

Betriebssysteme

Kapitel 3 Speicherverwaltung

Ziele der Vorlesung

- Einführung
 - Historischer Überblick
 - Betriebssystemkonzepte
- Prozesse und Threads
 - Einführung in das Konzept der Prozesse
 - Prozesskommunikation
 - Scheduling von Prozessen
 - Threads
- **Speicherverwaltung**
 - Einfache Speicherverwaltung
 - Virtueller Speicher
 - Segmentierter Speicher
- Dateien und Dateisysteme
 - Dateien
 - Verzeichnisse
 - Implementierung von Dateisystemen
- Grundlegende Eigenschaften der I/O-Hardware
 - Festplatten
 - Terminals
 - Die I/O-Software
- Deadlocks/Verklemmungen
- Virtualisierung und die Cloud
- IT-Sicherheit
- Multiprozessor-Systeme
- Fallstudien

Speicherverwaltung

- Speicher ist ein weiteres wichtiges Betriebsmittel, das sorgfältig verwaltet werden muss
 - Speicherkapazität wird immer größer, aber Programme wachsen schneller als verfügbarer Speicher
- Es gibt Speicherhierarchie von schnellem CPU-Speicher über Festplatten bis hin zu großen Bandlaufwerken



Verwaltung der Speicherressourcen ist Aufgabe des Betriebssystems

Was wünscht sich jeder Programmierer (auch Sie) bezogen auf den Hauptspeicher?

Bedingung:

- im Team?
- Zeit: 2 min



2 min

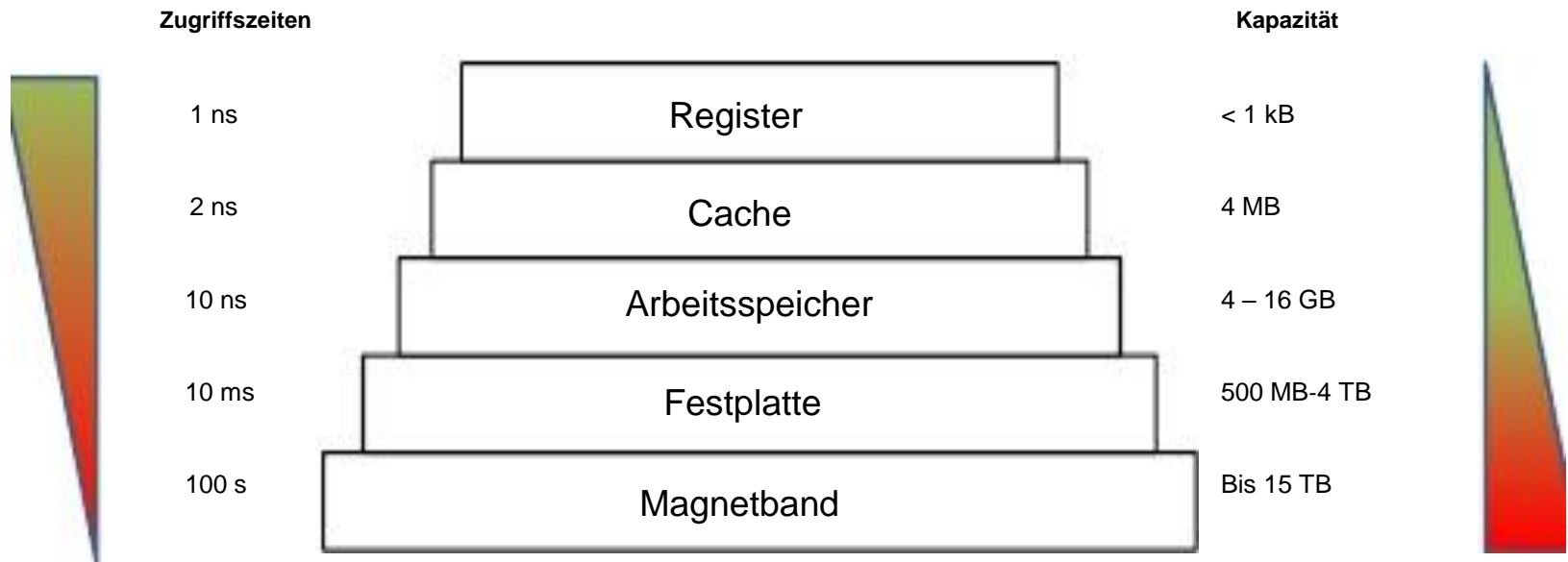
Antwort:

Jeder Programmierer wünscht sich einen Hauptspeicher, der

- nur ihm gehört und nicht geteilt werden muss (privat)
- persistent
- riesig und
- schnell

ist.

- **Speicher**



- Verschiedene Komponenten
 - Verwaltung durch Betriebssystem
 - Optimierung der Speicherverwendung
 - Cache-Verfahren

CPU und GPU – Optimaler Einsatz

- **Was ist eine CPU? Central Processing Unit**

Die aus Millionen oder Milliarden Transistoren aufgebaute CPU kann mehrere Prozessorkerne („Rechenkerne“) haben und wird verbreitet auch als Gehirn des Computers bezeichnet. **Die CPU ist das grundlegende Bauelement aller modernen Computersysteme, da sie die Programmbefehle und die für den Computer und das Betriebssystem erforderlichen Prozesse ausführt.** Außerdem bestimmt die CPU maßgeblich, wie schnell Programme ausgeführt werden können, vom Surfen im Internet bis zur Tabellenkalkulation.

- **Wofür steht die Abkürzung GPU? Graphics Processing Unit**

Die GPU ist ein Prozessor, der aus vielen kleineren und für speziellere Aufgaben konzipierten Kernen besteht. Durch ihr Zusammenspiel bieten die Kerne enorme Leistung, wenn Prozesse der Datenverarbeitung auf viele Kerne aufgeteilt werden können.

- **Was ist der Unterschied zwischen einer CPU und einer GPU?**

- CPUs und GPUs haben viele Gemeinsamkeiten.
 - wichtige Computerelemente
 - auf Halbleitertechnik basierende Mikroprozessoren
 - verarbeiten Daten
- Einsatz für unterschiedliche Zwecke
 - Die CPU eignet sich für eine Vielzahl von Aufgaben, insbesondere für diejenigen, bei denen es auf die **Latenz oder Leistungsfähigkeit pro Kern** ankommt. Als leistungsstarker Baustein für die Ausführung von Programmbefehlen nutzt die CPU die geringere Anzahl der Kerne für einzelne Aufgaben und deren schnelle Erledigung. Damit ist die CPU in besonderer Weise für Jobs von der seriellen Datenverarbeitung bis zum Betrieb von Datenbanken geeignet.
 - Inzwischen haben sich CPUs und die für sie erstellten Software-Bibliotheken zu wesentlich leistungsfähigeren Werkzeugen für Deep-Learning-Aufgaben weiterentwickelt.

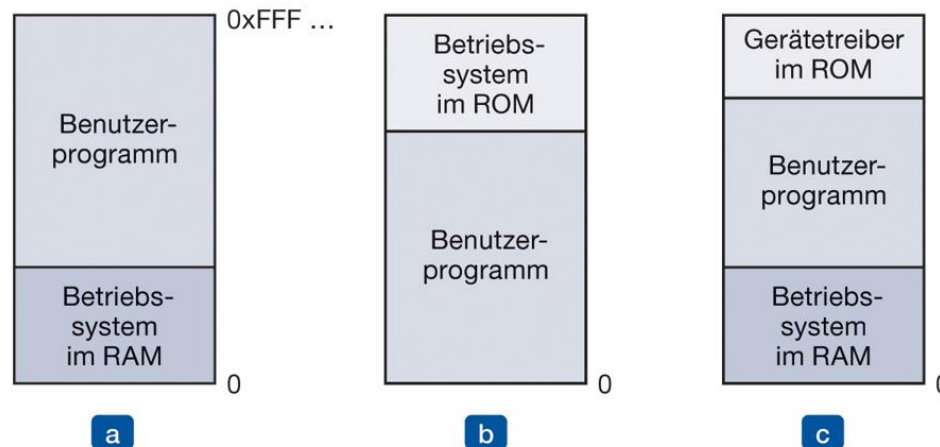
<https://www.intel.de/content/www/de/de/products/docs/processors/cpu-vs-gpu.html>

CPU und GPU – Optimaler Einsatz

- Einsatz für unterschiedliche Zwecke
 - GPUs waren anfangs **spezialisierte ASICs**, die zur Beschleunigung bestimmter 3D-Rendering-Prozesse entwickelt wurden. Im Lauf der Zeit wurden diese Grafik-Engines mit ursprünglich festgelegter Funktion besser programmierbar und flexibler. Während nach wie vor die Grafikdarstellung und die zunehmend naturgetreue Wiedergabe heutiger Top-Computerspiele die Hauptfunktion der GPUs ist, haben sie sich zwischenzeitlich auch zu Mehrzweck-Parallelprozessoren für ein größer werdendes Anwendungsspektrum entwickelt. Gegenwärtig werden GPUs für immer mehr Aufgaben wie Deep Learning und künstliche Intelligenz (KI) genutzt.
 - Für das Deep-Learning-Training mit mehrschichtigen neuronalen Netzen oder mit sehr großen zusammenhängenden Datenmengen, wie etwa 2D-Bildern, eignen sich GPUs oder andere Beschleuniger ideal.
- Die Kombination von CPU und GPU sowie ausreichend großem RAM bietet hervorragende Testmöglichkeiten für Deep Learning und KI.

Speicherverwaltung

- Systeme ohne Speicherabstraktion
 - Großrechner vor 1960, PCs vor 1980
 - Programm hat den gesamten physischen Speicher zur Verfügung
 - Eine Menge von Adressen
 - Untere Grenze: 0
 - Obere Grenze: Maximum des Speichers
 - Jede Adresse enthielt ein Wort (i.d.R. 8 Bits)
 - Jeweils **nur ein Programm in Ausführung (ein Fehler im Anwendungsprogramm kann Betriebssystem aushebeln)**



(a) Großrechner (veraltet)

(b) Handhelds, Embedded

(c) PC (frühe Modelle, ein Teil des Systems im ROM=BIOS))

Abbildung 3.1: Drei einfache Möglichkeiten, den Speicher für einen Benutzerprozess und das Betriebssystem zu organisieren. Es gibt aber noch andere Möglichkeiten.

Speicherverwaltung

- In einem so strukturierten System kann nur ein Prozess laufen
- Benutzer gibt Kommando ein:
 - Betriebssystem lädt das Programm von Platte in Speicher
 - Ausführung
 - Wenn der Prozess beendet ist, Freigabe (Prompt-Zeichen)
 - Benutzer kann neues Kommando eingeben, das Programm wird geladen und überschreibt das Alte
- Eine Möglichkeit für Parallelität in einem System ohne Speicherabstraktion ist die **Programmierung in mehrere Threads**
 - Alle Threads in einem Prozess sehen das gleiche Speicherabbild

Ziel: Voneinander **unabhängige** Programme sollen zur **gleichen Zeit** laufen, das bietet die Thread-Lösung nicht

- Die (gleichzeitige) Ausführung mehrerer Programme funktioniert auch ohne Speicherabstraktion
 - Ein Prozess wird komplett in den Speicher geladen und darf eine gewisse Zeit laufen
 - Betriebssystem speichert dann den gesamten Inhalt des Speichers in eine Plattendatei
 - Betriebssystem holt das nächste Programm (vorhandene Speicherdatei) und führt es aus
 - Solange jeweils nur ein Programm im Speicher ist, kein Konflikt.

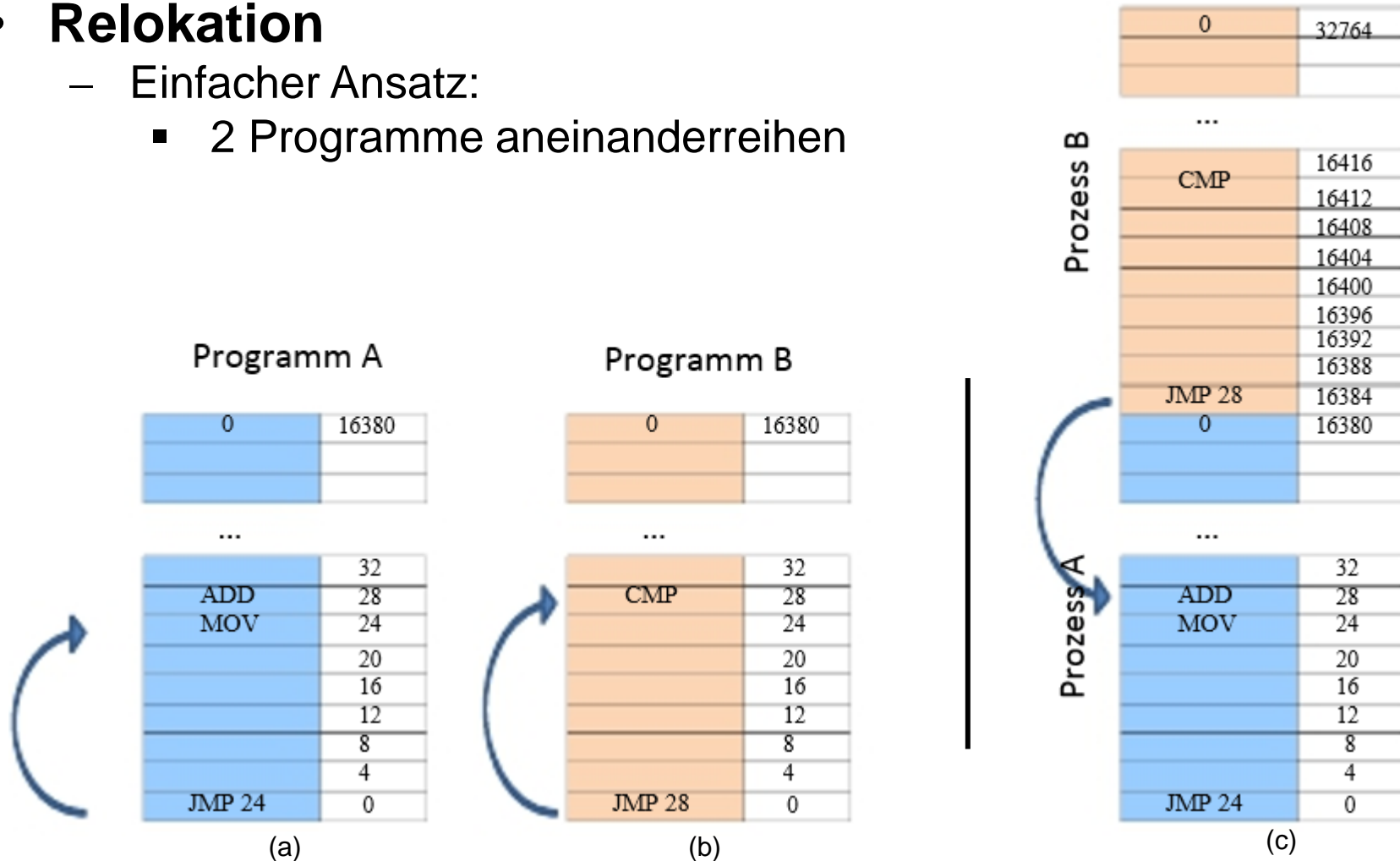
 **Swapping = Verschieben von Daten auf andere Datenträger, besonders um wieder Speicherkapazität zu gewinnen.**

