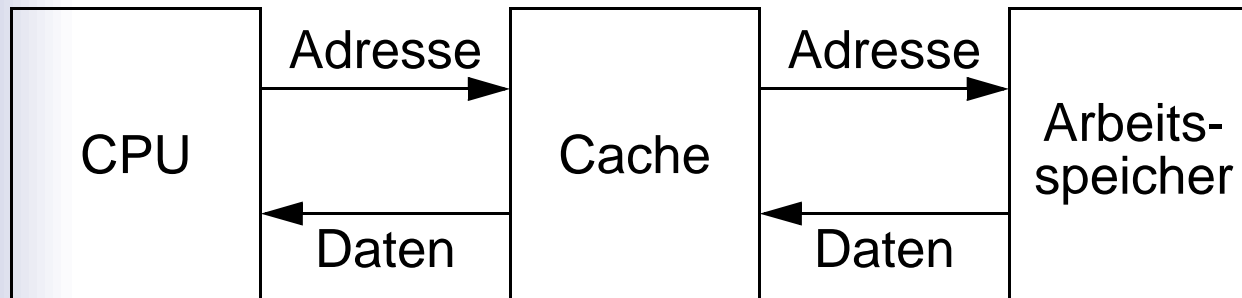


4.8X Caches

1 Grundlagen

■ Funktion



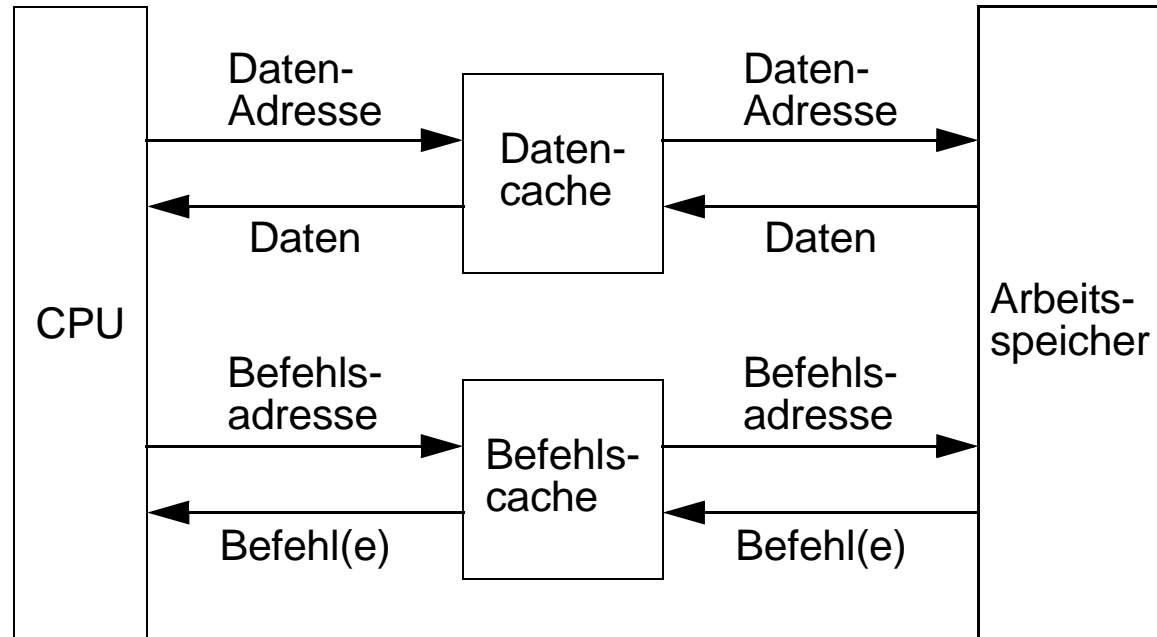
- ◆ Pufferspeicher zur Ausnutzung der zeitlichen und örtlichen Lokalität
- ◆ Zugriff auf gespeicherte Daten mit geringer Latenz und hoher Bandbreite
- ◆ Reduktion der Zugriffe auf den Arbeitsspeicher (wichtig in Multiprozessen)
- ◆ Puffer für asynchrone Prefetch-Operationen

1 Grundlagen (2)

- Zugriff:
 - ◆ Fehlzugriff (cache miss)
 - Compulsory (cold start, first reference)
Ein Datenblock war noch nie im Cache eingelagert.
 - Capacity
Die für die Ausführung benötigten Daten passen nicht in den Cache.
 - Conflict (collision, interference)
Die Daten eines Blocks behindern sich gegenseitig bei der Einlagerung in den Cache (siehe Abschnitt Organisation des Caches)
 - ◆ Treffer (cache hit)
 - ◆ Trefferrate (hit ratio))
- Virtuelle oder physische Adressen?

1 Grundlagen (3)

- Getrennte Pufferspeicher für Daten und Befehle (Harvard Architecture)



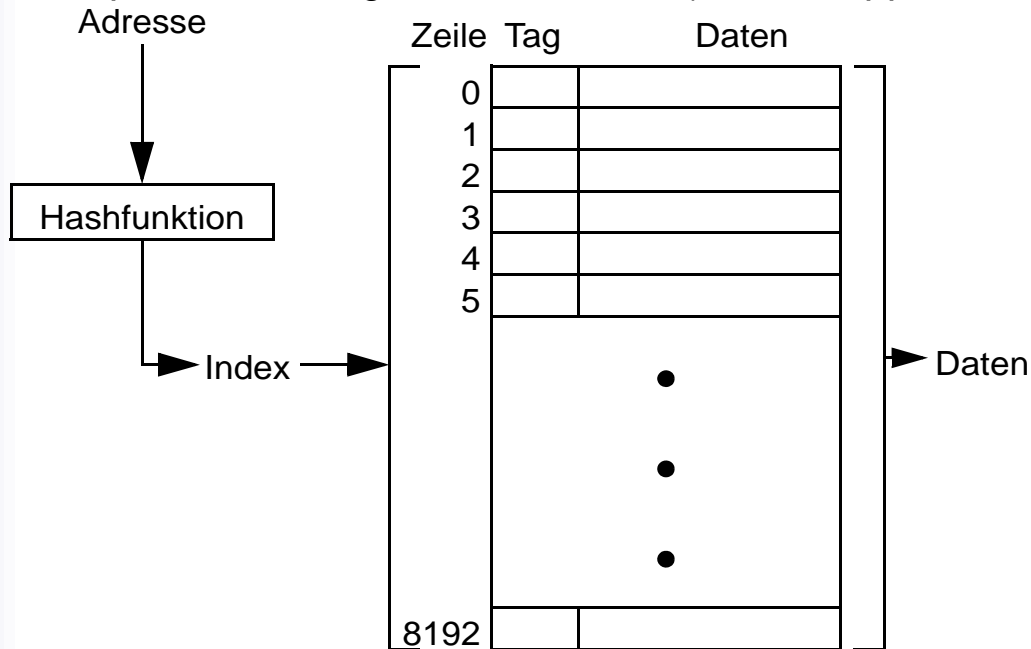
1 Grundlagen (4)

- Suchen im Pufferspeicher
 - ◆ Verwendung von Hash-Techniken
- Schreibstrategien
 - ◆ Treffer
 - **write-through**: Arbeitsspeicher immer auf dem neuesten Stand, zuweilen unnötige Verlangsamung
 - **write-back**: 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**: Modifikation wird nur im Arbeitsspeicher vorgenommen.
- Ersetzungsstrategien
 - ◆ Typischerweise LRU oder Approximationen von LRU, aber auch zufällige Auswahl

1 Grundlagen (5)

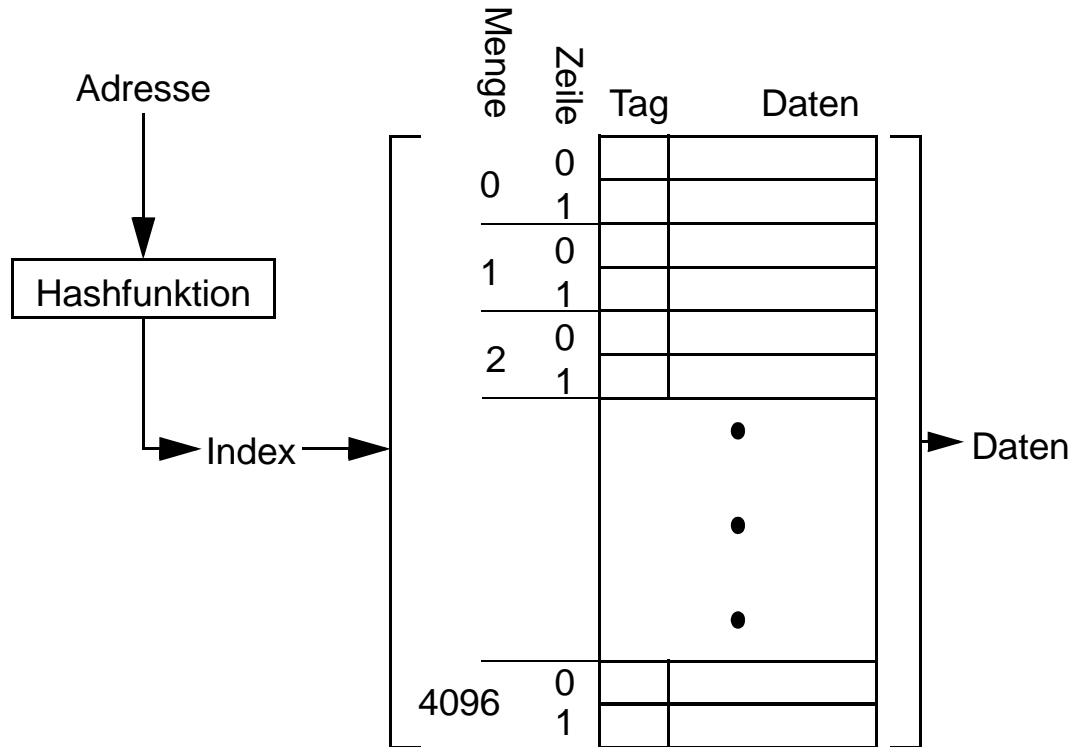
■ Organisation

- ◆ Aufbau aus Cache-Zeilen mit Daten fester Länge.(z.B. 32/64 Bytes)
- ◆ Abbildung der Adresse auf die Zeile
- ◆ Eindeutige Identifizierung der Daten aus dem Speicher durch Tag-Feld:
Beispiel: Direkt abgebildeter Cache (direct mapped cache)



