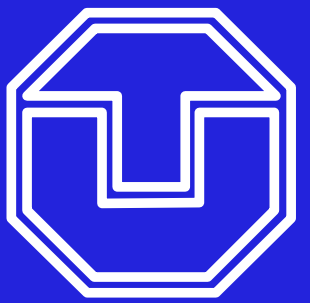
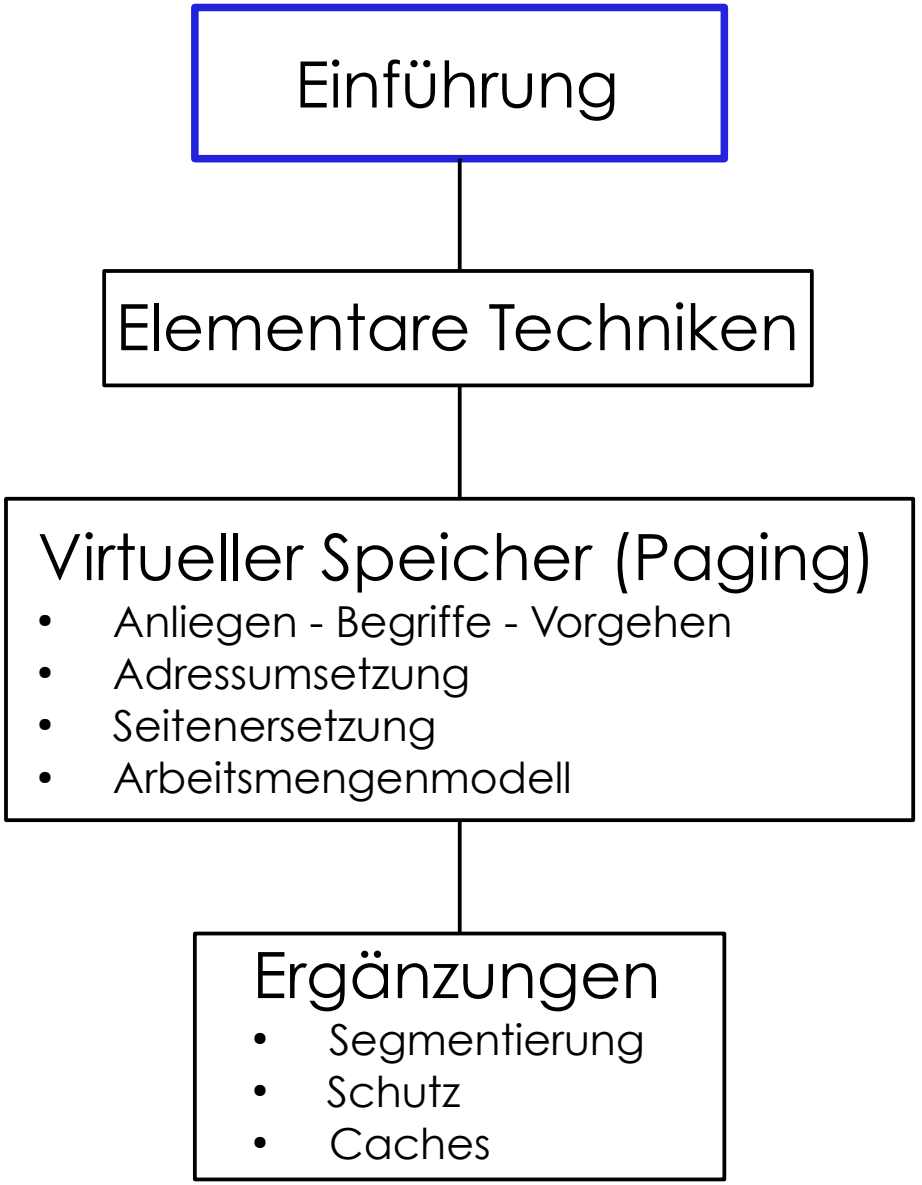
Speicher

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

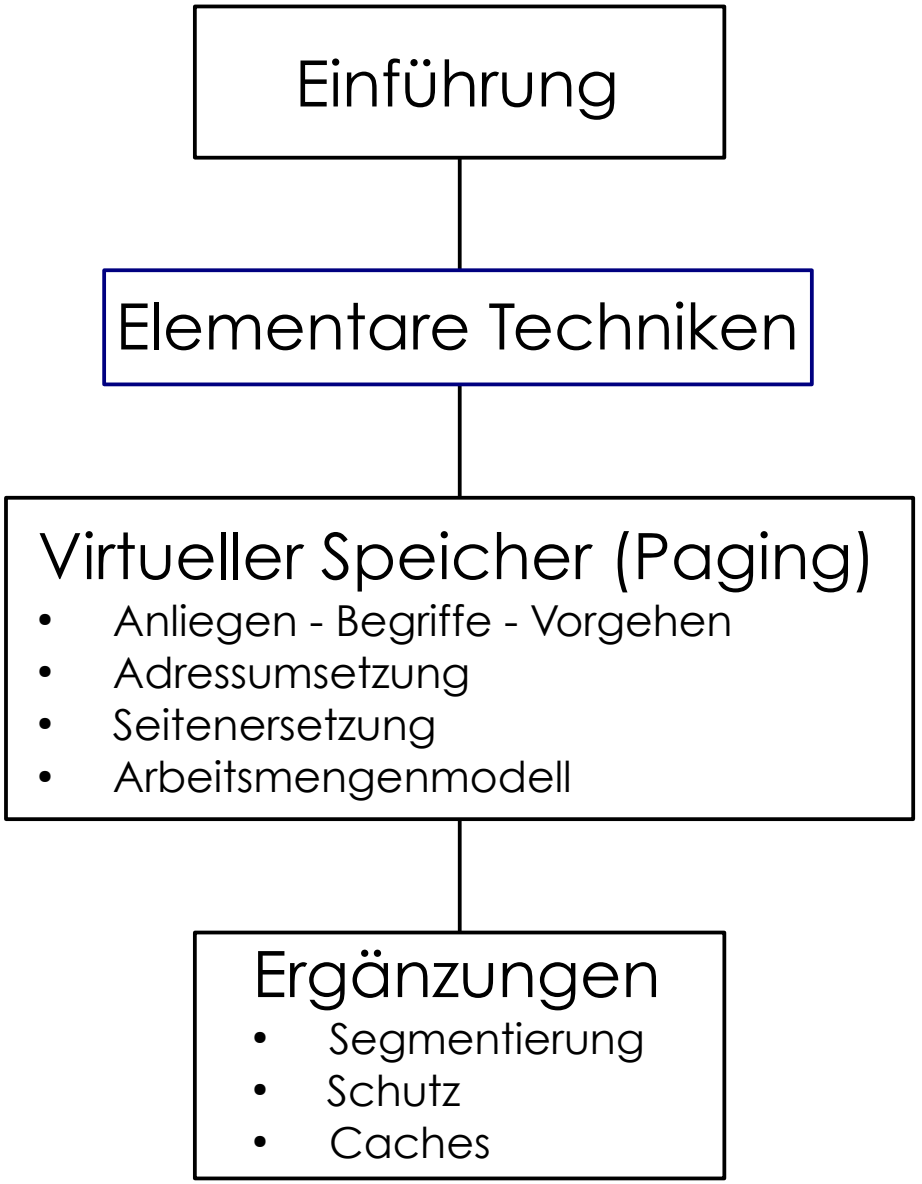
1965-er: *n* Prozesse, *n* disjunkte Speicherbereiche, statisch

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

Aufgaben der Speicherverwaltung

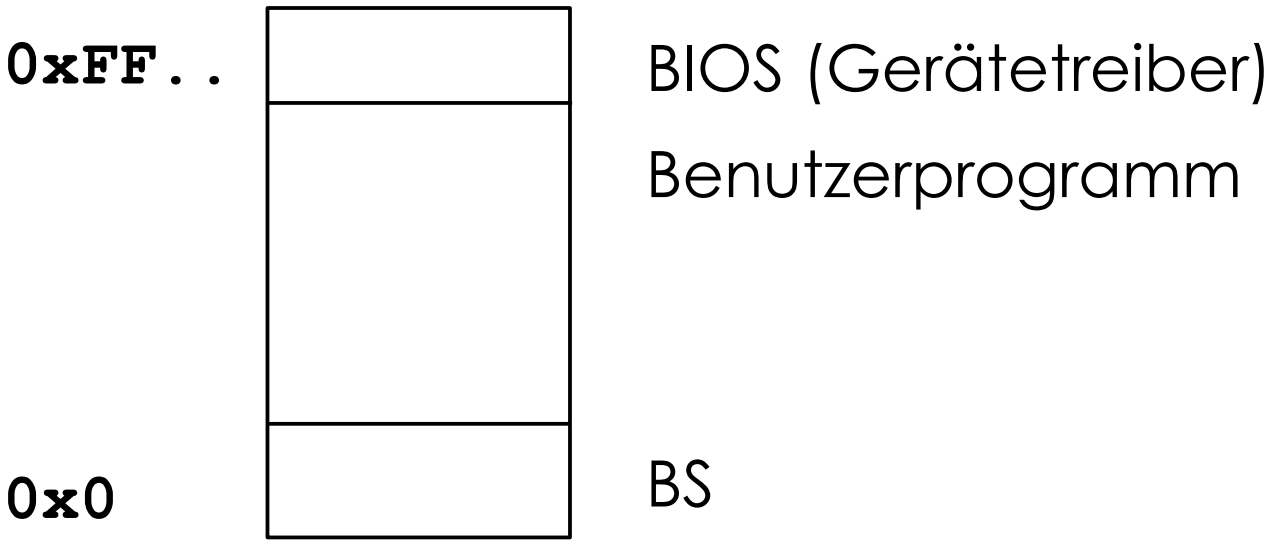
- Bereitstellung von Adressräumen
- Aufbau von Adressräumen durch Zuordnung logischer Objekte
- Verwaltung des Betriebsmittels Hauptspeicher
- Schutz vor unerlaubten Zugriffen
- Organisation gemeinsamer Nutzung („Sharing“) physischen Speichers und logischer Objekte

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

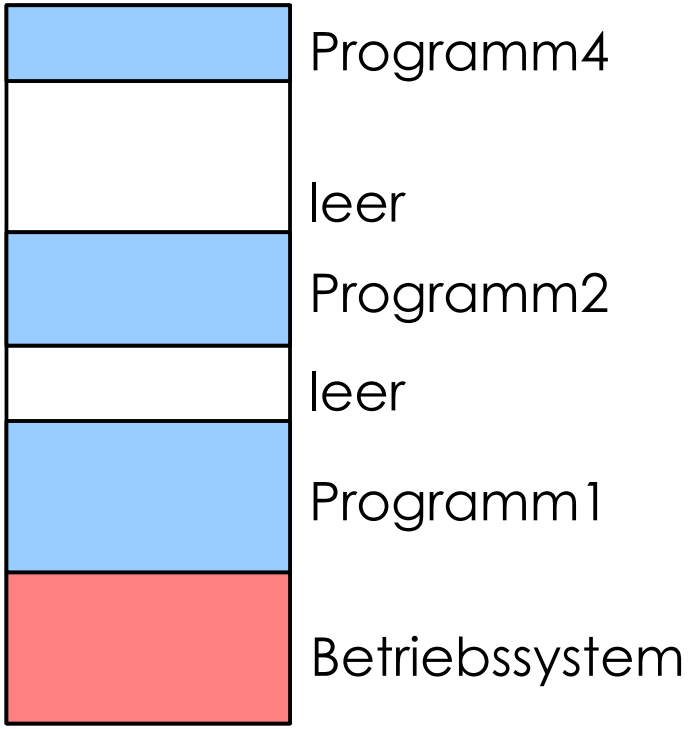
Fragestellungen

- Relokation

  Programme verwenden unterschiedliche Adressen, wenn sie in unterschiedlichen Partitionen ablaufen

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

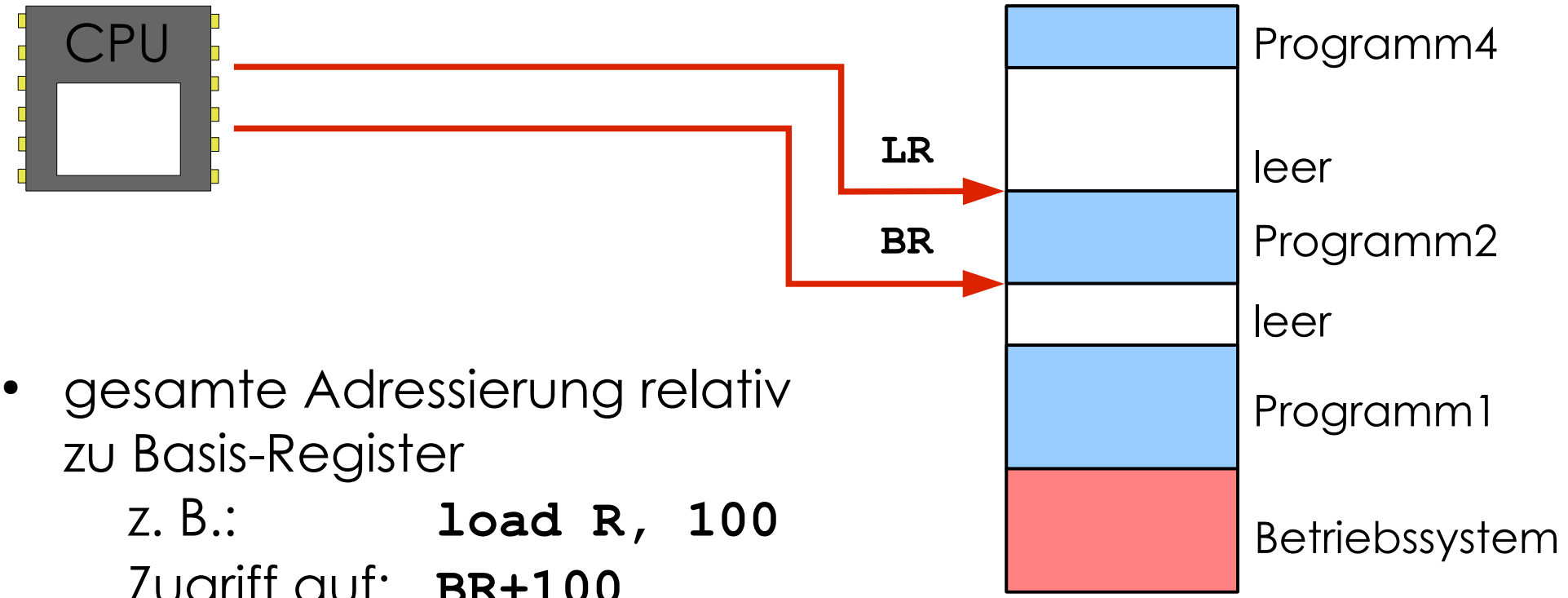
- Schutz der Partitionen voreinander
  - ➔ Abhilfe:
    - Überprüfung der Adressen zur Laufzeit (Hardware)



Limitationen

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

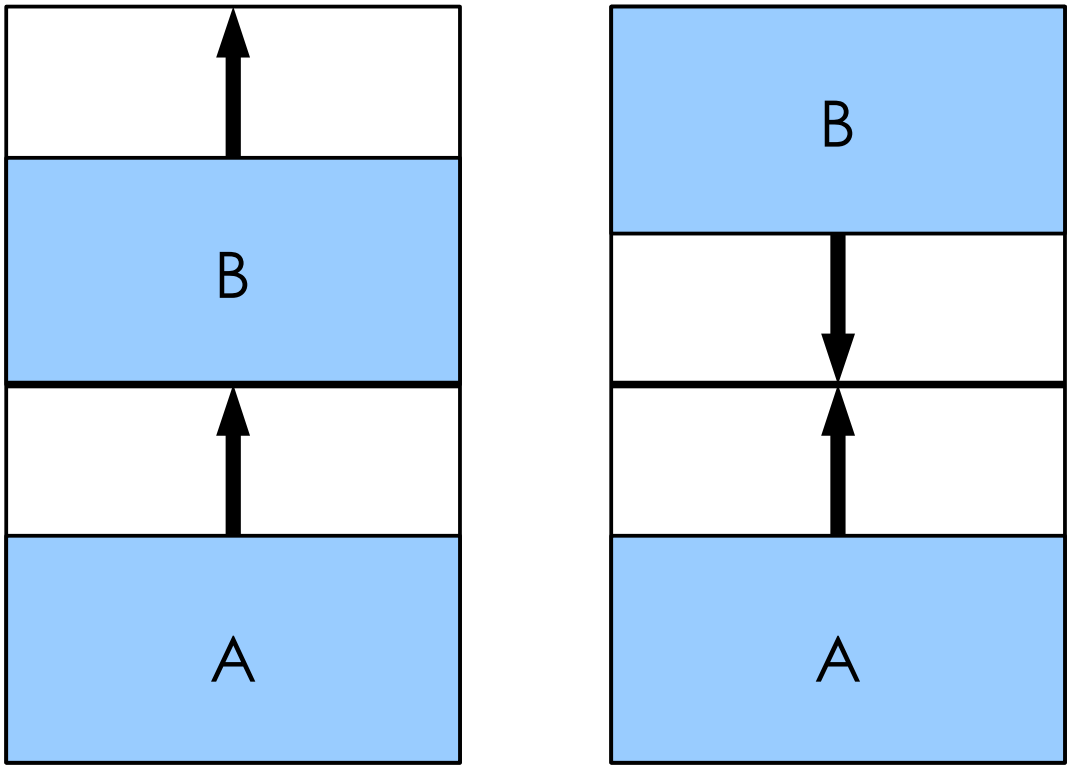
Einfaches Modell: Basis- und Limit-Register



