SPEICHERVERWALTUNG

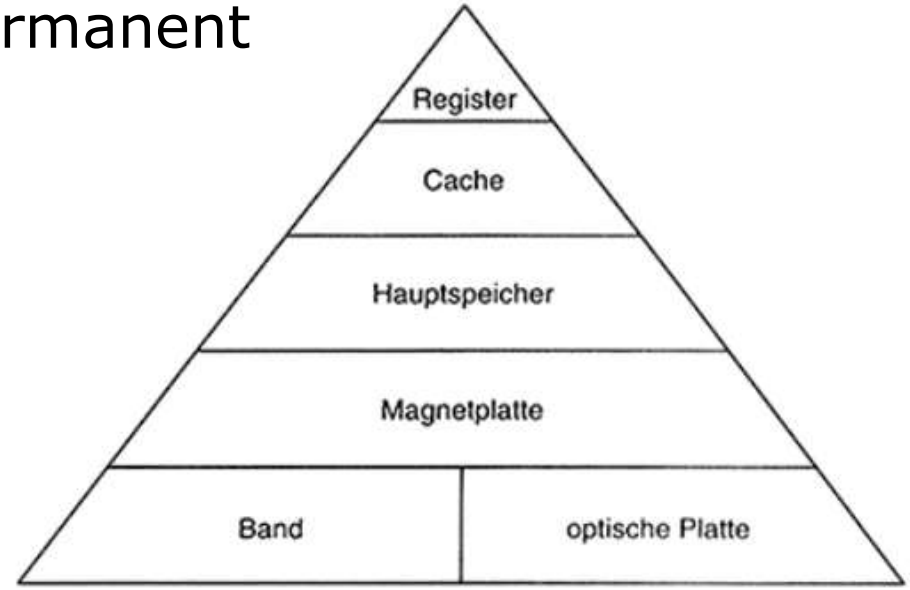
Motivation

- RAM ist eines der wichtigsten Betriebsmittel
- Programme wachsen schneller als der verfügbare Speicher
- Für Programmierer wünschenswert:
  - privaten,
  - unendlich großen,
  - unendlich schnellen Speicher,
  - der dazu noch nicht flüchtig ist.
- Konzept Speicherhierarchie

  -> verwaltet durch Speicherverwaltung (memory manager)

Speicherhierarchie

- **Register**: klein, teuer, schnell, direkte Verbindung in der CPU
- **primärer Cache (1st)**: wenig (8K Instruktionen, Daten), teuer, im CPU-Chip
- **sekundärer Cache (2nd)**: etwas mehr (bis zu 24 MB), rel. teuer, im CPU-Chip oder extern
- **Arbeitsspeicher** RAM: viel (bis zu 256 GB), billig, langsam, weit entfernt von der CPU
- **Massenspeicher**: riesig, sehr billig, sehr langsam, sehr weit entfernt von der CPU, permanent

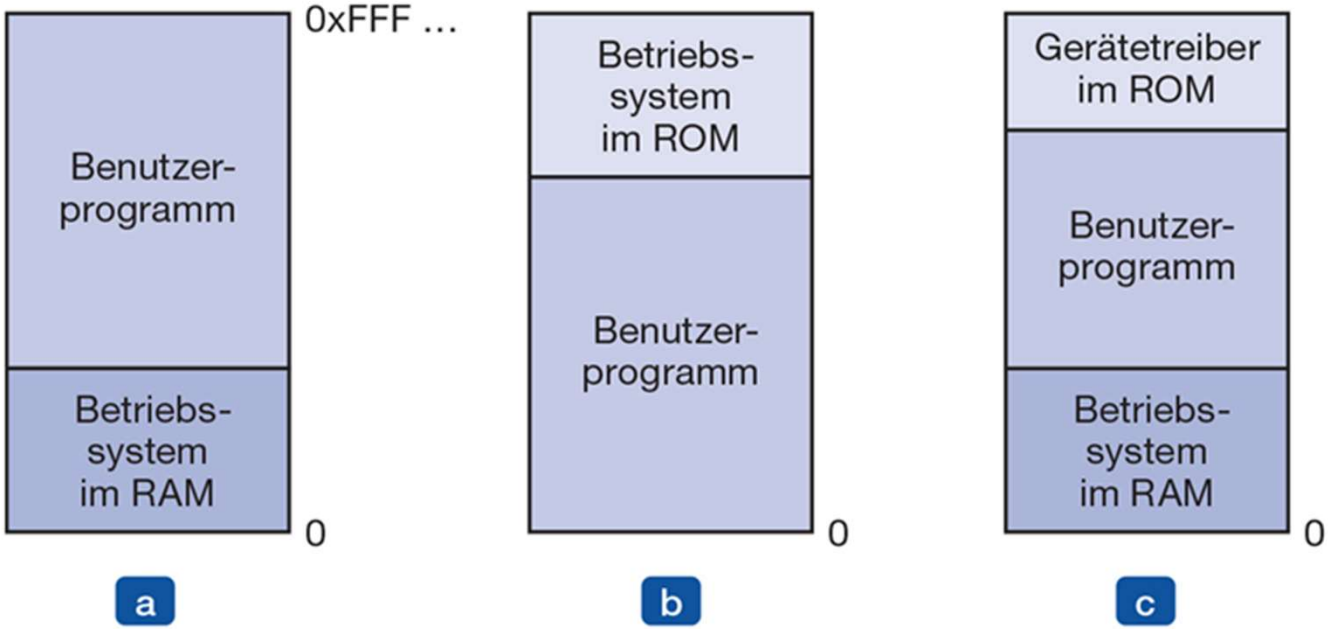


Aufgabe der Speicherverwaltung

- Hauptspeicher effizient zu verwalten, d.h.
  - den Prozessen **zuzuteilen**,
  - evtl. **verschieben** und
  - danach wieder **freizugeben**.
- Bei mehreren Prozessen gleichzeitig
  - **Relokation** von Programmen (Anpassung von absoluten Adressen bei Sprüngen und sonstigen Speicherzugriffen)
  - Gegenseitiger **Speicherschutz** von Prozessen
  - Prozessen **mehr (virtuellen) Hauptspeicher** bieten können, als physikalisch vorhanden

Systeme ohne Speicherabstraktion

- Logische Adresse entspricht physikalischer Adresse
- Keine Möglichkeit für Multiprogramming
- Systeme
  a) Frühere Großrechner und Minicomputer
  b) embedded Systems
  c) frühere PCs (z.B. MS-DOS)



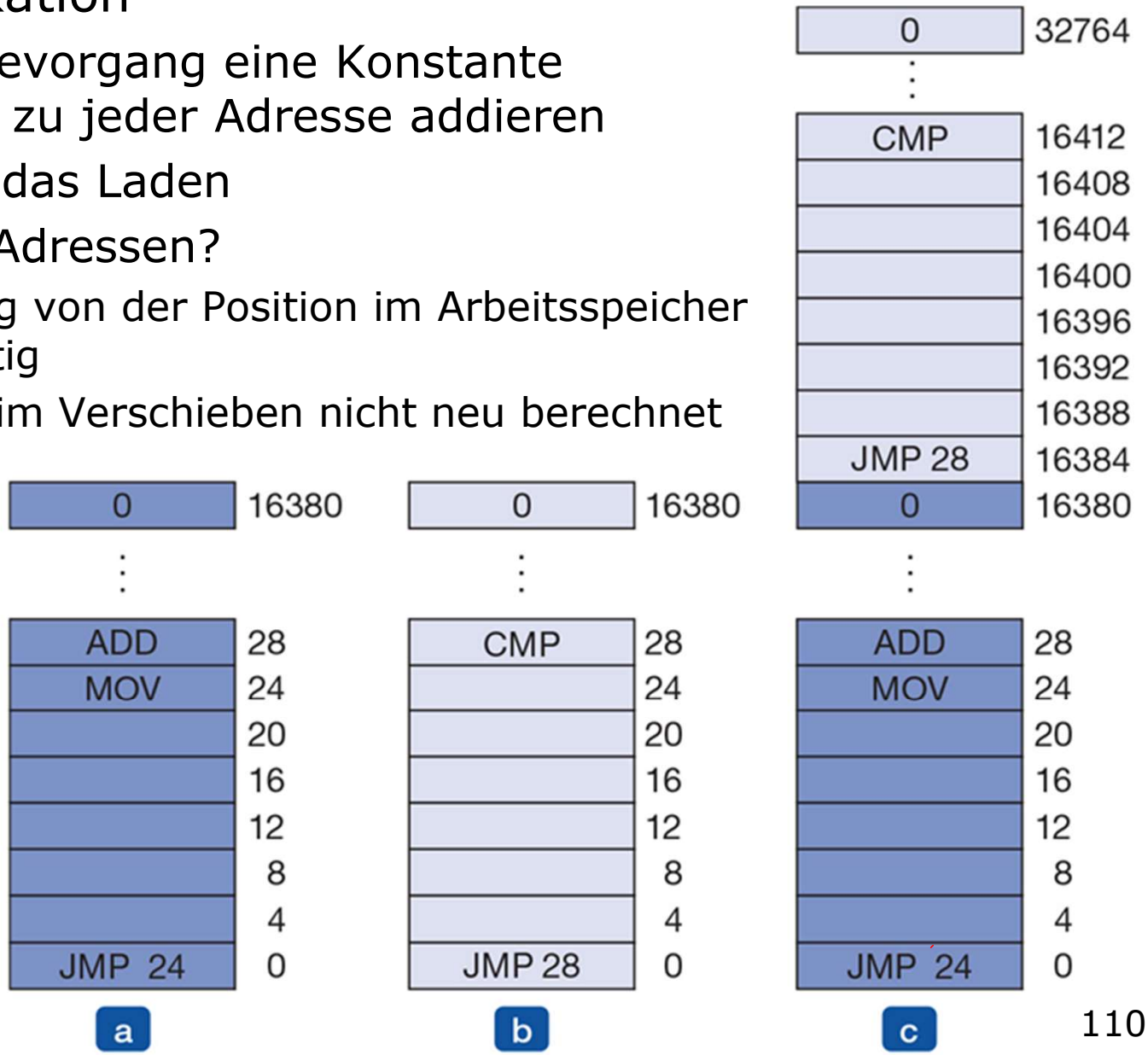
Systeme ohne Speicherabstraktion

Parallelität

- Durch Programmierung von mehreren Threads
  - Alle Threads gleicher Speicherbereich
  - Nachteil: keine voneinander *unabhängigen* Prozesse
- Auslagern des gesamten Speichers in Plattendatei
  - Swapping

