Betriebssysteme I

WS 2019/2020

Roland Wismüller
Betriebssysteme / verteilte Systeme
roland.wismueller@uni-siegen.de
Tel.: 0271/740-4050, Büro: H-B 8404

Stand: 23. Januar 2020

Betriebssysteme I

WS 2019/2020

6 Speicherverwaltung

6 Speicherverwaltung ...

Inhalt:

- Einführung, Grundlagen
- *Swapping* und dynamische Speicherverwaltung
- Seitenbasierte Speicherverwaltung (*Paging*)
  - Adreßumsetzung
  - Dynamische Seitenersetzung
  - Seitenersetzungsalgorithmen

- Tanenbaum 4
- Stallings 7, 8
- Nehmer/Sturm 4

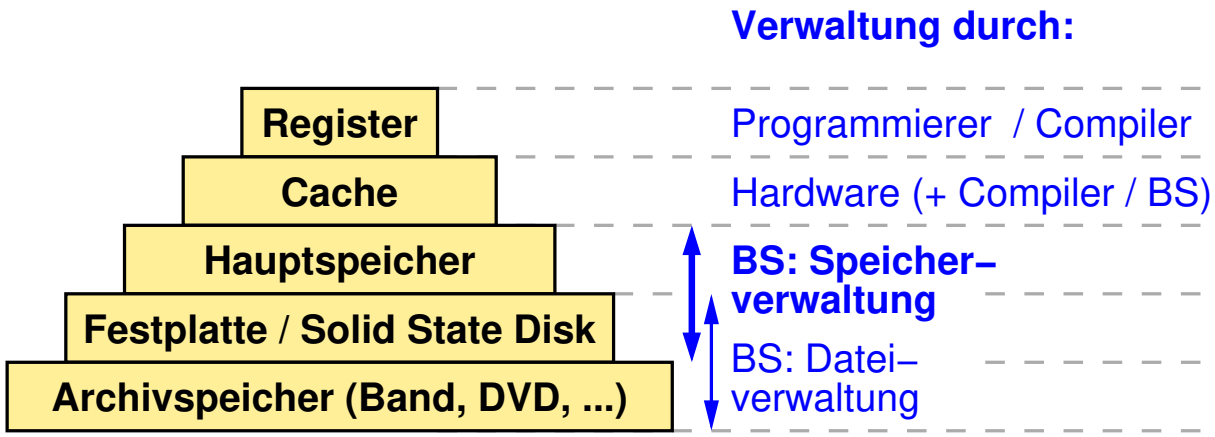
6.1 Einführung, Grundlagen

Wichtige Unterscheidung / Begriffe

- **Adreßraum**: Menge aller verwendbaren Adressen
  - **logischer (virtueller) Adreßraum**: aus Sicht eines Programms bzw. Prozesses gesehen
    - Adressen, die der Prozess* verwenden kann
  - **physischer Adreßraum**: aus Sicht der Hardware
    - Adressen, die die Speicher-Hardware verwendet / verwenden kann
  - nicht immer identisch! (siehe später: *Paging*)
- **Speicher**: das Stück Hardware, das Daten speichert ...
  - (oft: Speicher als Synonym für physischer Adreßraum)

* genauer: die Threads des Prozesses

Erinnerung: Speicherhierarchie

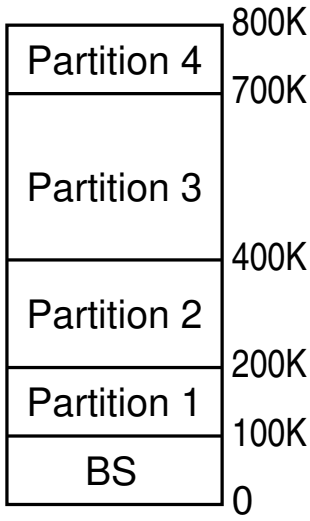


Aufgaben der Speicherverwaltung im BS

- Finden und Zuteilung freier Speicherbereiche
  - z.B. bei Erzeugung eines neuen Prozesses
  - oder bei Speicheranforderung durch existierenden Prozess
- Effiziente Nutzung des Speichers: verschiedene Aspekte
  - gesamter freier Speicher sollte für Prozesse nutzbar sein
  - Prozeß benötigt nicht immer alle Teile seines Adreßraums
  - Adreßräume aller Prozesse evtl. größer als verfügbarer Hauptspeicher
- **Speicherschutz**
  - Threads eines Prozesses sollten nur auf Daten dieses Prozesses zugreifen können

Mehrprogrammbetrieb mit festen Partitionen

- Einfachste Speicherverwaltung
- Hauptspeicher in Partitionen fester Größe eingeteilt
  - Festlegung bei Systemstart
  - Anzahl legt Multiprogramming-Grad fest
    - damit auch erreichbare CPU-Auslastung
  - unterschiedlich grosse Partitionen möglich
  - eine Partition für BS



- Bei Ankunft eines Auftrags (d.h. Start eines Programms)
  - Einfügen in Warteschlange für Speicherzuteilung
    - pro Partition oder gemeinsam

Anmerkungen zu Folie 286:

Getrennte Warteschlangen haben dabei das Problem, daß z.B. ein Job (bzw. Prozess) in der Warteschlange für eine kleine Partition warten kann, obwohl eine größere Partition inzwischen frei ist, da die Warteschlange beim Start des Jobs (bzw. Prozesses) festgelegt wird.

Auf der anderen Seite kann eine gemeinsame Warteschlange zu Speicherverschwendung führen, wenn kleine Jobs in (zu) großen Partitionen laufen.

Relokation und Speicherschutz

- Hintergrund: Programmcode enthält Adressen
  - z.B. in Unterprogrammaufrufen oder Ladebefehlen
- Ohne Hardware-Unterstützung:
  - beim Laden des Programms in eine Partition:
    - Adressen im Code müssen durch das BS angepaßt werden (**Relokation**)
    - möglich durch Liste in Programmdatei, die angibt, wo sich Adressen im Code befinden
  - durch Relokation beim Laden kann kein Speicherschutz erreicht werden
    - Threads des Prozesses können auf Partitionen anderer Prozesse zugreifen

Relokation und Speicherschutz ...

