Hochschule **RheinMain**
University of Applied Sciences
Wiesbaden Rüsselsheim

Kap. 9: Speicherverwaltung

Ausgangspunkt

Hochschule RheinMain
University of Applied Sciences
Wiesbaden Rüsselsheim

- Idealerweise sollte Speicher sein...
  - groß
  - schnell
  - nicht flüchtig („geht beim Ausschalten nicht verloren“)
- In der Realität (heute) nicht alle Ziele gleichzeitig zu akzeptablen Preisen mit einem Speichermedium zu erreichen
- Daher: Kombination verschiedener Speicherformen

Die Speicherhierarchie

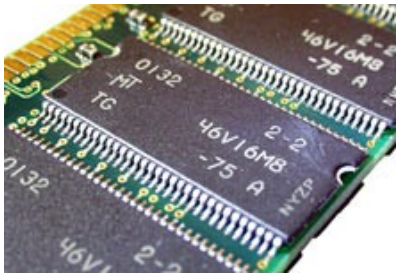
Hochschule RheinMain
University of Applied Sciences
Wiesbaden Rüsselsheim

schneller / teurer

Primärspeicher

- **CPU Register**
- **CPU Cache**
- **Hauptspeicher (RAM)**

direkter, wahlfreier Zugriff durch den Prozessor, sehr schnell



- **Sekundärspeicher (z.B. Festplatte)**

extern; wahlfreier Zugriff auf den Inhalt

- **Tertiärspeicher (z.B. Backup-Bänder)**

extern; langsam, oft nur sequenzieller Zugriff, hohe Kapazität

Anforderungen an Speicherverwaltung

Hochschule RheinMain
University of Applied Sciences
Wiesbaden Rüsselsheim

- **Schutz**
  - Prozesse sollten nicht unerlaubt auf fremde Speicherbereiche zugreifen können
- **Gemeinsame Nutzung**
  - Andererseits soll eine kontrollierte gemeinsame Nutzung von Speicherbereichen möglich sein
    (z.B. 10 gleichzeitige `bash`-Nutzer → Code nicht 10x laden)
- **Relokation**
  - Absolute Adressbezüge im Programmcode z.B. beim Laden in einen (anderen) konkreten Speicherbereich anpassen
- **Speicherorganisation**
  - Unterstützung von Programm-Modularisierung durch *Segmentierung*, abgestufter Schutz z.B. für Daten / Code
  - Ein-/Auslagern von Speicherbereichen zwischen Hauptspeicher und Sekundärspeicher („Festplatte“)

Worum geht es?

Hochschule RheinMain
University of Applied Sciences
Wiesbaden Rüsselsheim

Def Der Speicherverwalter ist der Teil des Betriebssystems, der den Arbeitsspeicher verwaltet.

Aufgaben:

- Verwaltung von freien und belegten Speicherbereichen.
- Zuweisung von Speicherbereichen an Prozesse, wenn diese sie benötigen (Allokation).
- Freigabe nach Benutzung oder bei Prozessende.
- Durchführung von Auslagerungen von Prozessen (oder Teilen) zwischen Arbeitsspeicher und Hintergrundspeicher (Platte) bei Speicherengpässen.

In diesem Kapitel wird eine Folge von einfachen bis hochentwickelten Speicherverwaltungsverfahren betrachtet.

Gliederung

Hochschule RheinMain
University of Applied Sciences
Wiesbaden Rüsselsheim

9.1 Statische Speicherverwaltung

Hochschule **RheinMain**
University of Applied Sciences
Wiesbaden Rüsselsheim

Statische Speicherverwaltung

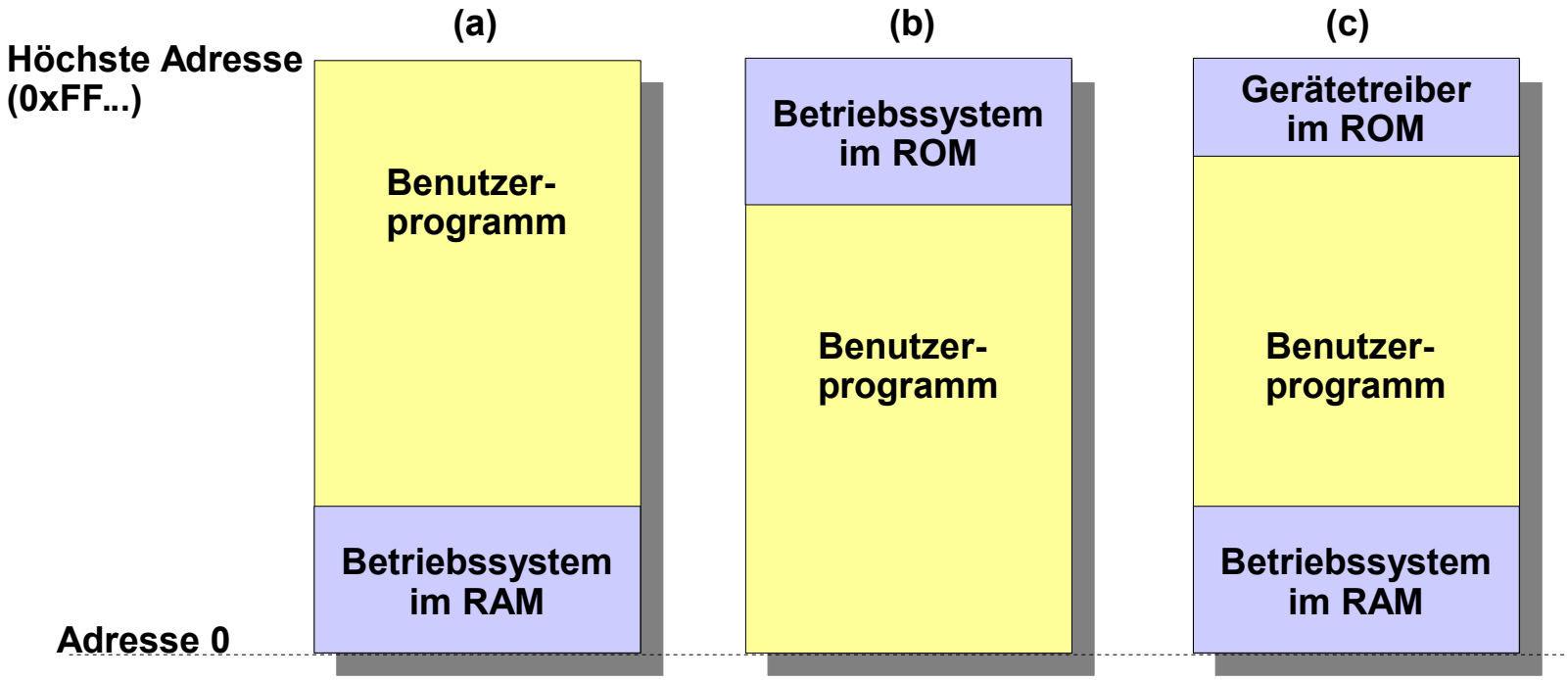
Def

- Verfahren mit fester Zuordnung von Speicher an Prozesse.
- Zuordnung ist keinen Veränderungen während der Laufzeit des Systems oder der Prozesse unterworfen.
- Nicht-statische Speicherverwaltung heißt auch dynamische Speicherverwaltung (ab Kap. 9.2).
- Im folgenden für den historischen Einprogramm- und Mehrprogrammbetrieb betrachtet.
- Heute noch in einfachen Echtzeitanwendungen üblich (z.B. Steuerung einfacher Maschinen).

9.1.1 Einprogrammbetrieb

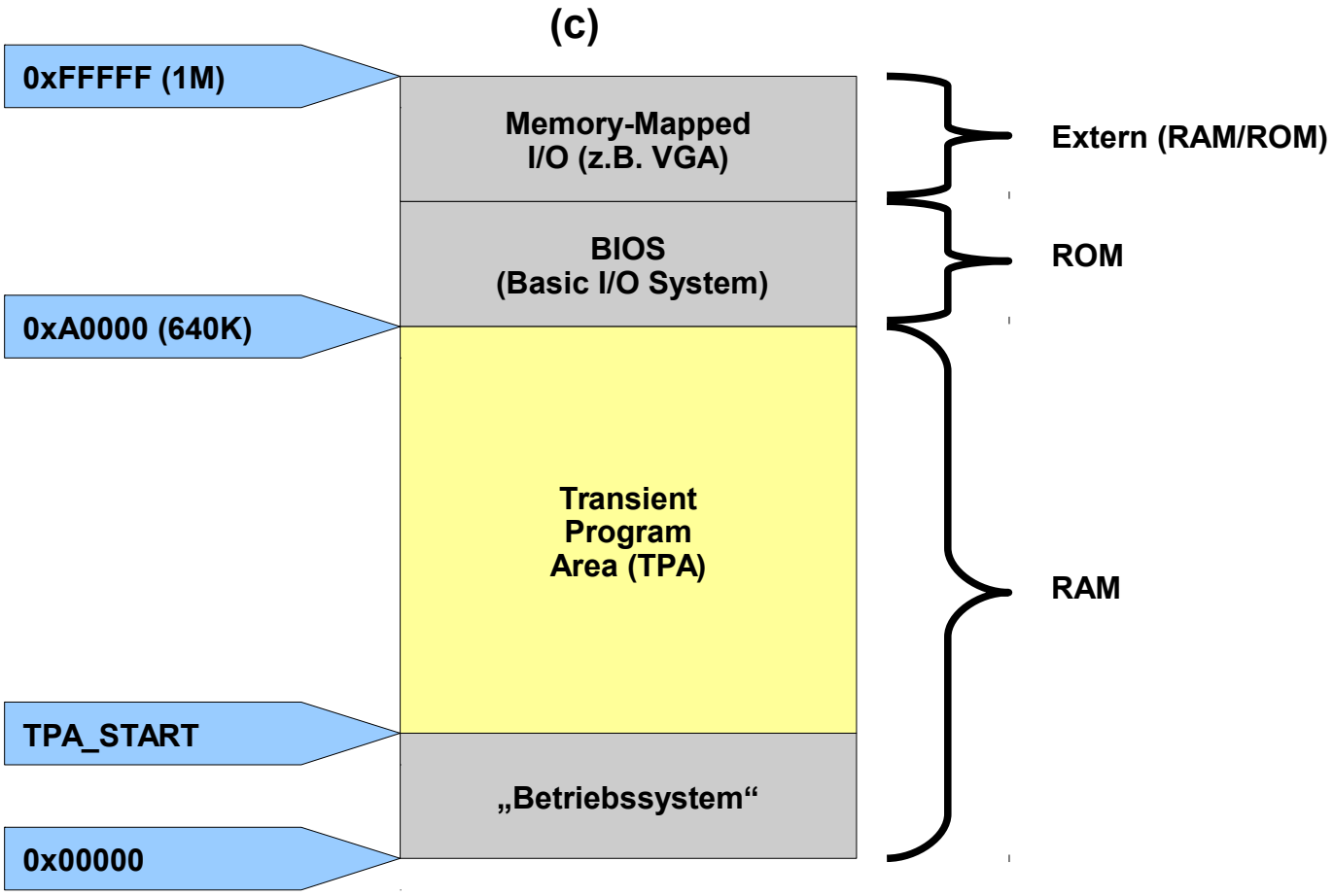
Hochschule RheinMain
University of Applied Sciences
Wiesbaden Rüsselsheim

- Nur ein Prozess befindet sich zu einem Zeitpunkt im Speicher. Zuweisung des gesamten Speichers an diesen Prozess.
- Einfache Mikrocomputer: Speicheraufteilung zwischen einem Prozess und dem Betriebssystem.
- Typische Alternativen:



Beispiel

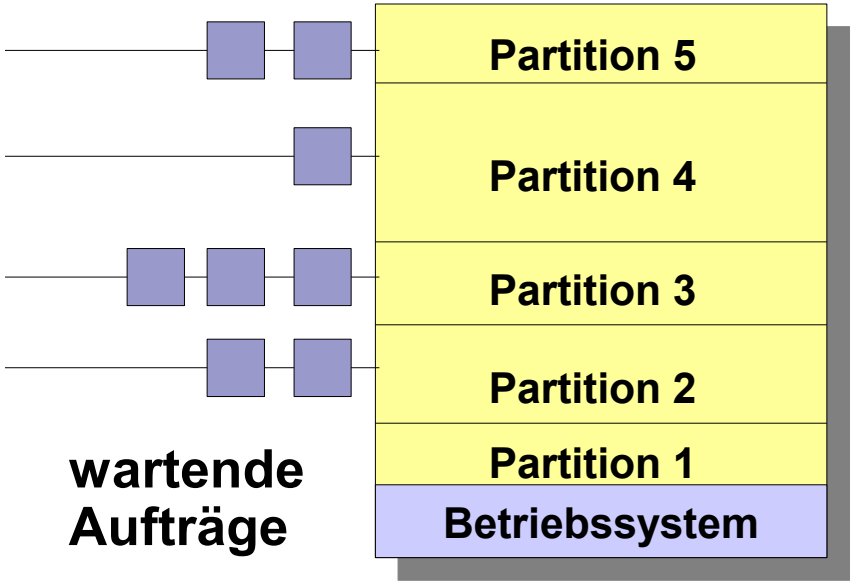
IBM PC (i8088):



9.1.2 Mehrprogrammbetrieb

Hochschule RheinMain
University of Applied Sciences
Wiesbaden Rüsselsheim

- Ziel: Bessere Ausnutzung der CPU (vgl. Kap. 1).
- Aufteilung des Speichers in feste Partitionen bei Systemstart. In jede Partition kann ein Programm geladen werden.
- Beispiel: IBM Mainframe /360 Betriebssystem OS/MFT (Multiprogramming with a Fixed number of Tasks).
- Programme können für eine Partition gebunden sein und sind dann nur in dieser lauffähig.

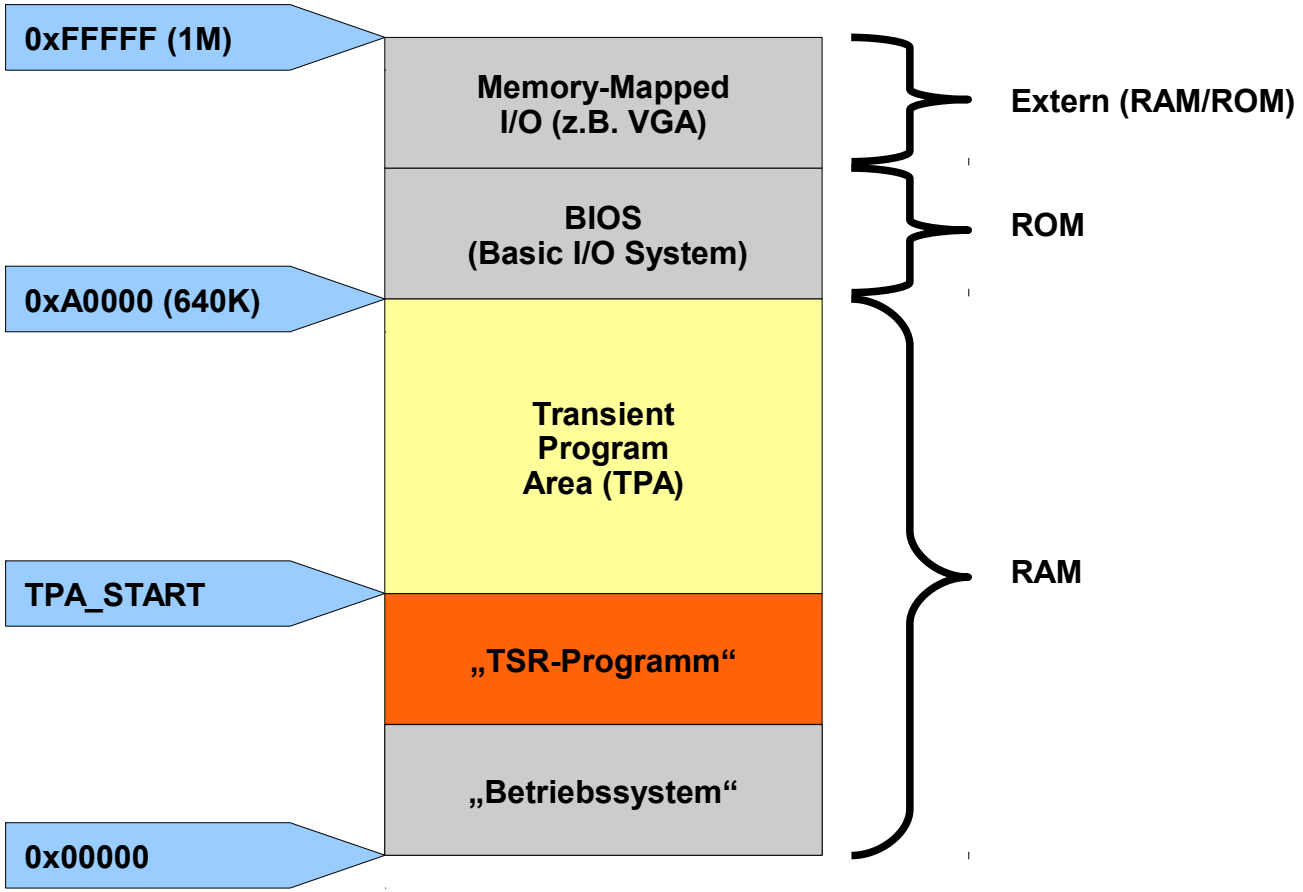


„Krudes“ Beispiel: MS-DOS

Hochschule RheinMain
University of Applied Sciences
Wiesbaden Rüsselsheim

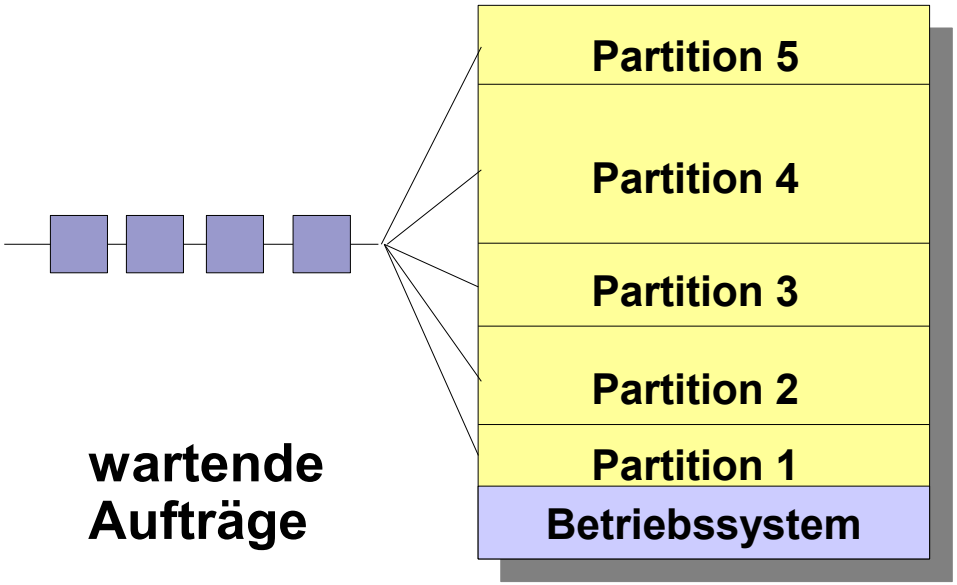
„Mehrprogrammbetrieb“ mit TSR(*)-Programm