Vorsicht: Ein-/Auslagerung von 1GB dauert Sekunden

- Mit spezieller zusätzlicher Hardware können auch mehrere Programme ohne Swapping parallel laufen (Beispiel IBM 360).
 - Einteilung des Hauptspeichers in 2KB-Blöcke
 - jeder Speicherblock erhält 4-Bit-Schutzschlüssel (in spez. CPU-Register gespeichert)
 - PSW enthält ebenfalls einen 4-Bit Schlüssel
 - Wenn ein laufender Prozess versucht auf den Speicher mit einem Schutzschlüssel zuzugreifen, der sich vom PSW-Schlüssel unterscheidet, löst 360er-Hardware Systemaufruf aus
 - Nur das Betriebssystem konnte den Schutzschlüssel verändern; Benutzerprozesse kommen untereinander und mit dem BS nicht in Konflikt
 - Sonst Verweigerung des Zugriffs

- **Relokation**

- Einfacher Ansatz:
 - 2 Programme aneinanderreihen



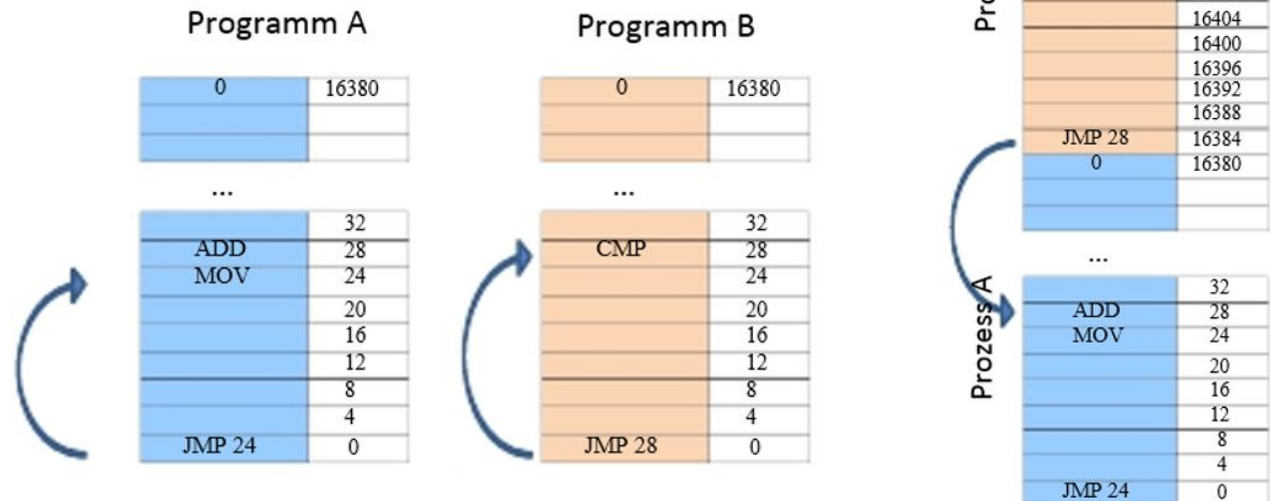
Darstellung des Relokationsproblems: (a) 16 KB-Programm, (b) zweites 16 KB-Programm, (c) zwei Programme wurden hintereinander in den Speicher geladen.

Speicherverwaltung

• Relokation

- 2 Programme, 16 KB groß, mit unterschiedlichen Speicherschlüsseln (32764, 16380)
- Beide Programme werden hintereinander (Speicheradresse 0) in Speicher geladen
- Beim Laden keine Beschädigung des anderen Programms aufgrund der unterschiedlichen Speicherschlüsseln
- 1. Programm funktioniert (JMP 24 ok)
- Betriebssystem entscheidet sich 2. Programm (bei 16384) auszuführen
- Erster Befehl 2. Programm führt zu Speicheradresse des ersten Programms

➔ Absturz, weil absolut phys. Adressen angesprochen werden



Betriebssystem ist im oberen Speicherbereich (nicht abgebildet)

Statische Relokation

- Modifizierung des Prozess B beim Laden in den Speicher
 - 2. Programm wird an **Speicheradresse 16384** geladen
 - Beim Ladevorgang erfolgt Addition von 16384 zu jeder Programmadresse, z.B. aus „JMP 28“ mache „JMP (28+16384)“ also „JMP 16412“
 - **Problem**
 - Es gibt Adressen, die unabhängig von der Position im Arbeitsspeicher gültig sind und nicht neu berechnet werden müssen
 - „MOV REGISTER1, 28“ = Schreibe die Zahl 28 in das Register REGISTER1
 - „28“ darf nicht verändert/verschoben werden
- Wie unterscheidet man „28“ von „28“?
- Das Ladeprogramm braucht ein Verfahren, um zu unterscheiden was eine Adresse und was eine Konstante ist

CMP	16416
	16412
	16408
	16404
	16400
	16396
	16392
	16388
JMP 28	16384
0	16380

...	
	32
ADD	28
MOV	24
	20
	16
	12
	8
	4
JMP 24	0

Konzept Adressräume

- Ziel: Direkter Zugriff auf Speicher ist zu vermeiden
 - Schutz: Programme dürfen sich und das Betriebssystem nicht beeinflussen
 - Relokation: Ausführung mehrerer Programme gleichzeitig ist langsam und kompliziert

Speicherabstraktion Adressraum

- Speicherabstraktion Adressraum
 - Adressraum erzeugt eine Art von abstraktem Speicher, in dem Programme leben können
 - Ein Adressraum ist die Menge von Adressen, die ein Prozess zur Adressierung des Speichers benutzen kann
 - Jeder Prozess hat seinen eigenen Adressraum, der unabhängig von den Adressräumen der anderen Prozesse ist
 - z.B. lokale Telefonnummern, IPv4-Adressen sind 32 Bit-Zahlen (0 bis $2^{32}-1$), .com-Internetdomains

Es muss ein Umrechnung von virtuellen Adressen in physische Adressen geben

Speicherverwaltung

Vergleich: Prozesse / Adressräume

- Prozess
 - Teilt die reale Ressource CPU **zeitlich** in virtuelle Ressourcen auf
 - Zeitliches Multiplexing

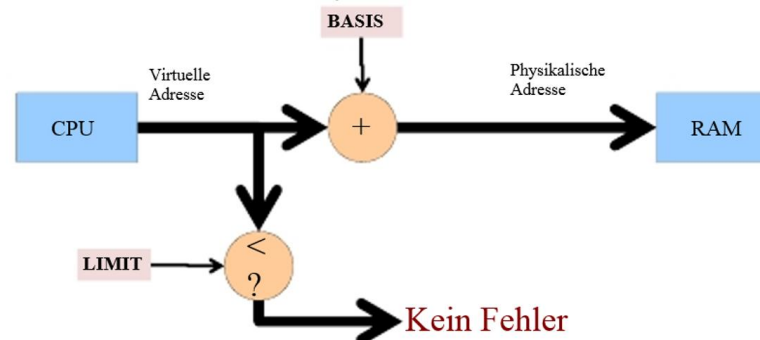


- Adressraum
 - Teilt die reale Ressource Speicher **räumlich** in virtuelle Ressourcen auf
 - Räumliches Multiplexing
 - Jedem Prozess wird ein Adressraum zugewiesen



Adressräume - Realisierung 1: Basis- und Limit-Register

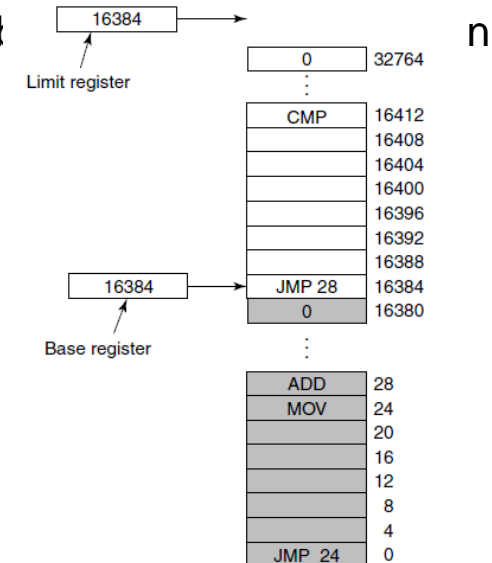
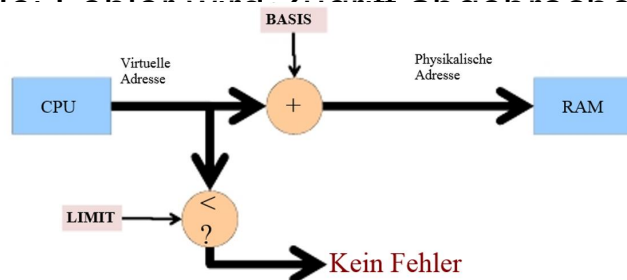
- Einfacher Ansatz der dynamischen Relokation
 - Jeder Adressraum eines Prozesses wird auf einen unterschiedlichen Teil des physischen Speichers abgebildet
 - Jede CPU hat 2 spezielle Hardware-Register: **Basis- und Limit-Register**
 - Basis-Register: untere Grenze des physischen Speichers
 - Limit-Register: obere Grenze des physischen Speichers
 - Beim Zugriff eines Prozesses auf Speicherreferenz
 - Addiert die CPU den Basiswert
 - Prüft, ob die neue Referenz nicht oberhalb des Limits liegt. Wenn ja, gibt's ne Exception
 - Zugriff auf Basis-/Limit-Register nur durch Betriebssystem
 - Ist physischer Speicher groß genug, um alle Prozess aufzunehmen → OK



Speicherverwaltung

Adressräume - Realisierung 1: Beispiel Basis- und Limit-Register

- Prozess A läuft → Laden Basis-Register mit der physischen Adresse, an der das Programm anfängt (im Beispiel 0)
- Prozess A → Limit-Register wird mit Länge des Programms beladen (im Beispiel 16384)
- Für Prozess B Basisregister 16384 und Limitregister 16384
- Ein weiteres 16 KB-Programm C würde über dem zweiten Programm geladen und ausgeführt werden; Basisregister 32768 und Limitregister 16384
- Jedes Mal wenn ein Prozess den Speicher referenziert, um z.B. Befehl zu holen oder ein Datenwort zu lesen bzw. zu beschreiben, addiert die CPU-Hardware automatisch den Basiswert zu der Adresse, die vom Prozess generiert wurde, bevor sie auf den Speicherbus gelegt wird.
- Gleichzeitig wird geprüft, ob die angebotene Adresse gleich oder größer ist als der Wert im Limitregister
- Bei Fehlen wird Zugriff abgebrochen



Was ist der Nachteil des Basis-Limit-Register-Ansatzes?

Bedingung:

→ ad hoc



Ad hoc

Nachteil:

Bei jedem Speicherzugriff

- Addition (langsam wenn keine Additionsschaltkreise benutzt werden) und
- Vergleich (dauert nicht lang)

Adressräume – Realisierung 2: Swapping

- Falls physischer Speicher groß genug ist, um alle Prozesse aufzunehmen → OK
- In Praxis benötigen alle Prozesse mehr als die Gesamtmenge an RAM
 - Beispiele:
 - Beim Hochfahren von typischen Windows, Linux, OS- X-Systemen starten ca. 50-100 Prozesse
 - Hintergrundprozesse wie z.B. Update, email-Daemon,... 5-10 MB
 - Benutzerprogramm, z.B. Photoshop min. 500 MB Speicher
 - SAP Hana
 - Riesige Menge an Speicherplatz notwendig
- Es kommt zu Überlastung des Speichers

Lösungen: Swapping und virtueller Speicher

Swapping (Permanenter Zyklus)

- Prozess wird komplett in den Speicher geladen
- Prozess läuft gewisse Zeit
- Prozess wird komplett auf die Festplatte ausgelagert
- Prozesse im Leerlauf werden meist auf der Platte gespeichert, verbrauchen keinen Hauptspeicher (RAM)

Adressräume – Realisierung 2: Swapping

- a) Prozess A liegt im Arbeitsspeicher
- b) Prozesse B und C werden erzeugt oder von Platte eingelagert
- c) Prozess A, B, C im Speicher
- d) Prozess A wird ausgelagert
- e) Prozess D wird eingelagert
- f) Prozess B wird ausgelagert
- g) Prozess A wird an einer anderen Speicherstelle wieder eingelagert (reloziert)

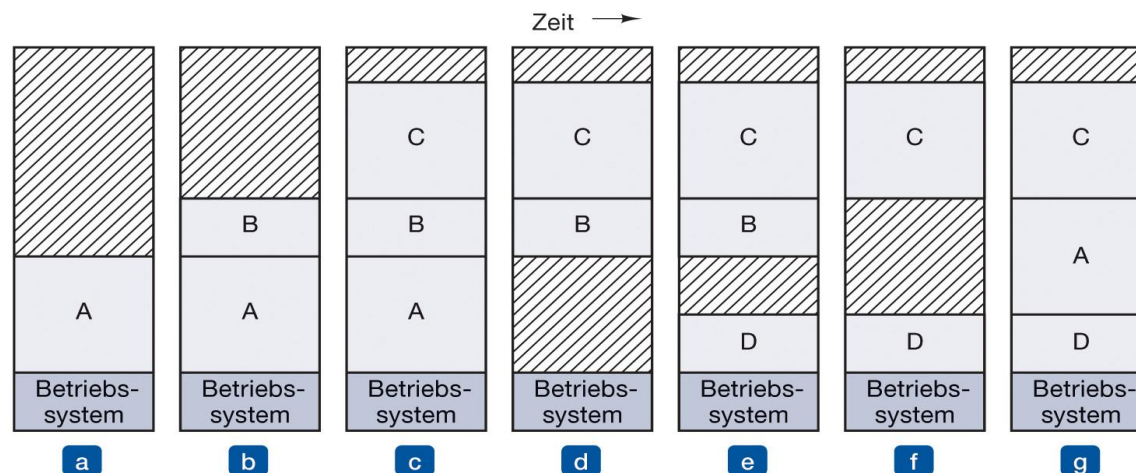
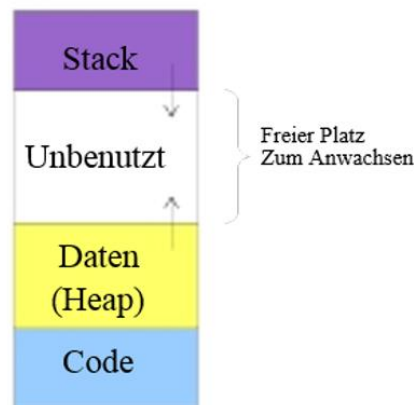


Abbildung 3.4: Mit der Ein- und Auslagerung von Prozessen ändert sich die Speicherbelegung. Die schraffierten Bereiche sind ungenutzt.