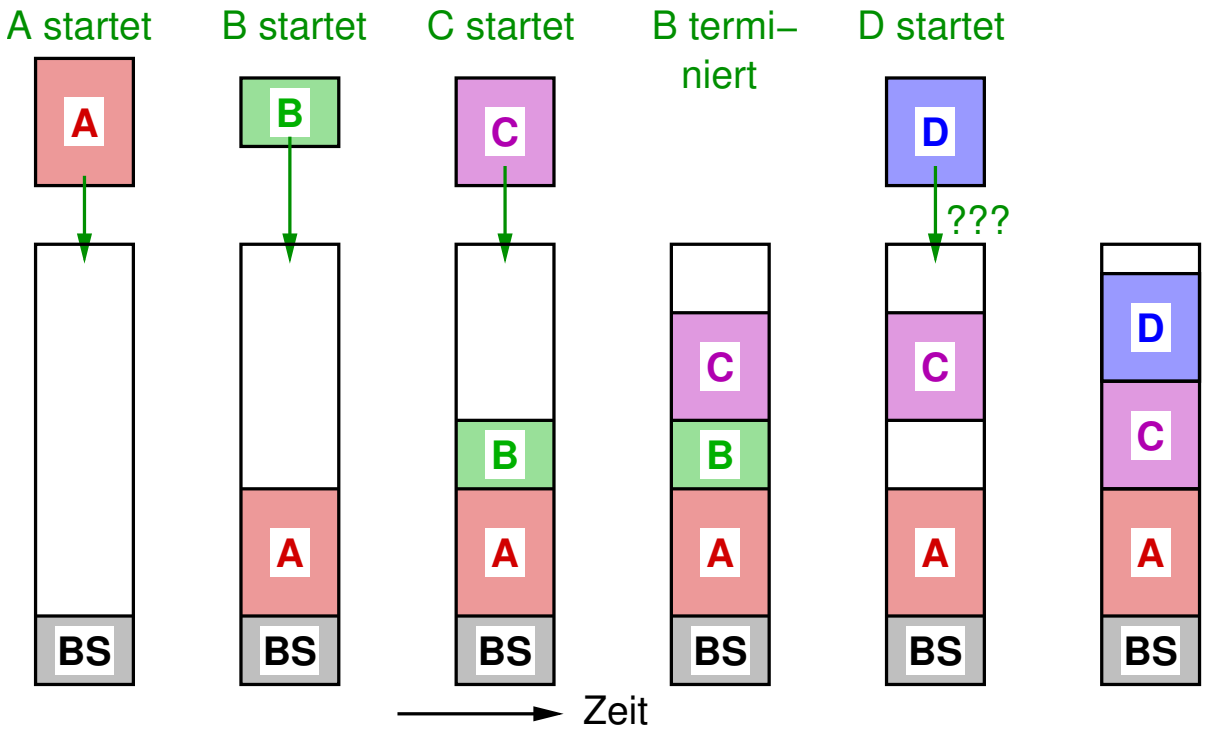
- Einfachste Unterstützung durch Hardware:
  - zwei spezielle, privilegierte Prozessorregister:
    - **Basisregister**: Anfangsadresse der Partition
    - **Grenzregister**: Länge der Partition
    - werden vom BS beim Prozeßwechsel mit Werten des aktuellen Prozesses geladen
  - bei jedem Speicherzugriff (in Hardware):
    - vergleiche (logische) Adresse mit Grenzregister
      - falls größer/gleich: Ausnahme auslösen
    - addiere Basisregister zur (logischen) Adresse
      - Ergebnis: (physische) Speicheradresse

- Feste Partitionen nur für Stapelverarbeitung geeignet
- Bei interaktiven Systemen:
  - alle aktiven Programme zusammen benötigen evtl. mehr Partitionen bzw. Speicher als vorhanden
- Lösung:
  - dynamische Partitionen
  - Partitionen ggf. temporär auf Festplatte auslagern
- Einfachste Variante: ***Swapping***
  - kompletter Prozeßadreßraum wird ausgelagert und Prozeß suspendiert
  - Wieder-Einlagern evtl. an anderer Stelle im Speicher
    - Relokation i.d.R. über Basisregister
- (Bessere Variante: ***Paging***, ☞ **6.3**)

6.2.1 Swapping

(Animierte Folie)

Beispiel



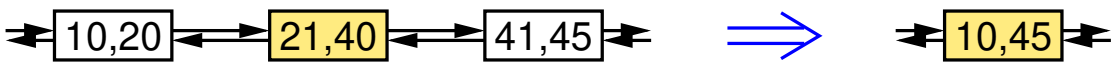
Diskussion

- Unterschied zu festen Partitionen:
  - Zahl, Größe und Ort der Partitionen variabel
- Löcher im Speicher können im Prinzip durch Verschieben zusammengefasst werden
- Probleme:
  - Ein- und Auslagern ist sehr zeitaufwendig
  - Prozeß kann zur Laufzeit mehr Speicher anfordern
    - benötigt evtl. Verschiebung oder Auslagerung von Prozessen
- Schlechte Speichernutzung: Prozeß benötigt i.d.R. nicht immer seinen ganzen Adreßraum

6.2.2 Dynamische Speicherverwaltung

(Animierte Folie)

- BS muß bei Prozeßerzeugung oder Einlagerung einen passenden Speicherbereich finden
- Dazu: BS benötigt Information über freie Speicherbereiche
- Typisch: Liste aller freien Speicherbereiche
  - Listenelemente jeweils im freien Bereich gespeichert
- Wichtig bei Speicherfreigabe: Verschmelzen der Einträge für aneinander grenzende freie Speicherbereiche
  - dazu: doppelt verkettete Liste, sortiert nach Adressen



- Anmerkung: Problem/Lösungen identisch zur Verwaltung des Heaps durch das Laufzeitsystem

Suchverfahren

- Ziel: finde einen passenden Eintrag in der Freispeicherliste
  - benötigter Anteil wird an Prozeß zugewiesen
  - Rest wird wieder in Freispeicherliste eingetragen
    - führt zu (externer) Fragmentierung des Speichers
- ***First Fit***: verwende ersten passenden Speicherbereich
  - einfach, relativ gut
- ***Quick Fit***: verschiedene Listen für Freibereiche mit gebräuchlichen Größen
  - schnelle Suche
  - Problem: Verschmelzen freier Speicherbereiche
  - Variante **Buddy-System**: Blockgrößen sind Zweierpotenzen
    - einfaches Verschmelzen, aber interne Fragmentierung

Betriebssysteme I

WS 2019/2020

09.01.2020

Roland Wismüller
Betriebssysteme / verteilte Systeme
roland.wismueller@uni-siegen.de
Tel.: 0271/740-4050, Büro: H-B 8404

Stand: 23. Januar 2020

Zur Klausur

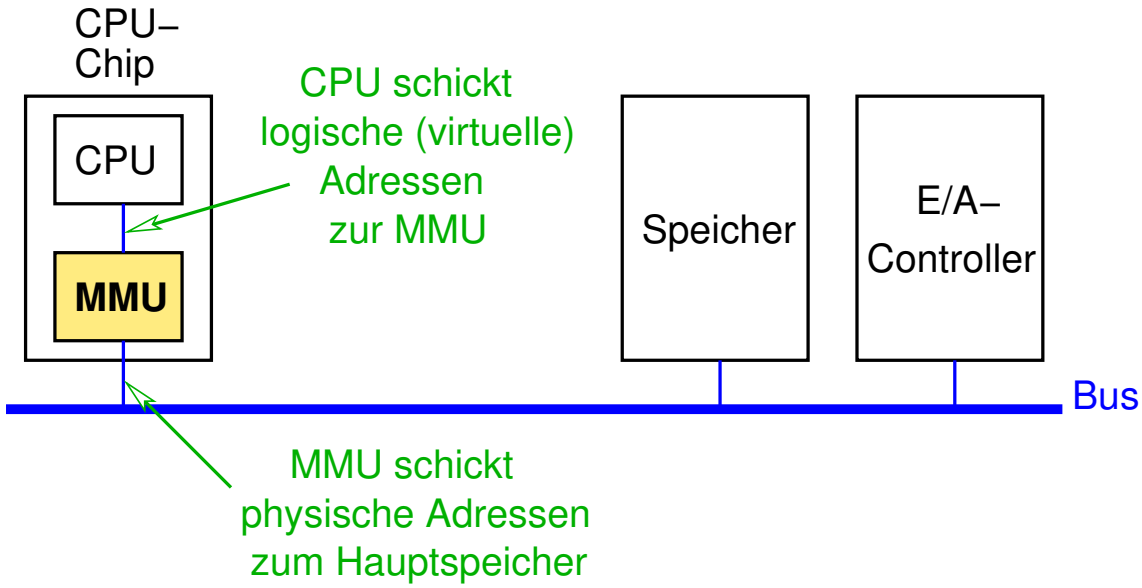
- Termine:
  - Montag, **17.02.2020**, 12:00 Uhr s.t., PB-C 101
  - nächster Termin nach dem SoSe 2020
- Dauer: 60 Minuten, ohne Hilfsmittel

