Kapitel 7

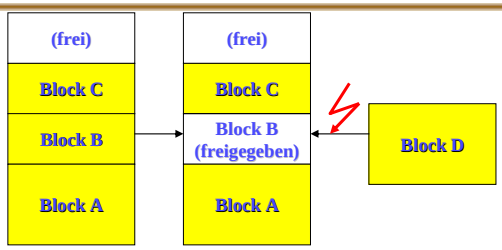
Speicherverwaltung

Speicherverwaltung durch das Betriebssystem

- **Freispeicherverwaltung**
- **Speicherhierarchie**
- **Virtuelle Adressierung**
- **Virtuelle Speicherung**

- **Freispeicherverwaltung, Dienste:**
  - Speicheranfrage (ob noch freier Speicher vorhanden ist), } oft miteinander verbunden
  - Zuteilung eines Blocks gegebener Größe (Allokation), } oft miteinander verbunden
  - Verlängerung eines bereits allozierten Blocks (ggfs. mit Adreßwechsel)
  - Rücknahme (mit Verschmelzung aneinandergrenzender Blöcke),
  - Kompaktifizierung (Lücken- oder Freispeichersammlung, garbage collection
- **Verschnittproblem**
  - interner Verschnitt: Speicherblöcke werden alloziert, die größer sind als gewünscht.
  - externer Verschnitt: (Stückelung, Fragmentierung)

Externer Verschnitt



- **Abhilfe gegen den externen Verschnitt**
  - Standardisierte Blockgrößen in einem Pool (z. B. 1 Kbyte, 8 Kbyte, 64 Kbyte, ...)
  - Kachelung: Aufteilung des angeforderten Blocks in physikalisch nicht aneinandergrenzende Bereiche einheitlicher Größe (Pages, Kacheln von z. B. 512 byte oder 4 Kbyte). Dazu ist eine spezielle Form der virtuellen Adressierung nötig
  - Lückensammlung: (garbage collection). Belegte Blöcke werden zusammengeschoben, so daß die Lücken größer werden bzw. eine einzige Lücke entsteht.
    - bei Bedarf (Erschöpfung des Vorrats). Das kostet spürbar Zeit.
    - bei CPU-Leerlauf durch den Leerlaufprozeß / nebenläufiger Kompaktifizierungsprozeß
    - Spezialform der Speicherkompaktifizierung: Prozeßauslagerung mit virtuellem Speicher

Belegungsstrategie für Speicheranforderungen

- **Das Betriebssystem führt eine Freiliste und eine Belegtliste (Adresse+Prozeßid.)**
- **Strategien**
  - Auswahl des ersten passenden Stücks (first fit, rotating first fit)
  - Auswahl des am besten passenden Stücks (best fit)
  - Auswahl des am schlechtesten passenden Stücks (worst fit)
  - Auswahl eines zufälligen, aber passenden Stücks (random fit)

- **Freiliste** $F = \{ (a_i, L_i) \mid i = 1, ..., M \}$

First-fit und next-fit

- **Auswahl des ersten passenden Stücks ((rotating-) first-fit-Methode)**
  - Freiliste der Blöcke nach aufsteigenden Anfangsadressen geordnet
    $a_1 < a_2 < ... < a_M$: Suche (a_i, L_i) mit $L_i >= x$ (angeforderte BlockLänge)
  - Falls $x <> L_i$ wird das Reststück auf der Freiliste belassen
  - First fit: Suche beginnt jeweils bei Index 1, Rotating-first-fit arbeitet zirkulär

Anforderung	freie Blöcke	A1	freie Blöcke	A2
---	200, 300, 500, 900	--	200, 300, 500, 900	--
400	200, 300, 100, 900	3	200, 300, 100, 900	3
100	100, 300, 100, 900	1	200, 300, 100, 800	1
150	100, 150, 100, 900	2	50, 300, 100, 800	1
200	100, 150, 100, 700	4	50, 100, 100, 800	1
300	100, 150, 100, 400	4	50, 100, 100, 500	2
500	geht nicht	4	50, 100, 100 ---	4
	first fit	Σ 18	*next fit*	Σ 12

