Betriebssysteme (BS)

08. Speicherverwaltung

https://sys.cs.tu-dortmund.de/de/lehre/bs/

25.05.2022

Peter Ulbrich

peter.ulbrich@tu-dortmund.de
bs-problems@ls12.cs.tu-dortmund.de
https://sys.cs.tu-dortmund.de/de/lehre/kummerkasten

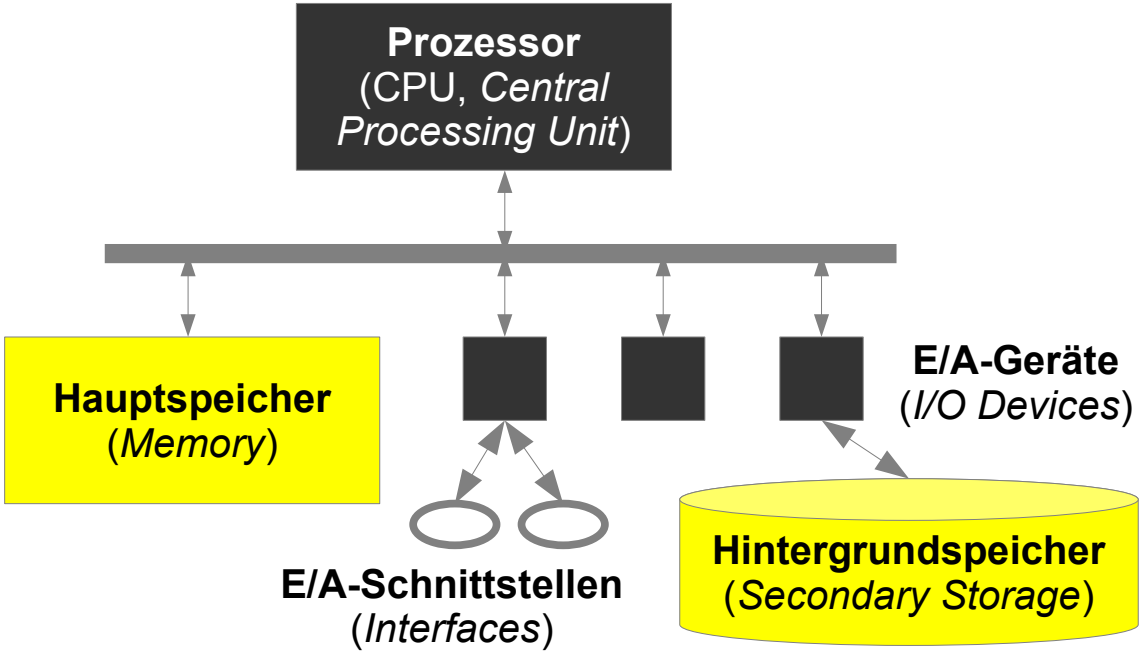
Basierend auf *Betriebssysteme* von Olaf Spinczyk, Universität Osnabrück

technische universität
dortmund

arbeitsgruppe
systemsoftware

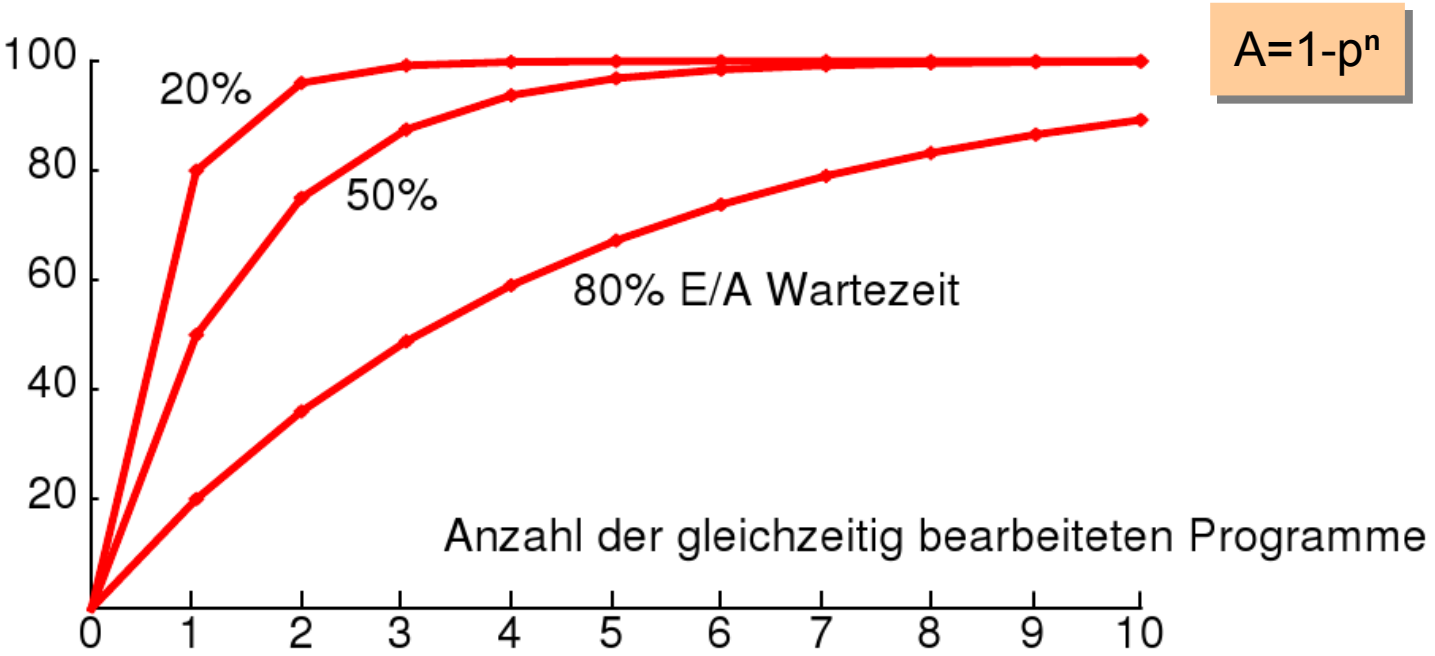
Wiederholung: Betriebsmittel

- Das Betriebssystem hat folgende Aufgaben:
  - Verwaltung der Betriebsmittel des Rechners
  - Schaffung von Abstraktionen, die Anwendungen einen einfachen und effizienten Umgang mit Betriebsmitteln erlauben
- **Bisher: Prozesse**
  - Konzept zur Abstraktion von der realen CPU
- **Nun: Speicher**
  - Verwaltung von Haupt- und Hintergrundspeicher



Wiederholung: Mehrprogrammbetrieb

- CPU-Auslastung unter Annahme einer bestimmten E/A-Wartewahrscheinlichkeit:



Quelle: Tanenbaum, Moderne Betriebssysteme

➔ **Mehrprogrammbetrieb ist essentiell für eine hohe Auslastung**

- Beim Starten und Beenden der Prozesse muss dynamisch Speicher zugewiesen bzw. zurückgenommen werden!

Inhalt

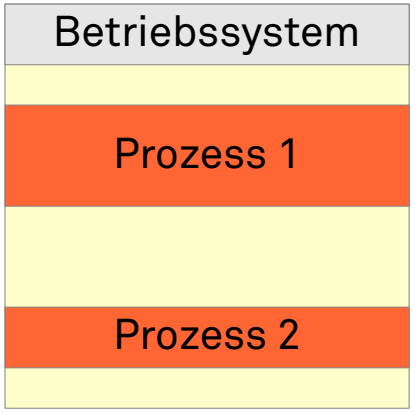
- Grundlegende Aufgaben der Speicherverwaltung
  - Anforderungen
  - Strategien
- Speichervergabe
  - Platzierungsstrategien
- Speicherverwaltung bei Mehrprogrammbetrieb
  - Ein-/Auslagerung
  - Relokation
- Segmentbasierte Adressabbildung
- Seitenbasierte Adressabbildung
- Zusammenfassung

Tanenbaum
3: Speicherverwaltung

Silberschatz
8: Memory-Management Strategies

Anforderungen

- Mehrere Prozesse benötigen Hauptspeicher
  - Prozesse liegen an verschiedenen Stellen im Hauptspeicher.
  - Schutzbedürfnis des Betriebssystems und der Prozesse untereinander
  - Speicher reicht eventuell nicht für alle Prozesse.



Das Betriebssystem und zwei Anwendungs-prozesse im Haupt-speicher

➔ **Freie Speicherbereiche** kennen, verwalten und vergeben

➔ **Ein- und Auslagern** von Prozessen

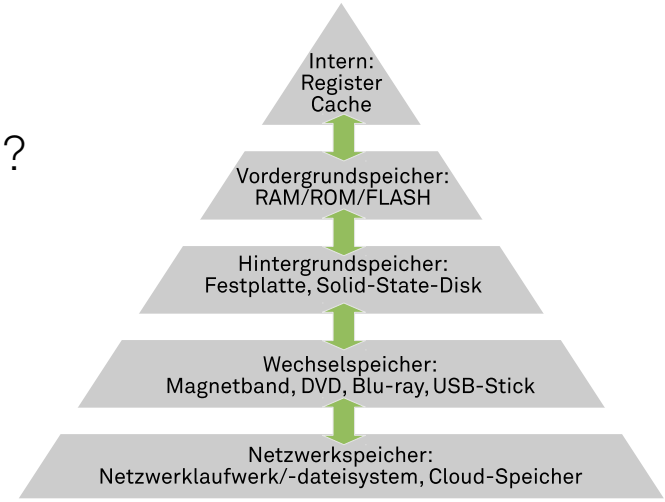
➔ **Relokation** von Programmbefehlen

➔ **Hardwareunterstützung** ausnutzen

Grundlegende Politiken/Strategien

... auf jeder Ebene der Speicherhierarchie:

- **Platzierungsstrategie** (placement policy)
  - **Woher** soll benötigter Speicher genommen werden?
    - wo der Verschnitt am kleinsten/größten ist
    - egal, weil Verschnitt zweitrangig ist
- **Ladestrategie** (fetch policy)
  - **Wann** sind Speicherinhalte einzulagern?
    - auf Anforderung oder im Voraus
- **Ersetzungsstrategie** (replacement policy)
  - **Welche** Speicherinhalte sind ggf. zu verdrängen, falls der Speicher knapp wird?
    - das älteste, am seltensten genutzte
    - das am längsten ungenutzte

