
5. Übung

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

Speicherhierarchie

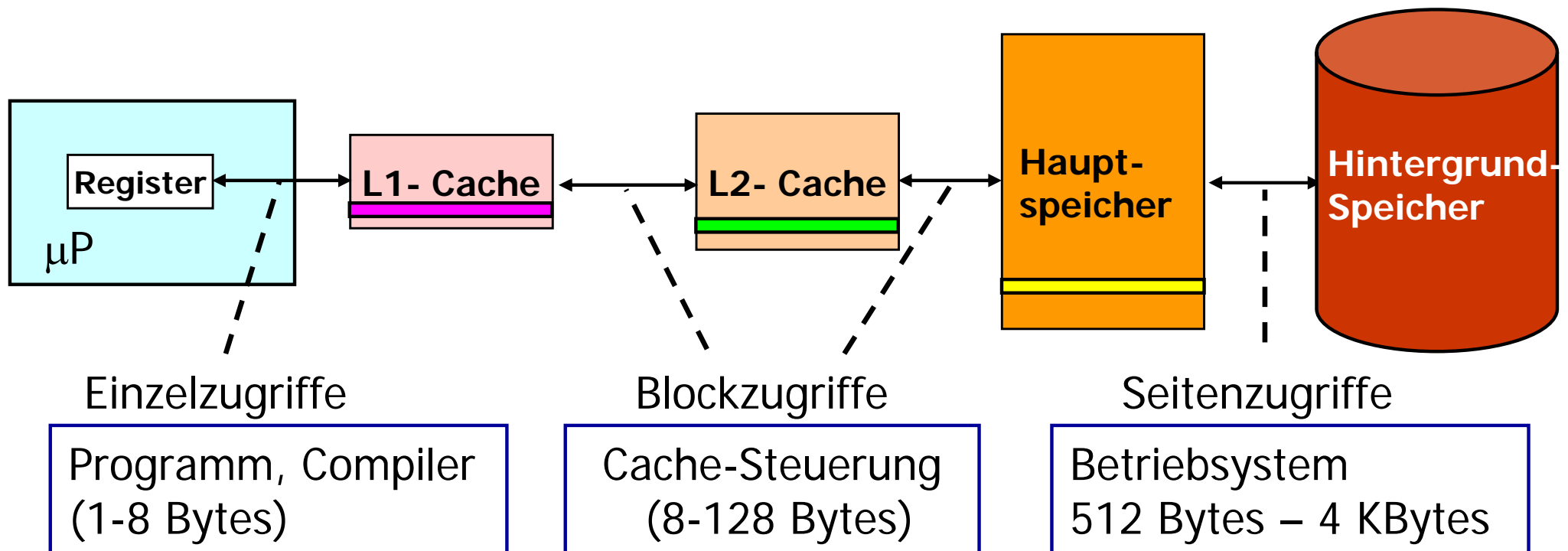
Speicherhierarchie zum Ausgleich der unterschiedlichen Zugriffszeiten der CPU und des Hauptspeichers.

2 Strategien:

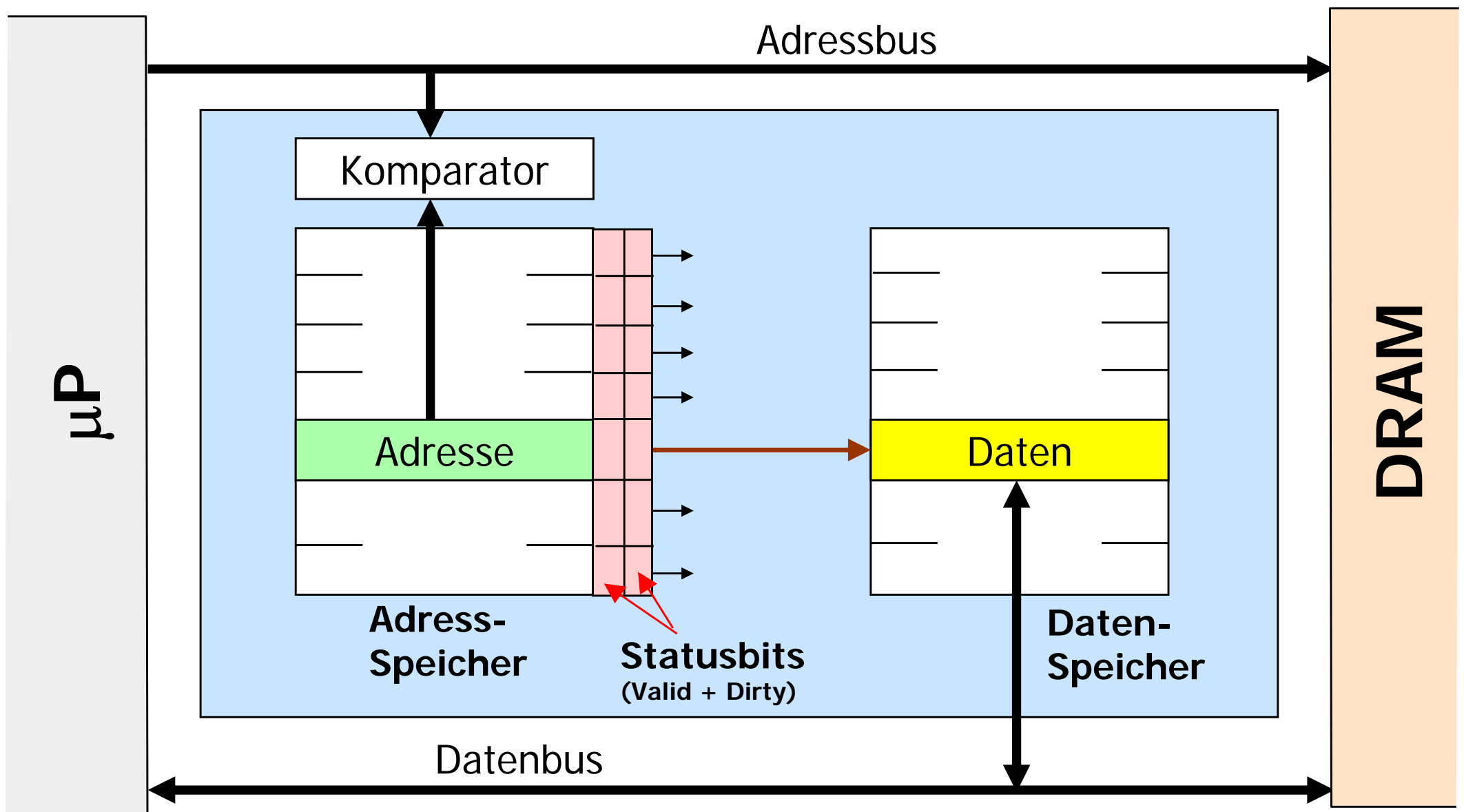
- **Cache-Speicher:**
Kurze Zugriffszeiten ➡ Beschleunigung des Prozessorzugriffs
- **Virtueller Speicher:**
Vergrößerung des tatsächlich vorhandenen Hauptspeichers (z. B. bei gleichzeitiger Bearbeitung mehrerer Prozesse)

