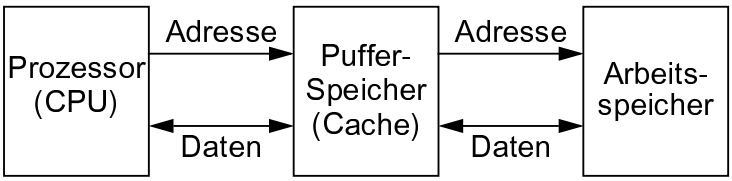
2 In Hardware realisierte Pufferspeicher

C. Schimmel: "UNIX Systems for Modern Architectures", Addison-Wesley Publishing Company, 1994.

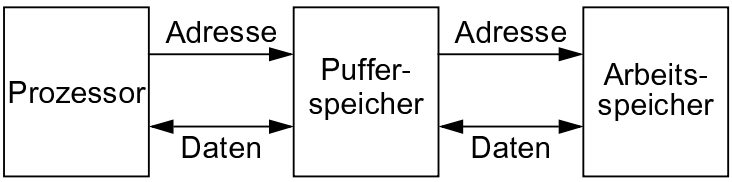
2.1 Monoprozessoren einschließlich E/A

2.1.1 Grundlagen

Zugriff



- **Fehlzugriff (cache miss)**
- **Treffer (cache hit)**
- **Trefferrate (hit ratio)**
- **Puffer-Zeile (cache line)**



Virtuelle oder physikalische Adressen?

Von großem Einfluß auf die Software

Entscheidung nicht revidierbar, da Eigenschaft des Prozessors

Suchen im Pufferspeicher

Durch Angabe des Speicherortes

Assoziativ unter Verwendung von Hash-Techniken

Ersetzungsstrategien

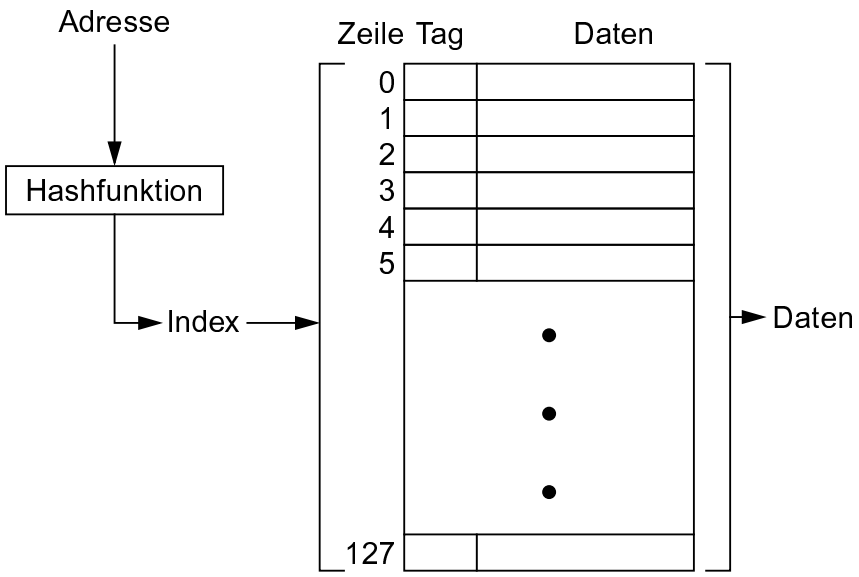
Typischerweise LRU oder Approximationen von LRU, aber auch zufällige Auswahl

Schreibstrategien

- Treffer
  - **write-through**: Arbeitsspeicher immer auf dem neuesten Stand, zuweilen unnötige Verlangsamung
  - **write-back** (auch *copy-back* genannt): Schreibt nur in den Cache; als Folge entsteht ein zeitweise inkonsistenter Arbeitsspeicherinhalt; erfordert besondere Maßnahmen beim Laden von Cachezeilen.
- Fehlzugriff
  - **write-allocate**: Erst (soweit erforderlich) aus Arbeitsspeicher lesen, dann schreiben mit einer der beiden vorangehenden Strategien.
  - **write-to-memory** (auch *write-nonallocating* genannt): Modifikation wird nur im Arbeitsspeicher vorgenommen.

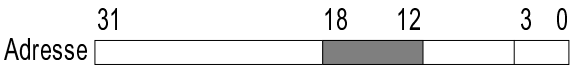
Direkt abgebildete Pufferspeicher

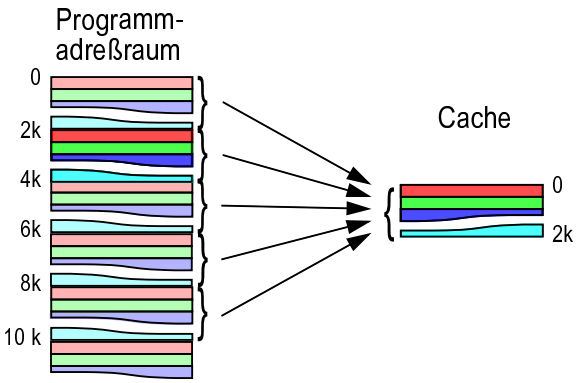
(***direct mapped*, *one-way set associative***)

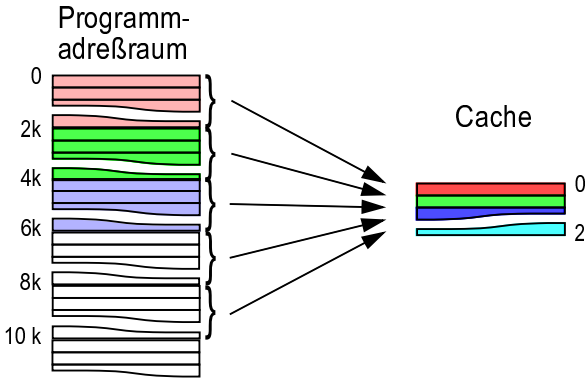


Hash-Algorithmen

- **Modulo-Hashing, z. B. Bits 0 - 3 zur Auswahl des Bytes innerhalb einer Zeile, Bits 4 - 10 zur Auswahl der Zeile, Bits 11 - 31 sind Tag**
- **Hashing durch Ausschnitt aus der Adresse, z. B. Bits 12 - 18 als Zeilenadresse**



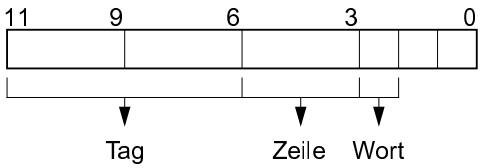




Beispiel: Intel i860

- **12 Bit breite Adressen**
- **Worte zu 4 Byte, ausgerichtet an 4-Byte-Grenze**
- **Cache mit 8 Zeilen; 8 Bytes pro Cache-Zeile**

Aufteilung der Adresse



Adressbeispiele

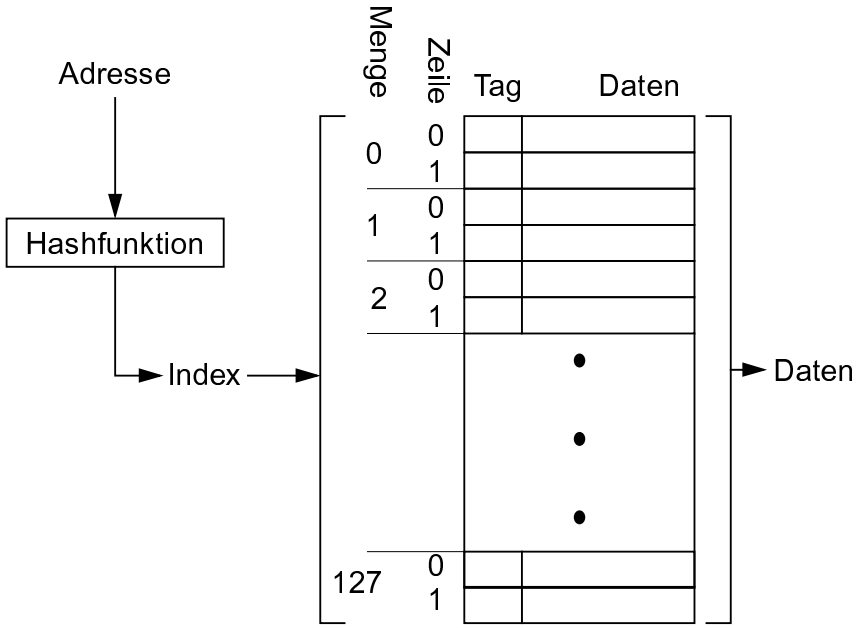
Adresse	Tag	Zeilen-index	Wort	Ergebnis-daten
01234	012	3	1	0777
01230	012	3	0	052
00130	001	3	0	miss
03574	035	7	1	0
06540	065	4	0	miss

Cache

Zeile	Tag	Daten	
0	---		
1	---		
2	---		
3	012	052	0777
4	---		
5	---		
6	---		
7	035	067	0

Zweifach assoziative Pufferspeicher

(*two-way set associative*)



Beispiel: Daten-Cache des Pentium

Vollständig assoziative Pufferspeicher (fully associative)

Gesamter Pufferspeicher wird behandelt wie eine Menge

Leeren des Pufferspeichers (*cache flushing*)

Alle Realisierungen sehen Möglichkeit zur zwangsweisen Leerung des gesamten Pufferspeichers oder einzelner Zeilen vor

- Aktualisierung des Arbeitsspeichers
- Invalidierung des Pufferspeichers

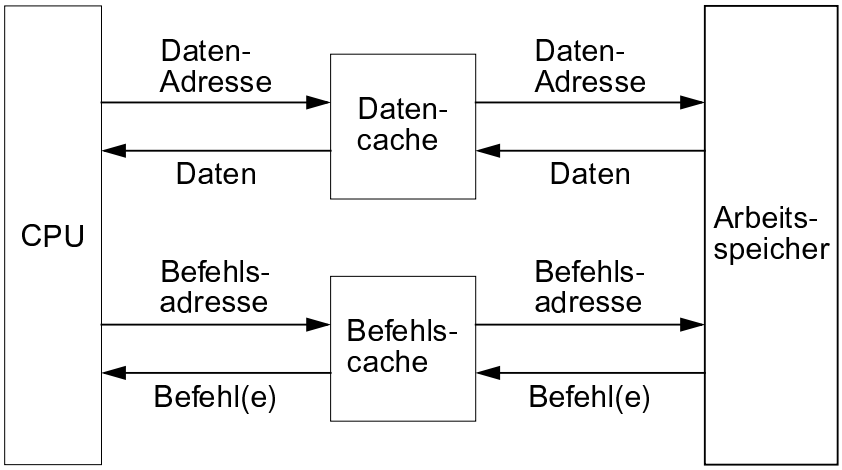
Wird benötigt zur Konsistenzsicherung

Ungepufferte Operationen

Meist wählbar auf Seitenbasis durch geeigneten Indikator in der Seiten-Kachel-Tabelle

Wichtig für Speicherbereiche, die sich unabhängig von Schreibaufrufen ändern können (*volatile* in C und C++), z. B. in den Speicher abgebildete Statusregister von E/A-Geräten

Getrennte Pufferspeicher für Daten und Befehle



Auslegung von Pufferspeichern

Zeitliche Lokalität spricht für write-back

Kleiner Pufferspeicher spricht für mehrfach-assoziative Vorgehensweisen mit großen Mengen

Adreßlokalität spricht für große Zeilenlänge

Erreichbare Leistungssteigerung durch Cache wird wesentlich vom Betriebssystem beeinflußt

Charakteristika der Realisierung von Pufferspeichern

Zeilenzahl

Zeilengröße

Mengengröße

Nutzung von ‘write-allocate’

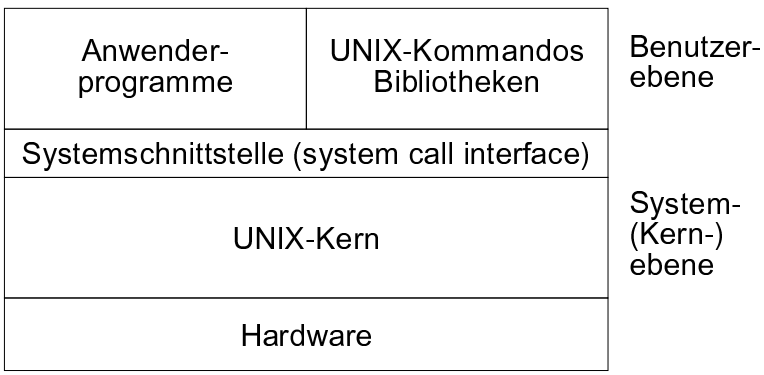
Ersetzungsstrategie

Aufsuchen mit virtueller oder physikalischer Adresse

Kennzeichnung der Zeilen

‘write-through’ oder ‘write-back’

