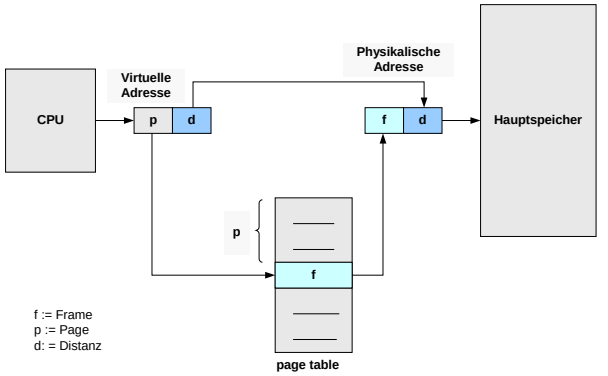
MAS: Betriebssysteme

Speicherverwaltung – Strategien

T. Pospíšek

Gesamtüberblick

1. Einführung in Computersysteme
2. Entwicklung von Betriebssystemen
3. Architekturansätze
4. Interruptverarbeitung in Betriebssystemen
5. Prozesse und Threads
6. CPU-Scheduling
7. Synchronisation und Kommunikation
8. **Speicherverwaltung**
9. Geräte- und Dateiverwaltung
10. Betriebssystemvirtualisierung



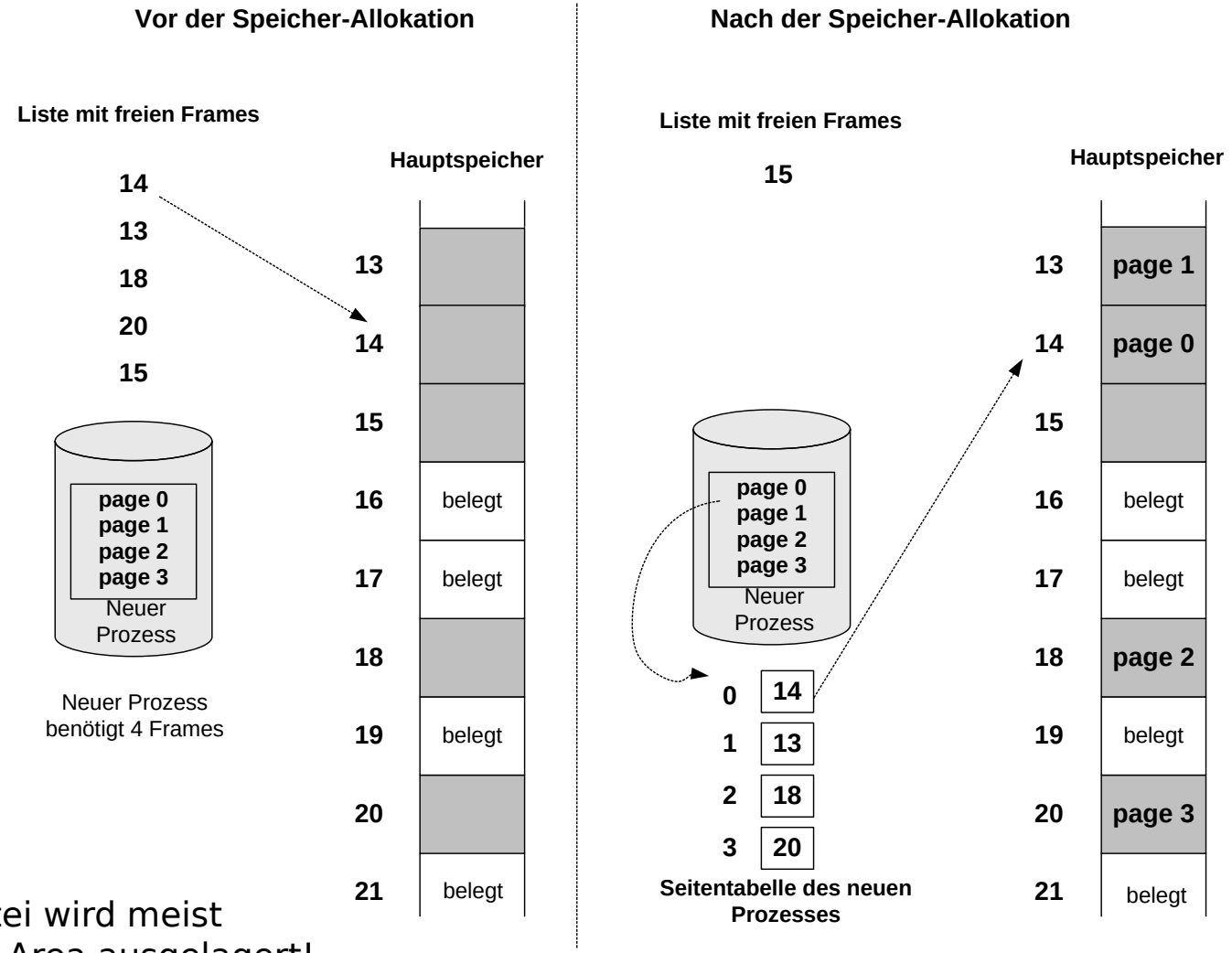
Zielsetzung

- Weiterführende Konzepte der Speicherverwaltung, insbesondere des Hauptspeichers, kennenlernen und verstehen

Überblick

1. **Seitenersetzung und Verdrängung (Replacement)**
2. Speicherbelegung und Vergabe (Placement)
3. Entladen (Cleaning)
4. Fallbeispiele: Windows, Unix, Linux

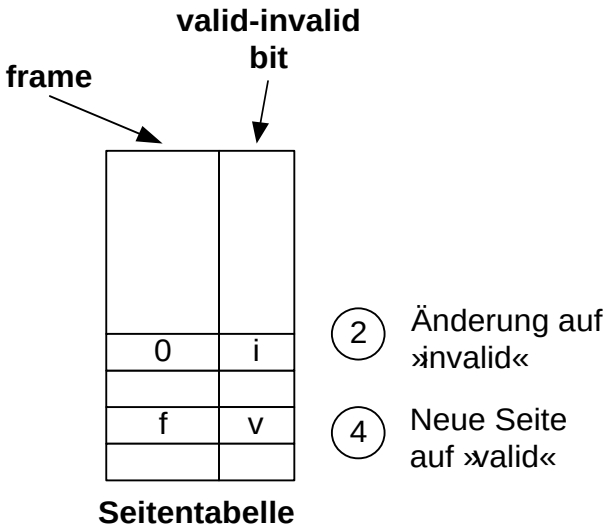
Szenario: Ein neuer Prozess benötigt Speicher und genug Platz im Hauptspeicher

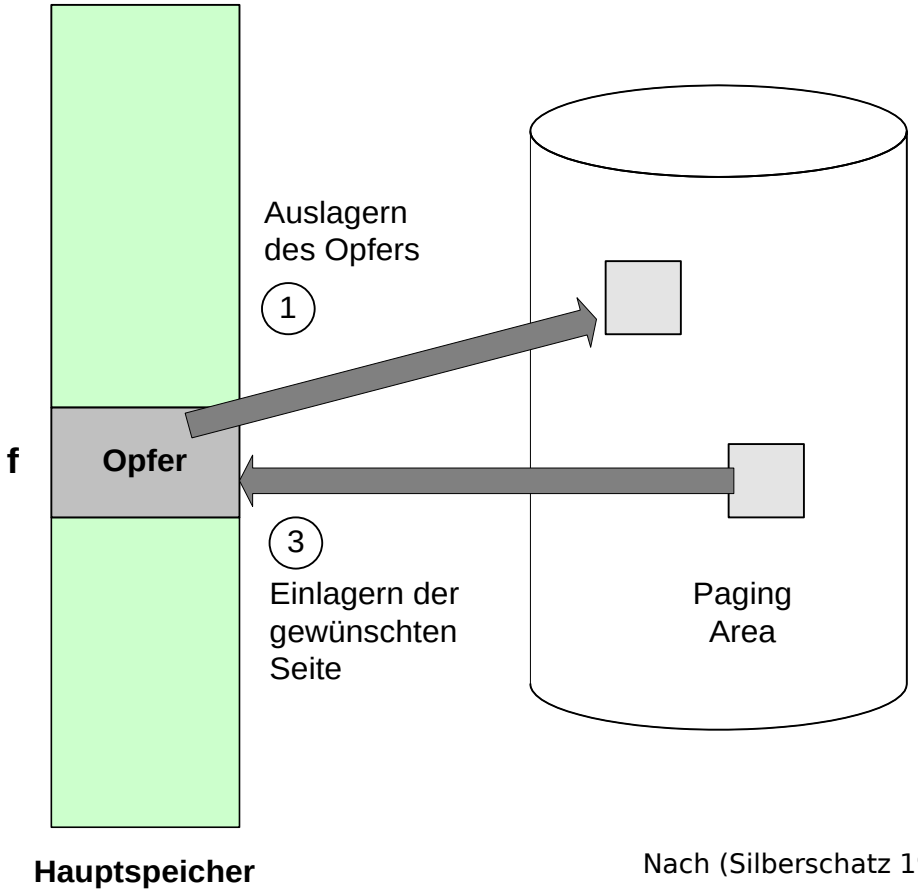


Hinweis:
Ausführbare Datei wird meist nicht auf Paging Area ausgelagert!

