

# Betriebssysteme

## Tutorium 6

Philipp Kirchhofer

`philipp.kirchhofer@student.kit.edu`

`http://www.stud.uni-karlsruhe.de/~uxbtt/`

**Lehrstuhl Systemarchitektur  
Universität Karlsruhe (TH)**

16. Dezember 2009

# Was machen wir heute?

## 1 Tutorien Übungsblatt

- Memory Management Basics
- Memory Allocation Policies
- Memory Management Data Structures
- Segmentation
- Paging
- Segmentation and Paging

## Frage 8.1.a

Was sind die Unterschiede zwischen interner und externer Fragmentierung?

## Interne Fragmentierung (Verschnitt)

Betriebssysteme bieten häufig nur Vielfache von festen Größen als Speicherblockgröße an. Die Differenz zwischen dem angeforderten und dem vom Betriebssystem tatsächlich belegten Speicher kann nicht genutzt werden und wird als „Interne Fragmentierung“ bezeichnet.

## Beispiel: Interne Fragmentierung (Seiteneinteilung)

Seitengröße: 8 KB

Speicheranforderung: 20 KB

Tatsächlich belegter Speicher: 3 Seiten zu 8 KB = 24 KB

## Externe Fragmentierung

Es gibt auch Speicherverwaltungen, die Speicherblöcke beliebiger Größe verwalten können. Aufgrund der unterschiedlichen Lebenszeit der angeforderten Speicherblöcke kann es passieren, dass für eine Speicheranforderung nicht mehr genügend freier linearer Speicher zur Verfügung steht, obwohl die Gesamtzahl des freien Speichers für die Anforderung ausreichen würde.

## Frage 8.1.b

Was ist der Unterschied zwischen einer logischen und einer physikalischen Adresse?

## Antwort

Ein Unterschied existiert nur bei Systemen, die keine 1:1 Abbildung von Anwendungsadressen zu physikalischen Adressen haben.

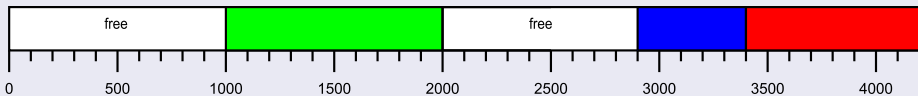
Adressen in einem Programm sind logische Adressen. Physikalische Adressen werden über den Speicherbus an die Speicherchips geschickt, um einzelne Speicherzellen auszulesen.

Jedes Anwendungsprogramm gehört zu einem logischen Adressraum. Alle Adressen im Anwendungsprogramm sind logische Adressen, d.h. das Programm arbeitet nicht direkt mit physikalischen Adressen.

Während der Ausführung müssen Teile des Programms in den physikalischen Speicher geschrieben werden. Eine „Memory Management Unit“ (MMU) übersetzt bei einem Speicherzugriff logische in physikalische Adressen.

# Tutorien Übungsblatt - Memory Allocation Policies

Es sei ein System gegeben mit 4200 Speicherzellen, davon sind 1000 Zellen an Position 1000, 500 Zellen an Position 2900 und 800 Zellen an Position 3400 belegt.

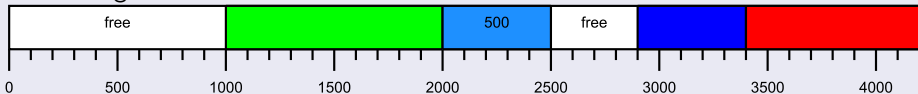


## Frage 8.2.a

Es sollen Speicherblöcke der Größen 500, 1200 und 200 nach der „Best-Fit“ Policy angefordert werden. Können alle Anforderungen erfüllt werden?

## Antwort

Nein. Der Block mit der Größe 1200 kann aufgrund von externer Fragmentierung nicht belegt werden.