1 Grundlagen (6)

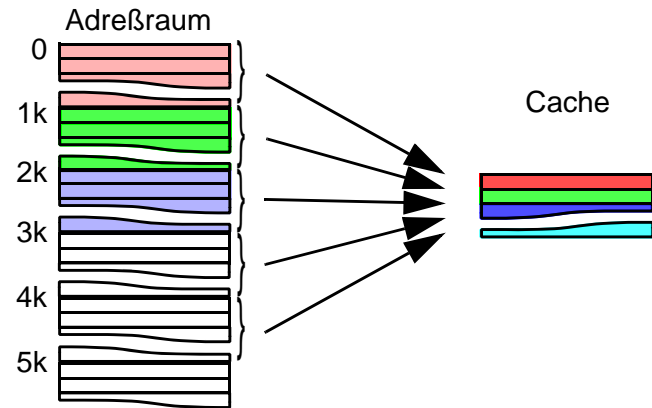
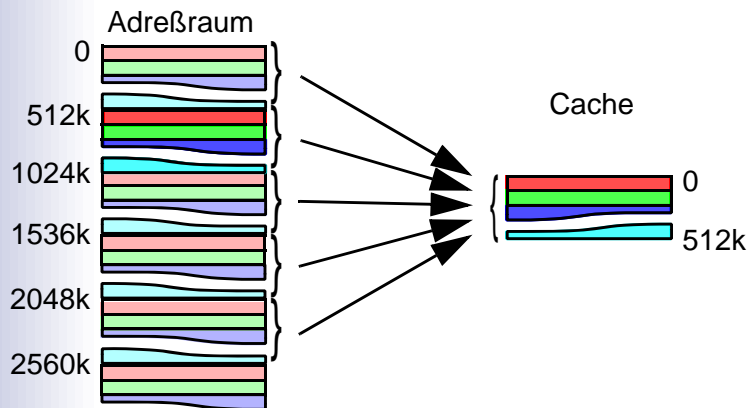
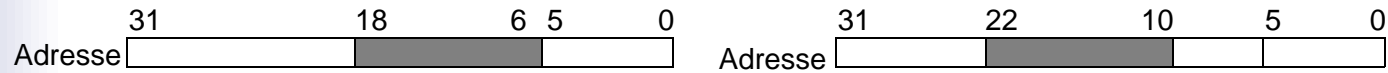
Beispiel: Zweifach assoziativer Cache (two-way set associative cache)



1 Grundlagen (7)

■ Hash-Algorithmen

- ◆ Modulo-Hashing, z. B. Bits 0 - 5 zur Auswahl des Bytes innerhalb einer Zeile, Bits 6 - 18 zur Auswahl der Zeile, Bits 19 - 31 sind Tag
- ◆ Hashing durch Ausschnitt aus der Adresse, z. B. Bits 10 - 22 als Zeilenadresse. Dieses Verfahren wird bei Caches jedoch nicht eingesetzt.



1 Grundlagen (8)

- Leeren des Pufferspeichers (cache flushing)
 - ◆ Alle Realisierungen sehen Möglichkeit zur zwangsweisen Leerung des gesamten Cache oder einzelner Zeilen vor
 - Aktualisierung des Arbeitsspeichers
 - Invalidierung des Cache
 - ◆ 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

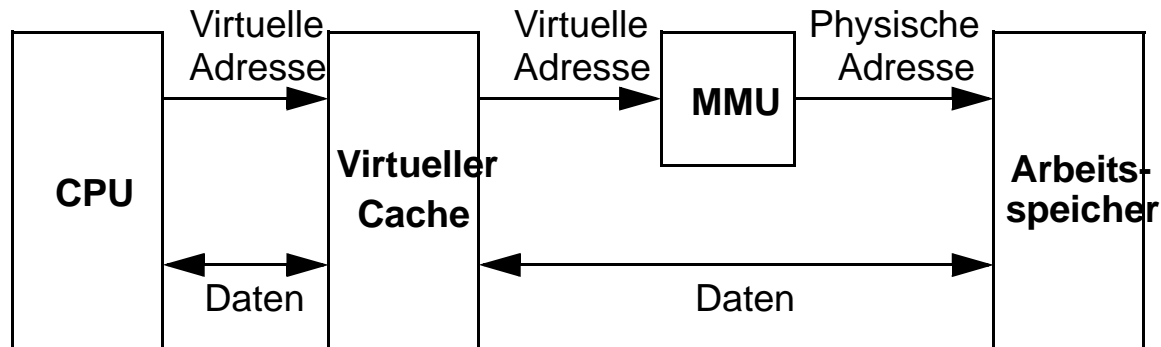
1 Grundlagen (9)

- Leistung von Pufferspeichern
 - Zeitliche Lokalität spricht für write-back.
 - Kleiner Cache spricht für mehrfach-assoziative Vorgehensweisen mit großen Mengen.
 - Adreßlokalität spricht für große Zeilenlänge.
 - Erreichbare Leistungssteigerung durch Cache wird wesentlich vom Betriebssystem beeinflusst.

- Unterschied in der Realisierung von Pufferspeichern
 - Größe
 - Zeilengröße
 - Mengengröße
 - Aufsuchen mit virtueller oder physischer Adresse
 - Kennzeichnung der Zeilen
 - Ersetzungsstrategie / Nutzung von 'write-allocate'
 - 'write-through' oder 'write-back'

4.8X.1 Virtuelle Caches

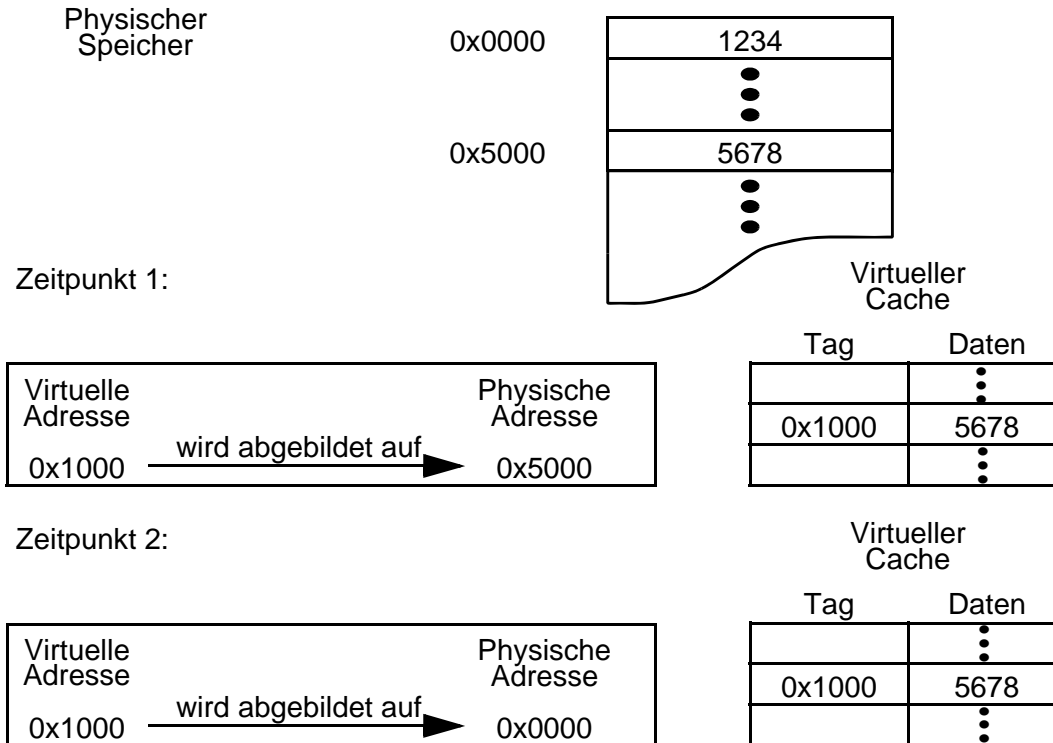
1 Virtually Indexed, Virtually Tagged (ARM v4/v5)



1 Virtually Indexed, Virtually Tagged (2)

◆ Probleme der Mehrdeutigkeit (ambiguity):

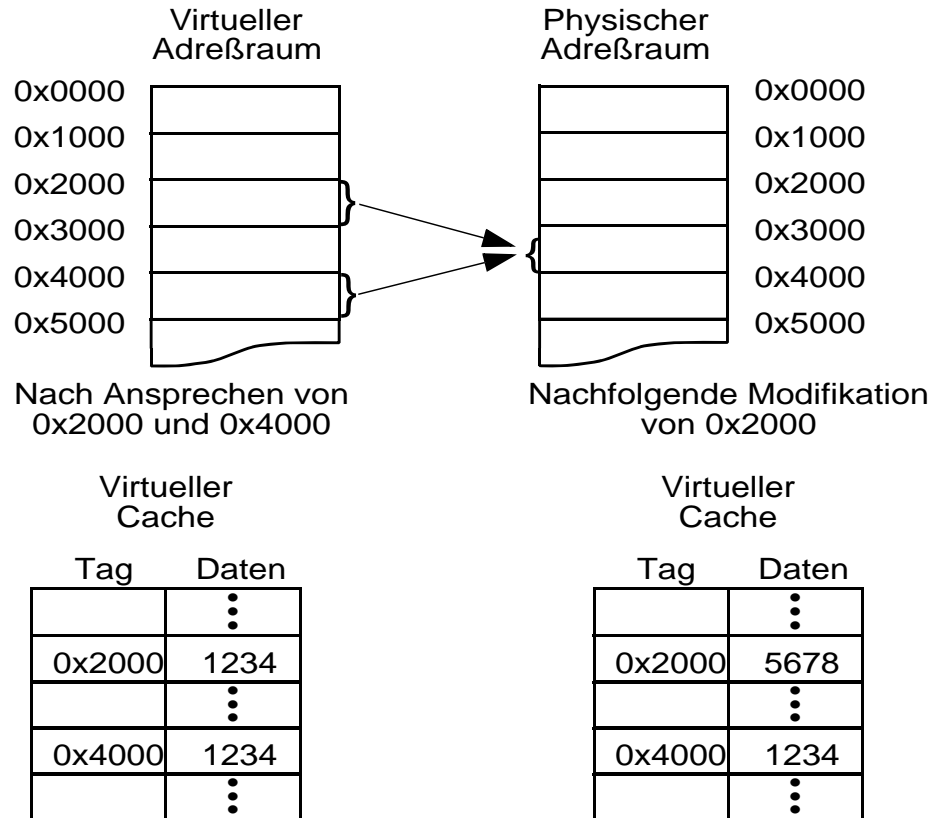
- Zwei identische virtuelle Adressen zeigen zu 2 Zeitpunkten auf verschiedene physische Speicherbereiche



1 Virtually Indexed, Virtually Tagged (3)

◆ Problem der Bedeutungsgleichheit (alias):

- Verschiedene virtuelle Adressen zeigen auf den gleichen Speicherbereich



1 Virtually Indexed, Virtually Tagged (4)

■ Cache Verwaltung

◆ Kontextumschaltung:

Da gleiche Adressen in unterschiedlichen Adreßräumen unterschiedliche physische Adressen referenzieren können, muß der Cache invalidiert werden (incl. Zurückschreiben bei write-back!)