Benutzerebene - Systemebene



Programm

- Enthält auszuführende Befehlsfolge
- Beschreibt Datenbereiche

Prozeß

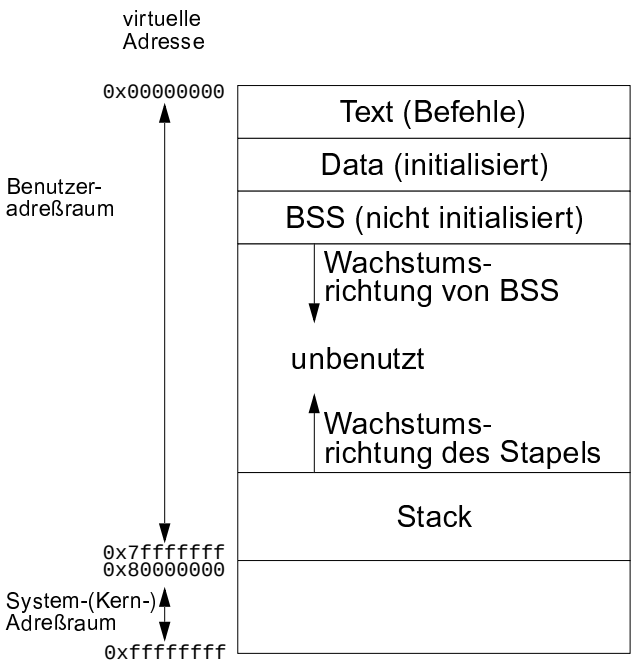
- Ausführung eines Programms
- Besteht aus Programm, Datenbereich, Hardwarezustand und Adreßraum

Faden (Thread)

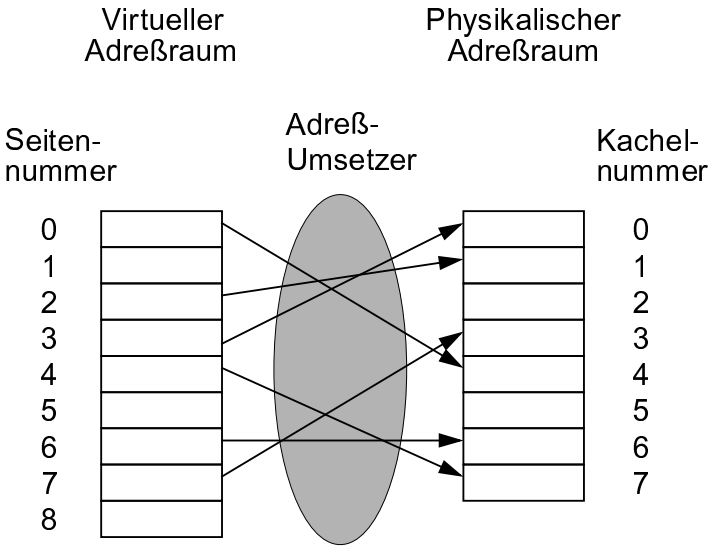
- Sequentielle Befehlsausführung in einem Prozeß
- Besteht aus Programm, Datenbereich, Hardwarezustand
- Wird in einer - evtl. mehreren Fäden gemeinsamen - Ausführungsumgebung abgearbeitet, die auch den Adreßraum stellt

Der Adreßraum von Prozessen

- Jeder Prozeß besitzt eigenen Adreßraum



- **Adreßraumabbildung**



Kontext-Umschaltung

Sichern und Neueinstellen

- **der Prozessorregister**
- **des Befehlszählers**
- **des Kellerzeigers**
- **der Adreßraumabbildung**

Thread-Umschaltung

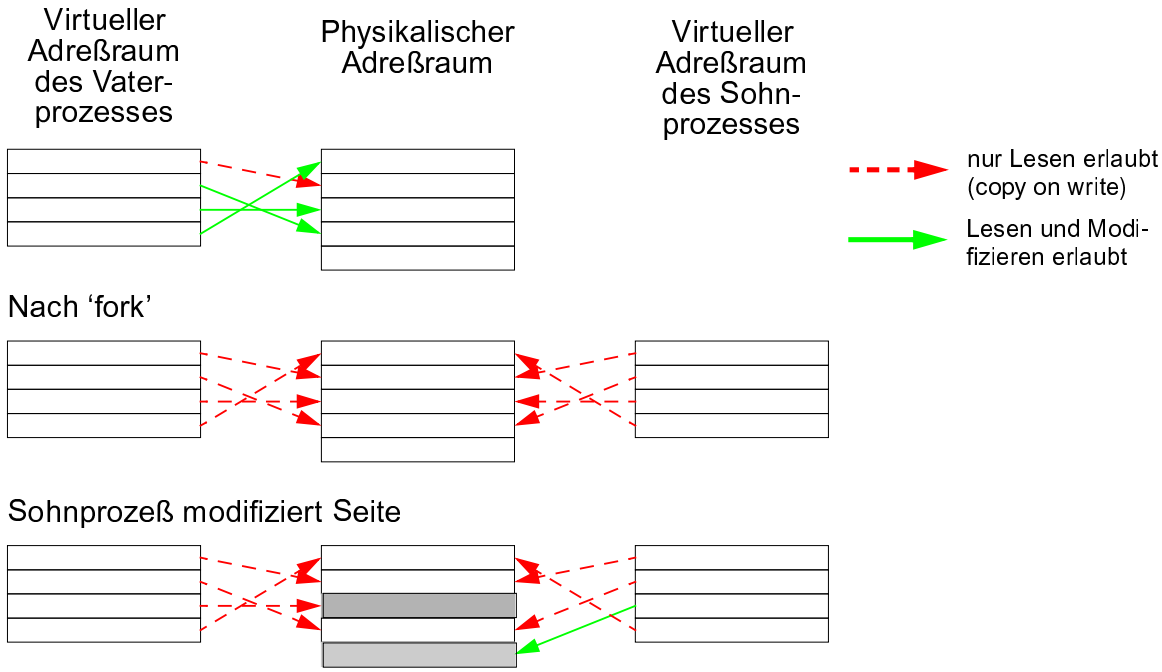
Sichern und Neueinstellen

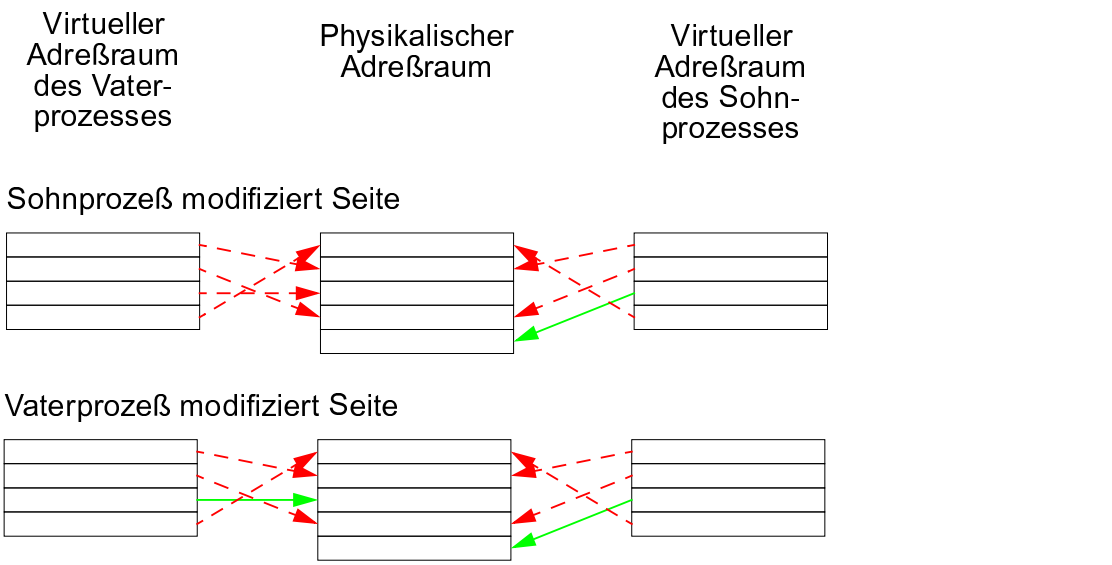
- **der Prozessorregister**
- **des Befehlszählers**
- **des Kellerzeigers**

Aber Beibehaltung der Adreßraumabbildung

Systemaufrufe zur Speicher- und Prozeßverwaltung

Der ‘fork’-Aufruf unter Benutzung von ‘copy on write’





Der ‘exec’-Aufruf

Ändert den Programmteil eines Prozesses, die Prozessorregister, den Befehlszähler und die Adreßraumabbildung.

Behält den Zustand bzgl. der meisten Unix-Dienste bei, wie geöffnete Dateien, ‘home directory’, ...

Der ‘exit’-Aufruf

Bewirkt zusammen mit der Tilgung eines Prozesses die Freigabe seines Adreßraums und das Abkoppeln von Unix-Diensten, z. B. durch Schließen seiner geöffneten Dateien.

Überführung in den Zombie-Zustand

Der ‘sbrk’- und ‘brk’-Aufruf

Verändern die Größe des BSS-Segmentes (block started by symbol: Assembler-Anweisung der IBM 7090).

sbreak: Parameter gibt Veränderung der Segmentlänge an

brk: Parameter gibt die kleinste, von BSS nicht mehr benutzte virtuelle Adresse an

Gemeinsamer Speicher (shared memory)

- **Einrichtung, indem in mehreren Adreßraum-Abbildungen Verweise auf die gleiche Kachel eingetragen werden, möglicherweise unter unterschiedlichen virtuellen Adressen.**
- **Wird vor allem zur schnellen Kommunikation zwischen Prozessen genutzt.**

- **Systemaufrufe:**

shared memory operations

```
char *shmat( int shmid, char *shmaddr,
             int shmflg)
int shmdt(char *shmaddr)
```

shared memory control operations

```
int shmctl (int shmid, int cmd,
            struct shmid_ds *buf)
```

get shared memory segment identifier

```
int shmget(key_t key, int size, int shmflg)
```

E/A-Operationen

gepuffert - ungepuffert

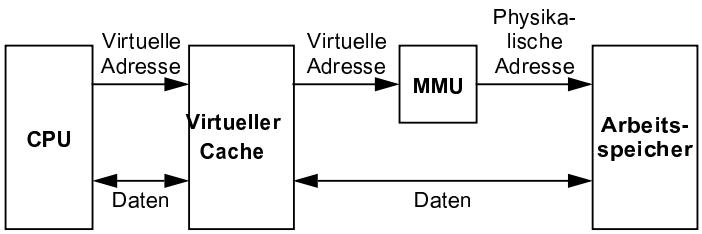
‘memory mapped files’

Verwendbar wie Felder eines Prozesses

Wirken logisch ähnlich ‘shared memory’

2.1.2 Virtuelle Pufferspeicher

2.1.2.1 Die Arbeitsweise



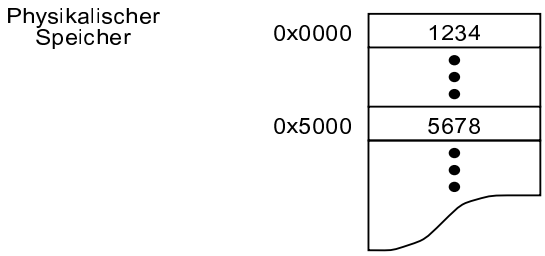
Beispiel: i860 on-chip cache

Prüfung der Zugriffsrechte bei write

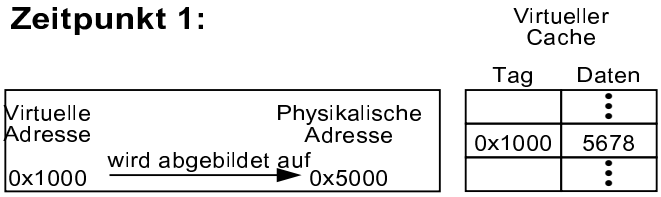
- **write-through**
  - **hit: virtuelle Adresse auch an MMU zur Prüfung**
  - **miss: bei write-allocate mit physikalischer Adresse aus ASP lesen**
- **write-back**
  - **miss: Prüfung verbunden mit Laden evtl. Rückschreiben unter Nutzung der MMU**
  - **hit: Falls schon modifiziert, dann erlaubt**
    falls nicht modifiziert, parallel Prüfung mittels MMU

