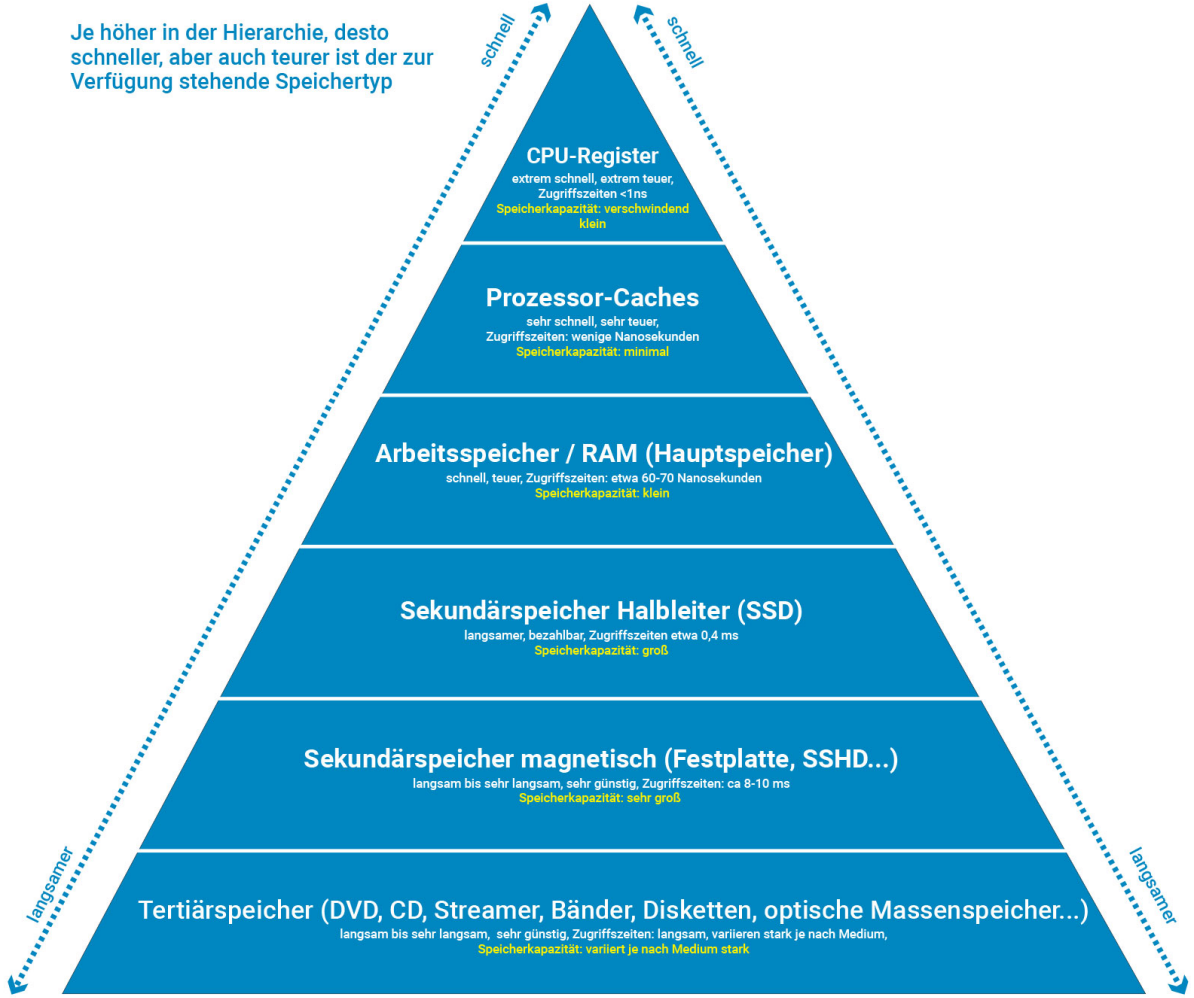
Speicherverwaltung

Margit Weber

Speicherhierarchie



Memory Management/Speicherverwaltung

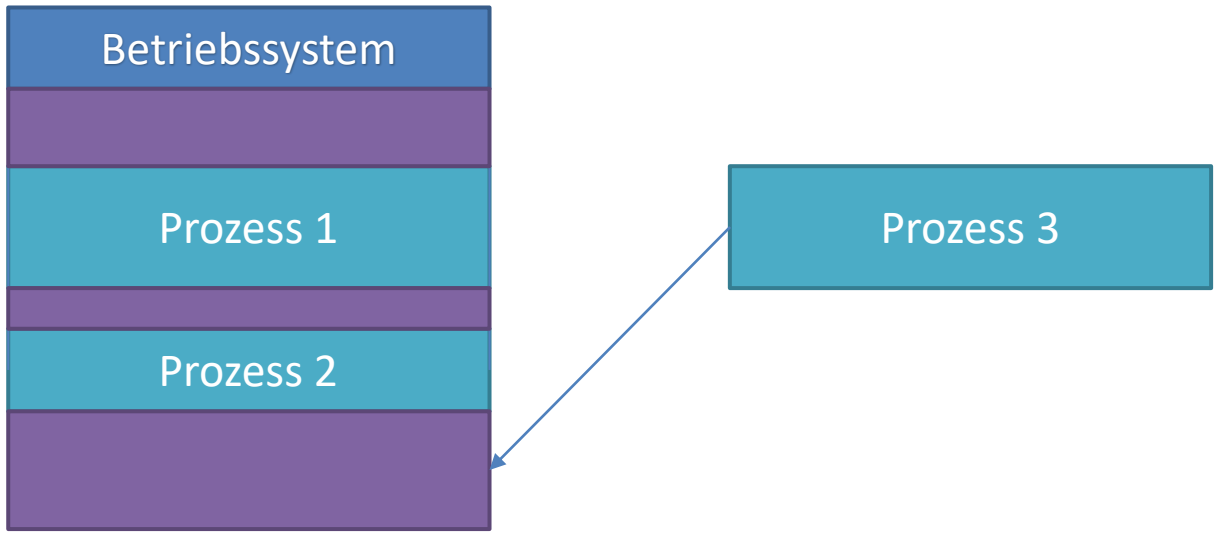
- Eine der zentralen Aufgaben eines Betriebssystems
- Organisiert die Zuweisung von Hauptspeicher (main memory, RAM) an Prozesse
- Stellt sicher, dass mehrere unabhängige Programme gleichzeitig im Speicher sein können
- Schutz vor unbefugtem Zugriff

Der Bedarf an Arbeitsspeicher ist oft größer als der verfügbare physische Speicher

- Moderne Methoden gehen davon aus, dass zu einem beliebigen Zeitpunkt nur auf wenige Speicherplätze zugegriffen wird
- Andere stehen für einen späteren Zeitpunkt bereit (und werden evtl. nie zugegriffen)
  - Speicherverwaltung legt fest welche Programmabschnitte geladen werden müssen
  - Welche bleiben im Hintergrundspeicher

Wozu Speicherverwaltung?

- Mehrere Prozesse benötigen Hauptspeicher
  - Prozesse liegen an verschiedenen Stellen im Hauptspeicher
  - Schutzbedürfnis des Betriebssystem und der Prozesse untereinander
  - Speicher reicht nicht für alle Prozesse
  - Verschieben von Prozessen zwischen Haupt- und Hintergrundspeicher



Wer benötigt den verfügbaren Speicher?

- Benutzerprogramme
  - Programmbefehle (Text/Code)
  - Programmdaten (Data)
  - Dynamische Speicheranforderungen (Stack/Heap)
- Betriebssystem
  - Betriebssystemcode und Betriebssystemdaten
  - Prozesskontrollblöcke
  - Datenpuffer für Ein-/Ausgabe

→ Zuteilung des Speichers nötig

Anforderungen an die Speicherverwaltung

- Effektive Aufteilung und Verwaltung des Arbeitsspeichers für BS und Programme
- Ziel aus Betriebssystemsicht: Möglichst viele Prozesse im Speicher
- Anforderungen
  - Relocation
  - Protection
  - Sharing
  - Logical Organization
  - Physical Organization

Relocation

- Der Hauptspeicher eines Multiprogrammingsystems wird von den verschiedenen Prozessen gemeinsam genutzt. Hierfür ist eine Verwaltungsstruktur notwendig, z.B. in Form von Listen
- Programme werden zum Teil ausgelagert, und nur das Prozess Image der aktiven Prozesse muss im Hauptspeicher verfügbar sein

→ die Prozessorhardware und die Betriebssystem-Software müssen in der Lage sein, Referenzen im Programmcode in aktuelle physische Adressen umzuwandeln

Protection

- jeder Prozess muß gegen ungewollte Einmischungen oder Störungen anderer Prozesse geschützt werden, egal ob diese Eingriffe beabsichtigt oder unbeabsichtigt sind

→ fremde Prozesse sollten nicht in der Lage sein, die gespeicherten Informationen eines anderen Prozesses zu lesen oder zu modifizieren, wenn sie dafür keine Erlaubnis haben