- 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

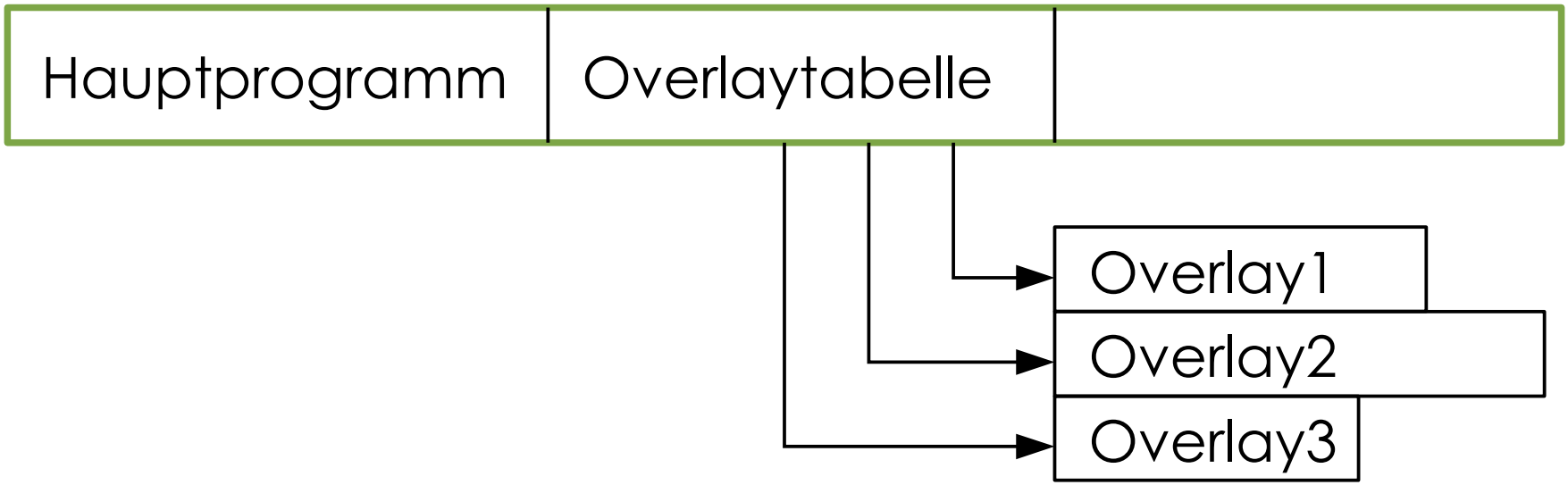
- Verschnitt hoch
  - ➔ quantitative Ermittlung später
- Programmstartzeiten
- Limitation der Größe eines Prozesses durch verfügbaren Hauptspeicher
- Ein-/Auslagerungszeit
  „ruhende“ Teile
- Platzbedarf auf Externspeicher

Overlays – Überlagerungstechnik

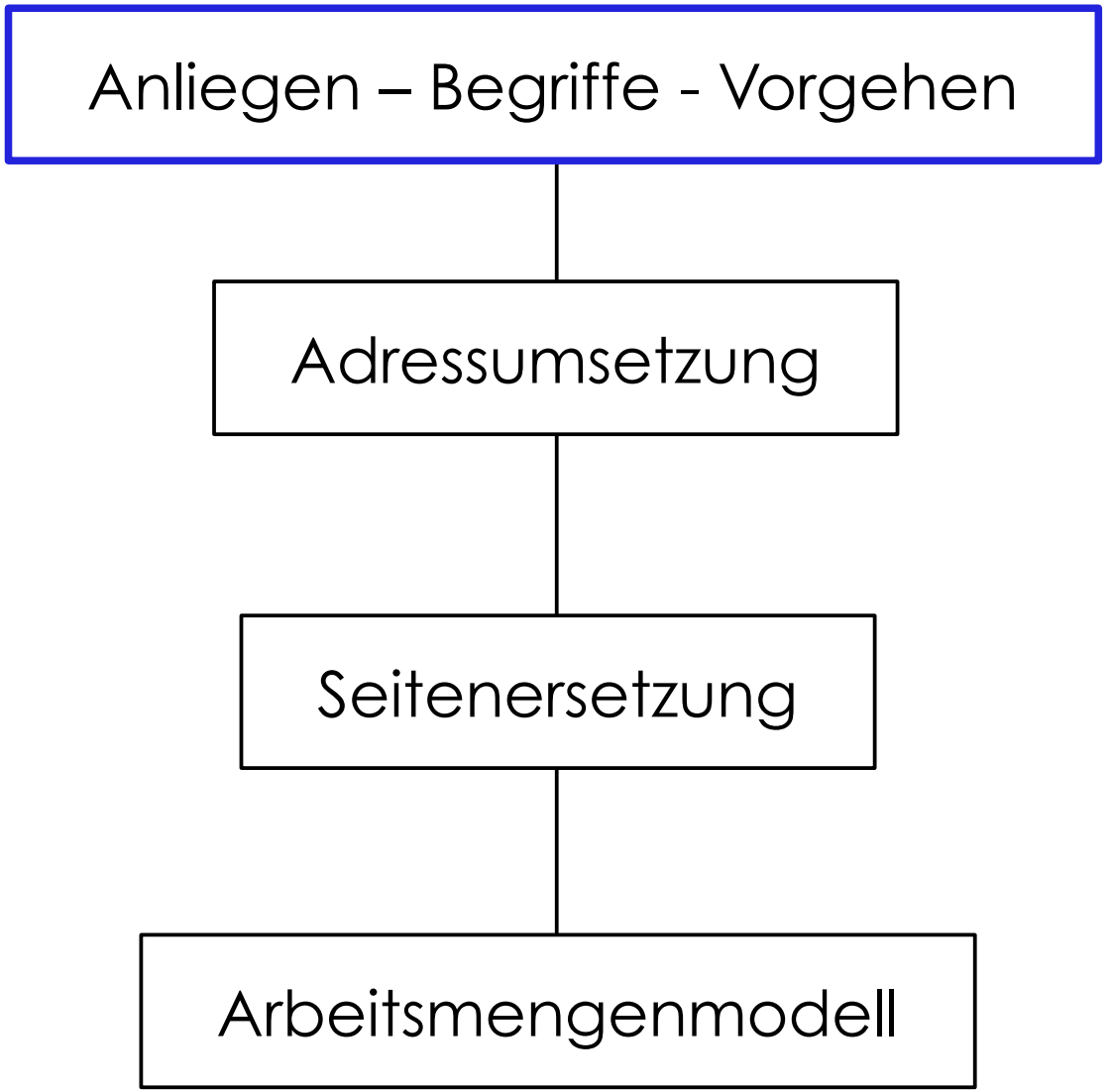
Beliebig große Programme → „Overlays“

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

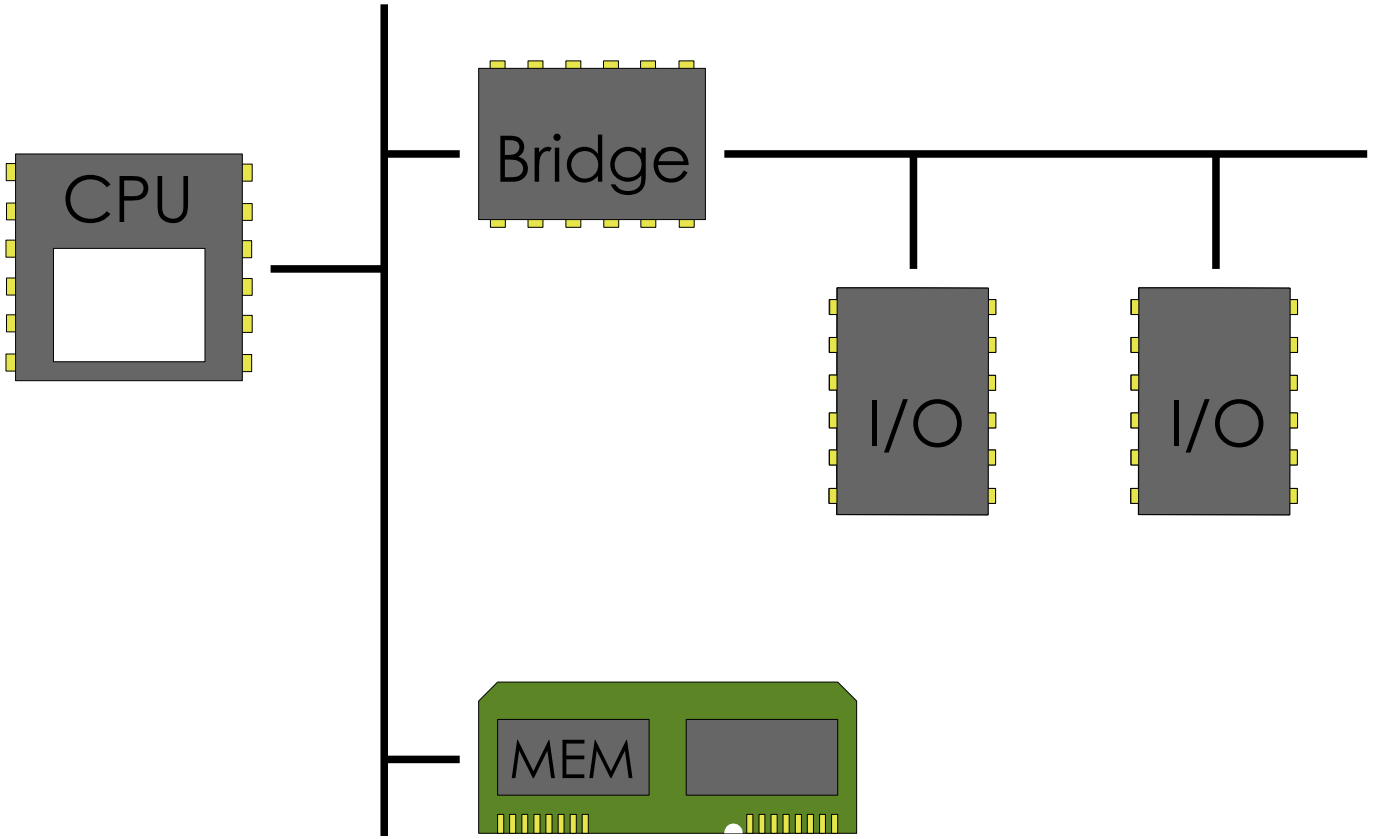
Physischer Adressraum

- 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

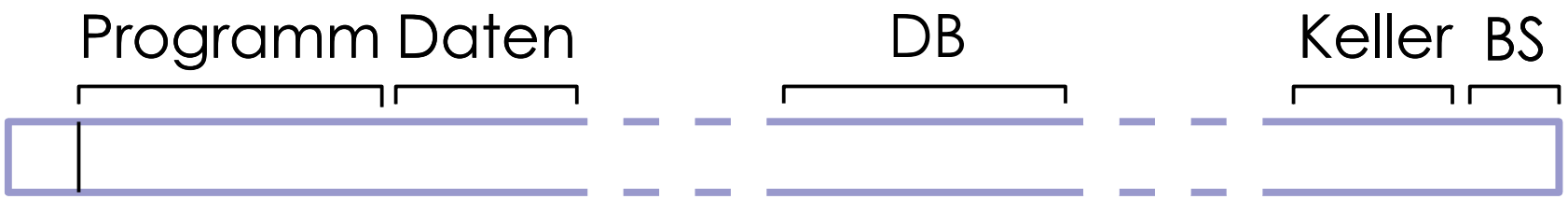
Physische Adressräume



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

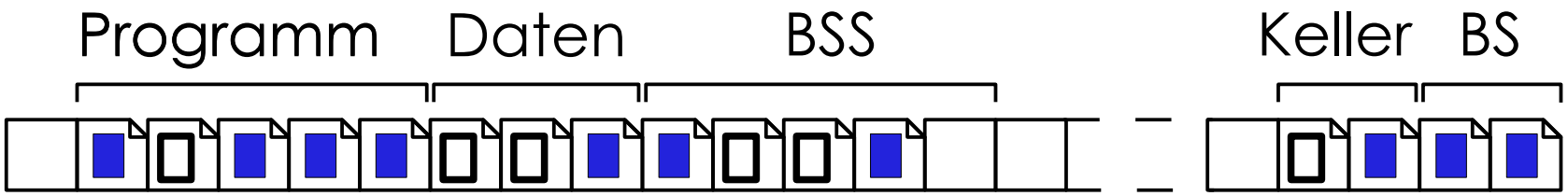
- sequentielle Arbeit eines VON-NEUMANN-Rechners
- Programmcode enthält Zyklen
- Programmierung in Modulen
- Zugriff auf gruppierte Daten

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

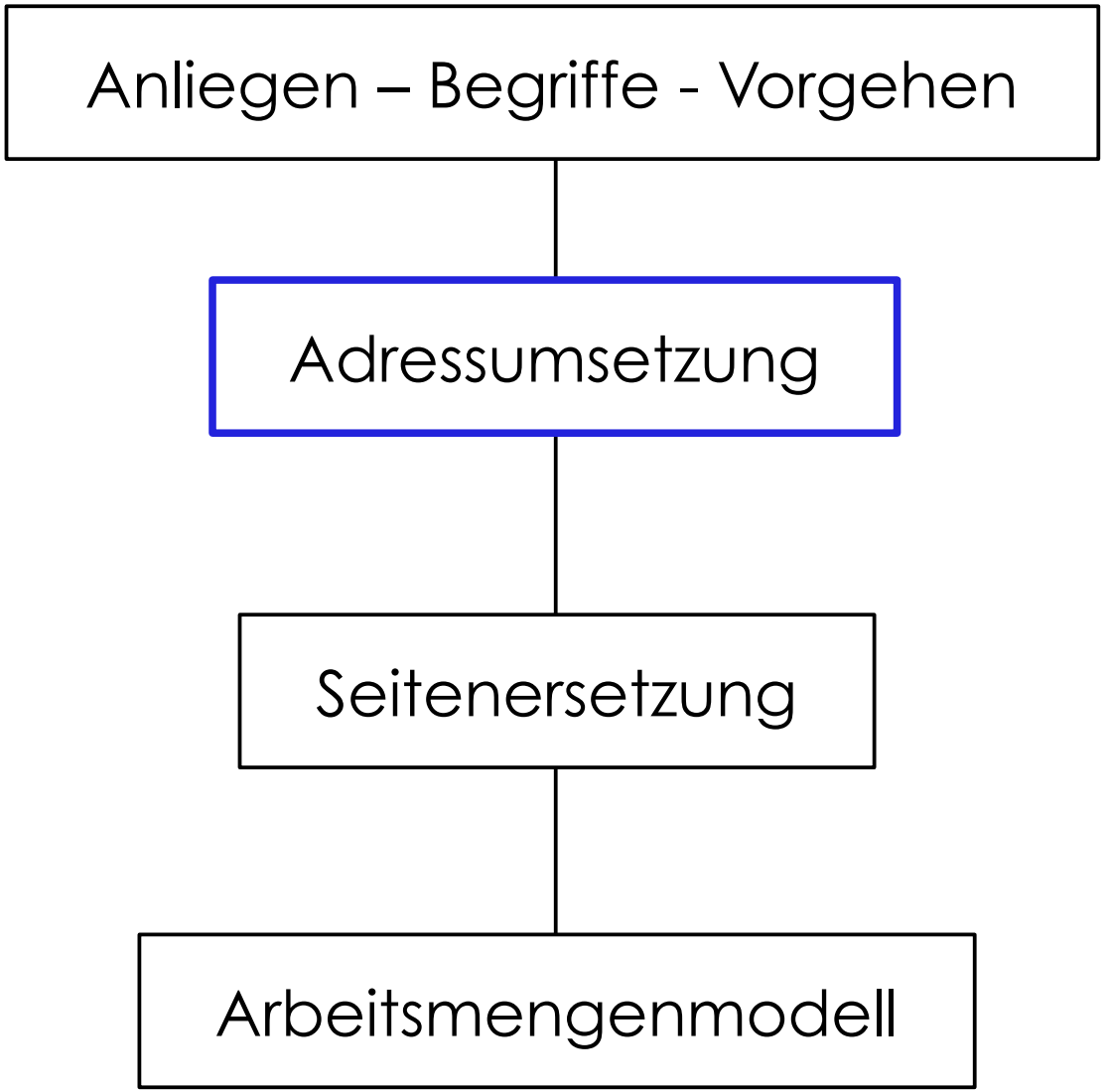
gerade nicht im Hauptspeicher, aber ein gültiger Bereich – z. B. ausgelagert auf Platte

