TECHNISCHE UNIVERSITÄT DRESDEN

Faculty of Computer Science Institute of Systems Architecture, Operating Systems Group

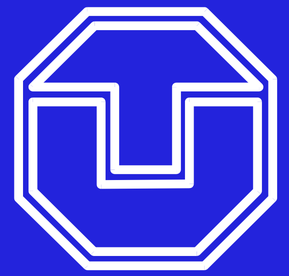
SPEICHER

TEILWEISE NACH ANDY TANENBAUM, MODERN OPERATING SYSTEMS

WS 2017/2018

Hermann Härtig

TU Dresden

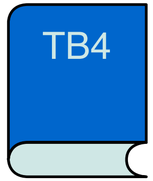


LERNZIELE

- von der BS-Schnittstelle zur HW !!
- “Virtuell”
- “past predicts future”, das Prinzip der Lokalität
- Lazy Copying (Copy on Write)
- Umgang mit Ressourcen
- …

Quellenangabe

Manche Grafiken und Beispiele nach:



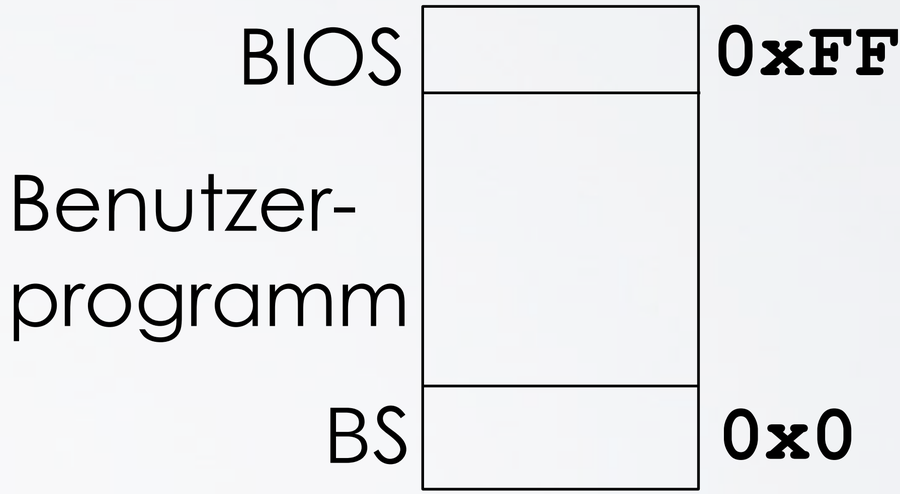
„Modern Operating Systems“
Andrew S. Tanenbaum und Herbert Bos
4. Ausgabe, Prentice Hall Press, 2014

WEGWEISER

- Einführung
- Elementare Techniken
- Virtueller Speicher (Paging)
  - Anliegen - Begriffe - Vorgehen
  - Adressumsetzung, Hardware, Lazy Copying, …
  - Seitenersetzung (Arbeitsmengenmodell)
  - Speicherobjekte
  - Integration
- Caches

STATISCH

- Statische Speicherverwaltung
  - d. h. keine Ein-/Auslagerung von Programmen/Daten
  - Einfachrechner (MS-DOS Version ??)
  - 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. Kapitel zu Threads):

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

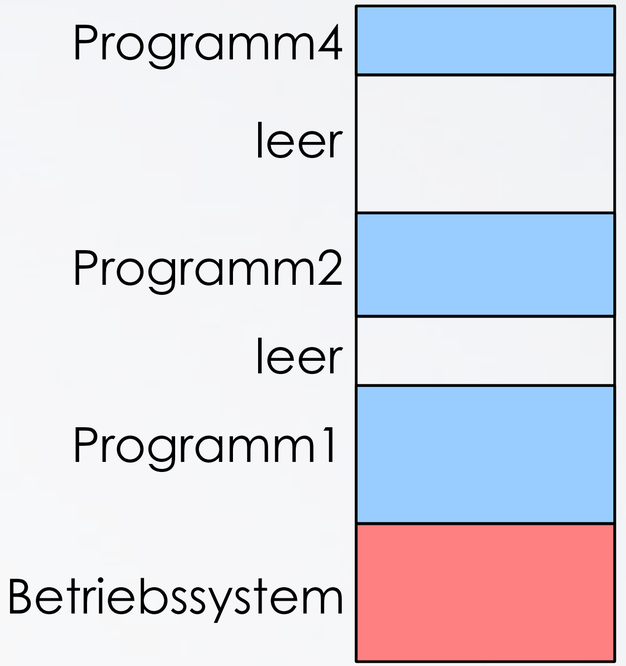
Multiprogramming vs. parallele Threads/Prozesse:

- parallele Prozesse Voraussetzung für Multiprogramming (mit oder ohne erzwungenen Prozesswechsel)
- 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
- Heute: Supercomputer werden in “Partitionen” aufgeteilt, Monoprogramming pro Partition

SPEICHER-PARTITIONIERUNG

Fragestellungen: **Relokation**

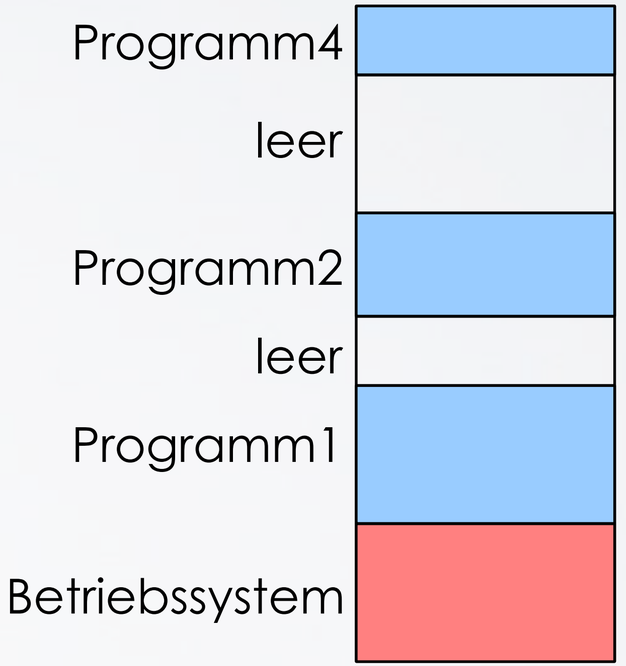
- Programme verwenden unterschiedliche Adressen, wenn sie in unterschiedlichen Partitionen ablaufen
- Abhilfe: Umsetzen der Adressen
  - beim/vor dem Laden (Software)
  - zur Laufzeit (Hardware)



SPEICHER-PARTITIONIERUNG

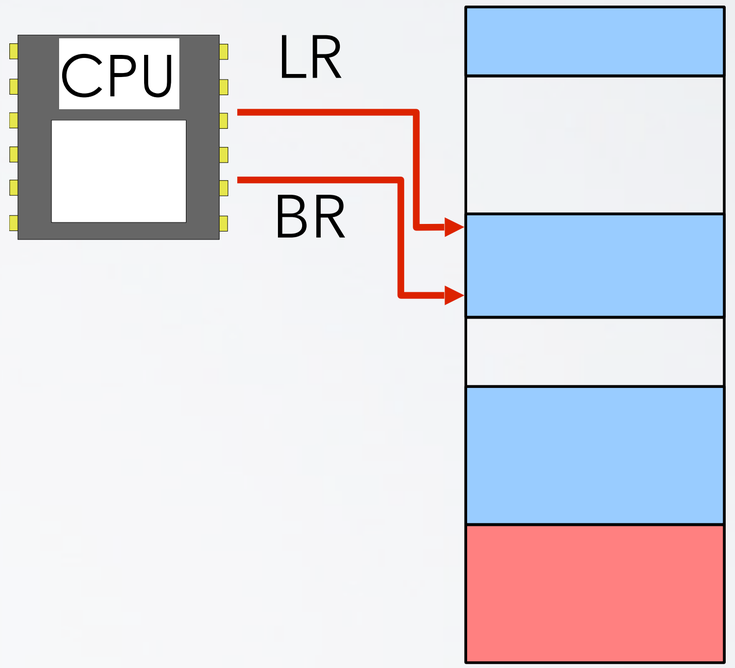
Fragestellungen: **Schutz**

- der Programme voreinander
- Abhilfe:
  Überprüfung der Adressen zur Laufzeit



EINFACHE VARIANTE: BR /LR

- 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 Prozessen:
  bei Umschaltung müssen auch
  BR und LR umgeschaltet werden



WACHSEN VON PARTITIONEN

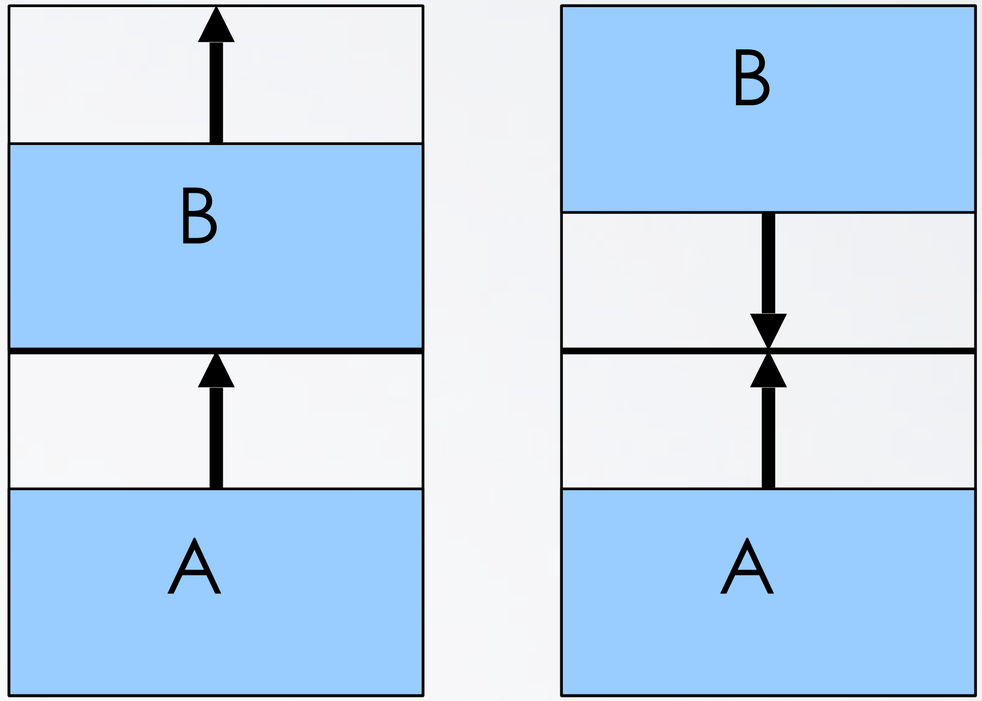
Algorithmen zur „Minimierung“ des Verschnitts

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

Verwaltung

- Bitmaps, Listen, Bäume

Ignoriert in dieser Vorlesung !

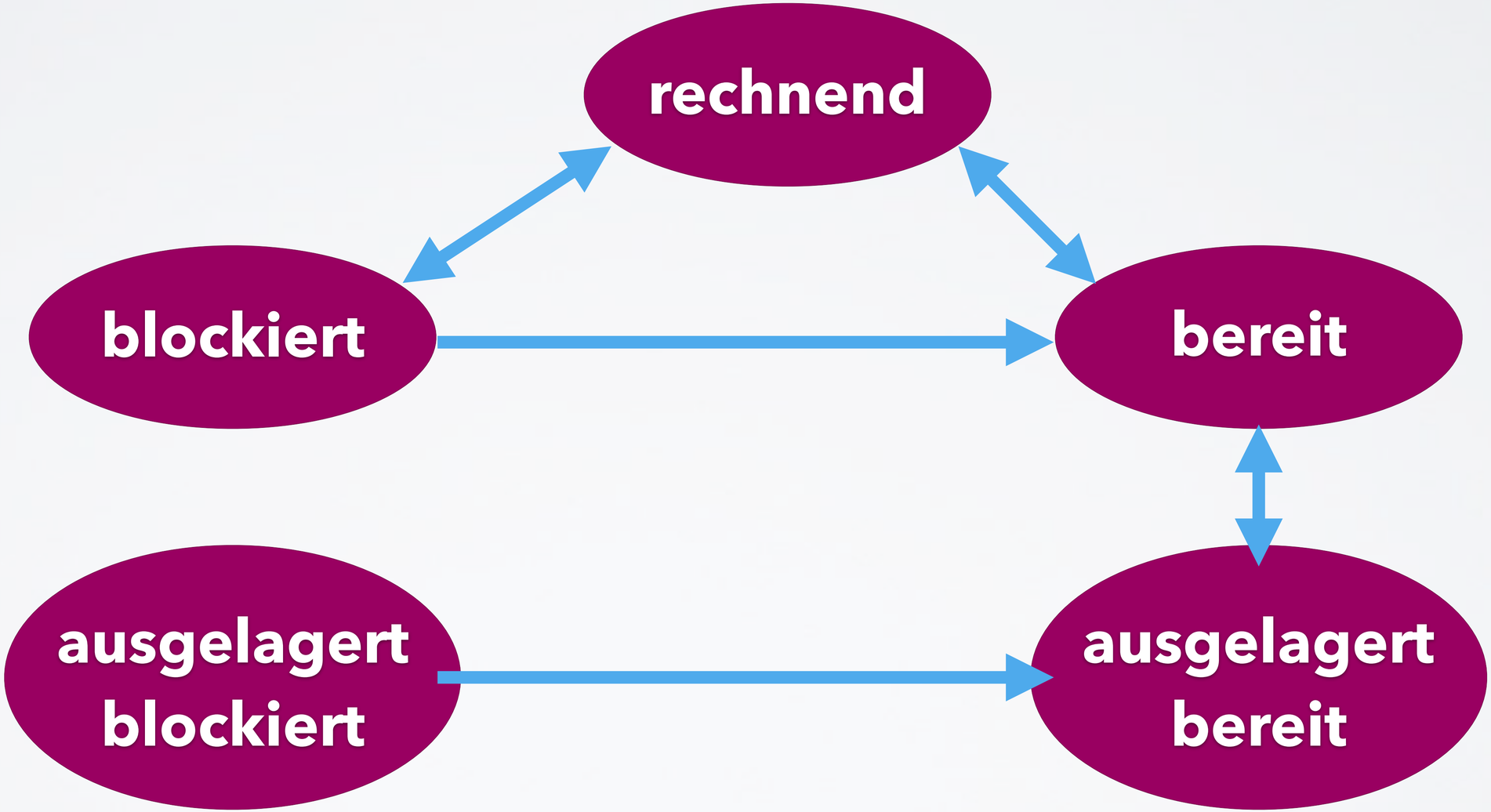


SWAPPING

Ein-/Auslagern ganzer Prozesse/Partitionen 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)

MEHREBENEN-SCHEDULING

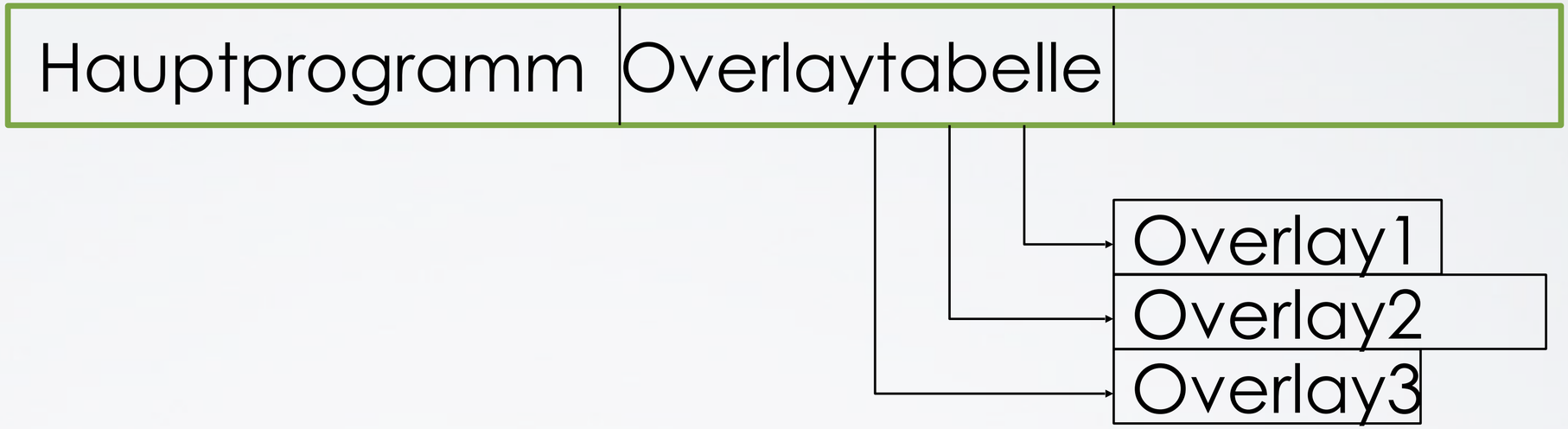


PARTITIONEN

Nachteile und Probleme:

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

"OVERLAYS"



Programmierer organisiert Programm in Stücke,
die nicht gleichzeitig im Hauptspeicher sein müssen

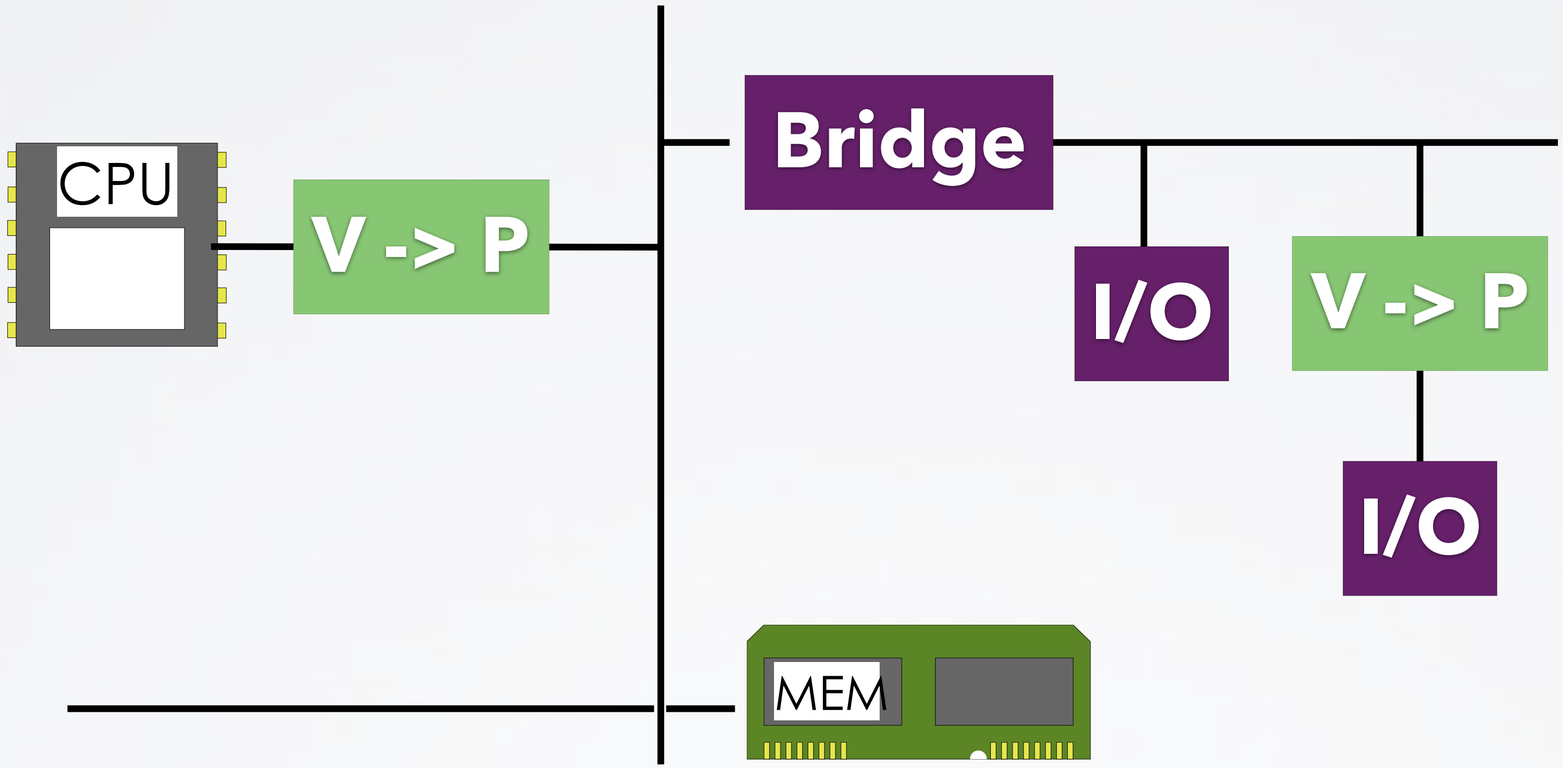
- Einführung
- Elementare Techniken
- **Virtueller Speicher (Paging)**
  - Anliegen - Begriffe - Vorgehen
  - Adressumsetzung, Hardware, Lazy Copying, …
  - Seitenersetzung (Arbeitsmengenmodell)
  - Speicherobjekte
  - Integration
- Caches

ADRESSRAUM

Menge direkt per load/store/transfer zugreifbarer Adressen, bzw. deren Inhalte

- Virtueller / Logischer AR
  dem laufenden Prozess sichtbar
- Physischer AR
  auf Maschinenebene “physisch” verwandt
  - Adressleitungen auf Speicher- und Peripheriebus
  - häufig statisch per HW interpretiert ??

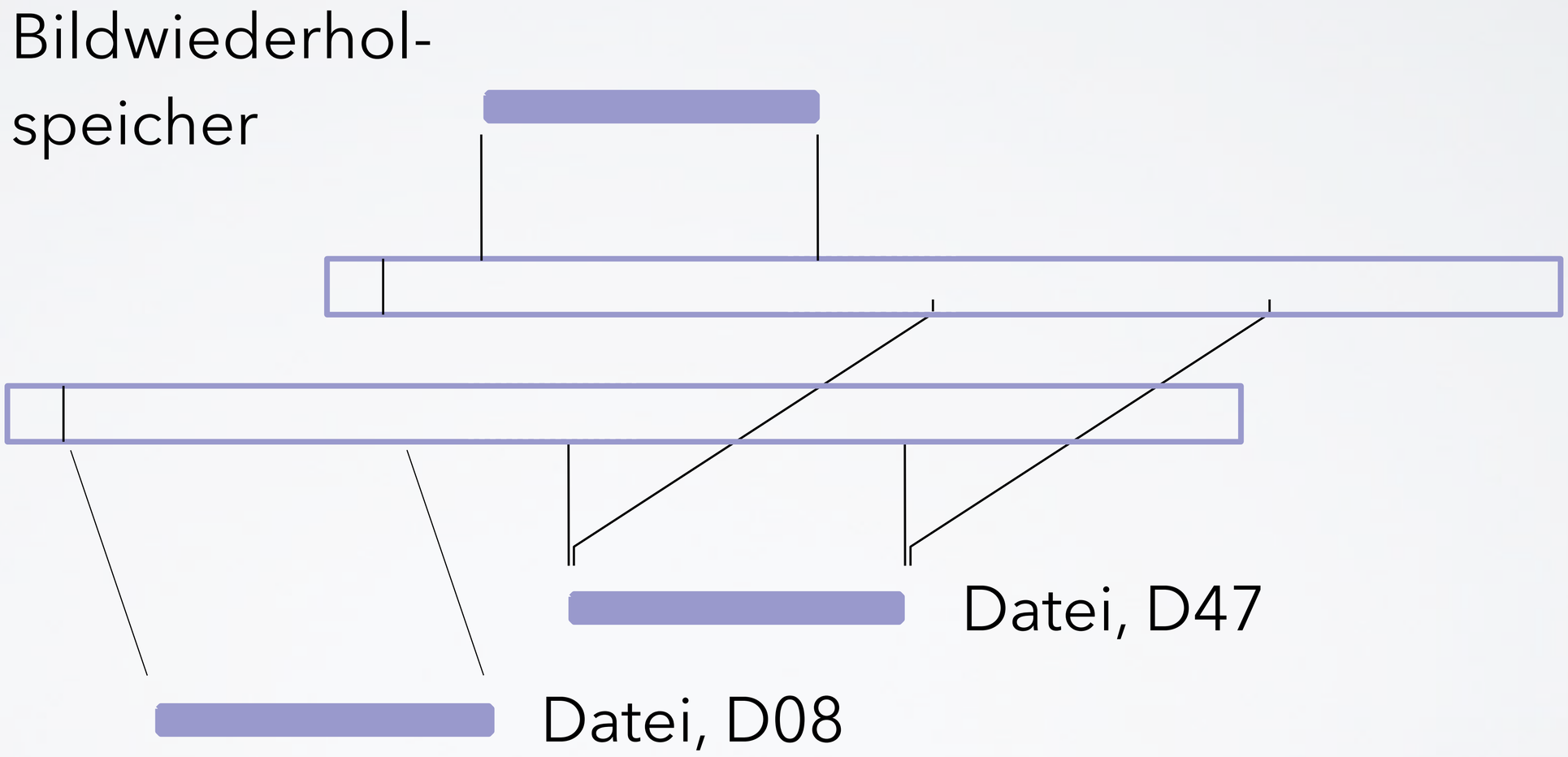
PHYSISCHE ADRESSRÄUME



NUTZUNG VIRT. ADRESSRÄUME

Unix-Prozesse (konventionell)

NUTZUNG VIRT. ADRESSRÄUME



REGIONEN /SPEICHEROBJEKTE

Aufbau von Adressräumen

- Logisch zusammenhängende Adressbereiche nennen wir Regionen
- Speicherobjekte werden Regionen zugeordnet („Mapping“)

ANFORDERUNGEN VAR

- Groß (soweit die Hardware zulässt, z. B. jeder bis zu 4 GB)
- Frei teilbar und nutzbar (“SHARED REGIONS”)
- 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
- Transparent für Programmierer (genauer: steuerbar, aber ohne Aufwand)

BEGRIFF

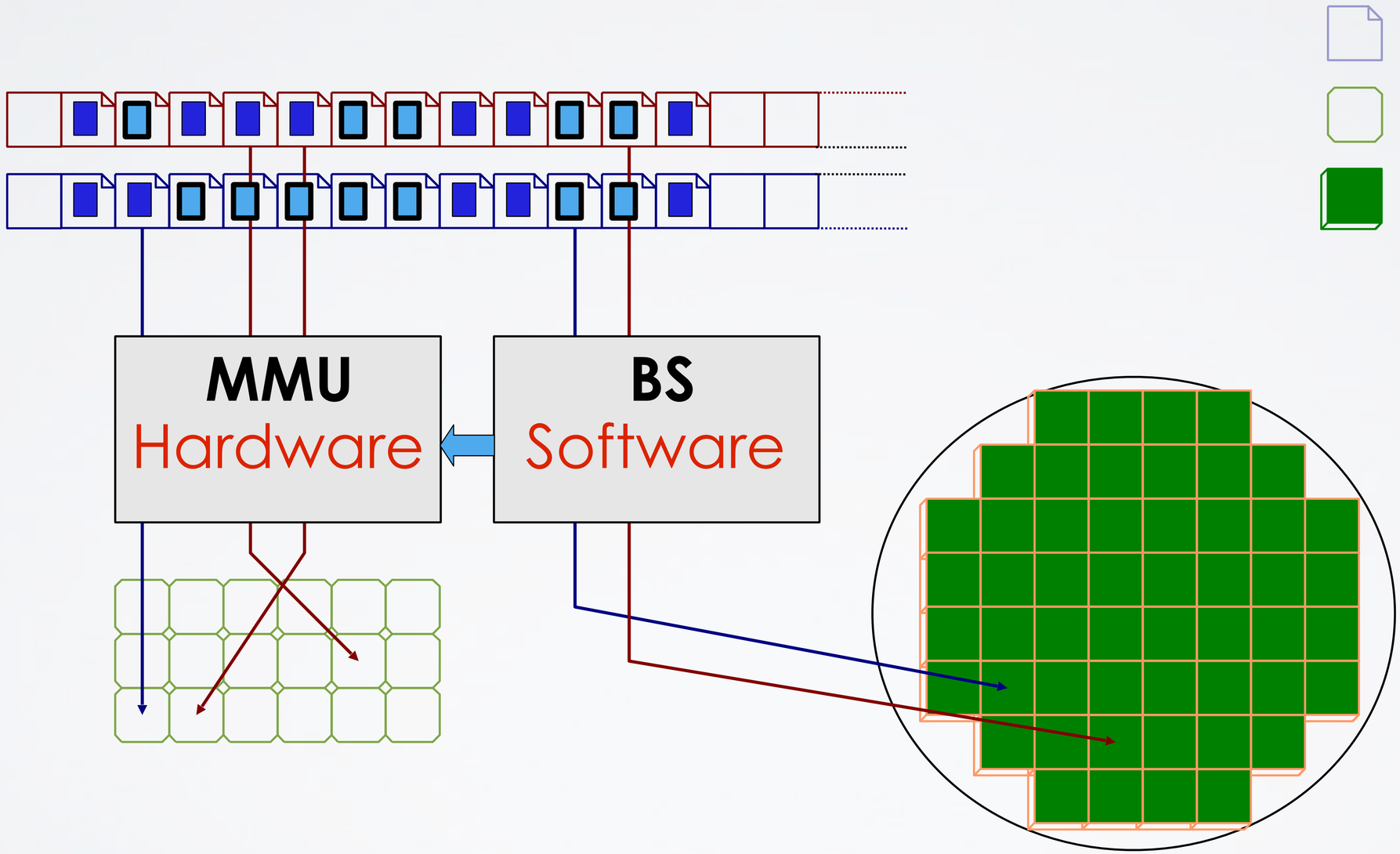
Virtueller Speicher

- gibt jedem Prozess einen eigenen, vom physischen Hauptspeicher unabhängigen “logischen” Adressraum,

