

# 3 Speicherverwaltung

- 1 Einführung
- 2 Prozesse und Threads
- 3 Speicherverwaltung**
- 4 Dateisysteme
- 5 Eingabe und Ausgabe
- 6 Deadlocks
- 7 Virtualisierung und die Cloud
- 8 Multiprozessorsysteme
- 9 IT-Sicherheit
- 10 Fallstudie 1: Linux
- 11 Fallstudie 2: Windows
- 12 Entwurf von Betriebssystemen

# Speicher

- Umschreibung des Parkinsonschen Gesetzes:  
*„Programme so erweitern, dass sie den Speicher füllen, der verfügbar ist, um sie zu halten.“ \**
- Der durchschnittliche Heimcomputer hat heutzutage 10.000 Mal mehr Speicher als der IBM 7094, der größte Computer der Welt in den frühen 1960er Jahren.

\* Die Parkinsonschen Gesetze sind ironisierende Darstellungen des britischen Soziologen C. Northcote Parkinson zur Verwaltungs- und Wirtschaftslehre. Am bekanntesten ist das Parkinsonsche Gesetz zum Bürokratiewachstum: "Work expands so as to fill the time available for its completion."

# 3 Speicherverwaltung

3.1 Systeme ohne Speicherabstraktion

3.2 Speicherabstraktion: Adressräume

3.3 Virtueller Speicher

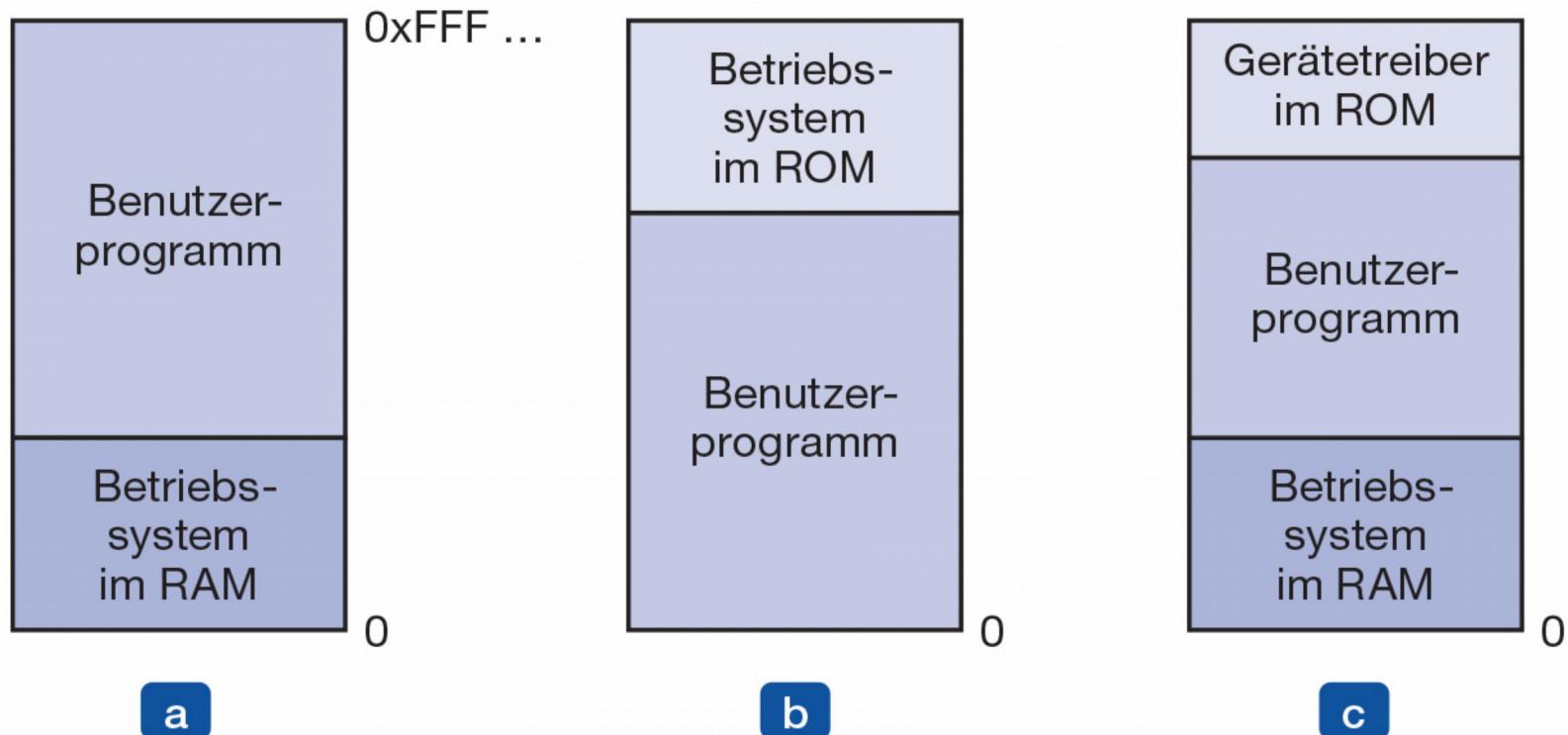
3.4 Seitenersetzungsalgorithmen

3.5 Entwurfskriterien für Paging-Systeme

3.6 Implementierungsaspekte

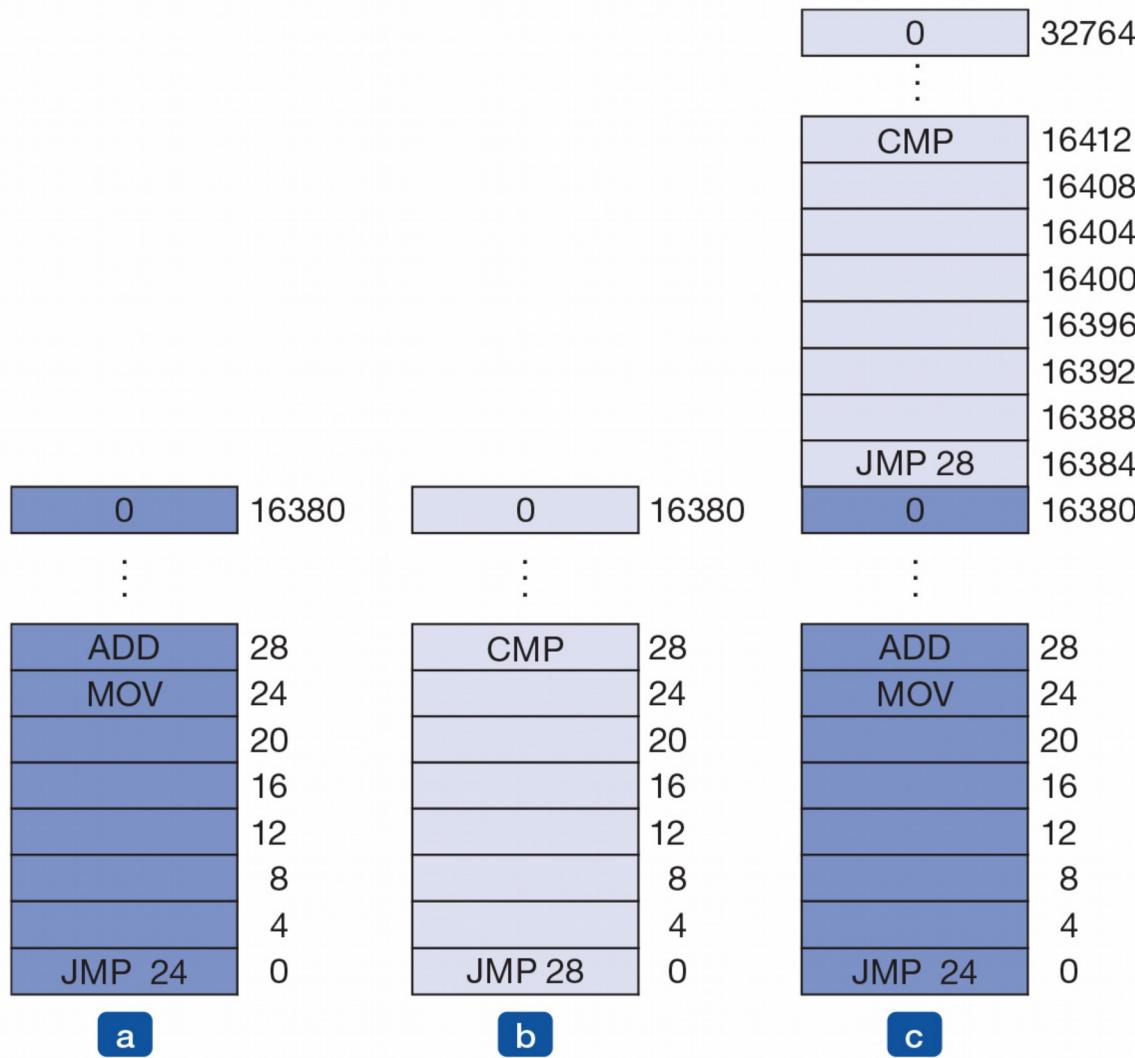
3.7 Segmentierung

# Systeme ohne Speicherabstraktion



**Abbildung 3.1:** Drei einfache Möglichkeiten, den Speicher für einen Benutzerprozess und das Betriebssystem zu organisieren. Es gibt aber noch andere Möglichkeiten.

# Ausführen mehrere Programme ohne Speicherabstraktion



**Abbildung 3.2:** Darstellung des Relocationsproblems: (a) Ein 16-KB-Programm; (b) ein weiteres 16-KB-Programm. (c) Die zwei Programme wurden hintereinander in den Speicher geladen.

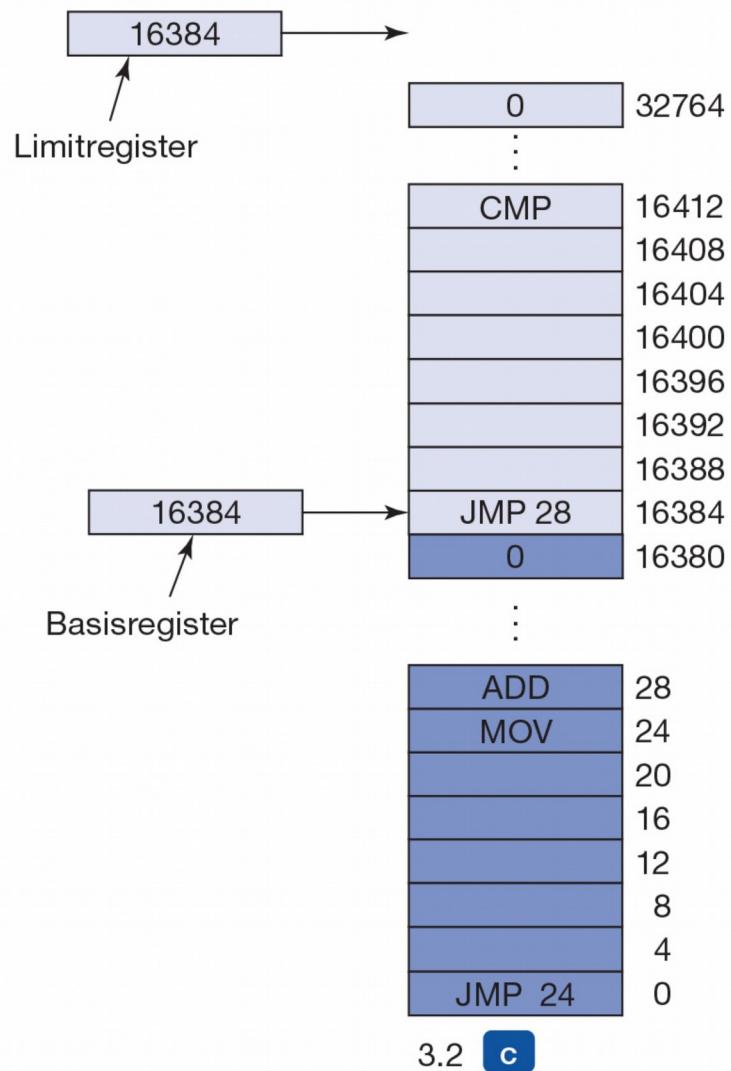
## 3.2 Speicherabstraktion: Adressräume

3.2.1 Das Konzept des Adressraumes

3.2.2 Swapping

3.2.3 Verwaltung von freiem Speicher

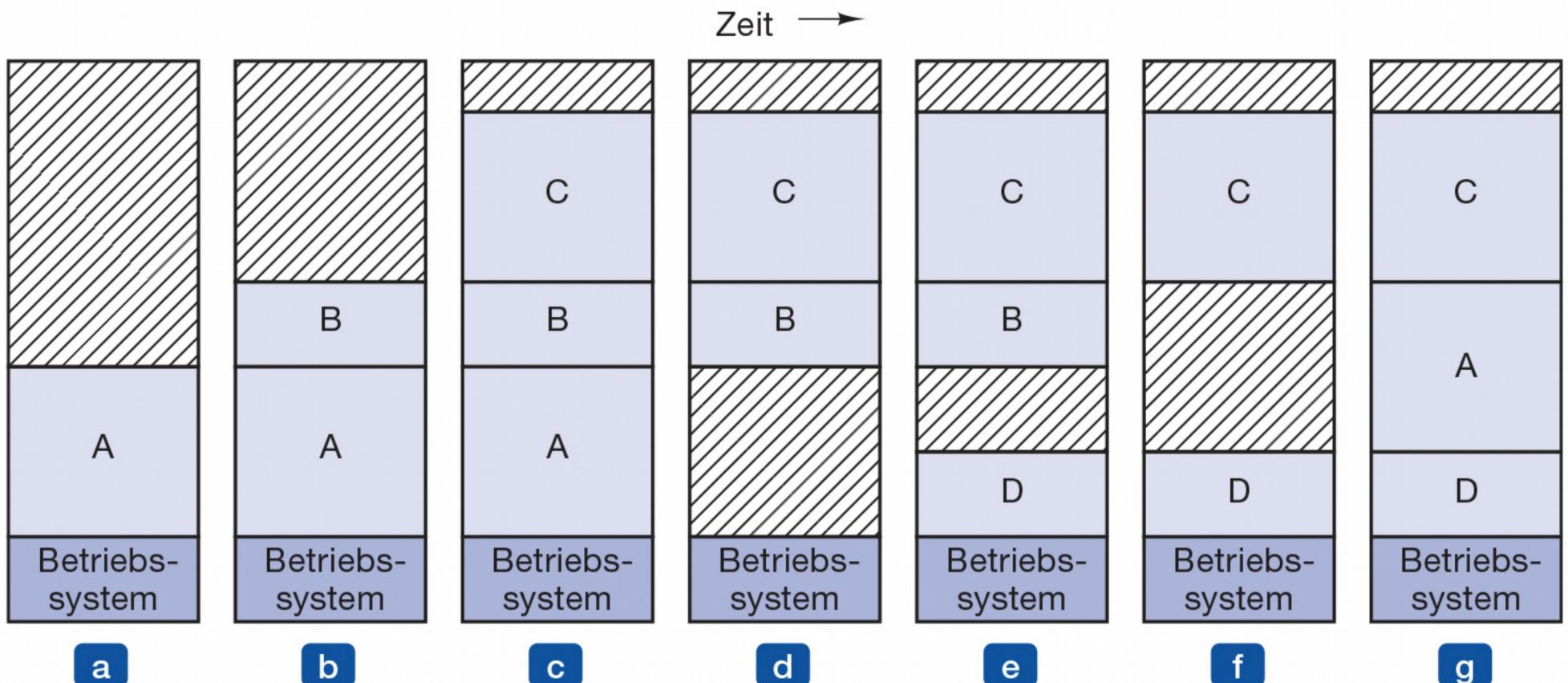
# Basis- und Limitregister



3.2 c

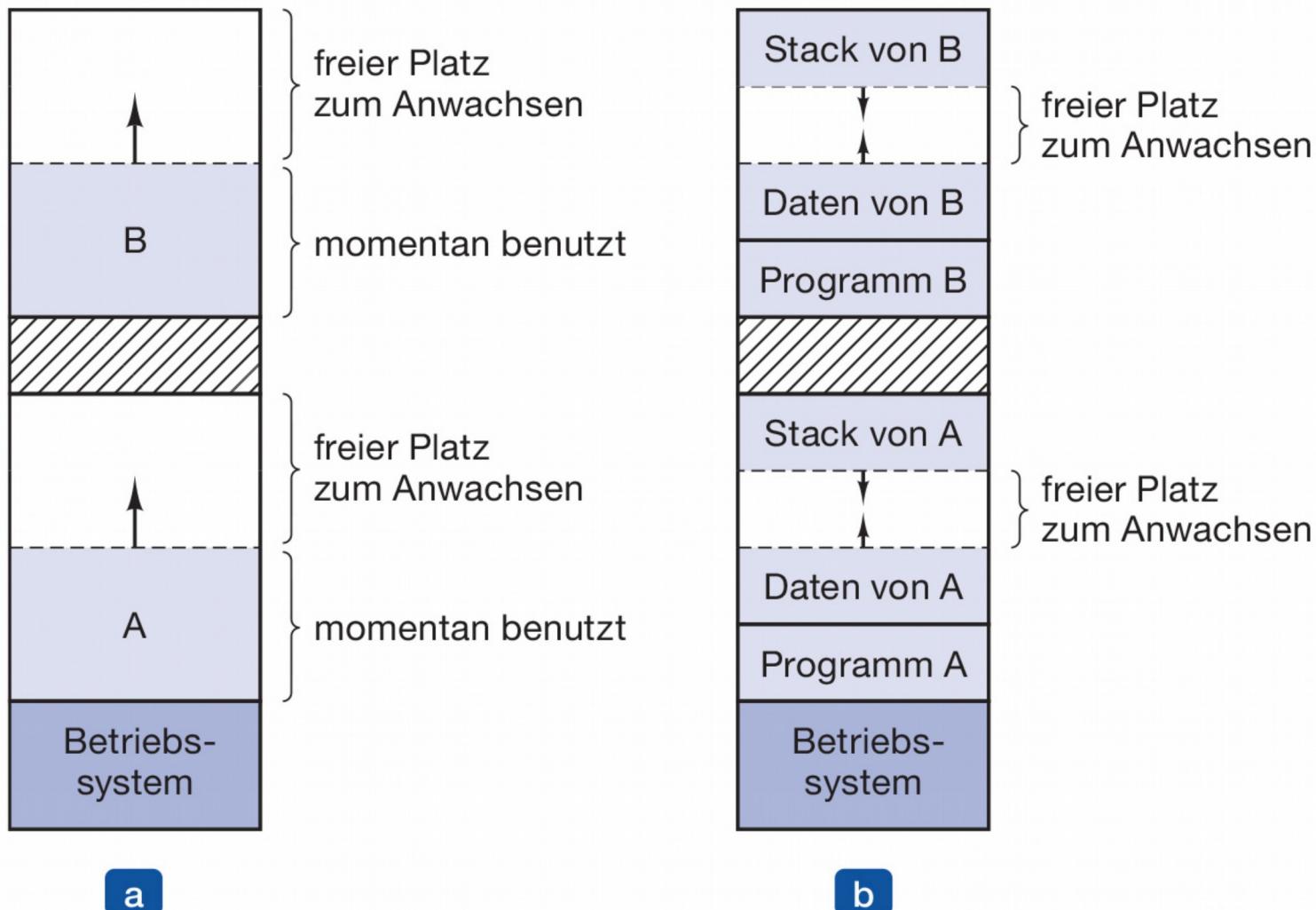
**Abbildung 3.3:** Basis- und Limitregister können benutzt werden, um jedem Prozess einen separaten Adressraum zu geben.