2.1.2.2 Probleme

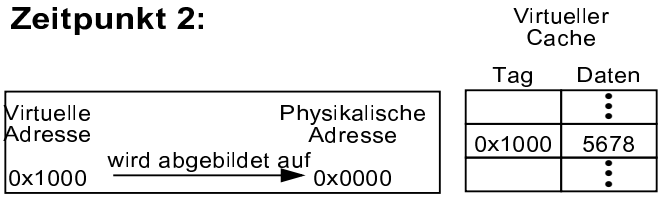
Mehrdeutigkeit (ambiguity)



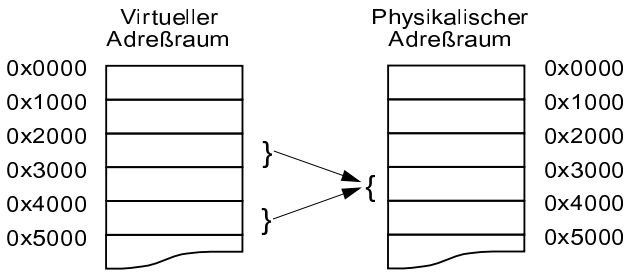
Zeitpunkt 1:



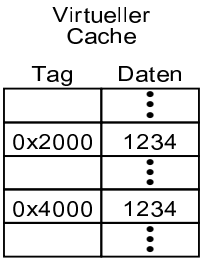
Zeitpunkt 2:



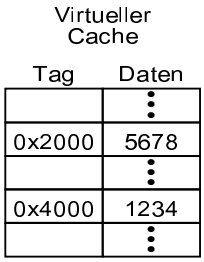
Bedeutungsgleichheit (alias, synonym)



Nach Ansprechen von 0x2000 und 0x4000



Nachfolgende Modifikation von 0x2000



2.1.2.3 Verwaltung virtueller Pufferspeicher

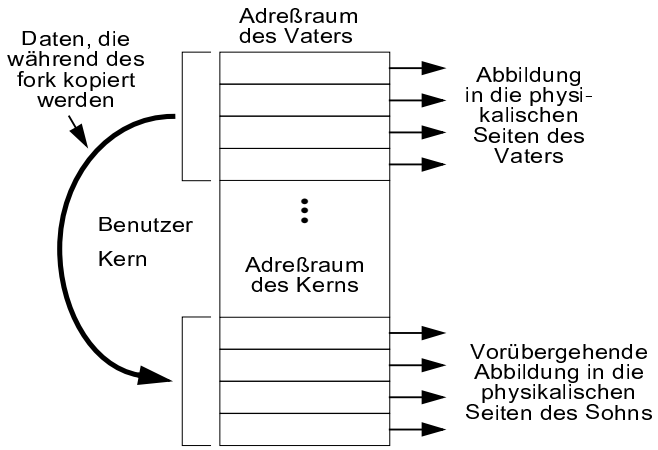
Kontextumschaltung

Da gleiche Adressen in verschiedenen Adreßräumen verschiedene physikalische Adressen referenzieren können, muß der Pufferspeicher invalidiert werden (Vorsicht bei write-back!)

fork

Bei write-back modifizierte Pufferspeicher-Zeilen in ASP transferieren

Wie wird Kopie angelegt, falls kein copy-on-write?



exec

Pufferspeicher muß invalidiert werden

Write-back benötigt kein Rückschreiben, da bisherige Daten aufgegeben werden.

exit

Kein Rückschreiben

brk und sbrk

Vergrößerung unproblematisch

Verkleinerung: Führt zu nicht mehr abgebildeten Seiten, die aber noch teilweise im Pufferspeicher sein können, also Pufferspeicher (selektiv) invalidieren

Gemeinsamer Speicher (shared memory) und speicherabgebildete Dateien (memory mapped files)

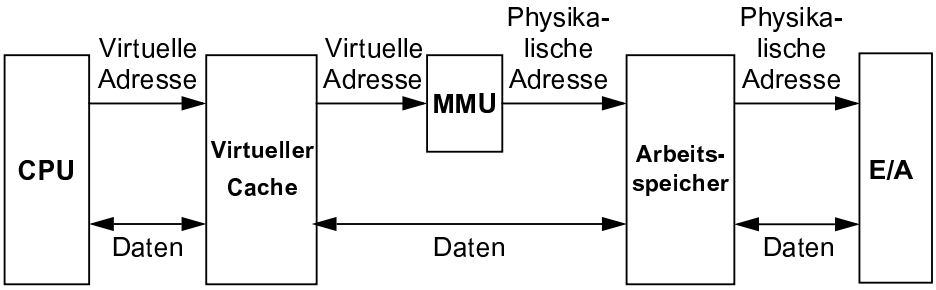
Führt zum Problem der Bedeutungsgleichheit

Lösungsmöglichkeiten bei Mehrfachzuordnung eines gemeinsamen Segments an einen Prozeß unter verschiedenen virtuellen Adressen

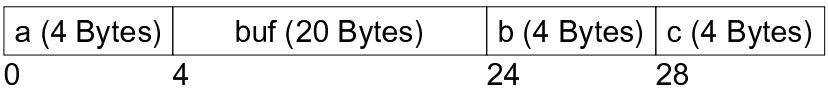
- **verbieten**
- **nicht in Pufferspeicher zwischenlagern**
- **Adreßraum direkt abgebildet zusammen mit write_allocate: nur solche virtuelle Anfangsadressen zulassen, die auf die gleiche Pufferspeicher-Zeile abgebildet werden (Beispiel: Sun 3/200)**
- **Kachel zu jedem Zeitpunkt nur unter einer virtuellen Anfangsadresse zugänglich machen**
- **Abkoppeln von gemeinsamen Segmenten wie Verkürzung von BSS behandeln**

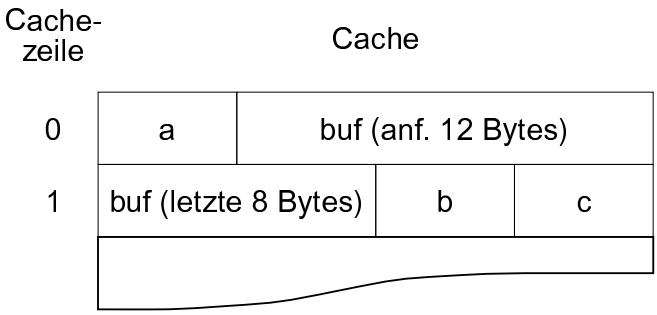
E/A

- **Gepufferte E/A ohne Komplikationen**
- **Ungepufferte E/A**
  - **bei write Informationen evtl. noch im Pufferspeicher, also Rückschreiben vor E/A-Start**
  - **bei read muß Pufferspeicher geleert werden**



- **Besondere Probleme, wenn E/A-Bereich nicht mit Anfangsadresse einer Pufferspeicher-Zeile beginnt oder seine Länge kein Vielfaches der Pufferspeicher-Zeilenlänge ist**





Mehrdeutigkeiten Benutzer-/Kernadreßraum

- **Pufferspeicher-Invalidierung bei Verlassen des Systemzustands**
- **write-back: Rückschreiben von Systemdaten muß bei Verlassen des Systemzustands erfolgen**

2.1.2.4 Schlußfolgerung

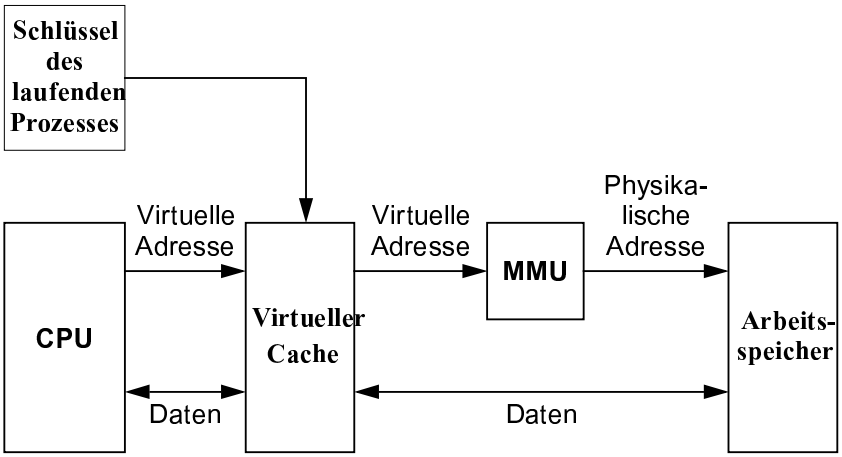
Virtuelle Pufferspeicher bieten hohe Verarbeitungsgeschwindigkeit, da MMU nur bei Zugriffsfehlern benutzt wird

Sie erfordern häufiges Räumen des Pufferspeichers, was bei großen Pufferspeichern zeitaufwendig ist

Das Betriebssystem muß die Phänomene der Mehrdeutigkeit und Bedeutungsgleichheit berücksichtigen, insbesondere bei asynchroner E/A

2.1.3 Virtuelle Caches mit Prozeß-Schlüsseln

2.1.3.1 Arbeitsweise



2.1.3.2 Verwaltung

Kontextumschaltung

- **Bei write-through kein Leeren bei Kontextumschaltung erforderlich**
- **Bei write-back Schwierigkeiten, da nach Kontextwechsel die alte, zuständige SKT nicht mehr bekannt ist**
- **Bei zu wenig Schlüsselwerten muß die Prozessorvergabe auf die Schlüsselverteilung Rücksicht nehmen, wenn nicht jeder Kontextwechsel zur Leerung des Cache führen soll**

fork

- **Sohn braucht neuen Schlüssel**
- **Daten des Vaters müssen im Arbeitsspeicher sein!**
- **Kein copy-on-write**
  - **write-through: Kern-Adreßraum muß nicht bereinigt werden**
  - **write-back: Cache muß explizit geleert werden**
- **Copy-on-write**
  - **write-back: Arbeitsspeicher muß bei jedem fork invalidiert werden, damit Zeilenersetzungen kein fehlerhaftes Verhalten bei copy-on-write verursachen (modifizierte Daten würden verdoppelt, nicht die originalen!)**

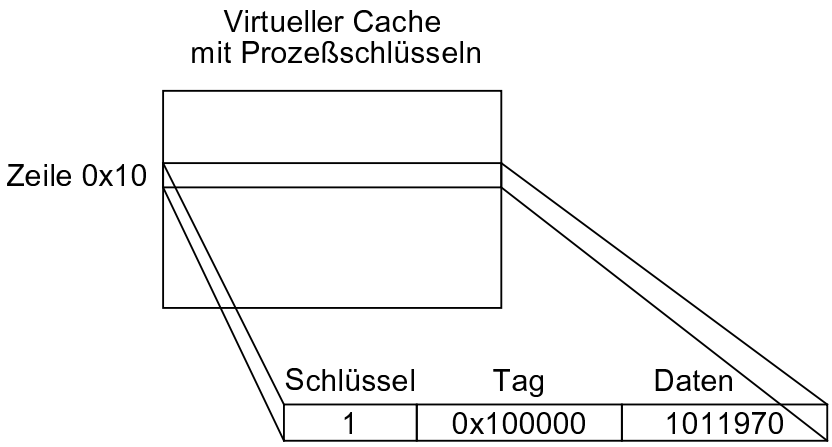
Beispiel

Schlüssel des Vaters: 1

Schlüssel des Sohnes: 2

Cache direkt abgebildet, write-through

Vater liest aus Befehlssegment:



Wenn Sohn liest, wird trotz copy-on-write erneut vom Speicher gelesen

Beispiel: Mehrfach abgebildeter Cache

2-fach assoziativ

Nur Vater hat aus virtueller Adresse 0x100000 geles

Menge 0x10

Schlüssel	Tag	Daten
1	0x100000	1011970
-	-	-

Nur Sohn hat aus virtueller Adresse 0x100000 geles

Menge 0x10

Schlüssel	Tag	Daten
2	0x100000	1011970
-	-	-

Vater und Sohn haben aus virtueller Adresse 0x100 gelesen:

Menge 0x10

Schlüssel	Tag	Daten
1	0x100000	1011970
2	0x100000	1011970

Rückschreiben in Kachel führt korrekt zur Beseitigung des copy-on-write

Exec

- **Prozeßschlüssel beibehalten**
- **Im Cache alle Einträge mit diesem Schlüssel vorher invalidieren**

exit

- **Da normalerweise kein Räumen des Pufferspeichers bei Prozeßumschaltung, Einträge dieses Schlüssels invalidieren**
- **Könnte auch bis zur Wiedervergabe des Schlüssels aufgeschoben werden**

brk, sbrk

- **Wie bei virtuellem Cache**

Shared memory und memory mapped files

Information könnte mehrfach unter verschiedenen Schlüsseln vorhanden sein

- **Unter gleicher virtueller Adresse**
  Unproblematisch, falls write-allocate

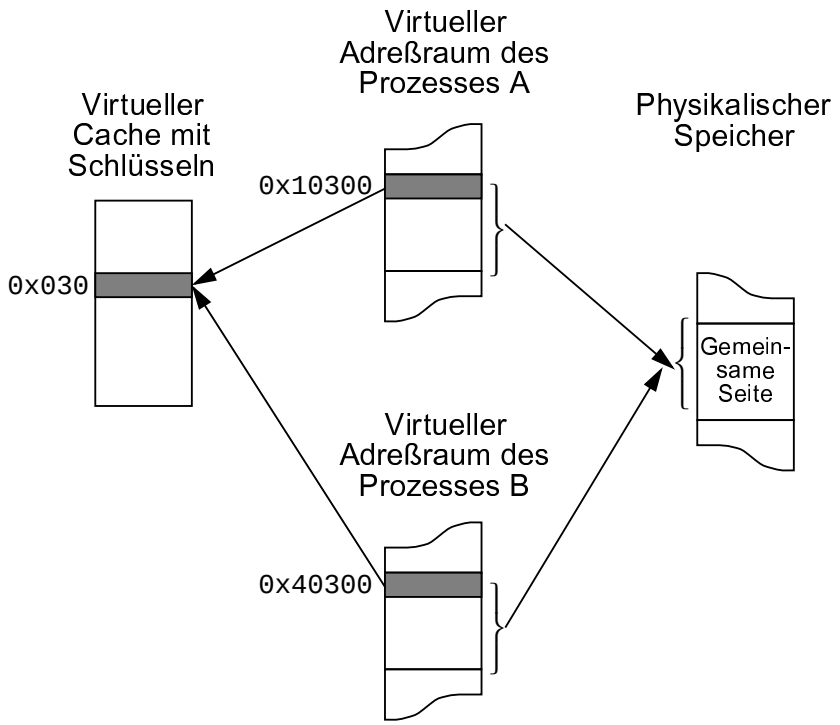
Schlüssel	Mod	Tag	Daten
1	M	0x100000	1011970

Nach Zugriffsfehler:

Schlüssel	Mod	Tag	Daten
2	-	0x100000	1011970

Problematisch, falls nicht write-allocate, da zweiter Zugriff Zugriffsfehler auslöst und direkt vom Arbeitsspeicher liest ohne den Pufferspeicher zu tangieren

- **Unter verschiedenen virtuellen Adressen**



- Möglichkeiten zur Verhinderung von Mehrdeutigkeiten
  - Entfernung aus dem Cache bei Adreßraumumschaltung
  - Puffern (“caching”) gemeinsamer Segmente nicht zulassen
  - Bei direkt abgebildetem Cache mit write_allocate, gemeinsame Segmente so anlegen, daß Anfangsadressen auf die gleiche Cachezeile abgebildet werden; aber Zugriff erzeugt Fehlzugriff!

E/A

Zu behandeln wie rein virtueller Cache

Mehrdeutigkeiten Benutzer - Kern

- Zusätzlicher Modus-Indikator
- Besonderer Schlüssel für Betriebssystem
  - Vorteil: Kern hat Zugang zu seinen Daten unabhängig von der Identität des aufrufenden Prozesses
  - Nachteil: Fehlzugriffe bei Zugriff auf Benutzerdaten

2.1.3.3 Benutzung virtueller Pufferspeicher in MMUs

Pufferspeicher werden in MMUs benutzt als

- ‘translation lookaside buffer’ (TLB), auch als ‘address translation cache’ (ATC) bezeichnet
- TLB ist virtueller Pufferspeicher, der Kacheladressen (also Daten der Seiten-Kachel-Tabelle) und Zugriffsrechte speichert
- Fehlzugriffe werden in manchen Systemen nicht durch Hardware sondern durch BS behandelt (z. B. MIPS)
- Als TLBs werden häufig virtuelle Pufferspeicher mit Schlüsseln verwendet; vorteilhaft, wenn (wie üblich) SKTs keine gemeinsamen Abschnitte haben

2.1.3.4 Schlußfolgerung

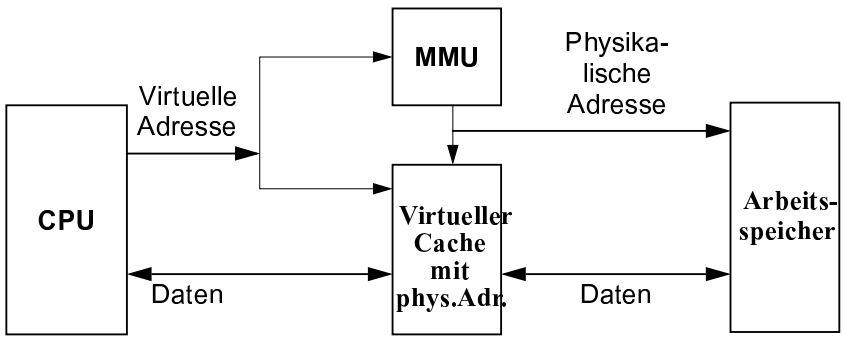
Virtuelle Pufferspeicher mit Schlüsseln reduzieren Zusatzaufwand: Pufferspeicher muß normalerweise bei Prozeßumschaltung nicht invalidiert werden.

Nachteil: gemeinsame Daten können im Pufferspeicher nicht als solche benutzt werden; verbesserbar, wenn BS Anfangsadressen vergeben kann, da dann bei fork (aber auch nur bei fork!) auf explizite Invalidierung des Pufferspeichers verzichtet werden kann

Zusätzliche Probleme treten auf, wenn nicht genügend Prozeßschlüssel zur Verfügung stehen

2.1.4 Virtuelle Pufferspeicher mit physikalischen Adressen als Tags

2.1.4.1 Arbeitsweise



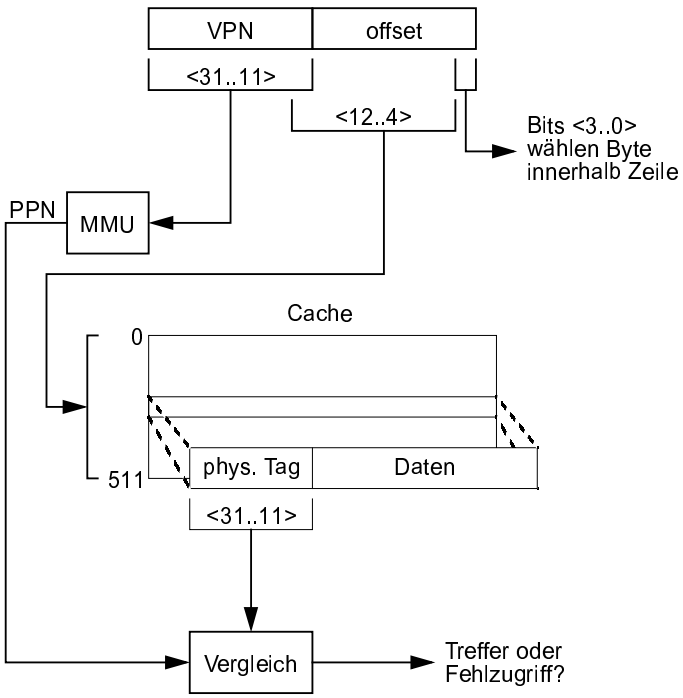
VPN Virtual Page Number
PPN Physical Page Number

Vorteile

- **Mehrdeutigkeiten (ambiguities) zwischen Bereichen unterschiedlicher Prozesse, die nicht gemeinsam benutzt werden, können nicht auftreten. Phys. Adressen haben in diesem Fall die Wirkung von Prozeßschlüsseln.**
- **Gemeinsame Bereiche können effizient betrieben werden (z. B. wenn sie dieselbe virtuelle Anfangsadresse besitzen)**
- **Bei write-back ist über einen Kontextwechsel verzögertes Rückschreiben möglich**
- **Zugriffsrechte werden ständig durch MMU überprüft**

Nachteile

- **Suchgeschwindigkeit im Pufferspeicher wird durch Umsetzgeschwindigkeit der MMU begrenzt**
- **Physikalische Tags erfordern mehr Speicherplatz als virtuelle, wie das nachfolgende Beispiel zeigt:**



2.1.4.2 Verwaltung virtueller Caches mit physikalischen Tags

Kontextumschaltung

- Verschiedene isolierte Prozesse benutzen verschiedene physikalische Adressen, also sind keine besonderen Maßnahmen erforderlich
- Bei gemeinsamen Segmenten kann Bereinigung erforderlich sein.

fork

- Cachebereinigung nicht erforderlich bei copy-on-write
- Daten können im Cache gemeinsam benutzt werden
- Auch bei Benutzung von write-back und bei zweifach assoziativen Caches wird keine Bereinigung benötigt
- Vorsicht bei Modifikation von copy-on-write-Seiten unter write-back

exec

- Invalidierung der alten Adressen

exit

- Wie exec

brk, sbrk

- Bei Schrumpfen analog exec

shared memory und memory mapped files

- Keine weiteren Maßnahmen erforderlich, wenn virtuelle Anfangsadressen auf dieselbe Cachezeile abgebildet werden (Aufgabe des BS!)

E/A

- Wie bei rein virtuellem Cache

Benutzer-Kern-Mehrdeutigkeit

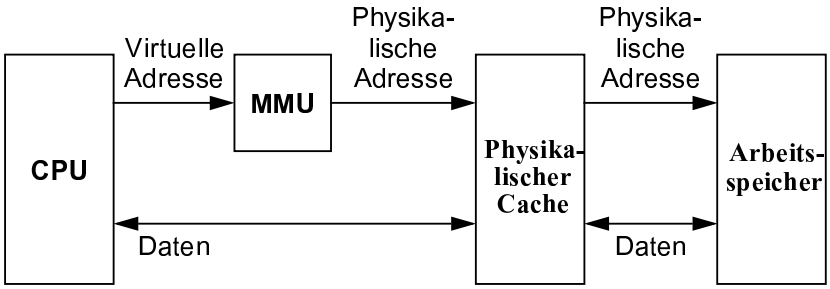
- Mehrdeutigkeiten zwischen Benutzer- und Systemadreßraum können nicht auftreten

2.1.4.3 Schlußfolgerung

- Erhebliche Vorteile gegenüber rein virtuellem Cache und virtuellem Cache mit Prozeß-Schlüsseln
- Wesentlicher Nachteil ist die Benutzung der MMU bei jedem Zugriff
- Geringer Nachteil ist, daß bei freizügiger Wahl der Anfangsadressen gemeinsamer Segmente gelegentlich Cachebereinigungen erforderlich werden.

