

<b>7</b>	<b>Speicher</b>
<b>7.1</b>	<b>Charakteristika</b>
<b>7.2</b>	<b>Speicherhierarchie</b>
<b>7.3</b>	<b>Hauptspeicher</b>
<b>7.4</b>	<b>Zwei-Stufen-Speicher</b>
<b>7.5</b>	<b>Cache</b>
<b>7.5.1</b>	<b>Assoziativer Cache</b>
<b>7.5.2</b>	<b>Direkter Cache</b>
<b>7.5.3</b>	<b>Zusammenfassung</b>
<b>7.6</b>	<b>Seitenverfahren</b>
<b>7.6.1</b>	<b>Modell</b>
<b>7.6.2</b>	<b>Aufwand</b>
<b>7.6.3</b>	<b>Verdrängungsalgorithmen</b>
<b>7.6.4</b>	<b>Seitenverfahren und Cache</b>

## Überblick über Speicher:

### Primärspeicher:

#### Halbleiterspeicher:

##### Flüchtig:

"Static RAM",  
"Dynamic RAM".

##### Nicht flüchtig:

ROM,  
PROM,  
EPROM,  
EEPROM,  
"Flash Memory".

### Sekundärspeicher:

#### Magnetischer Speicher:

"Floppy Disk",  
Plattenspeicher,  
Band.

#### Optischer Speicher:

CD-ROM, CD-R, CD-RW,  
DVD-ROM, DVD-R, DVD-RAM,  
HD-DVD, Blu-ray Disc.

---

<b>Abkürzungen:</b>	<b>RAM</b>	<b>=</b>	<b>Random Access Memory</b>
	<b>ROM</b>	<b>=</b>	<b>Read Only Memory</b>
	<b>PROM</b>	<b>=</b>	<b>Programmable ROM</b>
	<b>EPROM</b>	<b>=</b>	<b>Erasable PROM</b>
	<b>EEPROM</b>	<b>=</b>	<b>Electrically Erasable PROM</b>
	<b>CD</b>	<b>=</b>	<b>Compact Disc</b>
	<b>DVD</b>	<b>=</b>	<b>Digital Versatile Disc</b>

## Flash-Speicher:

Flash-Speicher ist eine Art von EEPROM dar, wobei größere Datenblöcke, die Löscheinheiten, auf einmal gelöscht werden.

Beispiel: Toshiba TC58NVG0S3AFT00

Seitengröße = (2048 + 64) Byte

Blockgröße = Seitengröße \* 64

Speicherkapazität = Blockgröße \* 1024

Es wird immer ein Block gelöscht. Dies dauert 2 bis 4 ms. Das Beschreiben einer Seite dauert durchschnittlich 200 bis 700 µs. Das serielle Lesen eines Byte nimmt etwa 50 ns in Anspruch. Die zusätzlichen 64 Byte einer Seite dienen der Aufnahme eines Korrekturcodes. Dieser Speicher ist auf mindestens 100.000 Löschoptionen ausgelegt. Für das Beschreiben läßt sich eine Seite in 4 Segmente a (512 + 16) Byte untergliedern.

Beispiel aus Patterson & Hennessy, 3. Aufl., Seite 691, Daten aus 2001:

read 1 Byte: 65 ns

write 1 Byte: 1,5 µs

erase 4 KiB: 5 ms

Lesen eines Datenblocks von 64 KiB: 4,3 ms

Neuschreiben eines Blocks von 64 KiB =  
Löschzeit + Schreibzeit: 178,3 ms

## Charakteristika von Speichern:

**Kapazität:** 1 Bit bis mehrere Terabyte

**Transporteinheit:** Speicherwort bis  
Bandblock

**Adressierbare Einheit:** Byte (Hauptspeicher)  
Sektor (Festplatte)

**Zugriffs-Methode:**

seriell	(Bandzugriff)
direkt	(Plattenzugriff)
wahlfrei	(Hauptspeicher)
assoziativ	("Translation Lookaside Buffer")

**Leistung:**

**Zugriffszeit:** 0,5 Nanosekunden bis zu Minuten

**Transfer-Rate:** 1 kB/s bis 1 TB/s

**Dauerhaftigkeit der Information:**

flüchtig	–	beständig
löschar	–	permanent

**Bemerkung:** Speicher lassen sich auch danach unterscheiden, ob der Lesezugriff informationszerstörend ist oder nicht.

**Typische Speicherparameter (2004):**

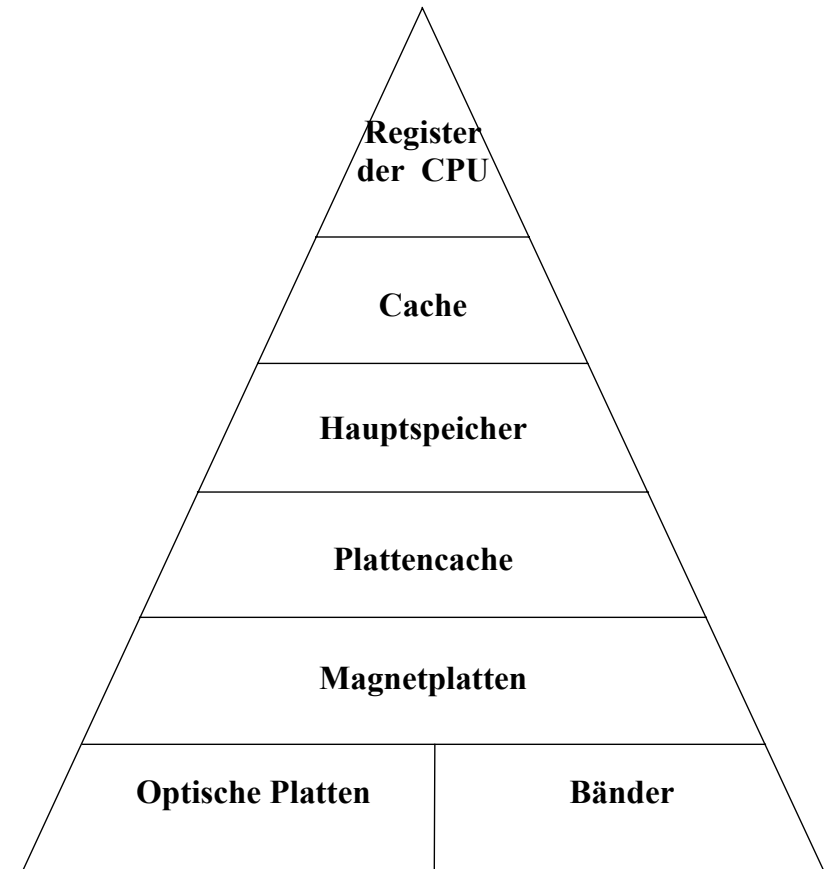
Speichertyp	Technologie	Größe	Zugriffszeit
Register		1 KiB	0,5 ns
Cache	Statisches Halbleiter-RAM	512 KiB	4 ns
Hauptspeicher	Dynamisches Halbleiter-RAM	512 MiB	50 ns
Magnetische Platte	Festplatte	200 GiB	10 ms
Optische Platte	CD-ROM	600 MiB	200 ms
Archivband	Magnetband	100 MiB bis 5 GiB	5 sec bis 30 min

