# Betriebssysteme 1

**Foliensatz H, Kap. 8**
Speicherverwaltung

Prof. Dr. Hans-Georg Eßer

Sommersemester 2022

v3.1 – 15.06.2022





## *Classics*: Overlay-Programmierung

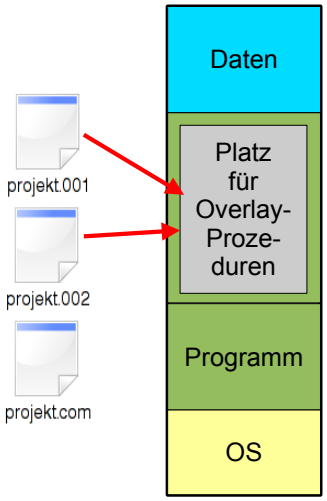
Turbo Pascal, um 1985-90:

```
program grossesprojekt;

overlay procedure kundendaten;
...

overlay procedure lagerbestand;
...

(* Hauptprogramm *)
begin
  while input <> "exit" do begin
    case input of
      "kunden": kundendaten;
      "lager":  lagerbestand;
    end;
  end;
end.
```






## Buddy-System (1)

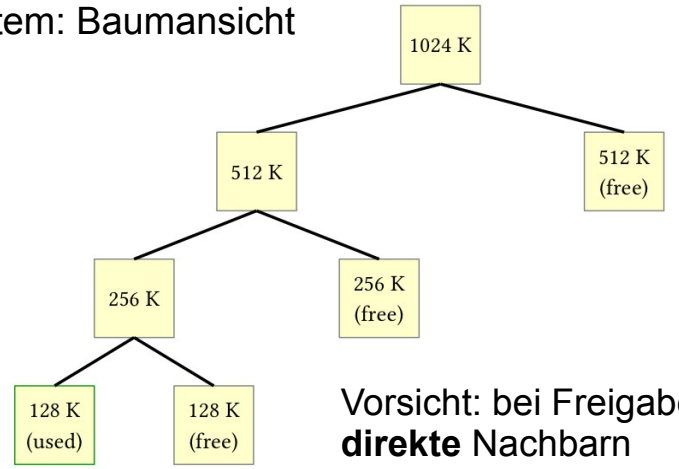
- Buddy-System (dynamische Zuteilung)
  - Speichergröße ist $2^n$ (für ein $n$)
  - Bei Anforderung schrittweise freien Speicherbereich halbieren, bis gerade noch passender Bereich verfügbar ist
  - Bei Rückgabe von Speicher diesen ggf. mit freiem Nachbarn verschmelzen (→ Rekursion?)
  - Beispiel: 1 MByte, alles frei, Anforderung 90 KByte

<table>
<tr><td colspan="4">1024 KB</td></tr>
<tr><td colspan="3">512 KB</td><td>512 KB</td></tr>
<tr><td colspan="2">256 KB</td><td>256 KB</td><td>512 KB</td></tr>
<tr><td>128 KB</td><td>128 KB</td><td>256 KB</td><td>512 KB</td></tr>
</table>



## Buddy-System (2)

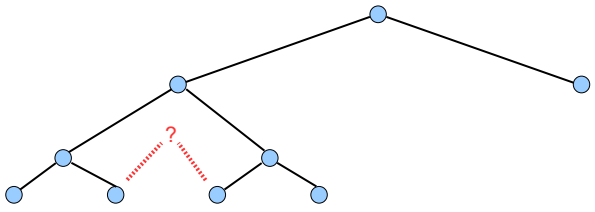
- Buddy-System: Baumansicht




Vorsicht: bei Freigabe nur **direkte** Nachbarn verschmelzen!

