 0; j < DIM; j++ )
        for ( i = 0; i < DIM; i++ )
            matrix[i][j] = 1;

    return 0;
}
```

Schleifen
vertauscht

Phänomene der Speicherverwaltung (3)

Messergebnisse (Intel Xeon 1220E3 ca. 3,1 GHz, 16GB)

■ Variante 1

- User time = **1,7 s**; System time = **10,3 s**; Gesamtzeit = **11,9 s**

■ Variante 2

- User time = **40,4 s**; System time = **120,6 s**; Gesamtzeit = **161,0 s**

■ Ursachen

- Variante 1 geht sequentiell durch den Speicher
- Variante 2 greift versetzt ständig auf gesamten Speicherbereich zu
- Beispiel: **matrix[4][4]** und die ersten fünf Zugriffe

■ Variante 1



■ Variante 2



Phänomene der Speicherverwaltung (4)

Ursachen

■ logischer Adressraum

- benutzte Adressen sind nicht die realen (*physikalischen*) Adressen
- *Abbildung* wird durch Hardware auf Seitenbasis vorgenommen (*Seitenadressierung*)
- Variante 2 hat weniger *Lokalität*, d.h. benötigt häufig wechselnde Abbildungen (viel mehr Seiten)

■ virtueller Speicher

- möglicher Adressraum ist größer als *realer* Speicher
- auf Seitenbasis werden Teile des benötigten Speichers ein- und ausgelagert
- viel mehr Speicher *Ein- und Auslagerungen* bei Variante 2



Bild von Mike by Pexels

(Grundlagen der) Betriebssysteme | G.2



Franz J. Hauck | Institut für Verteilte Systeme, Univ. Ulm

Inhaltsüberblick

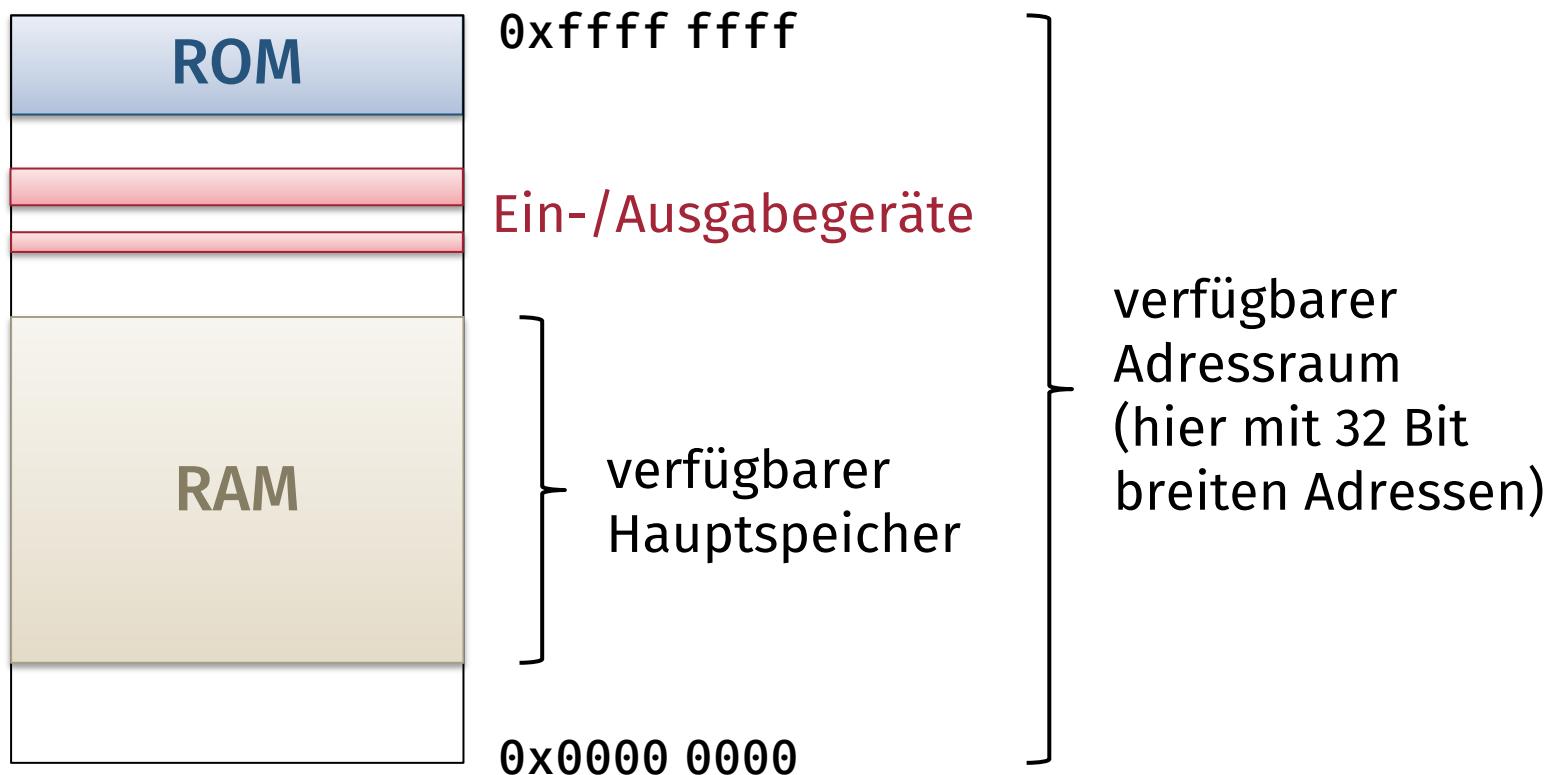
Speicherverwaltung

- Phänomene
- Speichervergabe
 - statische und dynamische Zuteilung, Vergabestrategien
- Mehrprozessbetrieb
 - Segmentierung, Kompaktieren
 - Seitenadressierung
 - TLB
- Virtueller Speicher
 - Seitenersetzungsstrategien, FIFO, B₀, LRU, 2nd-Chance
 - Mehrprozessbetrieb
- Systemaufrufe

Speichervergabe

Verfügbarer Speicher

- Beispiel eines Adressraums



Speichervergabe (2)

Belegung des verfügbaren Hauptspeichers durch

- Benutzerprogramme
 - Programmbefehle (Code, Binary)
 - Programmdaten
- Betriebssystem
 - Betriebssystemcode
 - Pufferspeicher
 - Systemvariablen

→ Zuteilung des Speichers nötig

Statische Speicherzuteilung

Feste Bereiche für Betriebssystem und jeden Prozess

■ Probleme

- Vorhersage notwendiger Größen
- ungenutzter Speicher vs. zu geringe Zuteilung
- zu geringe Zuteilung hat weitere Konsequenzen
 - z.B. geringe Datenrate beim Netzwerk wg. zu kleiner Puffer

→ Einsatz dynamischer Speicherzuteilung

Dynamische Speicherzuteilung

Segmente

- zusammenhängender Speicherbereich
 - mit aufeinanderfolgenden Adressen
- einzelne Segmente mit unterschiedlicher Länge

Speicherzuteilung auf Basis von Segmenten

- Anforderung von Segmenten (Allokation)
 - bei Bedarf
 - Zuteilung eines freien Bereichs geeigneter Größe
 - **Zuteilungsalgorithmus** zur Auswahl des Bereichs
- Freigabe nicht mehr benötigter Segmente
 - Weiterverwendung für nächste Anforderungen

Dynamische Speicherzuteilung (2)

Typische Segmente eines Prozesses

■ Code-Segment

- Instruktionen des Prozesses
- initialisiert durch Programmdatei

■ Daten-Segment

- vorbelegte Datenstrukturen
 - initialisiert durch Programmdatei
- andere Datenstrukturen
 - initialisiert mit Nullen

■ Stack-Segment

- für Verwaltungsinformationen
 - z.B. für verschachtelte Funktionsaufrufe oder Unterbrechungen

Dynamische Speicherzuteilung (3)

Zuteilungsoperation

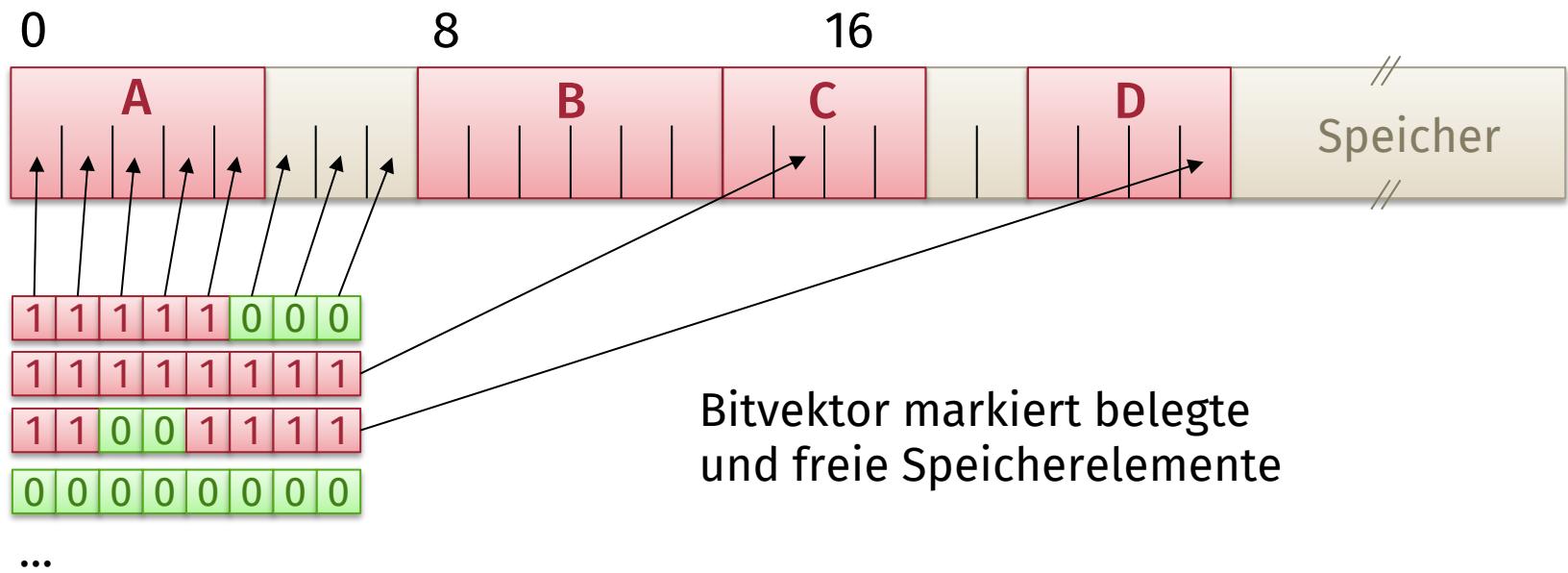
- Repräsentation des **freien Speichers**
 - als Datengrundlage für Zuteilungsalgorithmus
- Zuteilungsalgorithmus
 - Auswahl einer geeigneten freien Lücke im Speicher
 - eventuell Splitten der Lücke in den zu belegenden und dann noch freien Bereich

Freigabeoperation

- Eventuell Repräsentation des **belegten Speichers**
 - zur Absicherung von Freigaben

Freispeicherverwaltung

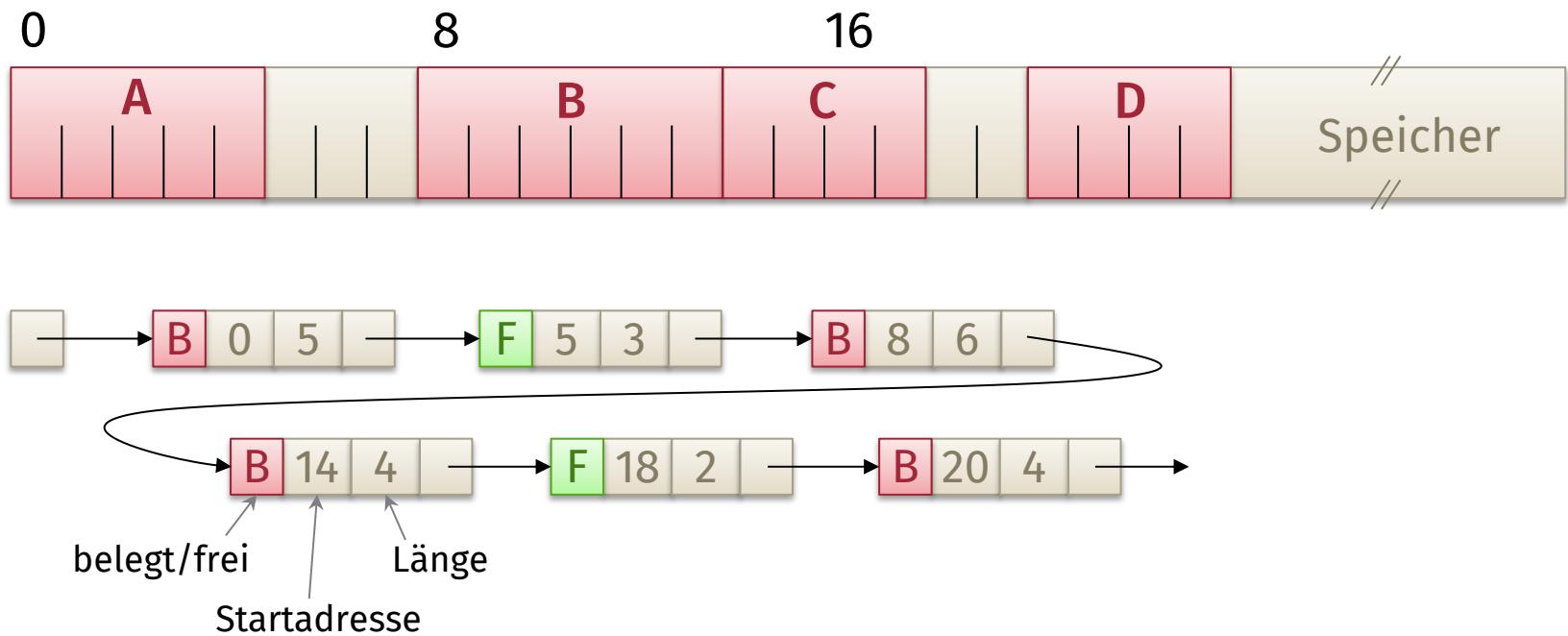
Variante: Bitlisten / Bitvektoren



- Speicherelemente gleicher Größe
 - z.B. 1 Byte, 64 Bytes, 1 KiB etc.