**Bemerkung:** Obige Angaben sind nur als "typisch" zu betrachten.

**Speicherhierarchie:**

Geringe Zugriffszeit

Geringe Kapazität

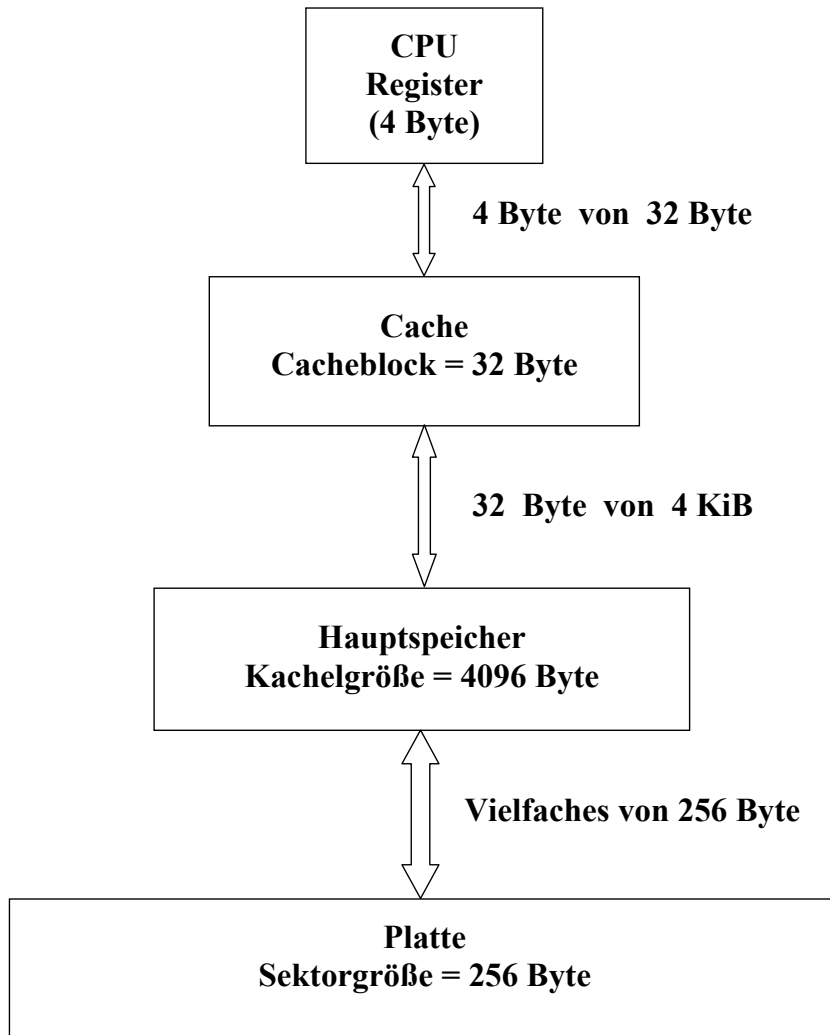


Hohe Zugriffszeit

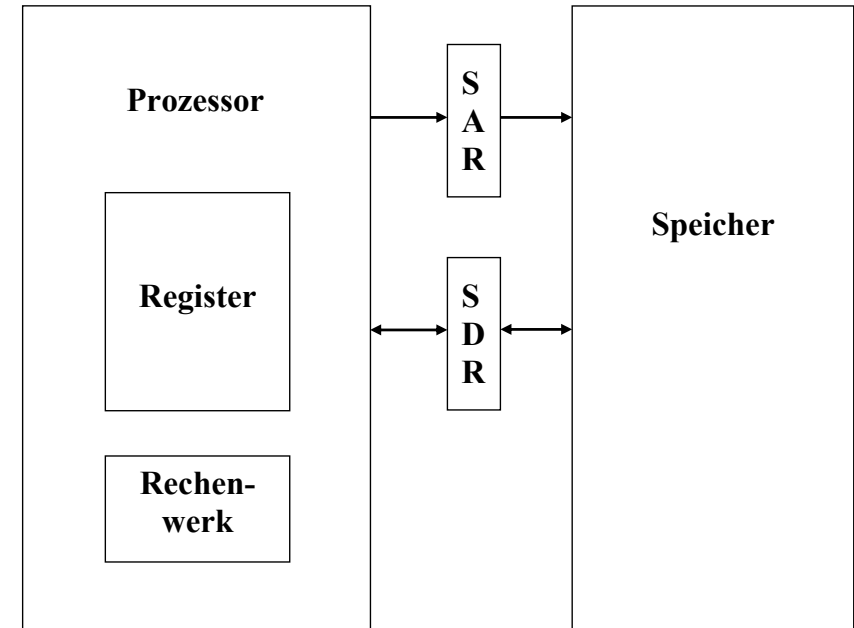
Hohe Kapazität

## Speicherhierarchie:

Beispiel: Transporteinheiten bei virtuellem Speicher, nur illustrativ



## Speicheranbindung an Prozessor:



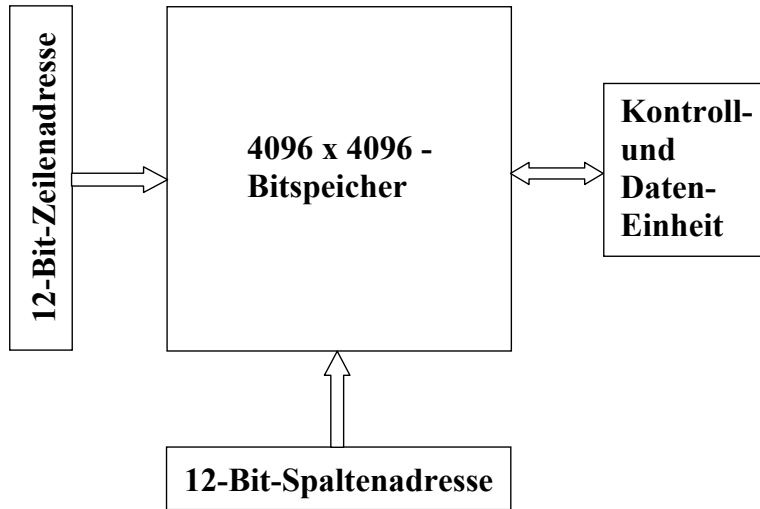
### Abkürzungen:

**SAR** = Speicheradreibregister  
**SDR** = Speicherdatenregister

### Ablauf eines Lesevorgangs:

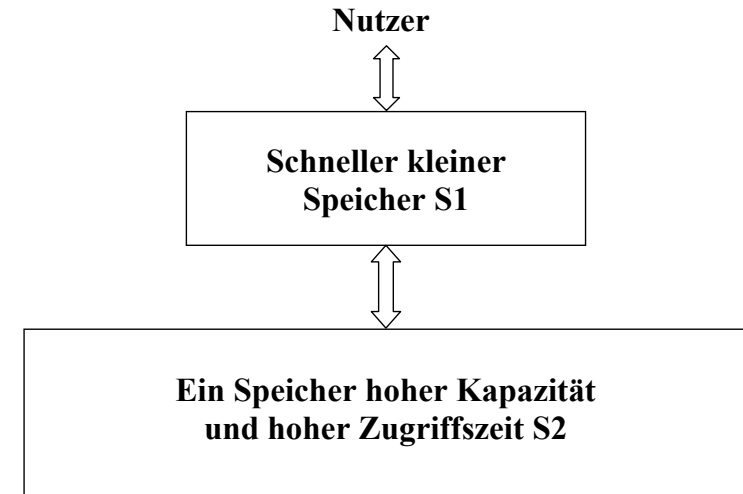
Um den Inhalt der Speicherzelle **Z** zu erhalten, schreibt der Prozessor die Adresse von **Z** in das Speicheradreibregister und signalisiert einen Lesewunsch; nach geraumer Zeit stellt der Speicher das Datum im Speicherdatenregister zur Verfügung. Der Lesevorgang erfolgt asynchron.

### Struktur eines 16M x 1 DRAM-Bausteins:



**Bemerkung:** Die von der Außenwelt gelieferte Adresse, in diesem Fall eine 24-Bit-Adresse, wird in einen Zeilenteil und einen Spaltenteil unterteilt, die beiden Adreßteile werden nacheinander aktiviert. Erst nachdem sich die Adreßsignale stabilisiert haben, kann der Inhalt einer Speicherzelle, in diesem Fall ein einziges Bit, gelesen oder geschrieben werden.

### Zwei-Stufen-Speicher:



### Drei Beispiele für Zwei-Stufen-Hierarchie:

#### Cache – Hauptspeicher:

- ◆ Verhältnis der Zugriffszeiten um 1 : 8,
- ◆ Verwaltung durch Hardware,
- ◆ transparent für Nutzer.

#### Virtueller Speicher:

- ◆ Verhältnis der Zugriffszeiten um 1 : 10000,
- ◆ Verwaltung durch Betriebssystem,
- ◆ transparent für einfachen Nutzer.

### Plattencache:

- ◆ Verhältnis der Zugriffszeiten um 1 : 10000,
- ◆ Verwaltung durch Software (evtl. durch Plattencontroller),
- ◆ transparent für einfachen Nutzer.