(*) terminate, stay resident



Mehrprogrammbetrieb (2)

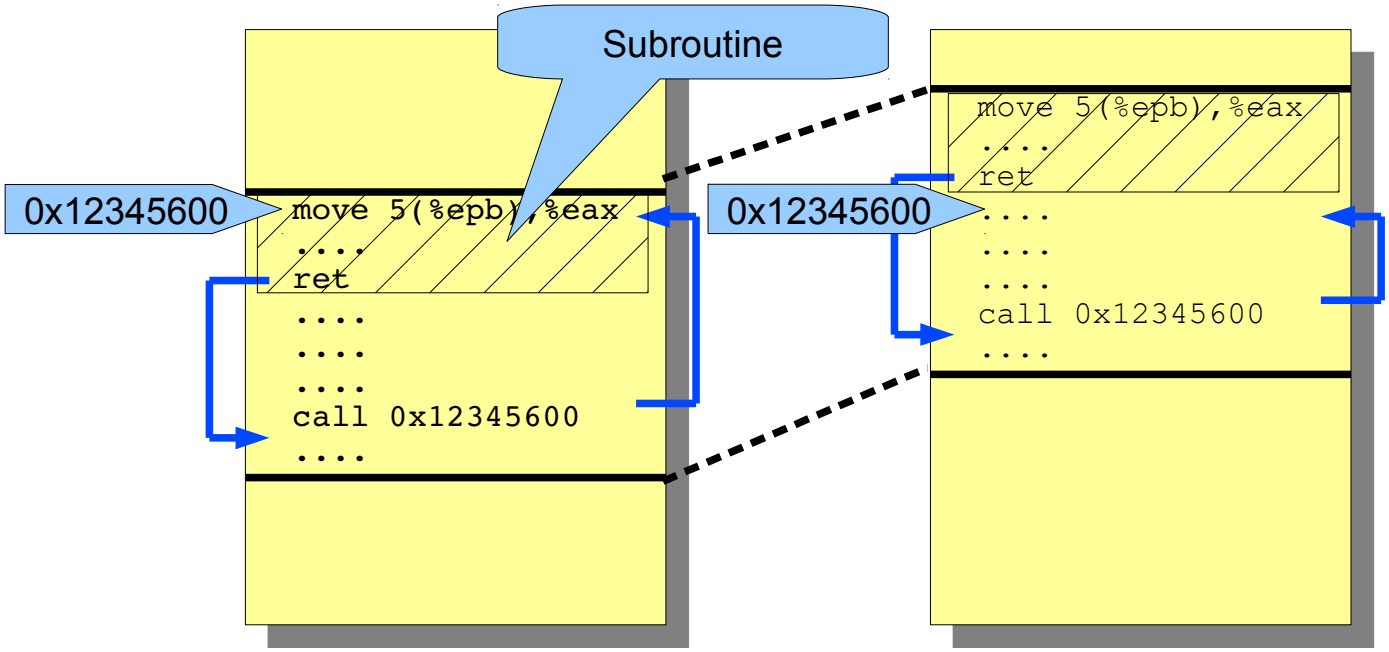
- Freie Auswahl einer Partition verlangt Lösung des sog. Relokationsproblems (Verschiebbarkeit), d.h. Anpassung aller (absoluten) Adressen des Programms in Abhängigkeit von der Ladeadresse.



Relokationsproblem

Hochschule RheinMain
University of Applied Sciences
Wiesbaden Rüsselsheim

- In der Regel sind Programme nicht Positionsunabhängig
  („PIC“=*position independent Code*, „PID“ = *position independent Data*)



- Mögliche Lösung des Relokationsproblems: Loader addiert die Ladeadresse auf alle Adressen gemäß eines vom Binder erstellten Adressverzeichnisses des Programms
  (Bsp.: OS/MFT).

Mehrprogrammbetrieb (3)

- Mehrprogrammbetrieb führt auch zum Schutzproblem:

  Programme können aufgrund der absoluten Adressierung Speicherbereiche anderer Benutzer lesen und schreiben (unerwünscht !).

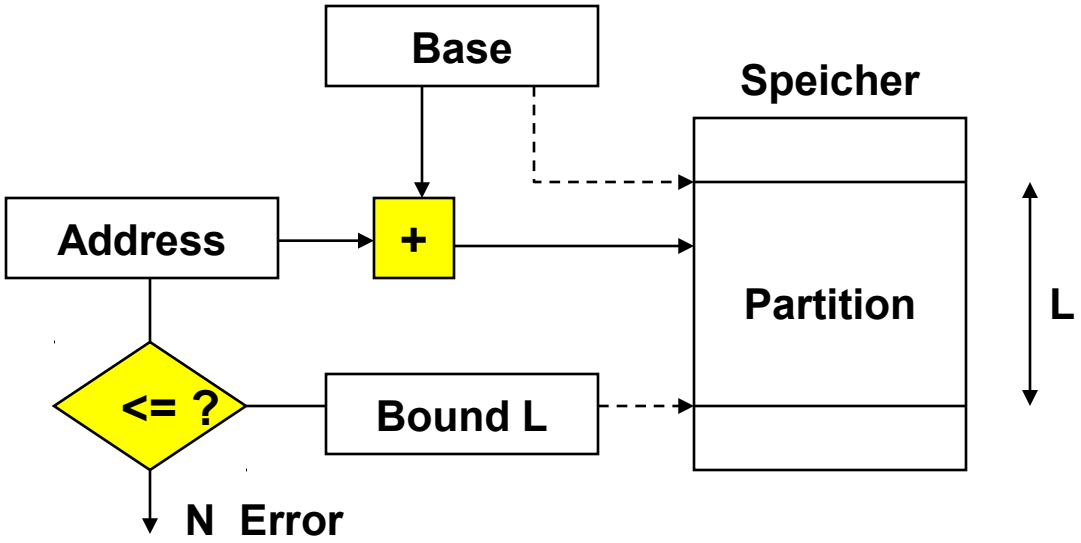
- Lösung des Schutzproblems in IBM /360:
  - Jedem 2kB-Block des Arbeitsspeichers wird ein 4-Bit Schutzcode (protection key) als Schloss zugeordnet.
  - Jeder Prozess besitzt einen ihm im Programmstatuswort (PSW) zugeordneten Schlüssel, der beim Zugriff auf einen Block passen muss, ansonsten Abbruch.
  - Nur das Betriebssystem kann Schutzcodes von Speicherblöcken und Schlüssel von Prozessen ändern.

⇒ Fazit: Ein Benutzerprozess kann weder andere Benutzerprozesse noch das Betriebssystem stören.

Mehrprogrammbetrieb (3)

- Alternative Vorgehensweise zur Lösung des Schutzproblems wie des Relokationsproblems:

  Ausstattung der CPU mit zwei zusätzlichen Registern, die Basisregister und Grenzregister genannt werden
  (base and bound register).



- Weiterer Vorteil eines Basisregisters zur Relokation: Verschieblichkeit des Programms nach Programmstart wird möglich.

Hochschule RheinMain
University of Applied Sciences
Wiesbaden Rüsselsheim

Einführung

- Bei klassischen Timesharing-Systemen gibt es normalerweise nicht genügend Hauptspeicher, um Prozesse aller Benutzer aufzunehmen.
- Swapping beinhaltet das Verschieben von Prozessen vom Hauptspeicher auf Platte (Auslagern) und umgekehrt (Einlagern).
- Im Hauptspeicher zu jedem Zeitpunkt nur eine Teilmenge der Prozesse, diese sind aber vollständig repräsentiert (im Gegensatz zu virtuellem Speicher, vgl. 9.3).
- Die restlichen (möglicherweise auch rechenwilligen) Prozesse werden in einem Bereich im Hintergrundspeicher abgelegt. Dieser Bereich heißt Swap-Bereich (Swap Area).
- Beispiel: Im Unix vor Einführung der virtuellen Speicherverwaltung genutzt.

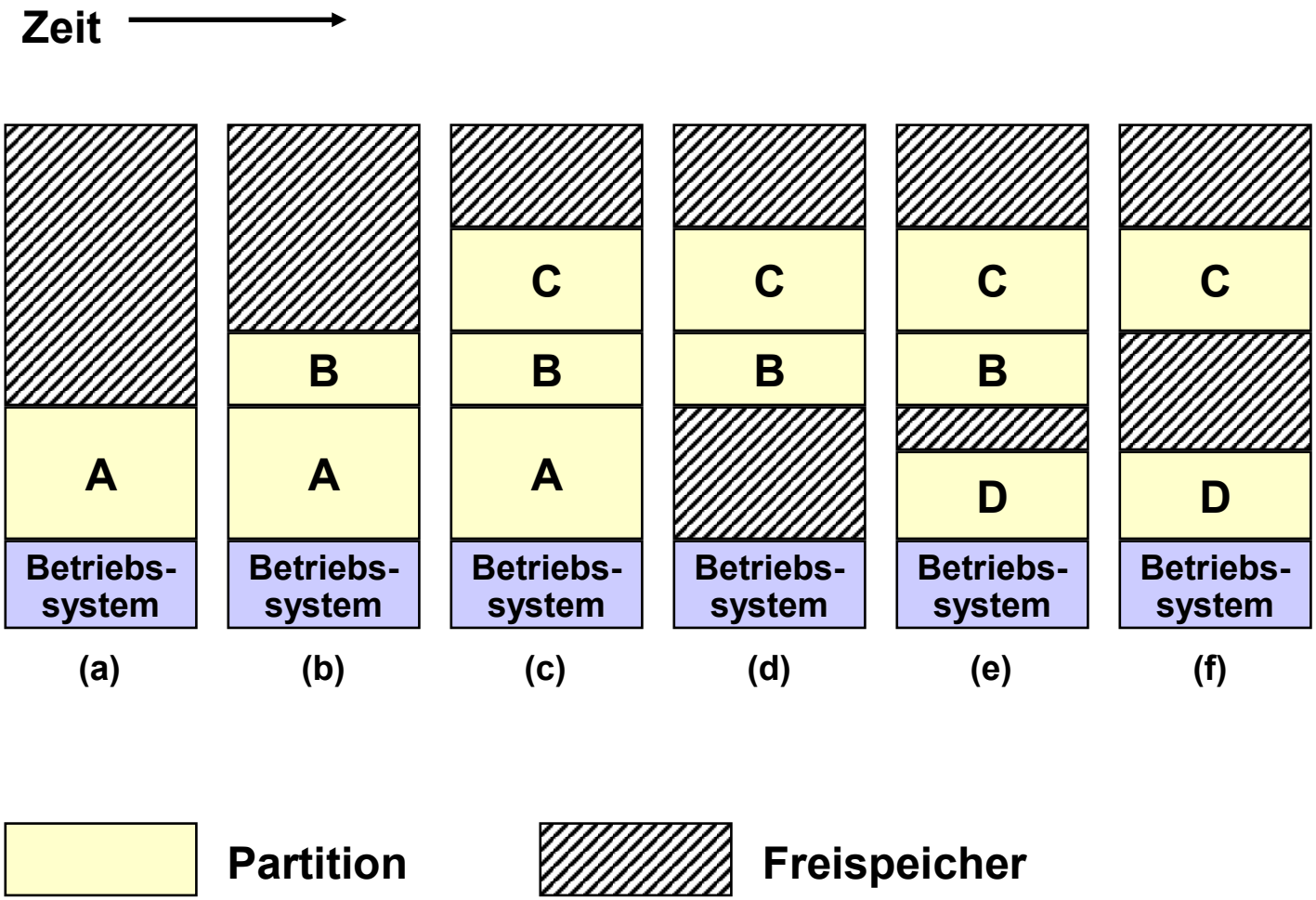
9.2.1 Variable Partitionen

Hochschule RheinMain
University of Applied Sciences
Wiesbaden Rüsselsheim

Def

- Mehrprogrammbetrieb kann in variablen Partitionen erfolgen, d.h.: Anzahl, Anfangsadresse und Länge der Partitionen und damit der eingelagerten Prozesse ändern sich dynamisch.
- Ein zusammenhängender Speicherbereich variabler Länge heißt auch Speichersegment oder einfach Segment.
- Ziele variabler Partitionen:
  - Anpassung an die tatsächlichen Speicheranforderungen
  - Vermeidung von Verschwendung.

Beispiel



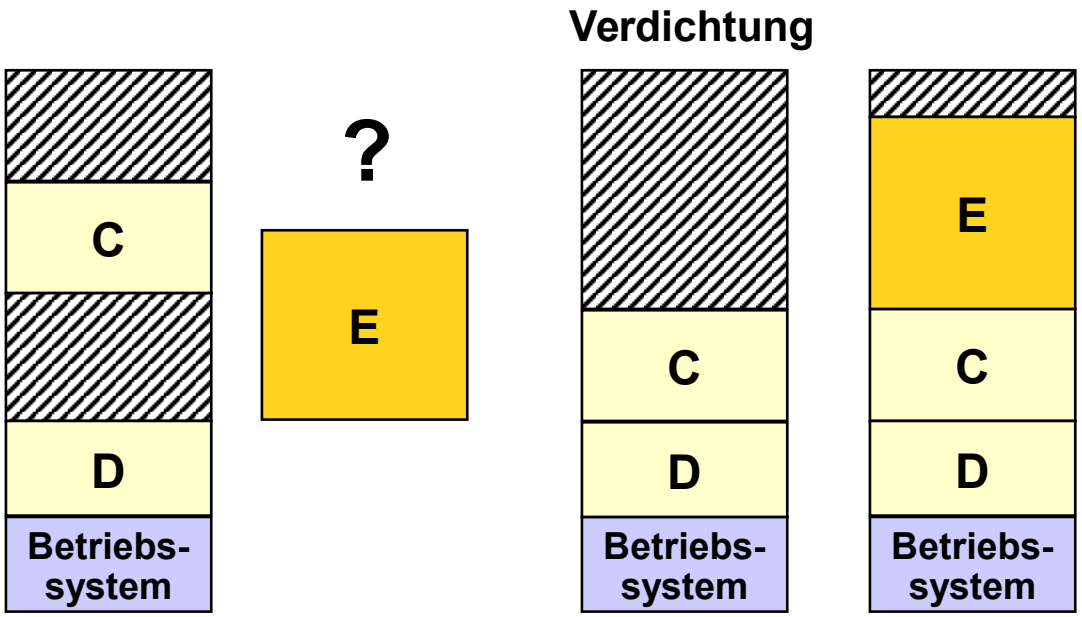
Speicherverdichtung

Hochschule RheinMain
University of Applied Sciences
Wiesbaden Rüsselsheim

- Zerstückelung von Freispeicher zwischen den belegten Segmenten wird externe Fragmentierung genannt.

