Grundlagen der Rechnerarchitektur und Betriebssysteme

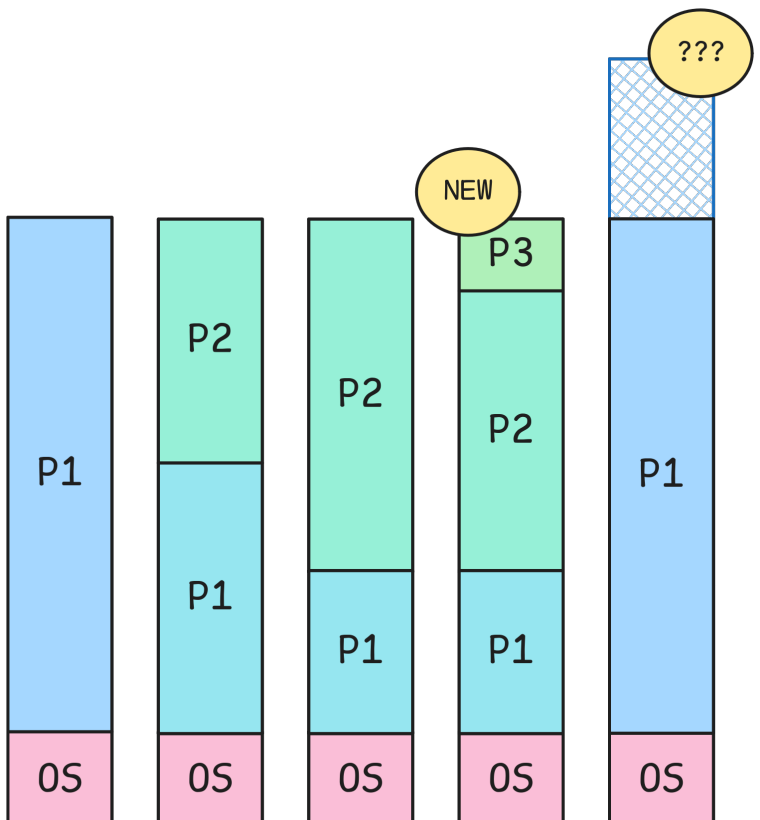
Virtueller Speicher

Florian Knoch · Lehrstuhl für Praktische Informatik,
insbes. Systemnahe Programmierung, Universität Bamberg

Einen Schritt zurück

Struktur des Hauptspeichers

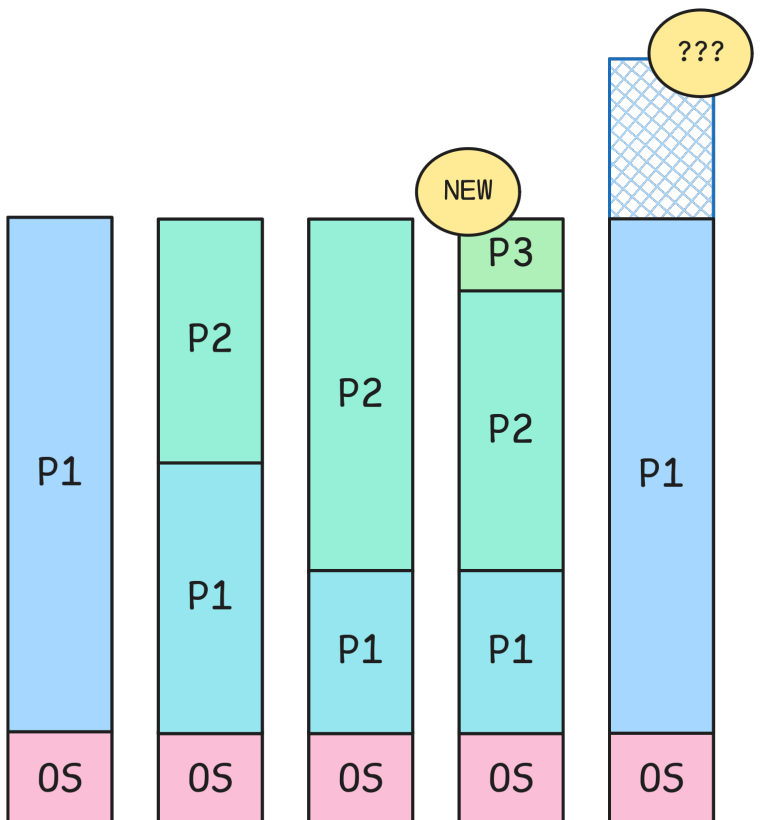
- Wie viel Platz bekommt die **Kernel-Seite**, wie viel die **User-Seite**?
-
-
-
-