Adressräume – Realisierung 2: Swapping

- Probleme:
 - Es entstehen Speicherlücken
 - Speicherverdichtung sehr aufwendig (bei 1 GB dauert das einige Sekunden)
 - Wird nicht eingesetzt, weil zu viel Rechenzeit verbraucht wird (z.B. 16 GB-Maschine kann 8 Bytes in 8 ns kopieren, braucht 16 sec zur Verdichtung des gesamten Speichers)
- Frage: Wieviel Speicher soll für einen Prozess reserviert werden?
 - Einfach, wenn jeder Prozess eine feste Größe hat, die sich niemals ändert. Das Betriebssystem teilt dem Prozess genauso viel zu, wie er braucht.



Adressräume – Realisierung 2: Swapping

- Viele Programmiersprachen erlauben es dynamisch Speicher auf einem Heap zu reservieren. Folge: Datensegmente können wachsen.
- Wenn Prozess versucht auch zu wachsen → Problem
- Grenzt eine Lücke an den Prozess: Prozess kann hinein wachsen
- Liegt der Prozess direkt neben einem anderen Prozess:
 - Wachsender Prozess wird in eine Lücke verschoben
 - Andere Prozesse müssen ausgelagert werden, um eine passende Lücke zu schaffen
- Kann ein Prozess nicht wachsen und der Swap-Bereich auf der Festplatte ist voll
→ Unterbrechung des Prozesses, bis Speicherplatz freigegeben wurde ODER der Prozess wird abgebrochen

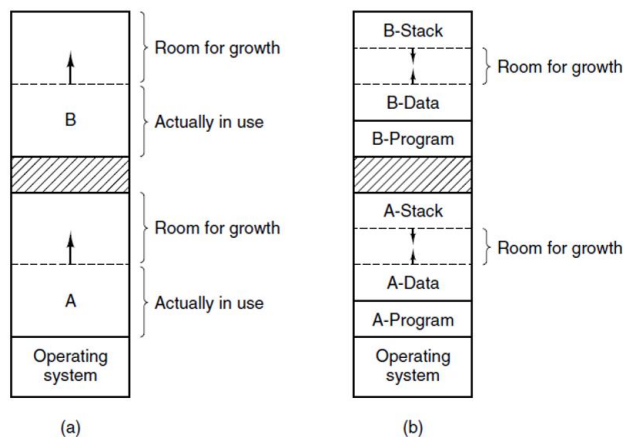
Adressräume – Realisierung 2: Swapping

- Normalerweise wachsen die Prozesse über Ihre Laufzeit
Idee: Beim Einlagern oder Verschieben Reservierung von mehr Speicher
Vorteil: kein Aufwand, wenn Prozess nicht mehr in seinen Speicherbereich passt
- Achtung: Bei Auslagerung eines Prozesses wird nur wirklich benutzter Speicher ausgelagert

Adressräume – Realisierung 2: Swapping

- Beispiel:

- (a) 2 Prozesse haben Platz zu wachsen
- (b) Jeder Prozess hat ein Stacksegment am oberen Ende seines Speicherbereichs (für lokale Variable und Rücksprungsadressen), das nach unten wächst, und ein Datensegment (Heap für dynamische Variable) genau oberhalb vom Programmcode, das nach oben wächst.
- (c) Freier Speicher in der Mitte kann für beide Speichersegmente benutzen.
- (d) Wenn der Platz nicht ausreicht, muss der Prozess in eine größere Lücke verschoben, ausgelagert (bis ausreichende Lücke existiert) oder abgebrochen werden.



- (a) Speicherreservierung für ein wachsendes Datensegment
- (b) Speicherreservierung für wachsende Stack-Datensegmente

Speicherverwaltung

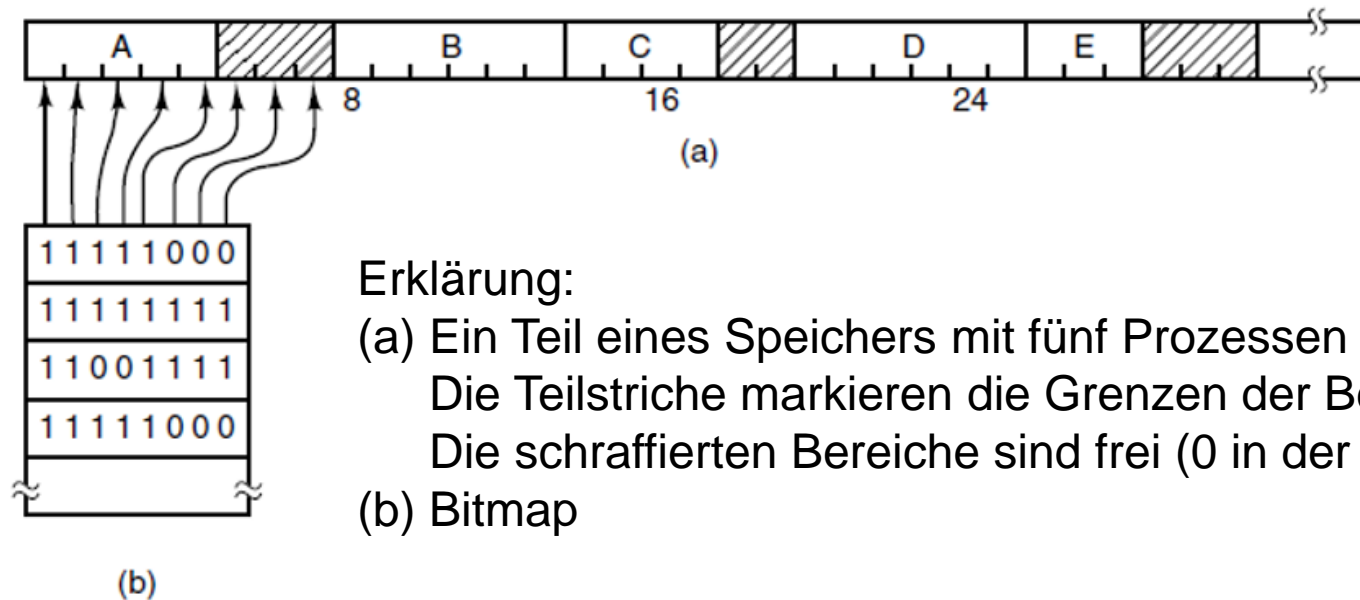
Bei dynamischer Zuteilung des Speichers, muss das Betriebssystem den Speicher verwalten.

Es gibt 2 Möglichkeiten Speicher zu verwalten:

- Speicherverwaltung mit Bitmaps
- Speicherverwaltung mit Freibereichslisten (freelist)

Speicherverwaltung mit Bitmaps

- Bei Speicherverwaltung mit Bitmaps wird der Speicher in Belegungseinheiten unterteilt, die nur einige KB groß sein können
- Jede Einheit entspricht ein Bit in Bitmap (0=Einheit frei, 1=belegt)



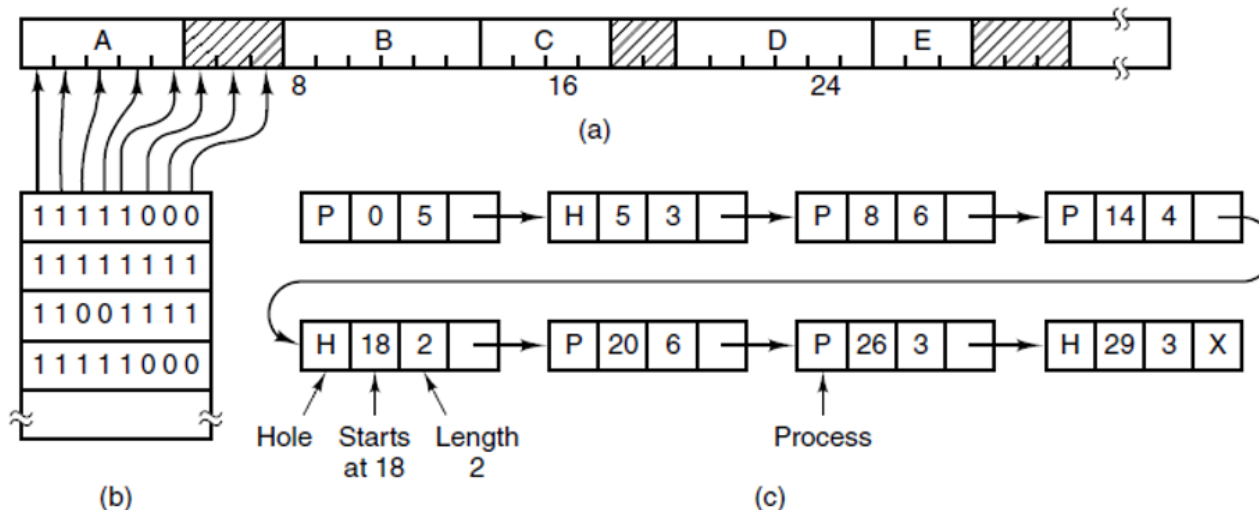
Erklärung:

(a) Ein Teil eines Speichers mit fünf Prozessen und drei Lücken. Die Teilstriche markieren die Grenzen der Belegungseinheiten. Die schraffierten Bereiche sind frei (0 in der Bitmap).

(b) Bitmap

Speicherverwaltung mit verketteten Listen

- Unterscheidung in freie und belegte Speichersegmente
- Ein Segment enthält ein Prozess oder eine Lücke zwischen 2 Prozessen



Erklärung:

(a) Ein Teil eines Speichers mit fünf Prozessen und drei Lücken. Die Teilstriche markieren die Grenzen der Belegungseinheiten. Die schraffierten Bereiche sind frei (0 in der Bitmap).

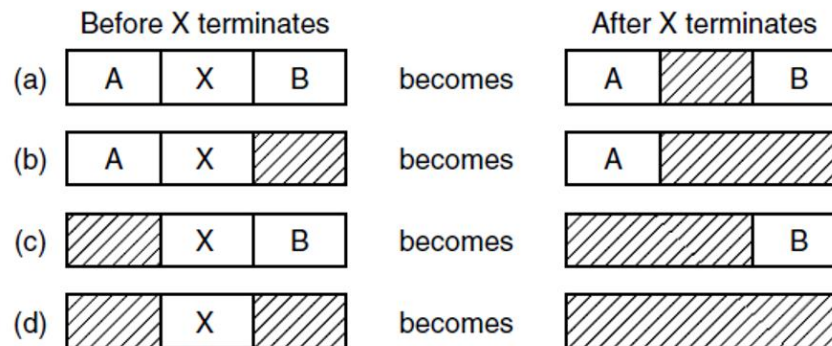
(b) Bitmap

(c) Jeder Eintrag in der Liste bezeichnet eine Lücke (L oder H) oder einen Prozess (P).

Enthalten sind Startadresse, Länge und ein Zeiger auf das nächste Element

Speicherverwaltung mit verketteten Listen

- Liste läßt sich leicht anpassen, wenn ein Prozess ausgelagert oder beendet wird.
- Ein terminierender Prozess hat normalerweise 2 Nachbarn (außer er ist ganz aussen im Speicher), die entweder Prozesse oder Lücken sein können
- Es ergeben sich so vier mögliche Kombinationen



Erklärung: Wenn X terminiert,

- muss in (a) nur ein Prozess durch ein L ersetzt werden
- In (b) + (c) verkürzt sich die Liste um einen Eintrag (Zusammenfassung von 2 Einträgen)
- In (d) verkürzt sich die Liste um 2 Einträge, da 3 Einträge verschmelzen

Speicherverwaltung mit verketteten Listen - Suchalgorithmen

- Um Speicher zu reservieren, kommen mehrere Algorithmen infrage
- Voraussetzung: Verwaltung von Prozessen und Listen nach Adressen geordnet
- Annahme: Speicherverwaltung weiß, wie viel Speicher nötig ist
- Suchalgorithmen
 - First Fit
 - Gehe die Liste durch bis ausreichend große Lücke gefunden ist
 - Next Fit
 - Wie First Fit, aber starte bei der letzten gefundenen Lücke
 - Schlechter als FirstFit
 - Best Fit
 - Suche die gesamte Liste durch und wähle kleinste passende Lücke
 - Schlechter als First Fit, erzeugt viele, sehr kleine Fragmente
 - Worst Fit
 - Verwende am schlechtesten passenden Bereich
 - Schlechter als First Fit, vernichtet große Freibereiche

Virtueller Speicher

- Größe von Computerprogrammen wächst schneller als die Speicherkapazitäten
 - Programme passen nicht mehr in den Arbeitsspeicher
 - Unterstützung von simultaner Ausführung von Programmen
 - Aufteilung des Adressraums eines Programms in kleinere Einheiten (Seiten, pages)
 - Seiten sind aneinander angrenzende Bereiche von Adressen
 - Seiten werden dem physikalischen Speicher zugeordnet
 - Damit das Programm läuft, müssen nicht alle Seiten im physischen Speicher vorhanden sein
 - Greift das Programm auf einen Teil des Adressraums zu, der nicht im Hauptspeicher ist, wird das Betriebssystem aktiv
 - Holt den ausgelagerten Bereich
 - Führt den fehlgeschlagenen Bereich nochmals aus
- Sehr gut bei Multiprogrammierung, da ein anderer Prozess rechnen kann, während der Speicher organisiert wird

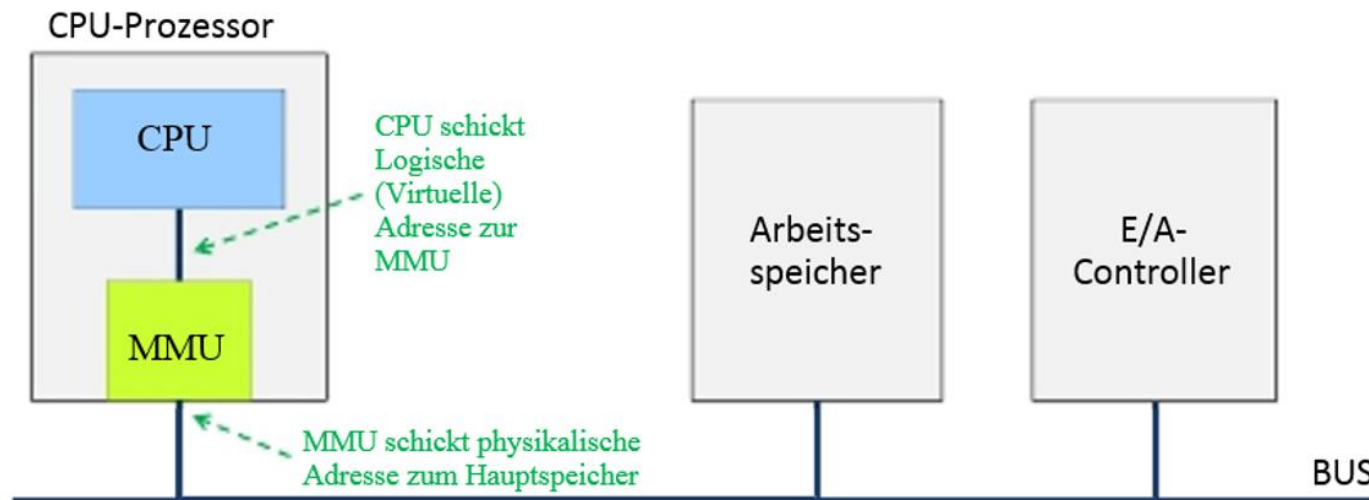
Virtueller Speicher - Paging

- Idee: Bei jedem Speicherzugriff wird die logische Adresse in eine physische Adresse abgebildet
- Trennung zwischen
 - Logischen (virtuellen) Adressen, die der Prozess sieht
 - Physischen Adressen, die der Hauptspeicher sieht
- Bei Systemen mit virtuellem Speicher gehen die virtuellen Adressen nicht direkt auf den Speicherbus sondern auf eine spezielle Hardwareeinheit, die MMU (Memory Management Unit, Speicherverwaltungseinheit)



