

MMU-Ansteuerung: einige wichtige Details

Aufgaben

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

Kontext: Verwendung der Attribute

BS-seitig

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

BS-seitig (später mehr Kontext)

```
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 alten AddrRäumen entfernen  
    }  
}
```

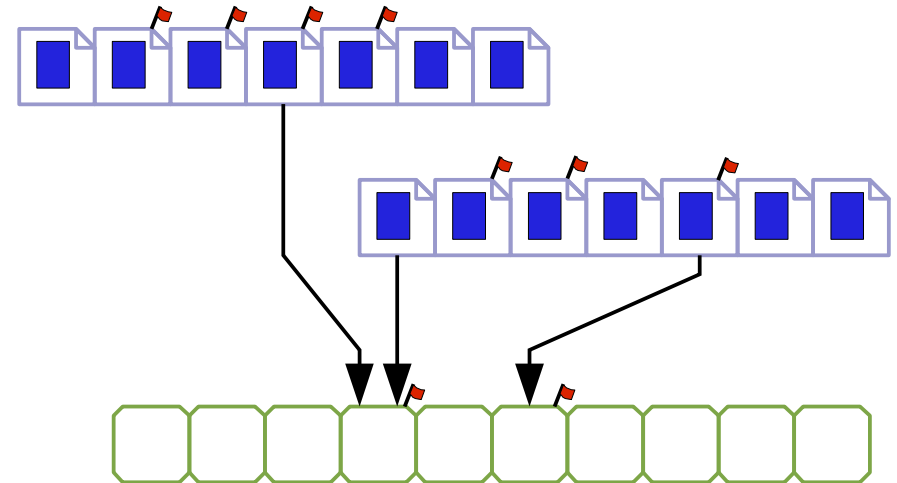
Hardwareseitig

```
... Prozesse arbeiten ...  
→ HW setzt referenced- und  
modified-Flag  
in Seitentabllen !!!
```

basieren auf „referenced“
und „modified“ Attributen
von Kacheln !!!

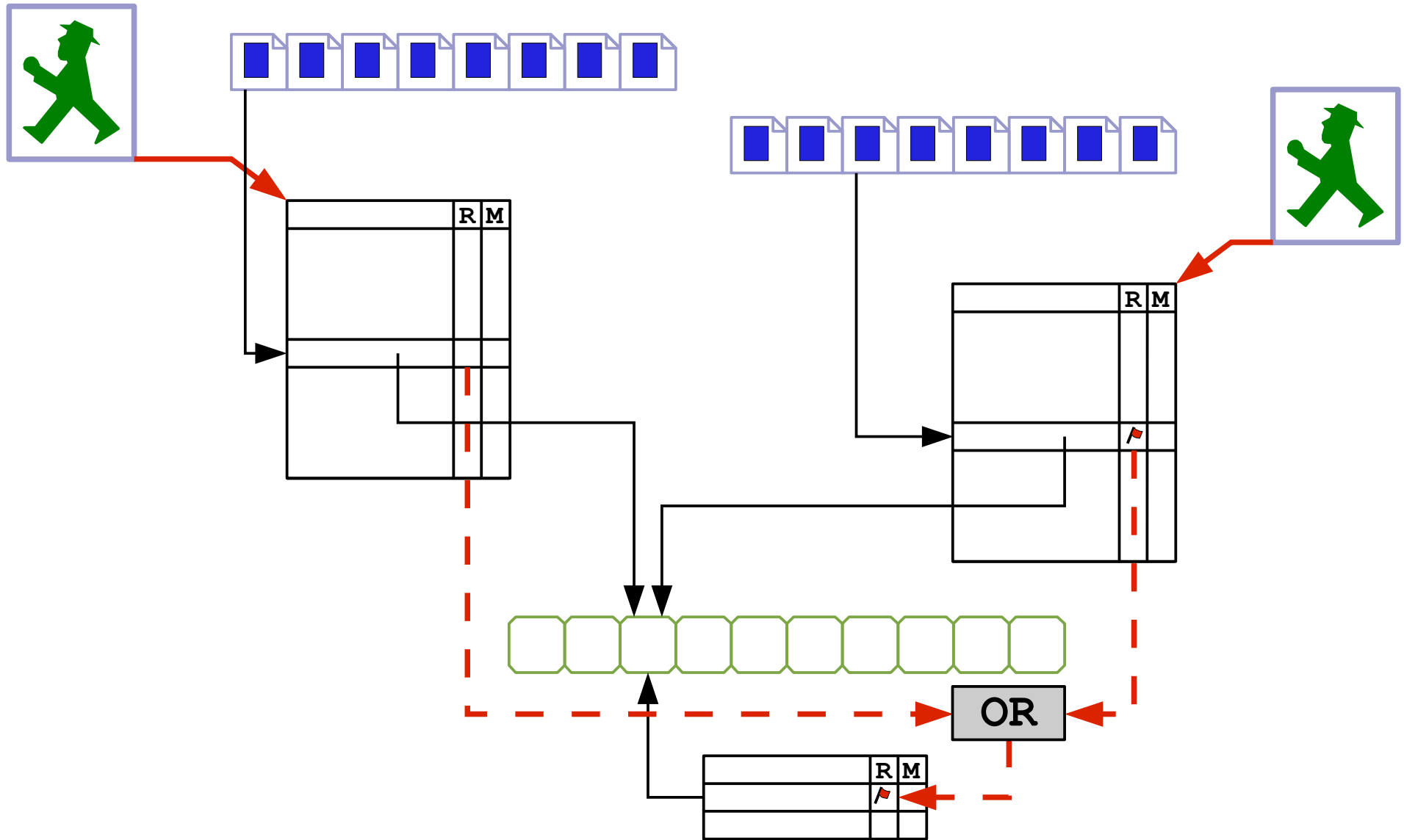
Propagation von Attributen

- 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
 - ♦ Später:
Identifizierung des Speicherobjektes, damit Inhalt weggeschafft werden kann

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

BS-seitig

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

Hardwareseitig

```
... Prozesse 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 alten AddrRäumen 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

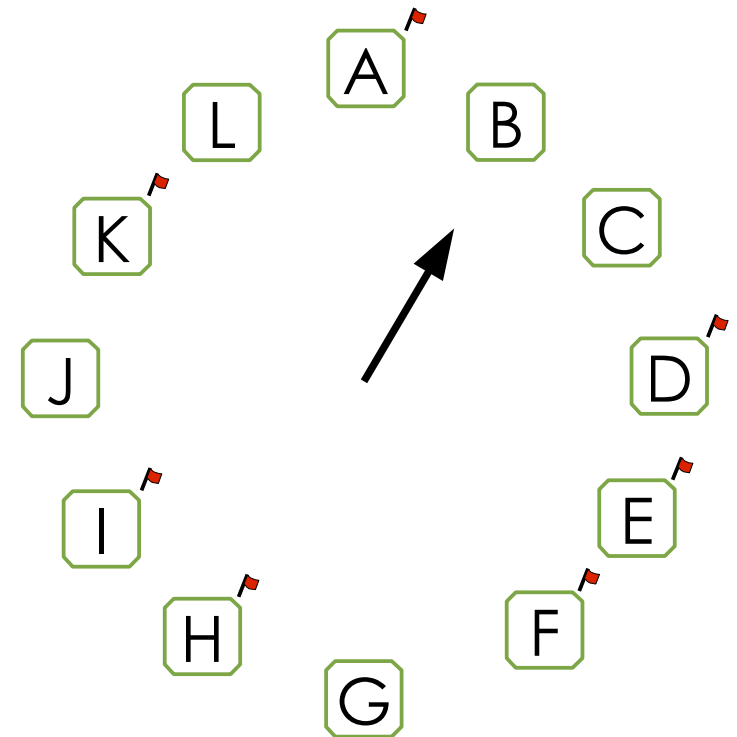
- gilt als gute Näherung für optimalen Algorithmus

Nachteil

- aufwendige Realisierung
jeder Speicherzugriff müsste berücksichtigt werden
 - HW-Unterstützung wird benötigt
- Annäherungen per Software

LRU Annäherung: Clock-Algorithmus

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



LRU Annäherung: Aging

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

niedrige Zahl → hohes Alter

LRU Annäherung: Aging

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

Hardware

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

BS-Thread:

```
// at every „Tick“ call:  
  