# Swapping (1)



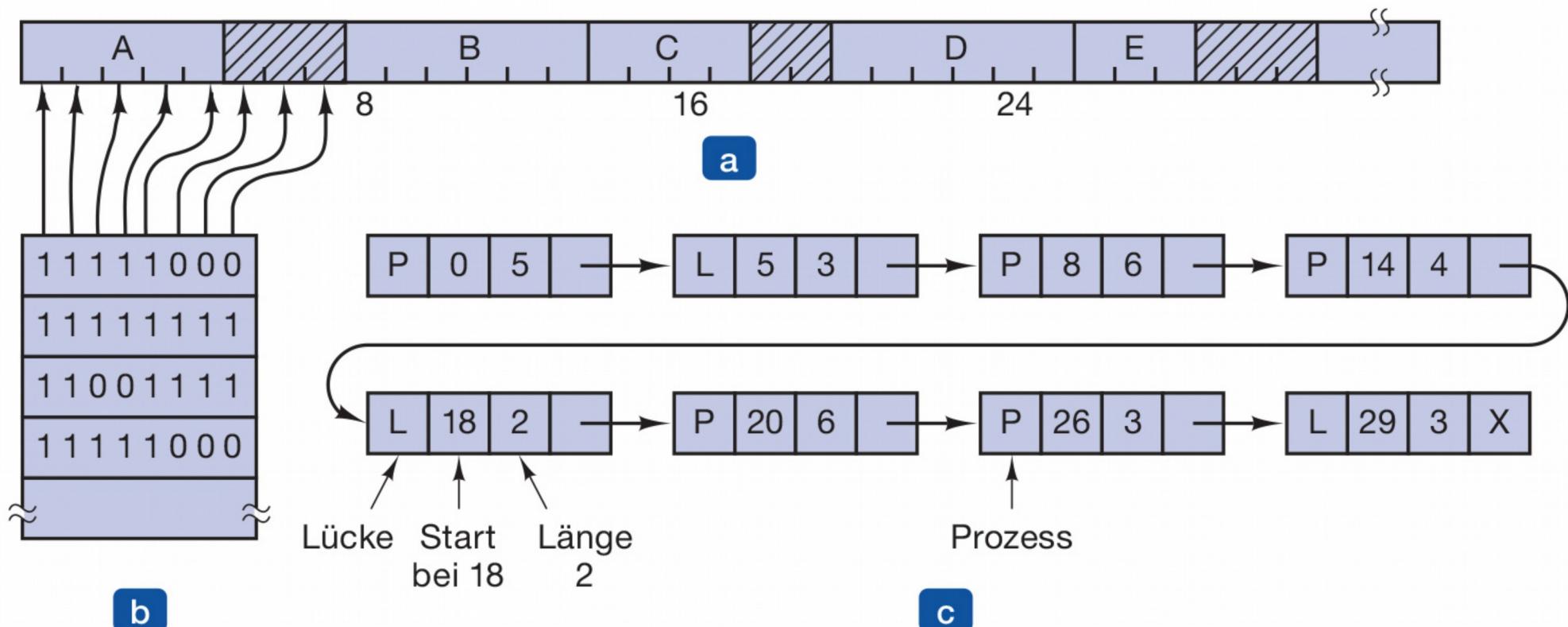
**Abbildung 3.4:** Mit der Ein- und Auslagerung von Prozessen ändert sich die Speicherbelegung. Die schraffierten Bereiche sind ungenutzt.

# Swapping (2)



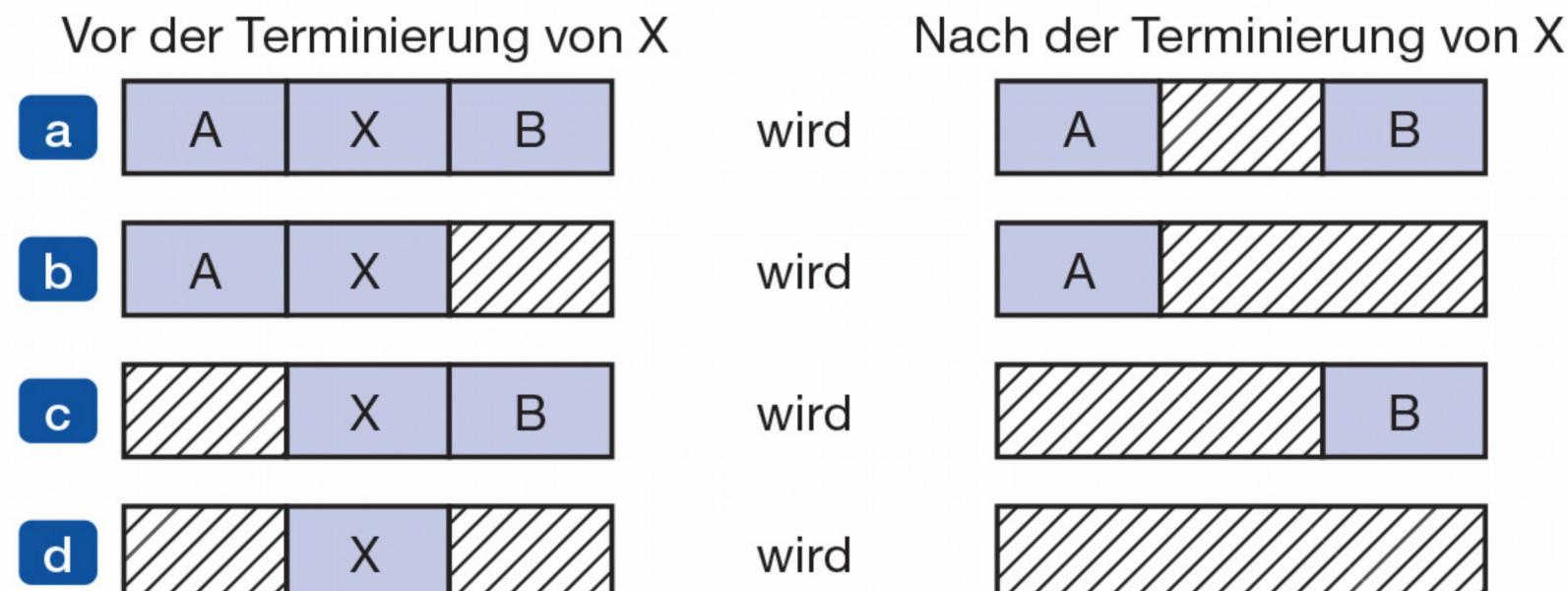
**Abbildung 3.5:** (a) Speicherreservierung für ein wachsendes Datensegment. (b) Speicherreservierung für wachsende Stack- und Datensegmente.

# Speicherverwaltung mit Bitmaps



**Abbildung 3.6:** (a) Ein Teil eines Speichers mit fünf Prozessen und drei Lücken. Die Teilstiche markieren die Grenzen der Belegungseinheiten. Die schraffierten Bereiche sind frei (0 in der Bitmap). (b) Die zugehörige Bitmap. (c) Dieselbe Information als Liste.

# Speicherverwaltung mit verlinkten Listen



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

# Algorithmen für die Speicherverwaltung

- First fit → erste ausreichende Lücke
- Next fit → nächster freier Speicher; startet an der Stelle, an der die letzten Daten eingefügt wurden (ringförmiges Durchsuchen)
- Best fit → kleinste ausreichende Lücke. Nachteil: externe Fragmentierung
- Worst fit → größte ausreichende Lücke
- Quick fit

## 3.3 Virtueller Speicher

3.3.1 Paging

3.3.2 Seitentabellen

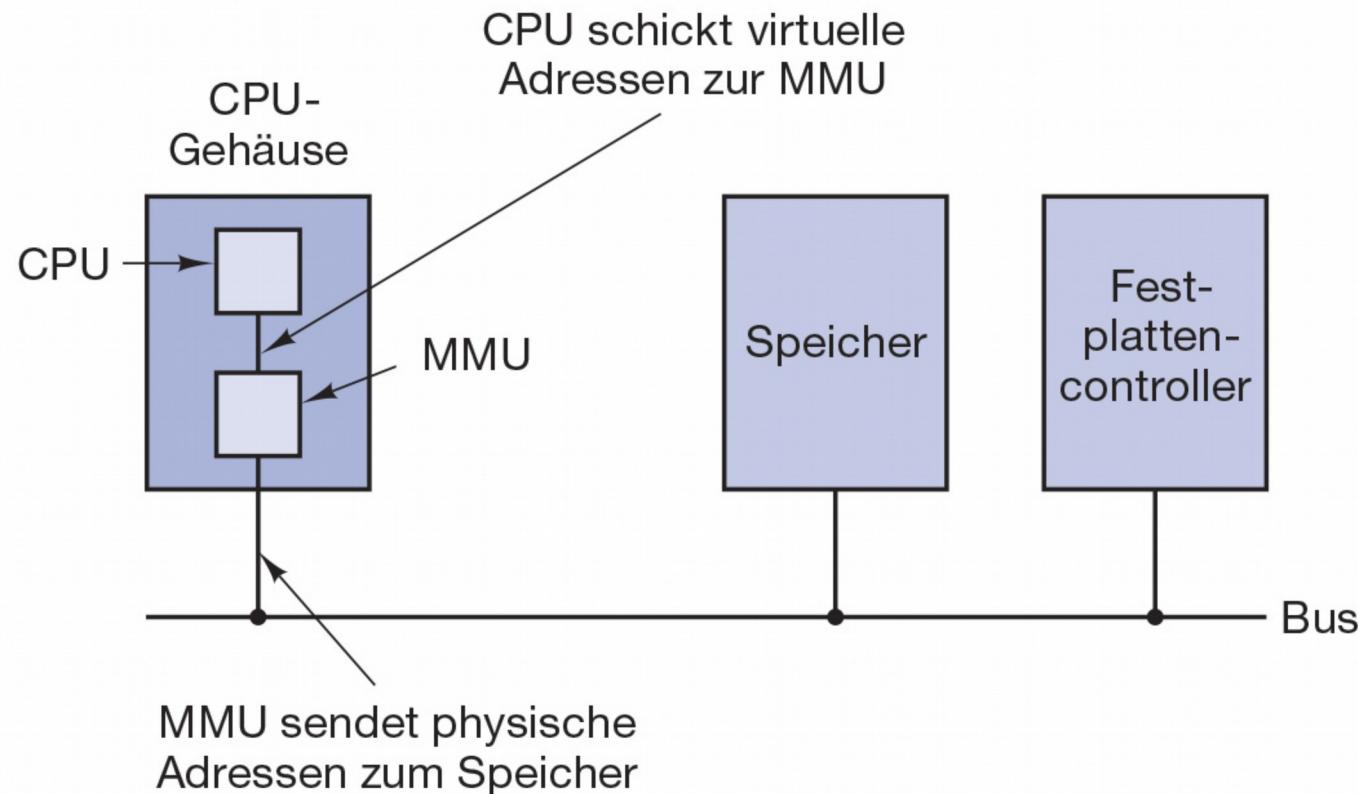
3.3.3 Beschleunigung des Paging

3.3.4 Seitentabellen für große Speicherbereiche

# Virtueller Speicher

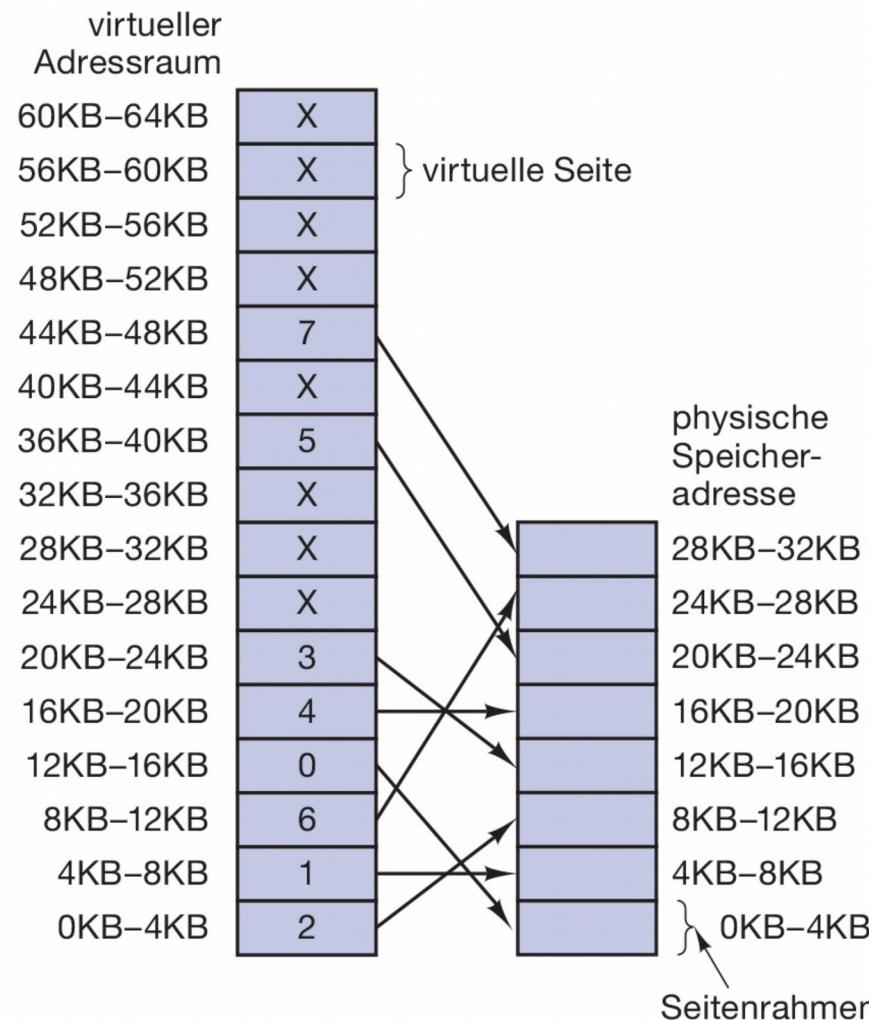
- Es besteht die Notwendigkeit Programme auszuführen, die nicht in den Speicher passen
- Eine Lösung wurde in den 1960er Jahren eingeführt:
  - Programme in kleine Stücke aufgeteilt, Overlays genannt
  - Auf der Festplatte gespeichert, werden sowohl in den Speicher kopiert als auch ausgelagert
- Virtueller Speicher: Jedes Programm hat seinen eigenen Adressraum, der in Seiten aufgeteilt ist.

# Paging (1)



**Abbildung 3.8:** Position und Funktion der MMU. Die MMU ist hier Teil des CPU-Chips, wie heutzutage üblich. Aus logischer Sicht könnte es aber auch wie von ein paar Jahren noch ein eigenständiger Chip sein.

# Paging (2)



**Abbildung 3.9:** Die Beziehung zwischen virtuellen und physischen Adressen ist durch eine Seitentabelle vorgegeben. Jede Seite beginnt bei einem Vielfachen von 4096 und endet 4096 Adressen höher. 4KB–8KB bedeutet also tatsächlich 4096–8191 und 8KB–12KB ist 8192–12287.

