DUALE HOCHSCHULE SH

Speicherverwaltung

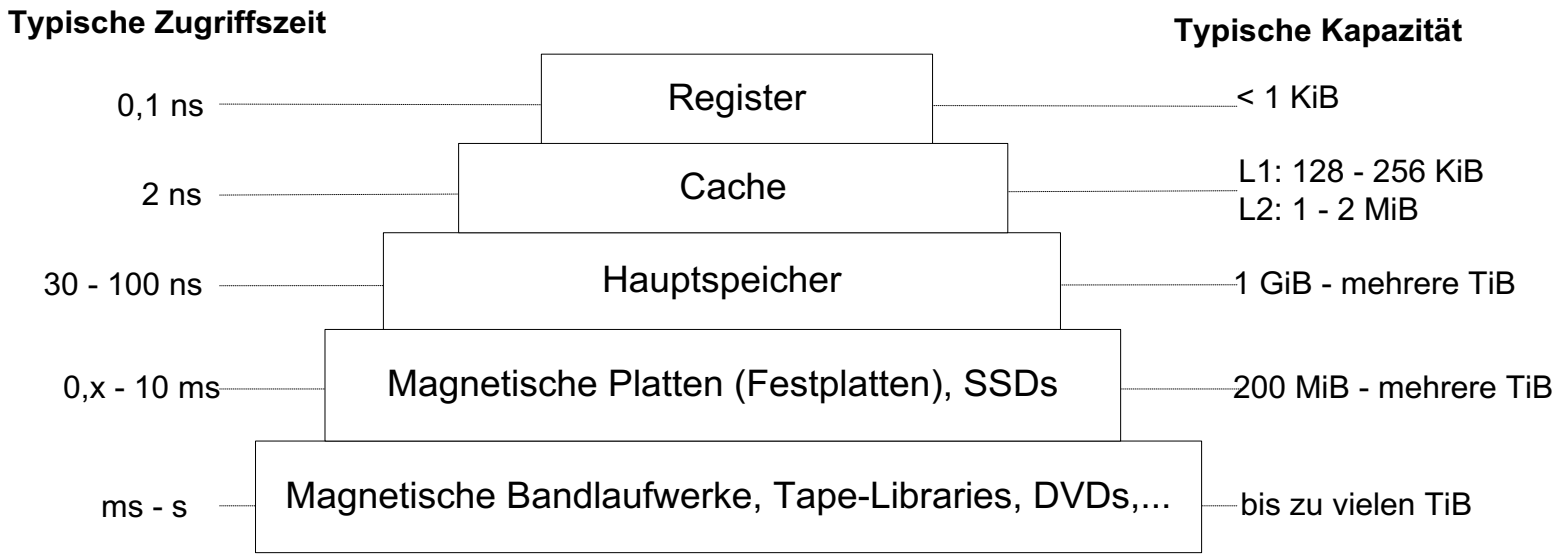
Computing-Plattformen und Netzwerke

auf Basis der Unterlagen von
Prof. P. Mandl (HS München)

Grundlagen der Speicherverwaltung

Speicherhierarchie moderner Rechnersysteme

- Je schneller der Speicher umso teurer ist er



- **Hinweis:** Zugriffszeit dient zur Positionierung, Übertragungsrate bestimmt die Zeit für die Übertragung der Daten
- **Beispiele für Übertragungsraten:**
  - Typische Übertragungsrate eines L3 Cache je Kern ~ 96 Gbyte/s
  - Übertragungsrate DDR4-3200-Speicherchips liegt bei 51,2 Gbyte/s
  - Typische Übertragungsrate von SSDs ~ 250 Mbyte/s

Hauptspeicherverwaltung im Betriebssystem

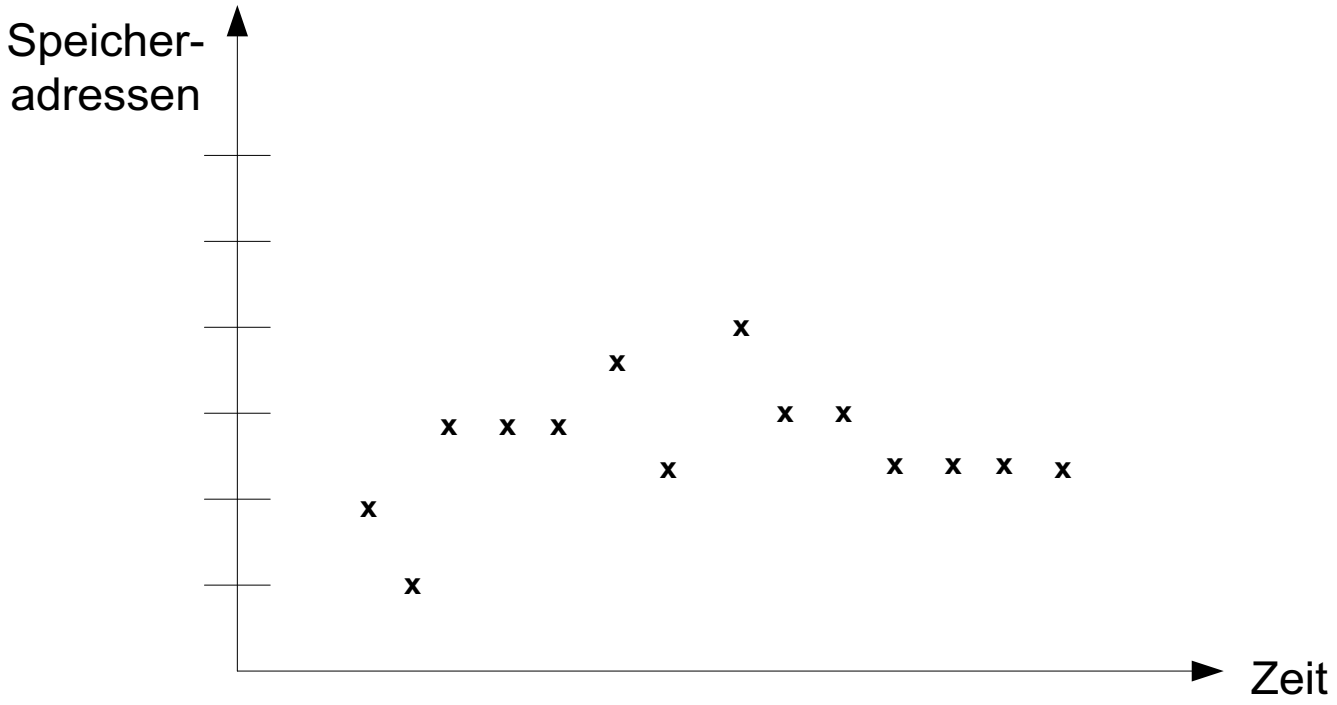
- Verantwortliche Kernel-Softwarekomponente
  - **Memory Manager (Speicherverwalter)**

- Aufgabe des Speicherverwalters
  - Versorgung der Prozesse mit den Betriebsmitteln „Arbeitsspeicher“ (Hauptspeicher)
  - Er verwaltet die freien und belegten Speicherbereiche des Kernels und aller laufenden Prozesse

Lokalitätsprinzip

- **Zeitlich**: Daten/Code-Bereiche, die gerade benutzt werden, werden mit hoher Wahrscheinlichkeit gleich wieder benötigt

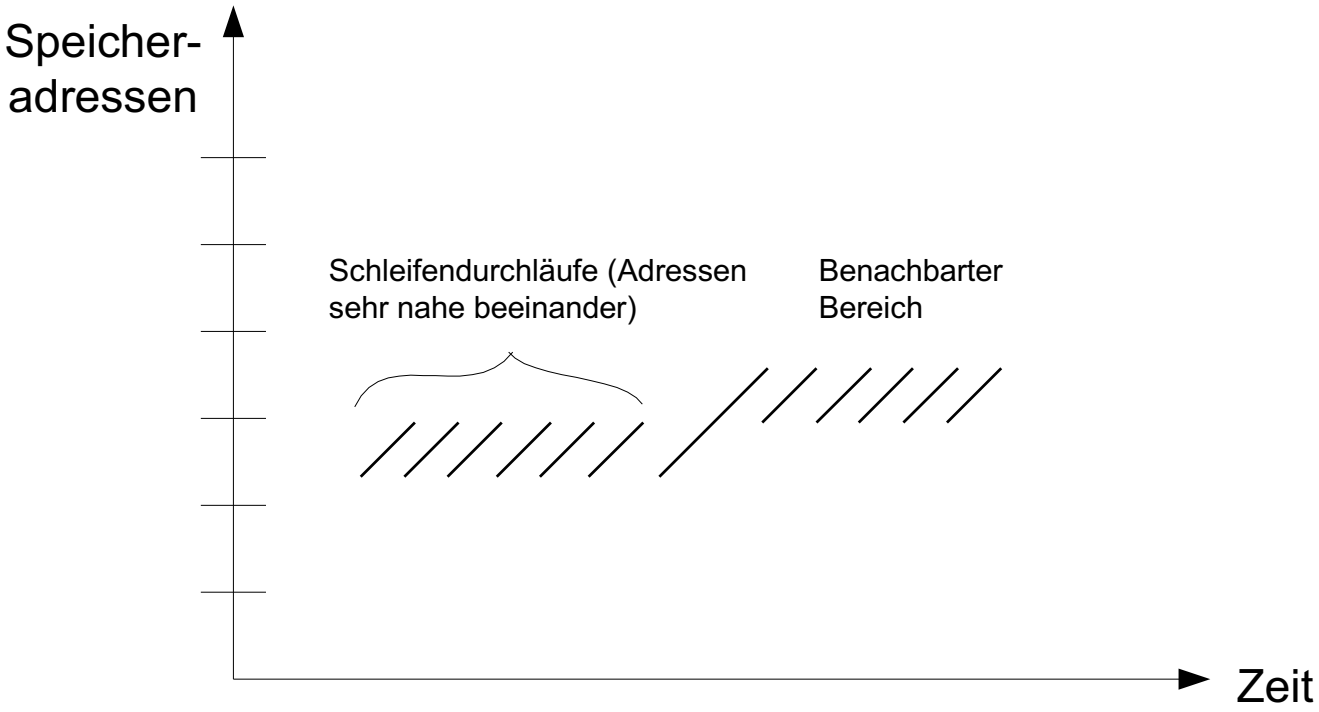
→ Diese sollten für den nächsten Zugriff bereitgehalten werden



Lokalitätsprinzip

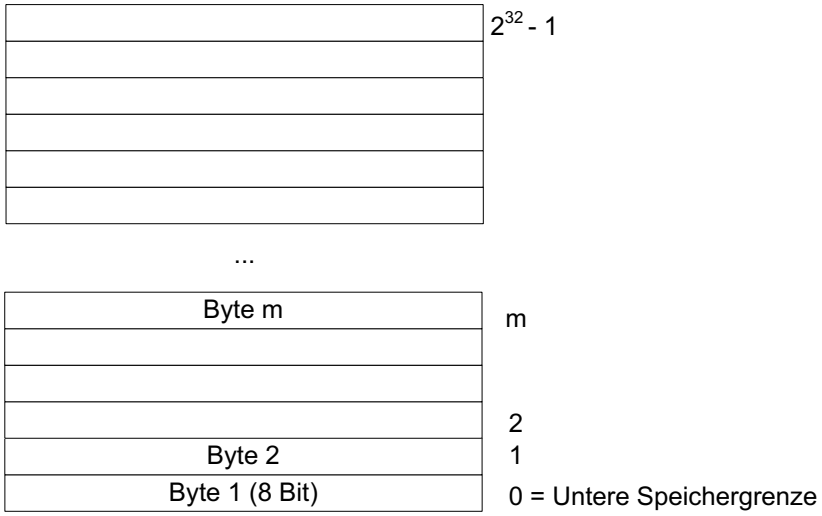
- **Örtlich**: Nächster Daten/Code-Zugriff ist mit hoher Wahrscheinlichkeit in der Nähe der vorherigen Zugriffe

→ Benachbarte Daten beim Zugriff auch gleich in schnelleren Speicher laden



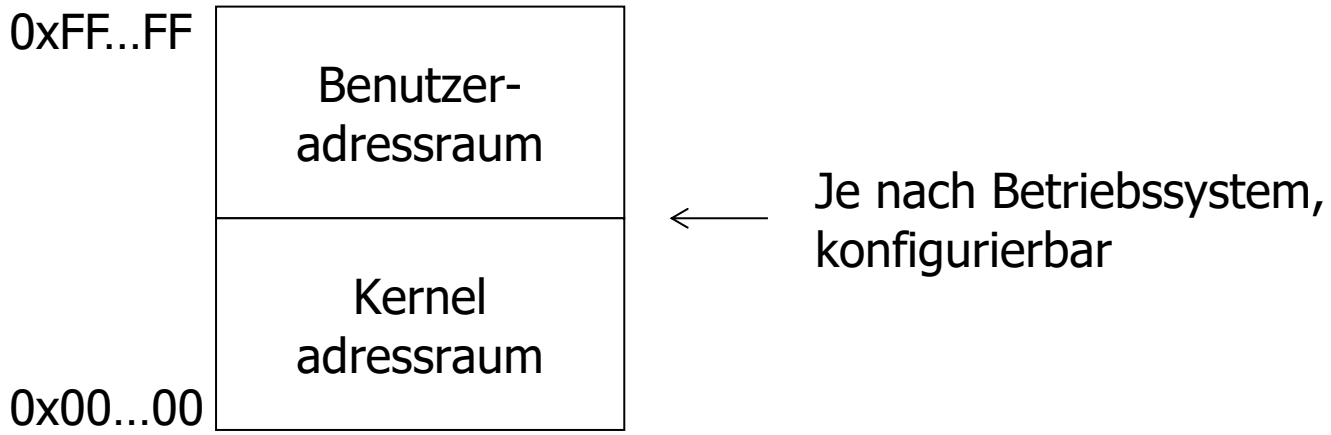
Adressen und Adressräume

- Hauptspeicher ist in logisch adressierbare **Speicherstellen** unterteilt, meist byteweise (8 Bit)
- Ein Byte ist also die kleinste adressierbare Einheit
- 32-Bit-Adressen → 2^{32} adressierbare Bytes
- Ein **Adressraum** ist die Menge aller adressierbaren Adressen
  - 32-Bit-Adressen → {0, 1, 2, ... , 2^{32} - 1}



