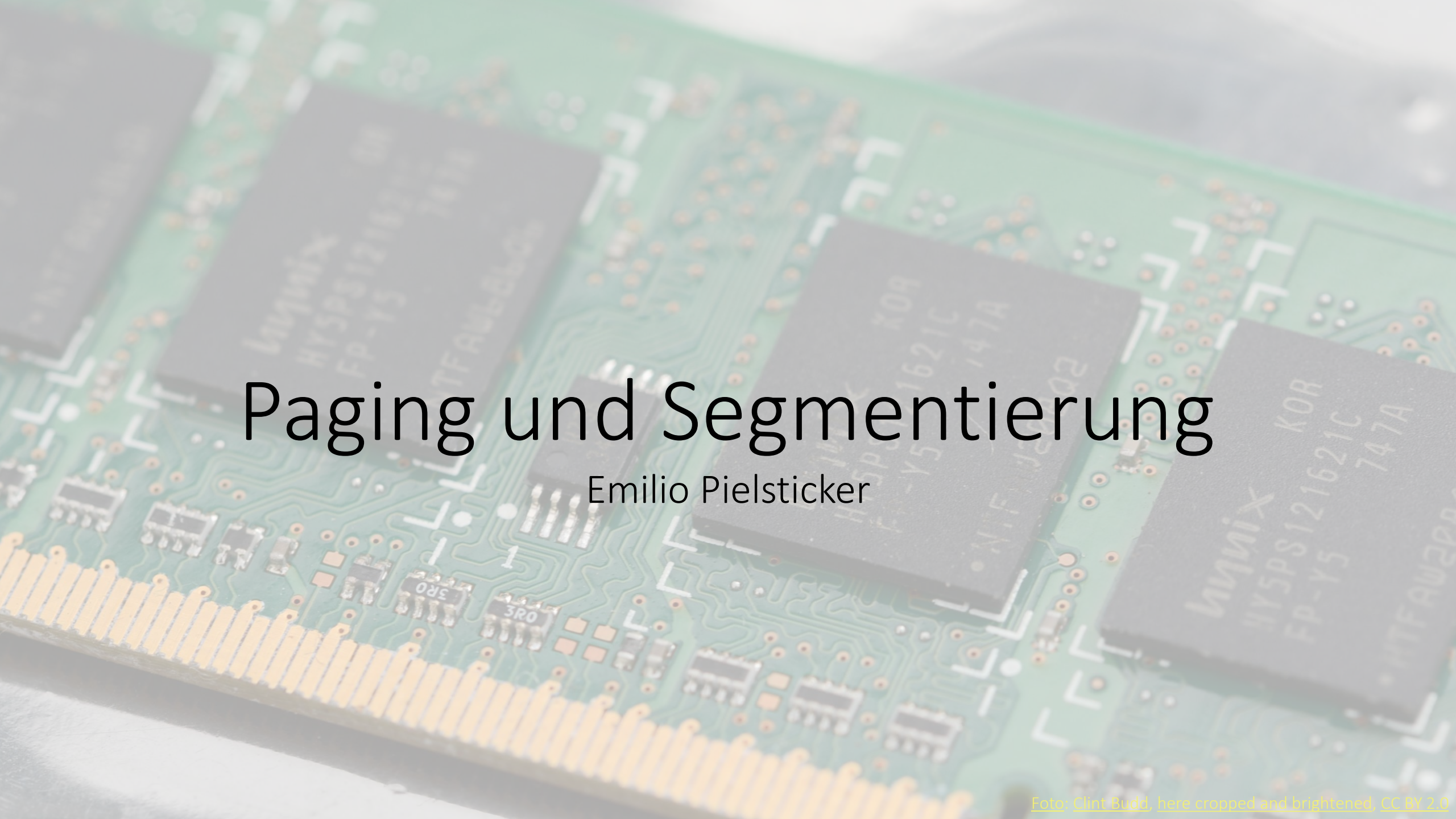
Paging und Segmentierung

Emilio Pielsticker



Adressraum

- Die Menge aller Adressen der Speicherzellen (die einem Prozess zur Verfügung stehen) wird als **Adressraum** des Prozesses bezeichnet
- Die skizzenhafte Darstellung eines Adressraumes als Balken wird **Speicherlandkarte** (engl. **memory map**) des Adressraumes genannt

Monolithisch / Defragmentiert:
Zusammenhängend

Fragmentiert:
Nicht zusammenhängend

Arbeitsspeicherverwaltung (Motivation)

Problem: In vielen realen System ist es notwendig, mehrere unterschiedliche Prozesse im Wechsel auszuführen:

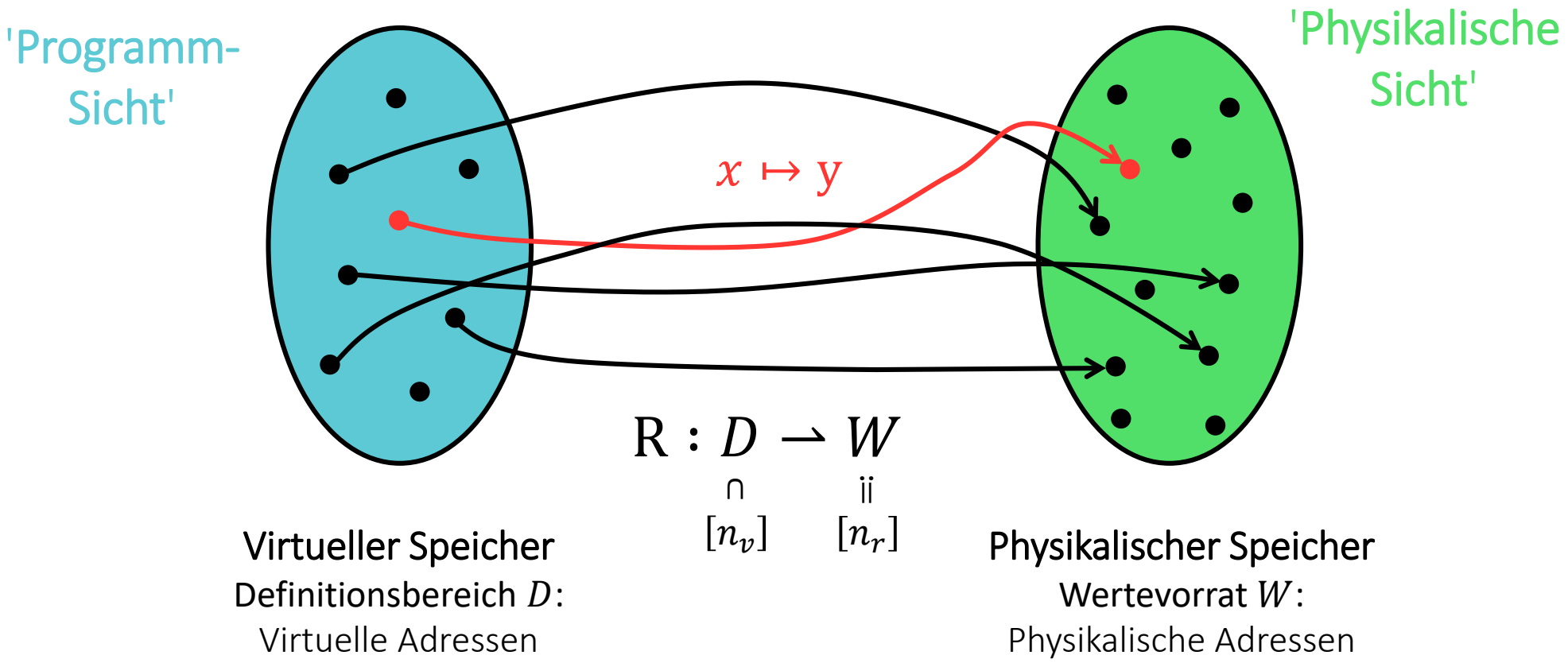
- Damit sich die Prozesse nicht 'in die Quere kommen' sollte jeder dieser Prozesse seinen eigenen Adressraum erhalten
- Prozesse sollen sich trotzdem auch Nur-lese-Daten teilen können, ohne das diese Daten für jeden dieser Prozesse kopiert werden müssen
- Auch bei der Verwendung geteilter Speichermedien soll jeder Prozess im eigenen linear-monolithischen Adressraum arbeiten können

Arbeitsspeicherverwaltung (Lösung)

Lösung: Organisation des Speichers mithilfe einer **Adressabbildung**:

- Die Menge der tatsächlichen Adressen der verwendeten Speichermedien heißt **realer**, **physikalischer Adressraum**
- Die im Programm verwendeten Adressen sind **virtuelle Adressen**
- Einer virtuellen Adresse wird durch eine rechtseindeutige Relation $\mathrm{R} \subset [n_v] \times [n_r]$ genau eine physikalische Adresse zugeordnet

Arbeitsspeicherverwaltung (Skizze)



R ist also eine partielle Funktion die sukzessive erweitert wird.