Einen Schritt zurück

Struktur des Hauptspeichers

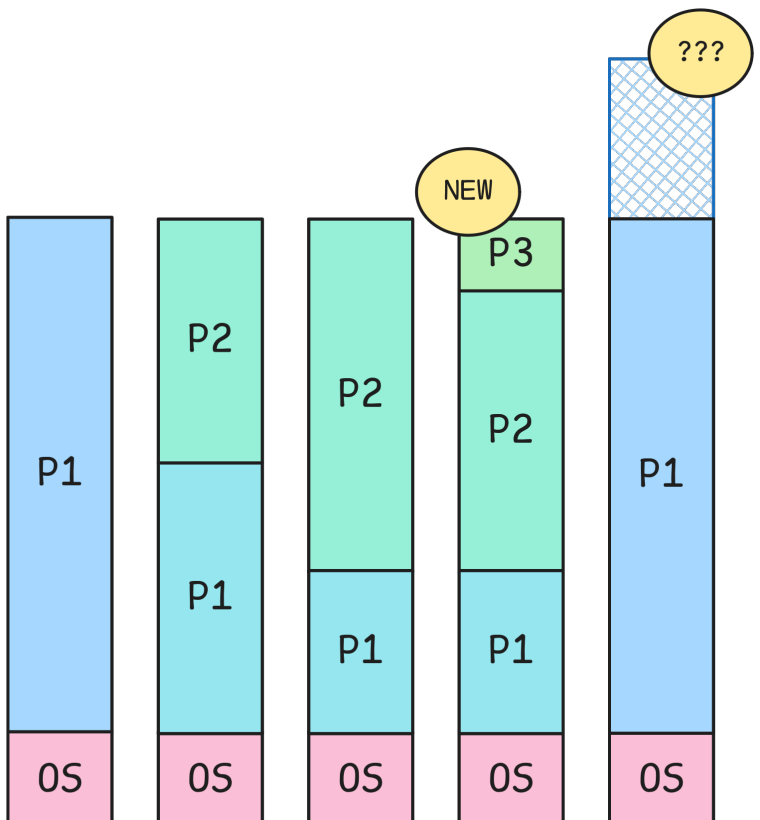
- Wie viel Platz bekommt die **Kernel-Seite**, wie viel die **User-Seite**?
- Was, wenn **mehrere User-Programme** gleichzeitig laufen sollen?
-
-
-



Einen Schritt zurück

Struktur des Hauptspeichers

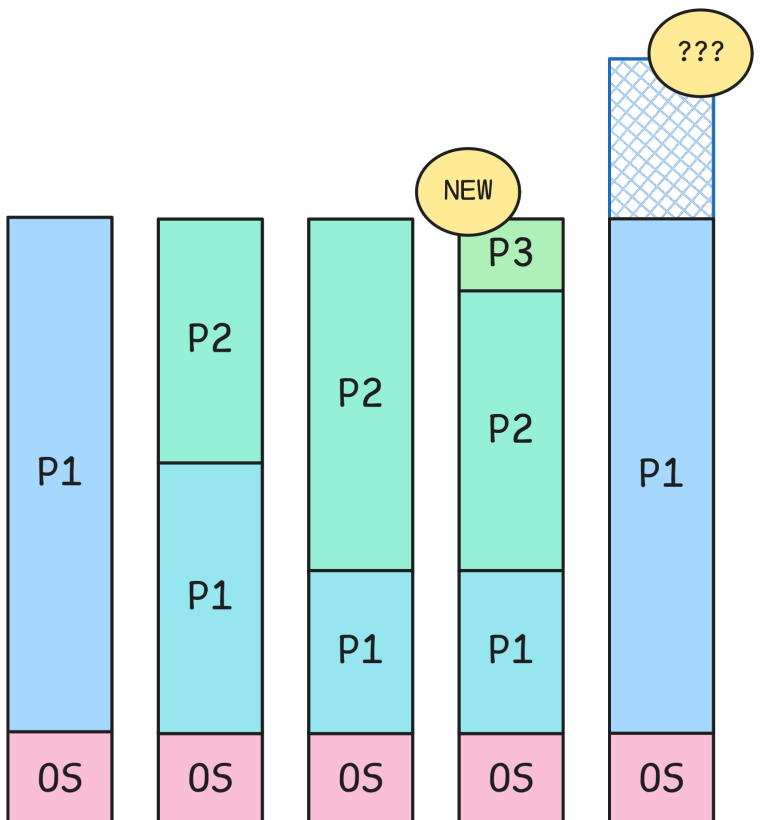
- Wie viel Platz bekommt die **Kernel-Seite**, wie viel die **User-Seite**?
- Was, wenn **mehrere User-Programme** gleichzeitig laufen sollen?
- Was, wenn die User-Programme **unterschiedlich viel Speicher** nutzen?
-
-



