ulm university universität ulm

Grundlagen der Betriebssysteme

[CS2100]

Sommersemester 2014

Heiko Falk

Institut für Eingebettete Systeme/Echtzeitsysteme
Ingenieurwissenschaften und Informatik
Universität Ulm

ulm university universität ulm

Kapitel 6

Speicherverwaltung

Inhalte der Vorlesung

Inhalte des Kapitels (1)

6. Speicherverwaltung

- Einleitung
  - Phänomen der Speicherverwaltung
  - Speicherhierarchien
  - Kapazität von Speichern
- Speichervergabe
  - Statische und dynamische Speicherzuteilung
  - Freispeicherverwaltung (Bitvektoren, verkettete Listen)
  - Vergabestrategien
    - *First Fit*
    - *Next Fit*
    - *Best Fit*
    - *Worst Fit*
- ...

Inhalte des Kapitels (2)

6. Speicherverwaltung

- ...
- Mehrprogrammbetrieb
  - Segmentierung
  - Kompaktierung
  - Seitenadressierung *(Paging)*
    - *Memory Management Unit* für *Paging*
    - Segmentierung und Seitenadressierung
    - Mehrstufiges *Paging*
    - *Translation Look-Aside Buffer (TLBs)*
- ...

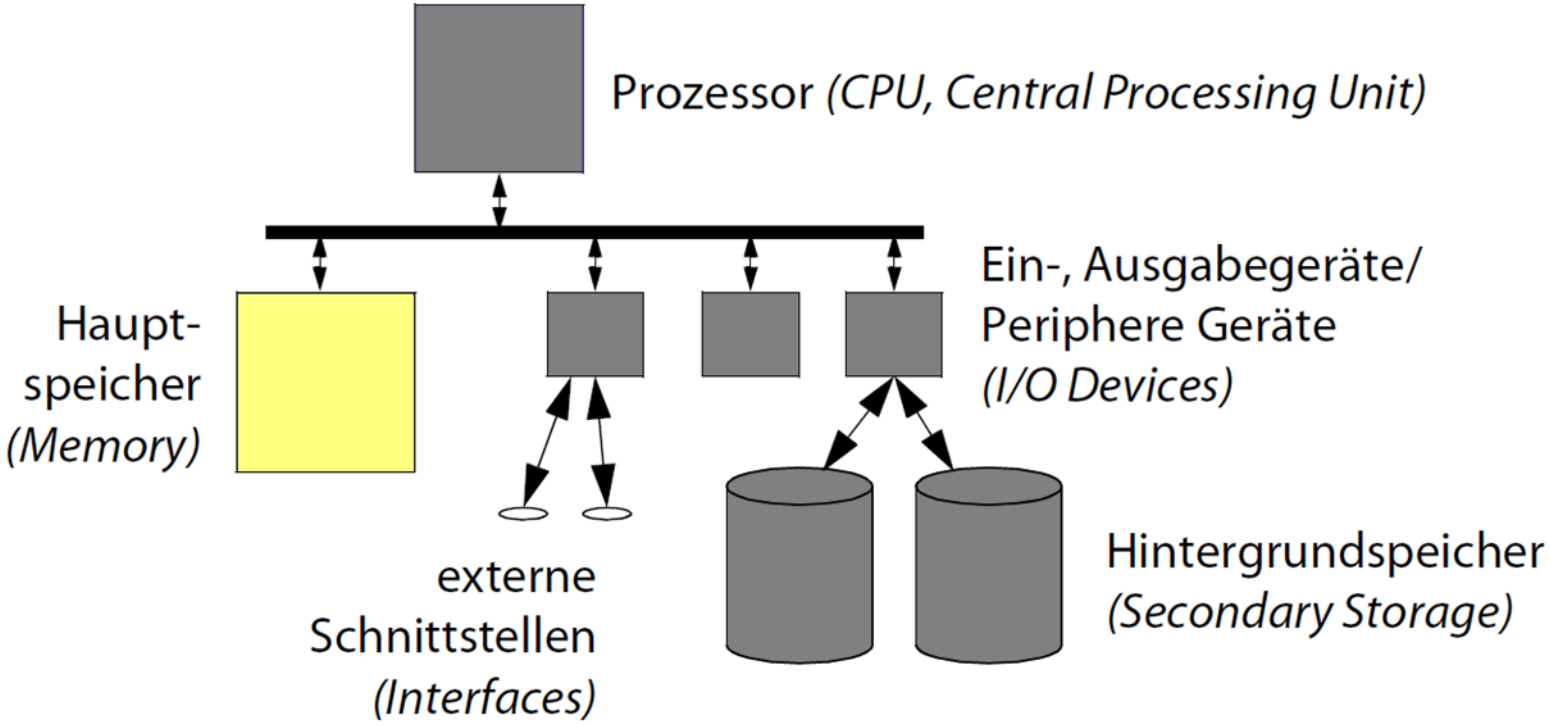
Inhalte des Kapitels (3)

6. Speicherverwaltung

- ...
- Virtueller Speicher
  - *Demand Paging*
  - Seiten-Ersetzungsstrategien
    - *First-In First-Out (FIFO)*
    - Optimale Ersetzungsstrategie
    - *Least Recently Used (LRU)*
    - *Second Chance (Clock)*
    - Freiseitenpuffer
  - Seitenadressierung und Mehrprogrammbetrieb

Einordnung

Betroffene physikalische Ressourcen



Phänomen der Speicherverwaltung (1)

Beispiel: Initialisierung großer Matrizen (geschrieben in C)

– Variante 1

```
#define DIM 10000

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;
}
```

Phänomen der Speicherverwaltung (2)

Beispiel: Initialisierung großer Matrizen (geschrieben in C)

