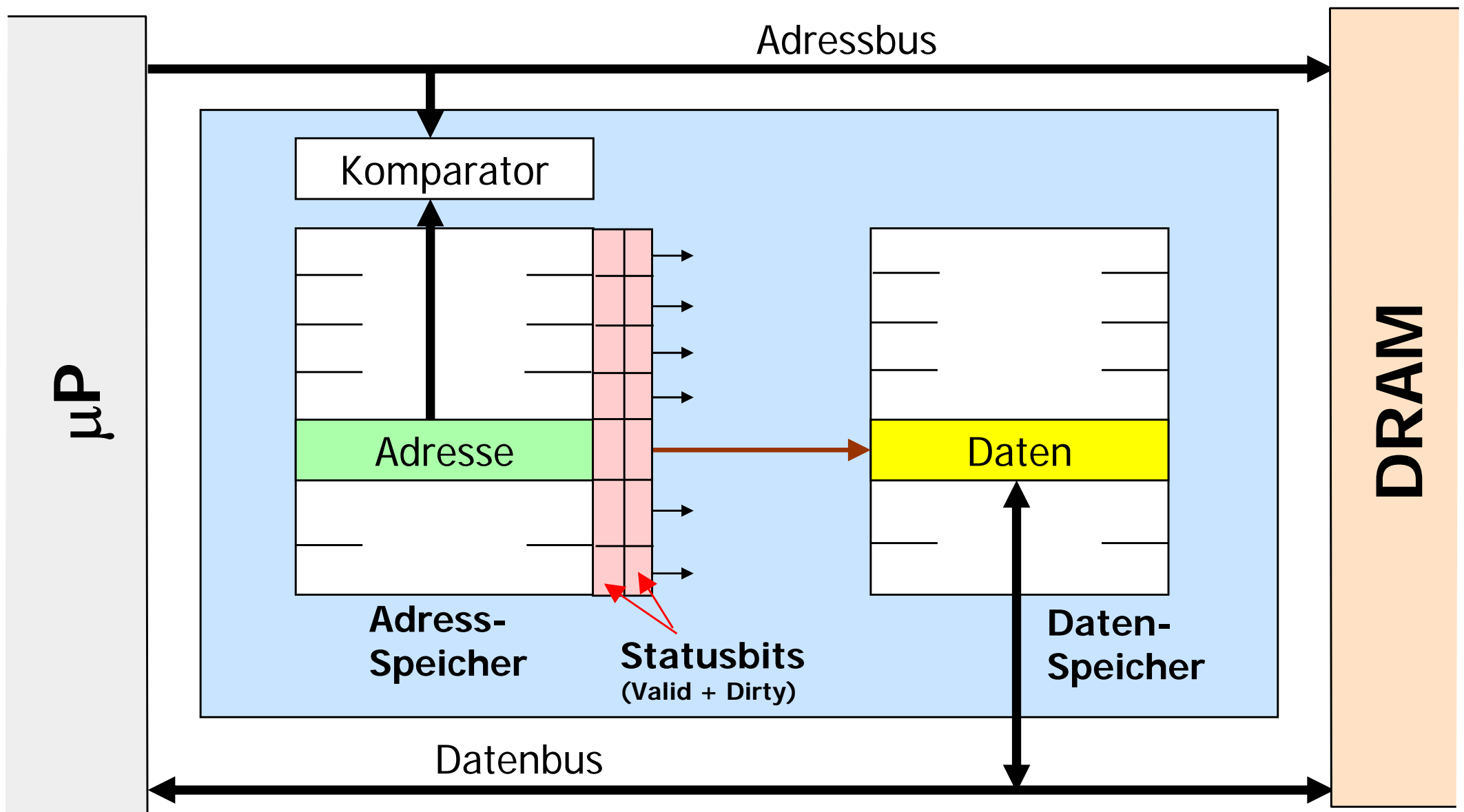

7. Übung

- ☐ **Cache-Speicher**
- ☐ **Virtuelle Speicherverwaltung**

Aufbau eines Cache-Speichers



Fehlzugriffe (Misses)

Klassifikation von Misses:

- ❑ **Compulsory-Misses:** Beim ersten Zugriff auf einen Block muss der Block in den Cache geladen werden.
(*cold start misses* oder *first reference misses*)
(Tritt auch bei einem unendlich großem Cache)
- ❑ **Capacity-Misses:** Wenn der Cache bei der Ausführung eines Programms nicht alle benötigten Blöcke aufnehmen kann.
- ❑ **Conflict:** Durch die Block-Zuweisungsstrategie (set associative oder direct mapped), auch *collision misses* oder *interference misses* genannt
- ❑ **Coherence–Misses:** durch Cache-Kohärenz.

Aufgabe 4

Cache Hit/Miss-Rate

Programmschleife zur Multiplikation von 512 16-Bit-Zahlen

```
for (mul=1, j=0; j<512; j++) mul *=g[j];
```

Vollassoziativer Daten-Cache (am Anfang leer) mit 64 Bytes pro Cache-Zeile.

→ 32 Zahlen pro Cache-Zeile

Lösung 4

	Miss	Hit
mul = 1	1 1	
j = 0	1 1	
loop: read j		1 1 512
if (j >= 512) exit		
else		
read g[j]	1 16	1 496
read mul		1 1 512
compute mul *g[j]		
write mul		1 1 512
read j		1 1 512
compute j+1		
write j		1 1 512
jump to loop		
1. Durchlauf	18	3056
2. Durchlauf	0,06 %	99,994 %

Aufgabe 5

Bei einem Cache-Speicher mit einer Speicherkapazität von **128 Kbyte** ist die Hauptspeicheradresse in ein **16 Bit** Tag-Feld, ein **12 Bit** Index-Feld und ein **4 Bit** Byte-Offset unterteilt.

1. Bestimmen Sie die Blockgröße in Bytes.
2. Wieviele Einträge besitzt der Cache-Speicher?
3. Wie ist der Cache-Speicher organisiert?

Aufgabe 5.1

Cache-Kapazität = **128 Kbyte**

16 Bit Tag-Feld, **12 Bit** Index-Feld, **4 Bit** Byte-Offset

Blockgröße: $2^4 = 16$ Bytes

Aufgabe 5.2

Cache-Kapazität = **128 Kbyte**

16 Bit Tag-Feld, **12 Bit** Index-Feld, **4 Bit** Byte-Offset

Anzahl der Einträge im Cache:

$$\frac{128 \text{ KByte}}{16}$$

Aufgabe 5.3

Cache-Kapazität = **128 Kbyte**

16 Bit Tag-Feld, **12 Bit** Index-Feld, **4 Bit** Byte-Offset

Organisation:

Mit 12 Bit Index-Feld kann man
 $2^{12} = 4K$ Sätze/zeilen selektieren.