## Buddy-System (3)

- Buddy-System: unmögliche Verschmelzung



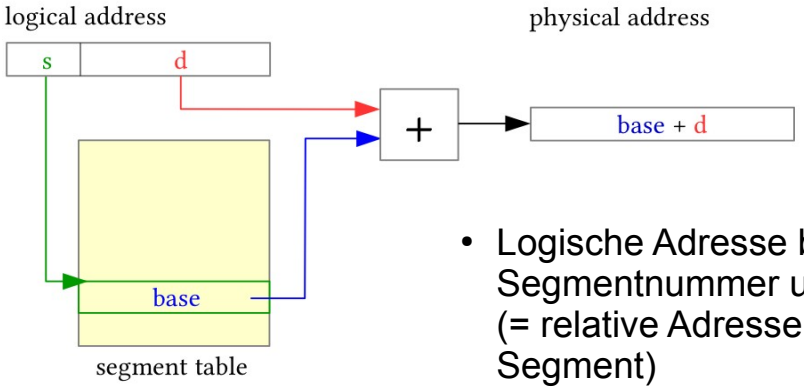

| | | | | |
|---|---|---|---|---|
| 128 KB | 128 KB | 128 KB | 128 KB | 512 KB |
| 128 KB | 256 KB | | 128 KB | 512 KB |

!



## Segmentierung (1)

- Über eine Segment-Tabelle wird Speicher in Segmente (zusammenhängende Bereiche) aufgeteilt

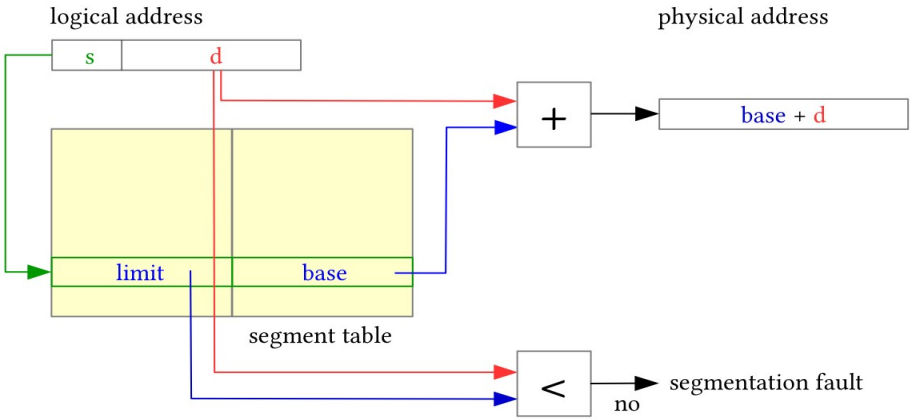



- Logische Adresse besteht aus Segmentnummer und Offset (= relative Adresse innerhalb Segment)



## Segmentierung (2)

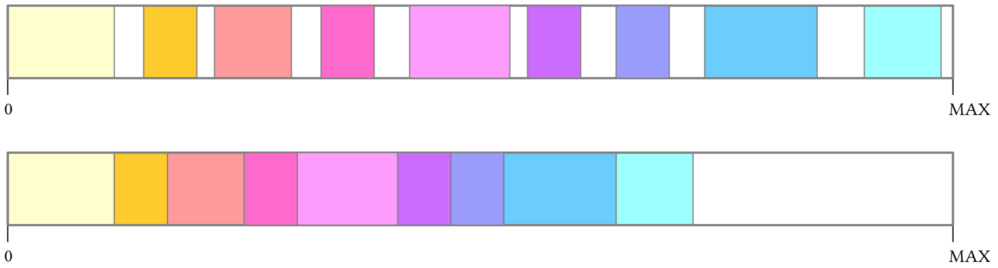
- Angabe einer Segmentgröße → Prüfung bei Zugriff






## Segmentierung (3)

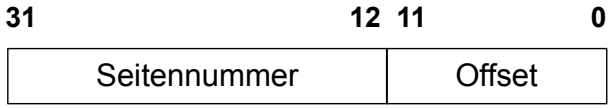
- Problem bei dynamischem Erzeugen und Löschen von Segmenten: Lücken
- Regelmäßig aufräumen („Kompaktierung“)