Freispeicherverwaltung (2)

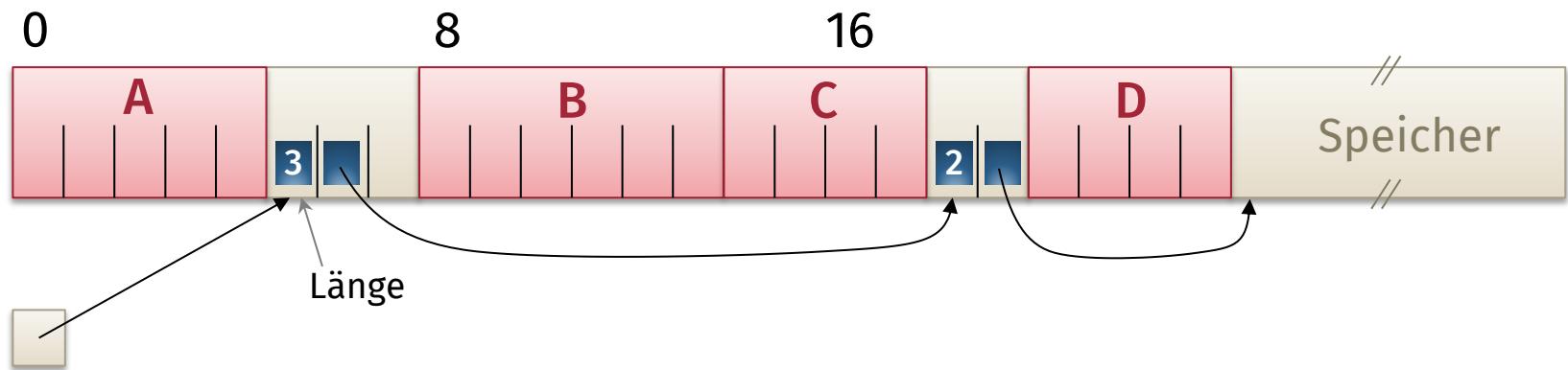
Variante: Verkettete Liste



- Repräsentation von freien und belegten Segmenten

Freispeicherverwaltung (3)

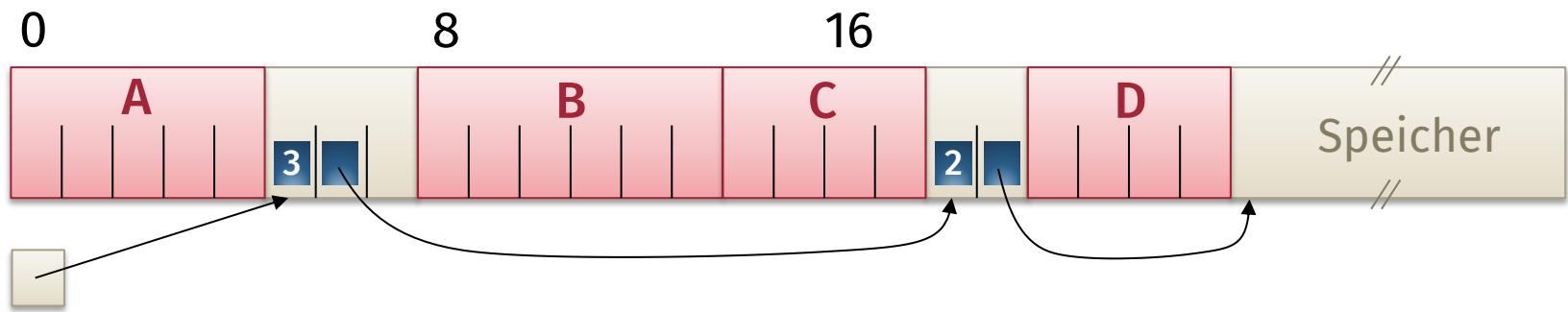
Variante: Verkettete Liste im freien Speicher



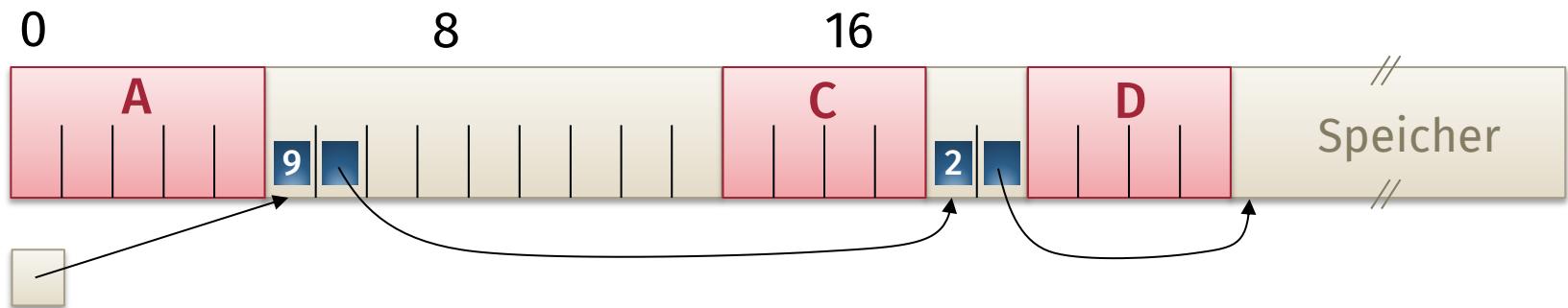
- Repräsentation nur der freien Segmente
- zur Effizienzsteigerung eventuell auf Rückwärtsverkettung
 - abhängig auch von der Zuteilungsstrategie

Freispeicherverwaltung (4)

Verkettete Liste: Verschmelzung von Lücken



Nach Freigabe von Segment B



Vergabestrategien

First Fit

- erste passende Lücke wird verwendet

Rotating First Fit / Next Fit

- wie First Fit, aber Start bei der zuletzt zugewiesenen Lücke

Best Fit

- kleinste passende Lücke wird gesucht

Worst Fit

- größte passende Lücke wird gesucht

Vergabestrategien (2)

Probleme

- **Speicherverschnitt** (Fragmentierung)
 - durch ständige unsystematische Anforderungen und Freigaben
 - zu kleine Lücken für große Anforderungen
- **Laufzeit des Algorithmus**
 - komplexe Algorithmen können eventuell durch geeignete Datenstrukturen effizient gemacht werden

Idealer Algorithmus?

- bräuchte **Kenntnisse** über Anforderungen und Freigaben
 - insbesondere Größen sind wichtig

Einsatz der Verfahren

Einsatz im Betriebssystem

- Verwaltung des Systemspeichers
- Zuteilung von Speicher an Prozesse und Betriebssystem

Einsatz innerhalb eines Prozesses

- Verwaltung des **Haldenspeichers (Heap)**
 - dynamische Allokation von Speicherbereichen durch den Prozess

Einsatz für Bereiche des Sekundärspeichers

- Verwaltung bestimmter Abschnitte des Sekundärspeichers
 - Festplatte und ähnliches
 - z.B. Speicherbereich für **Prozessauslagerungen (Swap Space)**

Exkurs: Haldenspeicher

Typischer Aufbau des Datensegments eines Prozesses

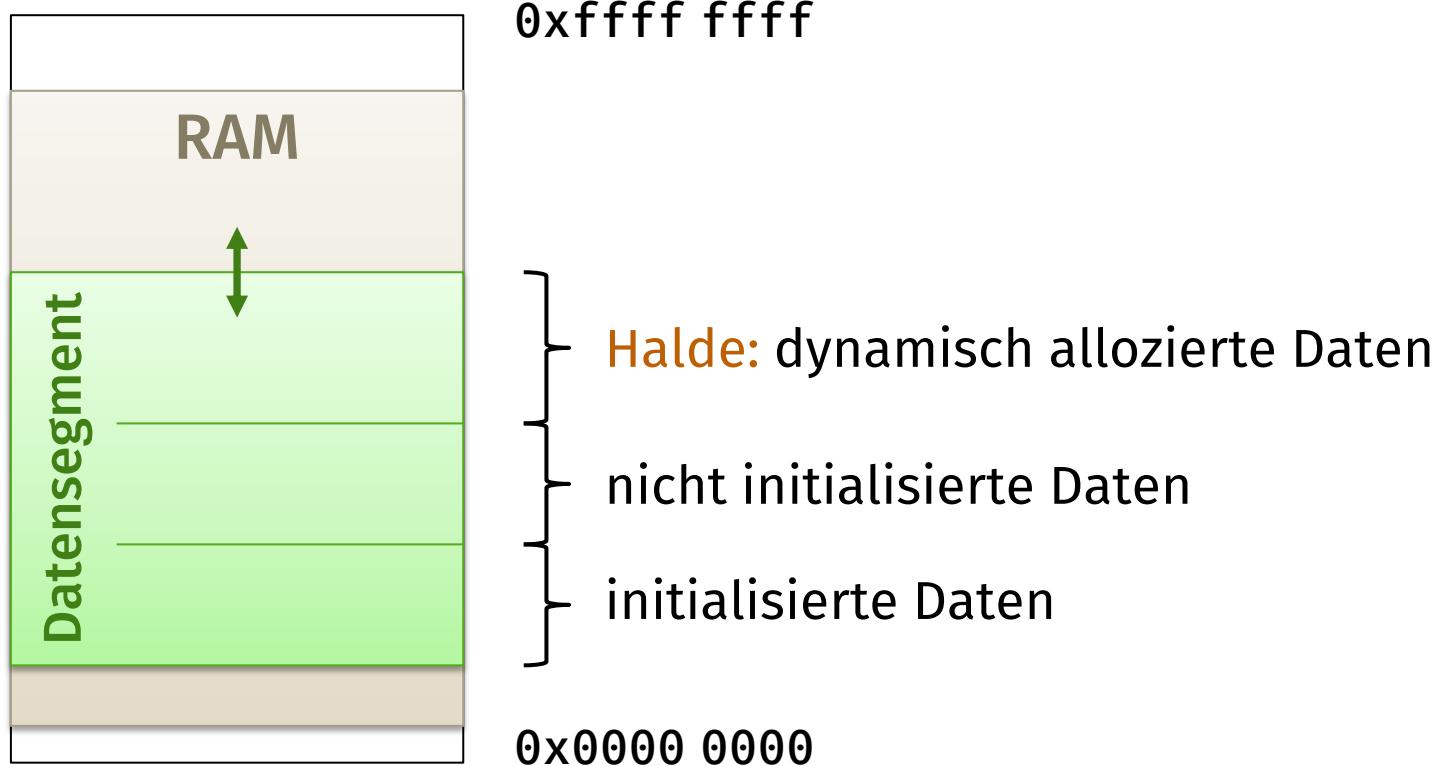




Bild von Mike by Pexels

(Grundlagen der) Betriebssysteme | G.3



Franz J. Hauck | Institut für Verteilte Systeme, Univ. Ulm

Inhaltsüberblick

Speicherverwaltung

- Phänomene
- Speichervergabe
 - statische und dynamische Zuteilung, Vergabestrategien
- Mehrprozessbetrieb
 - Segmentierung, Kompaktieren
 - Seitenadressierung
 - TLB
- Virtueller Speicher
 - Seitenersetzungsstrategien, FIFO, B₀, LRU, 2nd-Chance
 - Mehrprozessbetrieb
- Systemaufrufe

Mehrprozessbetrieb

Segmente mehrerer Prozesse im Hauptspeicher

- unterschiedliche Adressen für Segmente
 - z.B. zwei Prozesse des selben Programms
 - Instruktionen müssen in beiden Prozessen jeweils verschiedene Adressen ansprechen
 - Hauptspeicher **nicht ausreichend**
 - für alle Segmente und alle Prozesse
 - Prozesse **nicht** voreinander geschützt
 - Betriebssystem auch nicht
- ❖ Idee: Einführung logischer Adressräume

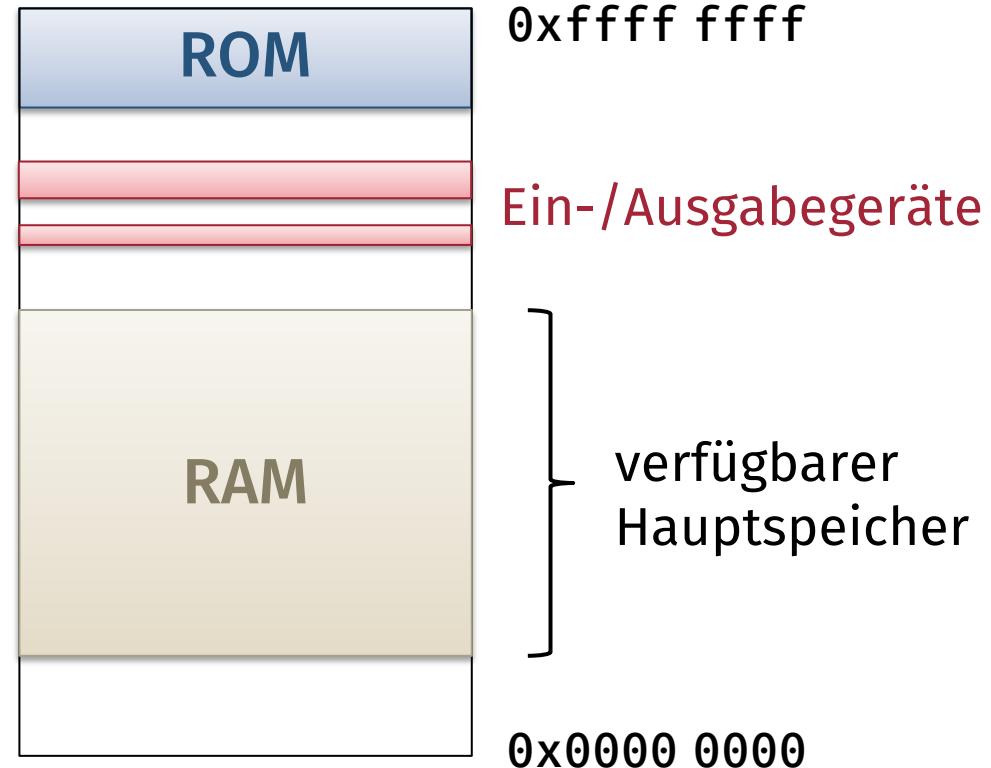


Adressraum

Physikalischer Adressraum

■ Adressraums der Hardware

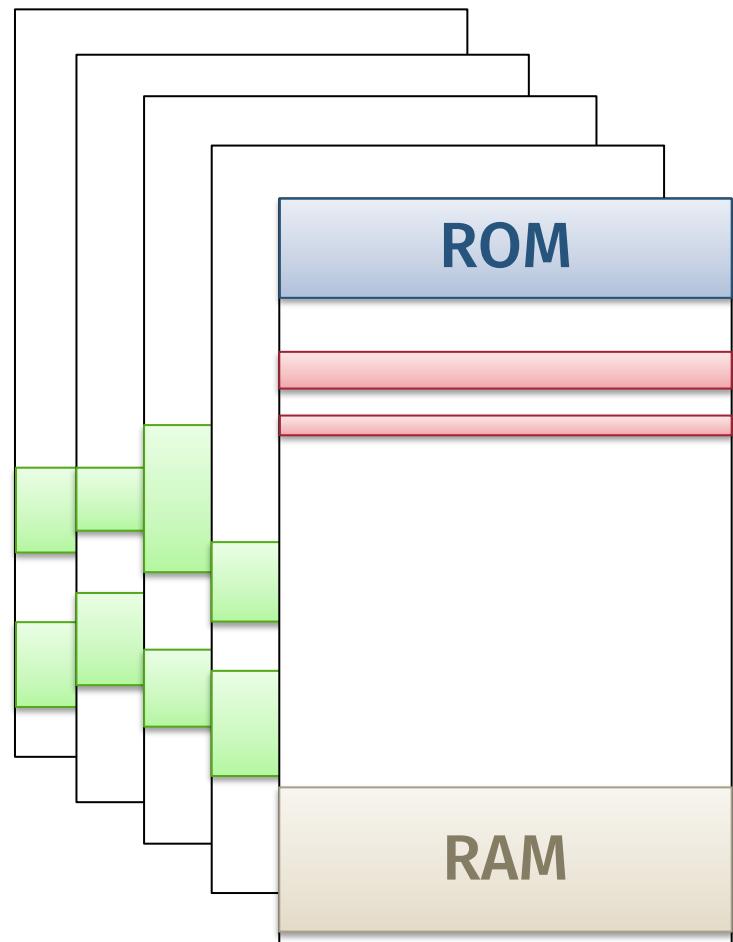
- besteht aus mehreren Bereichen
 - DRAM
 - Video RAM
 - Boot-ROM
 - Ein-, Ausgabe
 - ...