Virtueller Speicher - Paging

- Vorteile:
 - Kein Verschieben beim Laden eines Prozesses erforderlich
 - Auch kleine freie Speicherbereiche sind nutzbar
 - Speicherschutz ergibt sich (fast) automatisch
 - Teile des Prozessadressraums können ausgelagert werden

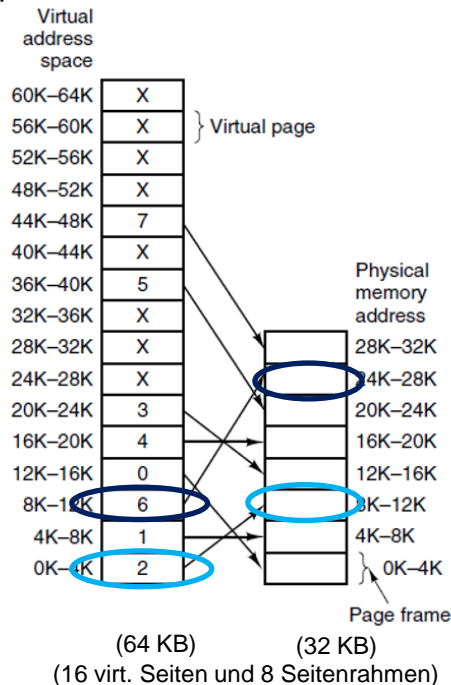


Speicherverwaltung

Virtueller Speicher

- Ziel: Abbildung von virtuellen Adressen auf physische Adressen
 1. Adressräume sind in Blöcke fester Größe unterteilt
 - **Seite** (page): Block im virtuellen Adressraum
 - **Seitenrahmen** (page frame): Block im physischen Adressraum
 - Seiten und Seitenrahmen sind in der Regel gleich groß, typische Größe von 512 Byte bis 1 GB

Beispiel



Erklärung

- Bereich mit 0KB-4KB Beschriftung bedeutet, die (virtuellen oder physischen) Adressen in diesem Bereich gehen von 0 bis 4095.
 - 4KB-8KB bedeuten Adressen von 4096 bis 8191.
 - Jede Seite enthält exakt 4096 Adressen
- Beispiel 1: MOV REG,0
- a) Programm versucht auf Adresse 0 zuzugreifen,
 - b) virtuelle Adresse 0 wird an die MMU geschickt
 - c) MMU stellt fest, dass die virtuelle Adresse (0 bis 4095) dem Seitenrahmen 2 entspricht (8192 bis 12287)
 - d) Adresse wird in 8192 umgewandelt und als Ausgabe auf den Bus gelegt
 - e) Speicher weiß nichts von MMU, er erkennt eine Anfrage zum Lesen oder Schreiben der Adresse 8192, die er bearbeitet.

Beispiel 2:

- a) MOV REG,8192 wird in MOV REG, 245746 umgewandelt

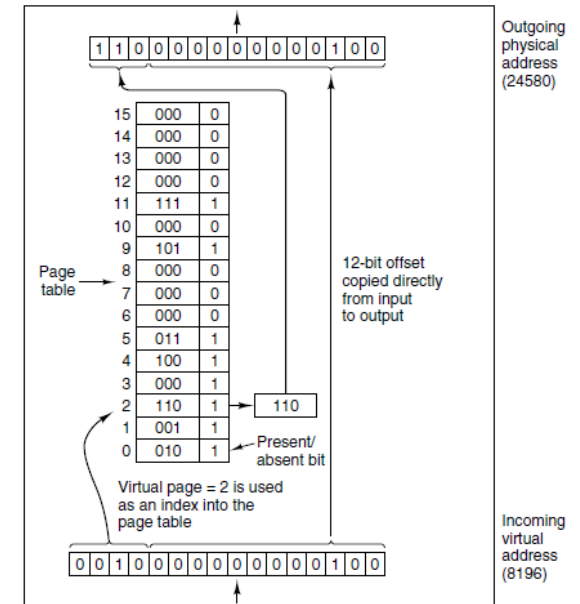
Speicherverwaltung

Virtueller Speicher

- Ziel: Abbildung von virtuellen Adressen auf physische Adressen

2. Betriebssystem erstellt und verwaltet **Seitentabelle**

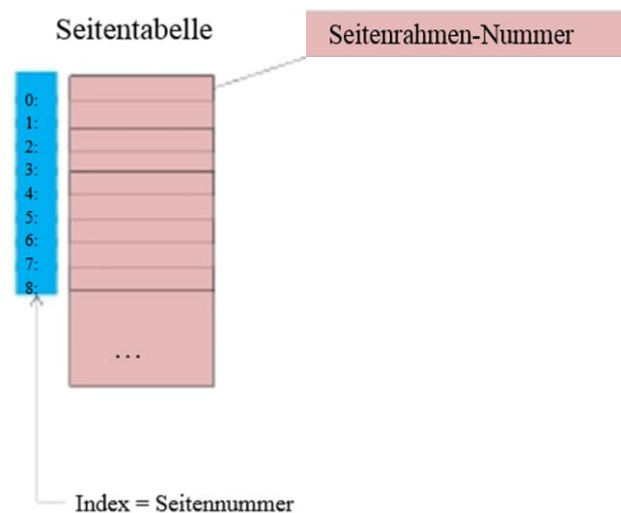
- Virtuelle Adresse wird in eine
 - virtuelle Seitennummer (höherwertige Bits) und
 - einen Offset (niederwertige Bits) geteilt.
 - Beispiel:
 - Ankommende virtuelle 16 Bit-Adresse wird in eine 4 Bit Seitennummer und einen 12 Bit Offset zerlegt.
 - Mit 4 Bit für die Seitennummer lassen sich 16 Seiten ansprechen
 - Mit dem 12 Bit Offset können alle 4096 Byte innerhalb einer Seite adressiert werden
- Virtuelle Seitennummer wird als Index benutzt, um in der Seitentabelle den Eintrag für diese Seite zu finden
- Seitentabelleneintrag enthält die Nummer des entsprechenden Seitenrahmens (falls vorhanden). Die Seitenrahmennummer wird an das höherwertige Offset angehängt und ersetzt die virtuelle Seitennummer.
- Seitenrahmennummer und Offset ergeben die physische Adresse, die an den Speicher geschickt wird.



Die interne Operation der MMU mit 16 4-KB-Seiten
 present-/absent-Bit=Seite vorhanden/Seite nicht vorhanden

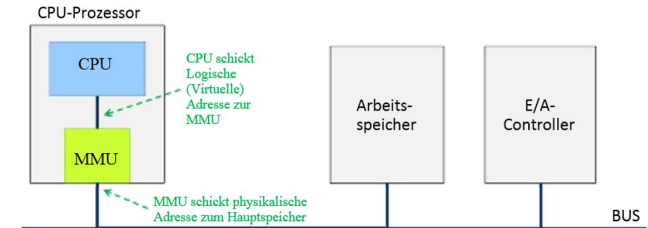
Virtueller Speicher

- Mathematisch:
 - Tabelle ist eine Funktion, mit der virtuellen Seitennummer als Argument und der Seitenrahmennummer als Ergebnis.
 - Mit dem Ergebnis dieser Funktion kann der Teil einer virtuellen Adresse, der die Seitennummer enthält, durch einen Adressteil für den Seitenrahmen ersetzt werden.
- Dadurch wird eine physische Adresse gebildet.



Virtueller Speicher – Allgemeiner Ablauf

1. Prozess 1 greift auf eine (virtuelle) Adresse zu
2. MMU bestimmt die Seite der virtuellen Adresse
3. Mit Hilfe der Seitentabelle wird der Seitenrahmen im physikalischen Speicher bestimmt
4. Seitentabelle bestimmt die Abbildung
5. Falls der Seitenrahmen nicht im physikalischen Speicher ist, (in der Seitentabelle durch ein Bit markiert) wird ein Page Fault erzeugt, und das Betriebssystem reorganisiert den Speicher und geht auf die nächste Seite



MMU wandelt die virtuelle Adresse in eine physikalische Adresse um (Offset berücksichtigen) und greift auf die physikalische Adresse zu (Lesen, Schreiben).

Virtueller Speicher – Ablauf bei Page Fault

- MMU scheitert mit der Abbildung von Seite auf Seitenrahmen
- MMU bemerkt, dass eine Seite ausgelagert ist und löst Systemaufruf aus
- Dieser Aufruf wird Seitenfehler (**Page Fault**) genannt.
- Betriebssystem wählt einen anderen (wenig benutzten) Seitenrahmen aus und schreibt dessen Inhalt auf die Festplatte
- Betriebssystem lädt die angeforderte Seite (von der Platte) in den frei gewordenen Seitenrahmen,
- Betriebssystem ändert die Seitentabelle
- Betriebssystem führt den unterbrochenen Befehl erneut aus

Aufgabe/Frage

- Die Seitentabelle ist die zentrale Entität für die Verwaltung von Adressräumen
- Neben Informationen für das „Management der Indirektion“ können weitere Informationen in der Speichertabelle abgelegt werden.

Frage:

1. Überlegen Sie sich welche Informationen das Betriebssystem für die Wahrnehmung seiner Aufgaben noch in der Tabelle ablegen könnte!
2. Wieviele Speichertabellen sind im System vorhanden?

Bedingung:

→ 2 min

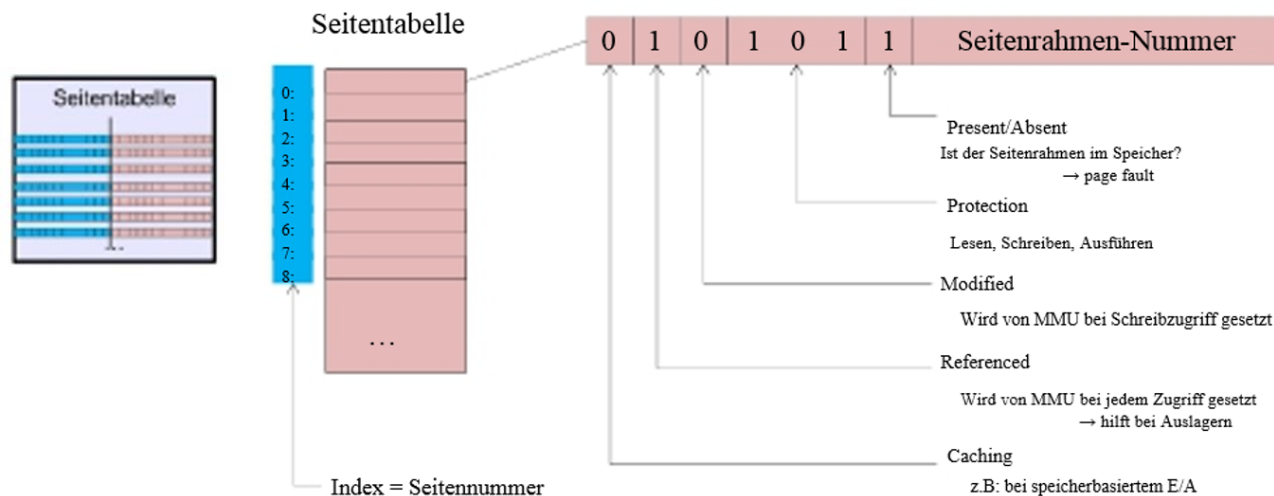


2 min

Antwort zu Frage 1:

Details Seitentabelleneintrag

- Aufbau ist stark maschinenabhängig
- Enthaltene Infos sind von Maschine zu Maschine in etwa gleich
- Größe unterscheidet sich von Computer zu Computer, heute typisch 32 Bit
- Felder:
 - Seitenrahmennummer (wichtig!)
 - Present-/Absent-Bit (Ist Seite im Speicher?)
 - Protection-Bit (Schutzbit regelt den Zugriff; 0=Lese und Schreib; 1=Lese)
 - Modified-/Referenced-Bit (protokollieren die Zugriffe auf die Seite; modified=Schreib; referenced=bei jedem Zugriff)
 - Caching (Regelung des Seitencaching)



Antwort zu Frage 2:

Für jeden Prozess gibt es eine Seitentabelle.

Prozesswechsel führt zum Austausch der Seitentabelle.

Virtueller Speicher - 2 Probleme beim Paging

- **Anforderung 1:** Umrechnung Seite nach Seitenrahmen muss schnell sein
 - Viele Befehle greifen auch auf Operanden im Speicher zu, d.h. es sind 1, 2, oder noch mehr Zugriffe auf die Seitentabelle notwendig. z.B. Dauer Ausführung Befehl = 1ns; Zugriff auf Tabelle $\leq 0,2\text{ns}$
 - Tabellenzugriffe dürfen nicht zum Engpass werden
- **Anforderung 2:** Seitentabelle muss großen virtuellen Adressraum realisieren können
 - virt. Adresse heutiger Rechner meist 64 Bit lang
 - Seitengröße von 4KB und 32 Bit-Adressraum $\rightarrow 1,04 \cdot 10^6$ Seiten $\rightarrow 4\text{ MB}$ notwendig
 - Seitengröße von 4KB und 64 Bit-Adressraum $\rightarrow 4,5 \cdot 10^{15}$ Seiten $\rightarrow 32\text{ PB}$ notwendig
 - Jeder Prozess hat seinen eigenen Adressraum, also auch eine eigene Seitentabelle

Wichtig: Große Seitentabellen mit schnellem Zugriff ist wichtige Randbedingung beim Entwurf eines Computers

Virtueller Speicher

- **Einfache Lösungsdesign**

- Eine **Einzige** Seitentabelle, die aus vielen schnellen Hardwareregistern besteht
 - BS lädt beim Start eines Prozesses seine Seitentabelle in die Register
 - Während Prozess läuft werden keine Speicherzugriffe für die Seitentabelle gebraucht
 - Vorteil: einfach, keine Speicherzugriffe für Umrechnung
 - Nachteile: sehr teuer; sehr langsam beim Prozesswechsel
- Seitentabelle komplett im Arbeitsspeicher
 - Hardware braucht nur **EIN einziges** Register, das auf den Seitenanfang der Seitentabelle zeigt.
 - Vorteil: Zuordnung von virtuellem zum physischen Speicher kann bei Prozesswechsel einfach durch Überschreiben des Registers geändert werden
 - Nachteil: sehr langsam wegen Umrechnung von Speicherreferenzen, die für die Ausführung jedes einzelnen Maschinenbefehls notwendig sind.