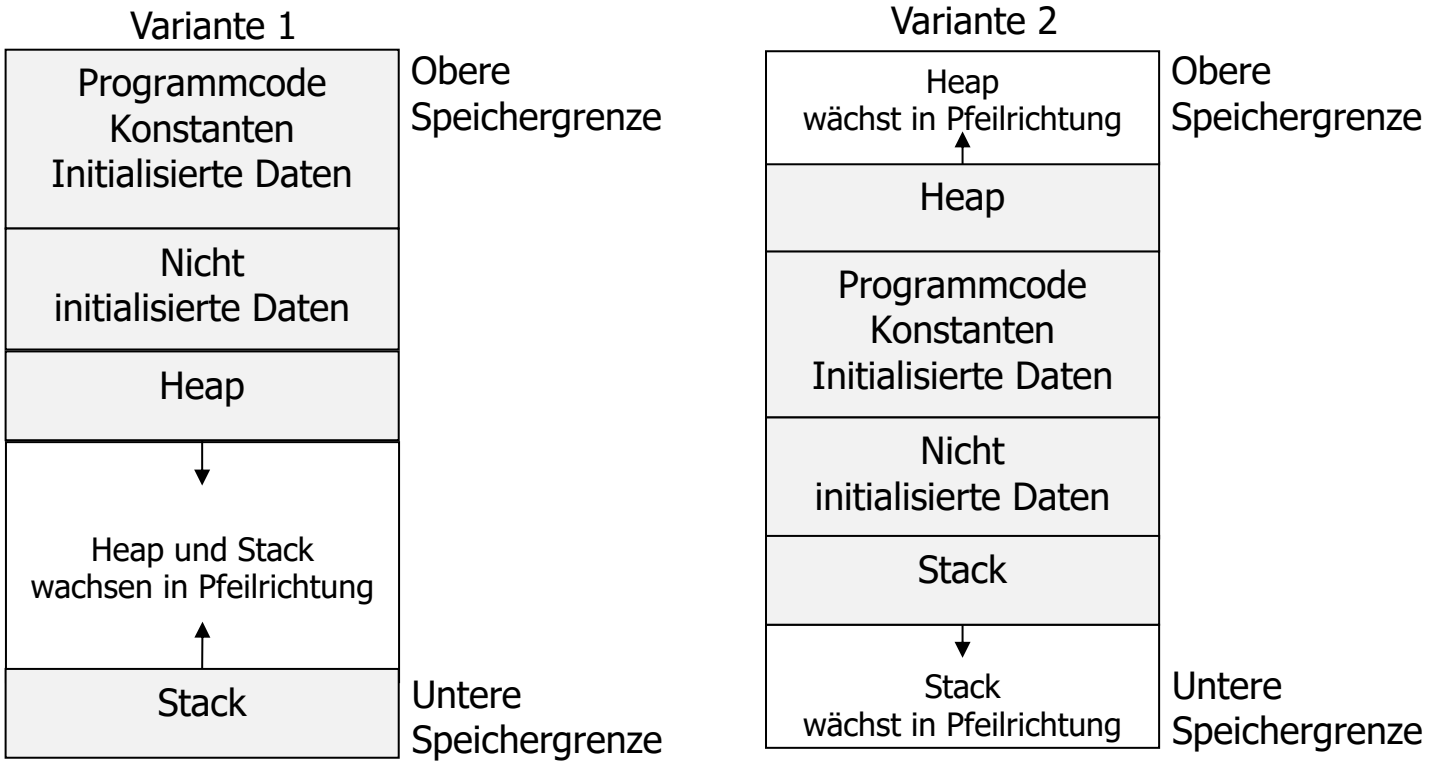
Adressraumbelegung

- Wird durch Adressraumbelegungsplan bestimmt
- Festlegung im Betriebssystem
- Ausrichtung auf Maschinenwörter wichtig wegen optimalem Zugriff



Adressraumbelegung für Benutzeradressraum

- Adressbereich der Anwendungsprogramme organisiert der Compiler/Interpreter bzw. das dazugehörige Laufzeitsystem
- Varianten sind abhängig von der Programmiersprache:

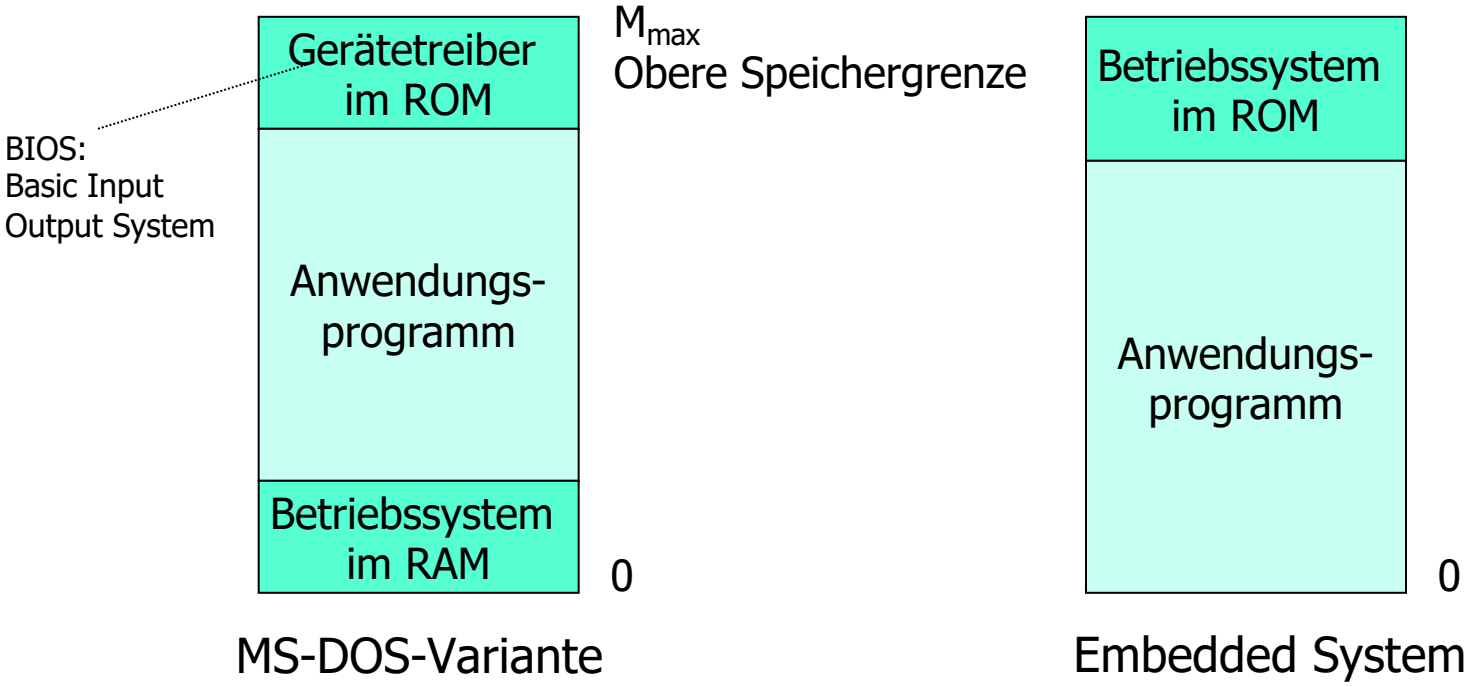


Verschiedene Mechanismen der Speicherverwaltung

- Es gibt verschiedene Mechanismen für die Speicherverwaltung
- Historische Entwicklung:
  1. Speicherverwaltung bei Monoprogramming (nur ein Programm im Speicher)
  2. Speicherverwaltung mit festen Partitionen (mehrere Programme gleichzeitig im Speicher)
  3. Swapping (mehrere Programme gleichzeitig im Speicher)
  4. Virtueller Speicher (mehrere Programme gleichzeitig im Speicher)

Speicherverwaltung bei Monoprogramming

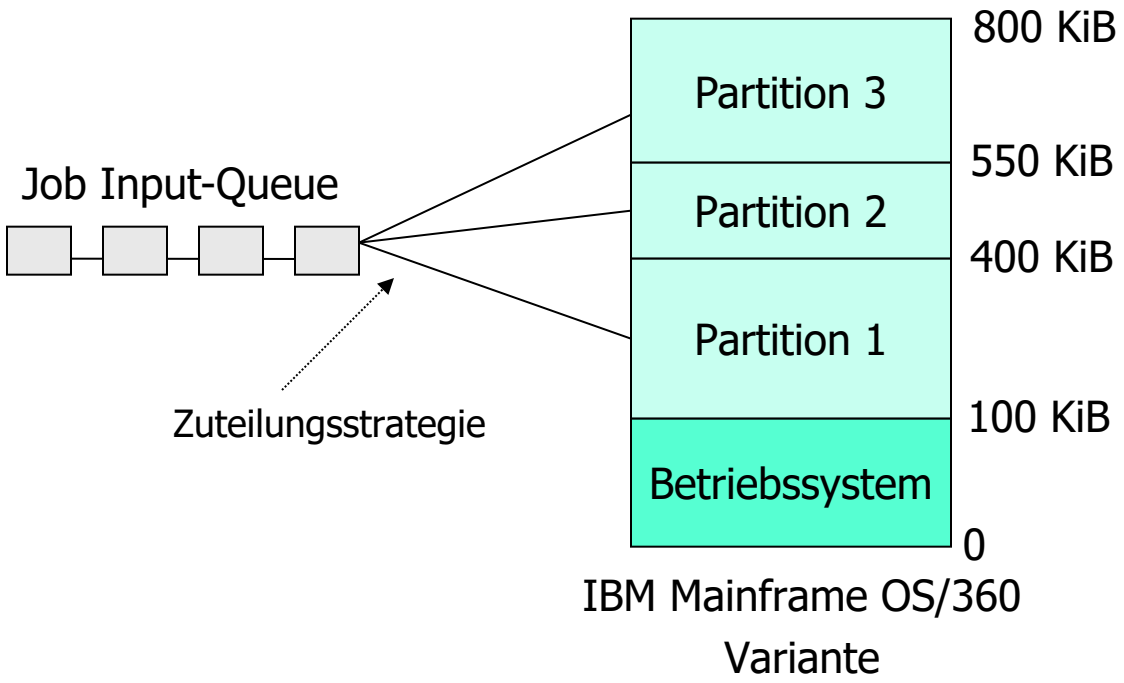
- Einfachste Form der Speicherverwaltung
- Nur ein Programm läuft zu einer Zeit



- **BIOS** = Programm zum Starten eines Rechnersystems, bis das Betriebssystem übernimmt. Es liegt in einem nicht flüchtigen ROM oder in einem Flashspeicher
- Weiterentwicklung von BIOS: **EFI** = Extensible Firmware Interface, unterstützt auch 64-Bit-Systeme

Speicherverwaltung mit festen Partitionen

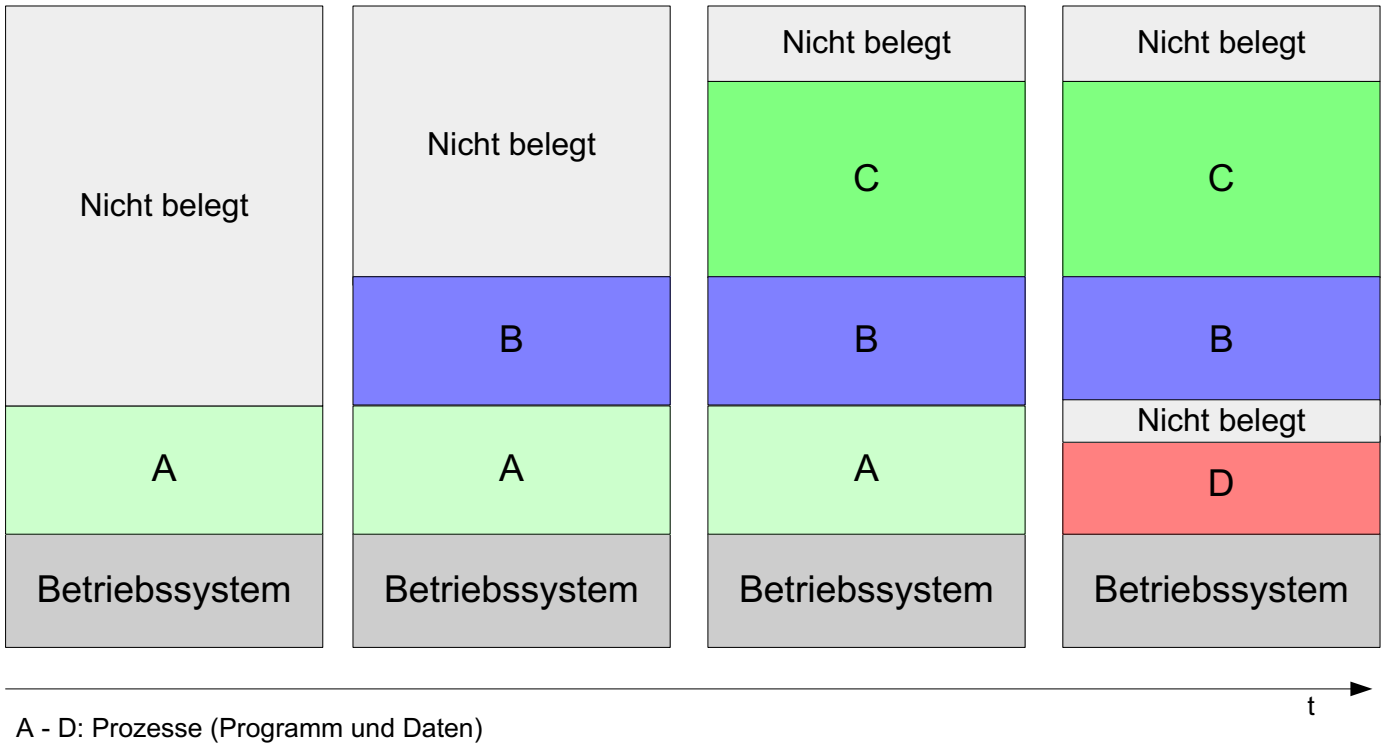
- Aufteilung des Speichers in **feste Teile**, sog. Partitionen, eine für ein Programm (Multiprogramming)
- Ankommender Job wird in eine Queue eingetragen
  - Für jede Partition eine Queue oder eine globale Queue