- **Anmeldefrist: 22.01.2020!**

6.3 Seitenbasierte Speicherverwaltung (*Paging*)

- Grundlage: strikte Trennung zwischen
  - **logischen Adressen**, die der Prozeß sieht / benutzt
    - auch **virtuelle Adresse** genannt
  - **physischen Adressen**, die der Hauptspeicher sieht
- Idee: **bei jedem Speicherzugriff** wird die vom Prozeß erzeugte logische Adresse auf eine physische Adresse abgebildet
  - durch Hardware: **MMU** (***Memory Management Unit***)
- Vorteile:
  - kein Verschieben beim Laden eines Prozesses erforderlich
  - auch kleine freie Speicherbereiche sind nutzbar
  - keine aufwendige Suche nach passenden Speicherbereichen
  - Speicherschutz ergibt sich (fast) automatisch
  - ermöglicht, auch Teile des Prozeßadreßraums auszulagern

Ort und Funktion der MMU im Rechner



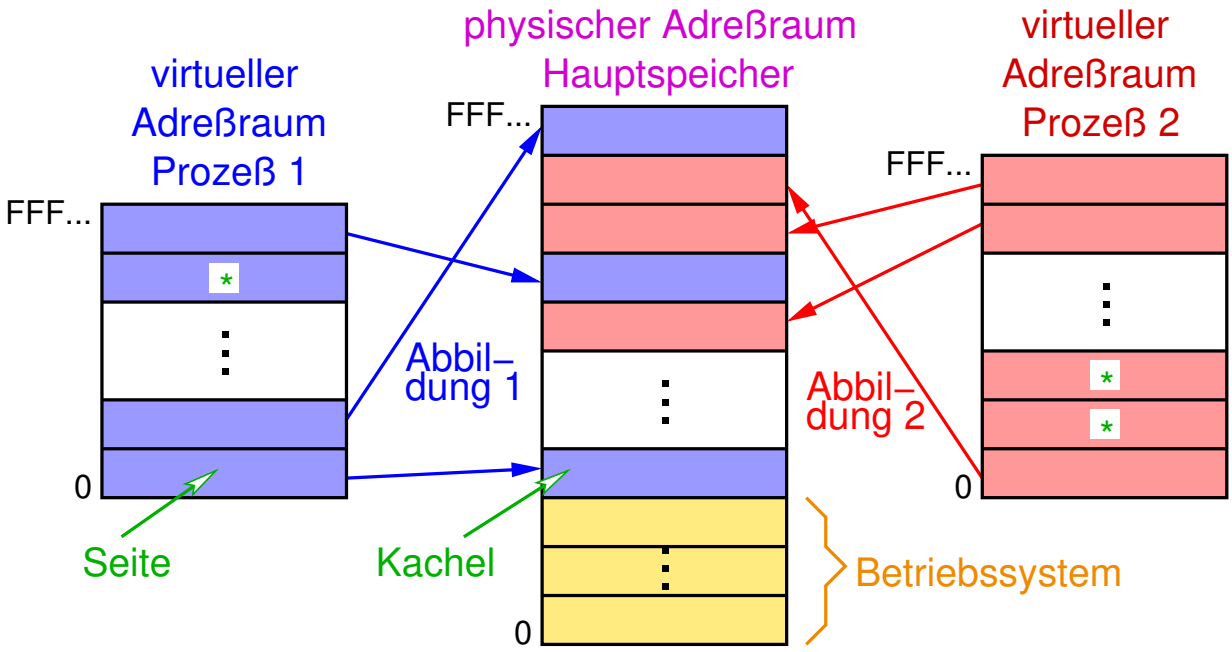
6.3.1 Adreßumsetzung

Seitenbasierte Speicherabbildung

- Abbildung von virtuellen auf physische Adressen über Tabelle
  - Tabelle wird vom BS erstellt und aktualisiert
- Dazu: Aufteilung der Adreßräume in Blöcke fester Größe
  - **Seite**: Block im virtuellen Adreßraum
  - **Kachel** (**Seitenrahmen**): Block im physischen Adreßraum
  - Typische Seitengröße: 4 KByte
- Umsetzungstabelle (**Seitentabelle**) definiert für jede Seite:
  - physische Adresse der zugehörigen Kachel (falls vorhanden)
  - Zugriffsrechte

(Animierte Folie)

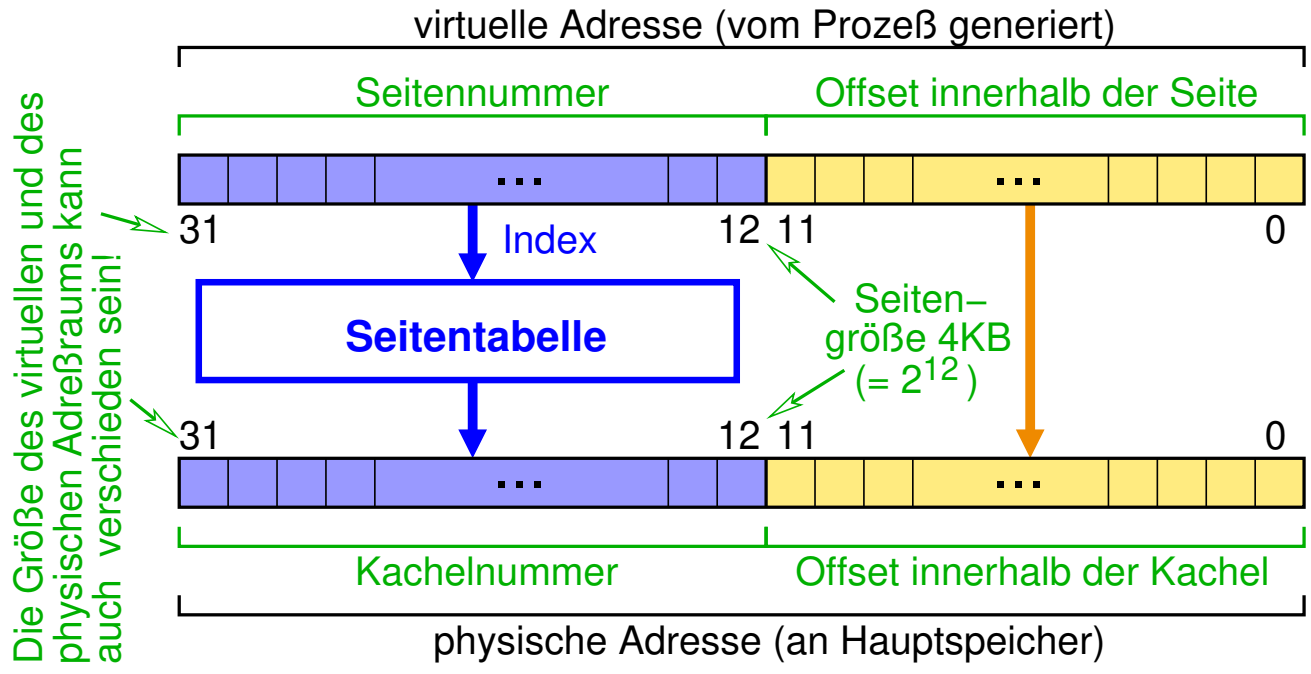
Grundidee der seitenbasierten Speicherabbildung



* Diese Seiten sind nicht in den Hauptspeicher abgebildet
Sie könnten z.B. auf Hintergrundspeicher ausgelagert sein.