- Implementierung Anforderung 1: Beschleunigung Paging durch

Translation Lookaside Buffer (TLB)

- Programme greifen oft auf sehr wenige Seiten zu

Virtueller Speicher

- **Implementierung Anforderung 1: Beschleunigung Paging durch Translation Lookaside Buffer (TLB)**
 - Programme greifen oft auf sehr wenige Seiten zu
 - Kleiner Anteil der Seiten wird ständig gelesen, andere nie
 - Kleine Hardwareeinheit
 - Interner Cache aus Registern
 - verwendet nicht die Seitentabelle, um virtuelle auf physische Adressen abzubilden
 - Teil der MMU
 - Max. 64 Einträge
 - Verwaltung des TLB
 - CISC (x86) Prozessoren: MMU
 - RISC Prozessoren: Software (Betriebssystem)
 - MMU erzeugt Ausnahme
 - Betriebssystem behandelt Ausnahme

Aufgabe/Frage

- Was bedeutet CISC-Prozessor? Beispiel?
- Was bedeutet RISC? Beispiel?

Bedingung:
→ ad hoc

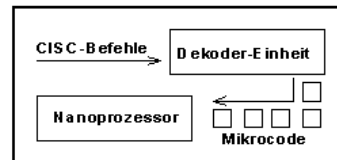


Ad hoc

Antwort:

CISC (Complex Instruction Set Computing)

- großer Befehlsumfang
- komplexe Adressierungsmöglichkeiten
- In der Praxis Verwendung von wenigen Befehlen und Adressierungen (ineffizient)



Funktionsprinzip eines CISC-Prozessors

Jeder Befehl hat einen eigenen Mikrocode im ROM-Speicher des CISC-Prozessors. Bei jedem Befehlsaufruf wird von der Dekoder-Einheit der Befehl in den Maschinenbefehl, die Adressierungsart, die Adressen und die Register aufgeteilt. Danach werden daraus kleine Anweisungen, der Mikrocode, an den Nanoprozessor geschickt. Das ist ein Prozessor im Prozessor, der den Mikrocode in seinen komplexen Schaltkreisen ausführt. Der Mikrocode wird der Reihe nach ausgeführt. Alle anderen Anweisungen müssen solange warten. Erschwerend kommt hinzu, dass der interne Bus die gesamte CPU beim Speicherzugriff ausbremst. Untersuchungen in den 70er Jahren ergaben, dass etwa 80% aller Berechnungen einer typischen Anwendung mit etwa 20% der im Prozessor vorhandenen Befehle ausgeführt wird. Außerdem sind für viele Berechnungen immer die gleichen Befehlsfolgen erforderlich.

Antwort:

RISC - Reduced Instruction Set Computing

- reduzierte Befehlssatz (enthält nur wenige elementare Befehle)
- Mehrzahl der Befehle können innerhalb weniger Taktzyklen ausgeführt werden
- Wenige und einfache Befehle haben einen einfacheren Prozessoraufbau zur
- Alle RISC-Befehle haben das gleiche Format
- es gibt nur eine Möglichkeit sie zu laden oder zu speichern.
- Weil sie nicht in kleinere Mikrocodes dekodiert werden müssen, sind RISC-Befehle viel schneller geladen als CISC-Befehle.
- Praktisch ist heute jeder Prozessor ein RISC-Prozessor oder er hat zumindest RISC-Elemente in sich.
- RISC-Prozessoren sind im wesentlichen billiger herzustellen, kleiner und von der Verschaltung her überschaubarer.

Virtueller Speicher

- Beispiel: Translation Lookaside Buffer, 8 Einträge (real: < 256)
 - Jeder Eintrag enthält Informationen über eine Seite
 - Virtuelle Seitennummer
 - Ein Bit, das gesetzt wird, wenn die Seite modifiziert wird
 - Der Schutzcode (=erlaubt Lesen/Schreiben/Ausführen)
 - Ein Gültig-Bit gibt an, ob der Eintrag momentan benutzt wird

Gültig	Virtuelle Seite	Verändert	Schutz	Seitenrahmen
1	140	1	RW	31
1	20	0	R X	38
1	130	1	RW	29
1	129	1	RW	62
1	19	0	R X	50
1	21	0	R X	45
1	860	1	RW	14
1	861	1	RW	75

Virtueller Speicher

- Beispiel für ein Programm, das den Translation Lookaside Buffer (unten) erzeugen könnte
 - Prozess in einer Schleife, die in den Seiten 19, 20, 21 liegt, so dass diese Seiten Schutzcodes für Lesen und Ausführen haben
 - Daten, die das Programm gerade benutzt, finden sich auf den Seiten 129 und 130
 - Seite 140 enthält die Feldindizes, die bei der Bearbeitung gebraucht werden
 - Seite 860 und 861 enthalten den Stack

Gültig	Virtuelle Seite	Verändert	Schutz	Seitenrahmen
1	140	1	RW	31
1	20	0	R X	38
1	130	1	RW	29
1	129	1	RW	62
1	19	0	R X	50
1	21	0	R X	45
1	860	1	RW	14
1	861	1	RW	75

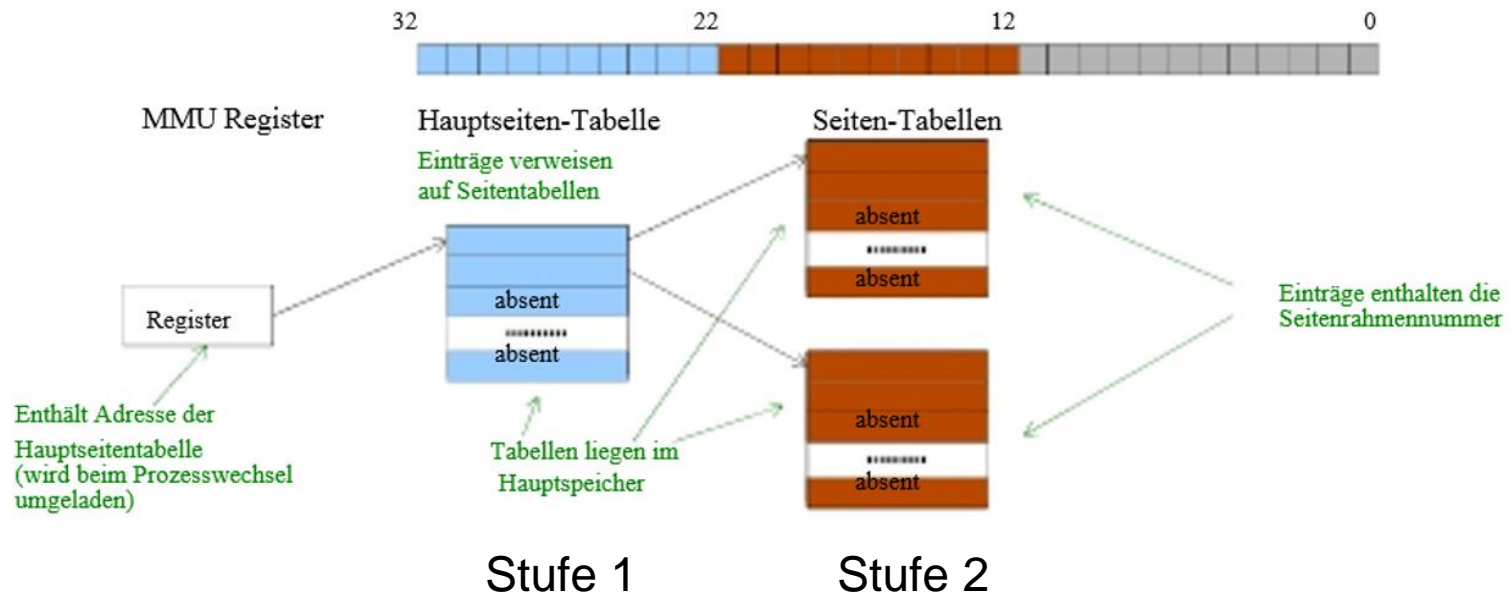
Virtueller Speicher

- Funktion des Translation Lookaside Buffers
 - Virtuelle Adresse wird zur Übersetzung an die MMU geschickt
 - HW überprüft, ob der entsprechende Eintrag im TLB liegt, indem die virtuelle Seitennummer mit allen Einträgen gleichzeitig (d.h. parallel) verglichen wird
 - Wird ein passender Eintrag gefunden und verstößt der Zugriff nicht gegen den Schutzcode, wird die Seitennummer aus dem TLB verwendet, ohne auf die Seitentabelle zuzugreifen
 - Steht die virtuelle Seitennummer nicht im TLB (MMU stellt dies fest)
 - MMU holt den Eintrag ganz normal aus der Seitentabelle und schreibt ihn in den TLB, wobei ein älterer überschrieben wird.
 - Beim nächsten Zugriff auf die Seite steht der Eintrag dann im TLB und muss nicht mehr aus der Seitentabelle geholt werden.

Speicherverwaltung

Virtueller Speicher

- **Anforderung 2:** Seitentabelle muss großen virtuellen Adressraum realisieren können
- **Implementierung Anforderung 2: Große Speicherbereiche durch mehrstufige Seitentabellen (multilevel page table)**
 - Stufe 1: Hauptseitentabelle mit 1024 Einträgen
 - Zeigt auf Stufe 2
 - Stufe 2: Seitentabellen mit je 1024 Einträgen



Virtueller Speicher

- **Implementierung Anforderung 2:** Große Speicherbereiche durch mehrstufige Seitentabellen
 - Bei 64 Bit Systemen explodiert die Anzahl der Tabellen
 - 4 KB Seitengröße → 2 hoch 52 Einträge
8 Byte pro Eintrag → 32 PB für Tabellen
 - **Lösung: invertierte Tabellen**
 - Es wird in der Seitentabelle nicht mehr ein Eintrag pro virtueller Seite angelegt, sondern nur noch je ein Eintrag pro realer Speicherseite.
 - Bei 64 Bit Systemen, 4 KB Speichergröße, 4 GB Speicher
 - 262144 Einträge
 - Jeder Eintrag speichert zu einem Seitenrahmen das zugehörige Paar (Prozess, Seitennummer)
- Verwaltungsaufwand ist größer, weil bei jedem Zugriff auf diese Tabelle wird ein Suchvorgang benötigt, um die virtuelle Adresse in der gesamten Seitentabelle zu finden und die zugehörige reale Adresse auszulesen. Beschleunigung des Suchens in der invertierten Seitentabelle durch das Vorschalten einer Hashtabelle.
- Auch hier Caching mit TLB möglich!

Aufgabe

- Mit dem Virtuellen Speicher haben wir ein abstraktes Modell für die Verwaltung von Speicher kennengelernt
- Es beinhaltet ein Verfahren mit dem wir aus logischen Adressen auf physikalische Adressen umrechnen können.
- Damit ist das Betriebssystem in der Lage Multiprogrammierung mit einfachem Speicherzugriff zu realisieren

Frage:

- Was muss das Betriebssystem bei der Speicherverwaltung noch tun? (Vergleich Konzept „Prozess“)

Bedingung:

→ 5 min



5 min

Antwort:

Seitenersetzung

- Wenn ein Seitenfehler (page fault) auftritt, muss das Betriebssystem Seiten bzw. Seitenrahmen managen/verwalten
 - Seitenrahmen auf Speicher auslagern
 - evtl. auf Festplatte speichern
 - Neue Seite einlesen
 - Seitentabellen aktualisieren
- **Es stellen sich folgende Fragen:**
 - Seitenrahmenersetzung: Welcher Seitenrahmen soll ersetzt werden?
 - Welches ist die beste Strategie?
 - Wie können die Kosten für den Wechsel minimiert werden?
 - Vergleich mit Cache-Problematik
 - Muss die verdrängte Seite dem gleichen Prozess angehören (lokale Strategie) oder kann eine beliebige Seite ausgelagert werden (globale Strategie)?

Seitenersetzung

• Vereinfachter Ablauf

- MMU löst Seitenfehler (Ausnahme) aus
- Betriebssystem ermittelt die virtuelle Adresse (Seite S) und den Grund der Ausnahme
- Falls Schutzverletzung vorlag: Prozess abbrechen, **Fertig!**
- Falls kein Seitenrahmen verfügbar
 - Bestimme die zu verdrängende Seite S',
 - Falls S' modifiziert (M-Bit) wurde: S' auf Platte schreiben
 - Seitentabelleneintrag von S' aktualisieren (u.a. P-Bit = 0)
 - Falls S' nicht modifiziert wurde; Überschreiben!
- Seite S von Platte in freien Seitenrahmen laden
- Seitentabelleneintrag von S aktualisieren (u.a. P-Bit = 1)
- Prozess fortsetzen
 - Abgebrochener Befehl wird fortgesetzt oder wiederholt

Aufgabe/Frage

- Welche einfache Eigenschaft sollte ein optimaler Seitenersetzungsalgorithmus erfüllen?
- Nehmen wir das Beispiel Webserver. Wie funktioniert ein Webserver? (Seitenersetzung?)

Bedingung:

→ 30 sec



30 sec

Antwort:

Zeit zum nächsten Seitenfehler sollte maximiert werden!

Cache

Blöcke im Cache müssen ersetzt werden (alles geht schneller von statten, wie im RAM)

Webserver

- Die viel benutzten Seiten liegen in einem Cache
- Wird eine weitere Seite bei vollem Cache benötigt, muss entschieden werden, welche Seite aus dem Speicher entfernt werden soll
- Webseiten werden niemals im Cache modifiziert → müssen nicht auf die Platte geschrieben werden

Seitenersetzungsalgorithmus

- Betriebssystem muss wissen, welche Seite wann in Zukunft aufgerufen wird
 - in die Zukunft kann man nicht schauen!
 - Das Betriebssystem kann nicht wissen, wann auf welche Seite zugegriffen wird
 - Evtl. kann ein Testlauf helfen, bessere Strategien für ein Programm zu entwickeln, diese sind dann aber nicht allgemeingültig
- **Computer mit virtuellem Speicher haben meist Status-Bits (R und M), die es dem Betriebssystem erlauben, nützliche Statistiken über die Benutzung von Seiten aufzustellen**
- M-Bit wird gesetzt, wenn Seite modifiziert wurde
 - R-Bit wird gesetzt, wenn auf die Seite zugegriffen wird (Lese- und Schreibzugriff, R-Bit wird vom Betriebssystem regelmäßig gelöscht.

Seitenersetzungsalgorithmus

- **Not-Recently-Used (NRU)**

- Verwende die durch MMU gesetzten Status-Bits in Seitentabelle
 - M-Bit: Seite wurde modifiziert
 - R-Bit: Seite wurde referenziert (R-Bit wird vom Betriebssystem regelmäßig gelöscht)
- Bei Verdrängung gibt es 4 Prioritätsklassen
 - Klasse 0: nicht referenziert nicht modifiziert
 - Klasse 1: nicht referenziert modifiziert
 - Klasse 2: referenziert nicht modifiziert
 - Klasse 3: referenziert modifiziert
- NRU entfernt eine zufällige Seite aus der niedrigsten nicht leeren Klasse
 - Vorteil: einfach, leicht verständlich
 - Nachteil: nicht optimal, aber in vielen Fällen ausreichend

Seitenersetzungsalgorithmus

- **First-In-First-Out (FIFO)**

- Verdränge die Seite die am längsten im Hauptspeicher ist
- Betriebssystem verwaltet Liste mit allen Seiten im Speicher
- Beim Wechsel: Entferne den ersten Eintrag und hänge den neuen Eintrag an das Ende der Liste
 - Einfache, aber schlechte Strategie
 - Zugriffsverhalten wird ignoriert
 - Nur selten eingesetzt!