– Variante 2

```
#define DIM 10000

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 sind vertauscht

Phänomen der Speicherverwaltung (3)

Messergebnisse (Intel Xeon 2,4 GHz, 2GB Hauptspeicher)

– Variante 1

User time = 0,26 sec; System time = 0,83 sec; Gesamtzeit = 1,09 sec

– Variante 2

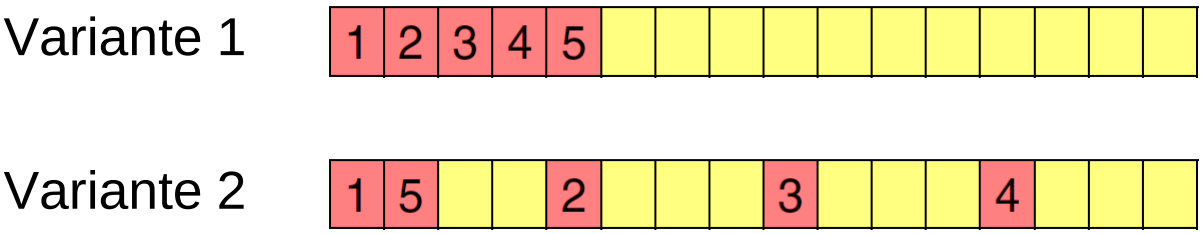
User time = 17 sec; System time = 0,81 sec; Gesamtzeit = 17,9 sec

Ursachen

– Variante 1 geht sequentiell durch den Speicher

– Variante 2 greift versetzt ständig auf den gesamten Speicherbereich zu

Beispiel: **`matrix[4][4]`** und die ersten fünf Zugriffe



Phänomen der Speicherverwaltung (4)

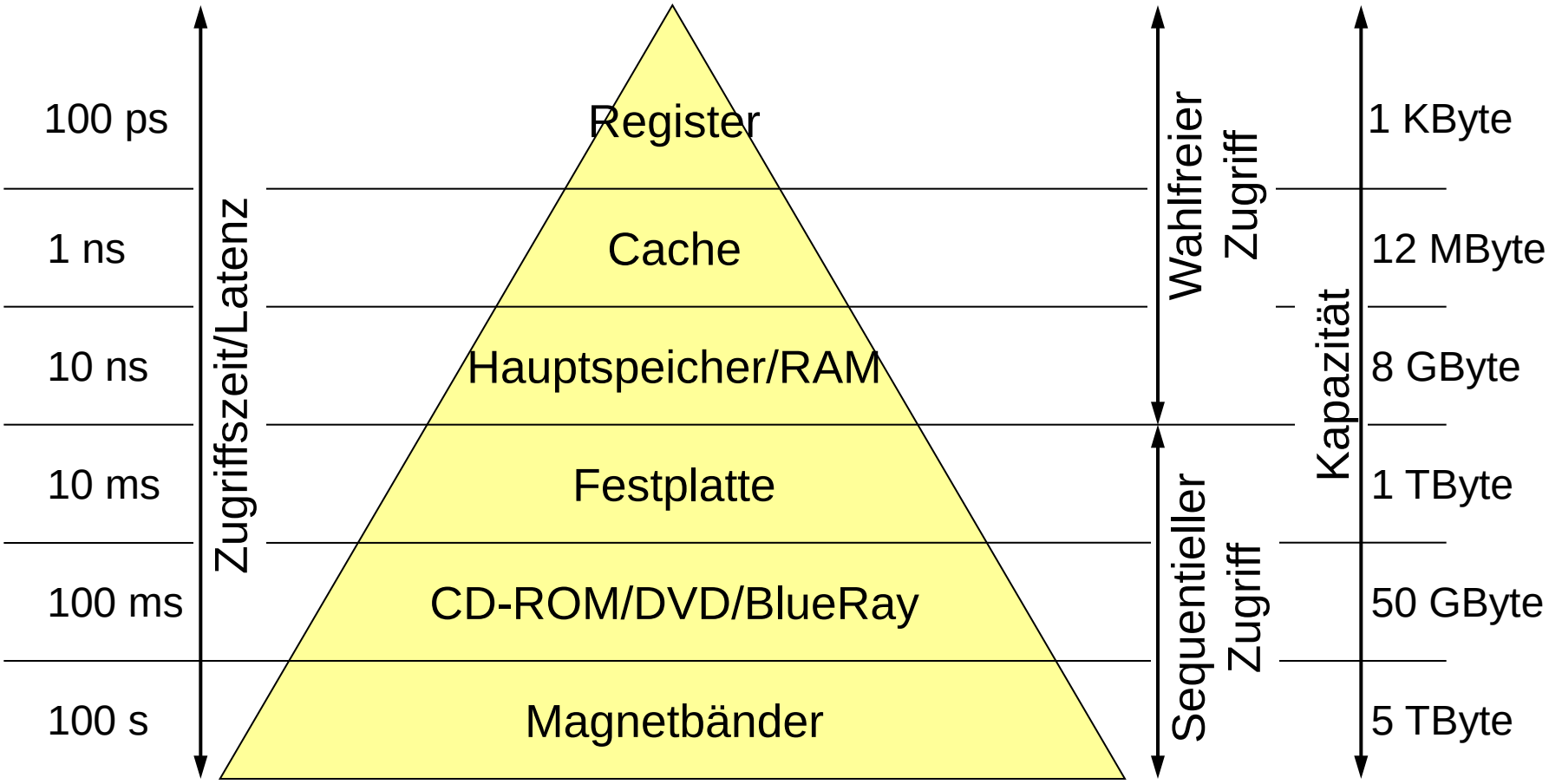
Ursachen

- Virtueller (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
- Virtueller Speicher
  - Möglicher Adressraum ist größer als realer Speicher
  - Auf Seitenbasis werden Teile des benötigten Speichers ein- und ausgelagert
  - Bei Variante 2 muss viel mehr Speicher ein- und ausgelagert werden

Speicher allgemein (1)

Speicherhierarchie

– Bis zu sechs Ebenen in modernen Systemen unterscheidbar



Speicher allgemein (2)

Messung der Kapazität von Speicherbausteinen in

- *Kilobyte (kByte, kB)*

  1 kByte = 1024 Bytes = 2^{10} Bytes
- *Megabyte (MByte, MB)*

  1 MByte = 1024 kBytes = 2^{20} Bytes = 1.048.576 Bytes
- *Gigabyte (GByte, GB)*

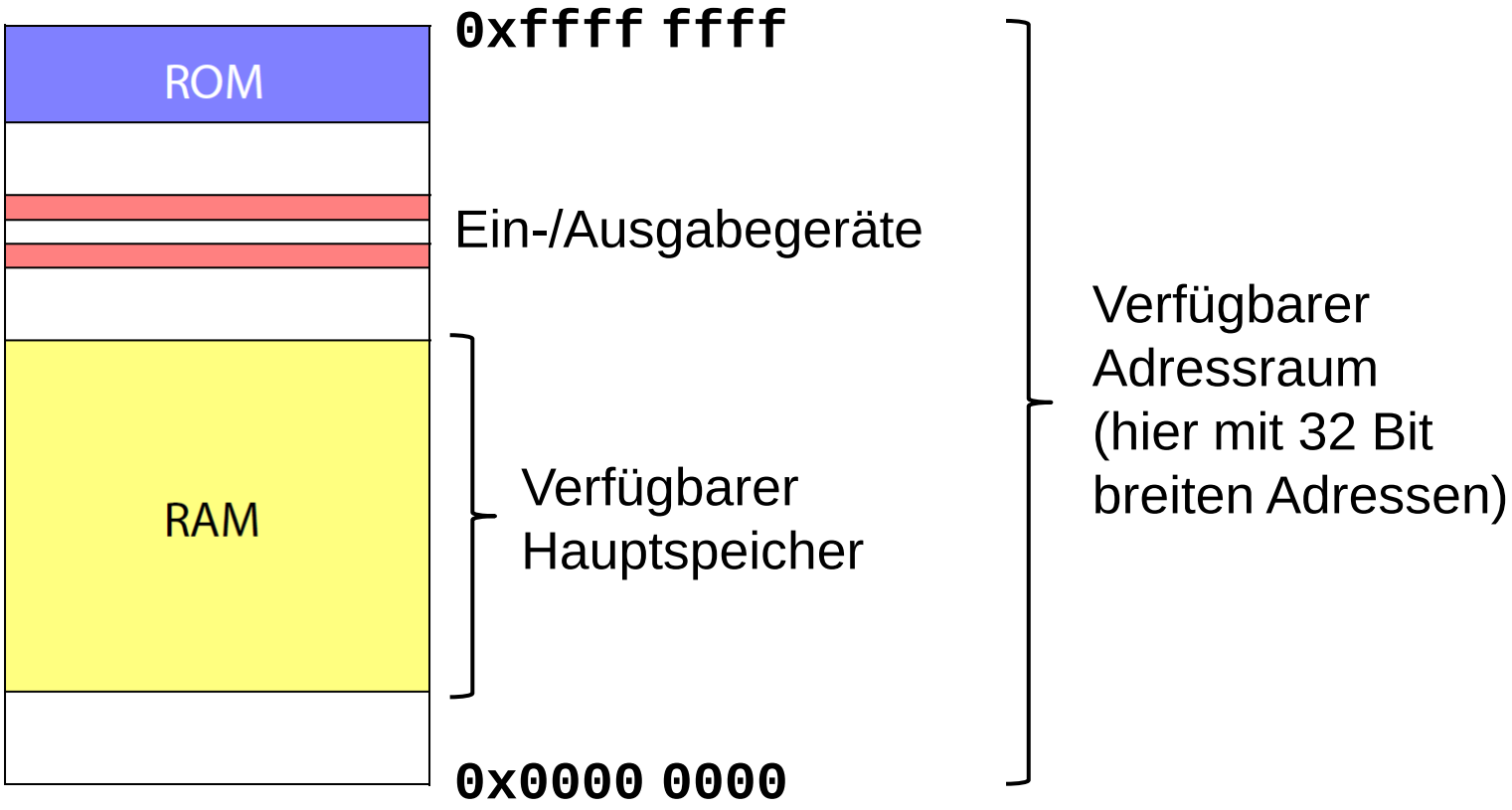
  1 GByte = 1024 MBytes = 2^{30} Bytes = 1.073.741.824 Bytes
- *Terabyte (TByte, TB)*

  1 TByte = 1024 GBytes = 2^{40} Bytes = 1.099.511.627.776 Bytes
- *Petabyte (PByte, PB)*

  1 PByte = 1024 TBytes = 2^{50} Bytes = 1.125.899.906.842.624 Bytes

Speichervergabe (1)

Verfügbarer Speicher



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 Benutzerprogramm

Probleme

– Begrenzung anderer Ressourcen
  (z.B. Bandbreite bei Ein-/Ausgabe wegen zu kleiner Systempuffer)
– Ungenutzter Speicher des Betriebssystems kann von
  Anwendungsprogramm nicht genutzt werden, und umgekehrt

☞ **Dynamische Speicherzuteilung einsetzen**

Dynamische Speicherzuteilung

Segmente

– Zusammenhängender Speicherbereich (Bereich mit aufeinanderfolgenden Adressen)
– Einzelne Segmente können unterschiedliche Länge haben

Erforderlich: Allokation (Anforderung) und Freigabe von Segmenten

Ein Anwendungsprogramm besitzt üblicherweise folgende Segmente

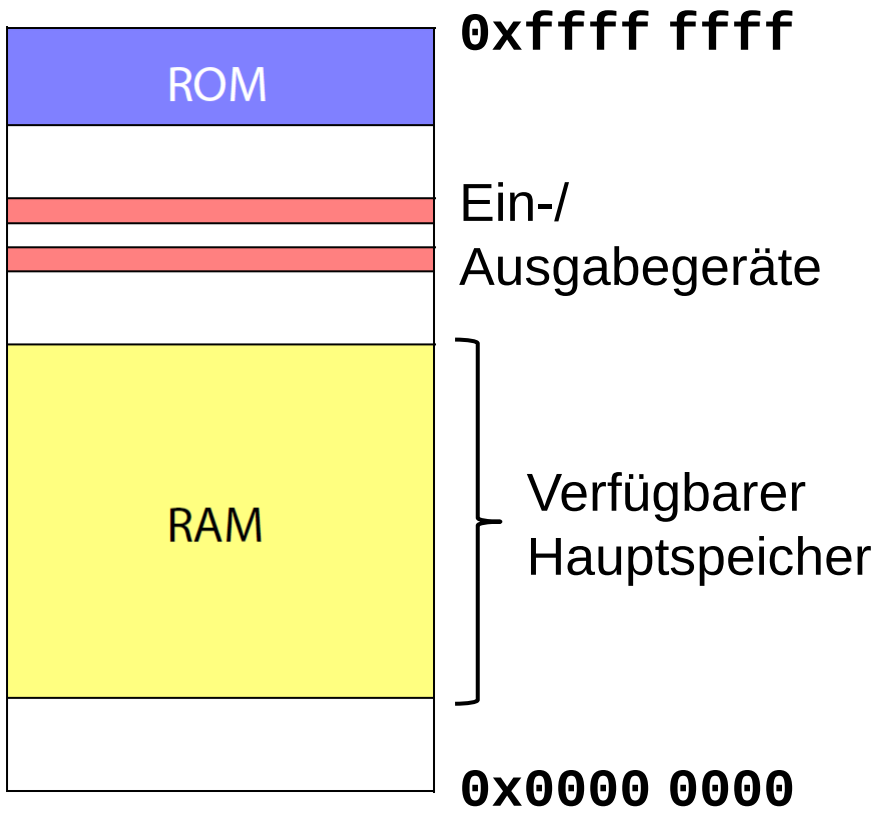
– Code-Segment
– Daten-Segment
– *Stack*-Segment (für Verwaltungsinformationen, z.B. bei Funktionsaufrufen)

☞ **Suche nach geeigneten Speicherbereichen zur Zuteilung**
☞ **Speicherzuteilungsstrategien**

Adressräume (1)

Physikalischer Adressraum

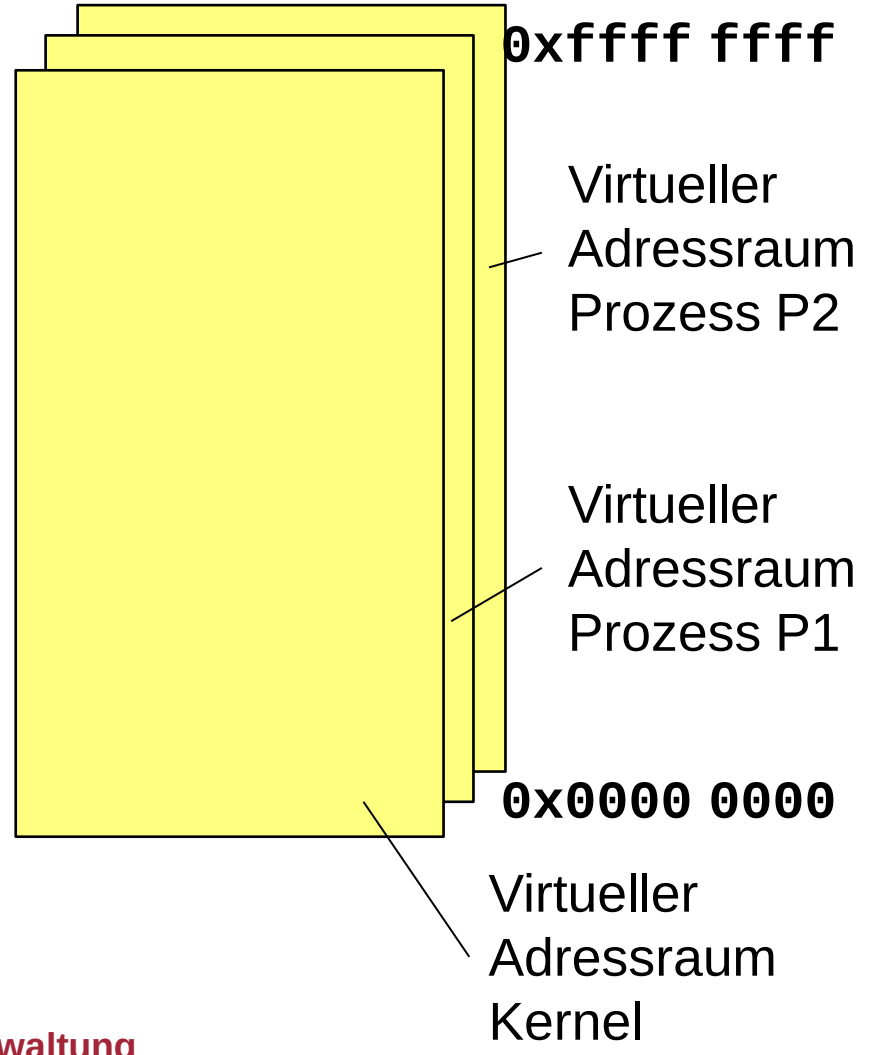
- Eine 32 Bit-Architektur kann physikalische Adressen von 0, ..., 2^{32}-1 erzeugen
- Nicht alle diese physikalischen Adressen gibt es in Hardware (weiße Bereiche rechts)
- Einzelne Teilbereiche des phys. Adressraums werden auf unterschiedliche Hardware-Komponenten abgebildet (bspw. DRAM-Module, Boot-ROMs, Video-Speicher, ...)



Adressräume (2)

Virtueller Adressraum

- Eine Software kann viel mehr Adressen benutzen als in physikalischem Adressraum überhaupt existieren
- Software wird ein virtueller Adressraum durch das Betriebssystem „vorgespielt“, der sämtliche möglichen Adressen 0, ..., 2^{32}-1 umfassen kann
- Jeder einzelne Anwender-Prozess hat seinen eigenen virtuellen Adressraum *(user space)*, und auch das Betriebssystem *(kernel space)*



Freispeicherverwaltung (1)

Freie (evtl. auch belegte) Segmente des Speichers müssen repräsentiert werden

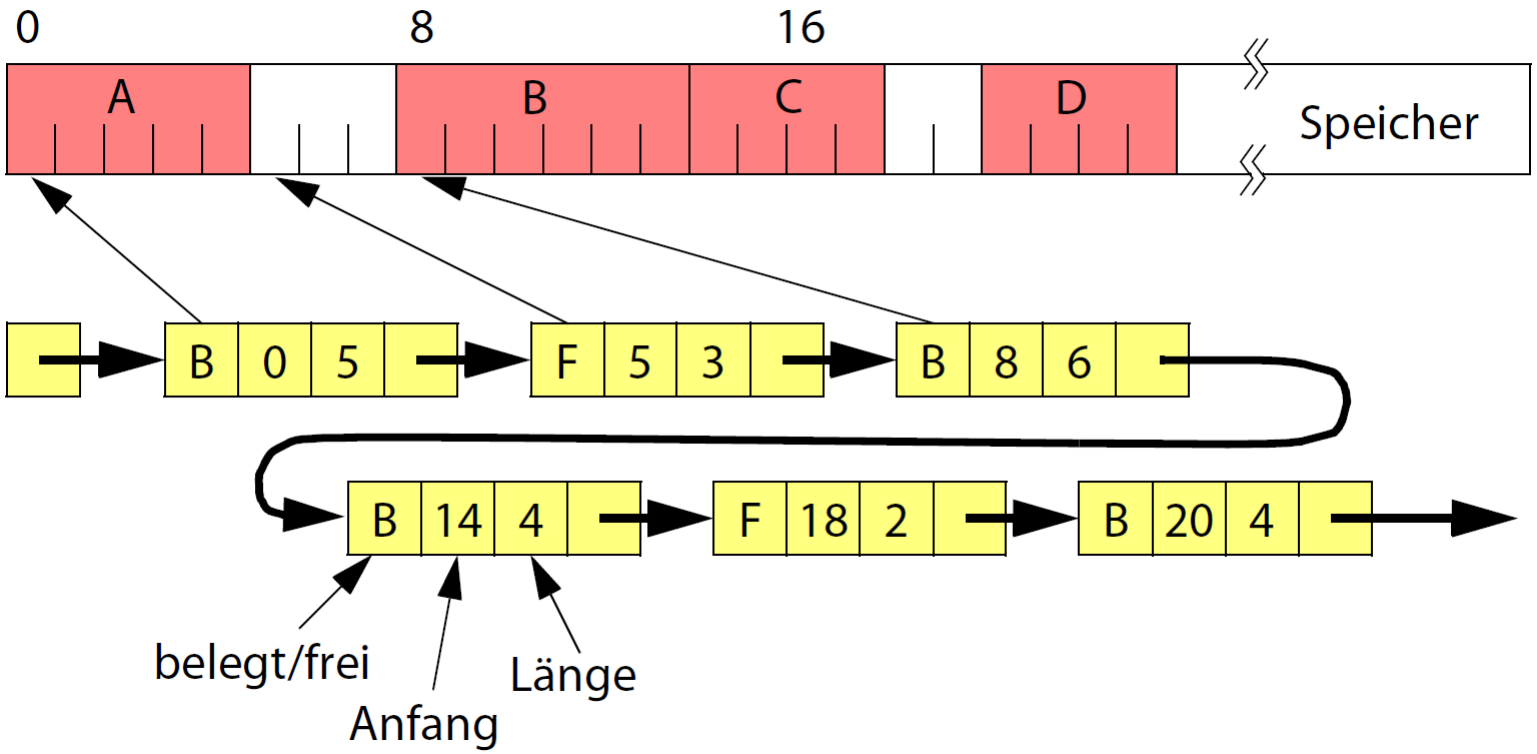
Bitvektoren

Bitvektor markiert belegte Speichereinheiten

Speichereinheiten gleicher Größe (z.B. 1 Byte, 64 Bytes, 1 kByte)

Freispeicherverwaltung (2)

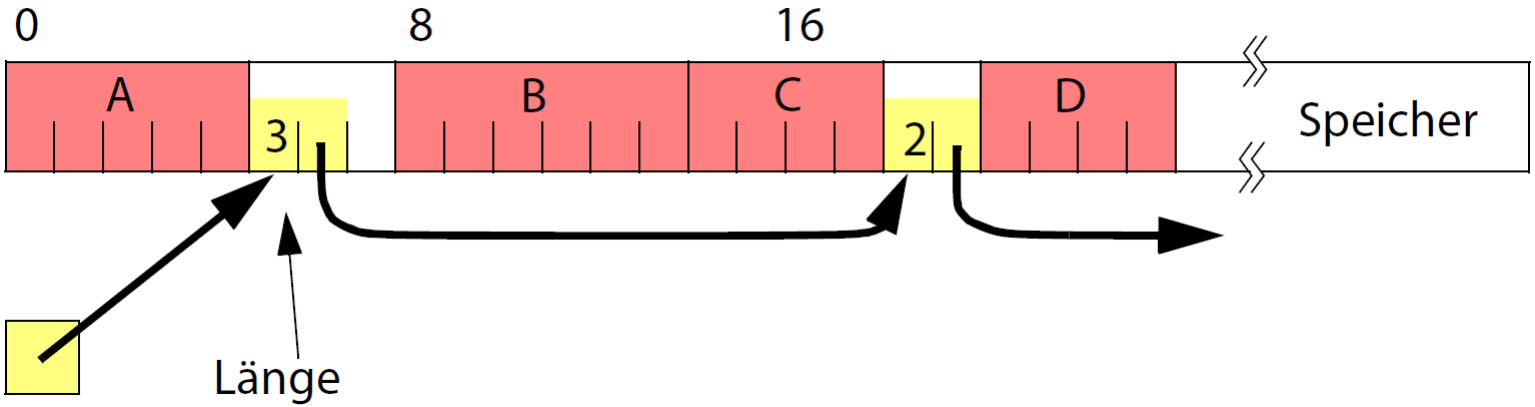
Verkettete Liste



Repräsentation auch von freien Segmenten

Freispeicherverwaltung (3)

Verkettete Liste im freien Speicher

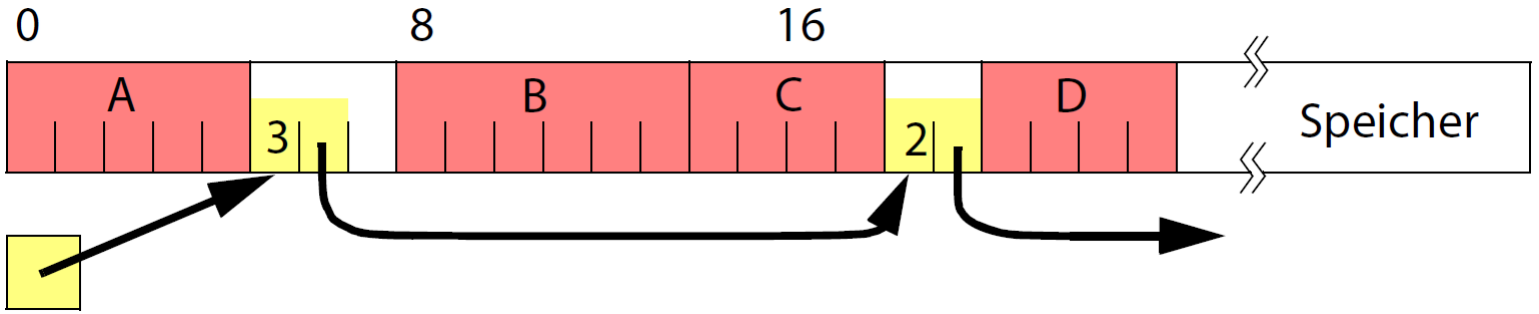


Zur Effizienzsteigerung evtl. Rückwärtsverkettung sinnvoll

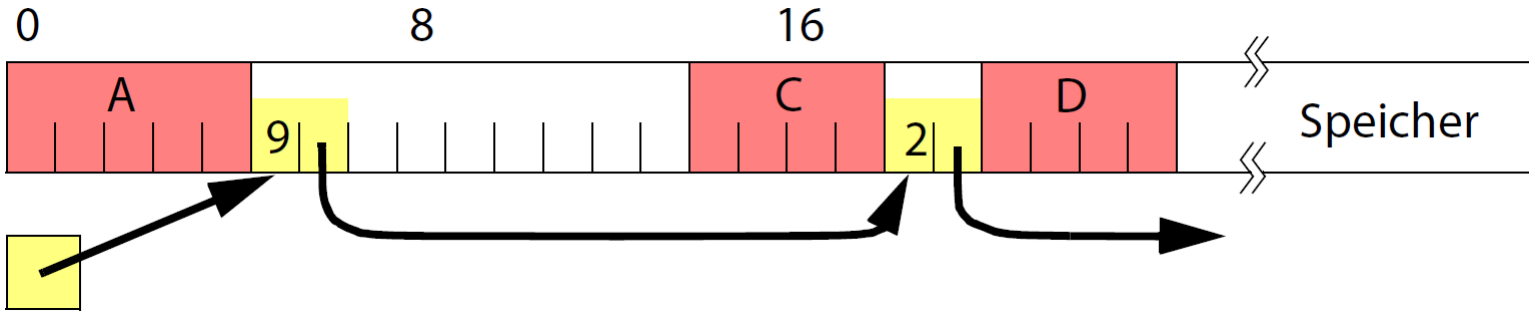
☞ Repräsentation letztlich auch von der Vergabestrategie abhängig

Speichervergabe

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

Probleme

- Speicherverschnitt (Fragmentierung)
- Zu kleine Lücken

Einsatz der Verfahren

Einsatz im Betriebssystem

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

Einsatz innerhalb eines Prozesses

- Verwaltung des Haldenspeichers *(Heap)*
- Erlaubt dynamische Allokation von Speicherbereichen durch den Prozess (**`malloc`** und **`free`**)

Einsatz für Bereiche des Sekundärspeichers

- Verwaltung bestimmter Abschnitte des Sekundärspeichers (Festplatte)
  - Z.B. Speicherbereich für Prozessauslagerungen *(swap space)*

Roter Faden

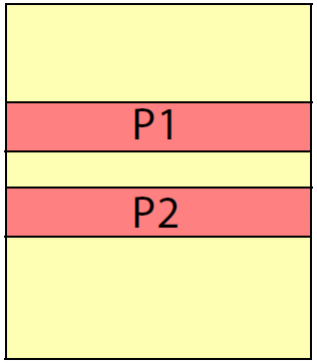
6. Speicherverwaltung

- Einleitung
- Speichervergabe
  - Statische und dynamische Speicherzuteilung
  - Freispeicherverwaltung (Bitvektoren, verkettete Listen)
  - Vergabestrategien
    - *First Fit*
    - *Next Fit*
    - *Best Fit*
    - *Worst Fit*
- Mehrprogrammbetrieb
- Virtueller Speicher

Mehrprogrammbetrieb

Problemstellung: Mehrere Prozesse benötigen Hauptspeicher

– Prozesse an verschiedenen Stellen im Hauptspeicher



Zwei Prozesse und deren Code-Segmente im Speicher

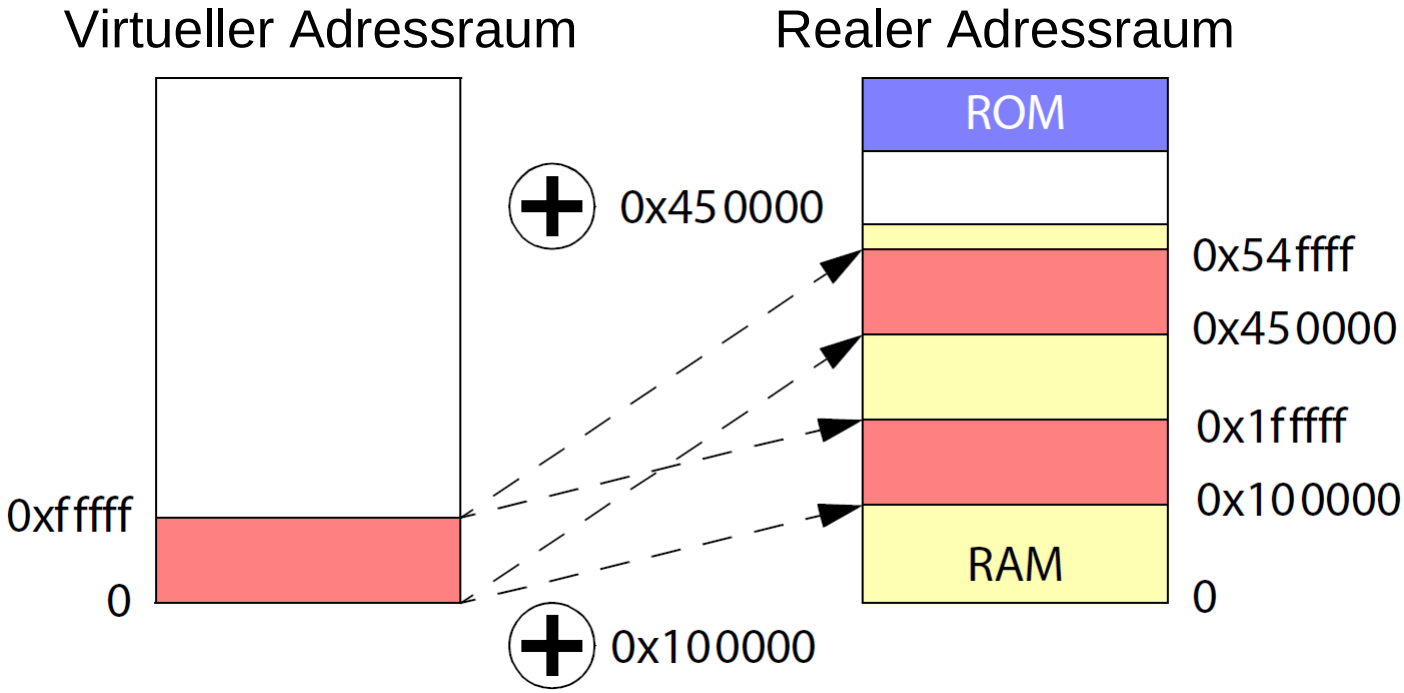
– Hauptspeicher nicht ausreichend

– Speicherschutz zwischen Betriebssystem und Prozessen sowie zwischen Prozessen untereinander

Segmentierung (1)

Hardwareunterstützung: Umsetzung virtueller in reale Adressen

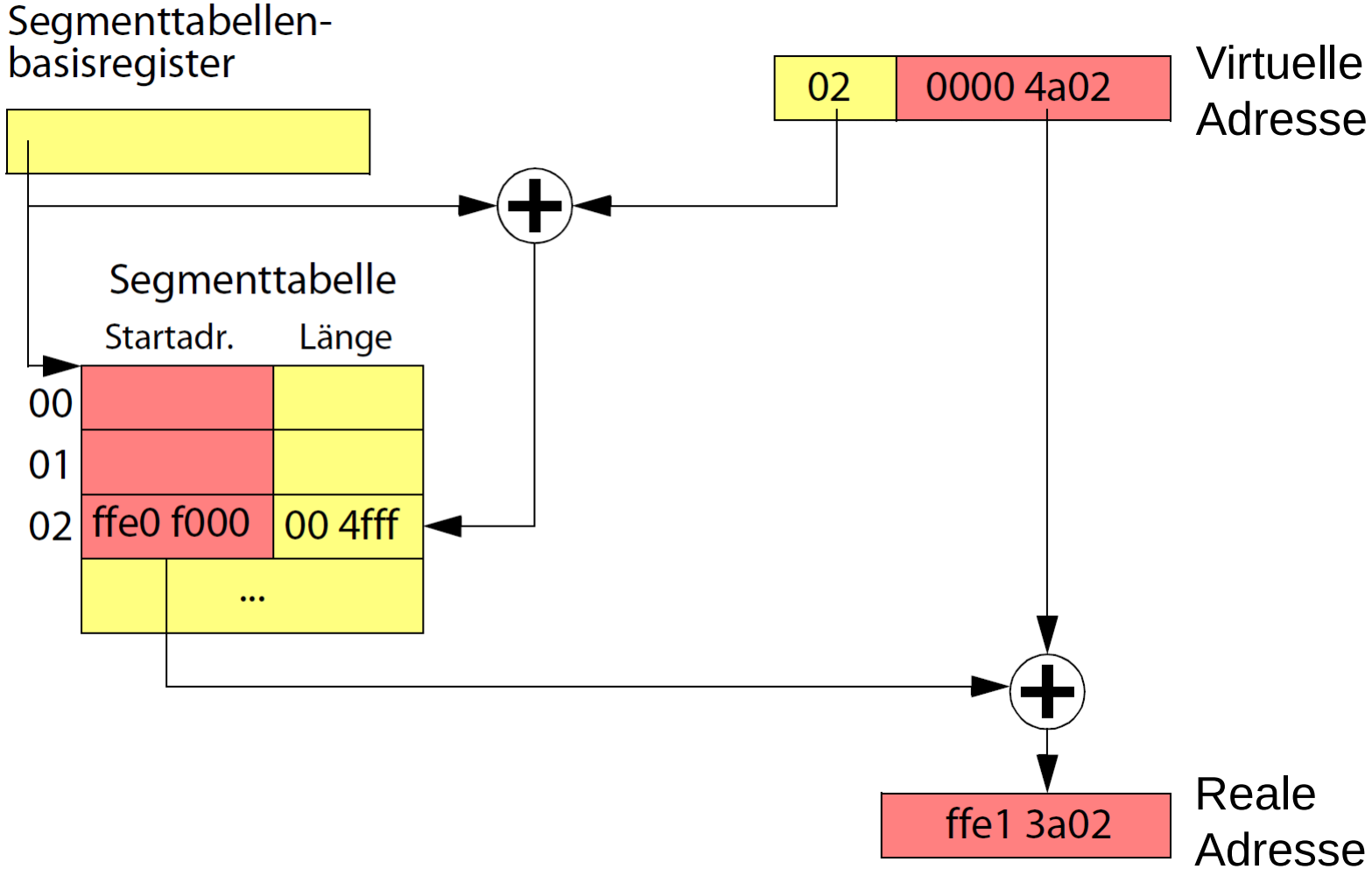
– Prozesse erhalten einen virtuellen Adressraum



Das Segment des virtuellen Adressraums kann an jeder beliebigen Stelle im realen Adressraum liegen

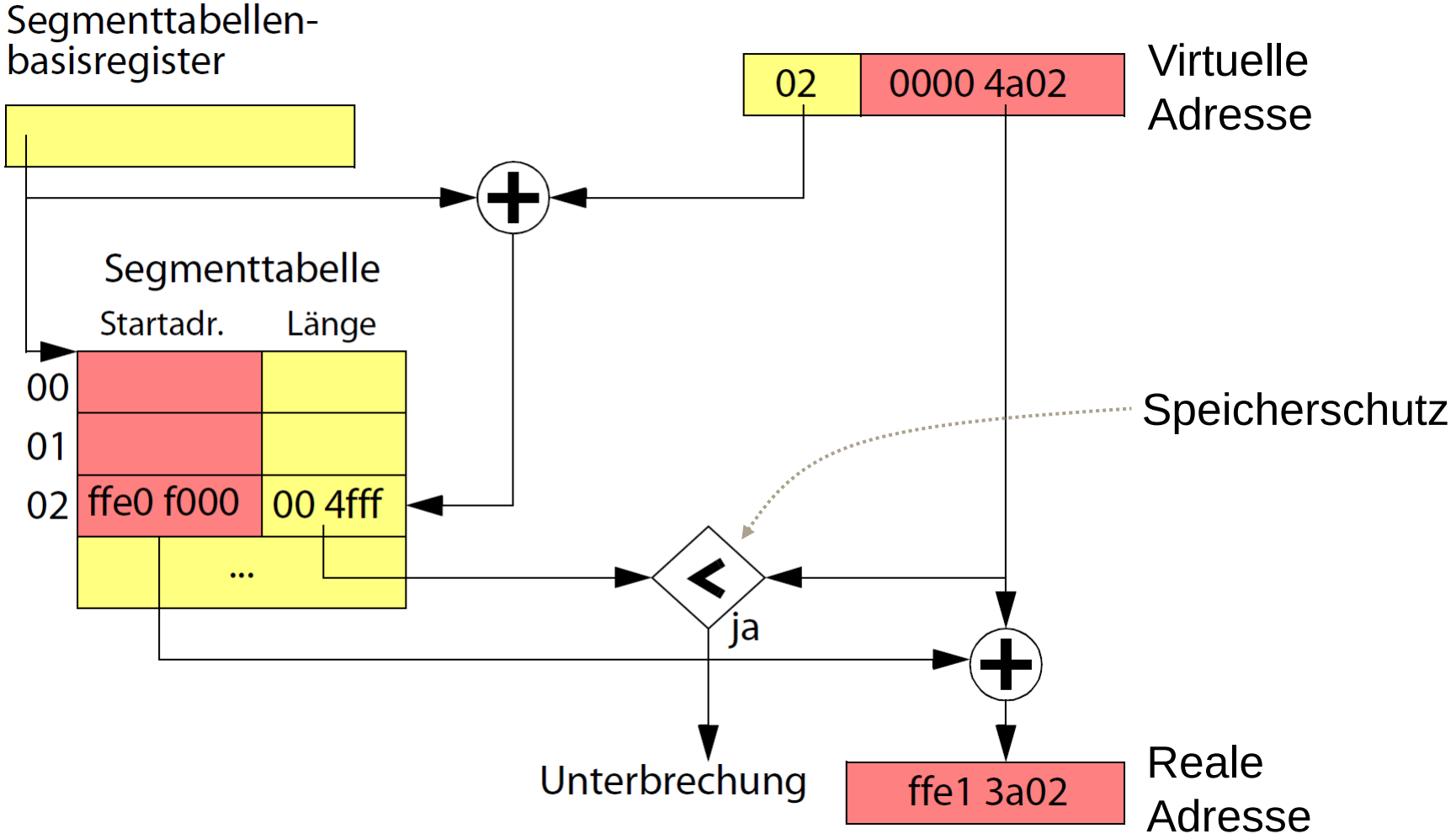
Segmentierung (2)

Realisierung mit Übersetzungstabelle



Segmentierung (3)