(Animierte Folie)

Prinzip der Adreßumsetzung

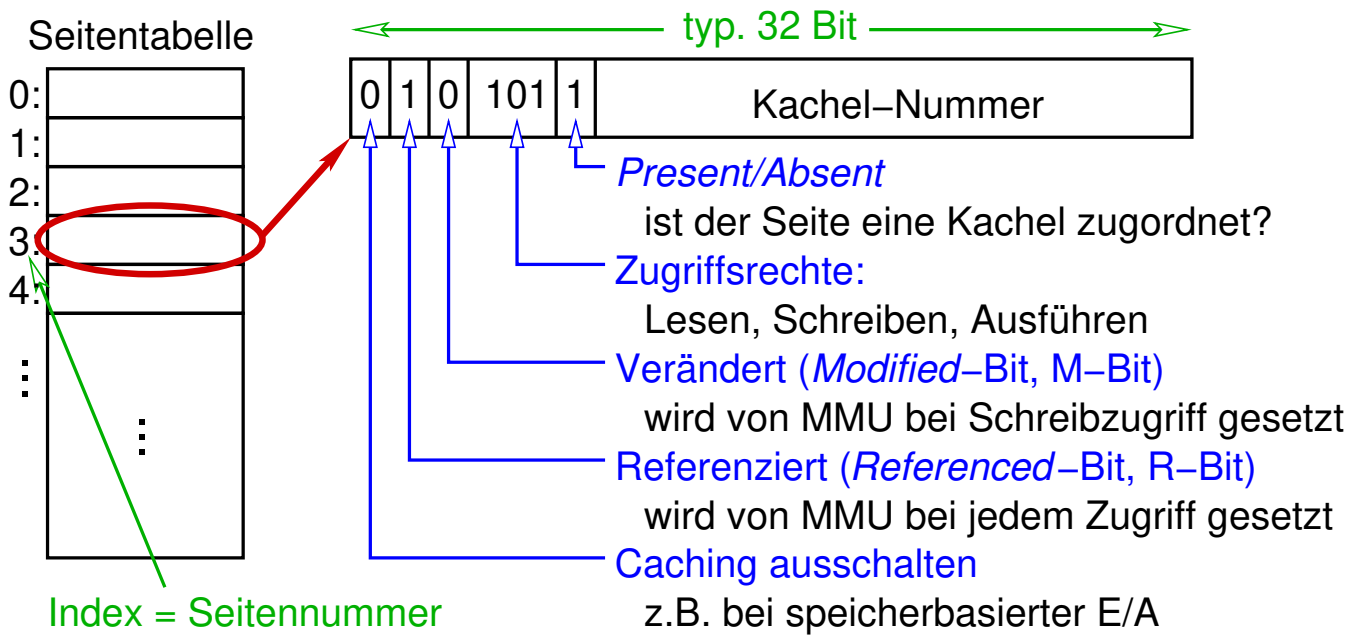


Anmerkungen zu Folie 298:

Bei den Intel Core Duo Prozessoren haben z.B. die virtuellen Adressen eine Länge von 64 Bit, die physischen Adressen eine Länge von 36 Bit.

6.3.1 Adreßumsetzung ...

Typischer Aufbau der Seitentabelleneinträge



Adreßabbildung ist Zusammenspiel von BS und MMU

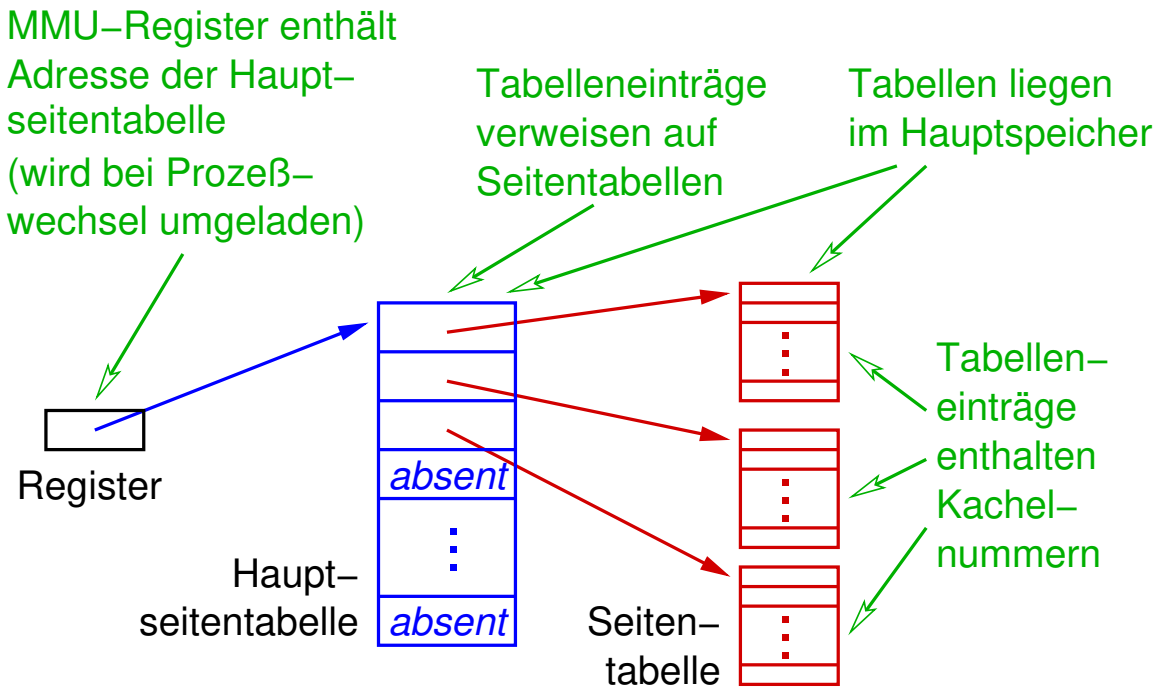
- BS setzt für jeden Prozeß eine Seitentabelle auf
  - Tabelle wird beim Prozeßwechsel ausgetauscht
- MMU realisiert die Adreßumsetzung
  - falls einer Seite keine Kachel zugeordnet ist (*Present*-Bit ist gelöscht):
    - MMU erzeugt Ausnahme (**Seitenfehler**)
  - Speicherschutz ist automatisch gegeben, da Seitentabelle nur auf Kacheln verweist, die dem Prozeß zugeteilt sind
  - zusätzlich: Zugriffsrechte für einzelne Seiten
    - z.B. Schreibschutz für Programmcode
    - bei Verletzung: Erzeugen einer Ausnahme