Arbeitsspeicherverwaltung

Problem:

- $\mathbf{R}$ selbst muss ebenfalls irgendwo gespeichert werden
- Wird jeder einzelnen virtuellen Adresse eine physikalische Adresse zugeordnet, wird $\mathbf{R}$ genau so groß wie der verwaltete Adressraum

Lösung: Virtueller und physikalischer Speicher wird in **Blöcke** geteilt:

- $\mathbf{R}$ ordnet Block auf 'virtueller Seite' jw. Block auf 'physikalischer Seite' zu

Methoden der Arbeitsspeicherverwaltung I

Arbeitsspeicherverwaltung durch Paging:

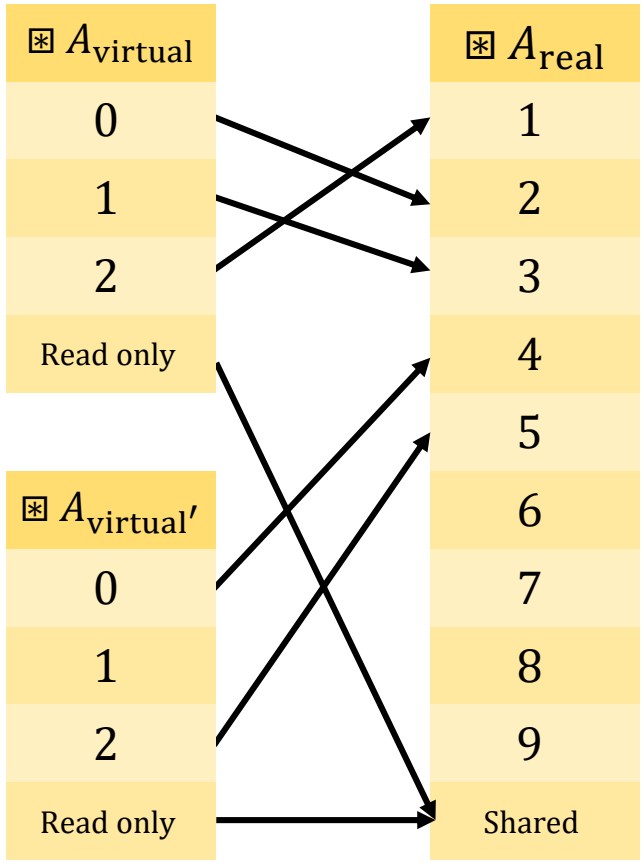
- Seiten im physikalischen Speicher heißen hier **Kacheln** / **Seitenrahmen**
- Blöcke im virtuellen Adressraum heißen hier **Seiten**
- Kacheln und Seiten sind gleich groß und besitzen eindeutige Nummern
- R wird als **Seitentabelle** eines jeweiligen Prozesses bezeichnet

Arbeitsspeicherverwaltung durch Segmentierung:

- Blöcke im physikalischen Speicher werden hier ebenfalls als Kacheln bezeichnet und können unterschiedliche Größen besitzen
- Blöcke im virtuellen Adressraum heißen hier Segmente
- R wird als **Segmenttabelle** bezeichnet (enthält zusätzlich jw. Größen)

Die Menge aller von einem Prozess eingenommenen Seiten bzw. Segmenten wird als der **Prozess im Speicher** bezeichnet.

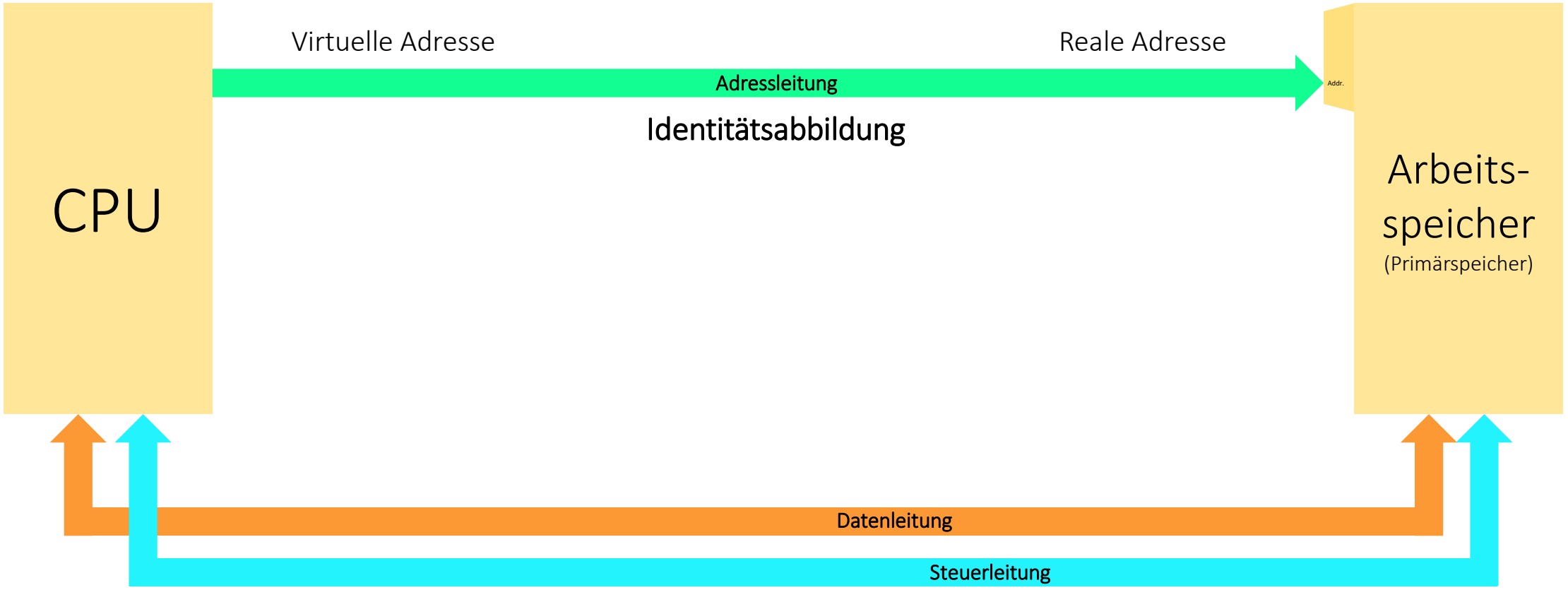
Paging (Skizze)



