FH Bielefeld
University of
Applied Sciences

Vorlesung

Betriebssysteme

Teil 7

Speicherverwaltung

FH Bielefeld
University of
Applied Sciences

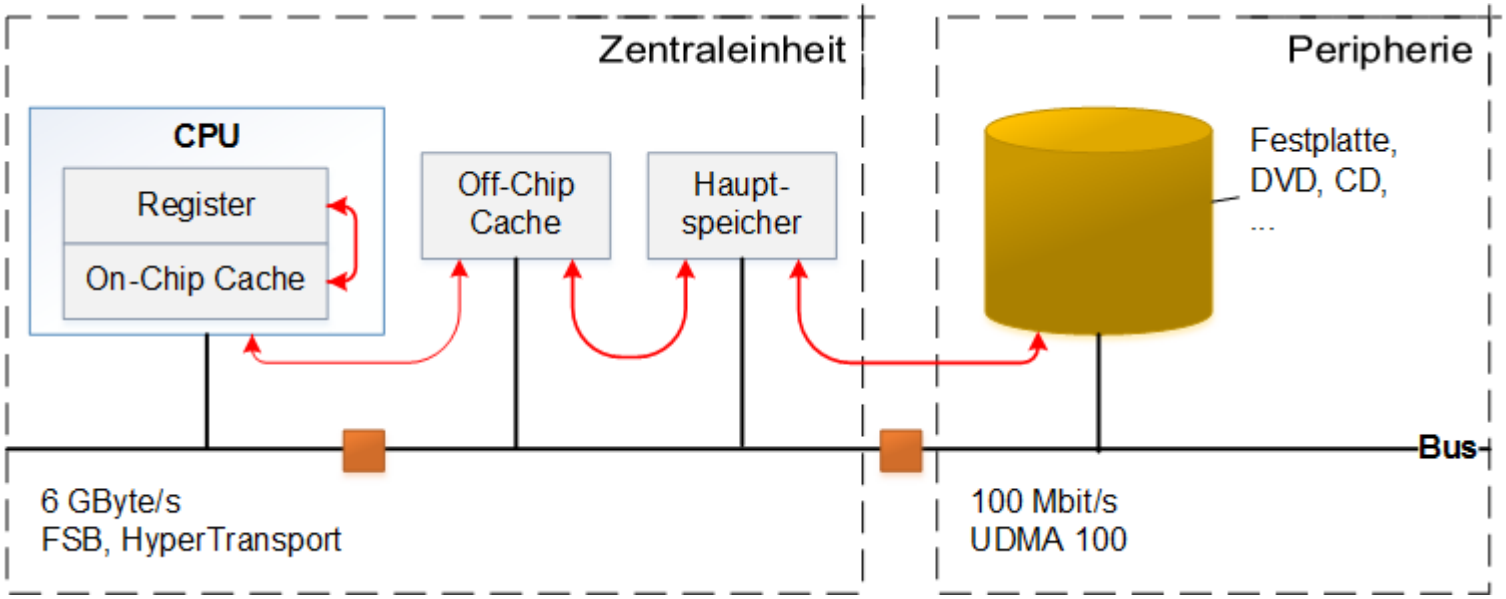
Inhalt der nächsten zwei Vorlesungstermine

- Speicherverwaltung
- Speicherhierarchie
- Swapping
- Virtueller Speicher
- Seitenersetzungsstrategie

- Übungsaufgaben

FH Bielefeld
University of
Applied Sciences

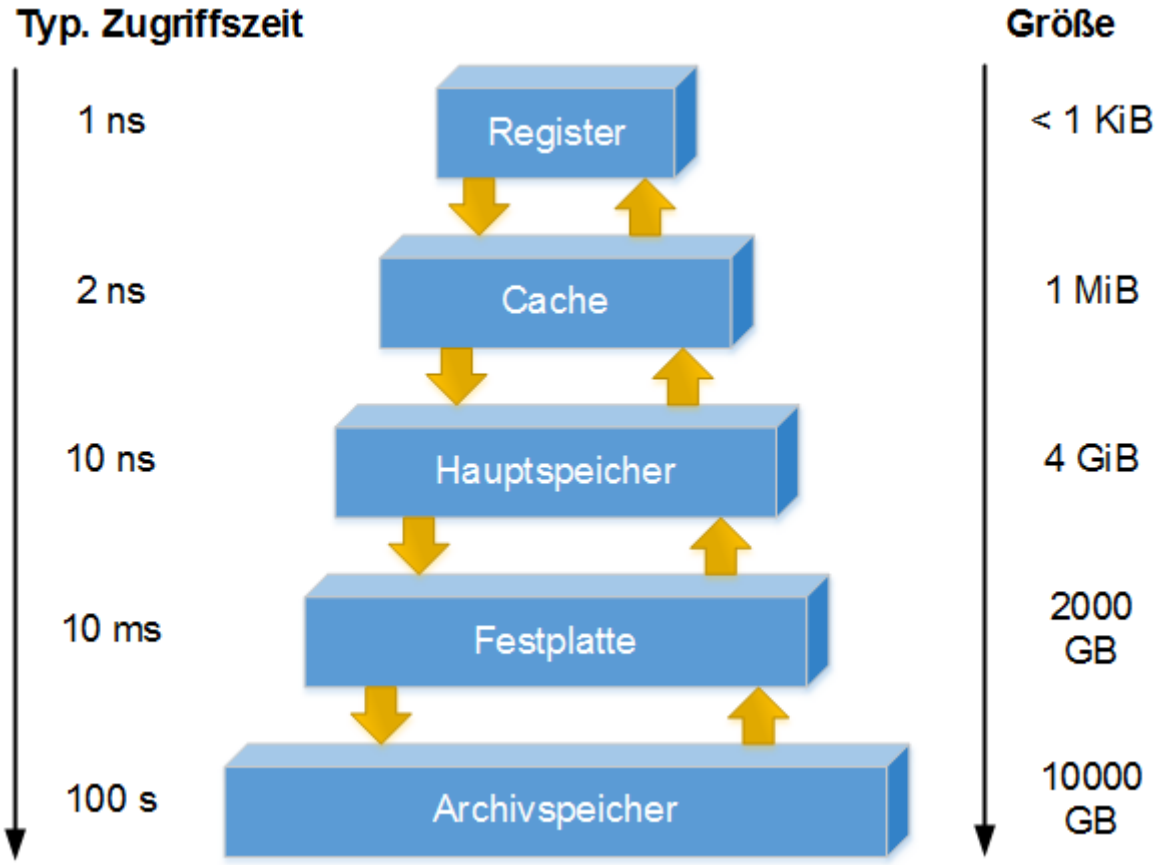
Speicherverwaltung: Einführung



- *Bemerkung*: In der Regel wird ein solches System durch mehr als einen Bus realisiert. Die Bussysteme haben auch verschiedene Charakteristiken (Datenrate, Durchsatz, Latenz,…).

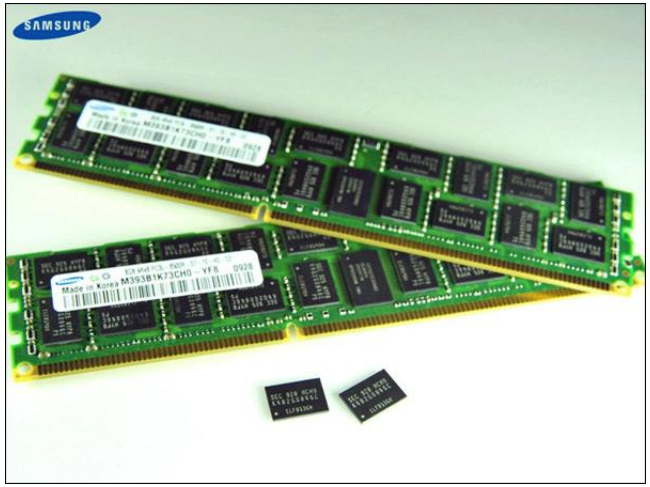
FH Bielefeld
University of
Applied Sciences

Speicherverwaltung: Hierarchie



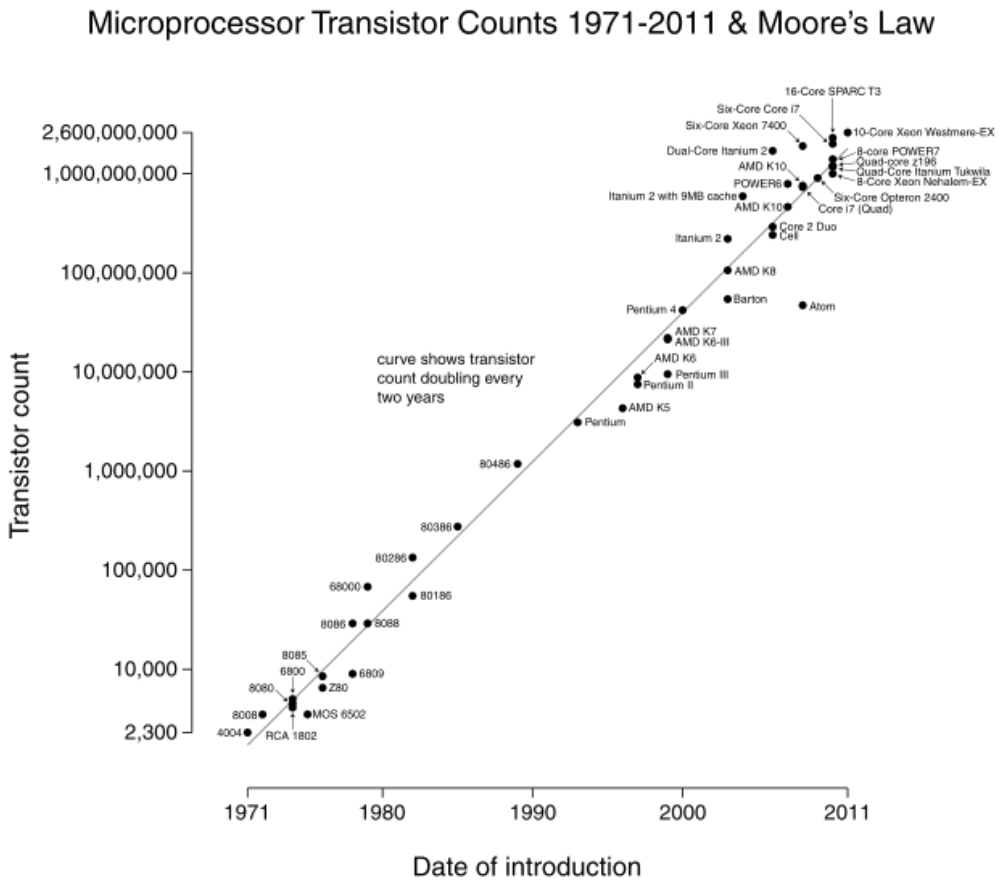
FH Bielefeld
University of
Applied Sciences