# Paging (3)

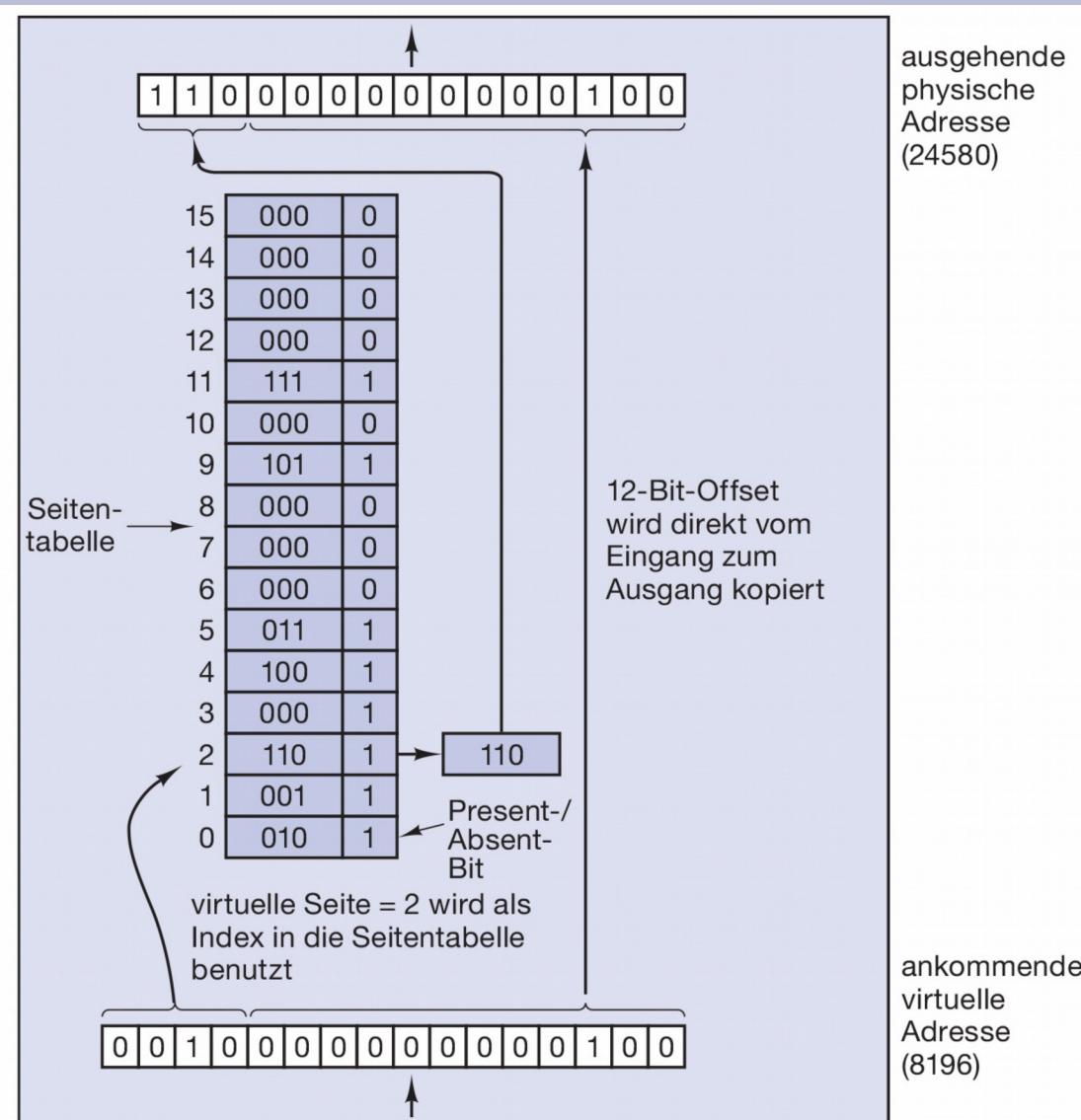
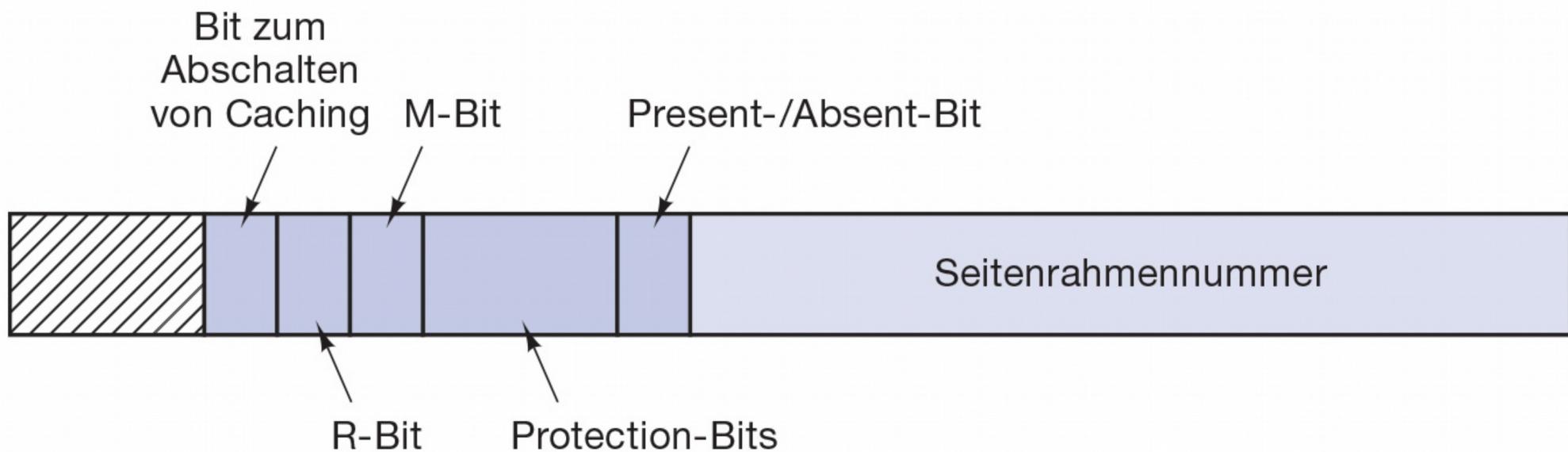


Abbildung 3.10: Interne Operation der MMU mit 16 4-KB-Seiten.

# Ein typischer Seitentabelleneintrag



**Abbildung 3.11:** Ein typischer Seitentabelleneintrag.

# Beschleunigung des Paging

Die wichtigsten Punkte sind:

1. Die Zuordnung von virtueller Adresse zu physischer Adresse muss schnell sein.
2. Wenn der virtuelle Adressraum groß ist, ist die Seitentabelle groß.

# Translation Lookaside Buffers

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

**Abbildung 3.12:** Ein TLB zur Beschleunigung des Paging.

# Eine mehrstufige Seitentabelle

Tanenbaum, A. S.; Bos, H.: Moderne Betriebssysteme. Pearson Studium 2016

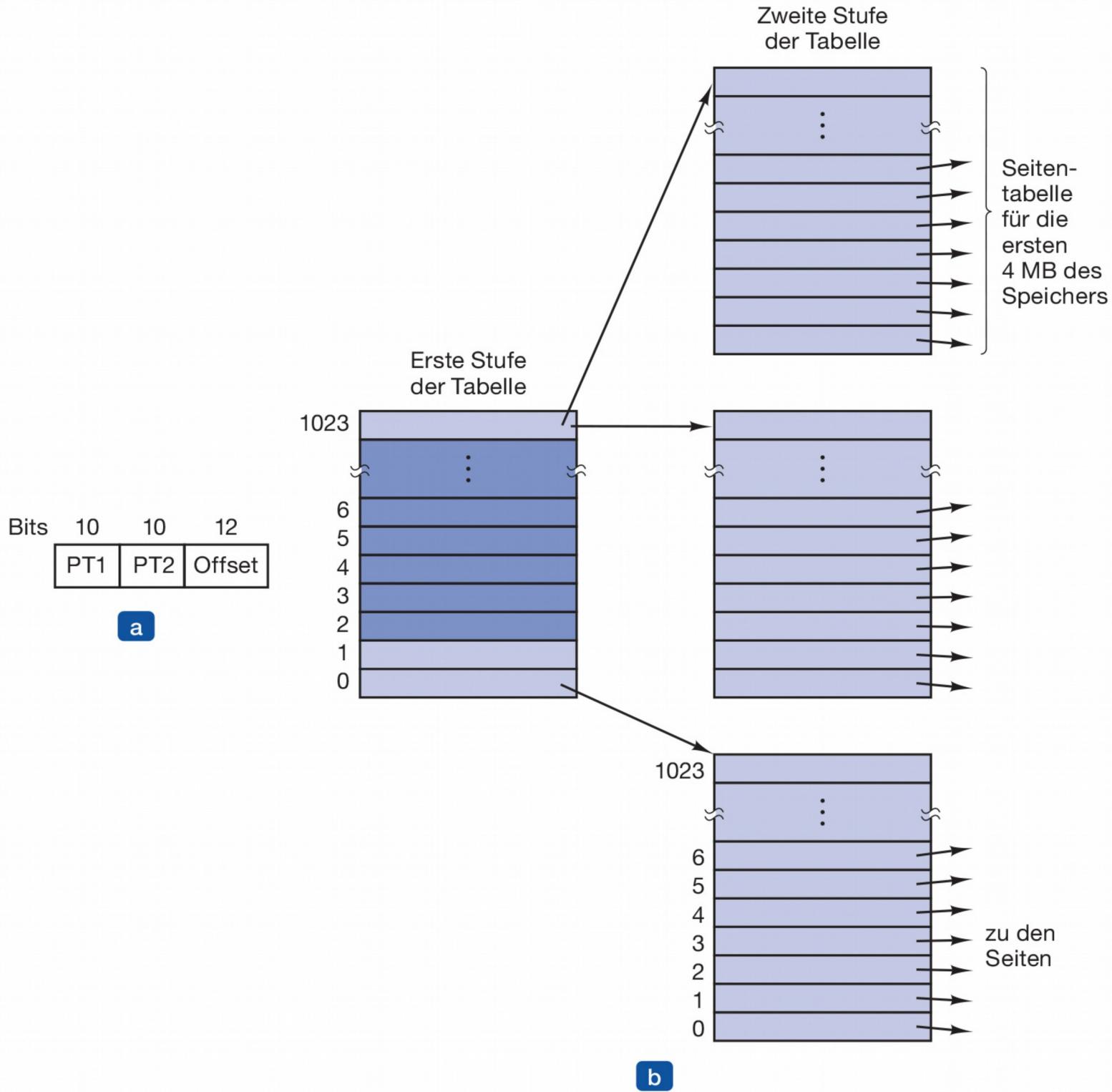
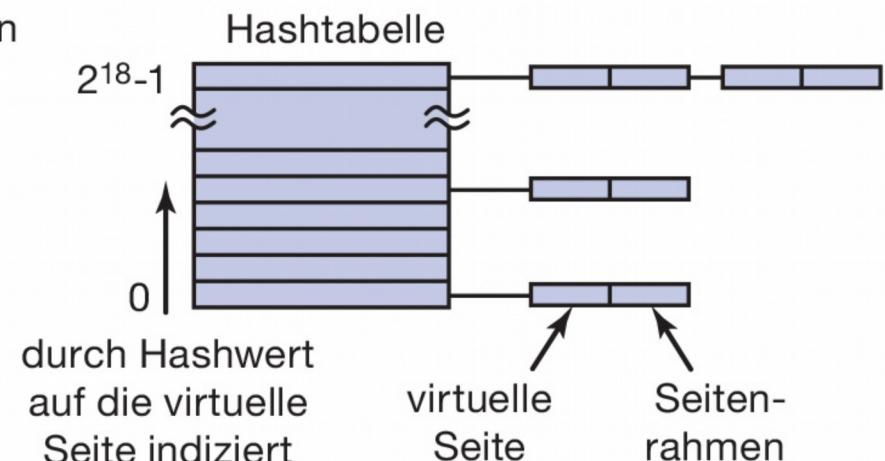
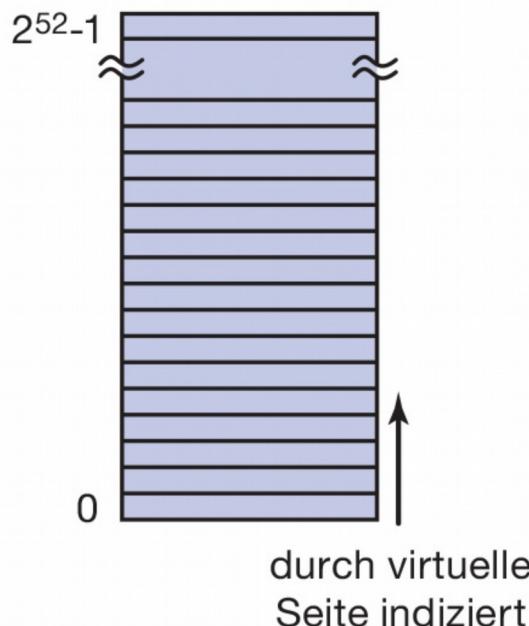


Abbildung 3.13: (a) Eine 32-Bit-Adresse mit zwei 10-Bit-Feldern für Seitennummern. (b) Zweistufige Seitentabellen.

# Eine invertierte Seitentabelle

normale Seitentabelle  
mit einem Eintrag für  
jede der  $2^{52}$  Seiten



**Abbildung 3.14:** Vergleich zwischen herkömmlicher und invertierter Seitentabelle.

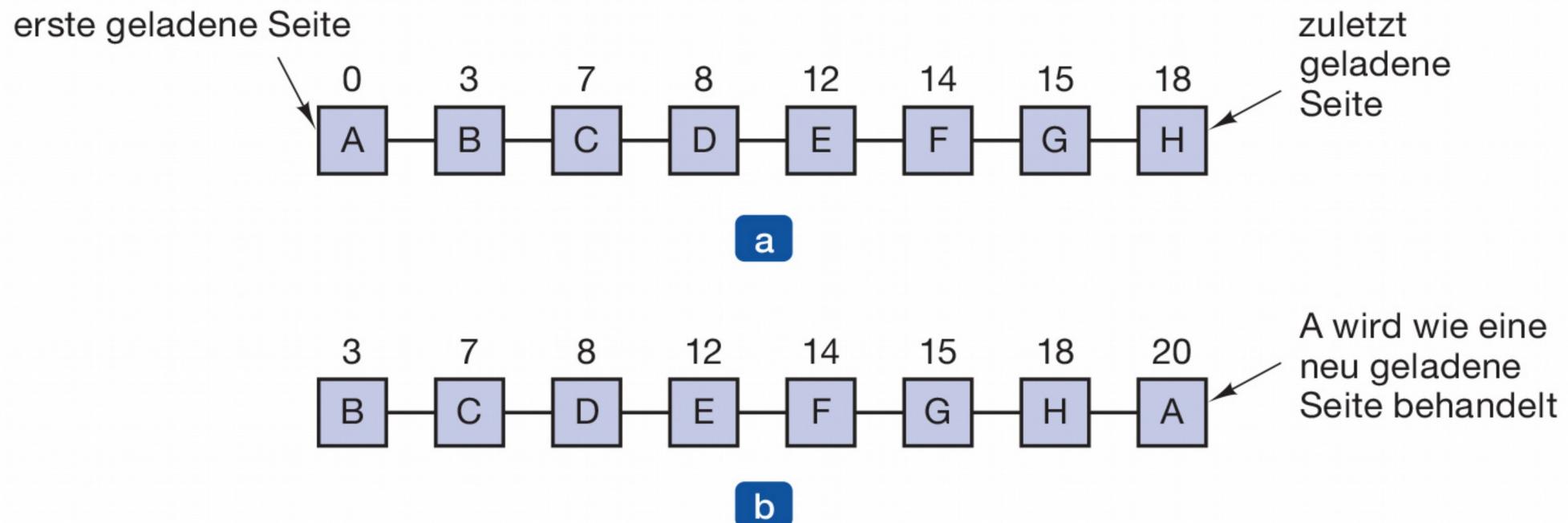
# Algorithmen für die Seitenersetzung

- Optimal algorithm
- Not recently used (NRU) algorithm
- First-in, first-out (FIFO) algorithm
- Second-chance algorithm
- Clock algorithm
- Least recently used (LRU) algorithm
- Working set algorithm
- WSClock algorithm

# Algorithmen für die Seitenersetzung

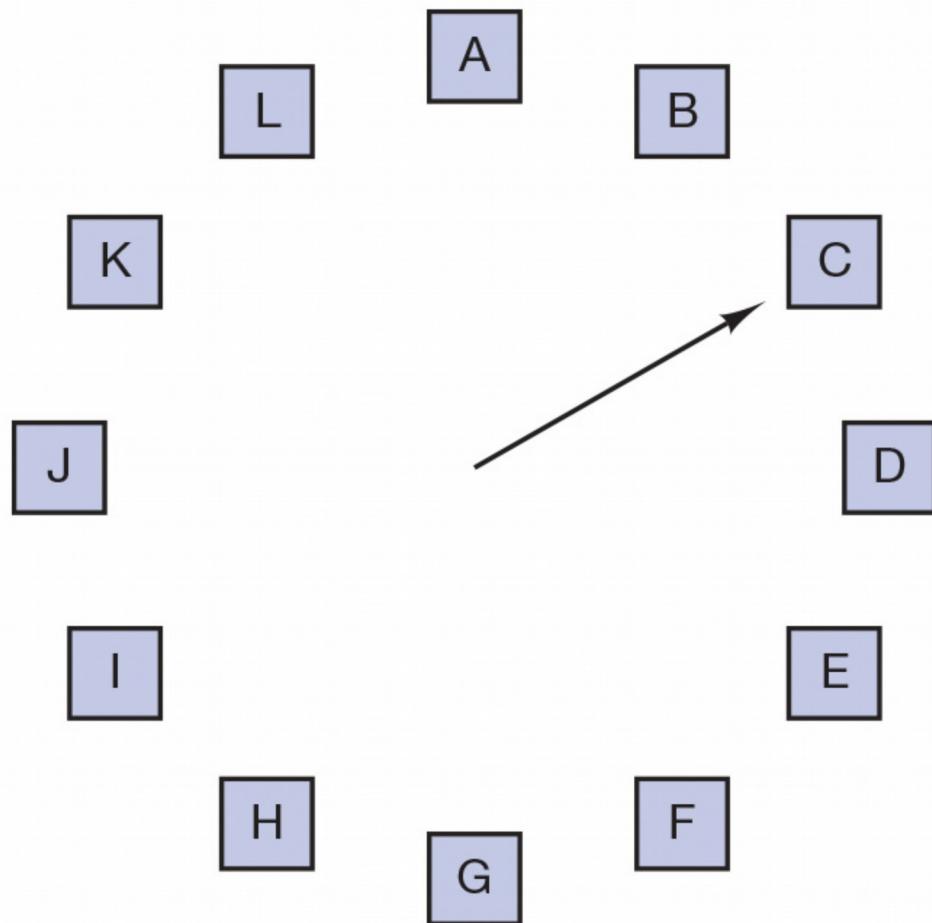
- Bei Seitenfehler inspiziert das Betriebssystem die Seiten
- Die Kategorien der Seiten basieren auf den aktuellen Werten Ihrer R und M Bits:
  - Class 0: not referenced, not modified.
  - Class 1: not referenced, modified.
  - Class 2: referenced, not modified.
  - Class 3: referenced, modified.

# Second-Chance Algorithm



**Abbildung 3.15:** Arbeitsweise von Second Chance: (a) Seiten in FIFO-Ordnung sortiert; (b) Seitenliste nach einem Seitenfehler zum Zeitpunkt 20, falls bei A das *R*-Bit gesetzt war. Die Zahlen sind Ladezeiten.