Nach (Silberschatz 1994)

Szenario: Seitenanforderung aber nicht genug Platz im Hauptspeicher





Nach (Silberschatz 1994)

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

Nicht genug Platz vorhanden
→ Ersetzung notwendig

Hinweise:

- Interne und externe Fragmentierung
- Lokale oder globale Ersetzung

Einschub: interne vs externe Fragmentierung

- **Intern:**
  es werden 23 Bytes gebraucht, das System kann aber nur minimal 32 Bytes liefern → 9 Bytes durch interne Fragmentierung verloren
- **Extern:**
  das System vergibt Speicher in gleich grossen Blöcken.
  Es hat 3 Blöcke am Stück.
  Die erste Anwendung braucht einen Block und bekommt vom System Block #2. Die nächste Anwendung braucht zwei Blöcke am Stück. Obwohl das System noch 2 freie Blöcke hat, kann es die Anwendung nicht bedienen, da diese, aufgrund der „dummen“ Vergabestrategie, nicht am Stück sind.

Einschub: lokale vs globale Seitenersetzung

- wenn ein Frame freigemacht werden muss, dann können bei der Entscheidung:
  - **lokal:**
    nur Frames des Prozesses zur Auswahl stehen
  - **global:**
    die Frames aller Prozesse zur Auswahl stehen

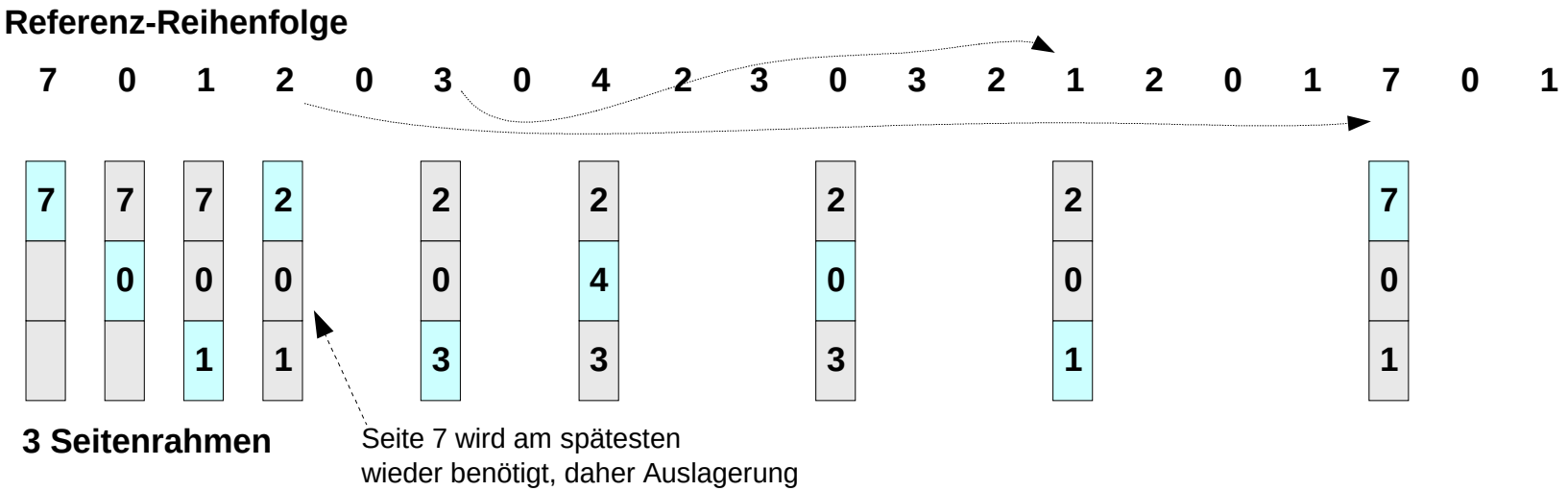
Page Fault und Belady

- Bei einem **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**!

Belady

- Einfaches Beispiel mit 3 Frames
- 6 Ersetzungen nach der ersten Belegung



Nach (Silberschatz 1994)

Beispiel Belady

- Zugriffsreihenfolge: 0-1-2-3-4-0-1-5-6-0-1
- Nach Belady: (4 Ersetzungen)

Frames

Zugr	0	1	2	3	4	0	1	5	6	0	1
RAM	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**
RAM	**-**	**1**	**1**	**1**	**1**	**1**	**1**	**1**	**1**	**1**	**1**
RAM	**-**	**-**	**2**	**(3)**	**(4)**	**4**	**4**	**(5)**	**(6)**	**6**	**6**
PA				**2**	**2**	**2**	**2**	**2**	**2**	**2**	**2**
PA					**3**	**3**	**3**	**3**	**3**	**3**	**3**
PA								**4**	**4**	**4**	**4**
PA									**5**	**5**	**5**

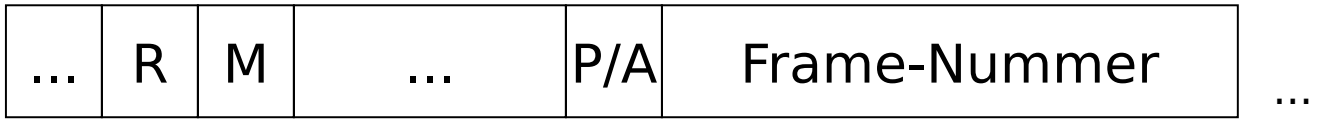
RAM = Realer Speicher (x) = Seitenersetzung notwendig
PA = Paging Area

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)
  - Not-Recently-Used (NRU)
  - 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



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

Reference-Bit: Zugriff auf Seite erfolgt.
Wird von der CPU (!) gesetzt!

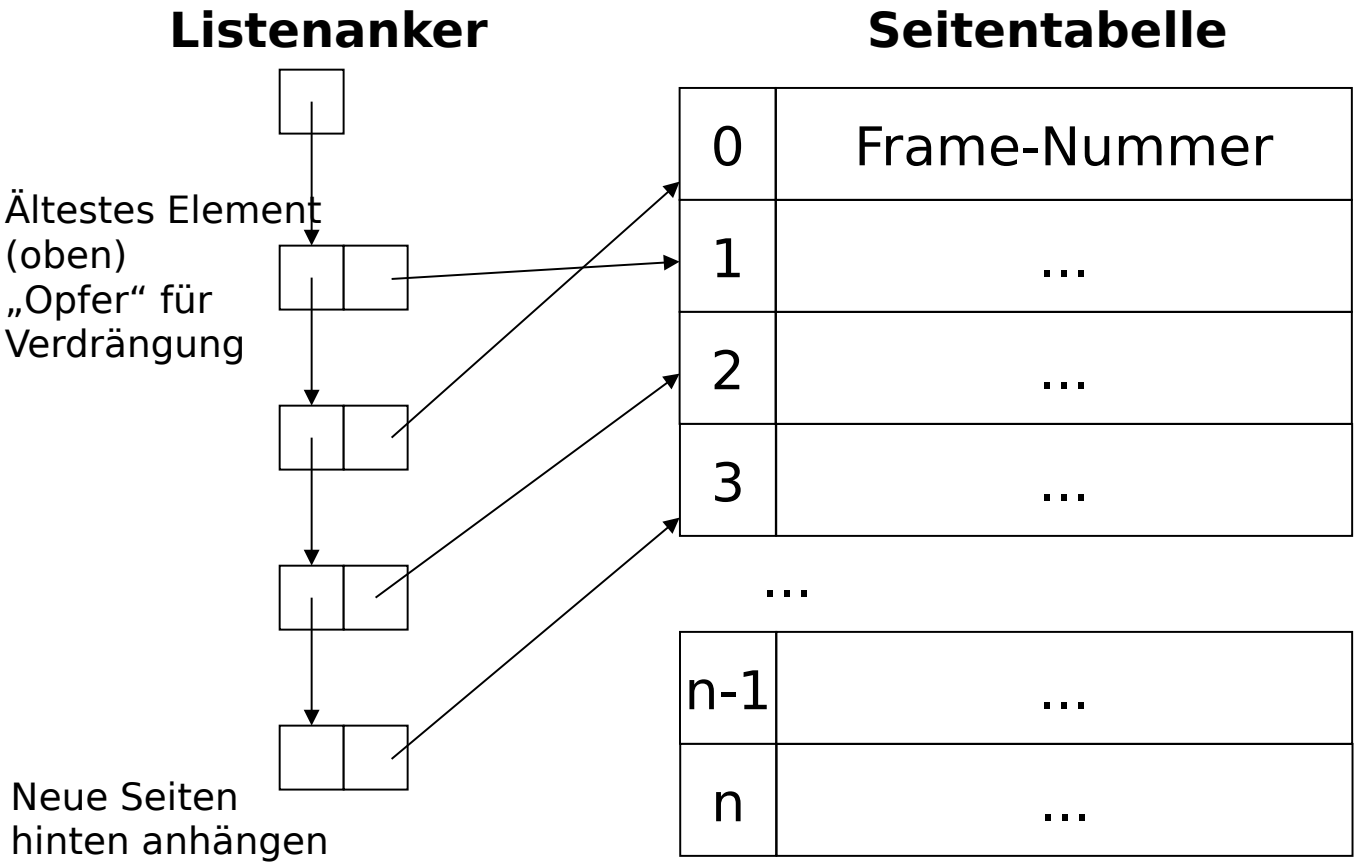
FIFO

- 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
- R-Bit nicht notwendig
- Seitentabelleneintrag:

...	...	M	Frame-Nummer

FIFO: Seitentabelleneintrag

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

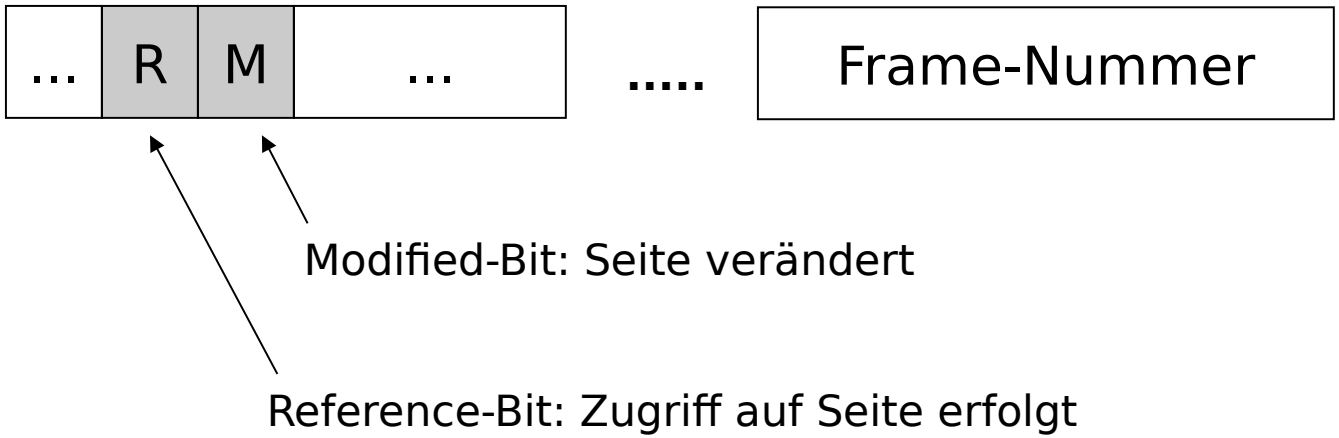


NRU

- Not Recently Used
- Seiten, die **in letzter Zeit nicht genutzt wurden**, sind Kandidaten für die Verdrängung
- Auch einfach zu implementieren (R/M-Bit nutzen), aber nur durchschnittliche Performance
  - 4 Klassen („Opfersuche“ in dieser Reihenfolge):
    - 1) R = 0, M = 0 (Seiten werden als erstes ausgelagert)
    - 2) R = 0, M = 1 (Verändert im vorhergehenden Intervall)
    - 3) R = 1, M = 0 (Nur lesender Zugriff im aktuellen Intervall)
    - 4) R = 1, M = 1 (Seiten werden als letztes ausgelagert)
  - Modifizierte Seiten sind besser gestellt
  - R-Bit wird periodisch vom Kernel zurückgesetzt, M-Bit nicht!

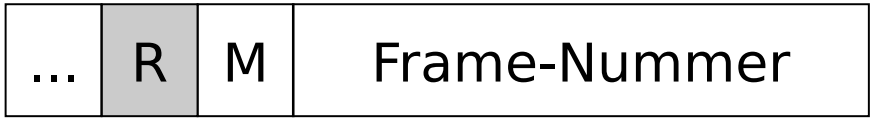
NRU: Seitentabelleneintrag

- Nur ein R- und M-Bit notwendig



Second Chance

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

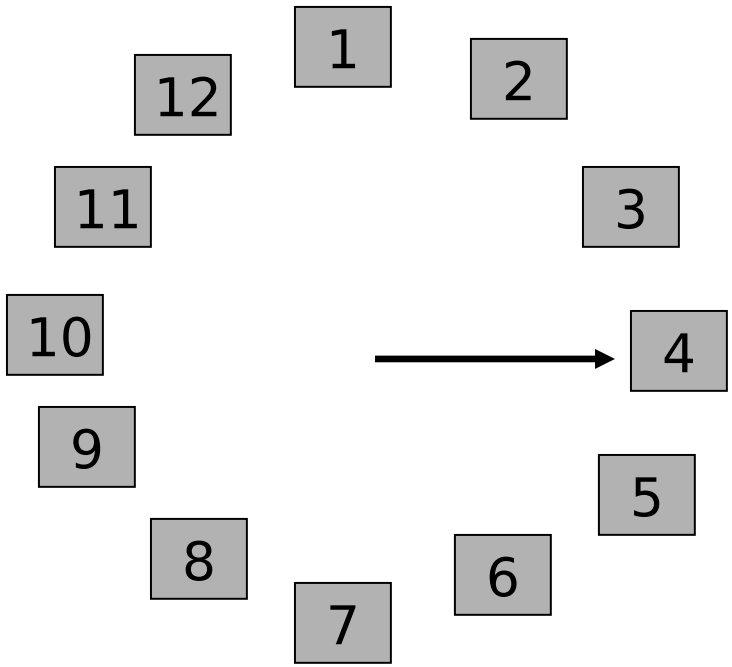


Clock Page (1)

- **Clock Page**
  - **Implementierungsverbesserung** 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)

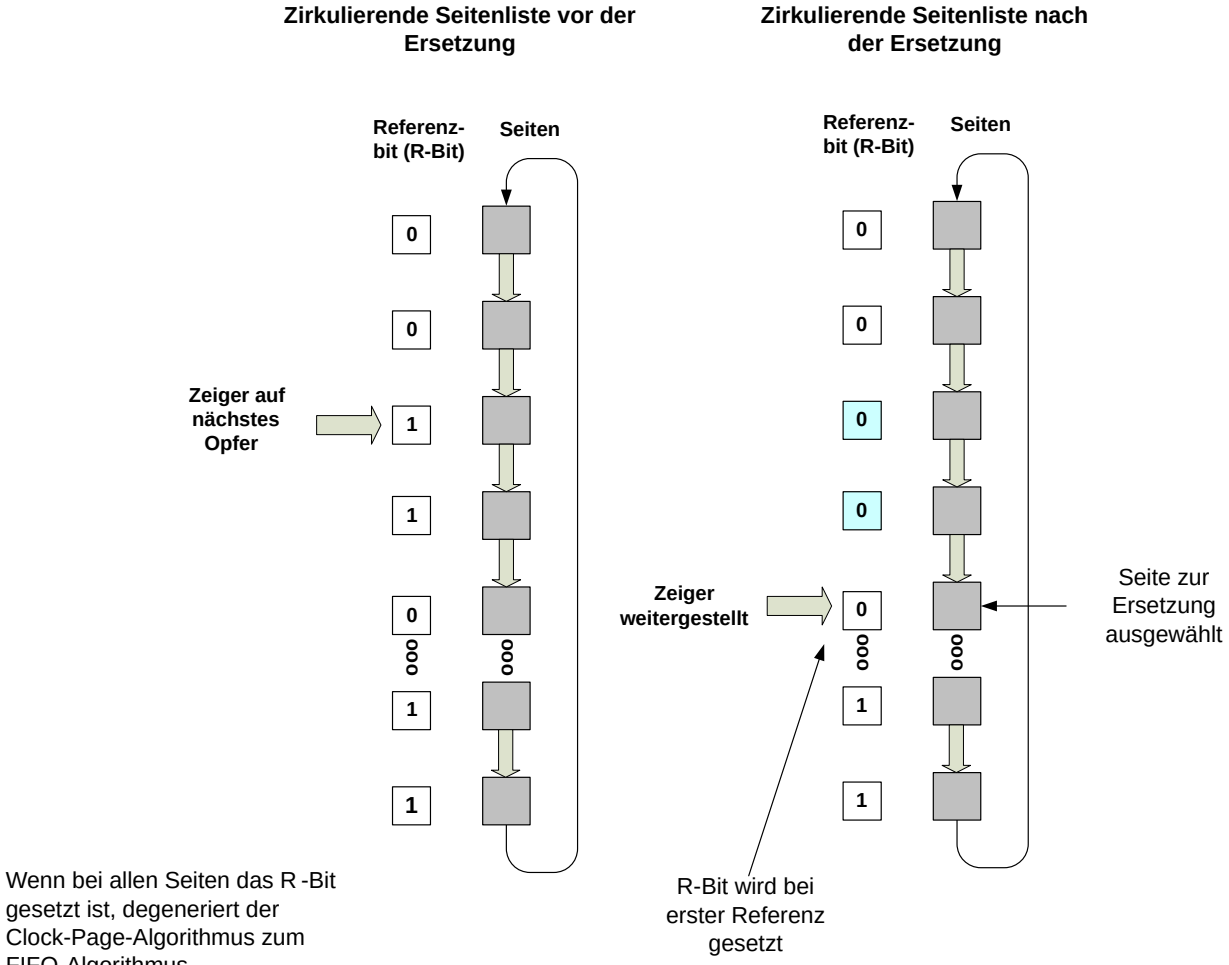
- **Clock Page Algorithmus**



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)