Einen Schritt zurück

Struktur des Hauptspeichers

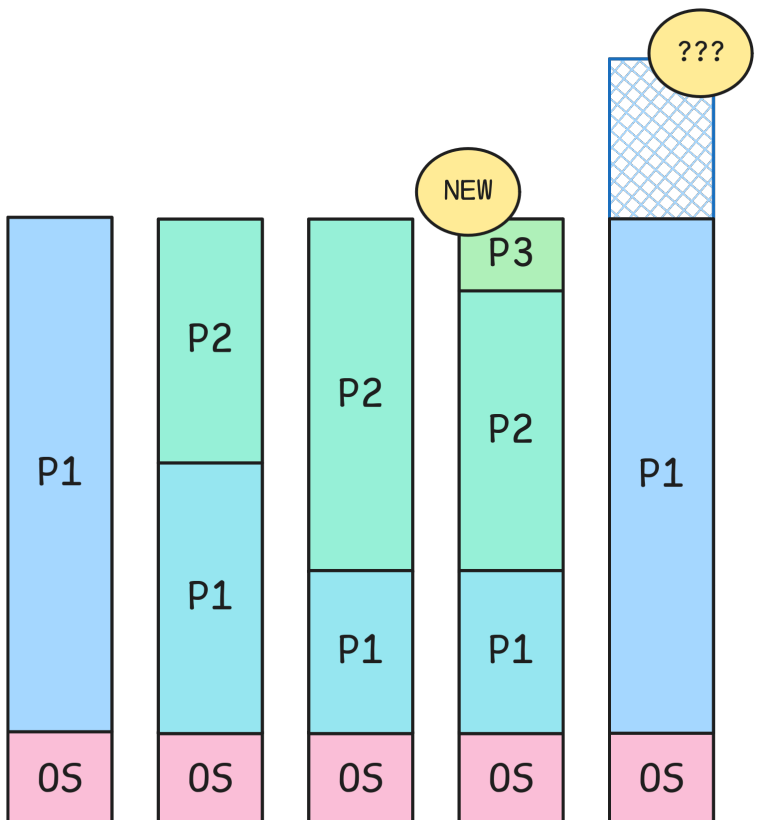
- Wie viel Platz bekommt die **Kernel-Seite**, wie viel die **User-Seite**?
- Was, wenn **mehrere User-Programme** gleichzeitig laufen sollen?
- Was, wenn die User-Programme **unterschiedlich viel Speicher** nutzen?
- Was, wenn sich die **Anzahl der User-Programme** dynamisch verändert?
-



Einen Schritt zurück

Struktur des Hauptspeichers

- Wie viel Platz bekommt die **Kernel-Seite**, wie viel die **User-Seite**?
- Was, wenn **mehrere User-Programme** gleichzeitig laufen sollen?
- Was, wenn die User-Programme **unterschiedlich viel Speicher** nutzen?
- Was, wenn sich die **Anzahl der User-Programme** dynamisch verändert?
- Was, wenn die User-Programme **mehr Speicher** benötigen, als wir zur Verfügung haben?



Virtueller Speicher

GRUNDIDEE

Jedem Prozess wird vorgegaukelt, dass ihm ein **eigener, zusammenhängender, isolierter Bereich** im Hauptspeicher zur Verfügung steht.

- virtueller Speicher hat virtuelle Adressen
- außerdem: Aufteilung des Speichers in gleich große Seiten (en. *Paging*)
- Pages können sogar auf die Festplatte ausgelagert werden

Elegante Lösung?