Swapping (1)

- **Grundgedanke:** Timesharing!
  - Es passen nicht immer alle Prozesse in den Hauptspeicher
  - Prozess wird **komplett** in Speicher geladen
  - Prozess wird nach einer gewissen Zeit wieder auf einen Sekundärspeicher (Platte) ausgelagert
  - Entstehende Löcher können durch Kombination benachbarter Speicherbereiche eliminiert werden, aber aufwändig!
- **Hauptunterschied** Swapping - feste Partitionen:
  - Anzahl, Speicherplatz und Größe des für einen Prozess verwendeten Speicherbereichs variieren dynamisch
  - Prozess wird immer dahin geladen, wo gerade ausreichend Platz ist

Swapping (2)



Vgl.: *Tanenbaum, A. S.; Bos, H. Modern Operating Systems, Forth Edition, 2015*

Virtueller Speicher

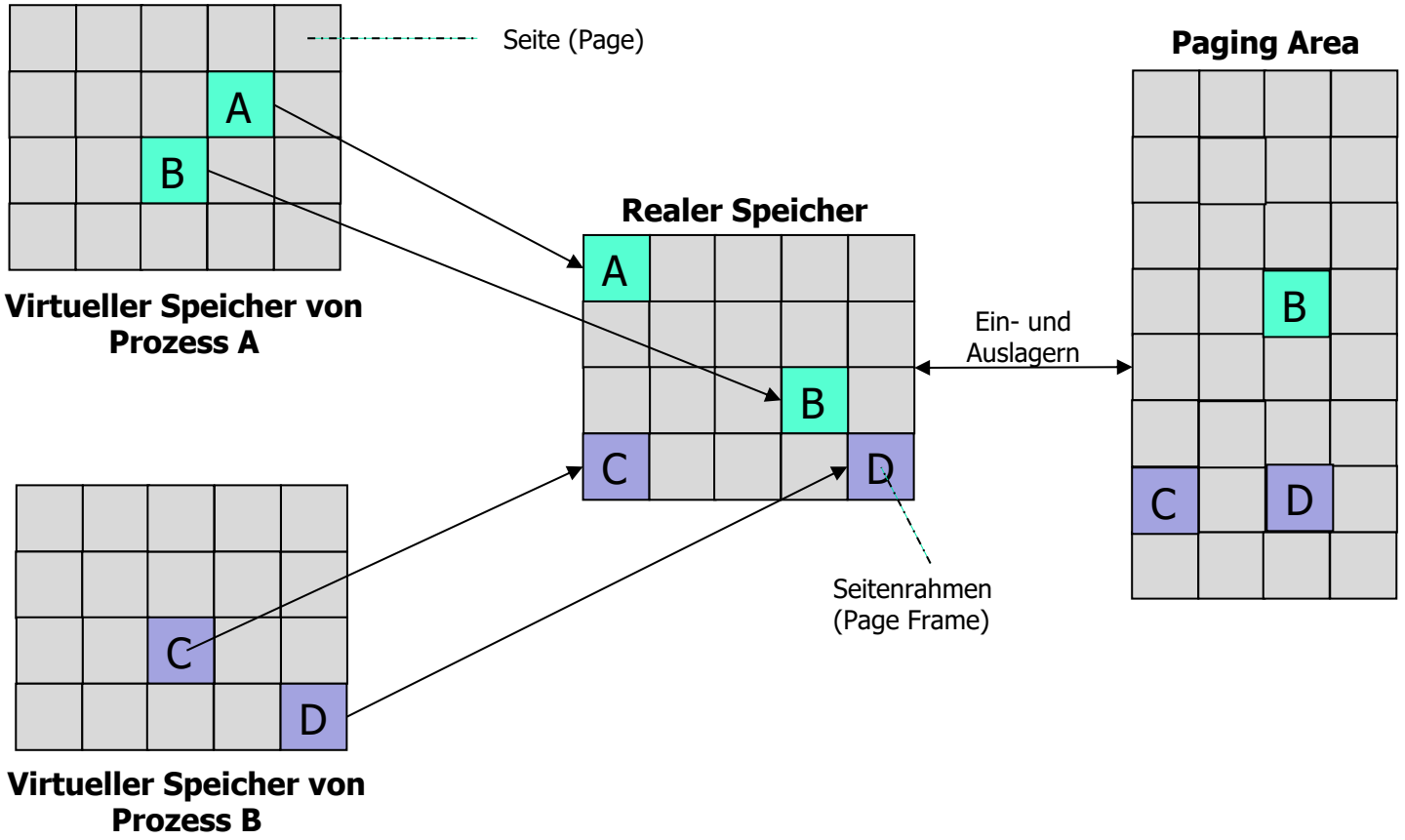
Grundlegende Überlegungen zur Virtualisierung

- Grundlegende Ideen zum virtuellen Speicher
  - Speichergröße eines Programms inkl. Daten und Stack darf den vorhandenen **physikalischen Hauptspeicher überschreiten**
  - Prozess kann auch ablaufen, wenn er **nur teilweise im Hauptspeicher** ist
  - Programmierer soll sich am besten nur mit einem **linearen Adressraum** befassen müssen
- Umsetzung:
  - Das Betriebssystem **gaukelt** jedem Prozess einen vollständigen Adressraum **vor**, so als gehöre jedem Prozess der ganze Hauptspeicher
  - Das Betriebssystem hält die gerade benutzten Teile im Hauptspeicher und den Rest auf einer Festplatte

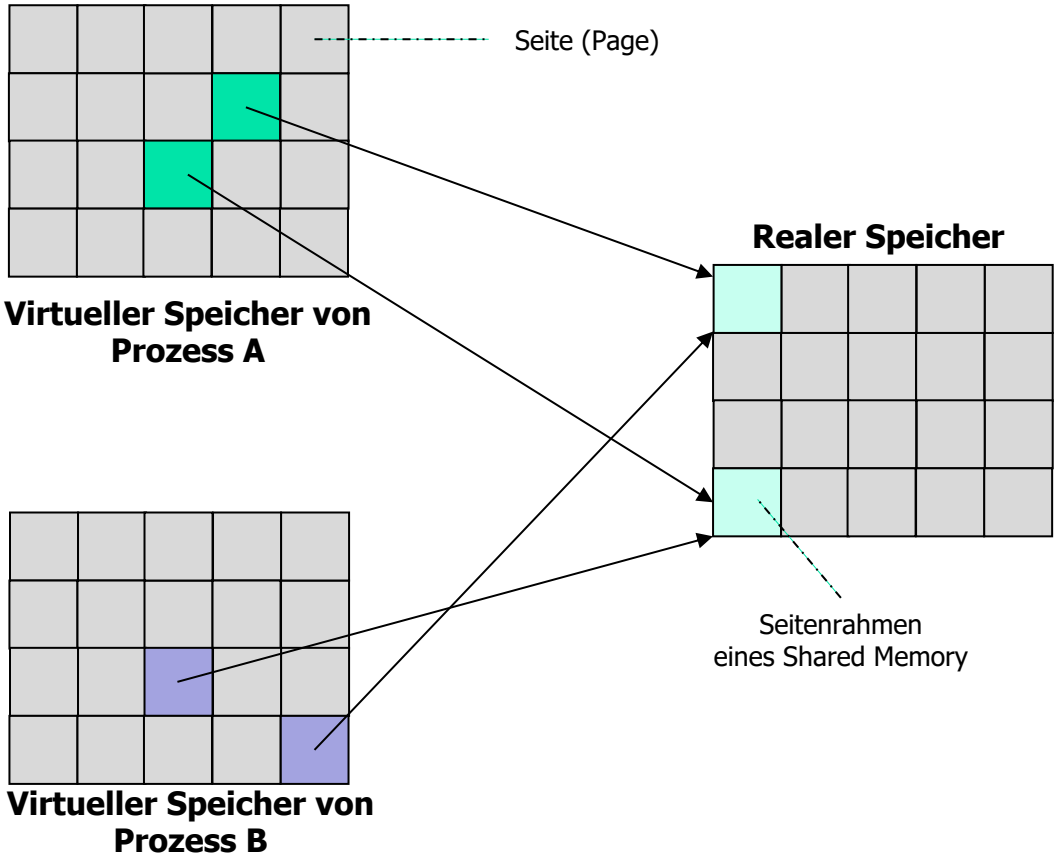
Grundprinzip und Grundbegriffe (1)

- Virtueller Adressraum
- Realer Adressraum
- Seiten (Pages)
- Seitenrahmen (Frames bzw. Page Frames)
- Paging und Paging Area (Schattenspeicher)
  - Für den Hauptspeicher wird ein Schattenspeicher in einem speziellen Plattenbereich reserviert (**Paging Area**)

Grundprinzip und Grundbegriffe (2)
Virtueller, realer Adressraum (Speicher), Paging Area



Einschub: Shared Memory



Grundprinzip und Grundbegriffe (3)
Paging

- Die Umlagerung zwischen Hauptspeicher und Platte wird als **Paging** bezeichnet
- Jeder Prozess darf alle Adressen verwenden, die aufgrund der HW-Architektur des Rechners möglich sind
  - unabhängig von der realen Größe des Hauptspeichers
- Bei Systemen mit 32-Bit-Adressen kann jeder Prozess einen Adressraum von 4 GiB verwenden
  - Dies **gilt auch** wenn der Hauptspeicher z.B. nur einige MiB realen Speicher hat
  - Dies hat aber seine Grenzen, wenn das System nicht ausschließlich mit **Paging** beschäftigt sein soll!

Grundprinzip und Grundbegriffe (4)
Pages und Frames

- Das Speicherabbild eines Prozesses besteht aus Speicherseiten
- Eine Speicherseite ist ein Segment einer vorgegebenen Größe (z.B. 4 KiB)
- Nur die wirklich benötigten Speicherseiten müssen im Arbeitsspeicher geladen sein, während der Prozess läuft
- Die Größe der Seiten ist meist gering, die Anzahl sehr groß
- Virtuelle Seiten (Pages) werden in reale Rahmen (Frames) übertragen

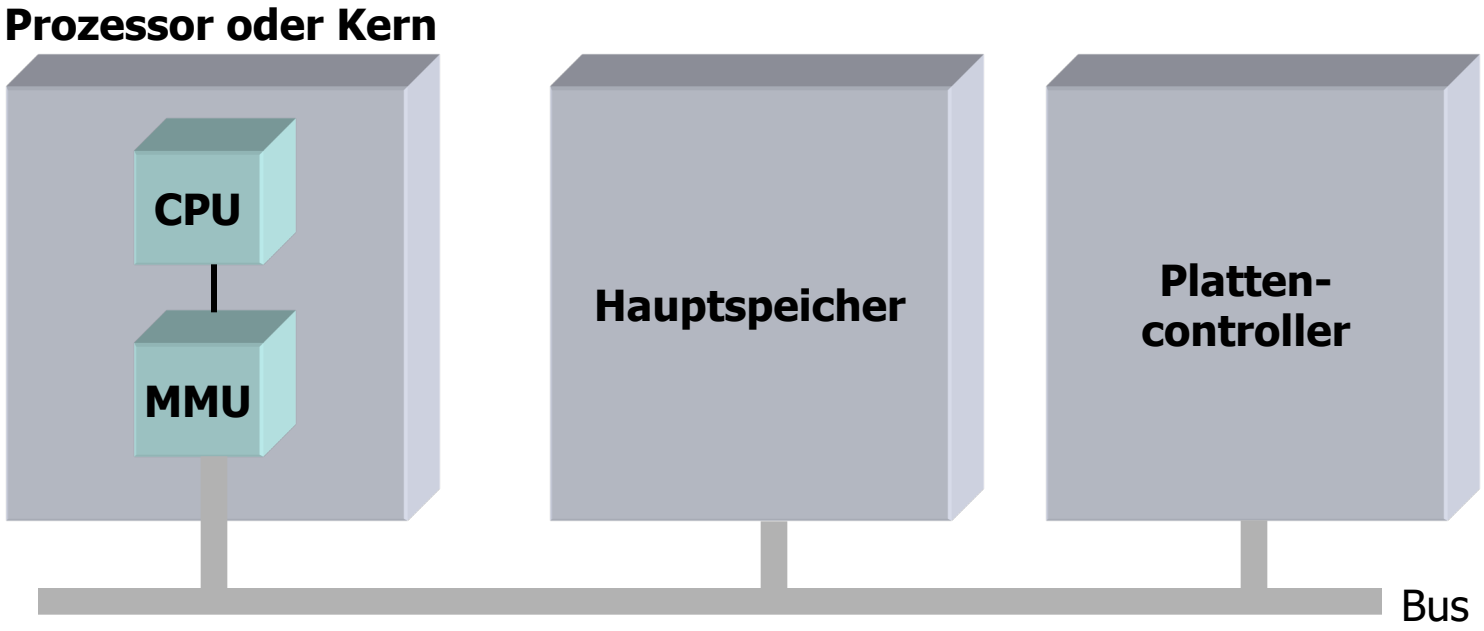
Grundprinzip und Grundbegriffe (5): Noch zu klärende Fragen

- Wie funktioniert das Mapping von virtueller Adresse auf eine reale Adresse?
- Wie wird der virtuelle Speicher verwaltet?
- Was macht die Hardware, was macht die Software?
- Wie groß sind Pages und Frames in realen Systemen?
- Was ist, wenn der Hauptspeicher voll ist, aber ein Prozess noch Speicher anfordert?
  - Seitenersetzung, Verdrängung