### Benennungen:

**Z1 = Zugriffszeit zu Information in S1**

**Z2 = Zugriffszeit zu Information in S2**

**Tq = Wahrscheinlichkeit, daß sich gesuchte Information in S1 befindet.**

### Mittlere Zugriffszeit MZ

$$= Tq * Z1 + (1 - Tq) * (Z1 + Z2)$$

$$= Z1 + (1 - Tq) * Z2$$

### Rechenbeispiel:

Seien **Z1 = 1 Zeiteinheit, Z2 = 8 Zeiteinheiten,**

**Tq = 99%, dann**

**MZ = 1,08 Zeiteinheiten.**

Sinke nun **Tq** um 4%, dann

**MZ = 1,40 Zeiteinheiten.**

Hieraus resultiert eine **Leistungseinbuße von 23%.**

**Resultat: Hohe Trefferquoten sind erstrebenswert.**

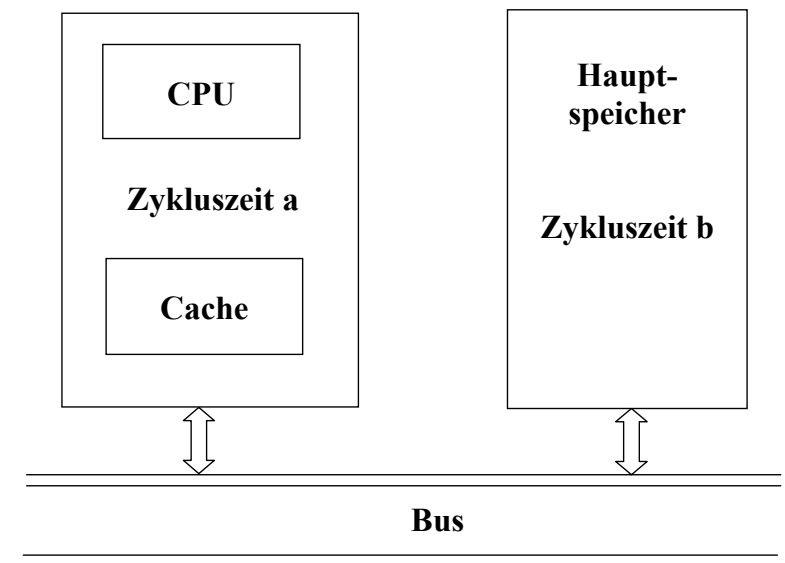
### Cache:

**Beobachtung: "Typische" Programme verbringen 90% ihrer Laufzeit in 10% des Codes.**

**Man unterscheidet zwei Arten der Lokalität:**

**Zeitliche Lokalität und  
räumliche Lokalität.**

**Ziel: Die wichtigsten Codeteile eines Programms sollten in einem schnellen Zwischenspeicher liegen.**



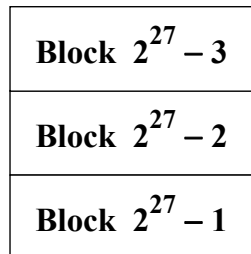
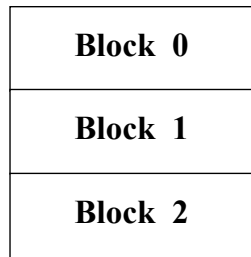
**Bemerkung: Das Verhältnis der Zugriffszeiten von CPU zu Hauptspeicher kann durchaus 1 : 100 betragen.**

## Assoziativer Cache:

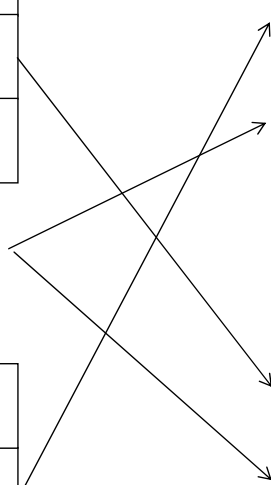
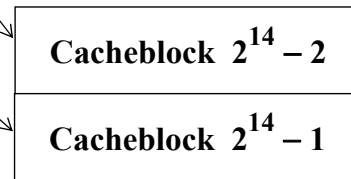
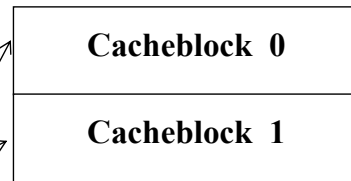
### Beispiel:

Wir betrachten einen Speicher mit  $2^{32}$  Wörtern, adressiert von 0 bis  $2^{32} - 1$ , jeweils 32 Wörter fassen wir zu einem Block zusammen. Der Cache möge ein Fassungsvermögen von  $2^{14}$  Blöcken haben.

### Hauptspeicher



### Cache-Speicher



**Bemerkung:** Jeder Cacheblock kann jeden Hauptspeicher-Block aufnehmen.

Man unterteilt die 32 Bit Speicheradresse in zwei Teile, eine 27 Bit Blockadresse, genannt Tag, und eine 5 Bit Wortadresse innerhalb eines Blockes.



Wortadresse innerhalb des Blocks

Transportiert man einen Speicherblock in den Cache, dann muß man zwecks Identifizierung auch die Blockadresse mit in den Cache aufnehmen, ferner benötigt man für jeden Cacheblock zusätzlich ein Bit, das anzeigt, ob im Cacheblock gültige Daten liegen.

### Verdrängungsstrategien:

- LRU (Least recently used),
- LFU (Least frequently used),
- FIFO (First in, first out),
- Random.

Für die ersten drei Verdrängungsstrategien benötigt man zusätzliche Verwaltungsinformation pro Cacheblock, etwa Zeitstempel, Zähler oder Ringzeiger. Simulationsstudien von A. Smith belegen, daß eine zufällige Opferwahl bei der Verdrängung durchaus konkurrenzfähig zu den anderen Strategien ist.

**Hauptnachteil des assoziativen Cache:** Aufwendige Bestimmung, ob ein Block im Cache residiert.

**Matrizenimplementation eines LRU-Algorithmus:**

**Beispiel für Zugriffsfolge:** 7, 5, 6, 2, 9, 5, 6, 7, 9, 2

**Leerer Speicher**

	1	2	3	4
0	0	0	0	0
0	0	0	0	0
0	0	0	0	0
0	0	0	0	0

**Eintrag von 7**

	1	2	3	4
7	0	1	1	1
0	0	0	0	0
0	0	0	0	0
0	0	0	0	0

**Eintrag von 5, 6**

	1	2	3	4
7	0	0	0	1
5	1	0	0	1
6	1	1	0	1
0	0	0	0	0

**Eintrag von 2, 9**

	1	2	3	4
9	0	1	1	1
5	0	0	0	0
6	0	1	0	0
2	0	1	1	0

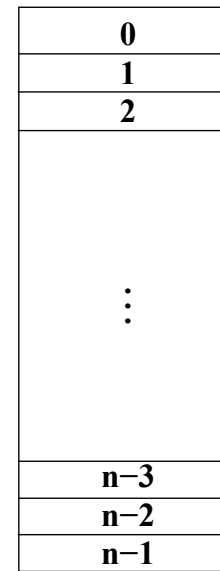
**Zugriffsfolge 5, 6, 7**

	1	2	3	4
9	0	0	0	0
5	1	0	0	0
6	1	1	0	0
7	1	1	1	0

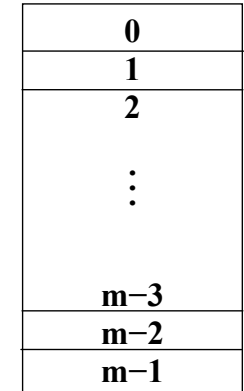
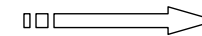
**Zugriffsfolge 9, 2**

	1	2	3	4
9	0	0	1	1
2	1	0	1	1
6	0	0	0	0
7	0	0	1	0

**Direkter Cache:**



**n-Block-Speicher**



**m-Block-Cache**

Bei der direkten Zuordnung von Speicherblöcken zu Cacheblöcken nutzt man folgende Funktion:

(Speicher-Block-Nr) modulo (Cache-Block-Zahl).

Ist  $m$  eine Zweierpotenz, dann reduziert sich diese Abbildung auf das Ausblenden der niederwertigen  $\text{ld}(m)$  Bit der binären Blockadressen des Grundspeichers.