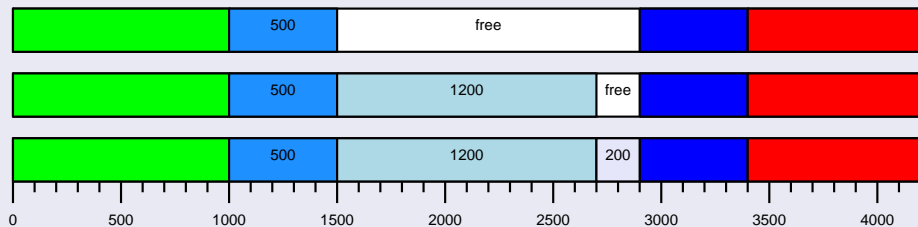


## Frage 8.2.b

Die externe Fragmentierung kann durch Verschieben von Blöcken verringert werden. Kann danach die Anforderung von 1200 Speicherzellen erfüllt werden?

## Antwort

Ja, die Anforderung kann erfüllt werden.



## Frage 8.3.a

Ein System habe  $128 \text{ MB} = 2^{27}$  Byte Speicher, der Speicher kann in Vielfachen von  $n$  Bytes angefordert werden. Eine verlinkte Liste habe pro Knoten ein 32 Bit Wort für die Adresse, die Länge und ein oder zwei Zeiger zum Vorgänger bzw. Nachfolger.

Wie verhält sich der Speicherbedarf einer Speicherverwaltung auf Bitmap-Basis im Vergleich zu einem System mit verlinkten Listen bei voller Speicherauslastung, minimaler Speicherauslastung und im Worst-Case?

## Bitmap

Eine Bitmap benötigt 1 Bit pro Zuordnungseinheit. Bei  $\frac{2^{27}}{n}$  Zuordnungseinheiten belegt die Bitmap also  $\frac{2^{24}}{n}$  Bytes. Der Speicherbedarf der Bitmap ist unabhängig vom Belegungsgrad des Speichers.

## Verlinkte Liste

Bei  $a$  belegten und  $f$  freien Speicherblöcken besitzt die verlinkte Liste  $a + f$  Knoten.

### Leerer Speicher

Wenn kein Speicher belegt ist belegt die verlinkte Liste nur einen Knoten, also 12 bzw. 16 Bytes.

### Voller Speicher

Bei voller Speicherauslastung belegt die verlinkte Liste ebenfalls nur einen Knoten.

### Worst-Case

Im Worst-Case ist der Speicher zur Hälfte belegt und zur Hälfte frei. In diesem Fall werden pro Block ein Knoten benötigt.

Die Speicherbelegung ergibt sich also zu:

$$a = \frac{2^{26}}{n} \text{ und } f = \frac{2^{26}}{n}$$

$$\Rightarrow 12 * \frac{2^{27}}{n} \text{ Bytes bzw. } 16 * \frac{2^{27}}{n} = \frac{2^{31}}{n} \text{ Bytes}$$

## Zusammenfassung

Der Einsatz von verlinkten Listen lohnt sich, wenn folgende Bedingungen vorliegen:

- Feingranulare Anforderungen sollen möglich sein
- Es gibt viele große Anforderungen

Es können allerdings zusätzliche Verzögerungen u.a. durch den nötigen Einsatz von Vereinigungsalgorithmen auftreten.

## Frage 8.4

Wie funktioniert Segmentierung?