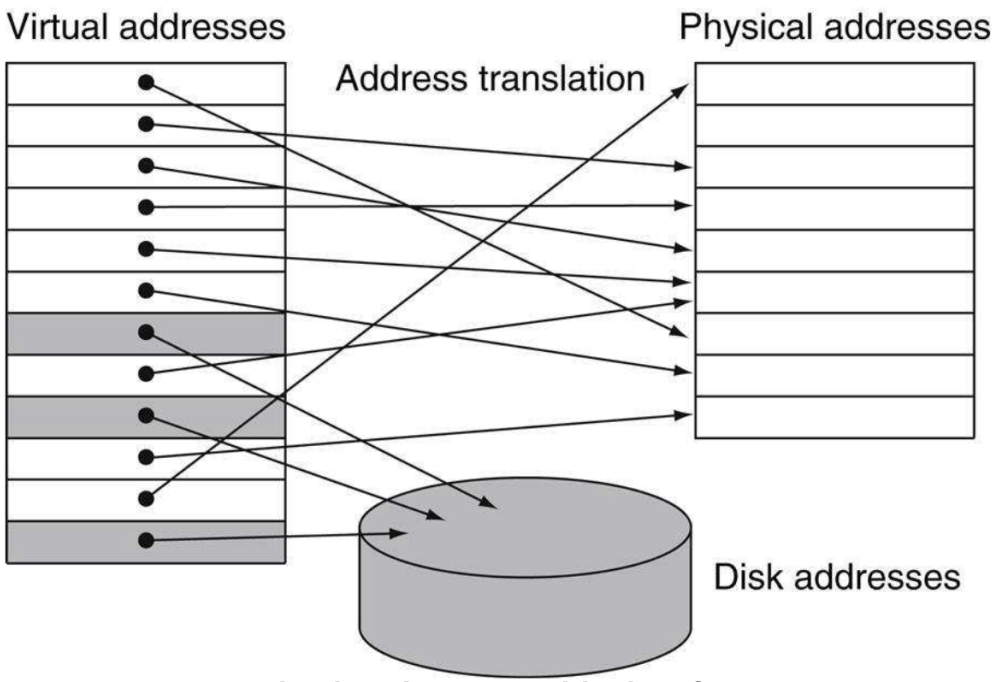


Abbildung 1: Adresszuordnung

Elegante Lösung?

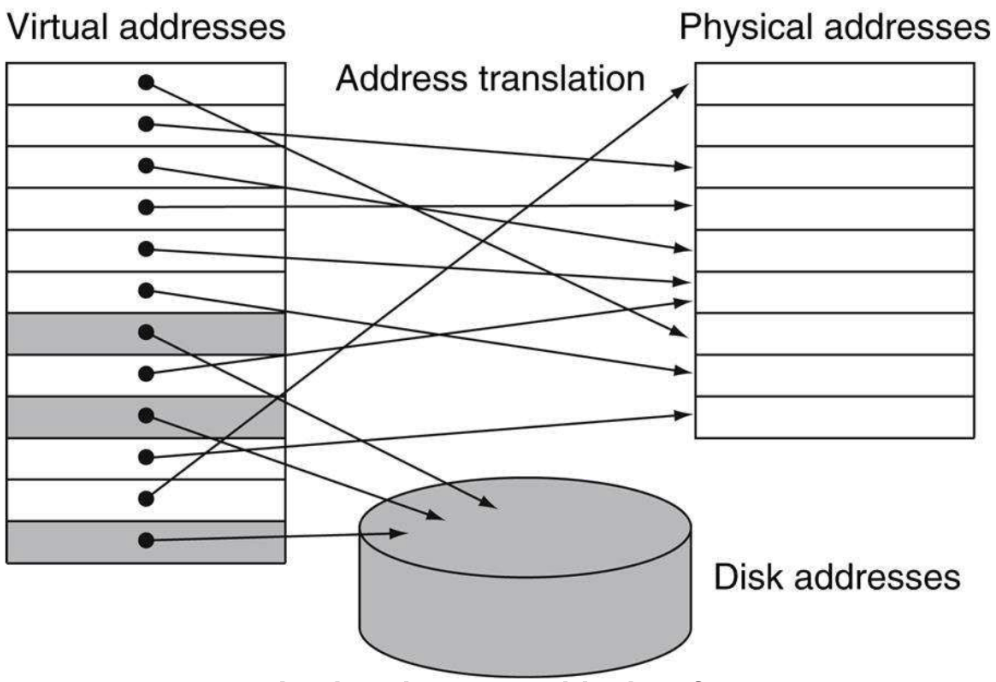


Abbildung 3: Adresszuordnung

© D. A. Patterson und J. L. Hennessy [1], Fig. 5.25

Abbildung 4: Ansichtssache

Seitenfehler

en. *Page Fault*

Es existiert keine Abbildung auf eine physische Adresse.

- Seite im Hintergrundspeicher (ausgelagert mittels Swapping)
- keine Abbildung für virtuelle Adresse vorhanden (Adressraumverletzung)
- fehlende Zugriffsrechte (z. B. Schreibzugriff auf *read-only*-Seite)
- ...