Def

- Abhilfe: Alle unbelegten Speicherbereiche können durch Verschiebung der belegten Segmente zu einem einzigen Bereich zusammengefasst werden. (⇒ Speicherverdichtung oder Kompaktifizierung).
- Beispiel:



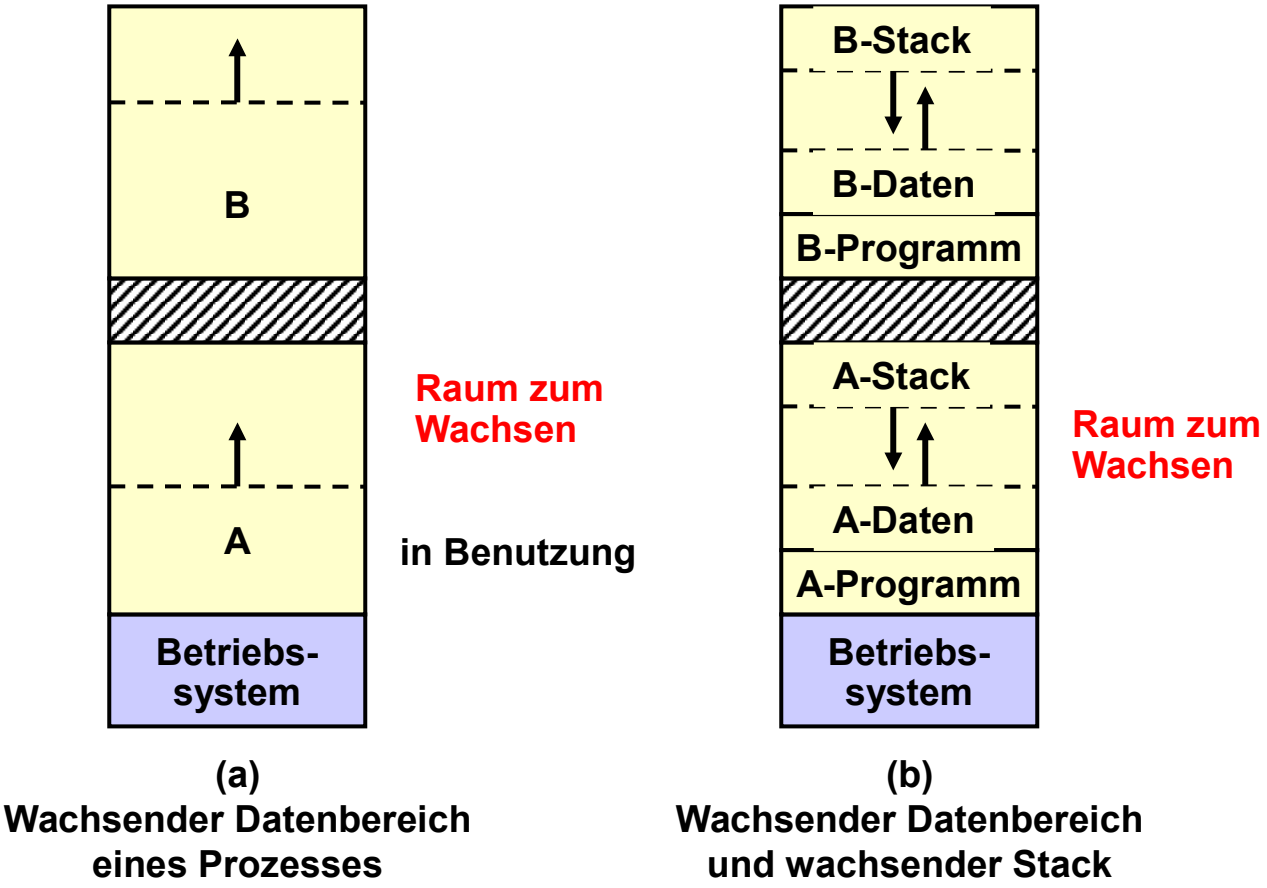
Dynamische Speicheranforderungen

Hochschule RheinMain
University of Applied Sciences
Wiesbaden Rüsselsheim

- Speicherbedarf eines Prozesses kann sich im Verlaufe ändern (i.d.R. wachsen).
- Prozess stellt dynamische Speicheranforderungen, z.B. zur Aufnahme von neu erzeugten Objekten auf der Halde (dem Heap).
- Behandlung dynamischer Speicheranforderungen:
  - Belegen angrenzender „Löcher", falls vorhanden.
  - Verschiebung von anderen Prozessen, falls möglich.
  - Auslagern von anderen Prozessen.
  - „Raum zum Wachsen": Bei der ersten Speicherbelegung vorab zusätzlichen Speicherplatz alloziieren.

Dyn. Speicheranforderungen (2)

- Beispiele für "Raum zum Wachsen":



(a)
Wachsender Datenbereich
eines Prozesses

(b)
Wachsender Datenbereich
und wachsender Stack

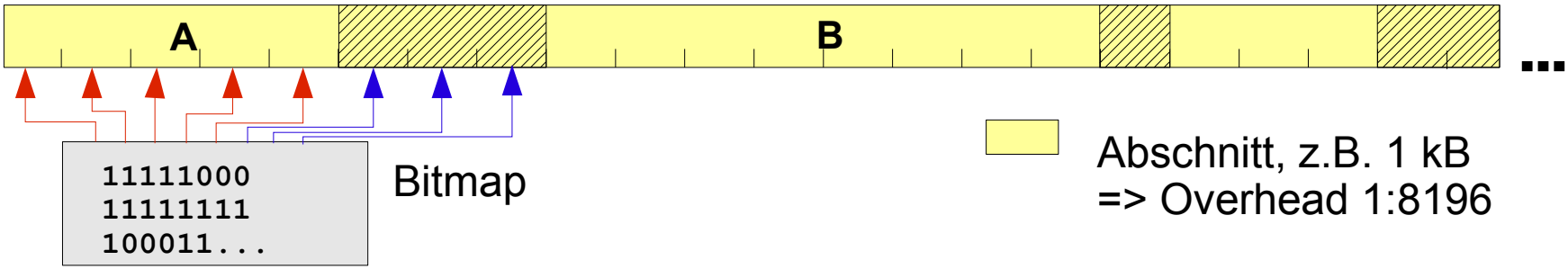
Algorithmen der Speicherverwaltung

Hochschule RheinMain
University of Applied Sciences
Wiesbaden Rüsselsheim

- Speicherverwaltung mit Bitmaps (9.2.2).
- Speicherverwaltung mit verketteten Listen (9.2.3).
- Speicherverwaltung mit dem Buddy-System (9.2.4).

9.2.2 Speicherverw. mit Bitmaps

Hochschule RheinMain
University of Applied Sciences
Wiesbaden Rüsselsheim

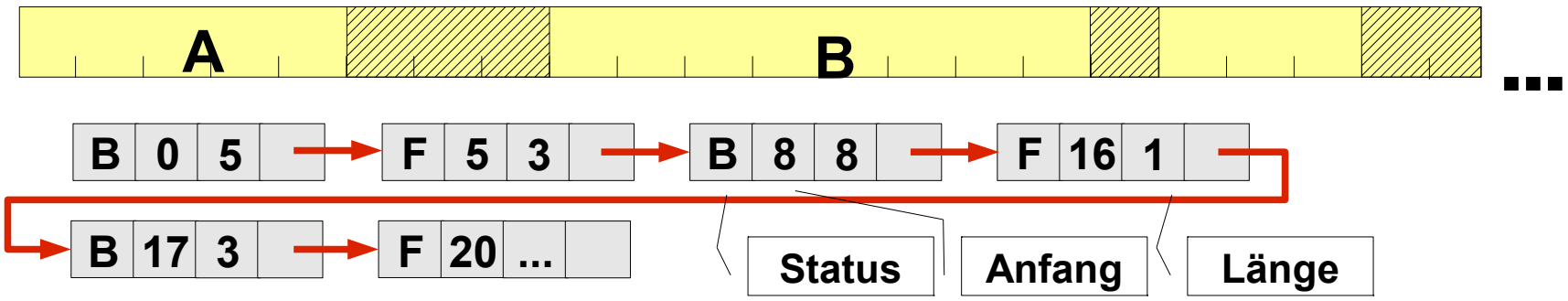


Arbeitsweise:

- Der zur Verfügung stehende Speicher wird in Einheiten fester Länge unterteilt (Granularität einige Worte bis einige KB).
- Jeder Speichereinheit wird ein Bit in einer Bitmap zugeordnet:
  0 ≅ Einheit ist frei, 1 ≅ Einheit ist belegt.
- Je kleiner die Einheit, je größer die Bitmap.
- Je größer die Einheit, je mehr Speicher wird verschwendet, da die letzte „angebrochene“ Einheit voll alloziiert werden muss.
- Hauptproblem: Belegen eines Speicherbereichs von k Einheiten erfordert Durchsuchen der Bitmap nach einer Folge von k Null-Bits. Aufwändig!
- Für Hauptspeicherverwaltung daher selten eingesetzt.

9.2.3 Verkettete Listen

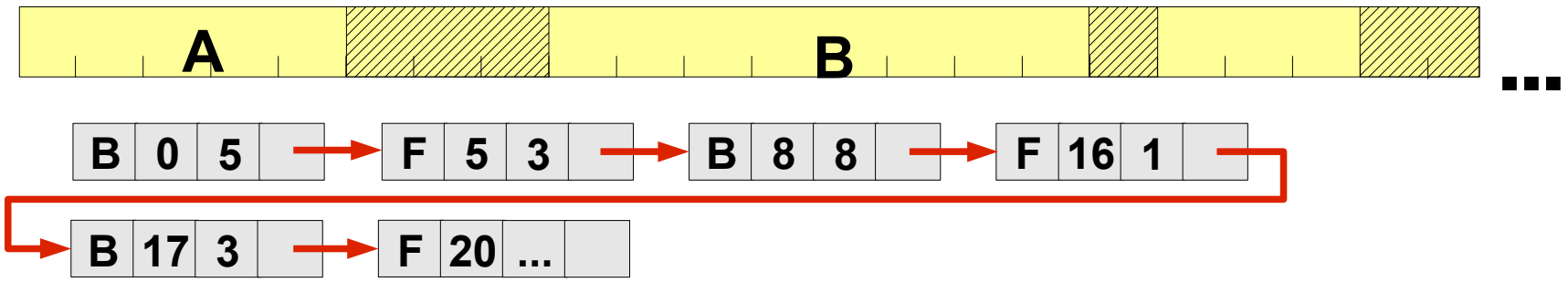
Hochschule RheinMain
University of Applied Sciences
Wiesbaden Rüsselsheim



- Jedem belegten und jedem freien Speicherbereich wird ein Listenelement zugeordnet.
- Segmente dürfen variabel lang sein.
- Jedes Listenelement enthält Startadresse und Länge des Segments, sowie den Status (B=belegt, F=frei).
- Gefundenes freies Segment wird (falls zu groß) aufgespalten.
- Freigegebenes Segment wird ggf. mit ebenfalls freien Nachbarsegmenten „verschmolzen“.

Verkettete Listen (2)

Hochschule RheinMain
University of Applied Sciences
Wiesbaden Rüsselsheim



- Die freien Segmente können auch in separater Liste geführt werden („Freiliste“). Die Freiliste kann in sich selbst gehalten werden, d.h. in den verwalteten freien Bereichen.
  (⇒ kein zusätzlicher Speicher notwendig).
- Die Segmentliste kann nach Anfangsadressen geordnet sein. Vorteil: freiwerdendes Segment kann mit benachbartem freien Bereich zu *einem* freien Segment „verschmolzen“ werden.
- Freiliste kann alternativ nach der Größe des freien Bereichs geordnet sein. Vorteil: Vereinfachung beim Suchen nach einem freien Bereich bestimmter Länge.

Algorithmen zur Speicherallozііerung

- Voraussetzungen:
  - Liste der Segmente nach Anfangsadressen geordnet.
  - Geforderte Speichergröße bekannt.

Def

- First Fit: Durchsuche Freiliste, bis ein freies Segment hinreichender Größe gefunden ist. Zerlege den freien Bereich in ein Segment der geforderten Größe und ein neues freies Segment für den übrig bleibenden Rest. Kurze Suchzeit.
- Rotating First Fit oder Next Fit: Variante von First Fit. Anstelle von vorn zu suchen, beginne Suche an der nachfolgenden Stelle, an der bei letzten Durchlauf ein Segment belegt wurde.
- Best Fit: Durchsuche die gesamte Liste, wähle das kleinste für die Anforderung gerade ausreichende freie Segment und spalte dieses. Langsam. Neigt zur ungewollten Erzeugung vieler kleiner Freisegmente (Fragmentierung).
- Worst Fit: Wähle das größte freie Segment und spalte dieses. Vermeidet vieler kleiner Freisegmente, aber keine gute Idee.

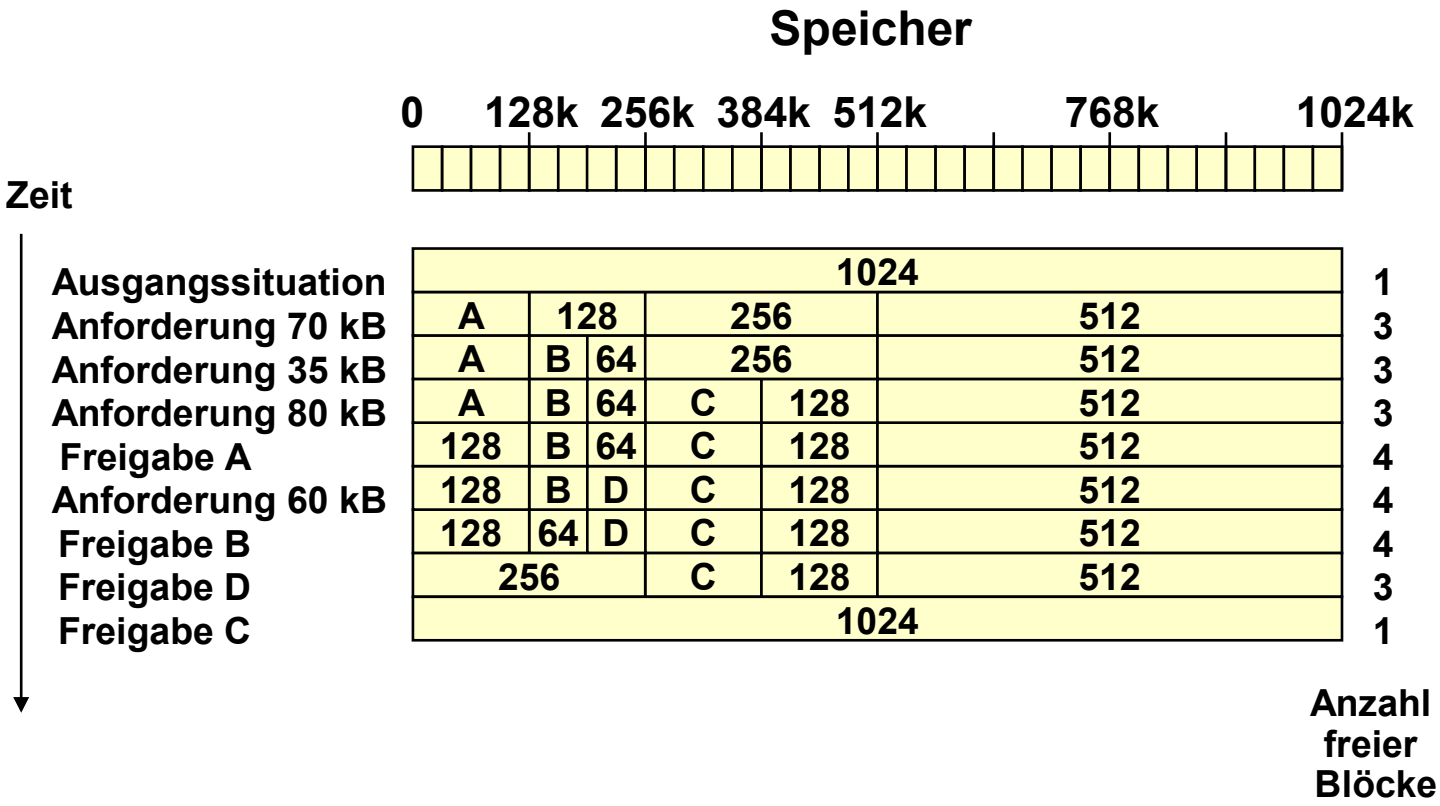
9.2.4 Buddy-System

Hochschule RheinMain
University of Applied Sciences
Wiesbaden Rüsselsheim

Arbeitsweise:

- Freie und belegte Speicherbereiche haben ausschließlich Längen, die 2-er Potenzen sind.
- Da nur bestimmte Längen auftreten, wird von Speicherblöcken (anstelle von Segmenten) gesprochen.
- Die maximale Blocklänge L_{max} entspricht der größten 2-er-Potenz kleiner gleich dem verfügbaren Speicher, z.B. $L_{max} = 2^{22}$ bei 4 MB oder 6 MB.
- Es kann eine minimale Blocklänge größer $2^0 = 1$ Byte geben, z.B. $L_{min} = 2^6 = 64$ Bytes.
- Eine Speicheranforderung wird auf die nächst mögliche 2-er-Potenz aufgerundet, z.B.: angefordert 70 KB, belegt 128 KB.
- Für jede der verwalteten Blocklängen zwischen L_{min} und L_{max} wird eine separate Freiliste gehalten.