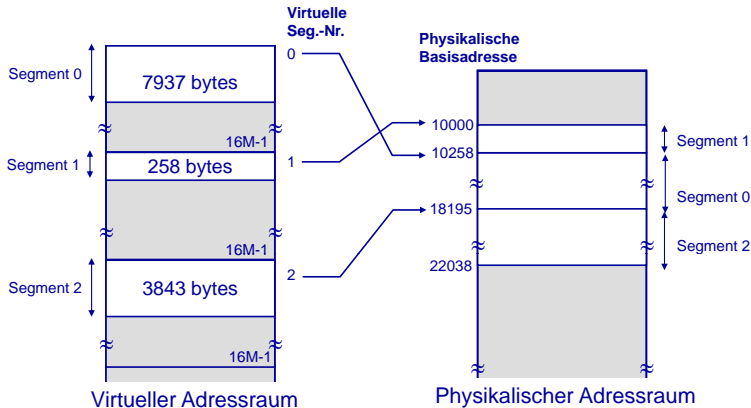
## Antwort

Bei dem Einsatz von Segmentierung besteht ein logischer Adressraum aus verschieden großen Segmenten. Jedes Segment steht für einen Bereich des Programms, z.B. Code, Daten, Stack usw.

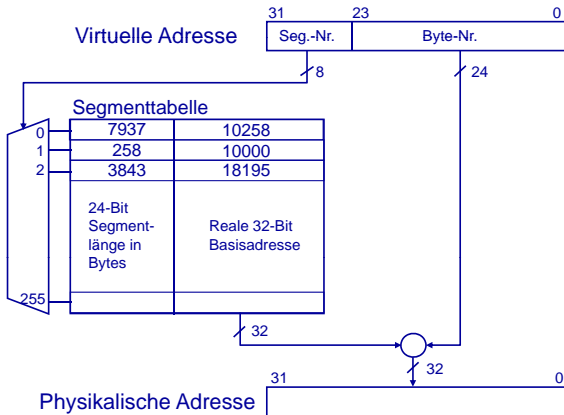
Die Umsetzung von logischen in physikalische Adressen geschieht mit Hilfe einer Segmentierungstabelle.

Weitere Informationen stehen auf den folgenden Folien (aus TI II bzw. Rechnerorganisation)

# Virtueller und physikalischer Adressraum



# Segmentbasierte Speicherverwaltung



## Frage 8.5.a

Wie funktioniert Paging?

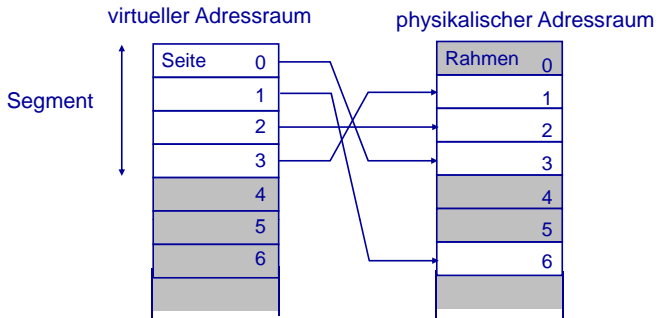
## Antwort

Der logische und physikalische Adressraum wird in gleichgroße Blöcke unterteilt. Logische Blöcke werden Seiten (pages), Physikalische Blöcke Rahmen (frames) genannt.

Jede Seite kann mit Hilfe einer Seitentabelle (pagetable) auf einen beliebigen Rahmen abgebildet werden.

Ein großer Vorteil von Paging ist, dass Programmteile nicht mehr am Stück im physikalischen Speicher liegen müssen.

# Virtueller und physikalischer Adressraum



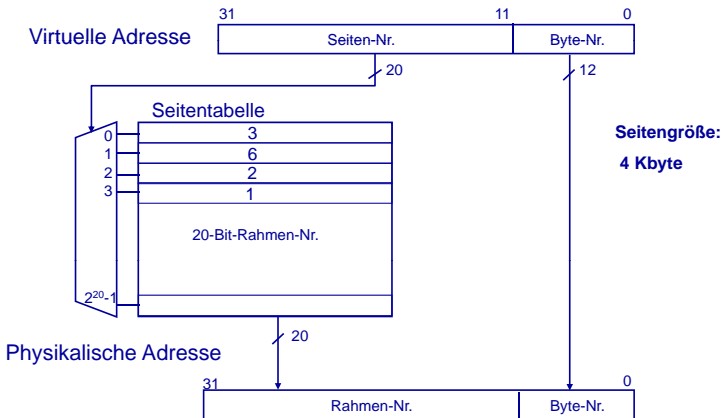
## Frage 8.5.b

Wie werden logische Adressen mit einer einstufigen Seitentabelle auf physikalische Adressen abgebildet?