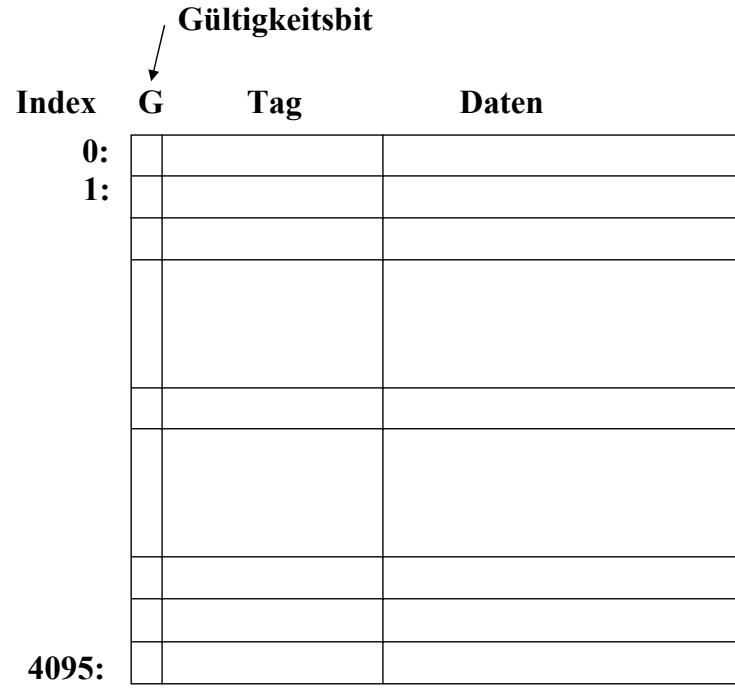
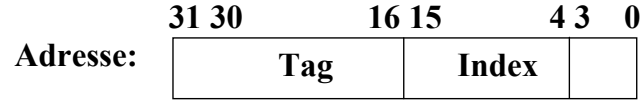
Beispiel:  $n = 2^{32}$ ,  $m = 2^{10}$ ;  
 Blockadresse = 0xB0AC87D2,  
 Cache-Adresse = 0x3D2.

**Annahmen für Bild:**

Speicheradresse: 32 Bit

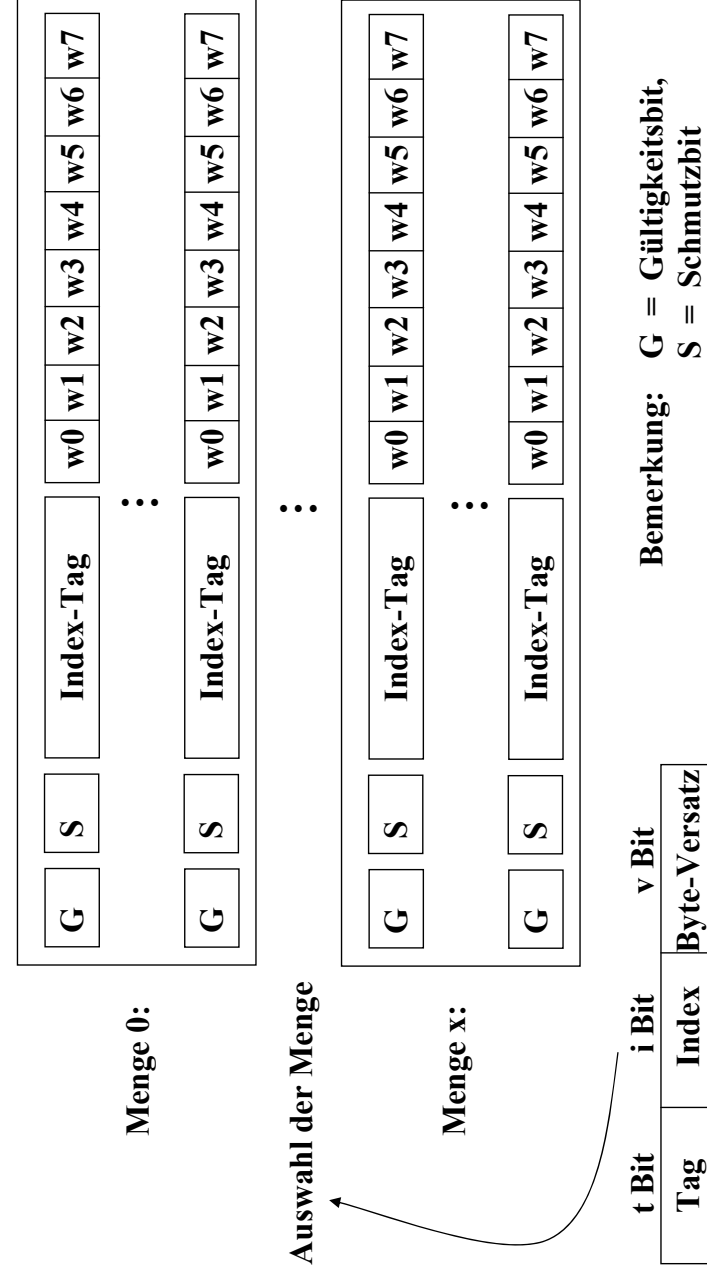
Cache-Block: 16 Speicherwörter

Cache-Kapazität: 4096 Blöcke



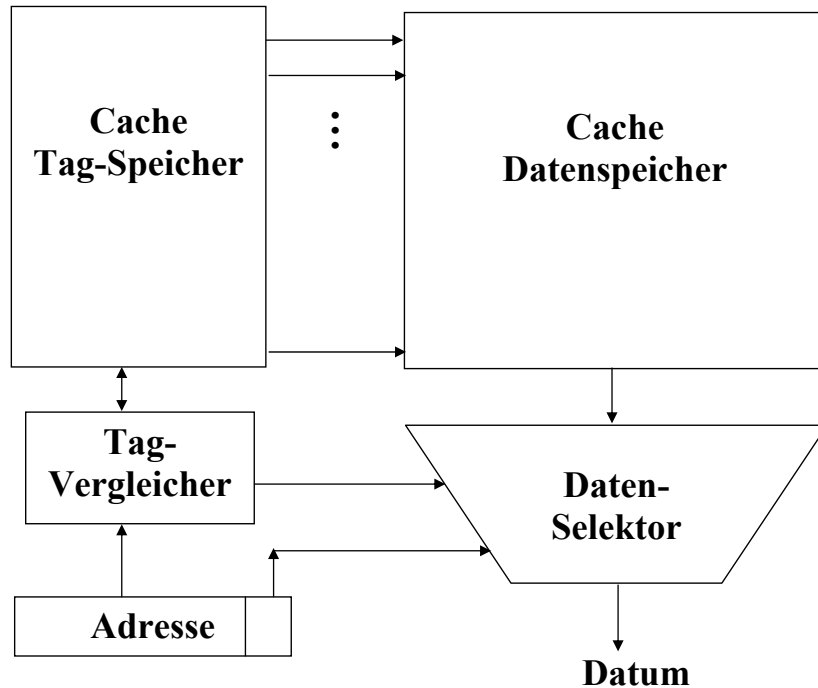
**Bemerkung:** Zur Bestimmung, ob sich ein Speicherwort im Cache befindet, genügt es, das Gültigkeitsbit und die obere Teiladresse zu prüfen.

**Bild eines mengenassoziativen Cache:**



## Zusammenfassende Bemerkungen:

### Cachespeicher-Blockbild:



- (i) Befindet sich ein Datum nicht im Cache, dann wird es aus dem Speicher nachgeladen. Während dieser Zeit ruht im allgemeinen die Abarbeitung des zugehörigen Befehls, es muß kein Prozessorzustand oder ähnliches gerettet werden.

- (ii) Da Daten und Code verschiedenes Verhalten zeigen, verwendet man gern zwei getrennte Cachespeicher, einen für Instruktionen und einen für Daten.
- (iii) Moderne Rechner verfügen über zwei oder drei hintereinandergeschaltete Cache-Speicher, genannt Level 1 Cache, Level 2 Cache und Level 3 Cache.
- (iv) Cache-Speicher werden in der Praxis gern mit niederzähligen Assoziativitätsgrad implementiert, z.B. 2-fach, 4-fach oder 16-fach assoziativ.
- (v) Beim Schreiben eines Datums in den Cache kennt man zwei grundsätzliche Strategien: "write through" und "write back". Durch Einfügen eines Zwischenpuffers läßt sich die Leistung bei "write back" steigern.
- (vi) Gedanklich befindet sich der Cache zwischen Prozessor und Hauptspeicher. Die Grundorganisationsformen sind "look-aside" und "look-through".
- (vii) Cache-Speicher werden seit 1968 eingesetzt (Rechner IBM 360/85 und IBM 360/91).