Paging

Seitentabelle

en. *Page Table*

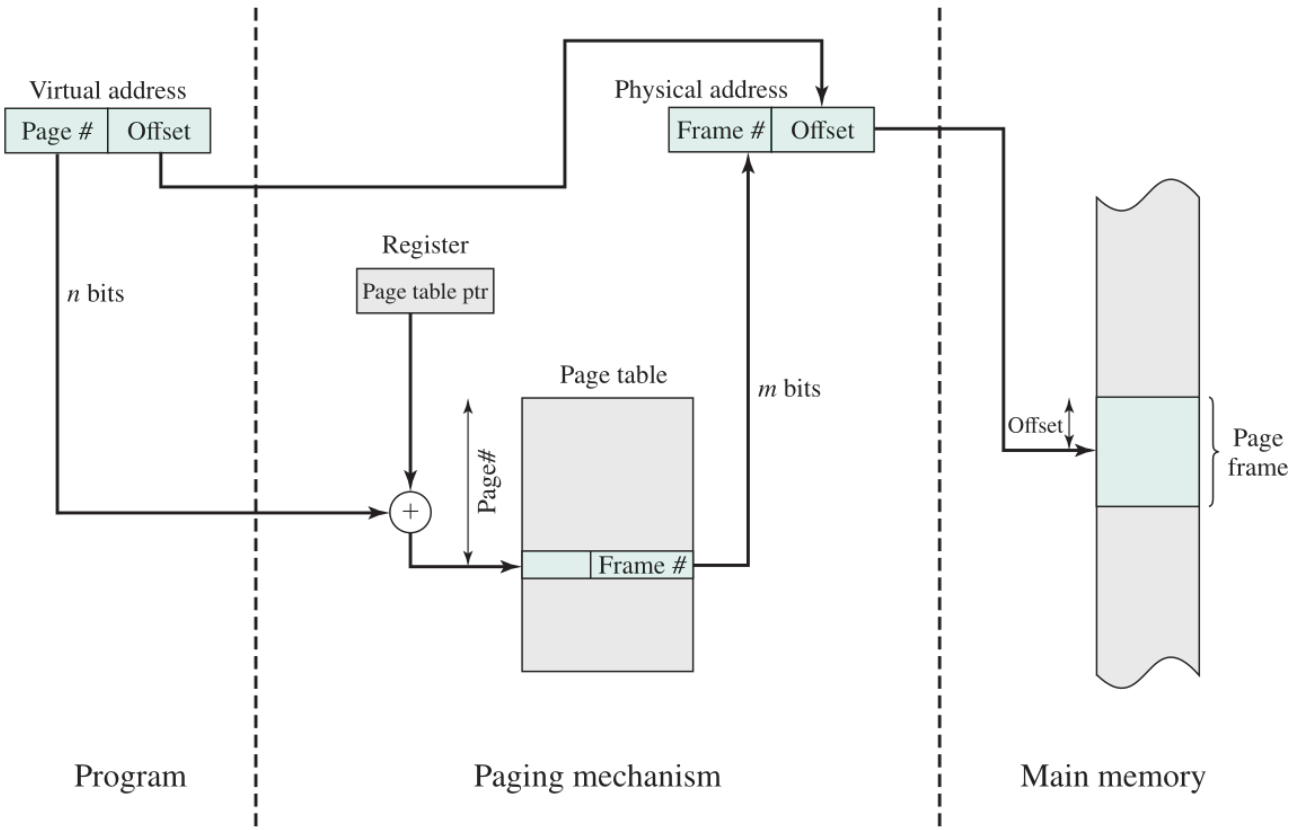


Abbildung 5: Adressübersetzung mittels Seitentabelle

Adressübersetzung

Einstufige Seitentabellen

AUFGABE 1

Ein System mit seitenbasiertem virtuellem Speicher besitzt die folgende Seitentabelle. Die Länge einer Adresse beträgt 16 Bit. Die 12 niederwertigsten Bit einer Adresse sind der Offset der Seite. Damit ergibt sich eine Seitengröße von 4.096 Byte. Es kommt einstufiges Paging zum Einsatz.

Übersetzen Sie die virtuellen Adressen `0x6AB1` und `0xF1B7` in ihre physikalischen Adressen. Geben Sie außerdem an, welche Zugriffsrechte für die Seiten bestehen.