Speicherhierarchie

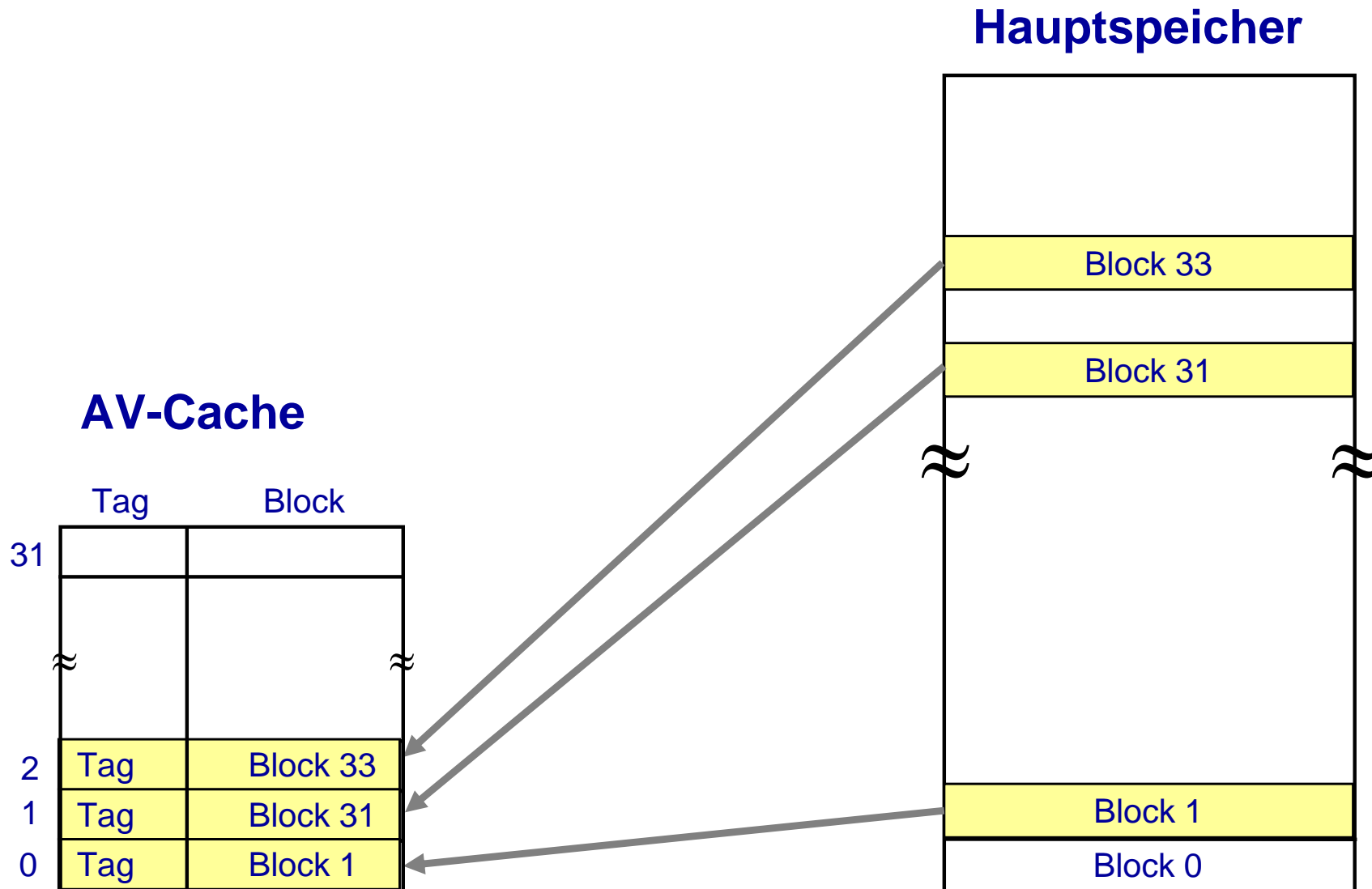
Daten werden nur zwischen aufeinanderfolgenden Ebenen der Speicherhierarchie kopiert.