◆ Fork:

- Flushen des Caches

◆ Exec:

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

◆ Exit: Flushen des Caches. Invalidierung des Caches nach dem Wait.

◆ Brk und Sbrk

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

1 Virtually Indexed, Virtually Tagged (5)

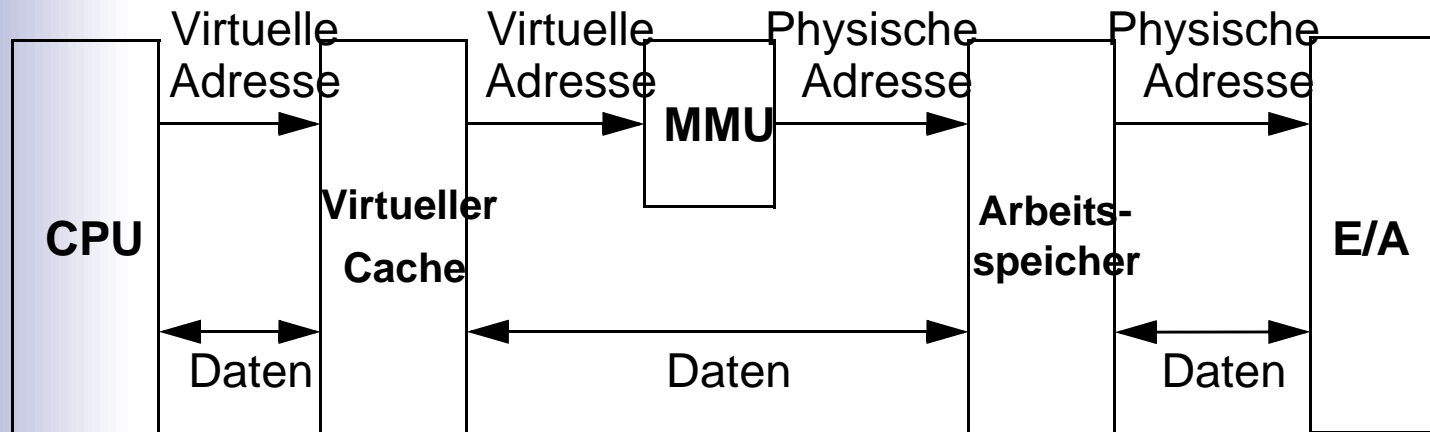
■ Cache Verwaltung

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

- Führt zum Problem der Bedeutungslosigkeit
- Mehrfachzuordnung eines gemeinsamen Segments an einen Prozeß unter verschiedenen virtuellen Adressen:
 - verbieten
 - nicht im Cache puffern
 - Adreßraum direkt abgebildet zusammen mit `write_allocate`: nur solche virtuelle Anfangsadressen zulassen, die auf die gleiche Cache-Zeile abgebildet werden (Beispiel: Sun 3/200)
 - Kachel zu jedem Zeitpunkt nur unter einer virtuellen Anfangsadresse zugänglich machen

1 Virtually Indexed, Virtually Tagged (6)

- Cache Verwaltung
 - ◆ Gepufferte E/A ohne Komplikationen
 - ◆ Ungepufferte E/A
 - Write: Informationen evtl. noch im Cache
→Rückschreiben vor E/A-Start
 - Read: Cache muß invalidiert werden.

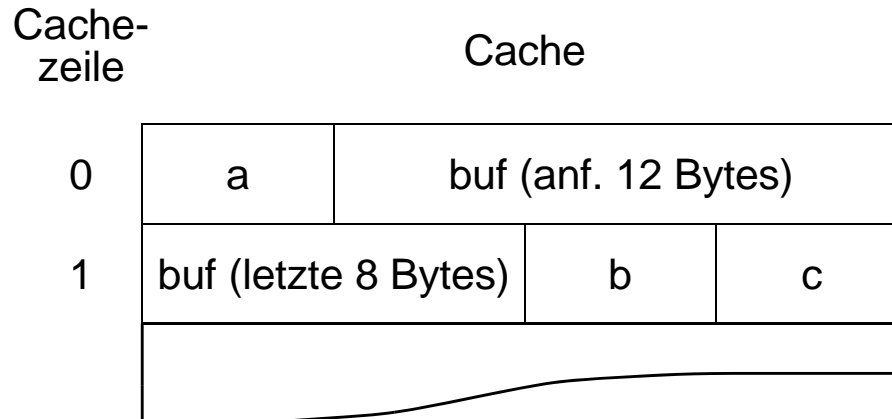
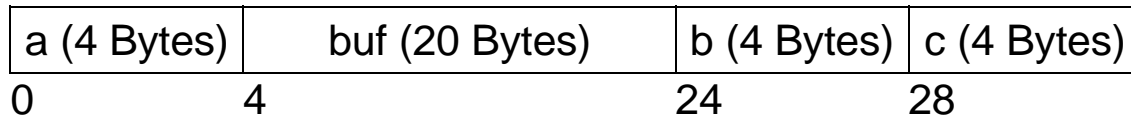


1 Virtually Indexed, Virtually Tagged (7)

■ Cache Verwaltung

◆ Ungepufferte E/A (2)

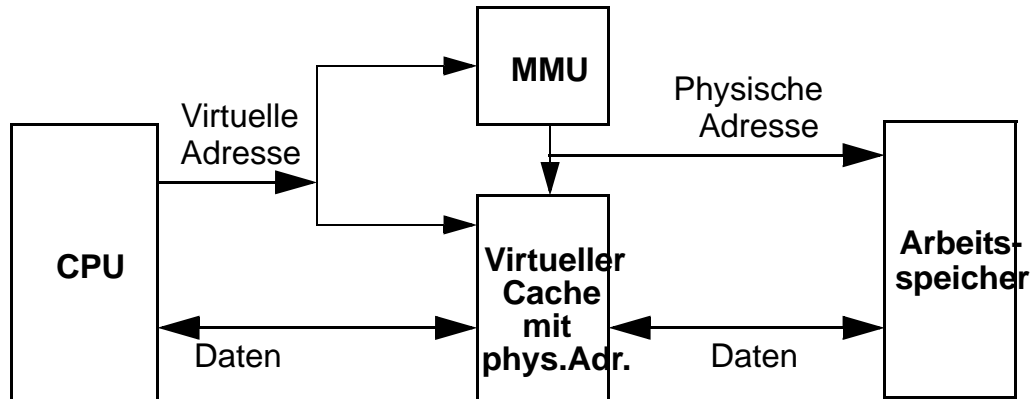
- Besondere Probleme, wenn E/A-Bereich nicht mit Anfangsadresse einer Cache-Zeile beginnt oder seine Länge kein Vielfaches der Cachezeilengröße ist.



1 Virtually Indexed, Virtually Tagged (8)

- Cache Verwaltung
 - ◆ Mehrdeutigkeiten Benutzer-/Kernadreßraum
 - Rückschreiben von Systemdaten beim Verlassen des Systemzustands
 - Cache-Invalidierung bei Verlassen des Systemzustands
- Echtzeiteigenschaften:
 - ◆ Hohe Verarbeitungsgeschwindigkeit der CPU, da die MMU nur bei Zugriffsfehlern benutzt wird.
 - ◆ Die Ablaufgeschwindigkeit eines Prozesses mit fester Speicherabbildung ist konstant.
 - ◆ Bei dynamischer Speicherzuteilung wird die Laufzeit durch Capacity Misses variable.
 - ◆ Kontextwechsel und E/A sind durch das häufige Invalidieren des Caches extrem aufwendig (insbesondere bei großen Caches).

2 Virtually Indexed, Physically Tagged (ARMv6/v7)

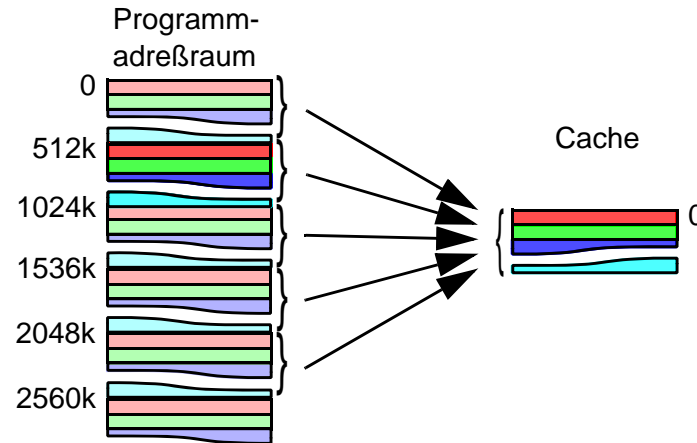


◆ Cache Verwaltung

- Keine Mehrdeutigkeiten mehr.
- Kontext-Wechsel:
Kein Cache-Flush
- Shared memory und Memory Mapped Files:
Virtuelle Anfangsadressen müssen auf dieselbe Cachezeile abgebildet werden.
- E/A:
Cache-Flush wie beim virtually indexed Cache

2 Virtually Indexed, Physically Tagged (2)

■ Zugriffskonflikte



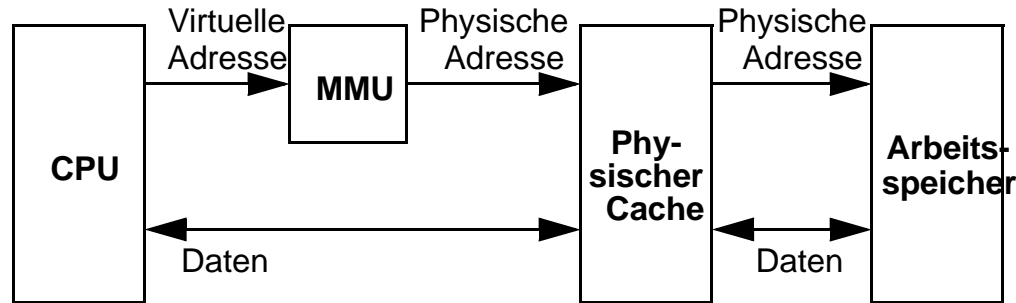
- ◆ Datenstrukturen, bei denen die Differenz der Adreßbereiche einem Vielfachen der Cachegröße entspricht, werden auf die selbe Cachezeile abgebildet (Conflict Misses)
- ◆ Virtuelle Caches mit physischen Tags werden häufig als first-level Caches eingesetzt (z.B. UltraSPARC II 16 KB direct mapped)

2 Virtually Indexed, Physically Tagged (3)

- Laufzeiteigenschaften:
 - ◆ Schnelle Kontextwechsel, Interrupt-Verarbeitung und Systemaufrufe, da ein Cache-Flush in den meisten Fällen vermieden wird.
 - ◆ Bei write-back ist über einen Kontextwechsel verzögertes Rückschreiben möglich.
 - ➔ Vermeidung von Schreiboperationen (Performance-Steigerung)
 - ➔ Variable Ausführungsdauer durch Compulsory-Misses je nach Referenzmuster des/der Vorgänger-Threads
 - ◆ Variable Ausführungsgeschwindigkeit bei dynamischer Speicherverwaltung durch Conflict-Misses
 - ◆ Variable Suchgeschwindigkeit durch Adreßumsetzung der MMU
 - ◆ Problematischer Einsatz in Multiprozessoren mit gemeinsamem Speicher (Welche Zeile soll invalidiert werden?)
 - Cache ist ein kleines Vielfaches der Seitengröße (Faktor 1-4). Dann müssen nur 1-4 Cache-Zeilen durch die Cache-Coherency Hardware invalidiert/geflushed werden.