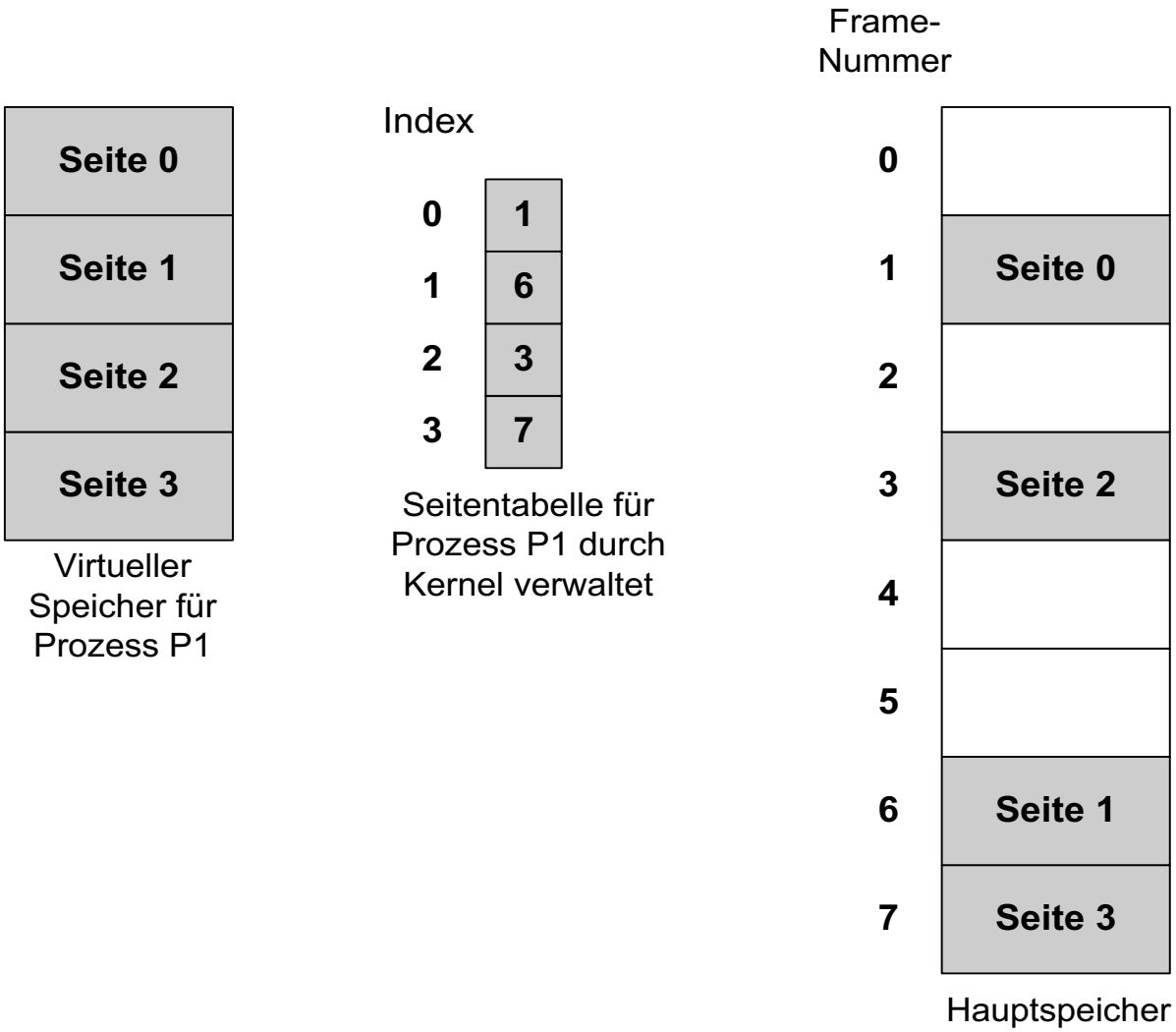
Speicherverwaltungsstrategien

Adressumsetzung: Hardwareunterstützung durch die MMU

- MMU = Memory Management Unit (Hardware)
- CPU sendet virtuelle Adressen an die MMU
- MMU sendet reale Adressen an den Hauptspeicher

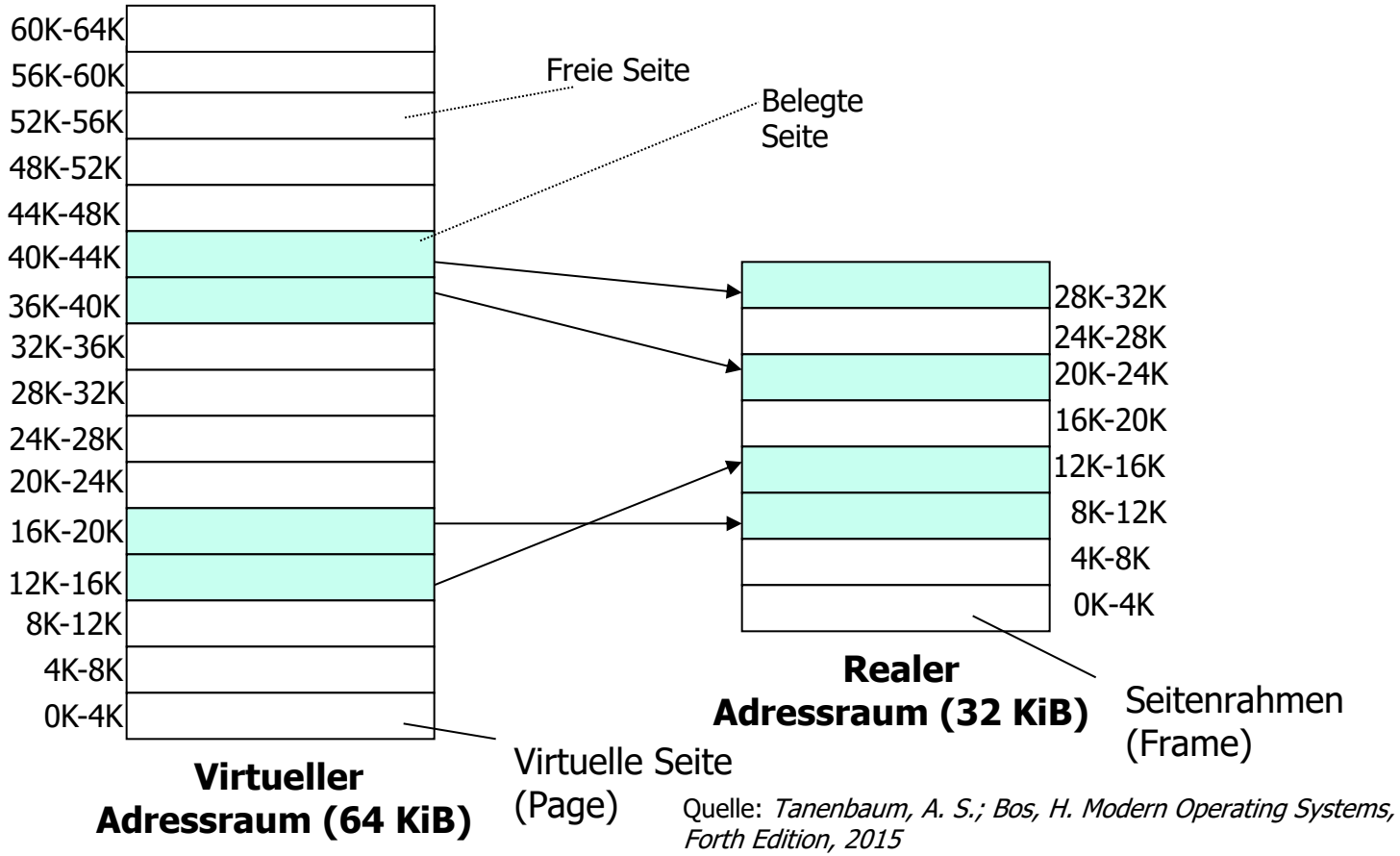


Adressumsetzung (1): Seitentabellen, vereinfachtes Modell



Adressumsetzung (2): Mapping am Beispiel eines kleinen Adressraums

- Virtueller Adressraum → Realer Adressraum
- Hier ein Beispiel eines Adressraums (K = Kibibyte = 1024 Byte, Binärpräfix-Notation)

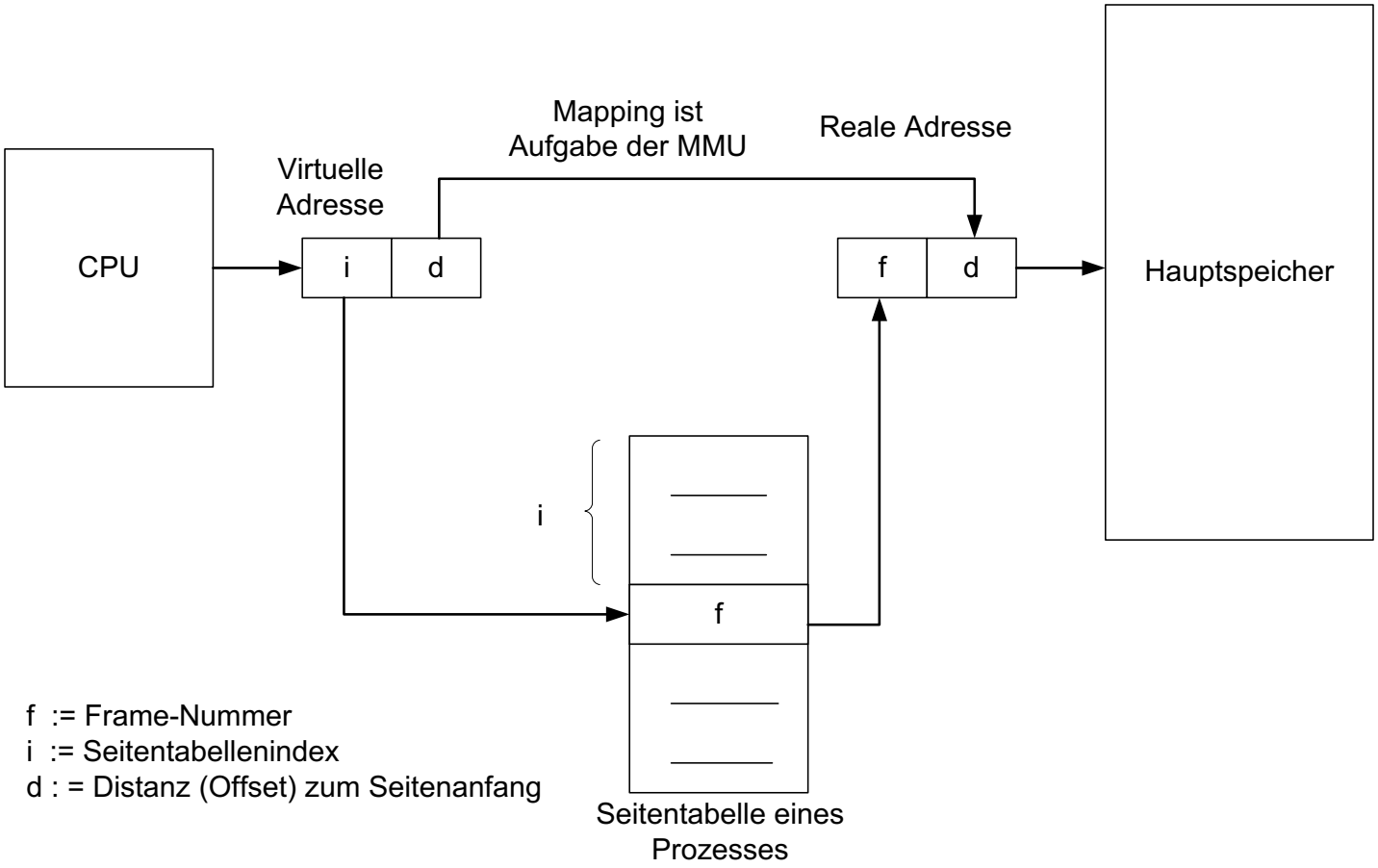


Quelle: *Tanenbaum, A. S.; Bos, H. Modern Operating Systems, Forth Edition, 2015*

Adressumsetzung (3)

- Die virtuelle Adresse wird in die virtuelle Seitennummer und einen Offset (Distanz) geteilt
- Die virtuelle Seitennummer ist ein Index auf die Seitentabelle
- Über diesen Index wird der zugehörige Eintrag in der Seitentabelle gefunden
- Im Eintrag steht die Frame-Nummer, falls die Seite einem Frame zugeordnet ist
- Also: f(Seitennummer) → Frame-Nummer
  - Falls Seite im Speicher
  - Sonst: page fault

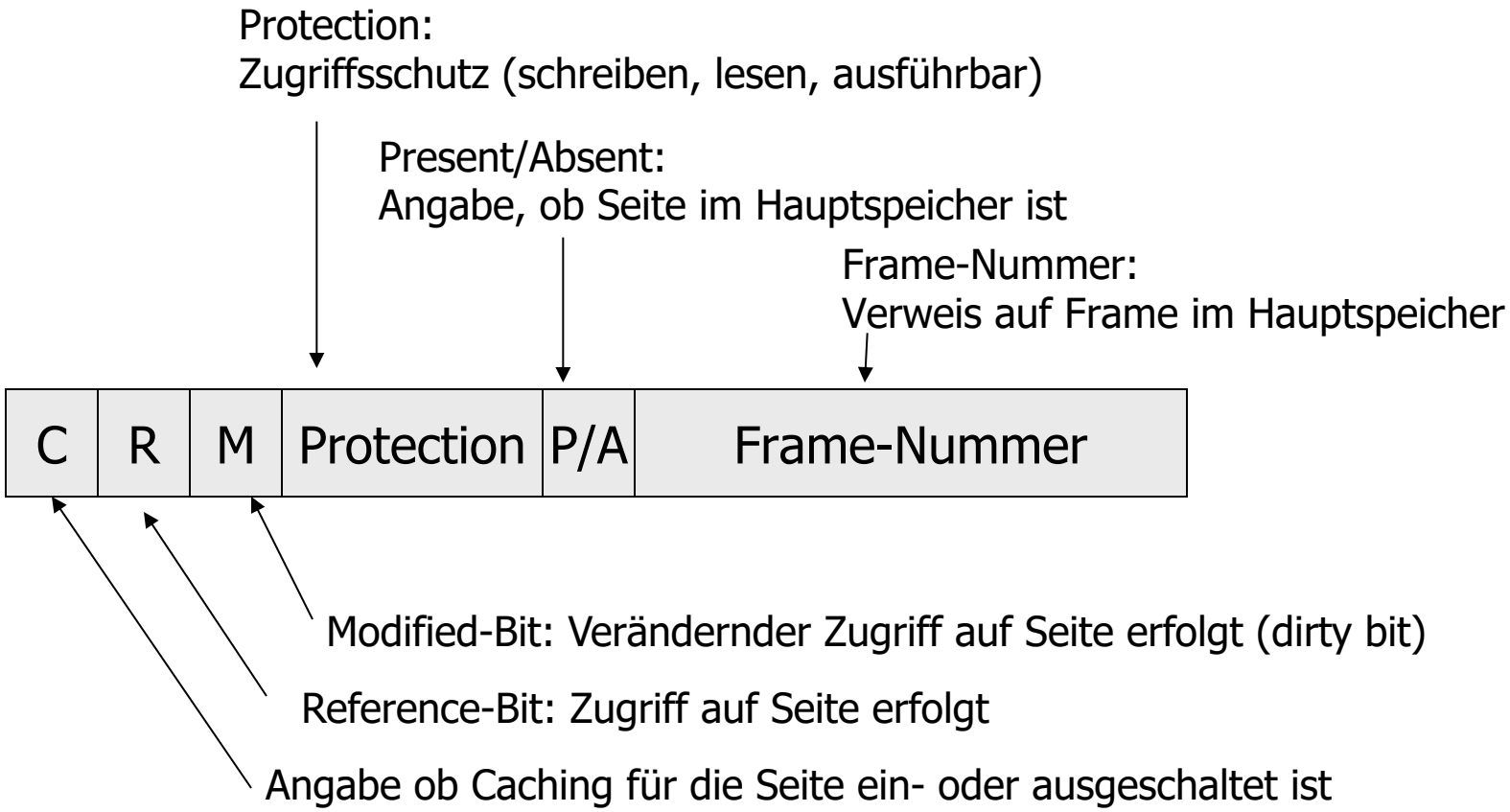
Adressumsetzung (4)
Einstufige Adressumsetzung