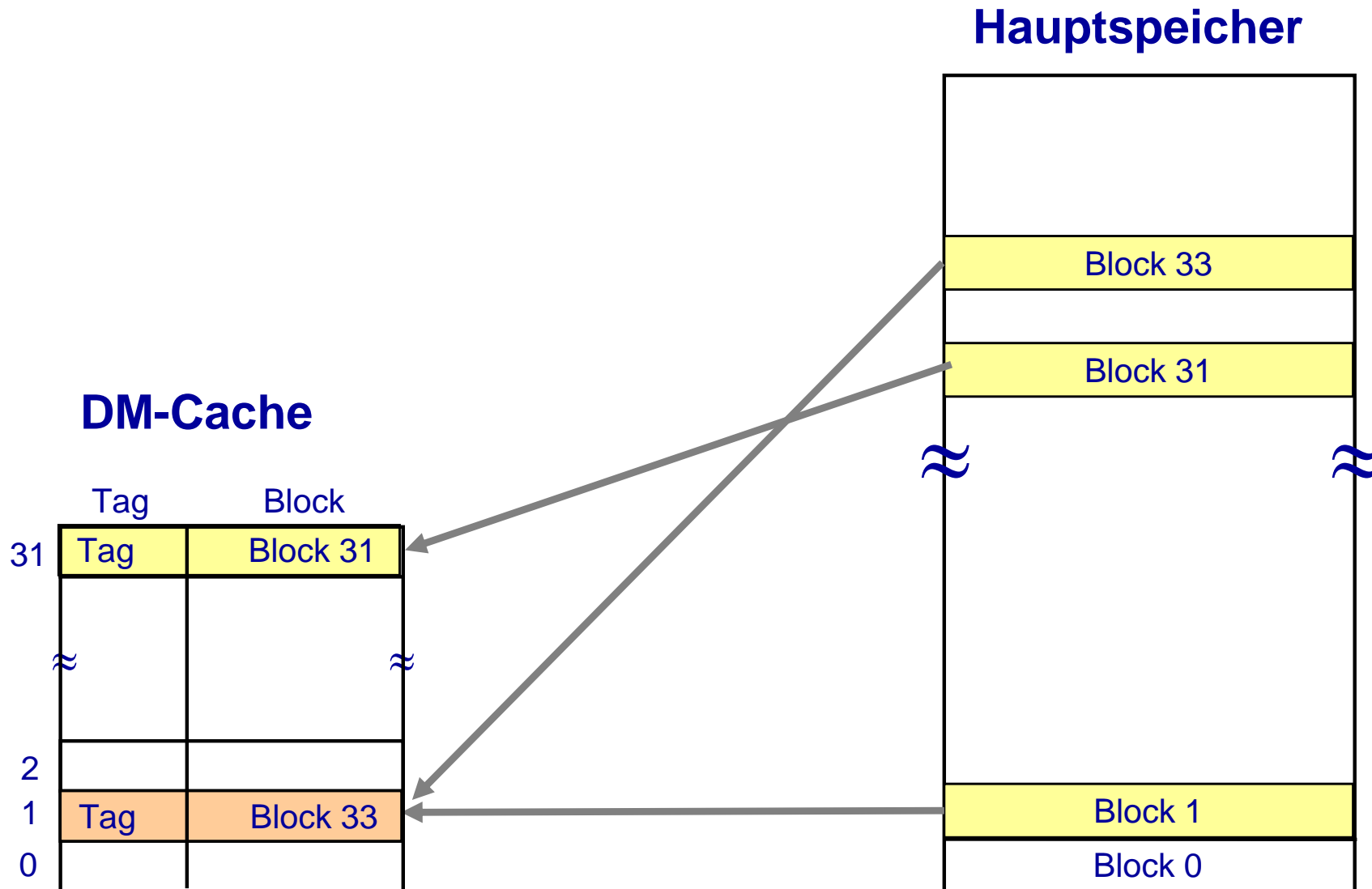
Aufbau eines Cache-Speichers



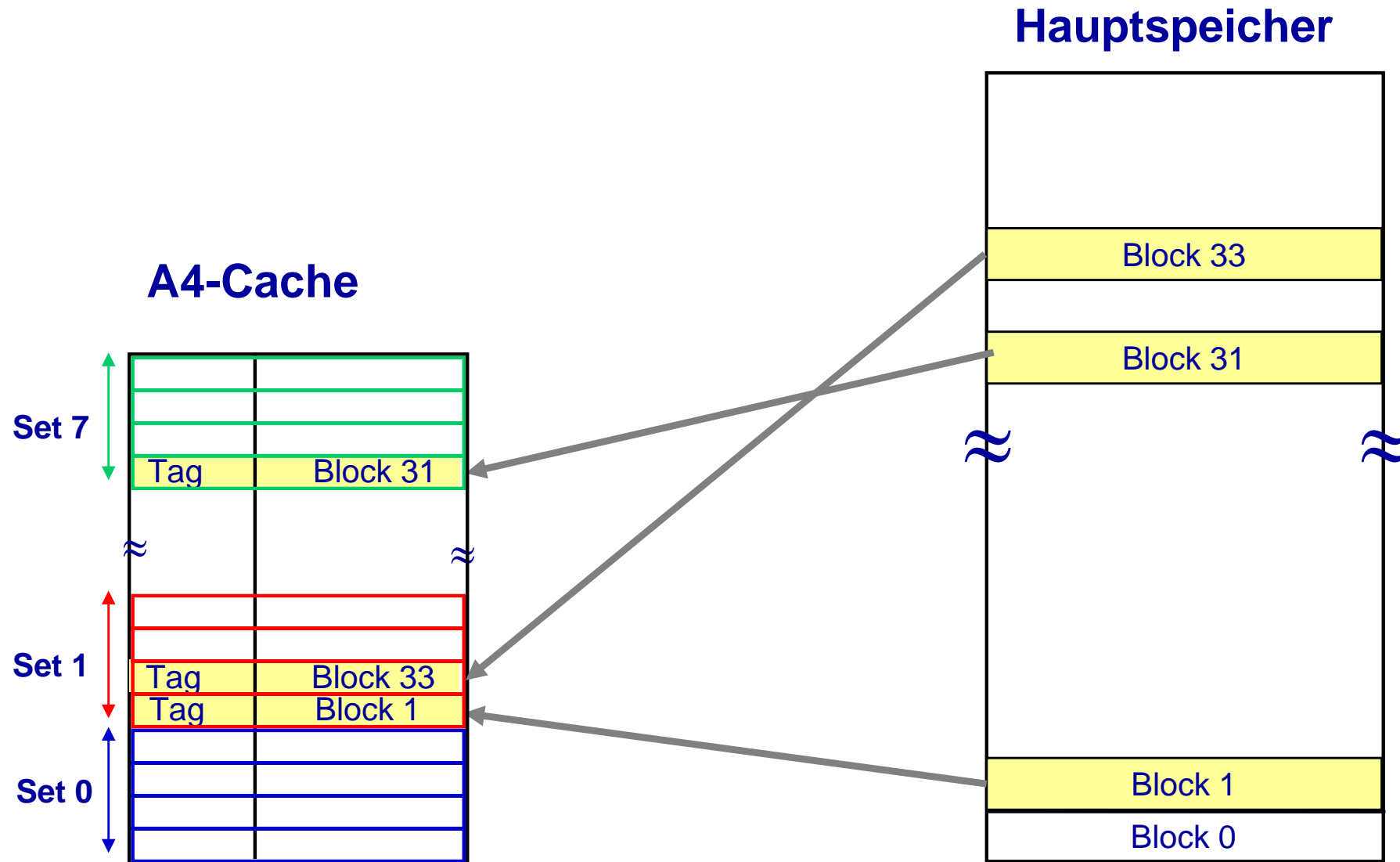
AV-Cache



DM-Cache



A4-Cache



Wohin wird ein Block abgebildet?

Blocknummer = (Hauptspeicheradresse) **div** (Blockgröße)

Zeilennummer = (Blocknummer) **mod** (Zeilenanzahl)

Satznummer = (Blocknummer) **mod** (Satzanzahl)

- 8_{16} Blocknummer: $8_{16} \text{ div } 8 = 1$
 Zeilennummer: $1 \text{ mod } 32 = 1$
 Satznummer: $1 \text{ mod } 8 = 1$
- $F8_{16}$ Blocknummer: $F8_{16} \text{ div } 8 = 31$
 Zeilennummer: $31 \text{ mod } 32 = 31$
 Satznummer: $31 \text{ mod } 8 = 7$
- 108_{16} Blocknummer: $108_{16} \text{ div } 8 = 33$
 Zeilennummer: $33 \text{ mod } 32 = 1$
 Satznummer: $33 \text{ mod } 8 = 1$

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:



Aufgabe 5.2

Cache-Kapazität = **128 Kbyte**

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

Anzahl der Einträge im Cache:

Wie viele Blöcke können sich gleichzeitig befinden:

$$\frac{128 \text{ KByte}}{16 \text{ Byte}} = 8K = 8.1024 \text{ Einträge}$$

Aufgabe 5.3

Cache-Kapazität = **128 Kbyte**

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

Organisation:

- mit 12 Bits Index können 2^{12} Zeilen/Sätze adressiert werden. \rightarrow 4K Zeilen/Sätze
- Anzahl der Cache-Einträge = 8K
- $\frac{8K}{4K} = 2 \rightarrow$ 2-fach asso. Cache (A2)

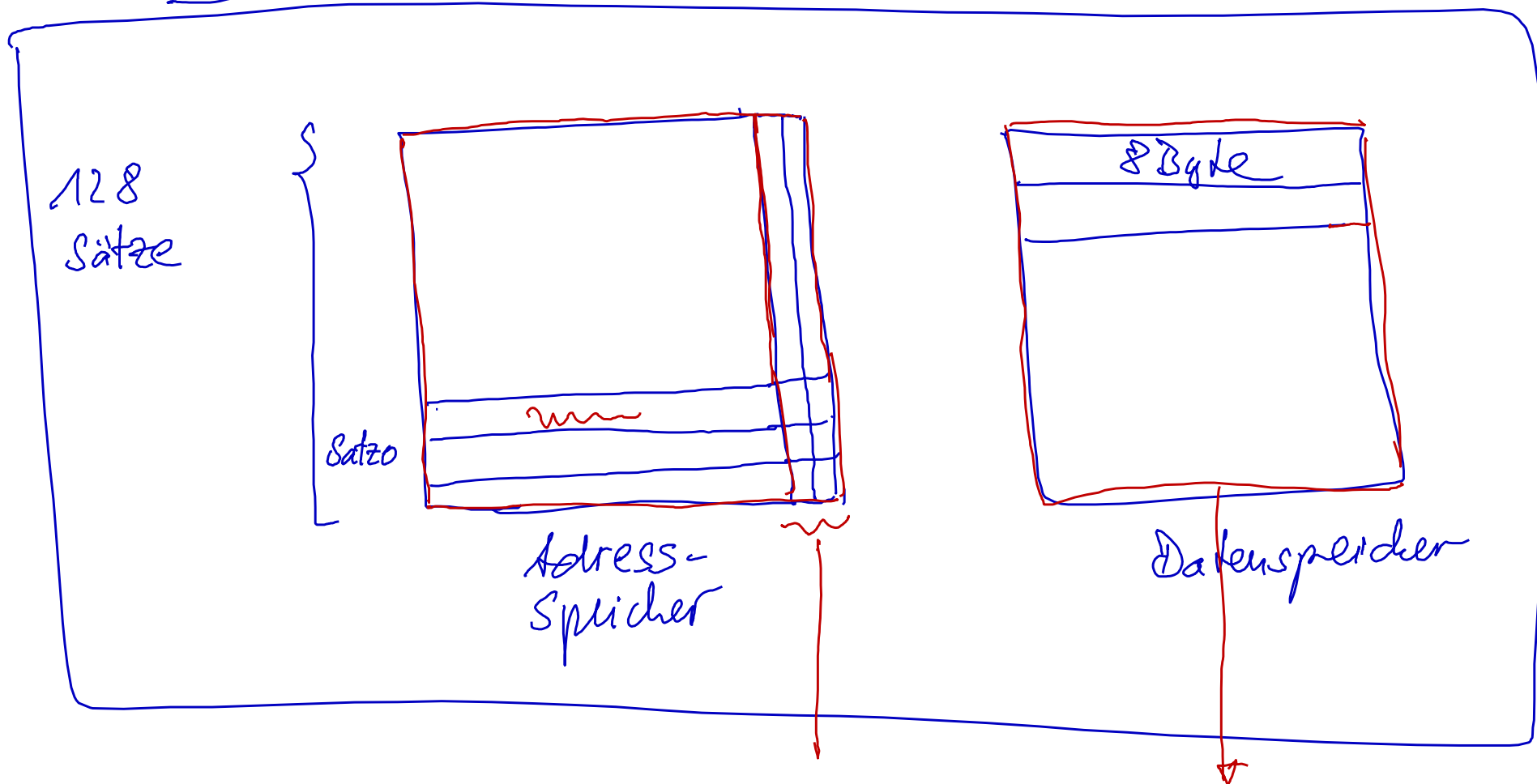
Aufgabe 6

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.