Wenn bei allen Seiten das R -Bit gesetzt ist, degeneriert der Clock-Page-Algorithmus zum FIFO-Algorithmus

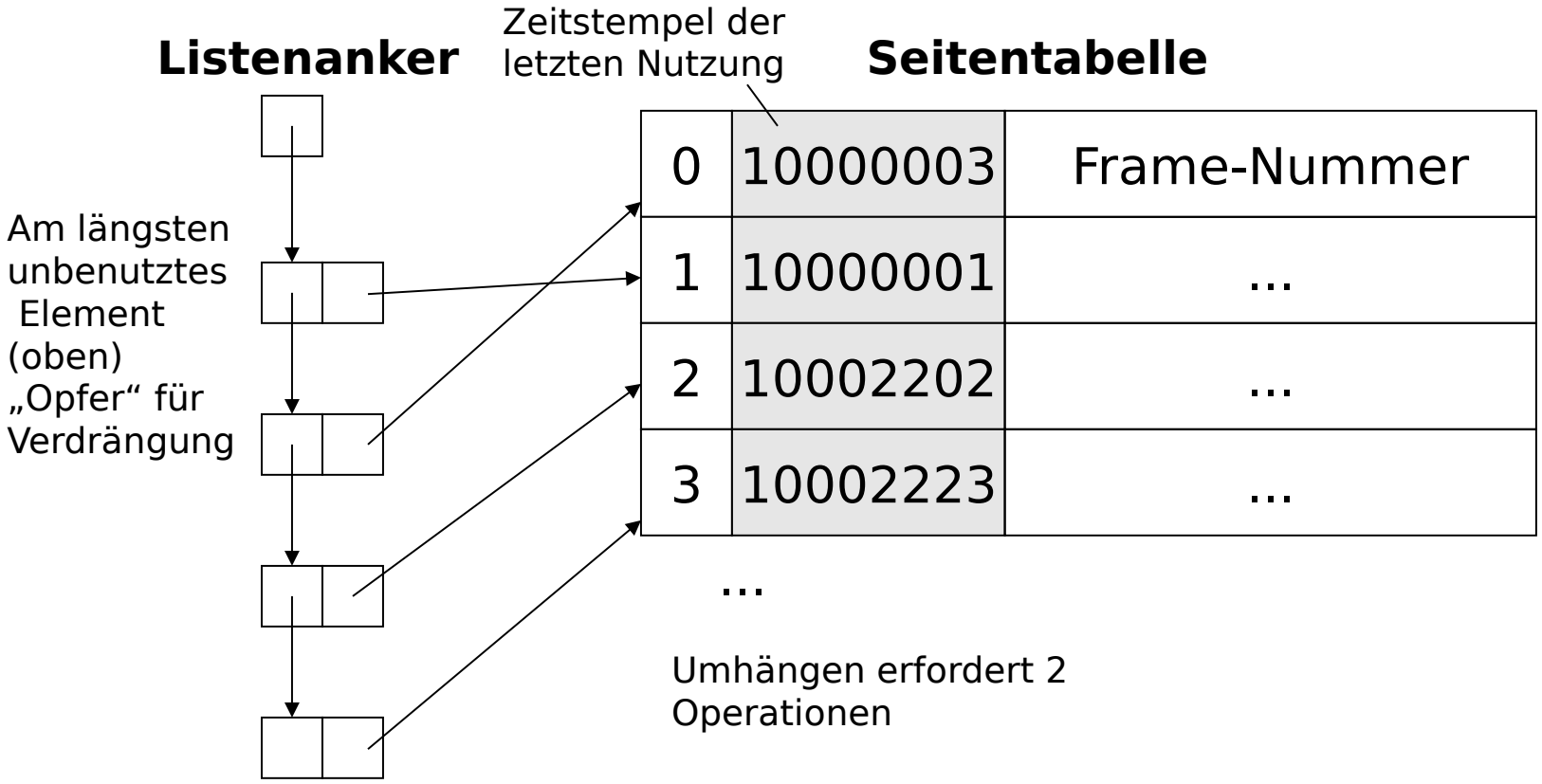
Nach (Silberschatz 1994)

LRU

- Least Recently Used: Seite wird ersetzt, deren letzte Nutzung zeitlich am weitesten zurückliegt
  - Der Zeitpunkt, seit dem die Seite unbenutzt ist, wird gemessen → quantitative Zeitmessung notwendig
- Gute Ergebnisse
- 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 (MMU) zur Berechnung sinnvoll (selten)

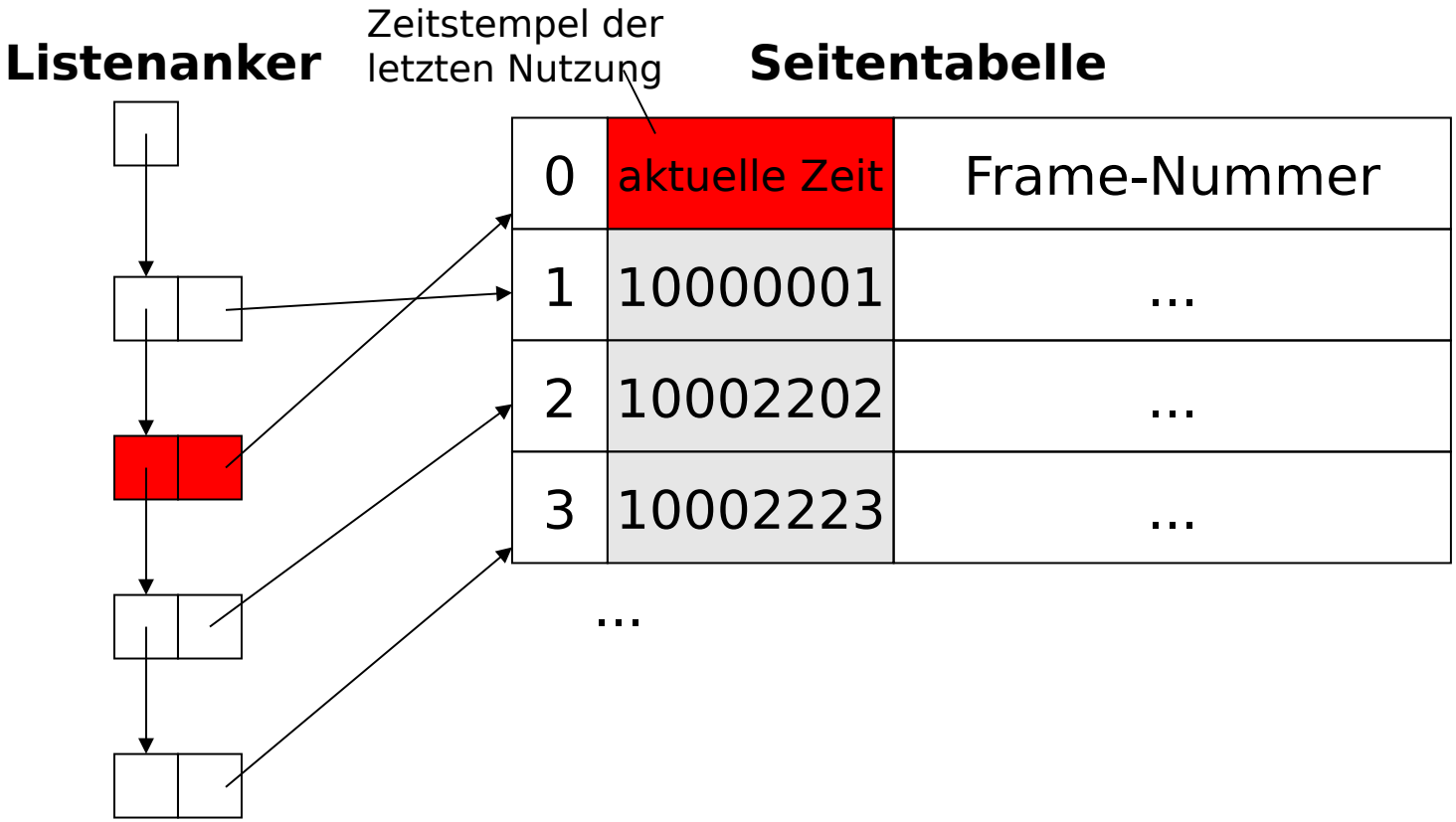
LRU: Verwaltung in einer Liste (1)