Seitenersetzungsalgorithmus

- **Second-Chance**

- Variante von FIFO, die das R-Bit verwendet
- Verdränge die älteste Seite, auf die seit dem letzten Seitenwechsel nicht zugegriffen wurde
- Betriebssystem verwaltet Liste aller Seiten im Speicher, geordnet nach Alter
 - Älteste Seite steht als erste in der Liste
- Beim Wechsel:
 - Falls erste Seite der Liste $R=0$: Seite entfernen, Fertig!
 - Sonst: Setze $R=0$, schiebe Seite ans Ende der Liste,
 - Gehe zu Schritt 1
 - Falls zu Beginn alle Seiten $R=1$ hatten, wird die älteste Seite beim zweiten Durchlauf verdrängt!

Seitenersetzungsalgorithmus

- **Second-Chance** (sucht nach einer alten Seite, auf die in den letzten Timerintervallen nicht zugegriffen wurde)

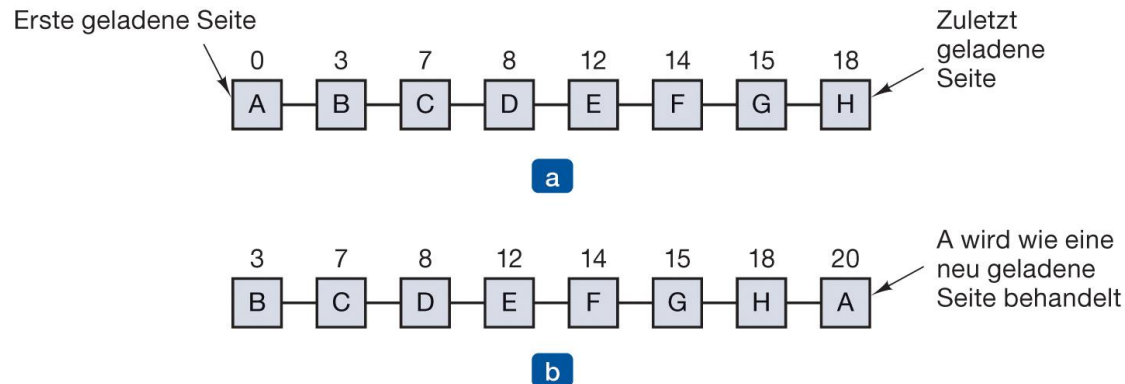


Abbildung 3.15: Arbeitsweise von Second Chance (a) Seiten in FIFO-Ordnung sortiert (b) Die Seitenliste nach einem Seitenfehler zum Zeitpunkt 20, falls bei A das R-Bit gesetzt war. Die Zahlen sind Ladezeiten.

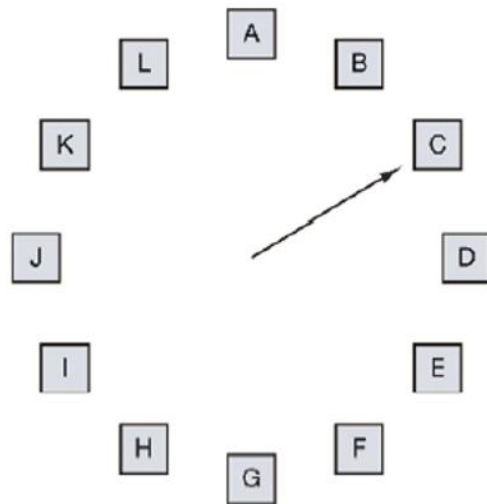
Erklärung:

- Verkettete Liste der Seiten A bis H, geordnet nach der Zeit, zu der sie geladen wurden (nach FIFO-Ordnung sortiert)
- Annahme: Zum Zeitpunkt 20 gibt es einen Seitenfehler. Die älteste Seite ist A, die beim Prozessstart zum Zeitpunkt 0 geladen wurde. Wenn das R-Bit von A **nicht** gesetzt ist, wird A aus dem Speicher genommen, entweder auf die Platte zurückgeschrieben (wenn M-Bit gesetzt, =1 → oder einfach überschreiben (wenn M-Bit=0).
Wenn das R-Bit **gesetzt** ist, wird A an das Ende der Liste verschoben und die Ladezeit auf die aktuelle Zeit (20) gesetzt, R-Bit wird gelöscht. Suche nach passender Seite geht bei B weiter.

Seitenersetzungsalgorithmus

• Clock

- Alle Seiten werden in einer ringförmigen Liste (Uhrform) gehalten
- ‚Uhrzeiger‘ zeigt auf die älteste Seite
- Effizientere Implementierung von Second Chance
 - Das Verschieben von Seiten in der Liste ist ineffizient



Bei Seitenfehler:

Betrachte Seite, auf die der Zeiger zeigt:

R=0: verdränge Seite, fertig

R=1
setze R=0
setze Zeiger eins weiter
wiederhole von vorn

Seitenersetzungsalgorithmus

- **Least Recently Used (LRU)**

- Verdränge die Seite, die am längsten nicht benutzt wurde
 - Vermutung: Die Seite wird auch in Zukunft nicht mehr benutzt
- Verkettete Liste mit neustem Eintrag am Anfang und dem am längsten nicht benutzten Eintrag am Ende
 - Problem: Bei jedem Speicherzugriff muss die Liste aktualisiert werden
 - Eine Seite zu finden, sie zu löschen und anschliessend an den Anfang der Liste zu verschieben, ist sehr aufwendig
 - Hardware-Unterstützung notwendig, aber selten vorhanden



Guter Algorithmus

Seitenersetzungsalgorithmus

- **Not Frequently Used (NFU)**

- Software-Variante von LRU
- Jede Seite hat einen Softwarezähler, der zu Beginn auf gesetzt ist.
- Bei jedem Timerinterrupt durchläuft das Betriebssystem alle Seiten im Speicher und addiert zu jedem Zähler das R-Bit der zugehörigen Seite (0 oder 1).
- Die Zähler zählen mit, wie oft auf jede Seite zugegriffen wird.
- Bei einem Seitenfehler, wird die Seite mit niedrigstem Zählerstand ersetzt

Problem: „Zähler vergisst nichts“

→ Beispiel Multipass-Compiler

- Seiten, die im ersten Durchlauf viel benutzt wurden, könnten auch in späteren Durchläufen noch einen hohen Zählerstand haben

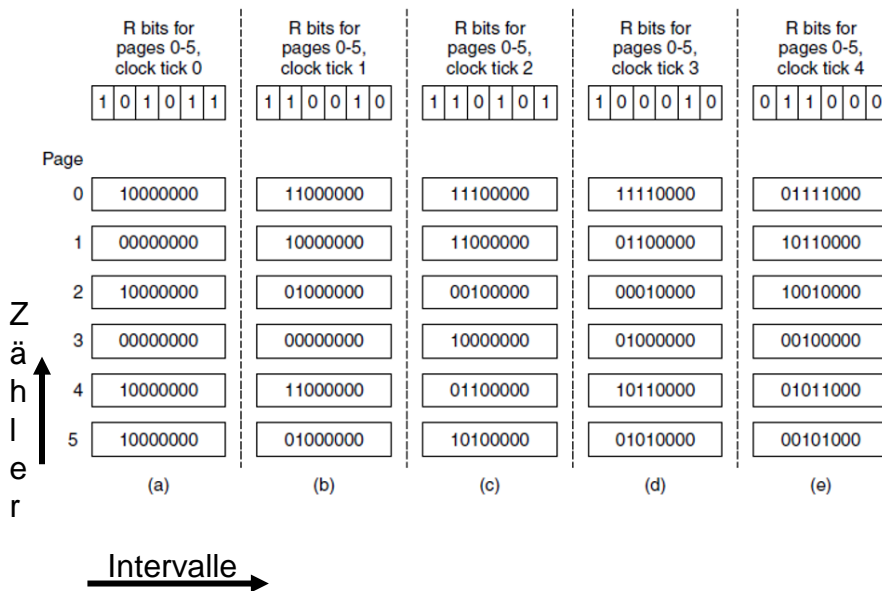


Optimierung:

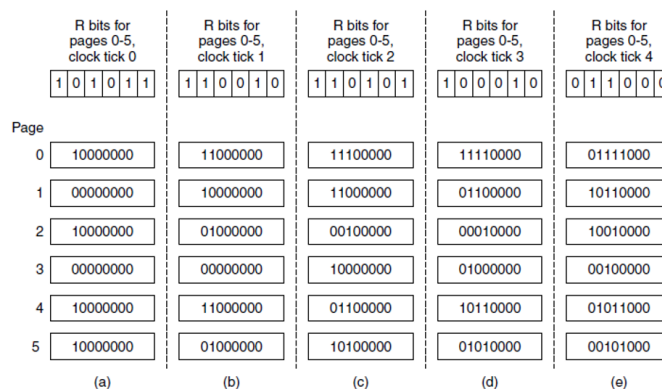
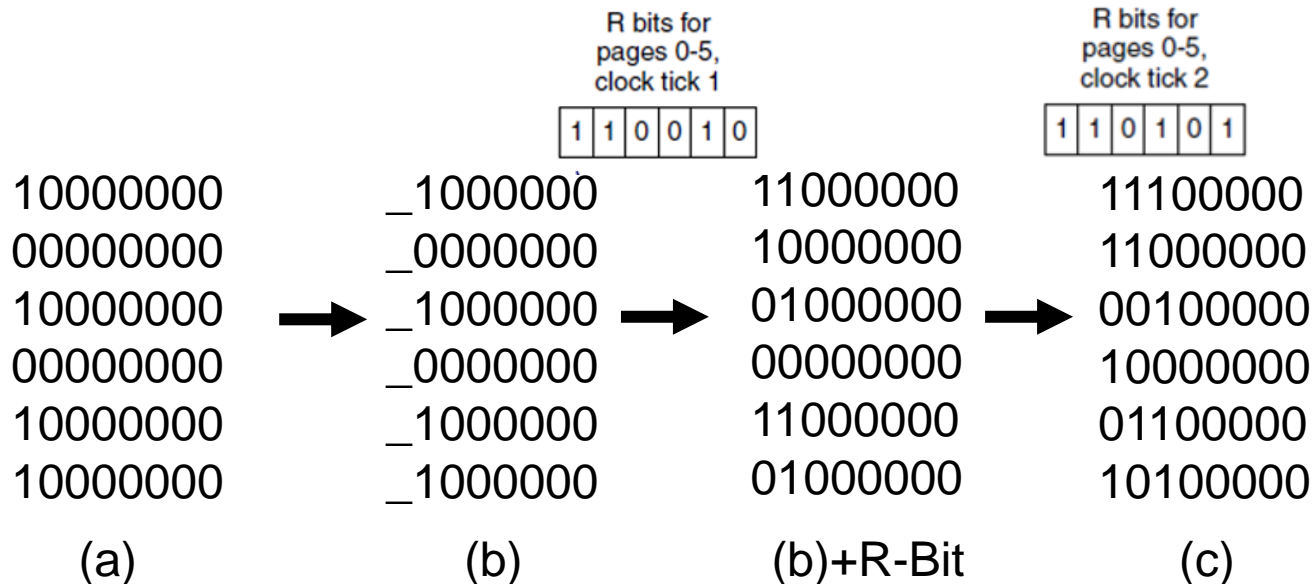
1. Zähler werden immer um ein Bit nach rechts verschoben, bevor das R-bit addiert wird.
2. Das R-Bit wird zum linken Bit des Zählers (höchstwertigen) addiert.

Seitenersetzungsalgorithmus - Aging

- Diesen veränderten Algorithmus nennt man Aging
 - Veränderung von NFU
 - Zähler werden immer ein Bit nach rechts geschoben, bevor R-Bit addiert wird
 - R-Bit wird zum ganz linken Bit des Zählers addiert
 - R-Bits löschen



Seitenersetzungsalgorithmus – Aging Details



Erklärung:

- a) Die Zähler der Seiten werden um ein Bit nach rechts verschoben und die R-Bits von links eingefügt
- c),d),e) Zustände nach den nächsten vier Intervallen

Aufgabe/Frage

- Überlegen Sie was beim Starten eines Programms passiert!
- Wieso kann der Start des Programms mit den hier beschriebenen Verfahren sehr lange dauern?
- Wie kann die Startphase beschleunigt werden?

Bedingung:

→ 60 sec



1 min

Antwort:

- Prozess startet mit keiner einzigen Seite im RAM.
- Versucht die CPU den ersten Befehl zu laden, gibt es einen Seitenfehler und das Betriebssystem lagert die Seite ein, die den ersten Befehl enthält
- Es folgen weitere Seitenfehler
- Nach einer Weile hat der Prozess die meisten Seiten zusammen, die er braucht.
- Jetzt kommt es zu relativ wenigen Seitenfehlern



Demand Paging (Einlagern bei Bedarf)

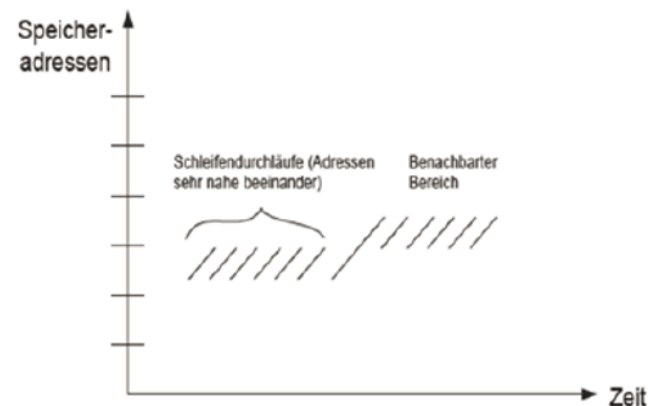
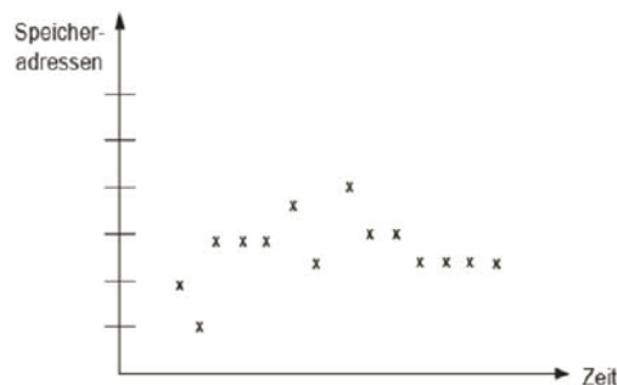
- Beim Start eines Programms kann es lange dauern, bis alle benötigten Seiten geladen sind
- Beschleunigung der Startphase durch Vorausladen von Pages



