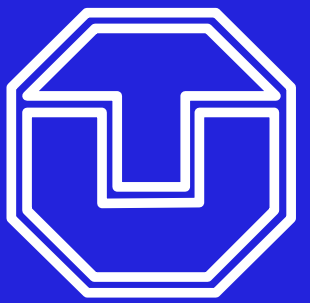
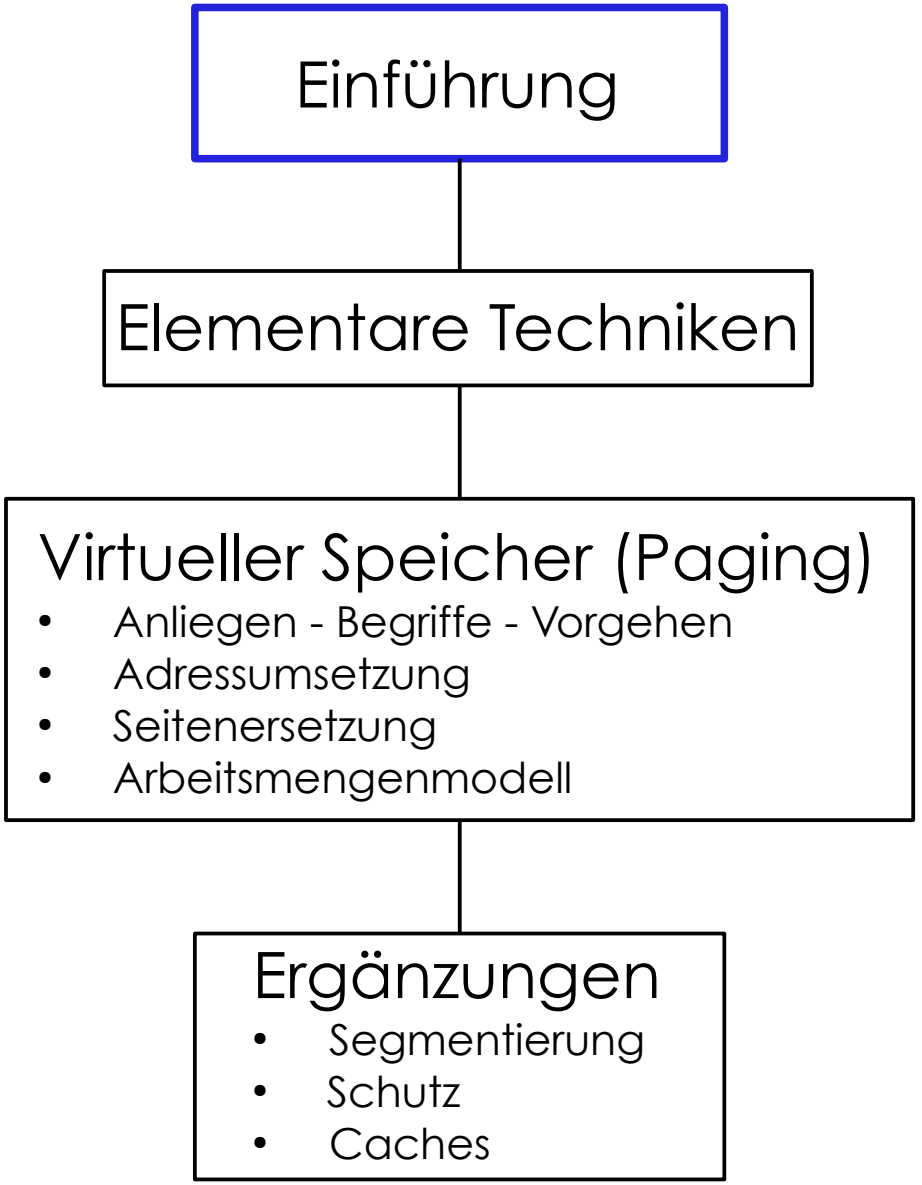
Speicher

Betriebssysteme
(zu großen Teilen nach Tanenbaum)

Hermann Härtig
TU Dresden



Wegweiser


Einführung
Elementare Techniken
Virtueller Speicher (Paging)
• Anliegen - Begriffe - Vorgehen
• Adressumsetzung
• Seitenersetzung
• Arbeitsmengenmodell
Ergänzungen
• Segmentierung
• Schutz
• Caches

Einleitung

Probleme

- Speicherhierarchie
- Programmgröße
- Parallelität von Prozessen

Historie

1950-er: 1 Prozess, Speicher fest zugeordnet

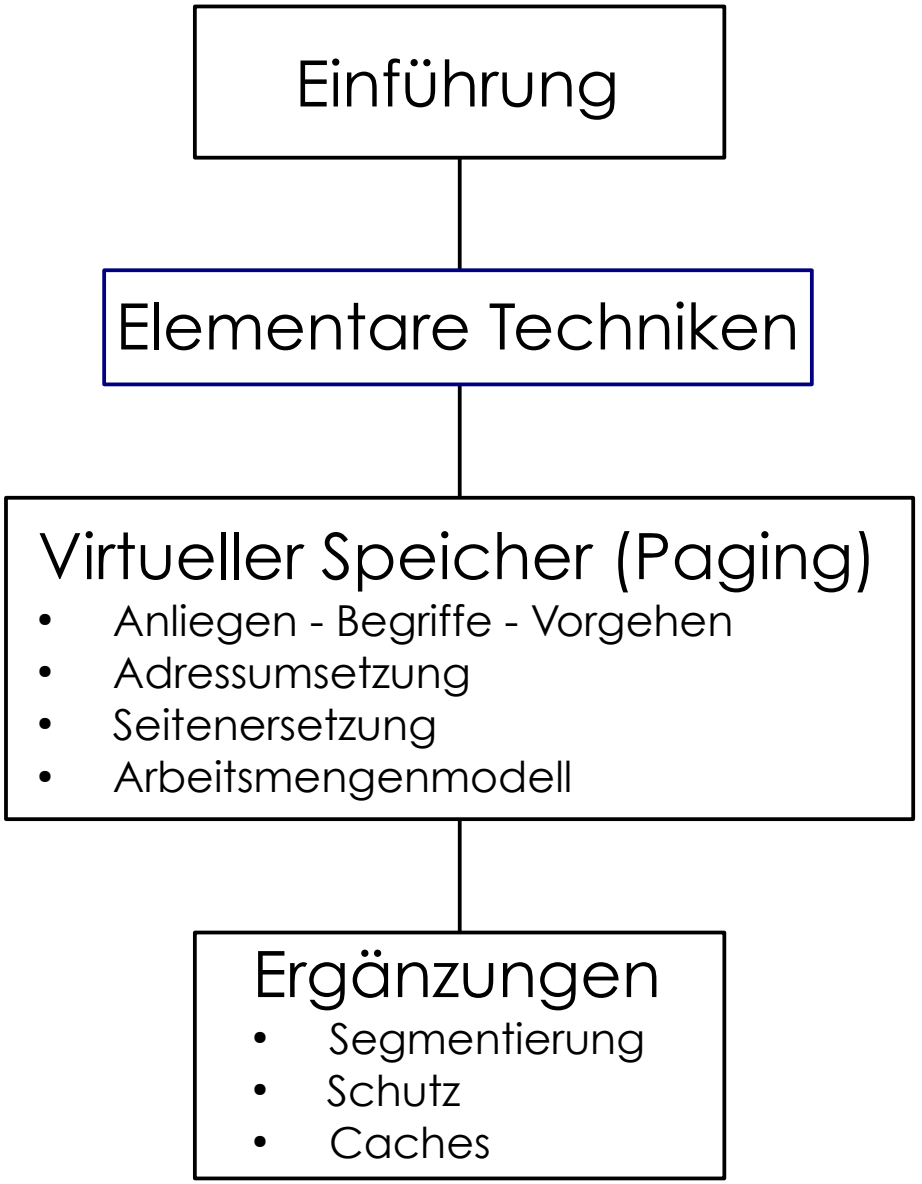
1965-er: *n* Prozesse, *n* disjunkte Speicherbereiche, statisch

1971: *m* Prozesse, *n* Speicherbereiche, dynamisch

Aufgaben der Speicherverwaltung

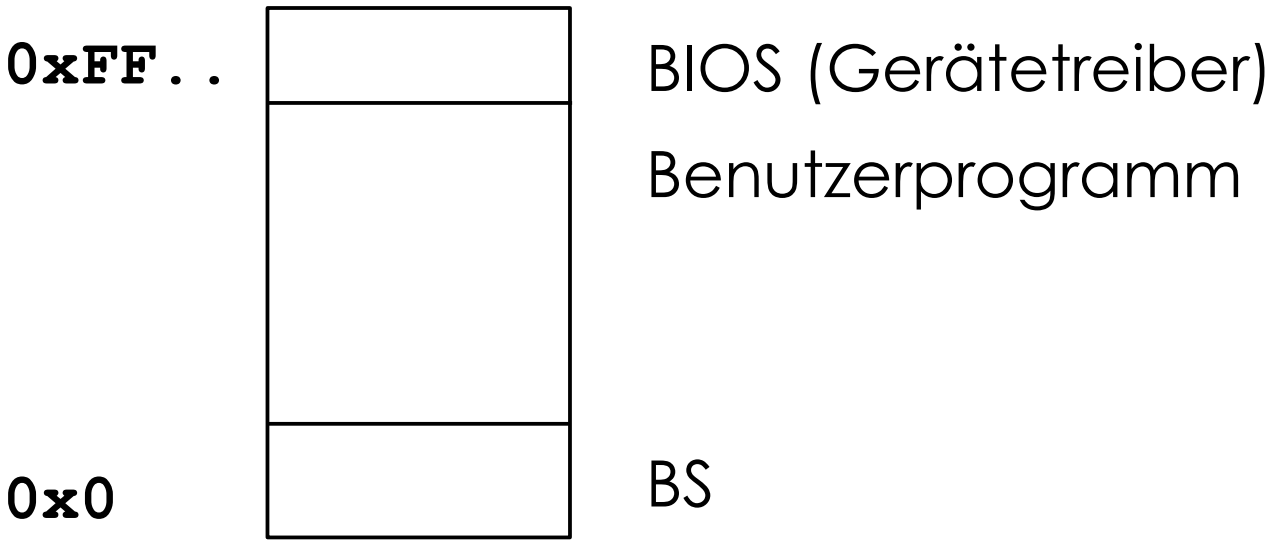
- Bereitstellung von Adressräumen
- Aufbau von Adressräumen durch Zuordnung logischer Objekte
- Verwaltung des Betriebsmittels Hauptspeicher
- Schutz vor unerlaubten Zugriffen
- Organisation gemeinsamer Nutzung („Sharing“) physischen Speichers und logischer Objekte

Wegweiser


Einführung
Elementare Techniken
Virtueller Speicher (Paging)
• Anliegen - Begriffe - Vorgehen
• Adressumsetzung
• Seitenersetzung
• Arbeitsmengenmodell
Ergänzungen
• Segmentierung
• Schutz
• Caches

Statisch

- Statische Speicherverwaltung
  ➔ d. h. keine Ein-/Auslagerung von Programmen/Daten
  - Einfachrechner (MS-DOS Version ??)
  - Gerätesteuerungen (embedded systems)
- Monoprogramming (ein Programm gleichzeitig)
- Programme werden nacheinander geladen und ausgeführt



„Multiprogramming“

Mehrere Benutzer-Programme gleichzeitig im Rechner; für jedes Benutzerprogramm gibt es einen oder mehrere Prozesse

Motivation (vgl. Prozesse):

- mehrere Benutzer eines Rechners (multiuser)
- mehrere Benutzerprozesse eines Benutzers
- Benutzerprozesse und Systemprozesse

Multiprogramming vs. parallele Threads/Prozesse:

- parallele Prozesse Voraussetzung für Multiprogramming (mit oder ohne erzwungenen Prozesswechel)
- denkbar ist Monoprogramming mit vielen parallelen Prozessen

  z. B.: die ersten BS für Parallelrechner erlaubten nur ein Benutzerprogramm zur gleichen Zeit, das aber aus vielen Prozessen/Threads bestehen konnte

Feste oder Variable Speicher-Partitionierung

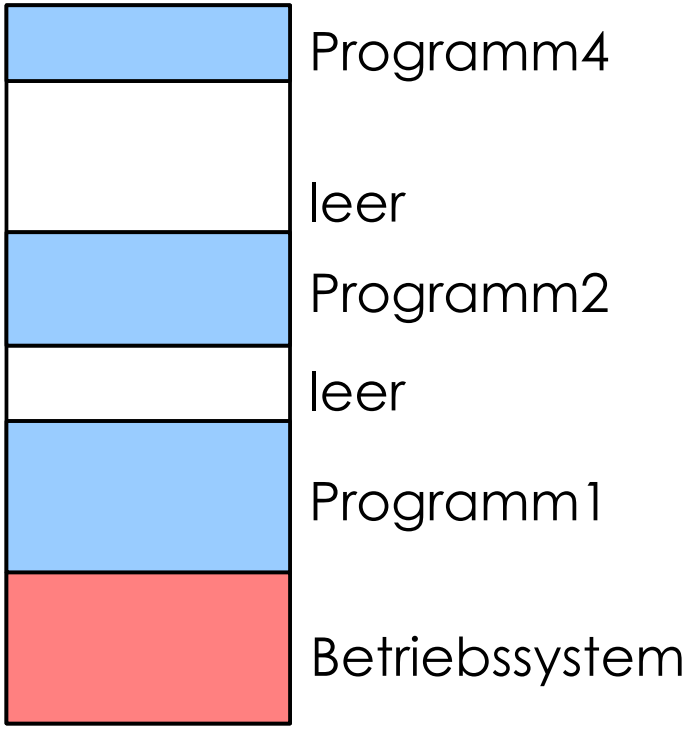
Fragestellungen

- Relokation

  Programme verwenden unterschiedliche Adressen, wenn sie in unterschiedlichen Partitionen ablaufen

  - ➔ Abhilfe: Umsetzen der Adressen
    - beim/vor dem Laden (Software)
    - zur Laufzeit (Hardware)

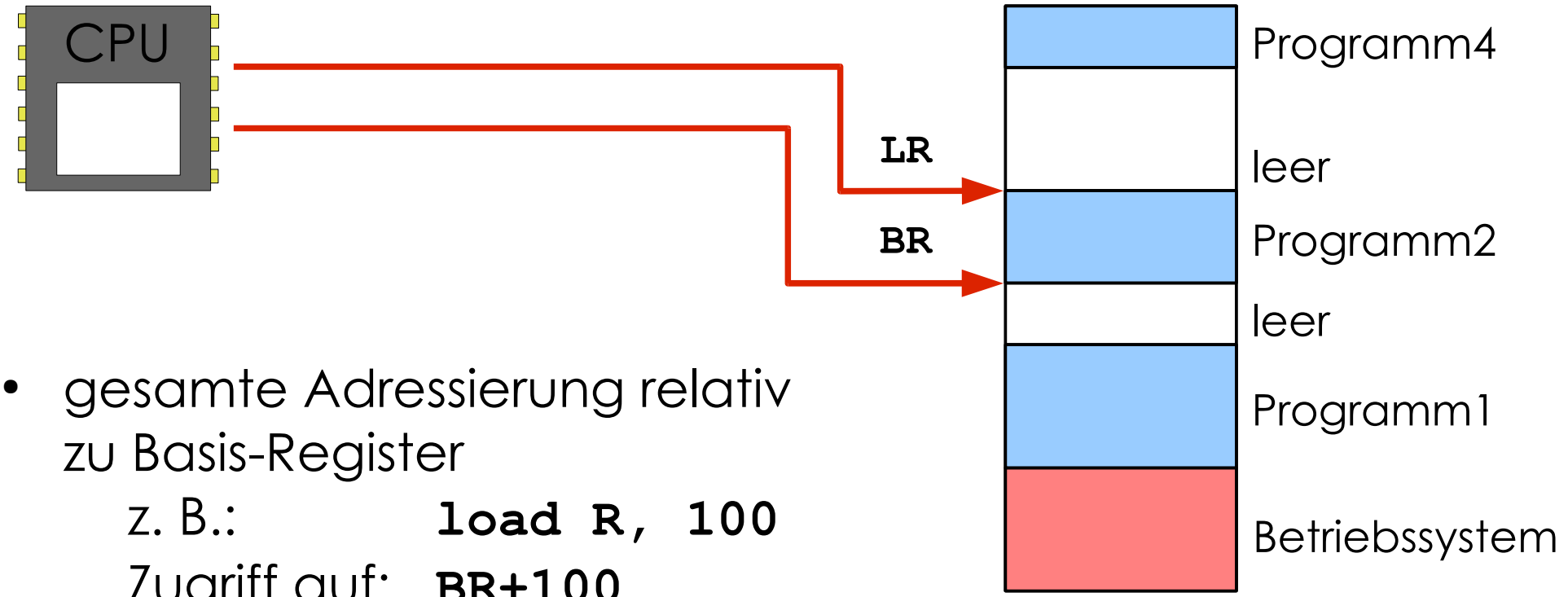
- Schutz der Partitionen voreinander
  - ➔ Abhilfe:
    - Überprüfung der Adressen zur Laufzeit (Hardware)



Limitationen

- Menge und Größe der Programme durch Real-Speicher
- Auslastung

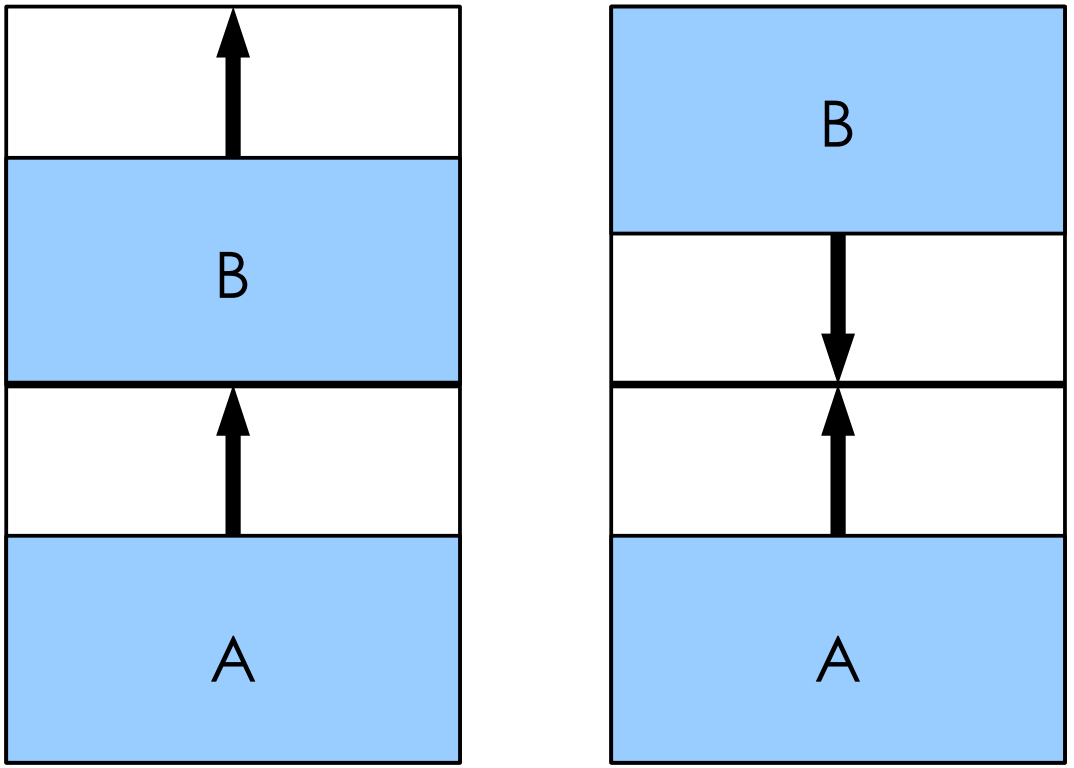
Einfaches Modell: Basis- und Limit-Register



- gesamte Adressierung relativ zu Basis-Register
  z. B.: **`load R, 100`**
  Zugriff auf: **`BR+100`**
- Unterbindung aller Zugriffe auf Bereiche außerhalb [**`BR,LR`**]

➔ Konsequenz für Implementierung eines Prozess-Systems:
bei Umschaltung müssen auch BR und LR umgeschaltet werden

Wachsen von Partitionen



Einlagerungsalgorithmen zur „Minimierung“ des Verschnitts

- First fit, Best fit, Buddy, ...

Verwaltung

- Bitmaps, Listen

Swapping: Ein-/Auslagern ganzer Prozesse

Vorgehen

Partitionen von blockierten Prozessen werden auf persistenten Speicher ausgelagert (z. B. auf Platte) und bei Gelegenheit wieder eingelagert.

Mehrebenen-Scheduling

auch bereite Prozesse werden ausgelagert

Fragestellungen

- wachsende Partitionen
- Speicherverschnitt (externe Fragmentierung)

Swapping: Nachteile und Probleme

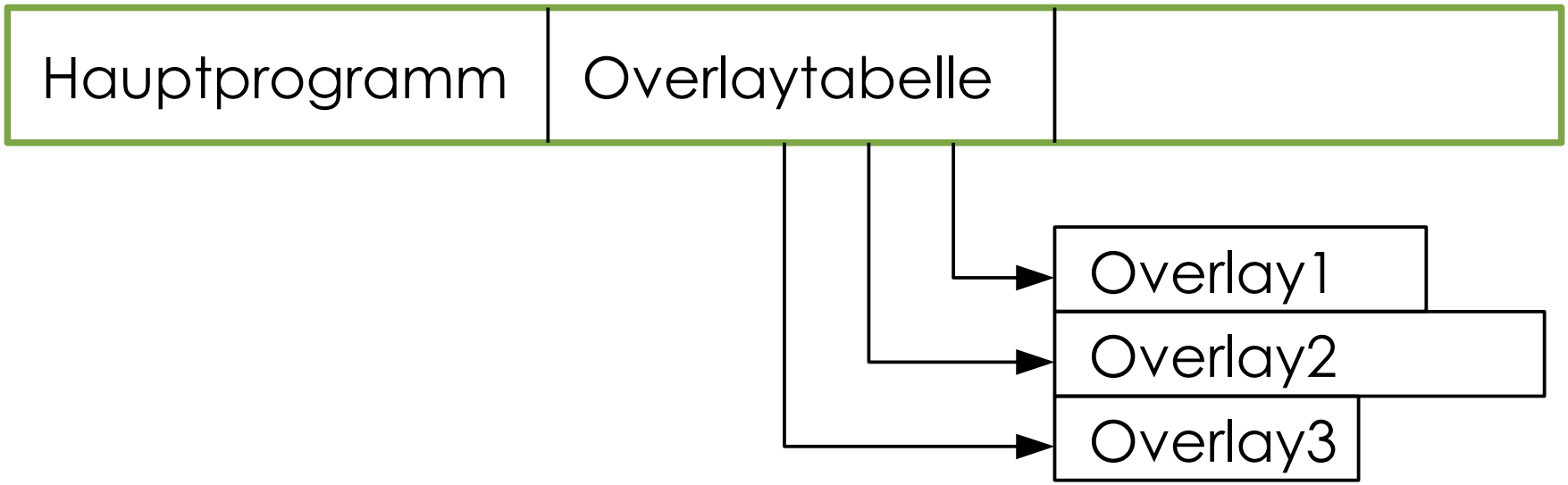
- Verschnitt hoch
  - ➔ quantitative Ermittlung später
- Programmstartzeiten
- Limitation der Größe eines Prozesses durch verfügbaren Hauptspeicher
- Ein-/Auslagerungszeit
  „ruhende“ Teile
- Platzbedarf auf Externspeicher

Overlays – Überlagerungstechnik

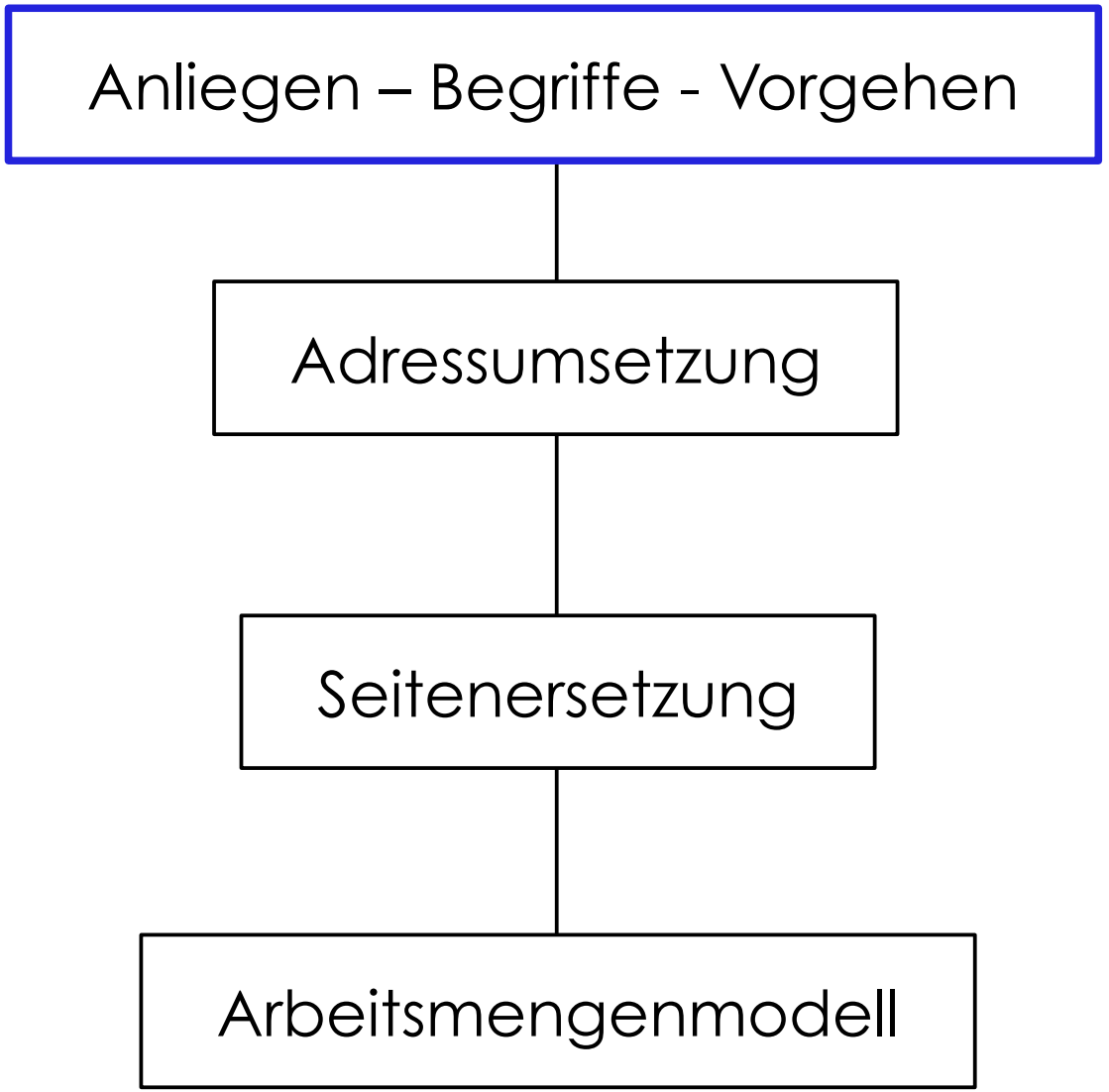
Beliebig große Programme → „Overlays“

Programmierer organisiert seine Programme und Daten in Stücken, von denen nicht zwei gleichzeitig im Hauptspeicher sein müssen

Hauptspeicher



Wegweiser: Virtueller Speicher



Adressraum

Begriff allgemein

- Menge direkt zugreifbarer Adressen und deren Inhalte
- Größe bestimmt durch Rechner-Architektur

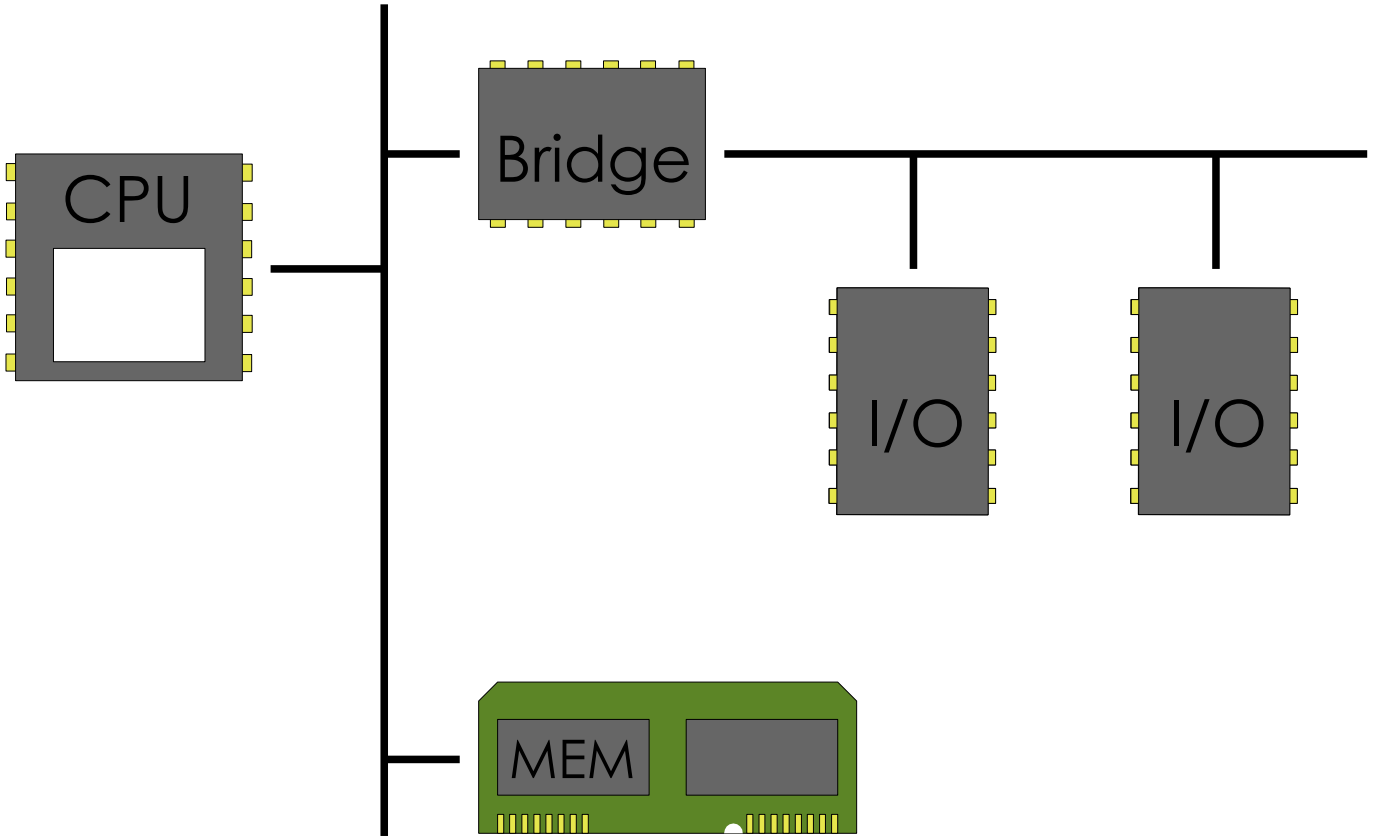
Physischer Adressraum

- durch Adressleitungen gebildeter AR, z. B. am Speicher oder Peripheriebus eines Rechners
- Abbildung der Prozessor-Adressen auf die vorhandenen Speicherbausteine und E/A-Controller
- Adressumsetzung statisch durch HW-Adressdecoder

Virtueller (logischer) Adressraum eines Prozesses

- dem Prozess zugeordneter Adressraum
- Adressumsetzung durch MMU, veränderliche Abbildungsvorschrift

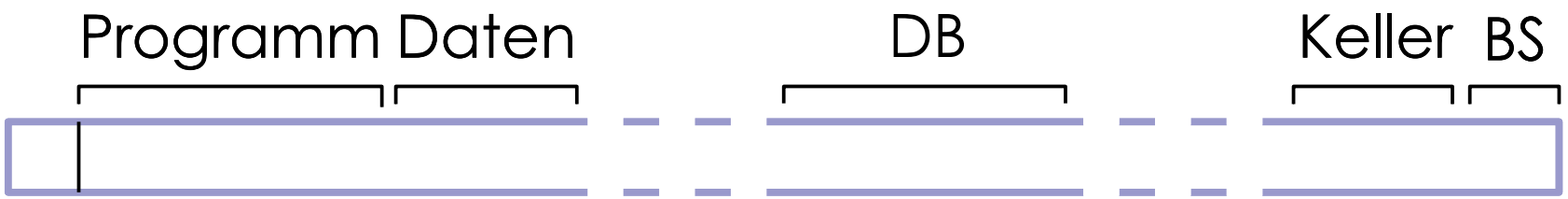
Physischer Adressraum



Beispiele für die Nutzung virtueller Adressräume

Unix-Prozesse (konventionell)

Moderne Datenbank-Implementierung



- Logisch zusammenhängende Adressbereiche nennt man **Regionen**
- **Speicherobjekte** werden Regionen zugeordnet („Mapping“)

Virtueller Speicher

Forderungen an Adressräume und ihre Implementierung

- groß (soweit die Hardware zuläßt, z. B. jeder bis zu 4 GB auf Pentium)
- frei teilbar und nutzbar
- Fehlermeldung bei Zugriff auf nicht belegte Bereiche
- Schutz vor Zugriffen auf andere Adressräume
- Einschränken der Zugriffsrechte auf bestimmte Bereiche (z. B. Code nur lesen)

- sinnvoller Einsatz des (Haupt-)Speichers
  - damit Speicher anderweitig nutzbar
  - wegen kurzer Ladezeiten
  - ohne großen Aufwand für Programmierer

Prinzipien der virtuellen Speichers

Idee

Zuordnen von Speicherobjekten (z. B. Segmente, Dateien, Datenbanken, Bildwiederholspeicher) bzw. Ausschnitten davon zu Regionen von Adressräumen

Basis: Partitionierung

- des Adressraums in Seiten (Pages)
- der Hauptspeichers in Kacheln
  auch Rahmen genannt (Page Frames)
- des Hintergrundspeichers in Blöcke (Blocks)

in Stücke gleicher Größe

→ Organisation der Zuordnung der Stücke zueinander durch Hardware und Betriebssystem

Voraussetzung: Lokalitätsprinzip

Beobachtung

Der von einem Prozess innerhalb eines bestimmten Zeitintervalls benötigte Teil seines Adressraumes verändert sich nur mehr oder weniger langsam.

Ursachen

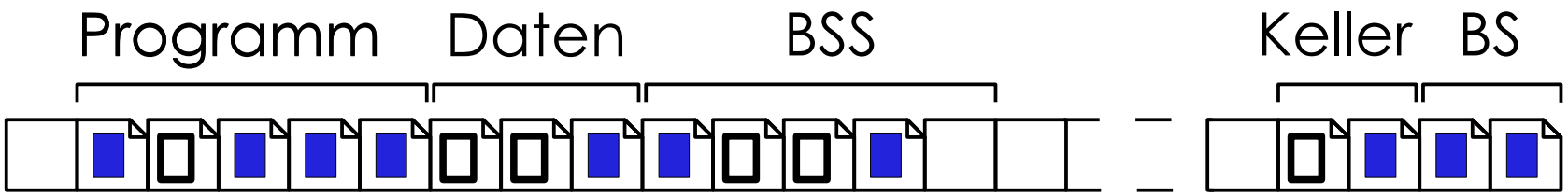
- sequentielle Arbeit eines VON-NEUMANN-Rechners
- Programmcode enthält Zyklen
- Programmierung in Modulen
- Zugriff auf gruppierte Daten

Virtueller Speicher – Begriff

Der virtuelle Speicher ist eine Technik, die jedem Prozess einen eigenen, vom physischen Hauptspeicher unabhängigen logischen Adressraum bereitstellt, basierend auf

- der Nutzung eines externen Speichermediums
- einer Partitionierung von Adressräumen in Einheiten einheitlicher Größe
- einer Adressumsetzung durch Hardware (und Betriebssystem)
- einer Ein- und Auslagerung von Teilen des logischen Adressraumes eines Prozesses durch Betriebssystem (und Hardware).

Eine denkbare Situation



unbenutzt, ungültig

gerade im Hauptspeicher

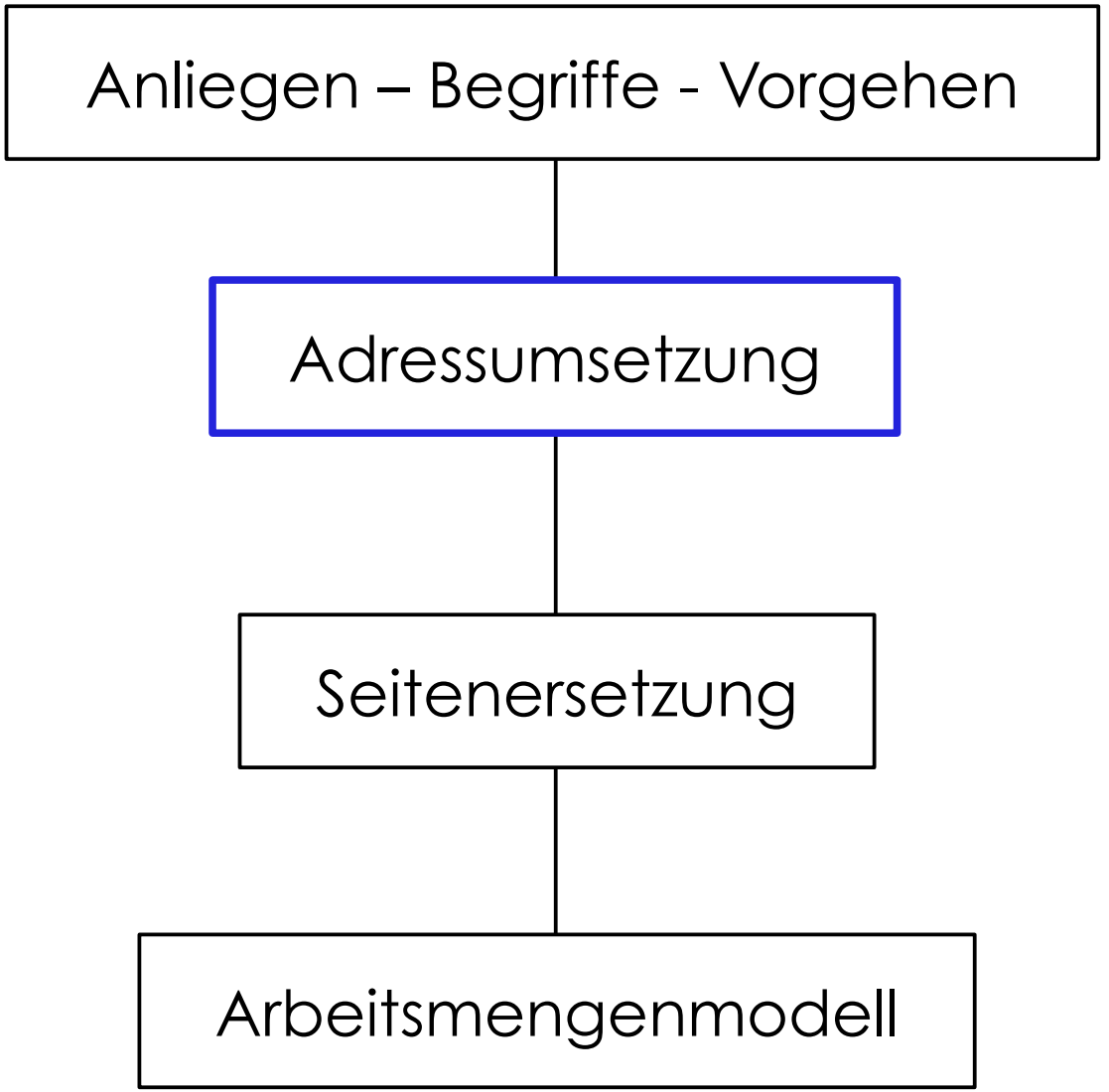
gerade nicht im Hauptspeicher, aber ein gültiger Bereich – z. B. ausgelagert auf Platte

Virtueller Speicher im Betriebssystem

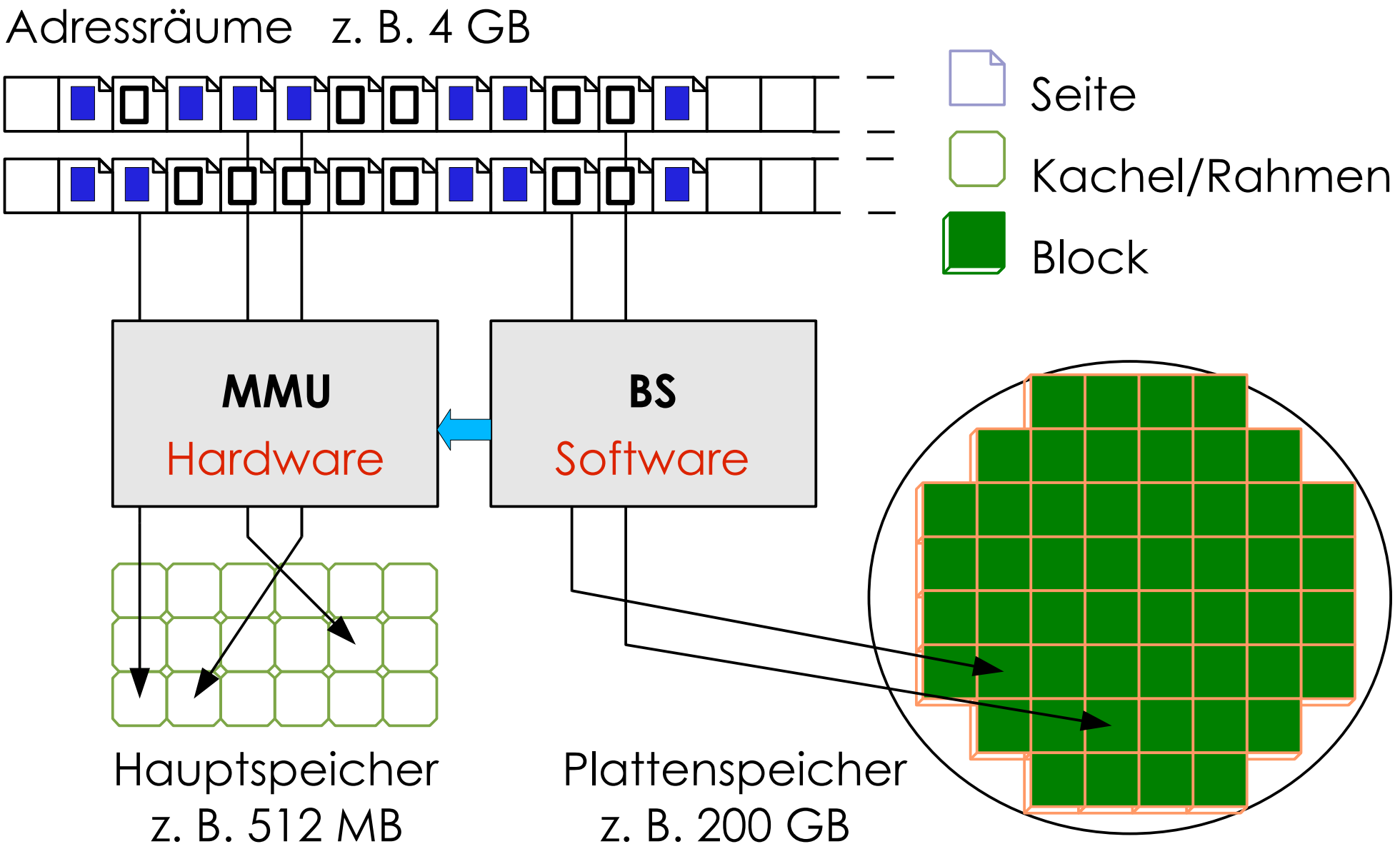
Teilaufgaben

- Seitenfehler-Behandlung
- Verwaltung des Betriebsmittels Hauptspeicher
- Aufbau der Adressraumstruktur
  (Speicherobjekte und Regionen)
- Bereitstellung spezifischer Speicherobjekte
- Interaktion Prozess- und Speicher-Verwaltung

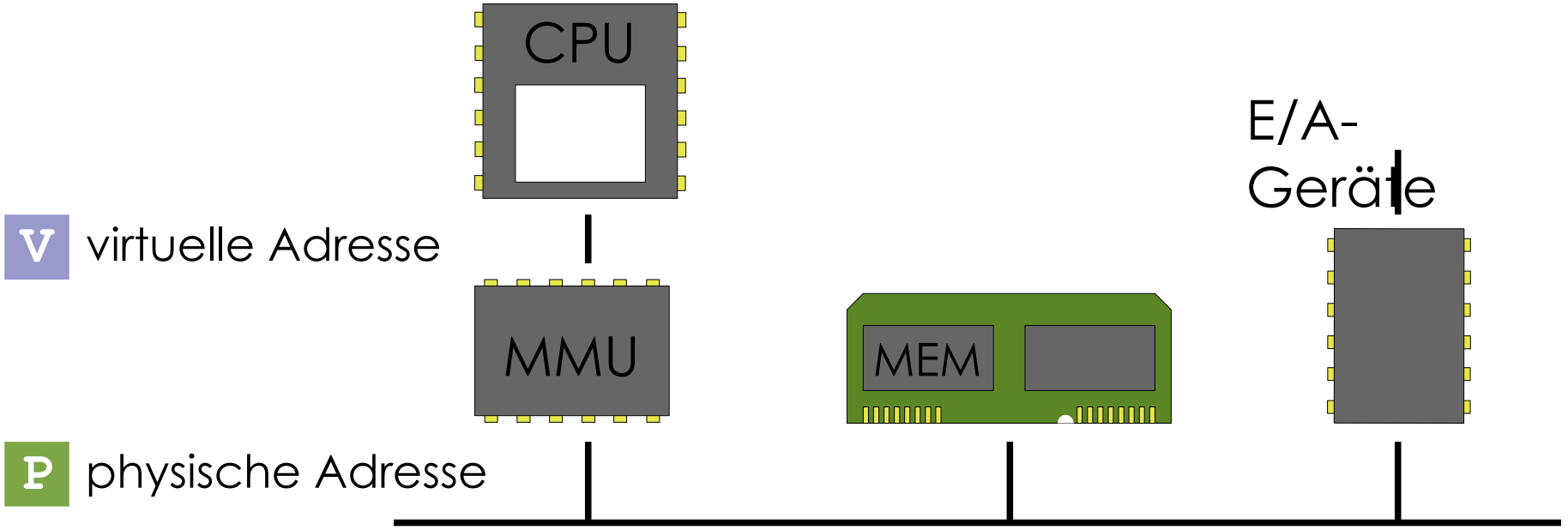
Wegweiser: Virtueller Speicher



Seiten, Kacheln, Blöcke



Rechnerarchitektur: Addressumsetzung etc.