Cache hat 8K Einträge

$\downarrow \frac{8K}{4K} = 2 \quad \downarrow$ 2-way-set-asso.

Aufgabe 6

Es soll ein 3-fach-assoziativer (*3-way set associative cache*) Cache-Speicher mit 128 Sätzen und einer Blockgröße von 8 Byte realisiert werden.

Nehmen Sie an, dass die Hauptspeicheradresse 32 Bit breit ist. Zur Verwaltung eines Cacheblocks wird zwei Statusbits (Valid-Bit: V und Dirty Bit: D) verwendet.

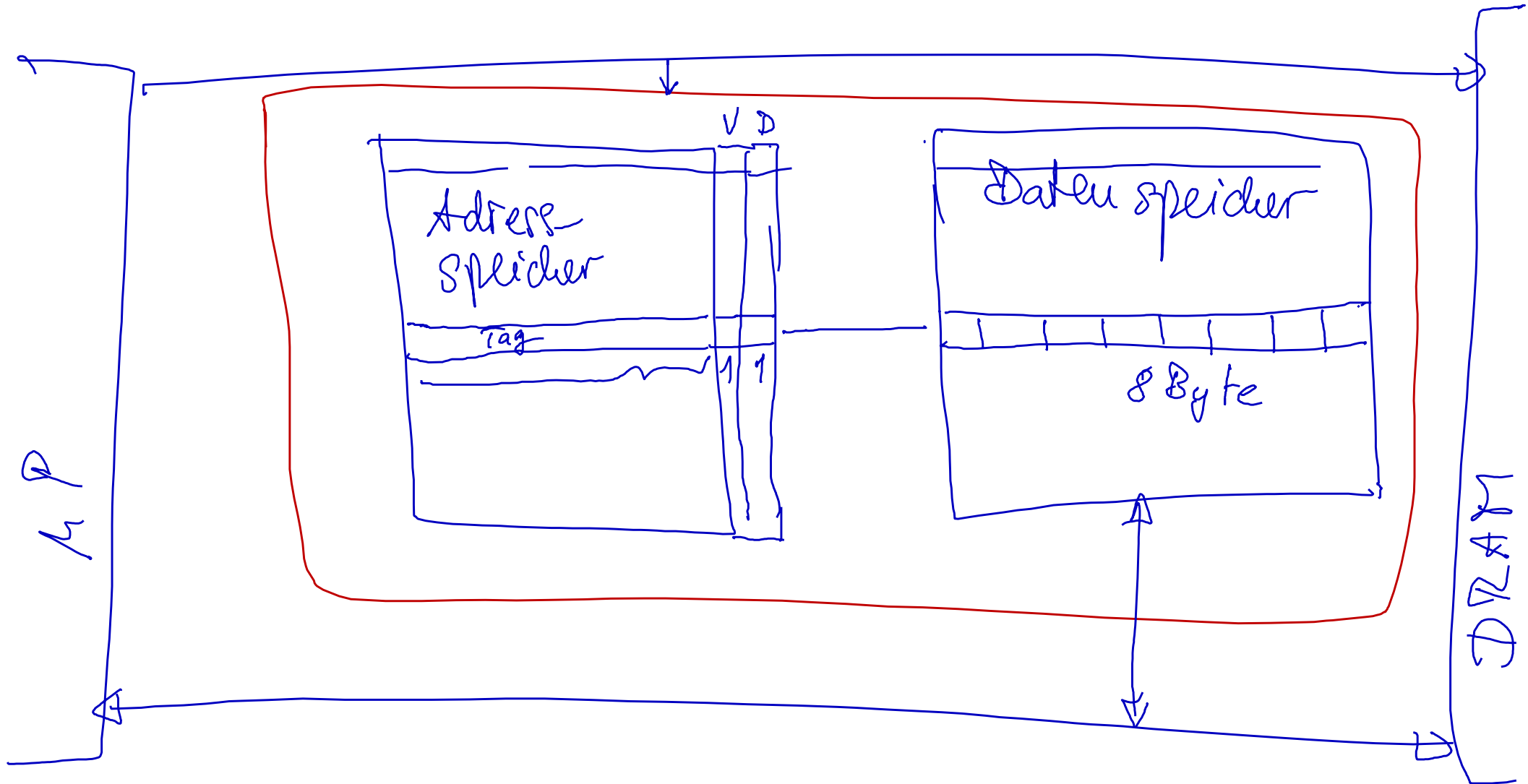
Bestimmen Sie den insgesamt erforderlichen Speicherbedarf zur Realisierung dieses Cache-Speichers.

128 Sätze

3-fach-asso.

8 Byte

128.3



Lösung 6

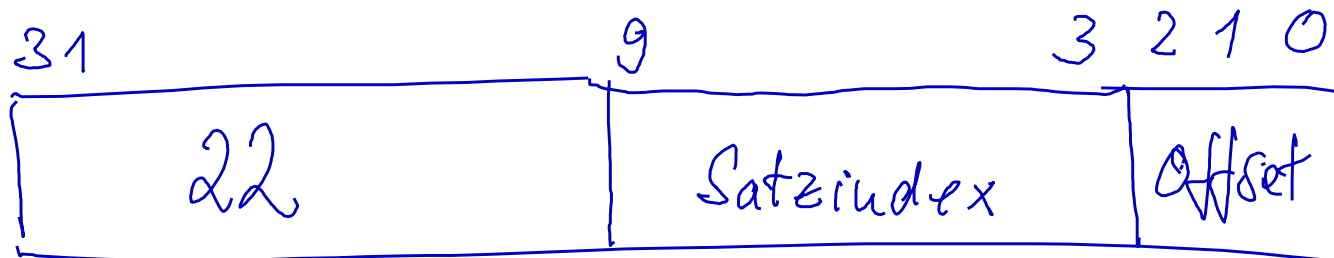
3-fach-assoziativer (*3-way set associative cache*)

Cache-Speicher mit **128 Sätzen** und einer Blockgröße von **8 Byte**

Speicherbedarf für eine Zeile:

Tag-Länge + Anzahl der Statusbits + Blockgröße

22 + 2 Bit + 8 Byte



$$\text{Tag} = 32 - 10$$

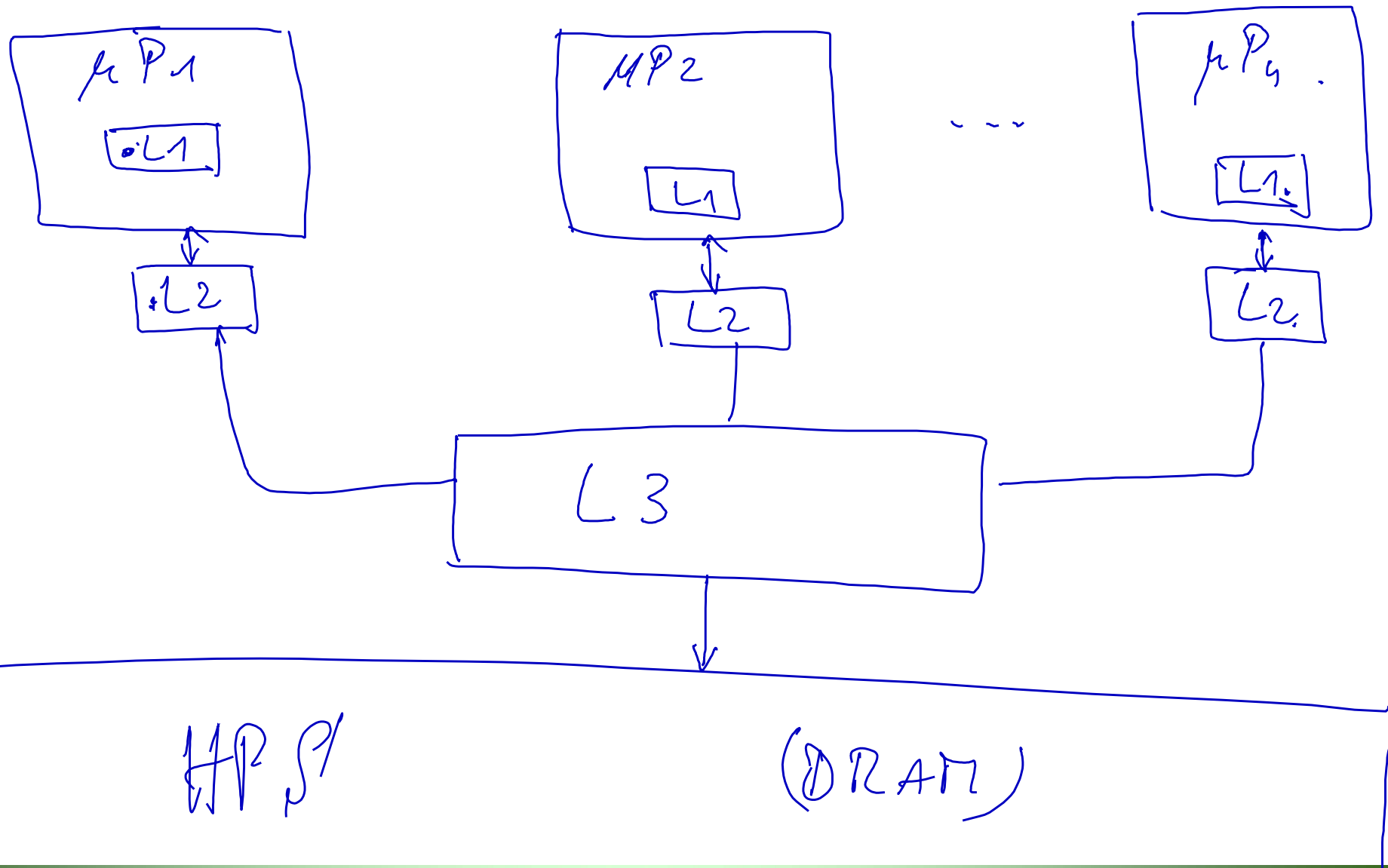
Speicherbedarf für den gesamten Cache

$$\underbrace{(22 \text{ Bit} + 2 \text{ Bit} + 8 \text{ Byte})}_{\text{pro Zeile}} \cdot 128 \cdot 3$$

$$= \underline{\underline{11 \cdot 128 \cdot 3 \text{ Byte}}}$$

Cache-Kohärenzproblem (*Cache Coherency Problem*)

- **Cache-Kohärenzproblem** (*Cache Coherency Problem*): Gültigkeitsproblem, das beim Zugriff mehrerer Verarbeitungselemente auf Speicherworte des Hauptspeichers entsteht.
- **Kohärenz** bedeutet das korrekte Voranschreiten des Systemzustands durch ein abgestimmtes Zusammenwirken der Einzelzustände.
- Im Zusammenhang mit dem Cache muss das System dafür sorgen, dass immer die aktuellen Daten und nicht die veralteten Daten gelesen werden.
- Ein System ist **konsistent**, wenn alle Kopien eines Datums im Hauptspeicher und den verschiedenen Cachespeichern identisch sind. Dadurch ist auch die *Kohärenz sichergestellt*, jedoch entsteht ein hoher Aufwand.



Bus-Schnüffeln (*Bus-Snooping*)

- In Mehrprozessorsystemen, bei denen mehrere Prozessoren mit lokalen Cachespeichern an einen gemeinsamen Bus/Hauptspeicher angeschlossen sind, verwendet man das sogenannte **Bus-Schnüffeln**.
- Die **Schnüffel-Logik** jedes Prozessors hört am Bus die Adressen mit, die die anderen Prozessoren auf den Bus legen. Die Adressen auf dem Bus werden mit den Adressen, der im Cache gespeicherten Daten, verglichen.
- Bei Adressübereinstimmung am Bus geschieht folgendes:

Bus-Schnüffeln (*Bus-Snooping*)

- ❑ Wenn ein **Schreibzugriff** auf dieselbe Adresse vorliegt, dann wird der im Cache gespeicherte Cacheblock für „ungültig“ erklärt (**Write-Invalidate-Verfahren**), oder mit aktualisiert (**Write-Update-Verfahren**).
- ❑ Wenn ein **Lesezugriff** auf dieselbe Adresse mit einer modifizierten Datenkopie im Cachespeicher festgestellt wird, dann legt der Cache-Controller ein Snoop Status Signal (SSTAT) auf den Bus.
 - Der Prozessor, der die Adresse auf den Bus gelegt hat, unterbricht seine Bustransaktion.
 - Der „schnüffelnde“ Cache-Controller übernimmt den Bus und schreibt den betreffenden Cacheblock in den Hauptspeicher.
 - Dann wird die ursprüngliche Bustransaktion erneut durchgeführt.
- ❑ Siehe MESI-Protokoll (bei Multiprozessoren)

Aktualisierungsstrategien

□ Write-through

- No-Write Allocation: Bei einem Write-Miss wird nur der Hauptspeicher und nicht der Cache aktualisiert
- Write-Allocation: Bei einem Write-Miss wird der Hauptspeicher und der Cache aktualisiert

□ Write-back: Bei Schreibzugriffen wird nur der Cache und nicht der Hauptspeicher aktualisiert.