Mehrere Programme Relokationsproblem

- statische Relokation
  - o während Ladevorgang eine Konstante (hier:16384) zu jeder Adresse addieren
  - – verlangsamt das Laden
  - – relozierbare Adressen?
    - – unabhängig von der Position im Arbeitsspeicher immer gültig
    - – werden beim Verschieben nicht neu berechnet



Mehrere Programme mit Speicherabstraktion

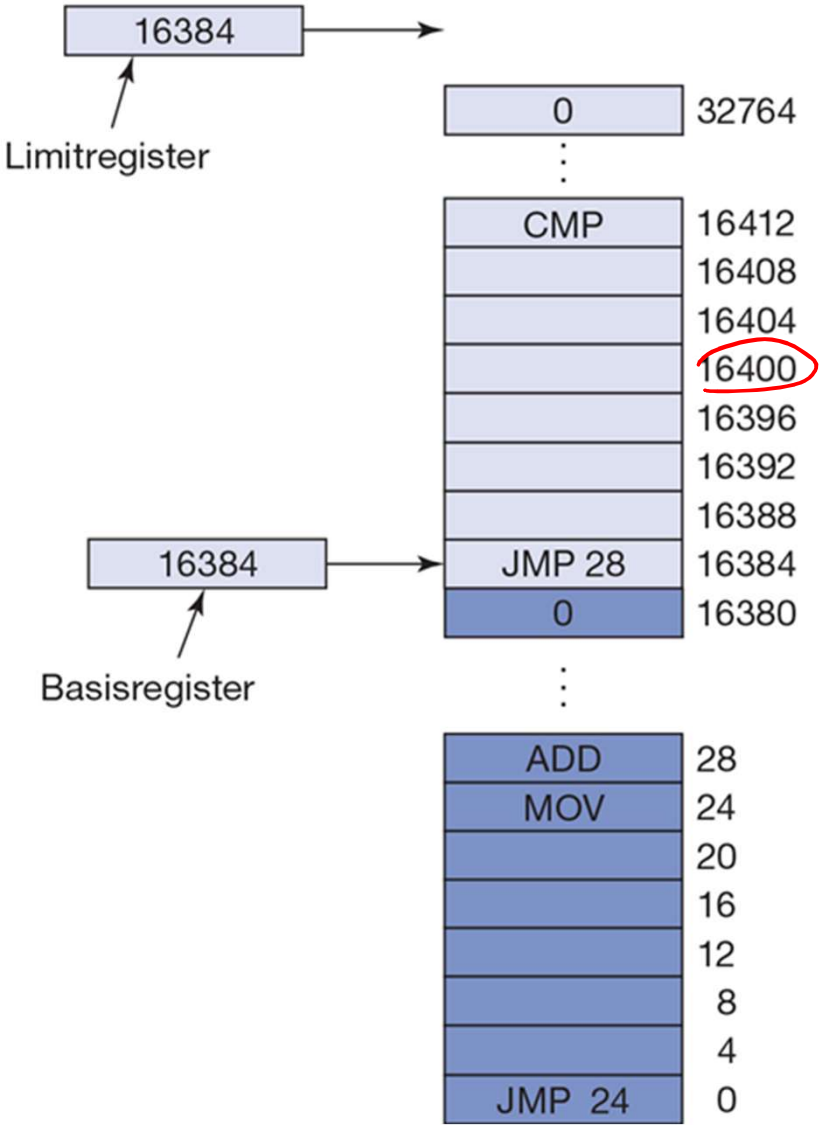
- Speicherabstraktion durch Adressraum (address space)
  - Menge von Adressen, die ein Prozess zur Adressierung des Speichers benutzen kann. -> abstrakter Speicher
  - Jeder Prozess hat seinen eigenen Adressraum.
  - Lösung zu **Schutz** und **Relokationsproblem.**

Mehrere Programme mit Speicherabstraktion

Einfacher Ansatz:

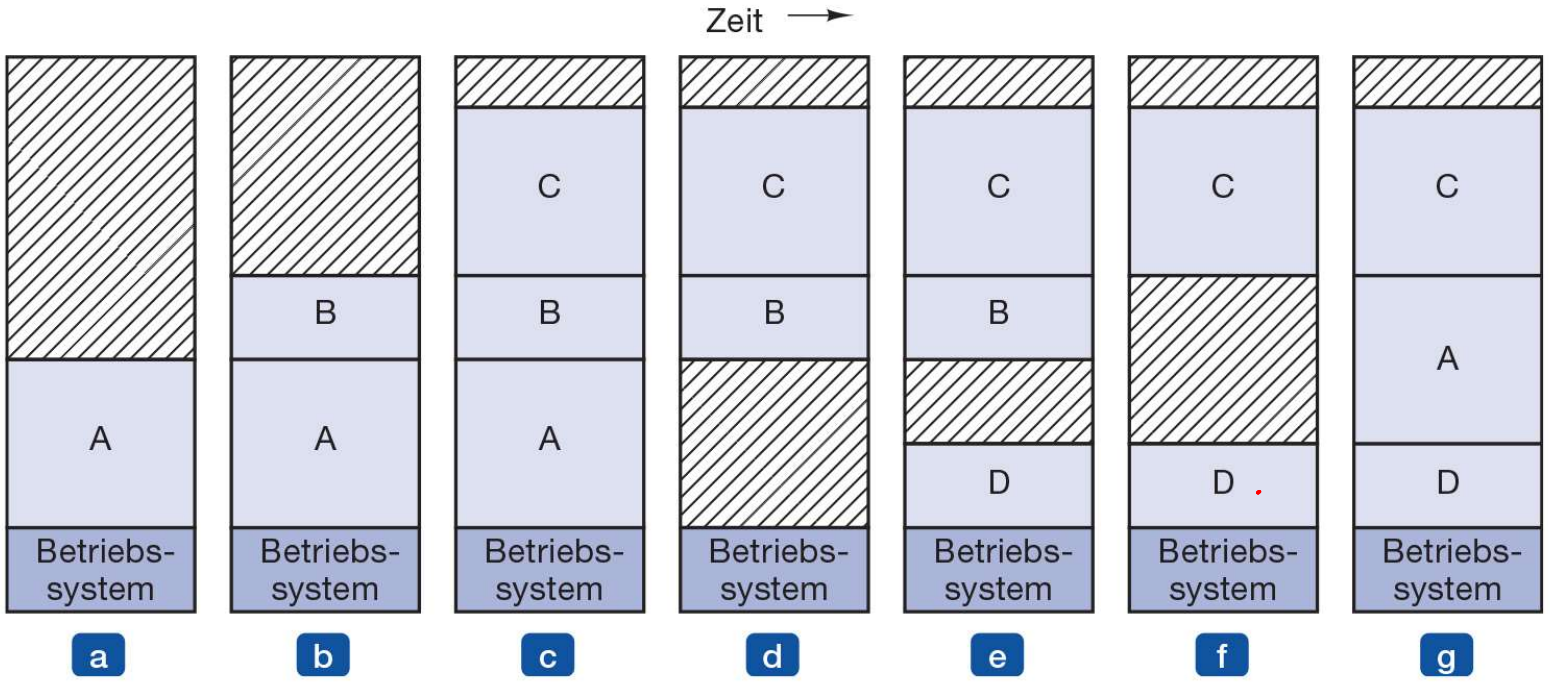
Basis- und Limitregister

- Basisregister beinhaltet Beginn des Speicherbereichs.
  - Basisregister + logische Adresse = physikalische Adresse
- Limitregister enthält Länge des Programms
  - Fehler wenn log. Adresse gleich oder größer als Limitregister
- Bei jedem Speicherzugriff -> Addition und Vergleich



Swapping

- Komplette Prozesse werden bei Leerlauf auf die Festplatte ausgelagert und erst bei Bedarf in den Hauptspeicher geladen.
- Probleme
  - Fragmentierung des Hauptspeichers
  - Speicherbedarf der Prozesse nicht konstant , kann wachsen und wieder schrumpfen
  - Prozess muss komplett in den Speicher passen
  - Performanz