void Aging(){  
    for(i = 0; i < maxK; i++) {  
        SetAge(Kachel[i].referenced);  
        // nächste Folien  
        Kachel[i].referenced = 0;  
    }  
}
```

LRU Annäherung: Aging



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

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

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

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

LRU Annäherung: Aging



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



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



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



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

LRU Annäherung: 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
---	---	---	---	---

LRU Annäherung: Aging



A

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

B

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

C

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

D

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

LRU Annäherung: Aging



A

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

B

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

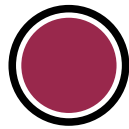
C

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

D

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

LRU Annäherung: 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
---	---	---	---	---

LRU Annäherung: 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
---	---	---	---	---

LRU Annäherung: Aging

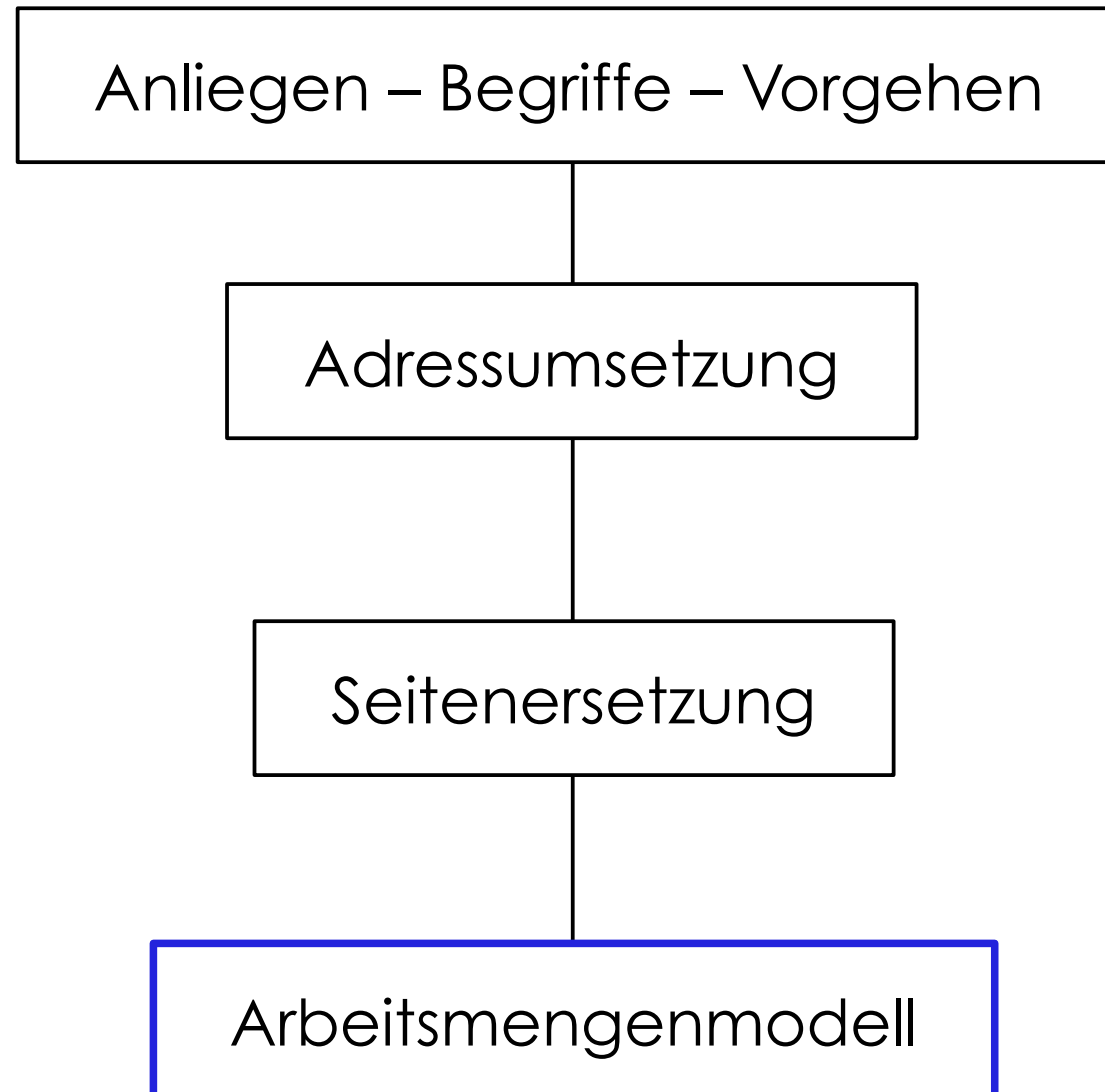
1 	Seitenersetzung	Age					
 A	<table border="1"><tr><td>1</td><td>0</td><td>1</td><td>0</td><td>0</td></tr></table>	1	0	1	0	0	20
1	0	1	0	0			
 B	<table border="1"><tr><td>0</td><td>1</td><td>0</td><td>0</td><td>0</td></tr></table>	0	1	0	0	0	8
