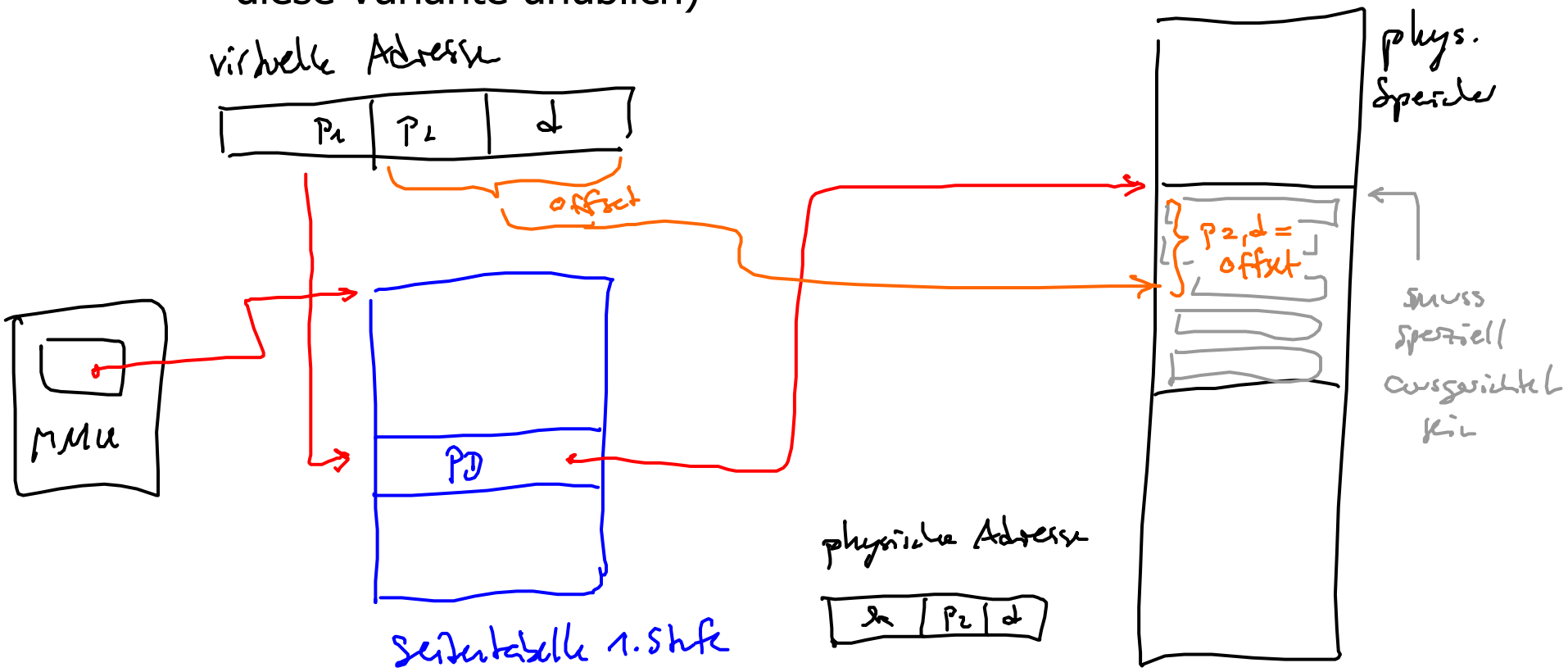
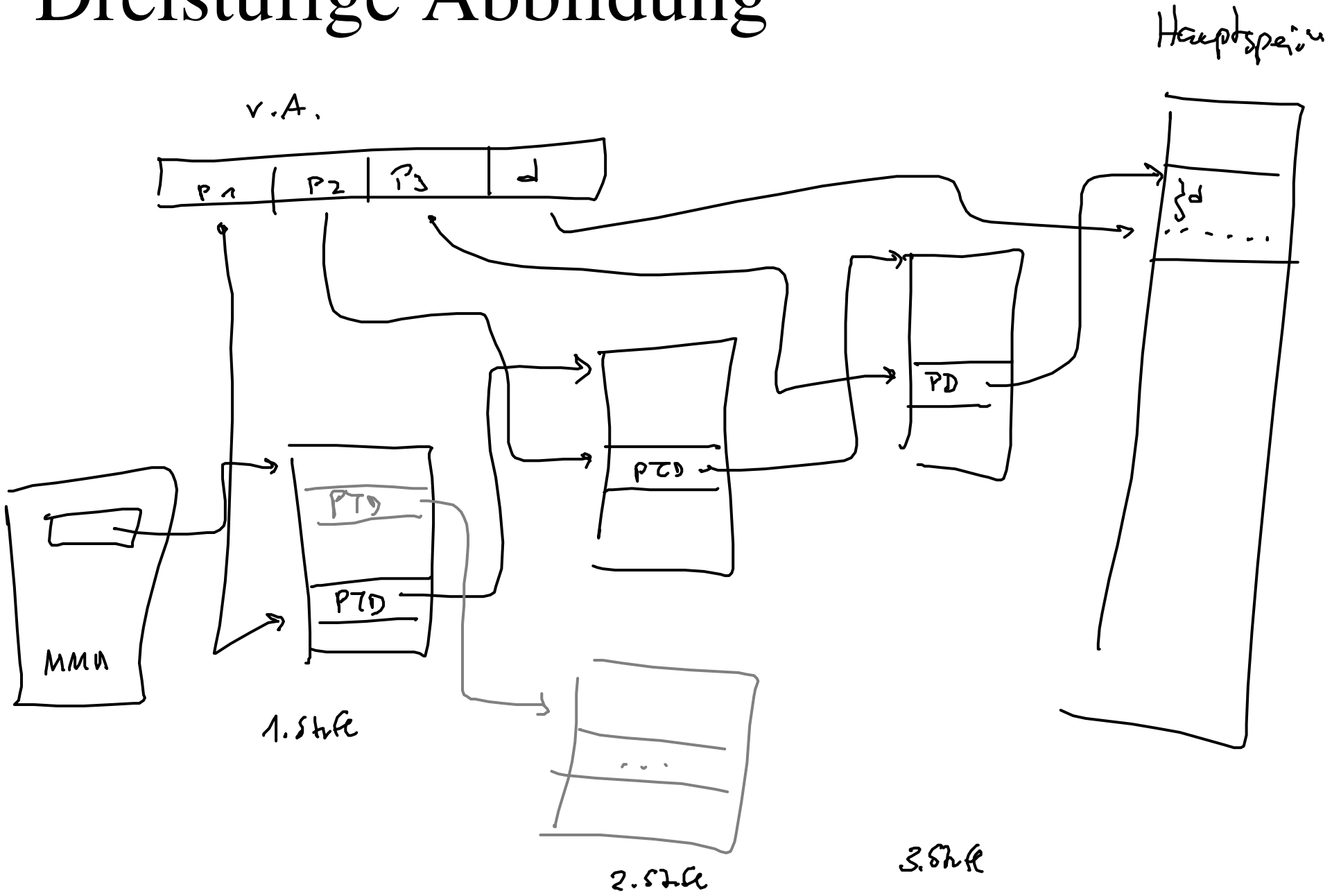
Mehrstufige Adressumsetzung