A3



$(Tag \cdot 128 \cdot 3) \text{ Bits}$ $(2 \cdot 128 \cdot 3) \text{ Bit}$ $(8 \cdot 128 \cdot 3) \text{ Byte}$

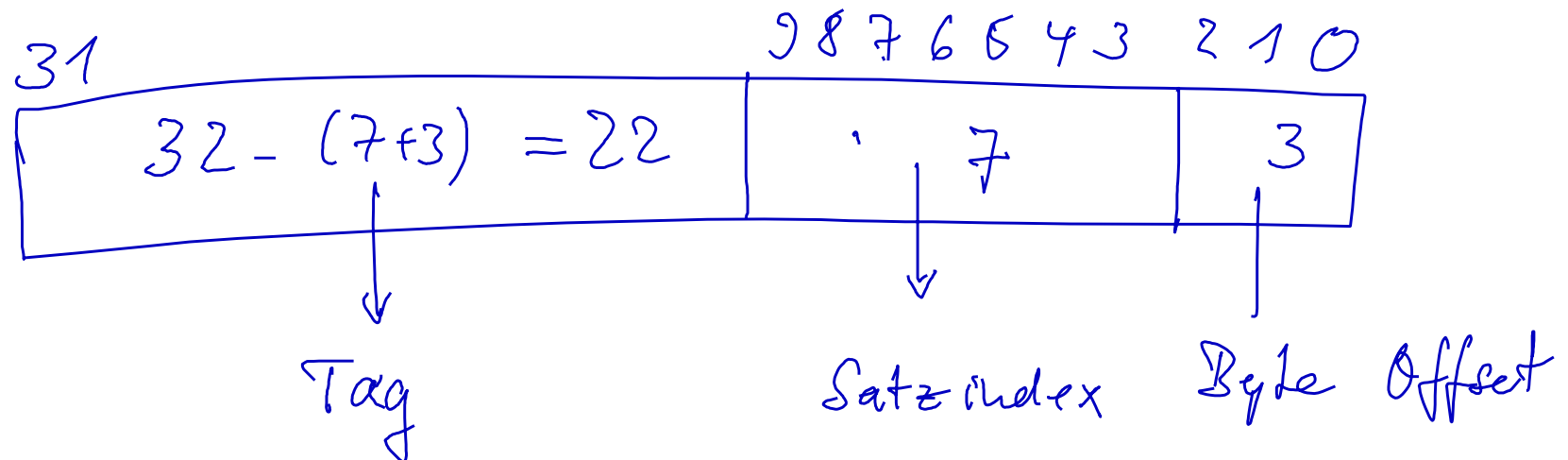
Lösung 6

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



Erforderlicher Speicherbedarf:

• pro Zeile:

$$(22 + 2) \text{ Bit} + 8 \text{ Byte} =$$

$$3 \text{ Byte} + 8 \text{ Byte} = 11 \text{ Byte}$$

• für den gesamten Cache:

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

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)



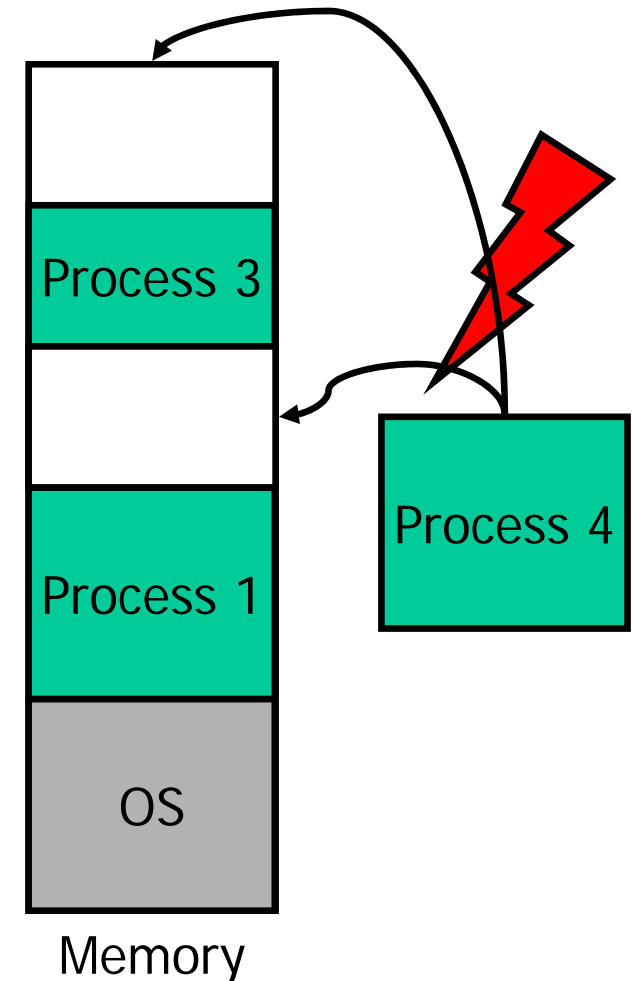
TI- Probeklausur

- am 12. Juli 2007
- 14:00 – 15:00 Uhr
- Audimax-Hörsaal
- Anmeldung im Tutorium



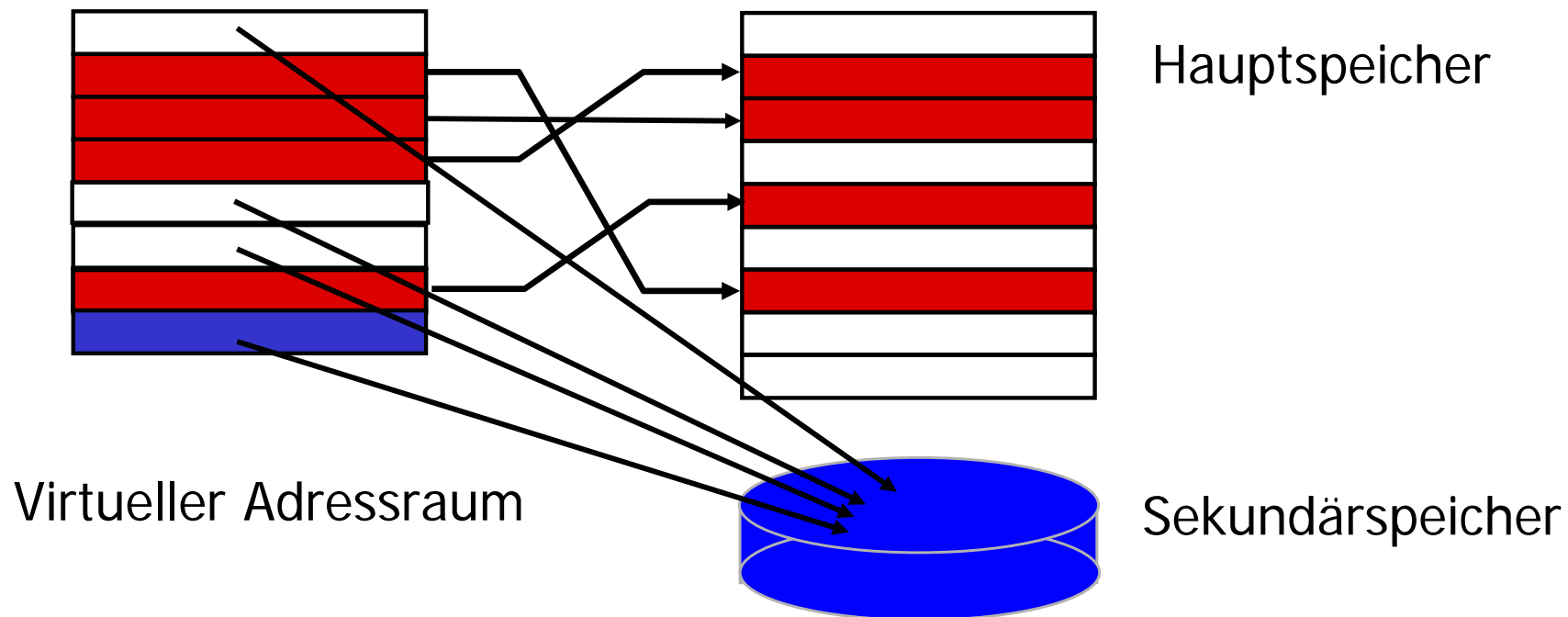
Speicherverwaltung

- ❑ Hauptspeicherkapazität ist begrenzt
 - Prozess benötigt mehr Speicher als der Hauptspeicher
 - Mehr aktive Prozesse als der Hauptspeicher "vertragen" kann.
- ❑ Effiziente Schutzmechanismen