# Clock Page Replacement Algorithm



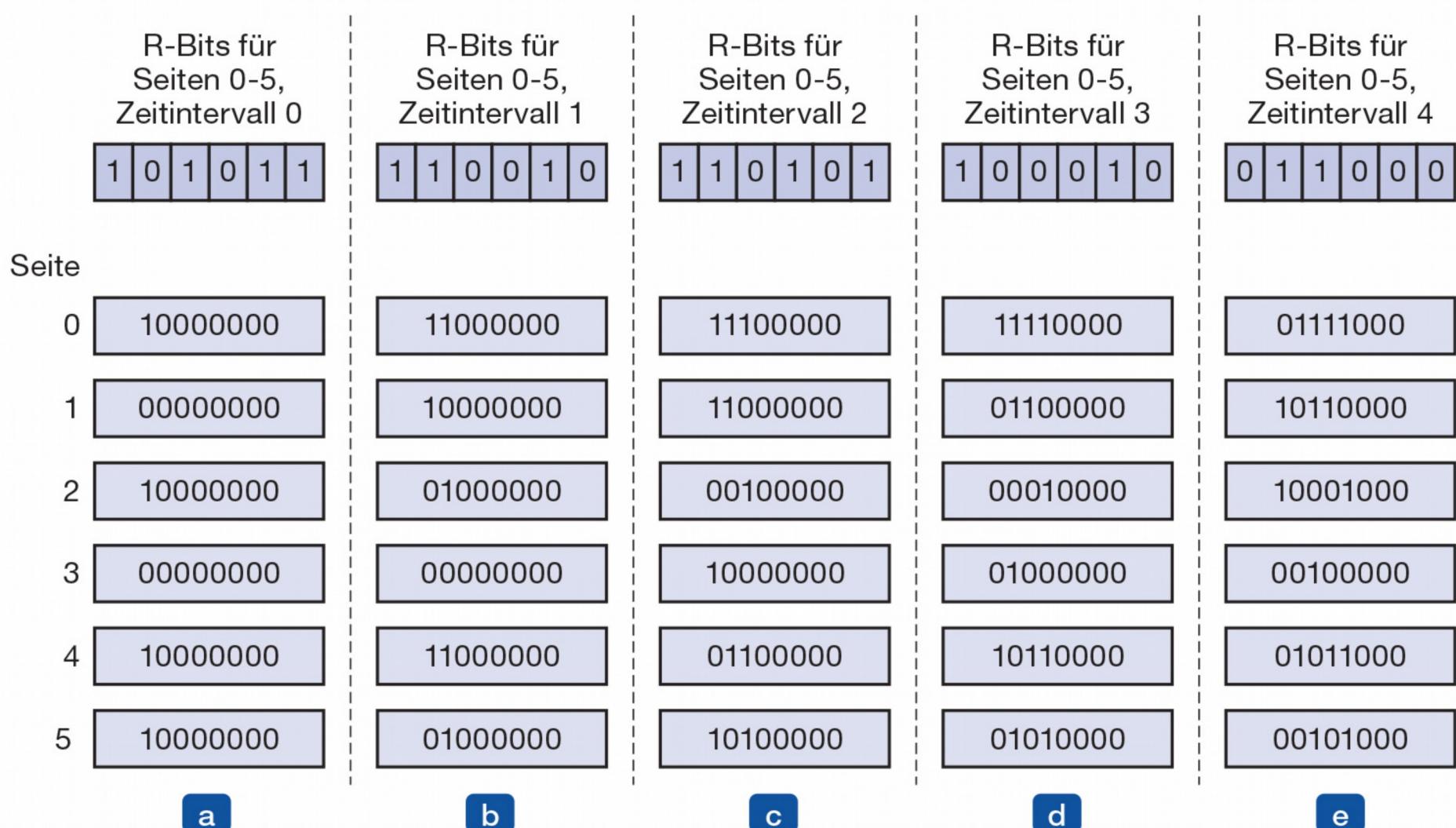
Wenn ein Seitenfehler auftritt, wird die Seite untersucht, auf die der Zeiger weist. Die Folgeaktion hängt dann vom R-Bit ab:

R = 0: verdränge die Seite

R = 1: lösche R und rücke Zeiger weiter

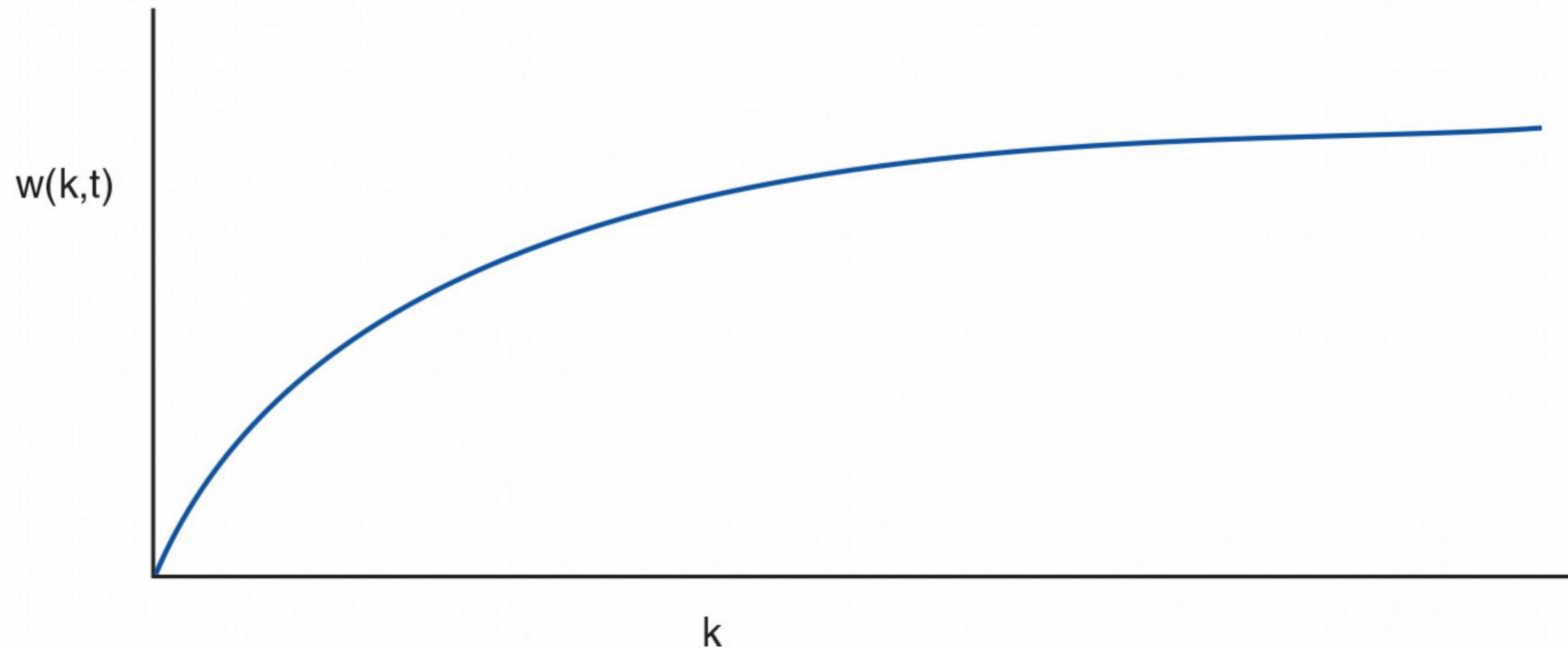
**Abbildung 3.16:** Der Clock-Algorithmus.

# Simulation von Least recently used (LRU)



**Abbildung 3.17:** Der Aging-Algorithmus ist eine Software-Simulation von LRU. Dargestellt werden die Zähler von sechs Seiten für fünf Intervalle. (a) bis (e) zeigen die Zustände nach den Intervallen 1–5.

# Working set algorithm (1)



**Abbildung 3.18:** Der Arbeitsbereich ist die Menge der Seiten, die von den letzten  $k$  Speicherzugriffen benutzt werden. Die Funktion  $w(k, t)$  ist die Größe des Arbeitsbereichs zum Zeitpunkt  $t$ .

# Working set algorithm (2)

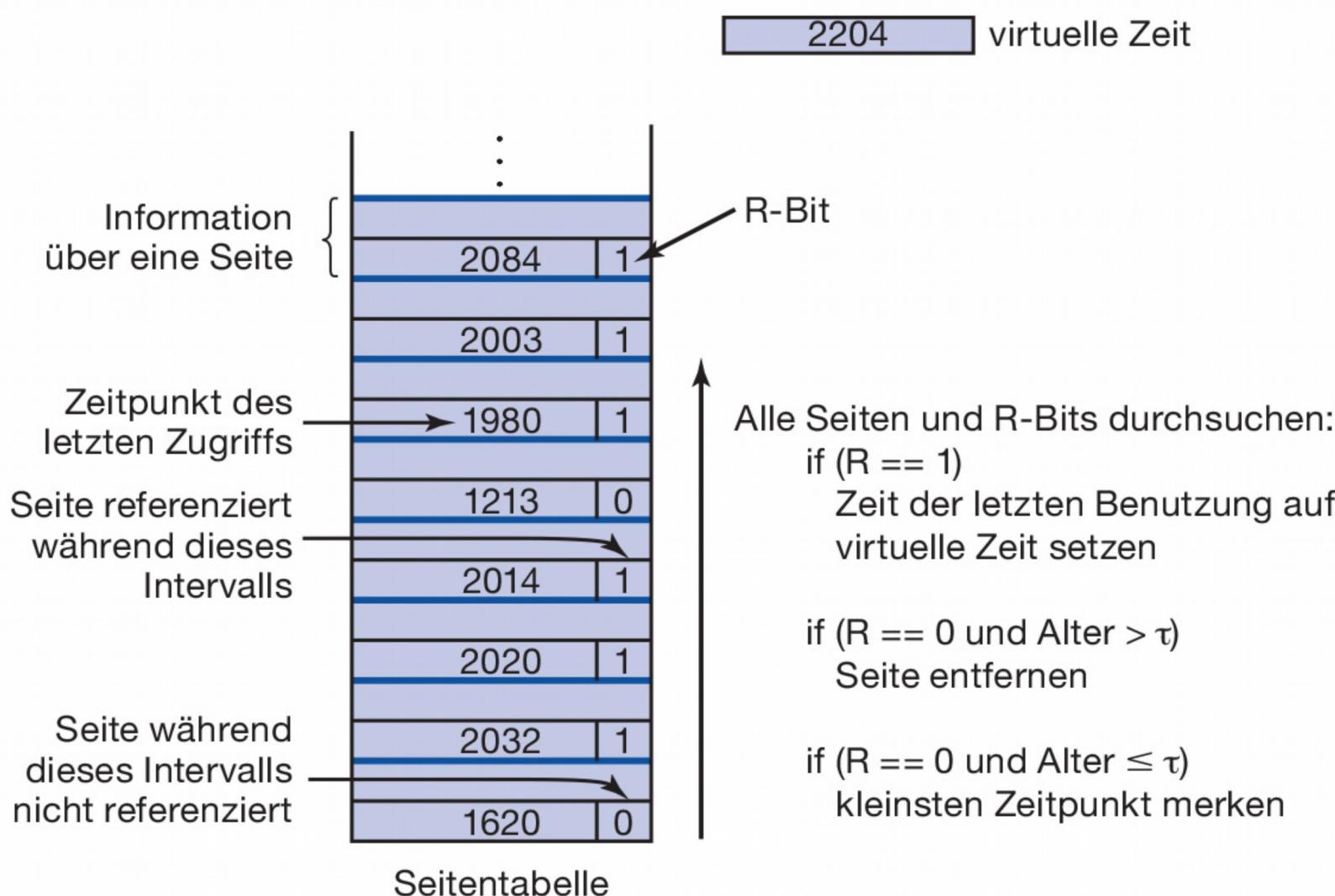
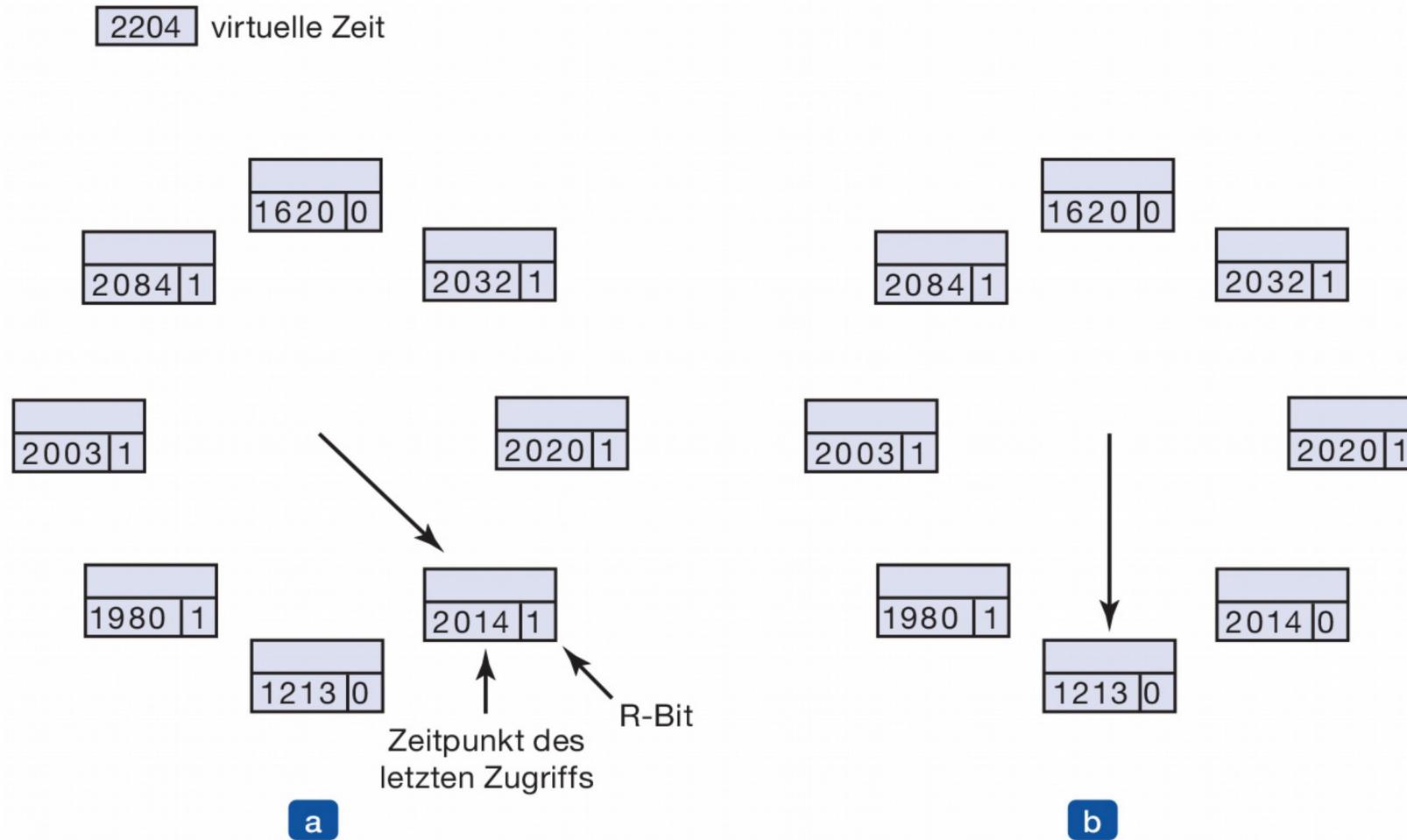
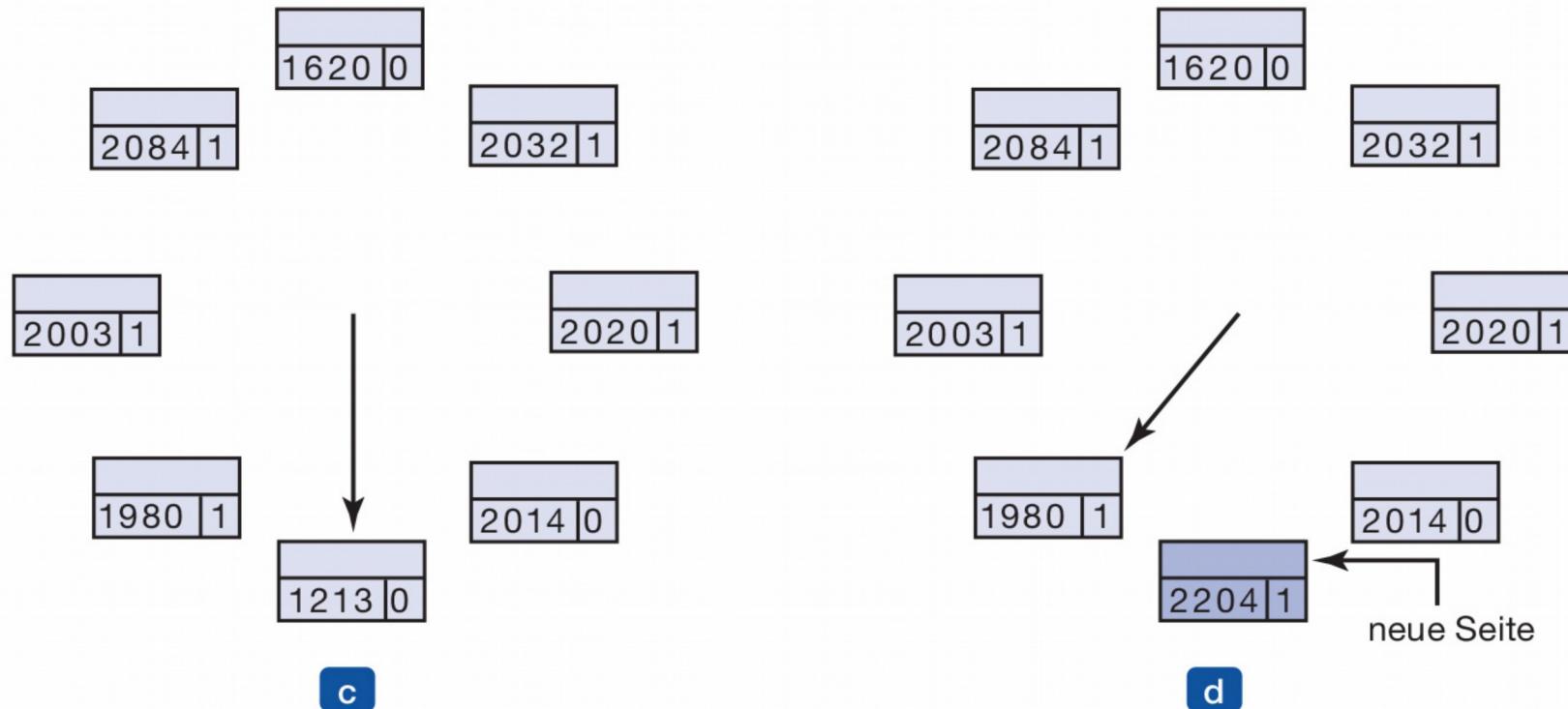


Abbildung 3.19: Der Working-Set-Algorithmus.

