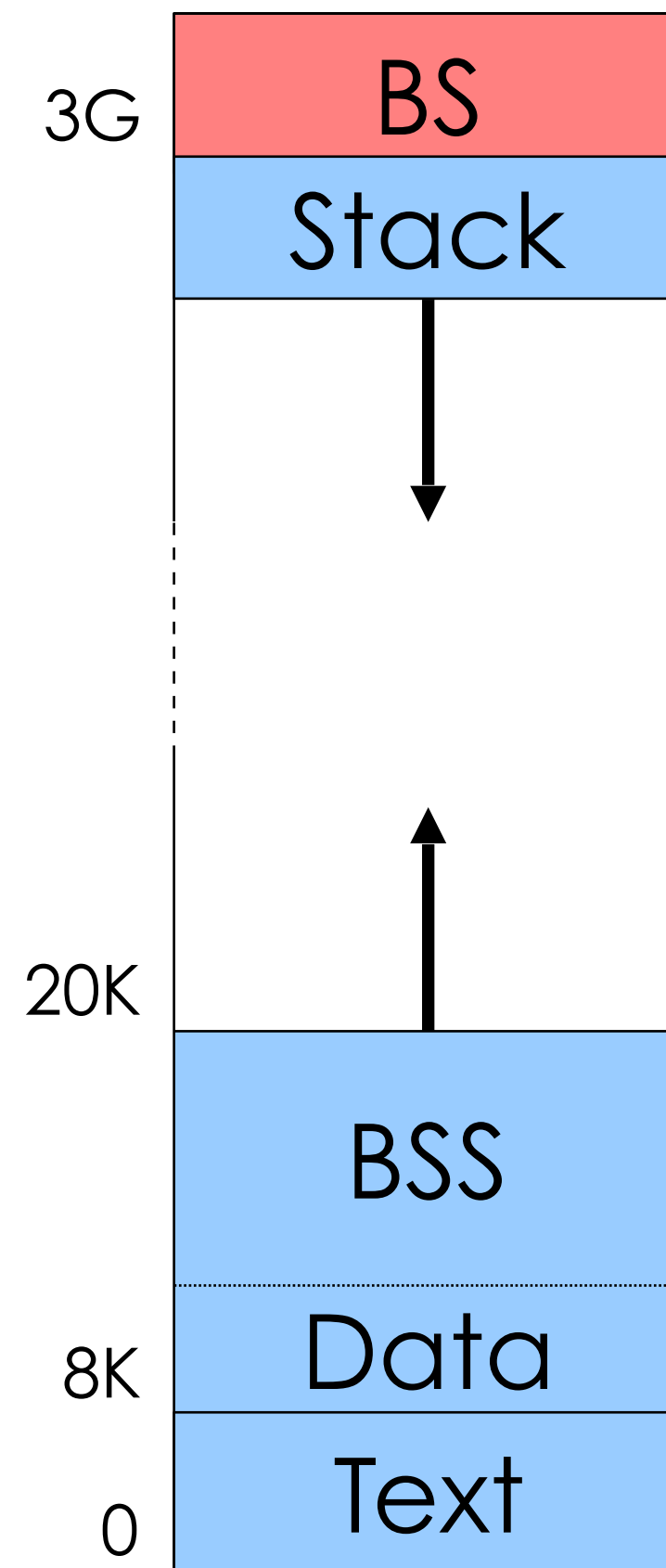


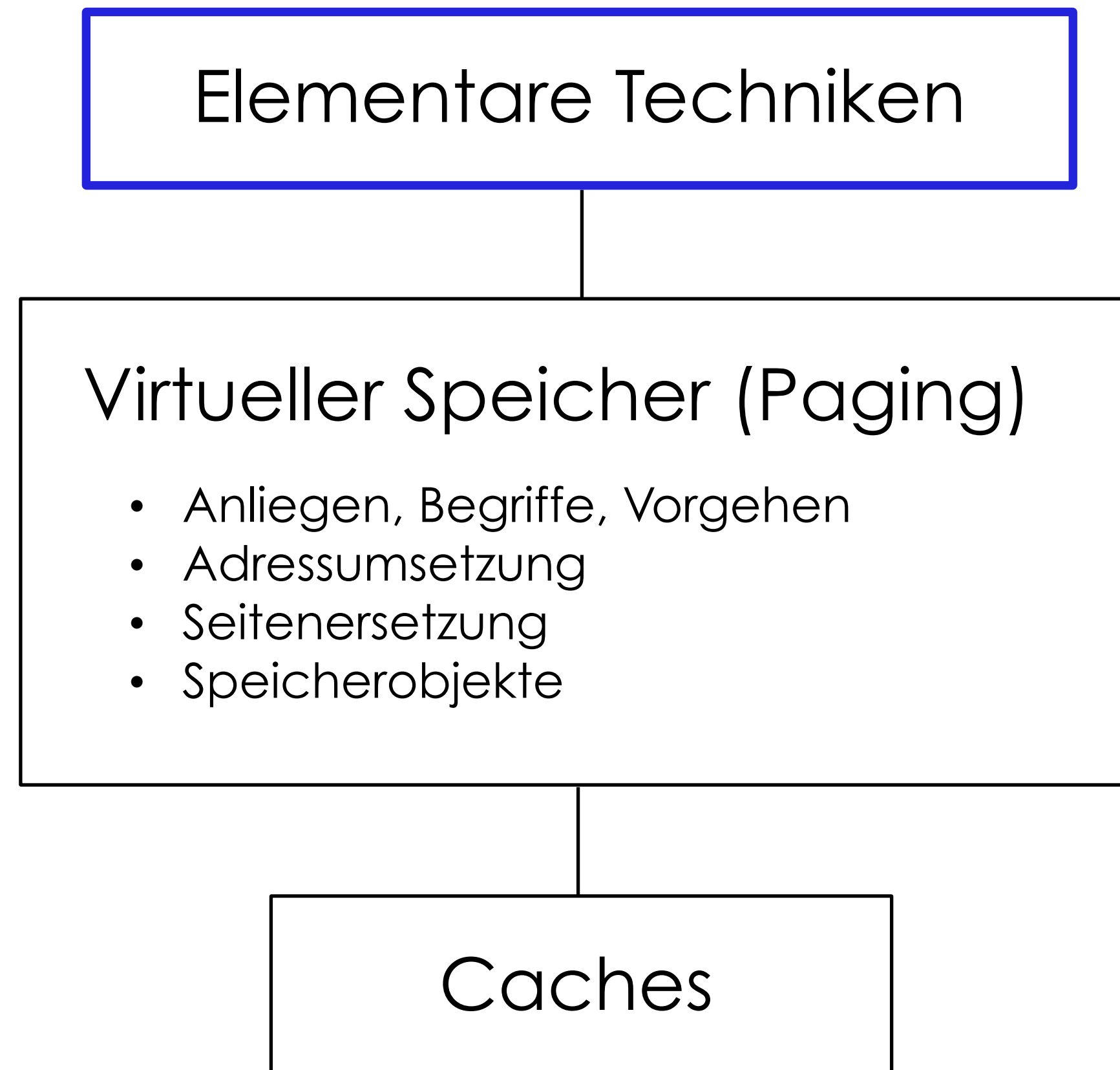
# **SPEICHER**

**MICHAEL ROITZSCH**

# Aufgaben der Speicherverwaltung

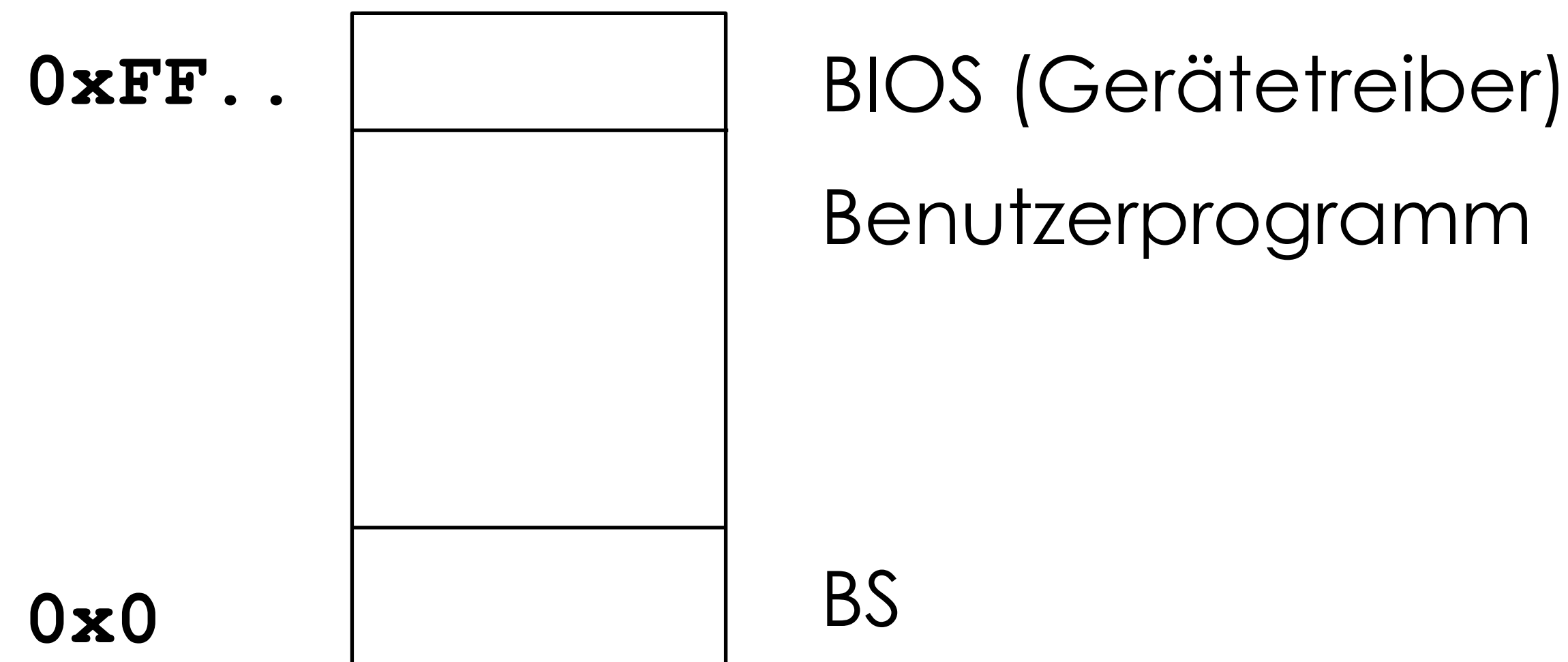


- Bereitstellung von Adressräumen
- Aufbau von Adressräumen durch Zuordnung logischer Objekte (Code-Segment, Daten-Segment, Stack, ...)
- Verwaltung des Betriebsmittels Hauptspeicher
- Schutz vor unerlaubten Zugriffen
- Organisation der gemeinsamen Nutzung („Sharing“) physischen Speichers und logischer Objekte



# Statisch

- Statische Speicherverwaltung
- keine Ein-/Auslagerung von Programmen/Daten
- Einfach-Rechner (MS-DOS), eingebettete Systeme
- Monoprogramming: ein Programm gleichzeitig
- Programme werden nacheinander geladen und ausgeführt



# Multiprogramming

---

Mehrere Benutzer-Programme gleichzeitig; 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
- denkbar ist Monoprogramming mit vielen parallelen Threads  
z. B.: die ersten Systeme für Parallelrechner erlaubten nur ein Benutzerprogramm, das aber aus vielen Threads bestehen konnte

# Feste oder variable Speicher-Partitionierung

## Relokation

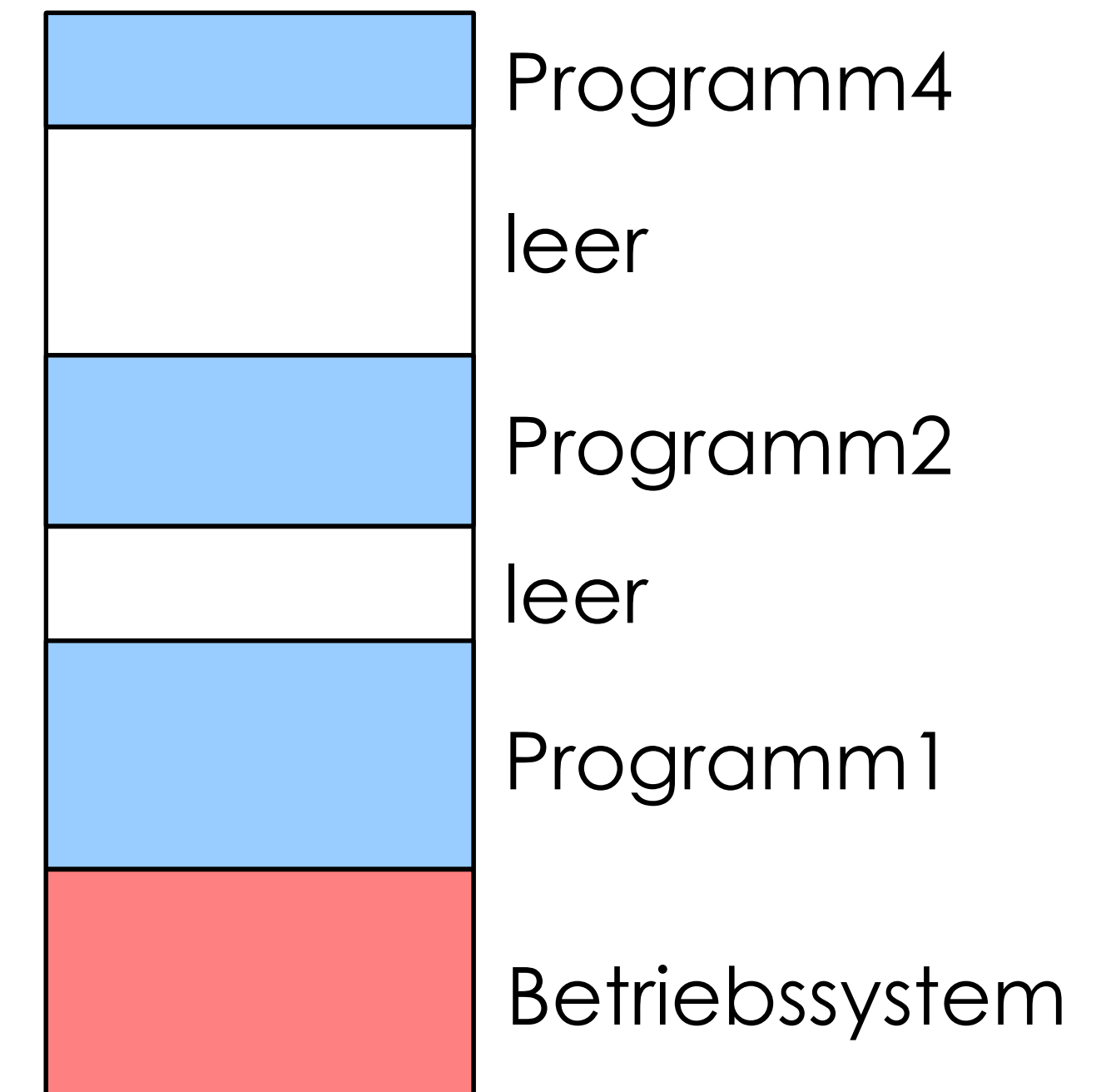
- Programme verwenden unterschiedliche Adressen, wenn sie in unterschiedlichen Partitionen ablaufen
- Umsetzen der Adressen
  - beim/vor dem Laden (Software)
  - zur Laufzeit (Hardware)

## Schutz der Partitionen voreinander

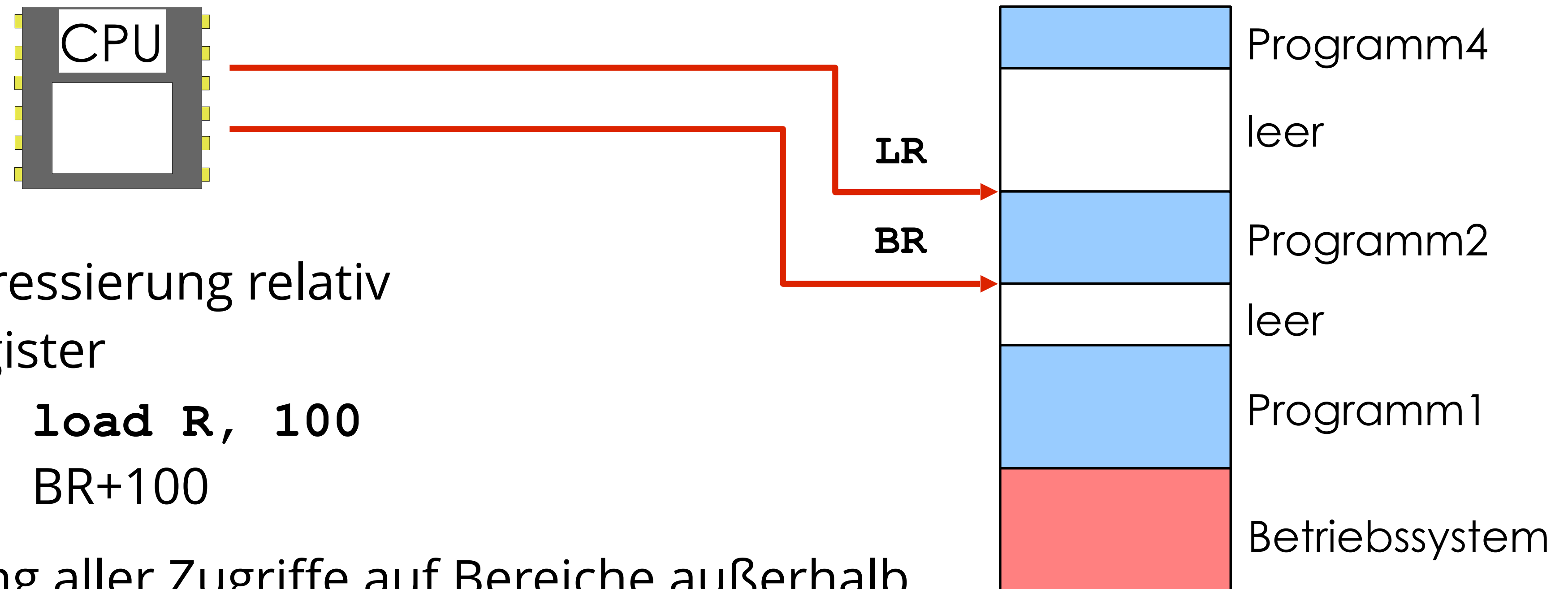
- Überprüfung der Adressen zur Laufzeit (Hardware)

## Idee

- Entkoppelung der Adressen im Programmcode von den Adressen beim Speicherzugriff



# 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

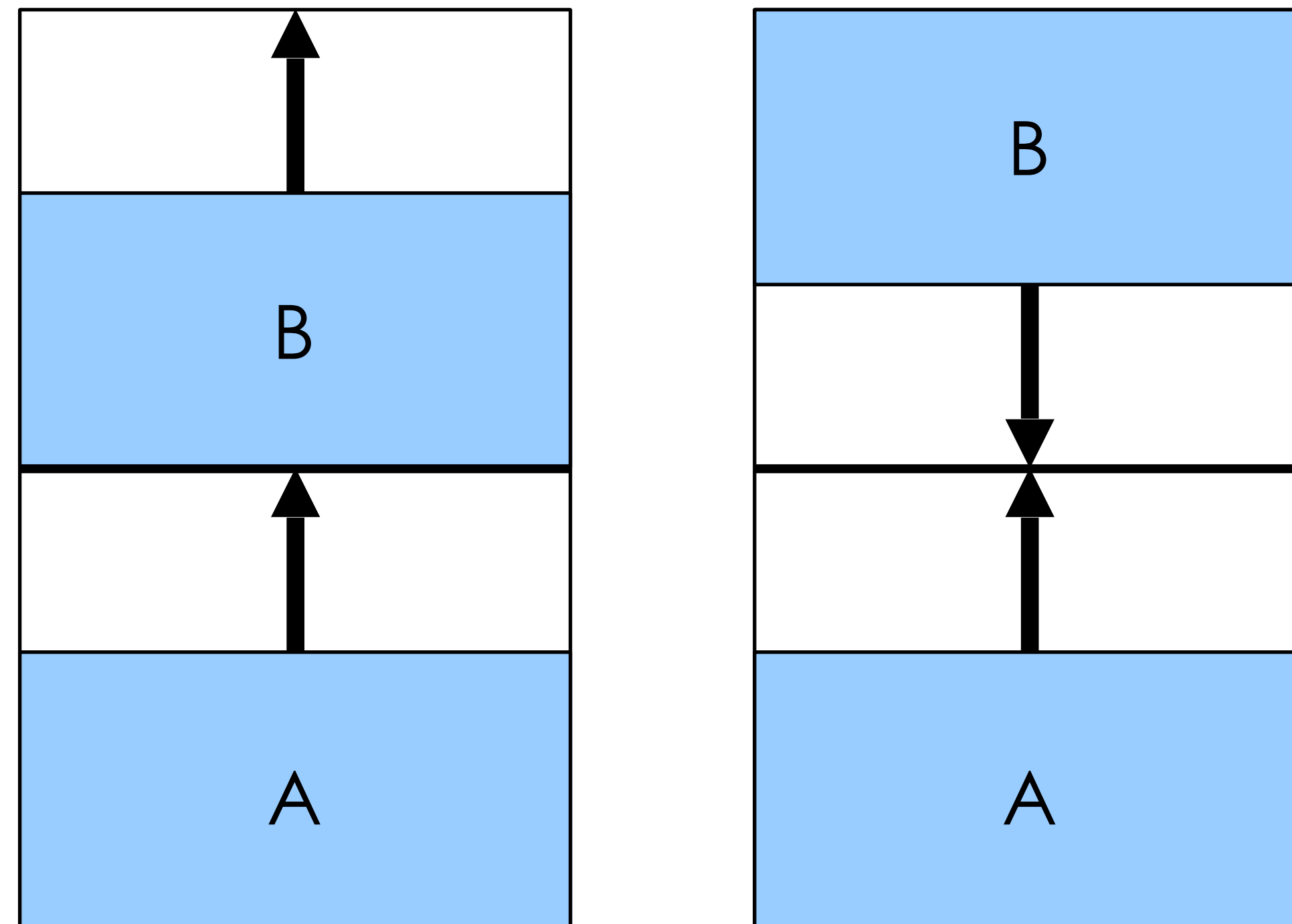
# Wachsen von Partitionen

Einlagerungsalgorithmen zur Verringerung des Verschnitts

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

Verwaltung

- Bitmaps, Listen, Bäume, ...





# Partitionen: Nachteile und Probleme

---

- Verschnitt: Speicherfragmente zwischen den Partitionen können unpassende Größe haben (externe Fragmentierung)
- Programmstartzeiten: vollständiges Laden vor Ausführung
- Limitation der Größe eines Prozesses durch verfügbaren Hauptspeicher
- „ruhende“ Teile: auch ungenutzte Bereiche belegen Speicher

# 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

## **Nachteile**

- Ein-/Auslagerungszeit: vollständiger Prozess wird ausgelagert
- Platzbedarf auf Externspeicher

# Overlays / Überlagerungstechnik

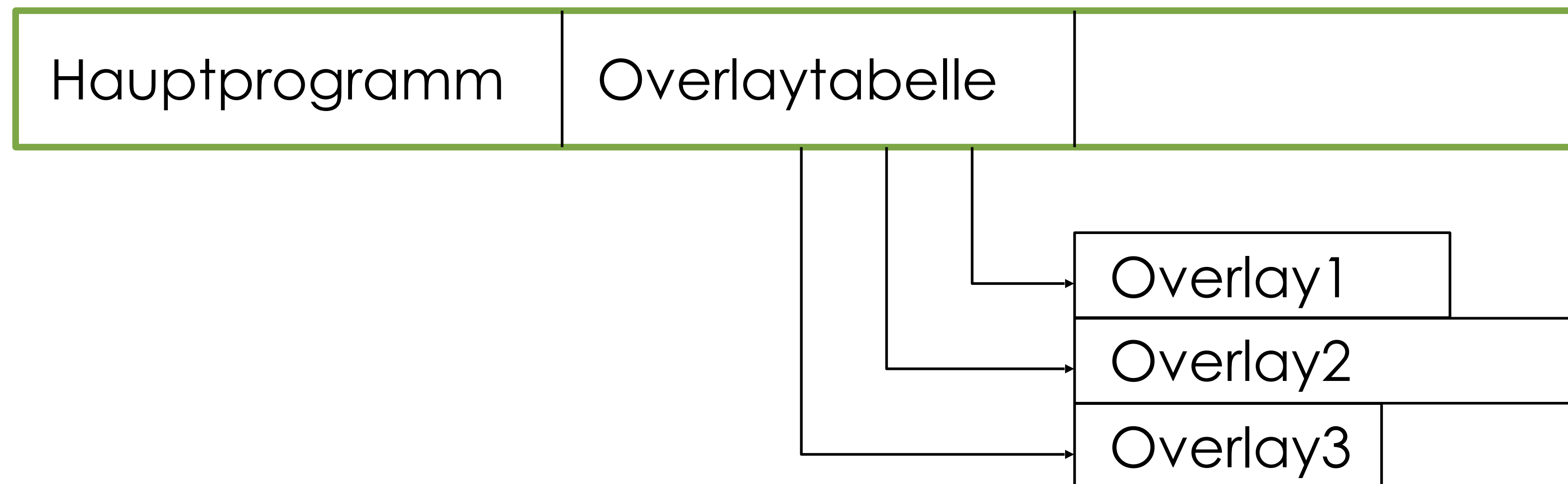
## Vorgehen

- Programm in Stücke zerlegen, die nicht gleichzeitig im Speicher sein müssen
- beliebig große Programme möglich

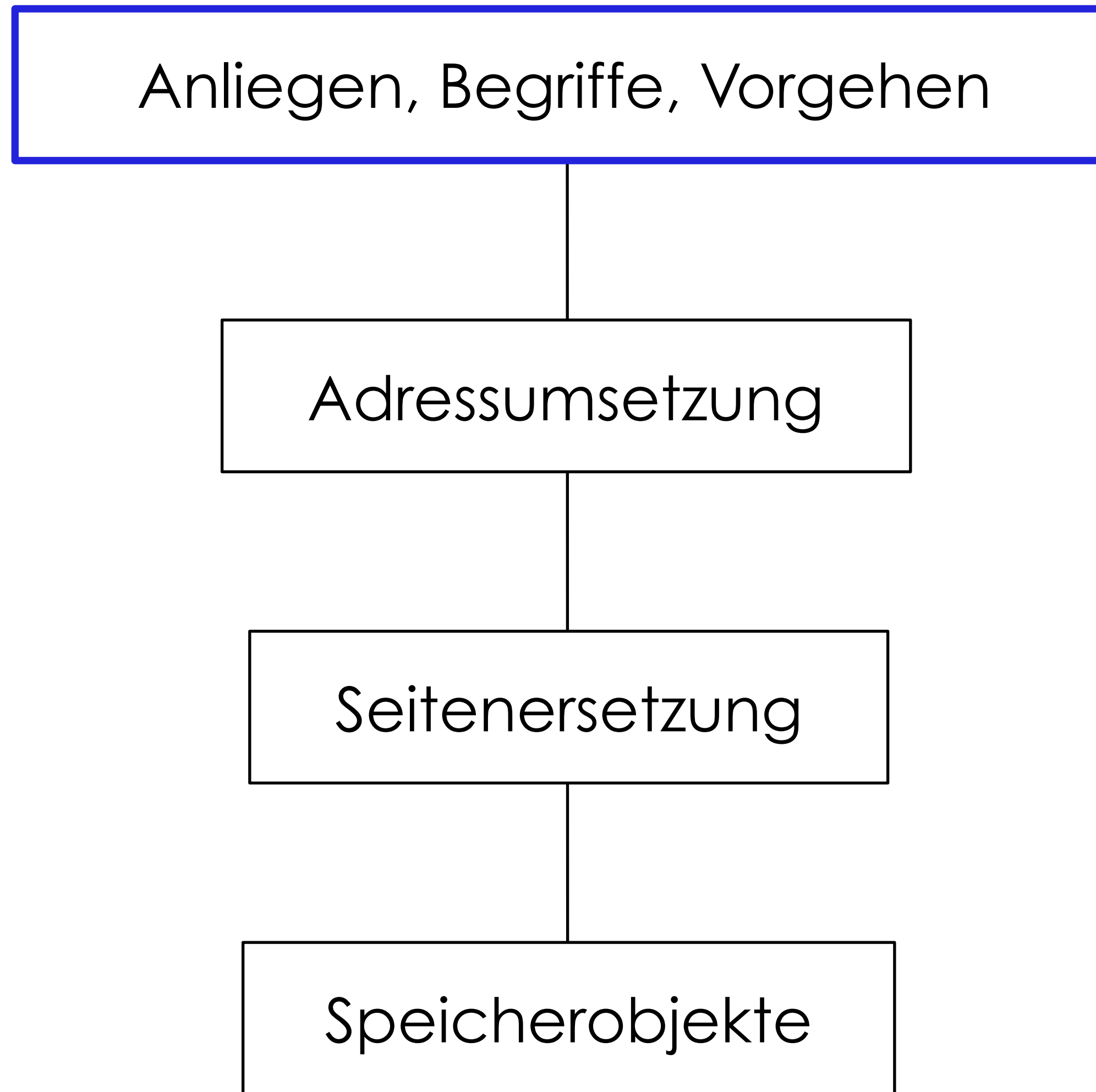
## Nachteil

- manueller Aufwand

Hauptspeicher



# Wegweiser: Virtueller Speicher



# Adressraum

---

## **Begriff**

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

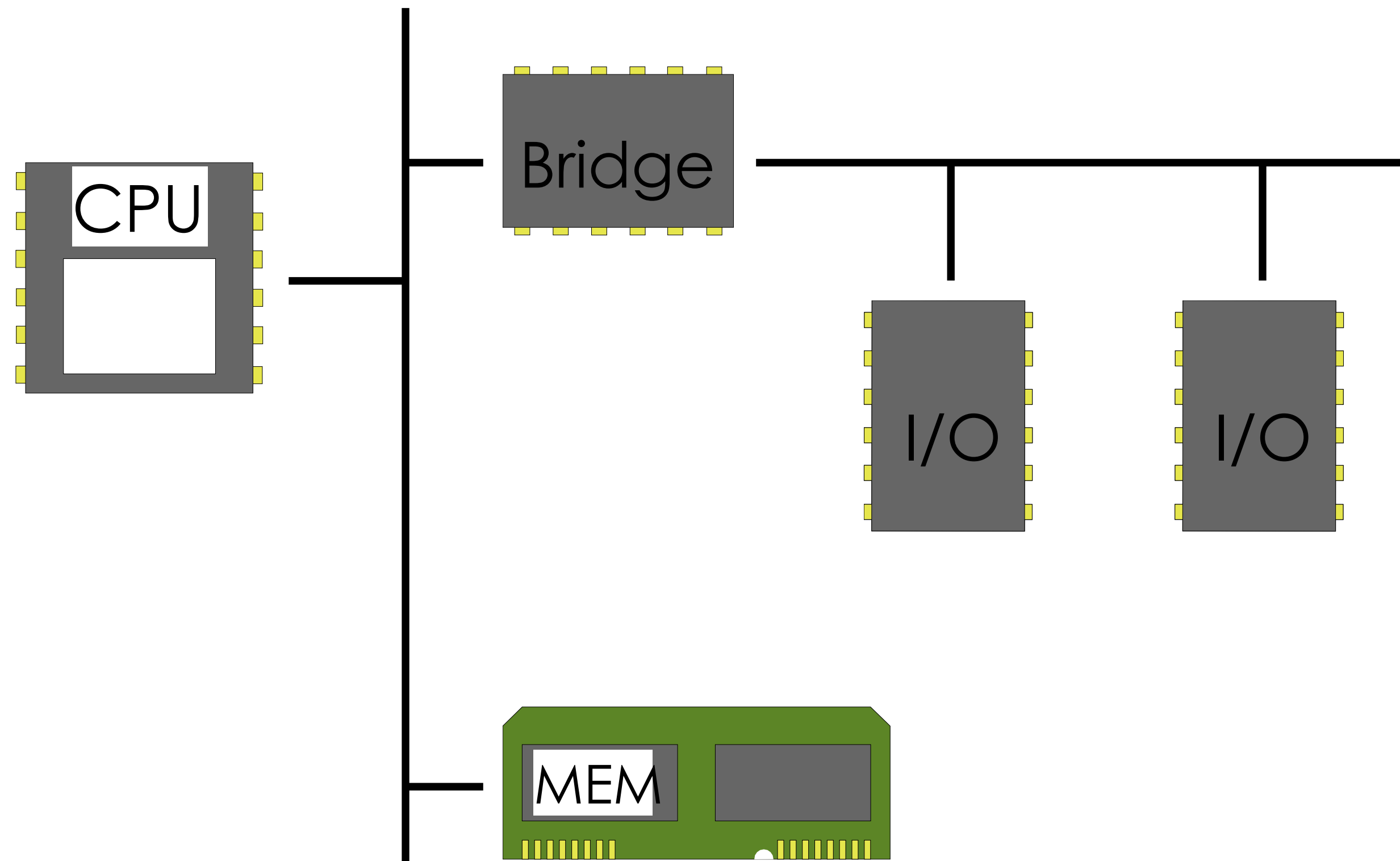
## **Physischer Adressraum**

- durch Adressleitungen gebildeter Adressraum,  
z. B. am Speicher- oder Peripherie-bus eines Rechners
- Abbildung der Prozessor-Adressen auf die vorhandenen Speicherbausteine und E/A-Controller