Speicherbausteine



FH Bielefeld
University of
Applied Sciences

Moore‘s Law



- Verdopplung der Speicherkapazität alle 2 Jahre

FH Bielefeld
University of
Applied Sciences

Grenzen von Moores Law

- Parallelisierung (Gesetz von Amdahl)
  - Doppelte Anzahl Prozessorkerne bedeutet nicht doppelte Rechenleistung

- Technologische Hürden müssen überwunden werden
  - Strukturgößen aktuell: 28nm
  - Kommende Generation: 16nm

  - Vergleich: Influenzavirus 50nm Durchmesser
  - Siliziumatom 0,1nm Durchmesser

FH Bielefeld
University of
Applied Sciences

Speicher

„Bauernregel“

- Speichergröße wächst um 60% pro Jahr
- Speicherpreis fällt um 25% pro Jahr

- Warum wächst der Speicherbedarf?
- („640 Kilobyte Arbeitsspeicher ist alles, was irgendeine Anwendung jemals benötigen sollte.“, Bill Gates)

- Speicherbedarf wächst:
  - Multimedia/Audio/Video
  - Spekulative Ausführung
  - High-Level Programmierung

FH Bielefeld
University of
Applied Sciences

Fragestellungen für Speicherverwaltungssysteme

- Ein oder mehrere Prozesse?
- Zuweisung von Speicherplatz an Prozesse:
  - fest oder dynamisch?
  - wenn fest: gleiche oder unterschiedliche Teile (partitions)?
- Zuweisung von Prozessen auf feste oder unterschiedliche Anfangsadressen?
- Zuweisung von zusammenhängenden oder verteilten Speicherbereichen?
- Organisation von Hauptspeicher (Primärspeicher) und Platte (Sekundärspeicher)
- Organisation von schnellem Pufferspeicher (Cache) und Hauptspeicher
  (hier nicht behandelt, da für das BS unsichtbar)

FH Bielefeld
University of
Applied Sciences

Speicherverwaltung: Motivation

Aufgaben des Betriebssystems:

- **Buchhaltung**: Welche Speicherbereiche sind belegt, welche sind frei?
- **Speichervergabe** an Prozesse
- **Speicherrücknahme** von Prozessen
- **Verschieben** von Daten zwischen den Speicherhierarchien

FH Bielefeld
University of
Applied Sciences

Speicherverwaltung: Arten

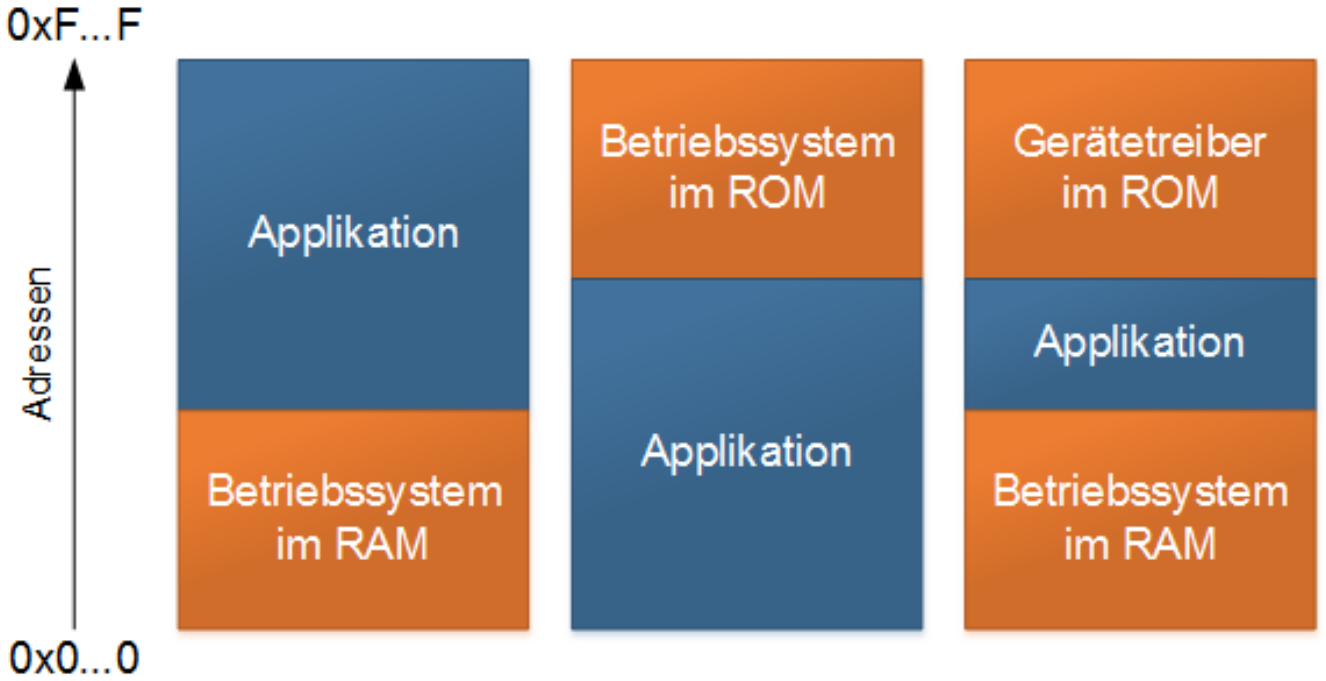
Zwei Arten der Speicherverwaltung können unterschieden werden:

1. Speicherbelegung ist **fest** (einfache Variante)
   - **Monoprogrammierung**:
     - Nur **ein** Prozess und das Betriebssystem teilen sich den Speicher
     - frühe Batchsysteme, Embedded Systems (BS im ROM)
     - **Problem**: Speicherschutz zwischen BS und Applikation
   - **Multiprogrammierung**:
     - **Mehrere** Prozesse gleichzeitig im Speicher,
     - jeder Prozess bekommt einen **festen** Speicherbereich.
2. Speicherbelegung ist **flexibel:**
   - Prozesse werden zwischen Hauptspeicher und Platte verschoben
   - Verfahren: *Swapping*, *virtueller* Speicher

FH Bielefeld
University of
Applied Sciences

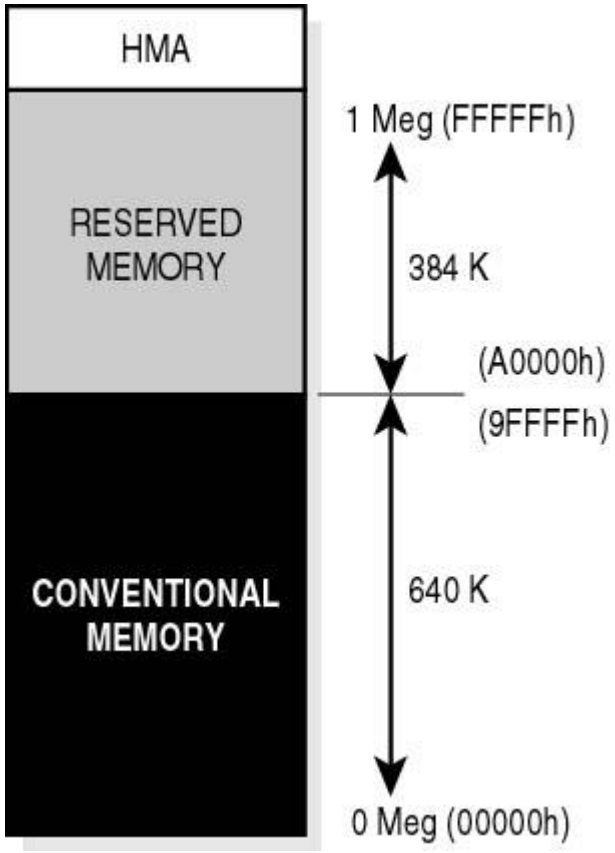
Speicherverwaltung: Monoprogrammierung

- **Varianten** der Speicherbelegung bei Mono-Programmierung.