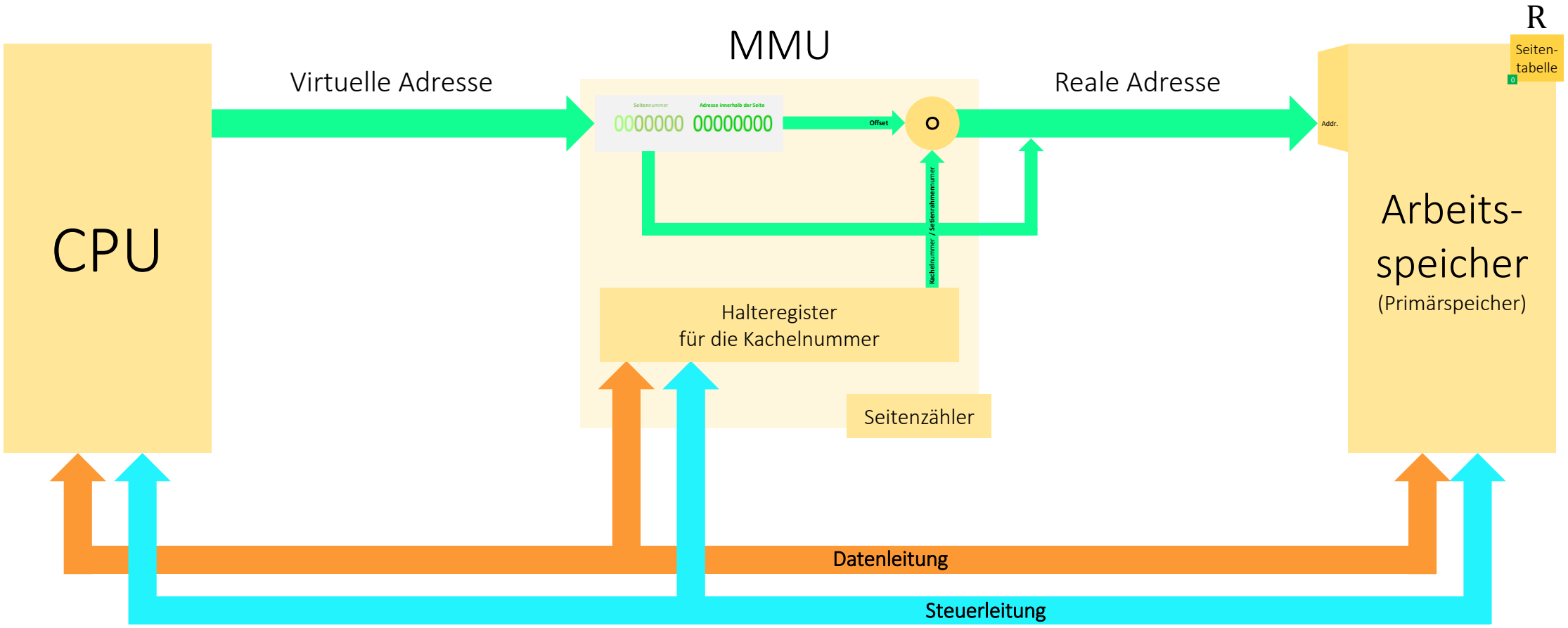
Speicherverwaltungseinheit (MMU)

- **Speicherverwaltungseinheit** (engl. **M**emory **M**anagement **U**nit, **MMU**) realisiert die Abbildung von **Seitennummer** auf **Kachelnummer**
- Für in R nichtvorhandene Seitennummern wird die Abbildung bei entsprechenden Zugriff erweitert → **Seitenzähler**
- Eine weitere Aufgabe der MMU besteht im sogenannten **Caching**:
  - Platzierung häufig verwendeter Seiten in schnellere Speichermedien
  - Platzierung selten verwendeter Seiten in langsame Speichermedien