Mehrstufige Seitentabellen

- Jeder Seite muß eine Kachel zugeordnet werden können
- Problem: Speicherplatzbedarf der Tabelle
  - z.B. bei virtuellen Adressen mit 32 Bit Länge: 2^{20} Seiten
    - für jeden Prozeß wäre eine 4 MByte große Tabelle nötig
  - bei 64-Bit Prozessoren sogar 2^{52} Seiten, d.h. 32 PByte!
- Aber: Prozeß nutzt virtuellen Adreßraum i.a. nicht vollständig
  - daher mehrstufige Tabellen sinnvoll, z.B. für 2^{20} Seiten:
    - 1. Stufe: Hauptseitentabelle mit 1024 Einträgen
    - 2. Stufe: $\leq$ 1024 Seitentabellen mit je 1024 Einträgen (Tabellen nur vorhanden, wenn notwendig)
- (Alternative: **Invertierte Seitentabellen**)

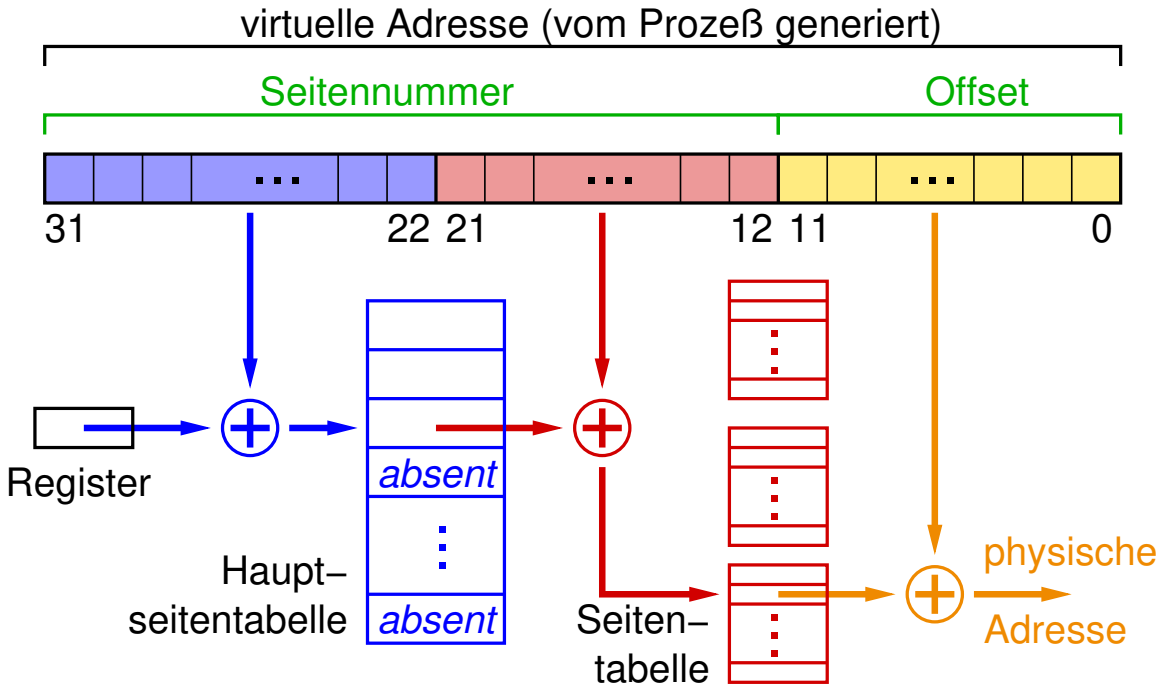
(Animierte Folie)

Adreßumsetzung mit zweistufiger Seitentabelle



(Animierte Folie)

Adreßumsetzung mit zweistufiger Seitentabelle

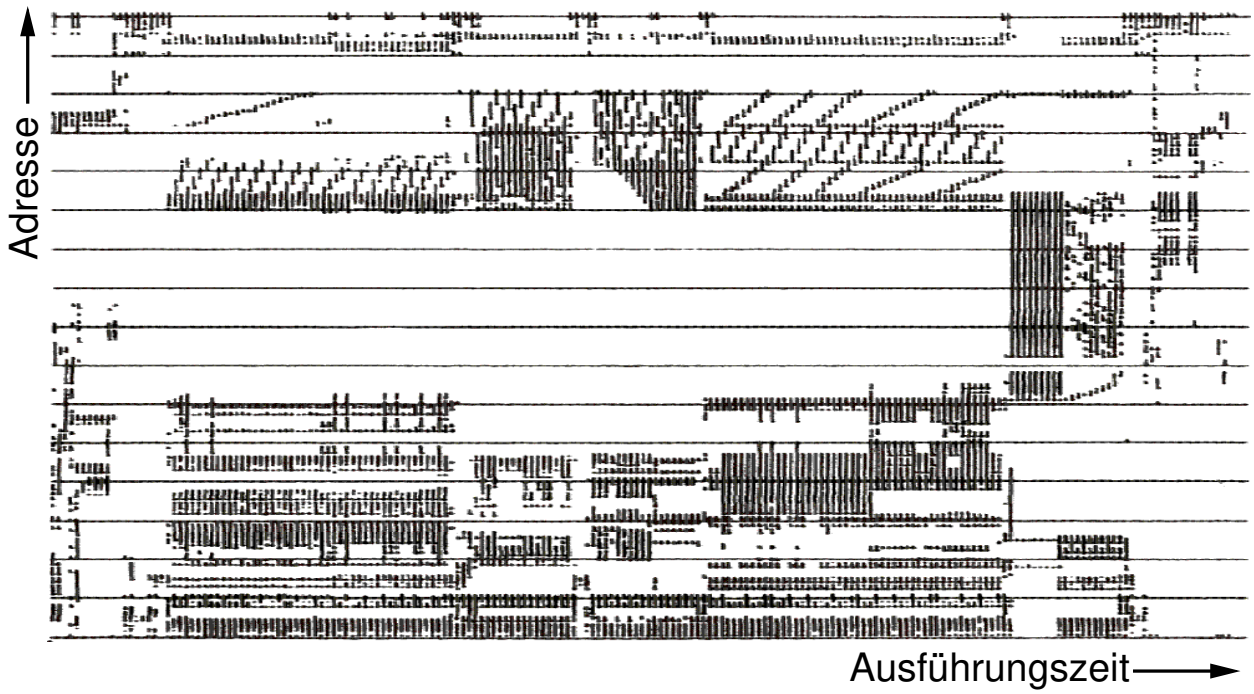


TLB: *Translation Lookaside Buffer*

- Seitentabellen liegen im Speicher
  - für jeden Speicherzugriff mehrere zusätzliche Zugriffe für Adreßumsetzung nötig!
- Optimierung: MMU hält kleinen Teil der Adreßabbildung in einem internen Cache-Speicher: **TLB**
- Verwaltung des TLB
  - durch Hardware (MMU): bei CISC-Prozessoren (z.B. x86)
  - durch Software (BS): bei einigen RISC-Prozessoren
    - MMU erzeugt Ausnahme, falls für eine Seite kein TLB-Eintrag vorliegt
    - BS behandelt Ausnahme: Durchsuchen der Seitentabellen und Ersetzen eines TLB-Eintrags