## **Virtueller (logischer) Adressraum eines Prozesses**

- dem Prozess zugeordneter Adressraum

# Physische Adressräume



# Nutzung eines virtuellen Adressraums

## Unix-Adressraum



- Jeder Prozess bekommt seinen eigenen virtuellen Adressraum zur exklusiven Nutzung
- Prozess kann diesen nach eigenen Anforderungen individuell gestalten

## Regionen (Segmente)

- Logisch zusammenhängende Adressbereiche
- 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)
- frei teilbar und nutzbar
- Fehlermeldung bei Zugriff auf nicht belegte Bereiche
- Schutz vor Zugriffen von anderen Prozessen
- Einschränken der Zugriffsrechte auf bestimmte Bereiche (z. B. Code nur lesen)
- sinnvoller Einsatz des physischen Speichers: effiziente Nutzung durch viele Prozesse, kurze Ladezeiten
- transparent für Programmierer, aber steuerbar



# Prinzipien der virtuellen Speichers

---

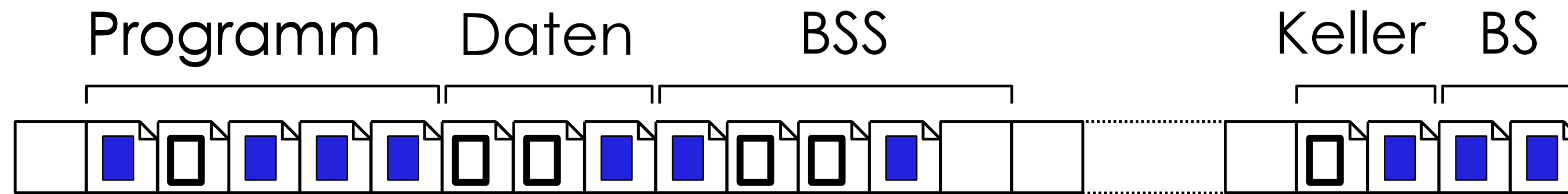
## **Idee: Indirektion**

- Prozess (Code, Daten) arbeitet mit virtuellen Adressen
- flexible Abbildung auf physische Adressen

## **Granularität durch Partitionierung**

- des virtuellen Adressraums in Seiten / Pages
- der Hauptspeichers in Kacheln / Rahmen / Page Frames
- jeweils gleicher Größe
- Organisation der Zuordnung der Stücke zueinander durch Hardware und Betriebssystem

# Eine denkbare Situation



 unbenutzt, ungültig

 gerade im Hauptspeicher

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

# 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 Schleifen
- 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 (virtuellen) Adressraum bereitstellt,

basierend auf

- einer Partitionierung von Adressräumen in Einheiten einheitlicher Größe (Seiten, Rahmen)
- einer Adressumsetzung durch Hardware, gesteuert durch das Betriebssystem
- der Nutzung eines externen Speichermediums
- einer Ein- und Auslagerung von Teilen des logischen Adressraums eines Prozesses durch Betriebssystem

# Virtueller Speicher im Betriebssystem

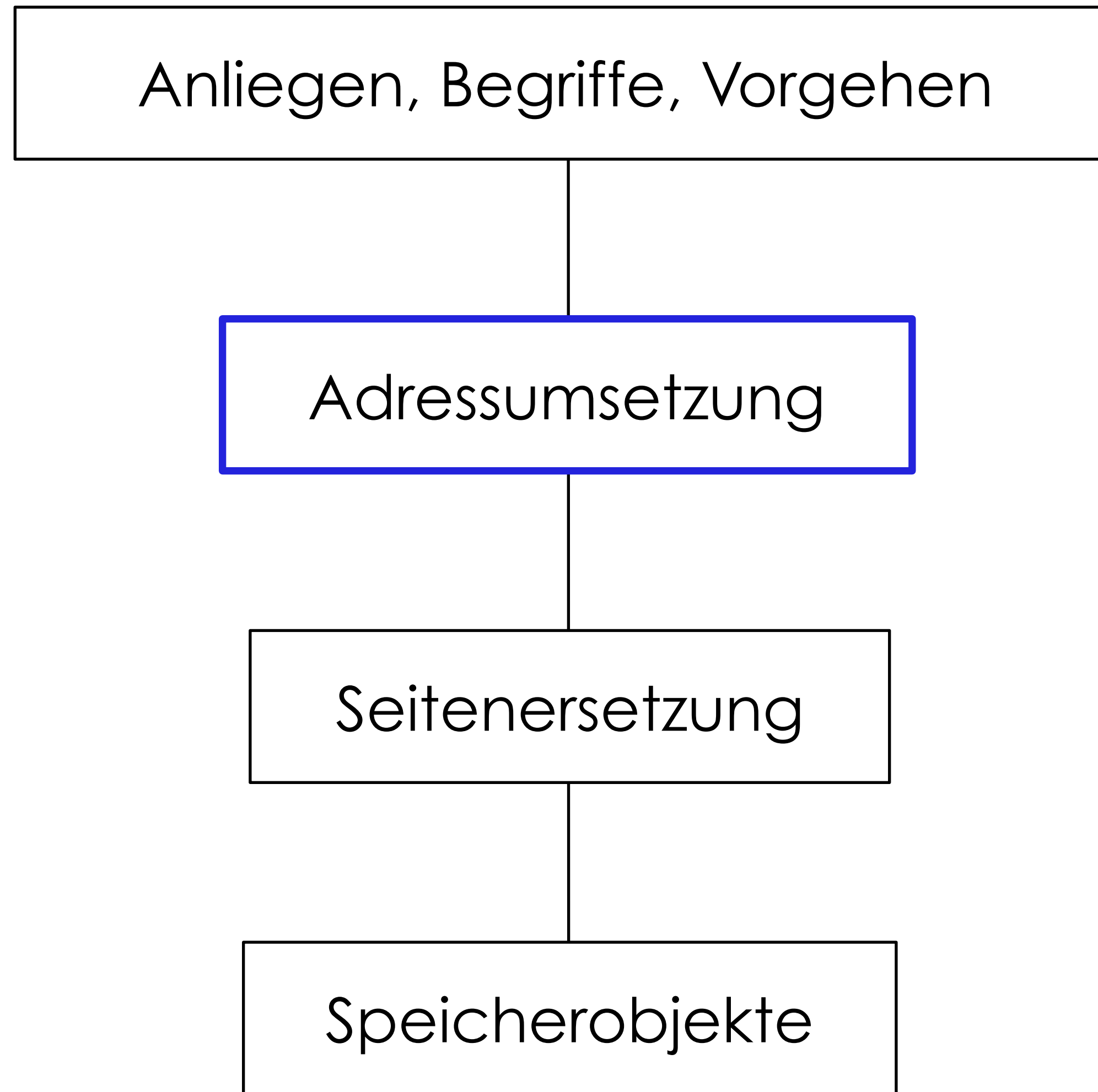
---

## **Teilaufgaben**

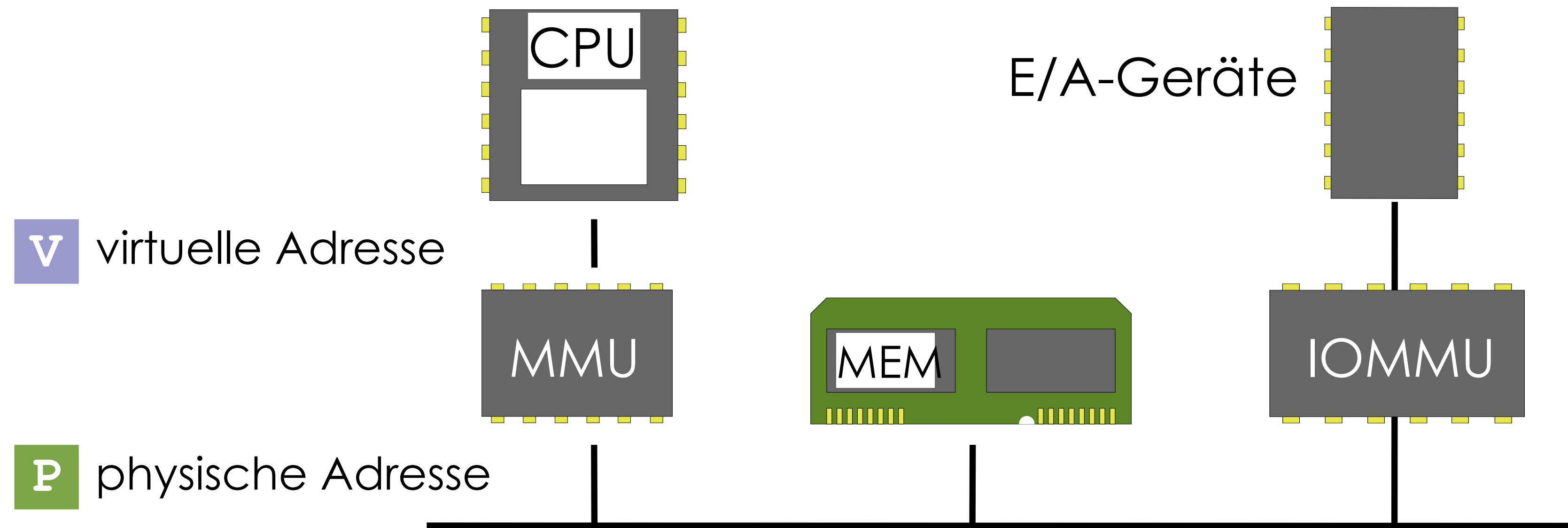
- Verwaltung des Betriebsmittels Hauptspeicher
- Seitenfehler-Behandlung
- Aufbau der Adressraumstruktur: Speicherobjekte und Regionen
- Interaktion der Prozess- und Speicher-Verwaltung

# Wegweiser: Virtueller Speicher

---



# Rechnerarchitektur: Addressumsetzung



## MMU (Memory Management Unit)

- Abbildung: virtuelle → reale (physische) Adresse
- Schutz bestimmter Bereiche (lesen/schreiben)
- Betriebssystemaufruf bei abwesenden/geschützten Seiten:  
Seitenfehler / Page Fault

# Prinzipielle Arbeitsweise einer MMU

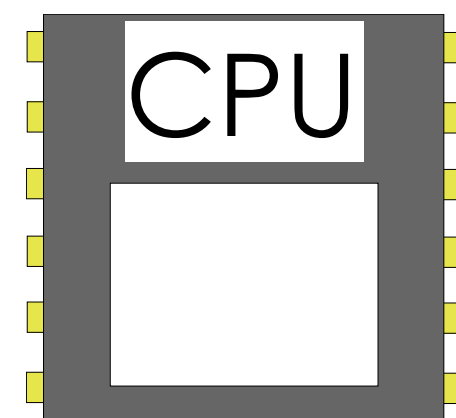
---

**V** 0 0 1 0 1 0 0 1 0 1 1 0 virtuelle Adresse

**P**  physische Adresse

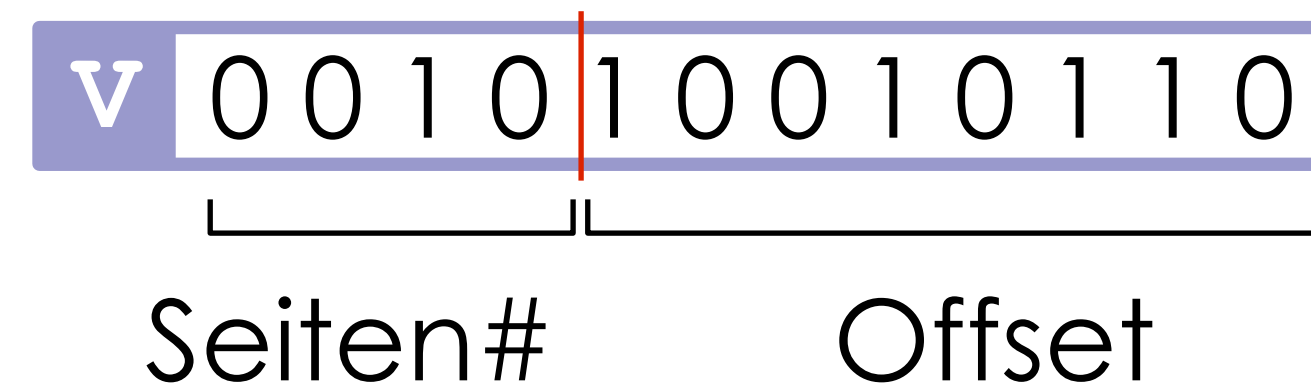


# Prinzipielle Arbeitsweise einer MMU



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



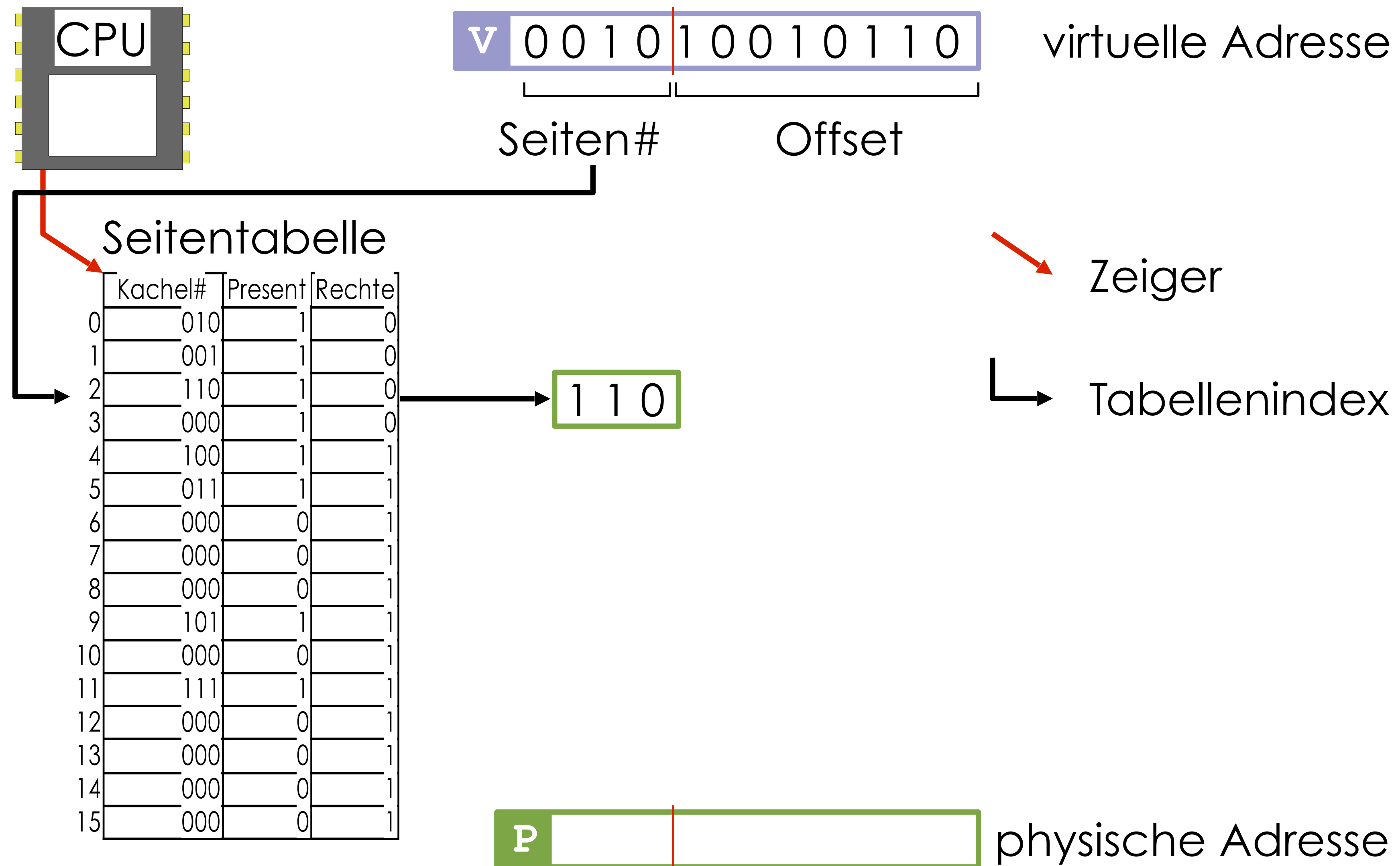
virtuelle Adresse

Zeiger

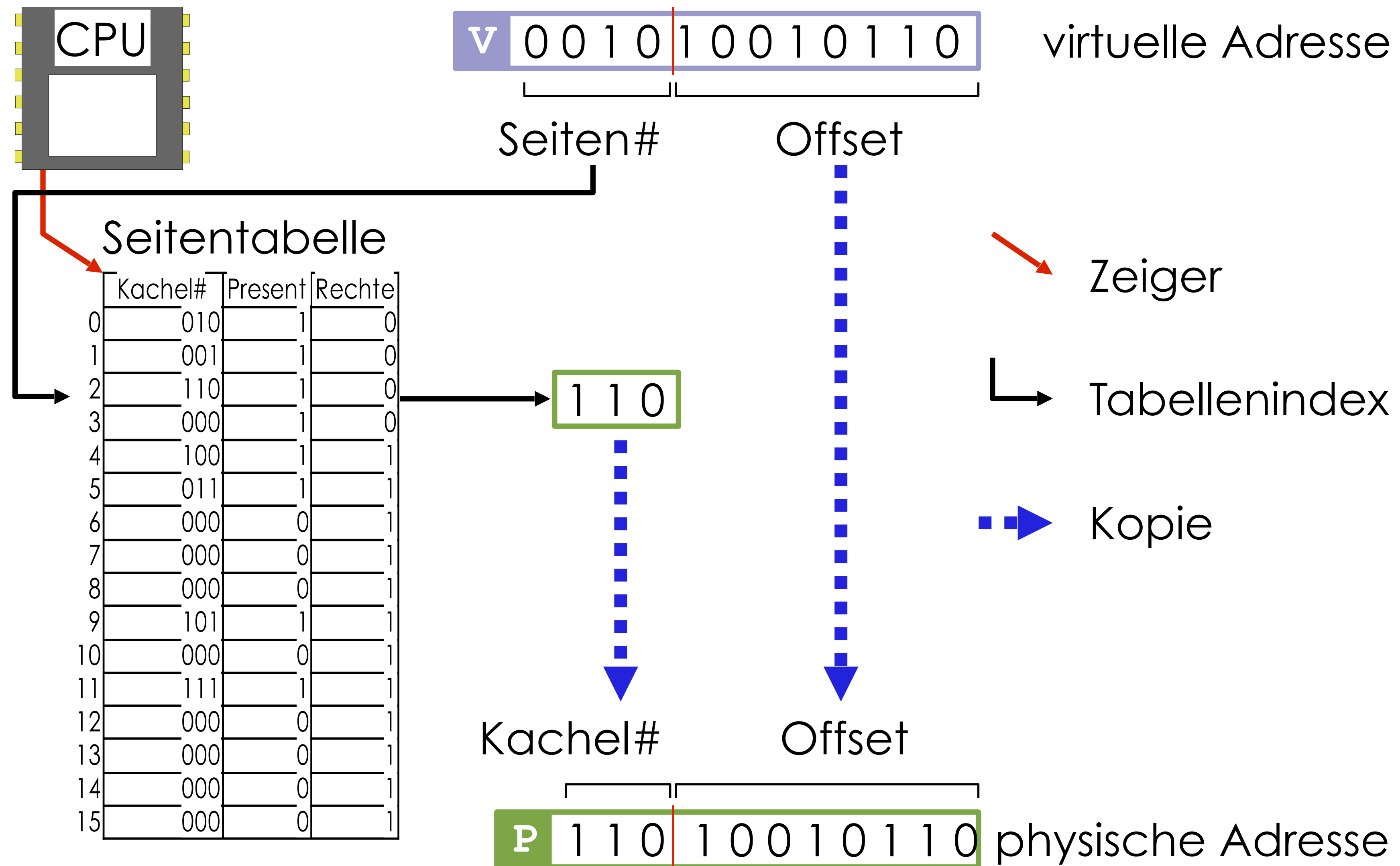


physische Adresse

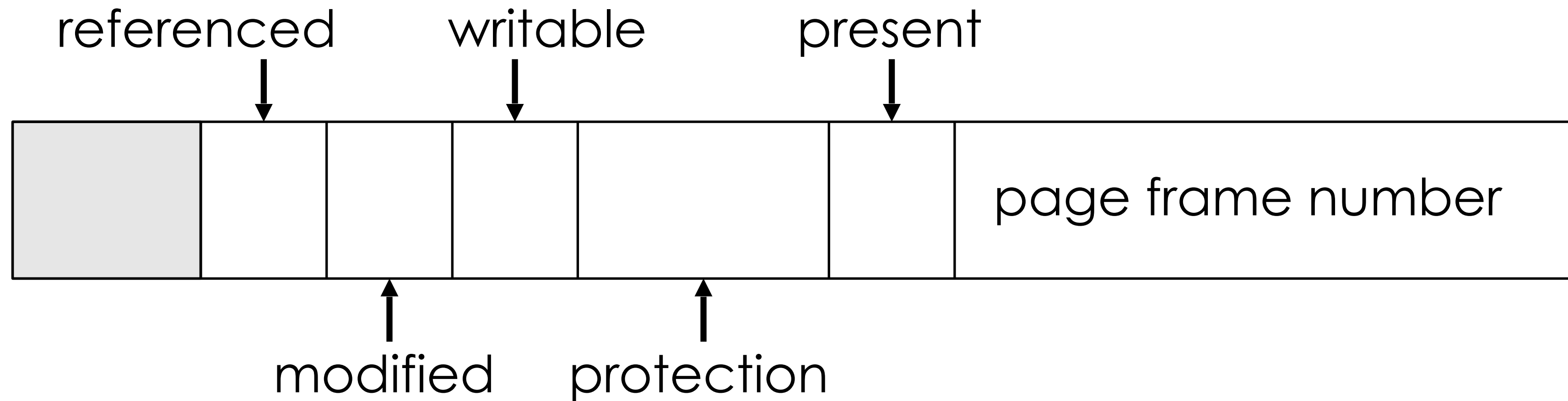
# Prinzipielle Arbeitsweise einer MMU



# Prinzipielle Arbeitsweise einer MMU



# Aufbau eines Seitentabelleneintrags



## Seiten-Attribute

- present Seite befindet sich im Hauptspeicher
- protection erlaubte Art von Zugriffen (CPU-Modus)
- writable schreibbar, manchmal auch executable-Bit
- modified schreibender Zugriff ist erfolgt („dirty“) – wird von MMU gesetzt
- referenced irgendein Zugriff ist erfolgt – wird von MMU gesetzt

# Speicherzugriff

---

## Bei jedem Speicherzugriff:

- MMU überprüft **Präsenz** und **Rechte**
- mögliche Fehler: not present, not writable, protected, ...

## Ablauf eines Seitenfehlers (exception):

- Zurücksetzen des auslösenden Befehls
- Umschaltung des Prozessormodus
- Ablegen der auslösenden Adresse und der Zugriffsart auf dem Stack
- Sprung in den Kern
- Seitenfehlerbehandlung durch Betriebssystem
- letzte Instruktion des Handlers: **iret**

# Seitenfehlerbehandlung

---

## **„Echter“ Fehler**

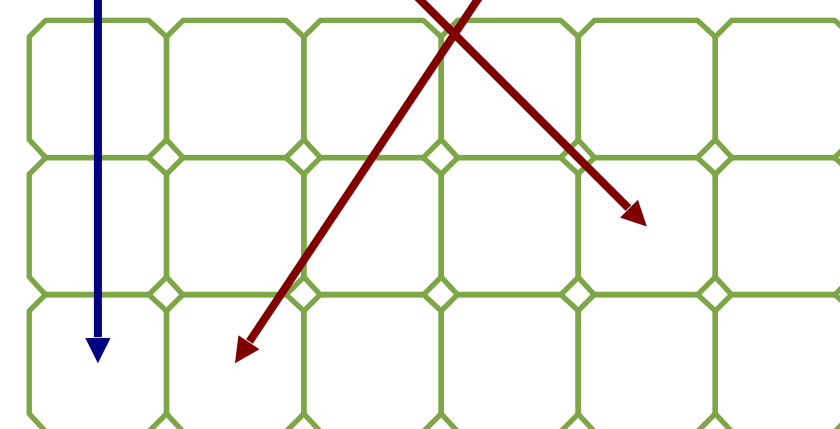
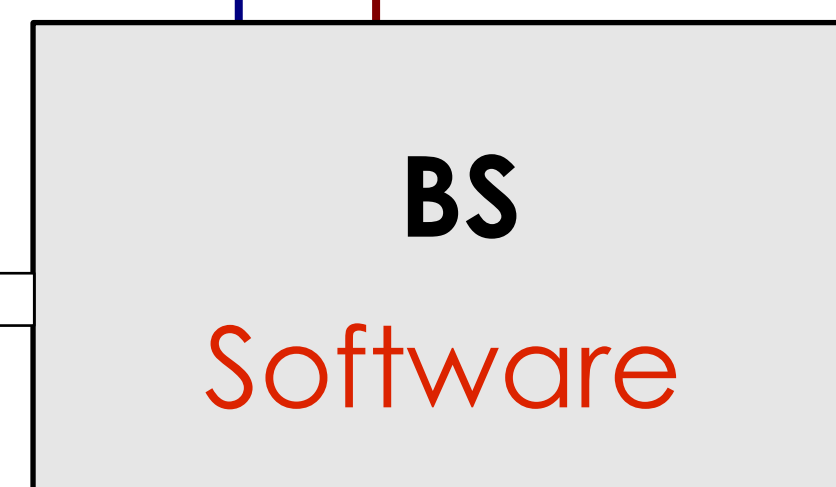
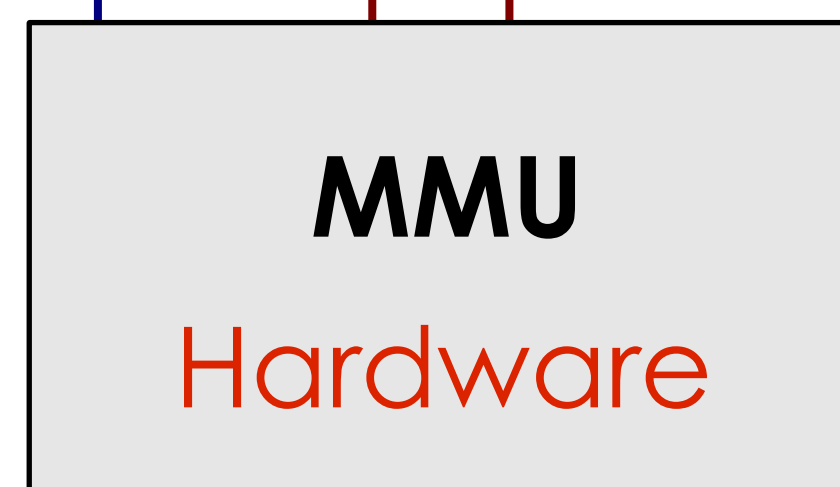
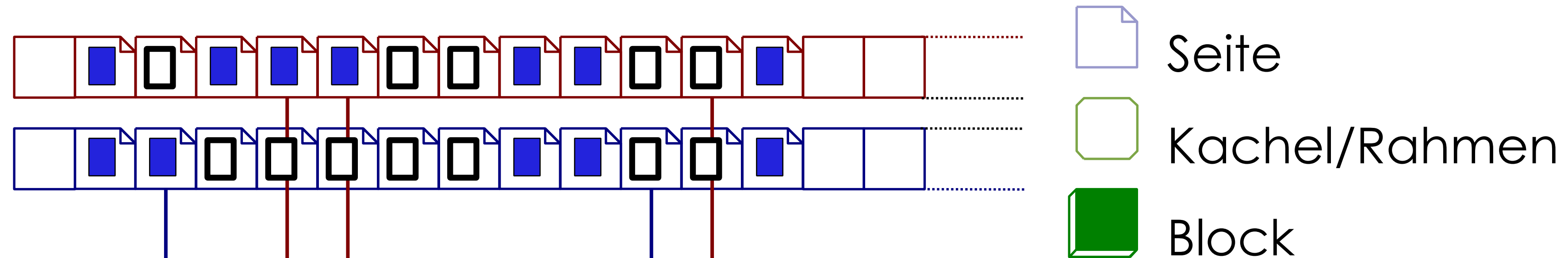
- zum Beispiel Zugriff auf ungültigen Speicherbereich
- Prozess wird Signal **SIGSEGV** zugestellt
- Standardreaktion: Prozess wird beendet