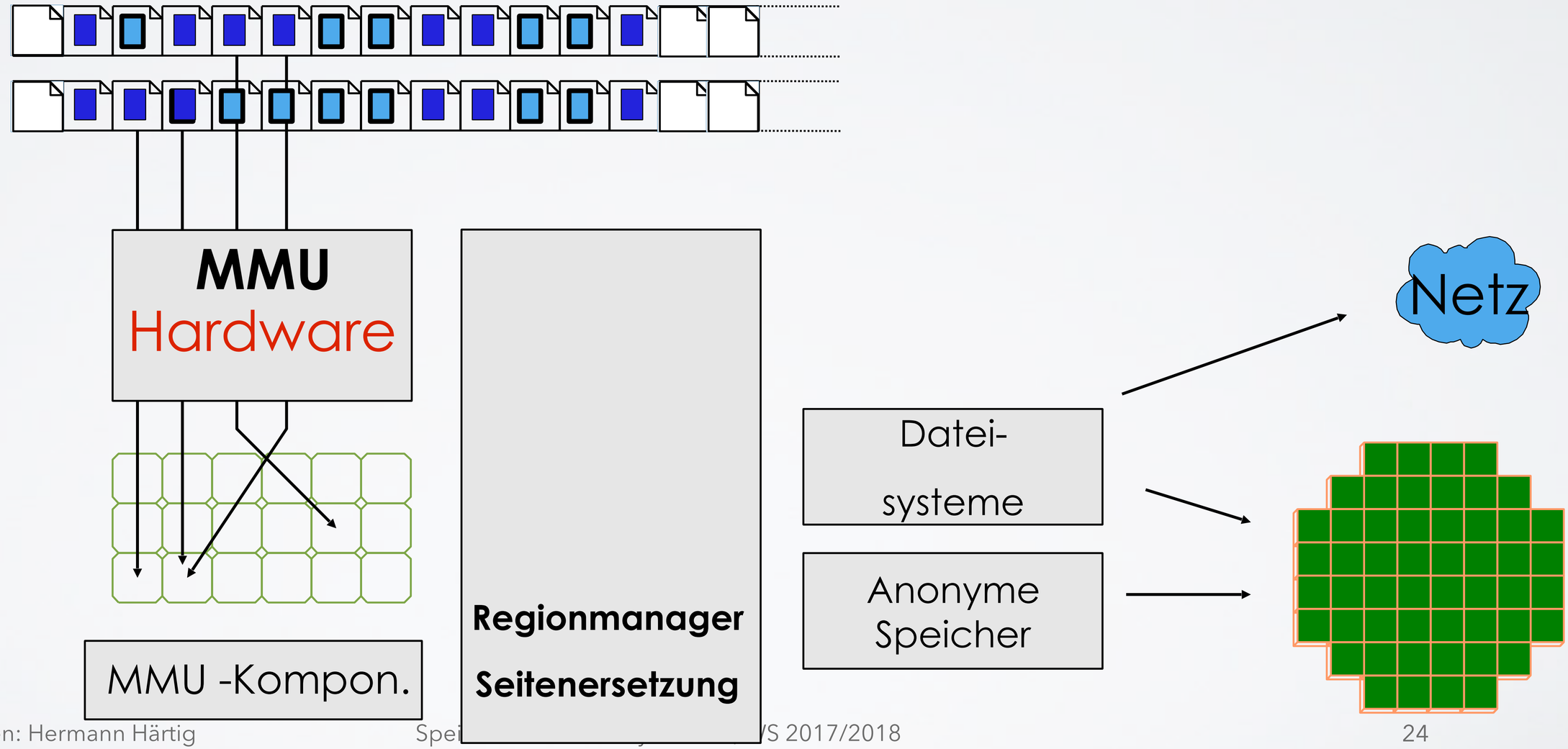
Basiert auf:

- einer Zerhackung Adressräumen/Hauptspeicher/… in Einheiten einheitlicher Größe (“*Seiten/Pages*”, “*Kacheln/Pageframes*”, “*Blöcke*”)
- einer Adressumsetzung durch Hardware und Betriebssystem
- der Nutzung eines externen Speichermediums
- einer Ein- und Auslagerung von Teilen des logischen Adressraumes eines Prozesses durch Betriebssystem

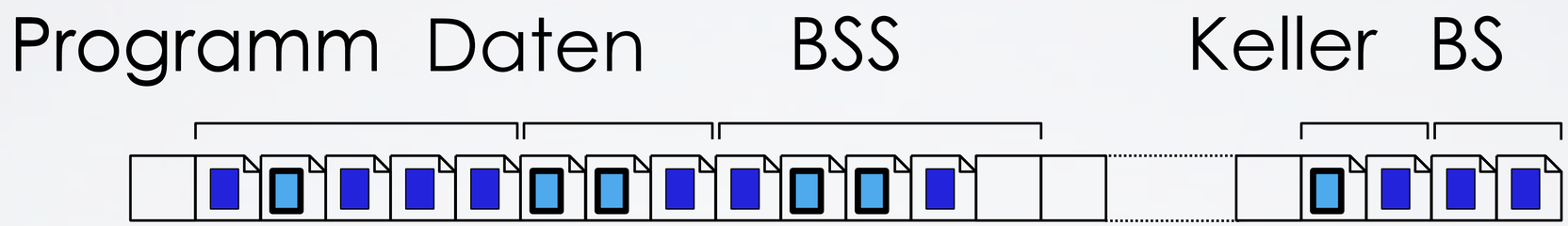
SEITEN, KACHELN, BLÖCKE



DAS ZUSAMMENSPIEL



EINE DENKBARE SITUATION



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

PRINZIP DER LOKALITÄT

Beobachtung

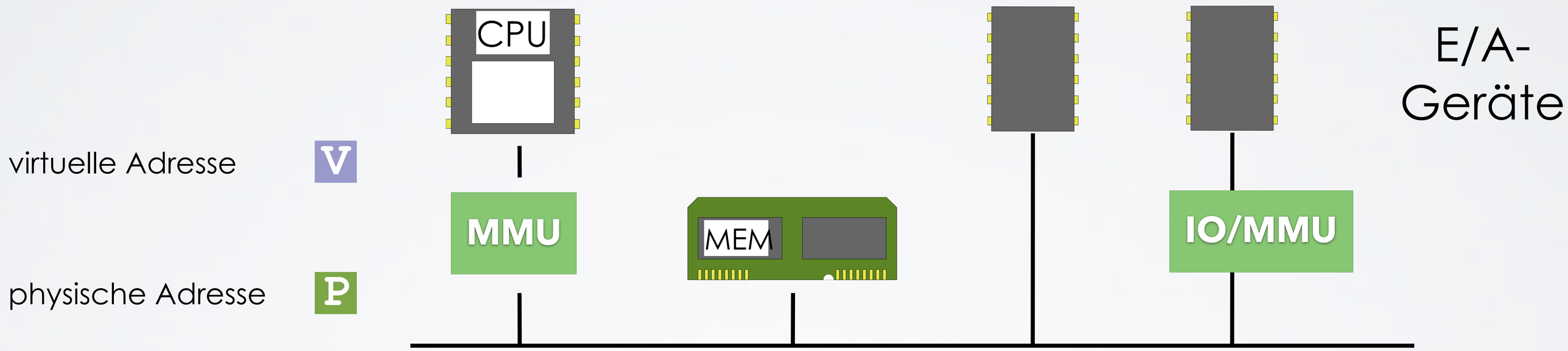
- Der von einem Prozess innerhalb eines bestimmten Zeitintervalls benötigte Teil seines Adressraumes verändert sich nur mehr oder weniger langsam.

Ursachen

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

- Virtueller Speicher (Paging)
  - Anliegen - Begriffe - Vorgehen
  - Adressumsetzung, Hardware, Lazy Copying, …
  - Betriebsmittel Hauptspeicher: Seitenersetzung (Arbeitsmengenmodell)
  - Speicherobjekte
  - Integration der Einzelkomponenten

HW: ADDRESSUMSETZUNG



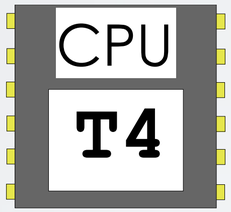
MEMORY MANAGEMENT UNIT

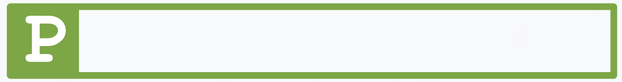
Aufgaben:

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

PRINZIPIELLE ARBEITSWEISE MMU

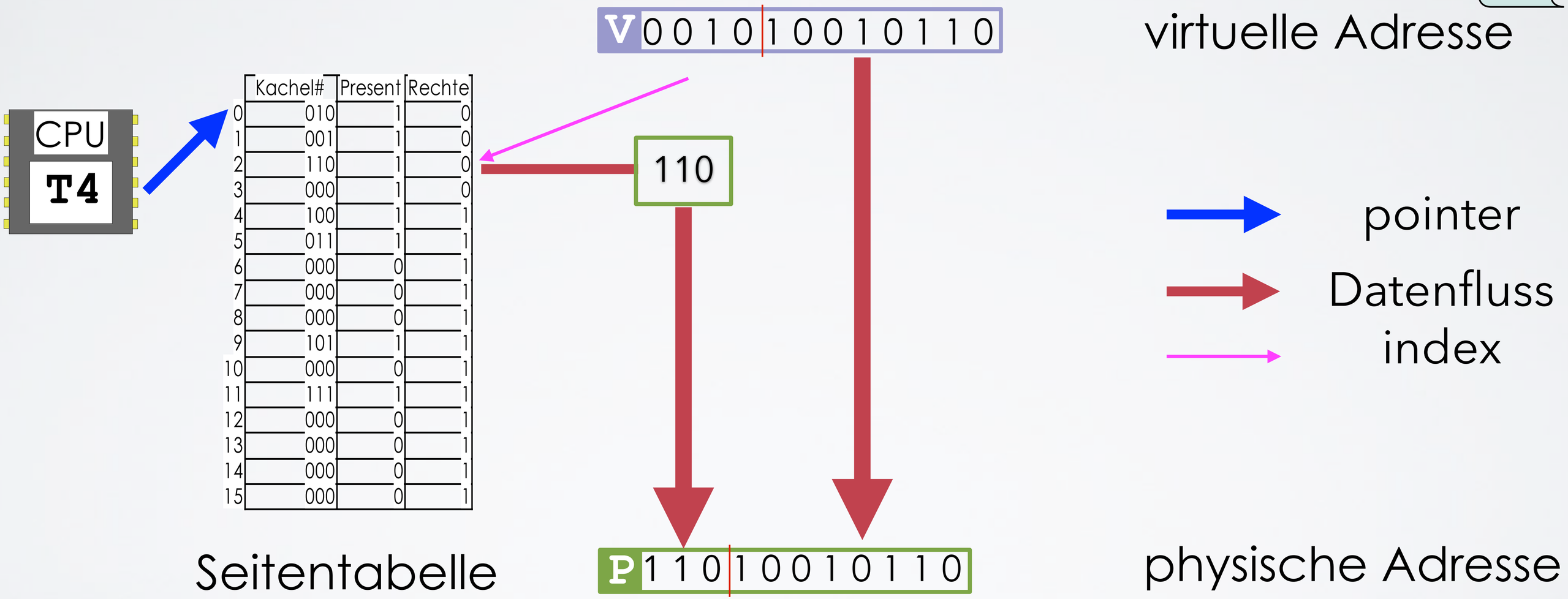
virtuelle Adresse



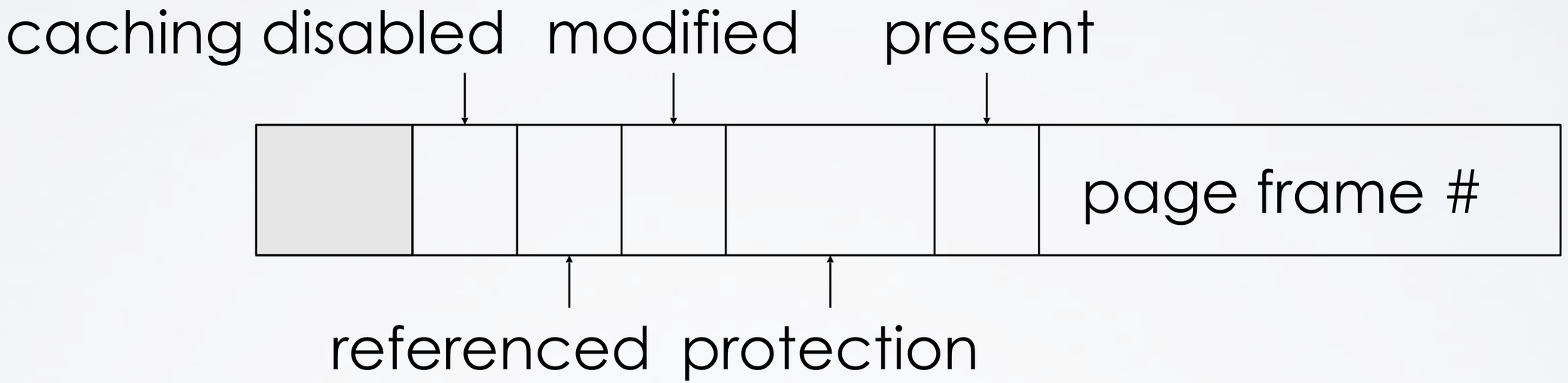


physische Adresse

PRINZIPIELLE ARBEITSWEISE MMU



PAGE TABLE ENTRY



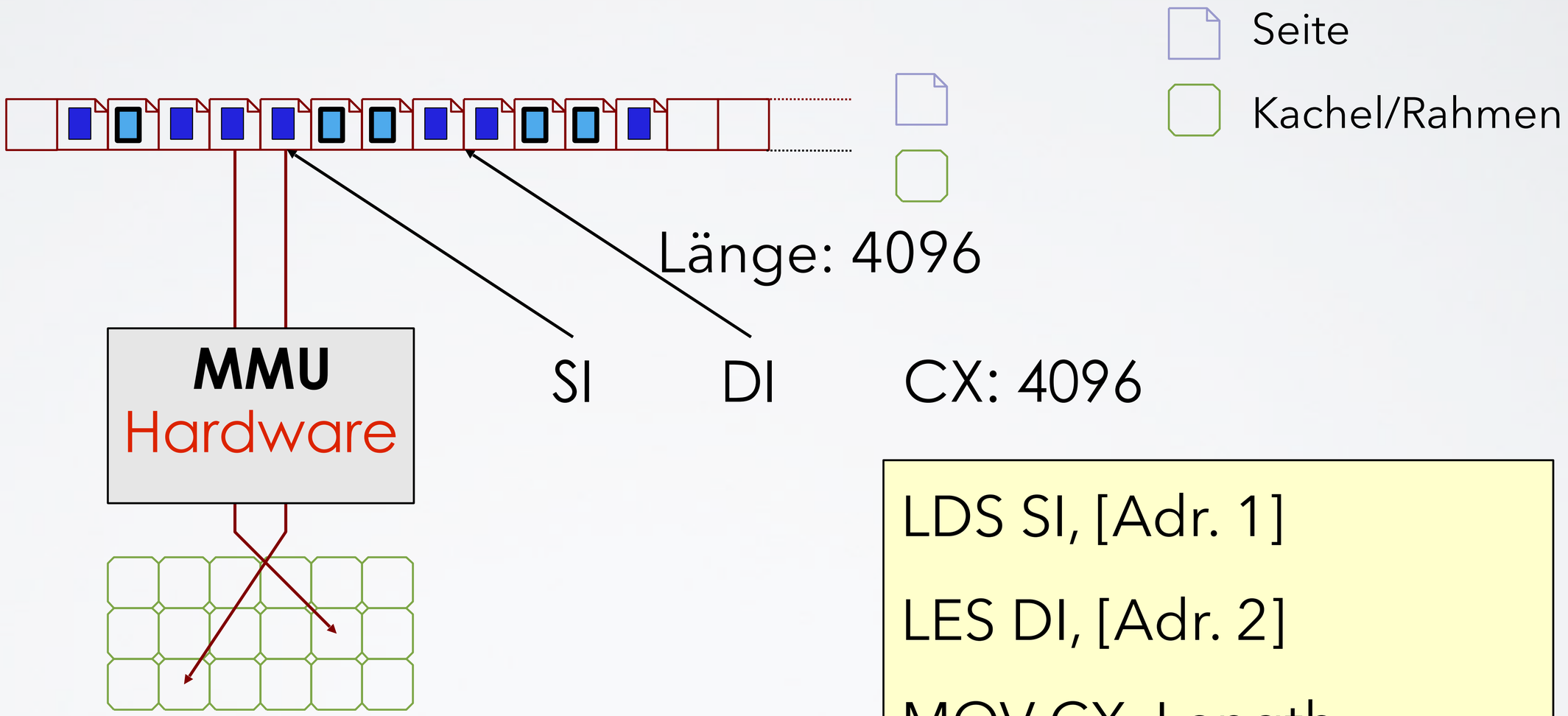
SEITEN ATTRIBUTE

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

ABLAUF SEITENFEHLER

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

BEISPIEL




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

MMU PROBLEM 1: GRÖßE

Beispiel (sehr alter Rechner oder eingebettetes System)

- 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)
- Seitentabelle pro AR: 2^{20} *4B → 4 MB
  Bei 32 Prozessen 32*4 MB für die Seitentabellen !

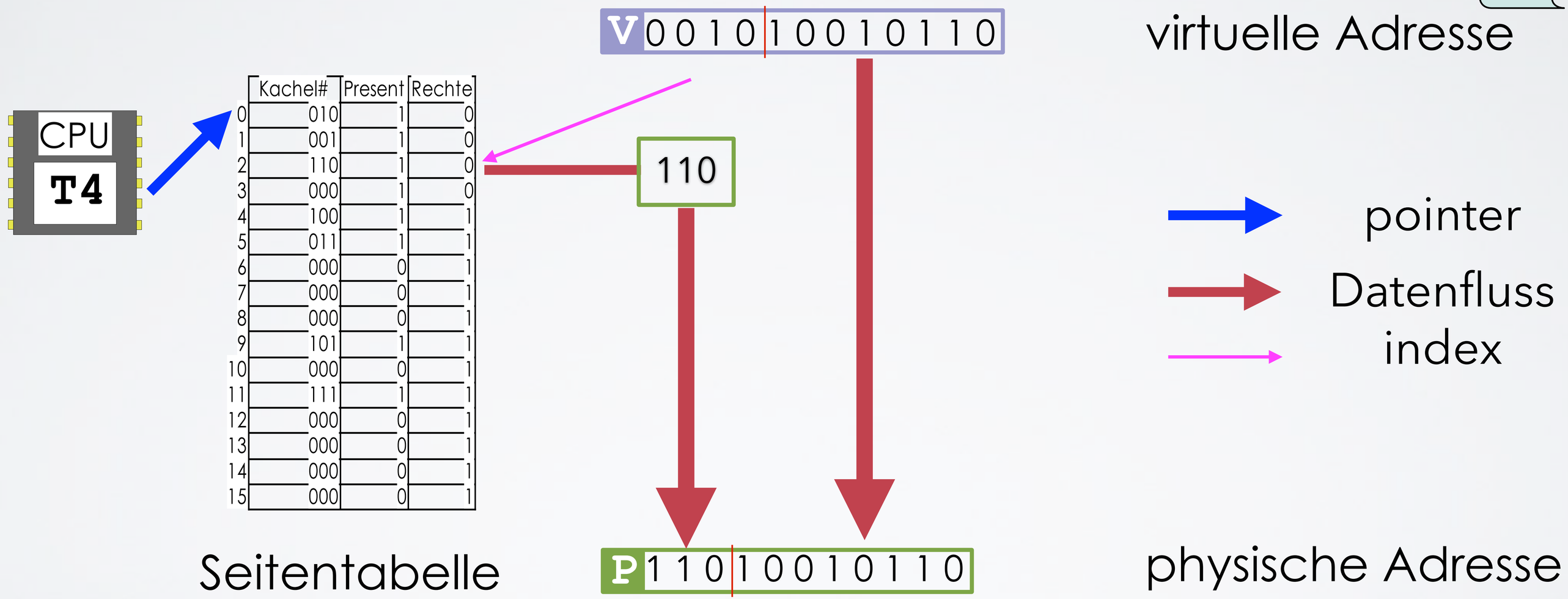
MMU PROBLEM 2: ZEIT

Geschwindigkeit der Abbildung – Beispiel

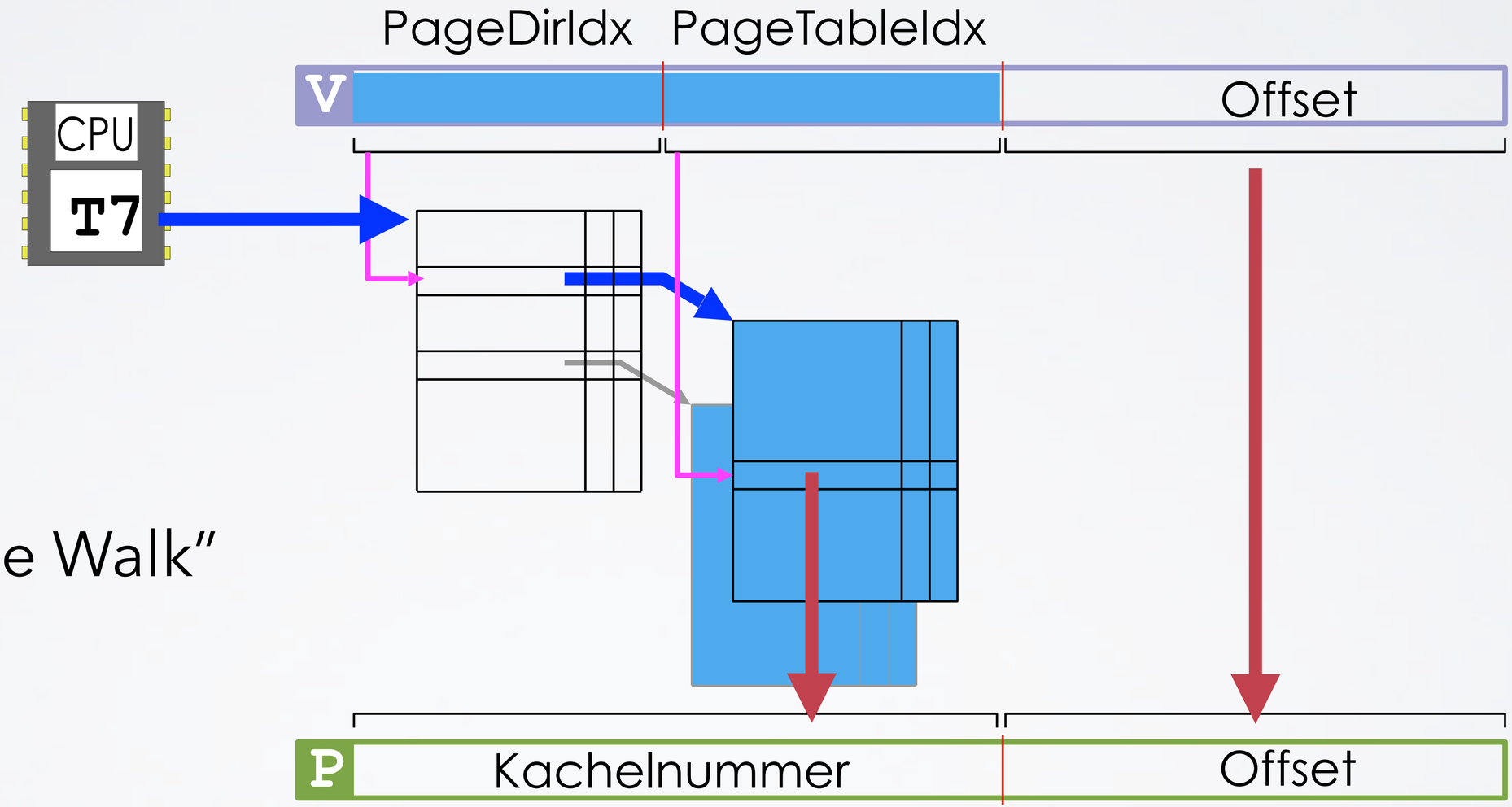
- CPU-Takt : 1 GHz → 2 Instruktionen in 1ns
  bis zu 4 Speicherzugriffe
- Speichertakt: 256 MHz → 4ns (... 70ns)

PROBLEM 1

TB4



PROBLEM 1: BAUMSTRUKTURIERTE

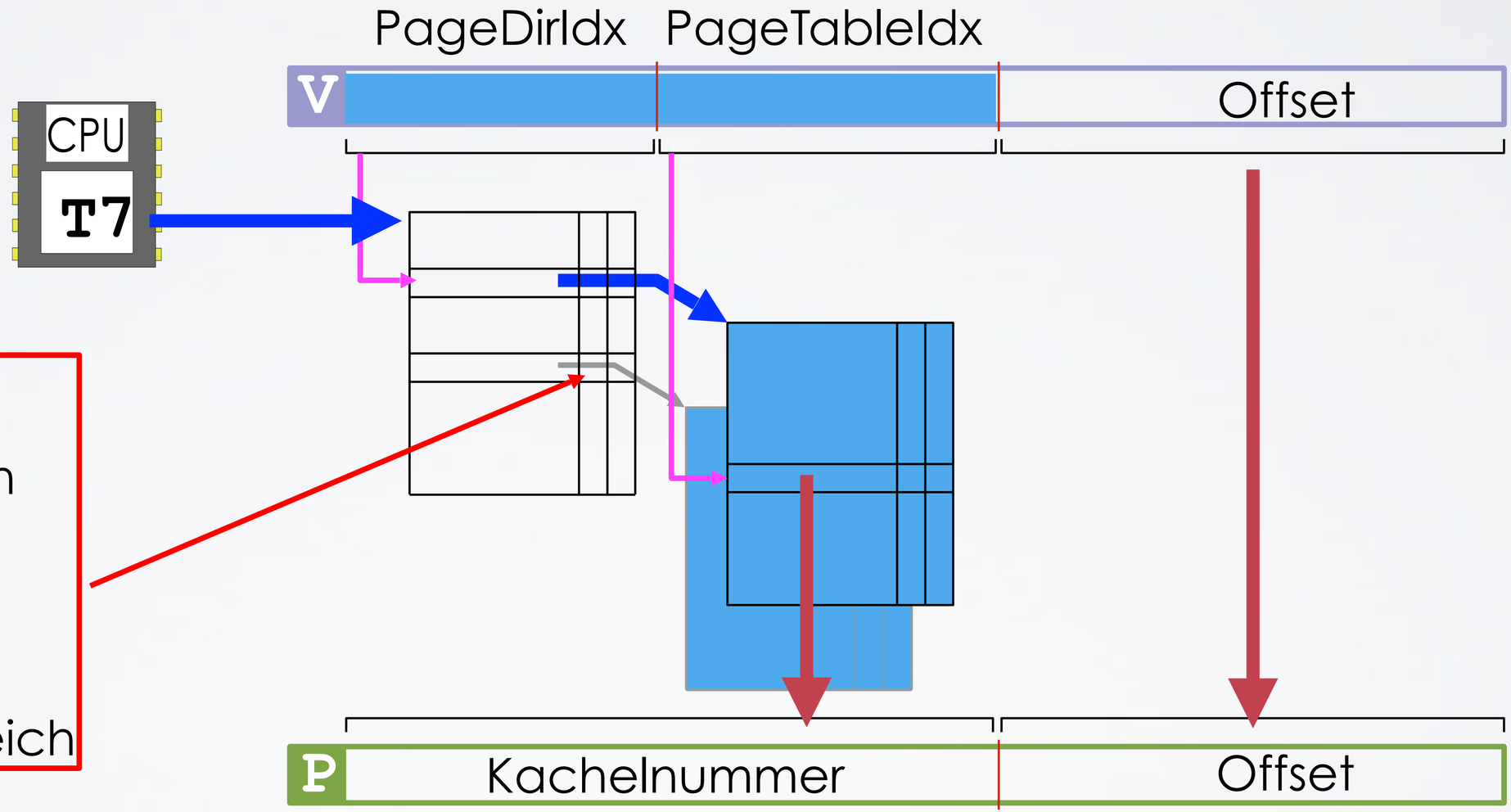


"Page Table Walk"

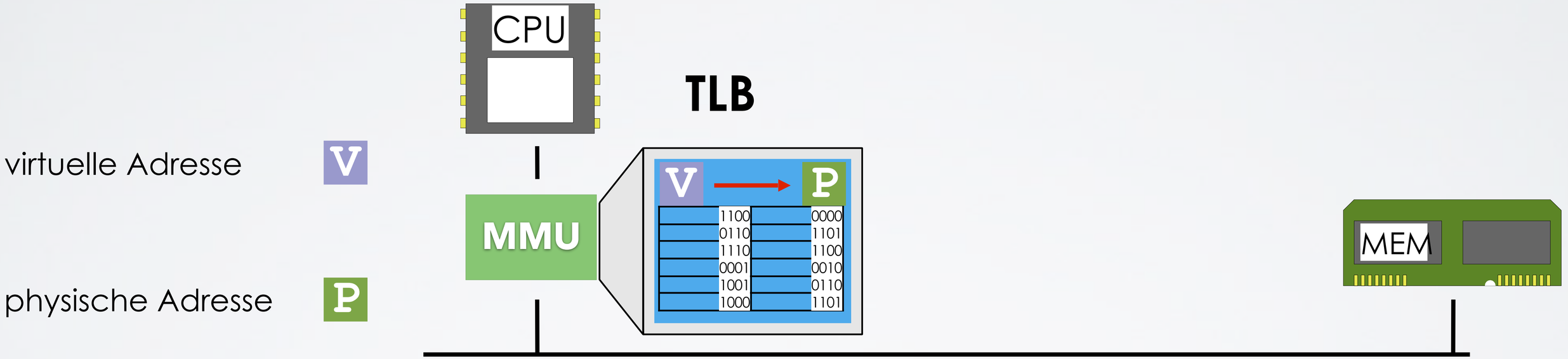
EIGENSCHAFTEN

- 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“)
  Seiten und größere Bereiche in mehreren Adressräumen gleichzeitig