Arbeitsweise am Beispiel

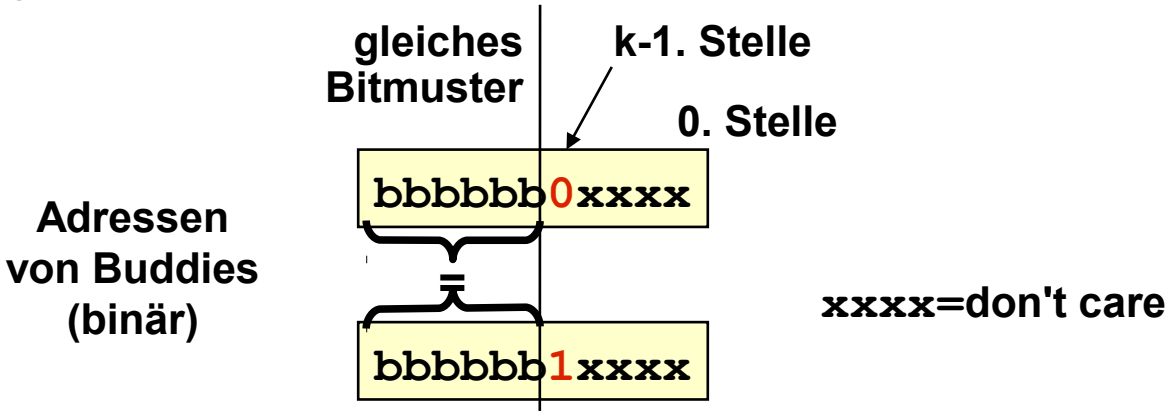


Freispeicherverwaltung

Hochschule RheinMain
University of Applied Sciences
Wiesbaden Rüsselsheim

- Kein freier Block der gesuchten Länge: Halbiere einen Block der nächsthöheren 2-er-Potenz in 2 freie Blöcke der gesuchten Länge. Diese werden Buddies genannt (deutsch: Kumpel).
- Freigabe eines Blocks der Länge 2^k : wenn sein Buddy bereits ein freier Block ist, mit diesem zu *einem* neuen freien Block der Länge 2^{k+1} vereinen.
- Es lassen sich **nicht** zwei beliebige benachbarte Blöcke gleicher Länge 2^k zusammenfassen!

  Die Entscheidung ist einfach aufgrund der Adresse der freien Blöcke möglich:



Fazit Buddy-System

Vorteile:

- Bei Speicheranforderung insgesamt schnelles Auffinden oder Erzeugen eines freien Blocks.
- Bei Freigabe muss nur *eine* Freiliste durchsucht werden. Falls Buddy bereits frei ist, einfaches Verschmelzen zu einem größeren Block.

Nachteile:

- Ineffizient in der Speicherausnutzung, da immer auf die nächste 2-er-Potenz aufgerundet werden muss.
- Diese Verschwendung wird als interne Fragmentierung bezeichnet, da sie innerhalb des alloziierten Speicherbereichs auftritt (im Gegensatz zur externen Fragmentierung).

Hochschule RheinMain
University of Applied Sciences
Wiesbaden Rüsselsheim

Ergebnisse:

- Speicherverwaltung mit Bitmaps oder verketteten Listen führt zu externer Fragmentierung.

Def

- 50%-Regel (Knuth): Sind im Mittel n Prozesse im Speicher, so gibt es im Mittel n/2 freie Bereiche. Ursache ist, dass zwei benachbarte Freibereiche zu einem zusammengefasst werden.

Def

- Ungenutzte-Speicher-Regel: Sei s die mittlere Größe eines Prozesses, und k*s sei die mittlere Größe eines Freibereichs für einen Faktor k, der abhängig vom Algorithmus ist.
  Dann gilt für den ungenutzten Anteil f des Speichers:

$$f = k / (k+2).$$

Beispiel: Sei k = 0.5. Dann werden 20% des Speicherplatzes für Freibereiche verbraucht.

9.2.6 Zusammenfassung

Swapping:

- Im Hauptspeicher wird zu jedem Zeitpunkt nur eine Teilmenge der Prozesse gehalten. Die restlichen Prozesse sind in einen Swap-Bereich auf Platte ausgelagert.
- Alle Prozesse im Hauptspeicher sind *vollständig* repräsentiert.
- Es wurden Verfahren zur Speicherverwaltung für derartige Systeme besprochen:
  - Verwaltung mittels Bitmaps
  - Verwaltung mittels verketteter Listen
  - Buddy-System
- Es wurden Strategien zur Speicherverwaltung besprochen:
  - First Fit
  - Rotating First Fit
  - Best Fit

9.3 Virtueller Speicher

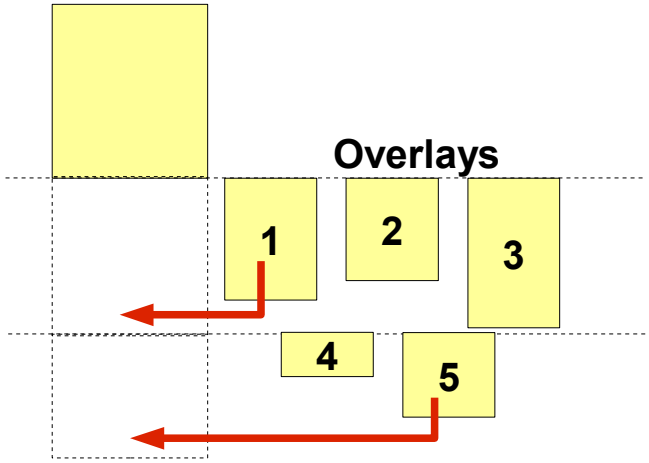
Hochschule RheinMain
University of Applied Sciences
Wiesbaden Rüsselsheim

Problem:

- Die Größe eines einzelnen Prozesses kann die Größe des insgesamt zur Verfügung stehenden Speichers übersteigen.
  ⇒ Swapping ist keine Lösung mehr.

Historische Lösung:

- Lösung der 60er Jahre arbeitet mit Overlays (Überlagerungen):
  - Overlay-Struktur des Programms wird vom Programmierer vorgeplant und vom Binder erzeugt.
  - Das Betriebssystem muss bei Bedarf Overlays dynamisch vom Hintergrundspeicher nachladen:



Heute z.T. noch/wieder aktuell
bei Echtzeitsystemen
(wg. planbarer Nachlade-Zeitpunkte)

Virtueller Speicher (2)

Hochschule RheinMain
University of Applied Sciences
Wiesbaden Rüsselsheim

Heutige Lösung:

Def

- Betriebssystem hält „gerade in Benutzung“ befindlichen Teile eines Programms im Hauptspeicher, Rest auf Hintergrundspeicher.
- Dieser Ansatz wird virtueller Speicher genannt.
- Die von den Instruktionen eines Programms generierten Adressen werden virtuelle Adressen genannt. Sie bilden zusammen den (virtuellen) Adressraum des Prozesses, der das Programm ausführt.
- Speicherverwaltungseinheit (Memory Management Unit, MMU) wandelt virtuelle Adressen in reale (physikalische) Adressen um.
- Virtueller Speicher und Mehrprogrammbetrieb passen gut zueinander: Werden Teile eines blockierten Prozesses eingelagert, kann der Scheduler den Prozessor einem anderen (rechenwilligen) Prozess zuordnen.
- Paging: eindimensionaler virtueller Speicher. Heute am stärksten verbreitet.
- Segmentierung: zweidimensionaler virtueller Speicher.

Zusammenfassung

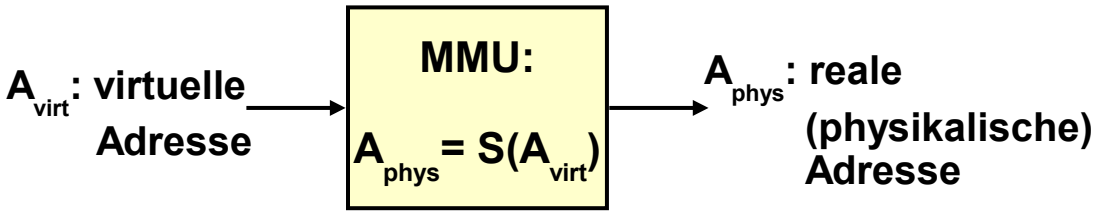
Hochschule RheinMain
University of Applied Sciences
Wiesbaden Rüsselsheim

- Grundbegriffe des Paging
  - Seiten (Pages)
  - Kachel oder Seitenrahmen (Page Frames)
  - Seitentabelle (Page Tables)
  - Seitenfehler (Page Fault)
- Prinzipielle Arbeitsweise einer MMU
- Realisierungsgesichtspunkte
  - Mehrstufige Seitentabellen
  - Hardware-Unterstützung mittels Assoziativspeicher
- Arbeitsweise einer MMU und verschiedene Formen der Übersetzung virtueller Adressen in reale unter Nutzung von mehrstufigen Seitentabellen und TLBs zum Caching von Übersetzungen wurden im Rechnerarchitekturteil der Vorlesung besprochen.

Grundbegriffe des Paging (1)

Hochschule RheinMain
University of Applied Sciences
Wiesbaden Rüsselsheim

- Die von den Instruktionen eines Programms generierten Adressen werden virtuelle Adressen genannt. Sie bilden zusammen den (virtuellen) Adressraum des Prozesses, der das Programm ausführt.
- Speicherverwaltungseinheit (Memory Management Unit, MMU):
  - wandelt virtuelle Adressen in reale (physikalische) Adressen um
  - reale Adressen werden dann für den Zugriff zum Arbeitsspeicher benutzt
  - Abbildung wird abstrakt auch als Speicherfunktion bezeichnet

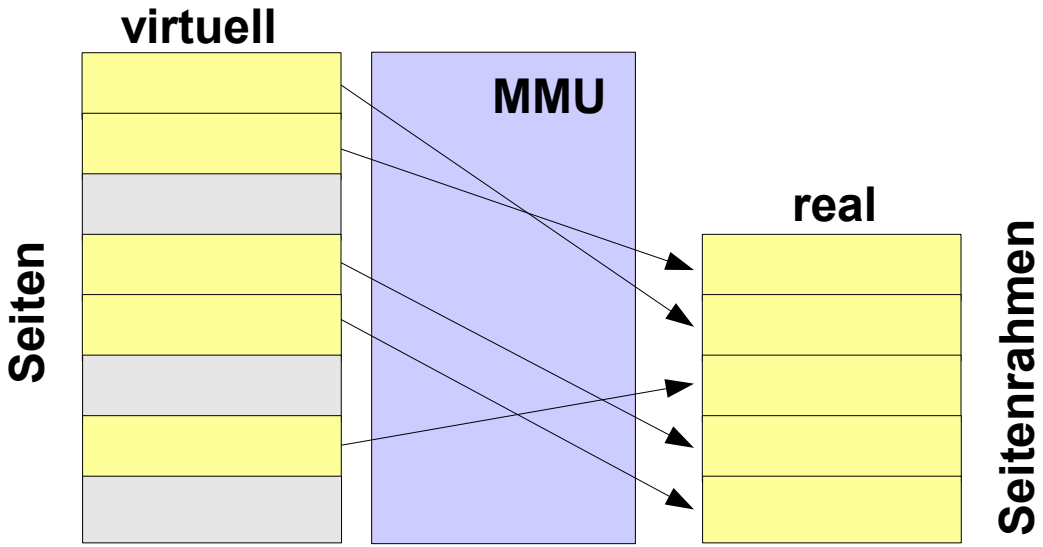


Grundbegriffe des Paging (2)

Hochschule RheinMain
University of Applied Sciences
Wiesbaden Rüsselsheim

Def

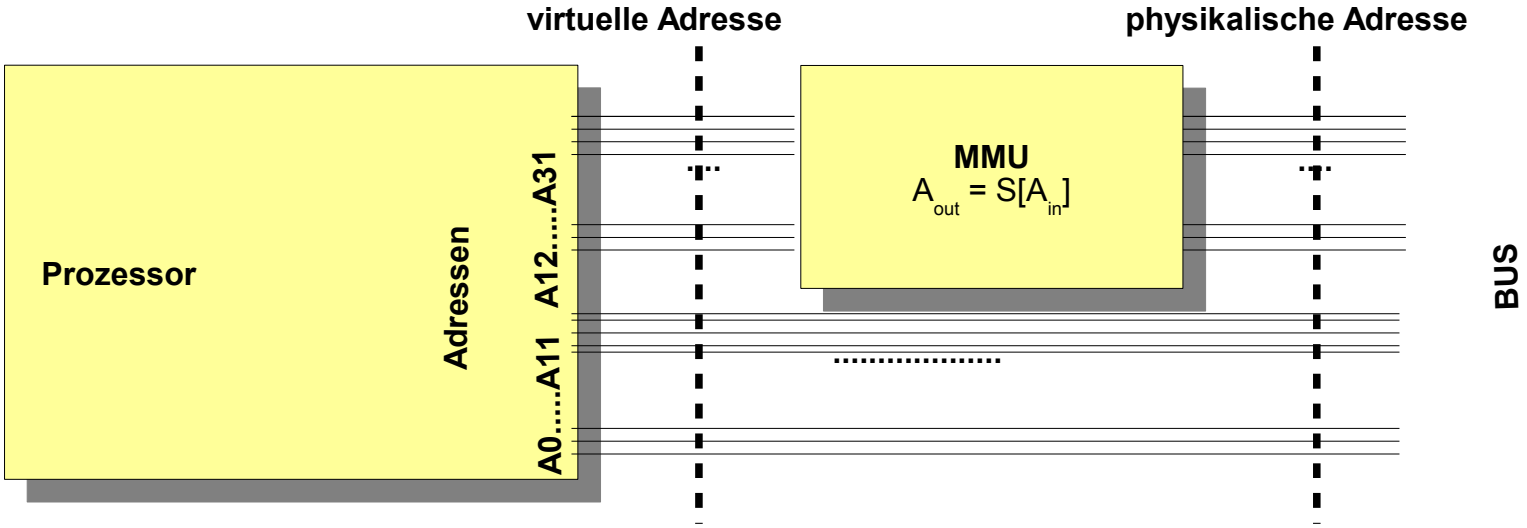
- Virtueller Adressraum ist in Einheiten fester Länge unterteilt, diese heißen Seiten (Pages).
- Physikalischer Speicher wird ebenfalls in Einheiten dieser Länge unterteilt, diese heißen Kacheln oder Seitenrahmen (Page Frames).
- Transfer zwischen Hauptspeicher und Hintergrundspeicher erfolgt grundsätzlich in Einheiten von Seiten.



Grundbegriffe des Paging (3)

Hochschule RheinMain
University of Applied Sciences
Wiesbaden Rüsselsheim

- Heute typische Größen:
  - virtueller Adressraum:
    2^{32} Bytes (4 GB) basierend auf 32-Bit Adressen,
    2^{48} Bytes (256 TB) basierend auf 64-Bit Adressen
    (AMD64, Intel64).
  - physikalischer Arbeitsspeicher:
    i.d.R. kleiner, 2-6 GB für Arbeitsplatzrechner,
    Tendenz steigend.
  - Seiten / Seitenrahmen: typisch 4 KB oder 8 KB.



Grundbegriffe des Paging (4)

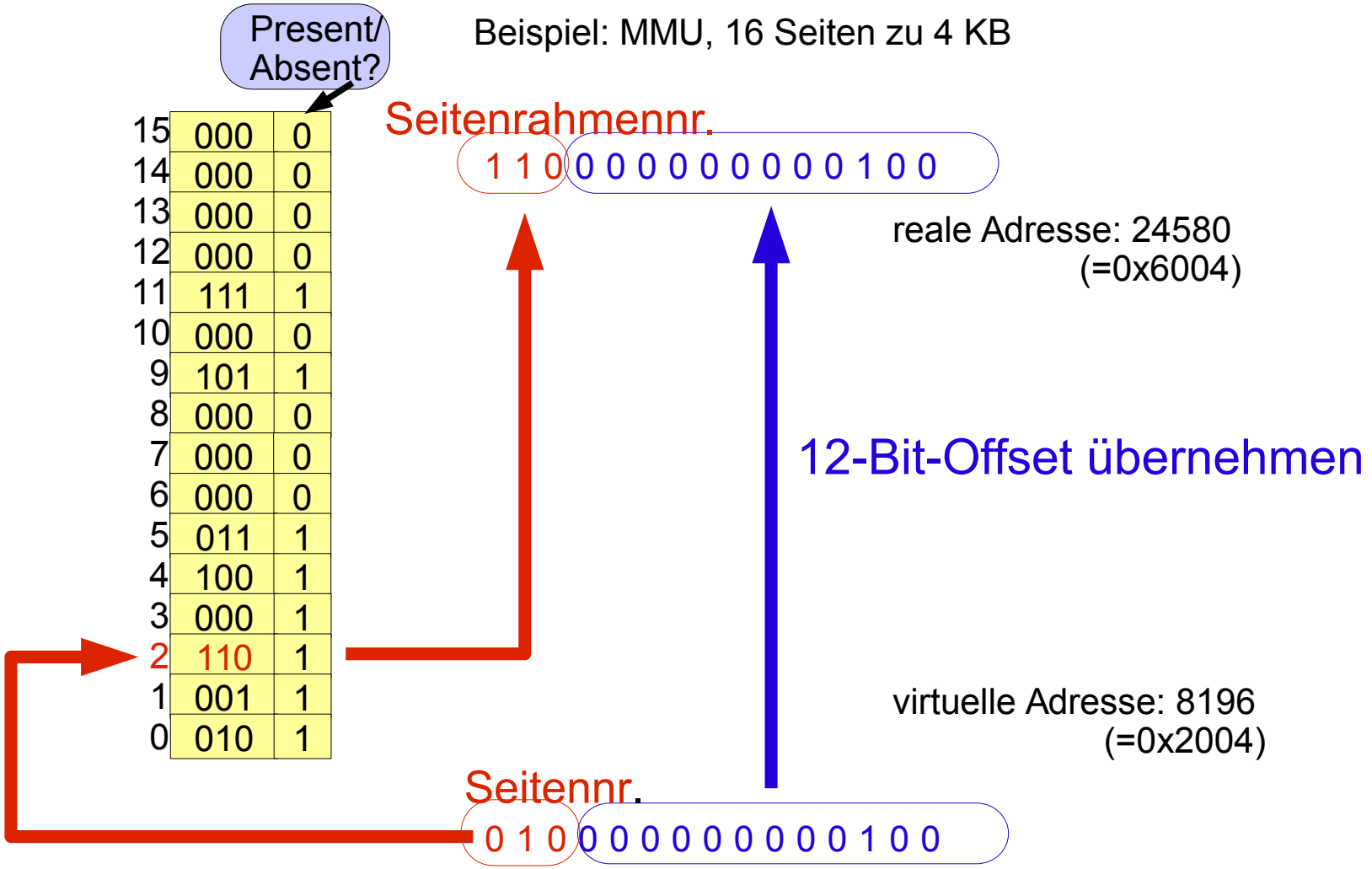
Hochschule RheinMain
University of Applied Sciences
Wiesbaden Rüsselsheim

Def

- Die abstrakte Speicherfunktion kann bildlich durch eine Tabelle repräsentiert werden, bezeichnet als Seitentabelle.
- Die Seitentabelle ordnet jeder Seite des virtuellen Adressraums einen Seitenrahmen des Hauptspeichers oder ein Kennzeichen zu, dass die Seite sich nicht im Hauptspeicher befindet (nicht eingelagert ist). Dieses Kennzeichen wird Present/Absent-Bit genannt.
- Benennt ein Programm eine virtuelle Adresse, deren zugehörige Seite nicht eingelagert ist, so erzeugt die MMU eine Unterbrechung der CPU. Diese Situation wird als Seitenfehler (Page Fault) bezeichnet.