Register CR3 bei Intel enthält Page Directory Adresse = Adresse der Seitentabelle

Seitentabelleneintrag

- Der Aufbau eines Eintrags in der Seitentabelle hängt stark vom System ab, hier ein Beispiel:



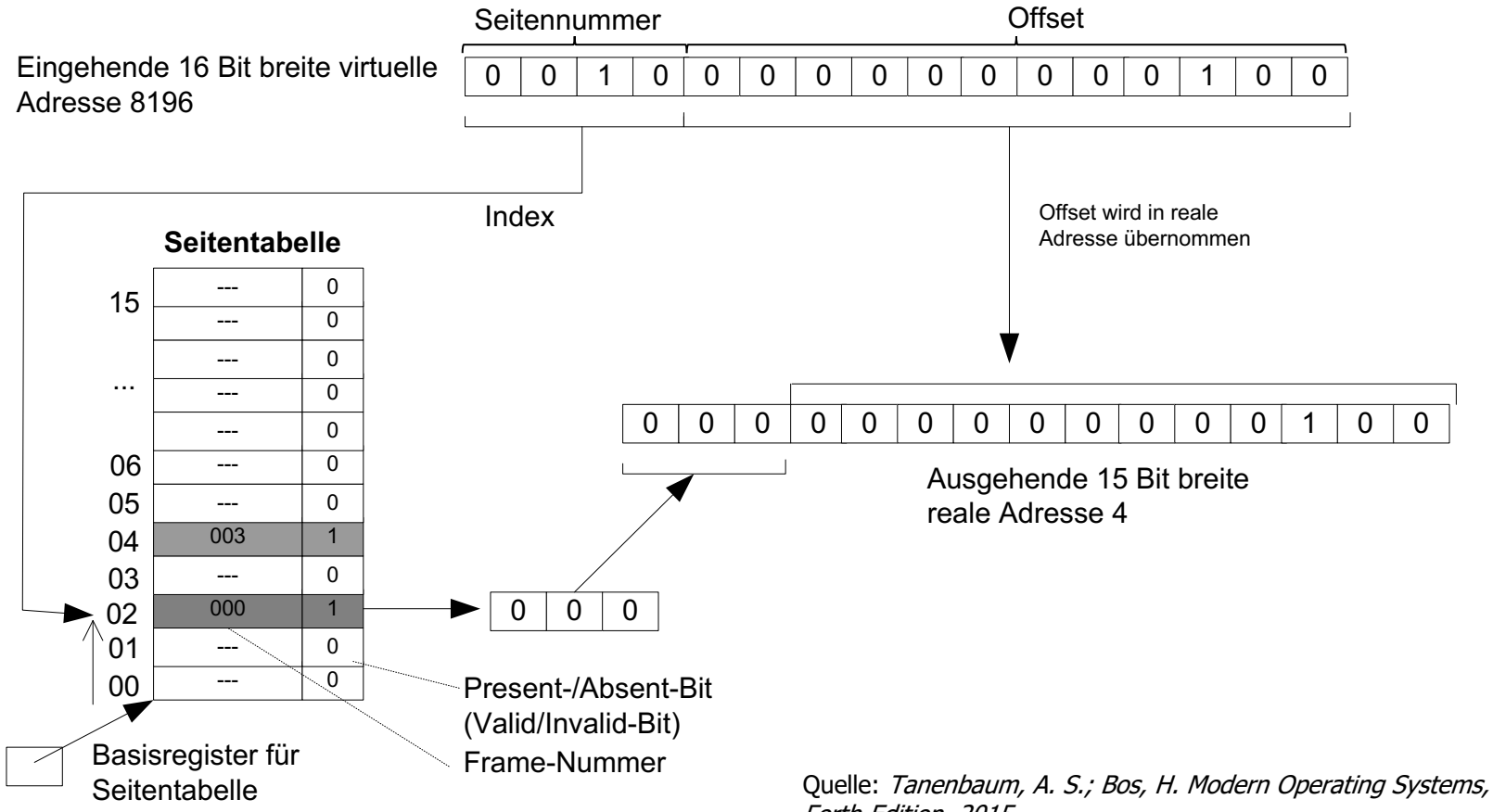
Beispiel einer Adressumsetzung (1)

- Beispieladressierung der Adresse 8196 über folgenden Befehl, der ausgeführt werden soll
  - **MOVE R1, 8196** mit R1 = CPU-Register
    - Adresse 8196 wird an die MMU gesendet
  - **MMU transformiert die** virtuelle Adresse 8196 in unserem Beispiel in die **reale Adresse 4**
    - Die Adresse 8196 befindet sich in Page 2 des virtuellen Adressraums
    - Adressiert wird damit im Hauptspeicher die Adresse 4 im untersten Frame
- Angesprochene Adresse befindet sich zur Befehlsausführung nicht im Hauptspeicher:
  - MMU verursacht bei der CPU einen **Trap** in das Betriebssystem (**page fault** genannt)
  - Seitenersetzungsstrategie notwendig

Beispiel einer Adressumsetzung (2)

MOVE R1, 8196

- Im Beispiel gehen wir von virtuellen 16-Bit-Adressen aus, in der die 4 höherwertigen Bits als Seitennummer der angeforderten Seite interpretiert werden



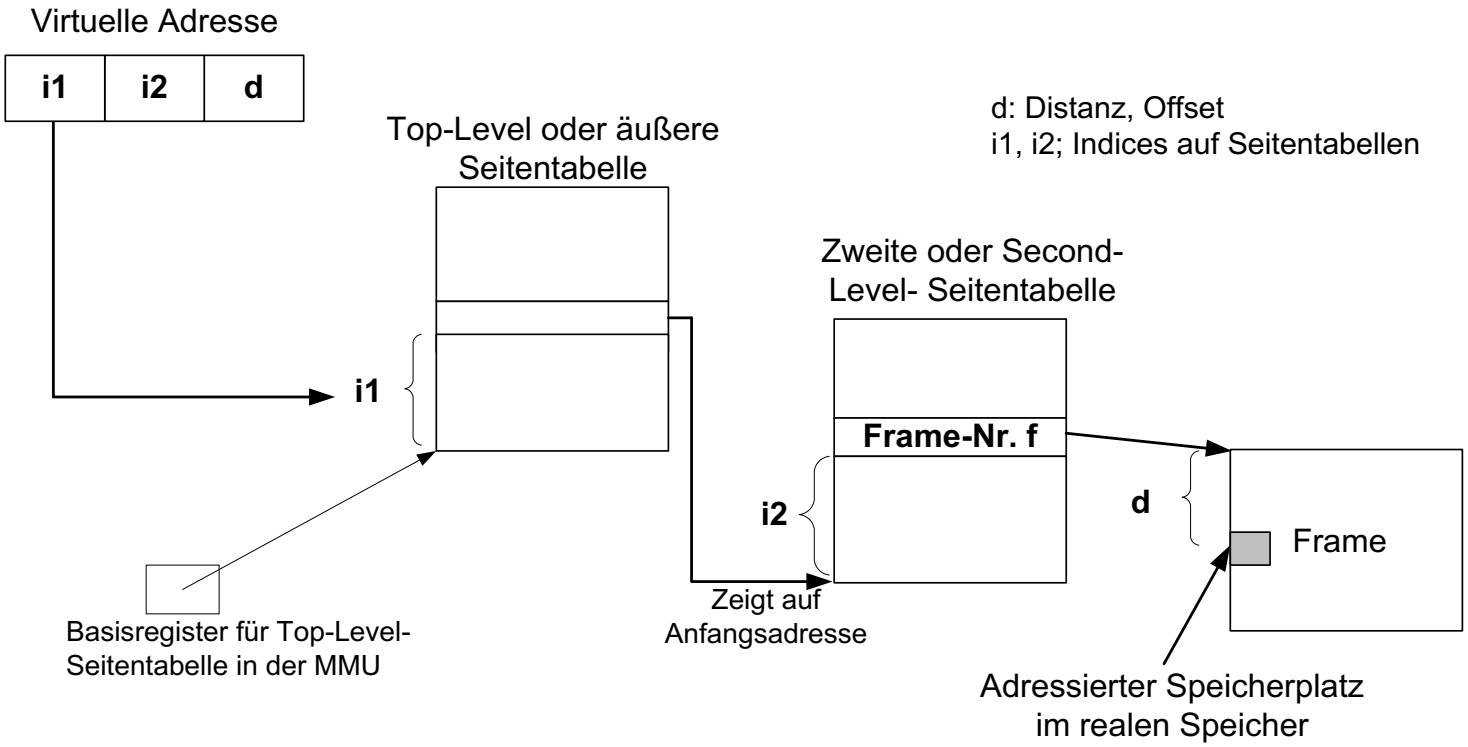
Quelle: *Tanenbaum, A. S.; Bos, H. Modern Operating Systems, Forth Edition, 2015*

Mehrstufige Adressumsetzung (1)

- Das **Mapping** muss **schnell** sein
- In großen Adressräumen sind sehr große Seitentabellen möglich
  - Z. B. bei einem 32 Bit Adressraum → über 1 Million Einträge in der Seitentabelle bei Seitengröße von 4 KiB
  - Bei 4 Byte pro Eintrag → 4 MiB Hauptspeicher notwendig
- **Jeder** Prozess benötigt **eigene Seitentabelle**
- Speicherersparnis durch **mehrstufige Seitentabellen** (zweistufige, dreistufig,...)
  - Damit wird erreicht, dass nicht immer alle Seitentabellen im Speicher gehalten werden müssen

Mehrstufige Adressumsetzung (2)

- Zweistufige Seitentabelle für 32-Bit-Adressen
  - Beispiel: i1 = 10 Bit, i2 = 10 Bit, d = 12 Bit



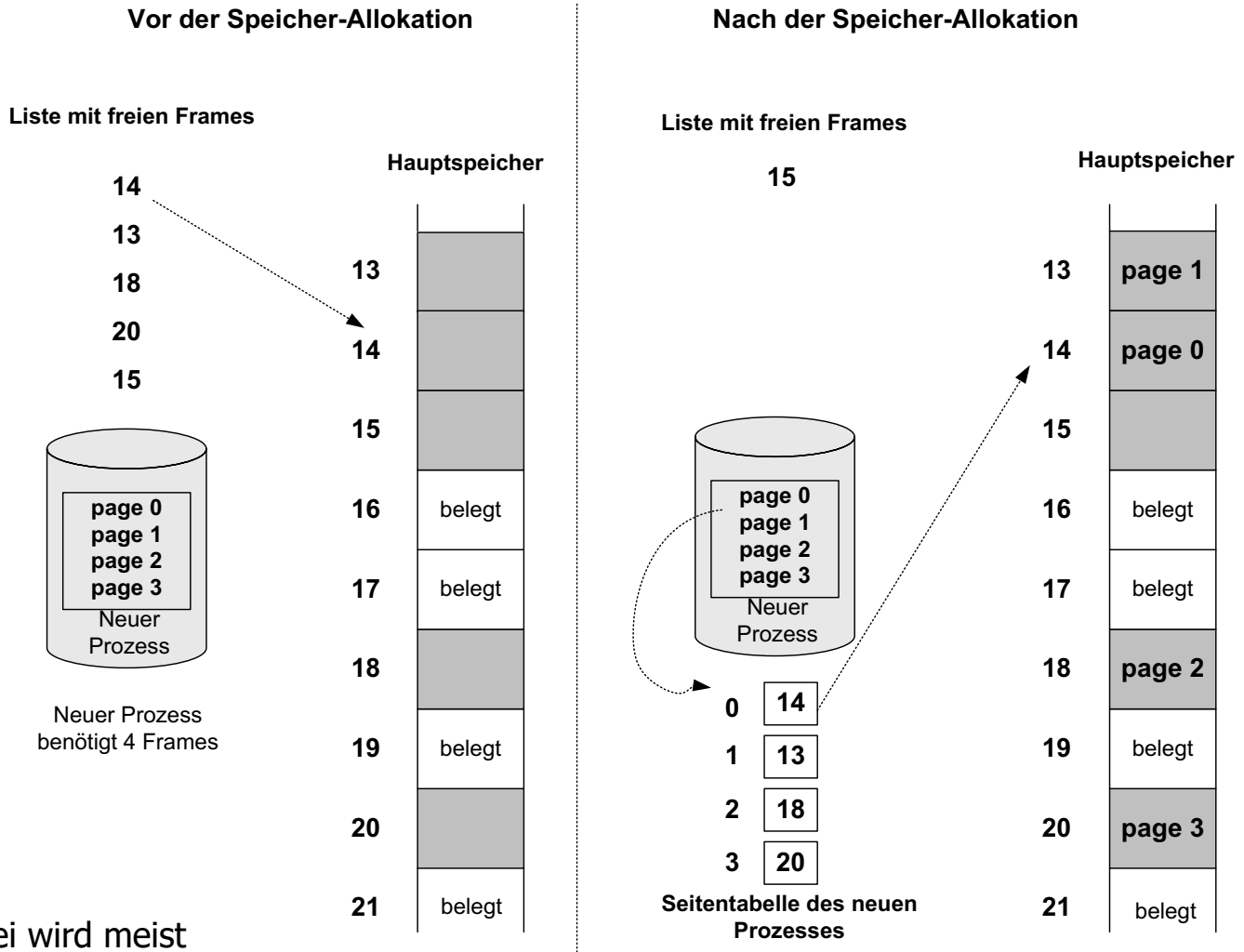
Quelle: *Tanenbaum, A. S.; Bos, H. Modern Operating Systems, Forth Edition, 2015*

