Speicher (2)

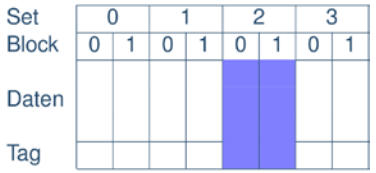
Caching, Paging

Cache-Verdrängung

- Bei Cache Miss (Block nicht im Cache):
  - Block aus Hauptspeicher holen
  - und im Cache speichern (für folgende Zugriffe)
    → Was bedeutet das für den Cache?
- Verdrängung:
  - *direkt-abbildender Cache*: Ort für neu geholten Block steht fest. Falls bereits belegt: alten Inhalt überschreiben
  - *n-Wege-assoziativer Cache* ($n \geq 2$): mehrere mögliche Orte für Block. Falls davon alle belegt: einen **auswählen** und überschreiben (→ Strategie)

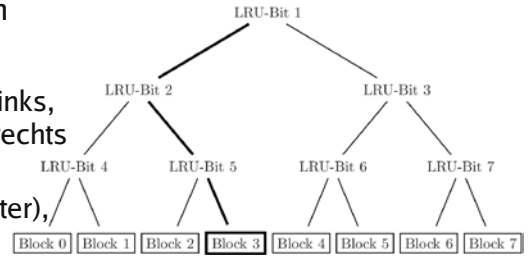
Verdrängungsstrategien

- sequentiell (Zähler für jedes Set)
- zufällig (keine Zähler)
- „lange unbenutzten“ oder „selten benutzten“ Eintrag verdrängen
  → Least-Recently-Used-Strategien (LRU) und Least-Frequently-Used-Strategien (LFU)
  - Reihenfolge oder Zugriffshäufigkeiten mit Zählern verwalten



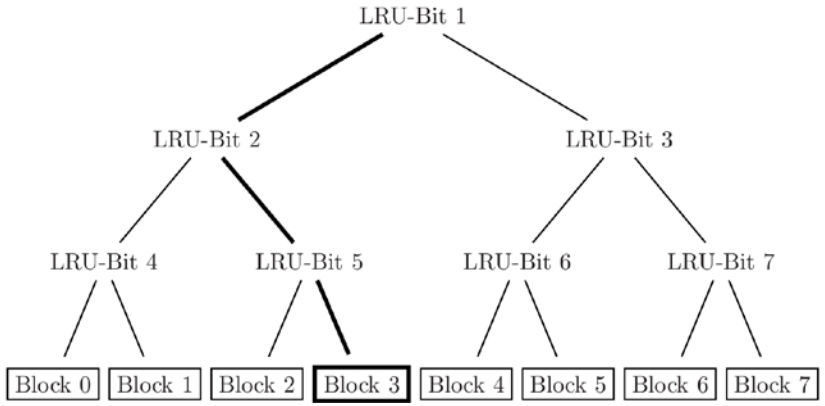
LRU und Pseudo-LRU

- Verdrängung mit LRU sehr aufwendig (braucht je Set $\log_2(n!)$ Bits, da $n!$ Reihenfolgen)
- Alternative: Pseudo-LRU
  - z. B.: 8-Wege-assoziativer Cache (8 Blöcke je Set)
  - Entscheidungsbaum
    - findet LRU-Block
    - 0 = letzter Zugriff links, 1 = letzter Zugriff rechts
    - Es gibt 7 „innere“ Knoten (Nicht-Blätter), 8 Blätter



Pseudo-LRU, Beispiel (1)

- Beispiel: **1**,**0**,0,1,**0**,1,1; zuletzt benutzt: [3]



Pseudo-LRU, Beispiel (2)

- Beispiel: **1**,**0**,0,1,**0**,1,1; zuletzt benutzt: [3]
- Block [3] verdrängen. Nach dem Update: alle Bits auf Pfad zu Block [3] „kippen“

  0,**1**,0,1,**1**,1,1

  sorgt dafür, dass mindestens diese drei Bits erneut zu kippen sind, bevor der Block verdrängt wird
- Lesezugriff: Pfad zu Block als letzten Zugriff markieren (Nullen, Einsen passend eintragen)
- keine echte Speicherung der Reihenfolge:
  Baum kennt 2^7=128 Zustände, vgl. 8! = 40.320

Paging

(dt. auch: „Kachelverwaltung“)