- Beispiel für zweistufige Abbildung mit Page Descriptor in der ersten Stufe
  - Hauptspeicherseite muss besonders ausgerichtet sein (deswegen diese Variante unüblich)



Dreistufige Abbildung



Rückblick und Ausblick 2.10.2008

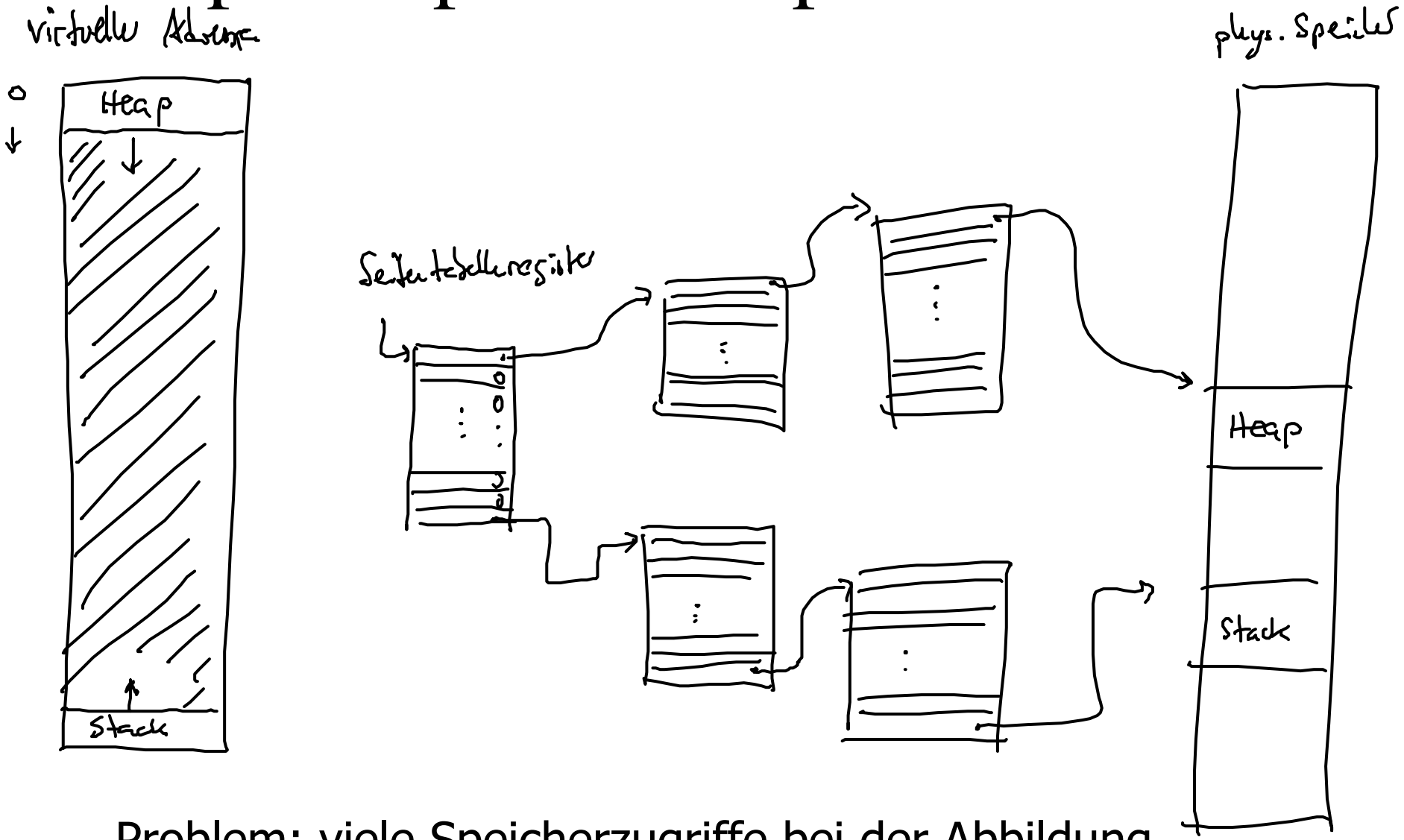
- Freitagsvorlesung: ULIX-Hardware und der ULIX-Emulator
- Freitagsübung: Implementierung von einfachem seitenbasierten virtuellem Speicher

- Heute: Dynamische Seitenersetzung
- Morgen: Feiertag

Diskussion

- Der Vorteil mehrstufiger Adressumsetzung liegt in der enormen Speicherersparnis
  - Große ungenutzte Teile des Adressraums können durch Nulldeskriptoren nahe der Wurzel früh ausgeblendet werden
  - Gerade bei typischen Anwendungen wird die Lücke zwischen Heap und Stack platzsparend abgebildet
- Beispiel: dreistufiges Verfahren, nur Heap und Stack im virtuellen Speicher, Deskriptoren der Größe 8 Byte,
  - Bei 128 Einträgen pro Tabelle benötigt man 1 Kbyte pro Tabelle
  - Man benötigt insgesamt 5 Kbyte (im Vergleich zu 8 Mbyte ohne mehrstufige Abbildung)
    - Eine Tabelle auf der ersten Stufe, zwei auf der zweiten und zwei auf der dritten Stufe
    - siehe nächste Folie...