Speicherreservierung für wachsende Prozesse

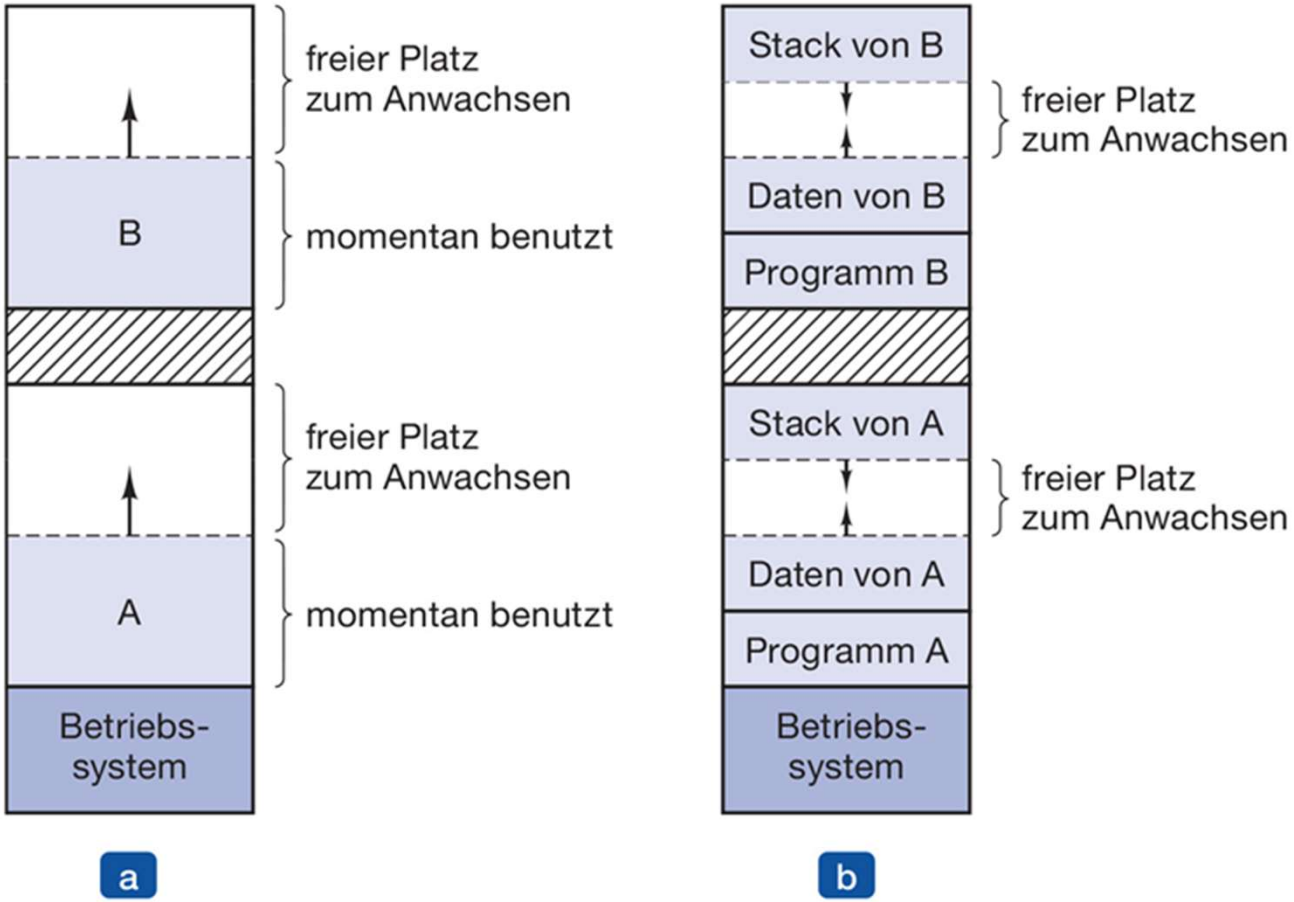
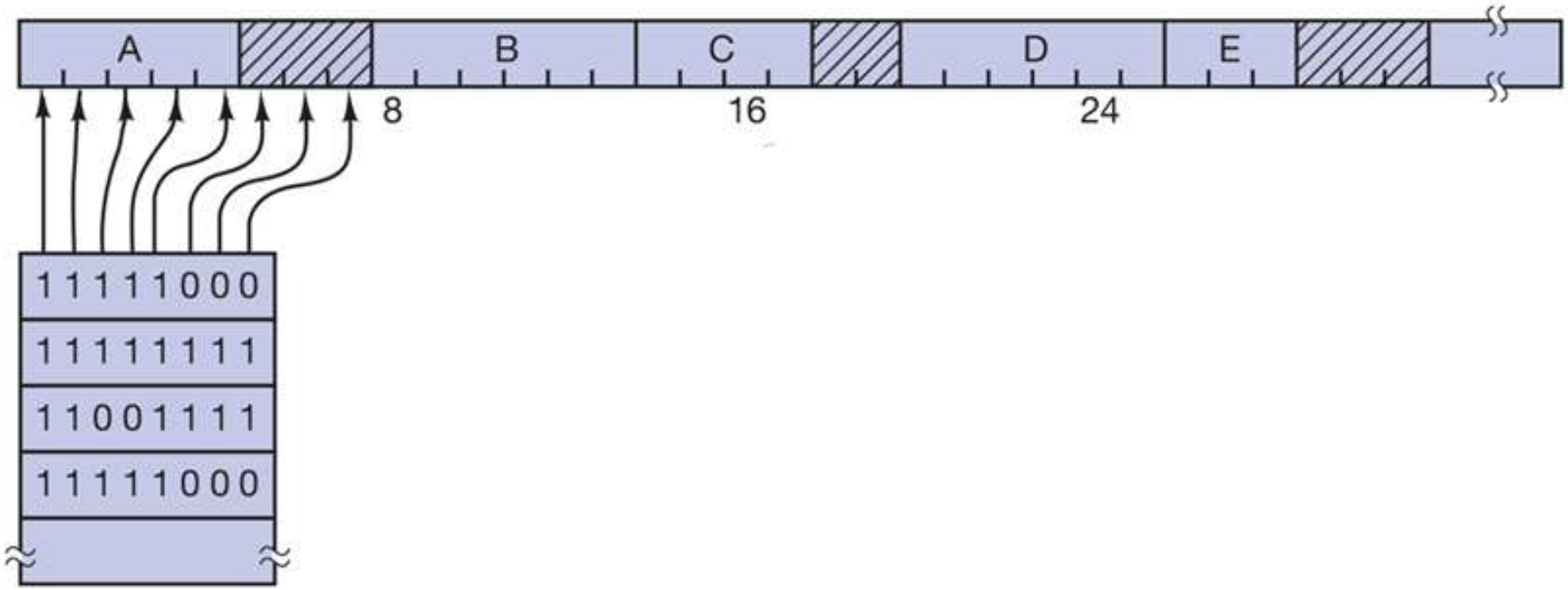


Abbildung 3.5: (a) Speicherreservierung für ein wachsendes Datensegment. (b) Speicherreservierung für wachsende Stack- und Datensegmente.

- Wenn Bereich nicht ausreicht:
  - ⇒ Verschieben
  - ⇒ Warten
  - ⇒ Verdichten
  - ⇒ abbrechen

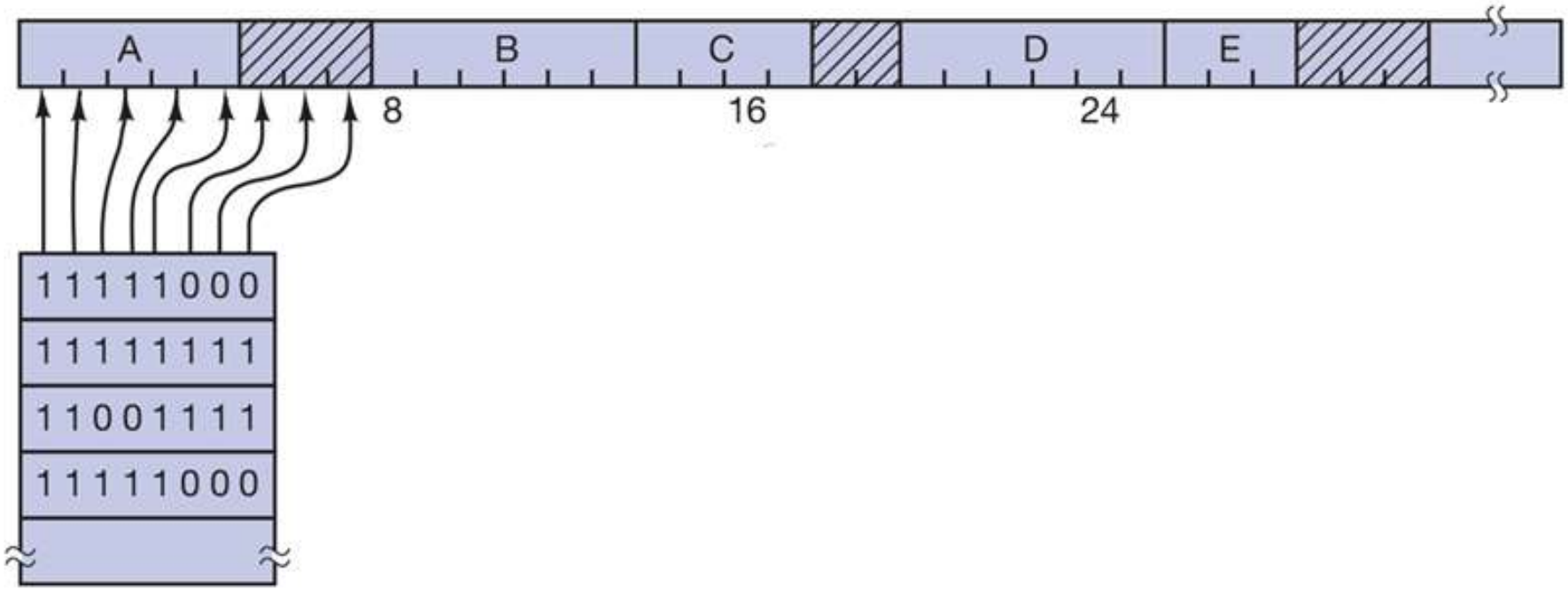
Verwaltung von freiem Speicher

- Einteilung des Speichers in Belegungseinheiten (Blöcke)
- Verwaltung mit Bitmaps:
  - Jedes Bit einer **Bitmap** repräsentiert eine Belegungseinheit
  - Der Zustand des Bits gibt an, ob belegt oder nicht