Die demnächst benötigten Seiten sollten gleich in den Speicher geladen werden

Seitenersetzungsalgorithmus

- Lokalitätseigenschaft
 - Prozesse beschränken ihre Zugriffe in jeder Phase ihrer Ausführung auf einen relativ kleinen Teil ihrer Seiten
 - Speicherzugriffe eines Programms sind selten gleichmäßig über den Adressraum verteilt, sie konzentrieren sich auf einige wenige Seiten
→ Nur ein Teil der Seiten muss im Speicher sein



Seitenersetzungsalgorithmus

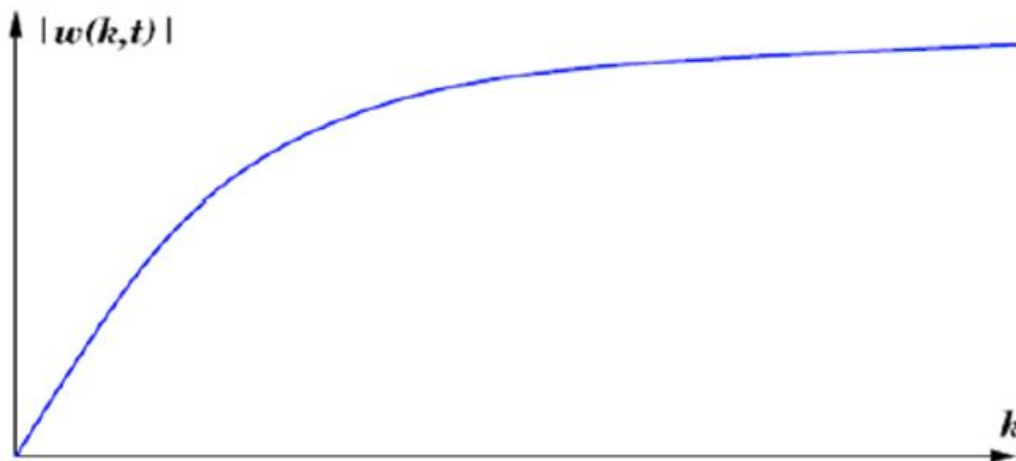
- Working Set (Arbeitsbereich) eines Prozesses P:
 - Wenn der Arbeitsbereich im Speicher ist, läuft der Prozess ohne viele Seitenfehler bis zur nächsten Phase seiner Ausführung
 - Wenn der Arbeitsbereich nicht in den Speicher passt, führt das zu vielen Seitenfehlern und damit zur langsamen Ausführung
 - Seitenflattern (Trashing):
 - Es werden regelmäßig Seiten mit hohen Kosten ein -und ausgelagert (Festplatte)
 - eine Ausführung eines Befehls liegt im ns Bereich, eine Seite von der Platte zu lesen ca. 10 ms
 - Bei Multiprogrammierung werden Prozesse oft ausgelagert
 - Evtl. viele Seitenfehler
 - Viele Betriebssysteme merken sich den Arbeitsbereich eines Prozesses, wenn sie ihn auslagern, und sorgen dafür, dass er wieder geladen wird, bevor sie den Prozess weiter ausführen



Working-Set-Modell (Seitenfehlerrate reduziert sich),
Prepaging (Seiten laden, bevor sie gebraucht werden)

Seitenersetzungsalgorithmus

- **Working Set** (Arbeitsbereich) eines Prozesses P
 - Ist die Menge der Seiten zur Zeit t , die P bei den letzten k Speicherzugriffen benutzt hat
 - Die Funktion $w(k,t)$ ist die Größe des Arbeitsspeichers



Größe des Arbeitsspeichers in Abhängigkeit von k

Erklärung:

- Die meisten Programme greifen zufällig auf eine kleine Anzahl von Seiten zu (steiler Anstieg)
- Mit der Zeit ändert sich das langsam
- Für große k annähernd konstant, aber noch kleiner als der Prozessadressraum

Seitenersetzungsalgorithmus

- Betriebssystem muss zu jedem Zeitpunkt wissen, welche Seiten im Arbeitsbereich eines Prozesses sind
 - Bei Page Fault:
 - Lagere eine Seite aus, die nicht zum Arbeitsbereich gehört
 - Problem:
 - Wie findet das Betriebssystem die Seite, die nicht zum Arbeitsbereich gehört?
 - Es muss Kriterium für Arbeitsbereich geben, welche Seiten zu einem bestimmten Zeitpunkt zum Arbeitsspeicher gehören
 - Der Arbeitsbereich ist die Menge der Seiten, die seit den letzten k Speicherzugriffen benutzt wurden (Siehe oben)
 - Der Wert von k muss im Voraus festgelegt werden
 - Nach jedem Zugriff ist die Menge der Seiten, die von den letzten k Speicherzugriffen benutzt wurden, eindeutig festgelegt.

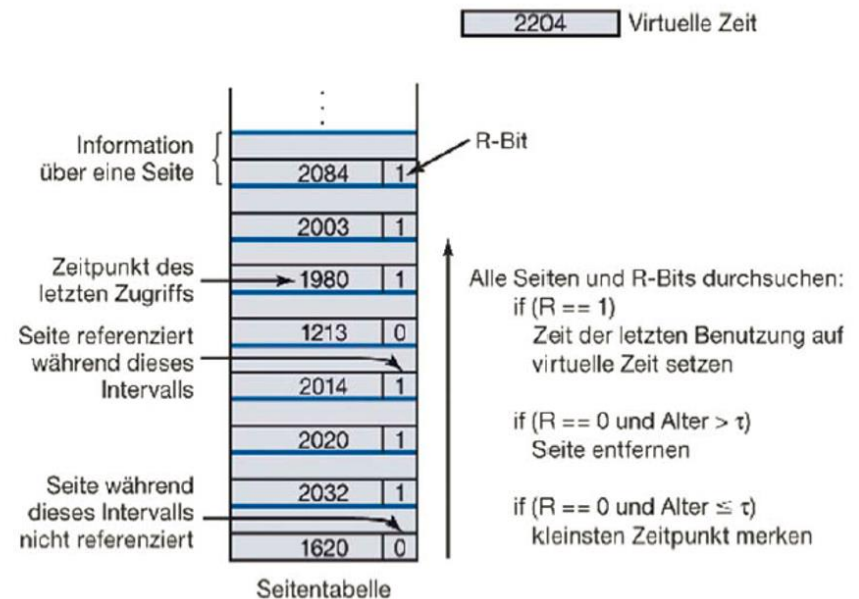
Seitenersetzungsalgorithmus

- Working Set Modelle
 - Korrekte Bestimmung des Arbeitsbereichs
 - Ansatz:
 - Lege k im Voraus fest
 - Schieberegister mit k Einträgen
 - Bei jedem Speicherzugriff wird die aktuelle Seitennummer von rechts hinzugefügt
 - Die Menge der k Seitennummern in dem Register wären dann der Arbeitsbereich.
 - Bei einem Seitenfehler könnte das Schieberegister analysiert werden
 - Sortieren, doppelte Einträge entfernen
 - Ergebnis wäre Arbeitsbereich
 - Wechsel des Arbeitsbereichs durchführen
 - Leider viel zu aufwändig das Schieberegister aktuell zu halten und beim Wechsel zu analysieren!

→ Bessere (Näherungs-)Methoden sind notwendig

Seitenersetzungsalgorithmus

- Working Set Algorithmus – Auslagerung einer Seite, die nicht zum Arbeitsbereich gehört
 - Es liegen nur Seiten im Speicher, die für die Auslagerung in Frage kommen. Der Algorithmus übergeht die ausgelagerten Seiten.
 - Jeder Eintrag enthält mind. 2 Informationen:
 - die ungefähre Zeit des letzten Zugriffs auf die Seite und
 - das R-Bit.
 - Ein leerer Bereich steht für die anderen Felder, die dieser Algorithmus nicht braucht (z.B. Seitenrahmennummer, M-Bit, P-Bit)
 - Die R-Bits werden softwaremäßig gelöscht (Timer-Interrupt)



Seitenersetzungsalgorithmus

• Working Set Algorithmus

- Voraussetzung: Hardware setzt R- und M-Bit
- Annahme: periodischer Timer-Interrupt löscht die R-Bits softwaremäßig
- Bei jedem Seitenfehler wird die Seitentabelle nach einer Seite durchsucht, die ausgelagert werden kann.

- R-Bit wird untersucht

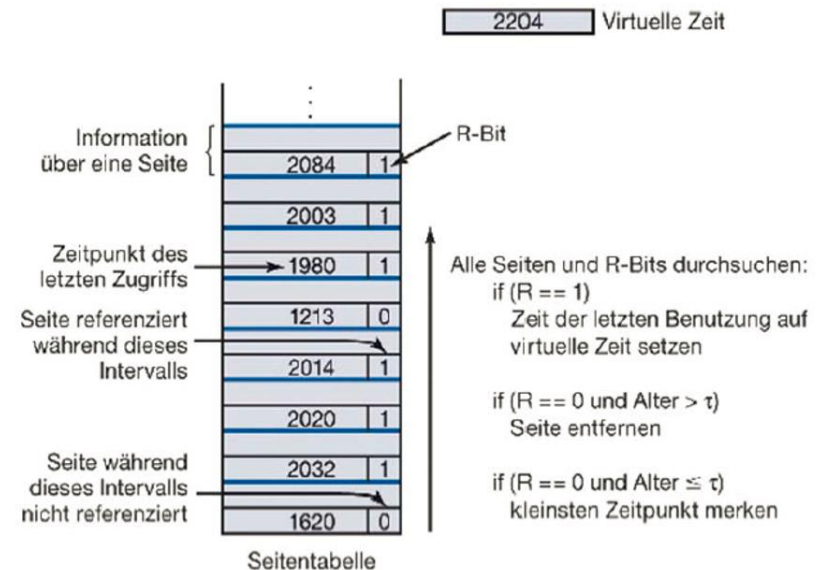
- Gesetz: virtuelle Zeit wird in das Feld für den Zeitpunkt des letzten Zugriff eingetragen
→ Seite wurde zum Zeitpunkt des Seitenfehlers benutzt
→ Arbeitsbereich

- R=0: auf Seite wurde seit dem letzten Timer-Interrupt nicht zugegriffen

→ Kandidat zur Auslagerung

Um zu entscheiden, ob die Seite zum Arbeitsbereich gehört, wird ihr Alter (virtuelle Zeit minus Zeitpunkt des letzten Zugriffs) berechnet und mit t verglichen.

- R=0, Alter ist gleich oder geringer als t , dann gehört die Seite zum Arbeitsbereich und wird nicht ausgelagert



Seitenersetzungsalgorithmus

- WS-Clock (Carr und Henessey, 1981)
 - Verbindung von Working Set und Clock
 - In realen Systemen weit verbreitet
 - Benutzung einer ringförmigen Liste von Seitenrahmen
 - Anfangs ist die Liste leer
 - Erste Seite wird in die Liste eingefügt
 - Mit der Zeit kommen immer mehr Seiten hinzu; Liste wird zu einem Ring
 - Jeder Listeneintrag enthält ein R-Bit, M-Bit und ein Feld für den Zeitpunkt des letzten Zugriffs

- WS-Clock = Verbindung von Clock und Working Set

- WS-Clock = Verbindung von Clock und Working Set

Figure 1 illustrates the evolution of a page table across four stages (a, b, c, d). Each stage shows a set of pages (numbered 2033, 2084, 1620, 2032, 1980, 2020, 2014, 1213) and their corresponding physical addresses. Arrows indicate the mapping of virtual pages to physical frames.

- (a) Initial state: Pages 2033, 2084, 1620, 2032, 1980, 2020, 2014, and 1213 are mapped to physical addresses. Page 2014 is highlighted with a 'Time of last use' and 'R bit'.
- (b) State after a page fault: Page 1213 is loaded into a physical frame, and page 2014 is mapped to a new physical address.
- (c) State after a page fault: Page 1213 is loaded into a physical frame, and page 2014 is mapped to a new physical address.
- (d) State after a page fault: Page 2204 is loaded into a physical frame, and page 2014 is mapped to a new physical address. The new page is highlighted with a 'New page' label.

- a) Bei Seitenfehler wird Seite untersucht, auf die der Uhrzeiger zeigt.
R-Bit=1, wurde die Seite im letzten Intervall benutzt, kein idealer Kandidat für Auslagerung,
- b) R-Bit=0 gesetzt, Zeiger rückt vor
- c) Wenn das Alter der Seite größer ist als t und die Seite nicht verändert wurde, gehört sie nicht zum Arbeitsbereich und es existiert eine gültige Kopie auf der Platte
- d) Seite wird gelöscht und durch die neue Seite ersetzt.

Seitenersetzungsalgorithmus – Zusammenfassung 1

- Aufgabe Seitenersetzung bedeutet Bestimmung der Seite, die verdrängt wird
 - Optimale Strategie?
 - Seite, die in Zukunft am längsten nicht gebraucht wird
 - NRU: vier Klassen gemäß R- und M-Bit
 - FIFO: Seite, die am längsten im Hauptspeicher ist
 - Second Chance: älteste Seite die seit letzten Seitenwechsel nicht benutzt wurde
 - Clock-Algorithmus: effizient Implementierung
 - LRU: Seite die am längsten nicht benutzt wurde
 - Aging: effiziente Implementierung
 - Working Set: verwendet virtuelle Zeit und Prepaging
 - WS-Clock: effiziente verbreitete Implementierung

Seitenersetzungsalgorithmus – Zusammenfassung 2

Algorithmus	Kommentar
Optimal	Nicht realisierbar, aber nützlich als Maßstab
NRU (Not Recently Used)	Sehr grobe Annäherung an LRU
FIFO (First In First Out)	Entfernt evtl. auch wichtige Seiten
Second Chance	Enorme Verbesserung gegenüber FIFO
Clock	Realistisch
LRU (Least Recently Used)	Exzellent, aber schwierig zu implementieren
NFU (Not Frequently Used)	Ziemlich grobe Annäherung an LRU
Aging	Effizienter Algorithmus, gute Annäherung an LRU
Working Set	Etwas aufwändig zu implementieren
WSClock	Guter und effizienter Algorithmus

Abbildung 3.22: Die behandelten Seitenersetzungsalgorithmen

Entwurfskriterien

- Welche Kriterien müssen beim Entwurf eines leistungsfähigen Paging-Systems neben den Seitenersetzungsalgorithmen noch berücksichtigt werden?
 - Aufteilung des Speichers zwischen konkurrierenden lauffähigen Prozessen
 - **Zuteilungsstrategie** – lokal versus global (bzw. darf die auszulagernde Seite einem anderen Prozess angehören?)
 - **Lokal** (nur die Seiten des Prozesses) oder
 - Fester Speicherbereich für jeden Prozess
 - Führt bei wachsendem Speicher zur Zunahme von Seitenfehlern
→ Trashing, Seitenflattern
 - **Global:** auch Seiten anderer Prozesse dürfen ausgelagert werden
 - Dynamische Speicherbereiche
 - Überwachung der Speicherbereich durch “PFF” (page fault frequency)
→ Entscheidet ob mehr oder wenig Speicher zugeteilt wird