## **Reparabler Seitenfehler**

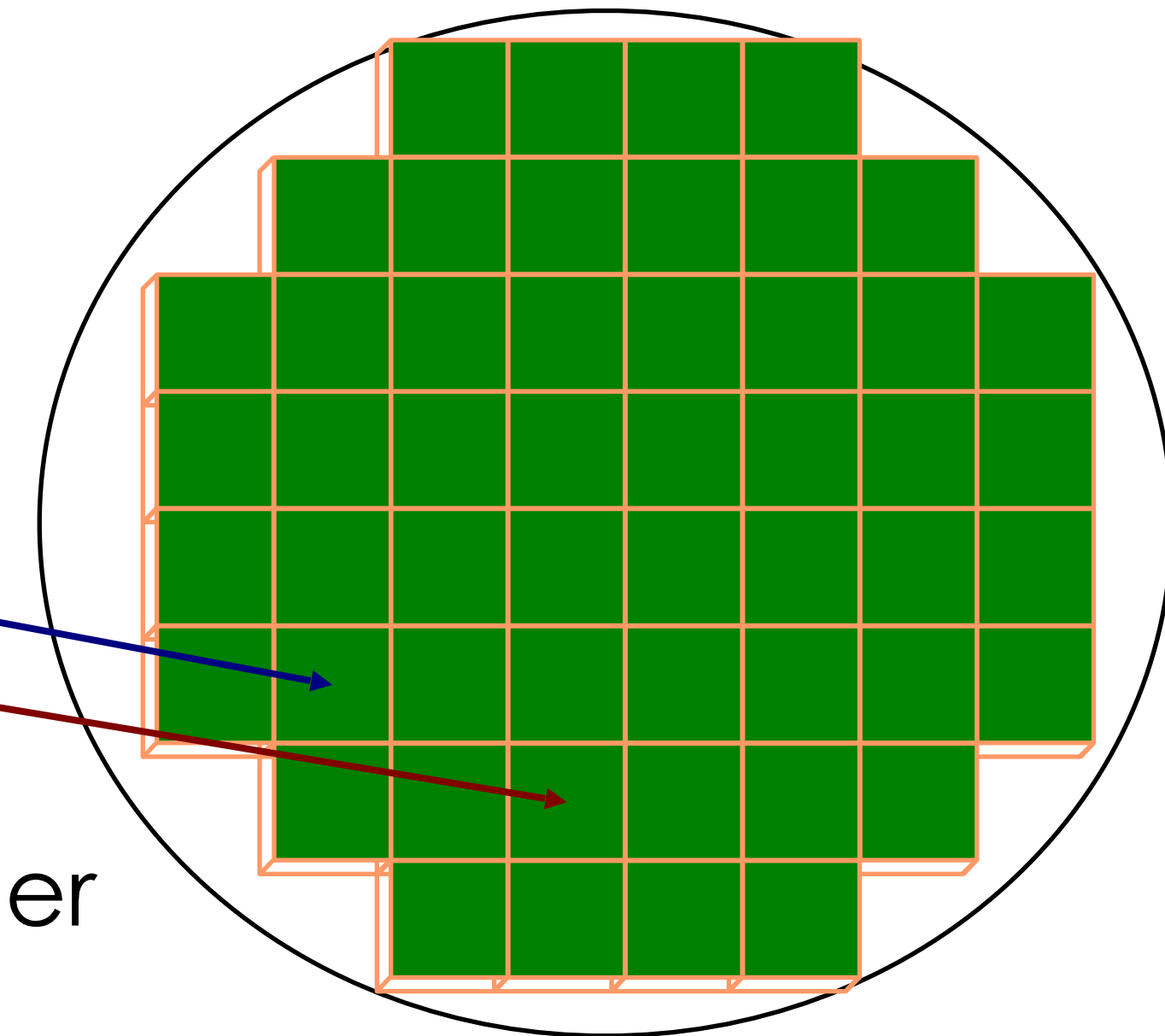
- Zugriff gültig, aber Seite momentan nicht im Speicher
- wird transparent im Betriebssystem behandelt
- je nach Speicherobjekt zum Beispiel durch Nachladen
- entsprechende Manipulation der Seitentabelle
- Prozess wird anschließend fortgesetzt

# Seiten, Kacheln, Blöcke

Adressräume z. B. 4 GB



Hauptspeicher  
z. B. 4 GB



Plattenspeicher  
z. B. 4 TB

# MMU-Probleme: Größe und Geschwindigkeit

## Problem 1: Größe – ein Beispiel

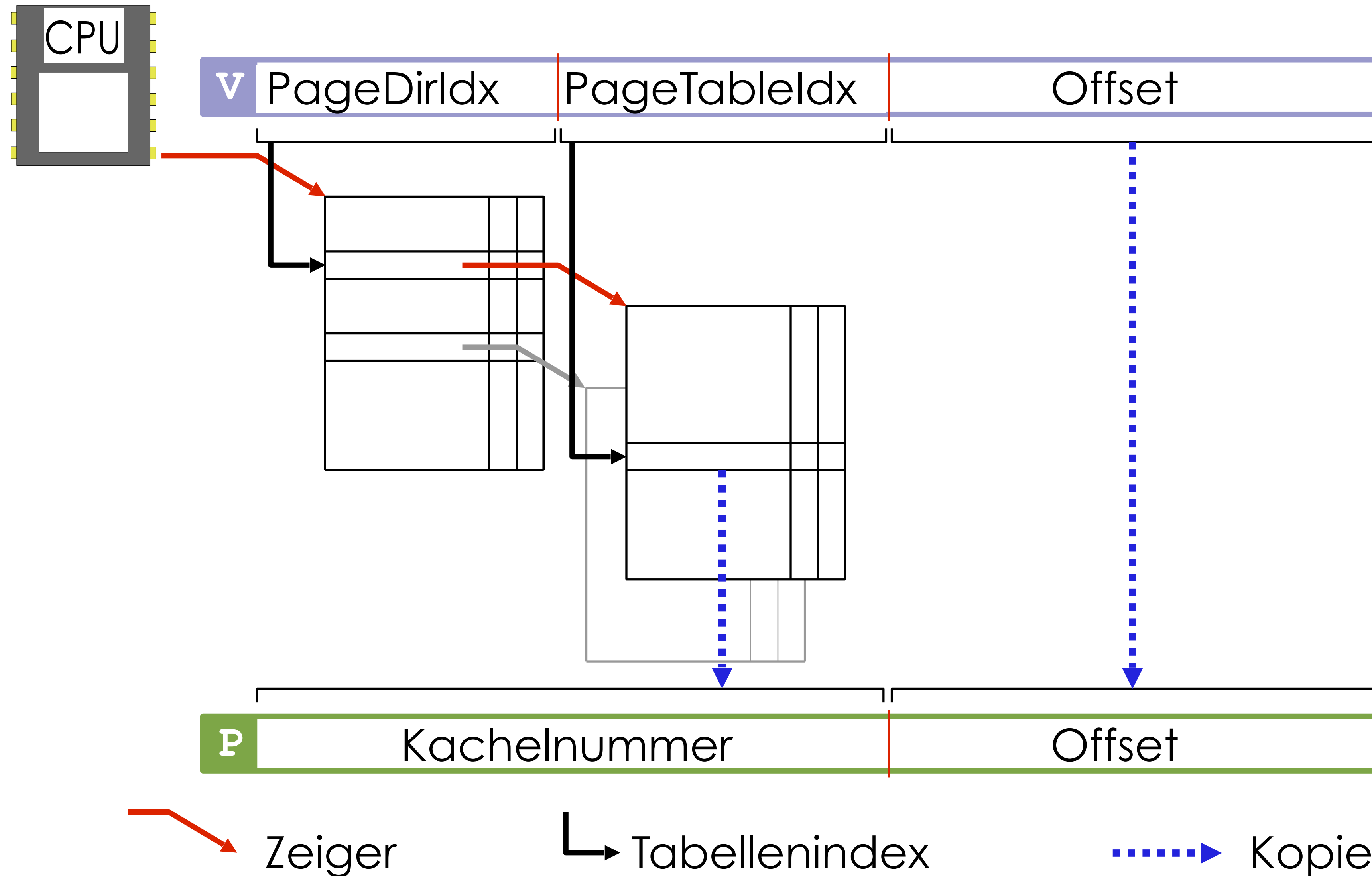
- Realspeicher: 4 GB
- virtuelle Adressen: 64 Bit
- Seitengröße: 4 KB
- Aufteilung virtuelle Adressen: 52 Bit Index in der Seitentabelle
- Größe Seitentabelle:  $2^{52} * 8B = 16PB$  pro Prozess

## Problem 2: Geschwindigkeit der Abbildung – ein Beispiel

- CPU-Takt: 1 GHz → 1 ns (4 Speicherzugriffe)
- Speicher-Latenz: 100 – 300 ns



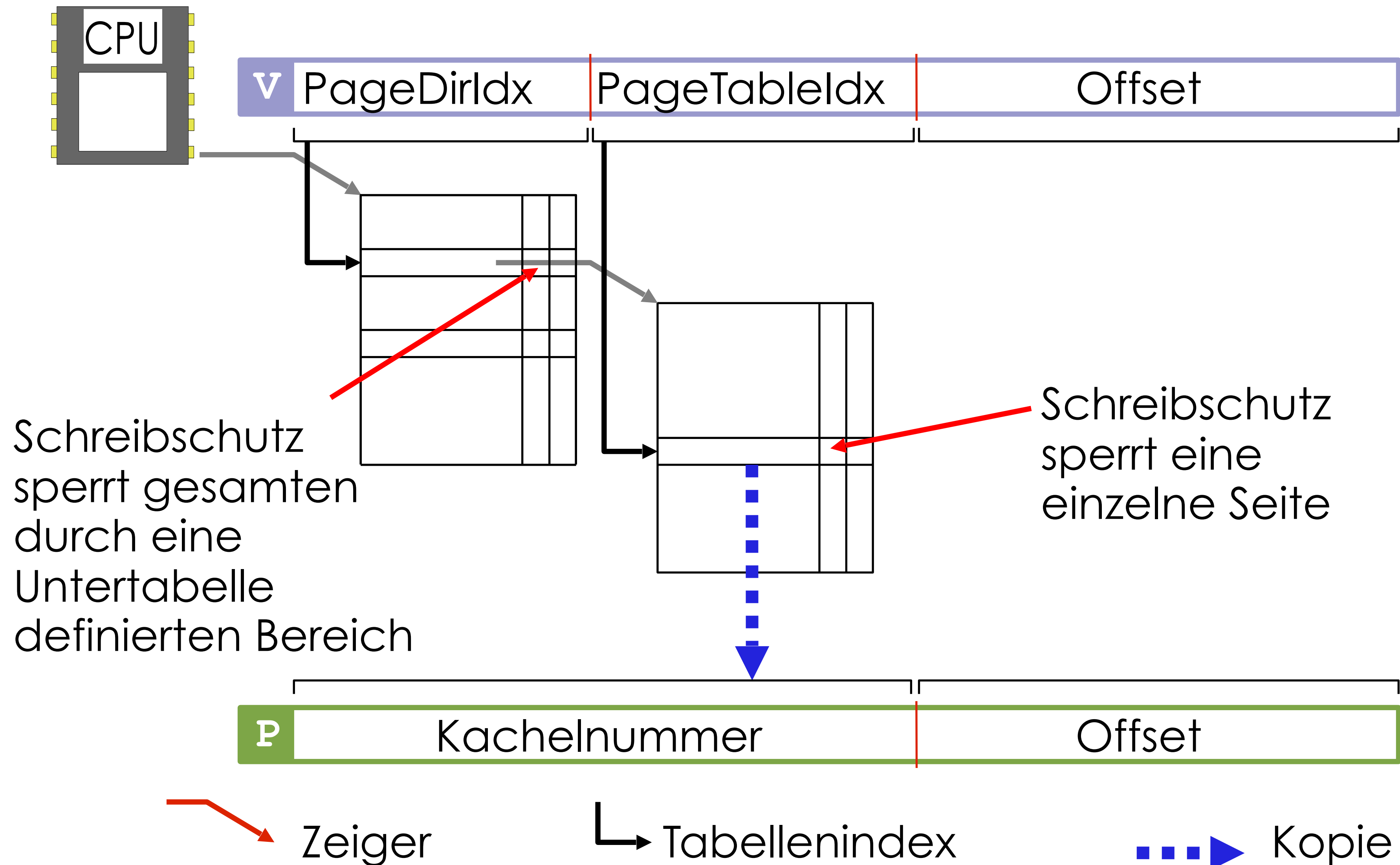
# Problem 1: Baumstrukturierte Seitentabellen



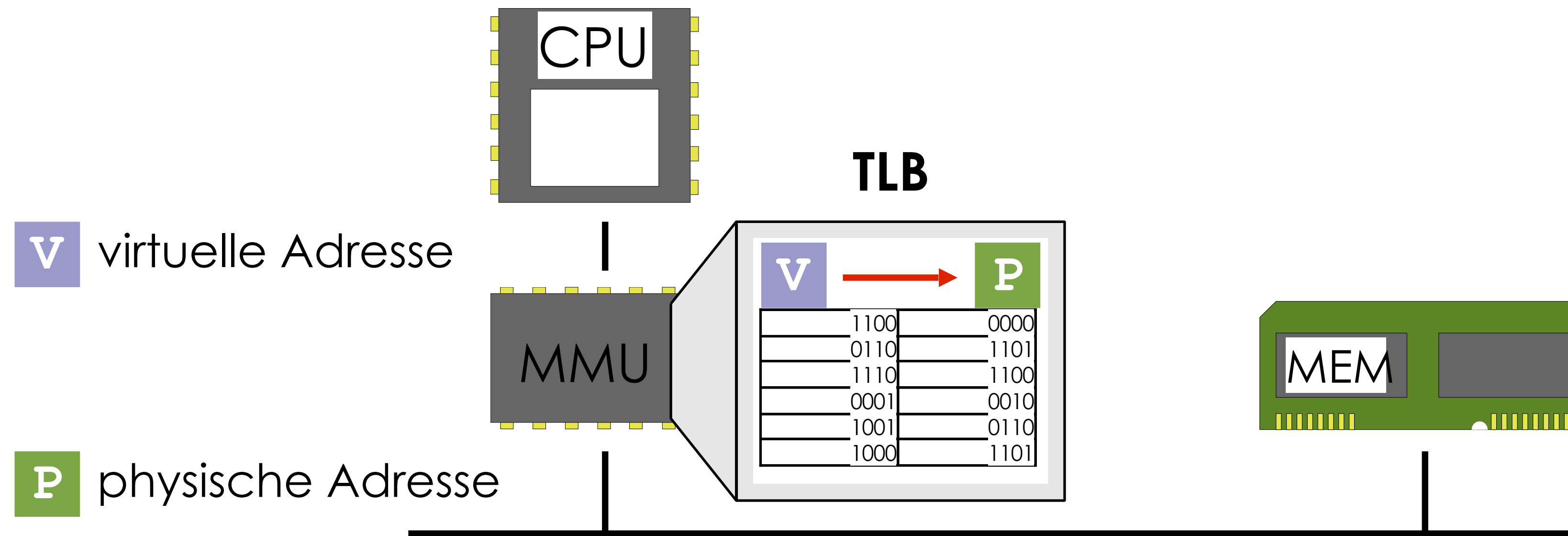
# Eigenschaften baumstrukturierter Seitentabellen

- Seitentabellen nur bei Bedarf im Hauptspeicher:  
auf höherer Stufe kann bereits vermerkt sein, dass keine Unter-Tabellen vorhanden sind, **dadurch Platz-Einsparung**
- Hierarchiebildung möglich (nächste Folie):  
z. B. durch Schreibsperre in höherstufiger Tabelle ist  
ganzer Adressbereich gegen Schreiben schützbar
- Gemeinsame Nutzung („Sharing“)  
Seiten und größere Bereiche in mehreren Adressräumen gleichzeitig
- beliebig schachtelbar:  
64-Bit-Adressräume
- Zugriff auf Hauptspeicher wird **noch langsamer**:  
2 oder mehr Umsetzungsstufen

# Hierarchiebildung



# Problem 2: Schnellere Abbildung



## Translation Look Aside Buffer (TLB)

- schneller Speicher für schon ermittelte Abbildungen virtueller auf physische Adressen
- wird vor Durchsuchen der Seitentabellen inspiziert
- muss eventuell bei Adressraumwechsel gelöscht werden

# Alternative Implementierungen

---

## **Alternative: Inverse Seitentabellen**

- zu jeder Kachel wird Prozess-Id, Seitennummer geführt
- bei TLB-miss wird gesucht
- Implementierung als Hash-Tabellen (Hashed Page Tables)
- z. B. PowerPC-Architektur
- Vorteile:      kleine Seitentabellen,  
                         nur abhängig von Anzahl der Kacheln
- Nachteile:    Suchaufwand, keine Hierarchiebildung (z. B.  
                         Schreibschutz für größere Bereiche),  
                         Sharing aufwändig

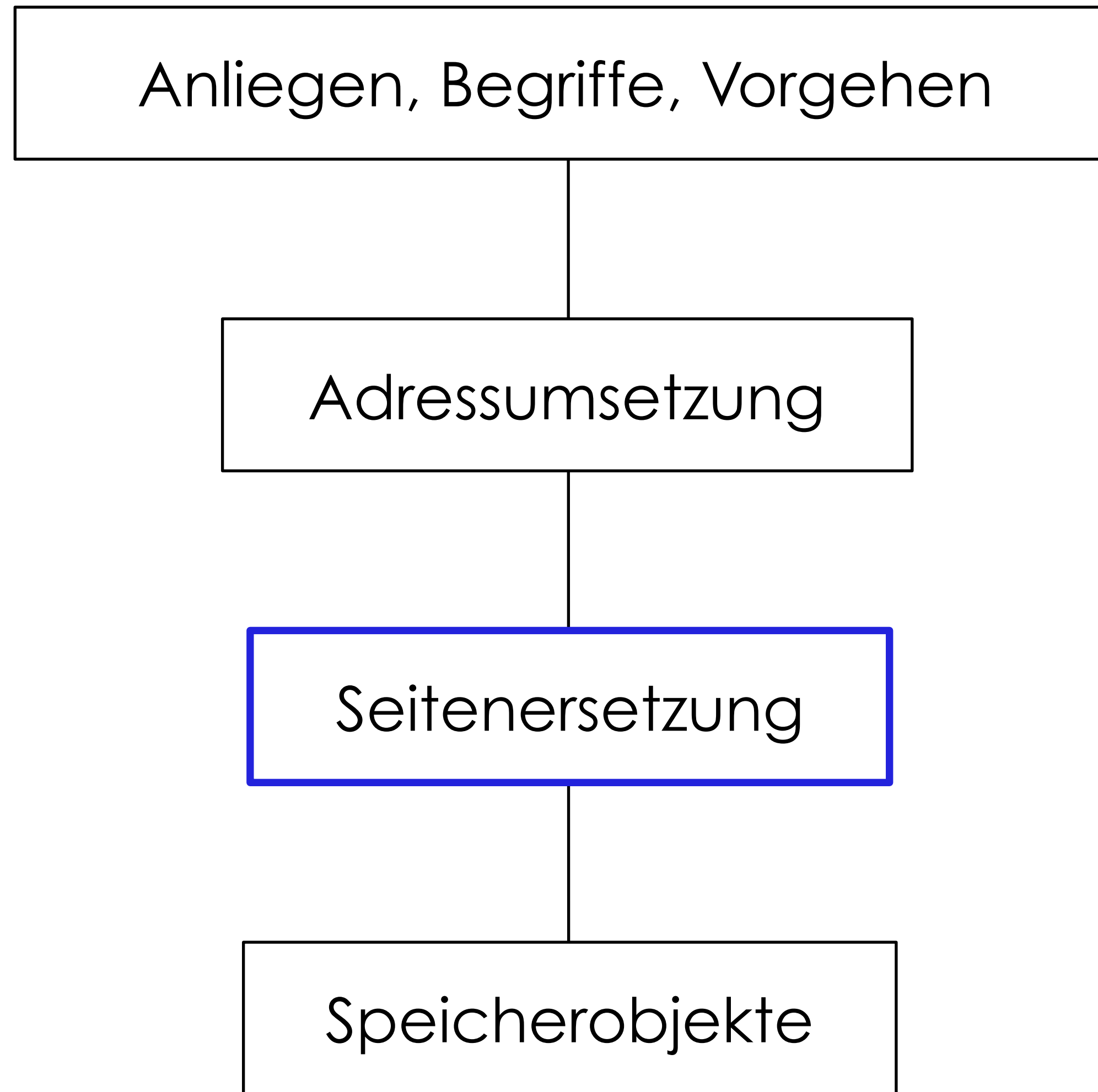
# Alternative Implementierungen

---

## **Alternative: Software-implementierte Seitentabellen**

- MMU benutzt nur TLB
- Seitentabellen oder beliebige andere Datenstruktur wird in Software vom Betriebssystem verarbeitet
- Seitenfehlerbehandlung lädt TLB neu
- z. B. Alpha-Architektur
- Vorteil: total flexibel, Prozess-spezifische Verwaltung
- Nachteil: hoher Overhead bei TLB-misses

# Wegweiser: Virtueller Speicher



# Hauptspeicherverwaltung

---

## Aufgaben

- Zuteilung von Kacheln bei Speicheranforderung
- Buchführung: welche Kacheln sind belegt und durch wen
- Entziehen von Kacheln bei Speicherknappheit
- Frage: Welche Kachel wird bei Knappheit verdrängt?



# Seitenersetzungsverfahren

---

## Problem

Eine Kachel wird benötigt, aber der Speicher ist voll. Eine Seite ist zu verdrängen.

## Wie wird verdrängt?

Kachel aus Seitentabelle austragen (TLB-Eintrag löschen)  
falls modifiziert: Sichern des Inhalts auf persistentem Speicher

## Welche Seite wird verdrängt?

- optimal: die Seite wird verdrängt, die in Zukunft am längsten nicht benötigt wird (im Besten Fall nie wieder)
- Realität: Heuristiken basierend auf Lokalitätsprinzip, Nutzung der Attribute *modified (dirty)* und *referenced (accessed)* aus den Seitentabellen-Einträgen

# Strategie: FIFO – First In First Out

---

Verdränge die älteste Seite, d.h. die Kachel, die schon am längsten ihren jetzigen Inhalt hat

## **Vorteil**

- keine Information über tatsächliches Referenzverhalten nötig
- einfach implementierbar

## **Nachteil**

- ignoriert Lokalitätsverhalten
- dadurch mit typischem Code viele unnötige Seitenfehler
- BELADY'sche Anomalie

# Strategie: LRU – Least Recently Used

---

Verdränge am längsten nicht genutzte Seite

## **Vorteil**

- in der Praxis gute Näherung für optimalen Algorithmus

## **Nachteil**

- sehr aufwendige Realisierung
- jeder Speicherzugriff müsste berücksichtigt werden
- Hardware müsste alle Zugriffe protokollieren
- daher nur Annäherungen per Software möglich

# Strategie: NRU – Not Recently Used

---

Verdränge eine aktuell ungenutzte Seite, d.h. Wahl einer beliebigen Kachel in folgender Rangfolge:

- nicht benutzt, nicht modifiziert
- benutzt, nicht modifiziert
- nicht benutzt, modifiziert
- benutzt, modifiziert

## **Ermitteln der zeitlichen Lokalität**

- durch regelmäßiges Zurücksetzen der referenced-Bits
- nach Zeitintervall oder reihum

# Strategie: Second Chance

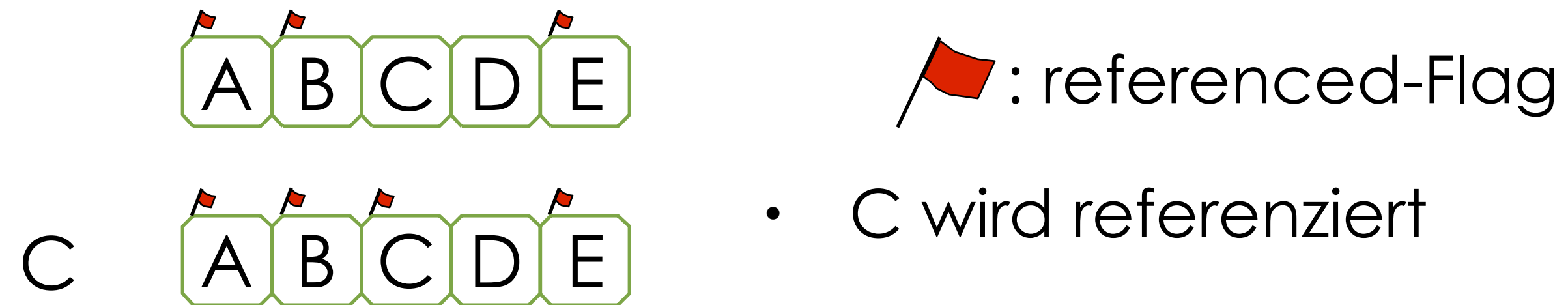
Verdränge die älteste (nach FIFO), im Inspektionsintervall nicht referenzierte (NRU) Seite