# WSClock Algorithm (1)



# WSClock Algorithm (2)



**Abbildung 3.20:** Die Funktionsweise des WSClock-Algorithmus: (a) und (b) zeigen, was bei  $R = 1$  passiert. (c) und (d) geben ein Beispiel für  $R = 0$ .

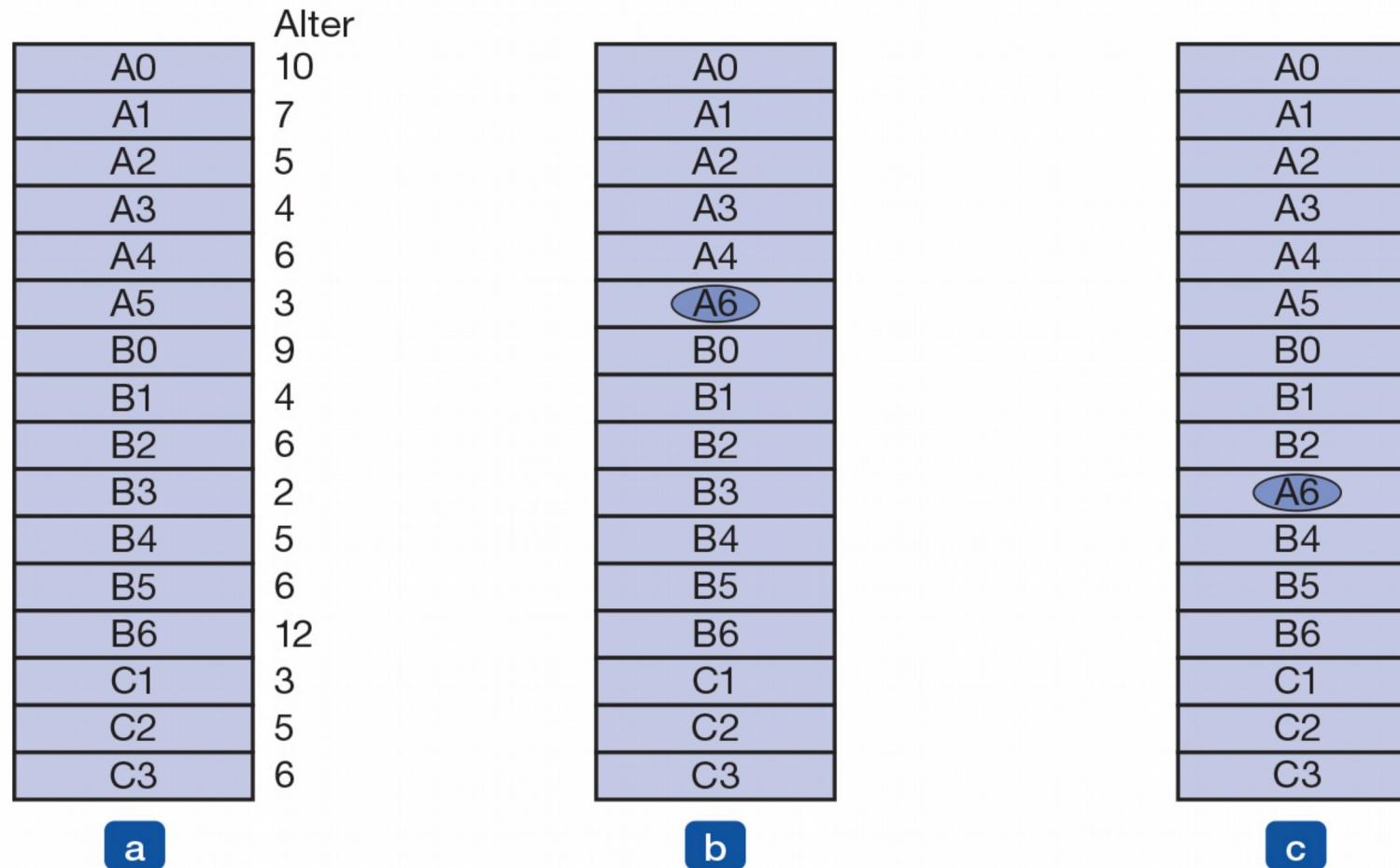
# Zusammenfassung der Seitenersetzungsalgorithmen

Algorithmus	Kommentar
Optimal	Nicht realisierbar, aber nützlich als Maßstab
NRU (Not Recently Used)	Sehr grobe Annäherung an LRU
FIFO (First In First Out)	Entfernt evtl. auch wichtige Seiten
Second Chance	Enorme Verbesserung gegenüber FIFO
Clock	Realistisch
LRU (Least Recently Used)	Exzellent, aber schwierig zu implementieren
NFU (Not Frequently Used)	Ziemlich grobe Annäherung an LRU
Aging	Effizienter Algorithmus, gute Annäherung an LRU
Working-Set	Etwas aufwendig zu implementieren
WSClock	Guter und effizienter Algorithmus

# 3.5 Entwurfskriterien für Paging-Systeme

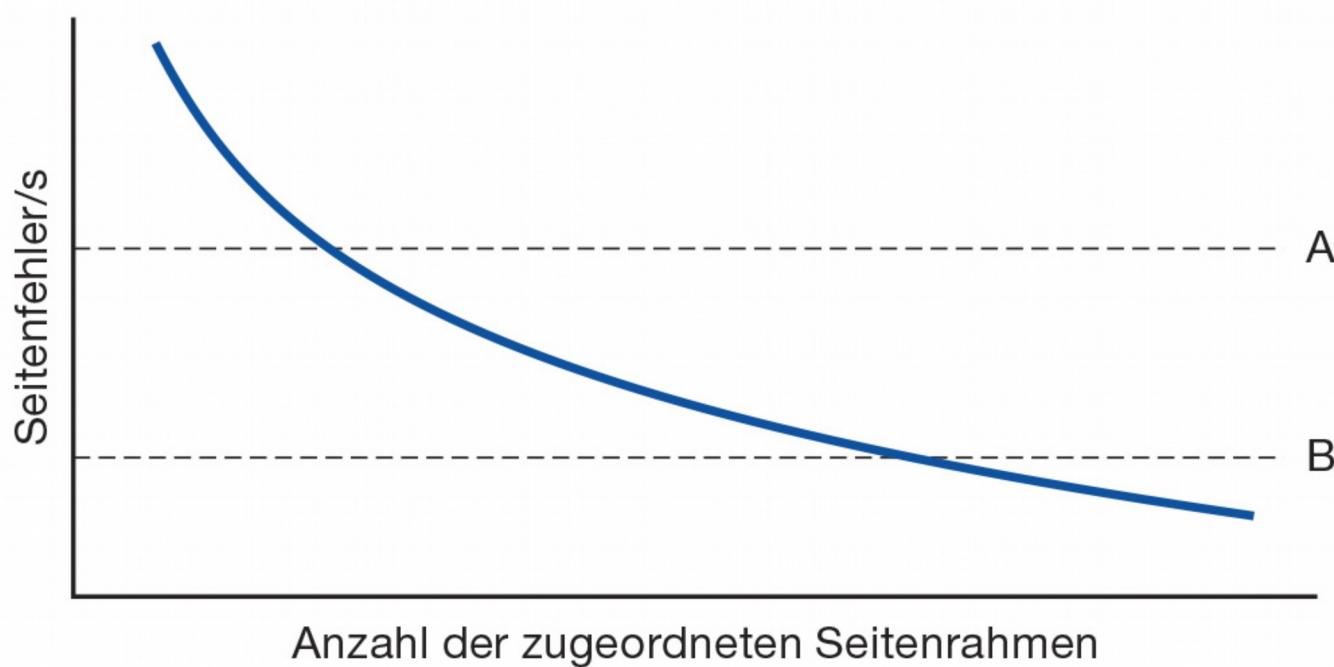
- 3.5.1 Lokale versus globale Zuteilungsstrategien
- 3.5.2 Lastkontrolle
- 3.5.3 Seitengröße
- 3.5.4 Trennung von Befehls- und Datenräumen
- 3.5.5 Gemeinsame Seiten
- 3.5.6 Gemeinsame Bibliotheken
- 3.5.7 Memory-Mapped-Dateien
- 3.5.8 Bereinigungsstrategien
- 3.5.9 Schnittstelle des virtuellen Speichersystems

# Lokale versus globale Zuteilungsstrategien (1)



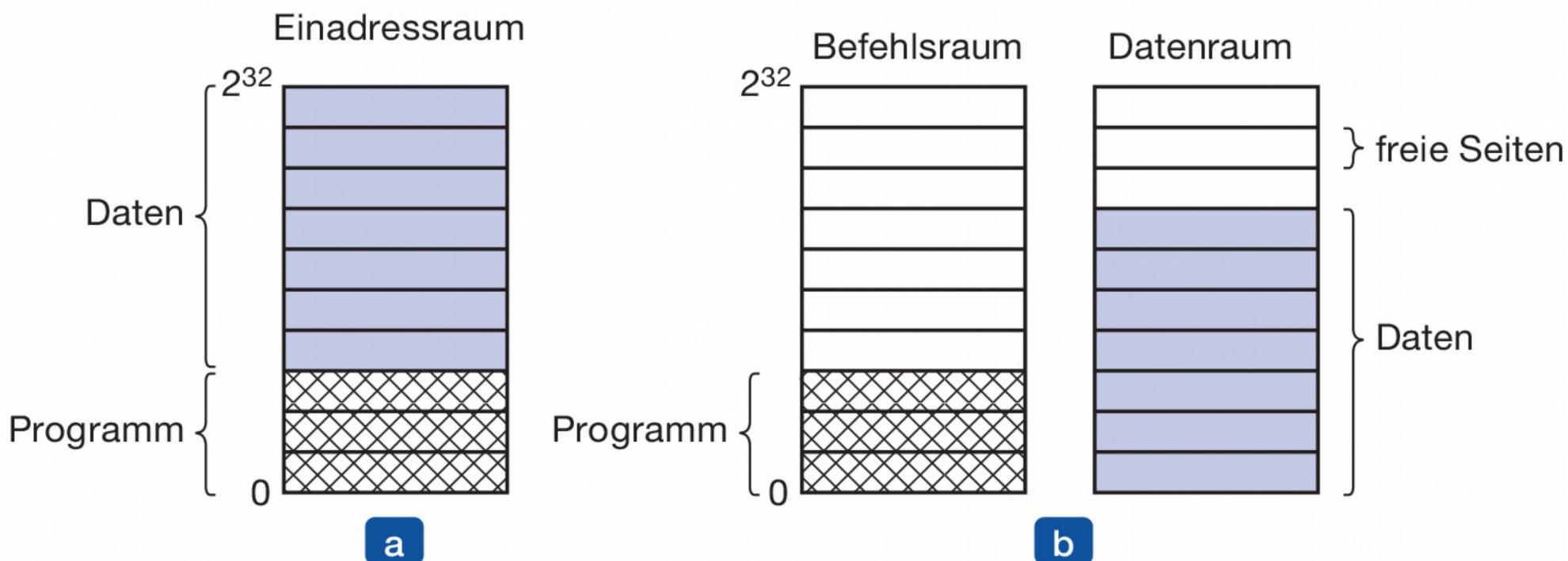
**Abbildung 3.22:** Lokale gegenüber globaler Seitenersetzung: (a) Ausgangszustand. (b) Lokale Seitenersetzung. (c) Globale Seitenersetzung.

# Lokale versus globale Zuteilungsstrategien (2)



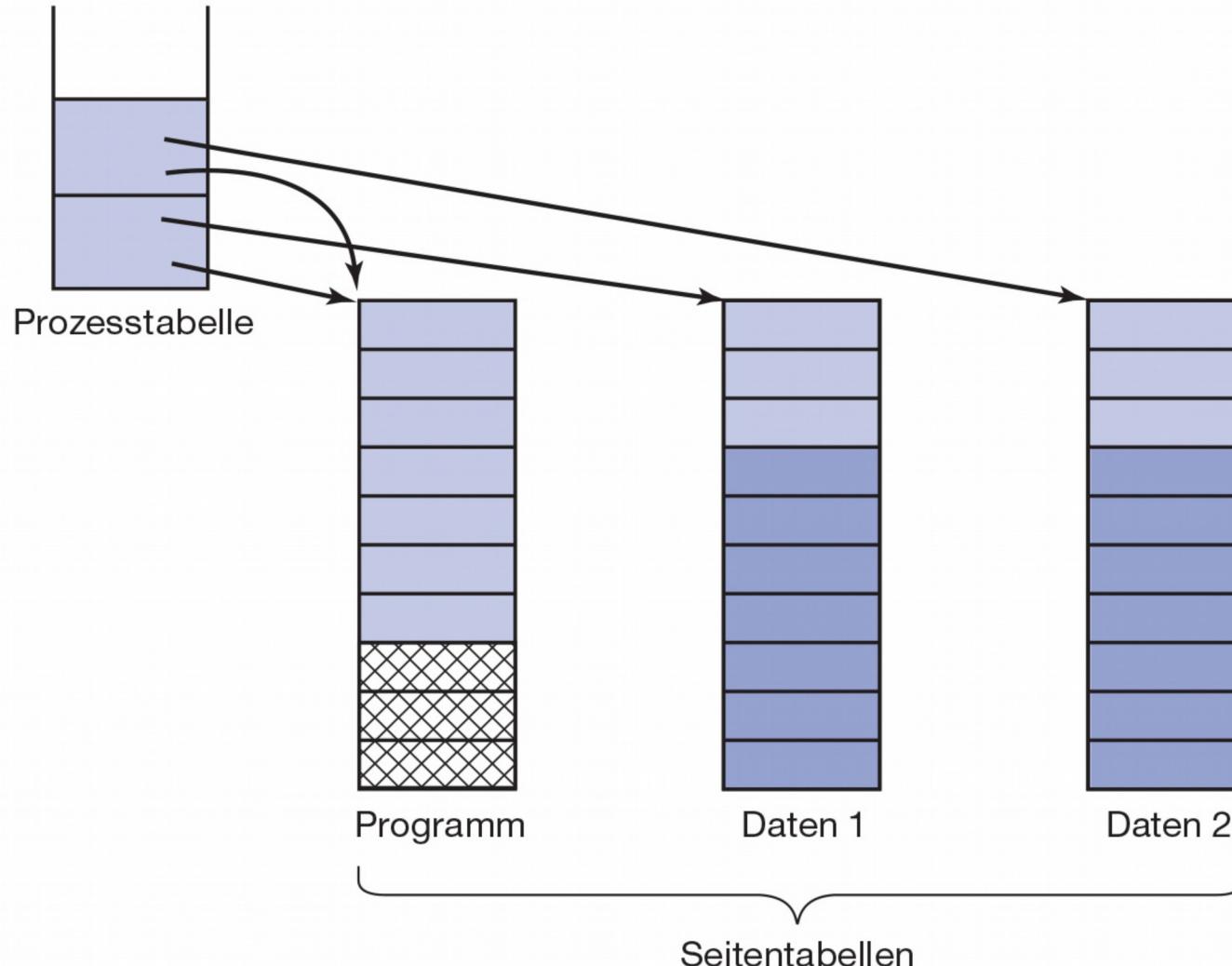
**Abbildung 3.23:** Die Seitenfehlerrate in Abhangigkeit von der Anzahl der zugewiesenen Seitenrahmen.

# Trennung von Befehls- und Datenräumen



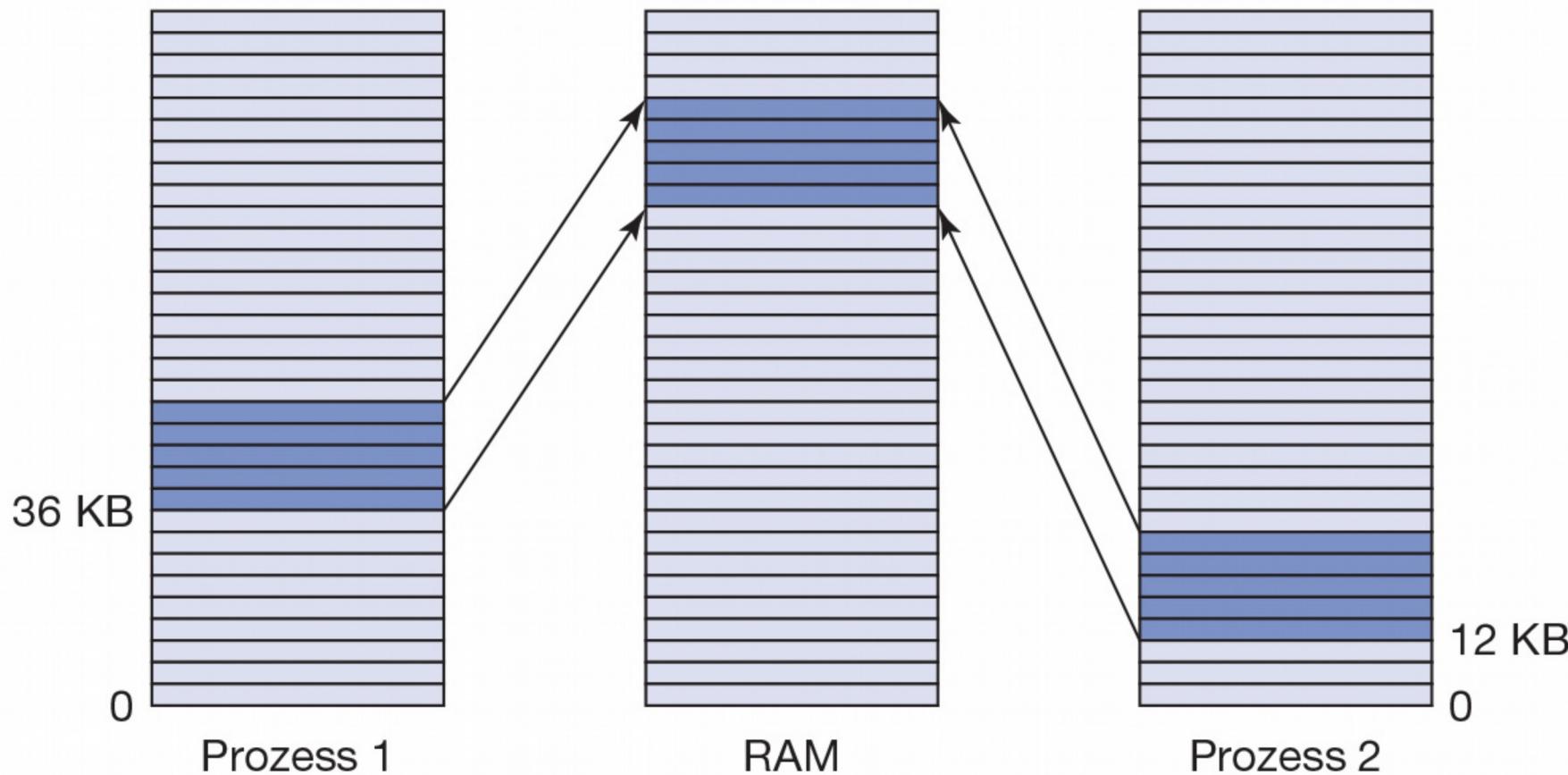
**Abbildung 3.24:** (a) Ein Adressraum. (b) Getrennte Adressräume für Code und Daten.