Mehrstufige Seitentabellen

Zuordnung der Adressräume

Probleme linearer Seitentabellen:

- Adressräume sind dünn besetzt
- Verwaltung einer linearen Tabelle ist aufwendig

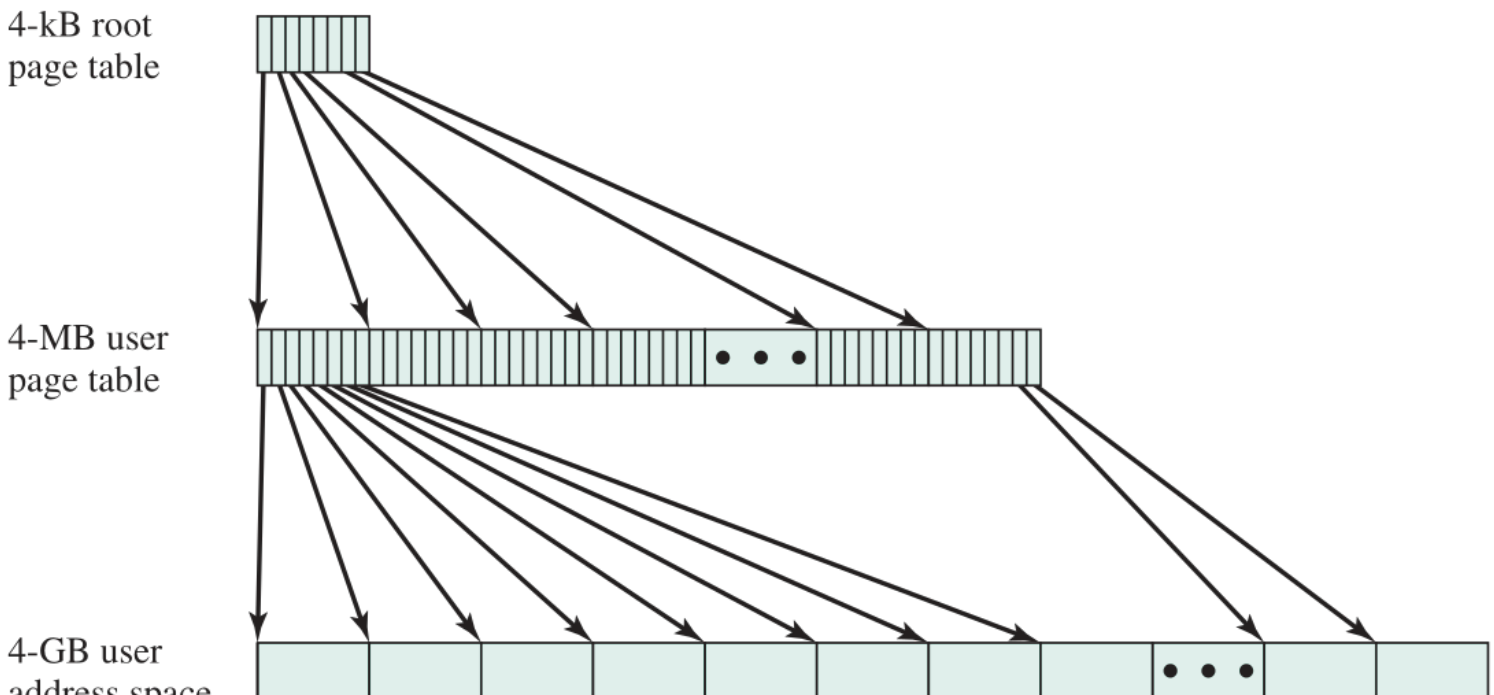


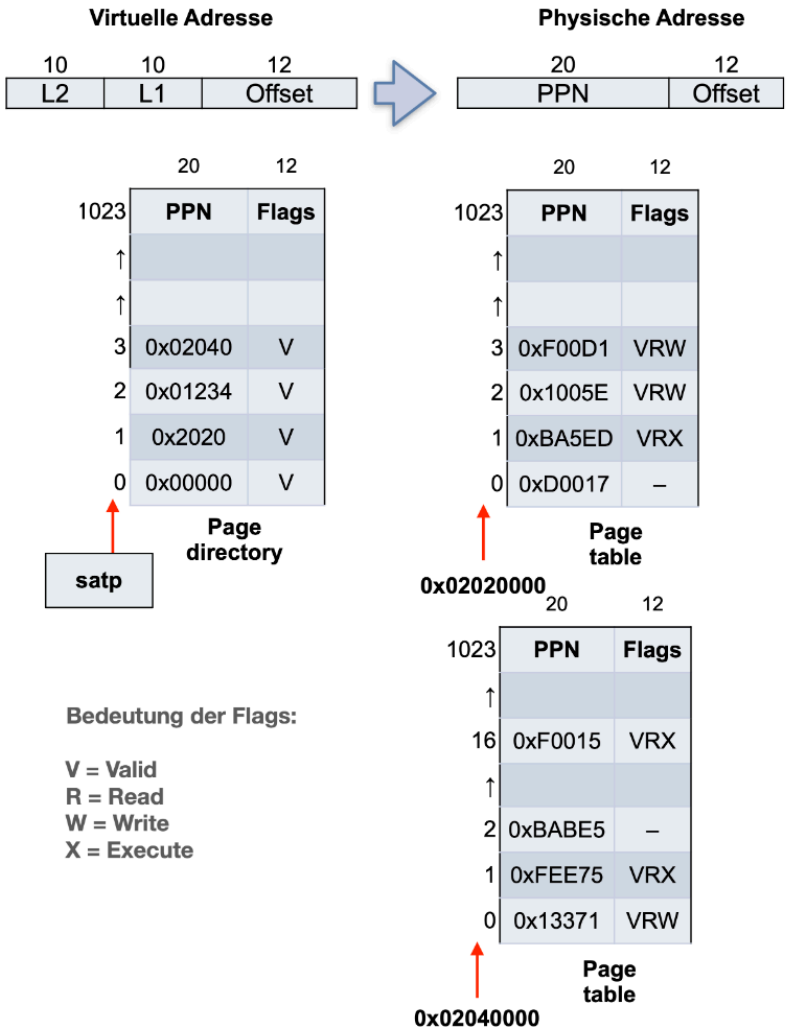
Abbildung 6: Mehrstufige Seitentabelle

Adressübersetzung

Mehrstufige Seitentabellen

AUFGABE 2

Sie untersuchen den Speicher eines 32-Bit RISC-V-Prozessors und gelangen zu dem nebenstehenden Diagramm. Bestimmen Sie für die angegebene zweistufige Seitentabelle die physische Adressen zur virtuellen Adresse `0x00C02CCC`.



Mehrstufige Seitentabelle

Adressübersetzung

Probleme mehrstufiger Seitentabellen:

- Seitentabellen mit zwei bis fünf Stufen im Einsatz
- Durchlaufen einer mehrstufigen Tabelle aufwändig