Sharing

- Die Speicherverwaltung muss verschiedenen Prozessen den Zugriff auf einen gemeinsam genutzten Speicherbereich ermöglichen. Falls mehrere Prozesse das gleiche Programm ausführen, so ist es z.B. vorteilhaft, dass jeder Prozess auf dieselbe Version des Programms zugreift anstatt eine eigene Kopie zu nutzen

Logical Organization

- Der Hauptspeicher (und in der Regel auch der Sekundärspeicher) ist immer als linearer (1-dimensionaler) Adressraum organisiert, der aus einer Folge von Bits (Bytes oder Wörtern) besteht
- Programme werden jedoch modular geschrieben, d.h. diese Darstellung im Speicher entspricht nicht der Art und Weise, in der Programme geschrieben werden

→ Eine modulare Organisation ist besser zu verwalten und sollte von der Speicherverwaltung unterstützt werden

Physical Organization

- Umfasst die Einteilung in Hauptspeicher (schneller Zugriff, hohe Kosten) und Hintergrundspeicher (langsamer und billiger). Nur der Hintergrundspeicher ist in der Lage, Daten permanent zu speichern. Die Daten werden jedoch vom Prozessor im Hauptspeicher benötigt

→ Die eigentliche Aufgabe der Speicherverwaltung ist der Transport von Daten zwischen Haupt- und dem Hintergrundspeicher

Adresse/Adressraum

Adressen:

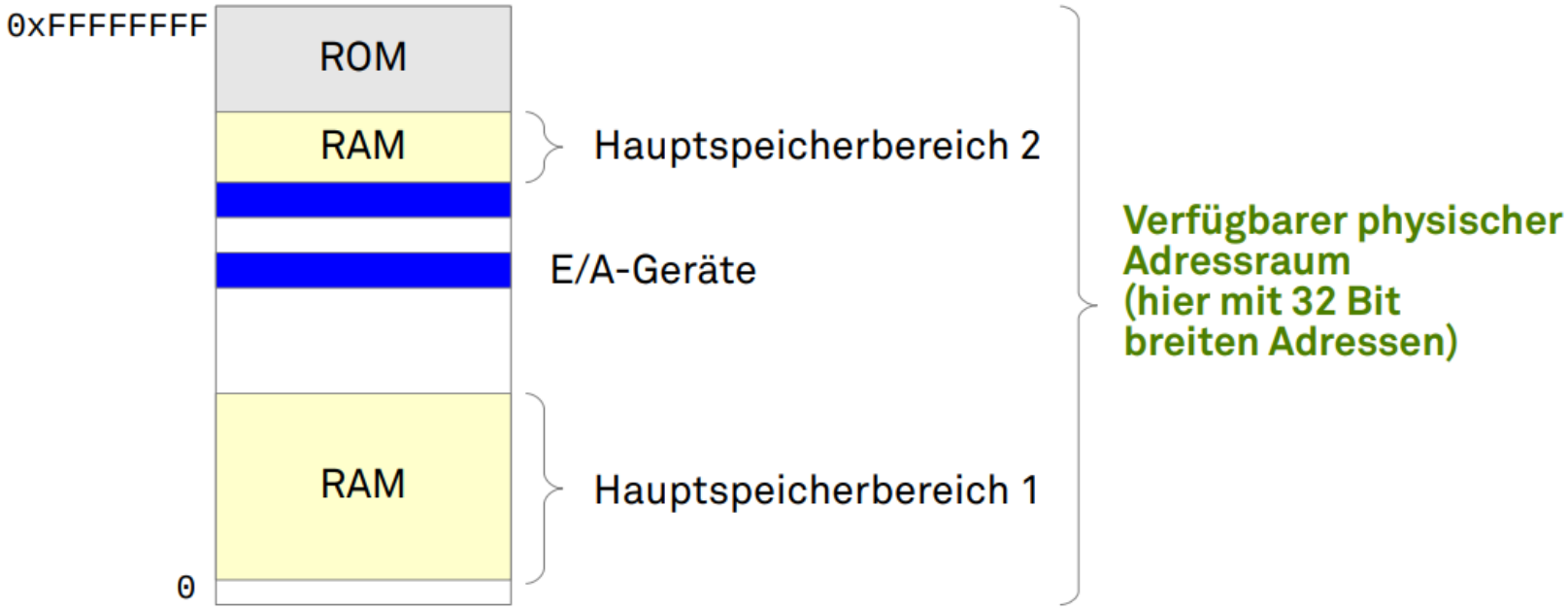
- Der Hauptspeicher ist nach Speicherstellen geordnet.
- Üblicherweise ist ein Byte (8 Bit) die kleinste adressierbare Einheit.
- Jedes Byte im Speicher wird daher mit einer Adresse versehen.
- Adressen werden meist hexadezimal (z. B. 0x40000000) notiert.
- Wenn man im Prozessor 32 Adressbits zur Verfügung hat, kann man maximal
  - 2^{32} Bytes, also 4 GiB, adressieren und der
  - komplette Adressbereich reicht von 0x00000000 bis 0xFFFFFFFF
  - Wie sieht das bei einem Prozessor/System mit 64 Adressbits aus? Und wieviel davon unser BS nutzen

Adressraum:

- Man spricht von einem Adressraum, wenn man die Menge aller möglichen Adressen meint. Bei einem 4-GiB-Adressraum sind dies die Adressen {0, 1, 2, … , 2^{32}-1}.
- Die Adressen eines Adressraums werden also durchnummeriert, wobei 0 die kleinste Adresse ist.

Adressen und Adressräume

- Ein **Adressraum** ist die Menge aller adressierbaren Adressen
  - 32-Bit-Adressen → {0, 1, 2, … , 2^{32} - 1}

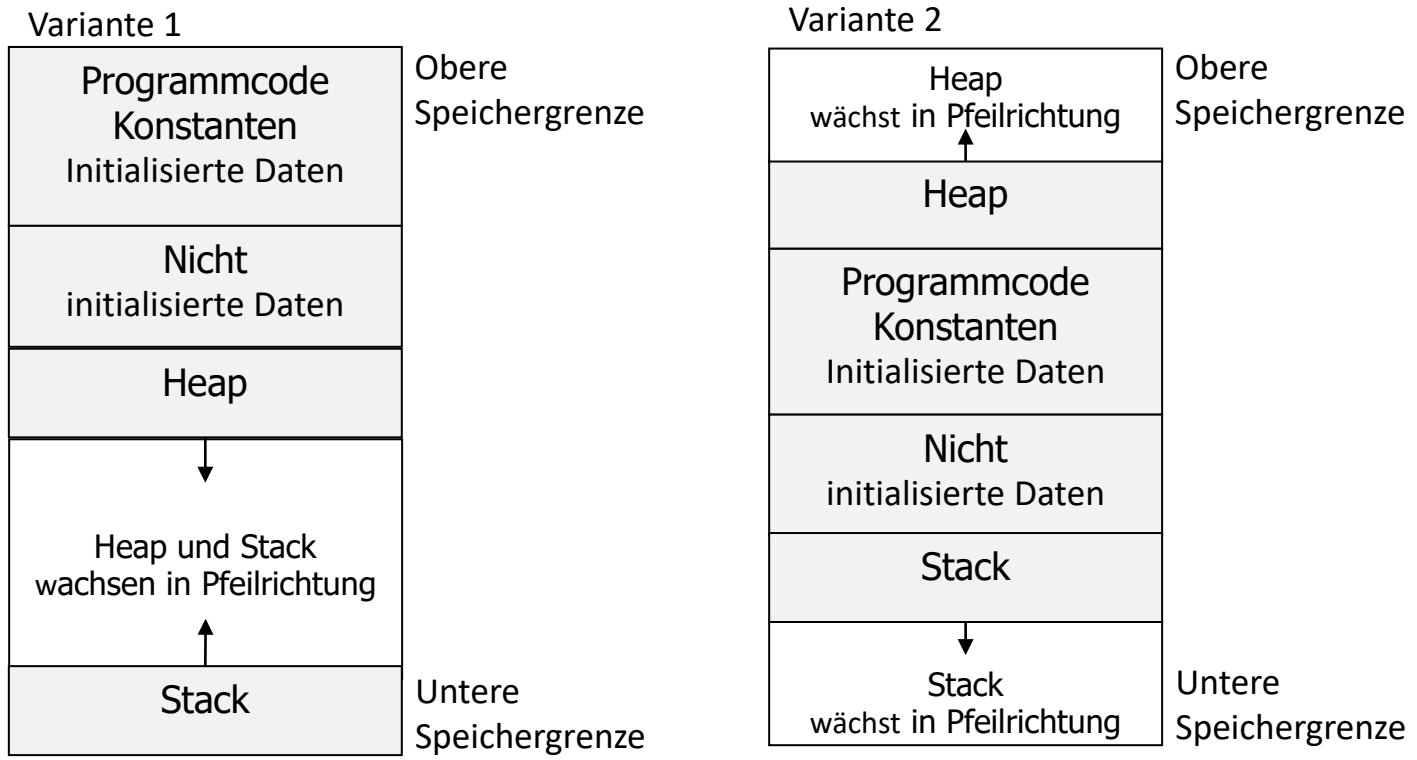


Speicherlandkarte (*Memory Map*)
eines fiktiven 32-Bit-Systems

htl wiener neustadt
Höhere technische Bundes- Lehr- und Versuchsanstalt Wiener Neustadt