- nicht alle Adressen verfügbar



Adressraum (2)

Logischer Adressraum

- Adressraum pro Prozess
 - enthält **nur** Segmente des jeweiligen Prozesses (*User Space*)
- Adressraum für Betriebssystem
 - enthält **Systemdaten, ROM und Ein-, Ausgabe** (*Kernel Space*)



Adressraum (3)

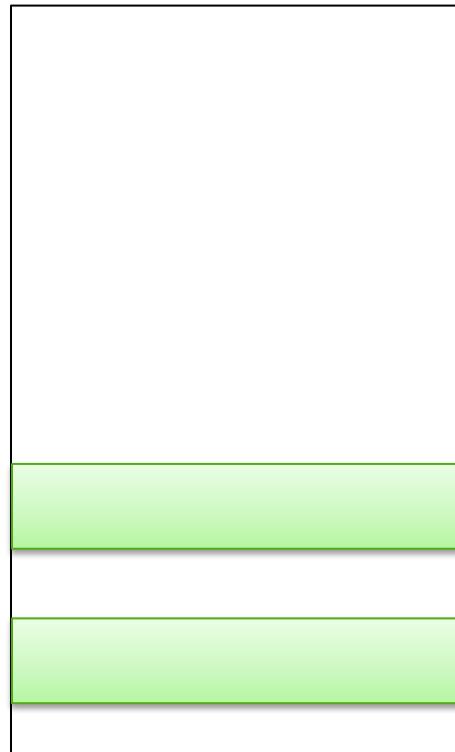
Logischer Adressraum (fortges.)

- benötigt **Abbildung** auf physikalische Adressen
 - evtl. **Verschiebung** gegenüber den logischen Adressen
 - zwei Prozesse mit **gleichen** logischen Adressen möglich
 - aber jeweils **unterschiedlichen** physikalischen Adressen
- nur abgebildete Adressen **zugreifbar**
 - Schutzfunktion bzw. Schutzumgebung
- bei aktiver Abbildung läuft **Prozessor mit logischen Adressen**
 - für **alle** Speicherzugriffe
 - erfordert weitere Hardware-Unterstützung
 - **Memory-Management-Unit (MMU)**

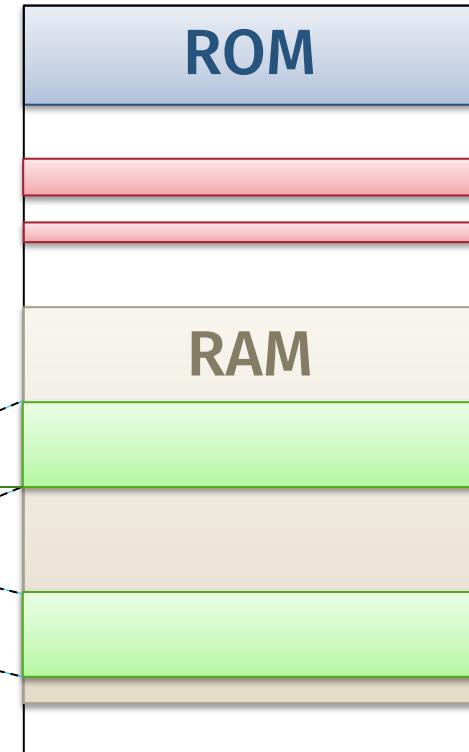
Segmentierung

Logischer Adressraum mit mehreren Segmenten

logischer Adressraum

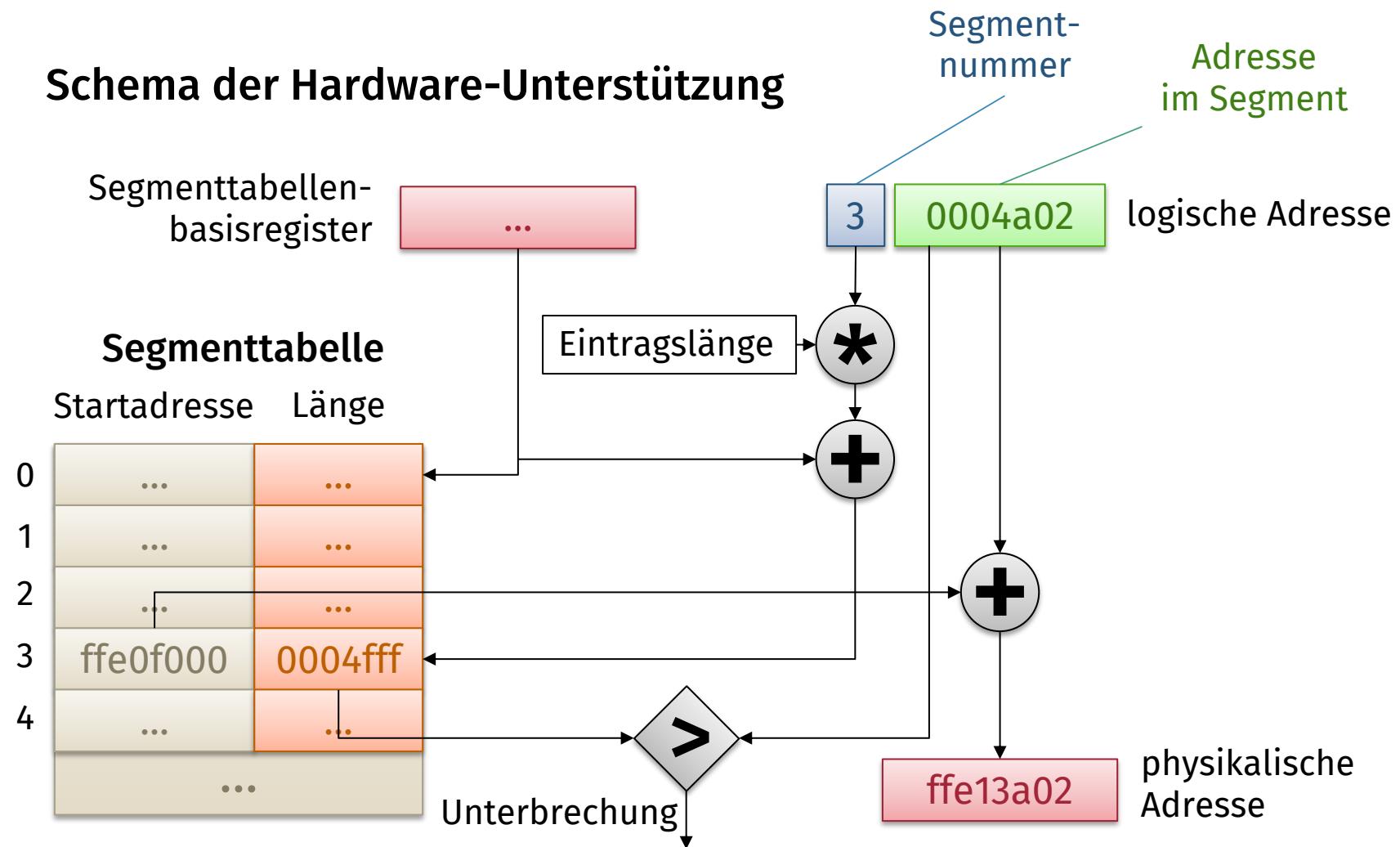


physikalischer Adressraum



Segmentierung (2)

Schema der Hardware-Unterstützung



Segmentierung (3)

Vorteile

- individuelle Schutzumgebung
 - Unterbrechung zeigt Speicherverletzung an
 - Prozesse und Betriebssystem voreinander geschützt
- Kontextwechsel kaum komplexer
 - zusätzlicher Austausch der Segmentbasis
- einfache Ein- und Auslagerung
 - Wiedereinlagerung an beliebige Stelle im Speicher
 - lediglich Anpassung der Segmenttabelle

Segmentierung (4)

Vorteile (fortges.)

■ gemeinsame Segmente

- Code-Segmente (Befehlssegmente)
- Datensegmente zur gemeinsamen Nutzung (*Shared Memory*)

■ einfache Rechteintegration

- z.B. Lesen, Schreiben, Ausführen
- Vermerk im Segmenttabelleneintrag und Prüfung durch MMU

Segmentierung (5)

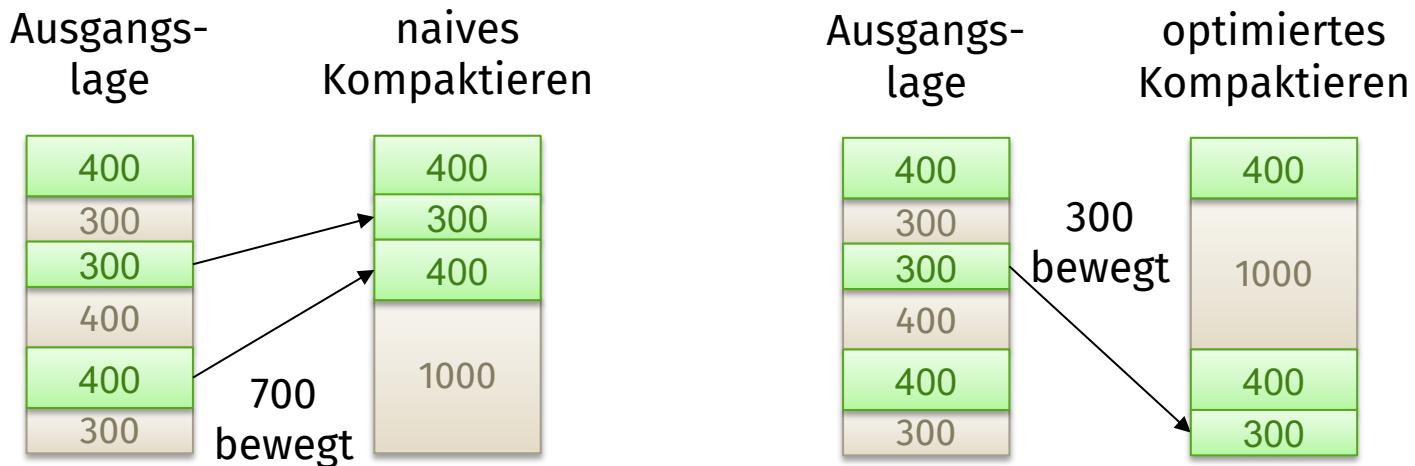
Probleme

- implizite Speicherzugriffe
 - zusätzlicher Zugriff auf Segmenttabelle
- Ein-, Auslagerung sehr grobgranular
 - lange I/O-Zeiten
 - nicht alle Teile eines Segments werden gleich häufig genutzt
- (externe) Fragmentierung des Speichers
 - nach Ein- und Auslagerungen evtl. kleine, nicht nutzbare Lücken
 - Lösungsansatz: Kompaktieren
 - Verschieben der Segmente, um Lücken zu schließen
 - Anpassung der Segmenttabelle

Kompaktieren

Verschieben von Segmenten

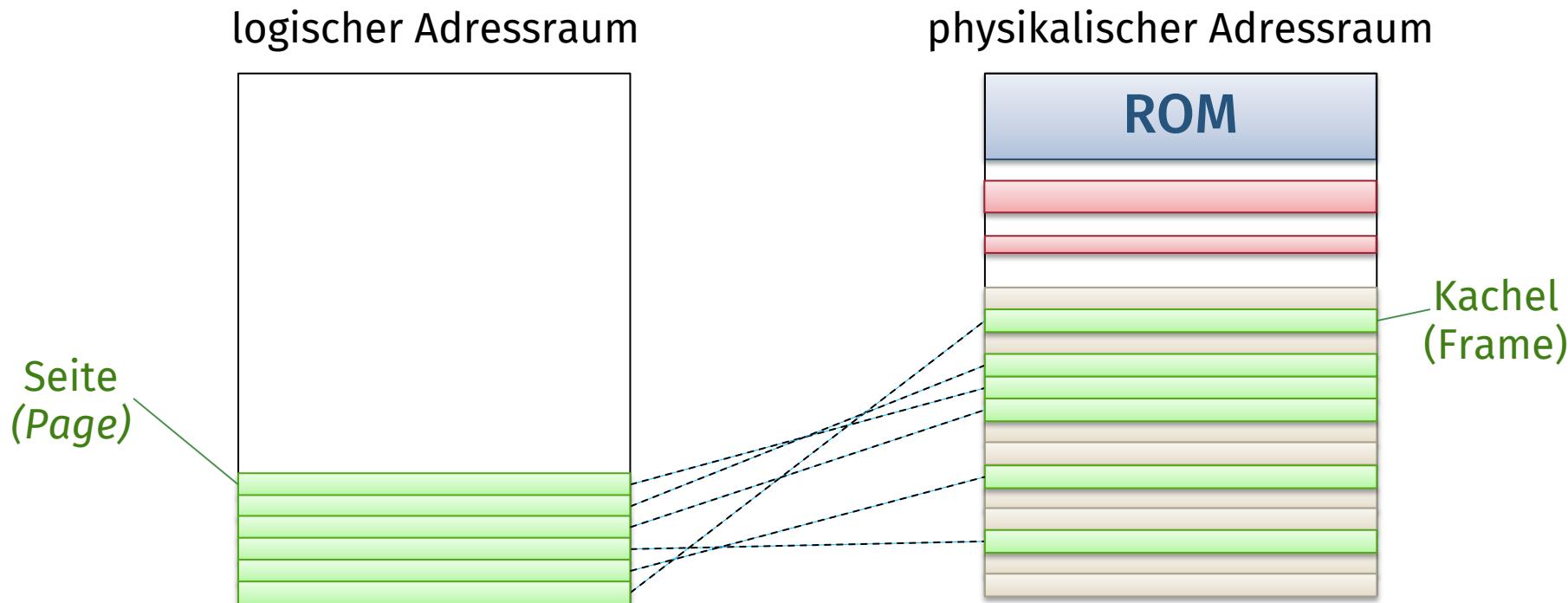
- weniger, aber größere Lücken verringern Verschnitt
- aufwändige Operation
 - abhängig von der Größe der zu verschiebenden Segmente
- Beispiel:



Seitenadressierung (Paging)

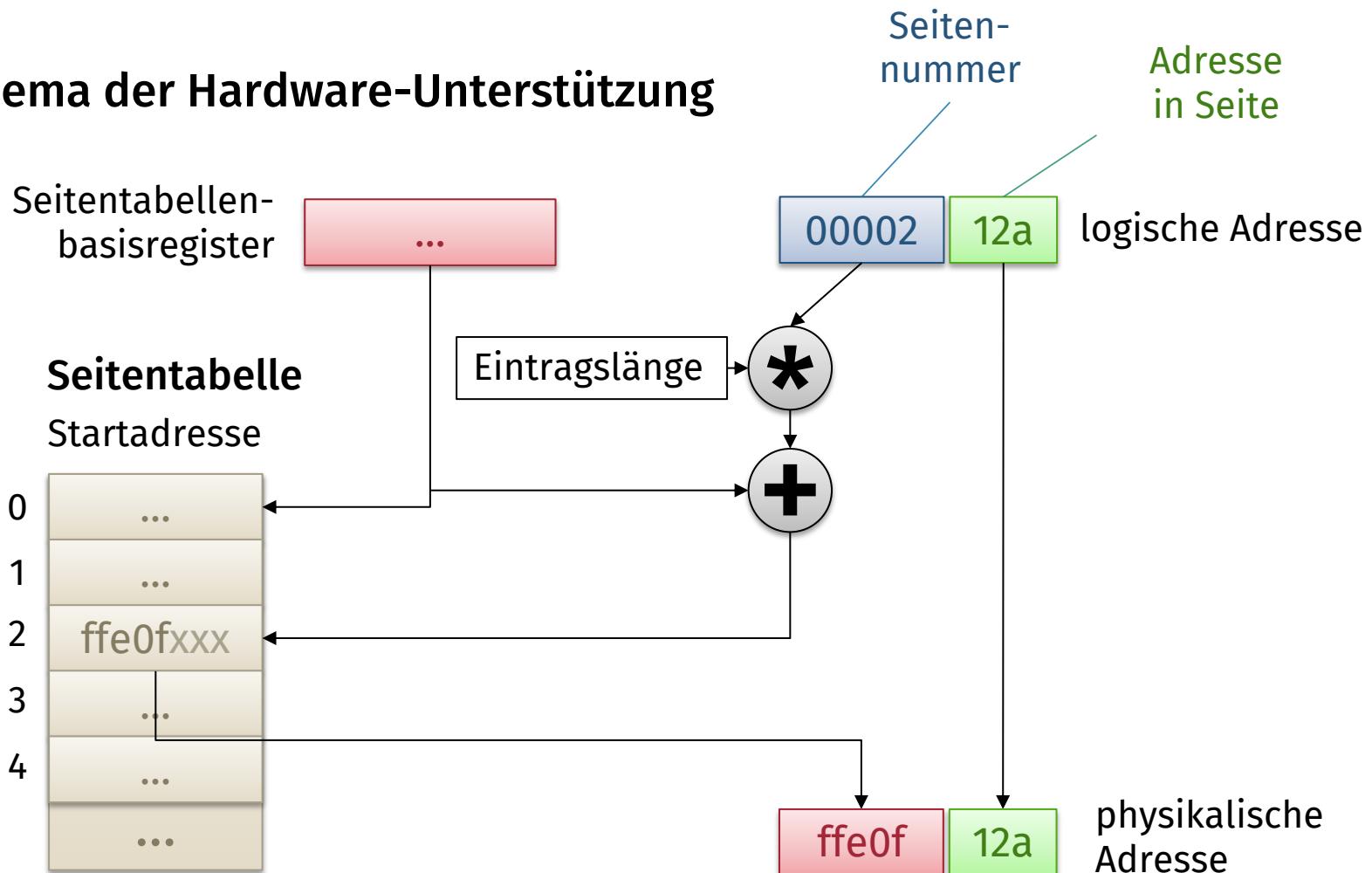
Logischer Adressraum aus vielen Seiten

- Seiten mit gleicher Größe
 - Abbildung jeder einzelnen Seite in den Hauptspeicher



Seitenadressierung (2)

Schema der Hardware-Unterstützung



Seitenadressierung (3)

Vorteile