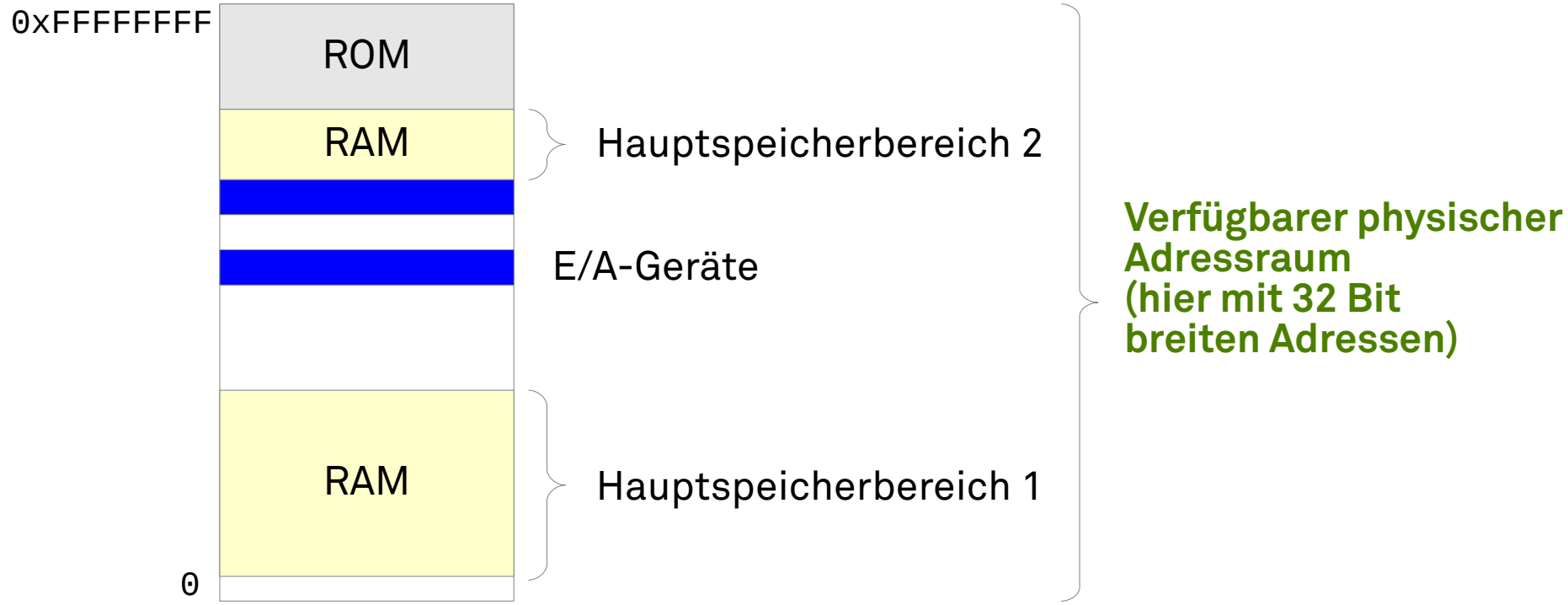


Inhalt

- Grundlegende Aufgaben der Speicherverwaltung
  - Anforderungen
  - Strategien
- **Speichervergabe**
  - Platzierungsstrategien
- Speicherverwaltung bei Mehrprogrammbetrieb
  - Ein-/Auslagerung
  - Relokation
- Segmentbasierte Adressabbildung
- Seitenbasierte Adressabbildung
- Zusammenfassung

Speichervergabe: Problemstellung

- Verfügbarer (physischer) Speicher



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

Speichervergabe: Problemstellung

Belegung des verfügbaren Hauptspeichers durch ...

- **Benutzerprogramme**
  - Programmbefehle (*Text*)
  - Programmdaten (*Data*)
  - Dynamische Speicheranforderungen (*Stack*, *Heap*)
- **Betriebssystem**
  - Betriebssystemcode und -daten
  - Prozesskontrollblöcke
  - Datenpuffer für Ein-/Ausgabe
  - ...

➔ **Zuteilung des Speichers nötig**

Statische Speicherzuteilung

- Feste Bereiche für Betriebssystem und Benutzerprogramme
- **Probleme:**
  - Grad des Mehrprogrammbetriebs begrenzt
  - Begrenzung anderer Ressourcen (z.B. Bandbreite bei Ein-/Ausgabe wegen zu kleiner Puffer)
  - Ungenutzter Speicher des Betriebssystems kann von Anwendungsprogrammen nicht genutzt werden und umgekehrt.

➔ **Dynamische Speicherzuteilung einsetzen**

Dynamische Speicherzuteilung

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

➔ **Platzierungsstrategien nötig**

  - Besonders wichtig dabei: **Freispeicherverwaltung**

Freispeicherverwaltung

- Freie (evtl. auch belegte) Segmente des Speichers müssen repräsentiert werden
- **Bitlisten**

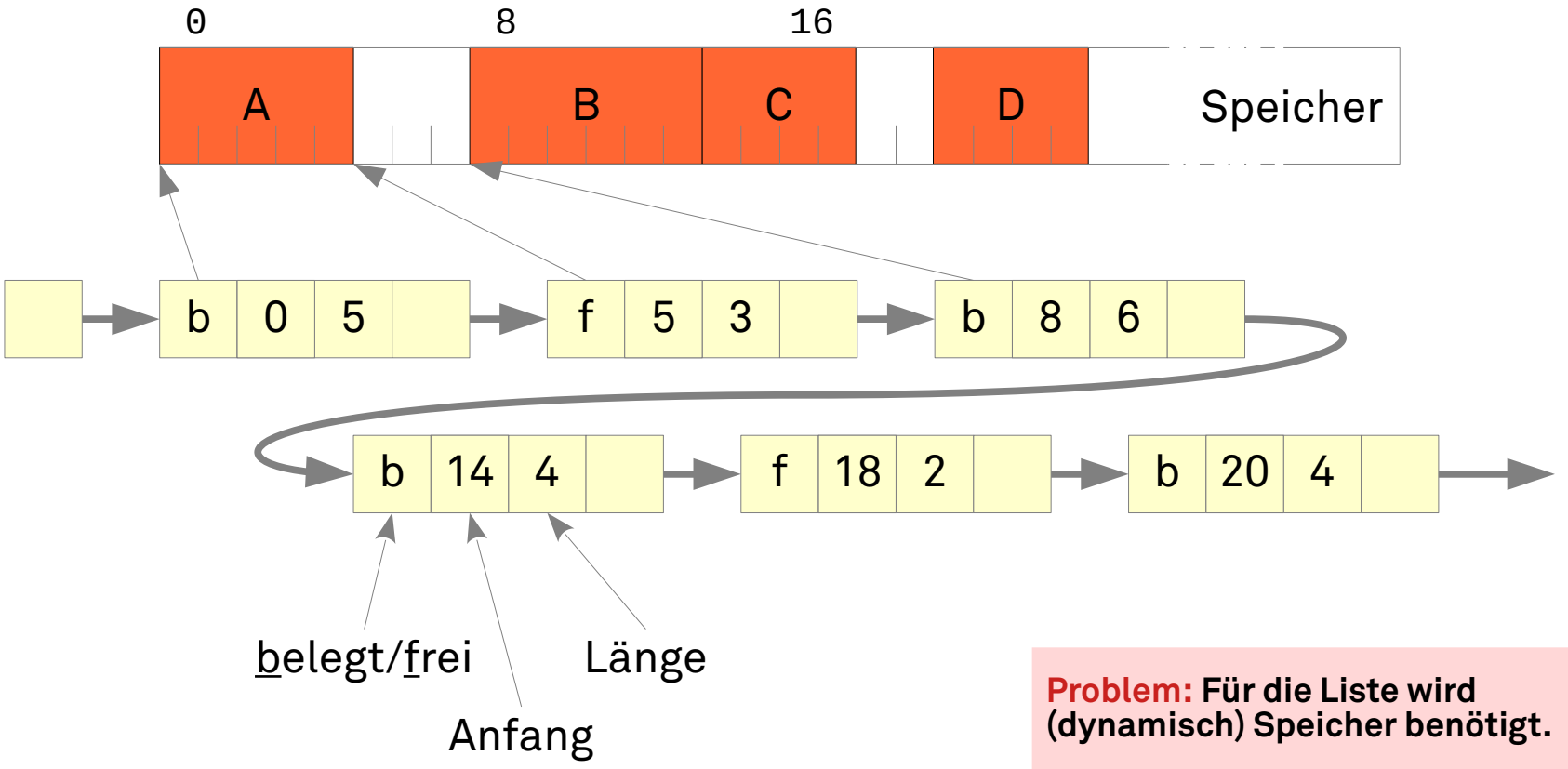
Bitliste markiert belegte Speicherbereiche

Speichereinheiten gleicher Größe
(z.B. 1 Byte, 64 Byte, 1024 Byte)

Probleme:
- **Bitliste kostet u. U. viel Speicher.**
- **Bei der Freigabe muss man die Größe des freizugebenden Speichers kennen bzw. mit angeben.**

Freispeicherverwaltung (2)

- **Verkettete Liste**

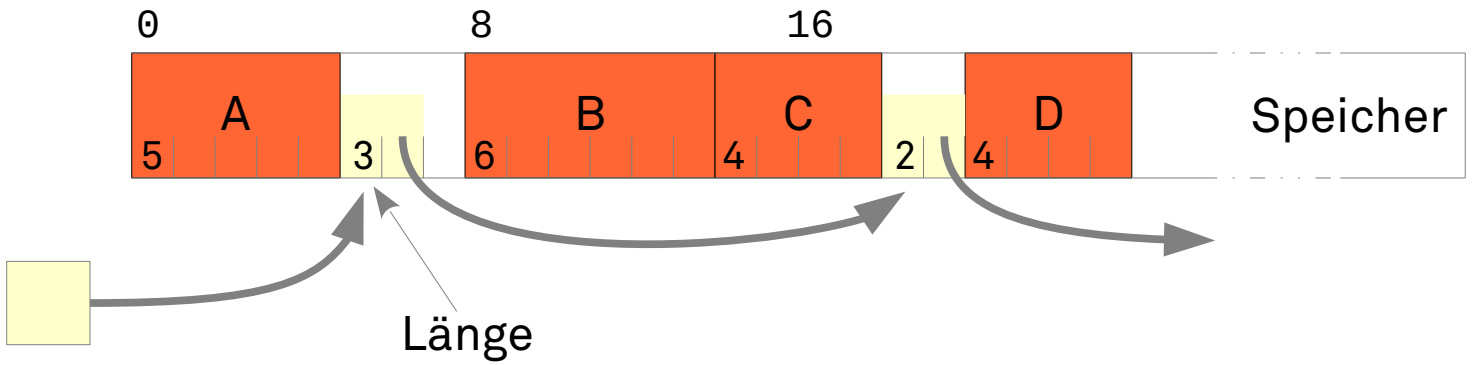


Problem: **Für die Liste wird (dynamisch) Speicher benötigt.**

Repräsentation von belegten und freien Segmenten

Freispeicherverwaltung (3)