PROBLEM 1: BAUMSTRUKTURIERTE



PROBLEM 2: ZEIT



TLB: Translation Lookaside Buffer

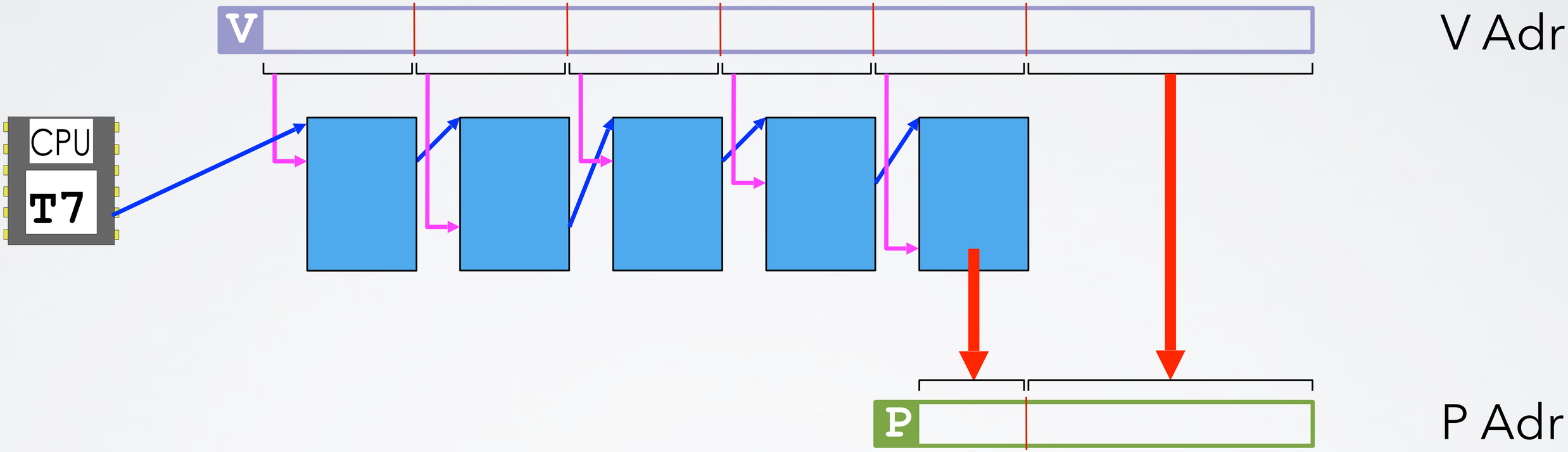
- schneller assoziativer Speicher für ermittelte Abbildungen
- wird vor page Table Walk durchsucht

PAGETABLEWALK PER SW

- Zugriff nur über TLB, bei TLB-Fehler („TLB-miss“) → Seitenfehler
  Seitenfehlerbehandlung in SW lädt TLB neu
  - Vorteil: total flexibel
  - Nachteil: häufigere SW-Seitenfehlerbehandlung

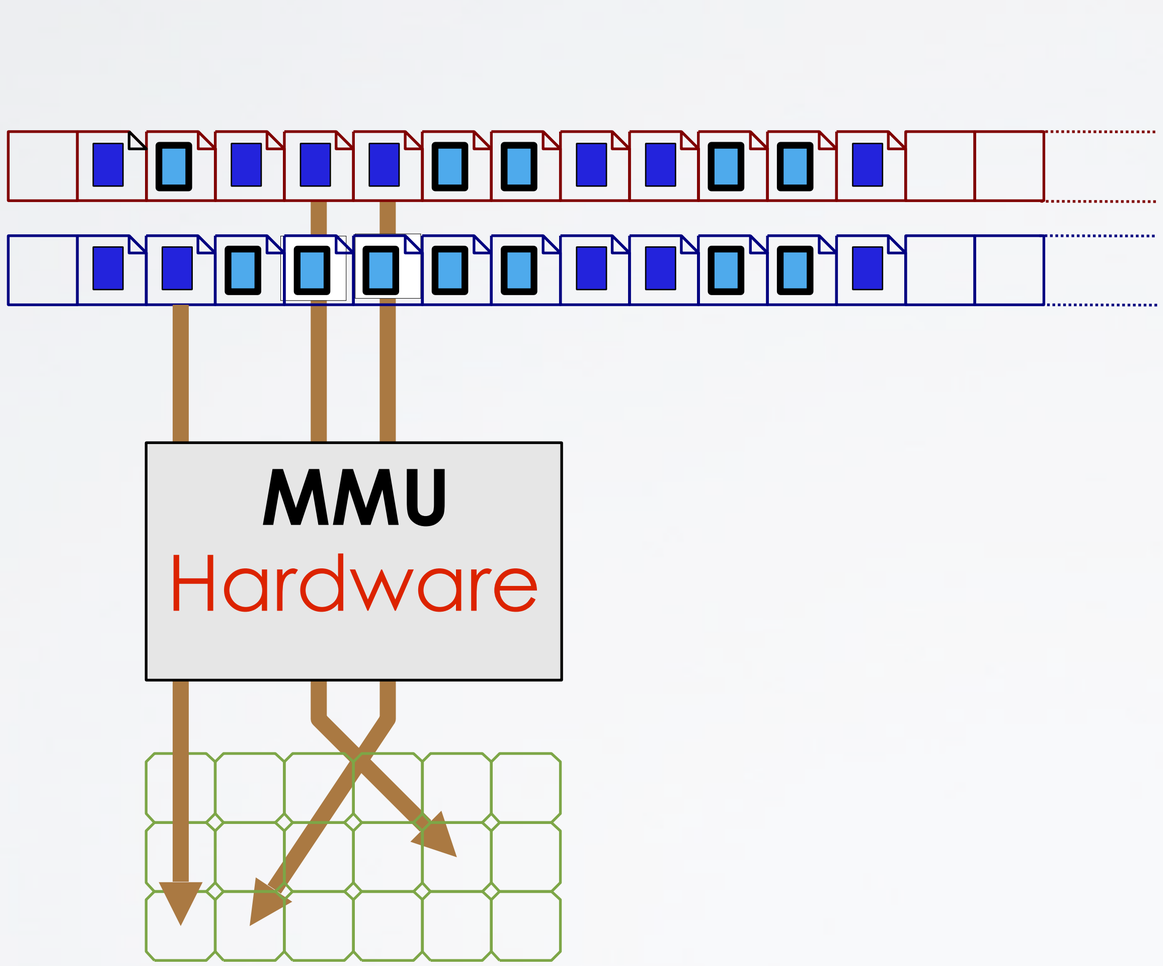
- z. B. Alpha, MIPS

NOCHMAL PROBLEM 1



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

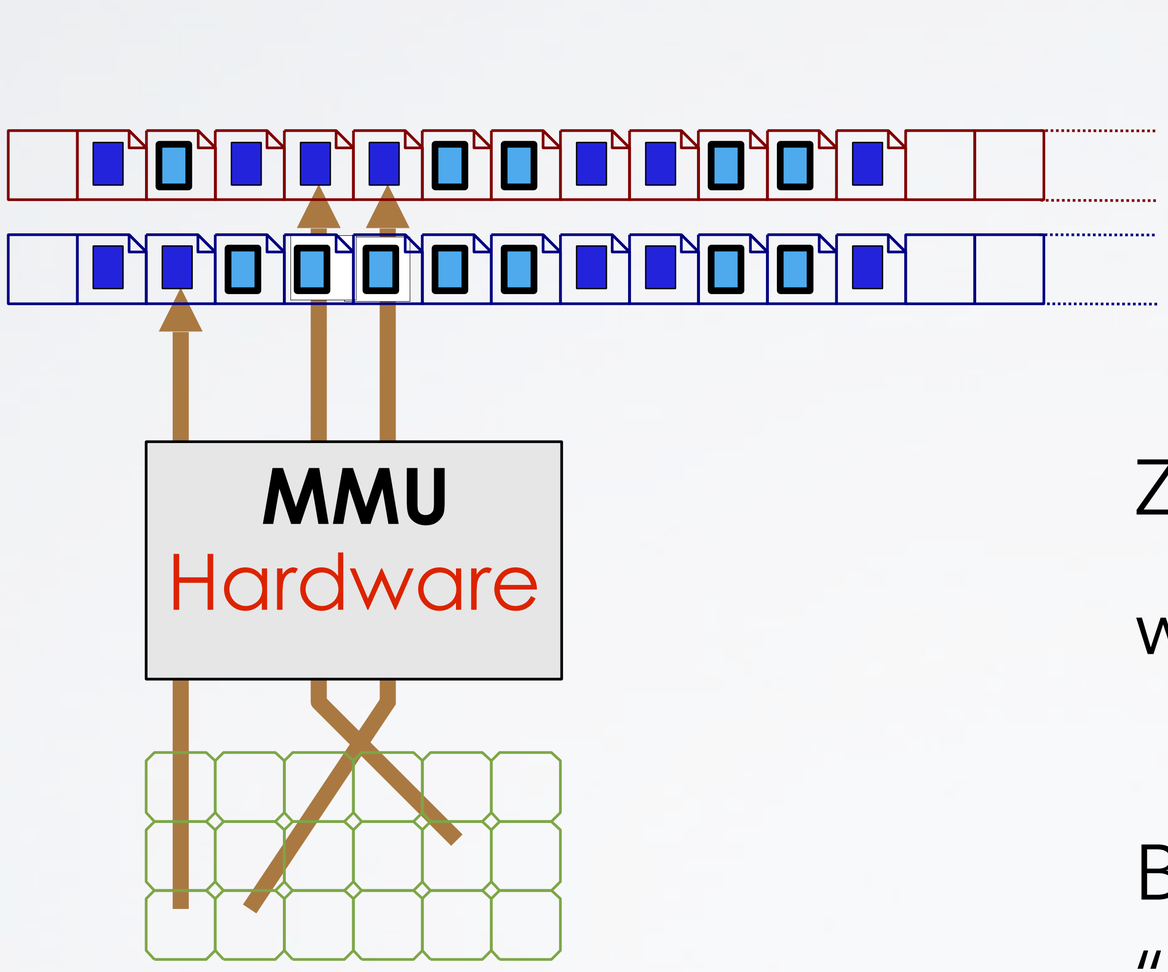
SEITENTABELLEN



Seite

Kachel/Rahmen

INVERSE SEITENTABELLEN



Seite

Kachel/Rahmen

Zu jeder Kachel:

welche virtuelle Adresse

Bei Zugriff auf virtuelle:
“durchsuchen” der Tabelle

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

Kachel#

	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	
		...	

INVERSE SEITENTABELLE

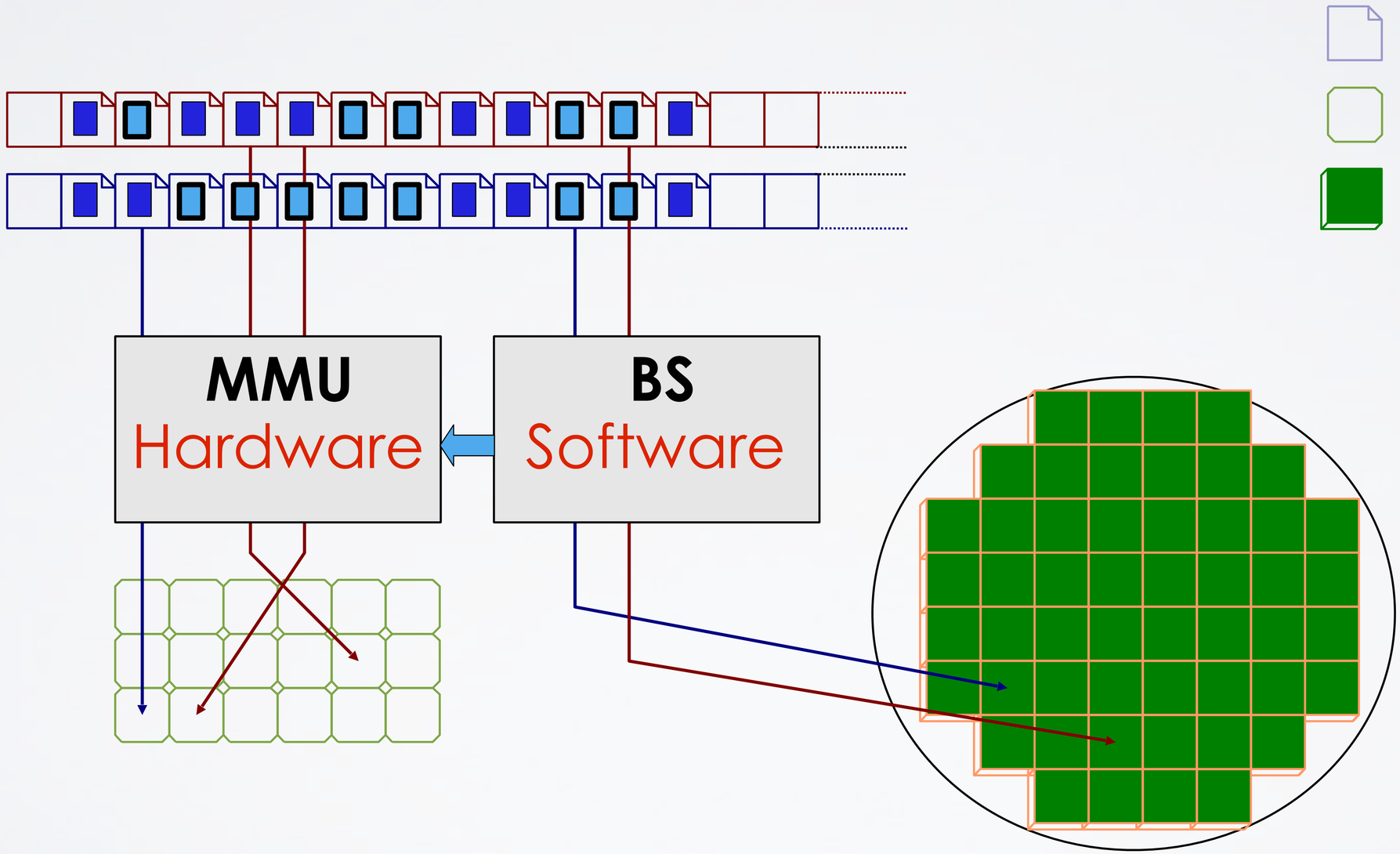
Grundidee

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

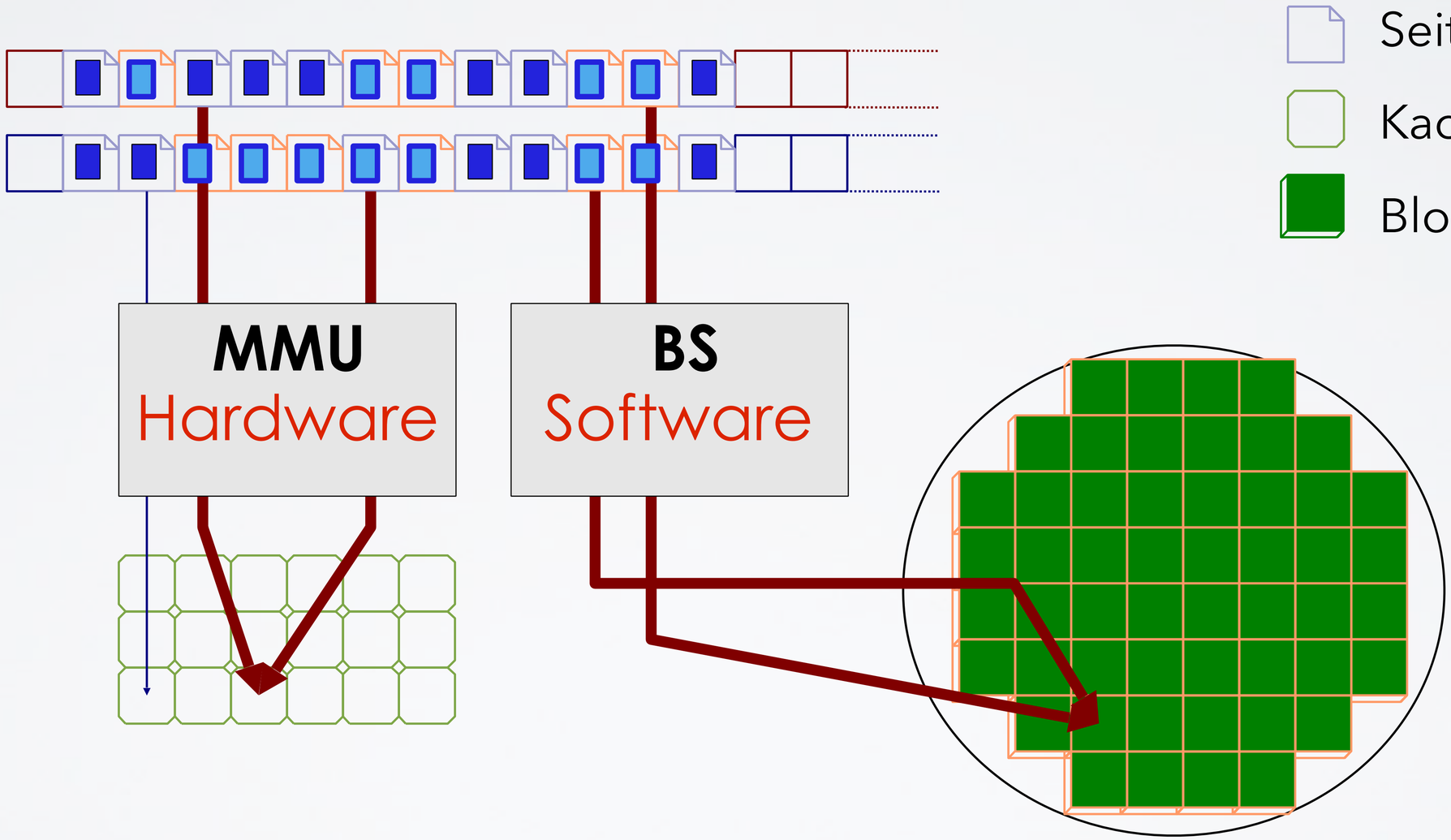
Implementierung als Hash-Tabellen (Hashed Page Tables)

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

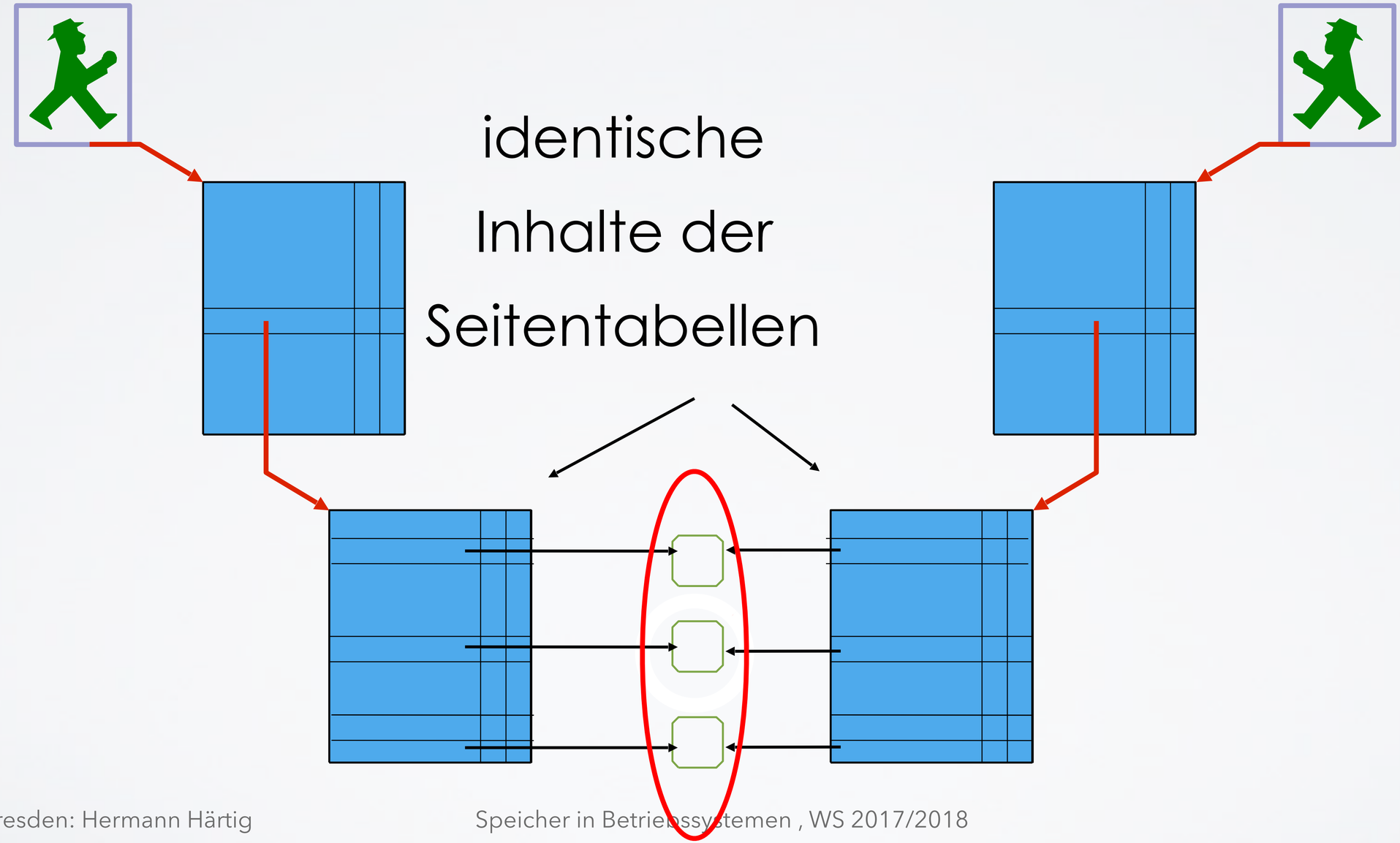
SEITEN, KACHELN, BLÖCKE



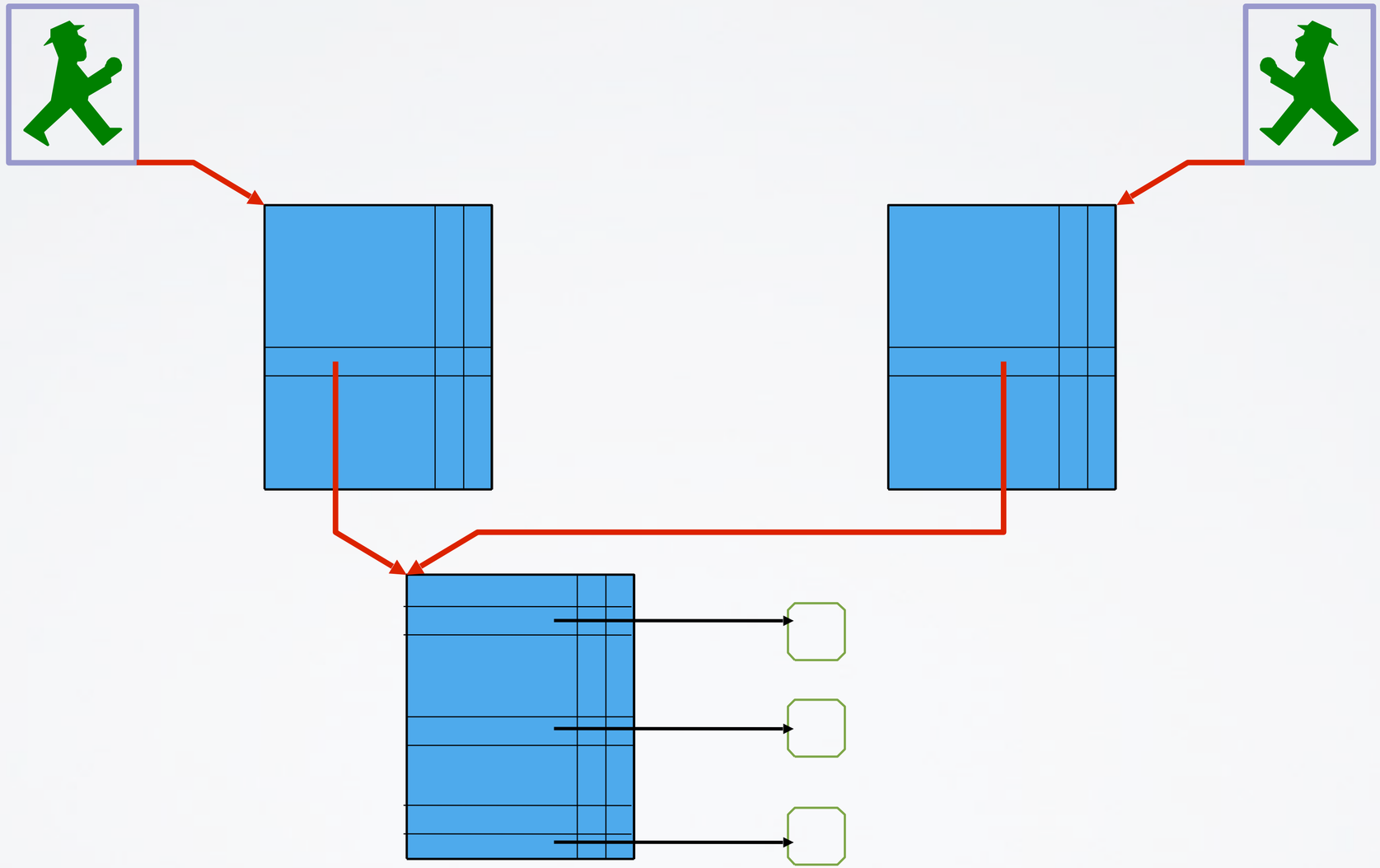
SHARING



SHARING BEI MEHRSTUFIGEN



SHARING BEI MEHRSTUFIGEN



LAZY COPYING

Verzögertes Kopieren (lazy copying, copy on write)

„Kopieren“:

- Eintragen des zu kopierenden Bereichs an Zieladresse
- beide gegen Schreiben schützen

LAZY COPYING

Verzögertes Kopieren (lazy copying, copy on write)

Entkoppeln:

- beim ersten schreibenden Zugriff → Seitenfehler
- Behandlung:
  - neue Kachel allokieren
  - physisch kopieren
  - neue Kachel ohne Schreibschutz in Seitentabelle eintragen

(Echtes ./. Verzögertes) Kopieren

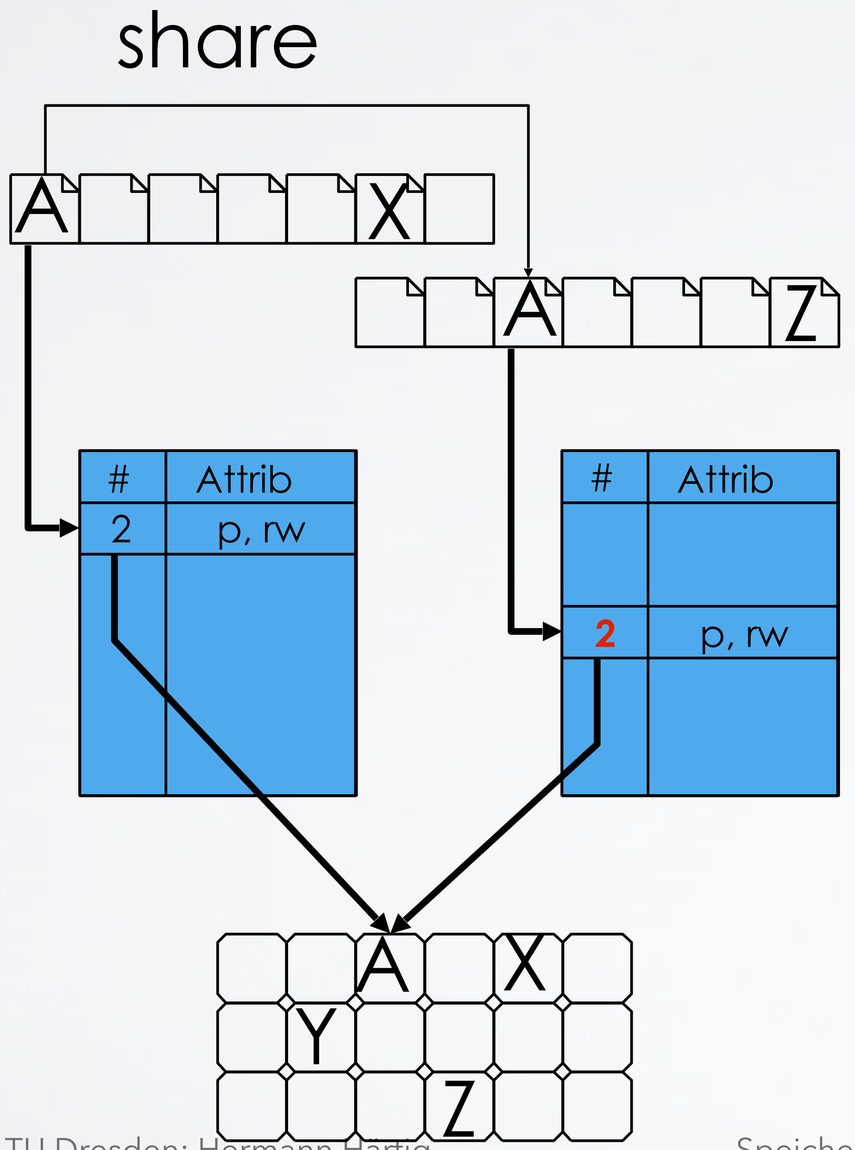
Einsatz:

- UNIX fork (lazy)
- effiziente lange Botschaften (lazy)
- Rücksetzpunkte (lazy)
- gemeinsame Nutzung von Daten/Programmen (echt)