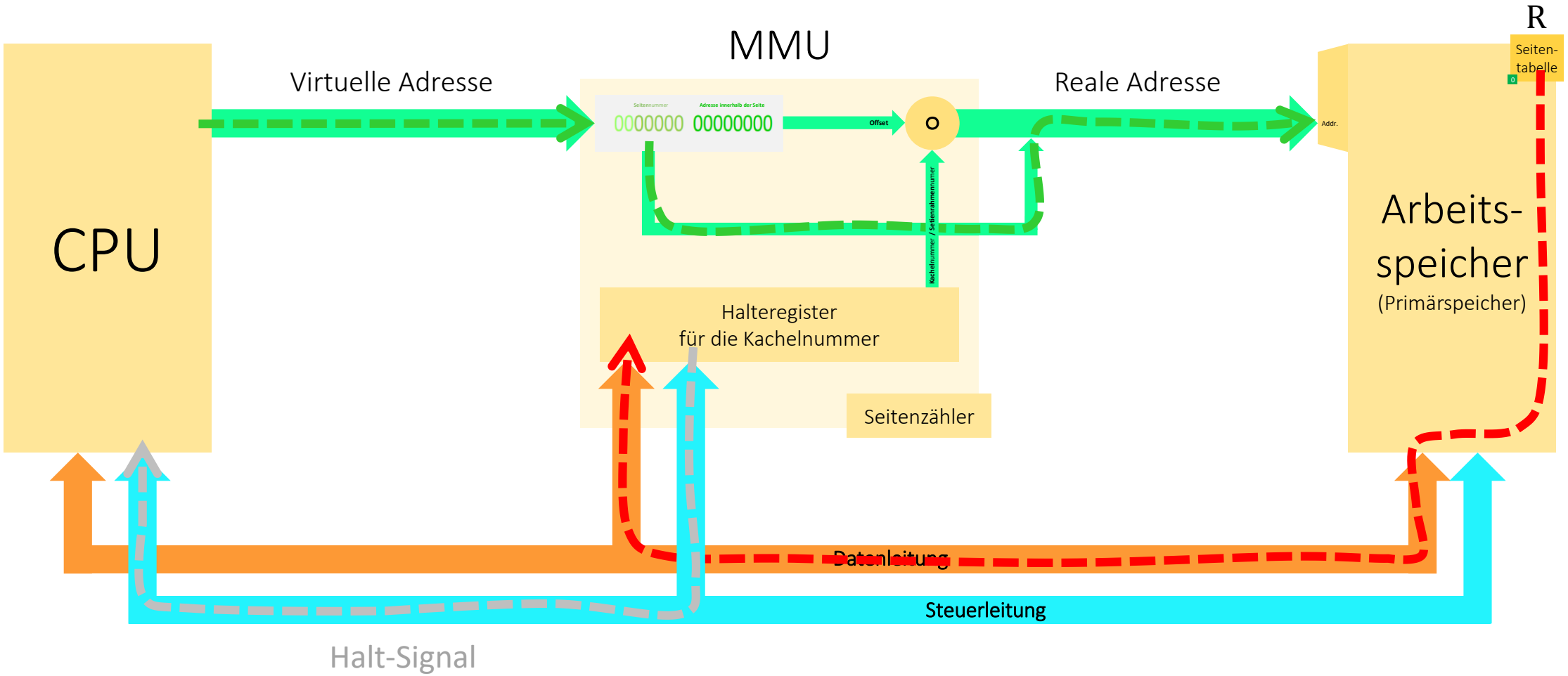
Speicherarchitektur ohne MMU



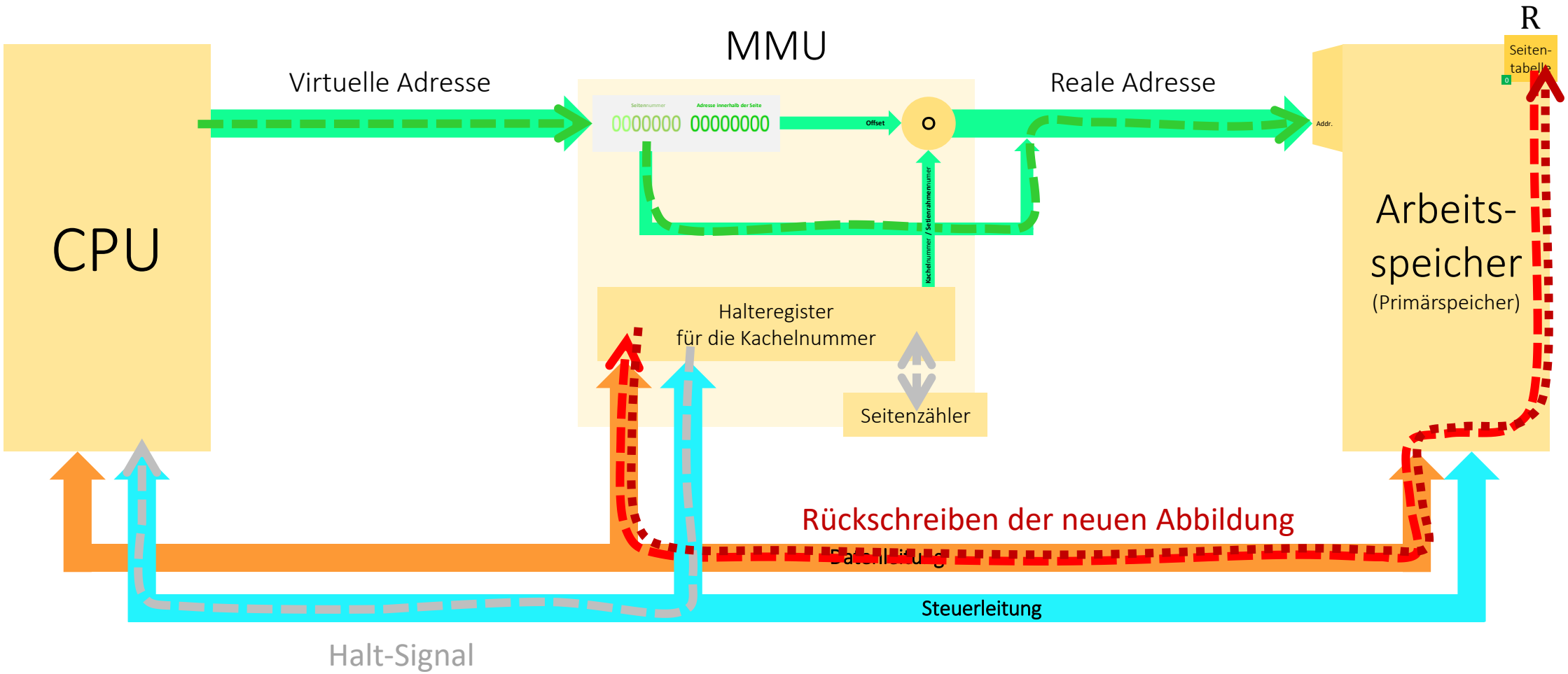
Aktivitätsträger mit MMU



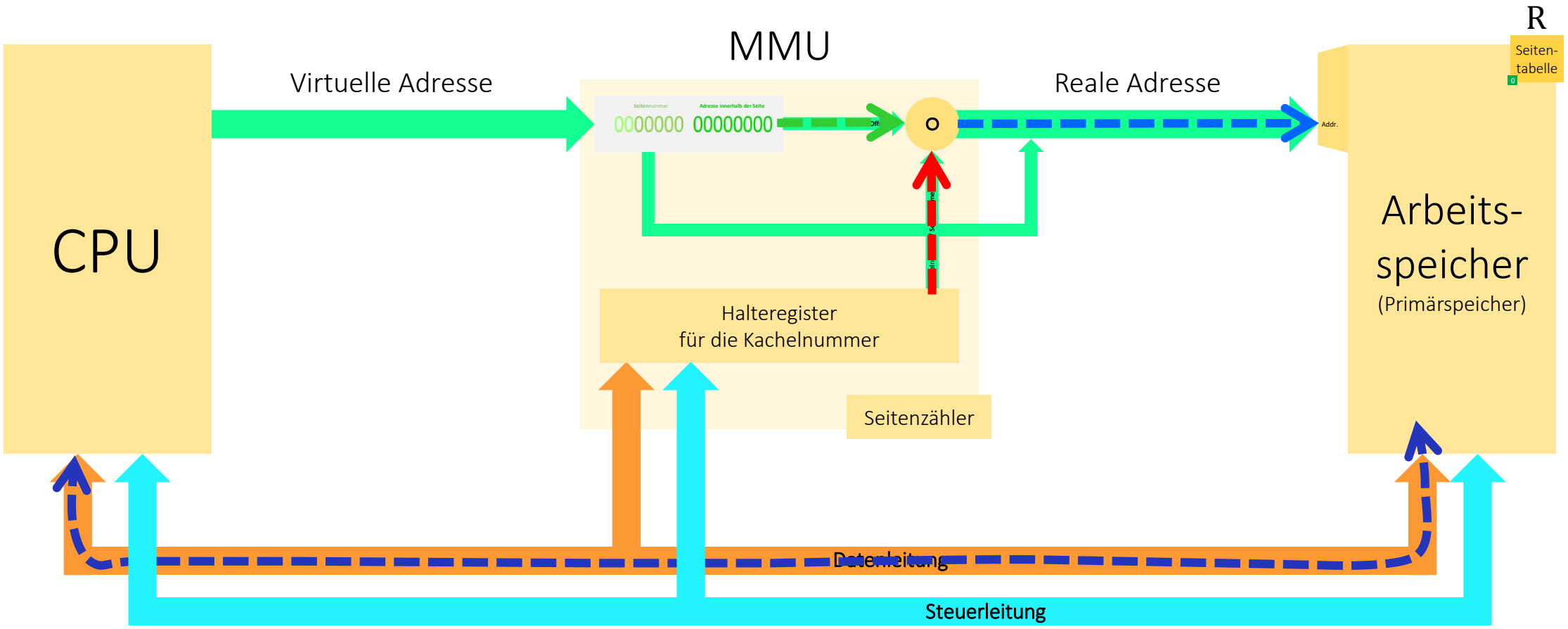
Holen der Kachelnummer (Falls vorhanden)



Sonderfall: Kachelnummer nicht vorhanden



Zusammensetzen der physikalischen Adresse I



Methoden der Arbeitsspeicherverwaltung I

Arbeitsspeicherverwaltung durch Paging:

- Seiten im physikalischen Speicher heißen hier **Kacheln** / **Seitenrahmen**
- Blöcke im virtuellen Adressraum heißen hier **Seiten**
- Kacheln und Seiten sind gleich groß und besitzen eindeutige Nummern
- R wird als **Seitentabelle** eines jeweiligen Prozesses bezeichnet

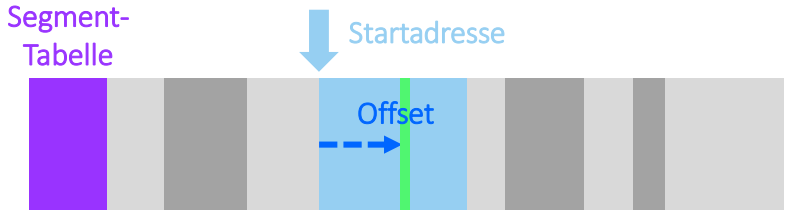
Arbeitsspeicherverwaltung durch Segmentierung:

- Blöcke im physikalischen Speicher werden hier ebenfalls als Kacheln bezeichnet und können unterschiedliche Größen besitzen
- Blöcke im virtuellen Adressraum heißen hier Segmente
- R wird als **Segmenttabelle** bezeichnet (enthält zusätzlich jw. Größen)

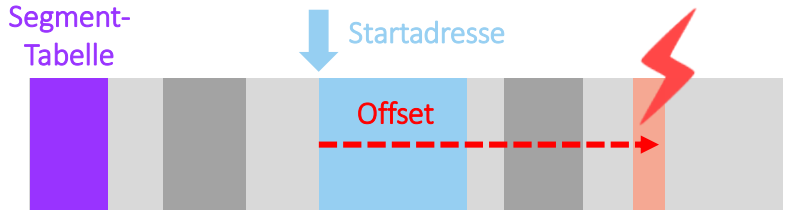
Die Menge aller von einem Prozess eingenommenen Seiten bzw. Segmenten wird als der **Prozess im Speicher** bezeichnet.