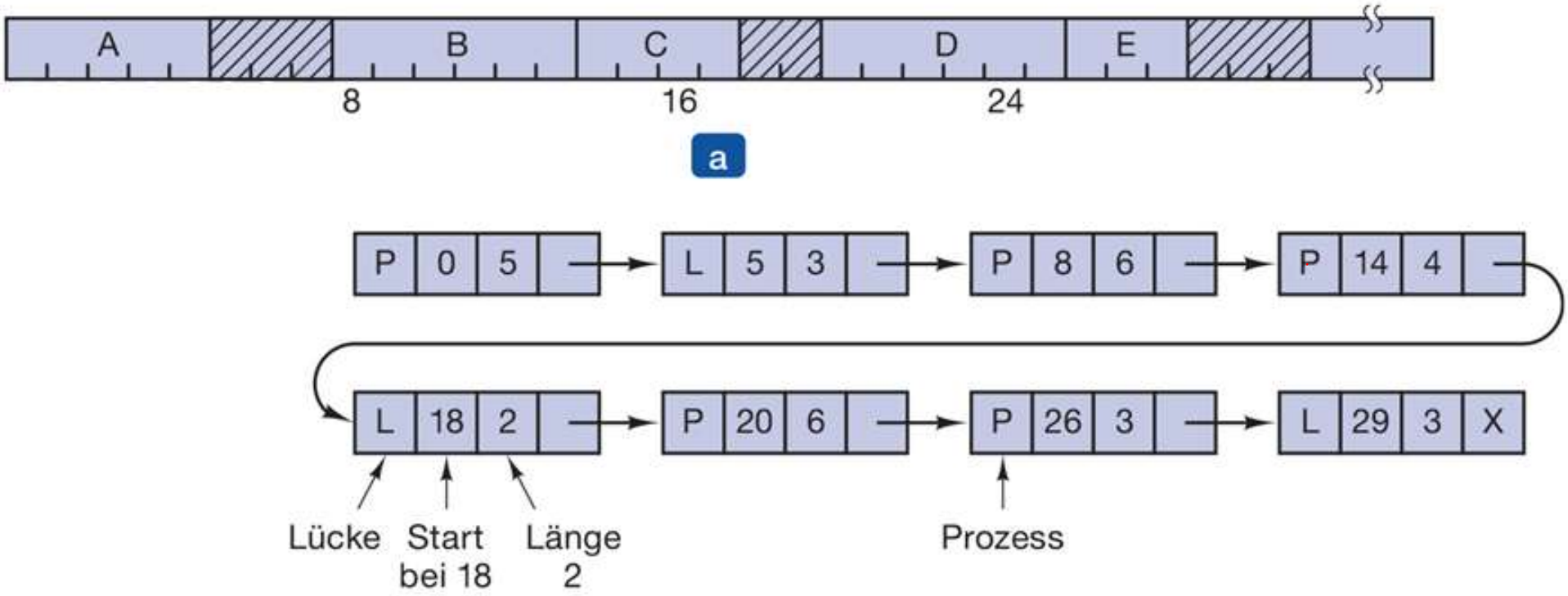
Verwaltung von freiem Speicher
Bitmaps

- Verwaltung mit Bitmaps
  - o wichtige Entwurfsfrage: Blockgröße
    große Verwaltungsstruktur ⇔ großer Verschnitt
  - o 4 Byte pro Block, 1 Bit für Belegungsinformation
    ⇒ 1/32 des Speichers für Bitmap benutzt
  - o Suchen nach freien Blöcken in der gesamten Bitmap



Verwaltung von freiem Speicher verkettete Listen

- Speicherverwaltung mit verketteten Listen
  - enthält Informationen über
    - freie Blöcke (L, Lücke)
    - belegte Blöcke (P, Prozess)
  - Größe abhängig von der Anzahl der Prozesse



Verwaltung von freiem Speicher verkettete Listen

- Terminierung von Prozessen erfordert Anpassen der Listenstruktur
- doppelt verkettete Liste bietet Vorteile beim Verschmelzen von Lücken

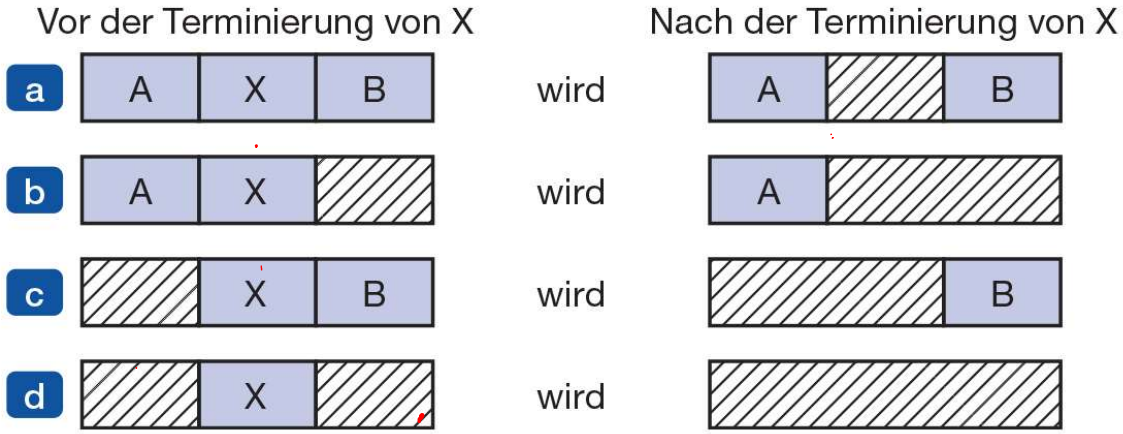
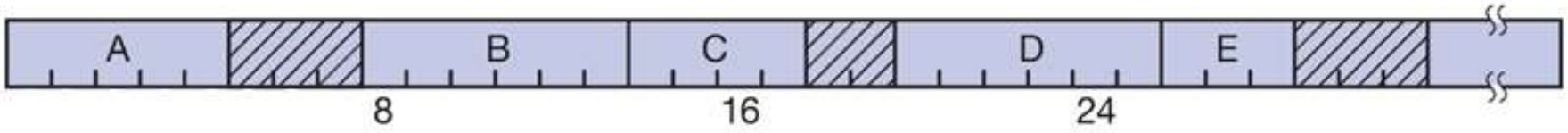


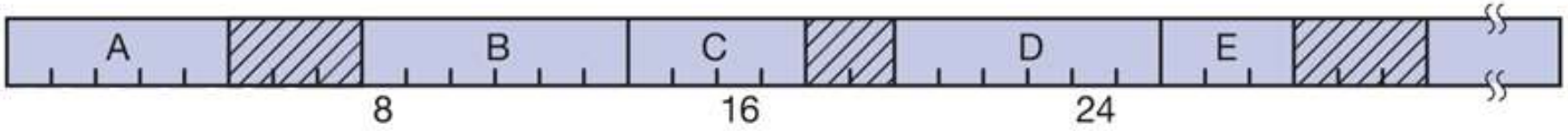
Abbildung 3.7: Die vier möglichen Kombinationen für die Nachbarn des terminierenden Prozesses *X*.

Verwaltung von freiem Speicher verkettete Listen

- Verkettete Liste im freien Speicher



- Verschmelzung von Lücken



Verwaltung von freiem Speicher

- Strategien für die Reservierung eines neuen Speicherbereichs
  - **First Fit** – erste passende Lücke wird verwendet
  - **Next Fit** – nächste passende Lücke (ausgehend von der letzten Lücke)
  - **Best Fit** – Lücke mit geringsten Verschnitt wählen
  - **Worst Fit** – Lücke mit brauchbarem Verschnitt wählen, indem immer die größte Lücke genutzt wird.
- Bewertung
  - **First Fit** – schnell, erzeugt eher große Lücken
  - **Next Fit** – in Simulationen leicht geringere Leistung als First Fit
  - **Best Fit** – ist langsam, verschwendet mehr Speicher (kleiner Rest)
  - **Worst Fit** – Simulationen zeigen schlechte Leistung

Verwaltung von freiem Speicher Optimierungen

- Lücken und Prozesse in getrennten Listen verwalten
  - gezielte Suche nach Lücken
  - Abarbeiten von zwei Listen ist aufwendiger
- Lücken nach Größe sortieren
  - optimale Suche bei Best Fit,
  - aber Verschmelzung von Einheiten schwierig
  - First Fit, Best Fit sind identisch, Next Fit überflüssig
- Quick Fit: Lücken je nach Größe in verschiedenen Listen verwalten
  - schnelles Finden von Lücken
  - Nach Terminierung eines Prozesses ist das Verwalten der Listen aufwendig

Aufgabe

In einer Speicherverwaltung befinden sich zwischen zwei Prozessen jeweils Lücken der folgenden Größe (nach aufsteigender Adresse geordnet):

10 KB, 4 KB, 20 KB, 18 KB, 7 KB, 9 KB, 12 KB, 15 KB.

Welche Lücken füllen **First Fit**, **Best Fit**, **Worst Fit** und **Next Fit** jeweils aus, wenn folgende Prozesse eingelagert werden sollen?