Best-fit / worst-fit

- **Best-fit**
  - Freiliste nach steigenden Blocklängen geordnet, also $L_1 \le L_2 \le ... \le L_M$
    Suche Index i, so daß $L_{i-1} < x <= L_i$
  - Falls ein Restblock übrigbleibt, muß er neu in die Freiliste eingeordnet werden
  - Problem: best-fit erzeugt besonders viele nutzlos kleine Blöcke erzeugt
- **Worst-fit**
  - Suche unter den "passenden" Speicherbereichen, den größten
  - Vorteil: Verbleibende Reststücke können besonders gut wieder genutzt werden

Anforderung	freie Blöcke	geordnet	geordnet
---	800, 500, 300	300, 500, 800	800, 500, 300
200	600, 500, 300	100, 500, 800	600, 500, 300
300	300, 500, 300	100, 200, 800	500, 300, 300 U
400	300, 100, 300	100, 200, 400	300, 300, 100 U
250	50, 100, 300	100, 150, 200 U	300, 100, 50 U
250	50, 100, 50	geht nicht	100, 50, 50 U
	first fit	*best fit*	*worst fit*

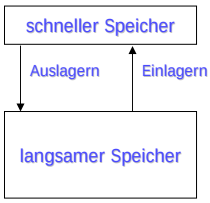
Vergleich der Verfahren

- Best-fit: am schlechtesten
- worst- und best-fit-Verfahren: das Verschmelzen freier Blöcke ist besonders aufwendig
- rotating-first-fit: am besten, da die Zerstückelung am geringsten ist

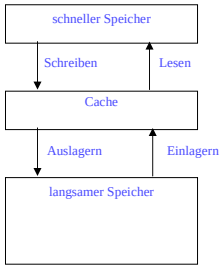
- **Bemerkungen zur Implementierung**
  - Vektormethode: Allokationsvektor zu Blöcken hinreichender Größe; Verschmelzen von Blöcken erfolgt automatisch
  - Tabellen-/Listenmethode: Verzeigerte Datenstruktur; explizites Verschmelzen nötig
- **Buchhaltung**
  - Intern: Eigener Speicherbereich für die Aufnahme von Blockinformationen
  - Extern: Einbettung der Informationen vor dem angeforderten Speicherbereich

Speicherhierarchie

Speicherebene	Zugriffszeit	Kapazität	Kosten
Prozessor-Register		100 byte	50 000 DM/Mbyte
prozessorinterner Cache		10 Kbyte	5 000 DM/Mbyte
externer Cache	10 ns	100 Kbyte	500 DM/Mbyte
Hauptspeicher (RAM)	100 ns	10 Mbyte	50 DM/Mbyte
Platten-Cache (Hauptspeicher)	100 ns	1 Mbyte	50 DM/Mbyte
Festplattenspeicher	10 ms	1000 Mbyte	5 DM/Mbyte
Magnetbandspeicher	10 s	1000 Mbyte	0,5 DM/Mbyte



- **Hierarchie mit den Operationen Einlagern und Auslagern**
  - Je häufiger auf bestimmte Daten zugegriffen wird, desto schneller sollte der Zugriff auf sie sein.
- **Oder umgekehrt:**
  - Zugriffsoperationen, die häufig vorkommen, sollten schnell ausgeführt werden.