6.3.2 Dynamische Seitenersetzung

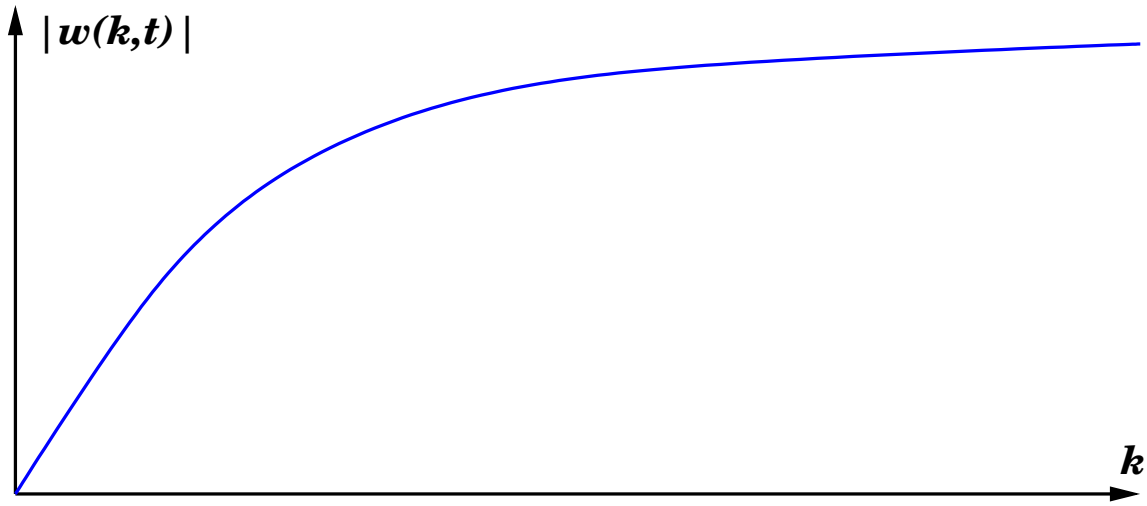
Motivation: Lokalitätsprinzip



Virtueller Speicher

- Idee: nur eine Teilmenge der Seiten eines Prozesses wird im Hauptspeicher gehalten: ***Resident Set***
  - alle anderen Seiten sind auf Festplatte ausgelagert
- Auswirkungen:
  - höherer Multiprogramming-Grad möglich
  - Prozeßadreßraum (logischer Adreßraum) kann größer als Hauptspeicher (physischer Adreßraum) sein
- ***Working Set*** eines Prozesses P: $w(k,t)$
  - zur Zeit t: Menge der Seiten, die P bei den letzten k Speicherzugriffen benutzt hat

Abhängigkeit des *Working Set* von k



- Für genügend großes k: $|w(k,t)|$ ist beinahe konstant
  - aber meist noch deutlich kleiner als Prozeßadreßraum

Working Set und Speicherzuteilung

- BS muß genügend Speicher bereitstellen, um für jeden Prozeß das *Working Set* $w(k,t)$ für genügend großes k im Hauptspeicher zu halten
- Falls nicht: ***Thrashing***, **Seitenflattern**
  - Prozesse benötigen ständig ausgelagerte Seiten
  - zum Einlagern muß aber andere Seite verdrängt werden
  - führt zu ständigem Ein- und Auslagern
  - Systemleistung sinkt dramatisch
- Mögliche Abhilfe: *Swapping*
  - verdränge einen Prozeß komplett aus dem Speicher

Strategieentscheidungen

- **Abrufstrategie**: wann werden Seiten eingelagert?
  - erst bei Bedarf, also bei Seitenfehler: ***Demand Paging***
  - im Voraus: ***Prepaging***
    - z.B. bei Programmstart
    - oder: lade Folgeseiten auf Platte gleich mit
  - (bei *Swapping* werden immer alle zuvor residenten Seiten wieder eingelagert)
- **Zuteilungsstrategie**: welche Kacheln werden einem Prozeß zugeteilt?
  - nur in Spezialfällen (Parallelrechner) relevant

Strategieentscheidungen ...

- **Austauschstrategie**
  - welche Seite wird verdrängt, wenn keine freie Kachel mehr vorhanden ist?
    - Seitenersetzungsalgorithmen (☞ **6.3.3**)
  - wo wird nach Verdrängungskandidaten gesucht?
    - **lokale Strategie**: verdränge nur Seiten des Prozesses, der neue Seite anfordert
      - Adreßraumgröße fest oder variabel
      - Größe sollte *Working Set* entsprechen
      - Einstellung z.B. aufgrund der Seitenfehlerfrequenz
    - **globale Strategie**: suche Verdrängungskandidaten unter den Seiten aller Prozesse

Ablauf eines Seitenwechsels (vereinfacht)

0. MMU hat Seitenfehler (Ausnahme) ausgelöst
1. BS ermittelt virtuelle Adresse (Seite S) u. Grund der Ausnahme
2. Falls Schutzverletzung vorlag: Prozeß abbrechen, Fertig.
3. Falls keine freie Kachel verfügbar:
   a) bestimme die zu verdrängende Seite S'
   b) falls S' modifiziert (*Modified*-Bit = 1): S' auf Platte schreiben
   c) Seitentabelleneintrag für S' aktualisieren (u.a. *Present*-Bit = 0)
4. Seite S von Platte in freie Kachel (ggf. die von S') laden
5. Seitentabelleneintrag für S aktualisieren (u.a. *Present*-Bit = 1)
6. unterbrochenen Thread fortsetzen
   - abgebrochener Befehl wird weitergeführt oder wiederholt

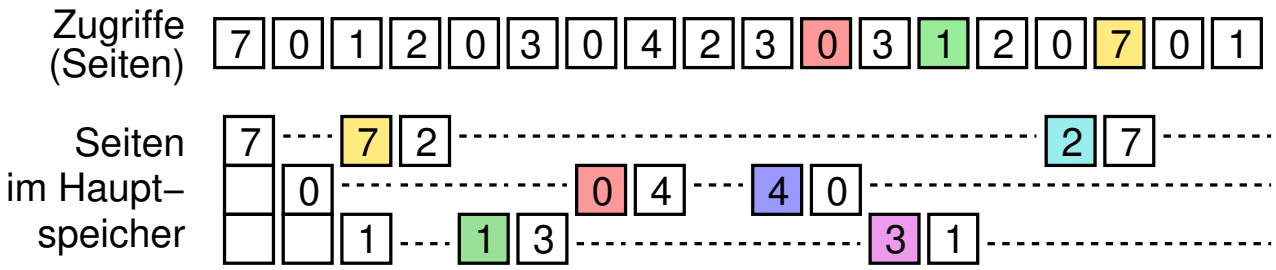
Zu den Kosten von Seitenwechseln

- Mittlere Speicherzugriffszeit bei Wahrscheinlichkeit p für Seitenfehler:
  - $t_Z = t_{HS} + p \cdot t_{SF}$
  - t_{HS}: Zugriffszeit des Hauptspeichers (ca. 10-100 ns)
  - t_{SF}: Zeit für Behandlung eines Seitenfehlers
    - dominiert durch Plattenzugriff (ca. 10 ms)
- Um Leistungsverlust im Rahmen zu halten:
  - p muß sehr klein sein
  - max. ein Seitenfehler bei mehreren Millionen Zugriffen