Virtueller Speicher im Betriebssystem

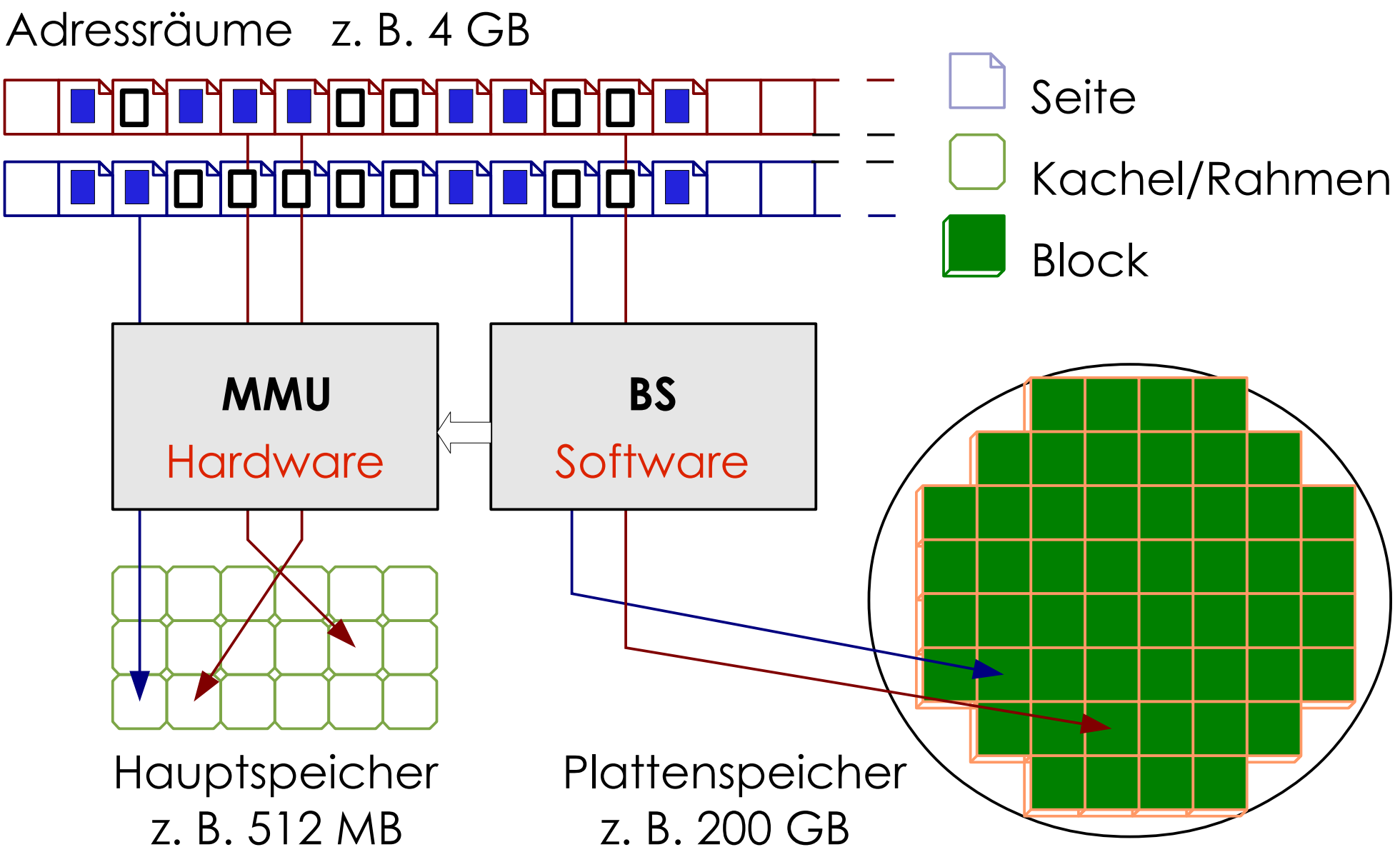
Teilaufgaben

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

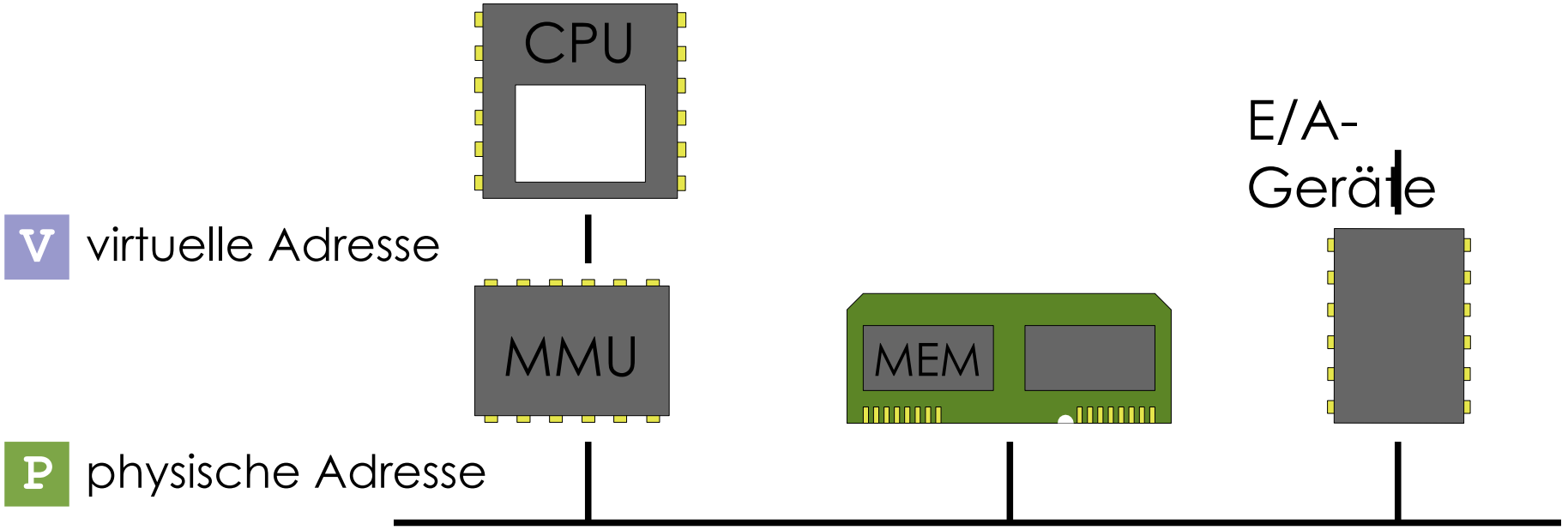
Wegweiser: Virtueller Speicher



Seiten, Kacheln, Blöcke



Rechnerarchitektur: Addressumsetzung etc.



Aufgaben einer MMU (Memory Management Unit)

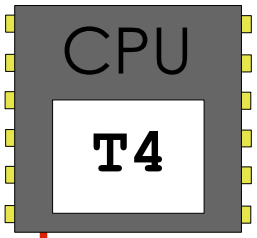
- Abbildung: virtuelle → reale (physische) Adresse
- Schutz bestimmter Bereiche (lesen/schreiben)
- Betriebssystemaufruf bei abwesenden/geschützten Seiten
  → Seitenfehler (page fault)
- Schutz der Adressräume untereinander

Prinzipielle Arbeitsweise einer MMU

virtuelle Adresse

physische Adresse

Prinzipielle Arbeitsweise einer MMU



virtuelle Adresse

Seiten# Offset

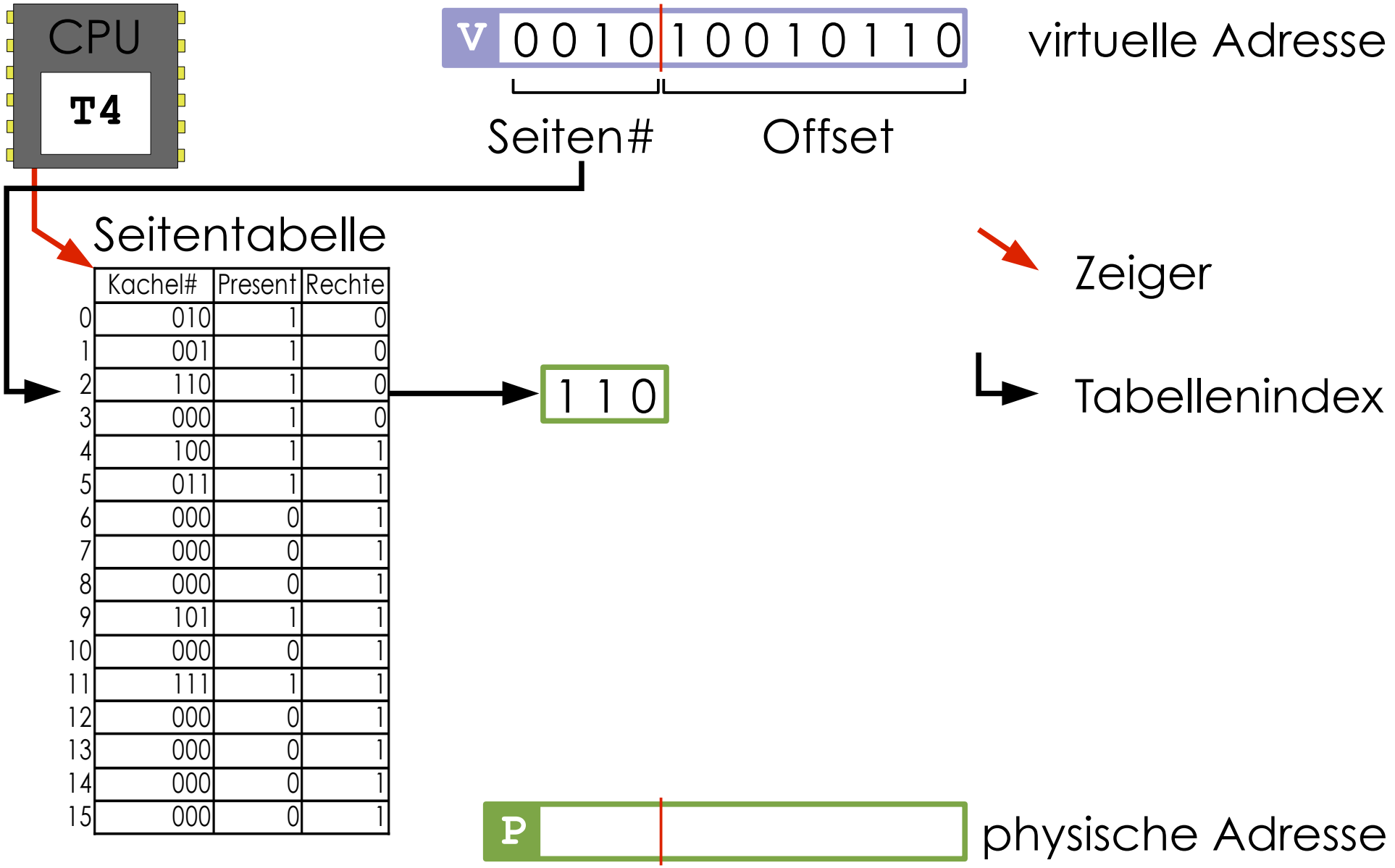
Seitentabelle

	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

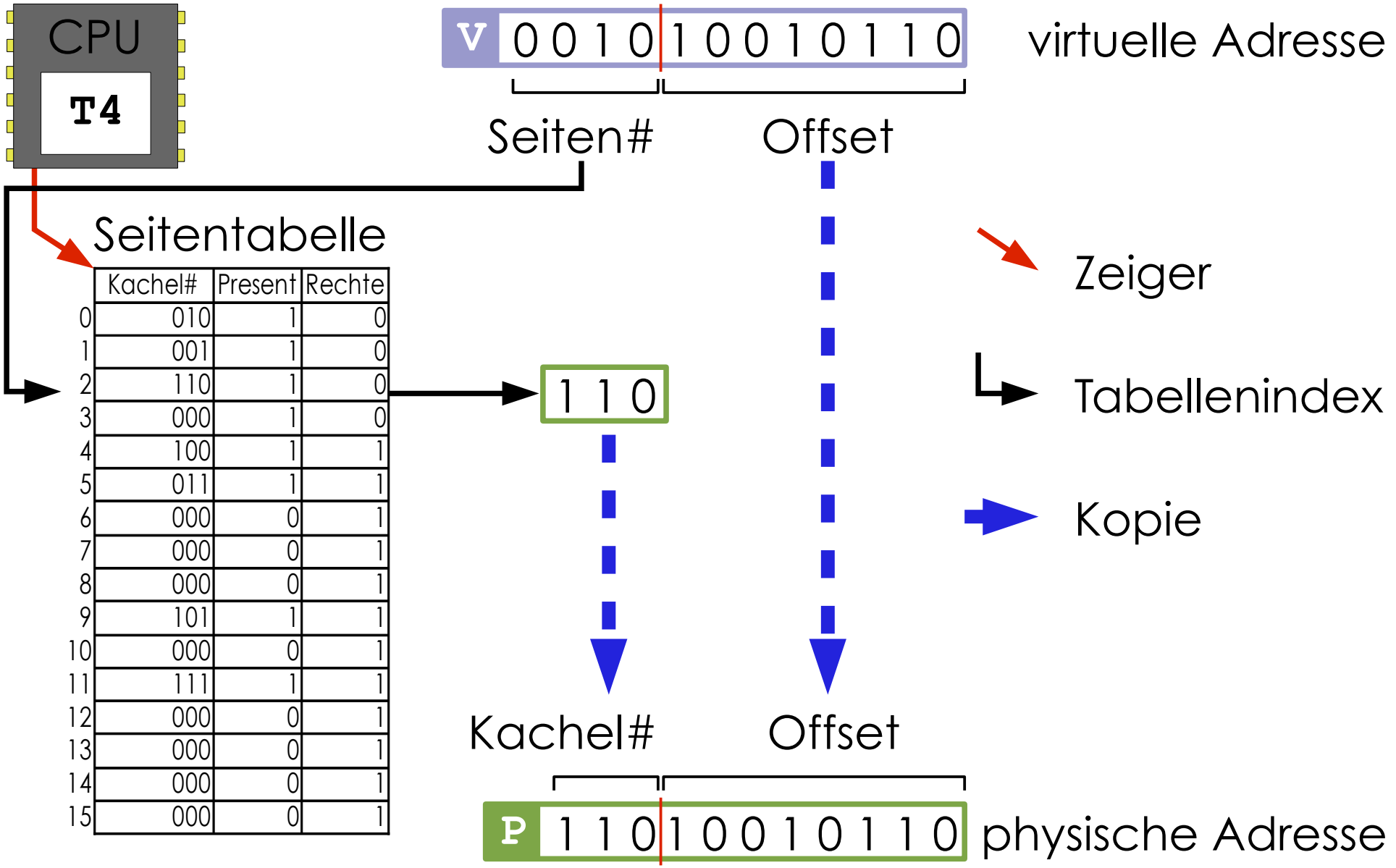
Zeiger

physische Adresse

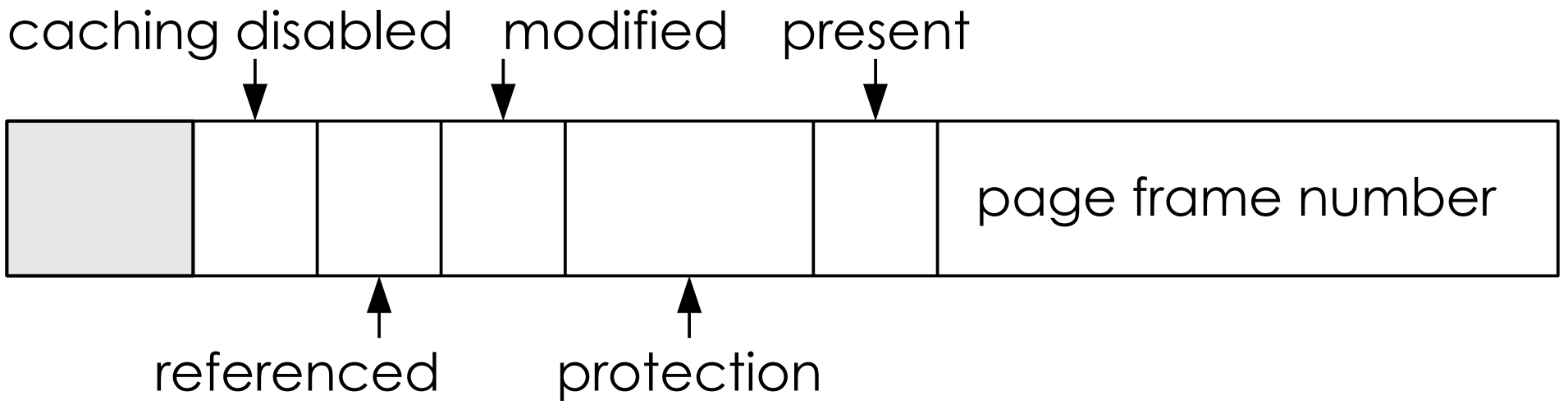
Prinzipielle Arbeitsweise einer MMU



Prinzipielle Arbeitsweise einer MMU



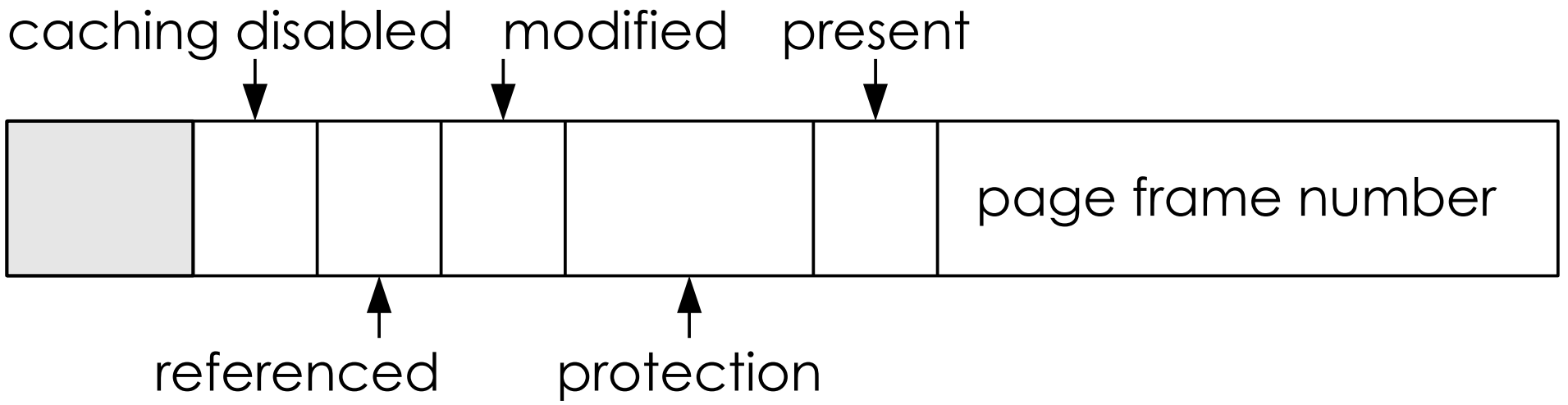
Prinzipieller Aufbau eines Seitentabelleneintrags