# Gemeinsame Seiten



**Abbildung 3.25:** Zwei Prozesse benutzen ein Programm gemeinsam, indem sie auf dieselben Seitentabellen zugreifen.

# Gemeinsame Bibliotheken



**Abbildung 3.26:** Eine gemeinsame Bibliothek, die von zwei Prozessen benutzt wird.

# Memory-Mapped-Dateien

Memory Mapping: → Speichereinblendung:

## 1. Memory Mapped I/O

Bei Memory Mapped I/O wird der externe Speicher einer Komponente (z. B. Grafikkarte) in den Arbeitsspeicher eingeblendet („gemappt“), um einen schnelleren Zugriff auf diesen zu ermöglichen (anstelle eines Schnittstellenzugriffs).

## 2. Caching eines Datensatzes.

Bei der in den Arbeitsspeicher eingeblendeten Datei wird ein Bereich der Datei in einen Speicherbereich eines laufenden Prozesses eingeblendet. Das Schreiben in den Speicherbereich bewirkt einen indirekten Schreibzugriff bezüglich der Datei.

# Bereinigungsstrategien

Paging-Deamon:

- Prüft regelmäßig den Zustand des Speicher
- Sind nicht genügend Seitenrahmen frei:
  - Wählt mit Seitenersetzungsalgorithmus Seiten für Auslagerung aus
  - Wenn Seite geändert wurde: zurückschreiben auf Platte
  - Inhalt der Seite bleibt aber im Speicher (kann ggf. zurückgeholt werden)

# Schnittstelle des virtuellen Speichersystems

Bisherige Annahme:

- Der virtuelle Speicher ist für Prozesse und Programmierer transparent:
  - Sichtbar ist großer virtueller Speicher
  - Tatsächlich ist der physische Speicher kleine

Fortgeschrittene Systeme:

- Programmierer hat etwas Kontrolle über die Speicherabbildung
  - Verhalten der Programme lässt sich so verbessern (auf unkonventionelle Weise)

## 3.6 Implementierungsaspekte

- 3.6.1 Aufgaben des Betriebssystems beim Paging
- 3.6.2 Behandlung von Seitenfehlern
- 3.6.3 Sicherung von unterbrochenen Befehlen
- 3.6.4 Sperren von Seiten im Speicher
- 3.6.5 Hintergrundspeicher
- 3.6.6 Trennung von Strategie und Mechanismus

# Aufgaben des Betriebssystems beim Paging

Im Leben eines Prozesses muss das Betriebssystem an vier Punkten Arbeit im Zusammenhang mit Paging leisten:

1. Erzeugung Prozess
2. Scheduler bringt Prozess zur Ausführung
3. Bei einem Seitenfehler
4. Prozess terminiert

# Behandlung von Seitenfehlern (1)

1. Die Hardware speichert den Programmzähler auf dem Stack und löst einen Sprung in den Kernel aus.
2. Assembler-Routine speichert allgemeine Register und andere flüchtige Informationen
3. System entdeckt Seitenfehler und versucht herauszufinden, welche virtuelle Seite benötigt wird
4. Sobald der durch die virtuelle Adresse verursachte Fehler bekannt ist, prüft das System, ob die Adresse gültig und der Zugriffsschutz konsistent ist.

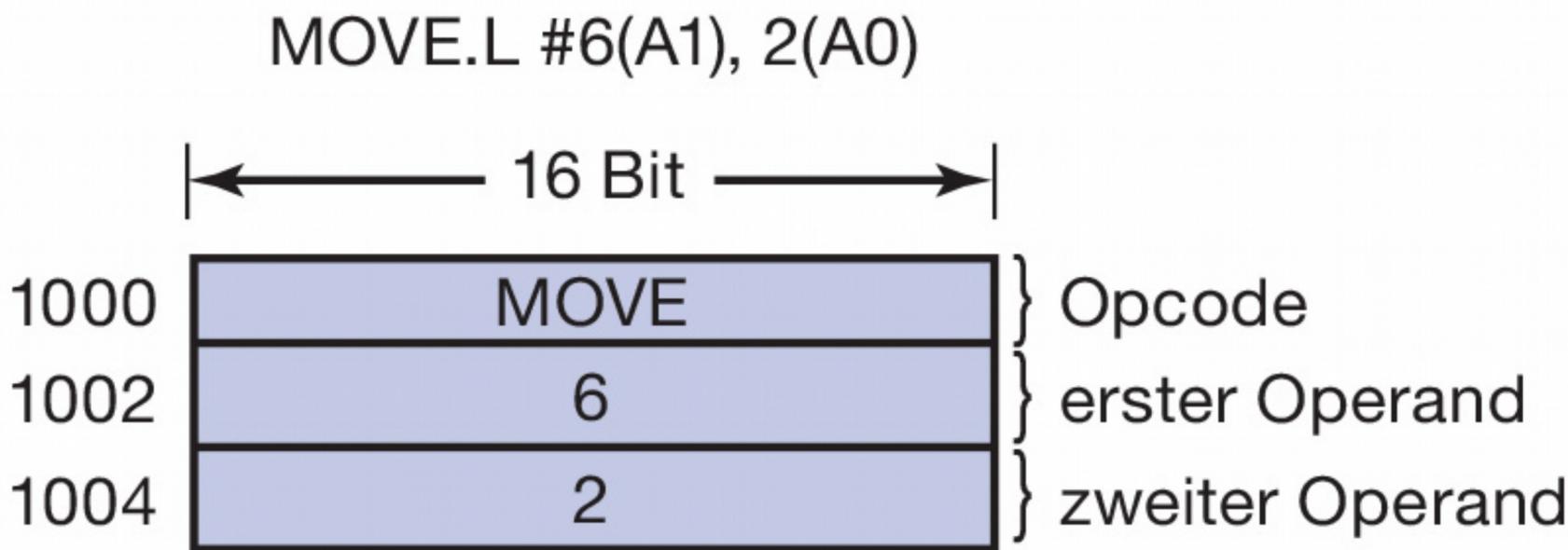
# Behandlung von Seitenfehlern (2)

5. Wenn der Frame als "dirty" markiert ist, wird die Seite für die Übertragung auf die Festplatte geplant, der Kontextwechsel findet statt und der fehlerhafte Prozess wird ausgesetzt.
6. Sobald der Frame sauber ist, sucht das Betriebssystem die Festplattenadresse, wo die benötigte Seite ist, und plant die Festplattenoperation, um sie zu laden.
7. Wenn der Festplatten-Interrupt anzeigt, dass die Seite angekommen ist, werden die Tabellen aktualisiert, um die Position zu reflektieren, und der Frame wird als normal markiert.

# Behandlung von Seitenfehlern (3)

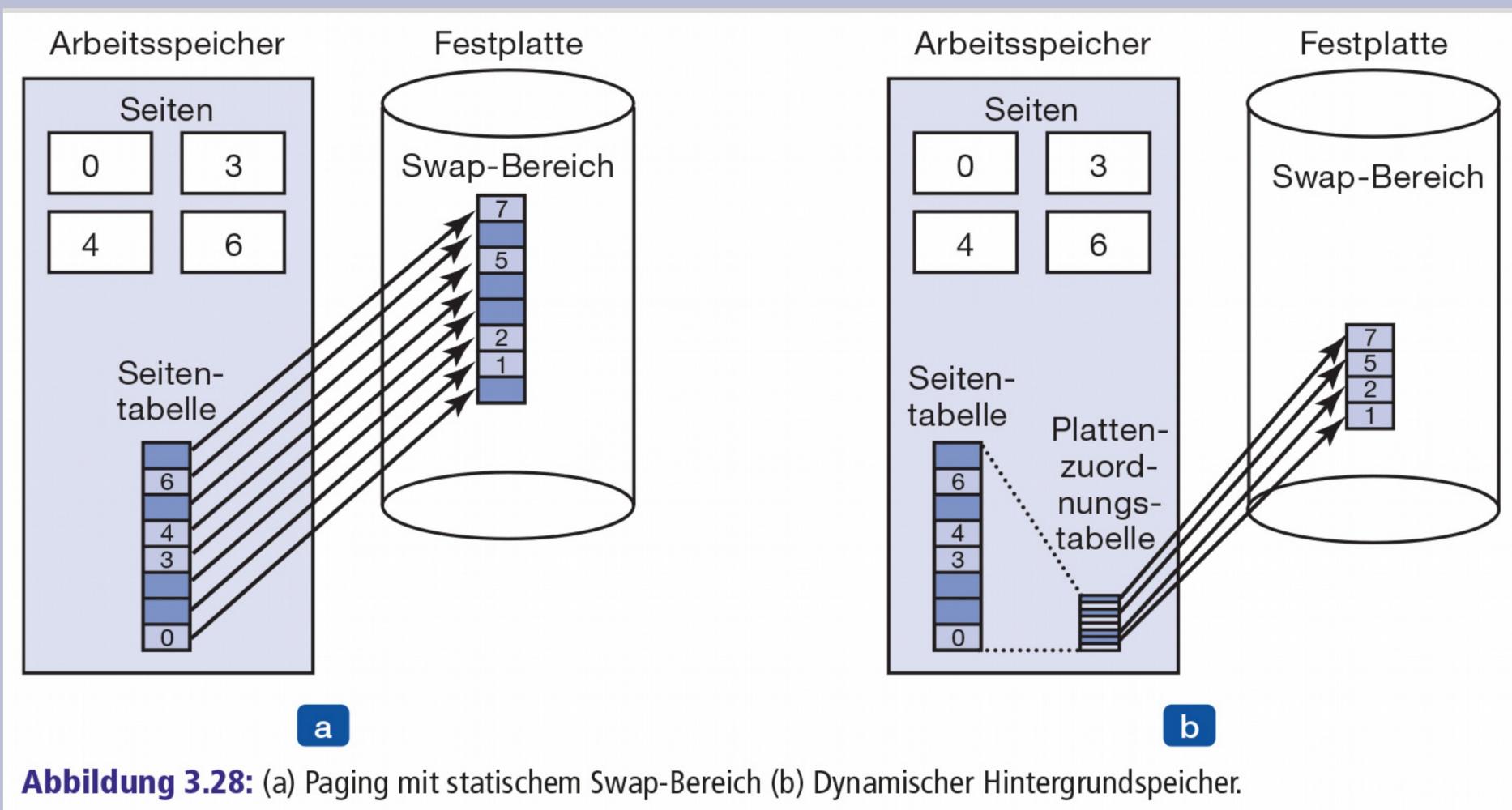
8. Sicherung der Fehlinstruktion in dem Zustand, wo es begann, und zurücksetzen des Programmzählers
9. Der Fehlerprozess ist geplant, das Betriebssystem kehrt zur Routine zurück, die es aufgerufen hat.
10. Routine lädt Register und andere Zustandsinformationen neu und kehrt zum Benutzerraum zurück, um die Ausführung fortzusetzen

# Sicherung von unterbrochenen Befehlen



**Abbildung 3.27:** Ein Befehl, der einen Seitenfehler auslöst.

# Hintergrundspeicher



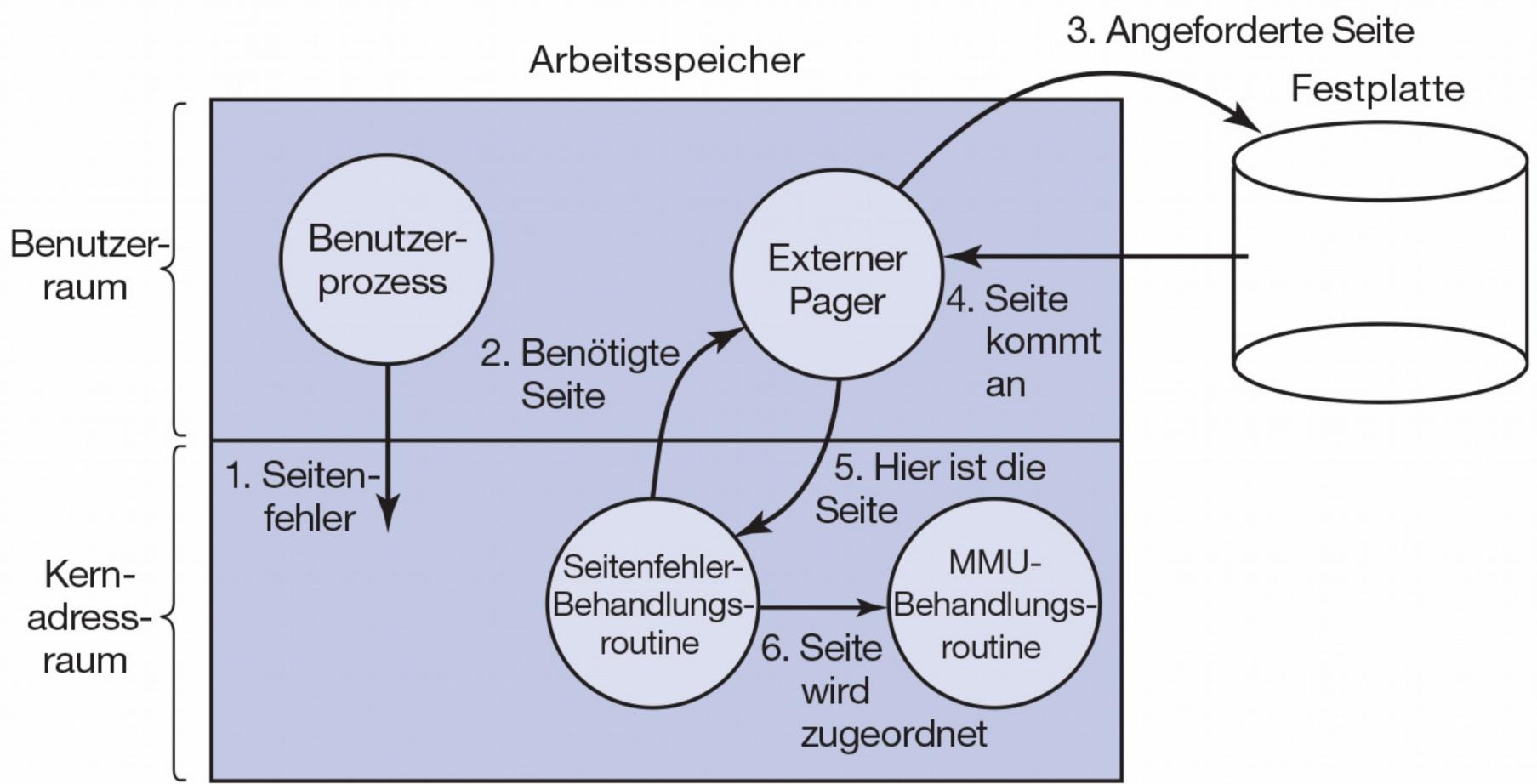
**Abbildung 3.28:** (a) Paging mit statischem Swap-Bereich (b) Dynamischer Hintergrundspeicher.

# Trennung von Strategie und Mechanismus (1)

Memory Management System ist in drei Teile unterteilt:

1. Ein Low-Level-MMU-Handler.
2. Ein Seitenfehlerhandler, der Teil des Kernels ist.
3. Ein externer Pager läuft im Benutzerbereich.