Aufgaben einer MMU (Memory Management Unit)

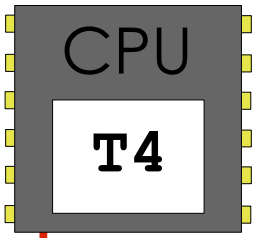
- Abbildung: virtuelle → reale (physische) Adresse
- Schutz bestimmter Bereiche (lesen/schreiben)
- Betriebssystemaufruf bei abwesenden/geschützten Seiten
  → Seitenfehler (page fault)
- Schutz der Adressräume untereinander

Prinzipielle Arbeitsweise einer MMU

virtuelle Adresse

physische Adresse

Prinzipielle Arbeitsweise einer MMU



virtuelle Adresse

Seiten# Offset

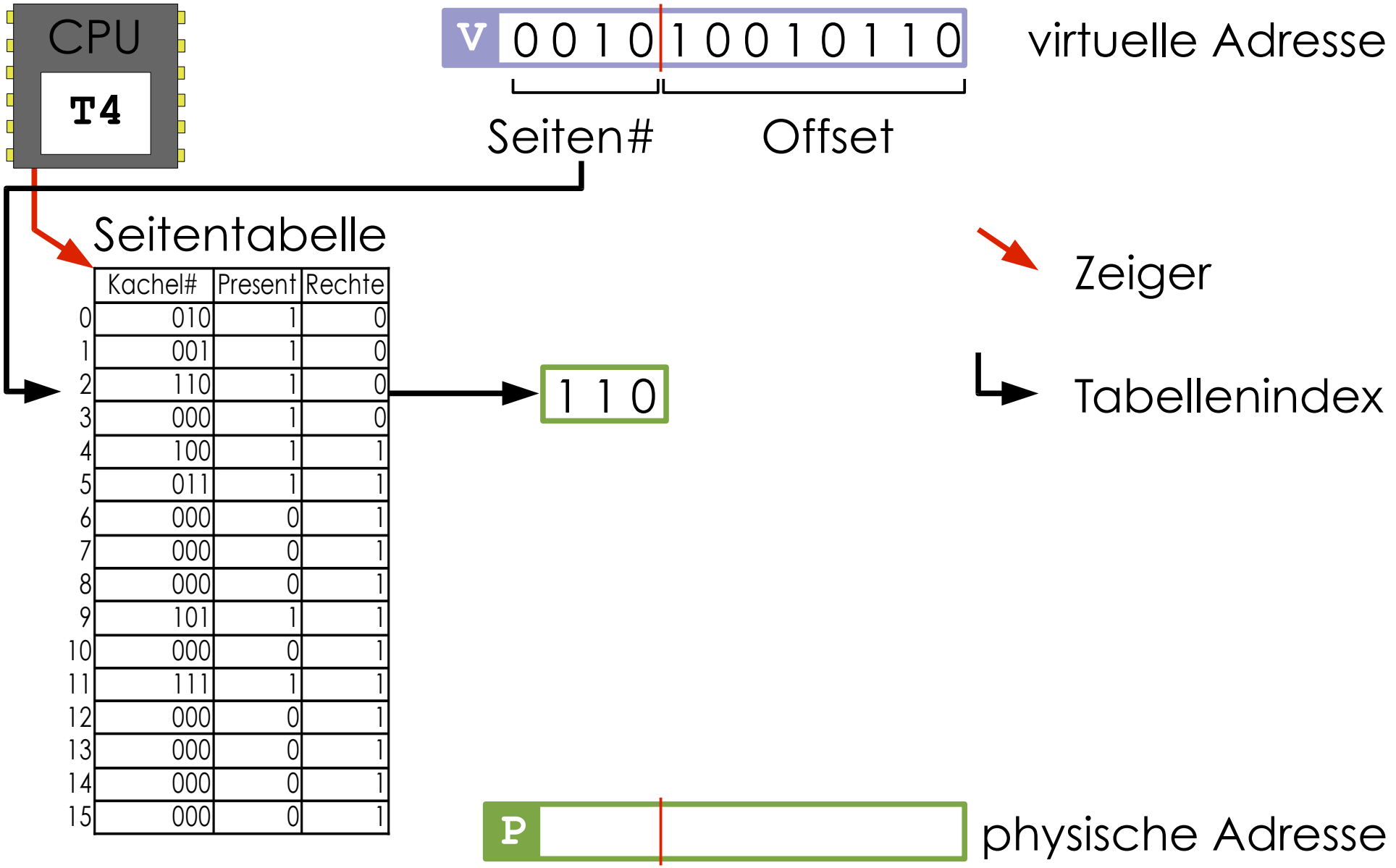
Seitentabelle

	Kachel#	Present	Rechte
0	010	1	0
1	001	1	0
2	110	1	0
3	000	1	0
4	100	1	1
5	011	1	1
6	000	0	1
7	000	0	1
8	000	0	1
9	101	1	1
10	000	0	1
11	111	1	1
12	000	0	1
13	000	0	1
14	000	0	1
15	000	0	1

Zeiger

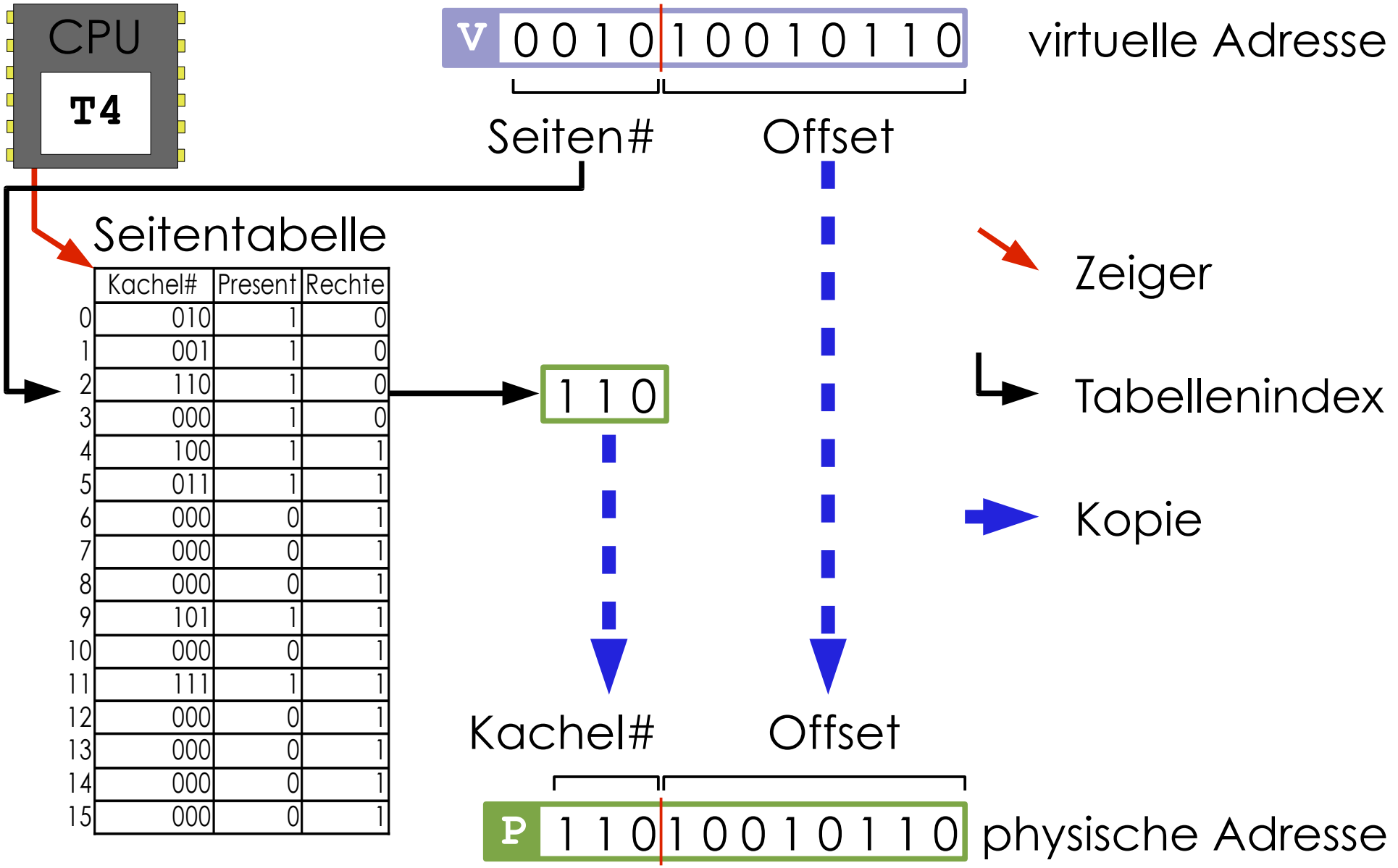
physische Adresse

Prinzipielle Arbeitsweise einer MMU

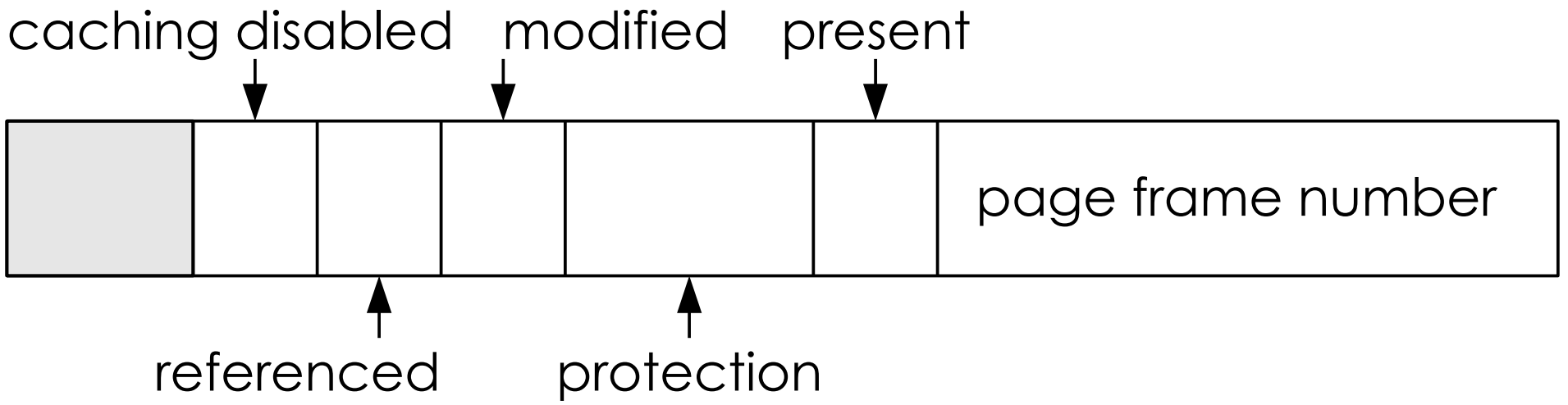


	Kachel#	Present	Rechte
0	010	1	0
1	001	1	0
2	110	1	0
3	000	1	0
4	100	1	1
5	011	1	1
6	000	0	1
7	000	0	1
8	000	0	1
9	101	1	1
10	000	0	1
11	111	1	1
12	000	0	1
13	000	0	1
14	000	0	1
15	000	0	1

Prinzipielle Arbeitsweise einer MMU



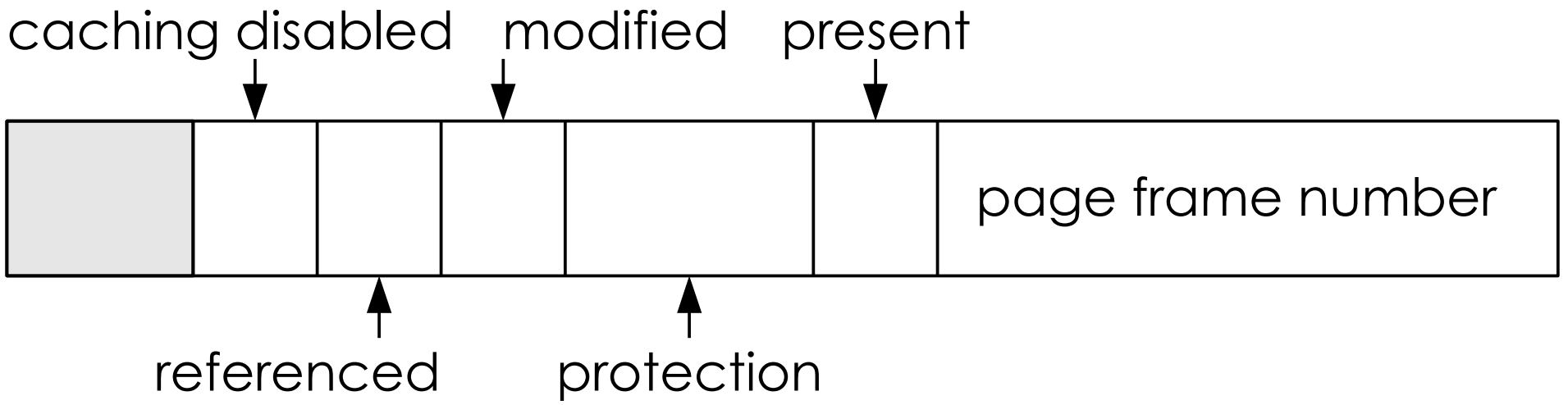
Prinzipieller Aufbau eines Seitentabelleneintrags



Seiten-Attribute

- present — Seite befindet sich im Hauptspeicher
- modified — schreibender Zugriff ist erfolgt („dirty“)
- used — irgendein Zugriff ist erfolgt
- caching — ein/aus (z. B. wegen E/A)
- protection — erlaubte Art von Zugriffen in Abhängigkeit von CPU-Modus

Prinzipieller Aufbau eines Seitentabelleneintrags



protection:

mode \ operation	read	write	execute
kernel			
user			

Virtueller Speicher: Hardware-Anteil

- Bei jedem Speicherzugriff:
  Überprüfung von Präsenz und Rechten

Ablauf eines Seitenfehlers (exception):

- Zurücksetzen des auslösenden Befehls
- Umschaltung des Prozessormodus und des Kellers
- Ablegen einer Beschreibung des Zustandes, der auslösenden Adresse und der Zugriffsart auf dem Keller
- Sprung in den Kern
  → Seitenfehlerbehandlung durch Betriebssystem
- **`iret`** (letzte Instruktion des Handlers)

Beispiel

Beispiel

```
LDS SI, [Adr. 1]
LES DI, [Adr. 2]
MOV CX, Length
REP MOVSB
```

- bei Seitenfehler wird Zwischenzustand per HW konsistent gehalten
- bei Rückkehr wird alles wiederhergestellt und Befehl wiederholt

MMU-Probleme: Größe und Geschwindigkeit

Problem 1: Größe – ein Beispiel

Realspeicher : 256 MB

virtuelle Adressen: 32 Bit

Seitengröße: 4 KB

Aufteilung virt. Adr.: 20 Bit Index in der Seitentabelle
12 Bit innerhalb einer Seite (Offset)

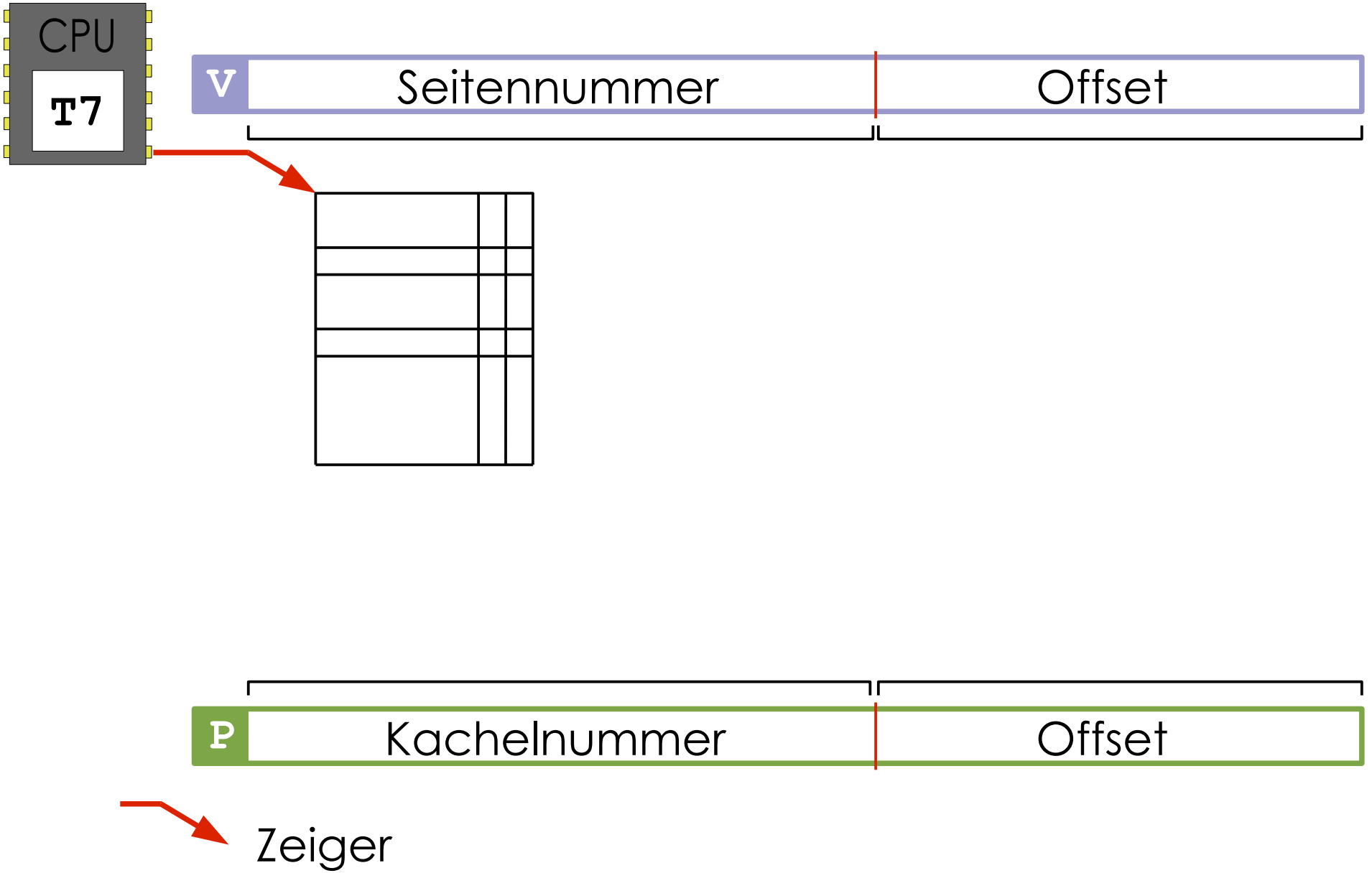
- Größe der Seitentabelle für einen Adreßraum:
  $2^{20} * 4B \rightarrow 4$ MB
- Bei 32 Prozessen 4*32 MB für die Seitentabellen !

Problem 2: Geschwindigkeit der Abbildung – ein Beispiel

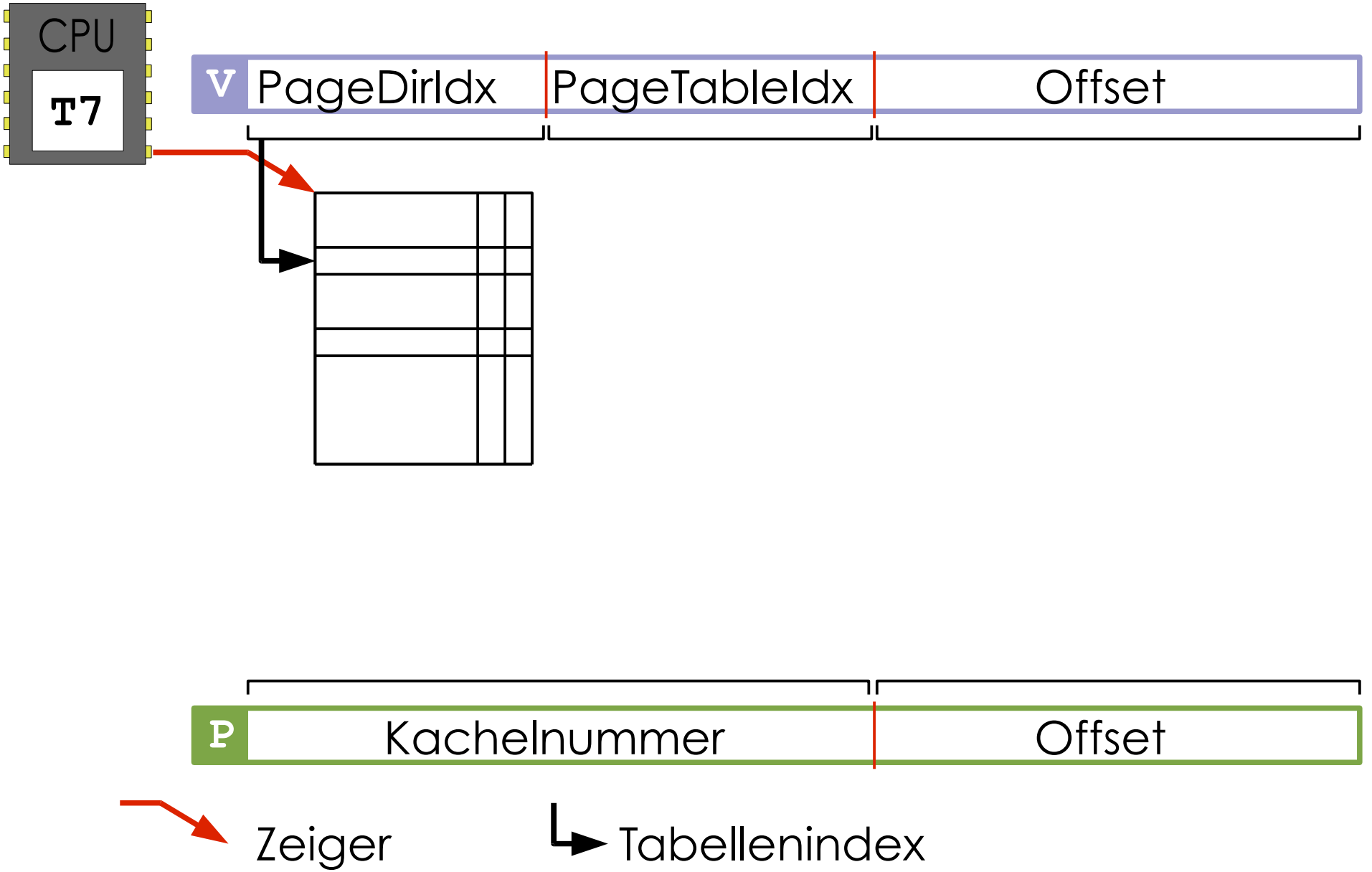
CPU-Takt : 1 GHz → 2 Instruktionen in 1ns
(4 Speicherzugriffe)

Speichertakt: 256 MHz → 4ns (... 70ns)

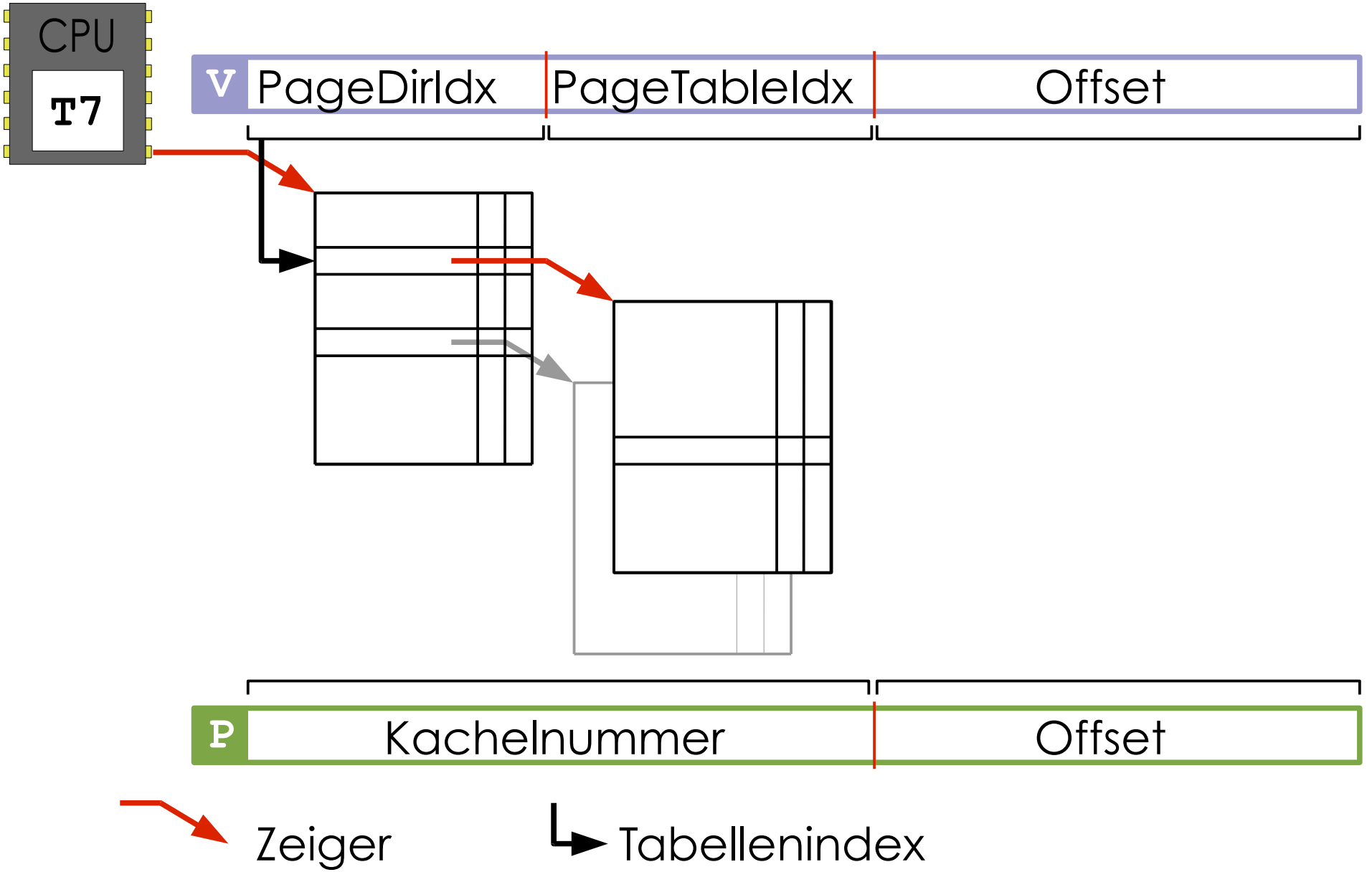
Problem 1: Baumstrukturierte Seitentabellen



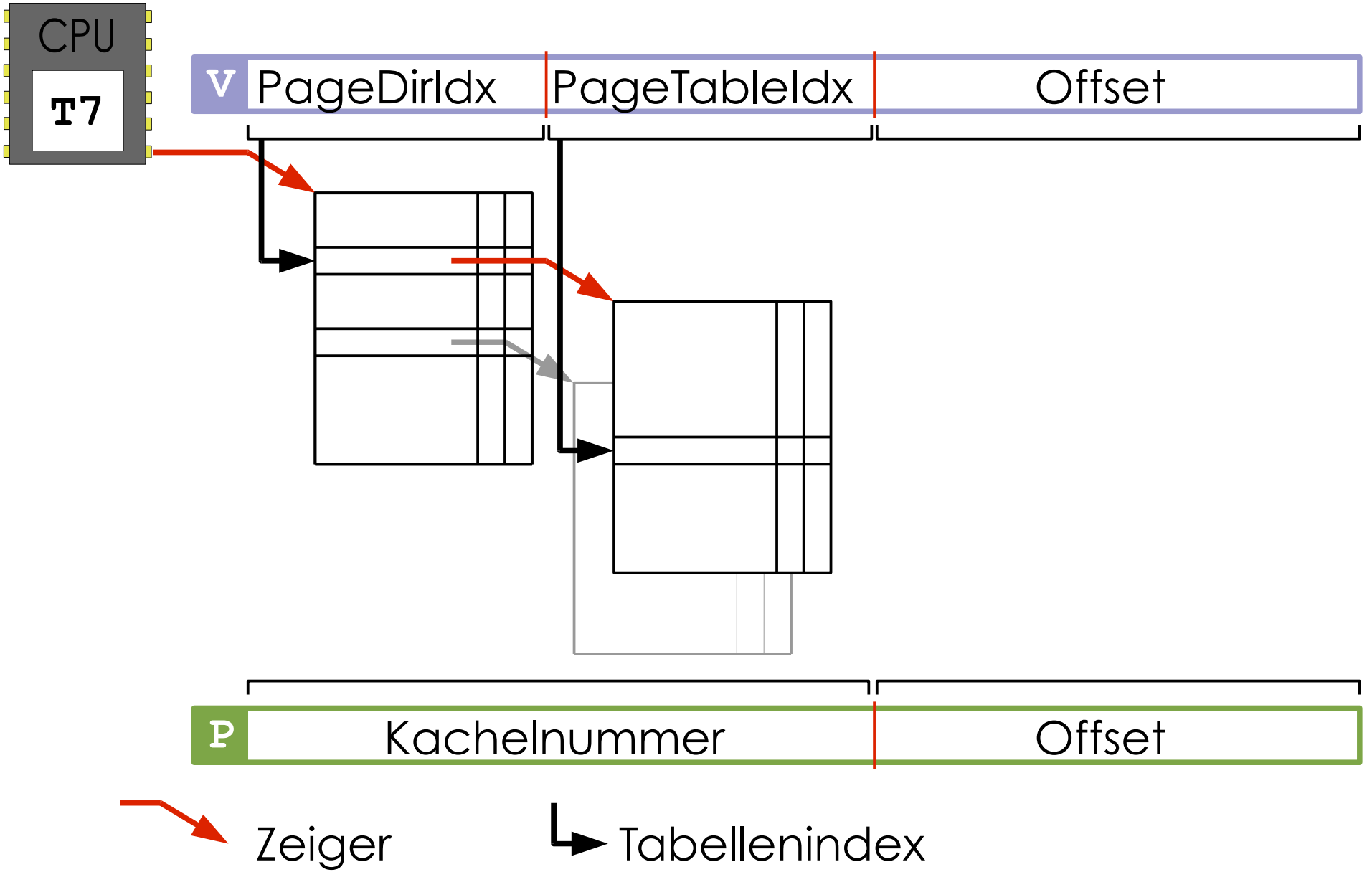
Problem 1: Baumstrukturierte Seitentabellen



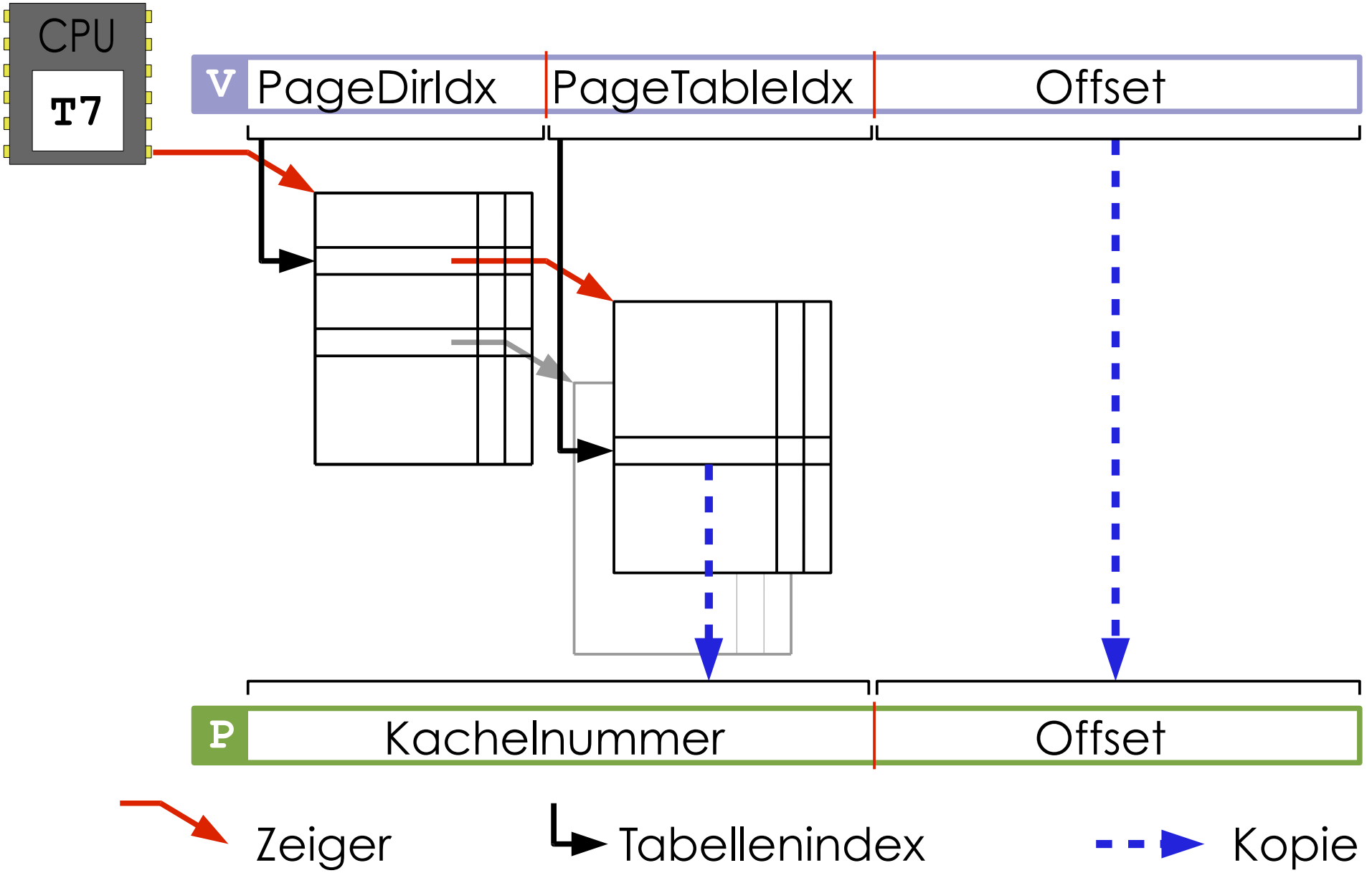
Problem 1: Baumstrukturierte Seitentabellen



Problem 1: Baumstrukturierte Seitentabellen



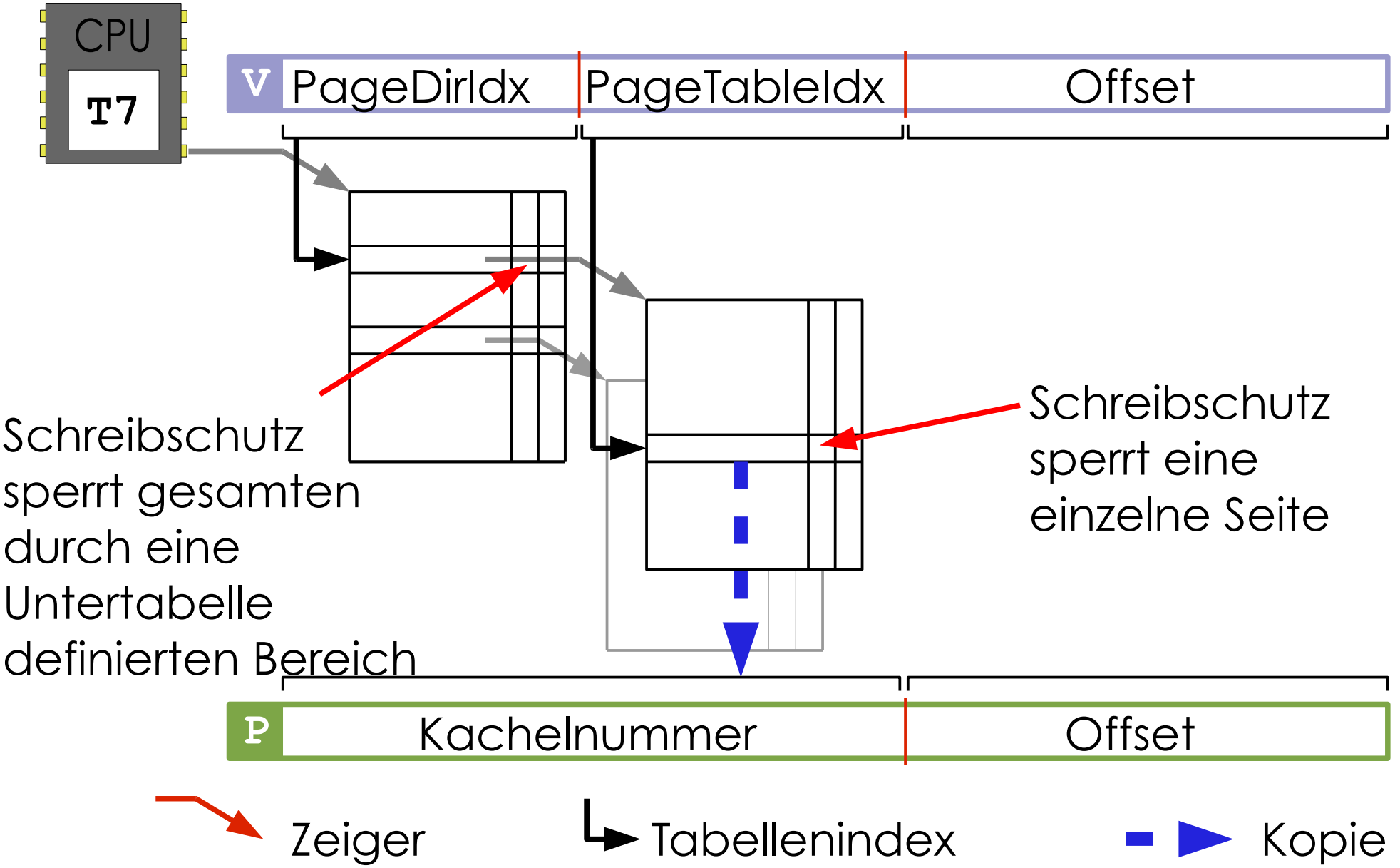
Problem 1: Baumstrukturierte Seitentabellen



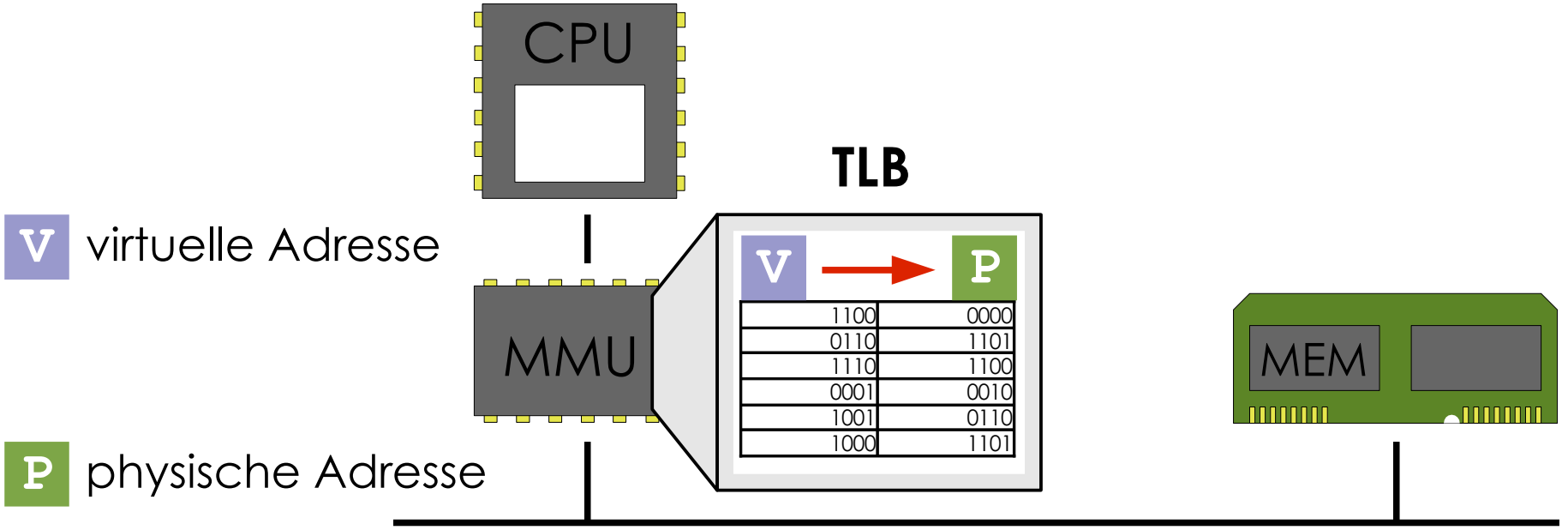
Eigenschaften baumstrukturierter Seitentab.

- beliebig schachtelbar
  → 64-Bit-Adressräume!
- Seitentabellen nur bei Bedarf im Hauptspeicher
  bei Zugriff auf Seitentabelle kann Seitenfehler auftreten
- Zugriff auf Hauptspeicher wird noch langsamer
  2 oder mehr Umsetzungsstufen
- Hierarchiebildung möglich (nächste Folie)
  z. B. durch Schreibsperre in höherstufiger Tabelle ist ganzer Adressbereich gegen Schreiben schützbar
- Gemeinsame Nutzung („Sharing“) → später
  Seiten und größere Bereiche in mehreren Adressräumen gleichzeitig

Hierchiebildung baumstrukturierte Seitentab.