Adressraumbelegung

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



Partitionierung des Speichers

- Prozesse werden als ganzes im Hauptspeicher abgelegt
- Statische Partitionierung
- Dynamische Partitionierung
- Buddy System

Statische Partitionierung

- Feste Bereiche für Betriebssystem und Benutzerprogramme
- Einteilung des Speichers in feste Anzahl von Partitionen

Alle Partitionen mit gleicher Länge

Betriebssystem
8 MB
8 MB
8 MB
8 MB
8 MB
8 MB

Partitionen mit unterschiedlicher Länge

Betriebssystem
8 MB
4 MB
4 MB
8 MB
10 MB
14 MB

Statische Partitionierung

- Probleme:
  - Grad des Mehrprogrammbetriebs begrenzt
  - Begrenzung anderer Ressourcen (z.B. Bandbreite bei Ein-/Ausgabe wegen zu kleiner Puffer)
  - Ungenutzter Speicher des Betriebssystems kann von Anwenderprogrammen nicht genutzt werden und umgekehrt
  - Es können nur Prozesse geladen werden die kleiner gleich einer Partition sind können geladen werden
  - Auslagern von Prozessen wenn alle Partitionen belegt sind

→ Dynamische Partitionierung einsetzen

Dynamische Partitionierung

- Segmente
  - Zusammenhängender Speicherbereich (Bereich mit aufeinanderfolgenden Adressen)
- Allokation (Belegung) und Freigabe von Segmenten
- Ein Anwendungsprogramm besitzt üblicherweise folgende Segmenten
  - Textsegment
  - Datensegment
  - Stapelsegment (lokale Variablen, Parameter, Rücksprungadressen, …)
- Suche nach geeigneten Speicherbereichen zur Zuteilung
  - Insbesondere beim Programmstart

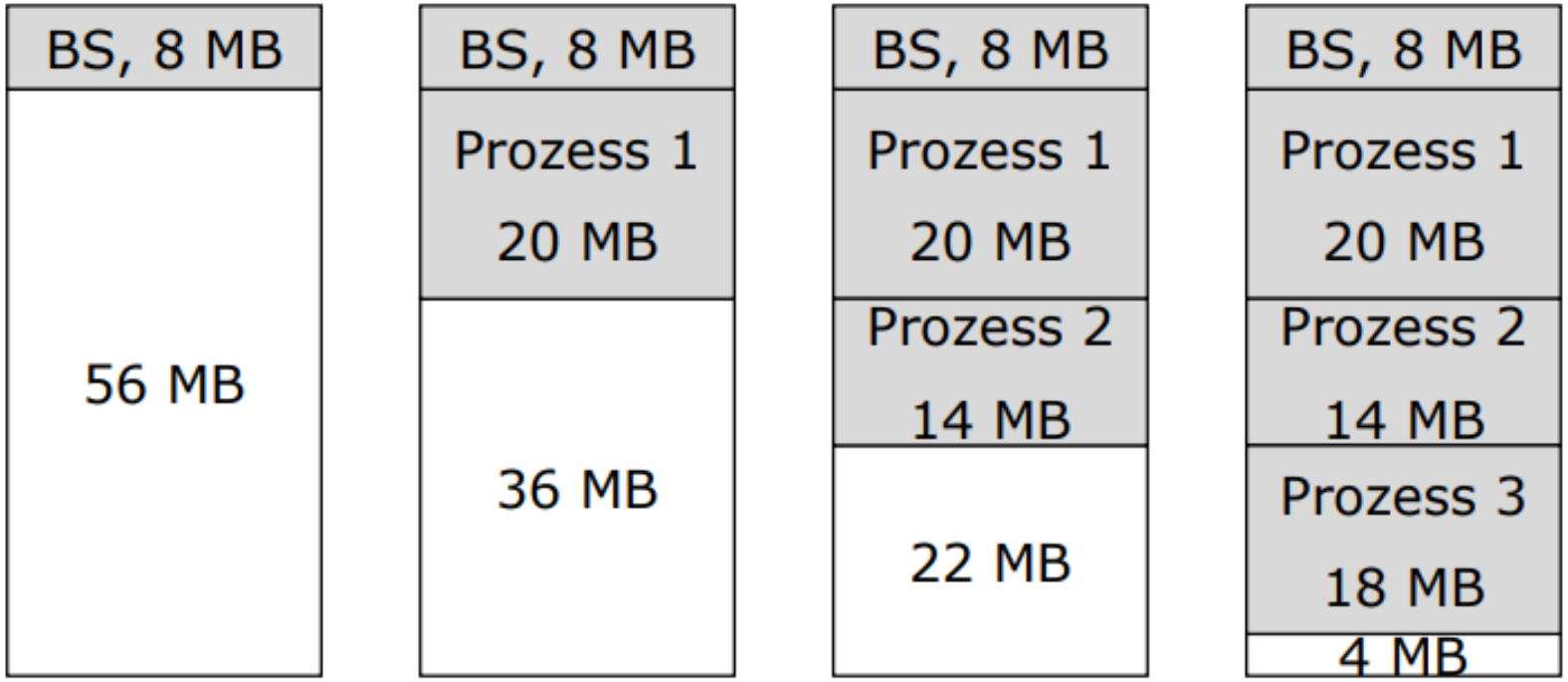
→Platzierungsstrategien nötig (Besonders wichtig: Freispeicherverwaltung)

Dynamische Partitionierung

- Einteilung des Speichers in Partitionen
  - variable Länge
  - variable Anzahl
- Prozesse erhalten exakt passende Speicherbereiche
- Aber: Ein- und Auslagern führt zu externer Fragmentierung, Vielzahl kleiner Lücken, Speicherauslastung nimmt ab
- Lücken können zu größeren zusammengefasst werden (Speicherverdichtung), wird normalerweise nicht gemacht (großer Aufwand)

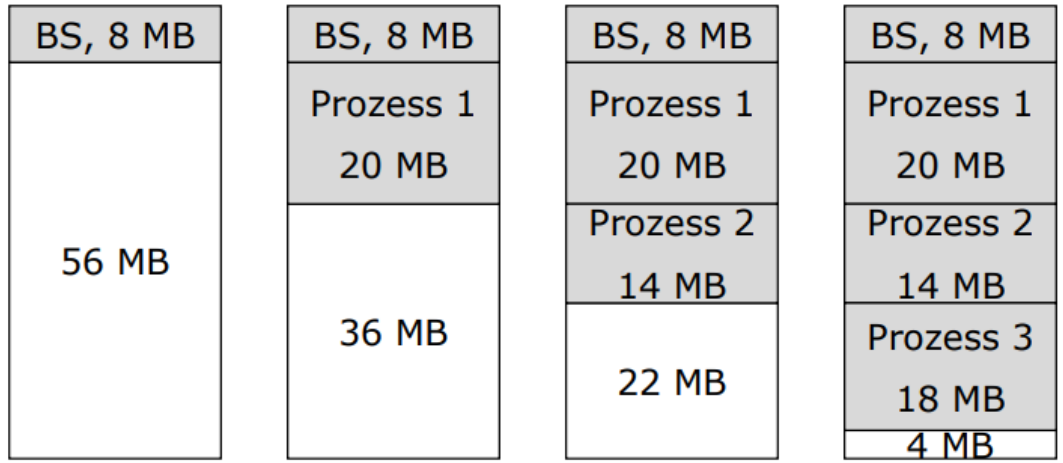
htl wiener neustadt
Höhere technische Bundes- Lehr- und Versuchsanstalt Wiener Neustadt