- kein externer Verschnitt
- vereinfachte Speichervergabe
 - nur noch auf Seitenbasis, d.h. Anforderungen gleicher Größe
- kein Kompaktieren nötig
- vereinfachte Ein- und Auslagerung
 - auf Seitenbasis

Seitenadressierung (4)

Implementierungsdetails

- Seitengröße
 - typisch 512 Byte bis 8 KiB
 - heute auch 2, 4 oder 8 MiB (*Jumbo Pages*)
 - Größe ist Zweierpotenz

- Startadresse der Kachel durch Seitengröße teilbar
 - erlaubt vereinfachte Zusammensetzung der physikalischen Adresse

Seitenadressierung (5)

Probleme von Seitenadressierung

■ interner Verschnitt

- letzte Seite nicht vollständig genutzt
- kleinere Seiten verringern internen Verschnitt
 - vergrößern Seitenkacheltabelle

Probleme der gezeigten Variante

■ nur ein „Segment“

■ sehr große Seitenkacheltabelle

- abhängig von der Größe des logischen Adressraums
- muss im Speicher gehalten werden

■ implizite Speicherzugriffe

gelöst durch
Kombination mit
Segmentierung



Bild von Mike by Pexels

(Grundlagen der) Betriebssysteme | G.4



Franz J. Hauck | Institut für Verteilte Systeme, Univ. Ulm

Inhaltsüberblick

Speicherverwaltung

■ Phänomene

■ Speichervergabe

- statische und dynamische Zuteilung, Vergabestrategien

■ Mehrprozessbetrieb

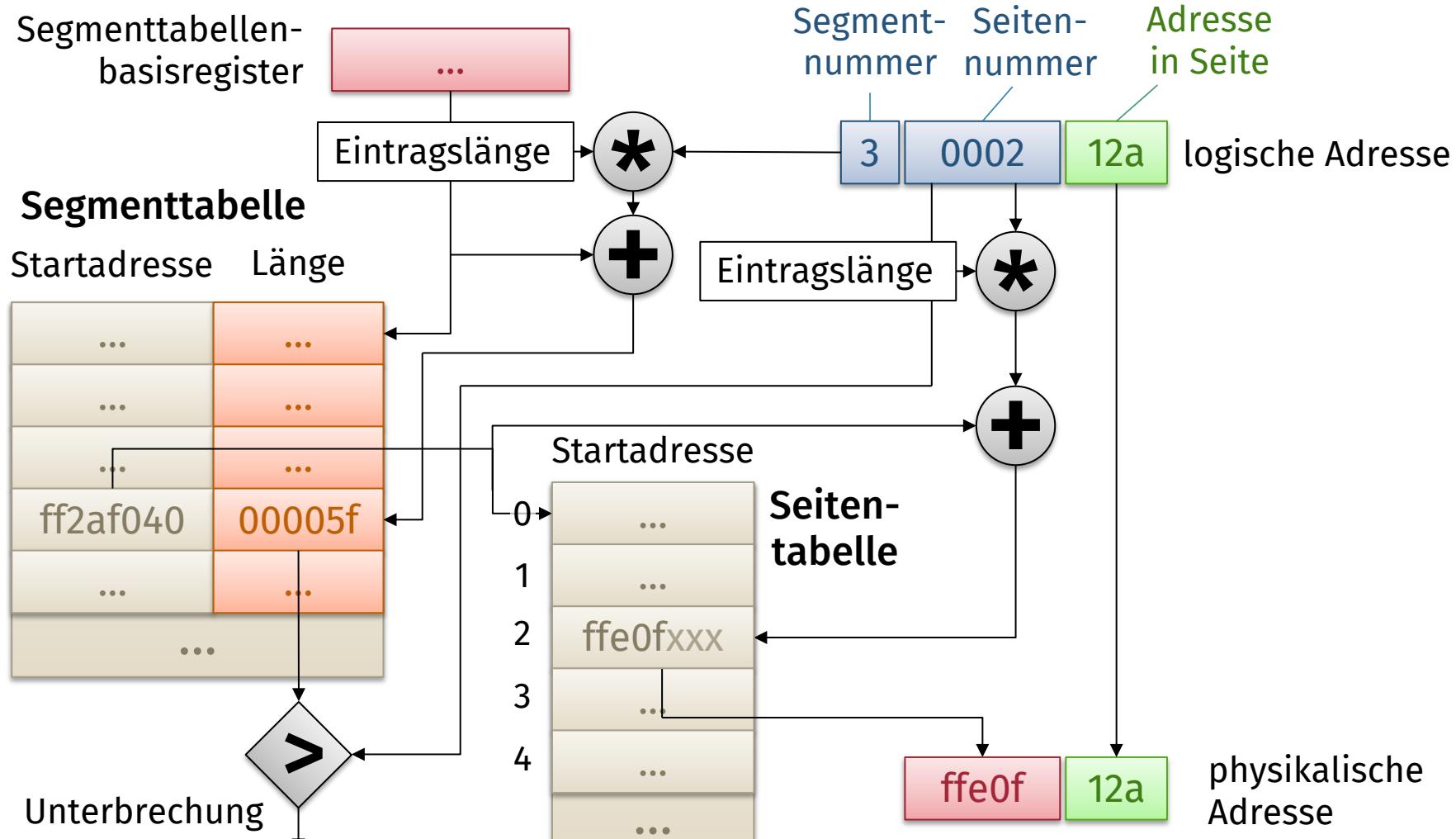
- Segmentierung, Kompaktieren
- Seitenadressierung
- TLB

■ Virtueller Speicher

- Seitenersetzungsstrategien, FIFO, B₀, LRU, 2nd-Chance
- Mehrprozessbetrieb

■ Systemaufrufe

Segmentierung und Seitenadressierung



Segmentierung und Seitenadressierung (2)

Vorteile

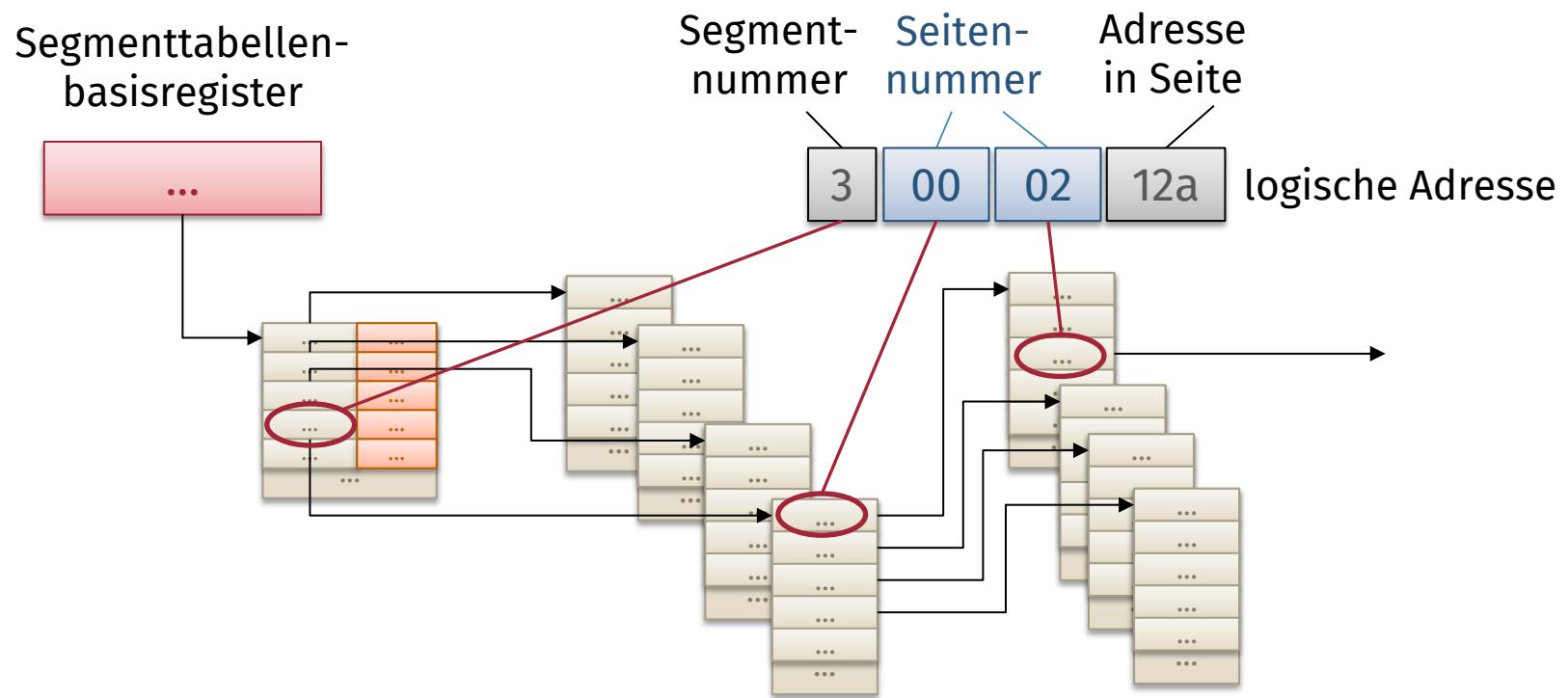
- Segmente mit **definierter Seitenzahl** möglich
- Seitentabelle in **Größe begrenzt**

Nachteil

- Seitentabelle evtl. **sehr lang**
 - große Segmente
 - Speicherzuteilung für Seitentabellen und Kacheln **unterschiedlich**
 - variable Größe vs. feste Größe
 - noch mehr **implizite Speicherzugriffe**

Hierarchische Seitenadressierung

Begrenzung der Seitentabellen durch Indirektionsstufen



Hierarchische Seitenadressierung (2)

Vorteil

- Seitentabellen jeweils so groß wie Kachel
 - Speicherverwaltung stark vereinfacht

Nachteil

- für jede Ebene ein weiterer impliziter Speicherzugriff

Unterstützung für Ein- und Auslagerung

Ein-, Auslagerung auf Seitenbasis

- nicht das ganze Segment muss ausgelagert werden
- nicht alle Seiten müssen eingelagert sein
 - benötigt Hardware-Unterstützung

Hardware-Unterstützung für Ein- und Auslagerung

- Präsenzbit
 - in Seiten- und Segmentdeskriptoren (Einträgen)
 - **0:** Seite/Zwischenebene ist ausgelagert
 - **1:** Seite/Zwischenebene ist vorhanden
 - automatische Berücksichtigung des Präsenzbits beim Speicherzugriff

Unterstützung für Ein- und Auslagerung (2)

Hardware-Unterstützung für Ein- und Auslagerung (fortges.)