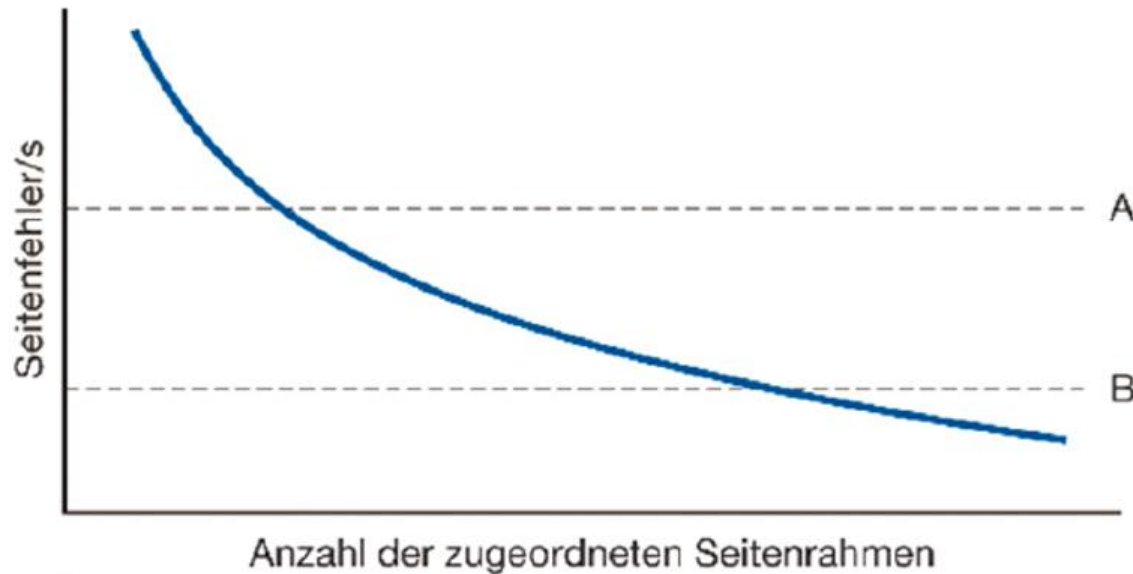
	Age		
A0	10	A0	A0
A1	7	A1	A1
A2	5	A2	A2
A3	4	A3	A3
A4	6	A4	A4
A5	3	A5	A5
B0	9	A6	B0
B1	4	B1	B1
B2	6	B2	B2
B3	2	B3	A6
B4	5	B4	B4
B5	6	B5	B5
B6	12	B6	B6
C1	3	C1	C1
C2	5	C2	C2
C3	6	C3	C3

(a) (b) (c)

(a) Lokal in A-Bereich
(c) global

Entwurfskriterien

Zuteilungsstrategie lokal versus global




PFF = Page fault frequency)

- PFF versucht die Speichergröße in einem bestimmten Bereich zu halten
 - » A minimaler Speicher (Anzahl Seitenrahmen)
 - » B maximaler Speicher (Anzahl Seitenrahmen)

Wahl der Zuteilungsstrategie hängt auch vom Ersetzungsalgorithmus ab

Entwurfskriterien

- Lastkontrolle
 - Meist ist zu wenig Speicher im System für N Prozesse
 - Damit wird der Speicherbereich für jeden Prozess kleiner
 - Erzeugt viele Seitenfehler
 - Reduzierung der Prozesse notwendig
 - Auslagern von Prozessen auf Festplatte
 - Aufteilung des Speichers zwischen den Prozessen, die zu viele Seitenfehler erzeugen
 - Swapping auch in Paging Systemen notwendig
-  Reduzierung der Seitenfehlerrate auf akzeptables Maß
- Einfluss auf Scheduling von Prozessen

Entwurfskriterien

- Seitengröße
 - Kann das Betriebssystem wählen
 - Auch wenn die Hardware für 4096-Byte-Seiten entworfen wurde, kann BS immer zwei aufeinanderfolgende 8192-Byte-Seitenrahmen zuordnen
 - Für kleine Seiten spricht
 - Interne Fragmentierung: für jedes Segment (Text, Daten, Stack) bleibt im Mittel die letzte Seite halb leer.
 - » → bei n Segmenten und einer Seitengröße p:
 $n * p / 2$ Speicher nicht belegt
 - Programme belegen in verschiedenen Phasen nur Teil des Adressraums
 - Für große Seiten spricht
 - Optimierte Festplattenzugriffe (nur ein Positionierungsschritt)
 - Kurze Seitentabelle

s: Durchschnittliche Prozessgröße
p: Seitengröße
e: Größe des Seitentabelleneintrag

$$\text{Verbrauch} = e * s/p + p/2$$

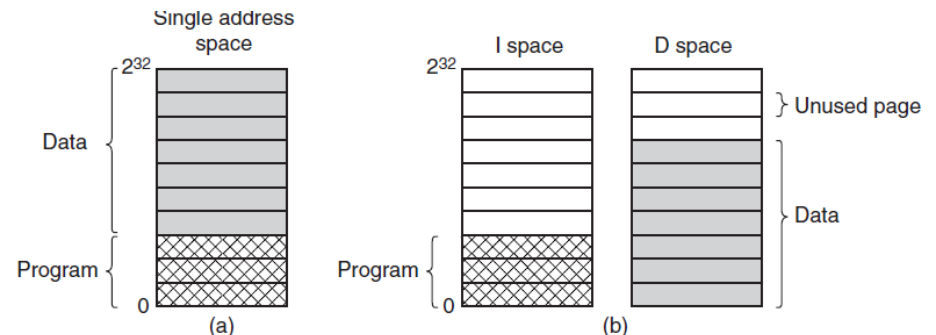
Optimum durch 1. Ableitung nach p

$$0 = -e * s / p^2 + 1/2$$

$$p = \text{SQRT}(2 * s * e)$$

Entwurfskriterien

- Die meisten Computer haben einen einzigen Adressraum, der sowohl die Programme als auch die Daten enthält.
- Trennung von Befehls- und Datenräumen
 - Mögliche Strategie bei kleinen Adressräumen
 - Befehlsraum (I-Space) für Befehle (Programmtext)
 - Datenraum (D-Space) für Daten
 - Getrennte Adressräume
 - Eigene Paging Mechanismen
- Gemeinsame Seiten
 - Mehrere Benutzer verwenden das gleiche Programm
 - Seiten, die Programmtext enthalten, können gemeinsam genutzt werden
 - Seiten mit Daten nicht



Entwurfskriterien

- Bereinigungsstrategien
 - Paging funktioniert am besten, wenn jederzeit ein reichliches Angebot an freien Seitenrahmen zur Verfügung steht
 - Um die Versorgung mit freien Seitenrahmen sicherzustellen wird ein regelmäßiges Bereinigen der Seitenrahmen durch Paging-Daemon durchgeführt
 - Wenn zu wenige freie Seitenrahmen vorhanden sind, wählt der Paging Daemon auf Basis des Seitenersetzungsalgorithmus Seiten aus, die ausgelagert werden
 - Speichern von veränderten Seiten auf Festplatte, ohne die Seite aus Speicher zu entfernen
 - Hält die freien Seitenrahmen sauber und vermindert Festplatten-Zugriffe

Implementierungsaspekte

- Das Betriebssystem interagiert mit dem Paging bei vier wichtigen Ereignissen:
 1. Bei der Erzeugung von Prozessen
 2. Bei der Ausführung von Prozessen
 3. Bei einem Seitenfehler
 4. Bei der Terminierung des Prozesses

Implementierungsaspekte

Das Betriebssystem interagiert mit dem Paging bei

1. Erzeugung von Prozessen

- Feststellen der anfänglichen Größe des Programmcodes und der Daten und Erzeugung einer Seitentabelle
- Tabelle bekommt Speicher zugeteilt und initialisiert
- Reservierung von Platz auf Festplatte für ausgelagerte Seiten
- Initialisierung des Swap-Bereichs mit dem Programmtext und den Daten
- Eintrag der Informationen über die Seitentabelle und Swap-Bereich in Prozesstabelle.

2. Ausführung von Prozessen

- Auswählen eines Prozesses durch Scheduler
- MMU muss auf neuen Prozess umschalten und der TLB muss geleert werden (Beseitigung aller Spuren des vorherigen Prozesse)
- Aktuelle Seitentabelle auf Seitentabelle des Prozesses setzen
- evtl. Laden einiger Seiten um Seitenfehlerrate zu senken

Implementierung

Das Betriebssystem interagiert mit dem Paging bei

3. Auftreten eines Seitenfehlers (page fault)

- Auslesen von Hardwareregistern um festzustellen, welche virtuelle Adresse den Fehler erzeugt hat
- Herleitung welche Seite benötigt wird und diese auf Festplatte finden
- Neuen Seitenrahmen für die Seite suchen
 - Wenn nötig alte Seite auslagern
- Befehlszähler auf den verursachenden Befehl zurücksetzen, damit er noch mal ausgeführt werden kann

4. Beenden des Prozesses

- Freigabe seiner Seitentabelle, Seitenrahmen und Plattenplatz für ausgelagerte Seiten
 - Evtl. gemeinsame genutzte Seiten erst später löschen

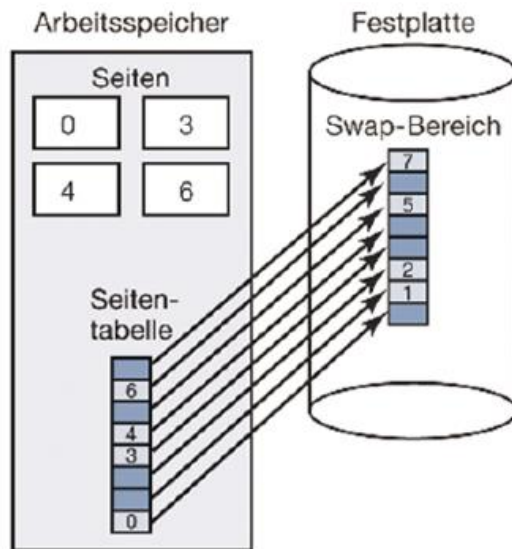
- **Implementierung**

- Wo werden die ausgelagerten Seiten hingeschrieben?
 - Swap-Partition (speziellen Bereich auf der Festplatte)
 - z.B. bei Linux ein spezielles Dateisystem
 - › Sehr einfach durch Verwenden von Blocknummern
 - z.B. bei Windows eine große vorreservierte Datei im Dateisystem
 - Bei Systemstart ist der Swap-Bereich leer
 - Jeder gestartete Prozess kopiert seinen Inhalt auf die Swap-Partition/den Swap-Bereich
 - › Die Position und Größe des Swap-Bereichs wird im PCB in der Prozesstabelle abgelegt
 - Wird ein Prozess gestoppt, dann wird der Bereich wieder freigegeben

- **Implementierung**

- Wie wird die Swap Partition initialisiert?
 - Zwei einfache Methoden
 - Alle Seiten zuerst in Swap-Partition initialisieren und bei Bedarf in den Speicher einzulagern
 - Alle Seiten im Speicher initialisieren und bei Bedarf in den Swap-Bereich auszulagern
- Wie ist der Swap-Bereich organisiert?
 - Feste Zuordnung oder dynamische Zuordnung
- Wie kann mit wachsenden Prozessen umgegangen werden?
 - Plattenzuordnungstabelle

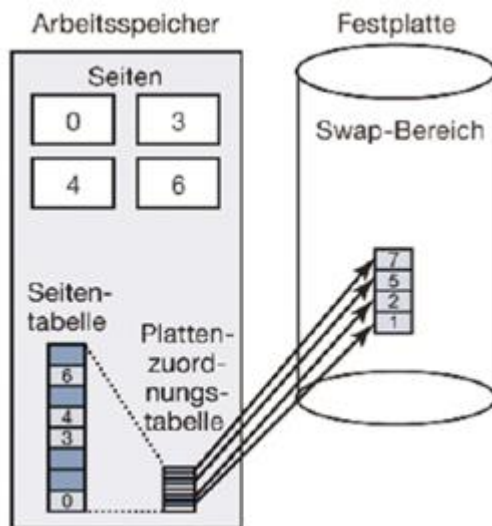
- **Implementierung**
 - Swap-Bereich



Erklärung:

- Seitentabelle mit acht Seiten
- Seiten 0, 3, 4 und 6 liegen im Arbeitsspeicher,
- Seiten 1, 2, 5, 7 auf der Platte
- Swap-Bereich ist genauso groß wie der virtuelle Adressraum (acht Seiten)
- **Jede Seite hat eine feste Adresse auf der Platte, an die sie geschrieben wird, falls sie ausgelagert wird.**
- Für die Berechnung dieser Adresse muss man nur die Anfangsadresse des Swap-Bereichs kennen (Seiten sind geordnet nach ihren Seitennummern gespeichert)
- Eine Seite im Speicher hat immer einen Schattenkopie auf der Platte (muss nicht aktuell sein)
- Dunkel dargestellten Seiten im Speicher zeigen an, dass sich diese Seiten nicht im Speicher befinden.
- Dunkel gezeichneten Seiten auf der Platte sind von den Kopien im Speicher verdrängt worden.
- Falls eine Speicherseite zurück auf die Platte geschrieben wird, die nicht verändert wurde, dann benutzt man die dunkler dargestellte Plattenkopie benutzt.

- Implementierung
 - Swap-Bereich



Erklärung:

- Seitentabelle mit acht Seiten
- Seiten 0, 3, 4 und 6 liegen im Arbeitsspeicher,
- Seiten 1, 2, 5, 7 auf der Platte
- **Seiten haben keine festen Adressen auf der Festplatte.**
- Bei Auslagerung einer Seite, sucht Betriebssystem eine leere Seite auf der Platte
- Eintrag der Adresse in der Plattenzuordnungstabelle, die virtuelle Seiten auf Festplattenadressen abbildet
- Eine Seite im Speicher hat keine Festplattenkopie
- Ihre Einträge in der Tabelle für Plattenadressen sind ungültig oder als unbenutzt markiert.

- Zusammenfassung
 - Betriebssystem verwaltet die Ressource Speicher
 - Swapping
 - Möglichkeit der Multiprogrammierung
 - Relokation von Speicherbereichen
 - Basis-/Limit-Register für dynamische Relokation
 - Speicherverwaltung mit verketteten Listen
 - Verschiedene Algorithmen: First Fit, NextFit, BestFit, WorstFit
 - Virtueller Speicher
 - Programme wachsen schneller als Hauptspeicher
 - Ausführung von Programmen, die nicht in Speicher passen
 - Paging
 - Aufteilung des Adressraums in kleine Einheiten (Seiten)
 - Trennung logische (virtuelle) und physische Adressen
 - Hardwareunterstützung durch MMU

- Zusammenfassung
 - Virtueller Speicher ...
 - Verwaltung des Speicher durch Seitentabellen
 - Verschiedene Algorithmen zur Seitenersetzung
 - Optimaler Algorithmus, NRU, FIFO, Second-Chance, Clock, LRU, NFU, Aging
 - Lokalitätseigenschaft von Prozessen (prepaging)
 - Arbeitsbereich (Working Set)
 - Algorithmen: WorkingSet, WS-Clock
 - Entwurfskriterien für Paging
 - Implementierungsaspekte beim Paging