**(viii) Beispiele für Cache-Speicher:**

**PowerPC 601:**

**Größe 32 KiB, 8-fach assoziativ, 128 Mengen von 32 Byte Blöcken.**

**AMD-K6-III:**

**L1 Cache: 64 KiB,**

**L2 Cache: 256 KiB, 4-fach assoziativ,**

**L3 Cache: 512 KiB - 2048 KiB.**

**AMD Opteron:**

**L1 Cache: 64 KiB I-Cache, 2-fach assoziativ**

**64 KiB D-Cache, 2-fach assoziativ**

**L2 Cache: 1024 KiB, 16-fach assoziativ**

**DEC Alpha 21164:**

**L1 Cache: 8 KiB I-Cache und 8 KiB D-Cache,  
direkter Cache, 32 Byte Blöcke;**

**L2 Cache: 96 KiB, 3-fach assoziativ,  
32 Byte Blöcke**

**L3 Cache: 1 MiB, direkt, 32 Byte Blöcke.**

**VAX 11/780:**

**Größe 8 KiB, 8 Byte Blöcke, 2-fach assoziativ,  
Zufallersetzung, write-through**

**Cache Hit: ein Zyklus**

**Cache Miss: sechs Zyklen für den Speicherzugriff  
+ ein Zyklus für Cache Hit.**

**(ix) Lese- und Schreibprotokolle für Cachespeicher:**

**Lesen:**

**Daten im Cache:**

**Daten werden ausgelesen.**

**Daten nicht im Cache:**

**(a) Daten werden in den Cache gelesen,  
dann weitergeleitet.**

**(b) "Load Through", Daten werden parallel in  
den Cache gelesen und weitergeleitet.**

**(c) "Requested Word First", das angeforderte  
Datum wird sofort geliefert, der Rest in  
zyklischer Folge.**

**Schreiben:**

**"Daten" im Cache:**

**(a) "Write Through", Daten werden parallel in  
den Cache und Hauptspeicher geschrieben.**

**(b) "Write Back", Daten werden nur im Cache  
geändert, später in den Hauptspeicher  
geschrieben.**

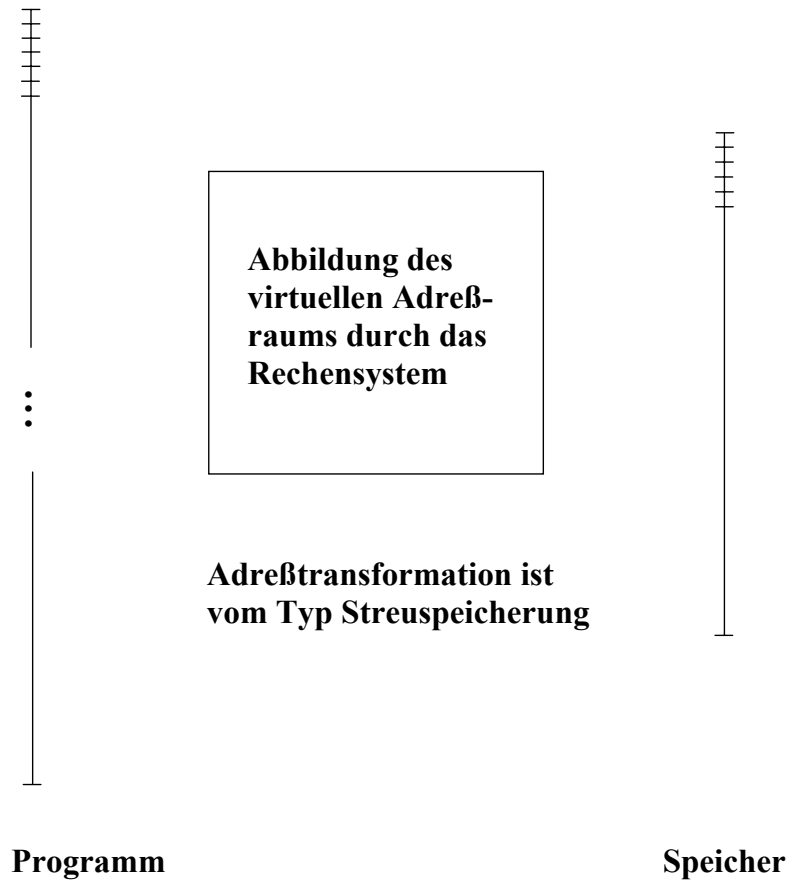
**"Daten" nicht im Cache:**

**(a) "Write Allocate": Daten werden in den  
Cache und Hauptspeicher geschrieben.**

**(b) Daten werden nur in den Hauptspeicher  
geschrieben.**

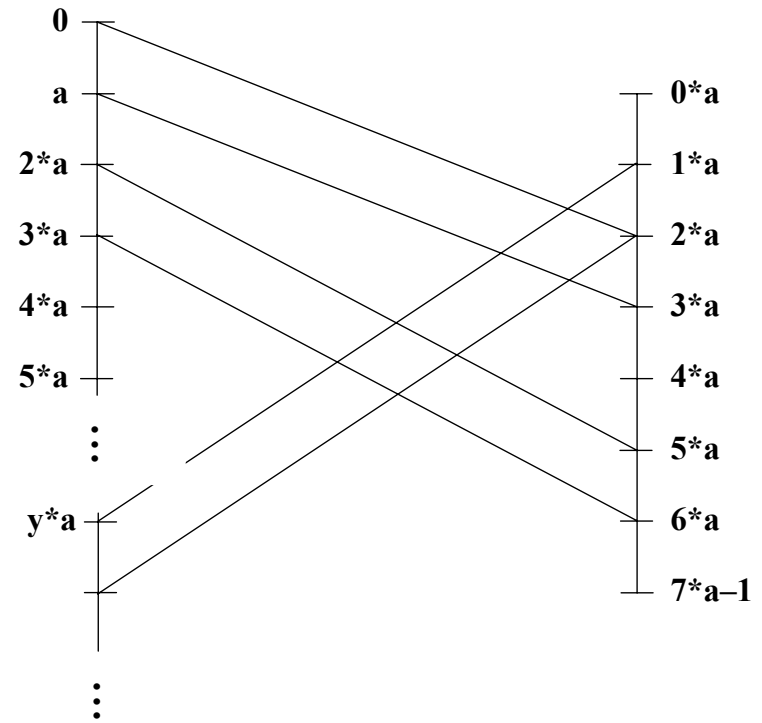
**Beobachtung:**

Ein großer homogener linearer Adreßraum erleichtert die Programmierung.



**Virtueller Adreßraum**

**Realer Adreßraum**



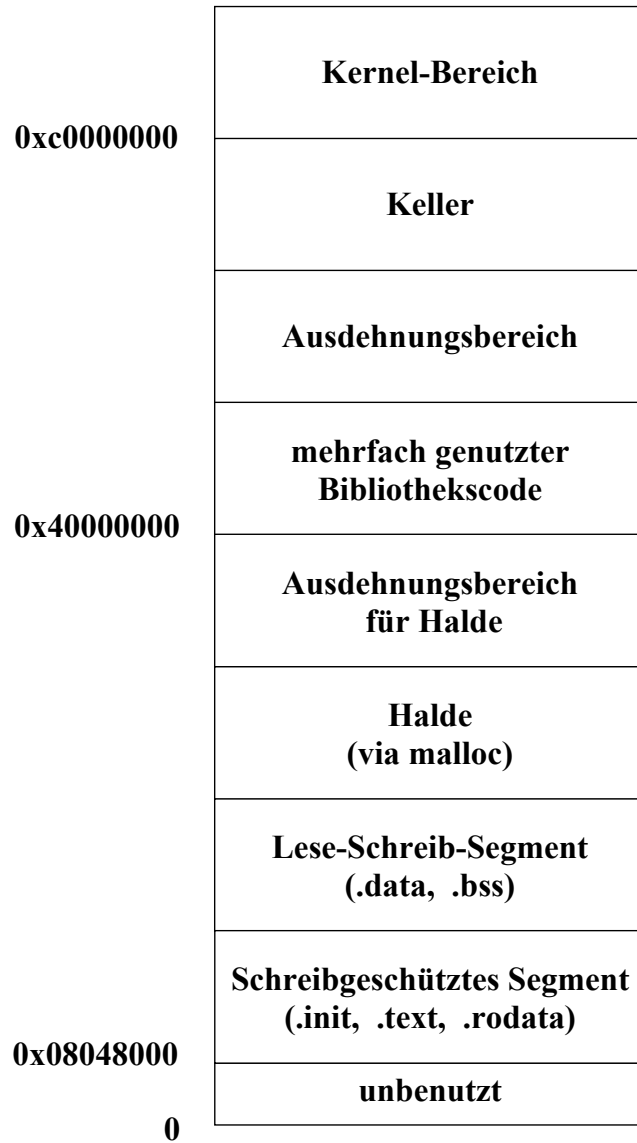
$a =$  Größe einer Seite

z. B.: Store R3, 40193

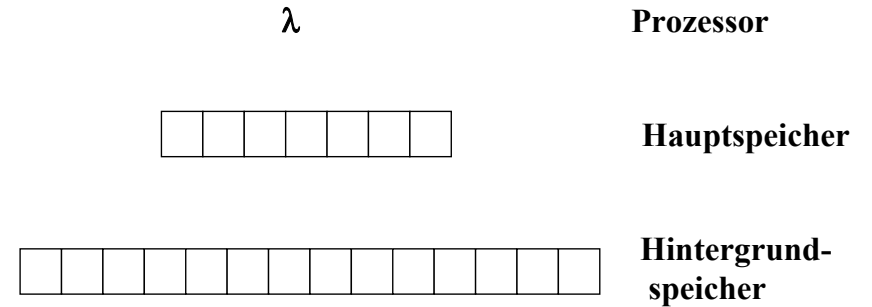
Adreßzuordnung:

virtuell	real
[ 0, a)	[2*a, 3*a)
[2*a, 3*a)	[5*a, 6*a)
⋮	⋮
[y*a, (y+1)*a)	[ a, 2*a)