Dynamische Partitionierung

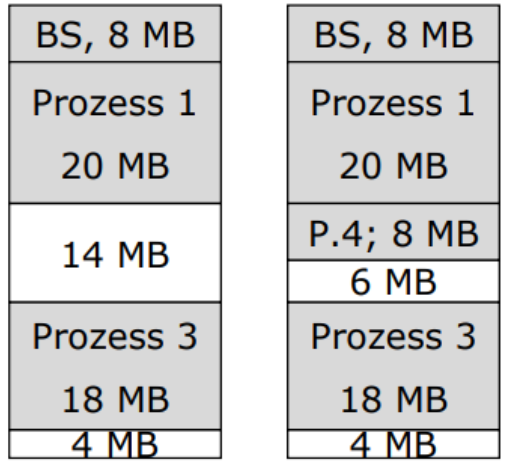


Anforderung:
Prozess 4
braucht 8 MB

Dynamische Partitionierung



Anforderung: Prozess 4 braucht 8 MB

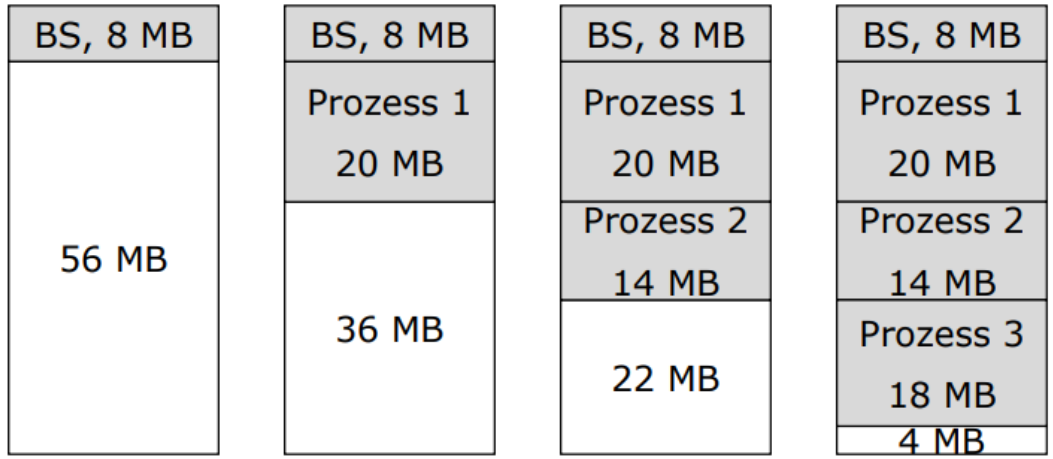


Prozess 2 wird ausgelagert

Genügend Platz für Prozess 4, aber Lücke entsteht

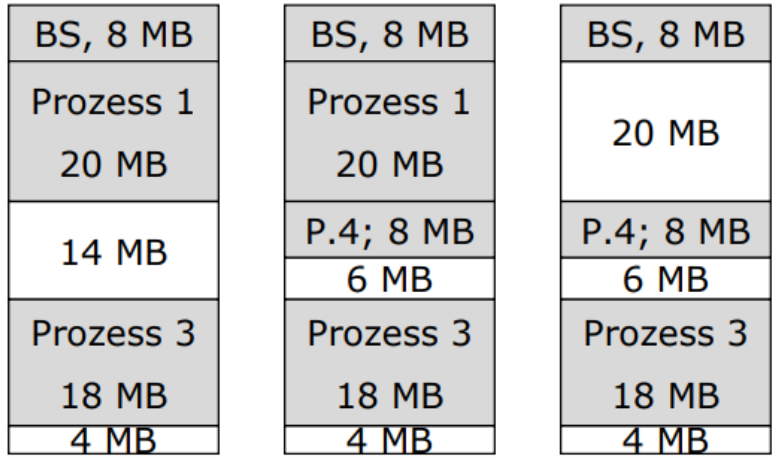
Annahme: Kein Prozess im Hauptspeicher bereit, aber ausgelagerter Prozess 2 (14MB) bereit

Dynamische Partitionierung



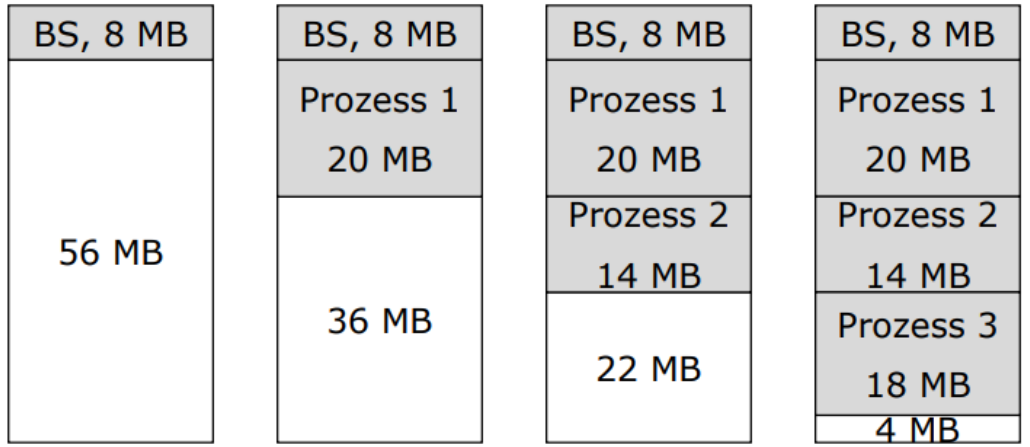
Anforderung:
Prozess 4
braucht 8 MB

Prozess 2 wird ausgelagert



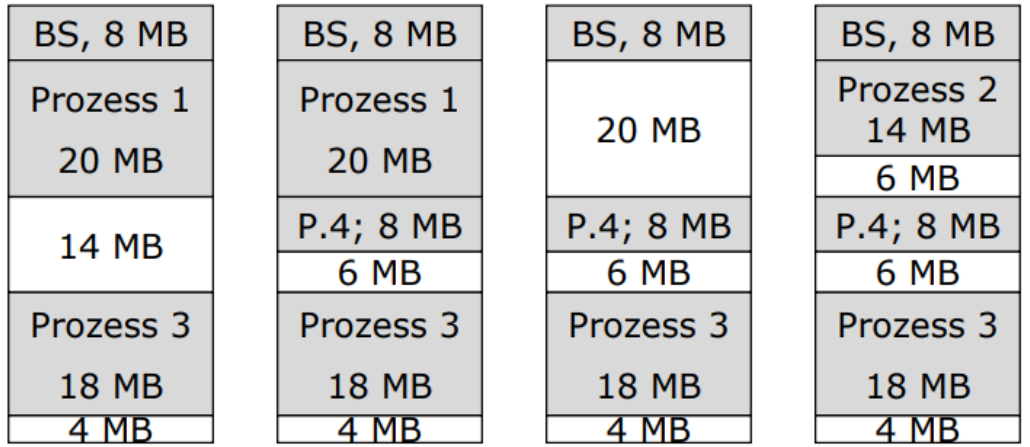
Da nicht genügend Platz für Prozess 2: Prozess 1 wird ausgelagert

Dynamische Partitionierung



Anforderung:
Prozess 4
braucht 8 MB

Prozess 2 wird
ausgelagert



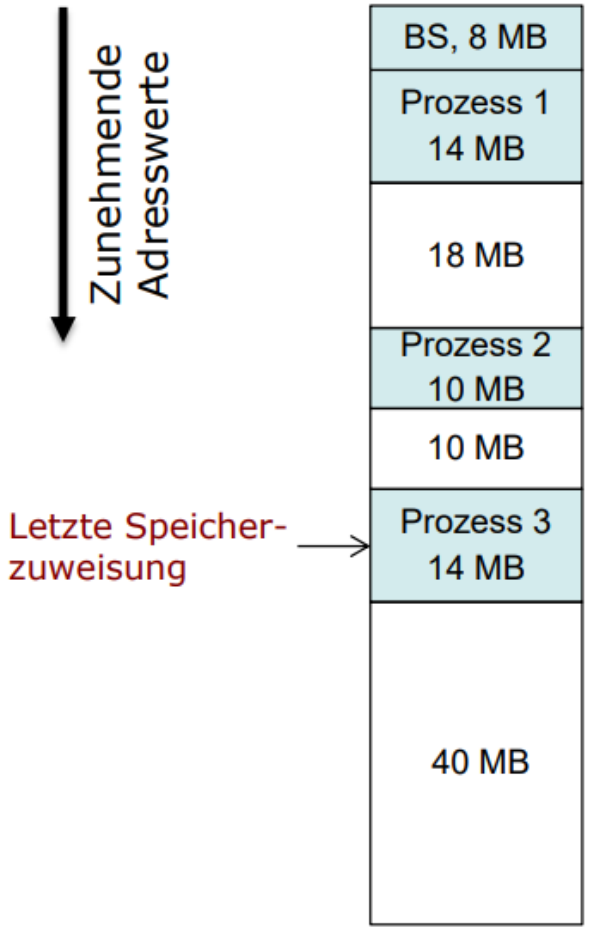
Prozess 2 wird
wieder
eingelagert

Dynamische Partitionierung

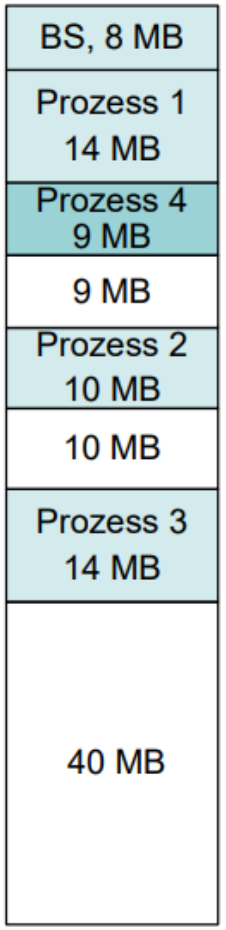
Defragmentierung möglich, aber

- Verschiebung aufwändig: Speicherzuteilungsstrategie wichtig
- Speicherverdichtung nur erfolgreich, wenn dynamische Relokation möglich
- Speicherzuteilungsalgorithmen:
  - **Best Fit:** Suche kleinsten Block, der ausreicht
  - **First Fit:** Suche beginnend mit Speicheranfang bis ausreichend großer Block gefunden
  - **Next Fit:** Suche beginnend mit der Stelle der letzten Speicherzuweisung