Problem 2: Schnellere Abbildung



Translation Look Aside Buffer (TLB)

- schneller Speicher für schon ermittelte Abbildungen virtueller auf reale Adressen (physische Adressen)
- wird vor Durchsuchen der Seitentabellen inspiziert

Per SW implementierte Seitentabellen

z. B. Alpha

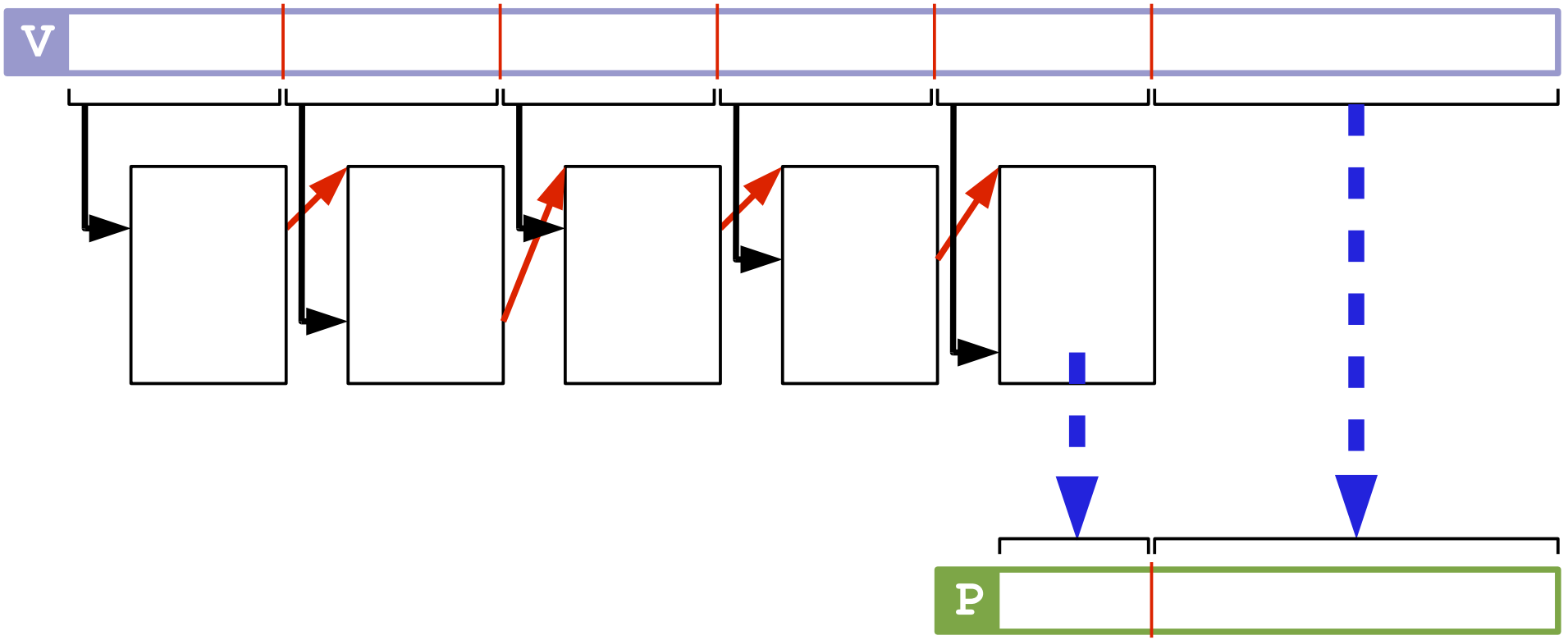
- Zugriff nur über TLB, bei TLB-Fehler („TLB-miss“) → Seitenfehler
- Seitenfehlerbehandlung in SW lädt TLB neu

➔ Vorteil: total flexibel

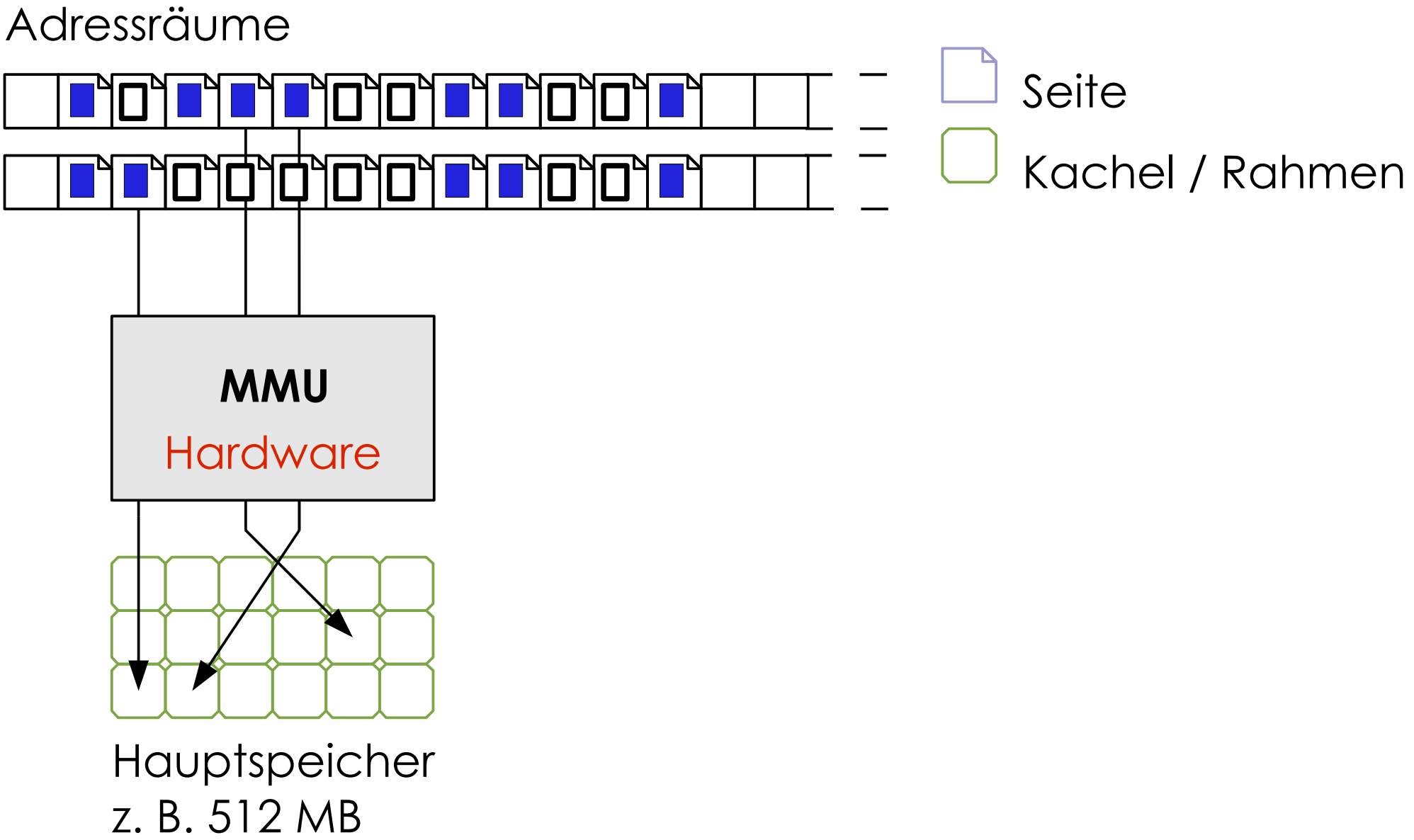
➔ Nachteil: häufigere SW-Seitenfehlerbehandlung

Nochmal Problem 1

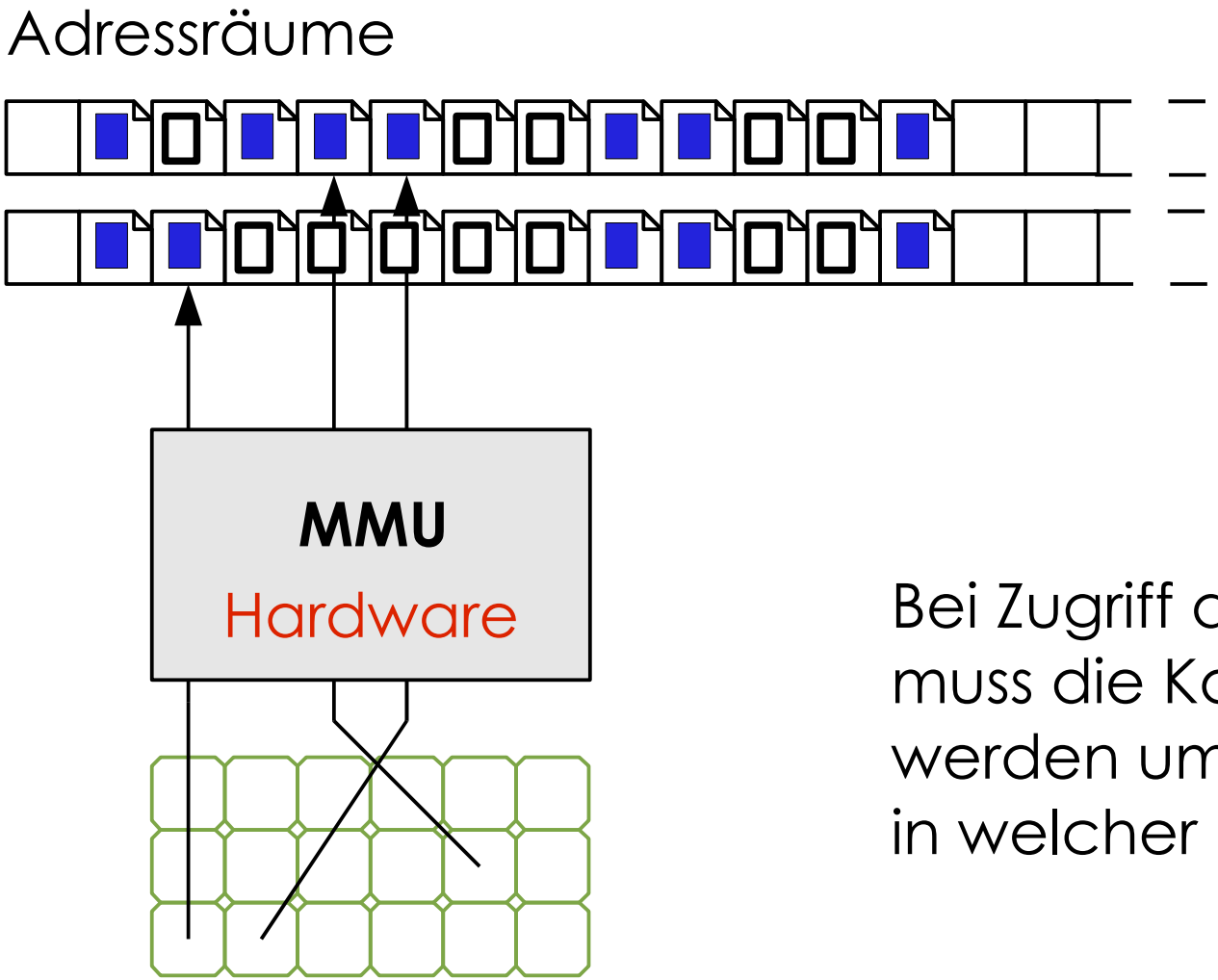
- Problem: riesige Seitentabellen auch bei baumorientierten Seitentabellen
  - bei CPU mit großen Adressräumen (64-Bit-Adressen)
  - bei lose besetzten Adressräumen



Seitentabellen



Invertierte Seitentabellen



Bei Zugriff auf virtuelle Adresse muss die Kacheln durchsucht werden um herauszufinden, in welcher Kachel die Seite steht

Vergleich der zwei Konzepte

Seiten-Kachel-Tabelle

Seiten#

	Rahmen#	Attribs
0	0001	
1	0000	
2	1001	
3	0110	
4	0111	
5	1100	
6	1101	
7	0110	
8	0010	
	. . .	

Kachel-Seiten-Tabelle

Seiten#

	PID	Seiten#	Attribs
0	1	0001	
1	1	0000	
2	1	1000	
3	2	1000	
4	7	0001	
5	-	0000	
6	1	!	
7	1	0100	
8	2	0010	
	. . .		

Invertierte Seitentabelle

Grundidee

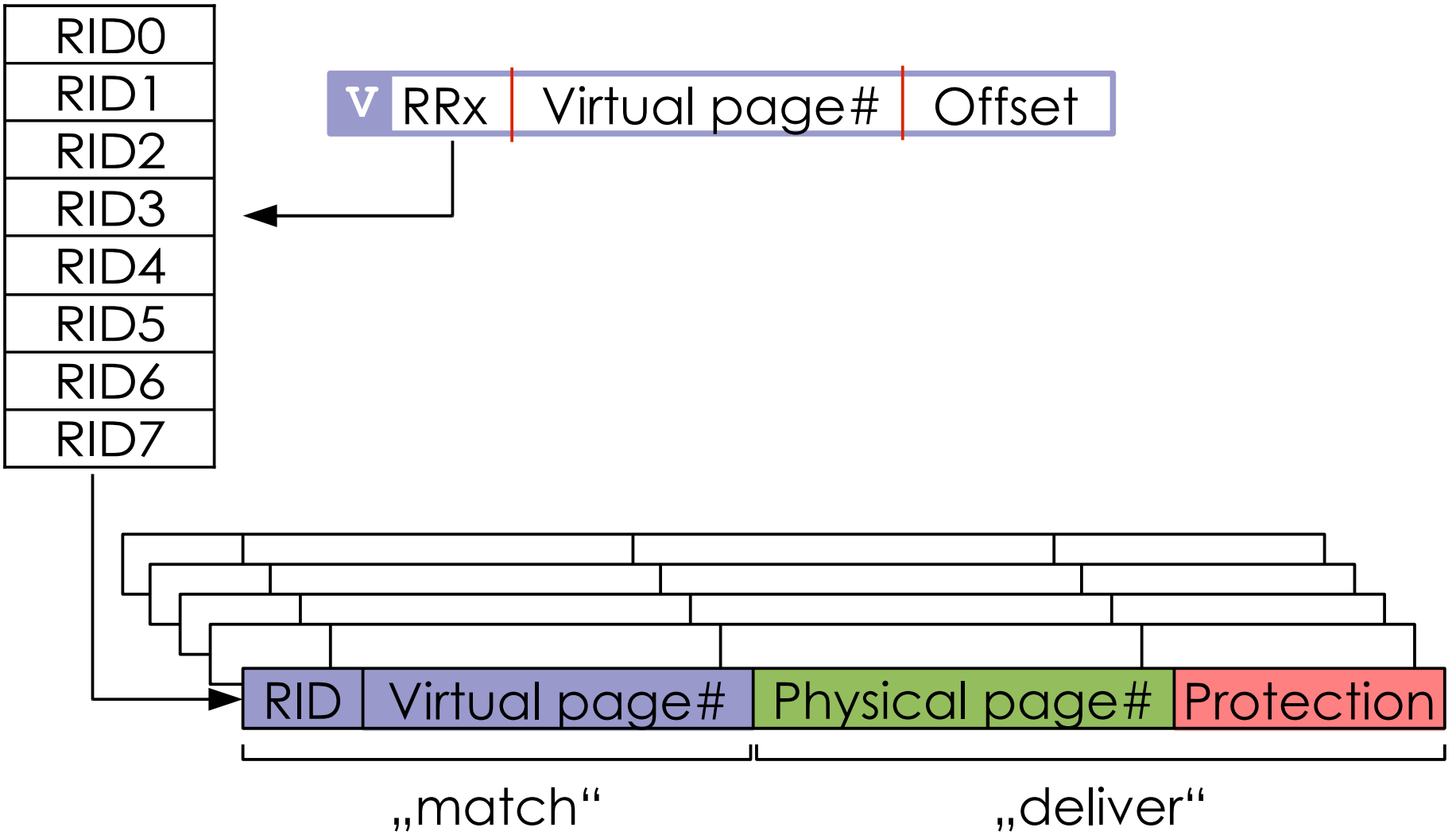
- zu jeder Kachel wird Prozess-Id, Seitennummer geführt
- bei Zugriff („TLB-miss“) wird gesucht

Implementierung als Hash-Tabellen (Hashed Page Tables)

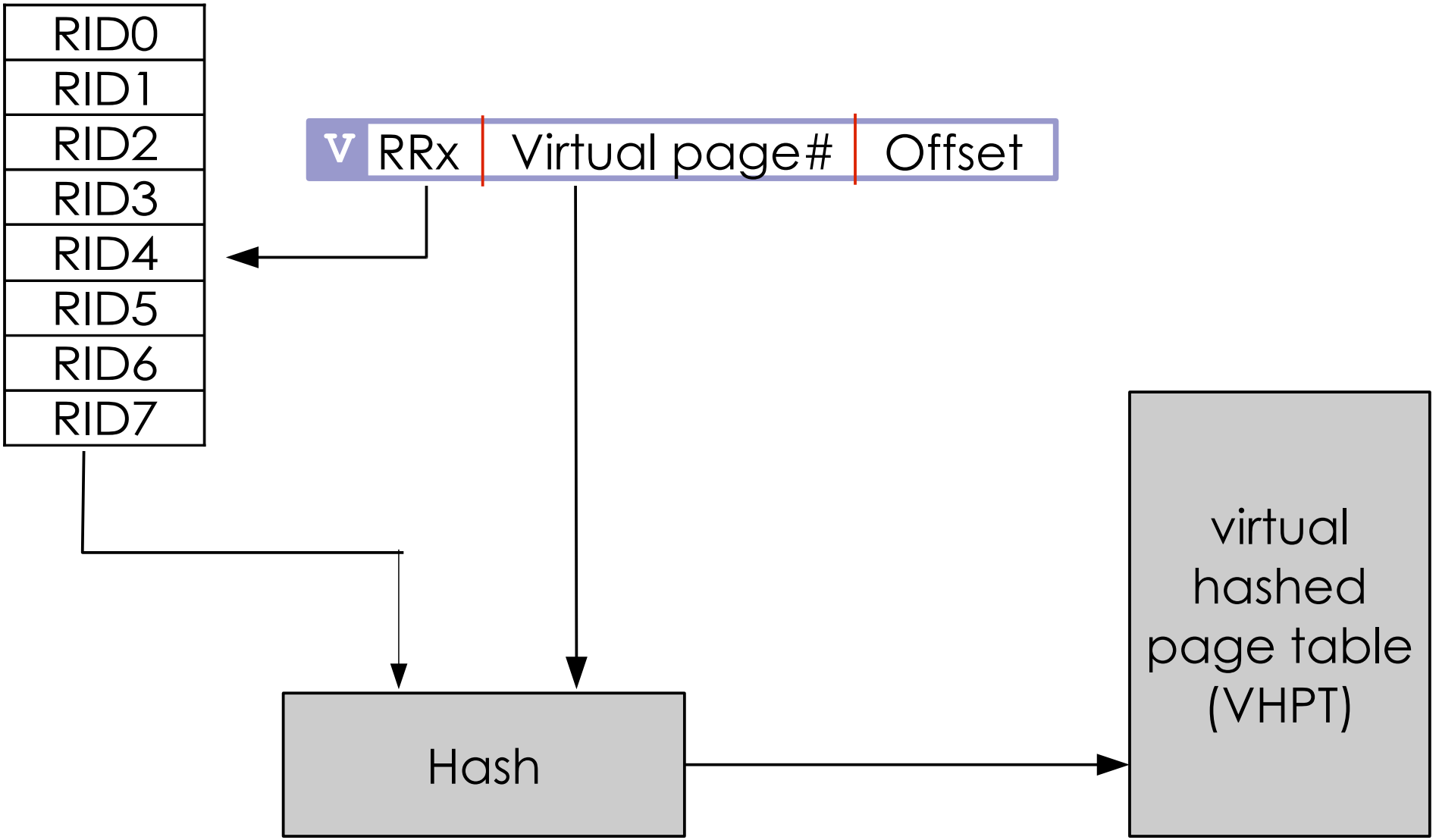
- Vorteile:
  - kleine Seitentabellen
  - abhängig von Anzahl Kacheln
    unabhängig von Größe des Adressraums
- Nachteile:
  - keine Hierarchiebildung (z. B. Schreibschutz für 4 MB)
  - Sharing aufwendig (→ später)
  - Suchaufwand

Beispiel: 64-Bit MMUs - Itanium

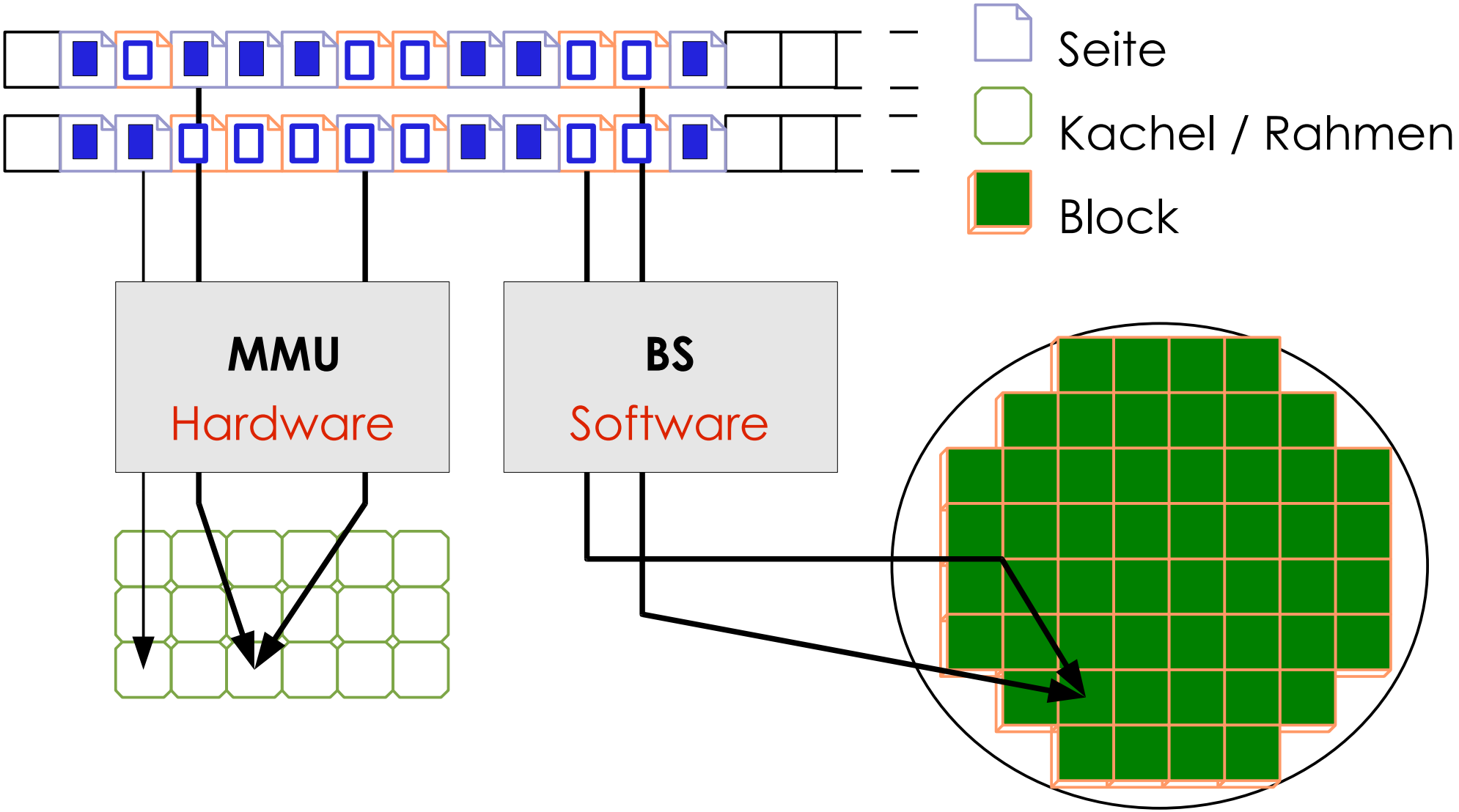
region registers



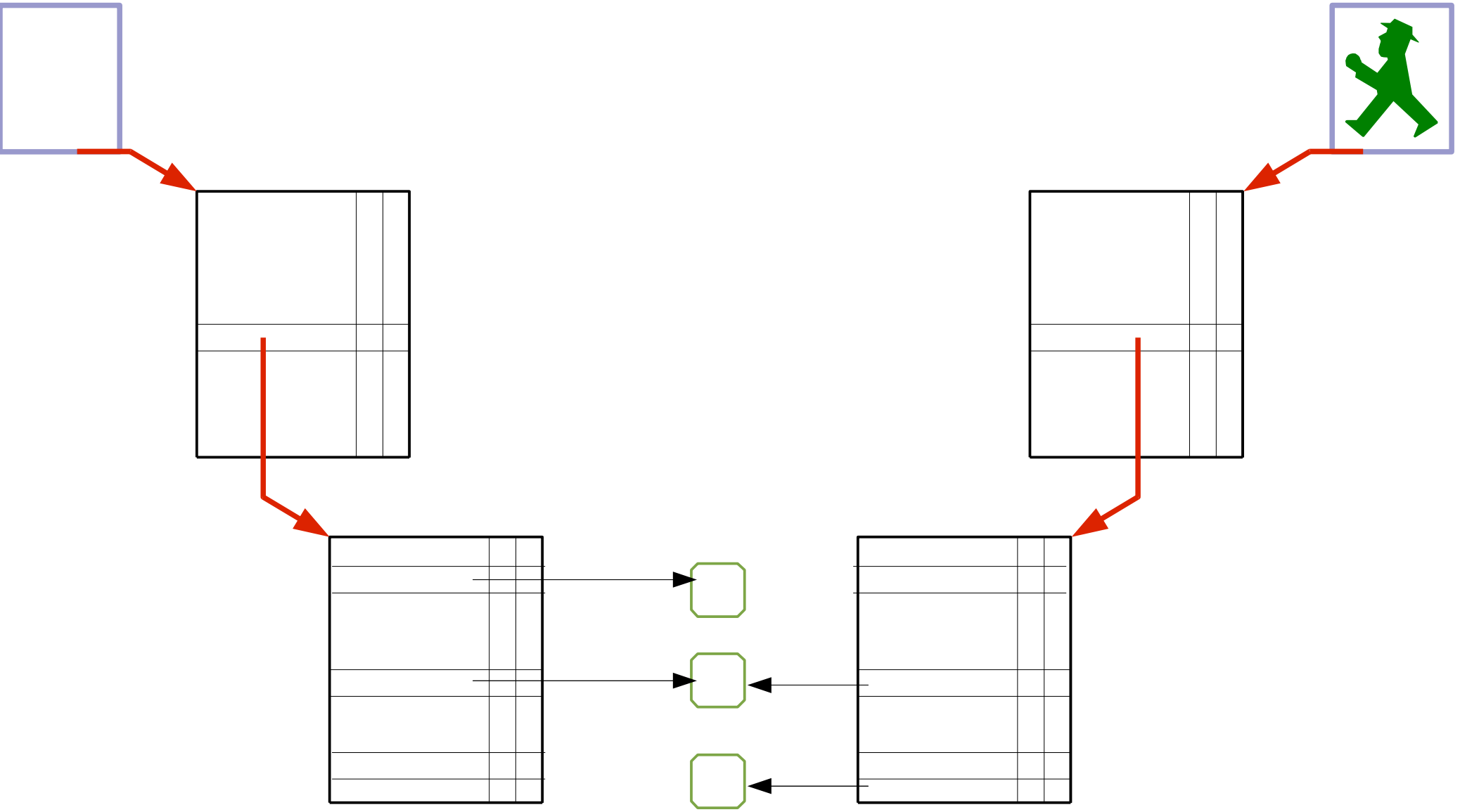
Itanium: Hardware Accessed Page Table



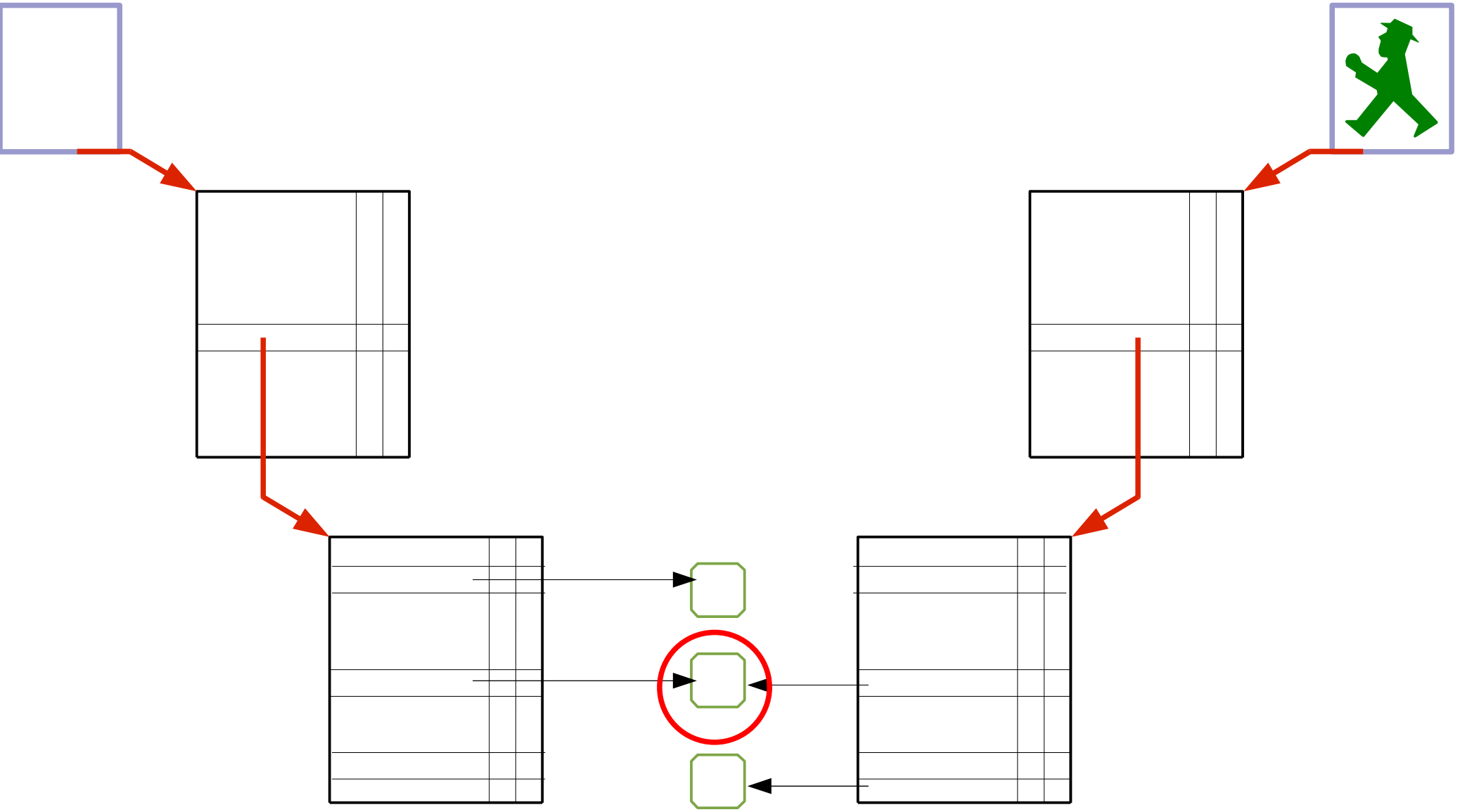
Überlappende Adressräume - „Sharing“



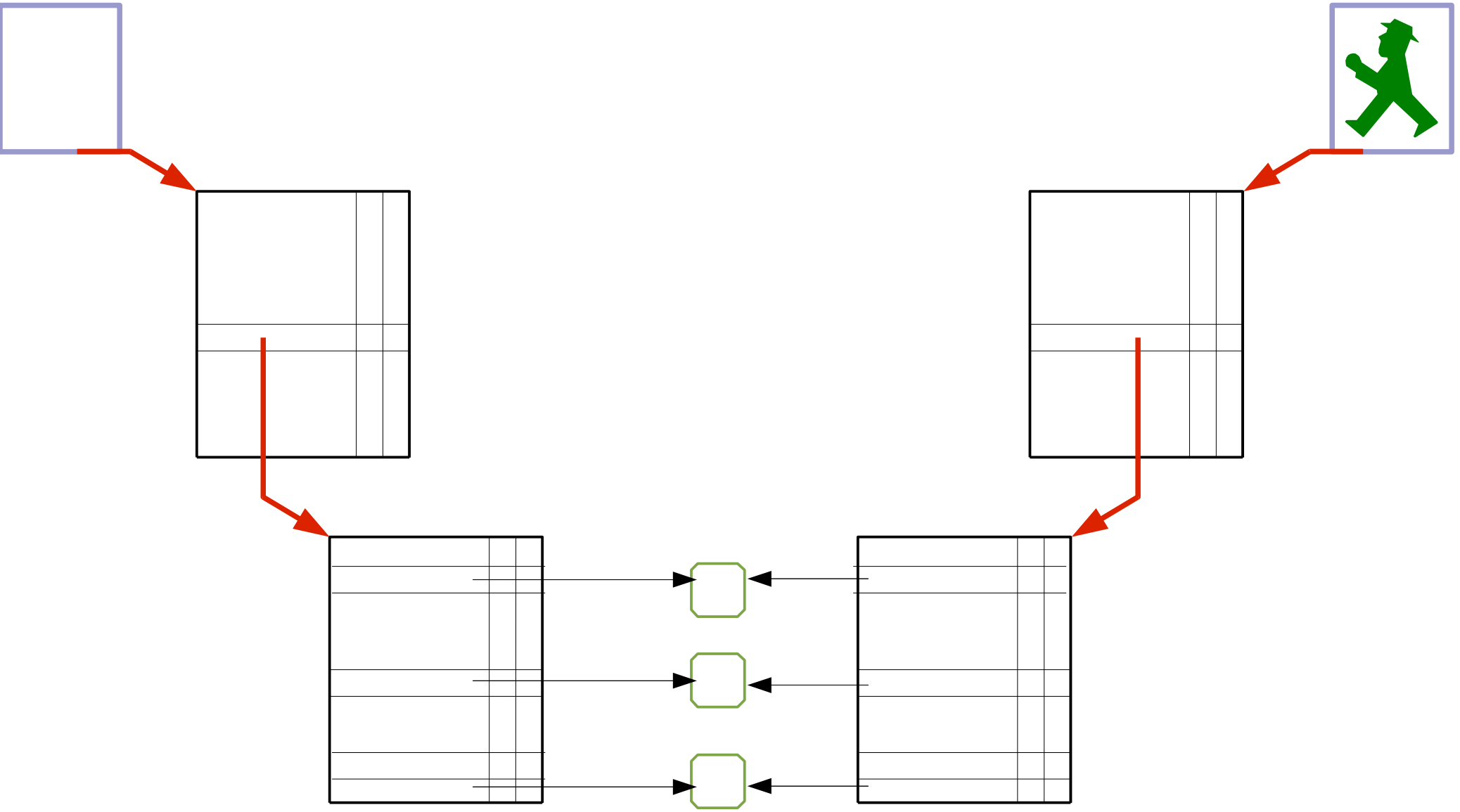
Sharing bei mehrstufigen Seitentabellen



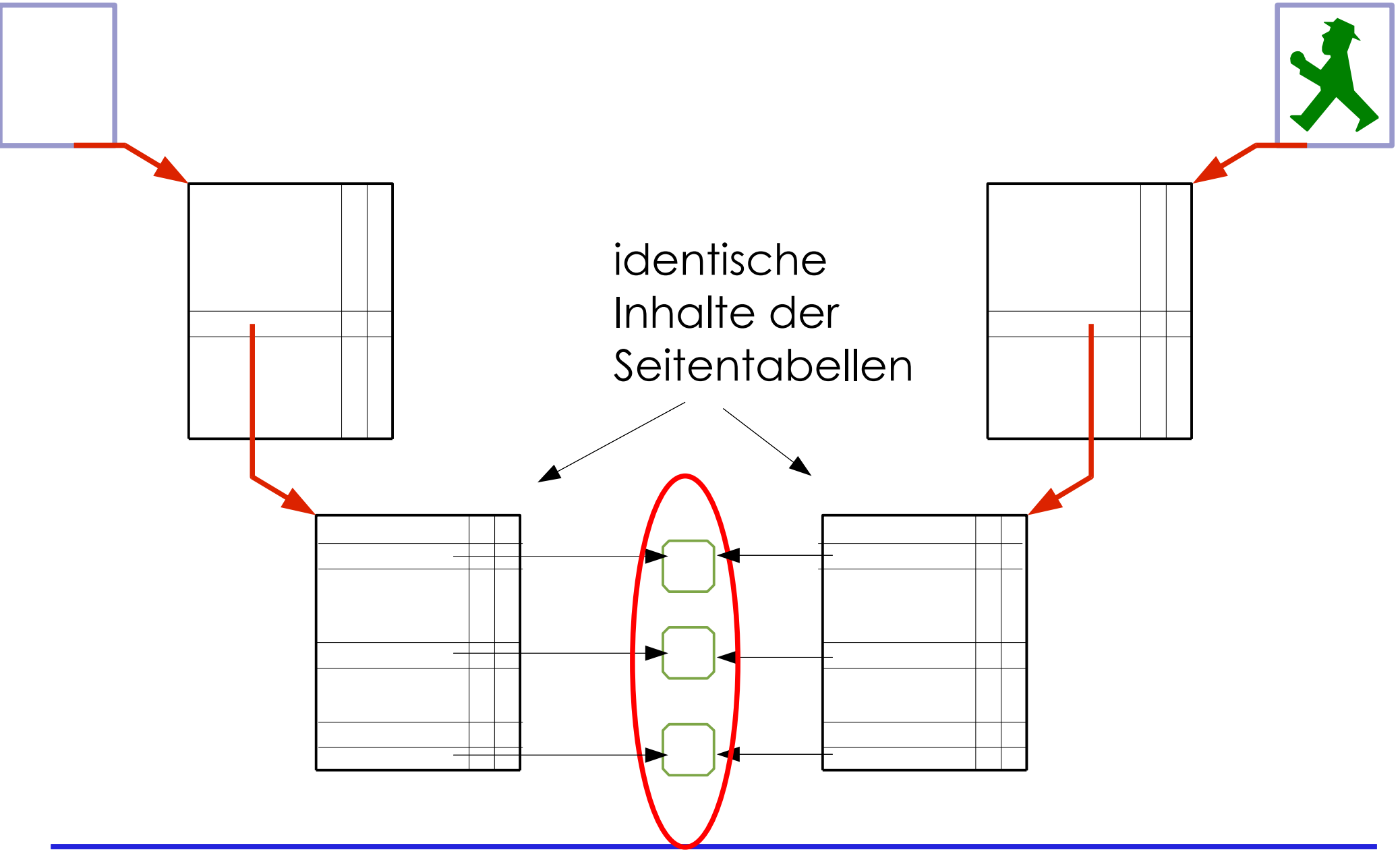
Sharing bei mehrstufigen Seitentabellen



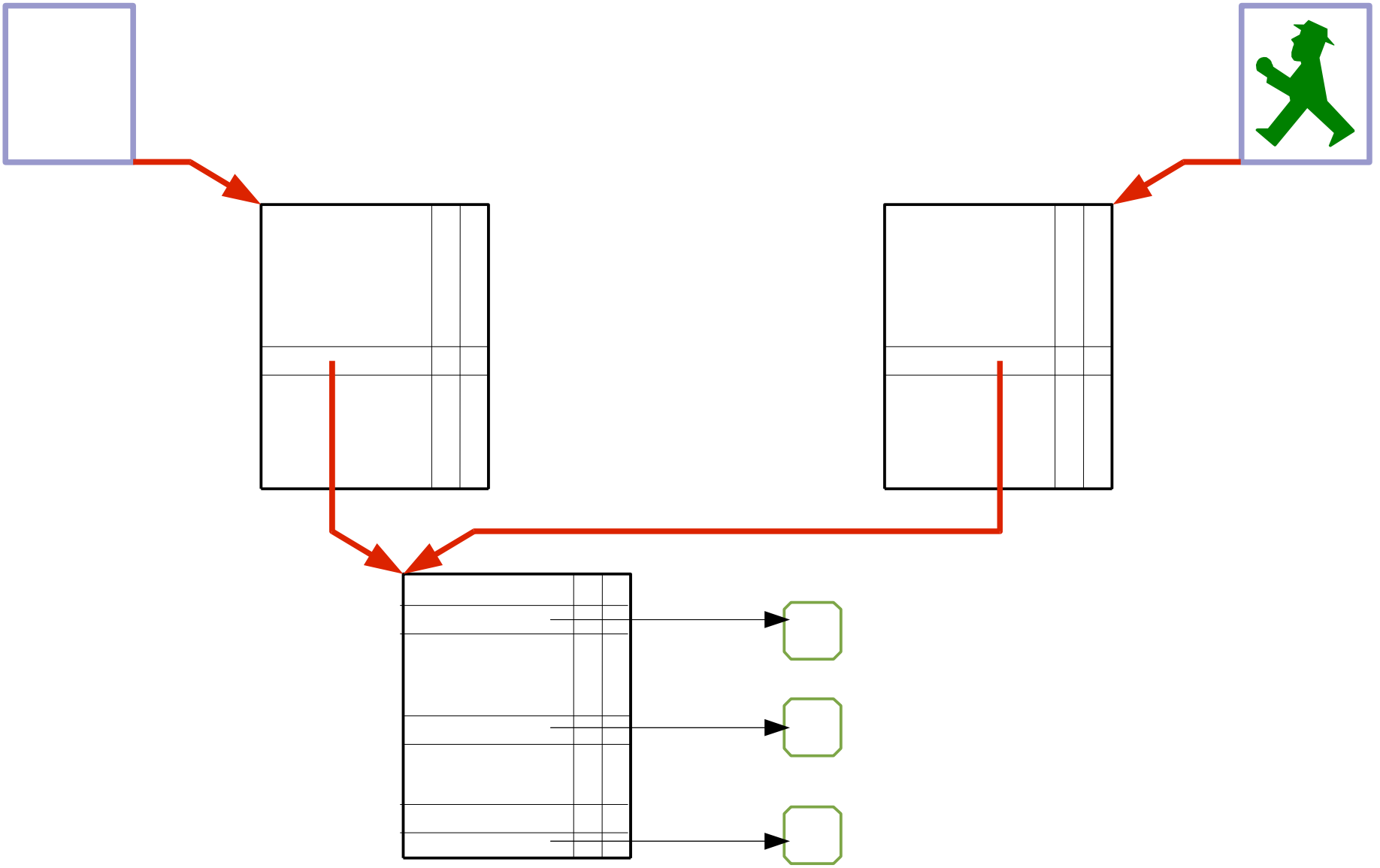
Sharing bei mehrstufigen Seitentabellen



Sharing bei mehrstufigen Seitentabellen



Sharing bei mehrstufigen Seitentabellen



Einsatz von Sharing

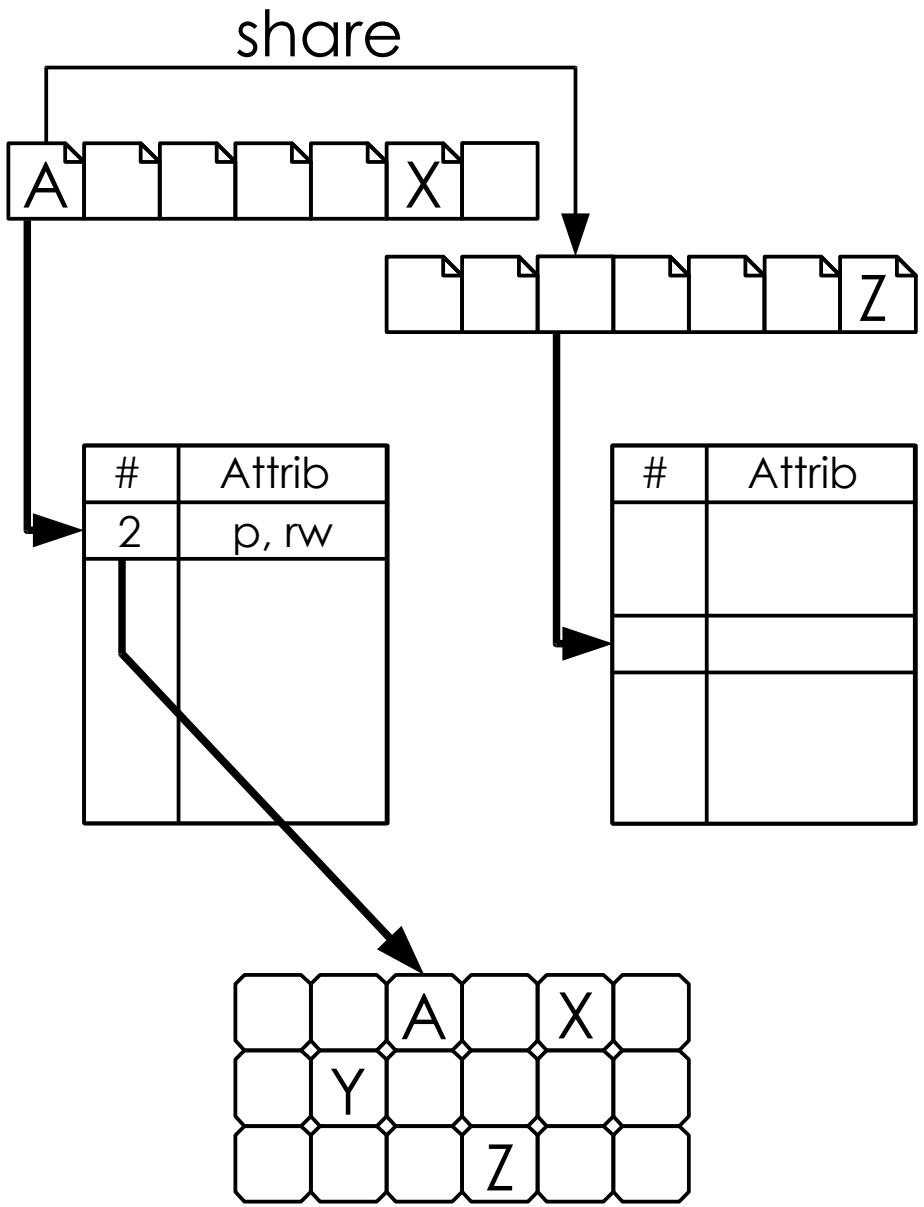
Verzögertes Kopieren (lazy copying, copy on write)

- „Kopieren“:
  - Eintragen des zu kopierenden Bereichs an Zieladresse
  - beide gegen Schreiben schützen
- „faules“ Kopieren:
  - beim ersten schreibenden Zugriff → Seitenfehler
  - Behandlung:
    - neue Kachel allokieren
    - physisch kopieren
    - neue Kachel ohne Schreibschutz in Seitentabelle eintragen

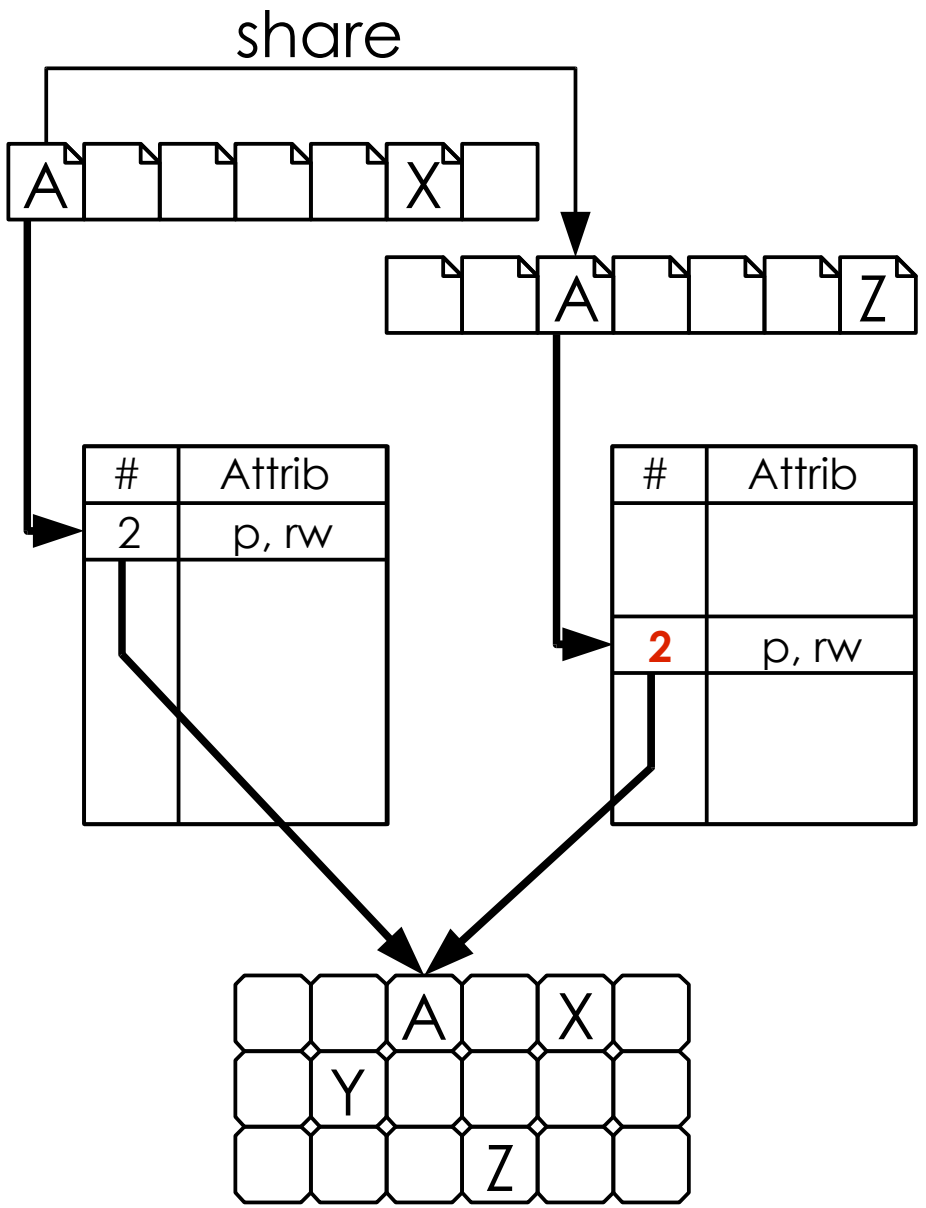
➔ sehr wichtig für effiziente Botschaften, Rücksetzpunkte etc.