# Trennung von Strategie und Mechanismus (2)



**Abbildung 3.29:** Seitenfehlerbehandlung mit externem Pager.

## 3.7 Segmentierung

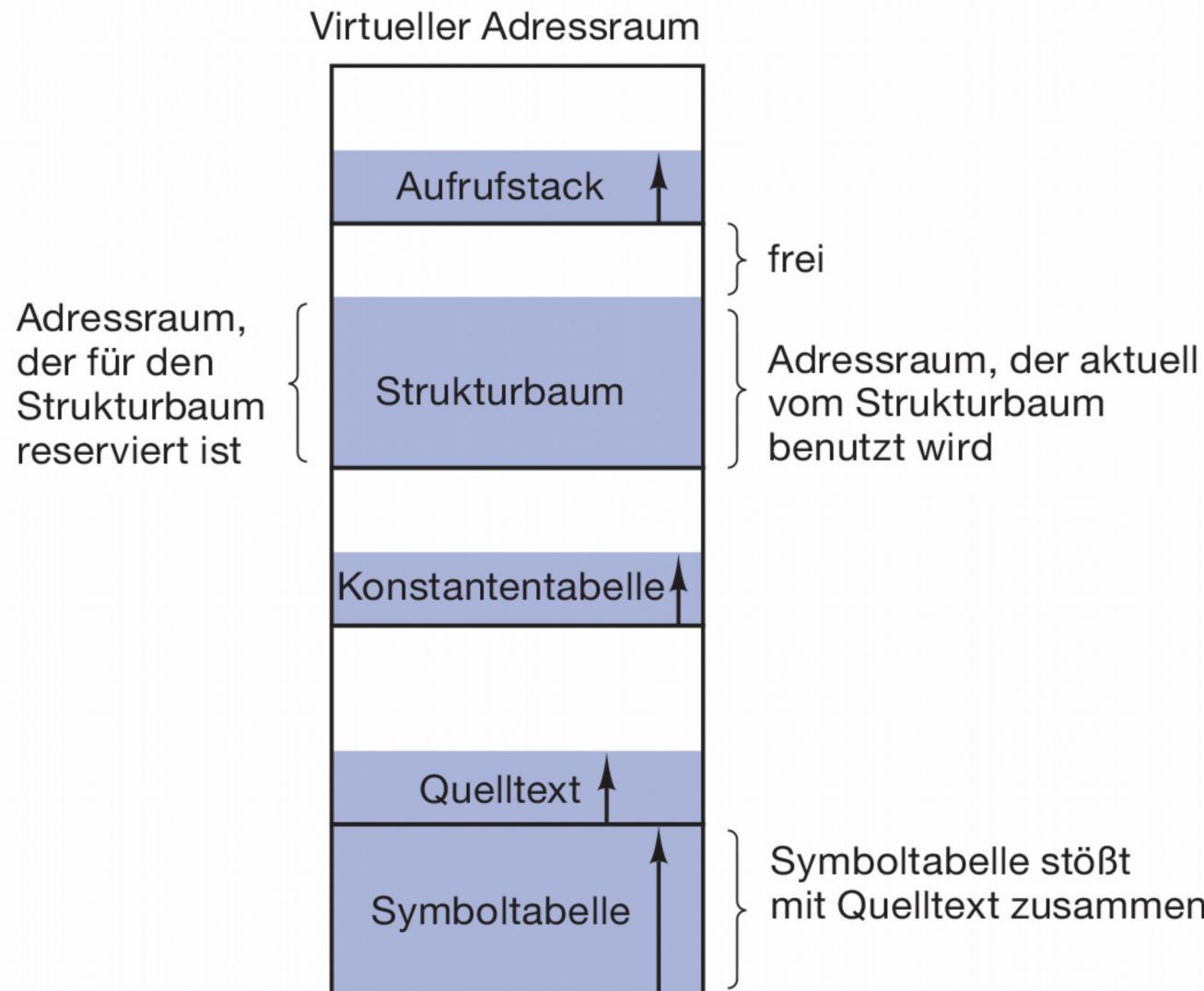
- 3.7.1 Implementierung von Segmentierung
- 3.7.2 Segmentierung mit Paging: MULTICS
- 3.7.3 Segmentierung mit Paging: x86 von Intel

# Segmentierung (1)

Beispiele für vom Compiler generierte Tabellen:

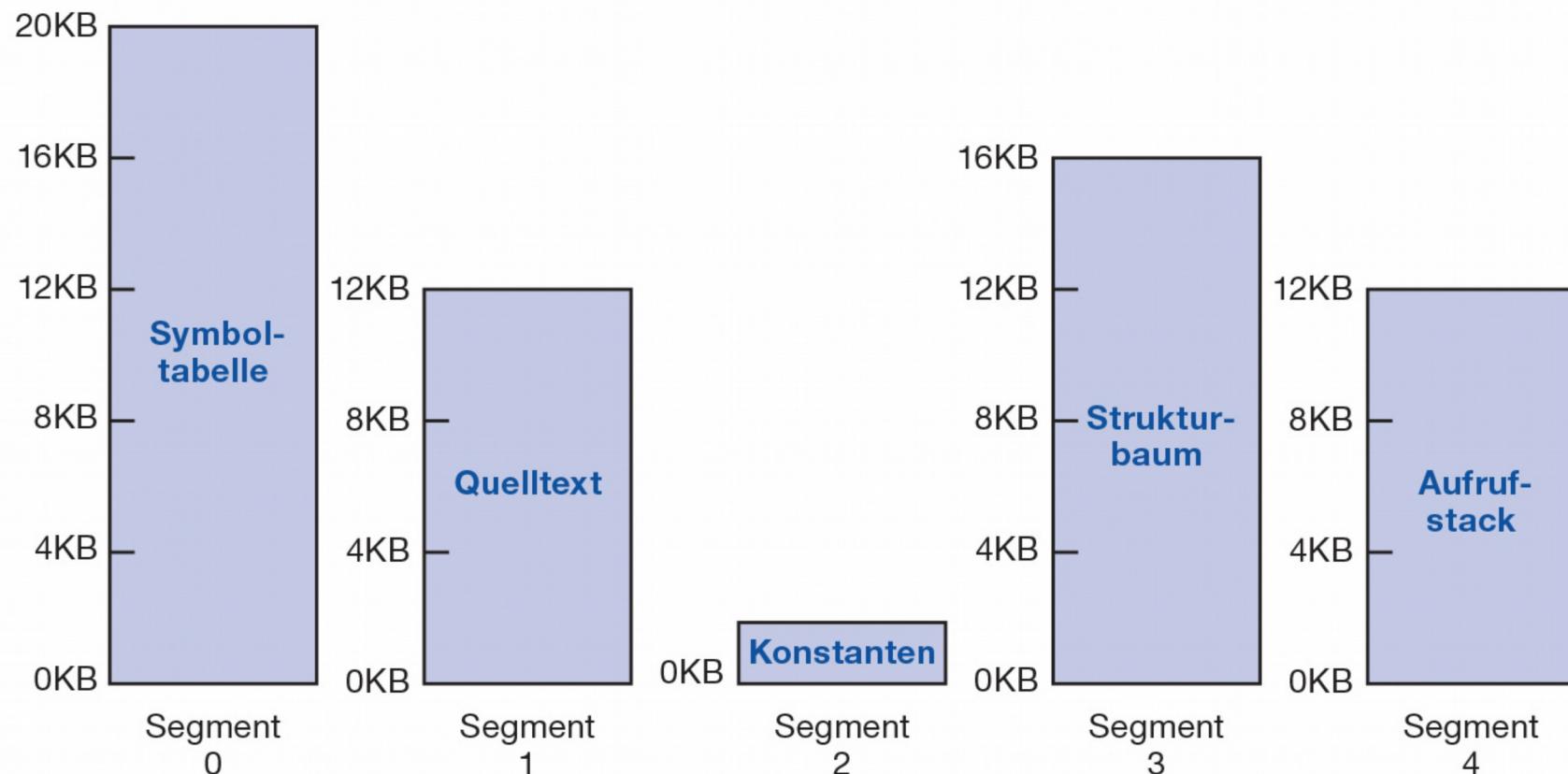
1. Der für das gedruckte Listing gespeicherte Quelltext
2. Die Symboltabelle, Namen und Attribute von Variablen.
3. Die Tabelle, welche Integer- und Gleitkommakonstanten enthält.
4. Der Parse-Baum, die syntaktische Analyse des Programms.
5. Der für Prozeduraufrufe verwendete Stack innerhalb des Compilers.

# Segmentierung (2)



**Abbildung 3.30:** In einem eindimensionalen Adressraum können wachsende Tabellen kollidieren.

# Segmentierung (3)



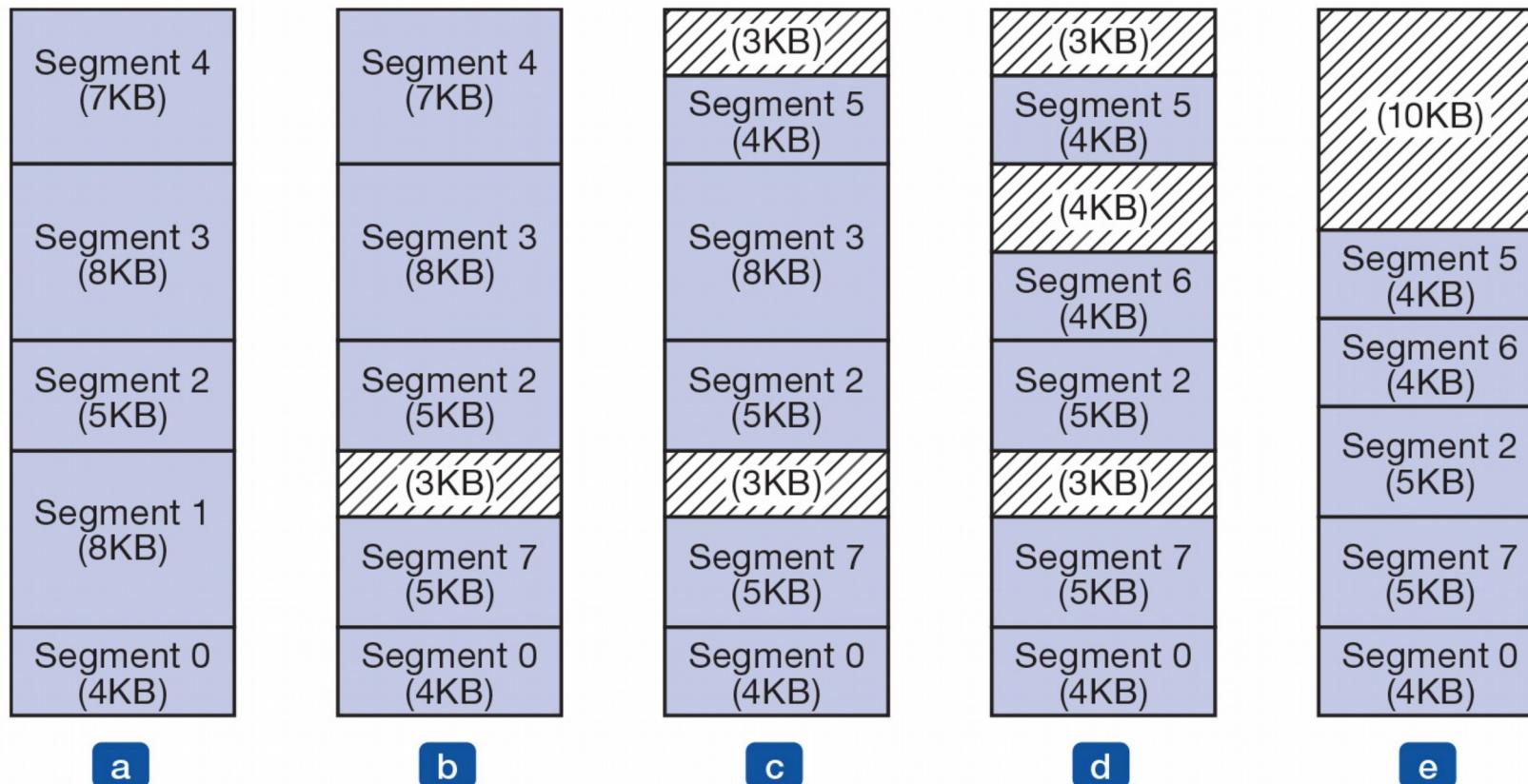
**Abbildung 3.31:** Segmentierter Speicher ermöglicht Tabellen, die unabhängig voneinander ihre Größe ändern

# Segmentierung (4)

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

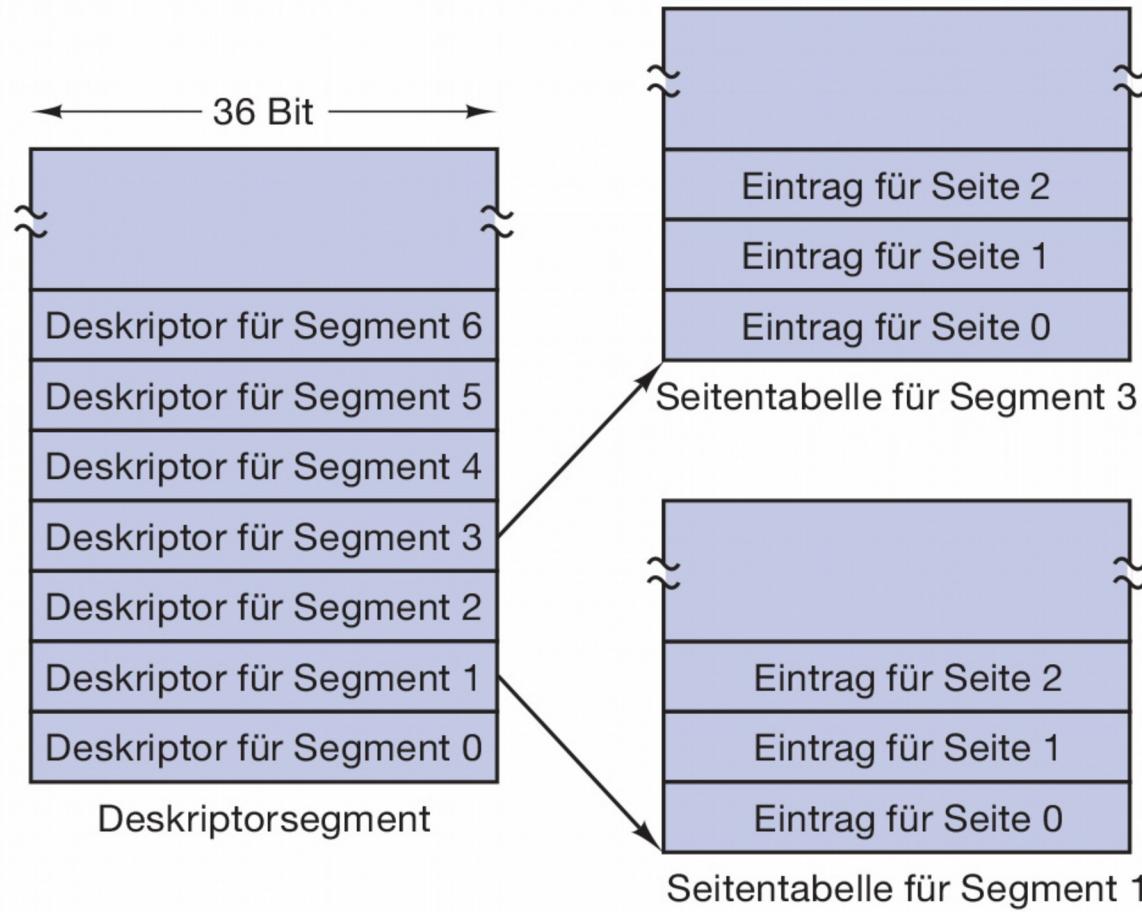
Abbildung 3.32: Vergleich von Paging und Segmentierung.

# Implementierung der reinen Segmentierung



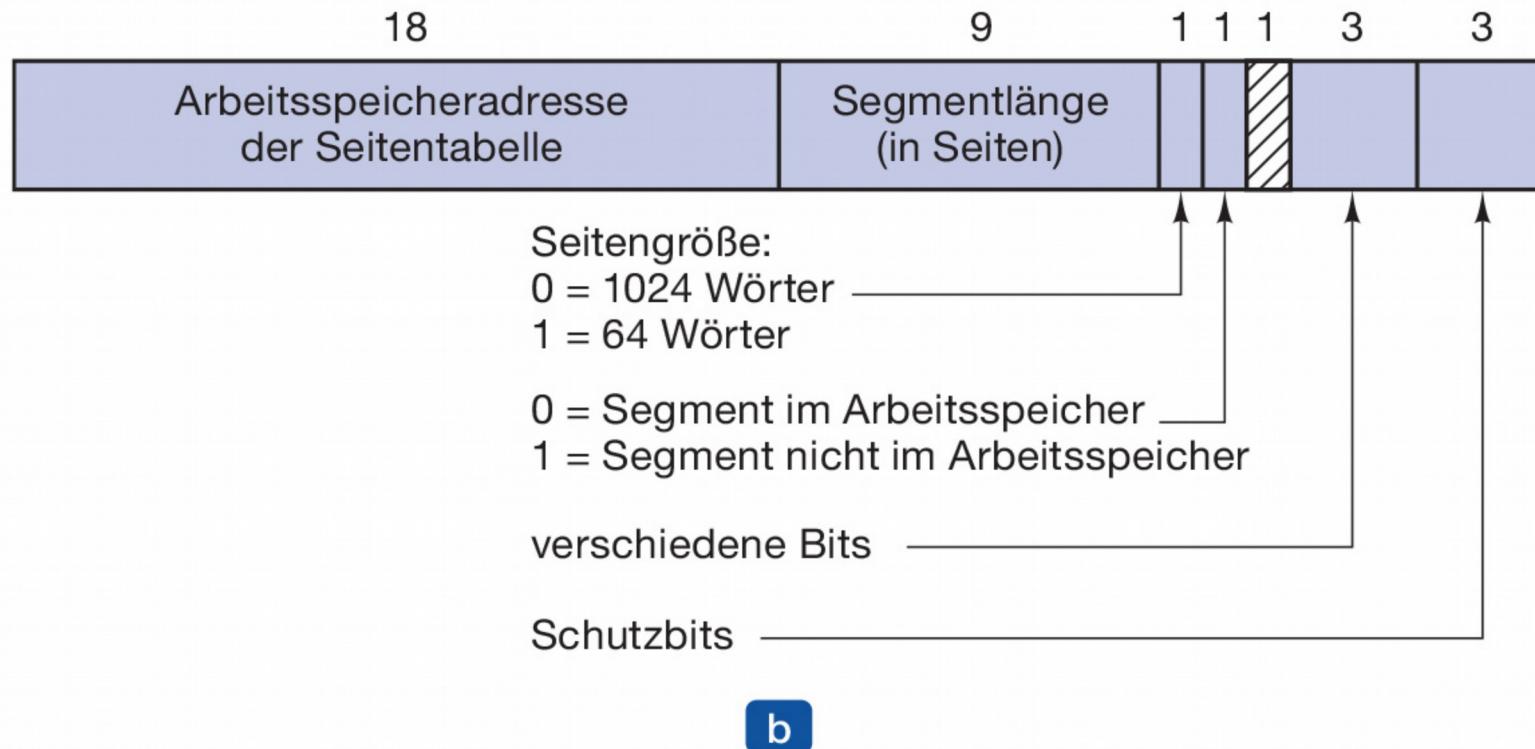
**Abbildung 3.33:** (a)–(d) Entstehung von externer Fragmentierung. (e) Behebung durch Speicherverdichtung.