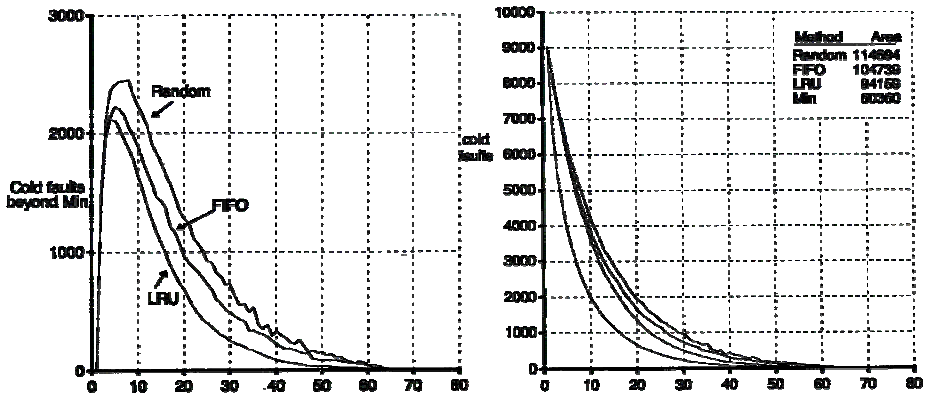
Caches

- **Bei Einlagerung wird nicht nur das benötigte Datum, sondern ein größerer Datenblock aus seiner Umgebung miteingelagert**
- **Lokalitätsprinzip**
  - viele Schleifen; größere Sprünge kommen relativ selten vor
  - Unterprogramme werden meist mehrfach hintereinander aufgerufen (in Schleifen)
  - Lokale Daten der Unterprogramme werden auf dem Stapel verwaltet
  - Verarbeitung von Zeichenketten --> relativ wenige, nah beieinanderliegende Daten
  - Codelokalität besser als Datenlokalität (dort gute Datenorganisation wichtig!)
- **Lokalitätsprinzip --> Trefferrate ist abhängig von der Cache-Größe, aber so gut wie nicht von der Größe des langsameren Speichers**
- **Rechenbeispiele:**
  - Trefferrate 95 %, Cache-Zugriffszeit 50 ns; Hauptspeicher-Zugriffszeit 200 ns; damit effektive Zugriffszeit: 0,95 * 50 ns + 0,05 x 200 ns = 57,5 ns.
  - Trefferrate 95 %, Plattencache-Zugriffszeit 1µs, Platte mit 10 ms: ~ 0,5 ms.
- **Trefferrate, abhängig von**
  - von der Cache-Größe
  - von der Ersetzungsstrategie
  - von der Programm- und Datenorganisation

Ersetzungsstrategien

- **LRU-(least recently used)-Prinzip**
  - Diejenige Cache-Einheit wird ausgewählt, die am längsten nicht mehr referiert wurde (gleichgültig, ob durch Schreib- oder Lesezugriff).
  - Führen eines Alterungszählers für jede Cache-Einheit (Zurücksetzen bei Zugriff)
  - Im Austauschfall wird vom Betriebssystem eine der Cache-Einheiten mit dem höchsten Zählerstand zum Austransfer ausgesucht.
- **LFA (Least Frequently Accessed)**
  - Austausch der am wenigsten benutzten Cache-Einheit
  - Feststellung durch einen Referenzzähler, der bei einem Zugriff auf die zugehörige Cache-Einheit durch die Hardware um 1 erhöht und nach bestimmten Zeitintervallen bzw. nach einem Austausch durch das Betriebssystem auf 0 gesetzt wird.
- **LRL (Least Recently Loaded)**
  - Austausch der am längsten geladenen Cache-Einheit (FIFO-Prinzip)
- **Zufallsauswahl**

Vergleich der Strategien



Kalt- und Warmstartverhalten der Ersetzungsstrategien

Behandlung von Schreiboperationen auf den Cache

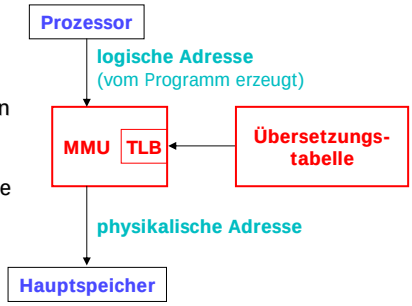
- **Durchschreiben (write through)**
  - Schreiben sowohl in Cache als auch in den unterlagerten Speicher
  - Größtmögliche Übereinstimmung (wichtig z. B. in Mehrprozessorsystemen)
  - Zeitaufwendig, allerdings kommen Schreiboperationen viel seltener vor
- **Zurückschreiben (write back)**
  - Erst bei Austausch der Cacheinhalte werden die Daten auch auf der langsameren Ebene geändert
  - Zeitlich günstiger; jedoch gefährlich (z.B. Systemfehler, Stromausfall)
- **Schreiben auf Anforderung (write on demand)**
  - Schreiboperationen werden nur auf besondere **Anforderung** durchgeführt (z. B. bei einem Prozeßwechsel)
  - Vorteilhaft bei temporär gültigen Cache-Inhalten, die nur begrenzte Lebensdauer haben (z. B. Daten auf Stapel, die bei Eintritt in ein Unterprogramm erzeugt werden und bei Verlassen desselben ihre Gültigkeit verlieren)
  - Diese Strategie erfordert im Falle eines Caches zwischen Prozessor und Hauptspeicher eine ausgefeiltes Zusammenspiel zwischen Cache-Hardware, Betriebssystem und Übersetzer.