- **Verkettete Liste im freien Speicher**

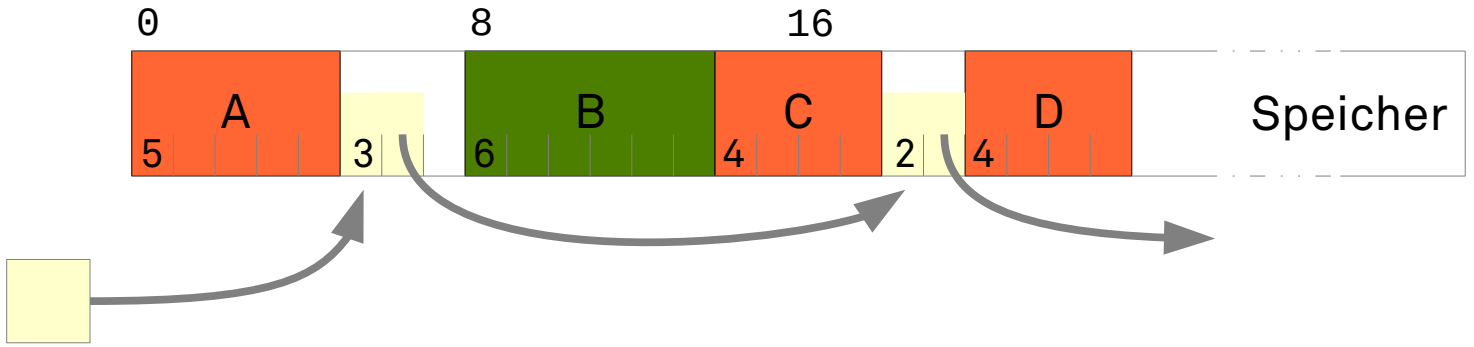


Mindestlückengröße muss garantiert werden

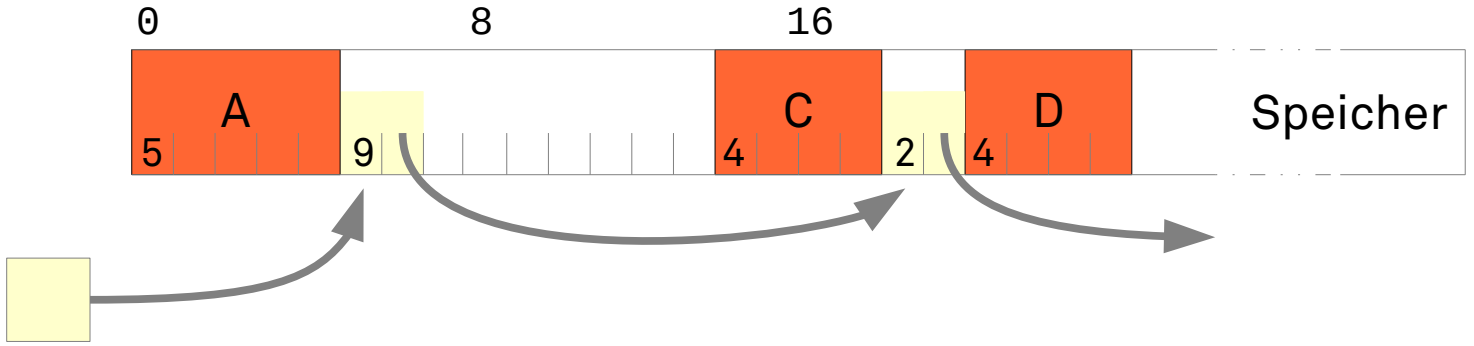
- zur Effizienzsteigerung eventuell Rückwärtsverkettung nötig
- Repräsentation letztlich auch von der Vergabestrategie abhängig

Speicherfreigabe

- **Verschmelzung von Lücken**



Nach Freigabe von B:



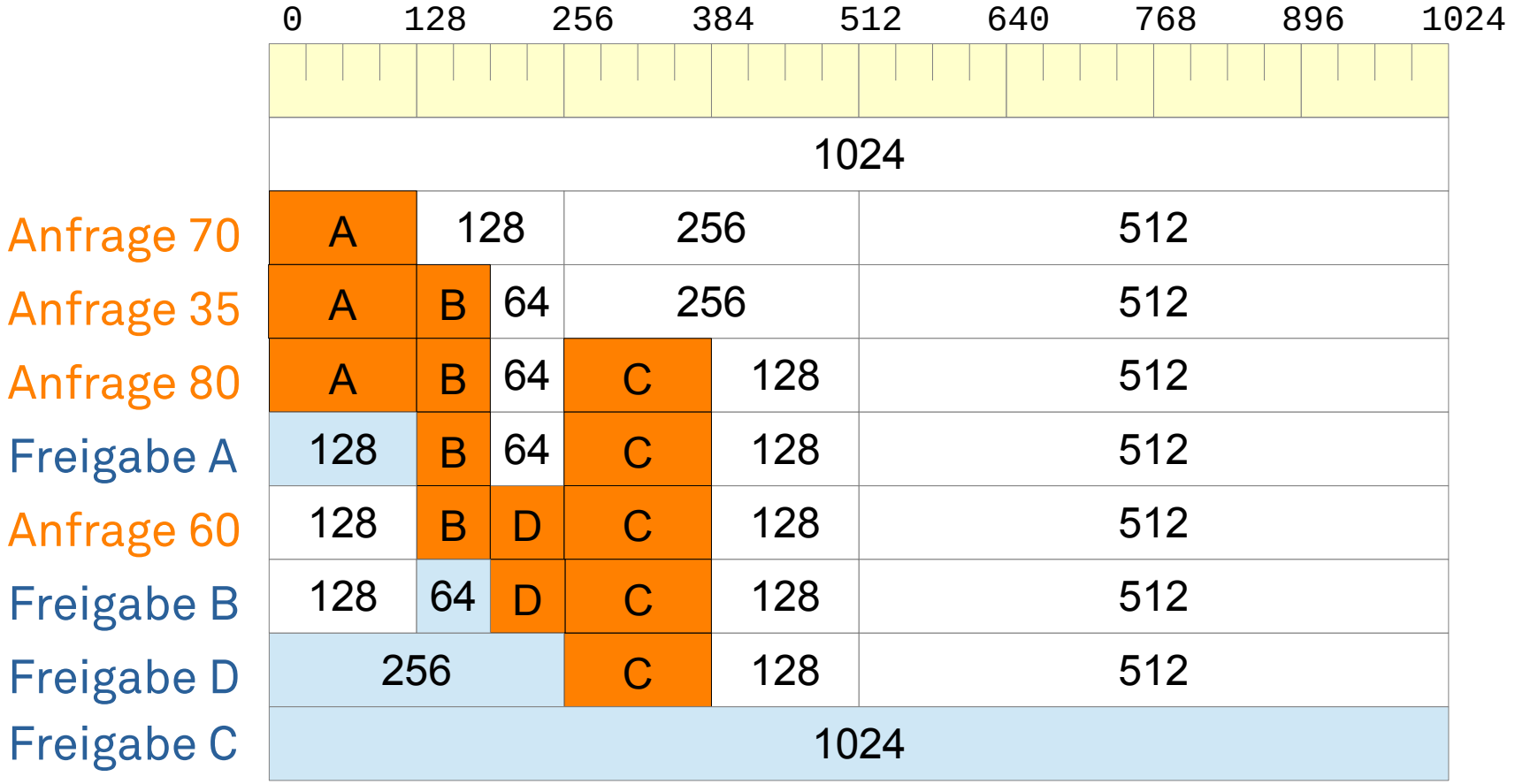
Platzierungsstrategien

... auf der Basis von unterschiedlich sortierten Löcherlisten:

- **First Fit** (Sortierung nach Speicheradresse)
  - erste passende Lücke wird verwendet
- **Rotating First Fit / Next Fit** (Sortierung nach Speicheradresse)
  - wie First Fit, aber Start bei der zuletzt zugewiesenen Lücke
  - vermeidet viele kleine Lücken am Anfang der Liste (wie bei First Fit)
- **Best Fit** (Sortierung nach Lückengröße – kleinste zuerst)
  - kleinste passende Lücke wird gesucht
- **Worst Fit** (Sortierung nach Lückengröße – größte zuerst)
  - größte passende Lücke wird gesucht
- **Probleme:**
  - zu kleine Lücken, Speicherverschnitt

Platzierungsstrategien (2)

- Das ***Buddy*-Verfahren**
- Einteilung in dynamische Bereiche der Größe 2^n



Diskussion: Verschnitt

- **Externer Verschnitt**
  - **Außerhalb** der zugeteilten Speicherbereiche entstehen Speicherfragmente, die nicht mehr genutzt werden können.
  - Passiert bei den listenbasierten Strategien wie ***First Fit*, *Best Fit*, ...**
- **Interner Verschnitt**
  - **Innerhalb** der zugeteilten Speicherbereiche gibt es ungenutzten Speicher.
  - Passiert z.B. bei ***Buddy***, da die Anforderungen auf die nächstgrößere Zweierpotenz aufgerundet werden.