FH Bielefeld
University of
Applied Sciences

Speicher unter MS-DOS



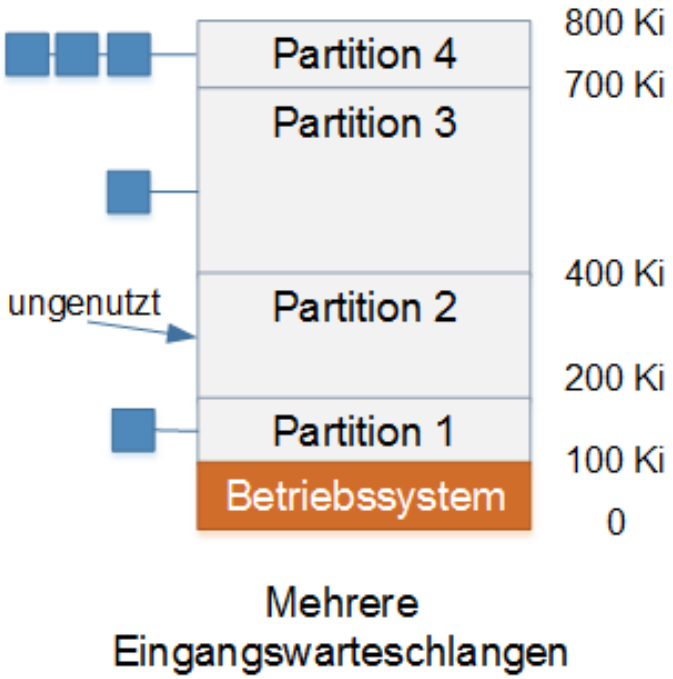
Memory Type	Total	=	Used	+	Free
Conventional	640 KB		122K		518K
Upper	155K		41K		144K
Reserved	128K		128K		OK
Extended (XMS)	7,269K		2,486K		4,783K
Total Memory	81,259K		21,777K		5,415K
Total under 1 MB	795K		163K		632K
Largest executable program size			518K (530,096 bytes)		
Largest free upper memory block			114K (116,352 bytes)		

FH Bielefeld
University of
Applied Sciences

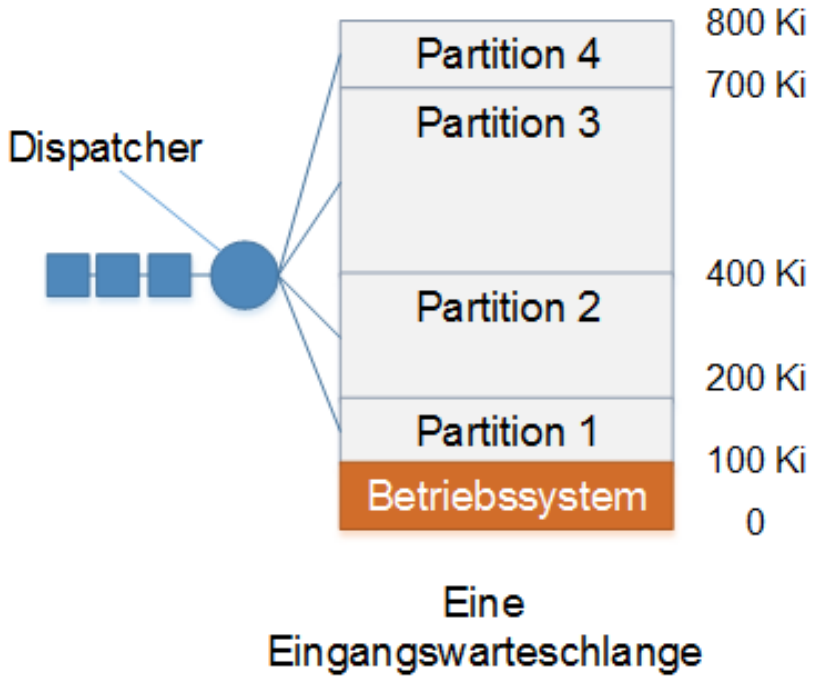
Speicherverwaltung: Multiprogrammierung

Multiprogrammierung bei **fester** Speicherbelegung:

- Der Hauptspeicher wird in (evtl. unterschiedlich große) Speicherpartitionen eingeteilt:



Mehrere Eingangswarteschlangen



Eine Eingangswarteschlange

FH Bielefeld
University of
Applied Sciences

Speicherverwaltung: Relokation

Möglichkeiten der **Relokation** von Prozessen:

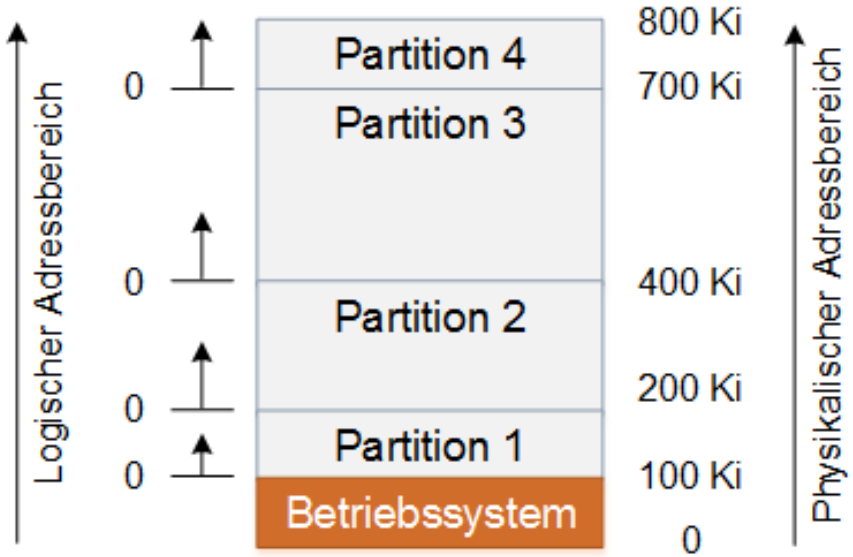
- Es wird **nur relative Adressierung** eingesetzt.
  - Geht nicht für jeden Prozessortyp
- Es wird eine **Relokationstabelle** benutzt:
  - Programm wird für Adresse 0 gelinkt.
  - **Relokationstabelle** enthält jede **absolute Adresse** in der Binärdatei.
  - **Beim Laden** des Programms wird der Startoffset zu den Adressen addiert.
- Hardwareunterstützung → *Memory Management Unit* (MMU):
  - **Basisregister** wird automatisch zu jeder Adressinformation addiert.

FH Bielefeld
University of
Applied Sciences

Speicherverwaltung: MMU

Die Hardware zur Unterstützung der Relokation ist die *Memory Management Unit* (MMU)

- Jeder Prozess hat seinen eigenen logischen Adressbereich.
- Bei jedem **Kontextwechsel** wird die **MMU umprogrammiert**, um den neuen logischen Adressbereich einzustellen.
- Keine Relokation des Programmcodes notwendig, da jeder Prozess ab der logischen Adresse 0 beginnt.
- **Speicherschutz** gegeben, da jeder Prozess nur seinen Speicher sieht (sonst: *Segmentation Fault*).



FH Bielefeld
University of
Applied Sciences

Speicherverwaltung: Probleme mit fester Speicherbelegung

Moderne Computersystem haben andere Anforderungen:

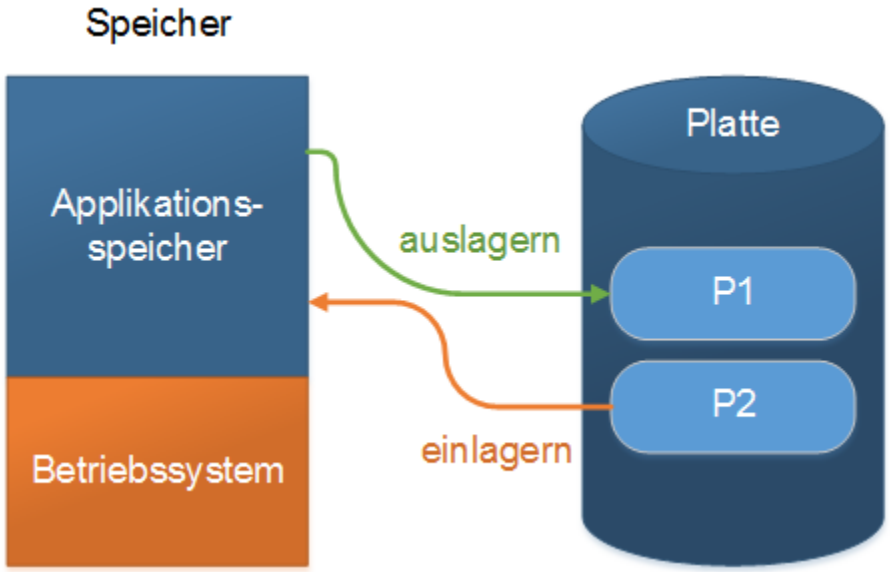
- Es laufen meist viele Prozesse, die **schnell reagieren** müssen (Interaktive Systeme)
- Prozesse haben **mehr Speicherbedarf, als physikalischer Speicher** vorhanden ist

Mögliche Lösungen:

- **Swapping**: Verschieben von kompletten Prozessen zwischen Platte und Hauptspeicher
- **Virtueller Speicher**: Daten der Prozesse sind nur zum Teil im Hauptspeicher

Swapping: Prinzip

- Der *komplette* Adressraum der Prozesse wird zwischen Hauptspeicher und Platte ausgetauscht.



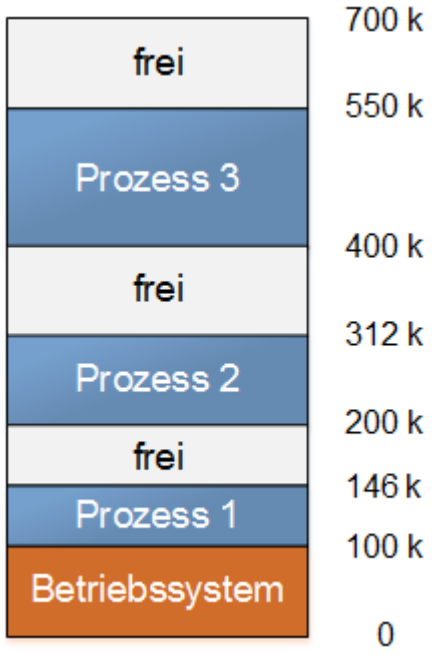
Kann vom Betriebssystem **ohne Hardwareunterstützung** realisiert werden.