Beispiel: Speicherersparnis

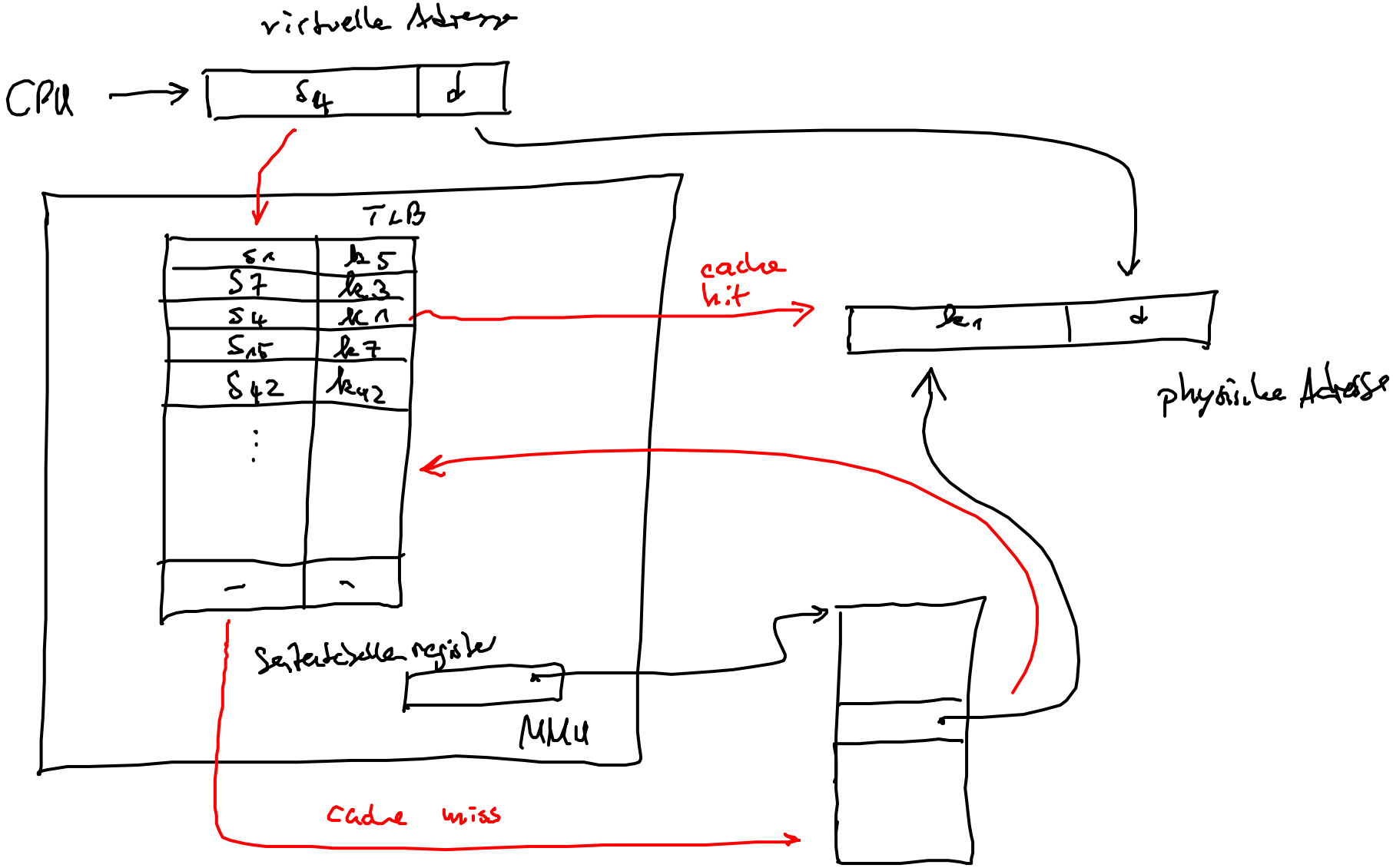


- Problem: viele Speicherzugriffe bei der Abbildung

Translation Lookaside Buffer (TLB)

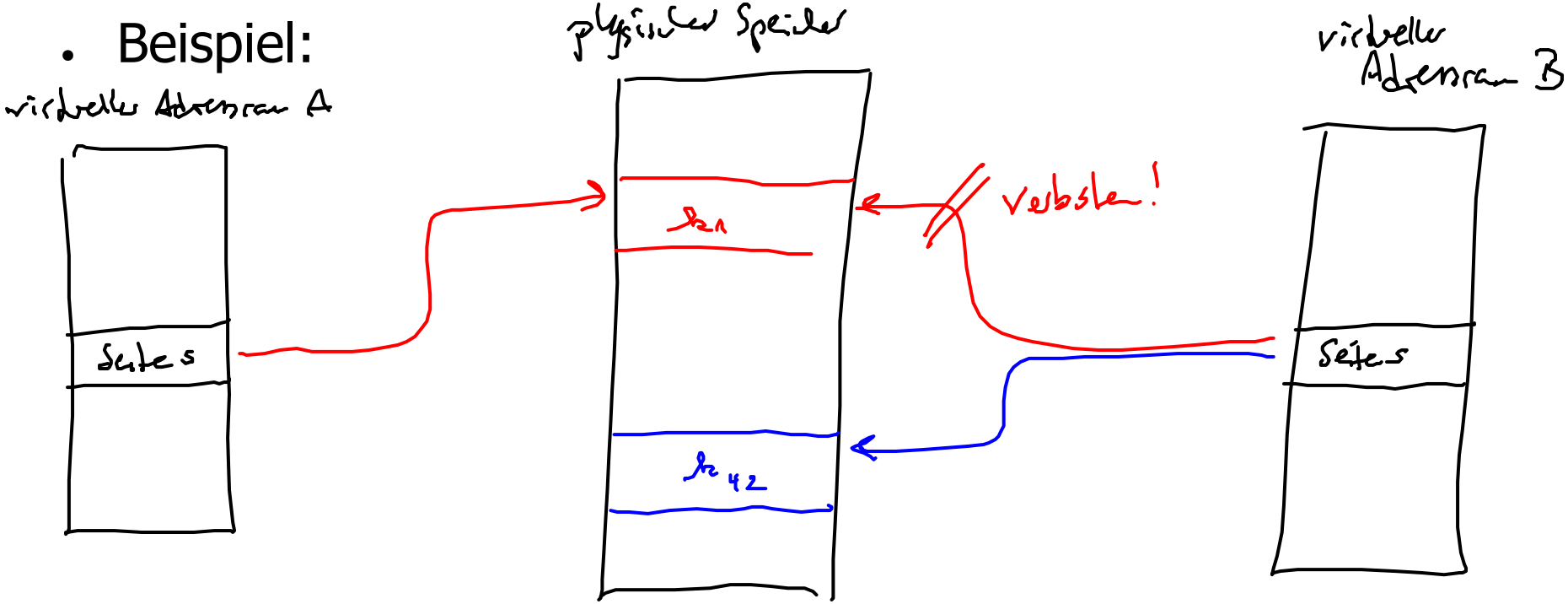
- Bei mehrstufigen Verfahren muss die MMU bei jeder Stufe einen kompletten Deskriptor aus dem Hauptspeicher laden
  - Bei 8 Byte Deskriptoren und L Stufen braucht man 8L Ladeoperationen!
- Lösung: Verwendung eines besonderen Caches
  - Translation Lookaside Buffer (TLB)
  - Cacht die Ergebnisse der Adressumsetzung:
    - Seitennummer zu Kacheladresse
- Durch Referenzlokalität kann man hohe Trefferraten erzielen
  - Trefferrate H 90 bis 98%
  - Verringert die durchschnittliche Zeit zur Adressumsetzung beträchtlich