Zwischenfazit: Einsatz der Verfahren

- Einsatz im **Betriebssystem**
  - Verwaltung des Systemspeichers
  - Zuteilung von Speicher an Prozesse und Betriebssystem

z.B. *Buddy*-Allokator in Linux

- Einsatz innerhalb eines **Prozesses**
  - Verwaltung des Haldenspeichers (*Heap*)
  - erlaubt dynamische Allokation von Speicherbereichen durch den Prozess (***malloc*** und ***free***)

typisch: listenbasiert

- Einsatz für Bereiche des **Sekundärspeichers**
  - Verwaltung bestimmter Abschnitte des Sekundärspeichers, z.B. Speicherbereich für Prozessauslagerungen (*swap space*)

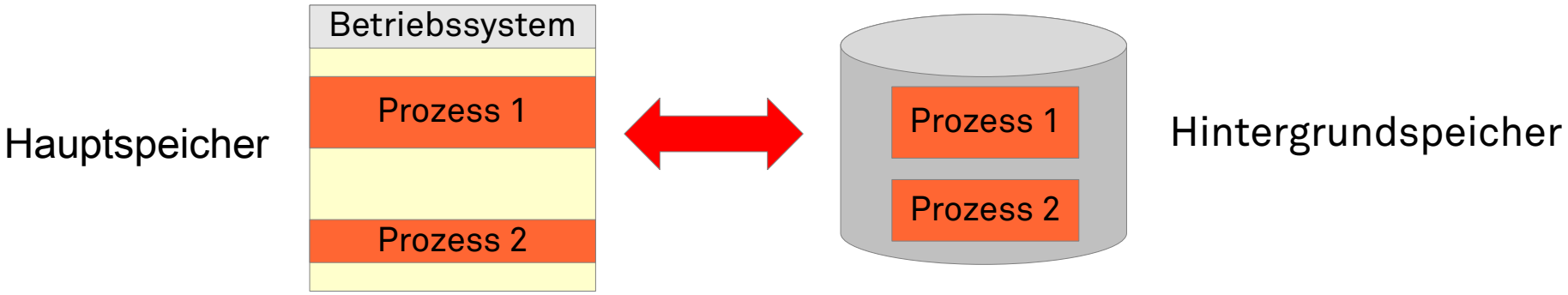
oft: Bitmaps

Inhalt

- Grundlegende Aufgaben der Speicherverwaltung
  - Anforderungen
  - Strategien
- Speichervergabe
  - Platzierungsstrategien
- **Speicherverwaltung bei Mehrprogrammbetrieb**
  - Ein-/Auslagerung
  - Relokation
- Segmentbasierte Adressabbildung
- Seitenbasierte Adressabbildung
- Zusammenfassung

Ein-/Auslagerung (*swapping*)

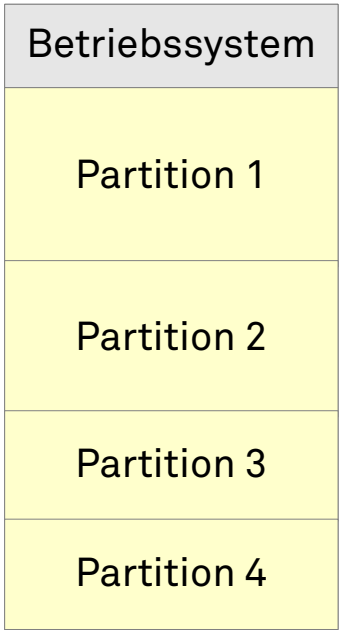
- Segmente eines Prozesses werden auf Hintergrundspeicher ausgelagert und im Hauptspeicher freigegeben
  - z.B. zur Überbrückung von Wartezeiten bei E/A
- Einlagern der Segmente in den Hauptspeicher am Ende der Wartezeit



- **Ein-/Auslagerzeit ist hoch**
  - Latenz der Festplatte (z.B. Positionierung des Schreib-/Lesekopfes)
  - Übertragungszeit

Ein-/Auslagerung (2)

- Adressen im Prozess sind normalerweise statisch gebunden
  - kann nur an dieselbe Stelle im Hauptspeicher wieder eingelagert werden
  - Kollisionen mit eventuell neu im Hauptspeicher befindlichen Segmenten
- Mögliche Lösung:
  Partitionierung des Hauptspeichers
  - In jeder Partition
    läuft nur ein Prozess,
  - Einlagerung erfolgt wieder
    in dieselbe Partition.
  - Großer Nachteil: Speicher kann nicht optimal genutzt werden.



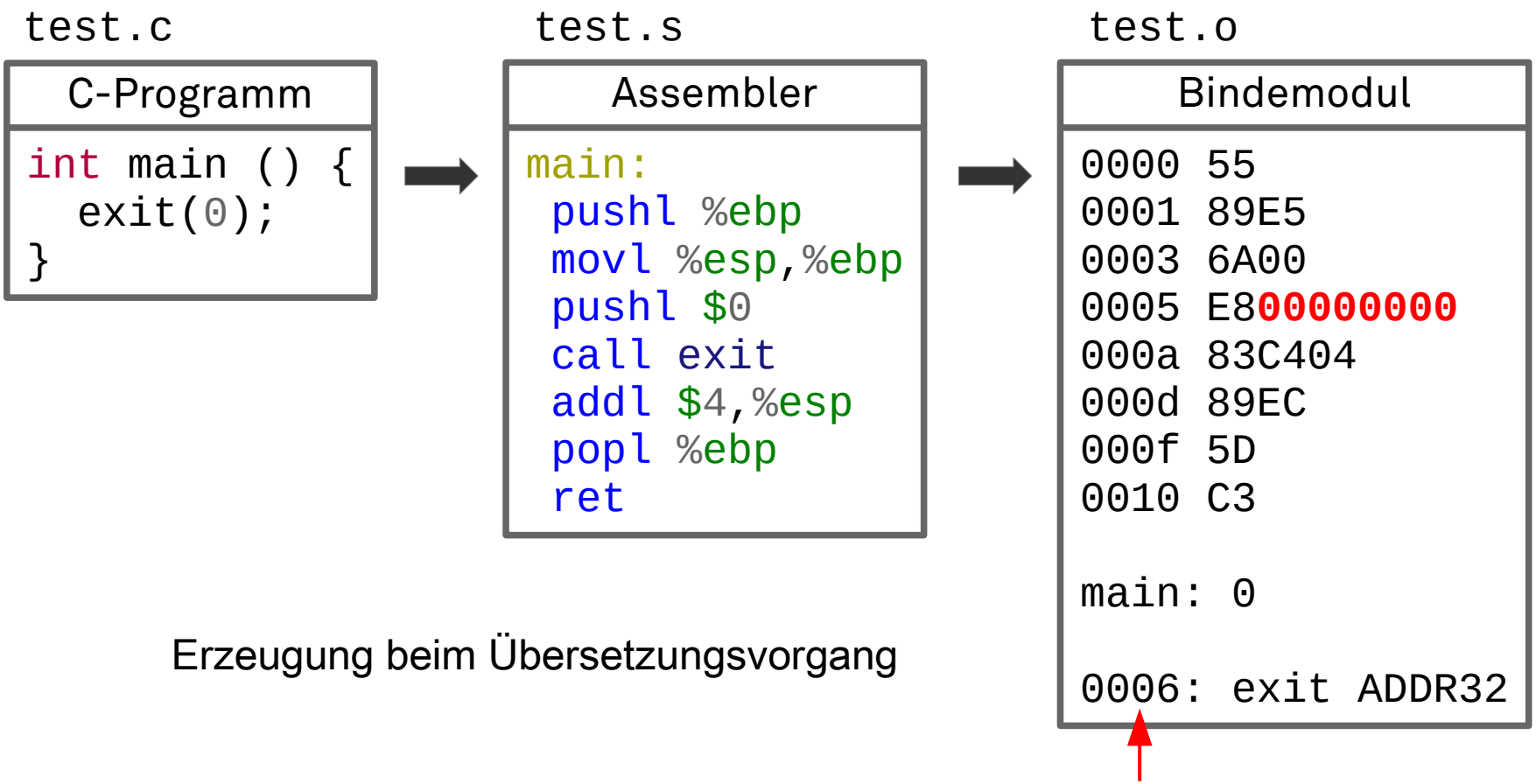
- Besser: Dynamische Belegung und **Programmrelokation**

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

Adressbindung und Relokation (2)

- **Übersetzungsvorgang**
  (Erzeugung der Relokationsinformationen)

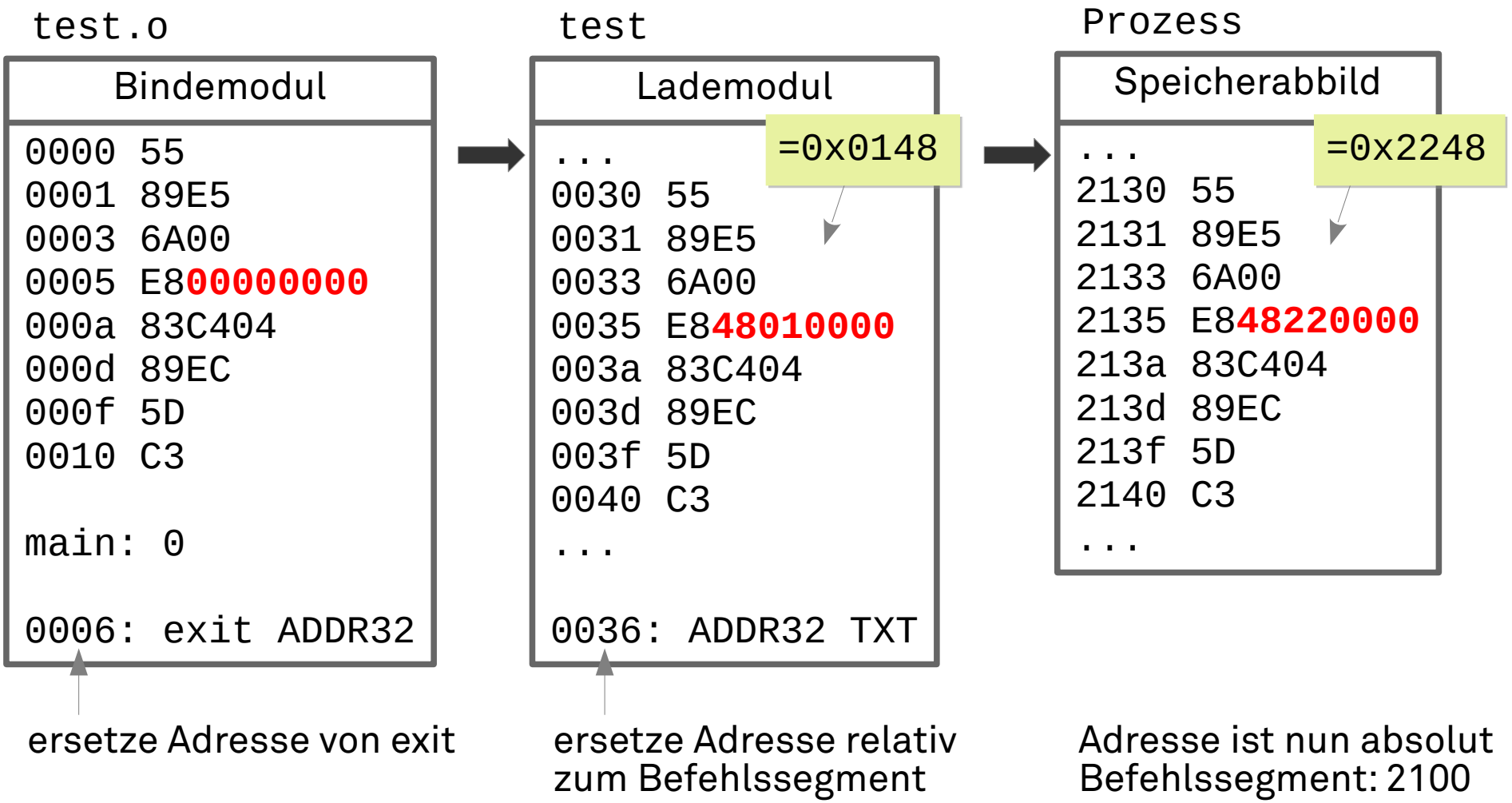


Erzeugung beim Übersetzungsvorgang

Relokationsinformation: „ersetze Adresse von exit"