Realisierung mit Übersetzungstabelle



Segmentierung (4)

Hardware zur Adress-Umsetzung: *Memory Management Unit (MMU)*

- **Schutz vor Segmentübertretung**
  - Unterbrechung zeigt Speicherverletzung an
  - Programme und Betriebssystem voreinander geschützt
- **Kontextwechsel durch Austausch der Segmentbasis**
  - Jeder Prozess hat eigene Übersetzungstabelle (gehört zum Kontext)
- **Ein- und Auslagerung vereinfacht**
  - Nach Einlagerung an beliebige Stelle muss lediglich die Übersetzungstabelle angepasst werden
- **Gemeinsame Segmente möglich**
  - Code-Segmente (Befehlssegmente)
  - Datensegmente *(shared memory)*

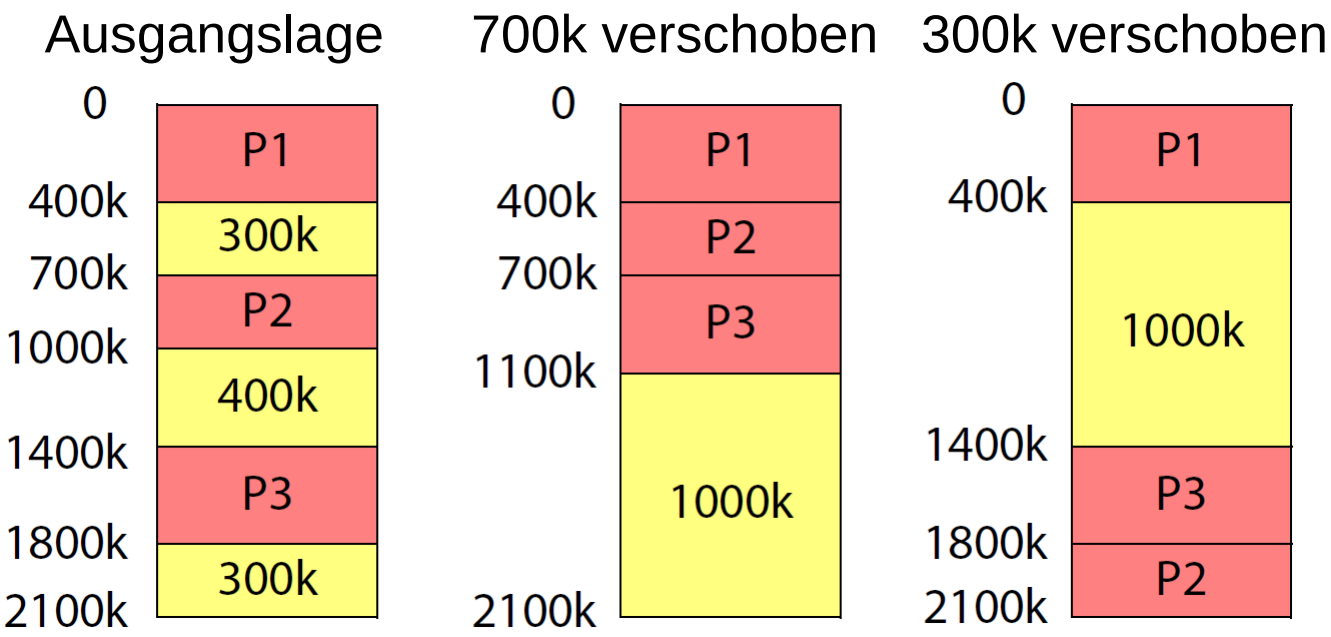
Segmentierung (5)

- **Zugriffsschutz einfach integrierbar**
  - Z.B. Rechte zum Lesen, Schreiben und Ausführen von Befehlen, die von der MMU geprüft werden
- **Externe Fragmentierung des Speichers durch häufiges Ein- und Auslagern**
  - Es entstehen kleine, nicht nutzbare Lücken
  - Lücken sind außerhalb (extern) der Segmente im Freispeicher-Bereich
- **Kompaktierung**
  - Segmente werden verschoben, um Lücken zu schließen
  - Segmenttabelle wird jeweils angepasst
- **Lange I/O-Zeiten für Ein- und Auslagerung**
  - Nicht alle Teile eines Segments werden gleich häufig genutzt

Kompaktierung

Verschieben von Segmenten

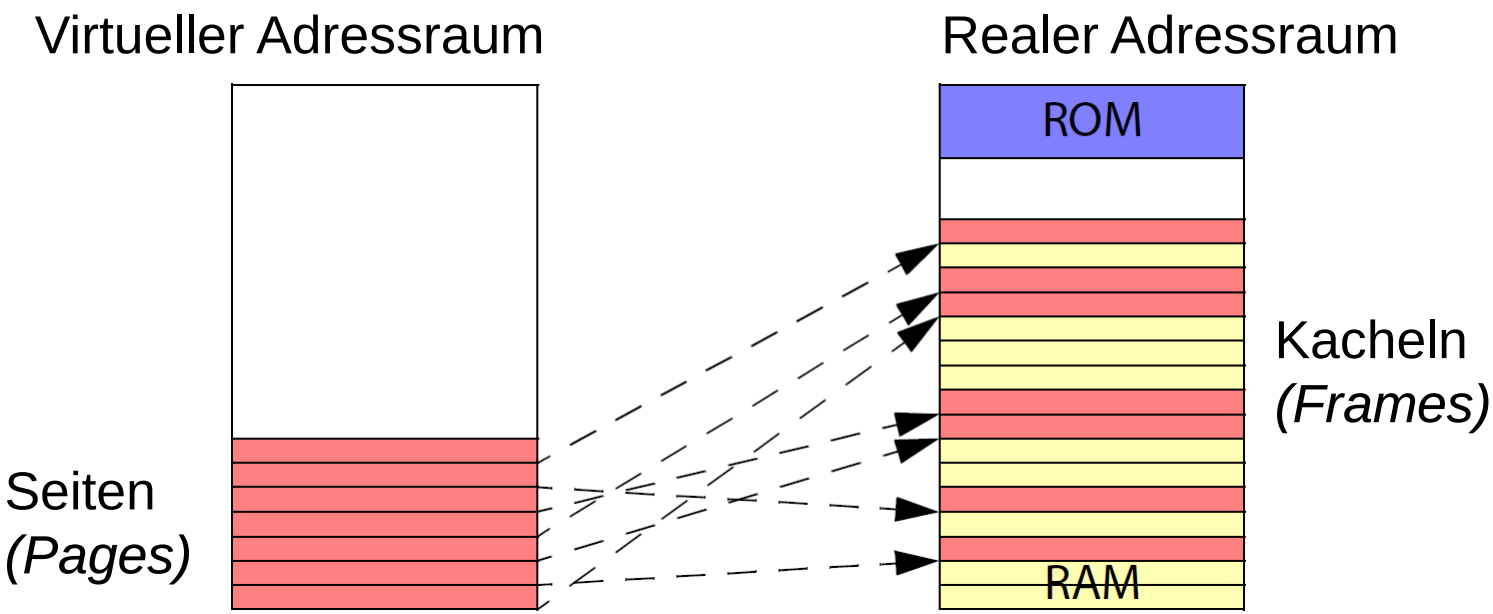
- Erzeugen von weniger, aber größeren Lücken
- Verringern des Verschnitts
- Aufwändige Operation, abhängig von der Größe der zu verschiebenden Segmente



Seitenadressierung *(Paging)*

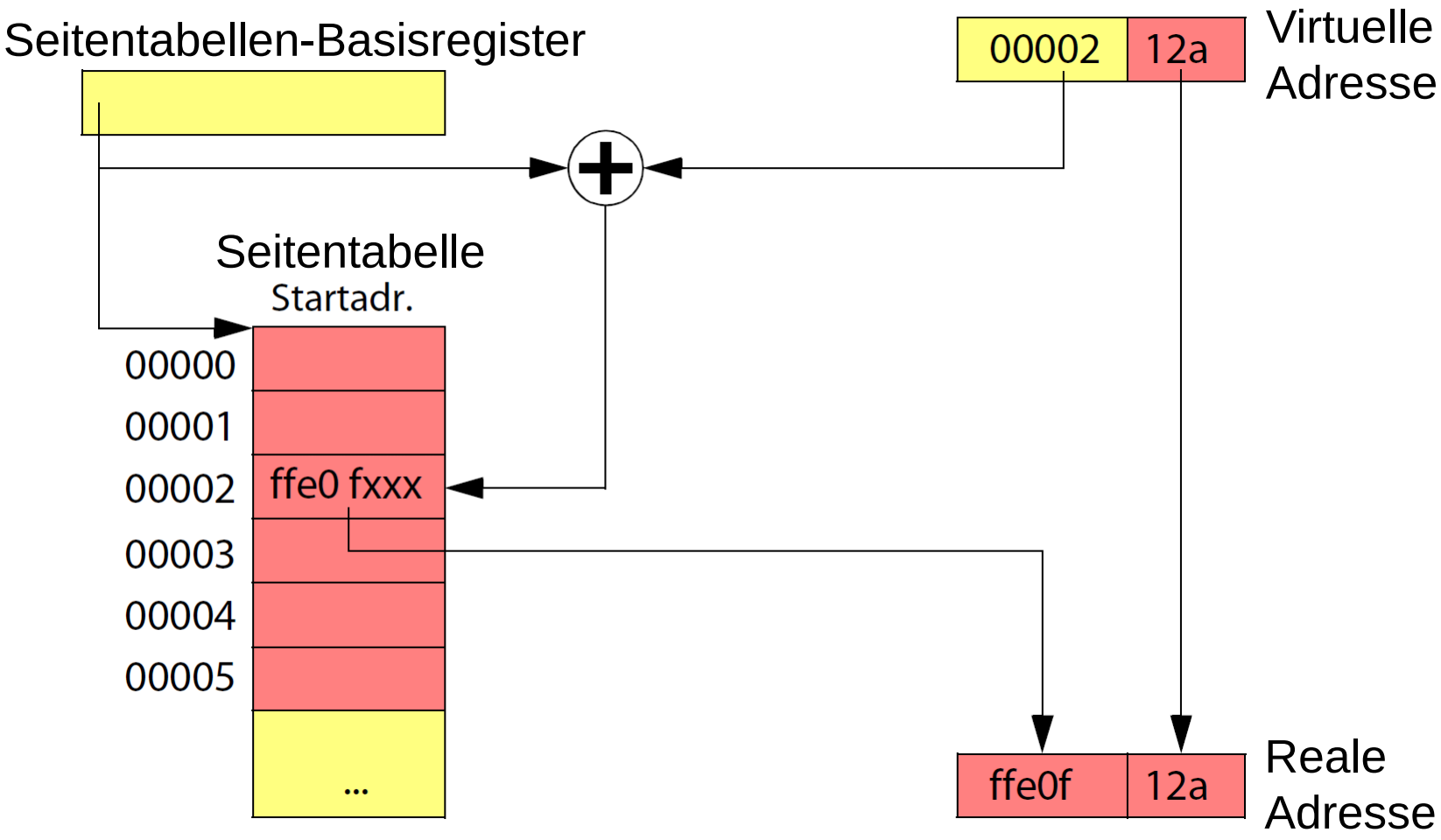
Einteilung des virtuellen Adressraums in gleichgroße Seiten, die an beliebigen Stellen im realen Adressraum liegen können

- Lösung des Problems der externen Fragmentierung
- Keine Kompaktierung mehr nötig
- Vereinfacht Speicherbelegung und Ein-/Auslagerungen



MMU mit Seitentabelle – *Page Table* (1)

Tabelle setzt Seiten in Kacheln um



MMU mit Seitentabelle – *Page Table* (2)

Paging erzeugt interne Fragmentierung

– Letzte Seite evtl. nicht vollständig genutzt

Seitengröße

– Kleine Seiten verringern interne Fragmentierung, vergrößern aber die *Page Table* (und umgekehrt)

– Übliche Größen: 512 Bytes – 8 kB

Eigenschaften

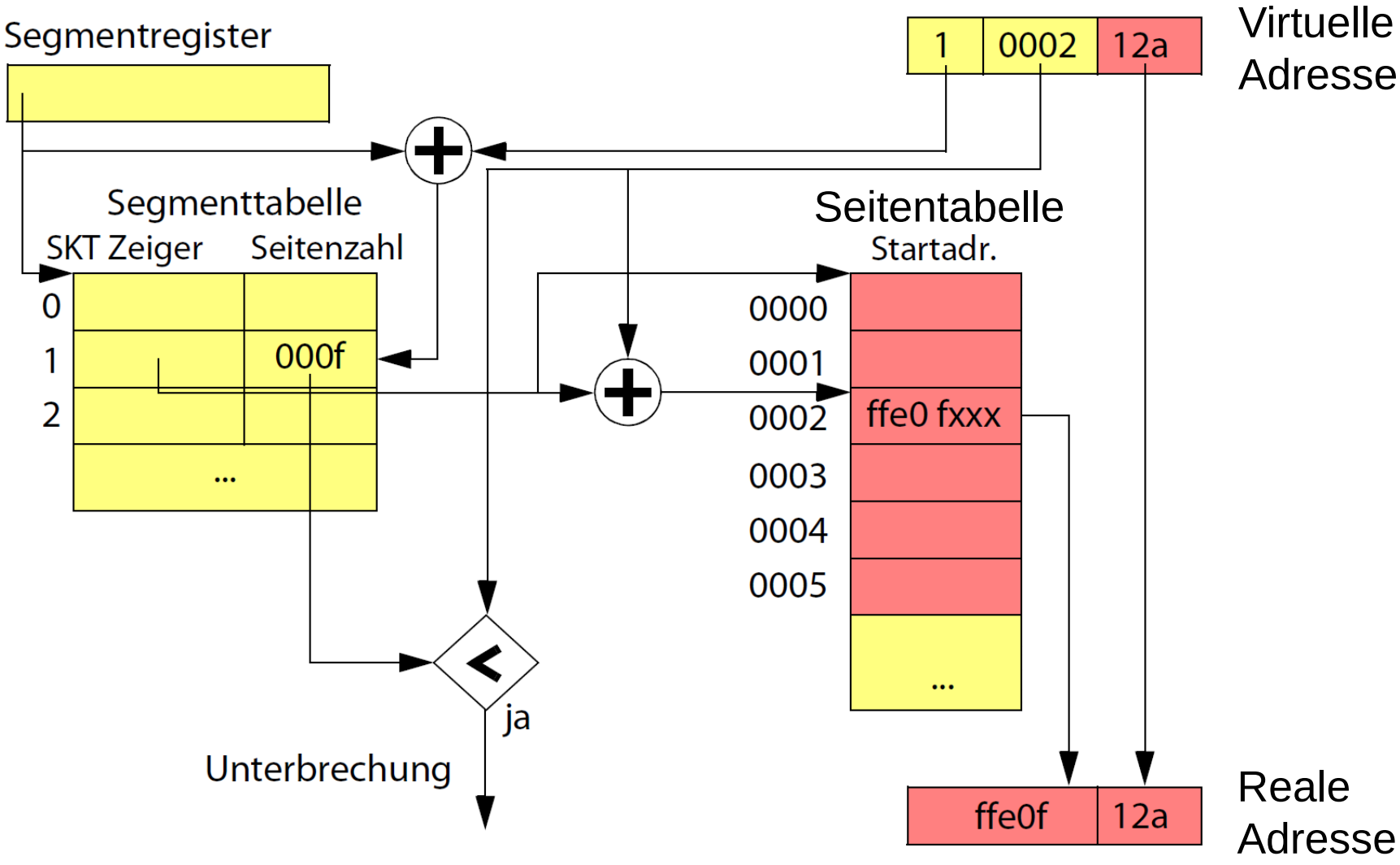
– Große Tabelle, die im Speicher gehalten werden muss

– Viele implizite Speicherzugriffe zur Adressumsetzung nötig

– Nur ein „Segment“ pro Prozess-Kontext (in dieser Variante)

☞ **Kombination mit Segmentierung**

Segmentierung und Seitenadressierung (1)



Segmentierung und Seitenadressierung (2)

Eigenschaften

- Noch mehr implizite Speicherzugriffe
- Große Tabellen im Speicher

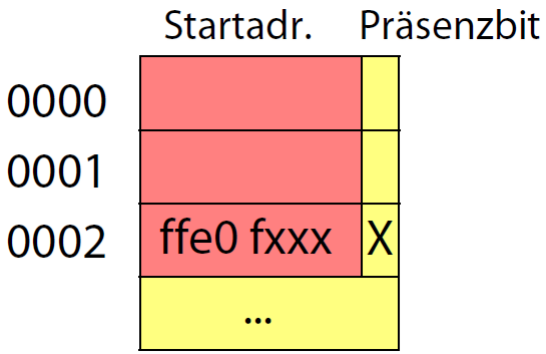
☞ **Mehrstufige Seitenadressierung mit Ein- und Auslagerung**

Ein- und Auslagerung von Seiten

Es ist nicht nötig, ein gesamtes Segment aus- bzw. einzulagern

– Seiten können einzeln ein- und ausgelagert werden

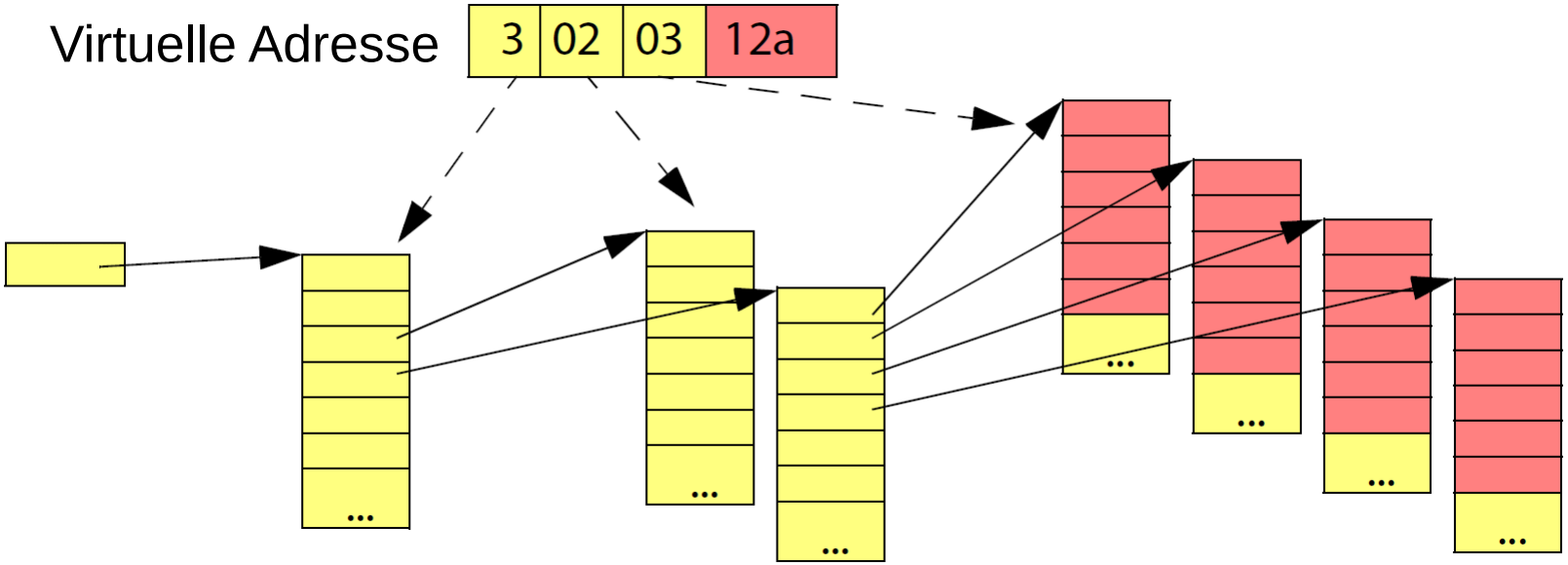
Hardware-Unterstützung



– Ist das Präsenzbit gesetzt, bleibt alles wie bisher
– Ist das Präsenzbit gelöscht, wird eine Unterbrechung ausgelöst *(page fault)*
– Die Unterbrechungsbehandlung kann nun für das Laden der Seite vom Hintergrund-speicher sorgen und den Speicherzugriff danach wiederholen
(benötigt HW-Unterstützung in der CPU)

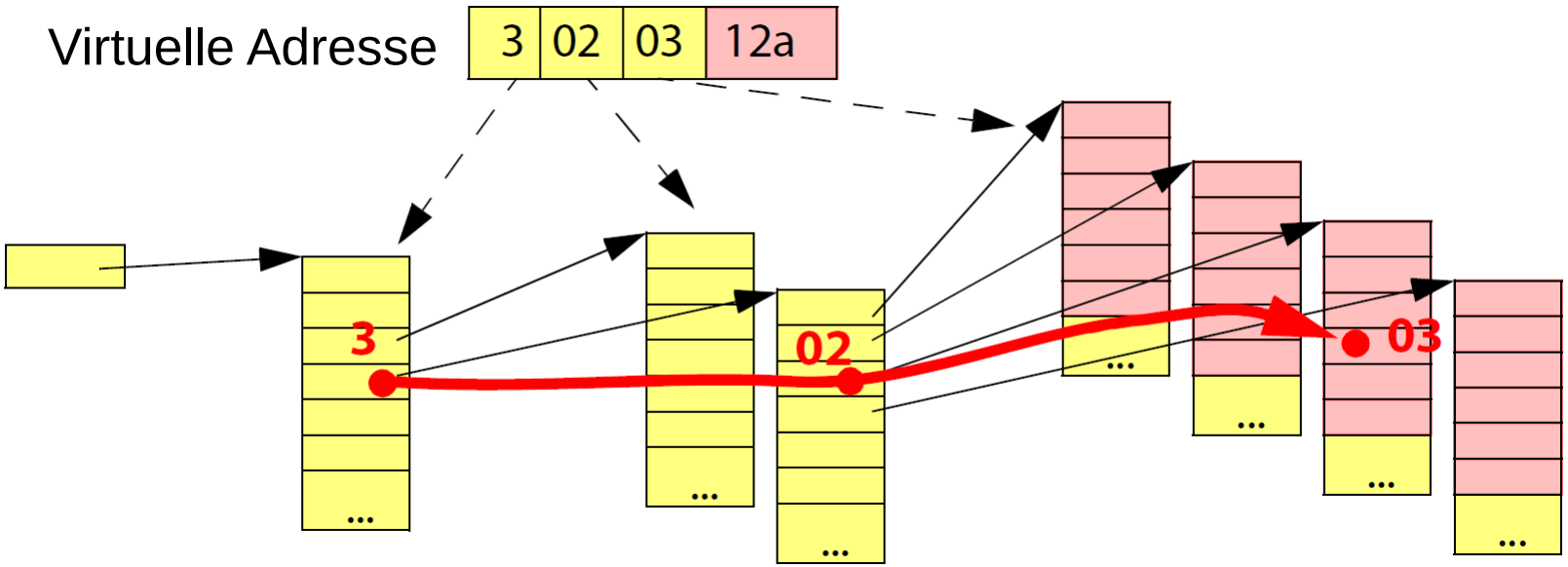
Mehrstufige Seitenadressierung (1)

Beispiel: Zweifach-indirektes *Paging*



Mehrstufige Seitenadressierung (2)

Beispiel: Zweifach-indirektes *Paging*



Aus- und einlagerbare Tabellen

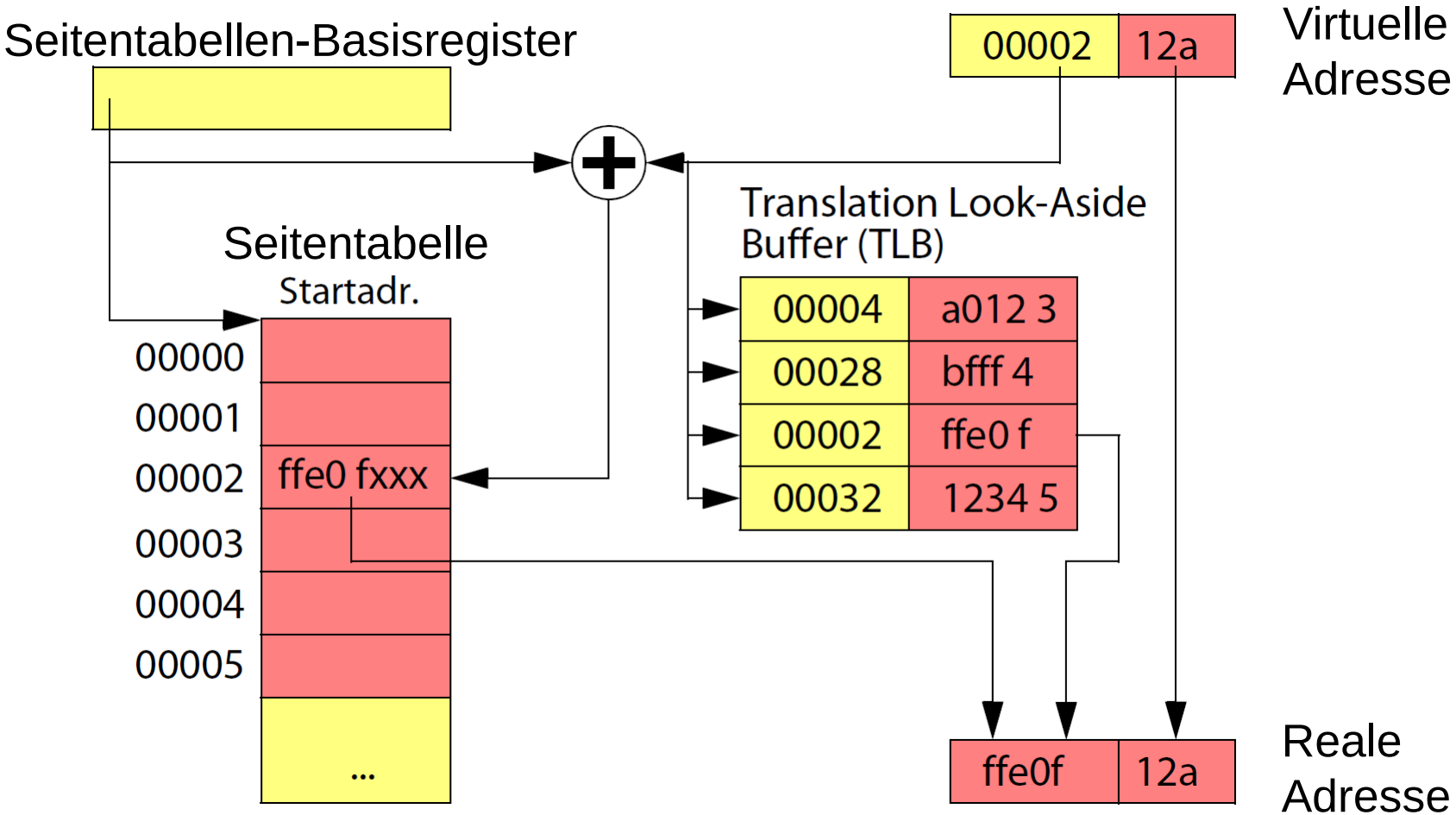
– Tabellen sind kleiner (ideal: Seitengröße)

– Alle Einträge mit Präsenzbit (auch in höheren Ebenen)

Noch mehr implizite Speicherzugriffe

Translation Look-Aside Buffer, TLB (1)

Schneller Pufferspeicher; wird konsultiert, bevor auf Seitentabelle zugegriffen wird



Translation Look-Aside Buffer, TLB (2)

Eigenschaften

- Schneller Zugriff auf Seitenabbildung, falls Information im TLB-Speicher gepuffert
  - Keine impliziten Speicherzugriffe nötig
- Bei Zugriffen auf eine nicht im TLB enthaltene Seite wird die entsprechende Zugriffsinformation in den TLB eingetragen
  - Ein alter Eintrag muss zur Ersetzung ausgesucht werden
- Bei Kontextwechseln muss TLB gelöscht werden *(Flush)*

Typische TLB-Größen

- *Pentium:* Daten-TLB: 64 Einträge, Code-TLB: 32, Seitengröße: 4 kB
- *Sparc V9:* Daten-TLB: 64, Code-TLB: 64, Seitengröße: 8 kB
- *Core i7:* Level 1 Daten-TLB: 64, Level 1 Code-TLB: 128
  Level 2 Daten- und Code-TLBs: 512, Seitengröße: 4 kB

Roter Faden

6. Speicherverwaltung

- Einleitung
- Speichervergabe
- Mehrprogrammbetrieb
  - Segmentierung
  - Kompaktierung
  - Seitenadressierung *(Paging)*
    - *Memory Management Unit* für *Paging*
    - Segmentierung und Seitenadressierung
    - Mehrstufiges *Paging*
    - *Translation Look-Aside Buffer (TLBs)*
- Virtueller Speicher

Virtueller Speicher

Entkopplung des Speicherbedarfs vom verfügbaren Hauptspeicher

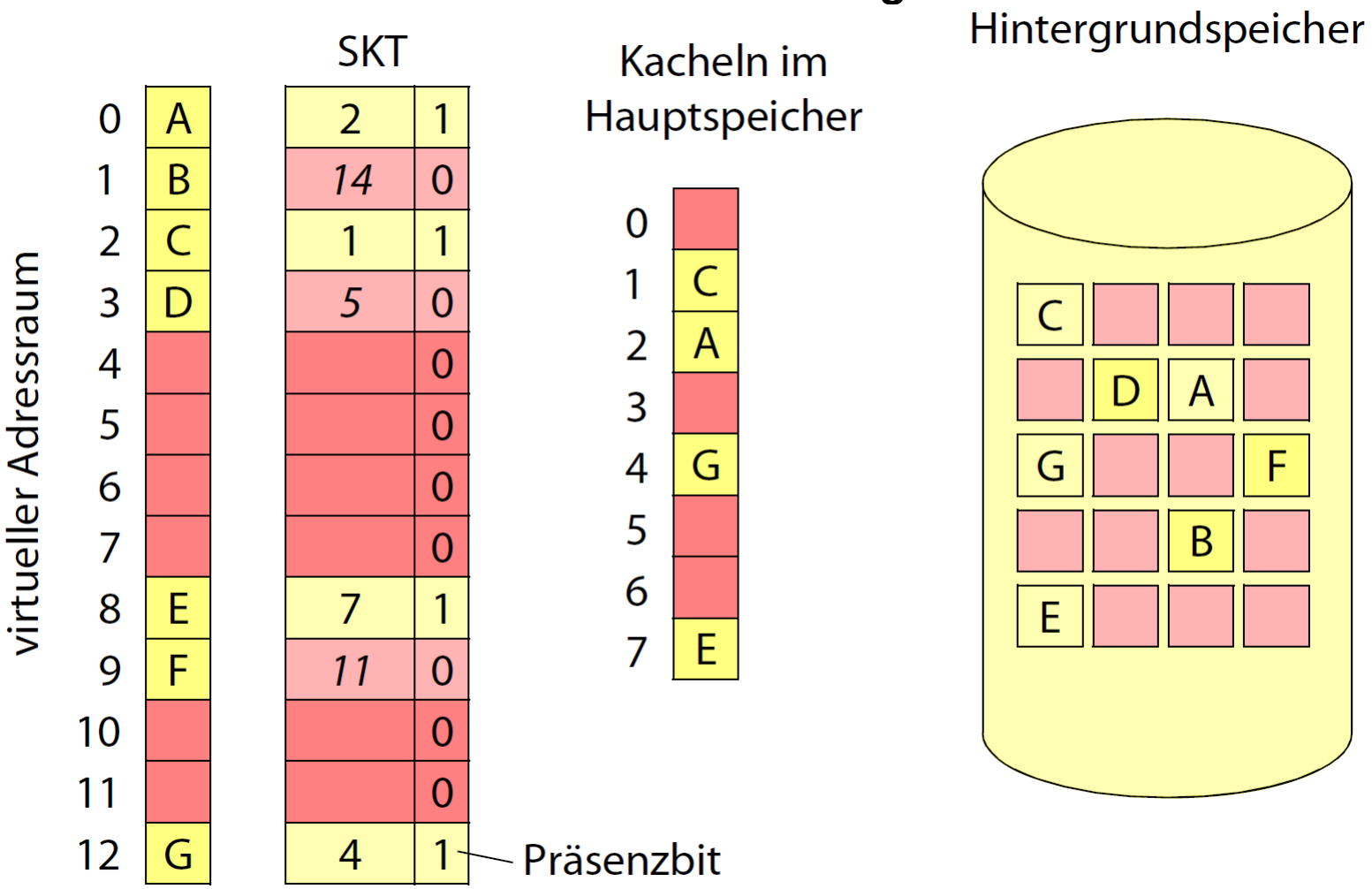
- Prozesse benötigen nicht alle Speicherstellen gleich häufig
  - Bestimmte Befehle werden selten oder gar nicht benutzt (z.B. Fehlerbehandlungen)
  - Bestimmte Datenstrukturen werden nicht voll belegt
- Prozesse benötigen evtl. mehr Speicher als Hauptspeicher vorhanden

Idee

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

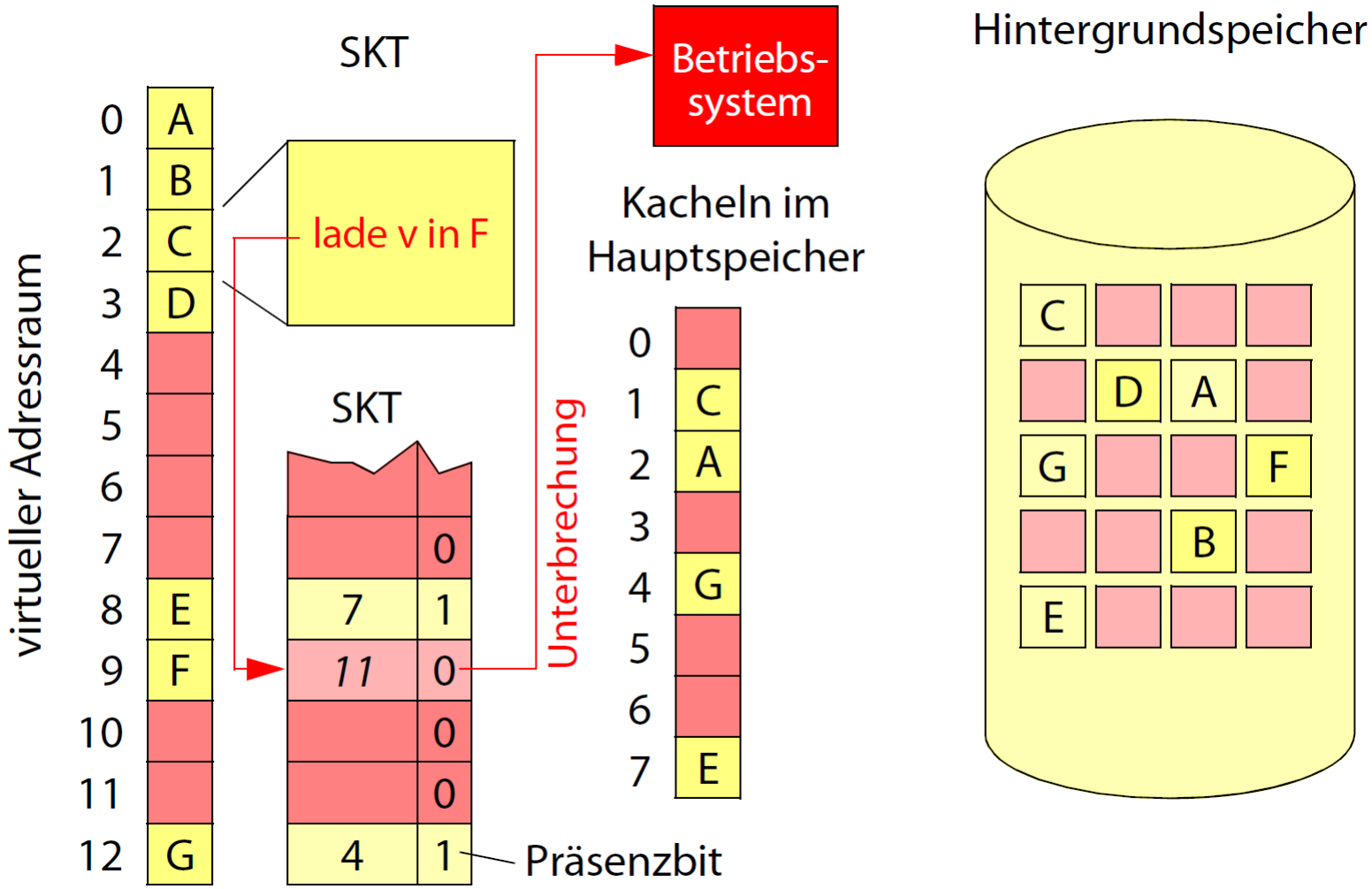
Demand Paging (1)

Bereitstellen von Seiten auf Anforderung



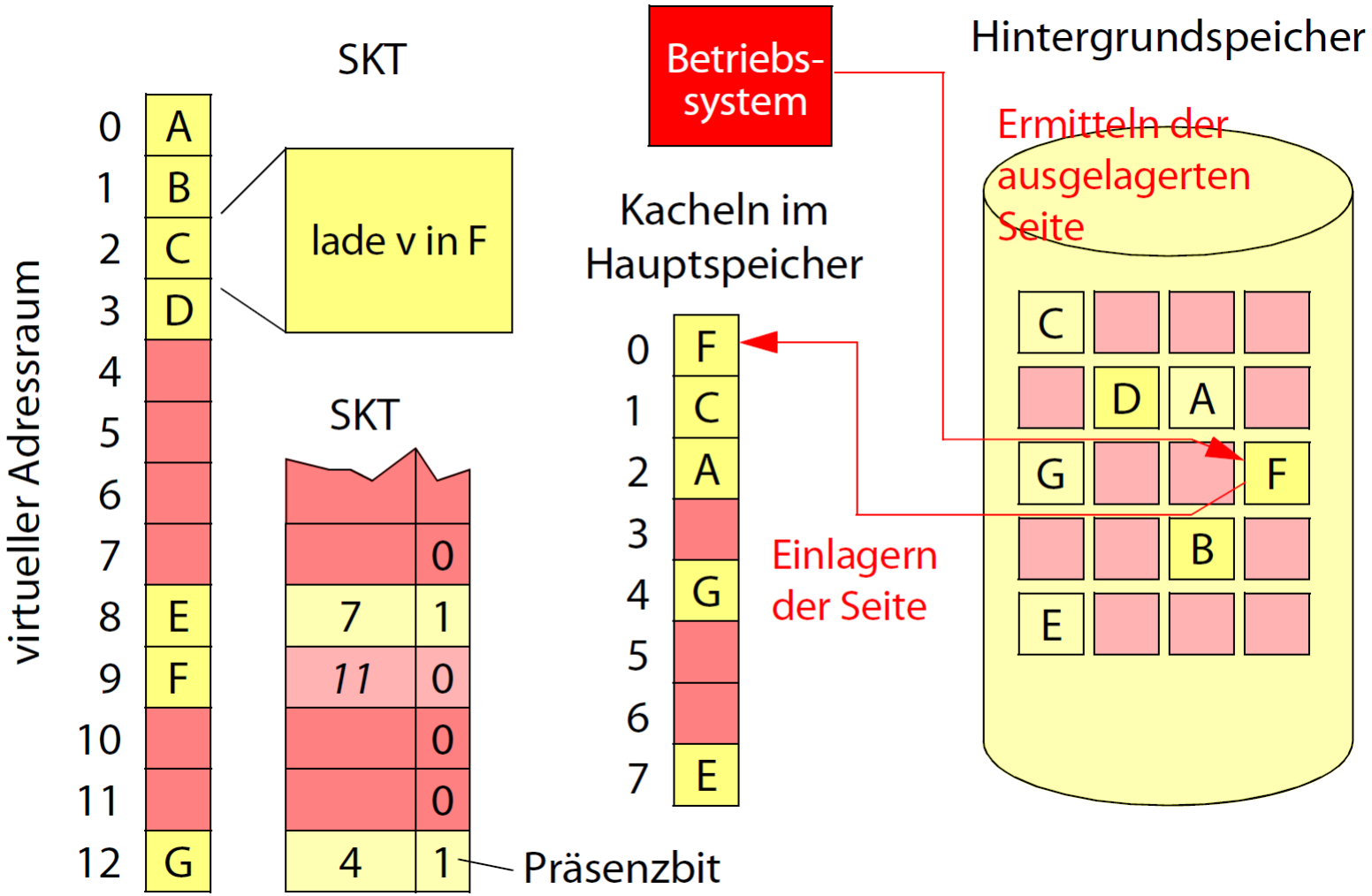
Demand Paging (2)

Reaktion auf Seitenfehler



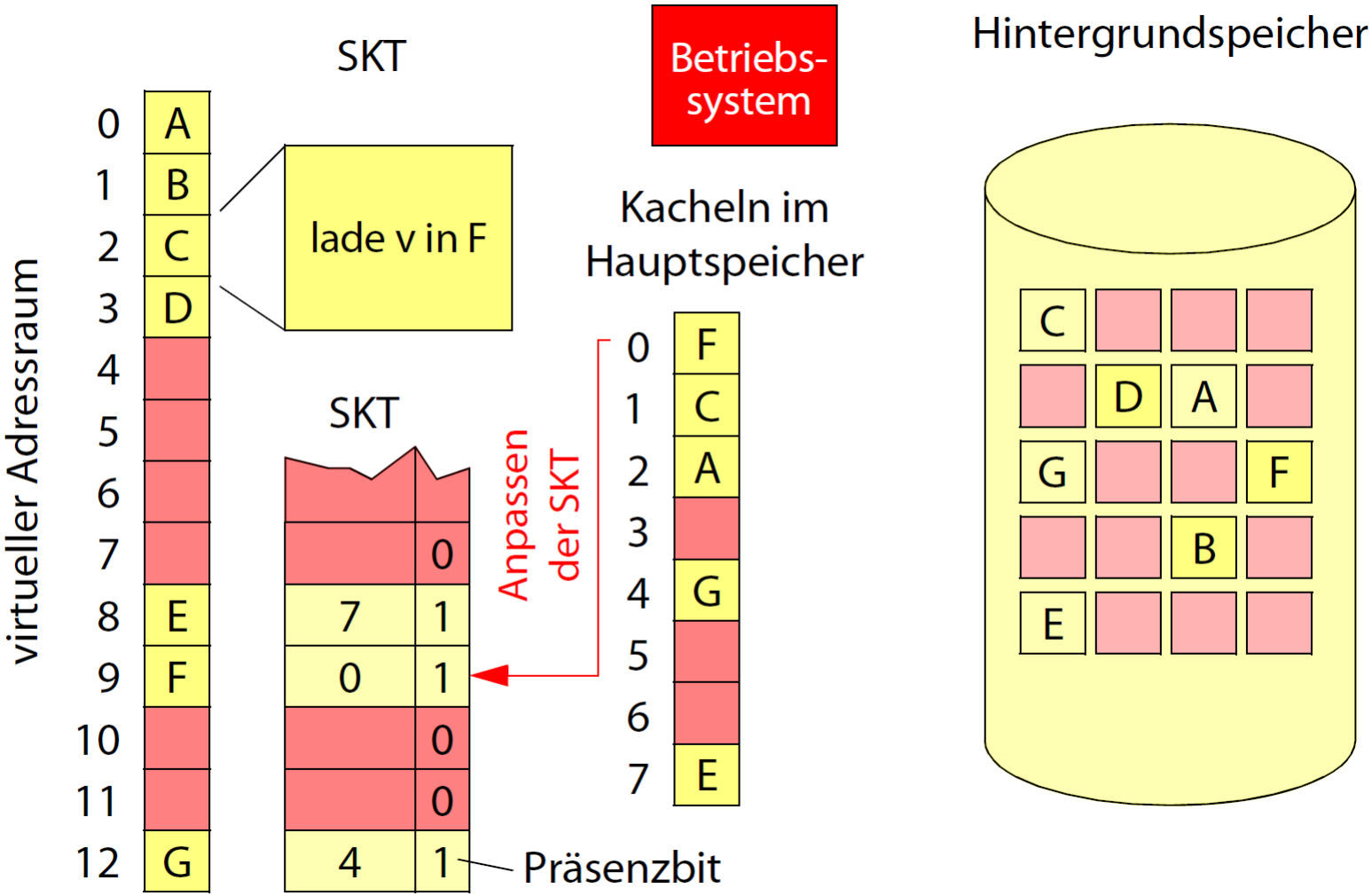
Demand Paging (3)

Reaktion auf Seitenfehler



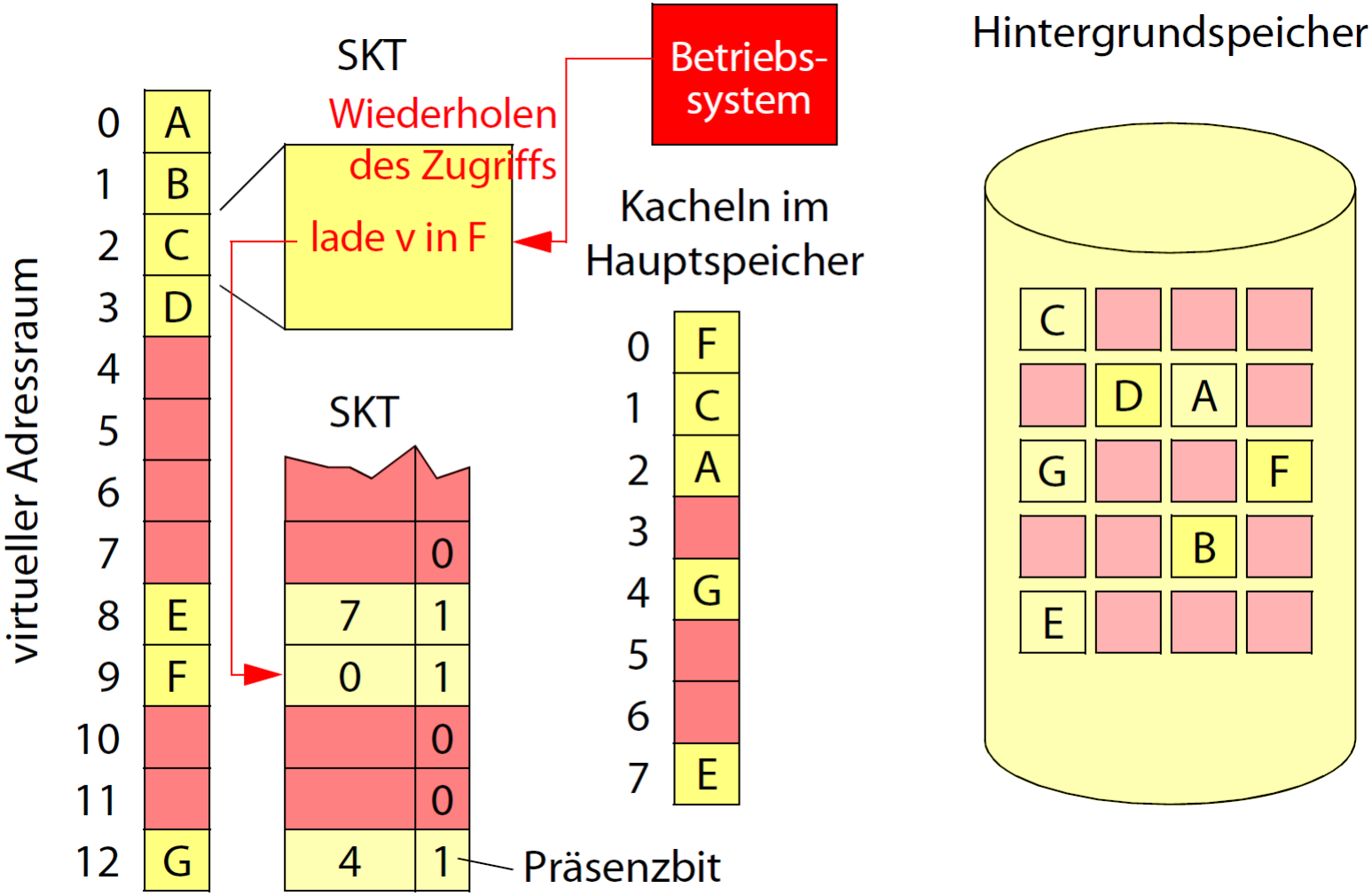
Demand Paging (4)

Reaktion auf Seitenfehler



Demand Paging (5)

Reaktion auf Seitenfehler



Demand Paging (6)

Performance von *Demand Paging*

- Keine Seitenfehler