■ Präsenzbit

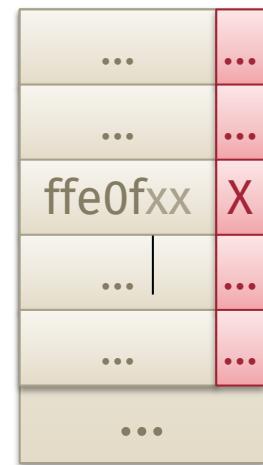
- typischerweise in niederwertigen Bits der Startadresse platziert

■ Zugriff mit Präsenzbit = 1

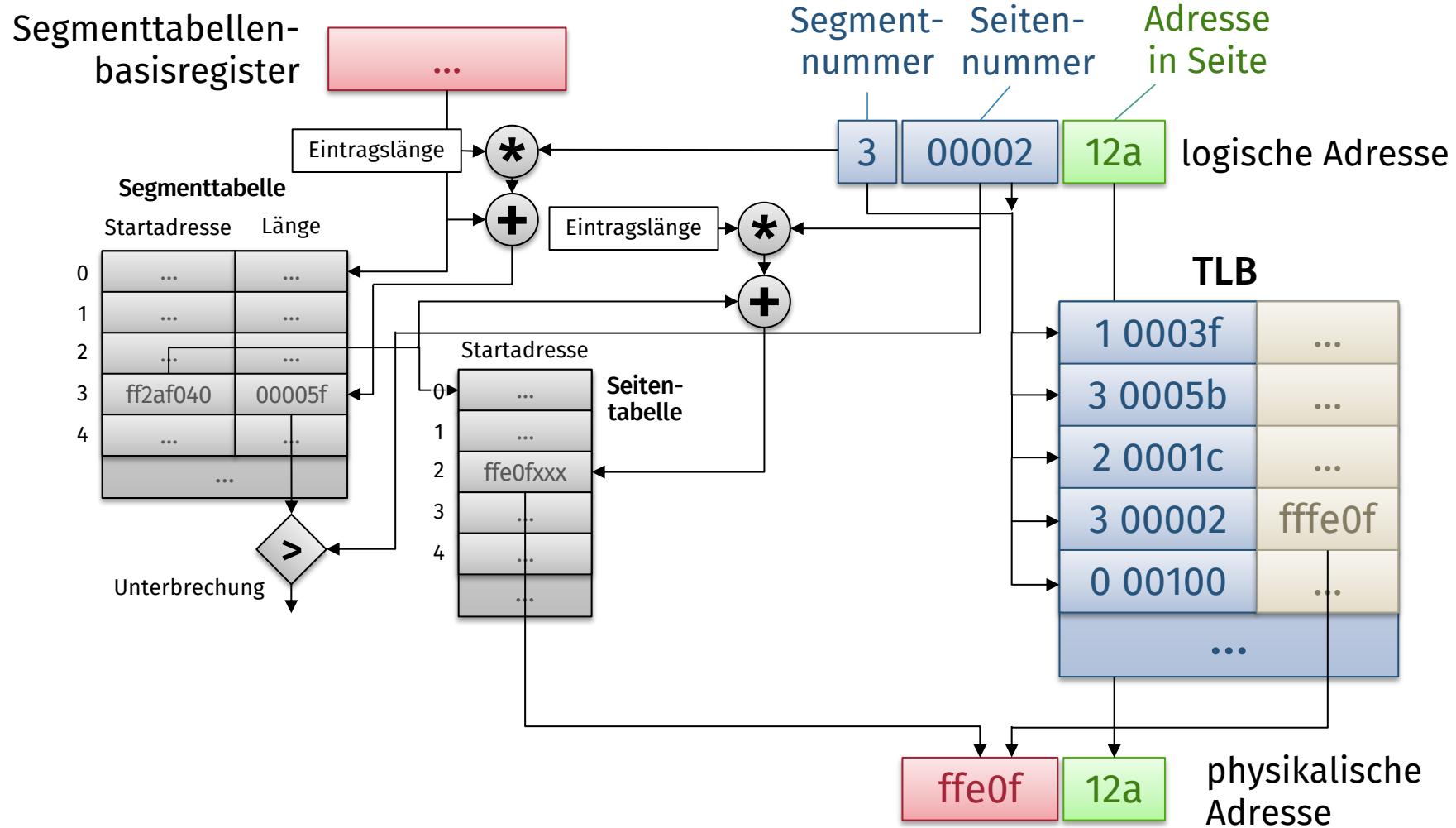
- wie bisher

■ Zugriff mit Präsenzbit = 0

- Auslösung einer internen Unterbrechung (*Page Fault*)
- egal auf welcher Hierarchieebene Präsenzbit fehlt
- Betriebssystem kann für **Einlagerung** sorgen
- unterbrochene Instruktion kann **wiederholt** werden
 - benötigt CPU-Unterstützung



Translation Look-Aside Buffer (TLB)



Translation Look-Aside Buffer, TLB (2)

Eigenschaften

- **schneller Zugriff auf Seitenabbildung**
 - falls Information im TLB-Speicher gespeichert (*TLB Hit*)
 - **keine impliziten Speicherzugriffe** nötig
- **Eintragung** bei fehlender Abbildung im TLB (*TLB Miss*)
 - **Ersetzung** eines alten Eintrags (z.B. LRU, siehe später)
 - bei erneutem Zugriff TLB Hit (Lokalität)
- bei **Kontextwechseln** muss TLB evtl. **gelöscht** werden (*Flush*)

Translation Look-Aside Buffer, TLB (3)

Typische TLB-Größen

■ Pentium:

- Daten-TLB: 64 Einträge, Code-TLB: 32, Seitengröße: 4 KiB

■ Sparc V9:

- Daten-TLB: 64, Code-TLB: 64, Seitengröße: 8 KiB

■ Core i7:

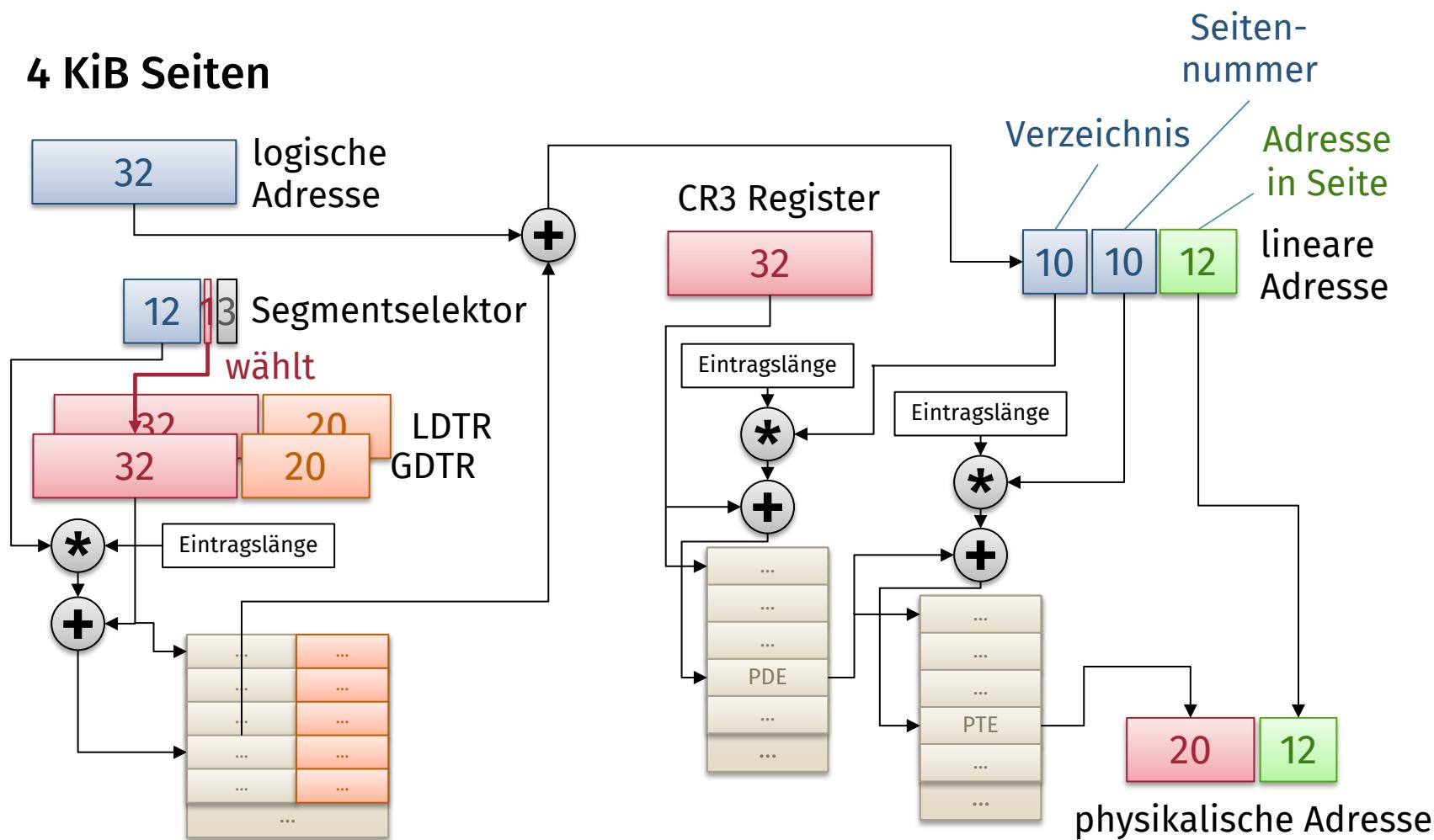
- Level 1 Daten-TLB: 64, Level 1 Code-TLB: 128
- Level 2 Daten- und Code-TLBs: 512, Seitengröße: 4 KiB

■ teilweise getrennte TLBs für kleine und große Seiten

- große Seiten in der Größenordnung von 2/4 MiB

Fallbeispiel: Intel x86-x32 Speicherarchitektur

4 KiB Seiten



Fallbeispiel: Intel x86-x32 Speicherarchitektur (2)

Begriffe

- SD – Segment Descriptor
 - für Segmente CS, DS, SS, ES, (FS), (GS) einzeln setzbar
 - Segmentauswahl direkt in der jeweiligen Instruktion
- LDTR/GDTR – Local/Global Descriptor Table Register
 - Verweis auf prozesslokale oder globale Segmenttabelle
- PD – Page Directory
- PT – Page Table
- historisch gewachsene Architektur aus vielen inkrementellen Erweiterungsschritten

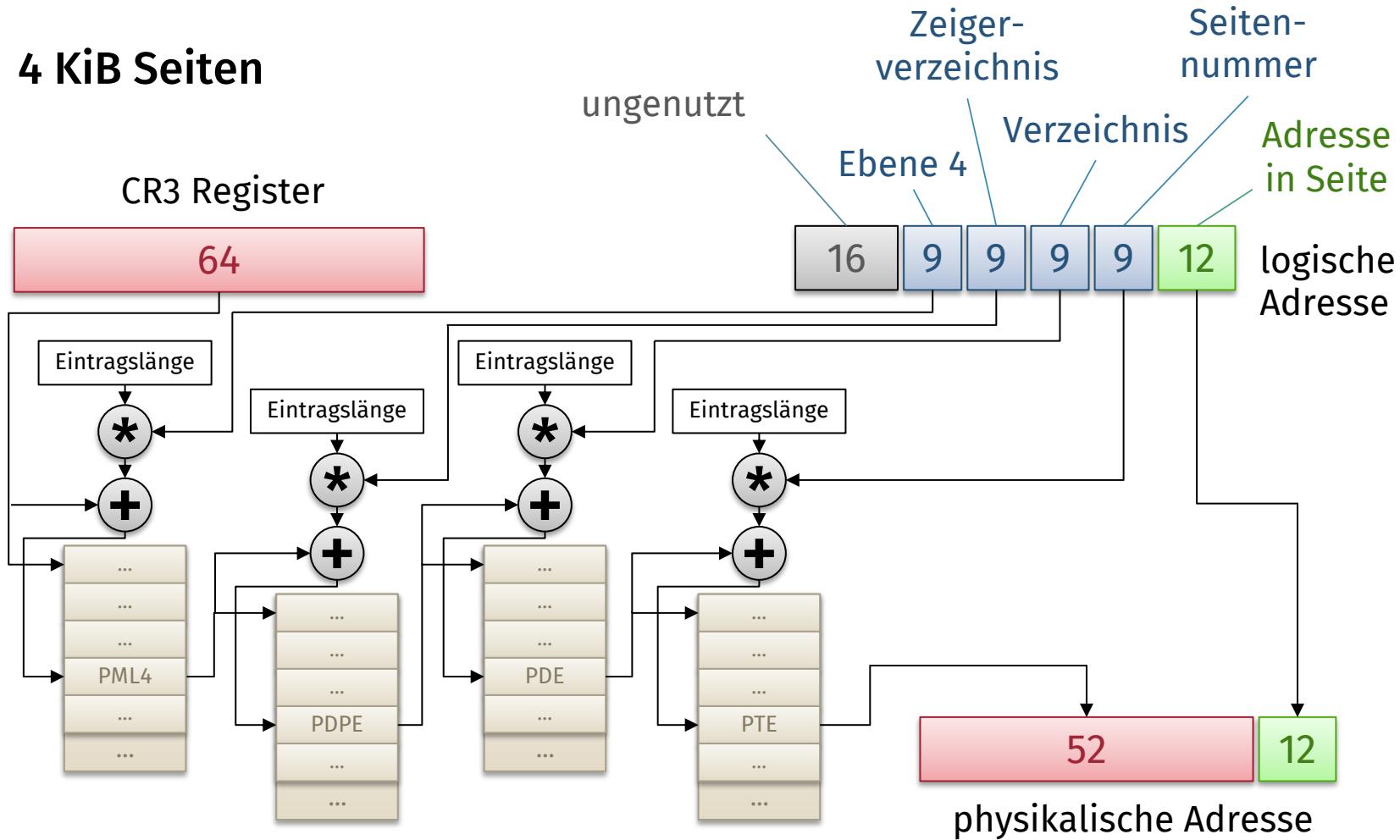
Fallbeispiel: Intel x86-x32 Speicherarchitektur (3)

Eigenschaften

- 1024 Einträge pro PD oder PT
- Segmentierung völlig unabhängig von Seitenadressierung
 - lineare Adresse als Zwischenstufe
 - sehr komplizierte Mechanik

Fallbeispiel: Intel x86-x64 Speicherarchitektur

4 KiB Seiten



Fallbeispiel: Intel x86-x64 Speicherarchitektur (2)

Begriffe

- PML4 – Page Mapping Level 4
- PDP – Page Directory Pointer
- PD – Page Directory
- PT – Page Table

Eigenschaften

- nur 48 Bit tatsächlich genutzt (momentan)
- **keine Segmentierung mehr**
 - alles in Tabellen geregelt
 - Segmente auf Ebene der PDP oder PML4 realisierbar

Fallbeispiel: Intel x86-x64 Speicherarchitektur (3)

Eigenschaften (fortges.)