Adressbindung und Relokation (3)

- **Binde- und Ladevorgang**



Adressbindung und Relokation (4)

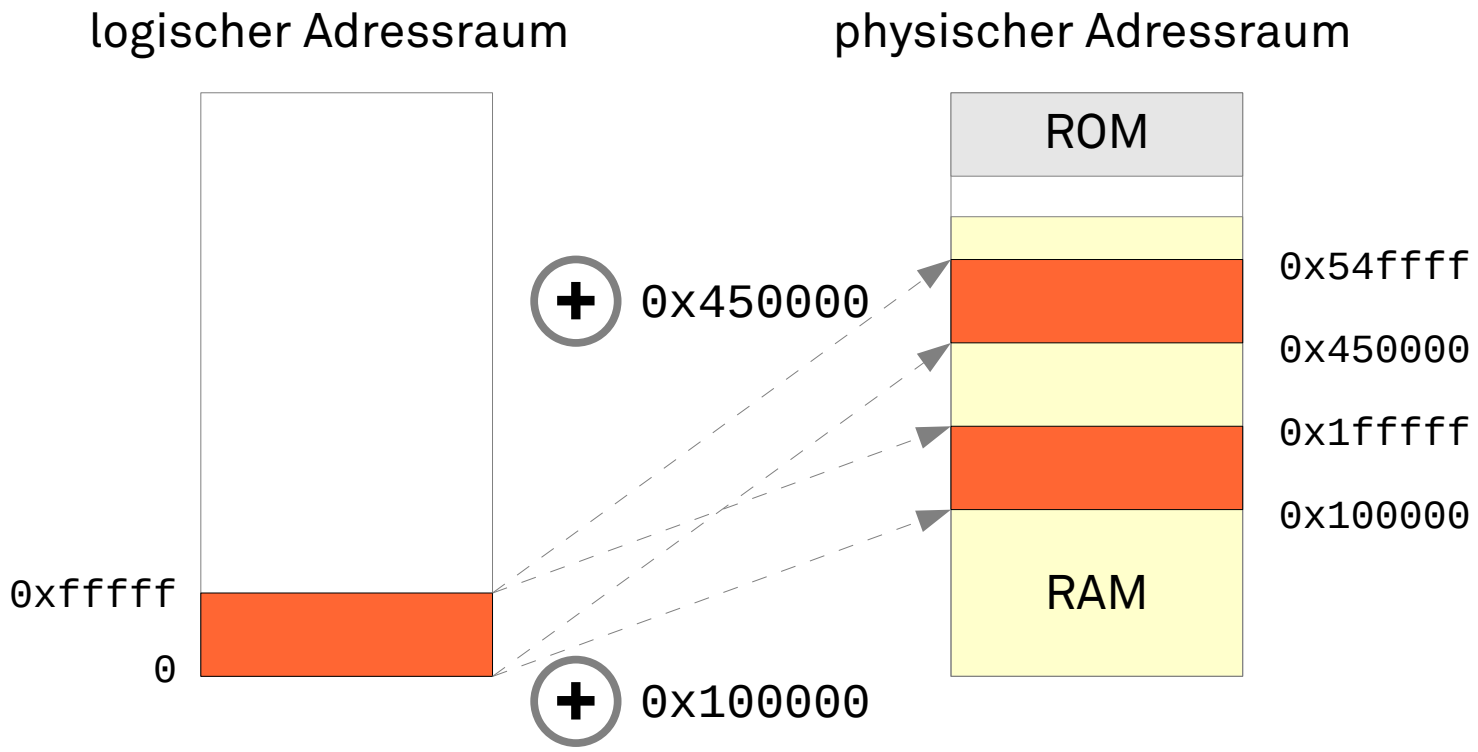
- Relokationsinformation im Bindemodul
  - erlaubt das Binden von Modulen in beliebige Programme
- Relokationsinformation im Lademodul
  - erlaubt das Laden des Programms an beliebige Speicherstellen
  - absolute Adressen werden erst beim Laden generiert
- Dynamisches Binden mit Compiler-Unterstützung
  - Programm benutzt keine absoluten Adressen und kann daher immer an beliebige Speicherstellen geladen werden
    - „***Position Independent Code***“
- Dynamisches Binden mit MMU-Unterstützung:
  - Abbildungsschritt von „logischen“ auf „physische“ Adressen
    - Relokation beim Binden reicht (außer für „***Shared Libraries***“)

Inhalt

- Grundlegende Aufgaben der Speicherverwaltung
  - Anforderungen
  - Strategien
- Speichervergabe
  - Platzierungsstrategien
- Speicherverwaltung bei Mehrprogrammbetrieb
  - Ein-/Auslagerung
  - Relokation
- **Segmentbasierte Adressabbildung**
- Seitenbasierte Adressabbildung
- Zusammenfassung

Segmentierung

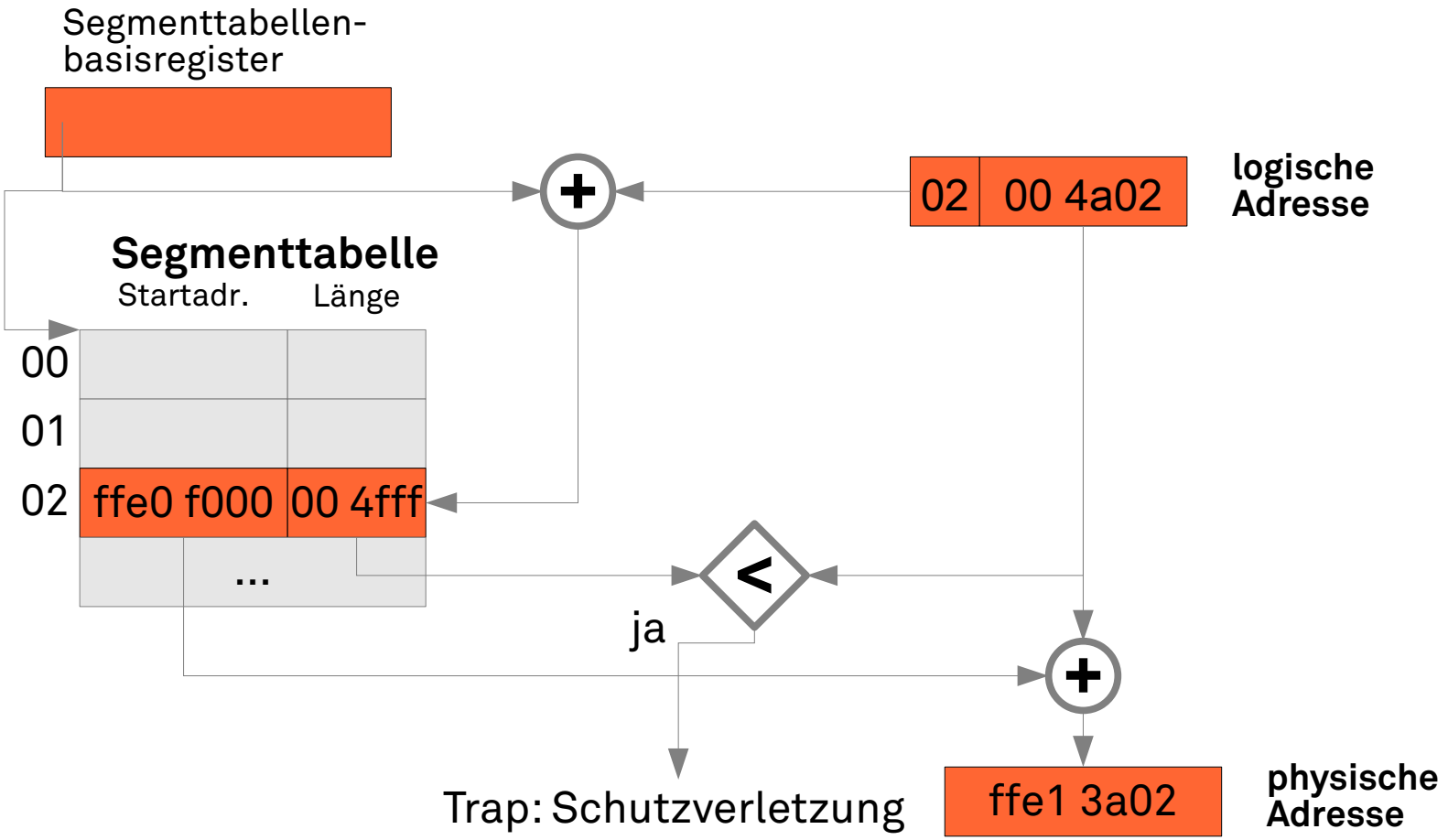
- Hardwareunterstützung:
  Abbildung logischer auf physische Adressen



Das Segment des logischen Adressraums kann an jeder beliebigen Stelle im physischen Adressraum liegen. Das Betriebssystem bestimmt, wo ein Segment im physischen Adressraum tatsächlich liegen soll.