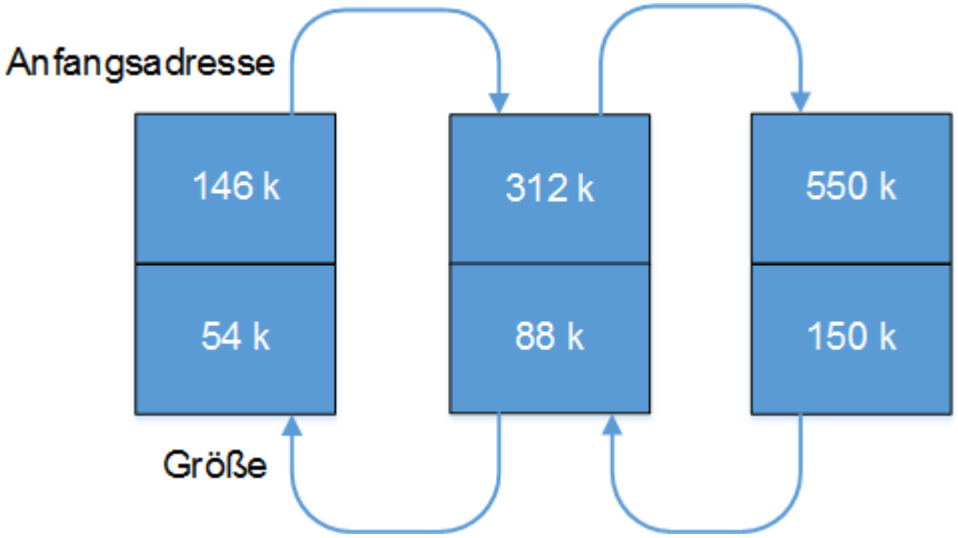
- Extrem **aufwändiger Prozesswechsel** durch Zugriff auf Festplatte

FH Bielefeld
University of
Applied Sciences

Swapping: Beispiel (1)

- Mögliche Speicherbelegung: → zugehörige Freibereichsliste:

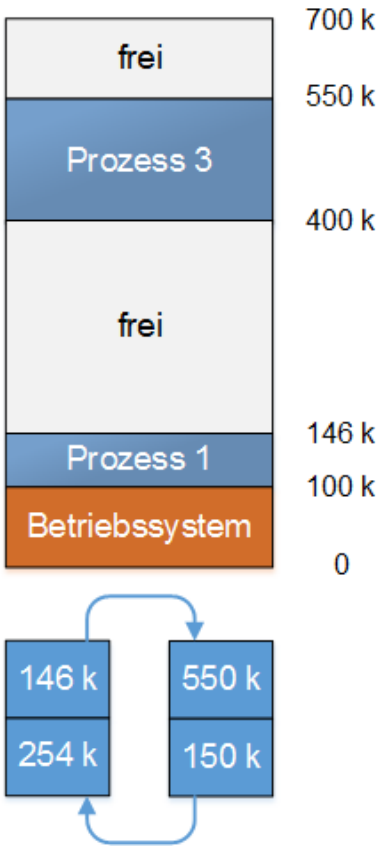




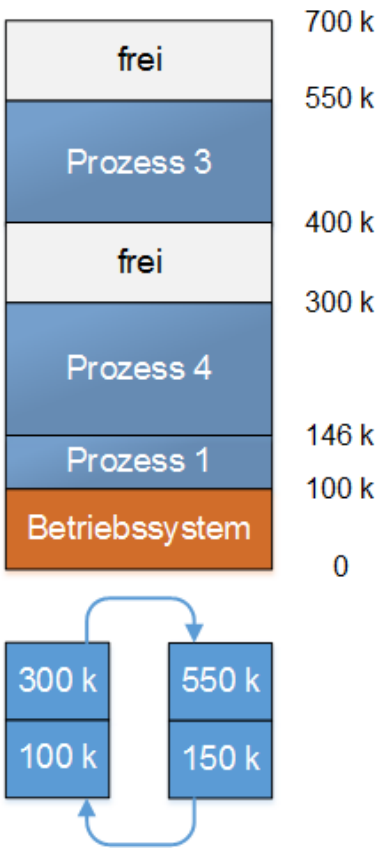
FH Bielefeld
University of
Applied Sciences

Swapping: Beispiel (2)

- Prozess 2 wird ausgelagert:



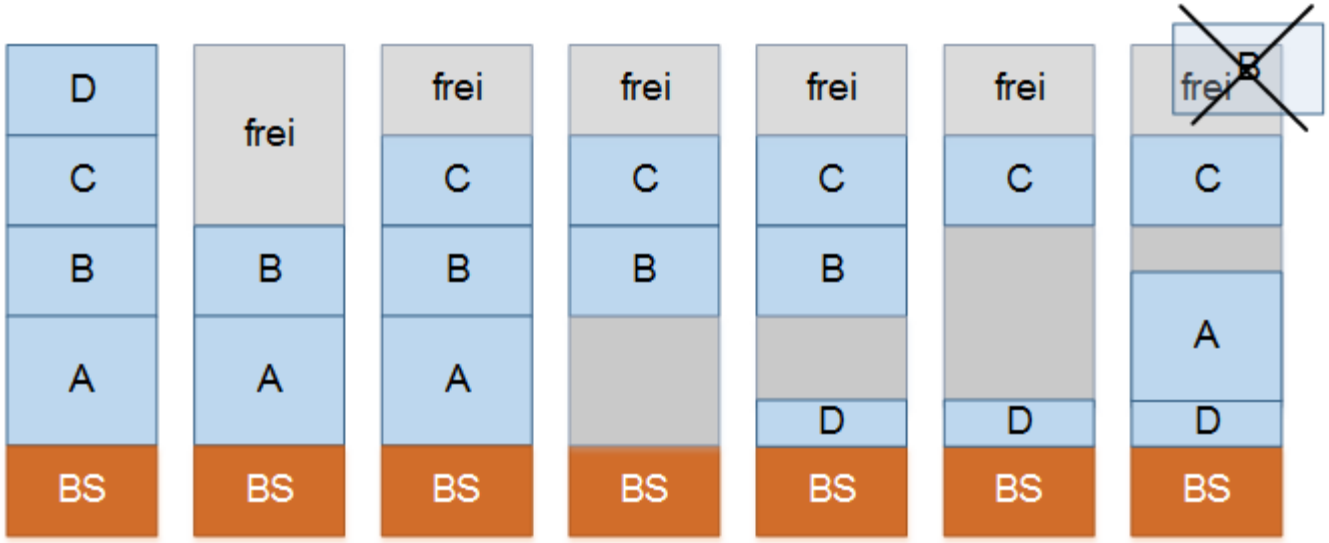
- Prozess 4 wird eingelagert:



FH Bielefeld
University of
Applied Sciences

Swapping: Fragmentierung

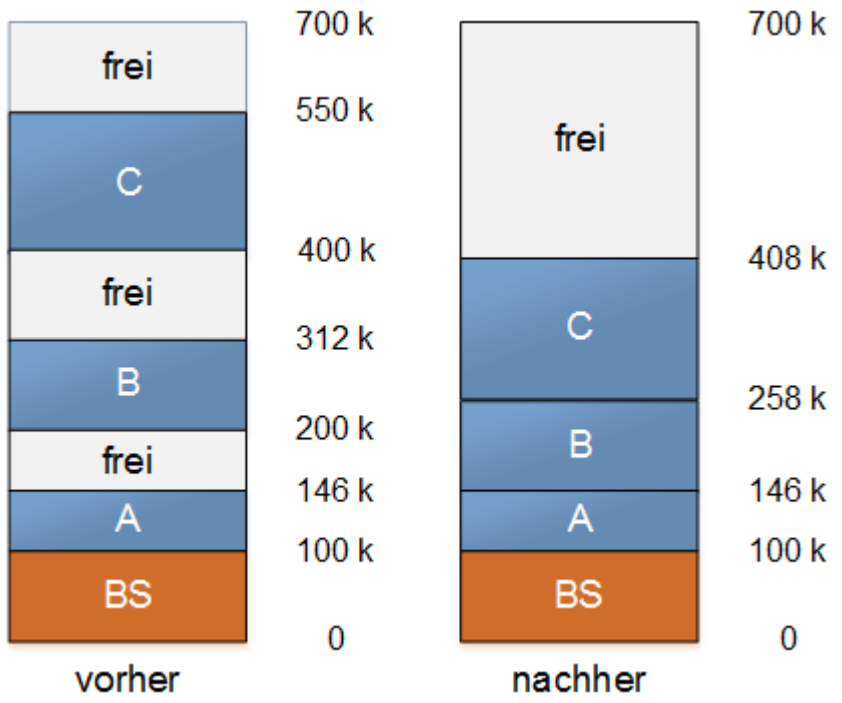
- Der Hauptspeicher wird in **Segmente variabler Länge** eingeteilt, die den Prozessen zugeteilt werden.
- Zur Verwaltung dienen **Segmenttabellen**.
- Durch Entfernen und Hinzufügen von Segmenten:
  - Entstehen langfristig kleine, **unbenutzte Speicherbereiche** (*externe Fragmentierung*).
  - Fragmente können verschoben und zu einem Segment zusammengefasst werden.
- Beispiel:



FH Bielefeld
University of
Applied Sciences

Swapping: Speicherverdichtung

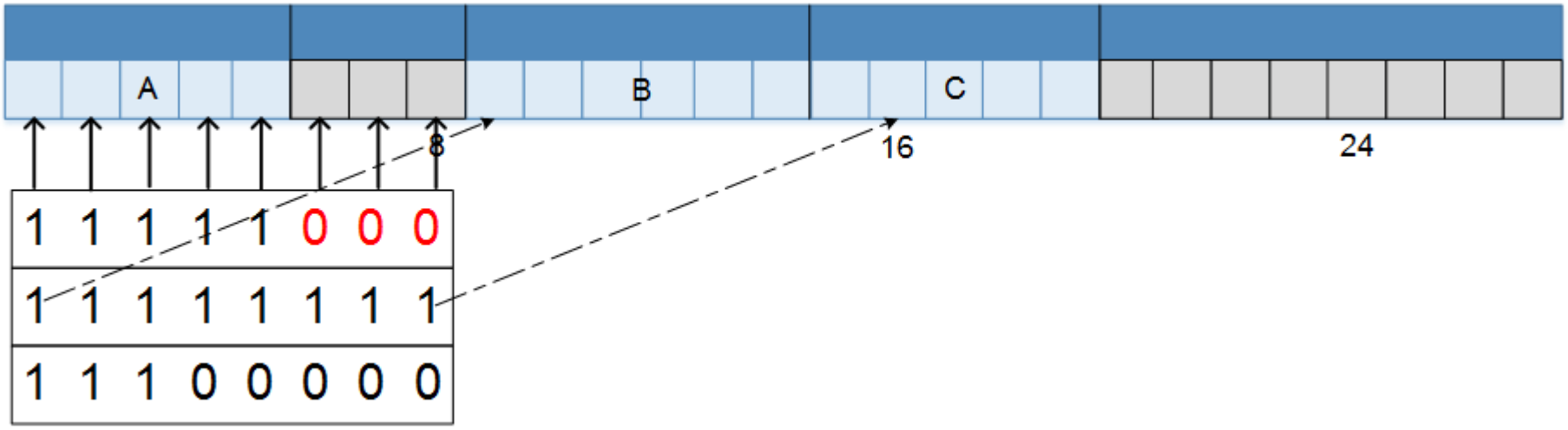
- Per **Speicherverdichtung** (*memory compaction*):
  - Viele kleine freie Speicherbereiche werden zu einem großen Bereich zusammengefasst
  - Ziel: **De-Fragmentierung**
- Optimale Verdichtungsstrategie ist schwer zu finden:
  - Wann / Wie oft?
- Speicherverdichtung erfordert immer **viel CPU-Leistung**.



FH Bielefeld
University of
Applied Sciences

Swapping: Speicherverwaltung mit Bitmaps

- Der dynamisch zugewiesene Speicher muss vom Betriebssystem verwaltet werden.
- Möglichkeit 1: Verwaltung mit einer **Bitmap**. Dazu wird der Speicher in Allokationseinheiten unterteilt.

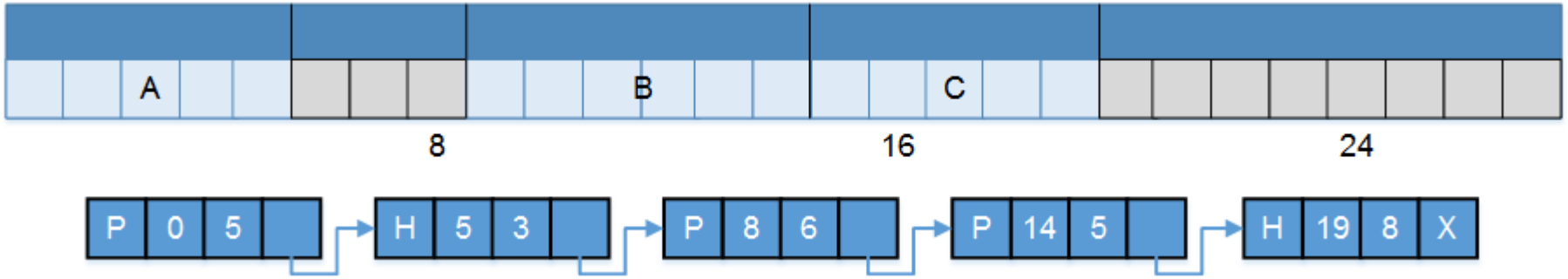


- Es wird eine Bitmap gebildet, in der jedes Bit eine Allokationseinheit (z.B. 1 kByte) identifiziert. Daher hat die Bitmap eine feste Größe.

FH Bielefeld
University of
Applied Sciences

Swapping: Speicherverwaltung mit Listen

- Möglichkeit 2: Verwaltung des Speichers mit einer **verketteten Liste**. Dazu wird der Speicher in Allokationseinheiten unterteilt.
- Die verketteten Liste verwalten den belegten und freien Speicher.



- Verwaltung und Suche im allgemeinen einfacher als bei Bitmaps (einfache Listenoperationen).

FH Bielefeld
University of
Applied Sciences

Swapping: Speicherbelegungsstrategien

Wie finde ich ein **passendes** Segment, wenn ein Prozess neuen Speicher benötigt?

- **First fit**: Das erste passende Loch in der Liste wird genommen. Vorteil: sehr effizient!
- **Next fit**: Wie *first fit*, allerdings wird die Suche an der Stelle, wo zuletzt Speicher gefunden wurde, fortgesetzt (→ Ringstruktur).
- **Best fit**: Sucht das **kleinste passende** Loch in der gesamten Liste. Große Löcher bleiben lange bestehen, es werden aber viele kleine, nutzlose Löcher erzeugt.
- **Worst fit**: Sucht das **größte freie Loch**. Alle Löcher verfügen mit der Zeit über ungefähr die gleiche Größe. Große Speicherbereiche können aber evtl. später nicht mehr bereit gestellt werden.
- **Quick fit**: Für die Löcher werden getrennte Listen für Löcher gebräuchlicher Größe erzeugt. Findet sehr schnell passende Löcher, die Speicherfreigabe ist allerdings aufwändig (Abgleich Listen bei verschmelzen Löcher).

FH Bielefeld
University of
Applied Sciences

Swapping: Fazit

- Swapping bei fester Speicherbelegung ist heute nicht mehr gebräuchlich.
- Grund:
  - Overhead zum Austausch von kompletten Prozessen zu hoch.
- Weiterhin ungeklärte Frage:
  - Was passiert, wenn ein Prozess mehr Speicher benötigt, als physikalisch vorhanden ist?

    →Virtueller Speicher

FH Bielefeld
University of
Applied Sciences

Virtueller Speicher: Überblick

- Idee des **virtuellen Speichers** (*virtual memory*, Fotheringham, 1961):
  - Ist ein **Programm größer** als der zur Verfügung stehende Speicher, dann wird nur der gerade benötigte Teil im Speicher gehalten.
- Wichtige Fragen:
  - Welche Teile werden gerade benötigt?
  - Welche Teile können ausgelagert werden? -> **Auslagerungs- und Einlagerungsstrategien**.
- **Zweistufiges Adressierungsschema**:
  - Die von den Programmen benutzten virtuellen Adressen werden von der *Memory Management Unit* (MMU) in physikalische Adressen umgewandelt und
  - dann erst an den Speicher gegeben.
- Wichtigstes Verfahren: **Paging**

Automatisches Paging (aus Tanenbaum)

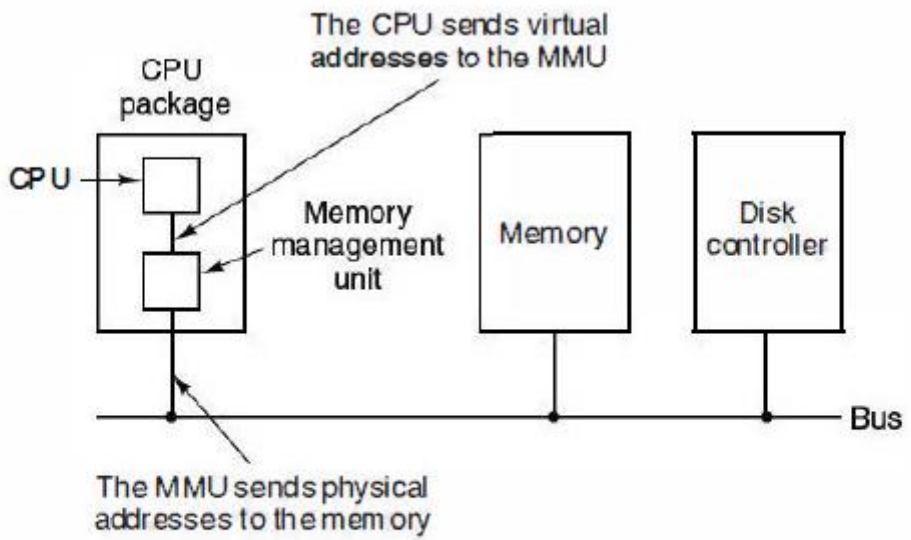


Figure 3-8. The position and function of the MMU. Here the MMU is shown as being a part of the CPU chip because it commonly is nowadays. However, logically it could be a separate chip and was in years gone by.

FH Bielefeld
University of
Applied Sciences

Virtueller Speicher: Paging

Prinzip des Paging:

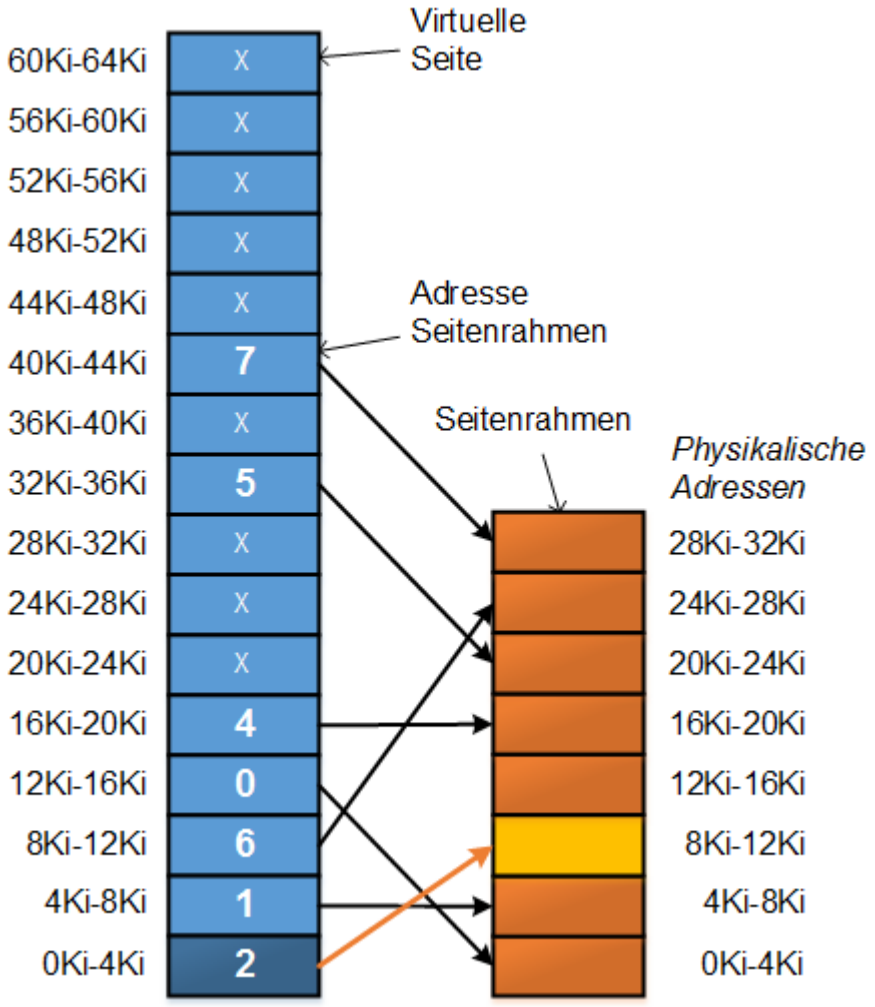
- Der virtuelle Adressraum ist in **Seiten** (*pages*) aufgeteilt.
- Der physikalische Speicher ist in **Seitenrahmen** / **Seitenkacheln** (*page frames*) aufgeteilt.
- Seiten und Seitenrahmen sind immer gleich groß!
- Die virtuelle Adresse wird in
  - eine **Seitennummer** (*page number*) und
  - eine **Adresse** innerhalb der Seite (*page offset*) aufgeteilt.
- Die Seitennummer adressiert einen Seitenrahmen über eine **Seitentabelle** (*page table*).
- Seiten, die nicht im Speicher gehalten werden können, werden auf Platte (Hintergrundspeicher) **auslagert**.

FH Bielefeld
University of
Applied Sciences

Virtueller Speicher: Beispiel Paging

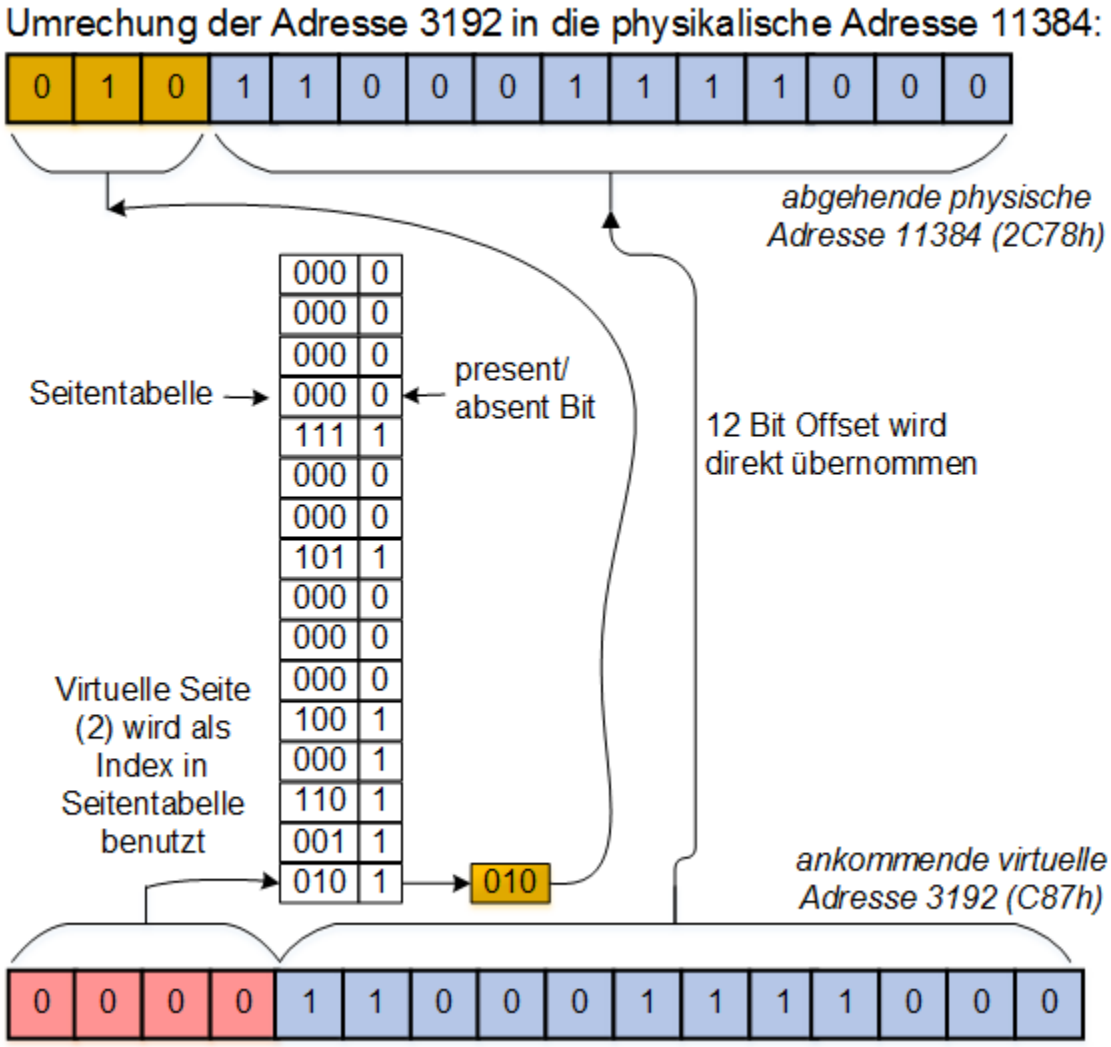
Beispiel für Paging:

- Ein System habe
  - 64 KiByte virtuellen Speicher (16 Bit Adressen)
  - aber nur 32 KiByte RAM.
  - Die Seitengröße betrage 4 KiByte.
- Der virtuelle Adressraum wird auf die physikalischen Adressen abgebildet.
- Aus dem Befehl: **MOV REG, 3192** im virtuellen Adressraum wird hier: **MOV REG, 11384** auf dem physikalischen Bus.



FH Bielefeld
University of
Applied Sciences

Virtueller Speicher: Umrechnung der Adressen in MMU



FH Bielefeld
University of
Applied Sciences

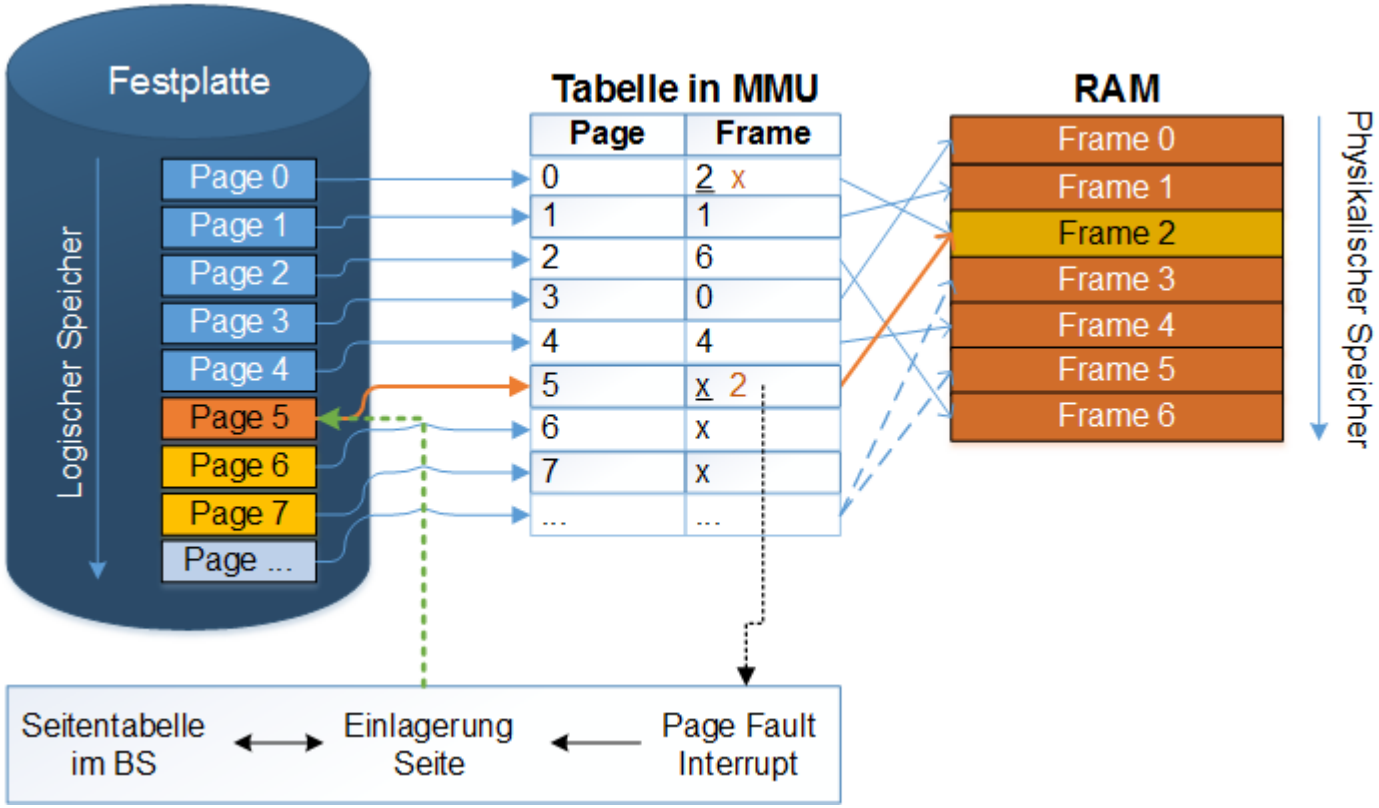
Virtueller Speicher: Seitenfehler

Was passiert bei einem Zugriff auf eine **nicht geladene Seite**?