Ansatz: *Translation-Lookaside-Buffer* als Cache

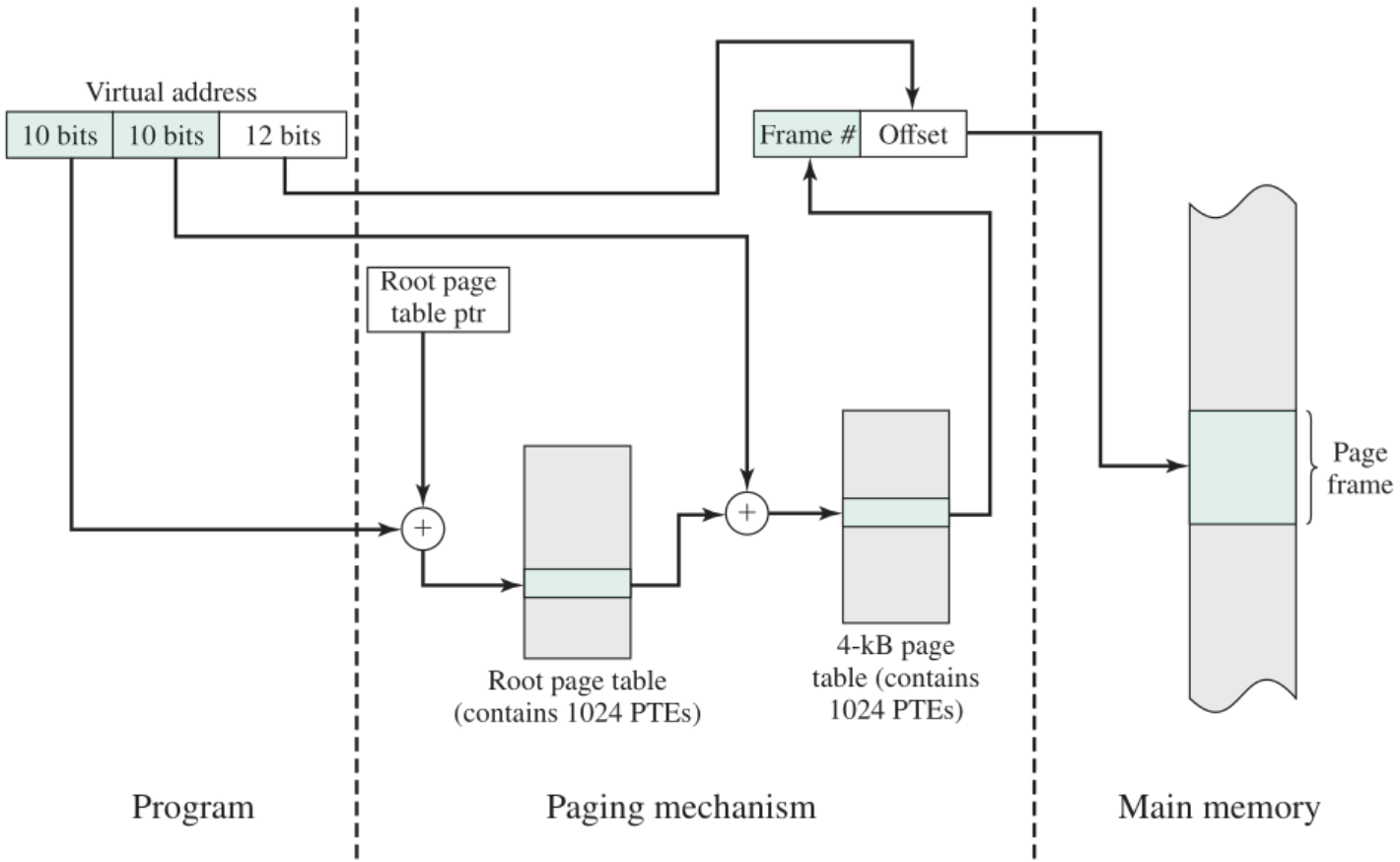


Abbildung 7: Adressübersetzung mittels Seitentabelle

Ausblick

Das war's noch nicht ganz …

- Unterschied „*TLB Miss*“ und „*Page Table Miss*“ bzw. „*Page Fault*“
- Was ist, falls Eintrag in TLB, aber nicht in Seitentabelle?
- verzögerte Allokation bei Speicheranforderung durch Anwendung
- mehrstufiger Hintergrundspeicher, z. B. `zswap`
- *Demand Zeroing*, z. B. zur Optimierung von `calloc()`
- *Copy-on-Write*, z. B. bei `fork()`
- *Memory-Mapped File*

Ersetzungsstrategien

Ersetzungsstrategien

Das sprengt den Rahmen

Algorithm 1: Vereinfachter Seitenzugriff

```
1: function AccessPage(pageFrames, page, pointer)
2:     if not IsPageInRAM(pageFrames, page) then
3:         LoadPageIntoRAM(pageFrames, page, pointer)
4:     end
5: end
```

AUSGANGSFRAGE

Welche Seite muss Platz machen, wenn wir keine freien Seitenrahmen mehr haben?

Für eine Liste von Seitenrahmen `pageFrames`, eine zuzugreifende Seite `page` und den Zeiger `pointer`, der den nächsten zu betrachtenden Seitenrahmen markiert.

Second Chance Algorithm

Algorithm 2: Seite in den Arbeitsspeicher laden nach Second Chance Algorithm

```
1:  function LoadPageIntoRAM(pageFrames, page, pointer)
2:      foundPlaceToInsert ← false
3:      while not foundPlaceToInsert do
4:          currentPageFrame ← pageFrames[pointer]
5:          if currentPageFrame.hasSecondChance then          ▷ zweite Chance gewähren
6:              currentPageFrame.hasSecondChance ← false