➔ gemeinsame Nutzung von Daten/Programmen

Gemeinsames Nutzen von Daten

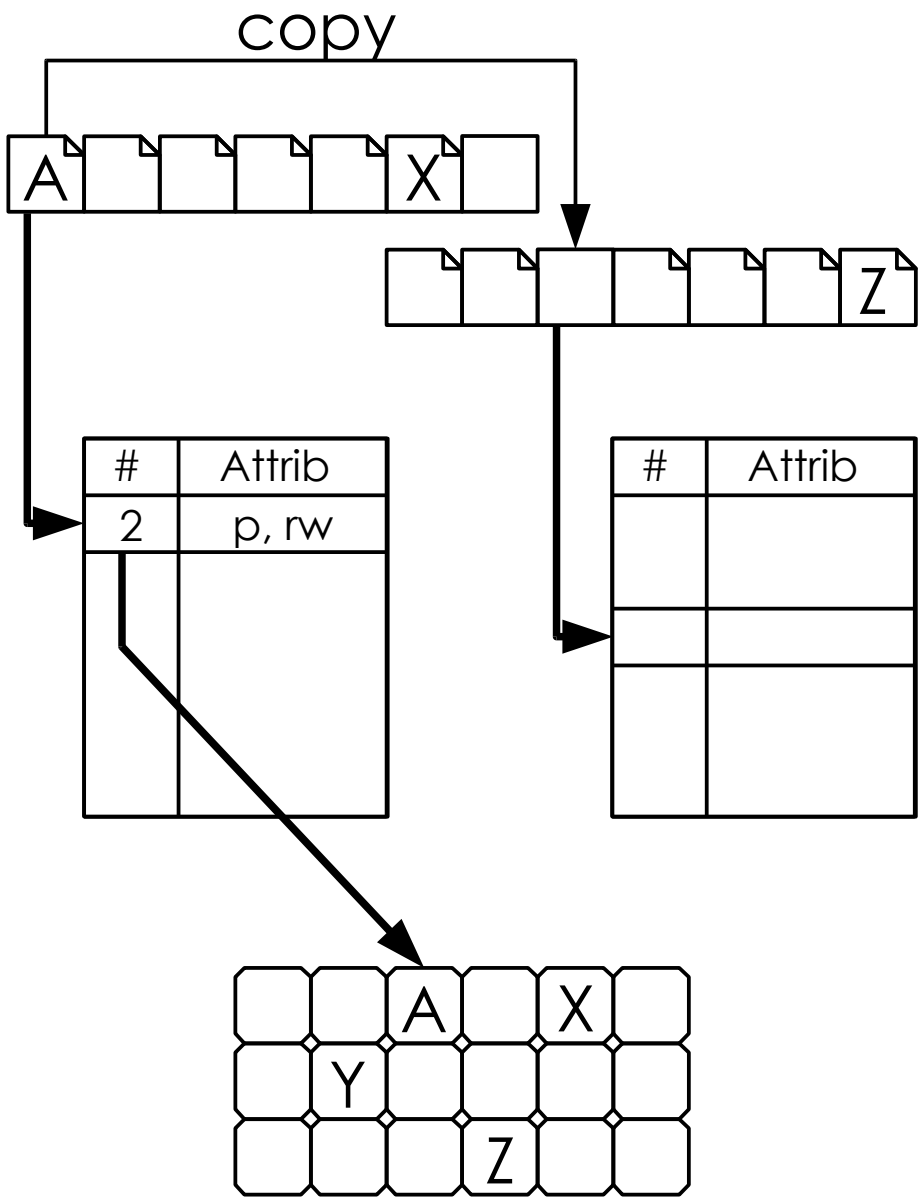


Gemeinsames Nutzen von Daten

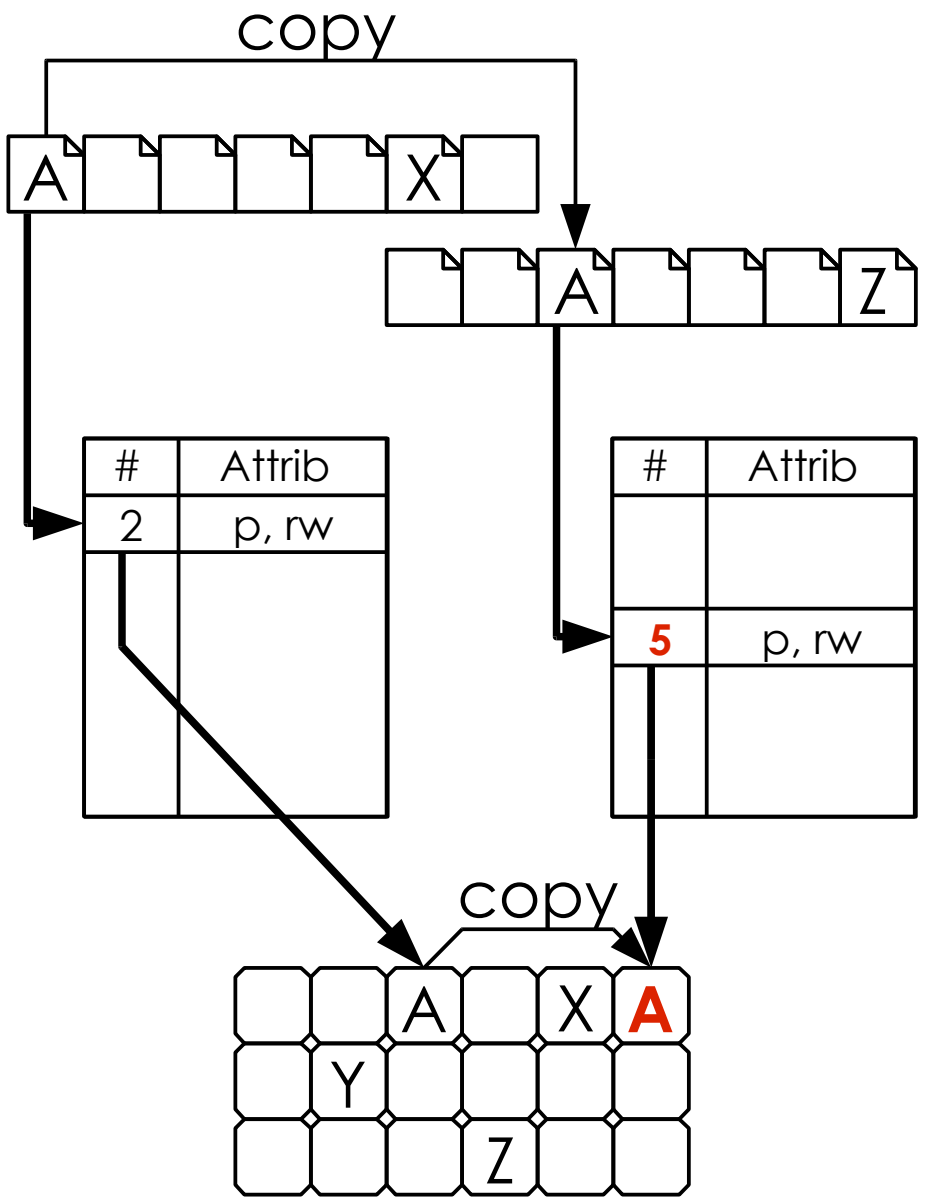


- neuen MMU Eintrag an Zieladresse

Echtes Kopieren

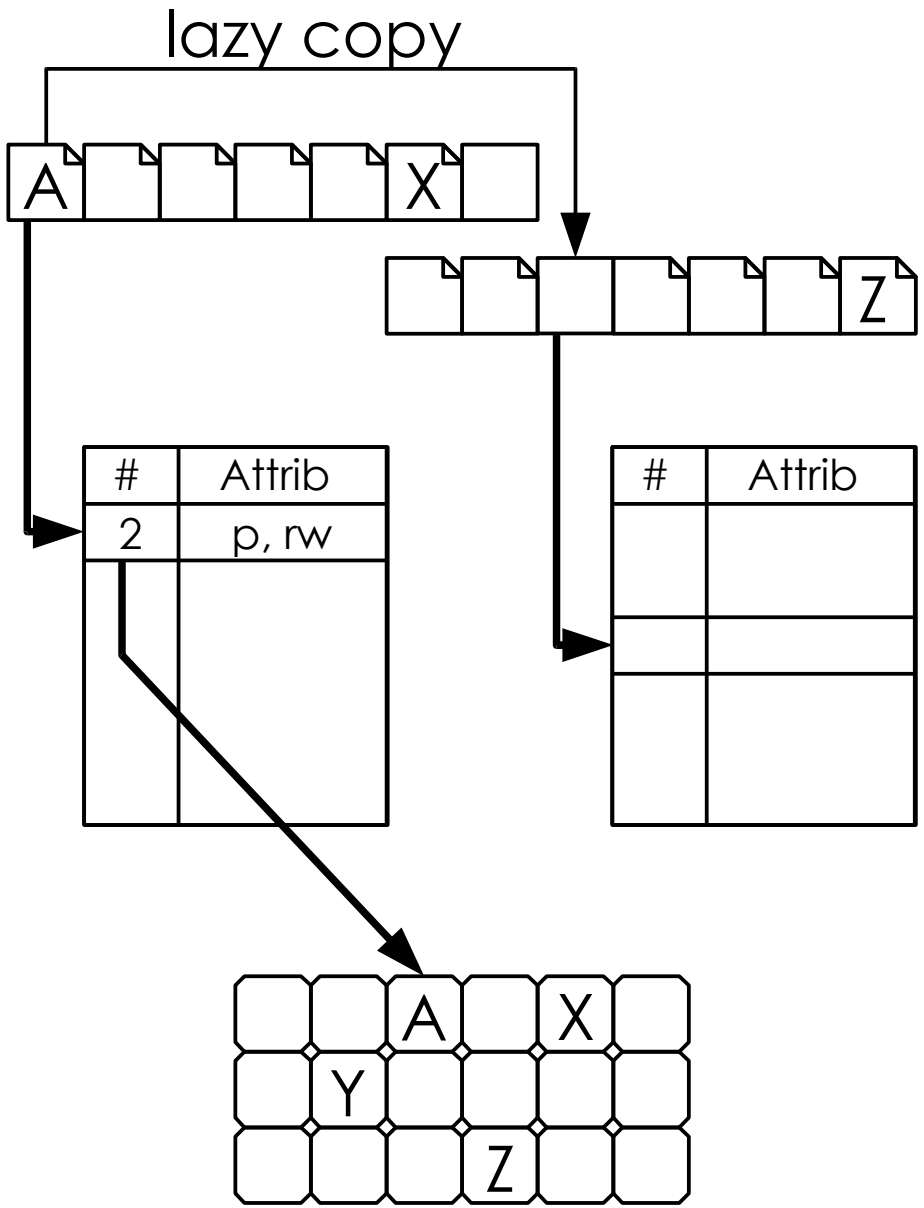


Echtes Kopieren

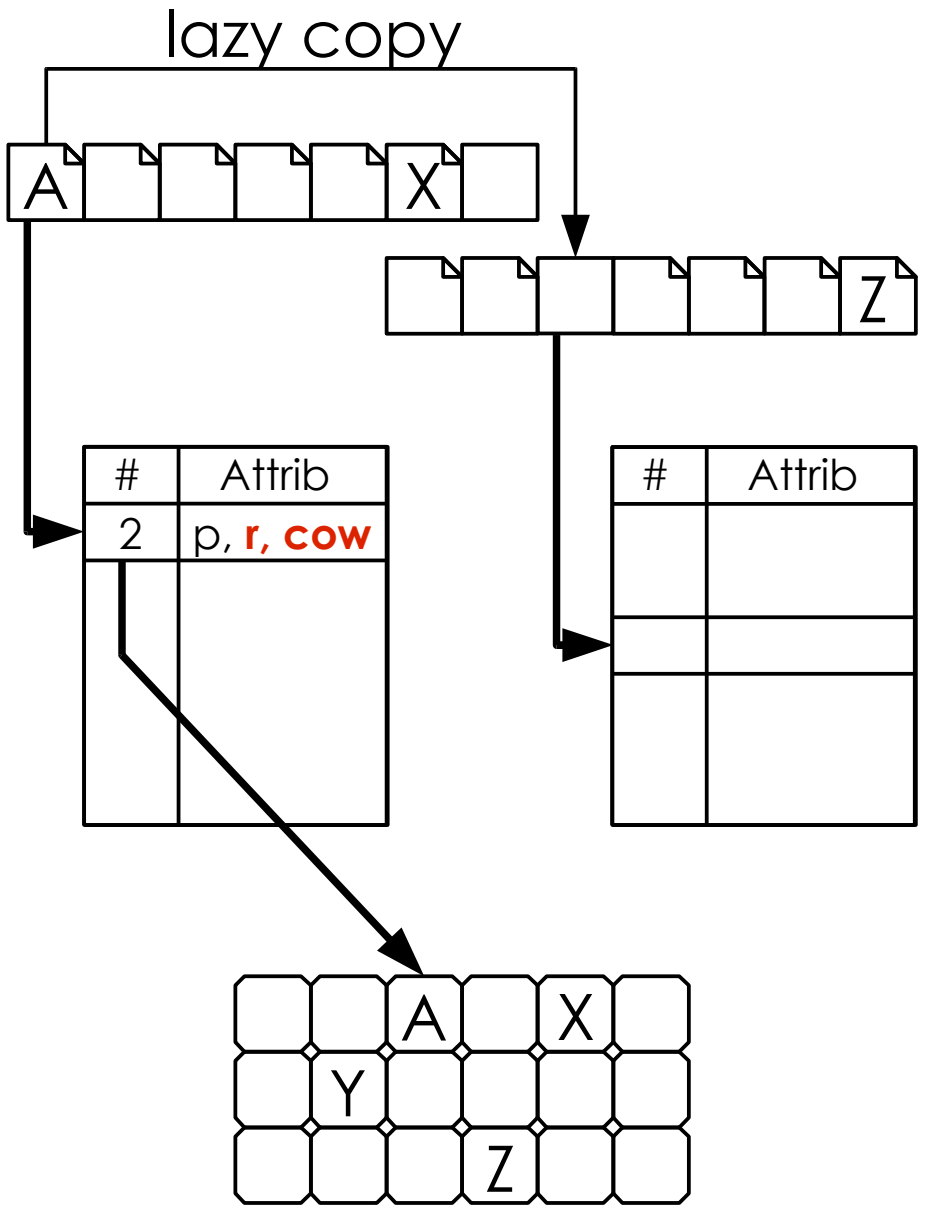


- neue Kachel besorgen
- Kacheln kopieren
- neuen Eintrag in die Seitentabelle für Zieladresse

Verzögertes Kopieren

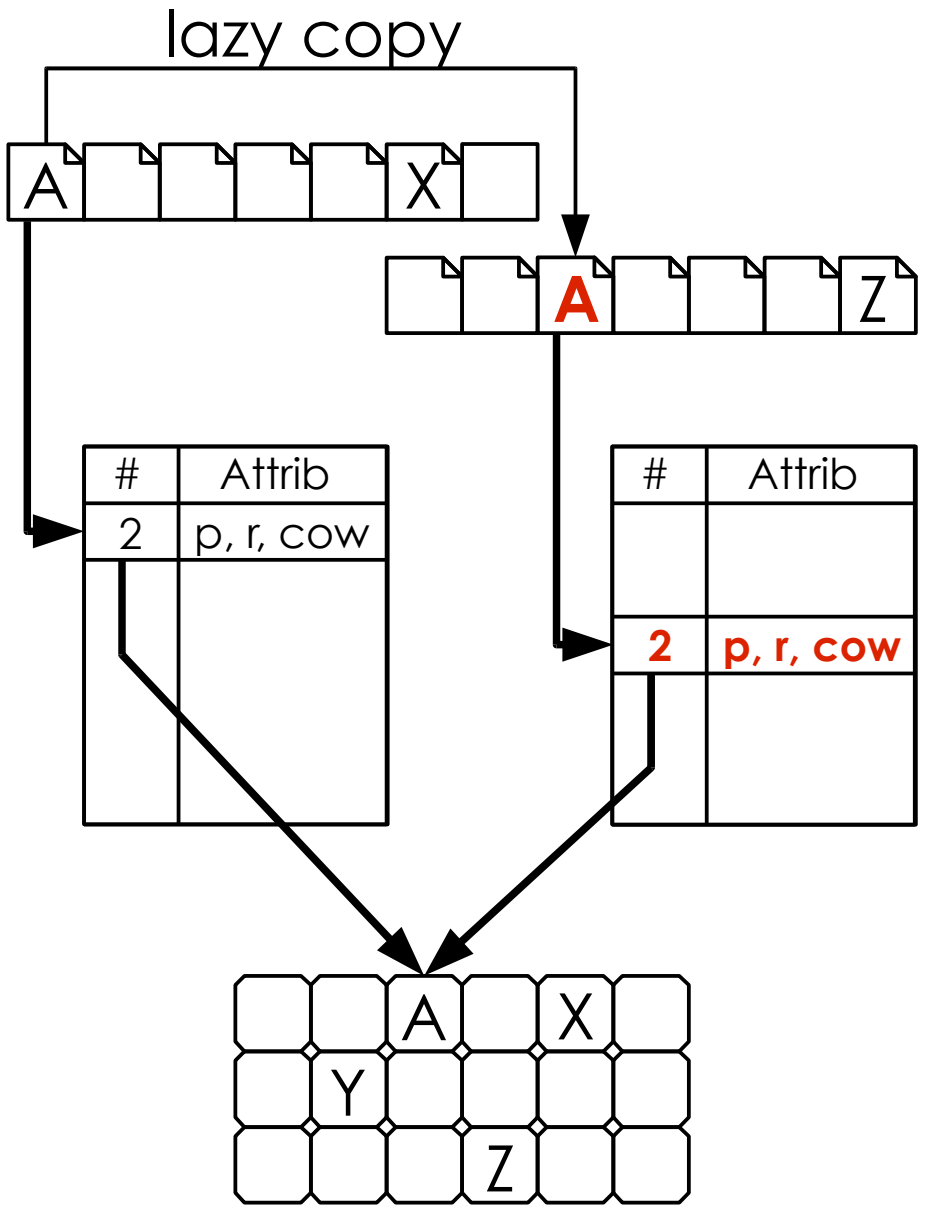


Verzögertes Kopieren



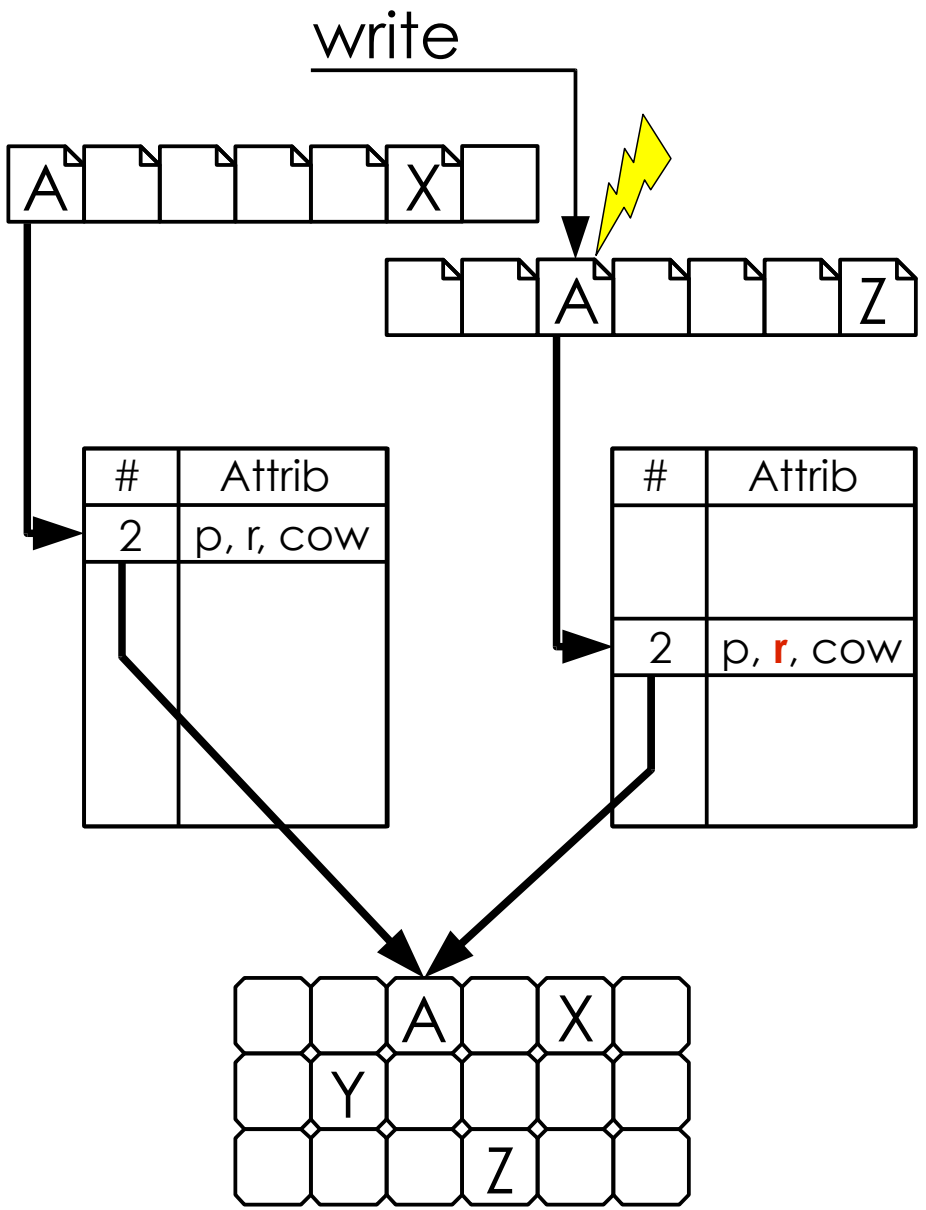
- Ändern des MMU-Eintrags an der Quelladresse: rw → (r, cow)

Verzögertes Kopieren



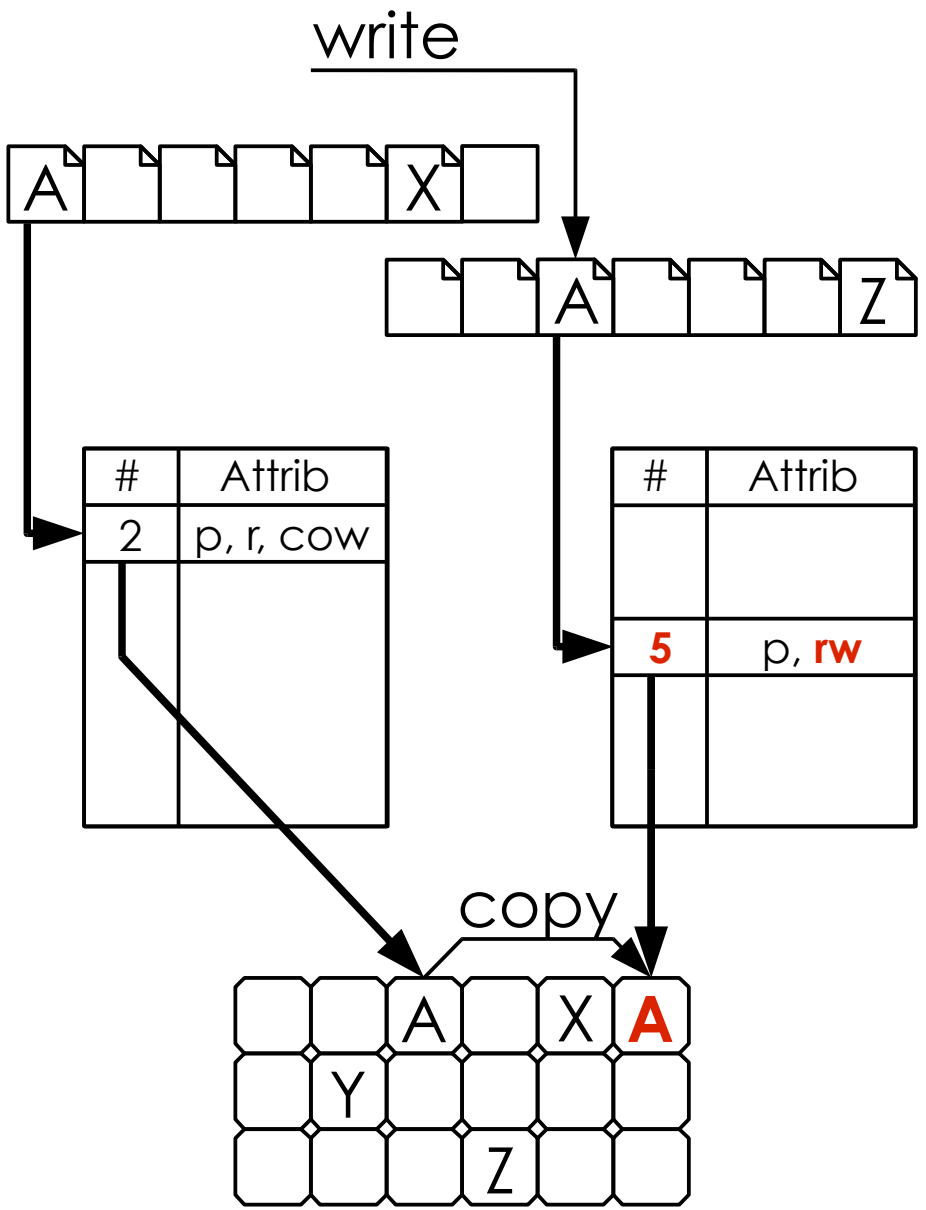
- Ändern des MMU-Eintrags an der Quelladresse: rw → (r, cow)
- neuen MMU-Eintrag an Zieladresse: (r, cow)

Verzögertes Kopieren



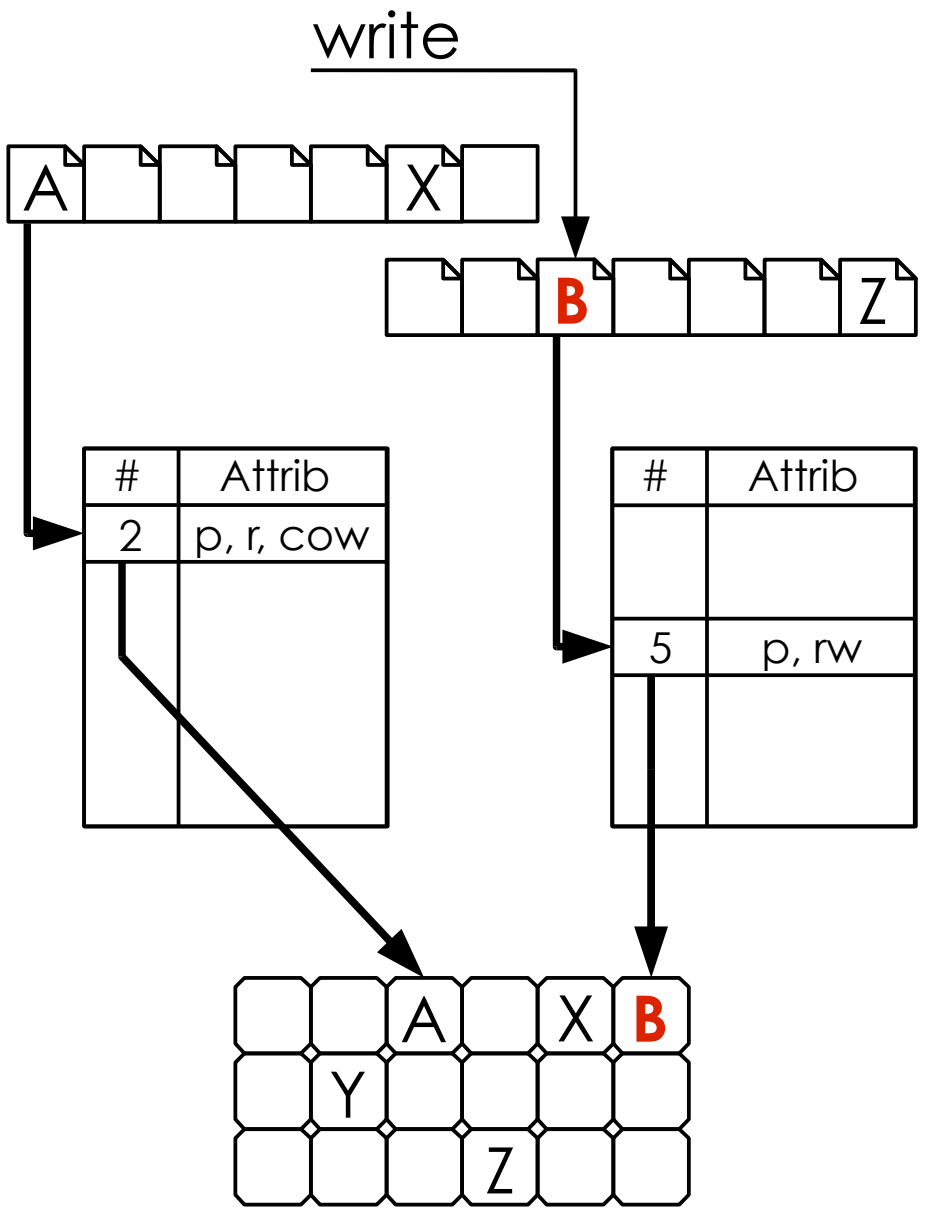
- schreibender Zugriff → Seitenfehler

Verzögertes Kopieren



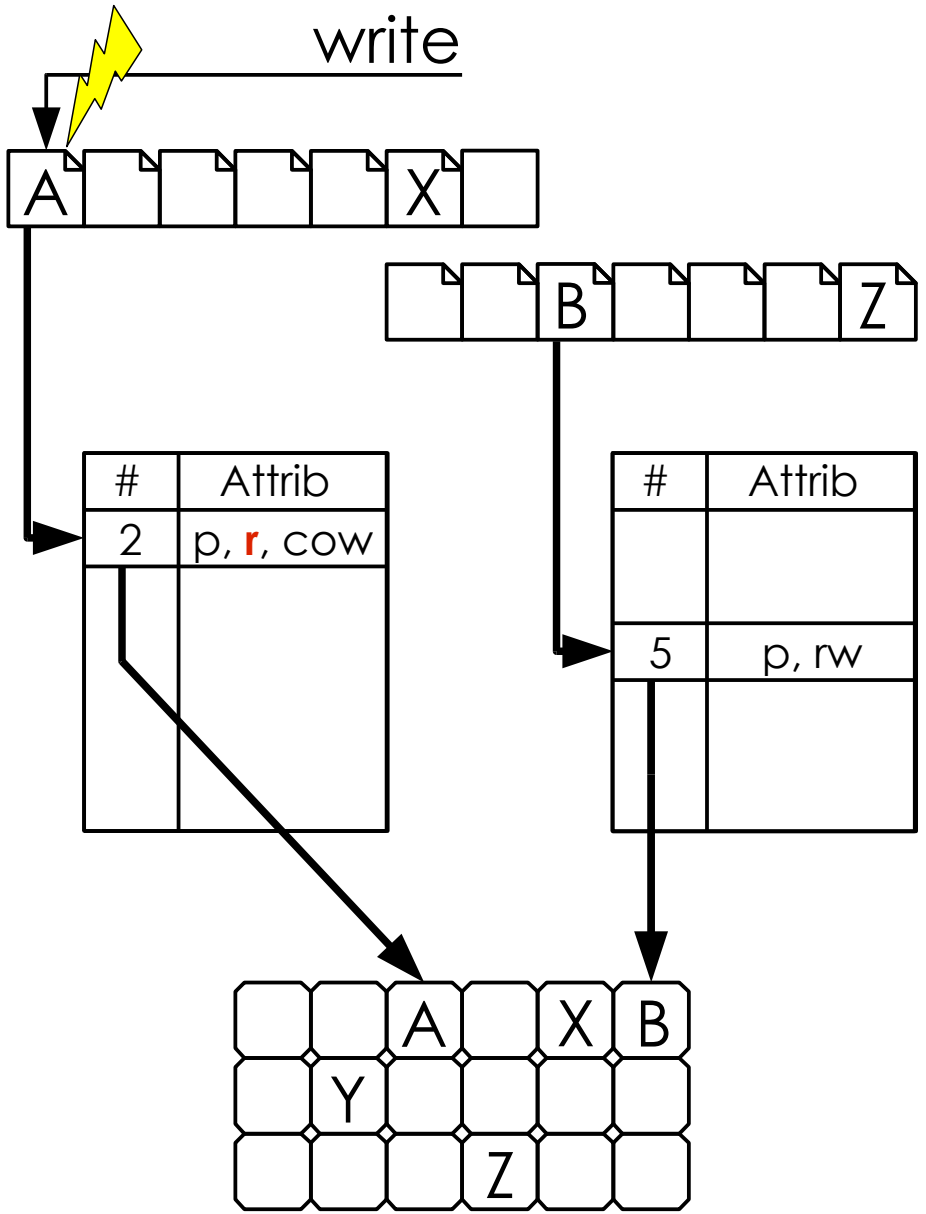
- schreibender Zugriff → Seitenfehler
- Das cow-Attribut bewirkt, dass eine neue Kachel allokiert und der Inhalt physisch kopiert wird
- neuen MMU-Eintrag an Zieladresse: rw

Verzögertes Kopieren



- schreibender Zugriff → Seitenfehler
- Das cow-Attribut bewirkt, dass eine neue Kachel allokiert und der Inhalt physisch kopiert wird
- neuen MMU-Eintrag an Zieladresse: rw
- danach erfolgt der Schreibzugriff

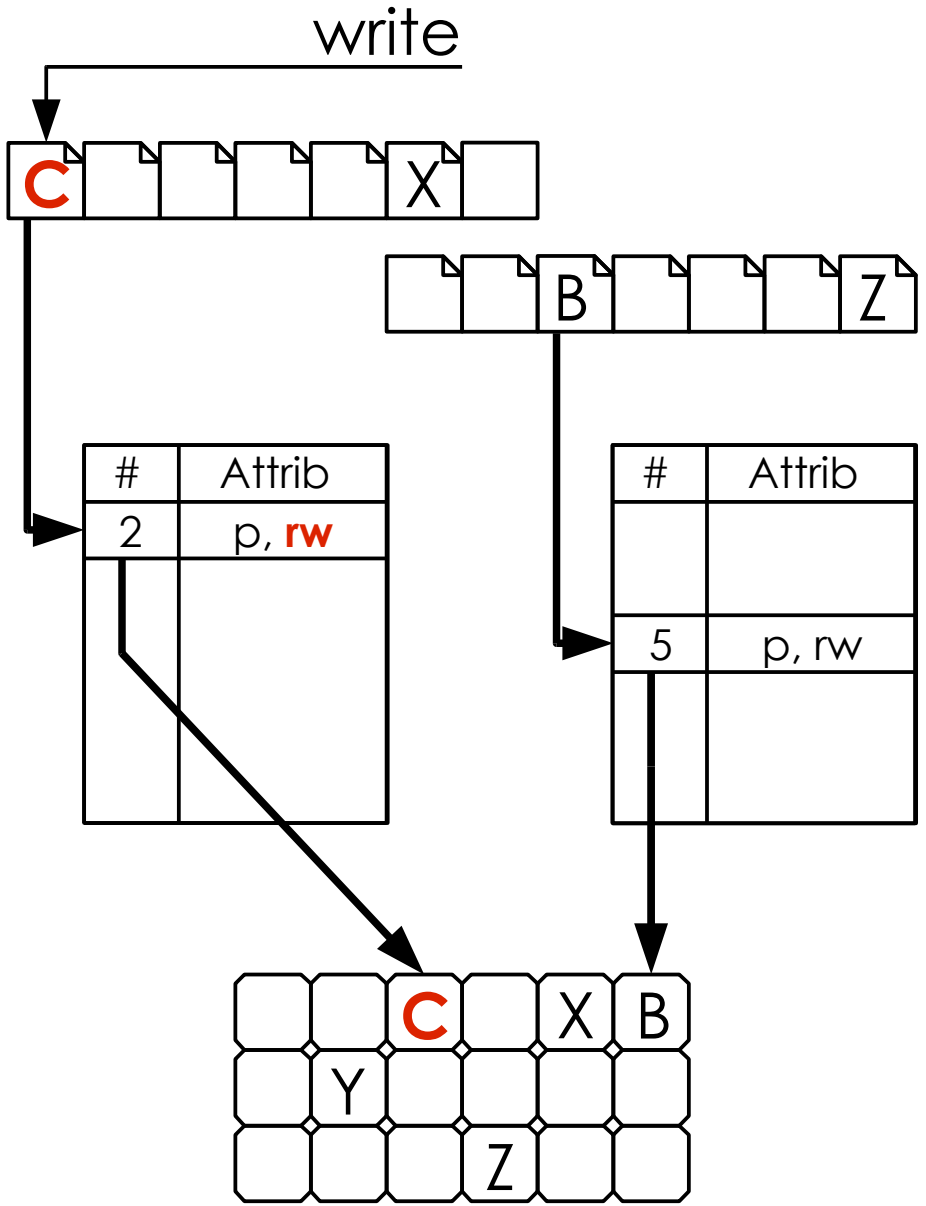
Verzögertes Kopieren



- schreibender Zugriff → Seitenfehler

➔ ist der Prozess der alleinige Besitzer der Kachel, wird lediglich das Schreibrecht gesetzt

Verzögertes Kopieren

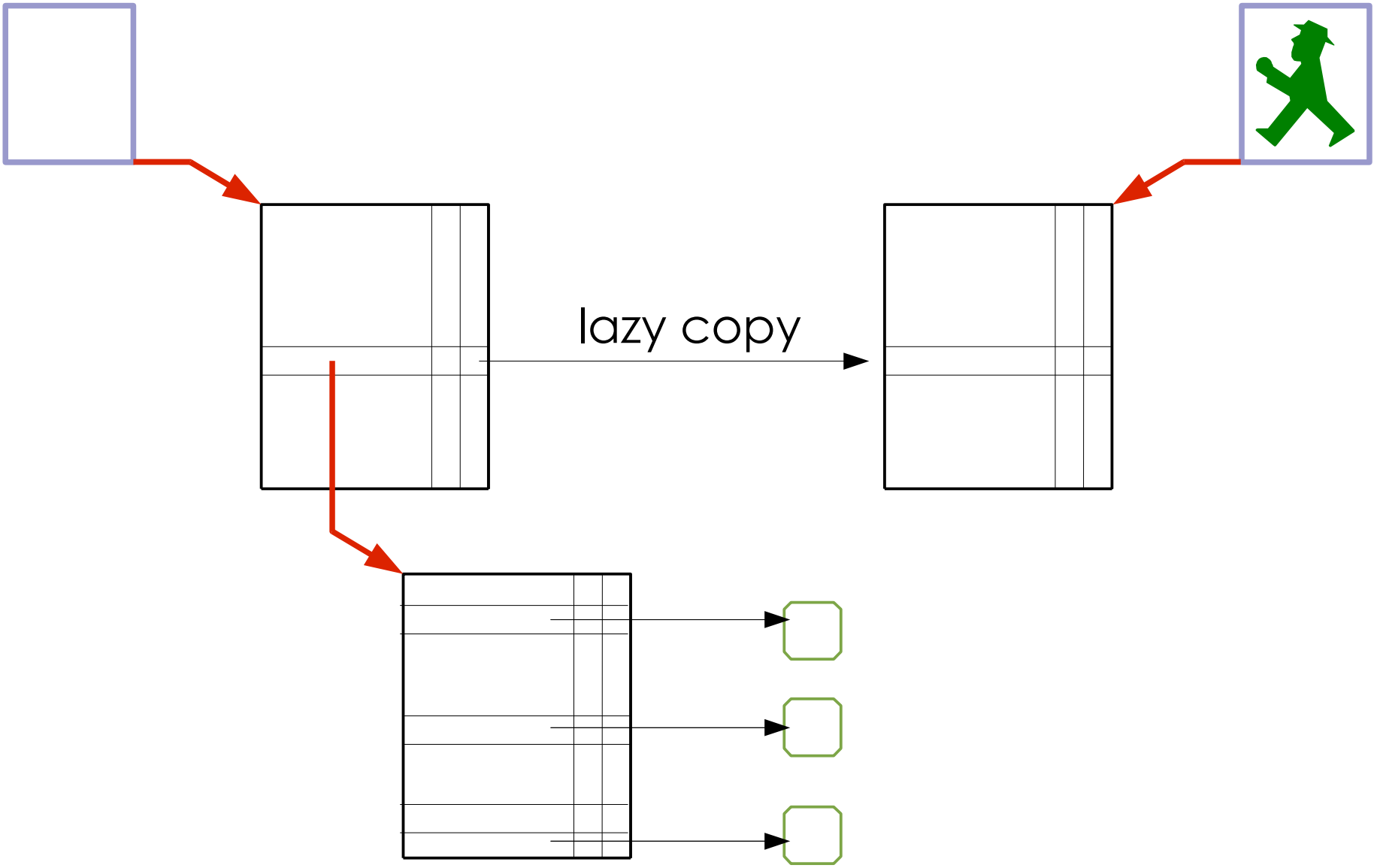


- schreibender Zugriff → Seitenfehler

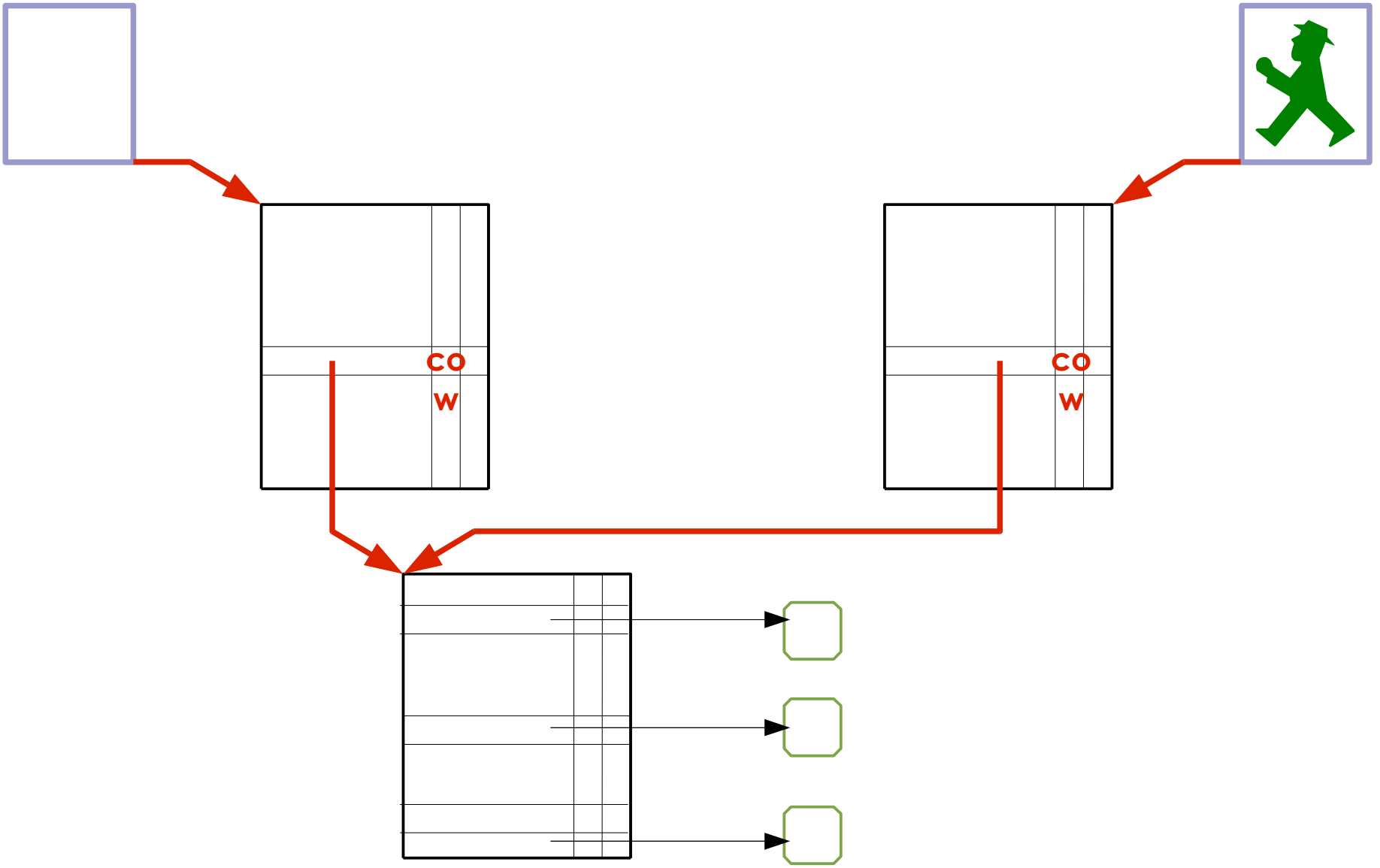
➔ ist der Prozess der alleinige Besitzer der Kachel, wird lediglich das Schreibrecht gesetzt