4.8X.2 Physische Caches (Physically Indexed)

1 Organisation

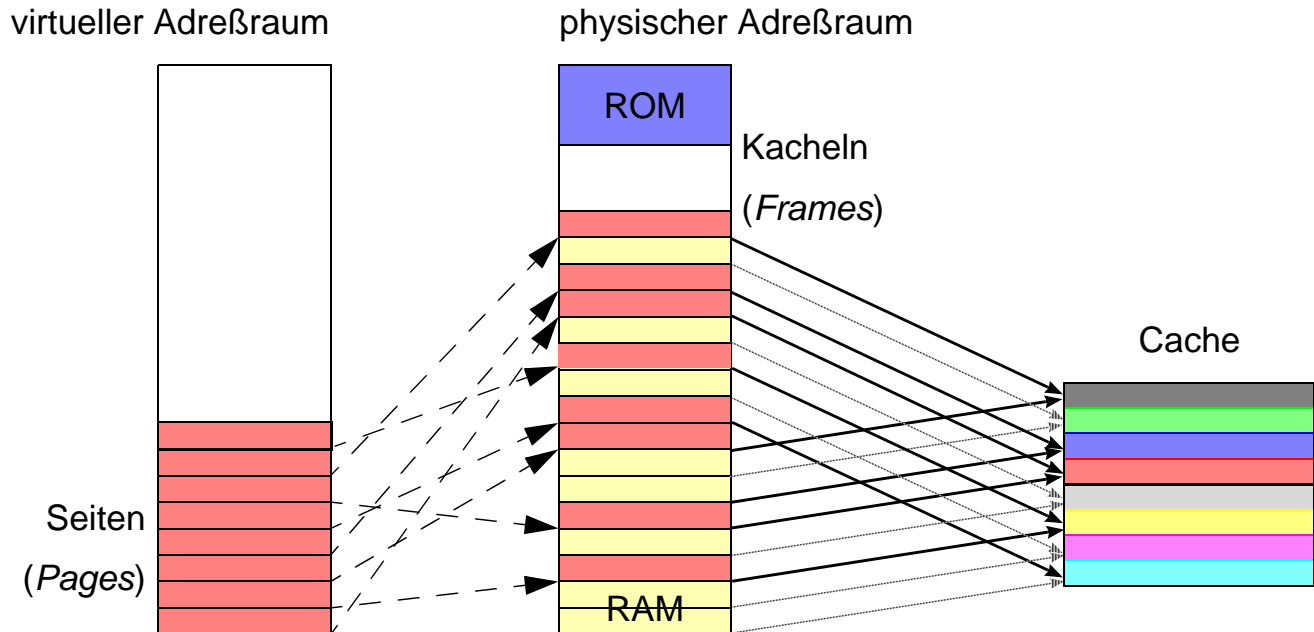


2 Vorteile

- Absolute Transparenz aus Sicht des Prozessors
- Keine performancekritische Systemunterstützung nötig (incl. E/A)
- Multiprozessoren mit gemeinsamem Adreßraum können mit einem Cache-Kohärenzprotokoll in Hardware aufgebaut werden.

3 Zugriffskonflikte

- ◆ Seitenkonflikte durch zufällige Anforderung von physischem Speicher:

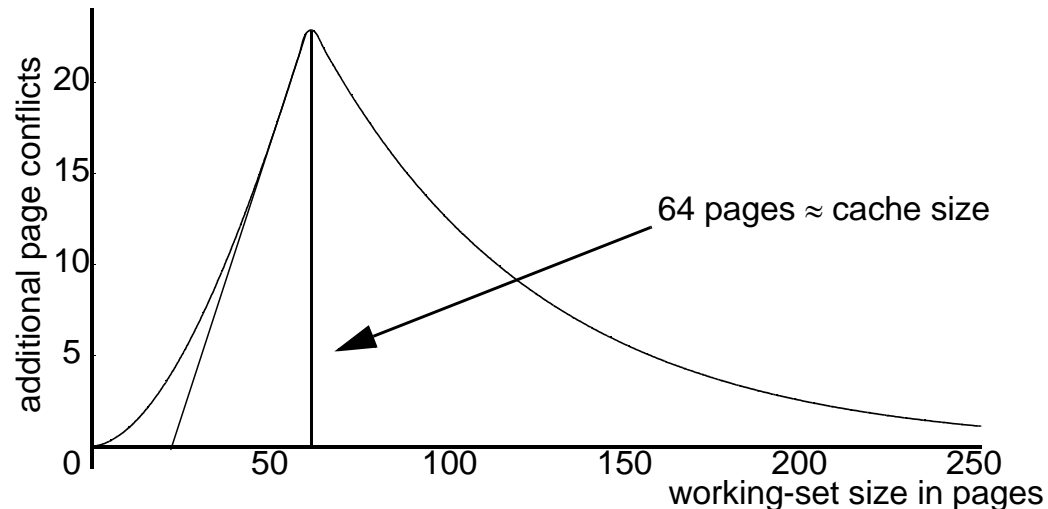


Zusammenhängender virtueller Speicher wird in der Regel auf beliebige freie physische Seiten abgebildet

3 Zugriffskonflikte (2)

- Auswirkungen des "random page coloring":
 - ➔ Es kommt zu Konflikten beim Cache-Zugriff
 - ➔ Nur ein Teil des Caches wird ausgenutzt
 - ➔ Erhebliche Laufzeitschwankungen!

$$X(p \text{ in bin}) = \binom{P}{p} \left(\frac{1}{C}\right)^p \left(1 - \frac{1}{C}\right)^{P-p}$$



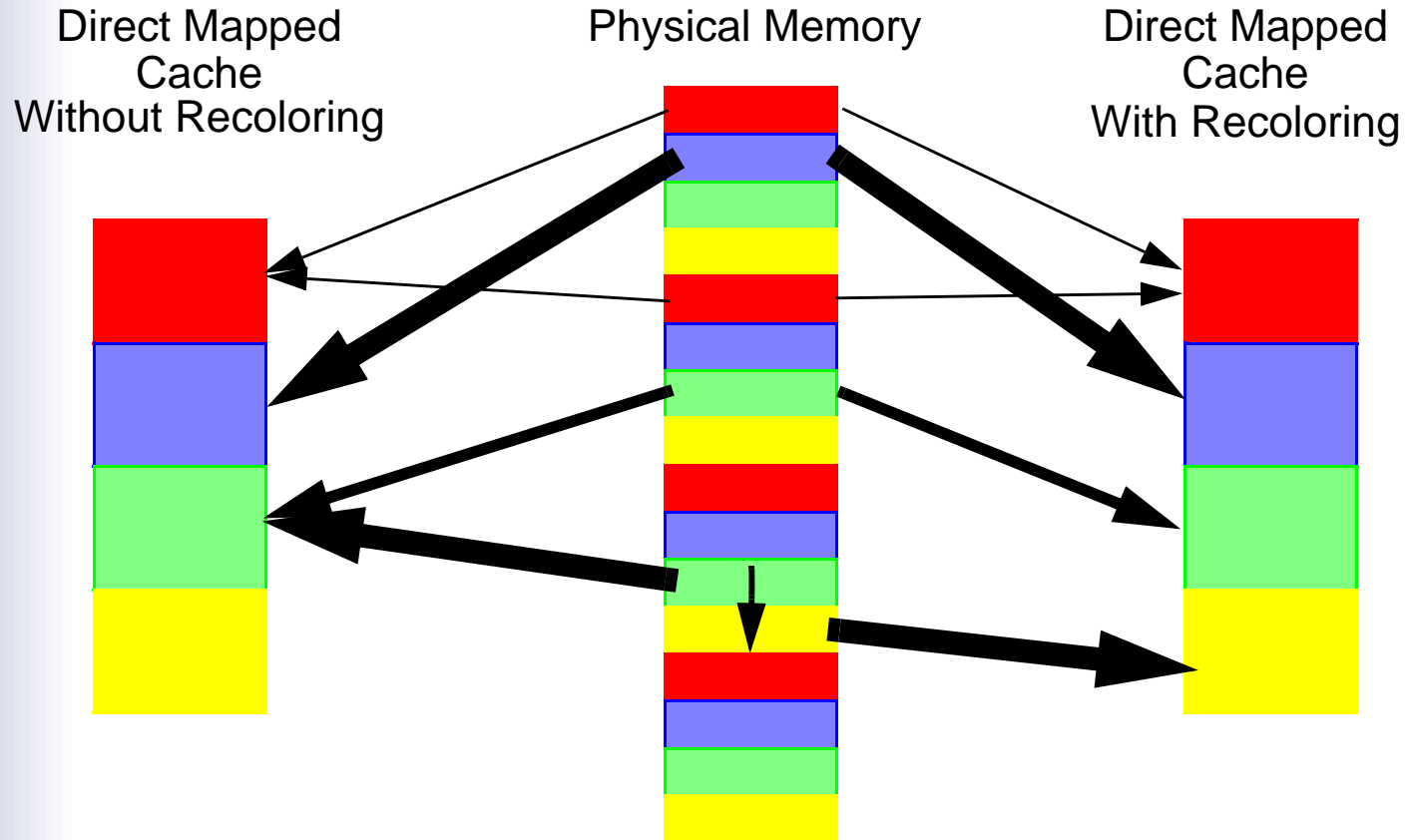
- ◆ Abhilfe: "Frischen" Speicher anfordern und im Speicher festhalten (mlock)