  - Effektive Zugriffszeit zwischen 10 und 200 Nanosekunden
- Mit Seitenfehler
  - p sei Wahrscheinlichkeit für Seitenfehler, p nahe Null
  - Annahme: Zeit zum Einlagern einer Seite vom Hintergrundspeicher gleich 25 Millisekunden (9 ms Latenz, 15 ms Positionierzeit, 1 ms Übertragungszeit)
  - Annahme: normale Zugriffszeit 100 ns
  - Effektive Zugriffszeit
    $(1 - p) * 100\ ns + p * 25.000.000\ ns = (100 + 24.999.900 * p)\ ns$

☞ **Seitenfehler müssen so gering wie möglich gehalten werden!**

Seitenersetzung

Was tun, wenn keine freie Kachel vorhanden?

– Eine Kachel muss verdrängt werden, um Platz für neue Kachel zu schaffen!
– Auswahl von Kacheln, die nicht geändert wurden (*dirty bit* in der Seitentabelle)
– Verdrängung erfordert Auslagerung, falls Kachel geändert wurde

Vorgang

– Seitenfehler *(page fault)*: Unterbrechung
– Auslagern einer Kachel, falls keine freie Kachel verfügbar
– Einlagern der benötigten Kachel
– Wiederholung des Zugriffs

Problem

– Welche Kachel soll ausgewählt werden?

Ersetzungsstrategien

Betrachtung von Ersetzungsstrategien und deren Wirkung auf Referenzfolgen

Referenzfolge

- 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
  - Zusammenfassung von unmittelbar hintereinander stehenden Zugriffen auf die gleiche Seite
- Beispiel für eine Referenzfolge: 1, 2, 3, 4, 1, 2, 5, 1, 2, 3, 4, 5

First-In First-Out, FIFO (1)

Älteste Kachel wird ersetzt

- Notwendige Zustände
  - Alter bzw. Einlagerungszeitpunkt für jede Kachel

Ablauf der Ersetzungen (9 Einlagerungen)

Referenzfolge		1	2	3	4	1	2	5	1	2	3	4	5
Hauptspeicher	Kachel 1	1											
	Kachel 2												
	Kachel 3												
Kontrollzustände (Alter pro Kachel)	Kachel 1	0											
	Kachel 2	>											
	Kachel 3	>											

First-In First-Out, FIFO (2)

Älteste Kachel wird ersetzt

- Notwendige Zustände
  - Alter bzw. Einlagerungszeitpunkt für jede Kachel

Ablauf der Ersetzungen (9 Einlagerungen)

Referenzfolge		1	2	3	4	1	2	5	1	2	3	4	5
Hauptspeicher	Kachel 1	1	1										
	Kachel 2		2										
	Kachel 3												
Kontrollzustände (Alter pro Kachel)	Kachel 1	0	1										
	Kachel 2	>	0										
	Kachel 3	>	>										

First-In First-Out, FIFO (3)

Älteste Kachel wird ersetzt

- Notwendige Zustände
  - Alter bzw. Einlagerungszeitpunkt für jede Kachel

Ablauf der Ersetzungen (9 Einlagerungen)

Referenzfolge		1	2	3	4	1	2	5	1	2	3	4	5
Hauptspeicher	Kachel 1	1	1	1									
	Kachel 2		2	2									
	Kachel 3			3									
Kontrollzustände (Alter pro Kachel)	Kachel 1	0	1	2									
	Kachel 2	>	0	1									
	Kachel 3	>	>	0									

First-In First-Out, FIFO (4)

Älteste Kachel wird ersetzt

- Notwendige Zustände
  - Alter bzw. Einlagerungszeitpunkt für jede Kachel

Ablauf der Ersetzungen (9 Einlagerungen)

Referenzfolge		1	2	3	4	1	2	5	1	2	3	4	5
Hauptspeicher	Kachel 1	1	1	1	4								
	Kachel 2		2	2	2								
	Kachel 3			3	3								
Kontrollzustände (Alter pro Kachel)	Kachel 1	0	1	2	0								
	Kachel 2	>	0	1	2								
	Kachel 3	>	>	0	1								

First-In First-Out, FIFO (5)

Älteste Kachel wird ersetzt

- Notwendige Zustände
  - Alter bzw. Einlagerungszeitpunkt für jede Kachel

Ablauf der Ersetzungen (9 Einlagerungen)

Referenzfolge		1	2	3	4	1	2	5	1	2	3	4	5
Hauptspeicher	Kachel 1	1	1	1	4	4							
	Kachel 2		2	2	2	1							
	Kachel 3			3	3	3							
Kontrollzustände (Alter pro Kachel)	Kachel 1	0	1	2	0	1							
	Kachel 2	>	0	1	2	0							
	Kachel 3	>	>	0	1	2							

First-In First-Out, FIFO (6)

Älteste Kachel wird ersetzt

- Notwendige Zustände
  - Alter bzw. Einlagerungszeitpunkt für jede Kachel

Ablauf der Ersetzungen (9 Einlagerungen)

Referenzfolge		1	2	3	4	1	2	5	1	2	3	4	5
Hauptspeicher	Kachel 1	1	1	1	4	4	4						
	Kachel 2		2	2	2	1	1						
	Kachel 3			3	3	3	2						
Kontrollzustände (Alter pro Kachel)	Kachel 1	0	1	2	0	1	2						
	Kachel 2	>	0	1	2	0	1						
	Kachel 3	>	>	0	1	2	0						

First-In First-Out, FIFO (7)

Älteste Kachel wird ersetzt

- Notwendige Zustände
  - Alter bzw. Einlagerungszeitpunkt für jede Kachel

Ablauf der Ersetzungen (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					
	Kachel 2		2	2	2	1	1	1					
	Kachel 3			3	3	3	2	2					
Kontrollzustände (Alter pro Kachel)	Kachel 1	0	1	2	0	1	2	0					
	Kachel 2	>	0	1	2	0	1	2					
	Kachel 3	>	>	0	1	2	0	1					

First-In First-Out, FIFO (8)

Älteste Kachel wird ersetzt

- Notwendige Zustände
  - Alter bzw. Einlagerungszeitpunkt für jede Kachel

Ablauf der Ersetzungen (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				
	Kachel 2		2	2	2	1	1	1	1				
	Kachel 3			3	3	3	2	2	2				
Kontrollzustände (Alter pro Kachel)	Kachel 1	0	1	2	0	1	2	0	1				
	Kachel 2	>	0	1	2	0	1	2	3				
	Kachel 3	>	>	0	1	2	0	1	2				

First-In First-Out, FIFO (9)

Älteste Kachel wird ersetzt

- Notwendige Zustände
  - Alter bzw. Einlagerungszeitpunkt für jede Kachel

Ablauf der Ersetzungen (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			
	Kachel 2		2	2	2	1	1	1	1	1			
	Kachel 3			3	3	3	2	2	2	2			
Kontrollzustände (Alter pro Kachel)	Kachel 1	0	1	2	0	1	2	0	1	2			
	Kachel 2	>	0	1	2	0	1	2	3	4			
	Kachel 3	>	>	0	1	2	0	1	2	3			

First-In First-Out, FIFO (10)

Älteste Kachel wird ersetzt

- Notwendige Zustände
  - Alter bzw. Einlagerungszeitpunkt für jede Kachel

Ablauf der Ersetzungen (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		
	Kachel 2		2	2	2	1	1	1	1	1	3		
	Kachel 3			3	3	3	2	2	2	2	2		
Kontrollzustände (Alter pro Kachel)	Kachel 1	0	1	2	0	1	2	0	1	2	3		
	Kachel 2	>	0	1	2	0	1	2	3	4	0		
	Kachel 3	>	>	0	1	2	0	1	2	3	4		

First-In First-Out, FIFO (11)

Älteste Kachel wird ersetzt

– Notwendige Zustände
  – Alter bzw. Einlagerungszeitpunkt für jede Kachel

Ablauf der Ersetzungen (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 (Alter pro Kachel)	Kachel 1	0	1	2	0	1	2	0	1	2	3	4	
	Kachel 2	>	0	1	2	0	1	2	3	4	0	1	
	Kachel 3	>	>	0	1	2	0	1	2	3	4	0	

First-In First-Out, FIFO (12)