## Antwort

Jede logische Adresse besteht aus zwei Teilen, einer Seitennummer und einem Seitenoffset. Die Seitennummer zeigt auf einen Eintrag in der Seitentabelle. Der Eintrag enthält die zugehörige Rahmennummer. Diese Rahmennummer wird mit der Blockgröße multipliziert und bildet addiert mit dem Seitenoffset die physikalische Adresse.

# Seitenwechsel (Paging)



## Frage 8.5.c

Welcher Nachteil haben einstufige Seitentabellen?

## Antwort

Die Größe einer Seitentabelle wächst linear mit der Größe des Adressraums.

## Beispiel: Minimale Größe von einstufigen Seitentabellen

Seitengröße: 4 KB =  $2^{12}$  Byte

32 Bit Adressraum:  $\frac{2^{32} \text{ Adressen}}{2^{12} \text{ Byte pro Seite}} = 2^{20}$  Seiten \* 20 Bit pro Seite =  $\sim 1,5$  MB

Seitengröße: 4 MB =  $2^{22}$  Byte

64 Bit Adressraum:  $\frac{2^{64} \text{ Adressen}}{2^{22} \text{ Byte pro Seite}} = 2^{42}$  Seiten \* 42 Bit pro Seite =  $\sim 168$  GB

## Frage 8.5.d

Es sei ein 32 Bit System mit einer Seitengröße von 4 KB gegeben.  
Weiterhin sei ein Seitentableneintrag (PTE, Page Table Entry) 4 Byte groß.  
Wie groß ist eine einstufige Seitentabelle?

## Antwort

Der logische Adressraum enthält  $2^{32}$  Adressen.

Jede Seite enthält 4 KB =  $2^{12}$  Bytes.

Damit enthält der logische Adressraum  $\frac{2^{32} \text{ Adressen}}{2^{12} \text{ Byte pro Seite}} = 2^{20}$  Seiten.

Jede Seite benötigt einen Seitentableneintrag, insgesamt ergibt sich ein  
Gesamtspeicherverbrauch von  $2^{20}$  Seiten \* 4 Byte pro Seite = 4 MB.

## Frage 8.5.e

Welche Alternativen gibt es zu einer einstufigen Seitentabelle?

## Gehashte Seitentabellen

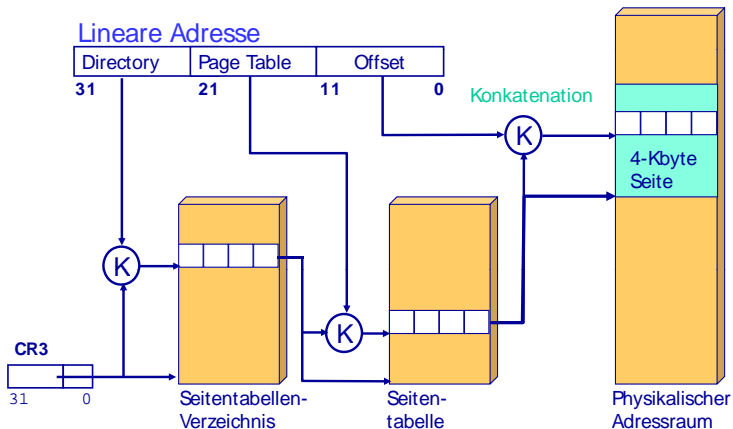
Die logische Adresse wird gehasht und als Zeiger in die Seitentabelle benutzt. Adressen mit Kollisionen müssen in eine verlinkte Liste eingetragen werden.