Ausgangslage

- Speicher zu knapp für große Programme
  → Overlay-Programmierung
- Programmteile dynamisch nachladen, wenn sie benötigt werden
- Programmierer muss sich um Aufteilung in Overlays kümmern

Overlay-Programmierung

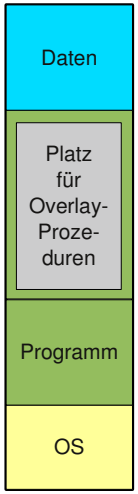
Turbo Pascal, um 1985-90:

```
program grossesprojekt;

overlay procedure kundendaten;
...

overlay procedure lagerbestand;
...

{ Hauptprogramm }
begin
  while input <> "exit" do begin
    case input of
      "kunden": kundendaten;
      "lager":  lagerbestand;
    end;
  end;
end.
```



Lösung des Problems

- Virtueller Speicher, der das gesamte Programm aufnehmen kann
- Programm sieht Speicherbereich, der ihm zur Verfügung gestellt wurde – wie viel wirklich vorhanden ist, spielt (für das Programm) keine Rolle

Virtuelle Speicherverwaltung (Paging)

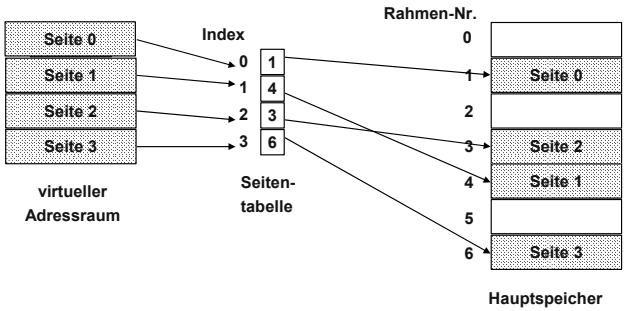
- Aufteilung des Adressraums in Seiten (pages) fester Größe und des Hauptspeichers in Seitenrahmen (page frames) gleicher Größe.
  - Typische Seitengrößen: 512 Byte bis 8192 Byte (immer Zweierpotenz).
- Der lineare, zusammenhängende Adressraum eines Prozesses („virtueller" Adressraum) wird auf beliebige, nicht zusammenhängende Seitenrahmen abgebildet.
- Eine einzige Liste freier Seitenrahmen wird vom Betriebssystem verwaltet.

Virtuelle Speicherverwaltung (Paging)

- Die Berechnung der physikalischen Speicheradresse aus der vom Programm angegebenen virtuellen Adresse
  - geschieht zur Laufzeit des Programms,
  - ist transparent für das Programm,
  - muss von der Hardware (MMU) unterstützt werden
- Vorteile der virtuellen Speicherverwaltung:
  - Einfache Zuteilung von Hauptspeicher
  - Keine externe Fragmentierung (unbenutzbare Speicherbereiche)
  - Kein Aufwand für den Programmierer

Virtueller Adressraum (1)

- Beim Paging wird der Zusammenhang zwischen Programmadresse und physikalischer Hauptspeicheradresse erst zur Laufzeit mit Hilfe der Seitentabellen hergestellt.



Virtueller Adressraum (2)

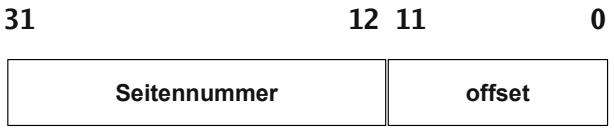
- Die vom Programm verwendeten Adressen werden deshalb auch virtuelle Adressen genannt.

- Der virtuelle Adressraum eines Programms ist der lineare, zusammenhängende Adressraum, der dem Programm zur Verfügung steht.

Adressübersetzung beim Paging (1)

- Programmadresse wird in zwei Teile aufgeteilt:
  - eine Seitennummer
  - eine relative Adresse (offset) in der Seite

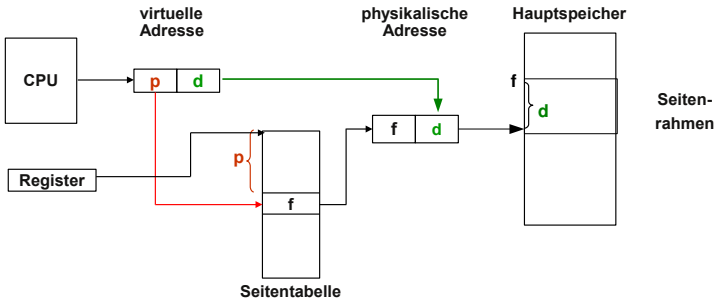
Beispiel: 32-bit-Adresse bei einer Seitengröße von 4096 ($=2^{12}$) Byte:



Adressübersetzung beim Paging (2)

- Für jeden Prozess gibt es eine Seitentabelle (page table). Diese enthält für jede Prozessseite
  - eine Angabe, ob die Seite im Speicher ist,
  - die Nummer des Seitenrahmens im Hauptspeicher, der die Seite enthält.
- Ein spezielles Register enthält die Anfangsadresse der Seitentabelle für den aktuellen Prozess.
- Die Seitennummer wird als Index in die Seitentabelle verwendet.

Adressübersetzung beim Paging (3)



- Für jeden Hauptspeicherzugriff wird ein zusätzlicher Hauptspeicherzugriff auf die Seitentabelle benötigt. Dies muss durch Caches in der Hardware beschleunigt werden!
- Seite nicht im Speicher → spezielle Exception, einen sog. page fault (Seitenfehler) auslösen.

Virtueller Speicher allgemein (1)

- Mehr Prozesse können effektiv im Speicher gehalten werden
  → bessere Systemauslastung
- Ein Prozess kann viel mehr Speicher anfordern als physikalisch verfügbar

Virtueller Speicher allgemein (2)

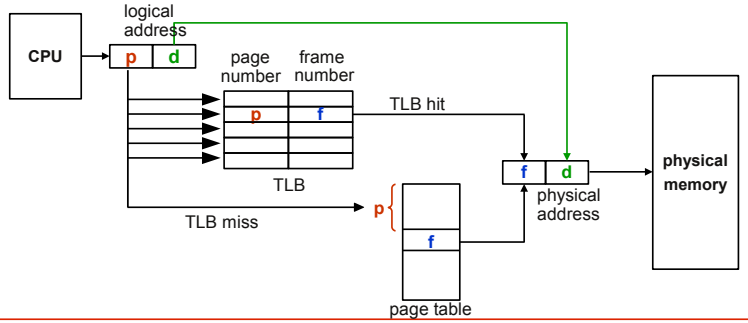
- allgemeiner Vorgang:
  - Nur Teile des Prozesses befinden sich im physikalischen Speicher
  - falls Zugriff auf eine Adresse, die ausgelagert ist:
    - BS setzt den Prozess auf blockiert
    - BS setzt eine Disk-I/O-Leseanfrage ab
    - Nach Laden des fehlenden Stücks (Seite oder Segment) wird ein I/O-Interrupt abgesetzt
    - das BS setzt Prozess zuletzt wieder in den Bereit- (Ready-) Zustand

Virtueller Speicher allgemein (3)

- Lokalitätsprinzip:
  - Zugriffe auf Daten und Programmcode häufig lokal gruppiert;
    → Annahme gerechtfertigt, dass nur wenige Prozessstücke während einer kurzen zeitlichen Periode gleichzeitig vorgehalten werden müssen

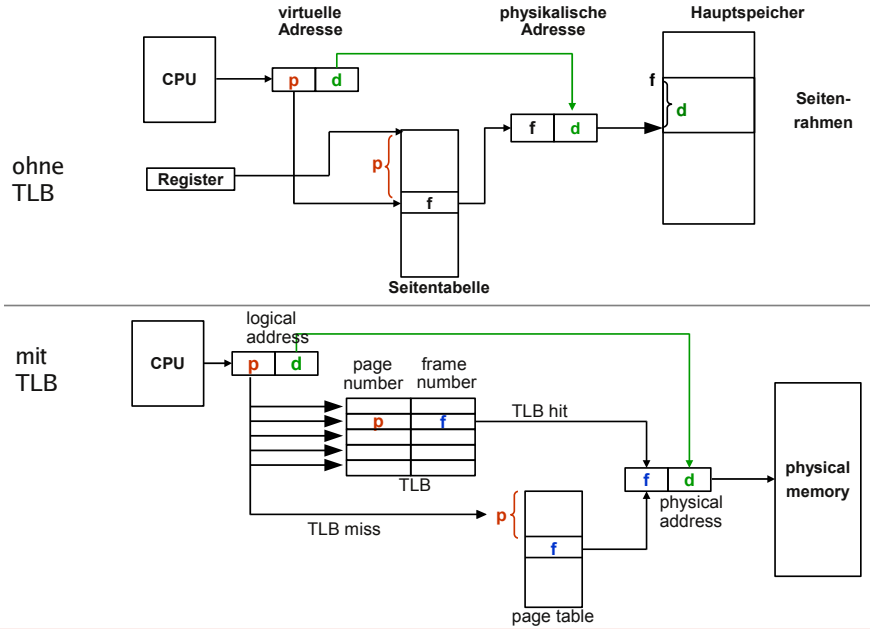
Translation Look-Aside Buffer (1)