## Adressübersetzung beim Paging (1)

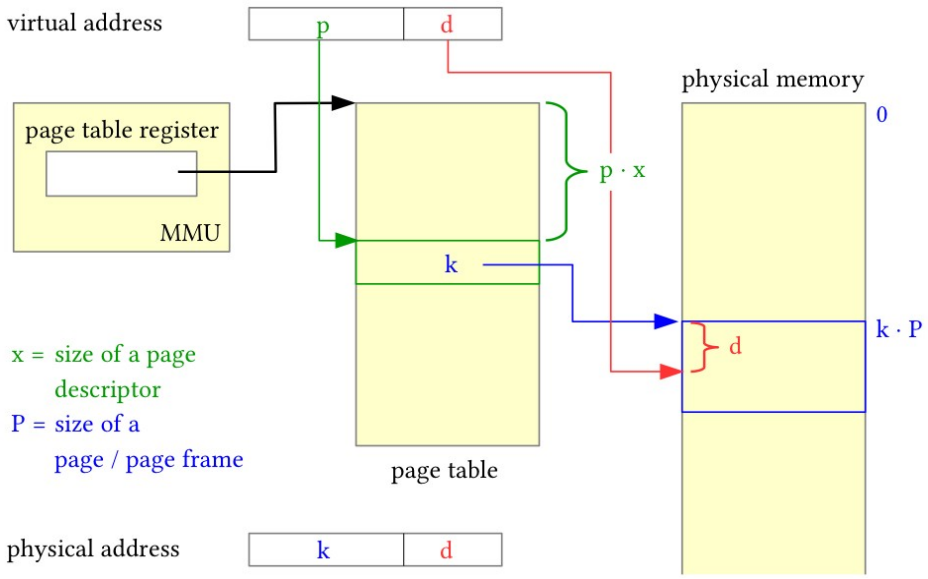
- Die Programmadresse wird in zwei Teile aufgeteilt:
  - eine Seitennummer
  - eine relative Adresse (offset) in der Seite

Beispiel: 32-bit-Adresse bei einer Seitengröße von 4096 (=$2^{12}$) Byte:





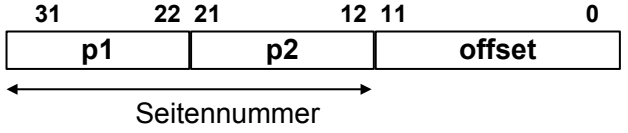

## Adressübersetzung beim Paging (2)






## Mehrstufiges Paging (1)

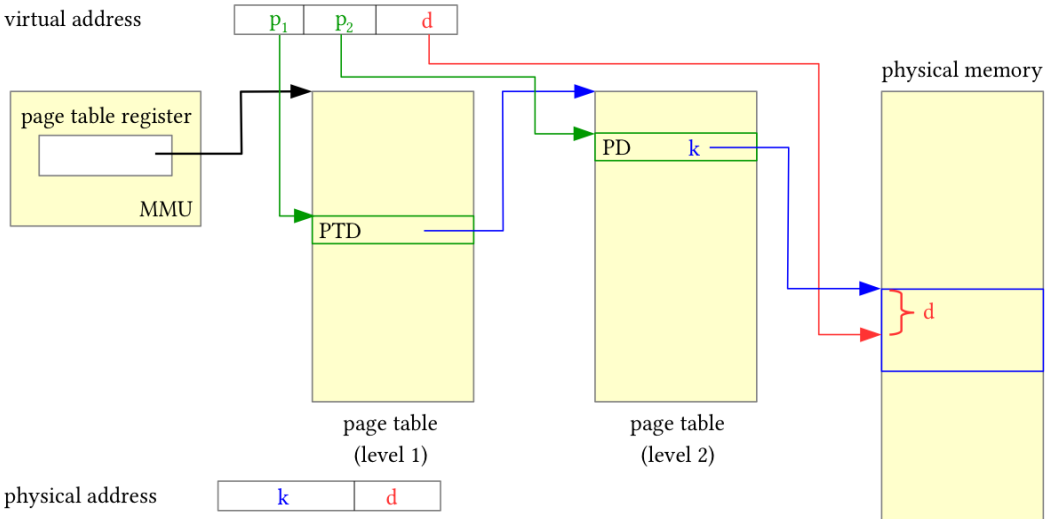
- Mehrstufiges Paging:
  - Seitennummer noch einmal unterteilen, z. B.:



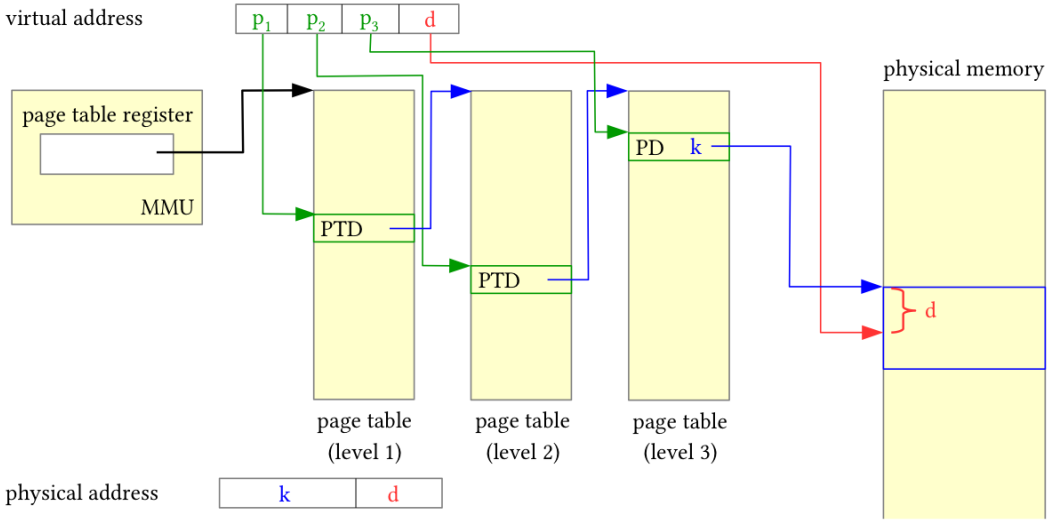

  - $p_1$: Index in **äußere Seitentabelle**, deren Einträge jeweils auf eine **innere Seitentabelle** zeigen
  - $p_2$: Index in eine der inneren Seitentabellen, deren Einträge auf Seitenrahmen im Speicher zeigen
  - Die inneren Seitentabellen müssen nicht alle speicherresident sein



## Mehrstufiges Paging (2)



## Mehrstufiges Paging (3)






## Mehrstufiges Paging (4)

- Größe der Seitentabellen:

  Beispiel:

| $p_1$ | $p_2$ | offset |
|---|---|---|
| 10 | 10 | 12 |

  - Die äußere Seitentabelle hat 1024 Einträge, die auf (potentiell) 1024 innere Seitentabellen zeigen, die wiederum je 1024 Einträge enthalten.
  - Bei einer Länge von 4 Byte pro Seitentabelleneintrag ist also jede Seitentabelle genau eine 4-KByte-Seite groß.
  - Es werden nur so viele innere Seitentabellen verwendet, wie nötig.



## Paging: Beispiel (1)

Paging mit folgenden Parametern:
- 32-Bit-Adressbus
- 32 KB Seitengröße
- 64 MB RAM
- 1-stufiges Paging

Zu berechnen:
a) maximale Anzahl der adressierbaren virtuellen Seiten
b) Größe der erforderlichen Seitentabelle (in KB)

a) 32 KB (Seitengröße) = $2^5 \times 2^{10}$ Byte = $2^{15}$ Byte
d.h.: Offset ist 15 Bit lang

| 17 Bit | 15 Bit |
|---|---|
| Seitennummer | Offset |

Also gibt es $2^{17}$ virtuelle Seiten

b) Zur Seitentabelle:
In 64 MB RAM passen 64 M / 32 K = 2 K = 2048 ($2^{11}$) Seitenrahmen
Ein Eintrag in der Seitentabelle benötigt darum 11 Bit, in der Praxis 2 Byte.