Älteste Kachel wird ersetzt

- Notwendige Zustände
  - Alter bzw. Einlagerungszeitpunkt für jede Kachel

Ablauf der Ersetzungen (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
Kontrollzustände (Alter pro Kachel)	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 (13)

Größerer Hauptspeicher mit 4 Kacheln (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 pro Kachel)	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

☞ *Vergrößern des Hauptspeichers kann zu mehr Einlagerungen führen*

☞ *FIFO-Anomalie (Belady´s Anomalie, 1969)*

Optimale Ersetzungsstrategie (1)

Ersetze immer die Seite mit dem größten Vorwärtsabstand

- Vorwärtsabstand: Zeitdauer bis zum nächsten Zugriff auf eine Seite
- Strategie B_0 (OPT oder MIN) ist optimal bei fester Kachelmenge: minimale Anzahl von Ersetzungen (hier: 7 Einlagerungen)

Referenzfolge		1	2	3	4	1	2	5	1	2	3	4	5
Hauptspeicher	Kachel 1	1											
	Kachel 2												
	Kachel 3												
Kontrollzustände (Vorwärts-abstand)	Kachel 1	4											
	Kachel 2	>											
	Kachel 3	>											

Optimale Ersetzungsstrategie (2)

Ersetze immer die Seite mit dem größten Vorwärtsabstand

- Vorwärtsabstand: Zeitdauer bis zum nächsten Zugriff auf eine Seite
- Strategie B_0 (OPT oder MIN) ist optimal bei fester Kachelmenge: minimale Anzahl von Ersetzungen (hier: 7 Einlagerungen)

Referenzfolge		1	2	3	4	1	2	5	1	2	3	4	5
Hauptspeicher	Kachel 1	1	1										
	Kachel 2		2										
	Kachel 3												
Kontrollzustände (Vorwärts-abstand)	Kachel 1	4	3										
	Kachel 2	>	4										
	Kachel 3	>	>										

Optimale Ersetzungsstrategie (3)

Ersetze immer die Seite mit dem größten Vorwärtsabstand

- Vorwärtsabstand: Zeitdauer bis zum nächsten Zugriff auf eine Seite
- Strategie B_0 (OPT oder MIN) ist optimal bei fester Kachelmenge: minimale Anzahl von Ersetzungen (hier: 7 Einlagerungen)

Referenzfolge		1	2	3	4	1	2	5	1	2	3	4	5
Hauptspeicher	Kachel 1	1	1	1									
	Kachel 2		2	2									
	Kachel 3			3									
Kontrollzustände (Vorwärts-abstand)	Kachel 1	4	3	2									
	Kachel 2	>	4	3									
	Kachel 3	>	>	7									

Optimale Ersetzungsstrategie (4)

Ersetze immer die Seite mit dem größten Vorwärtsabstand

- Vorwärtsabstand: Zeitdauer bis zum nächsten Zugriff auf eine Seite
- Strategie B_0 (OPT oder MIN) ist optimal bei fester Kachelmenge: minimale Anzahl von Ersetzungen (hier: 7 Einlagerungen)

Referenzfolge		1	2	3	4	1	2	5	1	2	3	4	5
Hauptspeicher	Kachel 1	1	1	1	1								
	Kachel 2		2	2	2								
	Kachel 3			3	4								
Kontrollzustände (Vorwärts-abstand)	Kachel 1	4	3	2	1								
	Kachel 2	>	4	3	2								
	Kachel 3	>	>	7	7								

Optimale Ersetzungsstrategie (5)

Ersetze immer die Seite mit dem größten Vorwärtsabstand

- Vorwärtsabstand: Zeitdauer bis zum nächsten Zugriff auf eine Seite
- Strategie B_0 (OPT oder MIN) ist optimal bei fester Kachelmenge: minimale Anzahl von Ersetzungen (hier: 7 Einlagerungen)

Referenzfolge		1	2	3	4	1	2	5	1	2	3	4	5
Hauptspeicher	Kachel 1	1	1	1	1	1							
	Kachel 2		2	2	2	2							
	Kachel 3			3	4	4							
Kontrollzustände (Vorwärts-abstand)	Kachel 1	4	3	2	1	3							
	Kachel 2	>	4	3	2	1							
	Kachel 3	>	>	7	7	6							

Optimale Ersetzungsstrategie (6)

Ersetze immer die Seite mit dem größten Vorwärtsabstand

- Vorwärtsabstand: Zeitdauer bis zum nächsten Zugriff auf eine Seite
- Strategie B_0 (OPT oder MIN) ist optimal bei fester Kachelmenge: minimale Anzahl von Ersetzungen (hier: 7 Einlagerungen)

Referenzfolge		1	2	3	4	1	2	5	1	2	3	4	5
Hauptspeicher	Kachel 1	1	1	1	1	1	1						
	Kachel 2		2	2	2	2	2						
	Kachel 3			3	4	4	4						
Kontrollzustände (Vorwärts-abstand)	Kachel 1	4	3	2	1	3	2						
	Kachel 2	>	4	3	2	1	3						
	Kachel 3	>	>	7	7	6	5						

Optimale Ersetzungsstrategie (7)

Ersetze immer die Seite mit dem größten Vorwärtsabstand

- Vorwärtsabstand: Zeitdauer bis zum nächsten Zugriff auf eine Seite
- Strategie B_0 (OPT oder MIN) ist optimal bei fester Kachelmenge: minimale Anzahl von Ersetzungen (hier: 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					
	Kachel 2		2	2	2	2	2	2					
	Kachel 3			3	4	4	4	5					
Kontrollzustände (Vorwärts-abstand)	Kachel 1	4	3	2	1	3	2	1					
	Kachel 2	>	4	3	2	1	3	2					
	Kachel 3	>	>	7	7	6	5	5					

Optimale Ersetzungsstrategie (8)

Ersetze immer die Seite mit dem größten Vorwärtsabstand

- Vorwärtsabstand: Zeitdauer bis zum nächsten Zugriff auf eine Seite
- Strategie B_0 (OPT oder MIN) ist optimal bei fester Kachelmenge: minimale Anzahl von Ersetzungen (hier: 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				
	Kachel 2		2	2	2	2	2	2	2				
	Kachel 3			3	4	4	4	5	5				
Kontrollzustände (Vorwärts-abstand)	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				

Optimale Ersetzungsstrategie (9)

Ersetze immer die Seite mit dem größten Vorwärtsabstand

- Vorwärtsabstand: Zeitdauer bis zum nächsten Zugriff auf eine Seite
- Strategie B_0 (OPT oder MIN) ist optimal bei fester Kachelmenge: minimale Anzahl von Ersetzungen (hier: 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			
	Kachel 2		2	2	2	2	2	2	2	2			
	Kachel 3			3	4	4	4	5	5	5			
Kontrollzustände (Vorwärts-abstand)	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			

Optimale Ersetzungsstrategie (10)

Ersetze immer die Seite mit dem größten Vorwärtsabstand

- Vorwärtsabstand: Zeitdauer bis zum nächsten Zugriff auf eine Seite
- Strategie B_0 (OPT oder MIN) ist optimal bei fester Kachelmenge: minimale Anzahl von Ersetzungen (hier: 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		
	Kachel 2		2	2	2	2	2	2	2	2	2		
	Kachel 3			3	4	4	4	5	5	5	5		
Kontrollzustände (Vorwärts-abstand)	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		

Optimale Ersetzungsstrategie (11)

Ersetze immer die Seite mit dem größten Vorwärtsabstand

- Vorwärtsabstand: Zeitdauer bis zum nächsten Zugriff auf eine Seite