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)

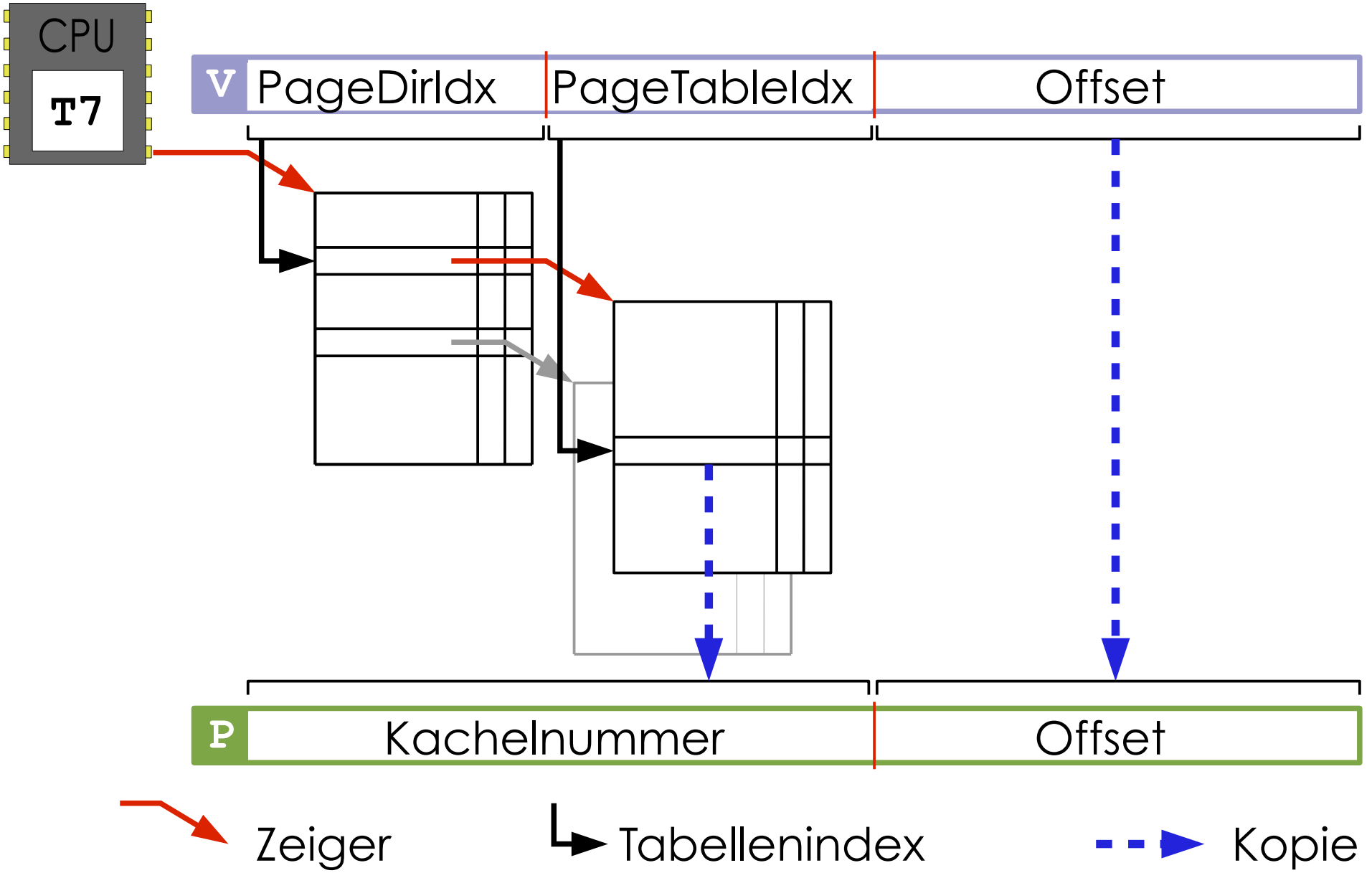
- Größe der Seitentabelle für einen Adreßraum:
  $2^{20} * 4B \rightarrow 4$ MB
- Bei 32 Prozessen 4*32 MB für die Seitentabellen !

Problem 2: Geschwindigkeit der Abbildung – ein Beispiel

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

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

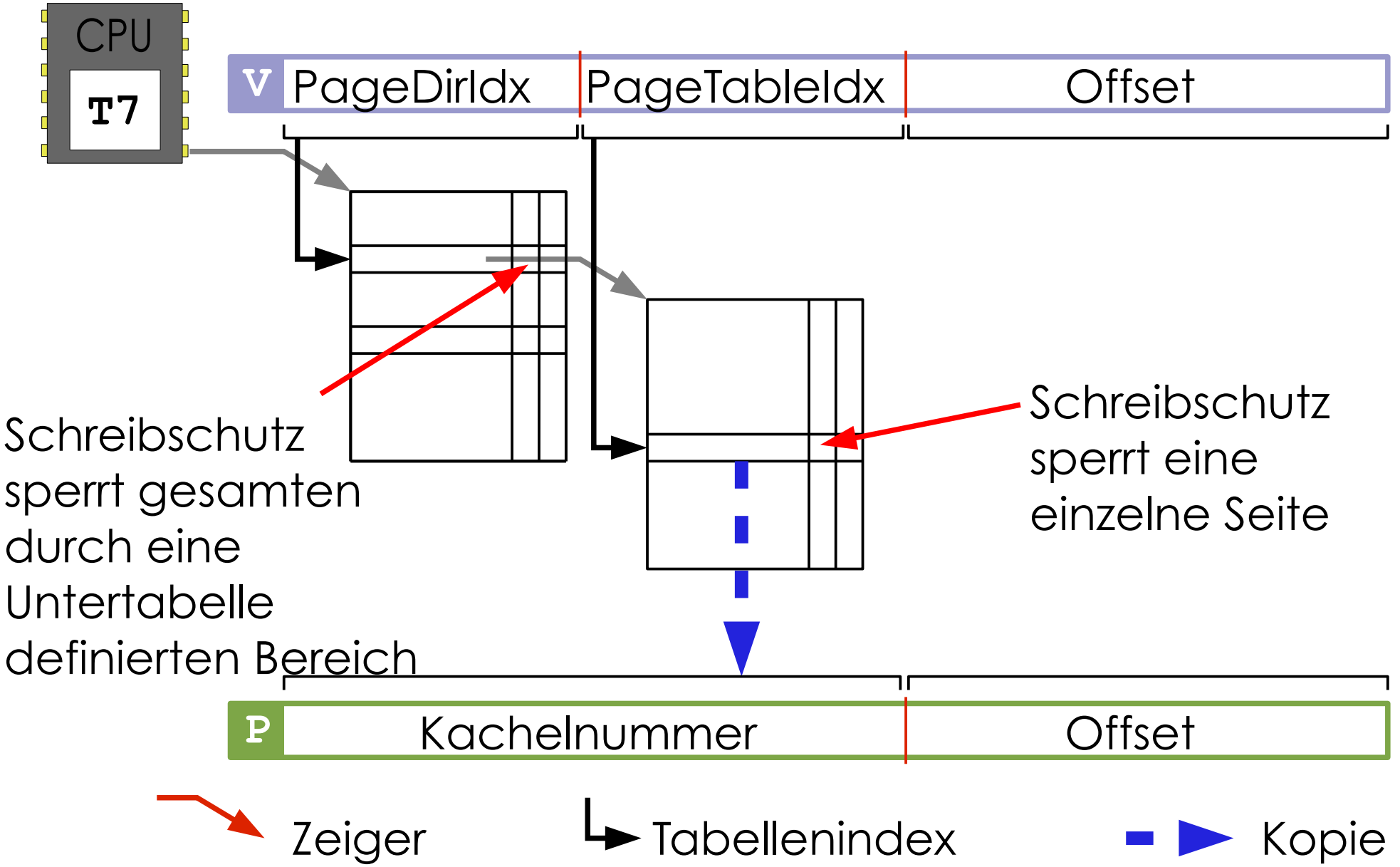
Problem 1: Baumstrukturierte Seitentabellen



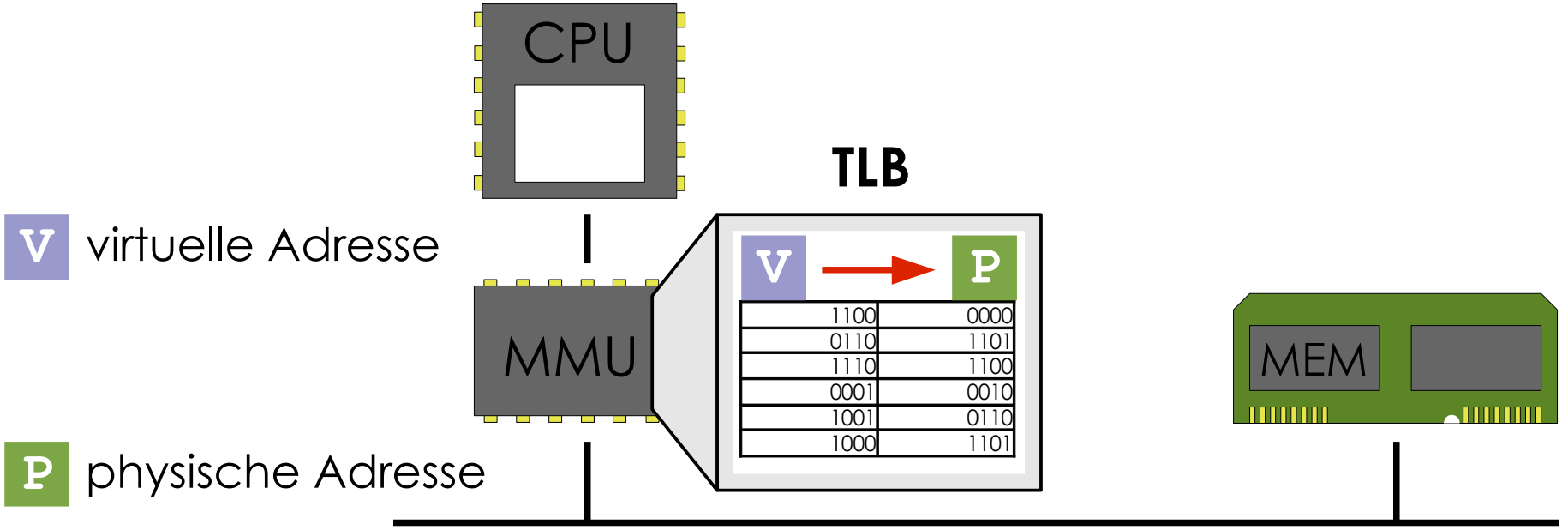
Eigenschaften baumstrukturierter Seitentab.

- 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

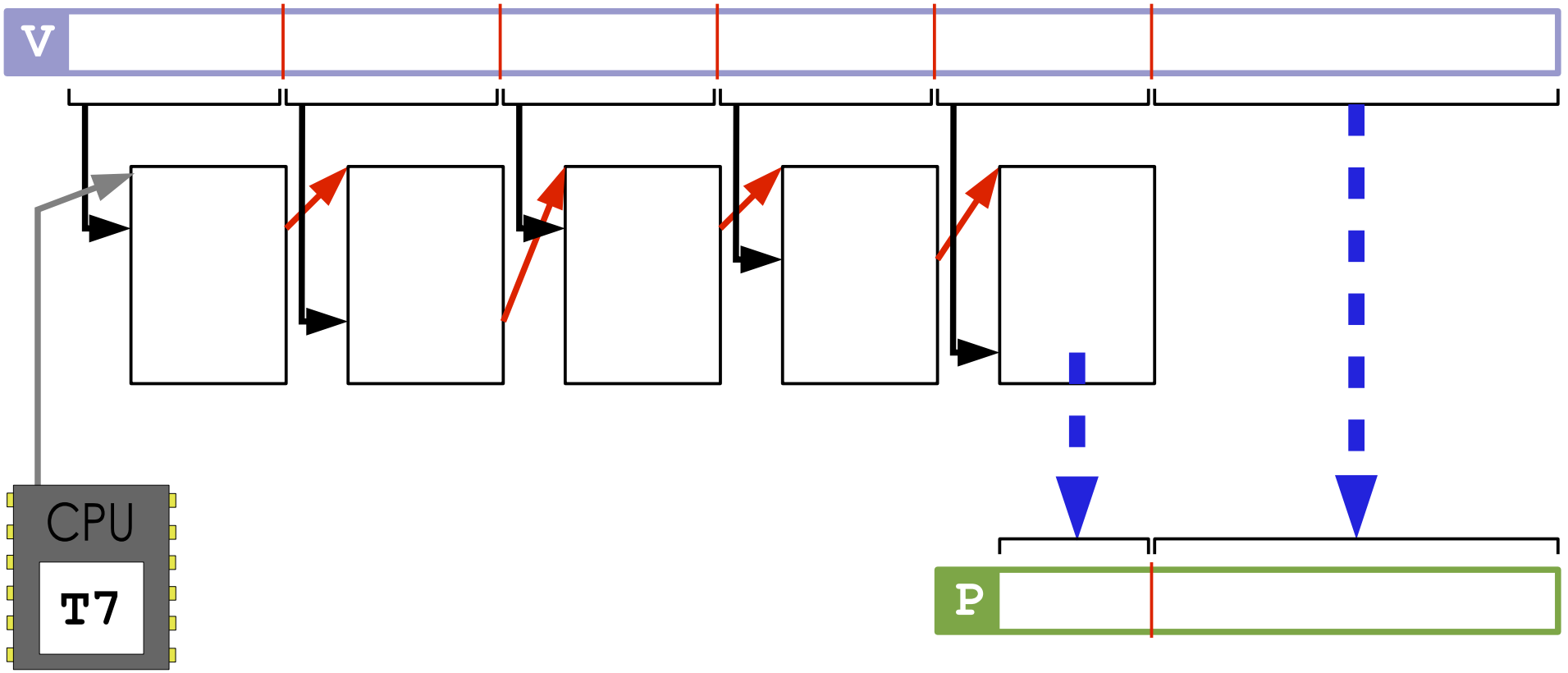
- Zugriff nur über TLB, bei TLB-Fehler („TLB-miss“) → Seitenfehler
- Seitenfehlerbehandlung in SW lädt TLB neu

➔ Vorteil: total flexibel

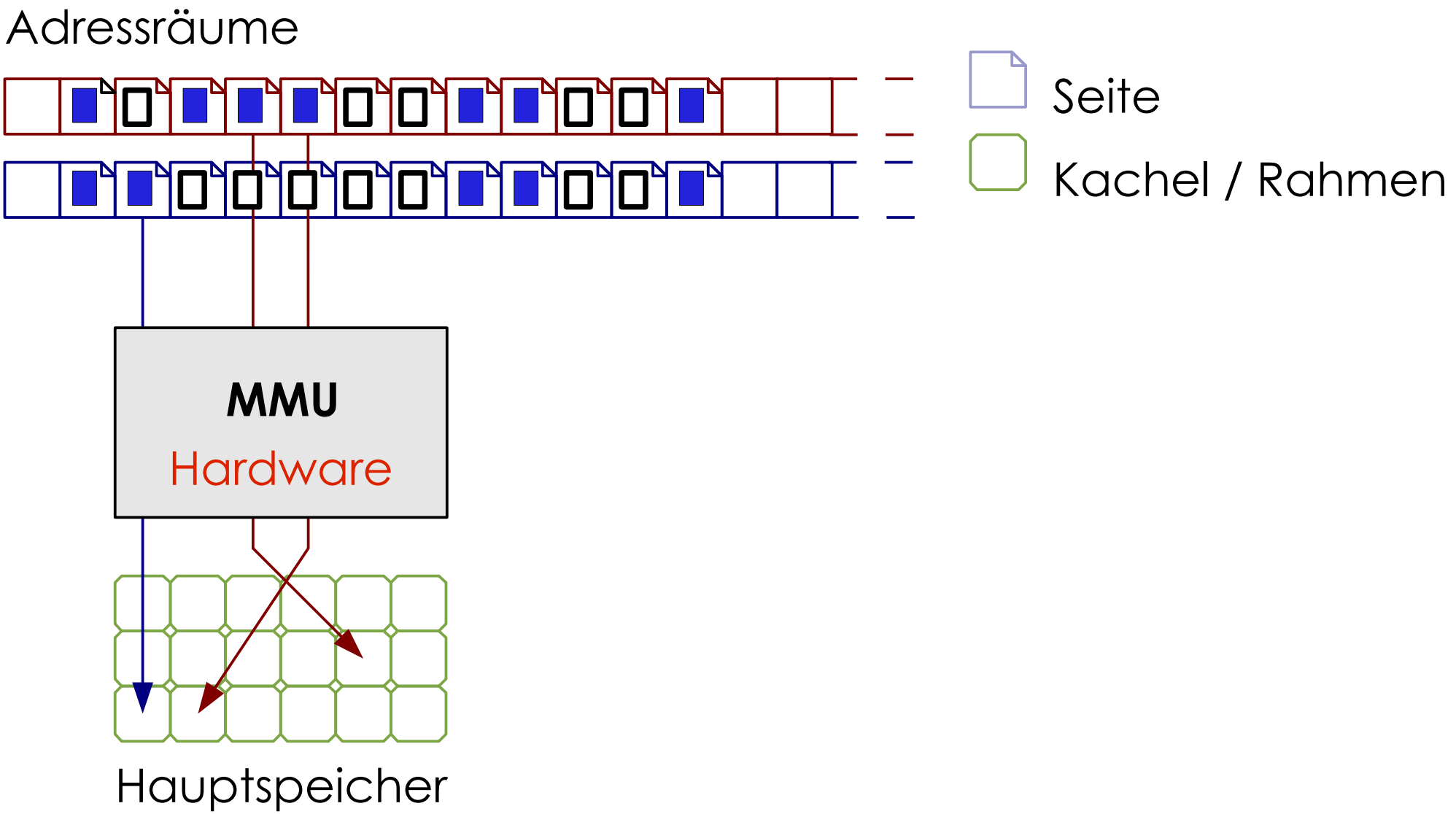
➔ Nachteil: häufigere SW-Seitenfehlerbehandlung