0	1	0	0	0			
 C	<table border="1"><tr><td>1</td><td>1</td><td>0</td><td>0</td><td>0</td></tr></table>	1	1	0	0	0	24
1	1	0	0	0			
 D	<table border="1"><tr><td>0</td><td>1</td><td>1</td><td>0</td><td>0</td></tr></table>	0	1	1	0	0	12
0	1	1	0	0			

LRU Annäherung: Aging

1 

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

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

Das Arbeitsmengenmodell (Working Set)

Vorgehen

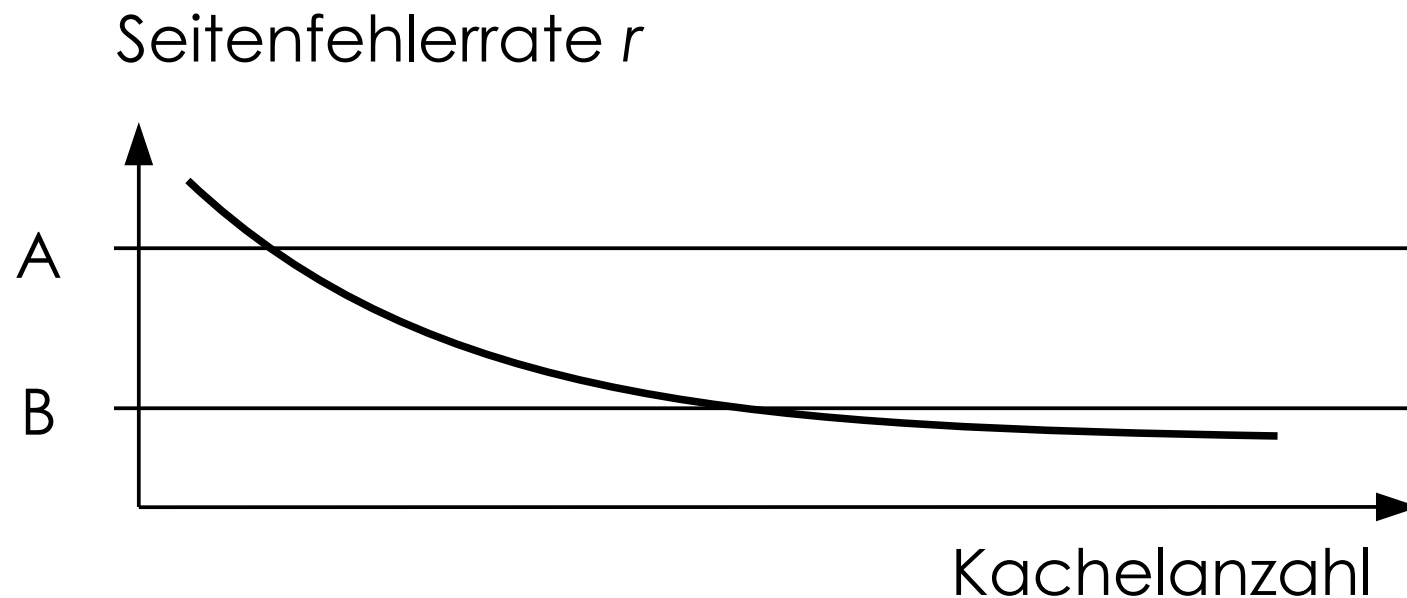
- Bestimmung der Arbeitsmenge bei jeder Seitenreferenz
- für aktive und bereite Prozesse muss sich genau die Arbeitsmenge im Speicher befinden
- Swapping, falls Rahmen fehlen
→ Prepaging bei Wiedereinlagerung

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

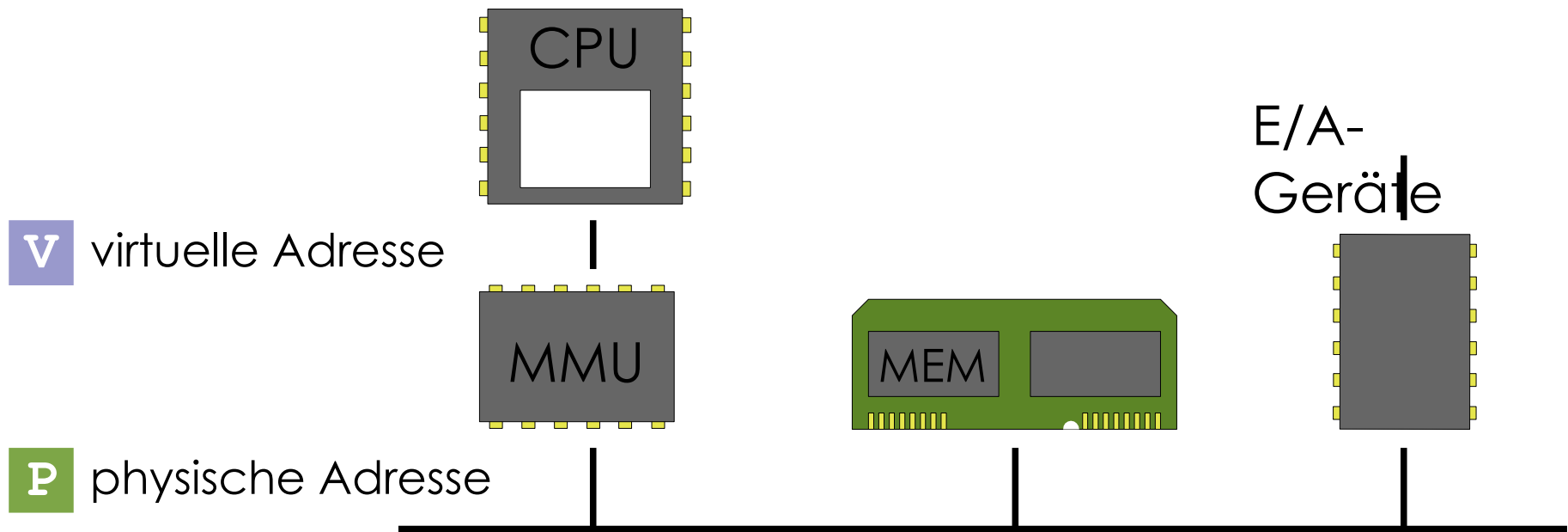
Weiterer Gesichtspunkt: Wahl der Seitengröße

- 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