2.1.5 Physikalischer Cache

2.1.5.1 Arbeitsweise physikalischer Caches



- Vorteile
  - Weder Mehrdeutigkeiten noch Bedeutungsgleichheit möglich
  - Im allgemeinen keine Cachebereinigungen erforderlich
  - Mit ‘snooping’ Bereinigung gänzlich vermeidbar
- Nachteil
  - Bei jedem Zugriff ist Adreßumsetzung durch MMU erforderlich

2.1.5.2 Verwaltung physikalischer Caches

Kontextumschaltung

- Keine besondere Behandlung erforderlich

fork

- Keine besondere Behandlung erforderlich

exec

- Keine besondere Behandlung erforderlich

exit

- Keine besondere Behandlung erforderlich

brk, sbrk

- Keine besondere Behandlung erforderlich

shared memory und memory mapped files

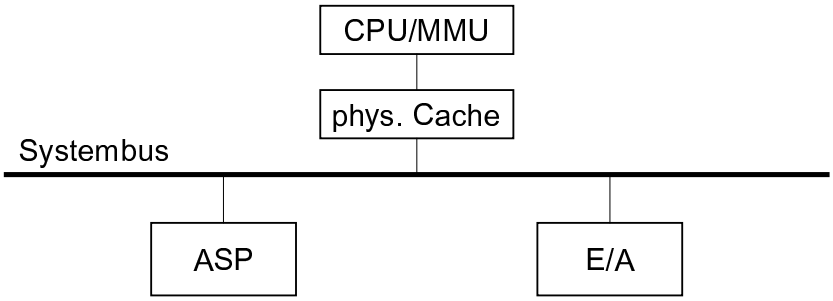
- Keine besondere Behandlung erforderlich

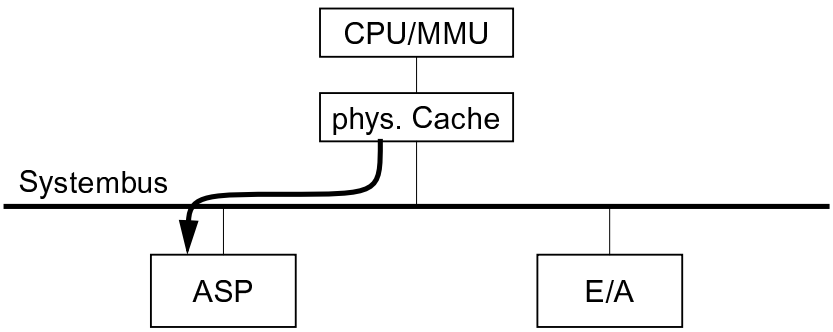
Benutzer-Kern-Mehrdeutigkeit

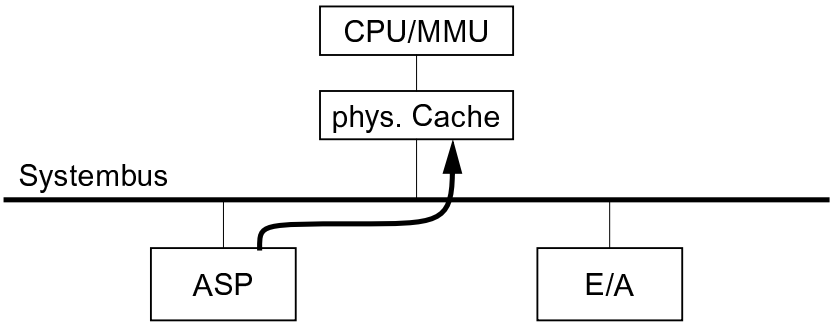
- Keine besondere Behandlung erforderlich

E/A

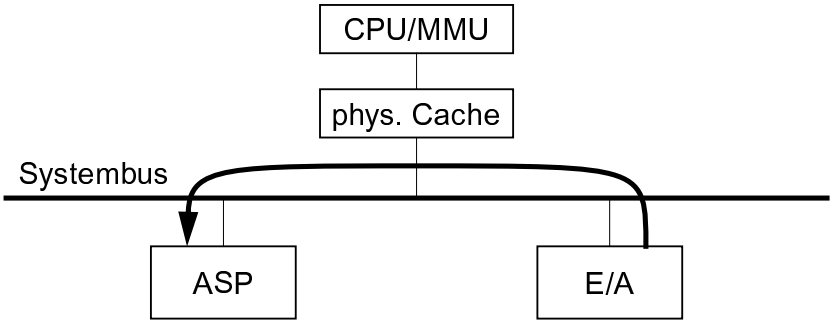
- **Konsistenzgewährleistung durch Busbeobachtung (bus watching, snooping)**

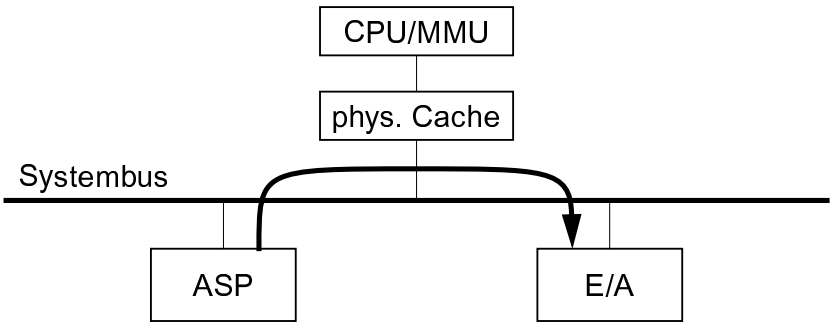




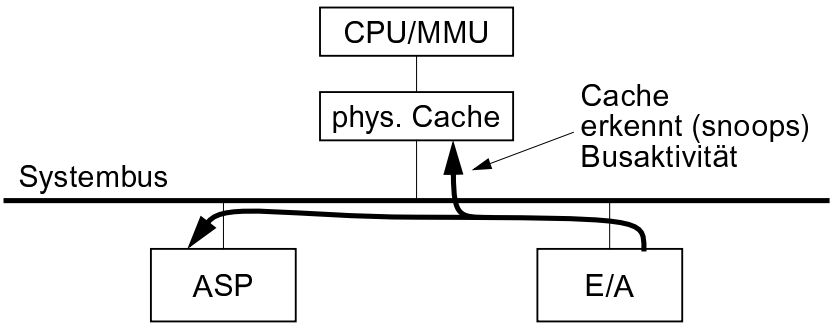


Caches: Physikalischer Cache



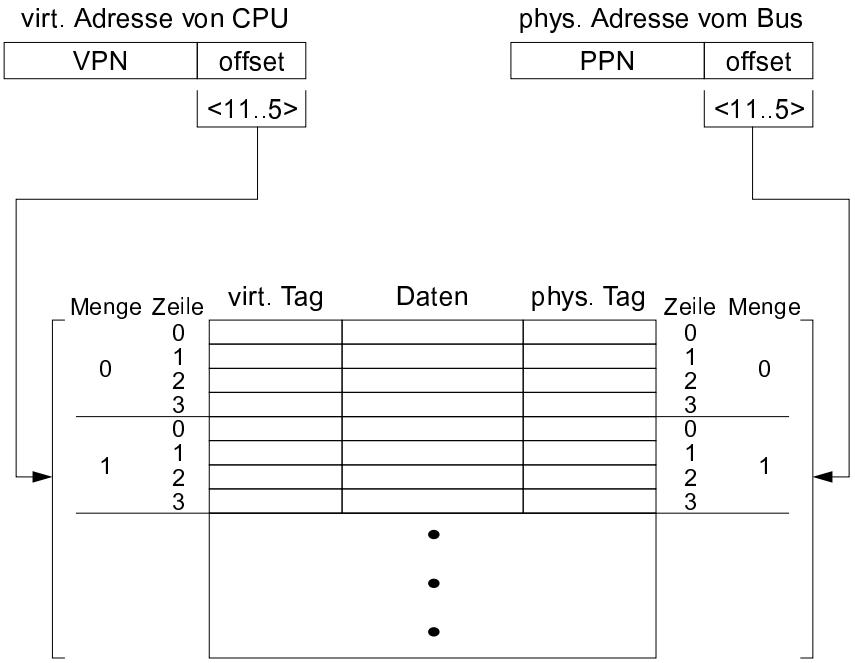


Caches: Physikalischer Cache

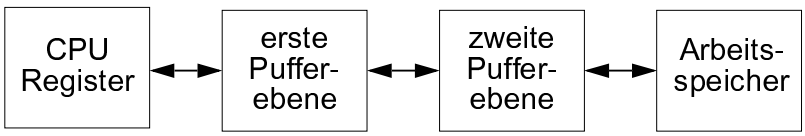


- **Bei write-through nur write-Aufrufe überwachen**
- **Bei write-back**
  - **Alle Transfers überwachen**
  - **Besondere Vorsichtsmaßnahmen am Pufferanfang und Pufferende, wenn sie nicht mit Cachezeilenanfang bzw. -ende übereinstimmen**

Mischung aus virtuellem und physikalischem Cache (i860 XP)



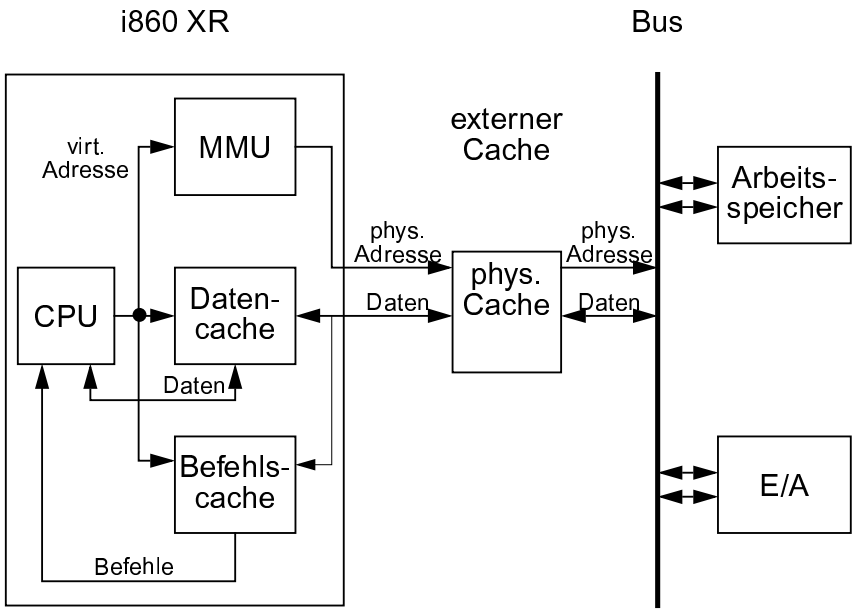
2.1.5.3 Mehrstufige Pufferspeicher



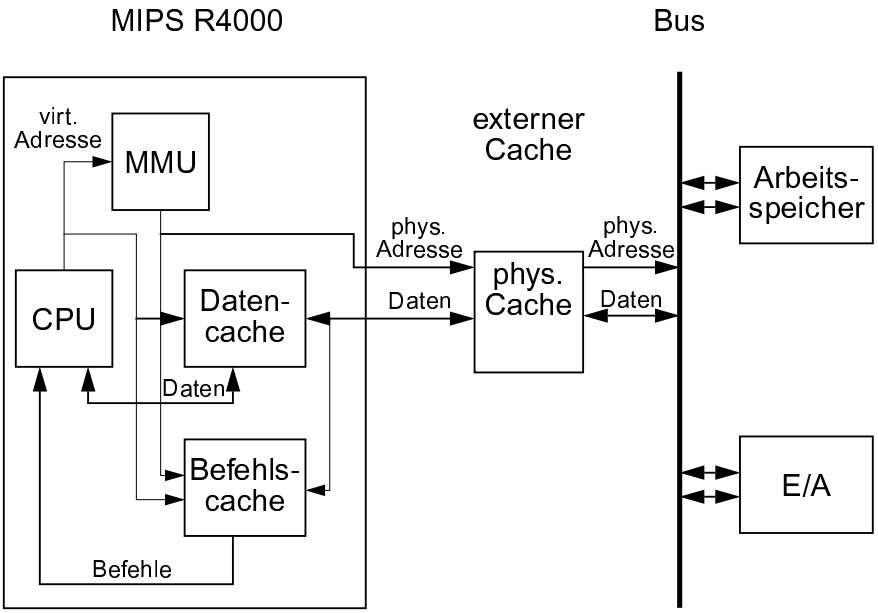
Die beiden Ebenen können unterschiedlich realisiert sein

Typischerweise erste Ebene (primärer Pufferspeicher) als virtueller Pufferspeicher, zweite Ebene (sekundärer Pufferspeicher) als physikalischer Pufferspeicher; Adreßumsetzung erfolgt parallel zum Zugriff zum primären Pufferspeicher.