- Strategie B_0 (OPT oder MIN) ist optimal bei fester Kachelmenge: minimale Anzahl von Ersetzungen (hier: 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	
	Kachel 2		2	2	2	2	2	2	2	2	2	2	
	Kachel 3			3	4	4	4	5	5	5	5	5	
Kontrollzustände (Vorwärts-abstand)	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 Ersetzungsstrategie (12)

Ersetze immer die Seite mit dem größten Vorwärtsabstand

- Vorwärtsabstand: Zeitdauer bis zum nächsten Zugriff auf eine Seite
- Strategie B_0 (OPT oder MIN) ist optimal bei fester Kachelmenge: minimale Anzahl von Ersetzungen (hier: 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 (Vorwärts-abstand)	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 Ersetzungsstrategie (13)

Vergrößerung des Hauptspeichers (4 Kacheln): 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	2	2
	Kachel 3			3	3	3	3	3	3	3	3	3	3
	Kachel 4				4	4	4	5	5	5	5	5	5
Kontrollzustände (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	>

☞ *Keine Anomalie*

Optimale Ersetzungsstrategie (14)

Implementierung von B_0 nahezu unmöglich

– Referenzfolge müsste vorher bekannt sein

– B_0 meist nur zum Vergleich von Ersetzungsstrategien brauchbar

Suche nach Strategien, die möglichst nahe an B_0 heran kommen

– Z.B. *Least Recently Used (LRU)*

Least Recently Used, LRU (1)

Ersetze immer die Seite mit dem größten Rückwärtsabstand

– Rückwärtsabstand: Zeitdauer seit dem letzten Zugriff auf eine Seite

– LRU-Strategie (hier: 10 Einlagerungen)

Referenzfolge		1	2	3	4	1	2	5	1	2	3	4	5
Hauptspeicher	Kachel 1	1											
	Kachel 2												
	Kachel 3												
Kontrollzustände (Rückwärts-abstand)	Kachel 1	0											
	Kachel 2	>											
	Kachel 3	>											

Least Recently Used, LRU (2)

Ersetze immer die Seite mit dem größten Rückwärtsabstand

– Rückwärtsabstand: Zeitdauer seit dem letzten Zugriff auf eine Seite

– LRU-Strategie (hier: 10 Einlagerungen)

Referenzfolge		1	2	3	4	1	2	5	1	2	3	4	5
Hauptspeicher	Kachel 1	1	1										
	Kachel 2		2										
	Kachel 3												
Kontrollzustände (Rückwärts-abstand)	Kachel 1	0	1										
	Kachel 2	>	0										
	Kachel 3	>	>										

Least Recently Used, LRU (3)

Ersetze immer die Seite mit dem größten Rückwärtsabstand

– Rückwärtsabstand: Zeitdauer seit dem letzten Zugriff auf eine Seite

– LRU-Strategie (hier: 10 Einlagerungen)

Referenzfolge		1	2	3	4	1	2	5	1	2	3	4	5
Hauptspeicher	Kachel 1	1	1	1									
	Kachel 2		2	2									
	Kachel 3			3									
Kontrollzustände (Rückwärts-abstand)	Kachel 1	0	1	2									
	Kachel 2	>	0	1									
	Kachel 3	>	>	0									

Least Recently Used, LRU (4)

Ersetze immer die Seite mit dem größten Rückwärtsabstand

– Rückwärtsabstand: Zeitdauer seit dem letzten Zugriff auf eine Seite

– LRU-Strategie (hier: 10 Einlagerungen)

Referenzfolge		1	2	3	4	1	2	5	1	2	3	4	5
Hauptspeicher	Kachel 1	1	1	1	4								
	Kachel 2		2	2	2								
	Kachel 3			3	3								
Kontrollzustände (Rückwärts-abstand)	Kachel 1	0	1	2	0								
	Kachel 2	>	0	1	2								
	Kachel 3	>	>	0	1								

Least Recently Used, LRU (5)

Ersetze immer die Seite mit dem größten Rückwärtsabstand

– Rückwärtsabstand: Zeitdauer seit dem letzten Zugriff auf eine Seite

– LRU-Strategie (hier: 10 Einlagerungen)

Referenzfolge		1	2	3	4	1	2	5	1	2	3	4	5
Hauptspeicher	Kachel 1	1	1	1	4	4							
	Kachel 2		2	2	2	1							
	Kachel 3			3	3	3							
Kontrollzustände (Rückwärts-abstand)	Kachel 1	0	1	2	0	1							
	Kachel 2	>	0	1	2	0							
	Kachel 3	>	>	0	1	2							

Least Recently Used, LRU (6)

Ersetze immer die Seite mit dem größten Rückwärtsabstand

– Rückwärtsabstand: Zeitdauer seit dem letzten Zugriff auf eine Seite

– LRU-Strategie (hier: 10 Einlagerungen)

Referenzfolge		1	2	3	4	1	2	5	1	2	3	4	5
Hauptspeicher	Kachel 1	1	1	1	4	4	4						
	Kachel 2		2	2	2	1	1						
	Kachel 3			3	3	3	2						
Kontrollzustände (Rückwärts-abstand)	Kachel 1	0	1	2	0	1	2						
	Kachel 2	>	0	1	2	0	1						
	Kachel 3	>	>	0	1	2	0						

Least Recently Used, LRU (7)

Ersetze immer die Seite mit dem größten Rückwärtsabstand

– Rückwärtsabstand: Zeitdauer seit dem letzten Zugriff auf eine Seite

– LRU-Strategie (hier: 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					
	Kachel 2		2	2	2	1	1	1					
	Kachel 3			3	3	3	2	2					
Kontrollzustände (Rückwärts-abstand)	Kachel 1	0	1	2	0	1	2	0					
	Kachel 2	>	0	1	2	0	1	2					
	Kachel 3	>	>	0	1	2	0	1					

Least Recently Used, LRU (8)

Ersetze immer die Seite mit dem größten Rückwärtsabstand

– Rückwärtsabstand: Zeitdauer seit dem letzten Zugriff auf eine Seite

– LRU-Strategie (hier: 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				
	Kachel 2		2	2	2	1	1	1	1				
	Kachel 3			3	3	3	2	2	2				
Kontrollzustände (Rückwärts-abstand)	Kachel 1	0	1	2	0	1	2	0	1				
	Kachel 2	>	0	1	2	0	1	2	0				
	Kachel 3	>	>	0	1	2	0	1	2				

Least Recently Used, LRU (9)

Ersetze immer die Seite mit dem größten Rückwärtsabstand

– Rückwärtsabstand: Zeitdauer seit dem letzten Zugriff auf eine Seite

– LRU-Strategie (hier: 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			
	Kachel 2		2	2	2	1	1	1	1	1			
	Kachel 3			3	3	3	2	2	2	2			
Kontrollzustände (Rückwärts-abstand)	Kachel 1	0	1	2	0	1	2	0	1	2			
	Kachel 2	>	0	1	2	0	1	2	0	1			
	Kachel 3	>	>	0	1	2	0	1	2	0			

Least Recently Used, LRU (10)

Ersetze immer die Seite mit dem größten Rückwärtsabstand

– Rückwärtsabstand: Zeitdauer seit dem letzten Zugriff auf eine Seite

– LRU-Strategie (hier: 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		
	Kachel 2		2	2	2	1	1	1	1	1	1		
	Kachel 3			3	3	3	2	2	2	2	2		
Kontrollzustände (Rückwärts-abstand)	Kachel 1	0	1	2	0	1	2	0	1	2	0		
	Kachel 2	>	0	1	2	0	1	2	0	1	2		
	Kachel 3	>	>	0	1	2	0	1	2	0	1		

Least Recently Used, LRU (11)

Ersetze immer die Seite mit dem größten Rückwärtsabstand

– Rückwärtsabstand: Zeitdauer seit dem letzten Zugriff auf eine Seite