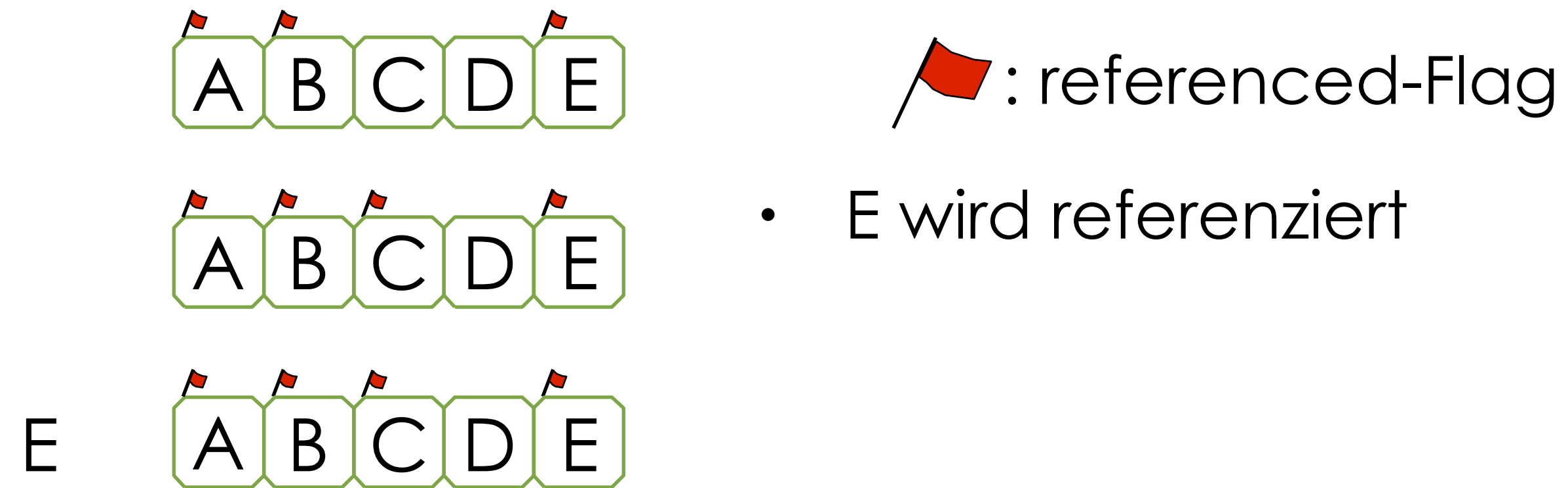
 : referenced-Flag

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

# Strategie: Second Chance



# Strategie: Second Chance



# Strategie: Second Chance



X

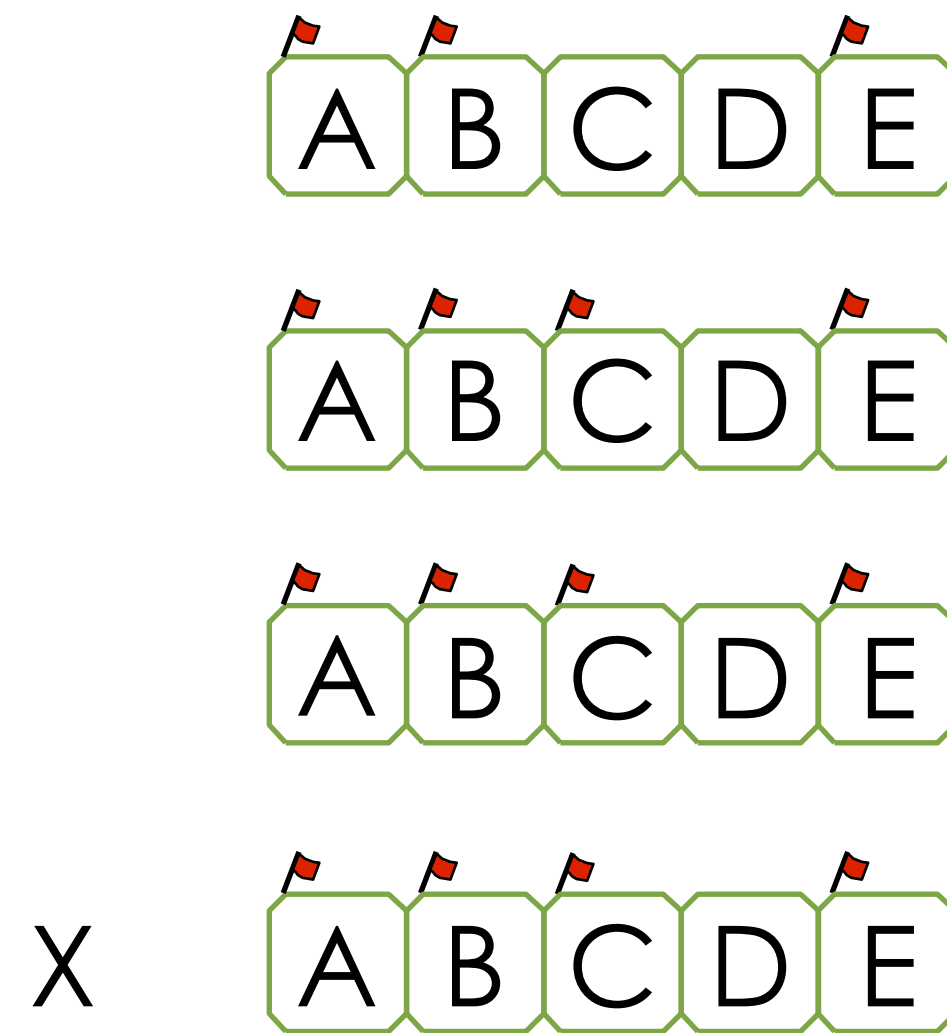


 : referenced-Flag

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



# Strategie: Second Chance



 : 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

# Strategie: Second Chance



 : 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

# Strategie: Second Chance



 : 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

# Strategie: Second Chance



A

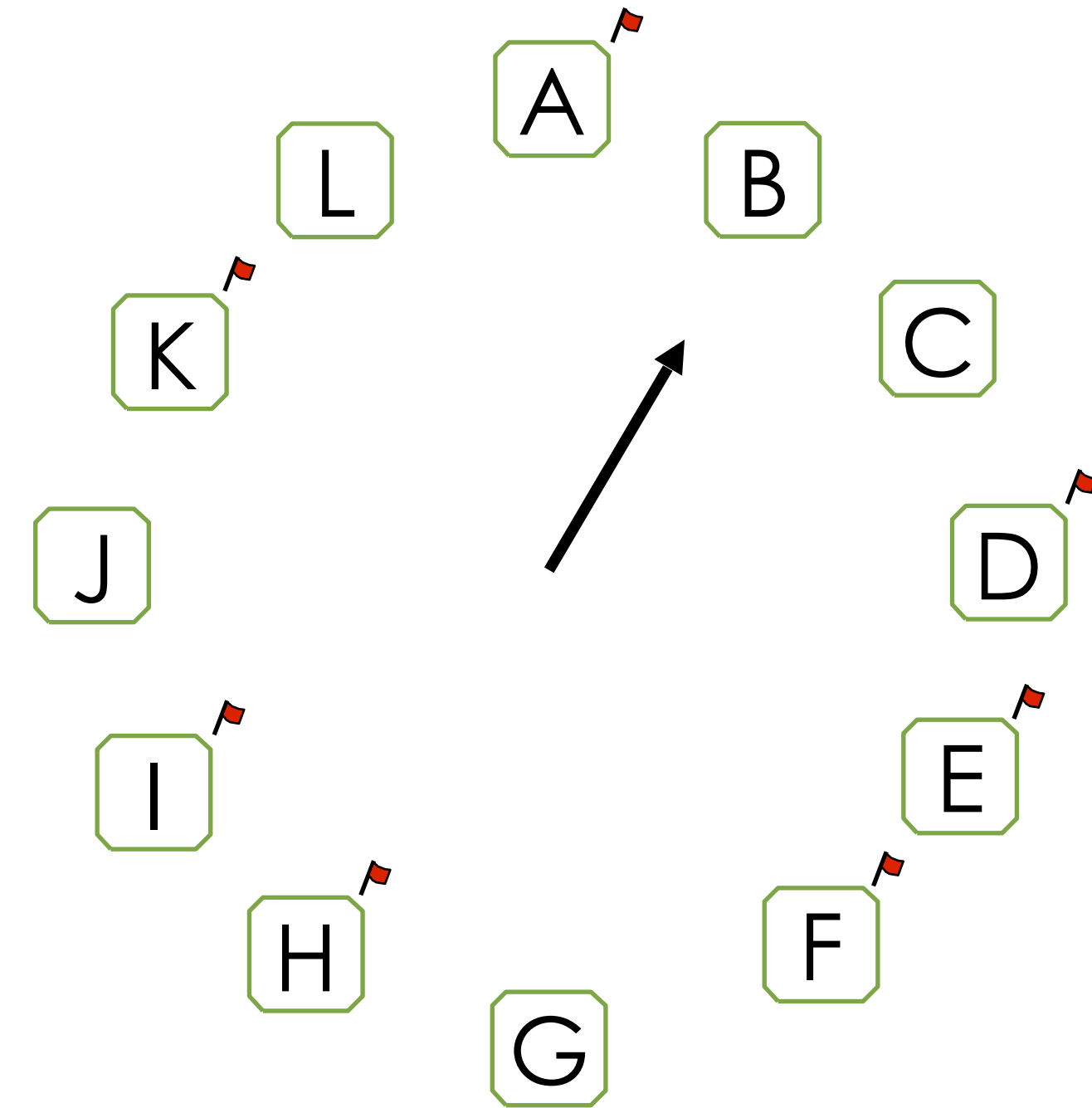


 : referenced-Flag

- A wird referenziert → „2. Chance“

# Strategie: Clock-Algorithmus

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



Elegantere Implementierung von Second Chance, ansonsten gleiches Verhalten

## **Vorteil**

effizient implementierbar, gute Näherung an LRU

# Strategie: Aging

---

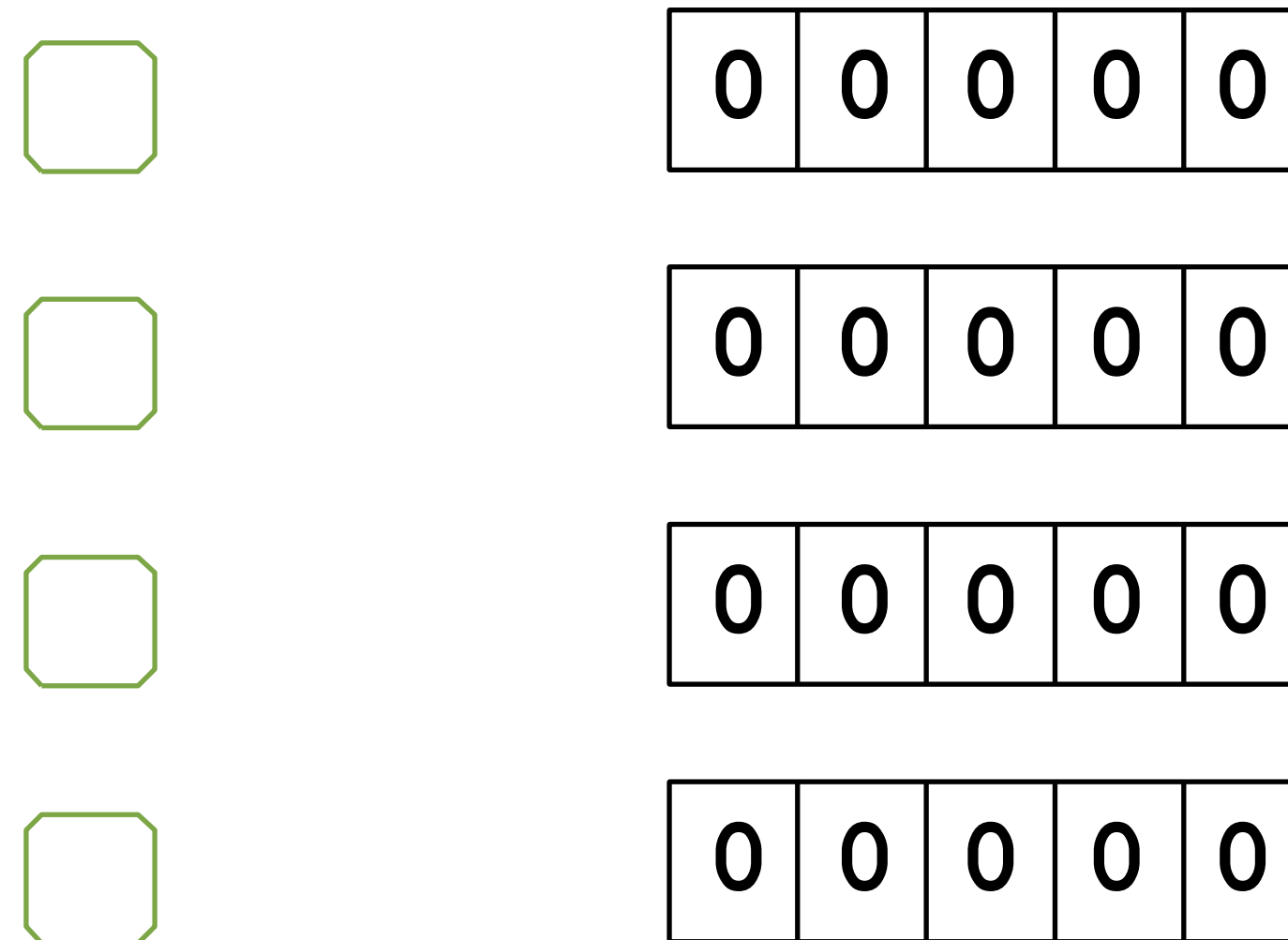
- bessere LRU-Näherung durch genaueres Speichern der Zugriffs-Historie
- Kachel mit ältestem Inhalt wird verdrängt
- Analogie: Geburtsjahr
  - 1997
  - 1998
  - 1999
- niedrige Zahl → hohes Alter

## **Nachteil**

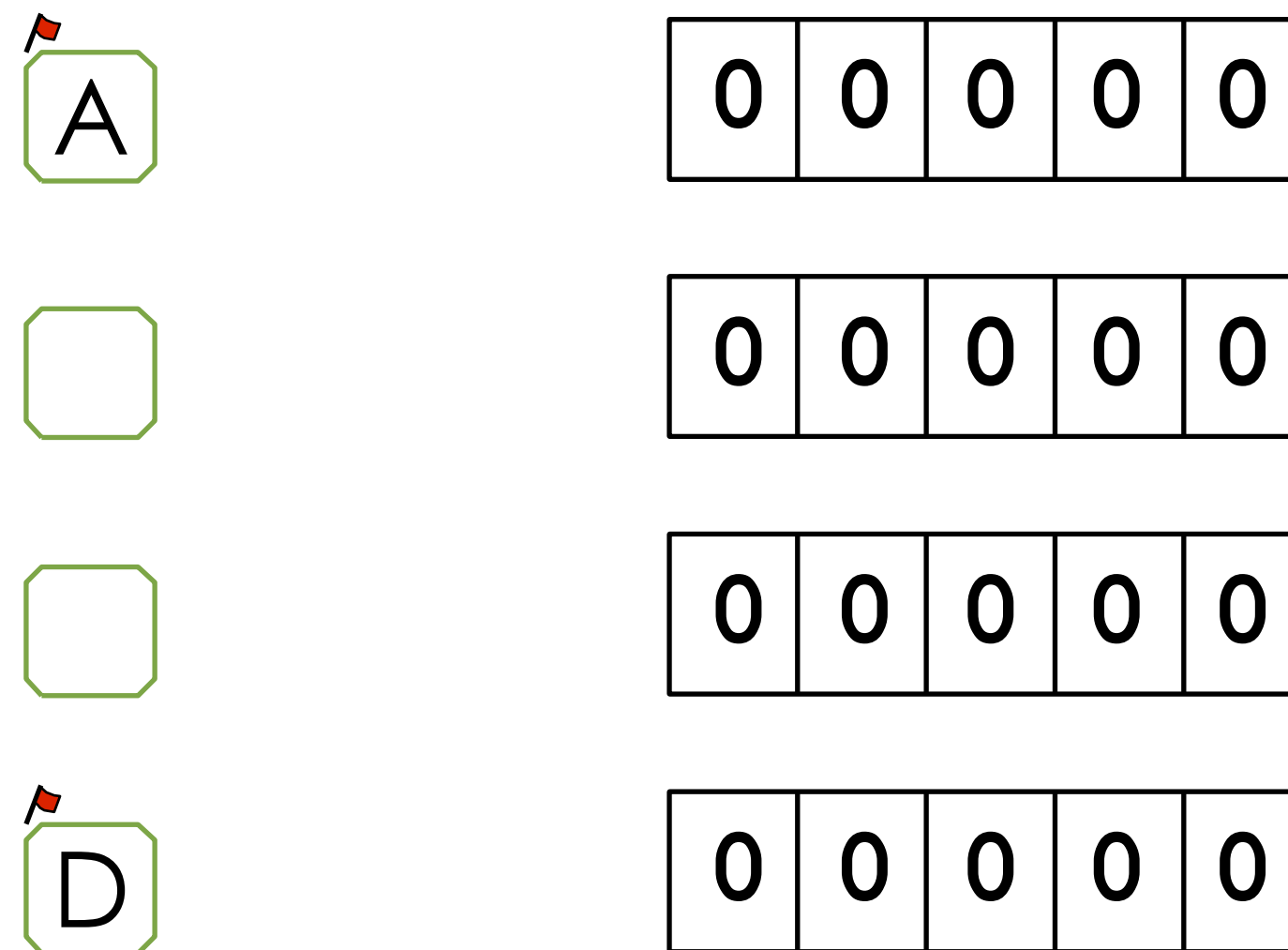
- höherer Overhead durch Auslesen der *referenced*-Bits in regelmäßigen Zeitintervallen (Ticks)

# Strategie: Aging

---



# Strategie: Aging





# Strategie: Aging

Tick 1

A



1	0	0	0	0
---	---	---	---	---

0	0	0	0	0
---	---	---	---	---

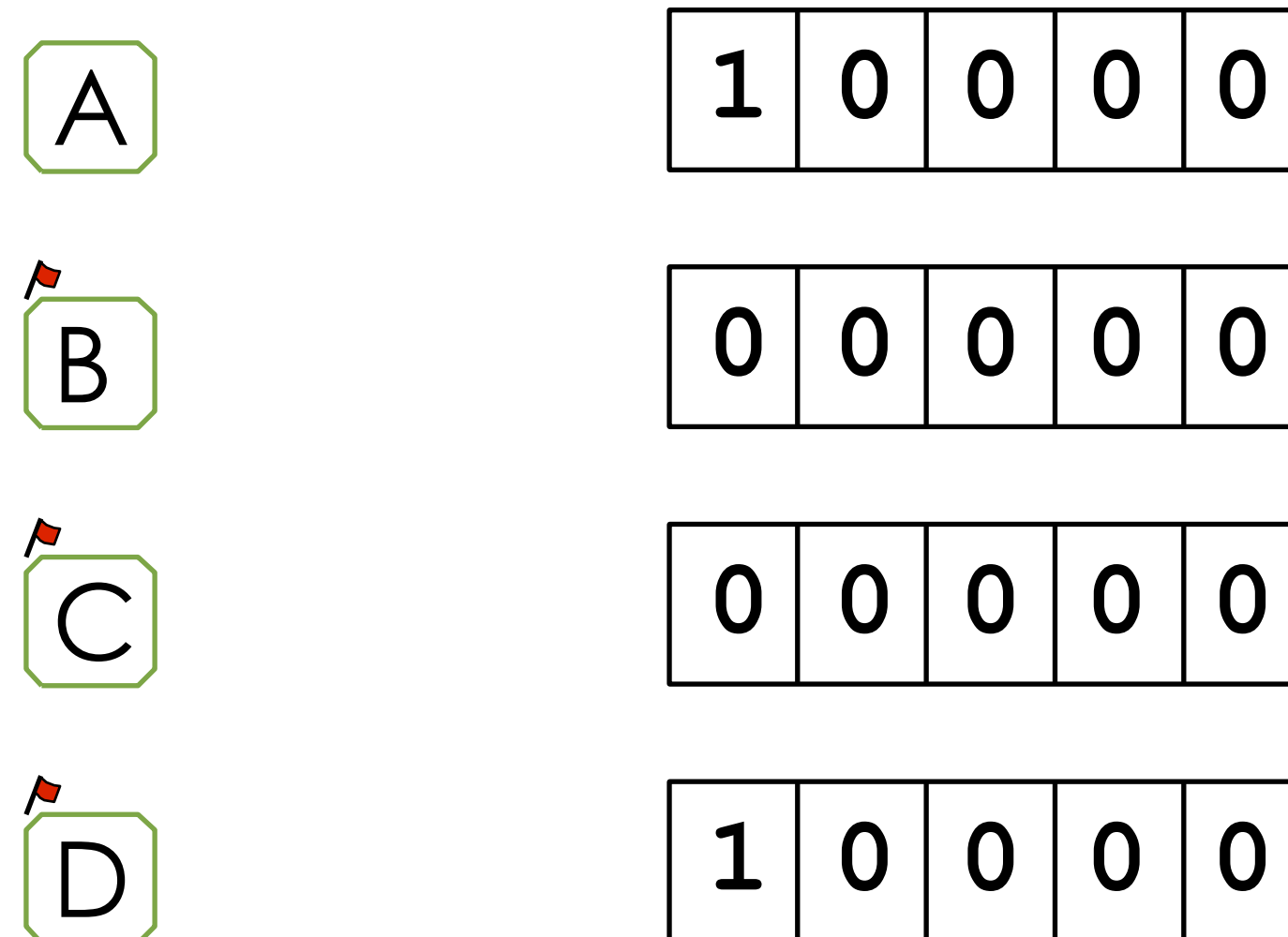
0	0	0	0	0
---	---	---	---	---

D



1	0	0	0	0
---	---	---	---	---

# Strategie: Aging



# Strategie: Aging

Tick 2

A

0	1	0	0	0
---	---	---	---	---

B



1	0	0	0	0
---	---	---	---	---

C





1	0	0	0	0
---	---	---	---	---

D



1	1	0	0	0
---	---	---	---	---

# Strategie: Aging

 A	0	1	0	0	0
B	1	0	0	0	0
 C	1	0	0	0	0
D	1	1	0	0	0

# Strategie: Aging

Tick 3

A



1	0	1	0	0
---	---	---	---	---

B

0	1	0	0	0
---	---	---	---	---

C



1	1	0	0	0
---	---	---	---	---

D

0	1	1	0	0
---	---	---	---	---

# Strategie: 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

# Strategie: Arbeitsmengenmodell

## Problem

- bisherige Strategien ignorieren Prozess-Zugehörigkeit von Kacheln, keine Fairness
- inaktiver Prozess kann vollständig verdrängt werden  
„Seitenflattern“ (Thrashing)

## Arbeitsmenge / Working Set

- Betrachtung der Seitenreferenzfolge eines Prozesses durch ein „Fenster“ der Länge  $T$  (sog. Arbeitsmengen-Parameter)
- Arbeitsmenge  $w(t, T)$  zum Zeitpunkt  $t$  bzgl.  $T$ :  
Menge der im Intervall  $[t - T, t]$  referenzierten Seiten
- Annahme: kleine Menge durch mehrfache Zugriffe auf wenige Seiten

# Strategie: Arbeitsmengenmodell

---

## Vorgehen

- Bestimmung der Arbeitsmenge durch regelmäßiges Auslesen der *referenced*-Bits
- für aktive und bereite Prozesse muss sich mindestens die Arbeitsmenge im Speicher befinden

## Nachteile

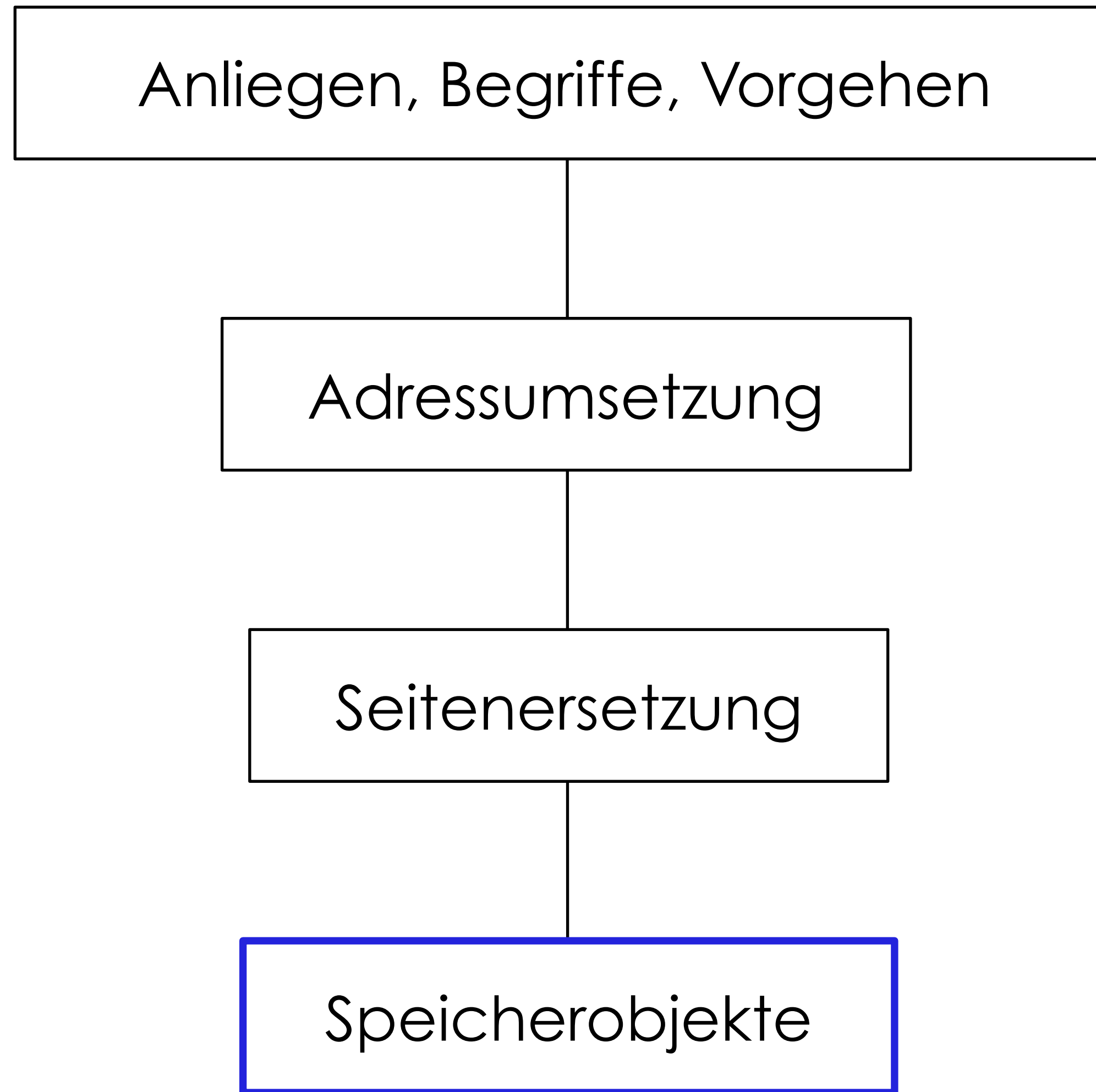
- wenn alle Kacheln in einer Arbeitsmenge sind, dann findet Algorithmus keinen Verdrängungskandidaten
- Reservestrategie nötig, zum Beispiel Prozess auslagern
- geeignete Wahl der Fenstergröße nötig



# Seitenverdrängung insgesamt

```
void pagefault(page) {  
  
    frame = get_free_frame();  
    if (!frame) {  
        // keine Kachel mehr frei  
        frame = replacement_strategy();  
  
        if (frame.modified) save_content(frame);  
        remove_from_pagetable(frame);  
        // aus zugehörigem Adressraum entfernen  
    }  
  
    fill_content(frame);  
    // siehe Speicherobjekte  
    insert_into_pagetable(frame);  
    // in Ziel-Adressraum eintragen  
}
```

# Wegweiser: Virtueller Speicher



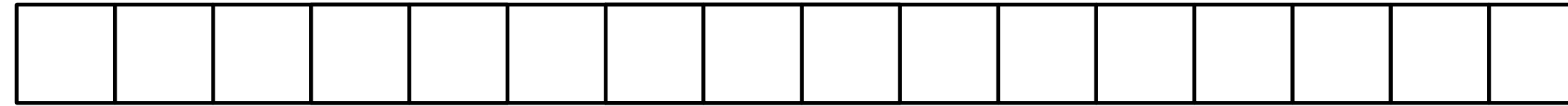
# Speicherobjekte

---

- Text/Code-, Daten- und andere Segmente eines Prozesses
- Dateien
- Gerätespeicher, zum Beispiel Grafikspeicher
- über Netzwerk verteilter Speicher

# Speicherobjekte und Regionen

---

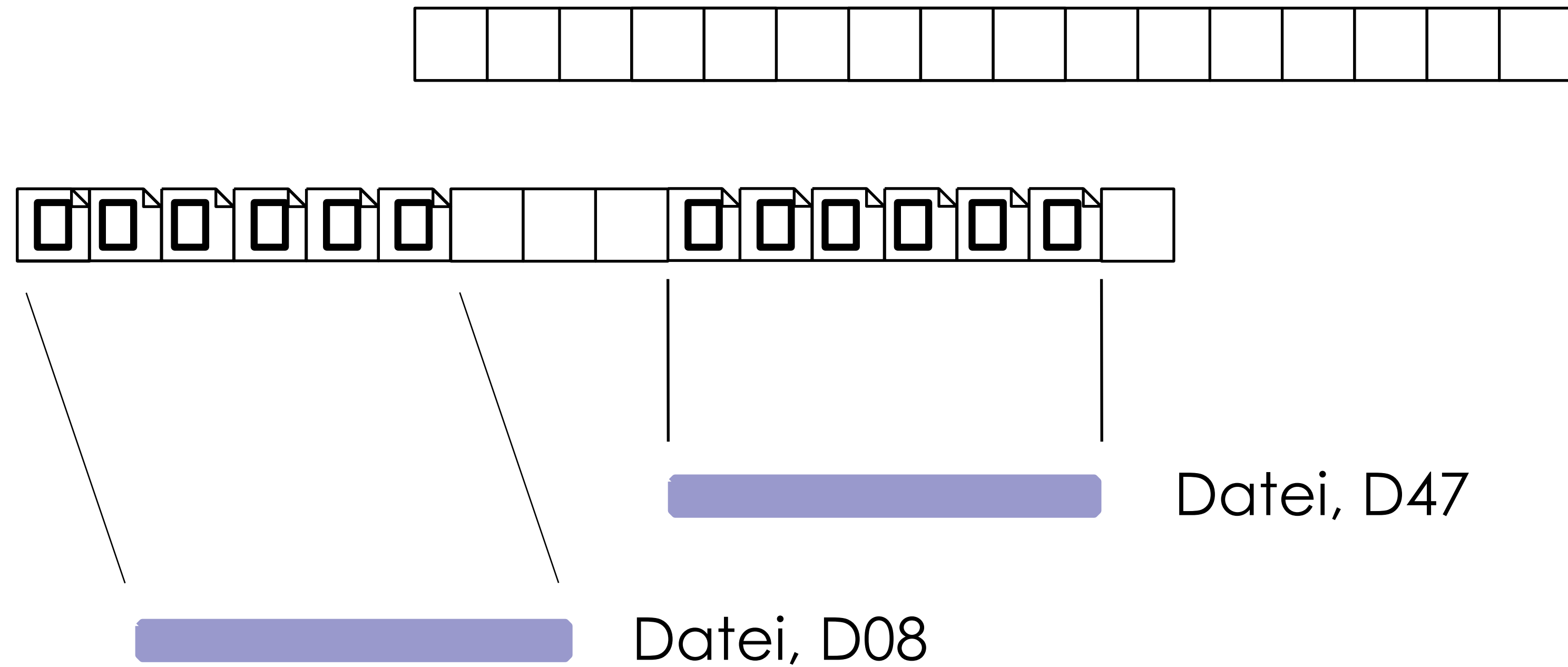


Datei, D47

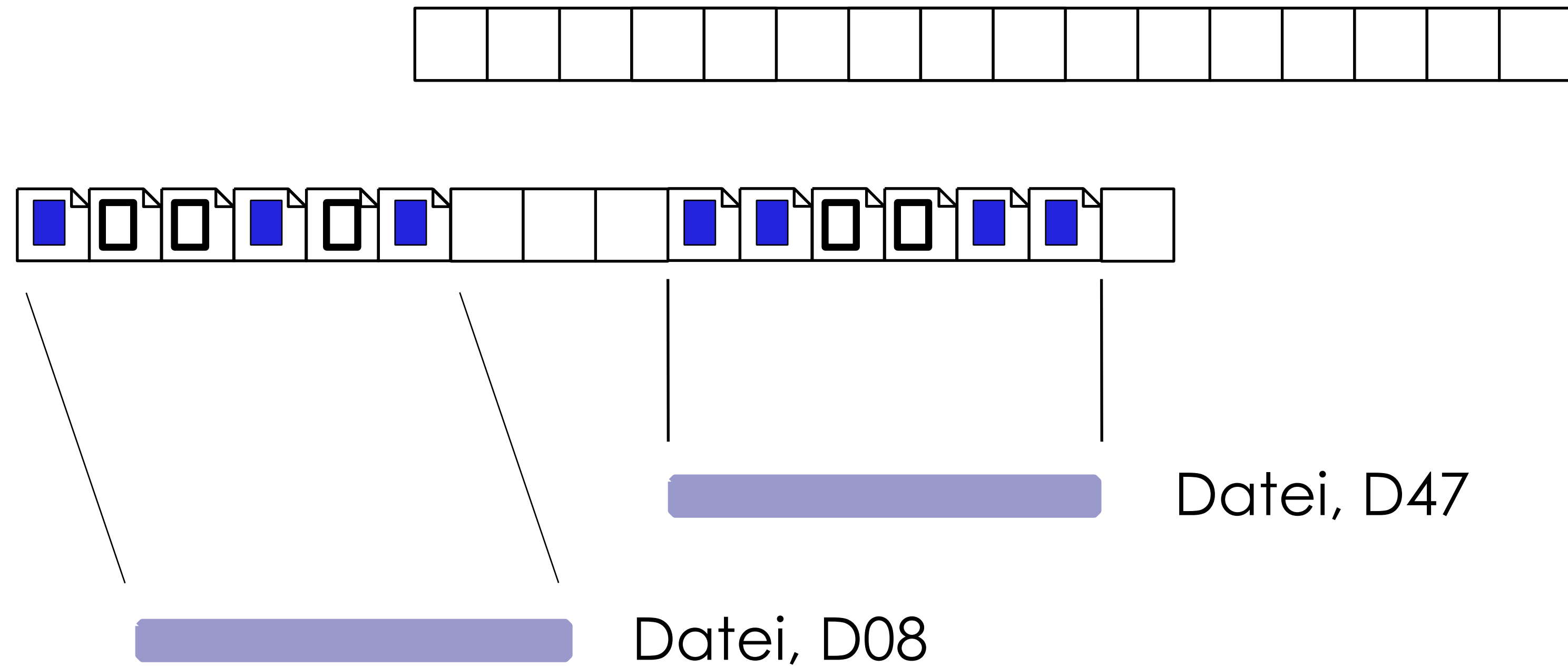


Datei, D08

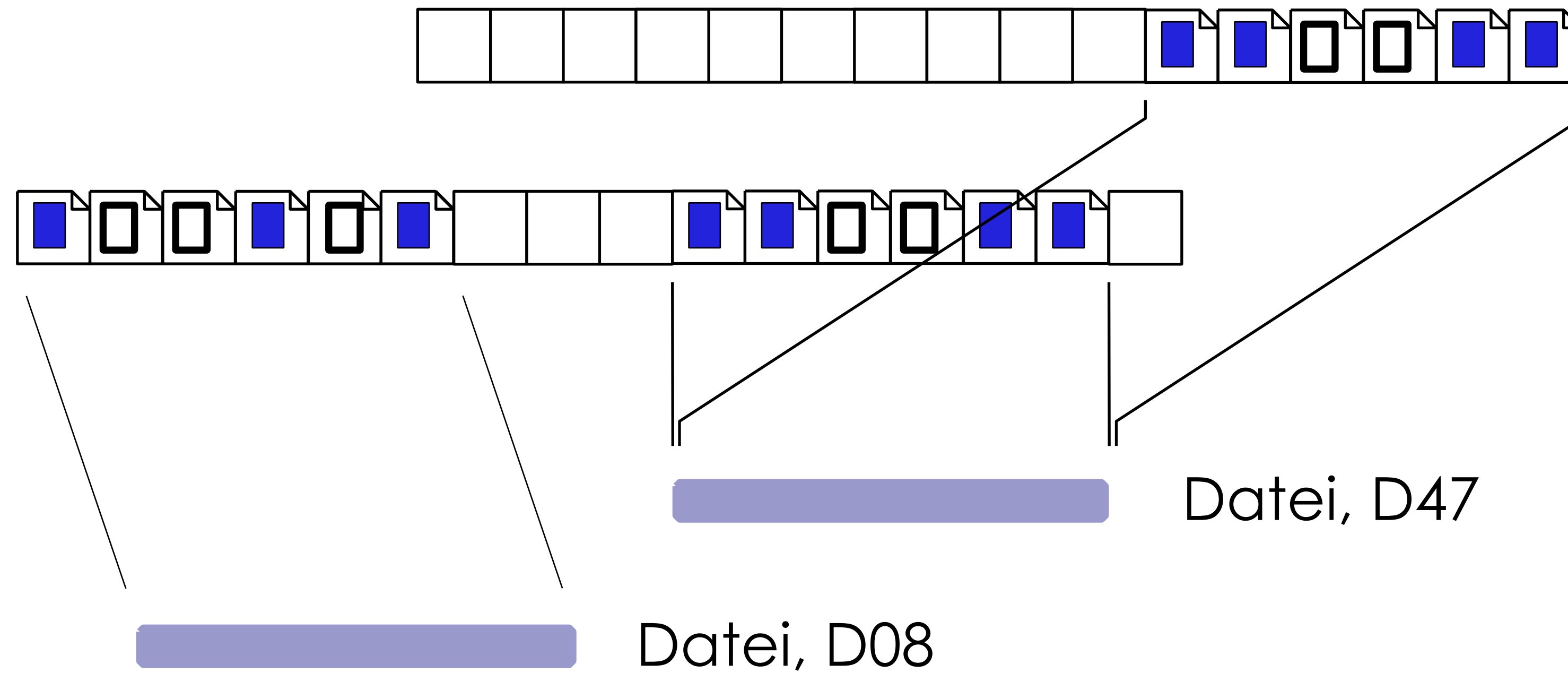
# Speicherobjekte und Regionen



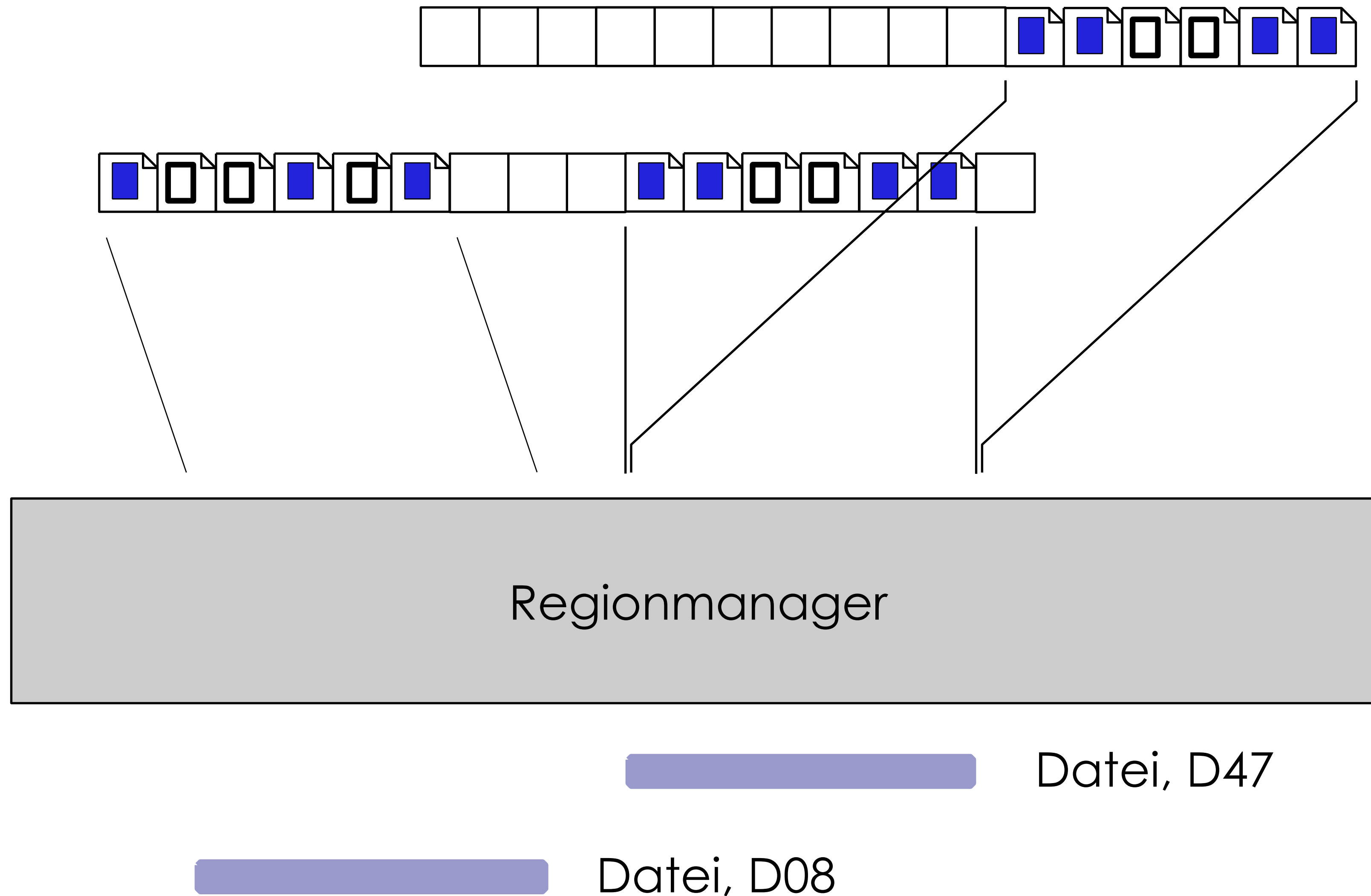
# Speicherobjekte und Regionen



# Speicherobjekte und Regionen



# Speicherobjekte und Regionen





# Regionen-Manager

---

- 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: reparabler Seitenfehler (lediglich not present)