Speicherverwaltung

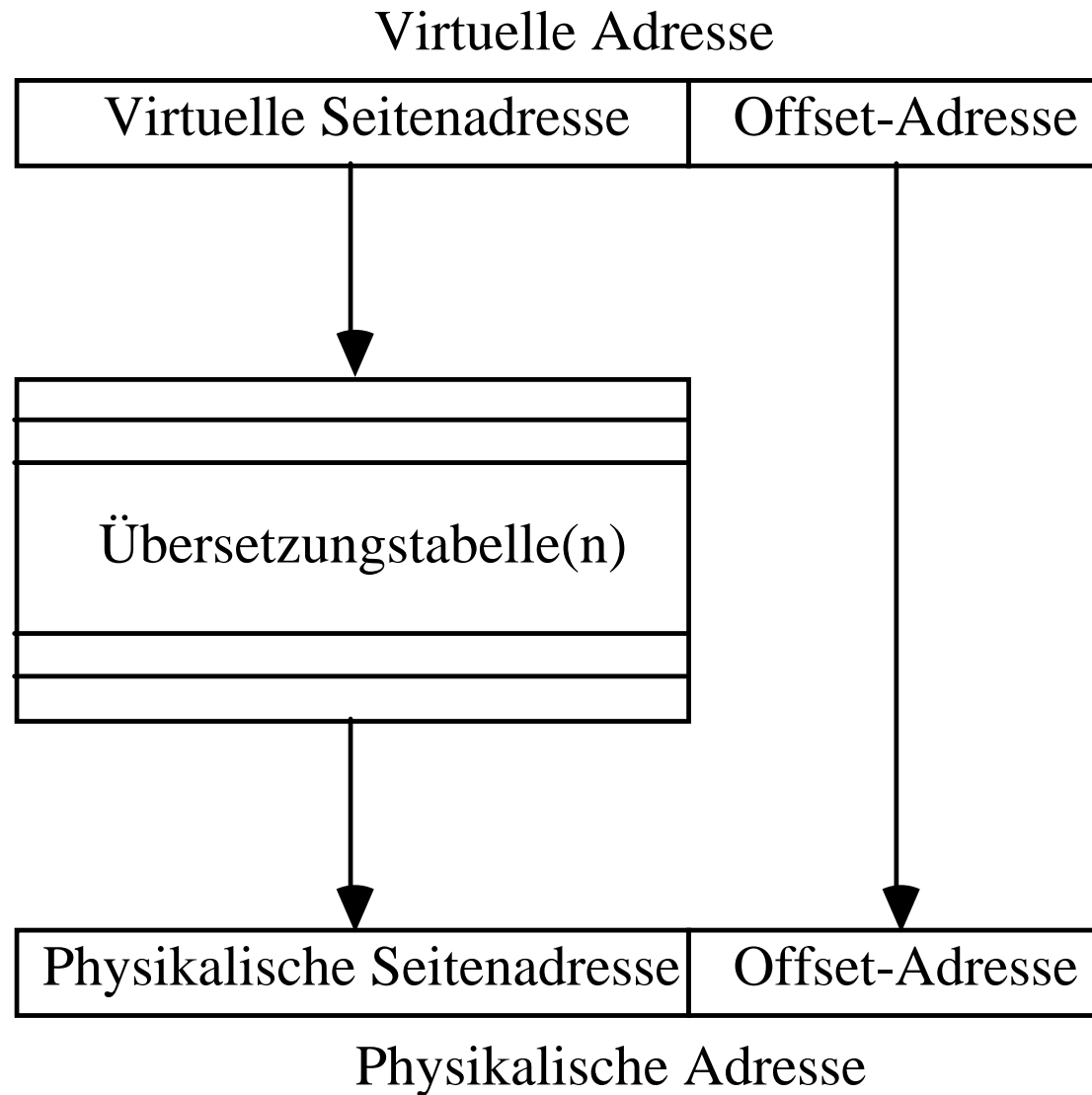
- ❑ Virtuelle Speicherkapazität > effektive Hauptspeicherkapazität
- ❑ Betriebssystem lagert bei Bedarf Speicherbereiche ein und aus
- ❑ Speicherverwaltungseinheiten (*memory management units, MMU*) unterstützt hardwaremäßig eindeutige Adressberechnung
- ❑ Abbildungsinformation in Übersetzungstabellen



Speicherverwaltung

- ❑ Das **Betriebssystem** führt Buch über die freien Speicherbereiche und lagert bei nicht ausreichendem Freiplatz im Hauptspeicher diejenigen Speicherinhalte auf den Hintergrundspeicher aus, die gegenüber ihrem Originalen auf dem Hintergrundspeicher verändert worden sind.
- ❑ Die eindeutige Abbildung des großen virtuellen Speichers auf die effektive Hauptspeicherkapazität wird von der Hardware durch **Speicherverwaltungseinheiten** unterstützt (*memory management units, MMU*)
- ❑ Die erforderliche Abbildungsinformation stellt das Betriebssystem in Form einer oder mehrerer **Übersetzungstabellen** zur Verfügung.

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**

Speicherverwaltung

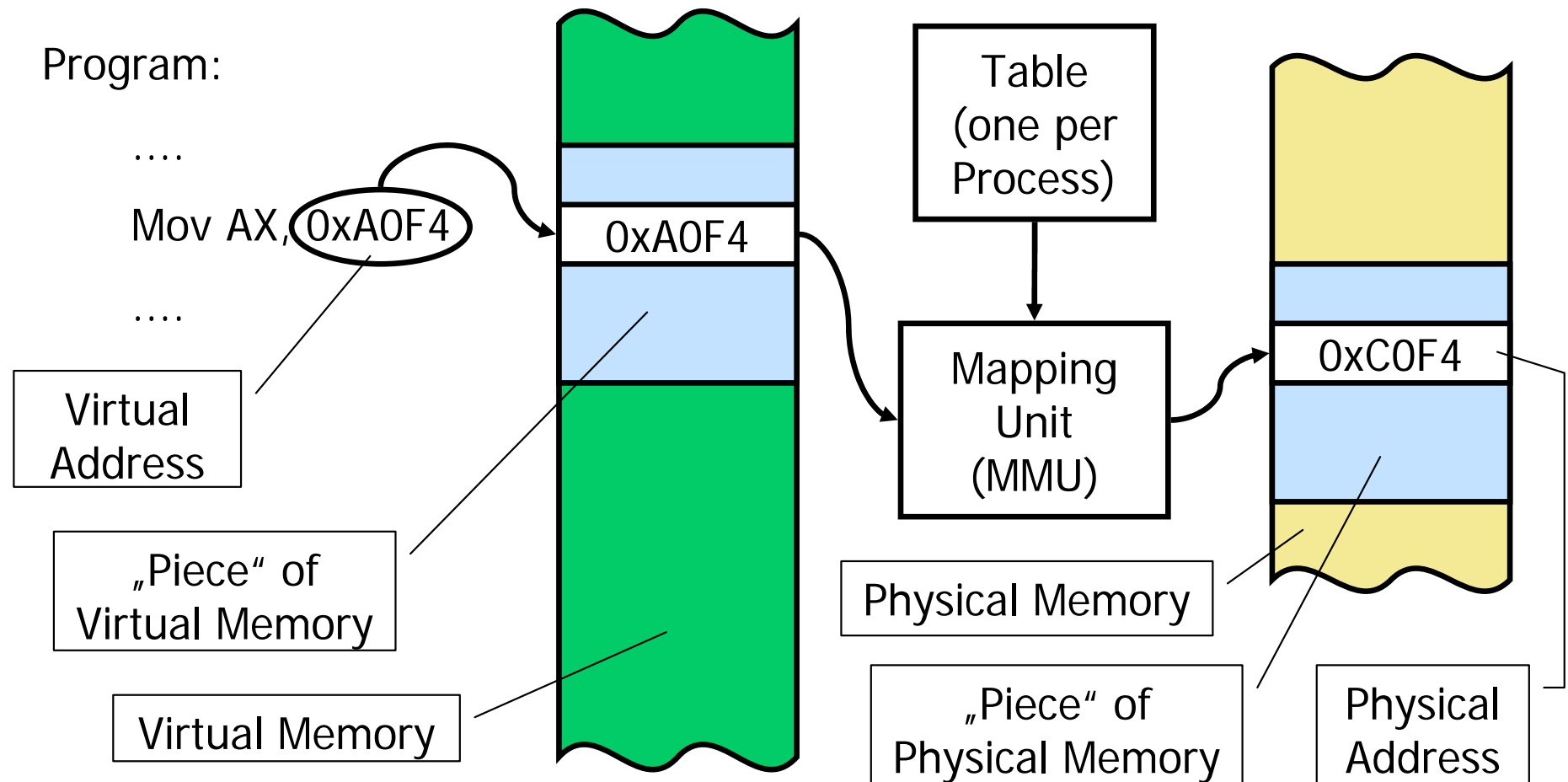
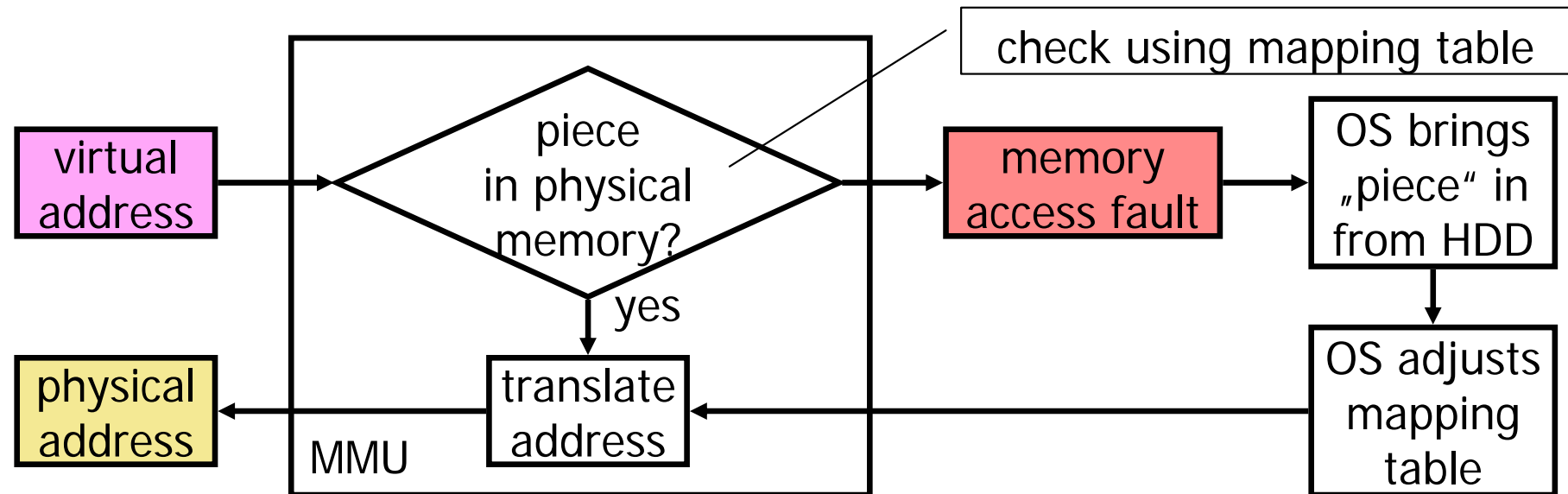