Segmentierung (2)

- Realisierung mit **Übersetzungstabelle** (pro Prozess)



Segmentierung (3)

- Hardwareunterstützung: **MMU** (***Memory Management Unit***)
- Schutz vor Segmentübertretung
  - MMU prüft Rechte zum Lesen, Schreiben und Ausführen von Befehlen
  - ***Trap*** zeigt Speicherverletzung an
  - Programme und Betriebssystem voreinander geschützt
- Prozessumschaltung durch Austausch der Segmentbasis
  - jeder Prozess hat eigene Übersetzungstabelle
- Ein- und Auslagerung vereinfacht
  - nach Einlagerung an beliebige Stelle muss lediglich die Übersetzungstabelle angepasst werden
- Gemeinsame Segmente möglich
  - Befehlssegmente
  - Datensegmente (*Shared Memory*)

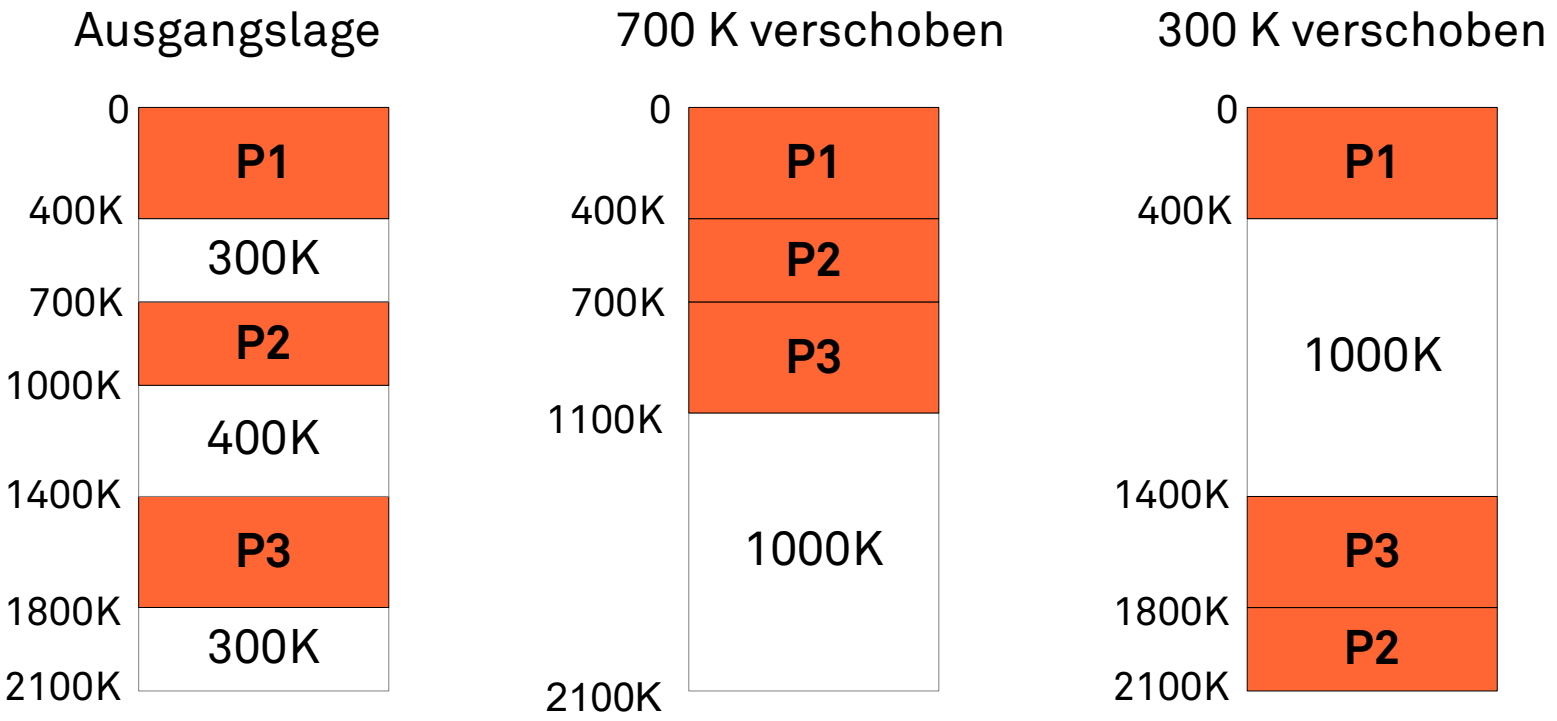
Segmentierung (4)

Probleme ...

- **Fragmentierung** des Speichers durch häufiges Ein- und Auslagern
  - Es entstehen kleine, nicht nutzbare Lücken: externer Verschnitt
- **Kompaktifizieren** hilft
  - Segmente werden verschoben, um Lücken zu schließen
  - Segmenttabelle wird jeweils angepasst
  - kostet aber Zeit
- Lange E/A-Zeiten für Ein- und Auslagerung
  - Nicht alle Teile eines Segments werden gleich häufig genutzt.

Kompaktifizieren

- **Verschieben von Segmenten**
  - Erzeugen von weniger – aber größeren – Lücken
  - Verringern des Verschnitts
  - **aufwendige Operation**, abhängig von der Größe der verschobenen Segmente

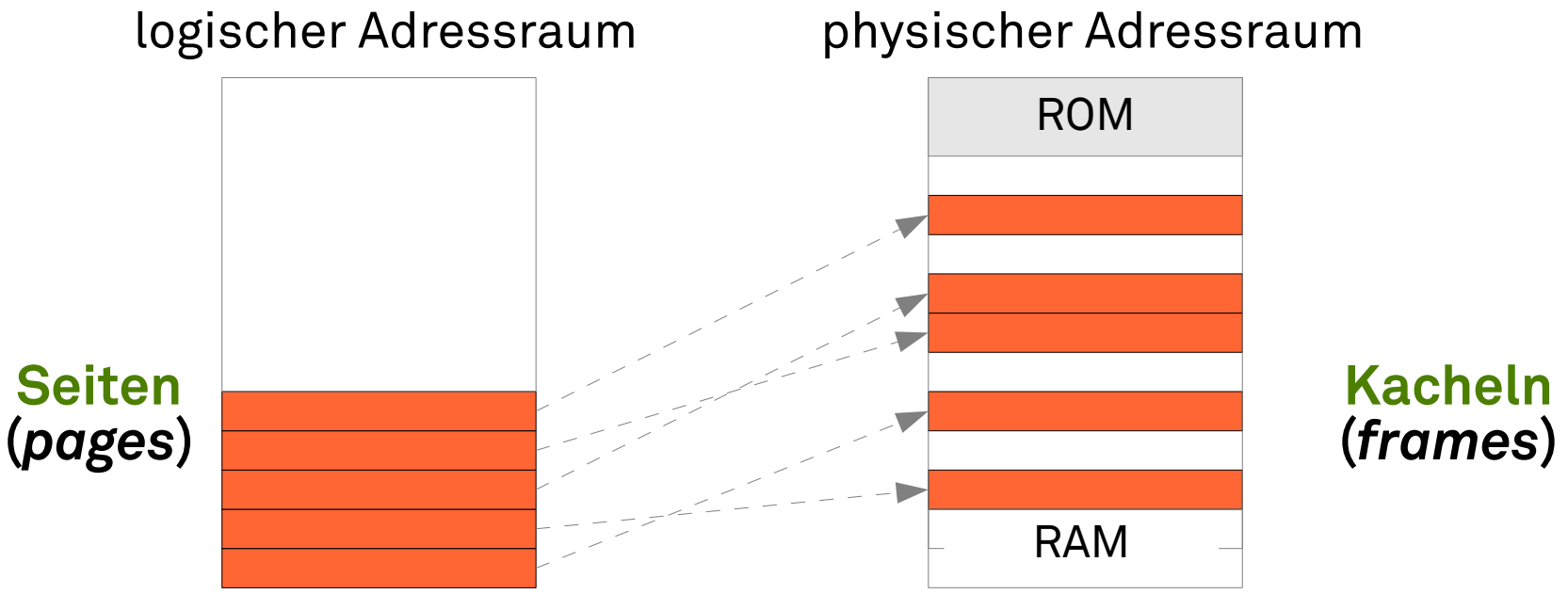


Inhalt

- Grundlegende Aufgaben der Speicherverwaltung
  - Anforderungen
  - Strategien
- Speichervergabe
  - Platzierungsstrategien
- Speicherverwaltung bei Mehrprogrammbetrieb
  - Ein-/Auslagerung
  - Relokation
- Segmentbasierte Adressabbildung
- **Seitenbasierte Adressabbildung**
- Zusammenfassung

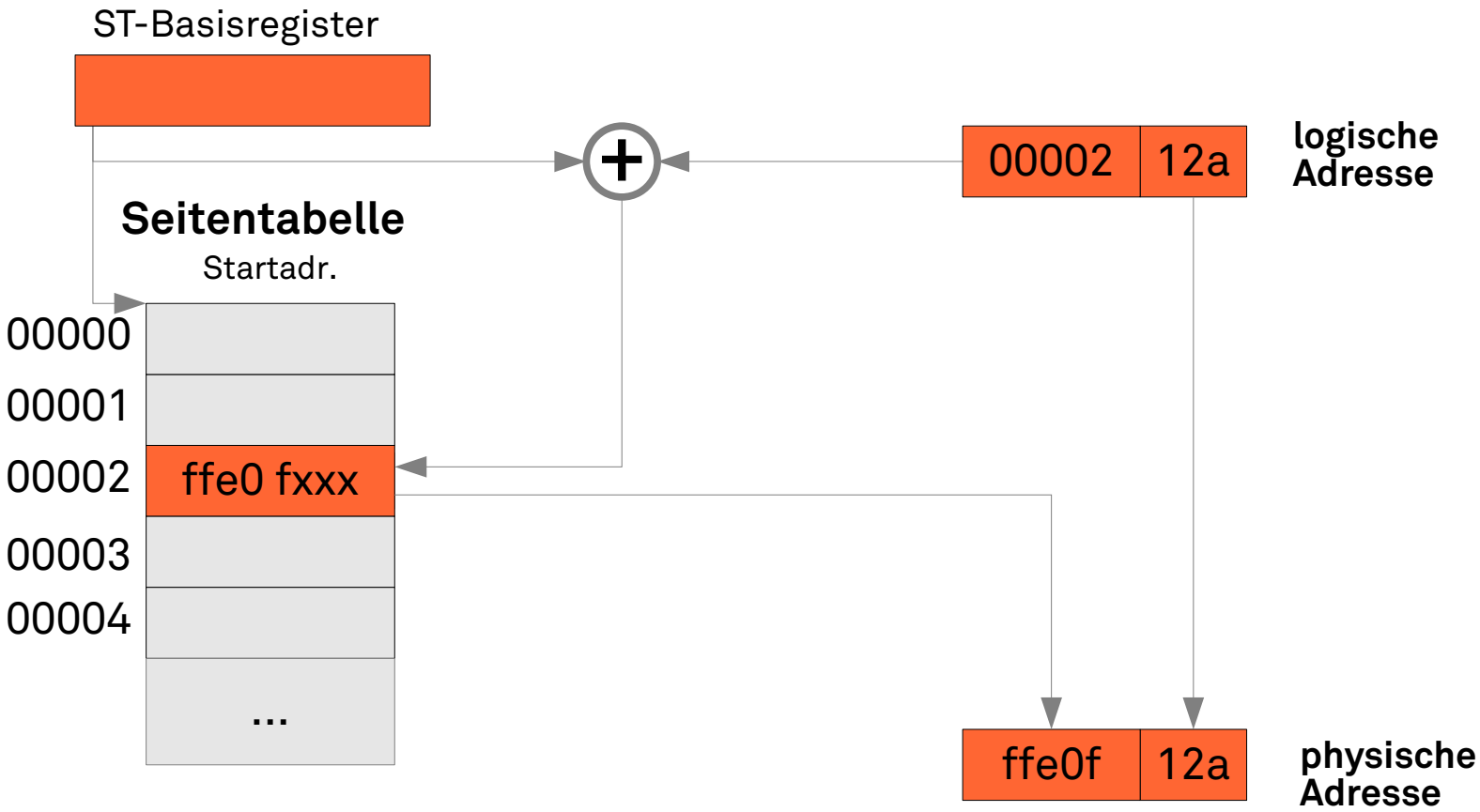
Seitenadressierung (*paging*)