Segmentbasierte Adressberechnung

- Die physikalische Adresse erhält man bei der **segmentbasierten Adressberechnung** durch die Addition von Startadresse mit dem in der virtuellen Adresse enthaltenen Offset (Byteposition in der Seite)
- Ist der Offset größer als die **Segmentlänge** entsteht eine **Speicherschutzverletzung** (engl. **Segmentation-Fault**):

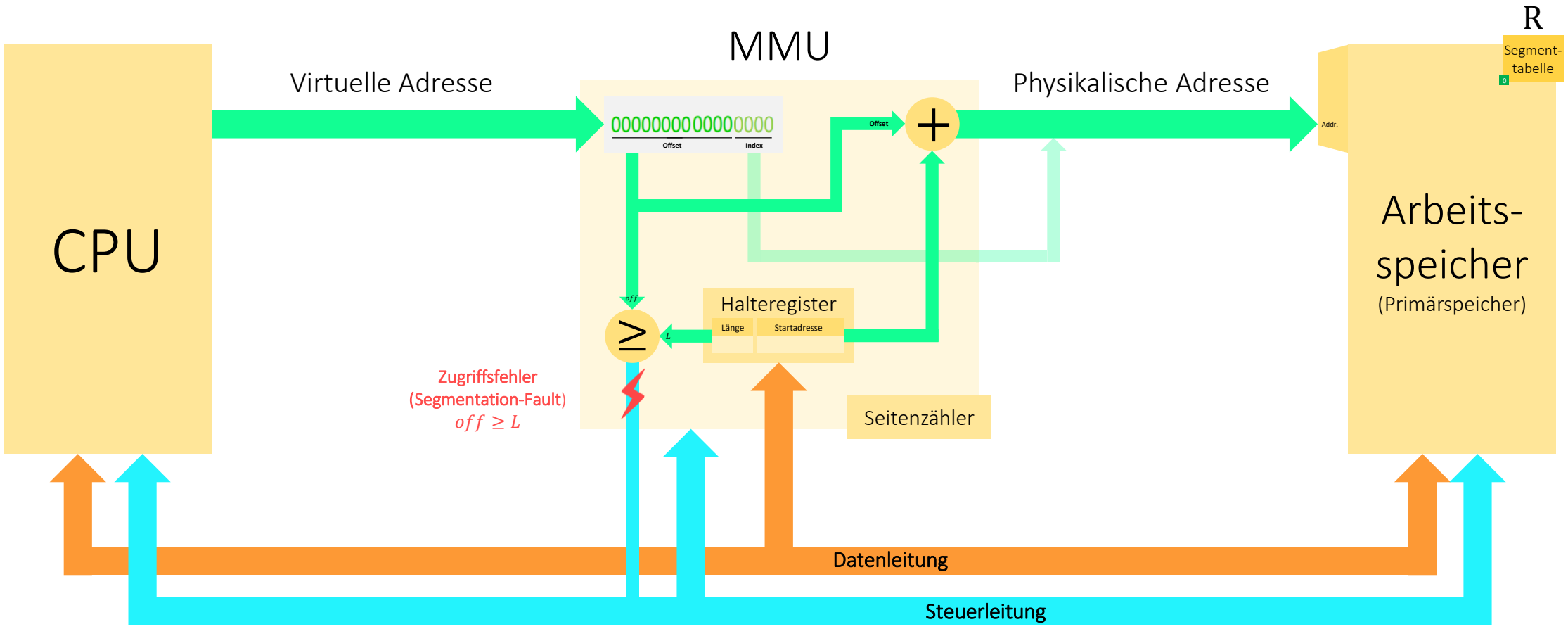


Speicherschutzverletzung:
Offset $<$ Länge

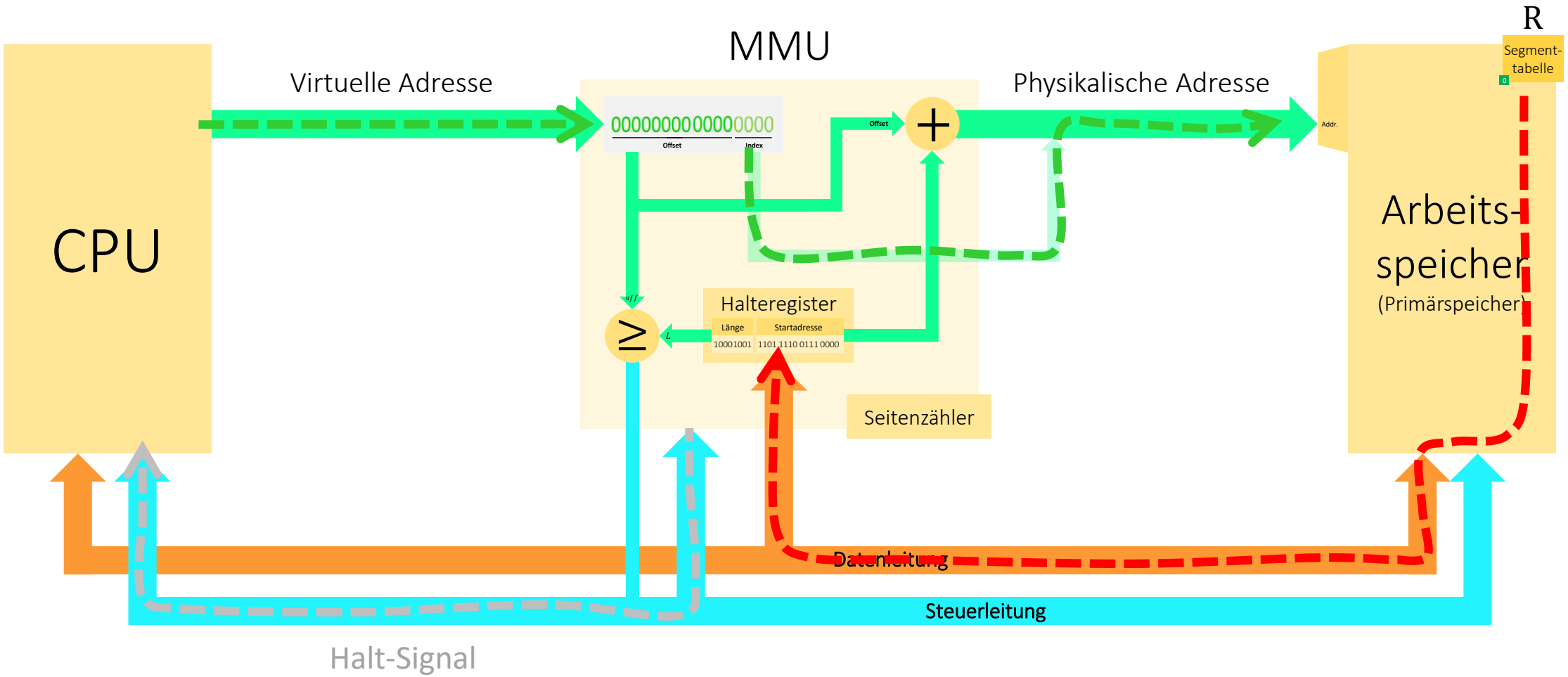


Speicherschutzverletzung:
Offset $\geq$ Länge

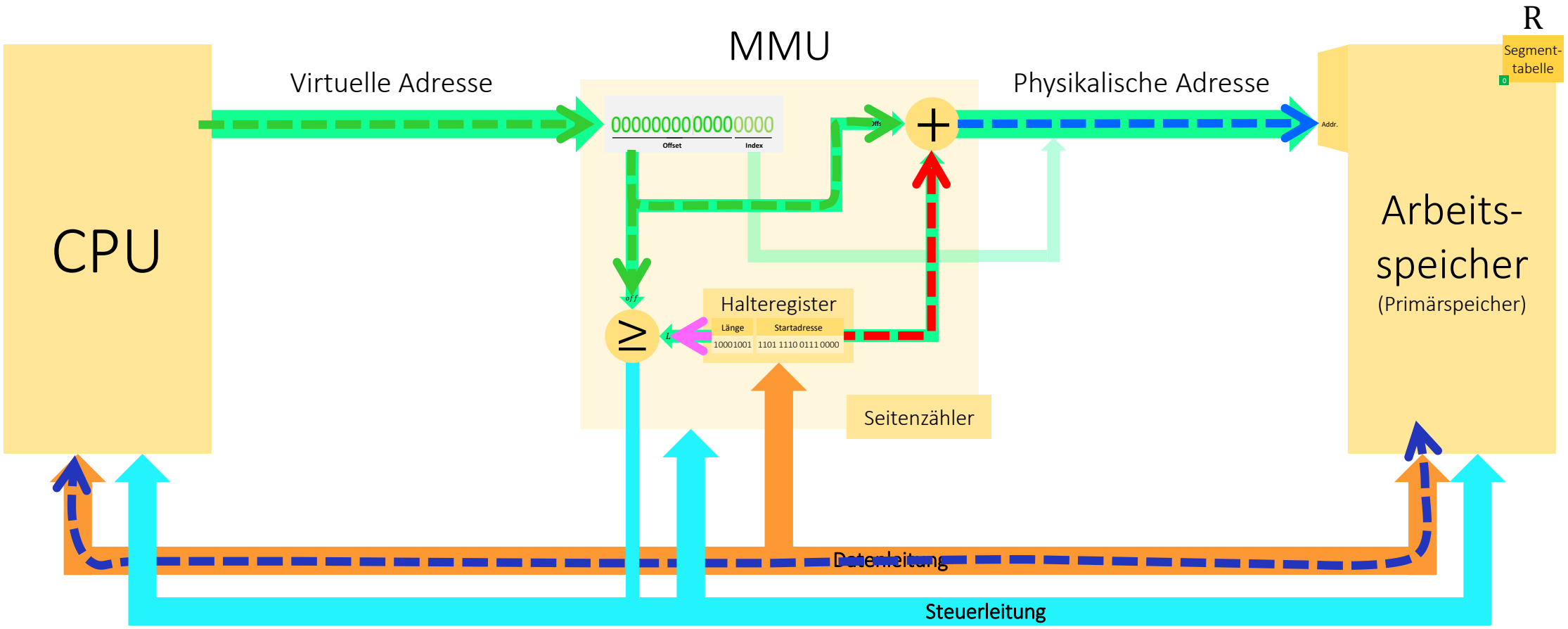
Segmentierung



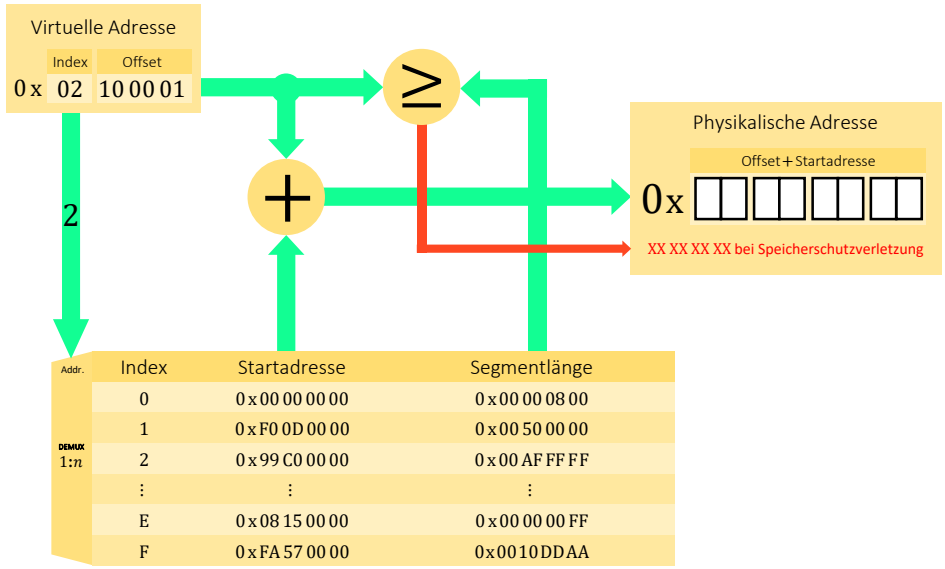
Holen der Segmentnummer (Falls vorhanden)

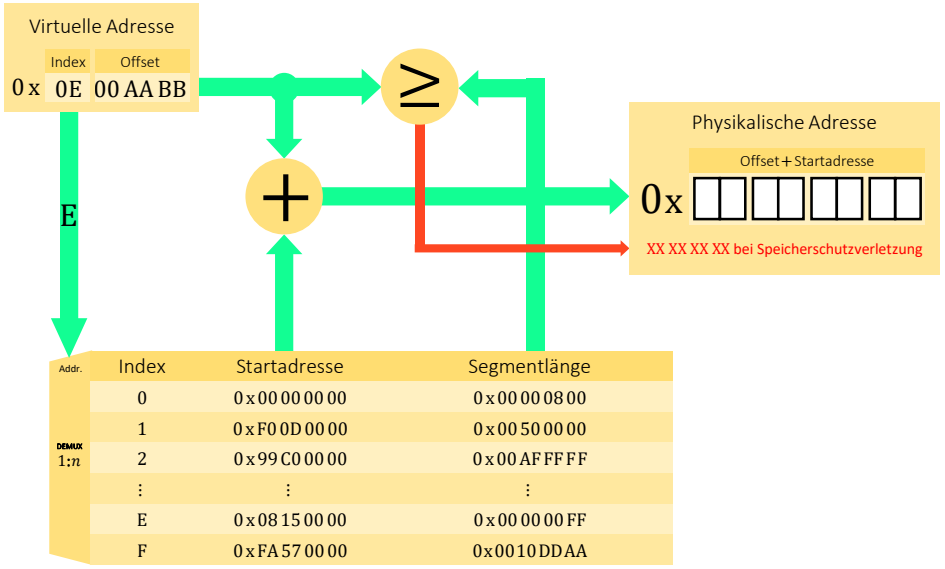


Zusammensetzen der physikalischen Adresse II



Segmentbasierte Adressberechnung (Beispiel)





Segmentadressierung mit Paging

Problem: Durch externe Fragmentierung kommt es bei der segmentbasierten Adressberechnung zu **Speicherverschnitt**

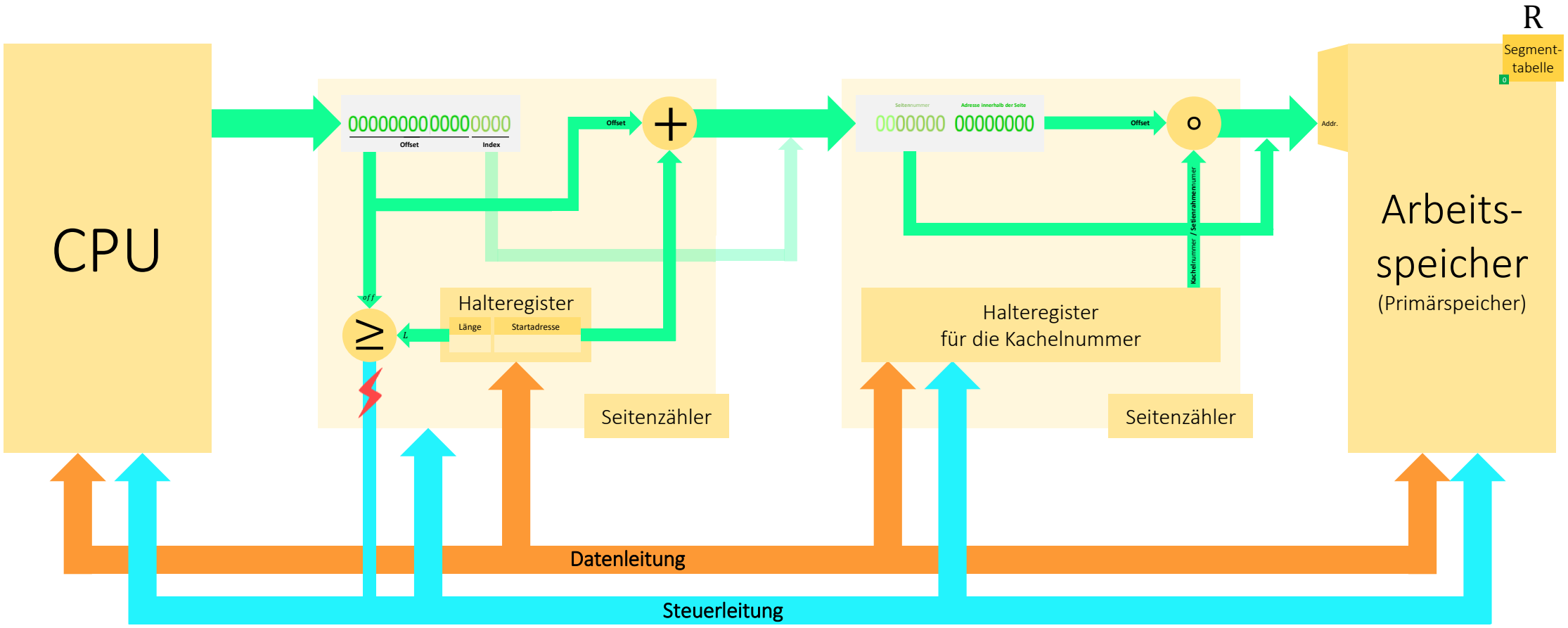
- Ungenutzte Lücken zwischen den Segment