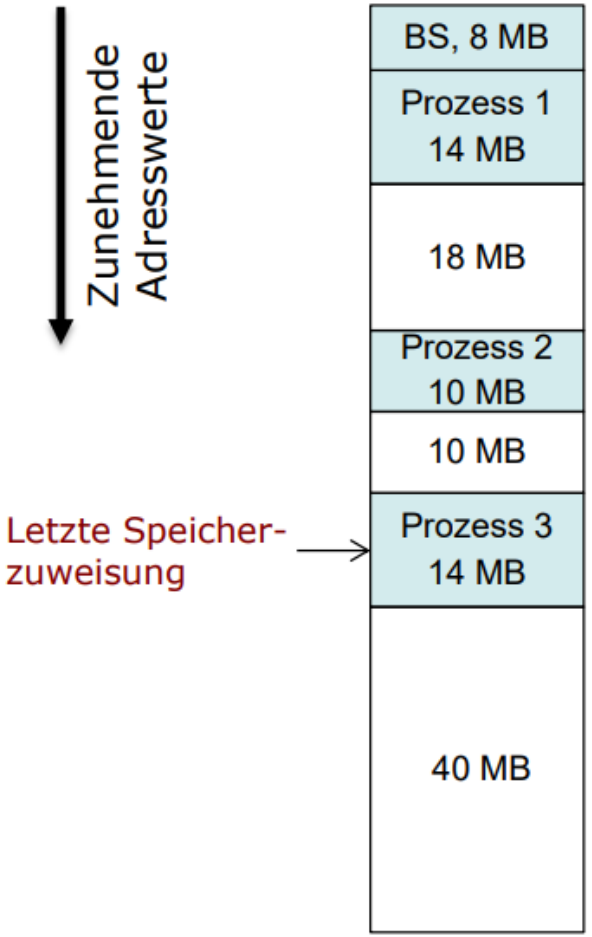
First Fit



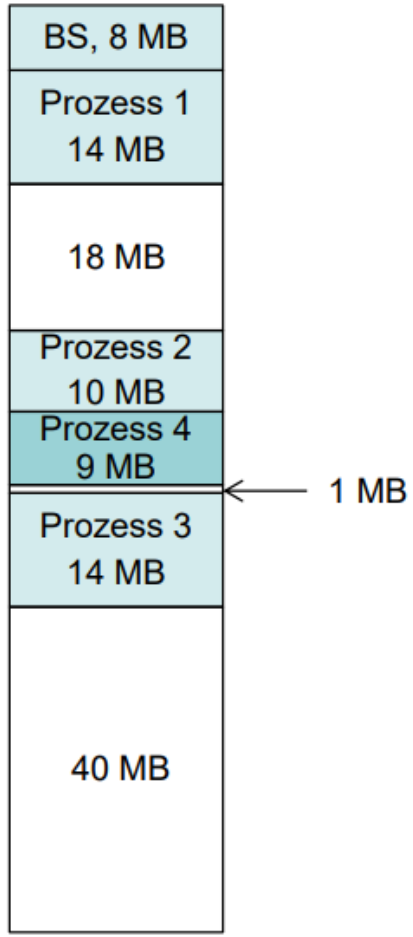
Anforderung:
Prozess 4
braucht 9 MB



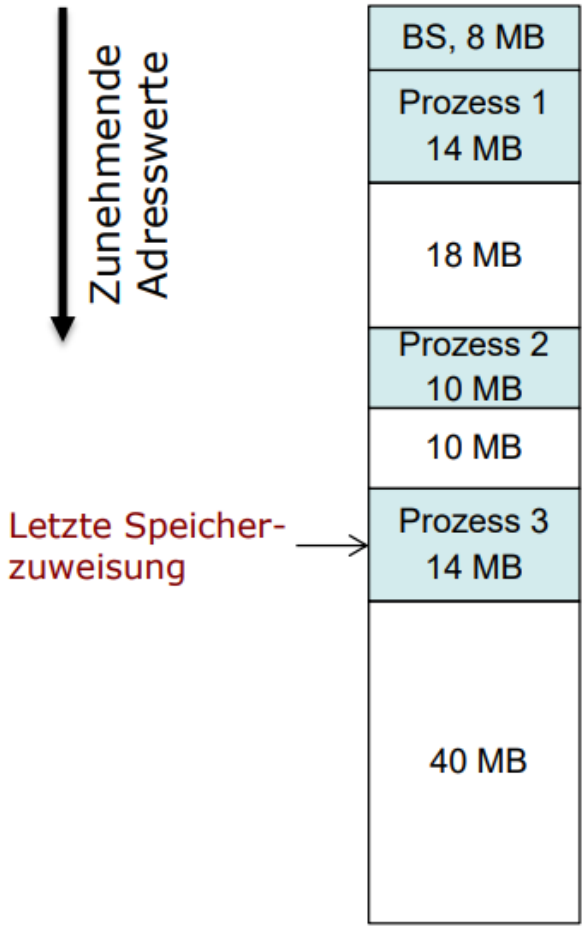
Best Fit



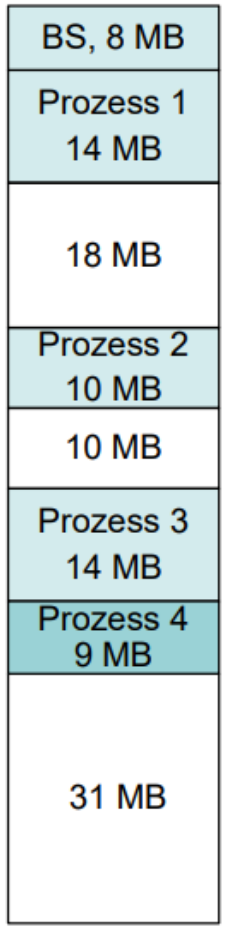
Anforderung:
Prozess 4
braucht 9 MB



Next Fit



Anforderung:
Prozess 4
braucht 9 MB



Wie gut schlagen sich die drei Algorithmen?

- Im Schnitt ist First Fit am besten!
- Next Fit: Etwas schlechter
  - Typischer Effekt: Schnelle Fragmentierung des größten freien Speicherblocks am Ende des Speichers
- Best Fit: Am schlechtesten
  - Schnell eine Reihe von sehr kleinen Fragmenten, Defragmentierung nötig
  - Außerdem: Suche braucht Zeit

Fragmentierung/Verschnitt

- Interne Fragmentierung
  - Verschwendung von Speicherplatz innerhalb einer Partition
  - Daten und Programme füllen Partition nicht aus (z.B.: Statische Partitionierung)
  - Beispiel: Speicherblöcke mit Größe: 4096 Bytes → Speicher mit 2568 wird benötigt → ein großer Teil bleibt frei
  - Reduktion des Problems mit Speicherblöcken variabler Größe
- Externe Fragmentierung
  - Tritt bei Dynamische Partitionierung auf (First Fit, Best Fit usw)
  - Zerstückelung des Speicherbereichs außerhalb der Partitionen
  - Prozesse werden aus dem Speicher geladen bzw. entfernt → viele Lücken im Speicher → Summe alle Lücken würde für den Bedarf des Prozesses ausreichen → aber auf viele kleine Lücken verteilt → daher kann dieser nicht benutzt werden
    - Paging und Segmentierung als Lösungen

Buddy-Systeme

Ein Speicher der Größe 2^N wird in Blöcke der Größe 2^K eingeteilt. Die kleinste Speichergröße wird auf 2^L, die größte auf 2^U festgelegt. Für unterschiedliche K muss in jedem Fall gelten: $L \leq K \leq U$, Im allgemeinen gilt: $U = N$.
($\rightarrow$L/U = Lower/Upper Bound (Grenze). L ist geeignet zu wählen, so dass hinreichend kleine Buddies möglich sind, deren Verwaltung noch sinnvoll ist.)

Anfangszustand: Der verfügbare Speicherplatz wird als einzelner Block der Größe 2^U betrachtet.

Algorithmus: Kommt nun eine Speicheranforderung S, so wird die Bedingung $2^{U-1} < S \leq 2^U$ geprüft und in diesem Fall der gesamte Speicher der Anforderung zugewiesen.
Ist $S > 2^U$, so kann die Anforderung nicht erfüllt werden.
In der Regel wird jedoch $S \leq 2^{U-1}$ gelten.
$\Rightarrow$Der Speicherblock wird in zwei Buddies der Größe 2^{U-1} aufgespaltet und für den ersten Block die Bedingung $2^{U-2} < S \leq 2^{U-1}$ geprüft. Und so weiter.

Das Buddy-System verwaltet pro Zweierpotenz eine Liste der freien Speicherplätze, d.h. der Lücken der Größen 2^i.