# Aufbau der Adressraumstruktur

---

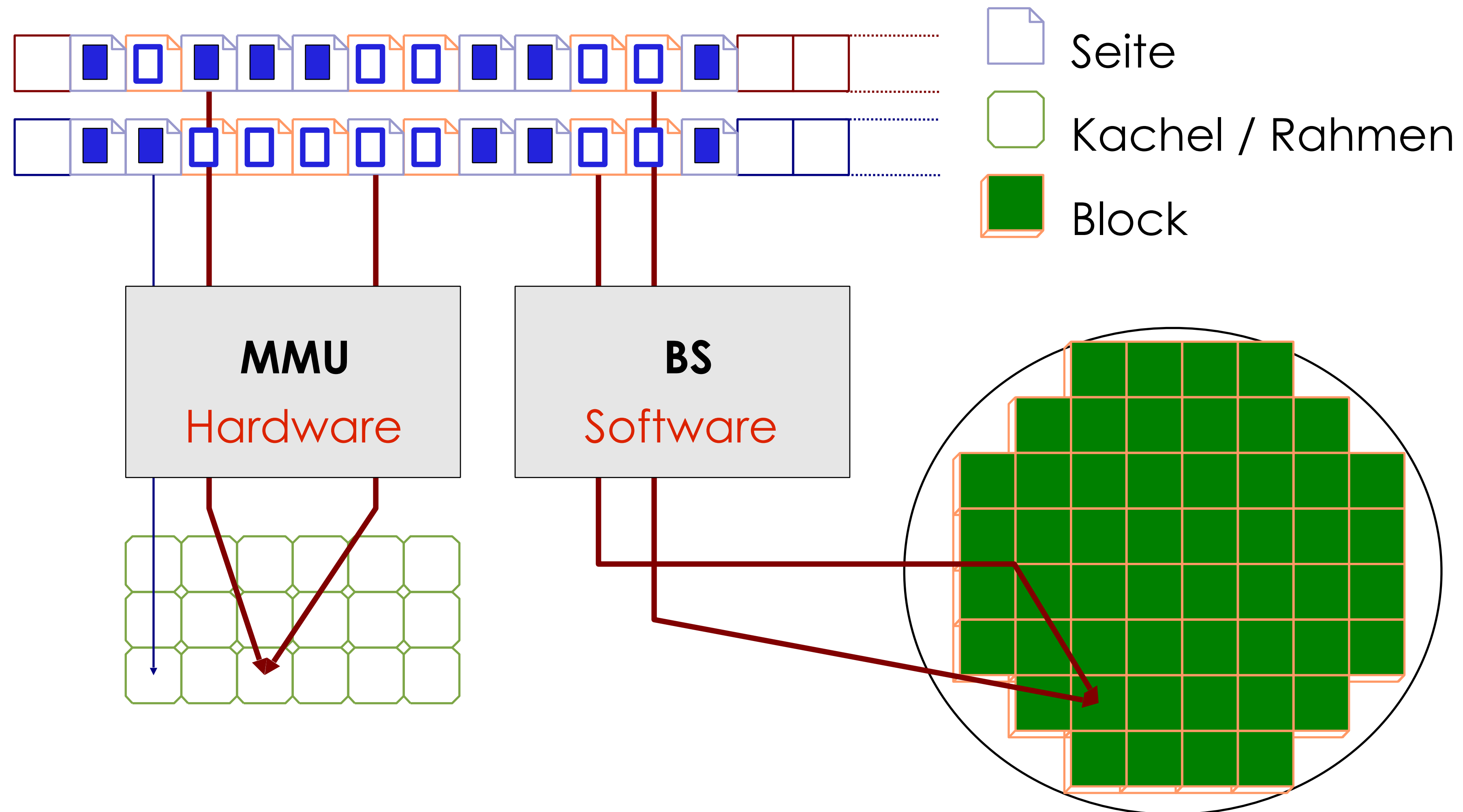
## Regionen im Adressraum

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

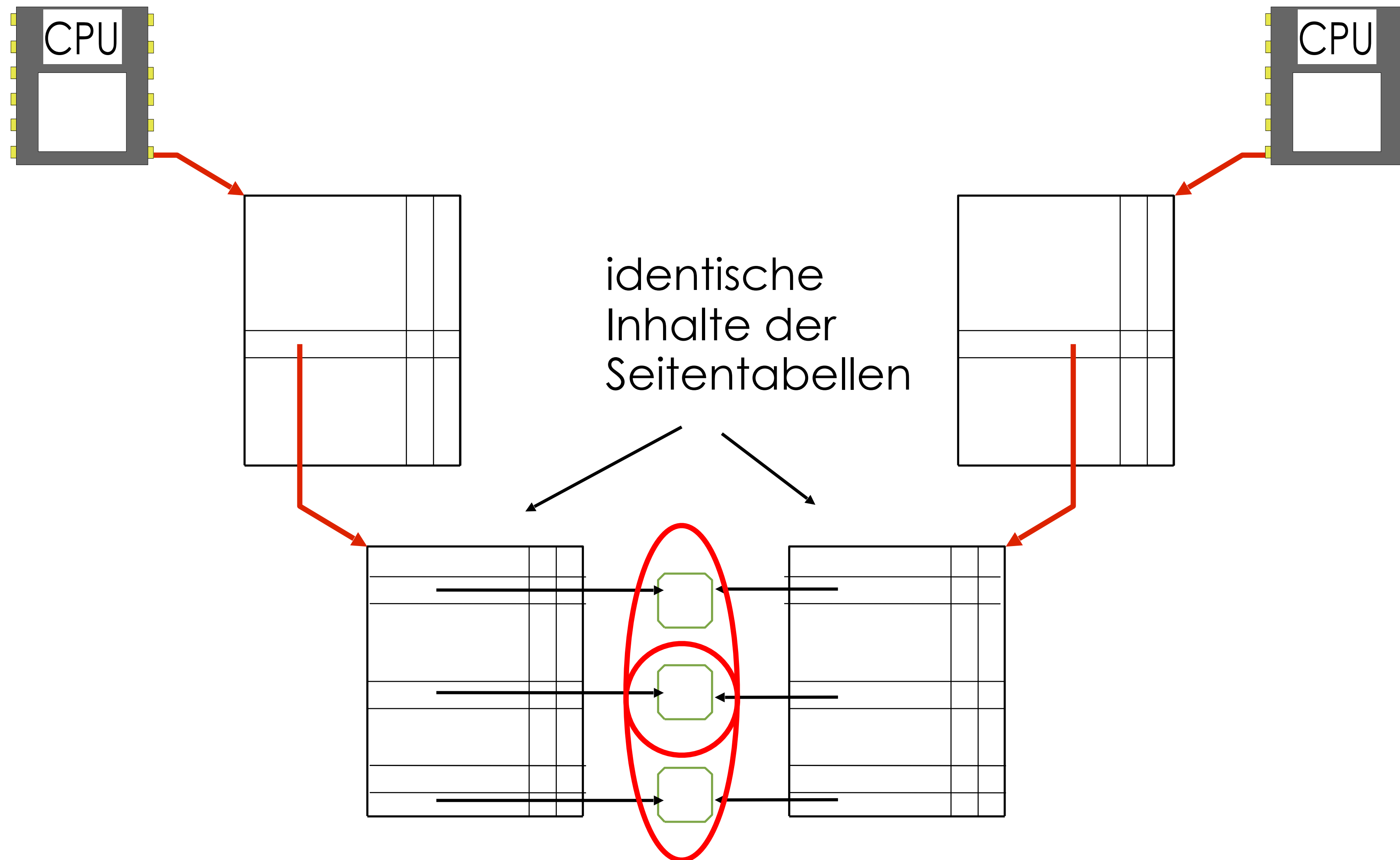
## Operationen

- **mmap** (Adresse, Länge, Zugriffsrechte, Speicherobjekt) ;
- implizit: bei Prozesserzeugung werden Code-, Daten-, Stack-Regionen gebildet (in Unix: „Segmente“)
- interner Lookup von Adresse und Zugriffsart
  - Ergebnis: Speicherobjekt und Seitennummer innerhalb Objekt
  - oder nicht definiert bzw. Rechteverletzung

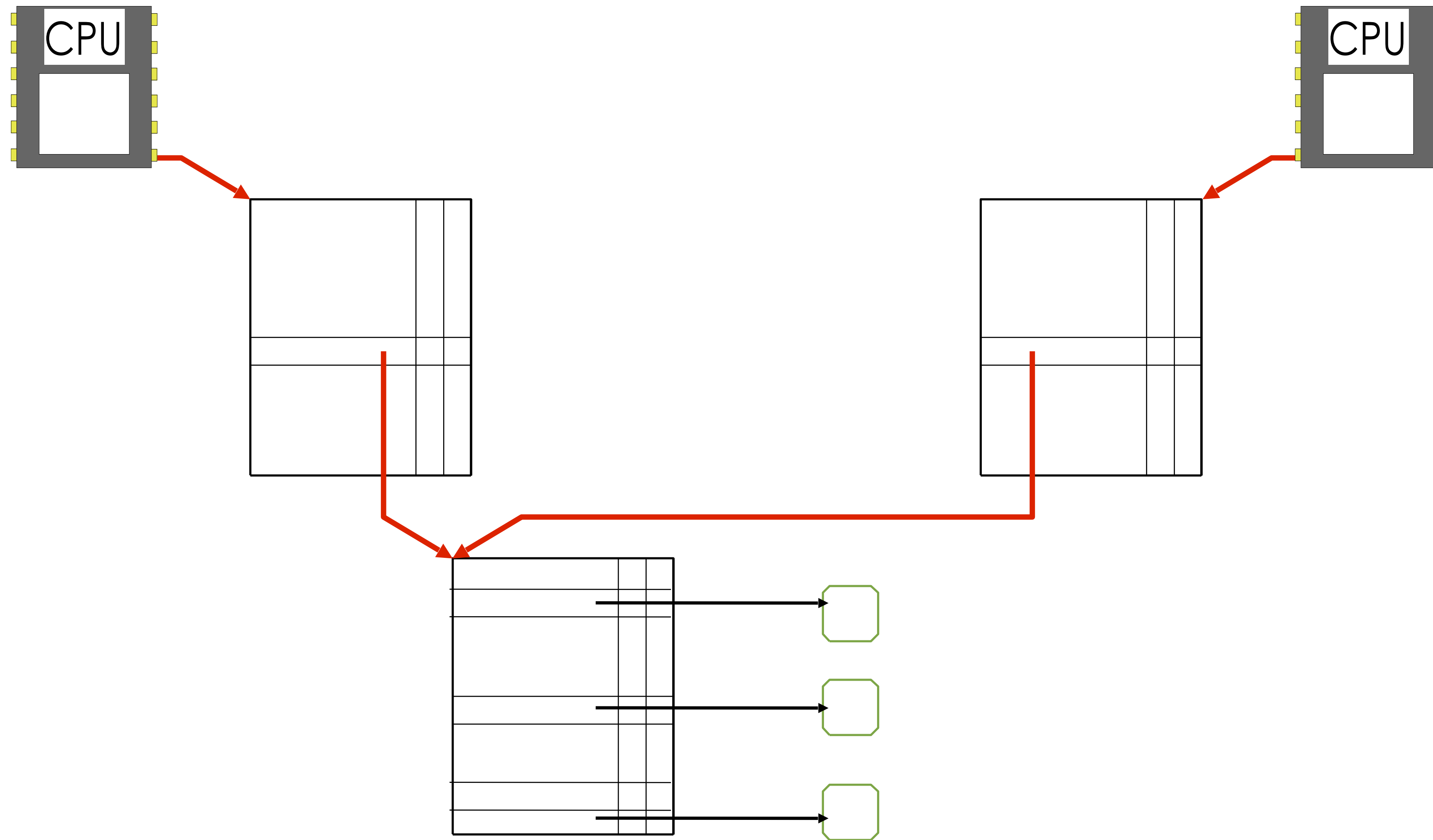
# Überlappende Adressräume - „Sharing“



# Sharing bei mehrstufigen Seitentabellen



# Sharing bei mehrstufigen Seitentabellen

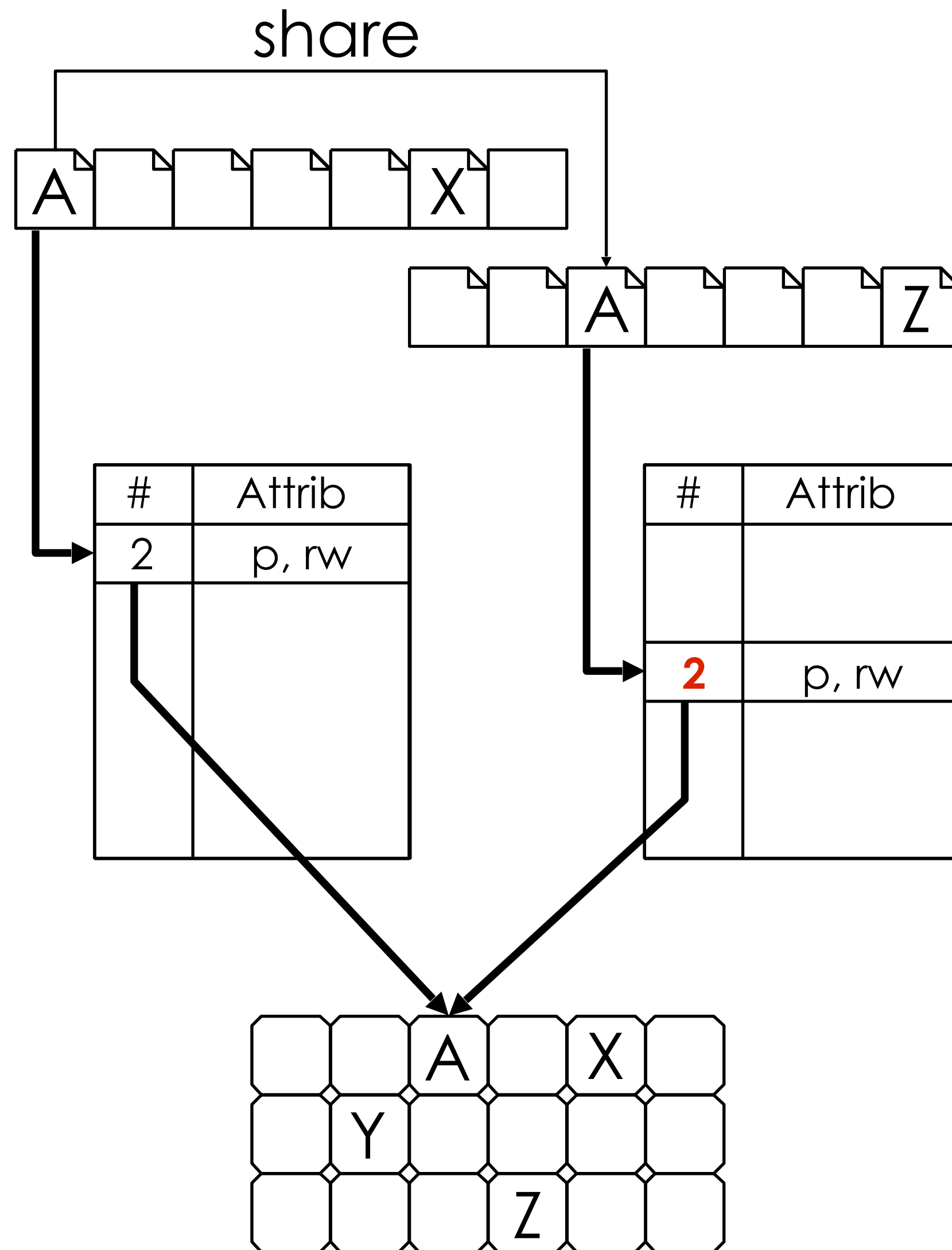


# Spezieller Einsatz von Sharing

## Verzögertes Kopieren / Lazy Copying / Copy on Write

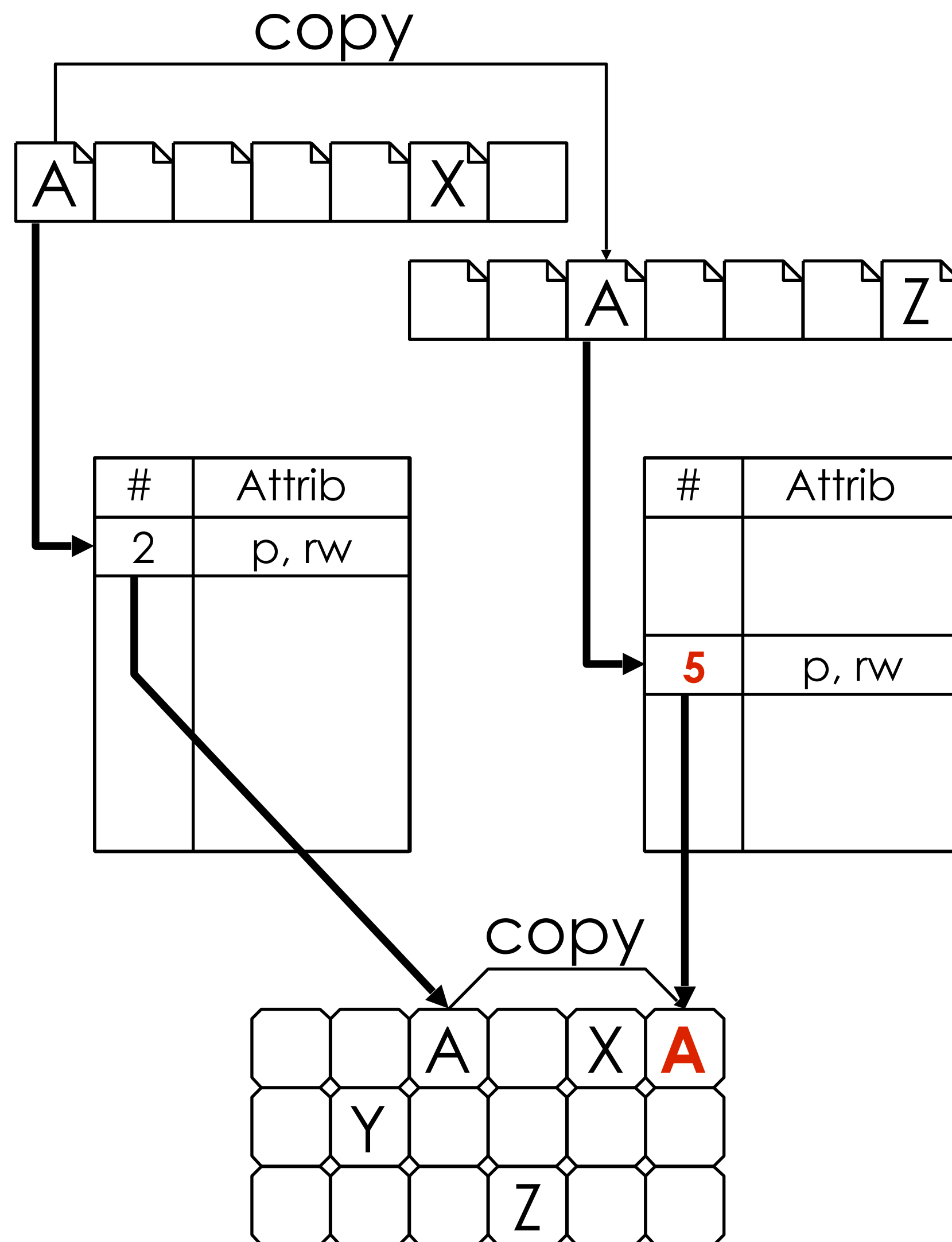
- Kopier-Anforderung: zunächst nur read-only Sharing
  - Eintragen des zu kopierenden Bereichs an Zieladresse
  - beide gegen Schreiben schützen
- verzögertes Kopieren: entkoppelt, nur bei Bedarf
  - beim ersten schreibenden Zugriff → Seitenfehler
  - Behandlung:
    - neue Kachel allokalieren, Inhalt echt kopieren
    - neue Kachel ohne Schreibschutz in Seitentabelle eintragen
- Nutzung bei **fork()**, für Prozess-Schnappschüsse, ...

# Gemeinsames Nutzen von Daten



- neuen MMU Eintrag an Zieladresse

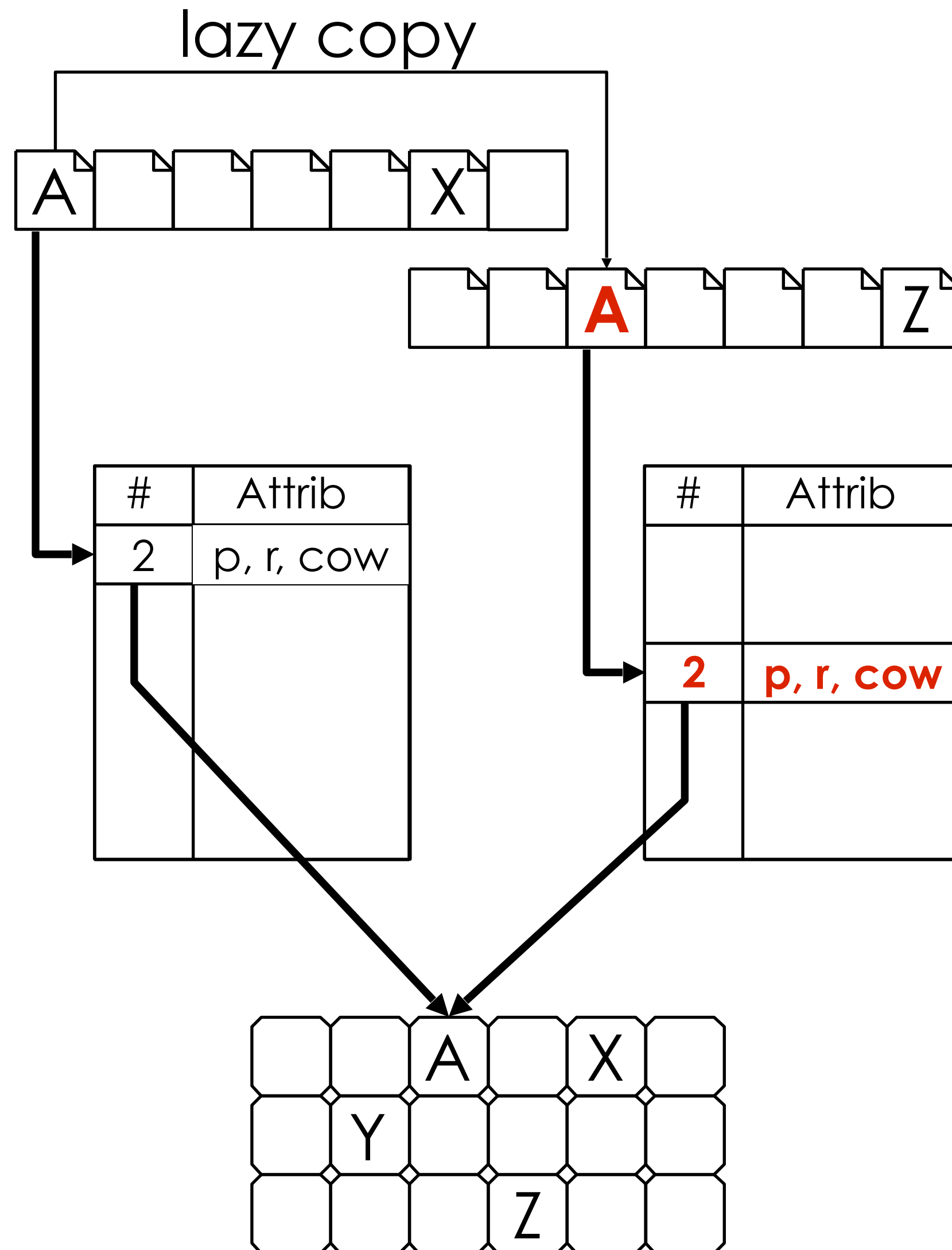
# Echtes Kopieren



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



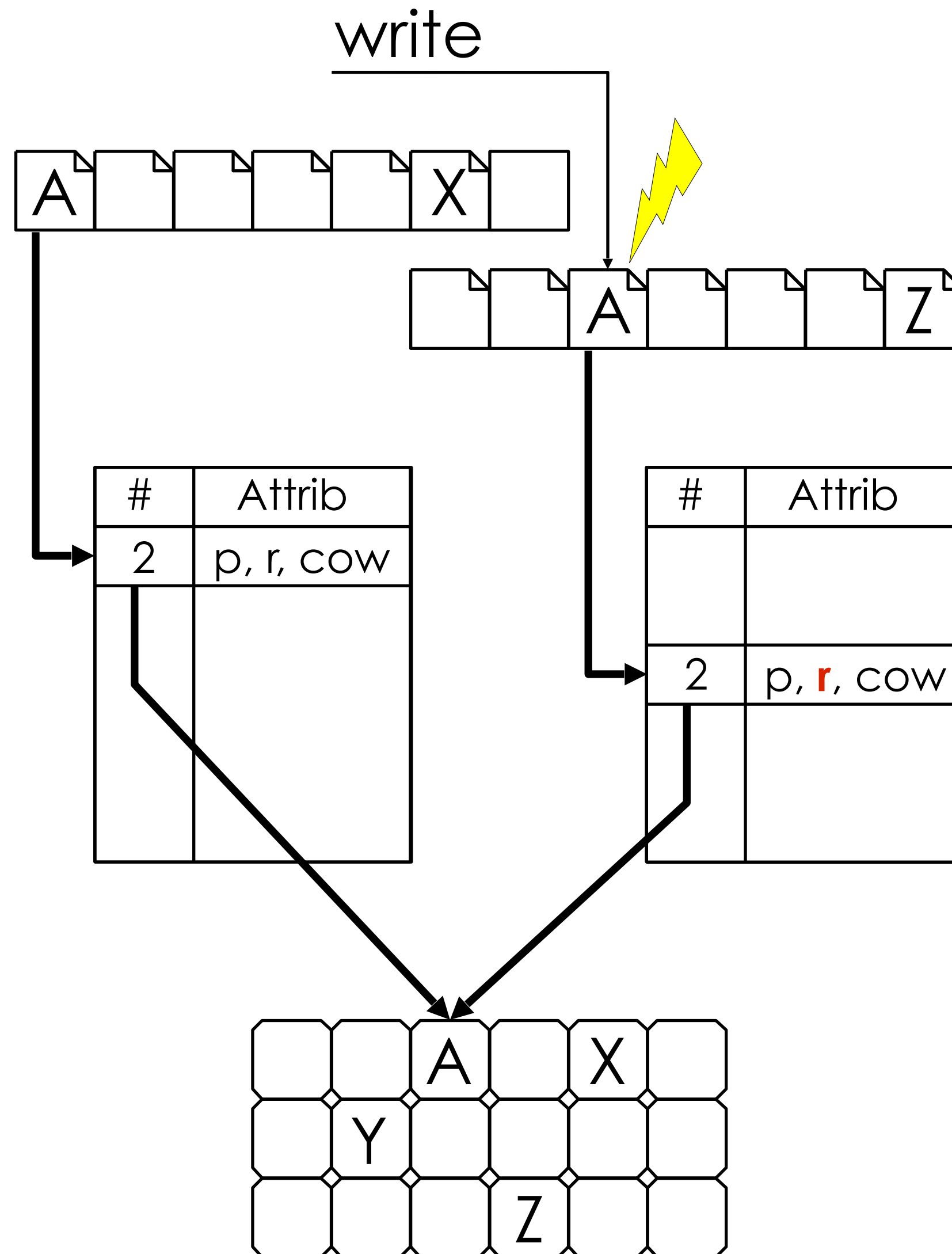
# Verzögertes Kopieren



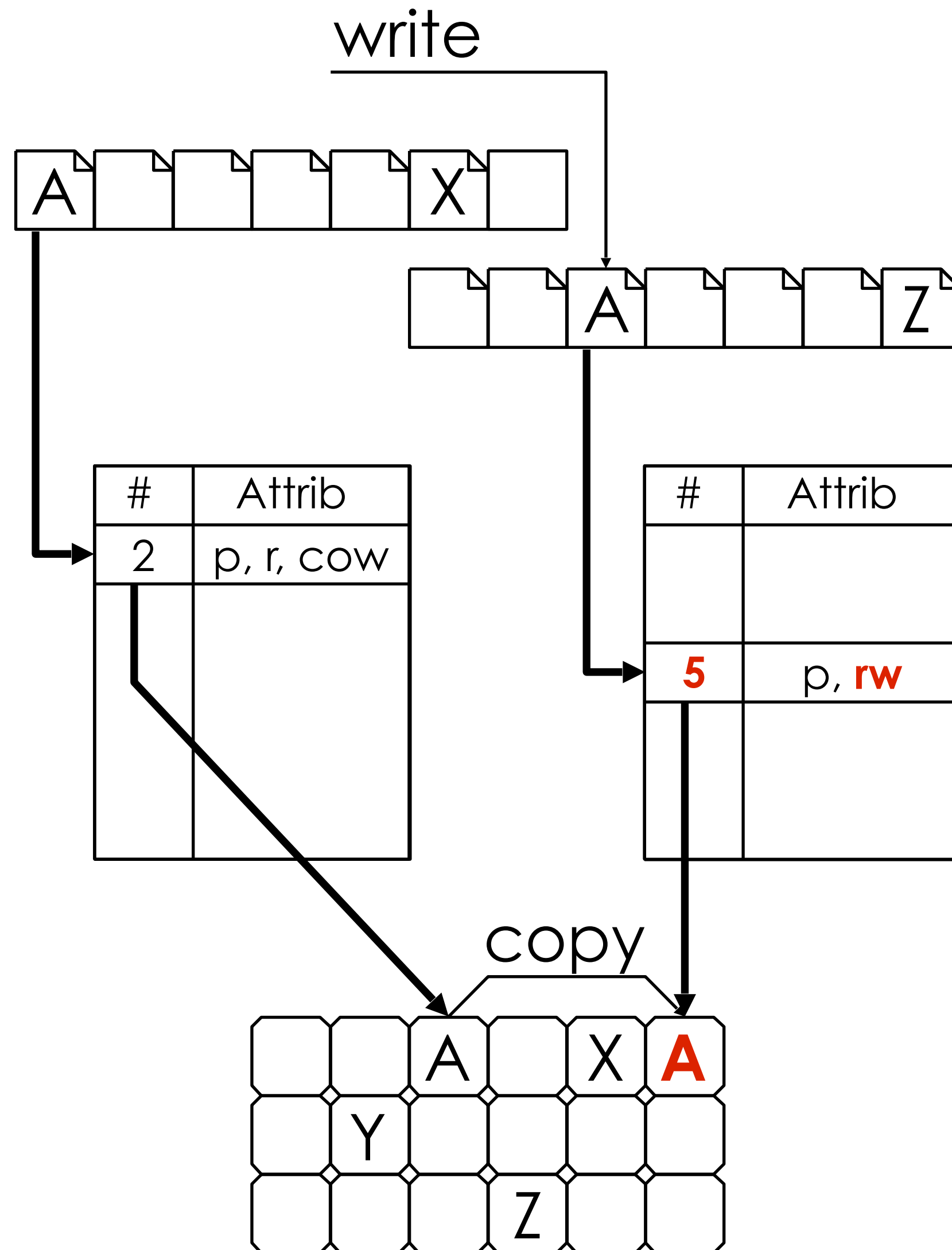
- Ändern des MMU-Eintrags an der Quelladresse:  
 $rw \rightarrow (r, cow)$
- neuen MMU-Eintrag an Zieladresse:  $(r, cow)$