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



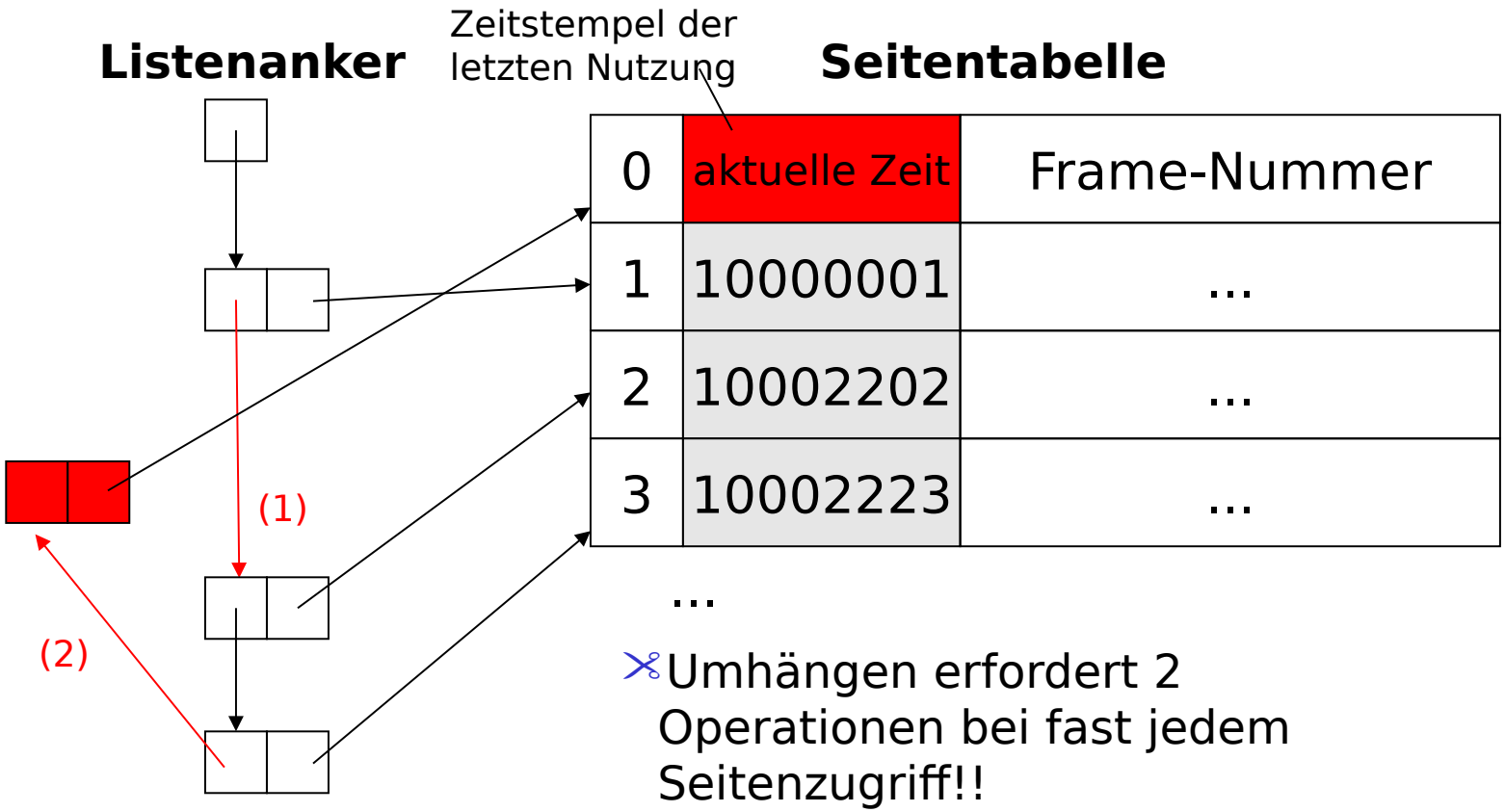
LRU: Verwaltung in einer Liste (2) - Umhängen

- 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)



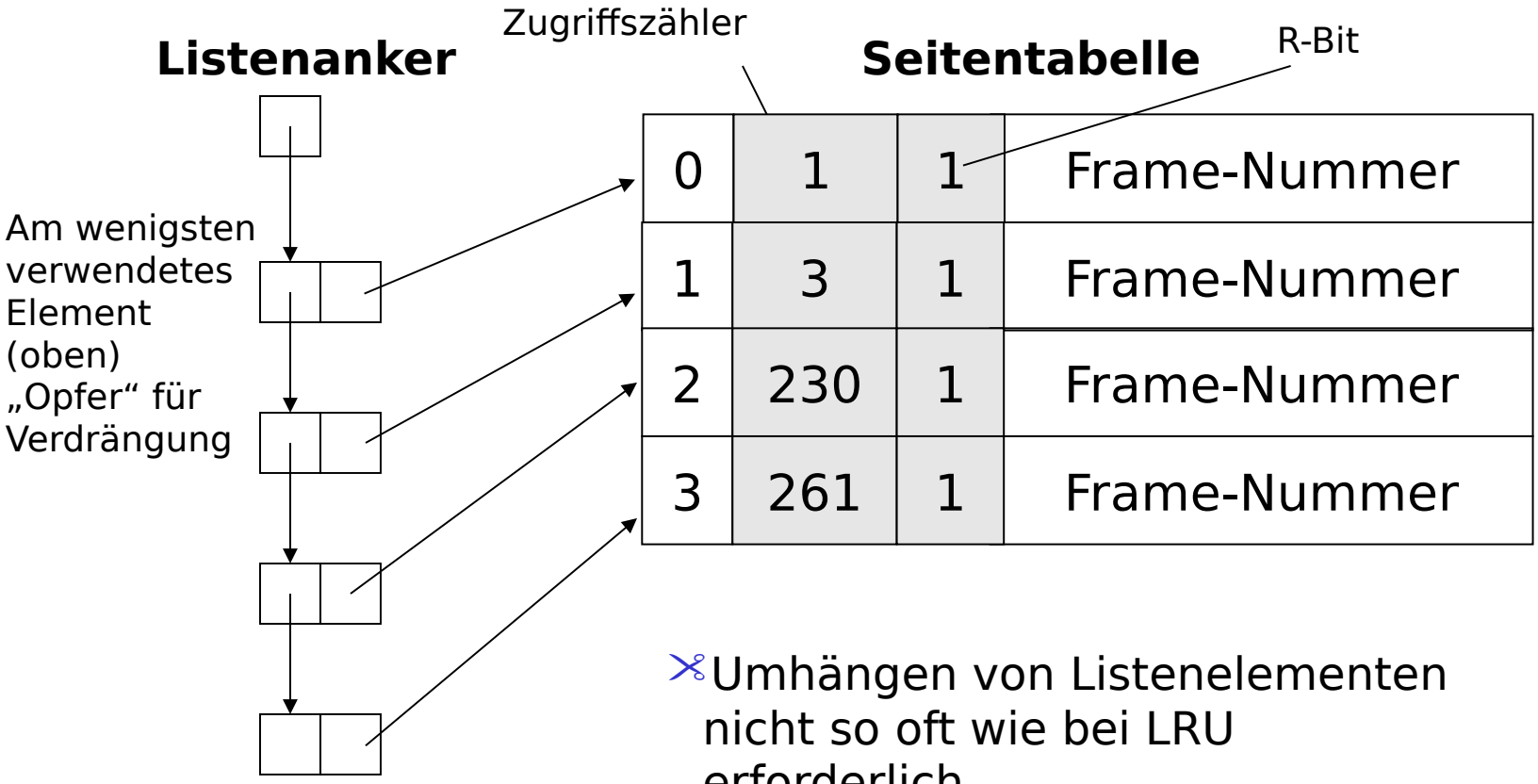
- Umhängen erfordert 2 Operationen bei fast jedem Seitenzugriff!!
- Sonderfälle beachten

NFU

- Eine gute Annäherung an LRU bietet das **NFU**-Verfahren (Not-Frequently Used)
  - Diejenigen Seiten ersetzen, die in einem Zeitintervall **selten genutzt** wurden
  - Eintrag in der Seitentabelle erhält einen **Zugriffszähler** (initialisiert mit dem Wert 0)
  - Der Zähler wird bei Benutzung (R-Bit = 1) erhöht
  - Bei einem Seitenzugriffsfehler wird die Seite mit dem kleinsten Wert im Zähler zur Ersetzung ausgewählt
- **Problem:** Auch alte, häufig zugegriffene Seiten, die nicht mehr verwendet werden, werden nicht ausgelagert
- Verbesserung: Alterung berücksichtigen → **NFU mit Aging kommt LRU schon sehr nahe**

NFU: Listenverwaltung (mit Aging)

- NFU-Liste muss verwaltet werden



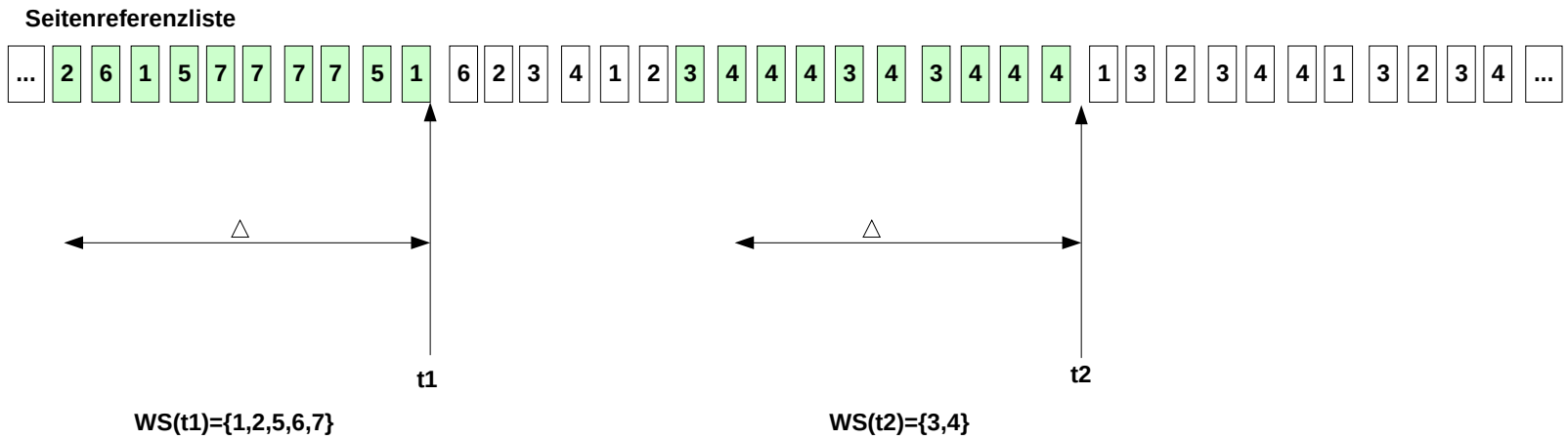
- Umhängen von Listenelementen nicht so oft wie bei LRU erforderlich