Virtuelle Speichertechnik: Vorteile zusammengefasst

- Prozesse müssen **nicht komplett im RAM** liegen, um ablaufen zu können
- **Lineare Speicheradressierung**, keine Fragmentierung aus Programmiersicht
- Beim Prozesswechsel **behält** ein Prozess seine hauptspeicherresidenten Seiten. Er verliert sie erst, wenn sie von der Verwaltung des realen Speichers verdrängt werden
- Anwenderprogramme können den **vollen virtuellen Adressraum** nutzen, wenn genügend Festplattenspeicher vorhanden ist
- Der tatsächlich zugewiesene reale Speicherplatz ändert sich dynamisch entsprechend Angebot u. Nachfrage
- Speicherschutzmechanismen sind einfach zu realisieren (Kerneladressraum-Schutz, Schutz vor anderen Prozessen)

Klassische Seitenersetzungsstrategien

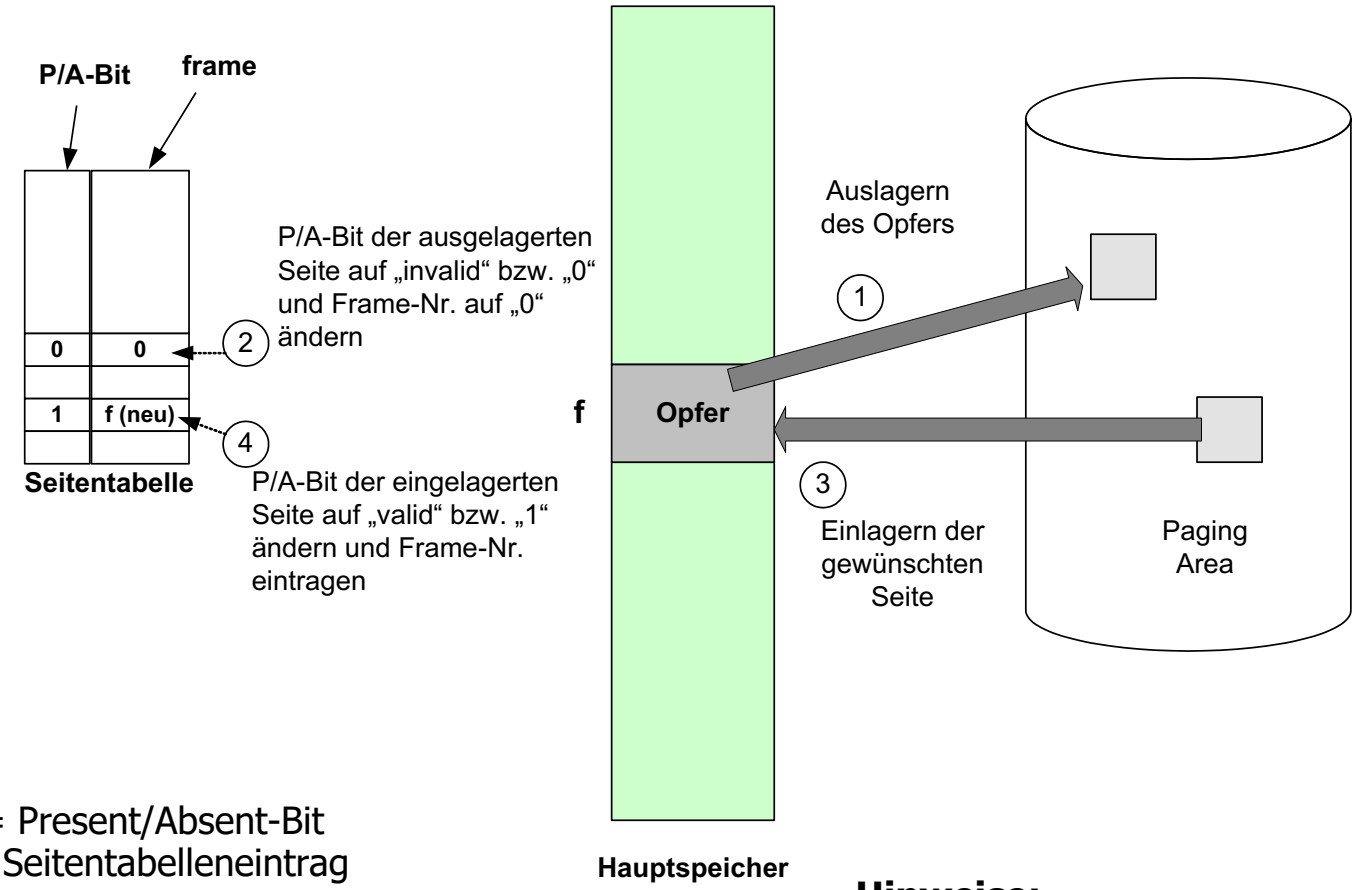
Szenario: Ein neuer Prozess benötigt Speicher und es ist genug Platz im Hauptspeicher



Hinweis:
Ausführbare Datei wird meist nicht auf Paging Area ausgelagert!

Vgl. auch Silberschatz et al.: Operating System Concepts, 2009

Szenario: Seitenanforderung aber nicht genug Platz im Hauptspeicher



P/A-Bit = Present/Absent-Bit
→ Siehe Seitentabelleneintrag

Nicht genug Platz vorhanden
→ Ersetzung notwendig

Hinweise:
- Interne und externe Fragmentierung
- Lokale oder globale Ersetzung

Vgl. auch Silberschatz et al.: Operating System Concepts, 2009

Page Fault und Belady

- Bei **Seitenzugriffsfehler** (page fault) muss ein Frame für die einzulagernde Seite gefunden werden
- Das Betriebssystem wählt ggf. eine Seite aus, die aus dem Speicher entfernt wird, um Platz zu schaffen
- Optimal wäre es, die zukünftigen Seitenzugriffe vorher zu bestimmen
- **Belady** (1966): Am wenigsten Ersetzungen sind erforderlich, wenn man die Seiten zur Verdrängung auswählt, die am spätesten in der Zukunft benutzt werden

  → **schwer zu realisieren, nur als Referenz**!

Demand Paging

- Die Strategie zur Auswahl dieser zu verdrängenden Seite wird in einem **Seitenersetzungs-algorithmus** festgelegt

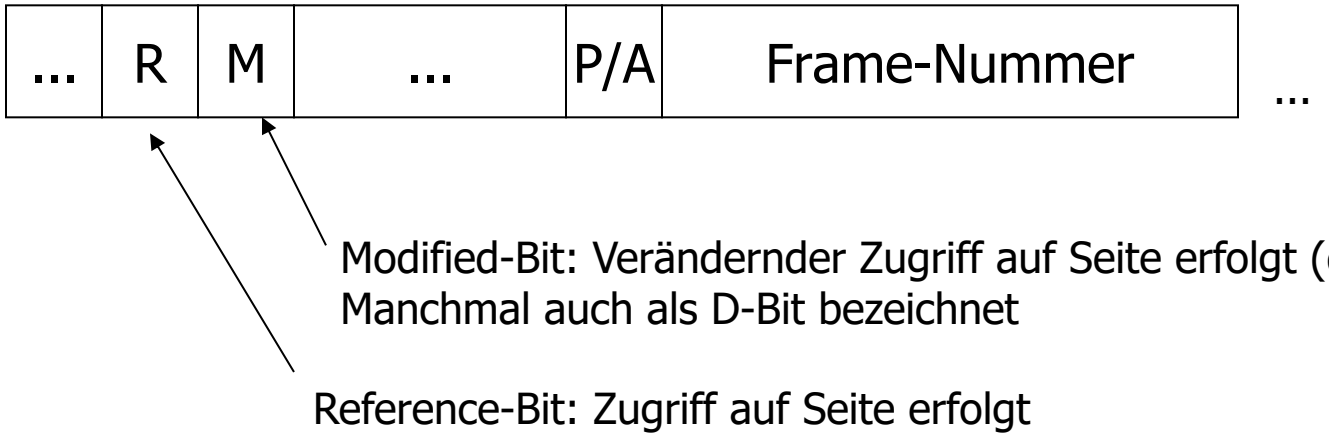
- Mögliche „bedarfsgerechte" Strategien (**Demand Paging**):

  - First-In, First-Out (FIFO)
  - Second-Chance, Clock-Page
  - Least-Recently-Used (LRU)
  - Not-Frequently-Used (NFU)
  - ...

- Kurzzeitstatistiken erforderlich: Speicherung in den Seitentabelleneinträgen

Zur Erinnerung: Seitentabelleneintrag

- Beispiel für einen Aufbau eines Eintrags in der Seitentabelle
- R- und M-Bit wichtig für Seitenersetzung



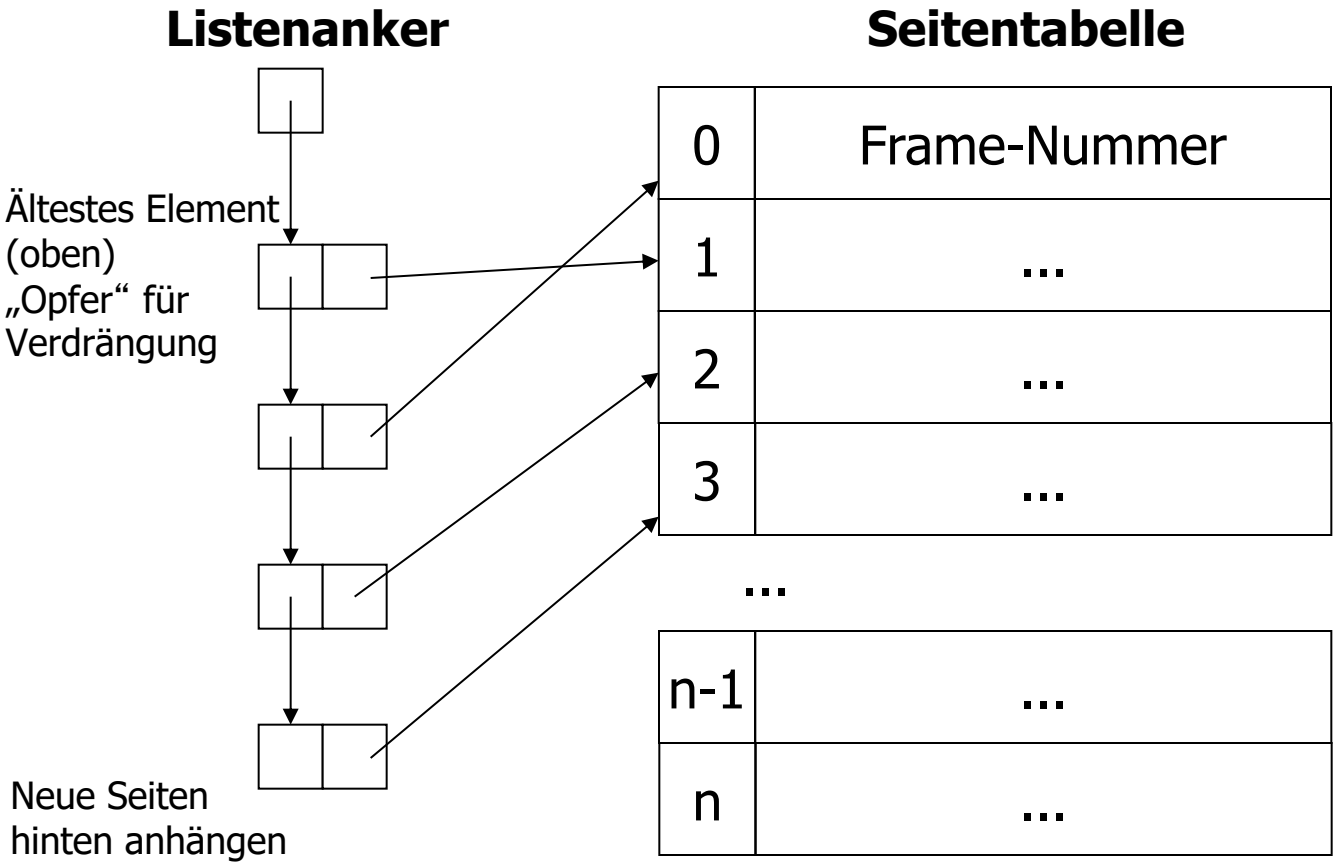
FIFO

- First-In First-Out: Die älteste Seite wird ersetzt
- Einfach zu implementieren
  - FIFO-Liste über alle Seitentabelleneinträge
  - Recht einfach zu implementieren, geringer Overhead, in konkreten Betriebssystemen im Einsatz
- Nachteil: Wirft möglicherweise wichtige Seiten aus dem Hauptspeicher (alt, aber oft benutzt)
- R-Bit nicht notwendig
- Seitentabelleneintrag:

Modified-Bit: Verändernder Zugriff auf Seite erfolgt (dirty bit)
Manchmal auch als D-Bit bezeichnet

FIFO: Seitentabelleneintrag

- FIFO-Liste muss verwaltet werden (kein Umhängen notwendig)