Beispielablauf TLB



Adressraumwechsel

- Vorsicht bei Adressraumwechsel
  - Die virtuellen Adresse anderer Adressräume haben eine andere Bedeutung
  - Ohne zusätzliche Konzepte würde man ggf. falsch auflösen
  - TLB muß bei Adressraumwechsel invalidiert werden
- Beispiel:



TLB-Temperatur

- Nach dem Adressraumwechsel ist der TLB immer kalt
- Ansatz 1: Man kann die Wärme des TLB über Adressraumwechsel retten, wenn man
  - den Inhalt des TLB bei Adressraumwechsel sichern und
  - anschließend (beim Zurückwechseln) wiederherstellen kann
- Ansatz 2: Die Einträge im TLB werden mit einem Adressraumindikator versehen
  - Assoziative Suche im TLB wird auf Einträge mit demselben Adressraumindikator eingeschränkt
  - Man hat dadurch den TLB in mehrere logisch getrennte Sub-TLBs unterteilt
  - Vor allem interessant, wenn der TLB vom Chip-Hersteller vergrößert wird

Zusammenfassung

- Seitenbasierte virtuelle Adressierungstechniken erfüllen alle eingangs angegebene Forderungen
  - Virtueller Speicher ist homogen und unabhängig vom physischen Speicher
  - Im Gegensatz zu rein segmentbasierten Verfahren gibt es keine externe Fragmentierung
    - Allerdings (vernachlässigbare) interne Fragmentierung
  - Schutz auf Basis einer Seite ist für fast alle Fälle ausreichend
- Nachteile:
  - Beim Schutz vor Überläufen der Bereiche ineinander sind segmentbasierte Verfahren günstiger (siehe später)
  - Bei seitenbasierten Verfahren muss man Überläufe durch zusätzliche (meist Software-)Mechanismen erkennen und behandeln

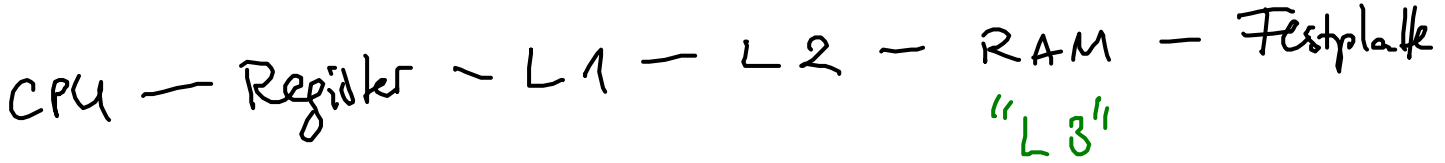
Übersicht

- Adressumsetzung (Einführung)
- Organisation von Adressräumen aus Anwendungssicht
- Seitenbasierter virtueller Adressraum
- Dynamische Seitenersetzung
  - Aus- und Einlagerungsstrategien für Hauptspeicherseiten
- Segmentbasierter virtueller Adressraum
- Implementierungsaspekte
- Physischer (nicht-virtueller) Adressraum

Dynamische Seitenersetzung

Idee: "Level 3 Cache"

- Seitenbasierte Verfahren besitzen eine zentrale Stärke: Das Ein- und Auslagern von Seiten auf den Hintergrundspeicher ist sehr einfach
  - Seitengröße möglichst gleich Blockgröße auf dem Hintergrundspeicher wählen
  - Hauptspeicher wird zum zusätzlichen "Level 3 Cache" für den Hintergrundspeicher (auf dem der virtuelle Speicher abgelegt ist)
    - Teile des Anwendungsadressraums werden im Hauptspeicher "zwischengespeichert"



Technische Realisierung

- Cache-Verwaltung geschieht über das P-Bit (in allen Deskriptoren vorhanden)
  - Bei P-Bit = 1 befinden sich die Deskriptoren bzw. der Seiteninhalt direkt im Speicher (Cache Hit)
  - Bei P-Bit = 0 liegt der gesuchte Inhalt auf dem Hintergrundspeicher (Cache Miss)
    - Erzeugt einen Seitenfehler-Interrupt: Inhalt muss aus dem Externspeicher in den Hauptspeicher eingelagert werden
      - Informationen über die Art des Fehlers stehen in Statusregistern der MMU oder einem Exception Frame auf dem Supervisor Stack zur Verfügung

Leistungsfähigkeit

- Wie gut funktioniert der Cache?
  - Zentrales Problem: Zugriffszeiten auf Hauptspeicher und Externspeicher sind sehr unterschiedlich!
    - Zugriffszeit Festplatte ca. 8 ms
    - Zugriffszeit Hauptspeicher ca. 8 ns
      - Unterschied: 10^6
  - Wenn Hauptspeicherzugriff 1 Sekunde dauert, dauert Externspeicherzugriff 11,5 Tage
  - Durchschnittliche Zugriffszeit auf den virtuellen Speicher ist