# Verzögertes Kopieren

- schreibender Zugriff → Seitenfehler

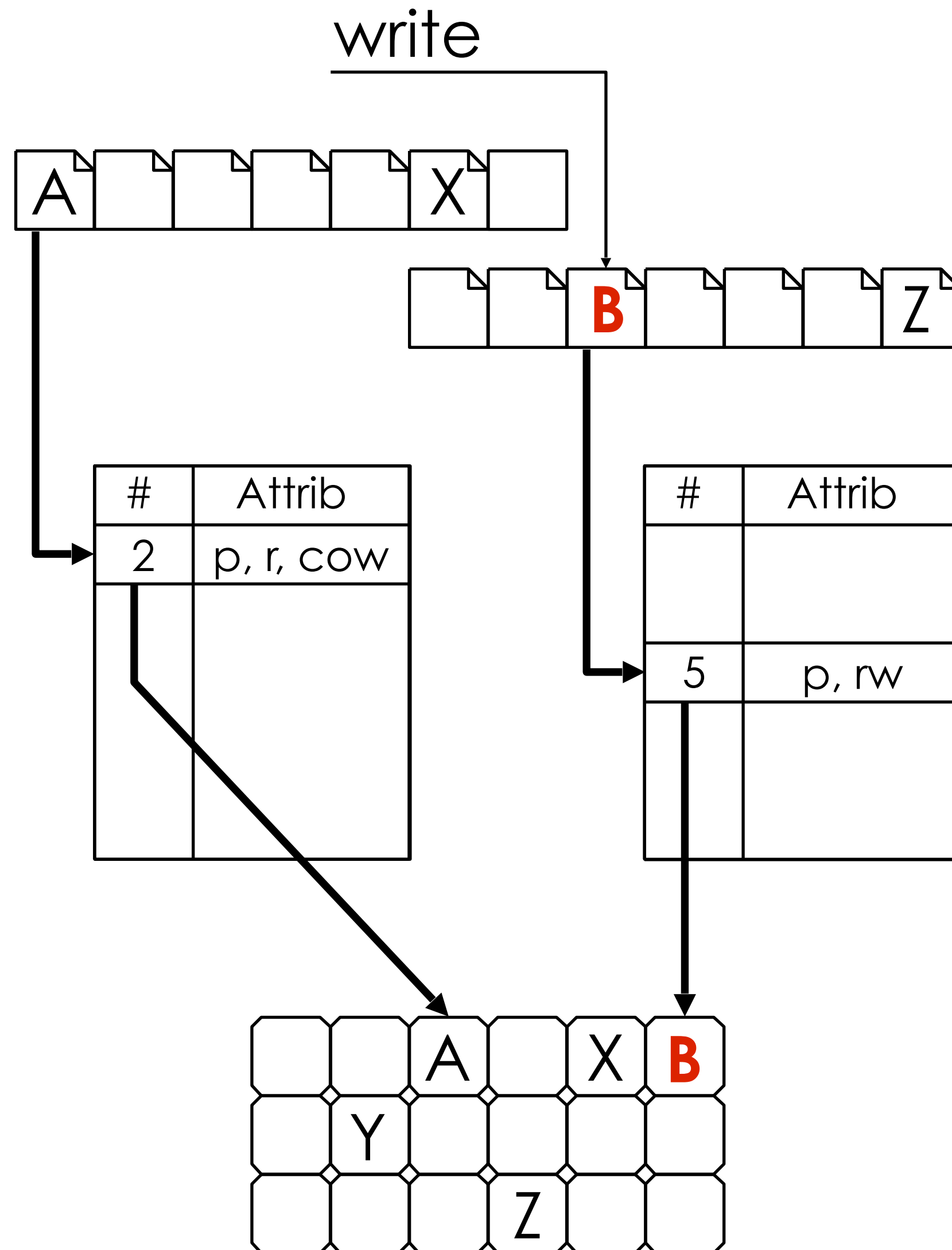


# Verzögertes Kopieren



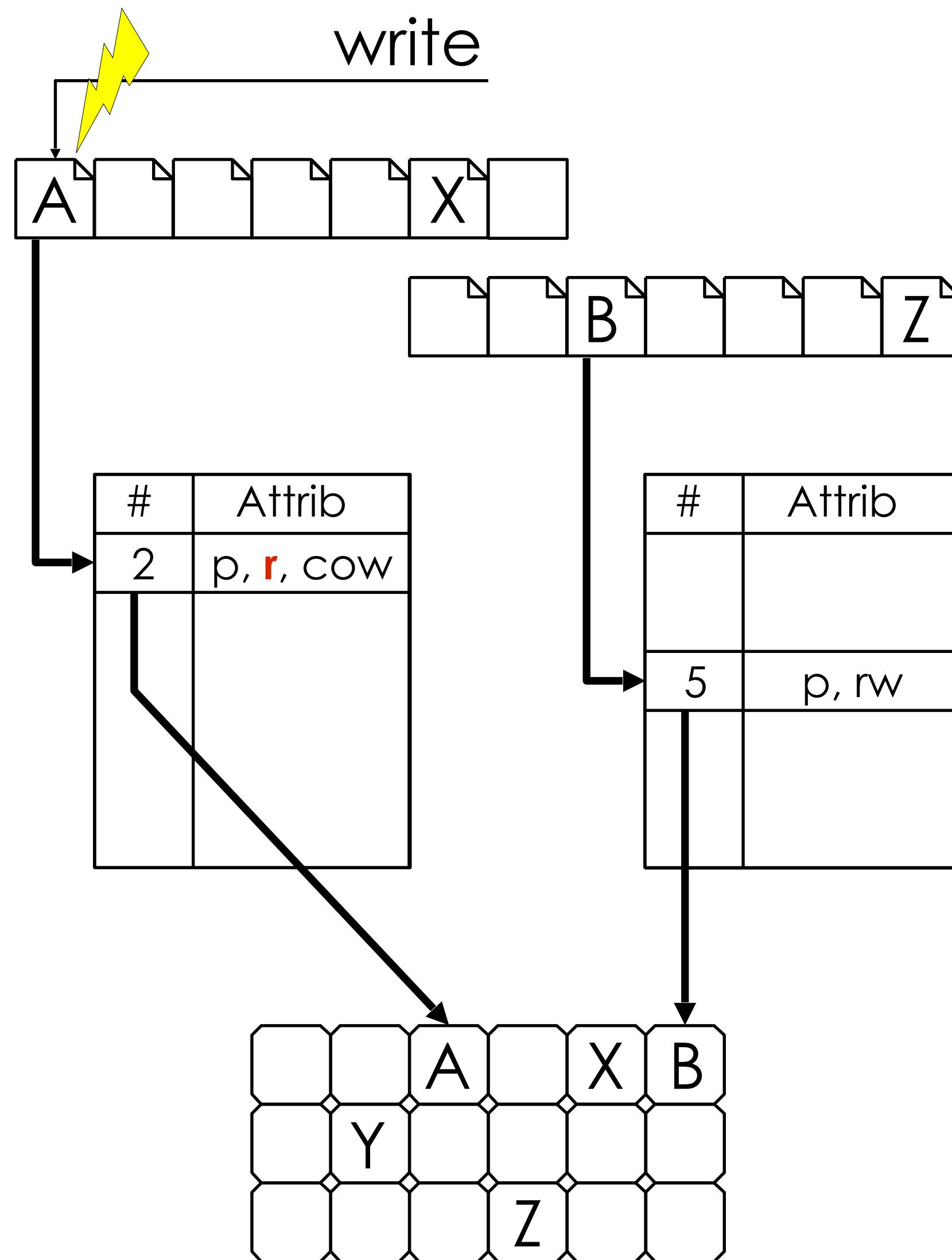
- schreibender Zugriff → Seitenfehler
- Das cow-Attribut bewirkt, dass eine neue Kachel allokiert und der Inhalt physisch kopiert wird
- neuen MMU-Eintrag an Zieladresse: rw

# Verzögertes Kopieren



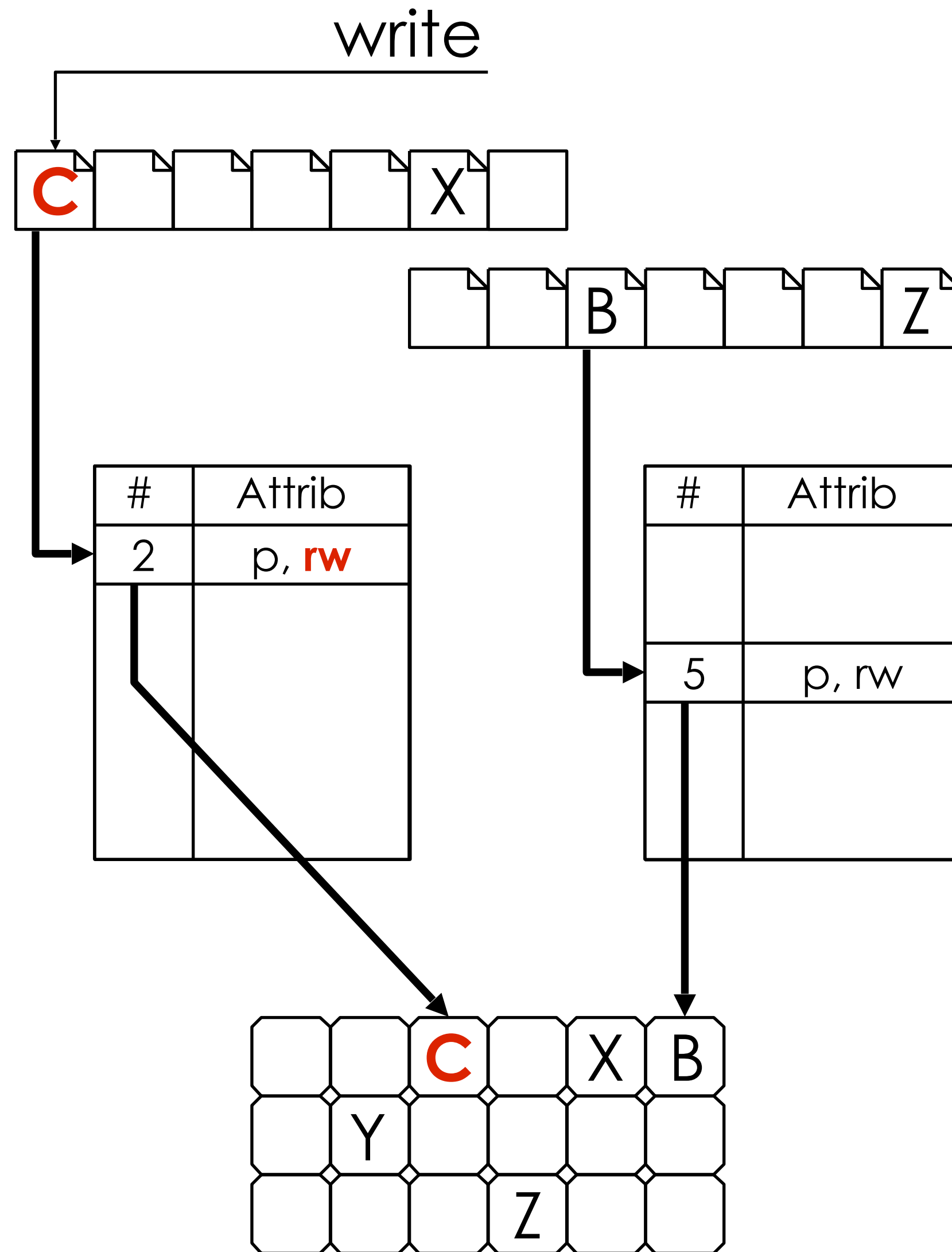
- schreibender Zugriff → Seitenfehler
- Das cow-Attribut bewirkt, dass eine neue Kachel allokiert und der Inhalt physisch kopiert wird
- neuen MMU-Eintrag an Zieladresse: rw
- danach erfolgt der Schreibzugriff

# Verzögertes Kopieren



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

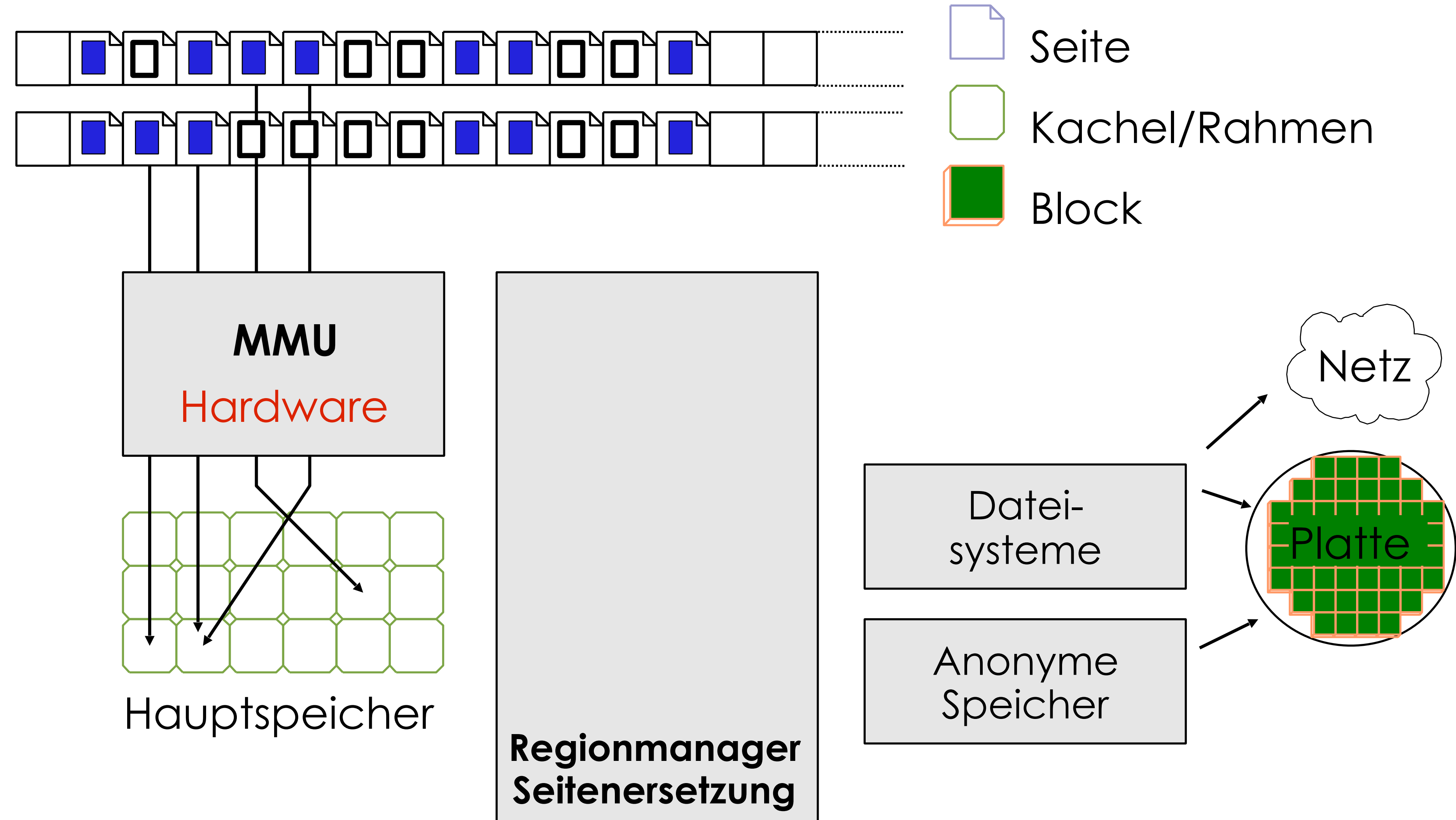
# Verzögertes Kopieren



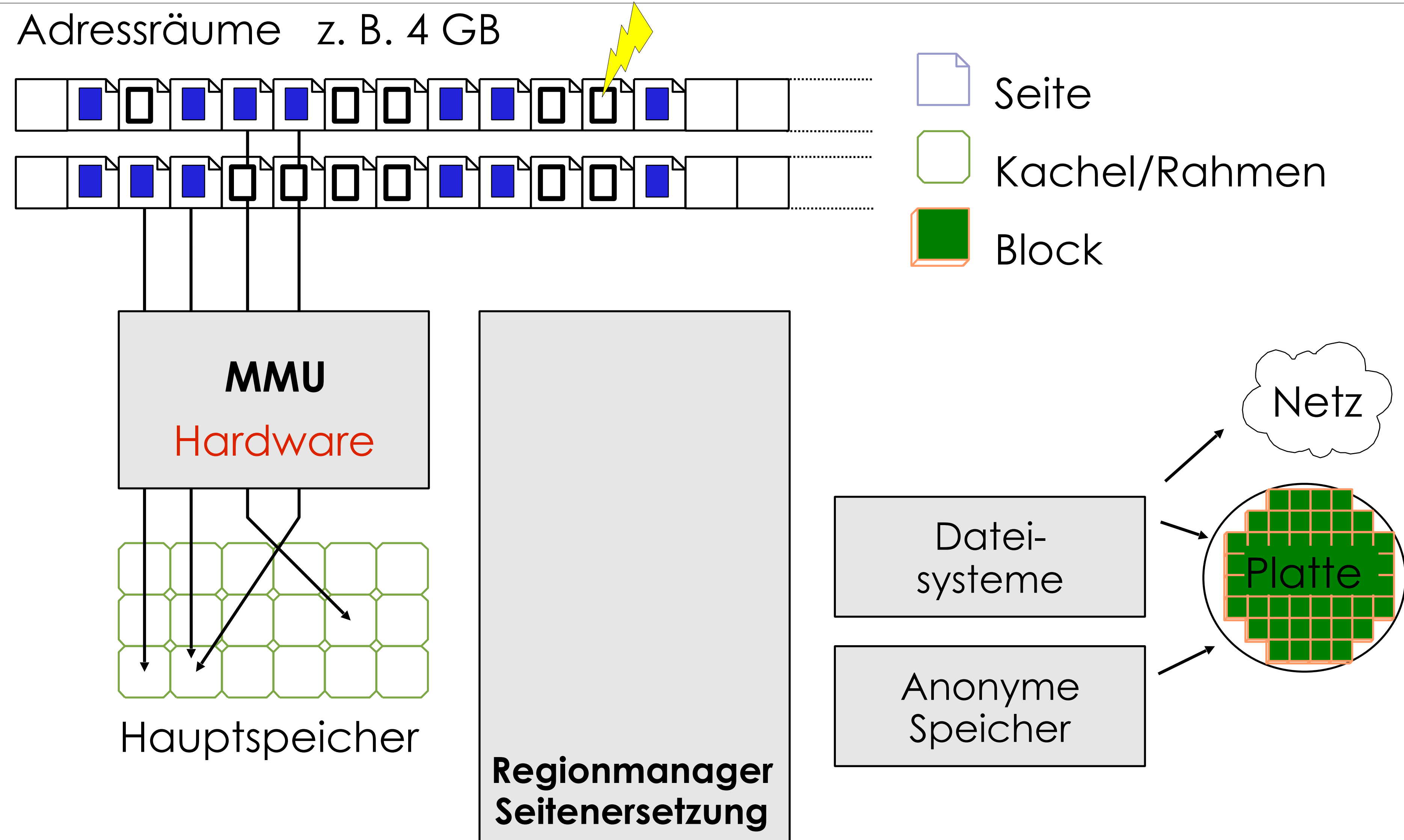
- schreibender Zugriff → Seitenfehler
- ist der Prozess der alleinige Besitzer der Kachel, wird lediglich das Schreibrecht gesetzt
- danach erfolgt der Schreibzugriff