BEISPIELE

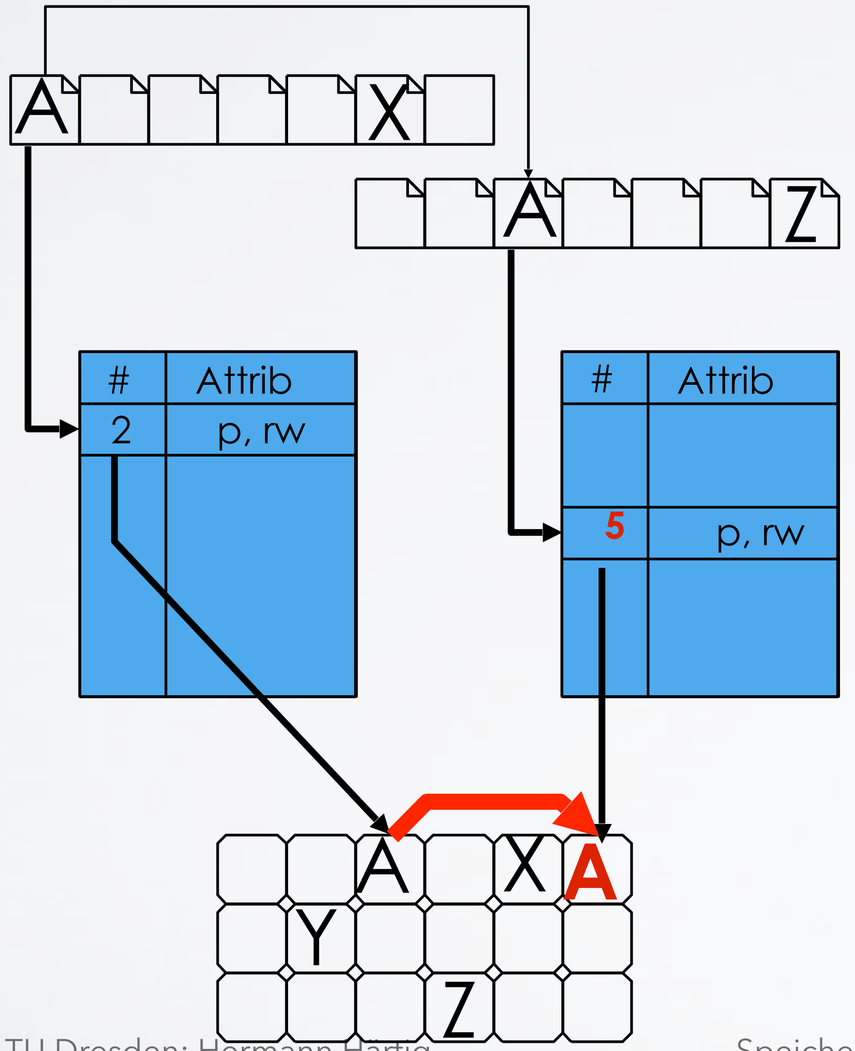
- Teilen (gemeinsames Nutzen, echtes Sharing)
- Echte Kopieren
- Lazy Copying

ECHTES SHARING



neuen MMU Eintrag an Zieladresse

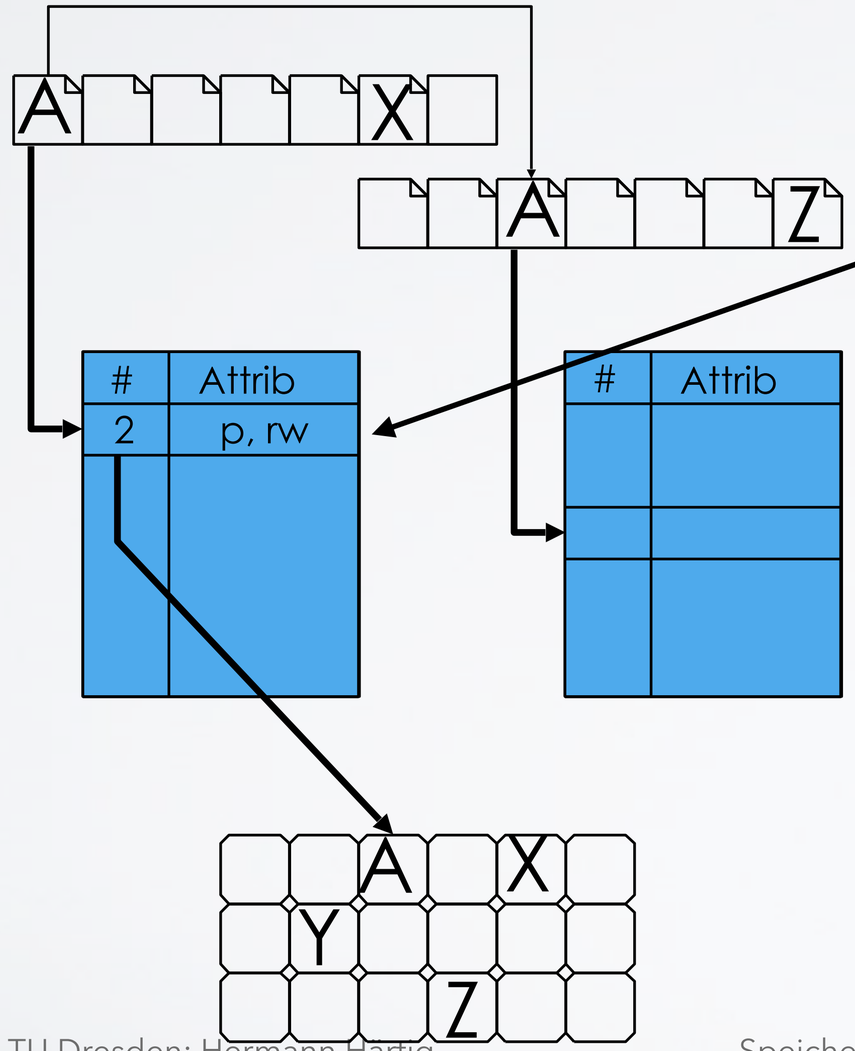
ECHTES KOPIEREN



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

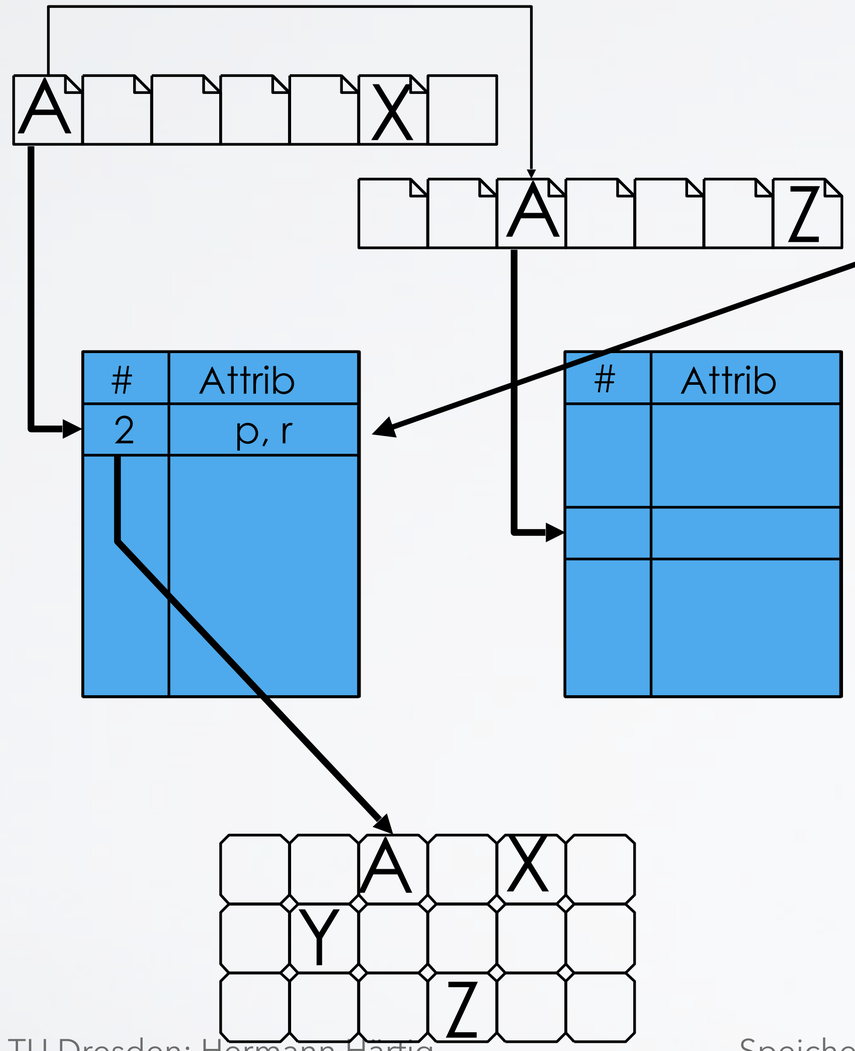
copy

VERZÖGERTES KOPIEREN



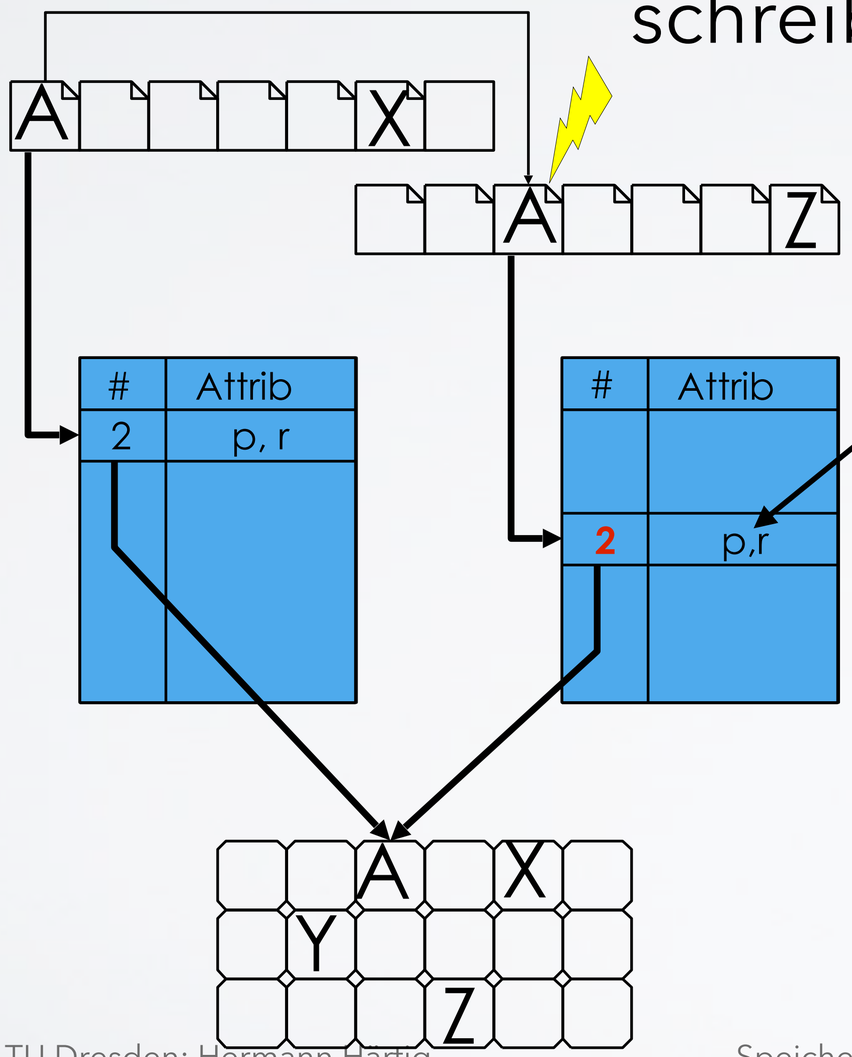
- Schreibrecht entziehen
-
-

VERZÖGERTES KOPIEREN



- Schreibrecht entziehen
-
-

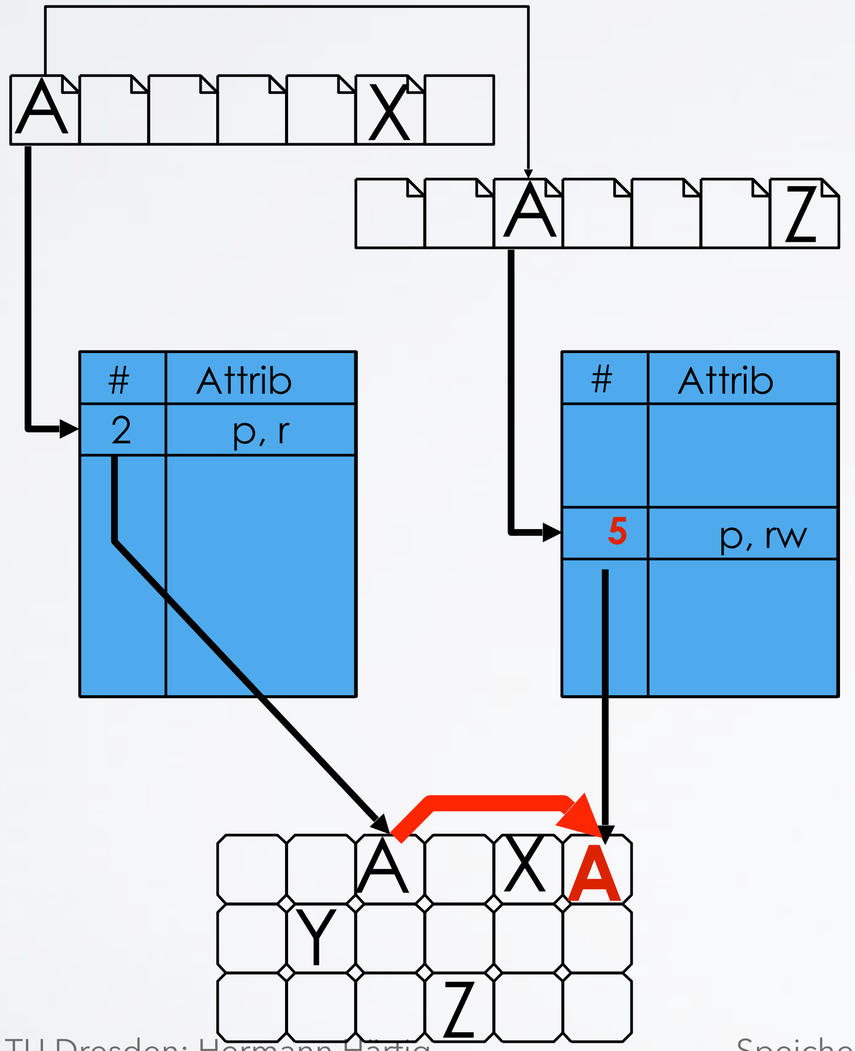
VERZÖGERTES KOPIEREN



schreibender Zugriff

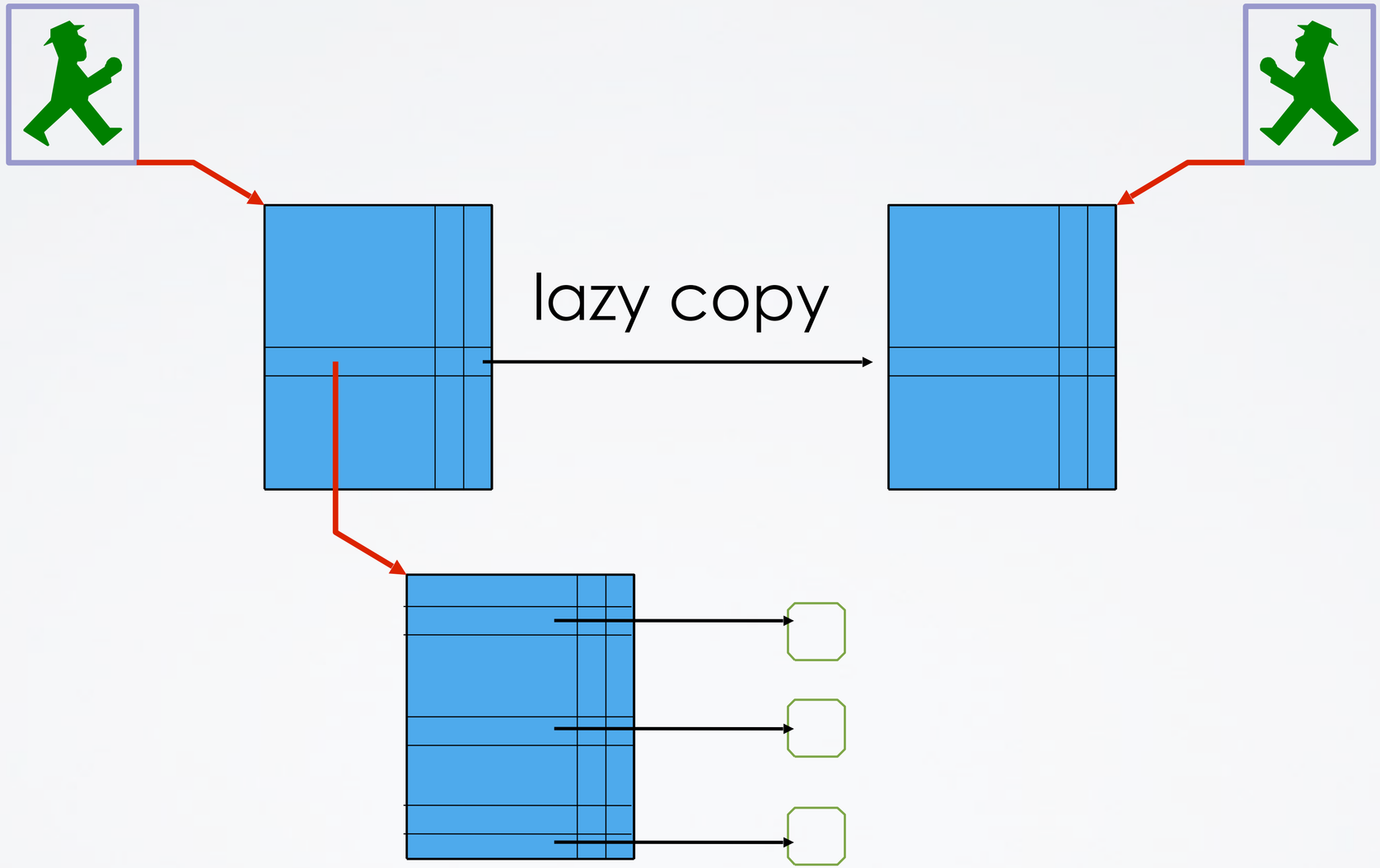
- Schreibrecht entziehen
- PTE an Zieladresse
-
-

VERZÖGERTES KOPIEREN

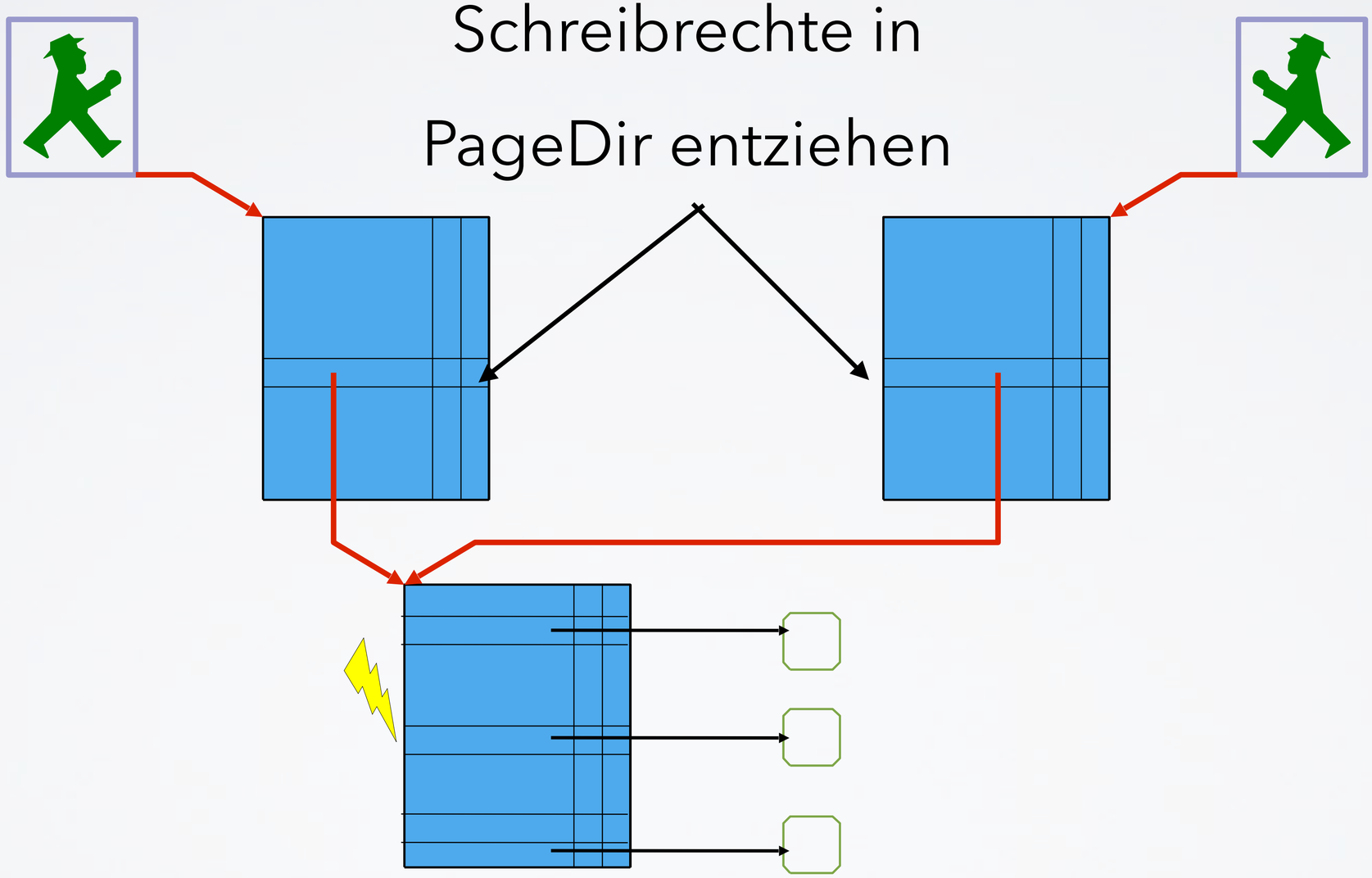


-
- Inhalt in neue Kachel kopieren
- neuen Eintrag in die Seitentabelle für Zieladresse

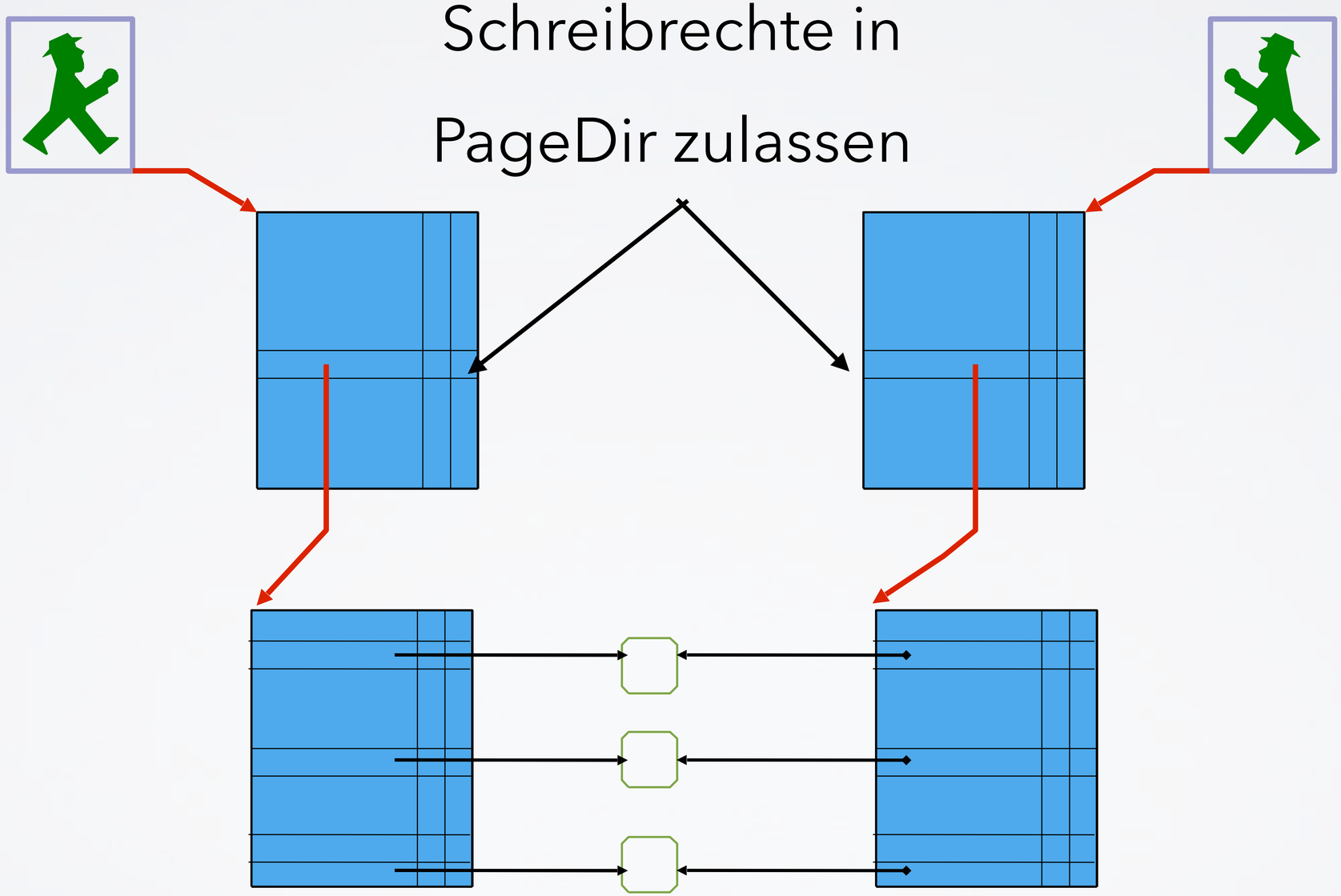
VERZÖGERTES KOPIEREN GROßER



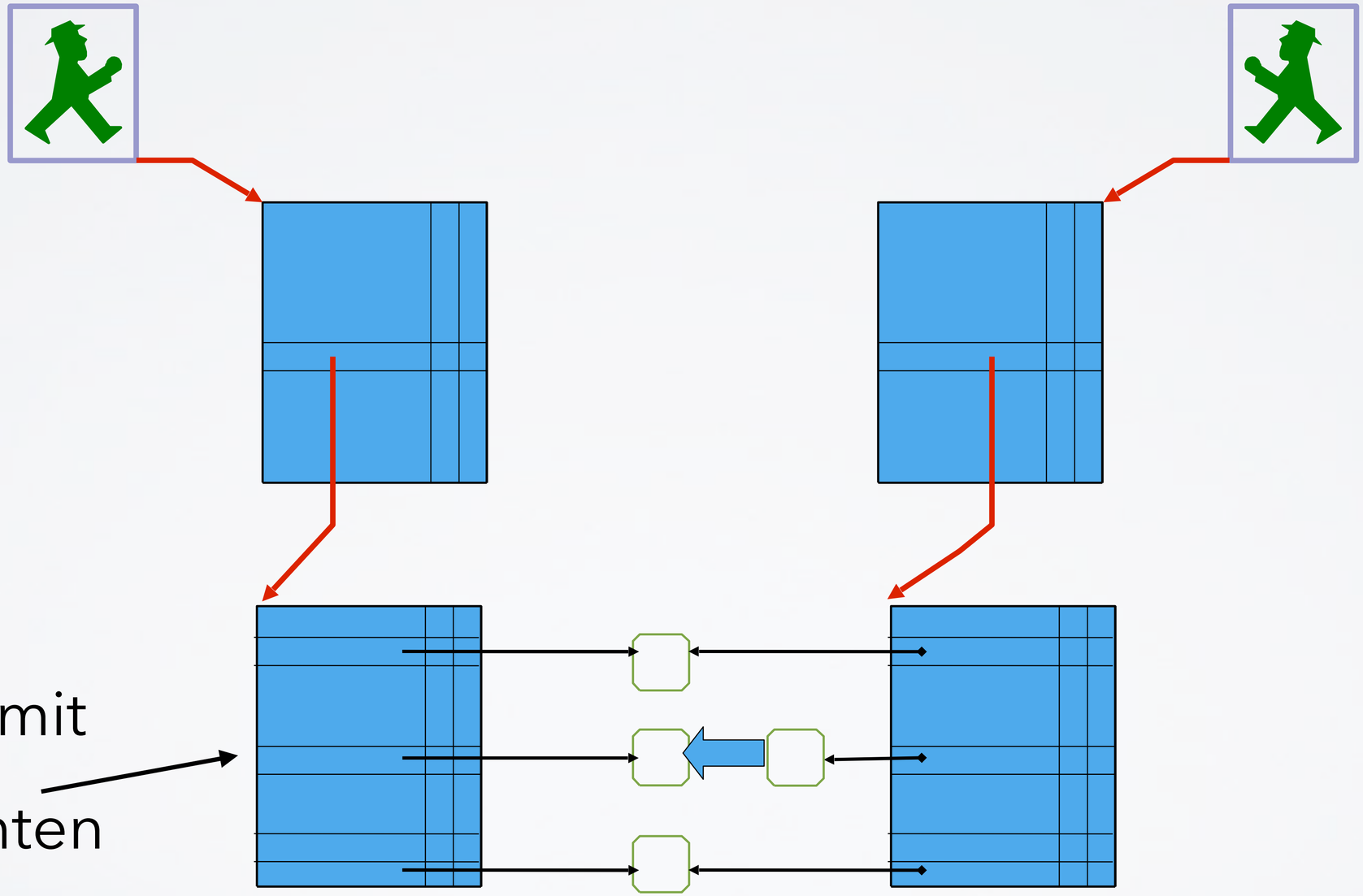
VERZÖGERTES KOPIEREN GROßER



VERZÖGERTES KOPIEREN GROßER

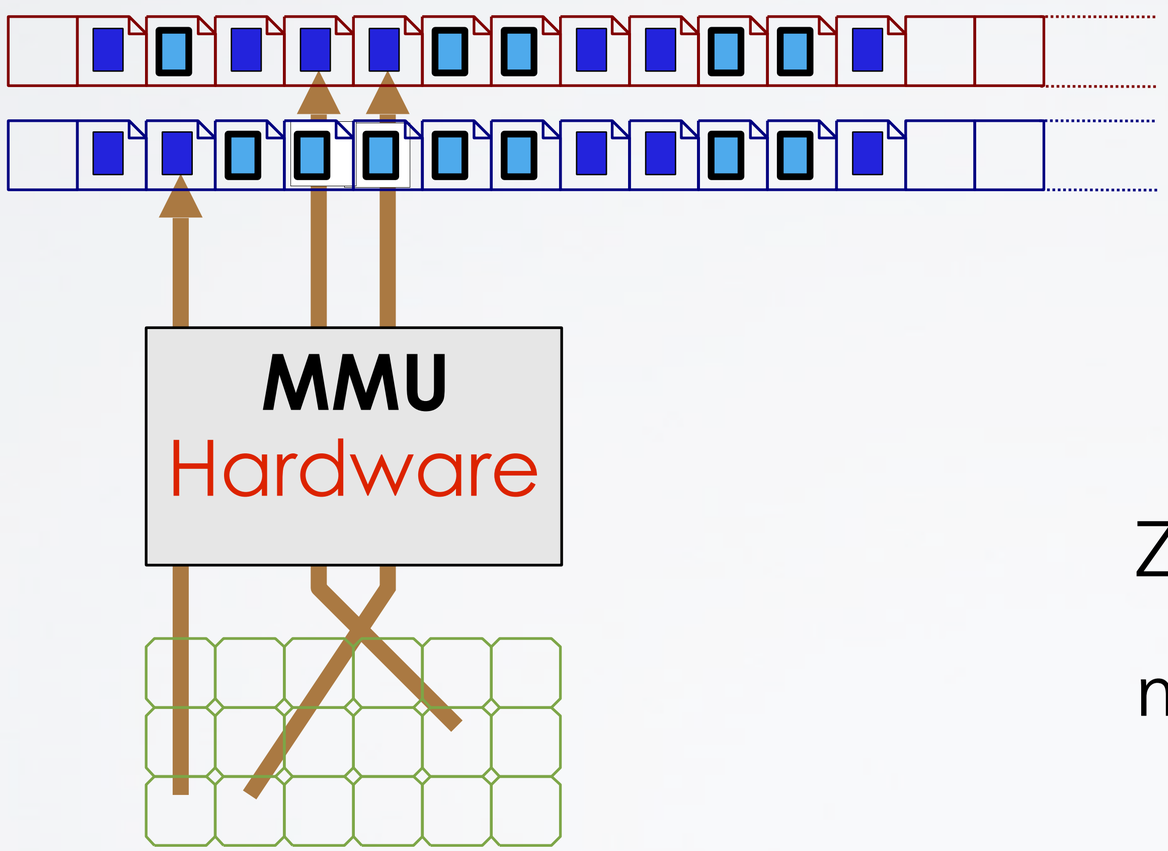


VERZÖGERTES KOPIEREN GROßER



neuer PTE mit Schreibrechten

INVERS ./. SHARING



Seite

Kachel/Rahmen

Zu einer Kachel:

mehrere virtuelle Adressen

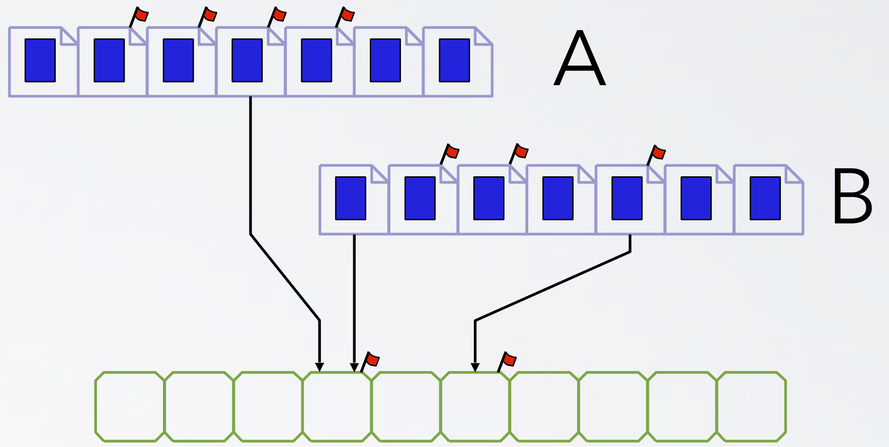
MMU: WICHTIGE DETAILS

Aufgaben

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

KACHEL ATTRIBUTE

- Attribute used, modified werden durch Hardware für Seiten (nicht Kacheln) gesetzt
- Propagation der Attribute von Seiten zu Kacheln
- Rückkehrabbildung: welchen Seiten ist eine gegebene Kachel zugeordnet