- 512 Einträge pro Seite auf jeder Indirektionsstufe
 - intern volle 64 Bit Adresse vorhanden
 - für spätere Erweiterungen
- Seiten auch in Größe 2 MiB und 4 MiB verfügbar
 - wird durch spezielle Bits auf Ebene PD angezeigt
 - PT-Ebene entfällt dann

Virtueller Speicher

Entkopplung vom verfügbaren Hauptspeicher

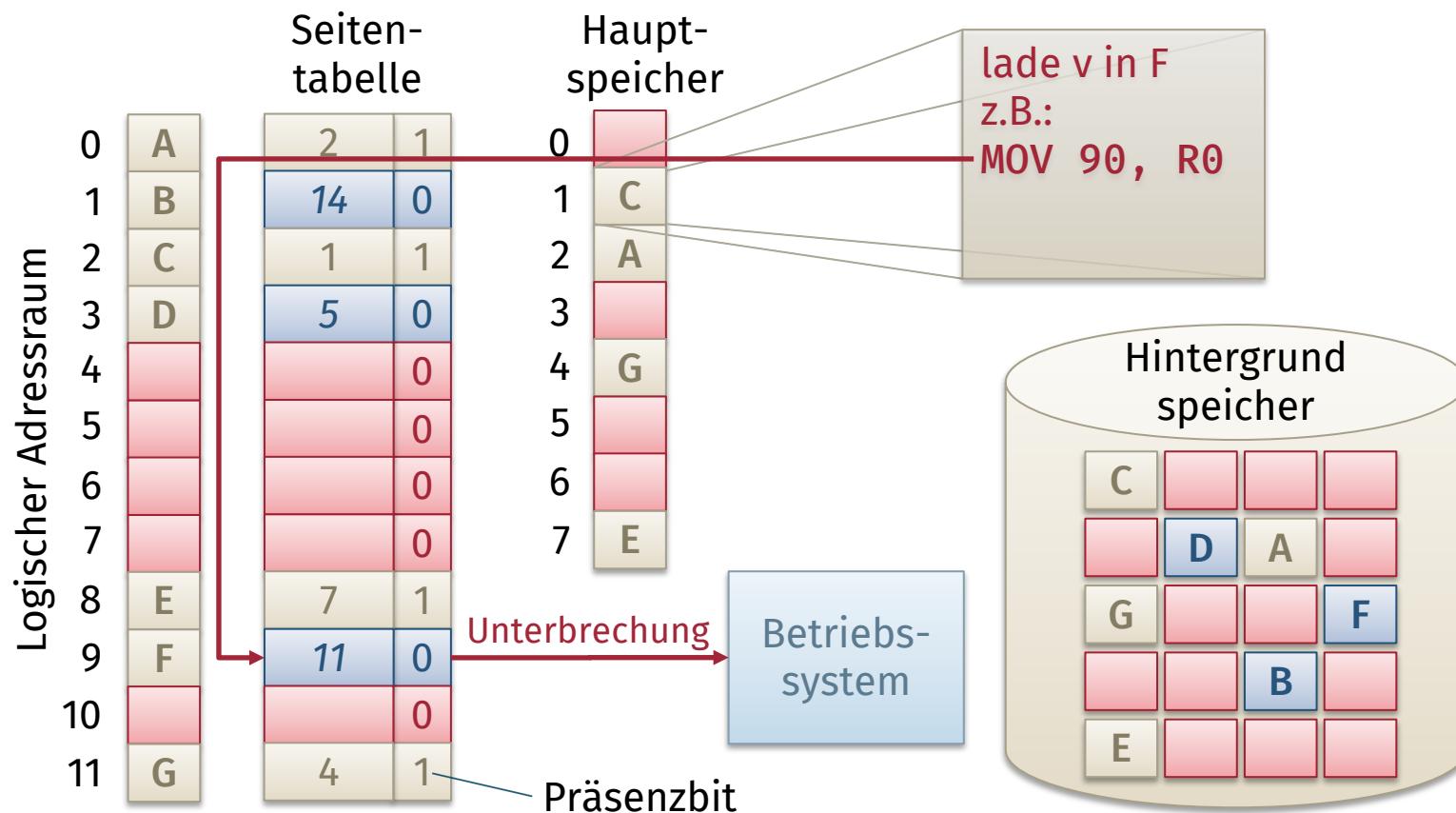
- Prozesse benötigen nicht alle Speicherstellen gleich häufig
 - bestimmte Befehle werden **selten oder gar nicht** benutzt (z.B. Fehlerbehandlungen)
 - bestimmte Datenstrukturen werden **nicht voll belegt**
- Speicherbedarf **höher als verfügbarer Hauptspeicher**

Idee

- **Vortäuschen eines beliebig großen Hauptspeichers**
 - **Einblenden** benötigter Speicherbereiche
 - **Abfangen** von Zugriffen auf nicht eingeblendete Bereiche
 - **Bereitstellen** der benötigten Bereiche auf Anforderung
 - **Auslagern** nicht benötigter Bereiche (**Swapping**)

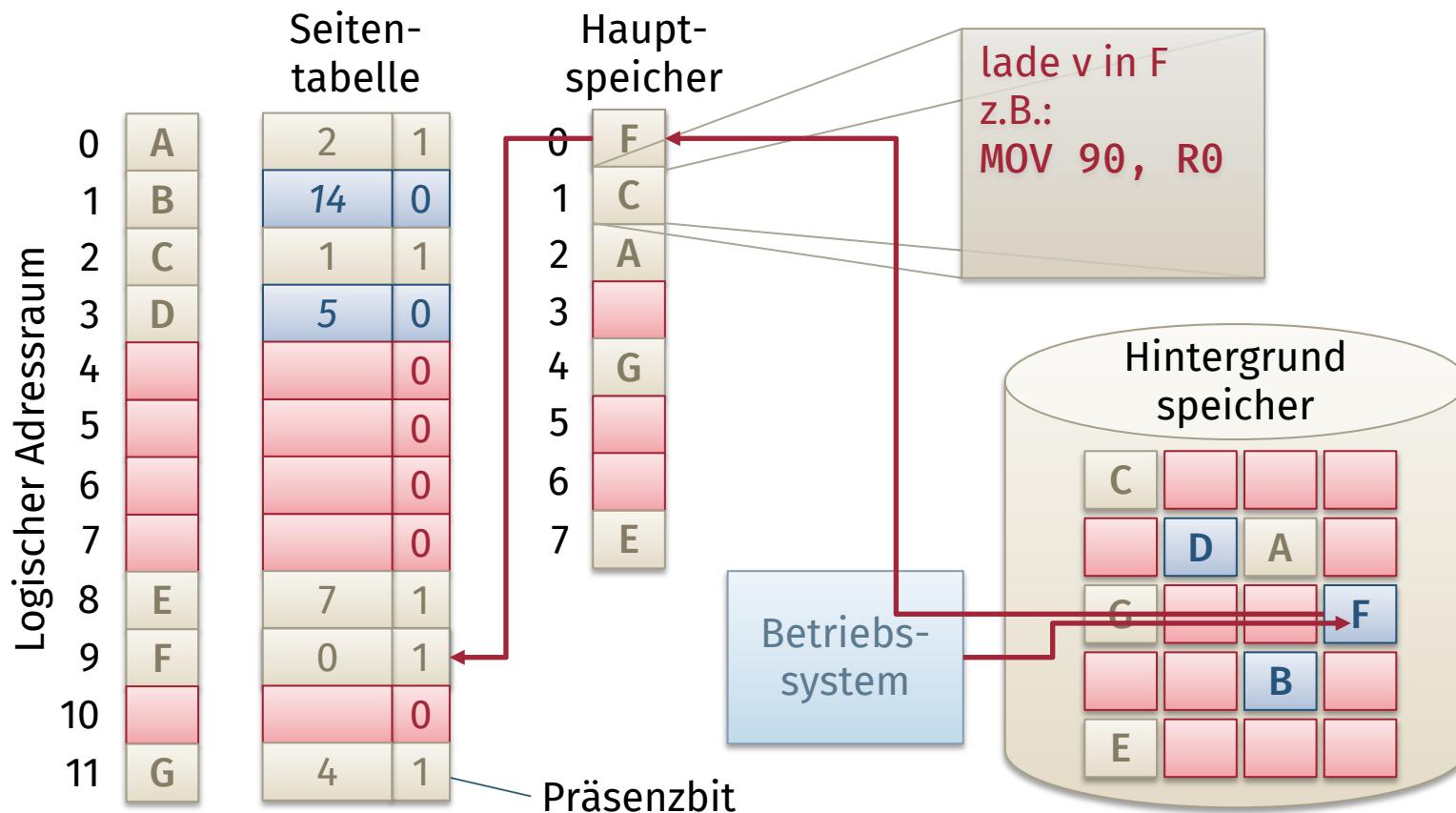
Virtueller Speicher (2)

Demand Paging – Seiteneinlagerung bei Bedarf



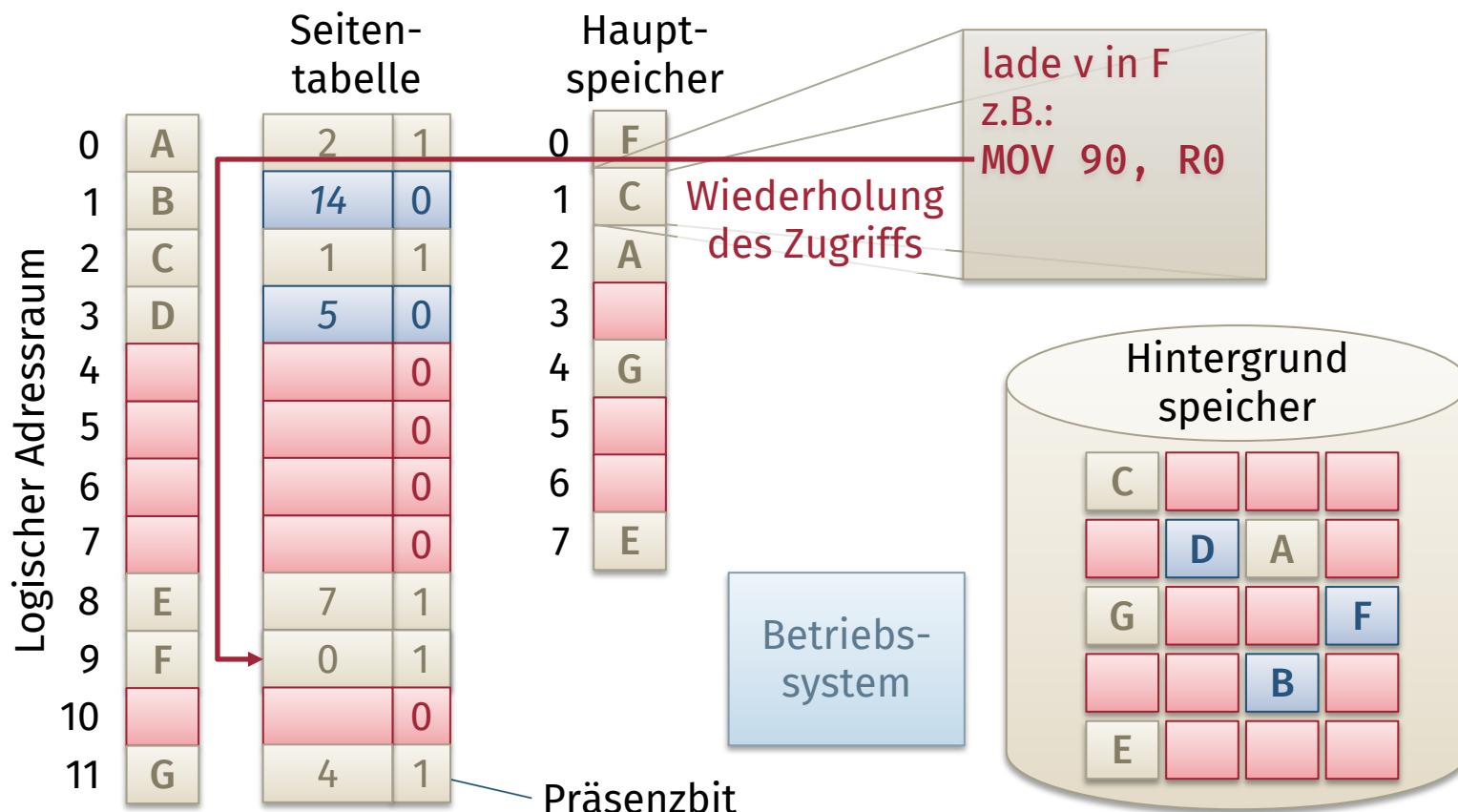
Virtueller Speicher (3)

Demand Paging – Seiteneinlagerung bei Bedarf



Virtueller Speicher (4)

Demand Paging – Seiteneinlagerung bei Bedarf



Demand Paging (6)

Performance von Demand Paging

■ keine Seitenfehler

- effektive Zugriffszeit zwischen 10 und 200 ns

■ mit Seitenfehler

- p sei Wahrscheinlichkeit für Seitenfehler, p nahe Null

- Annahmen:

- 25 ms zum Einlagern einer Seite vom Hintergrundspeicher
 - 9 ms Latenz, 15 ms Positionierzeit, 1 ms Übertragungszeit
 - 100 ns normale Zugriffszeit

- effektive Zugriffszeit:

$$(1 - p) * 100 \text{ ns} + p * 25.000.000 \text{ ns} = (100 + 24.999.900 * p) \text{ ns}$$

→ Seitenfehler müssen so gering wie möglich gehalten werden!



Bild von Mike by Pexels

(Grundlagen der) Betriebssysteme | G.5



Franz J. Hauck | Institut für Verteilte Systeme, Univ. Ulm

Inhaltsüberblick

Speicherverwaltung

- Phänomene
- Speichervergabe
 - statische und dynamische Zuteilung, Vergabestrategien
- Mehrprozessbetrieb
 - Segmentierung, Kompaktieren
 - Seitenadressierung
 - TLB
- Virtueller Speicher
 - Seitenersetzungsstrategien, FIFO, B₀, LRU, 2nd-Chance
 - Mehrprozessbetrieb
- Systemaufrufe

Seitenersetzung

Was tun, wenn keine freie Kachel vorhanden?