Primärer virtueller Pufferspeicher und sekundärer physikalischer Pufferspeicher



Primärer virtueller Pufferspeicher mit physikalischen Tags und sekundärer physikalischer Pufferspeicher



2.1.5.4 Schlußfolgerung

Physikalische Pufferspeicher reduzieren erheblich die Notwendigkeit von Eingriffen durch das BS

In Verbindung mit snooping für das BS transparent

Zugriffsgeschwindigkeit begrenzt durch MMU

Verbindung beider Vorteile durch zwei Puffer-Ebenen

2.1.6 Techniken zur effizienten Verwaltung von Pufferspeichern

2.1.6.1 Vorbemerkung

Wesentliche Faktoren, die die Effizienz beeinflussen

- **Design der Hardware**
- **Zeitliche und örtliche Lokalität der Zugriffe in Programmen**
- **Umgang des BS mit den Pufferspeichern**
- **BS-Techniken**
- **Layout von Adreßräumen**
- **Verzögerte Invalidierung von Pufferspeichern**
- **Ausrichtung der Anfangsadresse von Datenstrukturen**

2.1.6.2 Layout des Adreßraums von Prozessen

Virtueller Pufferspeicher

Die Anfangsadressen der Segmente Text, Data und Stack sollten so gewählt werden, daß sie nicht auf die gleiche Pufferspeicherzeile abgebildet werden.
Beispiel: 64 KB, direkt abgebildet, 16 Byte/Zeile
(Bit 0-3: Bytenummer; Bit 4-15: Zeilenindex)

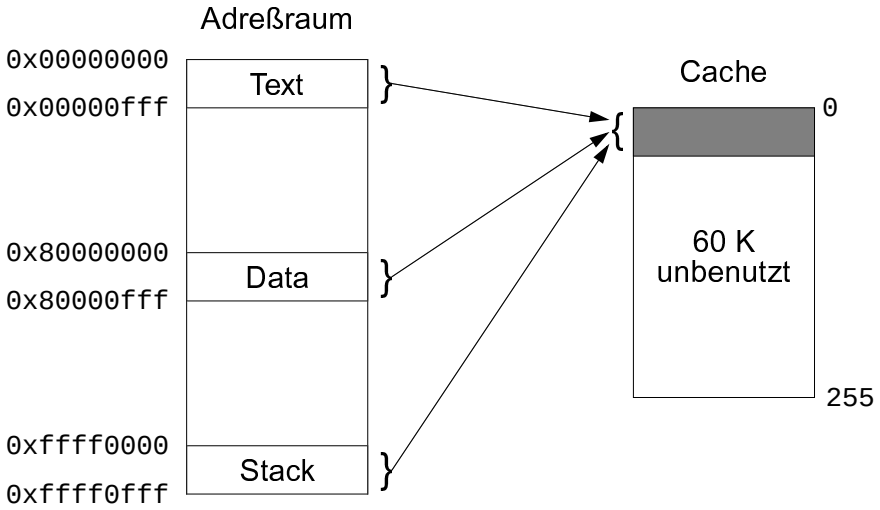
Region	Adresse
Text	0x00000000
Data/BSS	0x80000000
Stack	0xffff0000

Region	Adresse
Text	0x00000000
Data/BSS	0x80000000
Stack	0xffff0000

Adresse in *Text*	Index
0x00000000	0
0x00000010	1
0x00000020	2
0x00000030	3
• • •	• • •
0x00000ff0	255

Adresse in *Data*	Index
0x80000000	0
0x80000010	1
0x80000020	2
0x80000030	3
• • •	• • •
0x80000ff0	

Adresse in *Stack*	Index
0xffff0000	0
0xffff0010	1
0xffff0020	2
0xffff0030	3
• • •	• • •
0xffff0ff0	255



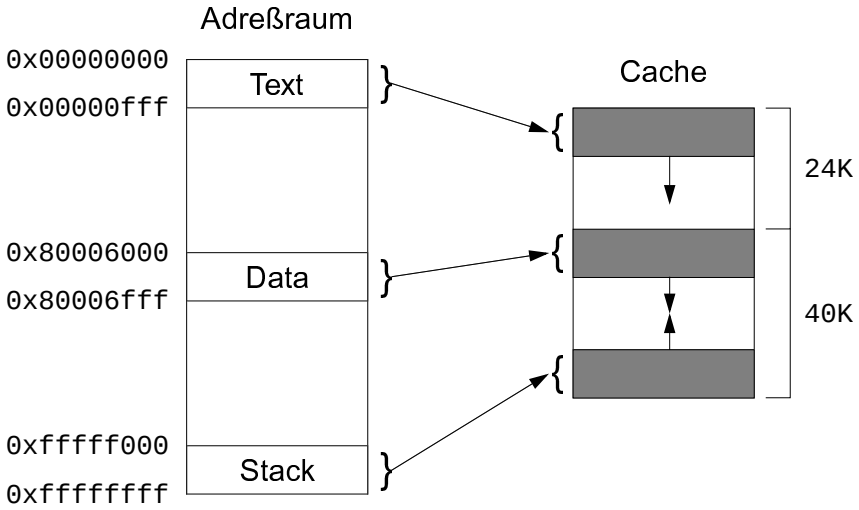
Andere Wahl der Anfangsadressen

Region	Adresse
Text	0x00000000
Data/BSS	0x80006000
Stack	0xffffff000

Adresse in *Text*	Index
0x00000000	0
0x00000010	1
0x00000020	2
0x00000030	3
•	•
•	•
•	•
0x00000ff0	255

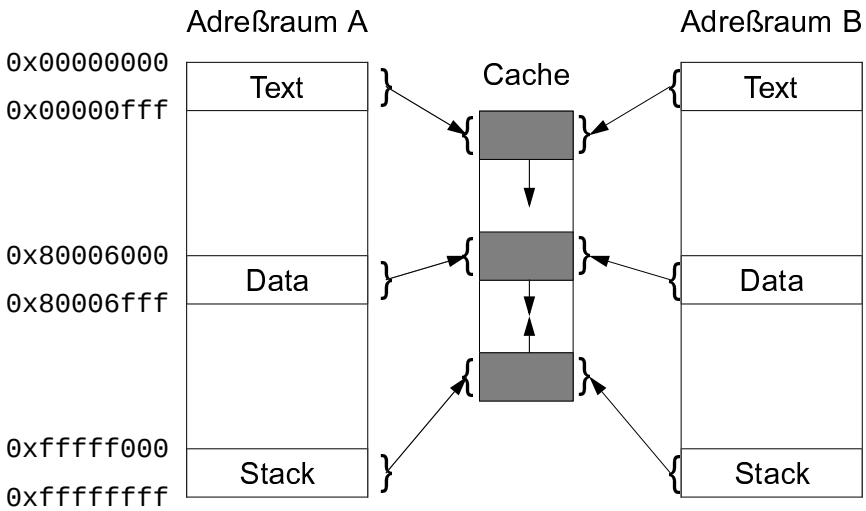
Adresse in *Data*	Index
0x80006000	1536
0x80006010	1537
0x80006020	1538
0x80006030	1539
•	•
•	•
•	•
0x80006ff0	1791

Adresse in *Stack*	Index
0xffff0000	3840
0xffff0010	3841
0xffff0020	3842
0xffff0030	3843
•	•
•	•
•	•
0xffff0ff0	4095

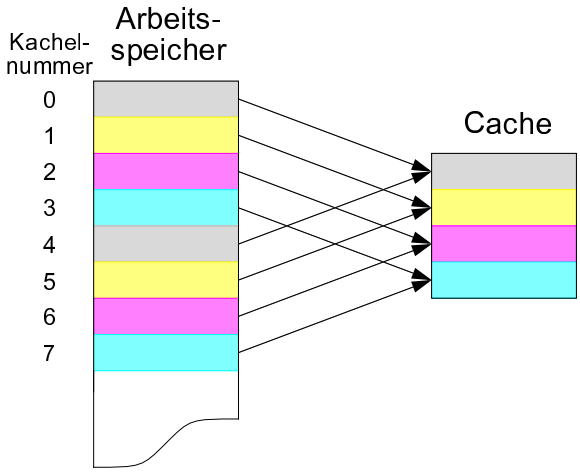


Dynamische Vergabe der Anfangsadressen von Segmenten

Beispiel für die Problematik



Physikalischer Cache



Einteilung der Kacheln in Gruppen, die jeweils aus den Kacheln bestehen, die auf die gleiche Cache-Zeile abgebildet werden. Zuteilung gemäß Round-Robin.

2.1.6.3 Verzögertes Invalidieren

Virtueller Pufferspeicher mit Schlüsseln

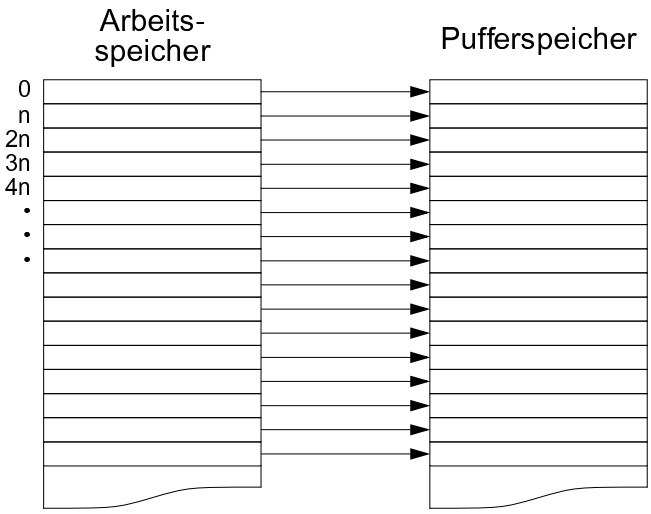
- **Invalidierung nach exit kann verzögert werden bis zur Neuvergabe des Schlüssels**
- **Sammeln mehrerer exits bis Schlüssel wieder vergeben werden**
- **Kompliziert bei write-back, da Kacheln anderweitig vergeben worden sein könnten**

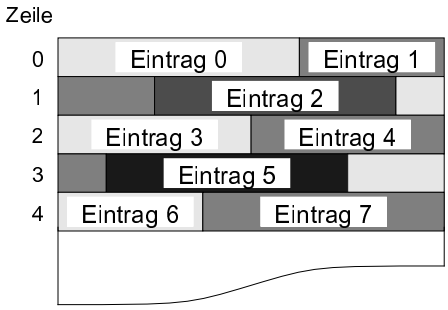
Virtueller Pufferspeicher mit physikalischen Markierungen, physikalischer Pufferspeicher ohne Busüberwachung

- zwei Listen für freie Kacheln
  - mit Zeilen im Pufferspeicher (Kachel unsauber)
  - ohne Zeilen im Pufferspeicher (Kachel sauber)
- bei exit
  - Kacheln in Liste der unsauberen einreihen
- bei Kachelzuordnung
  - aus Liste der sauberen Kacheln zuordnen
  - falls leer, aber unsaubere Kacheln verfügbar, dann Pufferspeicher invalidieren und alle unsauberen Kacheln in die Liste der sauberen umhängen
- analog bei Schrumpfen des Datensegmentes

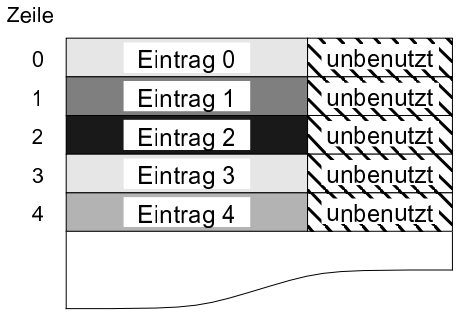
2.1.6.4 Geschickte Wahl der Anfangsadressen von Datenstrukturen