MMU-Funktionsprinzip

Hochschule RheinMain
University of Applied Sciences
Wiesbaden Rüsselsheim



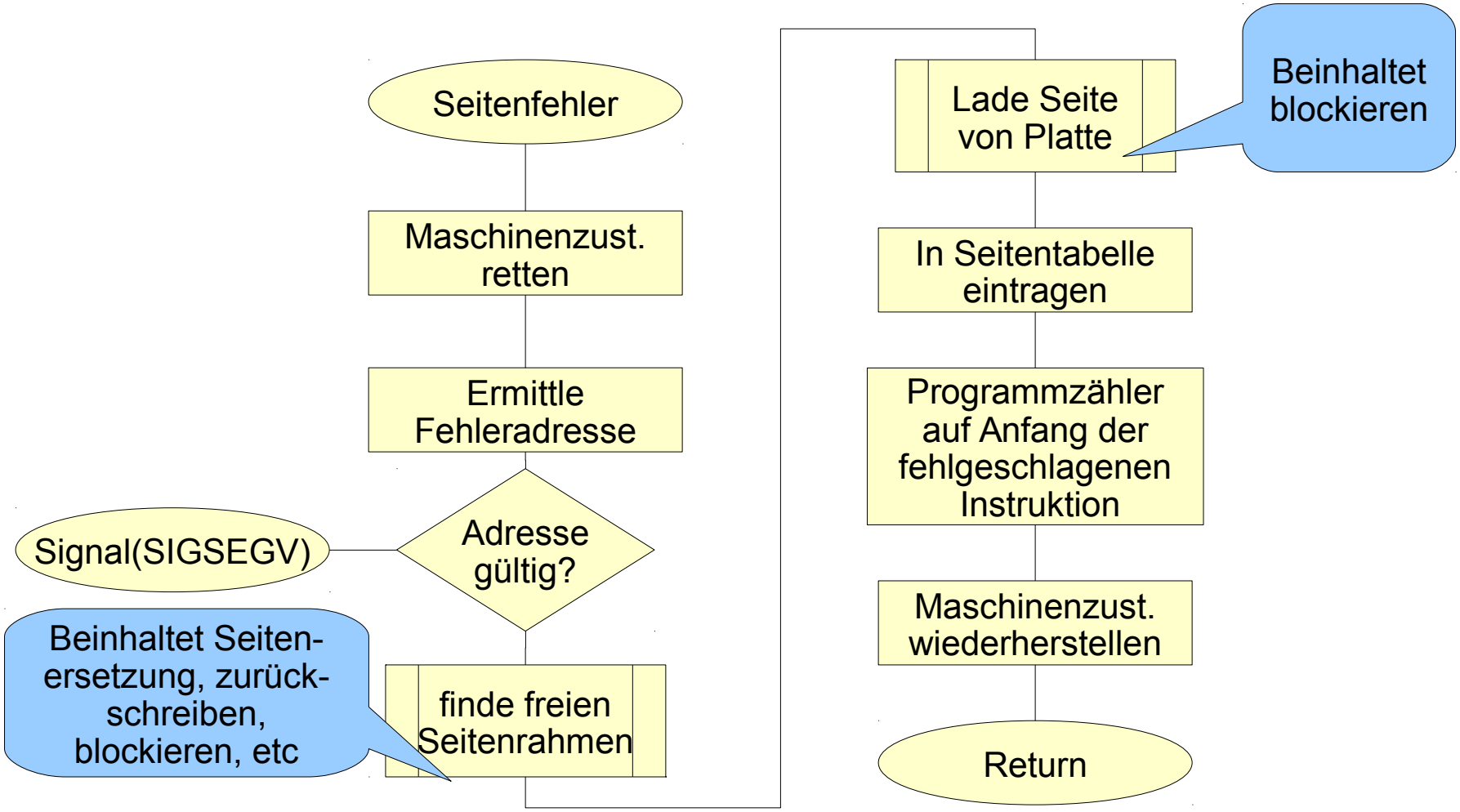
Behandlung eines Seitenfehlers

Hochschule RheinMain
University of Applied Sciences
Wiesbaden Rüsselsheim

- Behandlung eines Seitenfehlers besteht prinzipiell in der Einlagerung der fehlenden Seite vom Hintergrundspeicher in einen freien Seitenrahmen des Arbeitsspeichers.
- Einlagerung als Reaktion auf einen Seitenfehler wird als Demand Paging bezeichnet.
- Liegt kein freier Seitenrahmen vor, muss ein solcher gewonnen werden. I.d.R. hält das Betriebssystem einen Vorrat an freien Rahmen vor. (Algorithmen für die Auswahl zu ersetzender Seiten werden detailliert in 9.4 besprochen).
- Falls die zu ersetzende Seite während ihrer Lebenszeit im Speicher verändert wurde (als Dirty Page bezeichnet), wird sie auf die Platte zurückgeschrieben (Update). Der belegte Seitenrahmen ist danach frei.
- Falls die zu ersetzende Seite nicht verändert wurde (enthält z.B. Programmcode), ist kein Zurückschreiben erforderlich.

Behandlung eines Seitenfehlers (2)

Hochschule RheinMain
University of Applied Sciences
Wiesbaden Rüsselsheim



9.4 Seitenersetzungsalgorithmen

Hochschule RheinMain
University of Applied Sciences
Wiesbaden Rüsselsheim

Problem:

- Geschickte Auswahl von zu ersetzenden Seiten des Hauptspeichers:

  Wird eine häufig benutzte Seite ausgelagert, ist die Wahrscheinlichkeit groß, dass diese schon bald wieder eingelagert werden muss, was Zusatzaufwand für das Betriebssystem und Blockierungszeiten für den betroffenen Prozess mit sich bringt.

Ziel:

- Gute Performance des Systems.

Seitenersetzungsalgorithmen (Page Replacement Algorithms) sind ein klassisches Problem der Betriebssysteme, vielfach theoretisch und experimentell untersucht.

Überblick

In der BS-Literatur betrachtete Seitenersetzungsverfahren:

- Der optimale Seitenersetzungsalgorithmus.
- Not-Recently-Used (NRU).
- First-In, First-Out (FIFO) und Second-Chance.
- Clock-Algorithmus.
- Least-Recently-Used (LRU).
- Aging-Algorithmus.

9.4.1 „Optimales“ Verfahren

Hochschule RheinMain
University of Applied Sciences
Wiesbaden Rüsselsheim

Algorithmus:

- Wird Seitenrahmen benötigt, bestimme für alle eingelagerten Seiten, wieviele Instruktionen lang die Seite zukünftig durch die noch auszuführenden Instruktionen nicht referiert werden wird.
- Wähle die Seite zur Ersetzung aus, die am längsten nicht referiert werden wird.

Bemerkungen:

- Algorithmus schiebt den Seitenfehler, der dazu führt, dass die auszulagernde Seite wieder eingelagert werden muss, am weitesten in die Zukunft.
- Die benötigte Information über das zukünftiges Programmverhalten ist aber i.d.R. nicht vorhanden!
- Algorithmus ist daher nicht realisierbar, nur für Vergleichszwecke interessant.

9.4.2 Not-Recently-Used (NRU)

Hochschule RheinMain
University of Applied Sciences
Wiesbaden Rüsselsheim

Voraussetzungen:

- Verwendung der im Seitendeskriptor gespeicherten Status-Bits je Seite (vgl. VL Rechnerarchitektur):
  - R-Bit (Referenced: gesetzt, wenn auf die Seite lesend oder schreibend zugegriffen wird).
  - M-Bit (Modified: gesetzt, wenn auf die Seite schreibend zugegriffen wird).
- Status-Bits werden durch die Hardware aktualisiert.
- Gesetzte Status-Bits können durch das Betriebssystem (softwaremäßig) zurückgesetzt werden.

Not-Recently-Used (2)

Algorithmus:

- Bei Prozessstart werden beide Status-Bits für alle Seiten des Prozesses zurückgesetzt.
- Periodisch, z.B. bei jeder Uhr-Unterbrechung, wird das Referenced-Bit aller Seiten durch das Betriebssystem zurückgesetzt. Dadurch können im folgenden referierte Seiten von nicht referierten unterschieden werden.
- Uhr-Unterbrechungen löschen das Modified-Bit *nicht*. Es wird weiter für die Entscheidung über das Zurückschreiben benötigt.
- Tritt ein Seitenfehler auf, teile alle eingelagerten Seiten in die folgenden Klassen entsprechend der aktuellen Werte des R- und des M-Bits ein:
  - Klasse 0: nicht referiert, nicht modifiziert.
  - Klasse 1: nicht referiert, modifiziert.
  - Klasse 2: referiert, nicht modifiziert.
  - Klasse 3: referiert, modifiziert.
- Wähle zufällig aus der kleinstnummerierten nicht-leeren Klasse eine Seite zur Ersetzung aus.

Not-Recently-Used (3)

Bemerkungen:

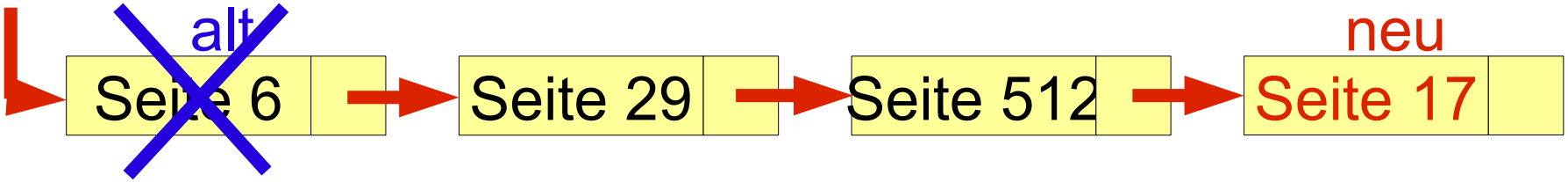
- Einfach.
- Effiziente Implementierung.
- Nicht optimale, aber akzeptable Performance.

9.4.3 FIFO/Second Chance

Hochschule RheinMain
University of Applied Sciences
Wiesbaden Rüsselsheim

First In, First Out (FIFO):

- **Idee:** Die zuerst eingelagerte Seite wird auch zuerst wieder ausgelagert
- Verwaltung über eine Liste:
  - Bei Einlagerung wird Eintrag am Listenende angehängt
  - Bei Seitenfehler: Auszulagernde (älteste) Seite steht im Listenkopf, Listenkopf wird danach entfernt.

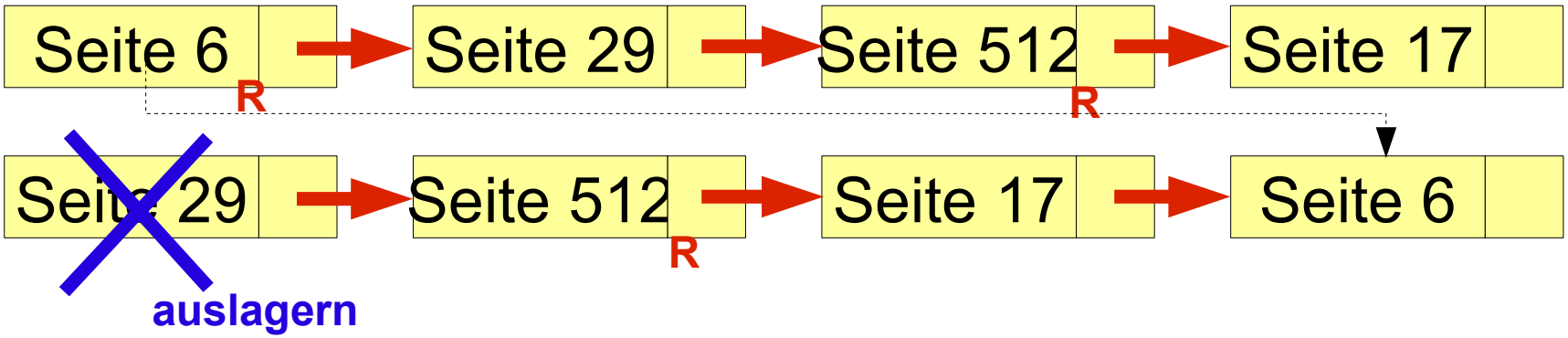


- Eher ungeschicktes Verfahren (die älteste Seite kann trotzdem ständig gebraucht werden, es würde dann bald wieder ein Seitenfehler für diese Seite erzeugt).

Second Chance

Hochschule RheinMain
University of Applied Sciences
Wiesbaden Rüsselsheim

- **Idee:** Ähnlich FIFO, aber mit Beachtung des Referenced-Bits
- Bei Seitenfehler:
  - Vom Listenkopf ausgehend Seiten-Knoten durchlaufen:
    - Wenn R-Bit gelöscht: Seite auslagern, fertig
    - Wenn R-Bit gesetzt: löschen und Seite hinten anhängen (→ Seite erhält eine „zweite Chance“)
  - Sollte überall R-Bit gesetzt sein, degeneriert Verfahren zu FIFO (erster Eintrag ist mit gelöschtem R-Bit wieder vorne)

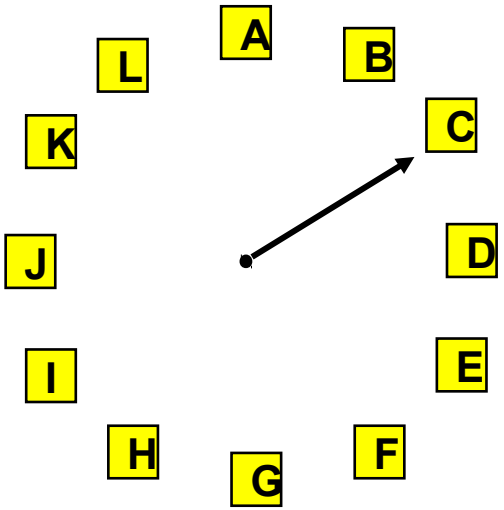


9.4.4 Clock-Algorithmus

Hochschule RheinMain
University of Applied Sciences
Wiesbaden Rüsselsheim

Voraussetzungen:

- **Alle Seiten werden als zyklische Liste in Form einer Uhr verwaltet.**
- **Der Zeiger der Uhr zeigt auf die älteste Seite.**



Algorithmus:

1. Bei Seitenfehler wird die Seite überprüft, auf die der Uhrzeiger zeigt.
2. Ist das Referenced-Bit nicht gesetzt, dann
   - Auslagern der Seite,
   - einzulagernde Seite an dieser Stelle einfügen
   - Zeiger der Uhr auf die nächste Seite vorschieben.
3. Ist das Referenced-Bit gesetzt, dann
   - Löschen des Referenced-Bit
   - Zeiger der Uhr auf die nächste Seite vorschieben
4. Wiederhole Schritt 3 , bis Seite mit einem nicht gesetzten Referenced-Bit gefunden wird

Bemerkung:

- Der Clock-Algorithmus ist eine bzgl. der Implementierung verbesserte Form des Second-Chance Algorithmus.

9.4.5 Least-Recently-Used (LRU)

Hochschule RheinMain
University of Applied Sciences
Wiesbaden Rüsselsheim

Voraussetzungen:

- Gute Annäherung an den optimalen Algorithmus ist möglich, wenn man aus dem Zugriffsverhalten in der jüngeren Vergangenheit auf das Verhalten in der Zukunft schließen kann.
- Beobachtung: Seiten, die schon lange nicht mehr benötigt wurden, werden auch mit hoher Wahrscheinlichkeit für längere Zeit nicht mehr benötigt werden.