- danach erfolgt der Schreibzugriff

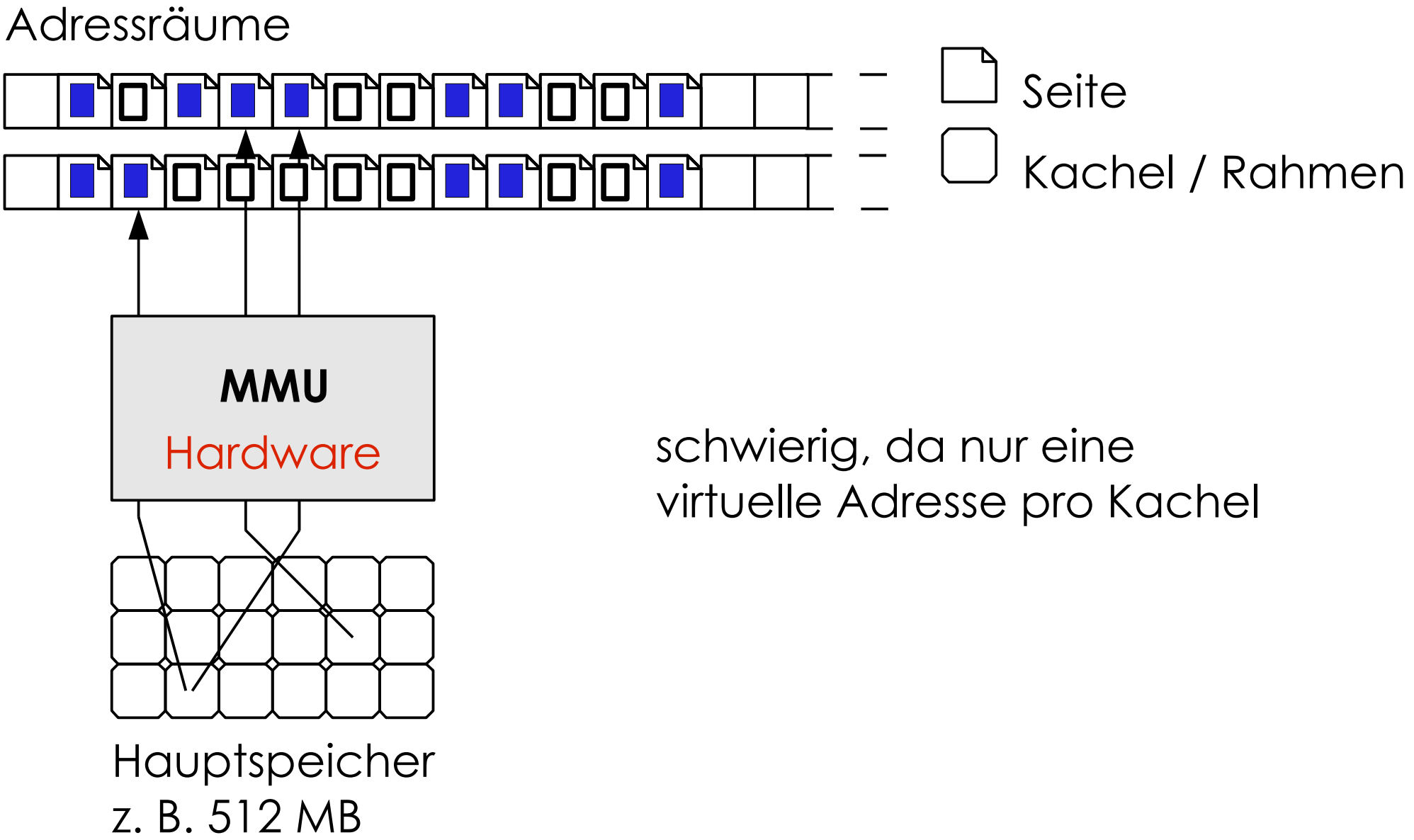
Verzögertes Kopieren großer Bereiche



Verzögertes Kopieren großer Bereiche

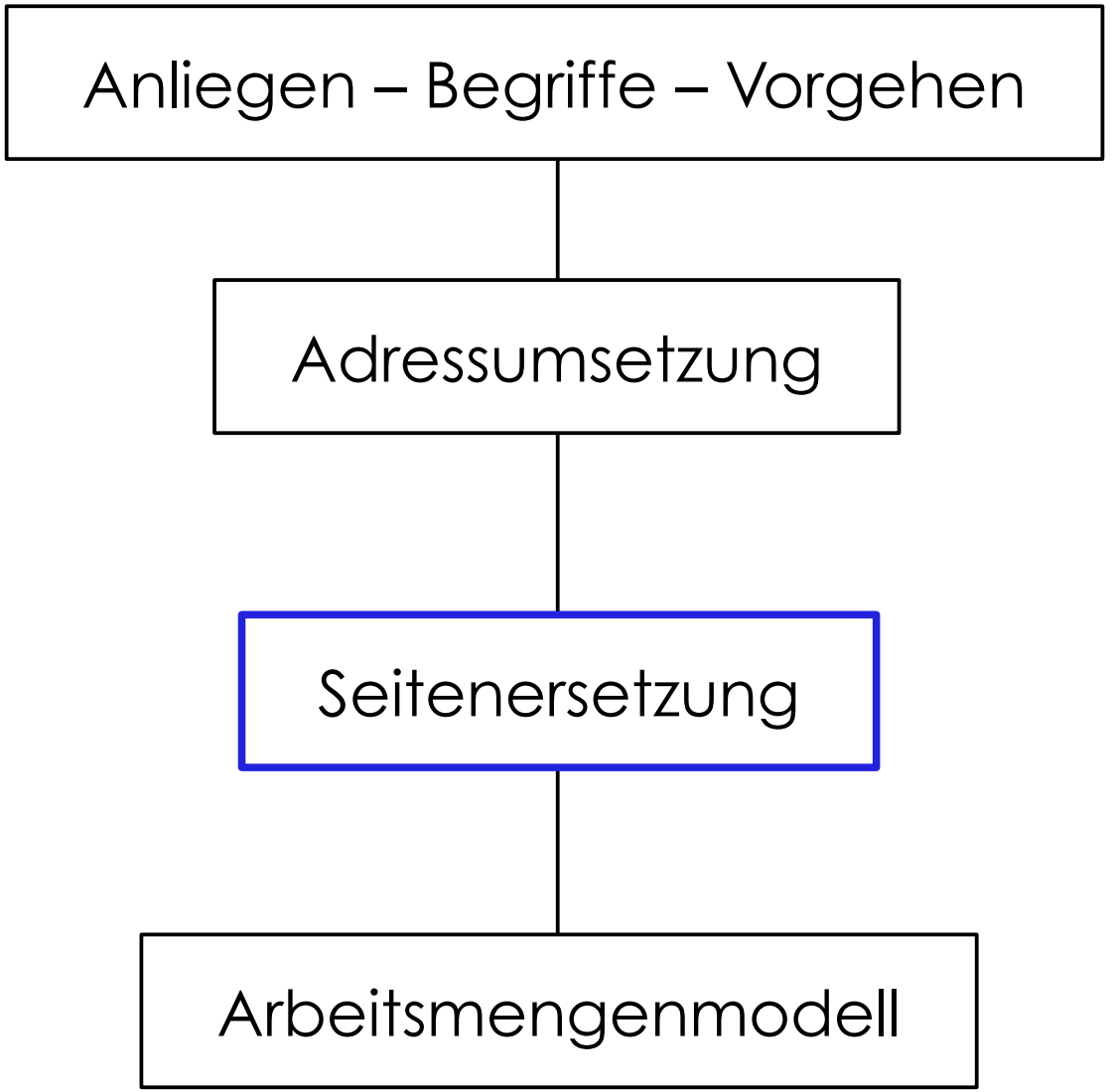


Sharing und invertierte Seitentabellen



schwierig, da nur eine
virtuelle Adresse pro Kachel

Wegweiser: Virtueller Speicher



Virtueller Speicher im Betriebssystem

Teilaufgaben

- Seitenfehler-Behandlung
- Verwaltung des Betriebsmittels Hauptspeicher
- Aufbau der Adressraumstruktur
  (Speicherobjekte und Regionen)
- Bereitstellung spezifischer Speicherobjekte
- Interaktion Prozess- und Speicher-Verwaltung

Seitenfehlerbehandlung (Überblick)

```
trap (HW) {

 • rette Register
 • Ermittlung der Seitenfehleradresse

   ◆ Regionmanager: Speicherobjekt bestimmen
   ◆ freie Kachel ermitteln oder verschaffen
   ◆ falls nötig, alten Kachelinhalt
        zurückschreiben (Prozess wartet)
   ◆ Seiteninhalt in Kachel laden
        (Prozess wartet)
   ◆ Seitentabelle aktualisieren

 •  return from trap (Instruktion wird
                      wieder aufgesetzt)
}
```

Hauptspeicher: Betriebsmittelverwaltung

Aufgaben

- gewährt und entzieht bei Knappheit Kacheln
- Buchführung → wo sind welche Kacheln?

➔ welche Kacheln werden verdrängt?

Seitenaustauschverfahren

- Seitenverdrängung
- Seitenersetzung

Seitenersetzungsverfahren

Problem

Eine Kachel wird benötigt, eine Seite ist zu verdrängen.

➔ wie und welche ??

Wie wird verdrängt?

Kachel aus Seitentabelle austragen (TLB-Eintrag löschen)
falls modifiziert → zurückschreiben auf persistenten Speicher

Welche Seite wird verdrängt?

- optimal: die Seite wird verdrängt, die am spätesten in der Zukunft benutzt werden wird
- stattdessen:

– Betrachtung der Vergangenheit (verschiedene Algorithmen)

– Ausnutzung des Lokalitätsverhaltens („Arbeitsmengen-Modell“)

Seitenersetzung allgemein

Hardwareseitig

```
    ... arbeiten ...
→   HW setzt referenced- und
    modified-Flag
```

BS-seitig

```
void pagefault(Seiten_NR) {

   NewK = getFreieKachel();

   if (!NewK) {
     //keine Kachel mehr frei
     NewK = ErsetzungsStrategie();

     //NewK ist jedoch noch in
     //Benutzung
     if (NewK.modified)
             writeToDisk(NewK, ...);
     removeFromPT(NewK);
     //aus altem AR entfernen
   }

   NewContent(NewK);
    //z.B. fillWithZero
   //oder readFromDisk(Seiten_Nr,NewK)

   insertIntoPT(Seiten_NR, NewK);
   //Kachel im richtigen AR, bei der
   //richtigen SeitenNR einfügen
}
```

Generelle Überlegung

- Annahme:

  Ersetzungsstrategie basierend auf festen Inspektionsintervallen

wähle beliebigen Rahmen im System nach folgender Priorität:

1) **nicht benutzt, nicht modifiziert**
2) **benutzt, nicht modifiziert**
3) **nicht benutzt, modifiziert**
4) **benutzt, modifiziert**

FIFO: First in first out

Verdränge ***älteste*** Seite

→ d.h. Inhalt der Kachel, die schon am längsten ihren jetzigen Inhalt hat

Vorteil

- keine Information über tatsächliches Referenzverhalten nötig
- einfach implementierbar

Nachteil

- ignoriert Lokalitätsverhalten
- BELADY's Anomalie

LRU: Least Recently Used

Verdränge am ***längsten*** *nicht genutzte* Seite

Vorteil

- gilt als gute Näherung für optimalen Algorithmus

Nachteil

- aufwendige Realisierung
  jeder Speicherzugriff müsste berücksichtigt werden
  - ➔ HW-Unterstützung wird benötigt

➔ Annäherungen per Software

Annäherung an LRU

Hardwareseitig

```
      ... arbeiten ...
→  HW setzt referenced- und
   modified-Flag
```

BS-seitig

```
void init(Kachel) {
   Kachel.modified = 0;
   Kachel.referenced = 0;
}
```

```
void resetThread(){
 //wird im BS aufgerufen

 for(i = 0; i < maxK; i++) {
 Auswahl(Kachel[i],referenced);
 Kachel[i].referenced = 0;
 }
}
```

bei Seitenfehler

```
void pagefault(Seiten_NR) {

   NewK = getFreieKachel();

   if (!NewK) {
     //keine Kachel mehr frei
     NewK = ErsetzungsStrategie();

     //NewK ist jedoch noch in
     //Benutzung
     if (NewK.modified)
             writeToDisk(NewK, ...);
     removeFromPT(NewK);
     //aus altem AR entfernen
   }

   NewContent(NewK);
   //z.B. fillWithZero
   //oder readFromDisk(Seiten_Nr,NewK)

   insertIntoPT(Seiten_NR, NewK);
   //Kachel im richtigen AR, bei der
   //richtigen SeitenNR einfügen
}
```

Seitenersetzungsverfahren: Second Chance

- Kombination: Not Recently Used und FIFO
- ➔ verdränge älteste, im Inspektionsintervall nicht referenzierte Seite


: referenced-Flag

- Bei den Seiten, die auf die Kacheln A, B und E abgebildet werden, ist das referenced-Flag momentan gesetzt

Seitenersetzungsverfahren: Second Chance

- Kombination: Not Recently Used und FIFO
- ➔ verdränge älteste, im Inspektionsintervall nicht referenzierte Seite

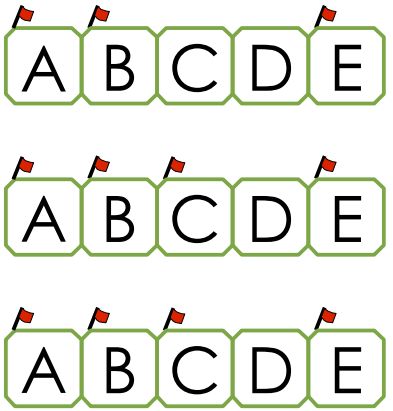

: referenced-Flag

C

- C wird referenziert

Seitenersetzungsverfahren: Second Chance

- Kombination: Not Recently Used und FIFO
- ➔ verdränge älteste, im Inspektionsintervall nicht referenzierte Seite

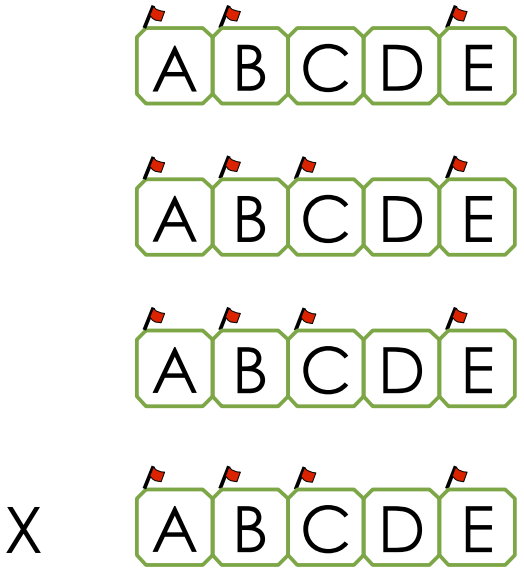



: referenced-Flag

- E wird referenziert

Seitenersetzungsverfahren: Second Chance

- Kombination: Not Recently Used und FIFO
- ➔ verdränge älteste, im Inspektionsintervall nicht referenzierte Seite

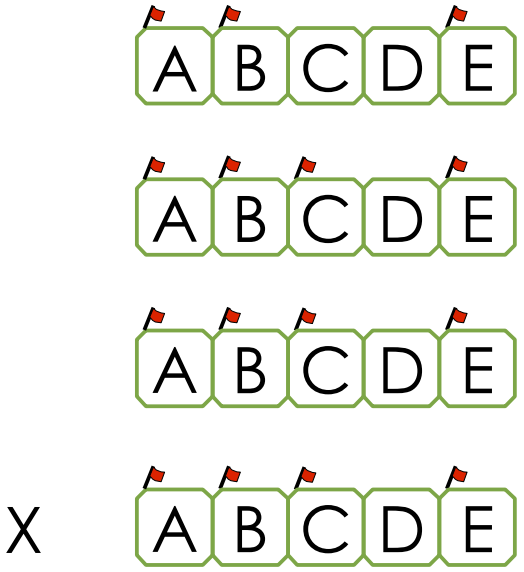



: referenced-Flag

- X wird referenziert → Seitenfehler
  → BS benötigt eine freie Kachel
  → Seitenersetzung

Seitenersetzungsverfahren: Second Chance

- Kombination: Not Recently Used und FIFO
- ➔ verdränge älteste, im Inspektionsintervall nicht referenzierte Seite

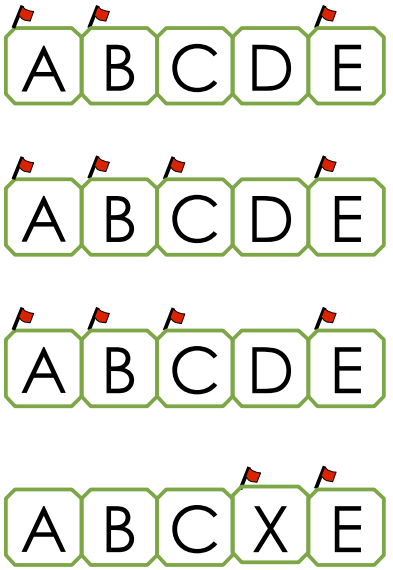


: referenced-Flag

- X wird referenziert → Seitenfehler
  → BS benötigt eine freie Kachel
  → Seitenersetzung
- von A beginnend sucht das BS eine Kachel ohne referenced-Flag, dabei werden die ref-Flags in den Seitentabellen gelöscht

Seitenersetzungsverfahren: Second Chance

- Kombination: Not Recently Used und FIFO
- ➔ verdränge älteste, im Inspektionsintervall nicht referenzierte Seite

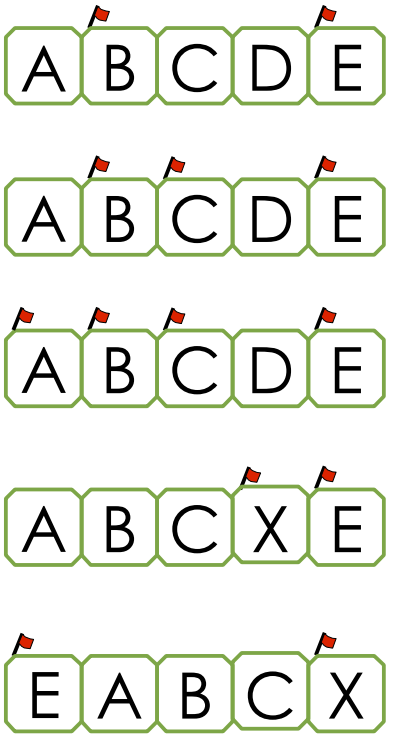


: referenced-Flag

- X wird referenziert → Seitenfehler
  → BS benötigt eine freie Kachel
  → Seitenersetzung
- von A beginnend sucht das BS eine Kachel ohne referenced-Flag, dabei werden die ref-Flags in den Seitentabellen gelöscht
- D wurde zuletzt nicht referenziert
  → D wird ersetzt

Seitenersetzungsverfahren: Second Chance

- Kombination: Not Recently Used und FIFO
- ➔ verdränge älteste, im Inspektionsintervall nicht referenzierte Seite

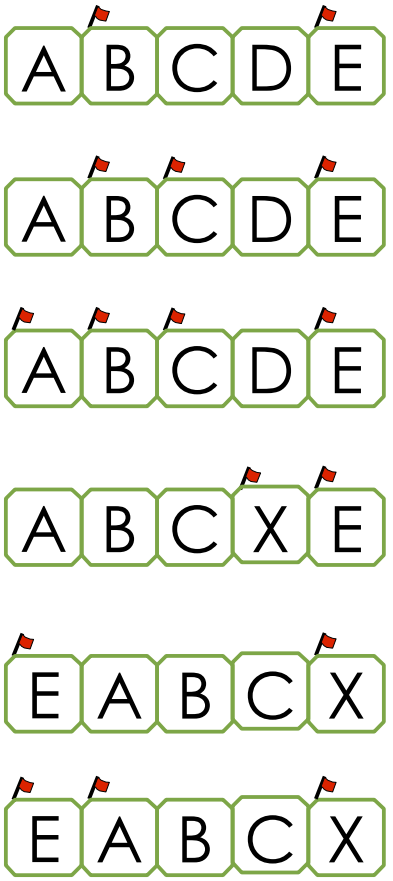


: referenced-Flag

- X wird referenziert → Seitenfehler
  → BS benötigt eine freie Kachel
  → Seitenersetzung
- von A beginnend sucht das BS eine Kachel ohne referenced-Flag, dabei werden die ref-Flags in den Seitentabellen gelöscht
- D wurde zuletzt nicht referenziert
  → D wird ersetzt
- FIFO-Liste wiederherstellen

Seitenersetzungsverfahren: Second Chance

- Kombination: Not Recently Used und FIFO
- ➔ verdränge älteste, im Inspektionsintervall nicht referenzierte Seite



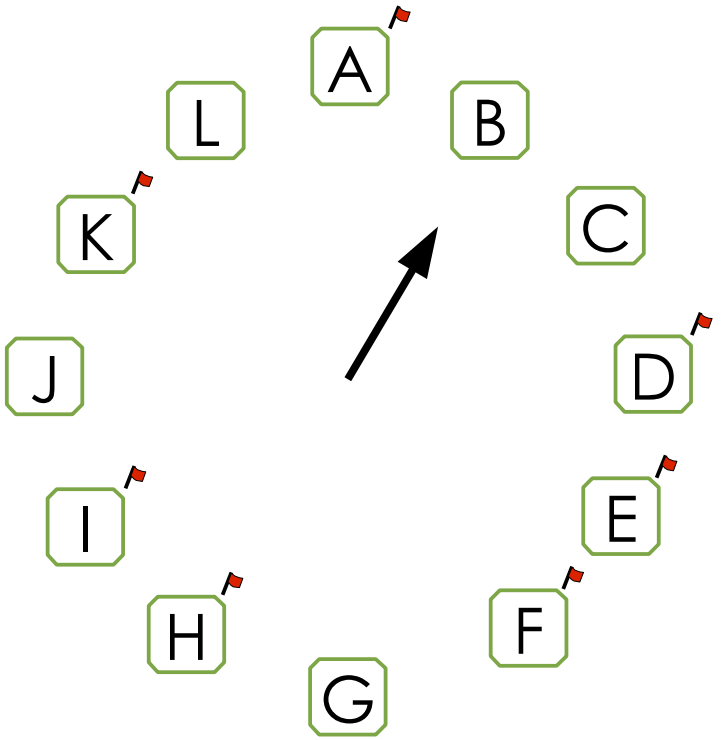
: referenced-Flag

- A wird referenziert → „2. Chance“

Seitenersetzungsverfahren: Clock-Algorithmus

```
...
if (!Kachel[i].referenced) {

  //Seite in Kachel Nummer i
  //verdrängen
} else {
  Kachel[i].referenced = 0;
  //Flag löschen
  i = (i + 1) % FRAME_COUNT;
  //Zeiger weiterdrehen
}
...
```

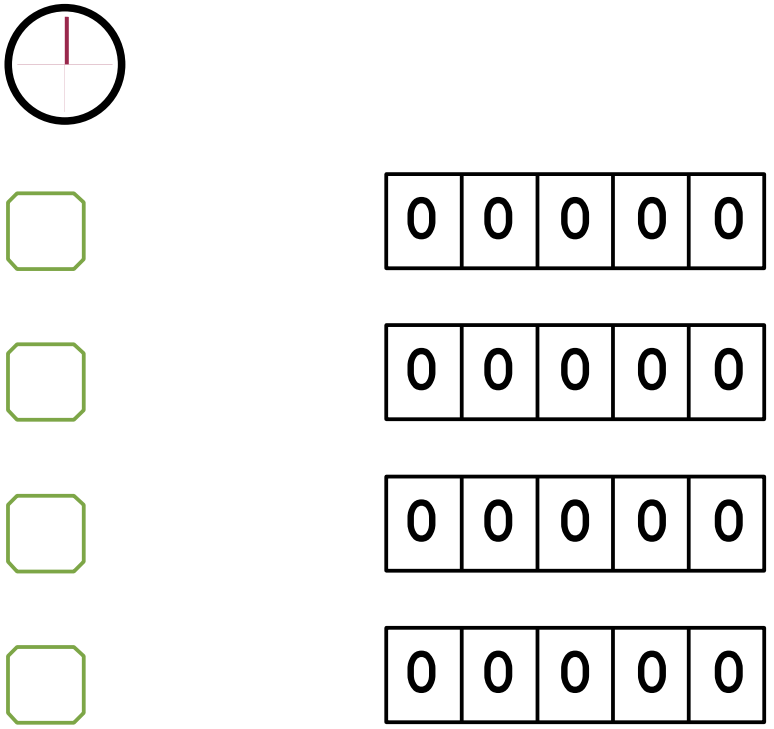


Seitenersetzungsverfahren: Aging

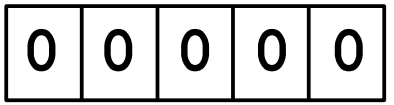
- Kachel mit ältestem Inhalt wird verdrängt
- bessere Näherung: statt „0“ „1“ → feineres Altersraster
- Analogie: Geburtsjahr
  - 1982
  - 1985
  - 1986

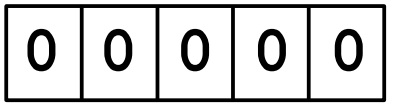
  niedrige Zahl → hohes Alter

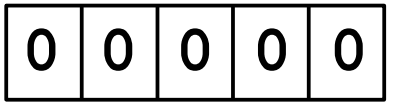
Seitenersetzungsverfahren: Aging

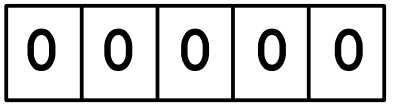


Seitenersetzungsverfahren: Aging

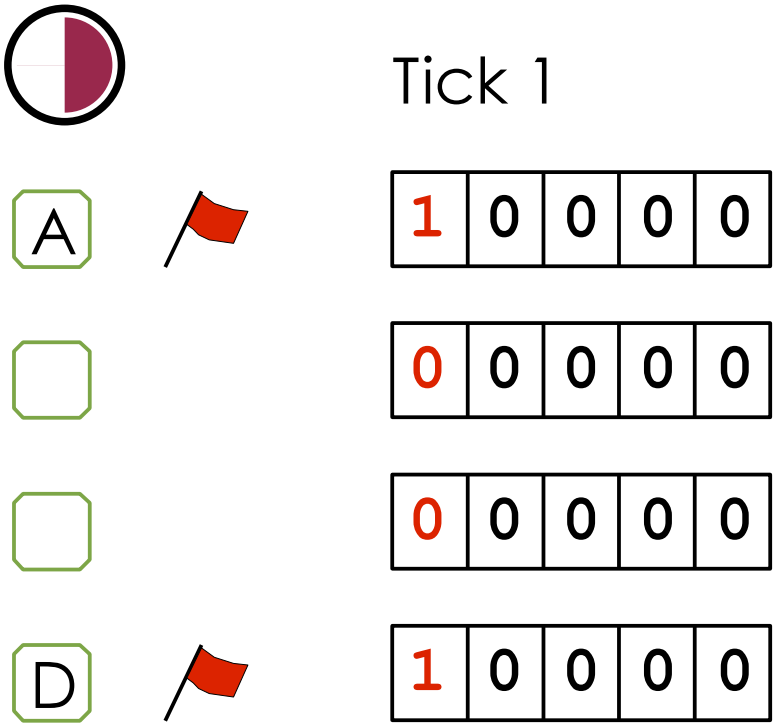








Seitenersetzungsverfahren: Aging



Seitenersetzungsverfahren: Aging

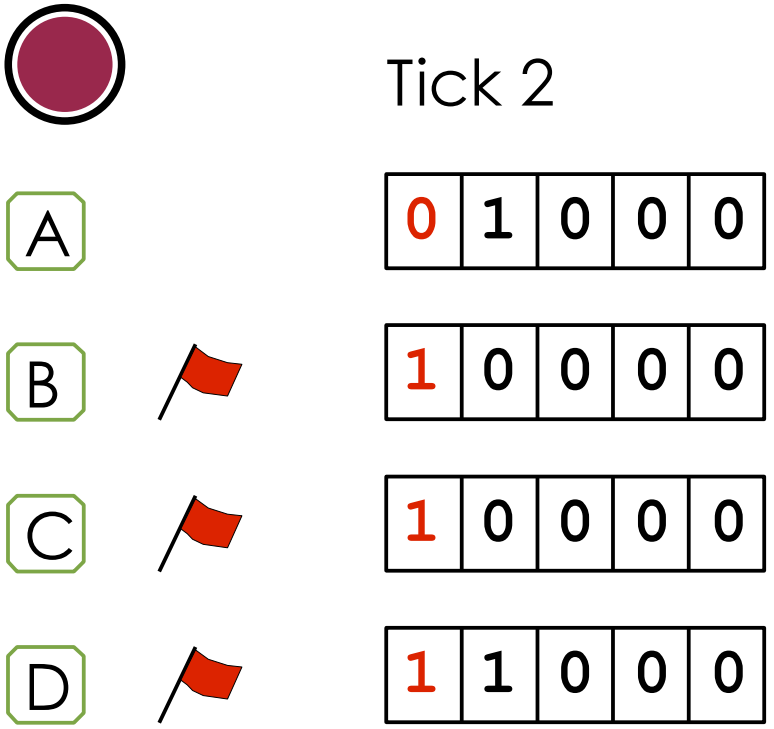
1	0	0	0	0

0	0	0	0	0

0	0	0	0	0

1	0	0	0	0

Seitenersetzungsverfahren: Aging



Seitenersetzungsverfahren: Aging

0	1	0	0	0

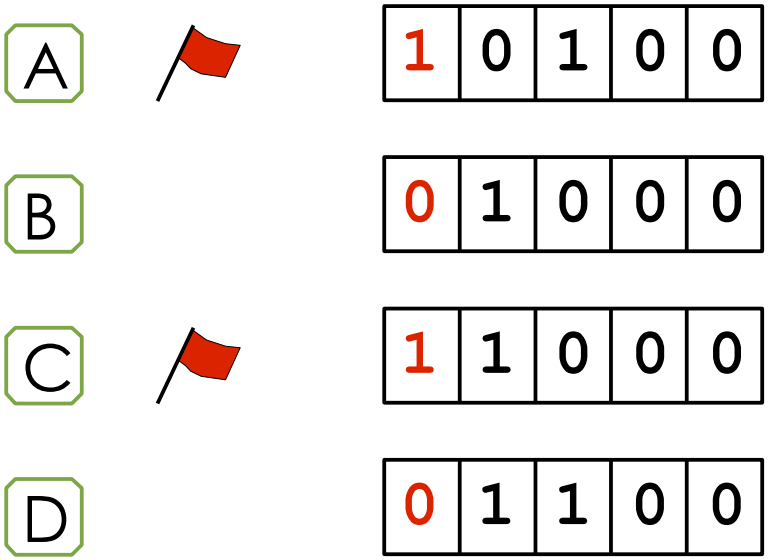
1	0	0	0	0

1	0	0	0	0

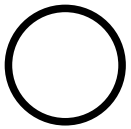
1	1	0	0	0

Seitenersetzungsverfahren: Aging

Tick 3



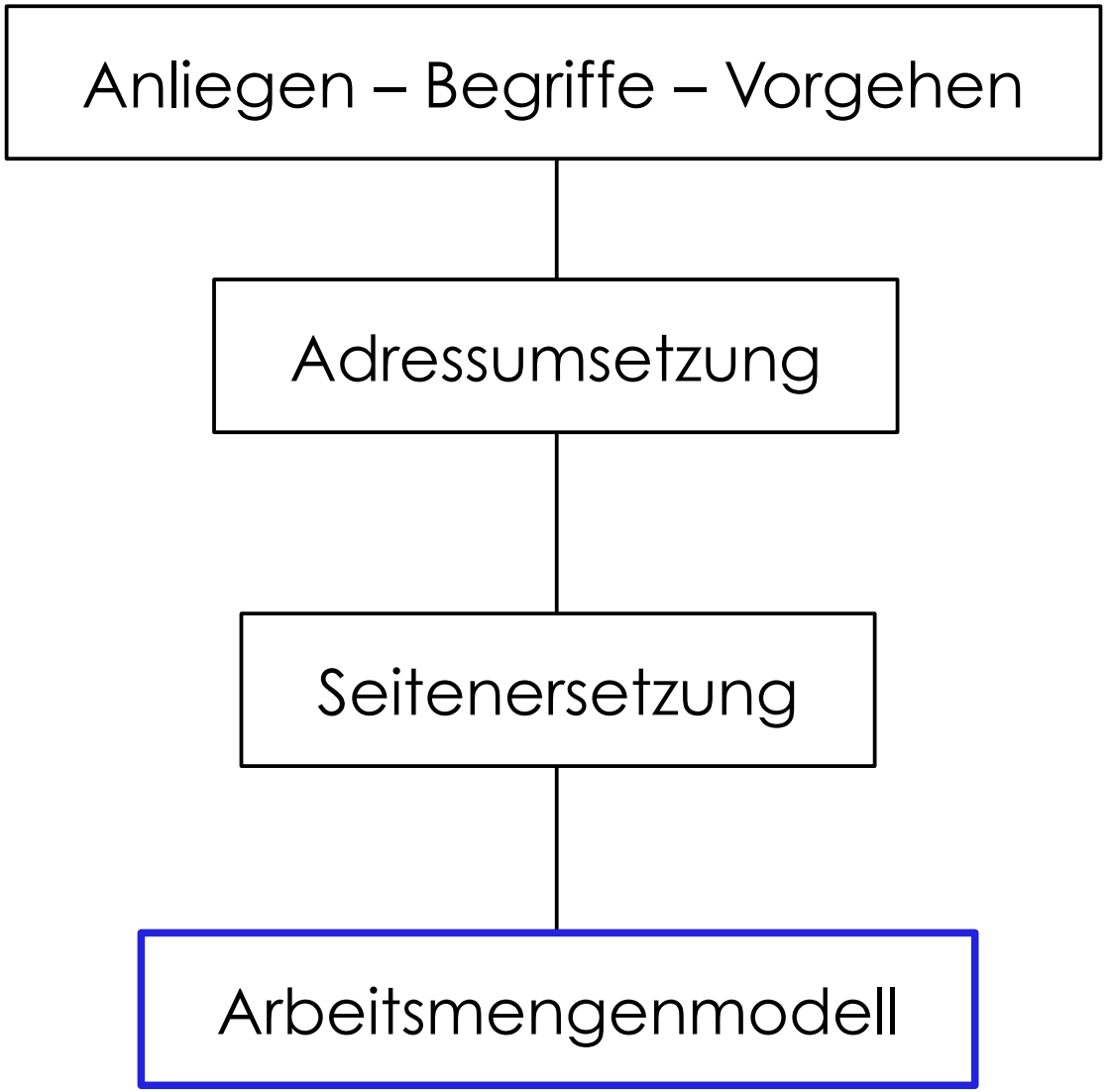
Seitenersetzungsverfahren: Aging



	Seitenersetzung	Age
A	1 0 1 0 0	20
B	0 1 0 0 0	8
C	1 1 0 0 0	24
D	0 1 1 0 0	12

Seitenersetzungsverfahren: Aging

Wegweiser: Virtueller Speicher



Das Arbeitsmengenmodell (Working Set)

Problem

→ Seitenfehlerhäufung bei Prozessaktivierungen

Grundgedanken

- Lokalitätsprinzip
- Betrachtung der Seitenreferenzfolge eines Prozesses durch ein „Fenster“ der Länge *T* (sog. Arbeitsmengen-Parameter)

Das Arbeitsmengenmodell (Working Set)

- Arbeitsmenge $w(t, T)$ zum Zeitpunkt t bzgl. T:
  Menge der im Intervall $[t - T, t]$ referenzierten Seiten

Erfahrung

- max ($Z(T)$): theoretisch mögliche Maximalzahl von Speicherzugriffen in T Zeiteinheiten

- $|w(t, T)| \ll \max(Z(T))$

Das Arbeitsmengenmodell (Working Set)

Vorgehen

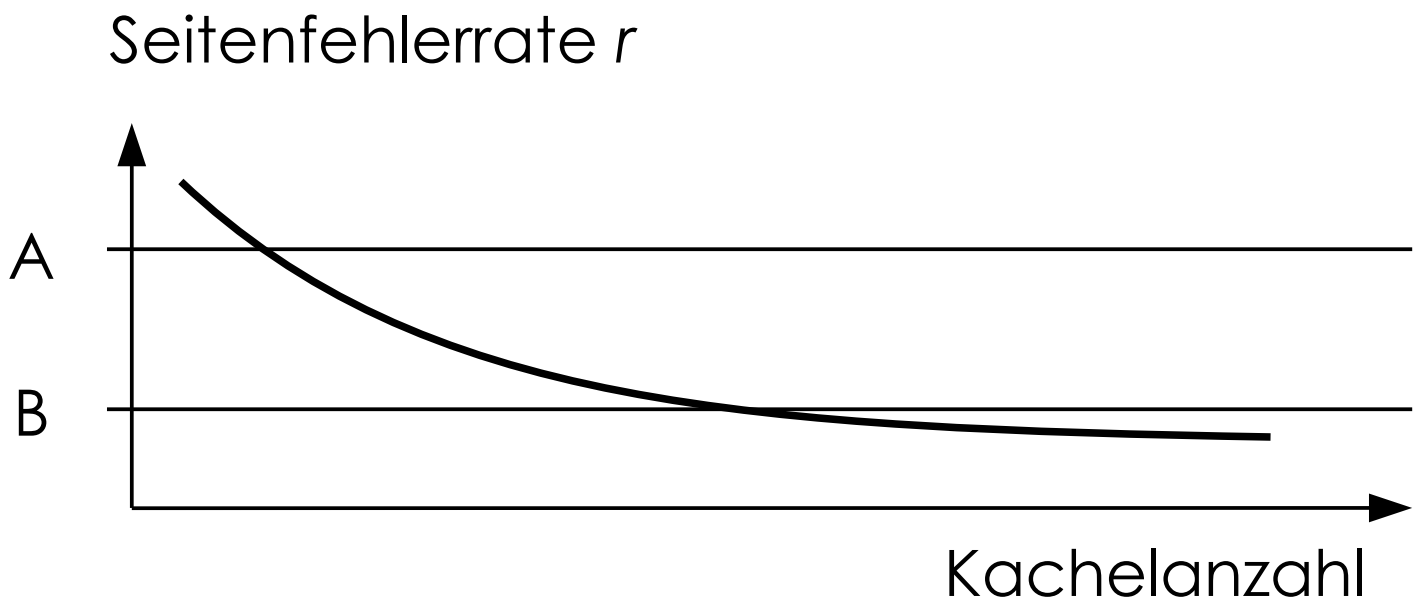
- Bestimmung der Arbeitsmenge bei jeder Seitenreferenz
- für aktive und bereite Prozesse muss sich genau die Arbeitsmenge im Speicher befinden
- Swapping, falls Rahmen fehlen
  → Prepaging bei Wiedereinlagerung

Probleme

- Fenstergröße
- Implementierung (prozesseigene Uhr!)
- „Seitenflattern“ (Thrashing)
- globale/lokale Ersetzung

Seitenfehlerrate

- Begrenzung der Anzahl von Prozessen und globaler Seitenaustausch
- Ermittlung der Kachelanzahl:
  abhängig von Seitenfehlerrate



Prozesslokaler vs. prozessglobaler Seitenaustausch

- Seitenaustausch global für alle Prozesse
  Problem:
  „thrashing“ falls zu viele Prozesse

- Seitenaustausch prozesslokal
  (Zuteilung von Hauptspeicher an Prozesse)

➔ Problem:
  Ermittlung der Anzahl benötigter Kacheln

	Age
A0	8
A1	3
A2	4
A3	15
A4	2
A5	6
B0	3
B1	9
B2	5
C0	12
C1	5
C2	13
C3	6
C4	3
C5	7
C6	4

Weiterer Gesichtspunkt: Wahl der Seitengröße

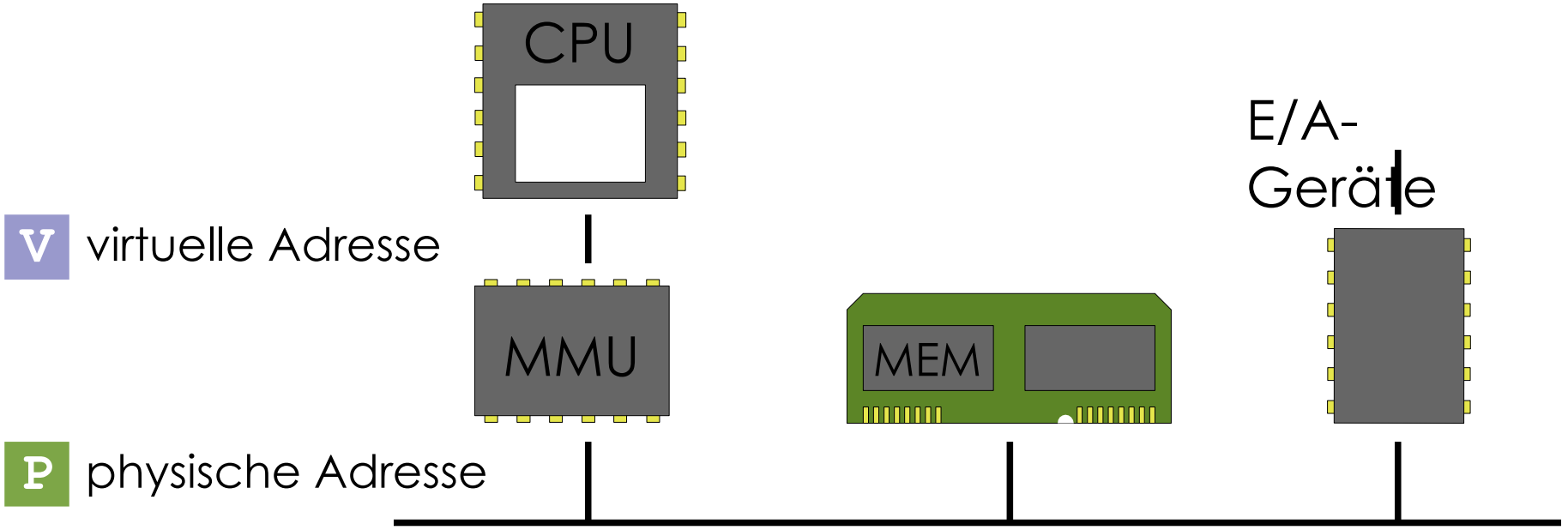
➔ Betriebssystem kann Seitengröße größer wählen als durch Hardware vorgegeben

- HW mit mehreren Seitengrößen (Itanium, ARM)

Entscheidungsgesichtspunkte

- Zeit für Seiteneinlagerung
- Verschnitt durch nicht voll genutzte Seiten
- Lokalität von Zugriffen
- TLB-Ausnutzung besser bei größeren Seiten
- Größe von Verwaltungsdatenstrukturen

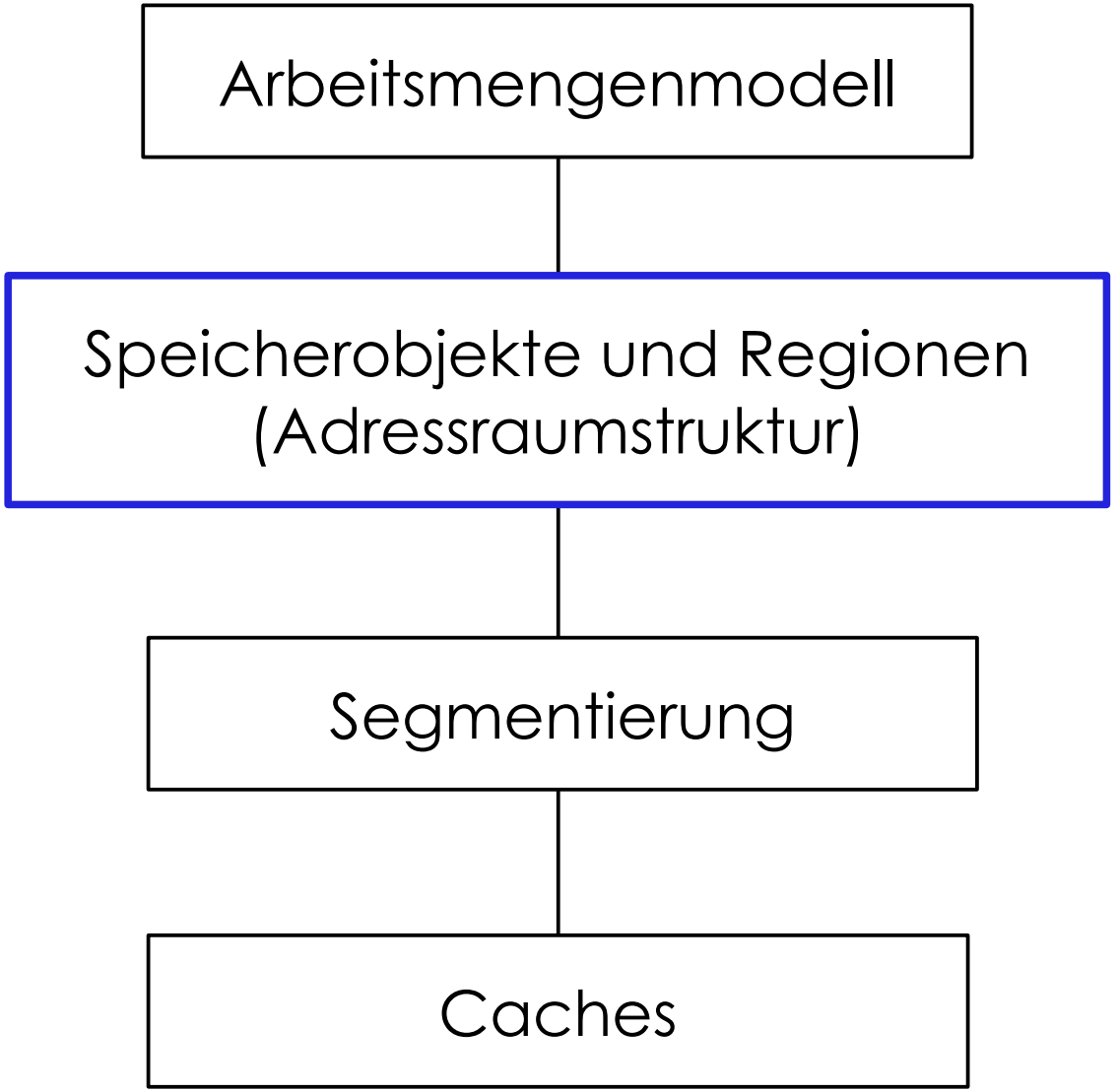
Residente Seiten



E/A – Einbindung

- häufige, aber schlechte Lösung:
  Kopieren in/aus residentem Zwischenpuffer
- ➔ besser: Residentsetzen beliebiger Adressbereiche,
  „Pinning“ - Umrechnen virtuell → real durch Treiber
- weitere Anwendung: zeitkritische Applikationen