- eine Kachel verdrängen, um Platz für neue Kachel zu schaffen
- verdrängte Kachel
 - Dirty Bit = 0: Seite noch im Hintergrundspeicher
 - Dirty Bit = 1: Seite muss erst ausgelagert werden

Ablauf

- Seitenfehler (Page Fault): Unterbrechung
- Verdrängung einer Kachel, falls keine Kachel frei
 - eventuell mit Auslagerung
- Einlagern der benötigten Kachel
- Wiederholung des Zugriffs

Seitenersetzung (2)

Problem

- ❖ Welche Kachel soll ausgewählt werden?

Ansatz

- Einsatz von Seitenersetzungsstrategien zur Auswahl

Seitenersetzung (3)

Bewertung der Wirkung von Ersetzungsstrategien

■ Einsatz von Referenzfolgen

- Folge von Seitennummern, die das Speicherzugriffsverhalten eines Prozesses abbildet

■ Ermittlung von Referenzfolgen

- z.B. durch Aufzeichnung der zugegriffenen Adressen
- Reduktion der aufgezeichneten Sequenz auf Seitennummern
- Reduktion von sequentiellen Zugriffen auf gleiche Seite
 - z.B. Folge $1, 1, 2, 2, 3, 4, 4, 1$ wird zu $1, 2, 3, 4, 1$

■ Beispielreferenzfolge: $1, 2, 3, 4, 1, 2, 5, 1, 2, 3, 4, 5$

First-In First-Out, FIFO

Älteste Kachel wird verdrängt

- notwendige Kontrollzustände:
 - Alter bzw. Einlagerungszeitpunkt für jede Kachel

Ablauf: 9 Einlagerungen													
Referenzfolge		1	2	3	4	1	2	5	1	2	3	4	5
Hauptspeicher	Kachel 1	1	1	1	4	4	4	5	5	5	5	5	5
	Kachel 2		2	2	2	1	1	1	1	1	3	3	3
	Kachel 3			3	3	3	2	2	2	2	2	4	4
Kontroll-zustände (Alter)	Kachel 1	0	1	2	0	1	2	0	1	2	3	4	5
	Kachel 2	>	0	1	2	0	1	2	3	4	0	1	2
	Kachel 3	>	>	0	1	2	0	1	2	3	4	0	1

First-In First-Out, FIFO (2)

Größerer Hauptspeicher – 4 Kacheln

Ablauf: 10 Einlagerungen													
Referenzfolge		1	2	3	4	1	2	5	1	2	3	4	5
Hauptspeicher	Kachel 1	1	1	1	1	1	1	5	5	5	5	4	4
	Kachel 2		2	2	2	2	2	2	1	1	1	1	5
	Kachel 3			3	3	3	3	3	3	2	2	2	2
	Kachel 4				4	4	4	4	4	4	3	3	3
Kontrollzustände (Alter)	Kachel 1	0	1	2	3	4	5	0	1	2	3	0	1
	Kachel 2	>	0	1	2	3	4	5	0	1	2	3	0
	Kachel 3	>	>	0	1	2	3	4	5	0	1	2	3
	Kachel 4	>	>	>	0	1	2	3	4	5	0	1	2

First-In, First-Out, FIFO (3)

FIFO-Anomalie

- auch Belady's Anomalie (1969)
- ❖ Größerer Hauptspeicher kann zu mehr Einlagerungen führen!
 - muss aber nicht

Optimale Strategie

Verdrängt Seite mit größtem Vorwärtsabstand

- d.h. Seite, die am längsten nicht verwendet werden wird

Ablauf: 7 Einlagerungen													
Referenzfolge		1	2	3	4	1	2	5	1	2	3	4	5
Hauptspeicher	Kachel 1	1	1	1	1	1	1	1	1	1	3	4	4
	Kachel 2		2	2	2	2	2	2	2	2	2	2	2
	Kachel 3			3	4	4	4	5	5	5	5	5	5
Kontrollzustände (Vorwärtsabstand)	Kachel 1	4	3	2	1	3	2	1	>	>	>	>	>
	Kachel 2	>	4	3	2	1	3	2	1	>	>	>	>
	Kachel 3	>	>	7	7	6	5	5	4	3	2	1	>

Optimale Strategie (2)

Größerer Hauptspeicher – 4 Kacheln

Ablauf: 6 Einlagerungen													
Referenzfolge		1	2	3	4	1	2	5	1	2	3	4	5
Hauptspeicher	Kachel 1	1	1	1	1	1	1	1	1	1	1	4	4
	Kachel 2		2	2	2	2	2	2	2	2	2	1	1
	Kachel 3			3	3	3	3	3	3	3	3	2	2
	Kachel 4				4	4	4	5	5	5	5	5	5
Kontroll-zustände (Vorwärts-abstand)	Kachel 1	4	3	2	1	3	2	1	>	>	>	>	>
	Kachel 2	>	4	3	2	1	3	2	1	>	>	>	>
	Kachel 3	>	>	7	6	5	4	3	2	1	>	>	>
	Kachel 4	>	>	>	7	6	5	5	4	3	2	1	>

Optimale Strategie (3)

Eigenschaften

- Strategie B_0 (OPT oder MIN) ist optimal bei fester Kachelmenge
- zeigt keine Anomalie
 - Beispiel ist kein Beweis, stimmt trotzdem

Problem

- Referenzfolge muss vorher bekannt sein
 - in der Praxis kaum realisierbar
- Einsatz meist nur zum theoretischen Vergleich von Strategien
- Suche nach Strategien, die optimale Strategie annähern
 - z.B. Least Recently Used, LRU



Bild von Mike by Pexels

(Grundlagen der) Betriebssysteme | G.6



Franz J. Hauck | Institut für Verteilte Systeme, Univ. Ulm

Inhaltsüberblick

Speicherverwaltung

- Phänomene
- Speichervergabe
 - statische und dynamische Zuteilung, Vergabestrategien
- Mehrprozessbetrieb
 - Segmentierung, Kompaktieren
 - Seitenadressierung
 - TLB
- Virtueller Speicher
 - Seitenersetzungsstrategien, FIFO, B₀, LRU, 2nd-Chance
 - Mehrprozessbetrieb
- Systemaufrufe

Least Recently Used, LRU

Verdrängt Seite mit größtem Rückwärtsabstand

- d.h. Seite, die am längsten nicht mehr zugegriffen wurde

Ablauf: 10 Einlagerungen													
Referenzfolge		1	2	3	4	1	2	5	1	2	3	4	5
Hauptspeicher	Kachel 1	1	1	1	4	4	4	5	5	5	3	3	3
	Kachel 2		2	2	2	1	1	1	1	1	1	4	4
	Kachel 3			3	3	3	2	2	2	2	2	2	5
Kontrollzustände (Rückwärtsabstand)	Kachel 1	0	1	2	0	1	2	0	1	2	0	1	2
	Kachel 2	>	0	1	2	0	1	2	0	1	2	0	1
	Kachel 3	>	>	0	1	2	0	1	2	0	1	2	0

Least Recently Used, LRU (2)

Größerer Hauptspeicher – 4 Kacheln

Ablauf: 8 Einlagerungen		1	2	3	4	1	2	5	1	2	3	4	5
Hauptspeicher	Referenzfolge	1	1	1	1	1	1	1	1	1	1	1	5
	Kachel 1	1	1	1	1	1	1	1	1	1	1	1	5
	Kachel 2		2	2	2	2	2	2	2	2	2	2	2
	Kachel 3			3	3	3	3	5	5	5	5	4	4
Kontrollzustände (Rückwärtsabstand)	Kachel 4				4	4	4	4	4	4	3	3	3
	Kachel 1	0	1	2	3	0	1	2	0	1	2	3	0
	Kachel 2	>	0	1	2	3	0	1	2	0	1	2	3
	Kachel 3	>	>	0	1	2	3	0	1	2	3	0	1
Kontrollzustände (Rückwärtsabstand)	Kachel 4	>	>	>	0	1	2	3	4	5	0	1	2

Least Recently Used, LRU (3)

Keine Anomalie bei LRU

- allgemein keine Anomalie bei Stack-Algorithmen
- Stack-Algorithmen
 - $S_{n,t}$:= {eingelagerter Seiten bei n Kacheln zum Zeitpunkt t }
 - zu jedem Zeitpunkt t gilt: $S_{n,t} \subseteq S_{n+1,t}$
 - LRU: $S_{n,t} = \{n$ zuletzt benutzten Seiten eingelagert}
 - B0: $S_{n,t} = \{n$ bereits benutzten Seiten, die als nächstes zugegriffen werden}

Problem Implementierung

- ohne Hardwareunterstützung unmöglich
 - jeder Speicherzugriff zu berücksichtigen

Least Recently Used, LRU (4)

Problem Implementierung (fortges.)

- Variante: Hardwareunterstützung durch Zähler
 - CPU besitzt einen **Zähler**
 - inkrementiert mit jedem Speicherzugriff
 - CPU schreibt **Zählerstand** in den Seitendeskriptor der zugegriffenen Seite
 - Auswahl der Seite mit dem kleinsten Zählerstand
 - am längsten nicht benutzt
- aufwändige, unpraktische Implementierung
 - weitere **implizite** Speicherzugriffe
 - Suche der geeigneten Kachel erfordert lineare **Suche** durch alle Seitendeskriptoren

Second Chance, Clock

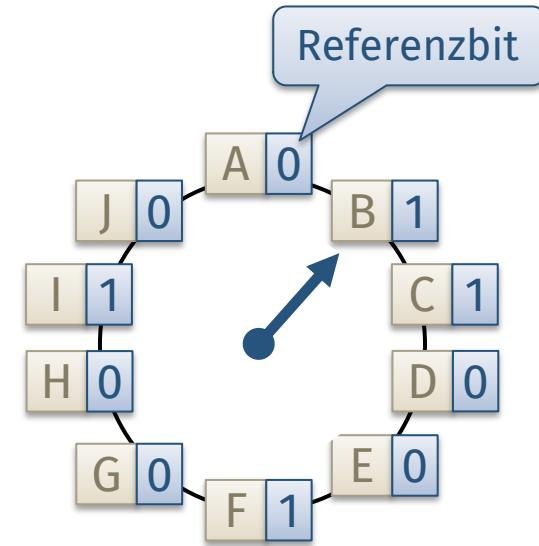
Annäherung von LRU

- Einsatz von Referenzbits (*Used Bit*)
 - weiteres Bit im Seitendeskriptor
 - wird von MMU bei Zugriff auf 1 gesetzt
 - Beispiel Intel: Access Bit
- Algorithmus sucht Seiten, die länger nicht zugegriffen wurden
 - Idee:
 - Rücksetzen des Referenzbits
 - Warten für gewisse Zeit Δt
 - Überprüfen des Referenzbits
 - bei 0 nicht zugegriffen innerhalb Δt
 - bei 1 dagegen schon

Second Chance, Clock (2)

Konzeptionelle Anordnung der Seiten im Kreis

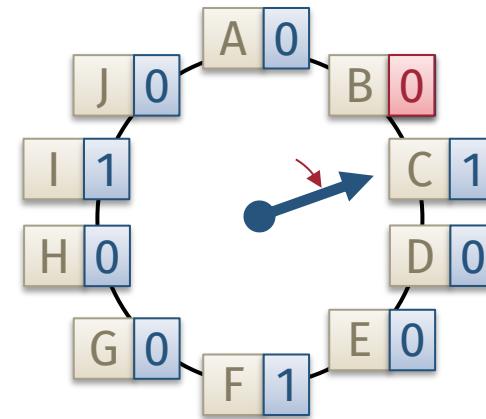
- wie Uhr
- Zeiger zeigt auf eine der Seiten
- Einzelschritt des Algorithmus
 - Überprüfen des Referenzbits am Zeigerende
 - bei 0: Seite wird ersetzt
 - bei 1: zweite Chance, d.h. Referenzbit wird auf 0 gesetzt
 - Weiterschaltung des Zeigers
- **Variante:** Algorithmus läuft solange bis Seite verdrängt
 - z.B. bei Bedarf



Second Chance, Clock (3)

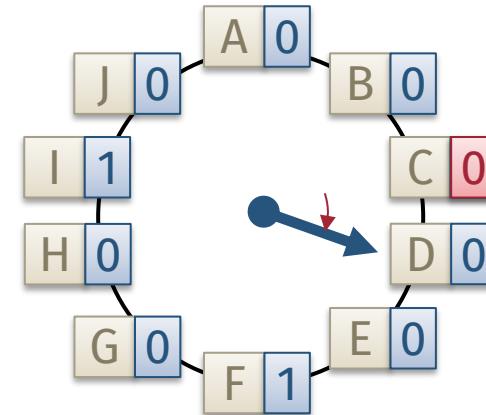
Beispielablauf: Schritt 1

- Überprüfen des Referenzbits bei B
 - ist 1: Rücksetzen auf 0
- Weiterschalten des Zeigers auf C



Beispielablauf: Schritt 2

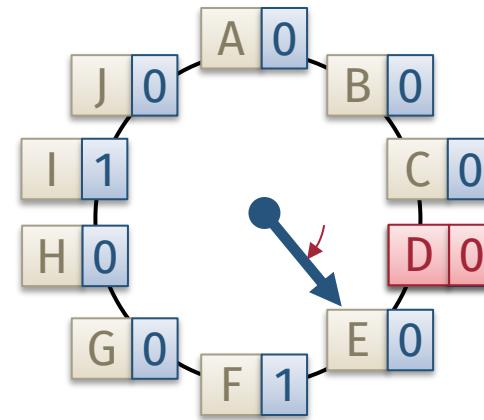
- Überprüfen des Referenzbits bei C
 - ist 1: Rücksetzen auf 0
- Weiterschalten des Zeigers auf D



Second Chance, Clock (4)

Beispielablauf: Schritt 3

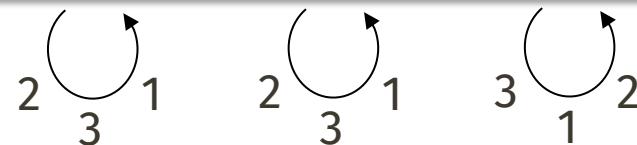
- Überprüfen des Referenzbits bei D
 - ist 0: Kandidat für Auslagerung
- Weiterschalten des Zeigers auf E
- Seite D kann ersetzt werden
 - Algorithmus stoppt



Second Chance, Clock (5)

Verdrängt Seiten gemäß Clock-Algorithmus

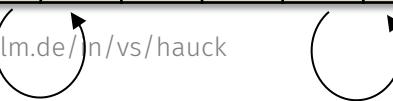
Ablauf: 9 Einlagerungen													
Referenzfolge		1	2	3	4	1	2	5	1	2	3	4	5
Hauptspeicher		Kachel 1	1	1	1	4	4	4	5	5	5	5	5
		Kachel 2		2	2	2	1	1	1	1	1	3	3