– LRU-Strategie (hier: 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	
	Kachel 2		2	2	2	1	1	1	1	1	1	4	
	Kachel 3			3	3	3	2	2	2	2	2	2	
Kontrollzustände (Rückwärts-abstand)	Kachel 1	0	1	2	0	1	2	0	1	2	0	1	
	Kachel 2	>	0	1	2	0	1	2	0	1	2	0	
	Kachel 3	>	>	0	1	2	0	1	2	0	1	2	

Least Recently Used, LRU (12)

Ersetze immer die Seite mit dem größten Rückwärtsabstand

– Rückwärtsabstand: Zeitdauer seit dem letzten Zugriff auf eine Seite

– LRU-Strategie (hier: 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 (Rückwärts-abstand)	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 (13)

Vergrößerung des Hauptspeichers (4 Kacheln): 8 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	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
	Kachel 4				4	4	4	4	4	4	3	3	3
Kontrollzustände (Rückwärts-abstand)	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
	Kachel 4	>	>	>	0	1	2	3	4	5	0	1	2

Least Recently Used, LRU (14)

Keine Anomalie

- Allgemein gilt:
  Es gibt eine Klasse von Algorithmen (*Stack*-Algorithmen), bei denen keine Anomalie auftritt:
  - Bei *Stack*-Algorithmen ist bei *n* Kacheln zu jedem Zeitpunkt eine Untermenge der Seiten eingelagert, die bei *n*+1 Kacheln zum gleichen Zeitpunkt eingelagert wären.
  - LRU: Es sind immer die letzten *n* benutzten Seiten eingelagert
  - B_0: Es sind die *n* bereits benutzten Seiten eingelagert, auf die als nächstes zugegriffen wird

Problem

- Implementierung von LRU nicht ohne Hardwareunterstützung möglich
- Es muss jeder Speicherzugriff berücksichtigt werden

Least Recently Used, LRU (15)

Hardwareunterstützung durch Zähler

– CPU besitzt einen Zähler, der bei jedem Speicherzugriff erhöht wird (inkrementiert wird)
– Bei jedem Seitenzugriff wird der aktuelle Zählerwert in den jeweiligen Seitendeskriptor geschrieben
– Auswahl der Seite mit dem kleinsten Zählerstand

Aufwendige Implementierung

– Viele zusätzliche Speicherzugriffe

Second Chance, Clock (1)

Einsatz von Referenzbits

– Alle Seiten sind gedanklich im Kreis auf einem Ziffernblatt angeordnet, ein Zeiger **p** wird „im Uhrzeigersinn“ weiterbewegt und zeigt den zu ersetzenden Eintrag an.

– Jede Seite hat ein Referenzbit (*used bit)*, das automatisch durch die Hardware auf ‚1‘ gesetzt wird, wann immer auf die Seite zugegriffen wird (bspw. die *access bits* der Pentium MMUs).
Das *Clock*-Verfahren sucht nach Eintrag mit *used bit* = ‚0‘ bei gleichzeitigem Rücksetzen des Bits. Stop nach max. 1 Runde

Second Chance, Clock (2)

Clock-Algorithmus

a) Wenn zu ersetzende Seite gesucht wird:

b) Zeigt Zeiger **p** auf Eintrag mit *used bit* = ‚1‘?

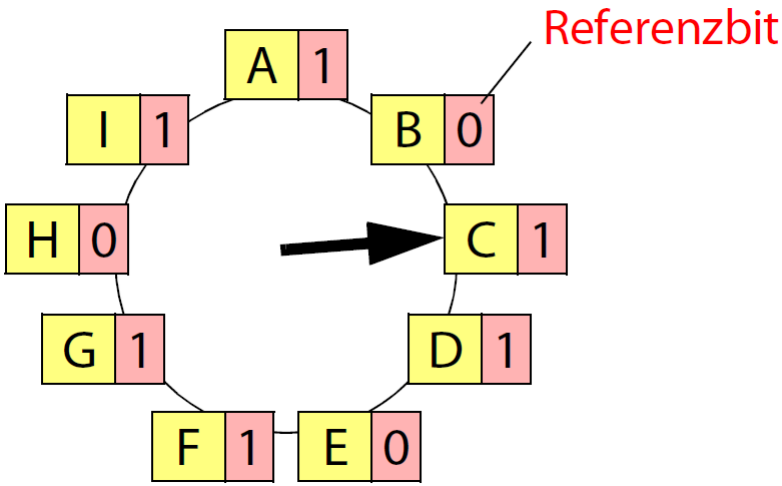
c) Falls ja: Setze *used bit* des Eintrags **p** auf ‚0‘; bewege Zeiger um eine Position im Uhrzeigersinn weiter (**p = p + 1**); gehe zu b)

d) Falls nein: Bewege Zeiger um eine Position im Uhrzeigersinn weiter (**p = p + 1**); gebe *alten* Zeigerwert **p-1** als zu ersetzenden Eintrag zurück

Wichtig: Wird auf eine Seite zugegriffen, die bereits im Hauptspeicher vorhanden ist, wird obiger Algorithmus nicht ausgeführt. Das *used bit* der Seite wird automatisch durch die Hardware auf ‚1‘ gesetzt, und der („Uhr-“) Zeiger **p** wird *nicht* verändert.

Second Chance, Clock (3)

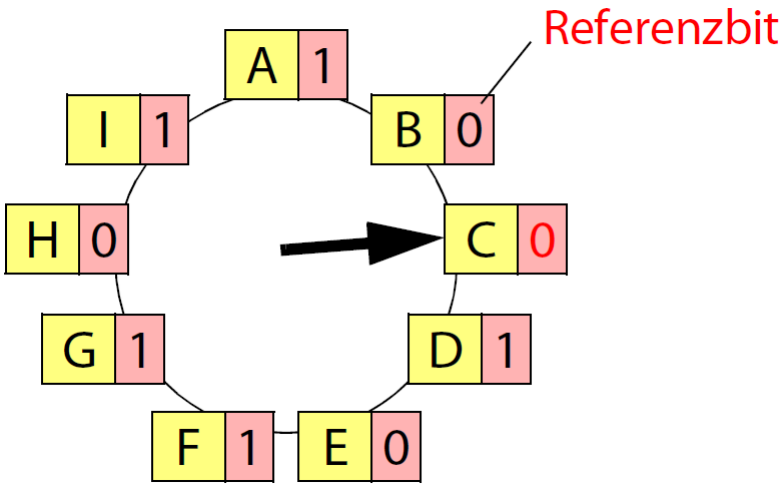
Implementierung mit umlaufenden Zeiger *(Clock)*



- An der Zeigerposition wird Referenzbit getestet
  - Falls Referenzbit 1, wird Bit gelöscht
  - Falls Referenzbit 0, wurde ersetzbare Seite gefunden
  - Zeiger wird weitergestellt; falls keine Seite gefunden: Wiederholung
- Falls alle Referenzbits auf 1 stehen, wird *Second Chance* zu FIFO

Second Chance, Clock (4)

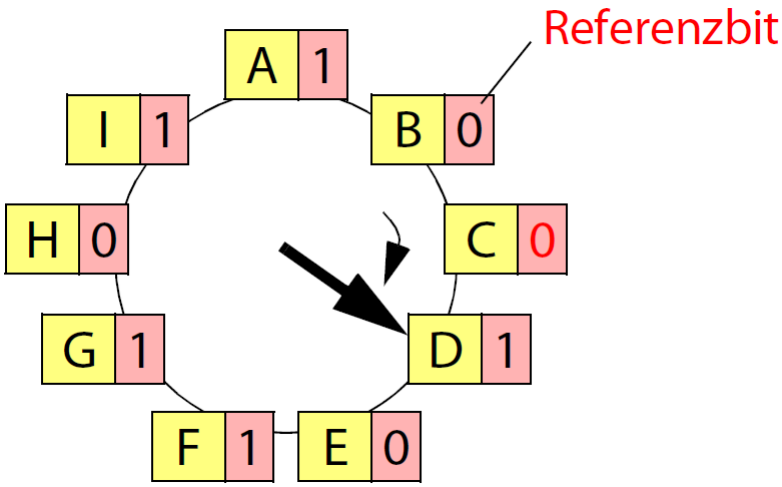
Implementierung mit umlaufenden Zeiger *(Clock)*



- An der Zeigerposition wird Referenzbit getestet
  - Falls Referenzbit 1, wird Bit gelöscht
  - Falls Referenzbit 0, wurde ersetzbare Seite gefunden
  - Zeiger wird weitergestellt; falls keine Seite gefunden: Wiederholung
- Falls alle Referenzbits auf 1 stehen, wird *Second Chance* zu FIFO

Second Chance, Clock (5)

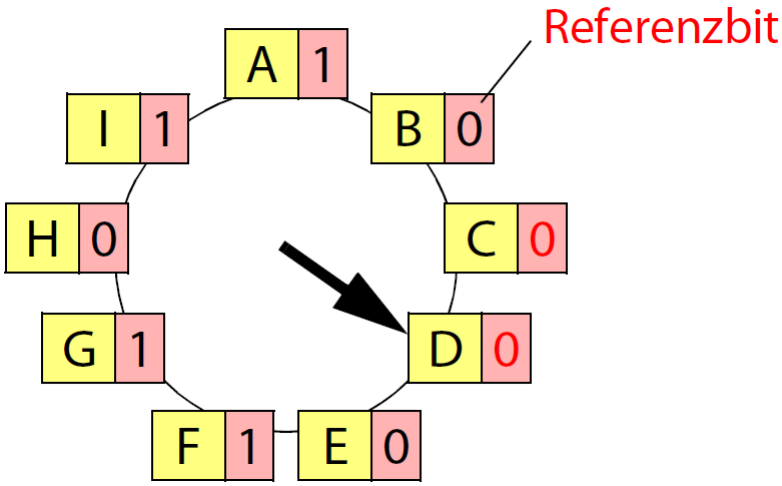
Implementierung mit umlaufenden Zeiger *(Clock)*



- An der Zeigerposition wird Referenzbit getestet
  - Falls Referenzbit 1, wird Bit gelöscht
  - Falls Referenzbit 0, wurde ersetzbare Seite gefunden
  - Zeiger wird weitergestellt; falls keine Seite gefunden: Wiederholung
- Falls alle Referenzbits auf 1 stehen, wird *Second Chance* zu FIFO

Second Chance, Clock (6)

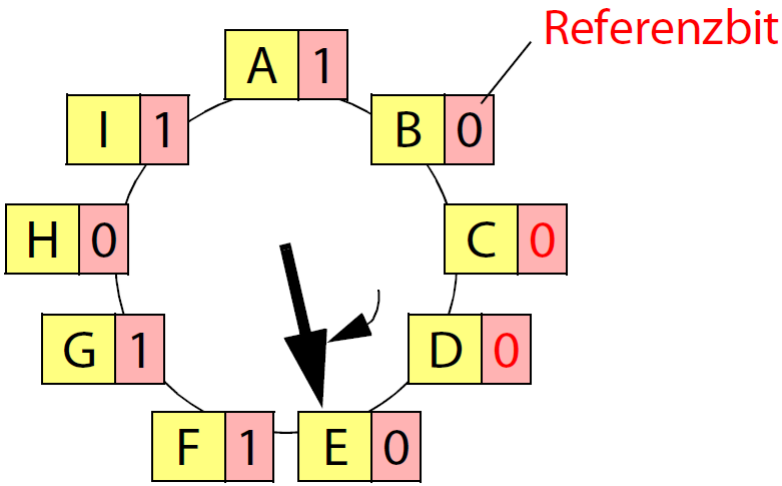
Implementierung mit umlaufenden Zeiger *(Clock)*



- An der Zeigerposition wird Referenzbit getestet
  - Falls Referenzbit 1, wird Bit gelöscht
  - Falls Referenzbit 0, wurde ersetzbare Seite gefunden
  - Zeiger wird weitergestellt; falls keine Seite gefunden: Wiederholung
- Falls alle Referenzbits auf 1 stehen, wird *Second Chance* zu FIFO

Second Chance, Clock (7)

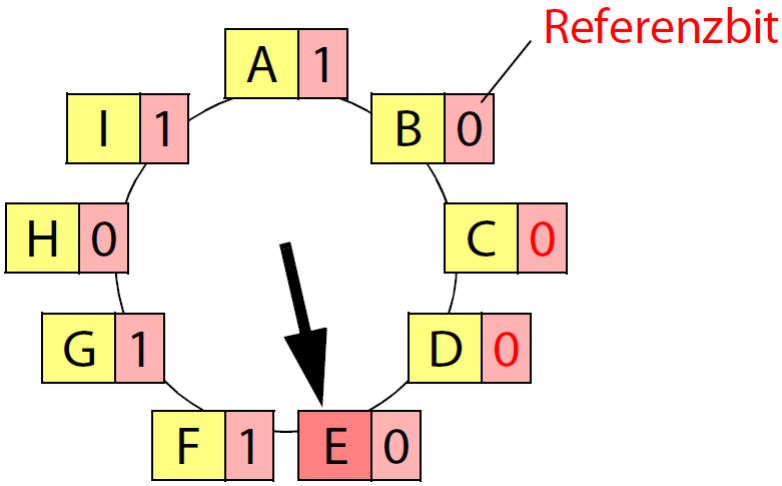
Implementierung mit umlaufenden Zeiger *(Clock)*



- An der Zeigerposition wird Referenzbit getestet
  - Falls Referenzbit 1, wird Bit gelöscht
  - Falls Referenzbit 0, wurde ersetzbare Seite gefunden
  - Zeiger wird weitergestellt; falls keine Seite gefunden: Wiederholung
- Falls alle Referenzbits auf 1 stehen, wird *Second Chance* zu FIFO

Second Chance, Clock (8)

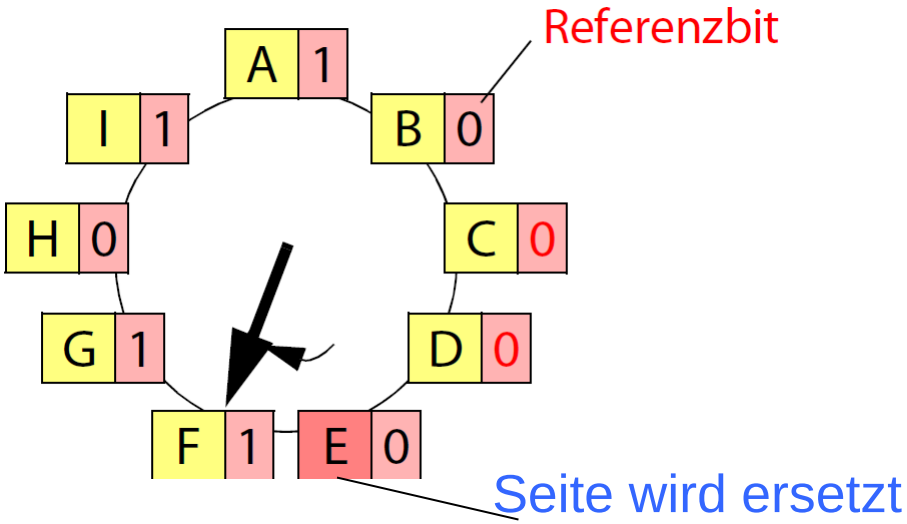
Implementierung mit umlaufenden Zeiger *(Clock)*



- An der Zeigerposition wird Referenzbit getestet
  - Falls Referenzbit 1, wird Bit gelöscht
  - Falls Referenzbit 0, wurde ersetzbare Seite gefunden
  - Zeiger wird weitergestellt; falls keine Seite gefunden: Wiederholung
- Falls alle Referenzbits auf 1 stehen, wird *Second Chance* zu FIFO

Second Chance, Clock (9)

Implementierung mit umlaufenden Zeiger *(Clock)*



- An der Zeigerposition wird Referenzbit getestet
  - Falls Referenzbit 1, wird Bit gelöscht