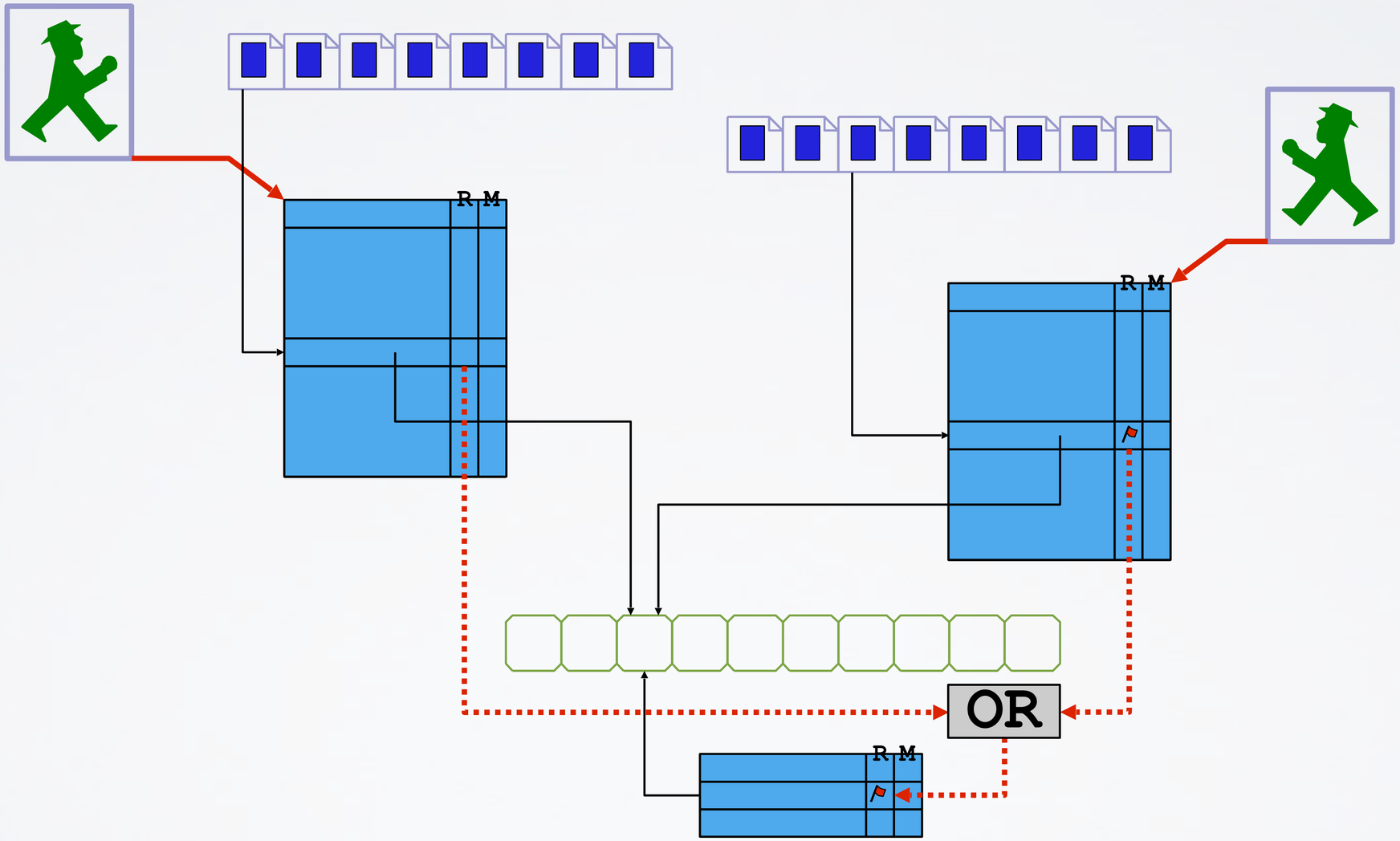


Seite in Adressraum A:
used, not modified

Seite in Adressraum B:
not used, modified

Kachel:
used, modified

PROPAGATION DER SEITENATTRIBUTE



KACHELN -> ADRESSRAUM

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

PROZESSE UND SPEICHER

Prozess-Zustand

- Seitentabellen
- mehrere parallele Aktivitäten je Adressraum: Threads

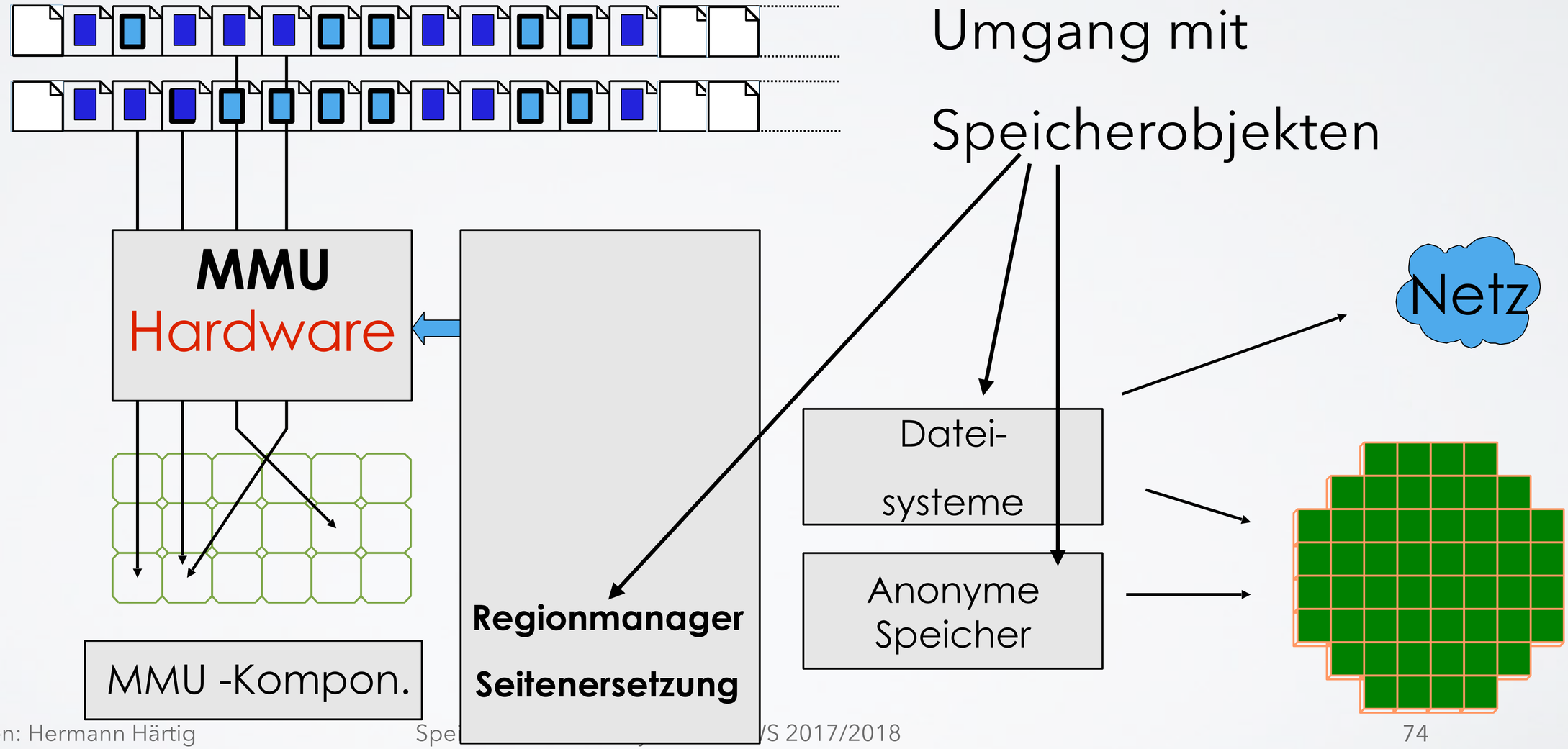
Prozess-Umschaltung

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

ABLAUF SEITENFEHLER

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

DIE BS-KOMPONENTEN



-
- **Virtueller Speicher (Paging)**
  - Anliegen - Begriffe - Vorgehen
  - Adressumsetzung, Hardware, Lazy Copying, …
  - Betriebsmittel Hauptspeicher: Seitenersetzung (Arbeitsmengenmodell)
  - Speicherobjekte
  - Integration der Einzelkomponenten
-

HAUPTSPEICHER

- verwaltet/beschafft/entzieht Kacheln
- welche ? -> Seitenaustauchverfahren
  Anhaltspunkte: referenced/modified Attribute
  in dieser Vorlesung sehr knappe Behandlung

ABLAUF GENERELL

```
NewK = getFreePageframe();
    if (!NewK) {
      //keine Kachel mehr frei und unbenutzt
      NewK = ReplacementPolicy();
      //NewK ist noch in Benutzung
      removeFromPT(NewK);
      //aus alten Adress Räumen entfernen
      if (NewK.modified)
            writeToStorageObject(NewK, ...);
    }
    NewContent(NewK);
    //z.B. fillWithZero oder readStorageObject(Seiten_Nr,NewK)
    insertIntoPT(Seiten_NR, NewK);
    //Kachel im richtigen AR, bei der richtigen SeitenNR einfügen
 }
```

Schnittstelle zu Speicherobjekten: später

WELCHE KACHEL ?

- Anhaltspunkte für die Entscheidung:
  referenced/modified Attribute von Kacheln aus Hardware/MMU
- optimal:
  die Seite verdrängen, die am längsten nich benötigt wird
- Annäherung durch “Lokalitätsprinzip”

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

GENERELLE ÜBERLEGUNG

- referenced: regelmäßig zurücksetzen
  - Zeitintervall
  - “Reihum”

1) nicht referenced, nicht modified
2) referenced, nicht modified
3) nicht referenced, modified
4) referenced, modified

LRU: LEAST RECENTLY USED

Verdränge am längsten nicht genutzte Seite

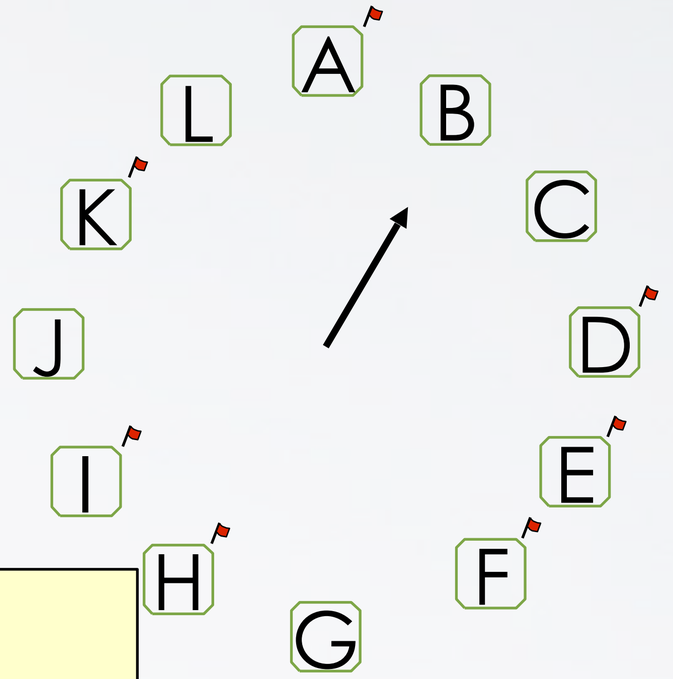
Vorteil

- gute Näherung für optimalen Algorithmus (Lokalität)

Nachteil

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

LRU ANNÄHERUNG: CLOCK-




```
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
  niedrige Zahl → hohes Alter

LRU ANNÄHERUNG: AGING

Tick 1

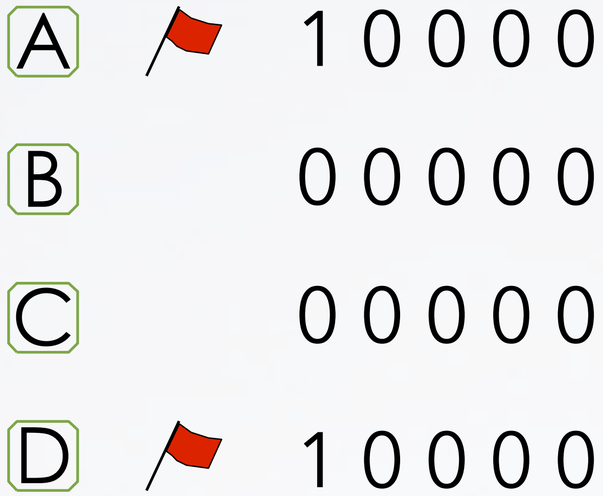
A	0 0 0 0 0
B	0 0 0 0 0
C	0 0 0 0 0
D	0 0 0 0 0

LRU ANNÄHERUNG: AGING

Tick 1

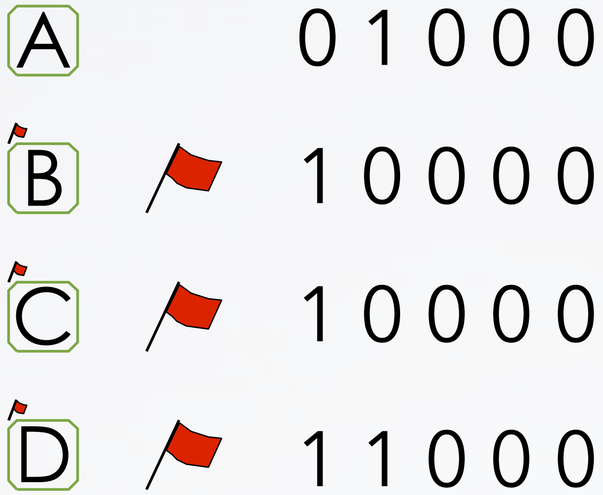
LRU ANNÄHERUNG: AGING

Tick 1



LRU ANNÄHERUNG: AGING

Tick 2



LRU ANNÄHERUNG: AGING

Tick 3

LRU ANNÄHERUNG: AGING

Tick 4

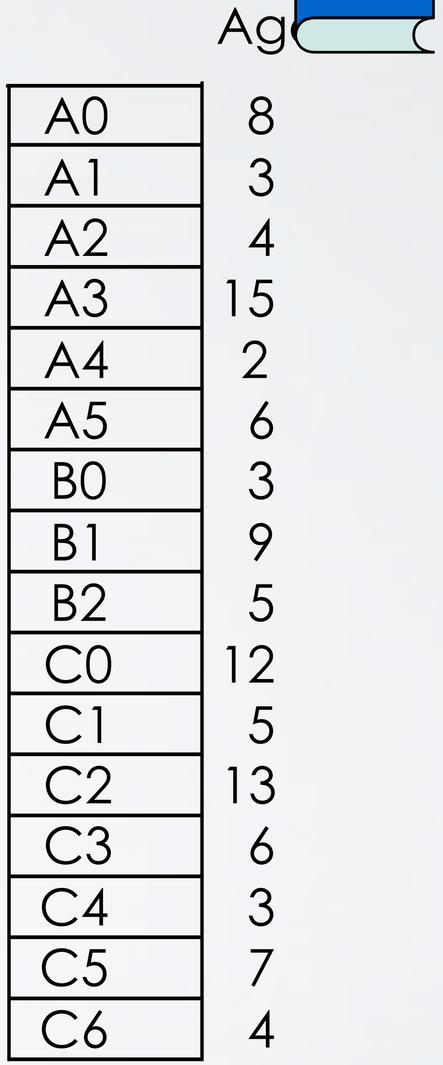
LRU ANNÄHERUNG: AGING

		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

Alter wie Geburtsdatum interpretieren

PROZESSLOKALER / GLOBALER 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
  → Arbeitsmengenmodell



DAS ARBEITSMENGENMODELL

Betrachtung der Seitenreferenzfolge eines Prozesses durch ein „Fenster“ der Länge T

- 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 benutzten Seiten in T Zeiteinheiten
- $\overline{|w(t, T)|}$ << max (Z(T))

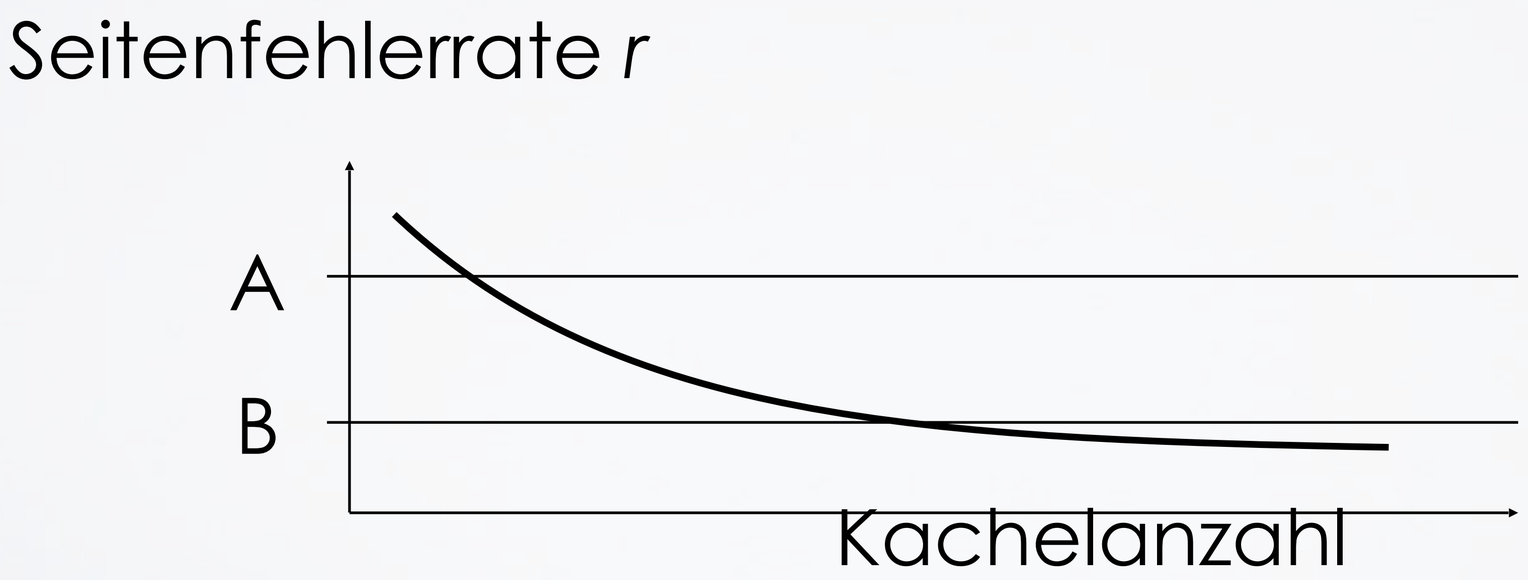
DAS ARBEITSMENGENMODELL

Vorgehen

- Bestimmung der Arbeitsmenge
- für aktive und bereite Prozesse muss sich mindestens die Arbeitsmenge im Speicher befinden
- Swapping, falls Kacheln fehlen
- Prepaging bei Wiedereinlagerung

SEITENFEHLERRATE

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



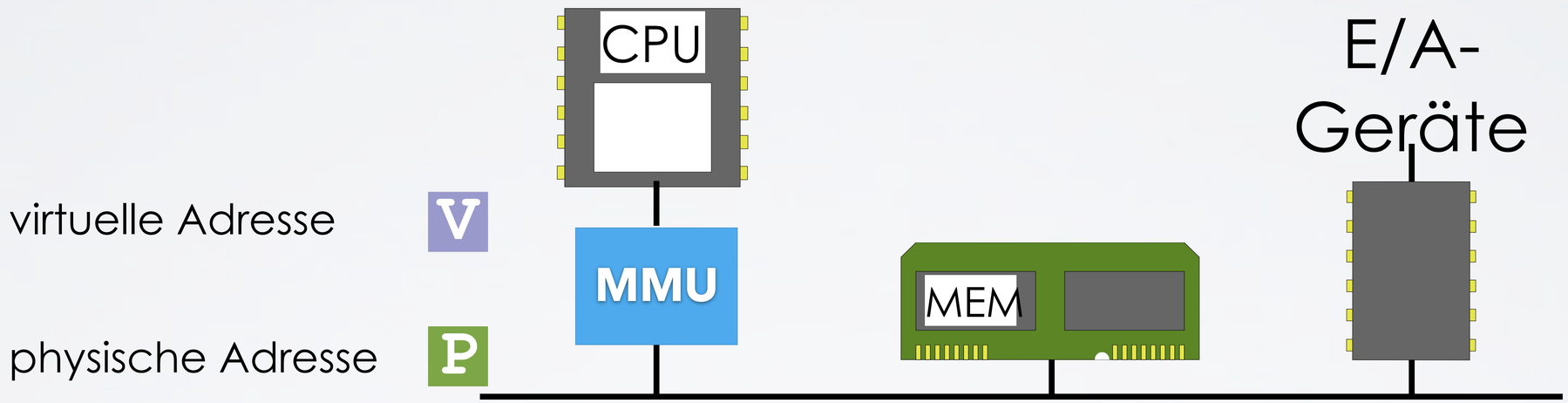
MISC: SEITENGRÖßE

- 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

MISC: RESIDENTE SEITEN



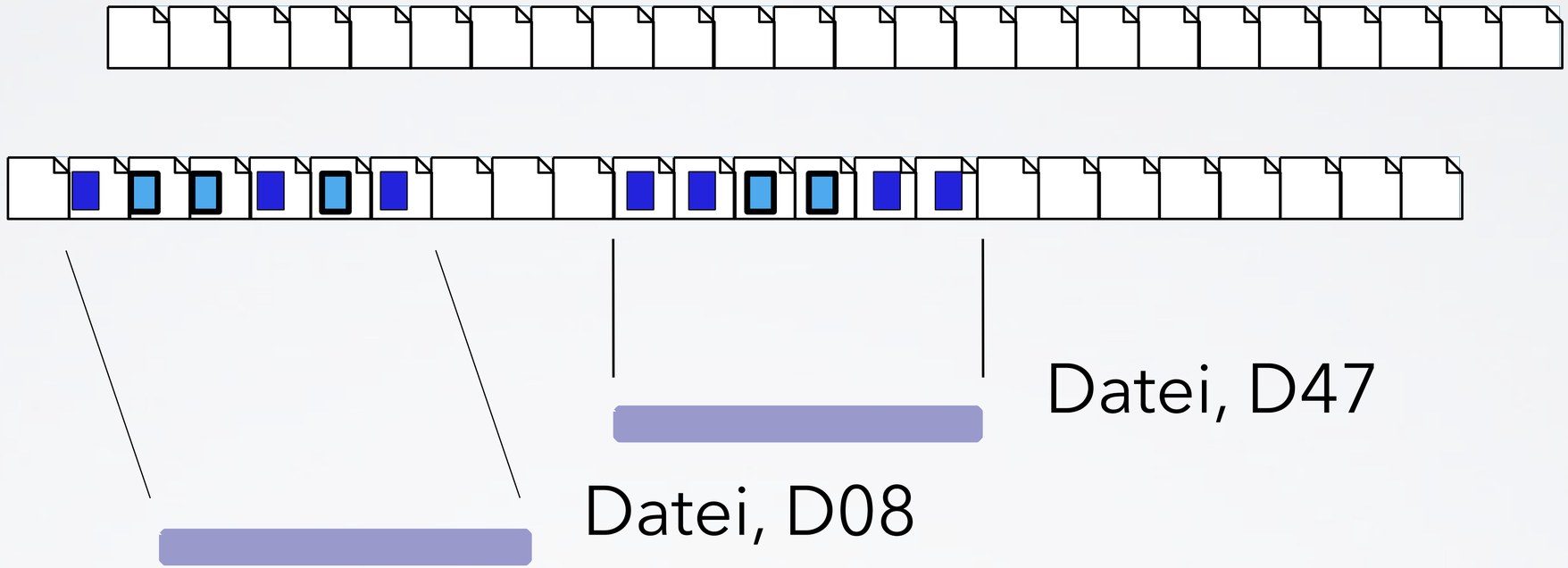
- häufige, einfache, aber limitierte Lösung: Kopieren in/aus festem residentem Zwischenpuffer
- besser: Residentsetzen beliebiger Adressbereiche, „Pinning“ - Umrechnen virtuell → real durch Treiber
- weitere Anwendung: zeitkritische Applikationen

- Elementare Techniken
- **Virtueller Speicher (Paging)**
  - Anliegen - Begriffe - Vorgehen
  - Adressumsetzung, Hardware, Lazy Copying, …
  - Betriebsmittel Hauptspeicher: Seitenersetzung (Arbeitsmengenmodell)
  - Speicherobjekte
  - Integration der Einzelkomponenten
- Caches

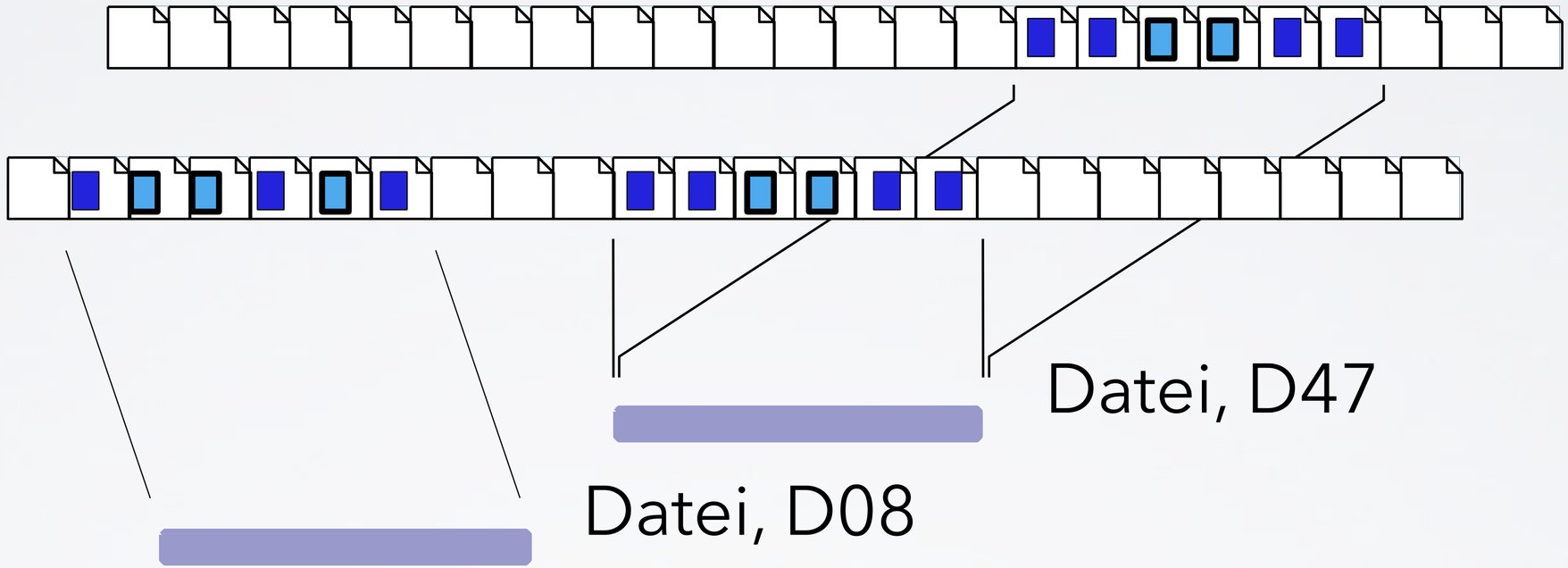
SPEICHEROBJEKTE

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

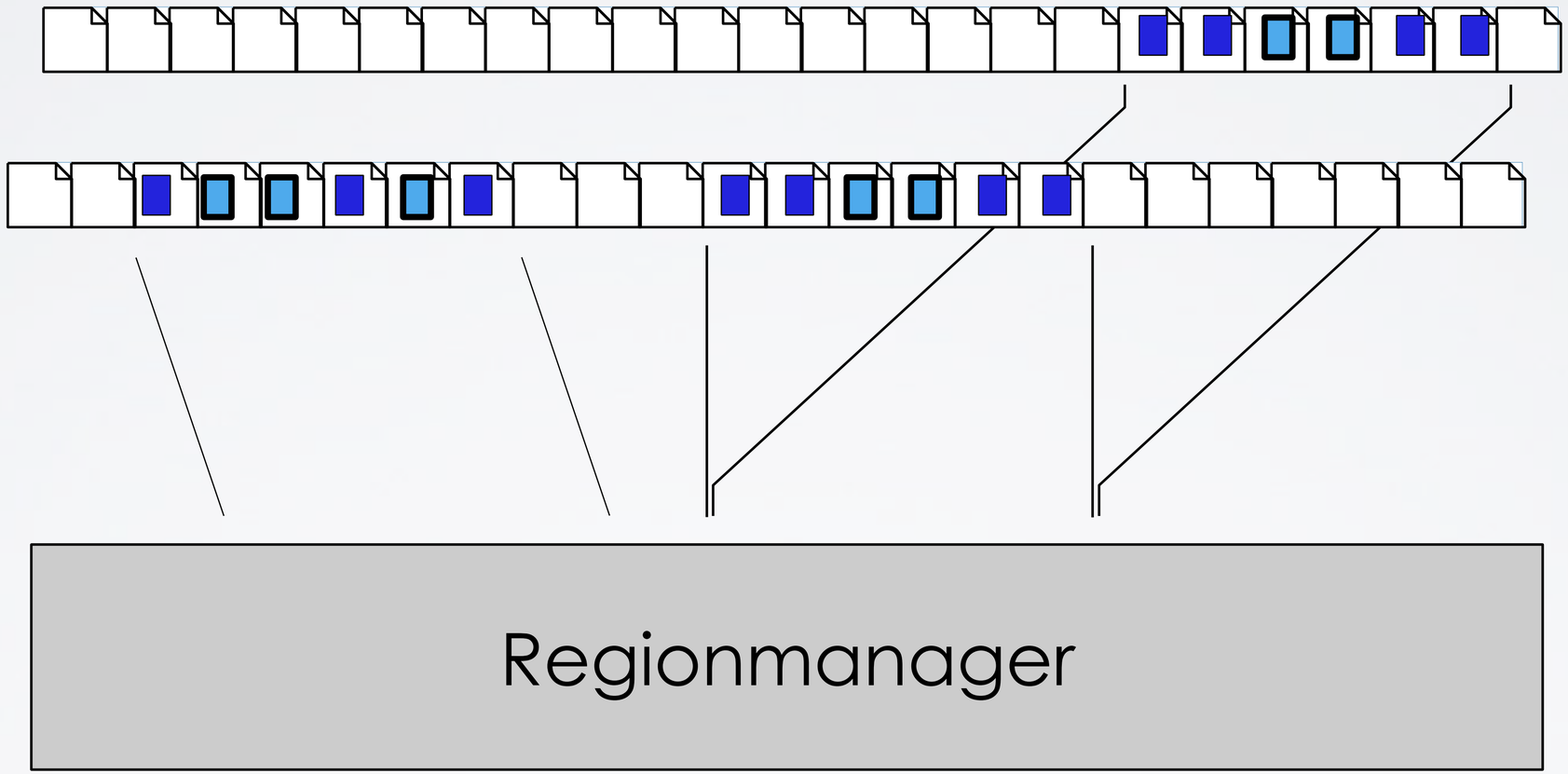
SPEICHEROBJEKTE UND REGIONEN



SPEICHEROBJEKTE UND REGIONEN



SPEICHEROBJEKTE UND REGIONEN



- 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

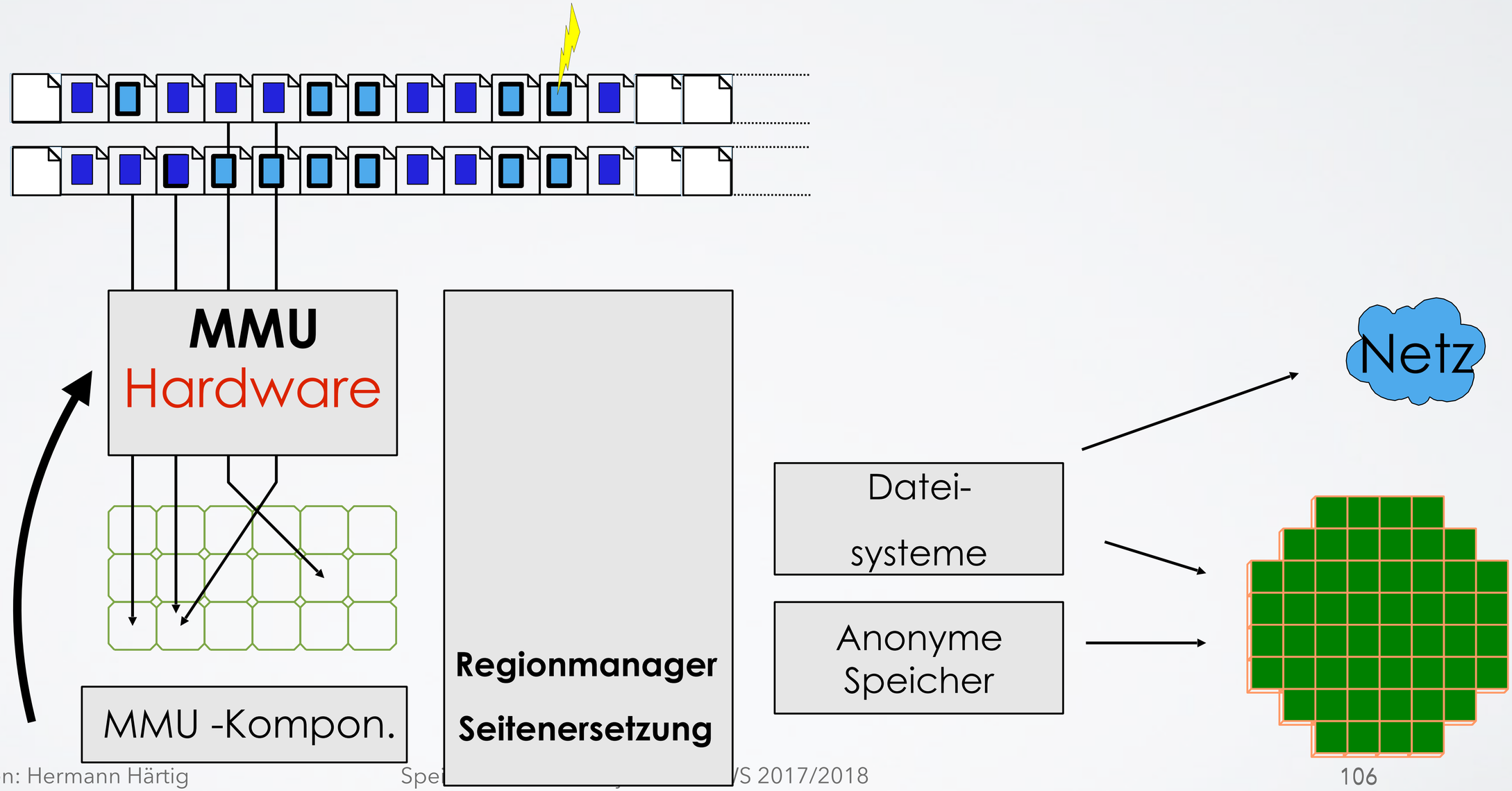
- z. B.
  ```
  map( ... , ... , Datei, R);