Buddy-System

- Beispiel: Speicher der Größe 1 GiB
- Folge von Anforderungen und Freigaben:
  - A fordert 100 MiB an
  - B fordert 240 MiB an
  - C fordert 64 MiB an
  - D fordert 256 MiB an
  - Freigabe B
  - Freigabe A
  - E fordert 75 MiB an
  - Freigabe C
  - Freigabe E
  - Freigabe D

Annahme:
Obergrenze der Blockgröße: 1GB
Untergrenze der Blockgröße: 64 MB

Nachteile Partitionierung

- Statische Partitionierung:
  - Anzahl von Prozessen im Speicher beschränkt
  - Interne Fragmentierung
- Dynamische Partitionierung:
  - Schwierigere Verwaltung
  - Externe Fragmentierung
- Buddy-System (Halbierungsverfahren 2er-Potenzen):
  - Kompromiss zwischen statischer und dynamischer Partitionierung
  - Es tritt sowohl interner als auch externer Verschnitt auf

Adressbindung und Relokation

- **Problem: Maschinenbefehle benutzen Adressen**
  - z.B. ein Sprungbefehl in ein Unterprogramm oder ein Ladebefehl für eine Variable aus dem Datensegment
  - Es gibt verschiedene Möglichkeiten, die Adressbindung zwischen dem Befehl und seinem Operanden herzustellen …
- **Absolutes Binden (Compile/Link Time)**
  - Adressen stehen fest
  - Programm kann nur an bestimmter Speicherstelle korrekt ablaufen
- **Statisches Binden (Load Time)**
  - Beim Laden des Programms werden die absoluten Adressen angepasst (reloziert)

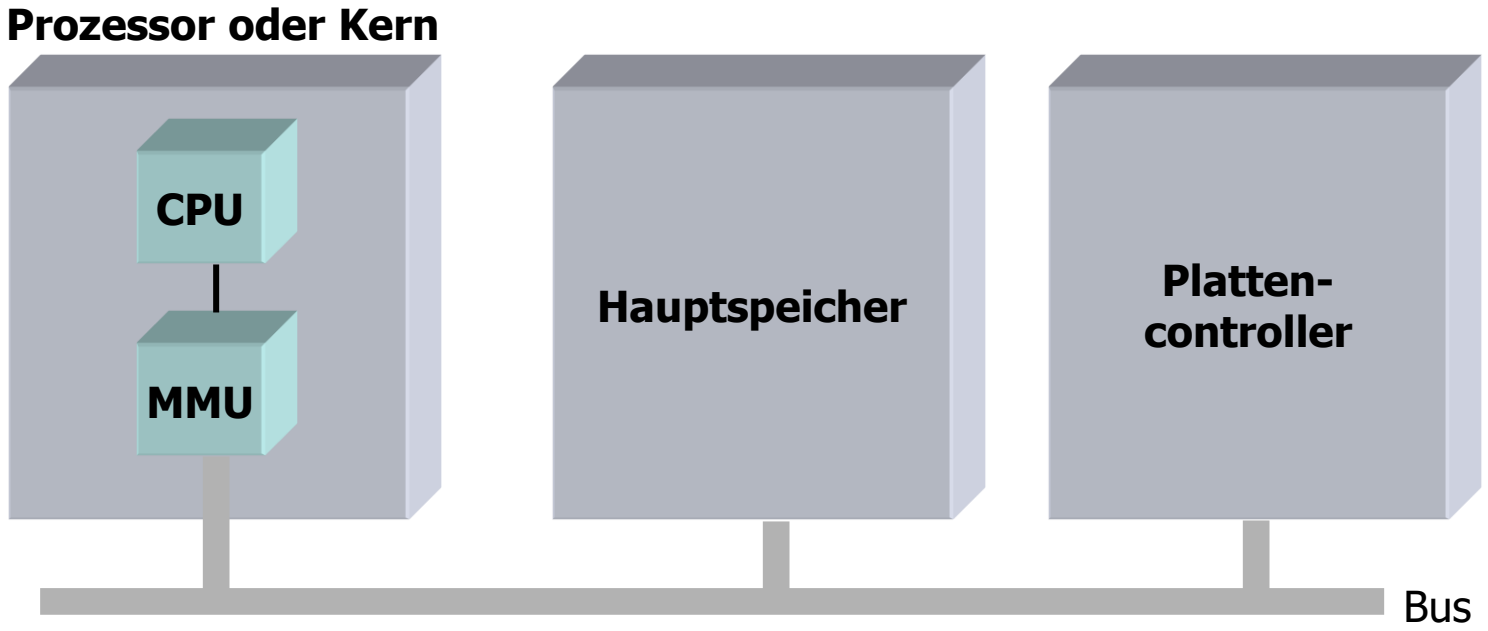
  ➔ Compiler/Assembler muss Relokationsinformation liefern
- **Dynamisches Binden (Execution Time)**
  - Der Code greift grundsätzlich nur indirekt auf Operanden zu
  - Das Programm kann jederzeit im Speicher verschoben werden

  ➔ Programme werden etwas größer und langsamer

Virtueller Speicher

Adressumsetzung: Hardwareunterstützung durch die MMU

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



Prinzip der Speicherverwaltung

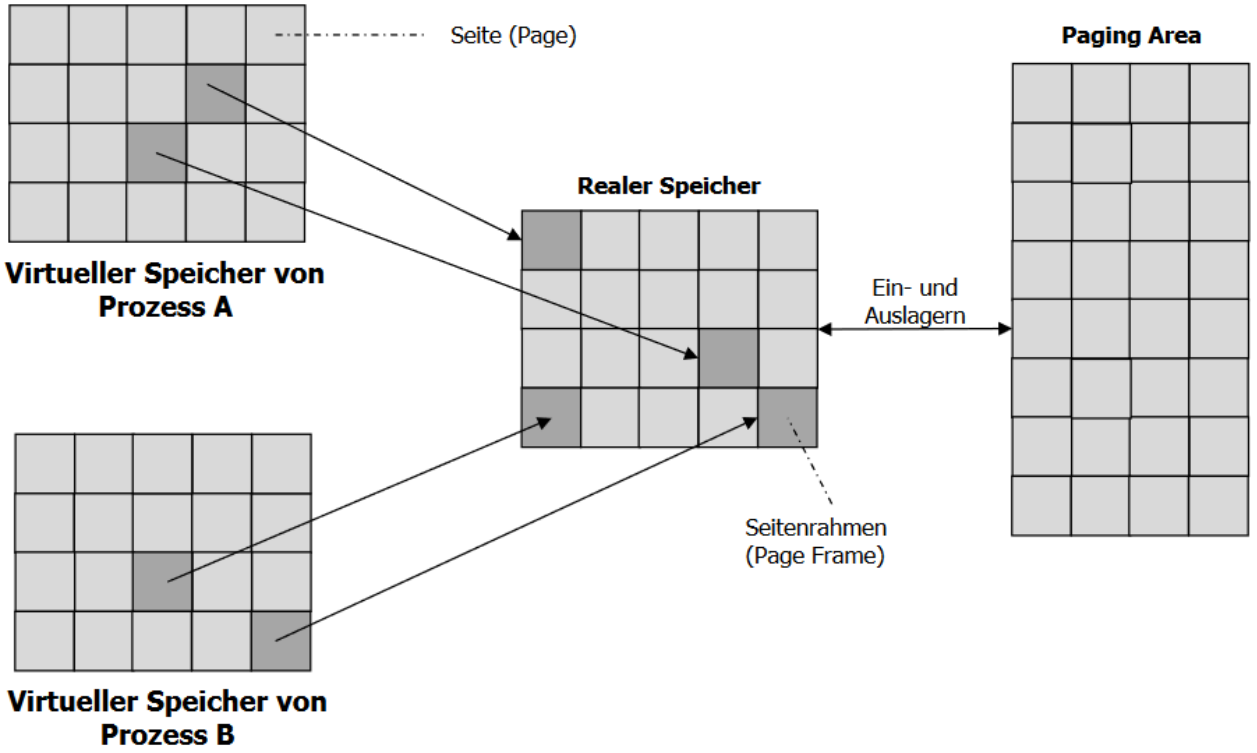
- Physischer Adressraum
  - Die Menge der im Arbeitsspeicher physisch vorhandenen Speicherplätze bildet den realen oder *physischen Adressraum* P
  - Speicherzellen, die eine einheitliche Größe besitzen (in der MI 8 Bit), linear angeordnet, d.h. jeder Zelle ist eindeutig eine Adresse zugeordnet, die mit 0 beginnen
- Logischer Adressraum
  - Dem physischen oder realen Adressraum steht ein virtueller oder *logischer Adressraum* L gegenüber
  - Wird benötigt, falls die Kapazität des physischen Arbeitsspeichers nicht ausreicht, um alle Daten zu speichern. Der so entstehende Kapazitätsengpaß soll durch die zusätzliche Nutzung des Hintergrundspeichers beseitigt werden
- **Die Aufgabe der Speicherverwaltung besteht in der Zuordnung von logischen zu physischen Adressen**, d.h. im Auffinden einer geeigneten Abbildung L → P

Zuordnung von logischen zu physischen Adressen

- Ist |L|<=|P|, also der logische Adressraum kleiner als der physische, so kann der logische Adressraum problemlos in den physischen aufgenommen werden
- Der Fall |L| > |P| ist jedoch der klassische Fall. Dies resultiert u.a. aus der weit verbreiteten Wortlänge von 32 Bit, mit der sich 232 Adressen ansprechen lassen. Nimmt man an, dass jede Adresse ein Speicherwort von 4 Byte adressiert, so lassen sich 232 ·4 = 234 Byte = 16 Gigabyte in L ansprechen