Second Chance

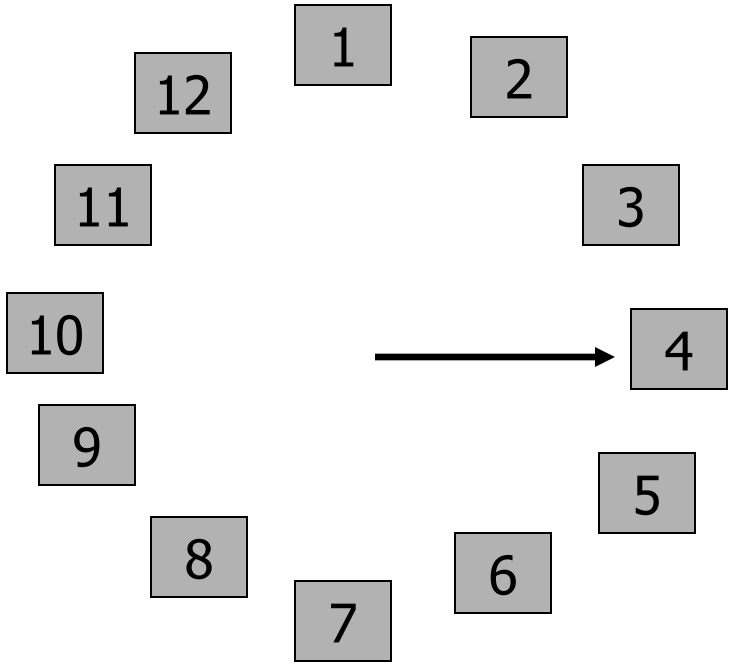
- Verbesserung von FIFO
- Auch das R-Bit (Referenz-Bit) wird inspiziert → Aging
- Ist älteste Seite benutzt worden, wird sie nicht ausgelagert, sondern an das Ende der Liste gehängt
  - Achtung: Einlagerung nicht gleich Nutzung!
- Wenn alle Seiten referenziert wurden, entspricht die Auswahl der zu ersetzenden Seite dem FIFO-Algorithmus
- Seitentabelleneintrag:

Clock Page (1)

- **Clock Page**
  - **Implementierung** zu Second Chance
  - Seiten werden in zirkulierender Liste wie eine Uhr verwaltet
  - Bei einem Seitenfehler wird immer die Seite untersucht, auf die gerade der „Uhrzeiger" verweist, der **Seitentabelleneintrag wird nicht umgehängt**

Clock Page (2)

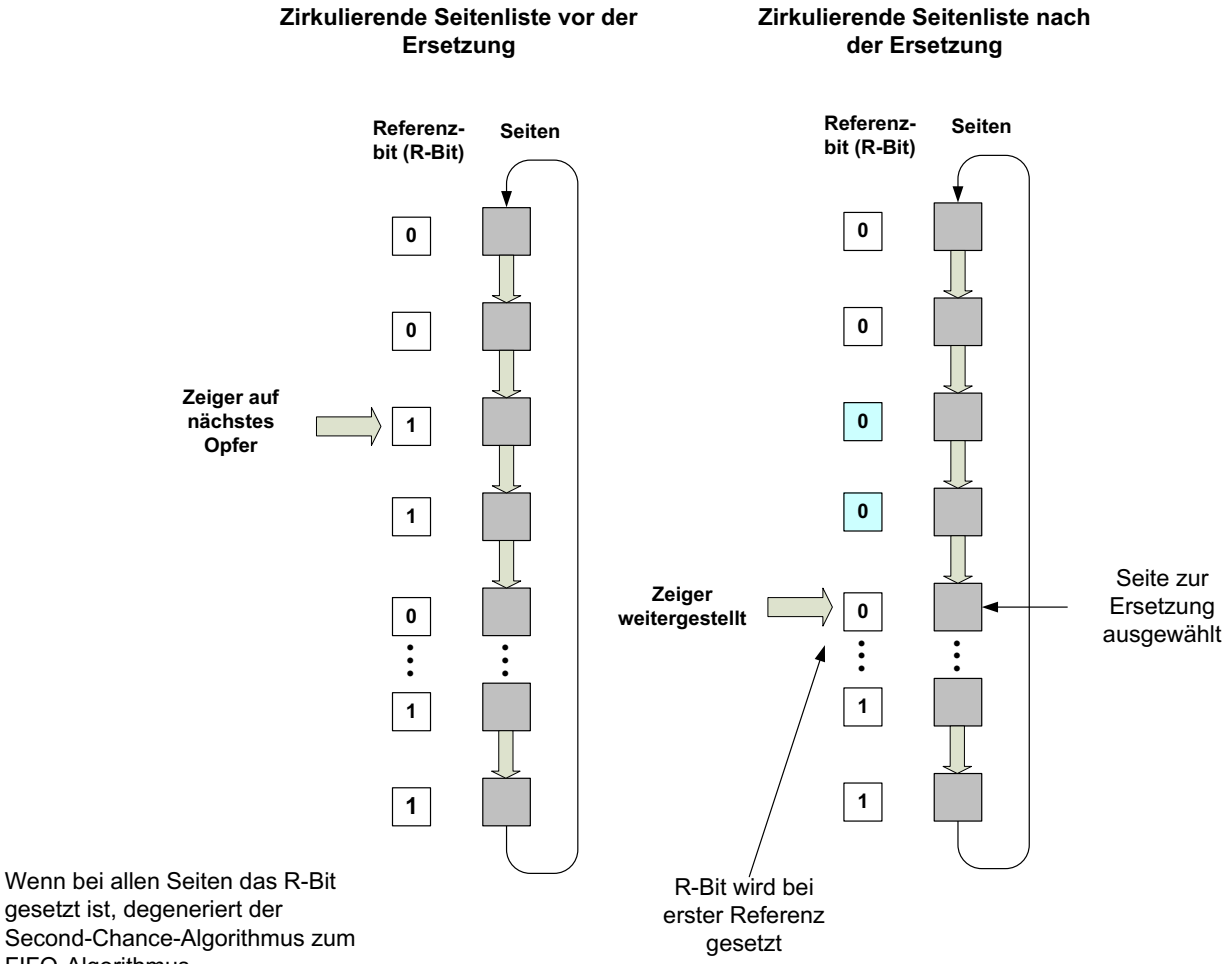
- **Clock Page Algorithmus**



Bei page fault:

- Seite, auf die Zeiger verweist wird ausgelagert, falls R-Bit = 0
- Wenn R = 1, wird R = 0 gesetzt und der Zeiger auf die nächste Seite gestellt
- Das geht solange, bis eine Seite mit R = 0 gefunden wird

Clock Page (3)



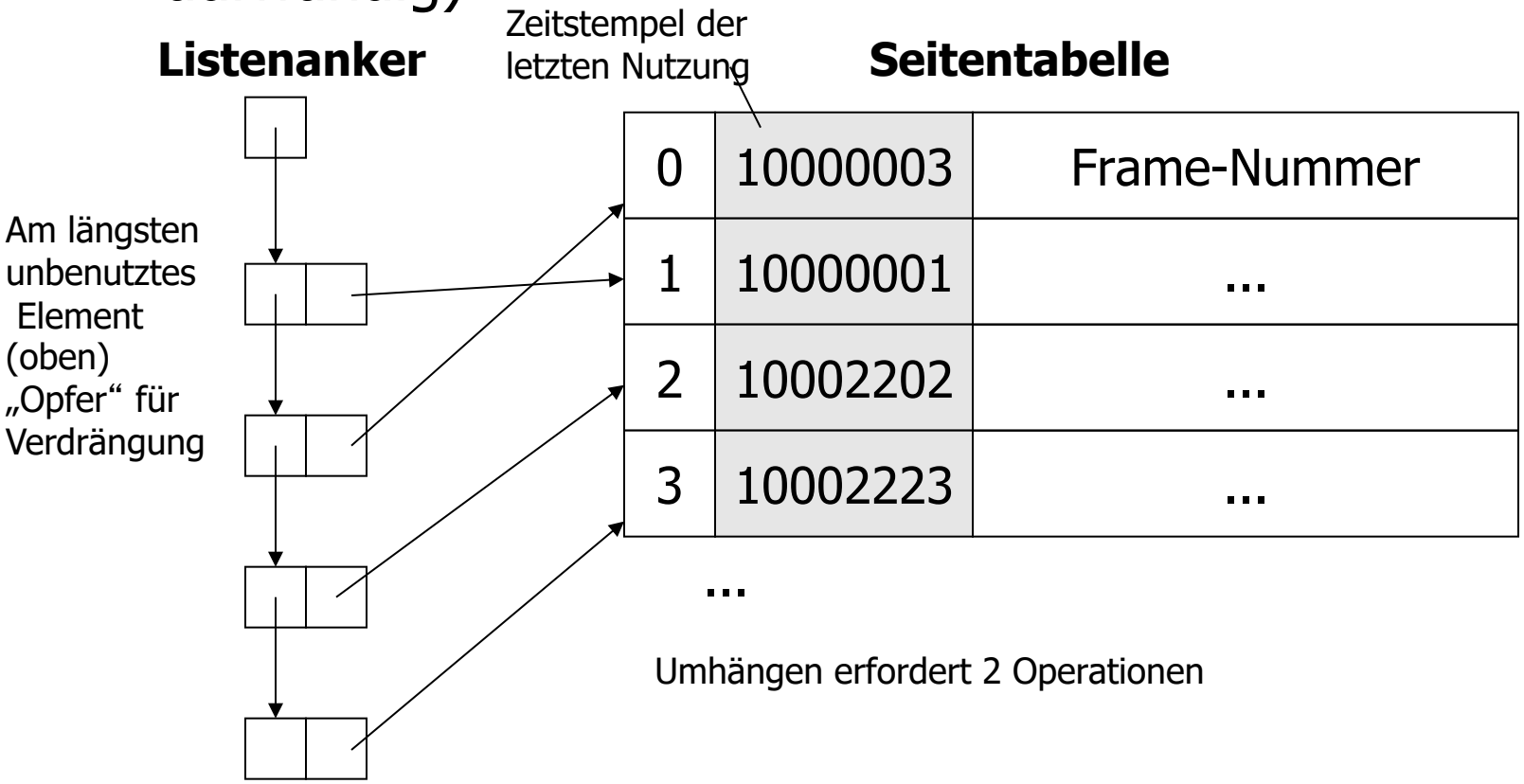
Vgl. auch Silberschatz et al.: Operating System Concepts, 2009

LRU

- **Least-Recently-Used:** Liefert gute Ergebnisse
- Die Seite wird ersetzt, deren **letzte Nutzung zeitlich am weitesten zurückliegt**
  - Der Zeitpunkt, seitdem die Seite unbenutzt ist, wird festgehalten → quantitative Zeitmessung notwendig
- Aber: Verfahren ist **aufwändig** zu realisieren:
  - Z. B.: Verkettete Liste mit den am weitesten in der Vergangenheit verwendeten Seiten am Anfang (absteigend sortiert)
  - Update der Liste bei **jedem** Zugriff auf den Speicher (Aufwand des Umhängens!)
  - Eigene Hardware (in der MMU) zur Berechnung **notwendig**, aber in der Regel nicht vorhanden

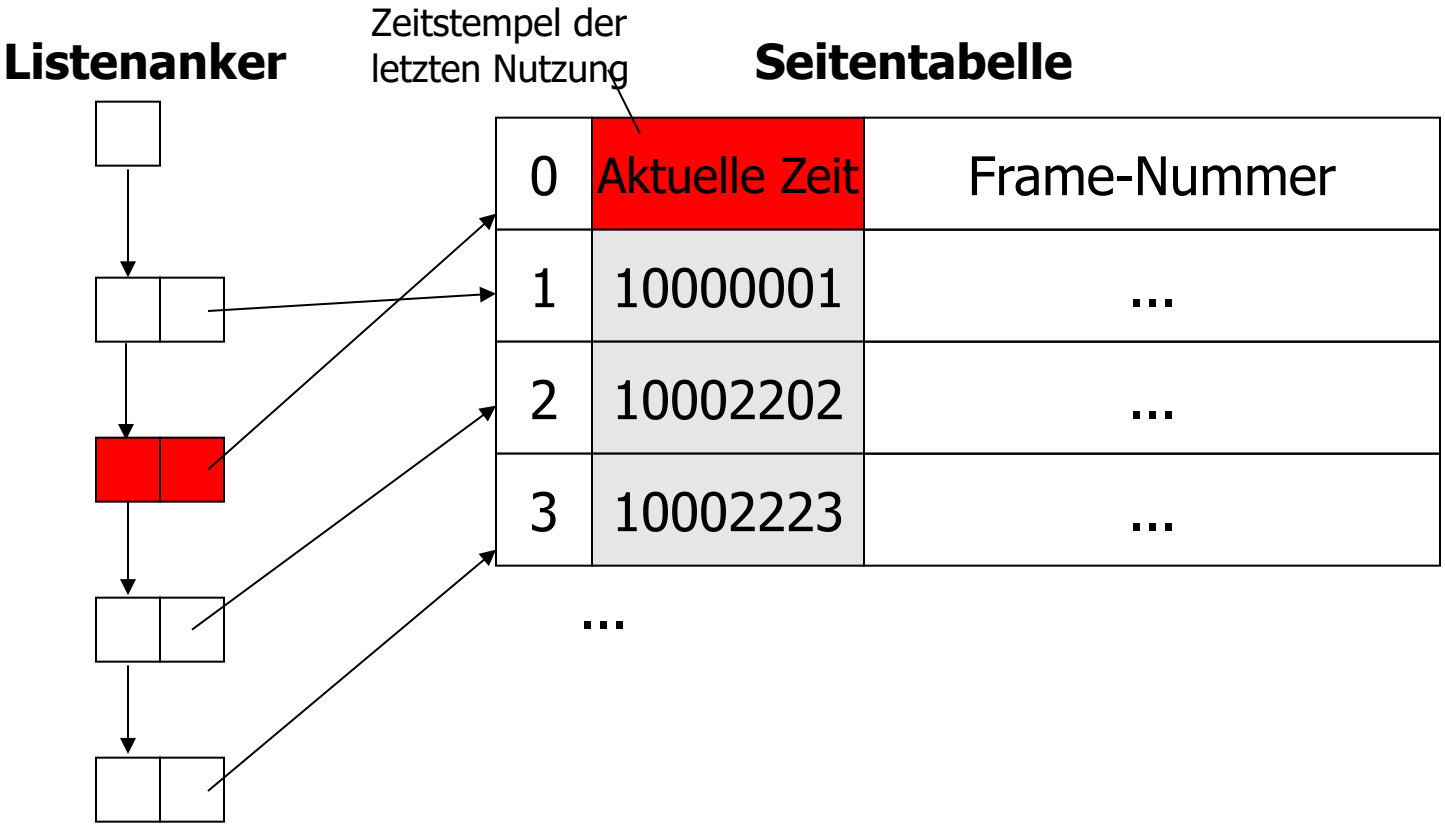
LRU: Verwaltung in einer Liste (1)