- Translation Lookaside Buffer (TLB): schneller Hardware-Cache, mit den zuletzt benutzten Seitentabelleneinträgen
- Assoziativ-Speicher: bei Übersetzung einer Adresse wird deren Seitennummer gleichzeitig mit allen Einträgen des TLB verglichen.



Translation Look-Aside Buffer (2)

- Treffer im TLB
  → Speicherzugriff auf Seitentabelle unnötig
- Fehltreffer
  → Zugriff auf die Seitentabelle
  Alten Eintrag im TLB durch neuen ersetzen
- Trefferquote (hit ratio) beeinflusst die durchschnittliche Zeit einer Adressübersetzung.
- Lokalitätsprinzip: Programme greifen meist auf benachbarte Adressen zu
  → auch bei kleinen TLBs hohe Trefferquoten (typisch: 80-98%).



Lokalitätsprinzip

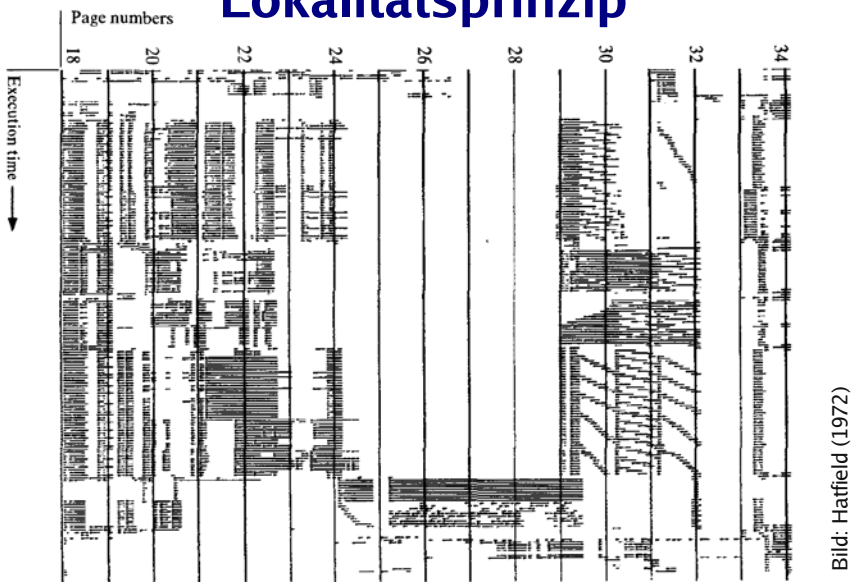


Bild: Hatfield (1972)

Demand Paging (1)

- Der Adressbereich eines Prozesses muss nicht vollständig im Hauptspeicher sein.
  - Das Lokalitätsprinzip besagt, dass ein Prozess in einer Zeitspanne nur relativ wenige, nahe beieinanderliegende Adressen anspricht.
  - Teile des Programms werden bei einem bestimmten Ablauf möglicherweise gar nicht benötigt (Spezialfälle, Fehlerbehandlungs-routinen etc.).

Demand Paging (2)

- Demand Paging bedeutet
  - dass eine Seite nur dann in den Speicher geladen wird, wenn der Prozess sie anspricht,
  - dass eine Seite auch wieder aus dem Speicher entfernt werden kann.

- Vorteile von Demand Paging:
  - Der Adressbereich eines Prozesses kann größer sein als der physikalische Hauptspeicher.
  - Prozesse belegen weniger Platz im Hauptspeicher, somit können mehr Prozesse gleichzeitig aktiv sein.