Aktualisierungsstrategien

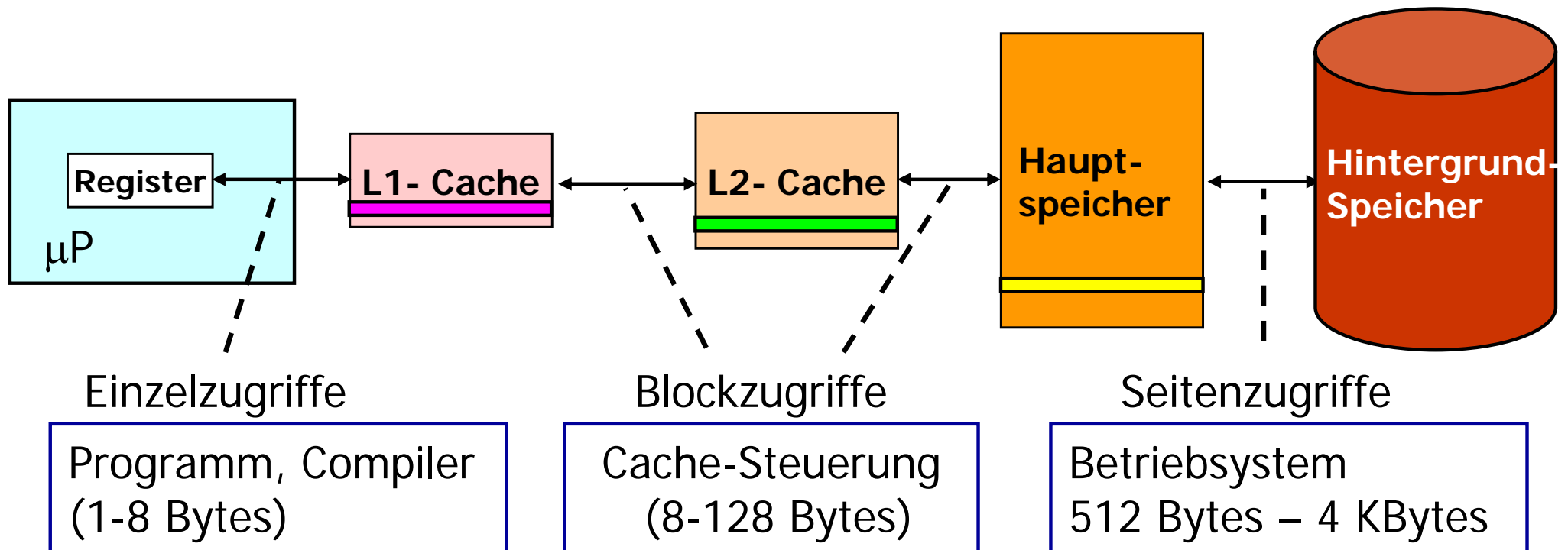
Cache-Operation	write through no-write-allocation	write through write allocation	Copy-back
Read-Hit	Cache-Datum → CPU	Cache-Datum → CPU	Cache-Datum → CPU
Read-Miss	Sp.-Block, Tag → Cache Sp.-Datum → CPU V = 1	Sp.-Block, Tag → Cache Sp.-Datum → CPU V = 1	Cache-Zeile → Speicher Sp.-Block, Tag → Cache Sp.-Datum → CPU V = 1, D = 0
Write-Hit	CPU-Datum → Cache, Speicher	CPU-Datum → Cache, Speicher	CPU-Datum → Cache D = 1
Write-Miss	CPU-Datum → Speicher	Sp.-Block, Tag → Cache V = 1 CPU-Datum → Cache, Speicher	Cache-Zeile → Speicher Sp.-Block, Tag → Cache V = 1 CPU-Datum → Cache D = 1

Nur dann erforderlich, wenn der Block gültig und Dirty ist (V=1, D=1)



Speicherhierarchie

Daten werden nur zwischen aufeinanderfolgenden Ebenen der Speicherhierarchie kopiert.



Speicherverwaltung

- ❑ Virtuelle Speicherkapazität > effektive Hauptspeicherkapazität
- ❑ Betriebssystem lagert bei Bedarf Speicherbereiche ein und aus
- ❑ Speicherverwaltungseinheiten (*memory management units, MMU*) unterstützt hardwaremäßig eindeutige Adressberechnung
- ❑ Abbildungsinformation in Übersetzungstabellen