- Beispiel: Befehl **MOV REG, 22870**
- Die MMU stellt fest, dass die virtuelle Seite (5 = 0101) nicht geladen ist und löst eine **Unterbrechung** aus →**Seitenfehler** (*page fault*). Der aufrufende **Prozess** wird **blockiert**.
- Das Betriebssystem sucht einen **freien Seitenrahmen** aus.
  - Ist kein Seitenrahmen frei, wird ein benutzter Seitenrahmen gewählt.
- Wurde dieser modifiziert, wird er auf die Platte zurückgespeichert.
- **Seite** wird von der Platte **geladen** und in den **Seitenrahmen** geschrieben.
- **Seitentabelle** wird **aktualisiert**.
- Der **Befehlszähler** des aufrufenden Prozesses wird **zurückgesetzt** (der letzte Befehl muss wiederholt werden) und der Prozess wird wieder in den ‚bereit' Zustand versetzt.

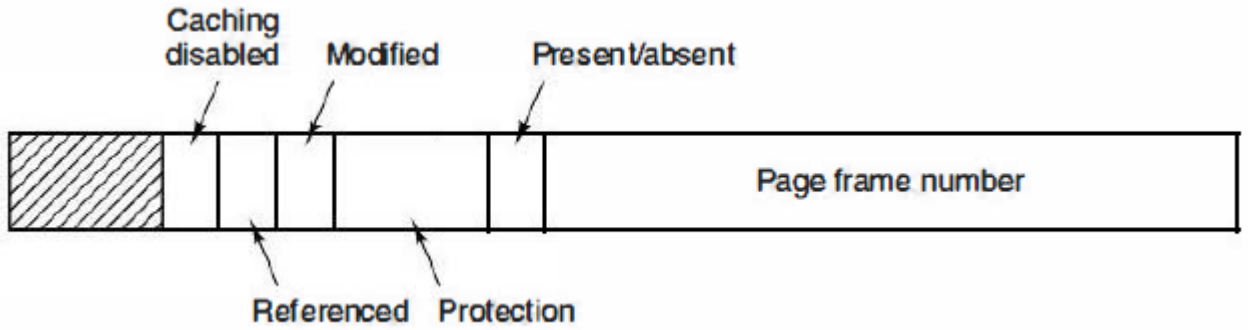
FH Bielefeld
University of
Applied Sciences

Virtueller Speicher: Arbeitsweise Paging



- Einlagern / Auslagern erfolgt unabhängig von Prozesszugehörigkeit
- Einlagern, falls Seite (hier: Seite 5) von einem Prozess referenziert wird
- Auslagern (Seite 0 in Rahmen 2), falls Rahmen für eine andere Seite benötigt wird

Struktur eines Eintrags in der Page Table



- Page frame number:
  - Verweis auf Seitenrahmen
- Present
  - 0: Seite noch nicht im Speicher (**hard miss**)
  - 1: Seite im Speicher enthalten
- Modified (**Dirty Bit**)
  - 0: Seite unverändert, (Inhalt identisch zum Hintergrundspeicher)
  - 1: Seite verändert
- Referenced (Zugriff erfolgt ja/nein?)
- Caching disabled (I/O Mapping)
- Protection: z.B. read/write/execute

FH Bielefeld
University of
Applied Sciences

Virtueller Speicher: Seitentabellen (1)

Problemstellungen bei Seitentabellen:

1. Die **Seitentabellen** können sehr **groß** werden.
   - Beispiel 1: Bei 32 Bit Adressbus-Breite und 4 KiByte Seitengröße existieren ca. 1 Millionen Einträge pro Prozess (4 GiByte Adressraum pro Prozess).
   - Beispiel 2: Bei 64 Bit Adressbus-Breite und 4 KiByte Seitengröße existieren 2^{52} (ca. $4{,}5 \times 10^{15}$) Einträge pro Prozess.

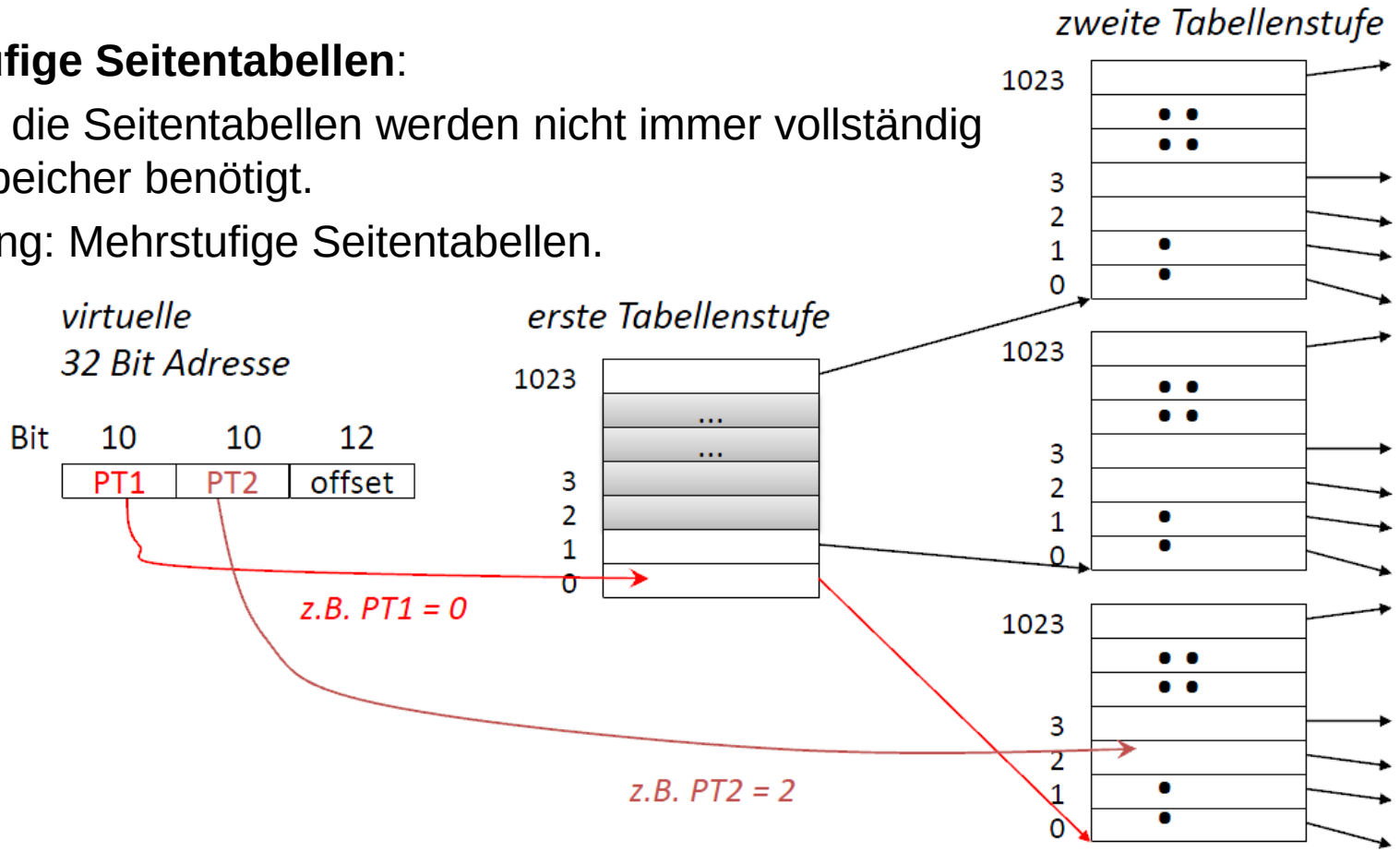
2. Die **Adressumrechnung** muss sehr **schnell** erfolgen.
   - Der Zugriff auf die Seitentabelle darf nicht zum Flaschenhals werden.
   - Pro Maschinenbefehl können mehrere Zugriffe auf Seitentabelle nötig sein (Befehlswort, Speicheroperand)

FH Bielefeld
University of
Applied Sciences

Virtueller Speicher: Seitentabellen (2)

Mehrstufige Seitentabellen:

- Auch die Seitentabellen werden nicht immer vollständig im Speicher benötigt.
- Lösung: Mehrstufige Seitentabellen.