- Einteilung des logischen Adressraums in gleichgroße **Seiten**, die an beliebigen Stellen im physischen Adressraum liegen können
  - Lösung des Fragmentierungsproblems
  - keine Kompaktifizierung mehr nötig
  - vereinfacht Speicherbelegung und Ein-/Auslagerungen



MMU mit Seitentabelle

- Tabelle setzt Seiten in Seitenrahmen (Kacheln) um

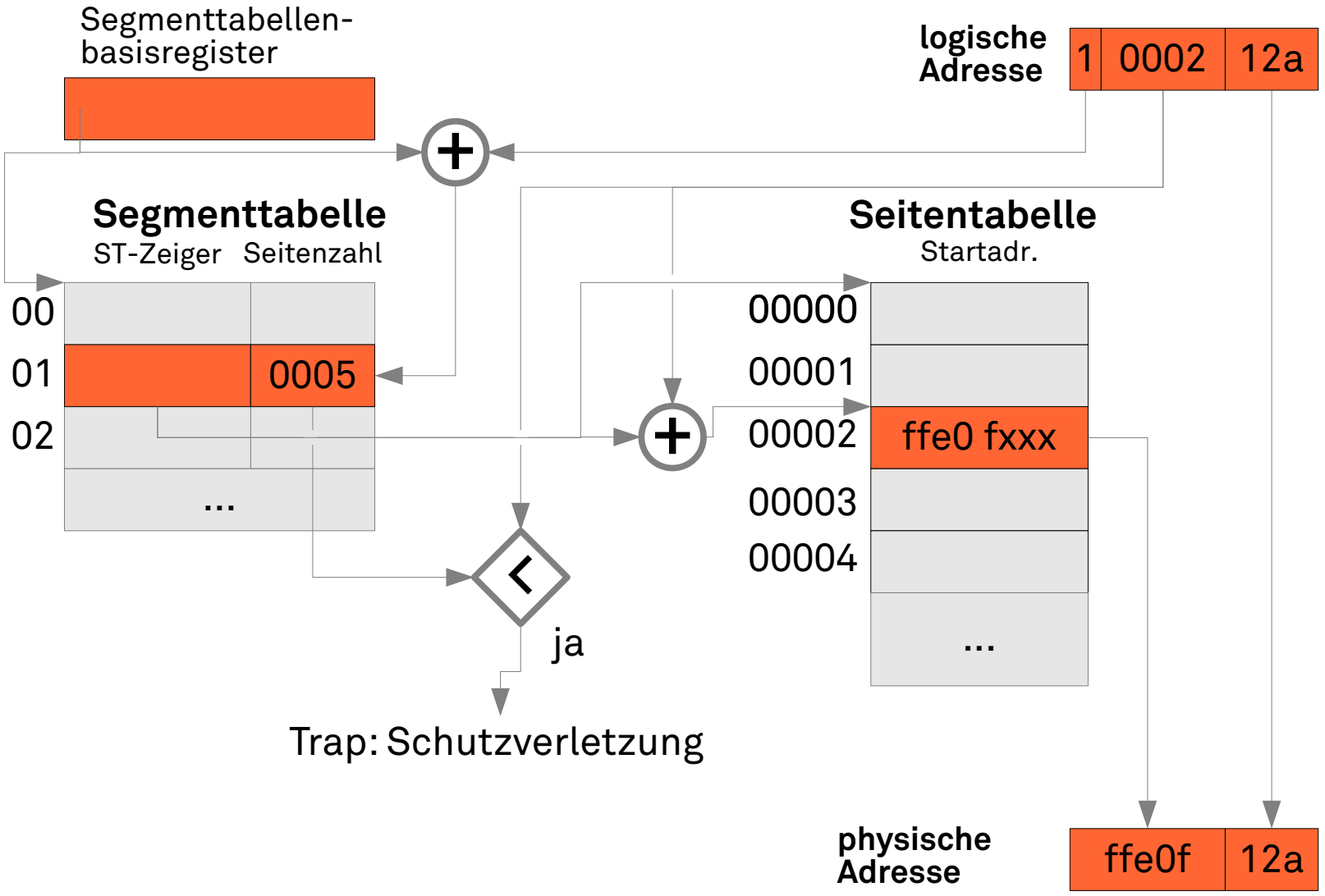


MMU mit Seitentabelle (2)

- Seitenadressierung erzeugt internen Verschnitt
  - letzte Seite eventuell nicht vollständig genutzt
- Seitengröße
  - kleine Seiten verringern internen Verschnitt, vergrößern aber die Seitentabelle (und umgekehrt)
  - übliche Größe: 4096 Bytes (4 KiB)
- große Tabelle, die im Speicher gehalten werden muss
- viele implizite Speicherzugriffe nötig
- nur ein „Segment“ pro Kontext
  - sinngemäße Nutzung des Speichers schwerer zu kontrollieren (push/pop nur auf „Stack“, Ausführung nur von „Text“, …)

➔ **Kombination mit Segmentierung**

Segmentierung und Seitenadressierung



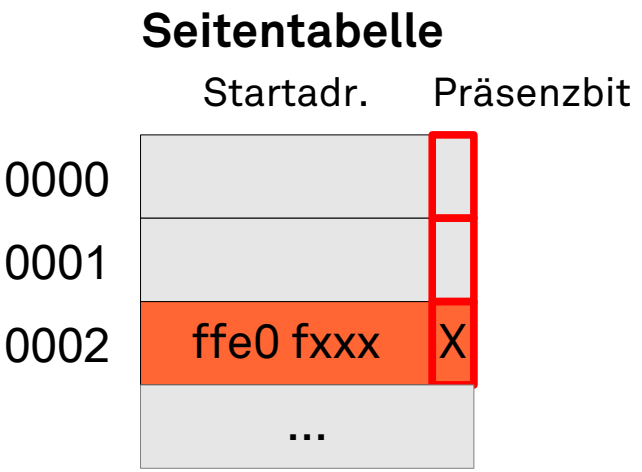
Segmentierung u. Seitenadressierung (2)

- Noch mehr implizite Speicherzugriffe
- Große Tabellen im Speicher
- Vermischung der Konzepte
- Noch immer Ein-/Auslagerung kompletter Segmente

➔ **Mehrstufige Seitenadressierung mit Ein- und Auslagerung**

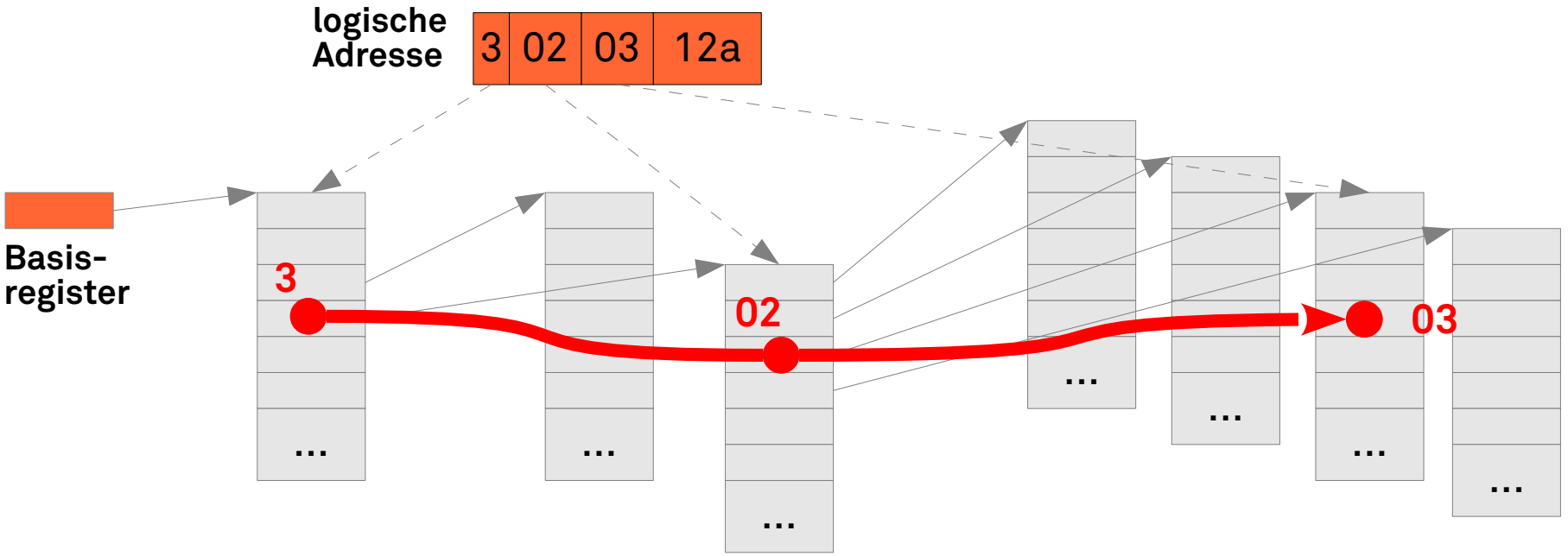
Ein-/Auslagerung von Seiten

- Es ist nicht nötig, ein gesamtes Segment aus- bzw. einzulagern
  - Seiten können einzeln ein- und ausgelagert werden
- Hardware-Unterstützung
  - Ist das Präsenzbit gesetzt, bleibt alles wie bisher.
  - Ist das Präsenzbit gelöscht, wird ein *Trap* ausgelöst (***page fault***).
  - Die *Trap*-Behandlung kann nun für das Laden der Seite vom Hintergrundspeicher sorgen und den Speicherzugriff danach wiederholen (benötigt *HW-Support* in der CPU).