(Animierte Folie)

Optimale Strategie (nach Belady)

- Verdränge die Seite, die in Zukunft am längsten nicht mehr benötigt wird



- In der Praxis nicht realisierbar
- Als Referenzmodell zur Bewertung anderer Verfahren

Betriebssysteme I

WS 2019/2020

16.01.2020

Roland Wismüller
Betriebssysteme / verteilte Systeme
roland.wismueller@uni-siegen.de
Tel.: 0271/740-4050, Büro: H-B 8404

Stand: 23. Januar 2020

6.3.3 Seitenersetzungsalgorithmen ...

Not Recently Used (NRU)

- Basis: von der MMU gesetzte Statusbits in Seitentabelle:
  - **R**-Bit: Seite wurde referenziert
  - **M**-Bit: Seite wurde modifiziert
- **R**-Bit wird vom BS in regelmäßigem Abstand gelöscht
- Bei Verdrängung: vier Prioritätsklassen

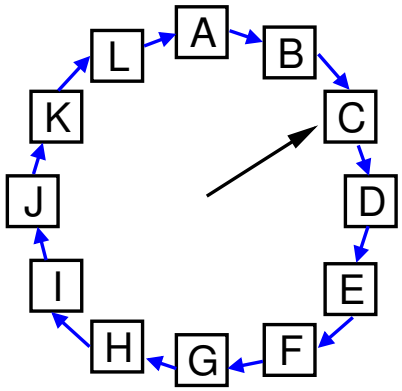
  Klasse 0: nicht referenziert, nicht modifiziert
  Klasse 1: nicht referenziert, modifiziert
  Klasse 2: referenziert, nicht modifiziert
  Klasse 3: referenziert, modifiziert
- Auswahl innerhalb der Klasse zufällig
- Nicht besonders gut, aber einfach

First In First Out (FIFO)

- Verdränge die Seite, die am längsten im Hauptspeicher ist
- Einfache, aber schlechte Strategie
  - Zugriffsverhalten (*Working Set*) wird ignoriert

Second Chance bzw. *Clock*-Algorithmus

- Erweiterung von FIFO
- Idee: verdränge älteste Seite, auf die seit dem letzten Seitenwechsel nicht zugegriffen wurde
- Seiten im Hauptspeicher werden nach Alter sortiert in Ringliste angeordnet, Zeiger zeigt auf älteste Seite



Bei Seitenfehler:

betrachte Seite, auf die der Zeiger zeigt:

R = 0: verdränge Seite, fertig

R = 1: lösche **R**-Bit,
setze Zeiger eins weiter,
wiederhole von vorne

Anmerkungen zu Folie 316:

Der Algorithmus terminiert auf jeden Fall: falls es zu Beginn keine Seite mit **R** = 0 gab, wird die älteste Seite verdrängt.

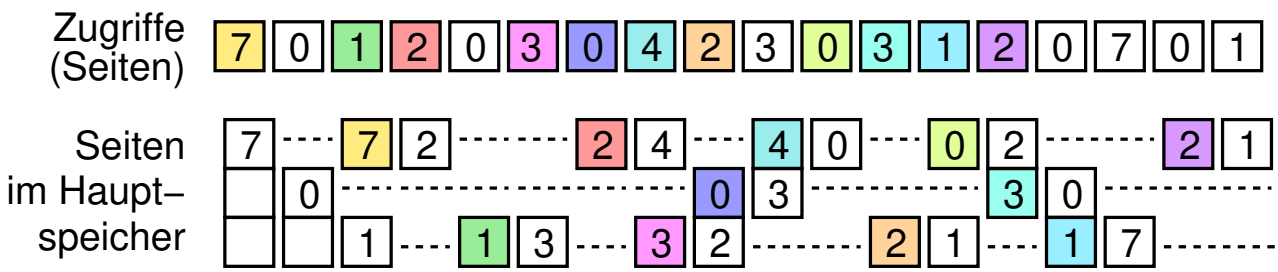
Die hier gezeigte Realisierung ist der *Clock*-Algorithmus, der eine effiziente Implementierung von *Second Chance* erlaubt. Bei Verwendung einer linearen Liste müssen die Seiten mit **R**=1 nach Löschen des **R**-Bits ans Ende der Liste verschoben werden. Diese Seiten werden jetzt also als neu betrachtet und bekommen dadurch ihre „zweite Chance“.

6.3.3 Seitenersetzungsalgorithmen ...

(Animierte Folie)

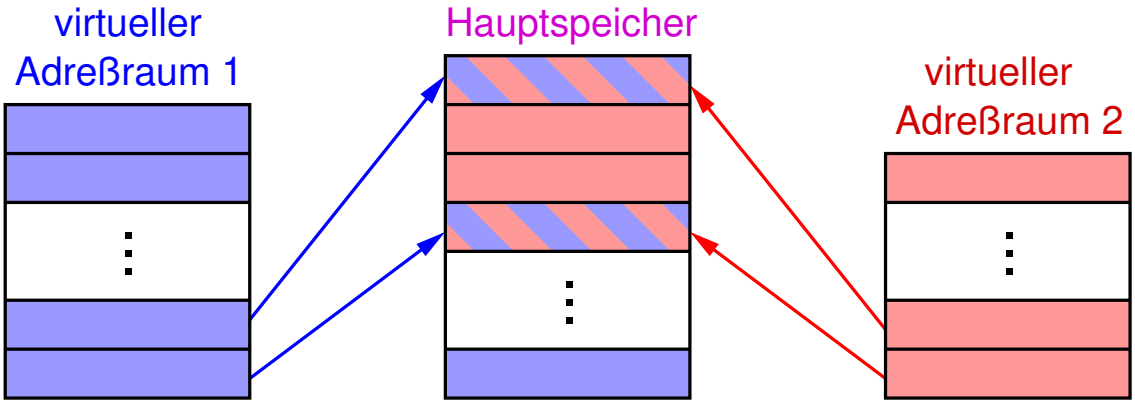
Least Recently Used (LRU)

- Verdränge die Seite, die am längsten nicht benutzt wurde
  - Vermutung: wird auch in Zukunft nicht mehr benutzt
- Nahezu optimal, wenn Lokalität gegeben ist
- Problem: (Software-)Implementierung
  - bei jedem Zugriff muß ein Zeitstempel aktualisiert werden
  - daher i.a. nur Näherungen (z.B. *Aging*)



6.3.4 Gemeinsamer Speicher zwischen Prozessen

- Adreßräume von Prozessen können teilweise überlappen



- Anwendung:
  - gemeinsame Speichersegmente zur Kommunikation
  - gemeinsam genutzte Bibliotheken (*Shared Library*, DLL)