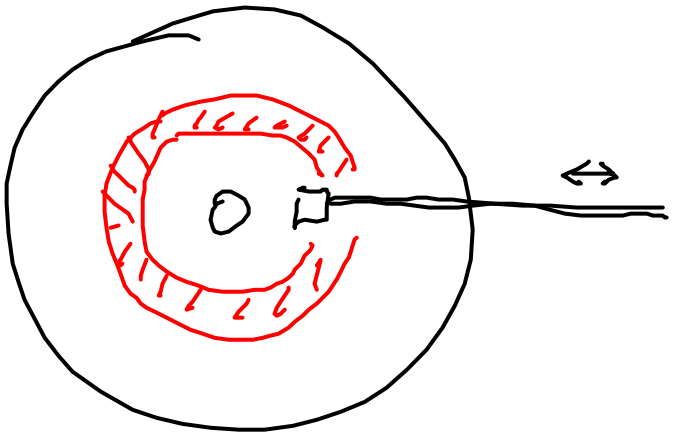
$$t_{VS} = (1-p)\,t_{HS} + p \cdot t_{SF}$$

p = Wahrscheinlichkeit eines Seitenfehlers

  - Zugriffszeit bei Seitenfehler wird dominiert von Zugriffszeit auf Externspeicher
- Folgerung: Wahrscheinlichkeit eines Seitenfehlers sollte sehr klein sein!

Technische Tricks

- Was kann man machen, um die durchschnittliche Zugriffszeit zu verkürzen?
  - Seitenfehler vermeiden!
  - Seitenauslagerungen einsparen
    - Wenn das Dirty-Bit nicht gesetzt ist, braucht man eine Seite nicht auszulagern!
  - Bei unterschiedlich schnellen Hintergrundspeichern: wähle die schnellste Platte für die Realisierung des virtuellen Speichers



Referenzstring

- Seitenfehlerrate kann in der Praxis relativ einfach im akzeptablen Bereich gehalten werden
  - Grund: Referenzlokalität
    - Besonders in prozedural strukturiertem Code
- Messen der Referenzlokalität: Benutzen Referenzstring
  - Beschreibt das Zugriffsverhalten eines Programms auf einzelne Seiten des Adressraums
  - Gibt für jeden Zugriff die Seite des Adressraums an
- Beispiel:

RS = 0, 0, 1, 1, 2, 1, 2, 2, 3, 1, 3, ...

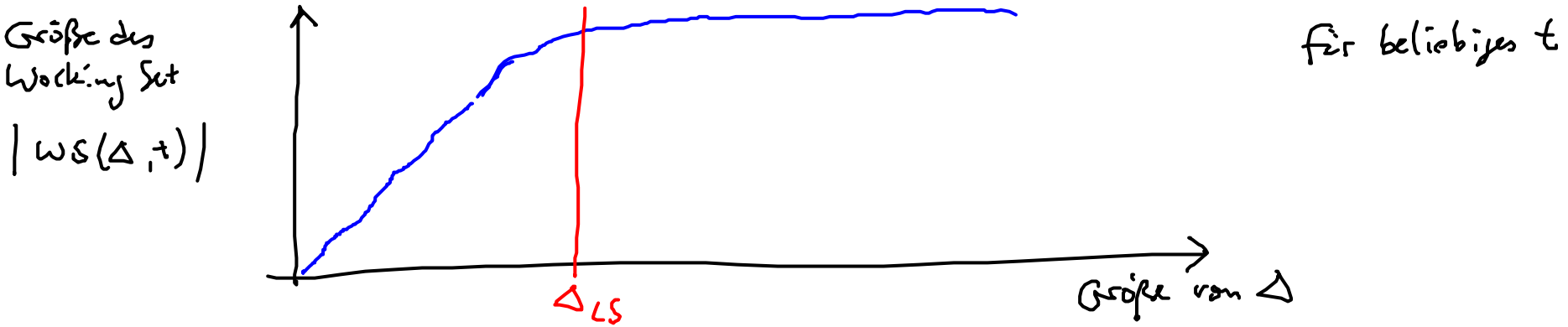
RS[i] = Seitennummer des i-ten Zugriffs eines Programms

Working Set

- Working Set = Menge der Hauptspeicherseiten, auf die innerhalb einer bestimmten Zeit zuletzt zugegriffen wurde

$$WS(\Delta, t) = \{ RS[t], RS[t-1], RS[t-2], \ldots, RS[t-\Delta] \}$$

- Größe des Working Set steigt ab einem gewissen Δ nicht mehr deutlich an (unabhängig von t):



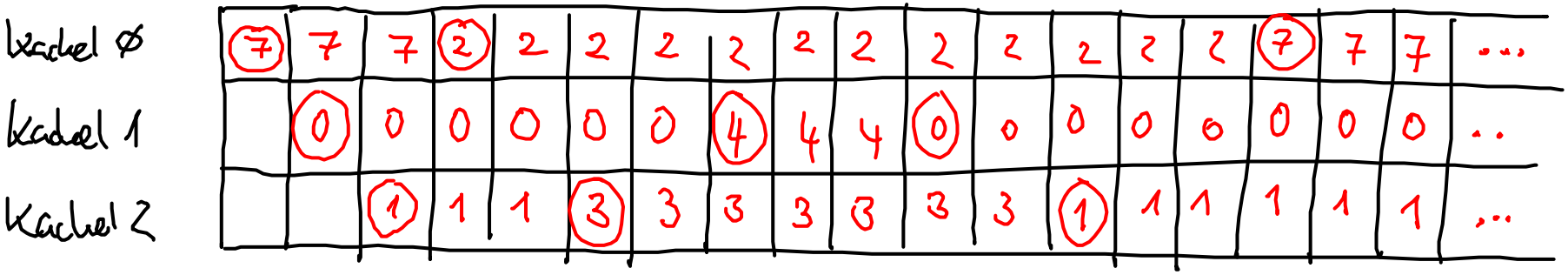
  - Über lange Zeit benutzt das Programm dieselben Seiten
    - Sehr viel weniger Seiten als im virtuellen Adressraum vorhanden
  - Lokalitätsmenge = Working Set mit optimalem Δ
  - Lokalitätsmenge sollte vollständig im Hauptspeicher liegen

Seitenverdrängung

- Wenn eine neue Seite eingelagert werden soll, muss ggf. eine andere Seite ausgelagert werden - welche?
  - Diese Frage beantwortet ein Seitenverdrängungsverfahren
- Vorüberlegungen:
  - Wenn für jedes Programm die Lokalitätsmenge im Hauptspeicher ist, dann ist die Seitenfehlerrate niedrig
  - Man sollte deshalb bevorzugt solche Seiten auslagern, die nicht zur Lokalitätsmenge eines Programms gehören
- Problem: Lokalitätsmenge kann sich dynamisch ändern, Änderungen sind schwer vorherzusehen
  - Wir kann man trotzdem (d.h. ohne Wissen der Lokalitätsmenge) sinnvolle Auslagerungskandidaten auswählen?