Mehrstufige Seitenadressierung

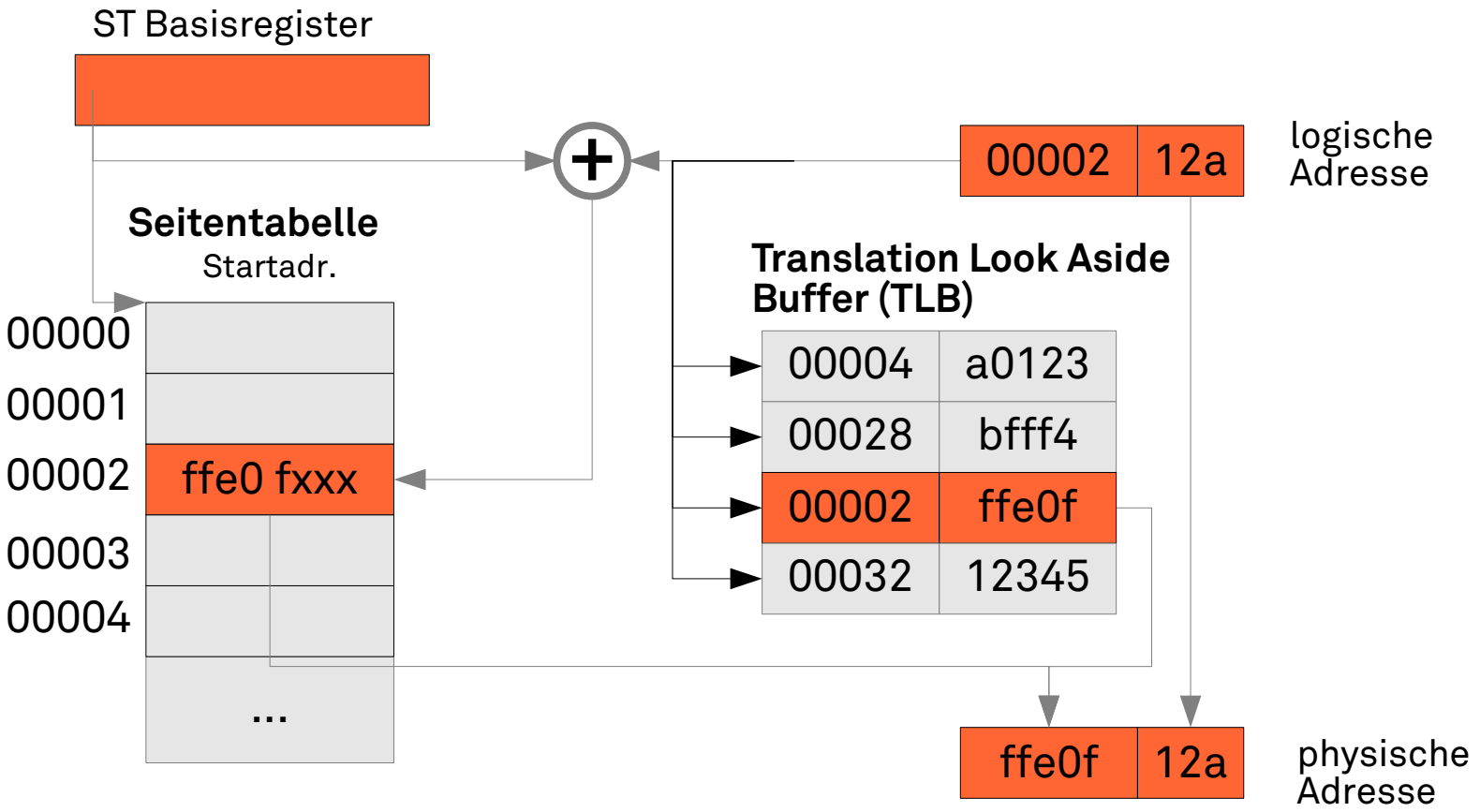
- Beispiel: zweifach indirekte Seitenadressierung



- Präsenzbit auch für jeden Eintrag in den höheren Stufen
  - Tabellen werden aus- und einlagerbar
  - Tabellen können bei Zugriff (=Bedarf) erzeugt werden (spart Speicher!)
- Aber: Noch mehr implizite Speicherzugriffe

Translation Look-Aside Buffer (TLB)

- Schneller Registersatz wird konsultiert, **bevor** auf die Seitentabelle zugegriffen wird:

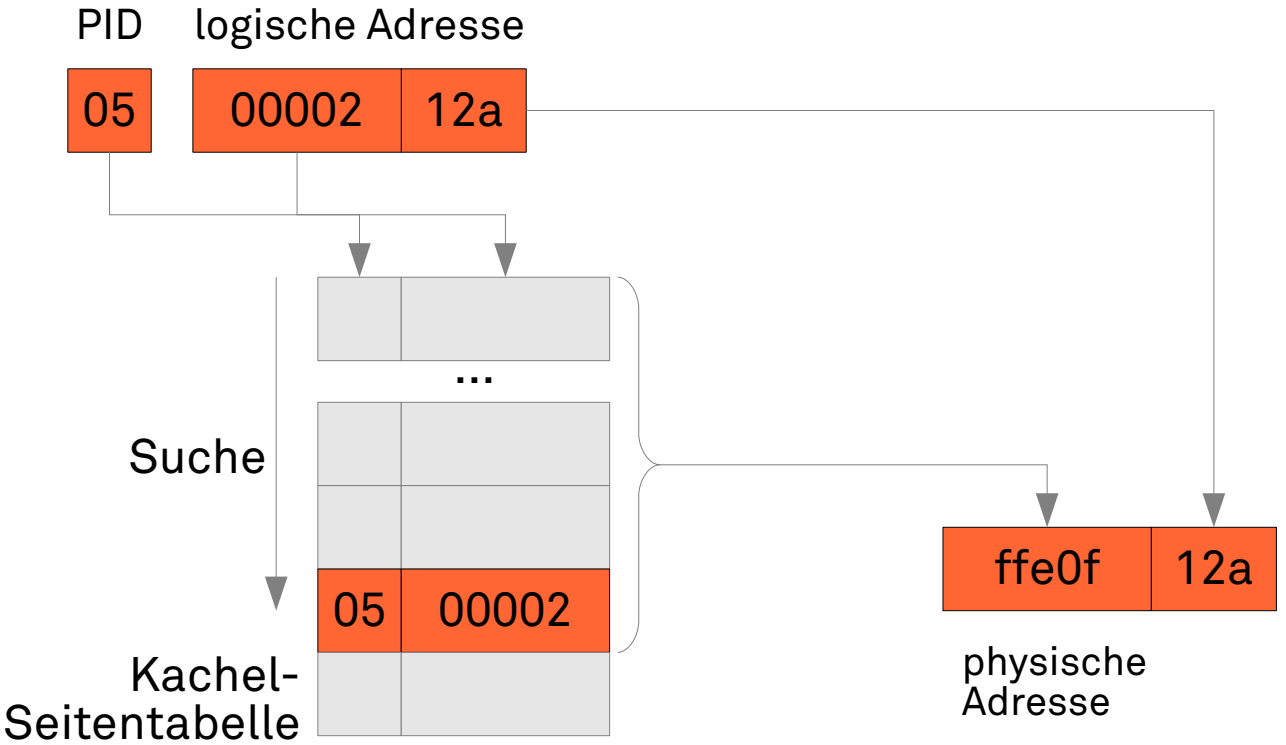


Translation Look-Aside Buffer (2)

- Schneller Zugriff auf Seitenabbildung, falls Information im voll-assoziativen Speicher des TLB
  - keine impliziten Speicherzugriffe nötig
- Bei Kontextwechseln muss TLB gelöscht werden (***flush***)
  - Process-Context ID (PCID): flush bei Intel-CPUs seit ~2010 (und AMD-CPUs ab Zen 3 / seit 2020) nicht mehr notwendig
- Bei Zugriffen auf eine nicht im TLB enthaltene Seite wird die entsprechende Zugriffsinformation in den TLB eingetragen.
  - Ein alter Eintrag muss zur Ersetzung ausgesucht werden.
- TLB-Größe:
  - Intel Core i7: 512 Einträge, Seitengröße 4K
  - UltraSPARC T2: Daten-TLB = 128, Code-TLB = 64, Seitengröße 8K
  - Größere TLBs bei den üblichen Taktraten zur Zeit nicht möglich.

Invertierte Seitentabelle

- Bei großen logischen Adressräumen (z.B. 64 Bit):
  - klassische Seitentabellen sehr groß (oder …)
  - sehr viele Abbildungsstufen
  - Tabellen sehr dünn besetzt
- **Invertierte Seitentabelle** (*Inverted Page Table*)



Invertierte Seitentabelle (2)

- **Vorteile**
  - wenig Platz zur Speicherung der Abbildung notwendig
  - Tabelle kann immer im Hauptspeicher gehalten werden
- **Nachteile**
  - Sharing von Seitenrahmen schwer zu realisieren
  - prozesslokale Datenstrukturen zusätzlich nötig für Seiten, die ausgelagert sind
  - **Suche in der Seitentabelle ist aufwendig**
    - Einsatz von Assoziativspeichern und Hashfunktionen
- Trotz der Nachteile setzen heute viele Prozessorhersteller bei 64-Bit-Architekturen auf diese Form der Adressumsetzung
  - PowerPC, UltraSparc, IA-64, (Alpha), ...
  - Nicht: x86-64/amd64, Arm (AArch64)

Inhalt

- Grundlegende Aufgaben der Speicherverwaltung
  - Anforderungen
  - Strategien
- Speichervergabe
  - Platzierungsstrategien
- Speicherverwaltung bei Mehrprogrammbetrieb
  - Ein-/Auslagerung
  - Relokation
- Segmentbasierte Adressabbildung
- Seitenbasierte Adressabbildung
- **Zusammenfassung**

Zusammenfassung

- Bei der Speicherverwaltung arbeitet das Betriebssystem sehr eng mit der Hardware zusammen.
  - Segmentierung und/oder Seitenadressierung
  - Durch die implizite Indirektion beim Speicherzugriff können Programme und Daten unter der Kontrolle des Betriebssystems im laufenden Betrieb beliebig verschoben werden.
- Zusätzlich sind diverse strategische Entscheidungen zu treffen.
  - Platzierungsstrategie (*First Fit*, *Best Fit*, *Buddy*, ...)
    - Unterscheiden sich bzgl. Verschnitt sowie Belegungs- und Freigabeaufwand.
    - Strategieauswahl hängt vom erwarteten Anwendungsprofil ab.
  - Bei Ein-/Auslagerung von Segmenten oder Seiten:
    - Ladestrategie
    - Ersetzungsstrategie

nächstes Mal mehr dazu