Virtueller Speicher

- **Problem: Programme sind zu groß für den verfügbaren Arbeitsspeicher**
- **Frühere Lösung**
  - Aufspaltung der Programme in Overlays, die auf der Festplatte gespeichert waren und dynamisch ein- und ausgelagert wurden
  - Zunächst wurde Overlay 0 ausgeführt
  - Wenn Overlay 0 beendet war, wurde Overlay 1 ausgeführt, etc.
  - Problem: Programmierer mussten Programme in kleine, modulare Teile aufspalten
- **Virtueller Speicher**
  - Grundidee: Programmcode, Daten und Stack dürfen zusammen größer sein als der verfügbare Arbeitsspeicher (Hauptspeicher)
  - Betriebssystem hält die benötigten Teile im Arbeitsspeicher und die anderen Teile liegen auf der Festplatte
  - Kombinierbar mit Multiprogrammierung: Teile verschiedener Programme liegen gleichzeitig im Hauptspeicher

Virtuelle Adressierung

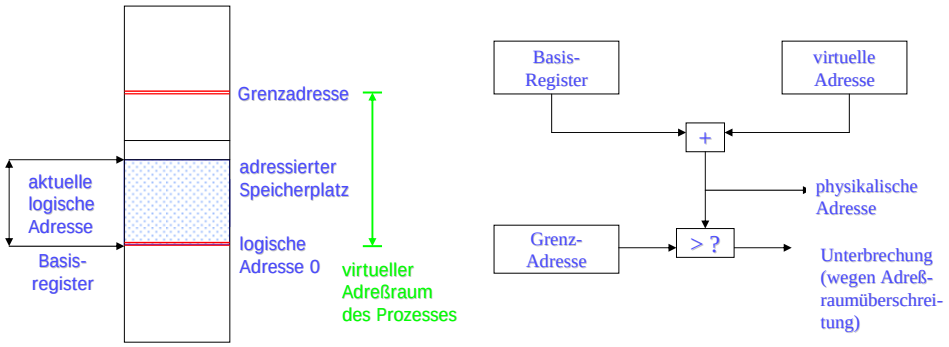
- **Virtuelle Adressierung**
  - virtuelle (logische) Adressen und physikalische Adressen können unterschiedlich sein
  - Virtuelle Adressen werden mittels einer Adreßabbildung in physikalische umgesetzt
  - Es wird spezielle Hardware (MMU, memory management unit) benötigt, deren Verwaltung dem Betriebssystem obliegt
- **Speicherverwaltungseinheit**
  - Zugriffskontrolle (Speicherschutz)
  - Übersetzungstabelle mit Adreßzuordnungen und Zugriffsrechten (Deskriptoren)
  - TLB (translation lookaside buffer) interner Cache der MMU mit nur dem jeweils gerade benötigten Teil der Übersetzungstabelle.



Virtuelle Adressierung

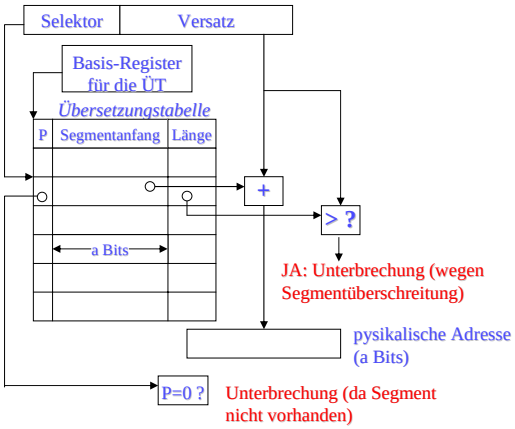
- **Betriebssystem muß die Übersetzungstabelle erstellen und pflegen**
  - Globaler Teil (für das Betriebssystem selbst)
  - Lokaler Teil (jeweils einer für jeden Prozeß)
  - Nur das Betriebssystem "weiß", in welchen physikalischen Speicherbereich es sich selbst oder einen Prozeß geladen hat.
  - Bei einem Kontextwechsel muß das Betriebssystem auch die prozessbezogenen Übersetzungstabellen umschalten und den TLB löschen
- **Vorteile der virtuellen Adressierung**
  - Unterschiedl. Größe von adressierbarem und physikalisch vorhandenem Speicher möglich
  - Verschiebbarkeit (Relozierbarkeit): Es muß nur die Adreßabbildung geändert werden
  - keine Änderung der virtuellen Adressen bei Verlängerung eines allozierten Speicherabschnittes im Rahmen der Freispeicherverwaltung nötig
  - Möglichkeit zur Speicherkompaktifizierung im laufenden Betrieb
  - Lageunabhängigkeit: Ein Programm beginnt immer bei der virtuellen Adresse 0
  - Speicherschutz
  - Abprüfung von Zugriffsrechten (Lesen/Schreiben) eines Prozesses "nebenbei"
  - Zugriffsstatistik: Möglichkeit zur Sammlung von Daten (z.B. Zugriffshäufigkeit)