# Das Zusammenspiel

Adressräume z. B. 4 GB

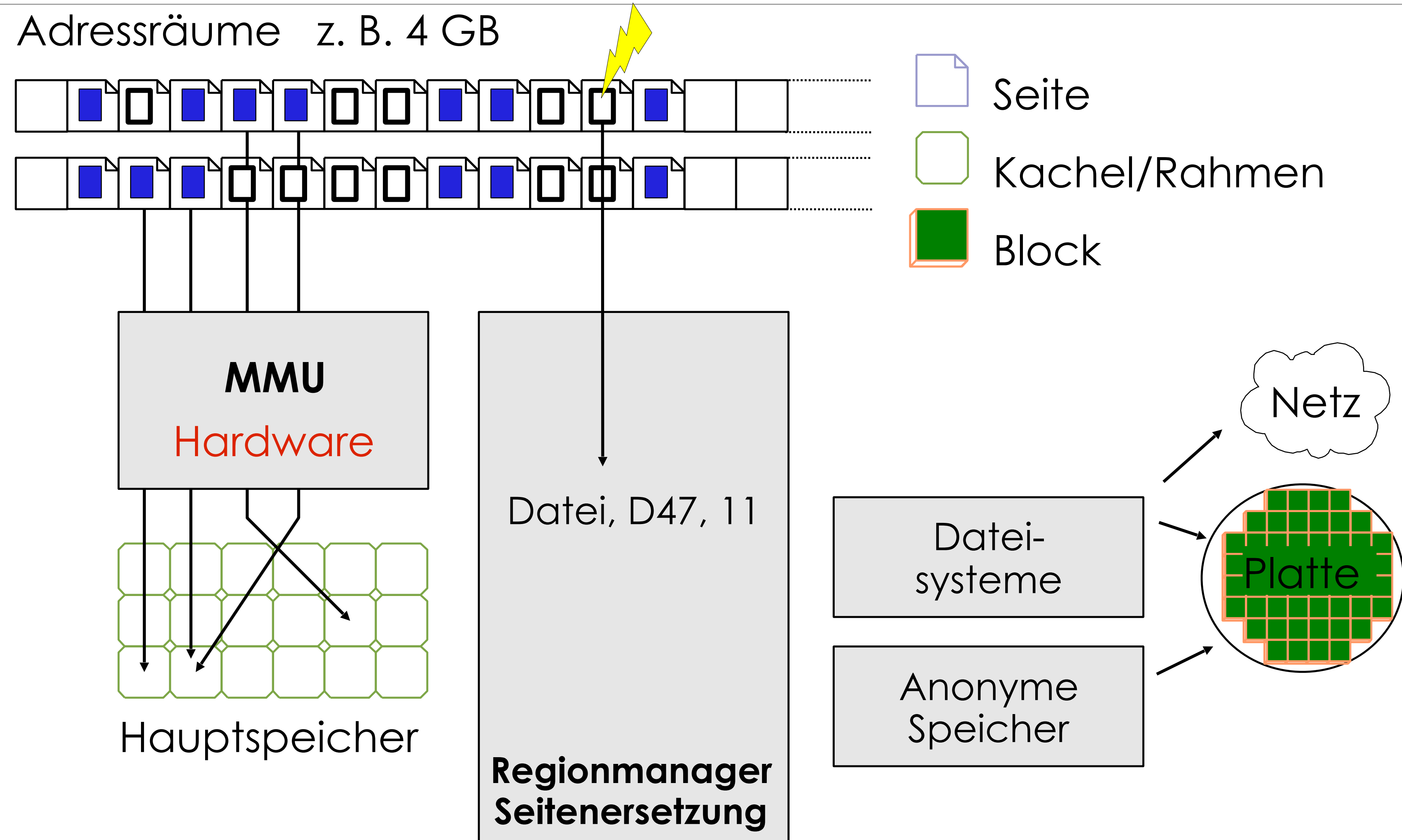


# Das Zusammenspiel

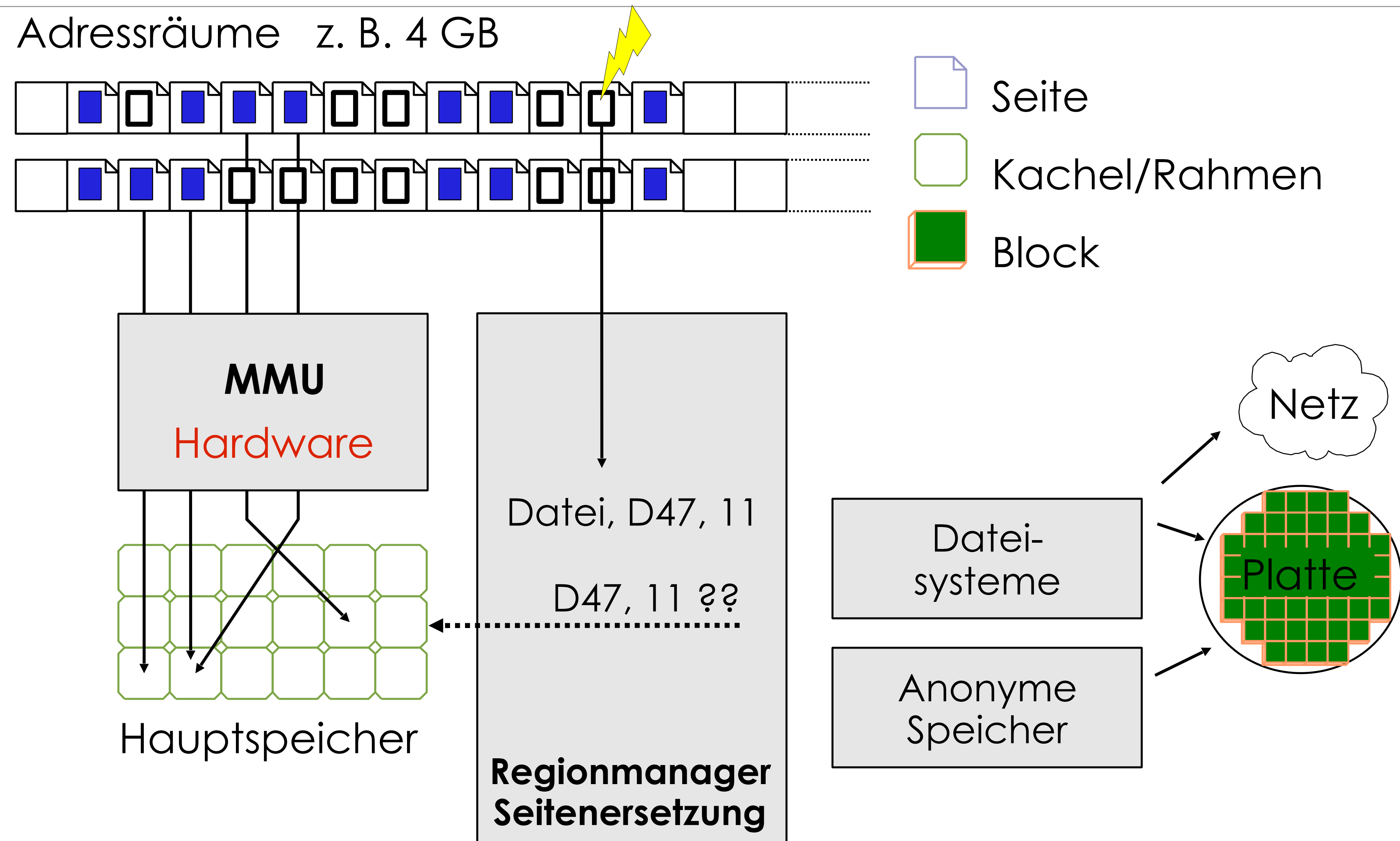




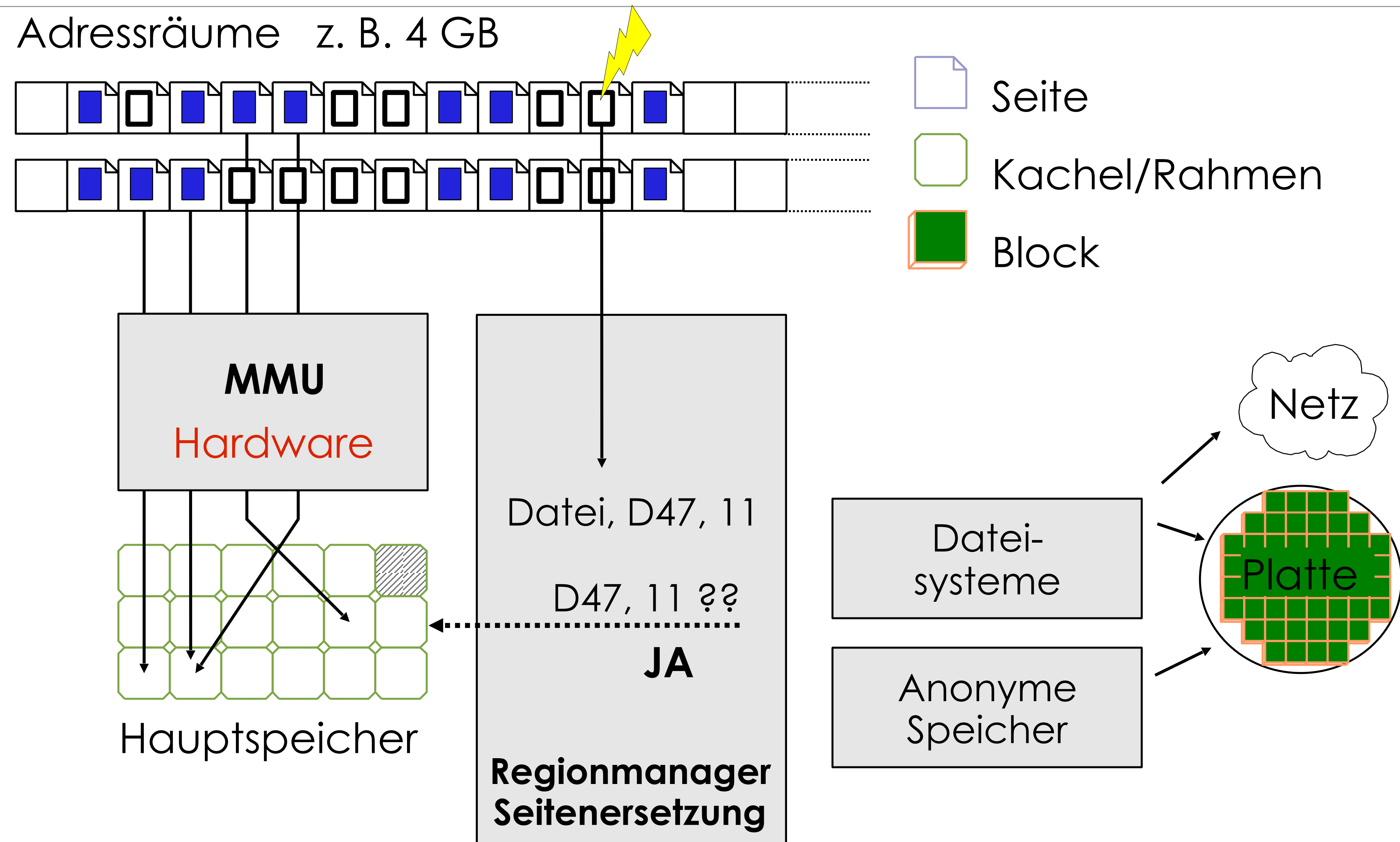
# Das Zusammenspiel



# Das Zusammenspiel

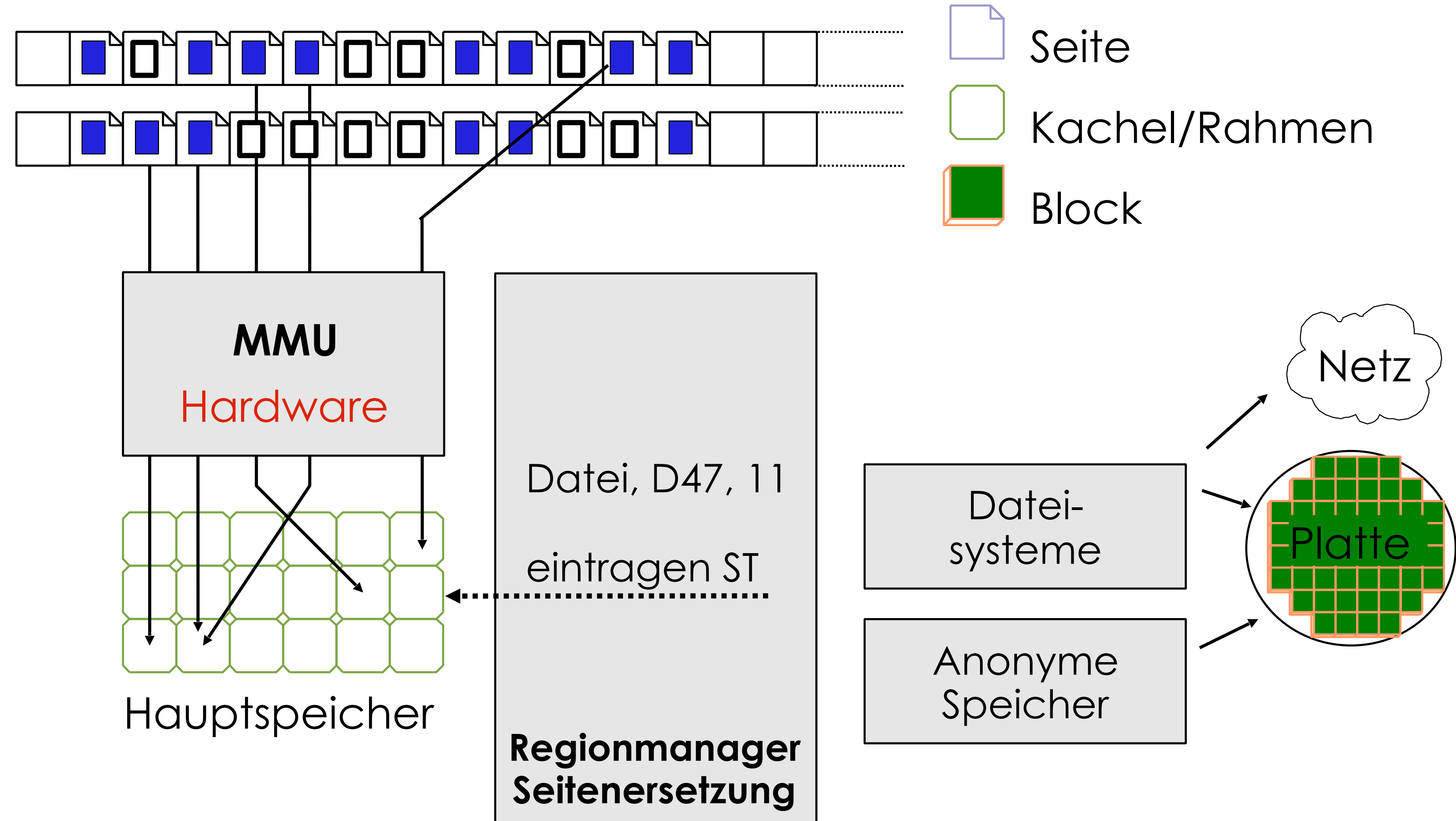


# Das Zusammenspiel

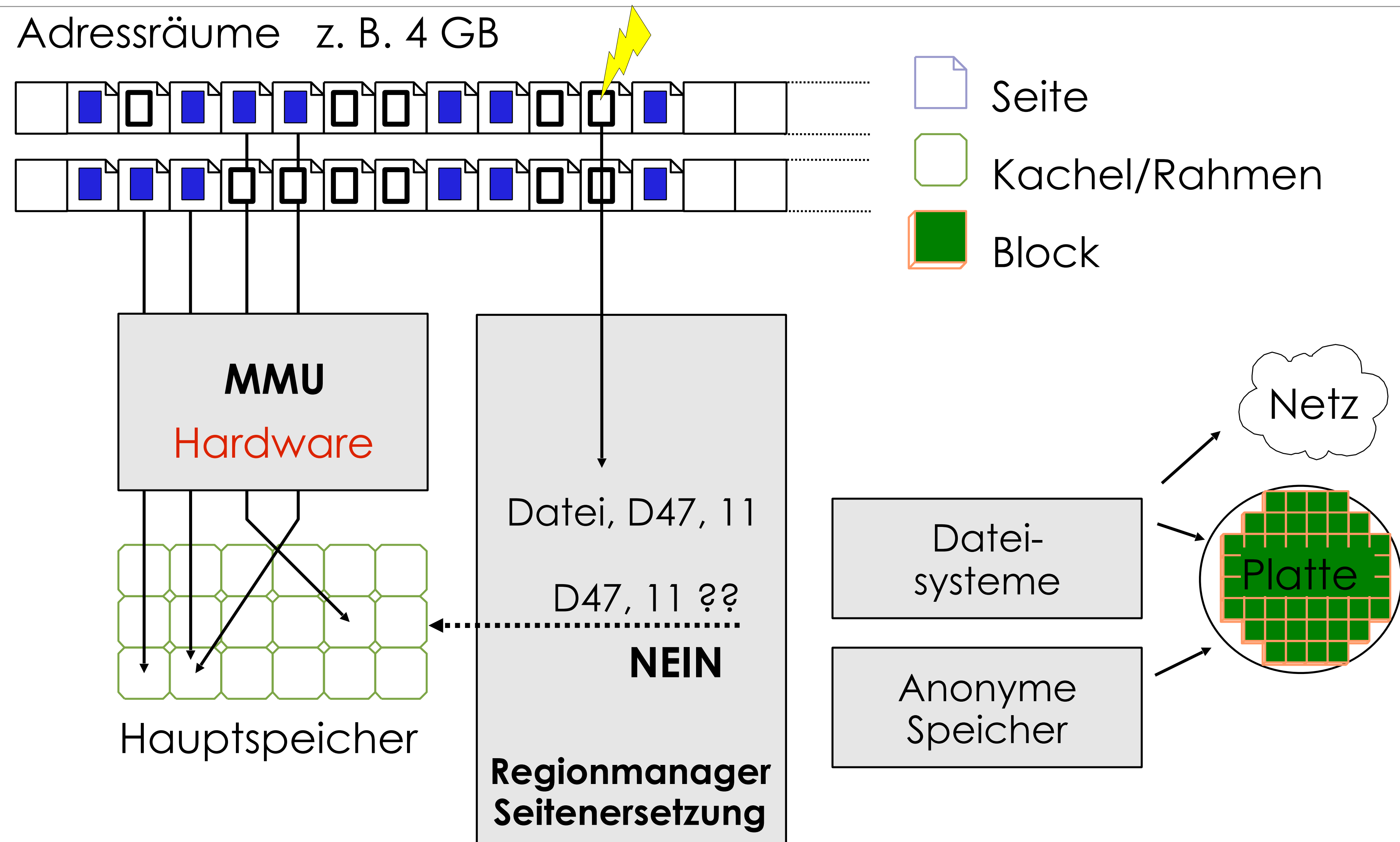


# Das Zusammenspiel

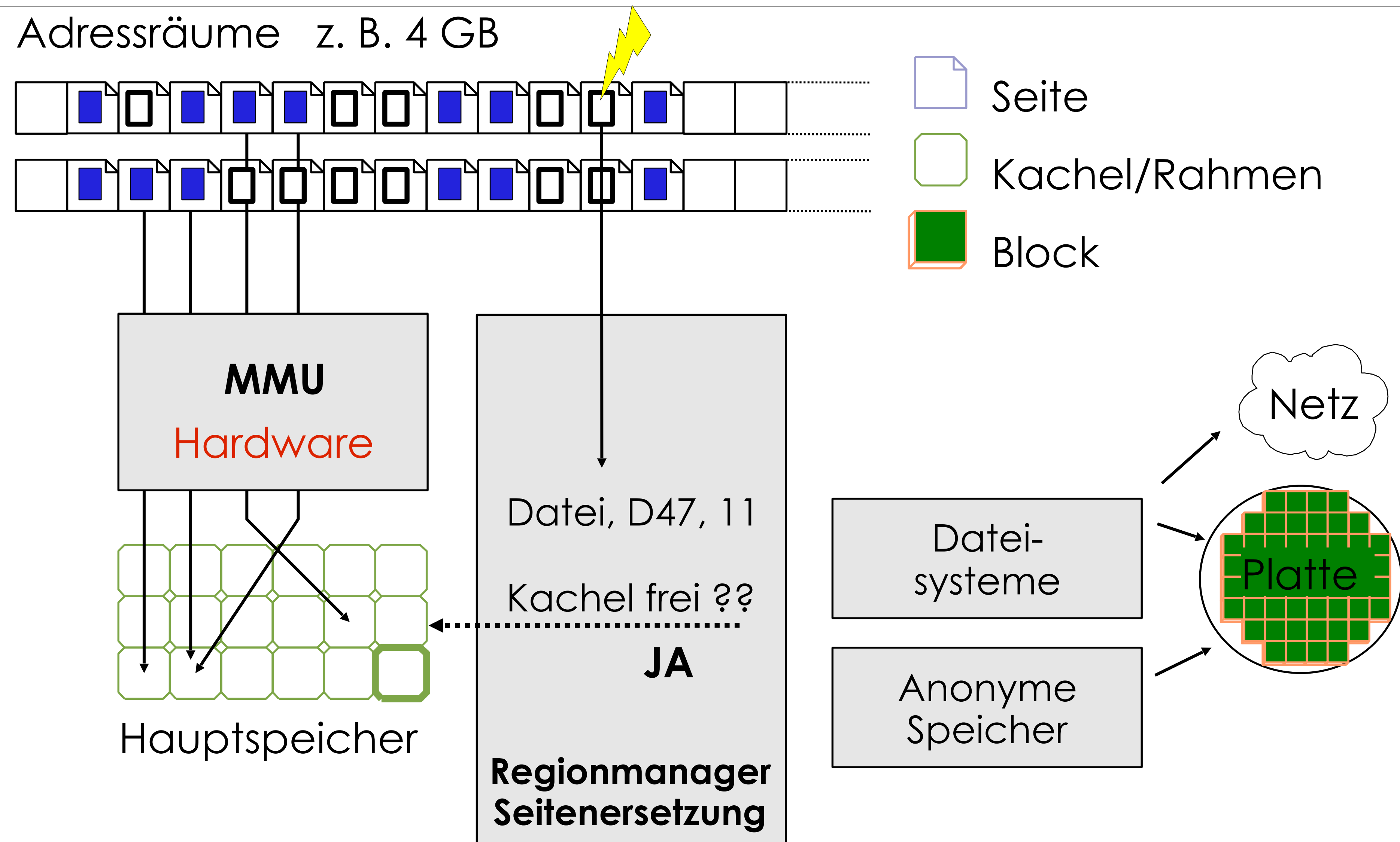
Adressräume z. B. 4 GB



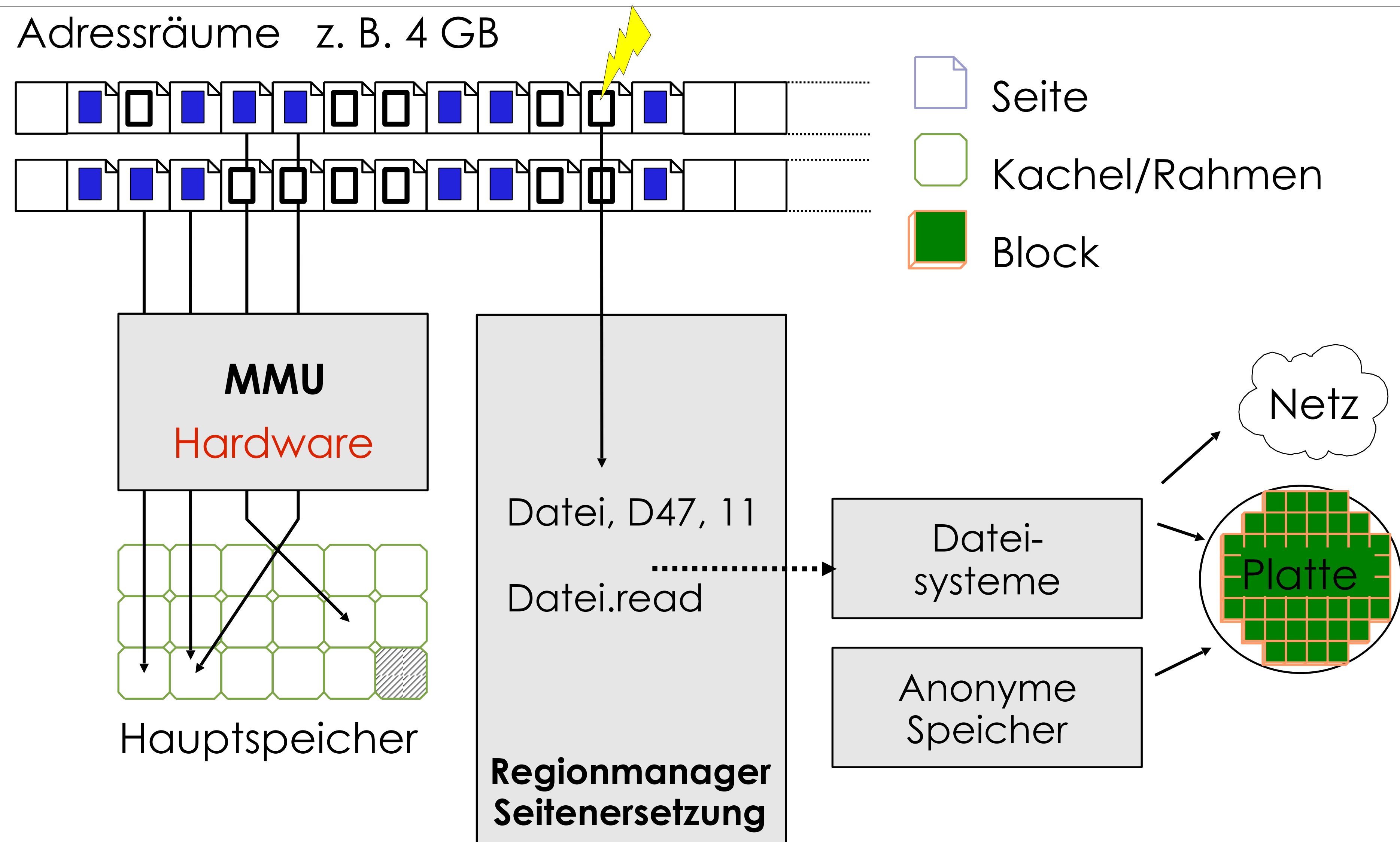
# Das Zusammenspiel



# Das Zusammenspiel

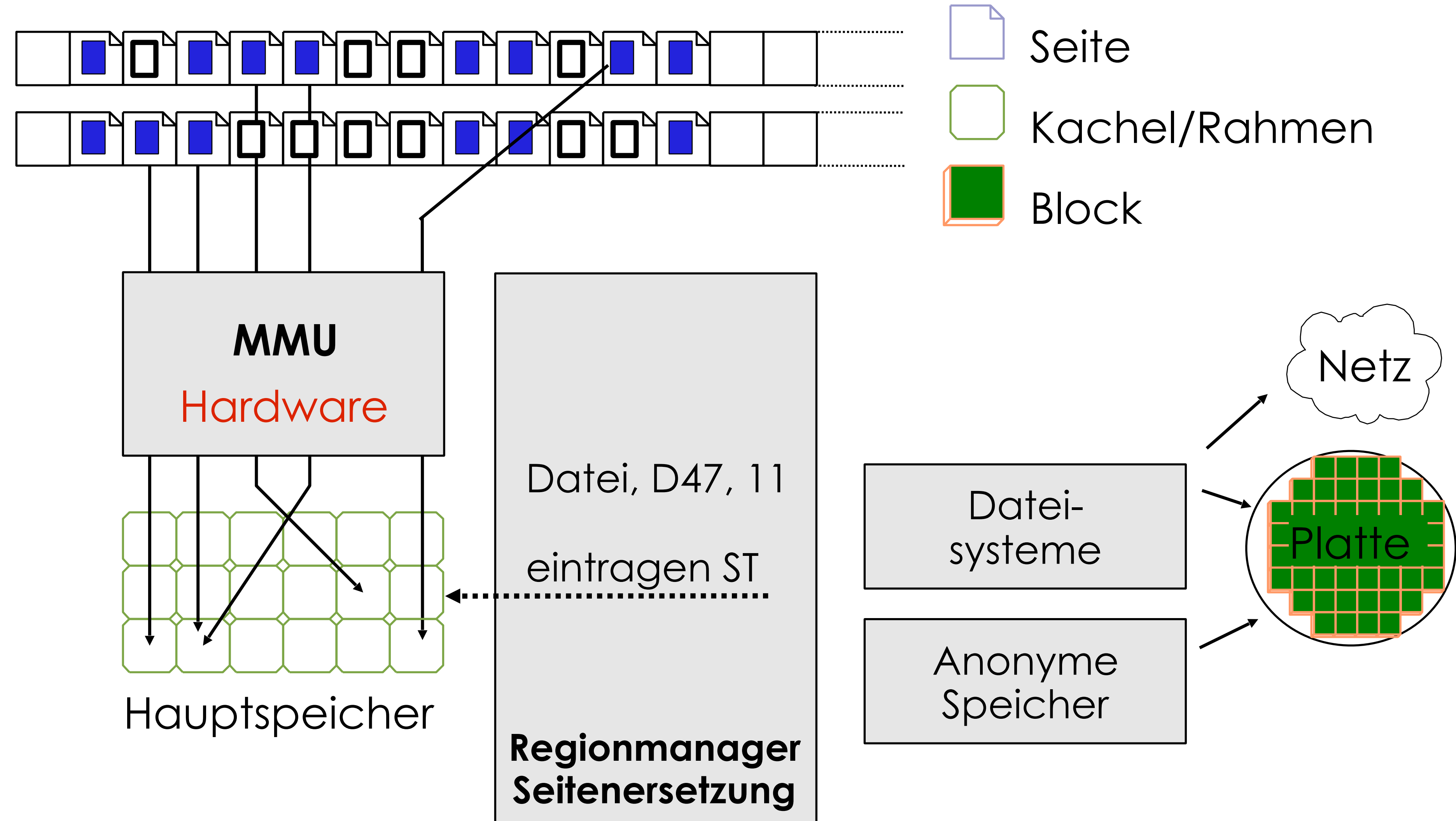


# Das Zusammenspiel



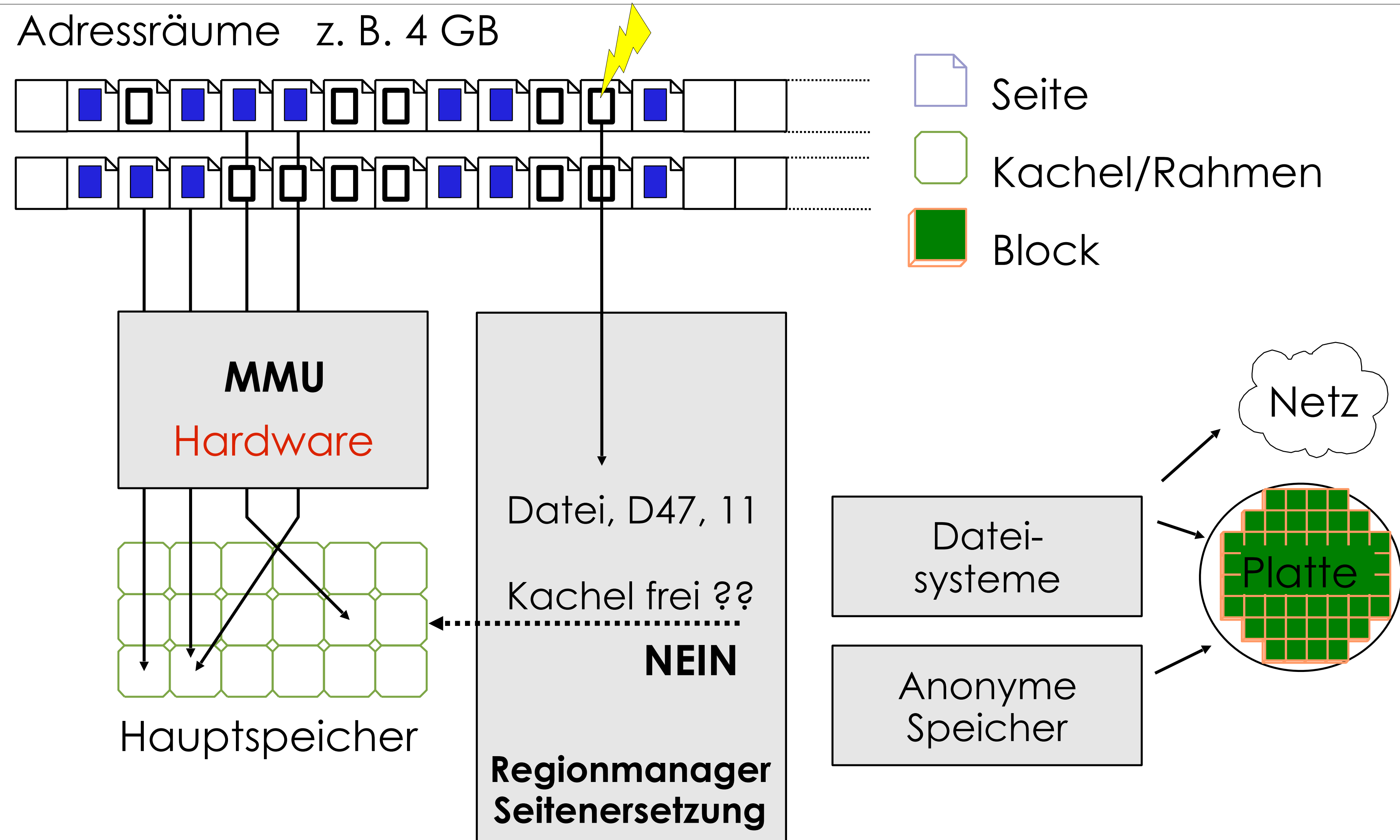
# Das Zusammenspiel

Adressräume z. B. 4 GB

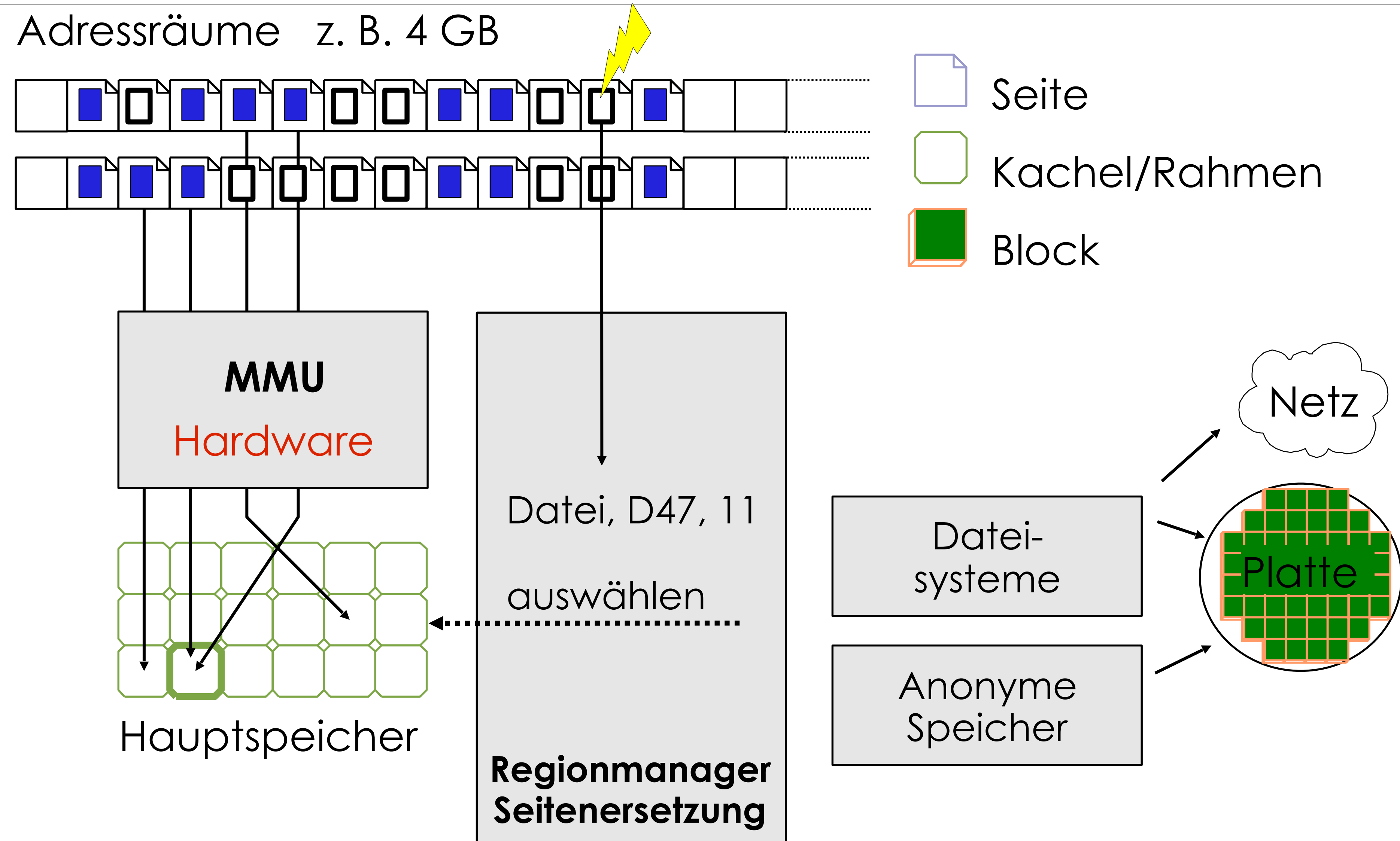




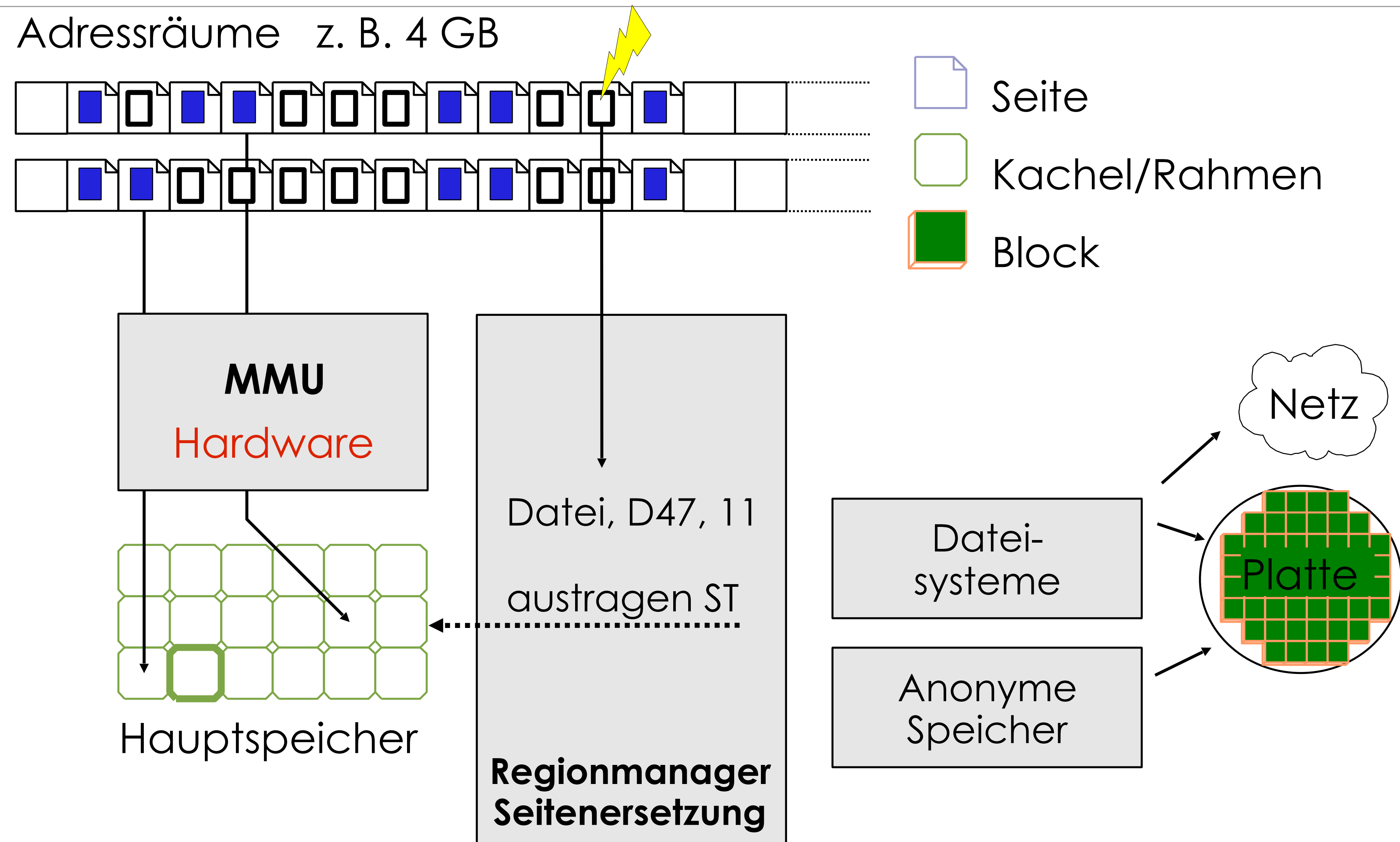
# Das Zusammenspiel



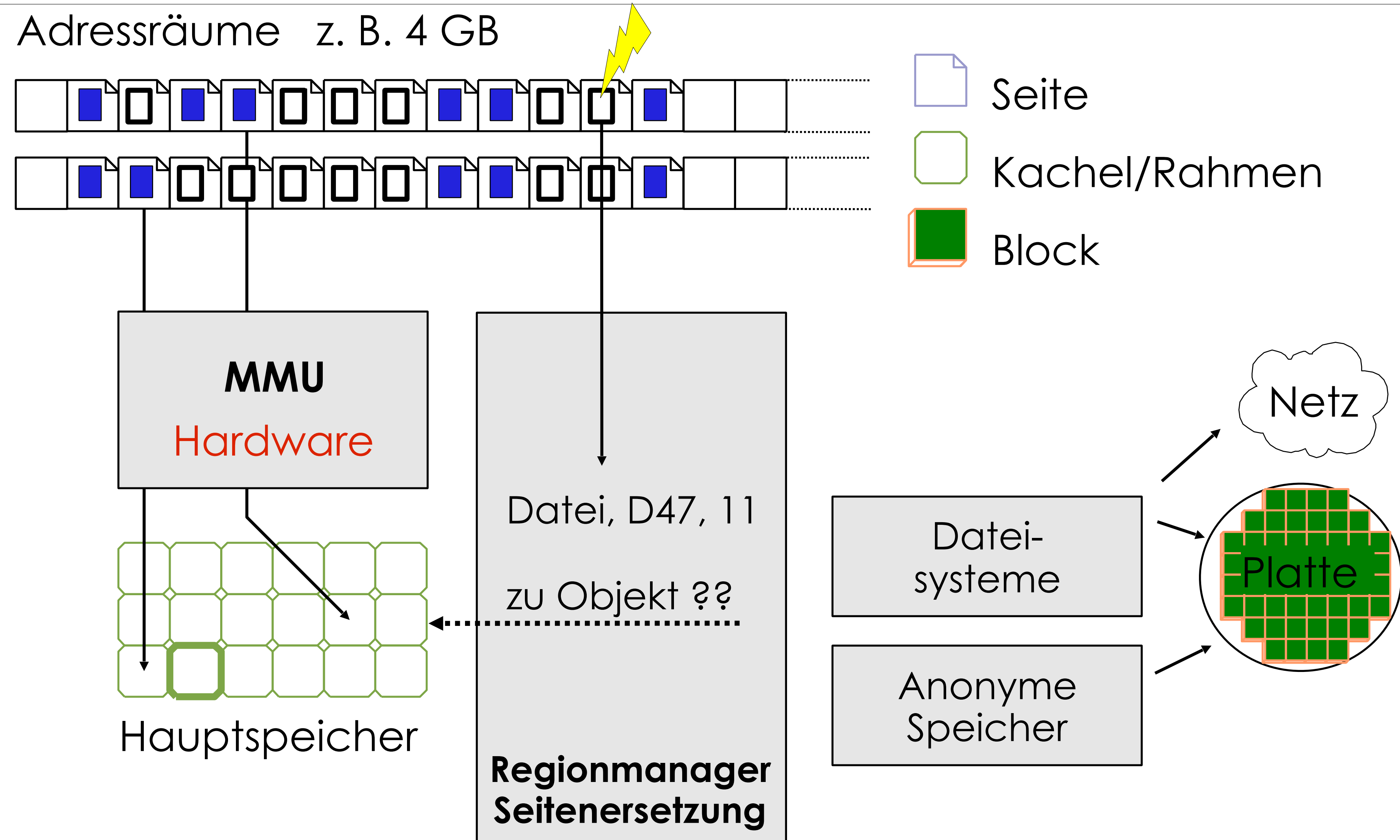
# Das Zusammenspiel



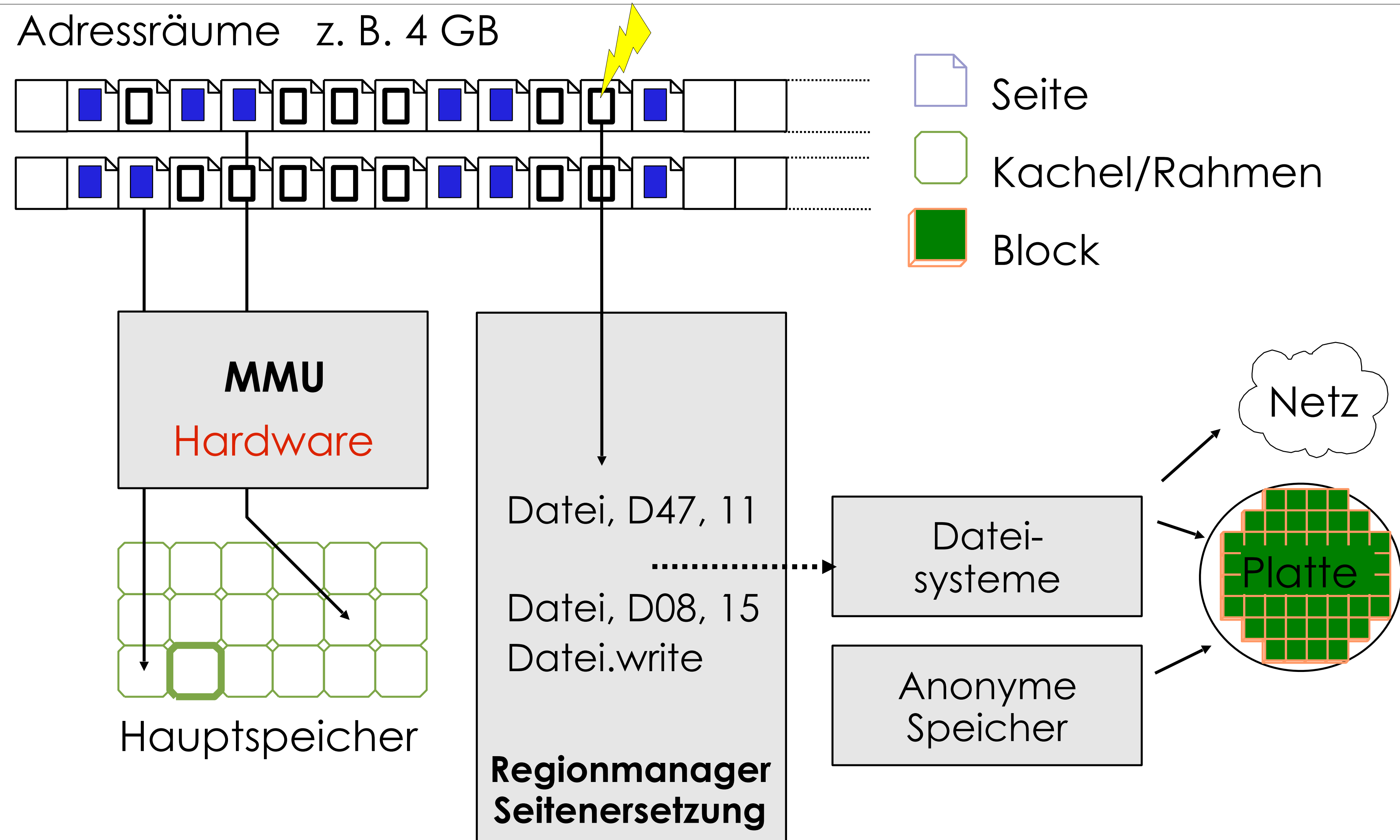
# Das Zusammenspiel



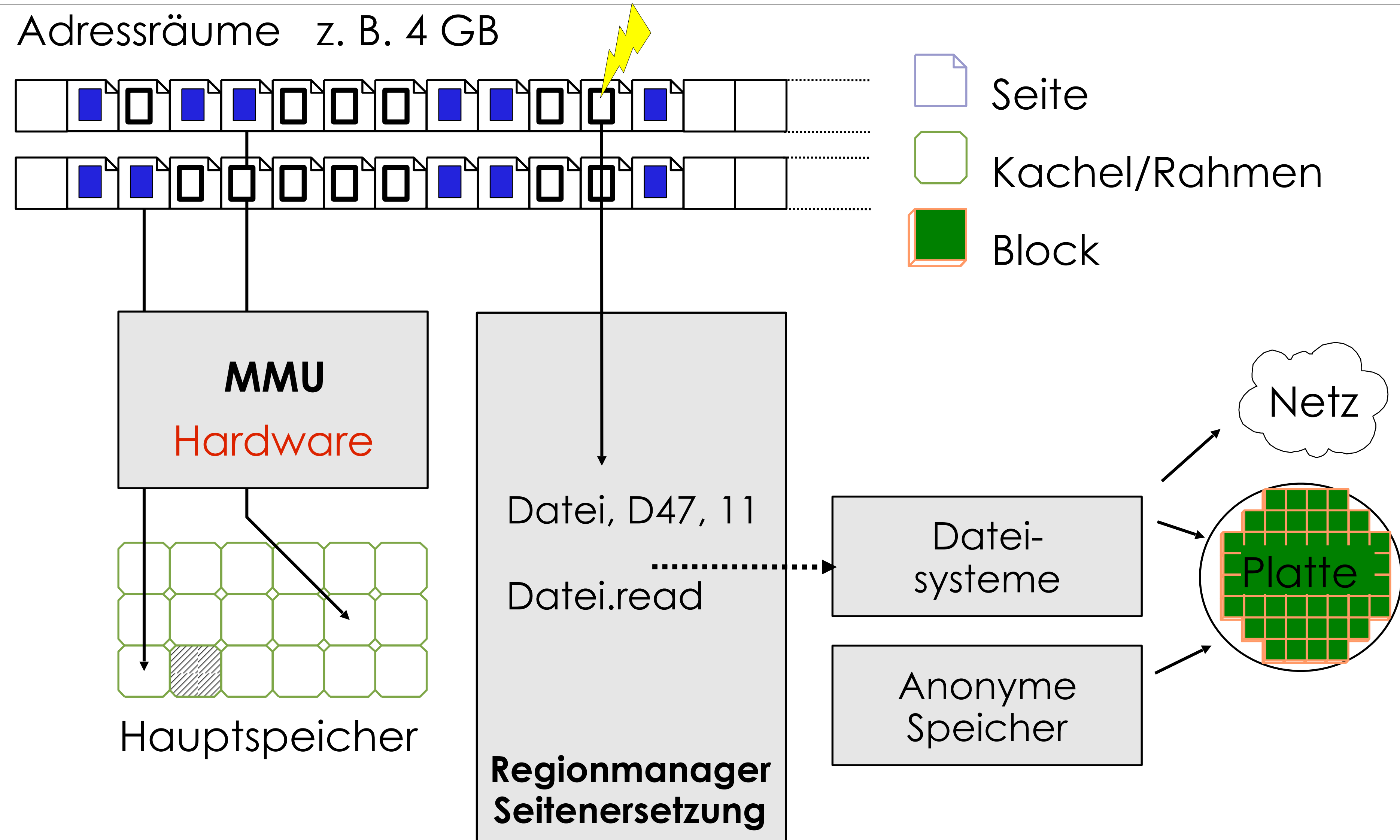
# Das Zusammenspiel



# Das Zusammenspiel

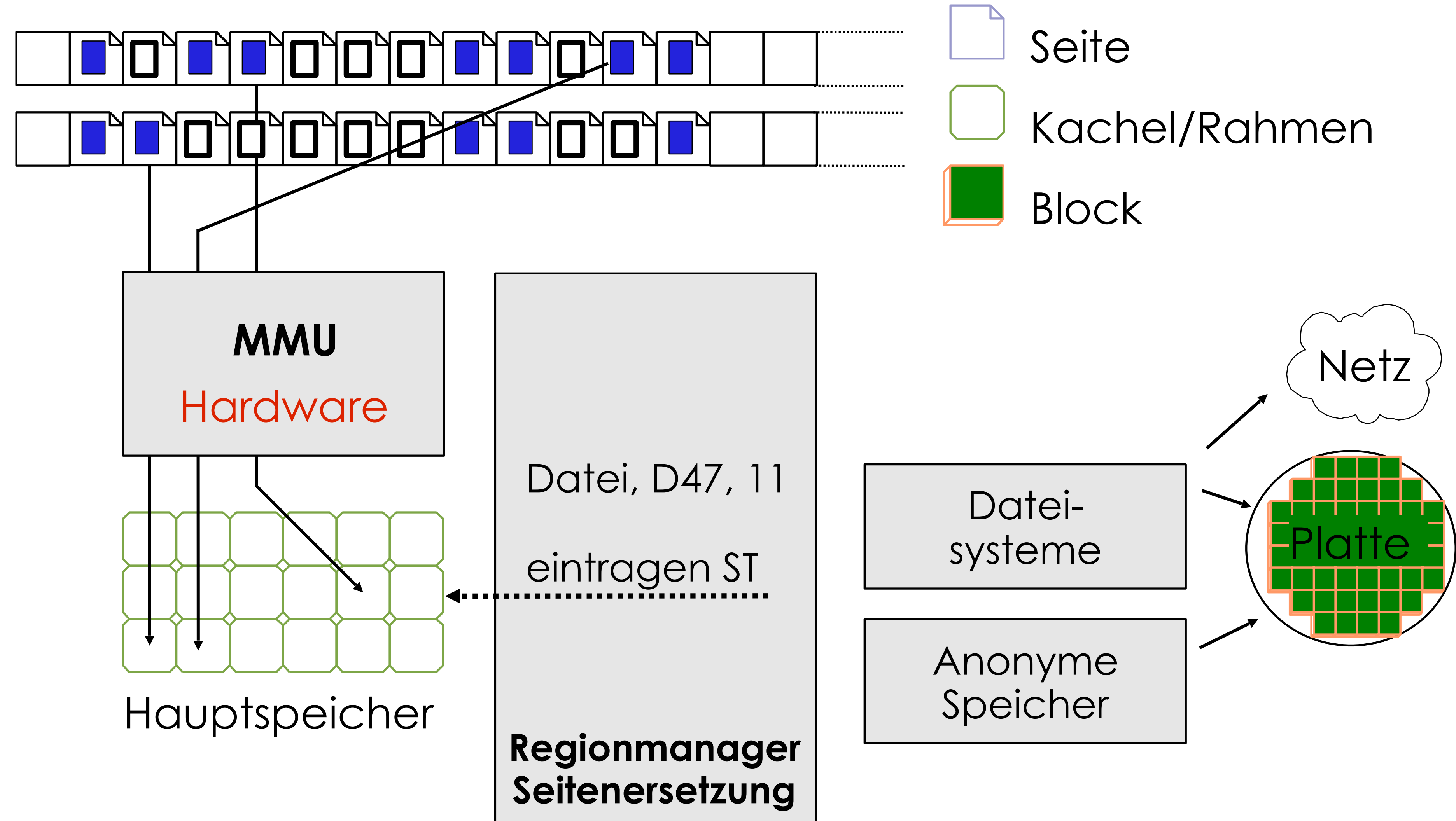


# Das Zusammenspiel



# Das Zusammenspiel

Adressräume z. B. 4 GB



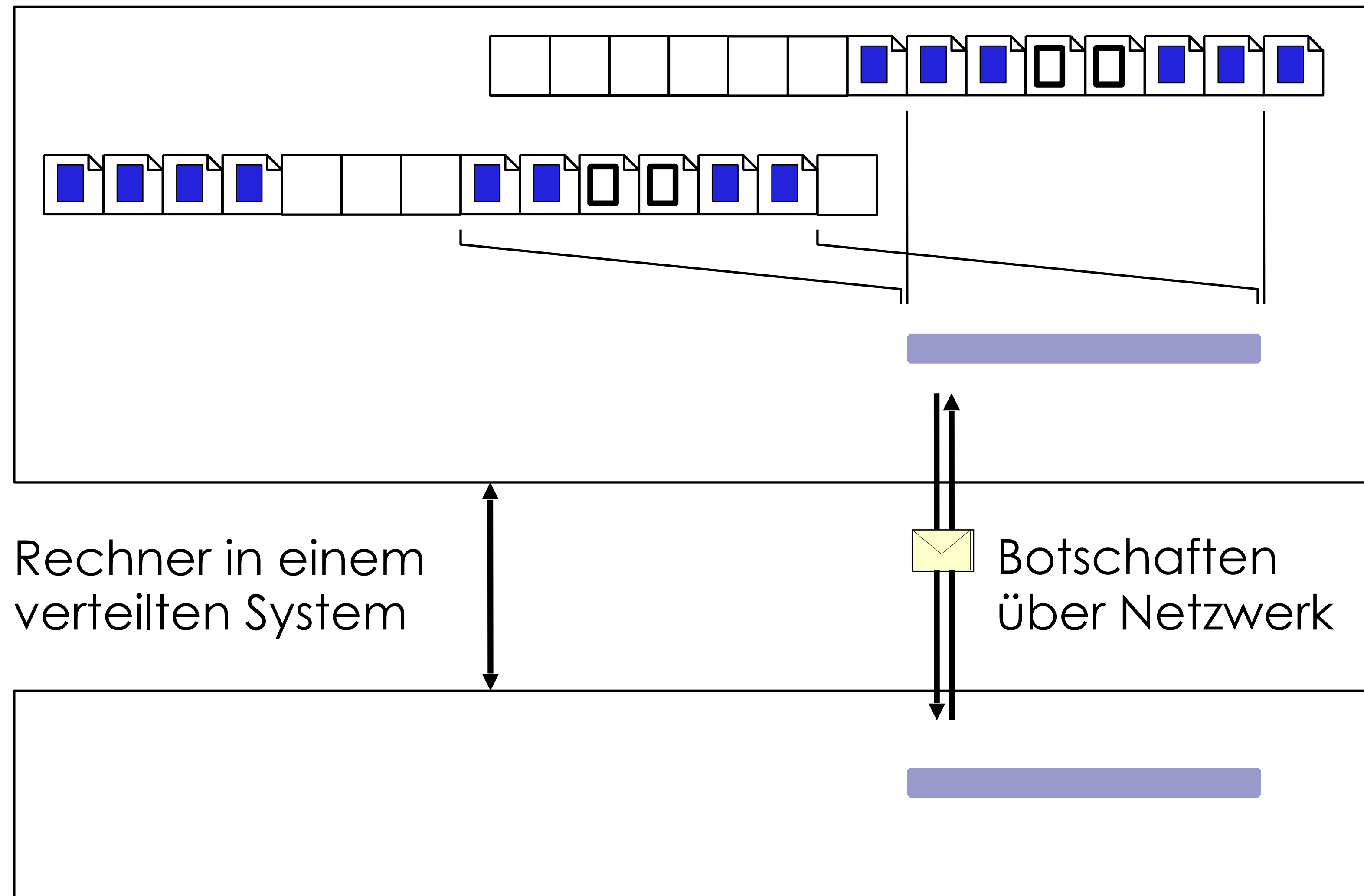
# Das Zusammenspiel

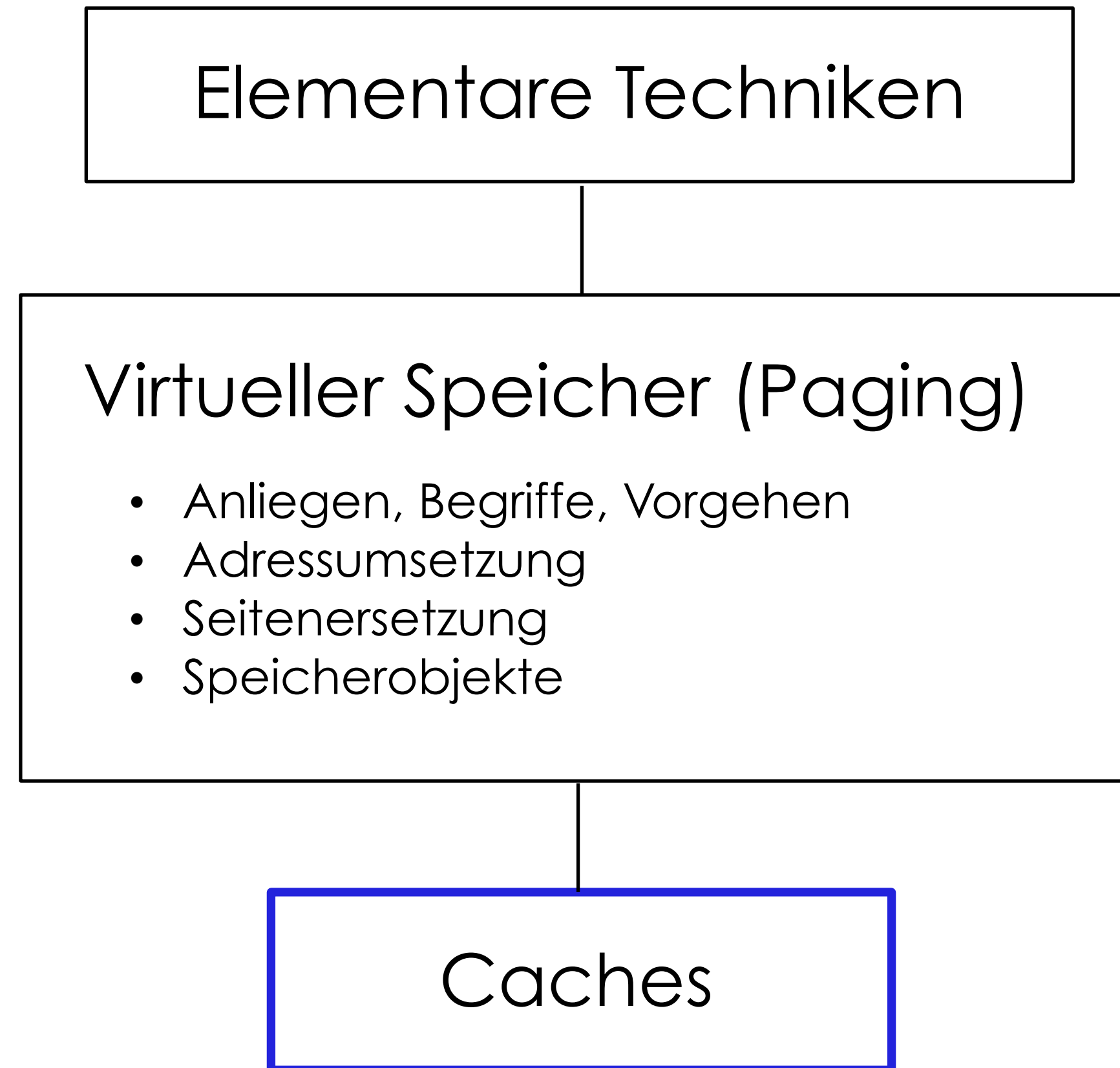
---

- Seitenfehler
- Finde Objekt + Seite (Offset-Behandlung)
- Suche ob Kachel schon vorhanden
- Falls ja: Seite in Seitentabelle eintragen → fertig
- Falls nein: freie Kachel vorhanden?
- Falls freie Kachel vorhanden: Inhalt lesen, Seite eintragen
- Falls keine freie Kachel vorhanden → verdrängen
- Aus allen beteiligten Seitentabellen austragen