Abbildung bei Zeilengröße n





Vermeidung zweier Fehlzugriffe bei Laden eines Eintrags



2.1.6.5 Schlußfolgerung

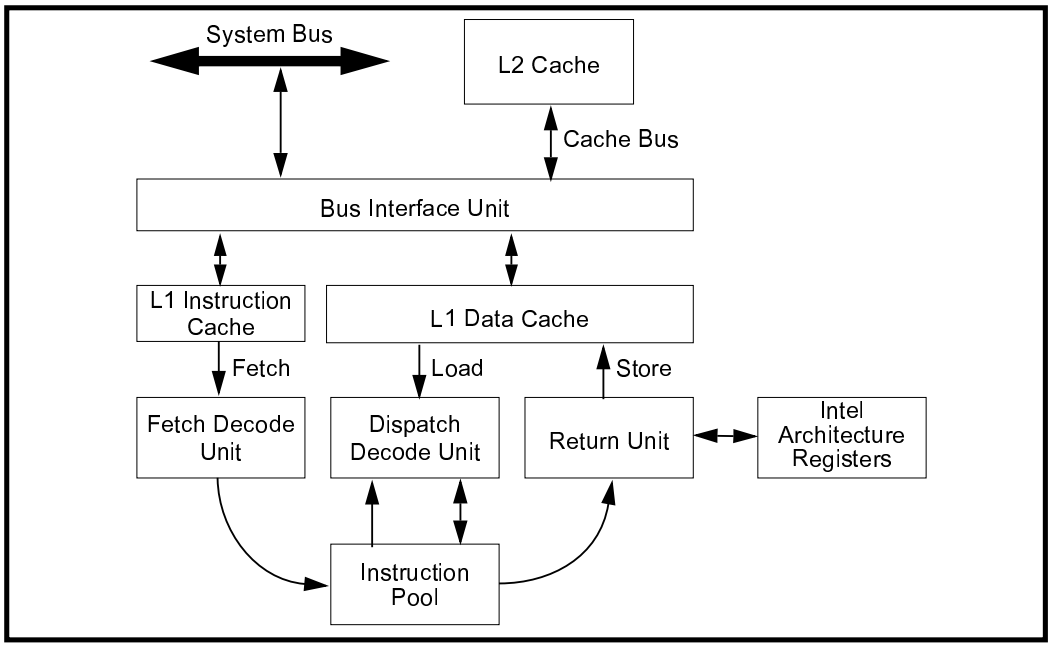
Sorgfältige Adreßraumgestaltung kann Effizienz deutlich erhöhen

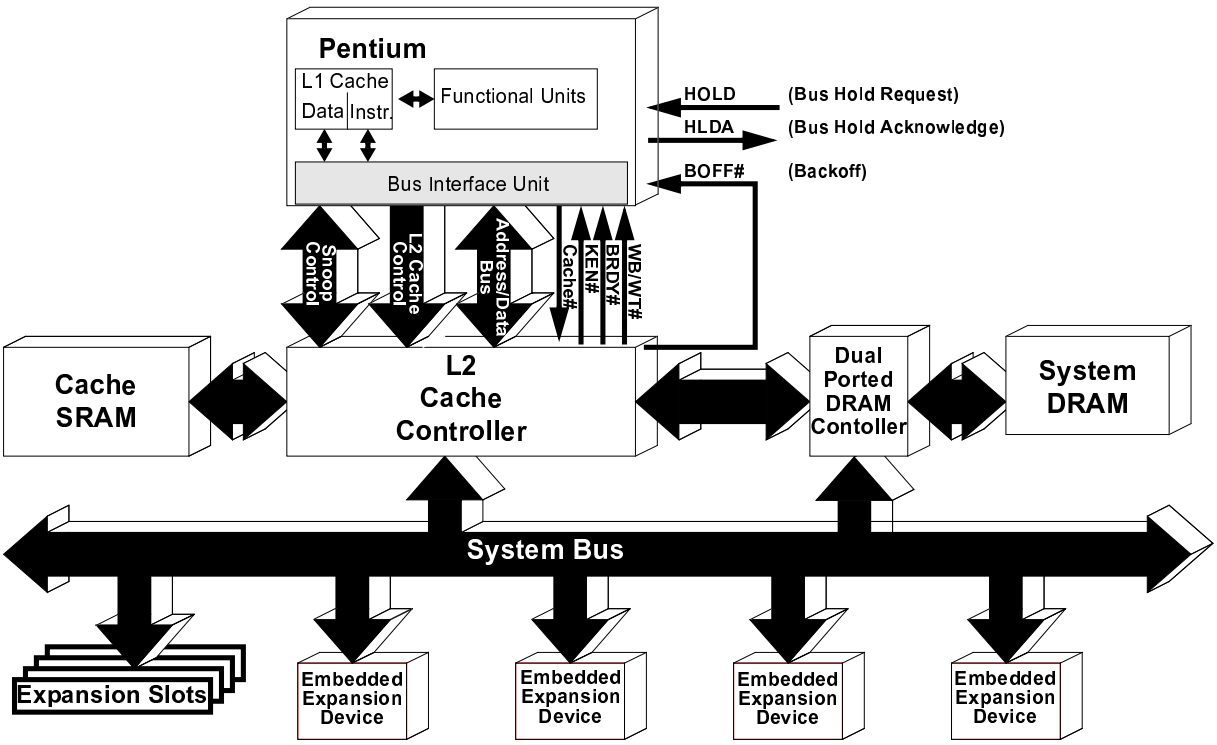
Verzögerte Bereinigung kann benutzt werden, um die Zahl der Invalidierungen wegen nicht mehr zugeordneter Bereiche (infolge exit, brk oder sbrk) zu reduzieren

Ausrichtung häufig benötigter Datenstrukturen auf Pufferspeicherzeilen reduziert Fehlzugriffe

2.1.7 Die Pentium-Architektur

The processing units in the Pentium® Pro processor microarchitecture and their interface with the memory subsystem





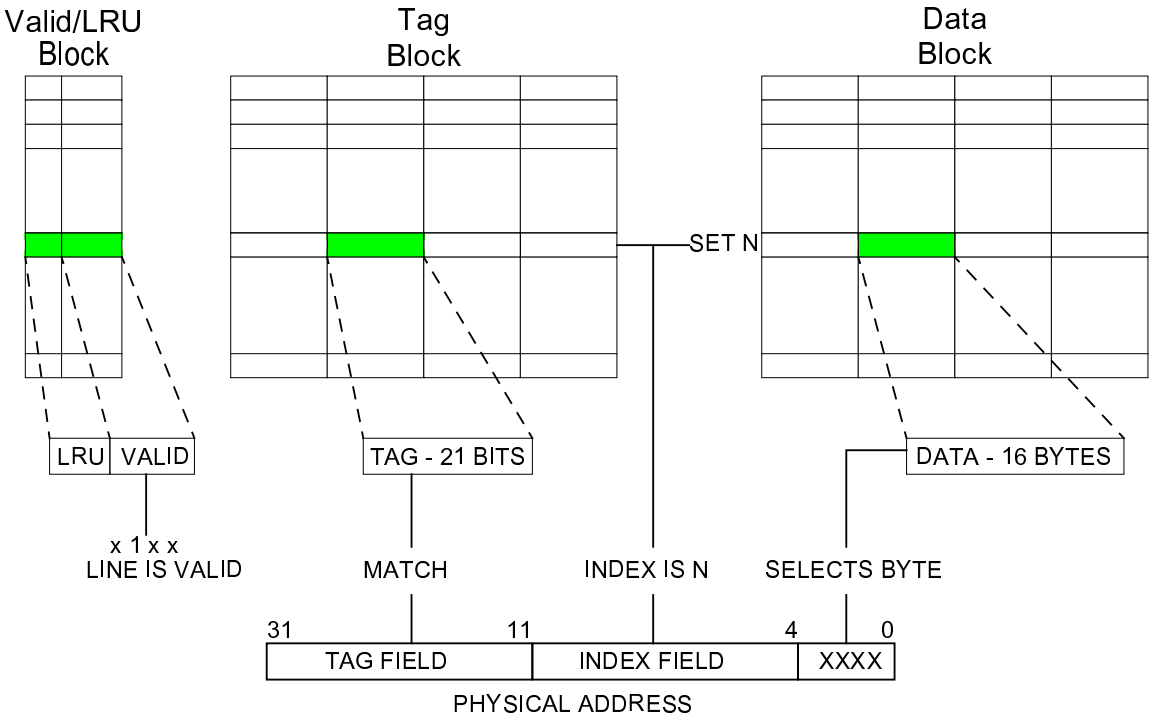
Characteristics of Caches, TLBs, and Write Buffer

Cache or Buffer	Characteristics
L1 instruction Cache[a]	- P6 family and Pentium® processors: 8 or 16 KBytes, 4-way set associative, 32-byte cache line size; 2-way set associative for earlier Pentium procesors. - Intel486™ processor: 8 or 16 KBytes, 4-way set associative, 16-byte cache line size, instruction and data cache combined.
L1 Data Cachea.	- P6 family processors: 8 or 16 KBytes, 2-way set associative, 32-byte cache line size. - Pentium processor: 8 or 16 KBytes, 4-way set associative, 32-byte cache line size; 2-way set associative for earlier processors. - Intel486 processor: System specific.
L2 Unified Cache[b]	- P6 family processors: 256 KBytes, 512 KBytes, or 1 MByte, 4-way set associative, 32-byte cache line size. - Pentium processor: System specific, typically 256 or 512 KBytes, 4-way set associative, 32-byte cache line size. - Intel486 processor: System specific.
Instruction TLB (4-Kbyte Pages)a.	- P6 family processors: 32 entries, 4-way set associative. - Pentium processor: 32 entries, 4-way set associative; fully set associative for Pentium processors with MMX™ technology. - Intel486 processor: 32 entries, 4-way set associative; instruction and data TLB combined.

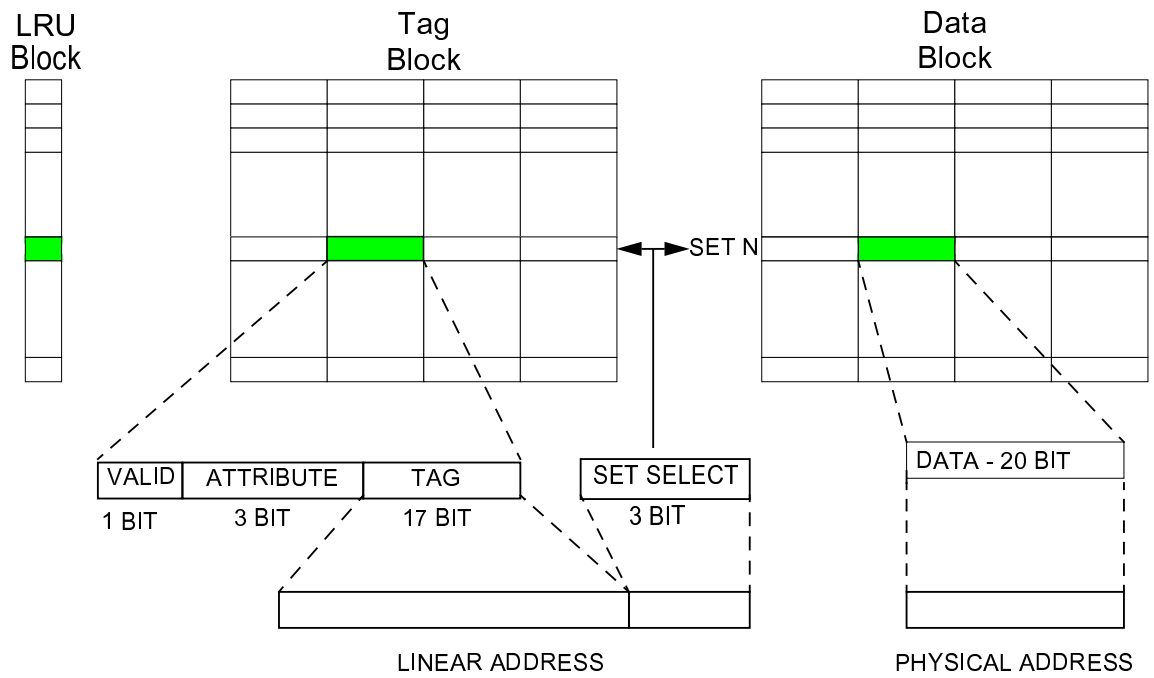
Cache or Buffer	Characteristics
Data TLB (4-KByte Pages)a.	- Pentium and P6 family processors: 64 entries, 4-way set associative; fully set associative for Pentium processors with MMX technology. - Intel486 processor: (see instruction TLB).
Instruction TLB (Large Pages)	- P6 family processors: 2 entries, 2-way set associative. - Pentium processor: Uses same TLB as used for 4-KByte pages. - Intel486 processor: None (large pages not supported).
Data TLB (Large Pages)	- P6 family processors: 8 entries, 4-way set associative. - Pentium processor: 8 entries, 4-way set associative; uses same TLB as used for 4-KByte pages in Pentium processors with MMX technology. - Intel486 processor: None (large pages not supported).
Write Buffer	- P6 family processors: 12 entries. - Pentium processor: 2 Buffers, 1 entry each (Pentium processors with MMX technology have 4 buffers for 4 entries). - Intel486 processor: 4 entries.

a. In the Intel486™ processor, the L1 cache is a unified instruction and data cache, and the TLB is a unified instruction and data TLB
b. In the Intel486 and Pentium® processors, the L2 cache is external to the processor package and optional. In Pentium-III-Xeon processor, the L2 cache comprising up to 2 MByte is on chip.