Nochmal Problem 1

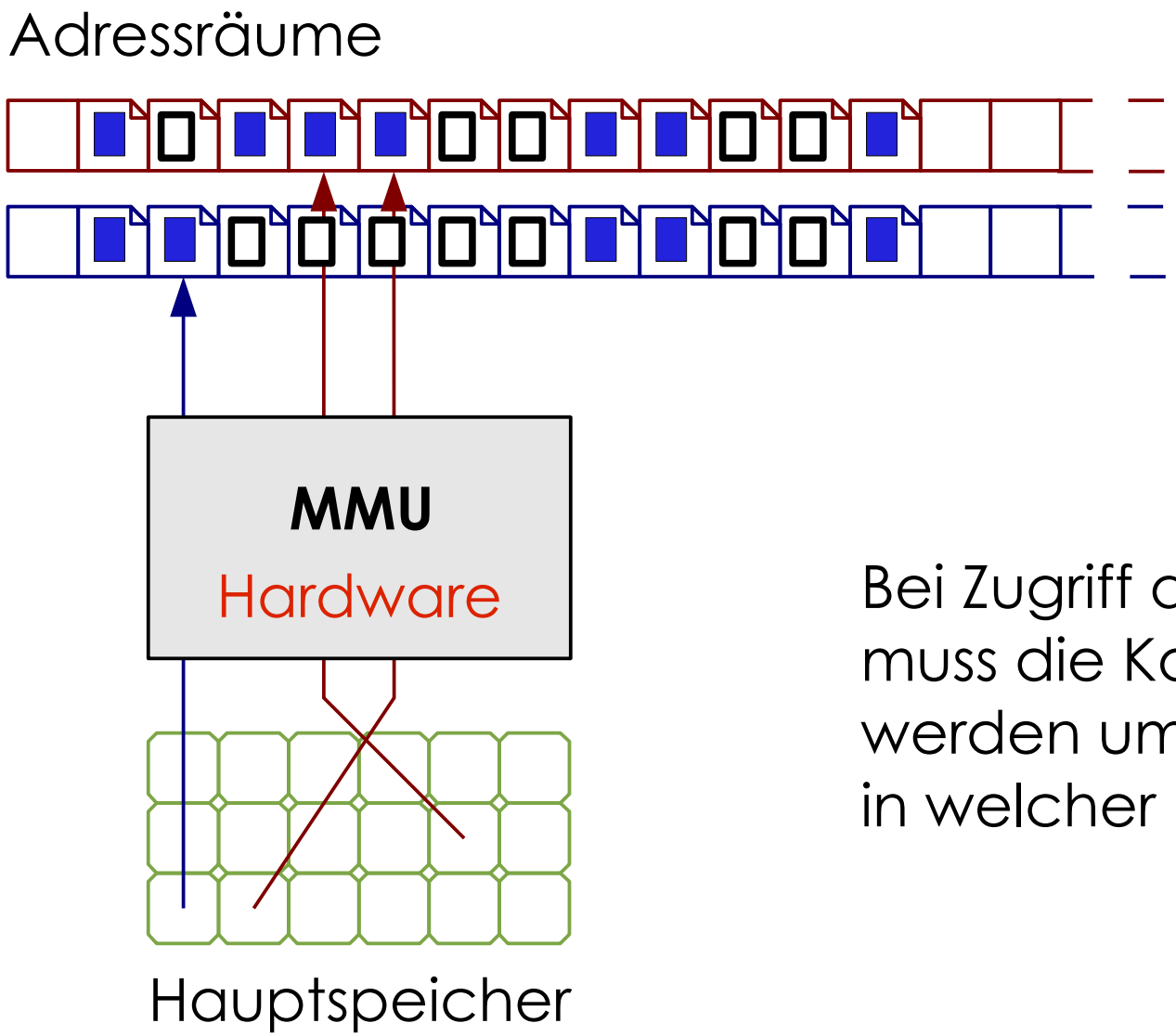
- Problem: riesige Seitentabellen auch bei baumorientierten Seitentabellen
  - bei CPU mit großen Adressräumen (64-Bit-Adressen)
  - bei lose besetzten Adressräumen



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

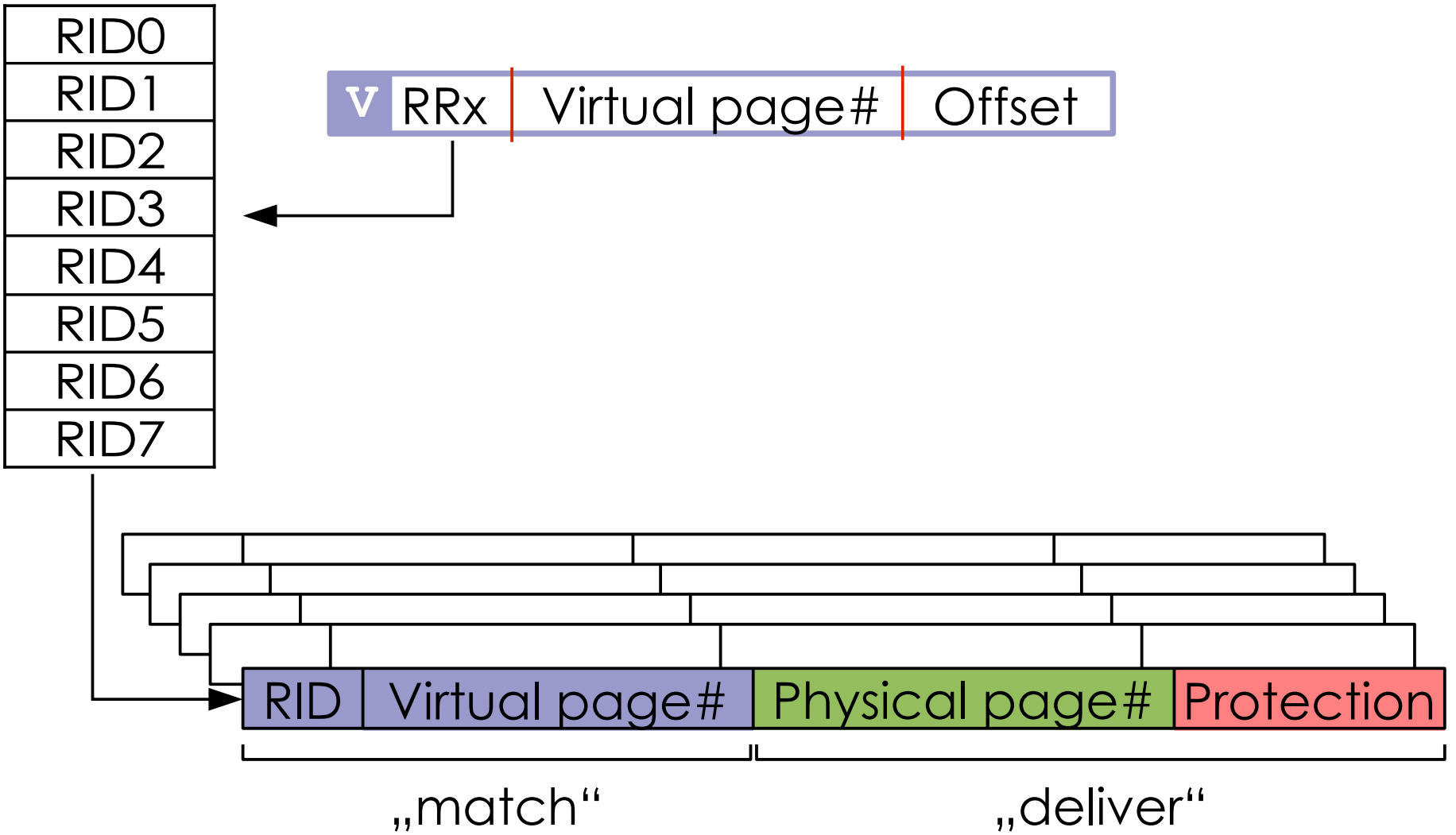
- zu jeder Kachel wird Prozess-Id, Seitennummer geführt
- bei Zugriff („TLB-miss“) wird gesucht

Implementierung als Hash-Tabellen (Hashed Page Tables)

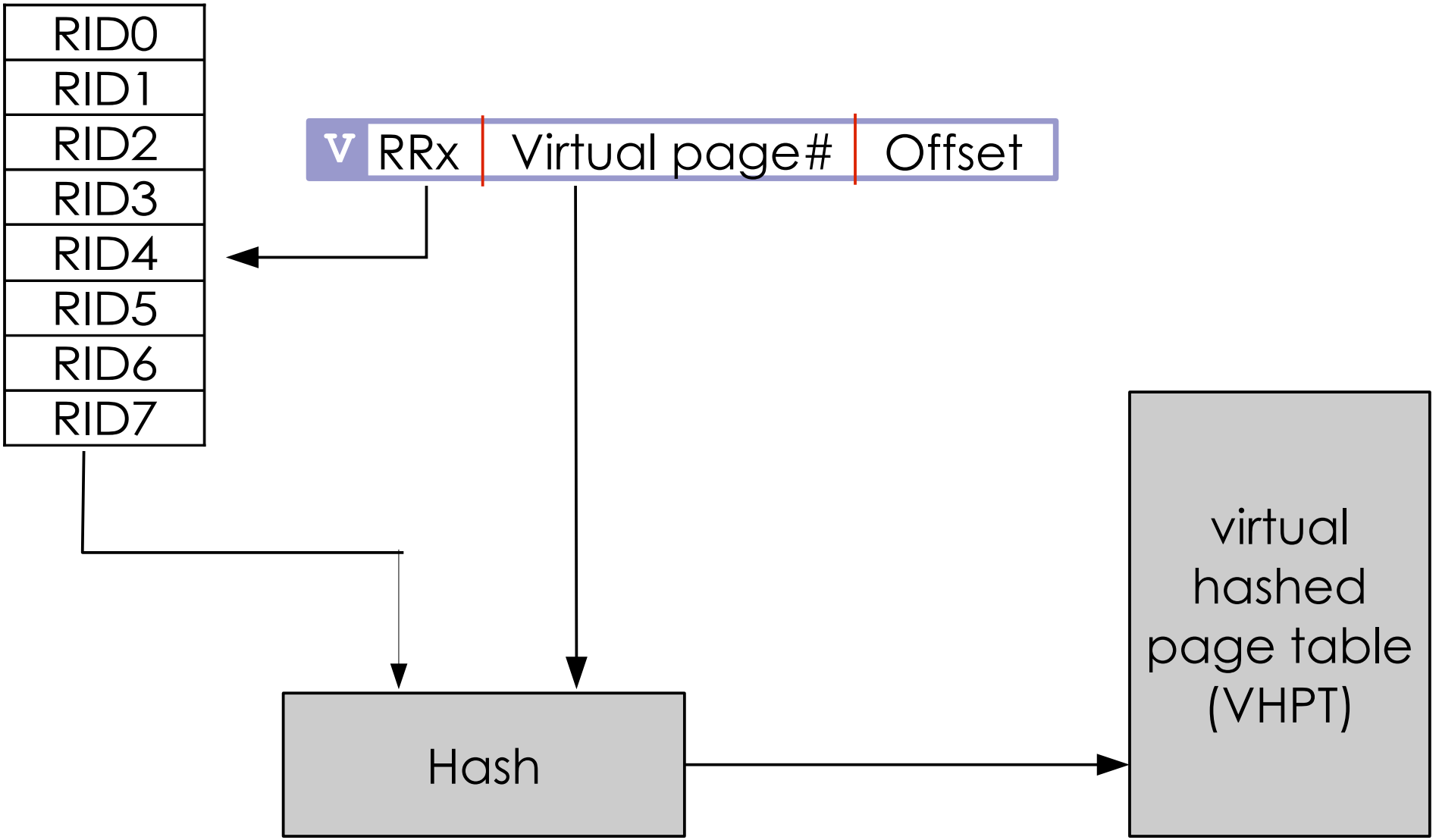
- Vorteile:
  - kleine Seitentabellen
  - abhängig von Anzahl Kacheln
    unabhängig von Größe des Adressraums
- Nachteile:
  - keine Hierarchiebildung (z. B. Schreibschutz für 4 MB)
  - Sharing aufwendig (→ später)
  - Suchaufwand