Optimale Verdrängung

- Bei einem gegebenen Referenzstring kann man für jedes Seitenverdrängungsverfahren die Qualität messen
  - Maß: Gesamtzahl der generierten Seitenfehler bei konstanter Kachelzahl
  - Was ist optimal?
- Beispiel mit drei Kacheln:
  - RS = 7 0 1 2 0 3 0 4 2 3 0 3 1 2 0 7 0 1 ...



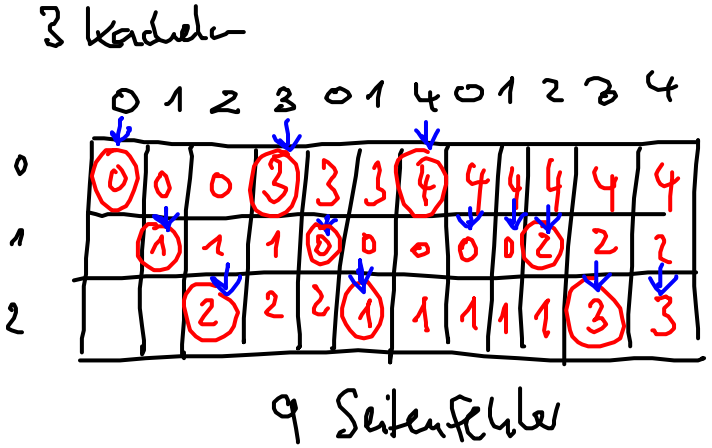
- Bemerkung: Zur Vereinfachung werden wir ab jetzt Teilfolgen mit gleichen Seitenreferenzen zu jeweils einem Wert zusammenfassen

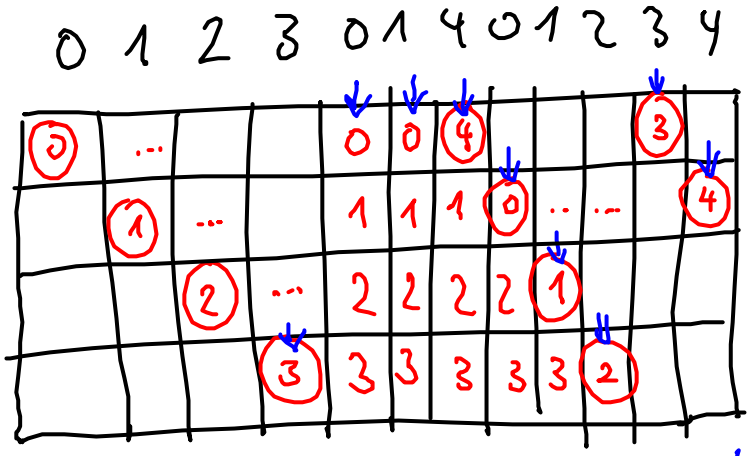
Belady-Algorithmus

- Optimales Auswahlkriterium:
  - Wähle diejenige Seite zur Verdrängung aus, die in Zukunft am längsten nicht referenziert wird
- Algorithmus ist nur theoretisch sinnvoll, kann aber gut als Vergleichsmaß für andere Algorithmen herangezogen werden
  - Statt die Zukunft vorherzusagen: einfache Approximation einer guten Seitenverdrängung wählen
  - Kandidaten:
    - FIFO
    - LRU
    - 2nd Chance
    - Clock-Algorithmus

First In First Out

- Auswahlkriterium:
  - Wähle diejenige Seite aus, die am längsten schon im Speicher ist
- Implementierbar durch Ringpuffer oder verkettete Liste
- Probleme:
  - Häufig referenzierte Seiten bleiben im Mittel genauso lange im Hauptspeicher wie selten referenzierte Seiten
  - Verfahren berücksichtigt also nicht die Lokalitätsmenge
  - Belady-Anomalie: Anzahl der Seitenfehler kann steigen bei vergrößerter Kachelzahl

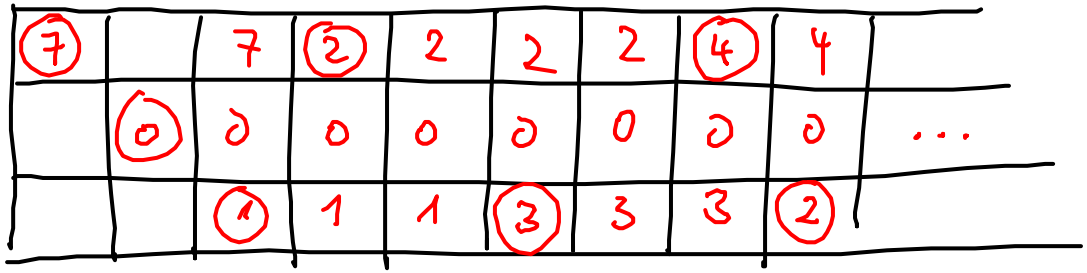




Least Recently Used (LRU)

- Auswahlkriterium:
  - Wähle diejenige Seite aus, die in der Vergangenheit am längsten nicht benutzt wurde
- Verfahren ist ziemlich gut, aber aufwendig zu implementieren
  - Garantiert bei Referenzlokalität eine nahezu optimale Seitenfehlerrate
- Beispiel mit 3 Kacheln:
  - RS = 7 0 1 2 0 3 0 4 2 ...