Prepaging, Working Set (1)

- Bisher diskutierte Verfahren: Demand Paging
- Nutzt man die Lokalität von Softwareprogrammen, so kann man auch sinnvoll **Prepaging** betreiben:
  - Also Seiten, die evtl. noch gar nicht angefordert wurden, in den Hauptspeicher lesen
- Die aktuell benötigte Seitenmenge wird auch als **Working Set** bezeichnet. Wenn diese Menge im Hauptspeicher ist, gibt es keinen Seitenzugriffsfehler
- Dieses Ziel versucht der Working-Set-Algorithmus zu erreichen

Prepaging, Working Set (2)

- Die letzten d Referenzen werden betrachtet und daraus wird der Working-Set ermittelt
- Beispiel: d = 10



Prepaging, Working Set (2)

- Der **Working-Set-Algorithmus** geht von folgender Annahme aus:
  - Die benötigten Seiten, also der Working Set, ändern sich nur langsam
  - Die in nächster Zukunft benötigten Seiten sind mit guter Wahrscheinlichkeit in der Nähe der gerade adressierten
- Das Verfahren macht es notwendig, sich die Menge der verwendeten Seiten zu merken
- Es wird **Prepaging** benutzt, um die erwarteten Seiten präventiv einzulagern
  - Einlagerung von **wahrscheinlich** benötigten Seiten eines schlafenden Prozesses, bevor dieser wieder aktiv ist

Überblick

1. Seitenersetzung und Verdrängung (Replacement)
2. **Speicherbelegung und Vergabe (Placement)**
3. Entladen (Cleaning)
4. Fallbeispiele: Windows, Unix, Linux

Speicherbelegungstrategien (Placement)

- Vermeidung von Fragmentierung anstreben
  - interne Fragmentierung!
  - externe Fragmentierung passiert nur, wenn verschieden grosse Pages eingesetzt werden
- Die Belegung des Hauptspeichers wird 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 nebeneinander liegenden Nullen

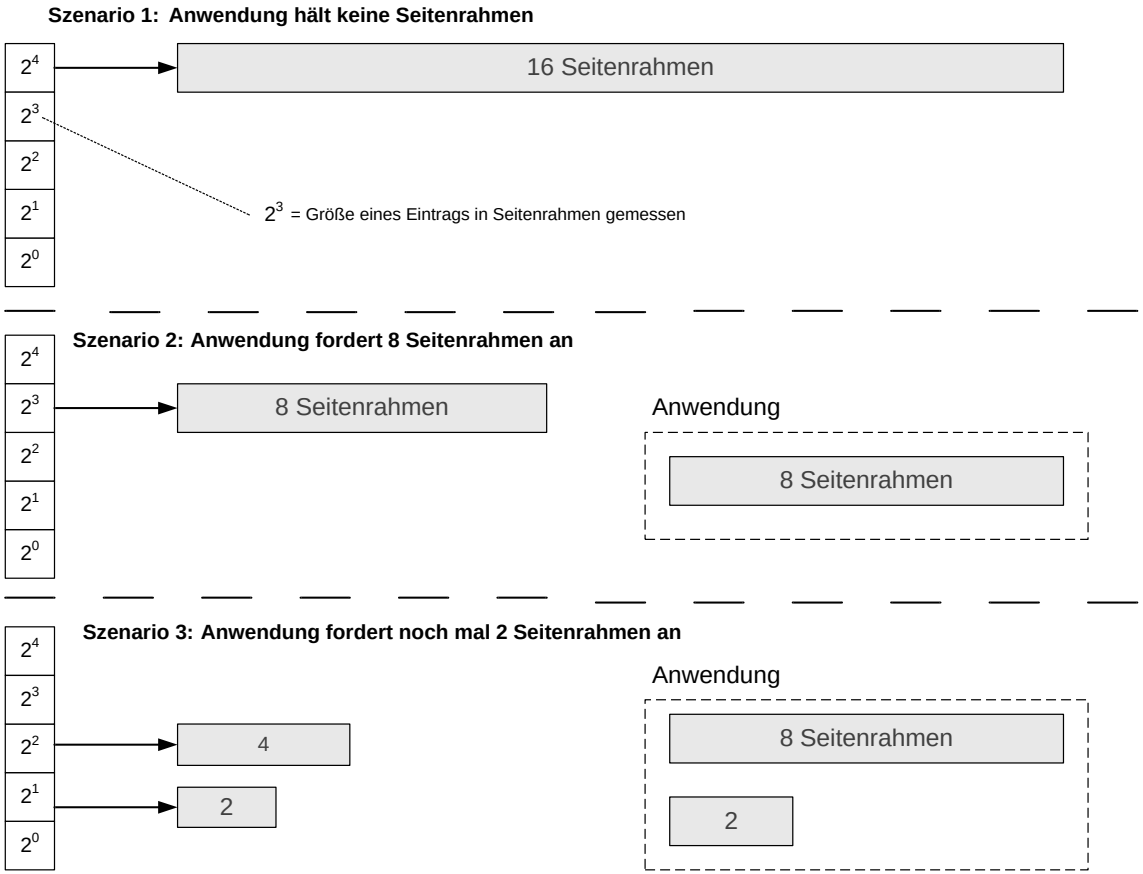
Bit Map 1110000000011111111000001111111111000100...

Speicherbelegungstrategien: Suche nach freien Seiten

- Vergabestrategien:
  - **Sequentielle Suche**, erster geeigneter Bereich wird vergeben (First-Fit)
  - **Optimale Suche** nach dem passendsten Bereich, um Fragmentierung möglichst zu vermeiden (Best-Fit)
  - **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

Prof. (emer.) Donald E. Knuth Stanford University

Speicherbelegungstrategien: Buddy-Technik (1)



- Reduziert externe Fragmentierung auf Kosten einer verstärkten internen Fragmentierung!

Überblick

1. Seitenersetzung und Verdrängung (Replacement)
2. Speicherbelegung und Vergabe (Placement)
3. **Entladen (Cleaning)**
4. Fallbeispiele: Windows, Unix, Linux

Entladestrategien (Cleaning)

- Legt den Zeitpunkt fest, wann eine modifizierte Seite auf die Paging-Area geschrieben wird
- Varianten:
  - **Demand-Cleaning**: Bei Bedarf
    - Vorteil: Seite lang im Hauptspeicher
    - Nachteil: Verzögerung bei Seitenwechsel
  - **Precleaning**: Präventives Zurückschreiben, wenn Zeit ist
    - Vorteil: Frames in der Regel verfügbar
  - **Page-Buffering**: Listen verwalten
    - Modified List: Wird zwischengepuffert
    - Unmodified List: Für Entladen freigegeben
    - Heute üblich (siehe Windows)

Überblick

1. Seitenersetzung und Verdrängung (Replacement)
2. Speicherbelegung und Vergabe (Placement)
3. Entladen (Cleaning)
4. **Fallbeispiele: Windows, Unix, Linux**

Speicherverwaltung unter Unix: Überblick

- Frühere Unix-Systeme bis zu BSD 3 nutzten ausschließlich **Swapping**
  - Ein Prozess namens **swapper** (daemon) mit PID 1 übernahm das Swapping bei bestimmten Ereignissen bzw. zyklisch im Abstand von mehreren Sekunden
  - Swapping → das ganze Programm wird auf Disk ausgelagert

Speicherverwaltung unter Unix: Überblick

- Ab BSD 3 wurde **Demand Paging** ergänzt, alle anderen Unix-Derivate (System V) haben es übernommen
- Ein sog. **Page Daemon** wurde eingeführt (PID 2)
- Im Page Daemon ist der Seitenersetzungsalgorithmus nach einem **Clock-Page** Algorithmus implementiert
- Heute: Variationen je nach Unix-Derivat