7:          else                                              ▷ zweite Chance verbraucht
8:              currentPageFrame.insert(page)
9:              currentPageFrame.hasSecondChance ← true
10:             foundPlaceToInsert ← true
11:         end
12:         pointer.move()
13:     end
14: end
```

Second Chance Algorithm

Algorithm 3: Seite im Arbeitsspeicher finden nach Second Chance Algorithm

```
1: function IsPageInRAM(pageFrames, page)
2:     for pageFrame in pageFrames do
3:         if pageFrame.contains(page) then
4:             pageFrame.hasSecondChance ← true          ▷ zweite Chance wiederherstellen
5:             return true
6:         end
7:     end
8:     return false
9: end
```

Fragen?

Literatur

[1] D. A. Patterson und J. L. Hennessy, *Computer Organization and Design*. Morgan Kaufmann, 2018.

[2] W. Stallings, *Operating Systems*, 9. Aufl. Pearson Education Limited, 2017.

Lizenzhinweise

Kursmaterialien · **Quelltext**

Diese Präsentation wurde zuletzt am 24.06.2026 bearbeitet. Sie basiert auf den Folien der zugehörigen Vorlesung von Prof. Dr. Michael Engel. Sofern nicht anders angegeben, sind die Inhalte unter der **Lizenz CC BY-SA 4.0** verfügbar. Das Universitätslogo sowie die Schriftart UB Scala Sans sind Eigentum der Otto-Friedrich-Universität Bamberg.

Folgende **Open-Source-Komponenten** kommen in der Präsentation zum Einsatz: **circuiteria** (Apache 2.0), **finite** (MIT), **fontawesome** (SIL), **pgf-tikz** (GPL 2.0), **typst** (Apache 2.0), **typst-ccicons** (MIT), **typst-polylux** (MIT).