3 Zugriffskonflikte (3)

- Vermeidung von Konflikten beim Seitenzugriff
 - ◆ Aufeinanderfolgende Page-Colors für einzelne allokierte Segmente
 - Beispiel: rot-gelb-grün-blau-rot-gelb.....
 - ◆ Cache Partitioning
 - Aufteilung des physischen Speichers in disjunkte Teilmengen. Alle Seiten einer Teilmenge werden auf eine Partition des Caches abgebildet.
 - Beispiel
 - Alle roten und blauen Seiten für das Betriebssystem
 - Alle gelben Seiten für "die" Echtzeitandwendung
 - Alle grünen Seiten für Hintergrundprozesse
 - Hintergrundliteratur:
Liedke: *"OS-Controlled Cache Predictability for Real-Time Systems"*

3 Zugriffskonflikte (4)

- Vermeidung von Konflikten beim Seitenzugriff
 - ◆ Analyse des Zugriffsmusters und Page-Recoloring



3 Zugriffskonflikte (5)

- Einfluß der Zeilenlänge des Caches auf die Datenstrukturen
 - ◆ Problem: Mehrfache Fehlzugriffe durch zeilenübergreifende 'non aligned' Datenstrukturen
 - ◆ Lösung: Auffüllen der Strukturen auf Vielfache der Zeilenlänge (Padding)

Zeile

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

Zeile

0	Eintrag 0	unbenutzt
1	Eintrag 1	unbenutzt
2	Eintrag 2	unbenutzt
3	Eintrag 3	unbenutzt
4	Eintrag 4	unbenutzt

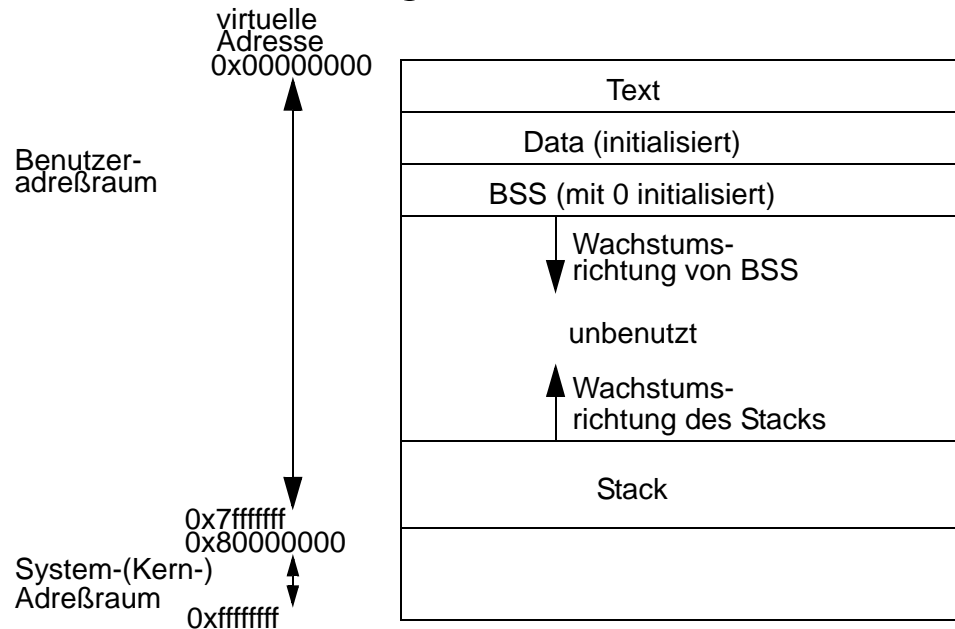
4 Laufzeiteigenschaften

- ◆ Variable Ausführungsgeschwindigkeit durch
 - Conflict Misses abhängig vom Page Coloring
- ◆ Variable Ausführungsgeschwindigkeit bei Write-Back durch
 - Compulsory Misses
- ◆ Variable Suchgeschwindigkeit durch Adreßumsetzung der MMU

4.8X.3 Virtuelle Adressierung

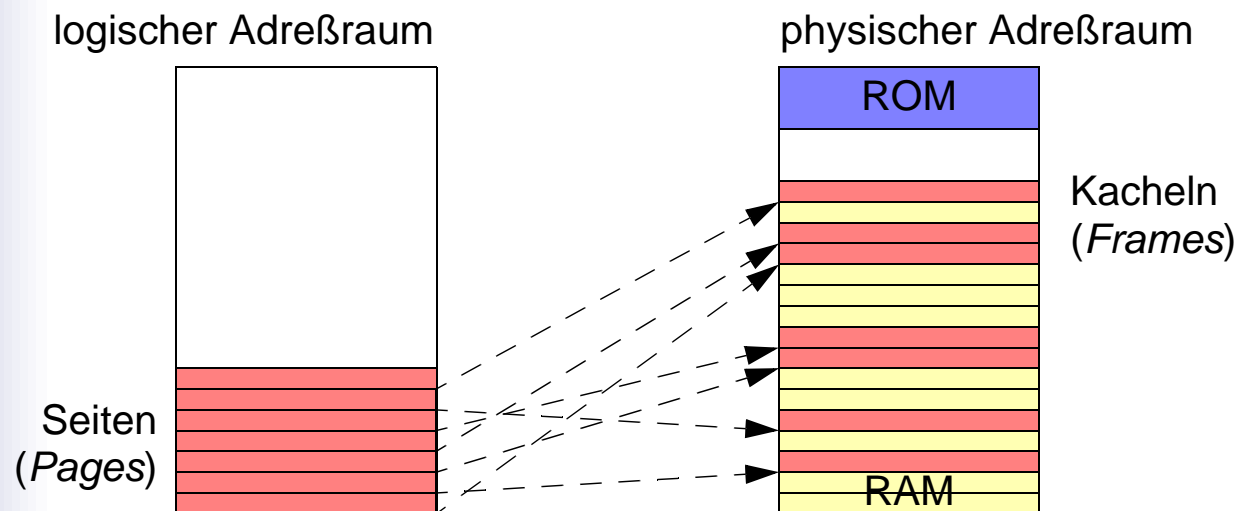
1 Grundlagen

- Jeder Prozeß besitzt eigenen virtuellen Adreßraum



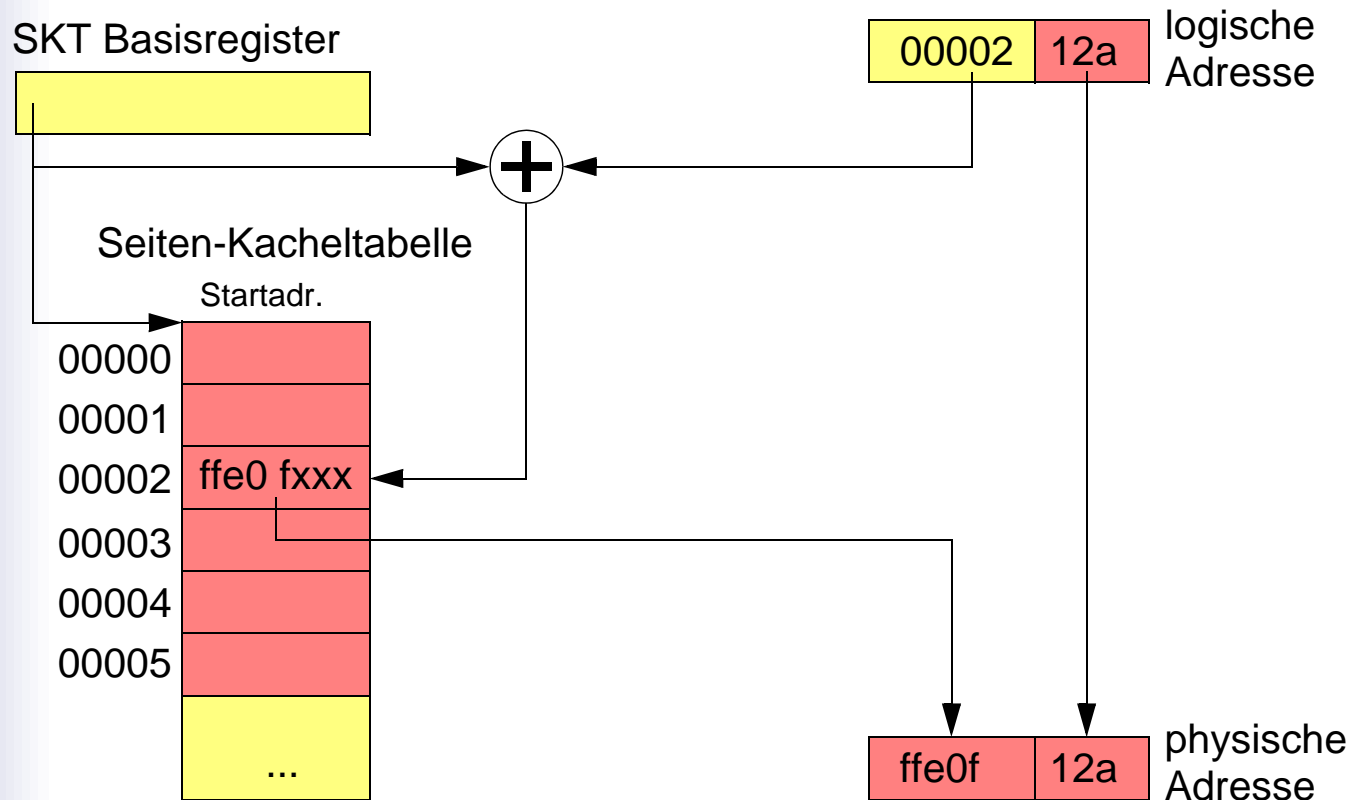
1 Grundlagen der virtuellen Adressierung (3)

- Einteilung des logischen Adreßraums in gleichgroße Seiten, die an beliebigen Stellen im physischen Adreßraum liegen können
 - ◆ Lösung des Fragmentierungsproblem
 - ◆ keine Kompaktifizierung mehr nötig
 - ◆ Vereinfacht Speicherbelegung und Ein-, Auslagerungen