Speicherverwaltung unter Linux: Varianten

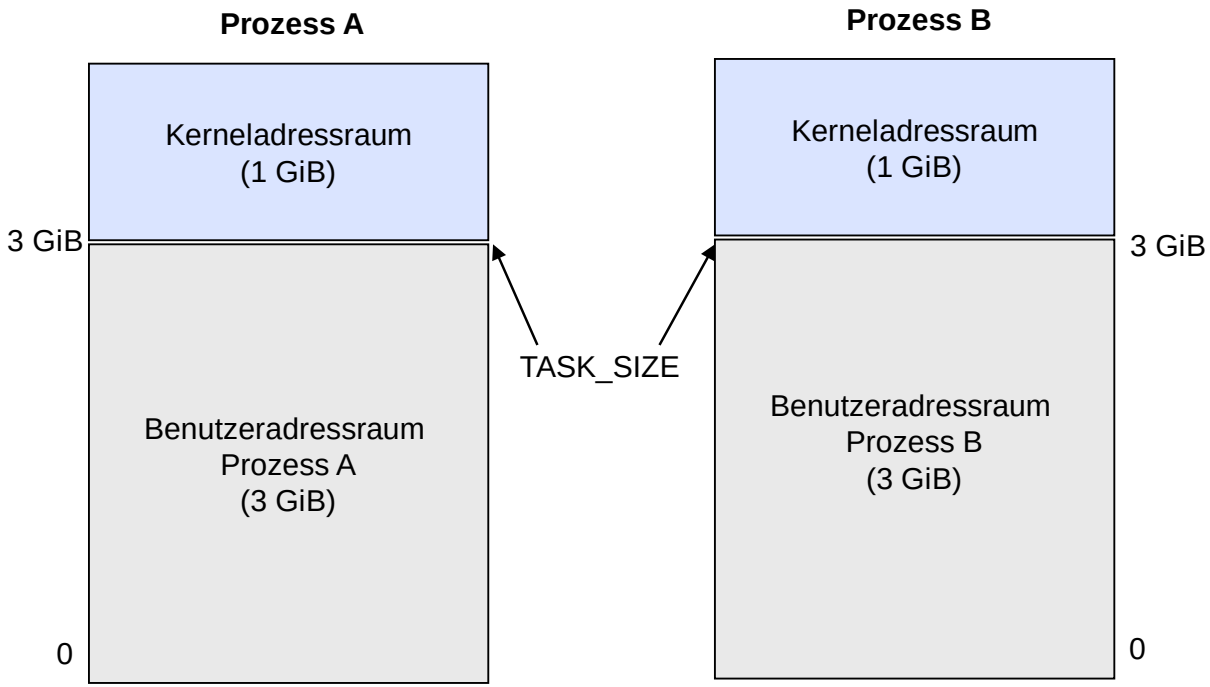
- Bei 32-Bit-Linux:
  - Virtuelle Adressen mit 32 Bit Länge, 1 GiB für den Kernel und die Seitentabellen, restliche 3 GiB für den User-Prozess
- Bei 64-Bit-Linux:
  - Bis zu 57-Bit-virtuelle Adressen und Adressraum der Größe 2^{57} (128 PB)
  - bedingt entsprechende Prozessoren, heute üblich: 2^{48} (256 TB)
- Adressumsetzung:
  - Linux verwendet **vierstufige** Seitentabellen, ab Version 4.12 **fünfstufige** Seitentabellen möglich
  - Evtl. Mapping auf zweistufige oder sonstige Seitentabelle, wenn Hardware es nicht kann

Speicherverwaltung unter Linux: Strategien

- **Fetch-Policy:**
  - Als Einlagerungsstrategie wird **Demand Paging** ohne Prepaging und ohne Working Set verwendet
- **Replacement- und Cleaning-Strategie:**
  - Replacement über eine Art Clock-Page-Algorithmus
  - Verwaltung mehrerer Listen mit Seitenrahmen (Page Buffering)
  - Mehrere Kernel-Threads zur Listenbearbeitung:
    - **kswapd** überprüft periodisch die Listen und lagert bei Bedarf um
    - **writeback** schreibt periodisch veränderte („dirty“) Seiten auf die Paging-Area
- **Placement-Policy:**
  - Speicherbelegung erfolgt über **Buddy-Technik**

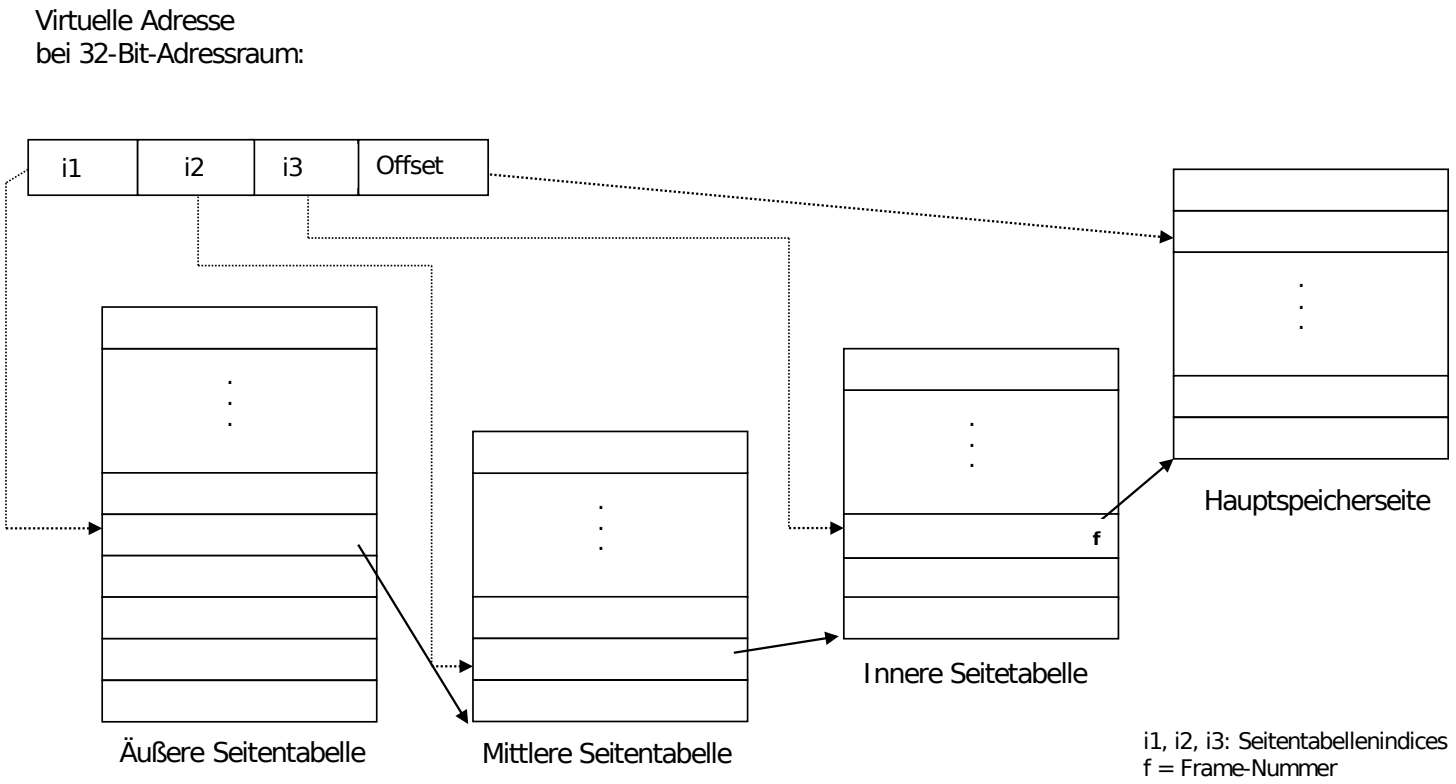
Speicherverwaltung unter Linux: Adressraumtopologie

- Adressraumbelegung bei 32-Bit-Architektur



Speicherverwaltung unter Linux (32-Bit): Adressumsetzung am Beispiel

- Virtuelle 32 Bit Adressen, hier: dreistufige Seitentabellen
- Abbildung bei Intel-Pentium auf zweistufiges Verfahren (Pentium unterstützt nur zwei Stufen)



Speicherverwaltung unter Windows: Überblick

- **Virtuelle Adressen** mit 32 Bits Länge, also **4 GiB** Adressraum, 2 davon für den User-Prozess und der Rest für den Kernel

  → linearer Adressraum ohne Segmentierung

- **Seitengröße** abhängig von Prozessorarchitektur:

Prozessorarchitektur	Größe der Small Page	Größe der Large Page
X86	4 KiB (12 Bit Offset)	4 MiB (22 Bit Offset)
x64 (AMD)	4 KiB (12 Bit Offset)	2 MiB (21 Bit Offset)
IA64 (Intel)	8 KiB (13 Bit Offset)	16 MiB (24 Bit Offset)