a) 12 KB, 10 KB und 9 KB

b) 12 KB, 10 KB und 16 KB

c) 12 KB, 10 KB und 8 KB

Lösung

10 KB, 4 KB, 20 KB, 18 KB, 7 KB, 9 KB, 12 KB, 15 KB

- **First Fit**
- **Best Fit**
- **Worst Fit**
- **Next Fit**

a) 12 KB, 10 KB und 9 KB

b) 12 KB, 10 KB und 16 KB

c) 12 KB, 10 KB und 8 KB

Virtueller Speicher

Basis- und Limitregister

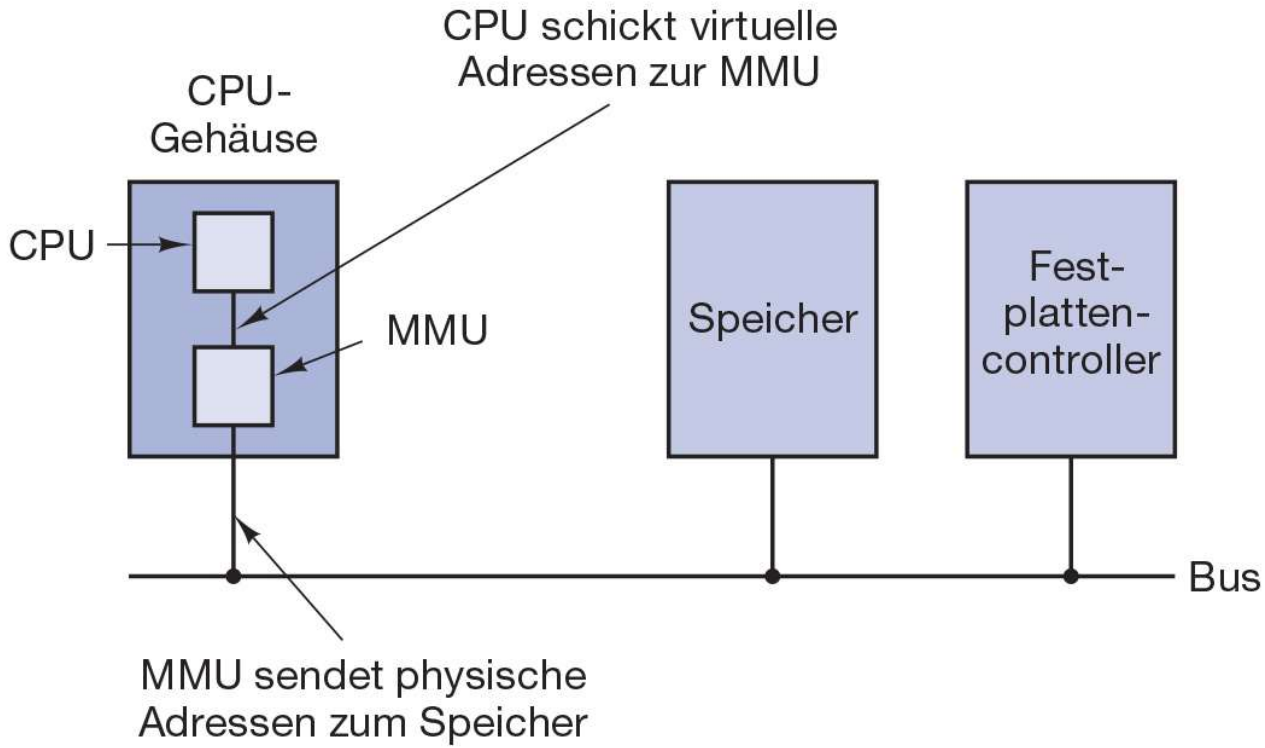
- Erlauben Speicherabstraktion
  - eigene Adressräume für jeden Prozess
  - Schutz des Speichers vor anderen Prozessen
- Erlauben Swapping
  - Ein-/Auslagern ganzer Prozesse
- bestehendes Problem
  - speicherhungrige Programme
  - von denen nur **nicht benutzte Teile** ausgelagert werden sollen

Virtueller Speicher

- Problemstellung: große Prozesse
  - passen nicht mehr in den Speicher
  - sind nicht mehr gut per Swapping ein-/auszulagern
    - Datenübertragungsrate von Festplatten zu niedrig (1GB ca. 10s)
- Lösungsansatz: virtueller Speicher und Paging
  - Adressraum wird in Seiten aufgeteilt
  - es müssen sich nicht alle Seiten eines Prozesses im Speicher befinden
  - fehlende Seiten werden nachgeladen, gerade nicht benötigte Seiten dafür ausgelagert.
  - jeder Prozess erhält eigenen virtuellen Speicher

Virtueller Speicher Paging

- Die von einem Programm generierten Adressen werden als virtuell angesehen und bilden den virtuellen Adressraum
- Die virtuellen Adressen werden von einer sogenannten Memory Management Unit (MMU) auf physische Adressen abgebildet



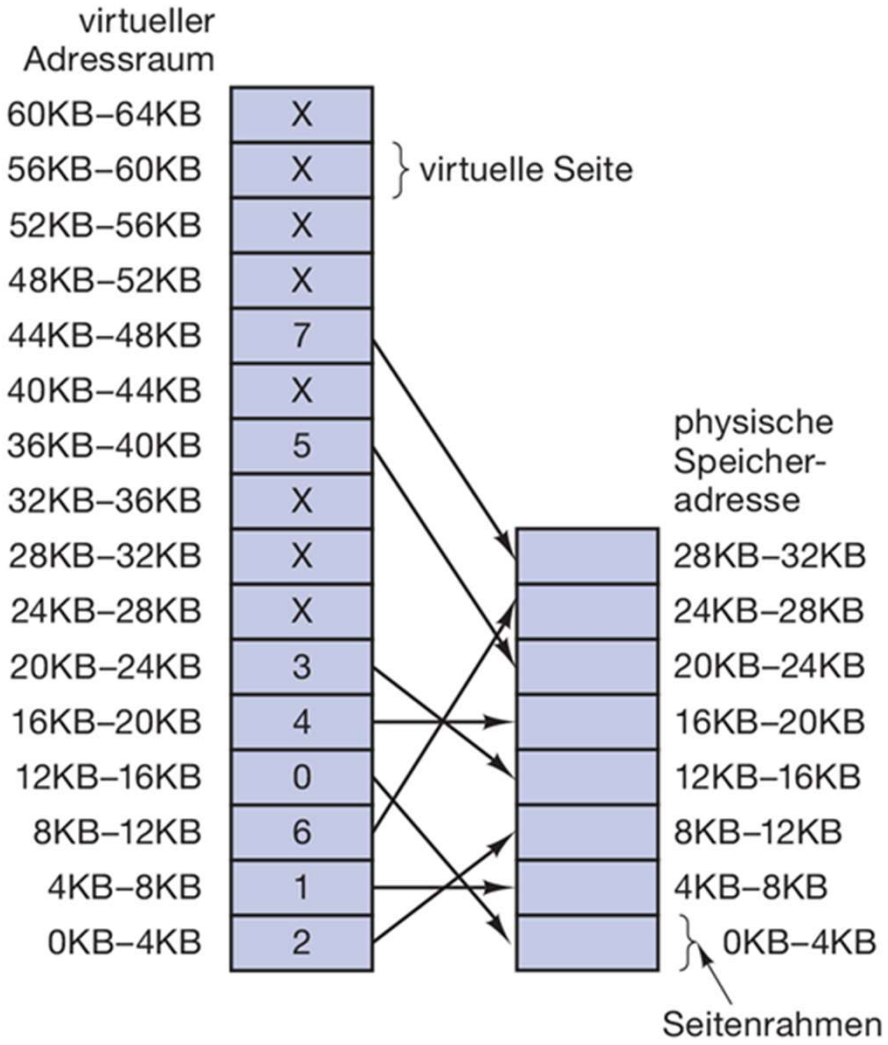
Virtueller Speicher
Paging

Begriffsdefinitionen

- **Seiten**
  Adressraum ist in Einheiten gleicher Größe (sog. Seiten) aufgeteilt
- **Seitenrahmen**
  physischer Speicher ist in Einheiten gleicher Größe (sog. Seitenrahmen oder Seitenkacheln) aufgeteilt
- **Seitentabellen**
  verwalten Zuordnung zwischen Seiten und Seitenrahmen
- **Segmente**
  ähnlich wie Seiten, nur ungleich groß
- **Segmenttabellen**
  ähnlich wie Seitentabellen, mit Größeninformation

Virtueller Speicher Paging

- MOV REG, 0
  - 0 wird an MMU geschickt
  - MMU stellt fest, dieser Bereich ist im Rahmen 2
  - MMU wandelt 0 in 8192 um
  - Speicher liefert Wert an Adr. 8192
- MOV REG, 32768
  - 32768 wird an MMU geschickt
  - MMU stellt fest, dieser Bereich ist ausgelagert (X)
  - Seitenfehler (page fault) wird erzeugt
  - Betriebssystem muss reagieren ...

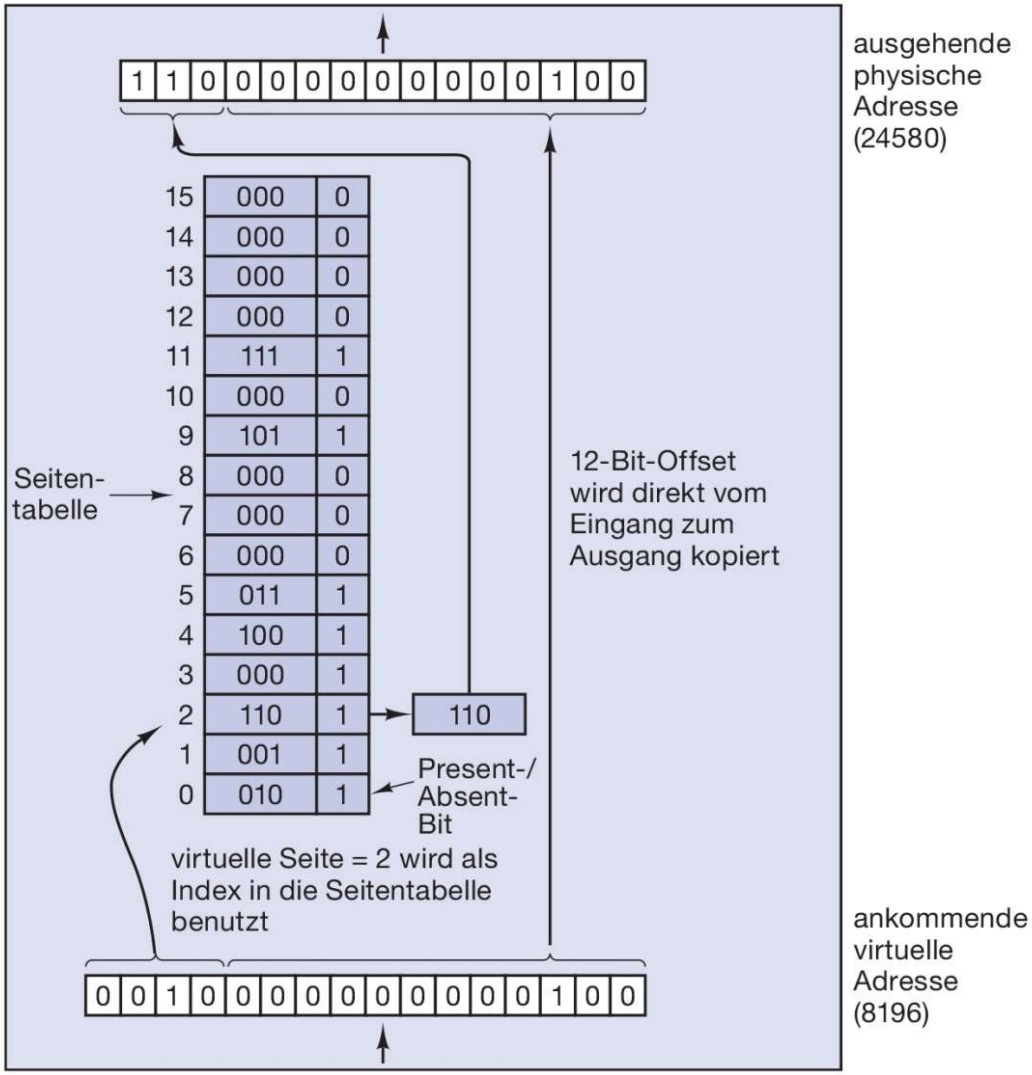


Virtueller Speicher
Memory Management Unit

Speicherverwaltungseinheit

Beispiel:

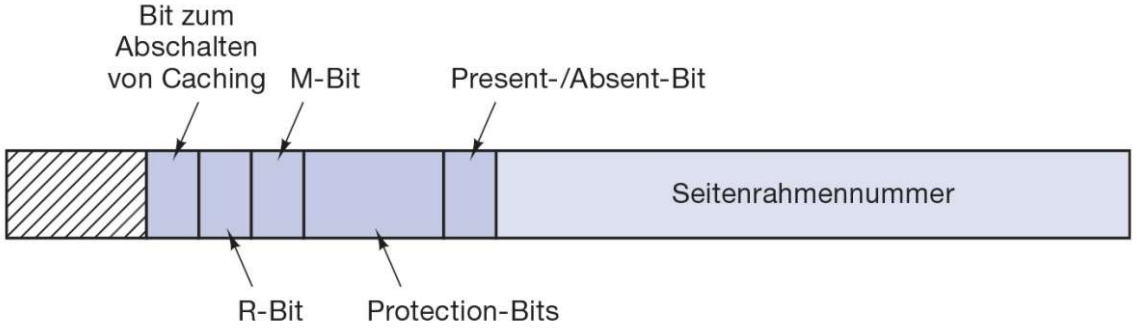
- verwaltet Seiten á 4KB
- virtueller Adressraum
  - 16 Seiten
- physischen Speicher
  - 8 Seitenrahmen
- Present-Bit/Absent-Bit



- reale Systeme haben Seitengrößen von 512Byte bis zu 1GB
- Viele CPUs unterstützen mehrere Seitengrößen z.B. x86-64: 4KB, 2MB, 1GB

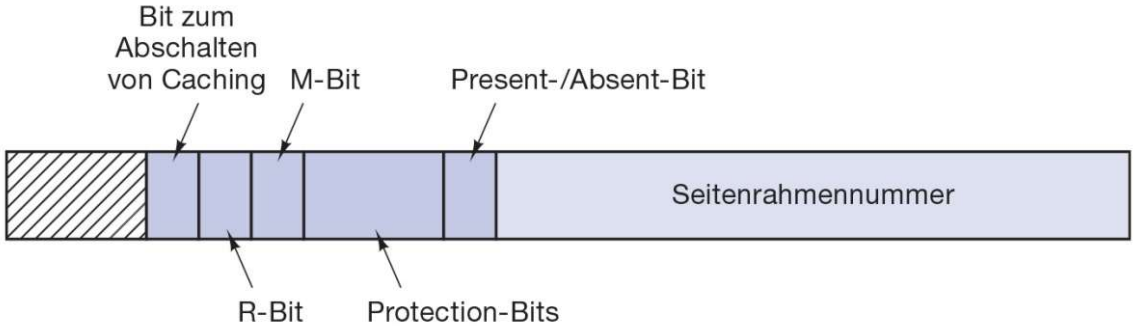
Seitentabelleneintrag

- Typisch 32Bit
- Seitennummer als Index
- Seitenrahmennummer falls vorhanden
- Present/Absent-Bit
  - 1 = Seite liegt im Speicher
  - 0 = Seite liegt nicht im Speicher -> Seitenfehler
- Caching-Bit
  - gibt an, ob Speicherzugriffe gecached werden dürfen
  - z. B. nicht geschehen, wenn hinter der Adresse in Wirklichkeit Ein-/Ausgabeeinheiten stehen



Seitentabelleneintrag

- Protection-Bits oder Schutzbits
  - regeln den Zugriff auf die Seite
    - 1 Bit: 0: Lese-/Schreibzugriff 1: Leserechte
    - 3 Bit: Je 1 Bit für Schreib-/Lese-/Ausführrecht
  - M-Bit (modified, dirty bit)
    - wird von der MMU gesetzt, wenn die Seite modifiziert wurde
    - wird vom BS zurückgesetzt, wenn Seite auf die Festplatte geschrieben wurde.
  - R-Bit (referenced)
    - wird bei jedem Zugriff (Lesen, Schreiben) von der MMU gesetzt
    - Wird vom BS gelesen und nach bestimmten Strategien zurückgesetzt



Beschleunigung des Paging

- Translation Lookaside Buffer (TLB)
  - Hardware-Cache für wenige häufig benutzte Seiten (selten über 256 Einträge)
  - z.B. Programm 19-21, Daten 129/130

Gültig	Virtuelle Seite	Verändert	Schutz	Seitenrahmen
1	140	1	RW	31
1	20	0	R X	38
1	130	1	RW	29
1	129	1	RW	62
1	19	0	R X	50
1	21	0	R X	45
1	860	1	RW	14
1	861	1	RW	75

Aufgabe

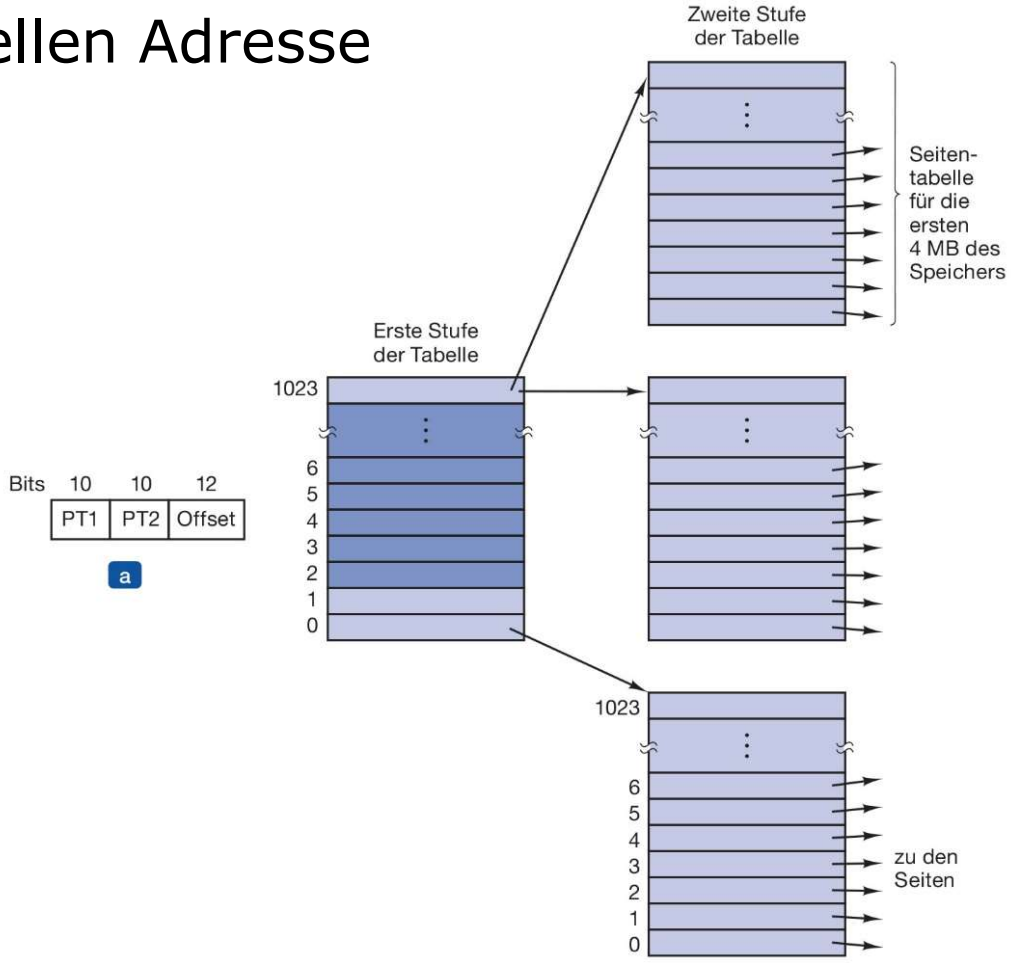
- Skizzieren Sie die Seitentabelle einer MMU, die
  - virtuelle 32 Bit-Adressen und
  - physikalische 24 Bit-Adressen verwendet und mit
  - einer Blockgröße von 16 KB arbeitet.
- Wie viele Einträge benötigt eine einstufige Seitentabelle?

Paging
Probleme mit großem Speicher

- virtueller Adressraum >> physikalischer Adressraum
  - Riesige Seitentabellen!
  - Sei Seitengröße = offset = 12 Bit, d.h. 4KB Seitenrahmen (page frame)
    - 16 Bit Adressraum:
    - 32 Bit Adressraum:
    - 64 Bit Adressraum:
- Adressbegrenzung
  - kleiner Adressraum, z.B. 30 Bit (1 GB) $\Rightarrow$ 18 Bit $\Rightarrow$ 256 KB Tabelle
  - jeder Prozess hat eigene Tabelle
  - Vergeudung von Platz, da meist nicht benötigt !