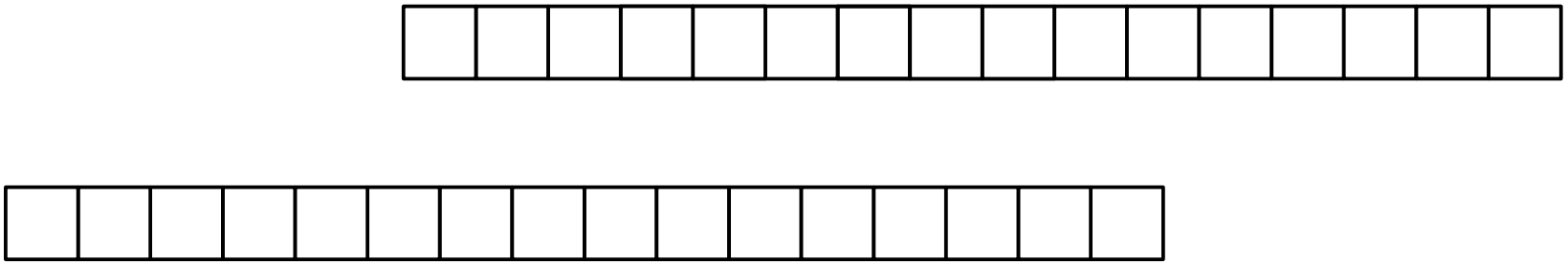
Wegweiser


Arbeitsmengenmodell
Speicherobjekte und Regionen
(Adressraumstruktur)
Segmentierung
Caches

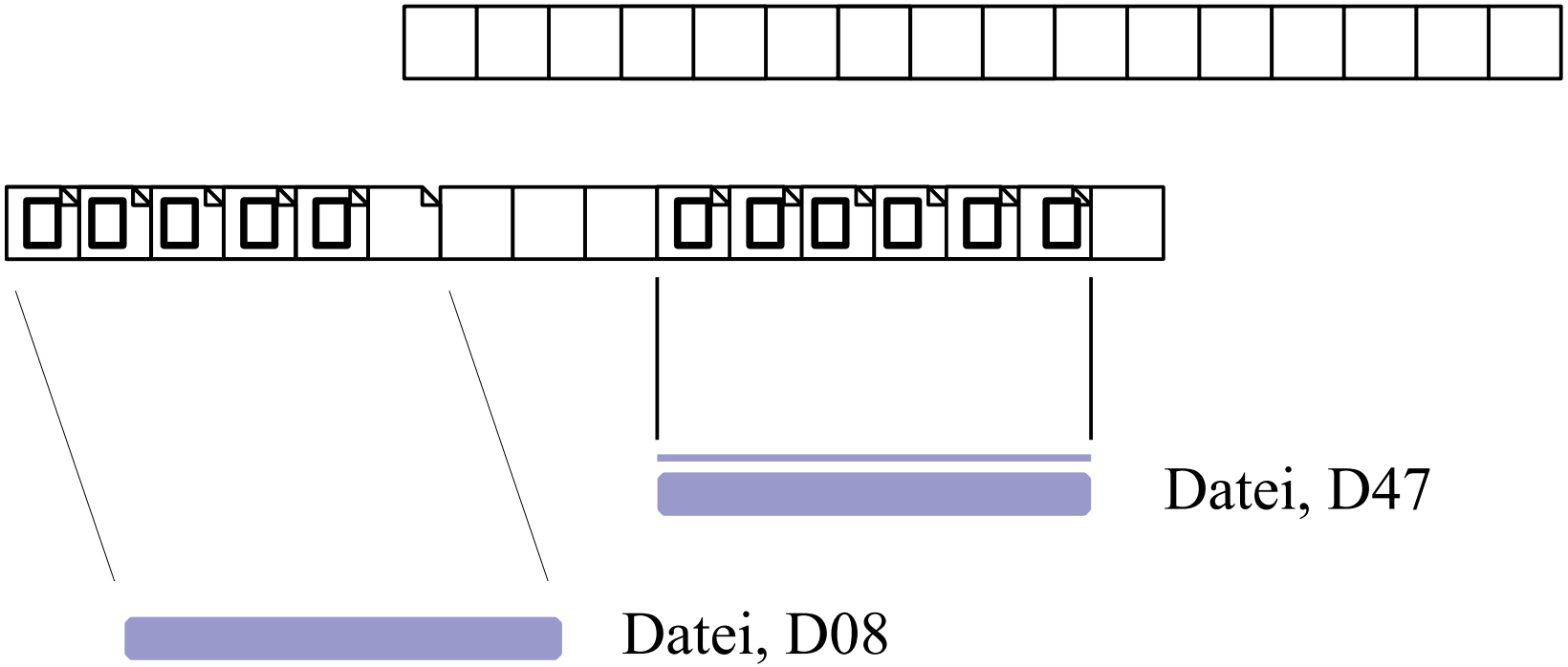
Speicherobjekte

- Text- (Code-), Daten-, ... -Segment eines Prozesses
- Dateien (nächstes Kapitel)
- Bildwiederholspeicher
- BS-Datenstrukturen
  - Thread Control Blocks
  - Seitentabellen
  - ...
- Netzwerkobjekte

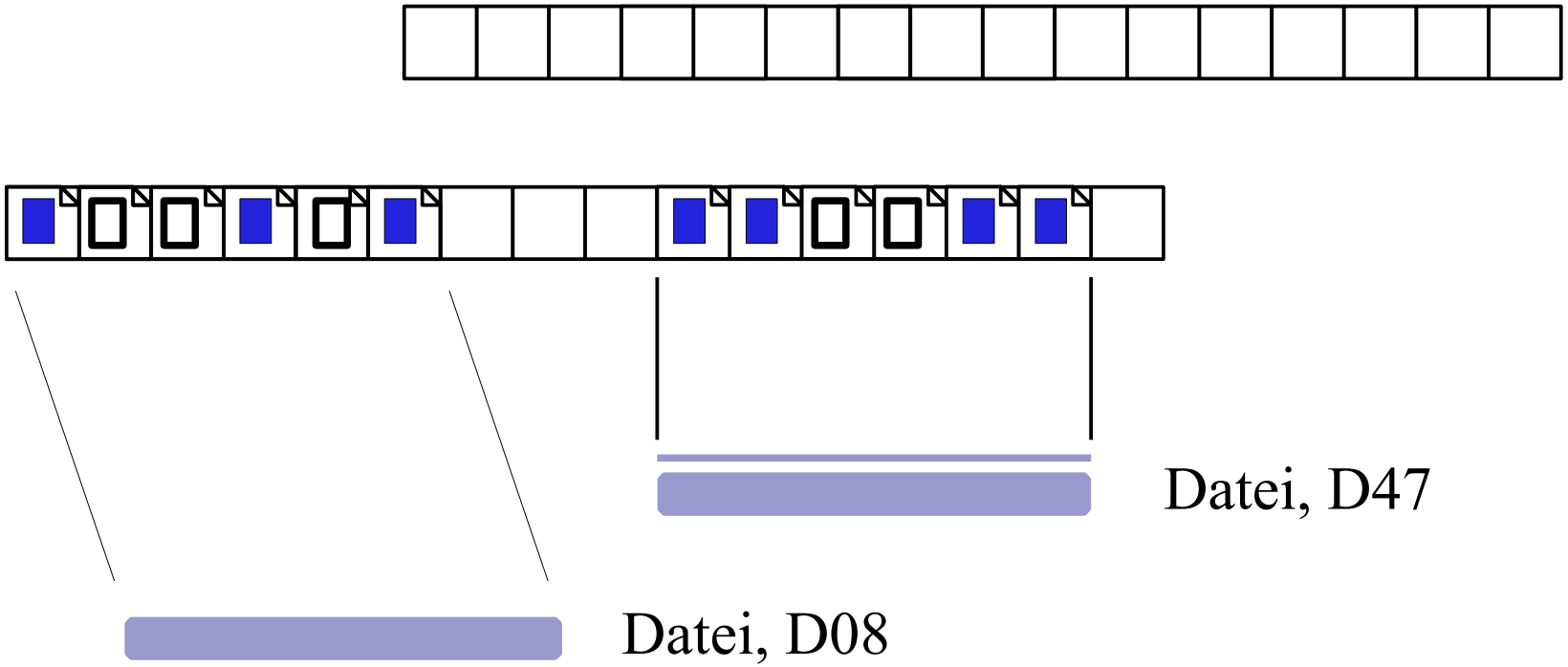
Speicherobjekte und Regionen



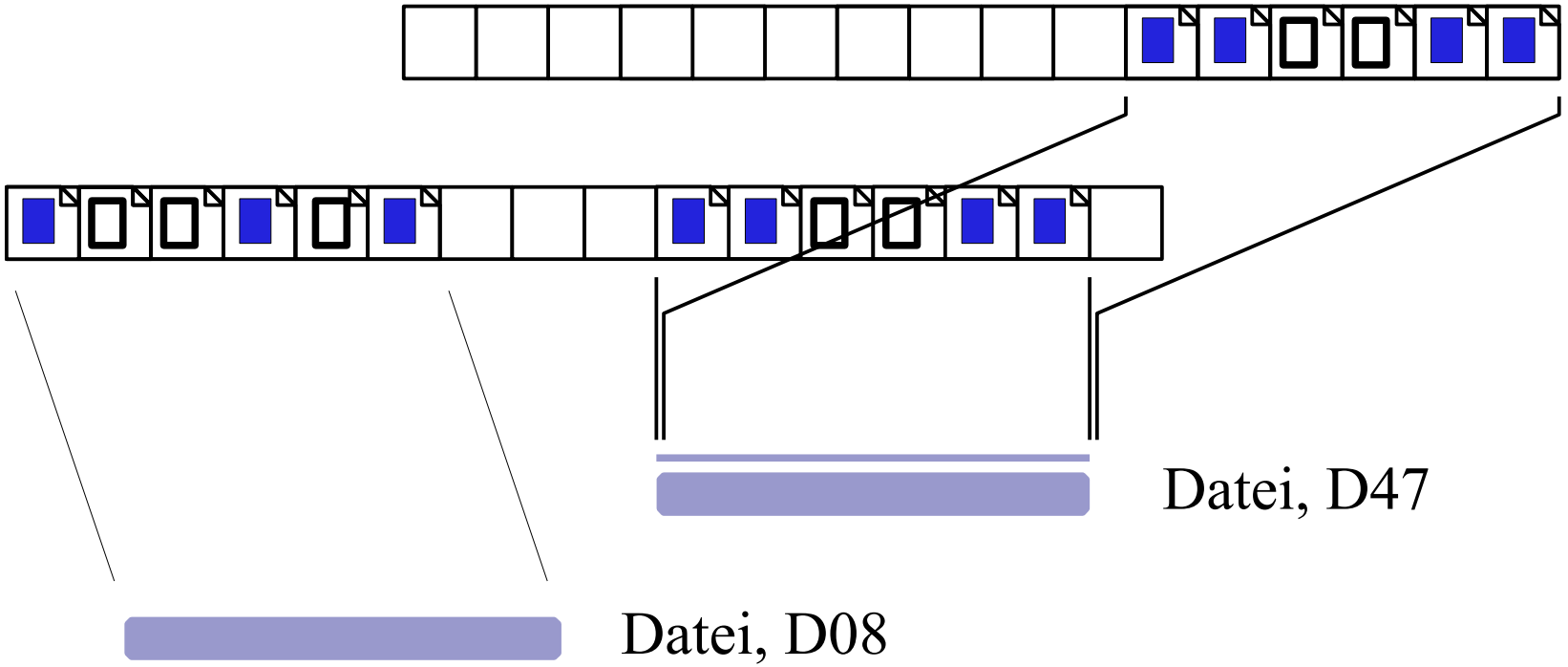
Speicherobjekte und Regionen



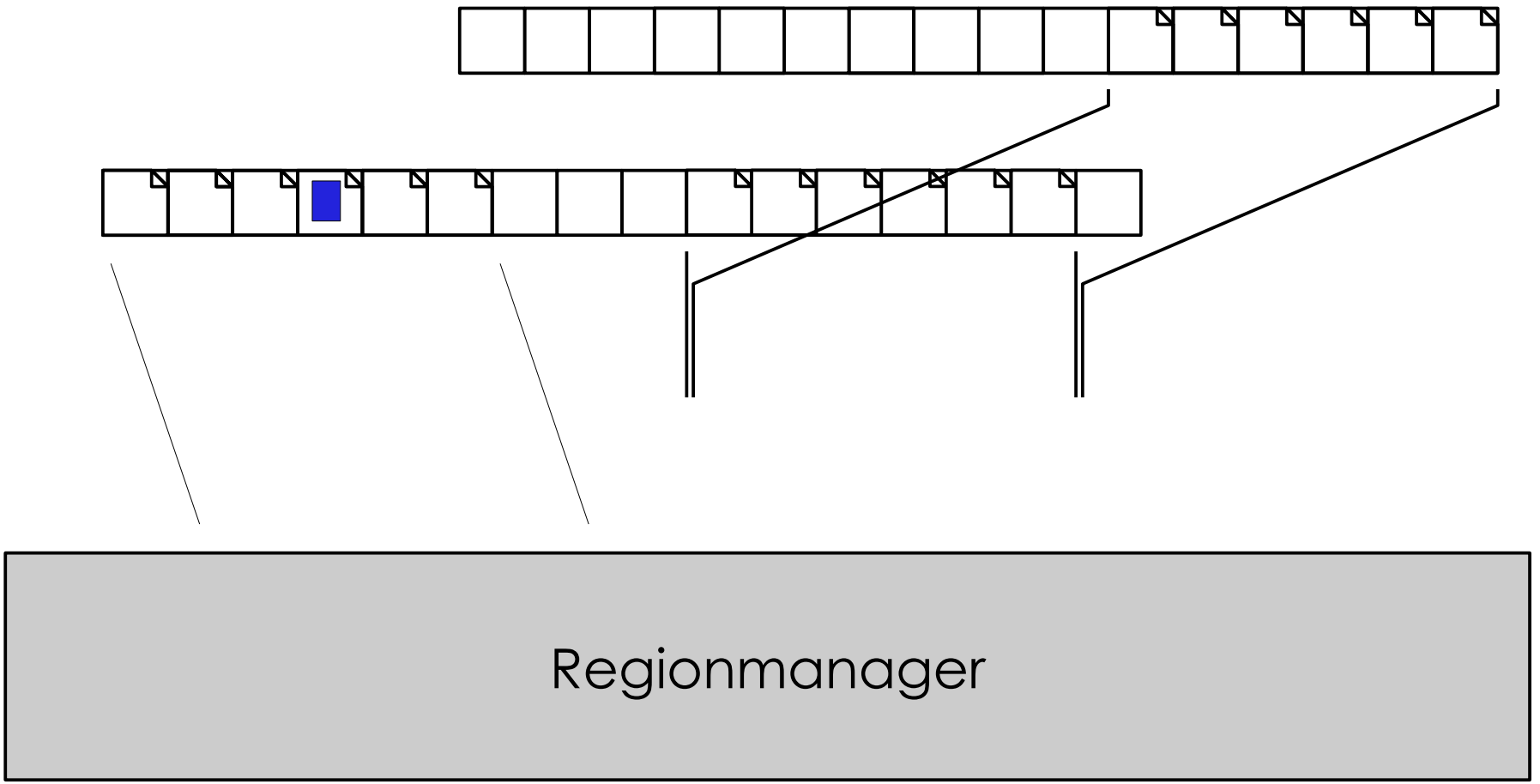
Speicherobjekte und Regionen



Speicherobjekte und Regionen



Speicherobjekte und Regionen



Aufbau der Adressraumstruktur

- Regionen im Adressraum:

  explizites oder implizites Einbinden („Mappen“) von Speicherobjekten in den virtuellen Adressraum eines Prozesses

- Operationen:

```
map(Adresse, Länge, Speicherobjekt,
                Zugriffsrechte);

lookup(Adresse, Zugriffart) ->
```

  - **Speicherobjekt, Seitennummer innerhalb Obkekt**
  - **oder nicht definiert**
  - **oder Rechteverletzung**

Aufbau der Adressraumstruktur

- z. B.

```
map( ... , ... , Datei, R );

map( ... , ... ,
     Bildwiederholspeicher, RW);
```

- implizit: bei Prozesserzeugung werden Programm-, Keller-, Daten-Regionen gebildet (in Unix: „Segmente“)

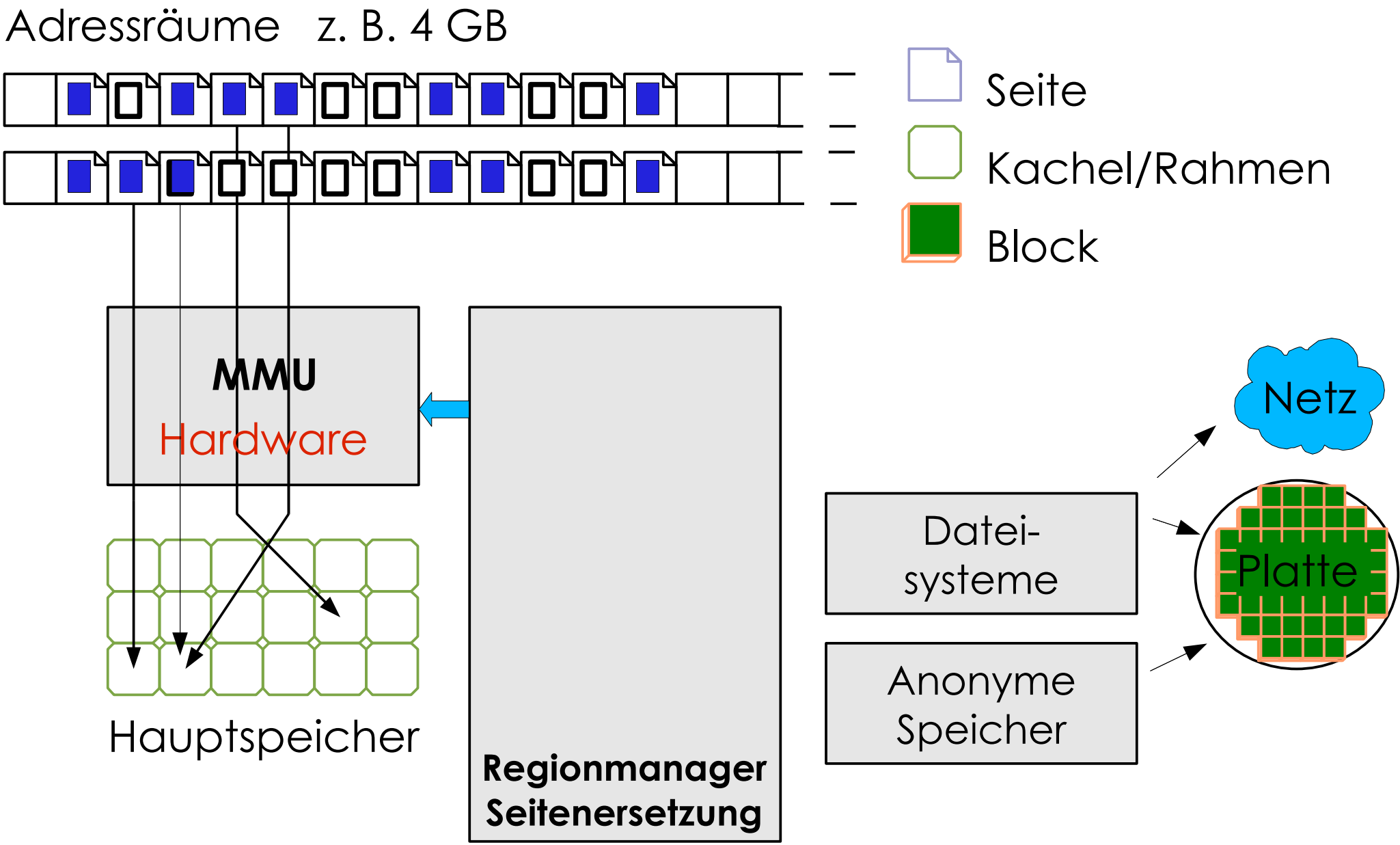
Regionmanager

- Verwaltung der Adressraum-Struktur
  - ➔ repräsentiert z. B. als verkettete Liste

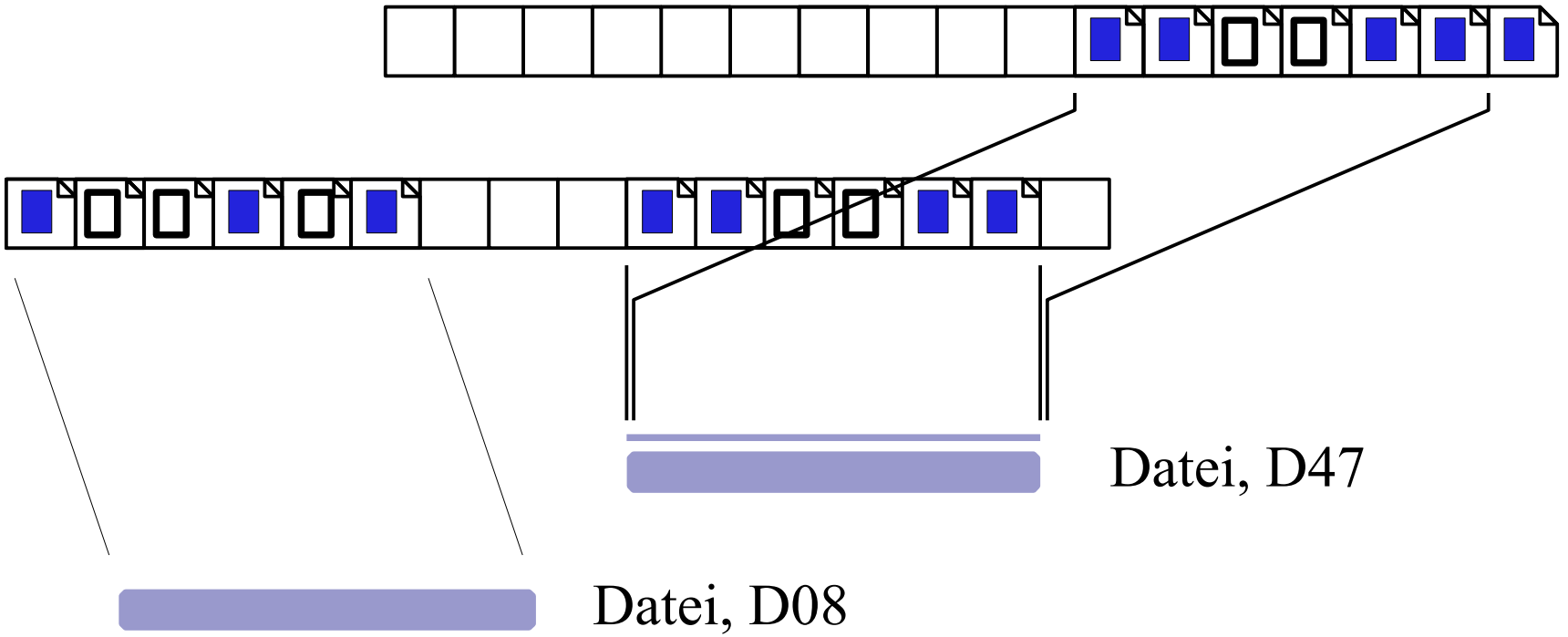
- Aufgabe bei Seitenfehler: Bestimmen von
  - Fehlerart
  - Speicherobjekt
  - Speicherseite

- Fehlerarten
  - Zugriff auf ungültigen Bereich (z.B. Pointerfehler)
  - Rechteverletzung (z.B. Schreiben in Code-Bereich)
  - Copy On Write Fault
  - sonst: Seite nicht in MMU eingetragen