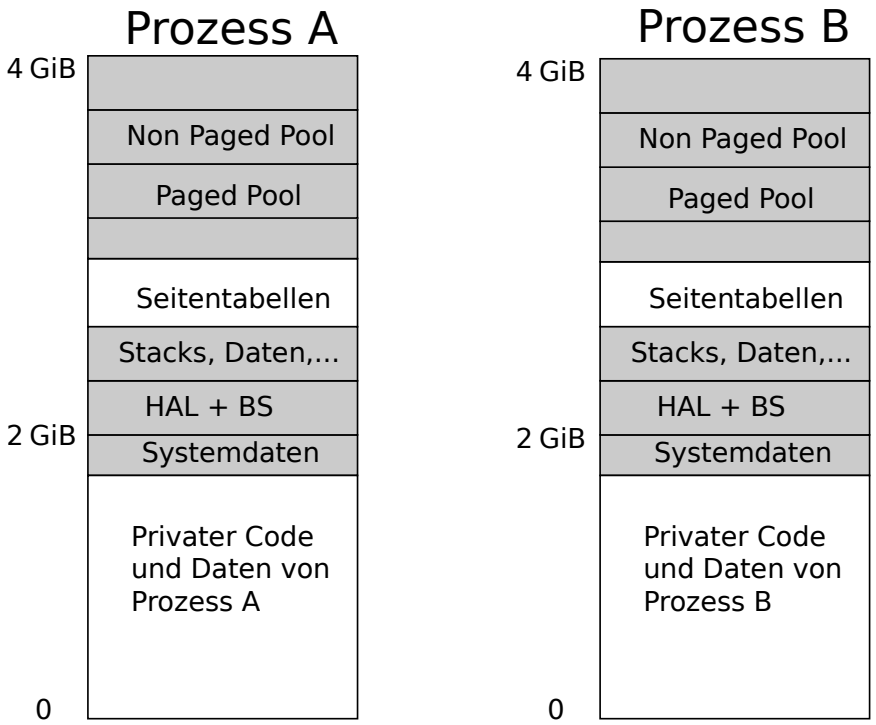
Hinweis: Large Pages werden von Grafikprozessoren genutzt

Speicherverwaltung unter Windows: Strategien

- **Fetch-Policy:**
  - Nutzung von **Demand Paging**
  - Ab Windows 2003 wird auch **Prepaging** verwendet
- **Replacement- and Cleaning-Policy:**
  - Kombination aus lokaler und globaler Ersetzungsstrategie
  - Eigenes Working-Set-Verfahren
  - FIFO bei Multiprozessormaschinen
  - Clock-Page bei Einprozessormaschinen
  - Mehrere Auslagerungslisten werden verwaltet
  - Mehrere Threads bearbeiten die Listen
- **Placement-Policy:**
  - Nicht näher erläutert

Speicherverwaltung unter Windows: Adressraumbelegung

- Aufbau eines virtuellen Adressraums
  (vgl. Tanenbaum)



HAL: Hardware Abstraction Layer

Speicherverwaltung unter Windows: Working-Sets (1)

- **Working Sets**
  - Jeder Prozess hat einen Working Set mit einer veränderbaren Größe (Minimum 50 Seiten, Maximum 345 Seiten, je nach vorhandenem Speicher)
  - Bei einem Seitenfehler wird nicht über den maximalen eigenen Working Set eines Prozesses eingelagert
  - Ausnahme:
    - Ein Prozess „paged“ stark und andere nicht, dann wird der „pagende“ Prozess erhöht, aber nicht mehr als die verfügbaren Seitenrahmen - 512, so dass immer noch ein paar Seitenrahmen frei bleiben

Speicherverwaltung unter Windows: Working-Sets (2)

- Ein zyklisch arbeitender **Working Set Manager Thread** versucht zusätzlich nach einem komplizierten Verfahren freie Seitenrahmen zu besorgen
- Ein Seitenrahmen (Frame) ist
  - entweder einem (oder mehreren) Working Set(s) zugeordnet
  - oder genau einer von vier Listen, in denen Windows freie Seitenrahmen verwaltet

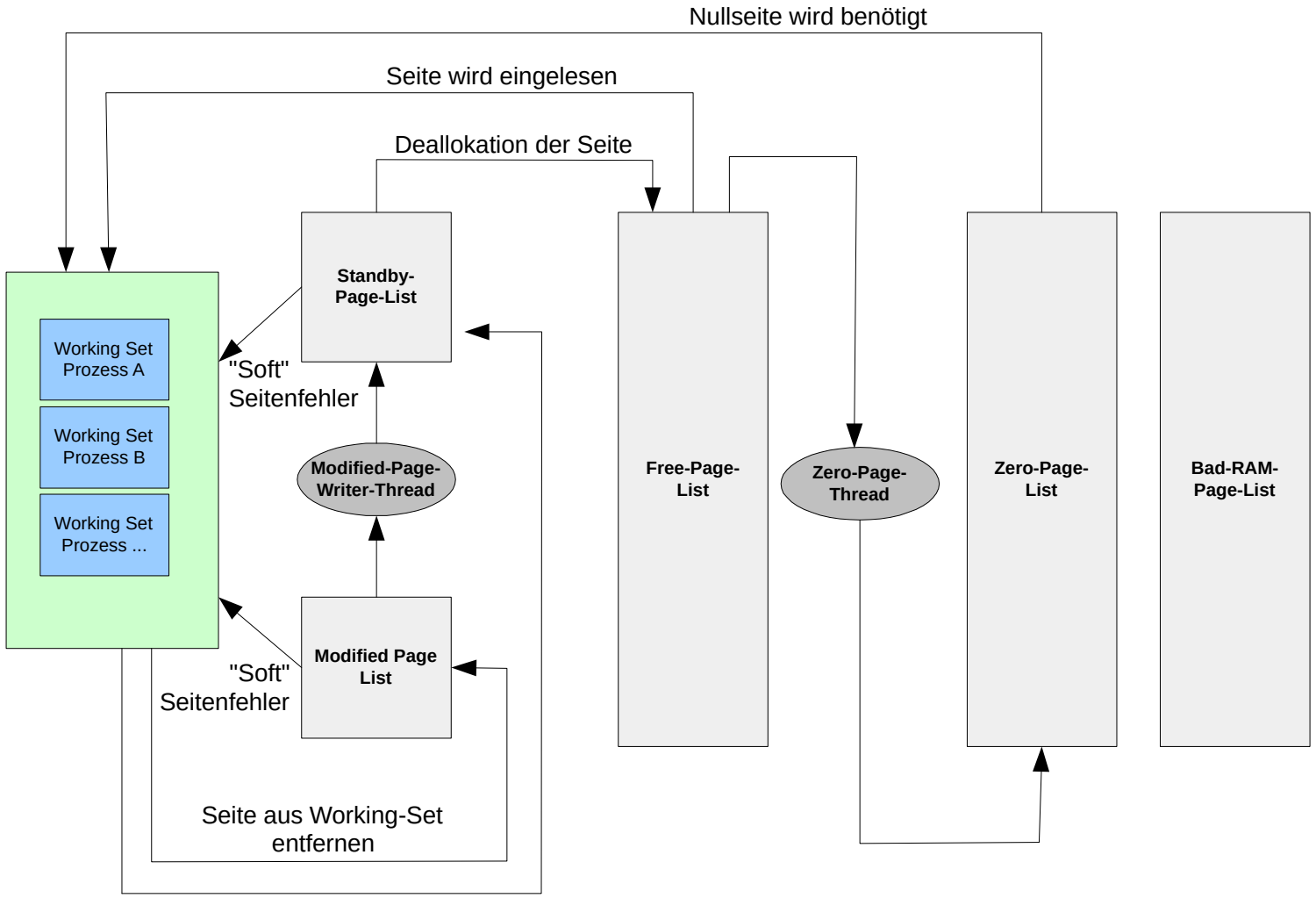
Speicherverwaltung unter Windows: Page Buffering - Listenverwaltung

- Die Listen im Einzelnen:
  - **Modified-Page-List**
    - Seiten, die bereits für die Seitenersetzung ausgewählt wurden, aber noch nicht ausgelagert wurden und auch dem nutzenden Prozess noch zugeordnet sind
  - **Standby-Page-List**
    - Wie modified page list, mit dem Unterschied dass sie „clean“ sind, also eine gültige Kopie auf der Paging Area haben
  - **Free-Page-List**
    - Frames, die bereits „clean“ sind und keinem Prozess mehr zugeordnet sind
  - **Zero-Page-List**
    - Wie die free page list und zusätzlich mit Nullen initialisiert
  - Weitere Liste hält defekte Speicherseiten (Bad-RAM-Page-List)

Speicherverwaltung unter Windows: Spezielle Systemthreads

- Einige Threads arbeiten an der Verwaltung dieser Listen mit
  - **Swapper-Thread:**
    - Läuft alle paar Sek., sucht nach Prozessen, die schon länger nichts tun (idle) und legt deren Frames in die Modified- oder Standby-Page-List
  - **Modified-Page-Writer-Thread**:
    - Laufen periodisch und sorgen für genügend saubere Seiten durch Umschichtung von der Modified-Page-List in die Standby-Page-List (vorher wird auf Platte gesichert)
  - **Zero-Page-Thread**:
    - Läuft mit niedriger Priorität, löscht Frames aus der Free-Page-List und legt sie in die Zero-Page-List

Speicherverwaltung unter Windows: Zusammenspiel von Threads und Listen



Zusammenfassung

- ✓ Seitenersetzung und Verdrängung (Replacement)
- ✓ Speicherbelegung und Vergabe (Placement)
- ✓ Entladen (Cleaning)
- ✓ Fallbeispiele: Windows, Unix, Linux

Gesamtüberblick

- ✓ Einführung in Computersysteme
- ✓ Entwicklung von Betriebssystemen
- ✓ Architekturansätze
- ✓ Interruptverarbeitung in Betriebssystemen
- ✓ Prozesse und Threads
- ✓ CPU-Scheduling
- ✓ Synchronisation und Kommunikation
- ✓ Speicherverwaltung
9. Geräte- und Dateiverwaltung
10. Betriebssystemvirtualisierung