## Mehrstufige Seitentabelle

Die große einstufige Seitentabelle wird in kleinere Stücke aufgeteilt und in einer Baumstruktur angeordnet. Normalerweise werden Adressräume nicht voll ausgenutzt, das Erstellen von einzelnen Untertabellen kann deshalb gespart werden. Weiterhin können die Untertabellen z.B. auf eine Festplatte ausgelagert werden.

Ein Nachteil ist die Erhöhung der Anzahl an Speicherzugriffen, da für jeden Tabellenzugriff ein Speicherzugriff nötig ist.

# Zweistufiges Seitenwechsel-Verfahren



## Invertierte Seitentabelle

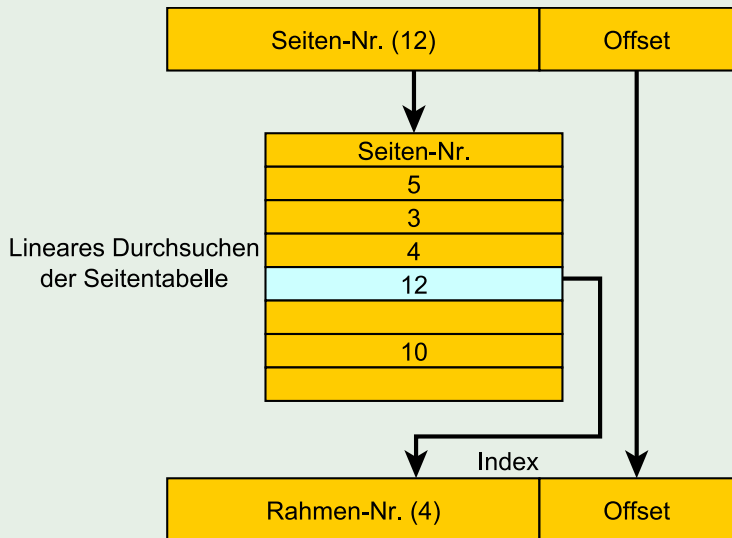
Normale Seitentabelle bildet von Logische Adresse  $\Rightarrow$  Physikalische Adresse ab.

Invertierte Seitentabelle bildet von Physikalische Adresse  $\Rightarrow$  Logische Adresse ab.

Es wird nur eine invertierte Seitentabelle für das gesamte System benötigt. Der Speicherbedarf der Seitentabelle hängt nur vom installierten physikalischen Speicher ab.

Ein Nachteil ist der hohe Zeitbedarf, da eine lineare Suche durch die Seitentabelle durchgeführt werden muss.

## Beispiel: Invertierte Seitentabelle



## Frage 8.5.f

Was ist ein TLB?

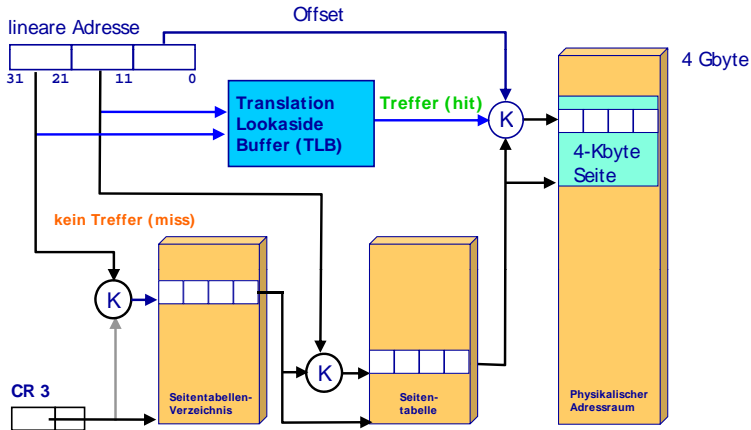
## Antwort

Ein TLB (Translation Lookaside Buffer) ist ein vollassoziativer Cache, der die Umsetzung von logischen in physikalische Adressen beschleunigt. Jeder Eintrag in einem TLB besteht aus einem Schlüssel (Seitennummer) und einem dazugehörigen Wert (Rahmennummer). Bei der Umsetzung einer logischen Adresse vergleicht der TLB parallel für alle Einträge die Seitennummer auf einen Treffer. Bei einem Treffer (Hit) wird die zugehörige Rahmennummer zurückgegeben. Falls kein Treffer vorliegt (Miss) muss die Seitentabelle durchlaufen werden.