Voraussetzungen für Demand Paging (1)

- Jeder Eintrag in der Seitentabelle enthält ein valid bit, das angibt, ob die Seite im Speicher ist oder nicht.
- Wenn ein Prozess eine Seite anspricht, die nicht im Speicher ist, wird eine spezielle Exception ausgelöst, ein sog. page fault.
- Eine Betriebssystem-Routine, der page fault handler, lädt bei einem page fault die benötigte Seite in den Speicher.

Voraussetzungen für Demand Paging (2)

- Falls kein freier Seitenrahmen im Speicher vorhanden ist, muss eine andere Seite ersetzt werden. Für die Auswahl der zu ersetzenden Seite muss eine Strategie implementiert werden.
- Die durch den page fault unterbrochene Instruktion muss erneut ausgeführt werden (können).

Seitenersetzung (1)

- Wenn bei einem Page Fault kein freier Seitenrahmen zur Verfügung steht, muss das Betriebssystem einen frei machen.
- Ein Algorithmus wählt nach einer bestimmten Strategie diesen Seitenrahmen aus.

Seitenersetzung (2)

- Falls die zu ersetzende Seite, seit sie zuletzt in den Speicher geholt wurde, verändert wurde, muss ihr aktueller Inhalt gesichert werden:
  - Ein modify bit (oder dirty bit) im Seitentabelleneintrag vermerkt, ob die Seite verändert wurde.
  - Eine veränderte Seite wird auf Platte gesichert (im sog. Page- oder Swap-Bereich).

Seitenersetzung (3)

- Eine unveränderte Seite kann später - bei Bedarf - wieder von der alten Stelle auf der Platte geladen werden.
- Im Seitentabelleneintrag für die ersetzte Seite wird
  - das valid bit gelöscht,
  - vermerkt, von wo die Seite wieder geladen werden kann.

Seitenersetzungsstrategien (1)

- Ziel: Es sollen so wenig Page Faults wie möglich auftreten
  (Nachladen von Platte sehr teuer)
- Verschiedene Strategien
  - FIFO
  - LRU (Least Recently Used, wie bei Cache)
  - ... (mehr dazu: Vorlesung Betriebssysteme)

First In First Out (FIFO)

Die Seite ersetzen, die schon am längsten im Speicher ist.

- Vorteil: Sehr einfach zu implementieren:
  - Es wird eine verkettete Liste der Seiten im Speicher (globale Strategie) bzw. der Seiten eines Prozesses (lokale Strategie) unterhalten.
  - Bei einem Page Fault wird die erste Seite der Liste ersetzt und die neue Seite ans Ende der Liste angefügt.
- Nachteil: Die ersetzte Seite kann in dauernder Benutzung sein und gleich wieder angefordert werden.

Least Recently Used (LRU) (1)

Die Seite ersetzen, die am längsten nicht benutzt worden ist.

- Vorteil: In der Regel weniger Page Faults als FIFO.
- Nachteil: Aufwändige Implementierung.

Zwei mögliche Implementierungen:

- mit Zähler
- mit verketteter Liste

Least Recently Used (LRU) (2)

- Implementierung mit Zähler:
  - Systemweiten Zähler bei jedem Speicherzugriff inkrementieren.
  - Aktuellen Wert des Zählers in einem Feld in der Datenstruktur vermerken, welche die angesprochene Seite beschreibt.
  - Seite mit dem kleinsten Zählerwert ersetzen.

Least Recently Used (LRU) (3)

- Implementierung mit verketteter Liste:
  - Eine verkettete Liste enthält alle Seiten.
  - Bei jedem Speicherzugriff wird die angesprochene Seite an den Anfang der Liste gebracht.
    (Liste durchsuchen und Reihenfolge ändern, also Zeiger umsetzen!)
  - Die Seite am Ende der Liste wird ersetzt.

Caching vs. Paging

Parameter	First-Level Cache	Virtueller Speicher
Größe Line/Page	16 – 128 Bytes	4.096 – 65.536 Bytes
Hit time	1 – 3 TZ *)	50 – 150 TZ
Miss penalty	8 – 150 TZ	1 – 10 Mio. TZ
Zugriffszeit	6 – 130 TZ	800.000 – 8.000.000 TZ
Transferzeit	2 – 20 TZ	200.000 – 2.000.000 TZ
Miss rate	0,1 % – 10 %	0,00001 % – 0,001 %

*) TZ = Taktzyklen

Vorschau 23.12.2010

- Memory Mapped I/O
- Lösungen der Übungen von heute und letzter Woche