Mehrstufige Seitentabellen

- Ziel: Reduktion der Anzahl der Zeilen in der Seitentabelle
- mehrere Indizes in einer virtuellen Adresse



Aufgabe

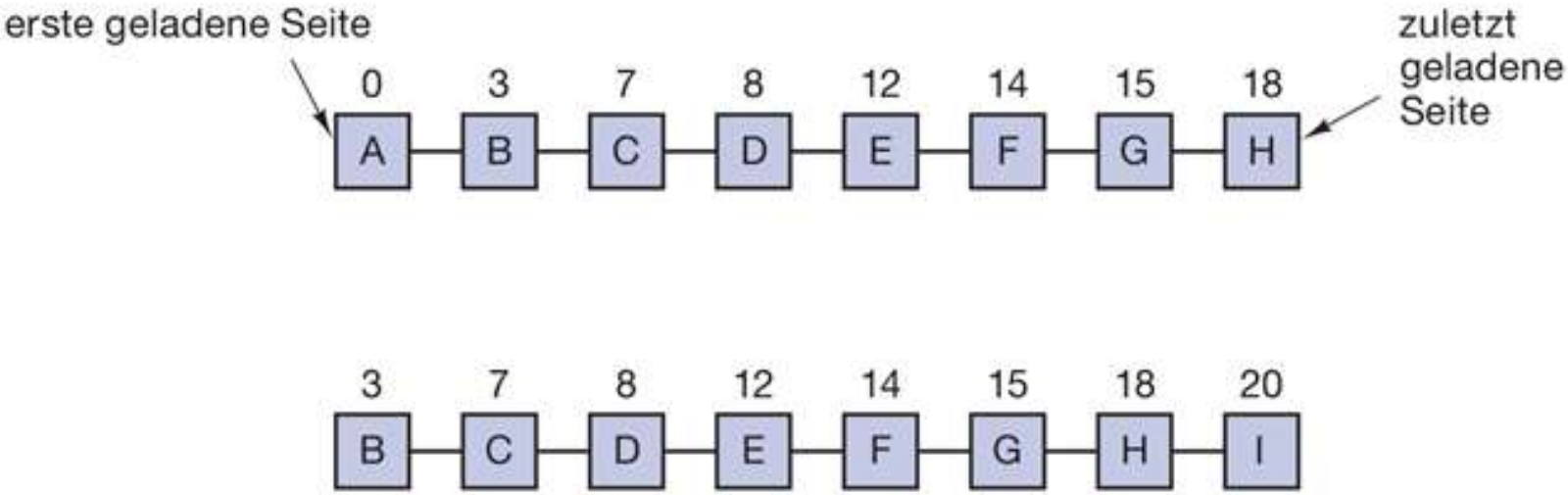
- Skizzieren Sie die Seitentabelle einer MMU, die
  - virtuelle 32 Bit-Adressen und
  - physikalische 24 Bit-Adressen verwendet und mit
  - einer Blockgröße von 16 KB arbeitet.
- Wie viele Einträge benötigt eine einstufige Seitentabelle?
- Wie sieht ein Seitentabelleneintrag für eine zweistufige Seitentabelle aus, d.h. wie viele Bits sollten für die erste und die zweite Ebene jeweils verwendet werden?

Seitenersetzungsalgorithmen

- Ein Seitenersetzungsalgorithmus wählt einen Seitenrahmen im physikalischen Speicher aus, dessen Inhalt durch eine zu ladende Seite ersetzt werden kann.
- Falls die Seite modifiziert wurde, muss sie zuvor geschrieben werden, damit kein Datenverlust auftritt.
- Der optimale Algorithmus wählt die Seite, deren Aufruf am weitesten in der Zukunft liegt.
- Leider ist dieser Zeitpunkt praktisch nicht bestimmbar, daher müssen wir uns mit Annäherungen behelfen.

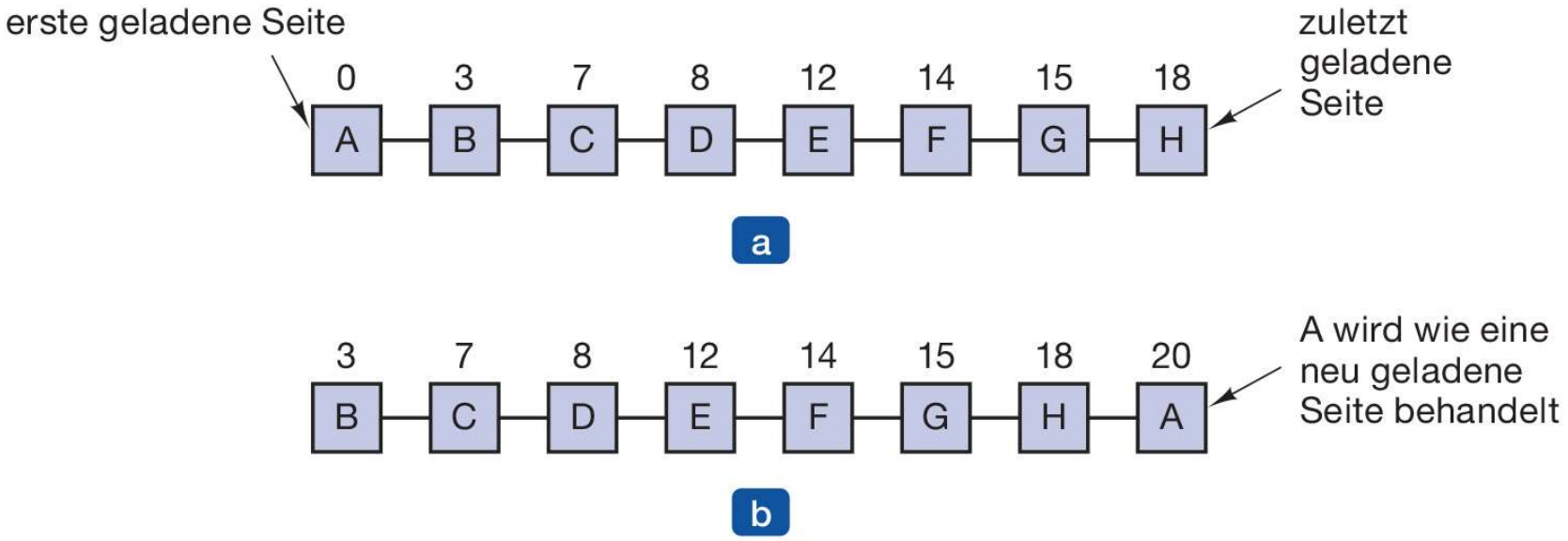
First-In-First-Out-Algorithmus (FIFO)

- Die Seiten werden in einer verketteten Liste gespeichert.
- Am Ende der Liste wird immer die zuletzt geladene Seiten angehängt.
- Muss eine Seite ausgelagert werden, so wird immer die älteste Seite gewählt (am Beginn der Liste)



Second-Chance-Algorithmus

- Die geladenen Seiten werden mit einem Zeitstempel des Einlagerungszeitpunkts in einer FIFO-Datenstruktur gespeichert (die Liste ist damit nach diesen Zeitpunkten sortiert)
- Prüfung des R-Bits
  - 0: ist die Seite alt, unbenutzt und wird entfernt
  - 1: R-Bit wird auf 0 gesetzt, der Zeitstempel wird aktualisiert und die Seite an das Ende verschoben. Die Suche wird fortgesetzt.
- Was passiert wenn alle R=1?

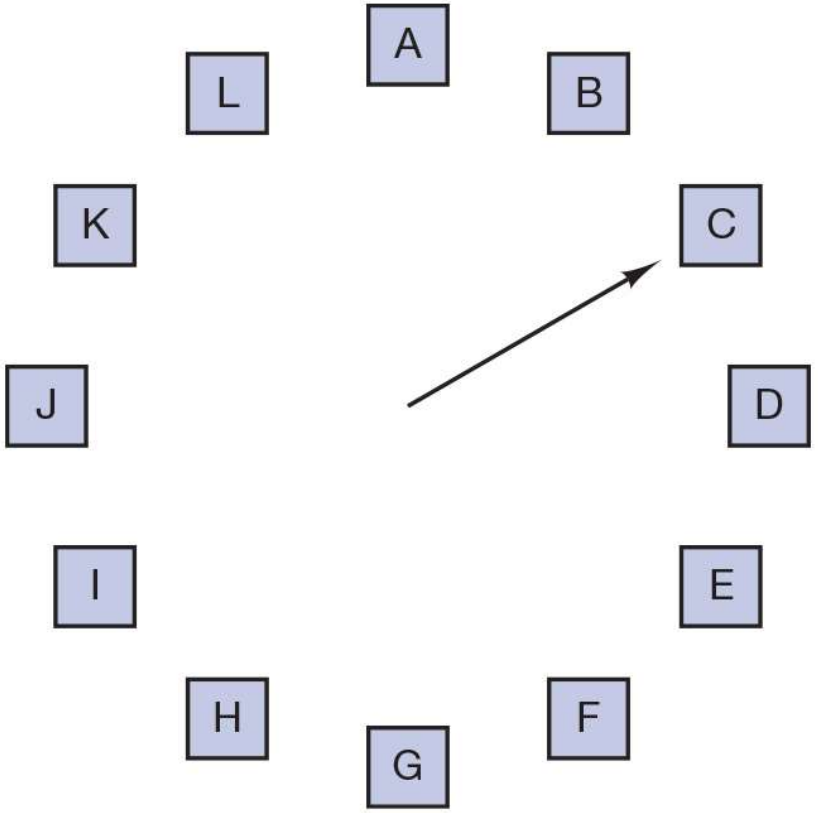


Not-Recently-Used-Algorithmus (NRU)

- Wenn ein Prozess gestartet wird, dann setzt das BS die R- (referenced) und M-Bits (modified) für alle Seiten auf 0.
- In bestimmten (periodischen) Zeitabständen (~20ms) setzt das BS das R-Bit aller Seiten zurück -> bei nicht mehr verwendeten Seiten steht das R-Bit damit auf 0.
- Bei einem Seitenfehler werden alle Seiten entsprechen R- und M Bit in vier Kategorien aufgeteilt:
  - Klasse 0: nicht referenziert, nicht modifiziert
  - Klasse 1: nicht referenziert, modifiziert
  - Klasse 2: referenziert, nicht modifiziert
  - Klasse 3: referenziert, modifiziert
- Der NRU-Algorithmus wählt eine zufällige Seite aus der niedrigsten, nicht leeren Klasse

Clock-Algorithmus

- Weiterentwicklung des Second-Chance-Algorithmus
- Verwendet einen Ringpuffer anstatt einer FIFO-Struktur, bei der ständig Seiten in Listen verschoben werden müssen
- Der Zeiger zeigt auf die jeweils älteste Seite



Wenn ein Seitenfehler auftritt, wird die Seite untersucht, auf die der Zeiger weist. Die Folgeaktion hängt dann vom R-Bit ab:

R = 0: verdränge die Seite
R = 1: lösche R und rücke Zeiger weiter

Least-Recently-Used-Algorithmus (LRU)

- Beobachtung
  - eine Seite, die während der letzten Befehle benötigt wurde, wird wahrscheinlich auch für die nächsten Befehle benötigt
  - Eine seit längerem unbenutzte Seite wird weiterhin unbenutzt bleiben
  - Lokalitätsprinzip
- LRU-Algorithmus wie Second-Chance, nur werden die geladenen Seiten mit dem
  - Zeitstempel des **Zugriffszeitpunkts** in einer FIFO-Datenstruktur gespeichert (Second Chance verwendet **Einlagerungszeitpunkt**)
  - Die Seite, die am längsten unbenutzt ist, wird zur Auslagerung ausgewählt
  - Verfahren ist technisch nur aufwändig zu realisieren

Aufgabe

- Ein Computer hat vier Seitenrahmen. Die Tabelle zeigt für jede Seite die Ladezeit, die Zeit des letzten Zugriffs sowie die R- und M-Bits.