		Kachel 3			3	3	3	2	2	2	2	2	4
Kontrollzustände (Referenzbits, Umlaufzeiger)		Kachel 1	1	1	1	1	1	1	1	1	0	0	1
		Kachel 2	0	1	1	0	1	1	0	1	1	1	1
		Kachel 3	0	0	1	0	0	1	0	0	1	0	1
		Zeiger	2	3	1	2	3	1	2	2	2	3	1



Second Chance, Clock (6)

Größerer Hauptspeicher – 4 Kacheln

Ablauf: 10 Einlagerungen													
Referenzfolge		1	2	3	4	1	2	5	1	2	3	4	5
Hauptspeicher		Kachel 1	1	1	1	1	1	5	5	5	5	4	4
		Kachel 2		2	2	2	2	2	1	1	1	1	5
		Kachel 3			3	3	3	3	5	2	2	2	2
		Kachel 4				4	4	4	4	4	3	3	3
Kontrollzustände (Referenzbits, Umlaufzeiger)		Kachel 1	1	1	1	1	1	1	1	1	1	1	1
		Kachel 2	0	1	1	1	1	0	1	1	1	0	1
		Kachel 3	0	0	1	1	1	0	0	1	1	0	0
		Kachel 4	0	0	0	1	1	1	0	0	0	1	0
		Zeiger	2	3	4	1	1	1	2	3	4	1	2



Second Chance, Clock (7)

Degeneration zu FIFO

- Annahme: **alle** Seiten haben genug Zeit Referenzbit zu setzen
 - Zeiger läuft **einmal um** und setzt alle Referenzbits auf 0
 - Algorithmus wählt Seite aus, auf die Zeiger zeigt
 - Zeiger steht dann eine Seite weiter
- Algorithmus wählt **eine Seite nach der anderen**, wie FIFO

Anomalie

- Second Chance ist kein Stack-Algorithmus
- zeigt Anomalie

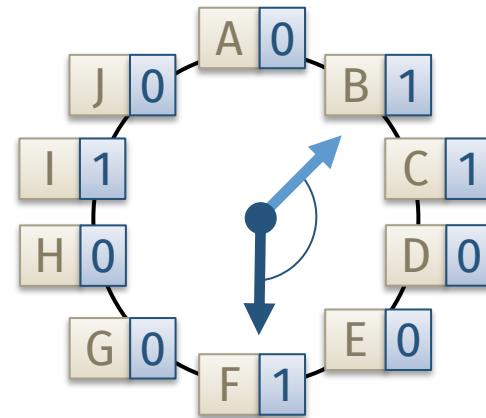
Second Chance, Clock (8)

Problem große Hauptspeicher

- viele Kacheln
 - Zeiger kommt erst spät bei selber Seite vorbei
 - Gefahr der Degeneration zu FIFO

Lösung durch gekoppelte Zeiger

- erster Zeiger **löscht** Referenzbit
- zweiter Zeiger **prüft** Referenzbit auf 0 und wählt Seite zur Auslagerung aus



Second Chance, Clock (8)

Variante

- zusätzlich Berücksichtigung des Modifikationsbit (*dirty bit*)
- Einteilung von Speicherzugriffen in vier Klassen:
 - RB = Referenzbit, DB = Modifikationsbit
 - RB = 0, DB = 0 Klasse 1
 - RB = 0, DB = 1 Klasse 2
 - RB = 1, DB = 0 Klasse 3
 - RB = 1, DB = 1 Klasse 4
- Suche nach zu ersetzender Kachel in der niedrigsten nicht-leeren Klasse (Einsatz in MacOS)



Bild von Mike by Pexels

(Grundlagen der) Betriebssysteme | G.7



Franz J. Hauck | Institut für Verteilte Systeme, Univ. Ulm

Inhaltsüberblick

Speicherverwaltung

- Phänomene
- Speichervergabe
 - statische und dynamische Zuteilung, Vergabestrategien
- Mehrprozessbetrieb
 - Segmentierung, Kompaktieren
 - Seitenadressierung
 - TLB
- Virtueller Speicher
 - Seitenersetzungsstrategien, FIFO, B₀, LRU, 2nd-Chance
 - Mehrprozessbetrieb
- Systemaufrufe

Freiseitenpuffer

Liste freier Kacheln

- Ersetzung von Seiten ineffizient
 - doppelte Transportkosten: Auslagerung und Einlagerung
- Idee: Vorhalten freier Seiten
 - Auslagerung geschieht „im Voraus“
 - Ersetzungszeit besteht im Wesentlichen nur noch aus der Einlagerungszeit

Freiseitenpuffer (2)

Optimierung

- Beibehalten der Seitenzuordnung nach der Auslagerung
 - bis zur Einlagerung einer neuen Seite
 - Präsenzbit auf 0
 - Zugriff führt zu Seitenfehler
 - bei Zugriff
 - Austragung aus Freiseitenpuffer
 - erneute Zuordnung zum Prozess ohne Einlagerung
 - Präsenzbit auf 1
 - erneuter Zugriff

Freiseitenpuffer (3)

Problem von 2nd Chance

- Umlaufgeschwindigkeit des Zeigers sehr **unregelmäßig**
 - abhängig von Einlagerungsbedarf
 - **ungleiche Chance** für Seiten, das Referenzbit zu setzen

Optimierung mit Freiseitenpuffer

- Zeiger rotiert mit **konstanter Geschwindigkeit** im Hintergrund
 - typischerweise dynamisch **angepasst** an Größe des Freiseitenpuffers
 - **genug freie Seiten:** Zeiger **steht**
 - **wenig freie Seiten:** Zeiger **rotiert schnell**

Mehrprozessbetrieb

Problem:

- Verteilung der Kacheln unter mehreren Prozessen

Begrenzungen

- **maximale Kachelmenge**
 - begrenzt durch Anzahl verfügbarer physikalischer Kacheln
- **minimale Kachelmenge**
 - abhängig von der Prozessorarchitektur
 - mindestens die Anzahl von Kacheln, die theoretisch maximal bei einem Maschinenbefehl benötigt werden

Mehrprozessbetrieb (2)

Minimale Kachelmenge

- Beispiel Spielprozessor
 - Kachelgröße: 16 Bytes
 - längster Maschinenbefehl: 2 Bytes
 - größte zugegriffene Datenstruktur: 1 Byte
 - minimale Kachelmenge:
 - 2 für Instruktion
 - 1 für zugegriffene Daten
 - insgesamt 3 Kacheln

Mehrprozessbetrieb (3)

Lokale und globale Ersetzungsstrategien

- **lokal:** Prozess ersetzt nur immer seine eigenen Seiten
 - statische Zuteilung von Kacheln pro Prozess
 - Seitenfehler-Verhalten liegt nur in der Verantwortung des jeweiligen Prozesses

- **global:** Prozess ersetzt auch Seiten anderer Prozesse
 - dynamisches Verhalten der Prozesse kann berücksichtigt werden
 - effizienter, da ungenutzte Kacheln von anderen Prozessen verwendet werden können

Mehrprozessbetrieb (4)

Globale Strategie

- Seitenersetzung **unabhängig** von Prozesszugehörigkeit
 - heute überwiegend im Einsatz

Lokale Strategie

- erfordert **Zuordnung** einer Kachelmenge zu Prozess
- Varianten:
 - statisch: Hauptspeicher / Anzahl der Prozesse
 - statisch: proportional zur Prozessgröße
 - z.B. nach Segmentgrößen
 - dynamisch: proportional zum Kachelbedarf
 - **Working-Set-Ansatz**

Seitenflattern (Thrashing)

Ausgelagerte Seite wird gleich wieder angesprochen

- Seitenersetzungszeit wesentlich größer als Ausführungszeit
 - Prozess wird spürbar langsam

Mögliche Ursachen

- Prozess ist nahe am Seitenminimum
- zu viele Prozesse gleichzeitig im System
- schlechte Ersetzungsstrategie

Seitenflattern (Thrashing) (2)

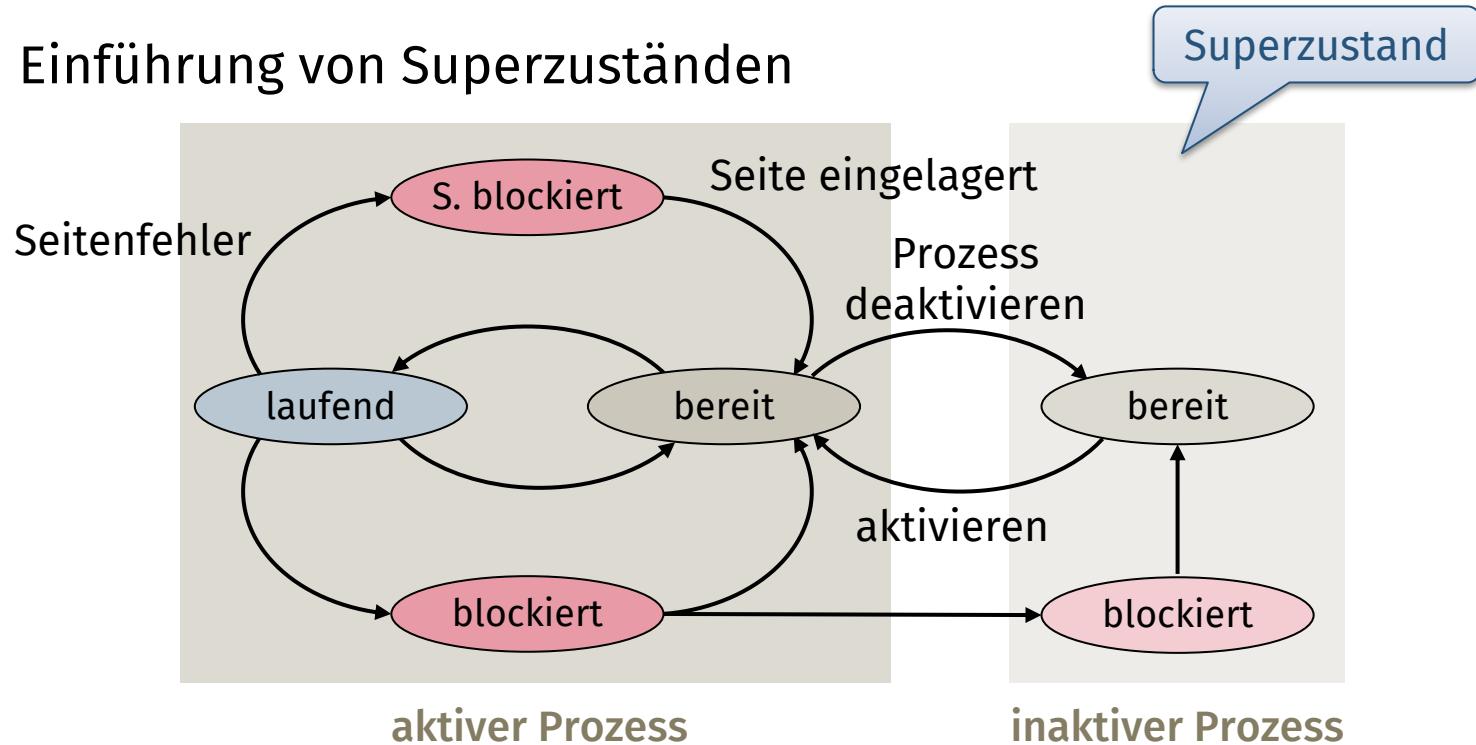
Lösungsansätze

- lokale Ersetzungsstrategie
 - behebt Thrashing zwischen Prozessen
- Zuteilung einer genügend großen Zahl von Kacheln
 - behebt Thrashing innerhalb der Prozess-Seiten
 - Begrenzung der Prozessanzahl

Deaktivieren von Prozessen

Reduktion der Anzahl der Prozesse

Einführung von Superzuständen



■ inaktiver Prozess benötigt keine Kacheln

- ist vollständig ausgelagert (swapped out)



Bild von Mike by Pexels

(Grundlagen der) Betriebssysteme | G.8



Franz J. Hauck | Institut für Verteilte Systeme, Univ. Ulm

Inhaltsüberblick

Speicherverwaltung

■ Phänomene

■ Speichervergabe

- statische und dynamische Zuteilung, Vergabestrategien

■ Mehrprozessbetrieb

- Segmentierung, Kompaktieren
- Seitenadressierung
- TLB

■ Virtueller Speicher

- Seitenersetzungsstrategien, FIFO, B₀, LRU, 2nd-Chance
- Mehrprozessbetrieb

■ Systemaufrufe

Systemaufrufe

Segmentgrößen

■ Code-Segment

- normalerweise statische Größe

■ Datensegment

- `int err = brk(void *endaddr);`
- `void *newend = sbrk(intptr_t increment);`
- dynamisches Anpassen des Segmentendes

■ Stacksegment

- normalerweise dynamisch und automatisch angepasst

Systemaufrufe (2)

Einblenden von Dateien in den Speicher

■ Einblenden

- `void *newaddr = mmap(void *addr, size_t length,
int prot, int flags, int fd, off_t offset);`
 - `addr`: Startadresse im logischen Adressraum
 - `length`: Länge des Mappings
 - `prot flags`: Konfigurationsangaben
 - `fd`: geöffnete Datei
 - `offset`: ab welcher Stelle in der Datei soll eingeblendet werden

■ Ausblenden

- `int errno = munmap(void *addr, size_t length);`

Systemaufrufe (3)

Gemeinsamer Speicher

- Anlegen eines Shared Memory Objects
 - `int fd = shm_open(const char *name, int oflag, mode_t mode);`
 - `name`: globaler Name für das Objekt
 - `oflag`: ähnliche Bedeutung wie bei `open()`
 - `mode`: Berechtigungen
 - Filedeskriptor kann dann mit `mmap()` verwendet werden
- Ändern der Länge (wie bei Dateien)
 - `int errno = ftruncate(int fd, off_t length);`
- Löschen eines Shared Memory Objects
 - `int errno = shm_unlink(const char *name);`

Inhaltsüberblick

Speicherverwaltung

■ Phänomene

■ Speichervergabe

- statische und dynamische Zuteilung, Vergabestrategien

■ Mehrprozessbetrieb

- Segmentierung, Kompaktieren
- Seitenadressierung
- TLB

■ Virtueller Speicher

- Seitenersetzungsstrategien, FIFO, B₀, LRU, 2nd-Chance
- Mehrprozessbetrieb

■ Systemaufrufe