Beispiel: 64-Bit MMUs - Itanium

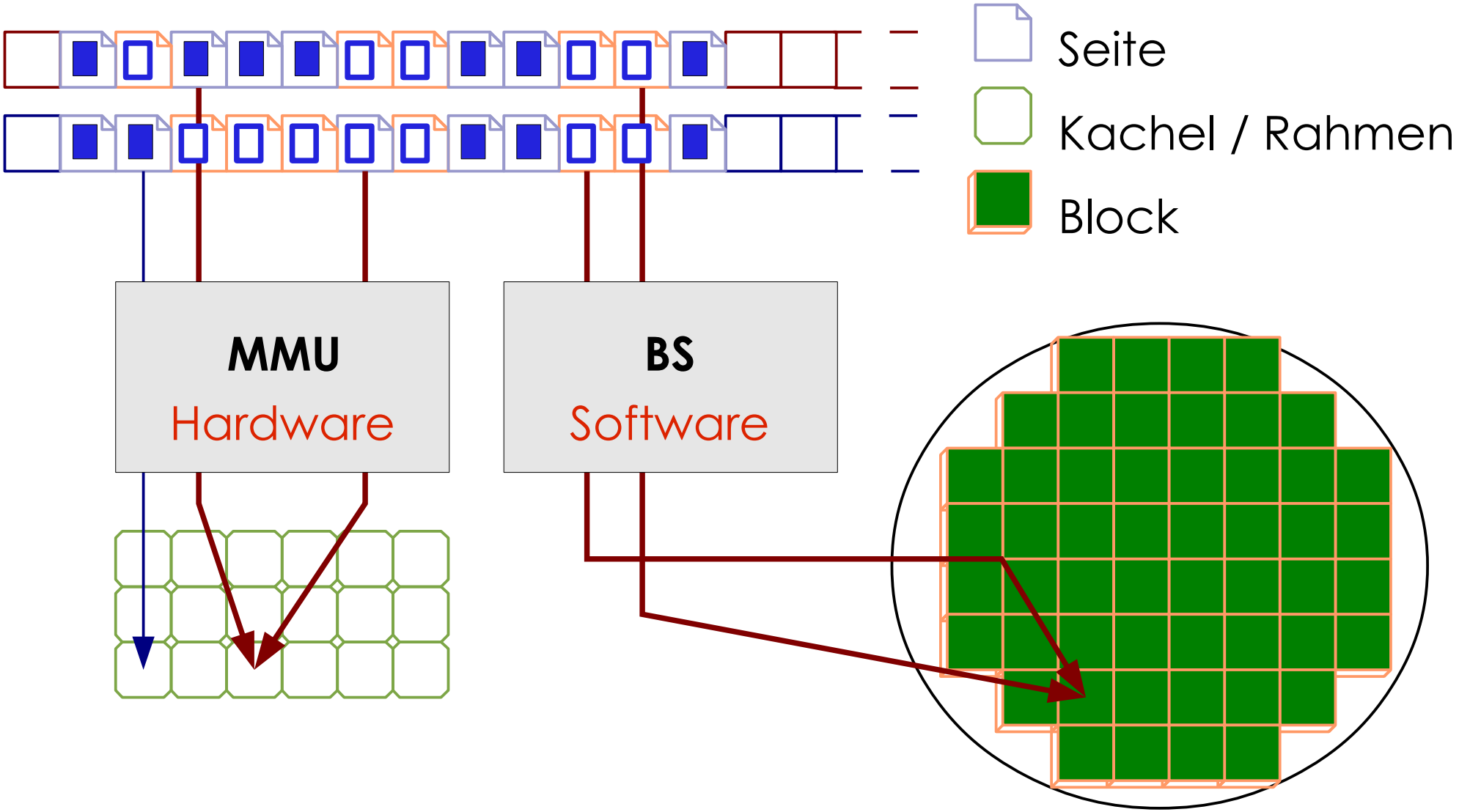
region registers



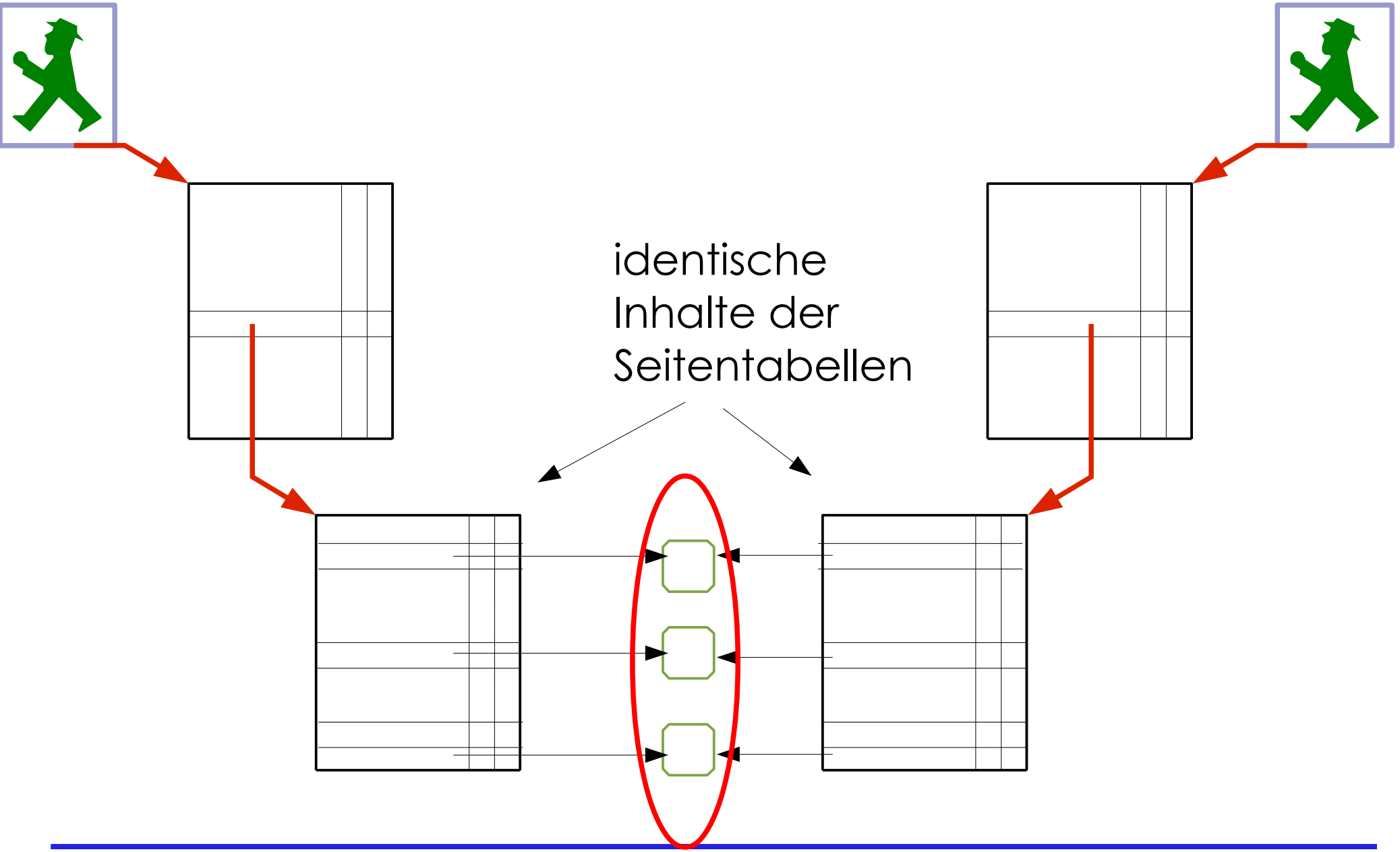
Itanium: Hardware Accessed Page Table



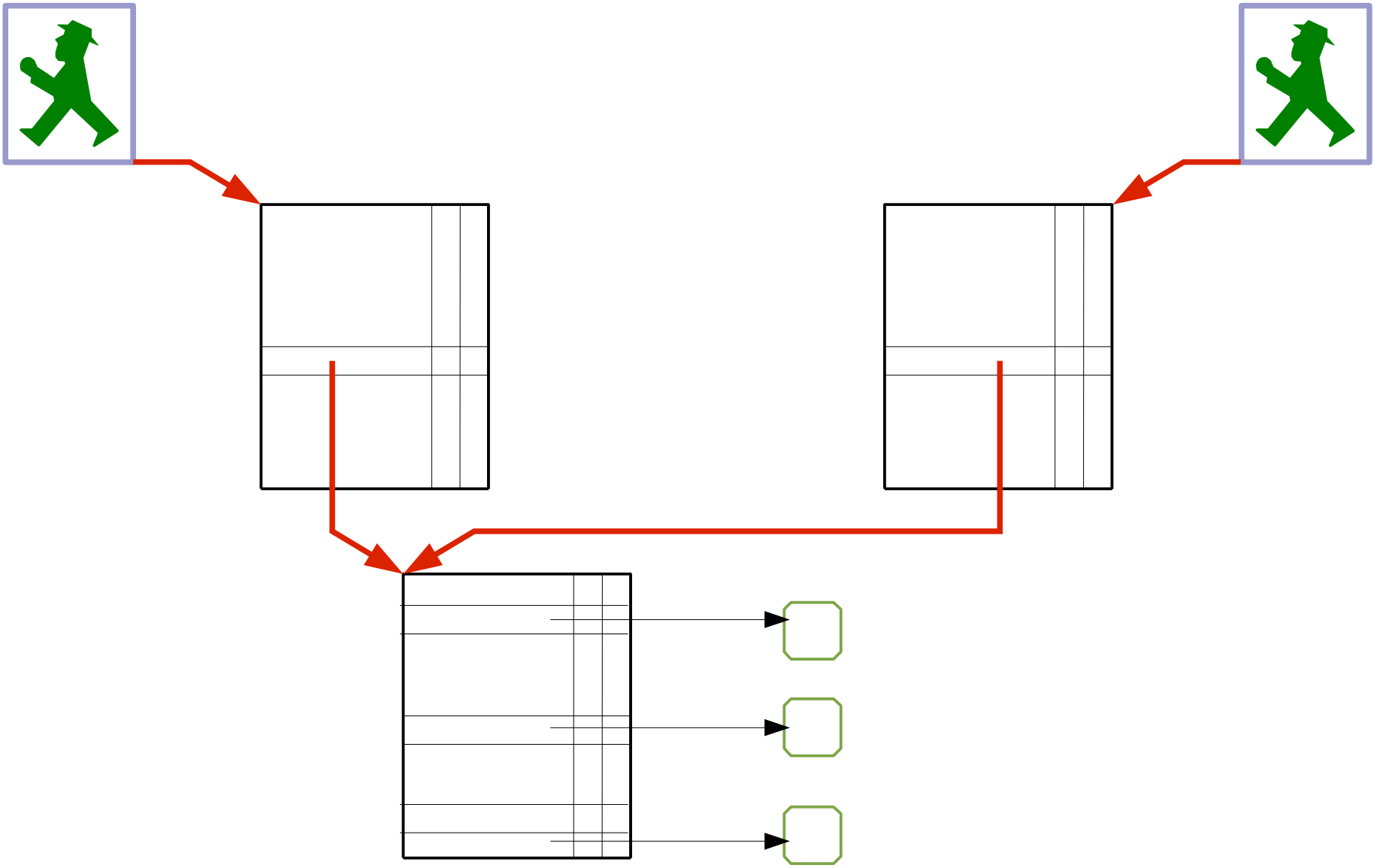
Überlappende Adressräume - „Sharing“



Sharing bei mehrstufigen Seitentabellen



Sharing bei mehrstufigen Seitentabellen



Spezieller Einsatz von Sharing

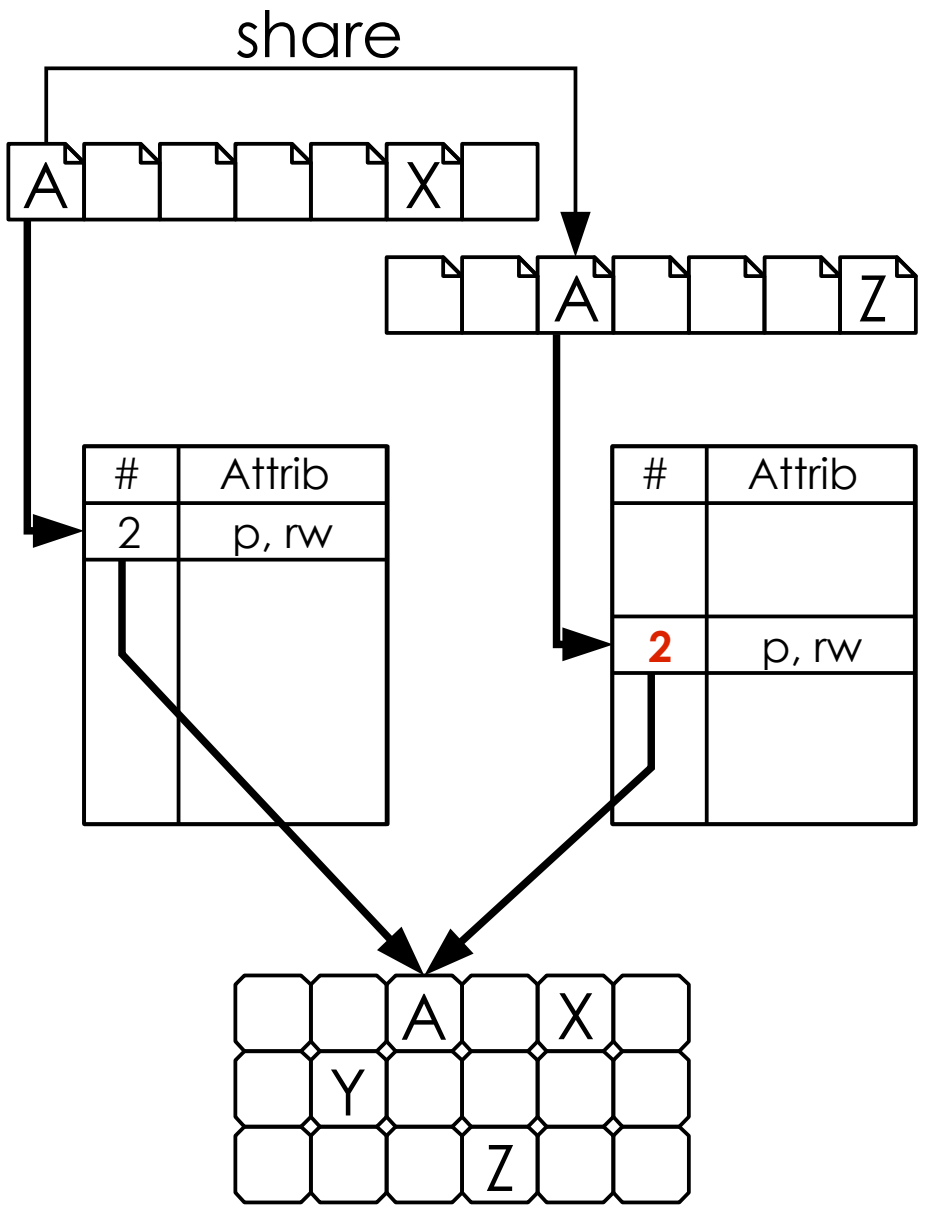
Verzögertes Kopieren (lazy copying, copy on write)

- „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.

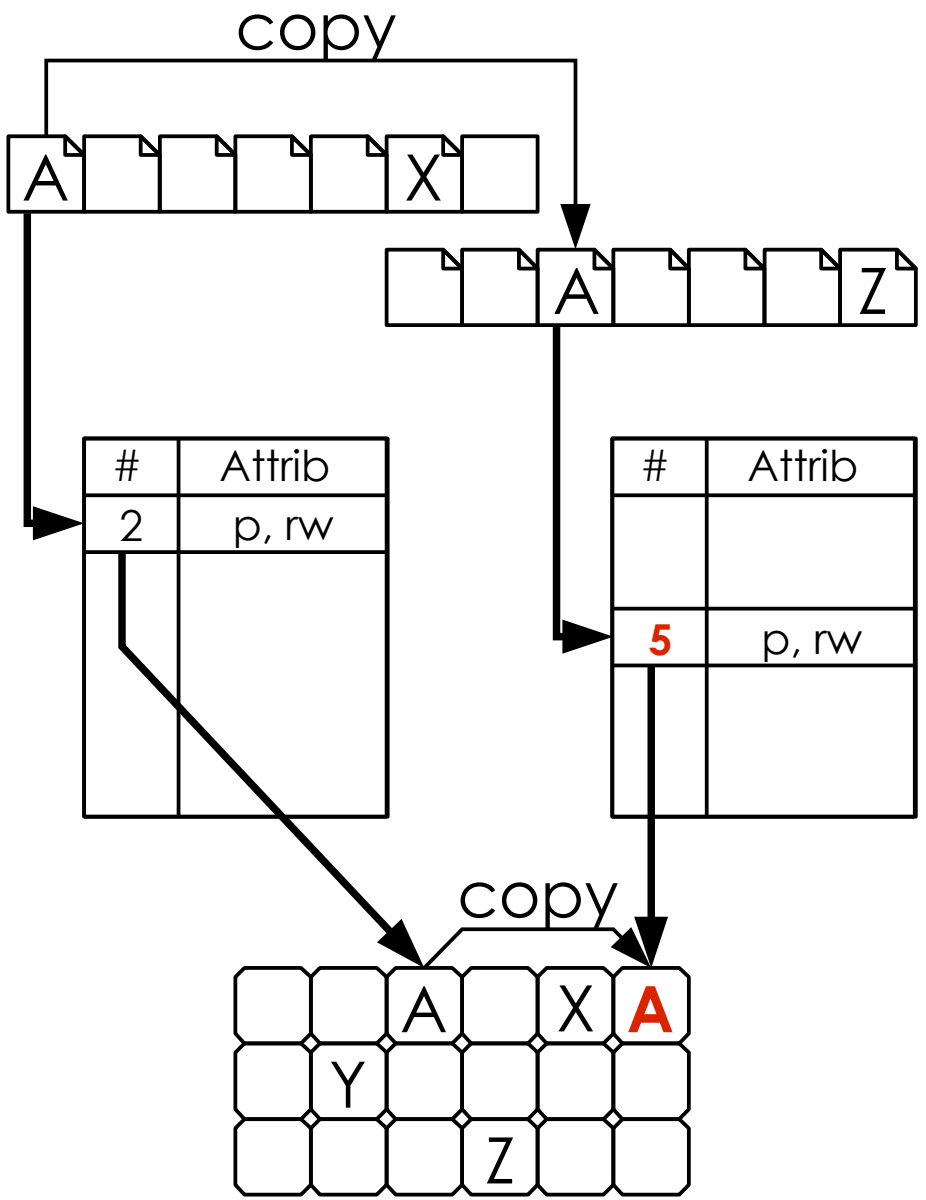
➔ gemeinsame Nutzung von Daten/Programmen