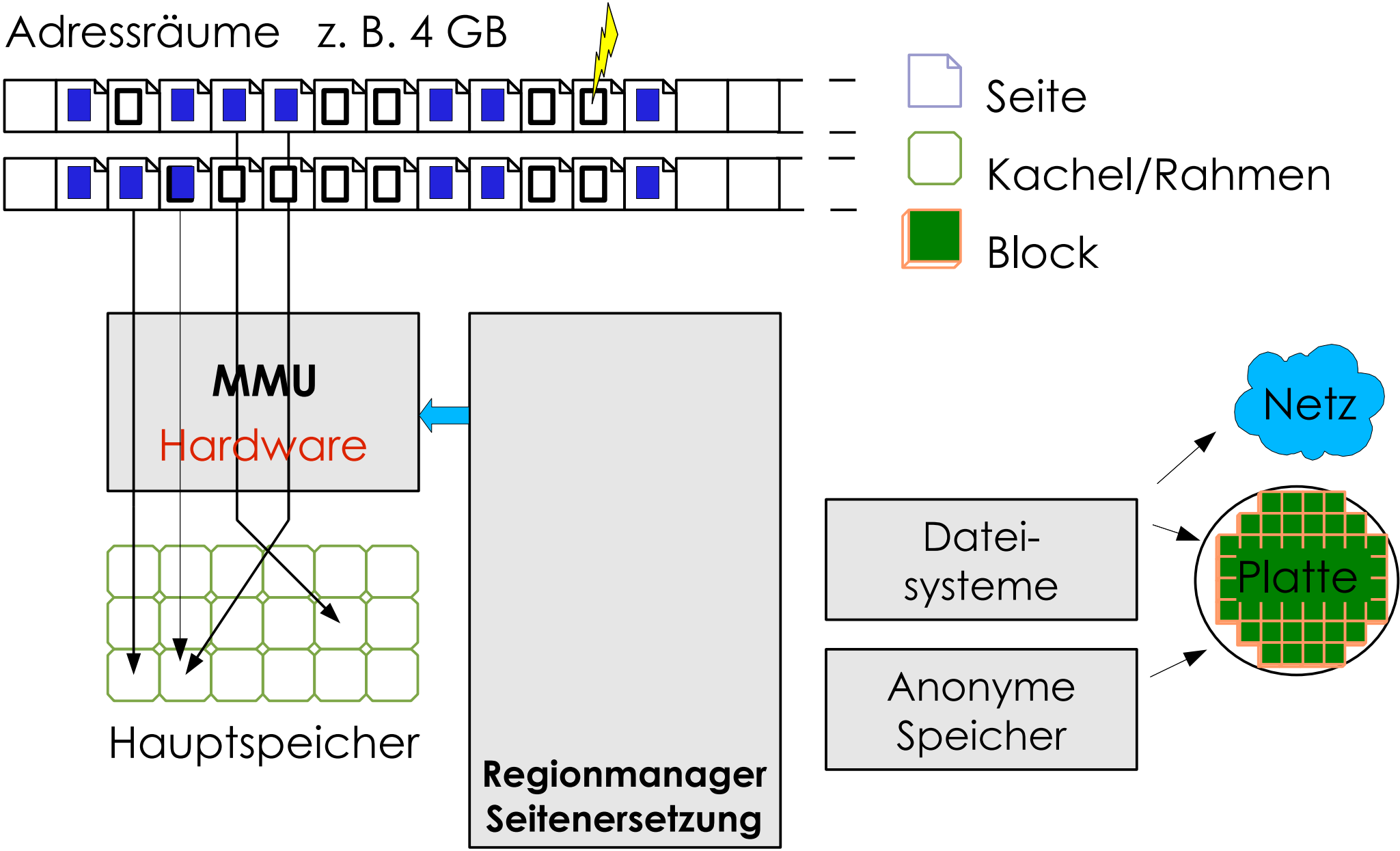
Das Zusammenspiel



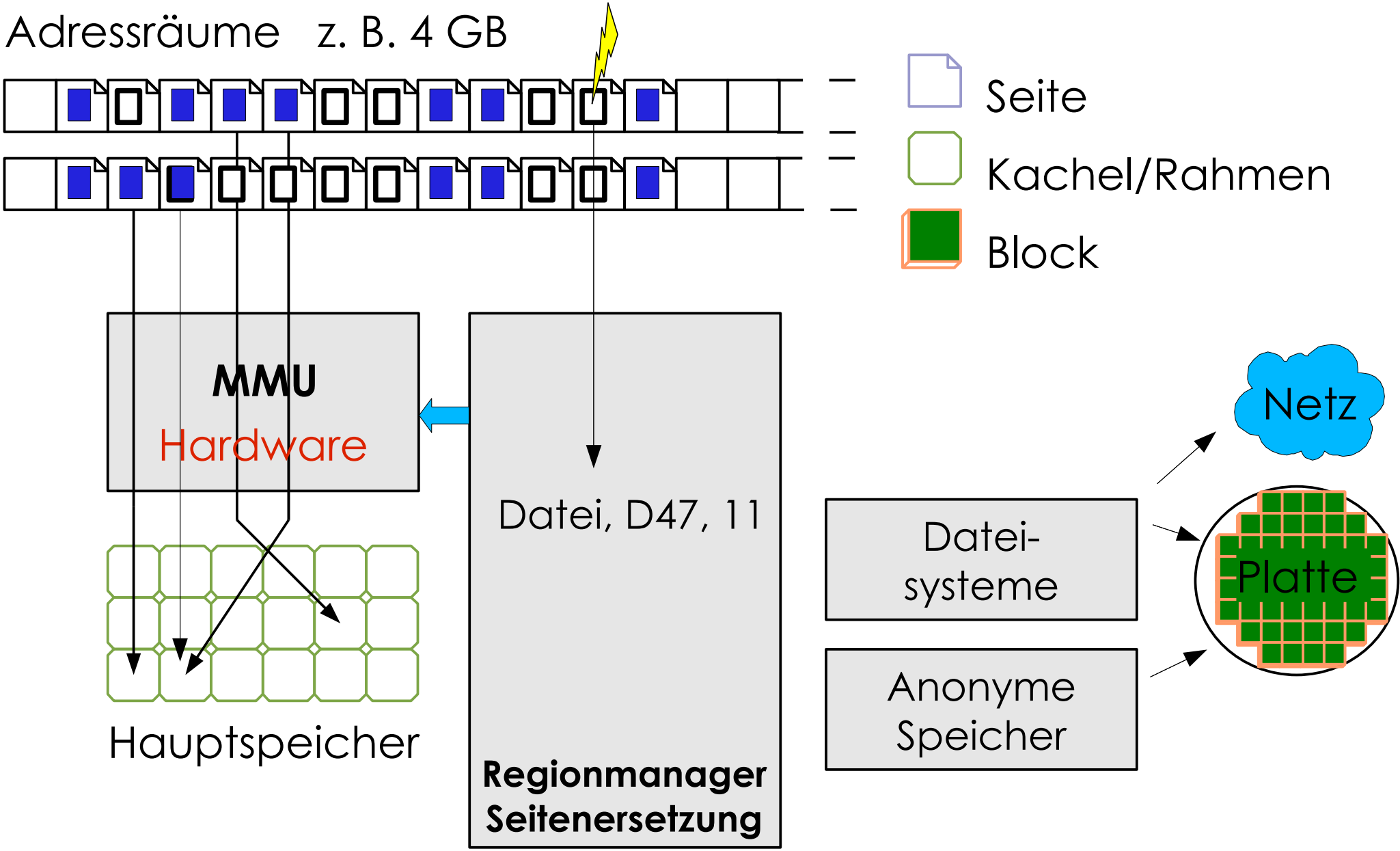
Speicherobjekte und Regionen



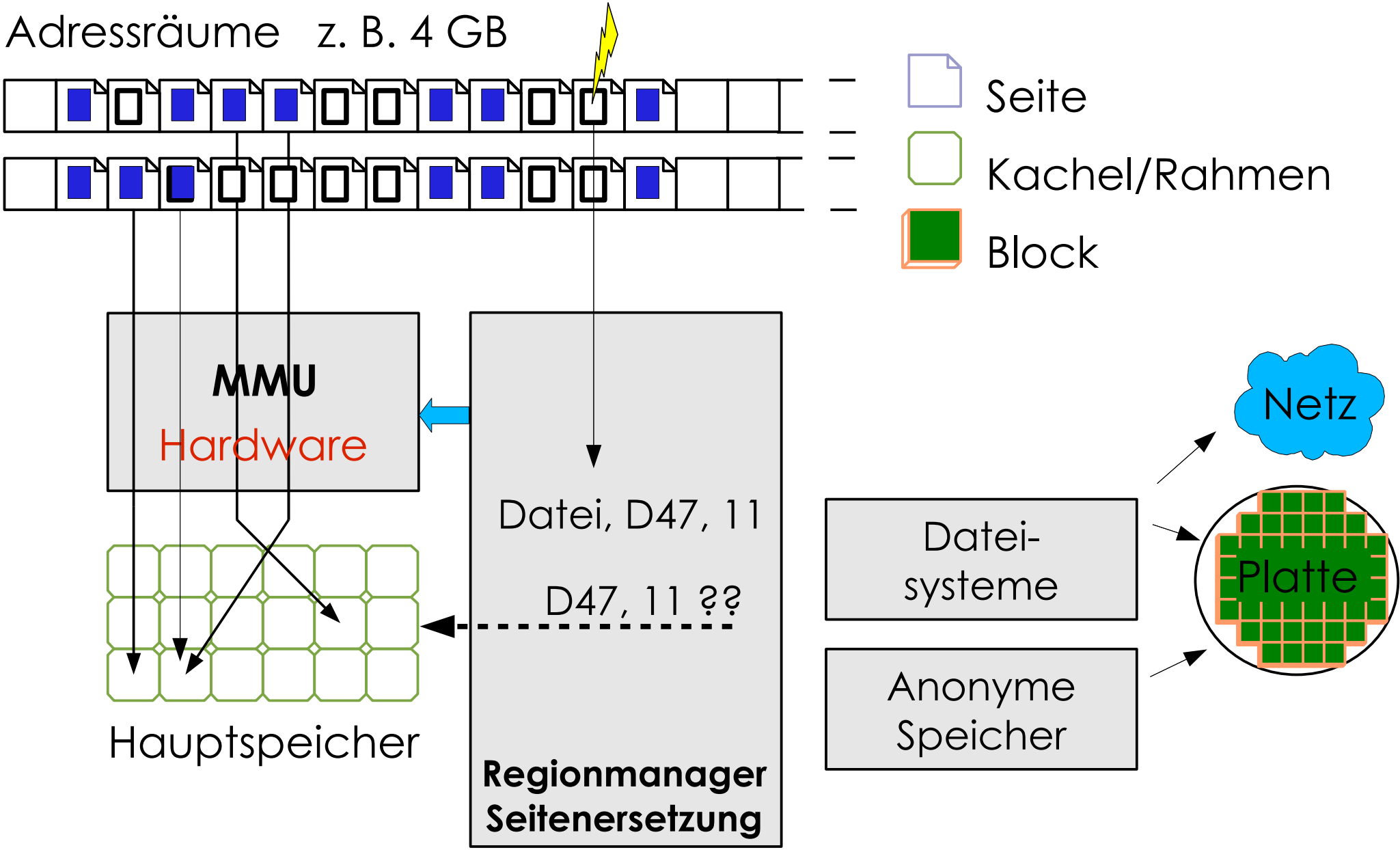
Das Zusammenspiel



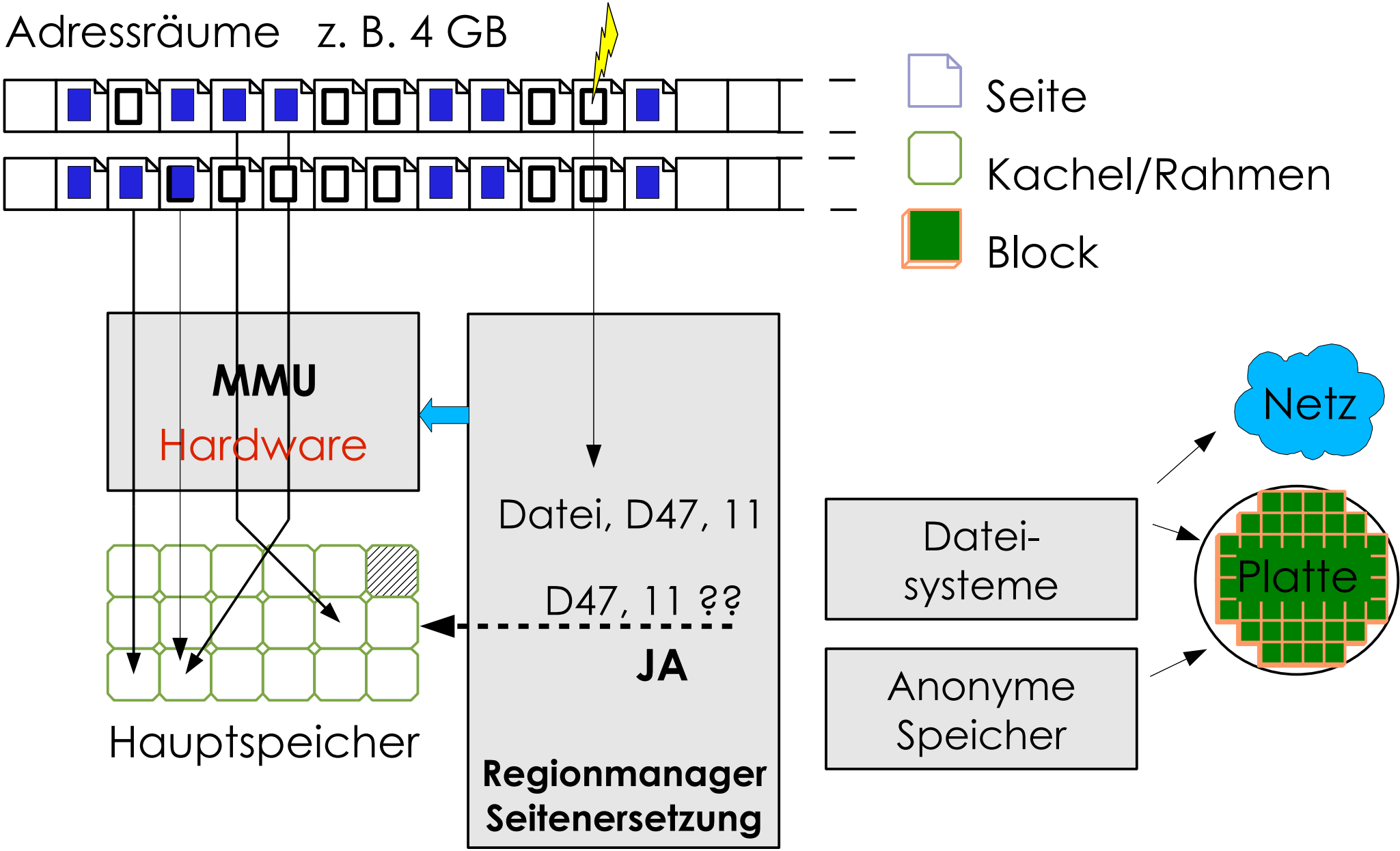
Das Zusammenspiel



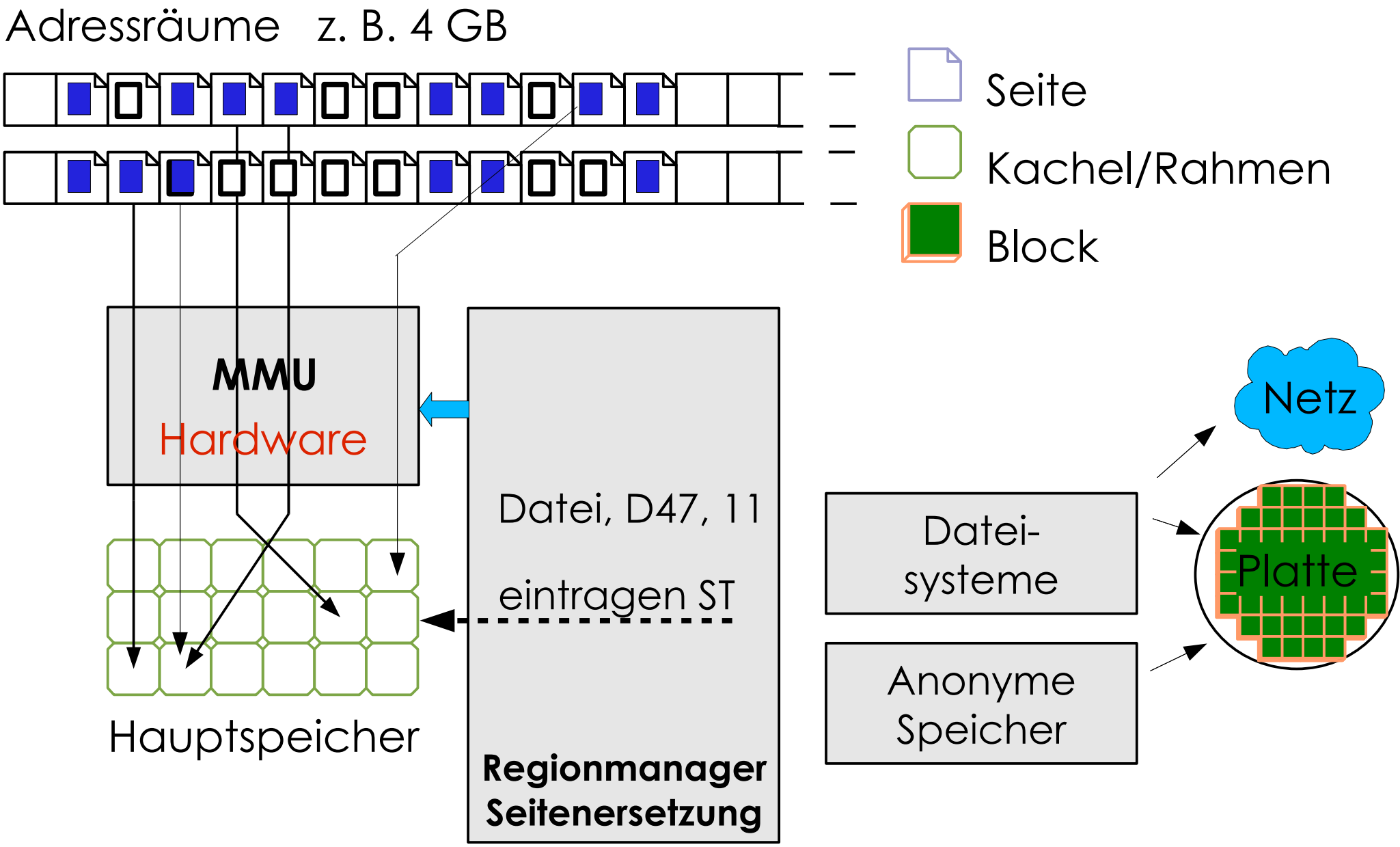
Das Zusammenspiel



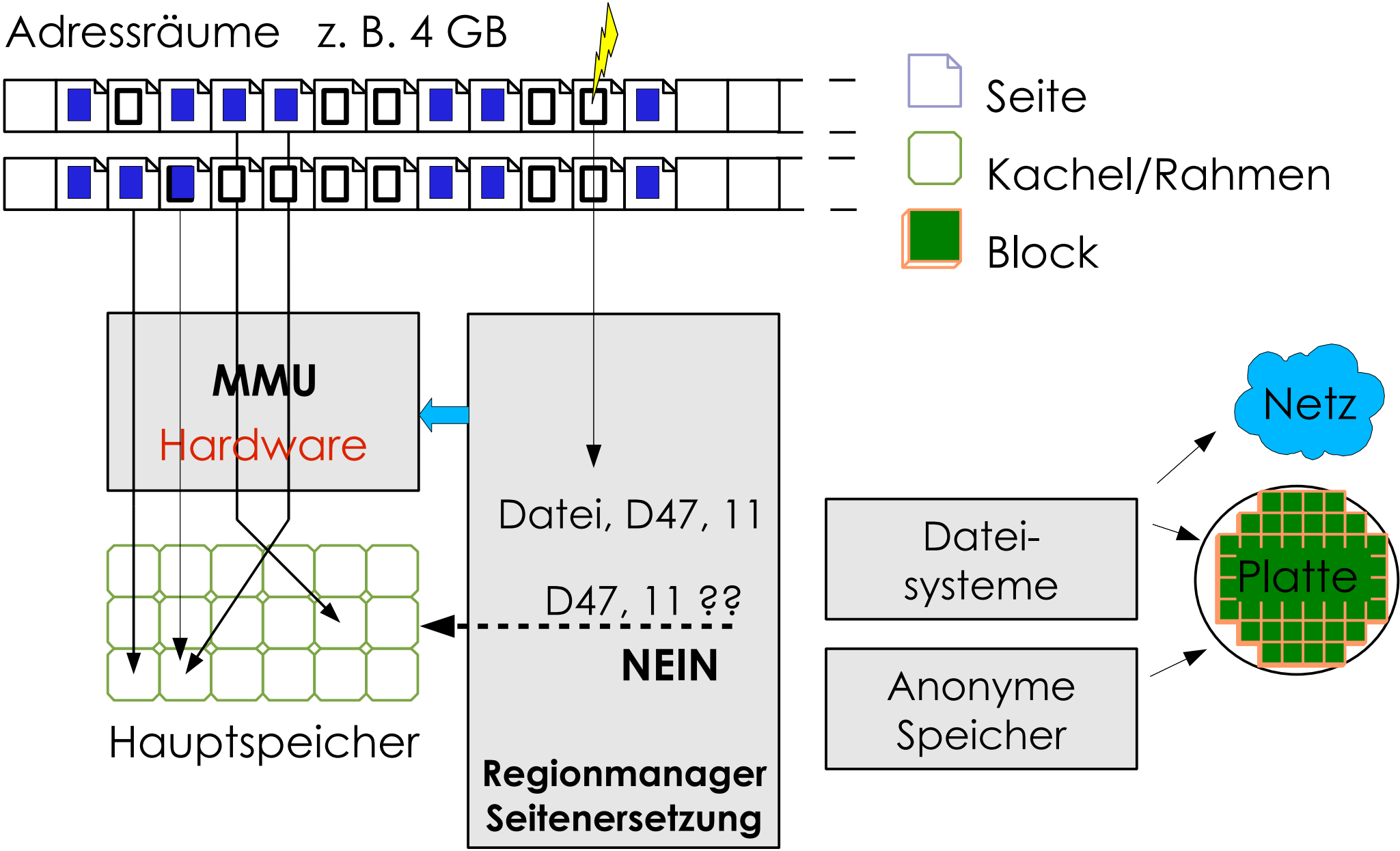
Das Zusammenspiel



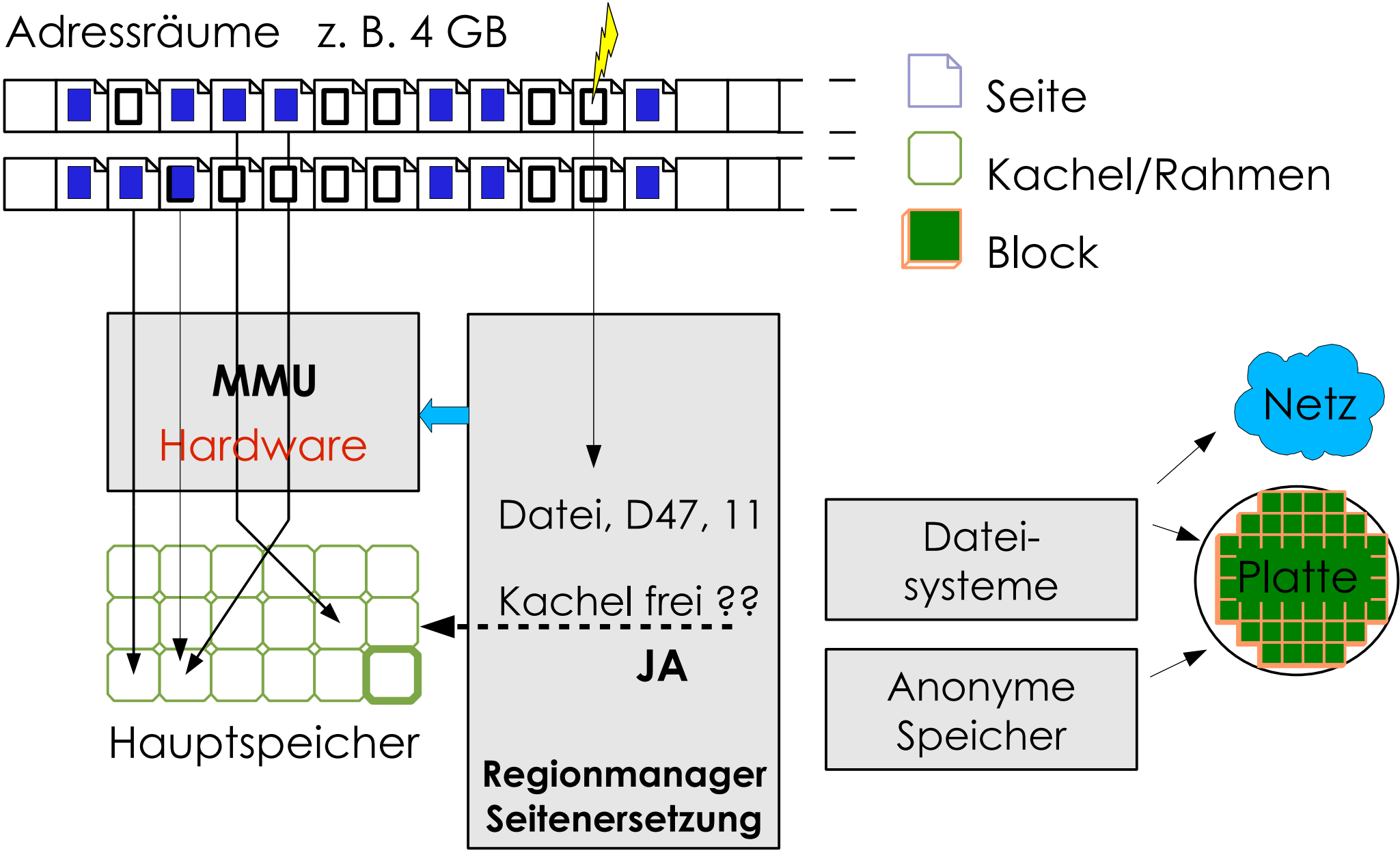
Das Zusammenspiel



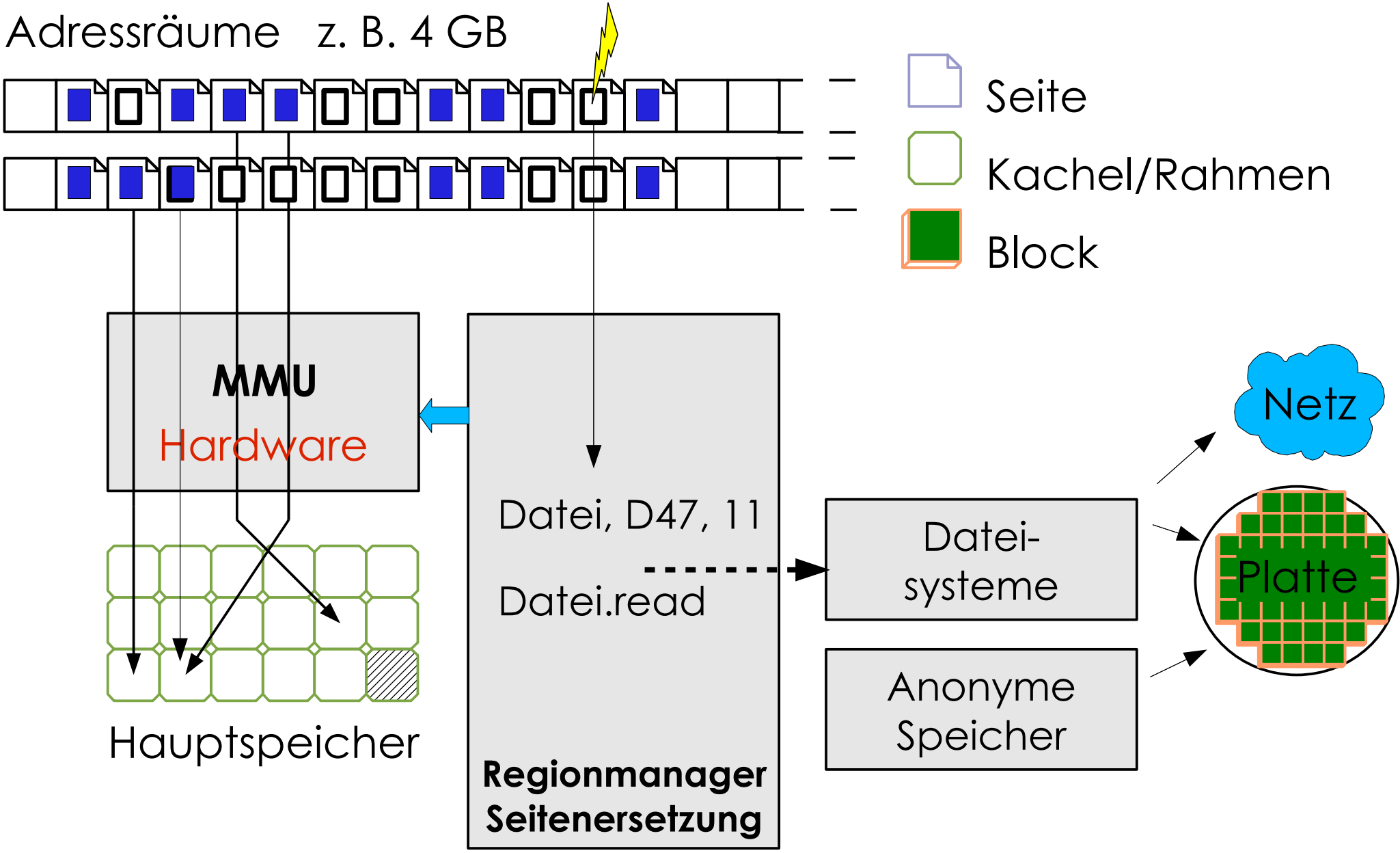
Das Zusammenspiel



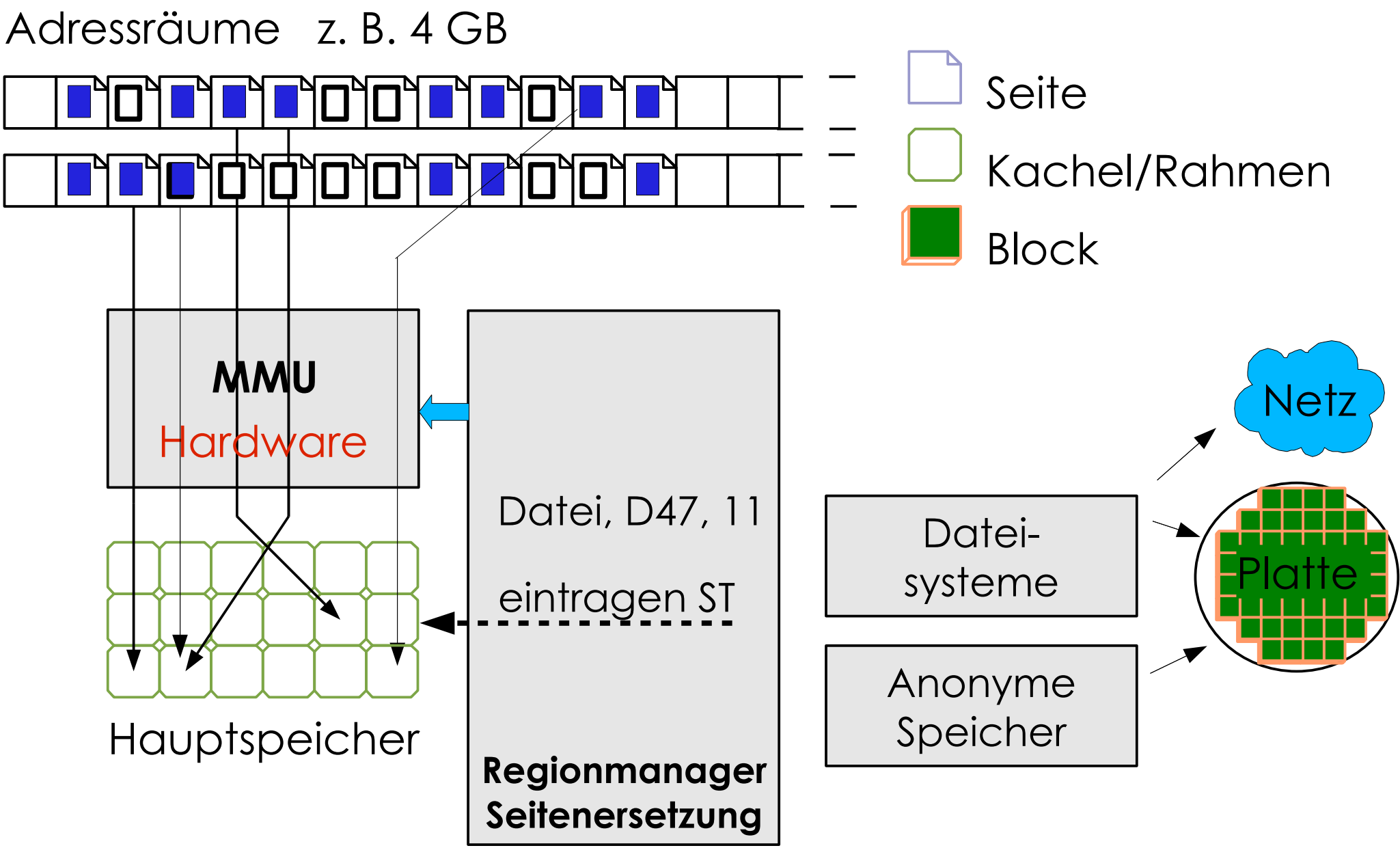
Das Zusammenspiel



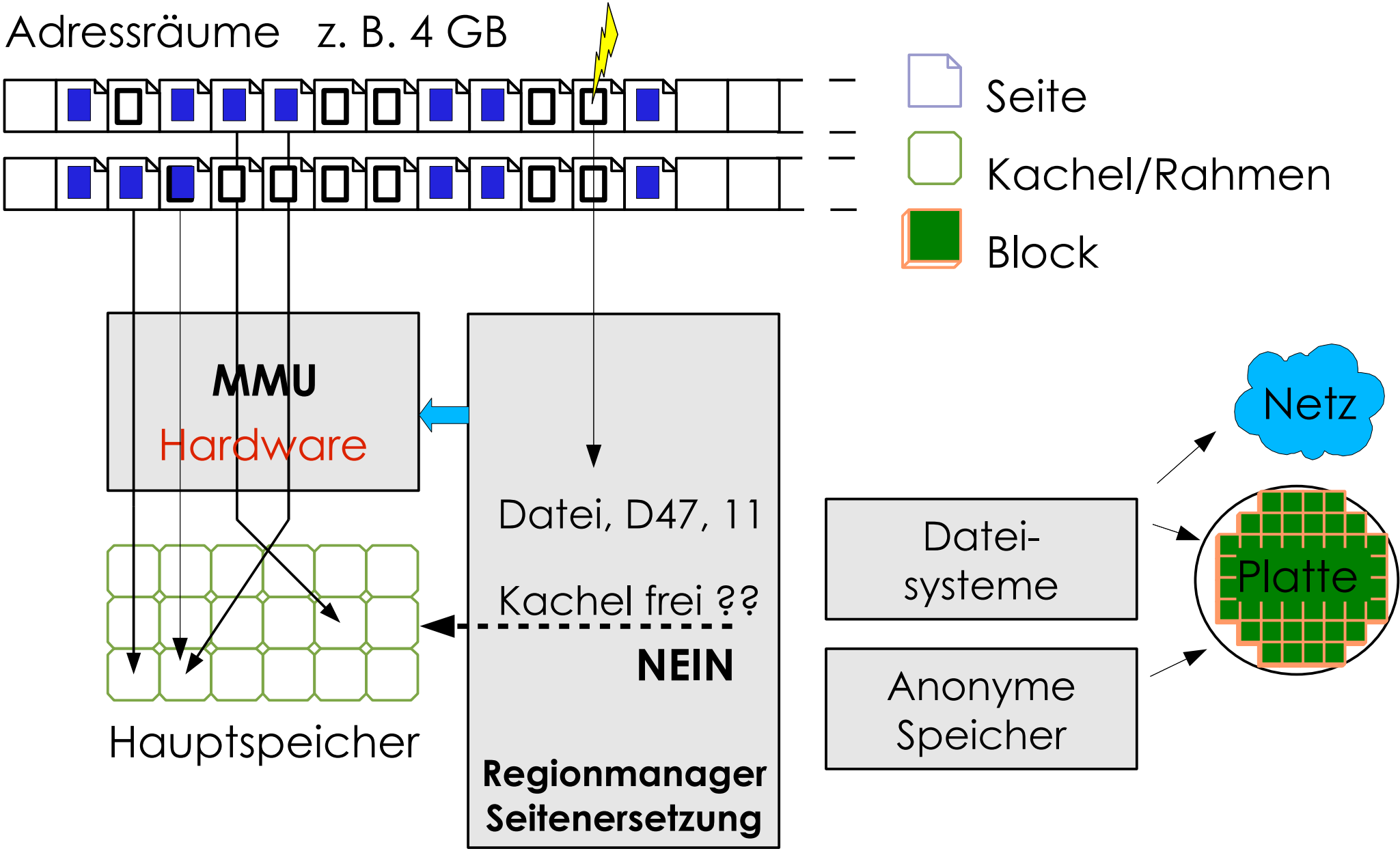
Das Zusammenspiel



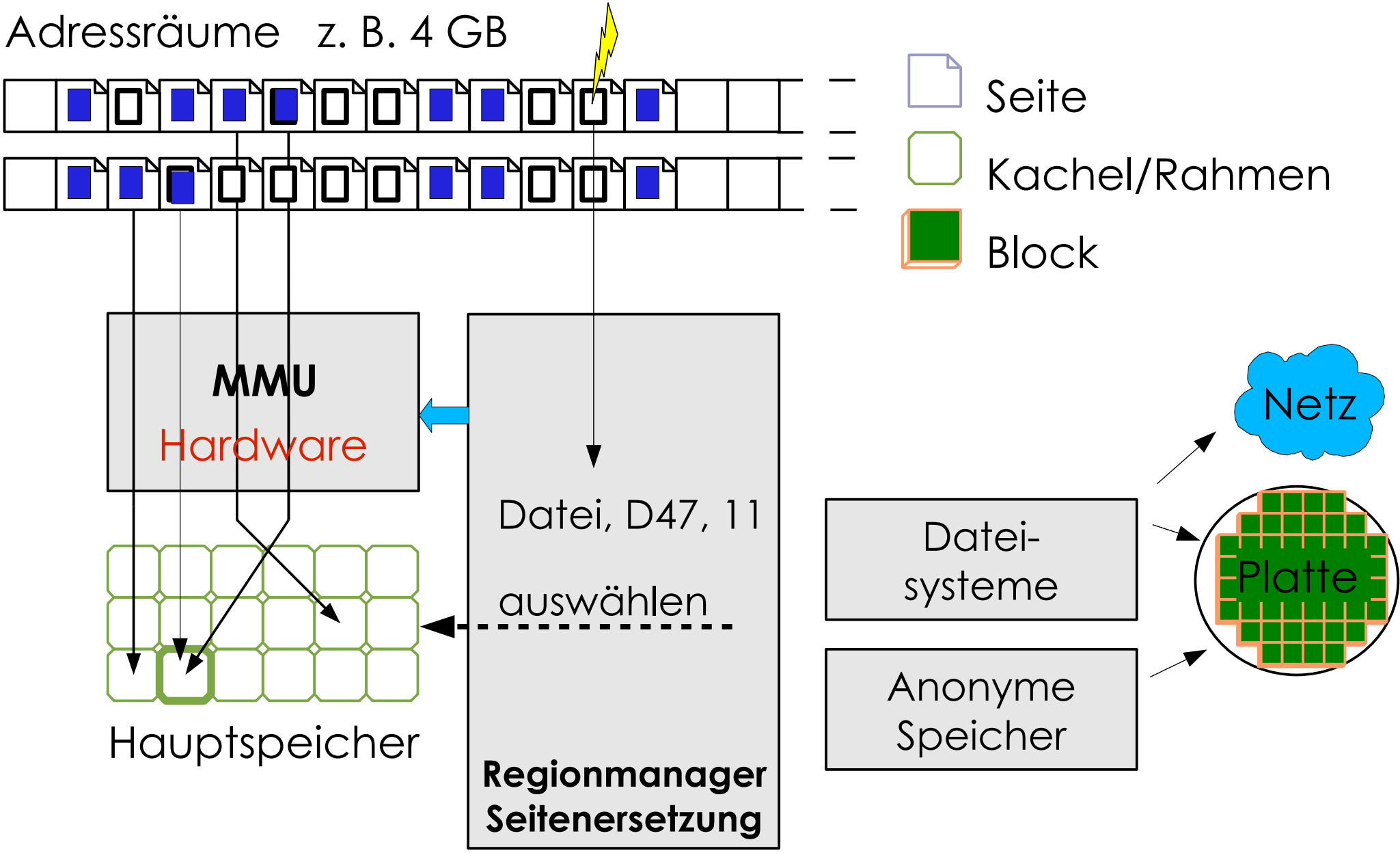
Das Zusammenspiel



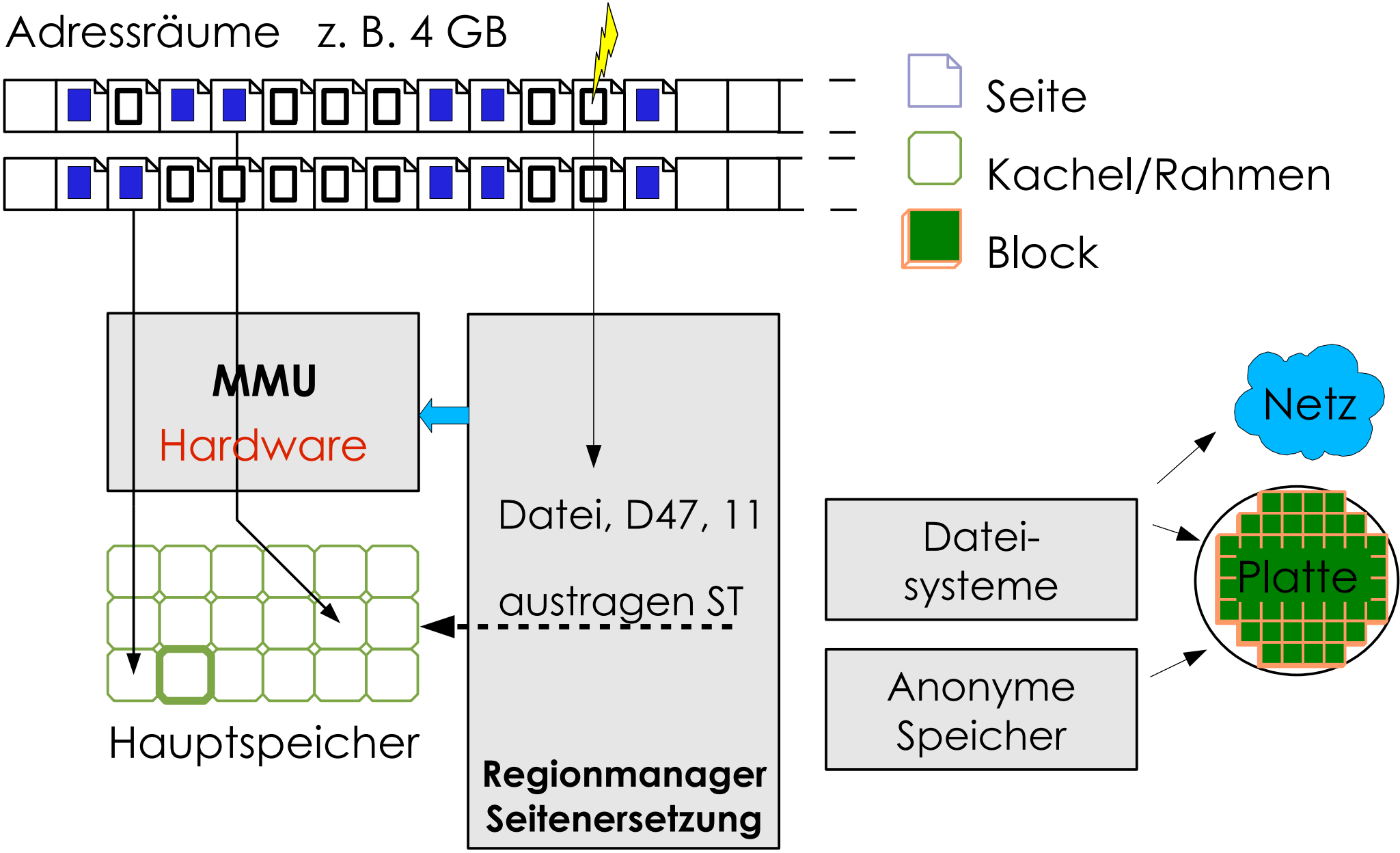
Das Zusammenspiel



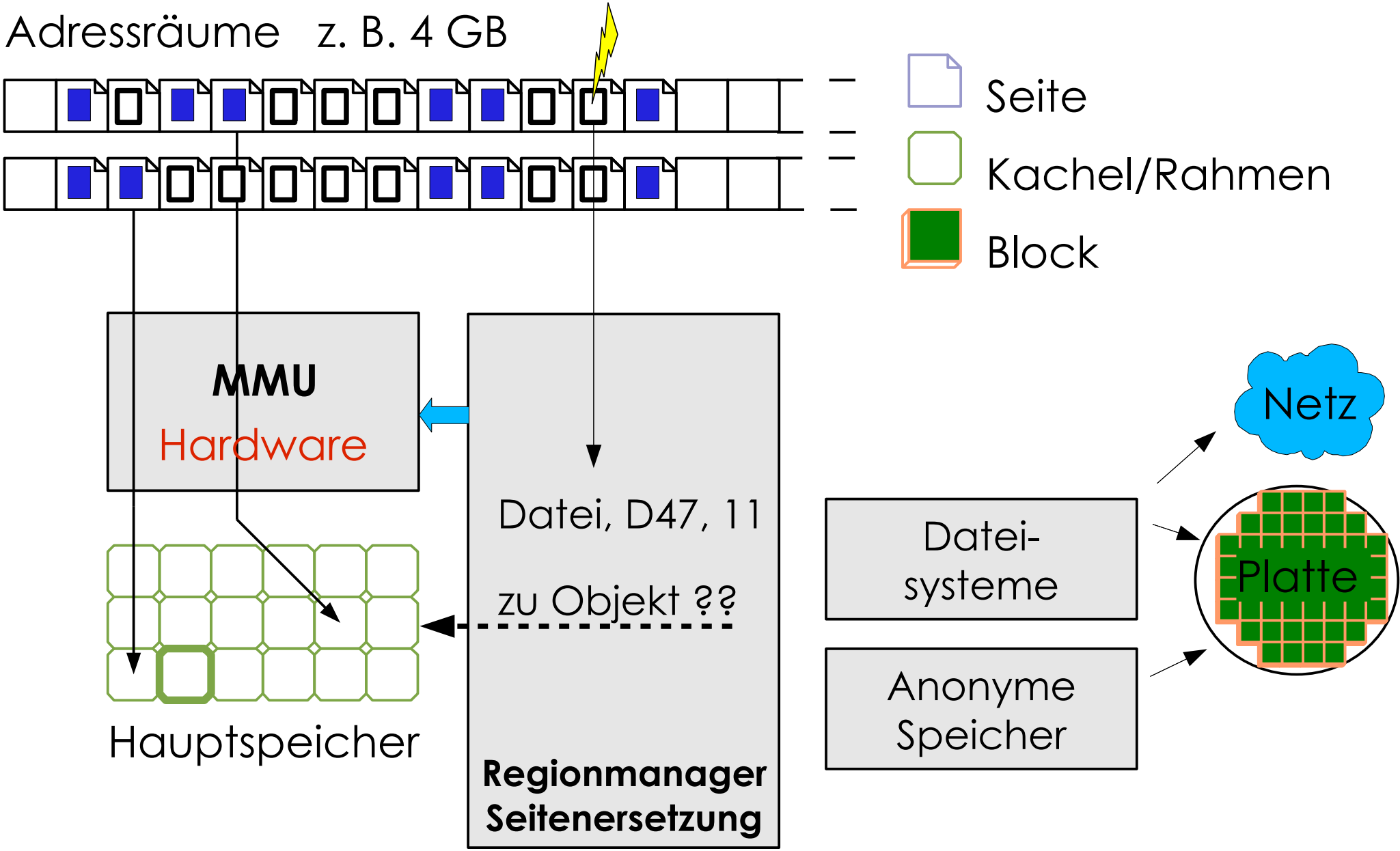
Das Zusammenspiel



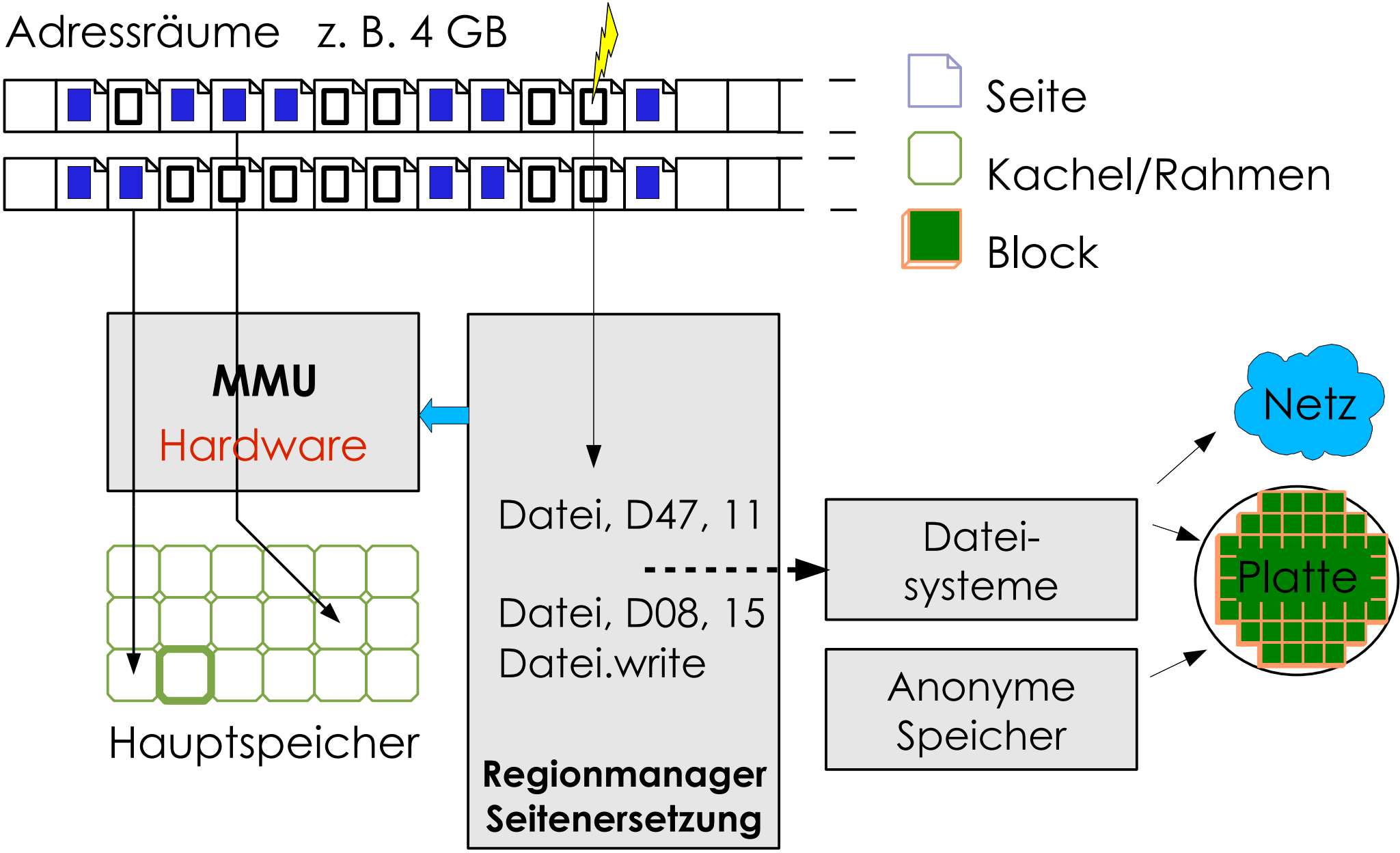
Das Zusammenspiel



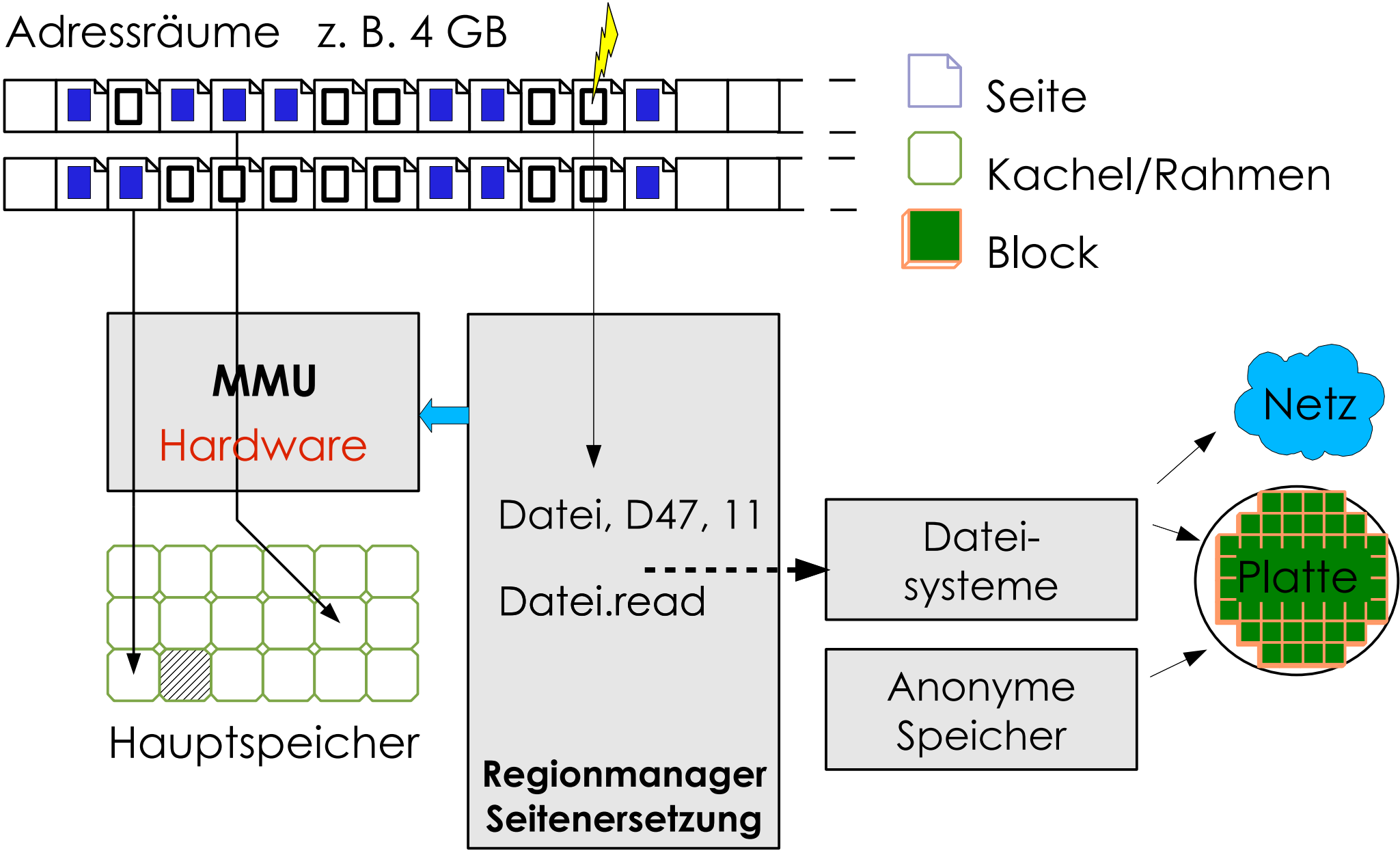
Das Zusammenspiel



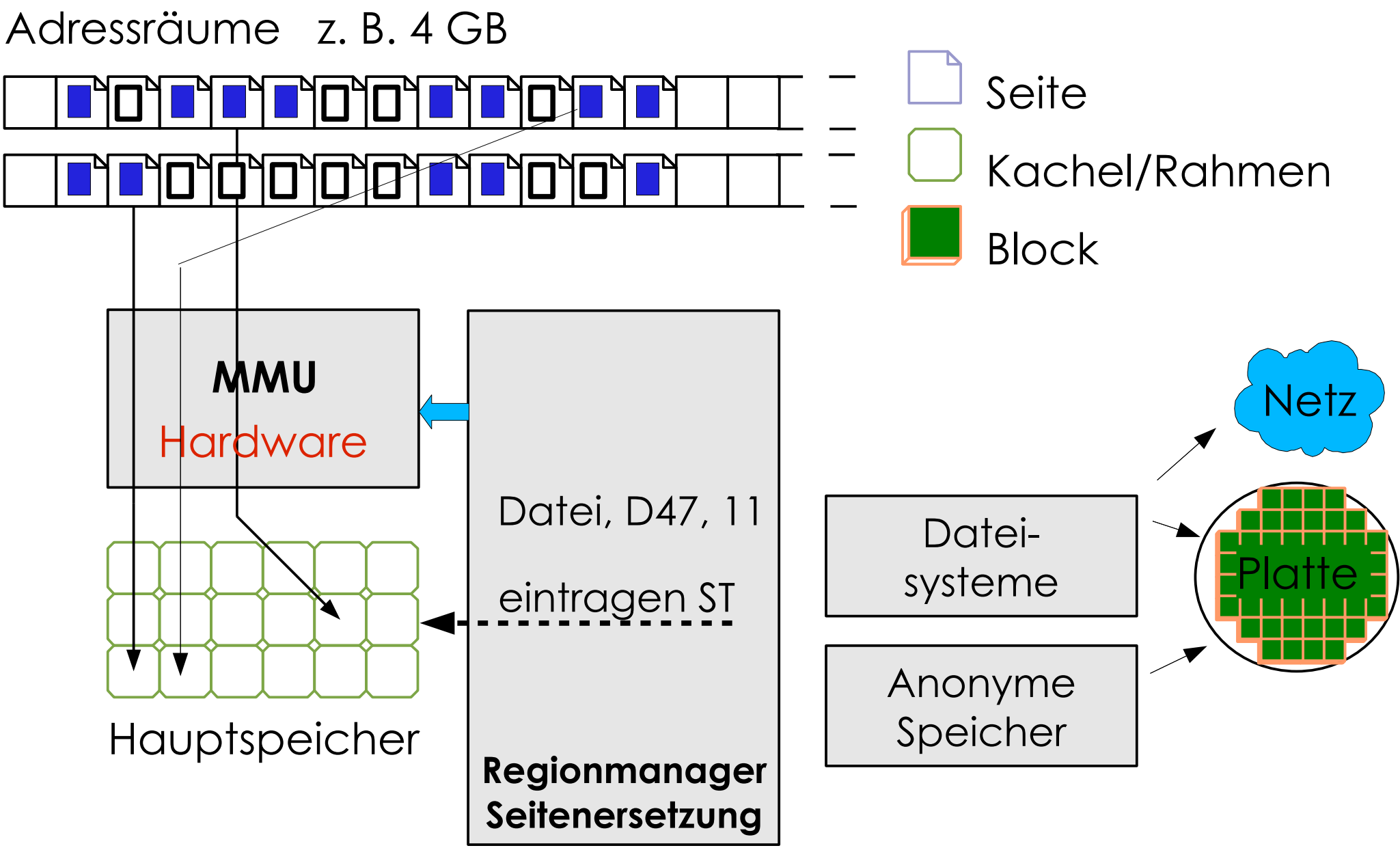
Das Zusammenspiel



Das Zusammenspiel



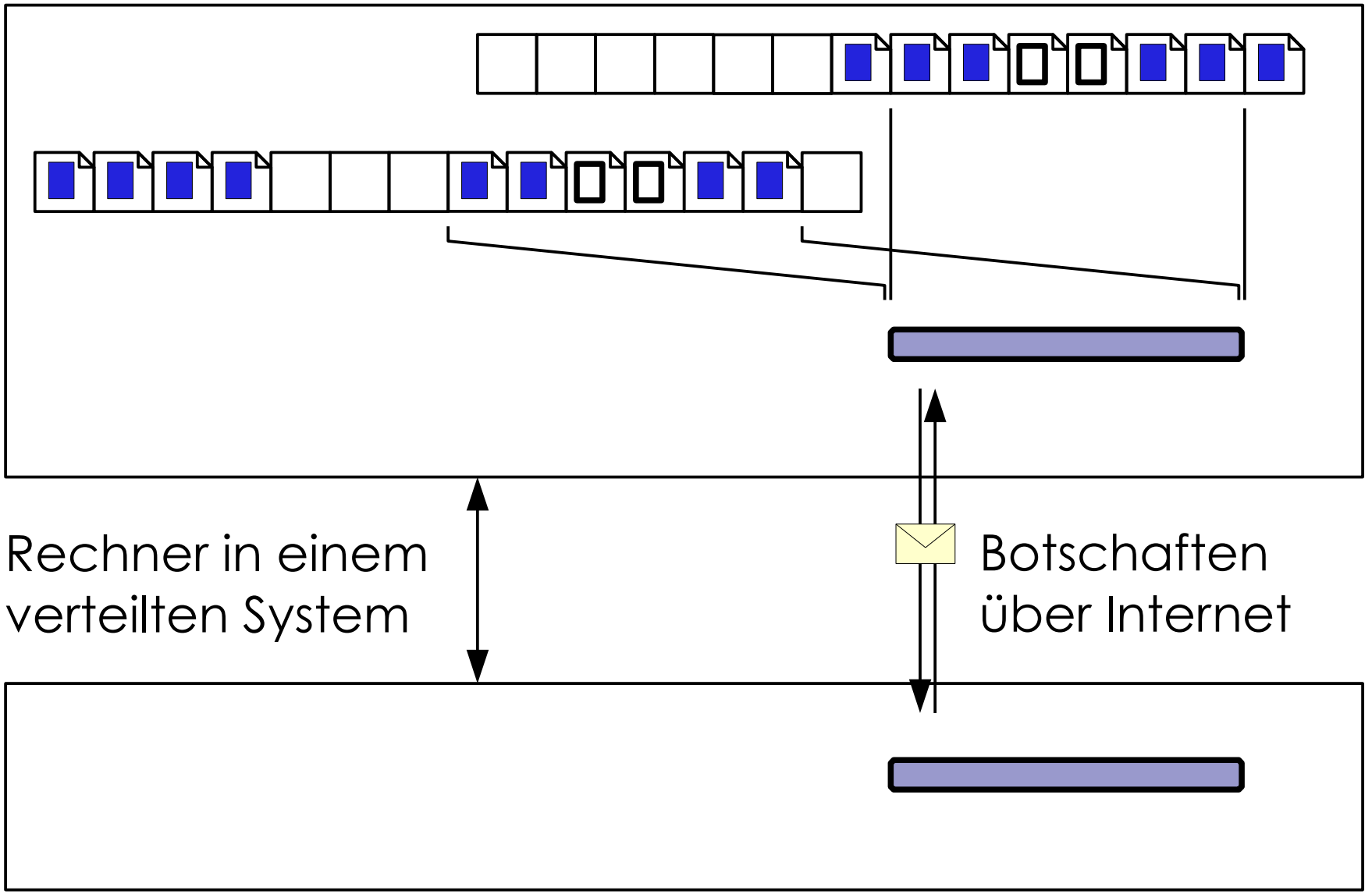
Das Zusammenspiel



Das Zusammenspiel

- Fehler
- Finde Objekt + Seite
- Suche ob Kachel schon vorhanden
- Falls ja eintragen fertig
- Falls nein, frei Kachel ???
- Falls frei Kachel vorhanden, Datei.read → eintragen in MMU #
- Falls keine freie Kchel → verdrängen (die geshared)
- Aus allen Seitentabelene austragen (Pfeile weg)
- Zu welche, Objekt gehört verdrängte Kachel? Datei.write
- Siehe oben

Speicherobjekte im Netz



Seitenfehlerbehandlung

- *MMU: welche Adresse, versuchte Operation*
- welche Region und welche Seite innerhalb dieser Region
- welches Speicherobjekt und welche Seite darin

- Anfrage bei Speicherobjekt:
  - Ist diese Speicherobjektseite schon in einer Kachel?
  - falls ja, weitergeben an MMU des auslösenden Prozesses, fertig
  - falls nein, wo ist sie sonst? (z. B. Platte)

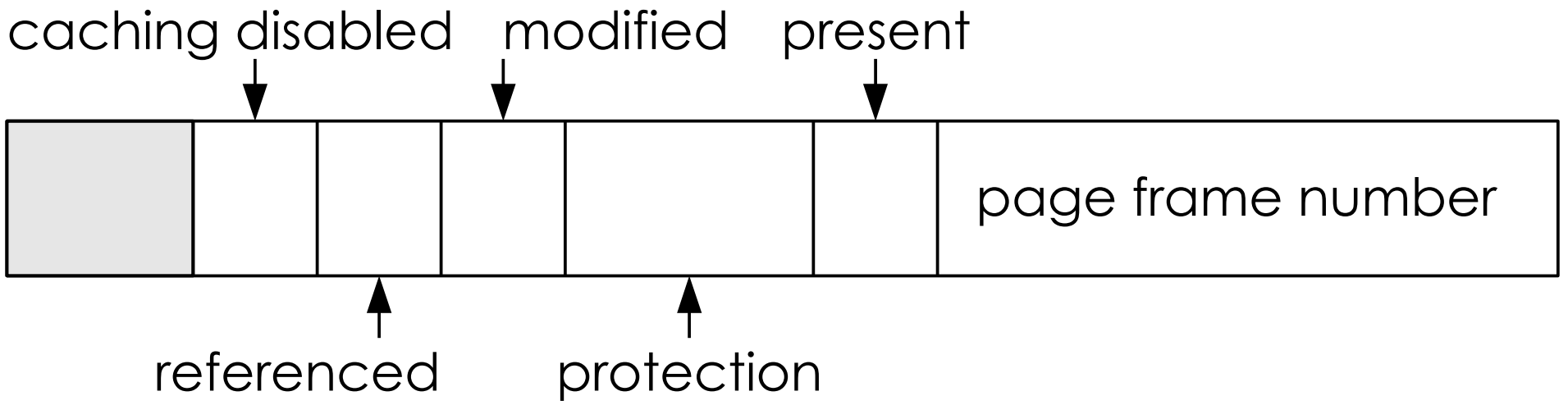
- *Beschaffen einer freien Kachel*
- *ggf. muss eine belegte Kachel freigemacht werden*
- *Füllen der Kachel mit Inhalt (z.B. E/A von Platte)*
- *Weitergeben an MMU des auslösenden Prozesses, fertig*