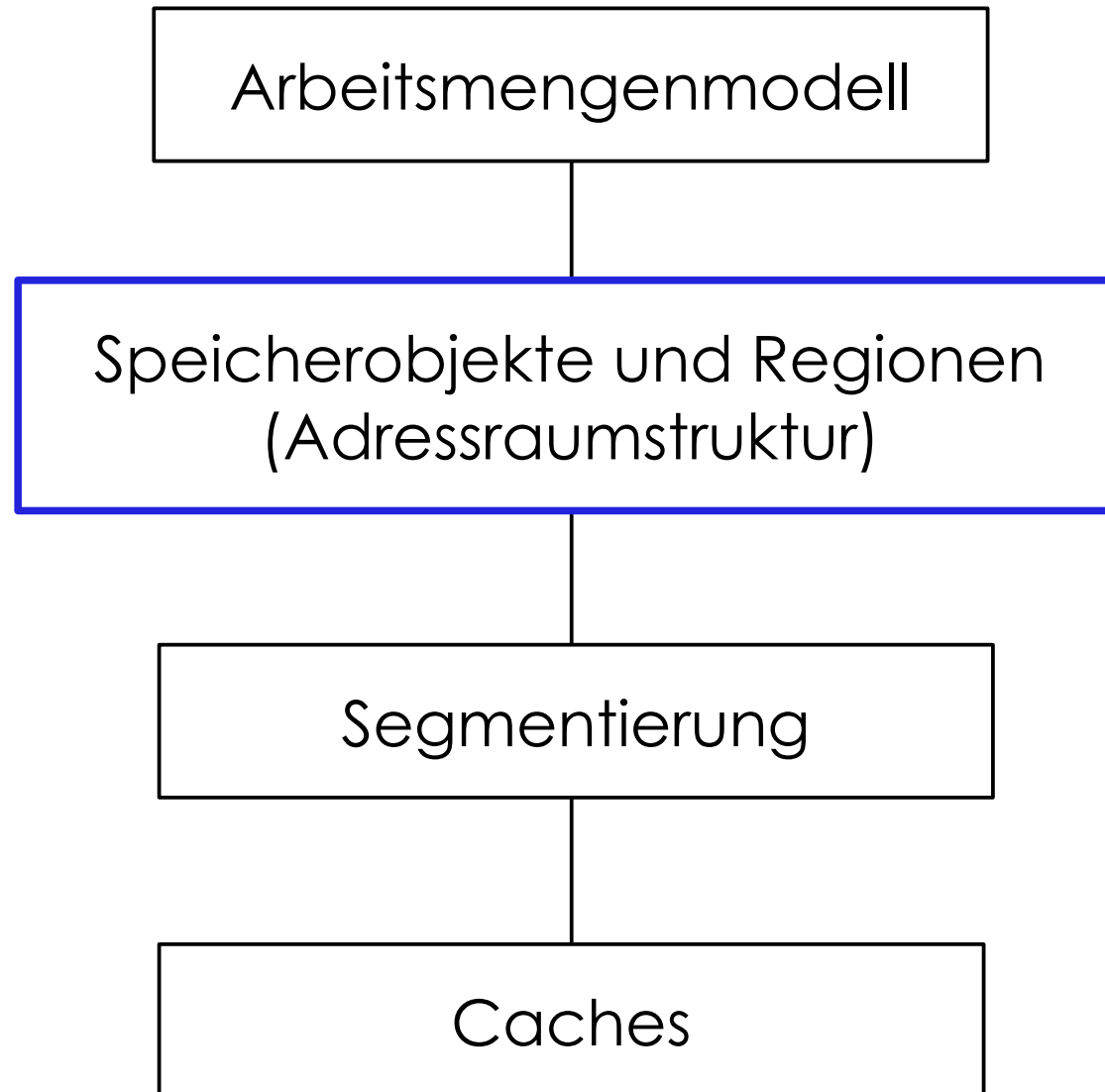
Residente Seiten



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



Speicherobjekte

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

Speicherobjekte und Regionen

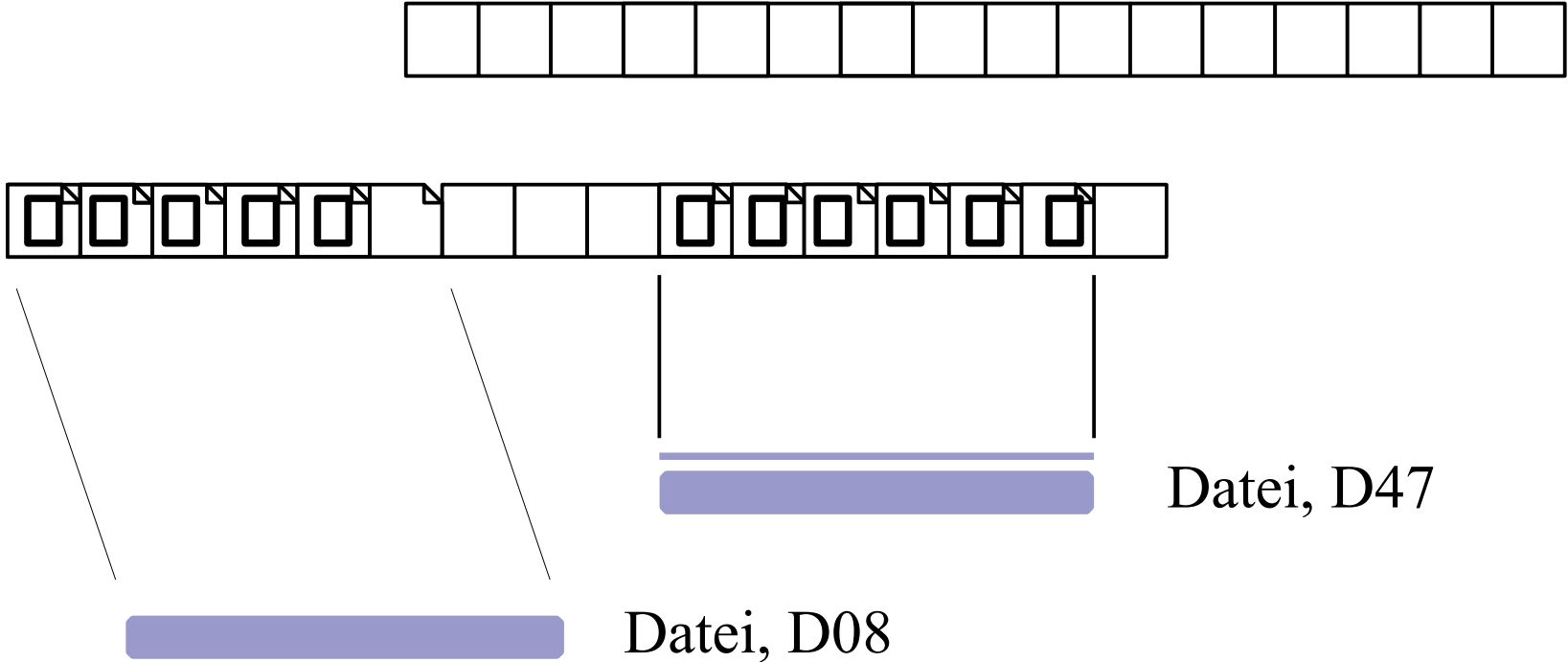


Datei, D47

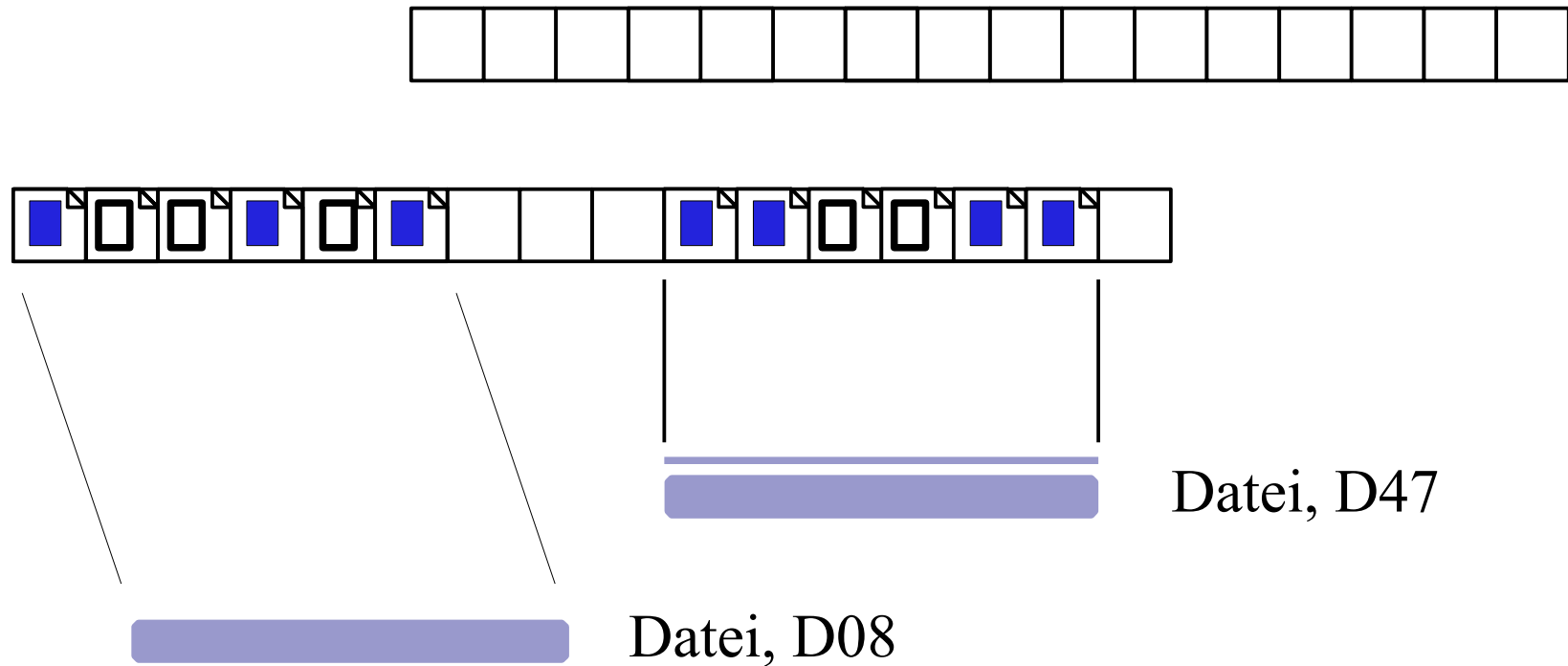


Datei, D08

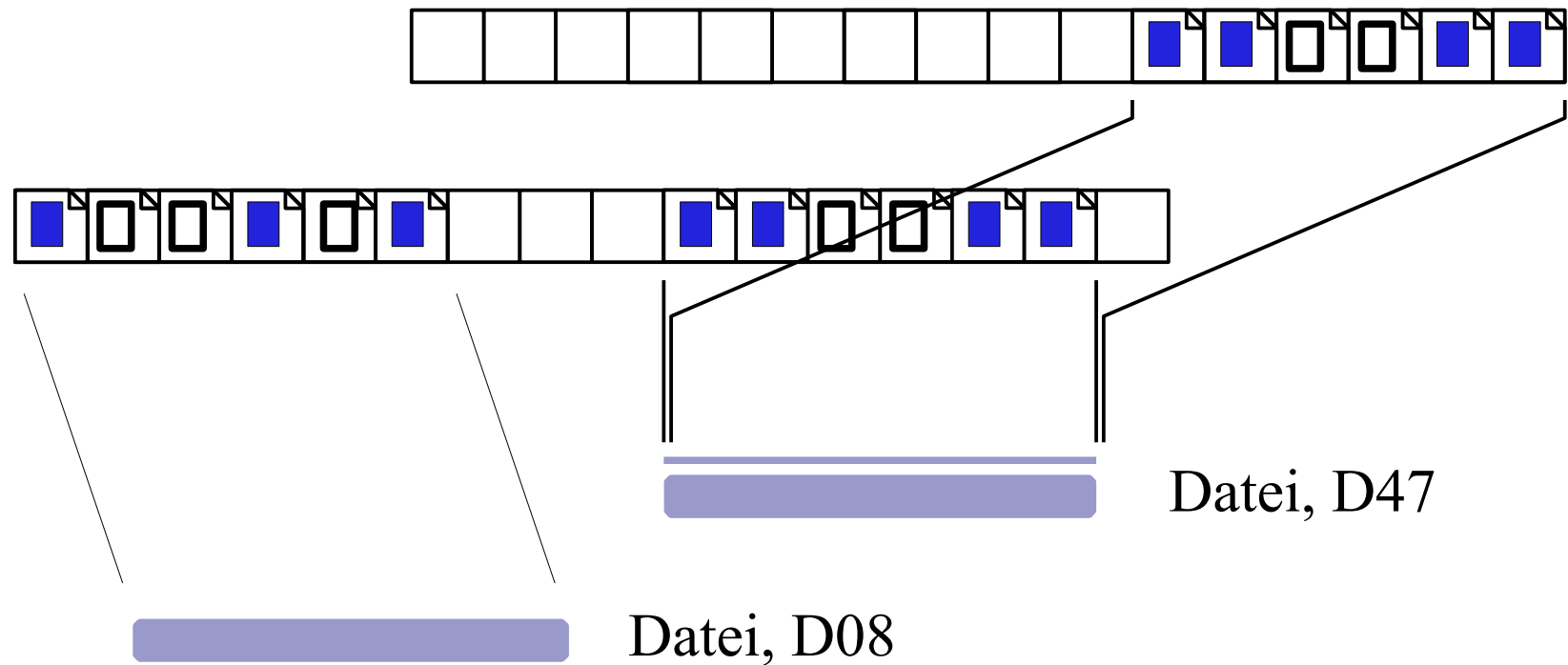
Speicherobjekte und Regionen



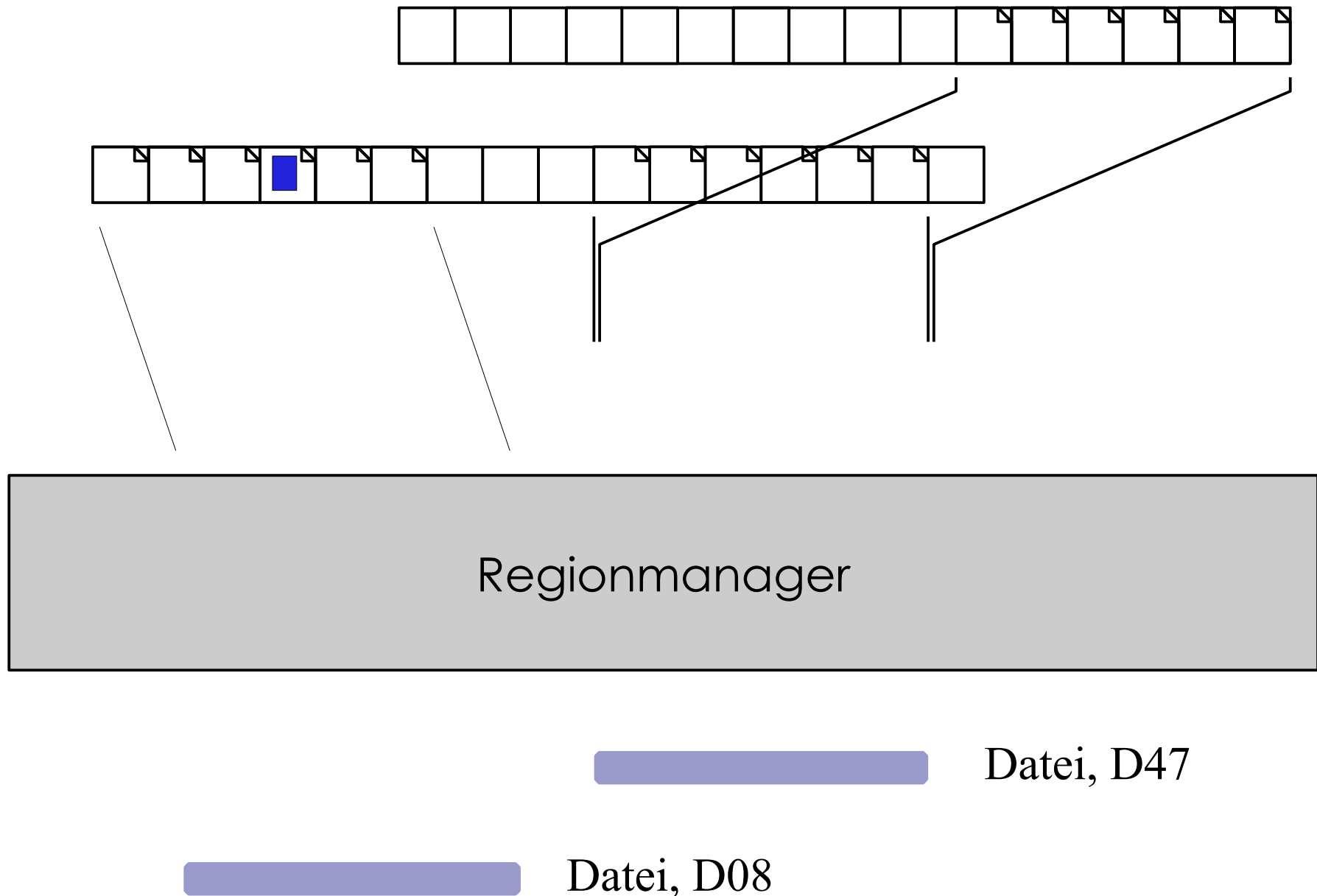
Speicherobjekte und Regionen



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 Objekt
- ♦ oder nicht definiert
- ♦ oder Rechteverletzung

Aufbau der Adressraumstruktur

- z. B.

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

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

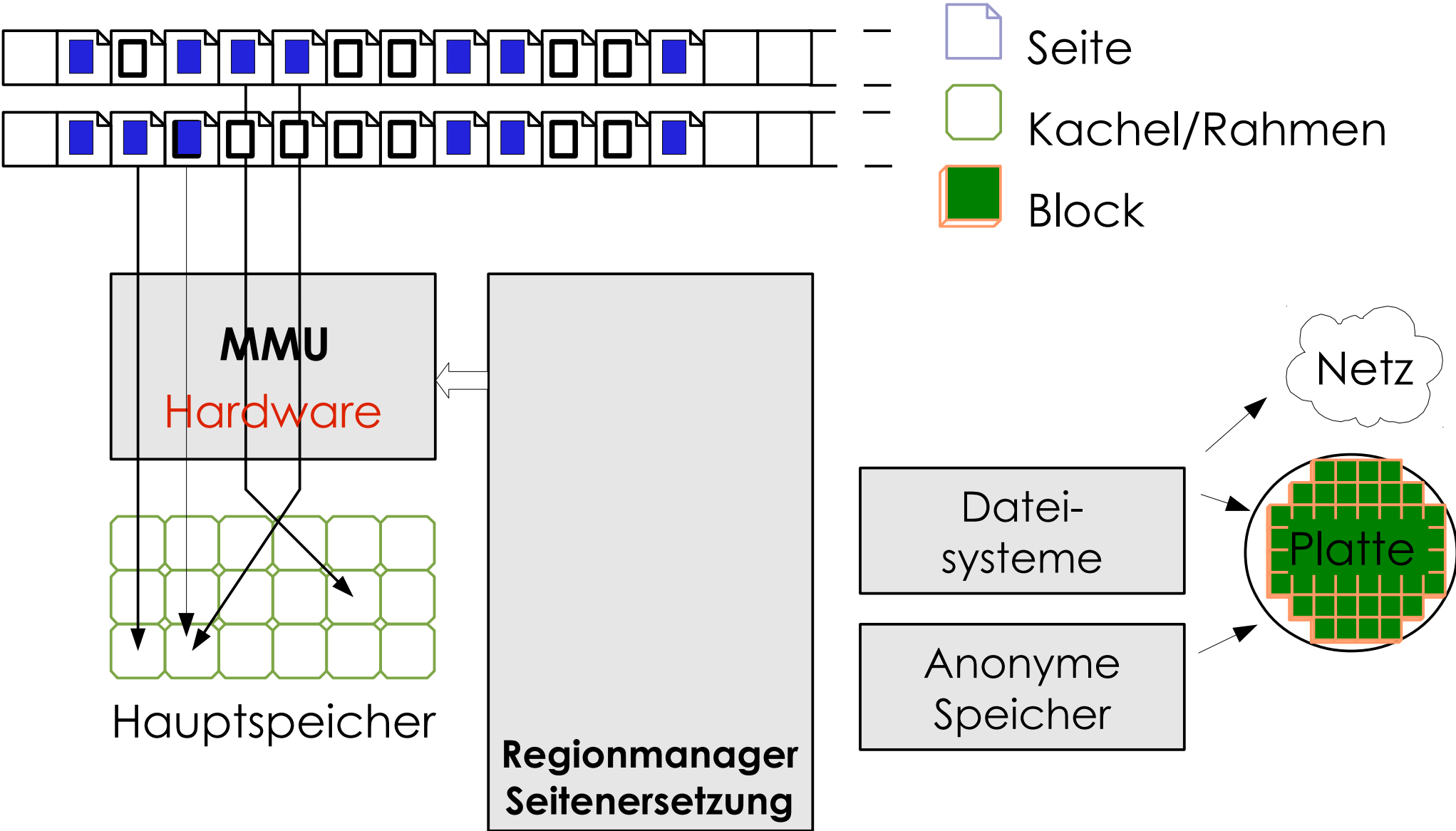
- implizit: bei Prozesserzeugung werden Programm-, Keller-, Daten-Regionen gebildet (in Unix: „Segmente“)

Regionmanager