- Zu welchem Objekt gehört verdrängte Kachel?
- Inhalt sichern, wenn er verändert wurde
- dann wie oben: neuen Inhalt lesen, Seite eintragen



# Speicherobjekte im Netz





# Rechnerarchitektur: Caches

---

## **Ziel**

größtmöglicher und schnellstmöglicher Speicher

## **Grundlage**

Lokalitätsverhalten bei Speicherzugriffen

## **Idee**

- Anordnung von schnelleren, kleineren Speichern vor größeren, langsameren Speichern
- bei jedem Zugriff auf langsameren Speicher wird auch in schnellerem Speicher eine Kopie angelegt und genutzt

# Caches für Hauptspeicher



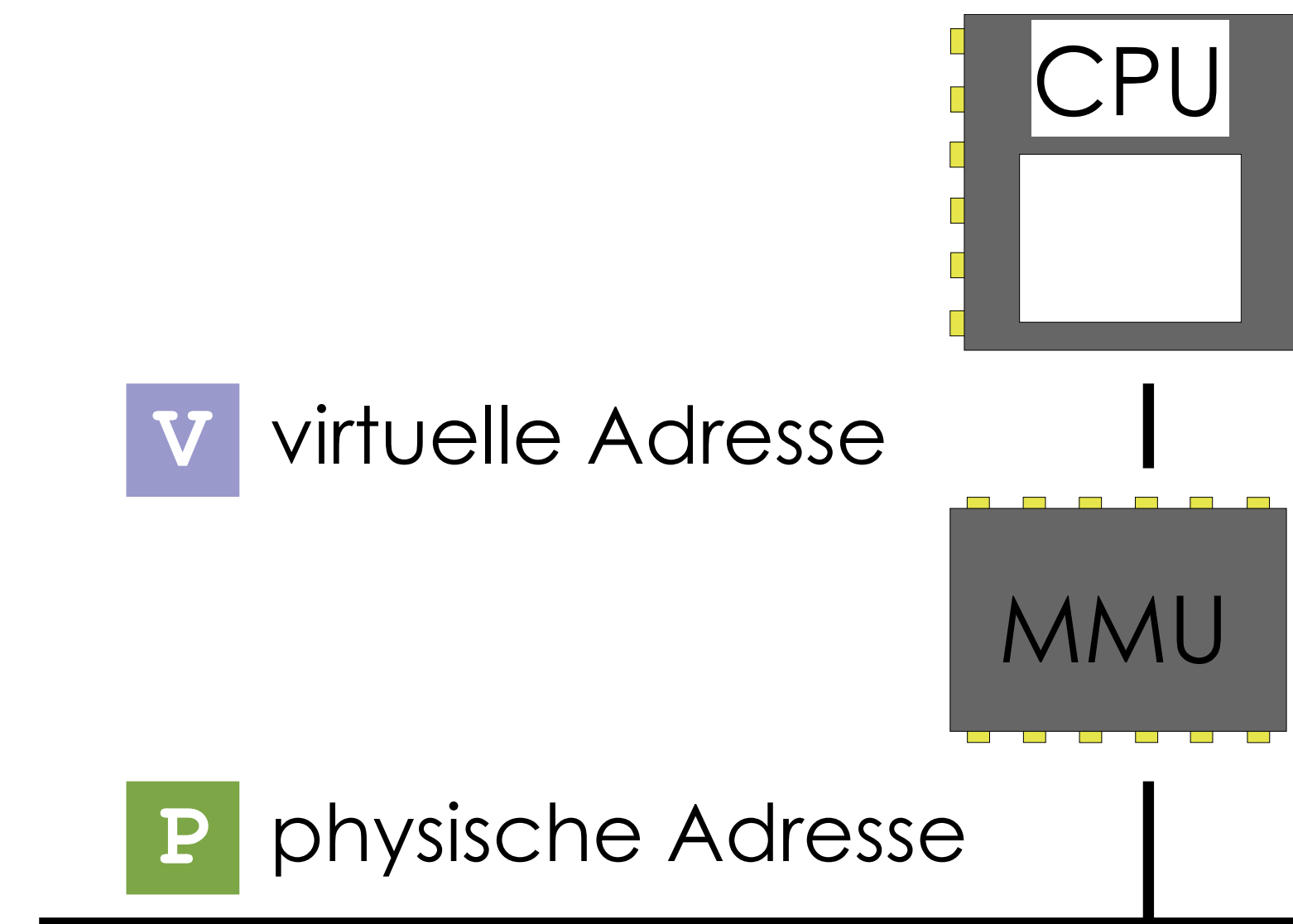
## Charakteristika

- Gesamtgröße, Cacheline-Größe, Geschwindigkeit
- Organisation: Austauschverfahren, Zugriffsart
- Index: aus Adresse wird Position im Cache berechnet
- Tag: im Cache gespeicherte Adresse

# Organisation von Caches

## Virtuelle vs. physische Adressen

- virtuelle Tags und Indizes (VIVT)
  - schnell (spart MMU-Zugriff)
  - Löschen bei AR-Wechsel
- physische Tags und Indizes (PIPT)
  - langsamer, da auf MMU gewartet werden muss
  - unabhängig von Adressraum-Organisation (Sharing)
- virtuelle Indizes, physische Tags (VIPT)
  - Kompromiss
  - Index-Zugriff parallel zur MMU, Tag-Vergleich physisch



# Zusammenfassung Speicher

---

- Ziele von virtuellem Speicher: Schutz, Abstraktion, Effizienz
- Adressumsetzung in der MMU
- mehrstufige Seitentabellen, TLB
- Behandlung von Seitenfehlern
- Seitenersetzung zur Verwaltung des Hauptspeichers
- logischer Aufbau des Adressraums: Regionen