Cache Organization



Translation Lookaside Buffer



Methods of caching

Write Combining (WC) - System memory locations are not cached (as with uncacheable memory) and coherency is not enforced by the processor's bus coherency protocol. Speculative reads are allowed. Writes may be delayed and combined in the write buffer to reduce memory accesses. This type of cache-control is appropriate for video frame buffers, where the order of writes is unimportant as long as the writes update memory so they can be seen on the graphics display.

Write-through (WT) - Writes and reads to and from system memory are cached. Reads come from cache lines on cache hits; read misses cause cache fills. Speculative reads are allowed. All writes are written to a cache line (when possible) and through to system memory. When writing through to memory, invalid cache lines are never filled, and valid cache lines are either filled or invalidated. Write combining is allowed. This type of cache- control is appropriate for frame buffers or when there are devices on the system bus that access system memory, but do not perform snooping of memory accesses. It enforces coherency between caches in the processors and system memory.

Write-back (WB) - Writes and reads to and from system memory are cached. Reads come from cache lines on cache hits; read misses cause cache fills. Speculative reads are allowed. Write misses cause cache line fills (in the P6 family processors), and writes are performed entirely in the cache, when possible. Write combining is allowed. The write- back memory type reduces bus traffic by eliminating many unnecessary writes to system memory. Writes to a cache line are not immediately forwarded to system memory; instead, they are accumulated in the cache. The modified cache lines are written to system memory later, when a write-back operation is performed. Write-back operations are triggered when cache lines need to be deallocated, such as when new cache lines are being allocated in a cache that is already full. They also are triggered by the mechanisms used to maintain cache consistency. This type of cache-control provides the best performance, but it requires that all devices that access system memory on the system bus be able to snoop memory accesses to insure system memory and cache coherency.

Write protected (WP) - Reads come from cache lines when possible, and read misses cause cache fills. Writes are propagated to the system bus and cause corresponding cache lines on all processors on the bus to be invalidated. Speculative reads are allowed. This caching option is available in the P6 family processors by programming the MTRRs.

Methods of caching available in P6 Family, Pentium®, and Intel486™ processors

Caching Method	P6 Family Processor	Pentium® Processor	Intel486™ Processor
Uncacheable (UC)	Yes	Yes	Yes
Write Combining (WC)	Yes[1]	No	No
Write Through (WT)	Yes	Yes[2]	Yes[2]
Write Back (WB)	Yes	Yes[2]	No
Write Protected (WP)	Yes[1]	No	No

1) Requires programming of MTRRs to implement.

2) Speculative reads not supported.

Konsistenzhaltung: Das MESI-Protokoll

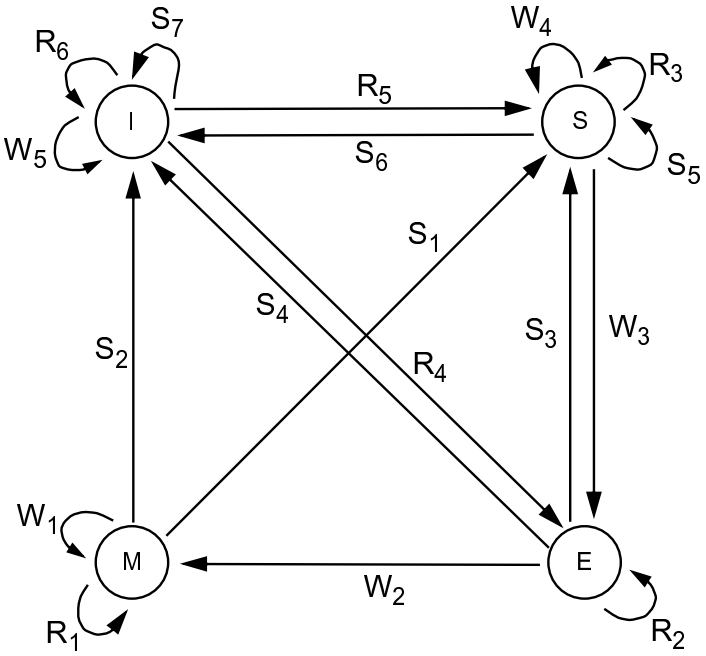
Zustände

Zustand der Pufferzeile	M (Modified)	E (Exclusive)	S (Shared)	I (Invalid)
Pufferzeile gültig?	ja	ja	ja	nein
Inhalt in E_{i+1}	veraltet	aktuell	aktuell	---
Kopien in E_i's anderen Prozessoren?	nein	nein	vielleicht	vielleicht

Entscheidungstabelle für Zustandsübergänge

Operation	read		write			snoop hit
wt/wb	write through	write back	write through		write back	
allocating/ nonallocating			allocating	nonalloca-ting	allocating	
M	nicht mög-lich	M	nicht mög-lich	nicht mög-lich	M	S/I + store
E	nicht mög-lich	E	nicht mög-lich	nicht mög-lich	M	S/I
S	S	S	S	S	E	S/I
I	Zeile füllen + S	Zeile füllen + E	Zeile füllen +S	I	Zeile füllen + E	I

Übergangsdiagramm



Beschreibung der Zustandsübergänge

Lesezugriffe

- **M nach M (R_1):** Ein Lesezugriff hat zu einem Treffer geführt, die Daten sind im Pufferspeicher enthalten und werden der CPU übergeben.
- **E nach E (R_2):** Auch hier hat der Lesezugriff zu einem Treffer geführt, und die Daten werden an die CPU übergeben.
- **S nach S (R_3):** Der Lesezugriff ergab einen Treffer, die Daten werden an die CPU übergeben. Selbstverständlich führen die drei angeführten Lesezugriffe (R_1 bis R_3) nicht zu einer Änderung des Pufferspeicher-Zustands oder der gespeicherten Daten.
- **I nach E (R_4):** Der Lesezugriff verursachte einen Fehlzugriff, die Daten sind nicht im Pufferspeicher abgelegt. Es wird ein externer Lesezyklus ausgelöst, um die Daten zu lesen. Damit sich die Pufferspeicherzeile nach dem Lesevorgang im 'exclusive'-Zustand befindet, müssen die angesprochenen Daten pufferbar sein, sonst erfolgt keine Übernahme in den Pufferspeicher. Außerdem muß eine 'write-back'-Strategie implementiert sein.

Lesezugriffe (Fortsetzung)

- **I nach S (R_5):** Der Lesezugriff hat wie oben zu einem Fehlzugriff geführt. Die Pufferspeicherverwaltung löst auch hier das Füllen der entsprechenden Pufferzeile aus, um die Daten zu lesen. Soll die Pufferzeile anschließend im 'shared'-Zustand sein, müssen die angesprochenen Daten pufferbar sein. Außerdem ist es notwendig, ein 'write-through' vorzusehen.

- **I nach I (R_5):** Der Lesezugriff hat zu einem Fehlzugriff geführt. Die Pufferspeicherverwaltung kann aber die entsprechende Pufferzeile nicht laden (die Daten sind z. B. als 'Memory-Mapped-Register' nicht pufferbar); sie bleibt weiter ungültig.

Schreibzugriffe

- **M nach M (W_1):** Der Schreibzugriff hat zu einem Treffer geführt, die Daten sind im Pufferspeicher vorhanden und werden dort aktualisiert. Nach dem MESI-Protokoll betrifft das einen 'write-back'-Pufferspeicher (für einen 'write-through'-Pufferspeicher sind nur die beiden MESI-Zustände 'shared' und 'invalid' gültig, um das Durchschreiben zu erzwingen), so daß kein Schreibzyklus auf den externen Bus gelegt wird.

- **E nach M (W_2):** Der Schreibzugriff hat einen Treffer auf eine vorher noch nicht überschriebene Pufferspeicherzeile verursacht. Die Pufferspeicherverwaltung kennzeichnet die Zeile nun als modifiziert. Nach dem MESI-Protokoll betrifft auch dieser Fall einen 'write-back'-Pufferspeicher, so daß kein Schreibzyklus auf dem externen Bus erfolgt.

- **S nach E (W_3):** Der Schreibzugriff hat einen Treffer ergeben. Weil die Zeile als 'shared' gekennzeichnet ist, kann sie auch in anderen Pufferspeichern als Kopie abgelegt sein. Diese Einträge müssen invalidiert werden, die Pufferspeicherverwaltung veranlaßt daher einen Schreibzyklus auf dem externen Bus. Die Zeile ist dann nur noch im lokalen Pufferspeicher enthalten und darf folglich als 'exclusive' gekennzeichnet werden.

Schreibzugriffe (Fortsetzung

- **S nach S (W_4):** Der Schreibzugriff hat auch hier einen Treffer ergeben. Der Übergang W_4 betrifft ausschließlich einen 'write-through'-Pufferspeicher (für einen 'write-back'-Pufferspeicher ist der Übergang W_3 vorgesehen), so daß die Pufferspeicherzeile als 'shared' gekennzeichnet bleibt.

- **I nach I (W_5):** Der Schreibzugriff hat zu einem Fehlzugriff geführt. Das MESI-Protokoll sieht keine 'write-allocate'-Strategie vor und somit wird der fehlende Eintrag nicht in den Pufferspeicher geschrieben, die Pufferspeicherzeile bleibt weiter ungültig.

Abfragezyklen (Snooping)

Abfragezyklen werden von einem externen 'Busmaster' ausgelöst, um festzustellen, ob ein Pufferspeicher eine bestimmte Zeile enthält und welchen MESI-Zustand sie gegebenenfalls aufweist.

- **M nach S (S_1):** Der Abfragezyklus hat eine modifizierte Pufferspeicherzeile getroffen, die nicht invalidiert werden soll. Sie wird trotzdem in den Hauptspeicher zurückgeschrieben.

- **M nach I (S_2):** Der Abfragezyklus betrifft wie oben eine Pufferspeicherzeile im Zustand 'modified'. Im Gegensatz zu S_2 soll sie hier aber invalidiert werden. Zusätzlich wird sie in den Hauptspeicher zurückgeschrieben.

- **E nach S (S_3):** Der Abfragezyklus betrifft hat eine Pufferspeicherzeile getroffen, die als 'exclusive' markiert ist. Sie ist also bisher nicht modifiziert worden und muß somit nicht zurückgeschrieben werden. Der Zweck von S_3 ist, eine vorher nur in einem Pufferspeicher vorhandene Zeile auch in einen anderen Pufferspeicher des Systems zu übertragen. Die Pufferspeicherzeile kann dann nicht mehr 'exclusive' sein, sondern muß als 'shared' gekennzeichnet werden.

Abfragezyklen (Fortsetzung)

- **E nach I (S_4):** Der Abfragezyklus betrifft wie bei S_3 eine Pufferspeicherzeile, die als 'exclusive' markiert ist. S_4 wird ebenfalls dazu verwendet, eine vorher nur in einem Pufferspeicher vorhandene Zeile auch in einen anderen Pufferspeicher zu übertragen. Im Gegensatz zu S_3 wird die Zeile aber invalidiert, so daß sie als Pufferspeicherzeile im Zustand 'exclusive' im anderen Pufferspeicher vorhanden sein kann (sonst ist nur 'shared' möglich).

- **S nach S (S_5):** Der Abfragezyklus betrifft eine Pufferspeicherzeile, die als 'shared' gekennzeichnet ist und somit möglicherweise als Kopie in einem anderen Pufferspeicher vorliegt. Der Übergang S_5 informiert das externe System also lediglich, daß die betreffende Pufferspeicherzeile im abgefragten Pufferspeicher vorhanden ist. Es erfolgt keine weitere Busaktivität.

- **S nach I (S_6):** Der Abfragezyklus hat wie oben zu einem Treffer auf eine als 'shared' markierte Pufferspeicherzeile geführt. Sie ist nicht modifiziert worden und muß bei der folgenden Invalidierung auch nicht zurückgeschrieben werden. Der abfragende externe 'Busmaster' weiß dann, daß seine Kopie ebenfalls aktuell ist.

Abfragezyklen (Fortsetzung)

- **I nach I (S_7):** Der Abfragezyklus hat eine Pufferspeicherzeile ermittelt, die als 'invalid' markiert ist. Sie enthält somit keine gültigen Daten. Es erfolgt keine weitere Busaktivität und der abfragende 'Busmaster' ignoriert den Inhalt der betreffenden Zeile im angesprochenen Pufferspeicher.