Virtueller Speicher: Lineare Adreßumsetzung



- **Probleme:**
  - gemeinsamer Speicher, Anforderung von weiteren Speichersegmenten
  - Belegungslücken lassen sich nur schwierig wieder nutzen
  - Es können lediglich ganze Prozesse ein- und ausgelagert werden
  - Ein Prozeß kann nicht größer als der zur Verfügung stehende Hauptspeicher sein

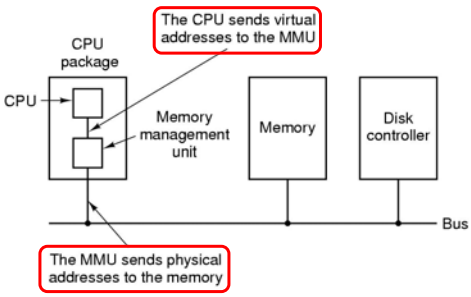
Virtueller Speicher: Segmentierung



- **virtuelle Adresse: Selektor+Versatz**
- **Eingreifen des Betriebsystems**
  - Prüfung, ob der Versatz die Segmentlänge überschreitet
  - Prüfung, ob das Segment nicht geladen ist

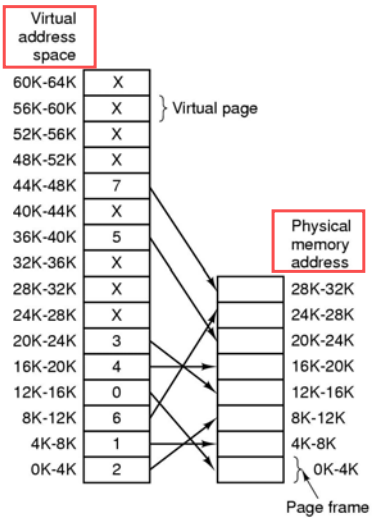
Virtueller Speicher: Paging

- **Paging**
  - Technik zur Verwaltung des virtuellen Speichers
- **Virtuelle Adressen**
  - Speicheradressen im Virtuellen Adressraum
- **Memory Management Unit (MMU)**
  - I. Allg. heute Teil des CPU-Chips
  - Aufgabe: Abbildung der virtuellen Adressen auf die physischen Adressen
  - Z.B. Abbildung der 16-Bit virtuellen Adressen (0 – 64 K) auf den 32 K physischen Arbeitsspeicher
  - Max. Programmgröße ist 64 K
  - Programme liegen komplett auf der Festplatte und Teile werden bei Bedarf eingelagert



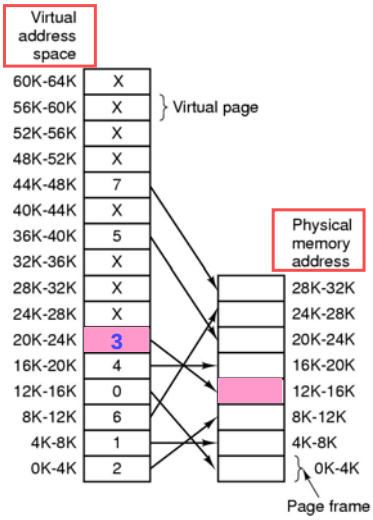
Virtueller Speicher: Paging

- **MMU – Beispiel :**
  - 16 virtuelle Seiten werden auf 8 Seitenrahmen abgebildet
- **Seiten (Pages)**
  - Virtueller Adressraum wird in Seiten unterteilt
  - Die entsprechenden Einheiten im physischen Speicher heißen Seitenrahmen oder Seitenkacheln (page frames)
  - Seiten und Seitenrahmen haben immer die gleiche Größe
    (in realen Systemen 512 Byte – 64 KB)
  - Zwischen Arbeitsspeicher und Festplatte werden immer komplette Seiten übertragen
  - Beispiel: virtuelle Adressen 0 bis 4K werden auf physische Adressen 8K bis 12K abgebildet