  map( ... , ... , Gerätespeicher, RW);
  ```
- implizit: bei Prozesserzeugung werden Programm-, Keller-, Daten-Regionen gebildet (in Unix: „Segmente“)

REGIONEN-MANAGER

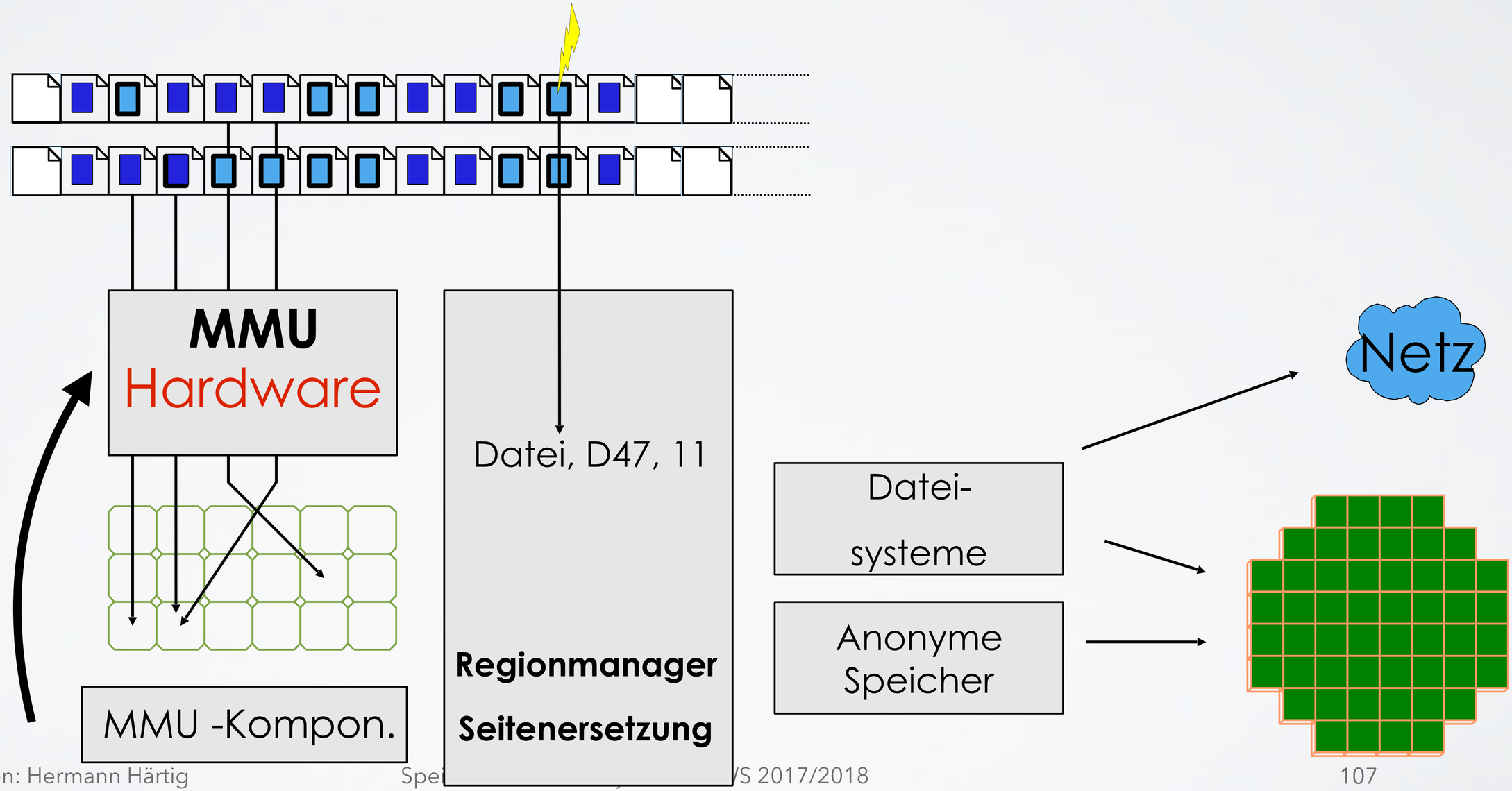
- Verwaltung der Adressraum-Struktur
  repräsentiert z. B. als verkettete Liste
- Aufgabe bei Seitenfehler: Bestimmen von
  - potentielle Fehlerart
    - Zugriff auf ungültigen Bereich (z.B. Pointerfehler)
    - Rechteverletzung (z.B. Schreiben in Code-Bereich)
  - Speicherobjekt
    - Seite innerhalb des Speicherobjektes
    - Copy On Write Fault

-
- **Virtueller Speicher (Paging)**
  - Anliegen - Begriffe - Vorgehen
  - Adressumsetzung, Hardware, Lazy Copying, …
  - Betriebsmittel Hauptspeicher: Seitenersetzung (Arbeitsmengenmodell)
  - Speicherobjekte ⟶ Dateisysteme im Januar
  - Zusammenspiel der Einzelkomponenten
-

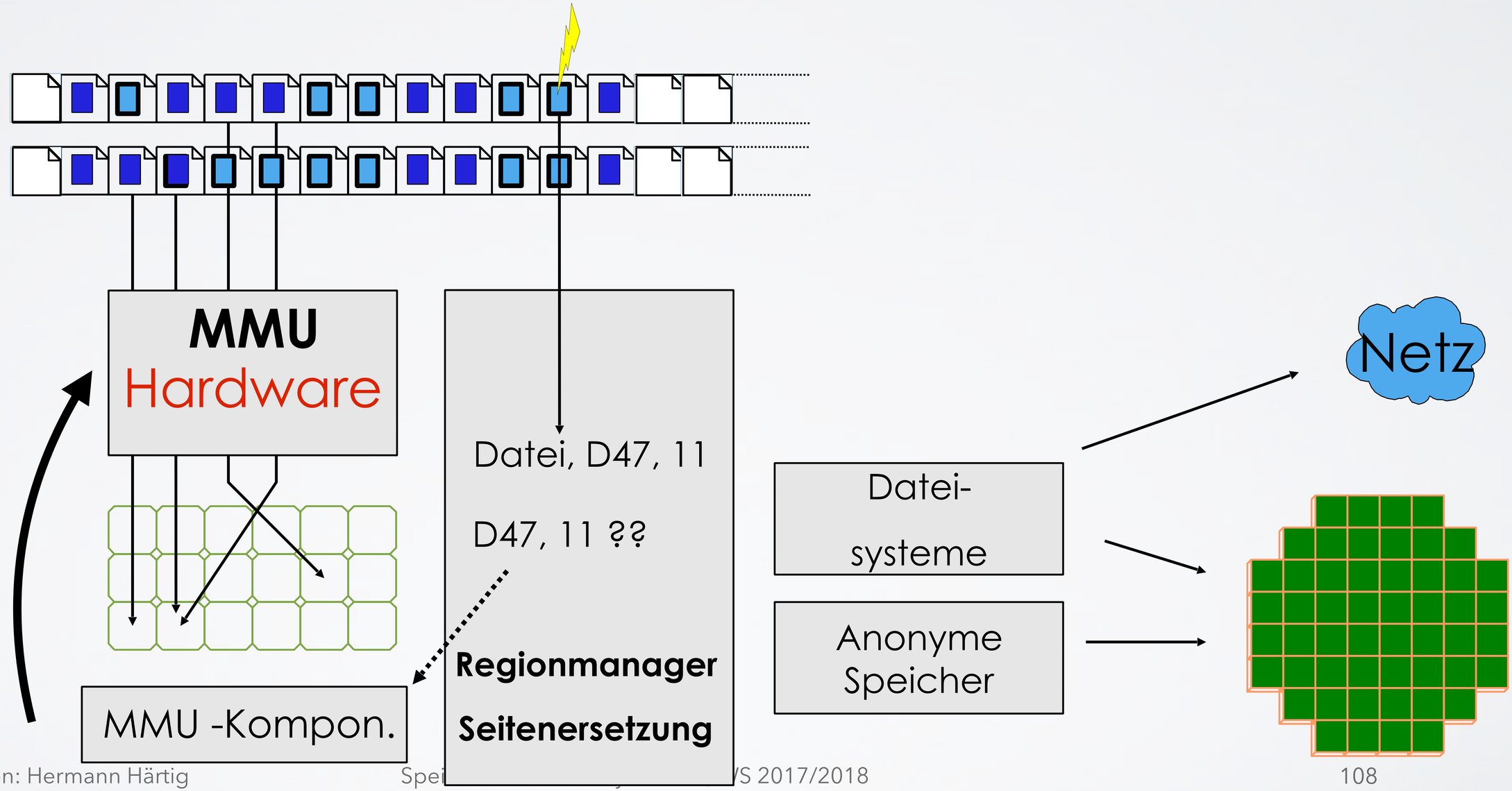
DAS ZUSAMMENSPIEL



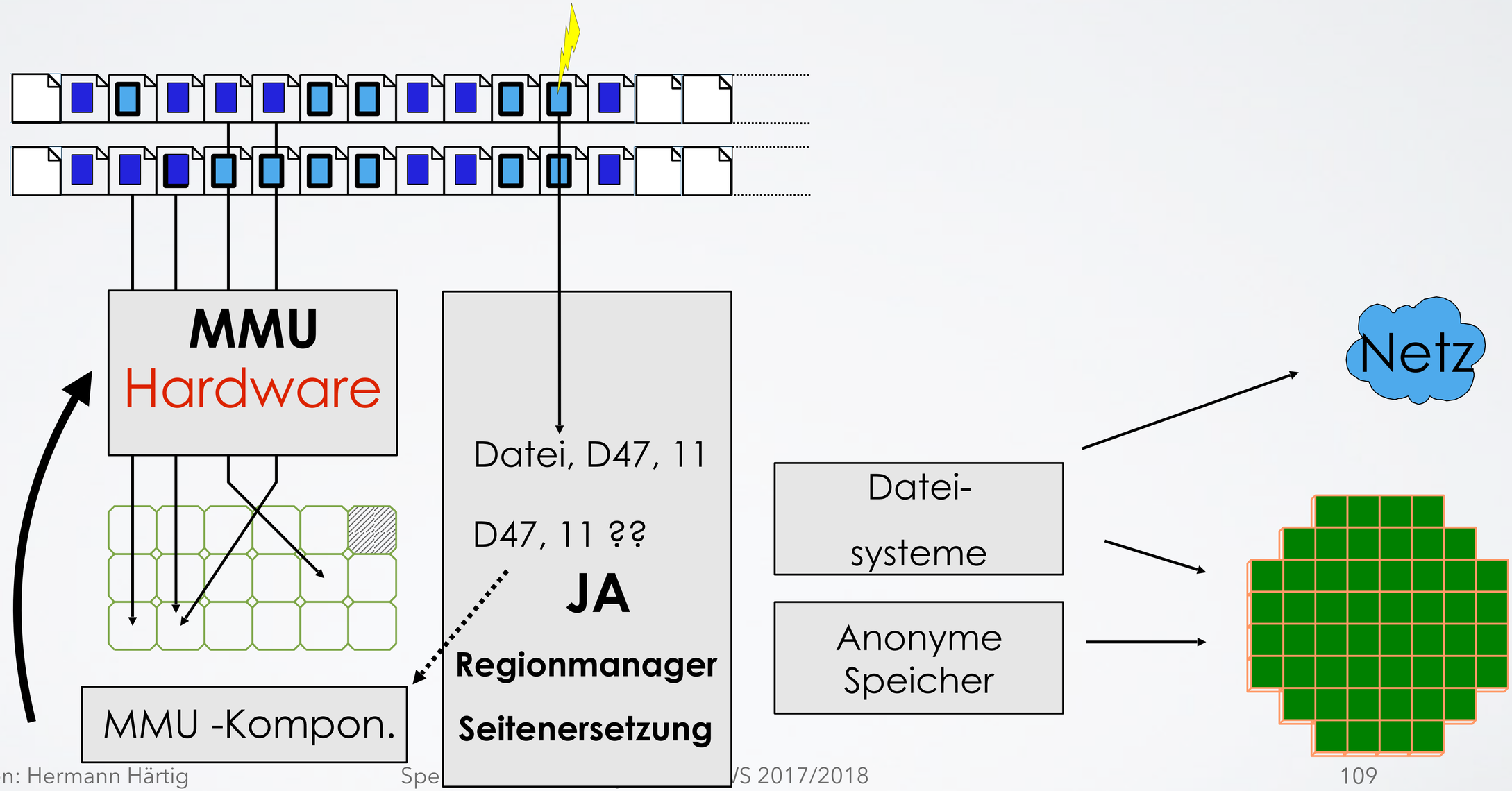
DAS ZUSAMMENSPIEL



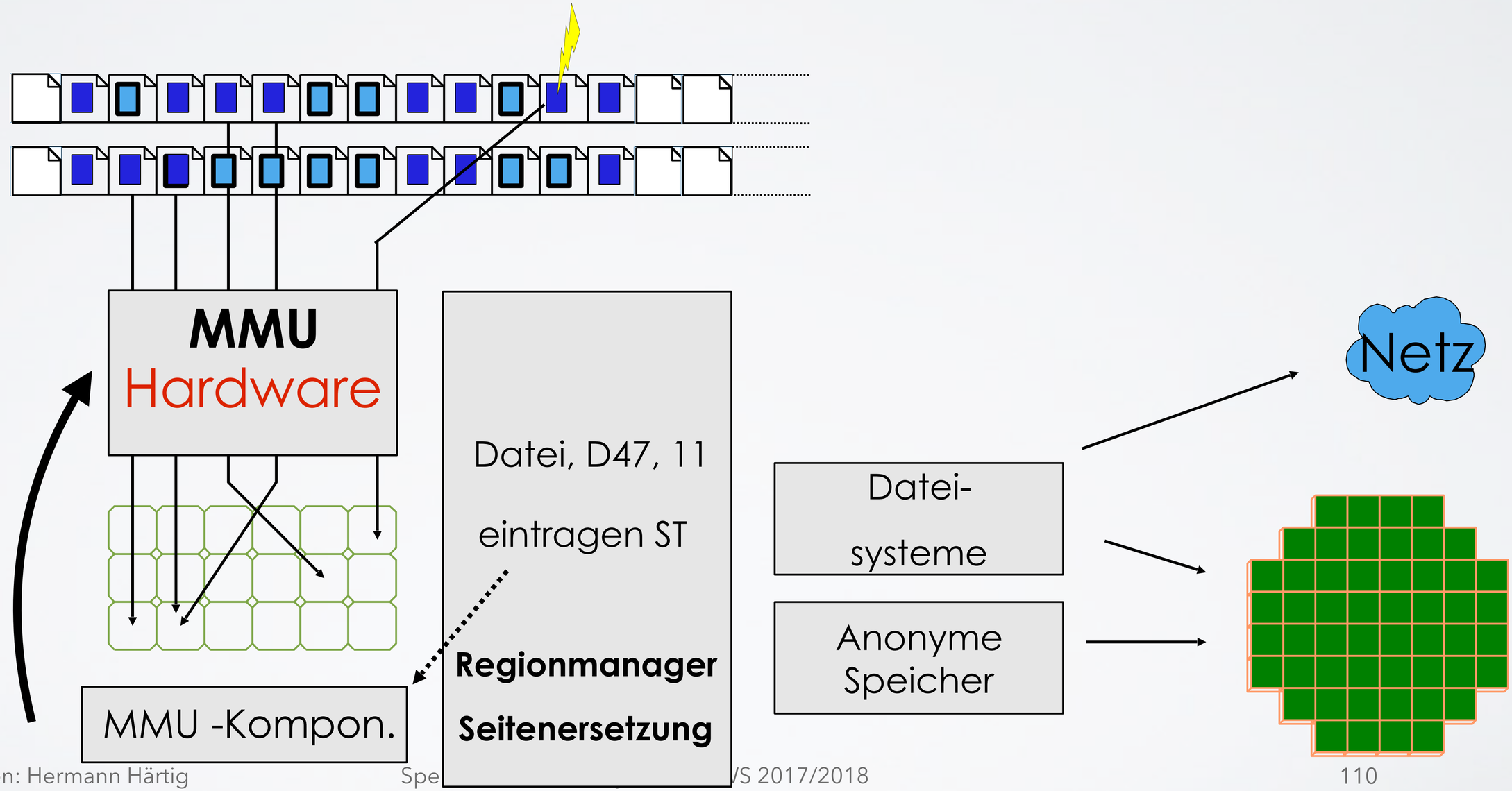
DAS ZUSAMMENSPIEL



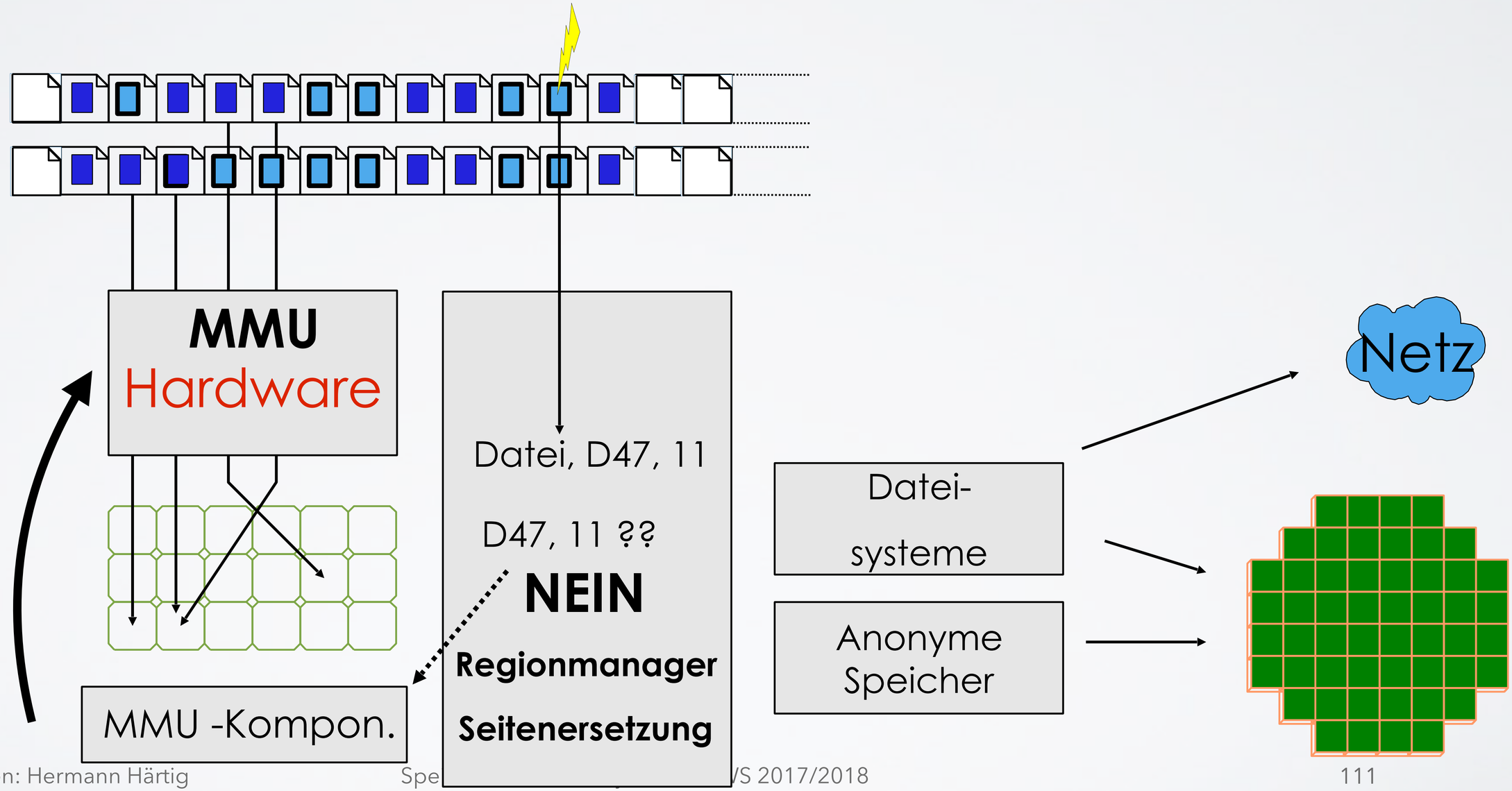
DAS ZUSAMMENSPIEL



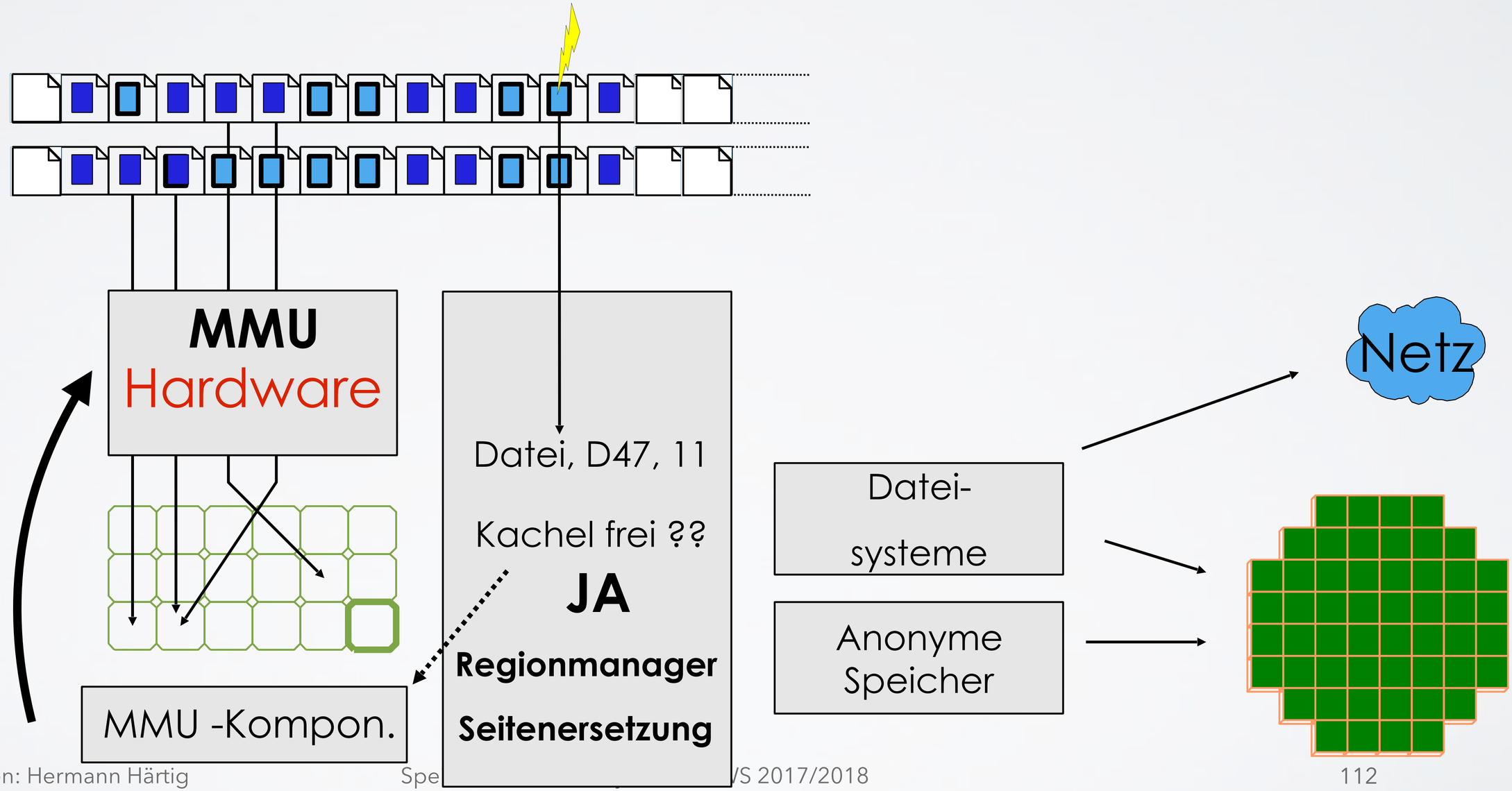
DAS ZUSAMMENSPIEL



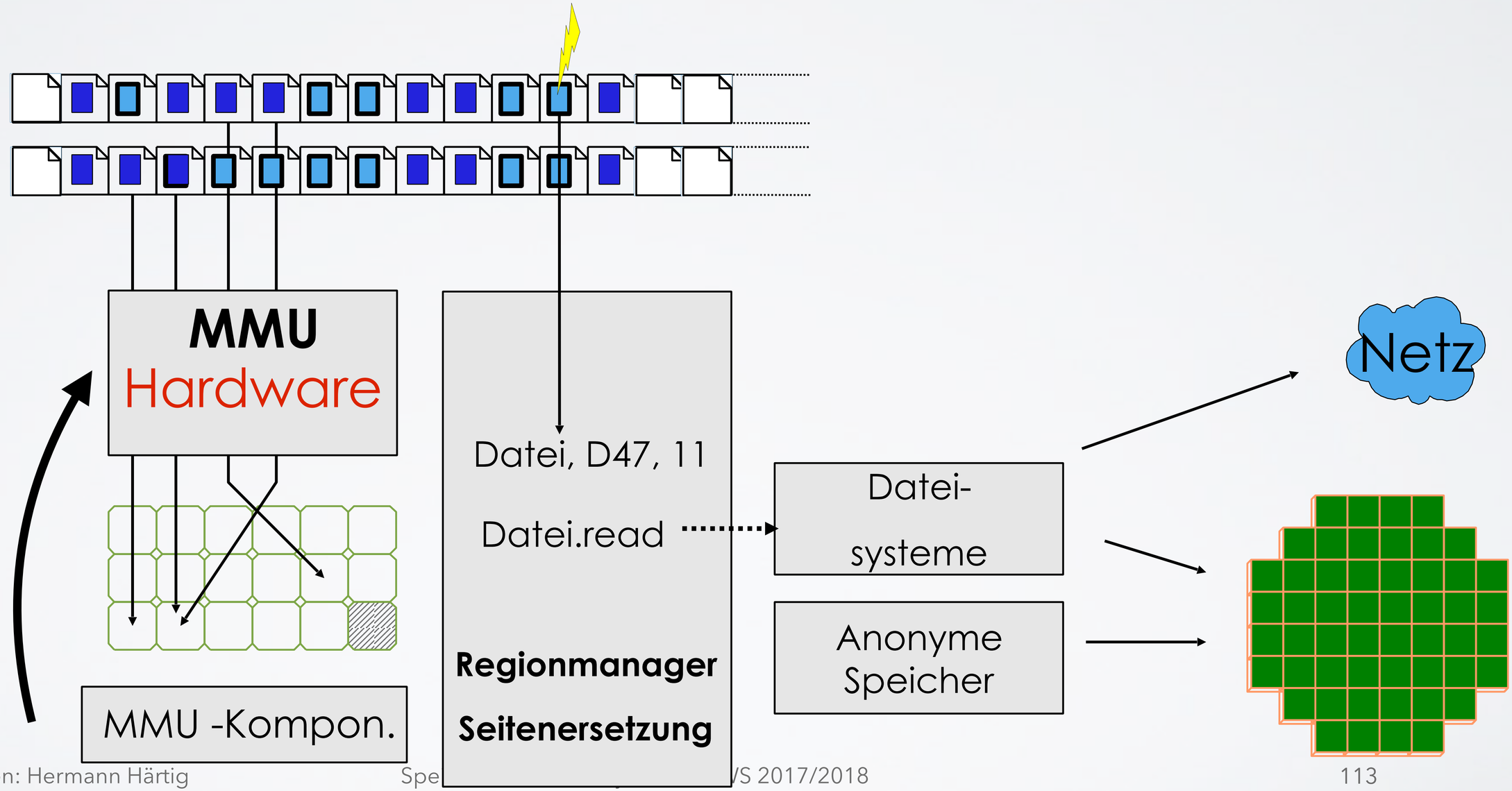
DAS ZUSAMMENSPIEL



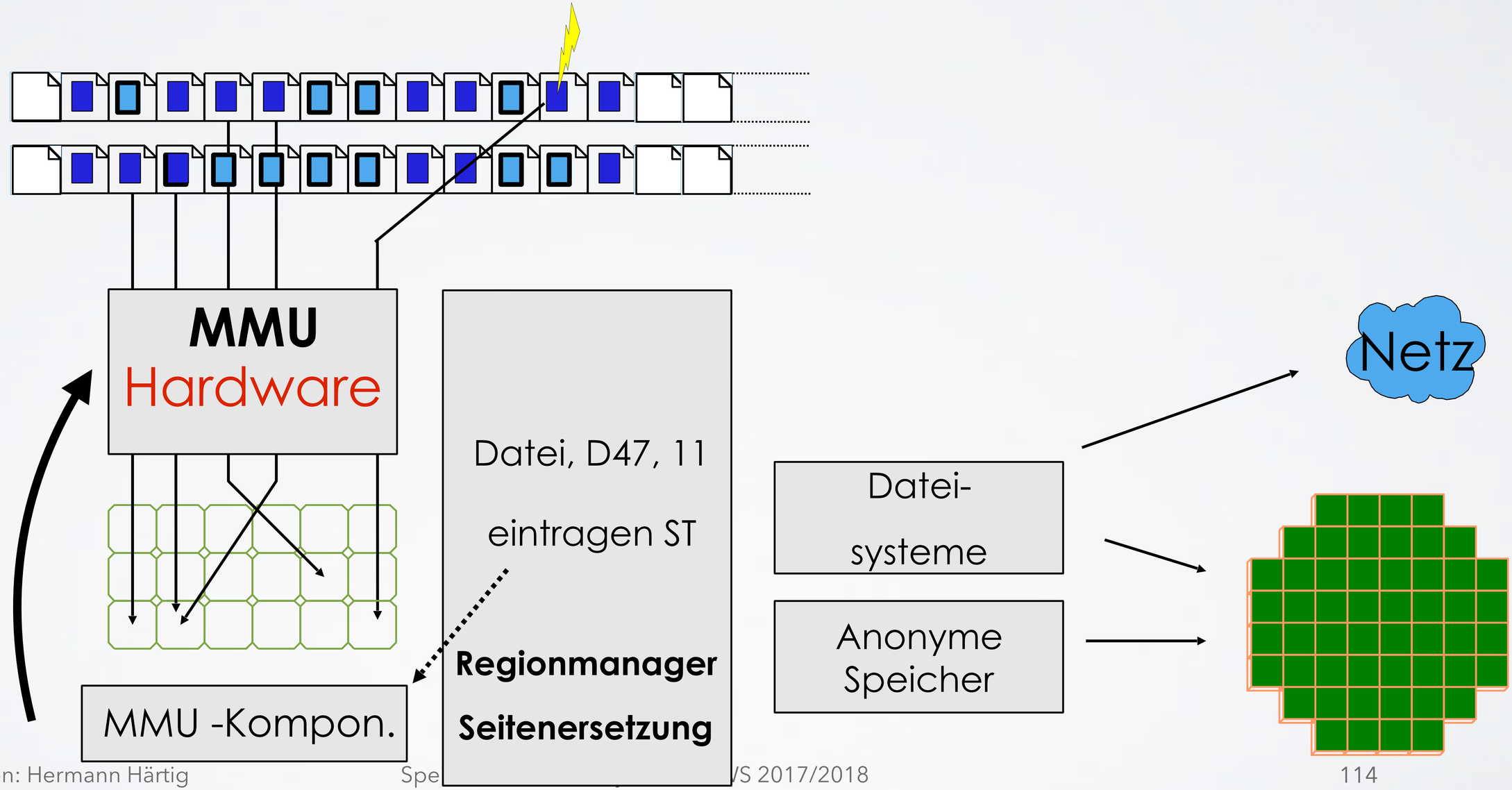