Gemeinsames Nutzen von Daten



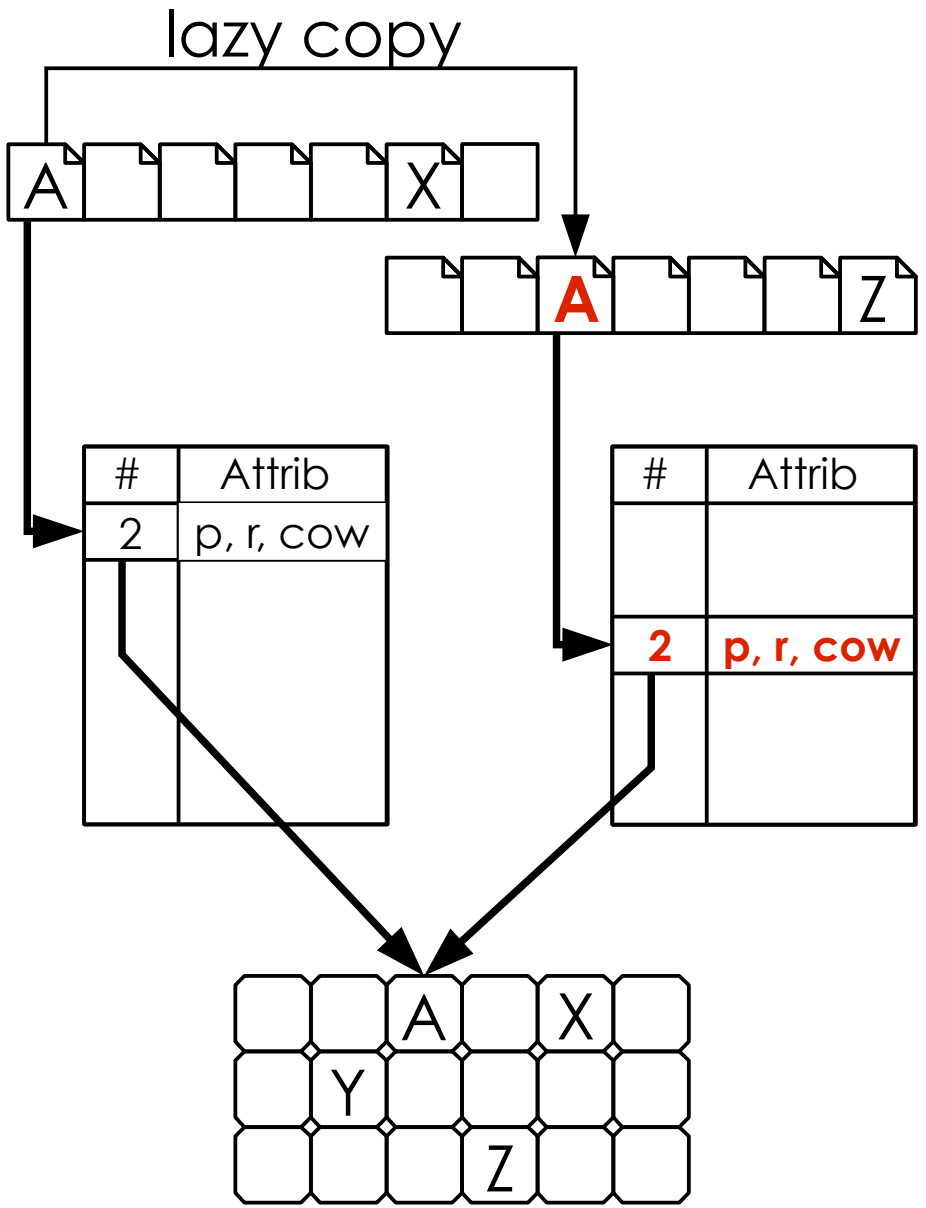
- neuen MMU Eintrag an Zieladresse

Echtes Kopieren



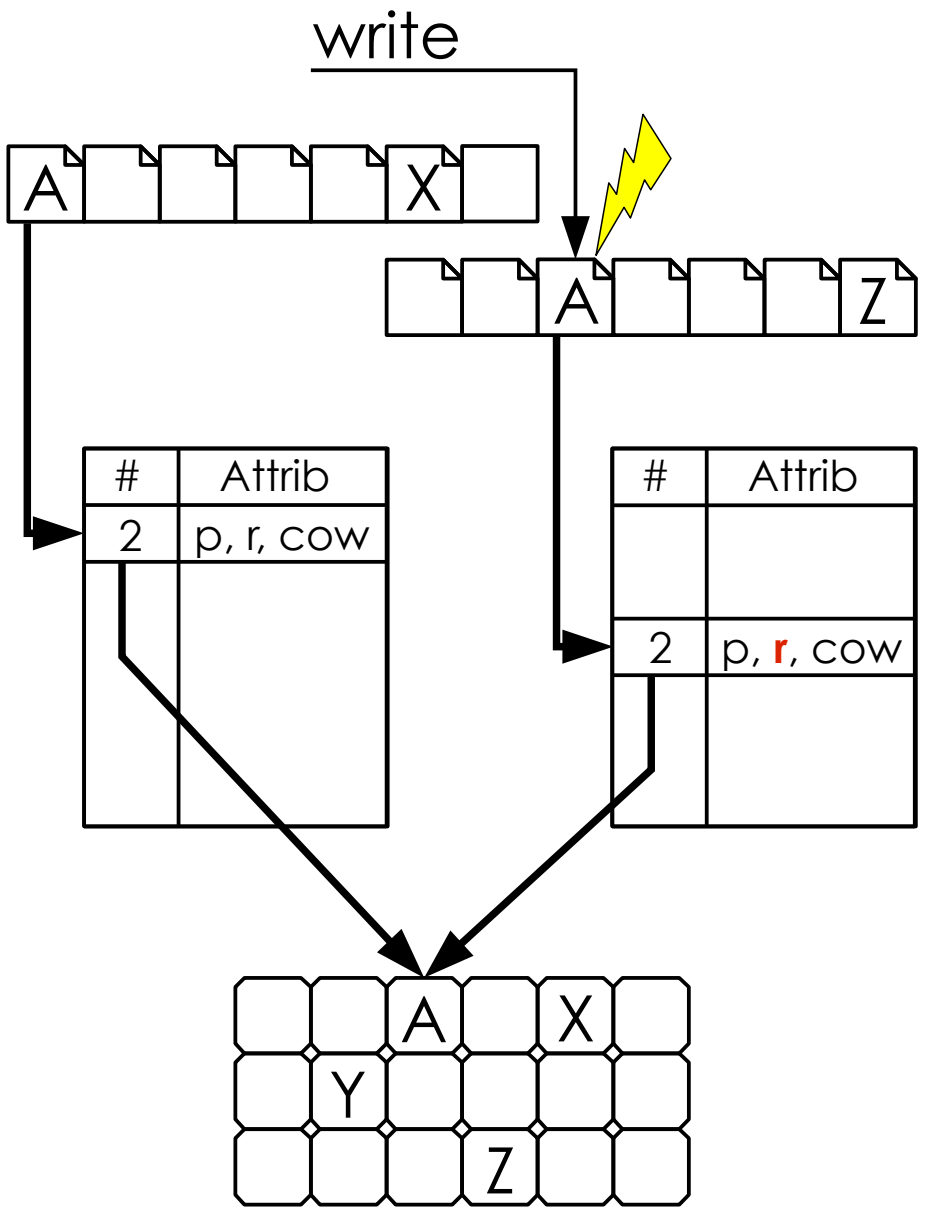
- neue Kachel besorgen
- Kacheln kopieren
- neuen Eintrag in die Seitentabelle für Zieladresse

Verzögertes Kopieren



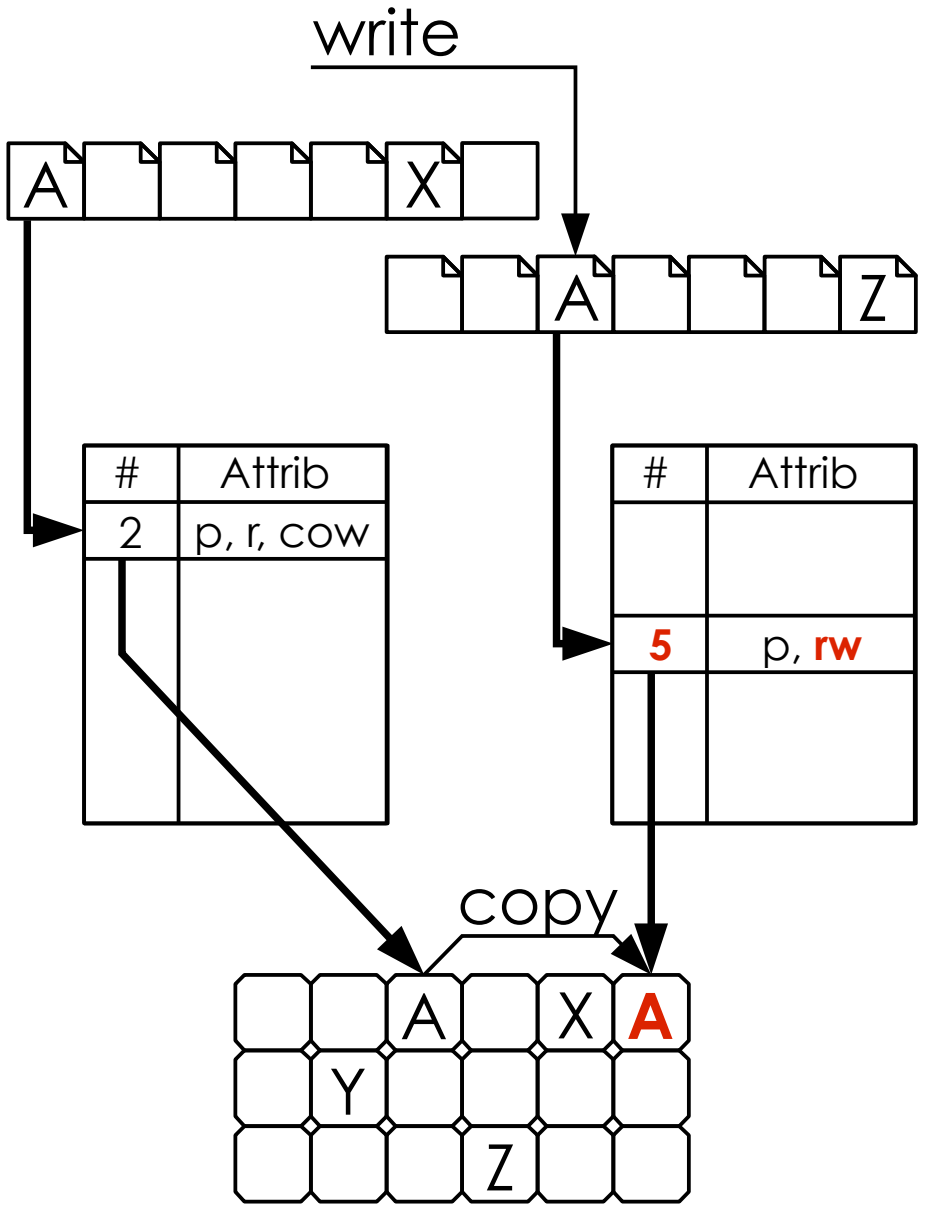
- Ändern des MMU-Eintrags an der Quelladresse: rw → (r, cow)
- neuen MMU-Eintrag an Zieladresse: (r, cow)

Verzögertes Kopieren



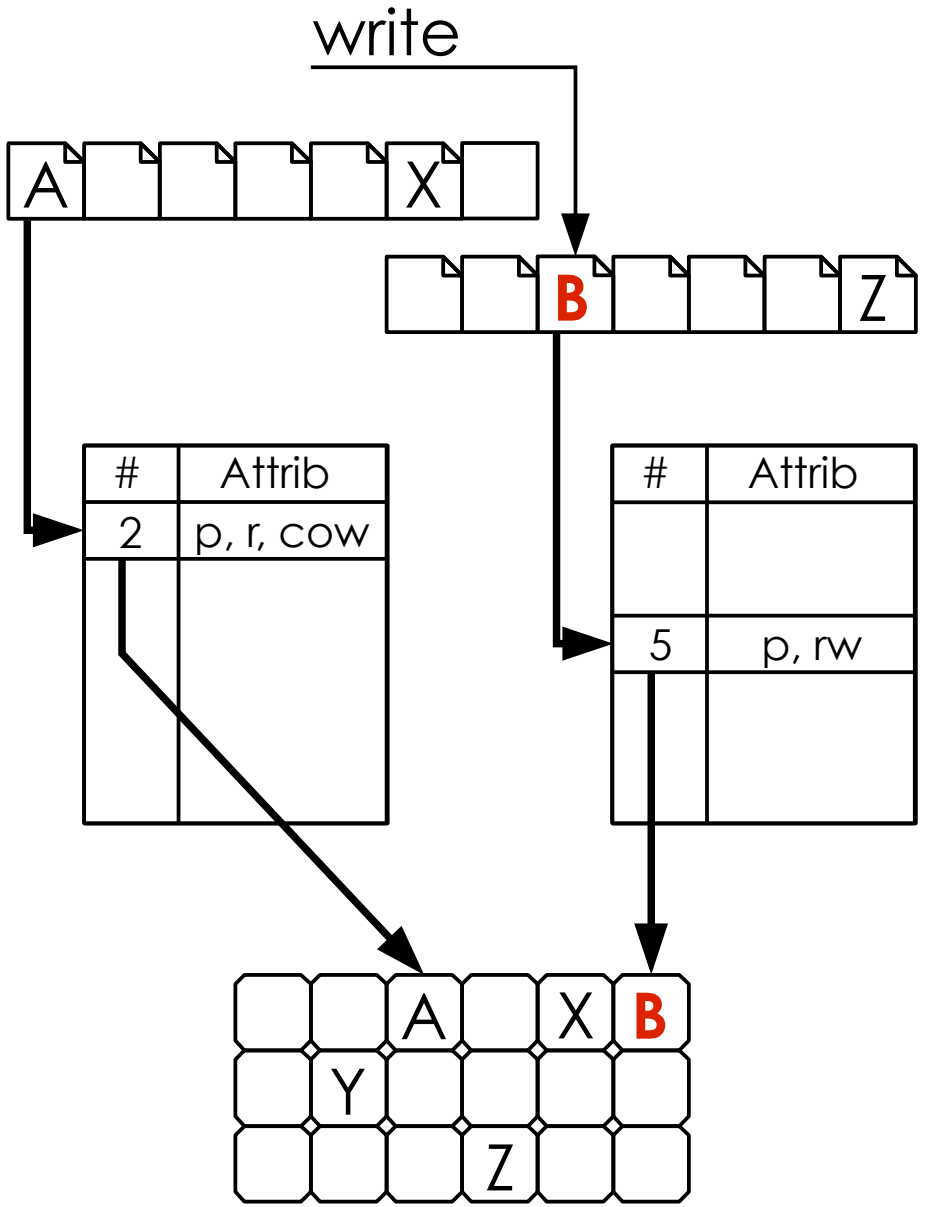
- schreibender Zugriff → Seitenfehler

Verzögertes Kopieren



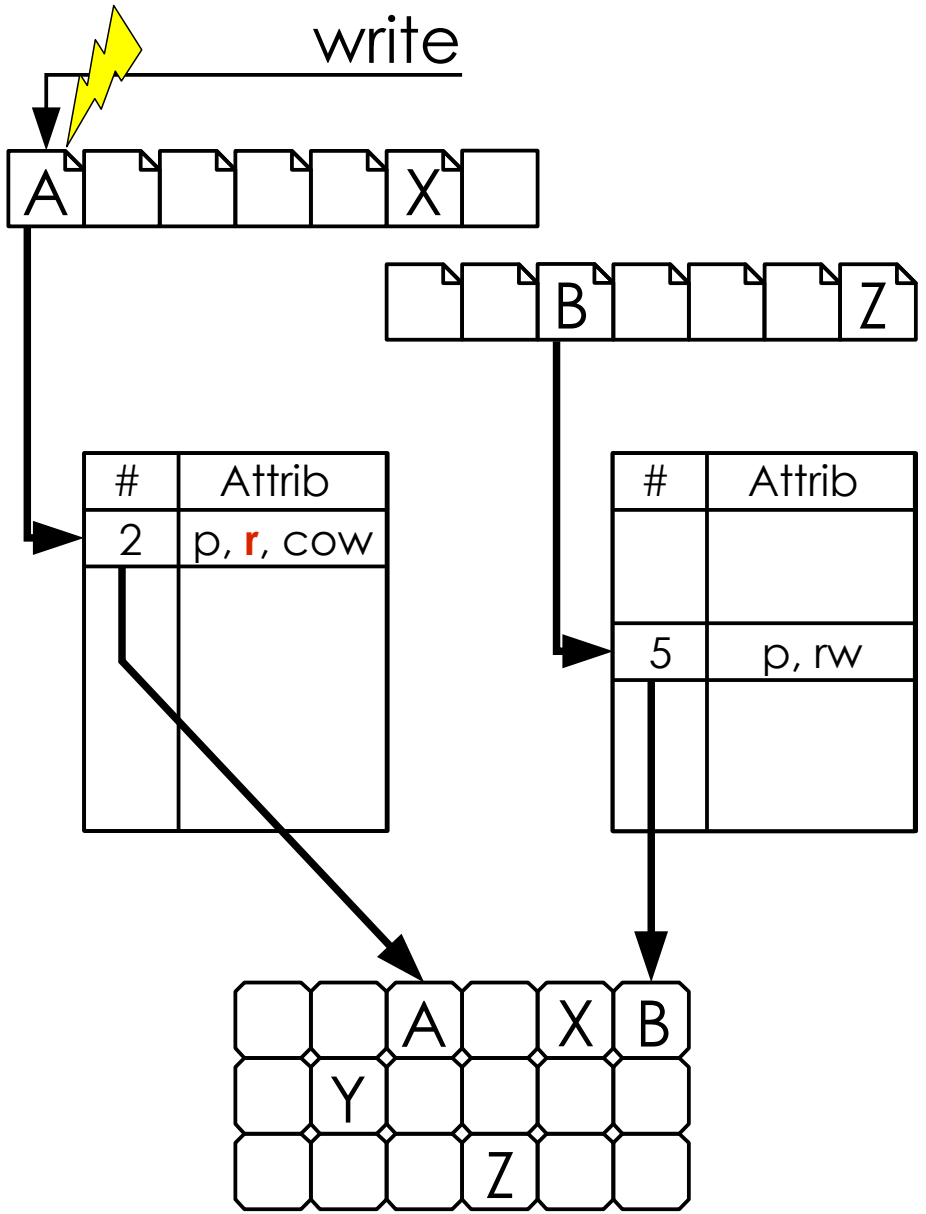
- schreibender Zugriff → Seitenfehler
- Das cow-Attribut bewirkt, dass eine neue Kachel allokiert und der Inhalt physisch kopiert wird
- neuen MMU-Eintrag an Zieladresse: rw