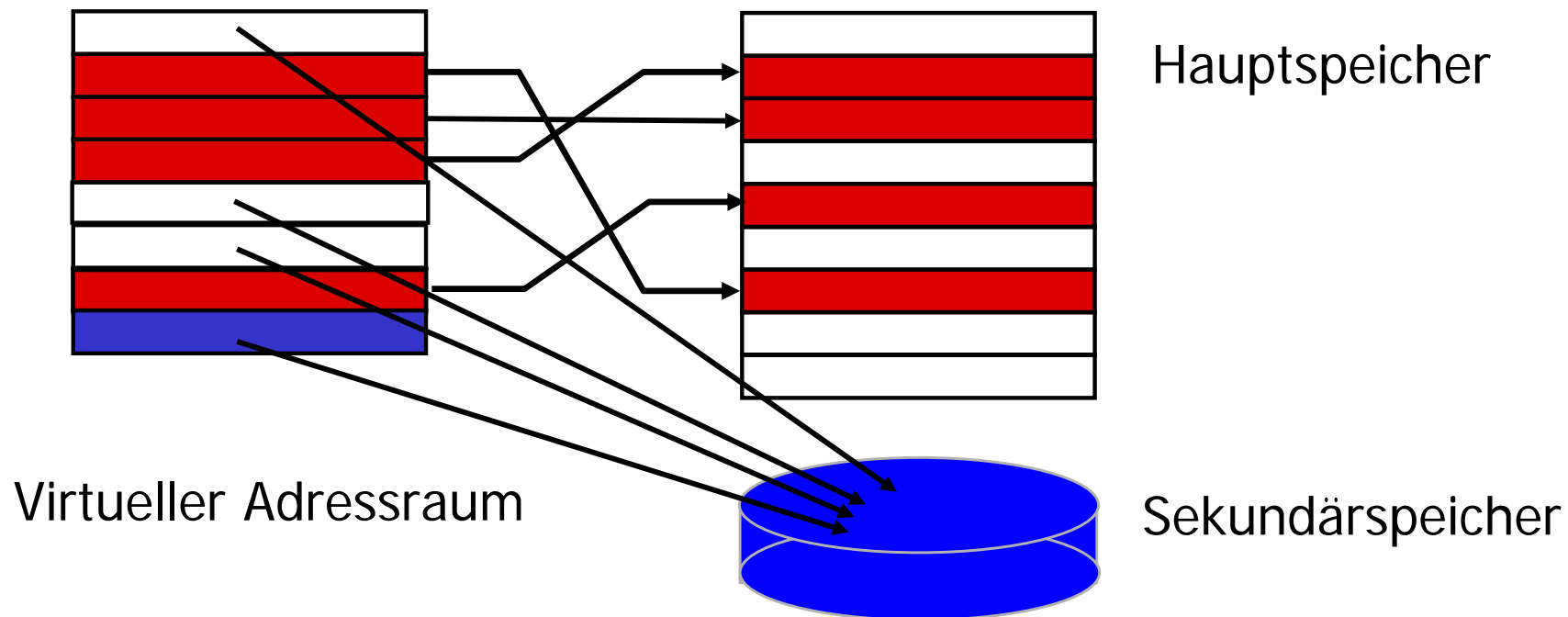
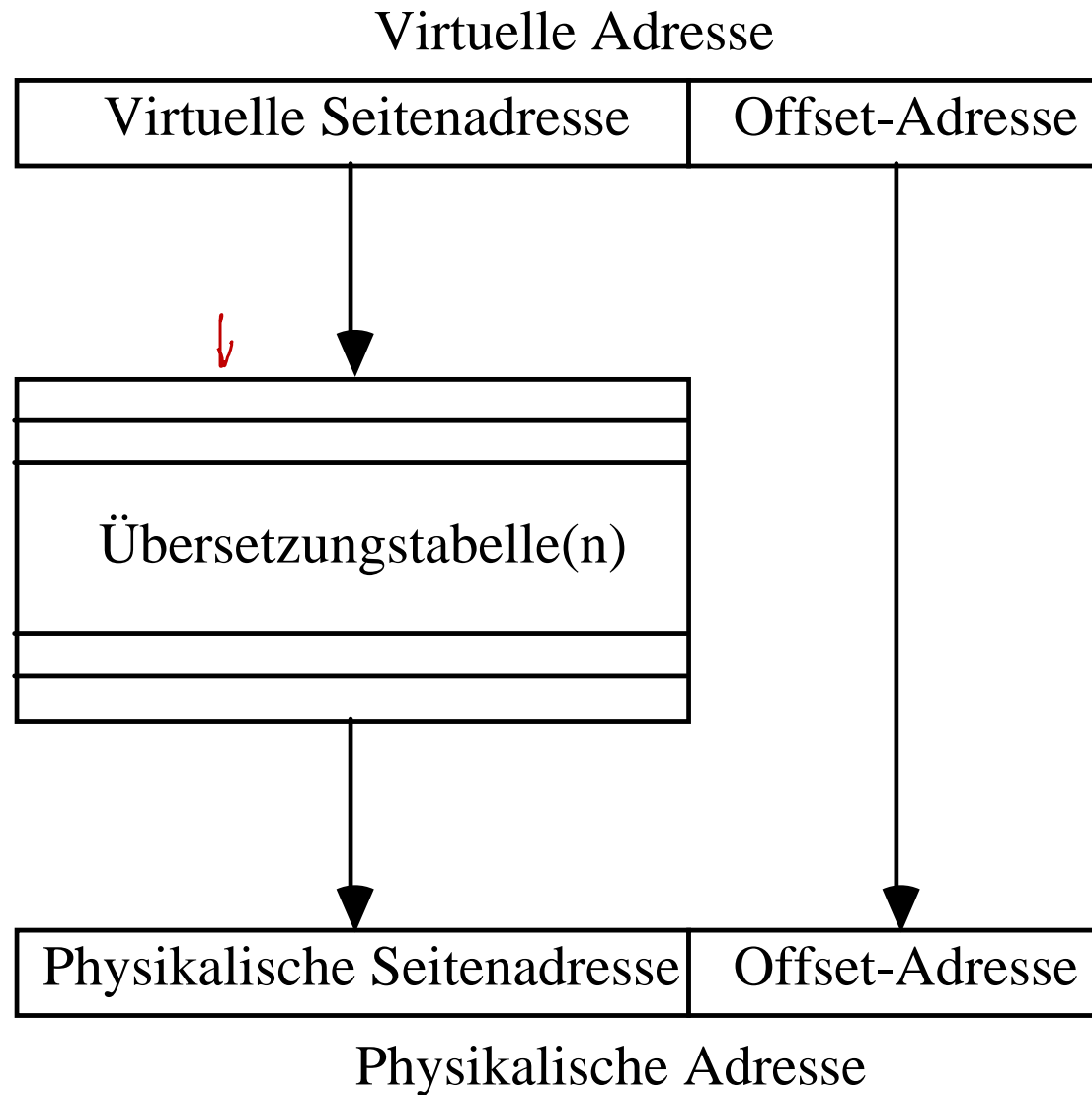


Abbildung virtueller auf physikalische Adressen



Speicherverwaltung

- ❑ Um den Umfang der Übersetzungstabellen gering zu halten, bezieht man die Abbildungsinformation nicht auf einzelne Adressen, sondern auf zusammenhängende Adressbereiche
- ❑ Es gibt zwei Möglichkeiten:
 - **Segmentierung**
 - **Seitenwechsel-Verfahren**

Segmentierungs- und Seitenwechselfverfahren

Es existieren zwei grundlegende Verfahren zur virtuellen Speicherverwaltung (Segmentierung und Seitenwechsel)

Segmentierung

- Hierbei wird der virtuelle Adressraum in Segmente verschiedener Länge zerlegt.
- Jedem Prozess sind ein oder mehrere Segmente z. B. für den Programmcode und die Daten, zugeordnet.
- Die einzelnen Segmente enthalten logisch zusammenhängende Informationen (Programm- und Datenmodule) und können relativ groß sein.

Segmentierungs- und Seitenwechselfverfahren

Aufteilung in Seiten

- Hierbei wird der logische und der physikalische Adressraum in "Segmente fester Länge", die sogenannten Seiten (pages) unterteilt.
- Die Seiten sind relativ klein (256 Byte - 4 kByte)
- Ein Prozess wird auf viele dieser Seiten verteilt (keine logischen Zusammenhänge wie bei der Segmentierung)

Segmentierung

Vorteile:

- Segmentierung spiegelt logische Programmstruktur wieder.
- durch große Segmente relativ seltener Datentransfer.

Nachteile:

- wenn Datentransfer, dann jedoch umfangreich.
- besteht ein Programm nur aus einem Code- und Daten-Segment (wird vom Compiler oder Benutzer festgelegt), so muss es vollständig eingelagert werden.