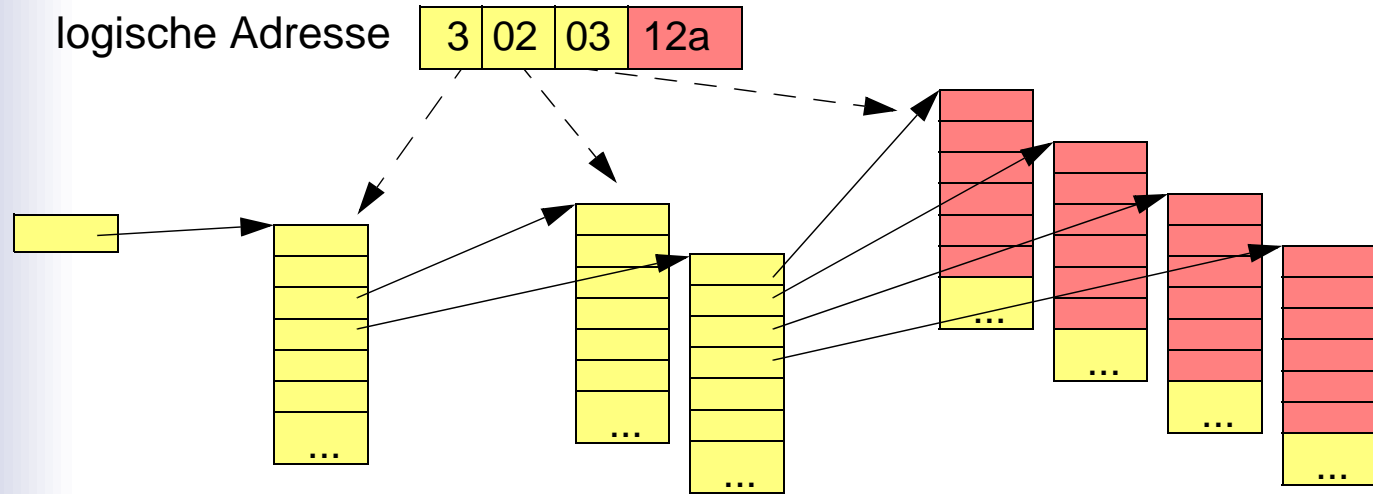
1 Grundlagen der virtuellen Adressierung (4)

- Tabelle setzt Seiten in Kacheln um

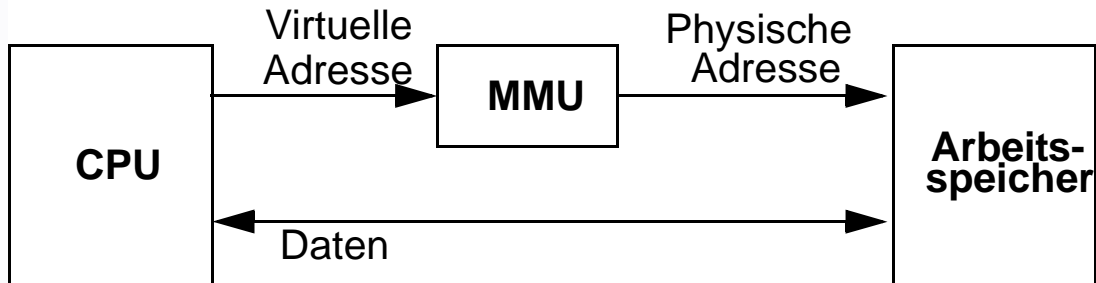


1 Grundlagen der virtuellen Adressierung (5)

■ Mehrstufige Seitenadressierung



■ MMU führt Adreßabbildung in Hardware durch

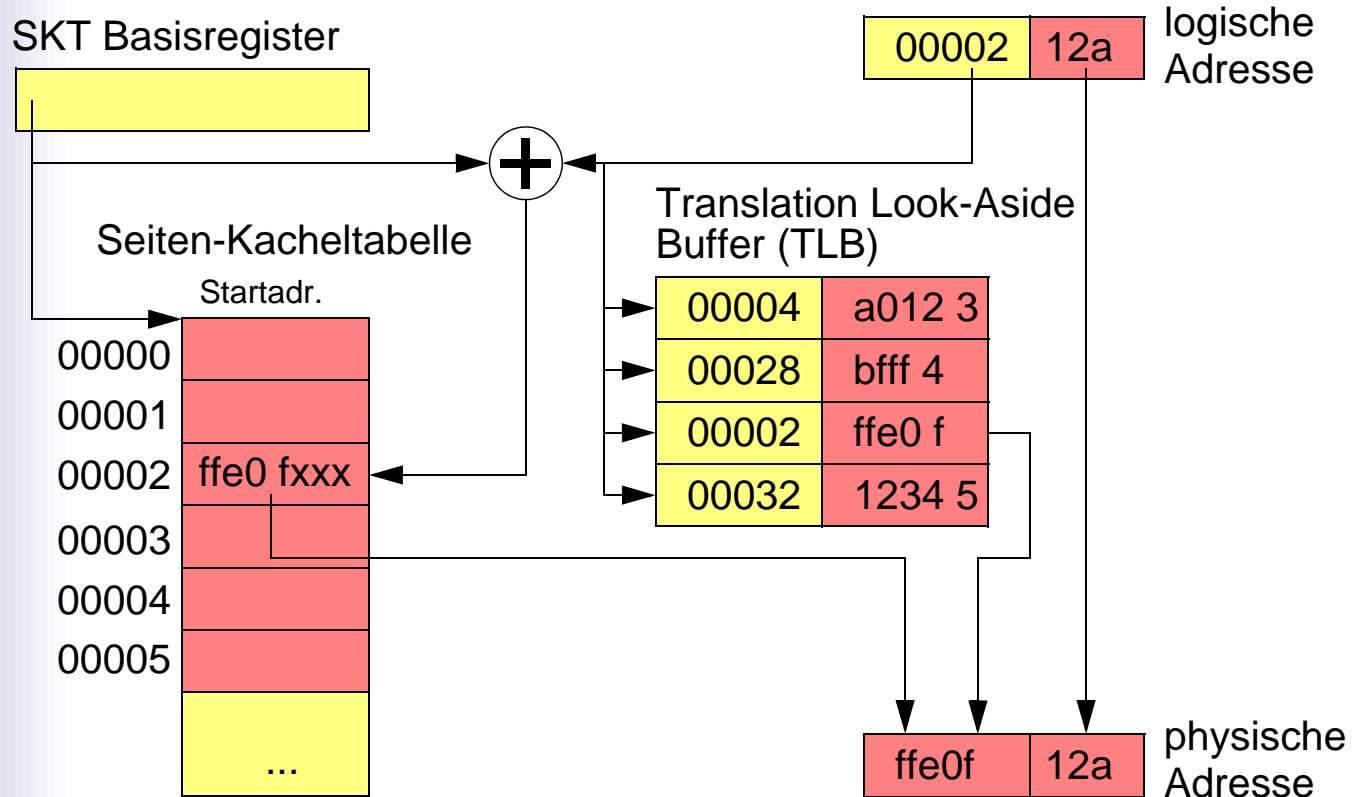


1 Grundlagen der virtuellen Adressierung (6)

- Seitenadressierung erzeugt internen Verschnitt
 - ◆ letzte Seite eventuell nicht vollständig genutzt
 - Seitengröße
 - ◆ kleine Seiten verringern internen Verschnitt, vergrößern aber die Seiten-Kacheltabelle (und umgekehrt)
 - ◆ übliche Größen: 512 Bytes — 8192 Bytes (8192 Bytes bei UltraSPARC)
 - ◆ Systemaufruf: `getpagesize()`
- viele implizite Speicherzugriffe nötig

2 Translation Look-Aside Buffer TLB

- Schneller Registersatz wird konsultiert bevor auf die SKT zugegriffen wird



2 Translation Look-Aside Buffer (2)

- Schneller Zugriff (< 1 Taktzyklus) auf Seitenabbildung, falls Information im voll-assoziativen Speicher des TLB
 - ◆ keine impliziten Speicherzugriffe nötig
- Bei Zugriffen auf eine nicht im TLB enthaltene Seite wird die entsprechende Zugriffsinformation in den TLB eingetragen
 - ◆ Ein alter Eintrag muß zur Ersetzung ausgesucht werden
 - Random
 - Round Robin
- TLB Größe
 - ◆ Pentium: Daten TLB = 64, Code TLB = 32, Seitengröße 4K
 - ◆ Sparc V9: Daten TLB = 64, Code TLB = 64, Seitengröße 8K
 - ◆ Größere TLBs bei den üblichen Taktraten zur Zeit nicht möglich

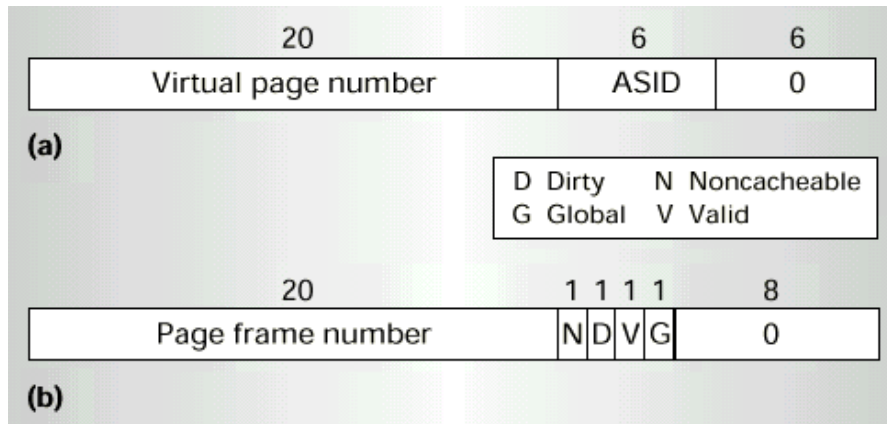
2 Translation Look-Aside Buffer (3)

- Der TLB kann nur einen kleinen Teil des Adreßraums abbilden.
(z.B. UltraSPARC Ili 64 Einträge x 8 KB = 512 KB)
- Ein TLB-Zugriffsfehler wird in der Regel durch Software aufgelöst.
(Durchsuchen der SKT und Einlagerung im TLB; < 32 Instruktionen).
 - ◆ Vorteil: Flexibles Design der SKT möglich
 - ◆ Nachteil: 1 Trap ist nötig
(Overhead of precise exceptions in pipelined/superscalar CPUs)
 - ◆ Ausnahmen: Intel x86, teilw. Motorola/IBM PowerPC, HP PA-RISC)
- TLB-Zugriffsfehler erscheinen nicht in der OS-Statistik.
- 1 TLB-Zugriffsfehler = 2 Speicherzugriffe (bis zu 1000 Zyklen!)
- Schon eine Arbeitsmenge von 8KB kann den TLB überfordern.