Paging - Begriffe

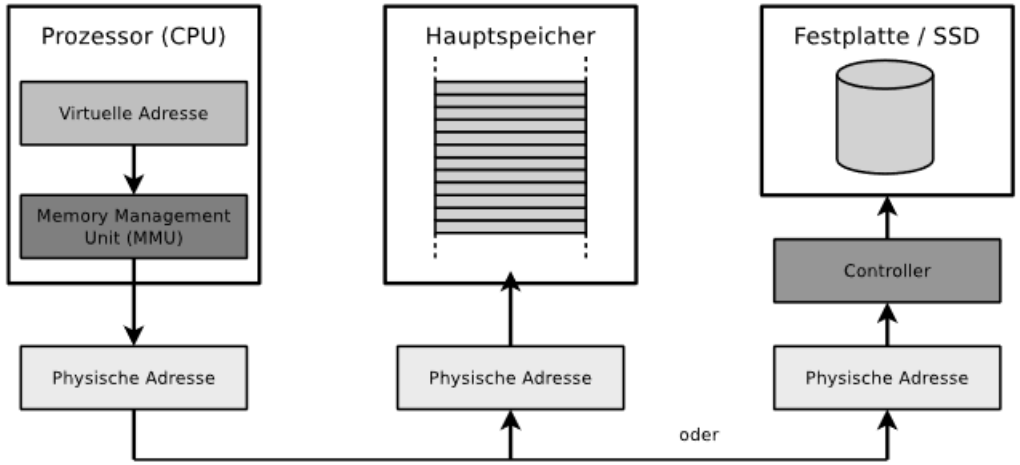
- **Seiten** (Pages) – virtueller/logischer Speicher
- **Seitenrahmen** (Frames) – physischer Speicher
- **Paging Area** (Schattenspeicher)
  - Für den Hauptspeicher wird ein Schattenspeicher in einem speziellen Plattenbereich reserviert (Paging Area)
- Mapping: **Page -> Frame**



htl wiener neustadt
Höhere technische Bundes- Lehr- und Versuchsanstalt Wiener Neustadt

Adressumsetzung MMU

- Virtuelle Speicheradressen übersetzt die CPU mit der MMU (Memory Management Unit) und der Seitentabelle in physische Adressen
- Das Betriebssystem prüft, ob sich die physische Adresse im Hauptspeicher, oder auf der SSD/HDD befindet



- Befinden sich die Daten auf der SSD/HDD, muss das Betriebssystem die Daten in den Hauptspeicher einlesen
- Ist der Hauptspeicher voll, muss das Betriebssystem andere Daten aus dem Hauptspeicher auf die SSD/HDD verdrängen

Paging

- Bei Bedarf werden von Seiten aus den Hintergrundspeicher in verfügbare Rahmen des Hauptspeichers geladen
- Vorteil
  - keine Speicherzerstücklung (externe Fragmentierung)
- Nachteil
  - Es kommt jedoch bei der Zuordnung der Seiten zu deutlichem Verschnitt(interne Fragmentierung)

Paging

- Demand Paging
  - Fehlt eine Seite im Hauptspeicher (d.h. liegt ein Seitenfehler vor), so wird diese on demand aus dem HGS nachgeladen
- Demand Prepaging
  - Fehlt eine Seite im Hauptspeicher, werden gleichzeitig *mehrere* Seiten geladen und verdrängt. Dadurch wird also die Zahl der Zugriffe auf den Hintergrundspeicher geringer gehalten
- Look-Ahead-Paging
  - Nicht nur bei Seitenfehlern, sondern auch nach anderen (näher zu definierenden) Kriterien können Nachladeoperationen stattfinden

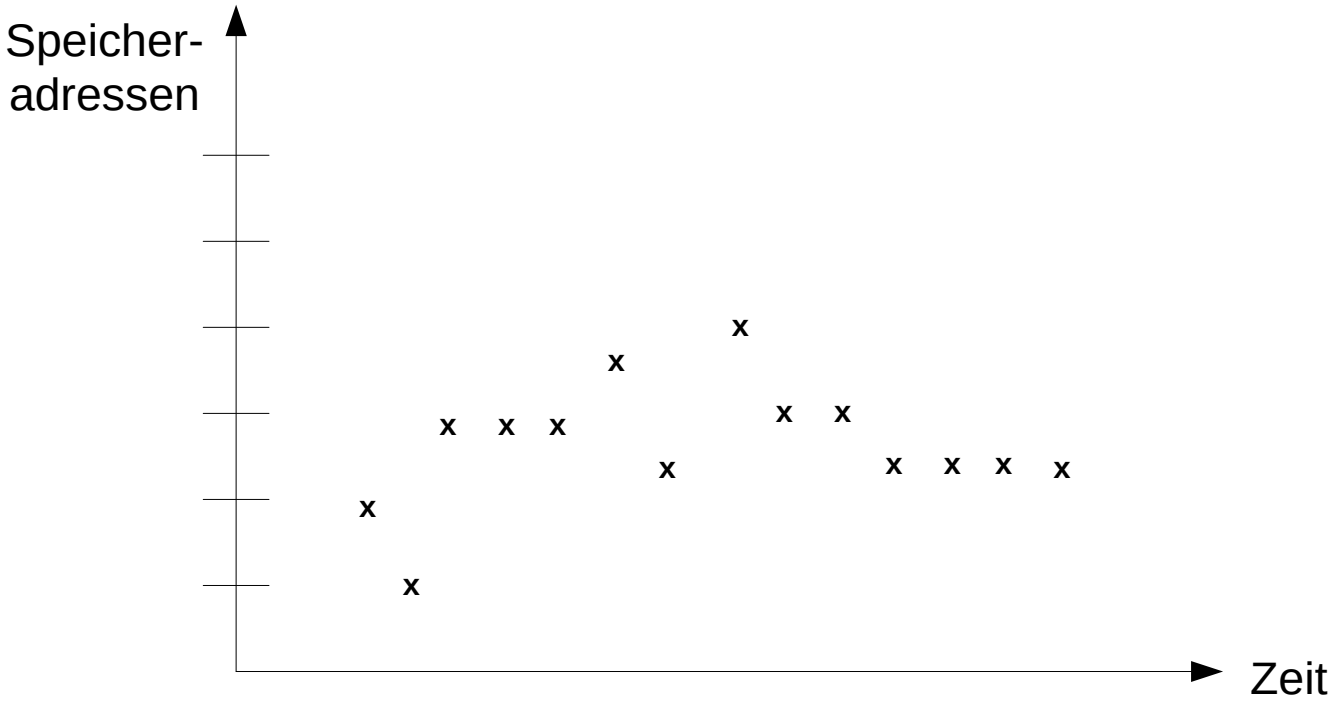
→ Nachladungen bei Demand Prepaing und Look-Ahead-Paging mit Lokalitätsprinzip

htl wiener neustadt
Höhere technische Bundes- Lehr- und Versuchsanstalt Wiener Neustadt

Lokalitätsprinzip

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

→ Diese sollten für den nächsten Zugriff bereitgehalten werden

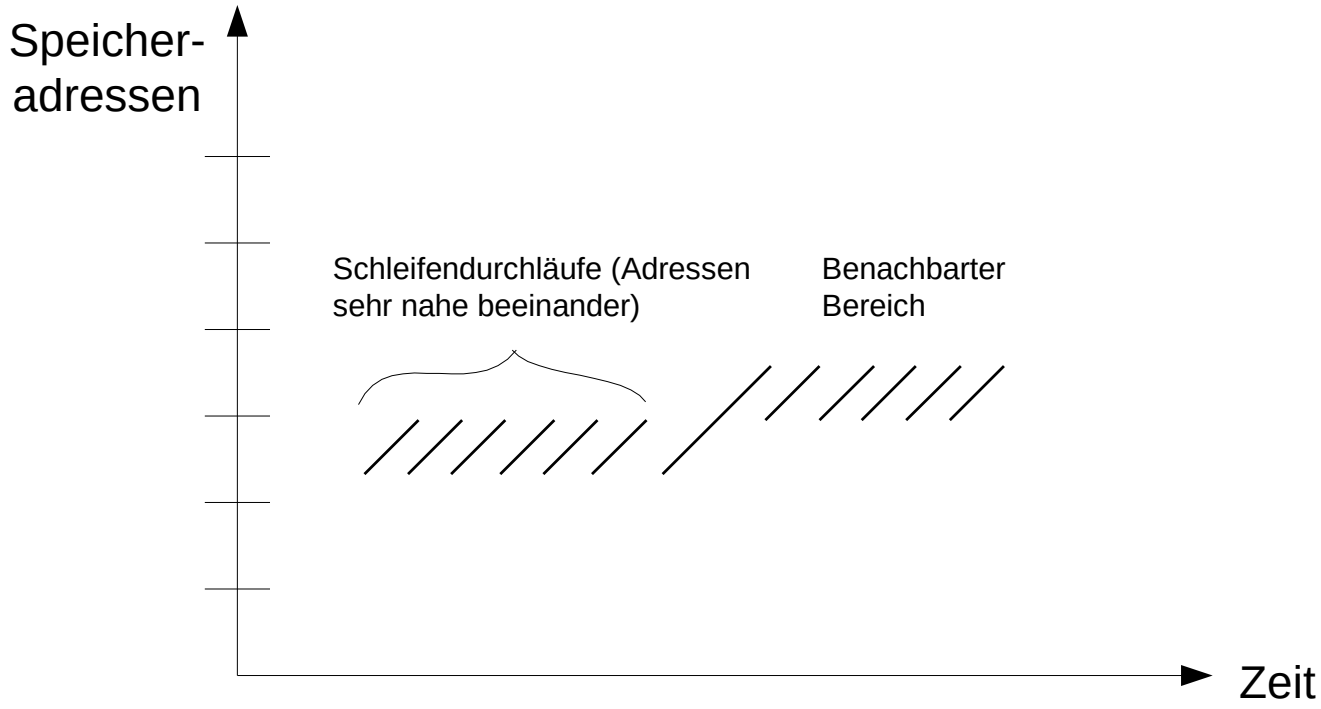


htl wiener neustadt
Höhere technische Bundes- Lehr- und Versuchsanstalt Wiener Neustadt