Es gibt zwei Varianten von TLBs:

Bei **Hardwarekontrollierten TLBs** durchläuft der TLB bei einem Miss selbstständig die Seitentabelle und fügt die fehlende Abbildung in den TLB ein. Bei **Softwarekontrollierten TLBs** wird bei einem Miss eine Ausnahme ausgelöst und das Betriebssystem muss die fehlende Abbildung hinzufügen.

# Beschleunigung der Adressberechnung durch einen Cache



## Intel-Terminologie

**logische Adresse:** Eine Adresse bestehend aus Segment-Selektor und Offset. Wird mittels Segmentierung in eine lineare Adresse umgesetzt.

**lineare Adresse:** Linearer,  $2^{32}$  Byte großer Adressraum, der mittels Paging auf physischen Speicher umgesetzt wird

In der IA-32 Architektur gibt es zwei Phasen für die Adressumsetzung:

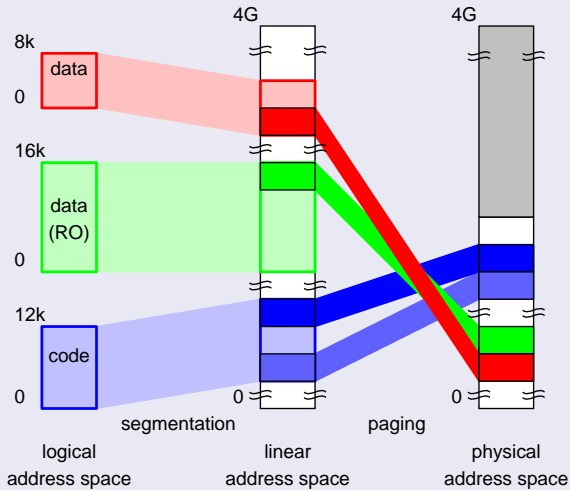
### 1. Phase: Segmentierung

Die erste Phase setzt segmentierte logische Adressen in lineare Adressen um. Dabei wird der Offset zur Segment-Basisadresse addiert und geprüft, ob die neue Adresse im Segment liegt.

### 2. Phase: Paging

Die zweite Phase setzt die lineare Adresse mit Hilfe einer zweistufigen Seitentabelle in eine physikalische Adresse um. Die Seitentabelle wird direkt von der Hardware ausgelesen.

## Segmentierung und Paging

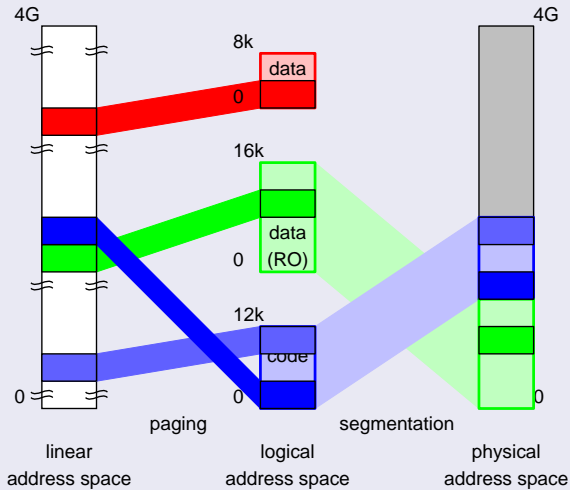


(a)

## Frage 8.6.a/b

Eine Alternative wäre erst das Paging und dann anschließend die Segmentierung durchzuführen. Welche Vor- und Nachteile ergeben sich zwischen der IA-32 Umsetzung und der genannten Alternative?