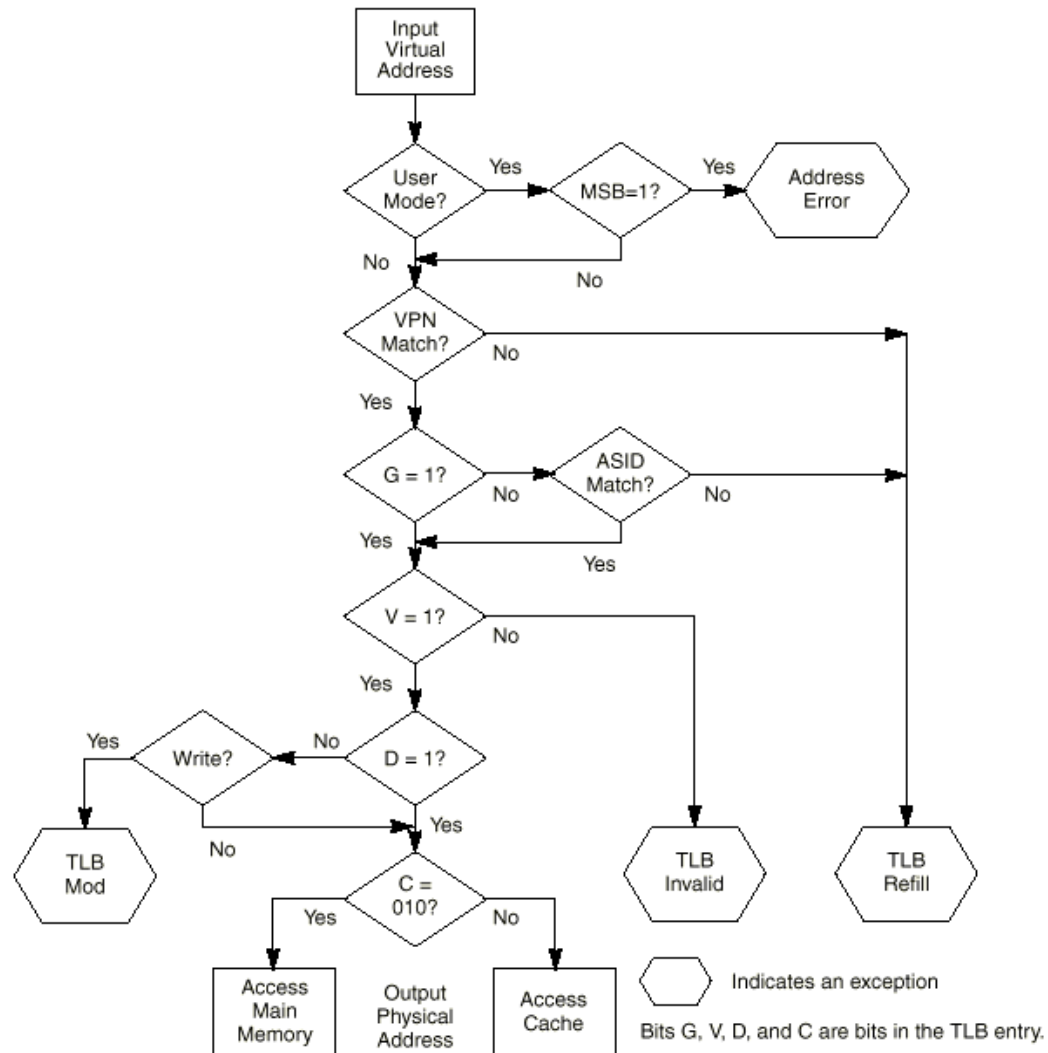
3 TLB Refill

■ Beispiel: MIPS R3000

◆ TLB-Register: EntryHi (a) und EntryLo (b)