**Speicherauslegung eines Linux-Prozesses:**



**Zweistufige Speicherhierarchie:**



**Bezeichnungen:**

**Kachelmenge**  $K = \{k_1, k_2, \dots, k_c\}$

**Seitenmenge**  $N = \{n_1, n_2, \dots, n_s\}$

**Speicherzustand**  $S = \{n \in N / n \text{ ist genau eine Kachel zugeteilt}\}$

**Folge von Zeitpunkten**  $T = t_0, t_1, t_2, \dots$

**Referenzkette der Länge L**  $\omega = r_0, r_1, r_2, \dots, r_{L-1}$

**Vorwärtsdistanz:**

$$a(t, x) = \begin{cases} k, & \text{falls } r_{t+k} \text{ erstes Auftreten von } x \text{ in} \\ & r_{t+1}, r_{t+2}, r_{t+3}, \dots \\ \infty & \text{sonst} \end{cases}$$

**Rückwärtsdistanz:**

$$b(t, x) = a(t, x) \text{ in } r_{t-1}, r_{t-2}, \dots, r_0.$$

Belady bewies, daß folgender Seitenersetzungsalgorithmus optimal ist: Bei Bedarf wird eine Seite mit maximaler Vorwärtsdistanz aus dem Kachelspeicher verdrängt.

Beispiel:

T:											1	1	1	1	1	1
	0	1	2	3	4	5	6	7	8	9	0	1	2	3	4	5
R:	a	b	c	d	a	b	c	d	a	b	c	d	e	f	g	h
A:	*	*	*	*			*			*			*	*	*	*
KS:	a	a	a	a	a	a	a	a	a	b	b	b	e	e	e	h
		b	b	b	b	b	c	c	c	c	c	c	c	f	f	f
			c	d	d	d	d	d	d	d	d	d	d	d	d	g

Bezeichnungen:

- T = Folge von Zeitpunkten
- R = Referenzkette
- A = Alarm
- KS = Kachelspeicher der Größe 3

Bemerkung: Zur Realisierung von Näherungen zu Beladys Algorithmus benötigt man umfangreiche Hardwareunterstützung, mindestens aber ein Referenzbit pro Seite.

Ablauf eines Seitenalarms bei "Demand Paging" bei vollem Kachelspeicher:

Bei Ausführung eines Befehls erfolgt ein Seitenalarm.

Der zugehörige Prozeß wird unterbrochen.

Die fehlende Seite wird durch den Speicher-  
verwalter lokalisiert.

Eine zu verdrängende Seite wird bestimmt und ihr  
Inhalt gerettet.

Die angeforderte Seite wird in die nun freigewor-  
dene Kachel geladen.

Die Seitentabelle wird aktualisiert.

Der unterbrochene Prozeß ist wieder lauffähig.

Bemerkungen:

- (i) Ein Befehl kann mehr als einen Seitenalarm auslösen.
- (ii) Man unterscheidet zwischen prozeßlokaler und globaler Seitenersetzungsstrategie.

**Realisierung der Adreßumsetzung über ein Tabellenwerk:**

virtuelle Adresse:



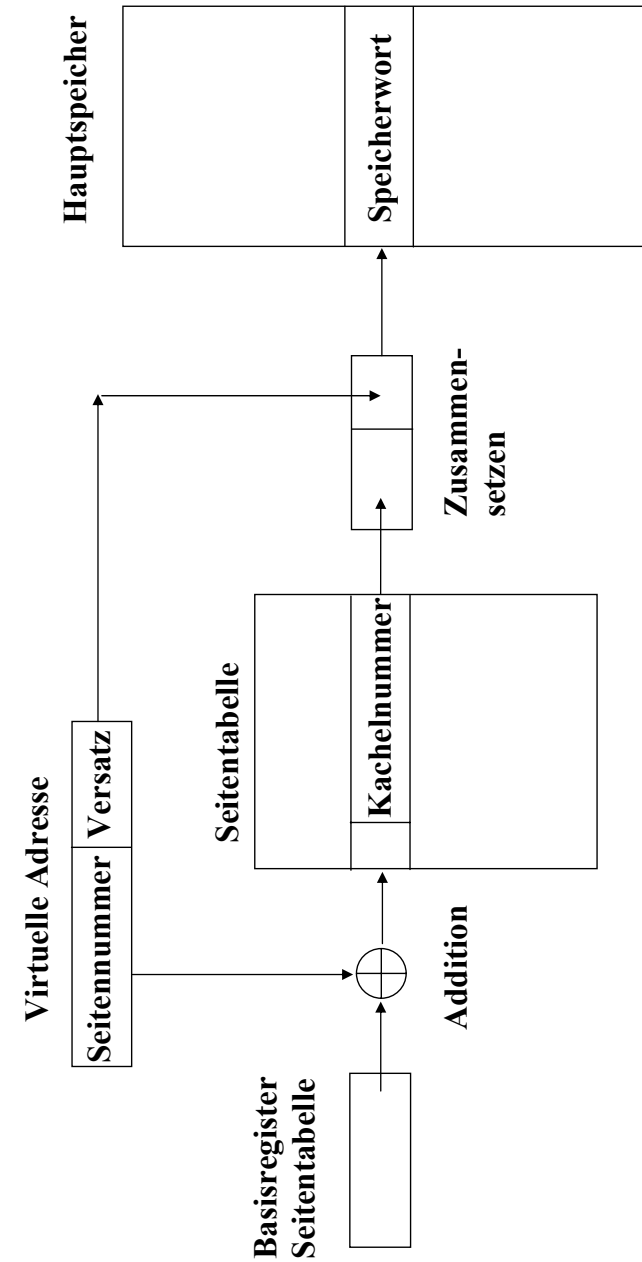
Seitennummer	Kachelnummer
0	5093
1	87
2	12321
3	nicht zugeordnet
4	nicht zugeordnet
5	763
⋮	⋮



physikalische Adresse:



**Einstufige Adreßumsetzung im Seitenverfahren:**



## Geschwindigkeitsbetrachtung für Seitenverfahren:



Rechenwert für Hauptspeicherzugriff: 100 ns  
Rechenwert für Hintergrundspeicherzugriff: 10 ms

Annahme: Nur 1% der Zugriffe erfolgen auf den Hintergrundspeicher.

Durchschnittliche Zeit für einen Zugriff

$$\begin{aligned} dz &= 0,99 * 10^2 \text{ ns} + 0,01 * 10^7 \text{ ns} \\ &\approx 10^5 \text{ ns} \\ &= 10^3 * 100 \text{ ns} \end{aligned}$$

Frage: Sind Seitenverfahren überhaupt sinnvoll?

## Platzproblem für Adreßumsetztabelle:

Fall 1:

Virtuelle Adresse: 16 Bit  
Distanzadresse: 9 Bit  
Adreßumsetztabelle: 128 Einträge

Ein gut zu verwaltende Größe.

Fall 2:

Virtuelle Adresse: 32 Bit  
Distanzadresse: 12 Bit  
Adreßumsetztabelle:  $2^{20}$  Einträge

Bei 4 Byte pro Tabelleneintrag benötigt man pro Prozeß eine 4 MiB große Tabelle. Diese läßt sich noch im Hauptspeicher unterbringen.

Fall 3:

Virtuelle Adresse: 64 Bit  
Distanzadresse: 12 Bit  
Adreßumsetztabelle:  $2^{52}$  Einträge

Dies ist größer als jeder bisher genutzte Speicher.