6.4 Zusammenfassung / Wiederholung

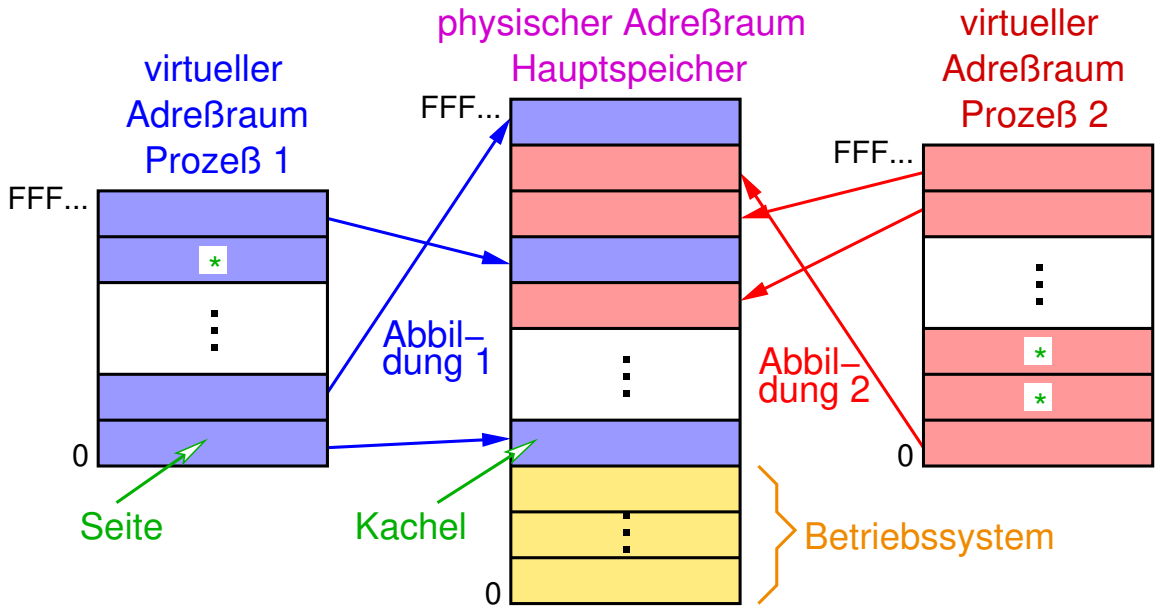
- Ziele der Speicherverwaltung:
  - effiziente Speicherzuweisung, Speicherschutz
- Wichtig: Unterscheidung logischer / physischer Adreßraum
- *Swapping*: Ein-/Auslagern kompletter Adreßräume
  - Suche nach freiem Speicher: *First Fit*, *Quick Fit*

6.4 Zusammenfassung / Wiederholung ...

- Virtuelle Speicherverwaltung: *Paging*
  - Einteilung des logischen Adreßraums in Seiten
  - Einteilung des physichen Adreßraums in Kacheln
  - jede Seite kann
    - auf beliebige Kachel im Speicher abgebildet werden
    - auf Festplatte ausgelagert werden
  - Hardware (MMU) bildet bei jedem Speicherzugriff logische auf physische Adresse ab
  - Beschreibung der Abbildung in Seitentabelle
    - pro Seite ein Eintrag, enthält u.a.:
      - Kachel-Nummer
      - *Present*-Bit: ist der Seite eine Kachel zugewiesen?

6.4 Zusammenfassung / Wiederholung ...

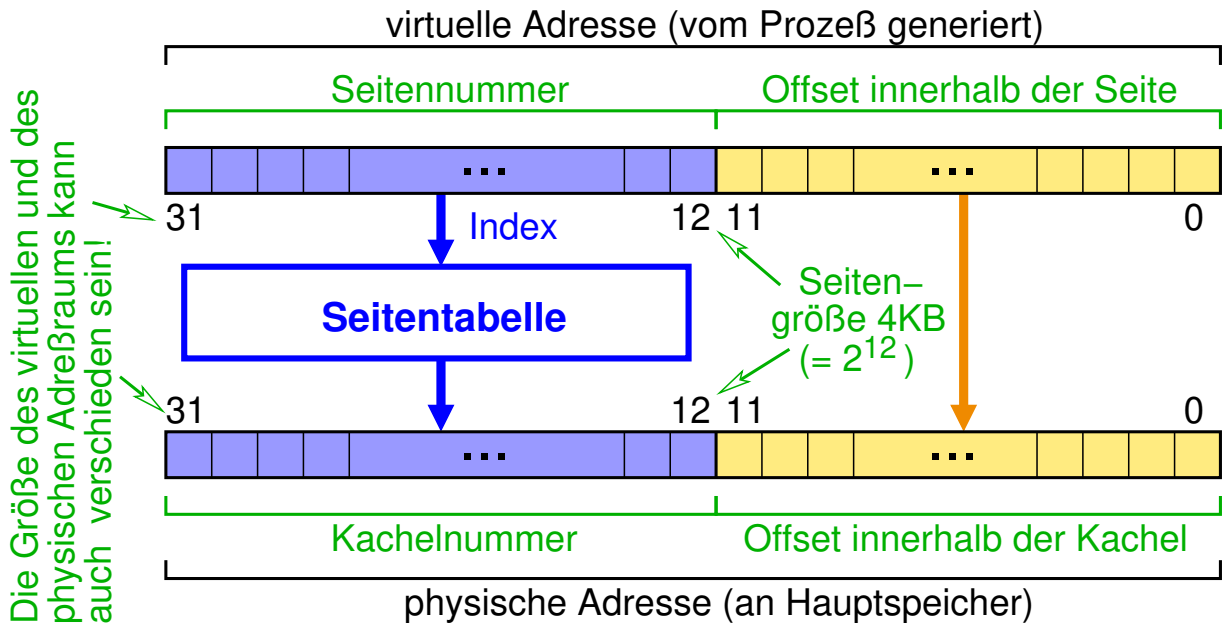
- Grundidee des *Paging*



* Diese Seiten sind nicht in den Hauptspeicher abgebildet
Sie könnten z.B. auf Hintergrundspeicher ausgelagert sein.

6.4 Zusammenfassung / Wiederholung ...

- Prinzip der Adreßumsetzung



6.4 Zusammenfassung / Wiederholung ...

- Mehrstufige Seitentabellen
  - Tabellen tieferer Stufen nur vorhanden, falls nötig
- TLB: Cache in der MMU
  - speichert die zuletzt verwendeten Tabelleneinträge
- Dynamische Seitenersetzung
  - nur die aktuell benötigten Seiten (*Working Set*) werden im Hauptspeicher gehalten
  - Rest auf Plattenspeicher verdrängt
  - bei Zugriff auf ausgelagerte Seite: Seitenfehler
    - BS lädt Seite in Hauptspeicher, muß ggf. andere Seite verdrängen (Seitenersetzung)

6.4 Zusammenfassung / Wiederholung ...

- Seitenersetzungsalgorithmen
  - bestimmen, welche Seite verdrängt wird
  - Optimale Strategie: die Seite, die in Zukunft am längsten nicht benötigt wird
  - NRU: vier Klassen gemäß **R** und **M**-Bit
  - FIFO: die Seite, die am längsten im Hauptspeicher ist
  - *Second Chance*: die älteste Seite, die seit letztem Seitenwechsel nicht benutzt wurde
    - *Clock*-Algorithmus: effiziente Implementierung
  - LRU: die Seite, die am längsten nicht benutzt wurde