3 Kacheln

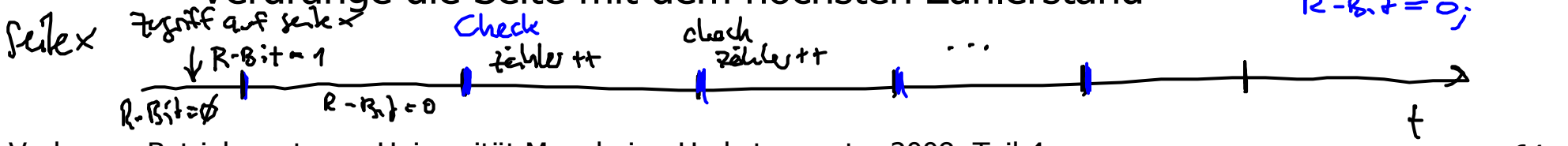


Implementierung von LRU

- Ideale Implementierung:
  - Speichere für jede Seite die letzte Zugriffszeit
  - Wähle im Verdrängungsfall die älteste Seite
  - Organisation der Seiten in doppelt-verketteter Liste
    - sortiert nach Zeitstempel
    - älteste Seite in konstanter Zeit auffindbar
    - Aktualisierung der Liste (Einfügen) aufwändig
- Näherungsverfahren:
  - Verwende R-Bits der Seitendeskriptoren
  - Jede Seite hat einen Zähler
  - Periodisch den Zähler einer Seite bei nicht gesetztem R-Bit erhöhen
  - Bei referenzierten Seiten R-Bit löschen
  - Verdränge die Seite mit dem höchsten Zählerstand

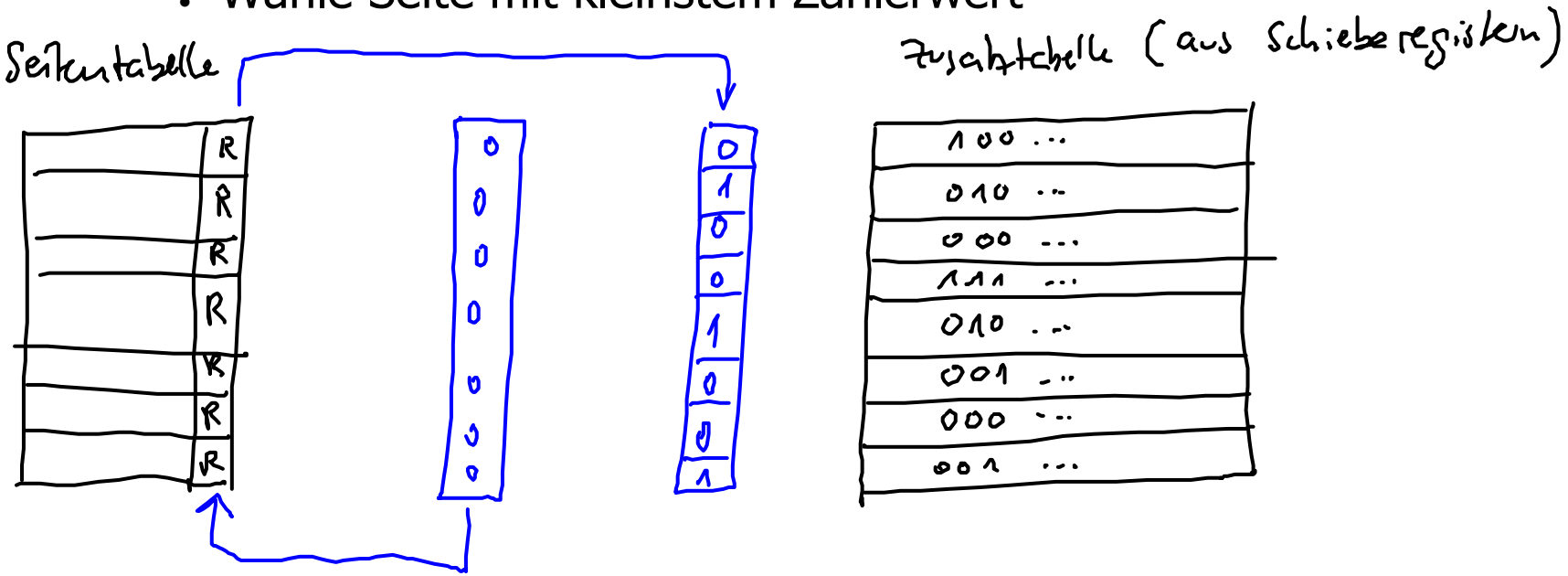
wenig referenzierte Seiten haben hohen Zählerstand

Check = falls R-Bit == 0 then Zähler++; R-Bit = 0;



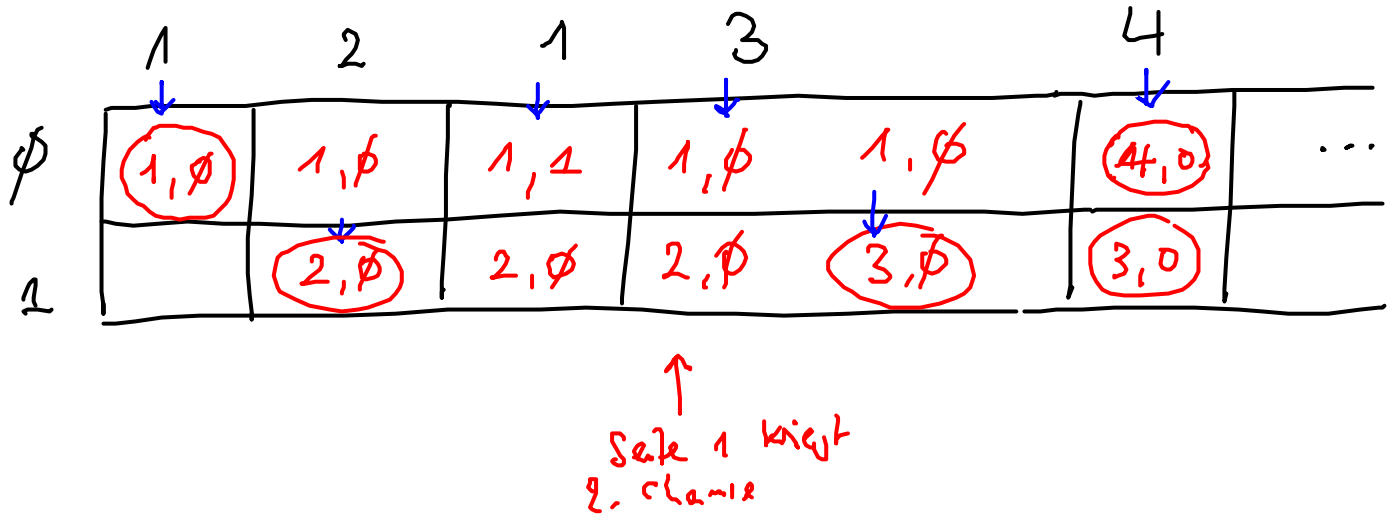
Alternatives LRU

- Alternatives Näherungsverfahren:
  - Benötigt Zusatztabelle
  - Vektor der Referenzstrings wird "von vorne" in die Zusatztabelle hineingeschoben
    - Referenzstrings anschließend zurückgesetzt
  - Sequenz von Bits auffassen als vorzeichenlose ganze Zahl
    - Wähle Seite mit kleinstem Zählerwert