FH Bielefeld
University of
Applied Sciences

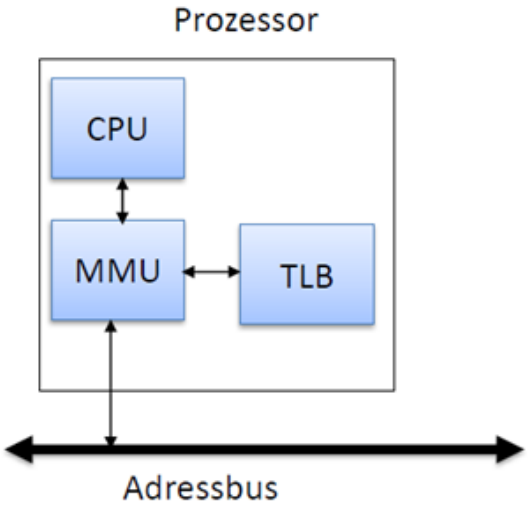
Translation Lookaside Buffer (TLB)

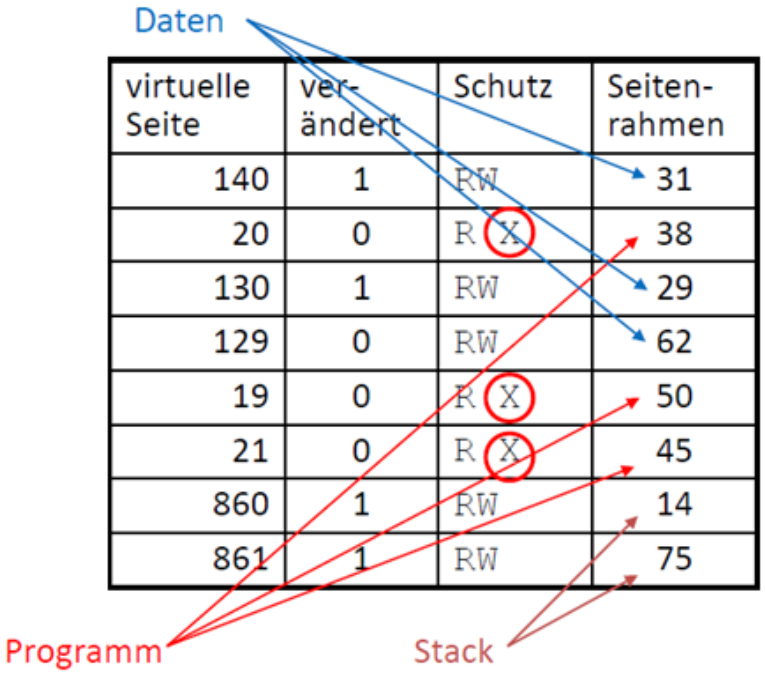
- Untersuchung:
  - Prozesse greifen **häufig** auf einige **wenige** Seiten zu
- Idee:
  - Der Paging-Tabelle, die im Arbeitsspeicher liegt, wird eine kleine schnelle Hardware-Tabelle (Registerebene) vorgeschaltet - **TLB**
- Oftmals wird die Seite im TLB gefunden
- Ist die Seite nicht im TLB enthalten (**soft miss**), wird die Pagingtabelle ausgelesen
  - Die Adresse wird dann in den TLB übernommen und ersetzt dort einen alten, länger nicht verwendeten Eintrag

FH Bielefeld
University of
Applied Sciences

Virtueller Speicher: TLB

- Zur Beschleunigung des Zugriffs auf die Seitentabelle wird die **MMU mit einem Cache** ausgestattet, dem Assoziativspeicher oder TLB (*Translation Lookaside Buffer)*.





FH Bielefeld
University of
Applied Sciences

Seitenersetzung: Strategien

- Die **Seitenersetzungsstrategie** bestimmt, welcher belegte Seitenrahmen bei einem Seitenfehler aus dem Speicher entfernt wird, damit eine neue Seite eingelagert werden kann.
- Ziel aller Strategien: **möglichst wenig Transfers** von Seiten zwischen Platte und Speicher (Effizienz).
- Ähnliche Probleme existieren in anderen Bereichen der Informatik, z.B.
  - Cache-Speicher für Datenzugriffe in Datenbanken oder auf Prozessorebene,
  - Zwischenspeichern von WWW-Seiten auf einem Web-Server.
  - Lösungen sind also auf andere Gebiete übertragbar.

FH Bielefeld
University of
Applied Sciences

Seitenersetzung: Optimale Strategie

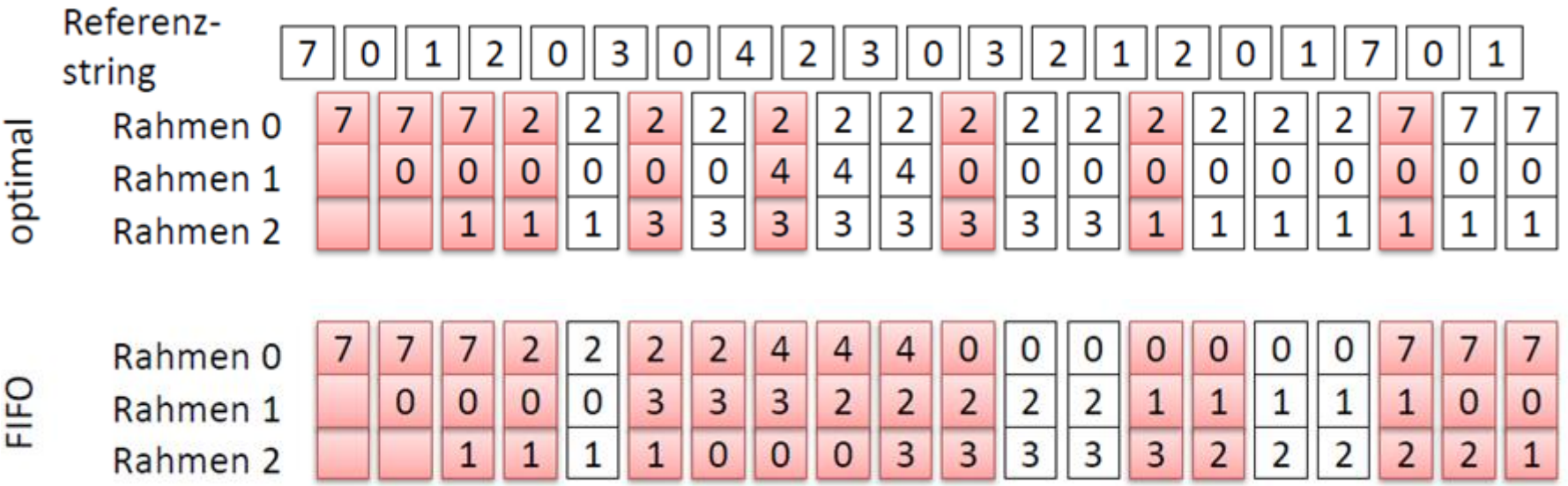
- Die **optimale** Strategie lagert bei einem Seitenfehler die Seite aus, auf die in der **Zukunft** am spätesten zugegriffen wird.
- Problem: Der Algorithmus ist nicht kausal und damit *nicht realisierbar*. Er müsste in die Zukunft schauen können.
- Nutzen des optimalen Algorithmus: Er kann bei definierten Programmabläufen und Eingangsdaten als Referenz für andere Algorithmen dienen, um deren Qualität zu beurteilen.
- Beispiel:

Referenz-string	7	0	1	2	0	3	0	4	2	3	0	3	2	1	2	0	1	7	0	1
Rahmen 0	7	7	7	2	2	2	2	2	2	2	2	2	2	2	2	2	2	7	7	7
Rahmen 1		0	0	0	0	0	0	4	4	4	0	0	0	0	0	0	0	0	0	0
Rahmen 2			1	1	1	3	3	3	3	3	3	3	3	1	1	1	1	1	1	1

FH Bielefeld
University of
Applied Sciences

Seitenersetzung: FIFO Strategie

- First-In, First-Out: Es wird immer die **älteste Seite** im Speicher **verdrängt**
- Vorteil: FIFO-Schlange einfach zu implementieren
- Nachteil: Die älteste Seite kann die am häufigsten benötigte sein
- Beispiel:



FH Bielefeld
University of
Applied Sciences

Seitenersetzung: LRU Strategie (1)

Least-Recently-Used (LRU) Algorithmus:

- LRU ist eine gute Annäherung an die optimale Strategie.
- Annahme:
  - Die Seite, die in der **Vergangenheit** häufig benutzt wurde, wird **auch in Zukunft** häufig benutzt.
  - **Lokalität** der Ausführung.
- Algorithmus: Bei einem Seitenfehler wird die **am längsten unbenutzte Seite ausgelagert**.
- Implementierung ist **schwierig** und ineffizient, da bei jedem Speicherzugriff die *Seitenliste neu sortiert* werden muss.

FH Bielefeld
University of
Applied Sciences

Lokalitätsprinzip

- Prozesse weisen zeitliche und räumliche Lokalität auf:
  - **zeitlich**: kürzlich angesprochene Adresse wird in naher Zukunft wieder angesprochen.
  - Gründe: Schleifen, Unterprogramme, Stacks, Zählvariable
  - **räumlich**: Adressen in der Nachbarschaft kürzlich angesprochener Adressen werden mit größerer Wahrscheinlichkeit angesprochen als weiter entfernte.
  - Gründe: Durchlaufen von Feldern, sequentieller Code-Zugriff

FH Bielefeld
University of
Applied Sciences

Seitenersetzung: LRU Strategie (2)

Beispiel für LRU-Algorithmus:

	Referenzstring	7	0	1	2	0	3	0	4	2	3	0	3	2	1	2	0	1	7	0	1
optimal	Rahmen 0	7	7	7	2	2	2	2	2	2	2	2	2	2	2	2	2	2	7	7	7
	Rahmen 1		0	0	0	0	0	0	4	4	4	0	0	0	0	0	0	0	0	0	0
	Rahmen 2			1	1	1	3	3	3	3	3	3	3	3	1	1	1	1	1	1	1
FIFO	Rahmen 0	7	7	7	2	2	2	2	4	4	4	0	0	0	0	0	0	0	7	7	7
	Rahmen 1		0	0	0	0	3	3	3	2	2	2	2	2	1	1	1	1	1	0	0
	Rahmen 2			1	1	1	1	0	0	0	3	3	3	3	3	2	2	2	2	2	1
LRU	Rahmen 0	7	7	7	2	2	2	2	4	4	4	0	0	0	1	1	1	1	1	1	1
	Rahmen 1		0	0	0	0	0	0	0	0	3	3	3	3	3	3	0	0	0	0	0
	Rahmen 2			1	1	1	3	3	3	2	2	2	2	2	2	2	2	2	7	7	7

optimal: 9 Ersetzungen
FIFO: 15 Ersetzungen
LRU: 12 Ersetzungen

FH Bielefeld
University of
Applied Sciences

Vorlesung

Vielen Dank für Ihre Aufmerksamkeit

Dozent

Prof. Dr.-Ing.
Martin Hoffmann
martin.hoffmann@fh-bielefeld.de