MMU-Ansteuerung (Hardware-abhängig)

Aufgaben

- Manipulation und Interpretation der MMU-Daten
- Atomare Operationen:
  - Eintrag einer Seite in einen Adressraum
  - Verdrängung einer Seite aus einem, mehreren oder allen Adressräumen
- Propagation von Attributen

Zur Erinnerung



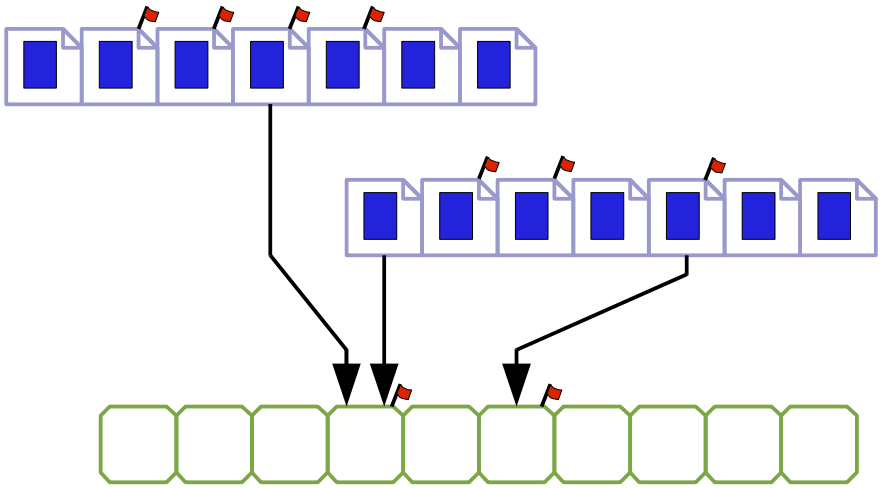
Seiten-Attribute

- present Seite befindet sich im Hauptspeicher
- protection welche Art Zugriffe in welchem CPU-Modus erlaubt sind
- modified schreibender Zugriff ist erfolgt („dirty“)
- used irgendein Zugriff ist erfolgt
- caching ein/aus (z. B. wegen E/A)

Propagation von Attributen

- Kacheln können mehreren Seiten mehrerer Adressräume zugeordnet sein
- Attribute *used, modified* werden durch Hardware für Seiten (nicht Kacheln) gesetzt

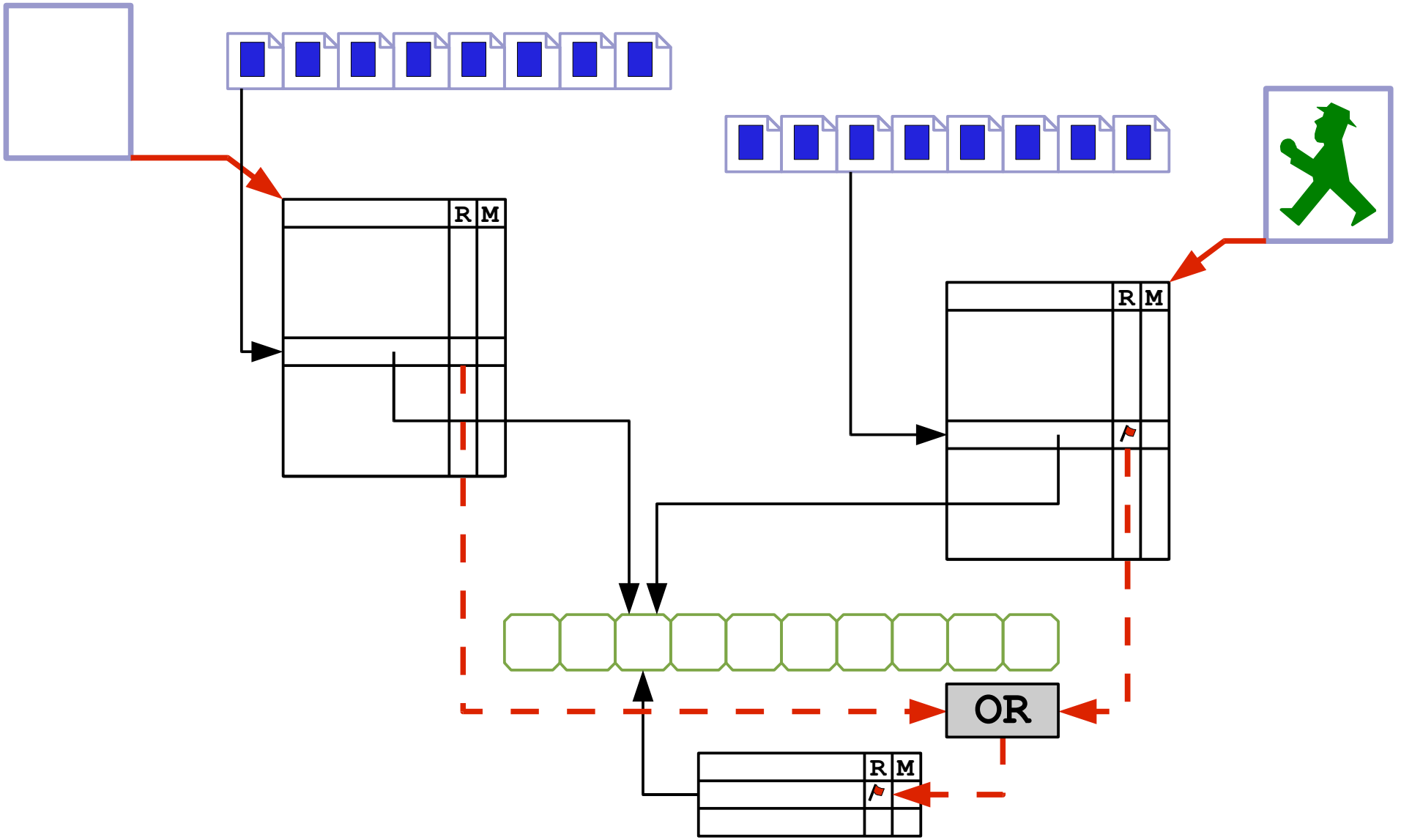
➔ notwendig:
  - Propagation der Attribute von Seiten zu Kacheln
  - Rückkehrabbildung: welchen Seiten ist eine gegebene Kachel zugeordnet



- Seite in Adressraum A: used, not modified
- Seite in Adressraum B: not used, modified

➔ Kachel: used, modified

Propagation der Seitenattribute

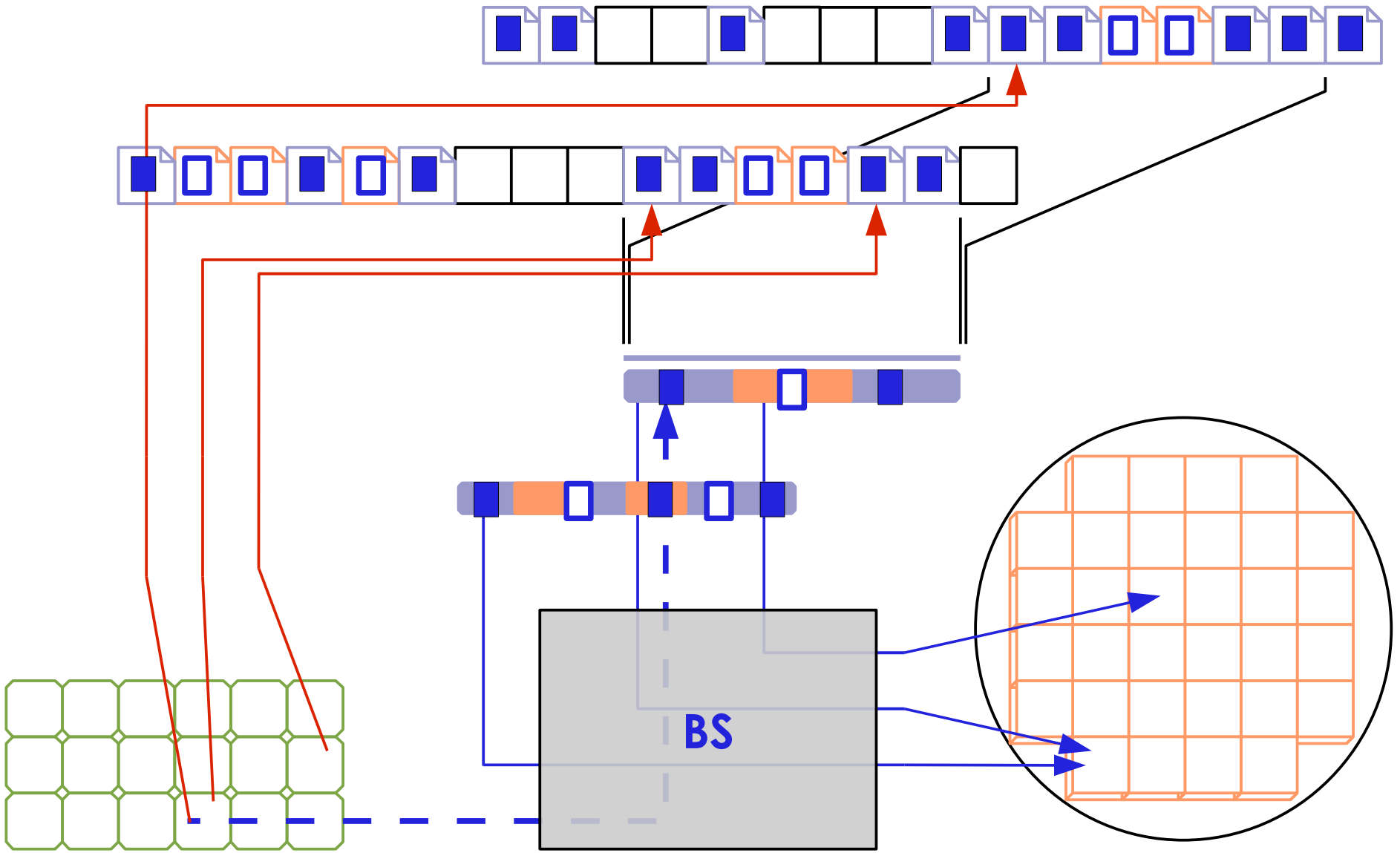


Umkehrabbildung: Kacheln zu Adressraum

Für:

- Attributpropagation
- bei Verdrängung einer Kachel:
  - Auffinden der Seitentabellen, aus denen Kachel ausgetragen werden muss
  - Identifierung des Speicherobjektes, damit Inhalt weggeschafft werden kann

Häufige Hilfskonstruktion: Block-Kachel-Tabelle



Häufige Hilfskonstruktion: Block-Kachel-Tabelle

Aufgabenteilung

- Speicherobjekte: Allokation von Blöcken
- Block-Kachel-Tabelle („Buffercache“): Ein-/Auslagern

Wozu

- gemeinsame Organisation der Ein-/Auslagerung
- bei Freischaufeln einer Kachel muss nicht das Speicherobjekt angefragt werden, sondern man kann direkt auf den zugeordneten Block schreiben

Systemstrukturen für Speicherobjekte

Unix/Windows/...: „monolithisch“

- teilweise durch Prozesse (mit eigenem Adressraum) im User-Mode
- großenteils aber fest in Betriebssystem integriert

Mikrokernbasiert

- alle Speicherobjekte bereitgestellt durch „Server“-Prozesse mit eigenem Adressraum und im User-Mode

Prozesse und Speicher

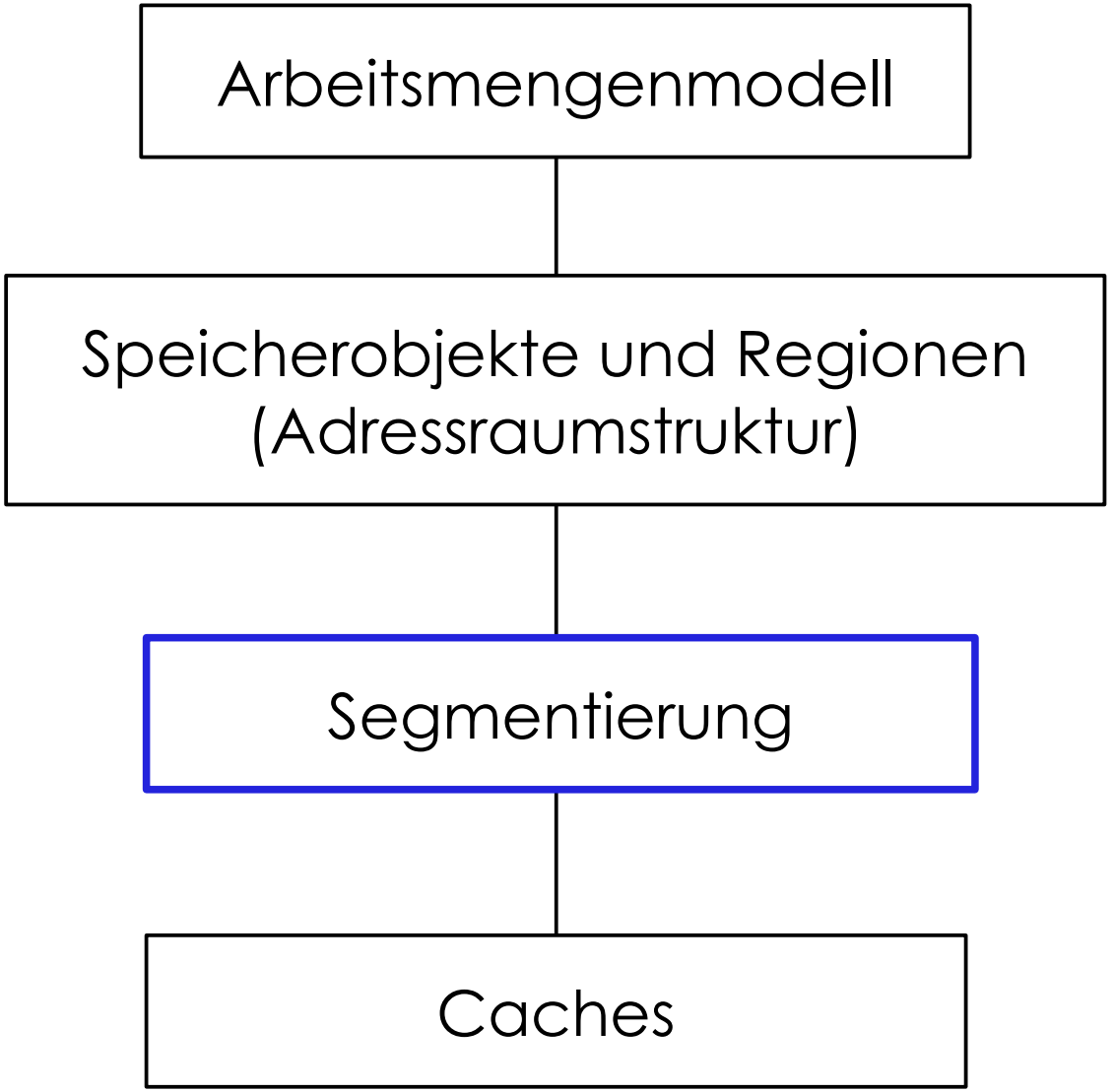
Prozess-Zustand

- Seitentabellen
- mehrere parallele Aktivitäten je Adressraum: Threads

Prozess-Umschaltung

- TLB behandeln (falls kein Prozess-Identifier in TLB → löschen)
- Caches
- Botschaftenimplementierung ...

Wegweiser


Arbeitsmengenmodell
Speicherobjekte und Regionen
(Adressraumstruktur)
Segmentierung
Caches

Segmentierung

Ausgangspunkt

- logisch zusammenhängende, aber untereinander unabhängige Bereiche

Beispiele

- Übersetzertabellen
- Bibliotheken
- Keller, Halde, Programm

Zweidimensionale Adressierung

Eigenschaften

- Segmentnummer
- Adresse innerhalb Segment
- beliebige Länge

Folgen/möglicher Einsatz

- Länge und Zugriffsart können überwacht werden, z. B. ARRAY
- Relokation einfach
- Schutz
- „Shared Libraries“

Hardware-Implementierung

- notwendig
- erfahrungsgemäß (Intel 486) teuer

Diskussion (nach Tanenbaum)

Consideration	Paging	Segmentation
Need the programmer be aware that this technique is being used?	No	Yes
How many linear address spaces are there?	1	Many
Can the total address space exceed the size of physical memory?	Yes	Yes
Can procedures and data be distinguished and separately protected?	No	Yes
Can tables whose size fluctuates be accommodated easily?	No	Yes
Is sharing of procedures between users facilitated?	No	Yes
Why was this technique invented?	The get a large linear address space without having to buy more physical memory	

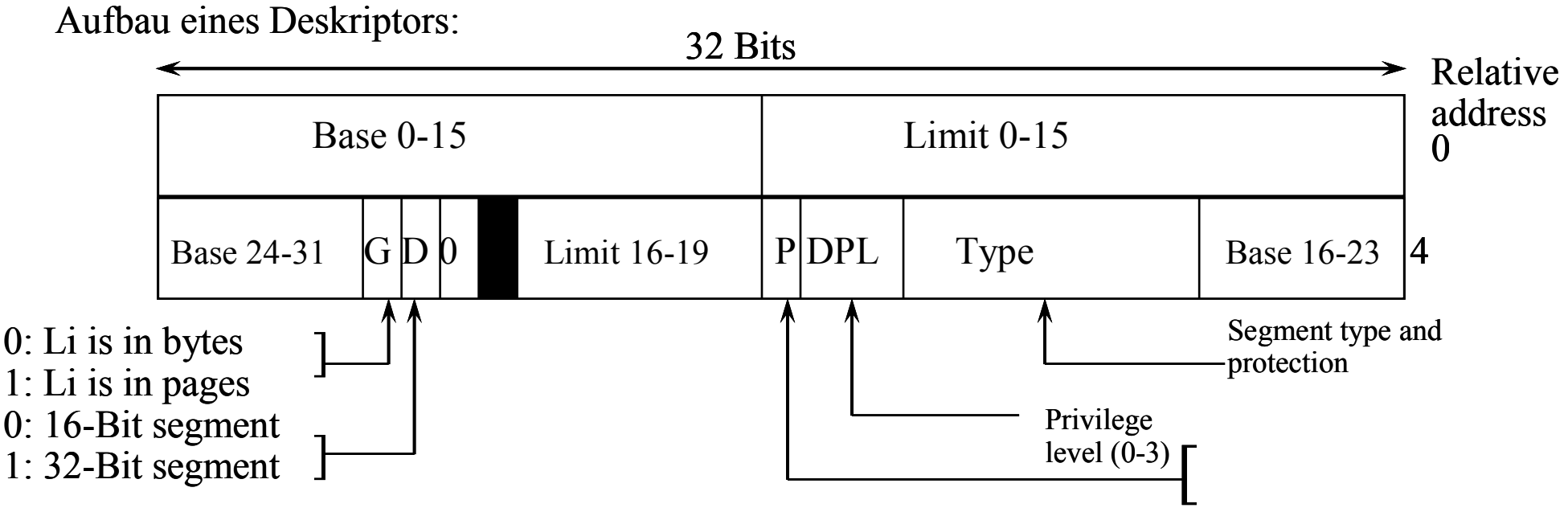
Segmentierung und Paging (z. B. i386)

Segmente

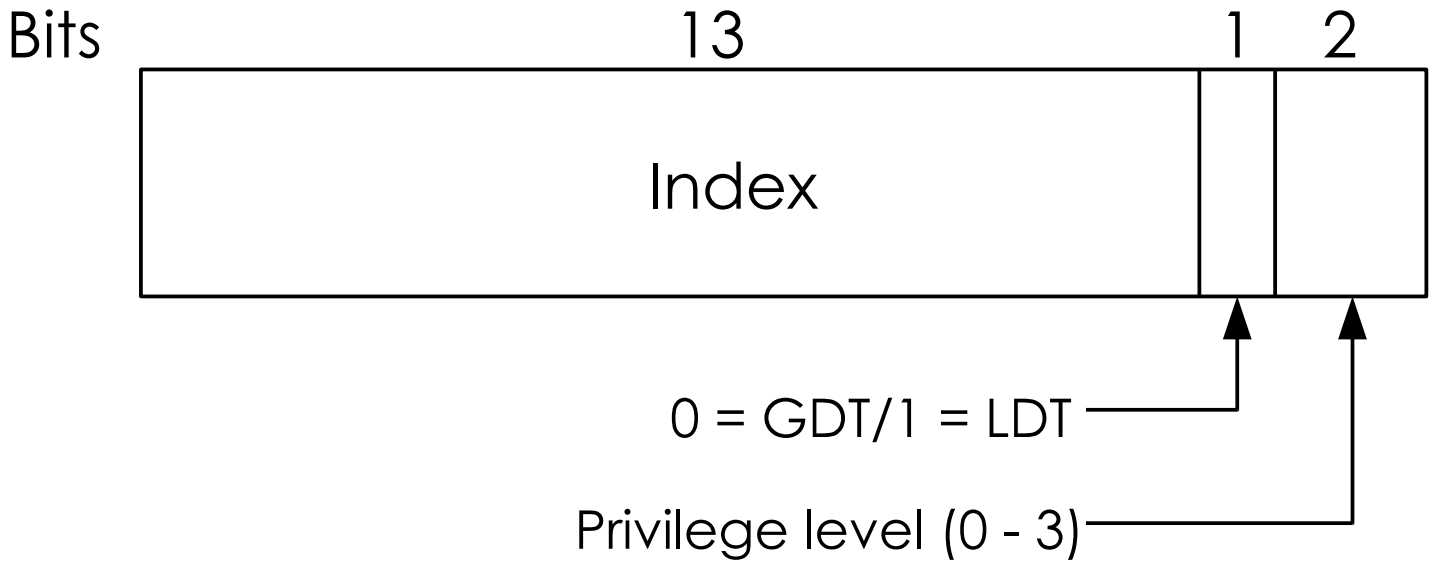
- Codesegment: **CS**
- Datensegment: **DS**
- Stacksegment: **SS**
- weitere: **ES**, **FS**, **GS**

Segmentdeskriptortabellen

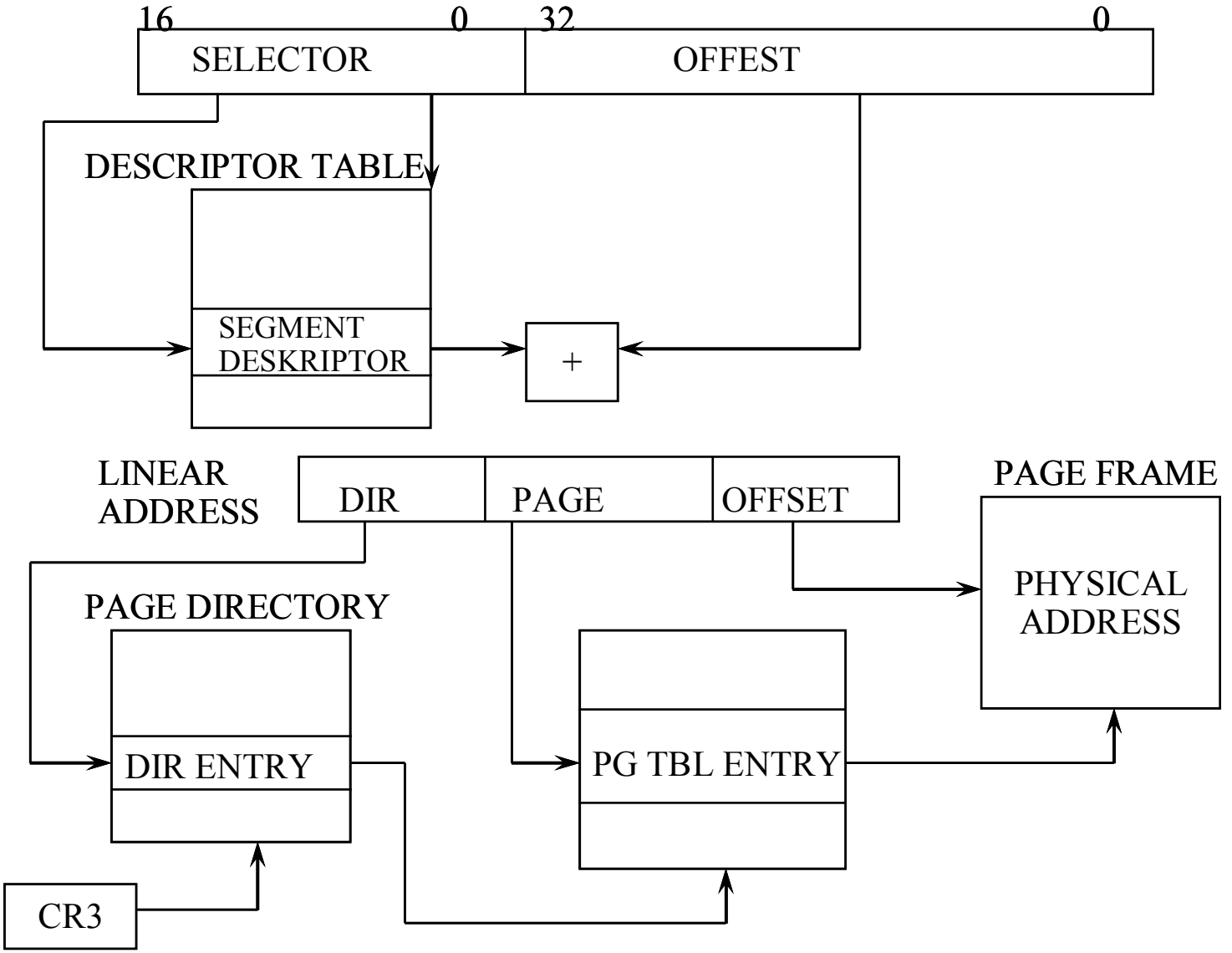
- per Prozess (local desc. table)
- per System (global desc. table)



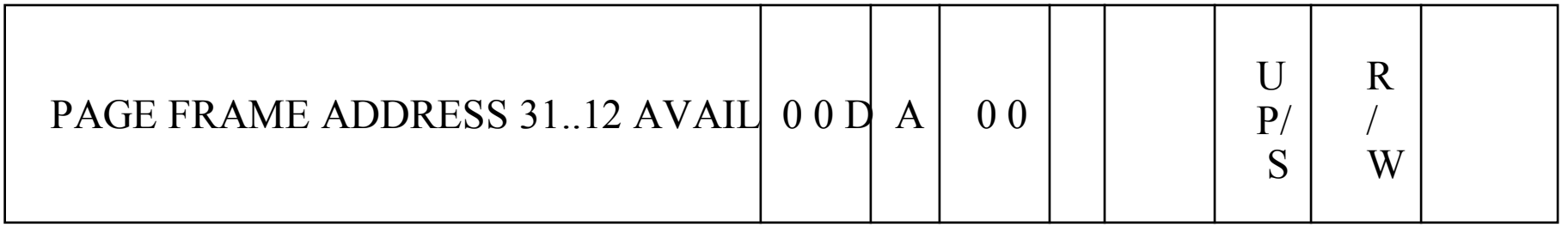
Auswahl per Selektor



Die vollständige Abbildung



Aufbau Seitentabellen-Eintrag



P - PRESENT
R/W - READ/WRITE
U/S - USER/SUPERVISOR
A - ACCESSED
D - DIRTY
AVAIL - AVAILABLE FOR SYSTEMS PROGRAMMER USE

NOTE: 0 INDICATES INTEL RESERVED. DO NOT DEFINE.

Schutz (1)

Segmente und Prozesse

- vier Privilegierungsstufen (Ringe)
- Prozess befindet sich jeweils in einer Stufe
- Segment wird Stufe zugeordnet
- Prozess darf auf Segmente derselben und niedrigerer Stufen zugreifen
- Übergang durch sogenannte „call gates“

Schutz (2)

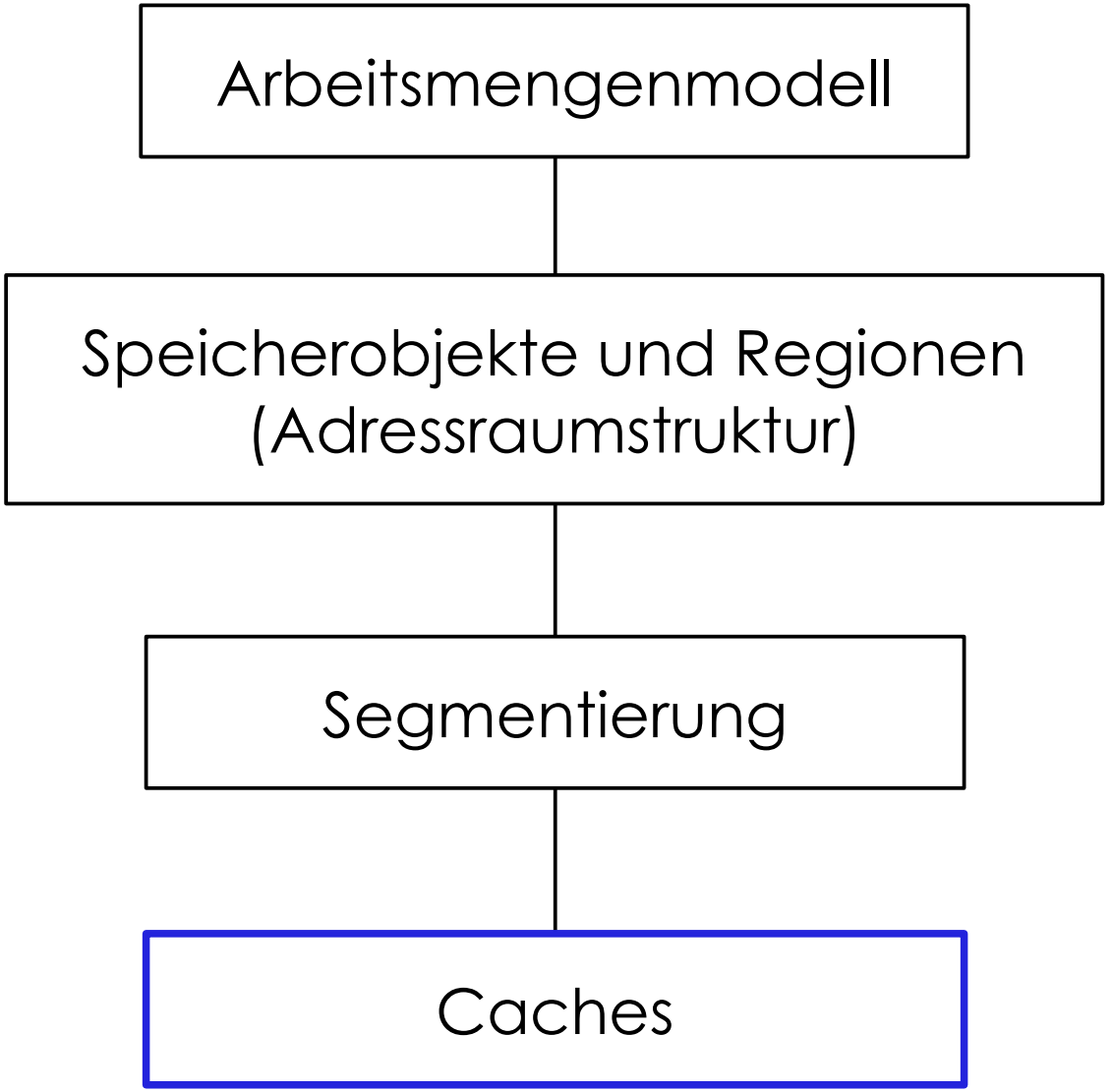
Seiten und Prozesse

- Prozesse befinden sich in „User oder System-Level“
  User = Ring 3, System = Ring 0..2
- Seiten geben Zugriffsrechte: r/w Lesen/Schreiben
- Regeln:
  - Prozess in System-Level: darf alles
  - Prozess in User-Level: darf nicht auf System-Level-Seiten zugreifen; darf nur dann auch schreiben, wenn R/W=W
- Entwurfsfehler!!! (in 486 korrigiert)

Quantitative Angaben : Anzahl TLB-Einträge

386, 486	32
Pentium (MMX)	Code 32, Daten 64
Pentium III (Coppermine)	Code: 34 (2 * 4MB, 32 * 4KB), Daten: 72 (8 * 4MB, 64 * 4KB)
Pentium IV	Code 64, Daten 64
Itanium 2	Code L1: 32, L2: 96, Daten: 128
Opteron	L1: Code: 40 (8 * 4MB, 32 * 4KB), Daten: 40 (8 * 4MB, 32 * 4KB) L2: 512 Code/Daten (flush filter!)
PowerPC (PPC 750)	Code 128, Daten 128
Alpha (21264)	Code 128, Daten 128 (tagged!)

Wegweiser


Arbeitsmengenmodell
Speicherobjekte und Regionen
(Adressraumstruktur)
Segmentierung
Caches

Rechnerarchitektur : Caches

Ziel

- größtmöglicher Speicher **+** schnellstmöglicher Speicher

Grundlage

- Lokalitätsverhalten bei Speicherzugriffen

Idee

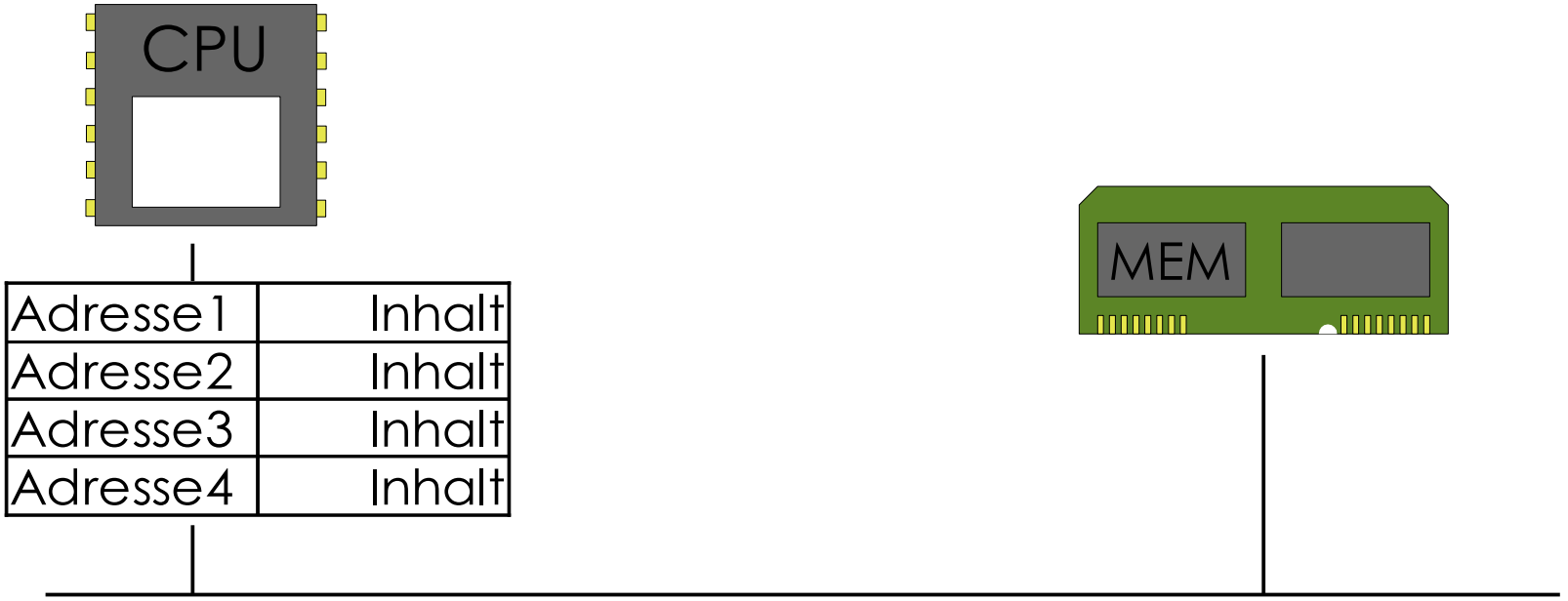
- Anordnung von schnelleren, kleineren Speichern zwischen größeren, langsameren Speichern und Nutzer

- bei jedem Zugriff auf langsameren Speicher wird auch in schnellerem Speicher eine Kopie angelegt und genutzt

Beispiele für Caches

- Adressumsetzung
  Assoziativspeicher für langsame Seitentabellenspeicher
- Platten
  schneller, aber nicht persistenter Speicher in Laufwerk oder Controller
- Verteilte Dateisysteme (später)
  Kopien von entfernten Dateien auf lokalem Rechner
- schneller Speicher zwischen CPU und Hauptspeicher

Caches für Hauptspeicher



Characteristika

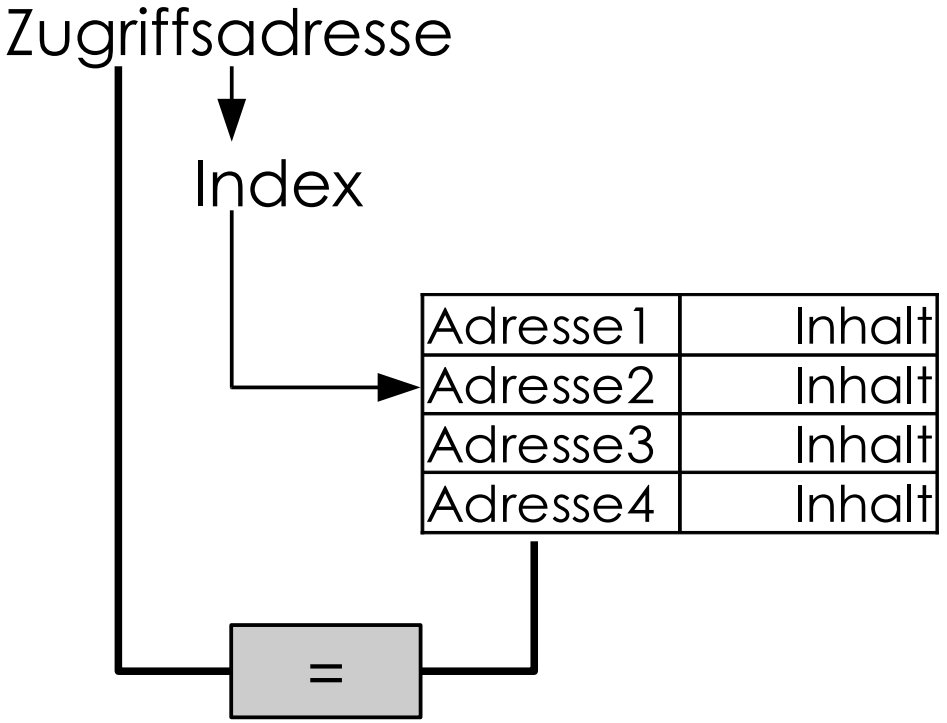
- Größe, Geschwindigkeit, Cachelinegröße
- Organisation: Austauschverfahren, Zugriffsart, physisch vs. virtuell ...

➔ Tag: im Cache gespeicherte Adresse

Organisation von Caches (1)

Zugriffsarten

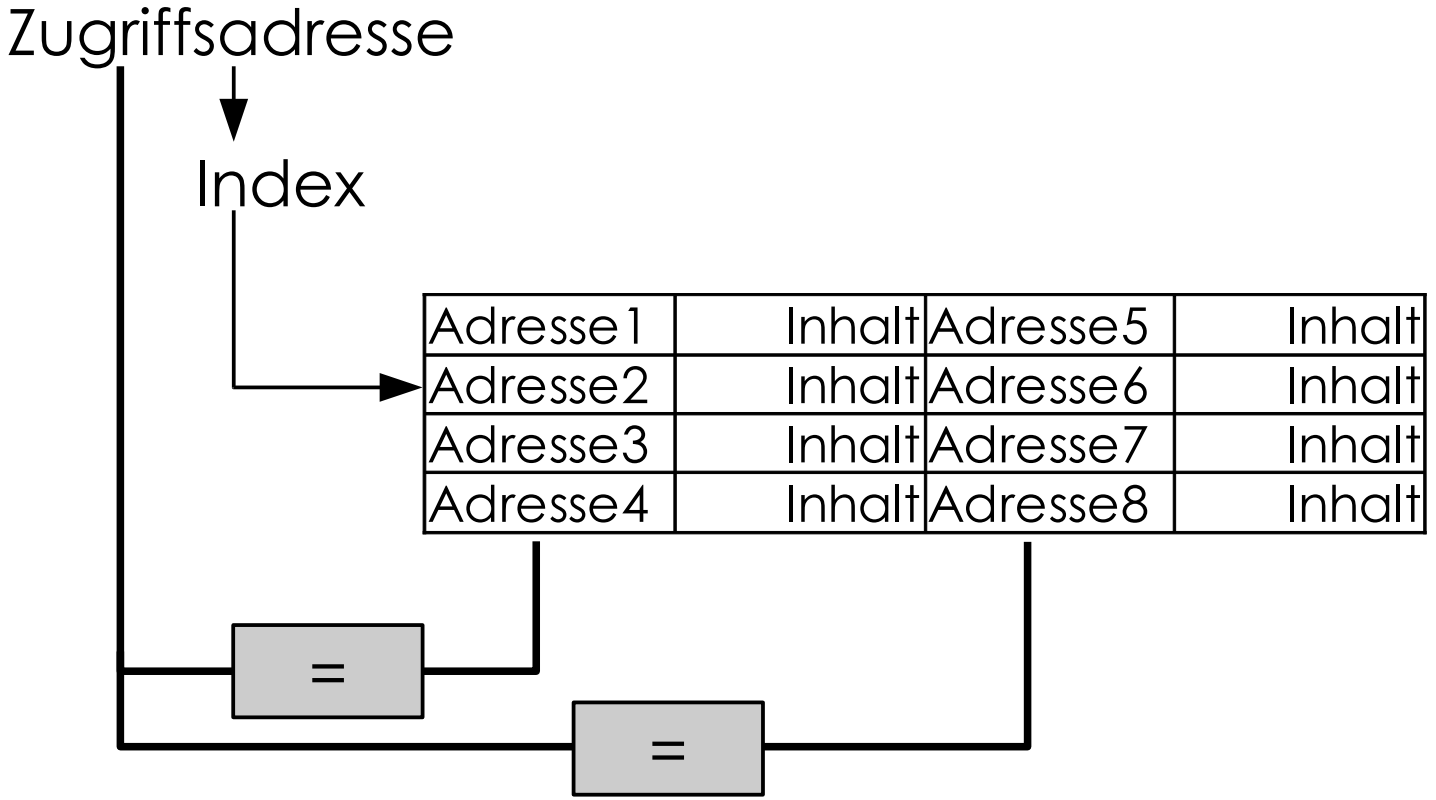
- direkt:aus Adresse wird Index in Cache berechnet
  Vergleich zwischen Zugriffsadresse und Tag



Organisation von Caches (2)

Zugriffsarten

- direkt:• n-Wege-assoziativ: zu jedem Index gibt es n Tag-Inhalt-Paare, Auswahl: assoziativ



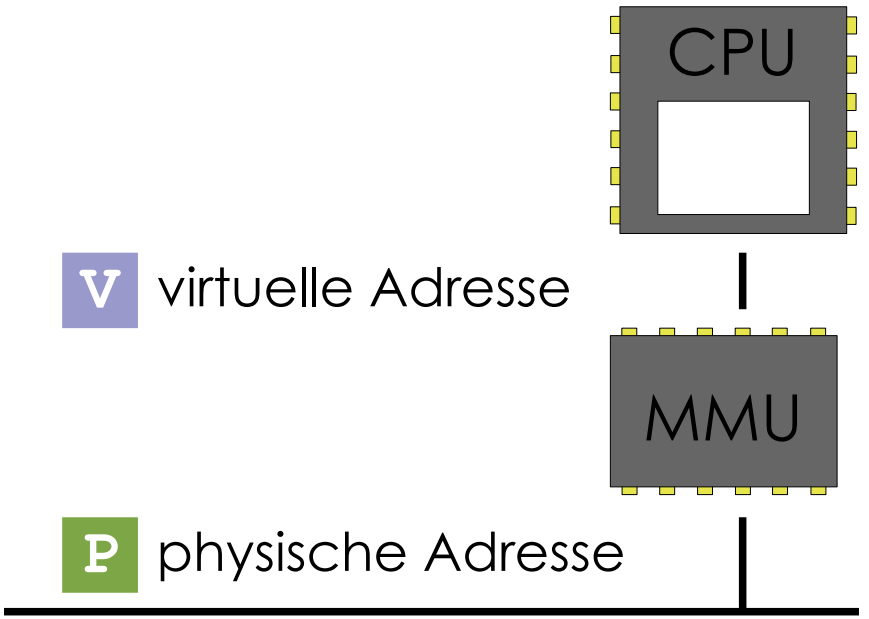
Organisation von Caches (3)

Virtuelle vs. physische Tags und Indizes

- virtuelle Tags und Indizes
  - schnell (spart MMU-Umsetzung)
  - bei integrierter CPU/MMU: nur auf Chip
  - sharing problematisch

- physische Tags und Indizes
  - langsamer
  - auch außerhalb CPU-Chip
  - sharing einfach

- physiche Tags, virtuelle Indizes
  - Limitation durch Seitengröße



Pentium III vs. Pentium IV

Pentium III

- virtuell indiziert / physische Tags

Pentium IV

- virtuelle Tags
- trace cache → dekodierte Instruktionen werden in einem Cache abgelegt

Konsequenzen für Effizienz der Prozessumschaltung ??

Zusammenfassung Speicher

- Wozu virtueller Speicher?
- Speicherhierarchie
- große Programme
- Locality of Reference
- Protection
- Vereinfachung der Programmierung