# Segmentierung mit Paging: MULTICS (1)



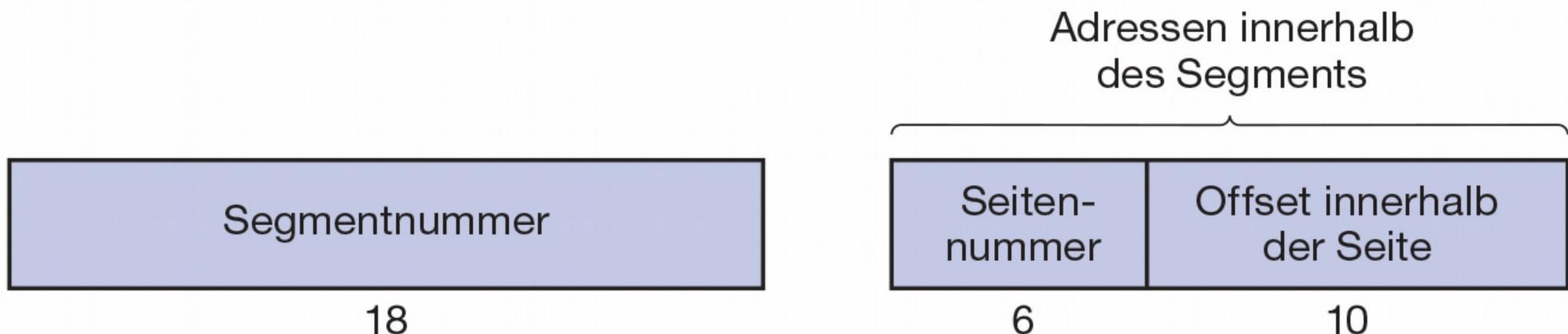
a

# Segmentierung mit Paging: MULTICS (2)



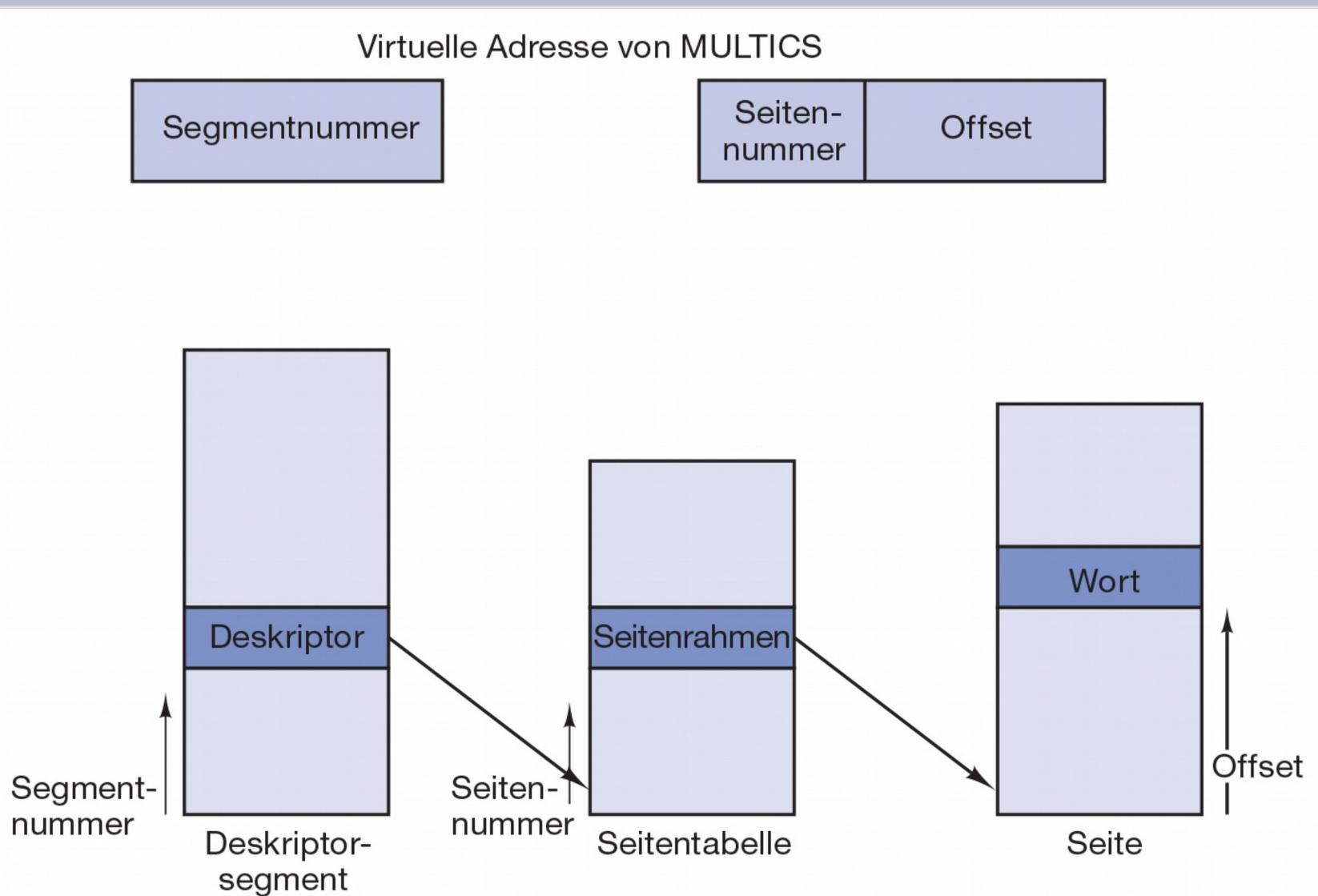
**Abbildung 3.34:** Der virtuelle Speicher unter MULTICS: (a) Die Deskriptoren im Deskriptorsegment zeigen auf die Seitentabellen. (b) Ein Segmentdeskriptor. Die Zahlen bezeichnen die Länge des Felds.

# Segmentierung mit Paging: MULTICS (3)



**Abbildung 3.35:** Eine virtuelle 34-Bit-Adresse unter MULTICS.

# Segmentierung mit Paging: MULTICS (4)



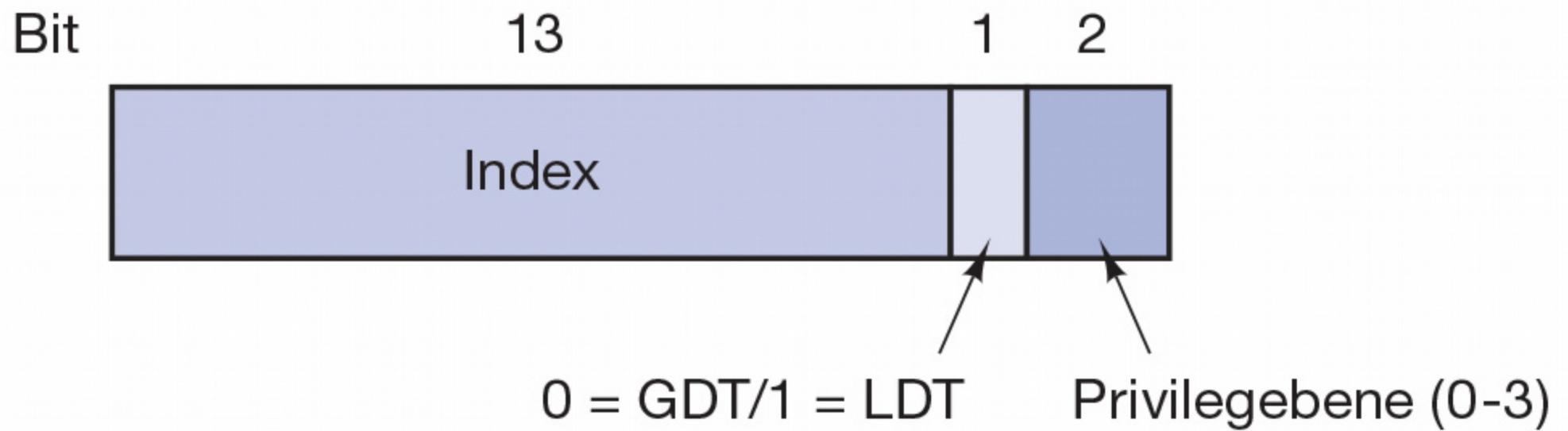
**Abbildung 3.36:** Umwandlung einer zweiteiligen MULTICS-Adresse in eine physische Speicheradresse.

# Segmentierung mit Paging: MULTICS (5)

Vergleichsfeld			Schutz	Wird dieser Eintrag benutzt?	
Segment-nummer	Virtuelle Seite	Seiten-rahmen		Alter	↓
4	1	7	Lesen/Schreiben	13	1
6	0	2	Nur Lesen	10	1
12	3	1	Lesen/Schreiben	2	1
					0
2	1	0	Nur Ausführen	7	1
2	2	12	Nur Ausführen	9	1

**Abbildung 3.37:** Eine vereinfachte Version des MULTICS-TLB. In Wirklichkeit war der TLB durch die zwei verschiedenen Seitengrößen komplizierter.

# Segmentierung mit Paging: Intel x86 (1)



**Abbildung 3.38:** x86-Selektor.

# Segmentierung mit Paging: Intel x86 (2)

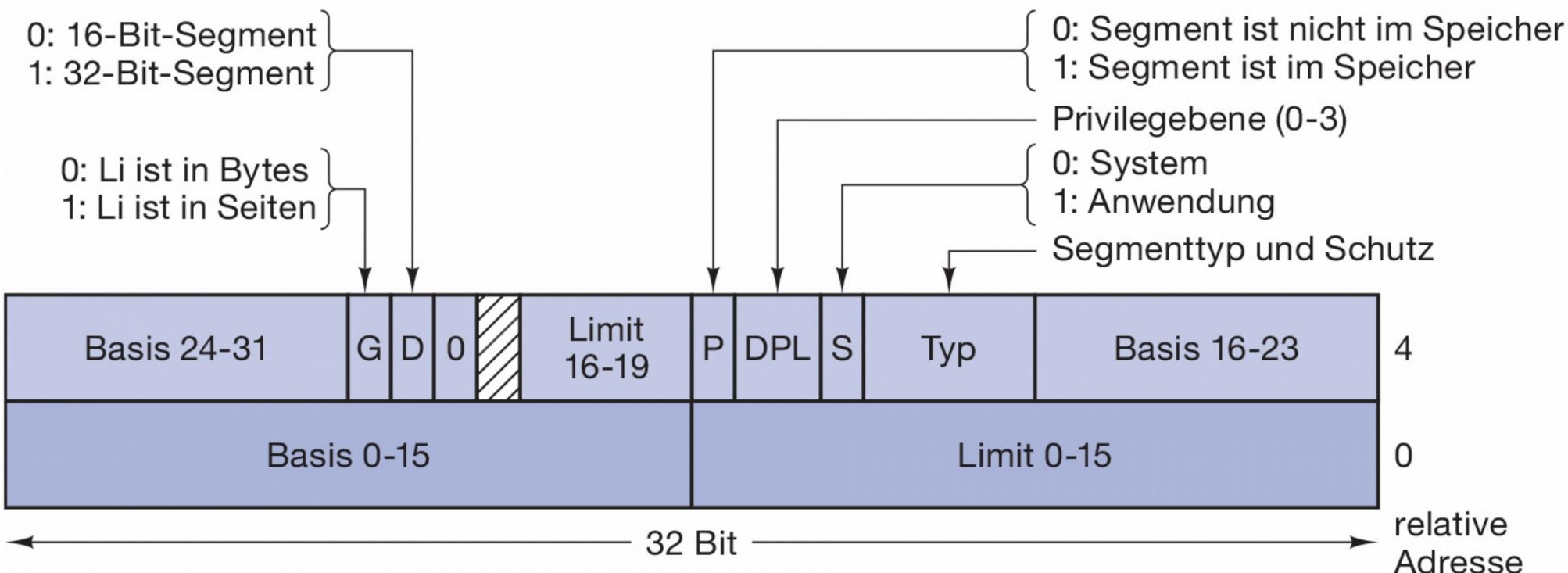
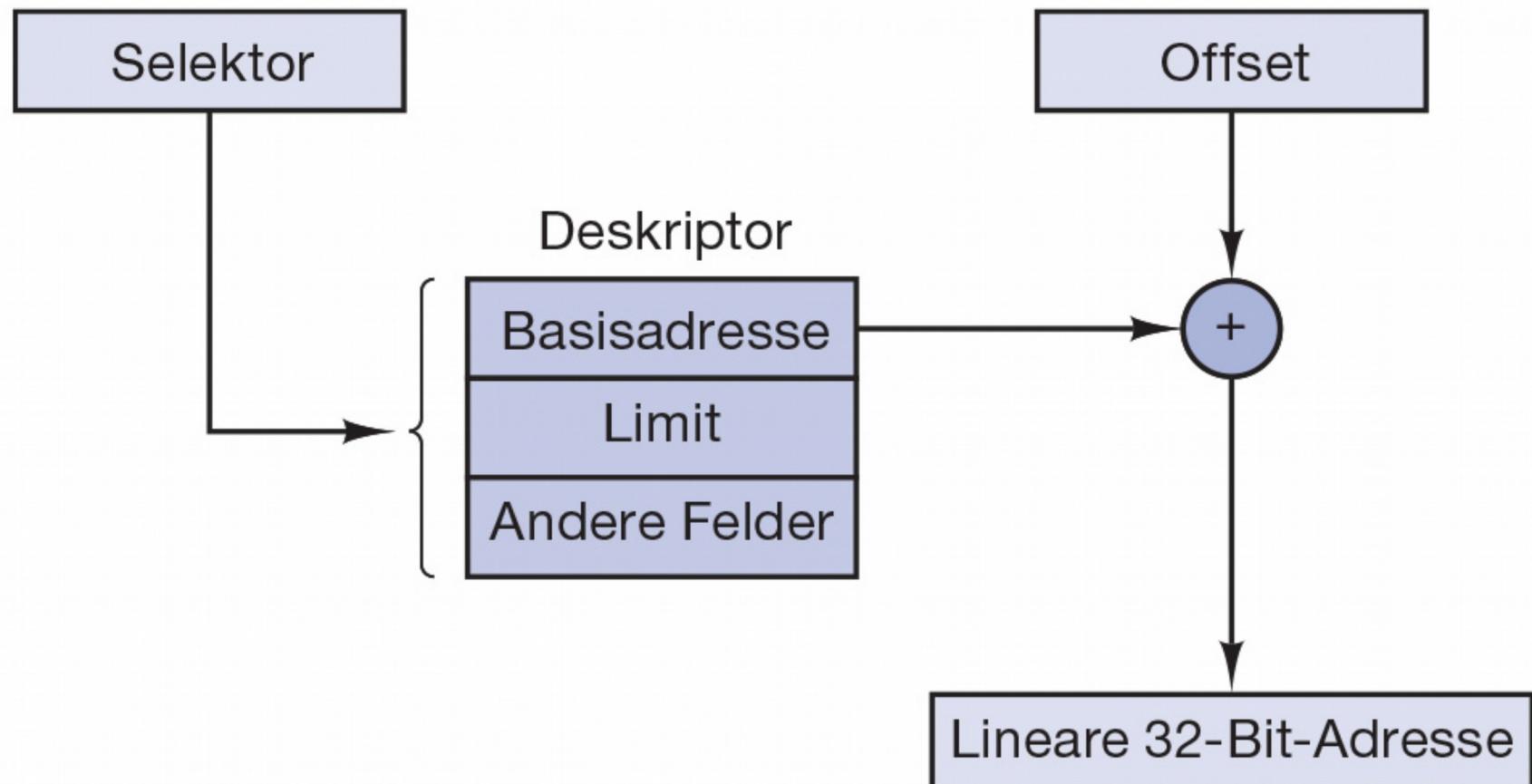


Abbildung 3.39: x86-Deskriptor für ein Codesegment. Datensegmente sehen etwas anders aus.

# Segmentierung mit Paging: Intel x86 (3)



**Abbildung 3.40:** Berechnung einer linearen Adresse aus einem (Selektor, Offset)-Paar.

# Segmentierung mit Paging: Intel x86 (4)

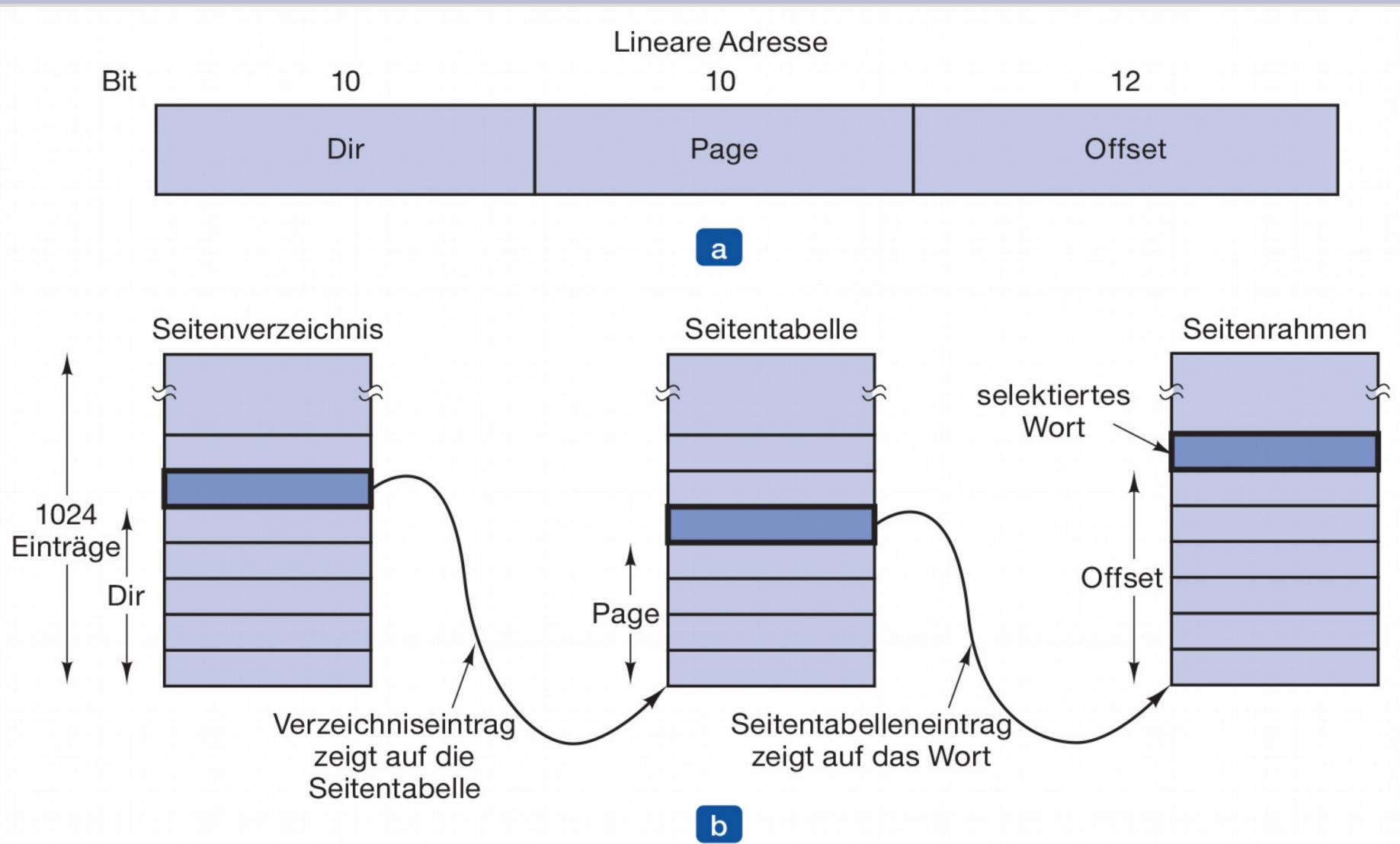


Abbildung 3.41: Abbildung einer linearen auf eine physische Adresse.