Algorithmus:

- Wenn ein Seitenfehler auftritt, ersetze die Seite, die am längsten unbenutzt ist (least-recently-used, LRU).

Least-Recently-Used (2)

Hochschule RheinMain
University of Applied Sciences
Wiesbaden Rüsselsheim

Bemerkungen:

- LRU ist realisierbar, Implementierung ist aber aufwändig.
- Implementierungsidee:
  - Die Menge aller Seiten im Speicher wird mit einer verketteten Liste verwaltet.
  - Die gerade benutzte Seite steht am Kopf der Liste, die am längsten nicht benutzte Seite am Ende.
  - Bei jedem Speicherzugriff wird die zugegriffene Seite in der Liste gesucht und an den Anfang gestellt (Umordnung).
  - Wähle die Seite am Ende der Liste als zu ersetzende Seite.
- Eine effiziente Implementierung ist nur mit spezieller Hardwareunterstützung möglich und nur in Ausnahmefällen sinnvoll (z.B. für die Verwaltung von TLBs mit wenigen Einträgen).

9.4.6 Aging-Algorithmus (Alterung)

Hochschule RheinMain
University of Applied Sciences
Wiesbaden Rüsselsheim

Ziel:

- Approximation von LRU in Software.

Algorithmus:

- Jeder Seite wird Software-Zähler der Länge L Bit zugeordnet (z.B. 8-Bit lang), initialisiert mit 0.
- Bei jeder Uhr-Unterbrechung für jede eingelagerte Seite:
  - Der Zähler der Seite wird um 1 Bit nach rechts verschoben (Aging = Altern).
  - Das Referenced-Bit der Seite ersetzt die führende Stelle des Zählers.
  - Das Referenced-Bit wird gelöscht. (Es wird durch die Hardware im Falle eines weiteren Zugriffs automatisch neu gesetzt).
- Wenn ein Seitenfehler auftritt, wird die Seite mit dem kleinsten Zählerwert ersetzt.

Aging-Algorithmus (2)

Hochschule RheinMain
University of Applied Sciences
Wiesbaden Rüsselsheim

Beispiel:

Seite	R	Zähler	R	Zähler	R	Zähler
0	1	10000000	1	11000000	1	11100000
1	0	00000000	1	10000000	1	11000000
2	1	10000000	0	01000000	0	00100000
3	0	00000000	0	00000000	1	10000000
4	1	10000000	1	11000000	0	01100000
5	1	10000000	0	01000000	1	10100000
Nach Uhrtick:		1		2		3
0	1	11110000	0	01111000	1	10111100
1	0	01100000	1	10110000	0	01011000
2	0	00010000	1	10001000	1	11000100
3	0	01000000	0	00100000	0	00010000
4	1	10110000	0	01011000	0	00101100
5	0	01010000	0	00101000	0	00010100
		4		5		6

Bemerkungen zum Verhältnis zu LRU:

- Information über das Zugriffsverhalten auf eine Seite während eines Uhr-Intervalls wird auf ein einziges Bit vergröbert.
- Information, die älter als L Zeitintervalle ist, wird als wertlos eingestuft (geht durch Rechts-Shifts verloren).
- Eine Seite, die während n Uhrticks nicht referiert wurde, besitzt n führende Nullen im Zähler.
- Die Auswertung des Zählers muss nur nach Ablauf dieses Zeitintervalls erfolgen, nicht bei jeder Speicherreferenz.
- Die Aufzeichnung von nur einem Bit je Zeitintervall erlaubt damit nicht die Unterscheidung einer früheren oder einer späteren Referenz in dem Intervall.
- Insgesamt ist die Approximation von LRU hinreichend gut.

Ziel:

- Kennenlernen von Aspekten, die beim Entwurf von Paging-Systemen besondere Beachtung finden, da sie starken Einfluss auf die Performance haben.

Überblick:

1. Working Set Modell
2. Lokale oder globale Seitenersetzung
3. Seitengröße
4. Implementierungsprobleme

9.5.1 Das Working Set Modell

Hochschule RheinMain
University of Applied Sciences
Wiesbaden Rüsselsheim

- Prozesse zeigen i.d.R. ein Lokalitätsverhalten.
- Def: Die Menge der Seiten, die ein Prozess augenblicklich nutzt, wird sein Working Set (Arbeitsbereich) genannt (P. Denning, 1968).
- Ist der zugeordnete Hauptspeicher zu klein, um das Working Set aufzunehmen, so entstehen sehr viele Page Faults.
- Wenn nach wenigen Zugriffen die Seite wieder benötigt wird, die gerade verdrängt wurde, so sinkt die Ausführungsgeschwindigkeit auf einige Instruktionen je Seiteneinlagerung (typ. mehrere msec). Man bezeichnet eine solche Situation als Thrashing.

Das Working Set Modell (2)

- Einlagern von Seiten, obwohl noch keine entsprechenden Seitenfehler vorliegen, wird als Prepaging bezeichnet (im Gegensatz zum üblichen Demand Paging).
- Ist das Working Set eines Prozesses bekannt, kann mittels Prepaging nach einem Prozesswechsel verhindert werden, dass der Prozess durch eine große Anzahl von Seitenfehlern seine Umgebung zunächst wieder aufbauen muss, in dem das Working Set vollständig eingelagert wird (Working Set Model, Denning, 1970).
- Durch das Working Set Modell kann die Seitenfehlerrate drastisch gesenkt werden.

Das Working Set Modell (3)

- Ansatz zur Bestimmung des Working Sets:
  - Verwendung des Aging-Algorithmus (siehe 9.4.6).
  - Eine 1 in den höherwertigen n Bits des Alterungszählers kennzeichnet die zugehörige Seite als zum Working Set gehörig.
  - Wird eine Seite für n Uhrticks nicht referiert, so scheidet sie aus dem Working Set aus.
- Verbesserung des Clock-Algorithmus (siehe 9.4.4) ist möglich:
  - Seiten, die zum Working Set gehören, werden bei gelöschtem Referenced Bit übersprungen.
  - Dieser Algorithmus wird wsclock (Working Set Clock) genannt.

9.5.2 Lokale oder globale Seitenersetzung

Hochschule RheinMain
University of Applied Sciences
Wiesbaden Rüsselsheim

- Im Mehrprogrammbetrieb haben mehrere Prozesse Seiten im Speicher.
- Lokale Seitenersetzung: bei der Auswahl einer zu ersetzenden Seite werden nur die Seiten des Prozesses selbst betrachtet
- Globale Seitenersetzung: *alle* im Speicher befindlichen Seiten, also auch die anderer Prozesse, werden in Betracht gezogen.
- Lokale Seitenersetzungs-Algorithmen ordnen jedem Prozess eine feste Anzahl von Seitenrahmen zu, globale Algorithmen alloziieren Seitenrahmen dynamisch.
- Globale Algorithmen arbeiten i.a. mit besserer Performance.
- Im Falle eines globalen Algorithmus muss das Betriebssystem dynamisch entscheiden, wie viele Seitenrahmen jeder Prozess zugewiesen bekommt. Einfacher Ansatz: Betrachtung des Working Set, verhindert aber kein Thrashing.

Lokale oder globale Seitenersetzung (2)

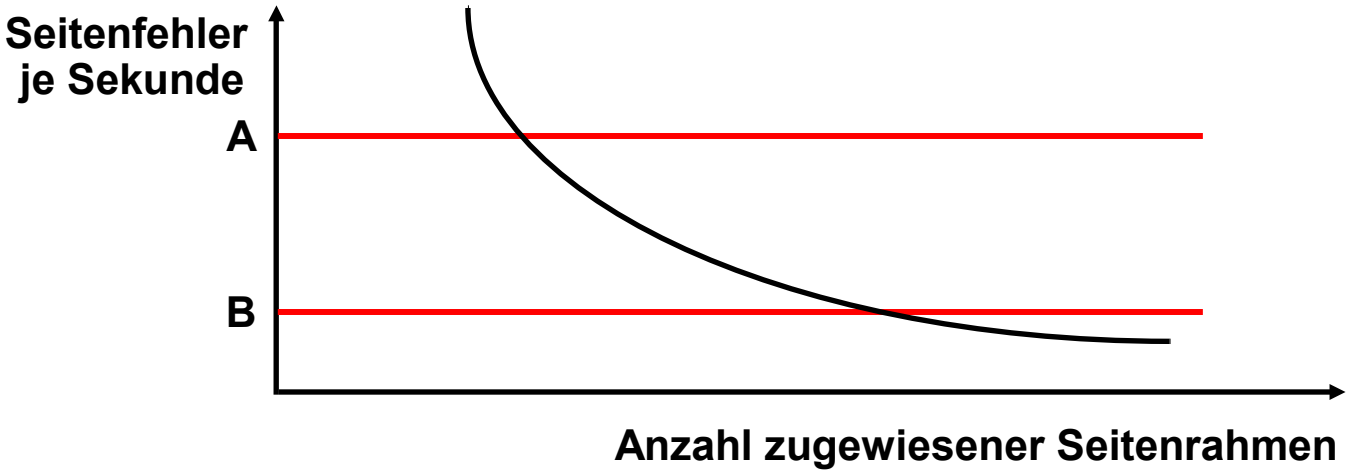
Mindestgröße:

- Wie viele Seiten kann eine einzige Maschineninstruktion referenzieren?
  1. Befehlscode lesen
  2. Quelloperand holen
  3. Zieloperand speichern
  - Operand/Maschineninstruktion i.d.R. > 1 Byte
  - Jeder kann (worst-case) über max. 2 Seiten verteilt sein
- Mindestens 6(*) Seitenrahmen für einen Prozess, sonst u.U. „Thrashing innerhalb einer Instruktion“! (✖)

(*) Architekturabhängig: z.B. MIPS: 2 Seitenrahmen

Lokale oder globale Seitenersetzung (3)

- Bester Ansatz:
  Kontrolle der Seitenfehlerfrequenz (Page Fault Frequency PFF)
  über die Anzahl der zugeordneten Seitenrahmen.
- Typischer Verlauf:



- **Seitenfehlerrate zwischen A und B wird als akzeptabel angesehen. Steigt die Seitenfehlerrate über A, werden zusätzliche Seitenrahmen zugeordnet, sinkt sie unter B, werden Seitenrahmen entzogen.**

9.5.3 Seitengröße

- Bestimmung der optimalen Seitengröße erfordert ein Ausbalancieren von verschiedenen widersprechenden Faktoren:
  - Geringe interne Fragmentierung spricht für eine kleine Seitengröße.
  - Kleine Seiten erfordern eine große Seitentabelle.
  - Kosten für die Ein-/Auslagerung sind stark von Positionierzeiten und Rotationsverzögerung der Platte beeinflusst. Spricht für große Seitengröße.
  - Hardware-Aufwand für Memory Management Unit.
- Heutige Seitengrößen: 512 B - 8 KB, typisch 4 KB

9.5.4 Implementierungsprobleme

Hochschule RheinMain
University of Applied Sciences
Wiesbaden Rüsselsheim

Instruction Retry nach Seitenfehler:

- Eine Instruktion besteht aus mehreren Worten
- Beispiel (MC68000): `MOVE 6(A1), 2(A2)`

1000	MOVE x(A1),y(A2)	Opcode
1002	6	1. Operand
1004	2	2. Operand

- Bei Seitenfehler kann PC=1000, 1002 oder 1004 sein
- „Wieviel von der Instruktion ist schon abgearbeitet?“
- „Wo/wie muss nach Seitenfehler fortgefahren werden?“

→ Kann kompliziert werden ...

Implementierungsprobleme (2)

- Sperren von Seiten im Speicher
  - z.B. für Pufferbereiche während gestarteter I/O-Aufträge.
  - Entsprechende Seiten von blockierten Prozessen dürfen dann nicht ausgelagert werden (reserviert für DMA-Transfer).
  - Typische Operationen: pin / unpin auf Seiten.
- Memory Sharing
  - Gemeinsam benutzte Seiten verschiedener Prozesse (insbesondere Code-Seiten) müssen berücksichtigt werden.
  - Auslagern würde Seitenfehler für Partner verursachen.
  - I.d.R. werden spezielle Datenstrukturen mit Referenzzählern verwendet.

Implementierungsprobleme (2)

Hochschule RheinMain
University of Applied Sciences
Wiesbaden Rüsselsheim

- Paging-Dämonen
  - Ein Dämon (Daemon) ist ein Hintergrundprozess, der nicht mit einem Benutzer in Interaktion tritt (hat kein Terminal).
  - Ziel: Vorhalten einer Anzahl freier Seitenrahmen, um bei Auftreten eines Seitenfehlers keine Verzögerung für das Auslagern einer Seite in Kauf nehmen zu müssen (Performance-Aspekt).
  - Der Page Daemon wird z.B. periodisch aktiviert, um die Speichersituation zu untersuchen:
    - Liegen zu wenige freie Rahmen vor, beginnt er, Seiten mithilfe des Seitenersetzungsalgorithmus zur Auslagerung auszuwählen.
    - Wurden sie modifiziert, veranlasst er das Zurückschreiben auf Platte.
  - Der ursprüngliche Inhalt als „frei“ geführter Seitenrahmen bleibt bekannt.
    - Erfolgt ein erneuter Zugriff auf die ursprüngliche Seite, bevor s überschrieben wird, wird sie aus dem „Frei“-Pool zurückgewonnen.
    - Dieser Vorgang wird Page Reclaiming genannt.

9.6 Segmentierung

Hochschule RheinMain
University of Applied Sciences
Wiesbaden Rüsselsheim

Ziel:

- Bisher wurden Paging-Systeme betrachtet, die einen sogenannten eindimensionalen virtuellen Speicher offerieren.
- Im folgenden wird sogenannter segmentierter virtueller Speicher besprochen.

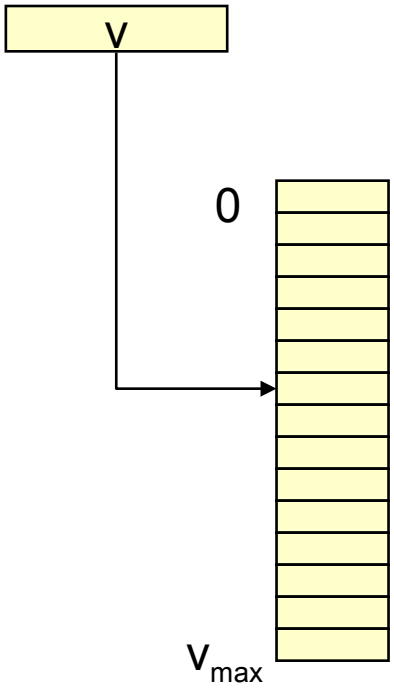
Begriffsbildung

Hochschule RheinMain
University of Applied Sciences
Wiesbaden Rüsselsheim

Eindimensionale / Zweidimensionale virtuelle Adressräume:

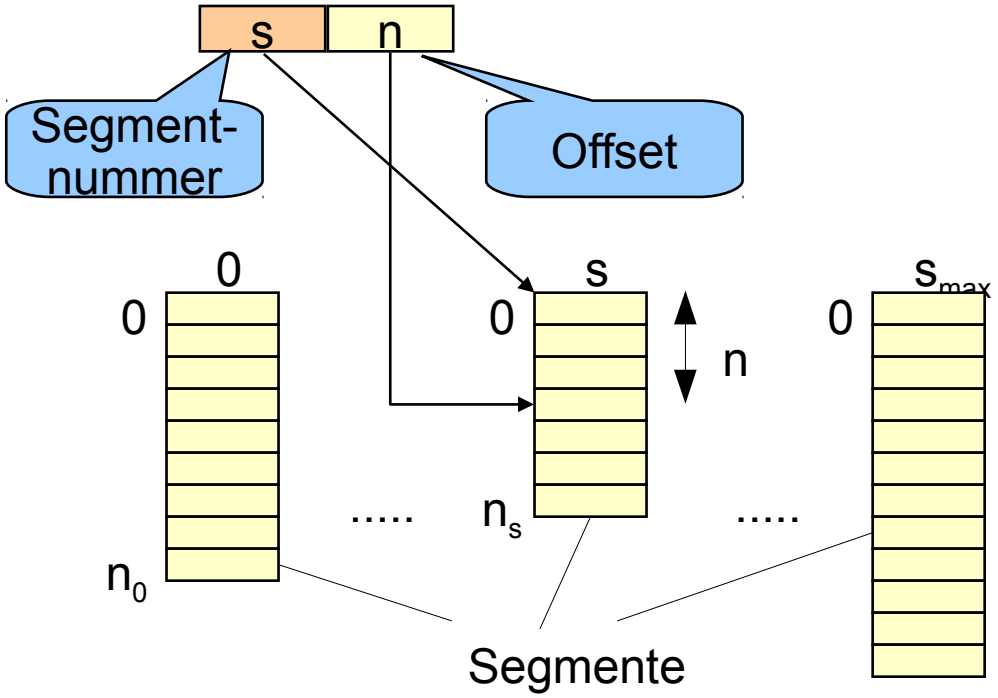
eindimensional:

virt. Adresse



zweidimensional:

virt. Adresse



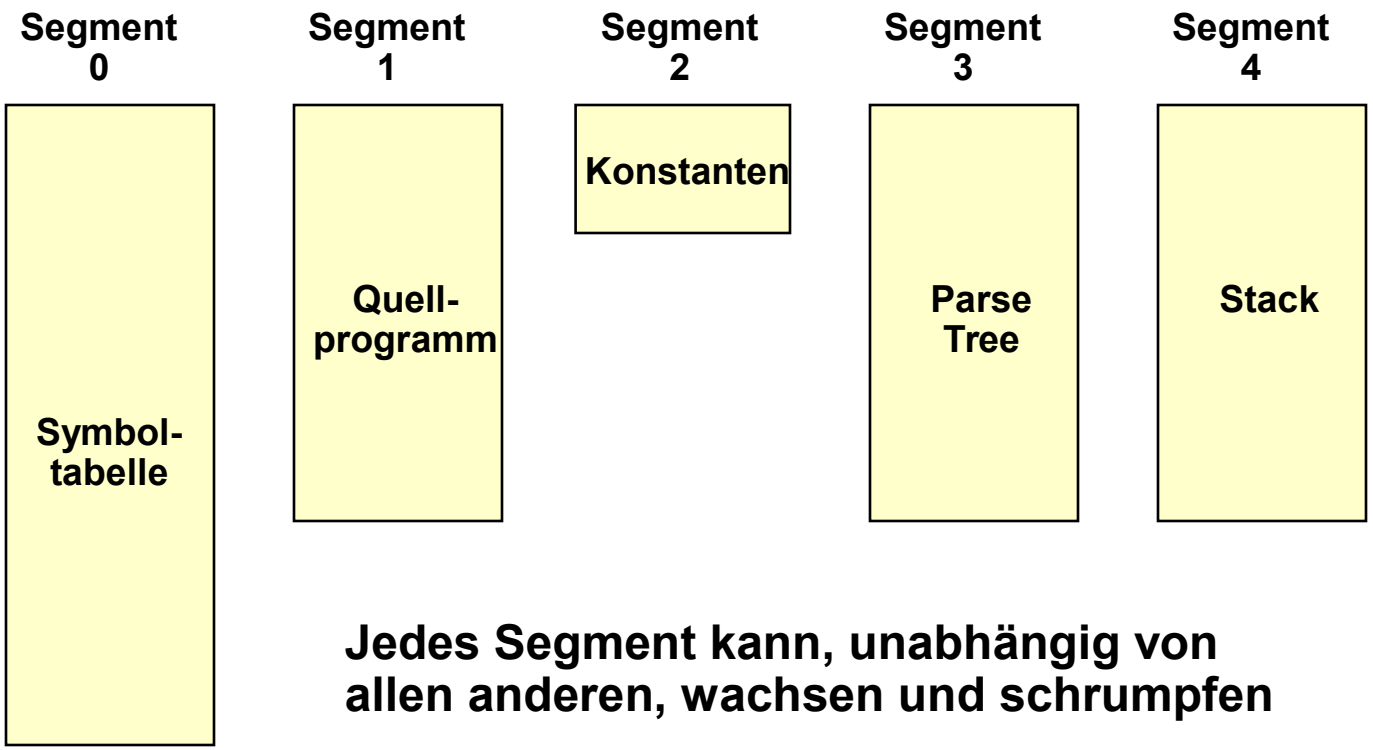
Begriffsbildung (2)

Def

- Zweidimensionale (zweiteilige) virtuelle Adresse:
  - Segmentnummer, die ein Segment selektiert,
  - Adresse im selektierten Segment (Offset).
- Die Menge aller Segmente definiert den sog. Segmentraum.
- Ein Segment besitzt einen maximalen linearen Adressbereich { 0, ..., L_{max}}. Jedes Segment hat damit die Eigenschaften eines (linearen) Adressraums.
- Verschiedene Segmente können unterschiedlich lang sein. Die Länge jedes Segments liegt zwischen 0 und dem erlaubten Maximum L_{max}.
- Die Segmentlänge darf sich i.d.R. während der Lebenszeit des Segments ändern.
- Ein Segment ist eine logische Speichereinheit und als solche dem Programmierer bewusst (sollte es zumindest sein).

Beispiel zur Nutzung

Mögliche Segmente für einen Compiler-Lauf:

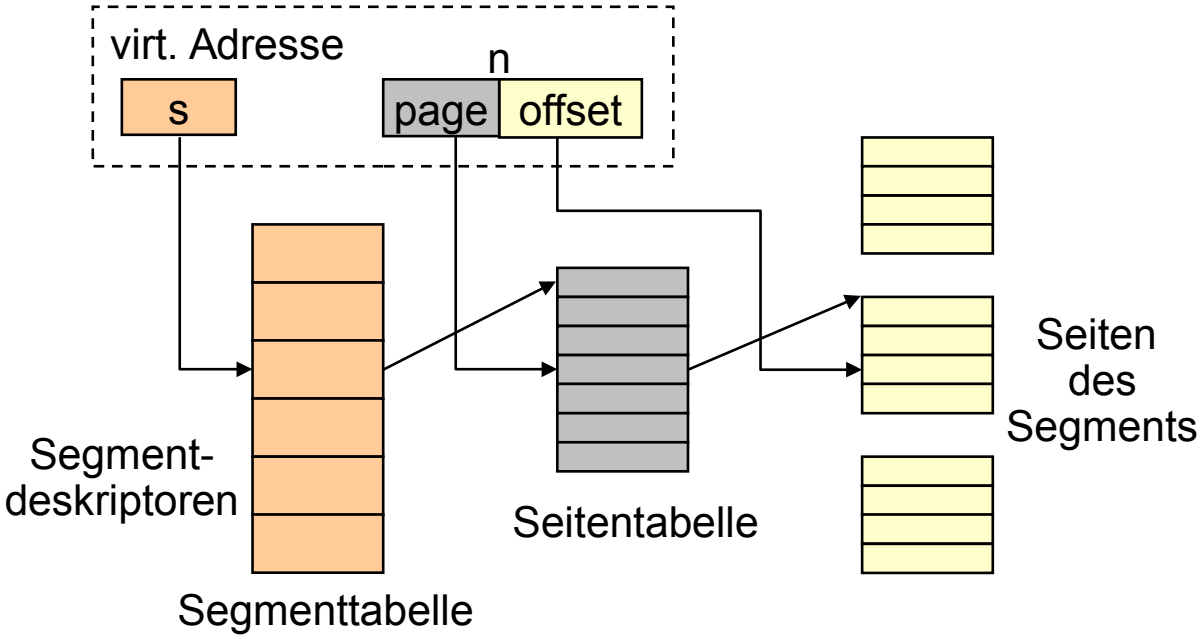


Jedes Segment kann, unabhängig von allen anderen, wachsen und schrumpfen

Segmentierung mit Paging

Hochschule RheinMain
University of Applied Sciences
Wiesbaden Rüsselsheim

- Ein eingelagertes Segment *muss nicht mehr* vollständig im Hauptspeicher sein.
- Jedes Segment besteht aus einer Folge von Seiten.
- Prinzipielle Adressierung:



9.7 Beispiel: Speicherverwaltung in UNIX

Hochschule RheinMain
University of Applied Sciences
Wiesbaden Rüsselsheim

Ziel:

- Kennenlernen der UNIX-Speicherverwaltung.

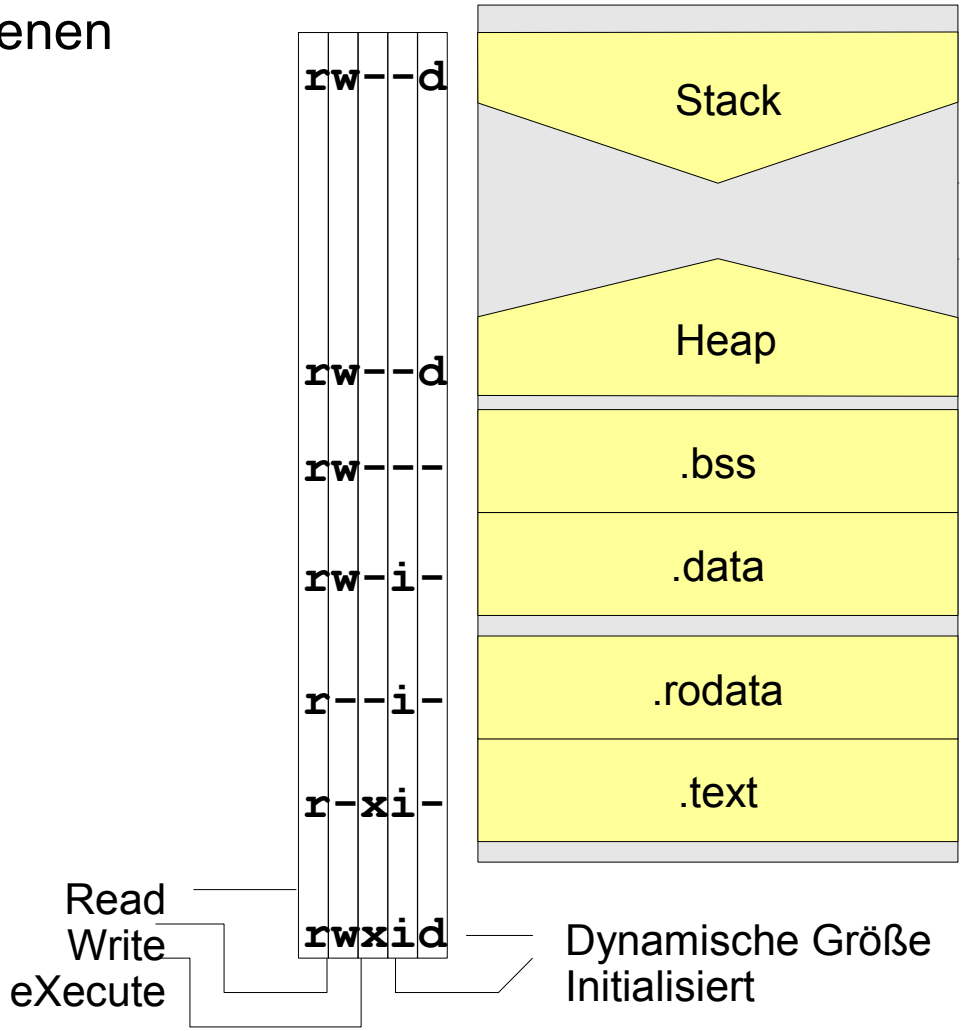
Überblick:

1. Speichermodell.
2. Systemaufrufe zur Speicherverwaltung.
3. Swapping.
4. Paging.

9.7.1 Speichermodell

Hochschule RheinMain
University of Applied Sciences
Wiesbaden Rüsselsheim

- Jeder Prozess besitzt einen eigenen 32-bit virtuellen Adressraum.
- Der Adressraum enthält aus Benutzersicht (im Anwender-adressbereich) 3 Segmente
  1. Text-Segment
     - Programmcode (.text)
     - Nur-Lese Daten (.rodata)
  1 Daten-Segment
     - Initialisierte Daten (.data)
     - Nicht-initialisierte Daten (.bss)
     - Heap (für malloc() & Co.)
  2 Stack-Segment
     - für lokale Variablen etc.

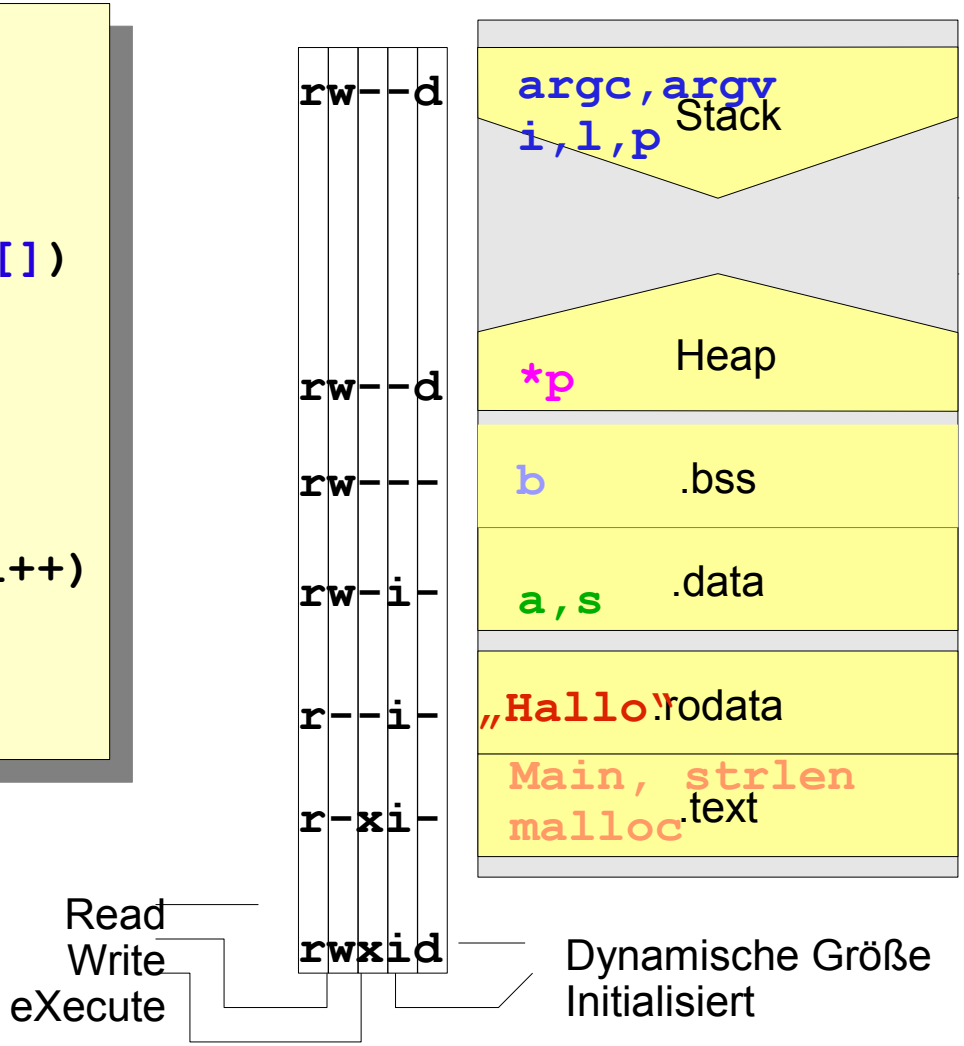


Speichermodell (2)

Hochschule RheinMain
University of Applied Sciences
Wiesbaden Rüsselsheim

```
static int a = 5;
int b;
char *s = „Hallo“;

main(int argc, char *argv[])
{
        int i;
        int l = strlen(s);
        char *p;
        p = malloc(l+1);
        for(i = 0; i < l; i++)
                p[i] = s[i];
}
```

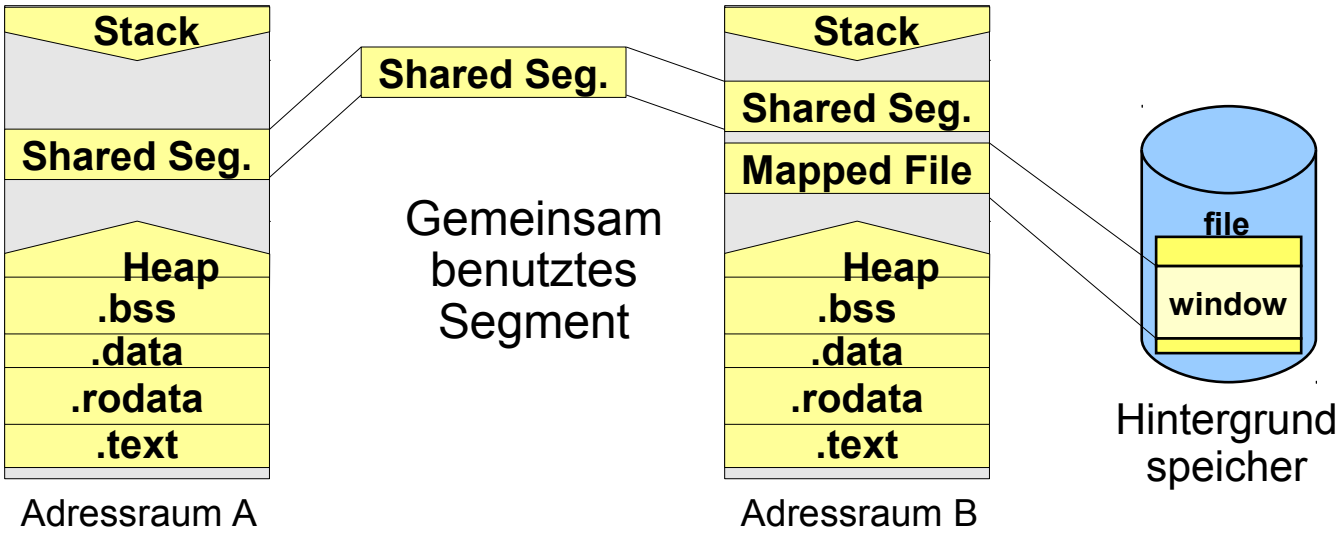


- Text-Segment kann von mehreren Prozessen gemeinsam benutzt werden, Daten und Stack sind privat.

Speichermodell (3)

Hochschule RheinMain
University of Applied Sciences
Wiesbaden Rüsselsheim

- Zusätzlich kann ein Adressraum in neueren UNIX-Systemen ein oder mehrere Segmente der folgenden beiden Typen enthalten:
  - Gemeinsam benutztes Speichersegment, das gleichzeitig in mehrere Adressräume eingeblendet ist (z.B. als Shared Memory Data Segment oder Shared Library),
  - Mapped File-Segment zur Aufnahme eines Fensters eines Memory Mapped Files (abgebildete Datei, siehe Kapitel Dateisysteme). Auf ein Mapped File kann ohne explizite E/A-Operationen zugegriffen werden.