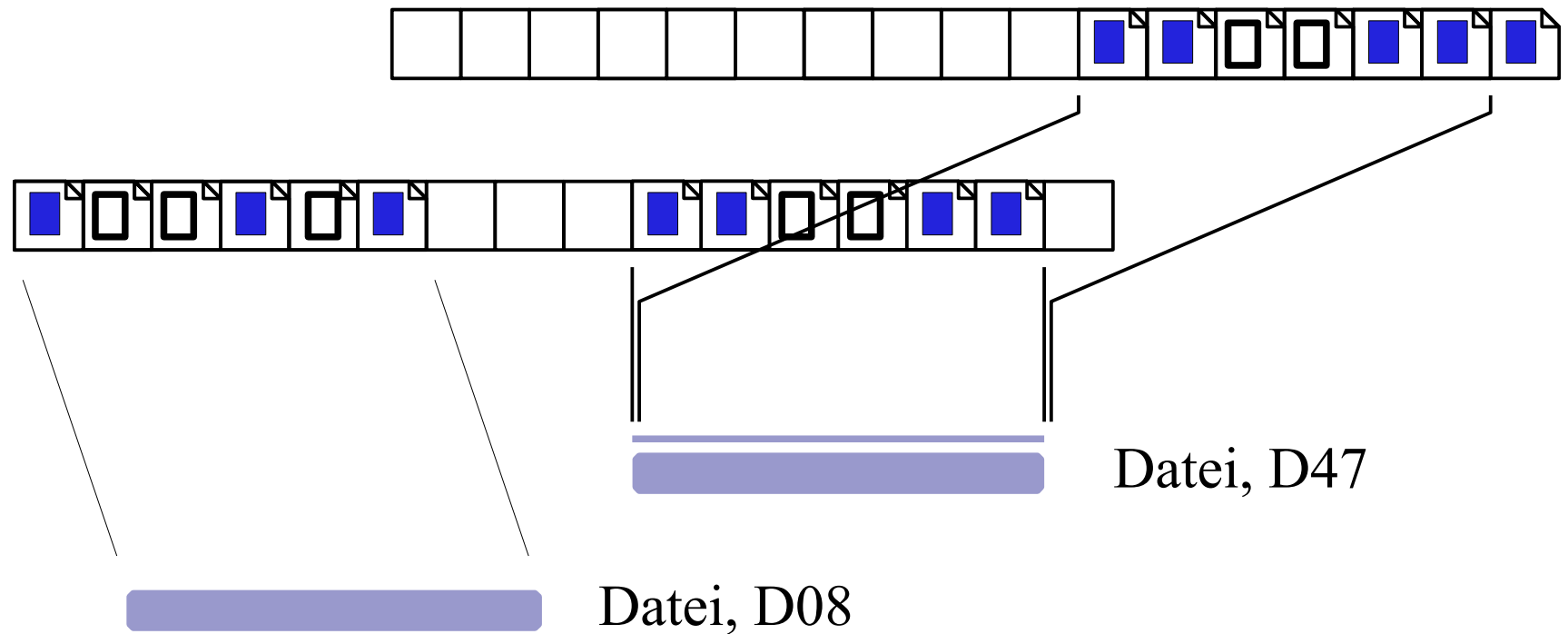
- Verwaltung der Adressraum-Struktur
 - repräsentiert z. B. als verkettete Liste
- Aufgabe bei Seitenfehler: Bestimmen von
 - ◆ Fehlerart
 - ◆ Speicherobjekt
 - ◆ Speicherseite
- Fehlerarten
 - ◆ Zugriff auf ungültigen Bereich (z.B. Pointerfehler)
 - ◆ Rechteverletzung (z.B. Schreiben in Code-Bereich)
 - ◆ Copy On Write Fault
 - ◆ sonst: Seite nicht in MMU eingetragen

Das Zusammenspiel

Adressräume z. B. 4 GB

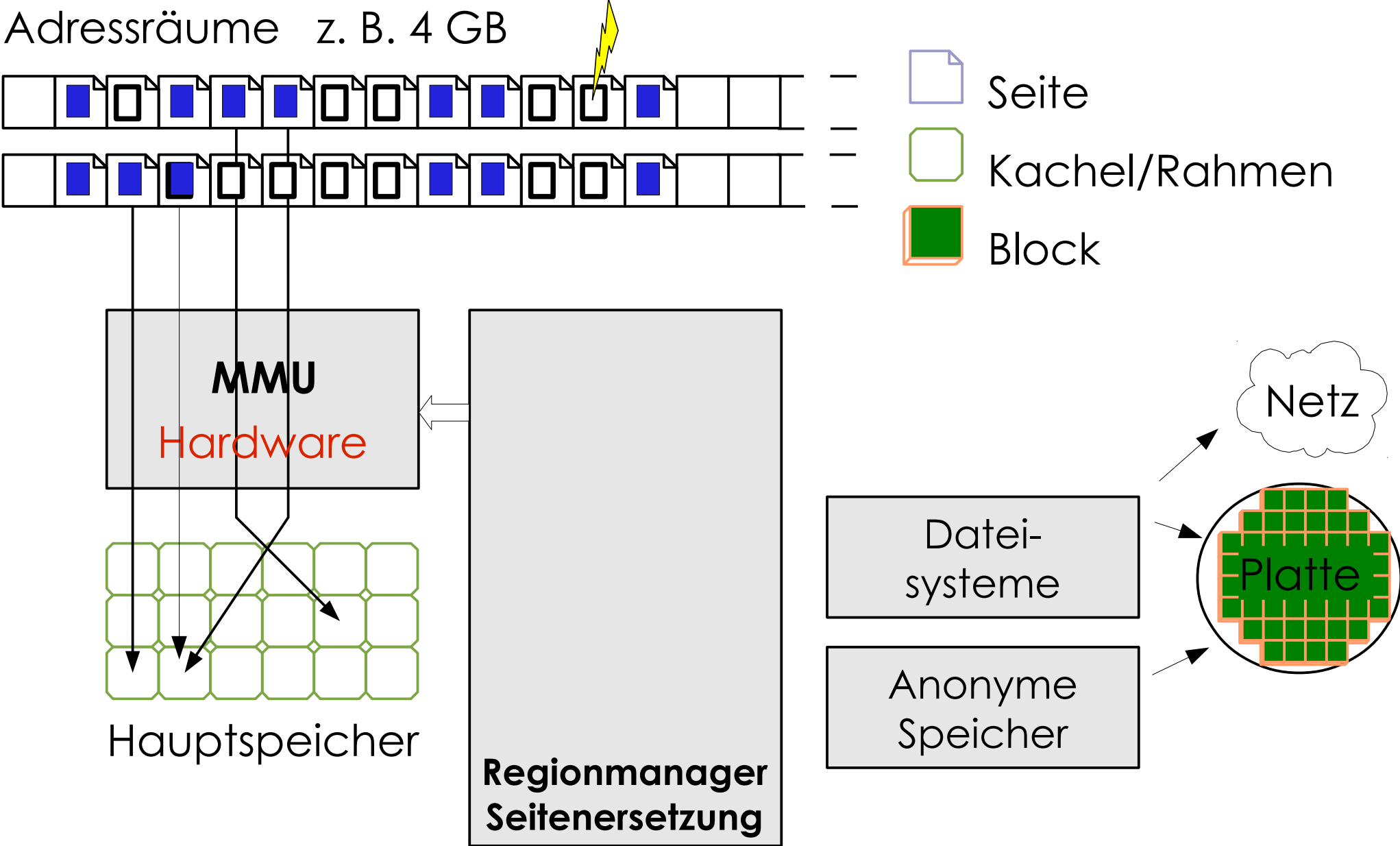


Speicherobjekte und Regionen



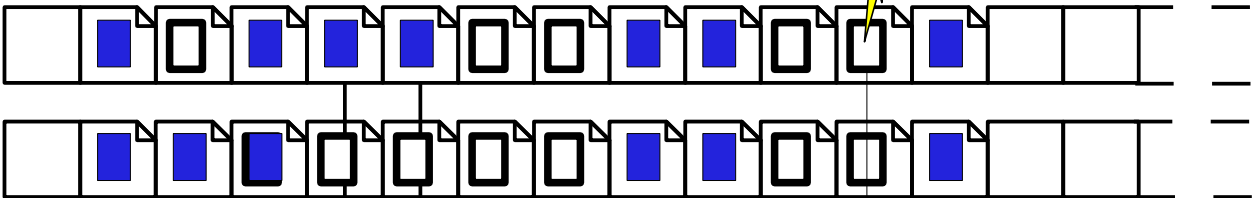
Das Zusammenspiel




Adressräume z. B. 4 GB

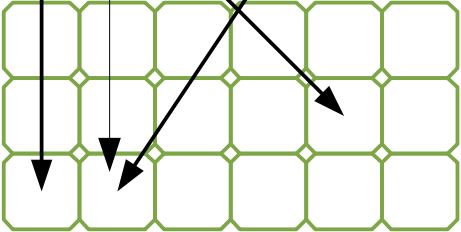
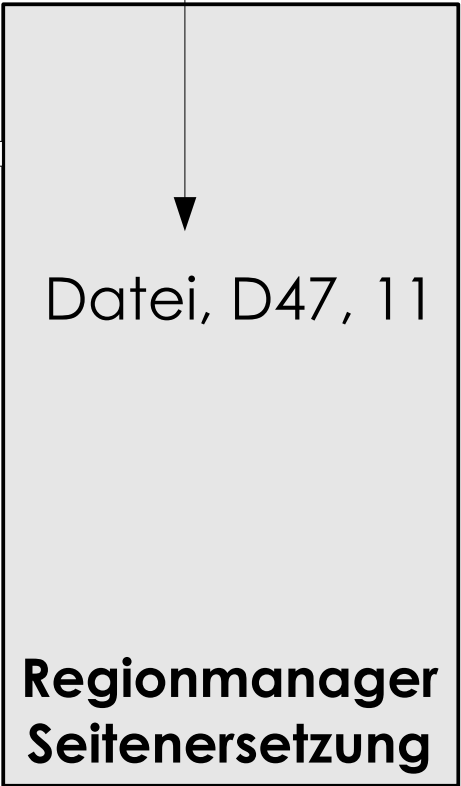
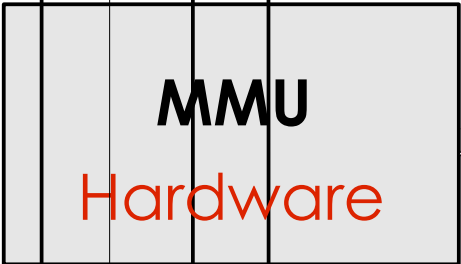


Das Zusammenspiel

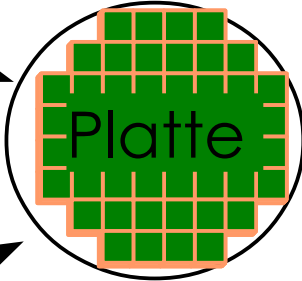
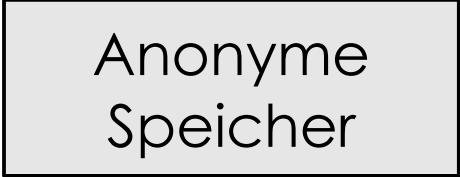
Adressräume z. B. 4 GB



-  Seite
-  Kachel/Rahmen