Seite	geladen	letzter Zugriff	R	M
0	126	280	1	0
1	230	265	0	1
2	140	270	0	0
3	110	285	1	1

  - Welche Seite ersetzt FIFO?
  - Welche Seite ersetzt NRU?
  - Welche Seite ersetzt LRU?
  - Welche Seite ersetzt Second Chance

Aufgabe

Ein Computer mit 4 Seitenrahmen und der FIFO-Strategie benötigt die folgenden Seiten:

0 1 2 3 3 4 0 1 5 6 0 1 2 3 5 6.

a) Wie viele Seitenfehler treten auf?

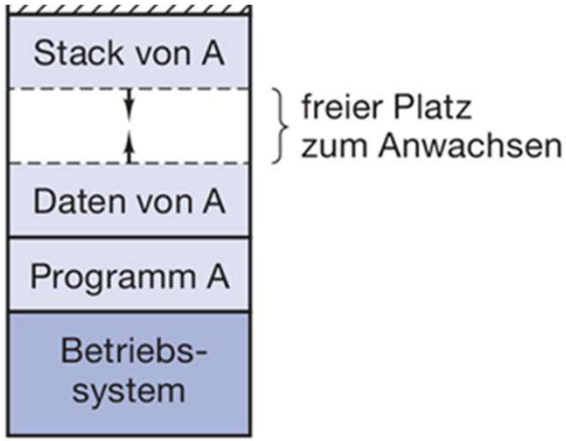
b) Der Computer wird um einen weiteren Seitenrahmen erweitert. Bestimmen Sie die Anzahl der Seitenfehler bei gleicher Strategie und Seitenfolge.

c) Erklären sie diese Anzahl.

Segmentierung

Segmente

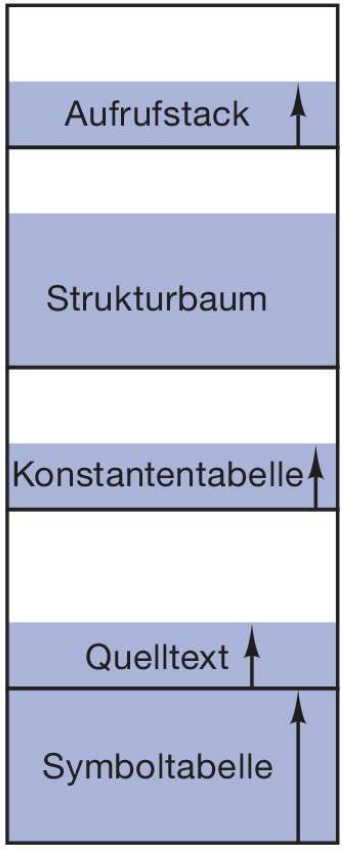
- Ein Prozess teilt seinen Adressraum in mehrere Segmente auf:
  - Das **Textsegment** enthält den Maschinencode (Programm) und konstante Variablen (das Text-Segment kann nicht verändert werden)
  - Das **Datensegment** die globalen Variablen und den Heap (dynamischer Speicherbereich, aus dem von einem Programm zur Laufzeit Speicher angefordert werden kann),
  - das **Stacksegment** die lokalen Variablen.
- **1-dimensionaler** Adressraum, da alle Segmente im gleichen Adressraum liegen.

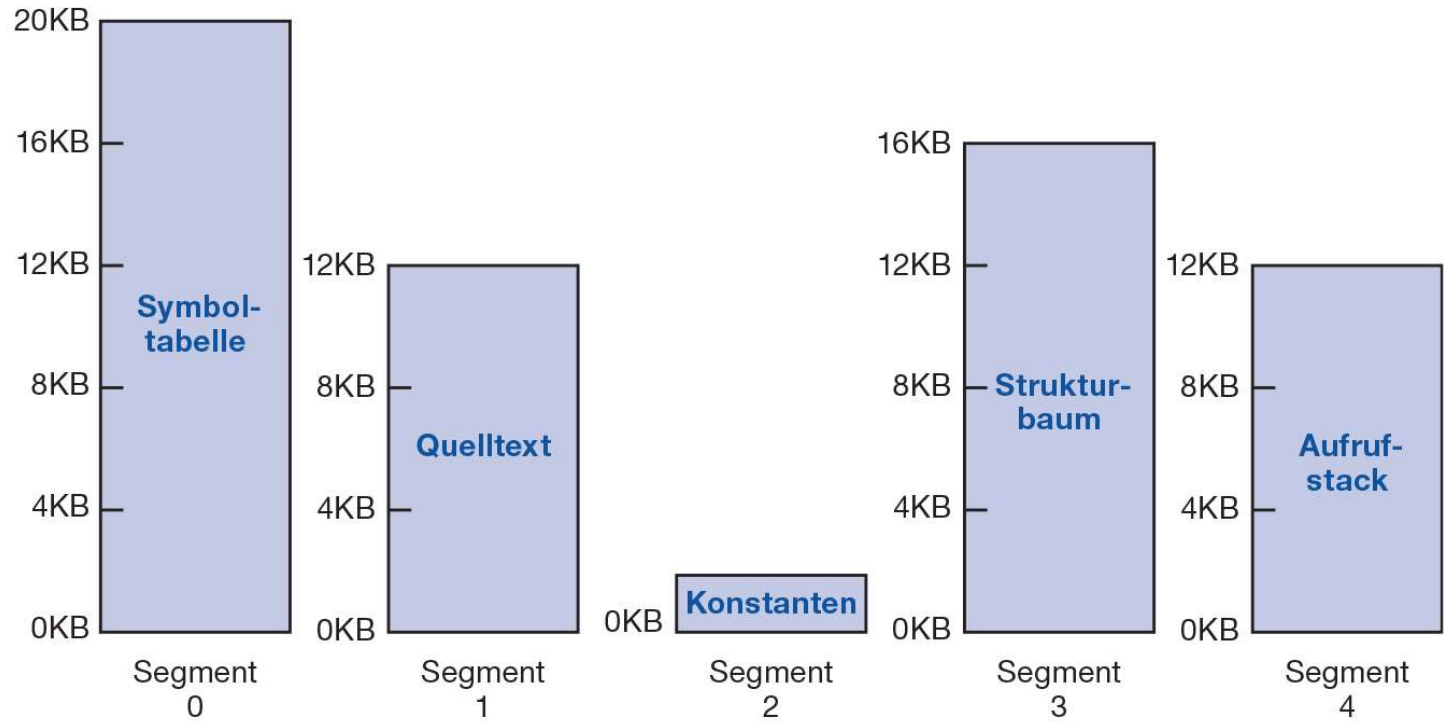


Segmente

- Moderne BS ordnen **jedem Segment** einen **separaten Adressbereich** zu
- ein Prozess hat damit einen **mehr-dimensionalen** Adressbereich
- Eigenschaften
  - Segmente können unabhängig voneinander ihre Größe ändern.
  - Zugriffsrechte können für jedes Segment separat vergeben werden
  - Eine dynamische Bibliothek kann über ein eigenes Segment realisiert werden, das von mehreren Prozessen genutzt wird.

1- vs. mehrdimensionaler Adressraum





Vergleich Paging – Segmentierung

Überlegung	Paging	Segmentierung
Muss der Programmierer wissen, dass diese Technik benutzt wird?	Nein	Ja
Wie viele lineare Adressräume gibt es?	1	Viele
Kann der gesamte Adressraum die Größe des physischen Speichers übersteigen?	Ja	Ja
Können Prozeduren und Daten unterschieden und getrennt voneinander geschützt werden?	Nein	Ja
Können Tabellen mit schwankender Größe verwaltet werden?	Nein	Ja
Wird das gemeinsame Benutzen von Prozeduren durch Anwender unterstützt?	Nein	Ja
Warum wurde diese Technik eingeführt?	Um einen großen linearen Adressraum benutzen zu können, ohne weiteren physischen Speicher zu kaufen	Um Programme und Daten in unabhängige logische Adressräume aufzuspalten und um gemeinsame Nutzung und Schutz zu unterstützen

Übungen

- Worin besteht der Unterschied zwischen einer physischen Adresse und einer virtuellen Adresse?
- Erläutern sie die folgenden Begriffe aus dem Bereich paging:
  - Seiten
  - Seitenrahmen
  - Seitentabellen
- Gibt es beim Paging mehr Seiten oder mehr Seitenrahmen?
- Eine Maschine hat virtuelle 48-Bit-Adressen und physische 32-Bit-Adressen. Wie viele 8-KB-Seiten sind in der einstufigen, linearen Seitentabelle?
- Ein Computer mit 32-Bit-Adressen benutzt eine zweistufige Seitentabelle. Virtuelle Adressen werden in ein 9-Bit-Feld für die oberste Seitentabelle, 11 Bit für die zweite Seitentabelle und einen Offset unterteilt. Wie viele Seiten sind im Adressraum und wie groß sind sie?

Übungen

- Betrachten Sie eine Maschine mit virtuellen 38-Bit-Adressen und physischen 32-Bit-Adressen.

  a.) Was ist der größte Vorteil einer mehrstufigen Seitentabelle gegenüber einer einstufigen?

  b.) Wie viele Bits sollten bei einer zweistufigen Seitentabelle mit 16-KB-Seiten und Einträgen der Länge 4 Byte für das obere Seiten-tabellenfeld reserviert werden?

  Und wie viele für das Feld der nächsten Ebene?

  Erläutern Sie Ihre Antwort.