Seitenaufteilung

□ Vorteile:

- durch kleine Seiten wird nur der wirklich benötigte Teil eines Programms eingelagert.
- geringerer Verwaltungsaufwand als Segmentierung

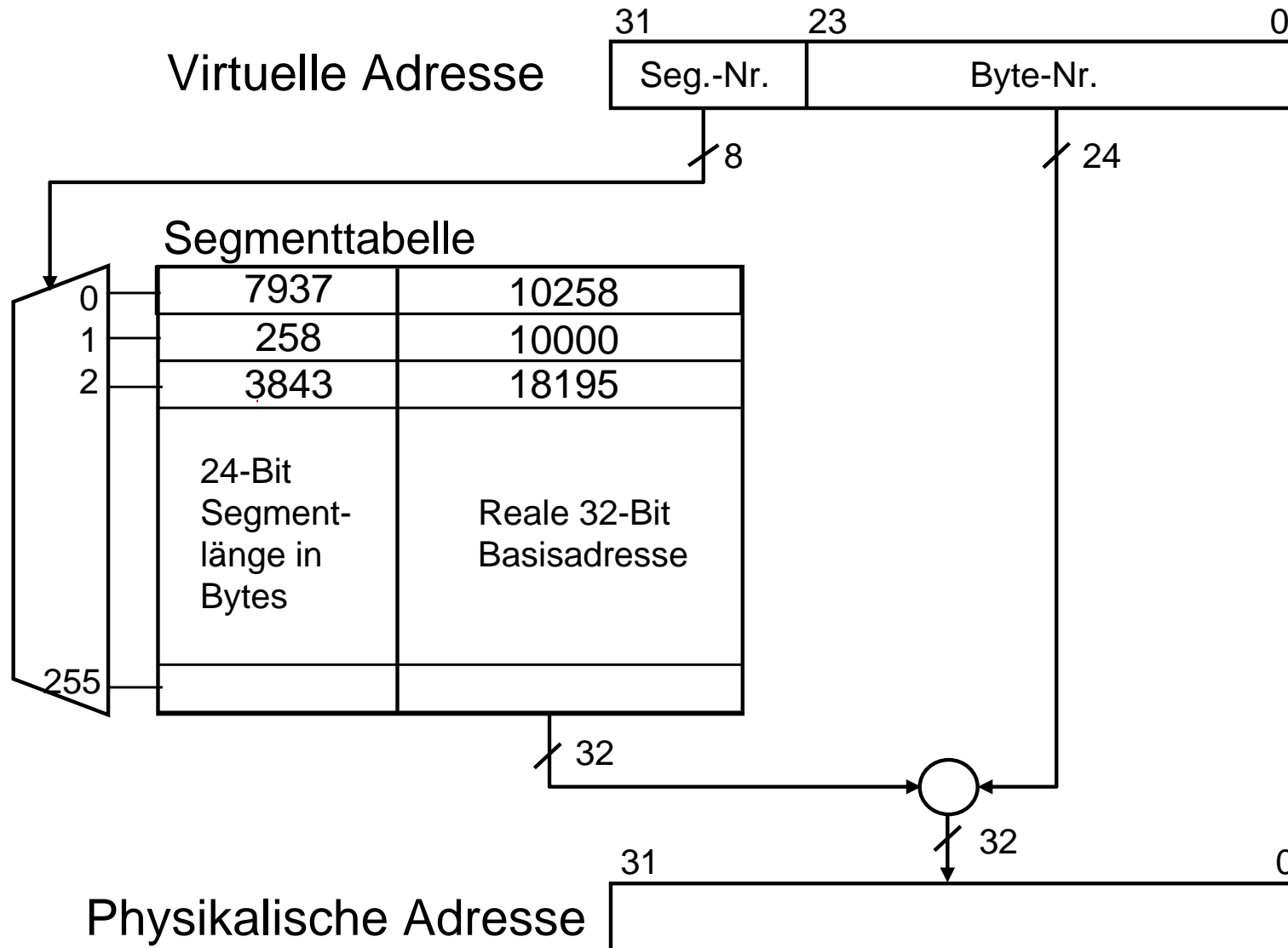
□ Nachteil:

- häufiger Datentransfer

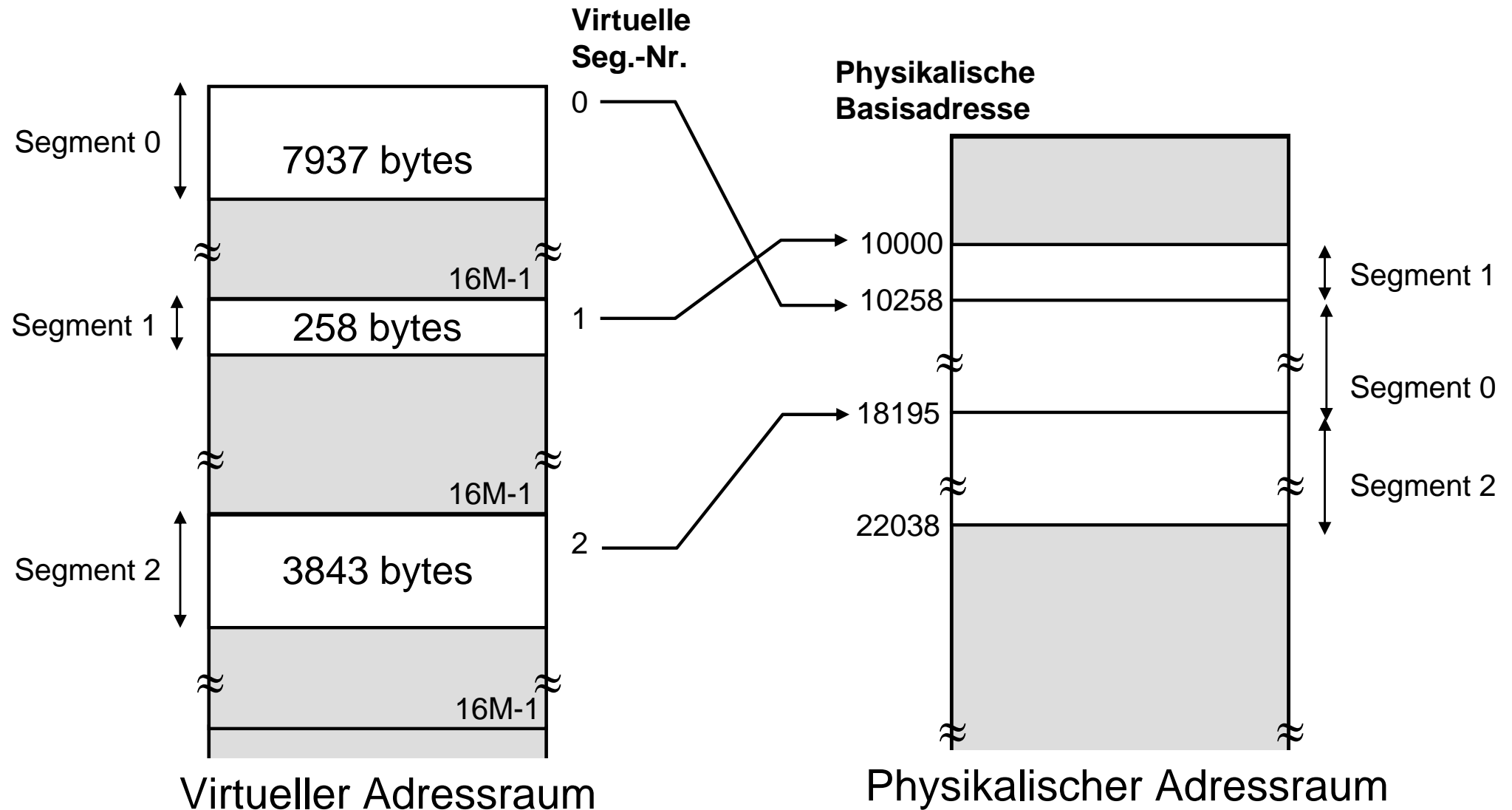
Ältere Prozessoren unterstützten jeweils nur ein Verfahren, z. B. Segmentierung bei 80286, 68010 und Seitenwechsel bei Z8003/4, National 32000