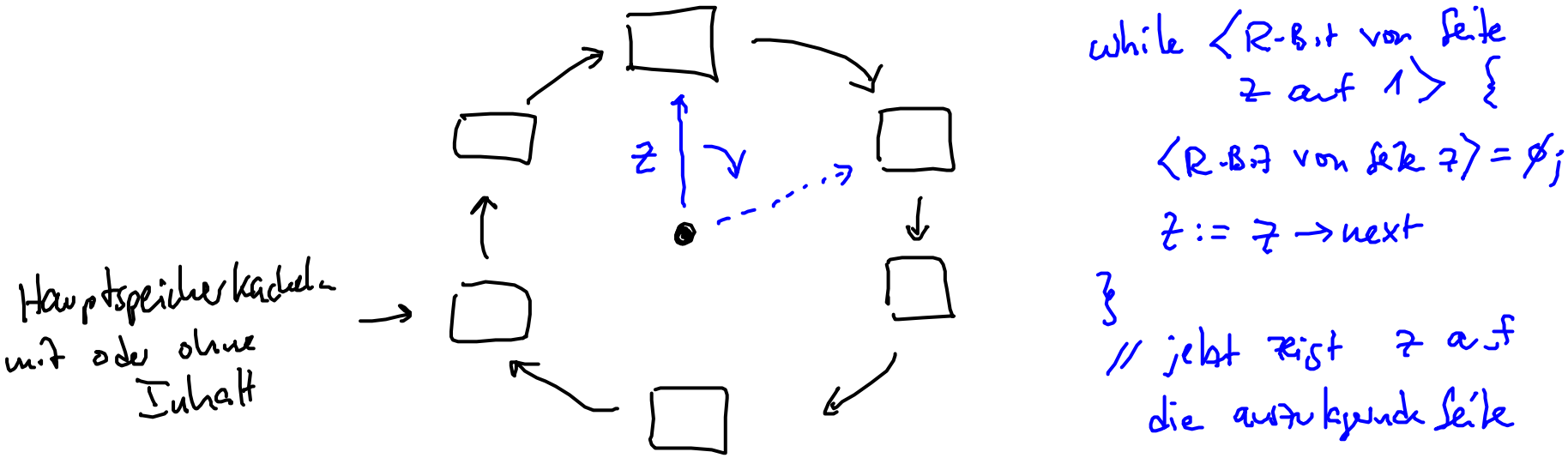
Second Chance

- Verbesserte Variante des FIFO-Verfahrens
  - Verwende FIFO-Liste
  - Reihe Seiten, deren R-Bit gesetzt ist, wieder hinten ein
    - und setze deren R-Bit wieder auf 0
- FIFO-Auslagerungskandidaten bekommen also eine zweite Chance!
- Beispiel: 2 Kacheln, gespeichert wird jeweils das Tupel (Seitennummer, R-Bit)



Clock-Algorithmus

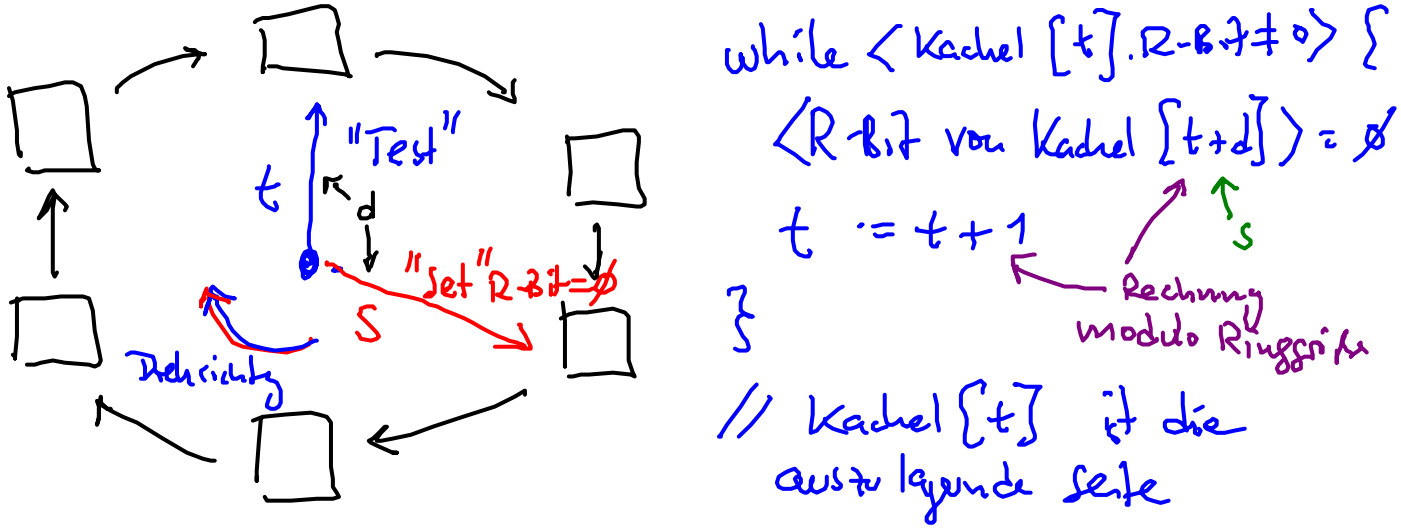
- Implementierungsvariante des 2nd Chance-Algorithmus
  - FIFO-Schlange ist als doppelt verkettete Liste ringförmig organisiert
  - Zeiger zeigt immer auf das aktuell inspizierte Element
  - Schalte Zeiger solange weiter, bis eine Seite mit R-Bit = 0 gefunden wird



  - Suche nach Verdrängungskandidaten kann bei vielen Kacheln lange dauern

Variante des Clock-Algorithmus

- Verwende zwei Zeiger:
  - Bewegen sich synchron in festem Abstand durch den Ring
  - Erster (vorderer) Zeiger setzt R-Bit zurück
  - Zweiter Zeiger sucht Verdrängungskandidaten (prüft R-Bit)



  - Abstand der Zeiger bestimmt, wieviele Schritte maximal durchgeführt werden müssen, um einen Auslagerungskandidaten zu finden