→ Platzbedarf:
#(virt. Seiten) x Größe(Eintrag)
= $2^{17}$ x 2 Byte = $2^{18}$ Byte = 256 KB



## Paging: Beispiel (2)

Paging mit folgenden Parametern:
- 32-Bit-Adressbus
- 16 KB Seitengröße
- 2 GB RAM
- 3-stufiges Paging

Zu berechnen:
a) # adressierbare virtuelle Seiten
b) Größe der Seitentabelle(n)
c) Anzahl der Tabellen

a) 16 KB (Seitengröße) = $2^4 \times 2^{10}$ Byte = $2^{14}$ Byte,
d.h.: Offset ist 14 Bit lang

| 6 Bit | 6 Bit | 6 Bit | 14 Bit |
|---|---|---|---|
| Seitennummer | | | Offset |

Also gibt es $2^{18}$ virtuelle Seiten

b) Zu den Seitentabellen:
In 2 GB RAM passen 2 G / 16 K = 128 K = $2^{17}$ Seitenrahmen
Ein Eintrag in der Seitentabelle benötigt darum 17 Bit, in der Praxis 4 Byte.

→ Platzbedarf **einer** Tabelle:
#(Einträge) x Größe(Eintrag)
= $2^6$ x 4 Byte = $2^8$ Byte = 256 Byte

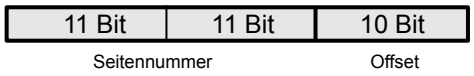
Es gibt 1 äußere, $2^6$ mittlere und $2^{12}$ innere Seitentabellen

## Paging: Beispiel (3) – **SKA 37**

Paging mit folgenden Parametern:
- 32-Bit-Adressbus
- 1 KB Seitengröße
- Eintrag in Seitentabelle: 4 Byte
- 2-stufiges Paging

Zu berechnen:
a) # adressierbare virtuelle Seiten
b) Größe der Seitentabelle(n)
c) Anzahl der Tabellen

a) 1 KB (Seitengröße) = $2^{10}$ Byte,
d.h.: Offset ist 10 Bit lang




Also gibt es $2^{22}$ virtuelle Seiten

b) Zu den Seitentabellen:

Platzbedarf **einer** Tabelle:
#(Einträge) x Größe(Eintrag)
= $2^{11}$ x 4 Byte = $2^{13}$ Byte = 8 KByte

Es gibt
- 1 äußere und
- $2^{11}$ innere Seitentabellen



## **SKA 38** – Wahr oder falsch? (1/2)

a) Beim Paging werden Seitenrahmen (*frames*) auf Seiten (*pages*) abgebildet.

b) Paging mit mehrstufigen Seitentabellen reduziert den Speicherverbrauch (im Vergleich zu einstufigem Paging).

c) Paging lagert bei Speicherknappheit den Speicher eines oder mehrerer Prozesse vollständig auf Festplatte aus. Dieses Aus- und Wiedereinlagern wird auch *Swapping* genannt.

d) Paging gehört zu den Verfahren der *zusammenhängenden Speicherverwaltung*.

e) Die feste Partitionierung verursacht hohe *externe Fragmentierung*.

f) Beim Paging kann ein Prozess auch mehr virtuellen Speicher nutzen als der Rechner an physischem RAM enthält.



## **SKA 38** – Wahr oder falsch? (2/2)

g) Wenn ein Prozess einen *Page Fault* verursacht, weil er auf eine Adresse zugreift, deren Seite ausgelagert ist, wird der Prozess blockiert – es entsteht eine vergleichbare Wartezeit (I/O) wie beim Lesen eines Datenblocks aus einer Datei.

h) ~~Aufgabe zu malloc / realloc (nicht behandelt)~~

i) Segmentierung gehört zu den Verfahren der *zusammenhängenden Speicherverwaltung*.

j) Bei Segmentierung auf einem 32-Bit-Intel-Prozessor mit maximaler Speicherausrüstung (4 GByte) kann ein 64 MByte großes Segment gebildet werden, das *gleichzeitig* aus den untersten 32 MByte RAM (0x0000 0000 – 0x01FF FFFF) und den obersten 32 MByte RAM (0xFE00 0000 – 0xFFFF FFFF) besteht.