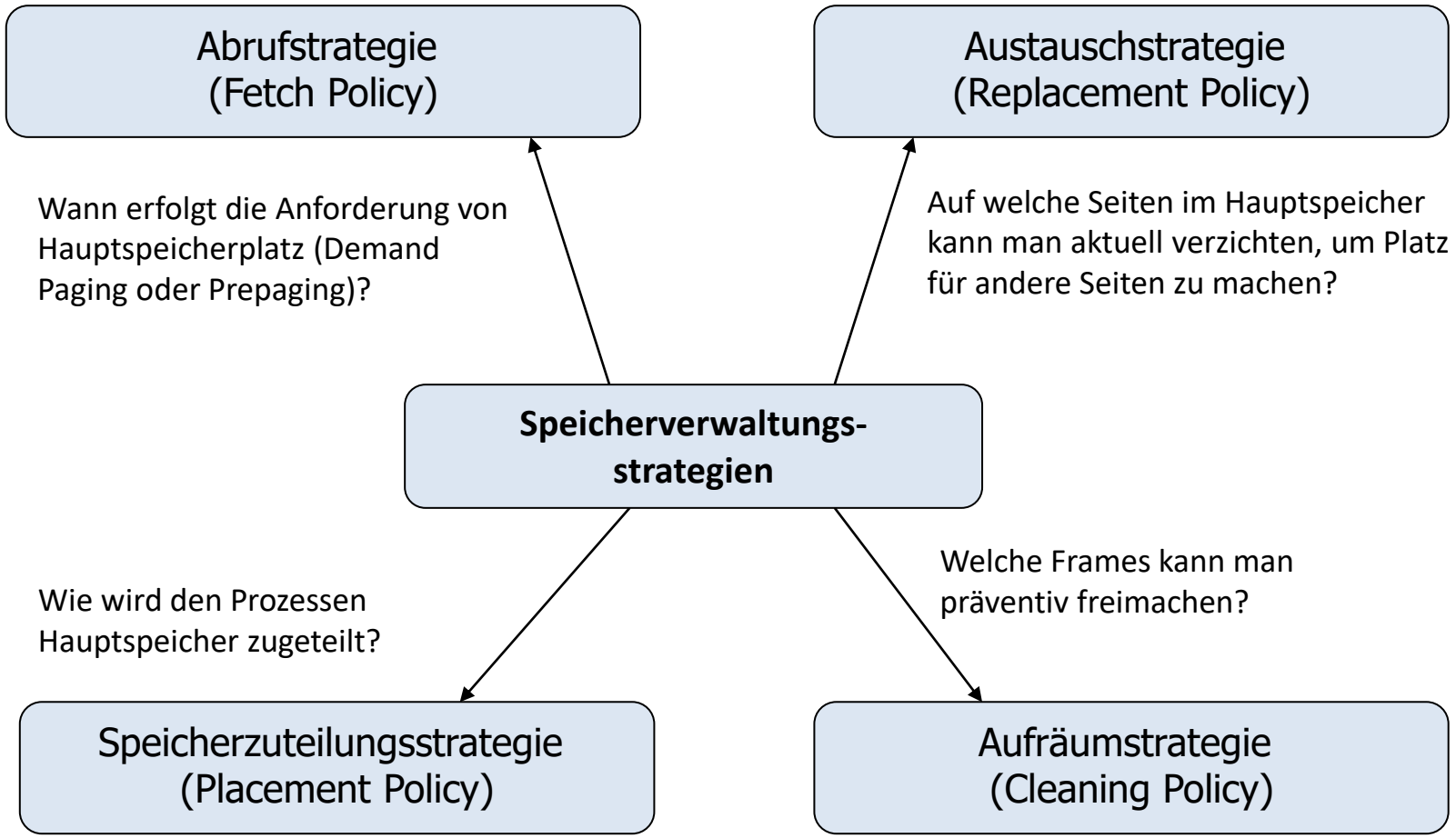
Lokalitätsprinzip

- **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



Speicherverwaltungsstrategien



Ersetzungsstrategien

- **OPT** (Optimalstrategie)
- **First In First Out**(FIFO-Strategie)
- **Least Recently Used** (LRU-Strategie)
- **LFU** (Least Frequently Used)
- **Climb** (Aufstieg bei Bewährung)
- **Clock-Strategie**

OPT-Strategie

- Lagert jeweils die Seite aus, also diejenige Seite, die am längsten nicht mehr gebraucht wird
- Diese Strategie ist wie ihr Name optimal und sinnvoll v.a. als Benchmarkingstrategie
- *Aber:* In der Praxis kann der Referencestring erst im Nachhinein aufgestellt werden, da zu einem bestimmten Zeitpunkt immer nur die als nächstes zu bearbeitende Seite sichtbar ist

1.Opt-Strategie Benchmarkstrategie — entfernen der Seiten, die am längsten nicht gebraucht werden — **nicht machbar in Praxis, da Zukunft betrachtet wird**

	2	3	2	1	5	2	4	5	3	2	5	2
Frame 1				1	5	5	5	5	5	5	5	5
Frame 2		3	3	3	3	3	3	3	3	3	3	3
Frame 3	2	2	2	2	2	2	4	4	4	2	2	2

Anzahl Seitenfehler	3
Anzahl Erstbesetzungen	3
	6

First In First Out(FIFO) - Strategie

- Ordnet Seiten nach dem Alter, im Bedarfsfall wird die älteste Seite, d.h. die Seite, die sich bereits am längsten im Hauptspeicher befindet, ausgelagert

2.FiFo (First in First out) entfernen der Seiten, die als erstes reinkommen

	2	3	2	1	5	2	4	5	3	2	5	2
Frame 1				1	5	2	4	4	3	3	5	2
Frame 2		3	3	3	1	5	2	2	4	4	3	5
Frame 3	2	2	2	2	3	1	5	5	2	2	4	3

Anzahl Seitenfehler	6
Anzahl Erstbesetzungen	3
	9

Least Recently Used(LRU) - Strategie

- Verdrängt Seiten nach dem Alter, d.h. die am längsten nicht mehr benutzte Seite wird aus dem Hauptspeicher ausgelagert
- Die zugrundeliegende Idee besteht in der Lokalität, d.h. die Seite, die gerade gebraucht wurde, wird wahrscheinlich wieder gebraucht

3.LRU (Least-Recently-Used) entfernen der Seiten, die am längsten nicht benötigt werden

neuerste Seite nach oben

	2	3	2	1	5	2	4	5	3	2	5	2
Frame 1				1	5	2	4	5	3	2	5	2
Frame 2		3	2	2	1	5	2	4	5	3	2	5
Frame 3	2	2	3	3	2	1	5	2	4	5	3	3

Anzahl Seitenfehler	4
Anzahl Erstbesetzungen	3
	7

Least Frequently Used(LFU) - Strategie

- entfernen der Seiten, mit der niedrigsten Nutzungshäufigkeit → meist genutzte Seiten bleiben im HS!
- Achtung: Anzahl der Zugriffe pro Seite muss verwaltet werden!
- Zählung möglich:
  - seit Laden der Seite
  - Zugriffe innerhalb der letzten Stunde
  - seit dem letzten Seitenfehler

htl wiener neustadt
Höhere technische Bundes- Lehr- und Versuchsanstalt Wiener Neustadt

Climb-Strategie (Aufstieg durch Bewährung)

- Hierbei steigt eine Seite bei jedem Aufruf eine Position höher, wenn sie bereits im Speicher vorhanden ist, d.h. sie tauscht ihre Position mit der vor ihr stehenden Seite
- Ist eine neue Seite nicht im Speicher vorhanden, so wird die unterste Seite ausgelagert und die neue Seite dorthin geladen

5.Climb (Aufstieg bei Bewährung)

	2	3	2	1	5	2	4	5	3	2	5	2
Frame 1	2	2	2	2	2	2	2	2	3	2	2	2
Frame 2		3	3	3	3	3	3	3	2	3	5	5
Frame 3				1	5	5	4	5	5	5	3	3

Anzahl Seitenfehler	3
Anzahl Erstbesetzungen	3
	6