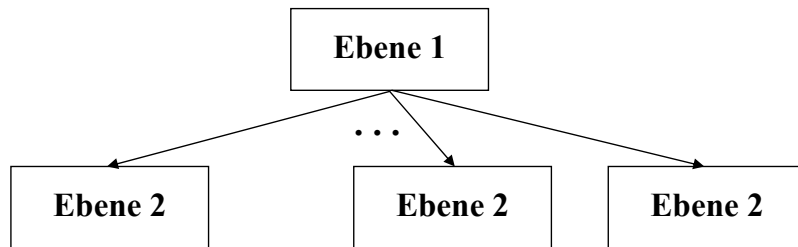
**Möglichkeiten dem Platzproblem für Seitentabellen zu begegnen:**

**(i) Begrenzungsregister für Seitentabellen:**

Da nur selten ein Prozeß den gesamten virtuellen Adreßraum nutzt, genügt es, nur den Anfang des Adreßraums bei der Speicherverwaltung zu berücksichtigen. Dies führt in modernen Systemen, in denen man mindestens die vier Prozeßsegmente Code, Daten, Keller und Halde unterscheidet, zu Schwierigkeiten.

**(ii) Baumstrukturierung der Seitentabellen:**

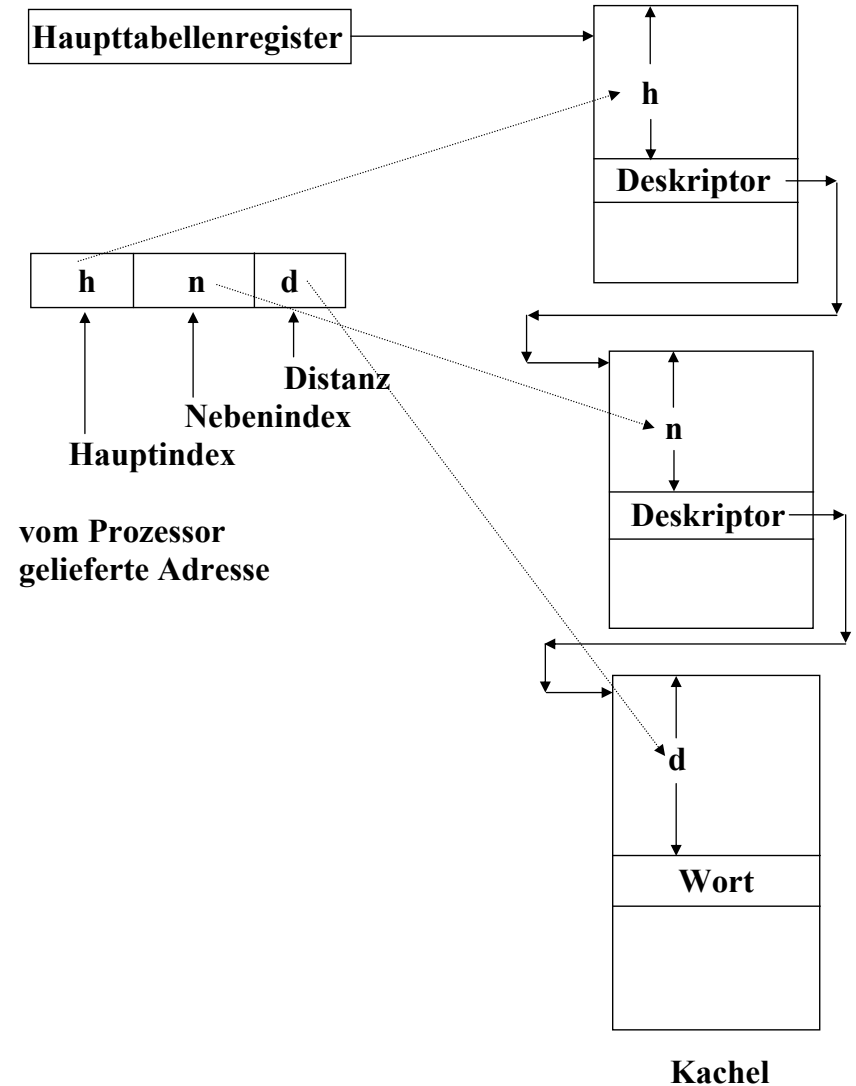
Man kennt Adressierungsbäume mit zwei, drei und vier Ebenen.



**(iii) Invertierte Seitentabellen:**

In diesem Fall benötigt man nur für jede dem virtuellen Mechanismus zur Verfügung gestellte Kachel einen Eintrag. Das Suchproblem löst man über eine gute Streufunktion.

**Beispiel eines zweistufigen Tabellenwerks zur Adreßumsetzung:**



**Invertierte systemweite Seitentabelle:**

virtuelle Adresse:

Seitennummer <i>s</i>	Distanz <i>d</i>
-----------------------	------------------

Streifunktion berechnet eine erste Kachelnummer.

Kachelnr.	Seitennr.	Prozeßinfo.	Verkettung
0	137		
1	20201		
2	56		
3			
kkk			→
			↕
nnn	xxx		← NIL
kmax			

**Zeitproblem der Adreßumsetzung:**

**Annahme:** Die Adreßumsetzungstabelle liegt im Hauptspeicher.

Bei einer dreistufigen Adreßumsetzung wird die Zahl der Speicherzugriffe vervierfacht. Dies führt in etwa zu einer Vervierfachung der Ablaufgeschwindigkeit der Programme.

**Abhilfe:** Einsetzung eines Assoziativspeichers.

Virtuelle Adresse	Kacheladresse
14327	39
4151	2
217306	24

**Bemerkung:** Assoziativspeicher sind klein, etwa 16, 32, 128 Einträge.

**Beispiele zur Größe des TLB:**

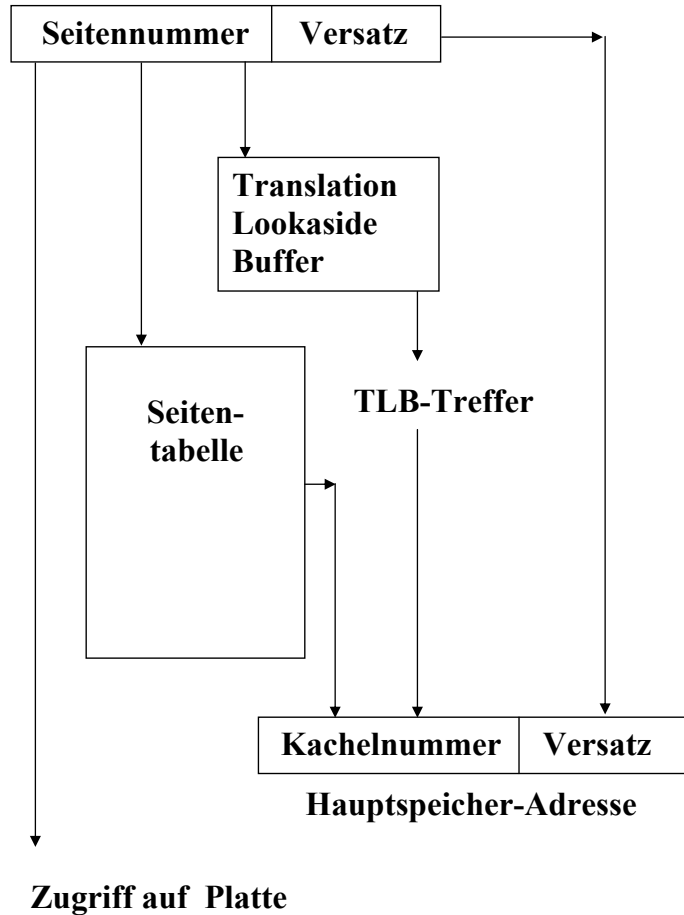
NS32532-25:	64 Einträge
Toshiba TMPR4955	48 Einträge
HP PA-8000	96 Einträge
HP PA-8200	120 Einträge

**Bemerkung:** TLB = Translation Lookaside Buffer

## Seitenverfahren:

Einfache Adreßumsetzung mit  
Adreßumsetzbeschleuniger:

Virtuelle Adresse



## Verdrängungsstrategien:

Situation: Ein Programm,  
ein fester Kachelspeicher.

Optimale Strategie bekannt: Verdränge Seite mit größter  
Vorwärtsdistanz.

### Näherungen:

#### 1. RANDOM:

Zufällig ausgewählte Seite wird verdrängt.