## Paging und Segmentierung



(b)

## Vorteile des IA-32 Schemas

- Compiler kann zusammengehörige Code- und Datenteile in Segmenten anordnen
- Zugriffsschutzsystem (read/write/execute) kann auf Segmentbasis sehr einfach implementiert werden
- Gruppierung von Programmteilen garantiert die nebeneinanderliegende Anordnung der einzelnen Teile im logischen und linearen Adressraum
- Keine externe Fragmentierung (durch Paging)

## Nachteile des Alternativen Schemas

- Segmente müssen vollständig in den physikalischen Speicher geladen werden
- Segmente müssen kontinuierlich im Speicher vorliegen  
⇒ Externe Fragmentierung

## Frage 8.6.c

Das CR3 Register enthält die physikalische Adresse der ersten Stufe der Seitentabelle. Weshalb ist das Ändern des CR3 Registers eine „teure“ Operation?

## Antwort

Das Ändern des CR3 Registers entspricht dem Laden einer neuen Seitentabelle. Dies geschieht z.B. bei einem Prozesswechsel. Durch das Laden einer neuen Seitentabelle hat sich die Umsetzung von linearen in physikalische Adressen geändert. Weiterhin müssen verschiedene Caches ungültig gemacht werden:

- TLB-Cache (wg. Änderung der Umsetzung von linear  $\Rightarrow$  physikalisch)
- L1-Cache (typischerweise linear indiziert)

Zusätzlich können Pipeline-Stalls auftreten, da bereits laufende Speicherzugriffe erst abgeschlossen werden müssen bevor eine neue Seitentabelle geladen werden kann.

## Frage 8.6.c

Durch welche Hardwaremodifikation kann man die negativen Auswirkungen abschwächen?

## Antwort

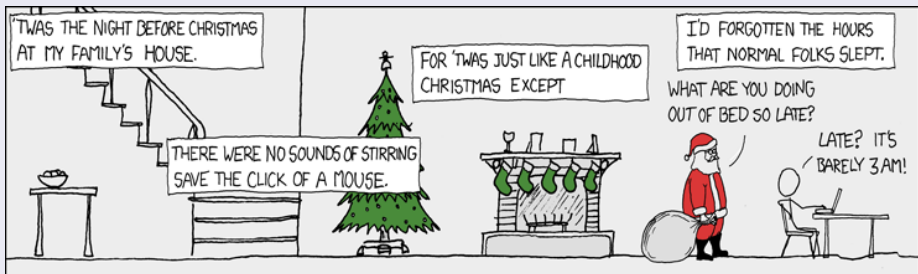
Eine Lösung ist der Einsatz von getaggtten TLB- und L1-Caches. Beim Einsatz von getaggtten Caches wird ein Tag, der für einen bestimmten Adressraum steht, an den Index angehängt. Dadurch können beim Wechsel des Adressraums die Einträge des TLB- und L1-Caches gültig bleiben.

Weiterhin können die Caches physikalisch indiziert werden. Dies bedeutet allerdings, dass bei jedem Zugriff auf einen Cache erst eine Zuordnung von lineare in physikalische Adressen durchgeführt werden muss. Der am häufigsten eintretende Fall, ein Cache-Hit, wird also verlangsamt.

Üblicherweise werden L1-Caches virtuell und L2- oder L3-Caches physikalisch indiziert, da während der Adressumsetzung für den Zugriff auf die L2- bzw. L3-Caches bereits die Prüfung des L1-Caches auf einen Treffer geschehen kann.

Fragen & Kommentare?

## Schöne Weihnachten und einen guten Rutsch!



xkcd: Christmas Back Home