Lösung: Komposition von segmentbasierter Adressberechnung und Paging

$$R_P \circ R_S$$

$$R_P[R_S[j]]$$

Segmentadressierung mit Paging (Skizze)



Übersetzungspuffer (TLB)

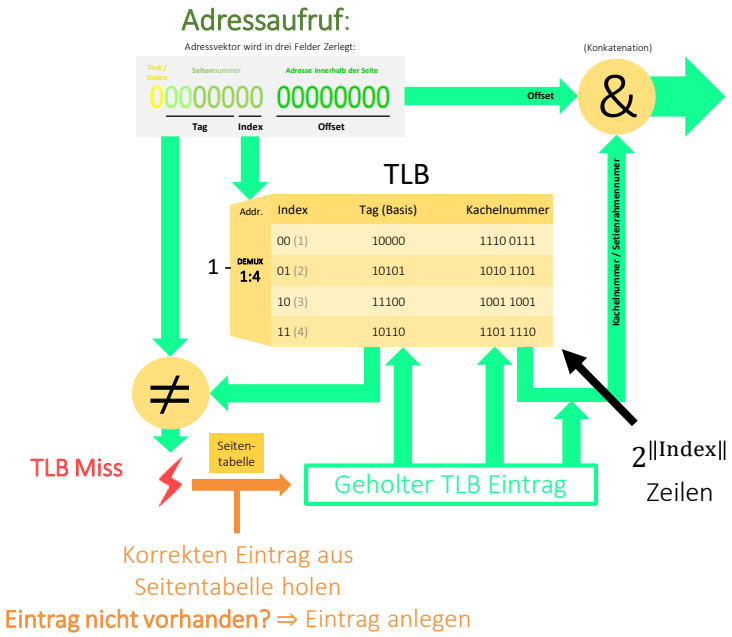
Problem: Bei jedem **Seitenaufruf** ist das lesen der Einträge aus R nötig:

- Jeder Seitenaufruf erfordert zwei Speicherzugriffe!

Lösung: Zuletzt verwendete Einträge der Seitentabelle lassen sich im **Übersetzungspuffer** (engl. **A**ddress **T**ranslation **M**emory, kurz **ATM** bzw. **T**ranslation **L**ookaside **B**uffer, kurz **TLB**) ablegen und nach einem Teil der Seitennummer indizieren → Aufteilung der Adresse: Index, Tag und Offset

- Bei Aufruf wird geprüft, ob Kachelnummer in TLB enthalten ist: **TLB-Hit**
- Kachelnummer ist nicht in der Zeile des TLB enthalten: **TLB-Miss** ⚡

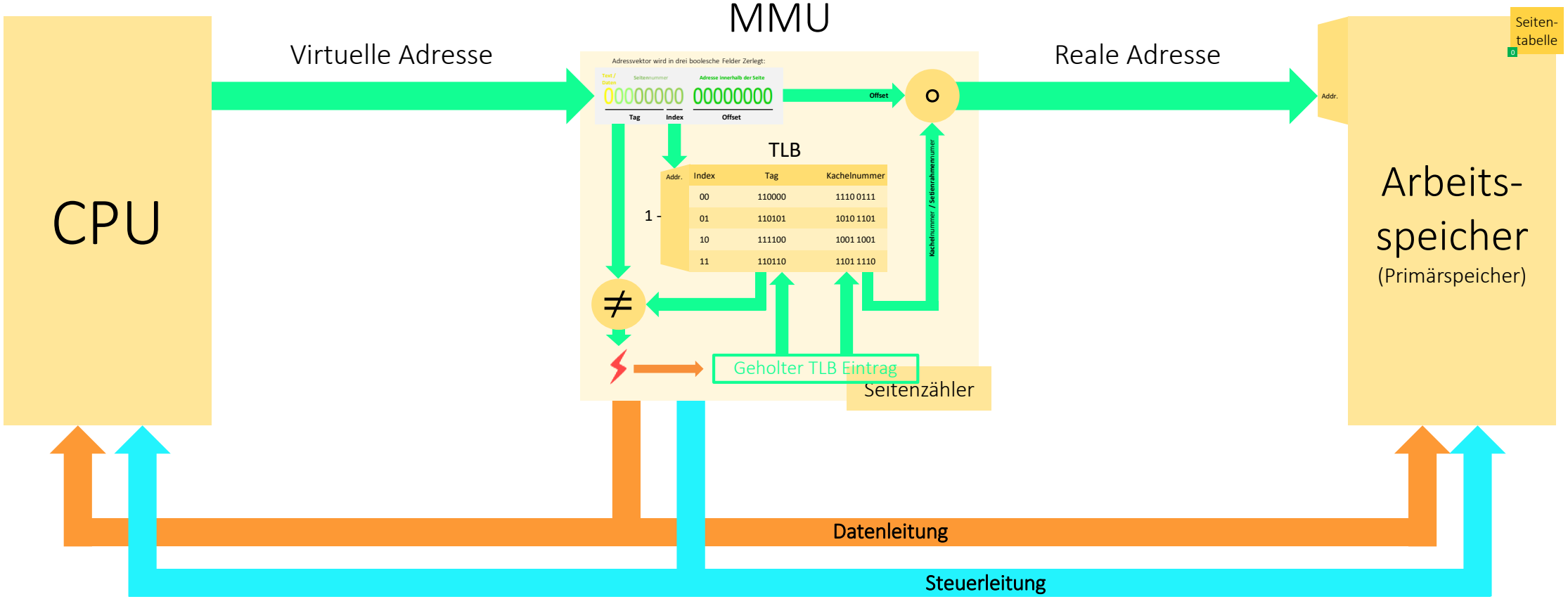
Übersetzungspuffer (Skizze)



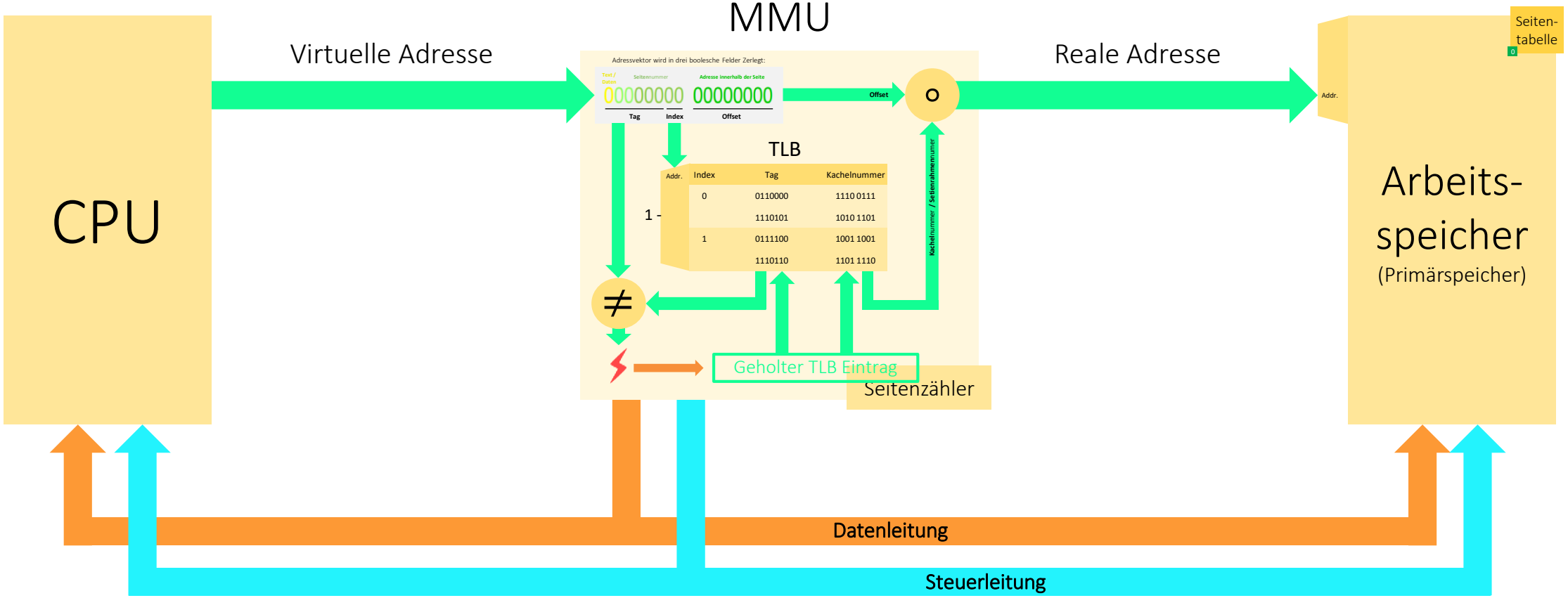
Übersetzungspuffer (Beispiel)