Abbildung: virtuell → physikalisch

- Jeder Prozess hat seinen virtuellen Adressraum und seine Abbildungstabelle



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

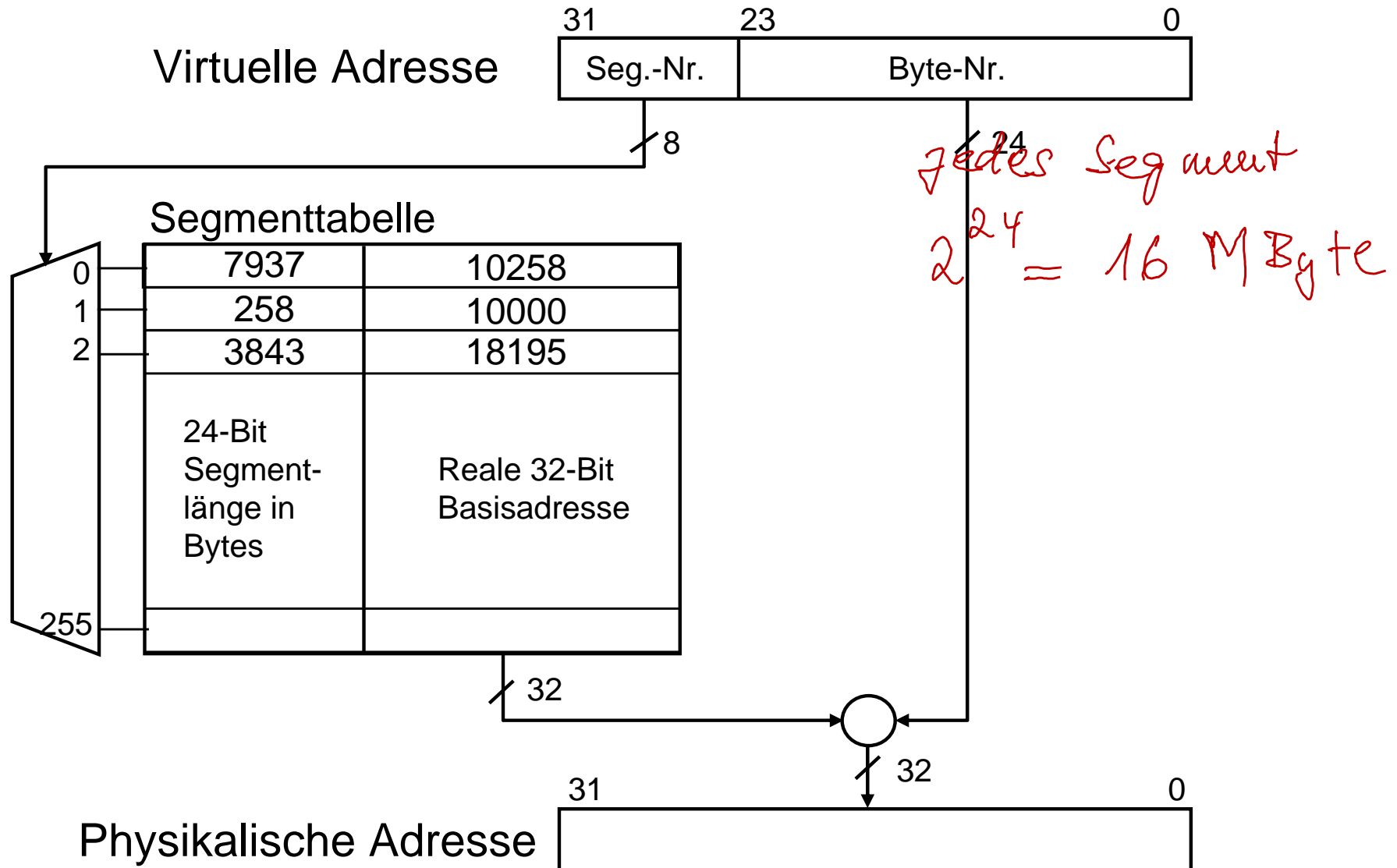
□ Vorteile:

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

□ Nachteil:

- häufiger Datentransfer

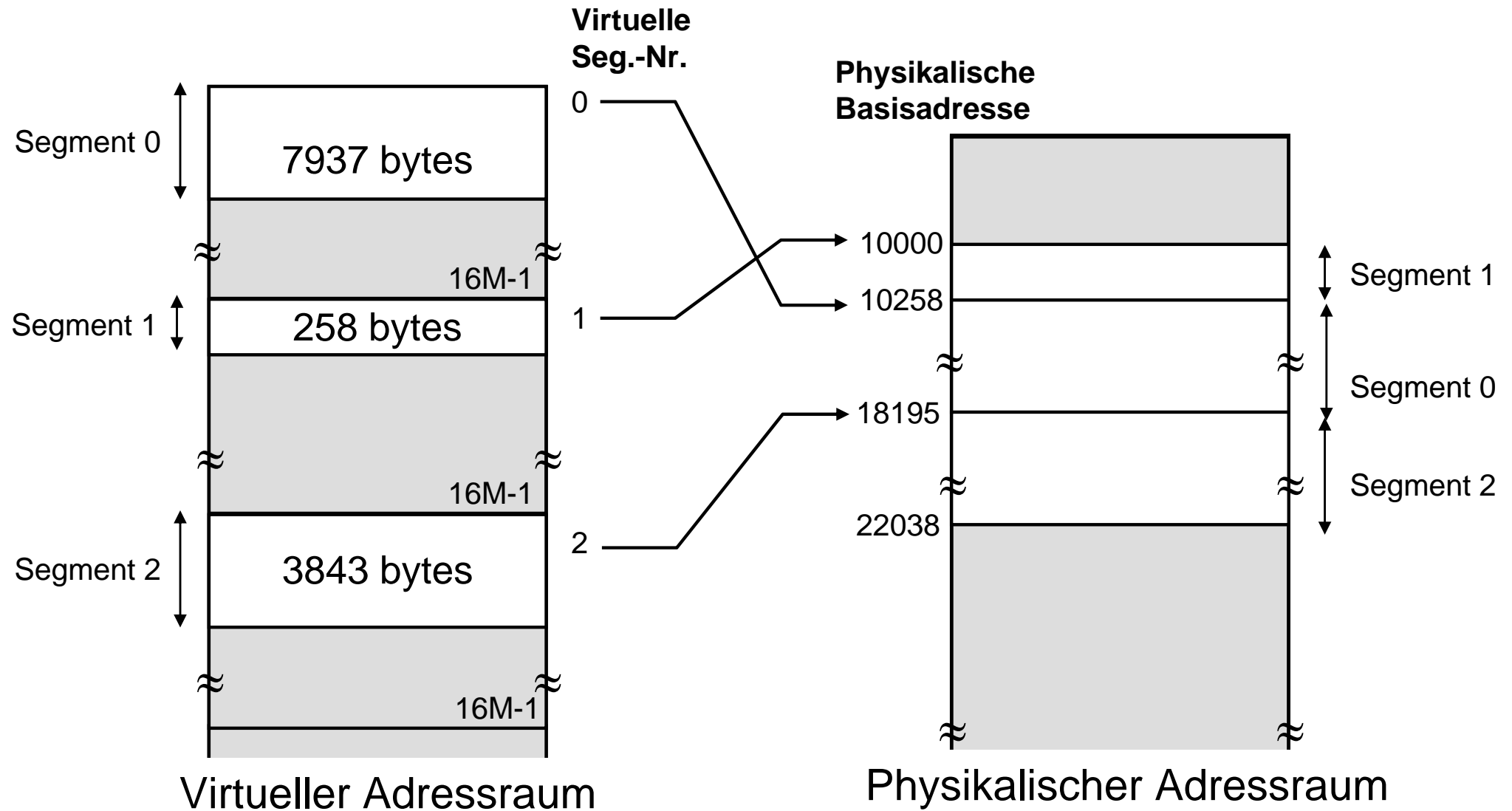
Segmentbasierte Speicherverwaltung



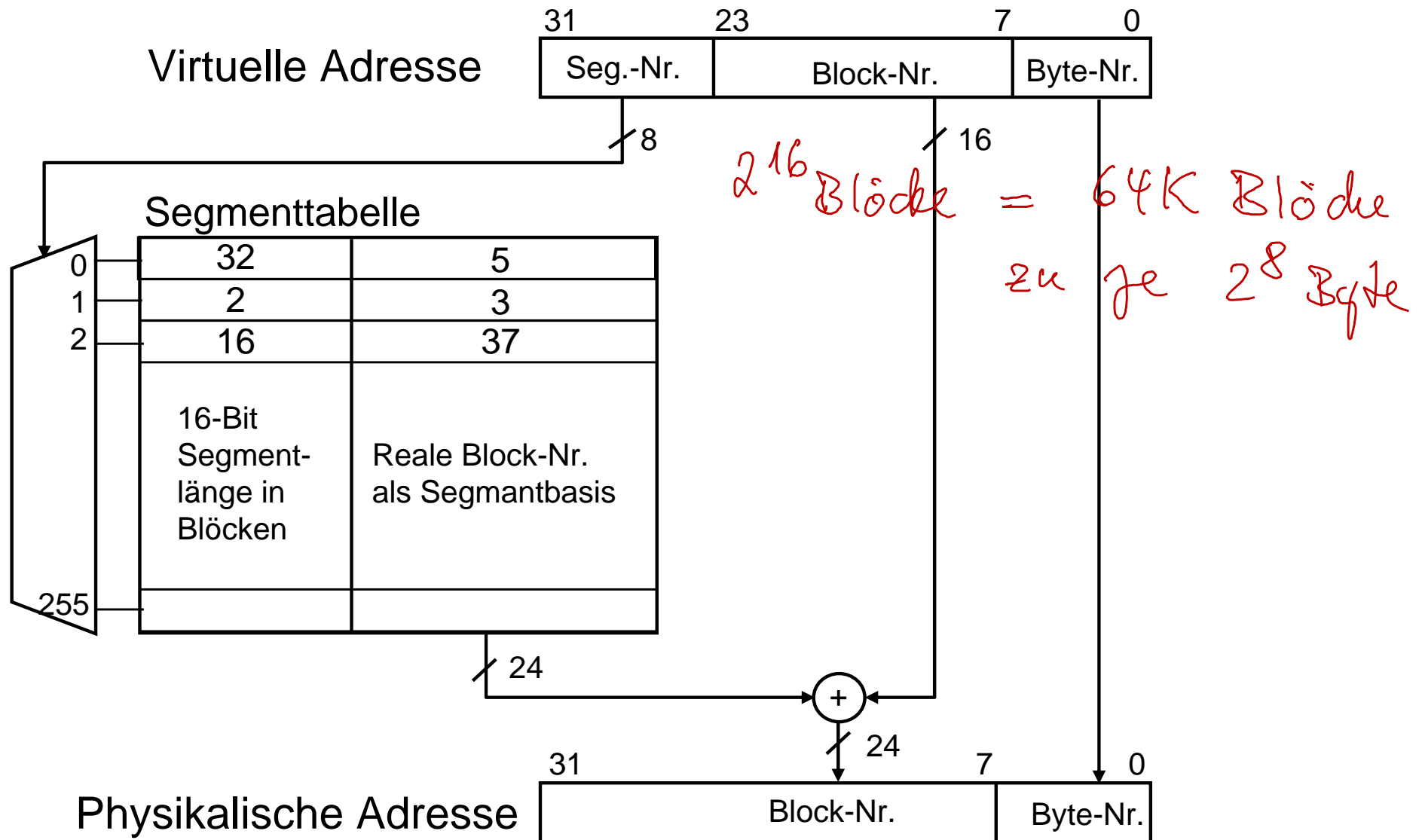
Segmentbasierte Speicherverwaltung

- ❑ Virtuelle Adresse wird in eine Segmentnummer (n höherwertige Bits der virtuellen Adresse) als Kennung eines Segments und in eine Bytenummer (verbleibenden m Bits der Adresse) als Abstand zum Segmentanfang unterteilt.
- ❑ Maximale virtuelle Segmentanzahl = 2^n , Maximale Segmentgröße = 2^m
- ❑ Die Adressabbildung erfolgt über eine Segmenttabelle (im Registerspeicher der MMU).
- ❑ Reale Adresse ergibt sich aus der Segmentbasisadresse, zu der die virtuelle Byte-Nummmmer addiert wird.
- ❑ Segmentlängenangaben in der Segmenttabelle, um segment-überschreitende Zugriffe feststellen und ggf. verhindern zu können.
- ❑ Bei Segmenten mit einer geringeren Größe als 2^m , gilt der ungenutzte Raum als Verschnitt.
- ❑ Gute Ausnutzung des Hauptspeichers, wenn man die Segmentgrenzen an jeder Byteadresse zulässt.