-  Block

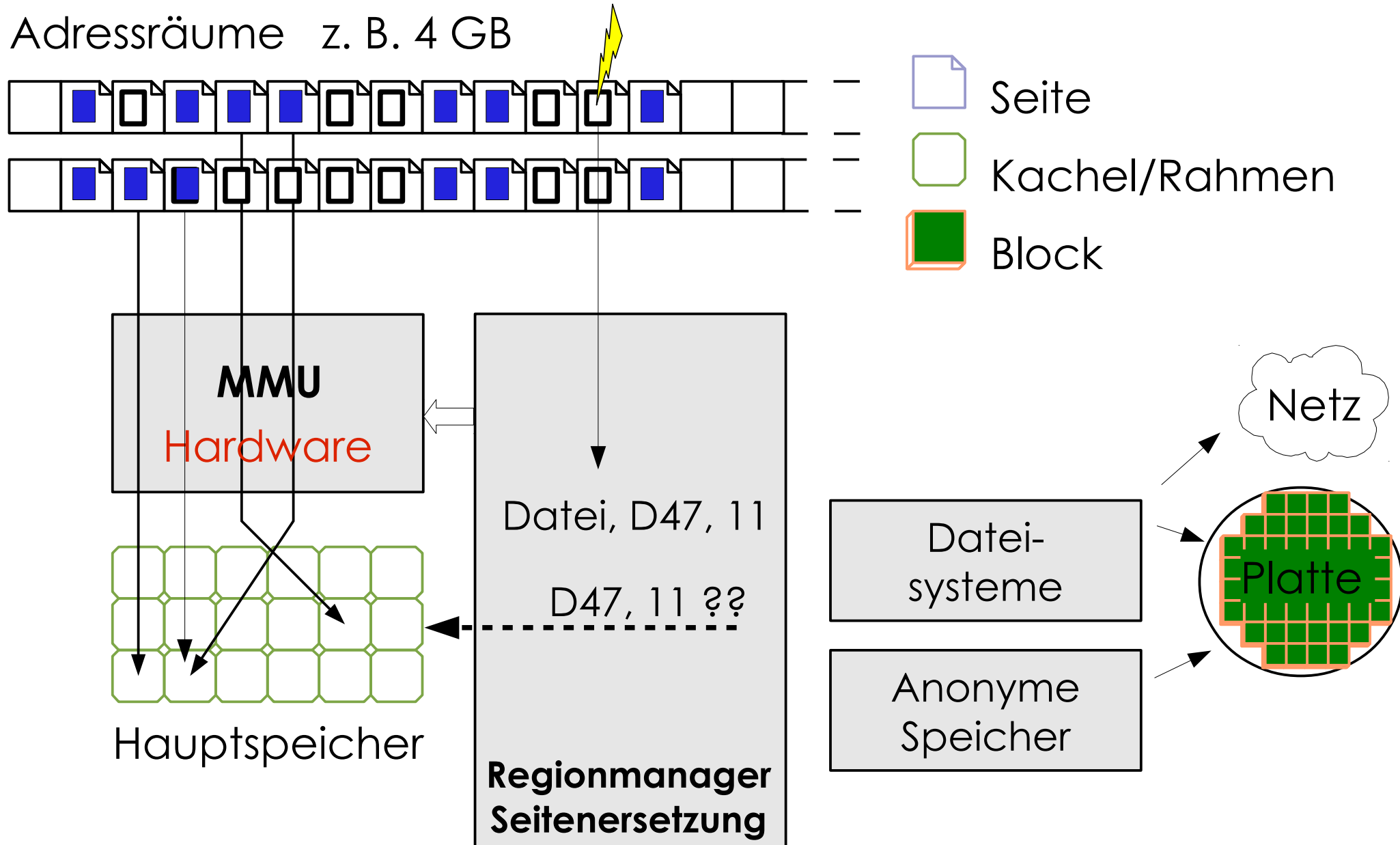


Hauptspeicher



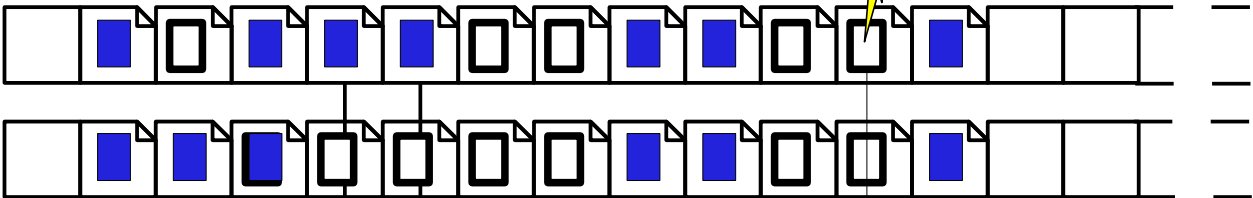
Das Zusammenspiel




Adressräume z. B. 4 GB

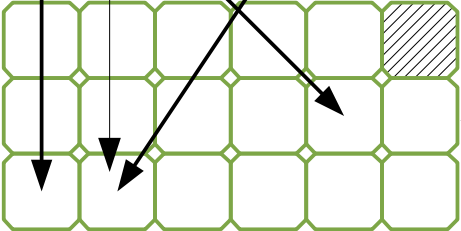
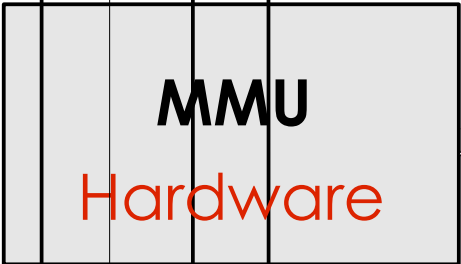


Das Zusammenspiel

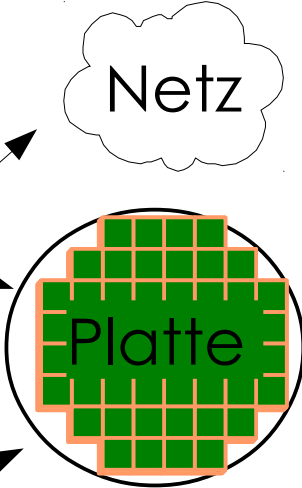
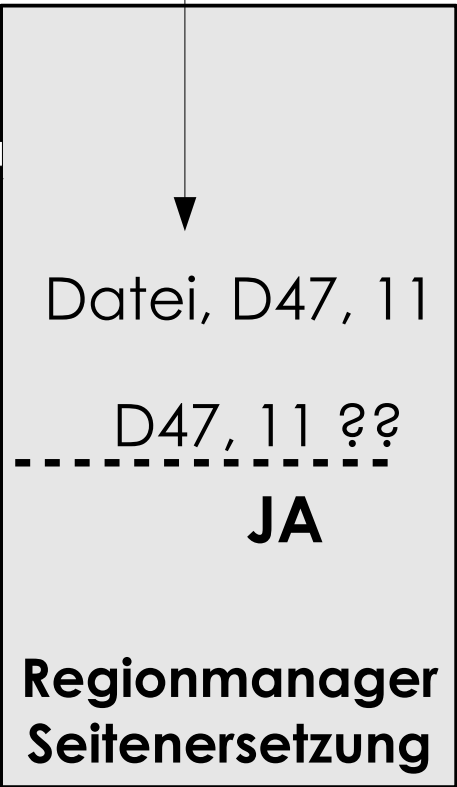
Adressräume z. B. 4 GB



-  Seite
-  Kachel/Rahmen
-  Block

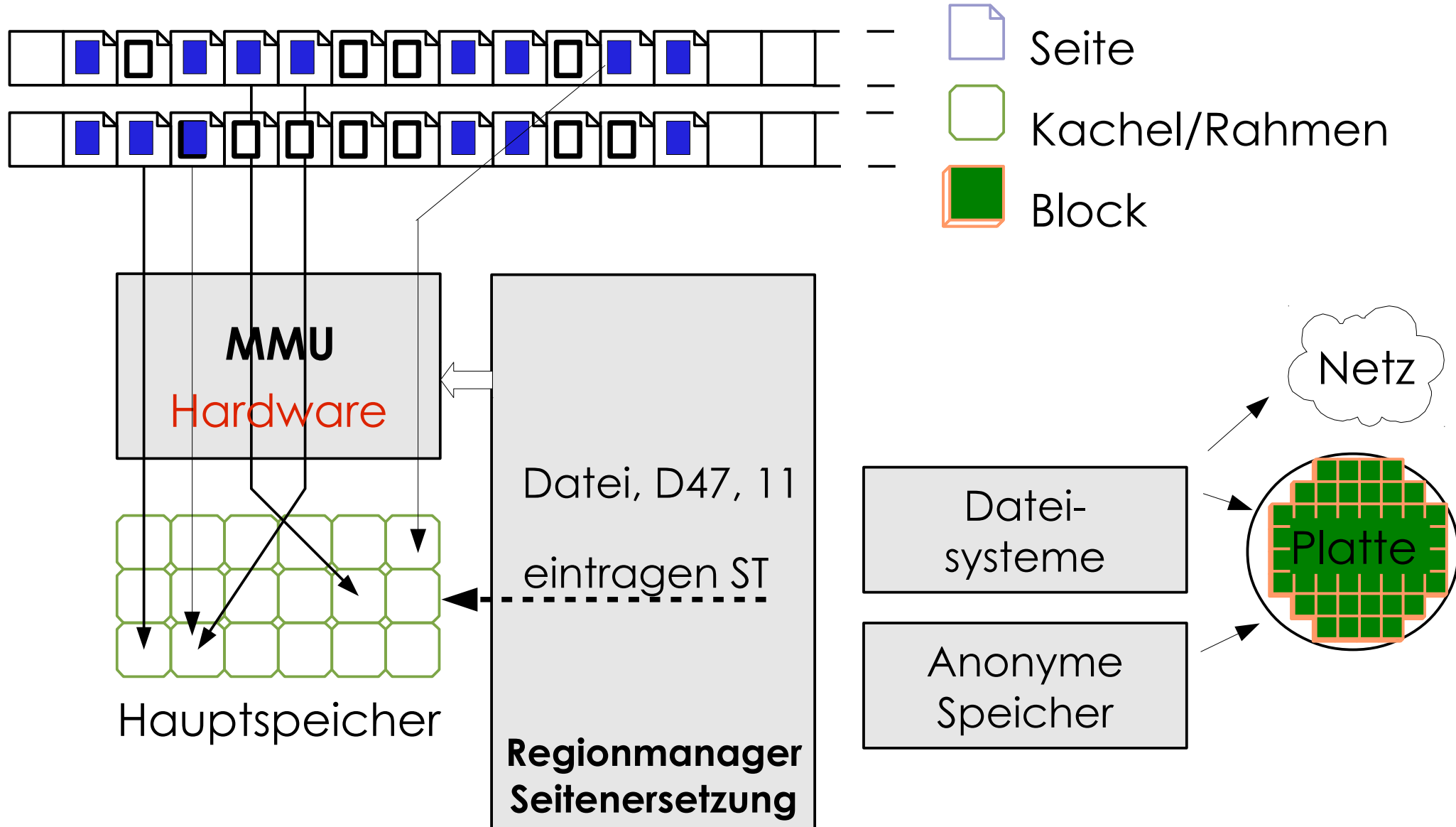


Hauptspeicher



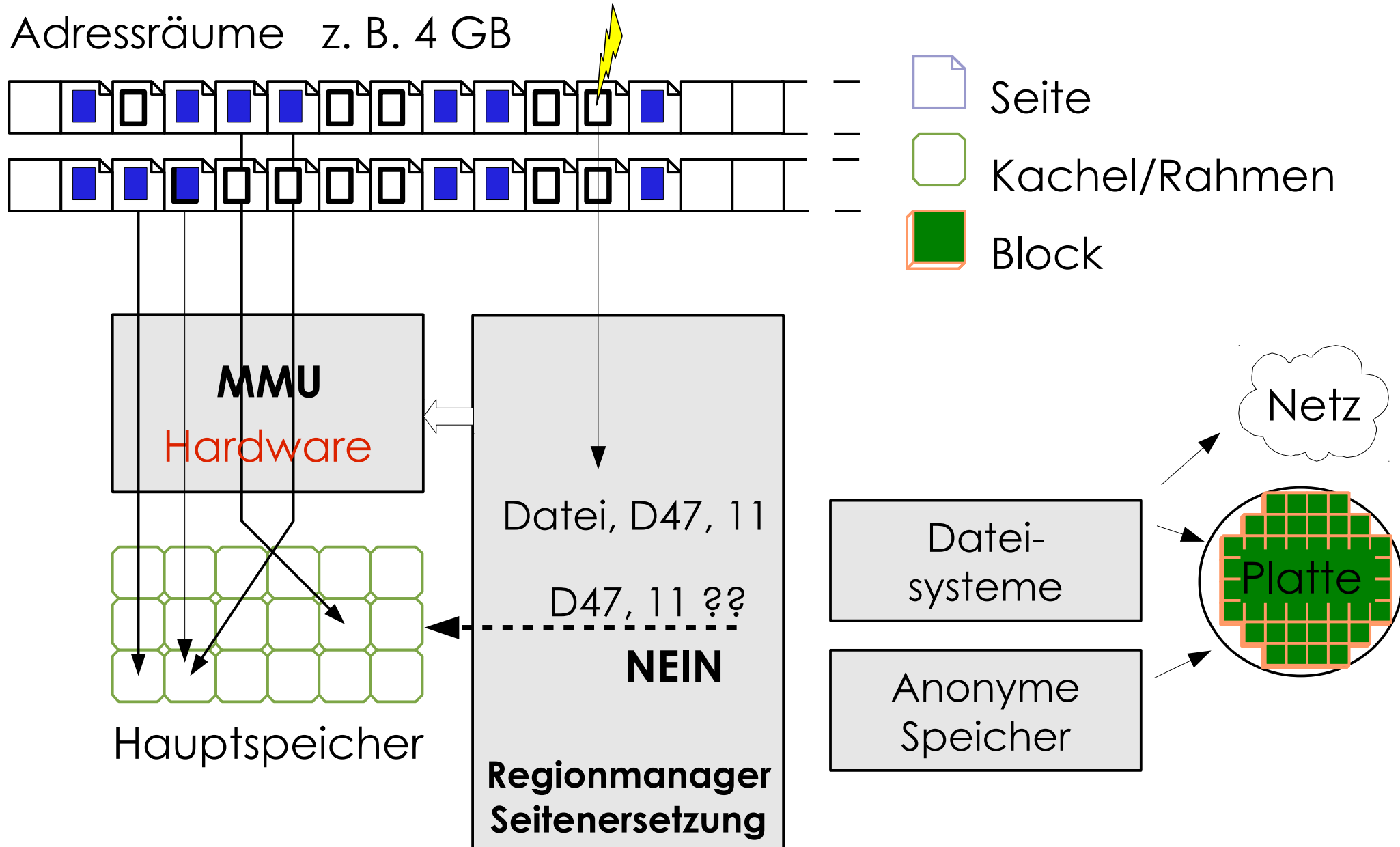
Das Zusammenspiel

Adressräume z. B. 4 GB



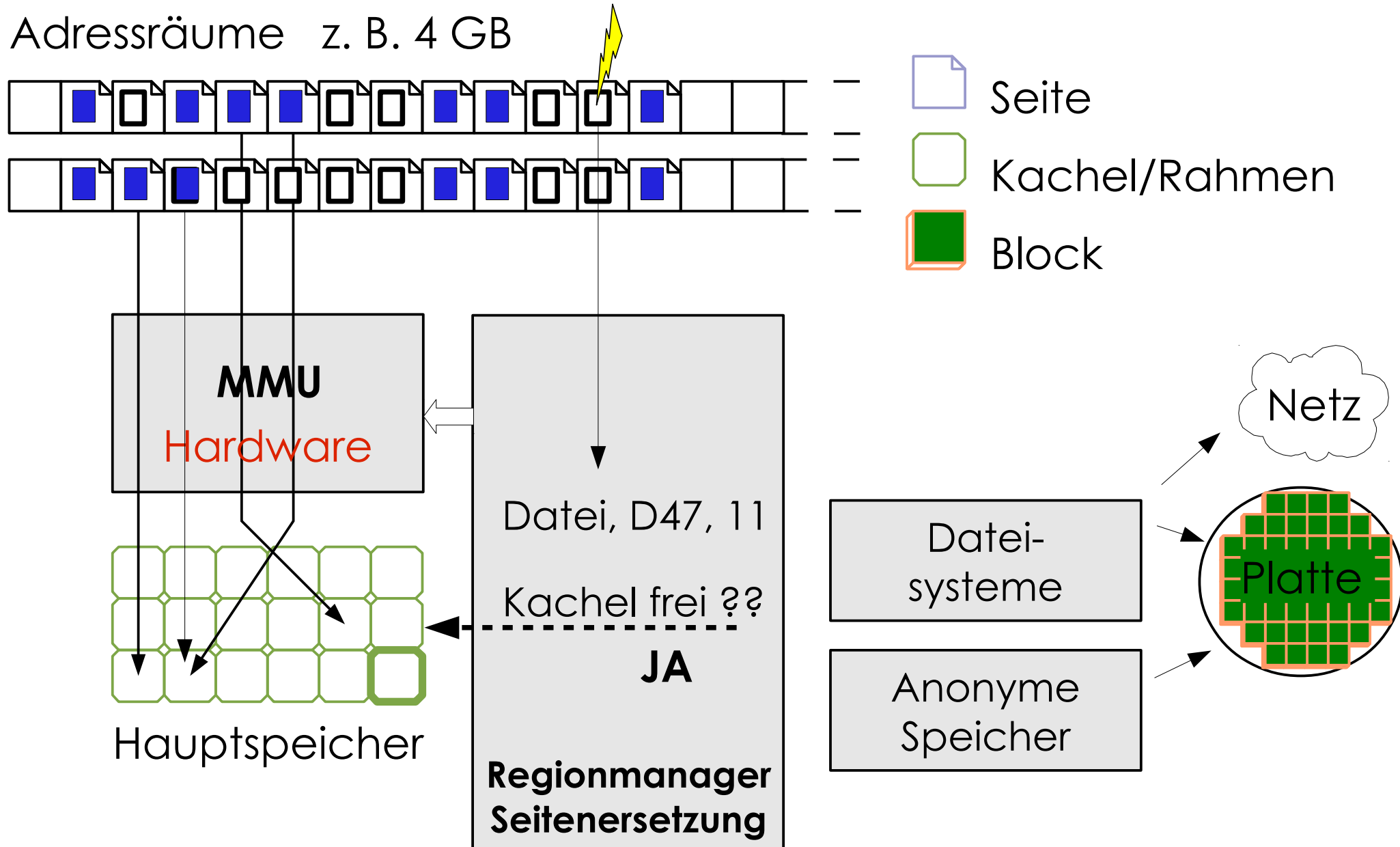
Das Zusammenspiel

Adressräume z. B. 4 GB



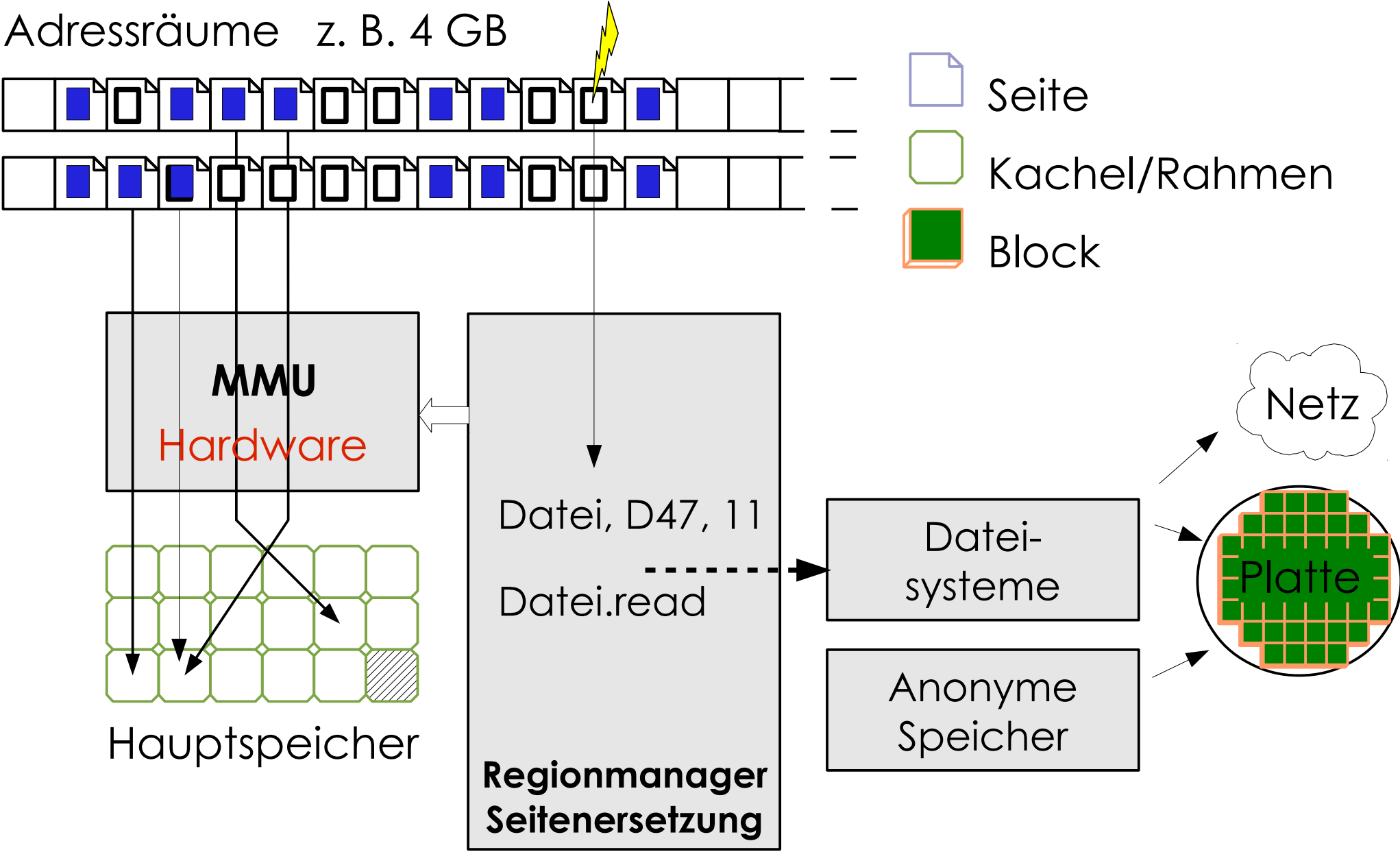
Das Zusammenspiel

Adressräume z. B. 4 GB



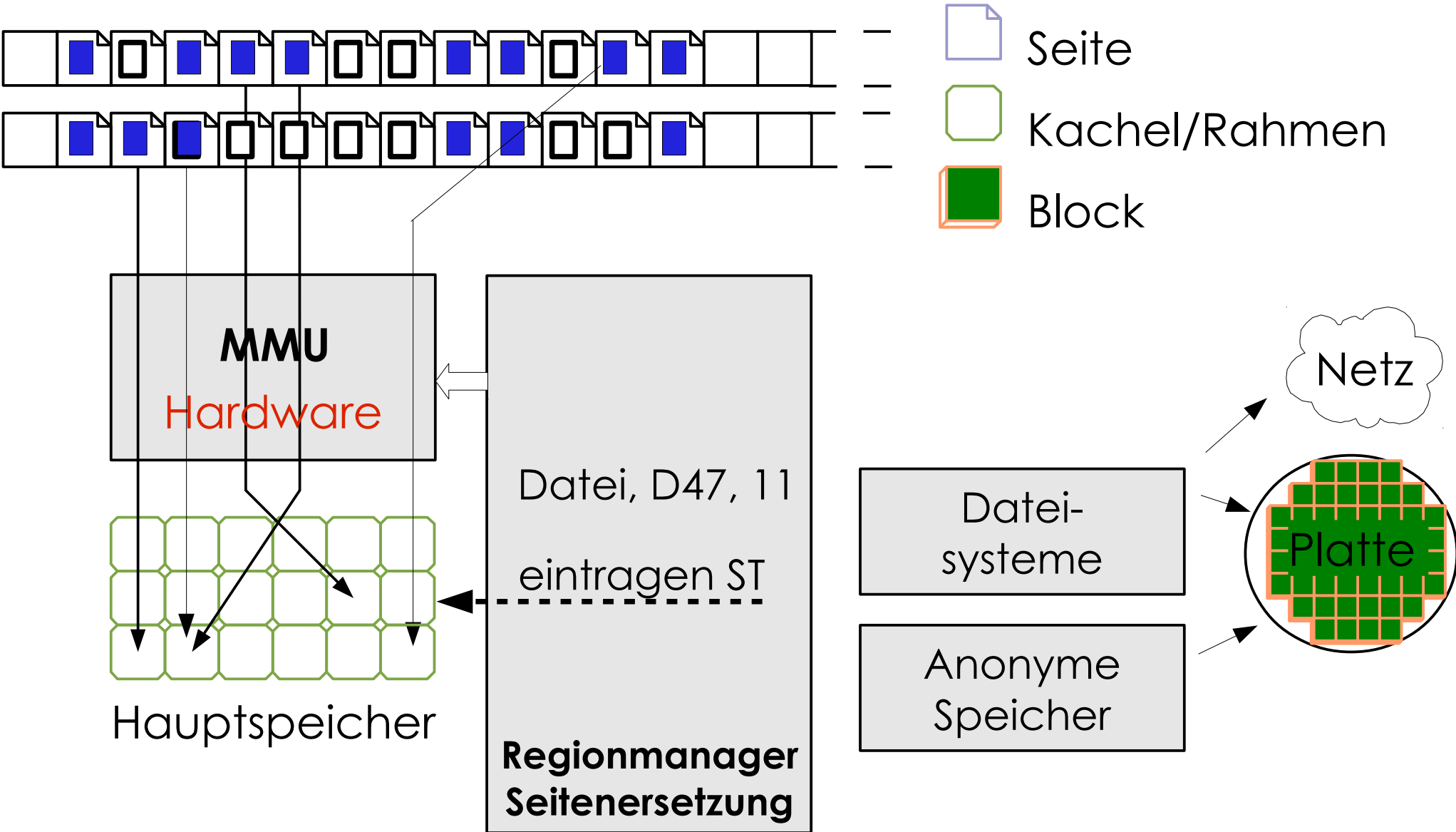
Das Zusammenspiel

Adressräume z. B. 4 GB



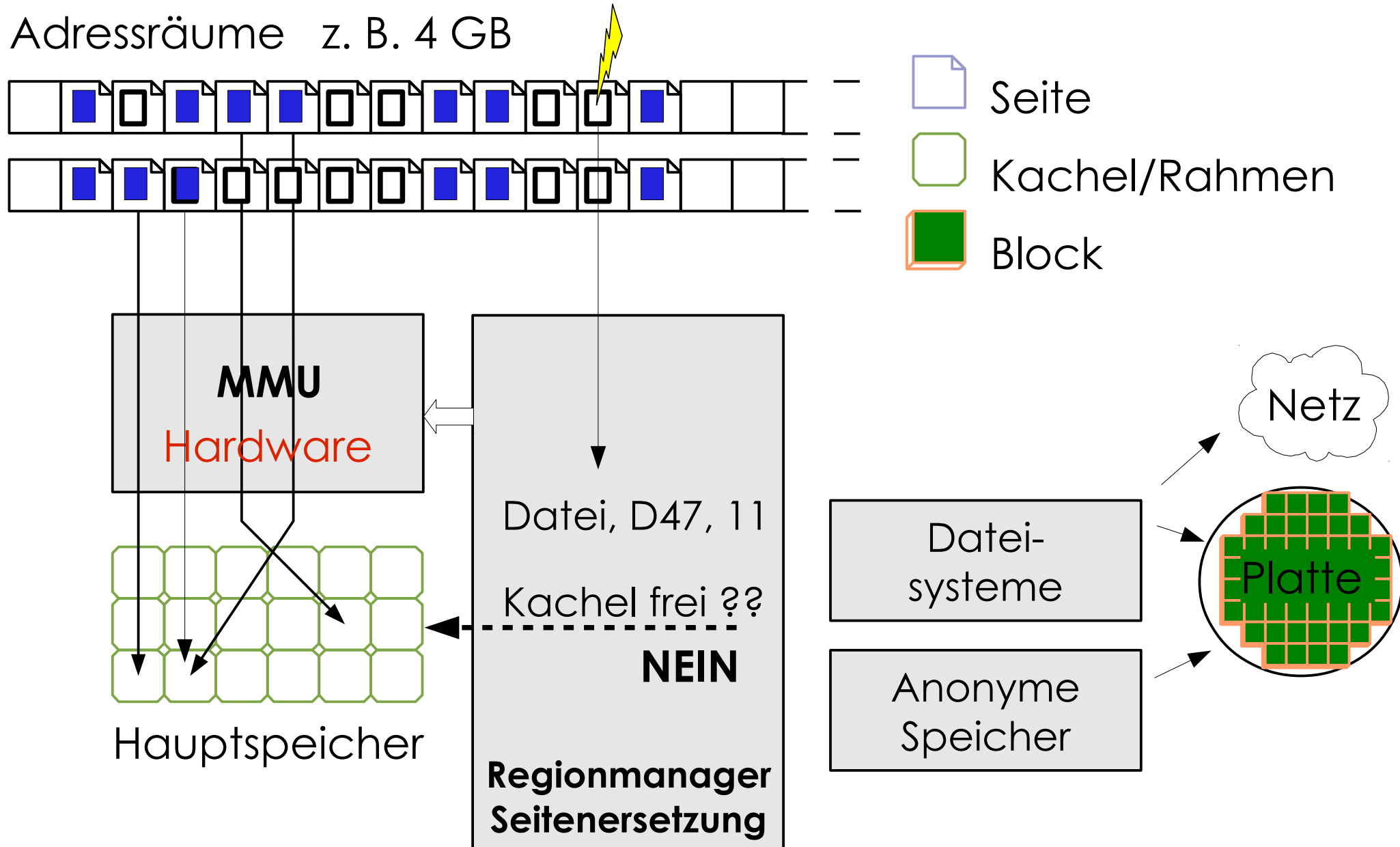
Das Zusammenspiel

Adressräume z. B. 4 GB