DAS ZUSAMMENSPIEL



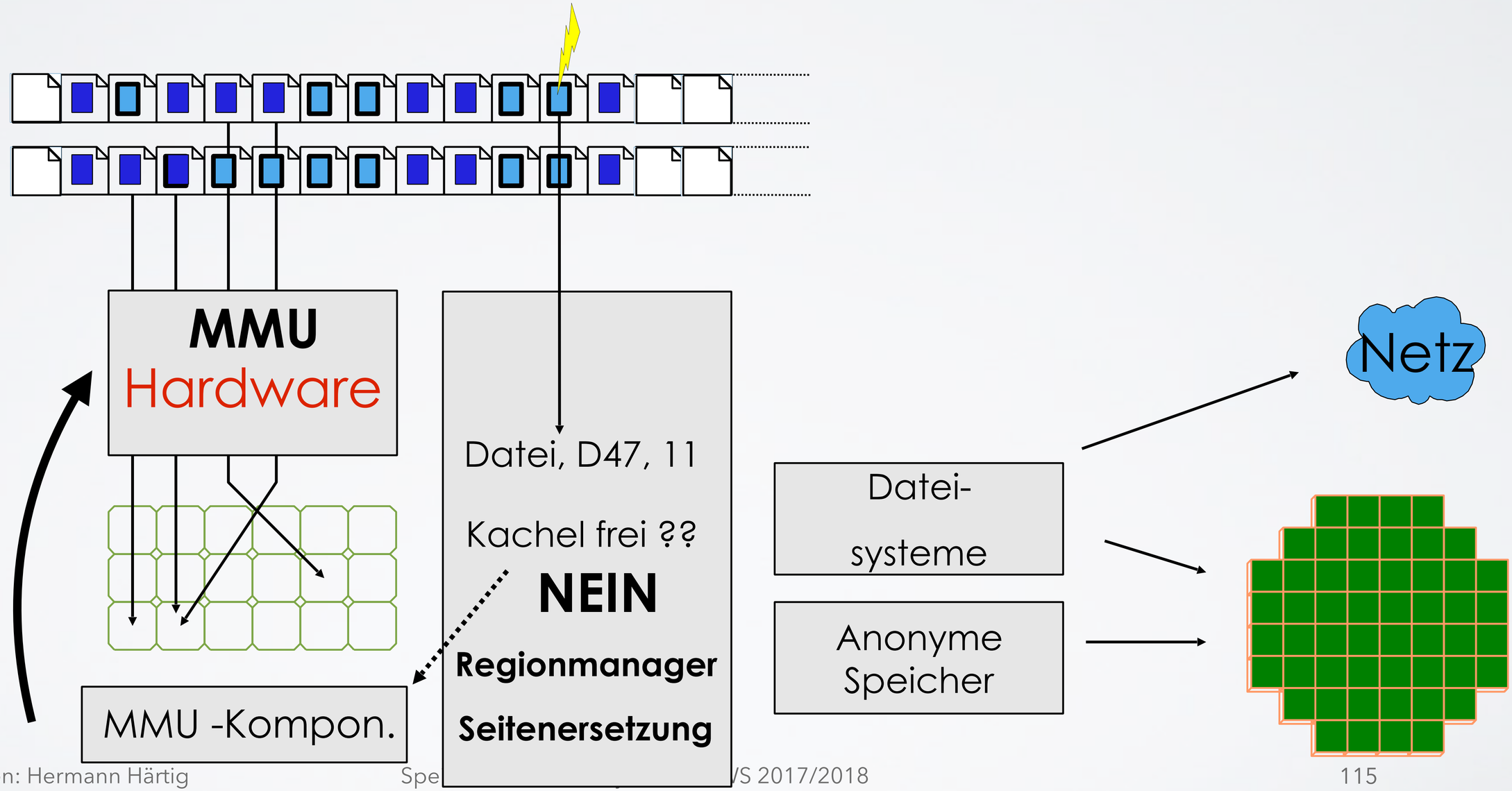
DAS ZUSAMMENSPIEL



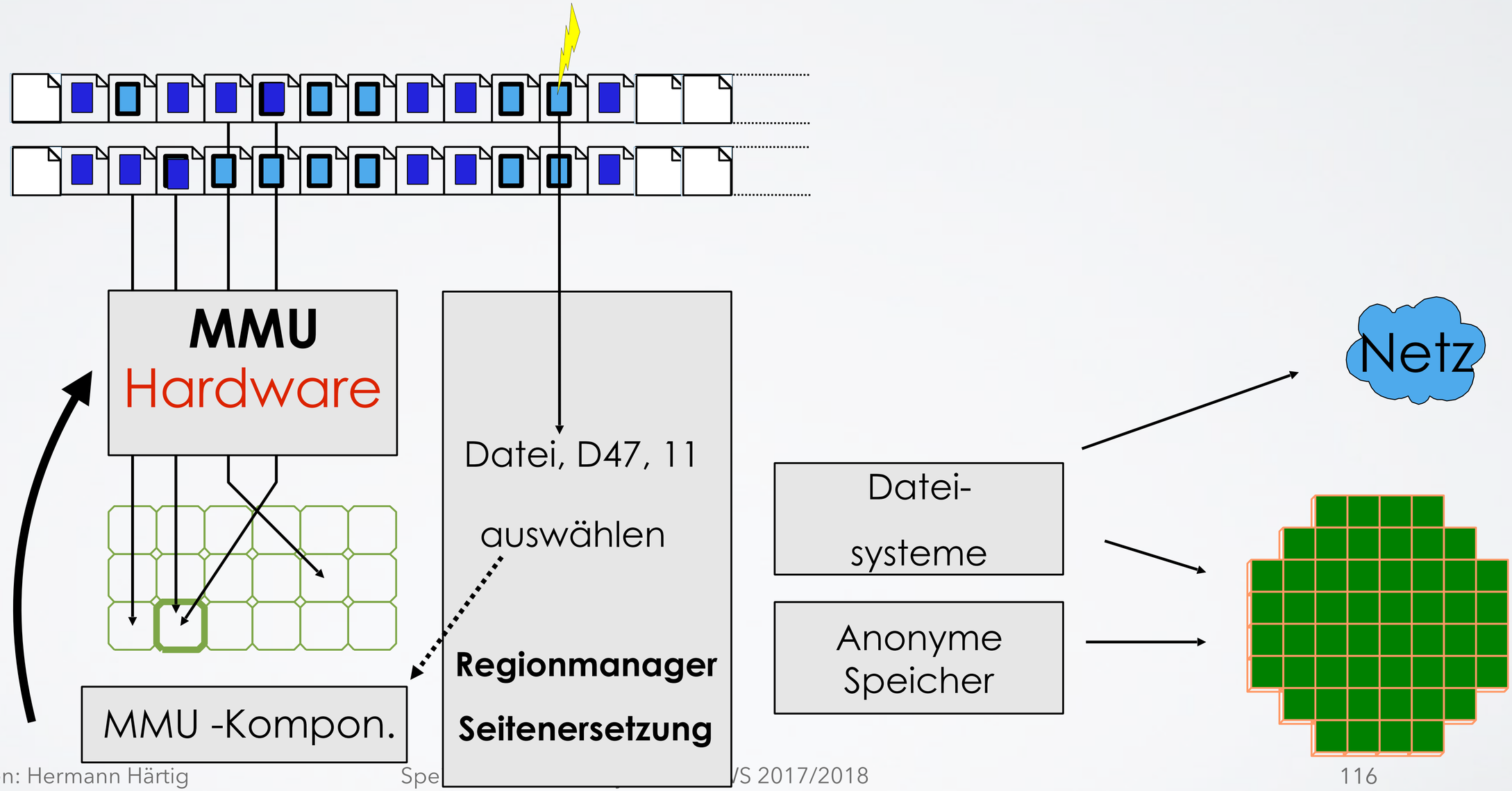
DAS ZUSAMMENSPIEL



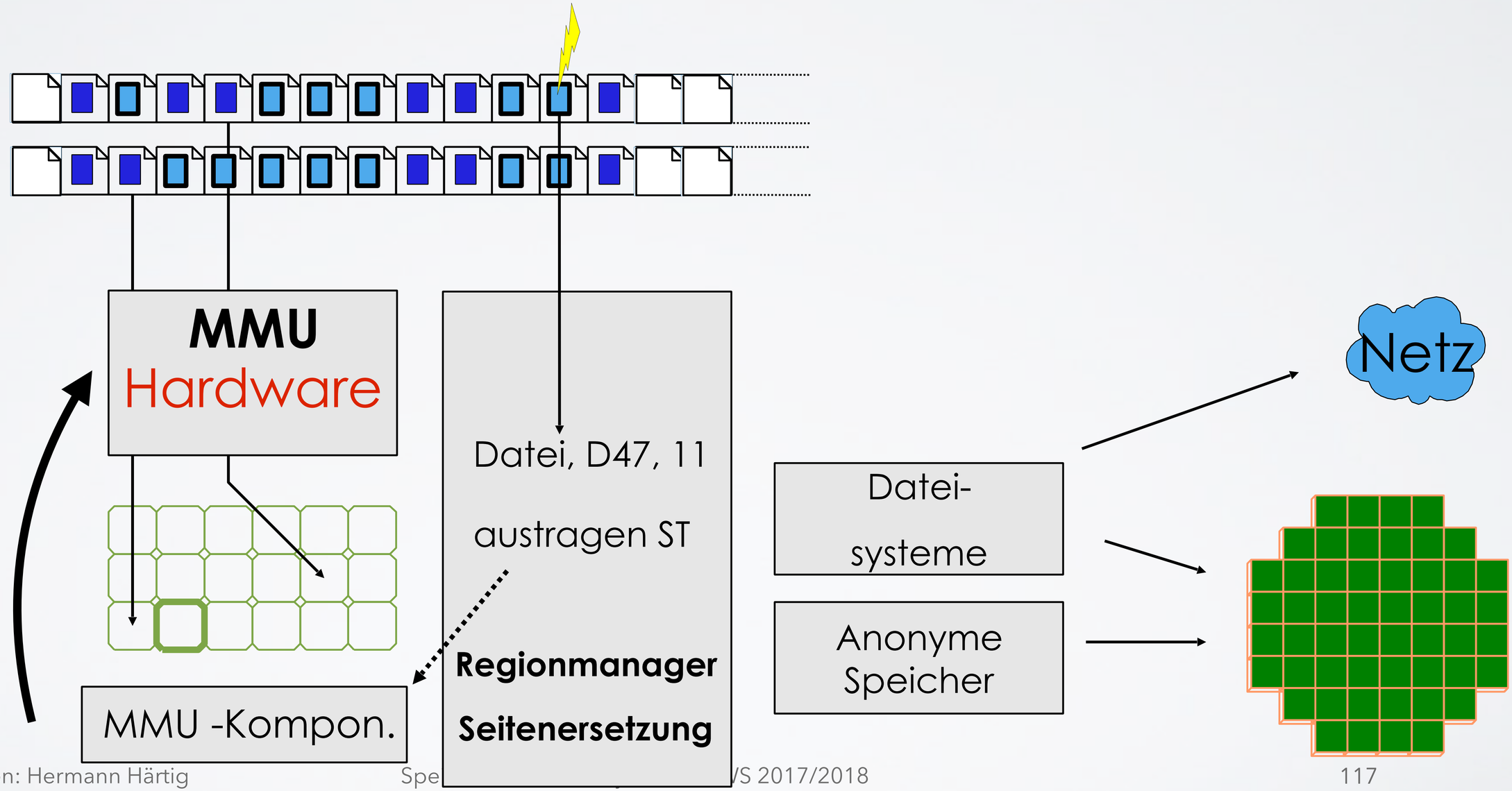
DAS ZUSAMMENSPIEL



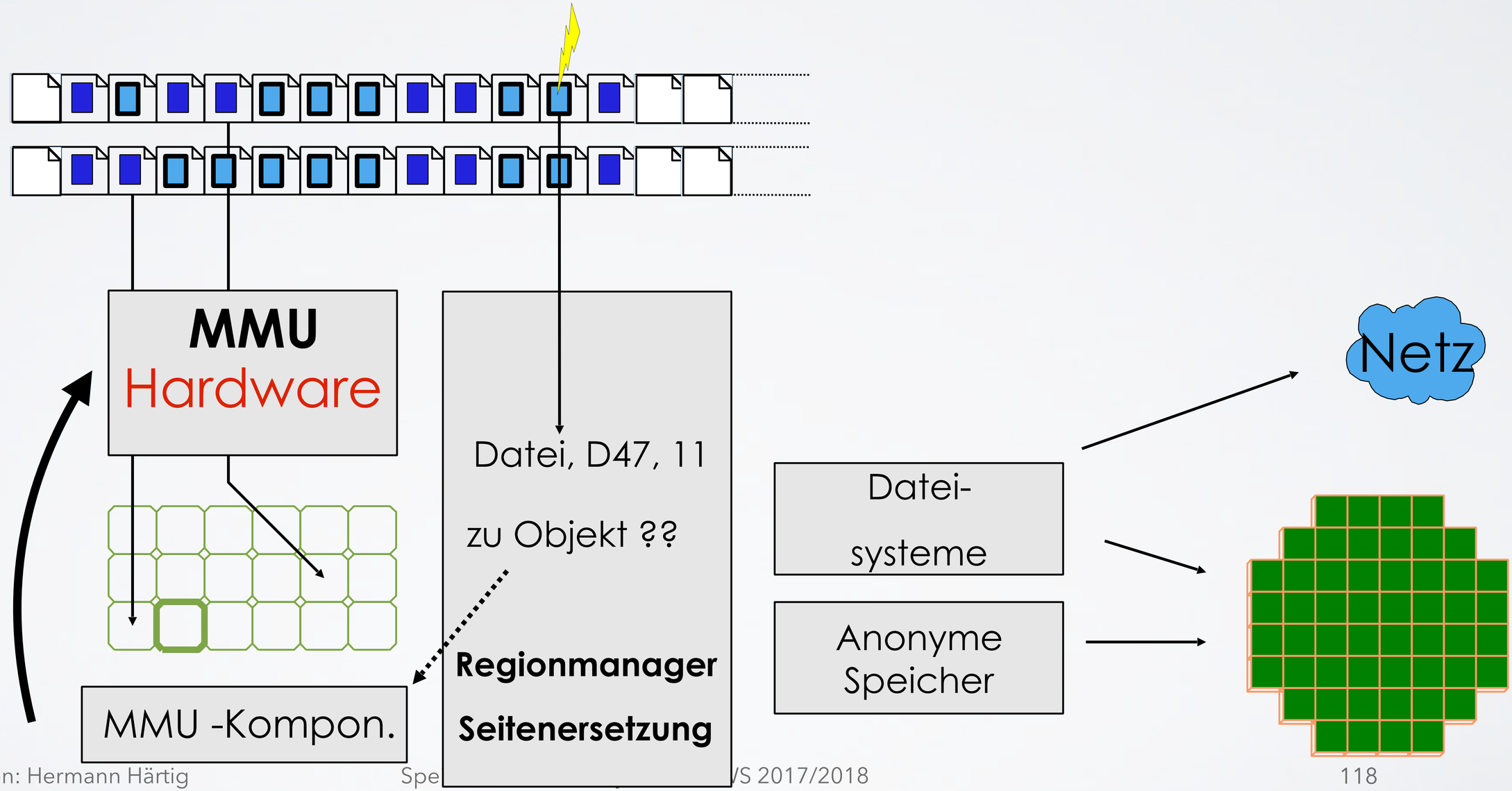
DAS ZUSAMMENSPIEL



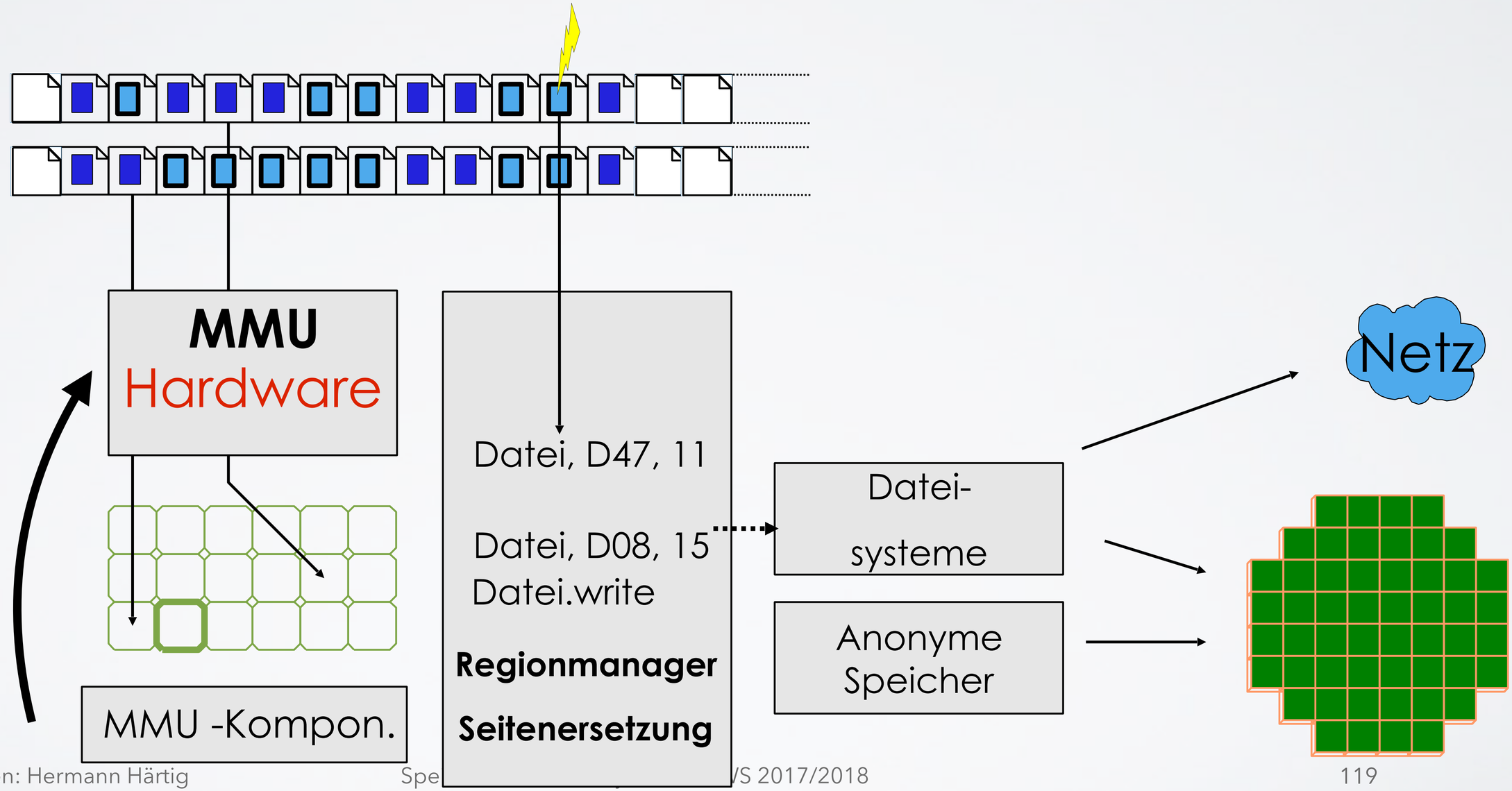
DAS ZUSAMMENSPIEL



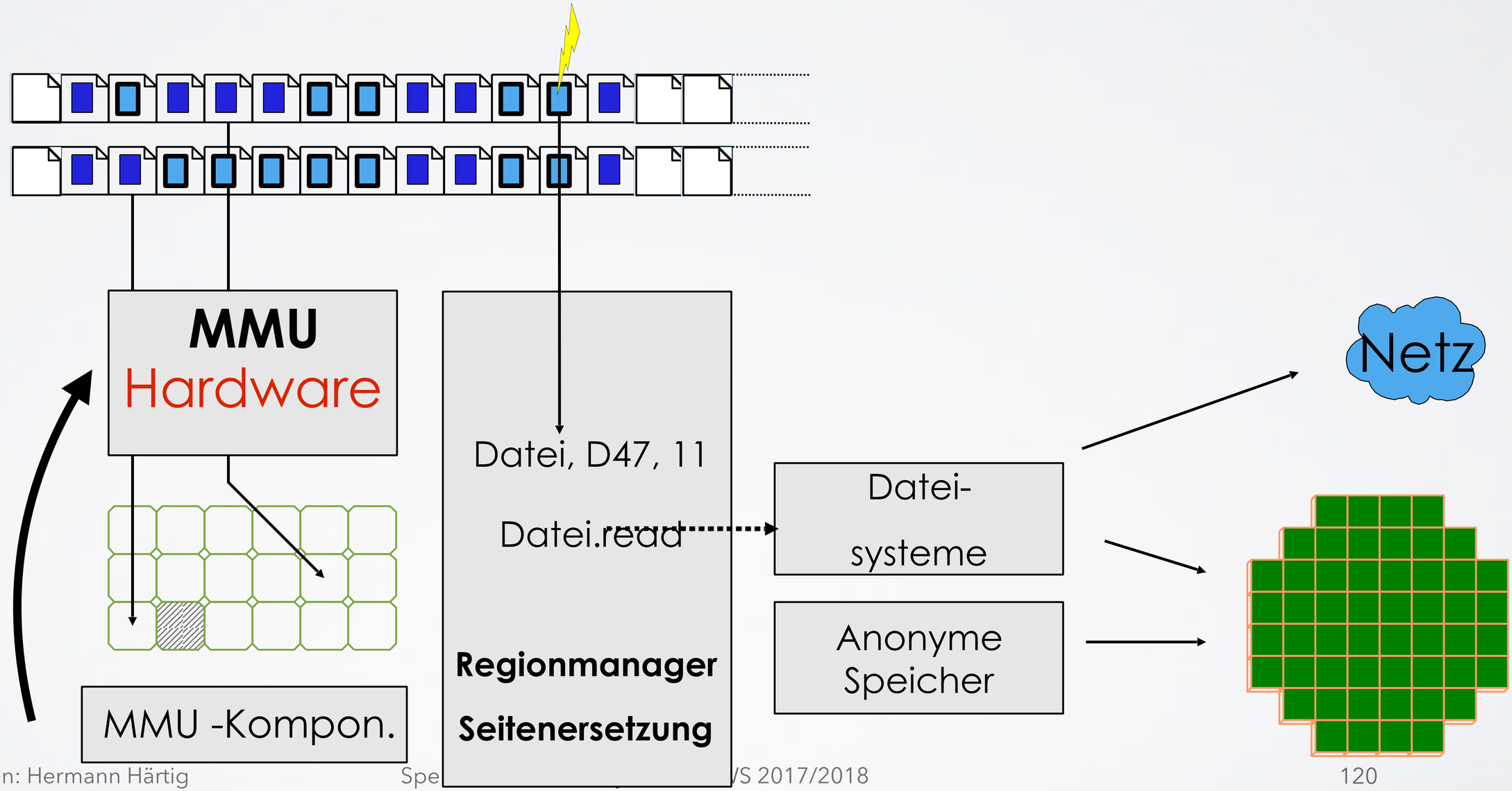
DAS ZUSAMMENSPIEL



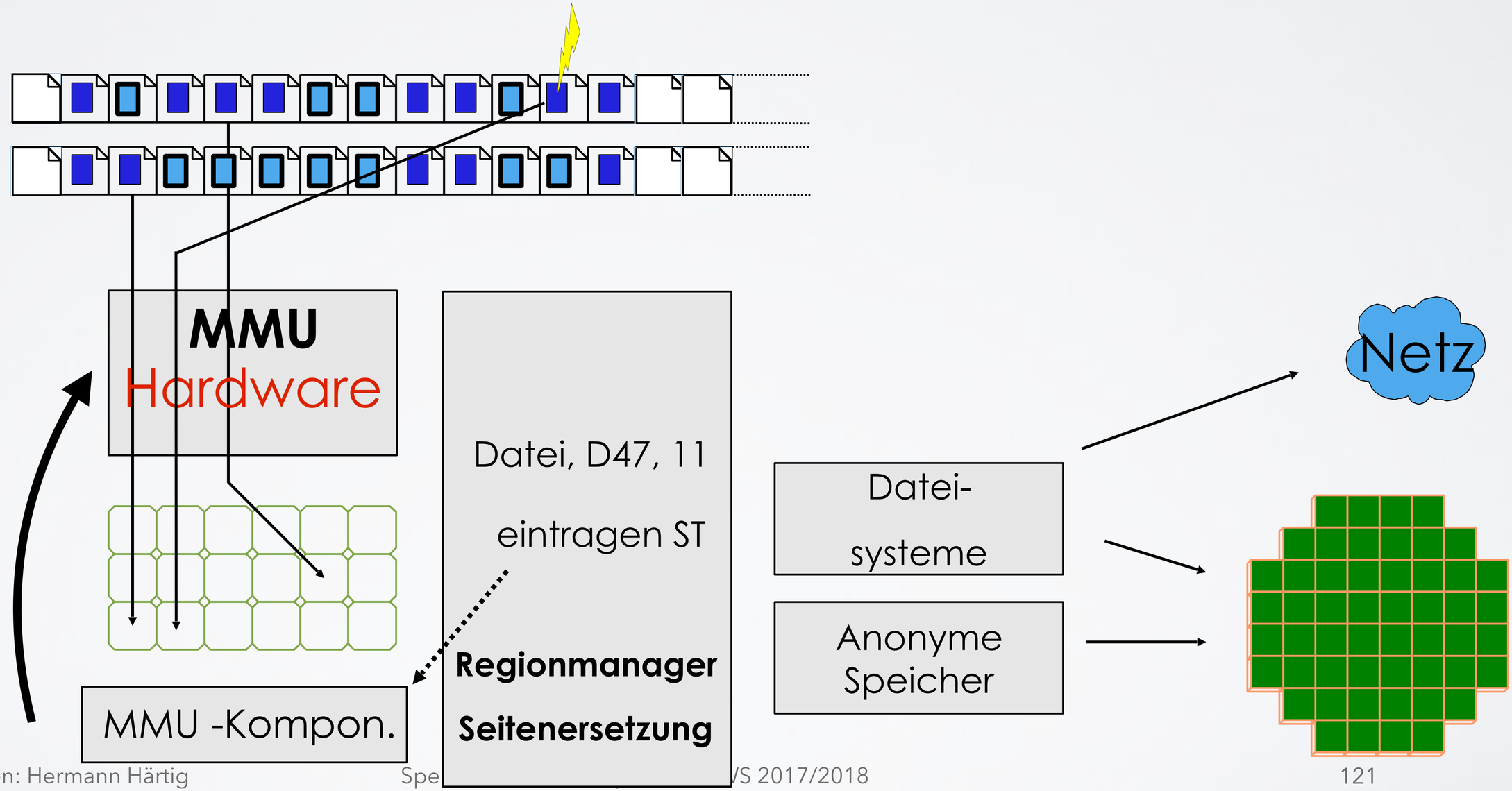
DAS ZUSAMMENSPIEL



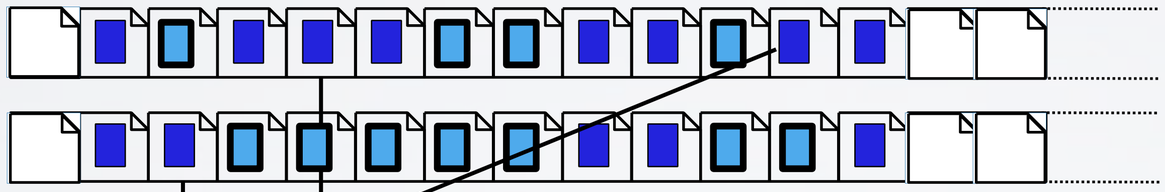
DAS ZUSAMMENSPIEL



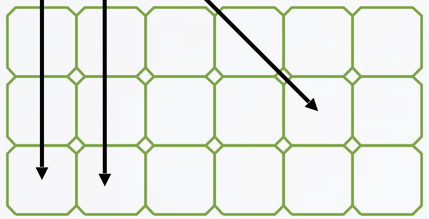
DAS ZUSAMMENSPIEL



DAS ZUSAMMENSPIEL



MMU
Hardware



MMU -Kompon.

IRET:
Wiederaufsetzen
des unter-
brochenen
Befehls

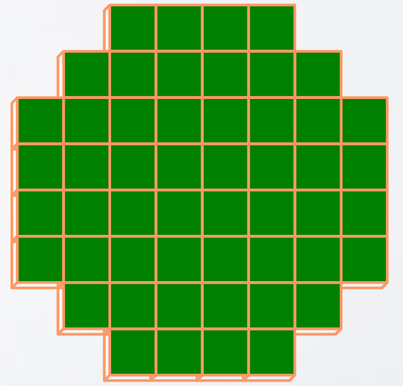
Regionmanager

Seitenersetzung

Datei-
systeme

Anonyme
Speicher

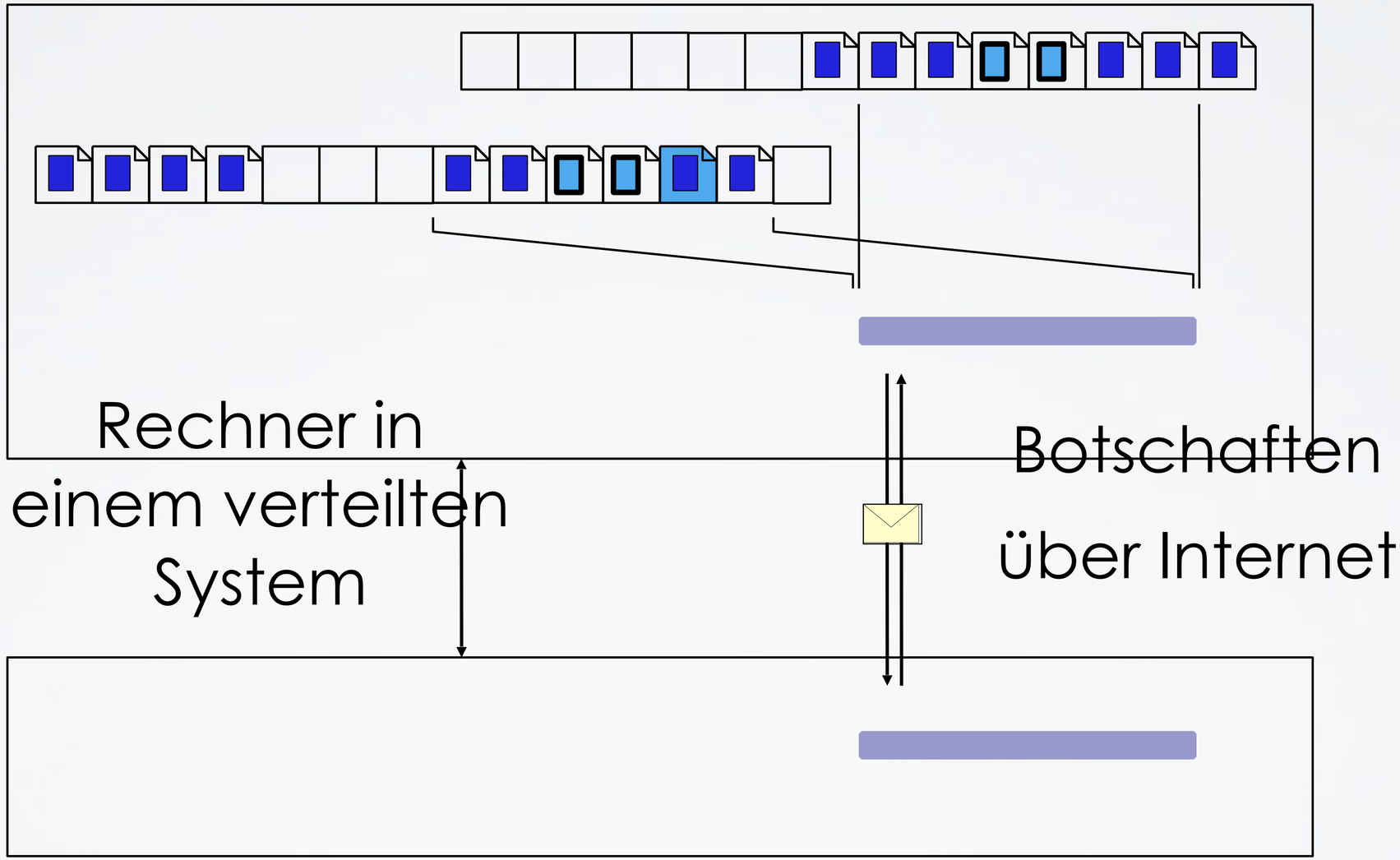
Netz



DAS ZUSAMMENSPIEL

- Seitenfehler-Exception
- 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 → verdrängen

SPEICHEROBJEKTE IM NETZ



SYSTEMSTRUKTUREN FÜR

Unix, Windows: „monolithisch“

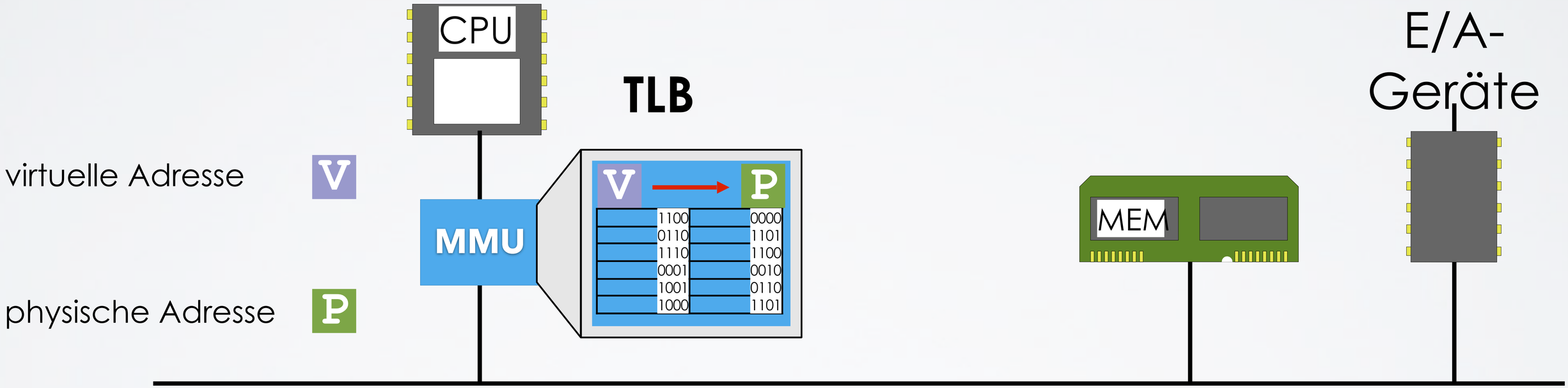
- teilweise durch Prozesse (mit eigenem Adressraum) im User-Mode
- großenteils aber fest in Betriebssystem integriert

Mikrokernbasiert

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