Adressraumregionen

- Virtueller Adressraum wird in Regionen unterteilt.
- Eine Region ist ein zusammenhängender Bereich virtueller Adressen und kann ein identifizierbares Segment (Objekt) aufnehmen (Text-, Daten-, Stack-Segment und andere, s.o.).
- Einem zugeordneten Segment entspricht eine Folge von Seiten des Adressraums.
- Für jede Region können individuelle Schutzattribute festgelegt werden (read-only, read-write, read-execute). Diese übertragen sich auf die repräsentierenden Seiten.
- Gemeinsam genutztes Segment:
  - entspricht Regionen in verschiedenen Adressräumen, denen das Segment zugeordnet ist.
  - Anfangsadresse der Regionen kann in Adressräumen unterschiedlich sein.
    (Achtung: Adressen im gemeinsam benutzten Segment sind dann i.d.R. bedeutungslos).

Überblick:

- Im POSIX-Standard existieren keine expliziten Festlegungen.
- Für die Anwendungsprogrammierung wird auf ANSI C-Standardbibliotheksfunktionen zur dyn. Speicherverwaltung zurückgegriffen.
- Im folgenden werden betrachtet:
  - klassische Dienste zur Behandlung der Standard-Segmente.
  - Dienste für Shared Memory Segmente.
  - Dienste für Memory-Mapped Files.
  - Dienste zur Adressraum-Manipulation.

  Dienste für Memory-Mapped Files sowie zur Adressraum-Manipulation sind noch nicht auf allen heutigen UNIX-Varianten verfügbar.

Dienste für Standard-Segmente

- Zur absoluten bzw. relativen Veränderung der Größe des Datensegments (Wachsen oder Schrumpfen) existieren die Systemaufrufe `brk()` bzw. `sbrk()` (letzterer ist häufig eine Bibliotheksfunktion):

```
caddr_t brk(caddr_t addr); caddr_t sbrk(int incr);
```

- Auf Programmebene werden i.d.R. die ANSI C-Bibliotheksfunktionen `malloc()` und `free()` benutzt. Implementierung von `malloc()` greift z.B. auf `brk()` zurück, wenn die Heap-Größe erweitert werden muss.
- Die Größe des Stack-Segments wird nicht auf Programmebene verändert, sondern vom Betriebssystem implizit vergrößert, wenn sich der Stack zur Ausführungszeit als zu klein erweist.
- Die Größe des Text-Segments ist statisch und damit nicht veränderbar.

Dienste für Shared Memory Segmente

Hochschule RheinMain
University of Applied Sciences
Wiesbaden Rüsselsheim

- Shared Memory Segmente sind Bestandteil der System V IPC-Mechanismen und stellen gemeinsam von mehreren Prozessen benutzbare Segmente mit festlegbaren Zugriffsattributen dar.
- Die Benutzung von Shared Memory Segmenten setzt i.d.R. die korrekte Synchronisation der Prozesse voraus (z.B. mittels Semaphoren).
- Shared Memory Segmente sind i.d.R. die effizienteste Form der Interprozesskommunikation.
- Maximale Anzahl von Shared Memory Segmenten im System (`SHMMNI`) und je Prozess (`SHMSEG`) und deren maximale und minimale Größe (`SHMMAX`, `SHMMIN`) werden bei der Konfiguration des BS-Kerns festgelegt.
- Shared Memory Segmente überleben, wie die anderen System V IPC-Objekte die sie erzeugenden Prozesse und sind ohne besondere Maßnahmen erst mit einem Neustart des Systems zerstört. Zur Information über die existierenden IPC-Objekte bzw. zu ihrer Zerstörung existieren die Kommandos `ipcs` bzw. `ipcrm`.
  (Bitte am Ende jeder Praktikumssitzung zum Aufräumen nutzen !)

Dienste für Shared Memory Segmente (2)

Hochschule RheinMain
University of Applied Sciences
Wiesbaden Rüsselsheim

- Operationen:
  - Header-Dateien: `<sys/types.h>, <sys/ipc.h>, <sys/shm.h>`
  - `int shmget(key_t key, int size, int shmflag):`

  erzeugt oder öffnet ein Shared Memory Segment mit dem externen Schlüssel `key` und liefert den zugehörenden Identifier shmid des Segments zurück (oder -1 bei Fehler). Die Behandlung von `key` und `shmflag` ist insgesamt analog wie bei Semaphoren. Ist insbesondere `key` auf `IPC_PRIVATE` gesetzt, so wird ein privates Segment (ohne öffentlichen key) erzeugt, das für verwandte Pozesse genutzt werden kann. Wird ein Schlüssel verwendet und in `shmflag` das `IPC_CREAT`-Bit gesetzt und existiert noch kein IPC-Objekt zu `key`, so wird ein neues Segment der Mindestgröße `size` erzeugt. Für das Öffnen eines existierenden Objekts darf das `IPC_CREAT`-Flag nicht gesetzt sein. Hat `size` einen Wert größer als die aktuelle Größe des Segments, so schlägt der Aufruf fehl, der Wert 0 akzeptiert das Segment in der aktuellen Größe. Die Zugriffsrechte für das Segment werden durch einen entsprechenden `mode` in `shmflag` bei der Erzeugung mit festgelegt, typisch `0600` für Lesen und Schreiben durch alle Prozesse des Eigentümers des Segments oder `0666` für beliebige Prozesse. Beim Öffnen eines Segments gibt der Zugriffsmodus die geforderten Rechte an.

Dienste für Shared Memory Segmente (3)

- `void * shmat(int shmid, void * addr, int flag)`

blendet das Segment `shmid` in den Adressraum des Aufrufers an der virtuellen Adresse `addr` ein (attach). Wird für `addr` der Wert `NULL` gewählt, so wird die Startadresse des Segments im Adressraum vom Betriebssystem festgelegt (Konflikt ausgeschlossen), ansonsten an der spez. Adresse, die i.d.R. innerhalb des Anwenderadressbereichs auf Seitengrenze ausgerichtet sein muss. Die getroffene Startadresse wird zurückgegeben (oder -1 bei Fehler). Ist in `flag` `SHM_RDONLY` gesetzt, so wird das Segment nur mit Leserecht in den Adressraum eingeblendet. `shmat` muss nach Ausführung von `shmget` und vor der eigentlichen Nutzung des Segments ausgeführt werden.

- `int shmdt(void * addr)`

blendet ein zuvor mittels `shmat` an die Adresse `addr` eingeblendetes Segment wieder aus dem Adressraum des Aufrufers aus (detach). Die Existenz des Shared Memory Segments bleibt hiervon unberührt.

Dienste für Shared Memory Segmente (4)

- `int shmctl(int shmid, int cmd, struct shmid_ds * arg)`

ermittelt/ändert den Status eines Shared Memory Segments oder löscht es. Kommandos (Werte für `cmd`) sind

- `IPC_STAT` (Abfragen des Status mit Ergebnis in einer Stuktur `shmid_ds`, auf die `arg` zeigt).
- `IPC_SET` (Setzen der Eigentümer-UID/GIG und der Zugriffsrechte bei passender effektiver UID).
- `IPC_RMID` (Löschen des Segments bei passender effektiver UID; wird effektiv, wenn der letzte Prozess `shmdt` ausführt, zwischenzeitlich ist kein `shmat` mehr möglich, Referenzzähler).
- `SHM_LOCK` (Sperren des Segments, nur durch Prozess mit root-Recht).
- `SHM_UNLOCK` (Entsperren des Segments, nur durch Prozess mit root-Recht).

Dienste für Memory-Mapped Files

Hochschule RheinMain
University of Applied Sciences
Wiesbaden Rüsselsheim

- Einblenden einer Datei oder eines Gerätes in eine Region des Adressraums (Kombination von Systemfunktionen der Speicherverwaltung, des I/O-Systems und des Dateisystems)
- Eingeblendete, zuvor geöffnete Datei wird wie normales Shared Memory Datensegment zugegriffen, d.h. *ohne* Nutzung der Dienste `read`, `write`, `lseek`.
- Daten werden mithilfe des normalen Paging-Mechanismus aus der hinterlagerten Datei herangeschafft.
- Veränderte Seiten der Datei werden irgendwann, bedingt durch Seitenverdrängung, dorthin zurückgeschrieben, wenn der letzte Prozess die zugrundeliegende Datei schließt oder durch `sync`-Vorgänge.

Dienste für Memory-Mapped Files (2)

- Vorteile:
  - Vermeiden von Kopiervorgängen.
  - Weniger Kontext-Umschaltungen, da I/O-Systemaufrufe wegfallen.
  - Bequemer Zugriff über Zeiger.
  - Veränderungen in der Datei sind für andere Prozesse, die dieselbe Datei eingeblendet haben, *unmittelbar* sichtbar.
- Die Anwendbarkeit des Einblendvorgangs für zahlreiche Geräteklassen erlaubt den Zugriff auf Hardware-Komponenten (z.B. Geräteregister, Graphik-Bildspeicher, usw.), auch aus einem Anwendungsprogramm heraus (hohe Flexibilität, hohe Performance).

Dienste für Memory-Mapped Files (3)

- Operationen (Überblick):
  - Header-Dateien: `<sys/types.h>, <sys/mman.h>`
  - `caddr_t mmap(caddr_t addr, size_t len, int prot, int flags, int fd, off_t offset);`

  blendet ein Fenster der durch `fd` angegebenen, geöffneten Datei (oder Gerät), beginnend in der Datei an der Stelle `offset`, in der Länge `len` in den Adressraum des Aufrufers an der virt. Adresse `addr` mit der Rechtefestlegung `prot` ein.

  - `int munmap(caddr_t addr, size_t len);`

  blendet eine zuvor eingeblendete Datei für der angegebenen Adressbereich aus.

  - `int msync(caddr_t addr, size_t len, int flags);`

  erlaubt in durch `flags` festlegbaren Varianten, den Inhalt der im angegebenen Adressbereich eingeblendeten Datei auf die Originaldatei zurückzuschreiben.

Dienste zur Adressraum-Manipulation

- In neueren UNIX-Varianten (wie System V R.4), die gewisse Echtzeitanforderungen erfüllen, wurden auch Funktionen zur Speicherkontrolle bereitgestellt, etwa um sicherzustellen, dass der BS-Kern einem Echtzeitprozess nicht unfreiwillig Seiten verdrängt und dadurch beim nächsten Zugriff (relativ lange) Pagein-Zeiten anfallen.
- Operationen
  - memctl(...)
  - memprotect(...)
  - mincore(...)
- Hier nicht weiter betrachtet

9.7.3 Swapping

- Das klassische Unix basierte nur auf Swapping
- Demand Paging wurde durch die BSD-Version 3 erstmals eingeführt. Ein heutiges UNIX enthält beide Mechanismen.
- Die Verlagerung zwischen Arbeitsspeicher und Hintergrundspeicher wird durch die obere Schicht des zweistufigen Schedulers durchgeführt. Dieser heisst Swapper (Prozess mit pid 0, unterliegt normalem Scheduling (mit hoher Priorität), läuft allerdings nur im Kernel Mode, in System V R.4 sched genannt).
- Der Swap Space ist mindestens eine Plattenpartition (oder auch eine Swap-Datei, UNIX System V R.4).
- Freispeicherverwaltung im Swap-Space geschieht nach dem First-Fit-Algorithmus mittels verketteter Listen, die alle Freibereiche aller Swap-Partitionen umfassen.

9.7.4 Paging

Hochschule RheinMain
University of Applied Sciences
Wiesbaden Rüsselsheim

Überblick:

- Paging wird im folgenden am Beispiel System V R.4 skizziert (Details in [Goodheart, Cox], siehe Literaturliste). Paging in System V R.4 ist weitgehend an das 4.3BSD-UNIX angelehnt.
- UNIX wendet Demand Paging an für durch das Swapping "eingelagerte" Prozesse.
- Seiten der Segmente aller Adressräume werden dynamisch "on demand" ein- und ausgelagert. UNIX benutzt kein Prepaging.
- Das Paging wird gemeinsam durch den BS-Kern sowie durch einen Prozess erbracht, der als Pager Daemon bezeichnet wird (Prozess mit pid 2, in System V R.4 pageout genannt) und im Kernmodus läuft.
- Der Pager wird periodisch geweckt, um seine Dienste zu erbringen. Insbesondere ist es seine Aufgabe, eine bestimmte Menge freier Seitenrahmen vorzuhalten.

Seitenersetzungsalgorithmus

- Der Seitenersetzungsalgorithmus wird vom Page Daemon (pageout) ausgeführt.
- Der angewendete Algorithmus ist
  - global (betrifft also die Seiten aller Prozesse),
  - eine Verallgemeinerung des Clock-Algorithmus, genannt Two-Handed-Clock (Zwei-Zeiger-Uhr-Algorithmus),
  - verwendet Page Reclaiming.
- Der erste Zeiger der Uhr (fronthand) setzt das Referenced-Bit zurück, der zweite Zeiger (backhand) überprüft das Referenced-Bit und gewinnt freie Seitenrahmen, wenn das Referenced-Bit nicht gesetzt ist. Die Zeiger sind als Indizes in das Seitenverzeichnis implementiert.
- Es ist damit kein voller Umlauf des Zeigers wie bei normalem Clock-Algorithmus erforderlich, bis mit Sicherheit ein erster freier Rahmen gefunden wird. (Performance-Problem wird beseitigt, wenn unmittelbar nach einer Scan-Pause enormer Seitenrahmenbedarf auftritt).

Seitenersetzungsalgorithmus (2)

Hochschule RheinMain
University of Applied Sciences
Wiesbaden Rüsselsheim

- Damit zusätzliche Steuerungsmöglichkeit über den Abstand der beiden Zeiger der Uhr (handspread):

  Werden beide Zeiger dicht beieinander gehalten, so wird das Referenced-Bit nur für sehr häufig benutzte Seiten schon wieder gesetzt sein, wenn der zweite Zeiger vorbeikommt.

  Liegen beide Zeiger maximal auseinander (360° - 1 Seite), so arbeitet der Algorithmus nahezu wie der ursprüngliche Clock-Algorithmus.

Seitenersetzungsalgorithmus (3)

Hochschule RheinMain
University of Applied Sciences
Wiesbaden Rüsselsheim

- Der Pager wird im Normalfall alle 250 ms geweckt. Er überprüft die Anzahl der freien Seitenrahmen (Scan). Es ist sein Ziel, eine gewünschte Anzahl `desfree` (desired) von Seitenrahmen in der Freiliste vorzuhalten (default: 1/16 des Arbeitsspeichers).
- Ist die Anzahl der freien Rahmen größer als der Systemparameter `lotsfree` (default: 1/4 des Arbeitsspeichers), so legt er sich unmittelbar wieder schlafen.
- Ist die Anzahl der freien Rahmen kleiner als `desfree`, so wird der Pager bei jedem Clock-Tick aktiv.
- Sinkt die Anzahl unter `minfree` (default: 1/32 des Arbeitsspeichers), so setzt unfreiwilliges Swapping ein.
- Die Scan-Rate ist hoch (`fastscan` Seiten werden untersucht), wenn die Menge freier Rahmen bei Start des Pagers `destfree` ist und sinkt kontinuierlich auf einen unteren Wert (`slowscan`), gerade wenn `lotsfree` freien Rahmen erreicht sind (ab `lotsfree` ist sie 0).
- `desscan` (zwischen `slowscan` und `fastscan`) legt die Anzahl der Seiten fest, für die der Pager bei einer Aktivierung das Referenced-Bit löscht.

Hochschule RheinMain
University of Applied Sciences
Wiesbaden Rüsselsheim

Was haben wir in Kap. 8 gemacht?

- Konzepte der Speicherverwaltung (Wichtig!).
- Statische Speicherverwaltung als einfachste Verfahren.
- Swapping unterstützt Systeme, deren Prozesse mehr Speicher benötigen als insgesamt zur Verfügung steht, lagern dabei aber stets ganze Prozesse aus und ein.
- Freispeicherverwaltung im Arbeitsspeicher und auf der Platte geschieht durch Verfahren basierend auf Bitmaps, verketteten Listen oder dem Buddy-System.
- Virtueller Speicher unterstützt Systeme, deren Prozesse sehr groß werden können.
- Paging und entsprechende Hardware-Unterstützung wurde im Rechnerarchitekturteil der Vorlesung besprochen.

Zusammenfassung (2)

Hochschule RheinMain
University of Applied Sciences
Wiesbaden Rüsselsheim

- Seitenersetzungsalgorithmen wurden vielfach untersucht. Clock-Algorithmus und Aging sind praktikabel und brauchbar, der optimale Algorithmus und LRU sind dagegen gut, aber nicht praktikabel.
- Modellierungstechniken, wie die besprochenen Distanzketten, können für eine Analyse hilfreich sein.
- In konkreten Paging-Systemen gewinnen weitere Aspekte an Bedeutung, etwa die Abwägung des Working Set Modells gegenüber reinem Demand Paging oder lokale gegenüber globaler Seitenersetzung.
- Abschließend wurden die in UNIX eingesetzten Verfahren zur Speicherverwaltung besprochen.