Heutige Mikroprozessoren, z.B. 80386, 80486, Pentium, ... unterstützen beide Verfahren

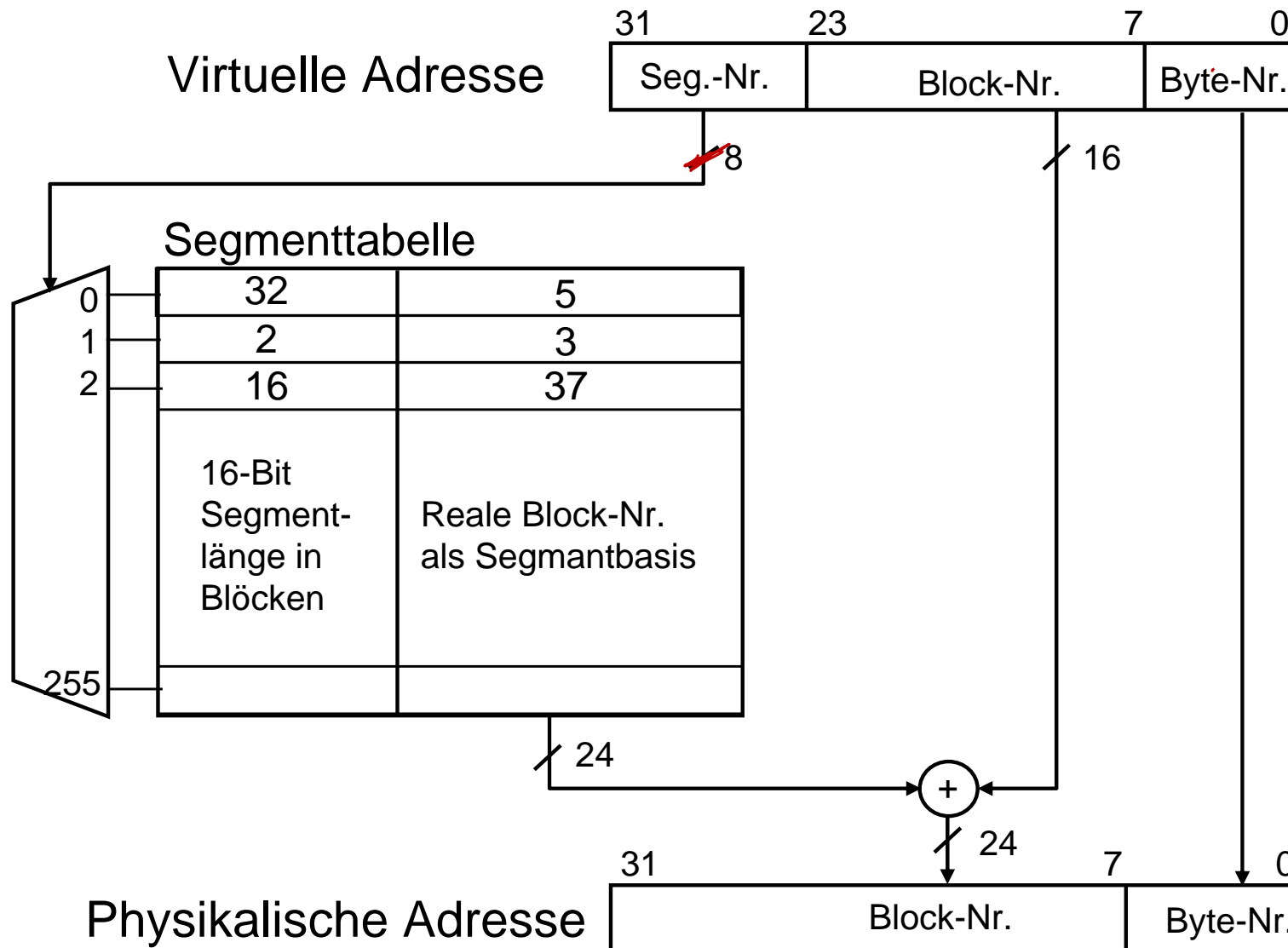
Segmentbasierte Speicherverwaltung



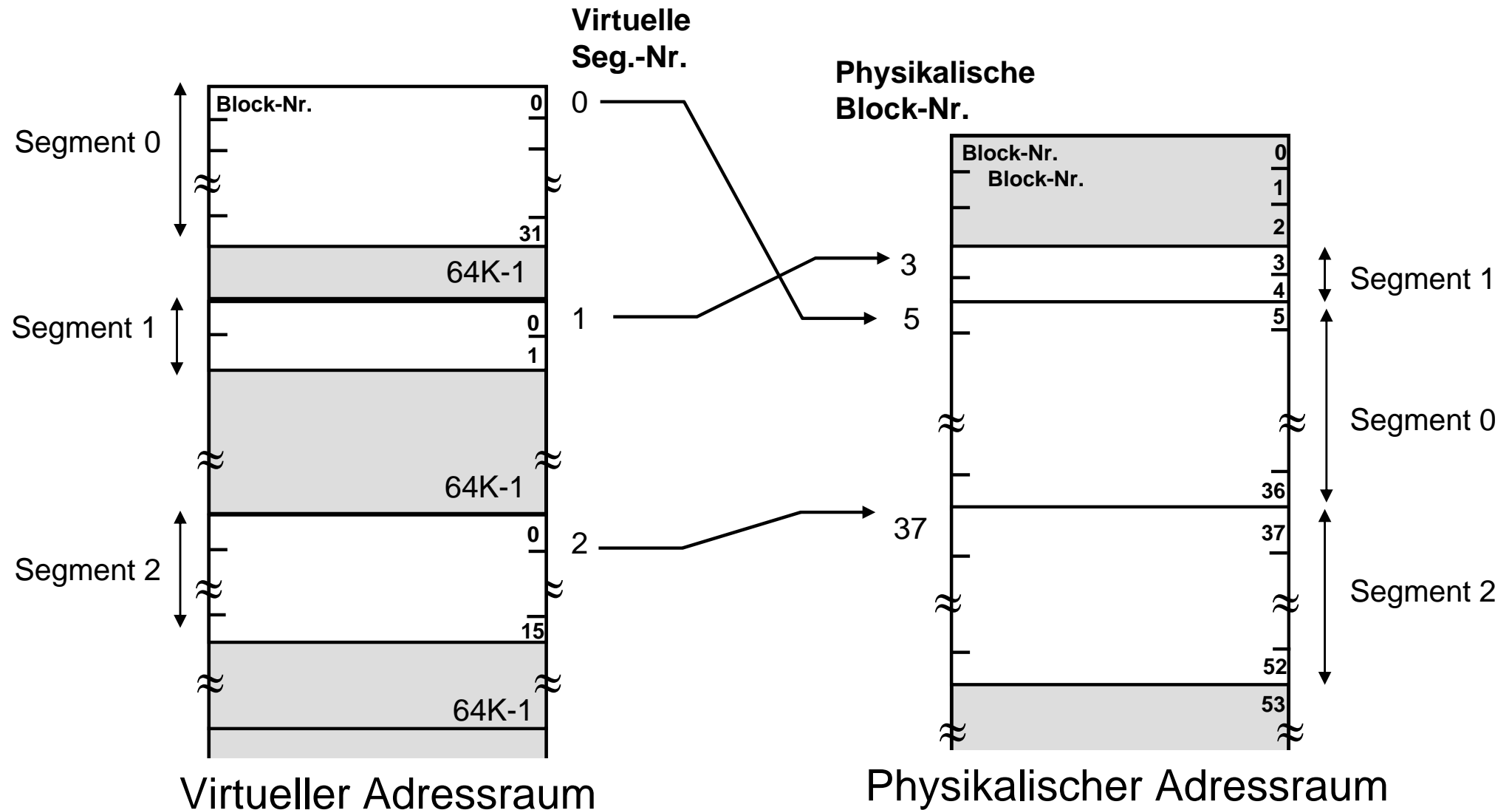