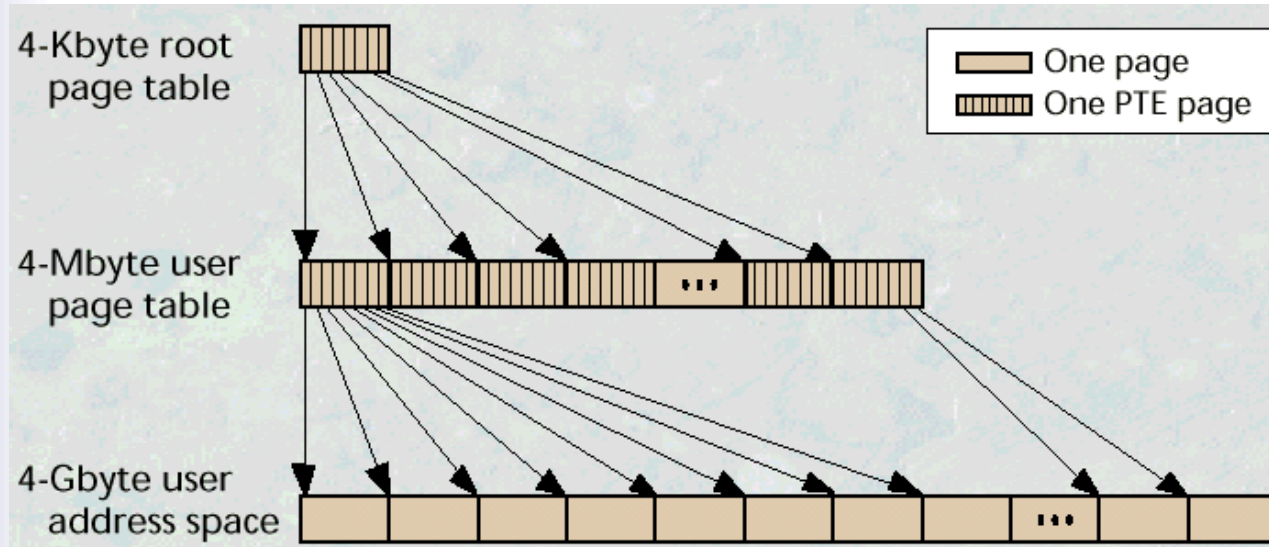


◆ Adressumsetzung mit TLB



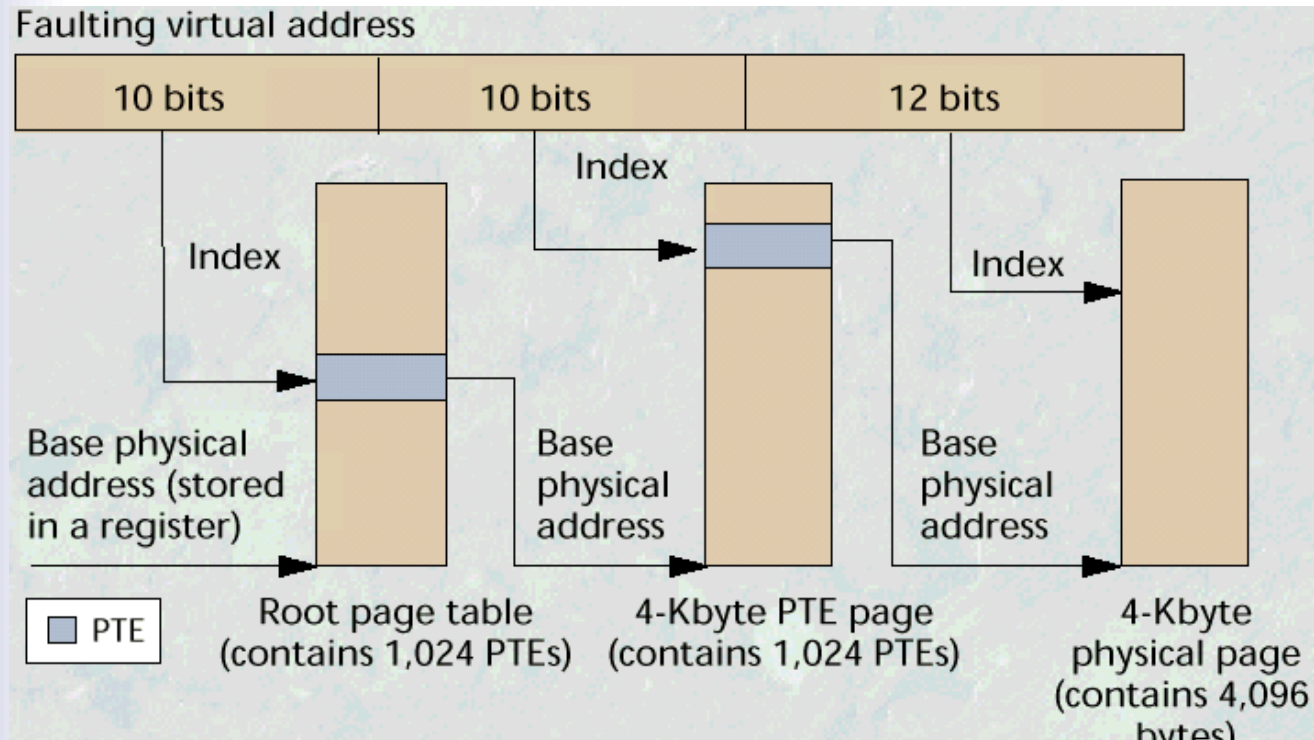
4 Page Table Walk

■ Hierarchical Page Table



4 Page Table Walk (2)

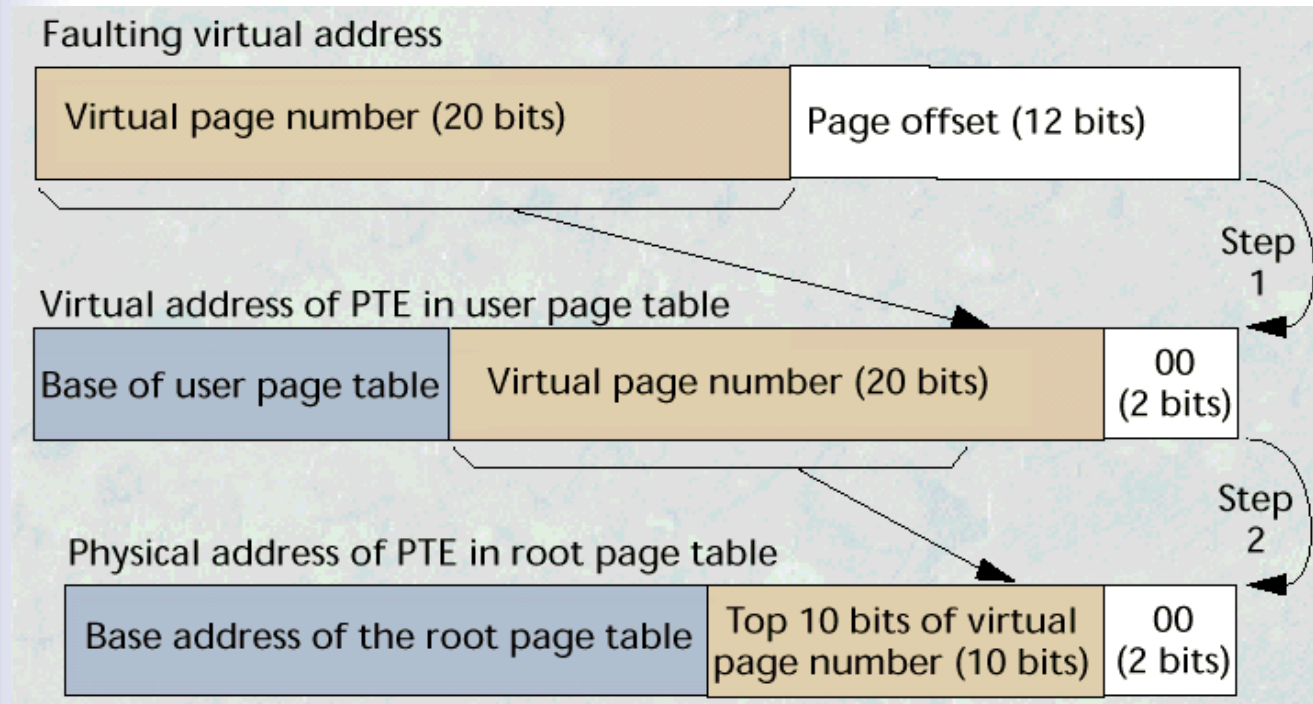
■ Top-Down Traversal



◆ Beispiel: Intel Pentium hardware table walk

4 Page Table Walk (3)

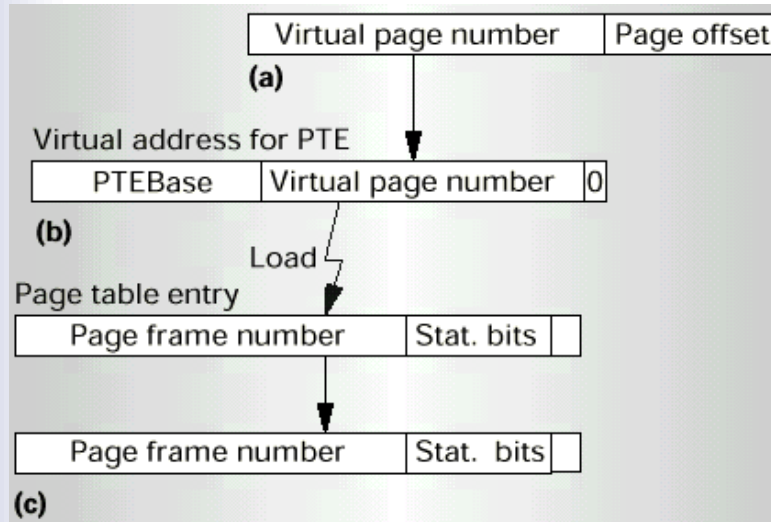
■ Bottom-Up Traversal



◆ Beispiel: MIPS, Alpha page table walk

4 Page Table Walk (4)

■ Hardware-Support für Bottom-up Traversal (MIPS R3000)

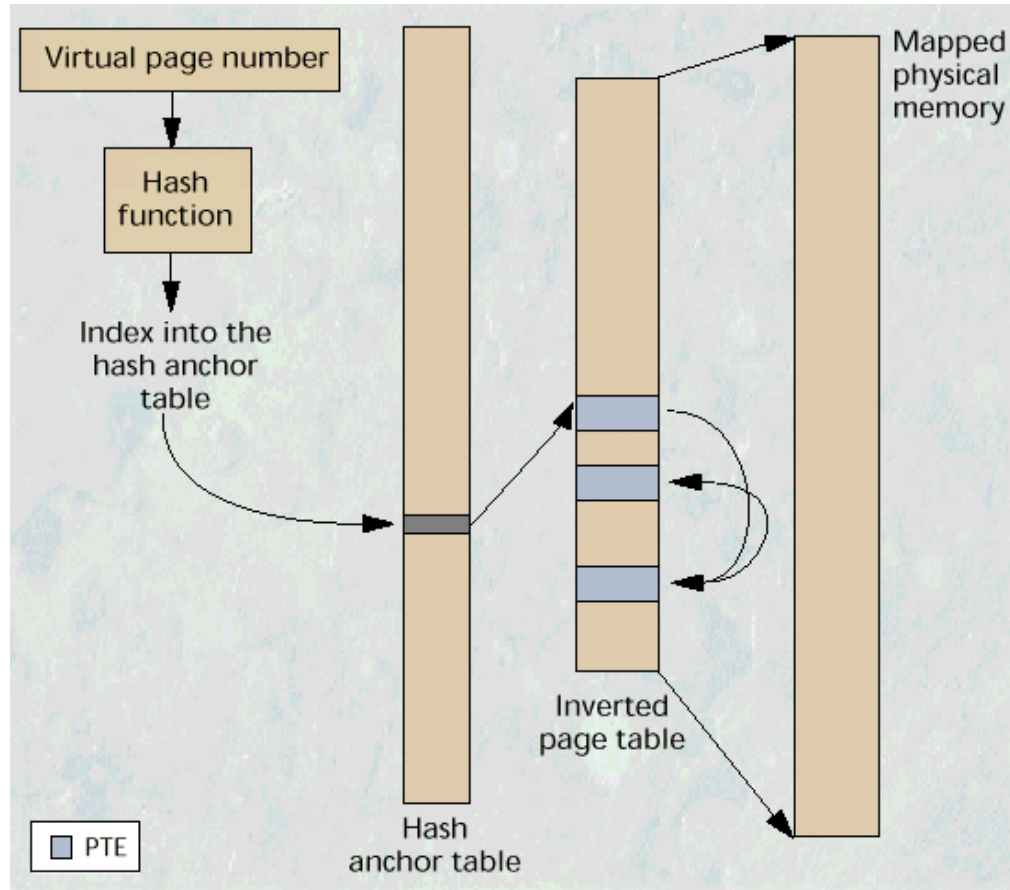


```

mfc0 k0,tlbcxt    #move the contents of TLB
                  #context register into k0
mfc0 k1,epc      #move PC of faulting load
                  #instruction into k1
lw k0,0(k0)      #load thru address that was
                  #inTLB context register
mtc0 k0,entry_lo #move the loaded value
                  #into the EntryLo register
tlbwr            #write entry into the TLB
                  #at a random slot number
j k1             #jump to PC of faulting
                  #load instruction to retry
rfe              #RESTORE FROM
                  #EXCEPTION
  
```

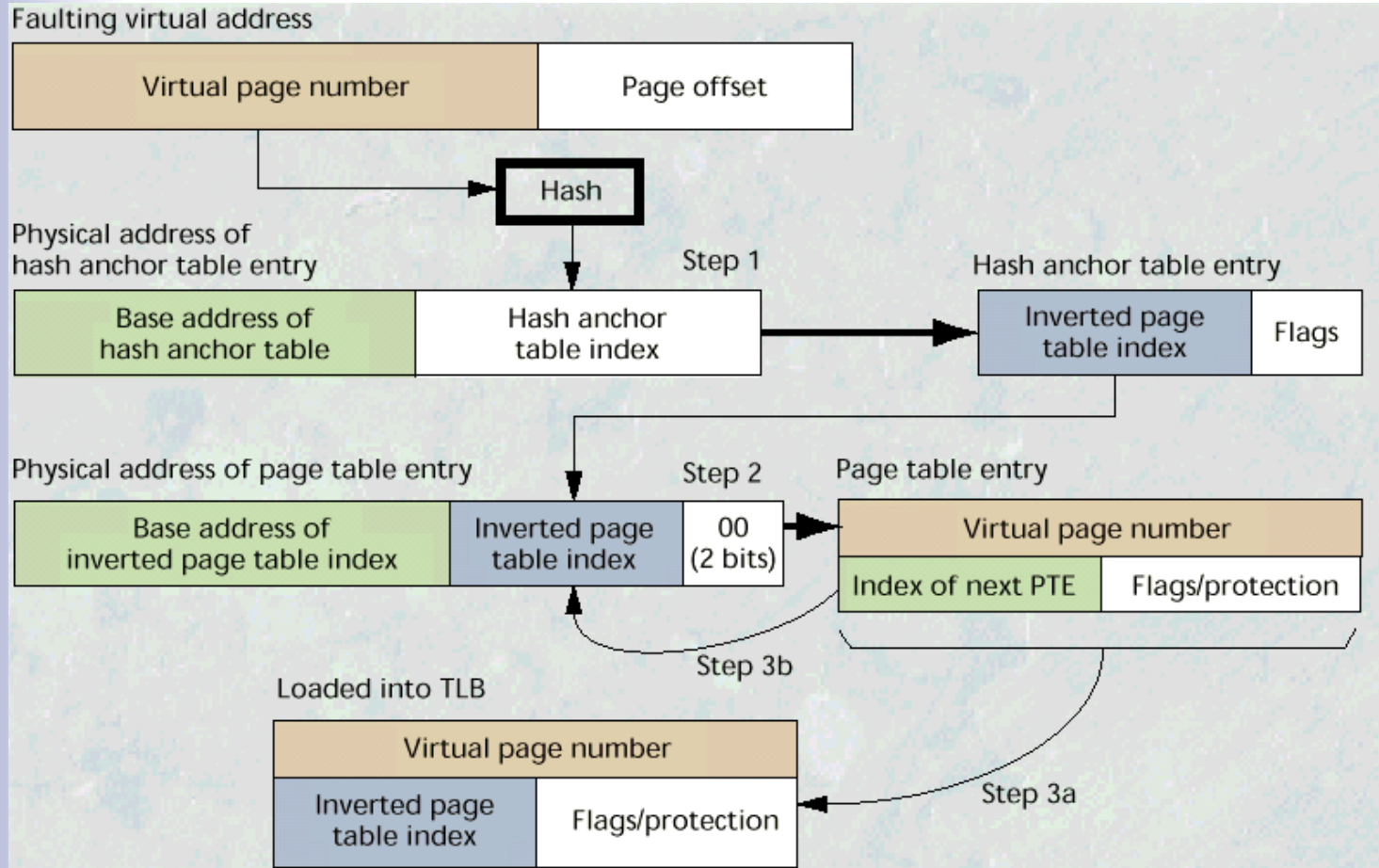
4 Page Table Walk (5)

■ Hashed Inverted Page Table



4 Page Table Walk (6)

■ Hashed Inverted Page Table (2)



5 Laufzeitverhalten

- Einflußfaktoren
 - ◆ Größe der Arbeitsmenge in Seiten
 - Deckt der TLB die Arbeitsmenge ab?
 - Kann man einzelne kritische Seiten im TLB einfrieren (z.B. bei MIPS)?
 - ◆ Ist die Zeit für das Laden des TLB deterministisch?
Gibt es eine Obergrenze?
 - ◆ Wie gut ist die Referenzlokalität der Seitentabelle, so daß der TLB überwiegend aus dem Cache nachgeladen werden kann?
 - ◆ ...