Virtueller und physikalischer Adressraum



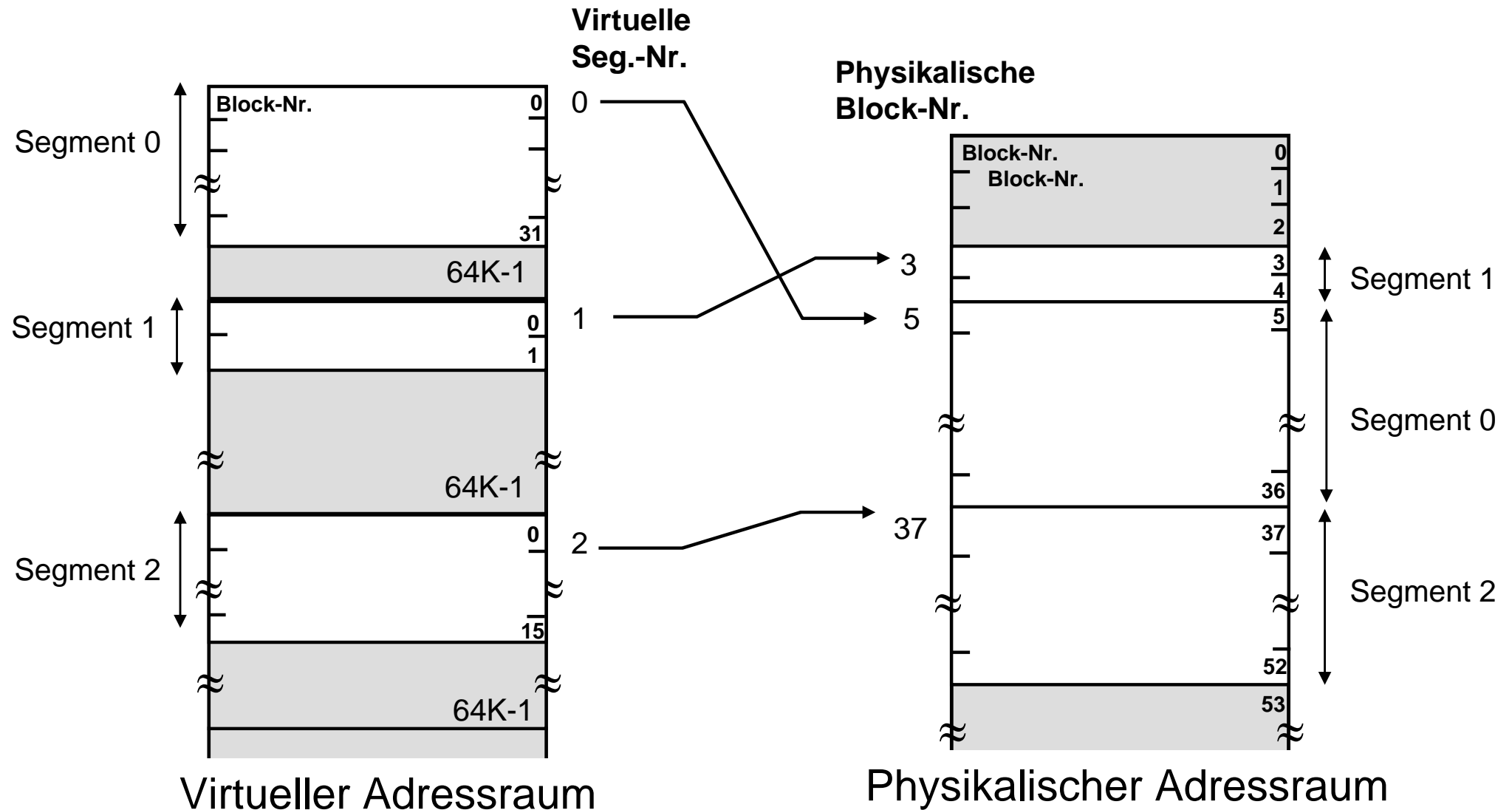
Segmentbasierte Speicherverwaltung



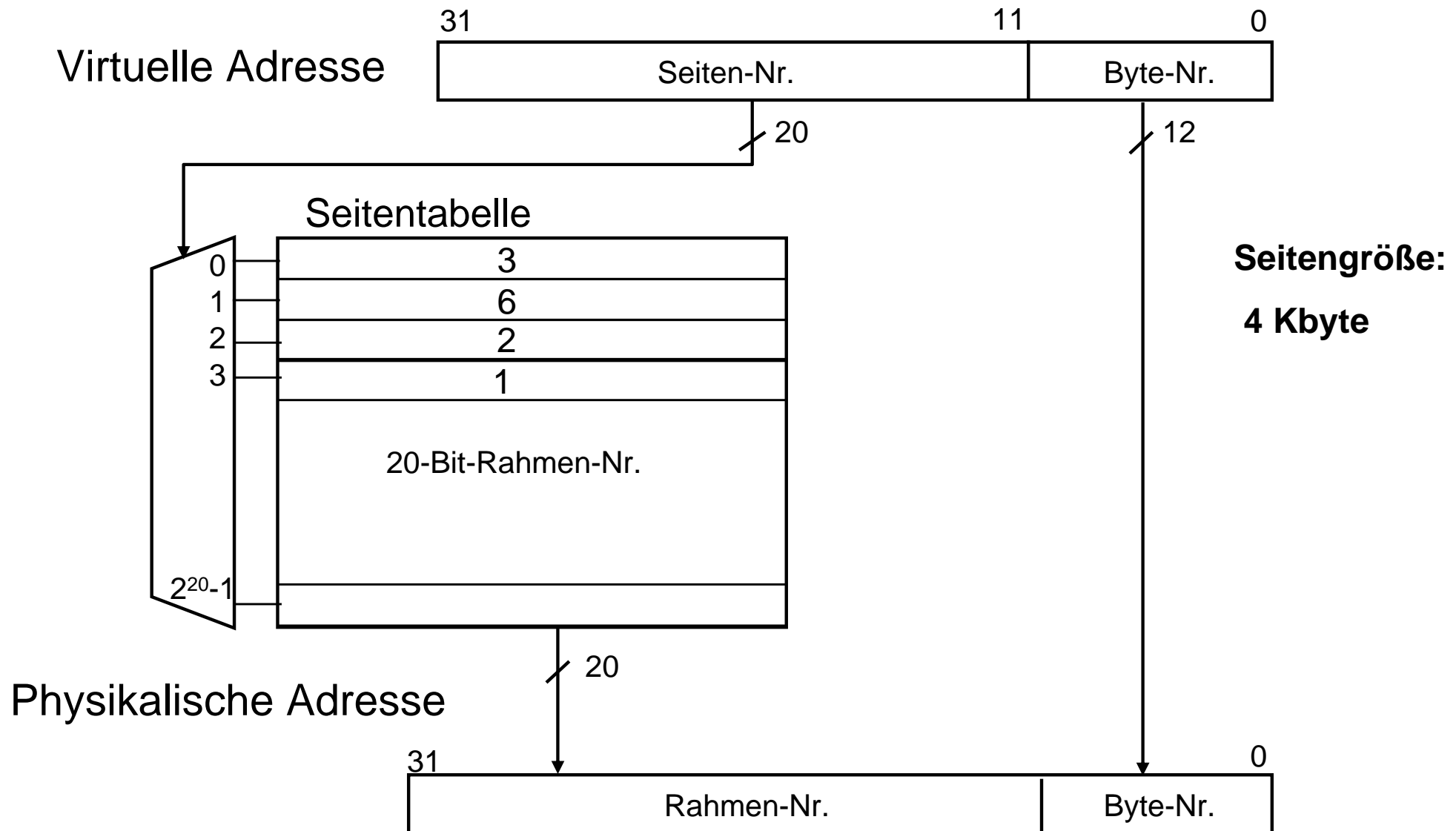
Segmentbasierte Speicherverwaltung

- ❑ Segmentgrenzen nicht an jeder Byteadresse, sondern an Vielfachen von Blöcken (hier von 256 Bytes).
- ❑ Segmente werden im virtuellen physikalischen Adressraum in Blöcke von 256 Bytes unterteilt.
- ❑ D.h. die Bytenummer aus m Bits wird aufgeteilt in eine kürzere Bytenummer für die Byteadressierung im Block und eine Blocknummer.
- ❑ Bei Adressumsetzung wird die virtuelle Segmentnummer auf eine reale 24-Bit-Blocknummer als Segmentbasis abgebildet.
- ❑ Virtuelle Bytenummer für die Adressierung innerhalb des Blocks wird unverändert übernommen.

Virtueller und physikalischer Adressraum



Seitenwechsel (Paging)



Virtueller und physikalischer Adressraum