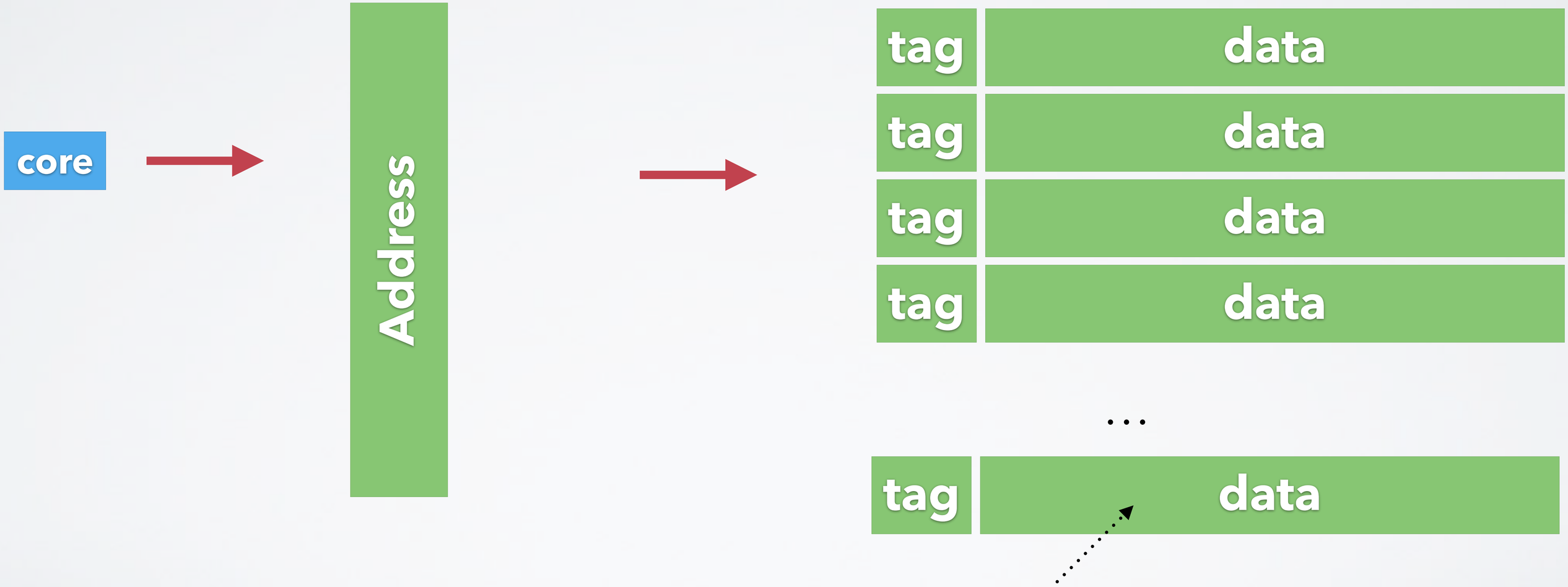
- Elementare Techniken
- **Virtueller Speicher (Paging)**
  - Anliegen - Begriffe - Vorgehen
  - Adressumsetzung, Hardware, Lazy Copying, …
  - Betriebsmittel Hauptspeicher: Seitenersetzung (Arbeitsmengenmodell)
  - Speicherobjekte
  - Integration der Einzelkomponenten
- Caches

CACHES: TLB

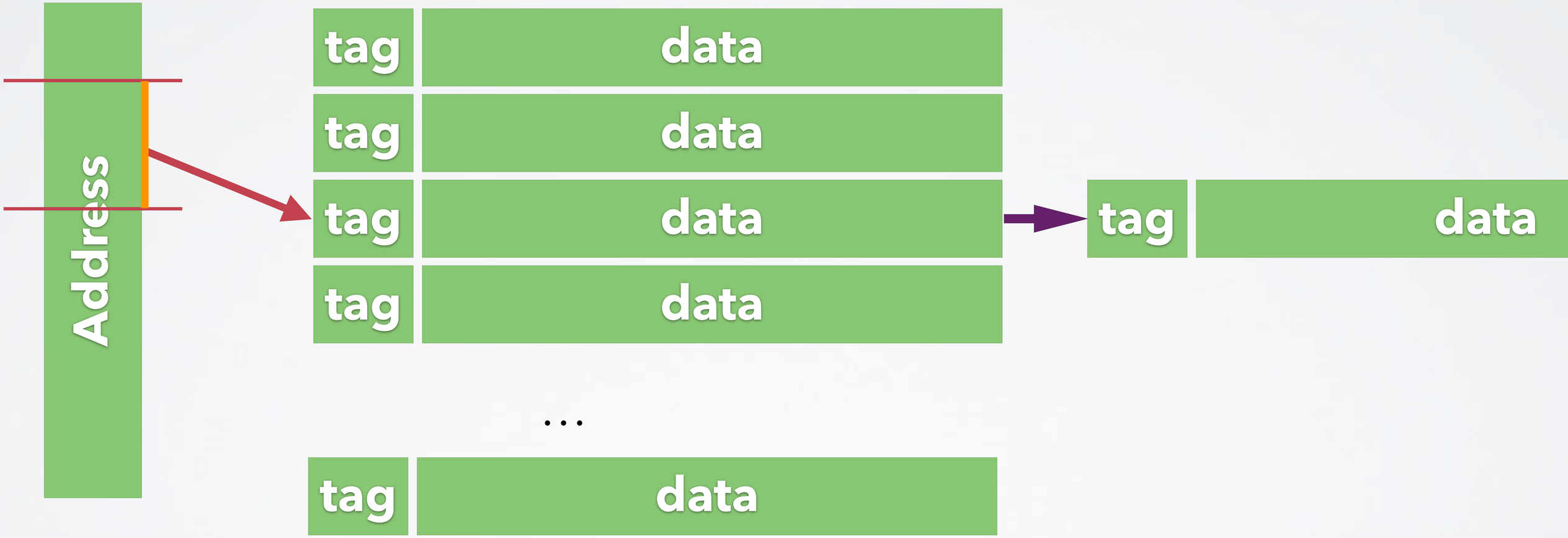

CPU
TLB
virtuelle Adresse
V
physische Adresse
P
MMU
V
P
1100
0110
1110
0001
1001
1000
0000
1101
1100
0010
0110
1101
MEM
E/A-
Geräte

CACHES: MEMORY

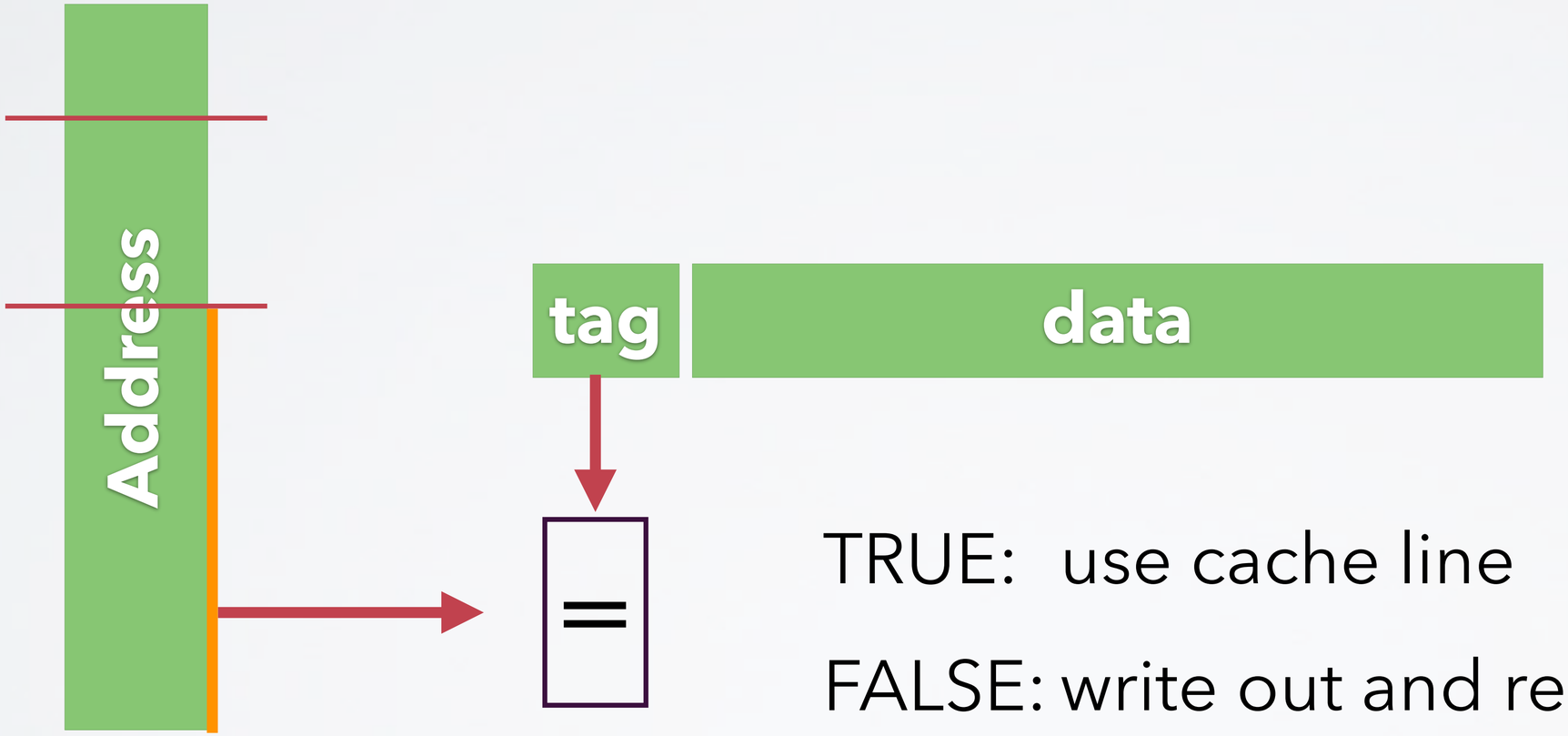
CACHES


core
Address
tag
data
tag
data
tag
data
tag
data
...
tag
data
"cache line"

CACHES: INDEX



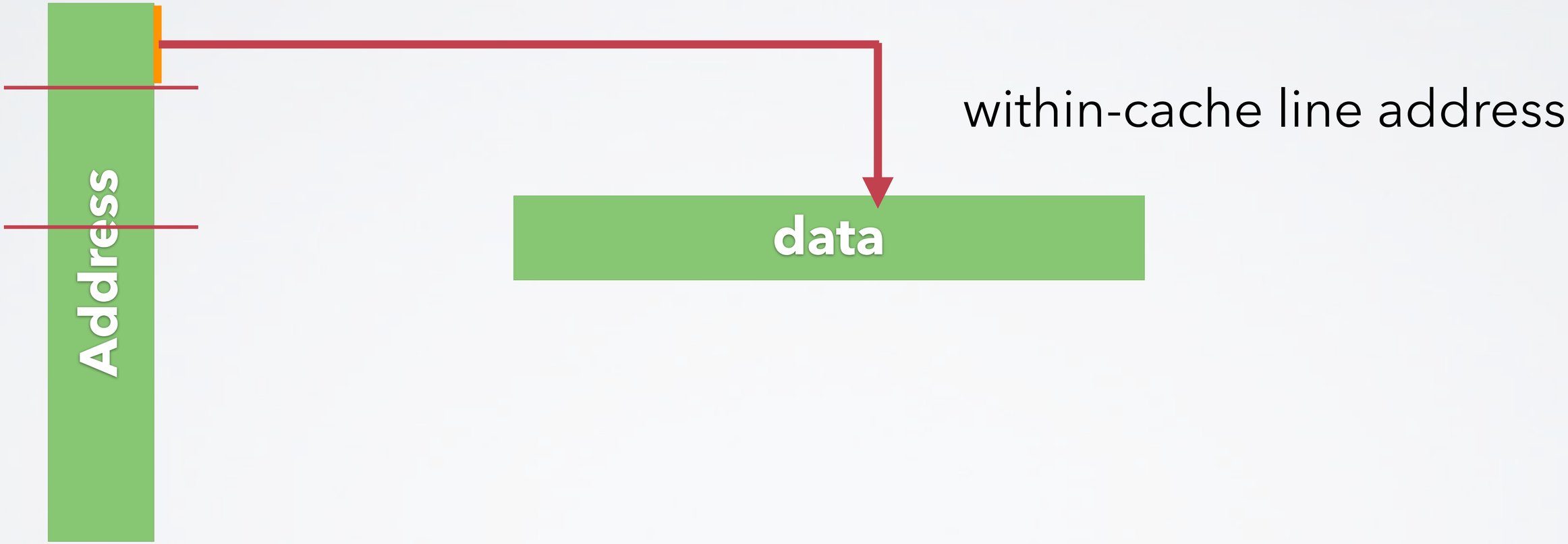
CACHES: TAG



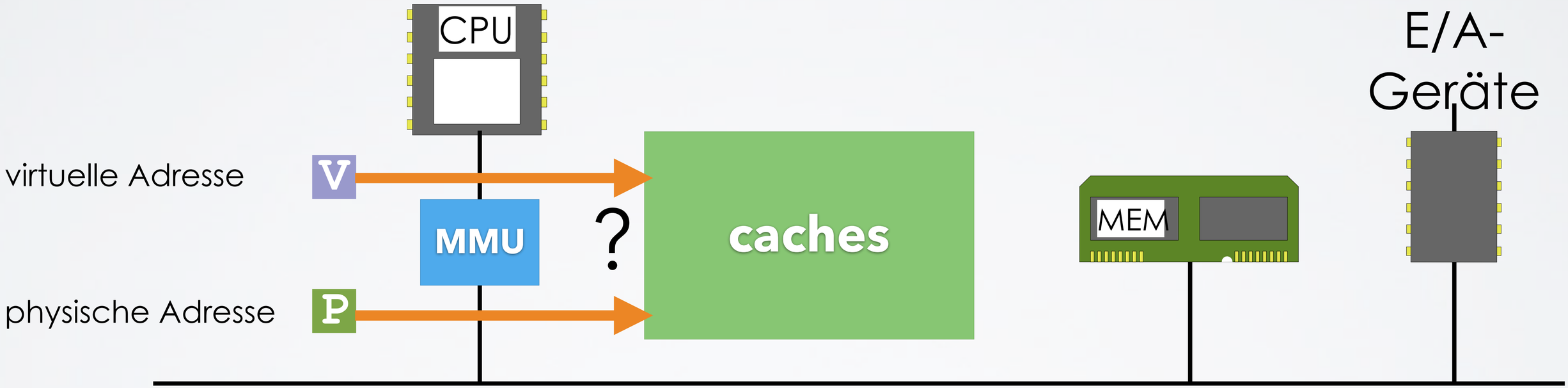
TRUE: use cache line

FALSE: write out and reload cache line

CACHES: THE LAST FEW BITS



VIRTUELL ./. PHYSISCH ?



NEUES

- viele neue Speichertechnologien
- NV-RAM

ZUSAMMENFASSUNG SPEICHER

- Ziele von virtuellem Speicher: Schutz, Abstraktion
- Adressumsetzung in der MMU
- Vorwärts-, inverse Seitentabellen, mehrstufig, TLB
- logischer Aufbau des Adressraums: Regionen
- Behandlung von Seitenfehlern
- Seitenersetzung vor Verwaltung des Hauptspeichers