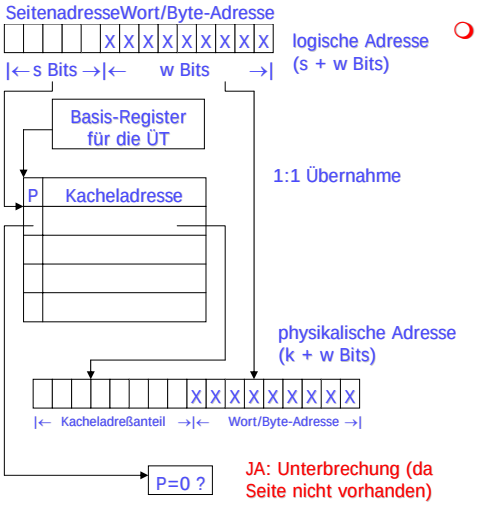


Virtueller Speicher: Paging

- **MMU – Beispiel**
  - Virtuelle Adresse 20500: 20 + 20480 = 20 + 20K
    -> 20 Byte vom Beginn der virtuellen Seite 5
  - Abbildung auf die physische Adresse
    12K + 20 = 12288 + 20 = 12308
- **Present/Absent-Bit**
  - Gibt an, ob eine virtuelle Seite auf den physischen Speicher abgebildet wird oder nicht
- **Seitenfehler (page fault)**
  - Zugriff auf eine virtuelle Seite, die momentan nicht im Arbeitsspeicher liegt, sondern erst geladen (eingelagert) werden muss



Virtueller Speicher: Paging / Kachelung (Seitenadressierung)



- **Virtuelle Adresse: Seitenadresse+Versatz (Offset)**

Segmentierung vs. Kachelung

- **Vorteile der Kachelung gegenüber der Segmentierung:**
  - Bei der Segmentierung externer Speicherverschnitt --> Lückensammlung
  - Der komplette Austausch großer Segmente kann viel Zeit in Anspruch nehmen
  - Verlängerung von Segmenten und Defragmentierung sind aufwendig
  - Der permanent geladene Adreßbereich läßt sich auf den Arbeitsbereich beschränken
- **Vorteile der Segmentierung gegenüber der Kachelung:**
  - Interner Verschnitt bei Kachelung
  - Natürliche Entsprechung der Segmente zu den Komponenten eines Programms
  - natürliche Abbildung von Zugriffsrechten
  - geringere Anzahl an Segmenten pro Prozeß, daher weniger Verwaltungsaufwand
  - Vereinfachung der Cache-Tabelle in der MMU (TLB) zur Speicherung der Segment-Deskriptoren
  - Für die Kachelung sind spezielle Prozessoren nötig (mit Befehlen, die bei Adreßfehlern unterbrechbar sind)

Virtuelle Speicherung
Alternativen

- **Auslagerung von ganzen Prozessen**
  - Auslagerung von Prozessen, die ...
    - **ohnehin warten**
    - **schon lange gerechnet haben**
    - **schlechte Priorität haben**
    - **besonders hohen Speicherbedarf haben**
  - Kandidaten für den Eintransfer sind umgekehrt Prozesse, die
    - **bereit (geworden) sind und**
    - **bessere Priorität als der derzeit rechnende Prozeß haben oder**
    - **lange gewartet haben**
- **Auslagerung von Segmenten**
  - Nur Teile eines Prozesses (Segmente) werden ausgelagert
- **Auslagerung von Kacheln**
  - Nur (kleine) Teile eines Prozesses müssen sich tatsächlich im physik. Speicher befinden, damit ein Prozeß lauffähig ist, sogenannter Arbeitsbereich (working set)