Virtueller und physikalischer Adressraum



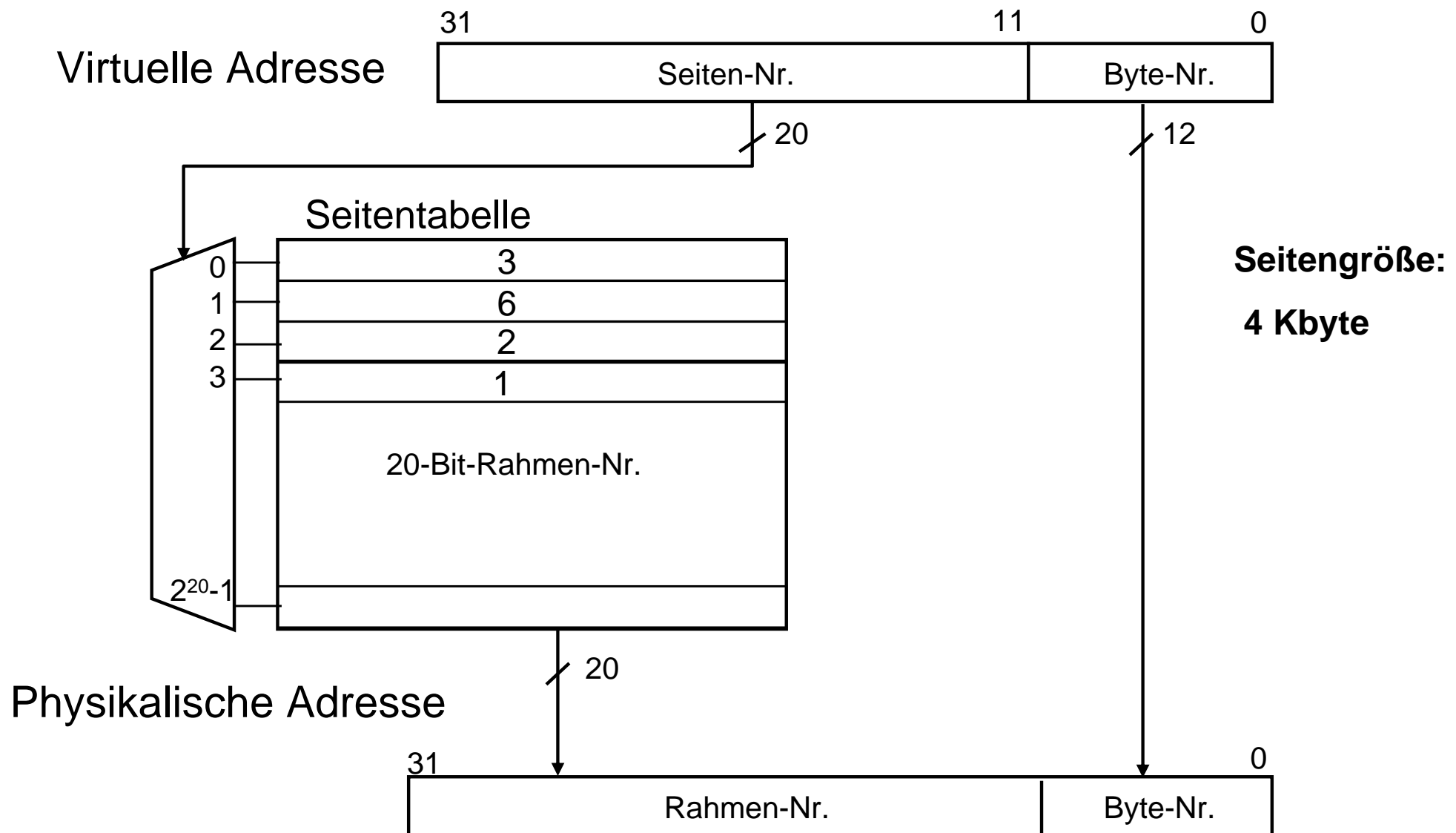
Segmentbasierte Speicherverwaltung



Virtueller und physikalischer Adressraum



Seitenwechsel (Paging)



Virtueller und physikalischer Adressraum