  - Falls Referenzbit 0, wurde ersetzbare Seite gefunden
  - Zeiger wird weitergestellt; falls keine Seite gefunden: Wiederholung
- Falls alle Referenzbits auf 1 stehen, wird *Second Chance* zu FIFO

Second Chance, Clock (10)

Ablauf bei 3 Kacheln: 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 (Referenzbits)	Kachel 1	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	1
	Kachel 3	0	0	1	0	0	1	0	0	1	0	1	1
	Umlaufzeiger	2	3	1	2	3	1	2	2	2	3	1	1

Second Chance, Clock (11)

Ablauf bei 4 Kacheln: 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
Kontroll-zustände (Refernzbits)	Kachel 1	1	1	1	1	1	1	1	1	1	1	1	1
	Kachel 2	0	1	1	1	1	1	0	1	1	1	0	1
	Kachel 3	0	0	1	1	1	1	0	0	1	1	0	0
	Kachel 4	0	0	0	1	1	1	0	0	0	1	0	0
	Umlaufzeiger	2	3	4	1	1	1	2	3	4	1	2	3

Second Chance, Clock (12)

Second Chance zeigt FIFO-Anomalie

– Wenn alle Referenzbits gleich 1 sind, wird nach FIFO entschieden

Erweiterung

– Modifikationsbit kann zusätzlich berücksichtigt werden *(dirty bit)*

– Einteilung von Speicherzugriffen in drei Klassen mittels Referenz- und *dirty bit* (RB bzw. DB):

RB = 0, DB = 0	Klasse 1
RB = 1, DB = 0	Klasse 2
RB = 1, DB = 1	Klasse 3

– Klasse 4 (RB = 0, DB = 1) nicht möglich

– Suche nach zu ersetzender Kachel in der niedrigsten nicht-leeren Klasse (Einsatz in MacOS)

Freiseitenpuffer

Statt eine Seite zu ersetzen, wird permanent eine Menge freier Seiten gehalten

- Auslagerung geschieht „im Voraus“
- Effizienter: Ersetzungszeit besteht im Wesentlichen nur aus der Einlagerungszeit

Behalten der Seitenzuordnung auch nach der Auslagerung

- Wird die Seite doch noch benutzt, bevor sie durch eine andere ersetzt wird, kann sie mit hoher Effizienz wiederverwendet werden.
- Seite wird aus Freiseitenpuffer ausgetragen und wieder dem entsprechenden Prozess zugeordnet

Seitenadressierung und Mehrprogrammbetrieb (1)

Problem: Verteilung der Kacheln unter mehreren Prozessen

Begrenzungen

- Maximale Kachelmenge: begrenzt durch Speicherausbau eines Rechensystems, d.h. durch Anzahl verfügbarer physikalischer Kacheln
- Minimale Kachelmenge: abhängig von der Prozessorarchitektur
  - Mindestens die Anzahl von Kacheln nötig, die theoretisch maximal bei einem Maschinenbefehl benötigt werden
  - Beispiel:
    Kachelgröße: 4 Bytes
    Länge des Befehls: 4 Bytes
    Länge eines durch den Befehl adressierten Datums: 8 Bytes
    ☞ Zwei Kacheln für den Befehl, drei Kacheln für die Daten

Seitenadressierung und Mehrprogrammbetrieb (2)

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
  - Bessere Effizienz, da ungenutzte Kacheln von anderen Prozessen verwendet werden können

Lokale Ersetzungsstrategien

Gleiche Zuordnung

– Anzahl der Prozesse bestimmt die Kachelmenge, die ein Prozess bekommt

Größenabhängige Zuordnung

– Größe des Programms fließt in die zugeteilte Kachelmenge ein

Seitenflattern *(Thrashing)*

Ausgelagerte Seite wird gleich wieder angesprochen

– Prozess verbringt mehr Zeit mit dem Warten auf das Beheben von Seitenfehlern als mit der eigentlichen Ausführung

Ursachen

– Prozess ist nahe am Seitenminimum
– Zu viele Prozesse gleichzeitig im System
– Schlechte Ersetzungsstrategie

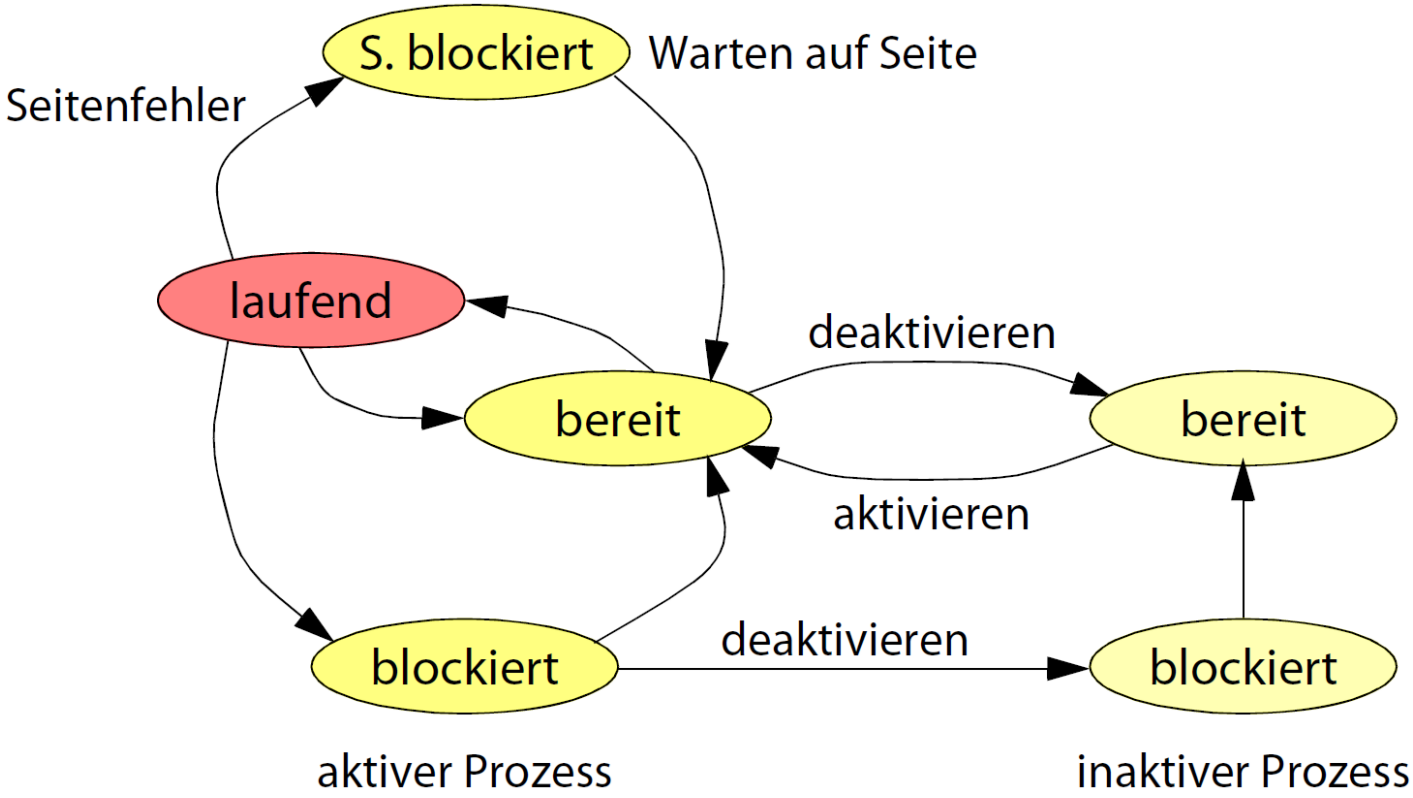
Lokale Ersetzungsstrategie behebt *Thrashing* zwischen Prozessen

☞ **Zuteilung einer genügend großen Zahl von Kacheln behebt *Thrashing* innerhalb der Prozess-Seiten**

  ☞ Begrenzung der Prozessanzahl

Deaktivieren von Prozessen (1)

Einführung von „Superzuständen“



– Inaktiver Prozess benötigt keine Kacheln; Prozess ist vollständig ausgelagert *(swapped out)*

Deaktivieren von Prozessen (2)

Sind zu viele Prozesse aktiv, werden welche deaktiviert

- Kacheln teilen sich auf weniger Prozesse auf
- Verbindung mit dem *Scheduler* nötig
  - Verhindern von Aushungerung
  - Erzielen kurzer Reaktionszeiten
- Guter Kandidat: Prozess mit wenigen Seiten im Hauptspeicher
  - Geringe Latenz bei Wiedereinlagerung bzw. wenige Seitenfehler bei Aktivierung und *Demand Paging*

Zusammenfassung (1)

Speichervergabe

- Problem: Zuteilung des Speichers an Programme (Code & Daten) und an Betriebssystem
- Statische Zuteilung: starre Bereiche für Programme und Betriebssystem, zu unflexibel
- Dynamische Zuteilung: auf Basis von Segmenten
- Freispeicherverwaltung: Bit in Bitvektor repräsentiert Verfügbarkeit einer Speichereinheit; verkettete Liste enthält Einträge für freie und belegte Bereiche
- Strategien zur Speichervergabe: führen zu Speicherverschnitt (Fragmentierung), produzieren u.U. viele kleine Lücken im Speicher

Zusammenfassung (2)

Mehrprogrammbetrieb

- Segmentierung: Umsetzung virtueller in reale Adressen über Segmenttabelle in *Memory Management Unit*
- Kompaktierung: Verschieben von Segmenten zur Reduktion der Fragmentierung
- Seitenadressierung *(Paging)*: keine externe Fragmentierung, nur interne; Größe der Seitentabelle und Anzahl impliziter Speicherzugriffe problematisch (auch bei Segmentierung + *Paging* und bei mehrstufigem *Paging*)
- *Translation Look-Aside Buffer (TLB)*: kleiner, schneller Speicher, der Abbildung virtueller auf reale Adressen zwischenpuffert

Zusammenfassung (3)

Virtueller Speicher

- Konzept: Prozessen wird ein größerer Hauptspeicher vorgetäuscht als tatsächlich real im Rechensystem vorhanden
- *Demand Paging:* Nicht benötigte Seiten werden in Hintergrundspeicher (Festplatte) ausgelagert und erst bei Bedarf wieder eingelagert *(Swapping)*
- Seitenersetzung: Verdrängung einer Kachel, um Platz für neue Seite zu schaffen; FIFO: Belady´s Anomalie; *Stack*-Ersetzungsstrategien (z.B. LRU): weisen keine Anomalie auf; *Clock:* verhält sich u.U. wie FIFO, kann zu Anomalien führen
- *Paging* und Mehrprogrammbetrieb: Prozess-lokale oder globale Seitenersetzung; Deaktivieren von Prozessen durch *Scheduler* um gleichzeitige Nutzung von Kacheln zu verringern