Zyk.	Gefordert: Tag	Index	≠	Basis für 1. Index	Basis für 2. Index	Basis für 3. Index	Basis für 4. Index
1	10000	00 (1)	TLB Hit	10000	10101	11100	10110
2	10101	01 (2)	TLB Hit	10000	10101	11100	10110
3	01101	10 (3)	⚡	10000	10101	01101	10110
4	10110	11 (4)	TLB Hit	10000	10101	01101	10110
5	11000	11 (4)	⚡	10000	10101	01101	11000
6	01101	10 (3)	TLB Hit	10000	10101	01101	11000
7	01101	00 (1)	⚡	01101	10101	01101	11000
8	01101	11 (4)	⚡	01101	10101	01101	01101

Direkte Abbildung mit TLB (Ein Tag pro Index)

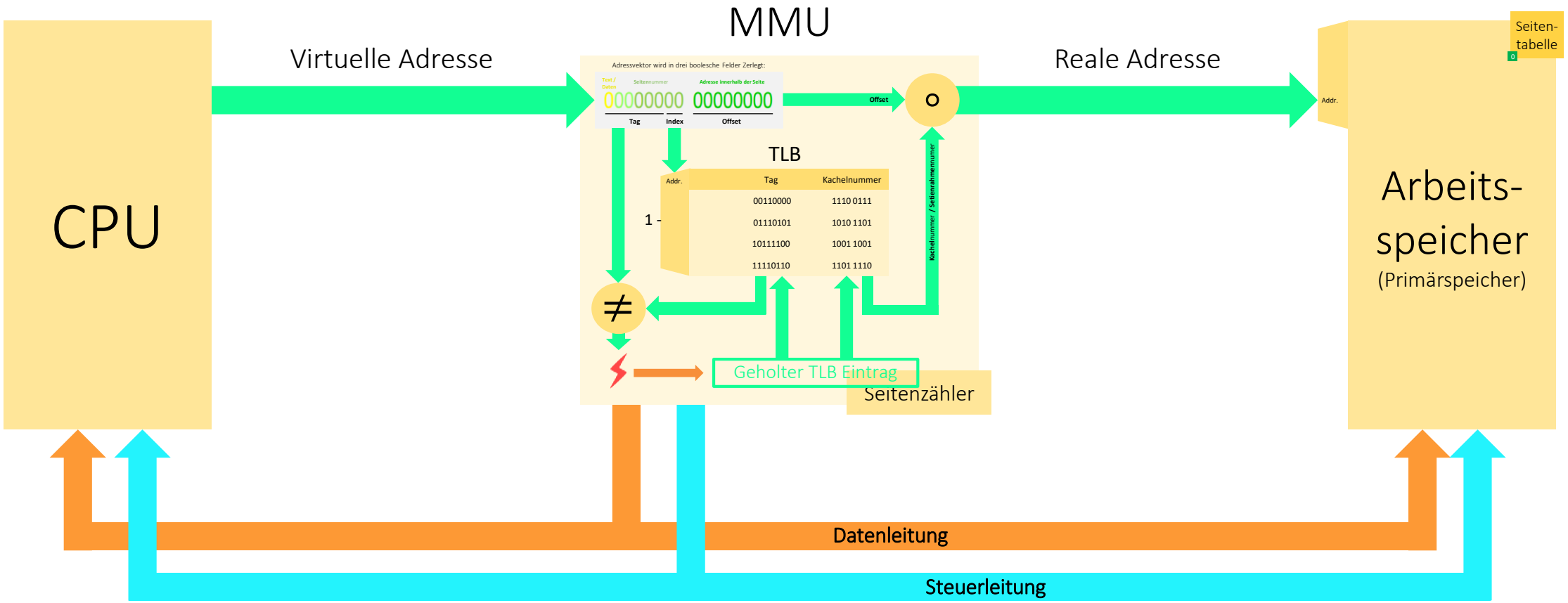


Mengenassoziativer Speicher (j Tags pro Index)



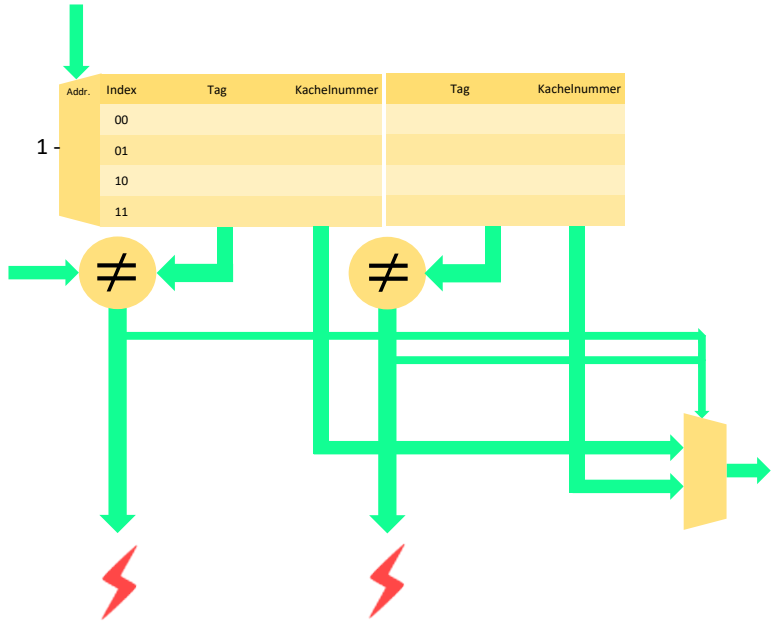
Problem: Welche Kachelnummer (mit Tag) wird ggf. ersetzt? → ERSETZUNGSPROBLEM

Vollassoziativer Speicher (Ohne Index)



Problem: Welche Kachelnummer (mit Tag) wird ggf. ersetzt? → ERSETZUNGSPROBLEM

Implementierung: 2-Weg assoziativ-Speicher



Abbildungsarten (Zusammenfassung)

- **Identitätsabbildung:** Seite entspricht Kachel: $\mathrm{id}(A_{\mathrm{virtual}}) = A_{\mathrm{real}}$
- **Direkte Abbildung ohne TLB** (engl. **D**irect **M**apping, kurz **DM**): MMU greift auf die Seitentabelle / Segmenttabelle im Arbeitsspeicher zu
- **Direkte Abbildung mit TLB:** MMU lädt (falls vorhanden) Kachelnummer aus dem TLB. Andernfalls wird diese aus dem Arbeitsspeicher geladen
- **Mengenassoziativer Speicher (MA):** Wie direkte Abbildung mit TLB, aber mit der Fähigkeit, zu jedem Index mehrere Tags im TLB halten zu können
- **Vollassoziativer Speicher (VA):** Wie MA-Speicher, wobei $\|\mathrm{Index}\| = 0$ gilt