- LRU-Liste muss verwaltet werden (Umhängen ist aufwändig)



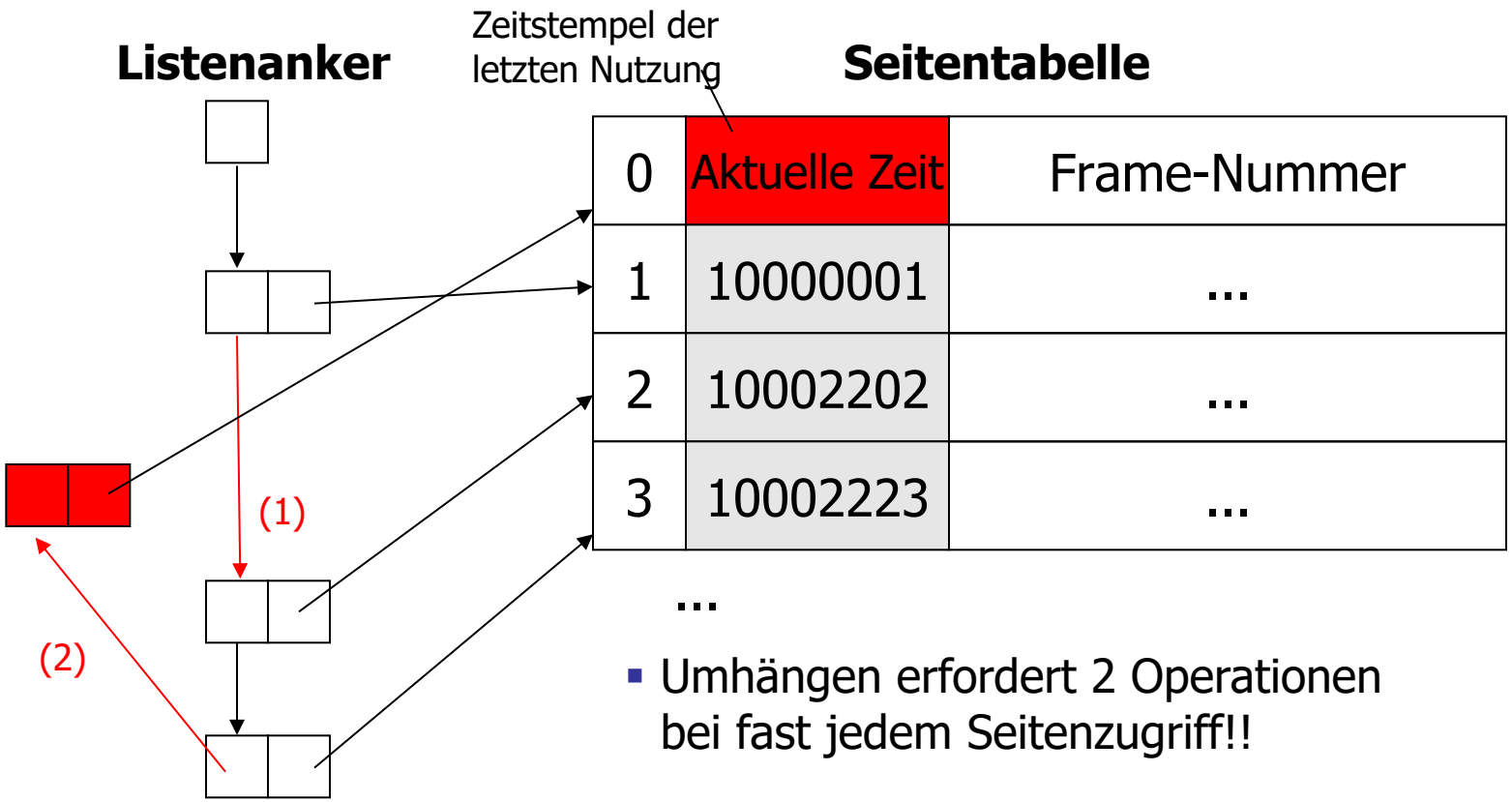
LRU: Verwaltung in einer Liste (2) - Umhängen

- Element in der Liste umhängen (1)
- Element im Zugriff kommt an das Ende der Liste



LRU: Verwaltung in einer Liste (3) - Umhängen

- Element in der Listen umhängen (2)



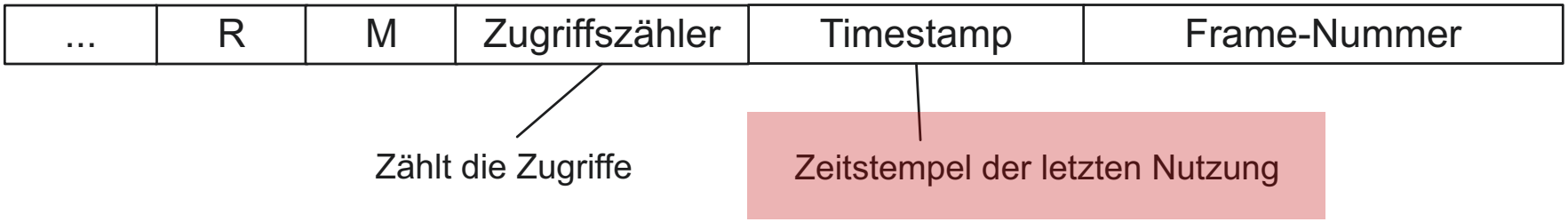
- Umhängen erfordert 2 Operationen bei fast jedem Seitenzugriff!!

NFU (1)

- **Not-Frequently Used:** Gute Annäherung an LRU
  - Diejenigen Seiten ersetzen, die in letzter Zeit **selten genutzt** wurden
  - Eintrag in der Seitentabelle erhält einen **Zugriffszähler** (initialisiert mit dem Wert 0)
  - Der Zähler wird bei Benutzung erhöht und auch das R-Bit wird auf 1 gesetzt
  - Bei einem Seitenzugriffsfehler wird die Seite mit dem kleinsten Wert im Zähler zur Ersetzung ausgewählt

NFU (2)

- **Problem bei NFU**
  - häufige (in der Vergangenheit liegende) Seitenzugriffe
    → Seiten werden nicht ausgelagert
- **Verbesserung**
  - Alterung durch zyklisches Rücksetzen des R-Bits berücksichtigen
  - → **NFU mit Aging kommt LRU schon sehr nahe**

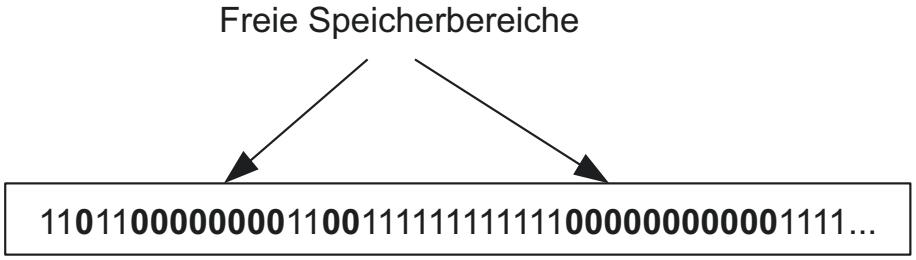


- Optimierte Implementierungsverfahren verfügbar

Placement und Cleaning

Speicherbelegungstrategien (Placement)

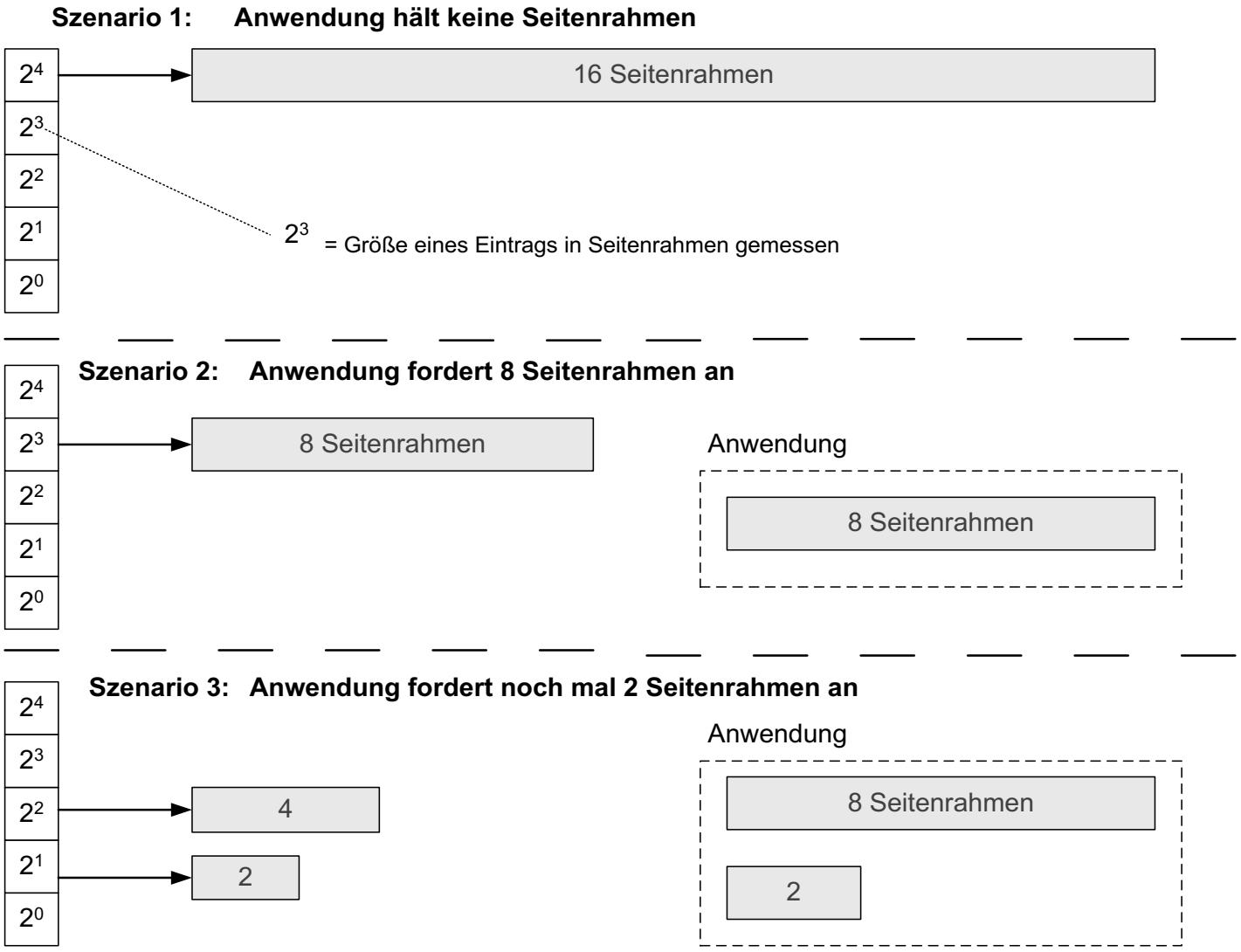
- Vermeidung von Fragmentierung anzustreben
  - Externe und interne Fragmentierung!
- Die Belegung des Hauptspeichers wird z. B. in **Speicherbelegungstabellen** verwaltet
- Die Realisierung kann z.B. als Bit Map erfolgen:
  - Jedem Rahmen wird ein Bit zugeordnet
    - 0 = frei
    - 1 = belegt
- Freie Hauptspeicherbereiche erkennt man dann an Nullfolgen:



Speicherbelegungstrategien: Suche nach freien Seiten

- Mögliche Vergabestrategien:
  - **Sequentielle Suche**, erster geeigneter Bereich wird vergeben (First-Fit)
  - **Optimale Suche** nach dem passendsten Bereich, um Fragmentierung möglichst zu vermeiden (Best-Fit) → aufwendig
  - **Buddy-Technik**: Schrittweise Halbierung des Speichers bei einer Hauptspeicheranforderung
    - Speichervergabe:
      - Suche nach kleinstem geeigneten Bereich
      - Halbierung des gefundenen Bereichs, solange bis gewünschter Bereich gerade noch in einen Teilbereich passt
    - Bei Hauptspeicherfreigabe werden Rahmen wieder zusammengefasst:
      - Zurückgegebenen Bereich mit allen freien Nachbarbereichen (und deren Partnern) verbinden und zu einem Bereich machen

Speicherbelegungstrategien: Buddy-Technik



Entladestrategien (Cleaning)

- Legt den Zeitpunkt fest, wann eine modifizierte Seite auf die Paging-Area geschrieben wird
- „Sauberhalten des Hauptspeichers“
- Varianten:
  - **Demand-Cleaning**: Bei Bedarf
    - Vorteil: Seiten sind lange im Hauptspeicher
    - Nachteil: Verzögerung bei Seitenwechsel
  - **Precleaning**: Präventives Zurückschreiben, wenn Zeit ist
    - Vorteil: Frames in der Regel verfügbar
  - **Page-Buffering**: Listen verwalten, z.B.:
    - Modified List: Liste mit geänderten Seiten (siehe Windows)
    - Unmodified List: Seiten sind zum Entladen freigegeben
    - Active und inactive List (siehe Linux)
- Heutige Betriebssysteme verwenden Mischstrategien

Fazit zum virtuellen Speicher

- Die virtuelle Adressierung ist **relativ aufwändig**, da
  - viele umfangreiche Tabellen benötigt werden (Seitentabelle pro Prozess)
  - ein Teil der Festplatte als Paging-Area verwendet wird
  - laufend untersucht werden muss, ob Seiten hauptspeicherresident bleiben oder auf Platte auszulagern sind (Seitenersetzungsalgorithmus)
- Trotz des Overheads: Virtuelle Adressierung ist das heute am meisten verwendete Verfahren
- Optimierungen notwendig
  - z. B. durch Hardwareunterstützung → Übersetzungspuffer bzw. Translation Lookaside Buffer (vgl. Cache)