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

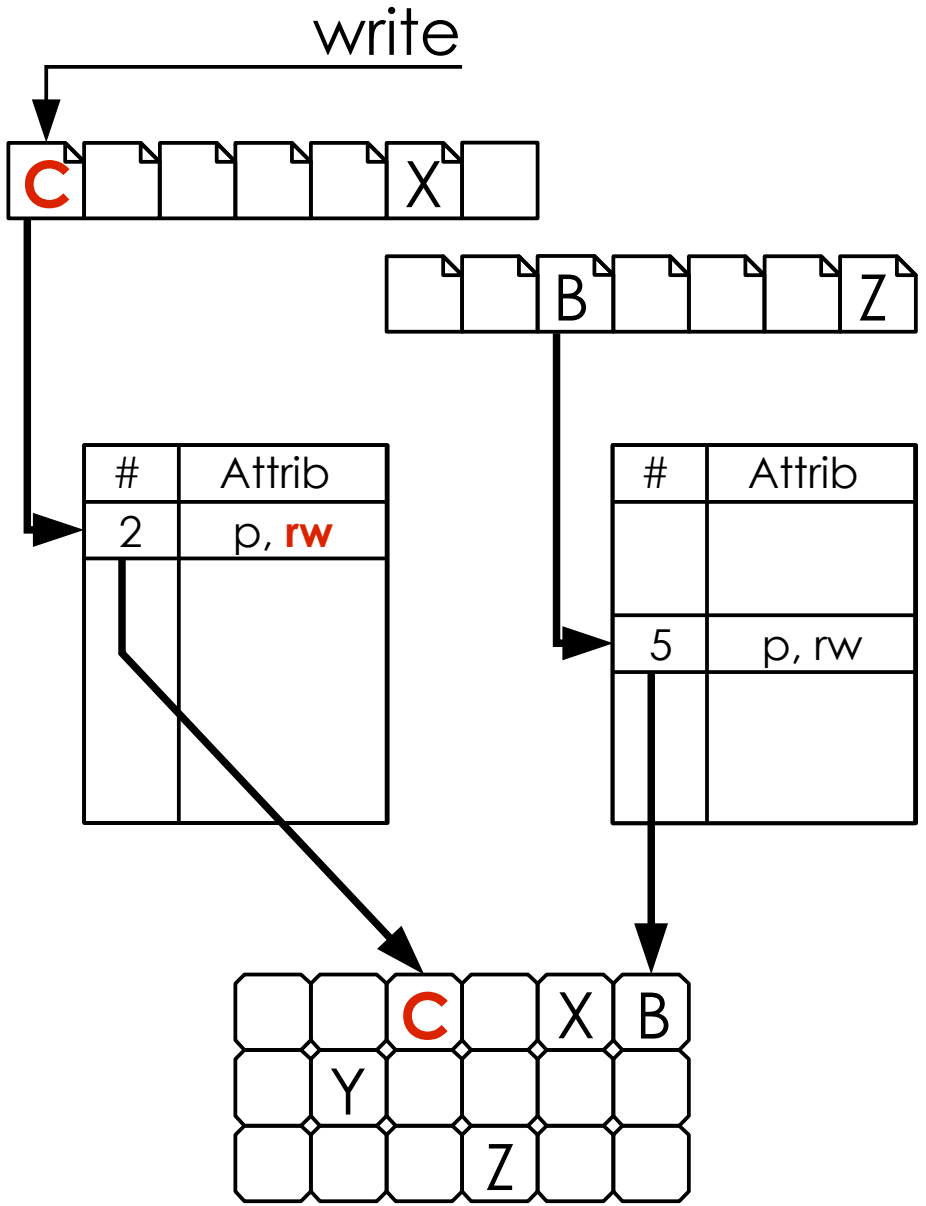
Verzögertes Kopieren



- schreibender Zugriff → Seitenfehler

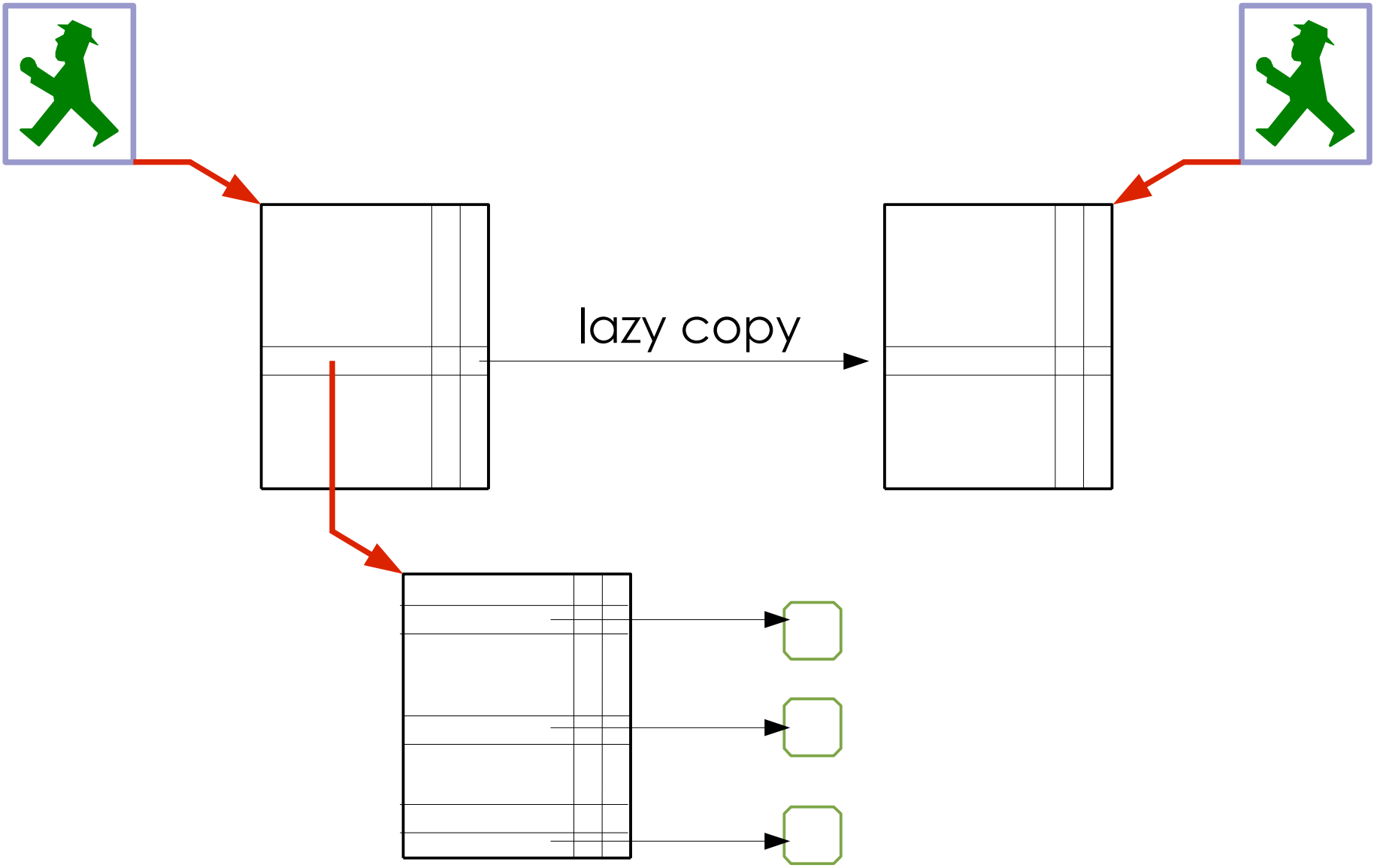
➔ ist der Prozess der alleinige Besitzer der Kachel, wird lediglich das Schreibrecht gesetzt

Verzögertes Kopieren

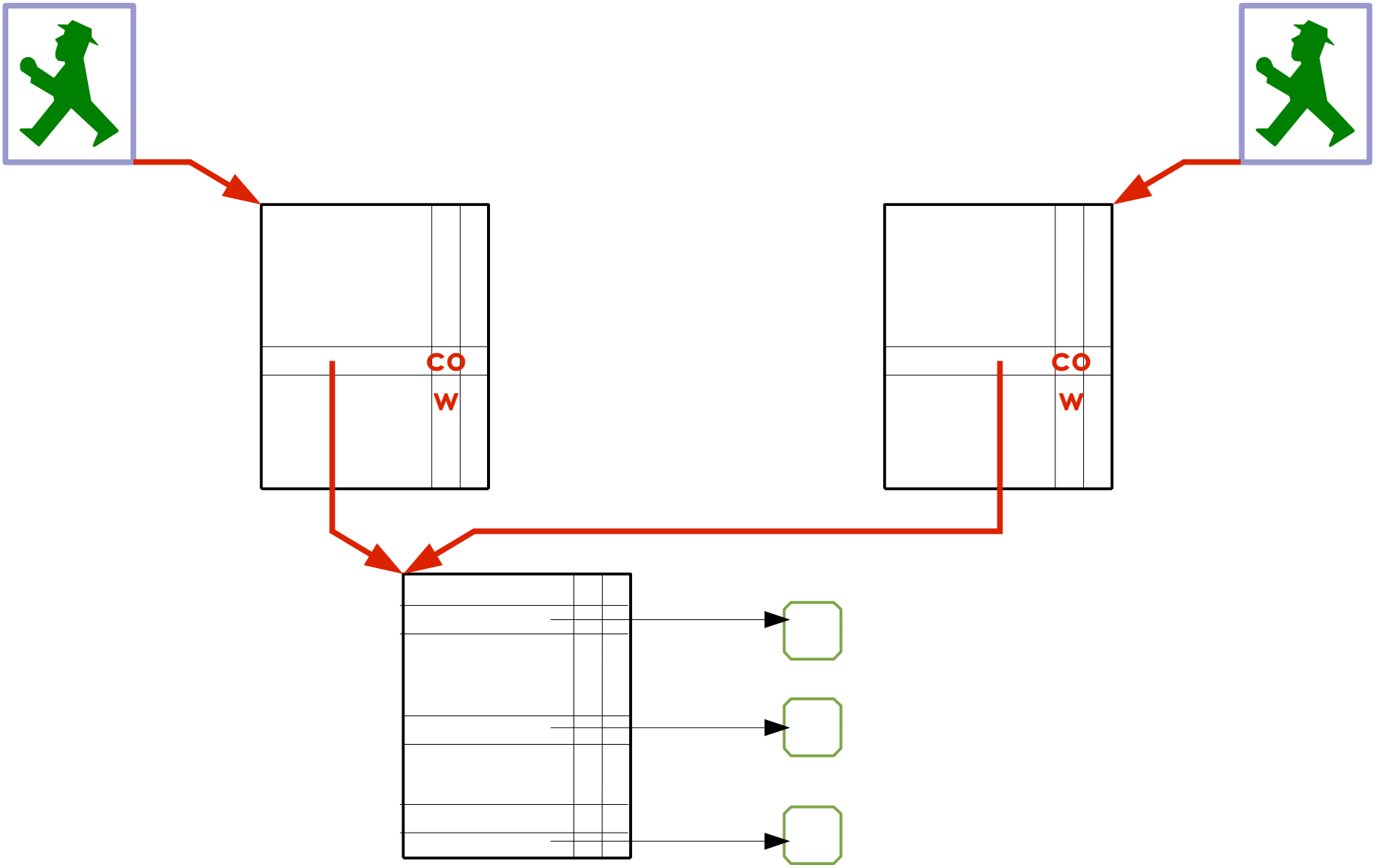


- schreibender Zugriff → Seitenfehler
- ➔ ist der Prozess der alleinige Besitzer der Kachel, wird lediglich das Schreibrecht gesetzt
- danach erfolgt der Schreibzugriff

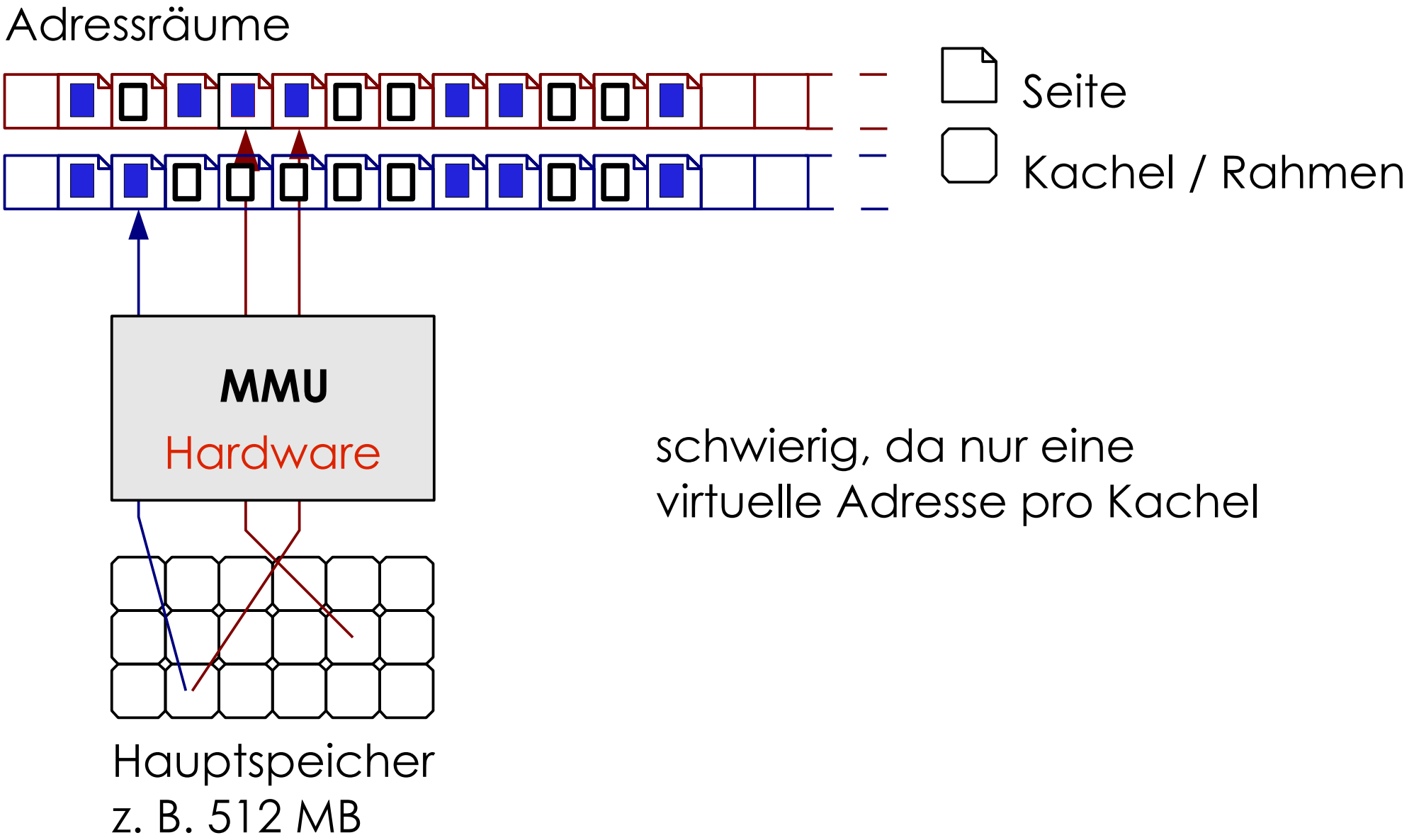
Verzögertes Kopieren großer Bereiche



Verzögertes Kopieren großer Bereiche

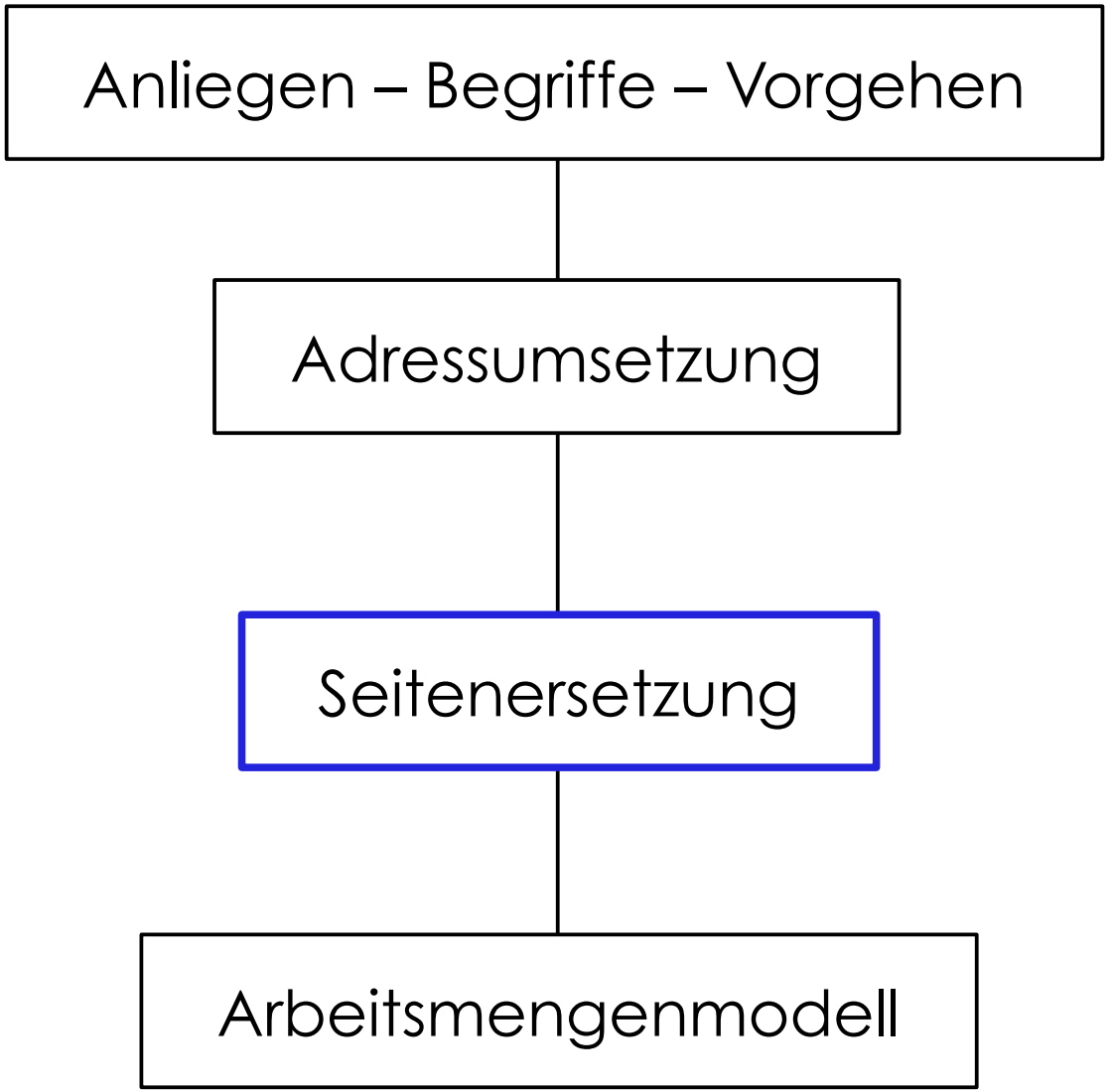


Sharing und invertierte Seitentabellen



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)
}
```