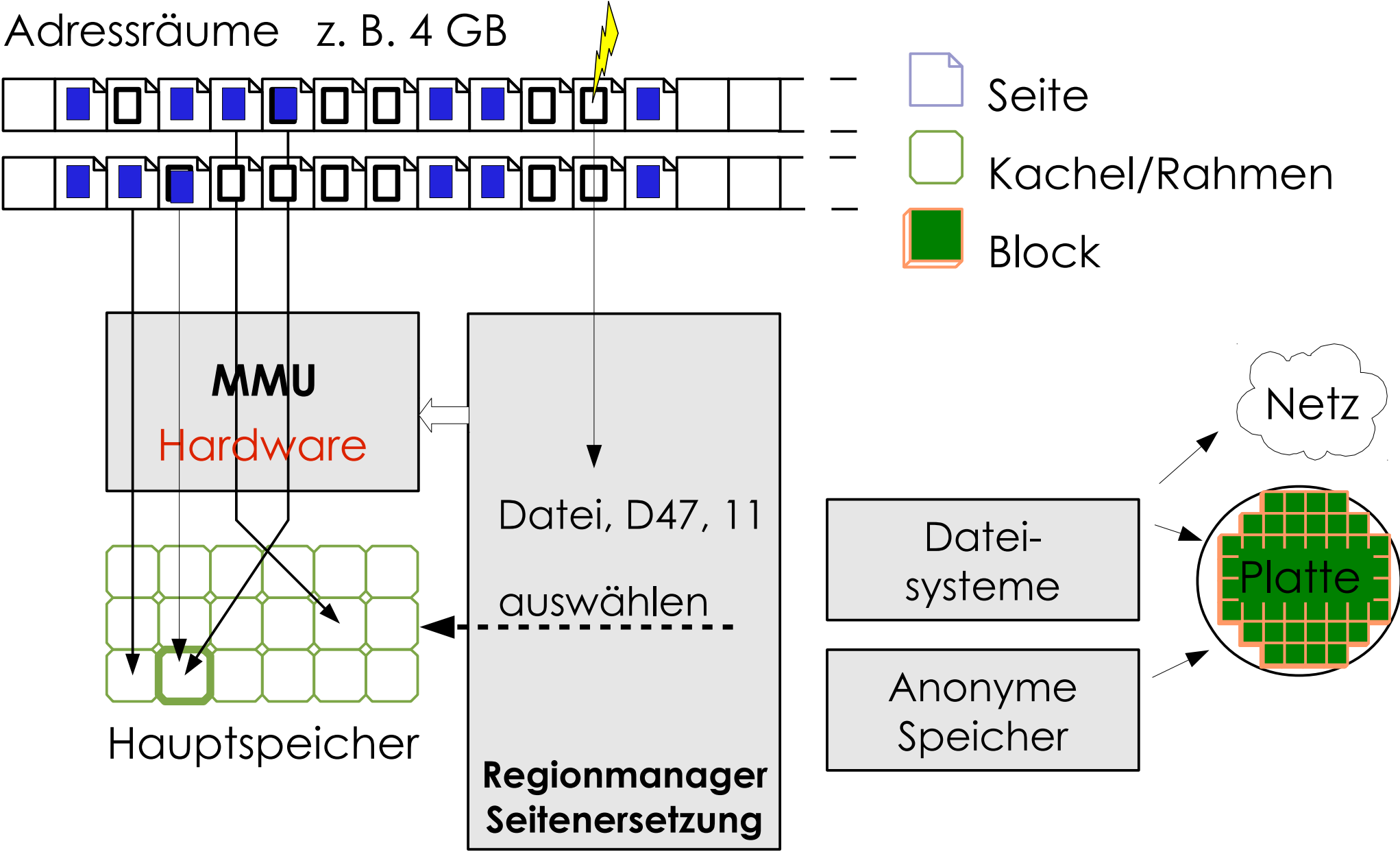
Das Zusammenspiel

Adressräume z. B. 4 GB



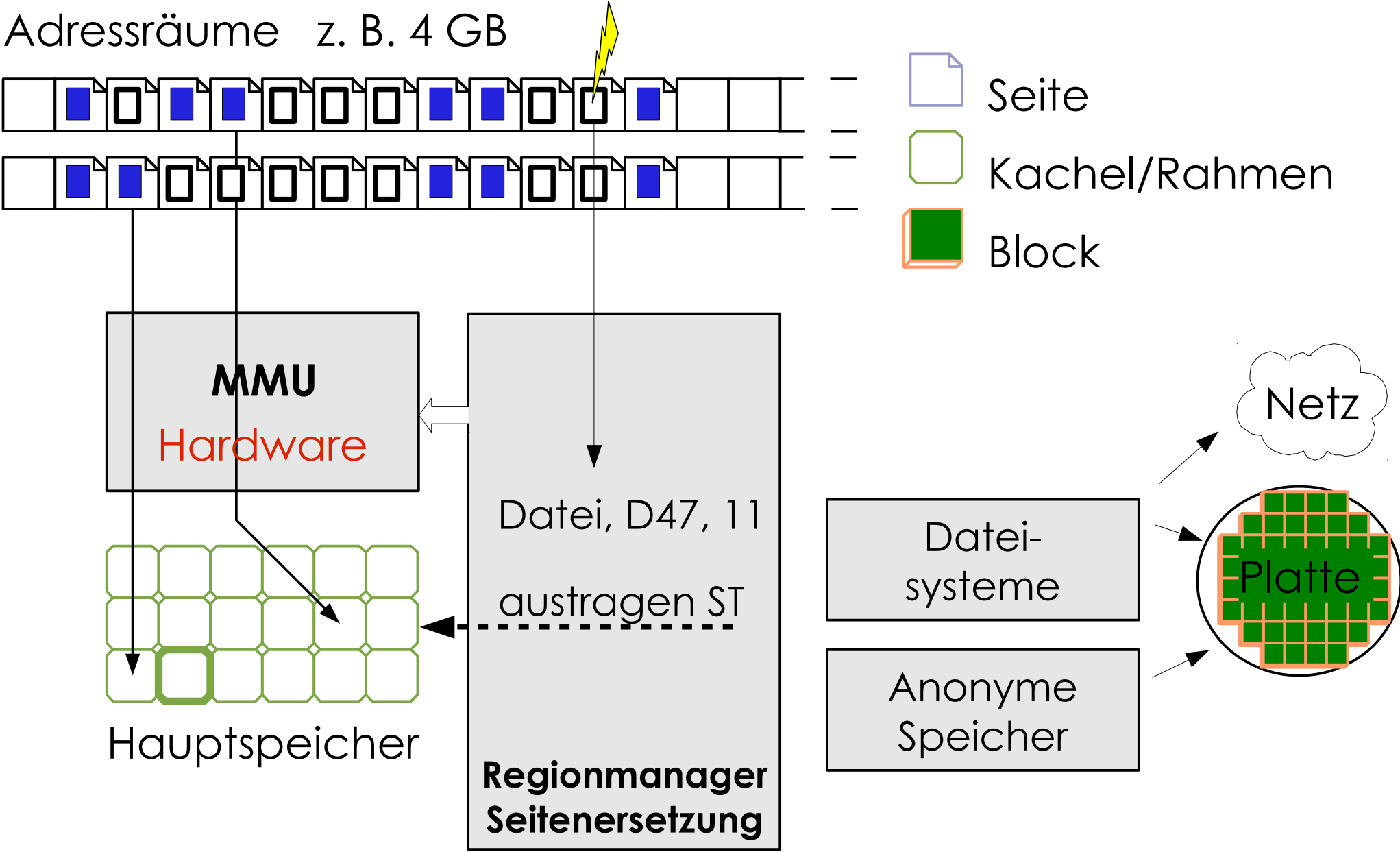
Das Zusammenspiel

Adressräume z. B. 4 GB



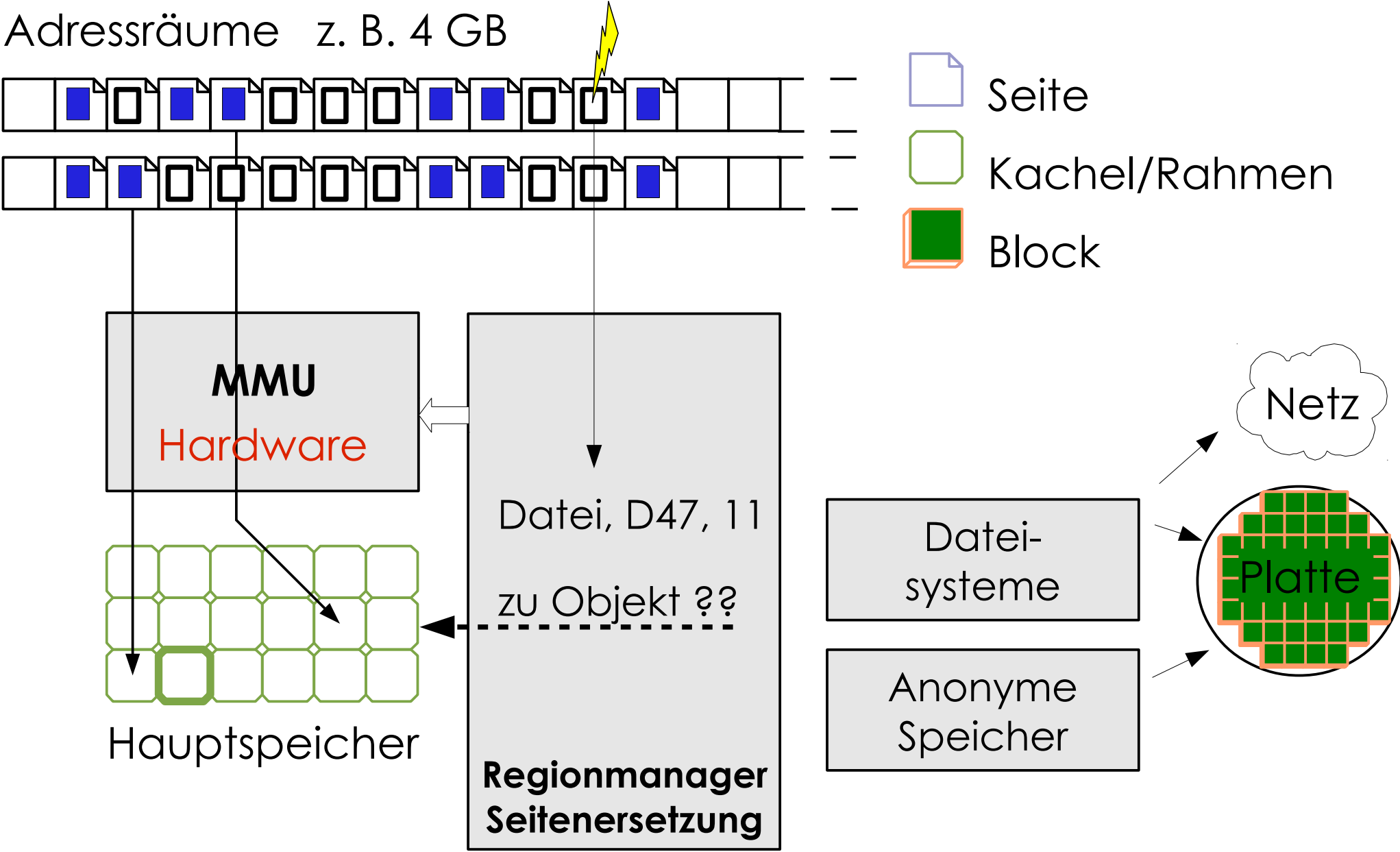
Das Zusammenspiel

Adressräume z. B. 4 GB



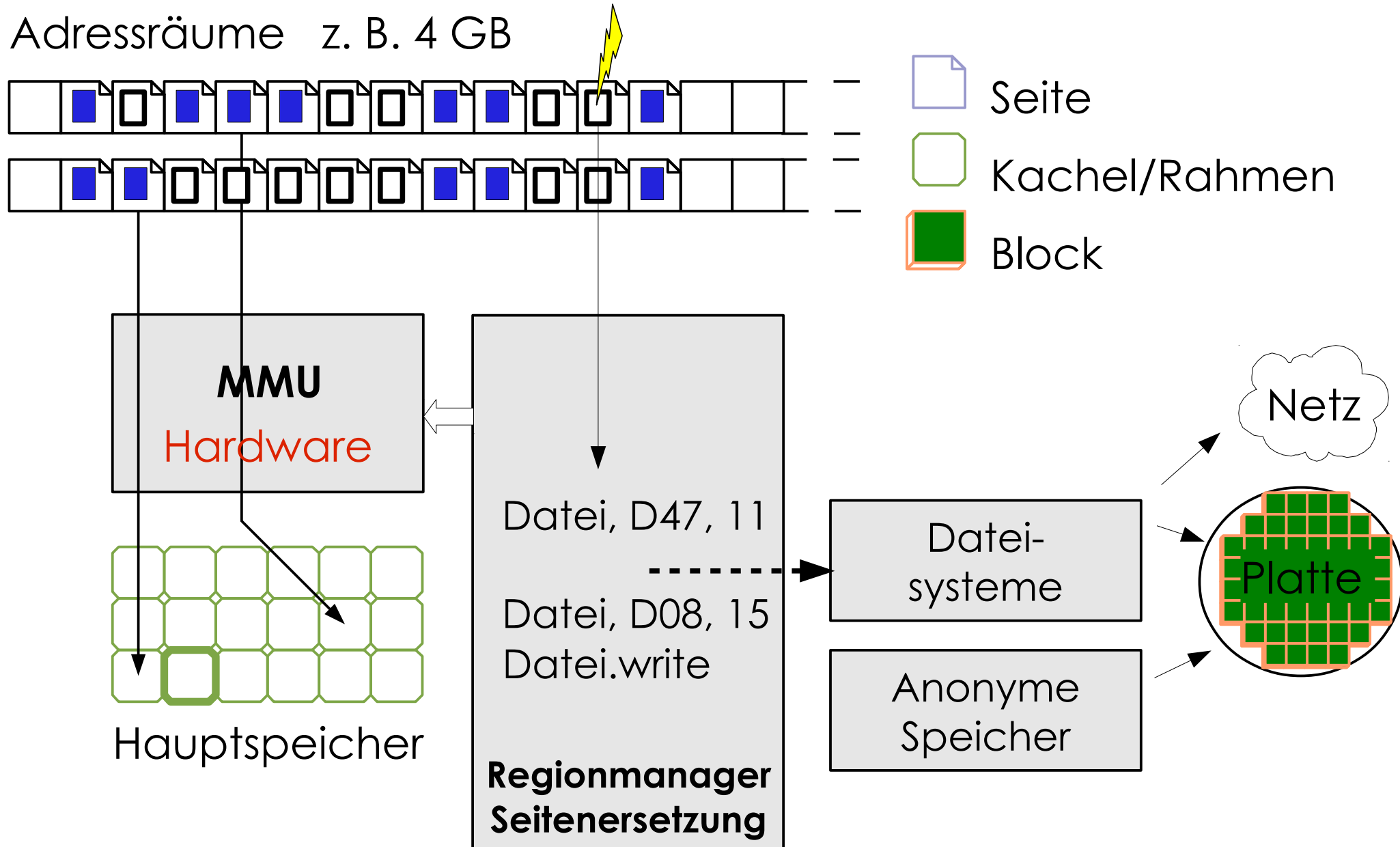
Das Zusammenspiel

Adressräume z. B. 4 GB



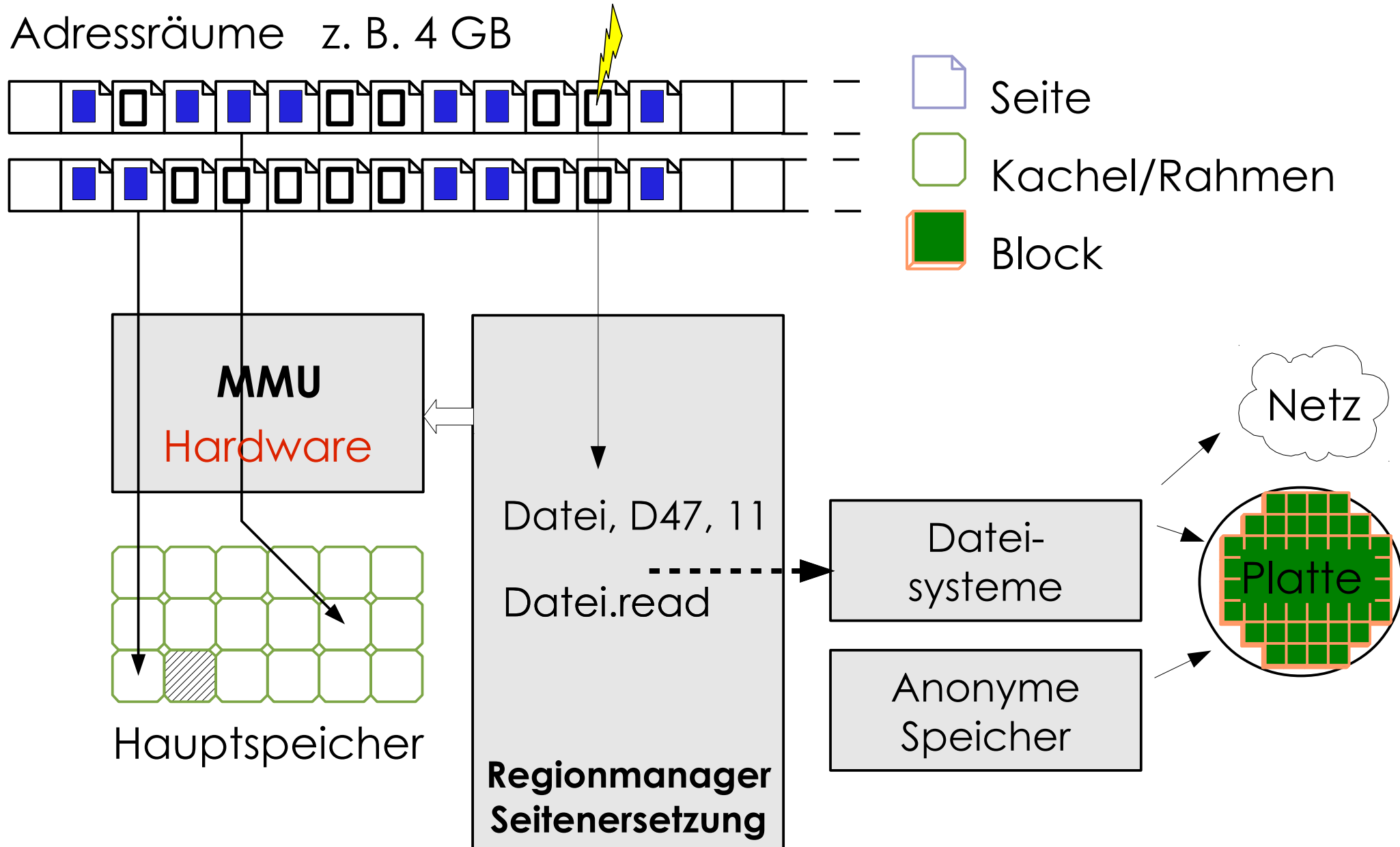
Das Zusammenspiel

Adressräume z. B. 4 GB



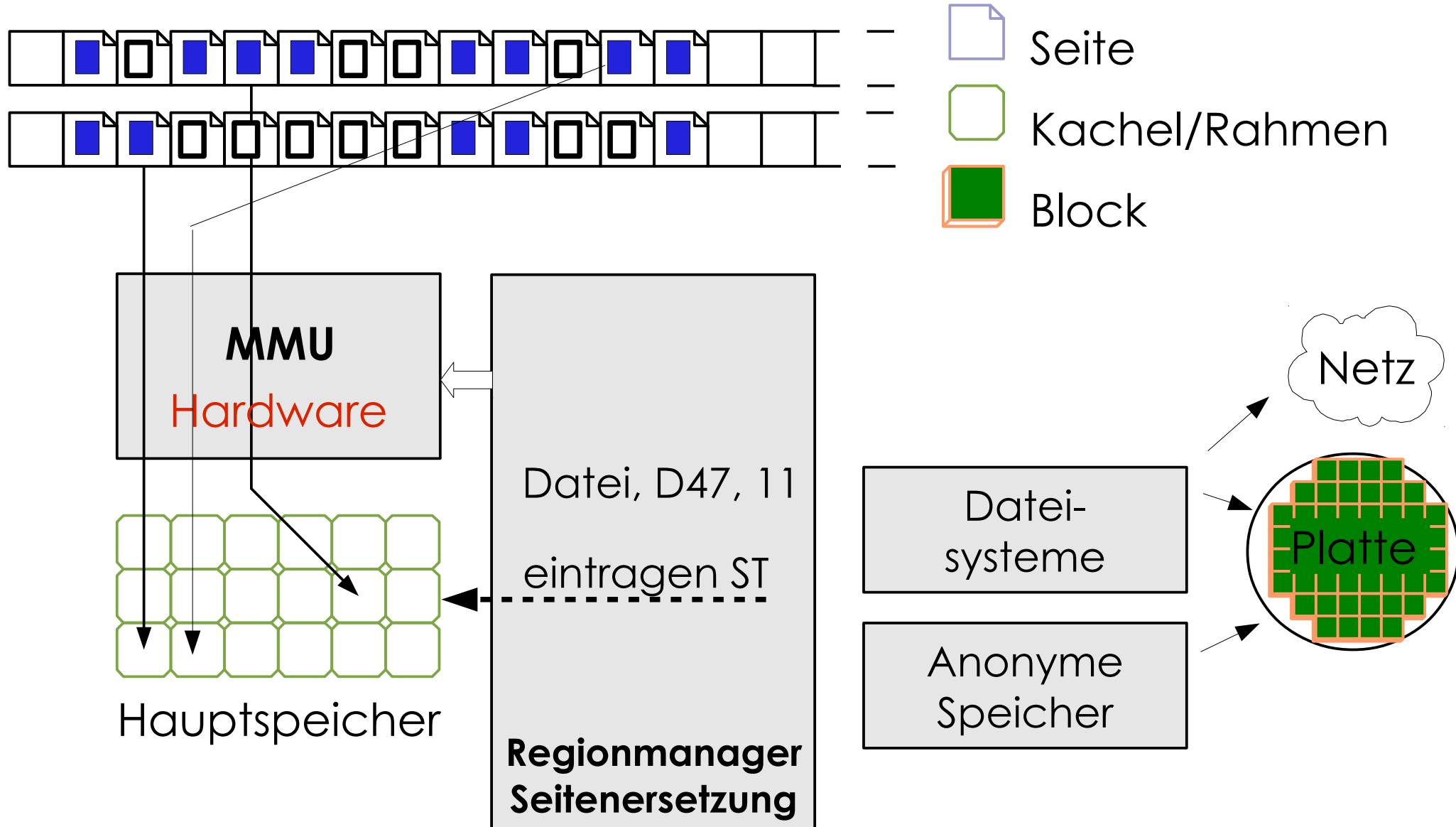
Das Zusammenspiel

Adressräume z. B. 4 GB



Das Zusammenspiel

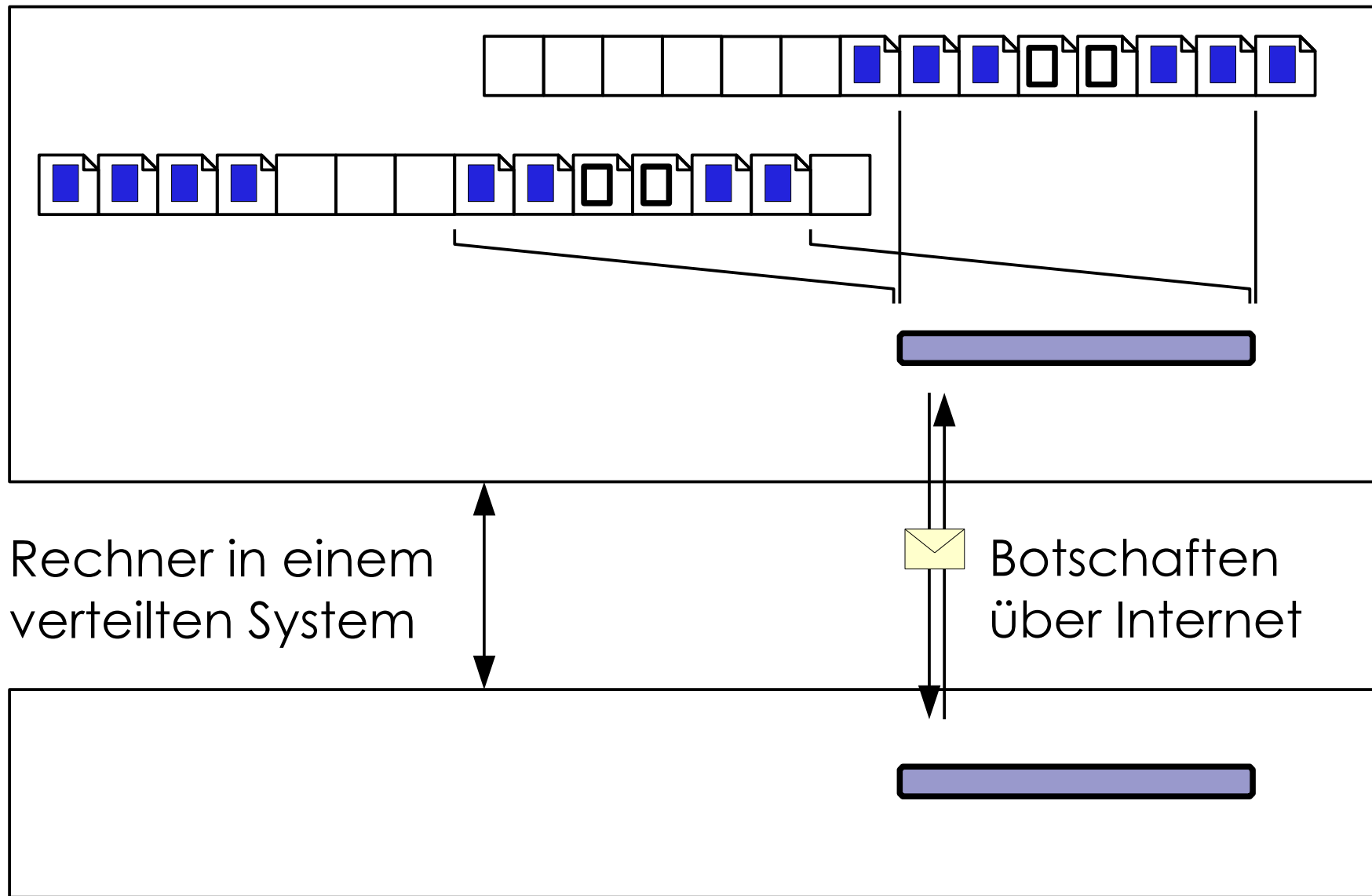
Adressräume z. B. 4 GB



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 Kachel → verdrängen (die geshared)
- Aus allen Seitentabelle 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*

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-Identifizier in TLB → löschen)
- Caches
- Botschaftenimplementierung ...