#### 2. FIFO:

Älteste Seite im Kachelspeicher wird verdrängt.

#### 3. LFU ("Least Frequently Used"):

Eine Seite mit den wenigsten Zugriffen wird verdrängt; Näherungsimplementation erfolgt über Benutztzähler.

#### 4. LRU ("Least Recently Used"):

Eine Seite mit maximaler Rückwärtsdistanz wird verdrängt; dies läßt sich mittels eines Kellers implementieren.

## 5. RNU ("Recently Not Used"):

Eine Seite, die im letzten Beobachtungsintervall nicht referiert wurde, wird verdrängt.

## 6. SECOND CHANCE:

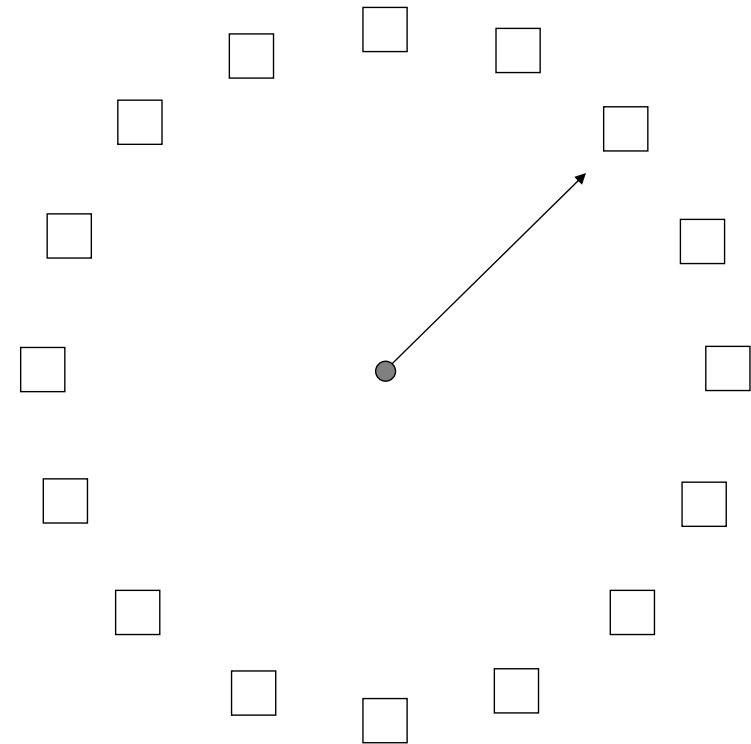
Dies Verfahren ist eine Modifikation des FIFO-Algorithmus; die Kacheln werden zirkulär verkettet; die erste Kachel, deren Benutzbit nicht gesetzt ist, wird verdrängt; während der Suche werden die Benutzbit gelöscht.

## 7. Schleifenentdecker (nur historisch):

Zu jeder Seite  $s$  führt man die letzte Zugriffsdistanz  $zd(s)$ , die Zahl der Zugriffe zwischen dem letzten und dem vorletzten Zugriff auf  $s$ . Nun berechnet man für jede Seite die Differenz  $D(s)$  zwischen Rückwärtsdistanz und letzter Zugriffsdistanz. Ist  $D(s) > 0$ , dann liegt die Seite  $s$  nicht in einer engen Schleife. Man verdrängt eine Seite maximalen  $D$ -Wertes, falls ein  $s$  mit  $D(s) > 0$  existiert. Gilt für alle  $s$   $D(s) < 0$ , dann wird eine Seite mit maximalem  $zd(s)$  ersetzt.

**Bemerkung:** Das Verfahren Second Chance trägt auch den Namen Uhralgorithmus.

**Illustration:**



**Bemerkung:** Dieser Algorithmus besitzt eine sehr einfache Implementation.

**Beispiele zu Seitenverfahren:**

**Referenzkette:**      a b a c d a e d b a d a

**Kachelspeicher-  
größe = 3  
FIFO**

```

a a a a b c d d a a e b
b b b c d a a e e b d
c d a e e b b d a
* * * * *
    
```

**Kachelspeicher-  
größe = 3  
Second Chance**

```

a a a a a a a a a a a
b b b d d d d d d d d
c c c e e b b b b
* * *
    
```

**Kachelspeicher-  
größe = 3  
OPT**

```

a a a a a a e e e a a a
b b b b b b b b b b b
c d d d d d d d d
* * *
    
```

**Kachelspeicher-  
größe = 3  
LRU**

```

a b a c d a e d b a d a
a b a c d a e d b a d
b a c d a e d b b
* * * *
    
```

**Kriterien für Verdrängungsstrategie:**

1. Einfache robuste Implementation.
2. Freiheit von erratischem Verhalten.

**Bemerkung:**

Bei der Implementation lassen sich Sondersituationen berücksichtigen, wie:

- (i) Seiten, die nicht modifiziert wurden, werden bevorzugt verdrängt.
- (ii) Ein Teil der Seiten wird dem Verdrängungsalgorithmus entzogen.

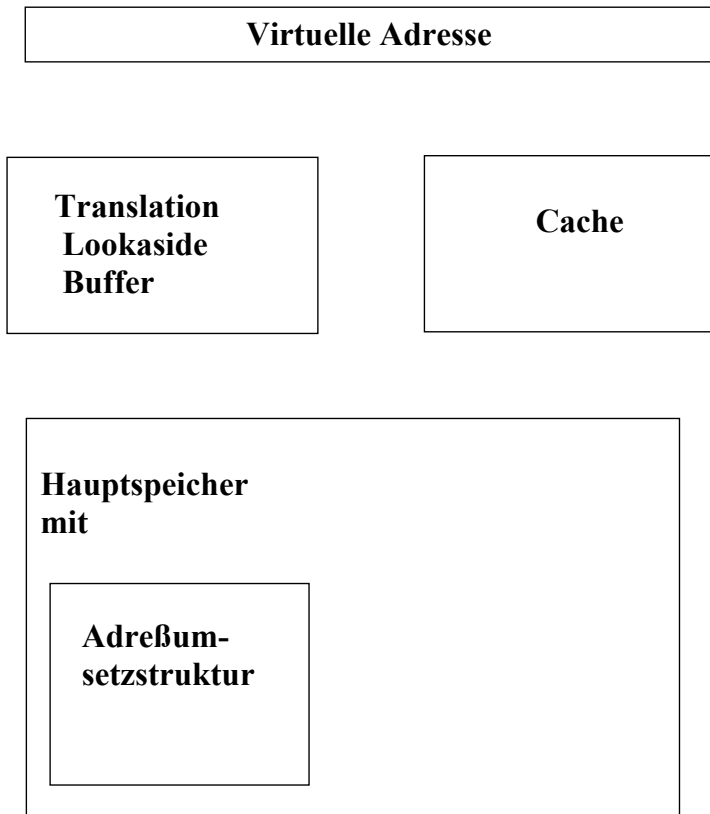
**Beispiel von Belady zum erratischen Verhalten des FIFO-Algorithmus:**

												1	1	
<b>Zeitpunkte:</b>	0	1	2	3	4	5	6	7	8	9	0	1		
<b>Referenzkette:</b>	a	b	c	d	a	b	e	a	b	c	d	e		
<b>Alarm:</b>	*	*	*	*	*	*	*			*	*			
<b>Kachelspeicher der Größe 3:</b>	a	b	c	d	a	b	e	e	e	c	d	d		
		a	b	c	d	a	b	b	b	e	c	c		
			a	b	c	d	a	a	a	b	e	e		

													1	1
<b>Zeitpunkte:</b>	0	1	2	3	4	5	6	7	8	9	0	1		
<b>Referenzkette:</b>	a	b	c	d	a	b	e	a	b	c	d	e		
<b>Alarm:</b>	*	*	*	*			*	*	*	*	*	*		
<b>Kachelspeicher der Größe 4:</b>	a	b	c	d	d	d	e	a	b	c	d	e		
		a	b	c	c	c	d	e	a	b	c	d		
			a	b	b	b	c	d	e	a	b	c		
				a	a	a	b	c	d	e	a	b		

**Bemerkung:** In diesem Beispiel führt eine Vergrößerung des Kachelspeichers zu einer Erhöhung der Seitenalarme.

**Seitenverfahren und Cache:**



**Mögliche Anordnungen:**

- (a) Der Cache arbeitet mit virtuellen Adressen.
- (b) Der Cache arbeitet mit physikalischen Adressen.
- (c) Der Cache arbeitet teilweise mit virtuellen und teilweise mit physikalischen Adressen.